



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

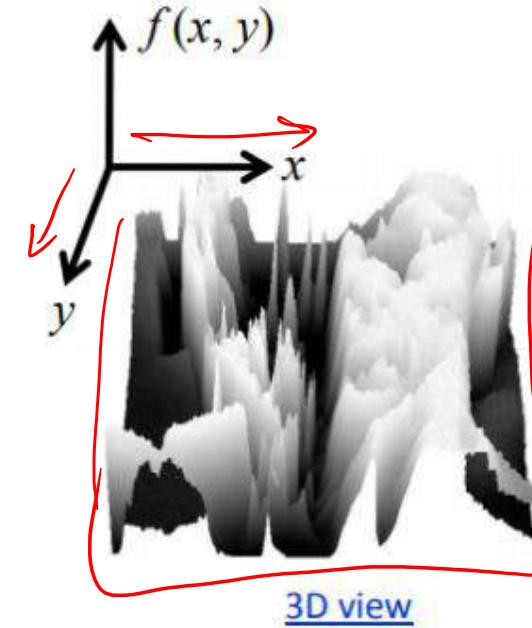
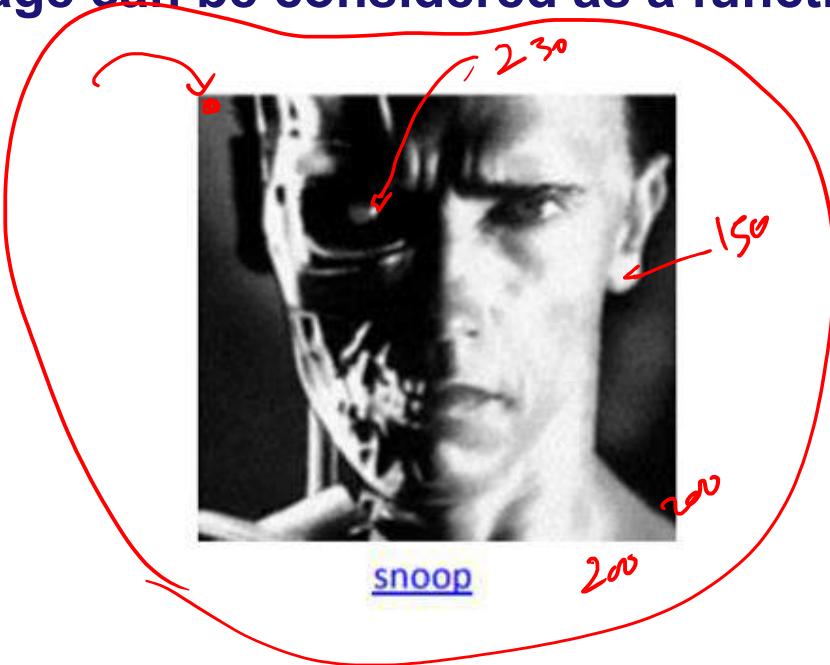
# CNN Prerequisites: Computer Vision

**Dr. Kamlesh Tiwari**

# Image

Grayscale image can be considered as a function  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$

0-255



- Being digital adds quantization and sampling. We may apply operators on this function

~~X~~

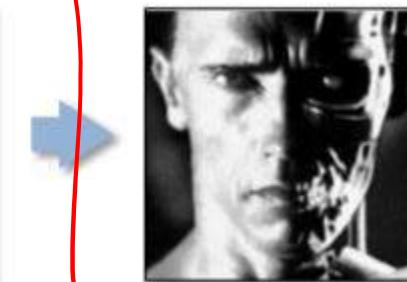
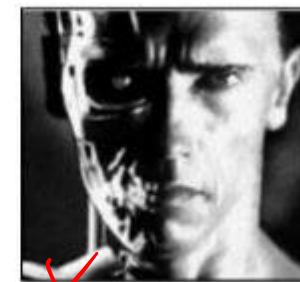


~~f~~



$$g(x, y) = f(x, y) + 20$$

~~g~~



~~g~~

$$g(x, y) = f(-x, y)$$

~~S~~

$$f(\mathbb{F}^{-1}, y)$$

# Computer Vision - history

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
PROJECT MAC

Artificial Intelligence Group  
Vision Memo. No. 100.

July 7, 1966

THE SUMMER VISION PROJECT  
Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

## Some Tasks

- Classification
- Annotation
- Detection
- Segmentation

How the Afghan Girl was Identified by her Iris Patterns 1984-2002



# Classification

airplane

automobile

bird

cat

deer

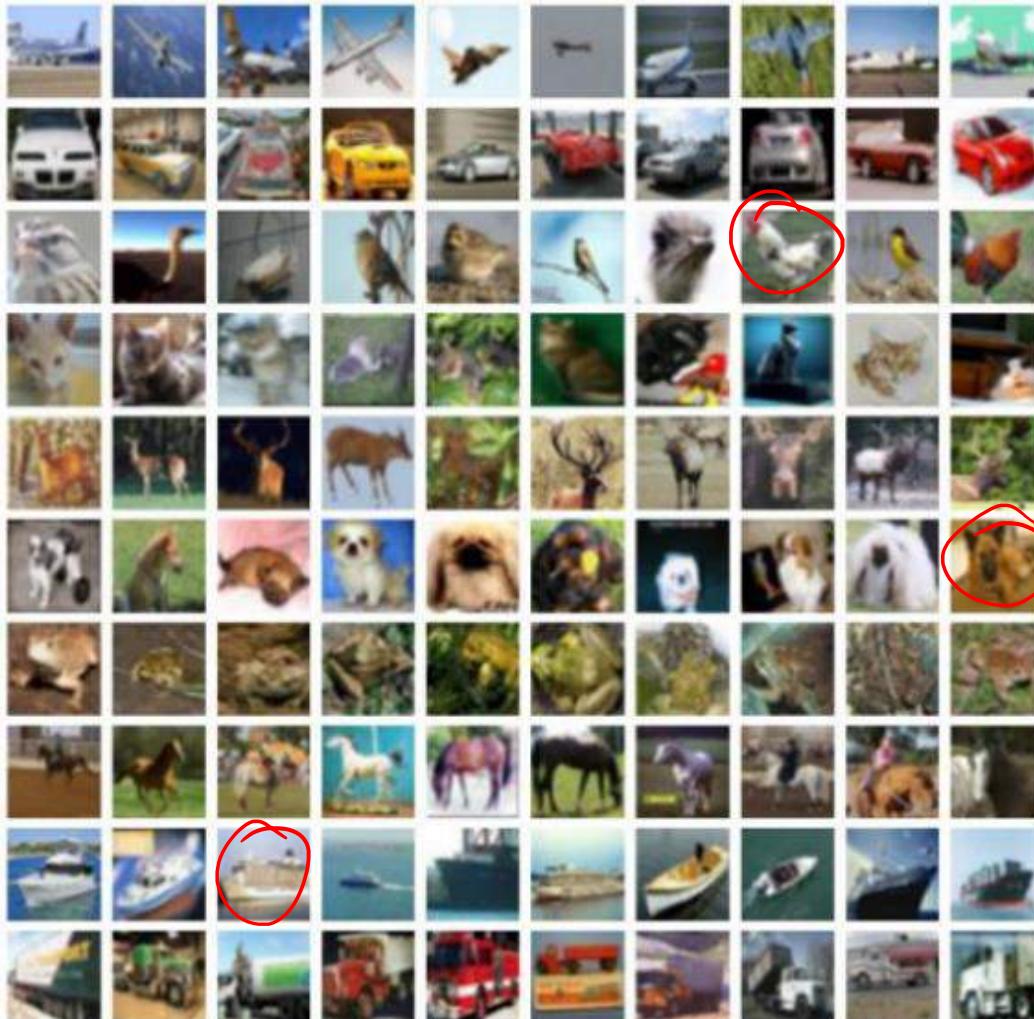
dog

frog

horse

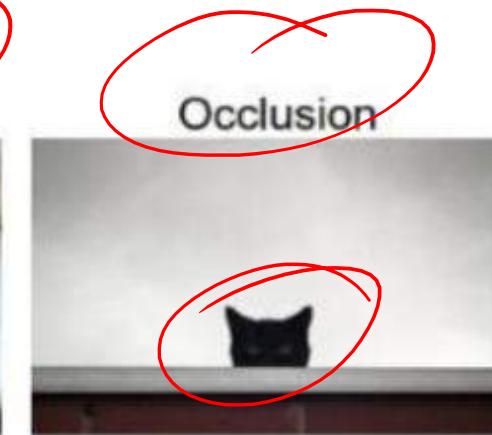
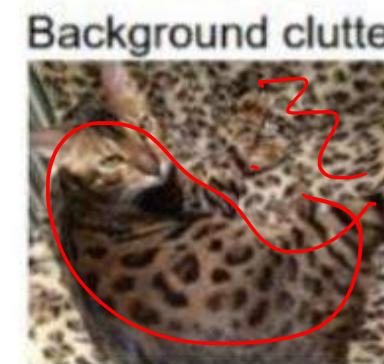
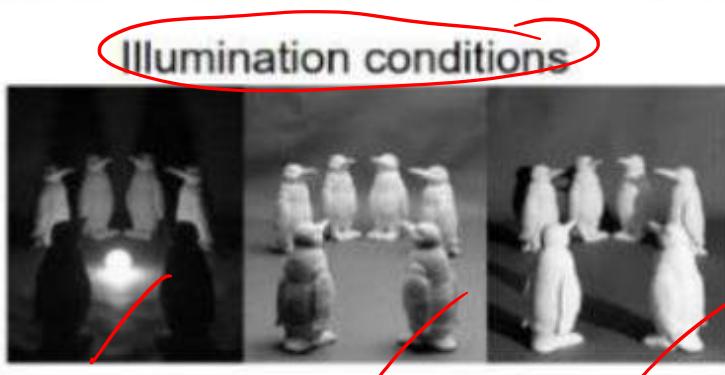
ship

truck



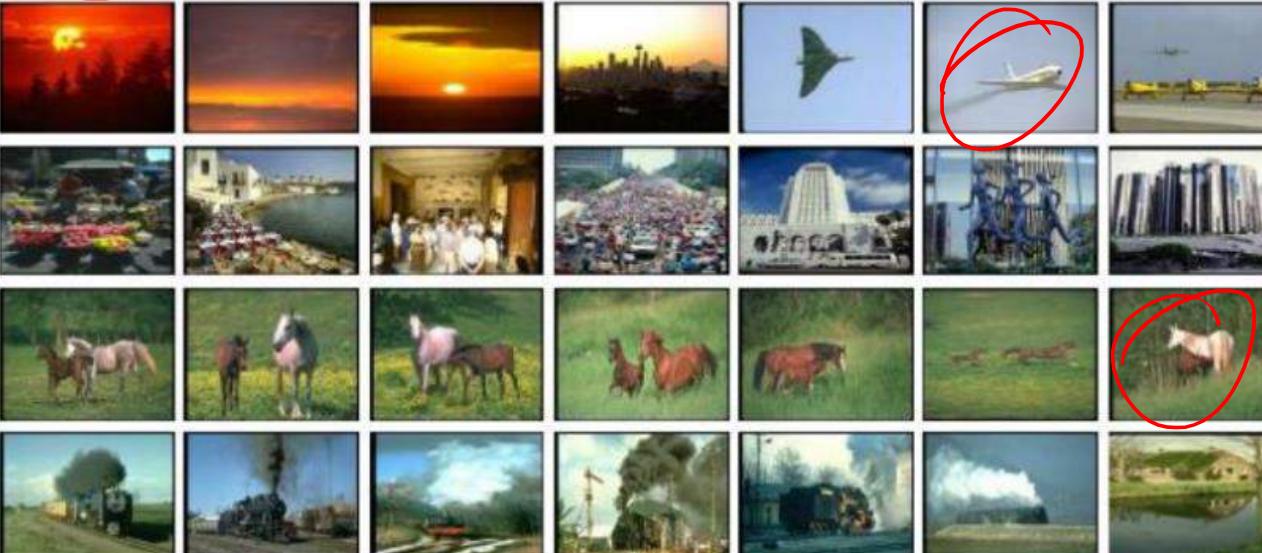
6

# Classification Challenges



# Annotation

					
Predicted keywords	sky, jet, plane, smoke, formation	grass, rocks, sand, valley, canyon	sun, water, sea, waves, birds	water, tree, grass, deer, white-tailed	bear, snow, wood, deer, white-tailed
Human annotation	sky, jet, plane, smoke	rocks, sand, valley, canyon	sun, water, clouds, birds	tree, forest, deer, white-tailed	tree, snow, wood, fox

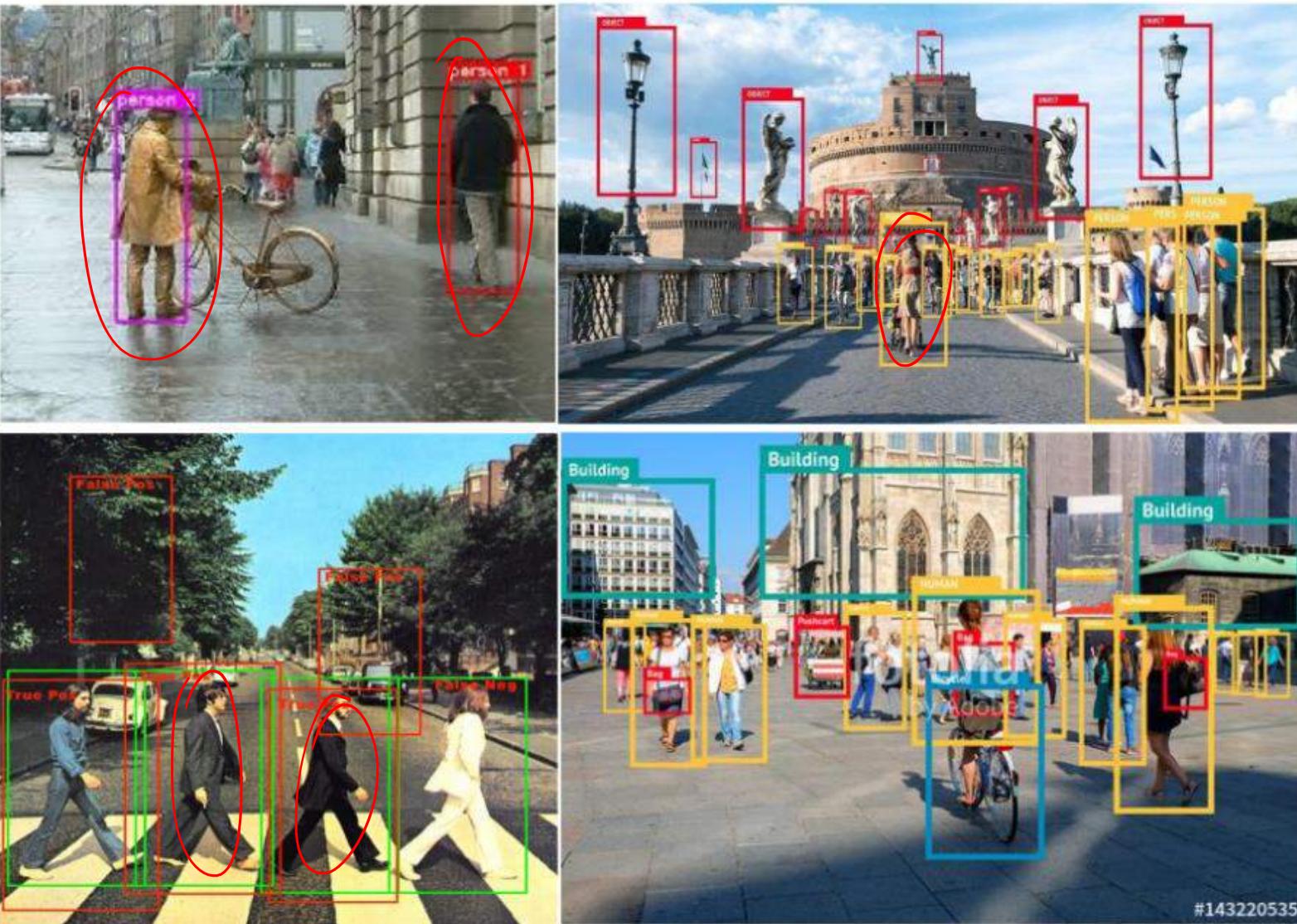
The grid displays four rows of images. Row 1 shows various sunsets and skies. Row 2 shows street scenes with people and vehicles. Row 3 shows horses (mares) in different settings. Row 4 shows trains and steam locomotives. Red circles highlight specific predictions in the first three rows that differ from the human annotations.

Prediction for queries: Sky (Row1), Street (Row2), Mare (Row3) and Train (Row4)

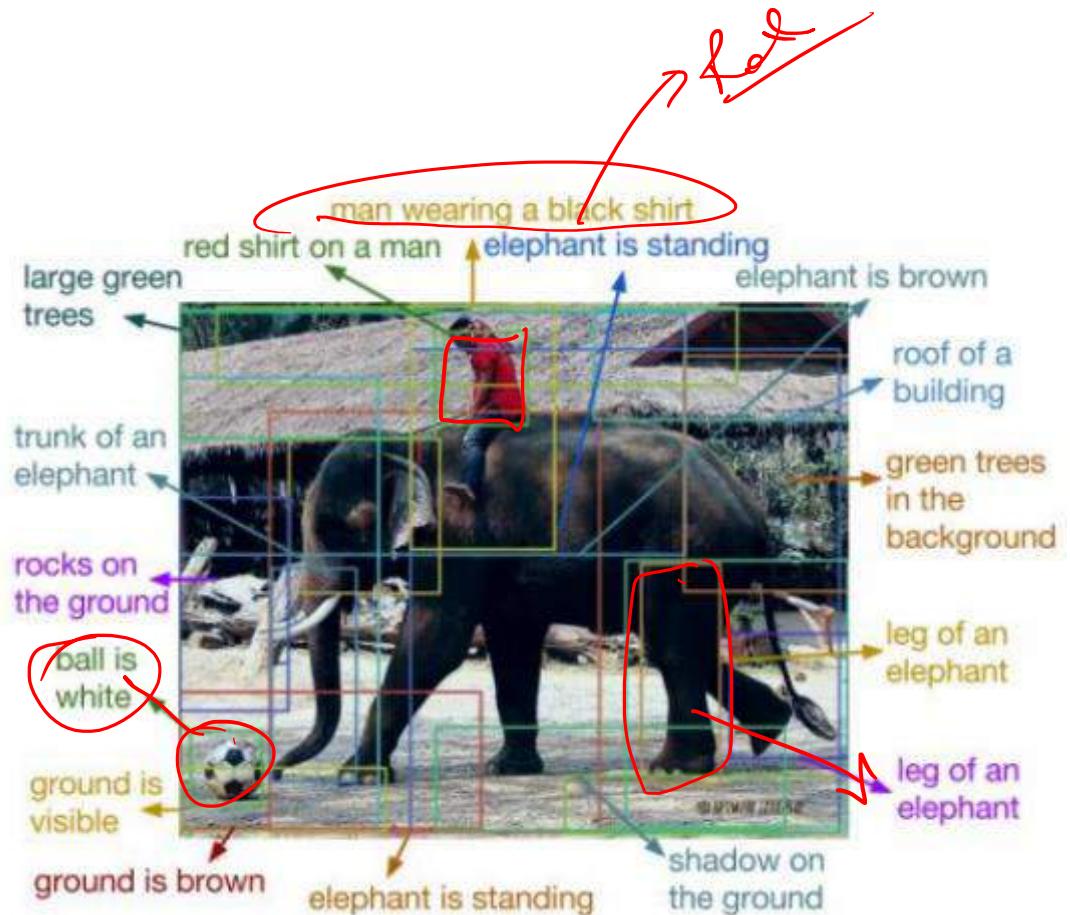
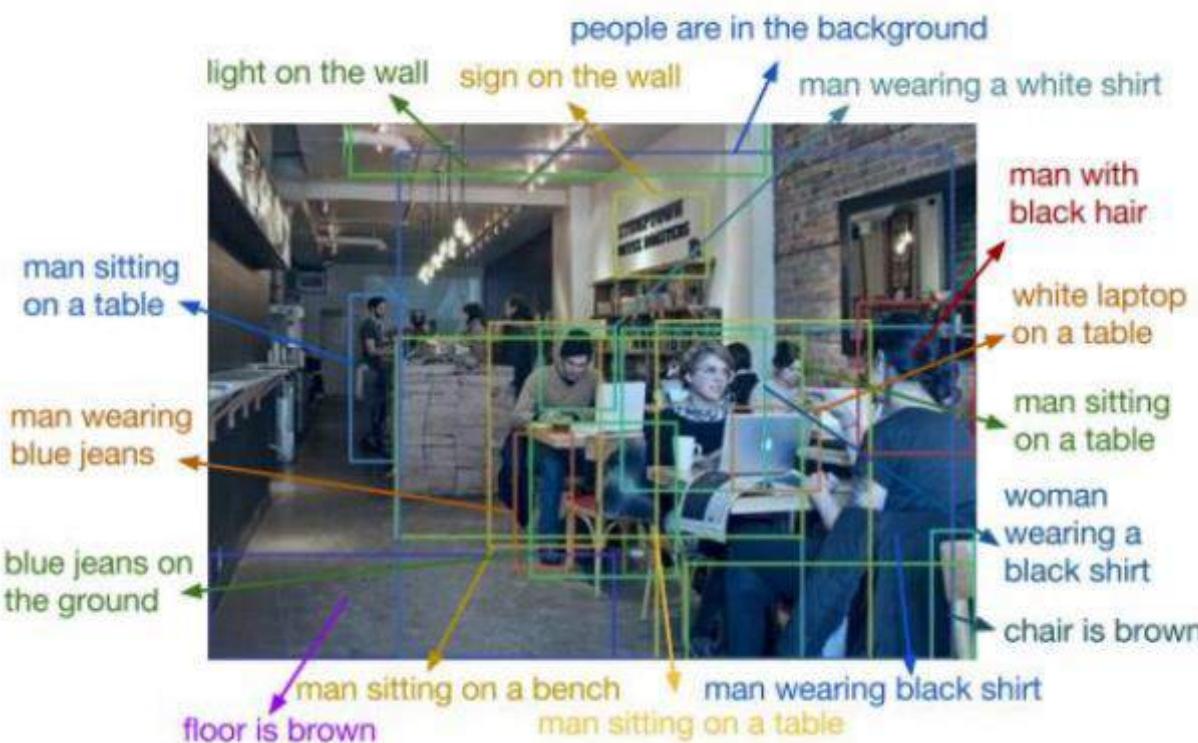
Ref: A. Makadia, V. Pavlovic, and S. Kumar.

A New Baseline for Image Annotation. In Proceedings of ECCV 08.

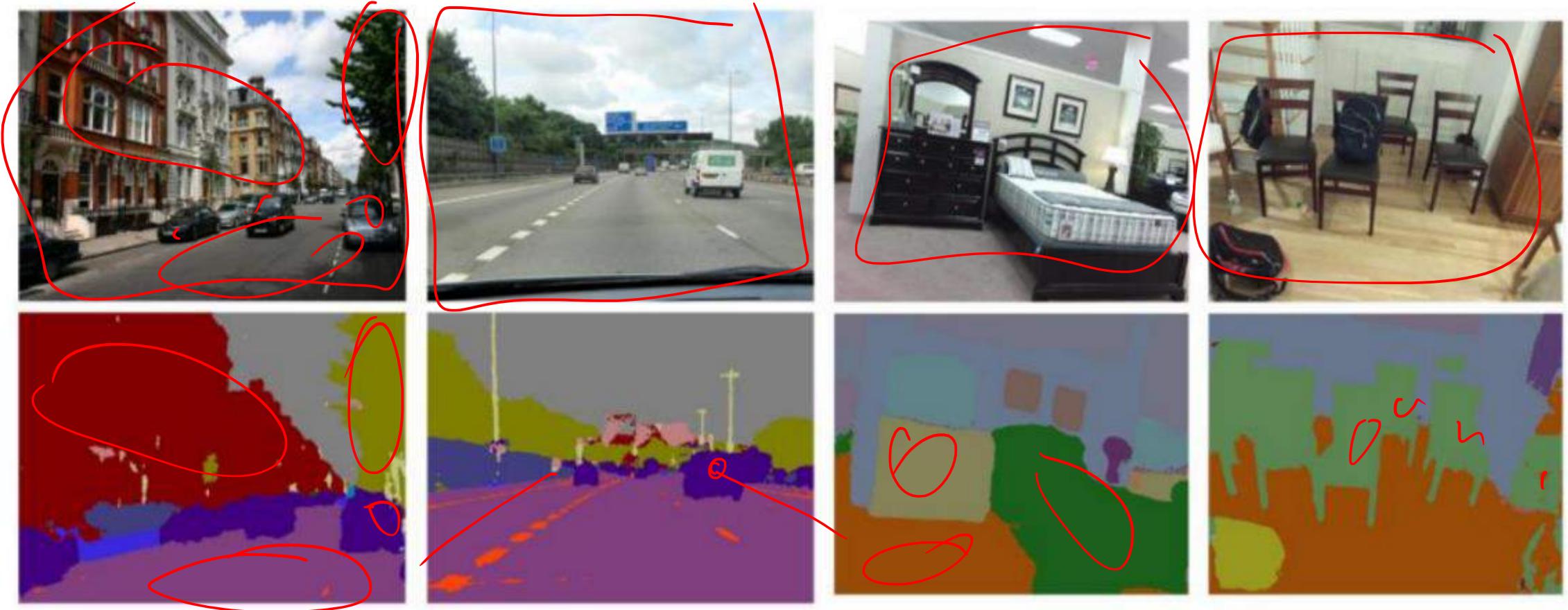
# Detection



# Description

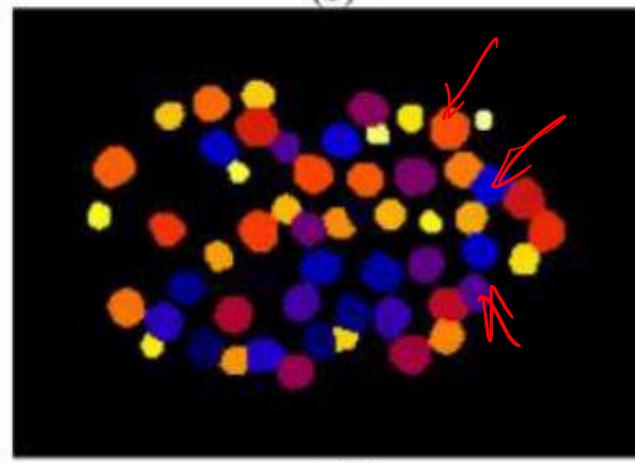
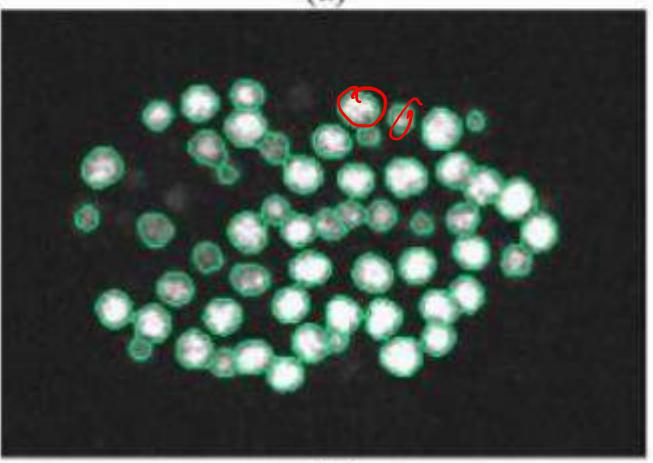
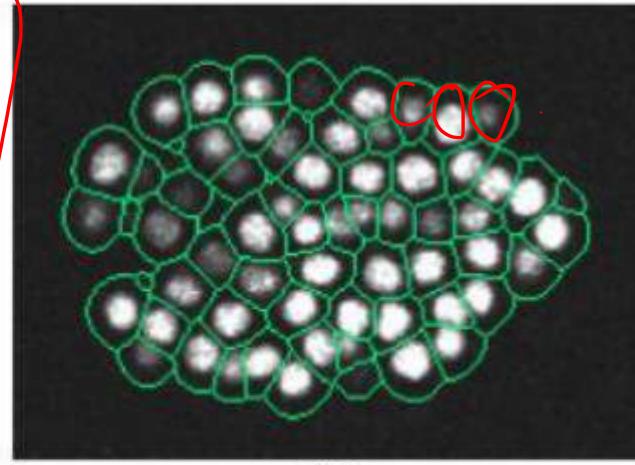
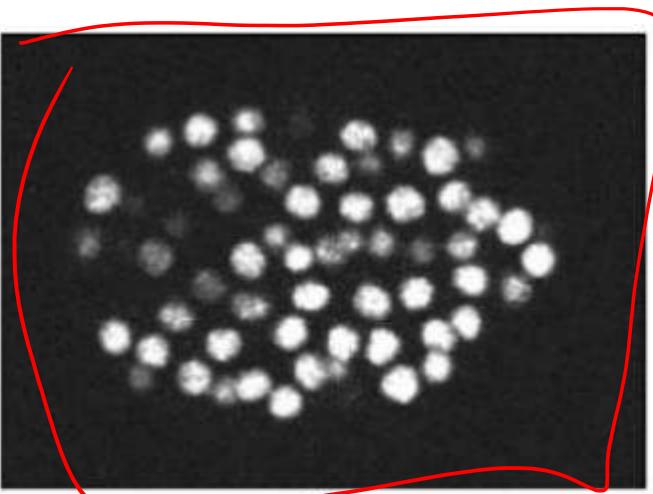


# Segmentation



Vijay Badrinarayanan, Alex Kendall and Roberto Cipolla "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." PAMI, 2017.

# Segmentation





# Thank You!

In our next session: Image Operations



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CNN Prerequisites: Computer Vision

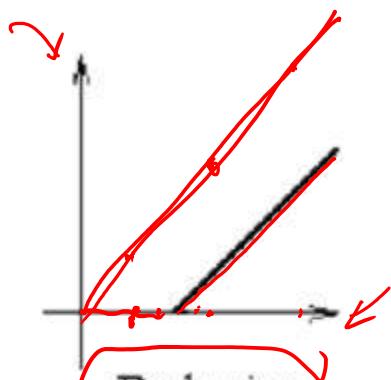
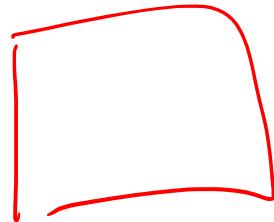
**Dr. Kamlesh Tiwari**

# Point Operations on Images

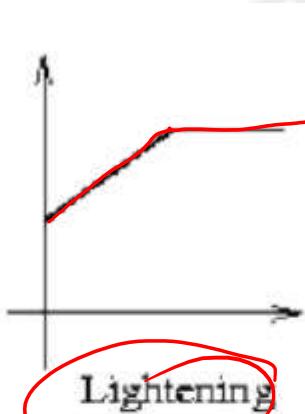
Point operation is defined as

$$s = M(r)$$

where r is source pixel intensity and s is destination pixel intensity



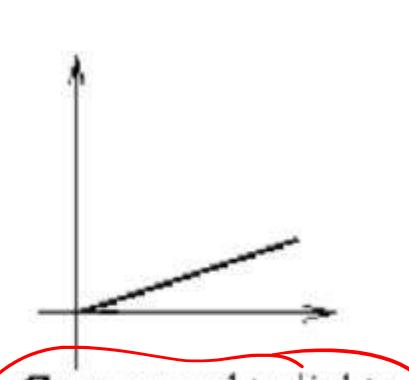
Darkening



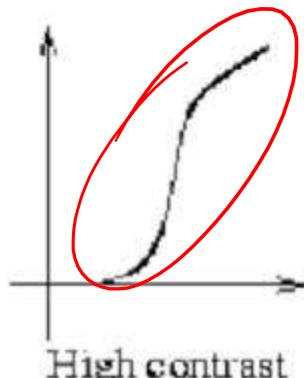
Lightening



Compressed to darks



Compressed to lights



High contrast



Low contrast



Emphasize shadows



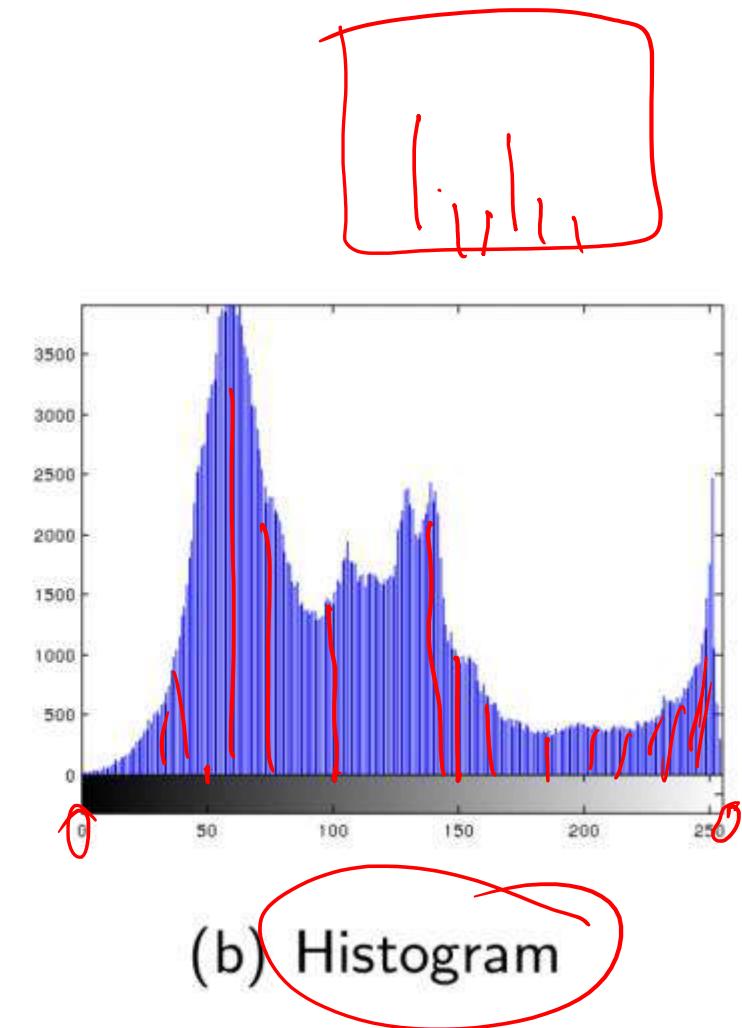
Emphasize lights

# Histogram

- Discrete probability distribution of the image intensity values



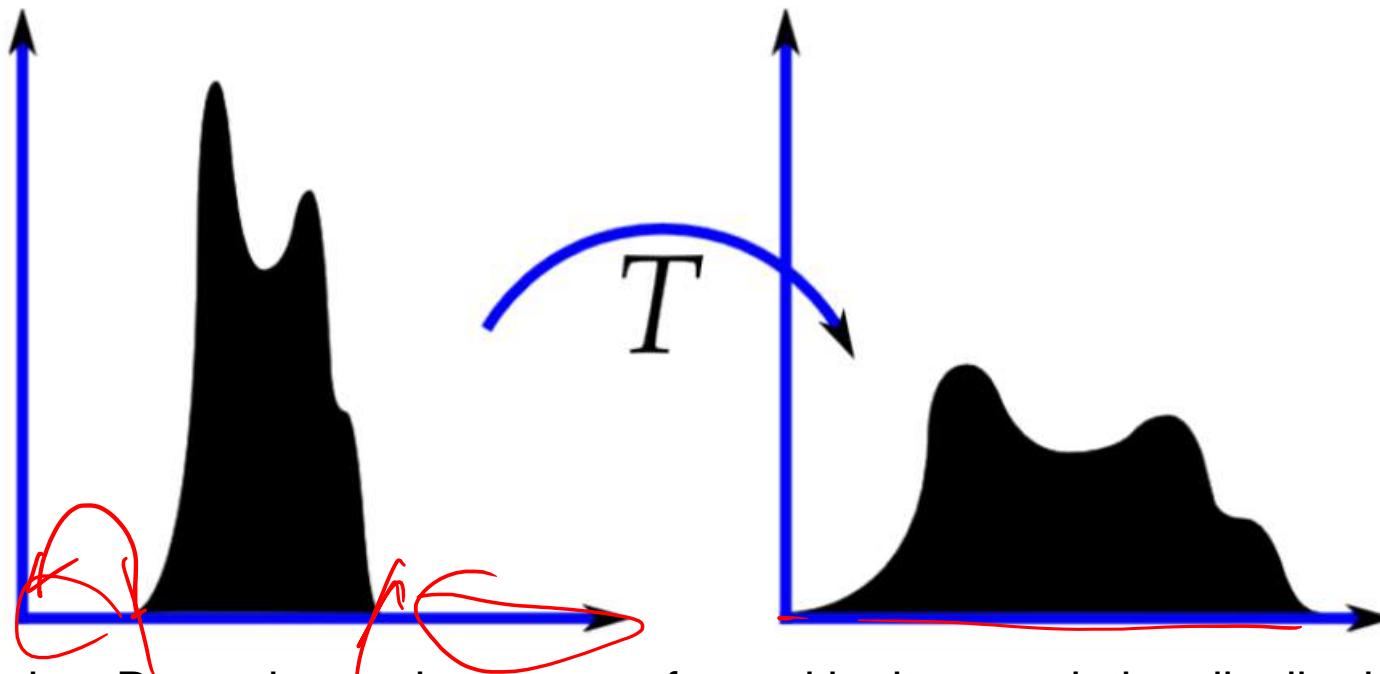
(a) Image



(b) Histogram

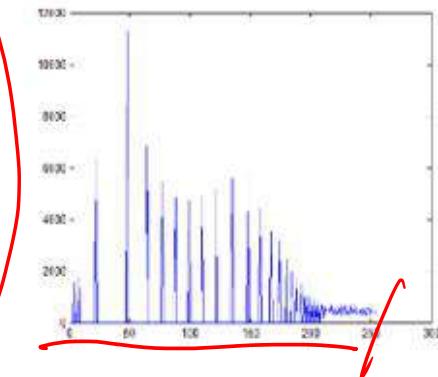
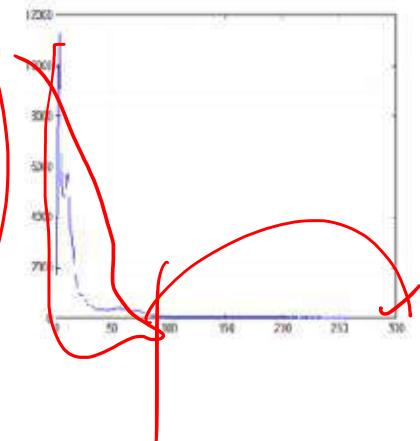
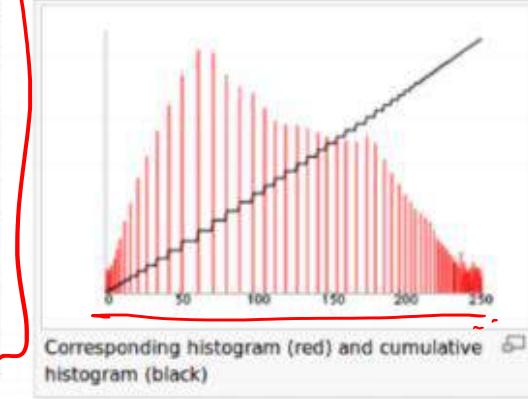
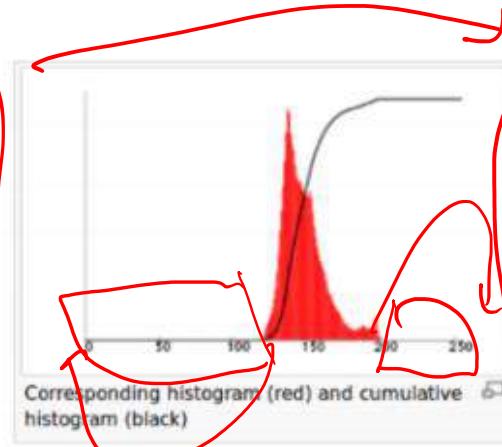
# Histogram Equalization

One method to enhance images is to equalize the histogram



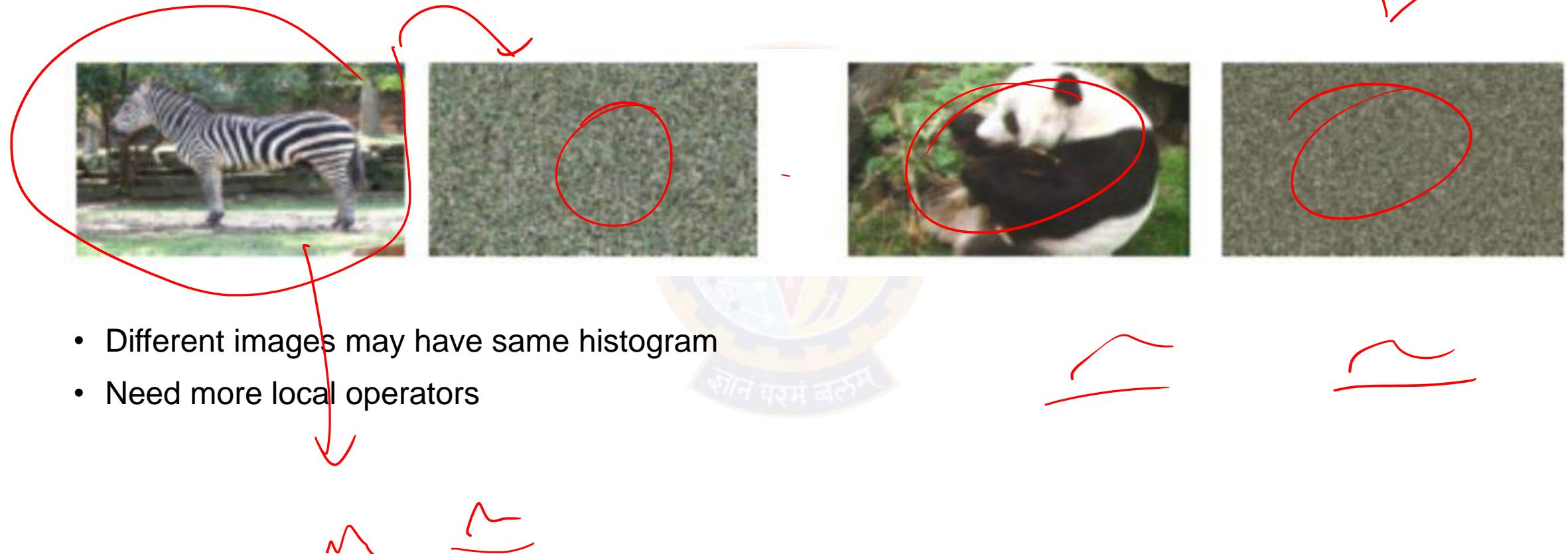
Exercise: Prove that an image transformed by its cumulative distribution function results in an image with uniform histogram. More generally, histogram can be modified by histogram specification

# Histogram Equalization Examples



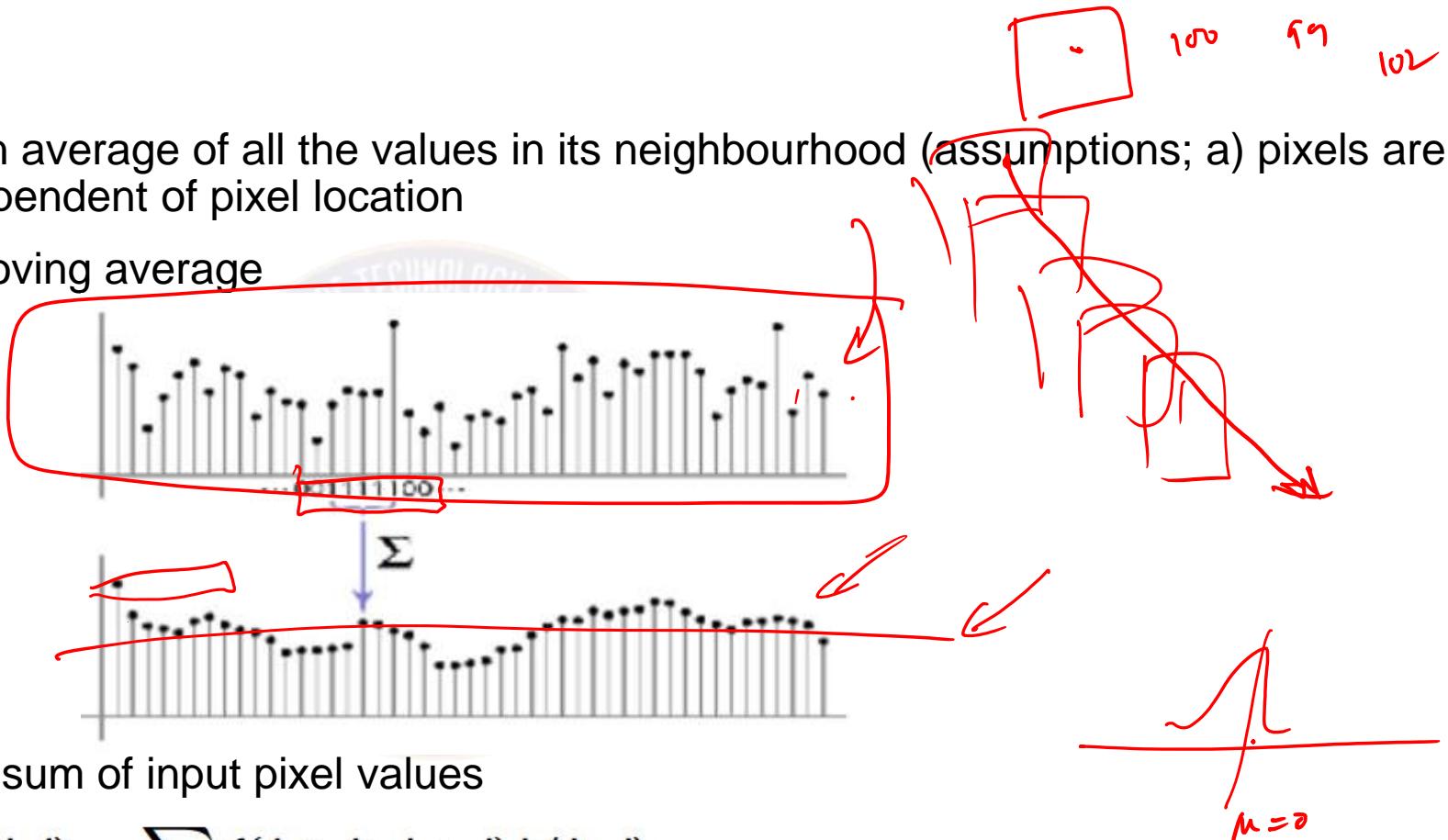
# Shuffling the pixels

- What happens if we shuffle all the pixels in an image randomly?



# First attempt at a solution

- Replace each pixel with an average of all the values in its neighbourhood (assumptions; a) pixels are like neighbour 2) noise is independent of pixel location
- Can add weights to our moving average



- Output pixel is a weighted sum of input pixel values

$$g(i, j) = \sum_{k,l} f(i + k, j + l) h(k, l)$$

- Entries in kernel  $h(k, l)$  are called the filter coefficients. Operator is termed correlation operator.  $g = f \otimes h$

# Convolution

$$g = f \otimes h.$$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

0	10	20	30	30	30	20	10
0	20	40	60	60	60	40	20
0	30	60	90	90	90	60	30
0	30	50	80	80	90	60	30
0	30	50	80	80	90	60	30
0	20	30	50	50	60	40	20
10	20	30	30	30	30	20	10
10	10	10	0	0	0	0	0

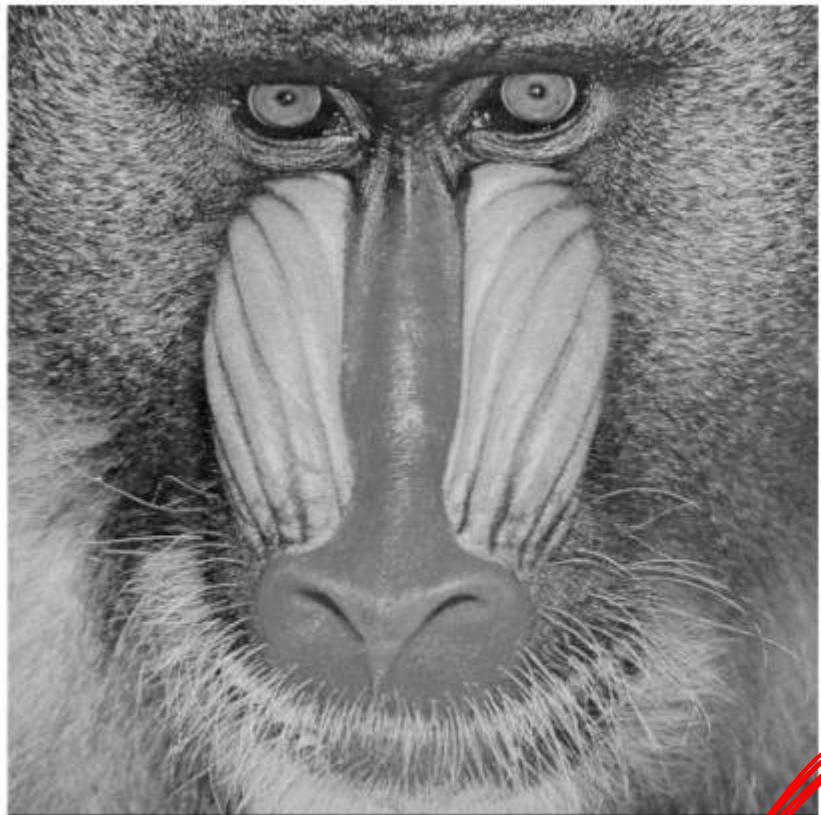


- Convolution is a simple variant of correlation termed as  $g = f * h$

$$g(i, j) = \sum_{k,l} f(i - k, j - l)h(k, l)$$

$$9+15+12 \\ +6+10+10+3+2+15 \rightarrow 72$$

# Box Filtering



(c)



(d)

25x25  
is 5x5.

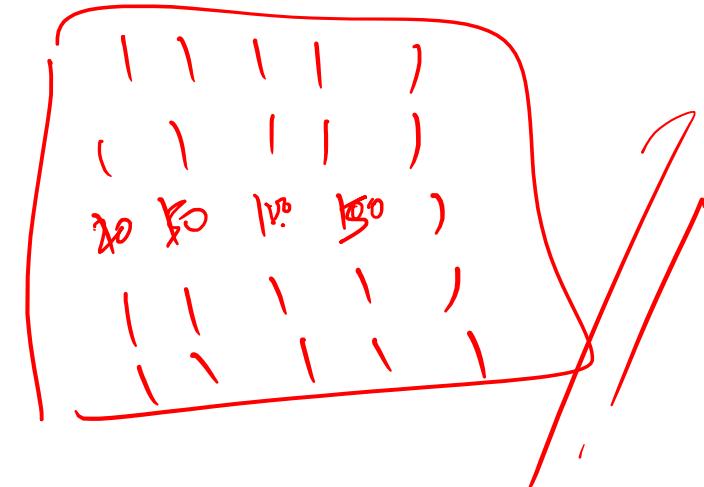
# Gaussian Filtering

Used when we want to have central pixels with more influence.

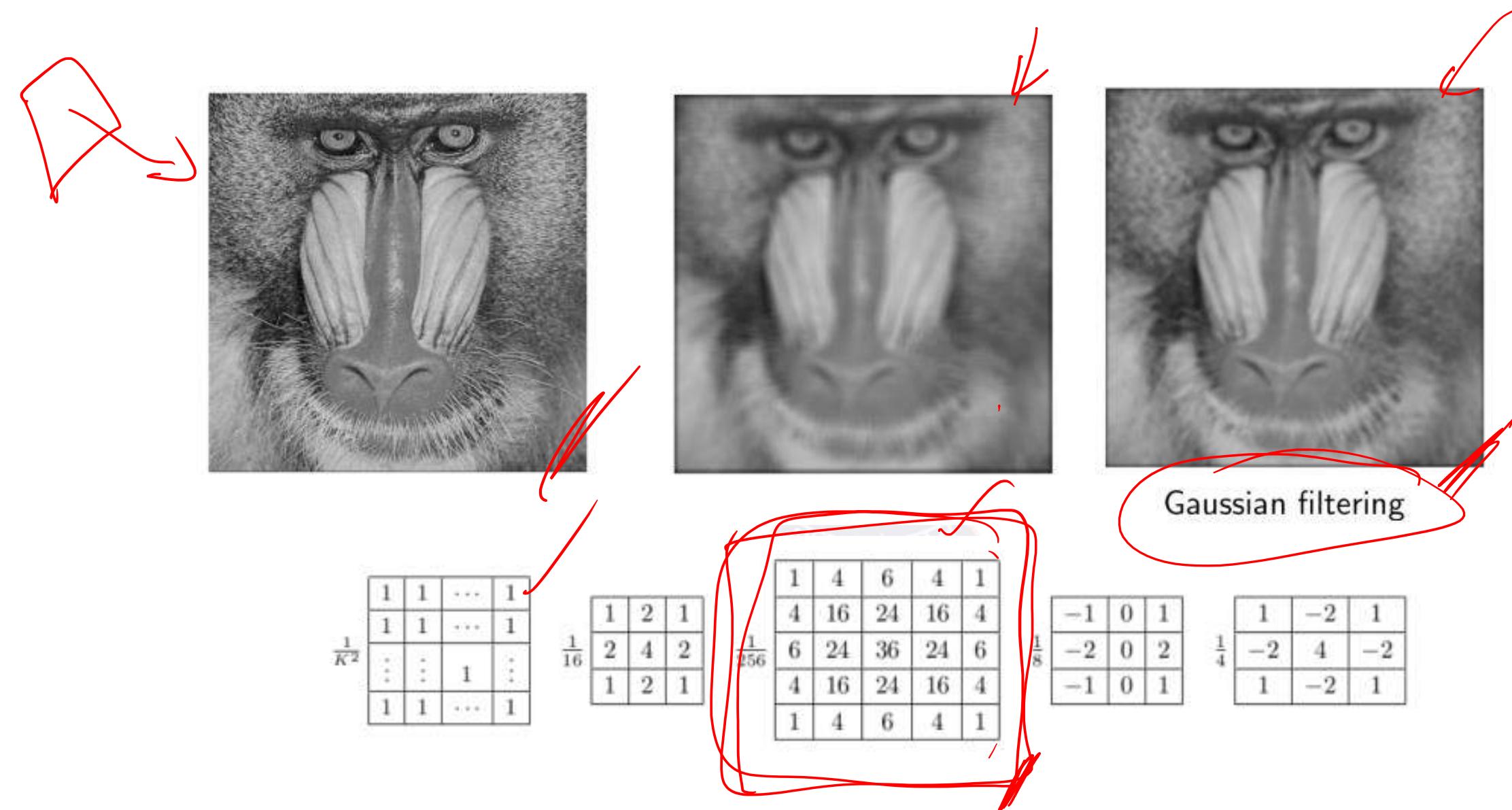
- Gaussian Kernel is defined as

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

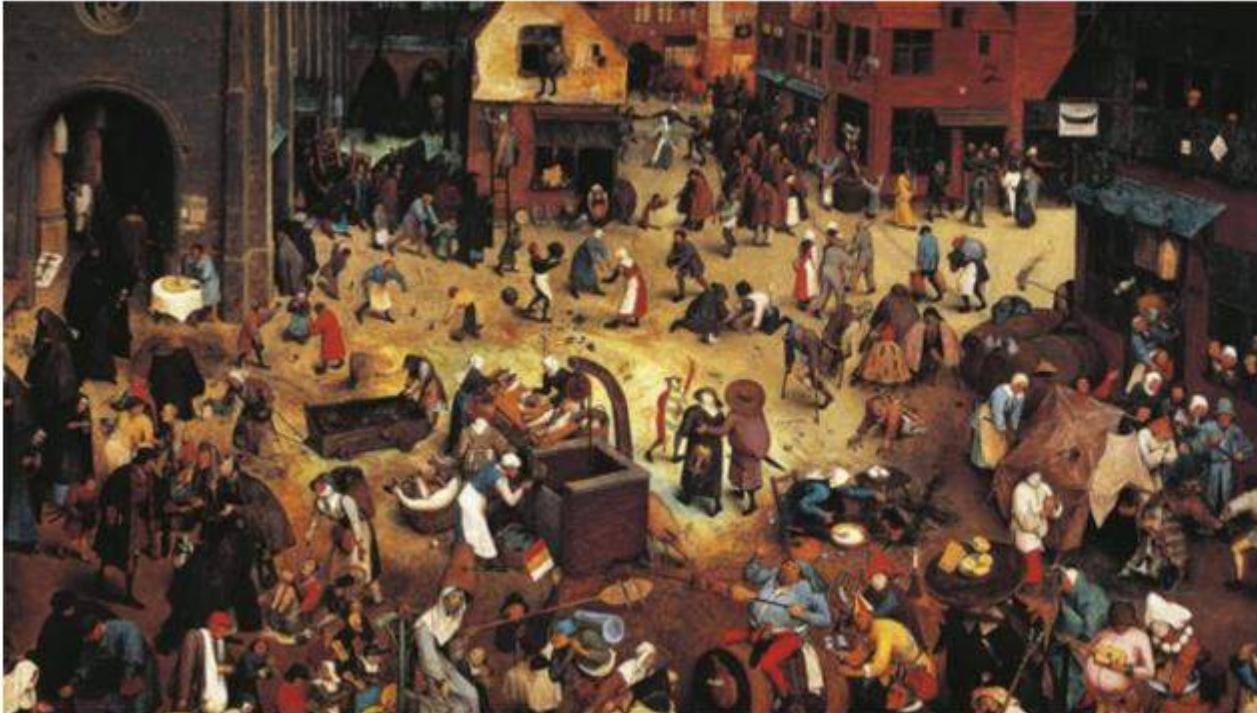
- It is a low pass filter



# Gaussian Filtering

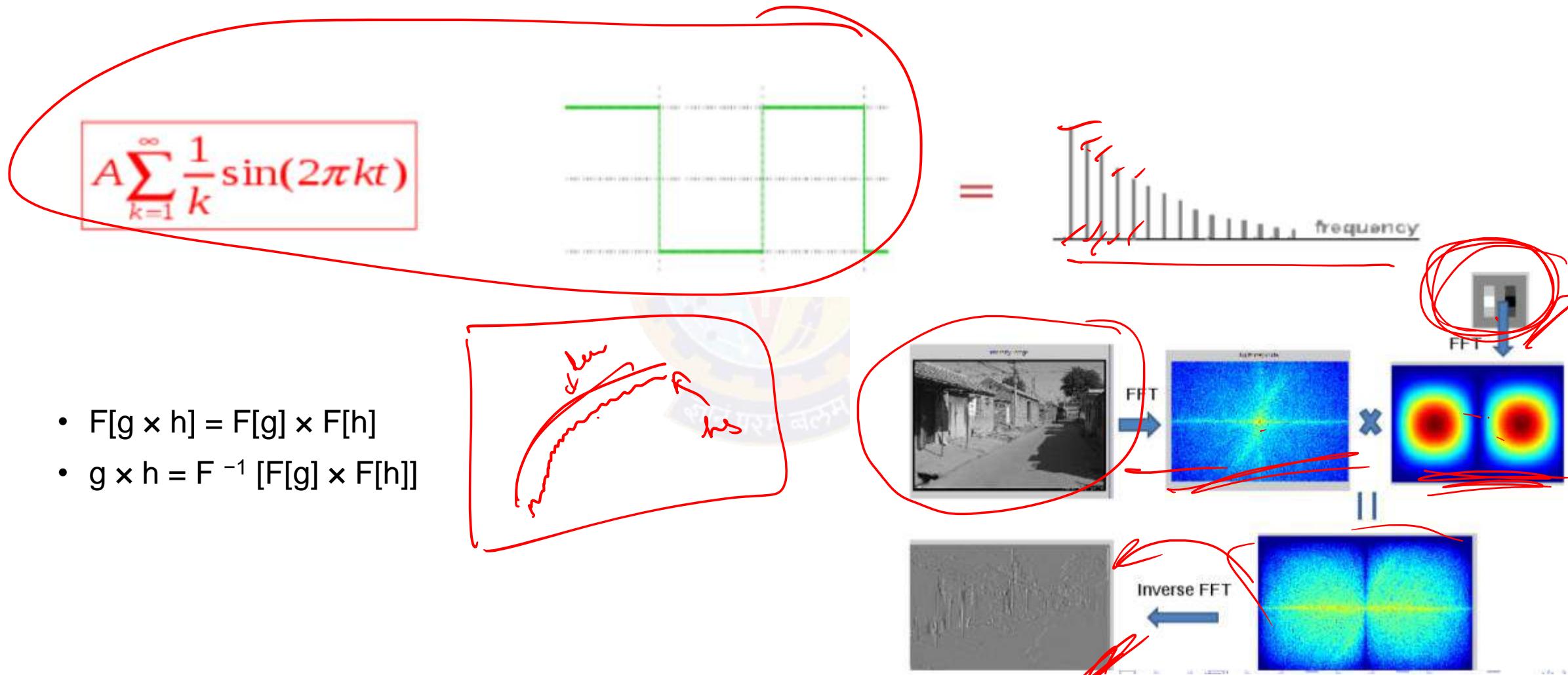


# Convolution



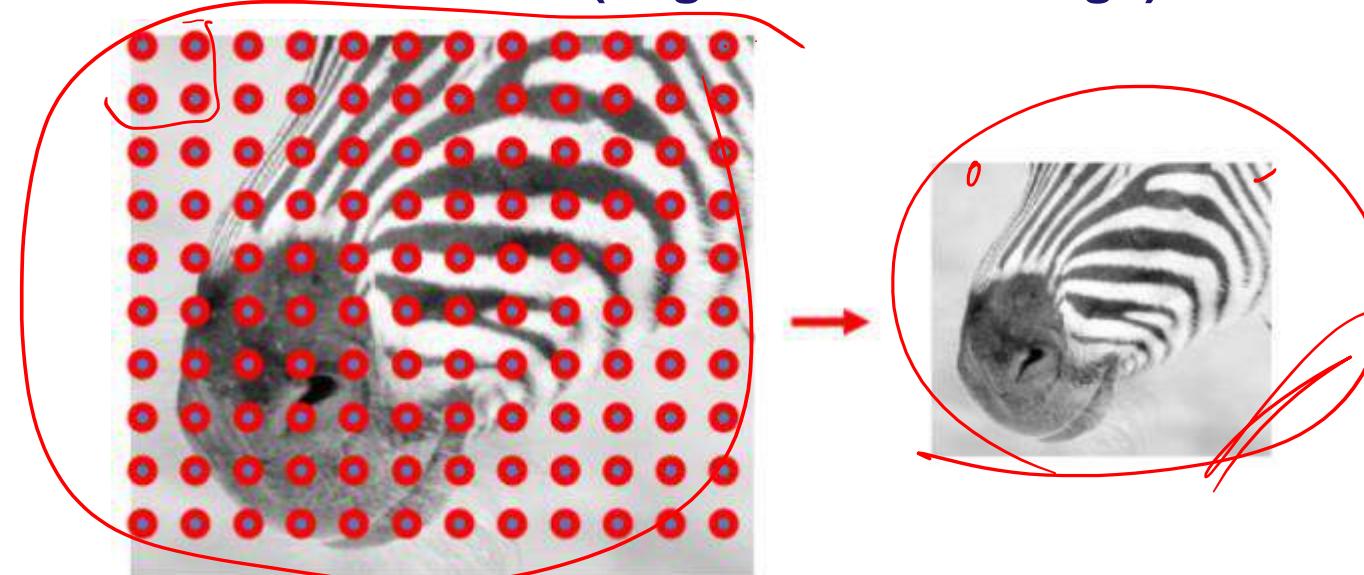
# Fourier Transform

Any signal (function) can be written as sum of various sin and cos waves (terms)

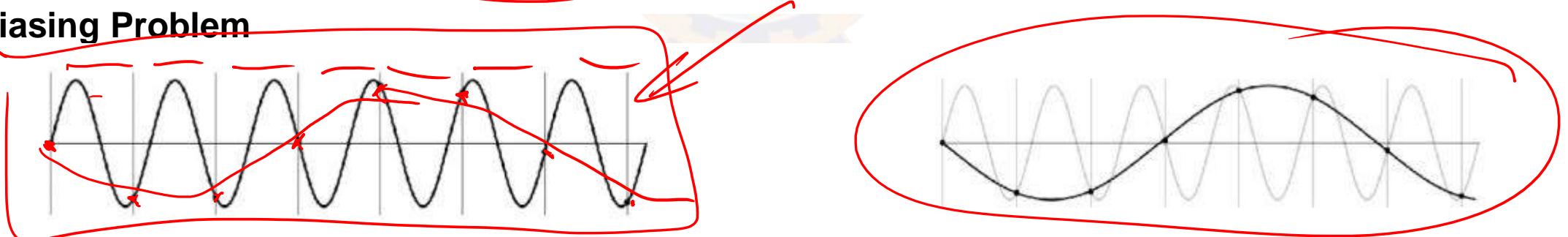


# Sampling

Throw away every other row and column (to get half size image)



Aliasing Problem

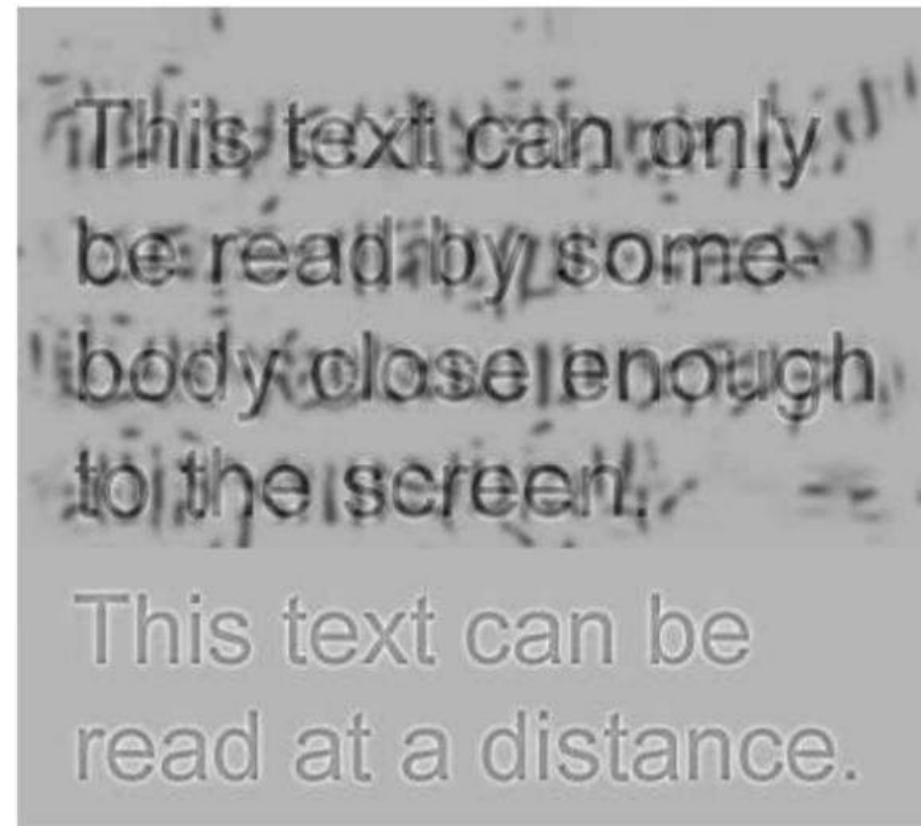


Sub sampling may be dangerous: Funny striped shirt. Wagon wheels rolling in the wrong way in movies.

# Hybrid Images



# Example Hybrid Image



The hybrid font becomes invisible at few meters. The bottom text remains easy to read at relatively long distances.



# Thank You!

In our next session: Filtering



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CNN Prerequisites: Computer Vision

**Dr. Kamlesh Tiwari**

# Linear filtering



Original

$$\text{Original} * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \text{Identical image}$$



Identical image

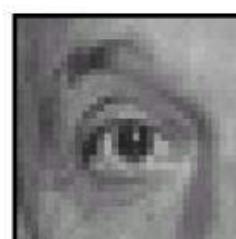


Original

$$\text{Original} * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \text{Shifted left By 1 pixel}$$



Shifted left By 1 pixel



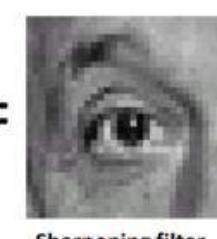
Original

$$\text{Original} * \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \text{Blur (with a mean filter)}$$

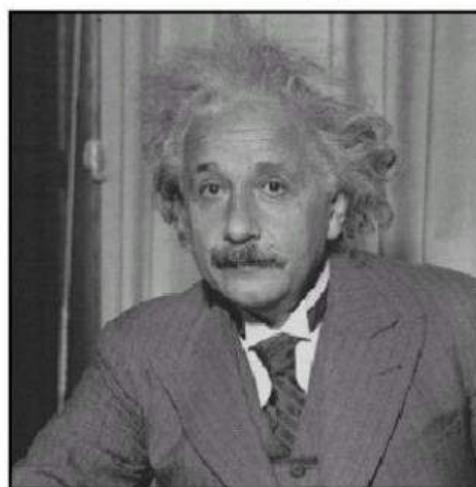


Blur (with a mean filter)

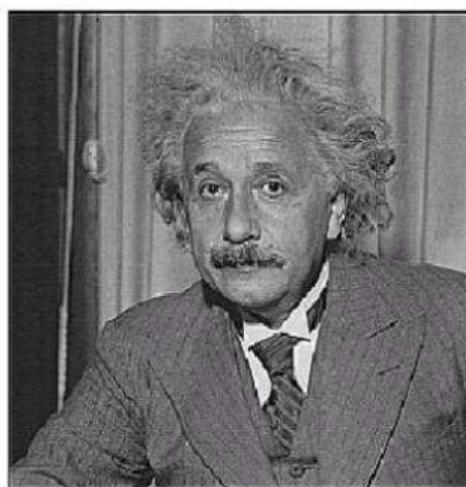
$$\text{Original} * \left( \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) = \text{Sharpening filter (accentuates edges)}$$



Sharpening filter  
(accentuates edges)



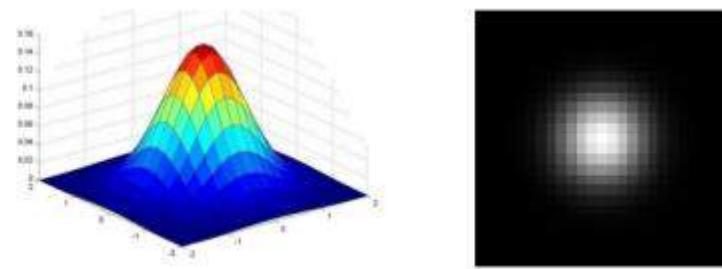
before



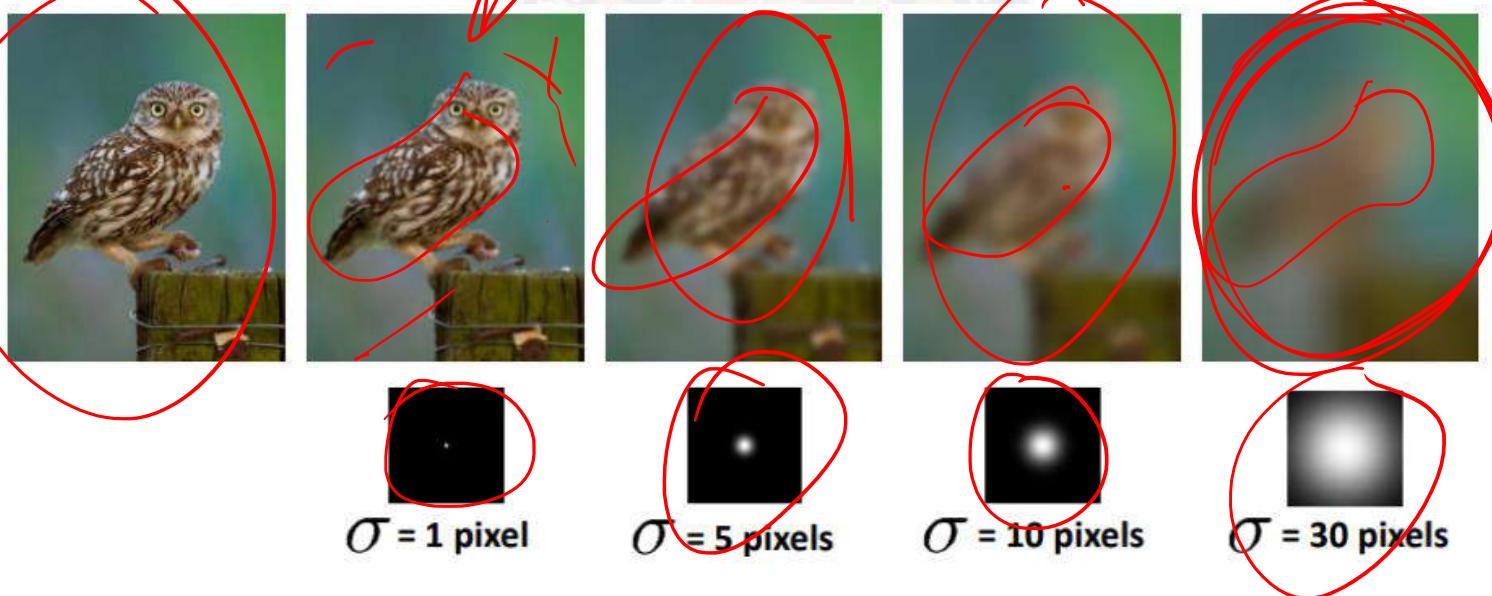
after

# Gaussian filter

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



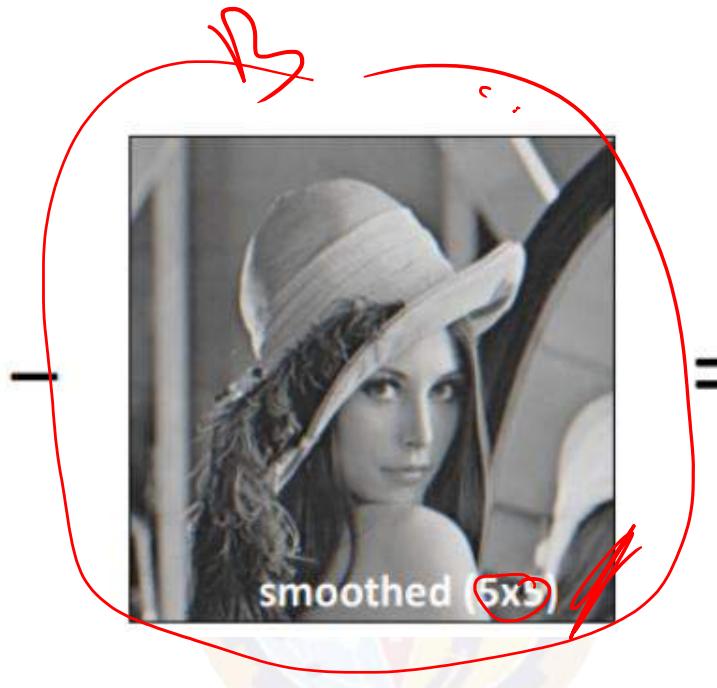
Act as low-pass filter as it removes high-frequency components. Convolution with self is another Gaussian.



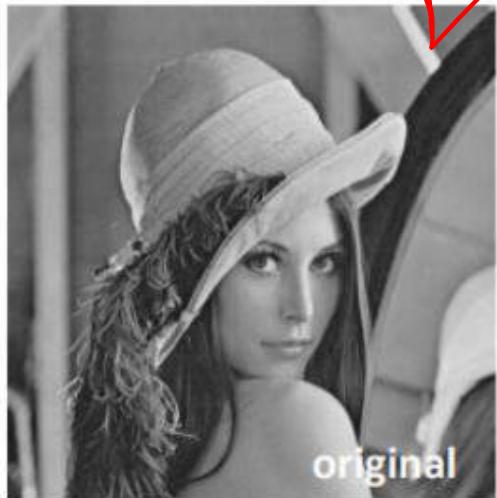
Convolving two times with Gaussian kernel of  $\sigma$  width is equivalent to Convoluting once with Gaussian kernel of  $\sqrt{2}\sigma$  width

# Sharpening

What does blurring take away?



Let's add it back:



# Edge detection

Common approximation of derivative of Gaussian

$$\frac{1}{8} \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

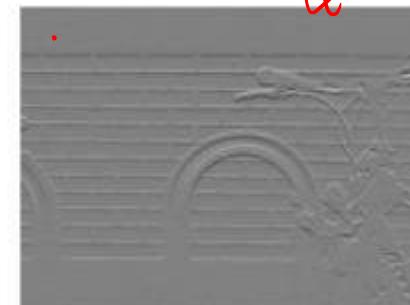
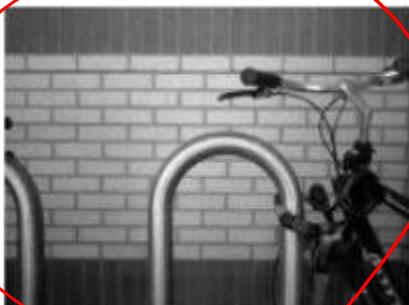
$s_x$

$$\frac{1}{8} \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

$s_y$

Edge magnitude is  $\sqrt{s_x^2 + s_y^2}$  and the direction is  $\tan^{-1}(s_y/s_x)$

Edge is detected by using threshold





# Thank You!

In our next session:



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) 1-D

**Dr. Kamlesh Tiwari**

# Motivation to Convolution

Suppose you have a noisy sensor to have a measurement

How could you estimate the actual reading

1. Average of some readings
2. in local neighbourhood
3. What about weighted average?



$$s_t = \sum_{i=0}^{\infty} x_{t-i} w_i$$

	$w_{-6}$	$w_{-5}$	$w_{-4}$	$w_{-3}$	$w_{-2}$	$w_{-1}$	$w_0$
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90
S							

1.80					
------	--	--	--	--	--

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

Here things are in one dimension only.



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) 2-D

**Dr. Kamlesh Tiwari**

# Motivation to Convolution

Suppose you have a noisy sensor to have a measurement

How could you estimate the actual reading

1. Average of some readings
2. in local neighbourhood
3. What about weighted average?



$$s_t = \sum_{i=0}^{\infty} x_{t-i} w_i$$

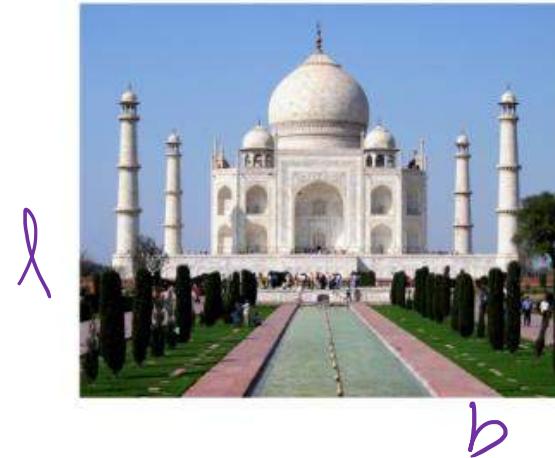
	$w_{-6}$	$w_{-5}$	$w_{-4}$	$w_{-3}$	$w_{-2}$	$w_{-1}$	$w_0$
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90
S							

1.80					
------	--	--	--	--	--

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

Here things are in one dimension only.

# Image is a 2D signal

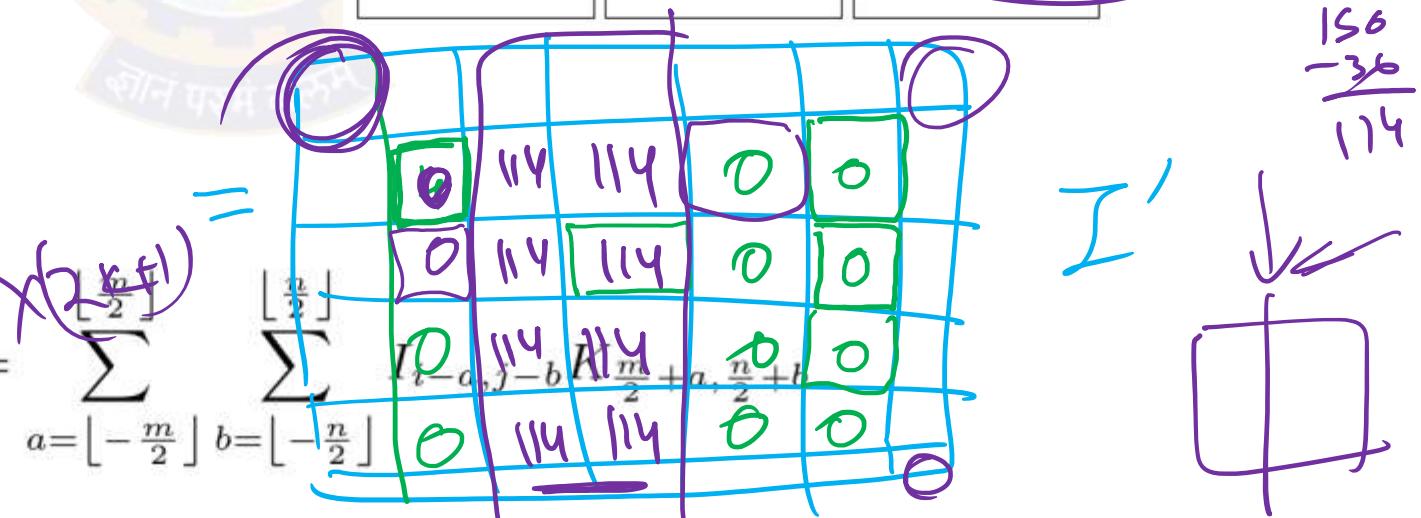
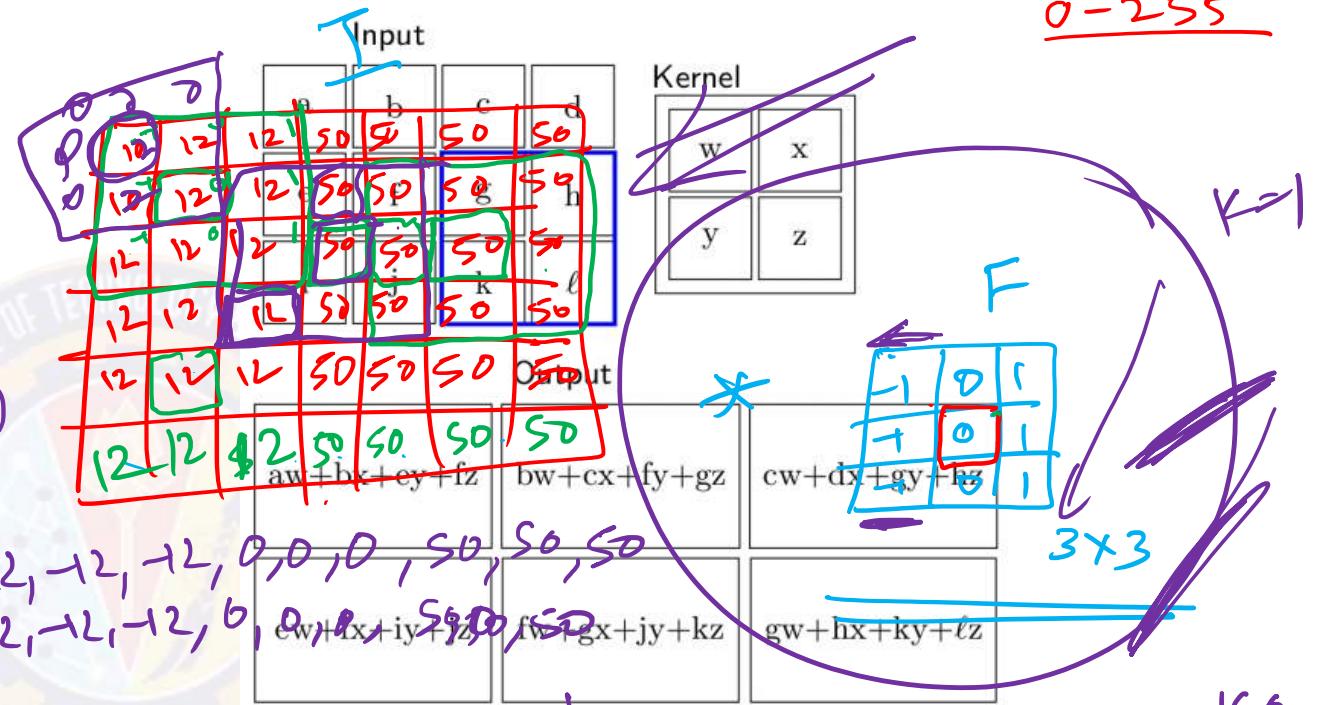


$$\rightarrow (l-2k) \times (b-2k)$$

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a, b}$$

General formula

$$S_{ij} = (I * K)_{ij} = \sum_{a=-\frac{m}{2}}^{\frac{n}{2}} \sum_{b=-\frac{n}{2}}^{\frac{n}{2}} I_{i+a, j+b} K_{a, b}$$





# Thank You!

In our next session:



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) 2-D

**Dr. Kamlesh Tiwari**

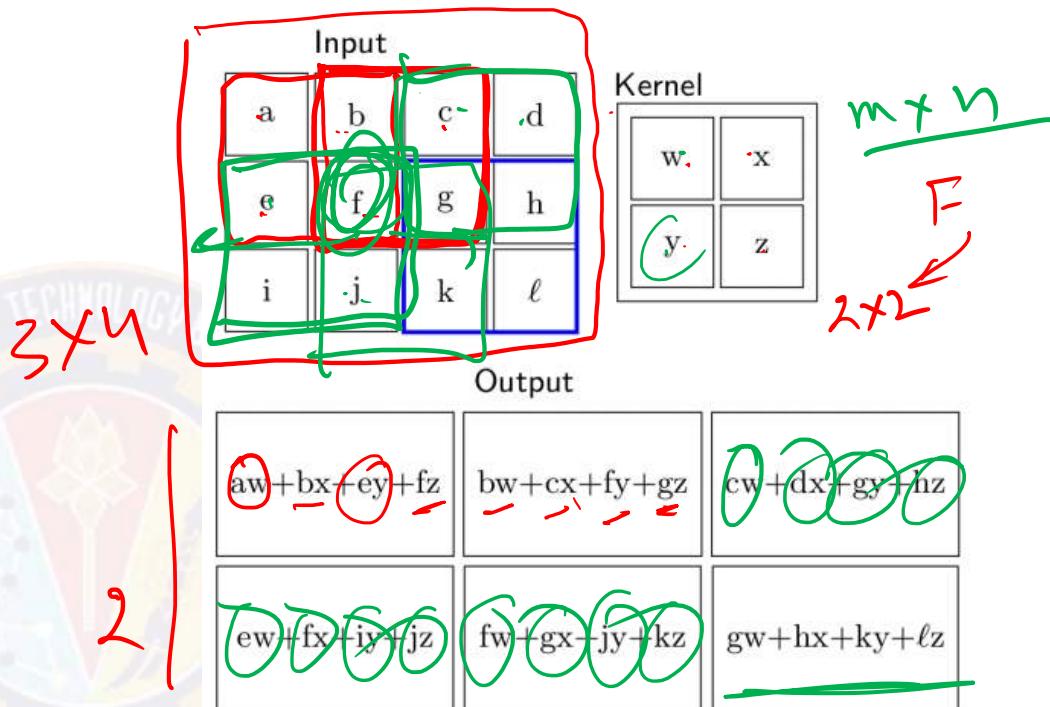
# Image is a 2D signal



$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a, b}$$

General formula

$$S_{ij} = (I * K)_{ij} = \sum_{a=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a, j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$



$$F = 2K+1$$

$$\lfloor \frac{F}{2} \rfloor = K$$

# Image is a 2D signal



$$* \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



blurs the image

$$\begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$* \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} =$$



sharpens the image

$$\begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

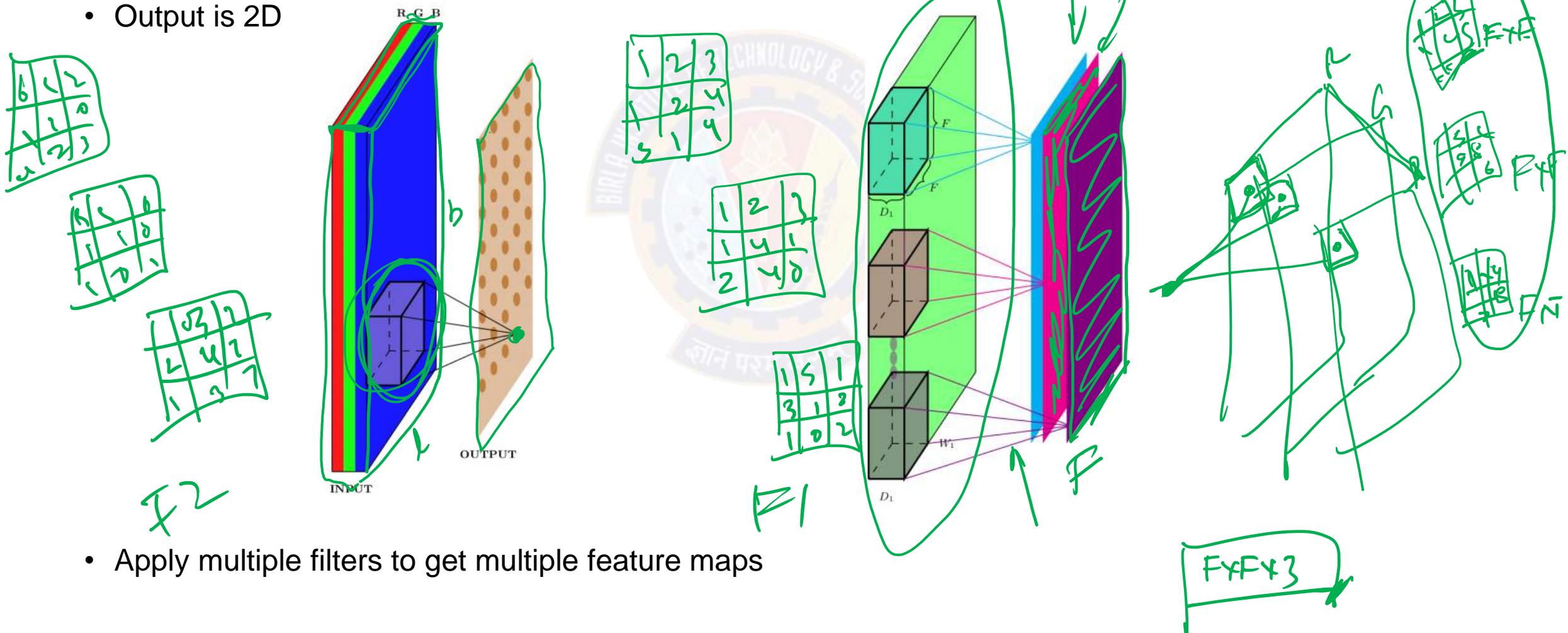
$$* \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & -8 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



detects the edges

# Filter in 3D

- It would refer to volume. Assume the filter extends to the depth
- Output is 2D





# Thank You!

In our next session:



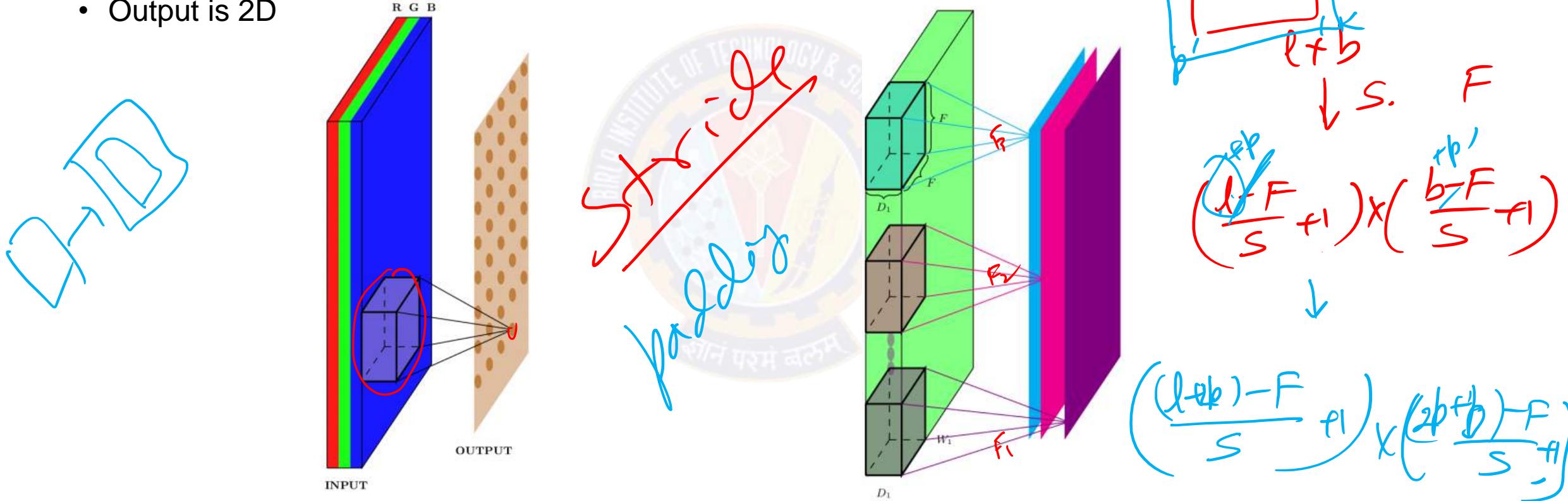
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) 2-D

**Dr. Kamlesh Tiwari**

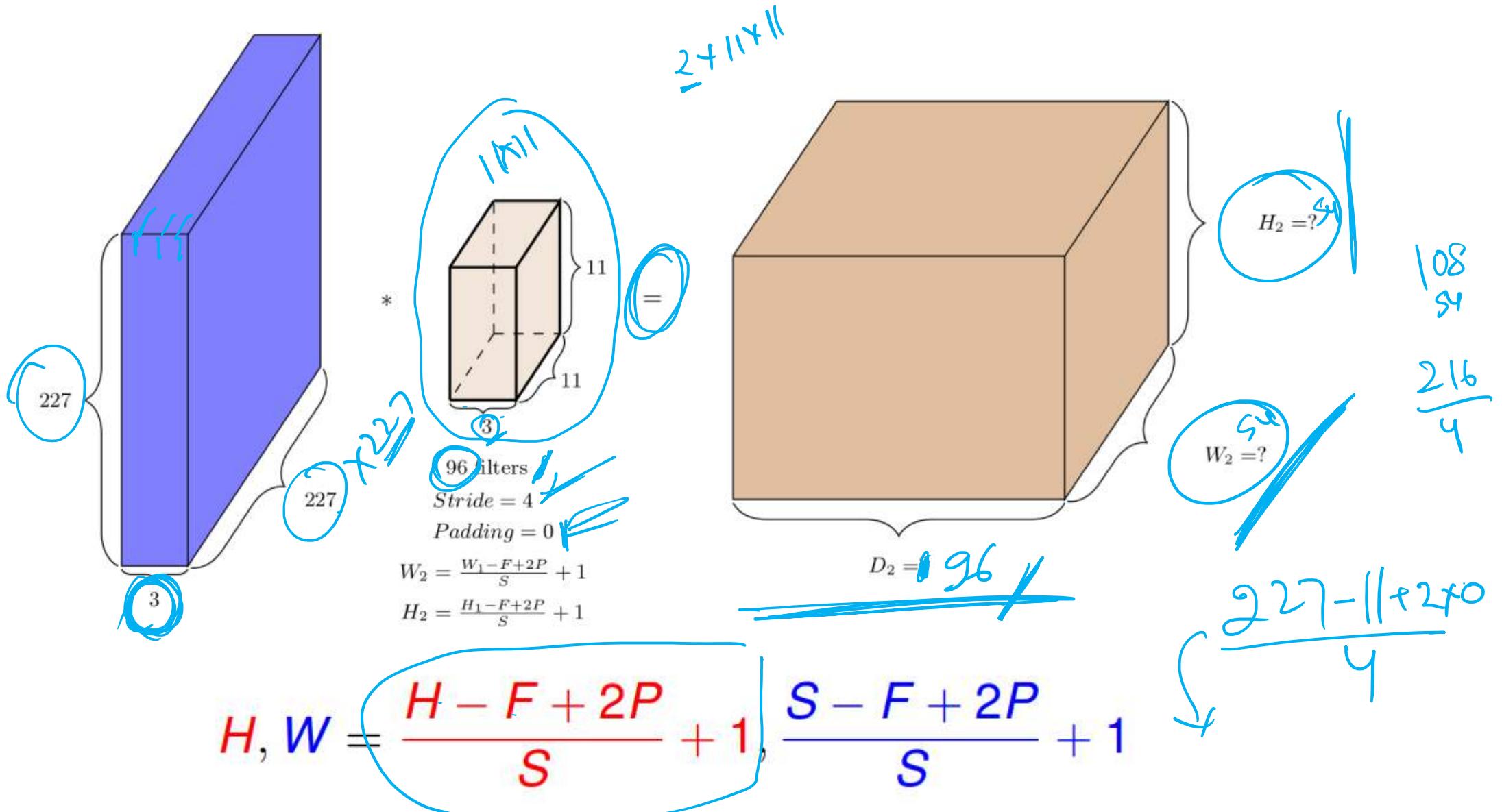
# Filter in 3D

- It would refer to volume. Assume the filter extends to the depth
- Output is 2D



- Apply multiple filters to get multiple feature maps

# Filters, Padding and Stride





**Thank You!**

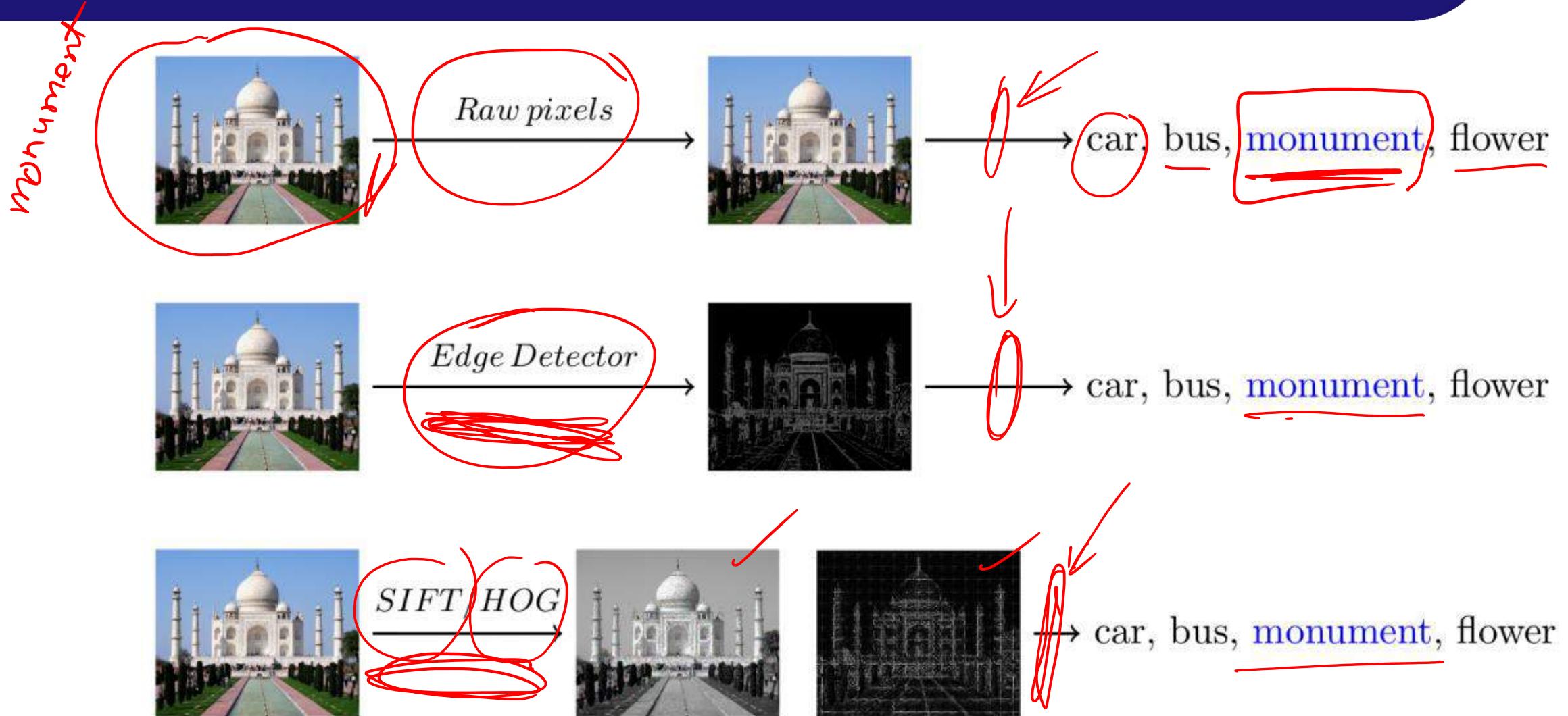


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) Classification Pipeline

**Dr. Kamlesh Tiwari**

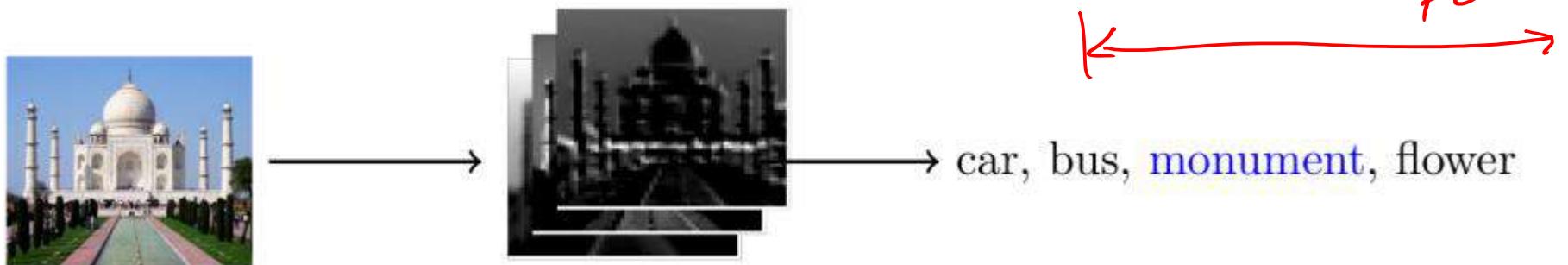
# Classification Pipeline



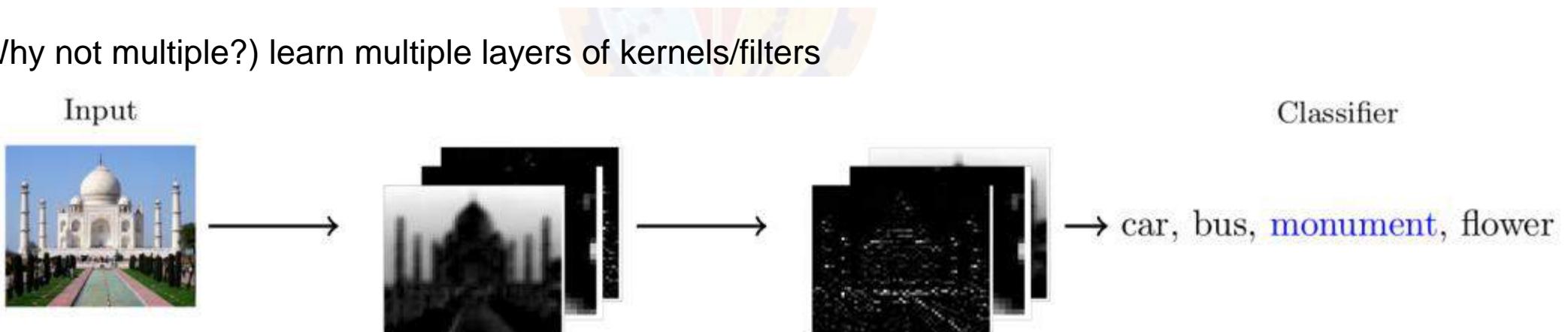
- Where is learning? (hand craft features, then learn weights for classification). One can see feature as a convolution.

# Automate feature kernel discovery

- Instead of handcrafted kernels, learn filters



- (Why not multiple?) learn multiple layers of kernels/filters



Treating these kernels as parameters and learning them.



# Thank You!

In our next session: sparse connectivity in CNN

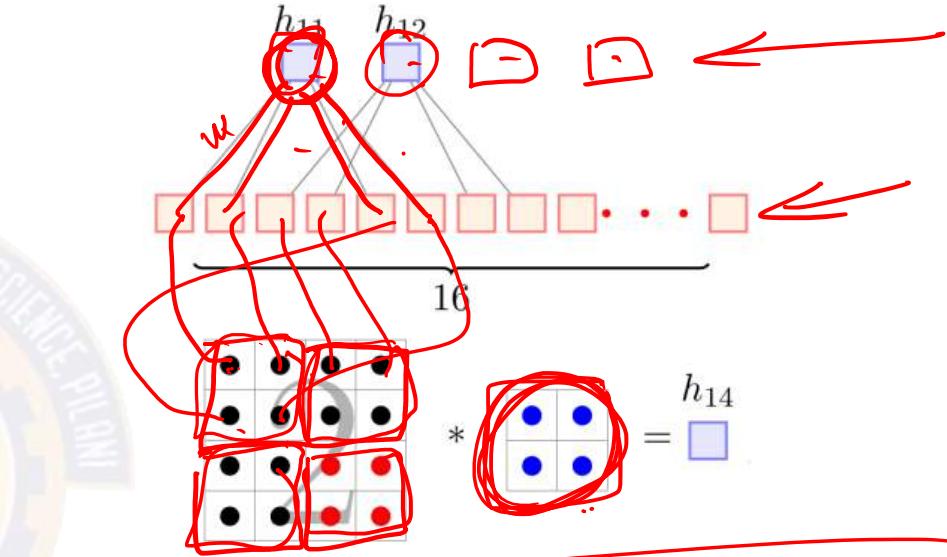
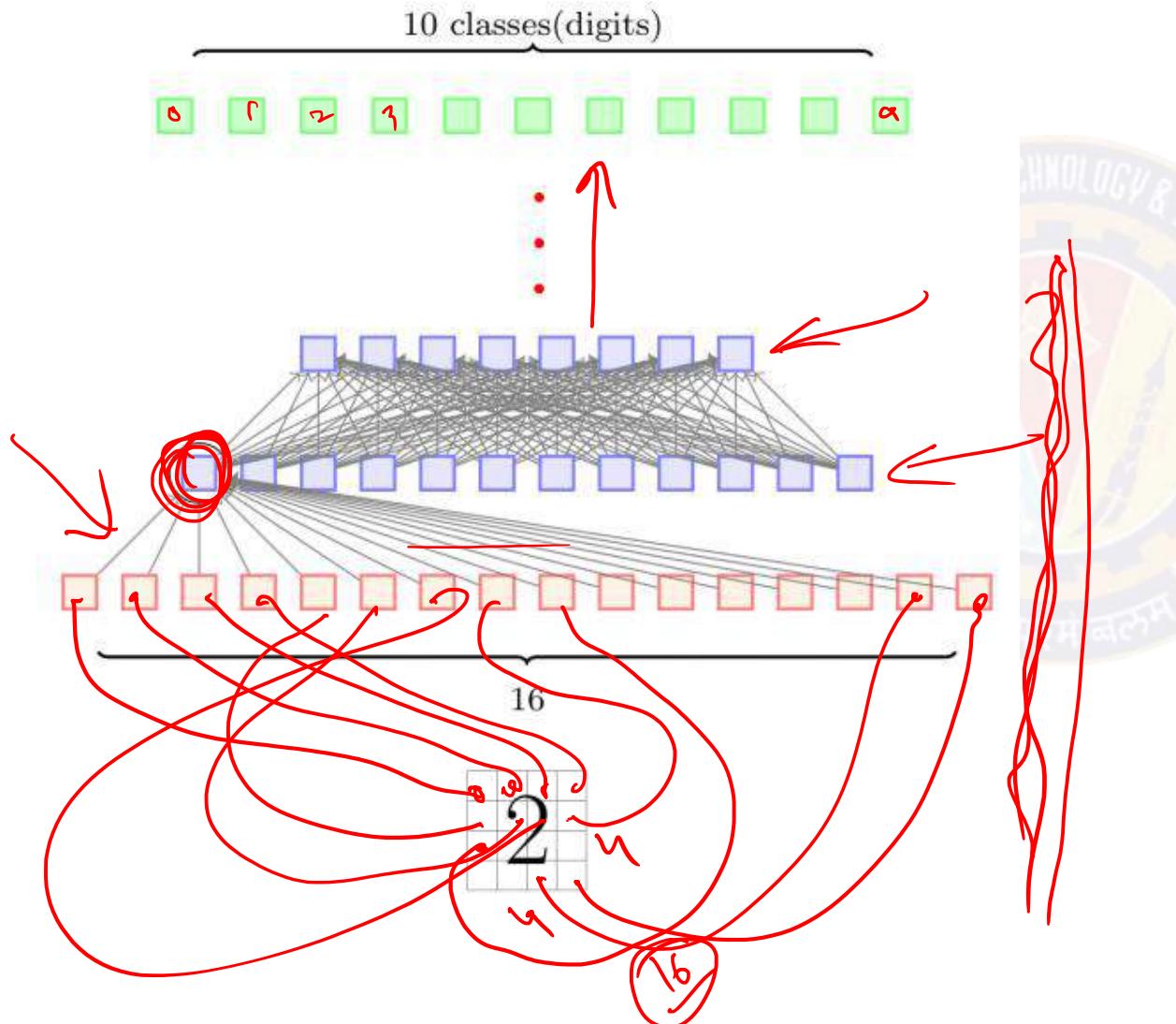


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

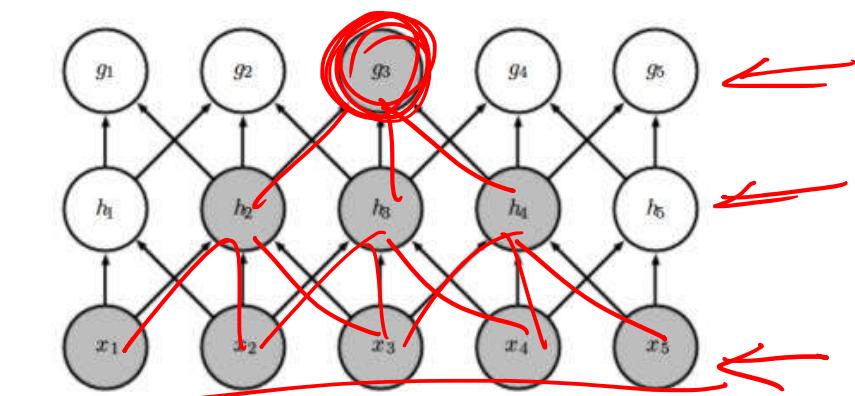
# CONVOLUTIONAL NEURAL NETWORKS (CNN) Sparse Connectivity in CNN

**Dr. Kamlesh Tiwari**

# CNN has sparse connectivity with respect to NN

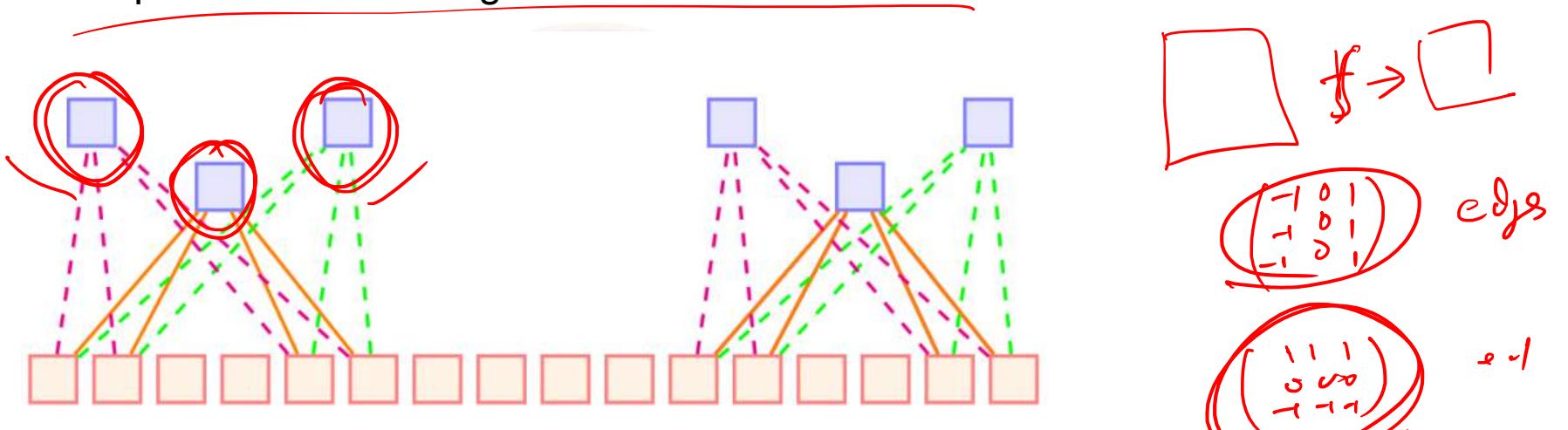


Interactions are preserved, even with reduced model parameters,

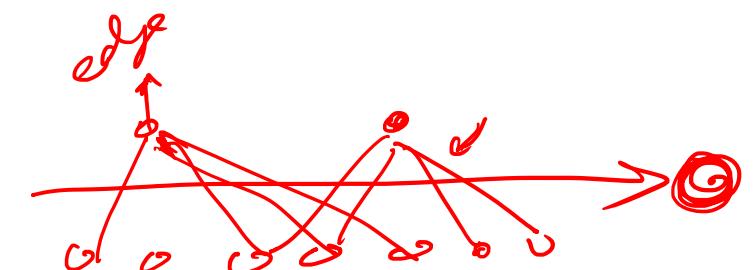


# Weight sharing in CNN

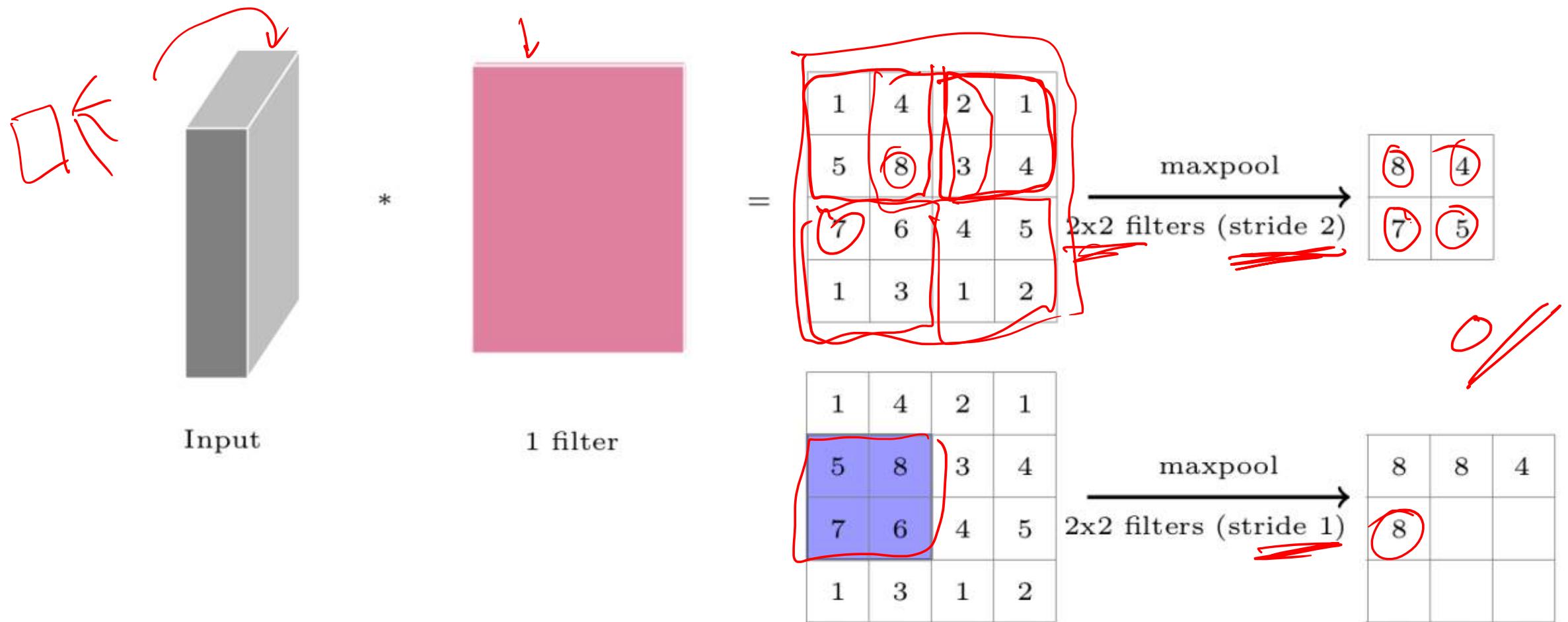
- One place you are extracting edge other place something else? So we do not want the kernel to be different for different portions of the image.



- Weight sharing in CNN makes the job of learning weights easier
- Multiple kernels help get different feature at the same level



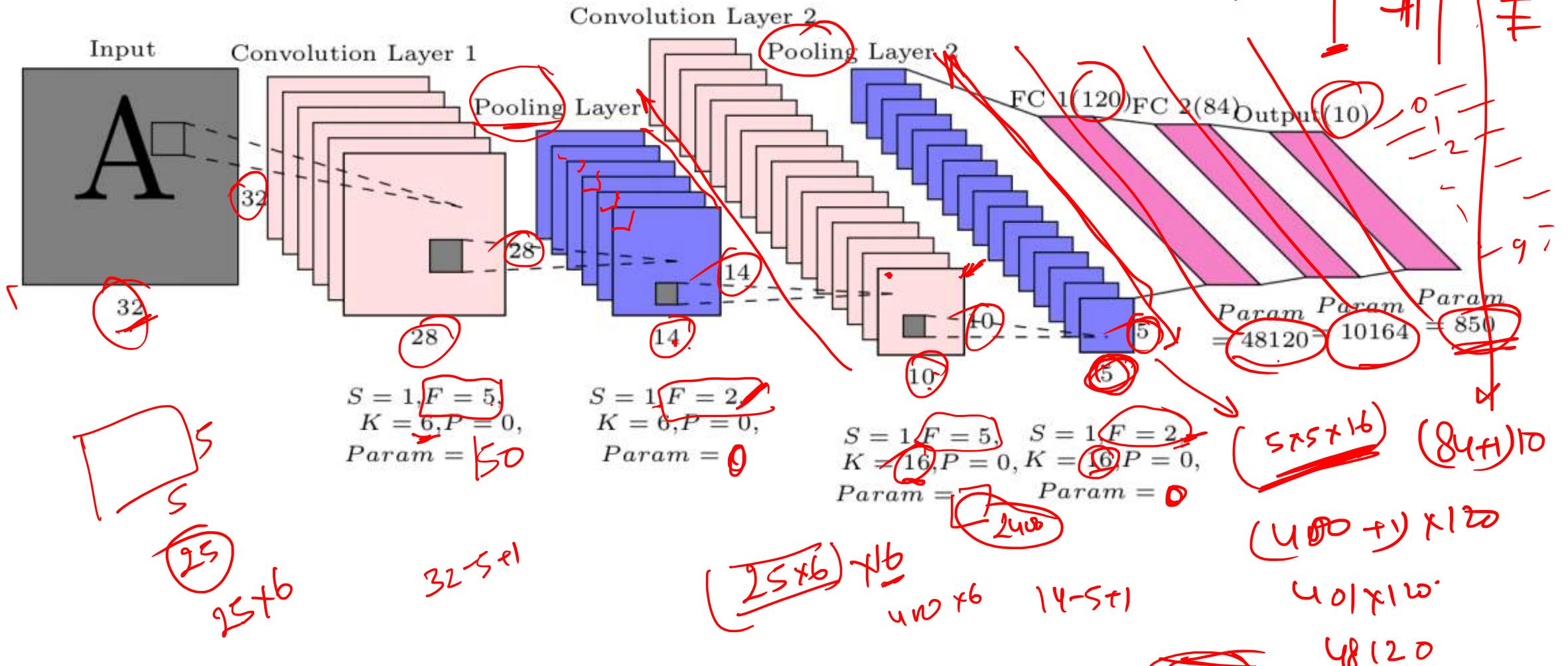
# Pooling (max, min, average)



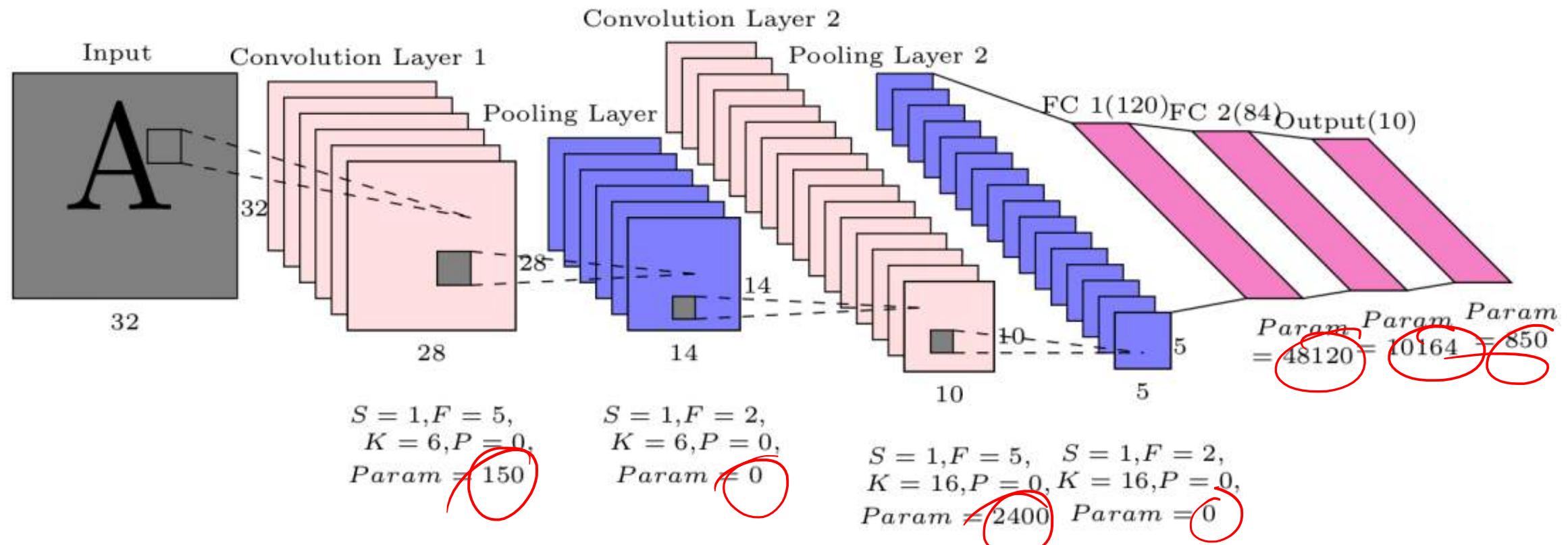
- Training CNN? backpropagation!

# Case-Study: LeNet-5

Handwritten character recognition



# LeNet-5 for handwritten character recognition





# Thank You!

In our next session: Popular CNN Architectures

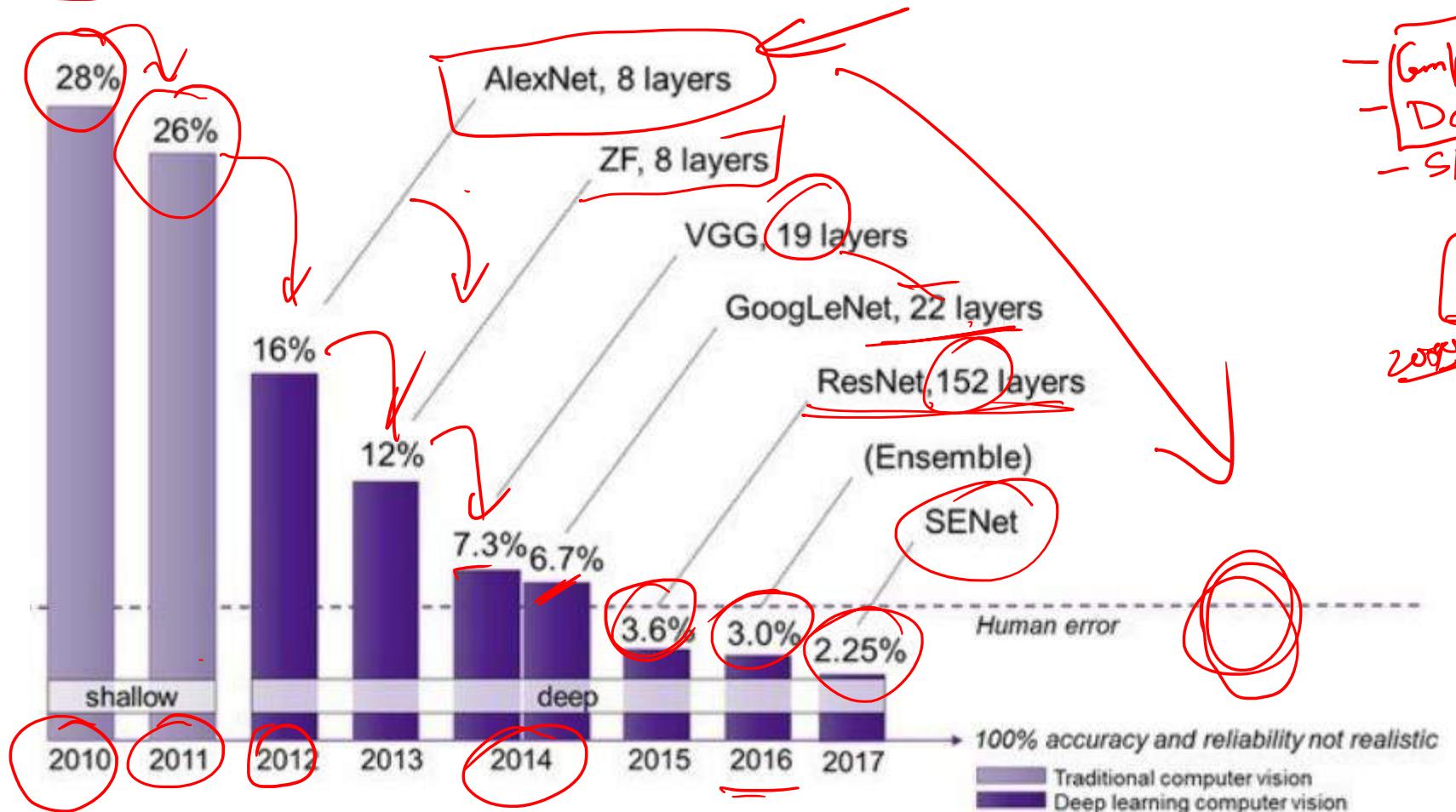


**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) Popular Architectures

**Dr. Kamlesh Tiwari**

# ImageNet ILSVRC



- (2009) 22K category, 14M images *aff, n, C, t...*
- Challenge 1000 class, 1431167 images
- HoG, LBP, SVM ...



# Thank You!

In our next session: Popular CNN Architectures

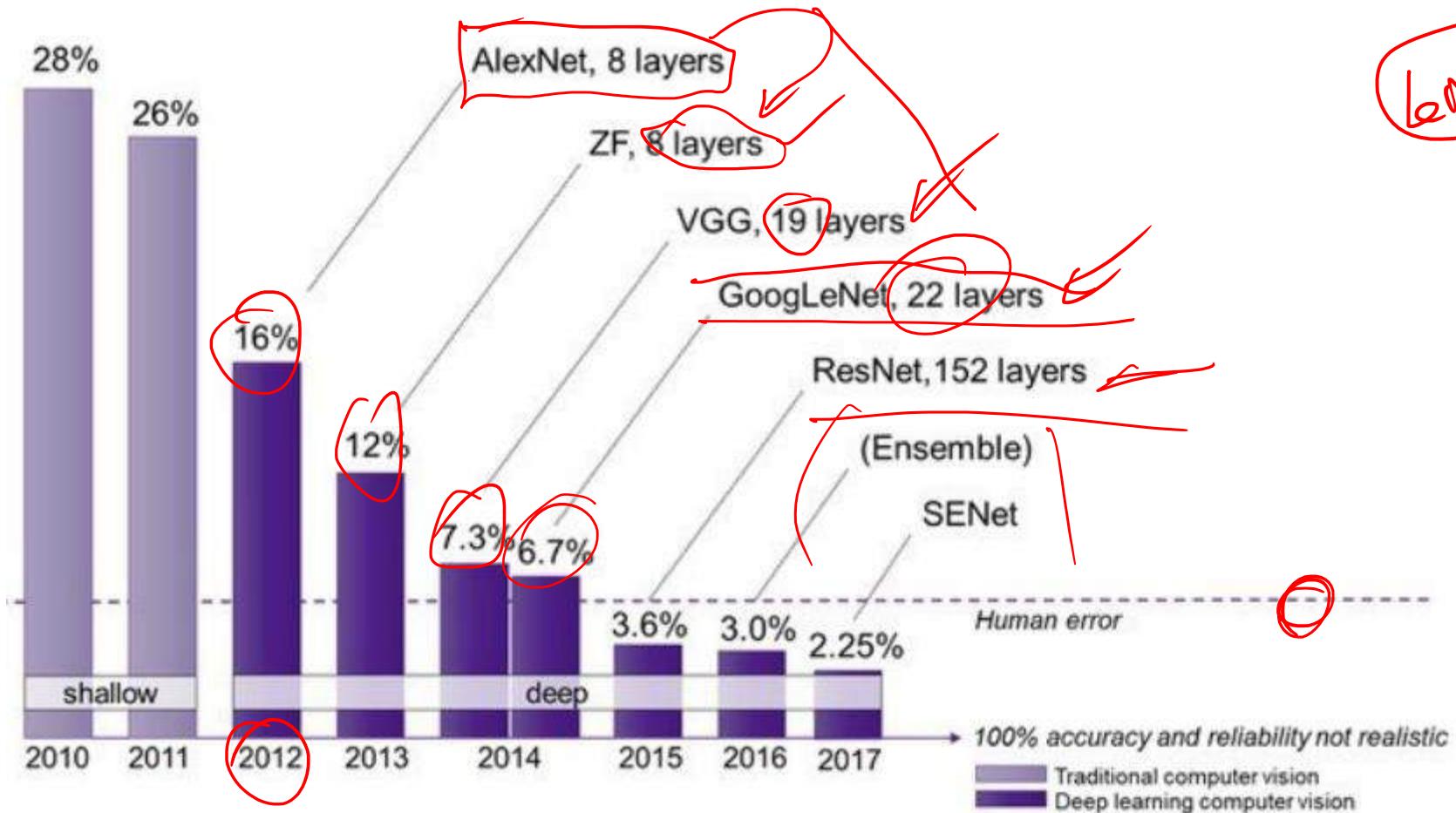


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) AlexNET, ZF-NET, VGG-16

**Dr. Kamlesh Tiwari**

# ImageNet ILSVRC



- (2009) 22K category, 14M images
- Challenge 1000 class, 1431167 images
- HoG, LBP, SVM ...

# AlexNet

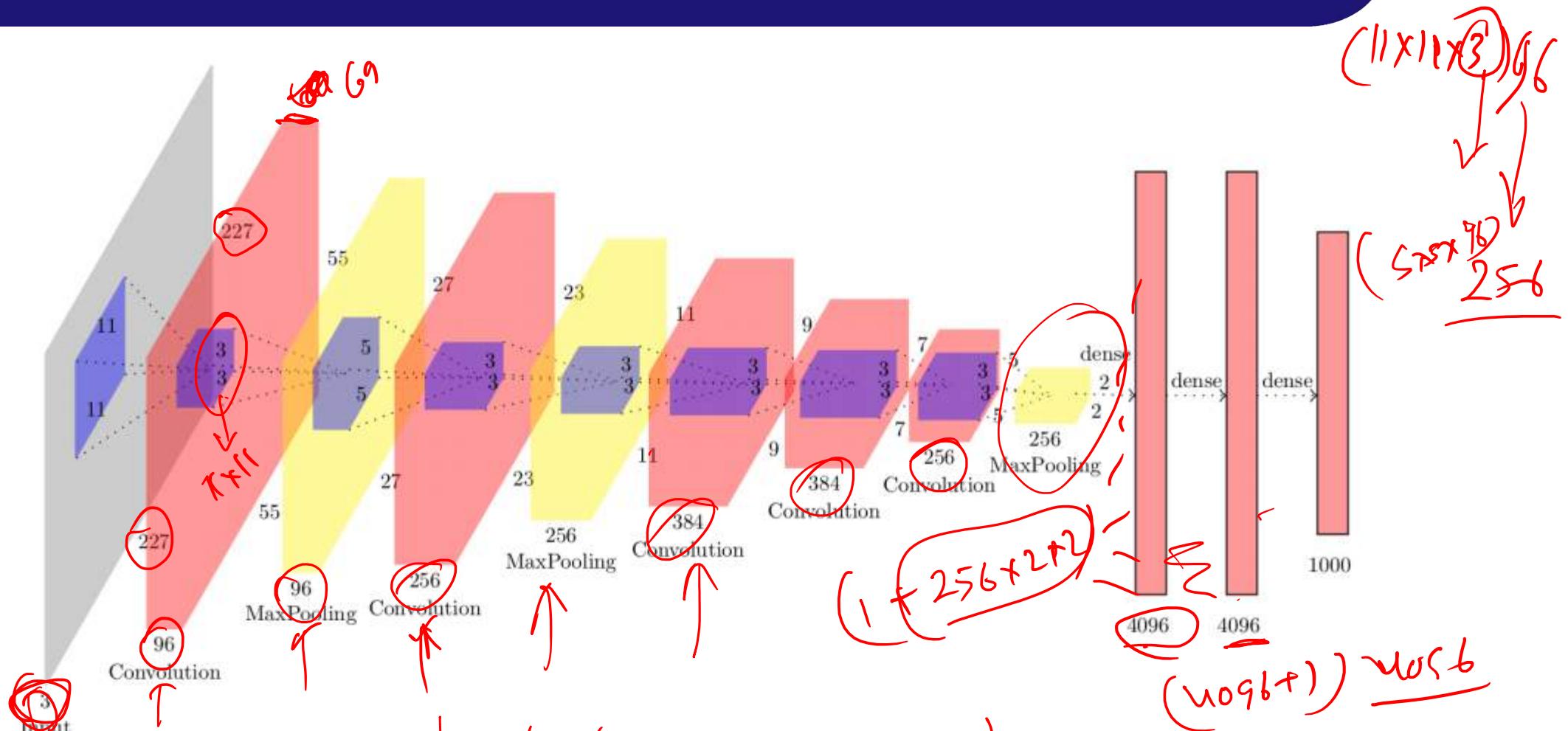


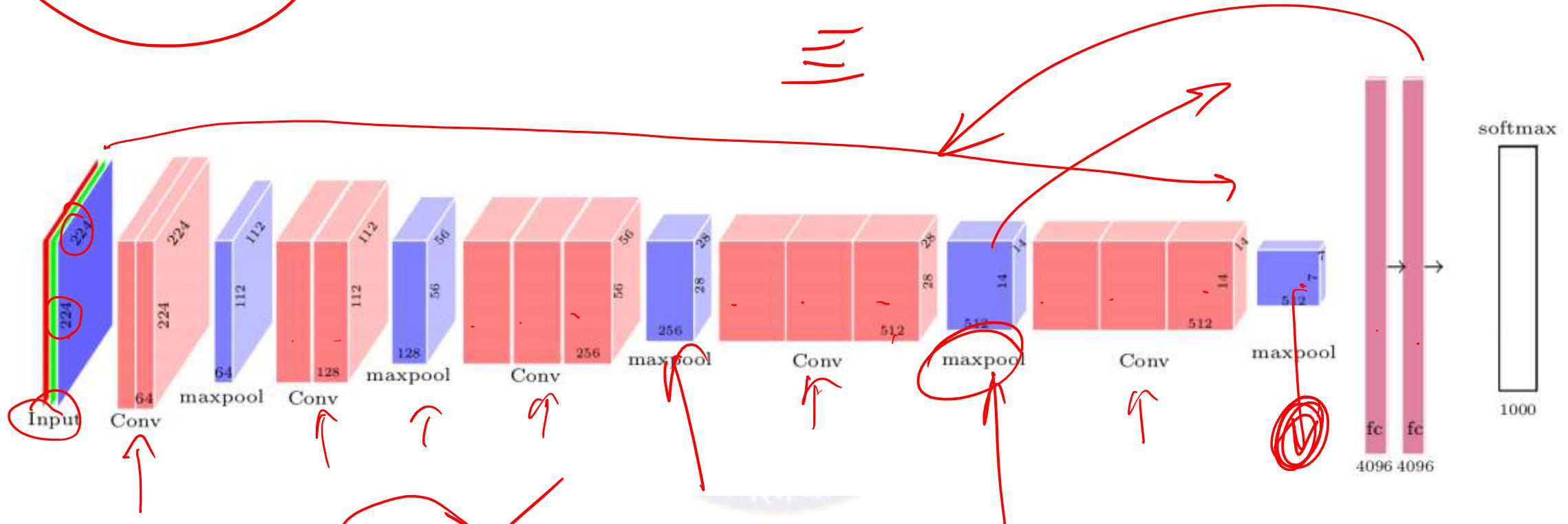
Image size  $227 \times 227 \times 3 \rightarrow [96 \text{ F=11, S=4, P=0}] \rightarrow [\text{MaxPool F=3, S=2}] \rightarrow [256 \text{ F=5, S=1, P=0}] \rightarrow [\text{MaxPool F=3, S=2}] \rightarrow [384 \text{ F=3, S=1, P=0}] \rightarrow [384 \text{ F=3, S=1, P=0}] \rightarrow [256 \text{ F=3, S=1, P=0}] \rightarrow [\text{MaxPool F=3, S=2}] \rightarrow \text{FC 4096} \rightarrow \text{FC 1000}$

# ZFNet



Image size  $227 \times 227 \times 3 \rightarrow [69 F=7, S=4, P=0] \rightarrow [\text{MaxPool } F=3, S=2] \rightarrow [256 F=5, S=1, P=0] \rightarrow [\text{MaxPool } F=3, S=2] \rightarrow [512 F=3, S=1, P=0] \rightarrow [1024 F=3, S=1, P=0] \rightarrow [512 F=3, S=1, P=0] \rightarrow [\text{MaxPool } F=3, S=2] \rightarrow \text{FC } 4096 \rightarrow \text{FC } 1000$

# VGG16



- Kernel size is always  $3 \times 3$
- 16M parameters in pre-FC and 122 in FC. First FC layer is huge
- Layers represent abstract representation and can be reused (FC or Conv)



# Thank You!

In our next session: Popular CNN Architectures Continues...

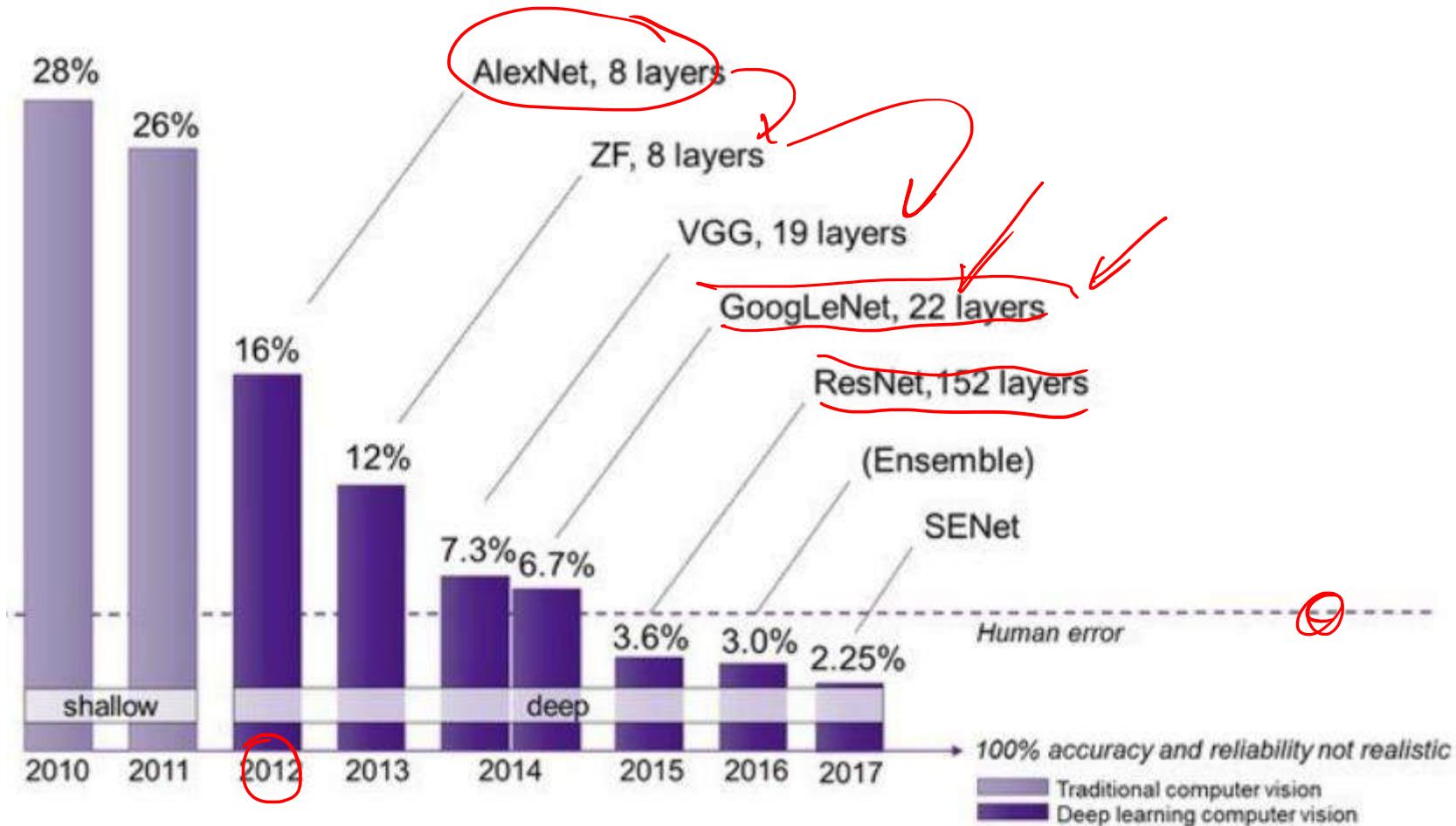


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) Inception, ResNET

**Dr. Kamlesh Tiwari**

# ImageNet ILSVRC



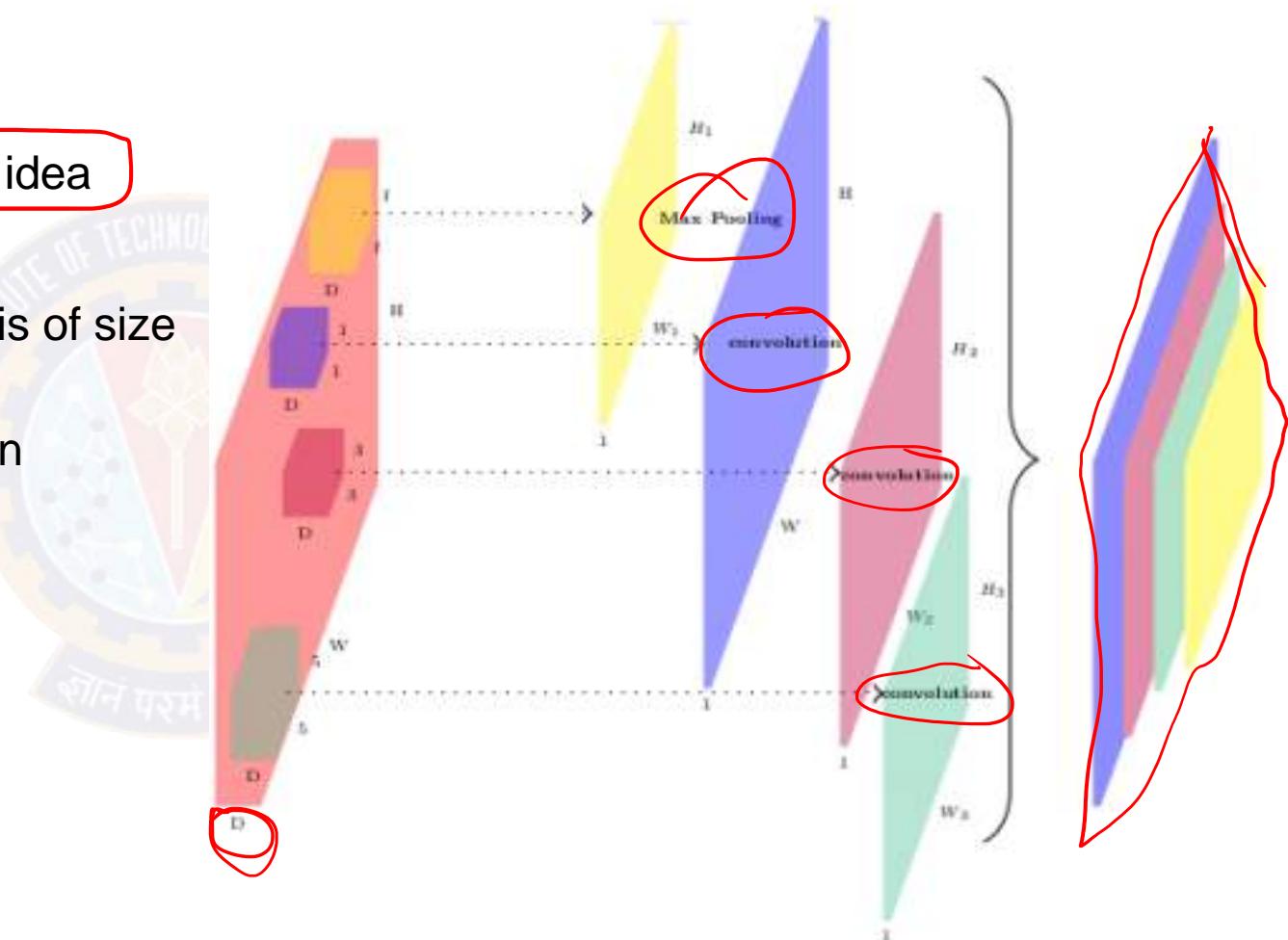
- (2009) 22K category, 14M images
- Challenge 1000 class, 1431167 images
- HoG, LBP, SVM ...

# GoogLeNet

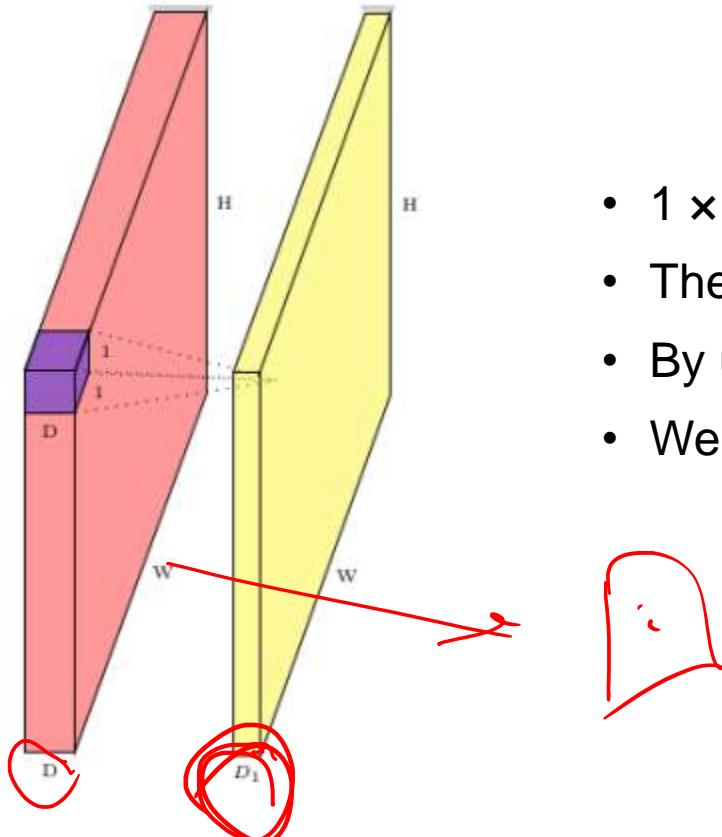
- Recall scale invariance in SIFT
- Multiple filters of different size is a good idea
- With  $W \times H \times D$  input and  $F \times F \times D$
- Filter and  $S = 1$  and no padding, output is of size  $(W - F + 1) \times (H - F + 1)$
- Each value needs  $F \times F \times D$  computation

Can we reduce this computation a bit?

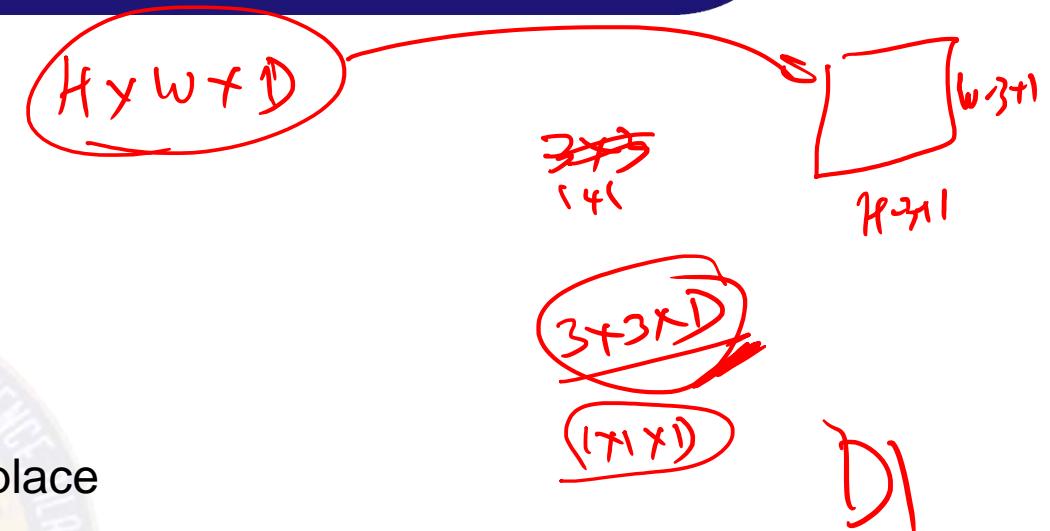
- Idea is to have  $1 \times 1$  computation



# $1 \times 1$ convolution

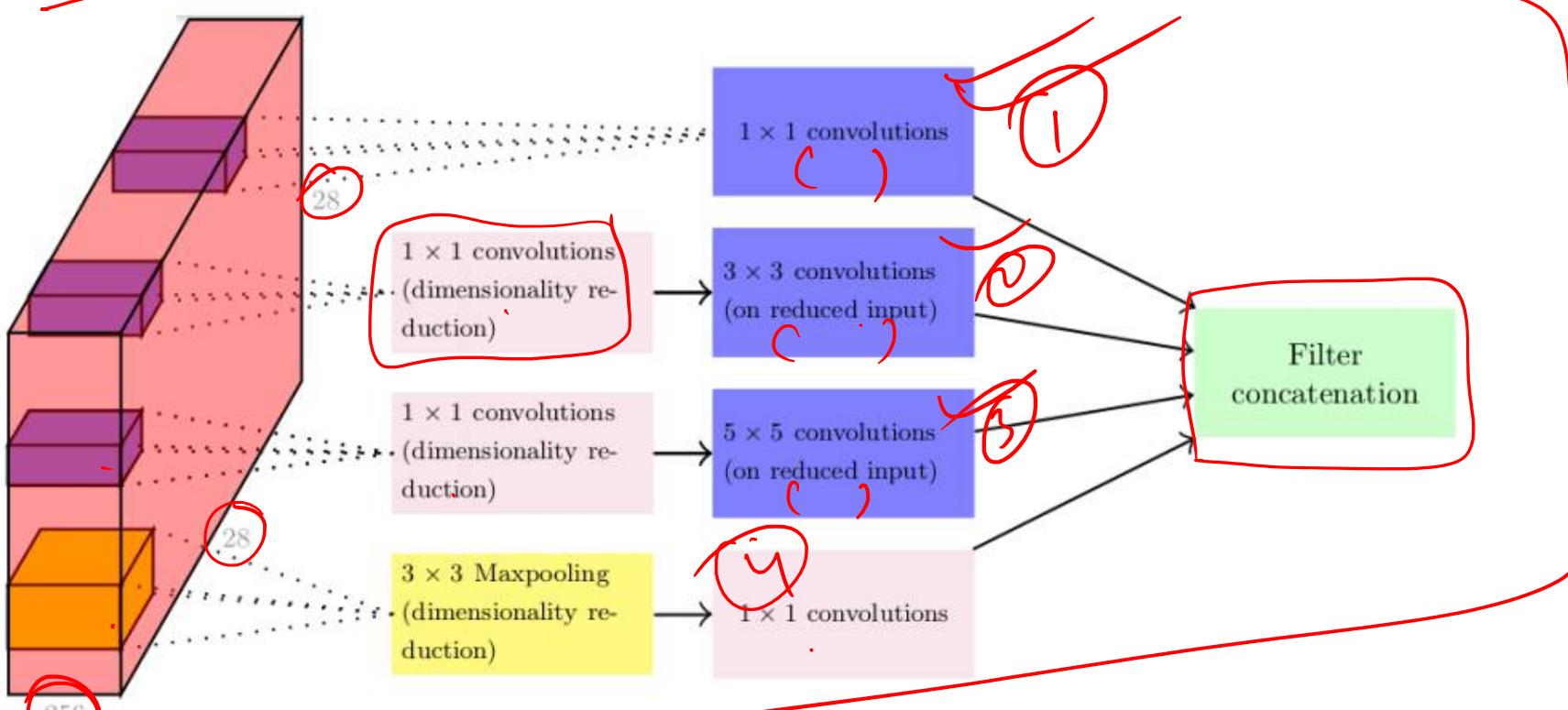


- $1 \times 1$  is  $1 \times 1 \times D$
- They produce one output place
- By using  $D_1$  such  $1 \times 1$  convolution output becomes  $F \times F \times D_1$
- We have  $D_1 < D$



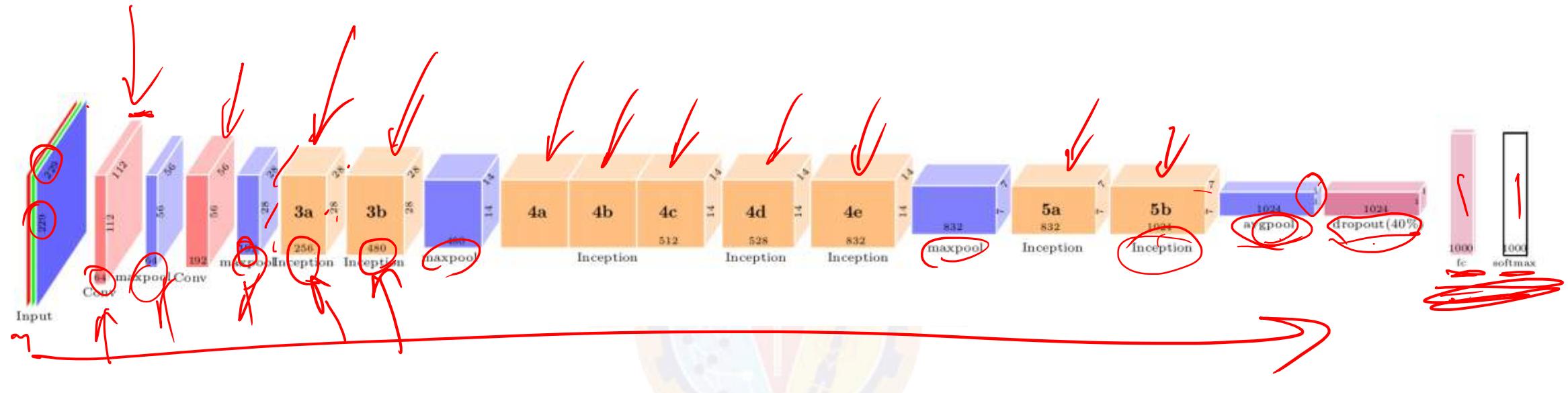
$D_1 < D$

# Inception Block: Multiple convolutions



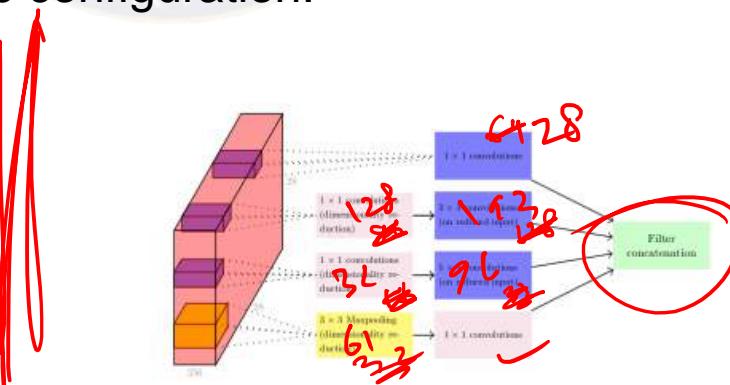
- 1 × 1 convolution
- 1 × 1 convolution followed by 3 × 3
- 1 × 1 convolution followed by 5 × 5
- 3 × 3 maxpool followed by 1 × 1
- Appropriate padding is done to make things of same size

# GoogLeNet

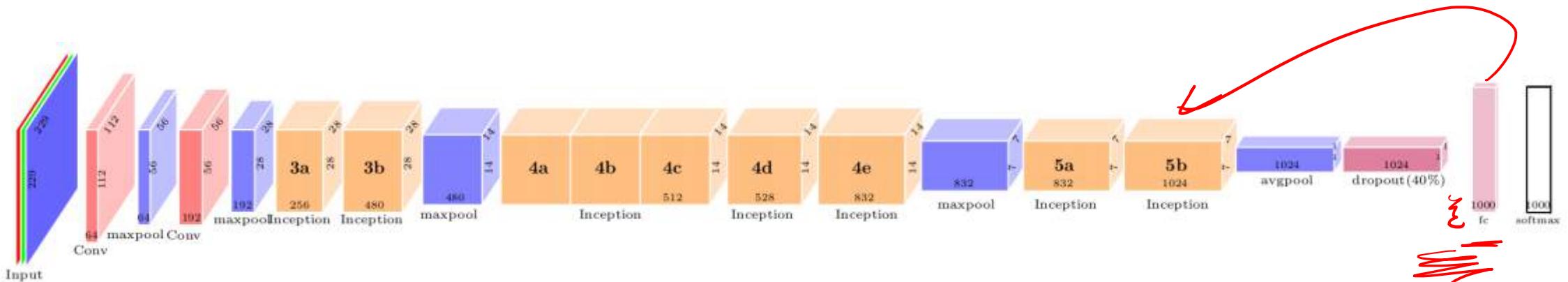


- Input is RGB  $229 \times 229$
- Each inception module have very specific configuration.

(3a)	$192 \times 28 \times 28$	64 96 128 16 32 32
(3b)	$256 \times 28 \times 28$	28 128 192 32 96 64
(4a)	$48 \times 14 \times 14$	192 96 208 16 48 96
(4b)	$512 \times 14 \times 14$	160 112 224 24 64 64
(4c)	$512 \times 14 \times 14$	128 128 256 24 64 64
(4d)	$512 \times 14 \times 14$	112 144 228 32 64 64
(4e)	$528 \times 14 \times 14$	256 160 320 32 128 128
(5a)	$832 \times 7 \times 7$	256 160 320 32 128 128
(5b)	$832 \times 7 \times 7$	384 192 384 48 124 128



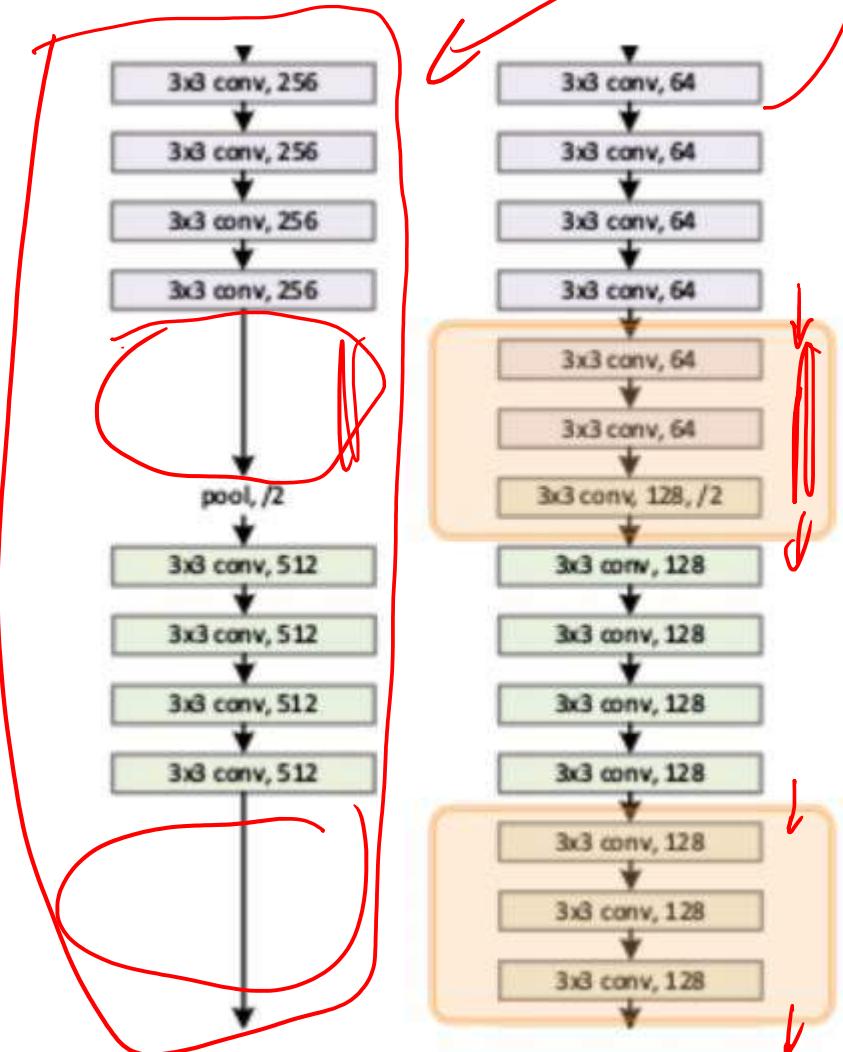
# GoogLeNet



- VGGNET has  $512 \times 7 \times 7$  size at pre-FC this was an issue to connect with 4096
- GoogLeNet applies a average pool. Gives 49 time reduction. has 1024 values only
- Dropout and connect to 1000
- 12 times less connections as compared to AlexNet
- 2 times more computation as compared to AlexNet
- Very high accuracy. Error reduced from 16% -to- 6.7%

# ResNet

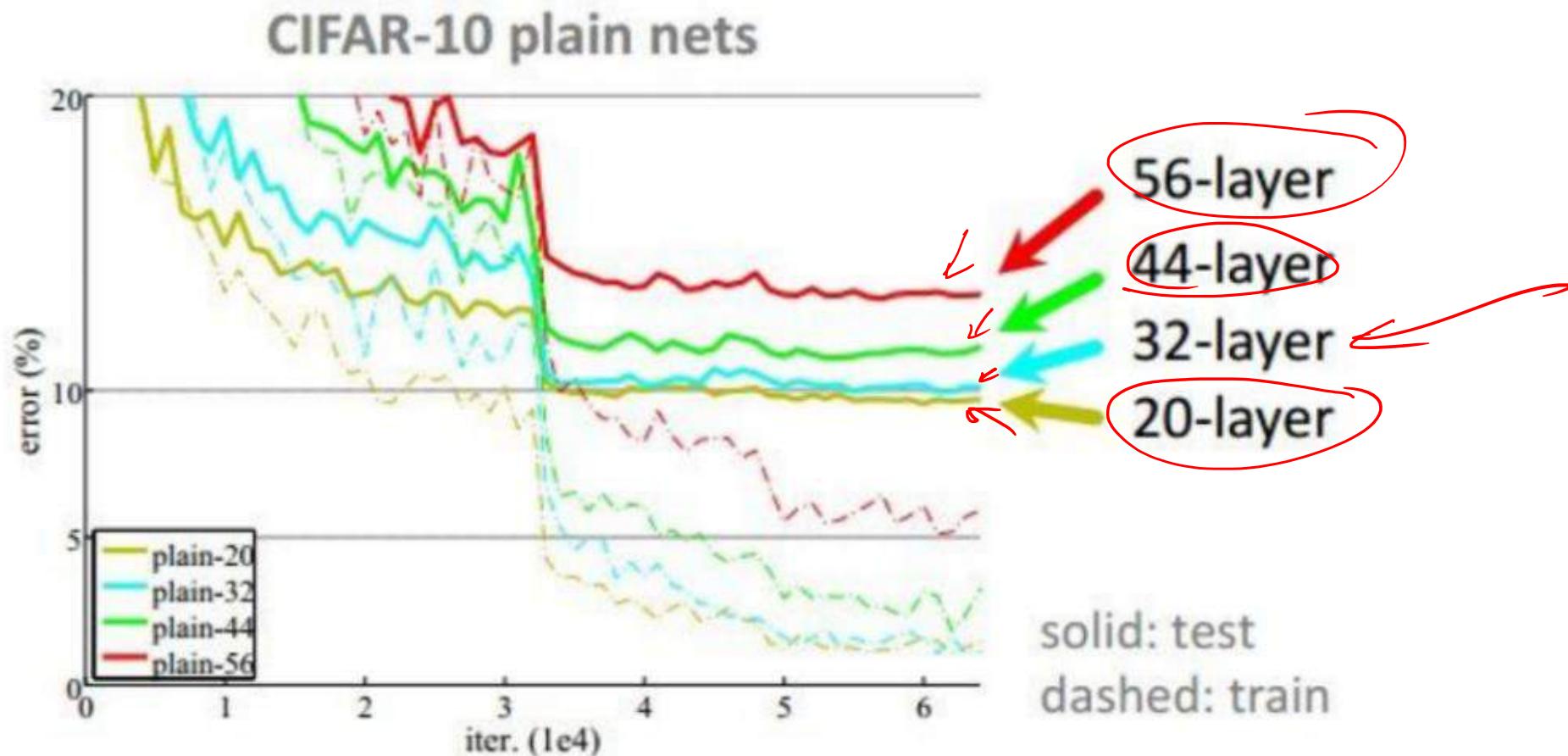
If a shallow neural network works well. What would happen if we add more layers?



- Deep network should also work well (It would learn identity in new layers)

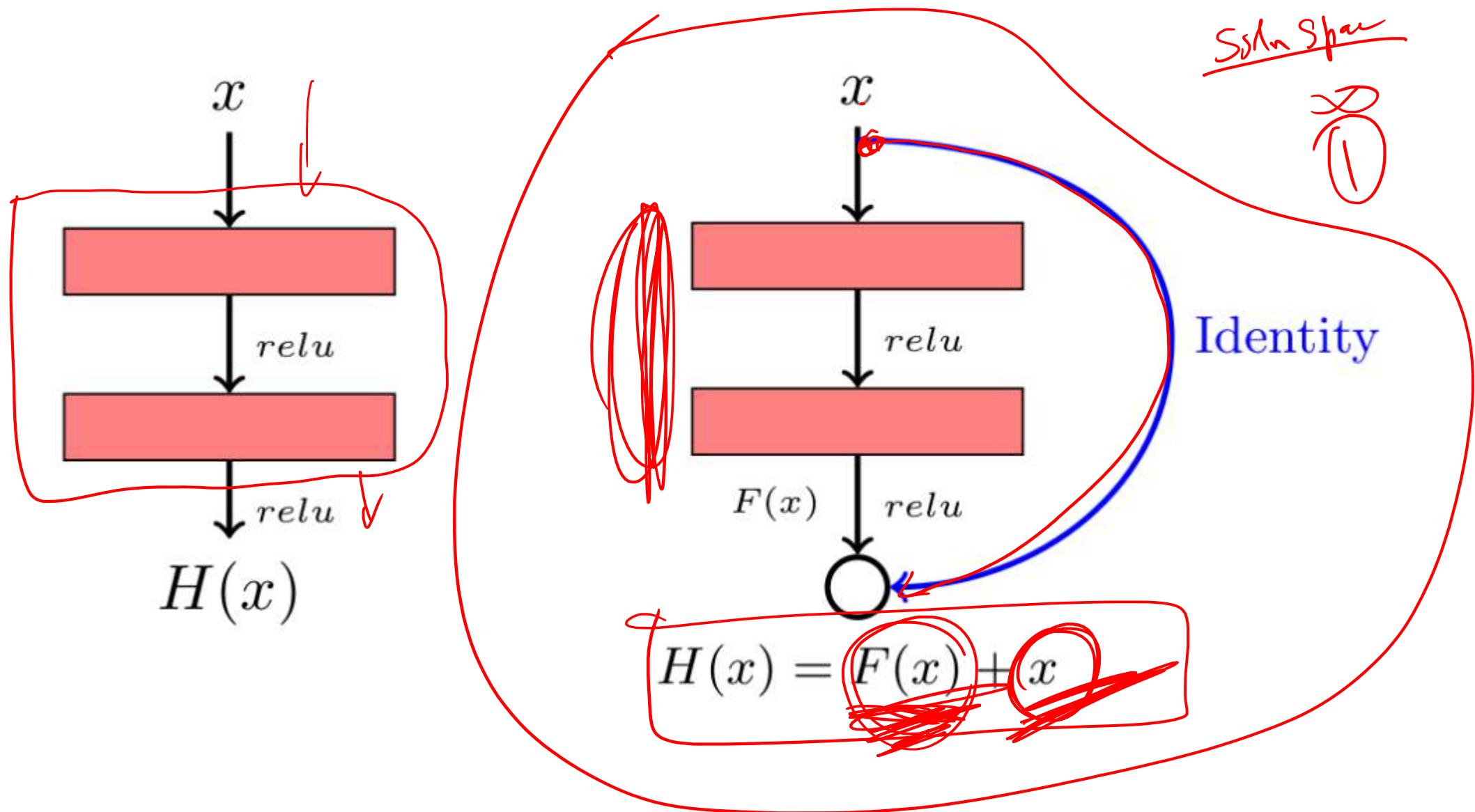
# ResNet

But, in practice it was not happening

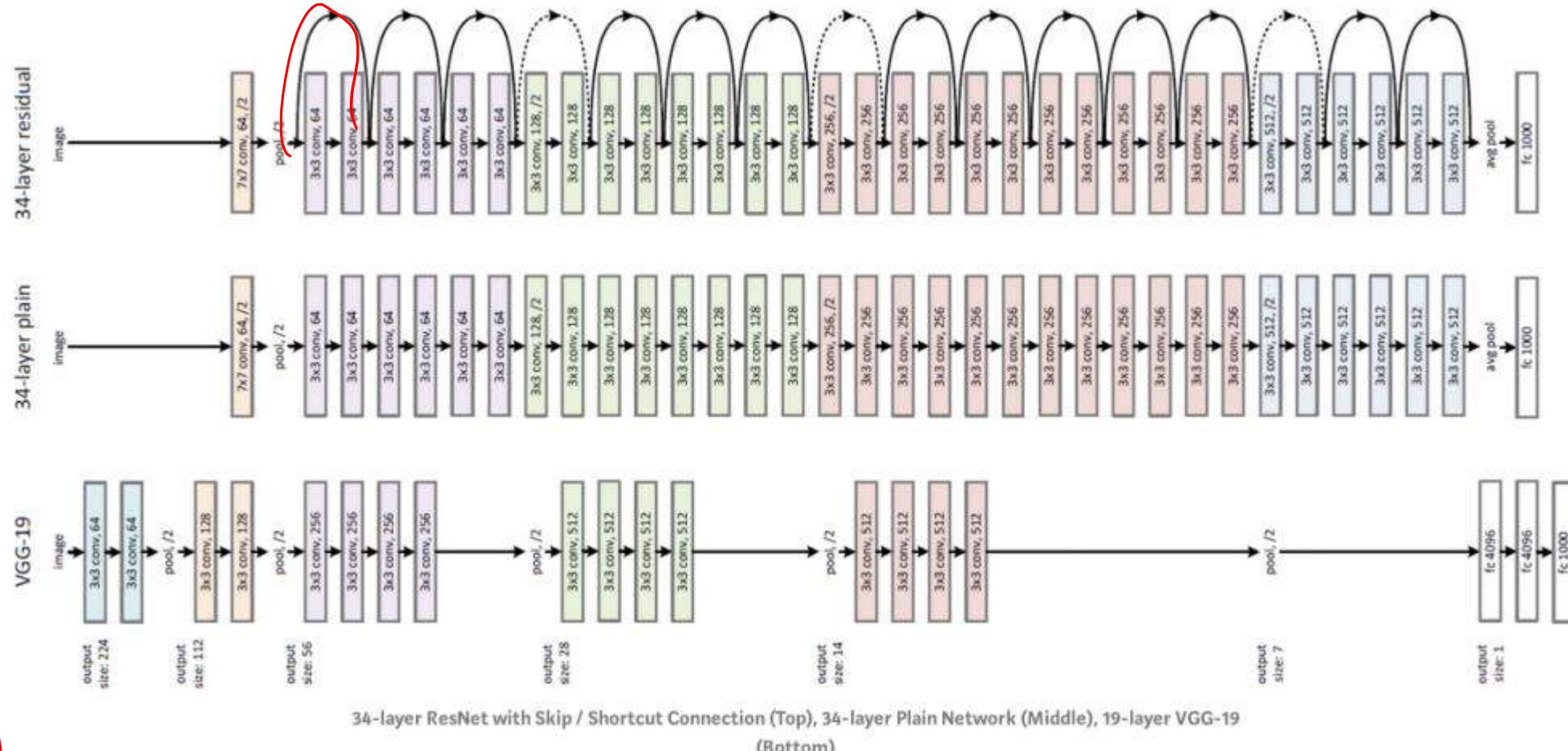


Why? Identity is one of the solution in large domain.

# Let me tell this to the network



# ResNet Comparison



152-layer deep net was better than human. Only 3.6% error rate ImageNet Classification

Better than the 2nd best system ImageNet Detection: 16% ImageNet Localization: 27% COCO  
Detection: 11% COCO Segmentation: 12%

# ResNet Hyper-parameters and Issues

- Training takes huge time ✓
- Batch Normalization ✓
- Zavier/2 initialization ✓
- SGD and momentum ✓
- Small learning rate 0.1 ✓
- Mini-batch size 256 ✓
- Weight decay ✓
- No Dropout ✓





# Thank You!

In our next session: Object Detection and Segmentation

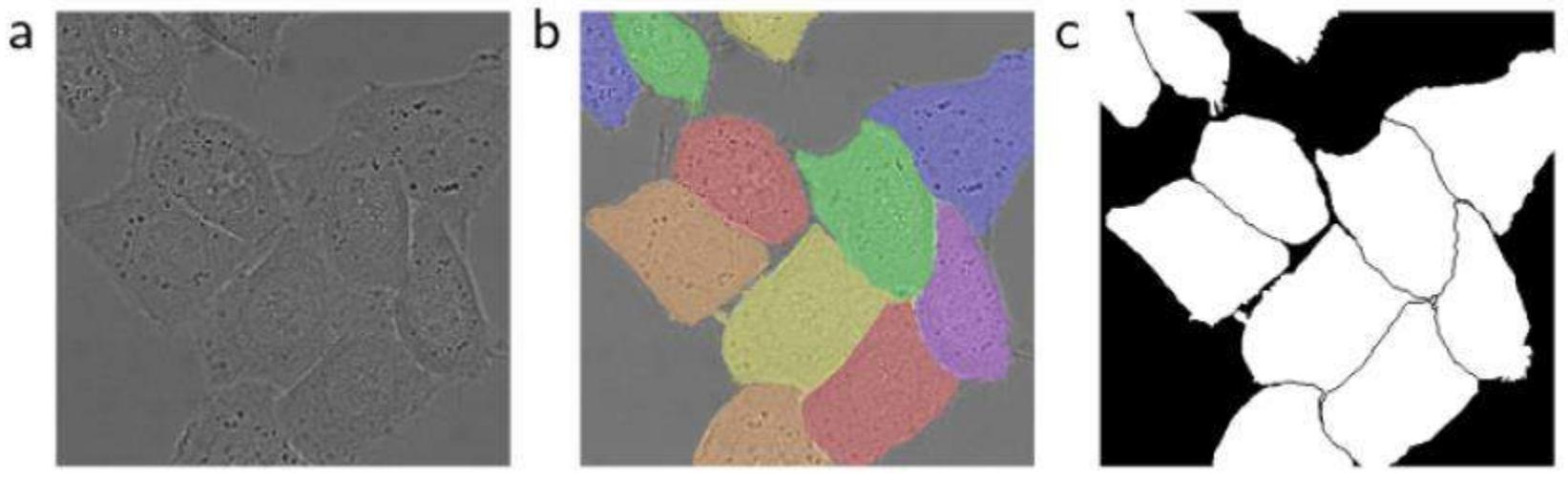


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CNN Case Studies (U-Net)

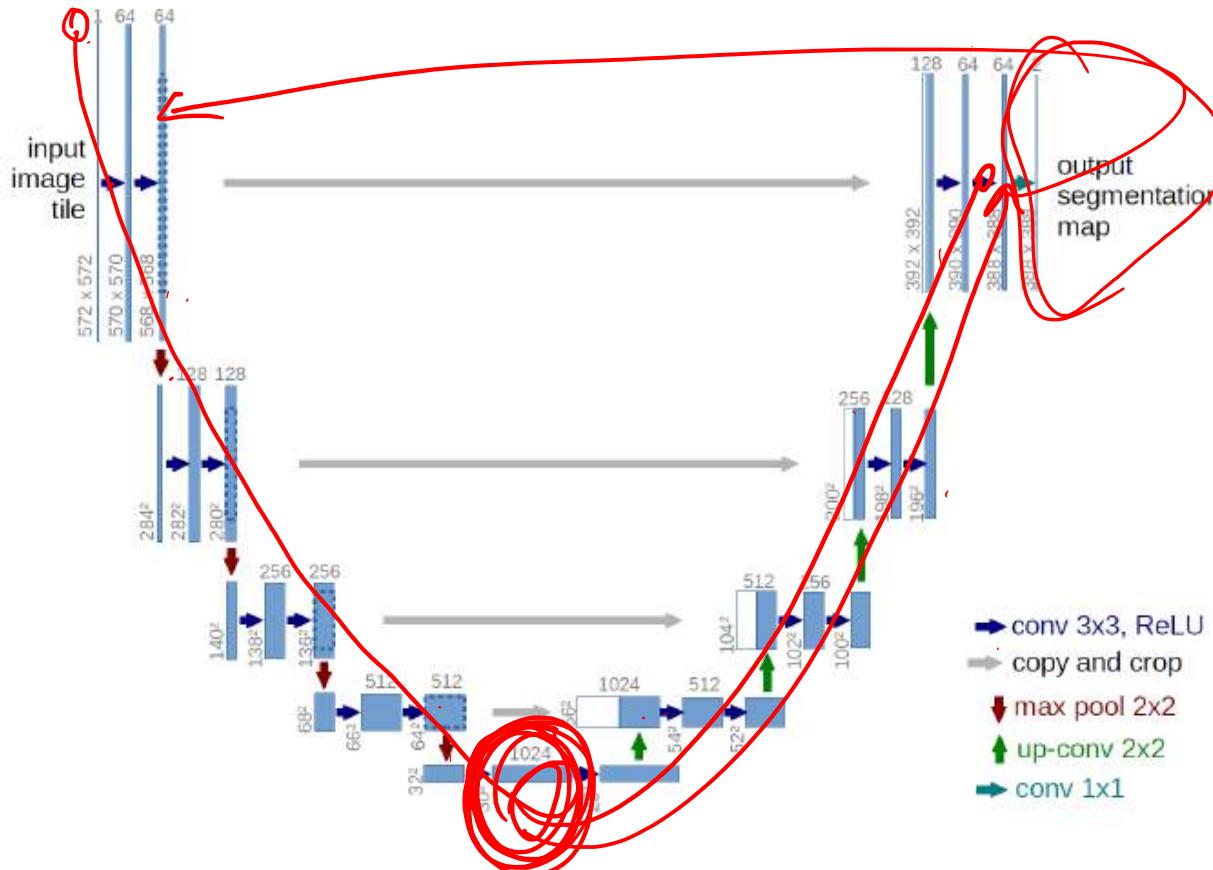
**Dr. Kamlesh Tiwari**

# Segmentation



- ISBI challenge for segmentation of neuronal structures in electron microscopic stacks
- Works with very few training images (30/application) and touching boundary. Yield more precise segmentation
- Data augmentation is essential (mainly shift, rotation and elastic deformation)

# U-Net



- ISBI DIC-HeLa achieved 77.6% IoU as compared to 46.0% second
- ISBI Cell tracking 2015, achieved 92% IoU as compared to 83% second



# Thank You!

Object Detection and Localization (R-CNN):

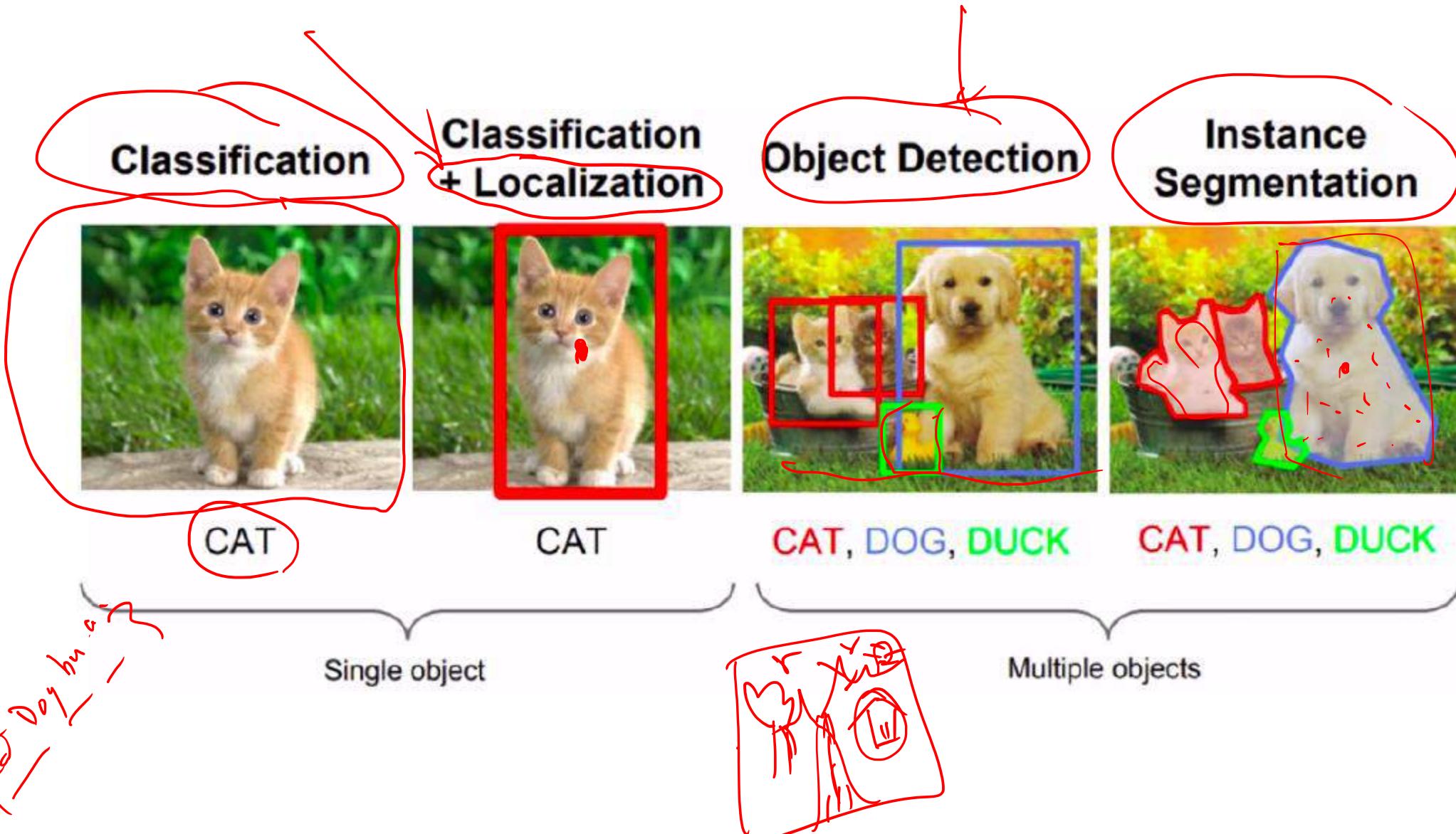


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

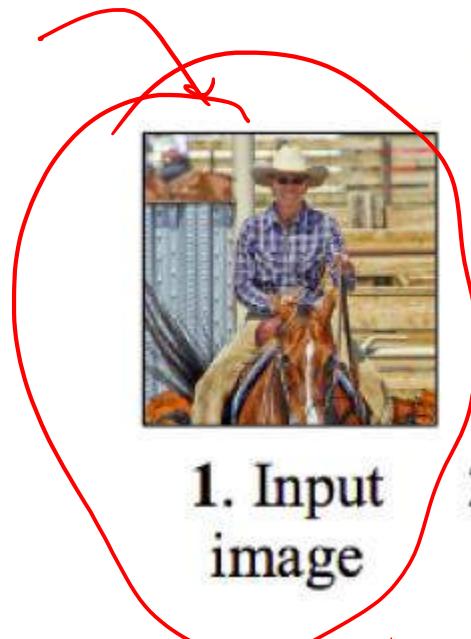
# CNN Case Studies (R-CNN)

**Dr. Kamlesh Tiwari**

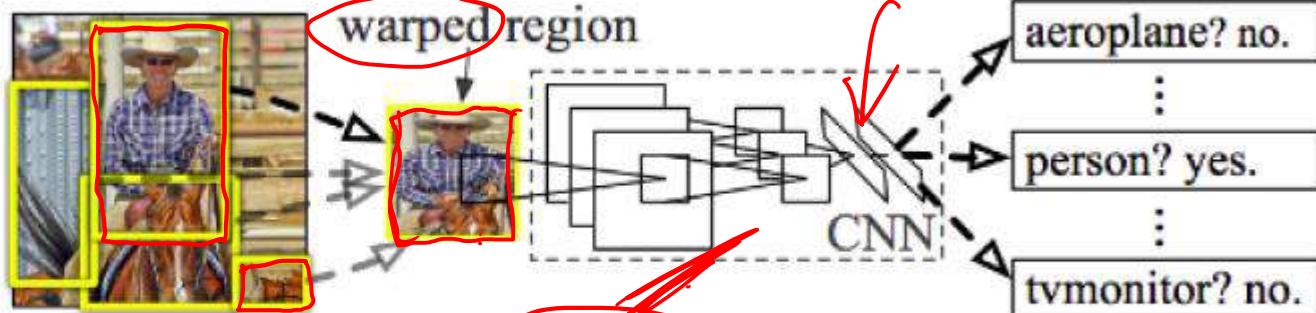
# Object Detection and Localization



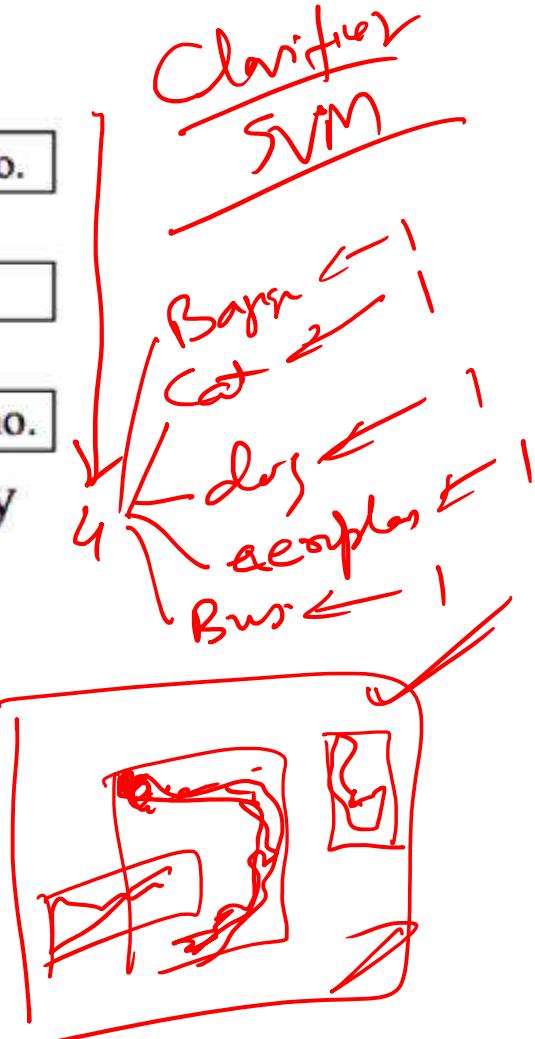
# R-CNN



## R-CNN: *Regions with CNN features*



4. Classify regions



- Region proposals 20K (from external selective search)
- Warped image (resizing)
- SVM for classification (one for each class)

# Thank You!

Object Detection and Localization (Fast and Faster R-CNN):



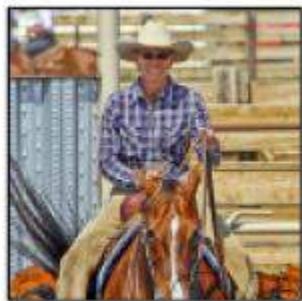
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CNN Case Studies (Fast R-CNN)

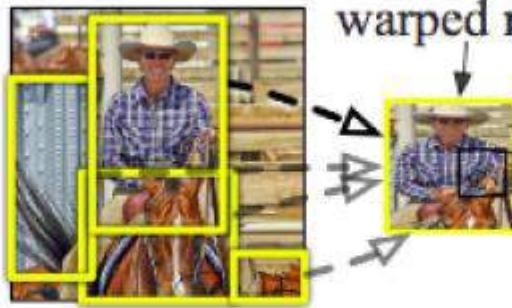
**Dr. Kamlesh Tiwari**

# R-CNN

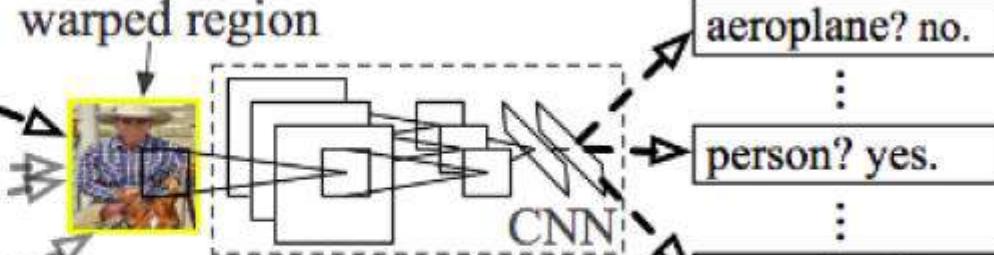
## R-CNN: *Regions with CNN features*



1. Input image



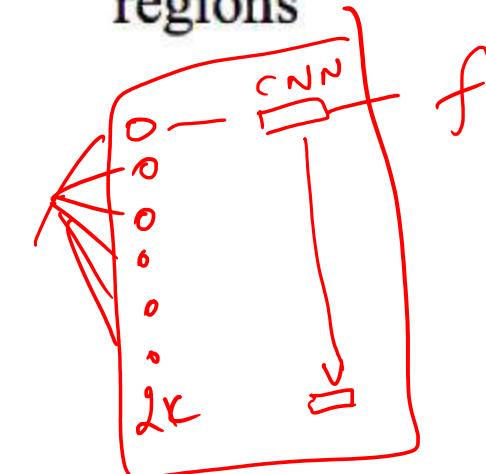
2. Extract region proposals (~2k)



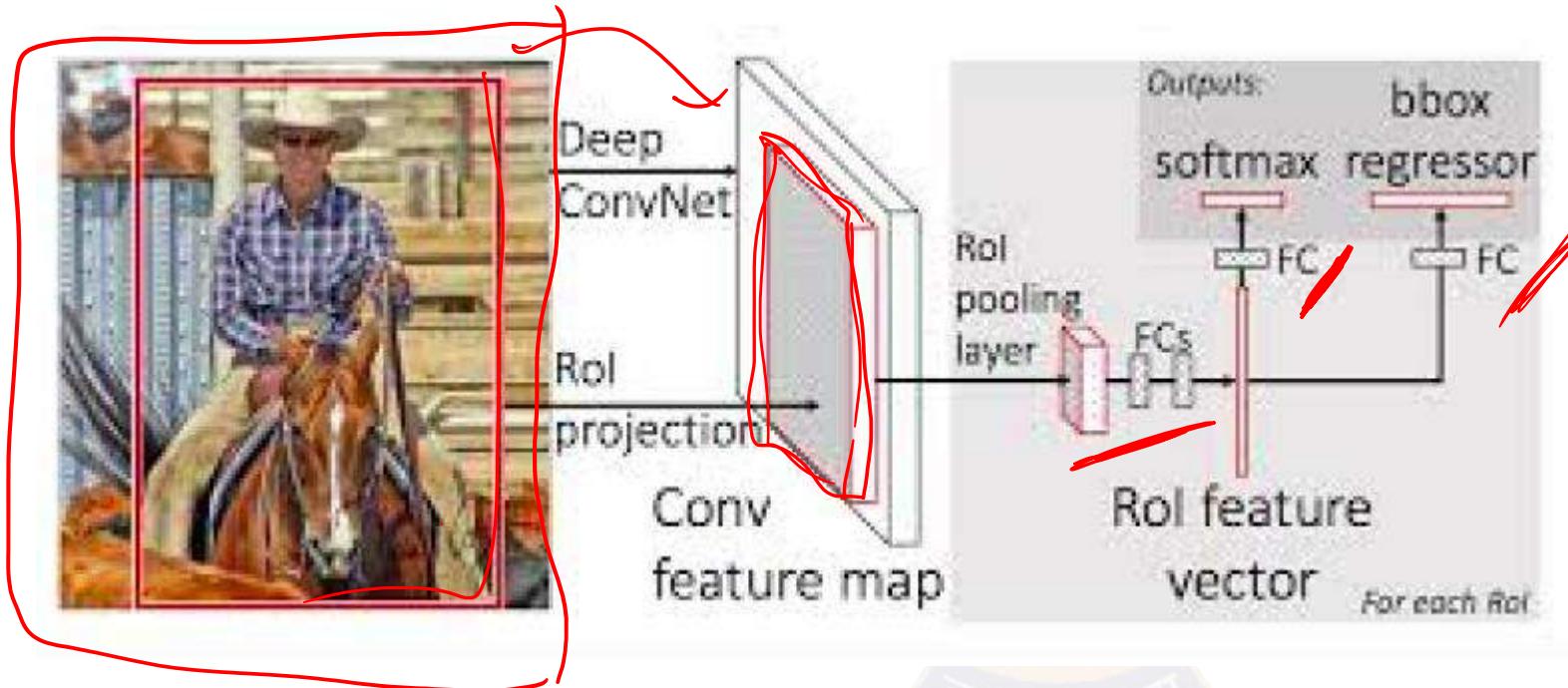
3. Compute CNN features

4. Classify regions

- Region proposals (from external **selective search**)
- Warped image (resizing)
- SVM for classification (one for each class)



# Fast R-CNN



- RoI pooling
- Multi-task loss

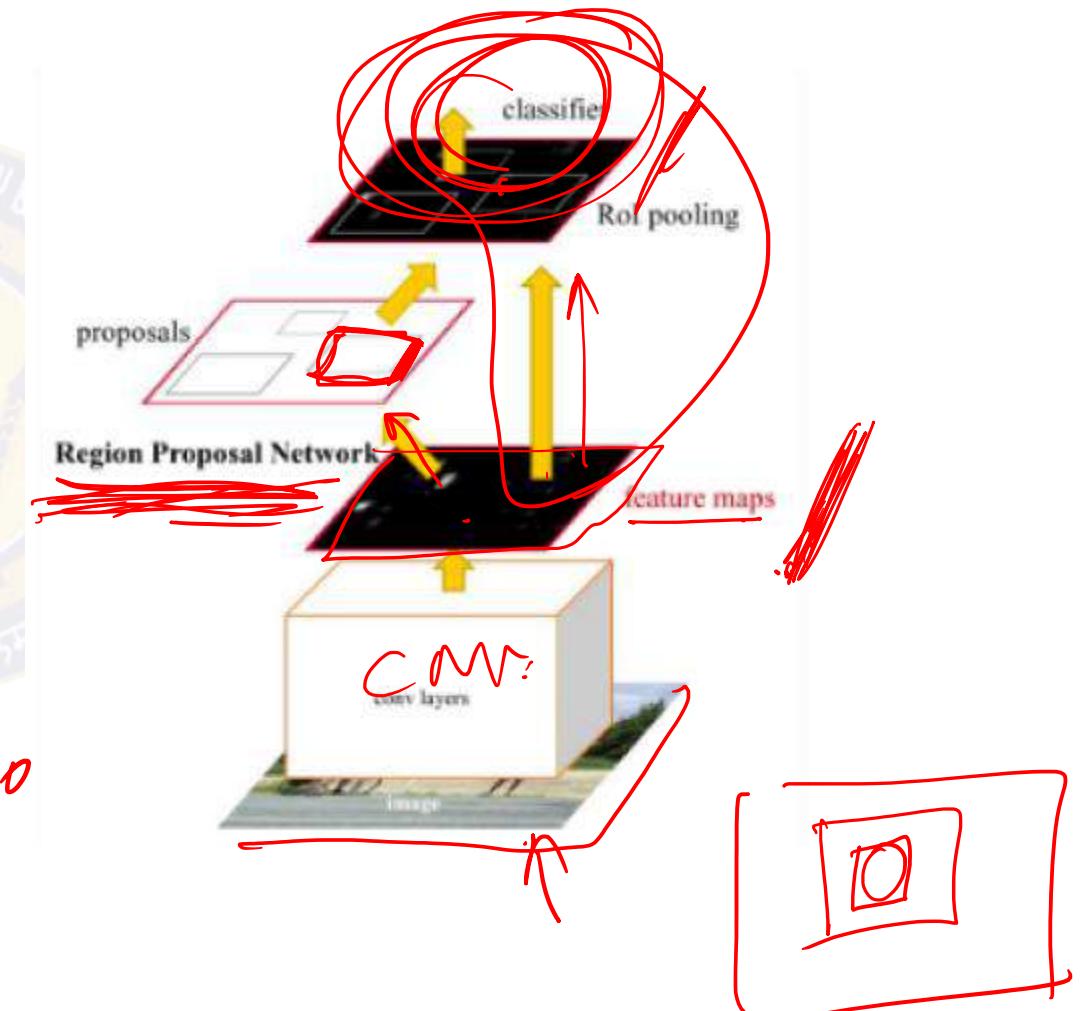


# Faster R-CNN

- Region Proposal Network (RPN)
- Four loss: RPN classification loss, RPN regress loss, Final classification loss, Final box coordinate loss
- 250 time faster than R-CNN. (Fast R-CNN is 25 times fast)

RCNN → 1<sup>st</sup>  
Fast R-CNN → 2<sup>nd</sup>  
faster R-CNN → 250

Cite 7885 Girshick, Ross Fast R-CNN, International Conference on computer vision, pages 1440–1448, IEEE-2015





# Thank You!

Object Detection and Localization (Yolo):



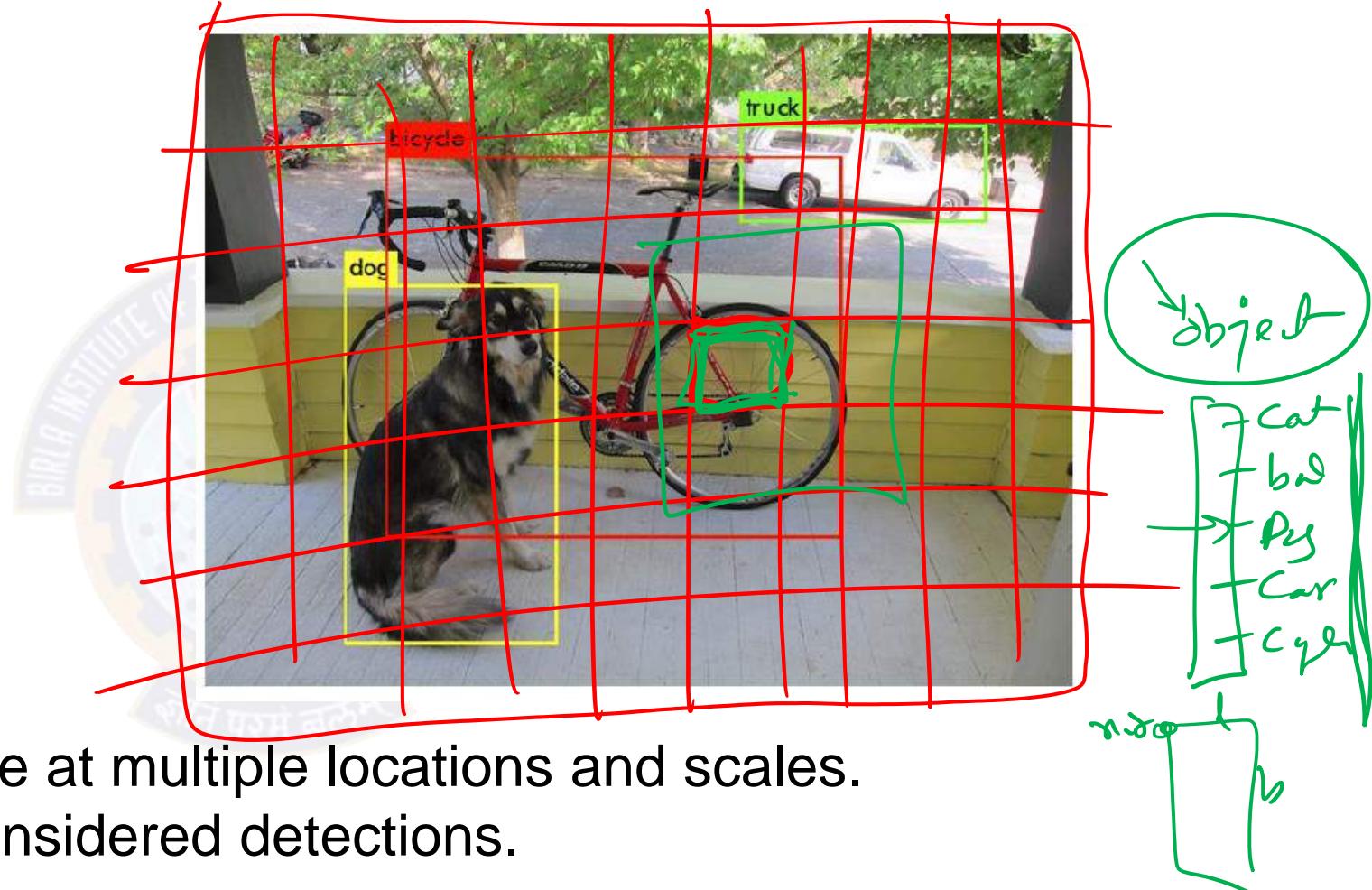
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CNN Case Studies (Yolo)

**Dr. Kamlesh Tiwari**

# Object Detection with Yolo

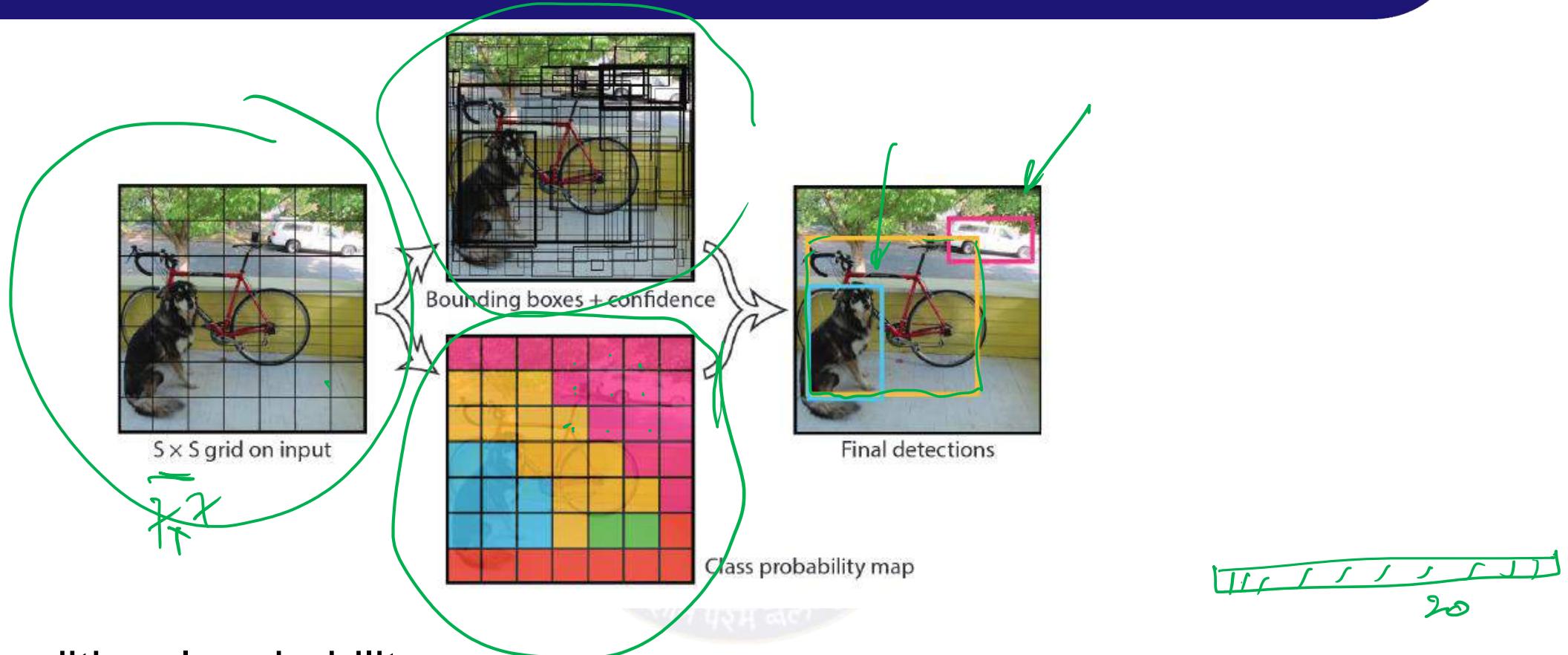
- What is there and where?
- Deformative Part Model, and F-RCNN



- Apply the model to an image at multiple locations and scales.
- High scoring regions are considered detections.

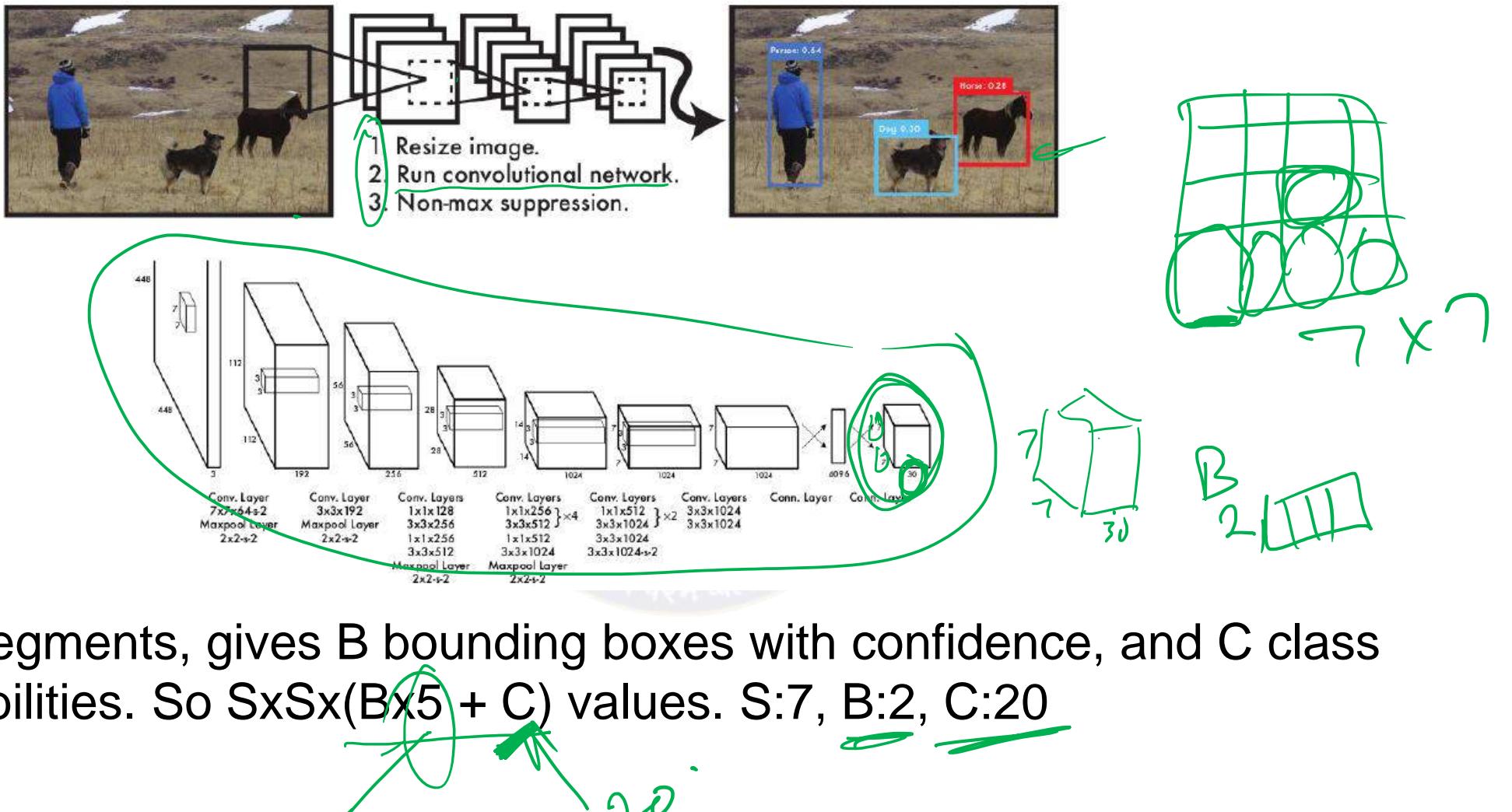
**YOLO:** apply a single neural network to the full image that divides it into regions and predicts bounding boxes and probabilities for each region

# Yolo



- Conditional probability map
  - See <https://pjreddie.com/darknet/yolo/>

# Yolo



- SxS segments, gives B bounding boxes with confidence, and C class probabilities. So  $S \times S \times (B \times 5 + C)$  values. S:7, B:2, C:20

# Yolo

Pascal 2007 mAP		Speed
DPM v5	33.7	.07 FPS 14 s/img
R-CNN	66.0	.05 FPS 20 s/img
Fast R-CNN	70.0	.5 FPS 2 s/img
Faster R-CNN	73.2	7 FPS 140 ms/img
YOLO	63.4	45 FPS 22 ms/img

- It is fast
- Speed comes at the price of accuracy. Improved to 69%
- Generalizes well
- Latest version YOLOv3 2018



# Thank You!

Module-5 Autoencoders:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

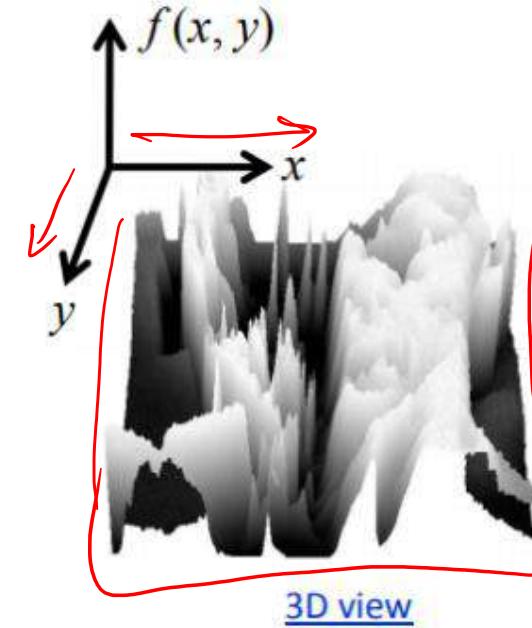
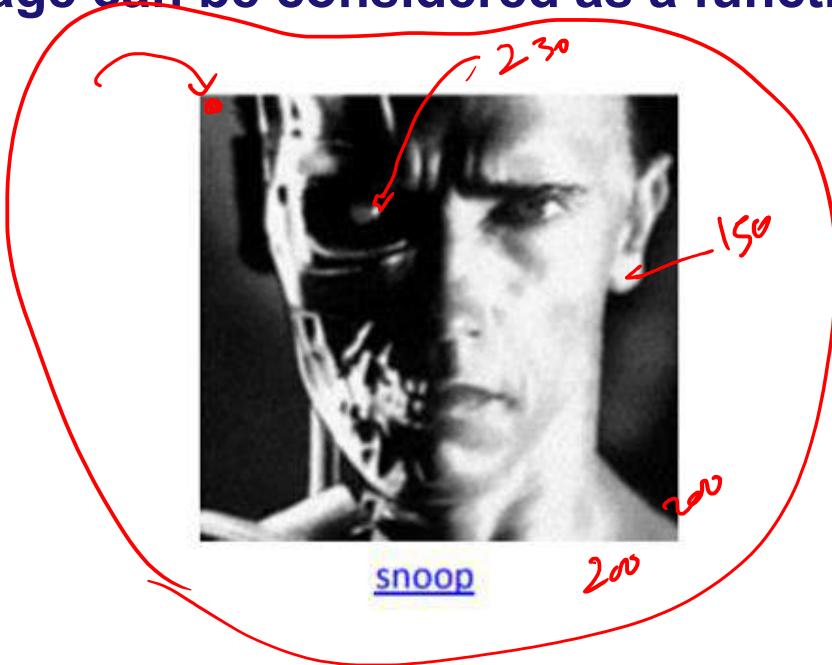
# CNN Prerequisites: Computer Vision

**Dr. Kamlesh Tiwari**

# Image

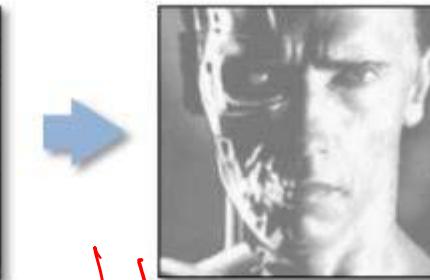
Grayscale image can be considered as a function  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$

0-255



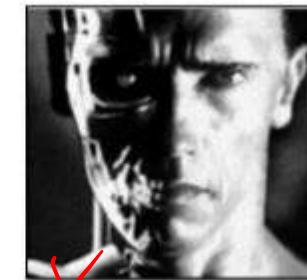
- Being digital adds quantization and sampling. We may apply operators on this function

X



f

$$g(x, y) = f(x, y) + 20$$



S

$$f(\mathbb{F}^{-1}, y)$$

# Computer Vision - history

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
PROJECT MAC

Artificial Intelligence Group  
Vision Memo. No. 100.

July 7, 1966

THE SUMMER VISION PROJECT  
Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

## Some Tasks

- Classification
- Annotation
- Detection
- Segmentation

How the Afghan Girl was Identified by her Iris Patterns 1984-2002



# Classification

airplane

automobile

bird

cat

deer

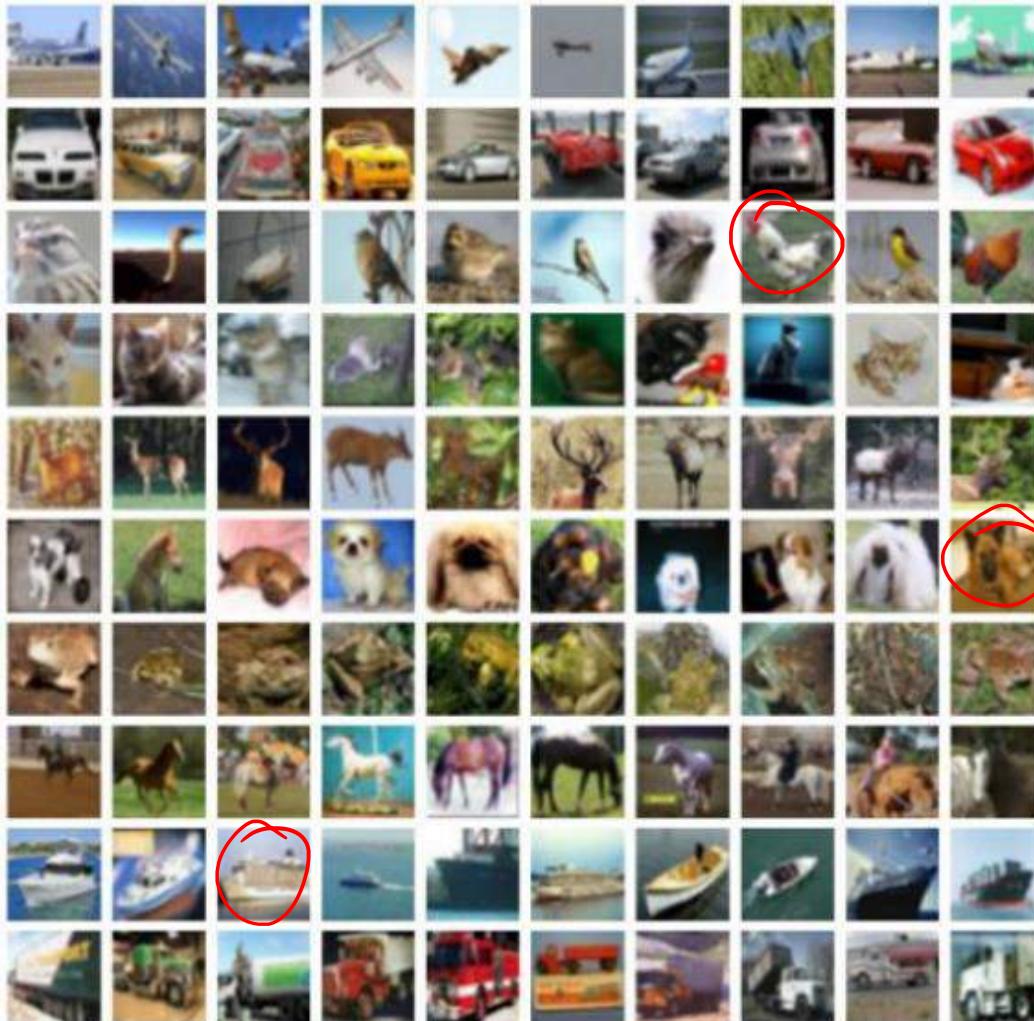
dog

frog

horse

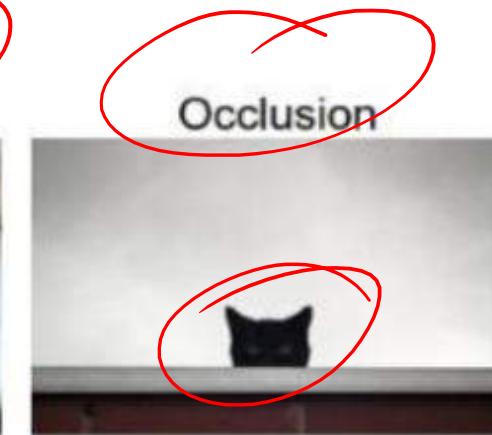
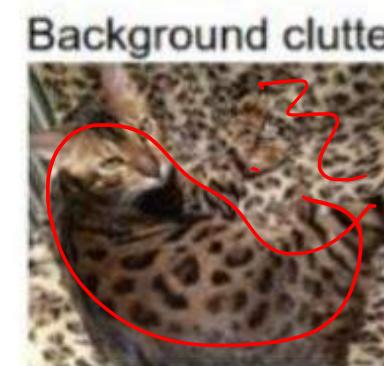
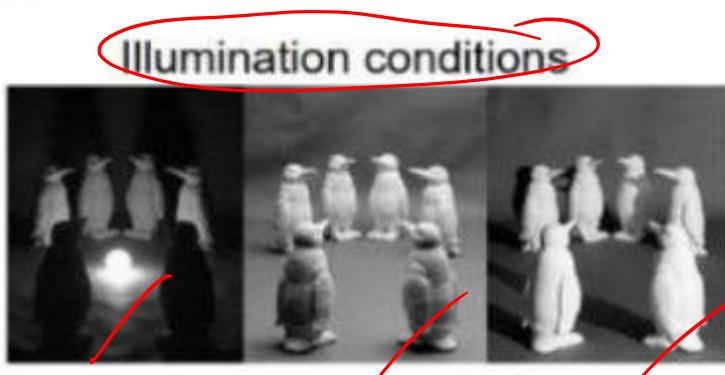
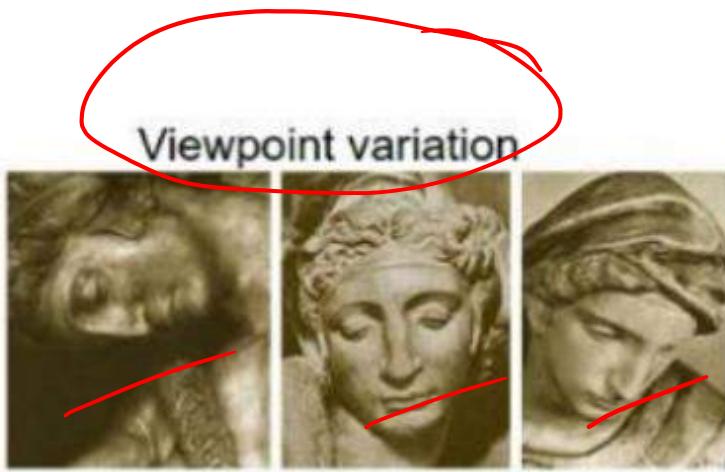
ship

truck



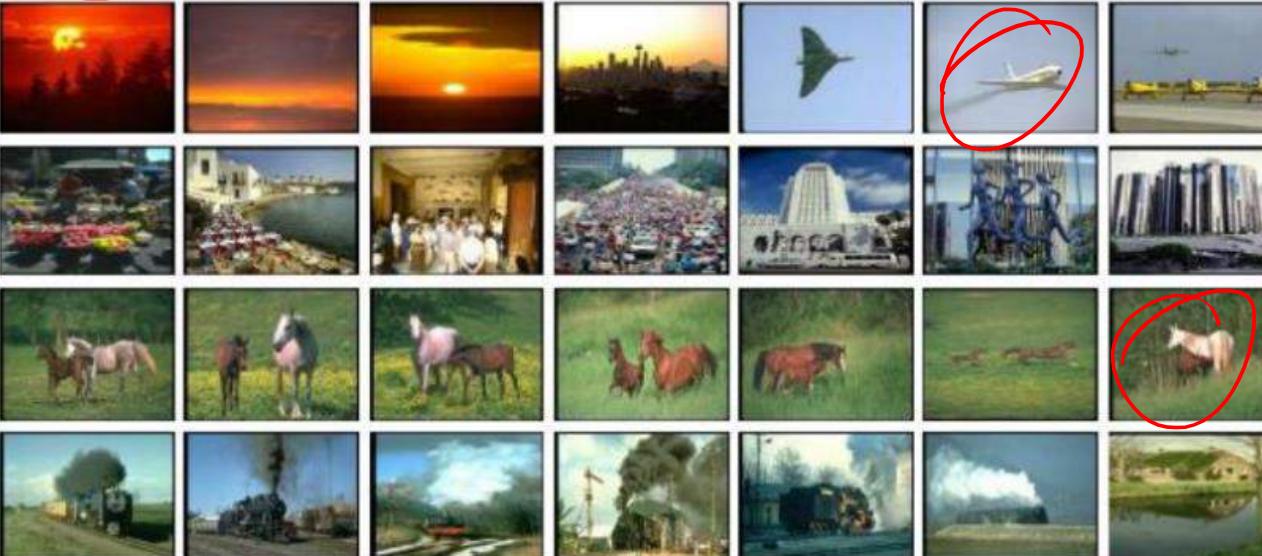
6

# Classification Challenges



# Annotation

					
Predicted keywords	sky, jet, plane, smoke, formation	grass, rocks, sand, valley, canyon	sun, water, sea, waves, birds	water, tree, grass, deer, white-tailed	bear, snow, wood, deer, white-tailed
Human annotation	sky, jet, plane, smoke	rocks, sand, valley, canyon	sun, water, clouds, birds	tree, forest, deer, white-tailed	tree, snow, wood, fox

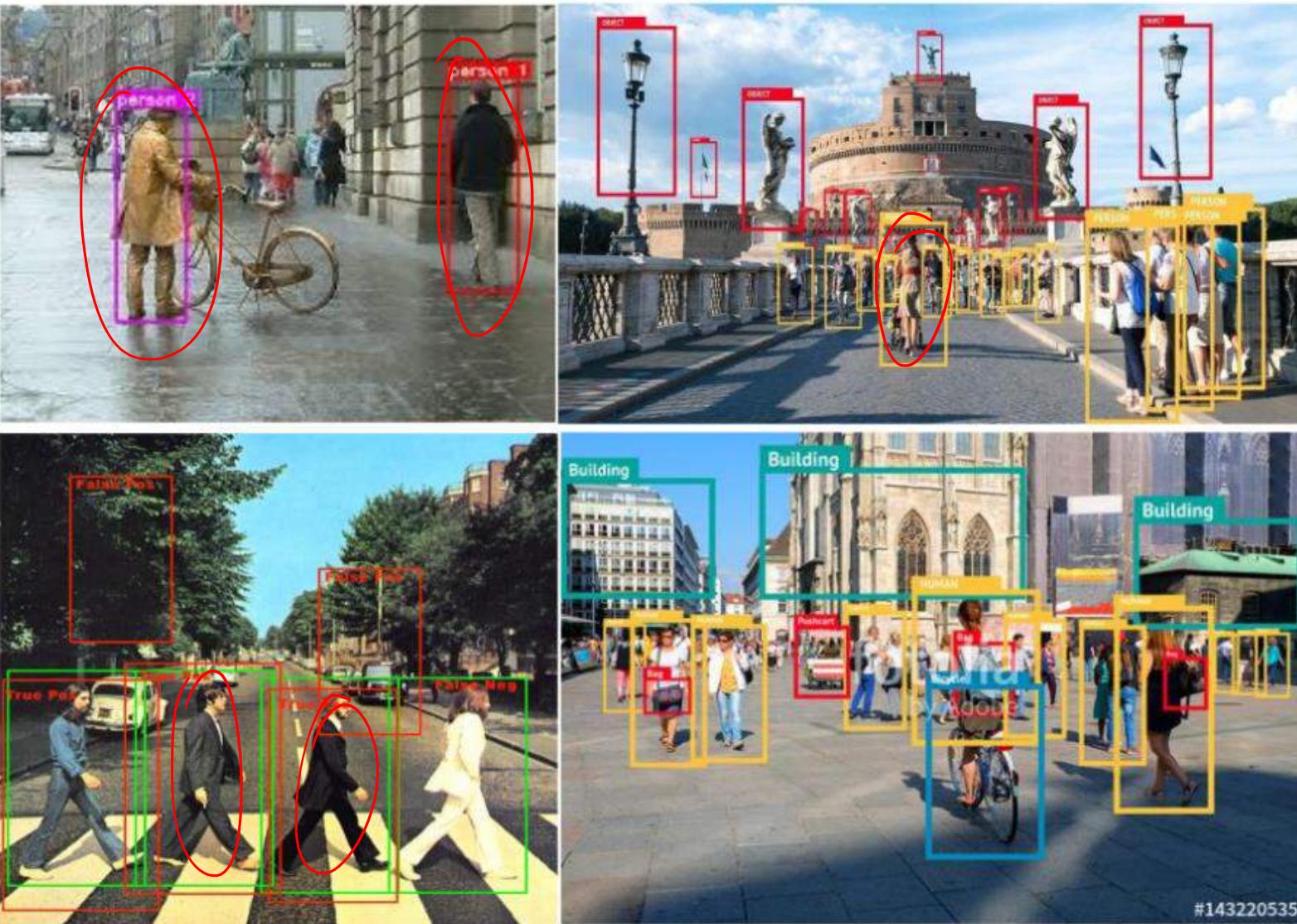
The grid displays four rows of images. Row 1 shows various sunsets and skies. Row 2 shows street scenes with people and vehicles. Row 3 shows horses (mares) in different settings. Row 4 shows trains and steam locomotives. Red circles highlight specific predictions in the first three rows that differ from the human annotations.

Prediction for queries: Sky (Row1), Street (Row2), Mare (Row3) and Train (Row4)

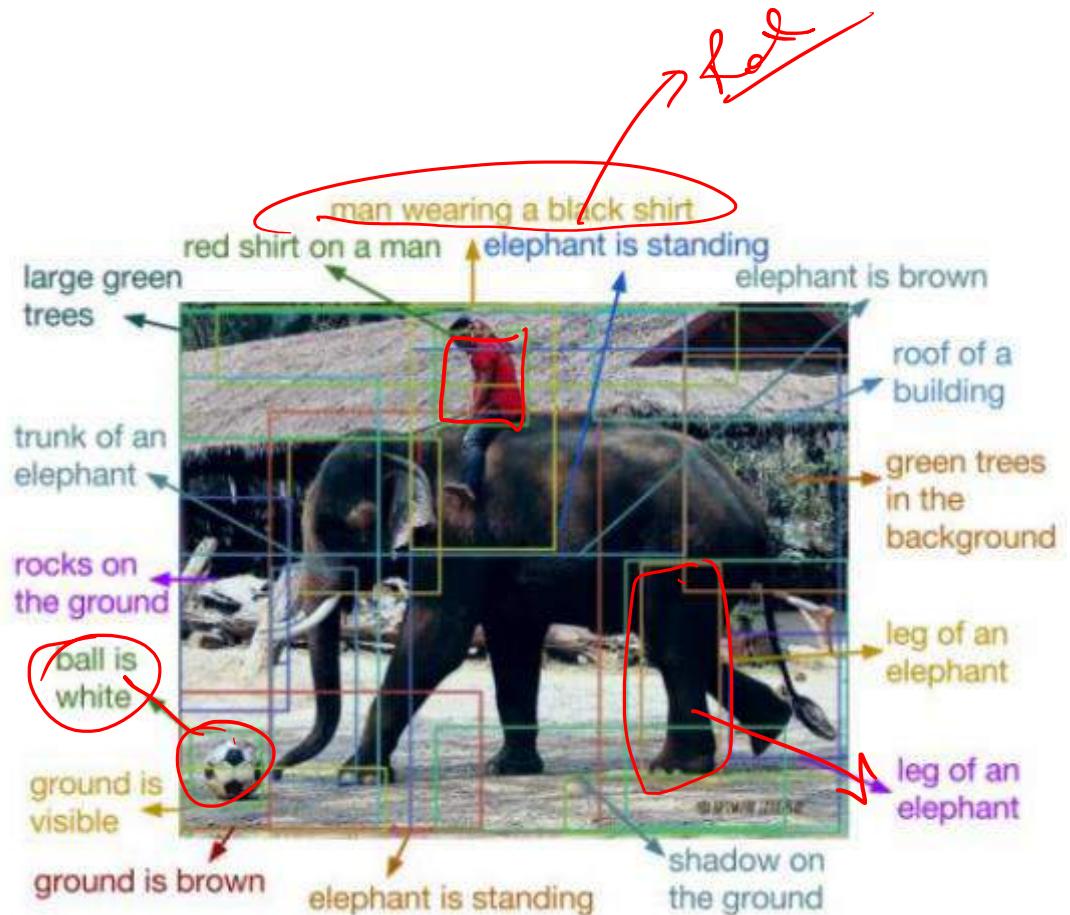
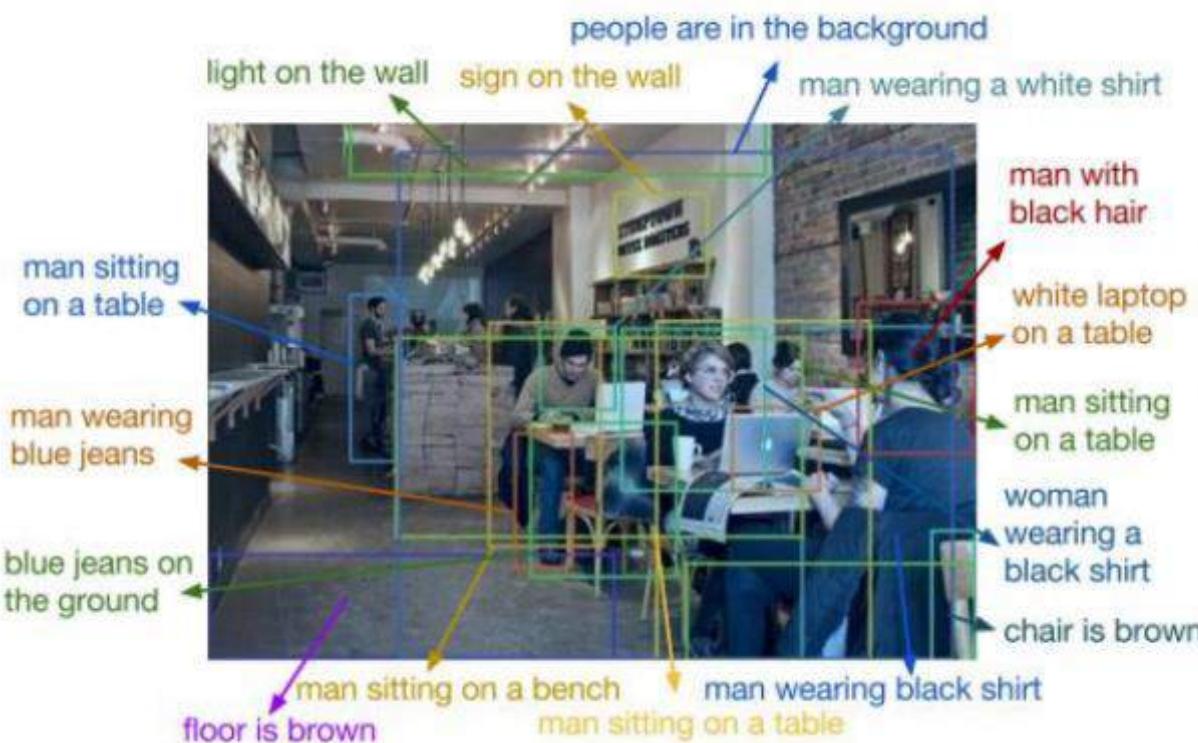
Ref: A. Makadia, V. Pavlovic, and S. Kumar.

A New Baseline for Image Annotation. In Proceedings of ECCV 08.

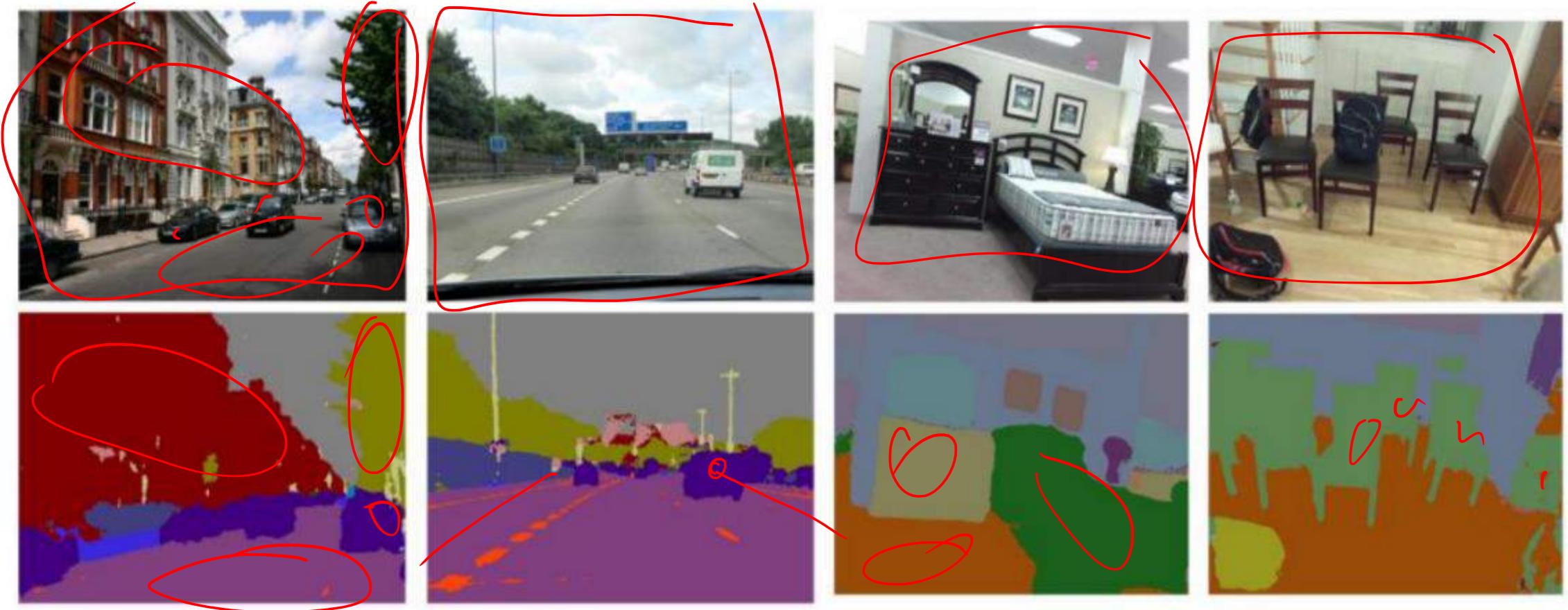
# Detection



# Description

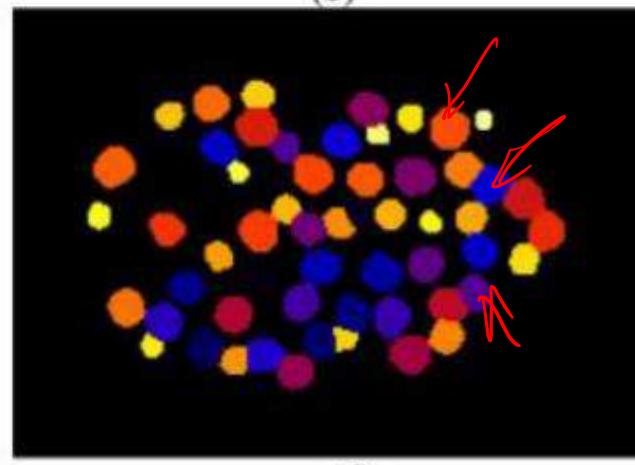
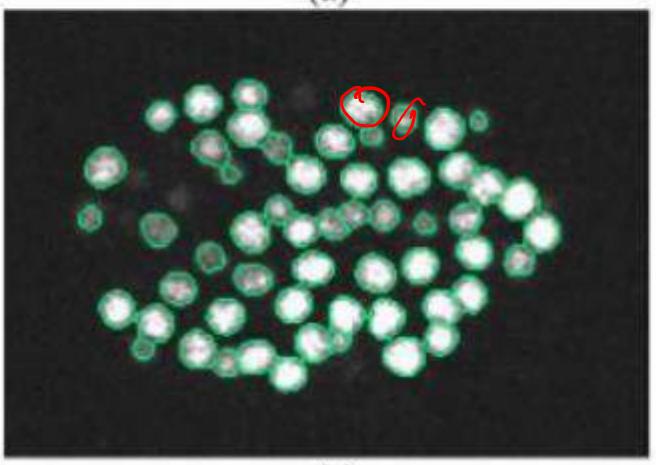
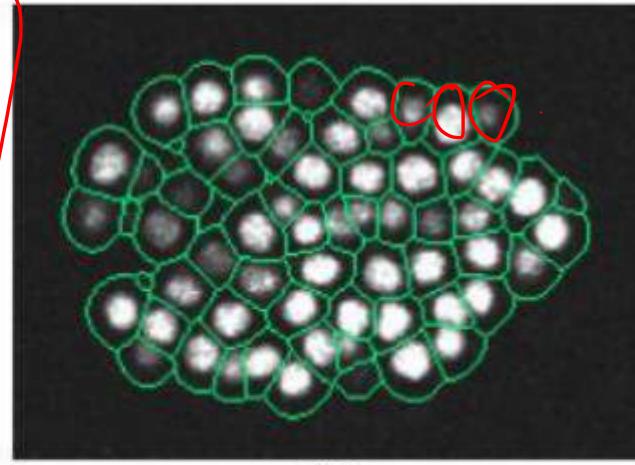
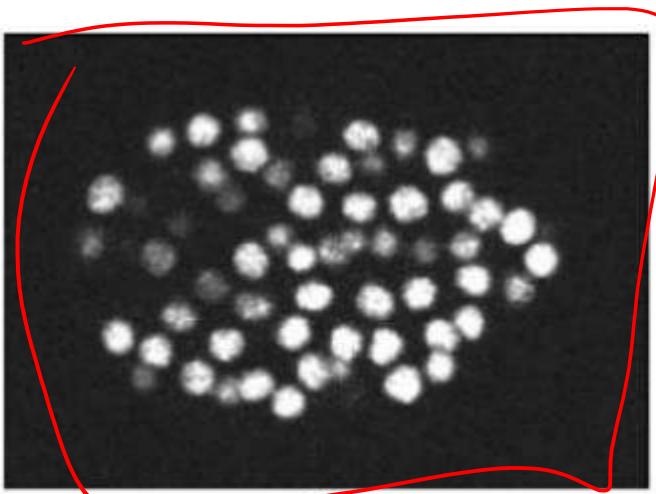


# Segmentation



Vijay Badrinarayanan, Alex Kendall and Roberto Cipolla "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." PAMI, 2017.

# Segmentation





# Thank You!

In our next session: Image Operations



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CNN Prerequisites: Computer Vision

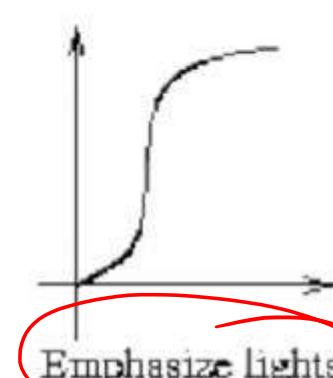
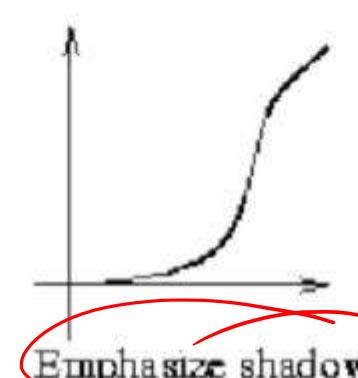
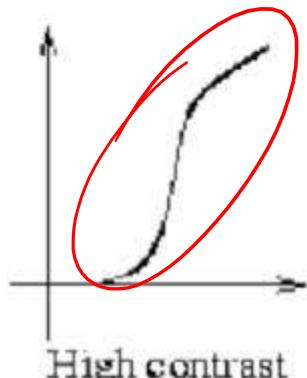
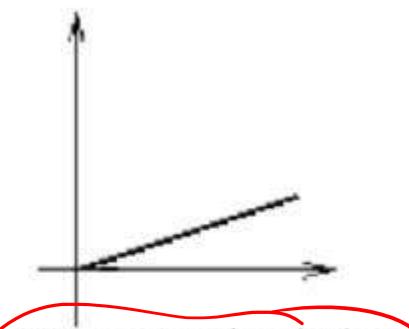
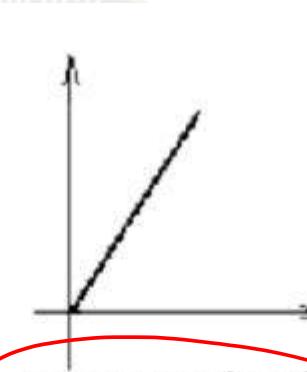
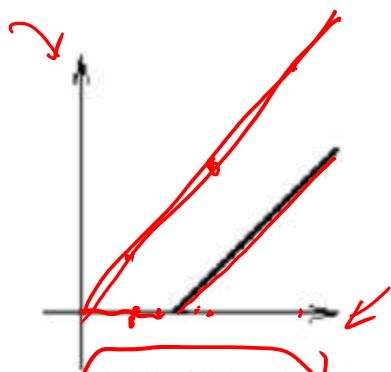
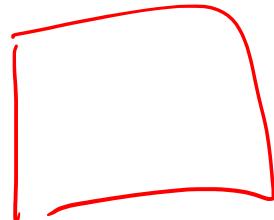
**Dr. Kamlesh Tiwari**

# Point Operations on Images

Point operation is defined as

$$s = M(r)$$

where r is source pixel intensity and s is destination pixel intensity

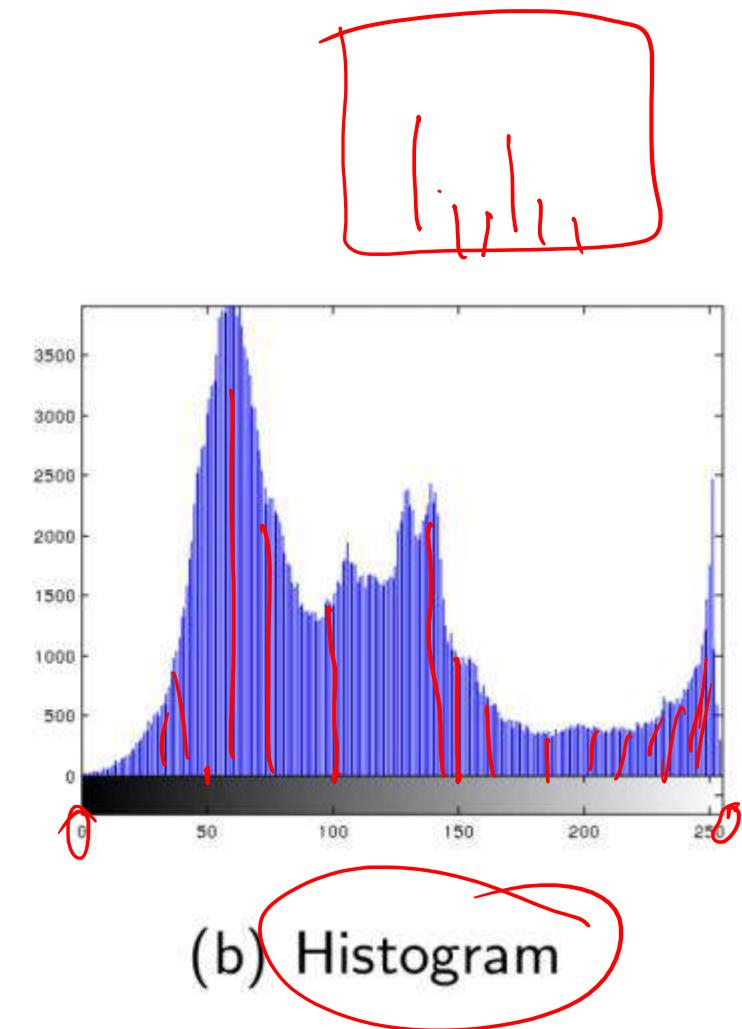


# Histogram

- Discrete probability distribution of the image intensity values



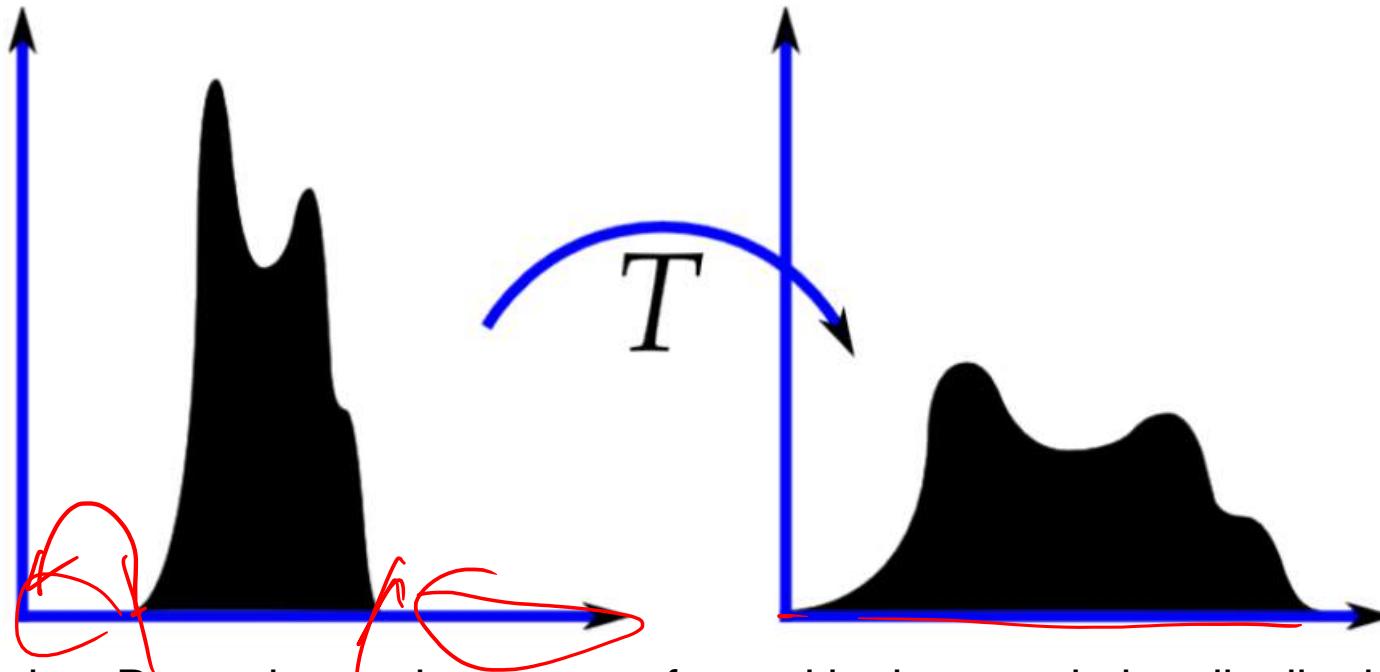
(a) Image



(b) Histogram

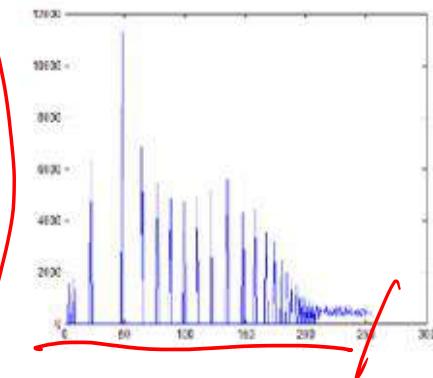
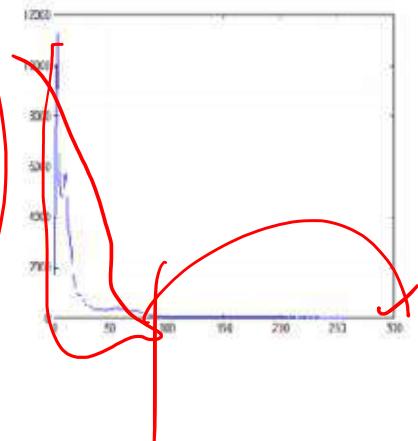
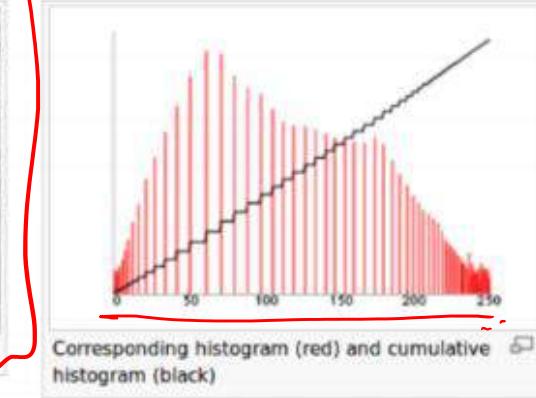
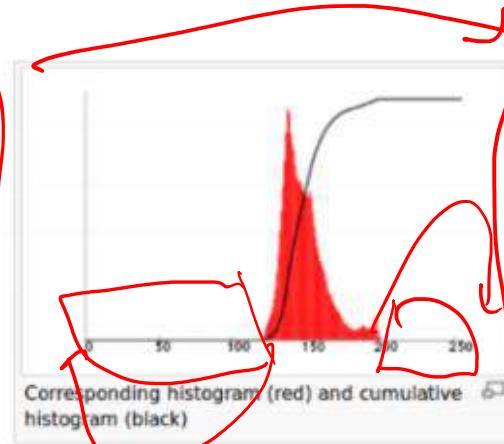
# Histogram Equalization

One method to enhance images is to equalize the histogram



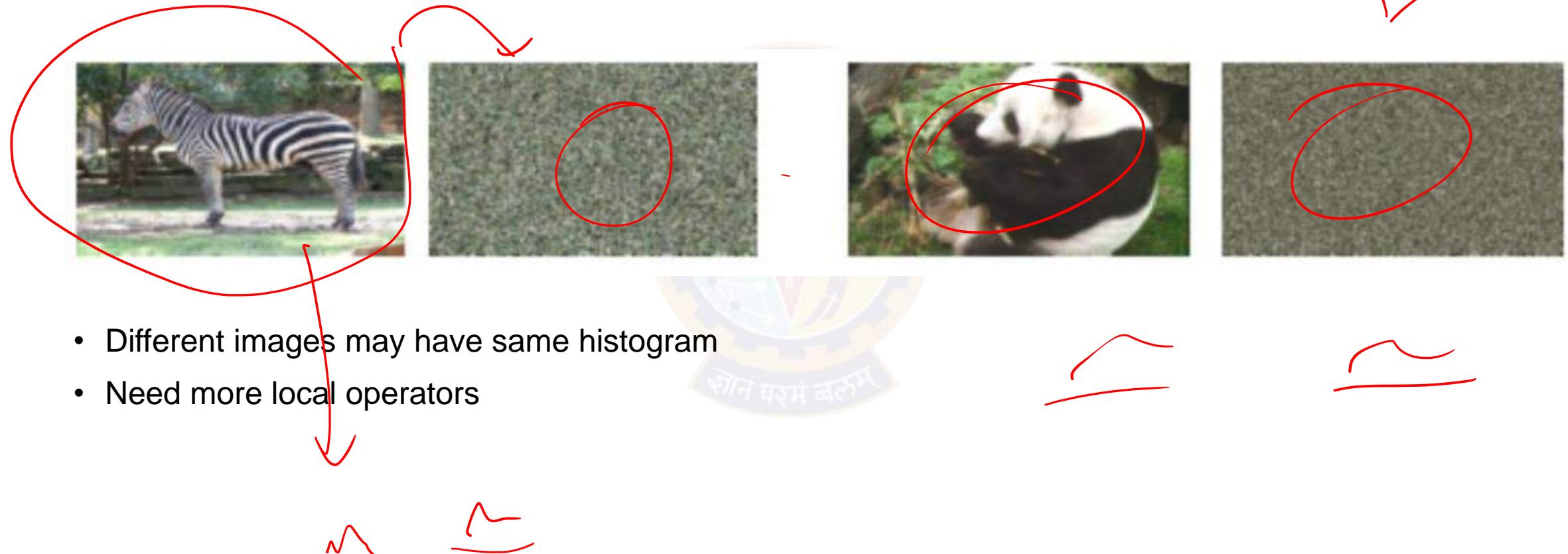
Exercise: Prove that an image transformed by its cumulative distribution function results in an image with uniform histogram. More generally, histogram can be modified by histogram specification

# Histogram Equalization Examples



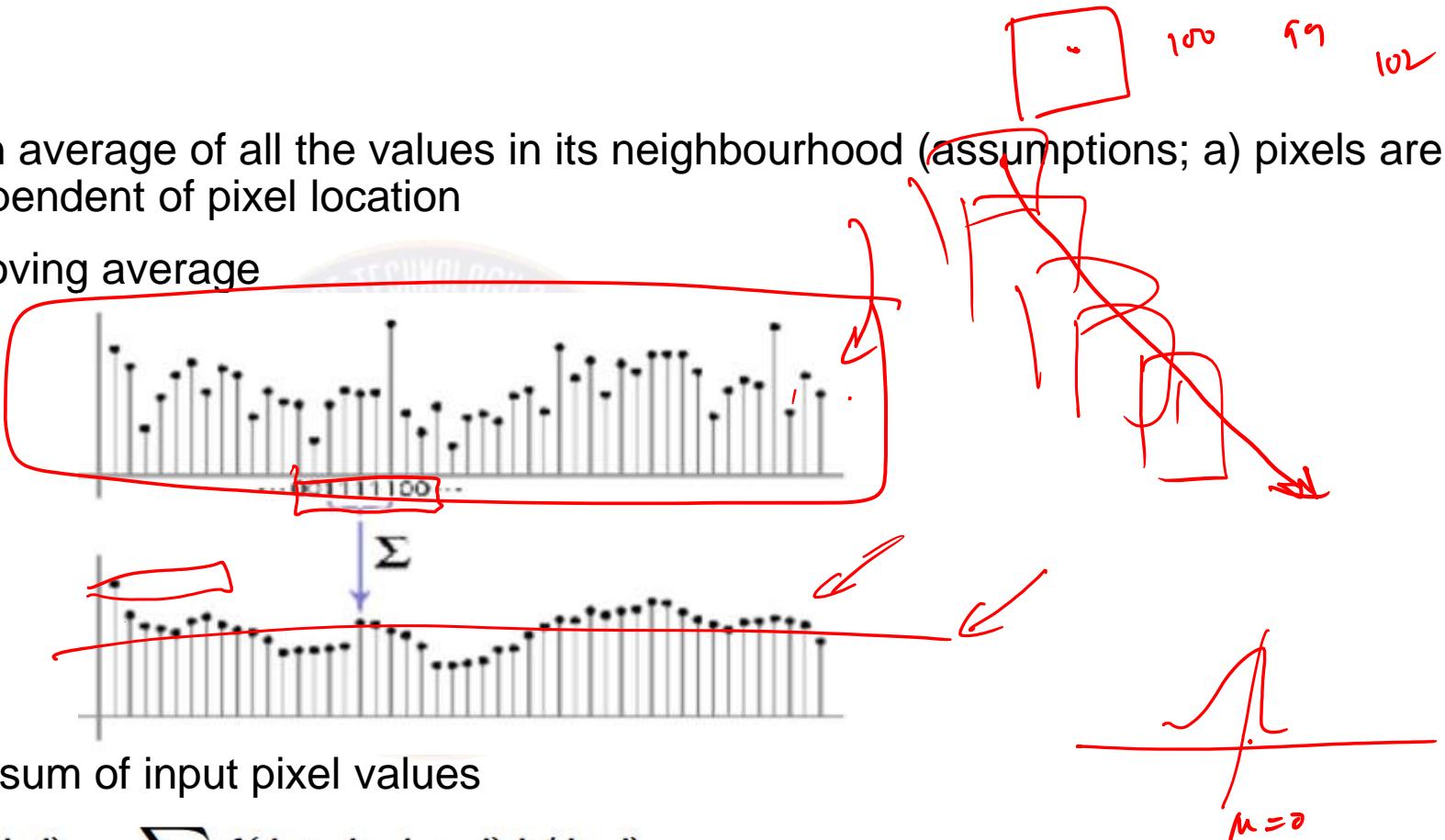
# Shuffling the pixels

- What happens if we shuffle all the pixels in an image randomly?



# First attempt at a solution

- Replace each pixel with an average of all the values in its neighbourhood (assumptions; a) pixels are like neighbour 2) noise is independent of pixel location
- Can add weights to our moving average



- Output pixel is a weighted sum of input pixel values

$$g(i, j) = \sum_{k,l} f(i + k, j + l) h(k, l)$$

- Entries in kernel  $h(k, l)$  are called the filter coefficients. Operator is termed correlation operator.  $g = f \otimes h$

# Convolution

$$g = f \otimes h.$$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

0	10	20	30	30	30	20	10
0	20	40	60	60	60	40	20
0	30	60	90	90	90	60	30
0	30	50	80	80	90	60	30
0	30	50	80	80	90	60	30
0	20	30	50	50	60	40	20
10	20	30	30	30	30	20	10
10	10	10	0	0	0	0	0

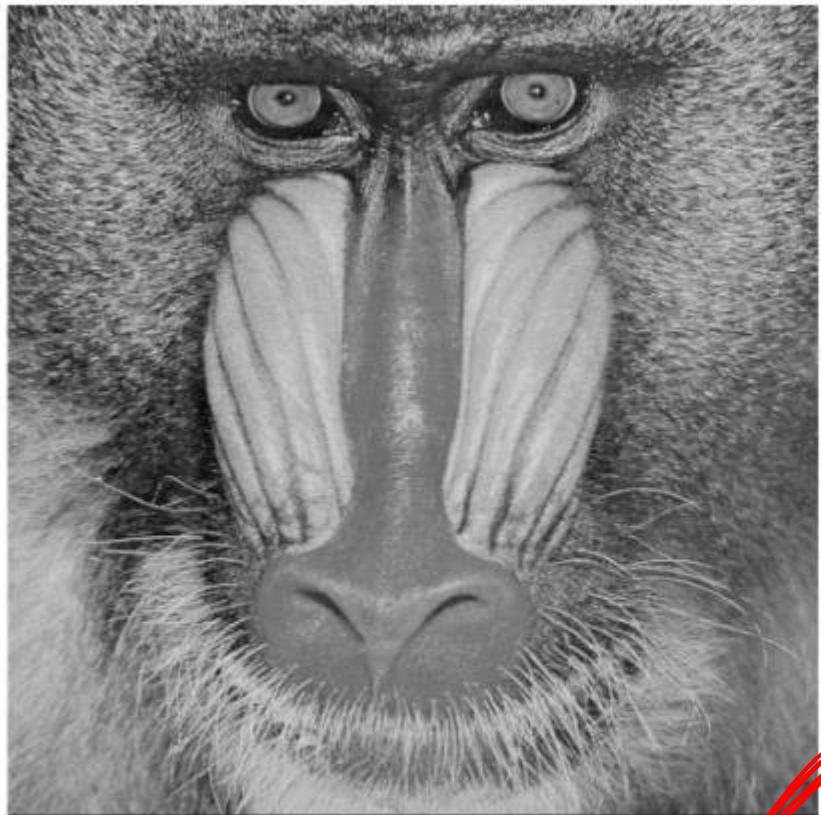


- Convolution is a simple variant of correlation termed as  $g = f * h$

$$g(i, j) = \sum_{k,l} f(i - k, j - l)h(k, l)$$

$$\begin{matrix} 9 & 15 & 12 \\ 6 & 14 & 10 \\ 3 & 12 & 15 \end{matrix} \rightarrow \begin{matrix} 9+15+12 \\ +6+10+10+3+2+15 \\ 9 \end{matrix}$$

# Box Filtering



(c)



(d)

25x25  
is 5x5.

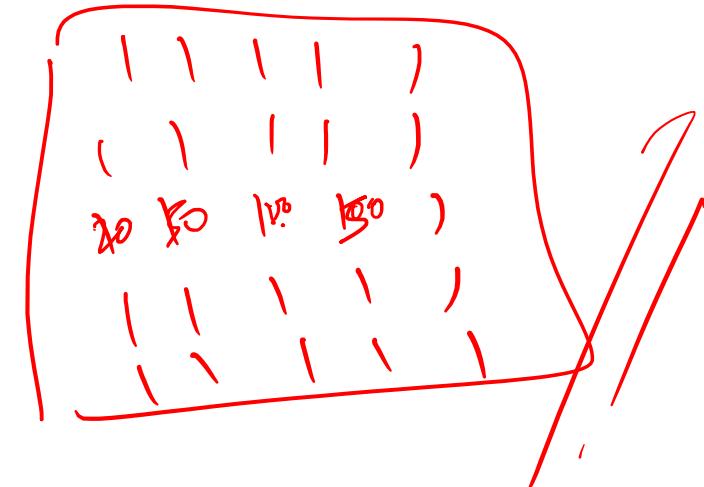
# Gaussian Filtering

Used when we want to have central pixels with more influence.

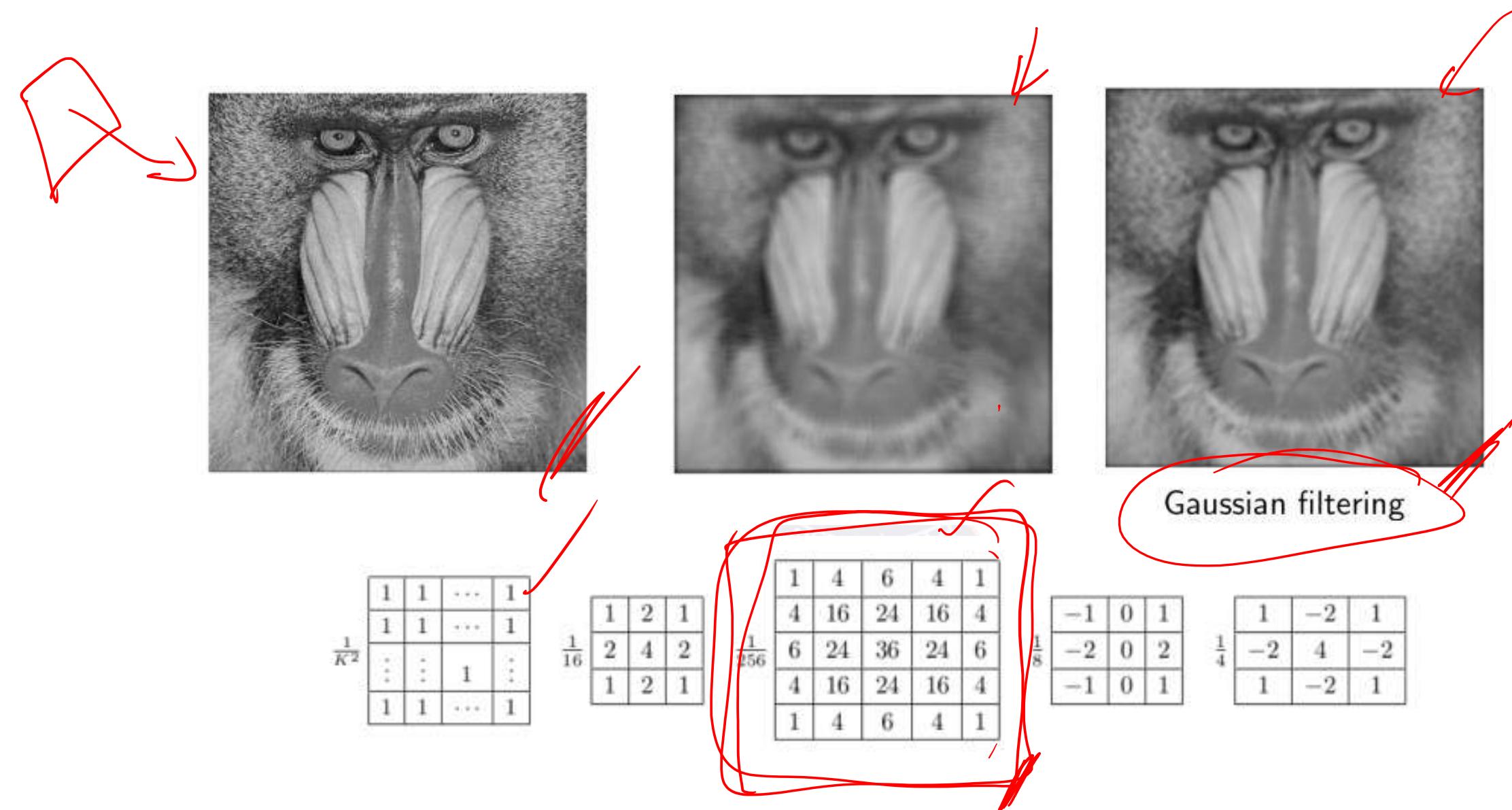
- Gaussian Kernel is defined as

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

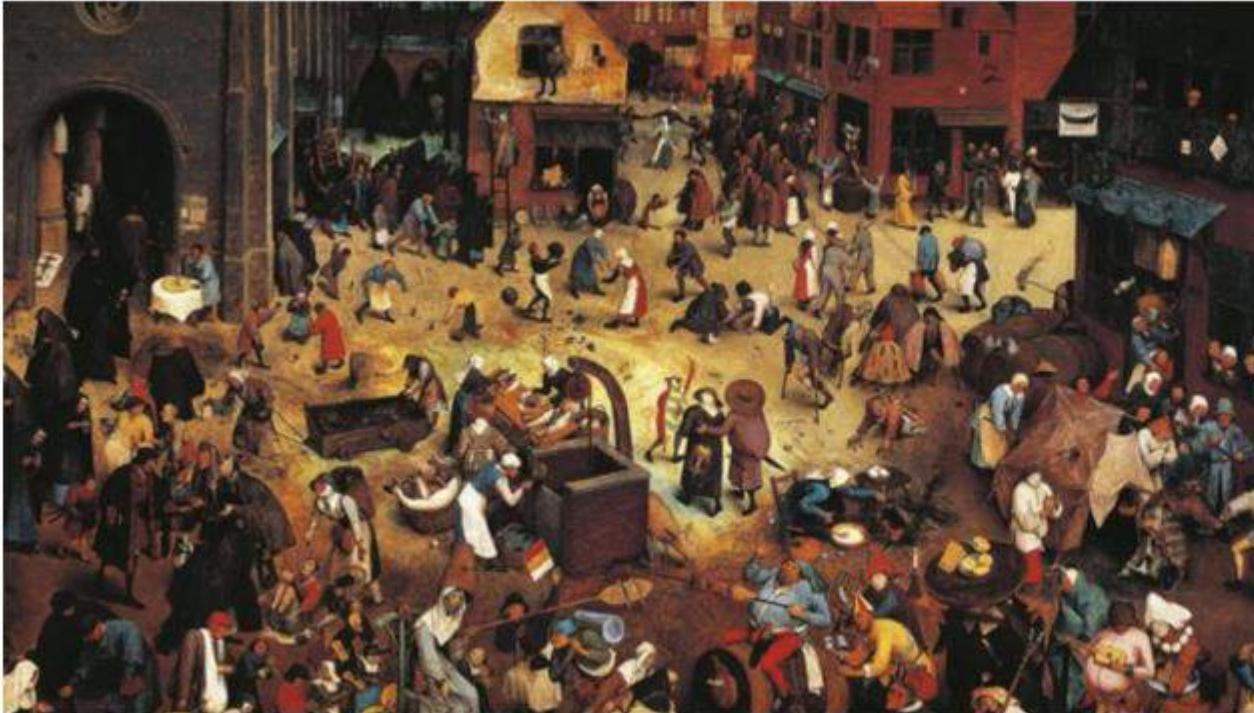
- It is a low pass filter



# Gaussian Filtering

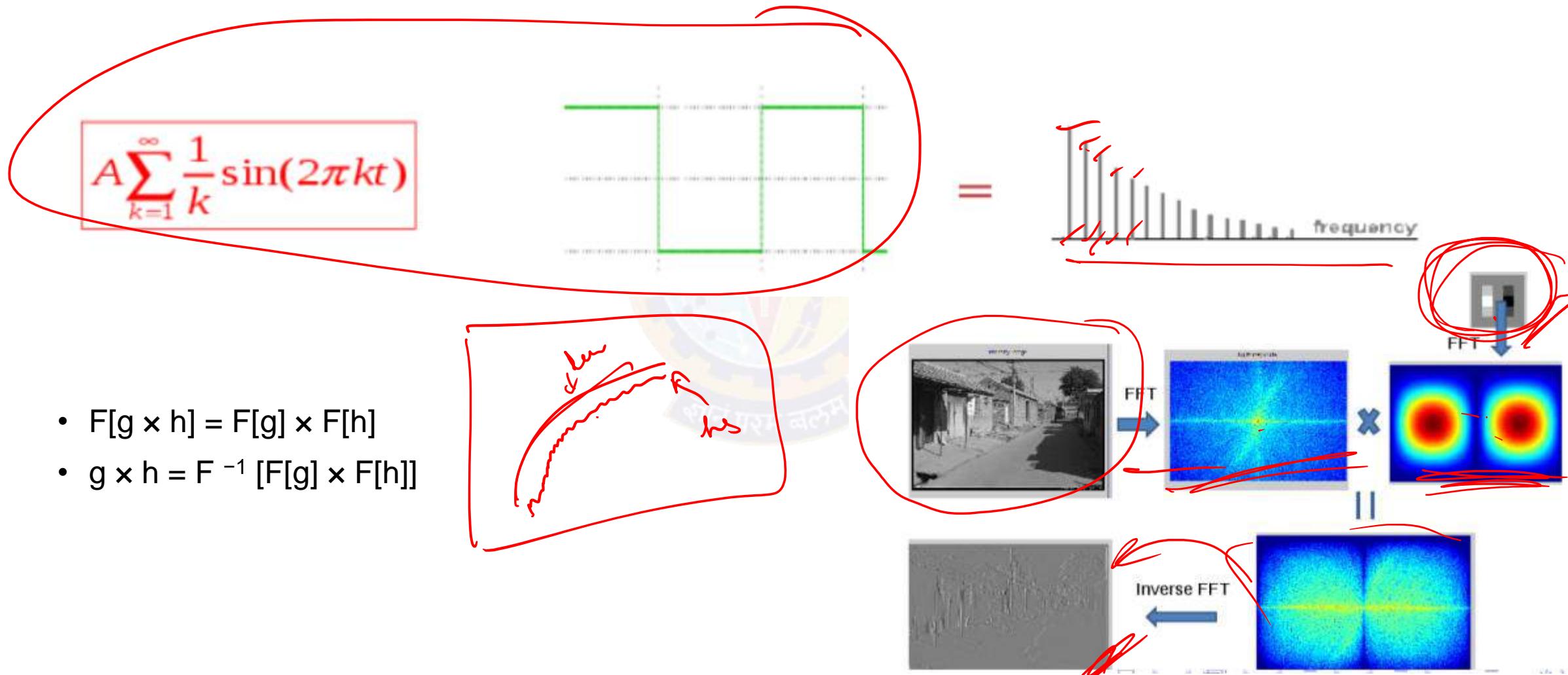


# Convolution



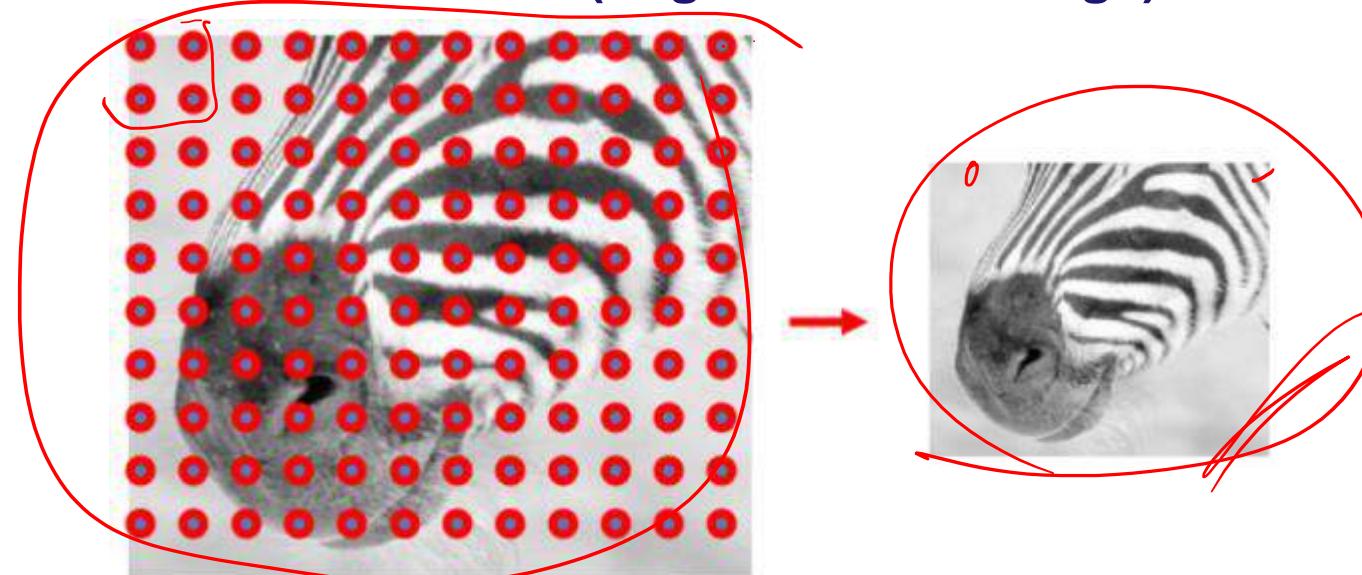
# Fourier Transform

Any signal (function) can be written as sum of various sin and cos waves (terms)

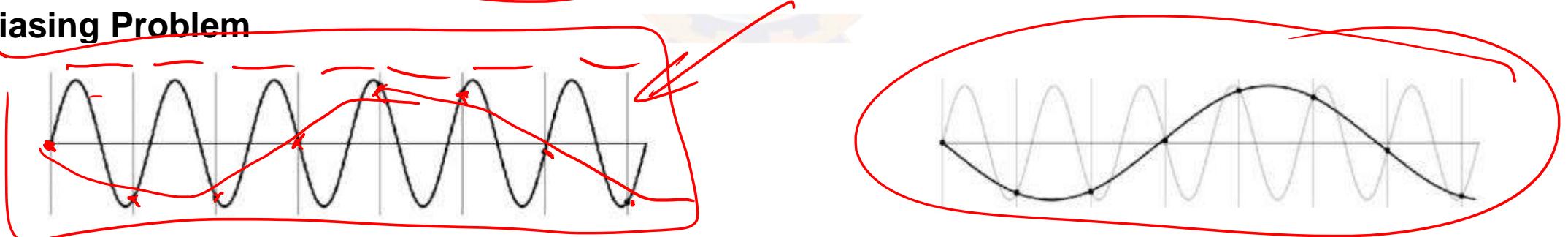


# Sampling

Throw away every other row and column (to get half size image)



Aliasing Problem

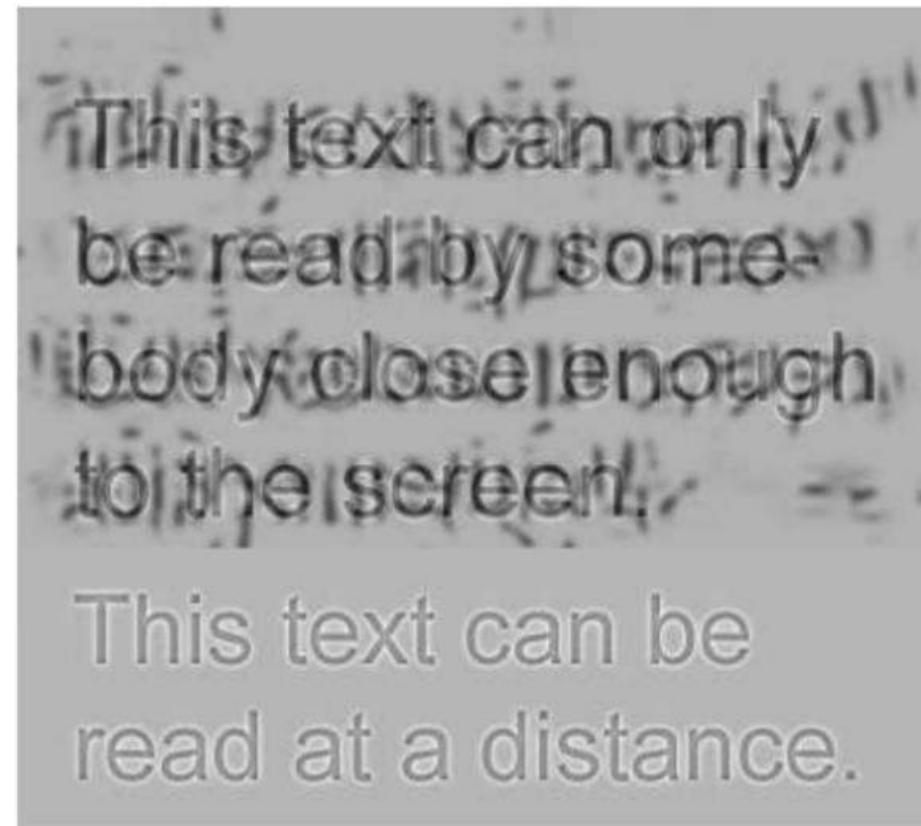


Sub sampling may be dangerous: Funny striped shirt. Wagon wheels rolling in the wrong way in movies.

# Hybrid Images



# Example Hybrid Image



The hybrid font becomes invisible at few meters. The bottom text remains easy to read at relatively long distances.



# Thank You!

In our next session: Filtering



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CNN Prerequisites: Computer Vision

**Dr. Kamlesh Tiwari**

# Linear filtering



Original

$$\text{Original} * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \text{Identical image}$$

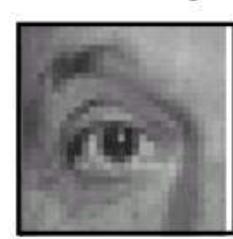


Identical image



Original

$$\text{Original} * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \text{Shifted left By 1 pixel}$$



Shifted left By 1 pixel



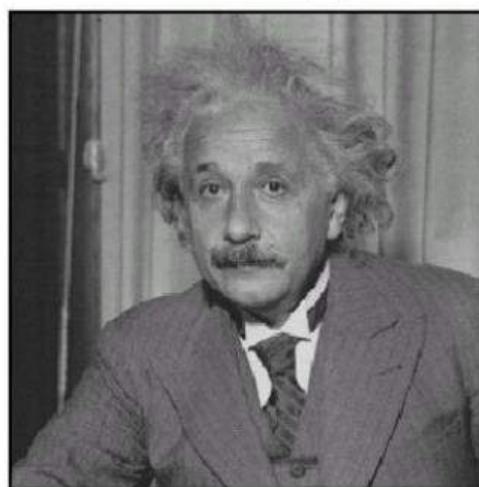
Original

$$\text{Original} * \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \text{Blur (with a mean filter)}$$

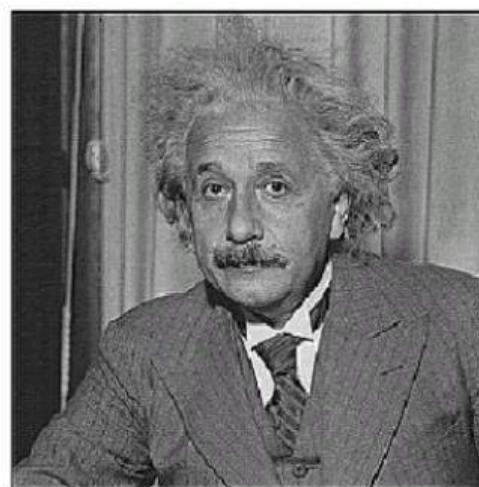
Blur (with a mean filter)

$$\text{Original} * \left( \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) = \text{Sharpening filter (accentuates edges)}$$

Sharpening filter  
(accentuates edges)



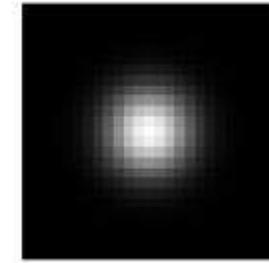
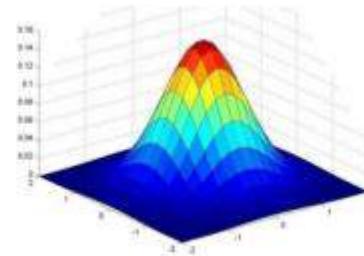
before



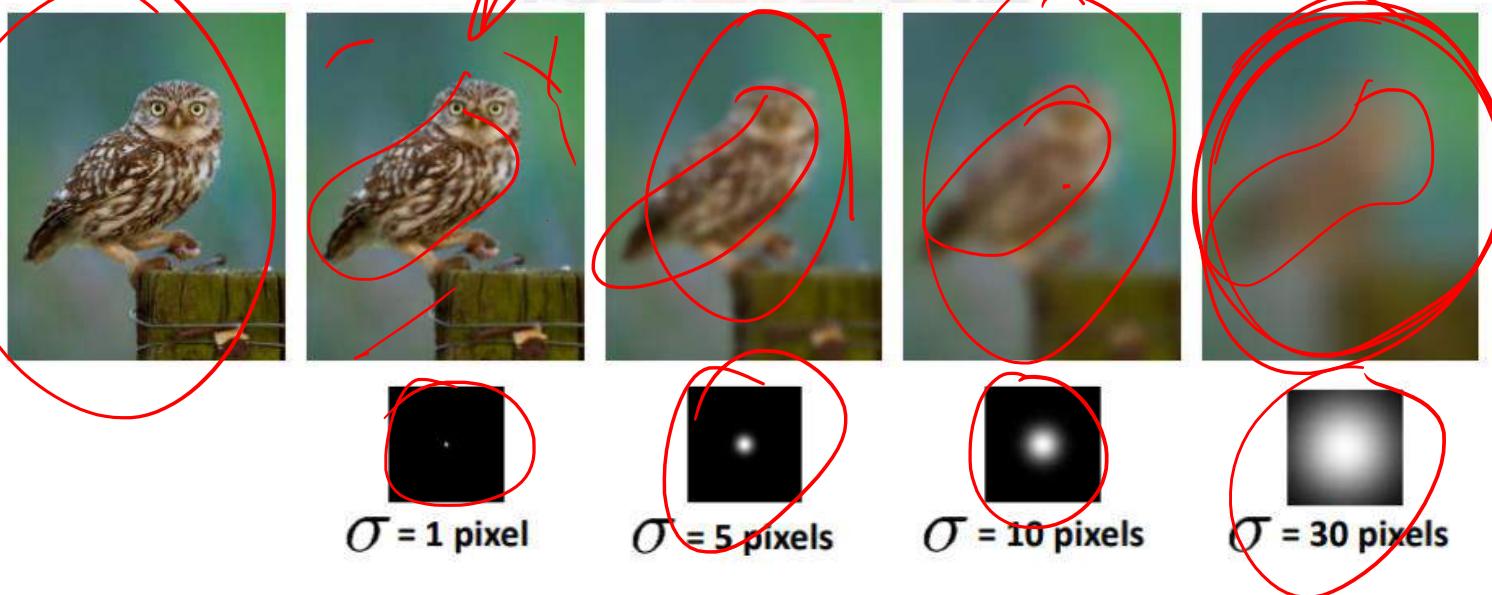
after

# Gaussian filter

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



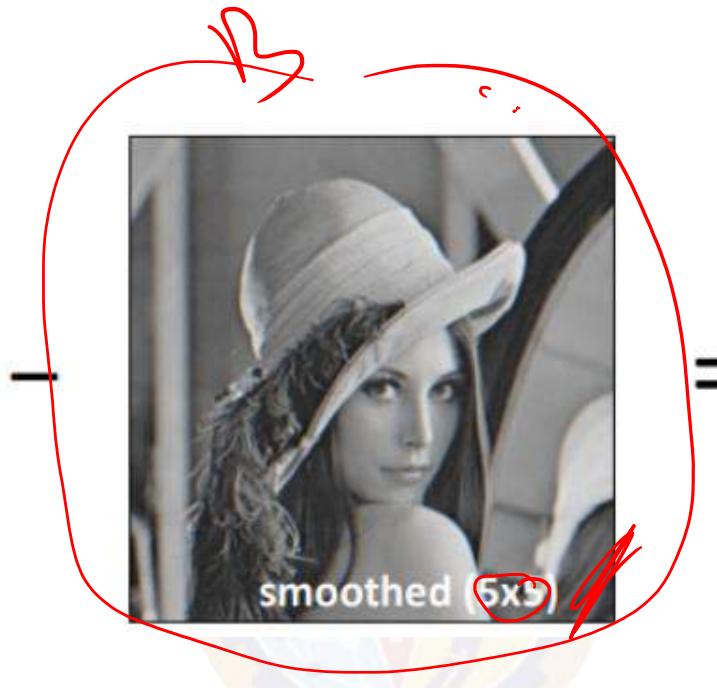
Act as low-pass filter as it removes high-frequency components. Convolution with self is another Gaussian.



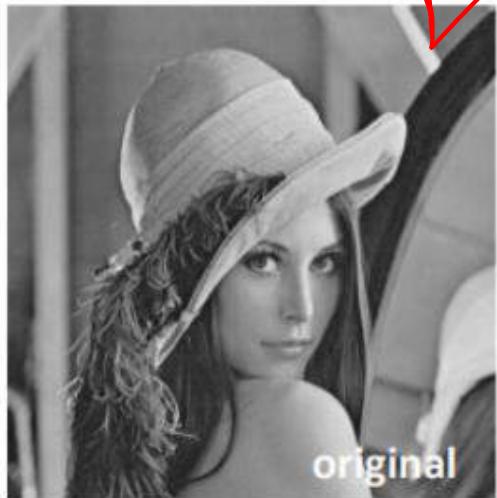
Convolving two times with Gaussian kernel of  $\sigma$  width is equivalent to Convoluting once with Gaussian kernel of  $\sqrt{2}\sigma$  width

# Sharpening

What does blurring take away?



Let's add it back:



# Edge detection

Common approximation of derivative of Gaussian

$$\frac{1}{8} \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

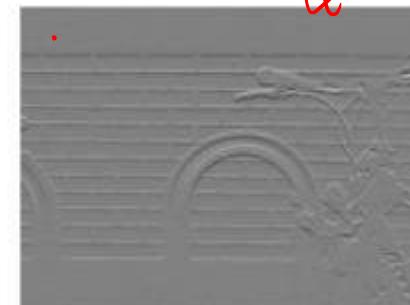
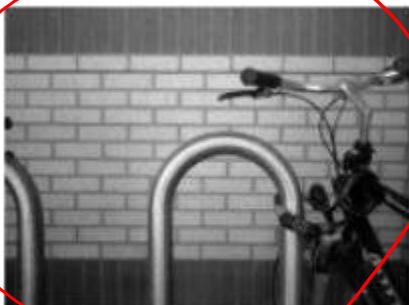
$s_x$

$$\frac{1}{8} \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

$s_y$

Edge magnitude is  $\sqrt{s_x^2 + s_y^2}$  and the direction is  $\tan^{-1}(s_y/s_x)$

Edge is detected by using threshold





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) 1-D

**Dr. Kamlesh Tiwari**

# Motivation to Convolution

Suppose you have a noisy sensor to have a measurement

How could you estimate the actual reading

1. Average of some readings
2. in local neighbourhood
3. What about weighted average?



$$s_t = \sum_{i=0}^{\infty} x_{t-i} w_i$$

	$w_{-6}$	$w_{-5}$	$w_{-4}$	$w_{-3}$	$w_{-2}$	$w_{-1}$	$w_0$
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90
S							

1.80					
------	--	--	--	--	--

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

Here things are in one dimension only.



# Thank You!

In our next session:



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) 2-D

**Dr. Kamlesh Tiwari**

# Motivation to Convolution

Suppose you have a noisy sensor to have a measurement

How could you estimate the actual reading

1. Average of some readings
2. in local neighbourhood
3. What about weighted average?



$$s_t = \sum_{i=0}^{\infty} x_{t-i} w_i$$

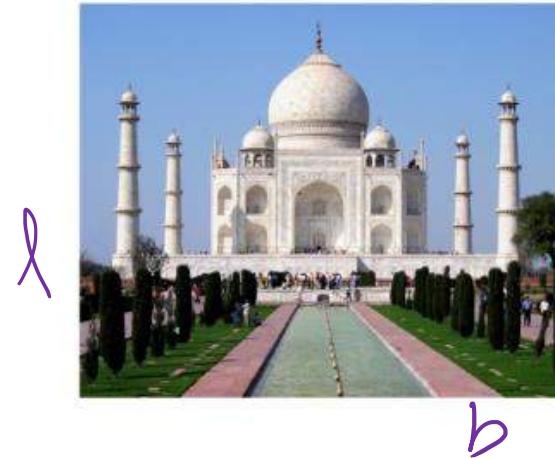
	$w_{-6}$	$w_{-5}$	$w_{-4}$	$w_{-3}$	$w_{-2}$	$w_{-1}$	$w_0$
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90
S							

1.80					
------	--	--	--	--	--

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

Here things are in one dimension only.

# Image is a 2D signal

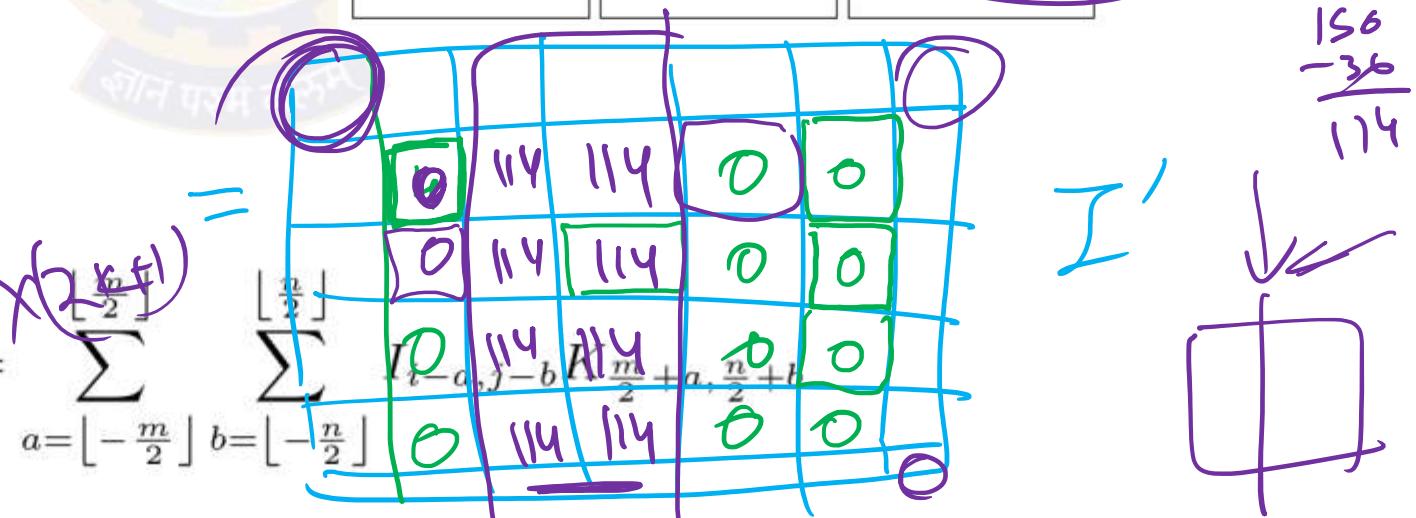
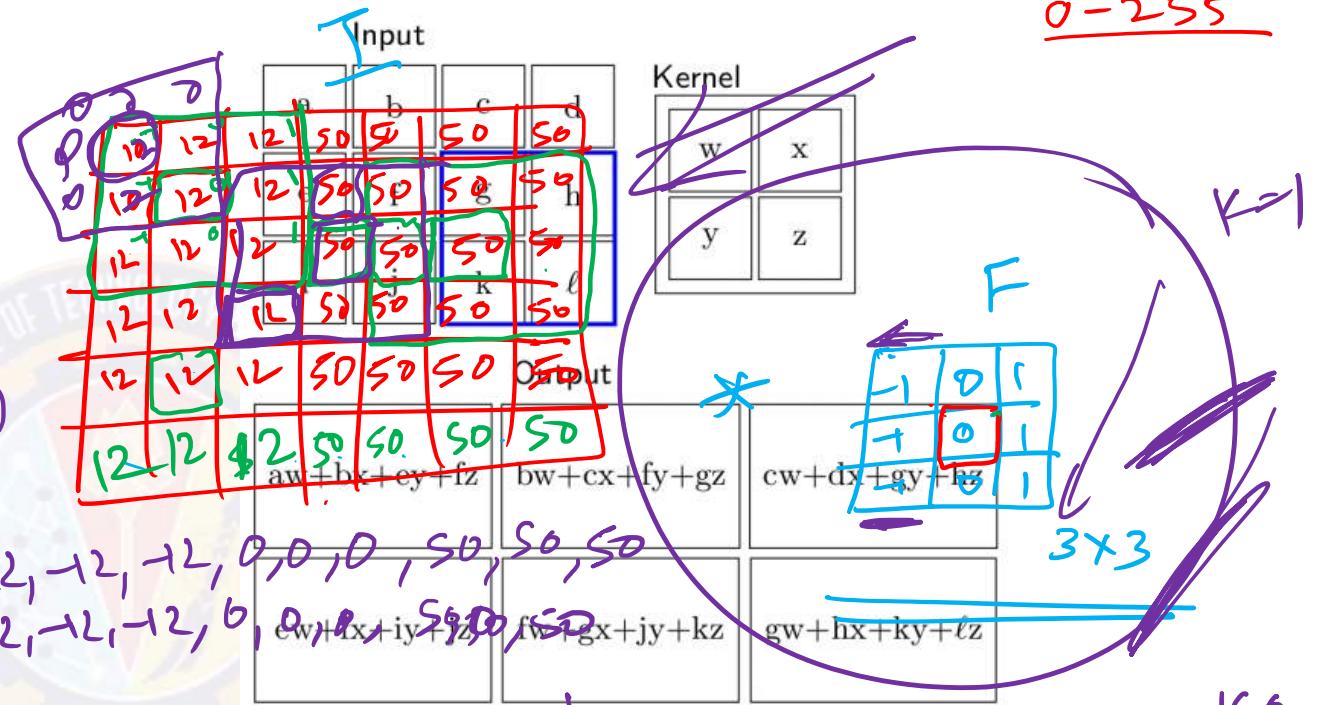


$$\rightarrow (l-2k) \times (b-2k)$$

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a, b}$$

General formula

$$S_{ij} = (I * K)_{ij} = \sum_{a=-\frac{m}{2}}^{\frac{n}{2}} \sum_{b=-\frac{n}{2}}^{\frac{n}{2}} I_{i+a, j+b} K_{a, b}$$





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) 2-D

**Dr. Kamlesh Tiwari**

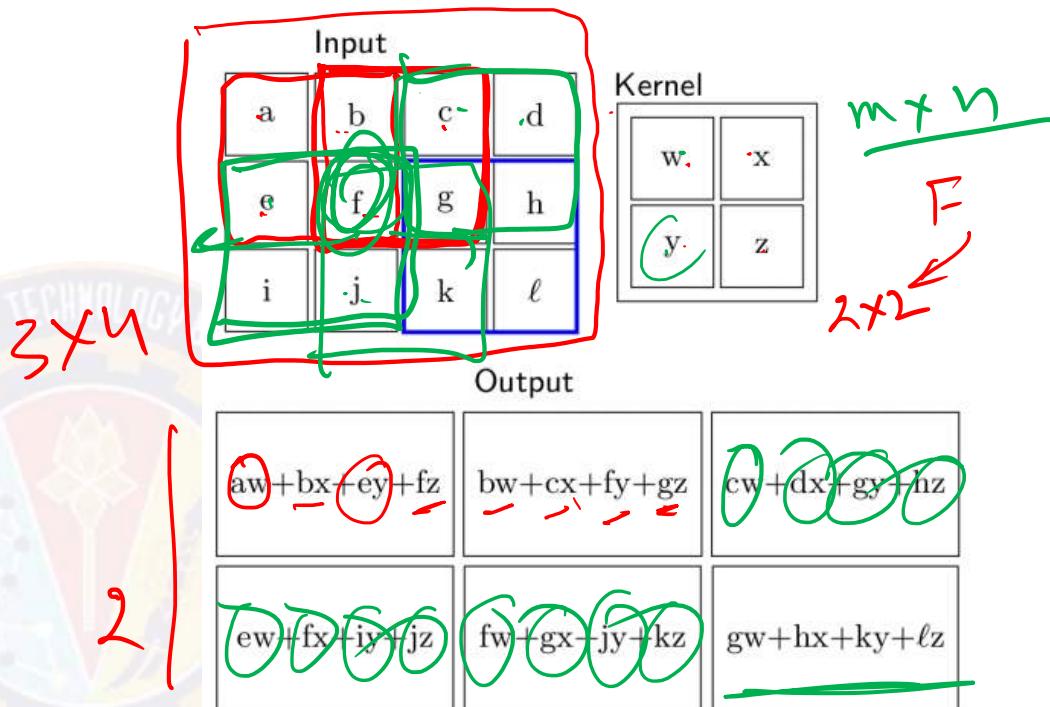
# Image is a 2D signal



$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a, b}$$

General formula

$$S_{ij} = (I * K)_{ij} = \sum_{a=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a, j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$



$$(3-2+1) \times (4-2+1) \\ 2 \times 3$$

$$F = 2K + 1 \\ \lfloor F/2 \rfloor = K$$

# Image is a 2D signal



$$* \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



blurs the image

$$\begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$* \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} =$$



sharpens the image

$$\begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

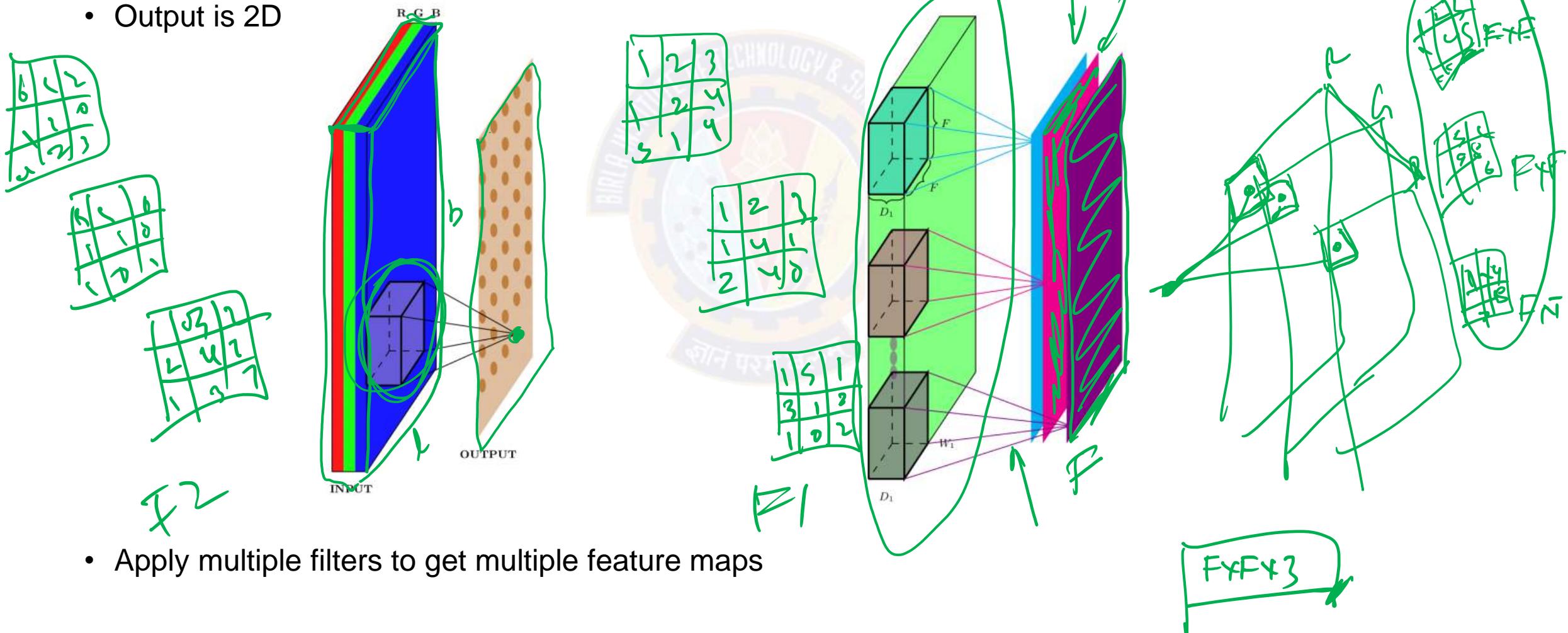
$$* \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & -8 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



detects the edges

# Filter in 3D

- It would refer to volume. Assume the filter extends to the depth
- Output is 2D





# Thank You!

In our next session:



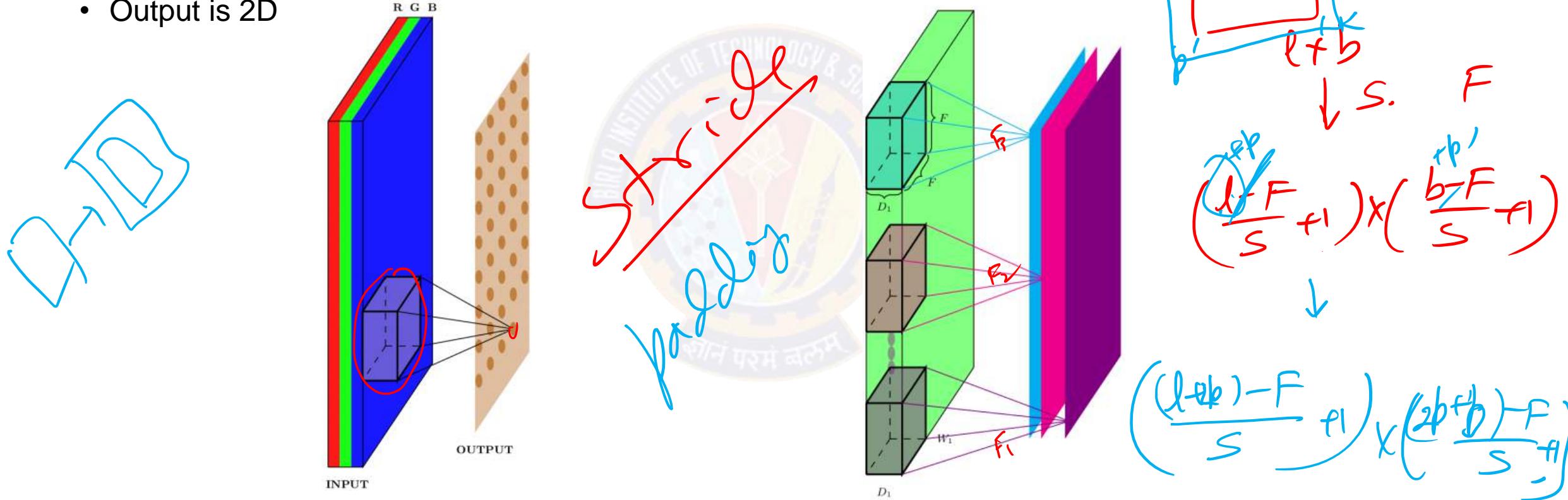
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) 2-D

**Dr. Kamlesh Tiwari**

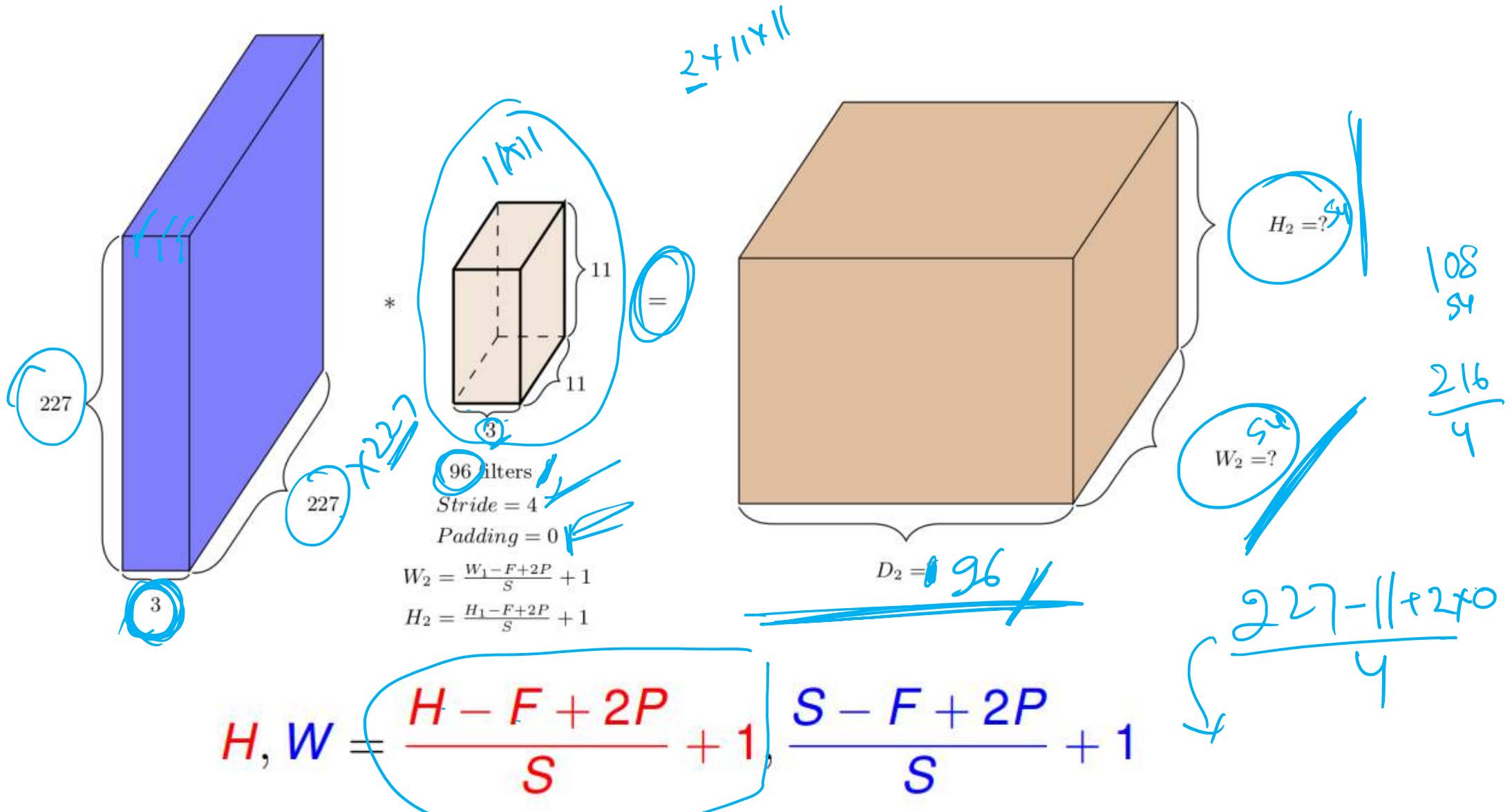
# Filter in 3D

- It would refer to volume. Assume the filter extends to the depth
- Output is 2D



- Apply multiple filters to get multiple feature maps

# Filters, Padding and Stride





**Thank You!**

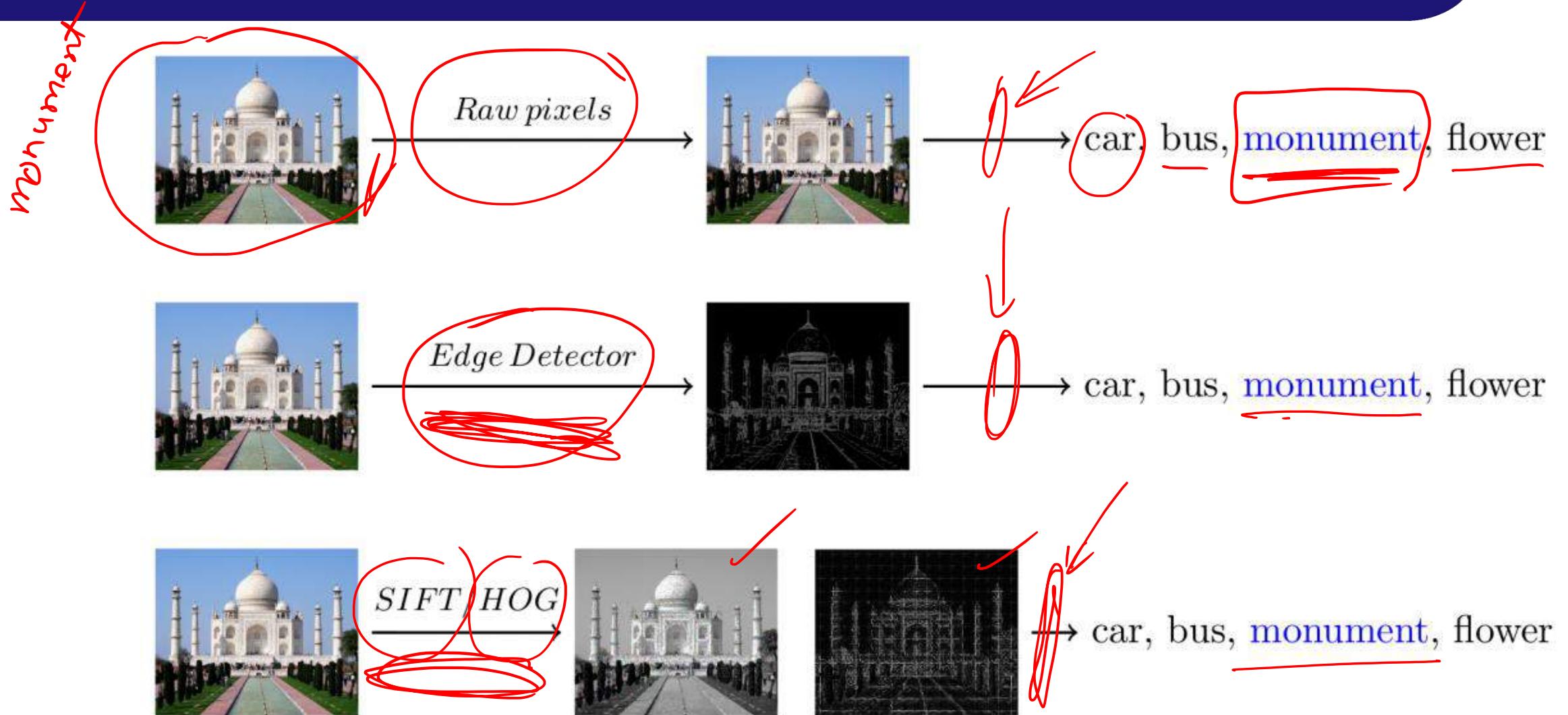


**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) Classification Pipeline

Dr. Kamlesh Tiwari

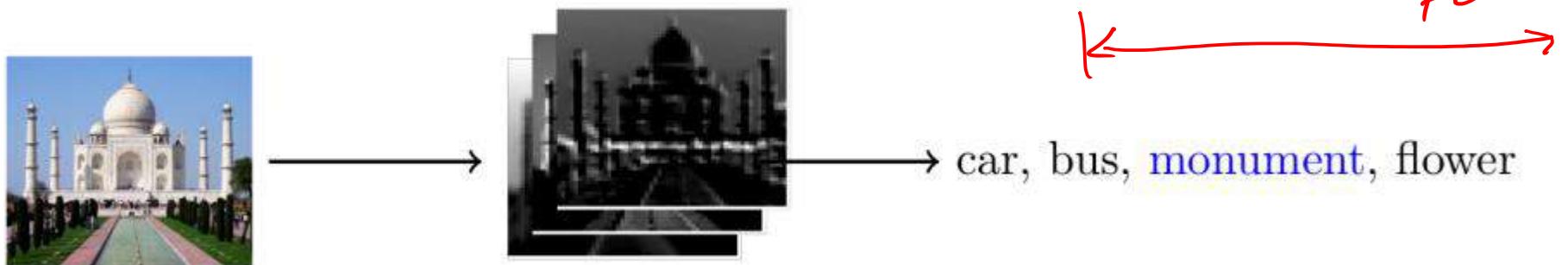
# Classification Pipeline



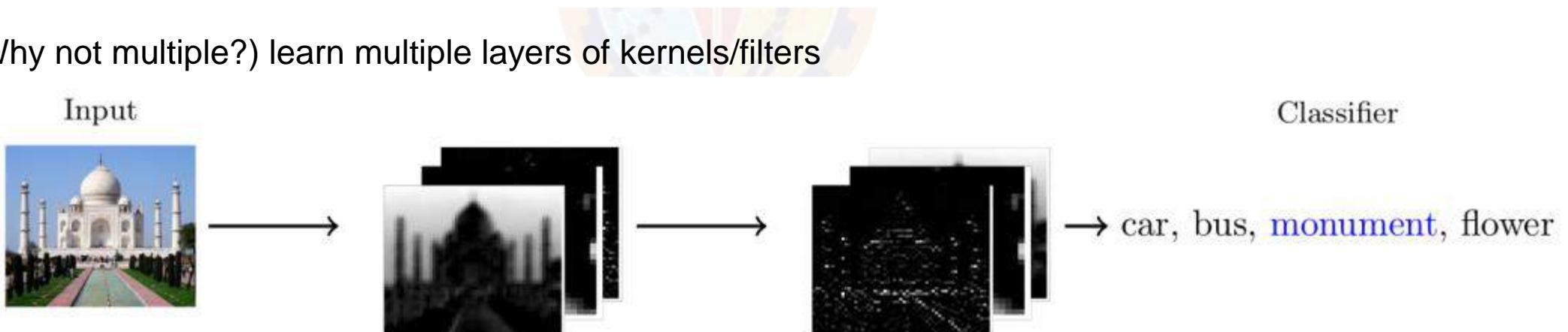
- Where is learning? (hand craft features, then learn weights for classification). One can see feature as a convolution.

# Automate feature kernel discovery

- Instead of handcrafted kernels, learn filters



- (Why not multiple?) learn multiple layers of kernels/filters



Treating these kernels as parameters and learning them.



# Thank You!

In our next session: sparse connectivity in CNN

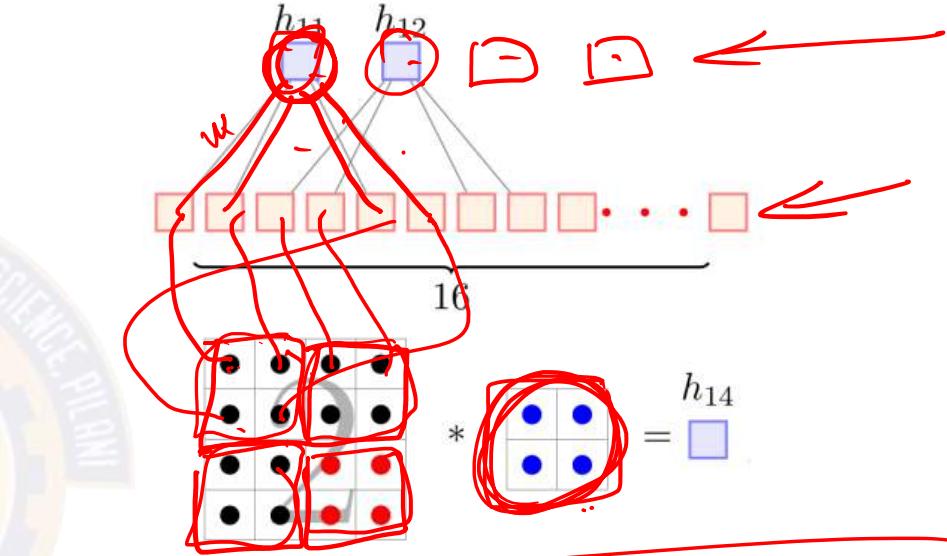
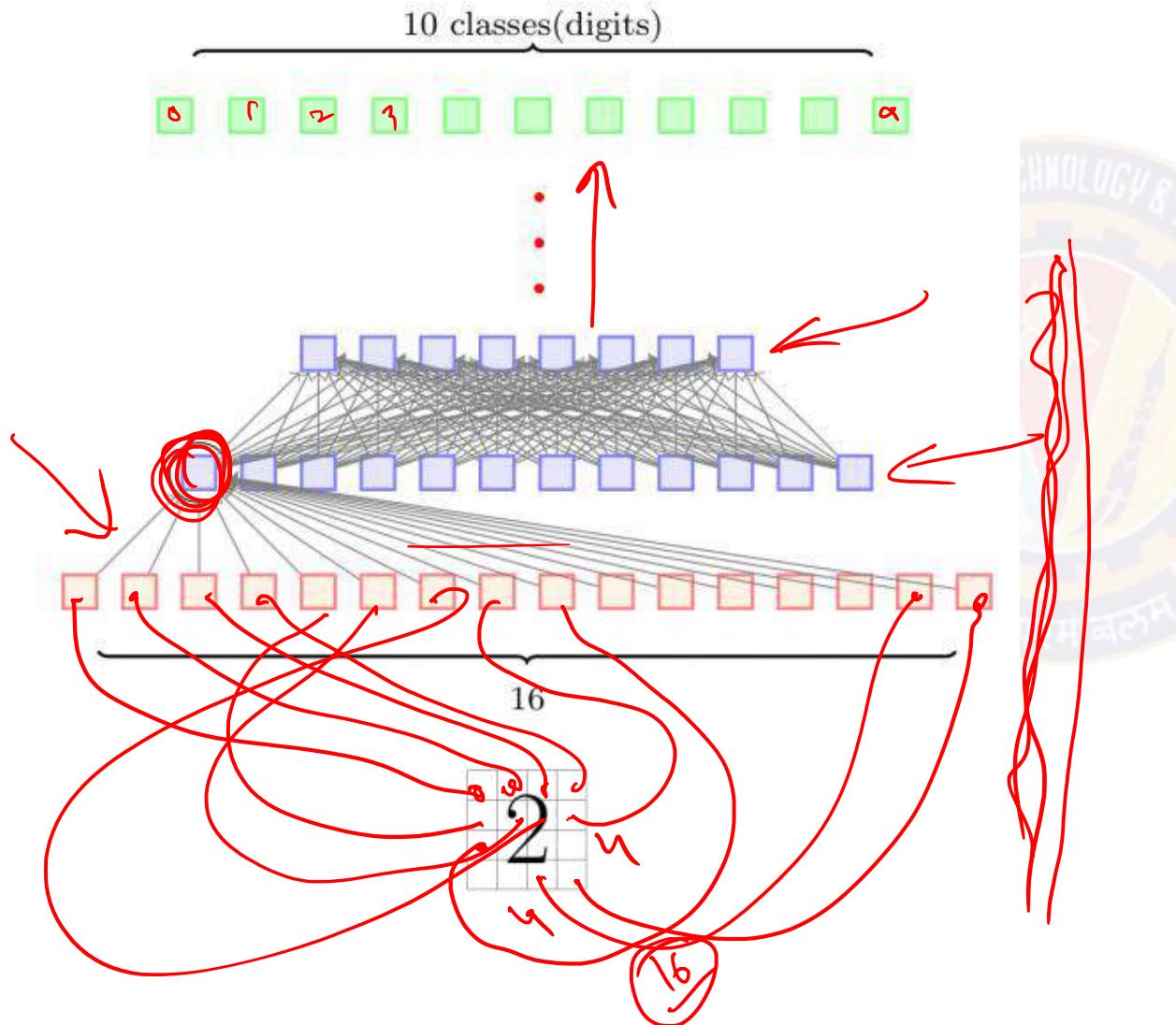


**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

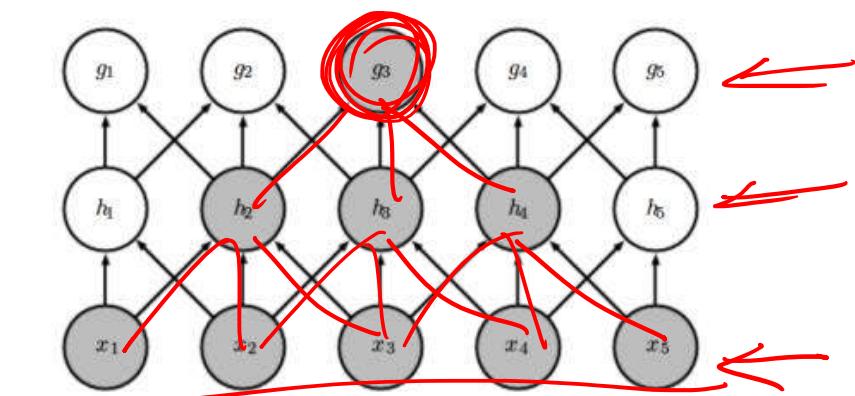
# CONVOLUTIONAL NEURAL NETWORKS (CNN) Sparse Connectivity in CNN

**Dr. Kamlesh Tiwari**

# CNN has sparse connectivity with respect to NN

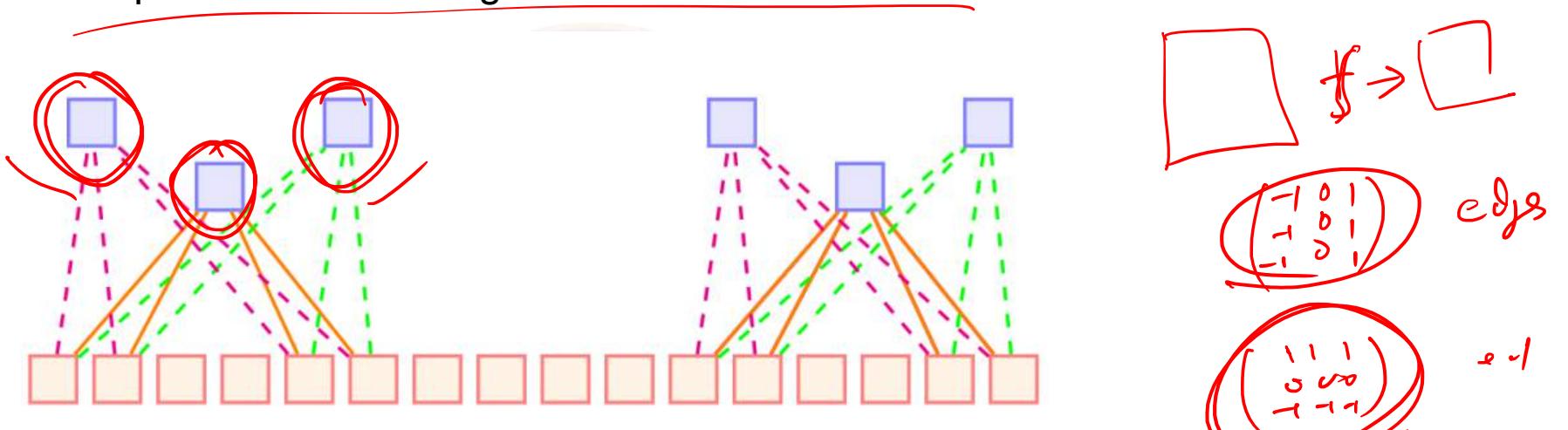


Interactions are preserved, even with reduced model parameters,

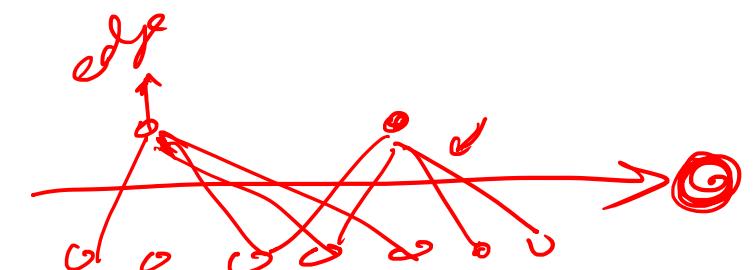


# Weight sharing in CNN

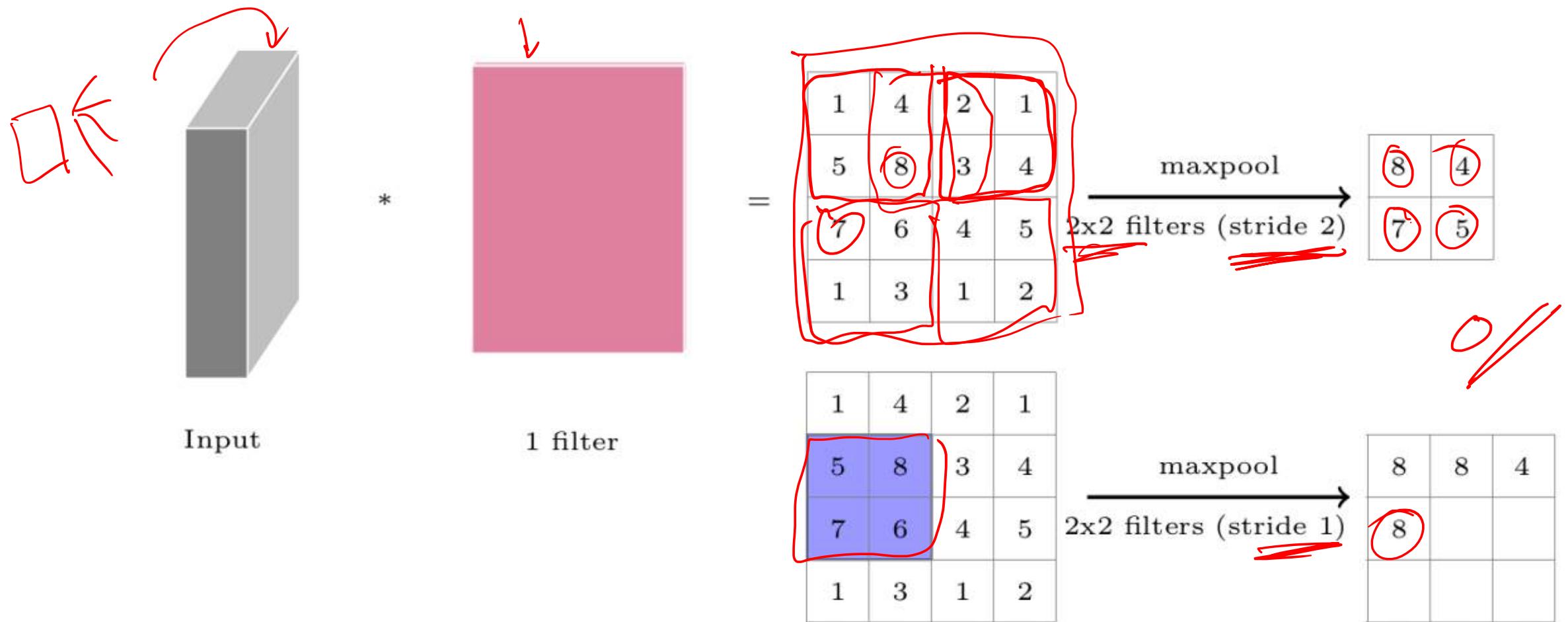
- One place you are extracting edge other place something else? So we do not want the kernel to be different for different portions of the image.



- Weight sharing in CNN makes the job of learning weights easier
- Multiple kernels help get different feature at the same level



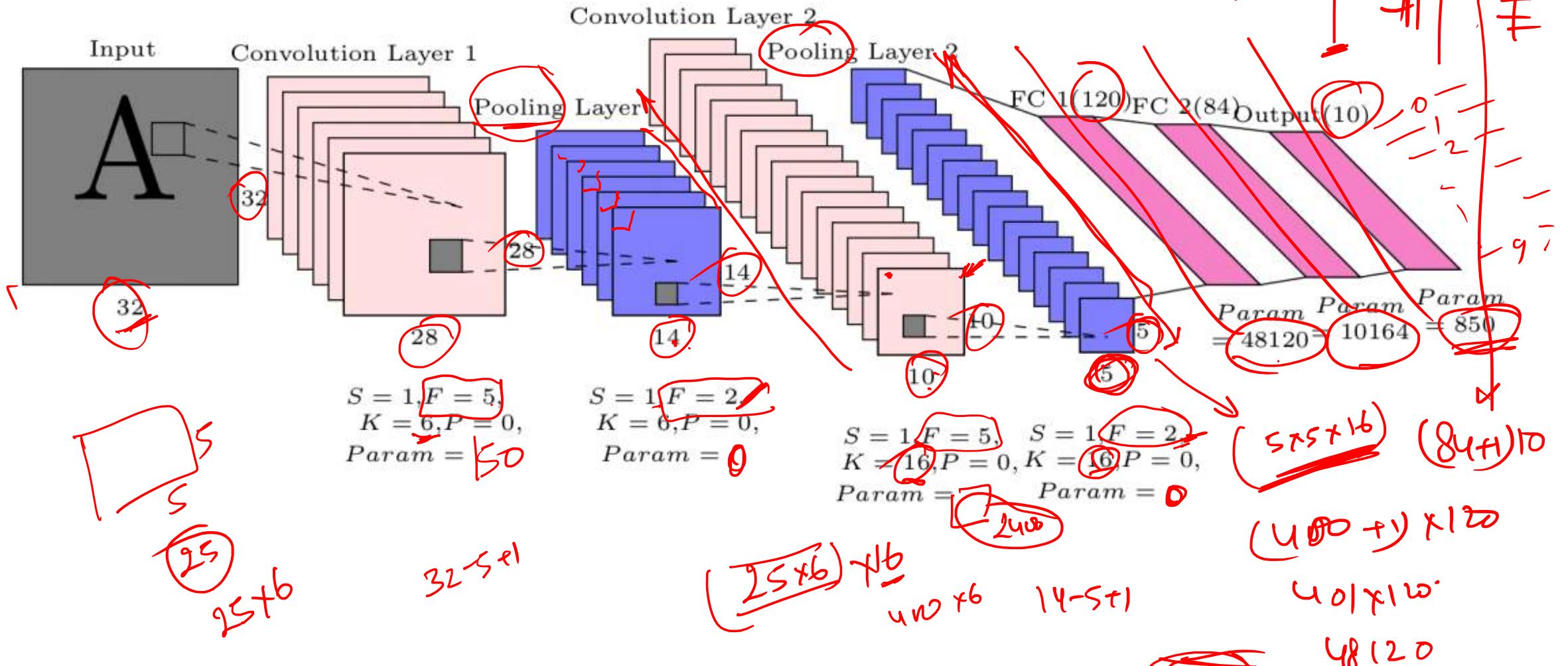
# Pooling (max, min, average)



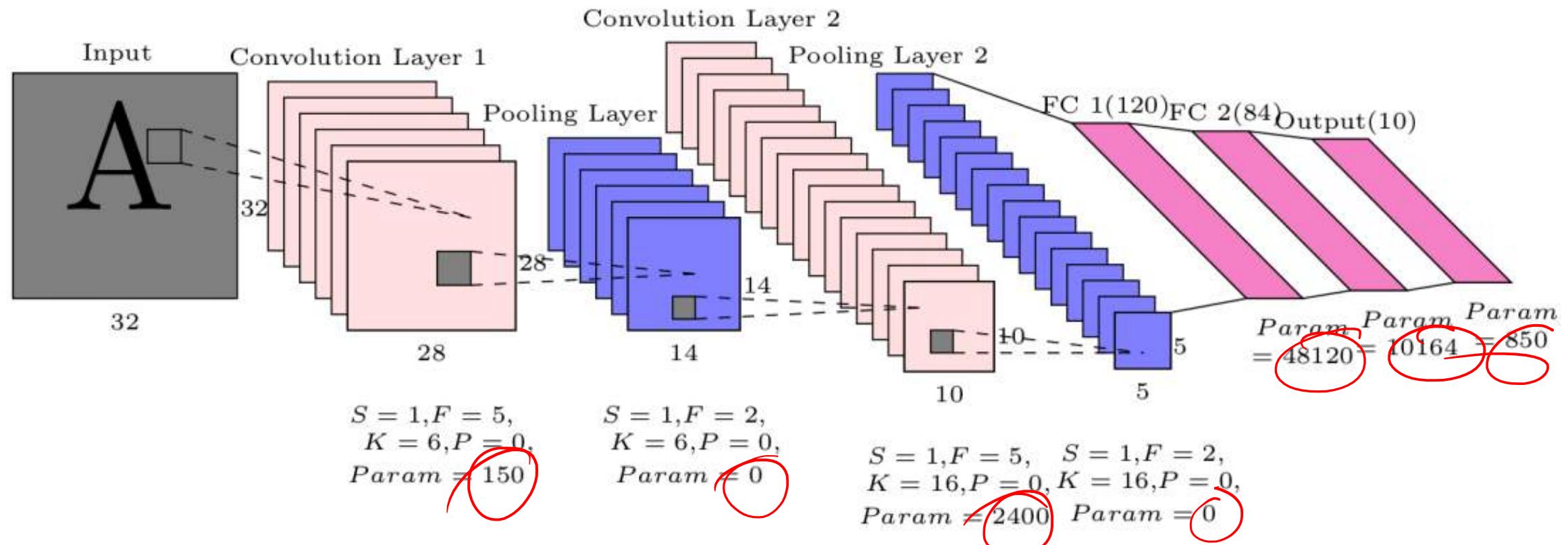
- Training CNN? backpropagation!

# Case-Study: LeNet-5

Handwritten character recognition



# LeNet-5 for handwritten character recognition





# Thank You!

In our next session: Popular CNN Architectures

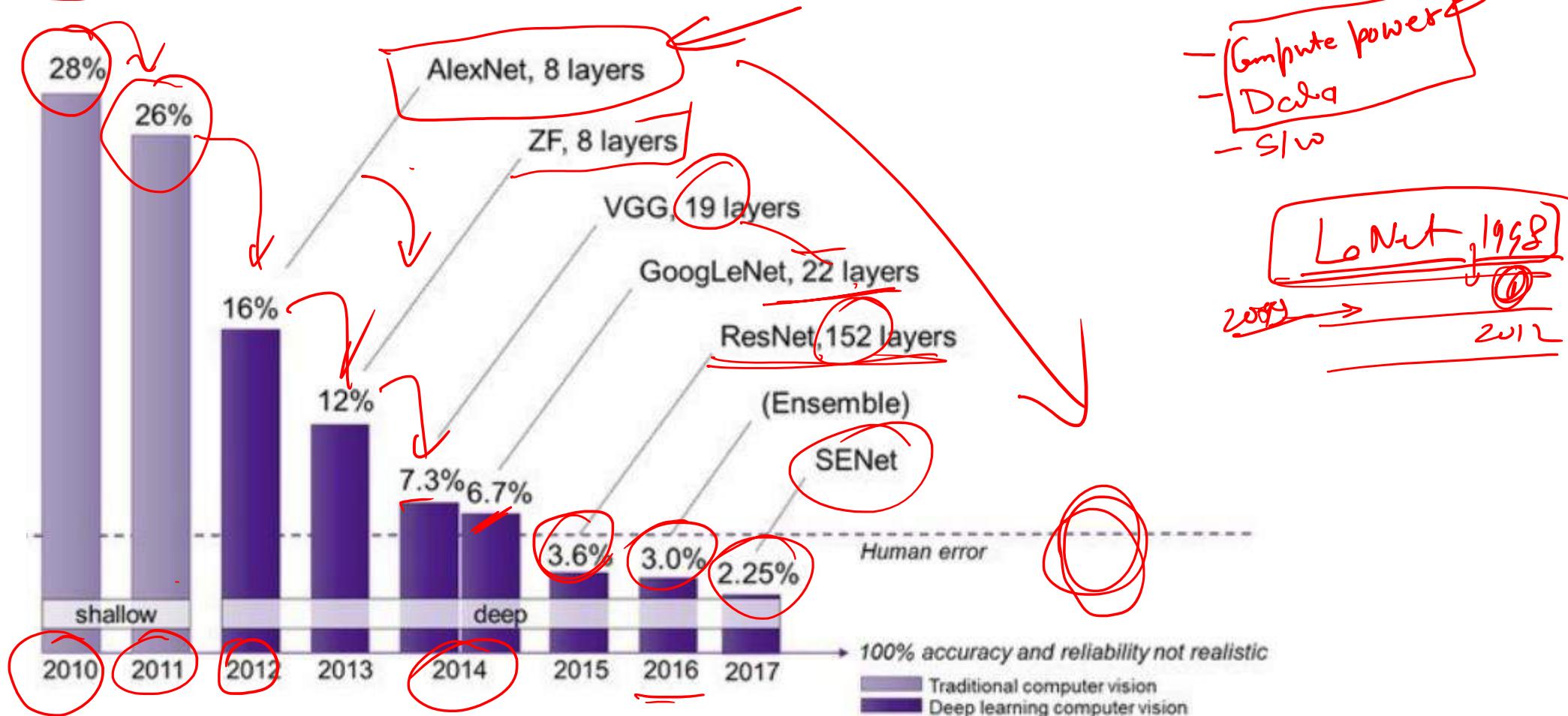


**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) Popular Architectures

**Dr. Kamlesh Tiwari**

# ImageNet ILSVRC



- (2009) 22K category, 14M images
- Challenge 1000 class, 1431167 images
- HoG, LBP, SVM ...



# Thank You!

In our next session: Popular CNN Architectures

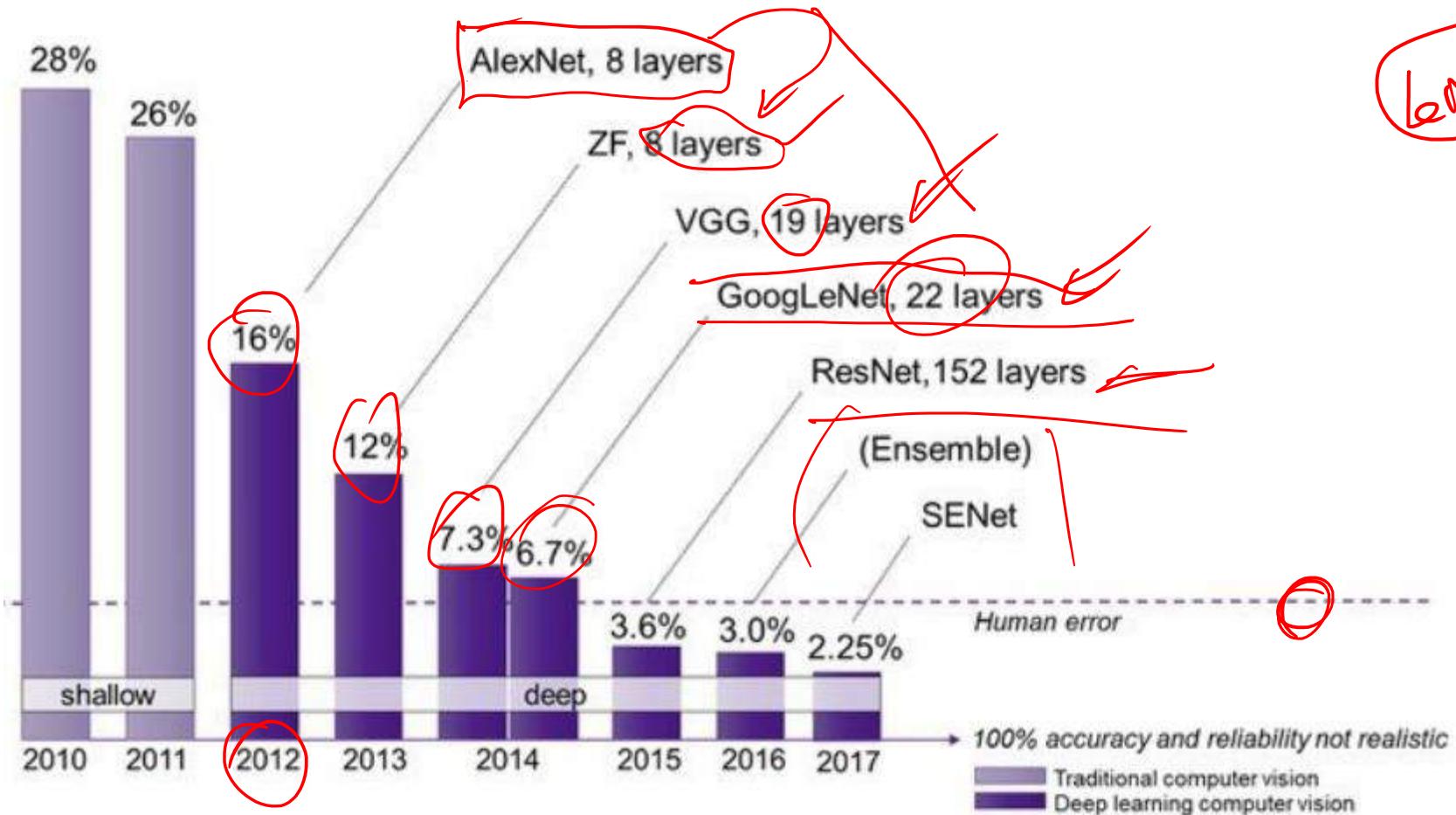


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) AlexNET, ZF-NET, VGG-16

**Dr. Kamlesh Tiwari**

# ImageNet ILSVRC



- (2009) 22K category, 14M images
- Challenge 1000 class, 1431167 images
- HoG, LBP, SVM ...

# AlexNet

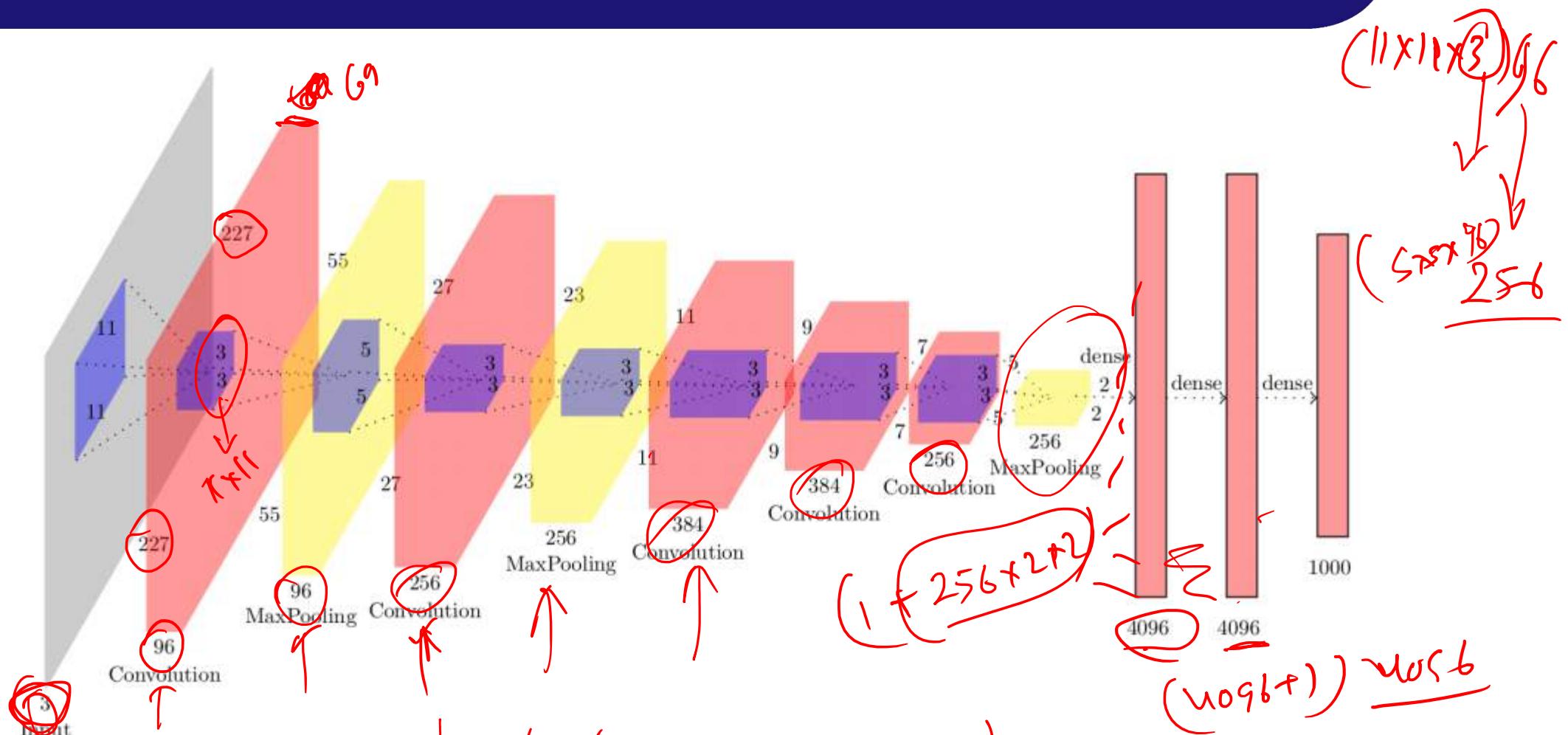


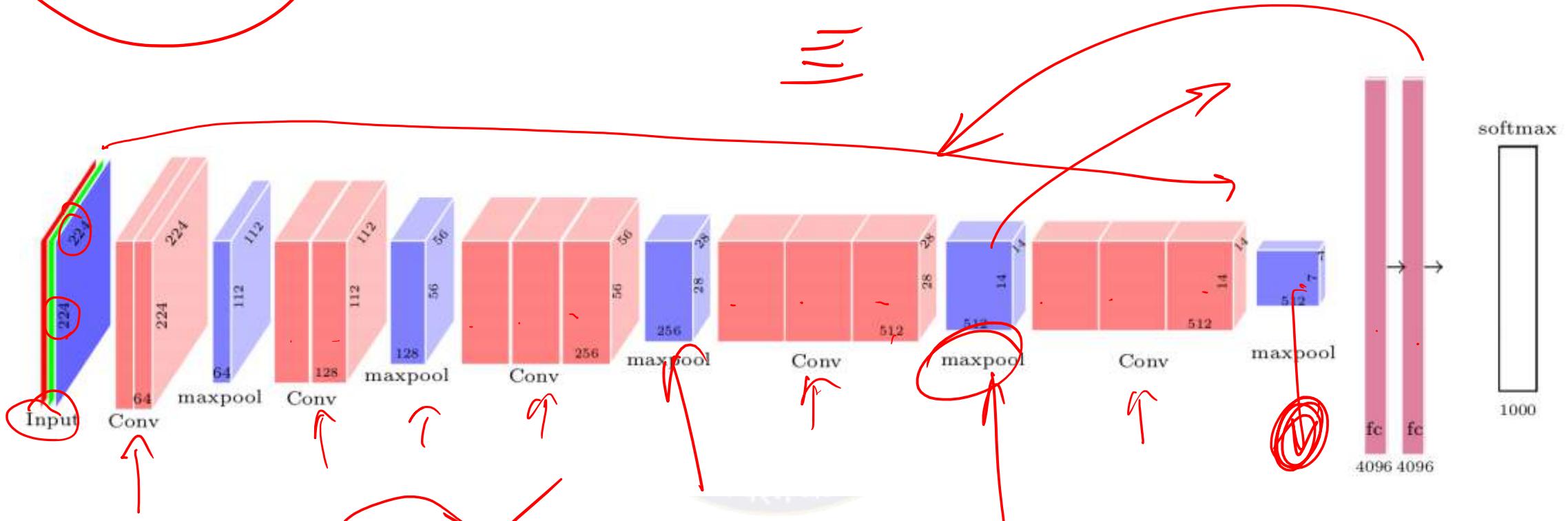
Image size  $227 \times 227 \times 3 \rightarrow [96 F=11, S=4, P=0] \rightarrow$  [MaxPool F=3, S=2]  $\rightarrow [256 F=5, S=1, P=0] \rightarrow$  [MaxPool F=3, S=2]  $\rightarrow [384 F=3, S=1, P=0] \rightarrow [384 F=3, S=1, P=0] \rightarrow [256 F=3, S=1, P=0] \rightarrow$  [MaxPool F=3, S=2]  $\rightarrow$  FC 4096  $\rightarrow$  FC 1000

# ZFNet



Image size  $227 \times 227 \times 3 \rightarrow [69 F=7, S=4, P=0] \rightarrow [\text{MaxPool } F=3, S=2] \rightarrow [256 F=5, S=1, P=0] \rightarrow [\text{MaxPool } F=3, S=2] \rightarrow [512 F=3, S=1, P=0] \rightarrow [1024 F=3, S=1, P=0] \rightarrow [512 F=3, S=1, P=0] \rightarrow [\text{MaxPool } F=3, S=2] \rightarrow \text{FC } 4096 \rightarrow \text{FC } 1000$

# VGG16



- Kernel size is always  $3 \times 3$
  - 16M parameters in pre-FC and 122 in FC. First FC layer is huge
  - Layers represents abstract representation and can be reused (FC or Conv)



# Thank You!

In our next session: Popular CNN Architectures Continues...

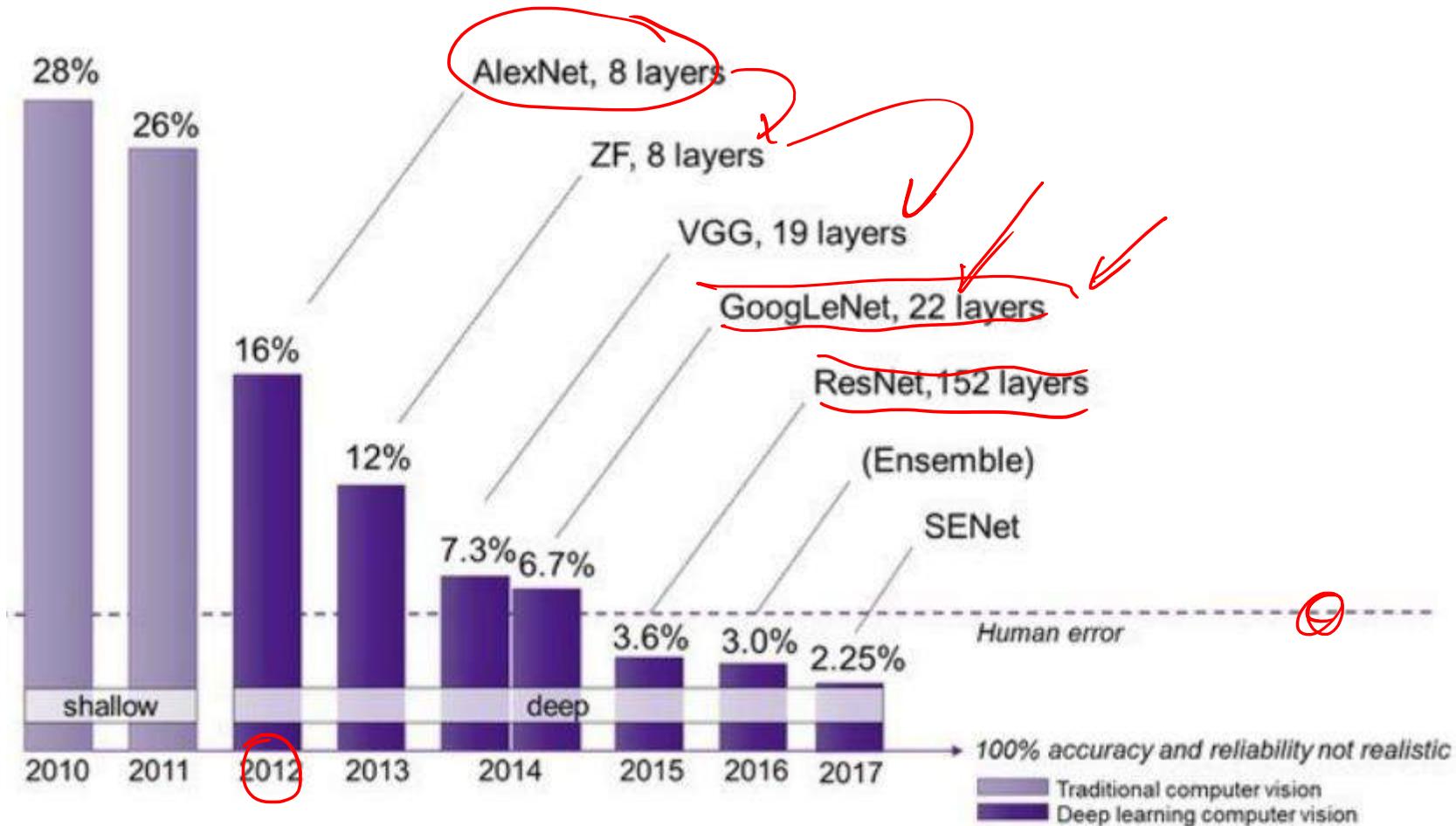


**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# CONVOLUTIONAL NEURAL NETWORKS (CNN) Inception, ResNET

**Dr. Kamlesh Tiwari**

# ImageNet ILSVRC



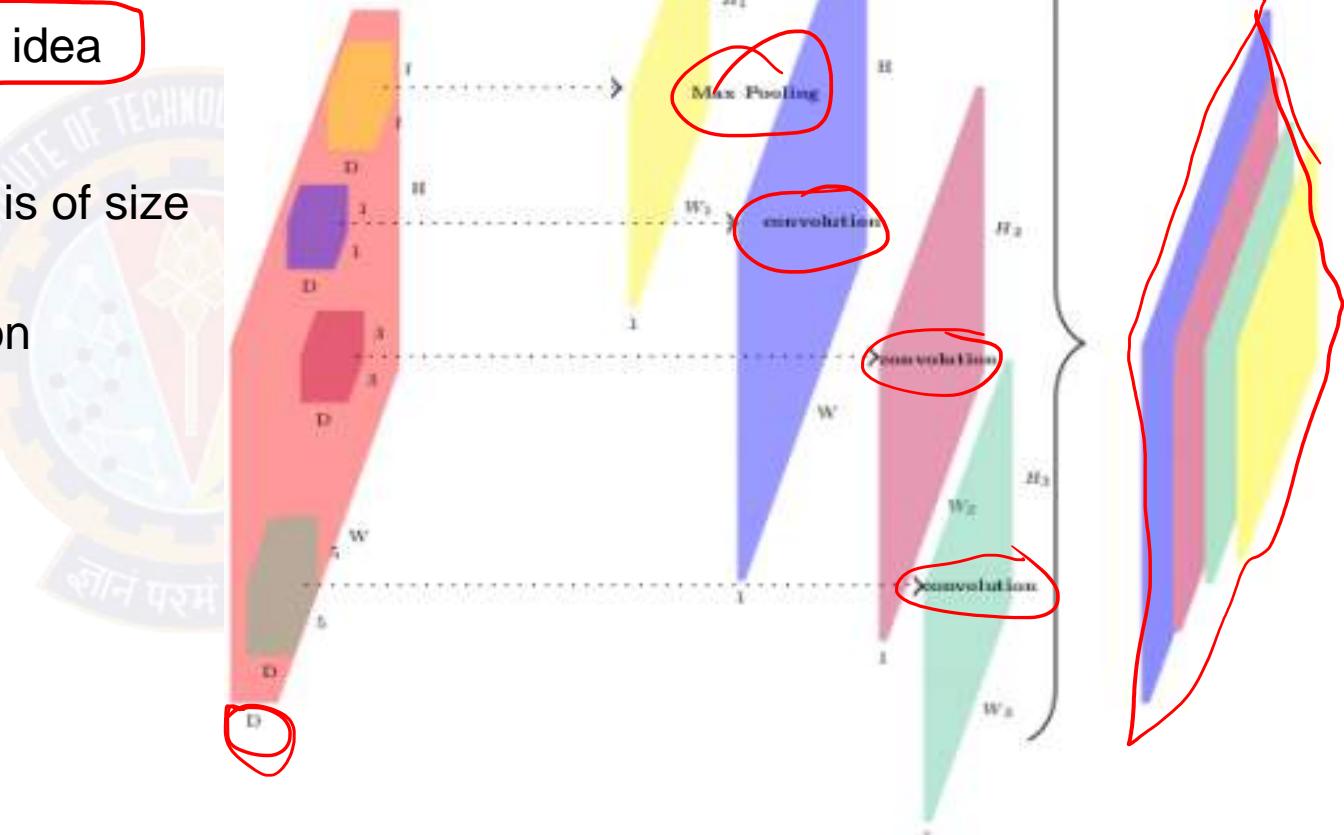
- (2009) 22K category, 14M images
- Challenge 1000 class, 1431167 images
- HoG, LBP, SVM ...

# GoogLeNet

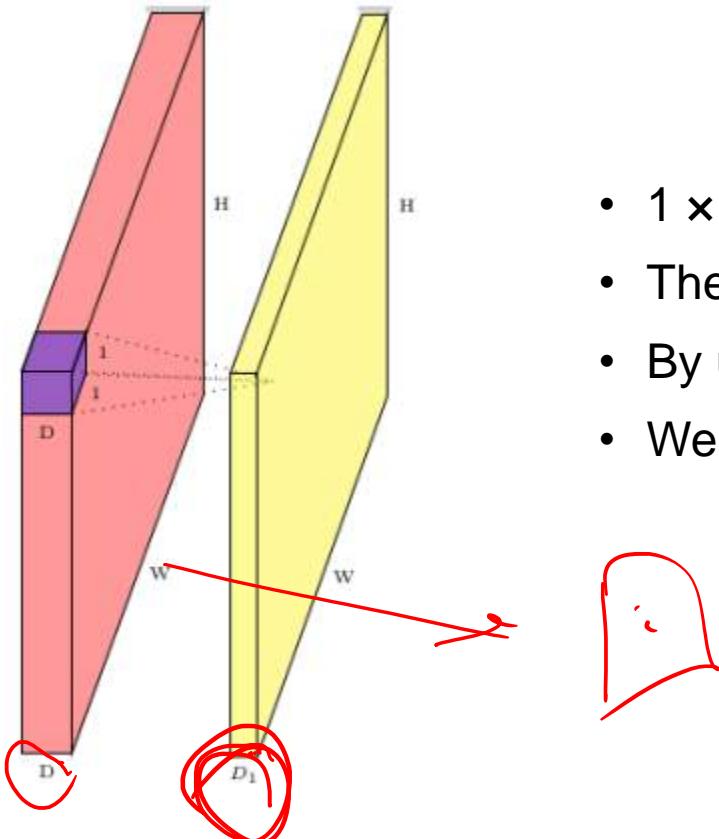
- Recall scale invariance in SIFT
- Multiple filters of different size is a good idea
- With  $W \times H \times D$  input and  $F \times F \times D$
- Filter and  $S = 1$  and no padding, output is of size  $(W - F + 1) \times (H - F + 1)$
- Each value needs  $F \times F \times D$  computation

Can we reduce this computation a bit?

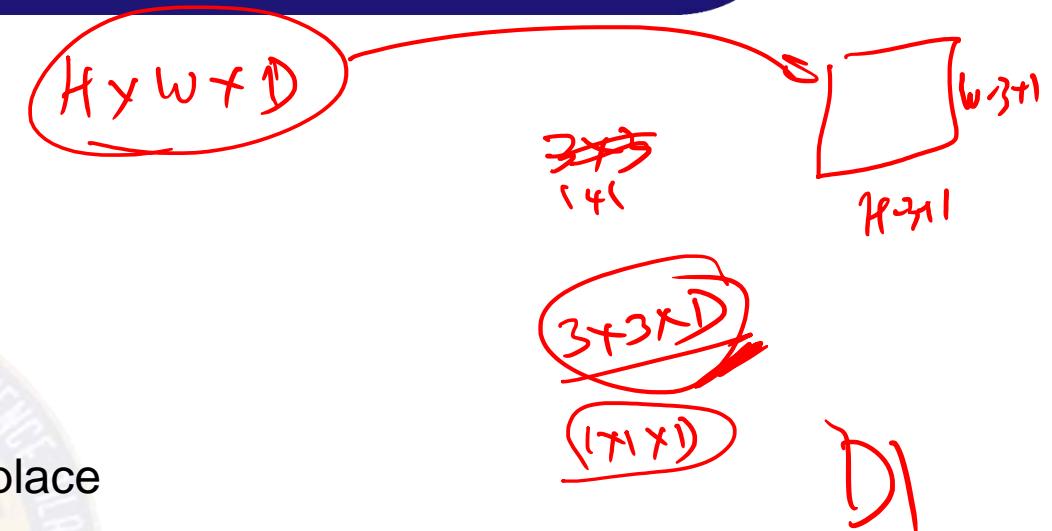
- Idea is to have  $1 \times 1$  computation



# $1 \times 1$ convolution

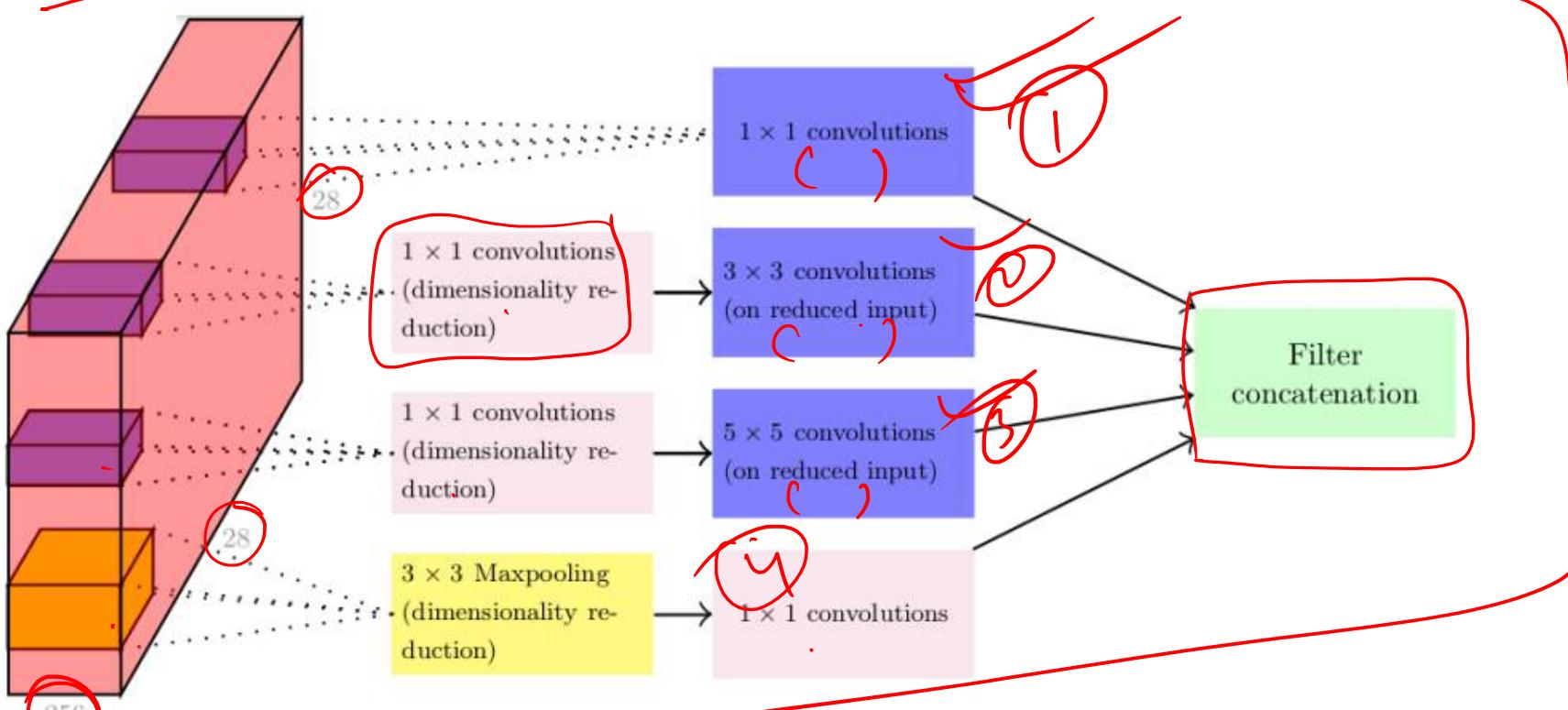


- $1 \times 1$  is  $1 \times 1 \times D$
- They produce one output place
- By using  $D_1$  such  $1 \times 1$  convolution output becomes  $F \times F \times D_1$
- We have  $D_1 < D$



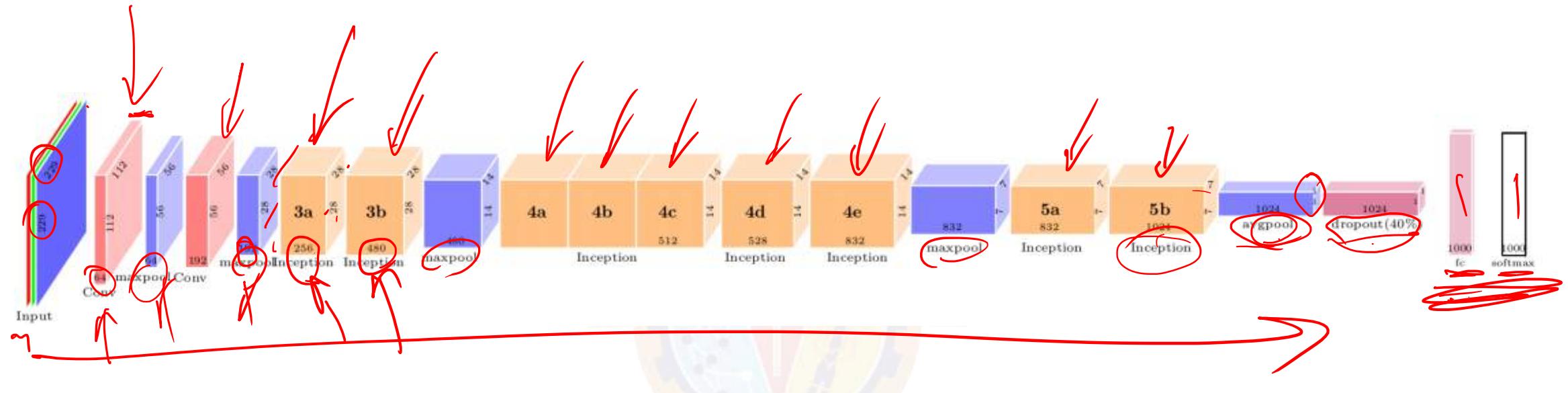
A diagram illustrating the computation of output dimensions. It shows a red oval containing the formula  $D_1 < D$ . To its right is a red box with dimensions  $1 \times 1 \times 1$ .

# Inception Block: Multiple convolutions



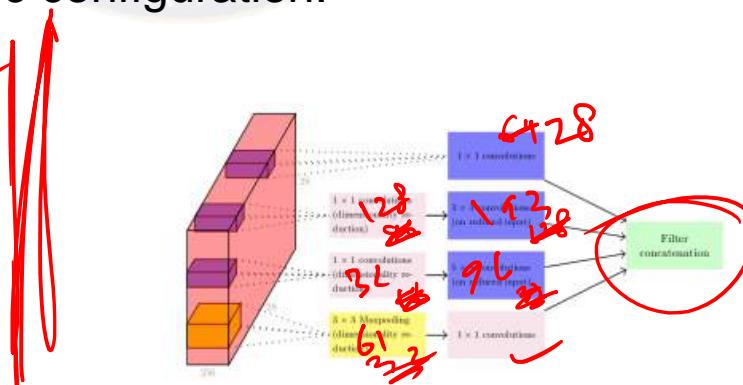
- 1 × 1 convolution
- 1 × 1 convolution followed by 3 × 3
- 1 × 1 convolution followed by 5 × 5
- 3 × 3 maxpool followed by 1 × 1
- Appropriate padding is done to make things of same size

# GoogLeNet

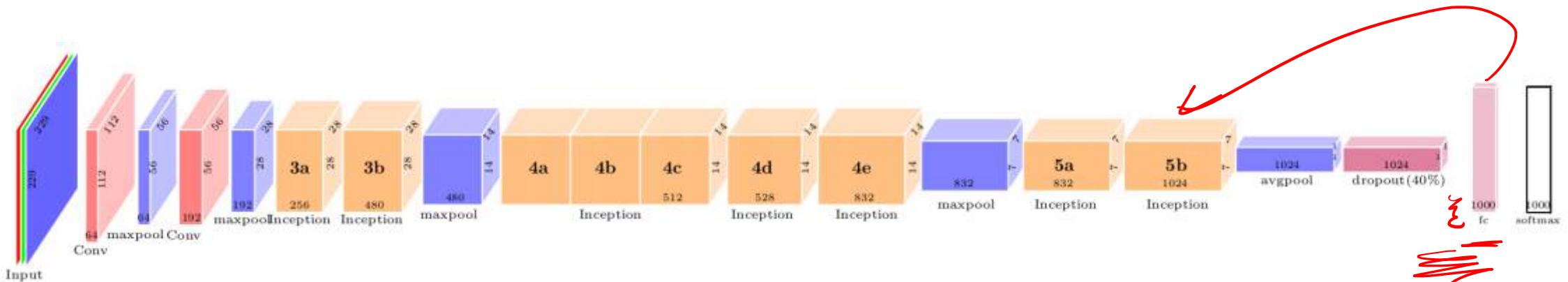


- Input is RGB  $229 \times 229$
- Each inception module have very specific configuration.

(3a)	$192 \times 28 \times 28$	64 96 128 16 32 32
(3b)	$256 \times 28 \times 28$	28 128 192 32 96 64
(4a)	$48 \times 14 \times 14$	192 96 208 16 48 96
(4b)	$512 \times 14 \times 14$	160 112 224 24 64 64
(4c)	$512 \times 14 \times 14$	128 128 256 24 64 64
(4d)	$512 \times 14 \times 14$	112 144 228 32 64 64
(4e)	$528 \times 14 \times 14$	256 160 320 32 128 128
(5a)	$832 \times 7 \times 7$	256 160 320 32 128 128
(5b)	$832 \times 7 \times 7$	384 192 384 48 124 128



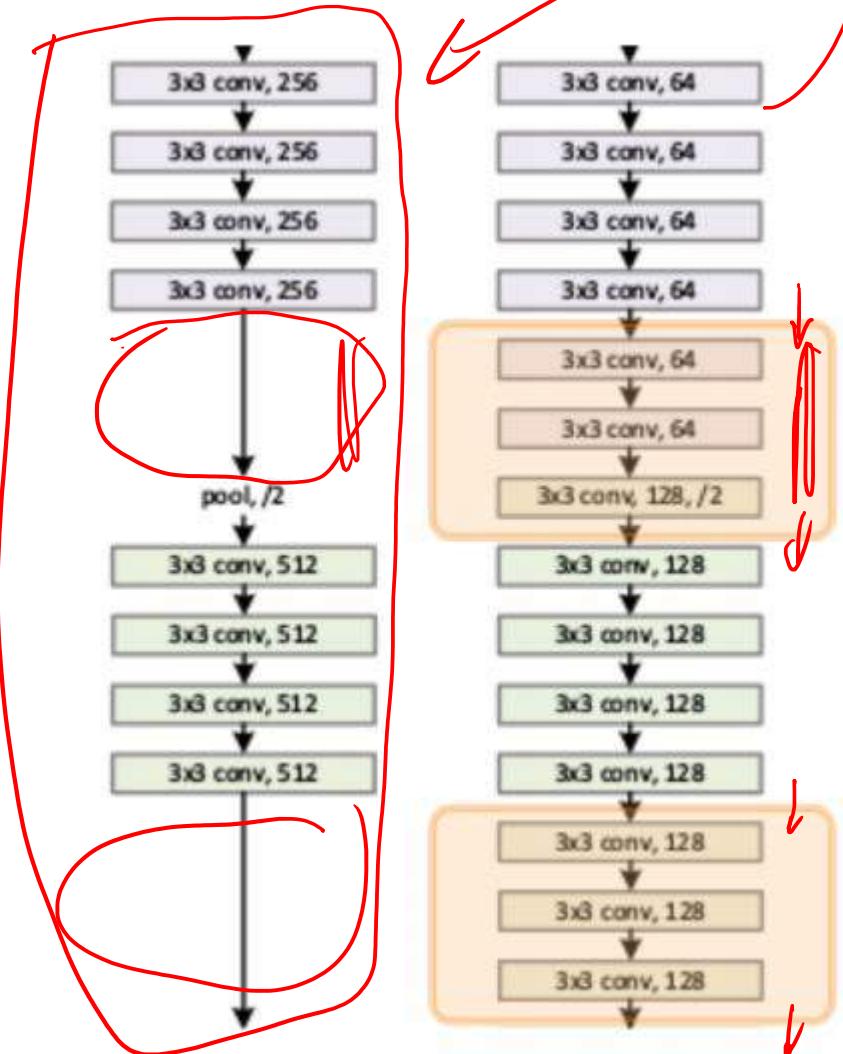
# GoogLeNet



- VGGNET has  $512 \times 7 \times 7$  size at pre-FC this was an issue to connect with 4096
- GoogLeNet applies a average pool. Gives 49 time reduction. has 1024 values only
- Dropout and connect to 1000
- 12 times less connections as compared to AlexNet
- 2 times more computation as compared to AlexNet
- Very high accuracy. Error reduced from 16% -to- 6.7%

# ResNet

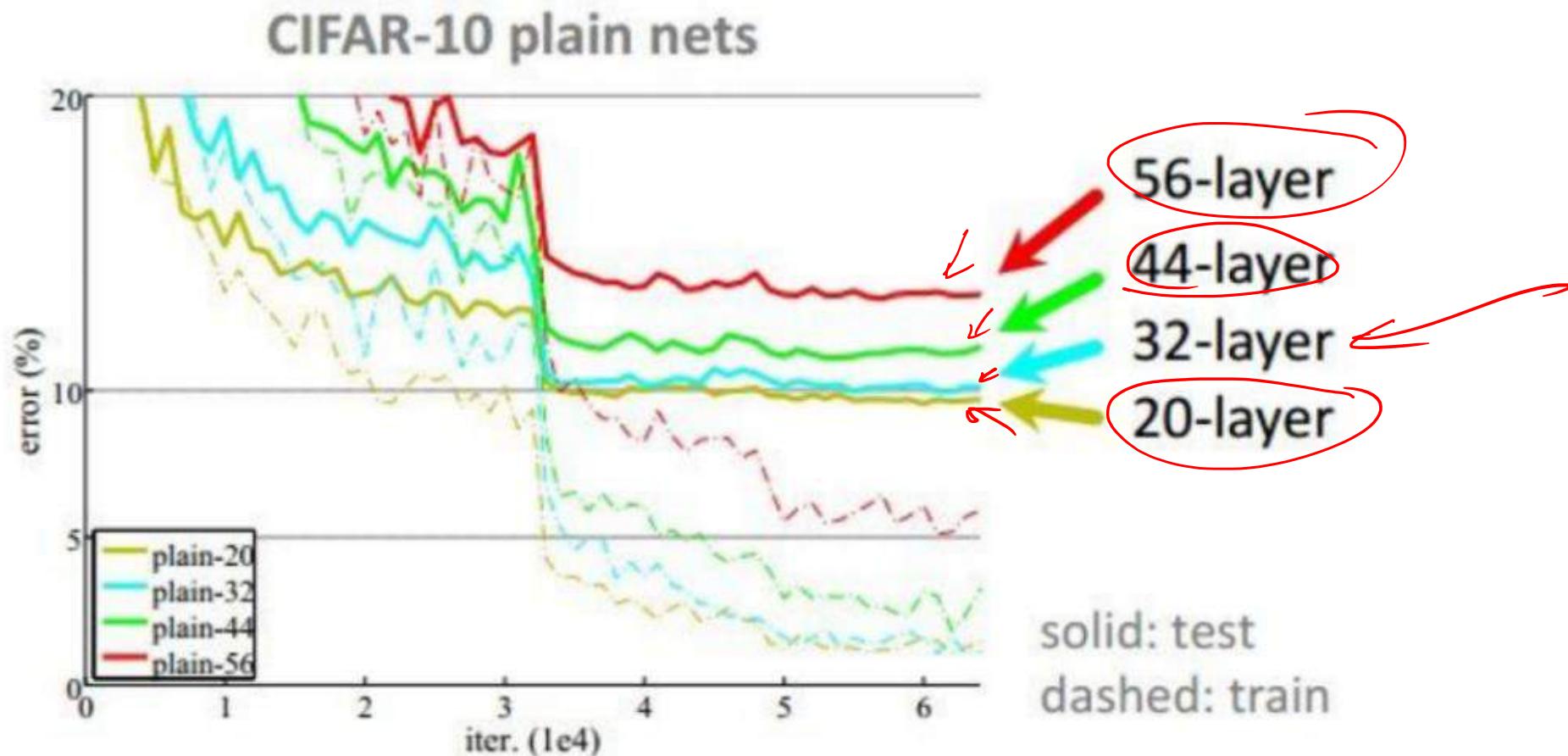
If a shallow neural network works well. What would happen if we add more layers?



- Deep network should also work well (It would learn identity in new layers)

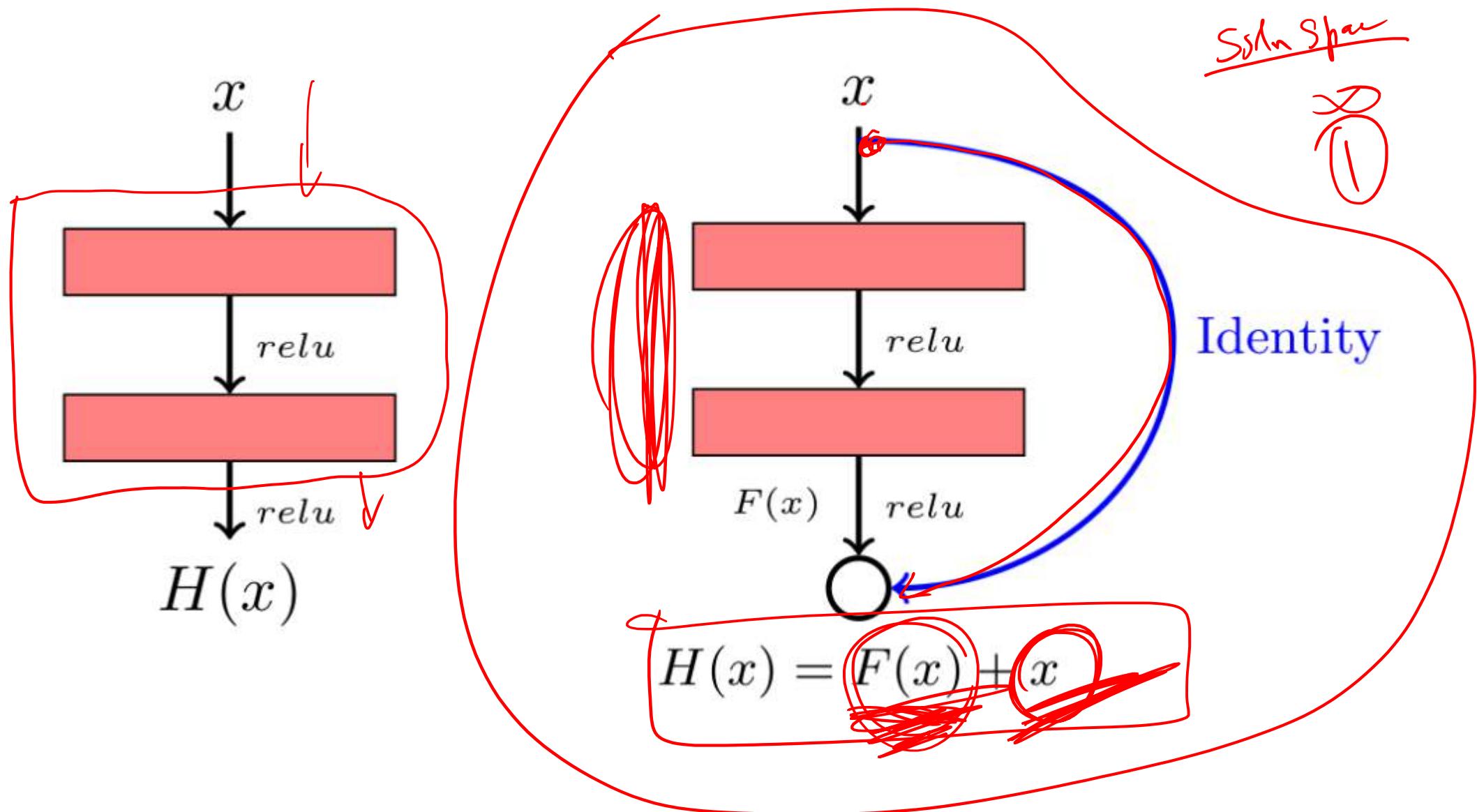
# ResNet

But, in practice it was not happening

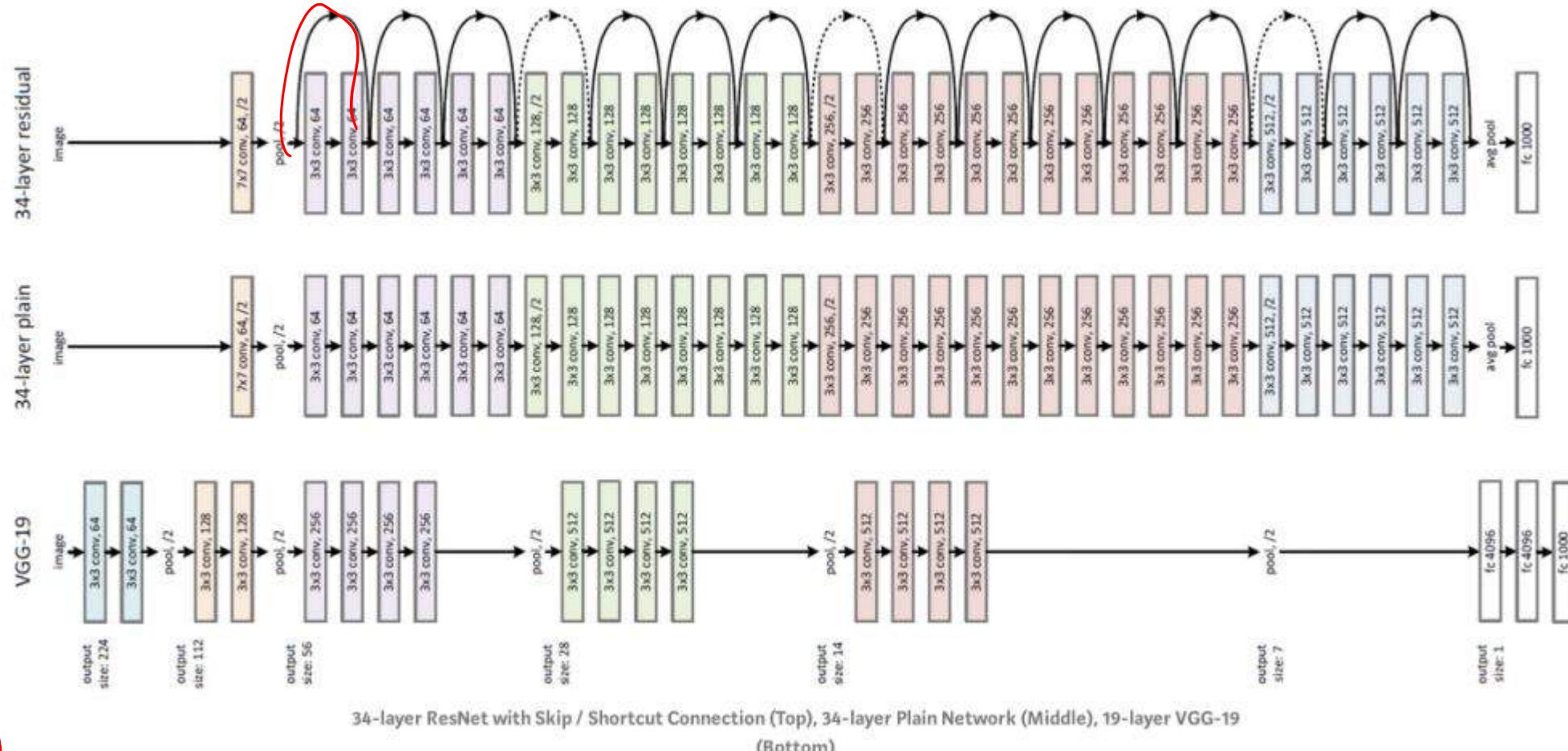


Why? Identity is one of the solution in large domain.

# Let me tell this to the network



# ResNet Comparison



152-layer deep net was better than human. Only 3.6% error rate ImageNet Classification

Better than the 2nd best system ImageNet Detection: 16% ImageNet Localization: 27% COCO  
Detection: 11% COCO Segmentation: 12%

# ResNet Hyper-parameters and Issues

- Training takes huge time ✓
- Batch Normalization ✓
- Zavier/2 initialization ✓
- SGD and momentum ✓
- Small learning rate 0.1 ✓
- Mini-batch size 256 ✓
- Weight decay ✓
- No Dropout ✓





# Thank You!

In our next session: Object Detection and Segmentation

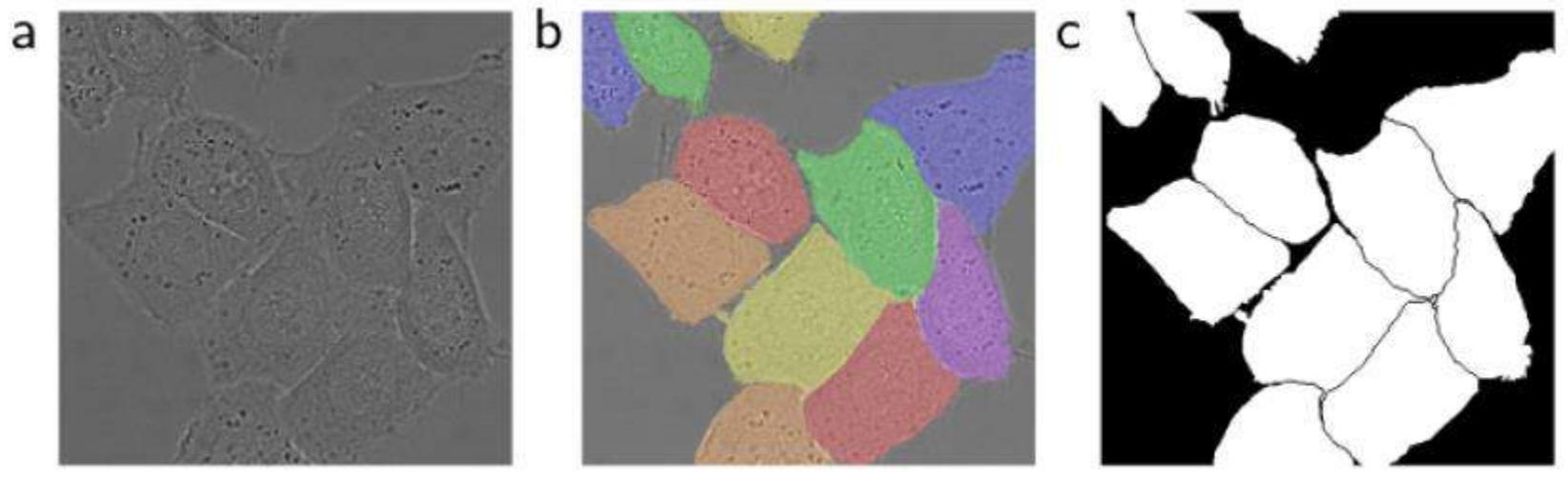


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CNN Case Studies (U-Net)

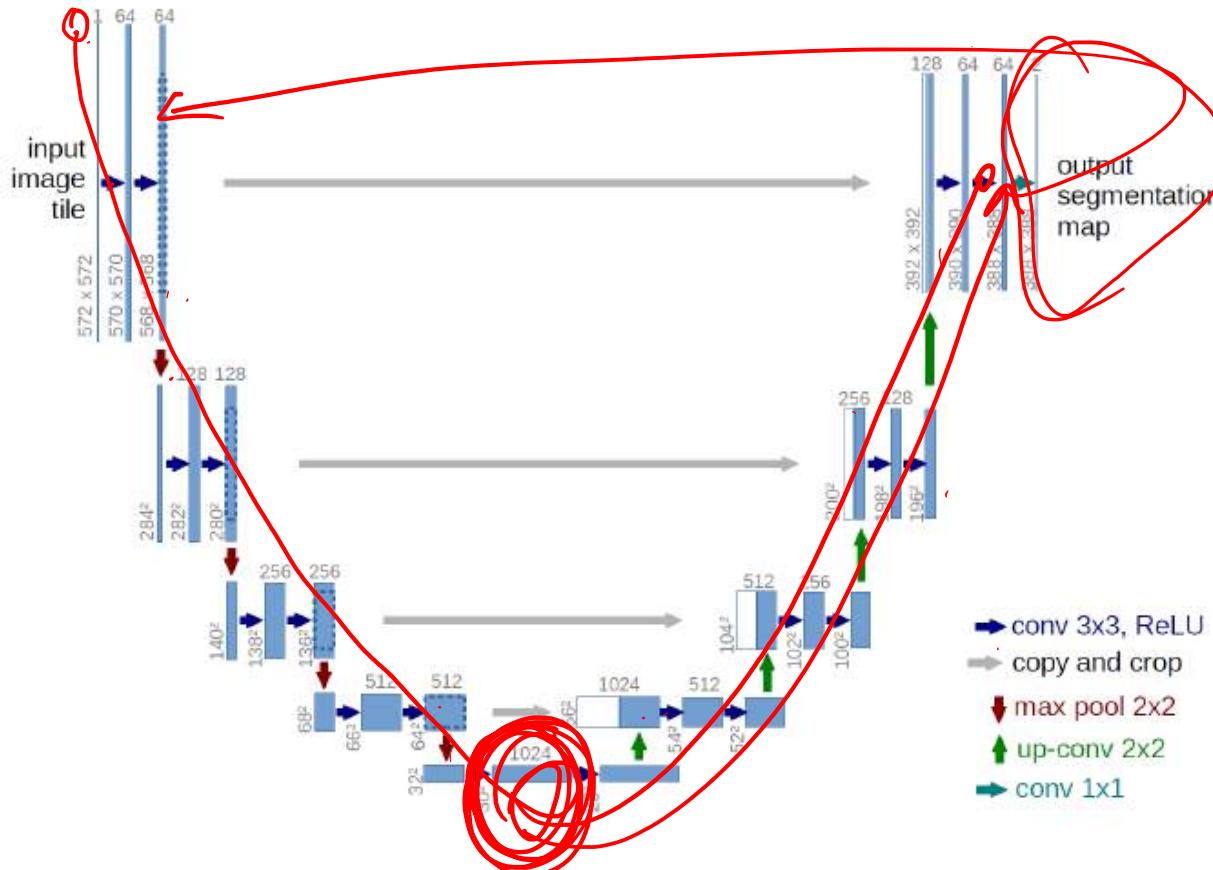
**Dr. Kamlesh Tiwari**

# Segmentation



- ISBI challenge for segmentation of neuronal structures in electron microscopic stacks
- Works with very few training images (30/application) and touching boundary. Yield more precise segmentation
- Data augmentation is essential (mainly shift, rotation and elastic deformation)

# U-Net



- ISBI DIC-HeLa achieved 77.6% IoU as compared to 46.0% second
- ISBI Cell tracking 2015, achieved 92% IoU as compared to 83% second



# Thank You!

Object Detection and Localization (R-CNN):

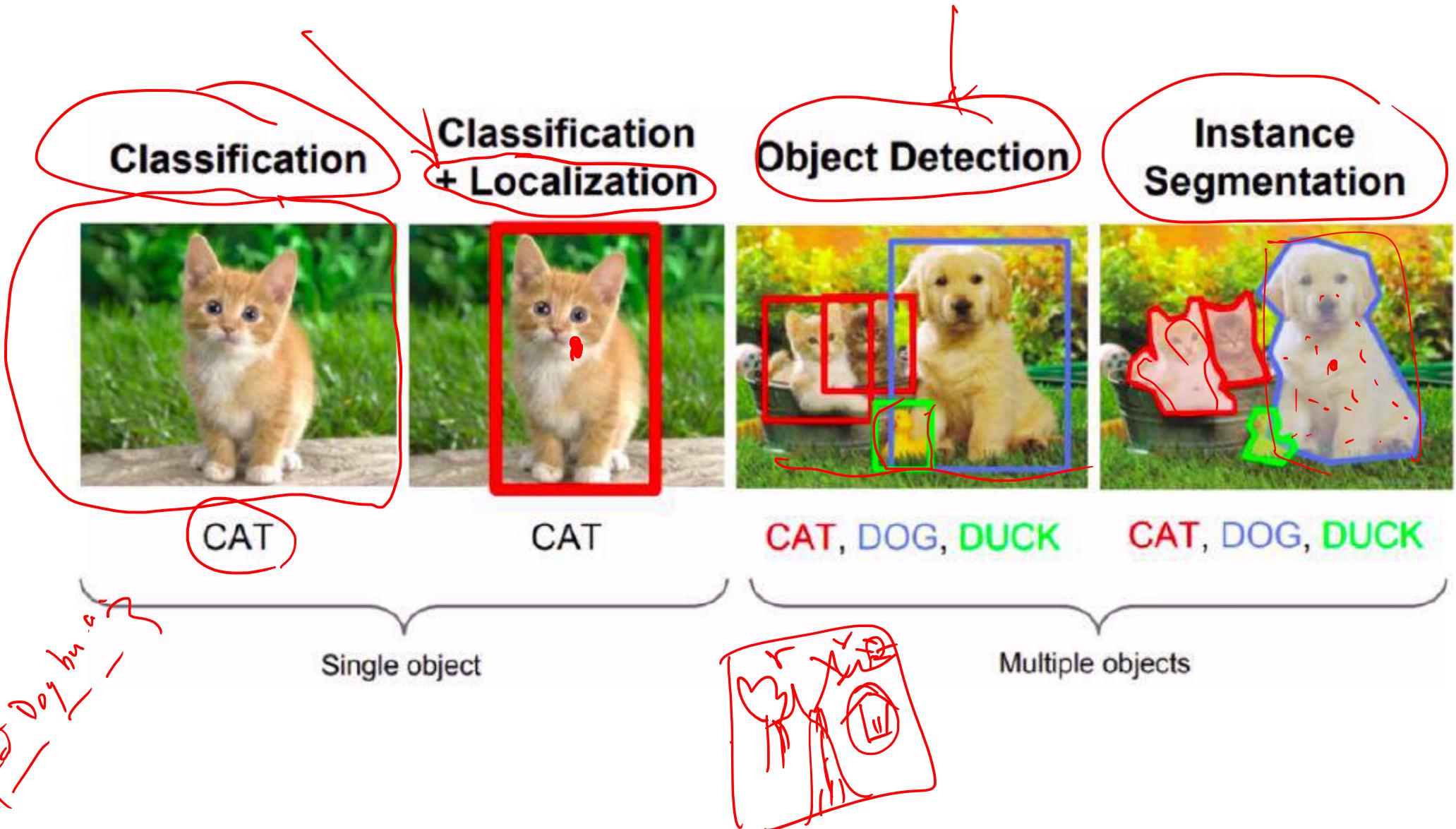


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

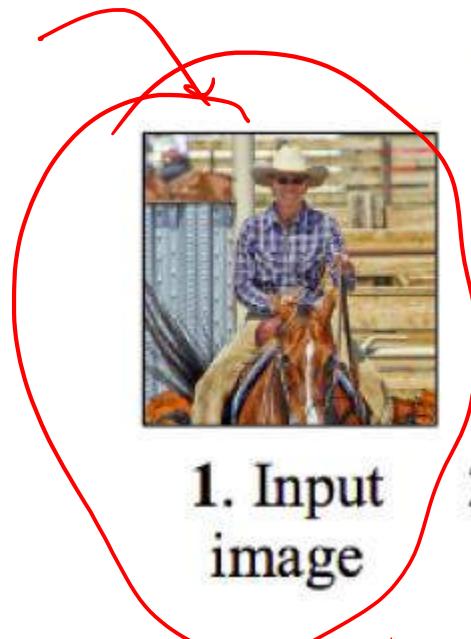
# CNN Case Studies (R-CNN)

**Dr. Kamlesh Tiwari**

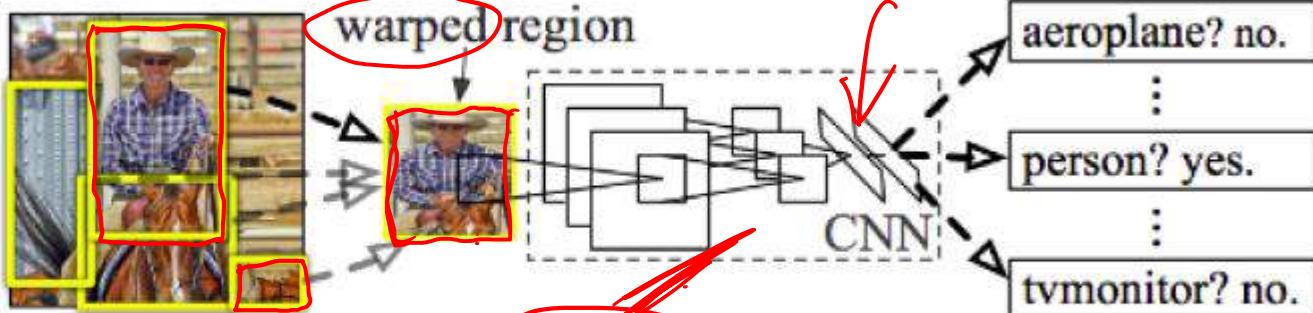
# Object Detection and Localization



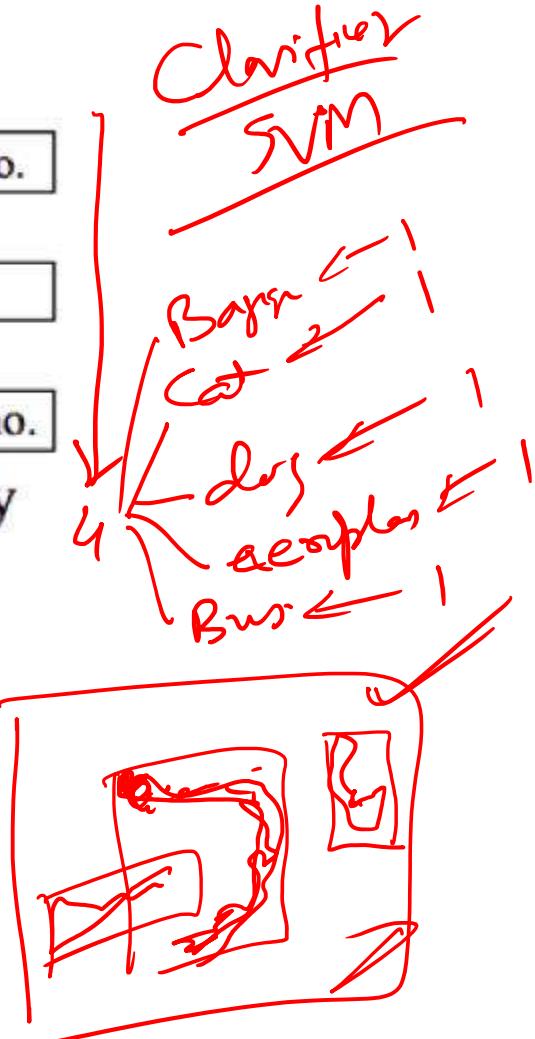
# R-CNN



## R-CNN: *Regions with CNN features*



4. Classify regions



- Region proposals 20K (from external selective search)
- Warped image (resizing)
- SVM for classification (one for each class)

# Thank You!

Object Detection and Localization (Fast and Faster R-CNN):



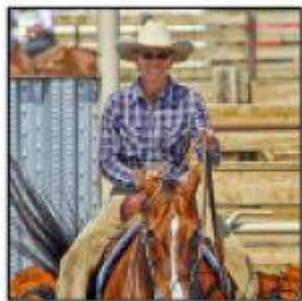
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CNN Case Studies (Fast R-CNN)

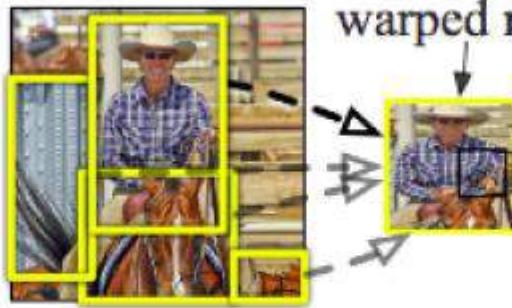
**Dr. Kamlesh Tiwari**

# R-CNN

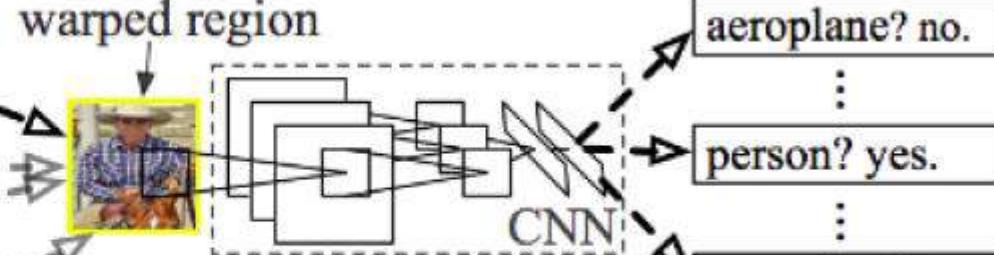
## R-CNN: *Regions with CNN features*



1. Input image



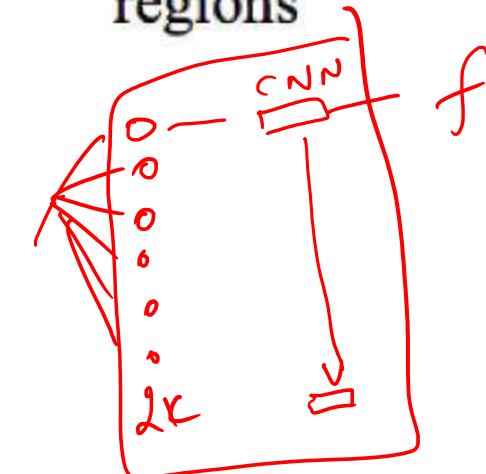
2. Extract region proposals (~2k)



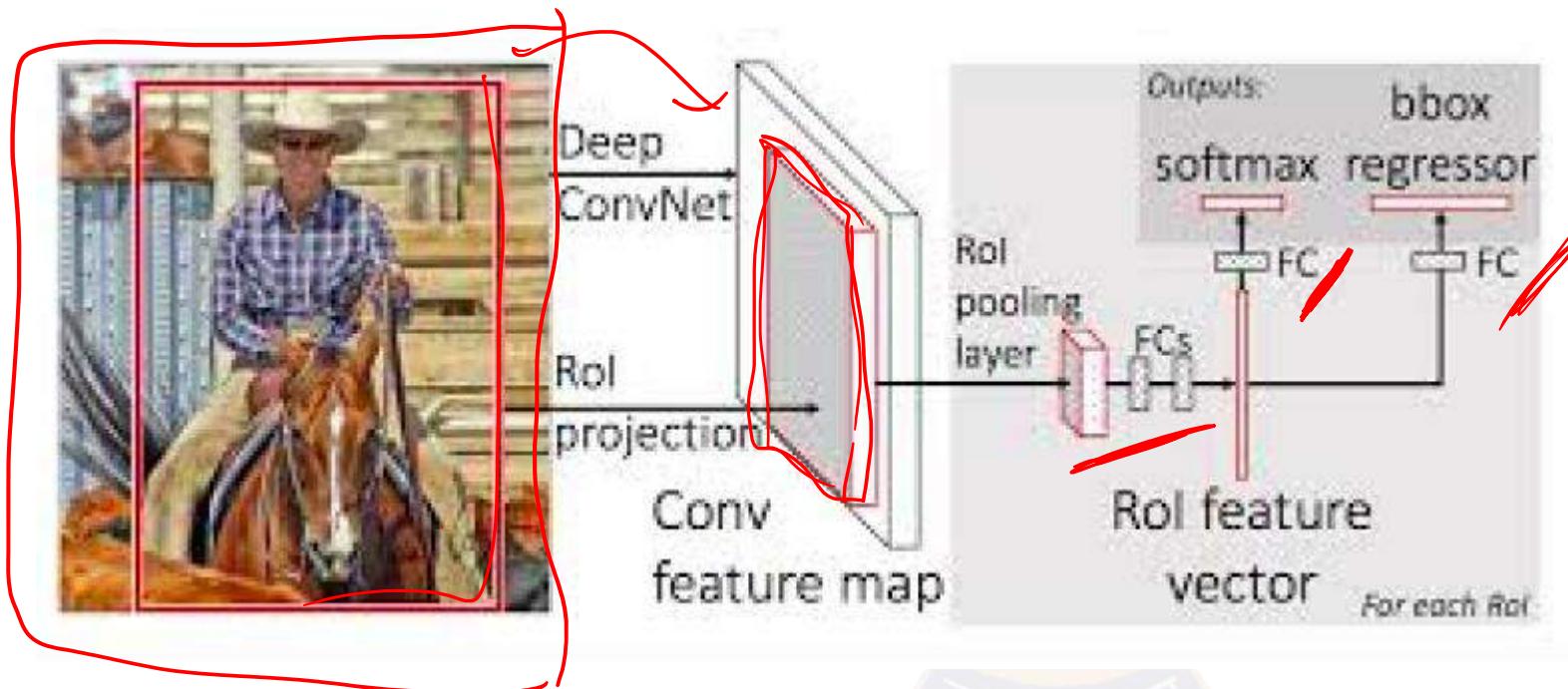
3. Compute CNN features

4. Classify regions

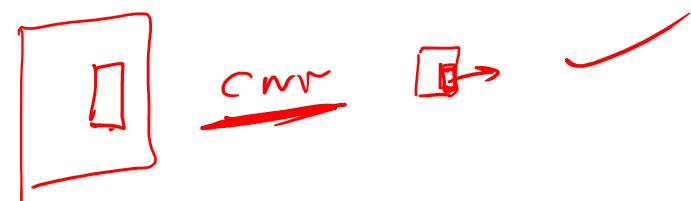
- Region proposals (from external **selective search**)
- Warped image (resizing)
- SVM for classification (one for each class)



# Fast R-CNN



- RoI pooling
- Multi-task loss

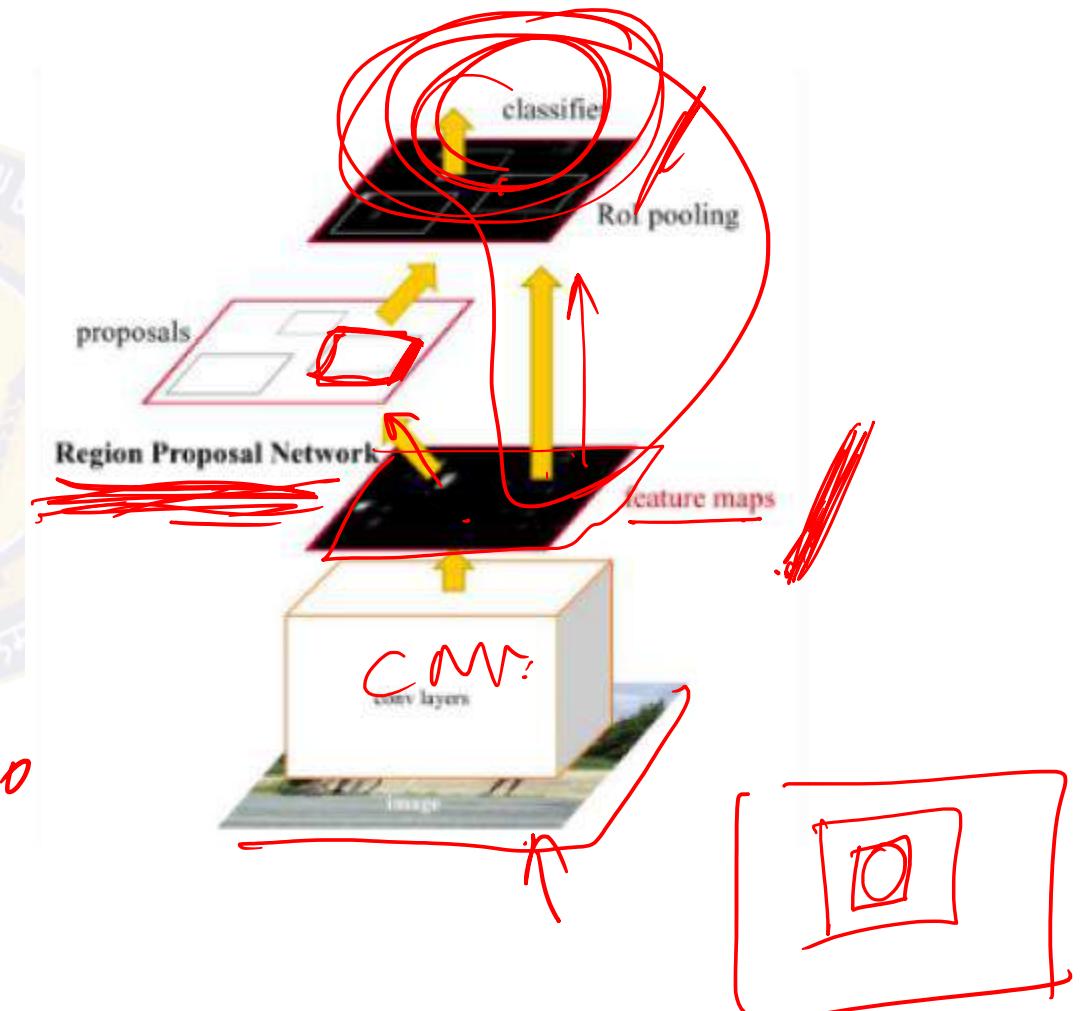


# Faster R-CNN

- Region Proposal Network (RPN)
- Four loss: RPN classification loss, RPN regress loss, Final classification loss, Final box coordinate loss
- 250 time faster than R-CNN. (Fast R-CNN is 25 times fast)

RCNN → 1<sup>st</sup>  
Fast R-CNN → 2<sup>nd</sup>  
faster R-CNN → 250

Cite 7885 Girshick, Ross Fast R-CNN, International Conference on computer vision, pages 1440–1448, IEEE-2015





# Thank You!

Object Detection and Localization (Yolo):



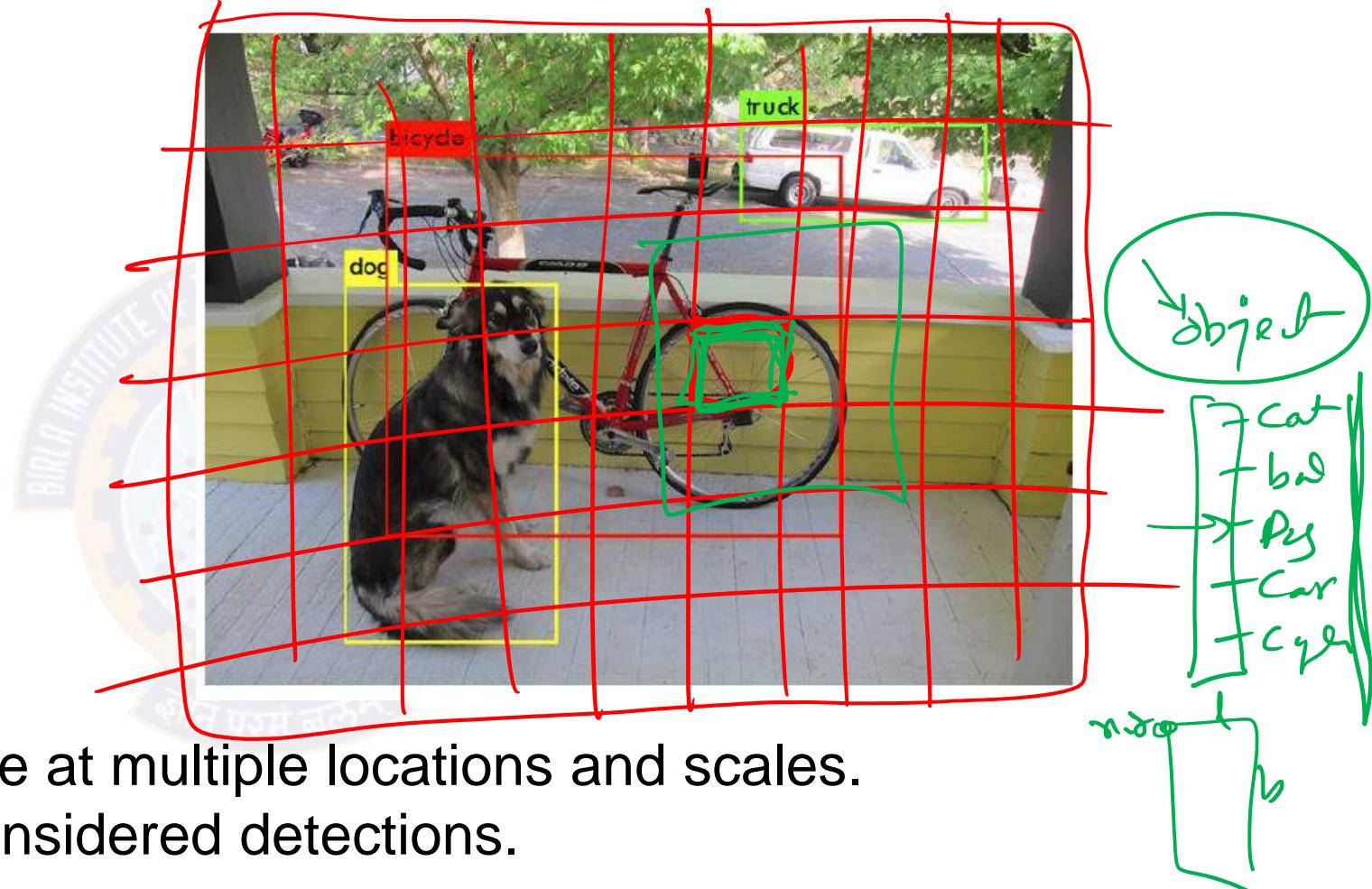
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# CNN Case Studies (Yolo)

**Dr. Kamlesh Tiwari**

# Object Detection with Yolo

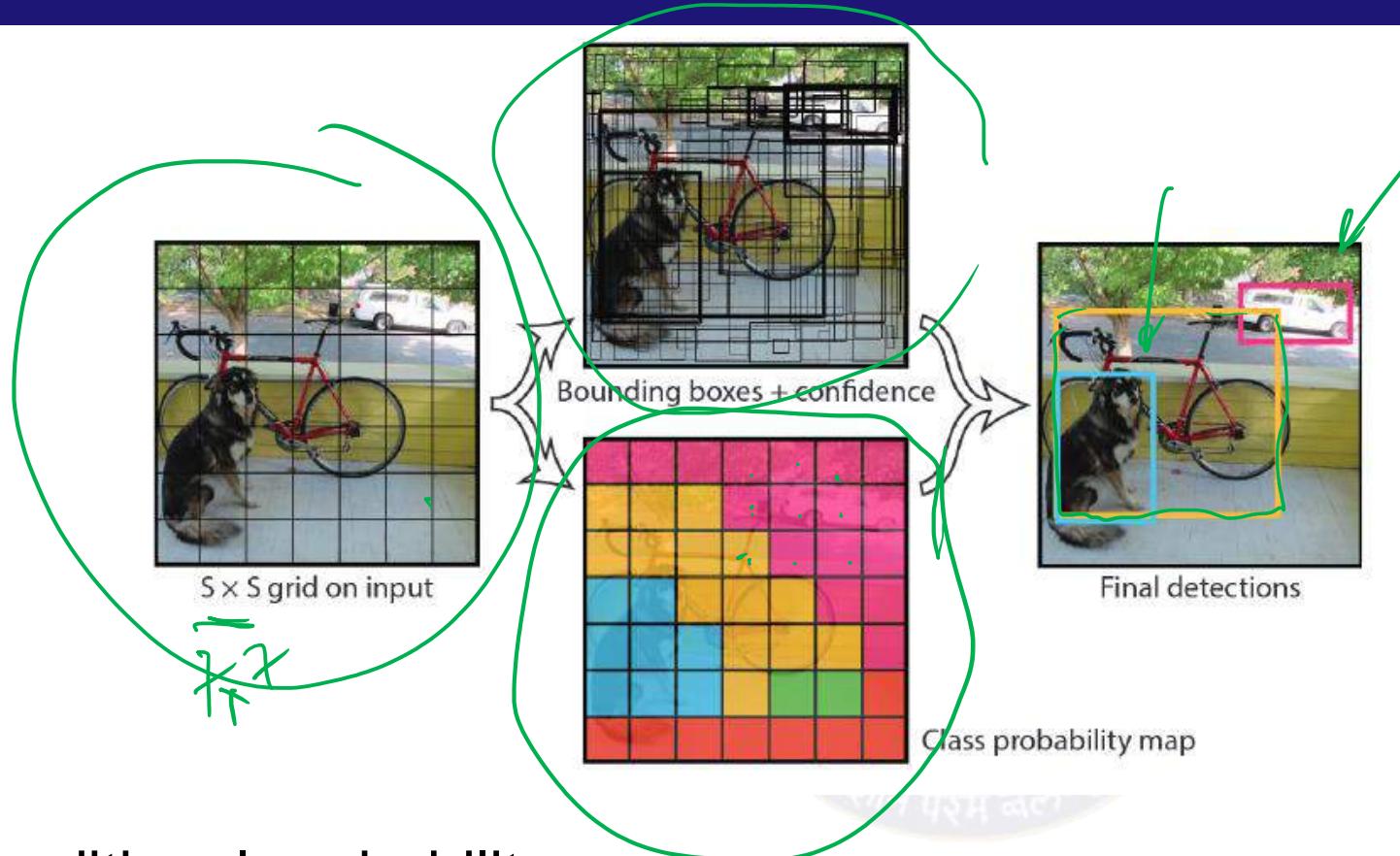
- What is there and where?
- Deformative Part Model, and F-RCNN



- Apply the model to an image at multiple locations and scales.
- High scoring regions are considered detections.

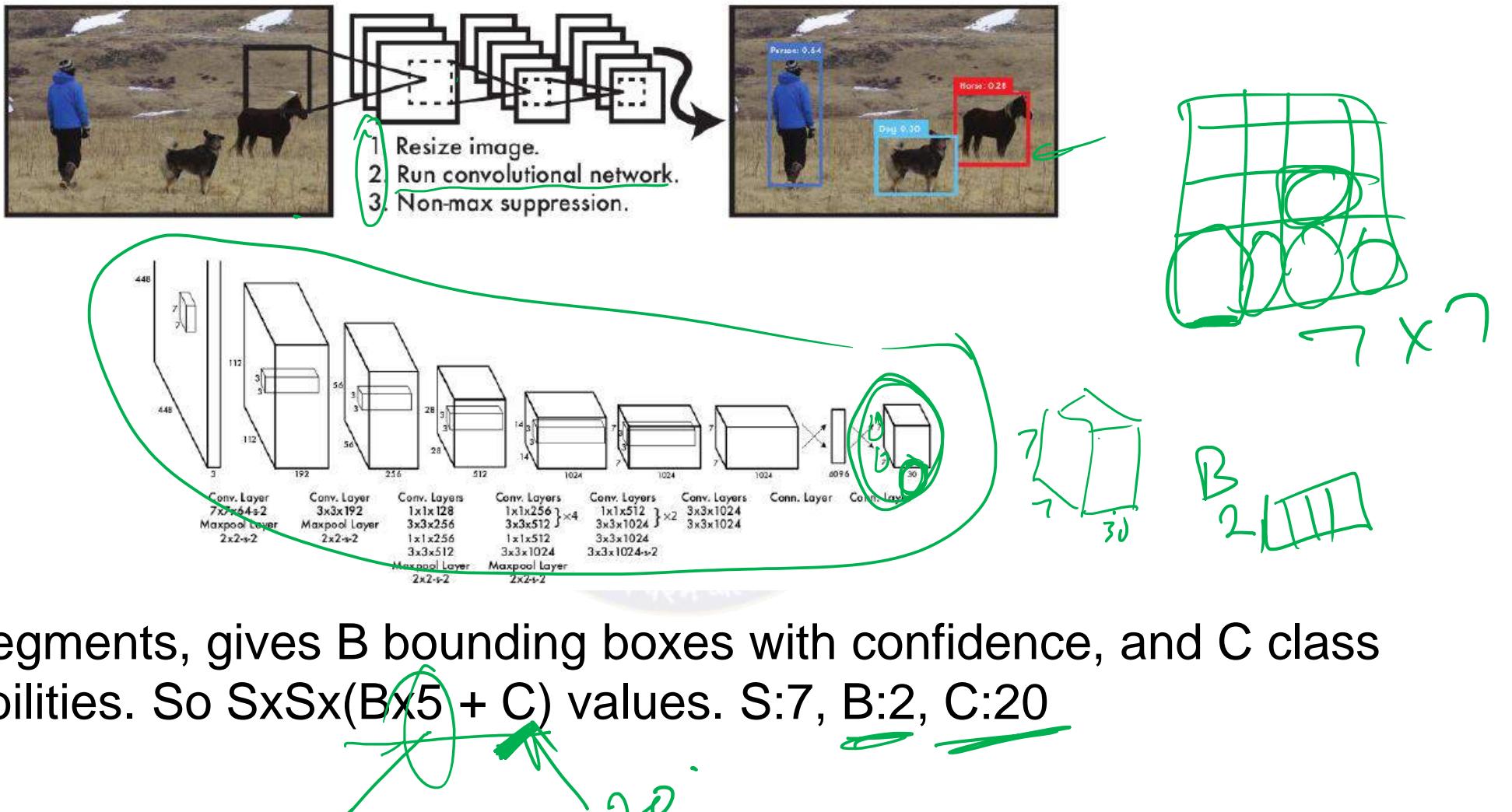
**YOLO:** apply a single neural network to the full image that divides it into regions and predicts bounding boxes and probabilities for each region

# Yolo



- Conditional probability map
- See <https://pjreddie.com/darknet/yolo/>

# Yolo



- SxS segments, gives B bounding boxes with confidence, and C class probabilities. So  $S \times S \times (B \times 5 + C)$  values. S:7, B:2, C:20

# Yolo

Pascal 2007 mAP		Speed
DPM v5	33.7	.07 FPS 14 s/img
R-CNN	66.0	.05 FPS 20 s/img
Fast R-CNN	70.0	.5 FPS 2 s/img
Faster R-CNN	73.2	7 FPS 140 ms/img
YOLO	63.4	45 FPS 22 ms/img

- It is fast
- Speed comes at the price of accuracy. Improved to 69%
- Generalizes well
- Latest version YOLOv3 2018



# Thank You!

Module-5 Autoencoders:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Convolutional Neural Networks (CNN)

**Dr. Sugata Ghosal**

---

[sugata.ghosal@pilani.bits-pilani.ac.in](mailto:sugata.ghosal@pilani.bits-pilani.ac.in)

# Queries

In which layer we should apply dropout(optimization) . Few papers showing dropout on convolution layer wont optimize and few paper showing dropout should be on deep layer only . So what is best approach in applying dropout optimization

In image classification assignment we have been asked to consider input image size to be 64x64x3 . In general terms what is the significance of input image size , large input size like 128x128x3 gives better output or even small input image size like 32x32x3 also gives same result .

In which case we need to convert RGB(color) image to greyscale image . What is the use of conversion and how it is useful .

For following CNN model how to approach:

In college a face detection model was build so that if any student comes to college ,face detection model will get the image from camera setup at entrance door , if model finds the student is detected in college CNN database it opens the door and allow student to enter in class. On one fine day I went to college with full "Beard" look , when I approached entrance camera , the face detection algorithm did not detect me because of full beard look . How to deal this kind of scenario . ( I got this question in CNN interview and interviewer said this is the only information I have ,rest other parameter you can assum if needed )

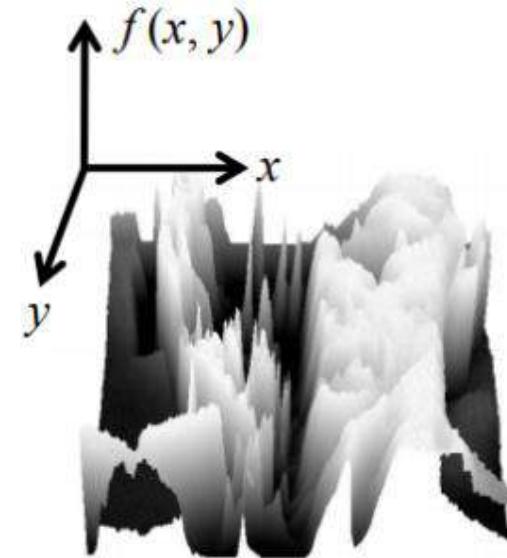
How do we decide the optimal number of CNN layers, filter size or other parameters for a given problem. Is there a scientific way, rather than modifying the parametrs at random?

# Image

Grayscale image can be considered as a function  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$



[snoop](#)



[3D view](#)

- Being digital adds quantization and sampling. We may apply operators on this function



$$g(x, y) = f(x, y) + 20$$



$$g(x, y) = f(-x, y)$$

# Computer Vision - history

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
PROJECT MAC

Artificial Intelligence Group  
Vision Memo. No. 100.

July 7, 1966

THE SUMMER VISION PROJECT  
Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

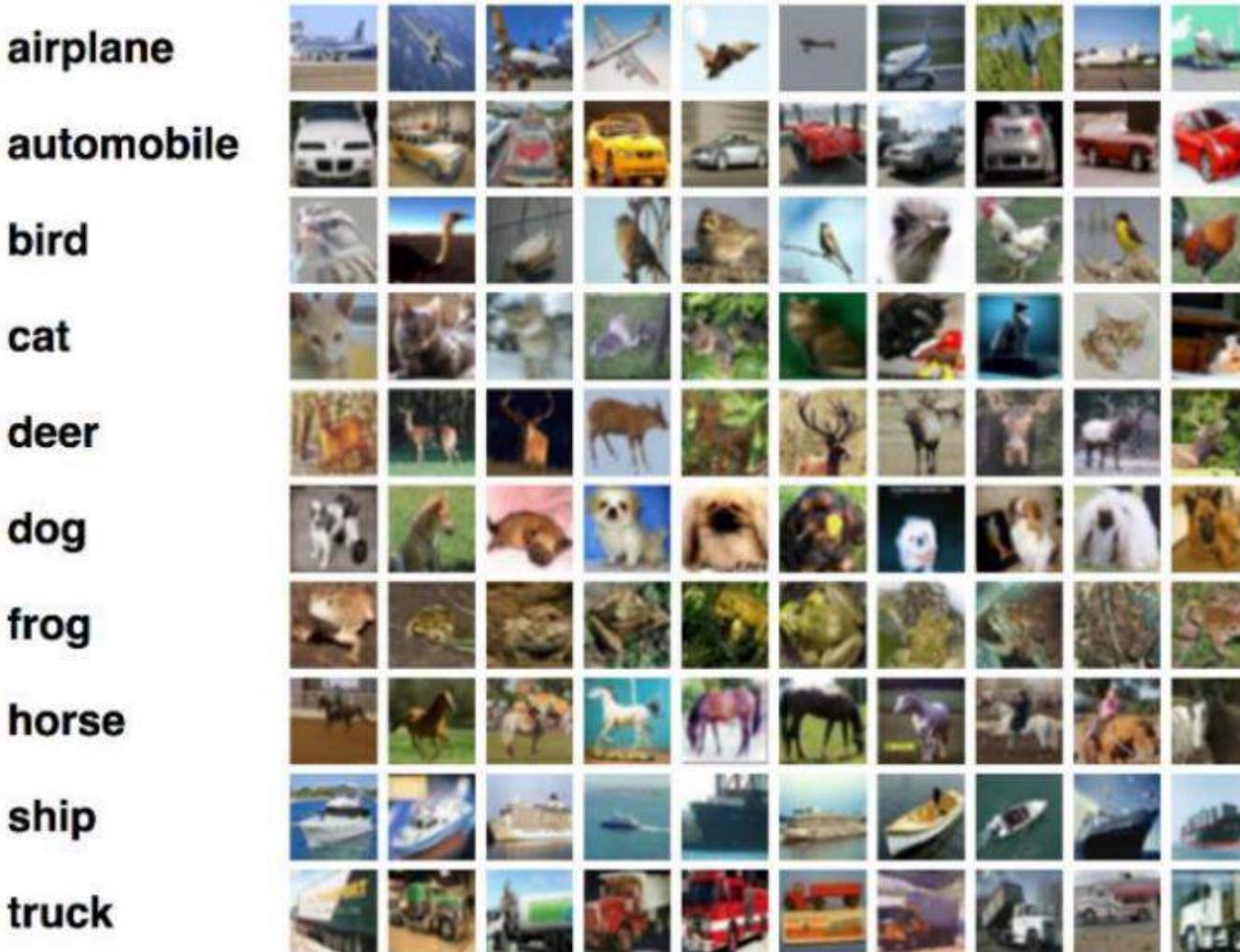
## Some Tasks

- Classification
- Annotation
- Detection
- Segmentation

How the Afghan Girl was Identified by her Iris Patterns 1984-2002



# Classification



# Classification Challenges

Viewpoint variation



Scale variation



Deformation



Occlusion



Illumination conditions



Background clutter



Intra-class variation



# Annotation

					
Predicted keywords	sky, jet, plane, smoke, formation	grass, rocks, sand, valley, canyon	sun, water, sea, waves, birds	water, tree, grass, deer, white-tailed	bear, snow, wood, deer, white-tailed
Human annotation	sky, jet, plane, smoke	rocks, sand, valley, canyon	sun, water, clouds, birds	tree, forest, deer, white-tailed	tree, snow, wood, fox

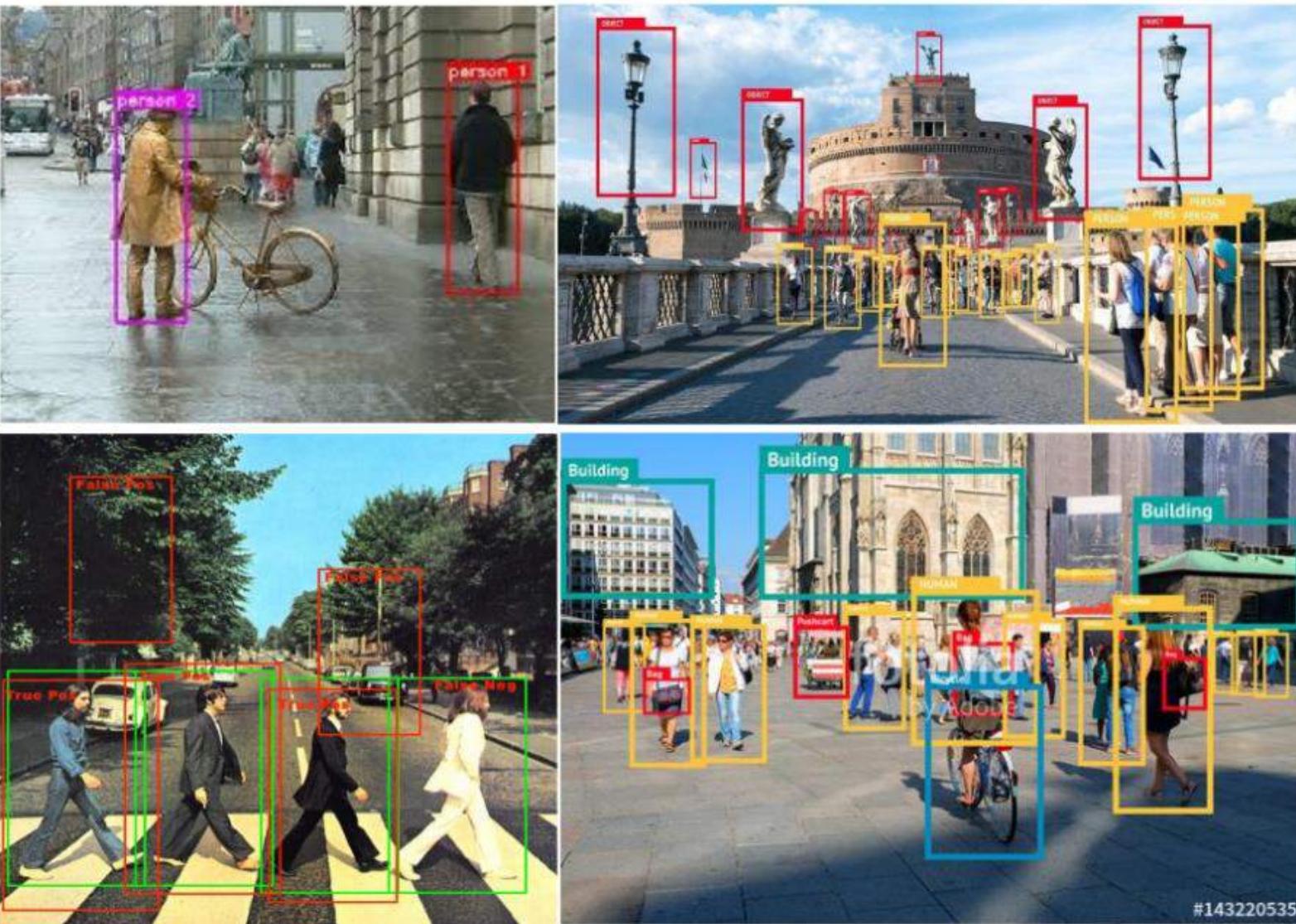


Prediction for queries: Sky (Row1), Street (Row2), Mare (Row3) and Train (Row4)

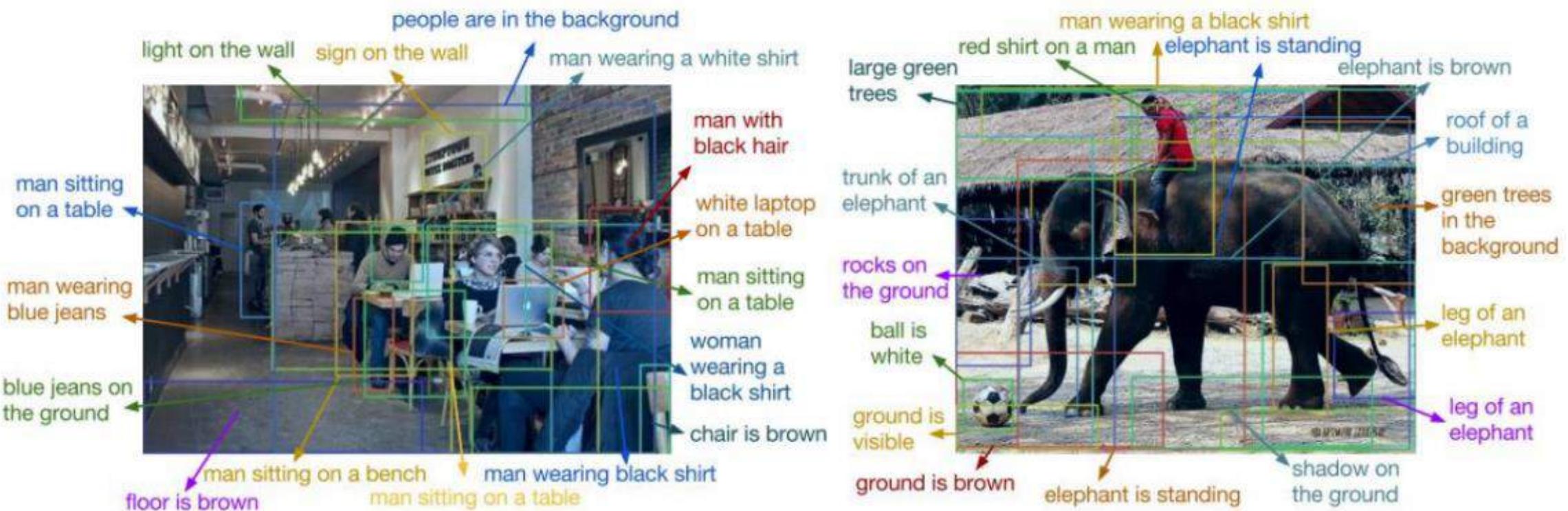
Ref: A. Makadia, V. Pavlovic, and S. Kumar.

A New Baseline for Image Annotation. In Proceedings of ECCV 08.

# Detection



# Description

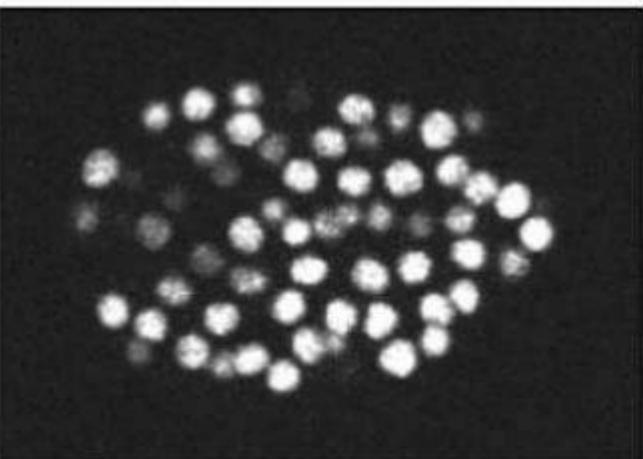


# Segmentation

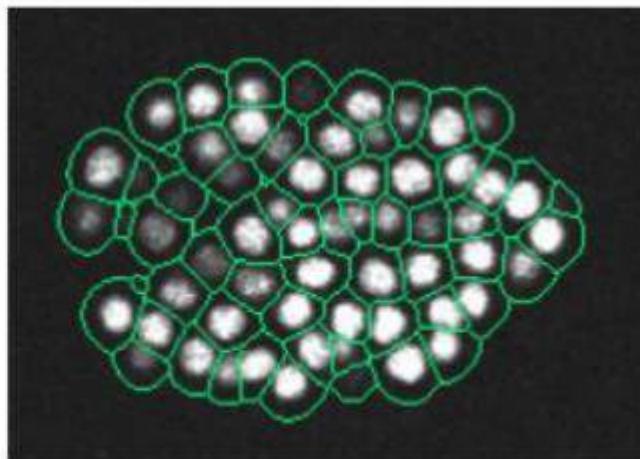


Vijay Badrinarayanan, Alex Kendall and Roberto Cipolla "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." PAMI, 2017.

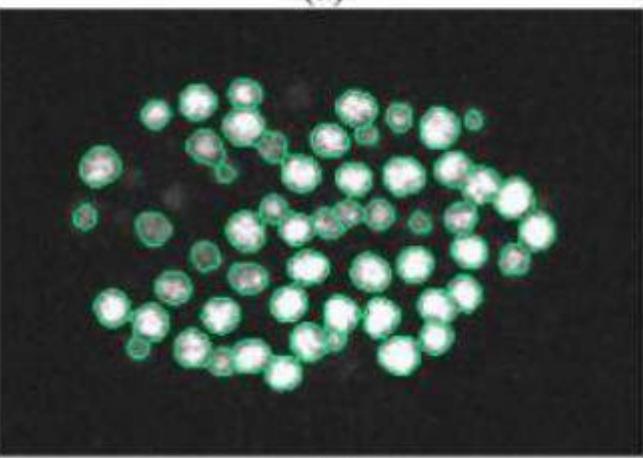
# Segmentation



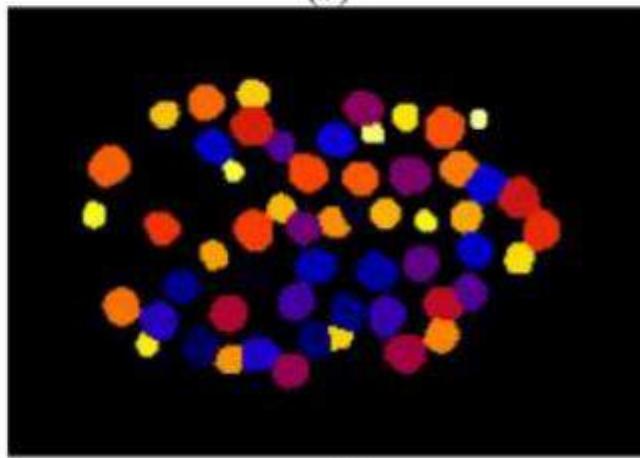
(a)



(b)



(c)



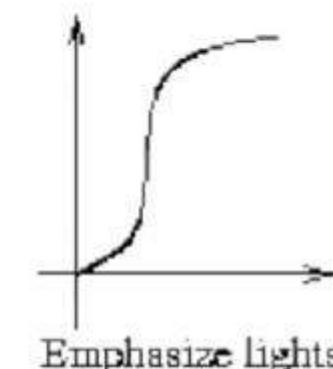
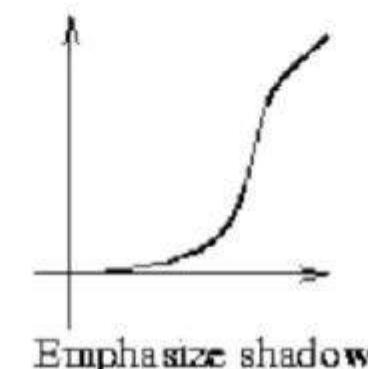
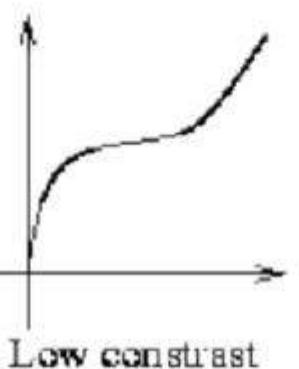
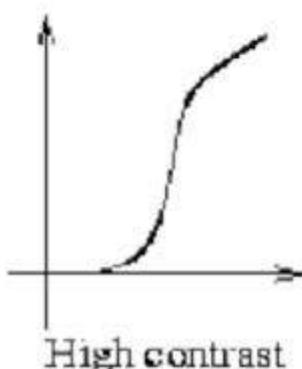
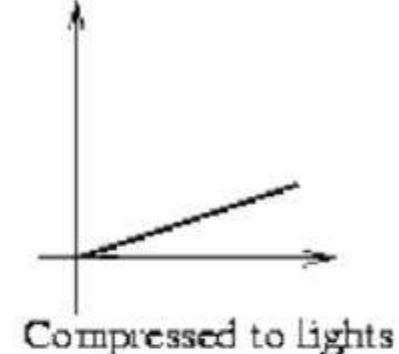
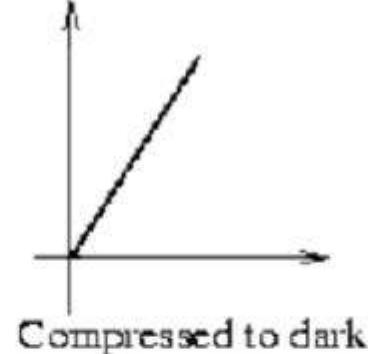
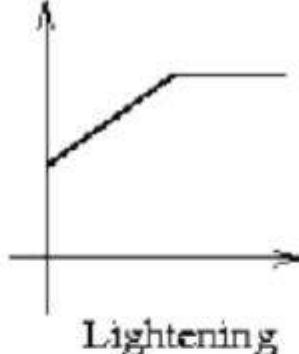
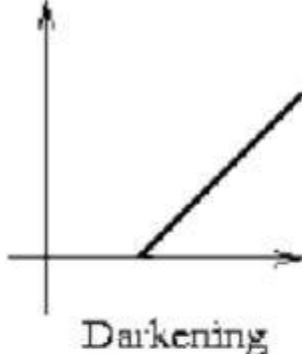
(d)

# Point Operations on Images

Point operation is defined as

$$s = M(r)$$

where r is source pixel intensity and s is destination pixel intensity

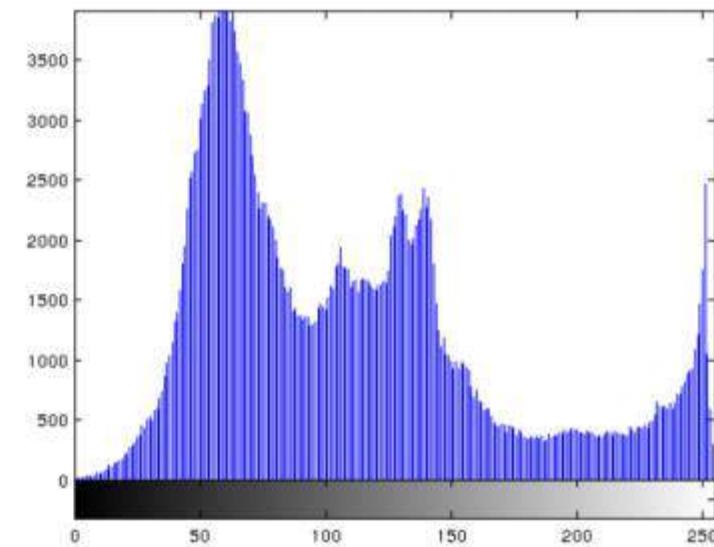


# Histogram

- Discrete probability distribution of the image intensity values



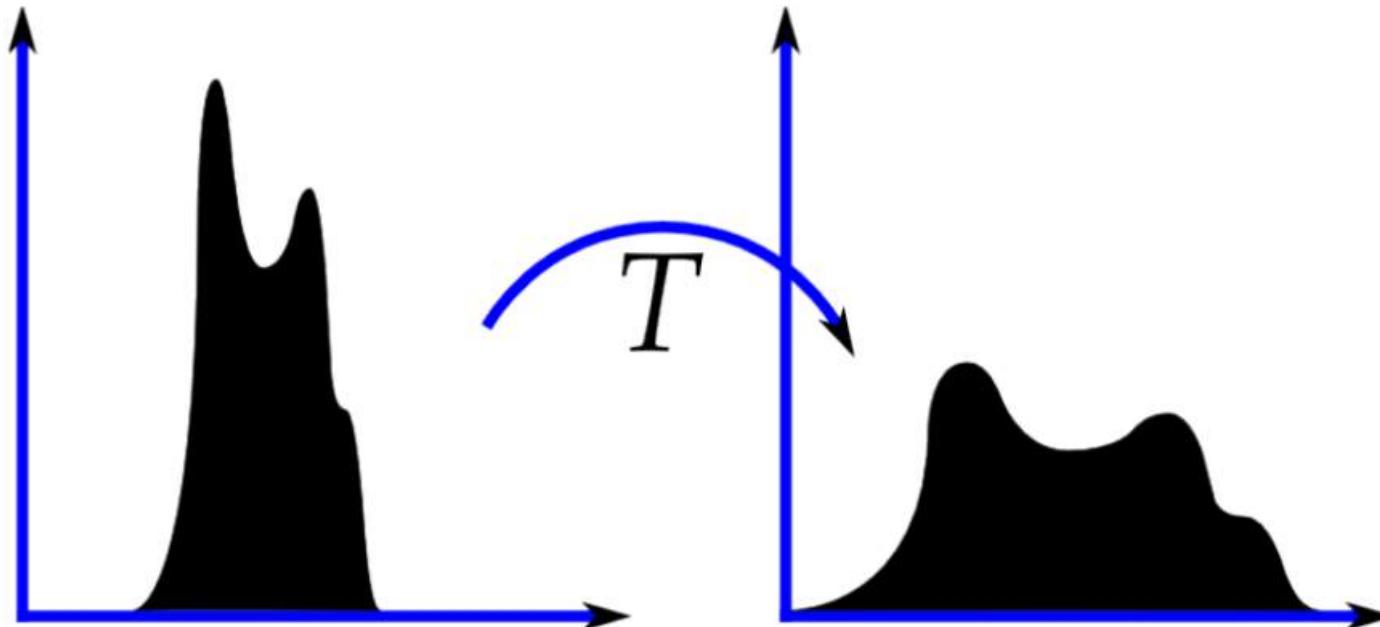
(a) Image



(b) Histogram

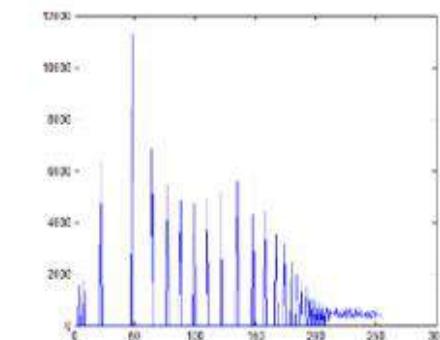
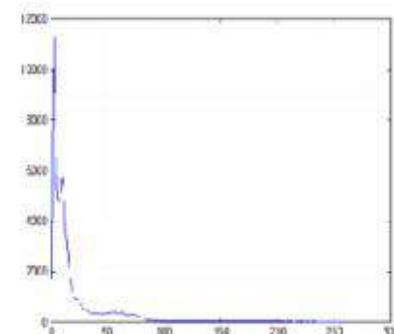
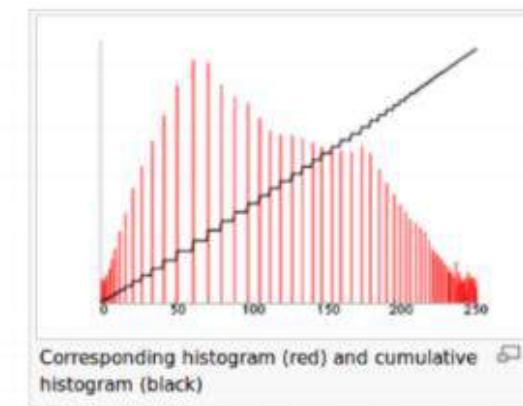
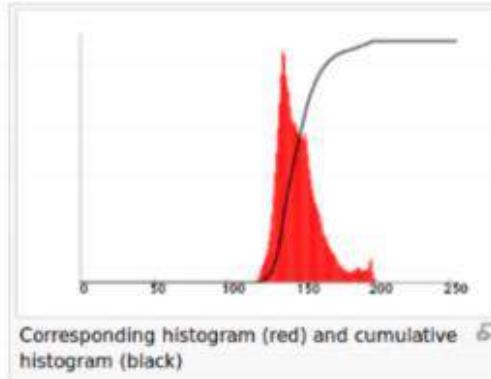
# Histogram Equalization

One method to enhance images is to equalize the histogram



Exercise: Prove that an image transformed by its cumulative distribution function results in an image with uniform histogram. More generally, histogram can be modified by histogram specification

# Histogram Equalization Examples



# Shuffling the pixels

- What happens if we shuffle all the pixels in an image randomly?

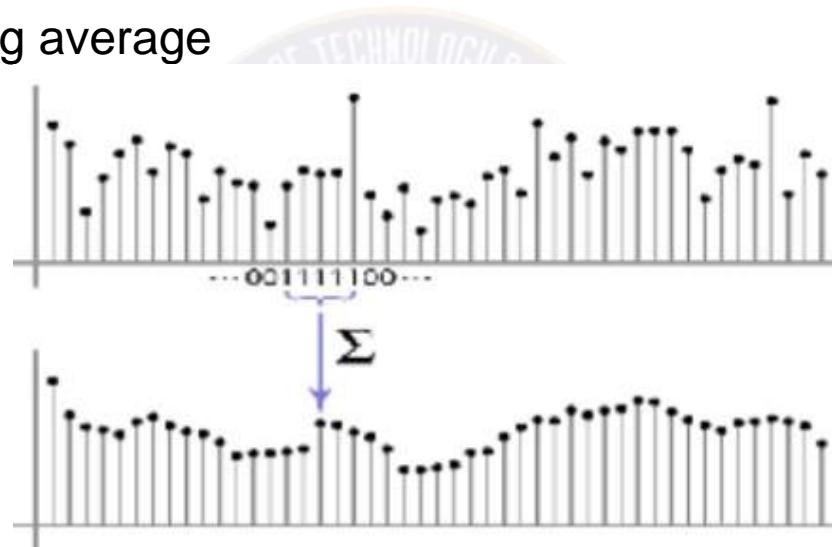


- Different images may have same histogram
- Need more local operators



# First attempt at a solution

- Replace each pixel with an average of all the values in its neighbourhood (assumptions; a) pixels are like neighbour 2) noise is independent of pixel location
- Can add weights to our moving average



- Output pixel is a weighted sum of input pixel values

$$g(i, j) = \sum_{k,l} f(i + k, j + l) h(k, l)$$

- Entries in kernel  $h(k, l)$  are called the filter coefficients. Operator is termed correlation operator.  $g = f \otimes h$

# Convolution

$$g = f \otimes h.$$

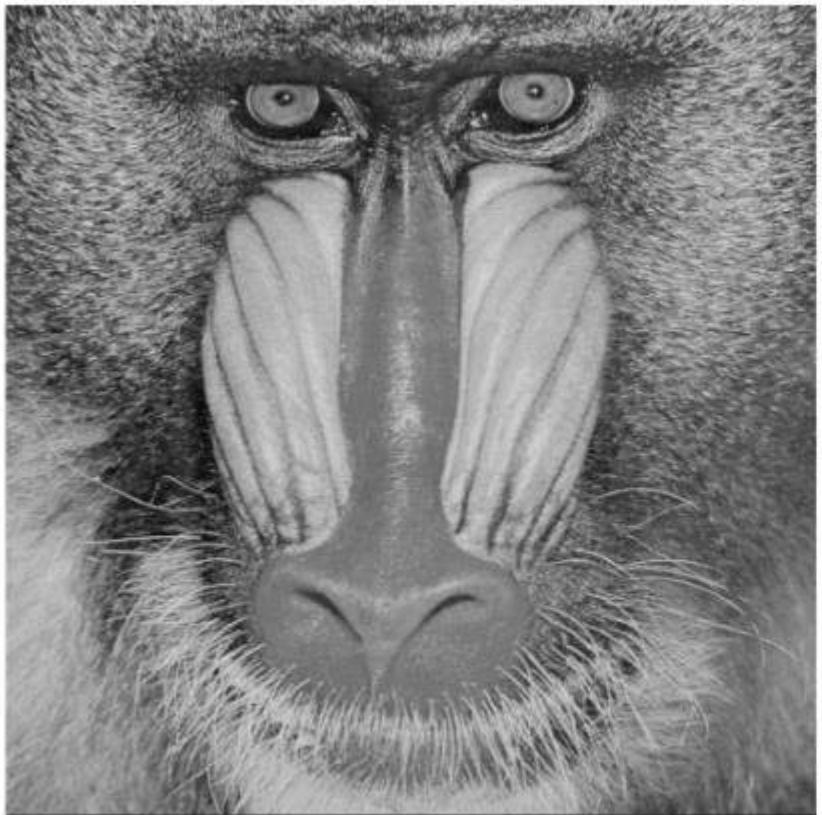
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30	30	20	10		
	0	20	40	60	60	60	40	20		
	0	30	60	90	90	90	60	30		
	0	30	50	80	80	90	60	30		
	0	30	50	80	80	90	60	30		
	0	20	30	50	50	60	40	20		
	10	20	30	30	30	30	20	10		
	10	10	10	0	0	0	0	0		

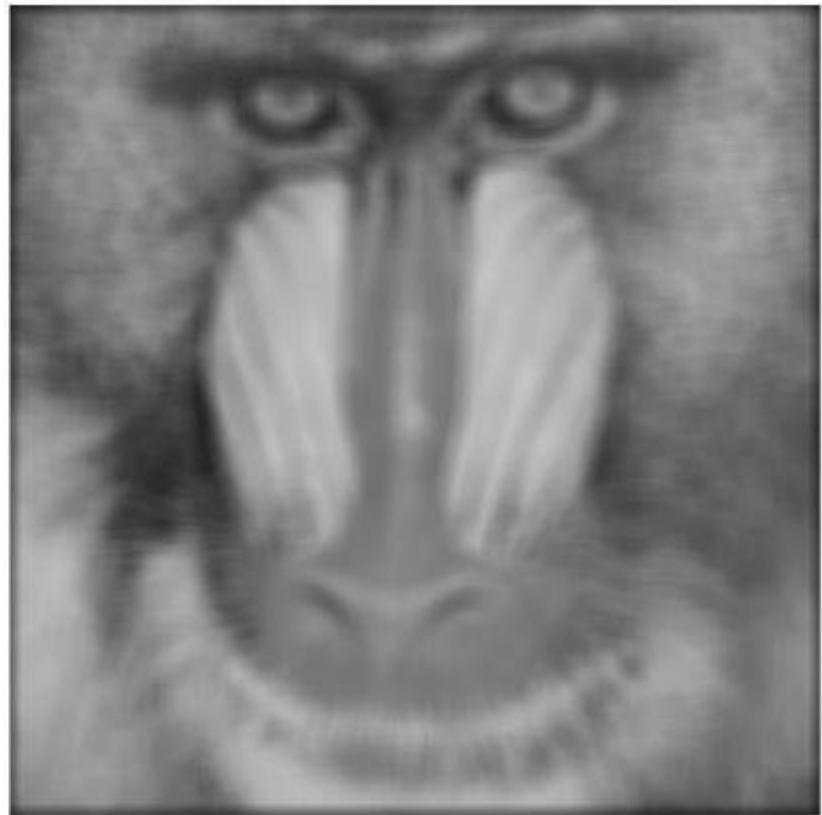
- Convolution is a simple variant of correlation termed as  $g = f * h$

$$g(i, j) = \sum_{k,l} f(i - k, j - l)h(k, l)$$

# Box Filtering



(c)



(d)

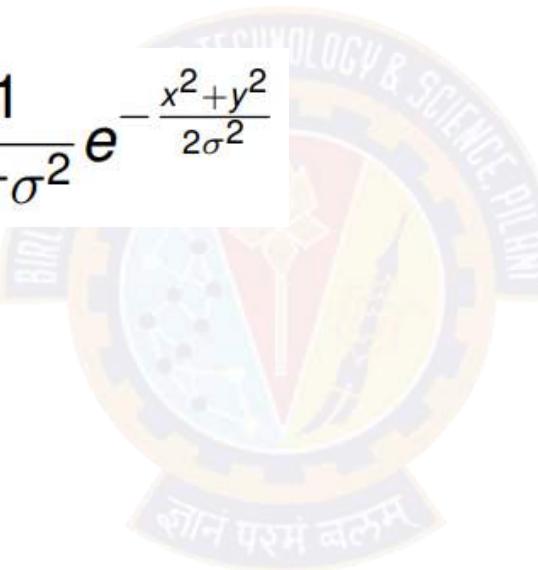
# Gaussian Filtering

Used when we want to have central pixels with more influence.

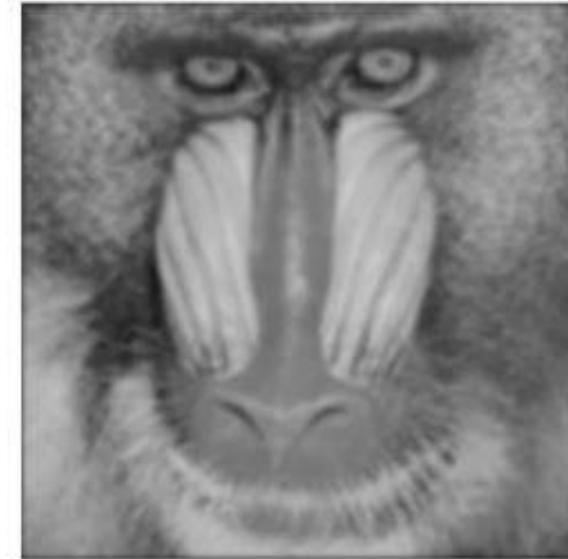
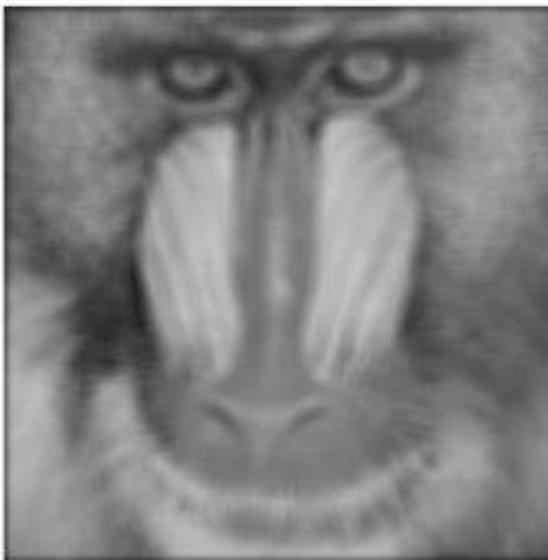
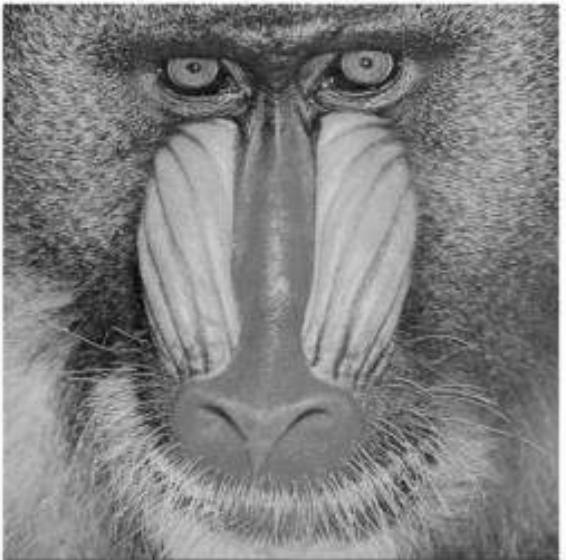
- Gaussian Kernel is defined as

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- It is a low pass filter



# Gaussian Filtering



Gaussian filtering

$$\frac{1}{K^2} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

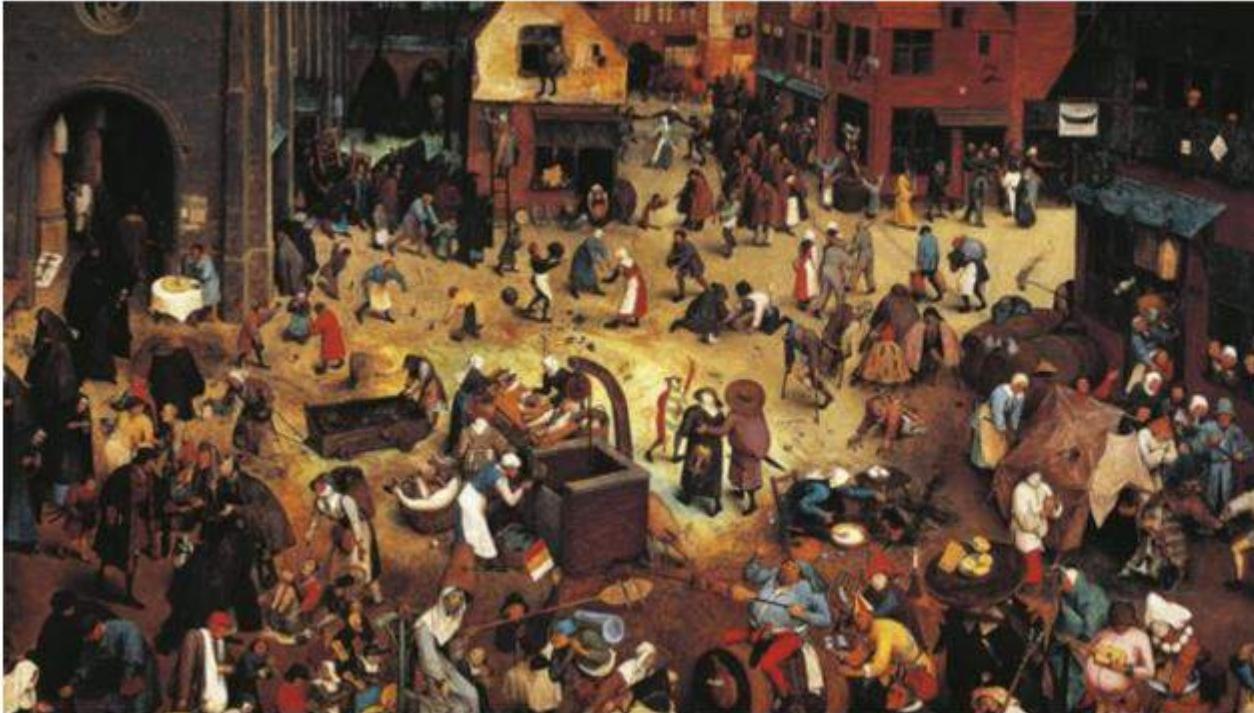
$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

$$\frac{1}{8} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\frac{1}{4} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

# Convolution

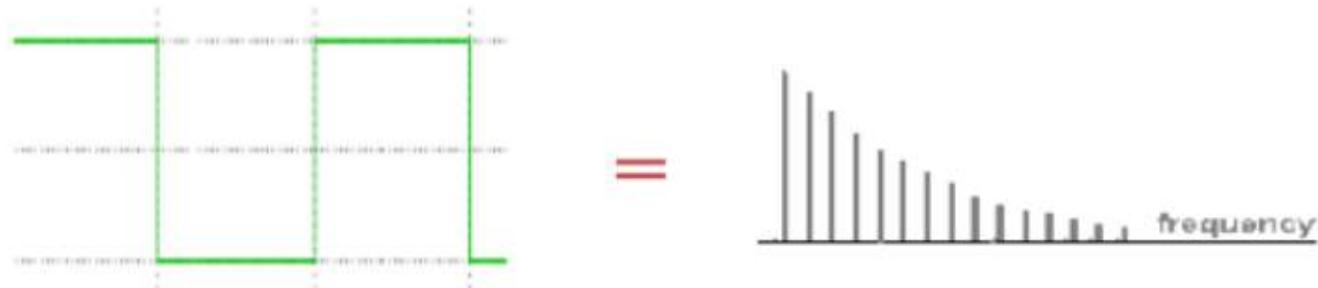


**Filter**

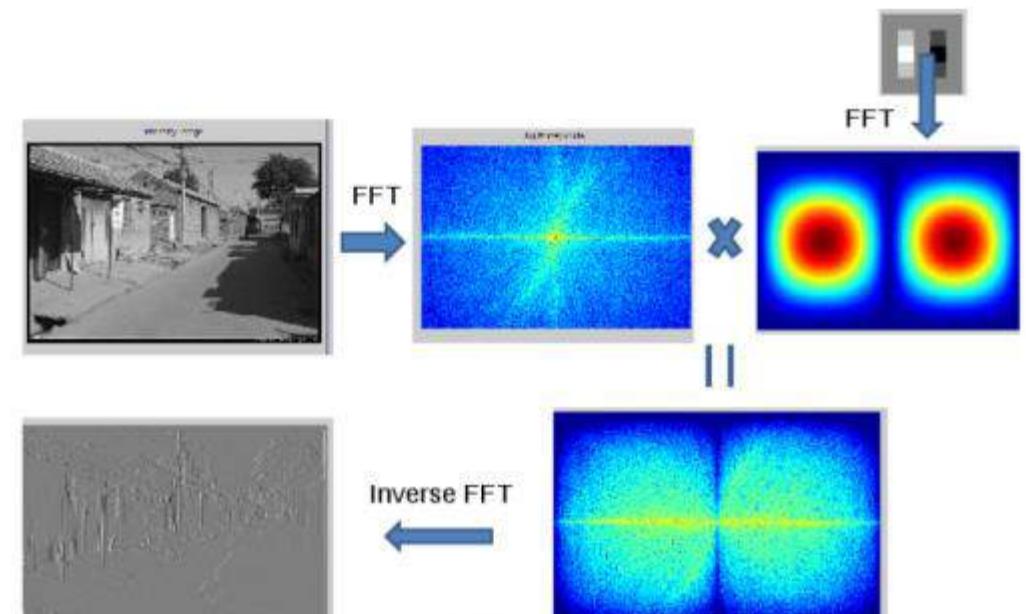
# Fourier Transform

Any signal (function) can be written as sum of various sin and cos waves (terms)

$$A \sum_{k=1}^{\infty} \frac{1}{k} \sin(2\pi kt)$$

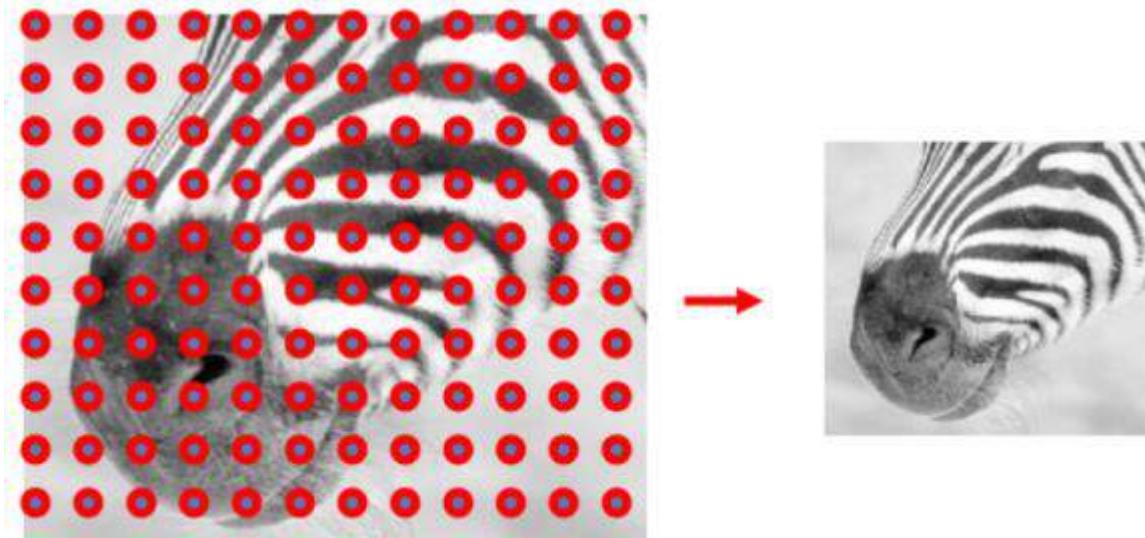


- $F[g \times h] = F[g] \times F[h]$
- $g \times h = F^{-1}[F[g] \times F[h]]$

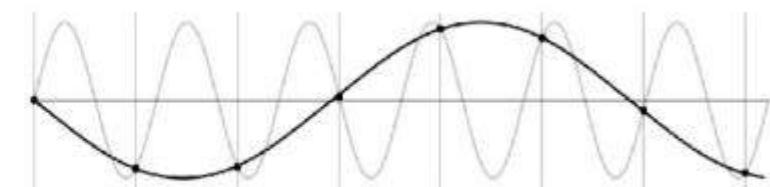
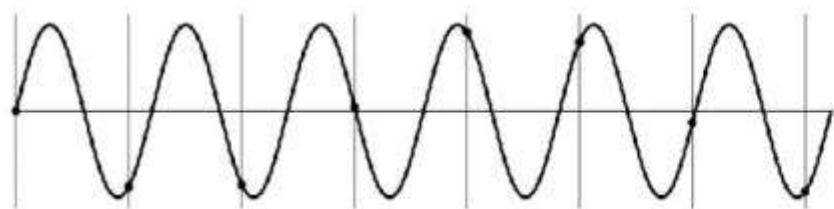


# Sampling

Throw away every other row and column (to get half size image)



Aliasing Problem

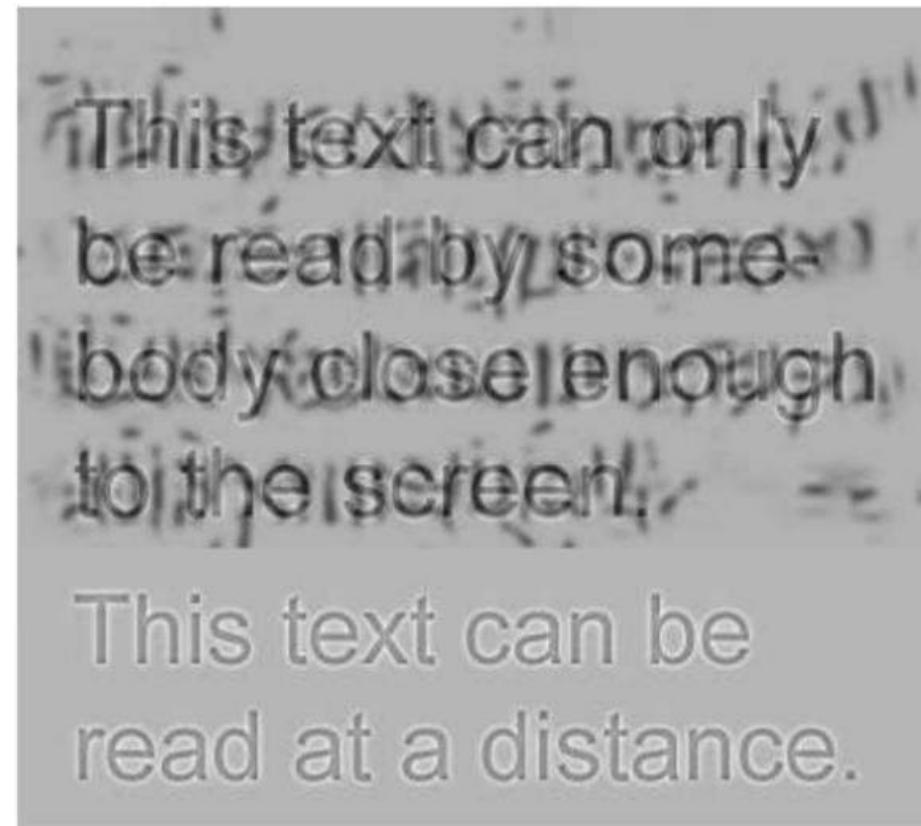


Sub sampling may be dangerous: Funny striped shirt. Wagon wheels rolling in the wrong way in movies.

# Hybrid Images



# Example Hybrid Image



The hybrid font becomes invisible at few meters. The bottom text remains easy to read at relatively long distances.

# Linear filtering

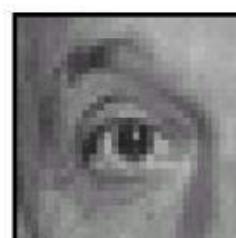


Original

$$\text{Original} * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \text{Identical image}$$

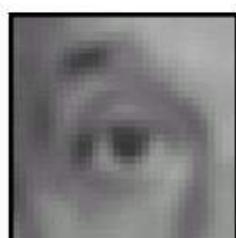


Identical image



Original

$$\text{Original} * \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \text{Blur (with a mean filter)}$$



Blur (with a mean filter)

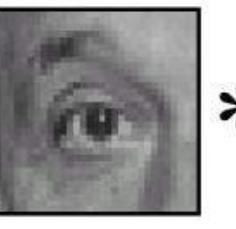


Original

$$\text{Original} * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \text{Shifted left By 1 pixel}$$



Shifted left By 1 pixel

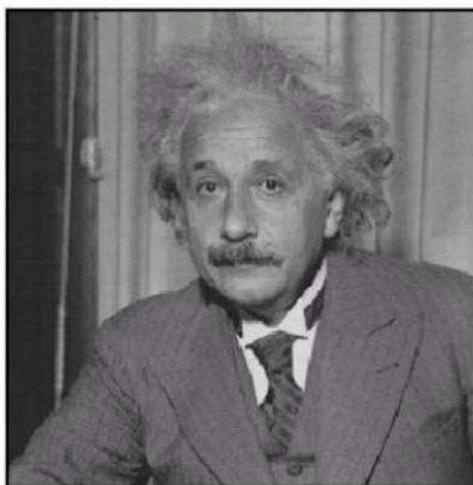


Original

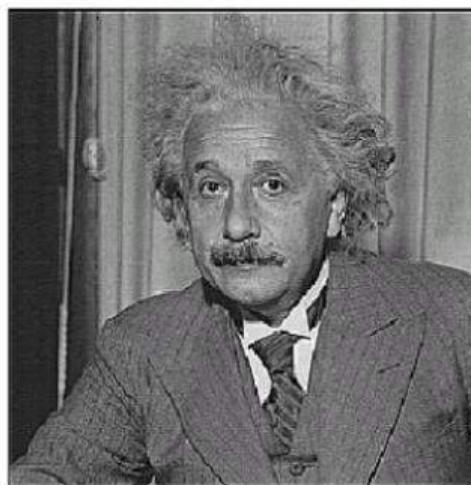
$$\text{Original} * \left( \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) = \text{Sharpening filter (accentuates edges)}$$



Sharpening filter  
(accentuates edges)



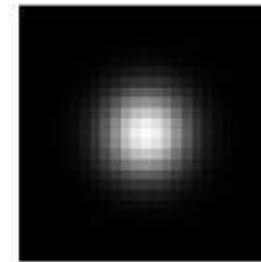
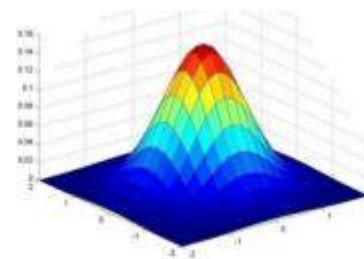
before



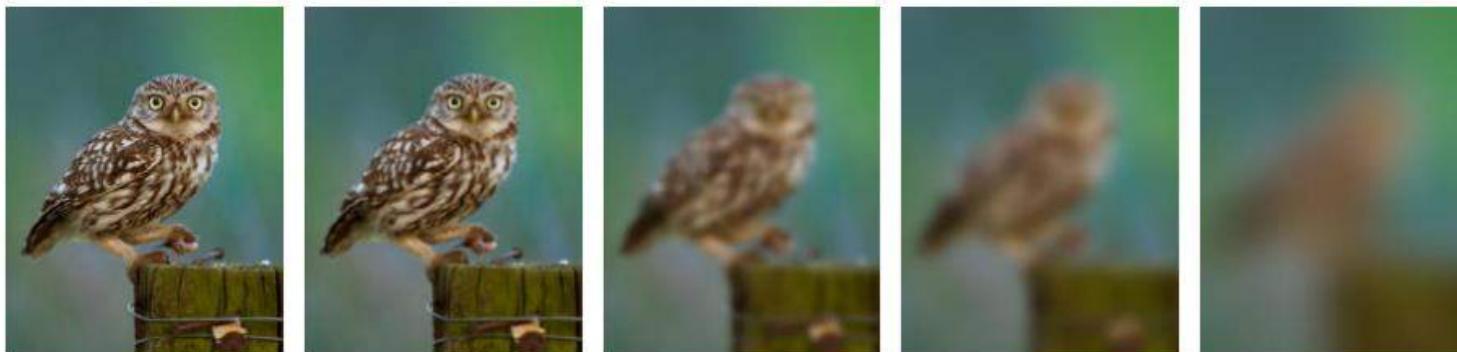
after

# Gaussian filter

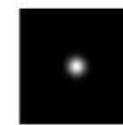
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



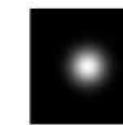
Act as low-pass filter as it removes high-frequency components. Convolution with self is another Gaussian.



$\sigma = 1$  pixel



$\sigma = 5$  pixels



$\sigma = 10$  pixels

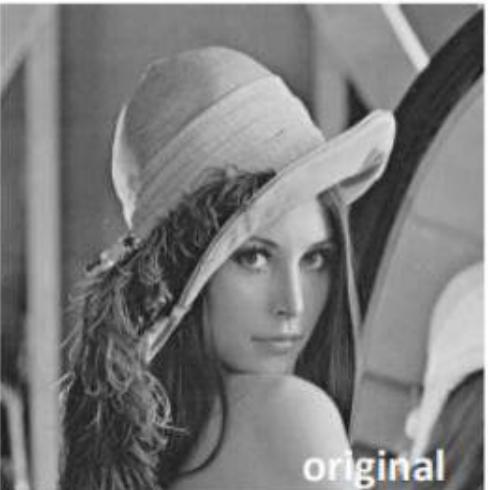


$\sigma = 30$  pixels

Convolving two times with Gaussian kernel of  $\sigma$  width is equivalent to Convoluting once with Gaussian kernel of  $\sqrt{2}\sigma$  width

# Sharpening

What does blurring take away?



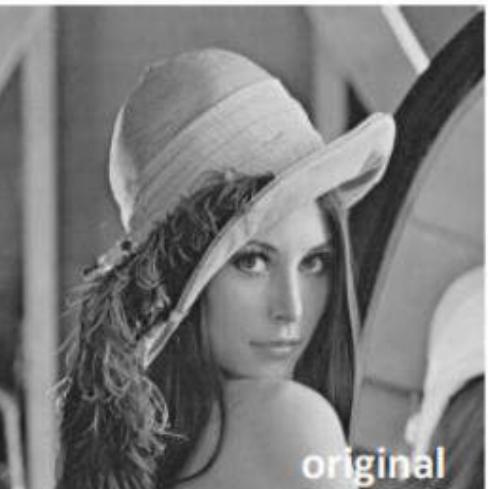
-



=



Let's add it back:



$+ \alpha$



=



# Edge detection

Common approximation of derivative of Gaussian

$$\frac{1}{8} \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

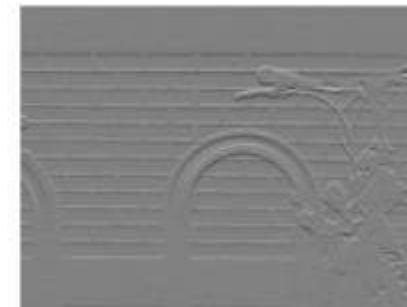
$s_x$

$$\frac{1}{8} \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

$s_y$

Edge magnitude is  $\sqrt{s_x^2 + s_y^2}$  and the direction is  $\tan^{-1}(s_y/s_x)$

Edge is detected by using threshold



# Motivation to Convolution

Suppose you have a noisy sensor to have a measurement

How could you estimate the actual reading

1. Average of some readings
2. in local neighbourhood
3. What about weighted average?



$$s_t = \sum_{i=0}^{\infty} x_{t-i} w_i$$

	$w_{-6}$	$w_{-5}$	$w_{-4}$	$w_{-3}$	$w_{-2}$	$w_{-1}$	$w_0$
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90
S							

1.80					
------	--	--	--	--	--

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

Here things are in one dimension only.

# Motivation to Convolution

Suppose you have a noisy sensor to have a measurement

How could you estimate the actual reading

1. Average of some readings
2. in local neighbourhood
3. What about weighted average?



$$s_t = \sum_{i=0}^{\infty} x_{t-i} w_i$$

	$w_{-6}$	$w_{-5}$	$w_{-4}$	$w_{-3}$	$w_{-2}$	$w_{-1}$	$w_0$
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90
S							

1.80					
------	--	--	--	--	--

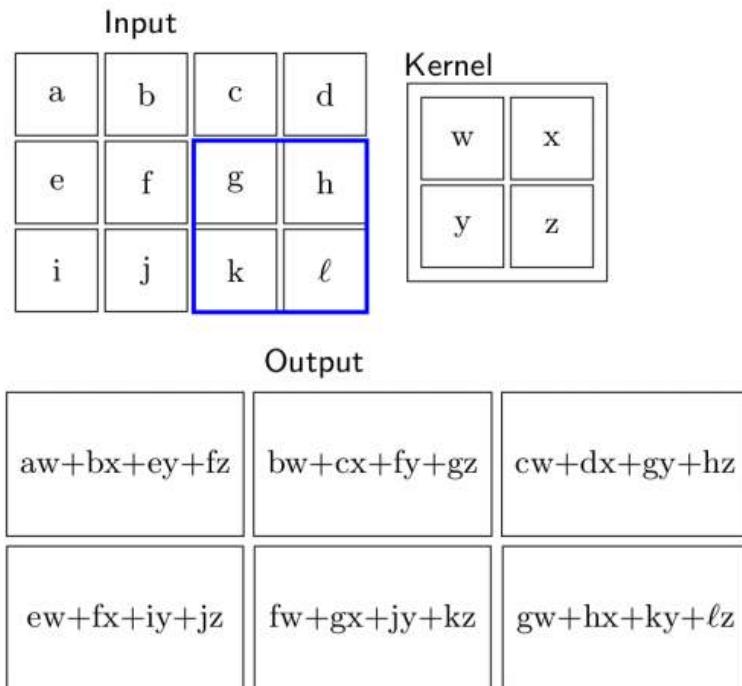
$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

Here things are in one dimension only.

# Image is a 2D signal



$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a, b}$$



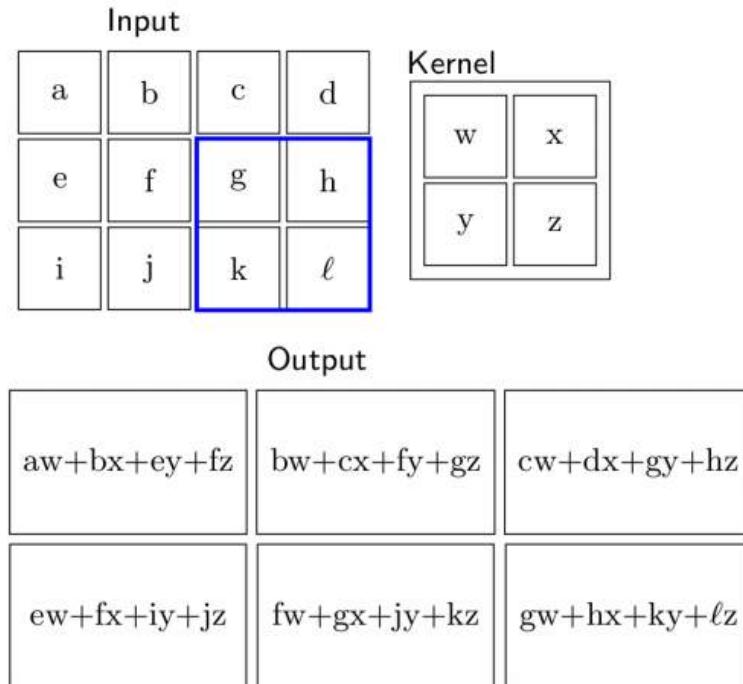
## General formula

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a, j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$

# Image is a 2D signal



$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a, b}$$



## General formula

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a, j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$

# Image is a 2D signal



$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$



blurs the image

$$* \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} =$$



sharpens the image

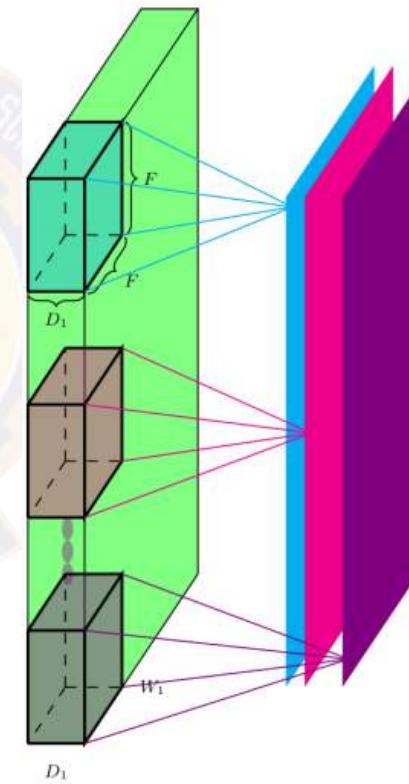
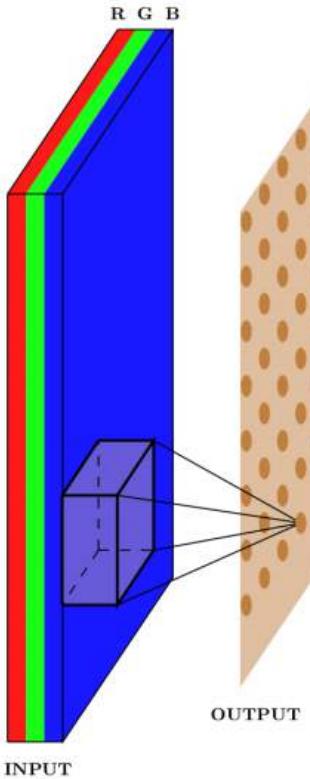
$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$



detects the edges

# Filter in 3D

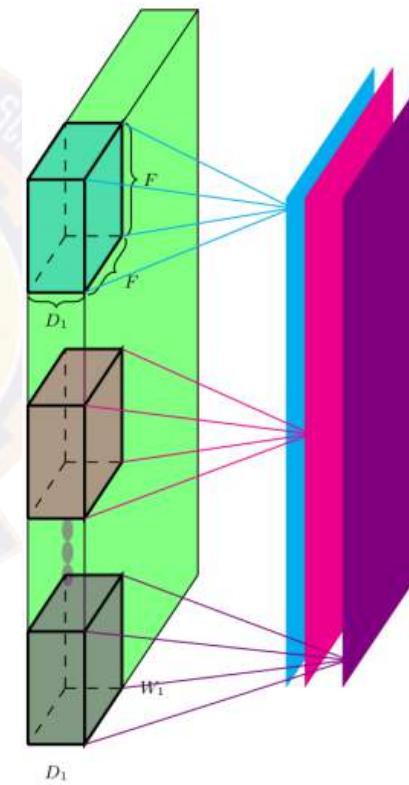
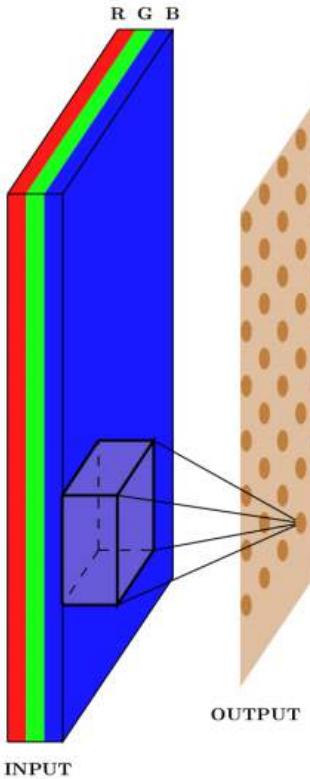
- It would refer to volume. Assume the filter extends to the depth
- Output is 2D



- Apply multiple filters to get multiple feature maps

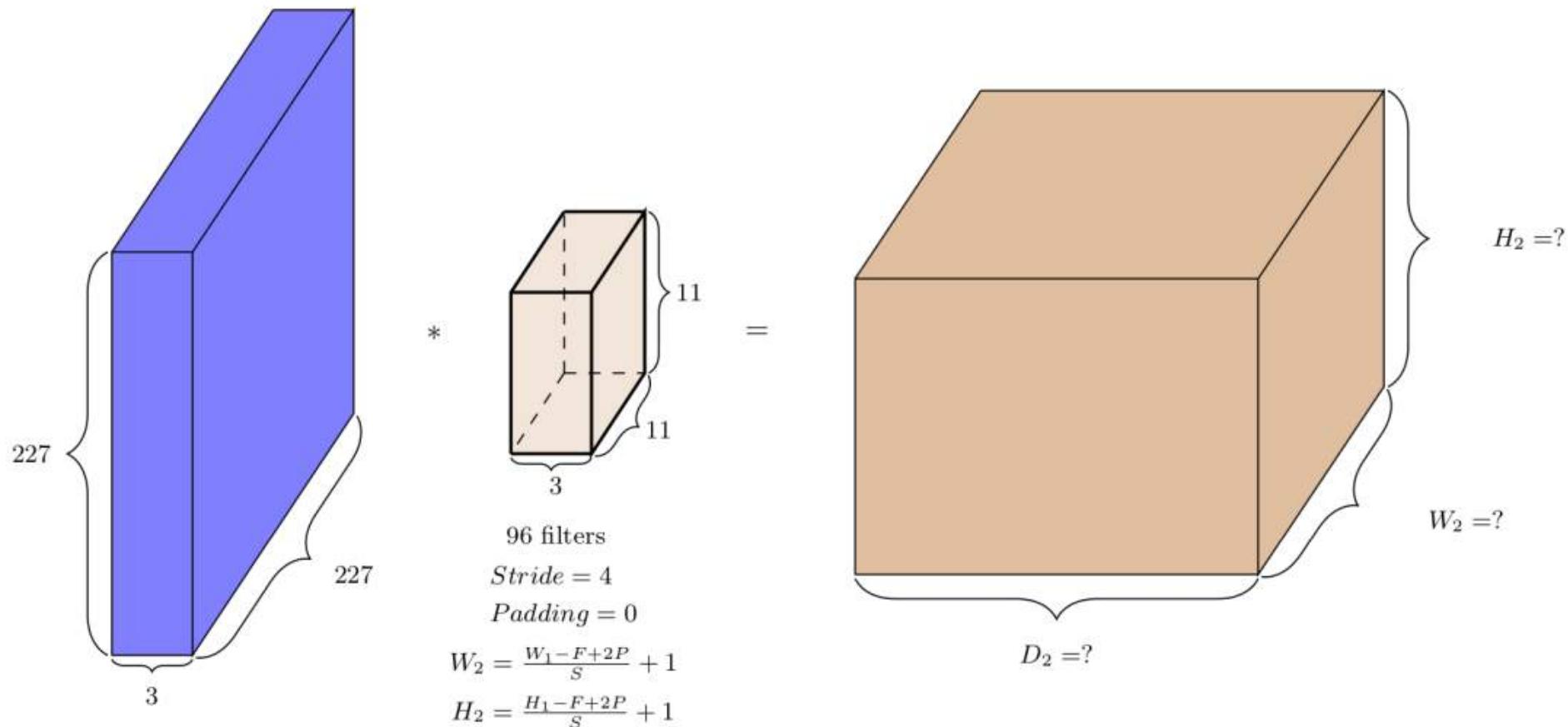
# Filter in 3D

- It would refer to volume. Assume the filter extends to the depth
- Output is 2D



- Apply multiple filters to get multiple feature maps

# Filters, Padding and Stride



$$H, W = \frac{H - F + 2P}{S} + 1, \frac{W - F + 2P}{S} + 1$$

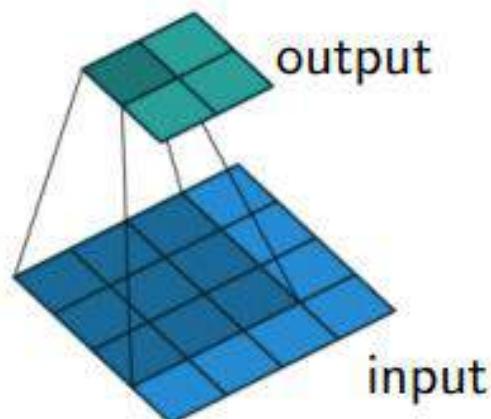
# Size Before and After Convolutions

Feature map size:

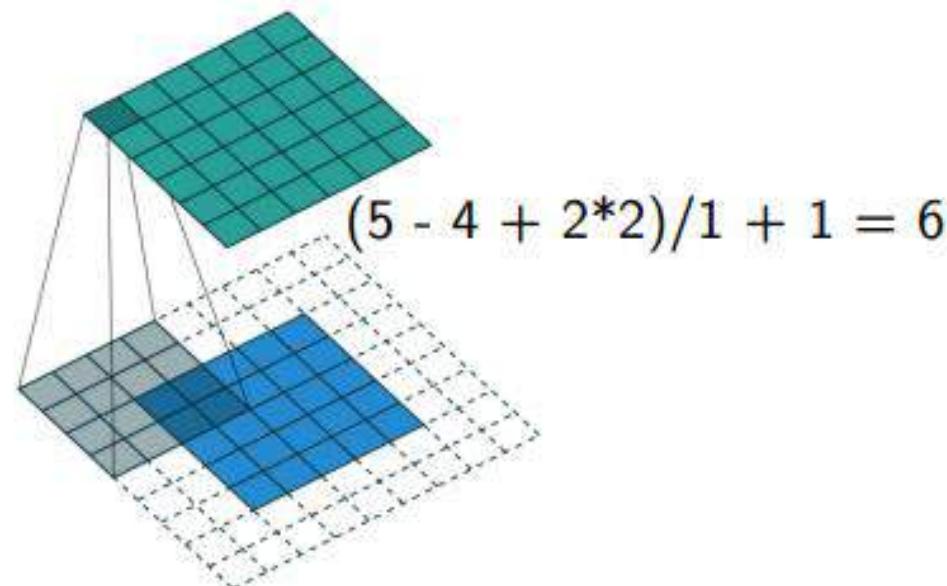
$$O = \frac{W - K + 2P}{S} + 1$$

The diagram illustrates the formula for calculating the output width of a feature map after a convolution. The formula is  $O = \frac{W - K + 2P}{S} + 1$ . The input width is labeled  $W$ , kernel width is  $K$ , padding is  $P$ , output width is  $O$ , and stride is  $S$ .

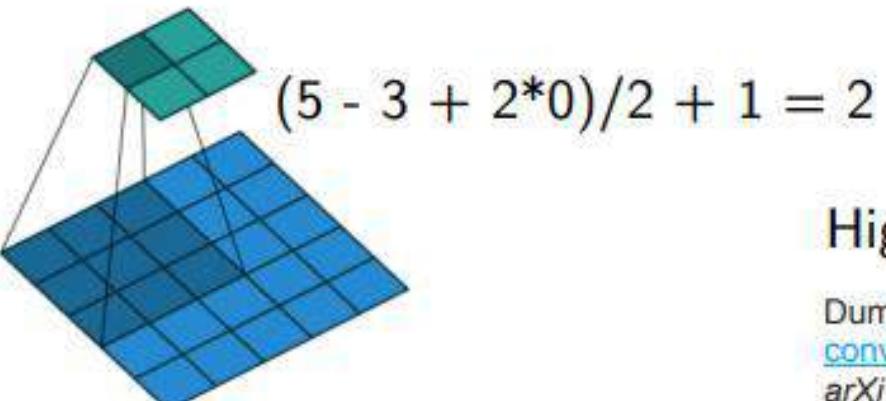
$$(4 - 3 + 2*0)/1 + 1 = 2$$



No padding, stride=1



padding=2, stride=1



No padding, stride=2

Highly recommended:

Dumoulin, Vincent, and Francesco Visin. "[A guide to convolution arithmetic for deep learning](#)." *arXiv preprint arXiv:1603.07285* (2016).

# Convolutions with Color Channels

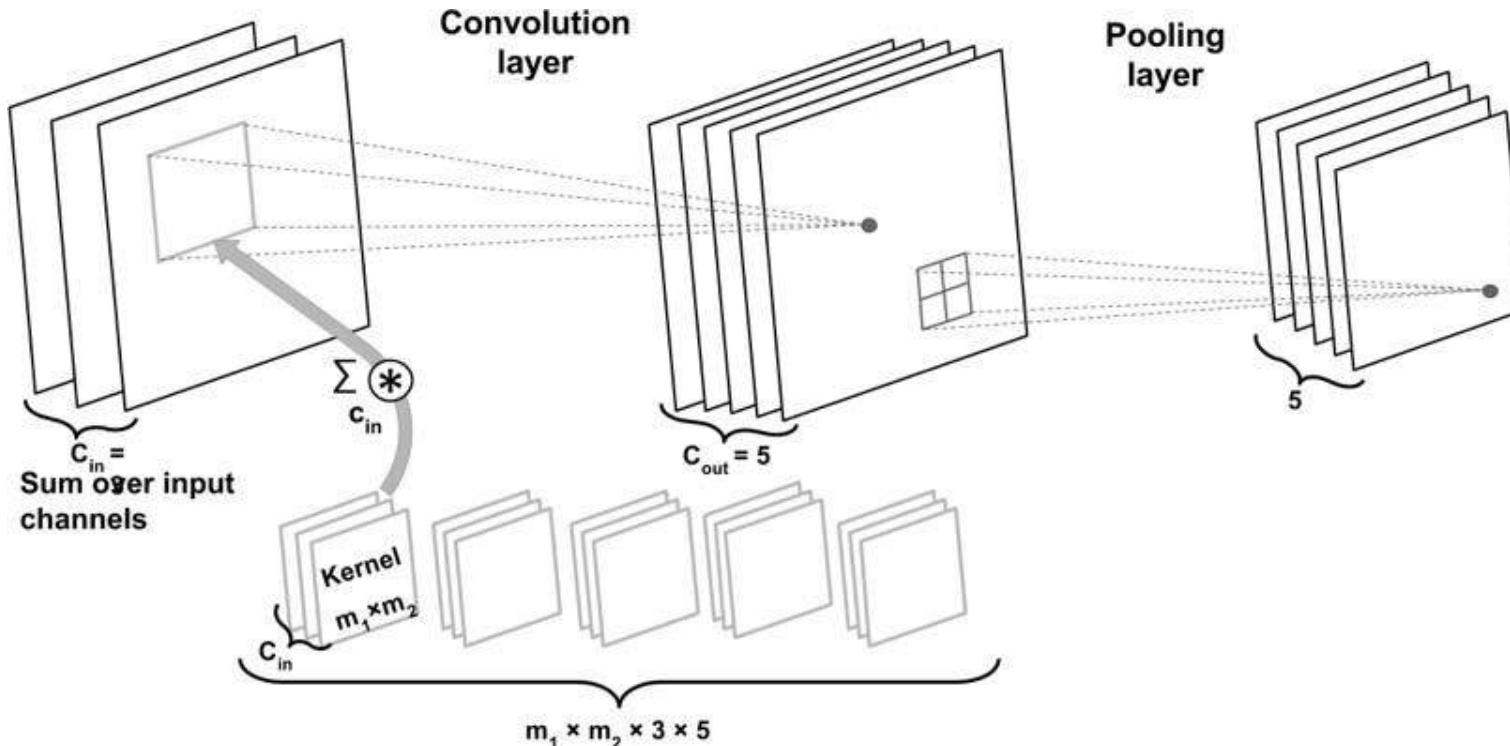
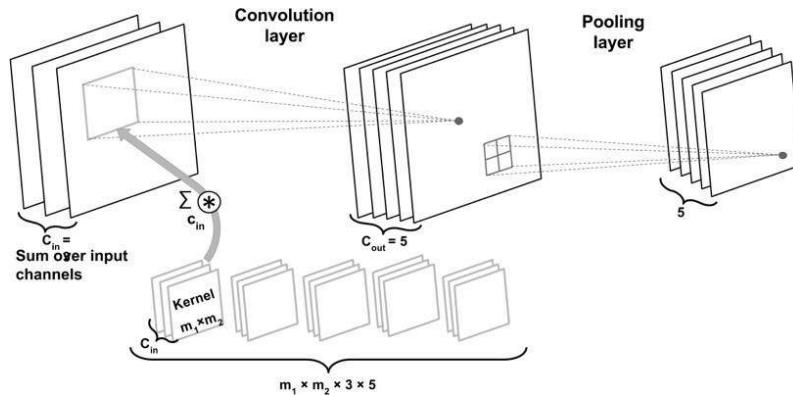


Image dimensions:  $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times c_{in}}$

Kernel dimensions:  $\mathbf{W} \in \mathbb{R}^{m_1 \times m_2 \times c_{in} \times c_{out}}$     $\mathbf{b} \in \mathbb{R}^{c_{out}}$

# Convolutions with Color Channels



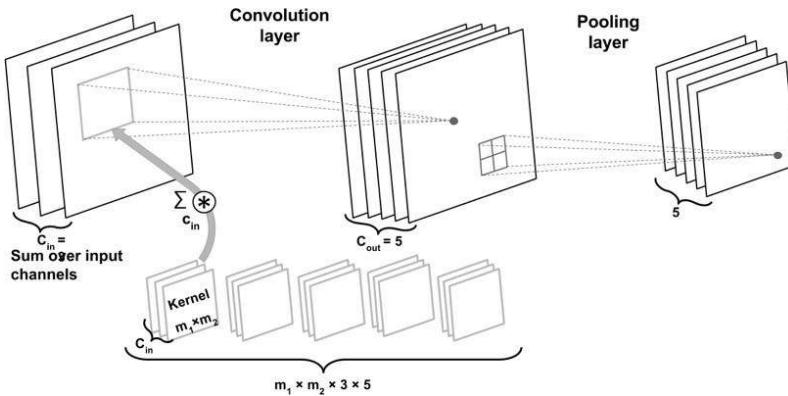
Number of parameters:

$$m_1 \times m_2 \times 3 \times 5 + 5$$

Assume 5x5 kernel:

$$5 \times 5 \times 3 \times 5 + 5 = 380$$

# Convolutions with Color Channels



Assume "same" padding  
(more on padding later),  
such that the hidden layer has  
the same height and width as  
the original images

If we use a CNN:

Number of parameters:

$$m_1 \times m_2 \times 3 \times 5 + 5$$

Assume 128x128 images and  
5x5 kernel:

$$5 \times 5 \times 3 \times 5 + 5 = 380$$

If we use a fully connected layer:

Number of parameters:

$$(n_1 \times n_2 \times 3) \times (n_1 \times n_2 \times 5) + (n_1 \times n_2 \times 5)$$

$$\begin{aligned} & (128 \times 128)^2 \times 3 \times 5 + 128 \times 128 \times 5 \\ & = 4,026,613,760 \end{aligned}$$

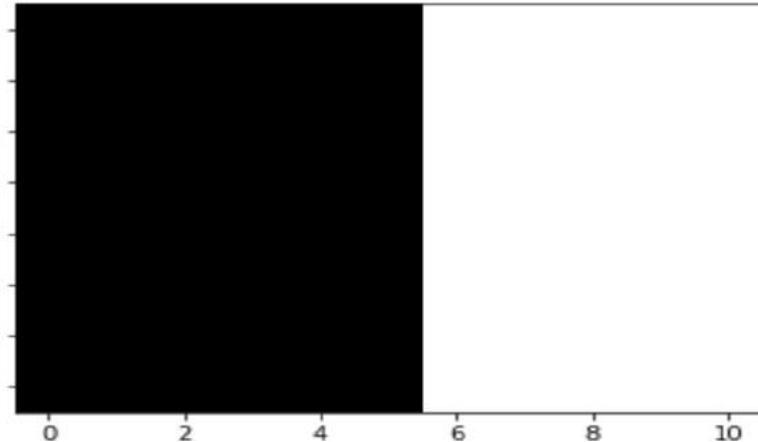
# What a CNN Can See

Simple example: vertical edge detector

```
conv.weight[0, 0, :, :] = torch.tensor([[1, 0, -1],
                                         [1, 0, -1],
                                         [1, 0, -1]]).float()

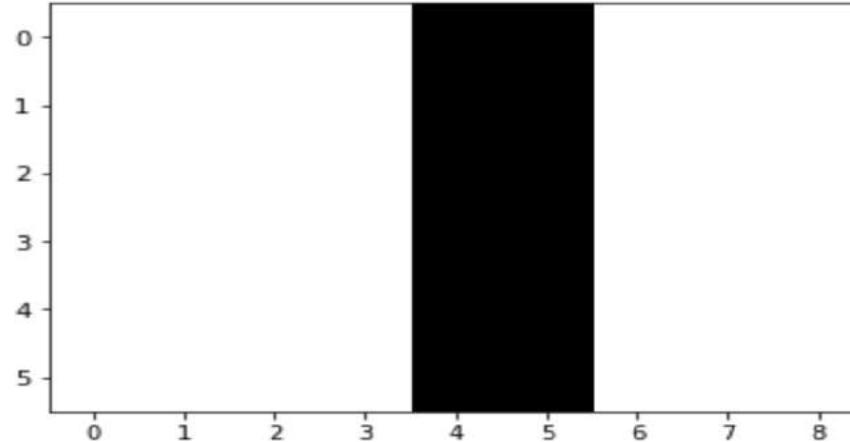
t = torch.tensor([
[0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 1., 1., 1., 1., 1.]
])

plt.imshow(t, cmap='gray');
```

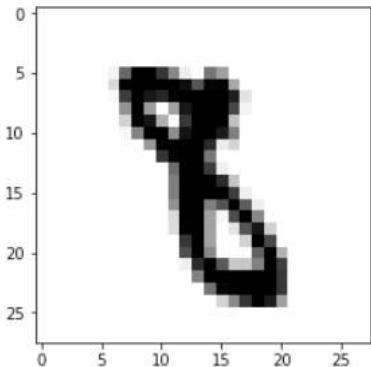


(From classical computer vision research)

```
: tt = torch.zeros([1, 1] + list(t.size()))
tt[0, 0, :, :] = t
after = conv(tt)
plt.imshow(after[0, 0, :, :].detach().numpy(), cmap='gray');
```



# Kernel Dimensions



```
a.shape
```

```
(1, 28, 28)
```

```
import torch
```

```
conv = torch.nn.Conv2d(in_channels=1,  
                      out_channels=8,  
                      kernel_size=(5, 5),  
                      stride=(1, 1))
```

```
conv.weight.size()
```

```
torch.Size([8, 1, 5, 5])
```

```
conv.bias.size()
```

```
torch.Size([8])
```

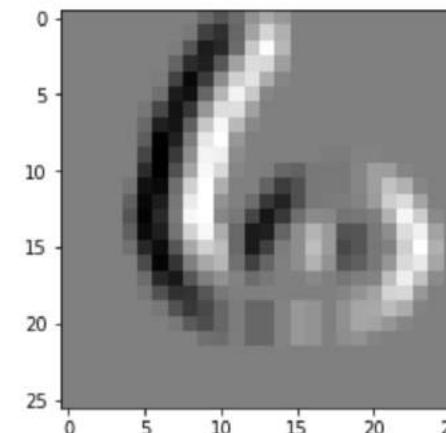
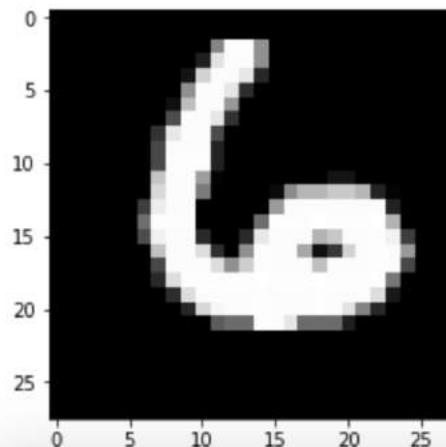
For a grayscale image with a 5x5 feature detector (kernel), we have the following dimensions (number of parameters to learn)

What do you think is the output size for this 28x28 image?

# What a CNN Can See

Simple example: vertical edge detector

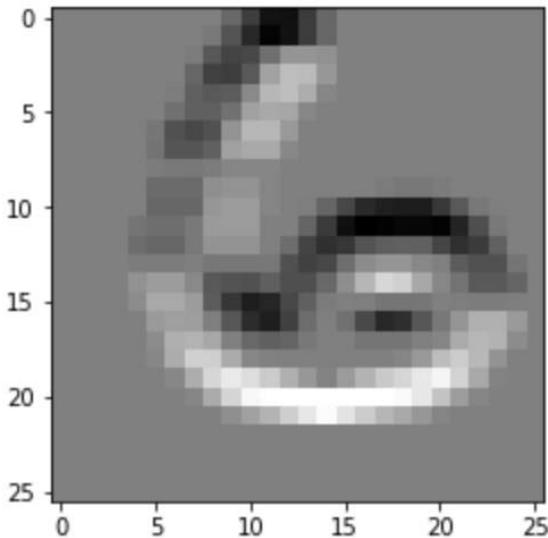
```
conv = torch.nn.Conv2d(in_channels=1,  
                      out_channels=1,  
                      kernel_size=(3, 3))  
  
conv.weight.size()  
  
torch.Size([1, 1, 3, 3])  
  
conv.weight[0, 0, :, :] = torch.tensor([[1, 0, -1],  
                                         [1, 0, -1],  
                                         [1, 0, -1]]).float()  
conv.bias[0] = torch.tensor([0.]).float()  
  
images_after = conv(images)  
  
plt.imshow(images[5, 0], cmap='gray');  
plt.imshow(images_after[5, 0].detach().numpy() , cmap='gray');
```



# What a CNN Can See

Simple example: horizontal edge detector

```
: conv.weight[0, 0, :, :] = torch.tensor([[1, 1, 1],  
                                         [0, 0, 0],  
                                         [-1, -1, -1]]).float()  
conv.bias[0] = torch.tensor([0.]).float()  
  
: images_after2 = conv(images)  
  
: plt.imshow(images_after2[5, 0].detach().numpy() , cmap='gray');
```



A CNN can learn whatever it finds best based on optimizing the objective (e.g., minimizing a particular loss to achieve good classification accuracy)

# Padding Jargon

"valid" convolution: no padding (feature map may shrink)

"same" convolution: padding such that the output size  
is equal to the input size

Common kernel size conventions:

3x3, 5x5, 7x7 (sometimes 1x1 in later layers to reduce channels)

# Padding

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

Assume you want to use a convolutional operation with stride 1 and maintain the input dimensions in the output feature map:

How much padding do you need for "same" convolution?

$$o = i + 2p - k + 1$$

$$\Leftrightarrow p = (o - i + k - 1)/2$$

$$\Leftrightarrow p = (k - 1)/2$$

# Padding

$$o = i + 2p - k + 1$$

$$\Leftrightarrow p = (o - i + k - 1)/2$$

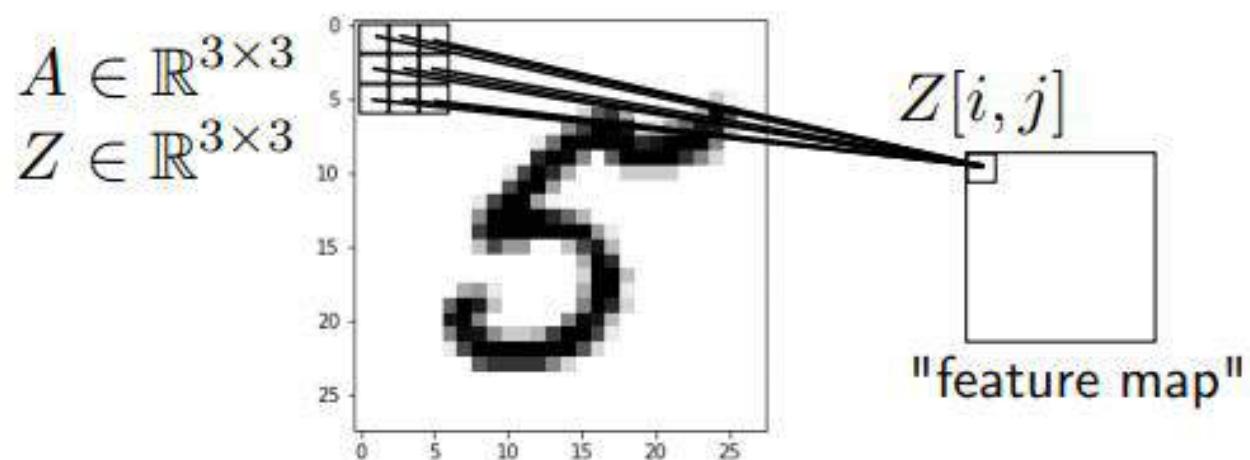
$$\Leftrightarrow p = (k - 1)/2$$

Probably explains why common kernel size conventions are  
3x3, 5x5, 7x7 (sometimes 1x1 in later layers to reduce channels)

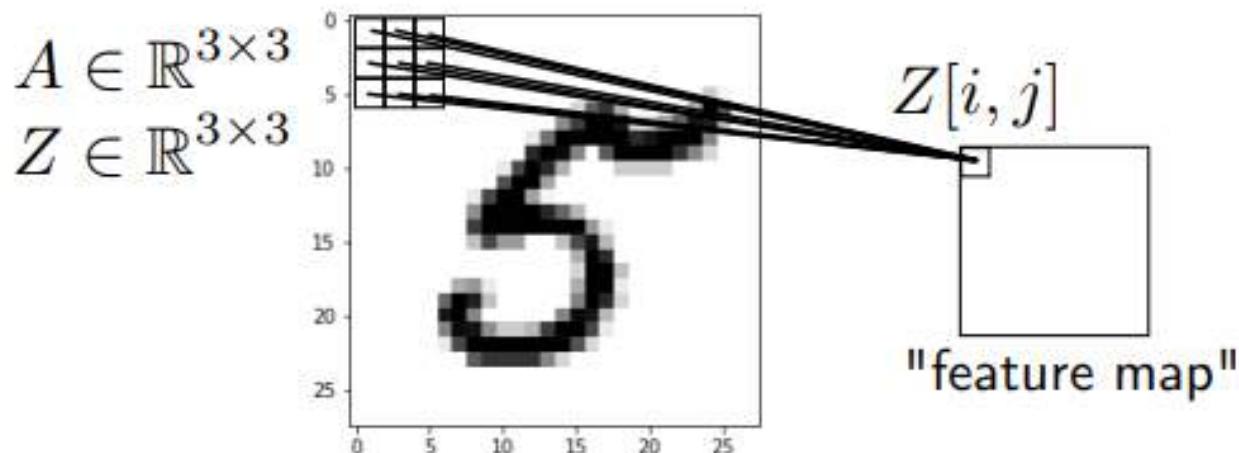
# Cross-Correlation vs Convolution

Deep Learning Jargon: convolution in DL is actually cross-correlation

Cross-correlation is our sliding dot product over the image



# Cross-Correlation vs Convolution



## Cross-Correlation:

$$Z[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k K[u, v] A[i + u, j + v]$$

$$Z[i, j] = K \otimes A$$

# Cross-Correlation vs Convolution

Cross-Correlation:

$$Z[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k K[u, v] A[i + u, j + v]$$

$$Z[i, j] = K \otimes A$$

Looping direction  
indicated in red

1) -1,-1	2) -1,0	3) -1,1
4) 0,-1	5) 0,0	6) 0,1
7) 1,-1	8) 1,0	9) 1,1

# Cross-Correlation vs Convolution

Cross-Correlation:  $Z[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k K[u, v]A[i + u, j + v]$        $Z[i, j] = K \otimes A$

Convolution:  $Z[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k K[u, v]A[i - u, j - v]$

$$Z[i, j] = K * A$$

Basically, we are flipping the kernel (or the receptive field) horizontally and vertically

Looping direction  
indicated in red

9)	8)	7)
-1,-1	-1,0	-1,1
6)	5)	4)
0,-1	0,0	0,1
3)	2)	1)
1,-1	1,0	1,1

# Cross-Correlation vs Convolution

Deep Learning Jargon: convolution in DL is actually cross-correlation

"Real" convolution has the nice associative property:

$$(A * B) * C = A * (B * C)$$

In DL, we usually don't care about that (as opposed to many traditional computer vision and signal processing applications).

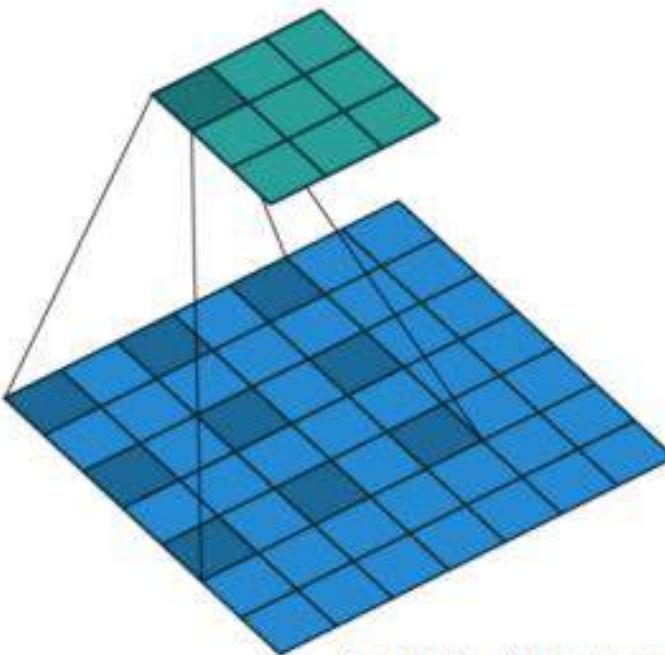
Also, cross-correlation is easier to implement.

Maybe the term "convolution" for cross-correlation became popular, because "Cross-Correlational Neural Network" sounds weird ;)

# Dilated Convolutions

$$o = \left\lfloor \frac{i + 2p - k - (k - 1)(d - 1)}{s} \right\rfloor + 1$$

A 2-dilated 2D convolution



Dumoulin, Vincent, and Francesco Visin. "[A guide to convolution arithmetic for deep learning](#)." *arXiv preprint arXiv:1603.07285* (2016).

# Transposed Convolution

- Allows us to increase the size of the *output* feature map compared to the *input* feature map
- Synonyms:
  - ▶ often also (incorrectly) called "deconvolution" or sometimes (mathematically, deconvolution is defined as the inverse of convolution, which is different from transposed convolutions)
  - ▶ the term "unconv" is sometimes also used
  - ▶ fractionally strided convolution is another term for that

## Regular Convolution:

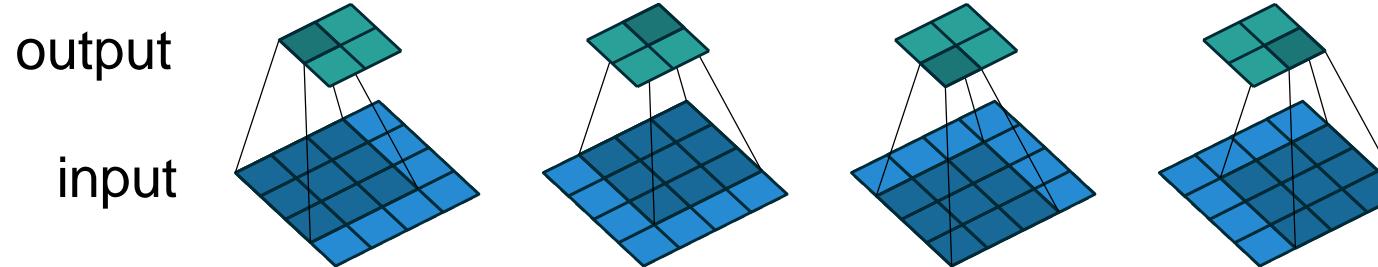
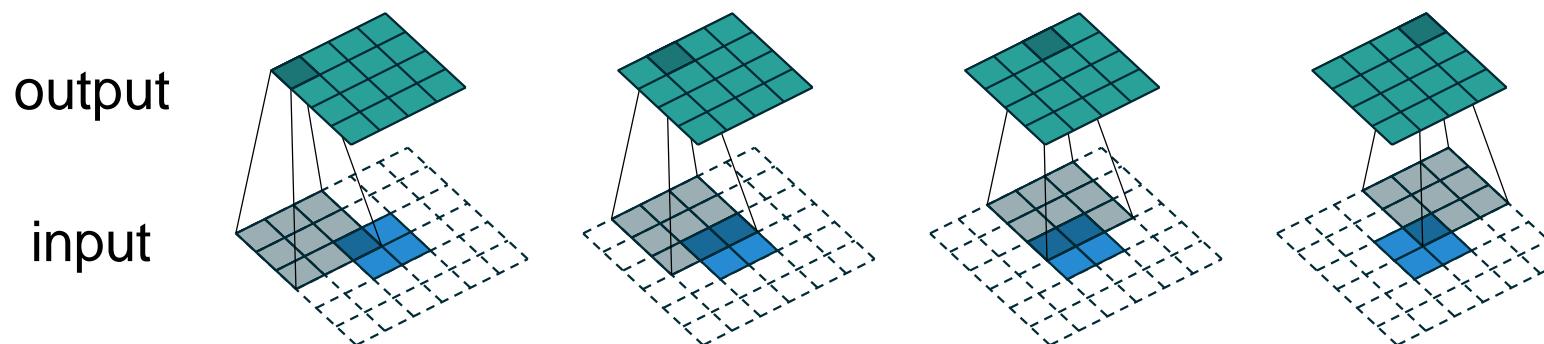


Figure 2.1: (No padding, unit strides) Convolving a  $3 \times 3$  kernel over a  $4 \times 4$  input using unit strides (i.e.,  $i = 4$ ,  $k = 3$ ,  $s = 1$  and  $p = 0$ ).

## Transposed Convolution (emulated with direct convolution):



Dumoulin, Vincent, and Francesco Visin. "[A guide to convolution arithmetic for deep learning](#)." *arXiv preprint arXiv:1603.07285* (2016).

# Transposed Convolution

strides: in transposed convolutions, we stride over the output; hence, larger strides will result in larger outputs (opposite to regular convolutions)

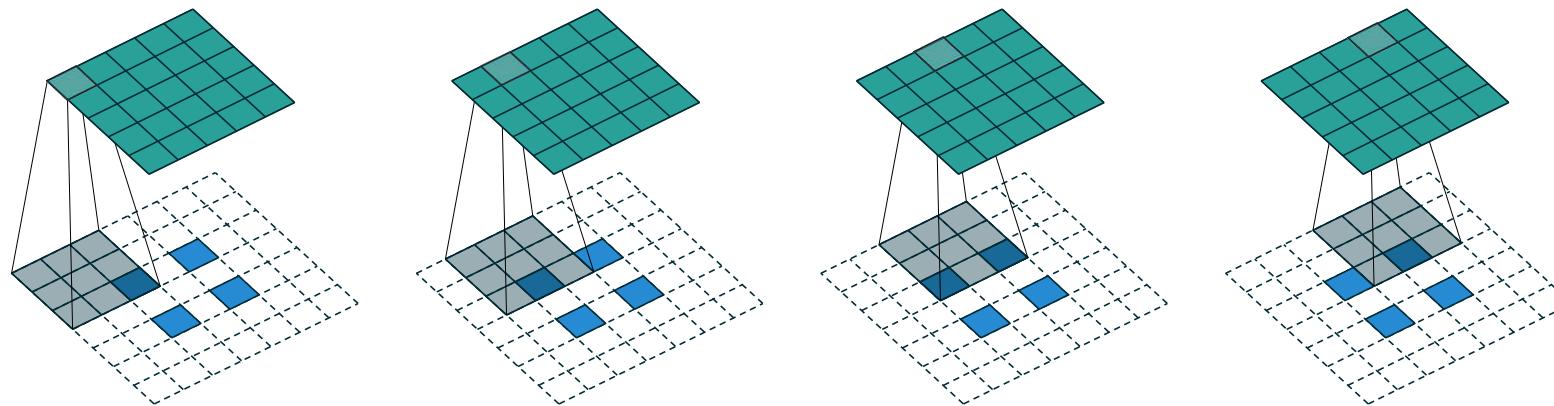
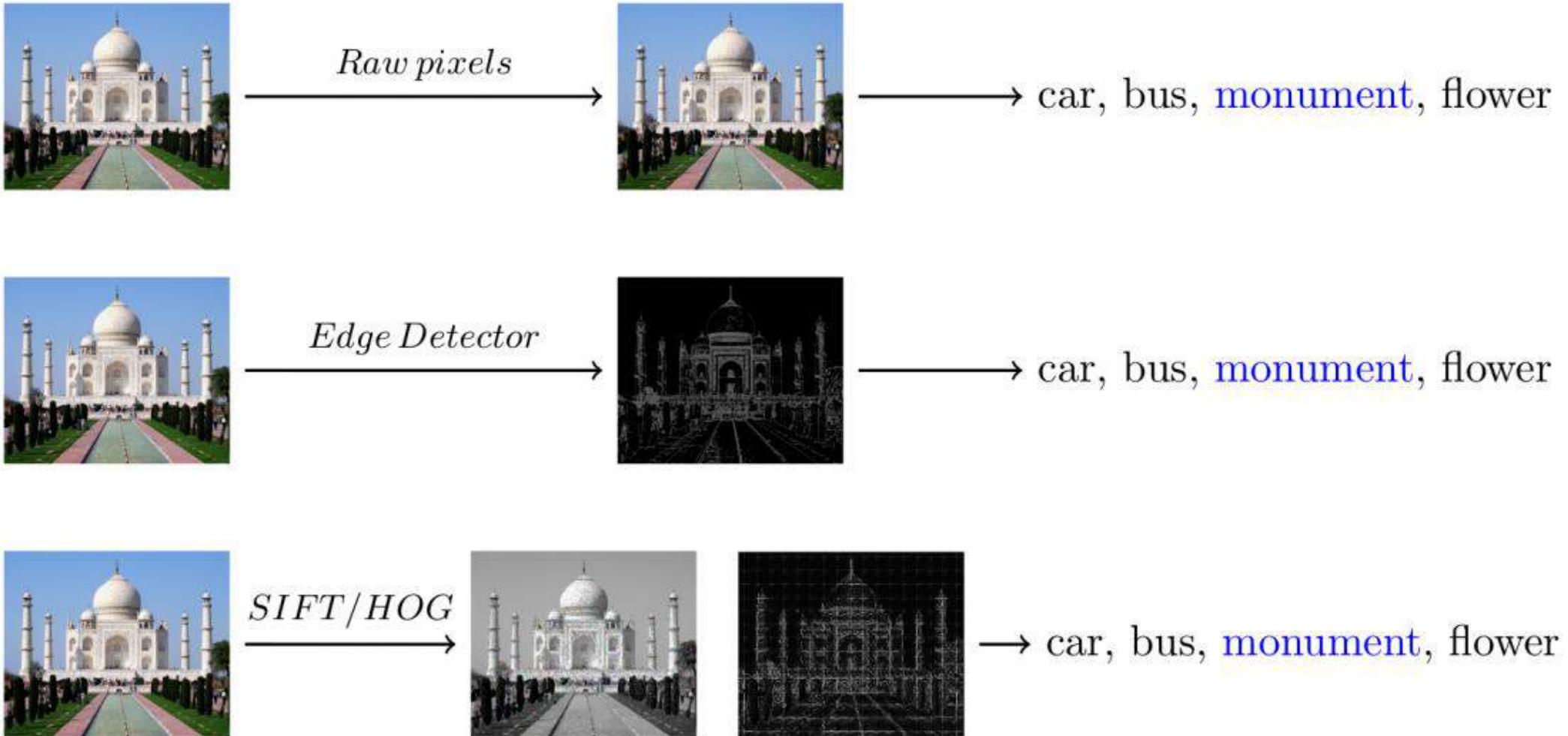


Figure 4.5: The transpose of convolving a  $3 * 3$  kernel over a  $5 * 5$  input using  $2 * 2$  strides (i.e.,  $i = 5, k = 3, s = 2$  and  $p = 0$ ). It is equivalent to convolving a  $3 * 3$  kernel over a  $2 * 2$  input (with 1 zero inserted between inputs) padded with a  $2 * 2$  border of zeros using unit strides

Dumoulin, Vincent, and Francesco Visin. "[A guide to convolution arithmetic for deep learning](#)." *arXiv preprint arXiv:1603.07285* (2016).

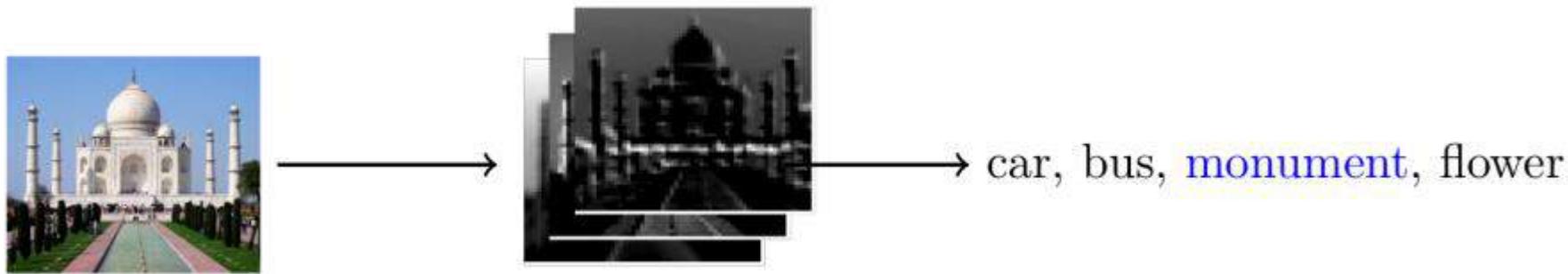
# Classification Pipeline



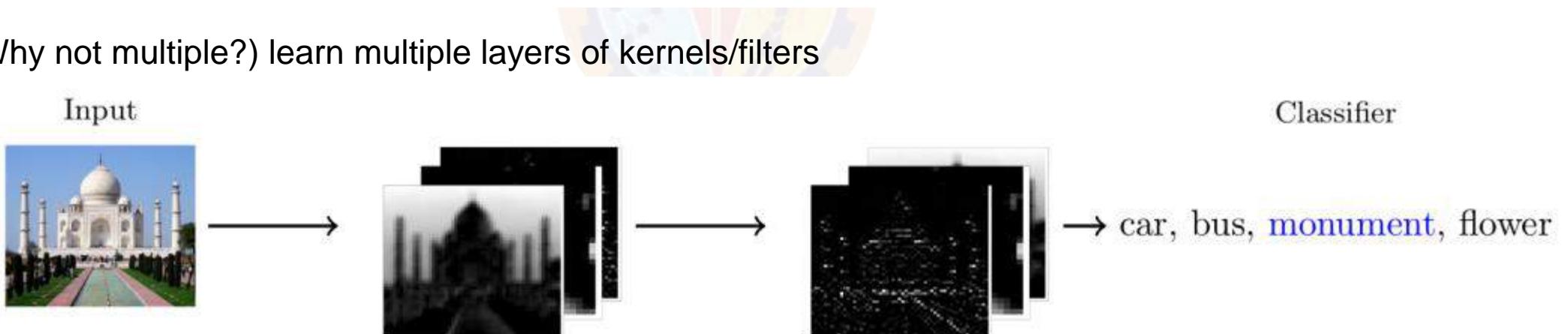
- Where is learning? (hand craft features, then learn weights for classification). One can see feature as a convolution.

# Automate feature kernel discovery

- Instead of handcrafted kernels, learn filters

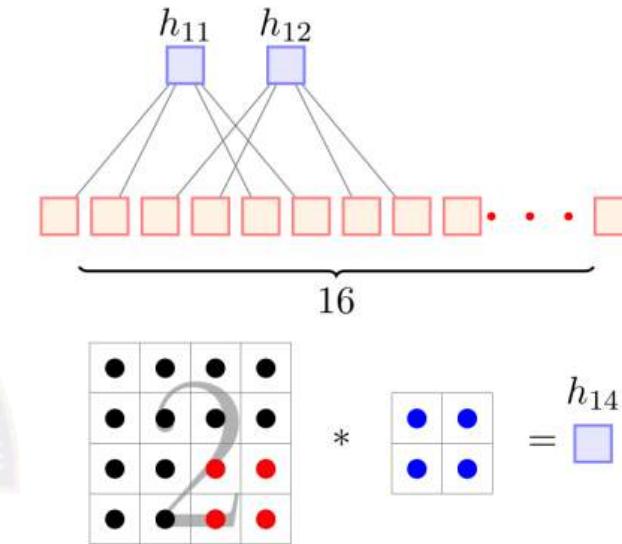
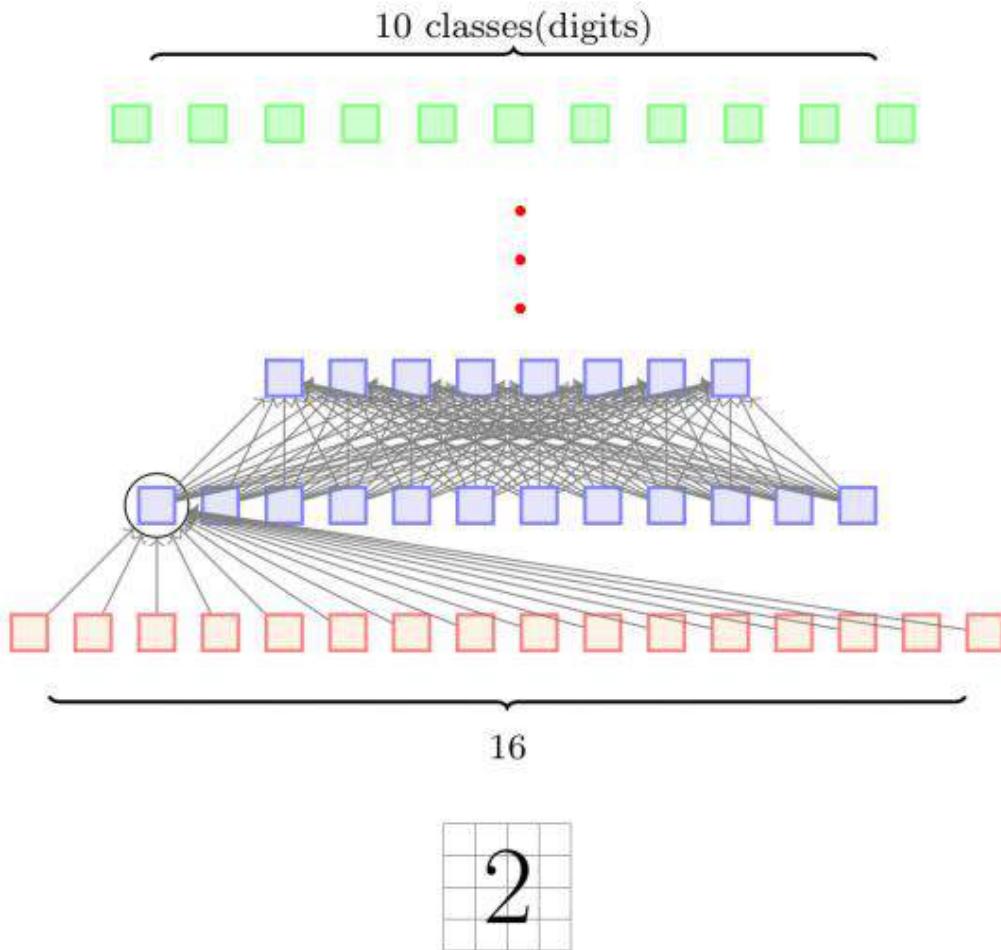


- (Why not multiple?) learn multiple layers of kernels/filters

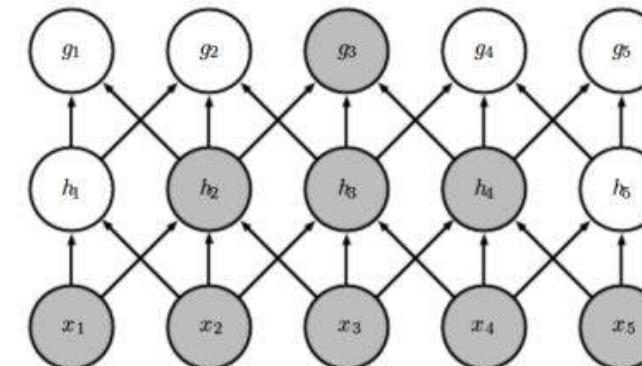


Treating these kernels as parameters and learning them.

# CNN has sparse connectivity with respect to NN

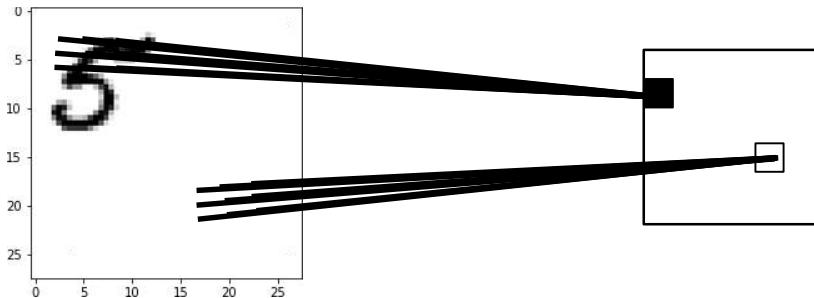


Interactions are preserved, even with reduced model parameters,

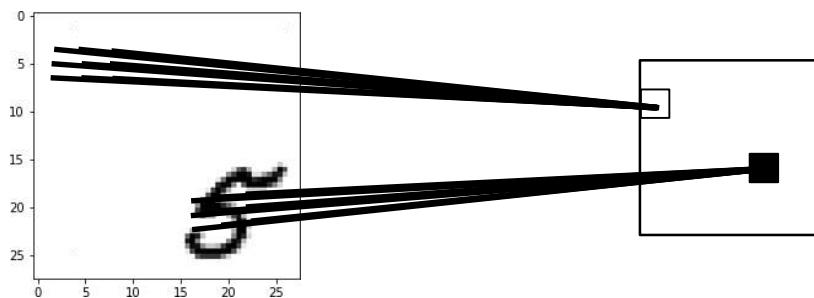


# CNNs and Translation/Rotation/Scale Invariance

Note that CNNs are not really invariant to scale, rotation, translation, etc.

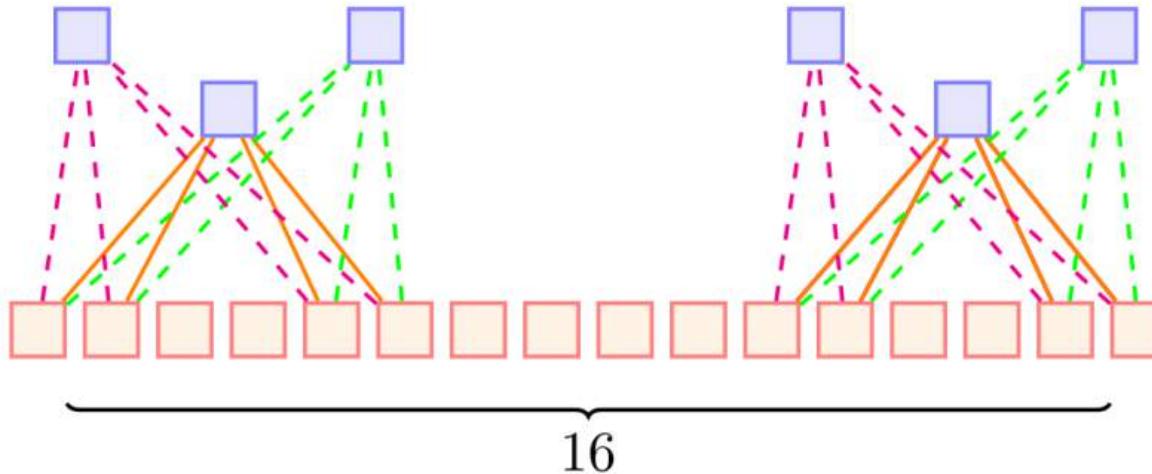


The activations are still dependent on the location, etc.



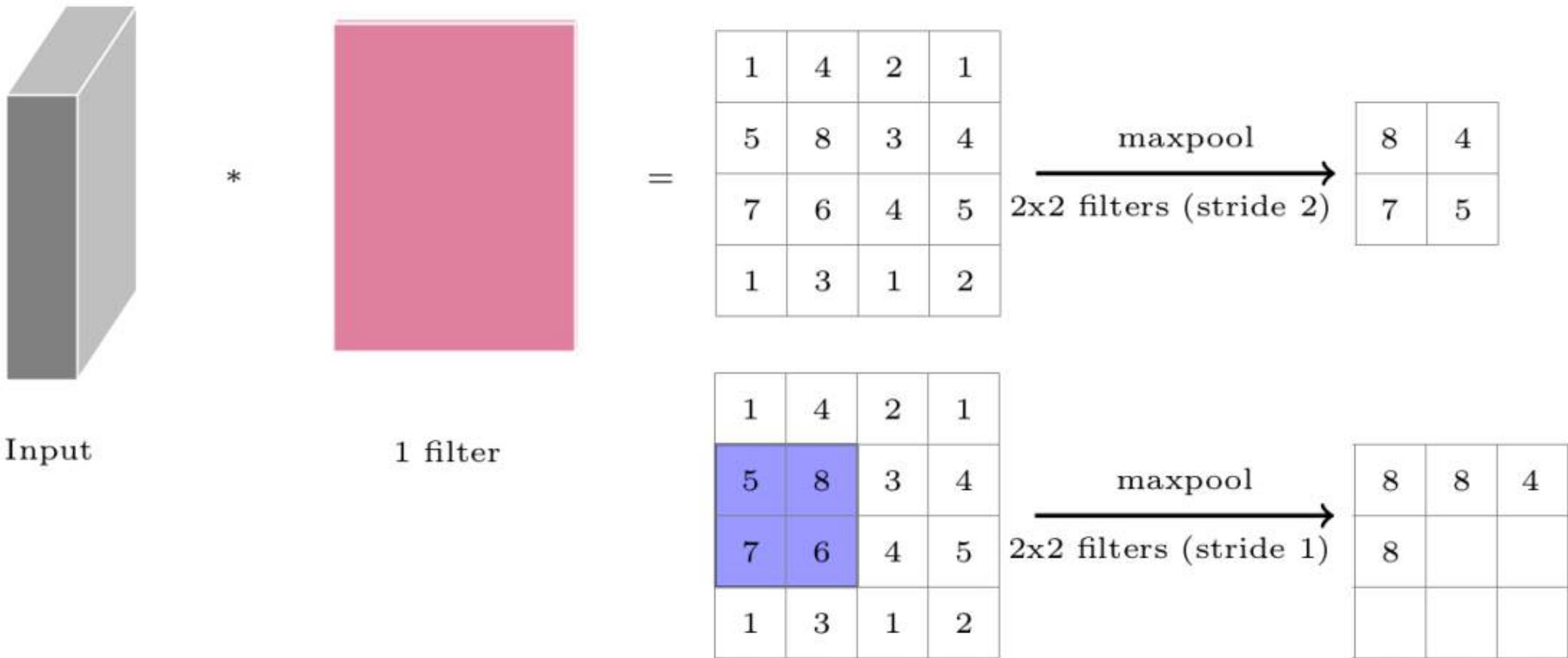
# Weight sharing in CNN

- One place you are extracting edge other place something else? So we do not want the kernel to be different for different portions of the image.



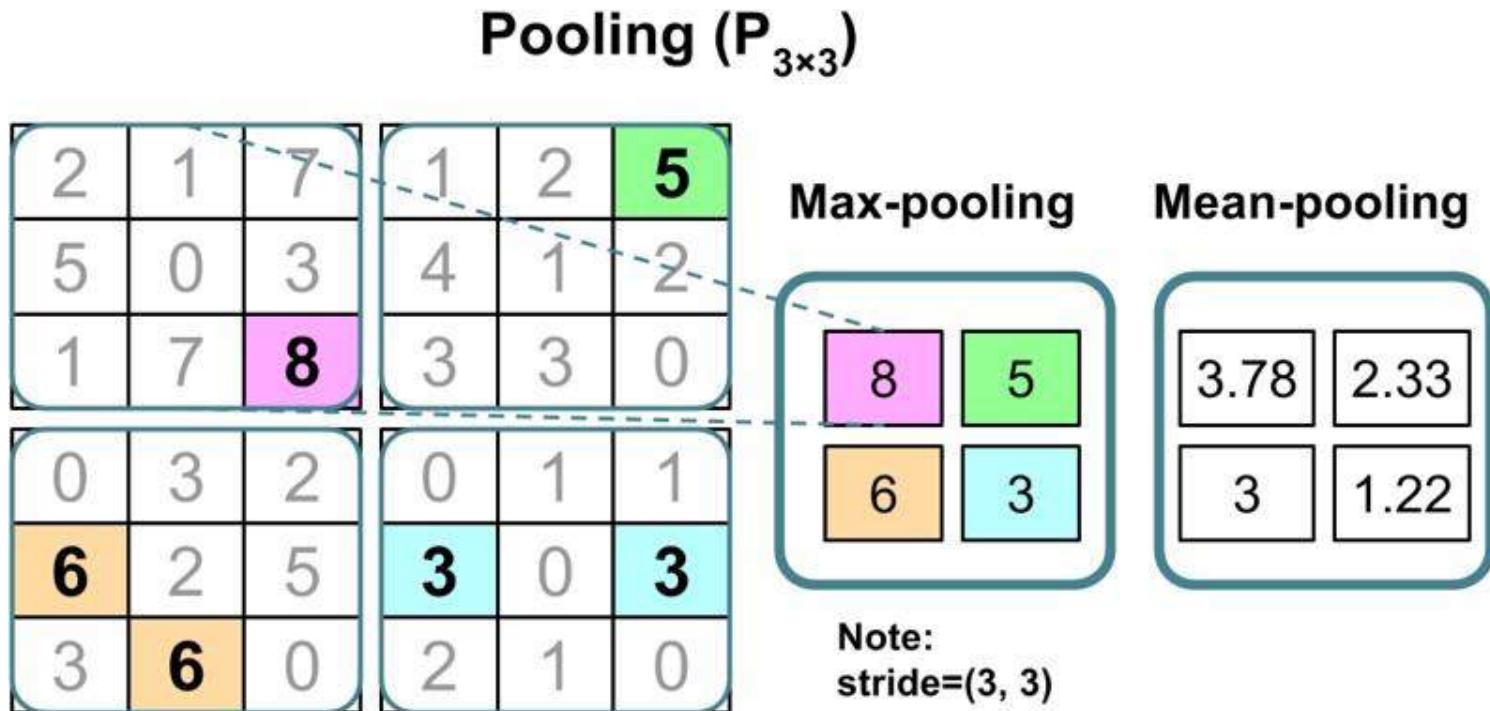
- Weight sharing in CNN makes the job of learning weights easier
- Multiple kernels help get different feature at the same level

# Pooling (max, min, average)



- Training CNN? backpropagation!

# Pooling Layers Can Help With Local Invariance



Note that typical pooling layers do not have any learnable parameters

Downside: Information is lost.

May not matter for classification, but applications where relative position is important (like face recognition)

In practice for CNNs: some image preprocessing still recommended

# Specify Architecture

```
# Create a Sequential model object
cnnModel = models.Sequential()

# Add layers Conv2D for CNN and sepcify MaxPooling

# Layer 1 = input layer
cnnModel.add(layers.Conv2D(32, (3,3), activation="relu", input_shape=(28,28,1) ))
cnnModel.add(layers.MaxPooling2D((2,2)))

# Layer 2
cnnModel.add(layers.Conv2D(64, (3,3), activation="relu"))
cnnModel.add(layers.MaxPooling2D((2,2)))
|
# Layer 3
cnnModel.add(layers.Conv2D(64, (3,3), activation="relu" ))
cnnModel.add(layers.Flatten())

# Add Dense layers or fully connected layers
# Layer 4
cnnModel.add(layers.Dense(64, activation="relu" ))
# Layer 5
cnnModel.add(layers.Dense(32, activation="relu" ))
# Layer 6
cnnModel.add(layers.Dense(10, activation="softmax" ))

cnnModel.summary()
```

# Specify Training

```
# Create a Sequential model object
cnnModel = models.Sequential()

# Add layers Conv2D for CNN and sepcify MaxPooling

# Layer 1 = input layer
cnnModel.add(layers.Conv2D(32, (3,3), activation="relu", input_shape=(28,28,1) ))
cnnModel.add(layers.MaxPooling2D((2,2)))

# Layer 2
cnnModel.add(layers.Conv2D(64, (3,3), activation="relu"))
cnnModel.add(layers.MaxPooling2D((2,2)))
,
```

```
[ ] # Configure the model for training, by using appropriate optimizers and regularizations
# Available optimizer: adam, rmsprop, adagrad, sgd
# loss: objective that the model will try to minimize.
# Available loss: categorical_crossentropy, binary_crossentropy, mean_squared_error
# metrics: List of metrics to be evaluated by the model during training and testing.

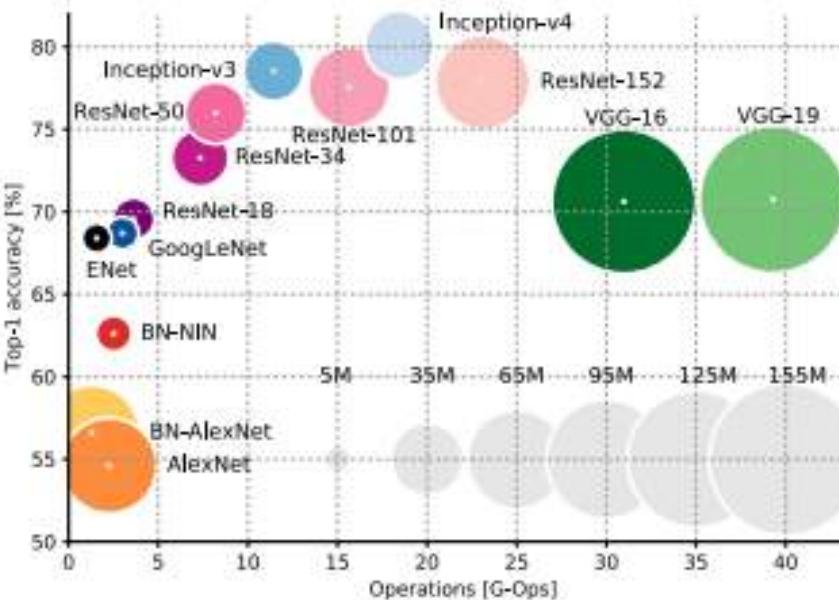
cnnModel.compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = ["accuracy"])
```

```
# train the model
```

```
history = cnnModel.fit(Xtrain, Ytrain, epochs = 25, batch_size = 64, validation_split = 0.1)
```

# Common Architectures Revisited

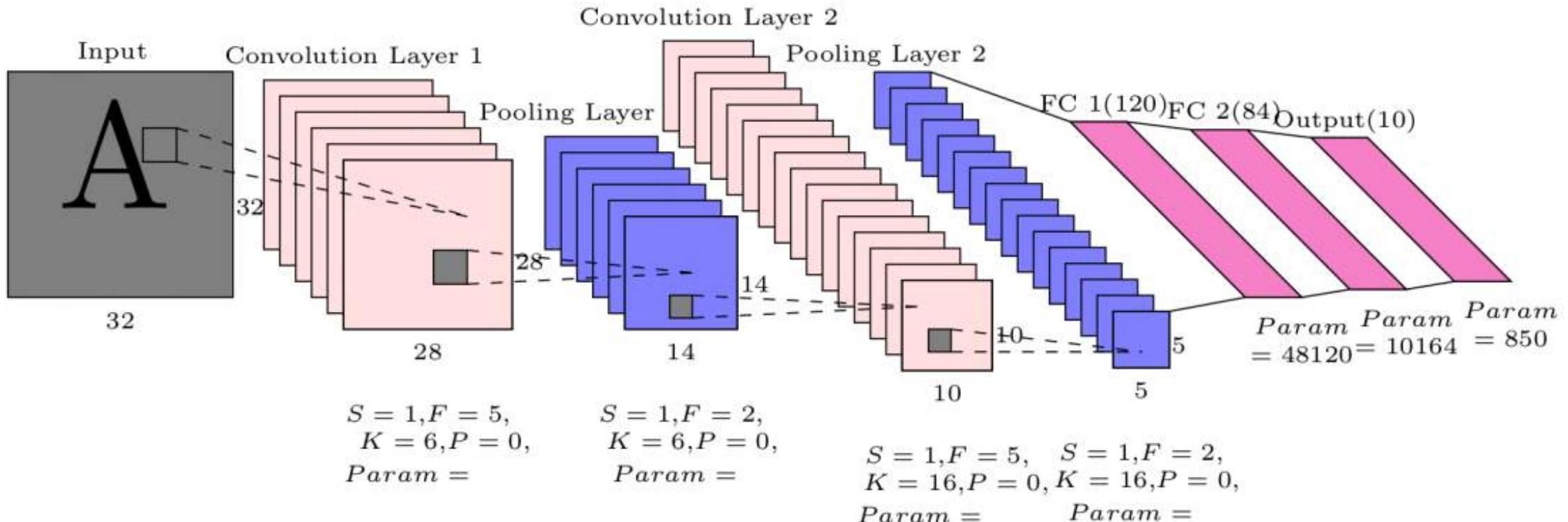
We will discuss some additional common CNN architectures since the field evolved quite a bit since 2012 ...



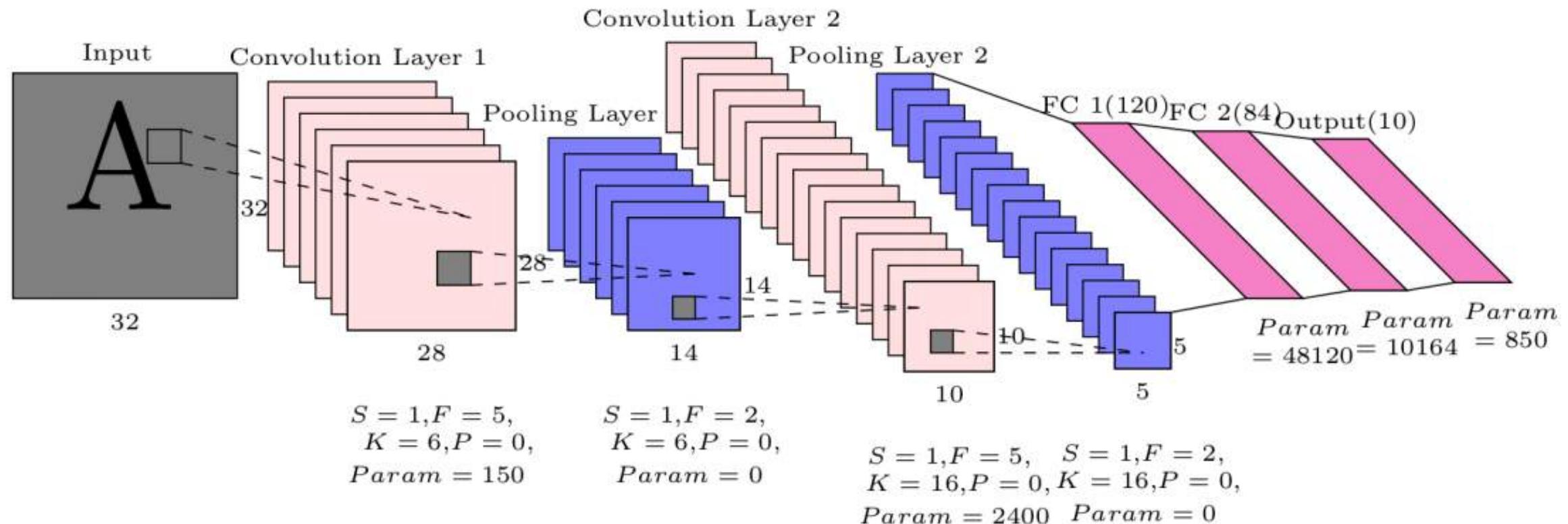
Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

# Case-Study: LeNet-5

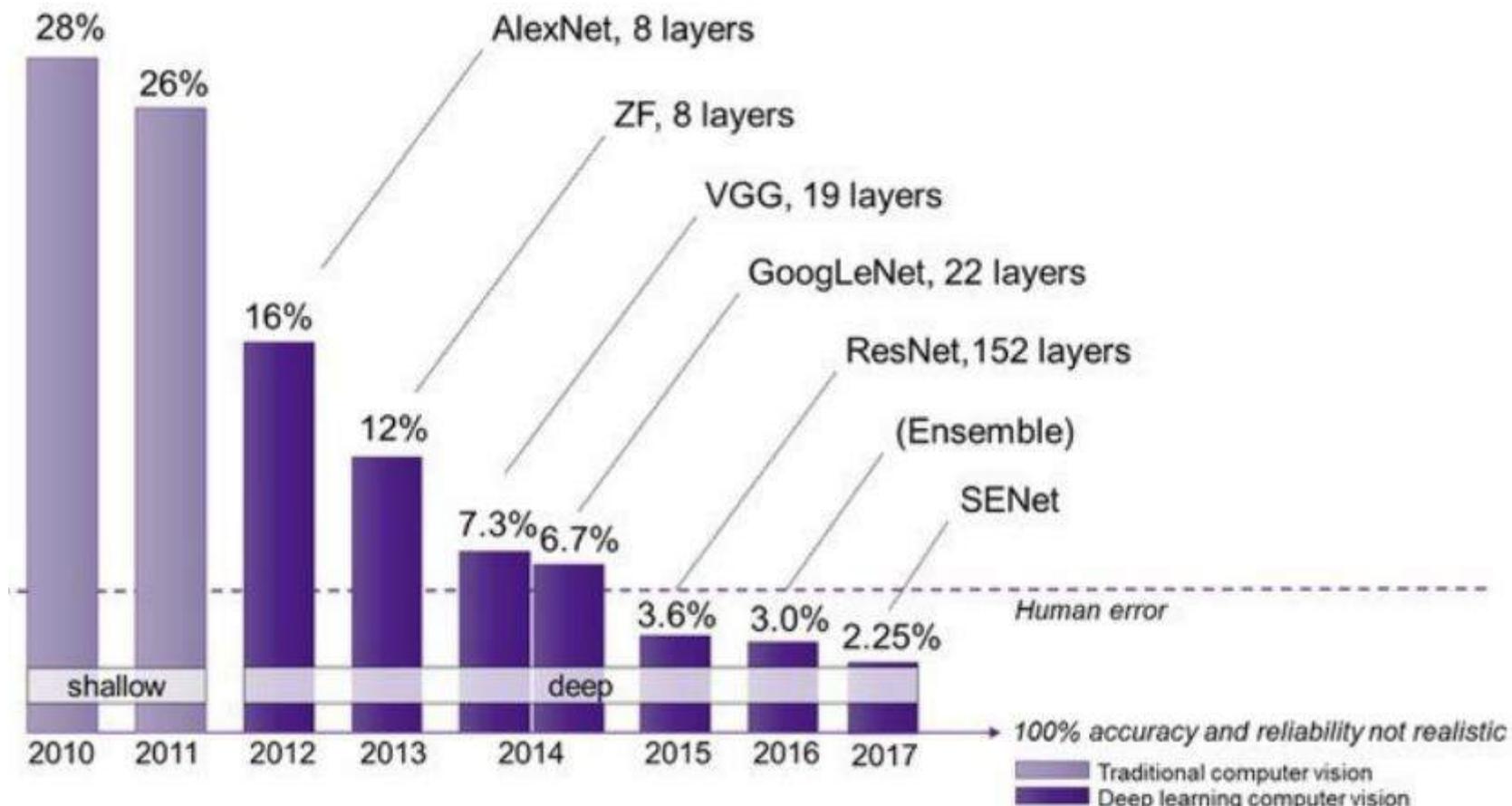
## Handwritten character recognition



# LeNet-5 for handwritten character recognition



# ImageNet ILSVRC



- (2009) 22K category, 14M images
- Challenge 1000 class, 1431167 images
- HoG, LBP, SVM ...

# AlexNet

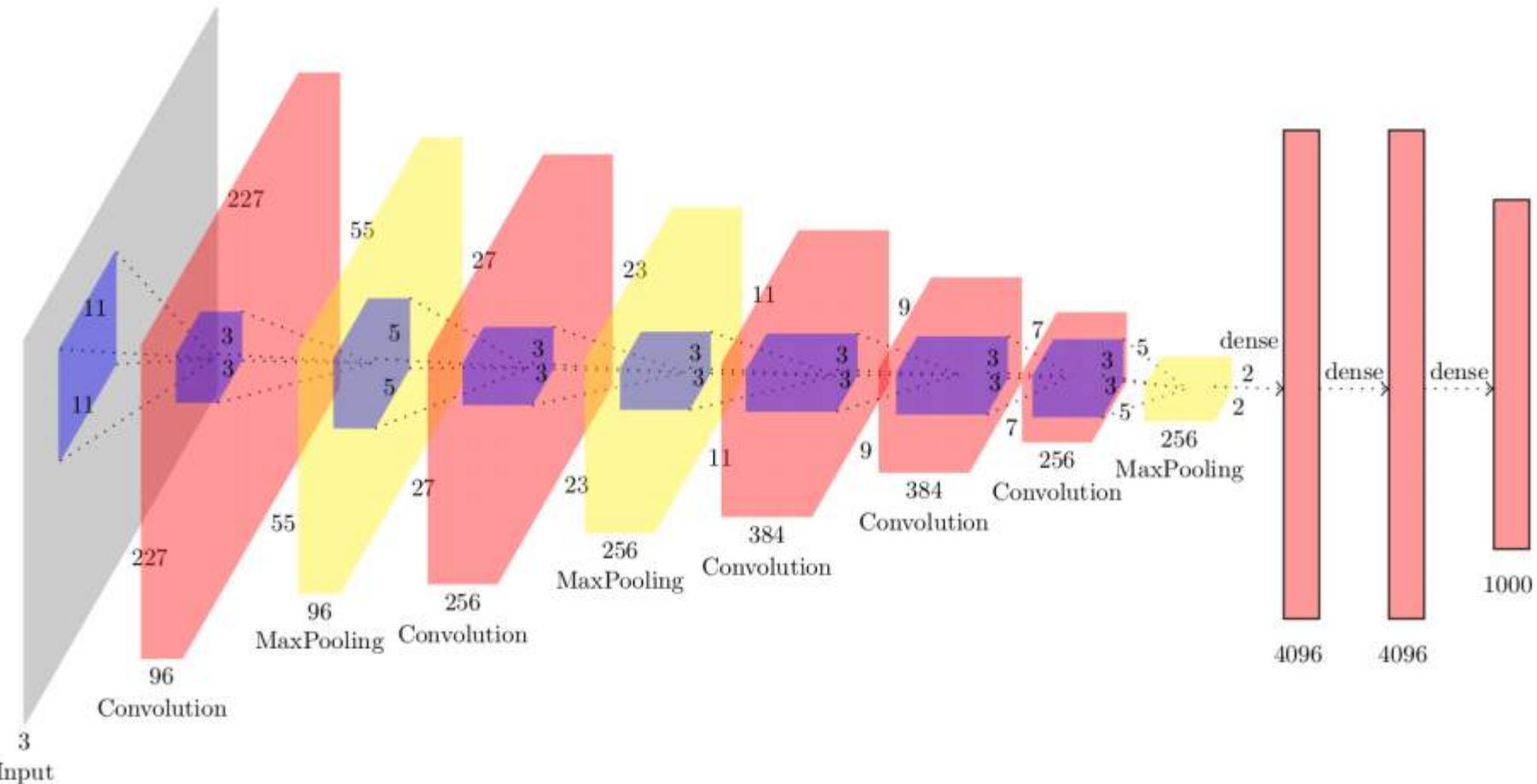


Image size  $227 \times 227 \times 3 \rightarrow [96 F=11, S=4, P=0] \rightarrow [\text{MaxPool } F=3, S=2] \rightarrow [256 F=5, S=1, P=0] \rightarrow [\text{MaxPool } F=3, S=2] \rightarrow [384 F=3, S=1, P=0] \rightarrow [384 F=3, S=1, P=0] \rightarrow [256 F=3, S=1, P=0] \rightarrow [\text{MaxPool } F=3, S=2] \rightarrow \text{FC } 4096 \rightarrow \text{FC } 4096 \rightarrow \text{FC } 1000$

# ZFNet

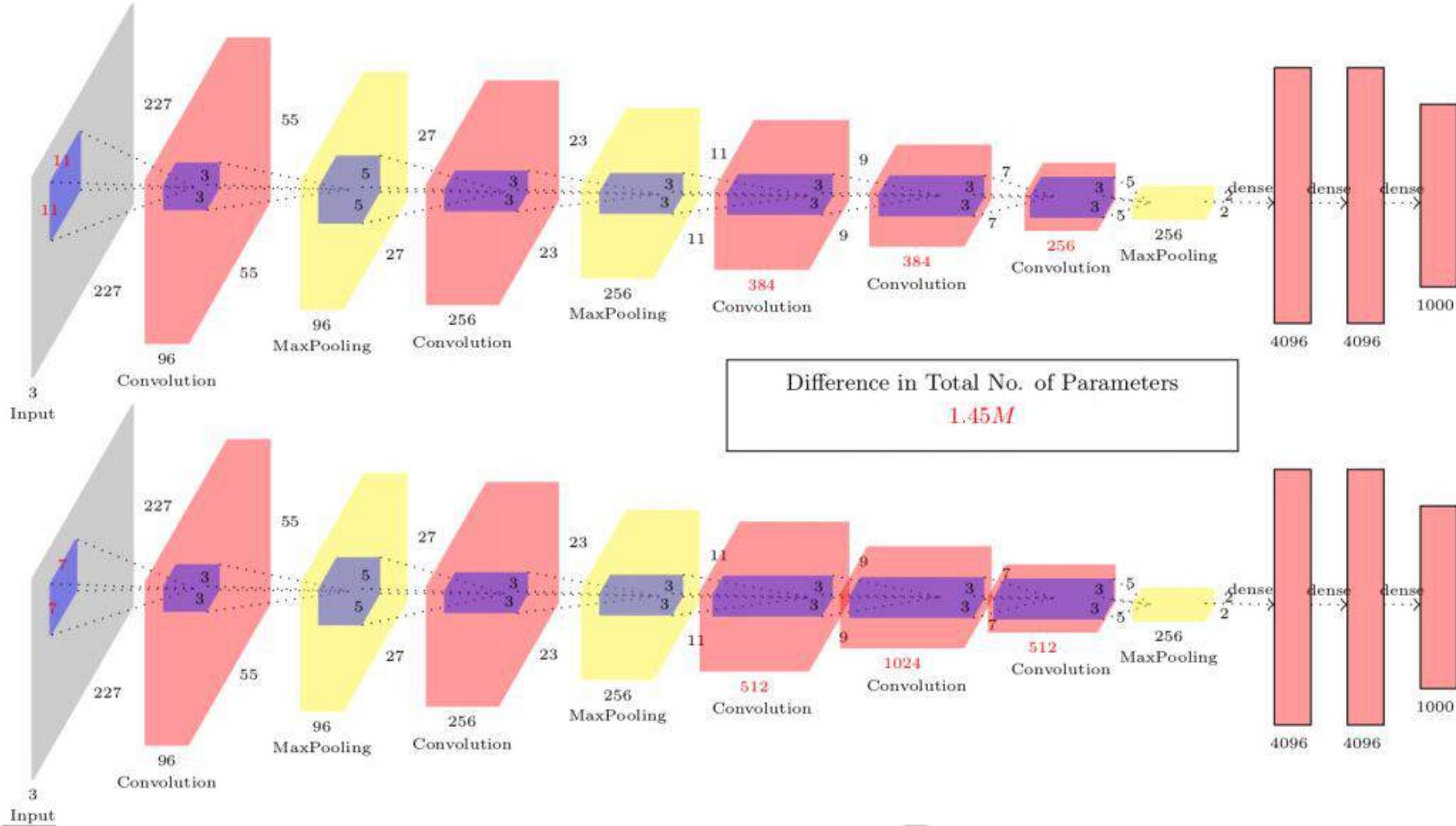
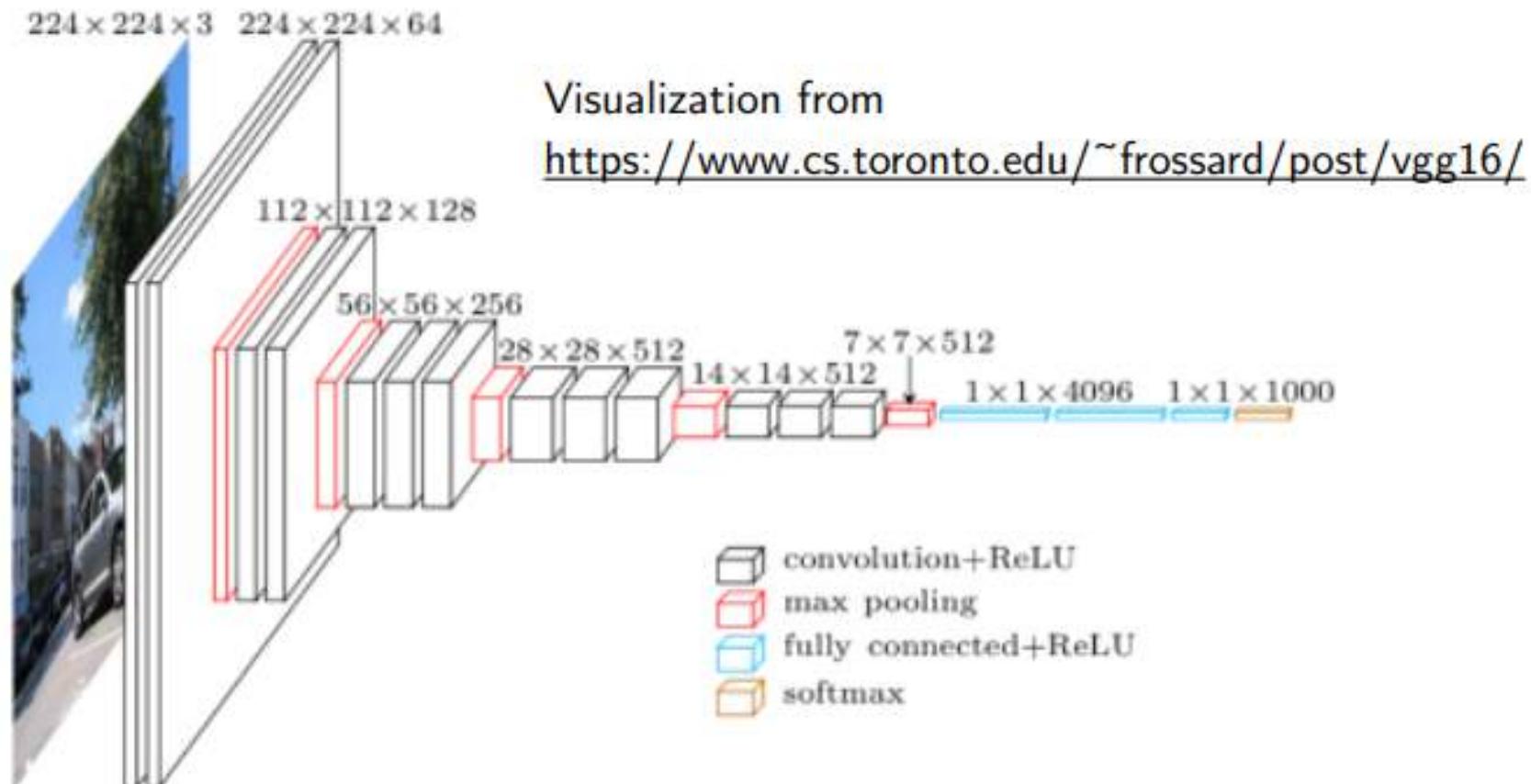


Image size  $227 \times 227 \times 3$  → [69 F=7, S=4, P=0] → [MaxPool F=3, S=2] → [256 F=5, S=1, P=0] → [MaxPool F=3, S=2] → [512 F=3, S=1, P=0] → [1024 F=3, S=1, P=0] → [512 F=3, S=1, P=0] → [MaxPool F=3, S=2] → FC 4096 → FC 4096 → FC 1000

# VGG-16

PyTorch implementation: [https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13\\_intro-cnn/code/vgg16.ipynb](https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/vgg16.ipynb)



Simonyan, Karen, and Andrew Zisserman. "[Very deep convolutional networks for large-scale image recognition.](#)" *arXiv preprint arXiv:1409.1556* (2014).

# VGG-16

PyTorch implementation: [https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13\\_intro-cnn/code/vgg16.ipynb](https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/vgg16.ipynb)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

## Advantages:

very simple architecture,  
3x3 convs, stride=1,  
"same" padding, 2x2 max pooling

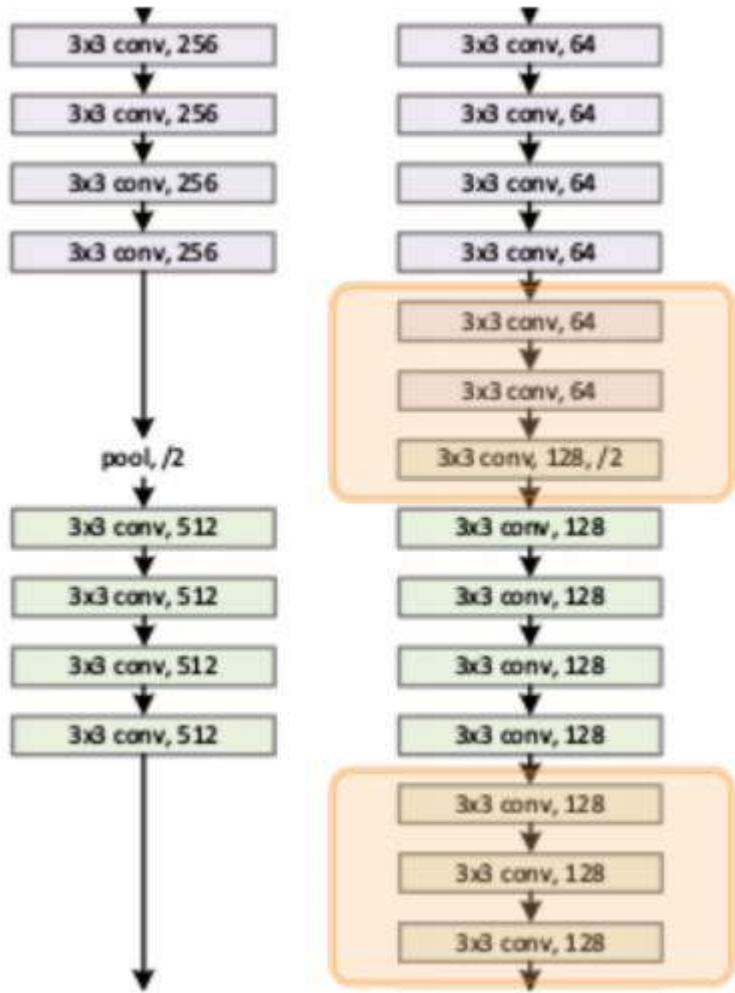
## Disadvantage:

very large number of parameters  
and slow  
(see previous slide)

Simonyan, Karen, and Andrew Zisserman. "[Very deep convolutional networks for large-scale image recognition.](https://arxiv.org/abs/1409.1556)" *arXiv preprint arXiv:1409.1556* (2014).

# ResNet

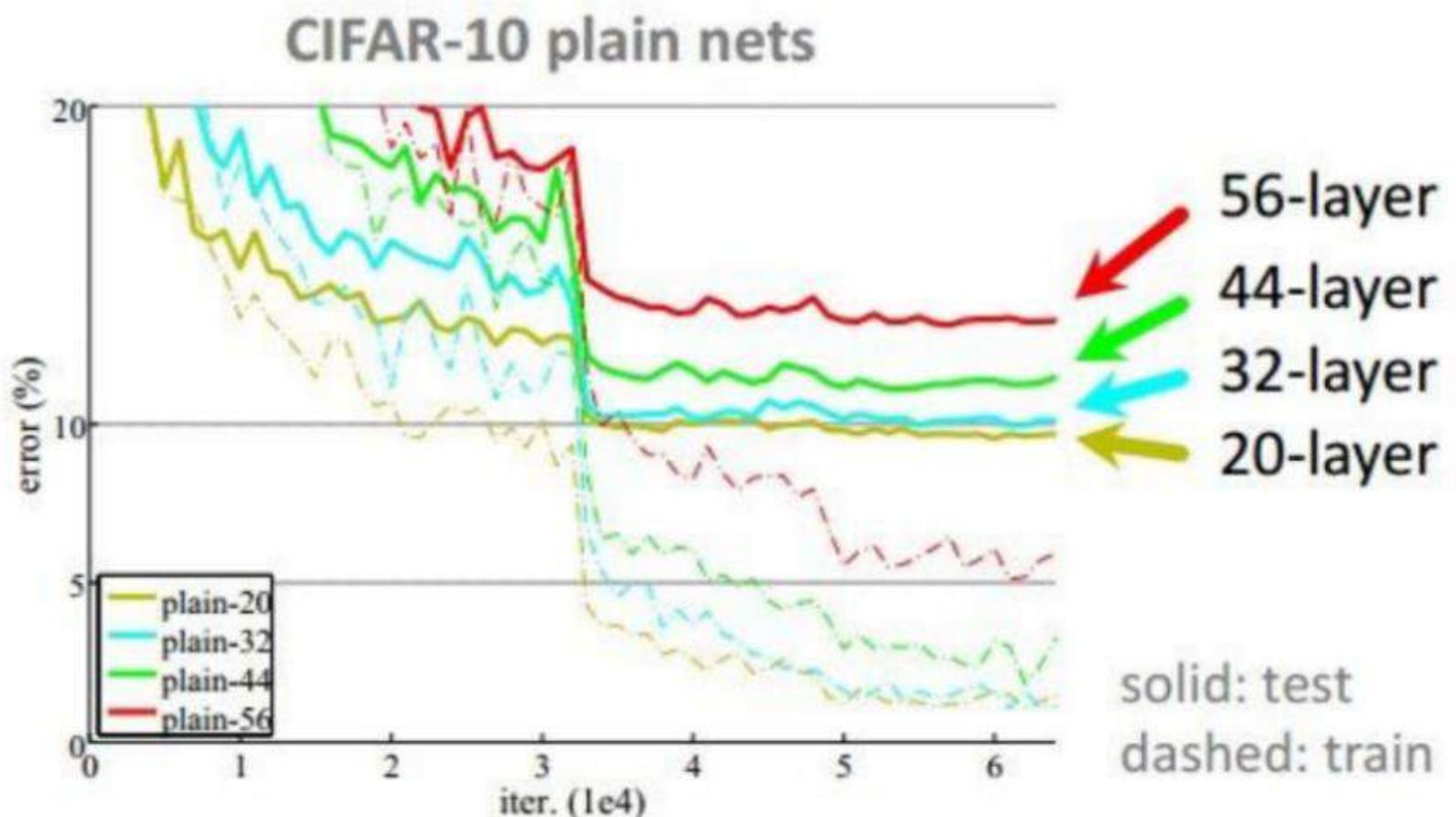
If a shallow neural network works well. What would happen if we add more layers?



- Deep network should also work well (It would learn identity in new layers)

# ResNet

But, in practice it was not happening



Why? Identity is one of the solution in large domain.

# ResNets

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "[Deep residual learning for image recognition.](#)" In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

With their simple trick of allowing skip connections (the possibility to learn identity functions and skip layers that are not useful), ResNets allow us to implement very, very deep architectures

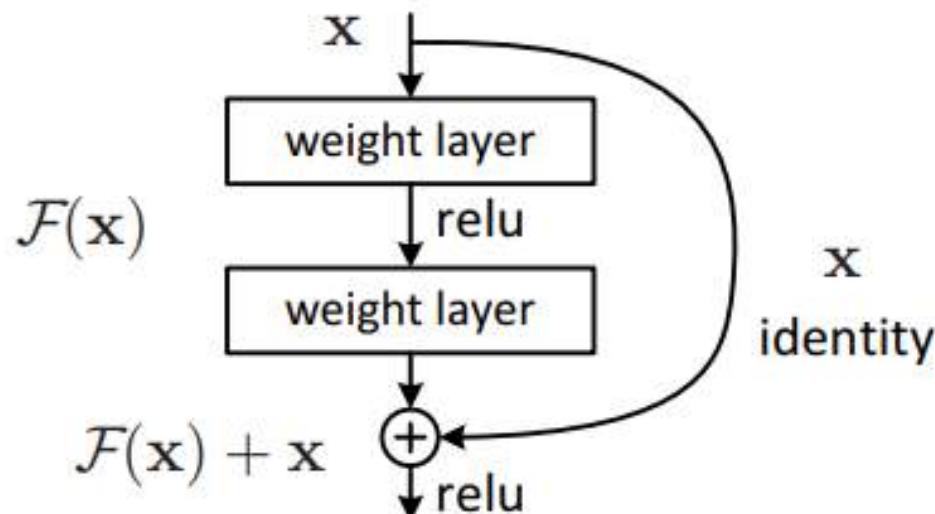
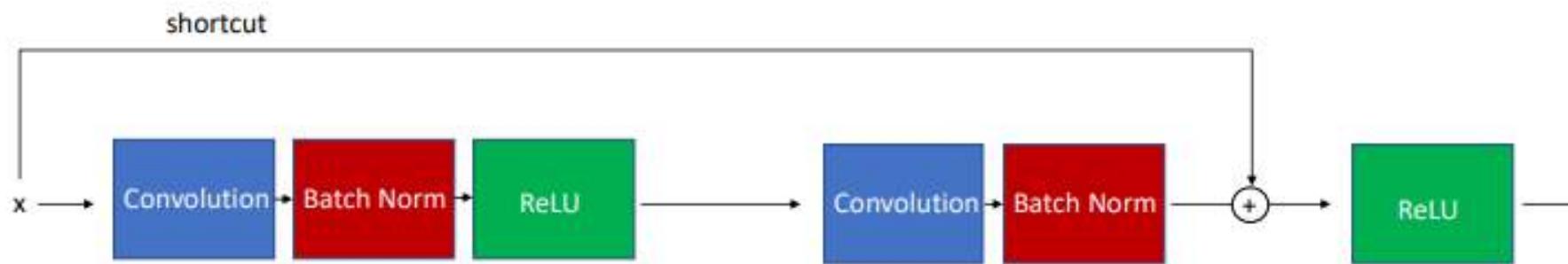


Figure 2. Residual learning: a building block.

<https://cv-tricks.com/keras/understand-implement-resnets/>

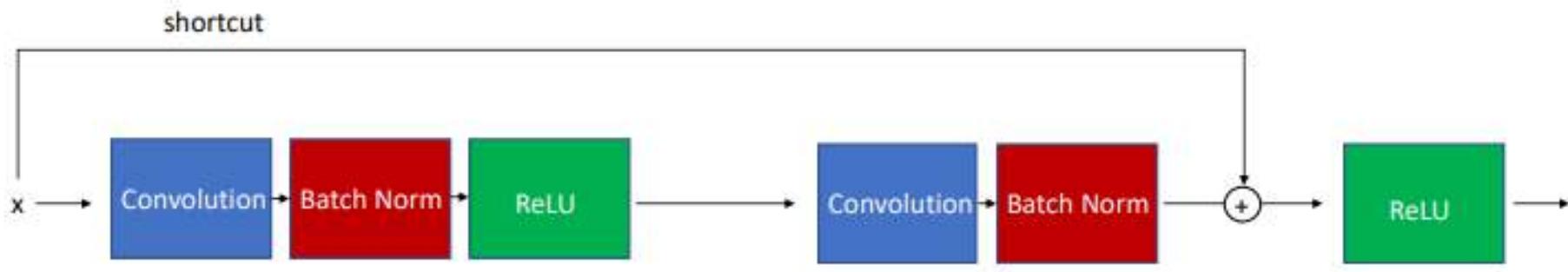
<https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035>

# ResNets



In general:  $a^{(l+2)} = \sigma(z^{(l+2)} + a^{(l)})$

# ResNets



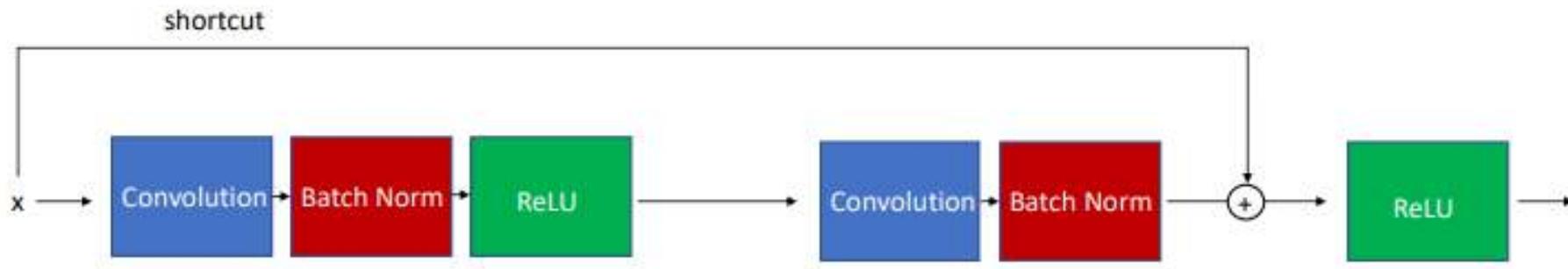
$$\begin{aligned}a^{(l+2)} &= \sigma(z^{(l+2)} + a^{(l)}) \\&= \sigma(a^{(l+1)}W^{(l+2)} + b^{(l+2)} + a^{(l)})\end{aligned}$$

If all weights and the bias are zero, then

$$= \sigma(a^{(l)}) = a^{(l)} \quad (\text{identity function})$$

due to ReLU

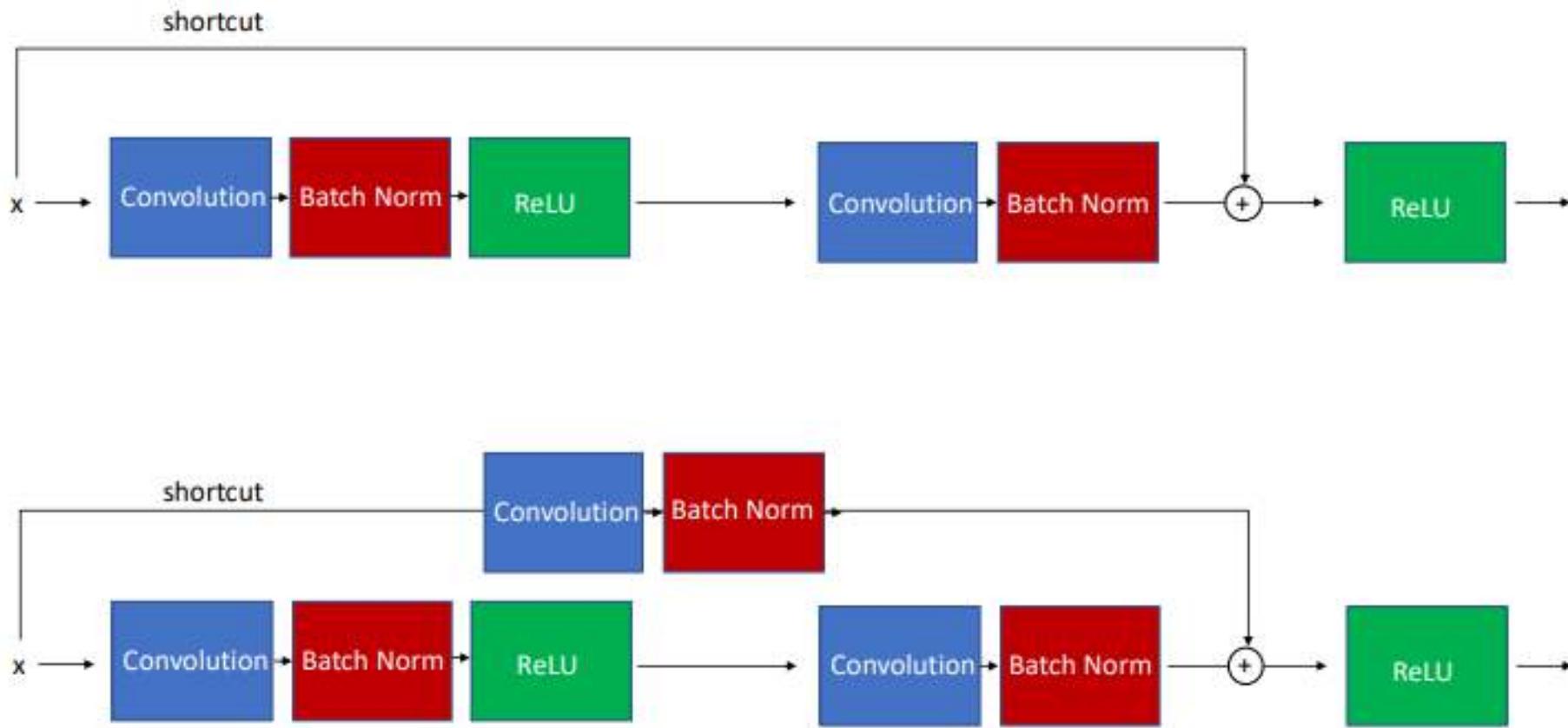
# ResNets



$$a^{(l+2)} = \sigma(z^{(l+2)} + a^{(l)})$$

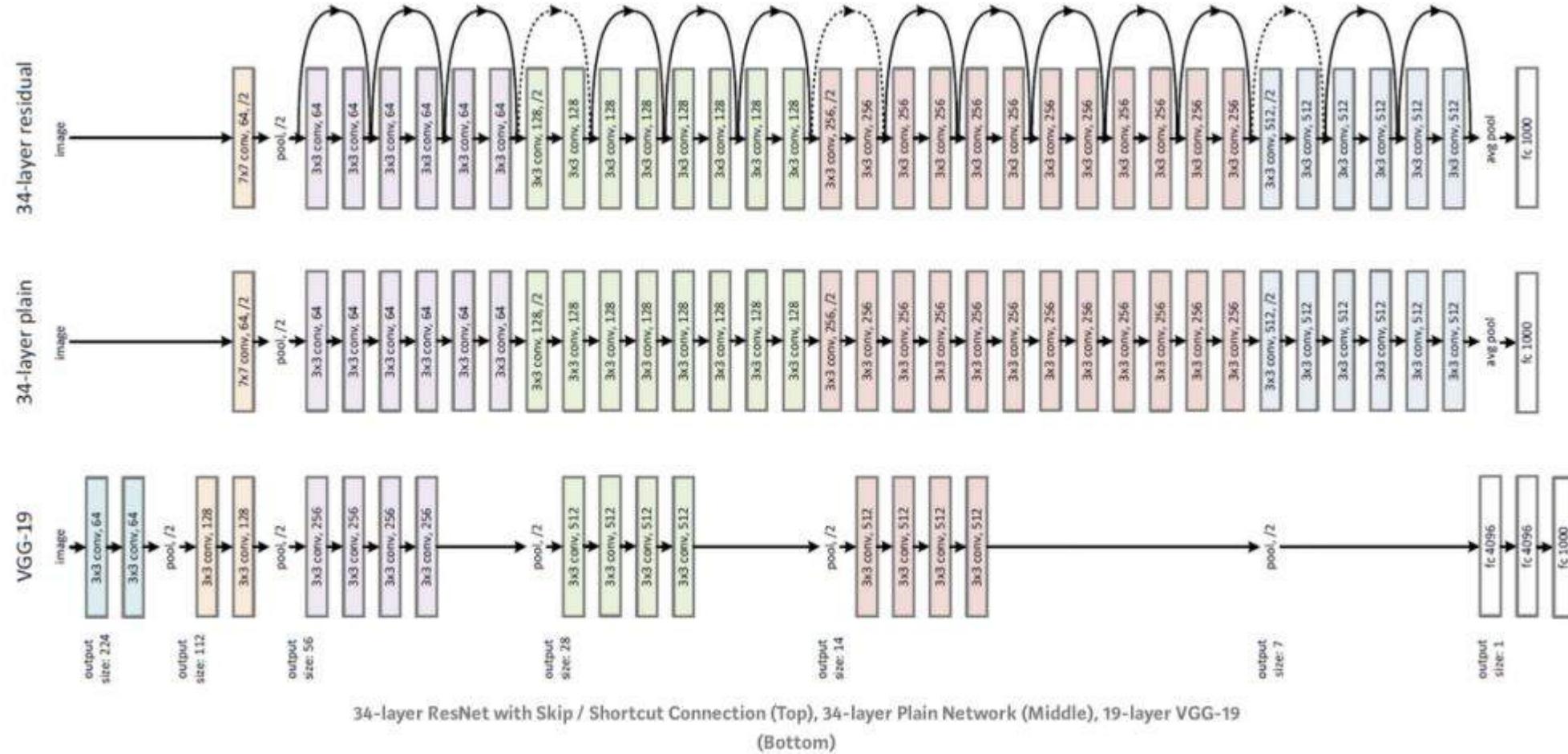
We assume these have the same dimension  
(e.g., via "same" convolution)

# ResNets



alternative residual blocks with skip connections such that the input passed via the shortcut is resized to dimensions of the main path's output

# ResNet Comparison



**152-layer deep net was better than human. Only 3.6% error rate ImageNet Classification**

Better than the 2nd best system ImageNet Detection: 16% ImageNet Localization: 27% COCO  
Detection: 11% COCO Segmentation: 12%

# ResNet Hyper-parameters and Issues

- Training takes huge time
- Batch Normalization
- Xavier/2 initialization
- SGD and momentum
- Small learning rate 0.1
- Mini-batch size 256
- Weight decay
- No Dropout

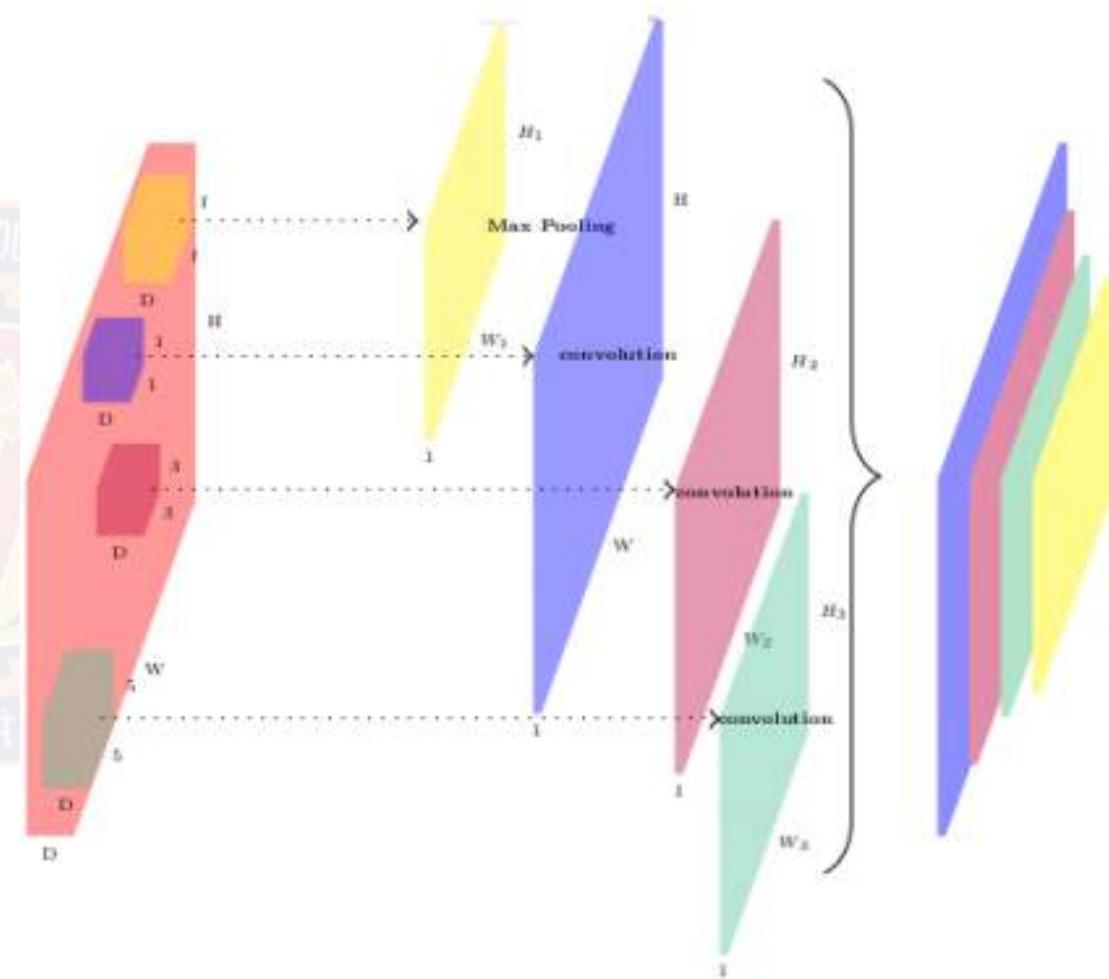
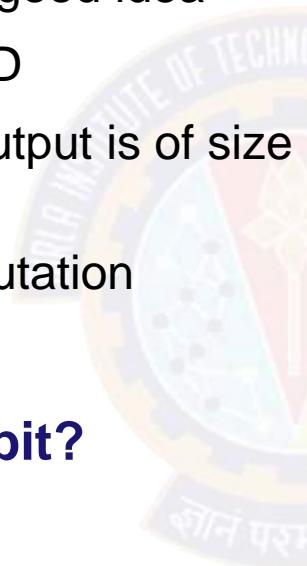


# GoogLeNet

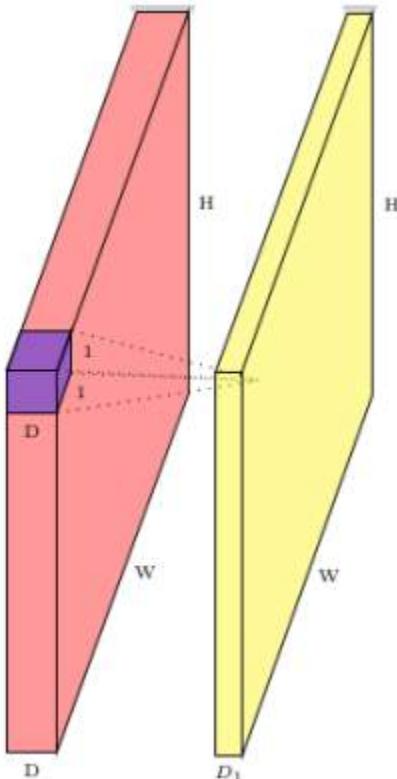
- Recall scale invariance in SIFT
- Multiple filters of different size is a good idea
- With  $W \times H \times D$  input and  $F \times F \times D$
- Filter and  $S = 1$  and no padding, output is of size  $(W - F + 1) \times (H - F + 1)$
- Each value needs  $F \times F \times D$  computation

Can we reduce this computation a bit?

- Idea is to have  $1 \times 1$  computation

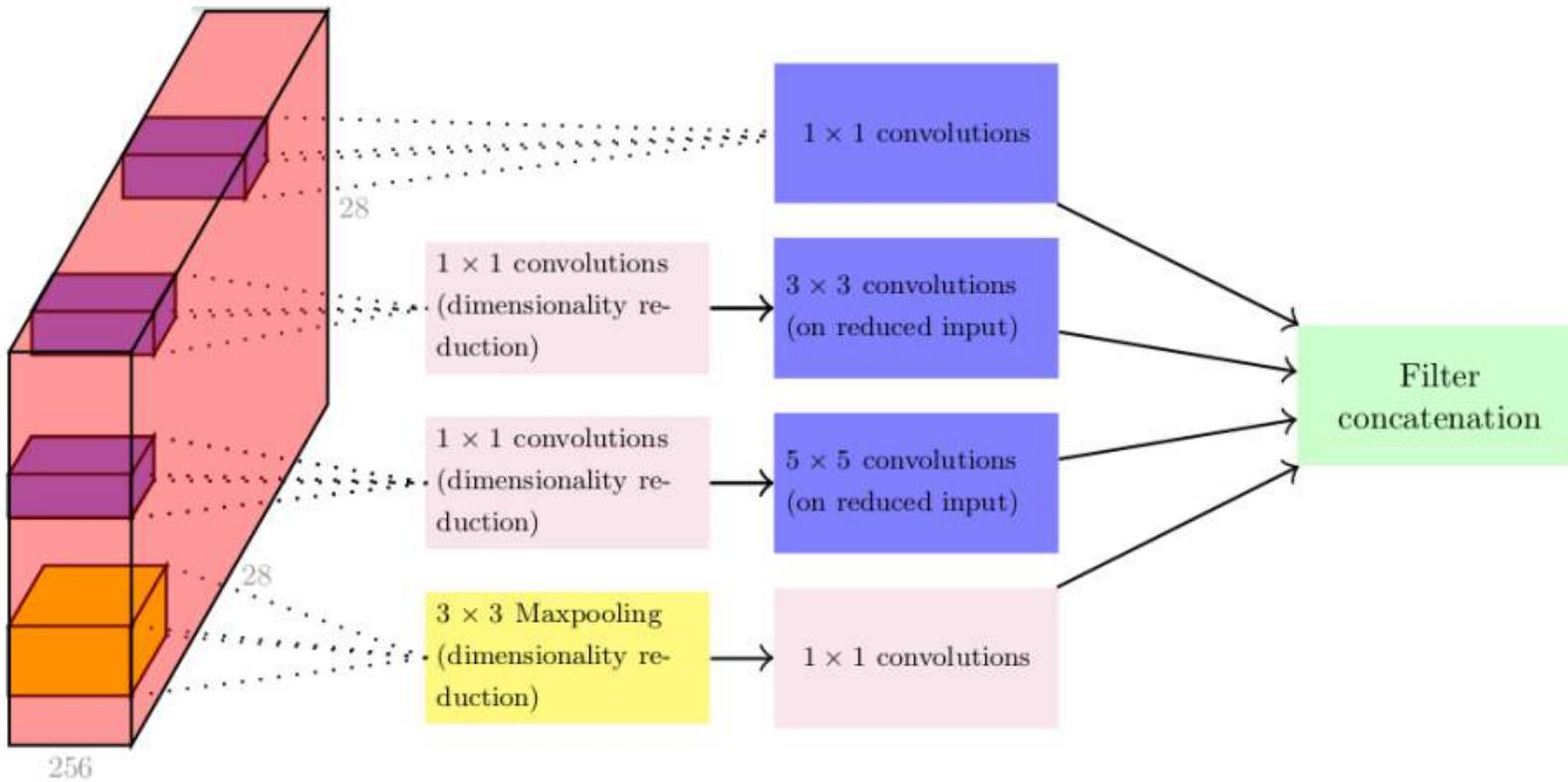


# $1 \times 1$ convolution



- $1 \times 1$  is  $1 \times 1 \times D$
- They produce one output place
- By using  $D_1$  such  $1 \times 1$  convolution output becomes  $F \times F \times D_1$
- We have  $D_1 < D$

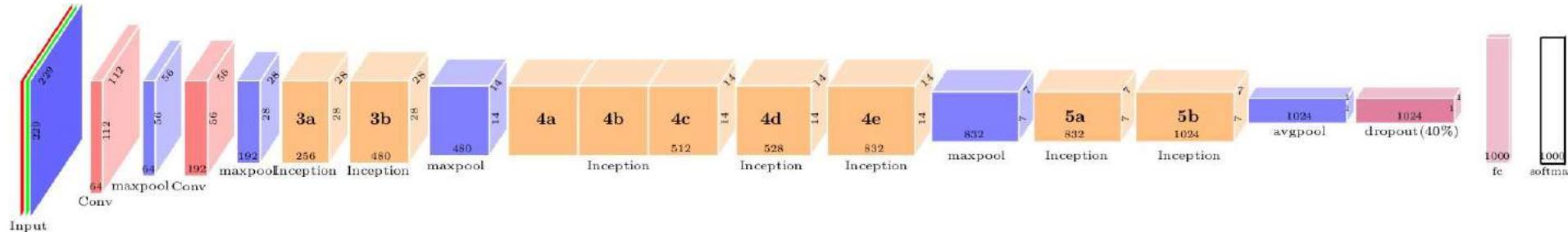
# Inception Block: Multiple convolutions



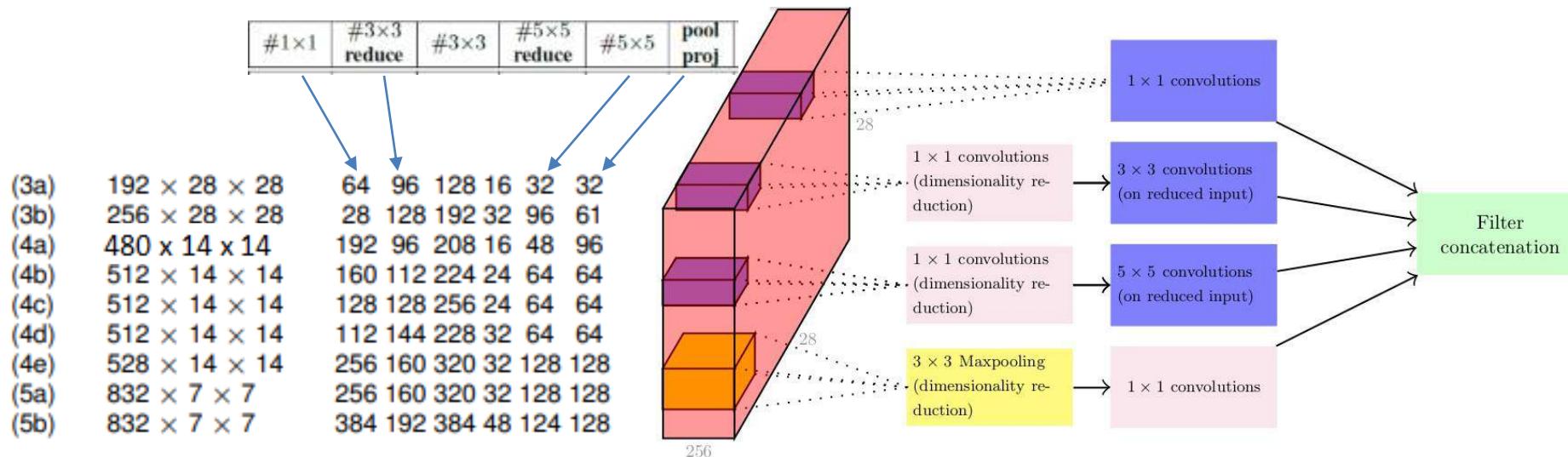
- $1 \times 1$  convolution
- $1 \times 1$  convolution followed by  $3 \times 3$
- $1 \times 1$  convolution followed by  $5 \times 5$
- $3 \times 3$  maxpool followed by  $1 \times 1$
- Appropriate padding is done to make things of same size

# GoogLeNet

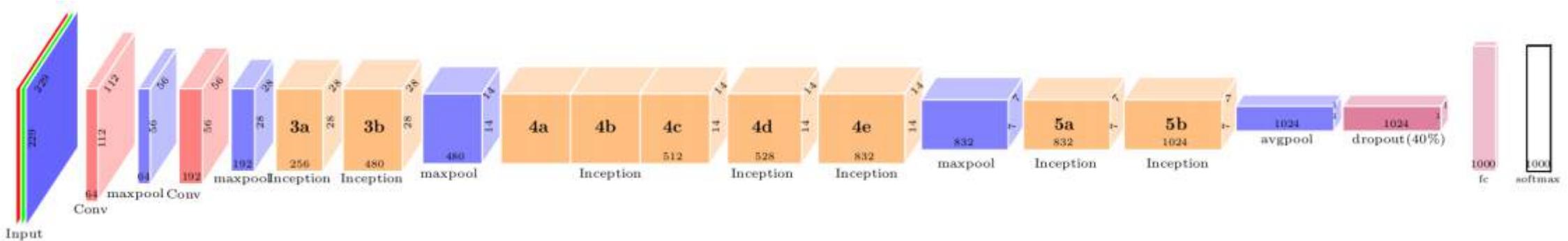
<https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvrc-2014-image-classification-c2b3565a64e7>



- 1 Input is RGB  $229 \times 229$
- 2 Each inception module have very specific configuration.



# GoogLeNet



- VGGNET has  $512 \times 7 \times 7$  size at pre-FC this was an issue to connect with 4096
- GoogLeNet applies a average pool. Gives 49 time reduction. has 1024 values only
- Dropout and connect to 1000
- 12 times less connections as compared to AlexNet
- 2 times more computation as compared to AlexNet
- Very high accuracy. Error reduced from 16% -to- 6.7%

# Object Detection and Localization

## Classification



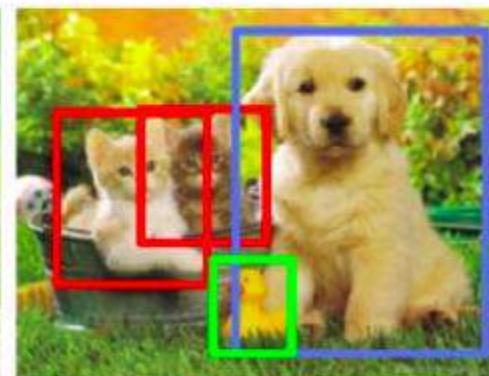
CAT

## Classification + Localization



CAT

## Object Detection



## CAT, DOG, DUCK

Instance  
Segmentation



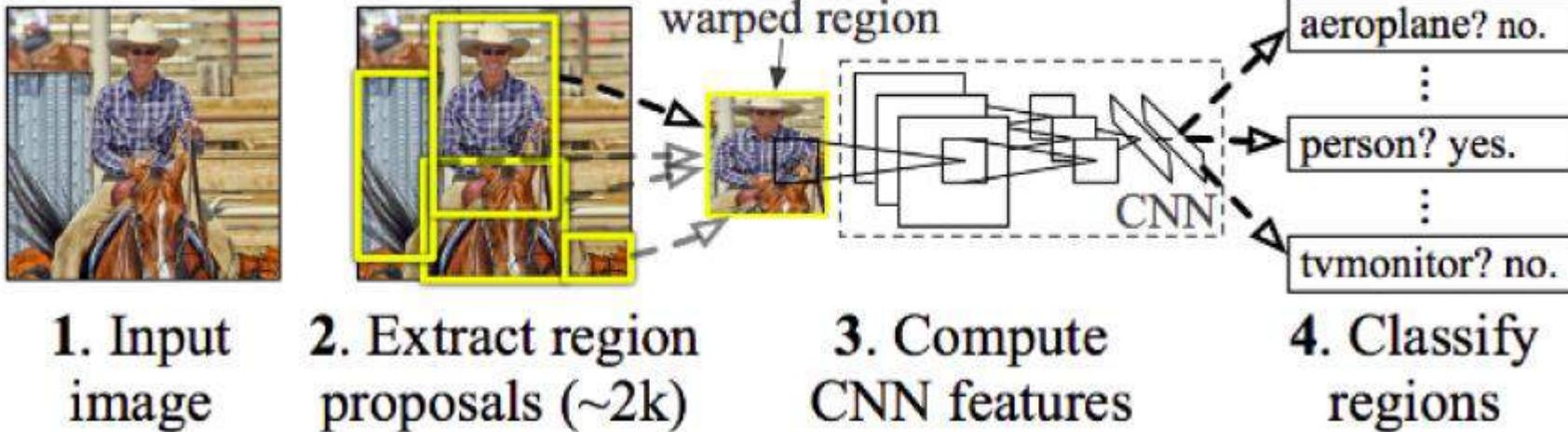
CAT, DOG, DUCK

## Single object

## Multiple objects

# R-CNN

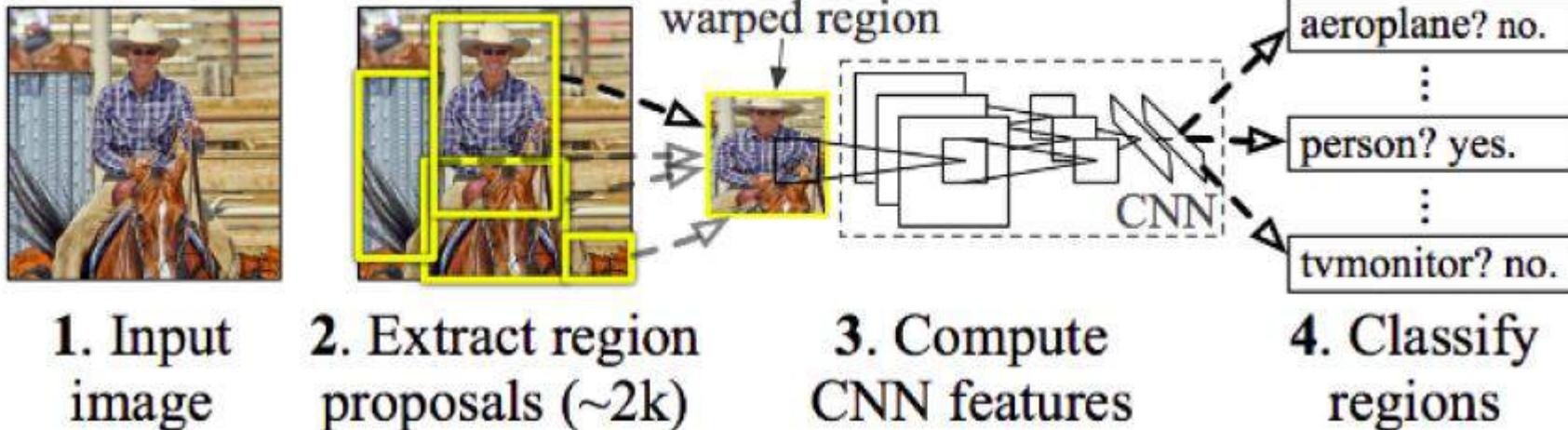
## R-CNN: *Regions with CNN features*



- Region proposals 20K (from external [selective search](#))
- Warped image (resizing)
- SVM for classification (one for each class)

# R-CNN

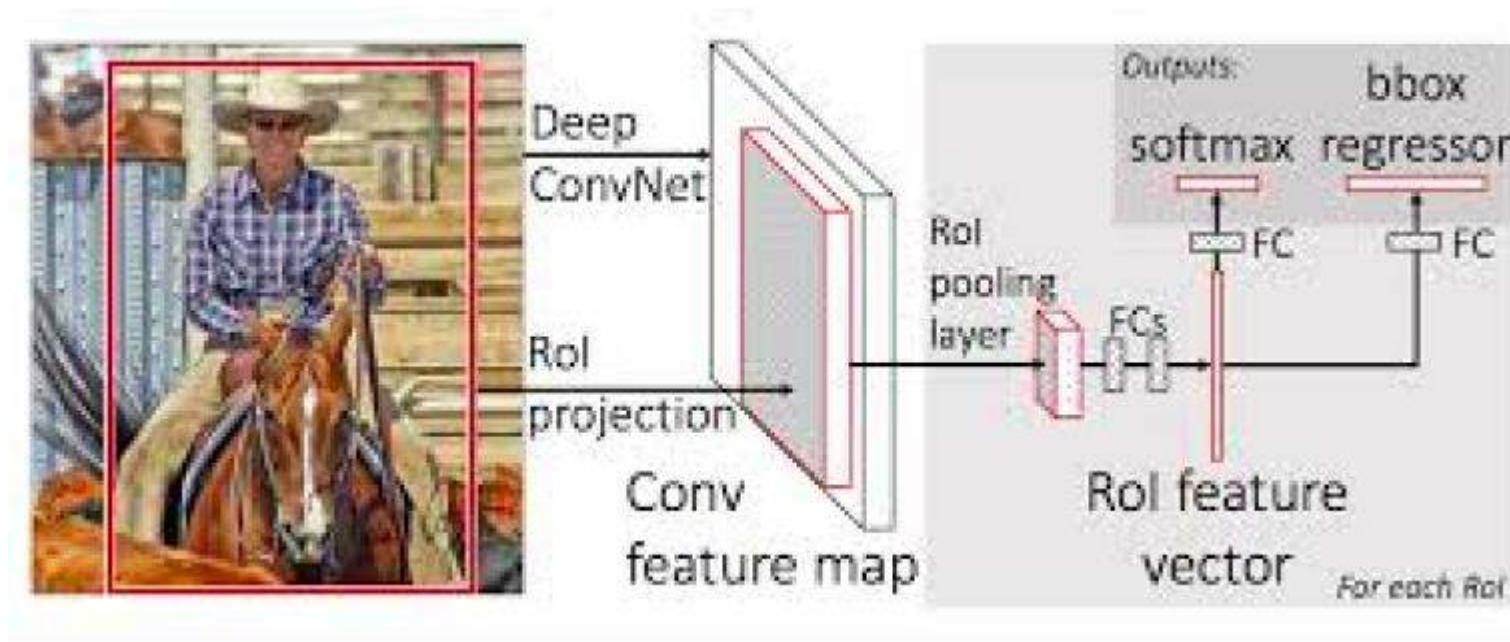
## R-CNN: *Regions with CNN features*



- Region proposals (from external **selective search**)
- Warped image (resizing)
- SVM for classification (one for each class)



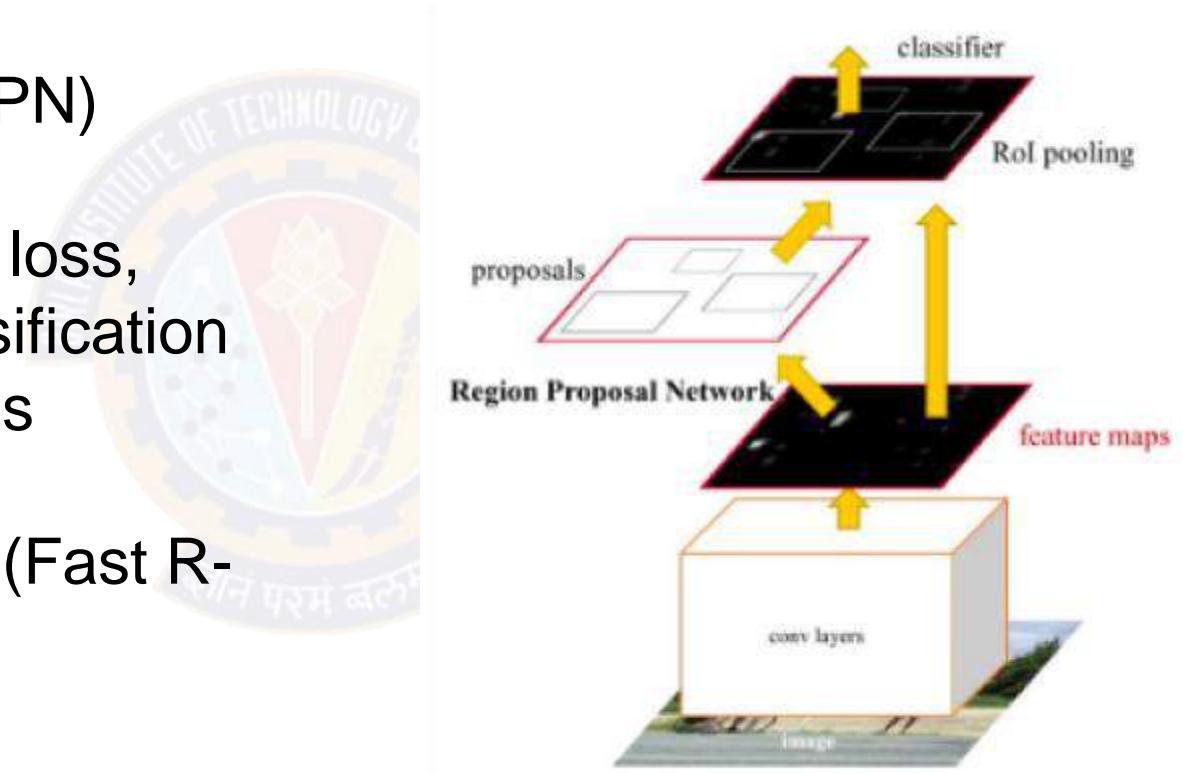
# Fast R-CNN



- RoI pooling
- Multi-task loss

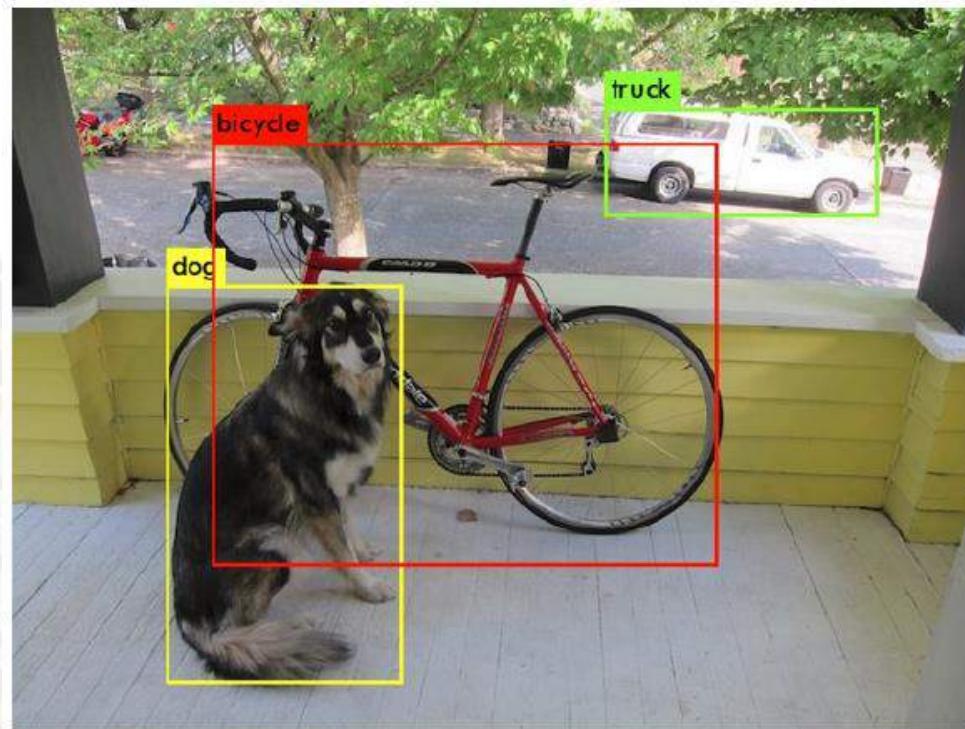
# Faster R-CNN

- Region Proposal Network (RPN)
- Four loss: RPN classification loss, RPN regress loss, Final classification loss, Final box coordinate loss
- 250 time faster than R-CNN. (Fast R-CNN is 25 times fast)



# Object Detection with Yolo

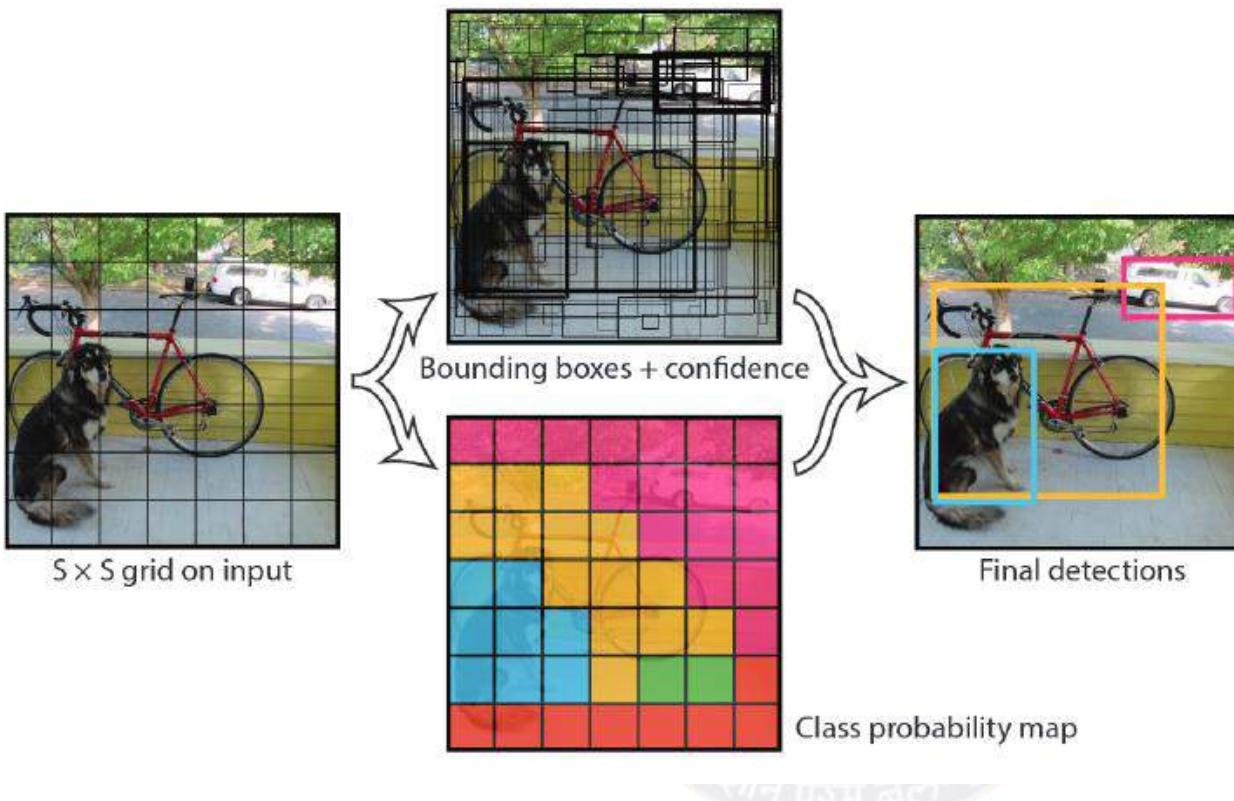
- What is there and where?
- Deformative Part Model, and F-RCNN



- Apply the model to an image at multiple locations and scales.
- High scoring regions are considered detections.

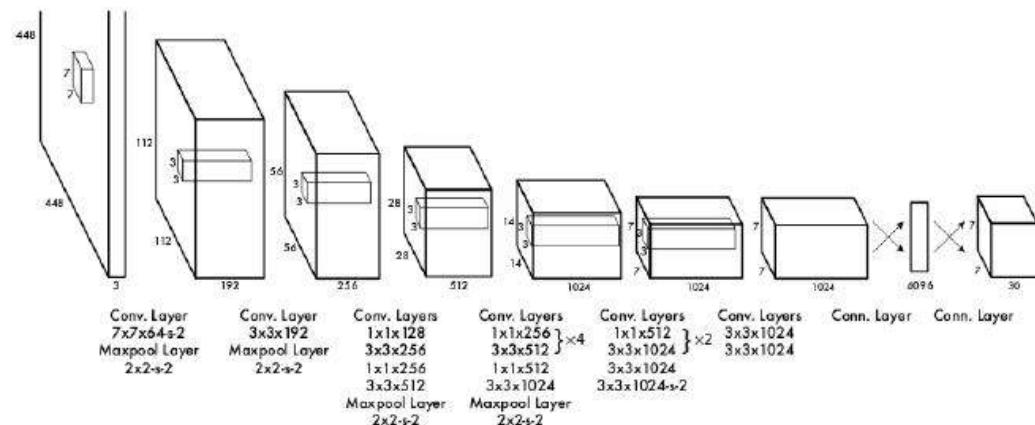
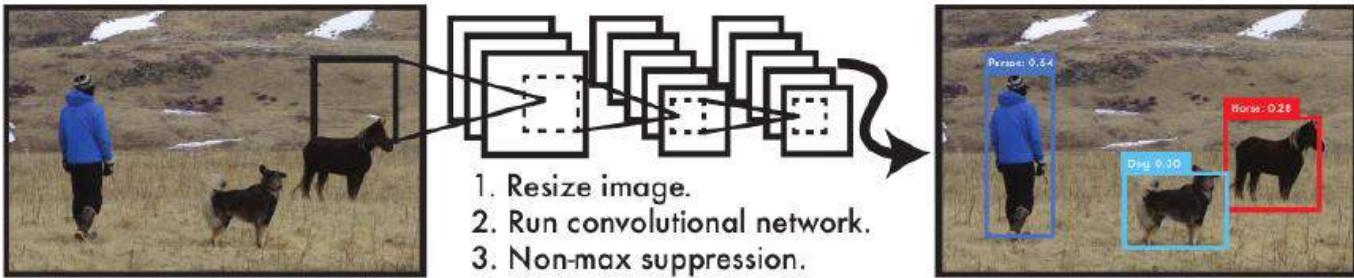
**Yolo:** apply a single neural network to the full image that divides it into regions and predicts bounding boxes and probabilities for each region

# Yolo



- Conditional probability map
- See <https://pjreddie.com/darknet/yolo/>

# Yolo



- SxS segments, gives B bounding boxes with confidence, and C class probabilities. So  $S \times S \times (B \times 5 + C)$  values. S:7, B:2, C:20

# Yolo

	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	63.4	45 FPS	22 ms/img

- It is fast
- Speed comes at the price of accuracy. Improved to 69%
- Generalizes well
- Latest version YOLOv3 2018



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Background of NN

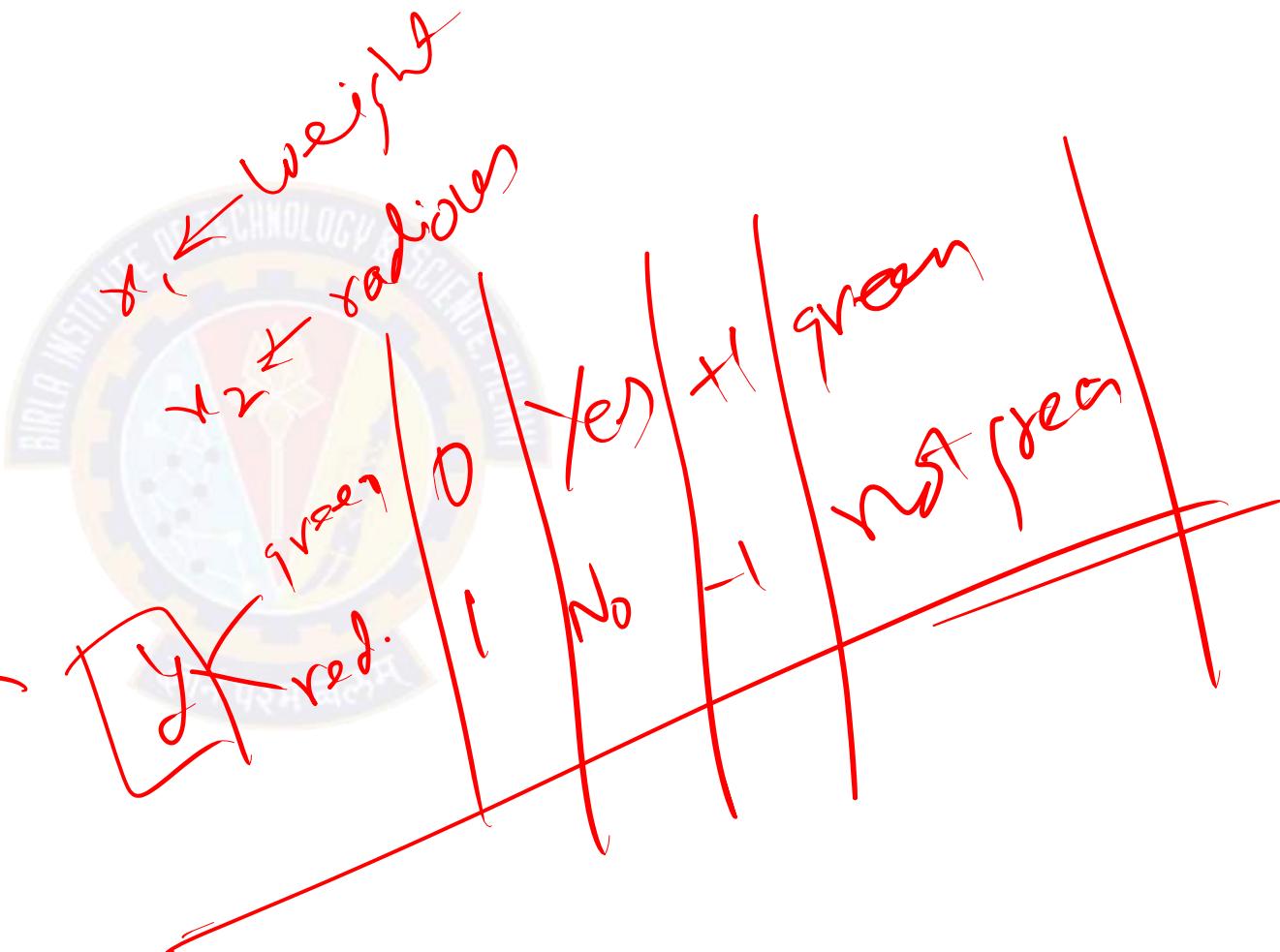
**Dr. Kamlesh Tiwari**

# Linear Classification

## Consider Following data

**Consider Following data**

$x_1$	$x_2$	$y$
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red

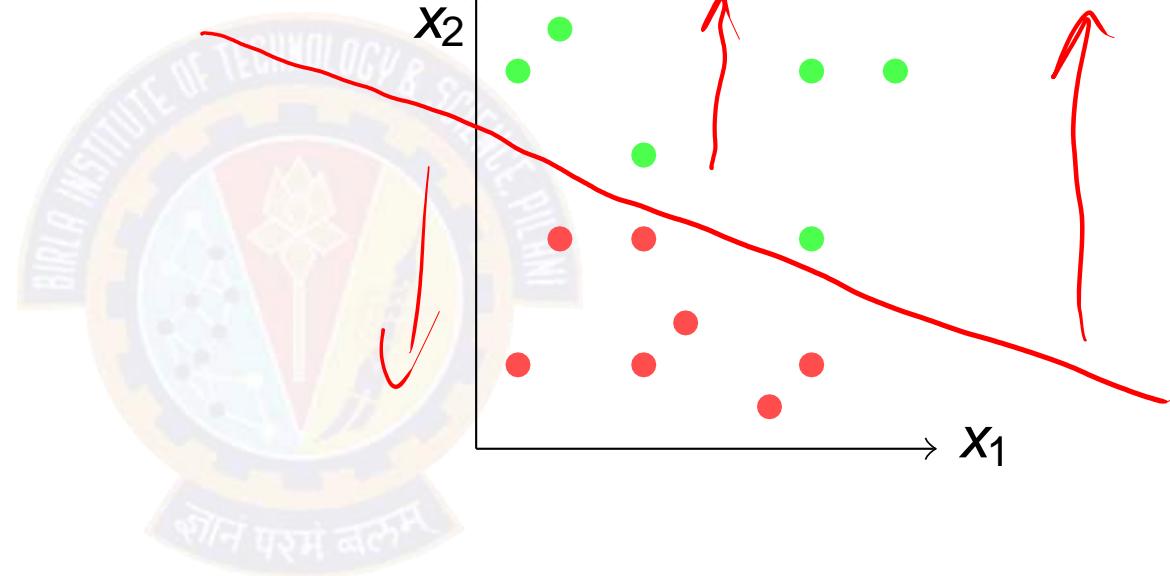


# Linear Classification

Consider Following data

$x_1$	$x_2$	y
1	9	Green ✕
10	9	Green ✕ \
4	7	Green ✕ \
4	5	Red -\
5	3	Red -\
8	9	Green
4	2	Red -\
2	5	Red -\
7	1	Red -\
2	10	Green
8	5	Green
1	2	Red
8	2	Red

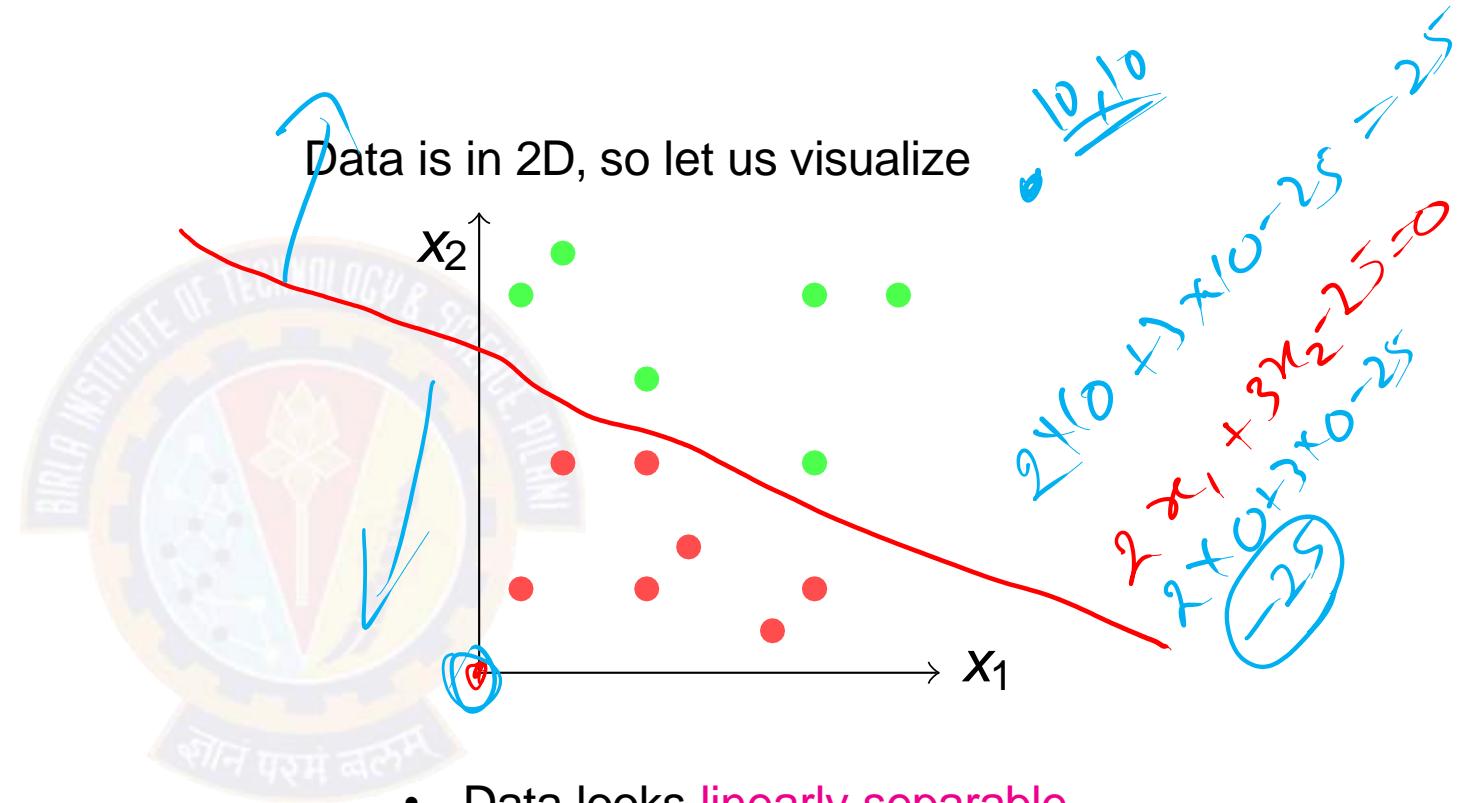
Data is in 2D, so let us visualize



# Linear Classification

Consider Following data

$x_1$	$x_2$	y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red



- Data looks **linearly separable**
- What is the decision boundary?

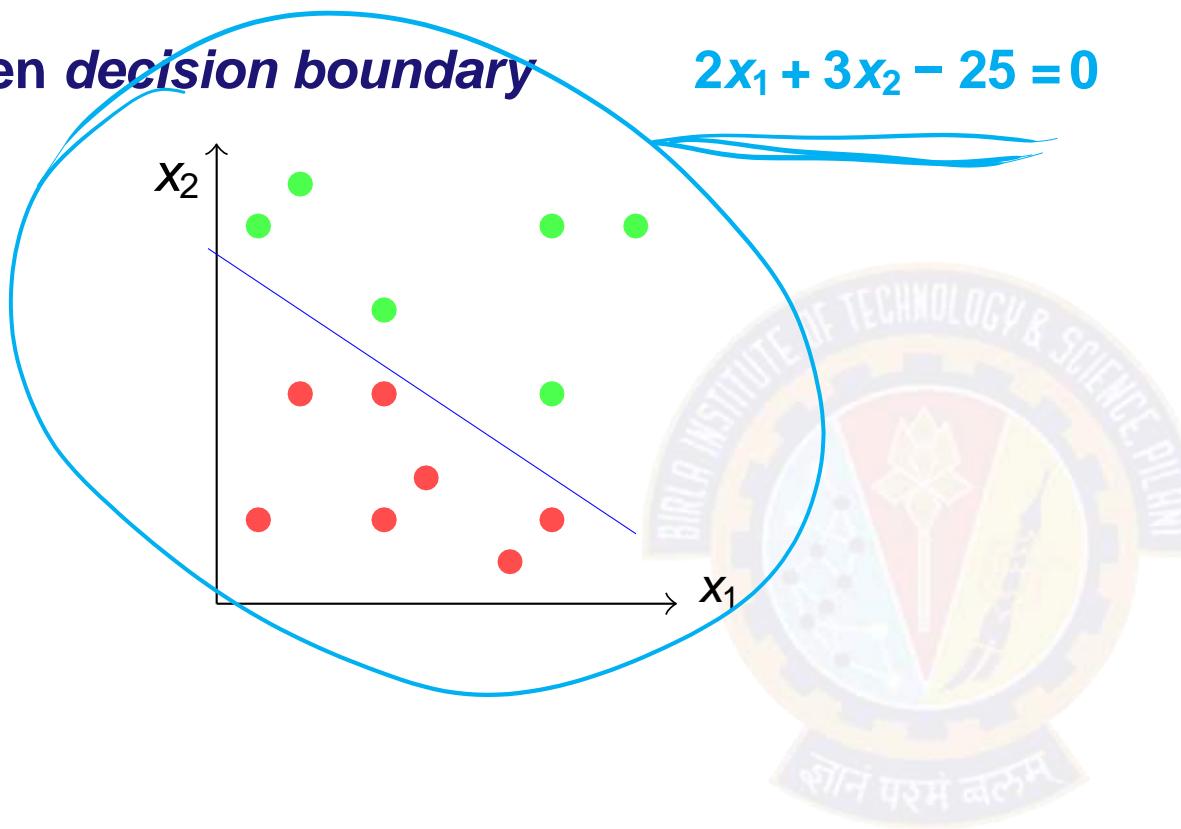
Many Possibilities, such as

if  $(2x_1 + 3x_2 - 25 > 0)$  it is **green**  
otherwise **red**

# What about this arrangement?

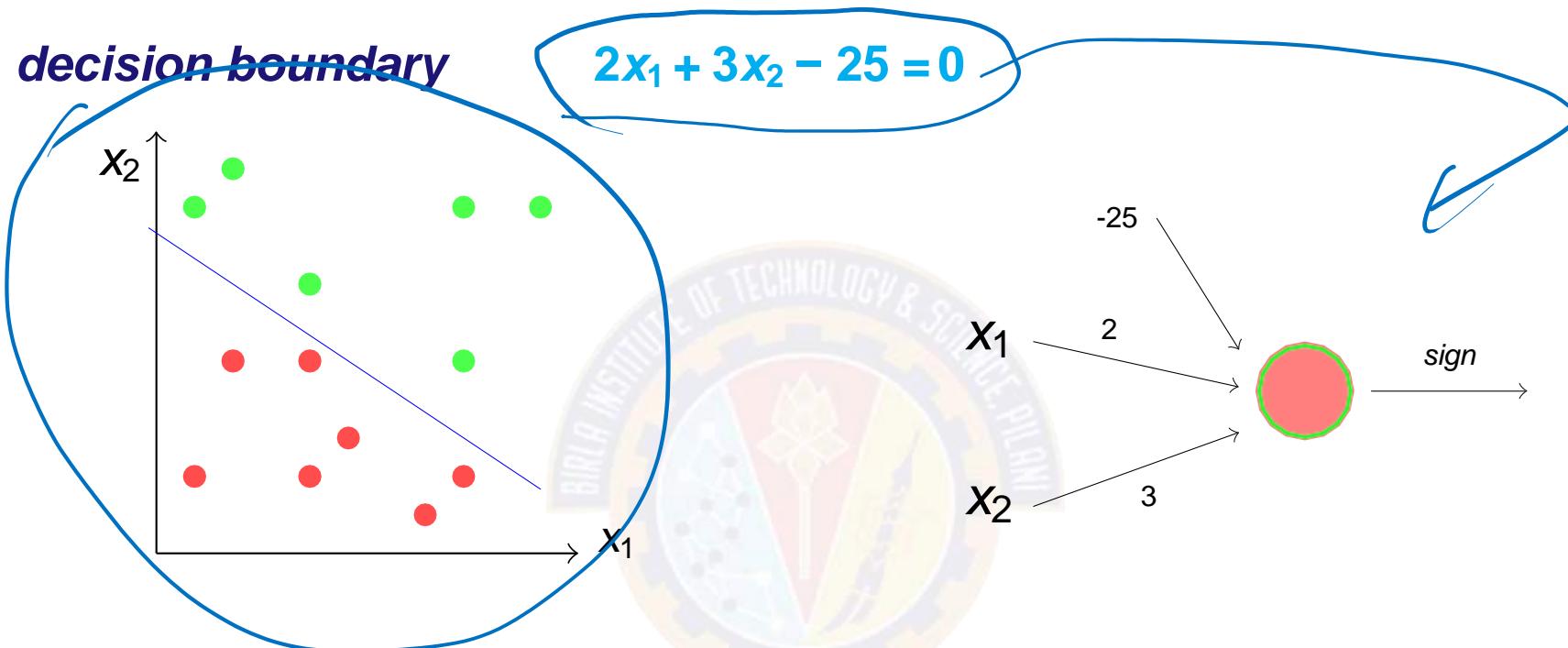
With chosen *decision boundary*

$$2x_1 + 3x_2 - 25 = 0$$



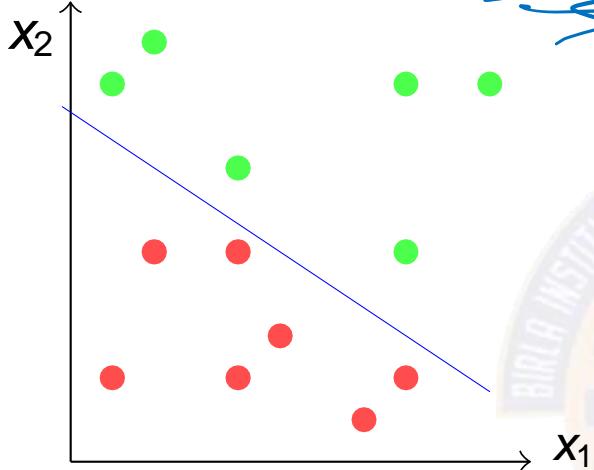
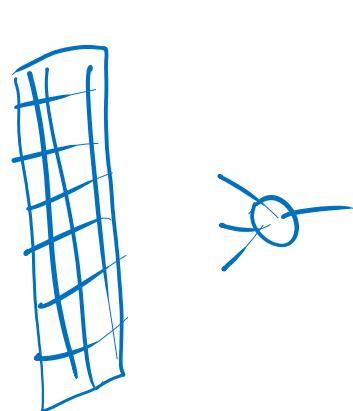
# What about this arrangement?

With chosen *decision boundary*



# What about this arrangement?

With chosen *decision boundary*



$$2x_1 + 3x_2 - 25 = 0$$

$x_1$

$x_2$

-25

2

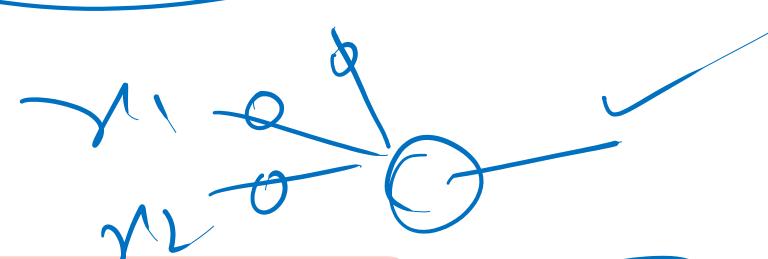
3

sign

- This illustration is called as **perceptron**
- Provides a graphical way to represent the linear boundary
- Values **3, 2, -25** are its parameters or weights

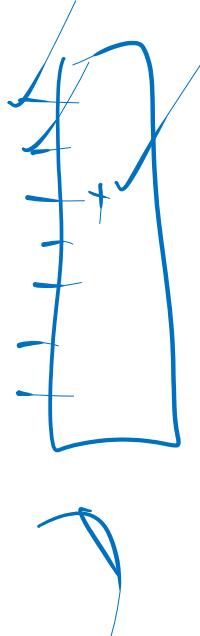
Given a data

“How to find appropriate parameters?” is an important **issue**



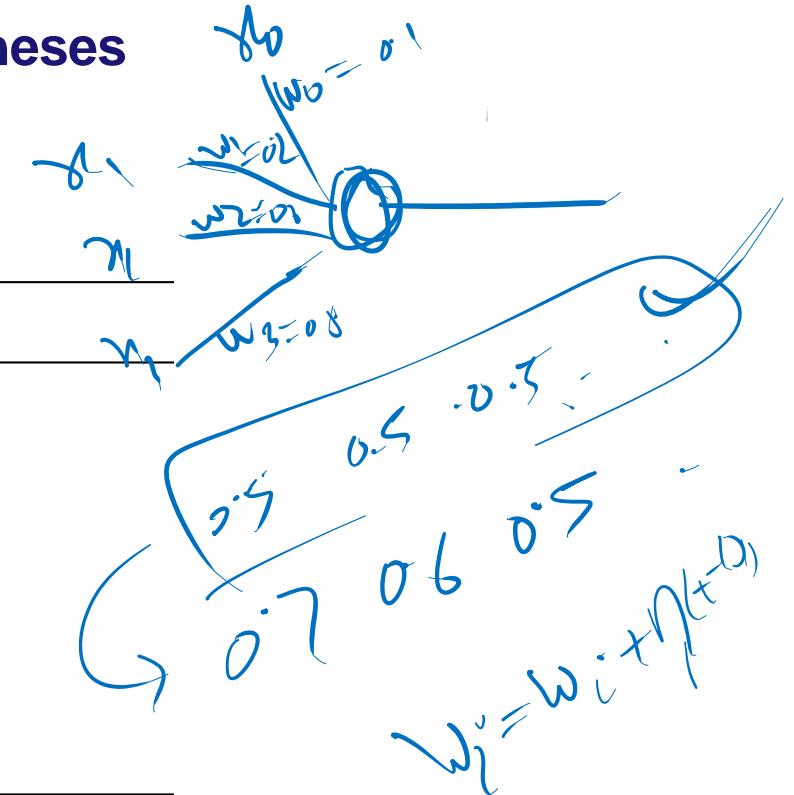
# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses



## Algorithm 1: Perceptron training rule

- 1 Begin with **random** weights  $w$
- 2 repeat
- 3   for each **misclassified** example do
- 4      $w_i = w_i + \eta(t - o)x_i$
- 5 until **all training examples are correctly classified;**
- 6 return  $w$



# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

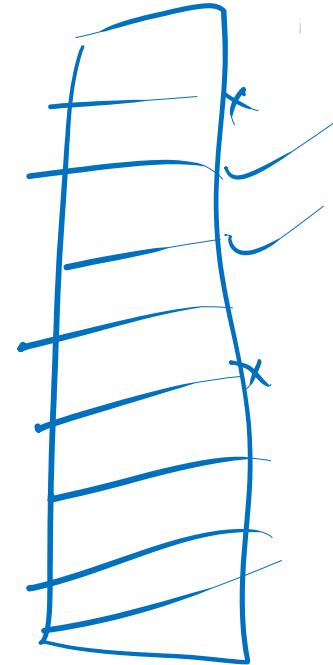
---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

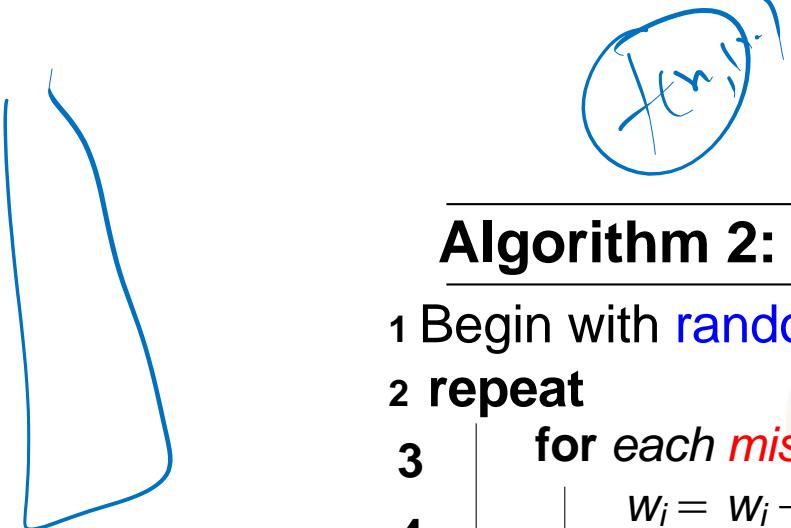
---



- Why would this strategy converge?
  - Weight does not change when classification is correct

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses



## Algorithm 2: Perceptron training rule

- 1 Begin with **random** weights  $w$
- 2 **repeat**
- 3     **for each *misclassified* example do**
- 4          $w_i = w_i + \eta(t - o)x_i$
- 5     **until all training examples are correctly classified;**
- 6 **return**  $w$

$$v_i = w_i + n(t - o)x_i$$
$$w_i = w_i + n o_i$$
$$w_i = w_i + \Delta$$

- Why would this strategy converge?
  - Weight does not change when classification is correct
  - If perceptron outputs  $-1$  when target is  $+1$ :

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?
  - Weight does not change when classification is correct
  - If perceptron outputs -1 when target is +1: weight increases ↑

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

## Algorithm 2: Perceptron training rule

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

- Why would this strategy converge?

- Weight does not change when classification is correct
- If perceptron outputs  $-1$  when target is  $+1$ : weight increases ↑
- If perceptron outputs  $+1$  when target is  $-1$ :

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?
  - Weight does not change when classification is correct
  - If perceptron outputs -1 when target is +1: weight increases ↑
  - If perceptron outputs +1 when target is -1: weight decreases ↓

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- **Why would this strategy converge?**

- Weight does not change when classification is correct
- If perceptron outputs -1 when target is +1: weight increases ↑
- If perceptron outputs +1 when target is -1: weight decreases ↓

Conversion with perceptron training rule is subject to linear separability of training example and appropriate  $\eta$

# Example

Consider the same data

X <sub>1</sub>	X <sub>2</sub>	Y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red



# Example

Consider the same data

X <sub>1</sub>	X <sub>2</sub>	Y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red

$$(w_0 + w_1x_1 + w_2x_2)$$

$$\eta = 0.01$$

$$w_0=0.500, w_1=0.500, w_2=0.500 \quad \text{err}=7$$

$$w_0=0.360, w_1=-0.120, w_2=0.100 \quad \text{err}=6$$

$$w_0=0.300, w_1=-0.180, w_2=0.060 \quad \text{err}=5$$

$$w_0=0.240, w_1=-0.140, w_2=0.140 \quad \text{err}=4$$

$$w_0=0.180, w_1=-0.200, w_2=0.100 \quad \text{err}=5$$

$$w_0=0.120, w_1=-0.160, w_2=0.180 \quad \text{err}=4$$

$$w_0=0.080, w_1=-0.060, w_2=0.180 \quad \text{err}=5$$

$$w_0=0.020, w_1=-0.120, w_2=0.140 \quad \text{err}=4$$

$$w_0=-0.040, w_1=-0.180, w_2=0.100 \quad \text{err}=5$$

$$w_0=-0.100, w_1=-0.140, w_2=0.180 \quad \text{err}=4$$

$$w_0=-0.140, w_1=-0.040, w_2=0.180 \quad \text{err}=5$$

$$w_0=-0.200, w_1=-0.100, w_2=0.140 \quad \text{err}=3$$

$$w_0=-0.260, w_1=-0.160, w_2=0.100 \quad \text{err}=4$$

$$w_0=-0.320, w_1=-0.120, w_2=0.180 \quad \text{err}=3$$

$$w_0=-0.360, w_1=-0.020, w_2=0.180 \quad \text{err}=3$$

$$w_0=-0.420, w_1=-0.080, w_2=0.140 \quad \text{err}=2$$

$$w_0=-0.420, w_1=-0.080, w_2=0.240 \quad \text{err}=2$$

Fourteen more iterations

$$w_0=-0.900, w_1=-0.020, w_2=0.180 \quad \text{err}=1$$

$$w_0=-0.900, w_1=-0.020, w_2=0.240 \quad \text{err}=2$$

$$w_0=-0.920, w_1=0.020, w_2=0.220 \quad \text{err}=2$$

$$w_0=-0.960, w_1=-0.020, w_2=0.220 \quad \text{err}=3$$

$$w_0=-0.980, w_1=0.020, w_2=0.200 \quad \text{err}=2$$

$$w_0=-1.000, w_1=0.060, w_2=0.180 \quad \text{err}=2$$

$$w_0=-1.040, w_1=0.020, w_2=0.180 \quad \text{err}=0$$

$$w_0=0.5$$

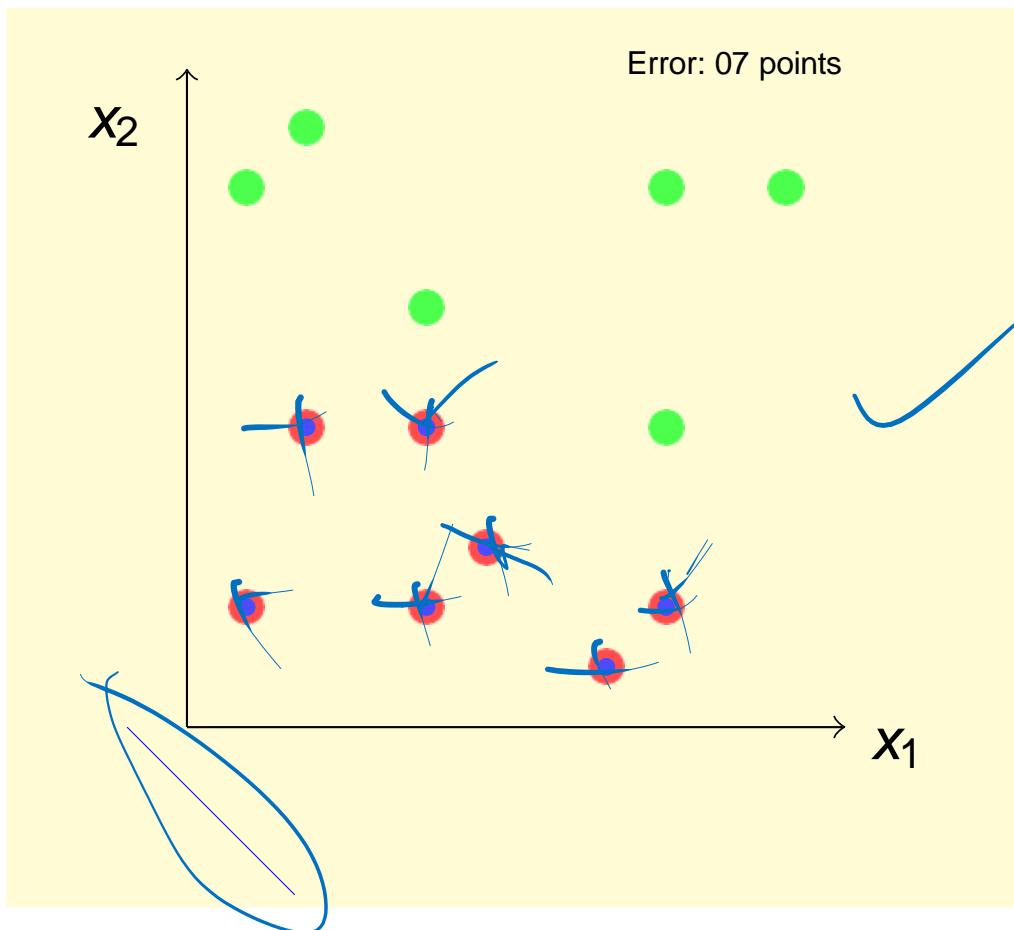
$$w_1=0.5$$

$$w_2=0.5$$

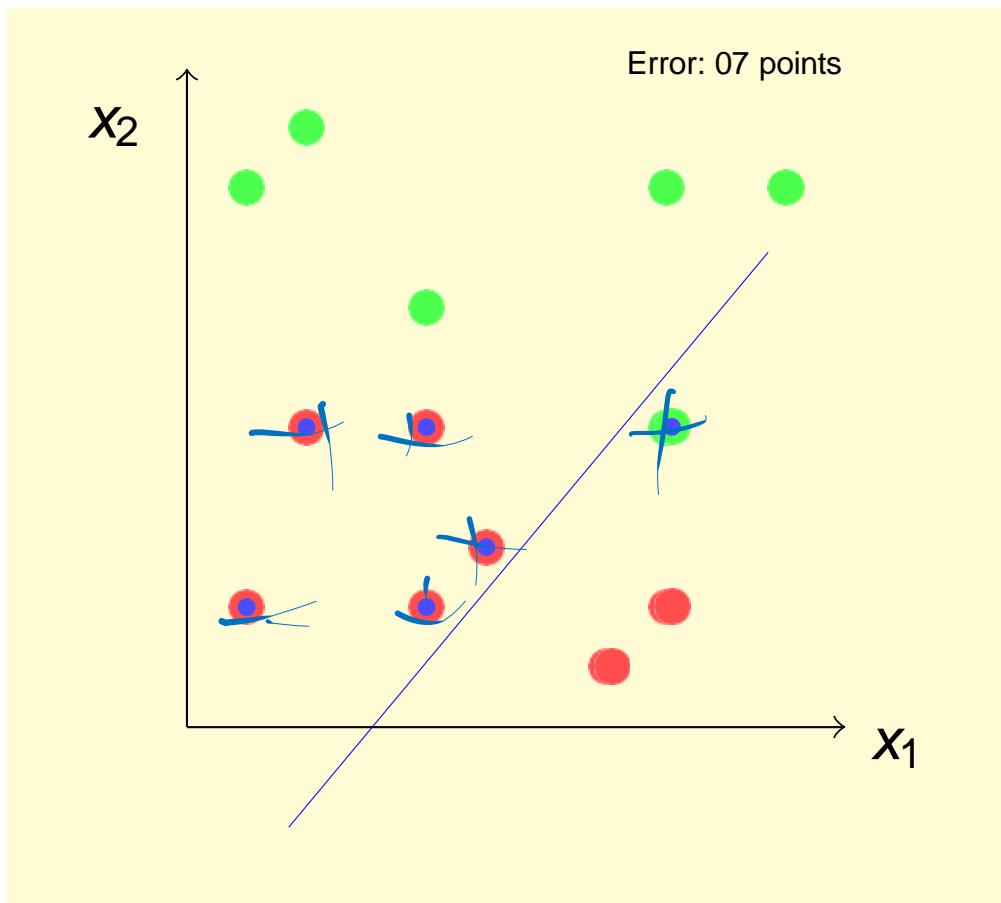
$$0.5 + 0.5x_1 + 0.5x_2$$

$$0.360 + 0.12x_1 + 0.5x_2$$

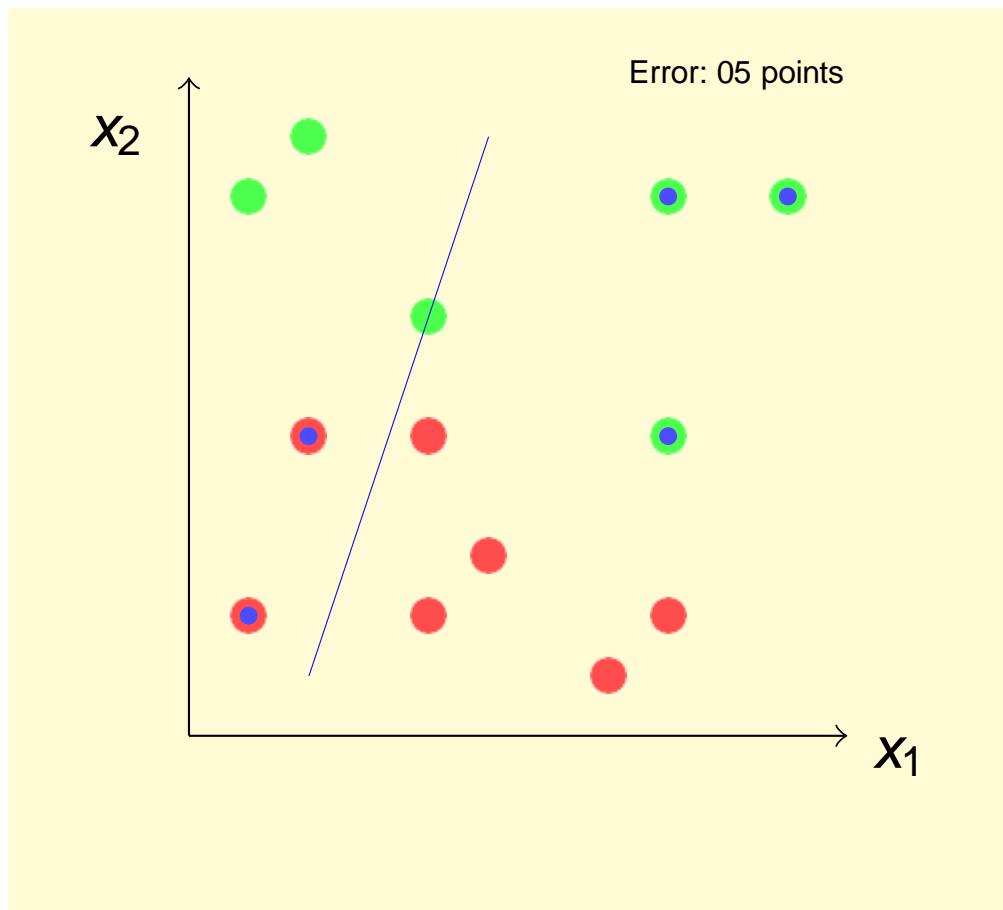
# Visual Interpretation



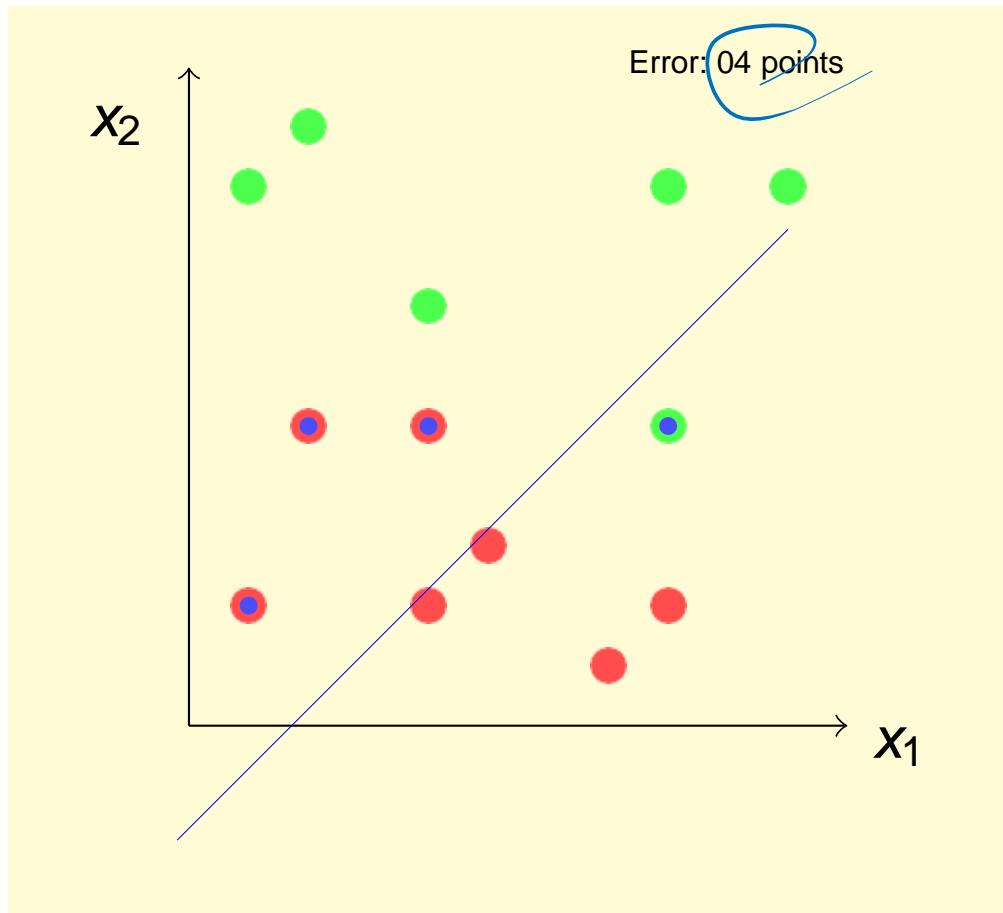
# Visual Interpretation



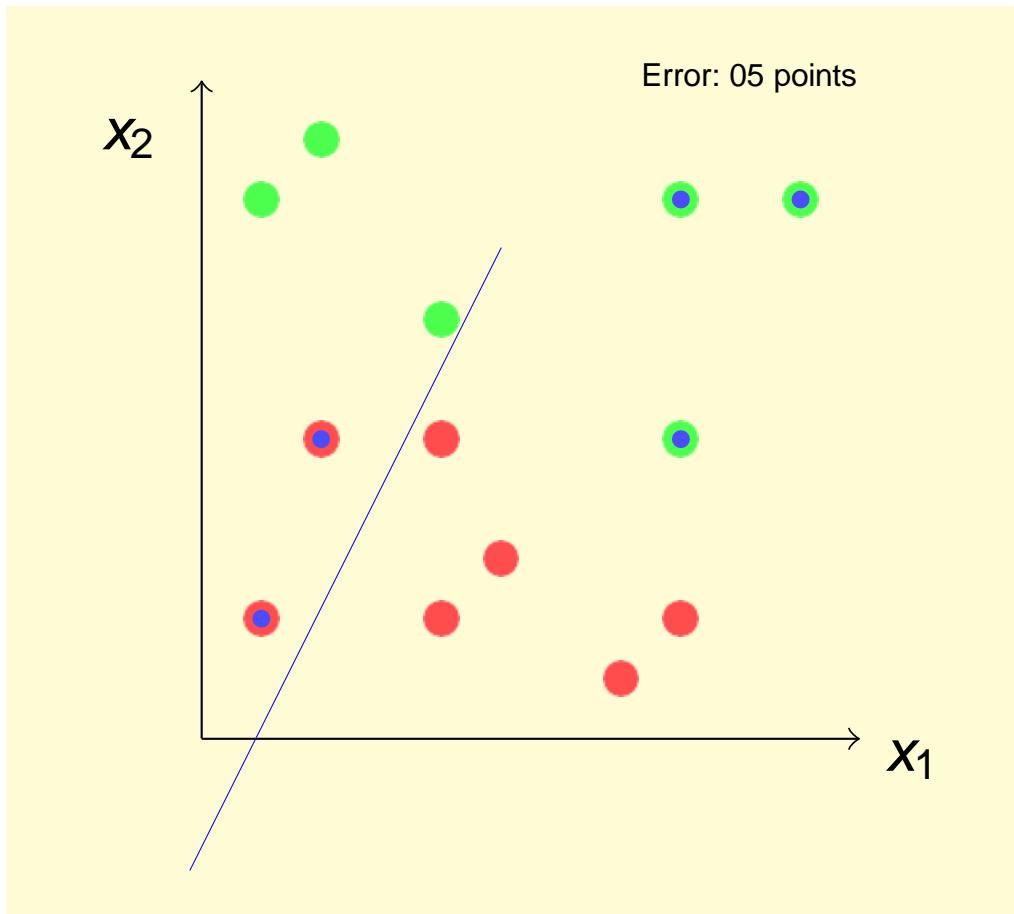
# Visual Interpretation



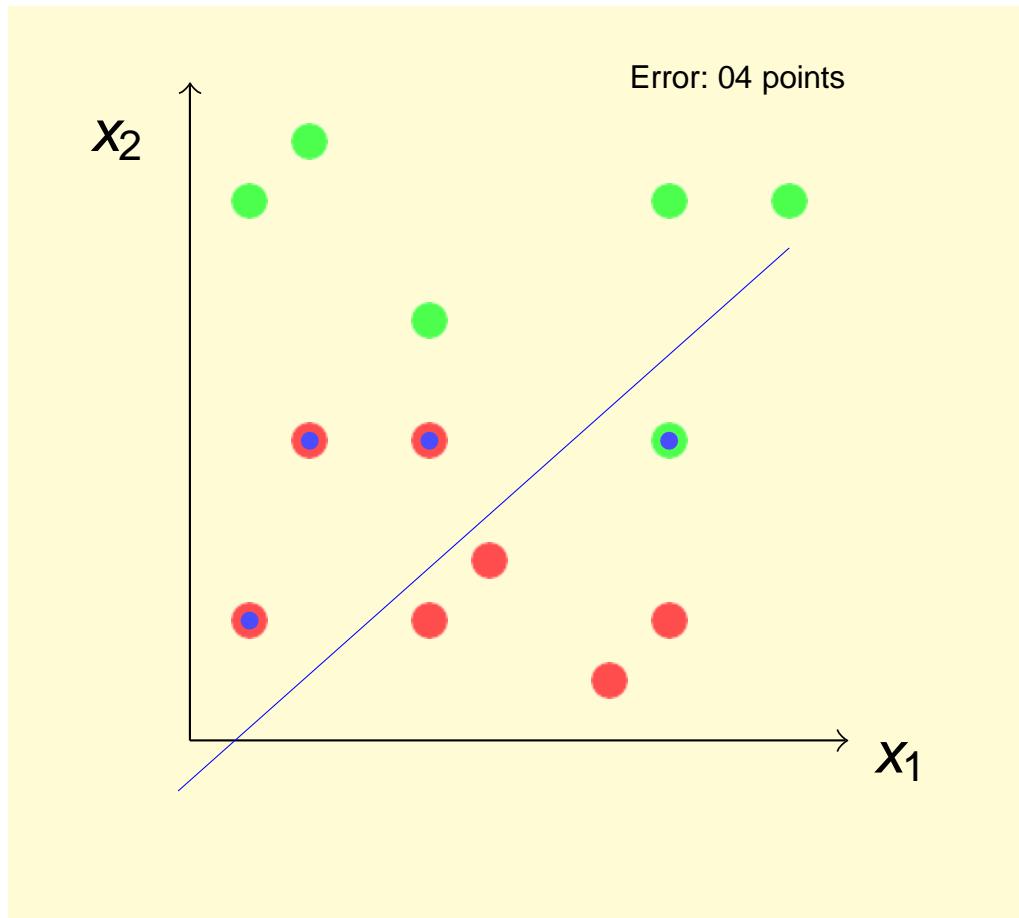
# Visual Interpretation



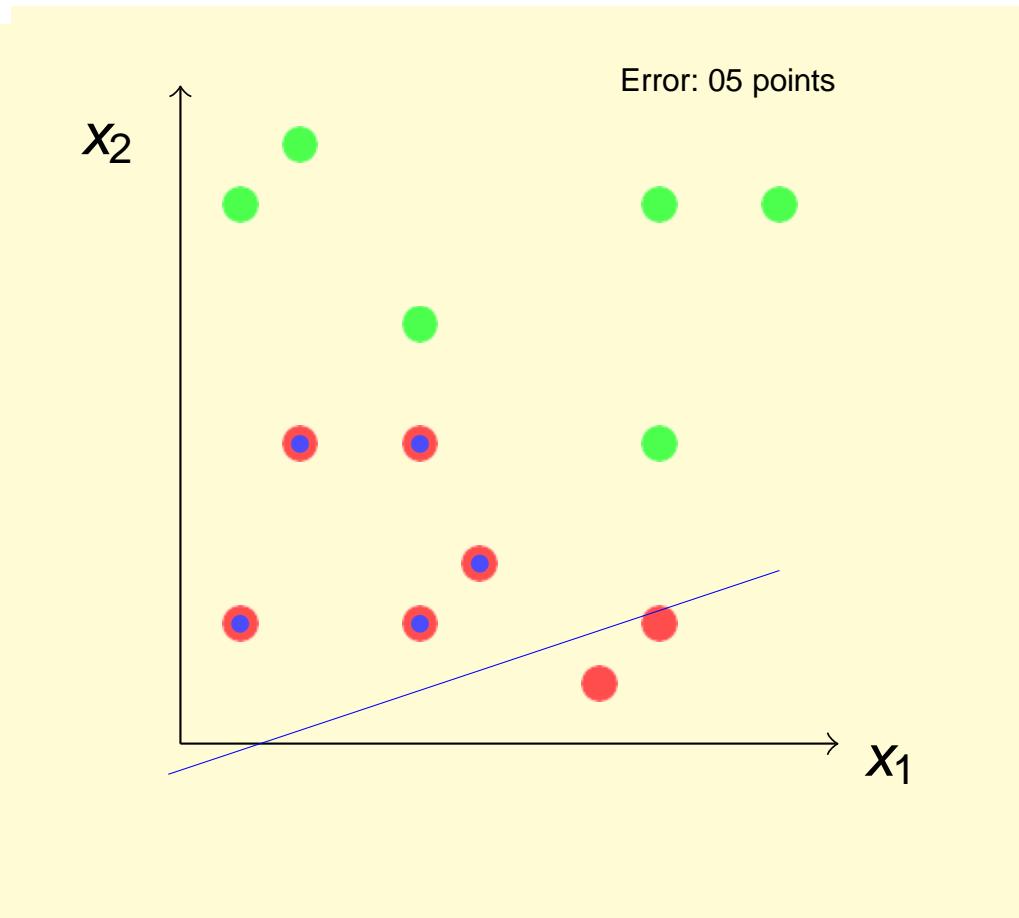
# Visual Interpretation



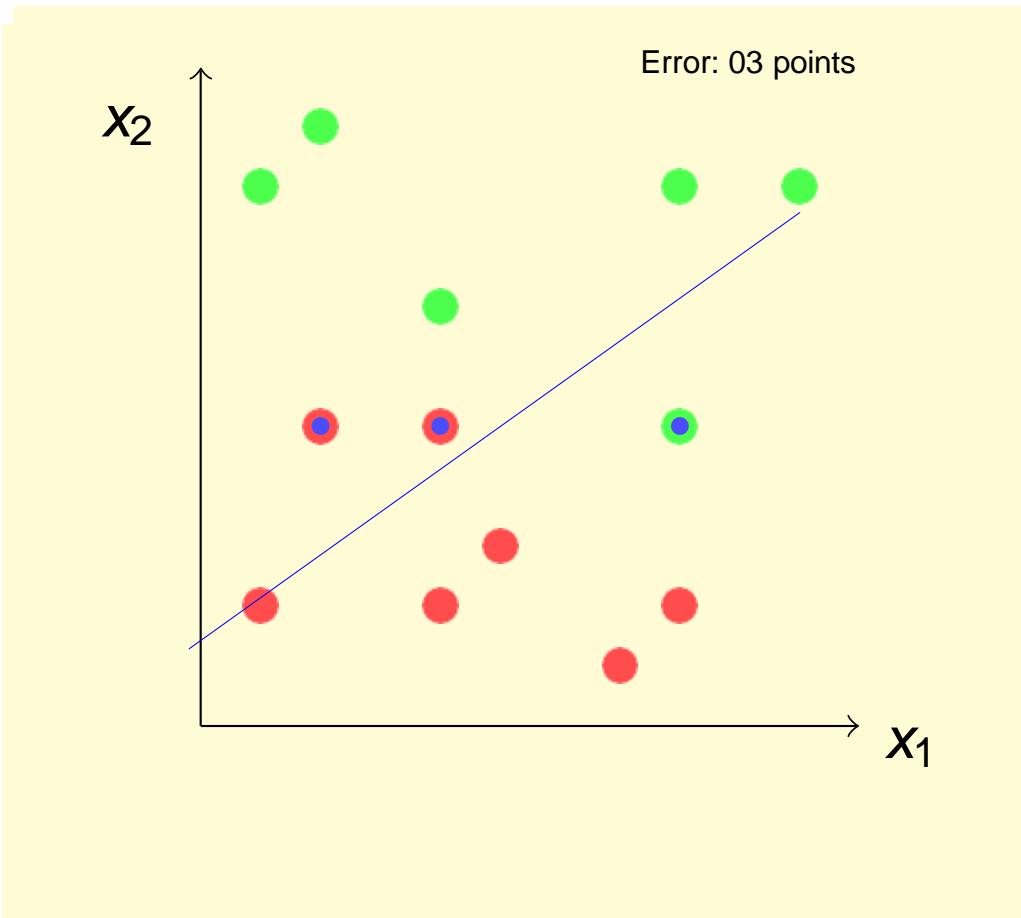
# Visual Interpretation



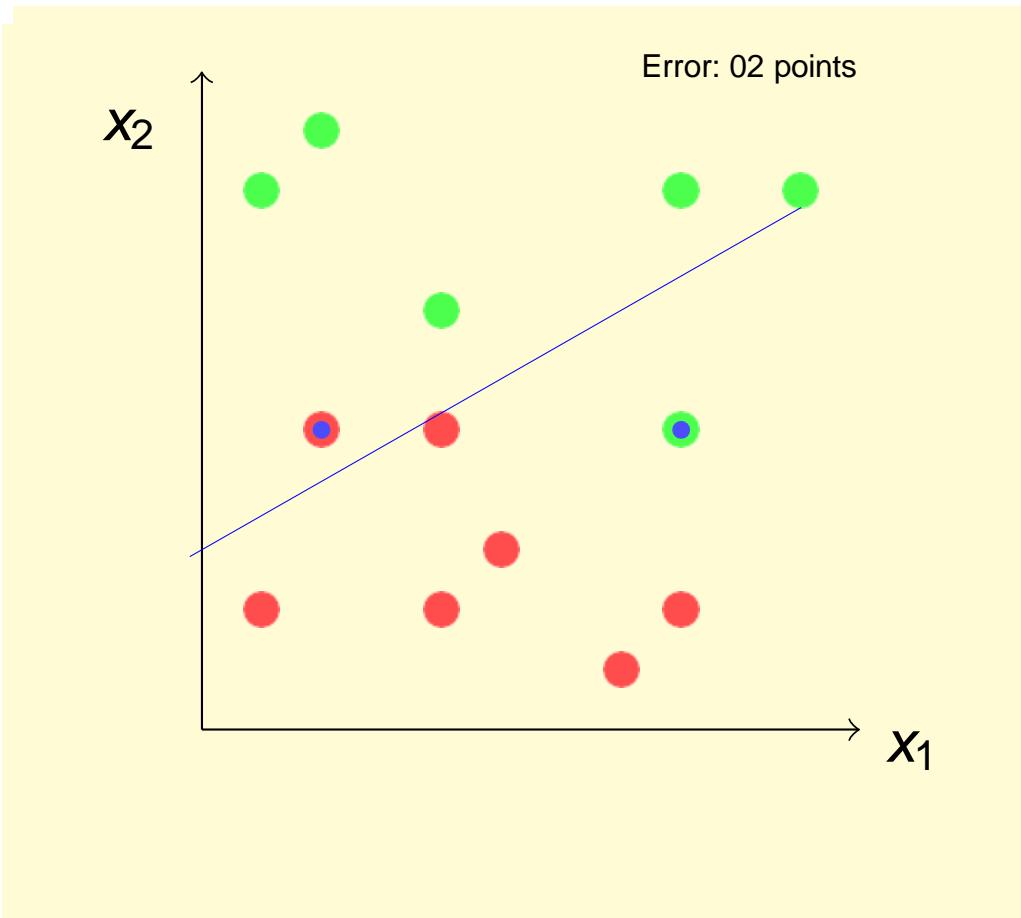
# Visual Interpretation



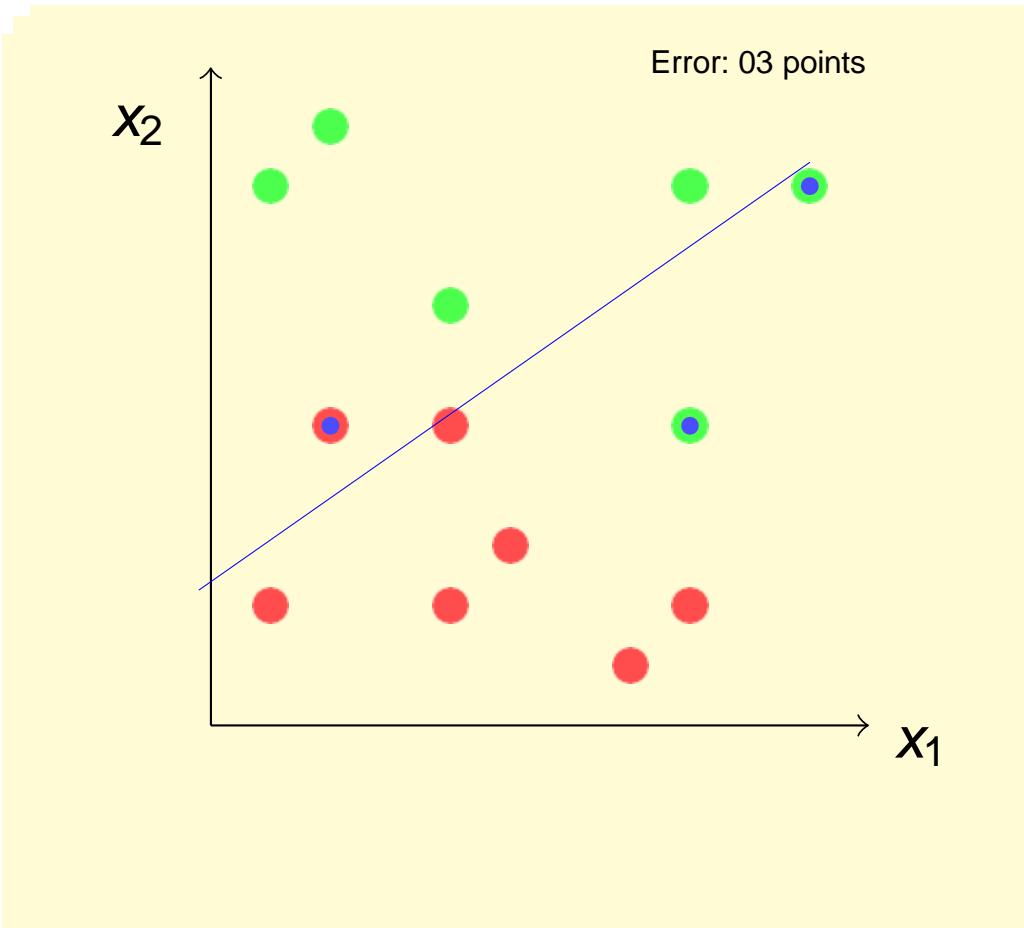
# Visual Interpretation



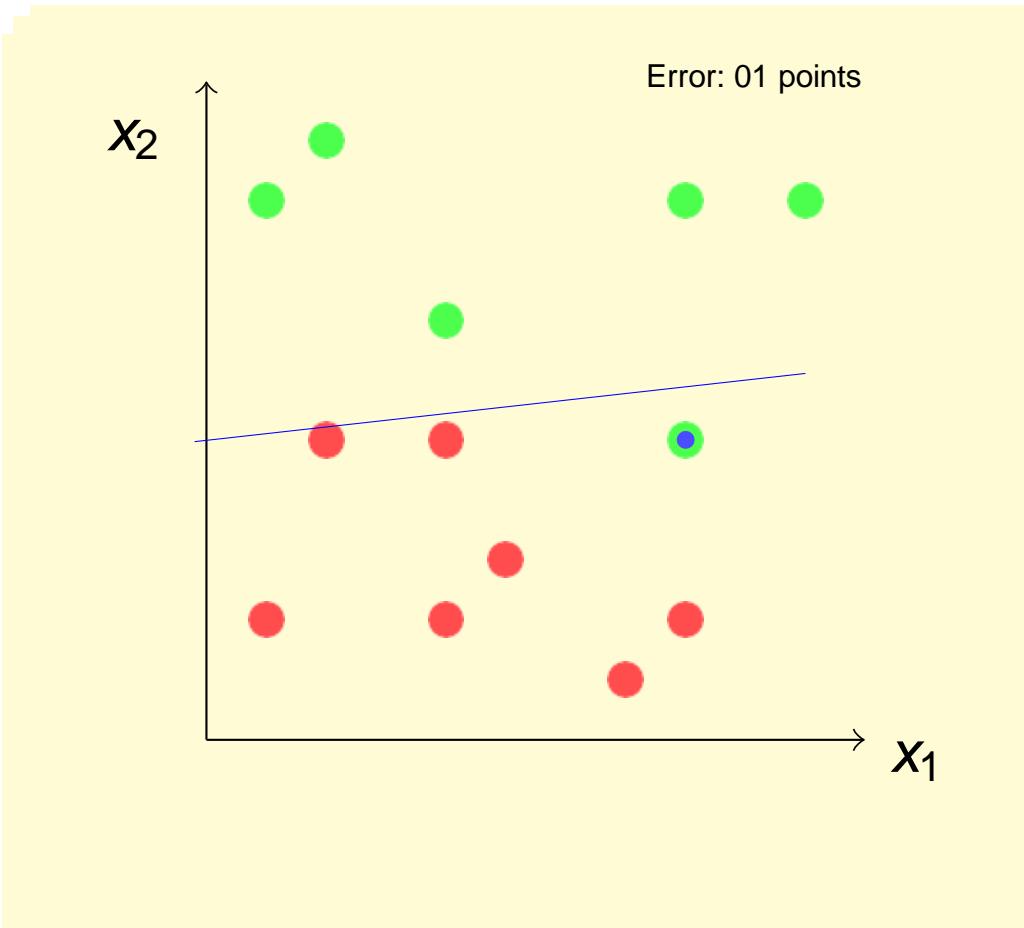
# Visual Interpretation



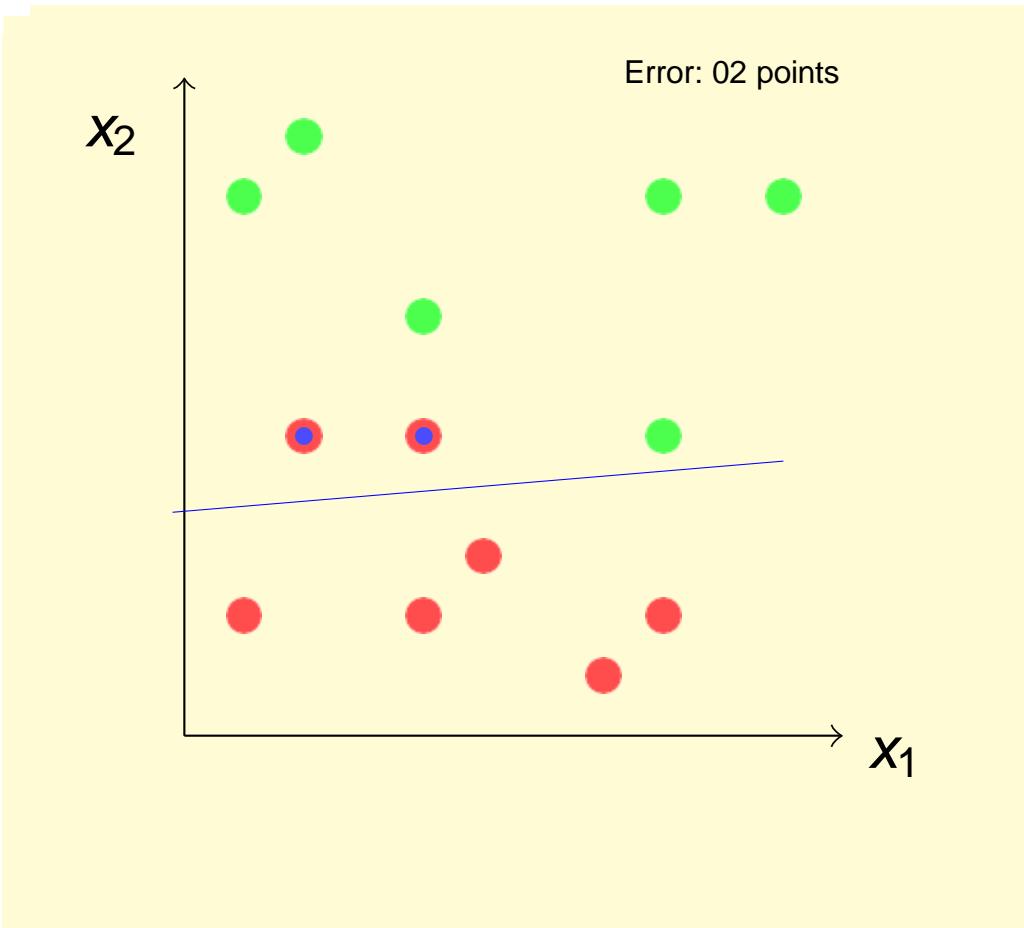
# Visual Interpretation



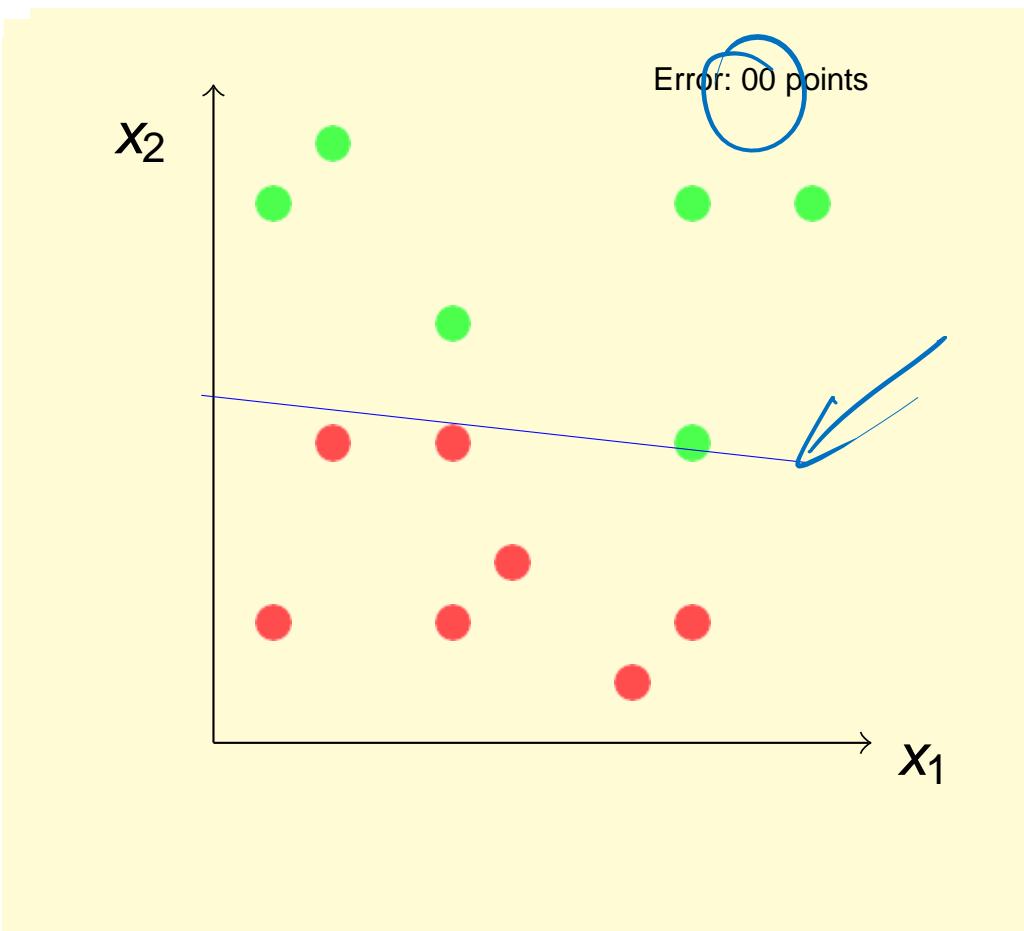
# Visual Interpretation



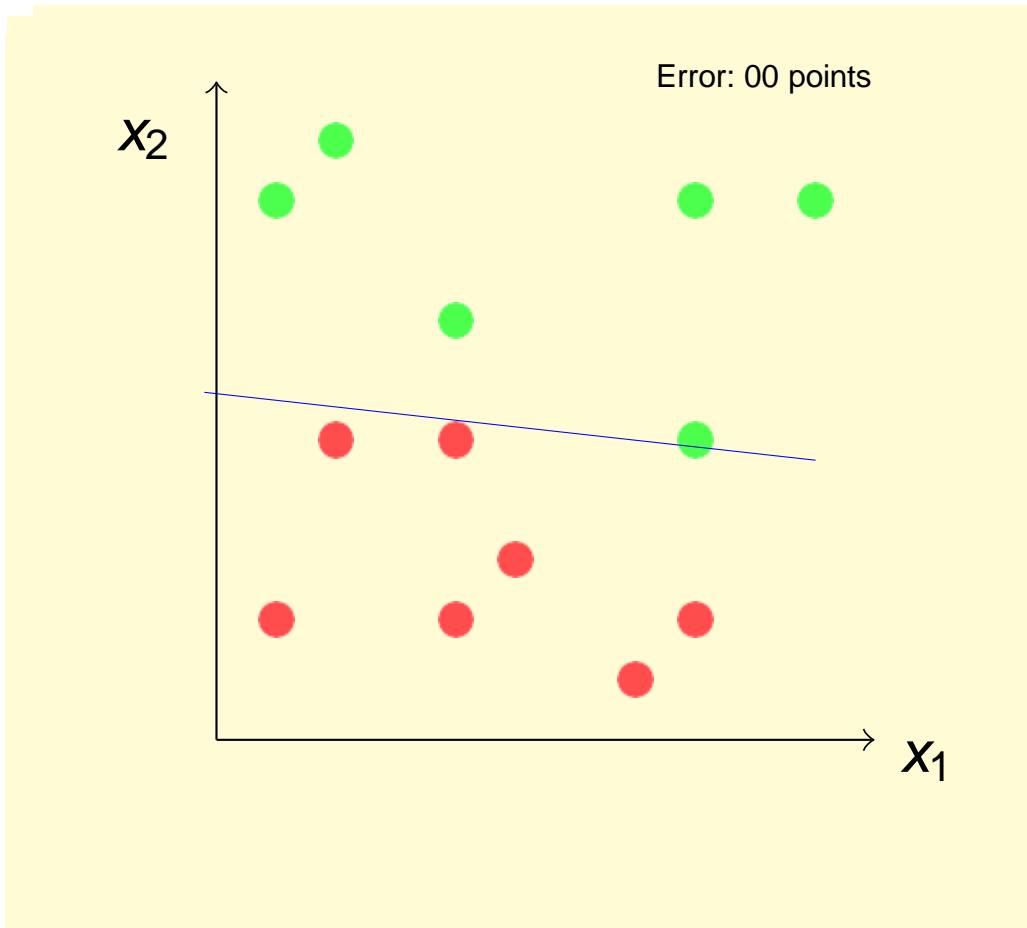
# Visual Interpretation



# Visual Interpretation



# Visual Interpretation



- Conversion is not gradual. (Error is NOT reducing monotonically)
- It is difficult to decide when to stop if data is not linearly separable



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

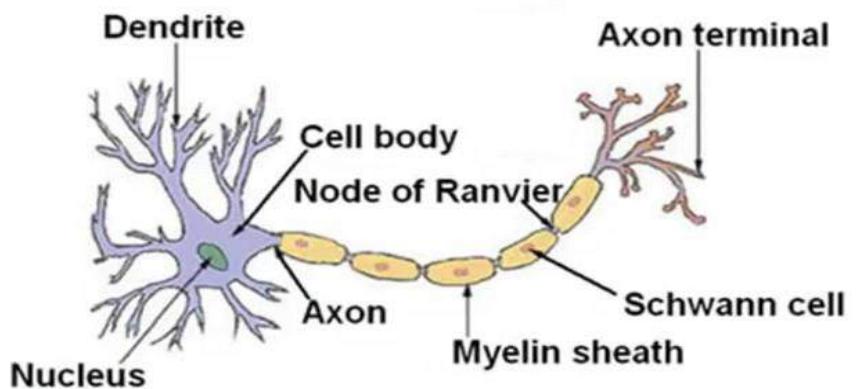
# Introduction to Perceptron

**Kamlesh Tiwari**

# Neural Network (NN)

NN is biologically motivated learning model that mimic human brain

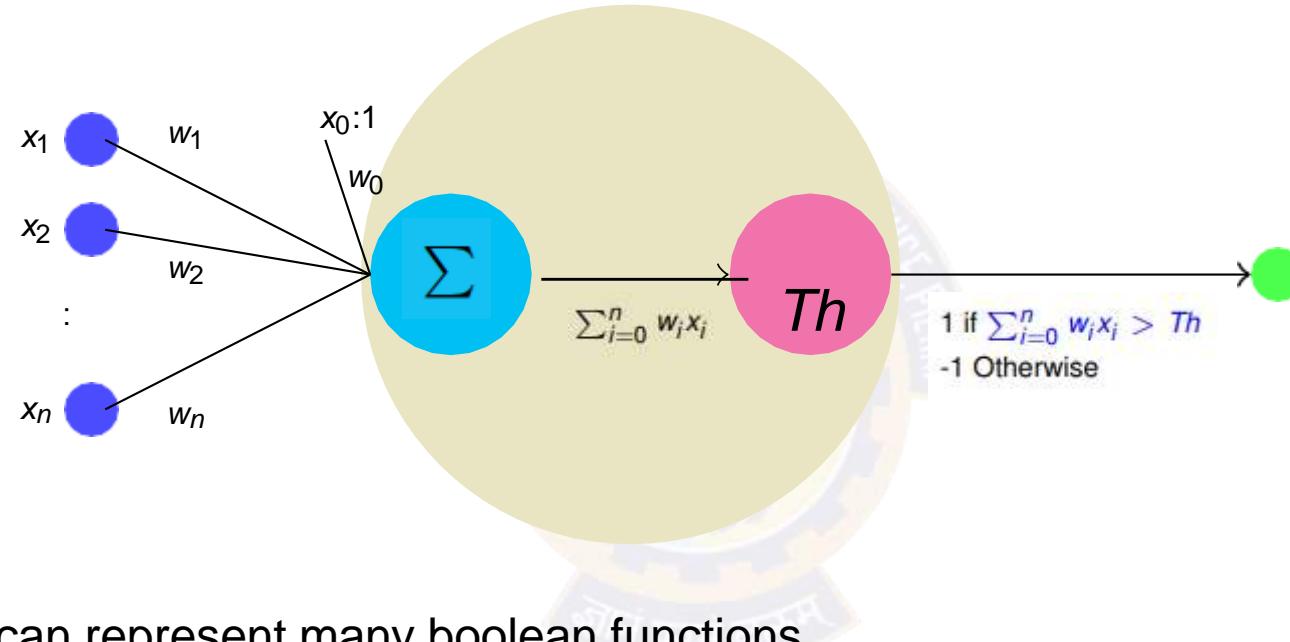
- Started by *W. McCulloch* study on working of neurons in 1943
- MADALINE (1959), an adaptive filter that eliminates echoes on phone lines was the first neural network
- Popularity of Neural Network diminished in 90's but, due to advances in **processing power** and availability of **large data** it again became state-of-the-art



- Cell, Axon, Synapses, Molecules, and Dendrites
- Humans have  $10^{11}$  neurons, each connected to  $10^4$  others, switches in  $10^{-3}$  sec

# A Single Perceptron

## Perceptron representation

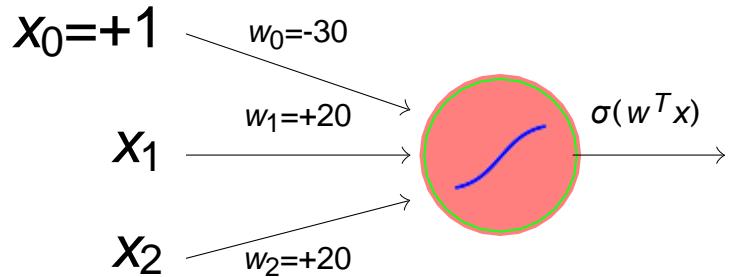


- A single perceptron can represent many boolean functions
- Any  $m$ -of- $n$  function (at least  $m$  of the  $n$  inputs must be true) can be represented by perceptron. OR ( $m=1$ ) and AND ( $m=n$ )

Two layer NN can represent any boolean function (Consider SOP)

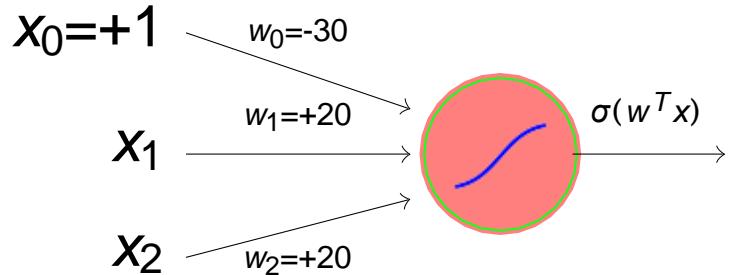
# An Example

Consider a perceptron with output 0/1 as below



# An Example

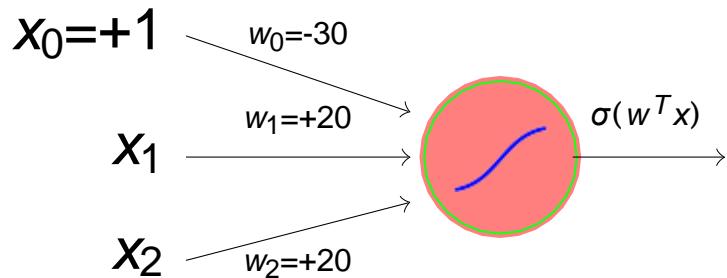
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	

# An Example

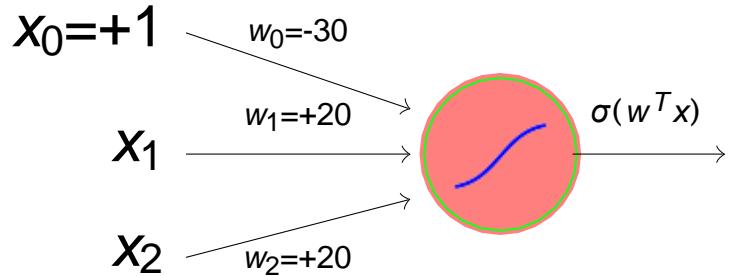
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	

# An Example

Consider a perceptron with output 0/1 as below

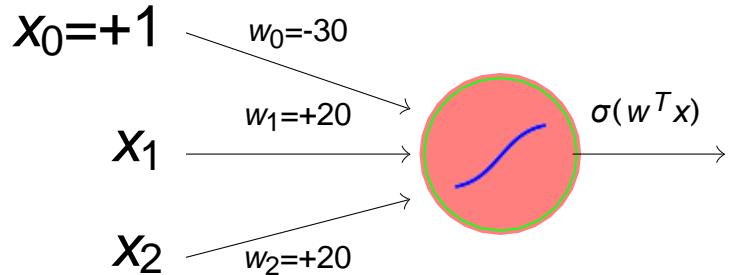


$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	



# An Example

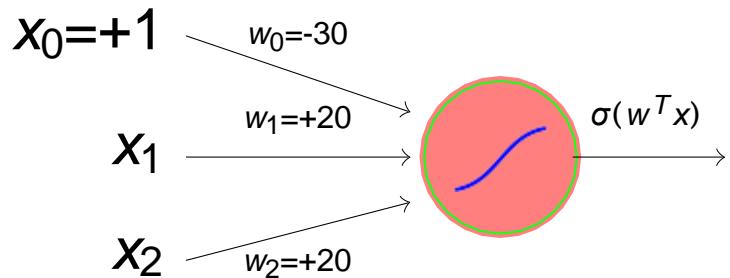
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	0
1	1	

# An Example

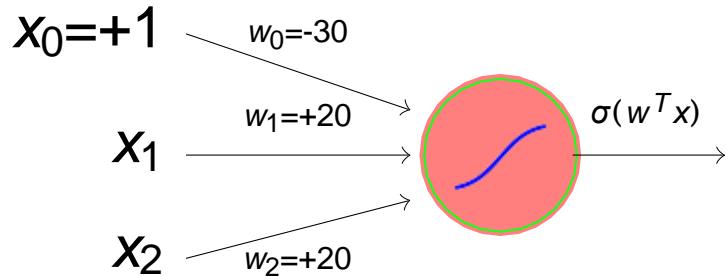
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	0
1	1	1

# An Example

Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	0
1	1	1

This perceptron computes logical AND

- $w_0 = -10$  gives logical OR
- $w_0 = 10$ ,  $w_1 = -20$  with single input gives logical NOT
- XOR is not possible

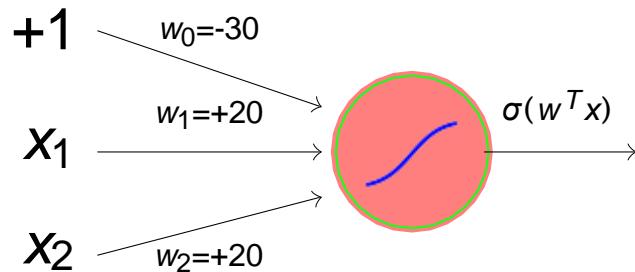


**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# Perceptron in Layer and Network

Kamlesh Tiwari

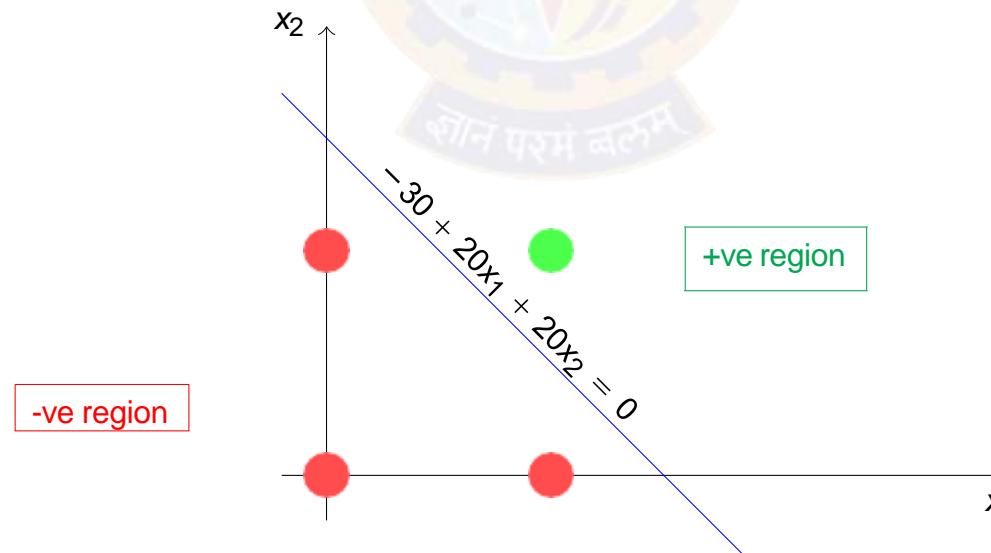
# Essentially it Represents A Decision Boundary



Provides **positive** classification if

$$-30 + 20x_1 + 20x_2 \geq 0$$

Represents a linear decision boundary

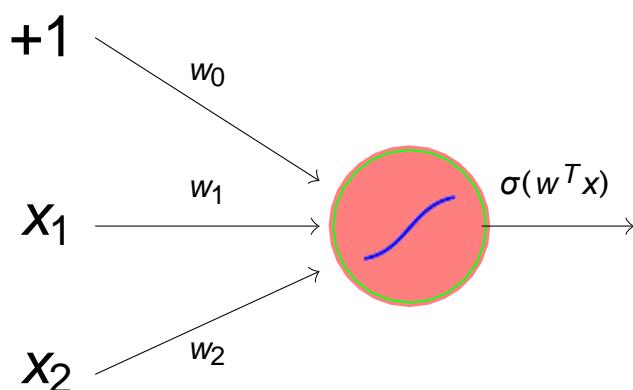


# An Example

Design a perceptron for

$x_1$	$x_2$	Classification
0	0	0
0	1	0
1	0	1
1	1	0

Let us assume following



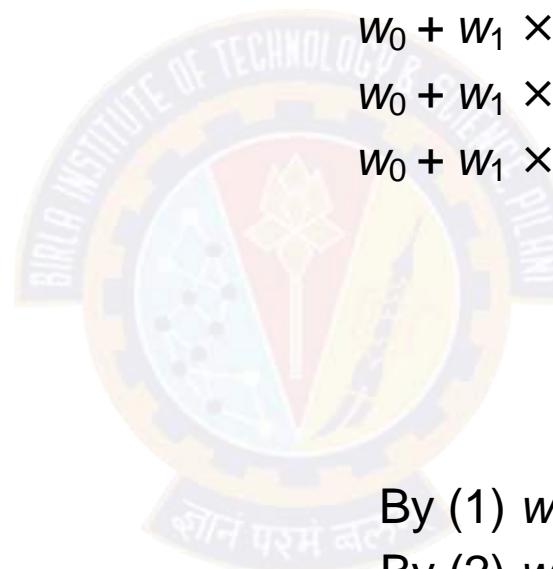
We have following four equations

$$w_0 + w_1 \times (0) + w_2 \times (0) < 0 \quad (1)$$

$$w_0 + w_1 \times (0) + w_2 \times (1) < 0 \quad (2)$$

$$w_0 + w_1 \times (1) + w_2 \times (0) \geq 0 \quad (3)$$

$$w_0 + w_1 \times (1) + w_2 \times (1) < 0 \quad (4)$$



By (1)  $w_0 < 0$  so let  $w_0 = -1$

By (2)  $w_0 + w_2 < 0$  so let  $w_2 = -1$

By (3)  $w_0 + w_1 \geq 0$  so let  $w_1 = 1.5$

By (4)  $w_0 + w_1 + w_2 < 0$  that is valid

So  $(w_0, w_1, w_2) = (-1, -1, 1.5)$

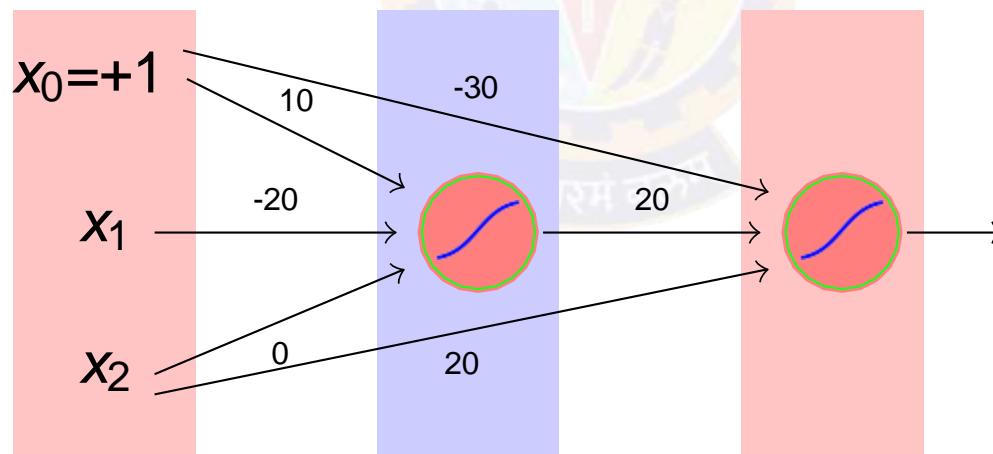
Other possibilities are also there

# An Example

Design a neural network for

$x_1$	$X_2$	Classification
0	0	0
0	1	0
1	0	1
1	1	0

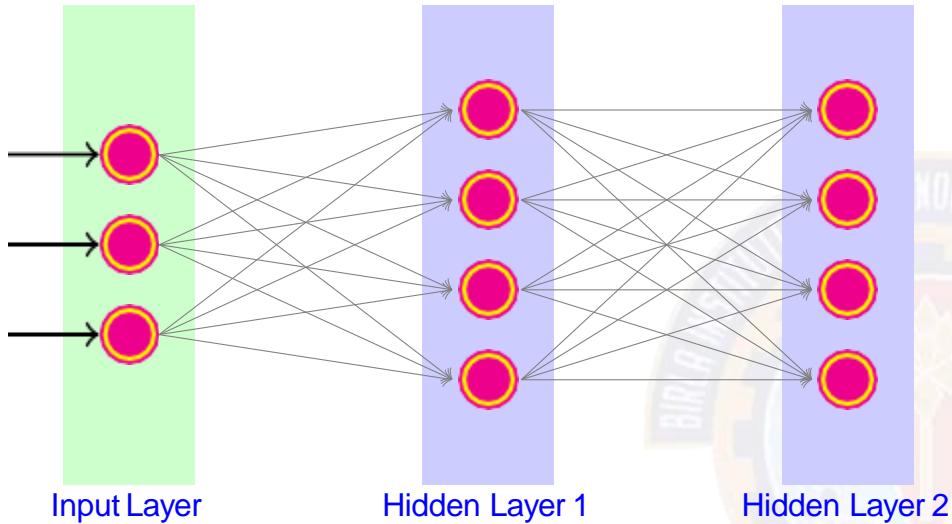
$x_1$	$X_2$	$\bar{x}_2$	$(x_1) \text{AND} (\bar{x}_2)$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0



This arrangement is mostly avoided, as training is more challenging

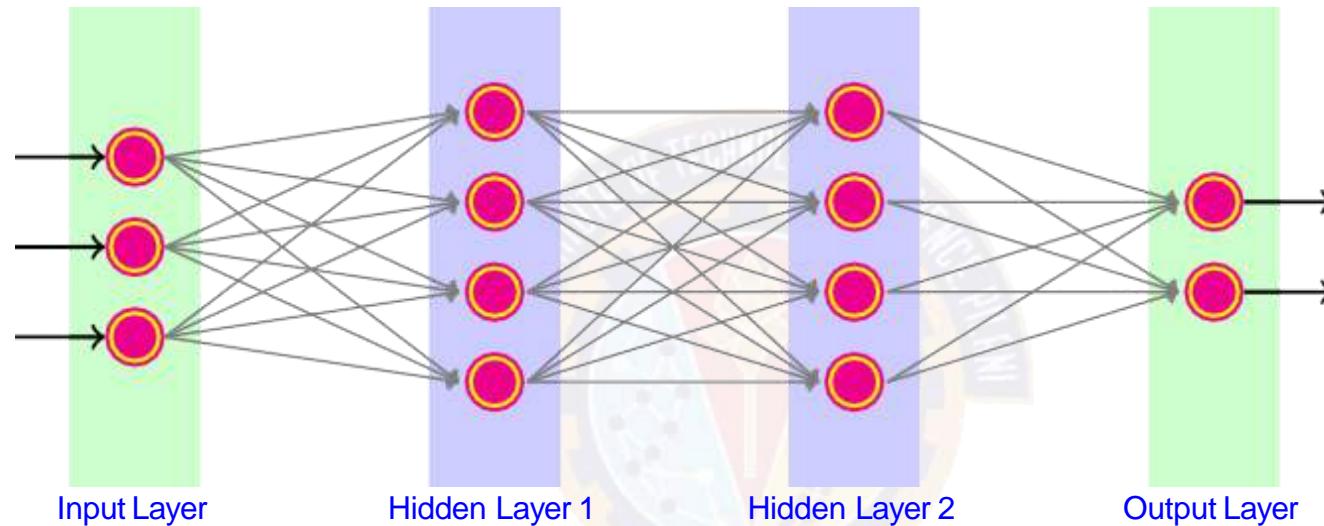
# Neural Network

When neurons are interconnected in layers



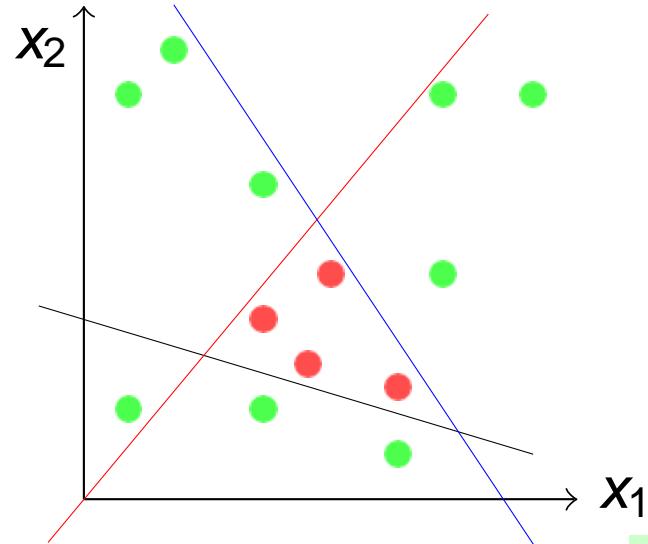
# Neural Network

When neurons are interconnected in layers



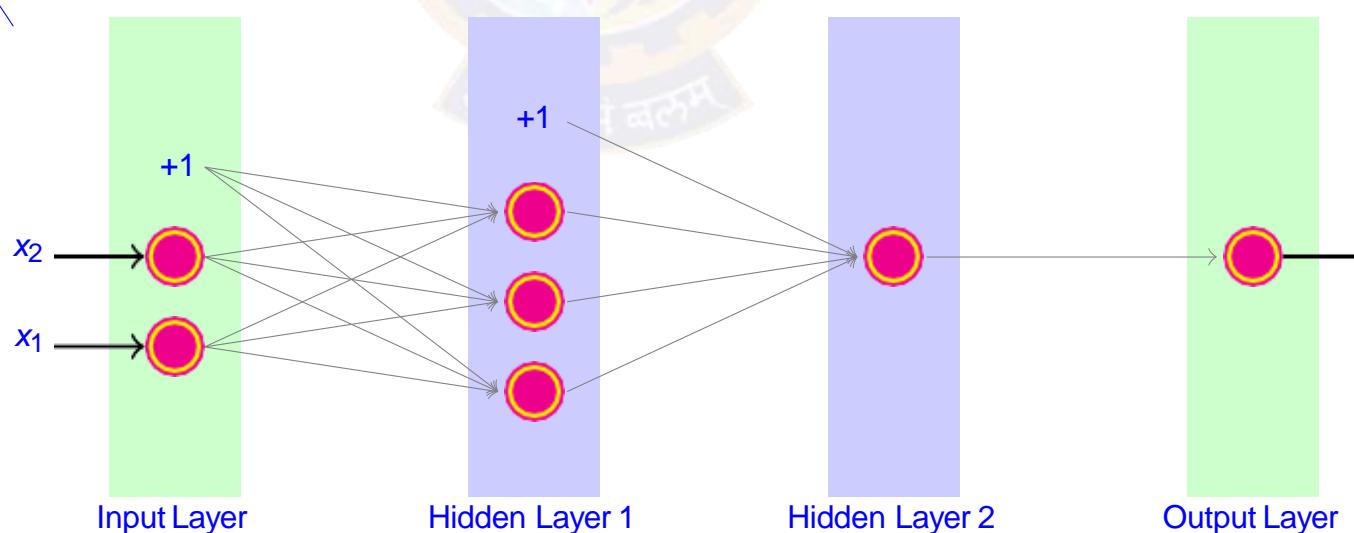
- Number of layers may differ
- Nodes in each intermediate layers may also differ
- Multiple output neurons are used for different class
- **Two levels deep** NN can represent any boolean function

# More Example: Design NN for the following data



Whether it is green?

Red-line	Blue-line	Black-line	Color
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



# Neural Network Applications

- NN is appropriate for problems with the following characteristics:
  - Instances are provided by many attribute-value pairs (more data)
  - The target function output may be discrete-valued, real-valued, or a vector of several real or discrete valued attributes
  - The training examples may contain errors Long training times are acceptable
  - Fast evaluation of the target function may be required
  - The ability of humans to understand the learned target function is not important



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Perceptron Training (Delta Rule)

Kamlesh Tiwari

# Perceptron Training (delta rule)

- Delta rule converges to a best-fit approximation of the target
- Uses gradient descent
- Consider unthresholded perceptron,  $o(\vec{x}) = \vec{w} \cdot \vec{x}$
- Training error is defined as

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- Gradient would specify direction of steepest increase

$$\nabla E(\vec{w}) = \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- Weights can be learned as  $w_i = w_i - \eta \frac{\partial E}{\partial w_i}$

- It can be seen that  $\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$

# Perceptron Training (delta rule)

---

## Algorithm 9: Gradient Descent ( $D, \eta$ )

---

```
1 Initialize  $w_i$  with random weights
2 repeat
3   For each  $w_i$ , initialize  $\delta w_i = 0$ 
4   for each training example  $d \in D$  do
5     Compute output  $o$  using model for  $d$  whose target is  $t$ 
6     For each  $w_i$ , update  $\delta w_i = \delta w_i + \eta(t-o)x_i$ 
7   For each  $w_i$ , set  $w_i = w_i + \delta w_i$ 
8 until termination condition is met,
9 return  $w$ 
```

---

# Perceptron Training (delta rule)

---

## Algorithm 10: Gradient Descent ( $D, \eta$ )

---

```
1 Initialize  $w_i$  with random weights
2 repeat
3   For each  $w_i$ , initialize  $\delta w_i = 0$ 
4   for each training example  $d \in D$  do
5     Compute output  $o$  using model for  $d$  whose target is  $t$ 
6     For each  $w_i$ , update  $\delta w_i = \delta w_i + \eta(t-o)x_i$ 
7   For each  $w_i$ , set  $w_i = w_i + \delta w_i$ 
8 until termination condition is met,
9 return  $w$ 
```

---

- A date item  $d \in D$ , is supposed to be multidimensional  $d = (x_1, x_2, \dots, x_n, t)$
- Algorithm converges toward the minimum error hypothesis.

# Perceptron Training (delta rule)

---

## Algorithm 11: Gradient Descent ( $D, \eta$ )

---

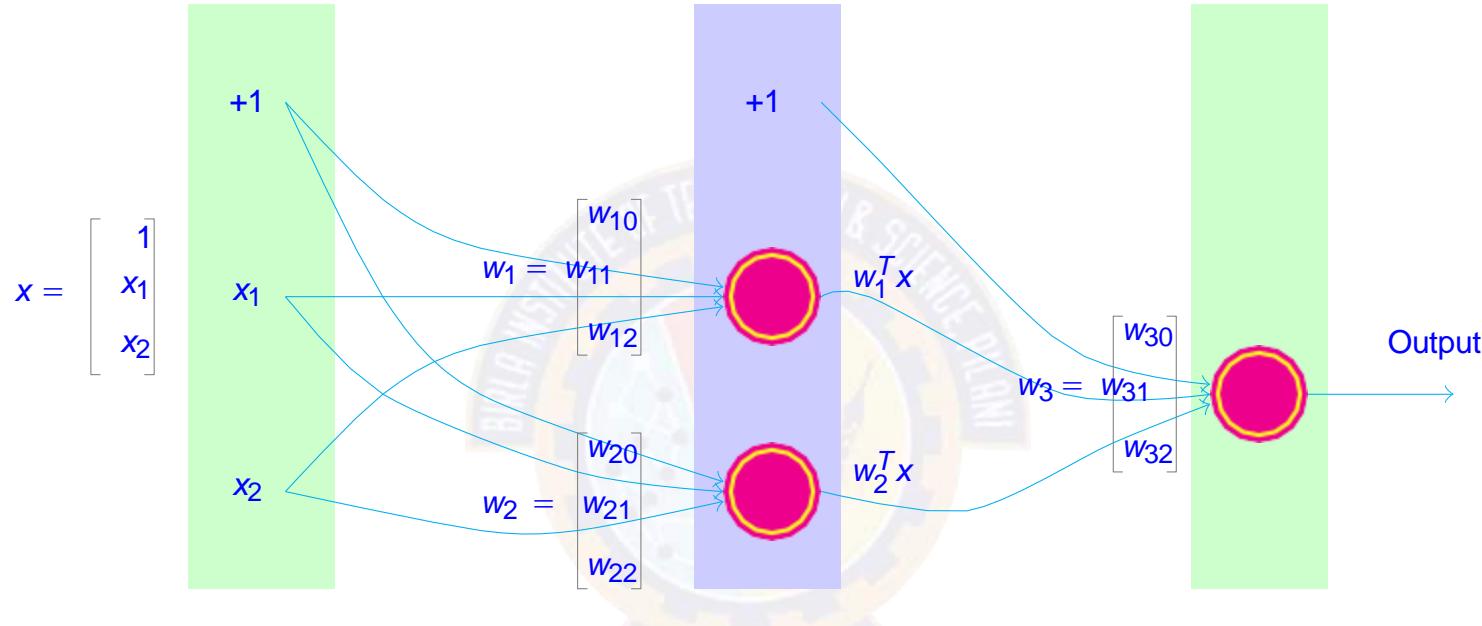
```
1 Initialize  $w_i$  with random weights
2 repeat
3   For each  $w_i$ , initialize  $\delta w_i = 0$ 
4   for each training example  $d \in D$  do
5     Compute output  $o$  using model for  $d$  whose target is  $t$ 
6     For each  $w_i$ , update  $\delta w_i = \delta w_i + \eta(t-o)x_i$ 
7   For each  $w_i$ , set  $w_i = w_i + \delta w_i$ 
8 until termination condition is met,
9 return  $w$ 
```

---

- A date item  $d \in D$ , is supposed to be multidimensional  $d = (x_1, x_2, \dots, x_n, t)$
- Algorithm converges toward the minimum error hypothesis.
- Linear programming can also be an approach

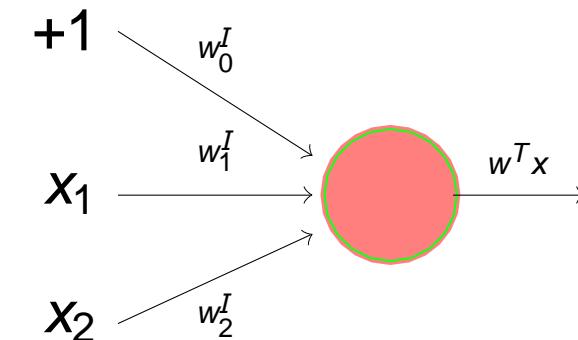
# Linear Activation is Not Much Interesting

NN with perceptrons have limited capability, even with many layers

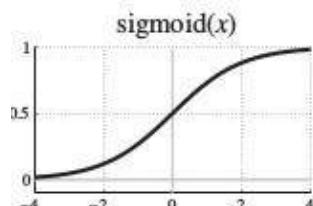
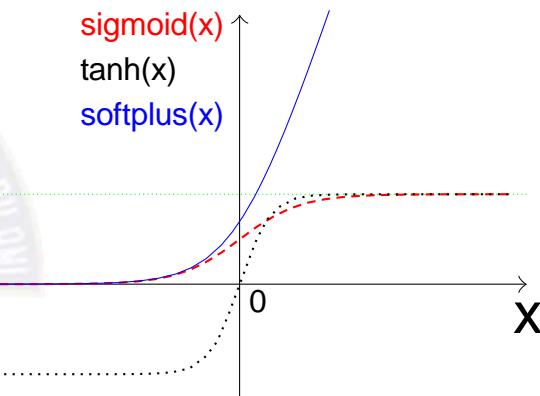
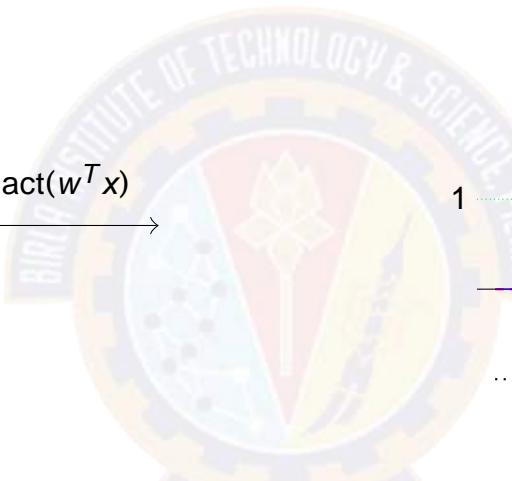
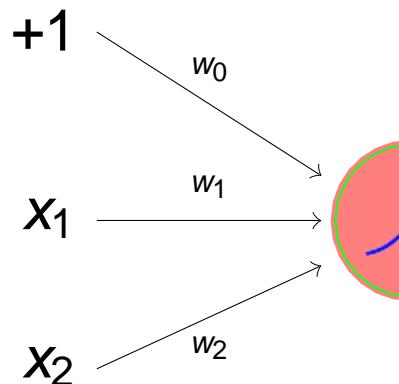


$$\begin{aligned} \text{Output} &= w_{30} \times 1 + w_{31} \times (w_1^T x) + w_{32} \times (w_2^T x) \\ &= w_{30} \times 1 + w_{31} \times [w_{10} \times 1 + w_{11} \times x_1 + w_{12} \times x_2] \\ &\quad + w_{32} \times [w_{20} \times 1 + w_{21} \times x_1 + w_{22} \times x_2] \\ &= (w_{30} + w_{31}w_{10} + w_{32}w_{20}) + (w_{31}w_{11} + w_{32}w_{21}) \times x_1 \\ &\quad + (w_{31}w_{12} + w_{32}w_{22}) \times x_2 \\ &= w'_0 + w'_1 \times x_1 + w'_2 \times x_2 \end{aligned}$$

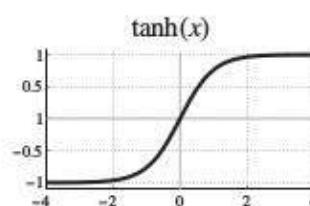
Expression of single perceptron



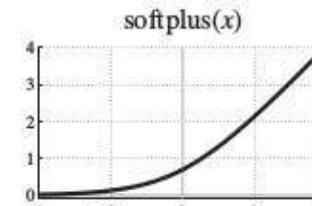
**Neuron uses nonlinear **activation functions** (**sigmoid, tanh, ReLU, softplus etc.**) at the place of thresholding**



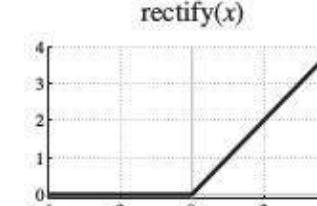
$$h(x) = \frac{1}{1 + \exp(-x)}$$



$$h(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



$$h(x) = \log(1 + \exp(x))$$



$$h(x) = \max(0, x)$$

# Multilayer Networks and Backpropagation

- Single perceptron can only express linear decision surface
- We need units whose output is a nonlinear function of its inputs AND is also differentiable (Use Neuron not Perceptron)

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x})$$

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

- **Backpropagation** algorithm learns the weights for a fixed set of units and interconnections
- It employs **gradient descent** to minimize the error between the network output values and the target values for these outputs
- Let Error function is redefined as

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

# Backpropagation (for 2 layers)

---

## Algorithm 12: Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

---

1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers  
2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$   
3 **repeat**  
4     **for each**  $\langle x, t \rangle \in D$  **do**  
5          $o_u = \text{get output from network } \forall \text{unit } u$   
6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit**  $k$   
7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit**  $h$   
8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$   
9 **until** converge;

---

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$

# Backpropagation (for 2 layers)

---

## Algorithm 13: Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

---

1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers  
2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$   
3 **repeat**  
4     **for each**  $\langle x, t \rangle \in D$  **do**  
5          $o_u = \text{get output from network } \forall \text{unit } u$   
6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**   
7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**   
8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$   
9 **until** converge;

---

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$
- For a single training example  $E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$

# Backpropagation (for 2 layers)

---

## Algorithm 14: Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

---

1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers  
2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$   
3 **repeat**  
4     **for each**  $\langle x, t \rangle \in D$  **do**  
5          $o_u = \text{get output from network } \forall \text{unit } u$   
6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**   
7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**   
8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$   
9 **until** converge;

---

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$
- For a single training example  $E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$
- Weight  $w_{ji}$  is updated by adding  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$



# Thank You!

In our next session:



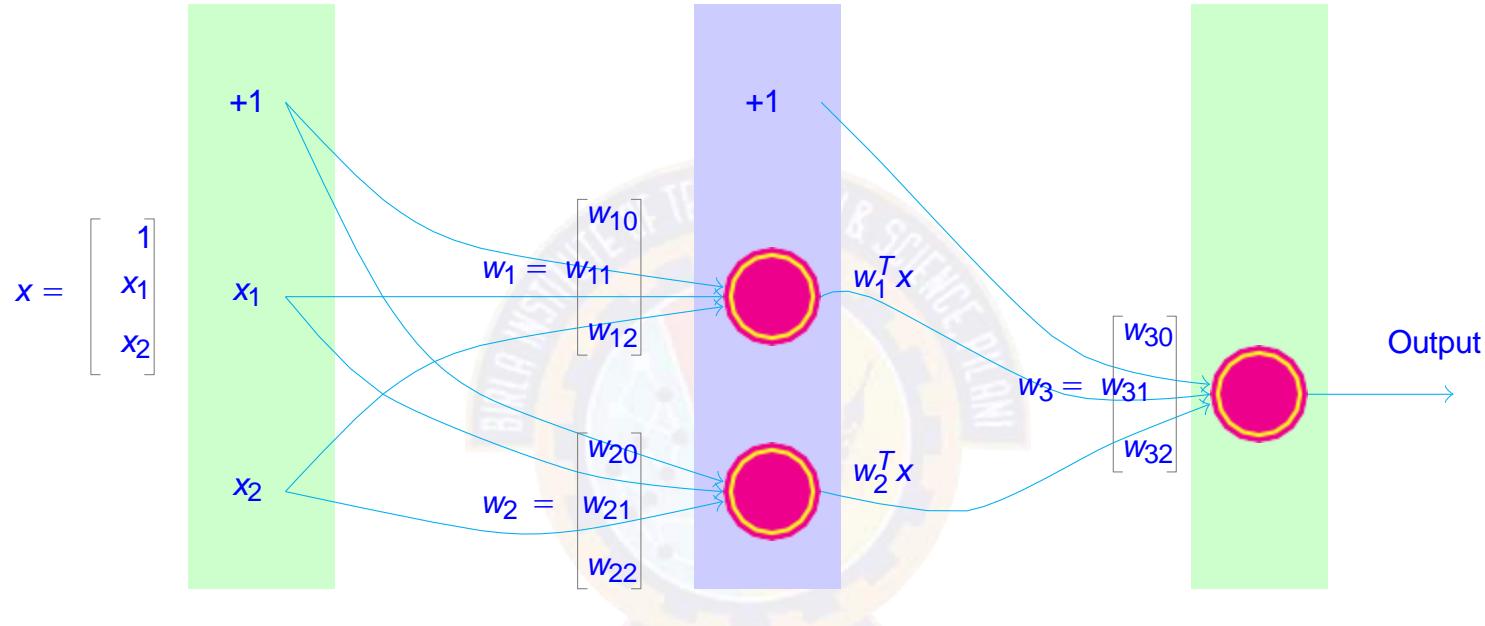
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Importance of Non-Linearity

**Dr. Kamlesh Tiwari**

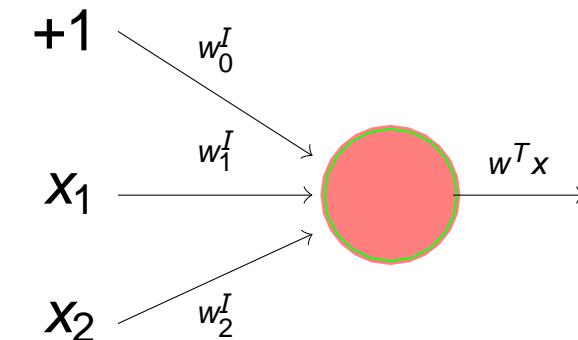
# Linear Activation is Not Much Interesting

NN with perceptrons have limited capability, even with many layers



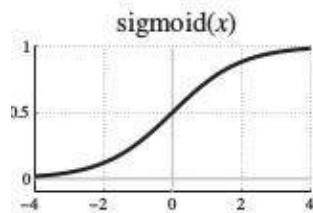
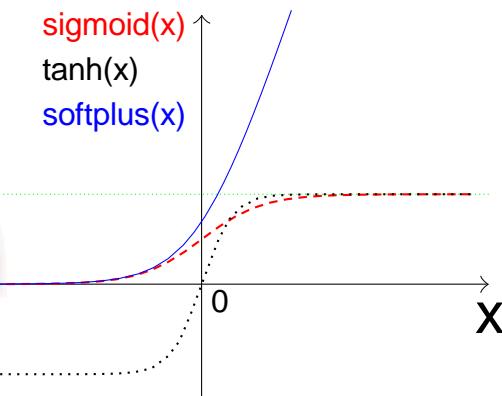
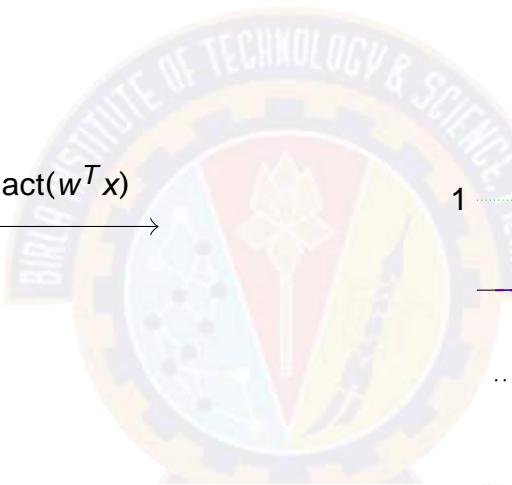
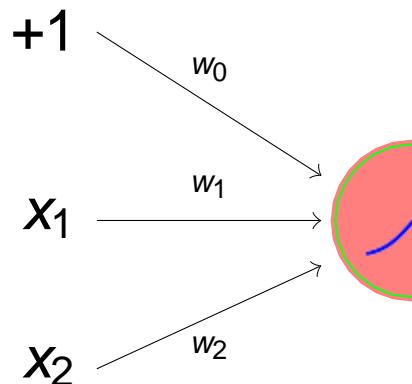
$$\begin{aligned} \text{Output} &= w_{30} \times 1 + w_{31} \times (w_1^T x) + w_{32} \times (w_2^T x) \\ &= w_{30} \times 1 + w_{31} \times [w_{10} \times 1 + w_{11} \times x_1 + w_{12} \times x_2] \\ &\quad + w_{32} \times [w_{20} \times 1 + w_{21} \times x_1 + w_{22} \times x_2] \\ &= (w_{30} + w_{31}w_{10} + w_{32}w_{20}) + (w_{31}w_{11} + w_{32}w_{21}) \times x_1 \\ &\quad + (w_{31}w_{12} + w_{32}w_{22}) \times x_2 \\ &= w'_0 + w'_1 \times x_1 + w'_2 \times x_2 \end{aligned}$$

Expression of single perceptron

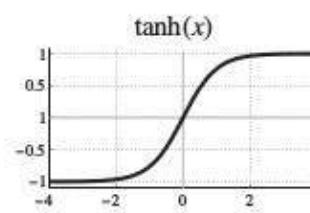


# Neuron

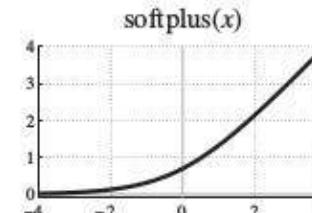
Neuron uses nonlinear **activation functions** (**sigmoid, tanh, ReLU, softplus etc.**) at the place of thresholding



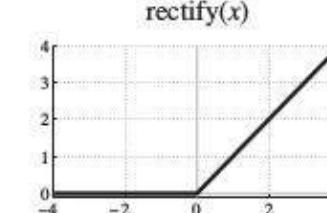
$$h(x) = \frac{1}{1 + \exp(-x)}$$



$$h(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



$$h(x) = \log(1 + \exp(x))$$



$$h(x) = \max(0, x)$$



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Backpropogation

**Kamlesh Tiwari**

# Multilayer Networks and Backpropagation

- Single perceptron can only express linear decision surface
- We need units whose output is a nonlinear function of its inputs AND is also differentiable (Use Neuron not Perceptron)

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x})$$

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



- **Backpropagation** algorithm learns the weights for a fixed set of units and interconnections
- It employs **gradient descent** to minimize the error between the network output values and the target values for these outputs
- Let Error function is redefined as

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

# Backpropagation (for 2 layers)

## Algorithm 12: Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

```
1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers
2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$ 
3 repeat
4   for each  $\langle \vec{x}, \vec{t} \rangle \in D$  do
5      $o_u = \text{get output from network } \forall \text{unit } u$ 
6      $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all output unit  $k$ 
7      $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all hidden unit  $h$ 
8      $w_{ji} = w_{ji} + \Delta w_{ji}$  where  $\Delta w_{ji} = \eta \delta_j x_{ji}$ 
9 until converge;
```

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$

# Backpropagation (for 2 layers)

## Algorithm 13: Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

```
1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers  
2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$   
3 repeat  
4   for each  $\langle \vec{x}, \vec{t} \rangle \in D$  do  
5      $o_u =$  get output from network  $\forall$  unit  $u$   
6      $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all output unit  $k$   
7      $\delta_h = o_h(1 - o_h) \sum_{k \in outputs} (w_{kh} \delta_k)$  for all hidden unit  $h$   
8      $w_{ji} = w_{ji} + \Delta w_{ji}$  where  $\Delta w_{ji} = \eta \delta_j x_{ji}$   
9 until converge;
```

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$

- For a single training example  $E_d(\vec{w}) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$

$$E(\vec{w}) = \sum_{d \in D} E_d(\vec{w})$$

# Backpropagation (for 2 layers)

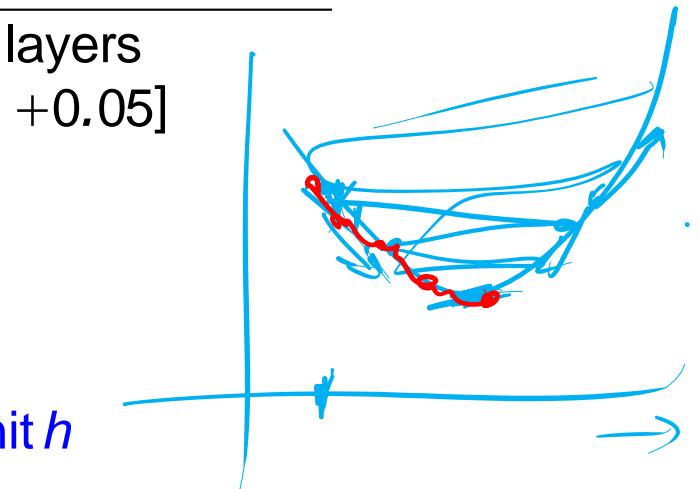
## Algorithm 14: Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

```
1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers  
2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$   
3 repeat  
4   for each  $\langle \vec{x}, \vec{t} \rangle \in D$  do  
5      $o_u =$  get output from network  $\forall$  unit  $u$   
6      $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all output unit  $k$   
7      $\delta_h = o_h(1 - o_h) \sum_{k \in outputs} (w_{kh} \delta_k)$  for all hidden unit  $h$   
8      $w_{ji} = w_{ji} + \Delta w_{ji}$  where  $\Delta w_{ji} = \eta \delta_j x_{ji}$   
9 until converge;
```

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$

- For a single training example,  $E_d(\vec{w}) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$

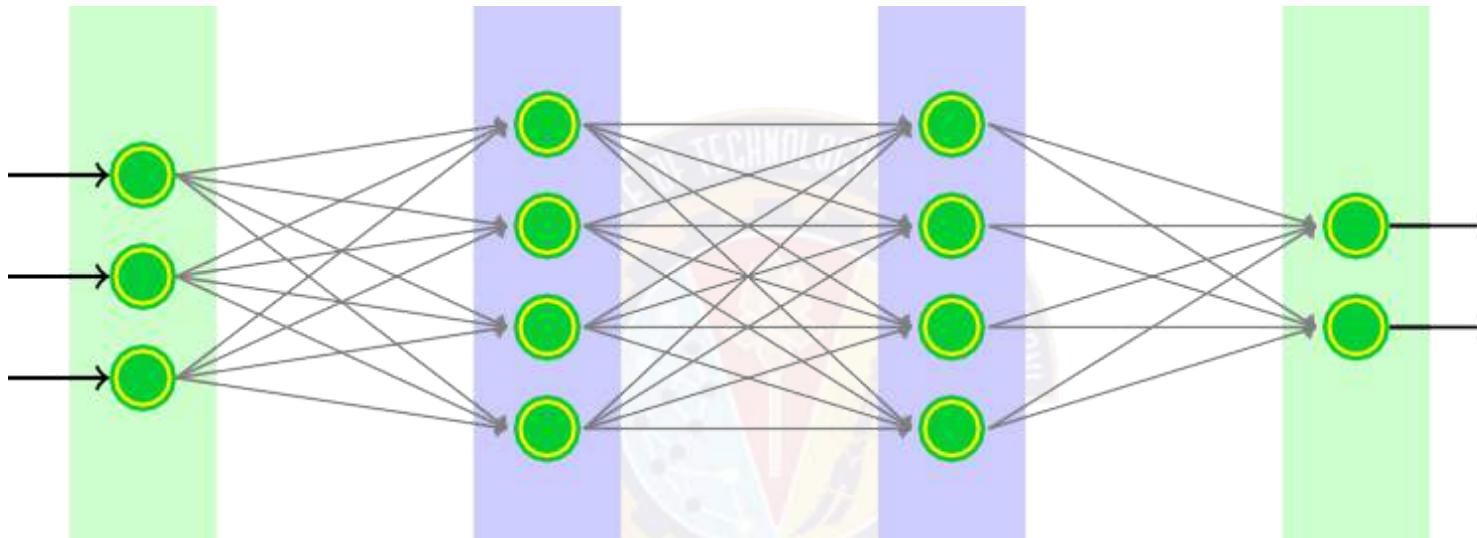
- Weight  $w_{ji}$  is updated by adding



$$\frac{\partial E_d}{\partial w_{ji}}$$

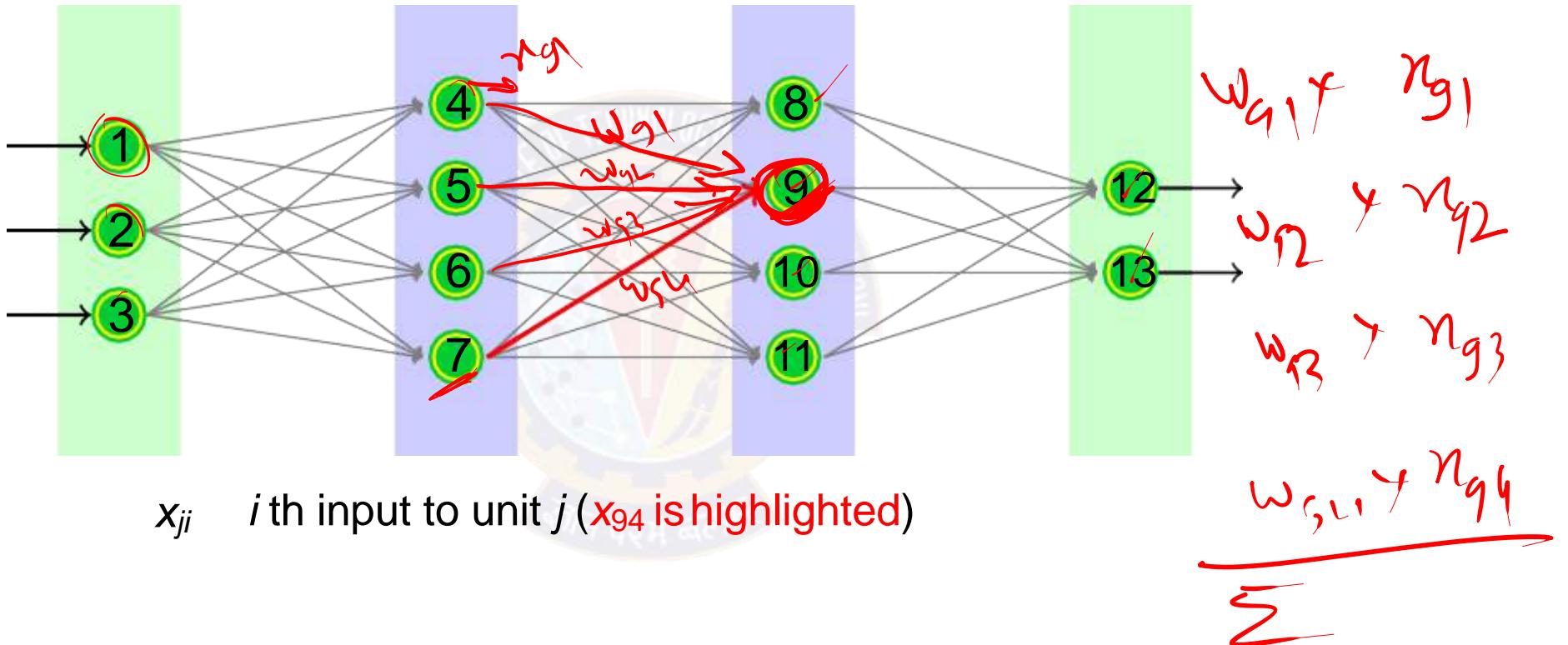
A hand-drawn diagram showing a red box containing the partial derivative of the error function with respect to weight  $w_{ji}$ . A blue arrow points towards this box, indicating the direction of weight update.

# Conventions Over The Network

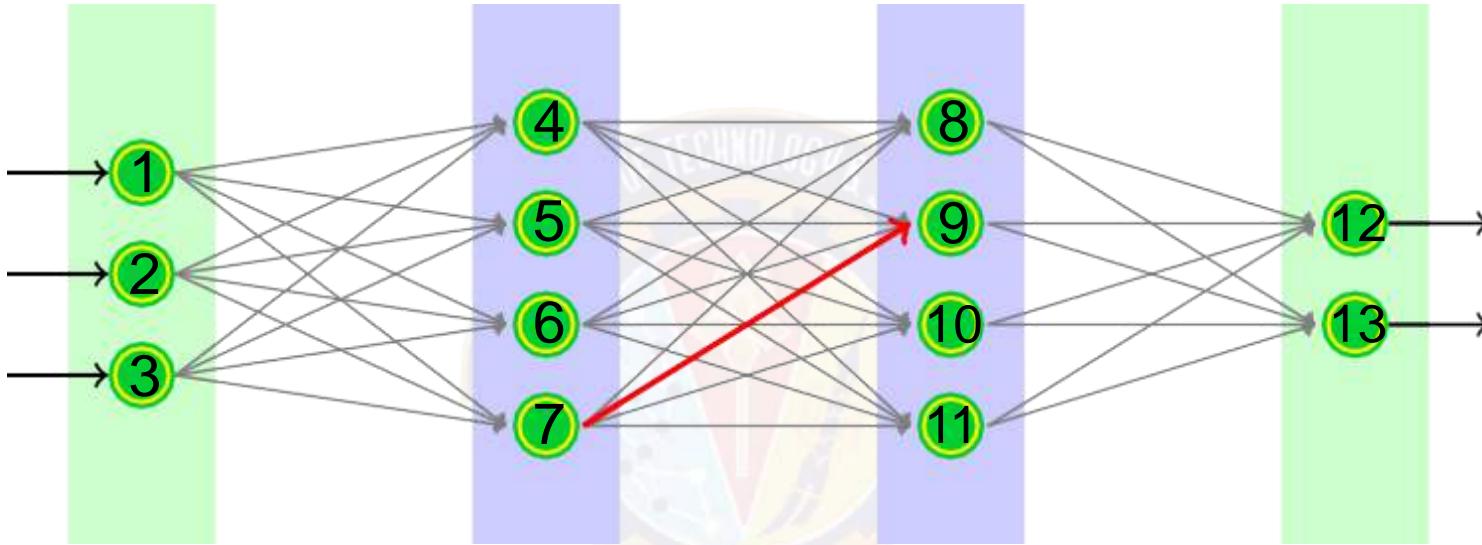


$x_{ji}$     $i$  th input to unit  $j$  ( $x_{94}$  is highlighted)

# Conventions Over The Network



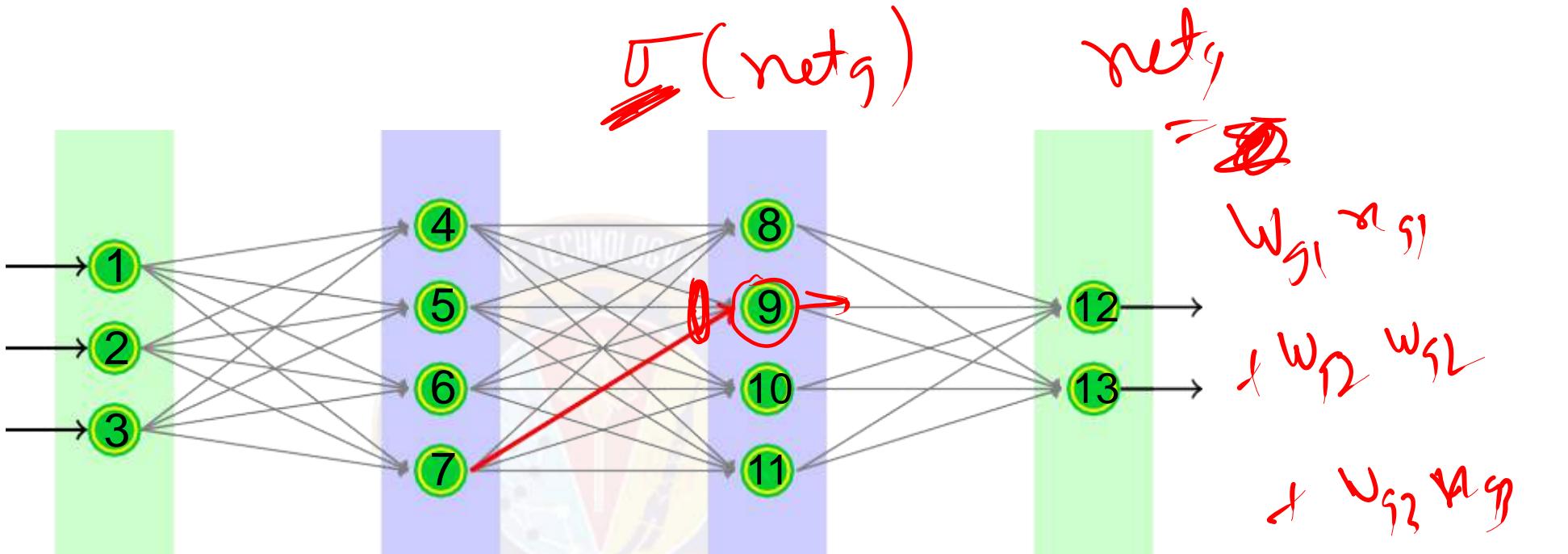
# Conventions Over The Network



$x_{ji}$     $i$  th input to unit  $j$  ( $x_{94}$  is highlighted)

$w_{ji}$    weight associated with  $i$  th input to unit  $j$

# Conventions Over The Network

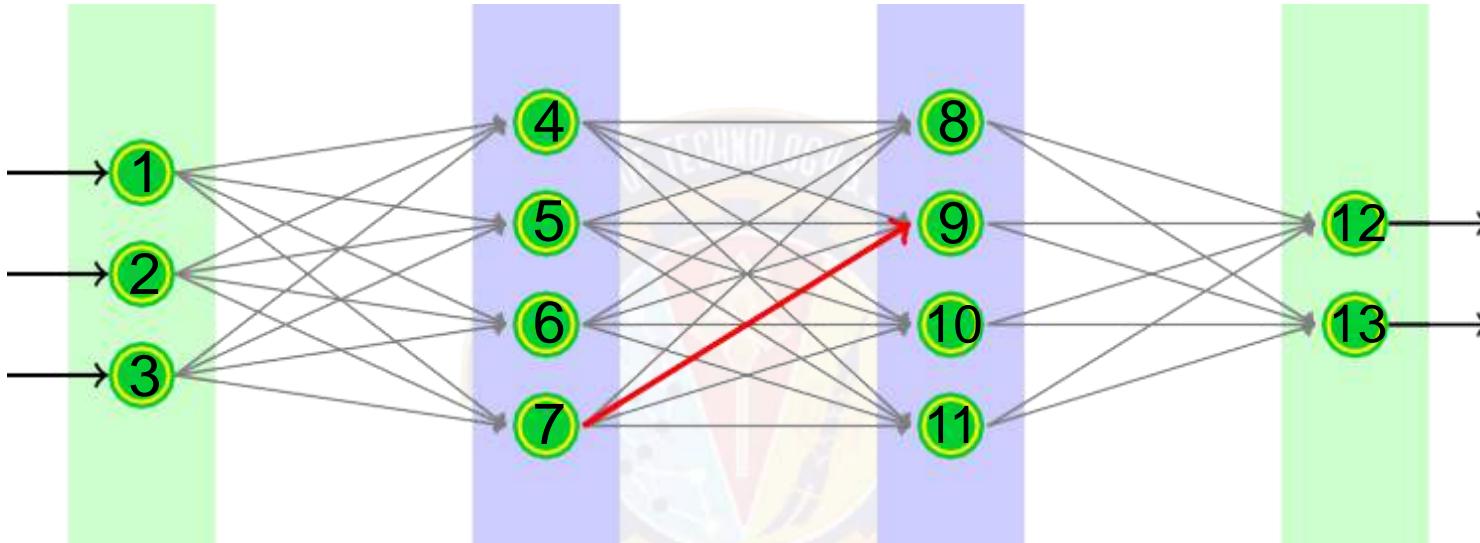


$x_{ji}$   $i$  th input to unit  $j$  ( $x_{94}$  is highlighted)

$w_{ji}$  weight associated with  $i$  th input to unit  $j$

$\text{net}_j$  be  $\sum_i w_{ji} x_{ji}$  the weighted sum of input for unit  $j$

# Conventions Over The Network



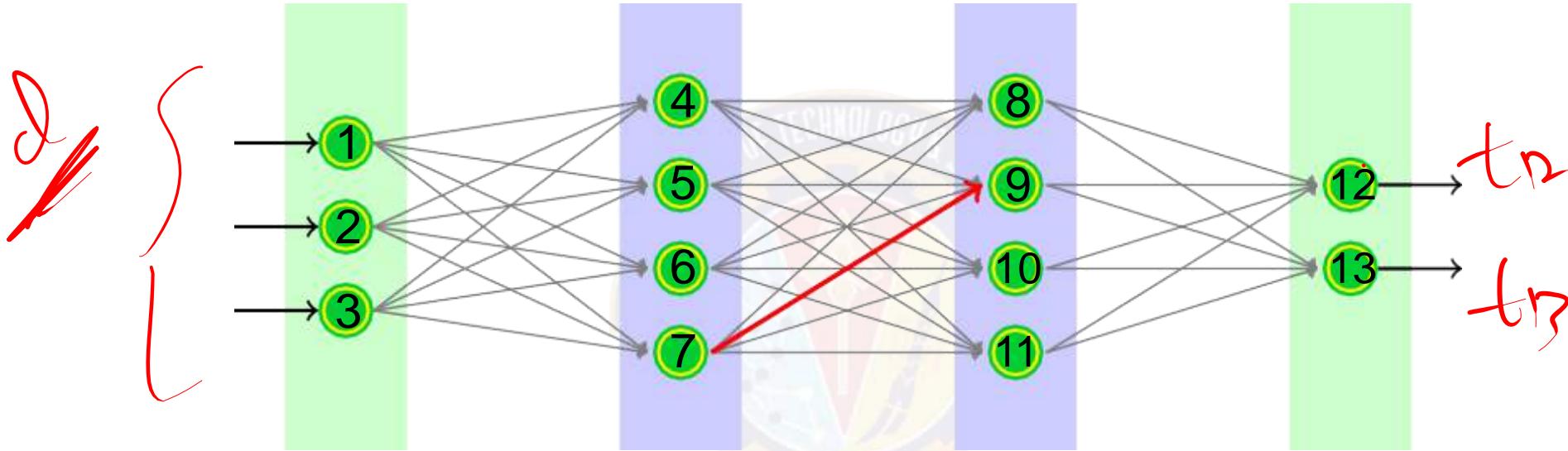
$x_{ji}$   $i$  th input to unit  $j$  ( $x_{94}$  is highlighted)

$w_{ji}$  weight associated with  $i$  th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$   
 $o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

$\sigma(net_j)$

# Conventions Over The Network



$x_{ji}$   $i$  th input to unit  $j$  ( $x_{94}$  is highlighted)

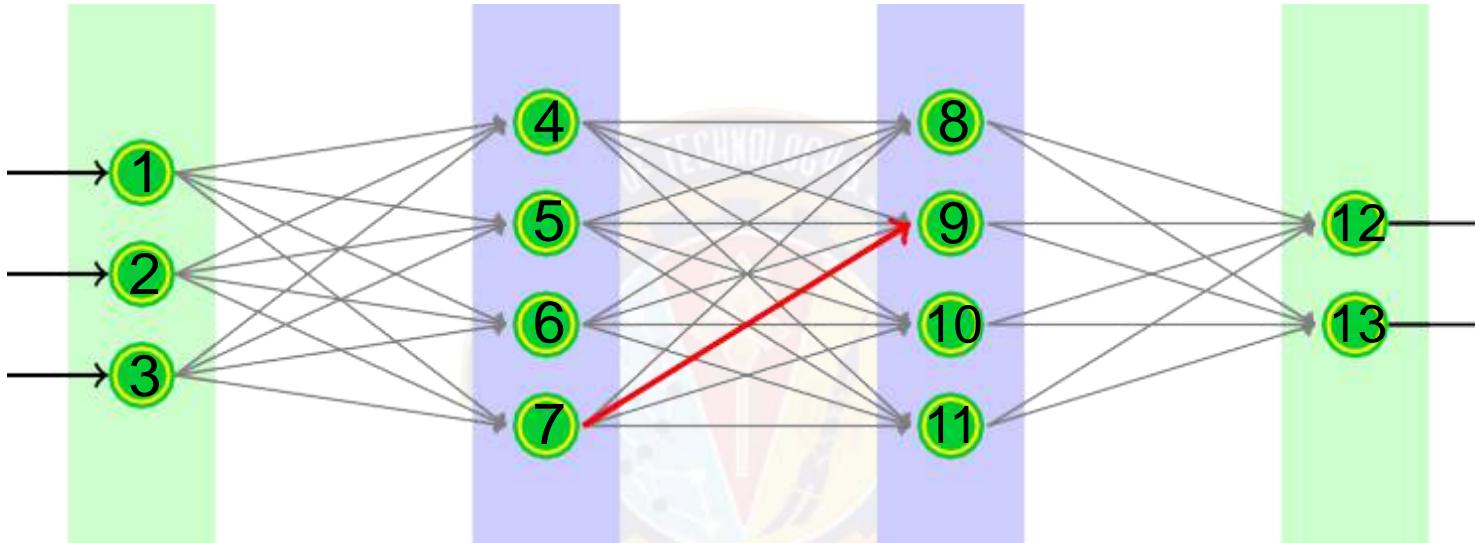
$w_{ji}$  weight associated with  $i$  th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

$t_j$  target output for unit  $j$

# Conventions Over The Network



$x_{ji}$   $i$  th input to unit  $j$  ( $x_{94}$  is highlighted)

$w_{ji}$  weight associated with  $i$  th input to unit  $j$

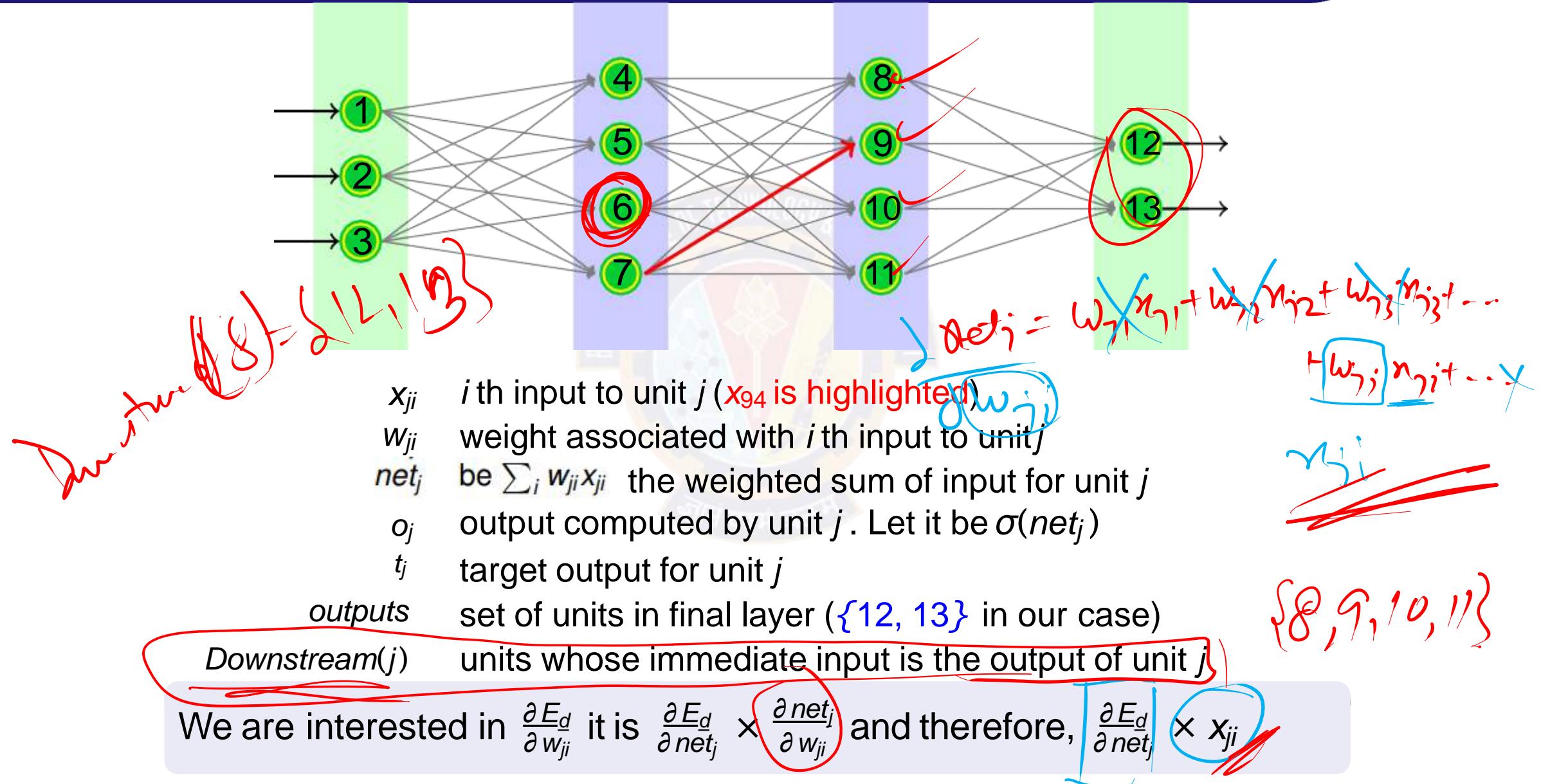
$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

$t_j$  target output for unit  $j$

**outputs** set of units in final layer ( $\{12, 13\}$  in our case)

# Conventions Over The Network



# Value of $\frac{\partial E_d}{\partial net_j}$ for output units

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial net_j}$$

$$o_j = \sigma(net_j)$$



# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

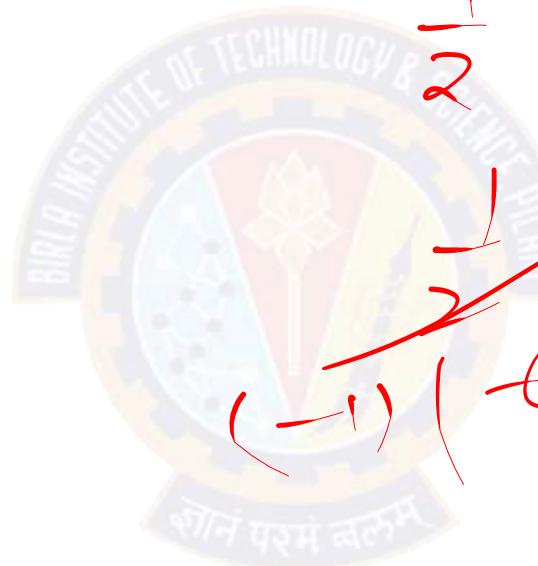
$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$$



# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2$$



$$\frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2$$

$$\frac{1}{2} \frac{\partial}{\partial o_j} (t_j - o_j)^2$$

$$\cancel{\frac{1}{2}} \cancel{\frac{\partial}{\partial o_j}} (t_j - o_j) \cancel{\frac{\partial}{\partial o_j}} (t_j - o_j)$$
$$(-1) (t_j - o_j) \cancel{(t_j - o_j)}$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\frac{\partial o_j}{\partial \text{net}_j} = \sigma'(\text{net}_j)$$



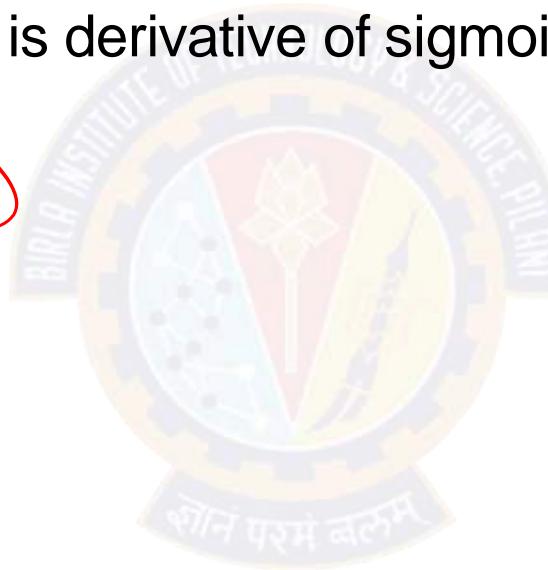
# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\frac{d}{dx} \sigma(x) = \frac{d}{dx} \frac{1}{1 + e^{-x}}$$



$$(-1) (1 + \bar{o}^n)^{-1} \frac{d}{dx} (1 + \bar{o}^n)$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\frac{d}{dx} \sigma(x) = \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x})$$

(  
D  
+  
 $e^{-x}$ )  
w.r.t



# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - \underline{e^{-x}})\end{aligned}$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\&= \cancel{(-1)(1 + e^{-x})} \cdot \cancel{(0 - e^{-x})} \\&= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} \quad \leftarrow \text{Red circles and annotations}\end{aligned}$$

$\frac{1}{1 + e^{-x}} \rightarrow \frac{e^{-x}}{1 + e^{-x}}$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2}(0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \boxed{\sigma(\text{net}_j)(1 - \sigma(\text{net}_j))}$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$
$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2}(0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

Therefore,  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \text{net}_j} \times x_{ji}$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2}(0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

$$\frac{\partial E_d}{\partial w_{ji}} = -\delta_j$$

Therefore,  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \text{net}_j} \times x_{ji} = \boxed{\eta(t_j - o_j)o_j(1 - o_j)x_{ji}}$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for hidden units

$$\begin{aligned}
 \frac{\partial E_d}{\partial \text{net}_j} &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E_d}{\partial \text{net}_k} \times \frac{\partial \text{net}_k}{\partial \text{net}_j} \\
 &= \sum_{k \in \text{Downstream}(j)} -\delta_k \times \frac{\partial \text{net}_k}{\partial \text{net}_j} \\
 &= \sum_{k \in \text{Downstream}(j)} -\delta_k \times \frac{\partial \text{net}_k}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j} \\
 &= \sum_{k \in \text{Downstream}(j)} -\delta_k \times w_{kj} \times o_j(1 - o_j)
 \end{aligned}$$

$\delta_j$  being  $-\frac{\partial E_d}{\partial \text{net}_j} = o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \delta_k \times w_{kj}$

Therefore,  $\Delta w_{ji} = \eta \delta_j x_{ji} = \boxed{\eta(o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \delta_k \times w_{kj}) x_{ji}}$

# Backpropagation (for 2 layers)

## Algorithm 14: Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers ✓

2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$  ✓

3 repeat

4     for each  $\langle \vec{x}, \vec{t} \rangle \in D$  do

5          $o_u$  = get output from network  $\forall$  unit  $u$

6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all output unit  $k$

7          $\delta_h = o_h(1 - o_h) \sum_{k \in outputs} (w_{kh} \delta_k)$  for all hidden unit  $h$

8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$

9 until converge;

# Backpropagation

- **Adding Momentum:** weight update during  $n^{th}$  iteration depend partially on the update that occurred during the  $(n - 1)^{th}$  iteration

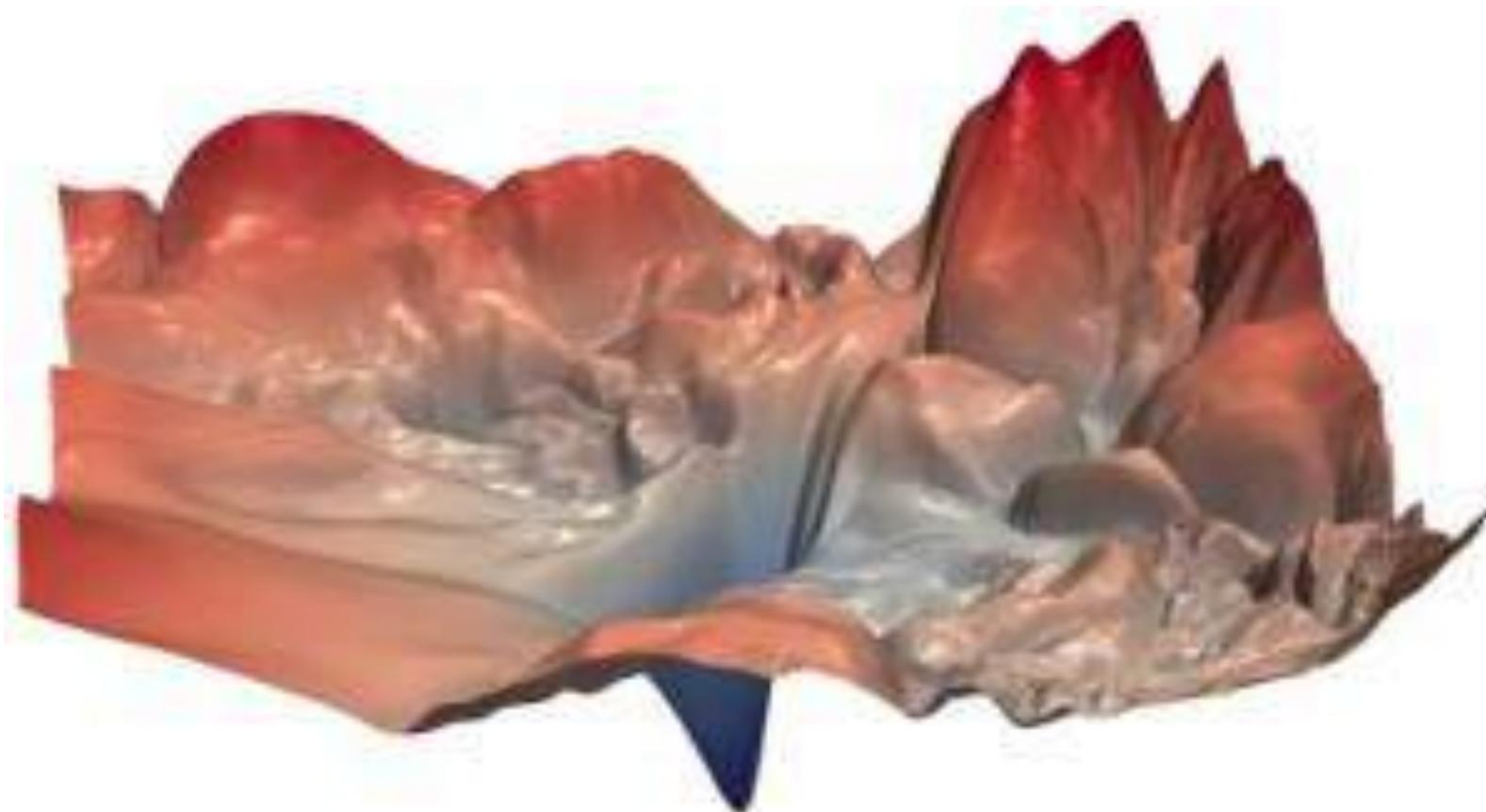
$$\Delta w_{jl}(n) = \eta \delta_j x_{jl} + \alpha \Delta w_{jl}(n-1)$$

- **Learning in arbitrary acyclic network:** for feed-forward networks of arbitrary depth,  $\delta_r$  value for a unit in hidden layer is determined as

$$\delta_r = o_r(1 - o_r) \sum_{s \in \text{Downstream}(r)} w_{sr} \times \delta_s$$

# Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error



# Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error
- No methods are known to predict with certainty when local minima will cause difficulties
- Suggested to use momentum, true gradient descent or multiple networks (initialized with different random weights)
- Any boolean function can precisely be represented by some network having only **two** layers of units (Cybenko 1989; Hornik et al. 1989)



# Thank You!

In our next session:



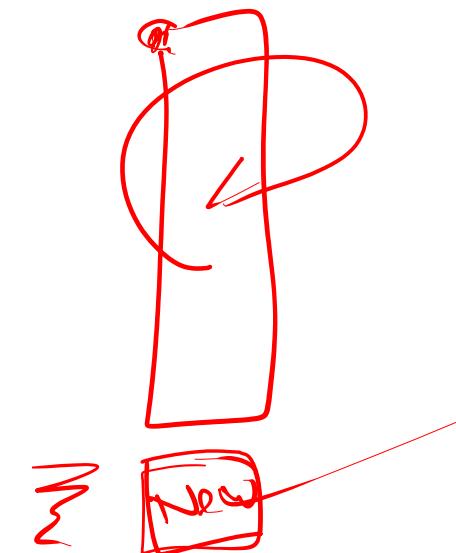
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Generalization Overfitting and Stopping Criteria

Kamlesh Tiwari

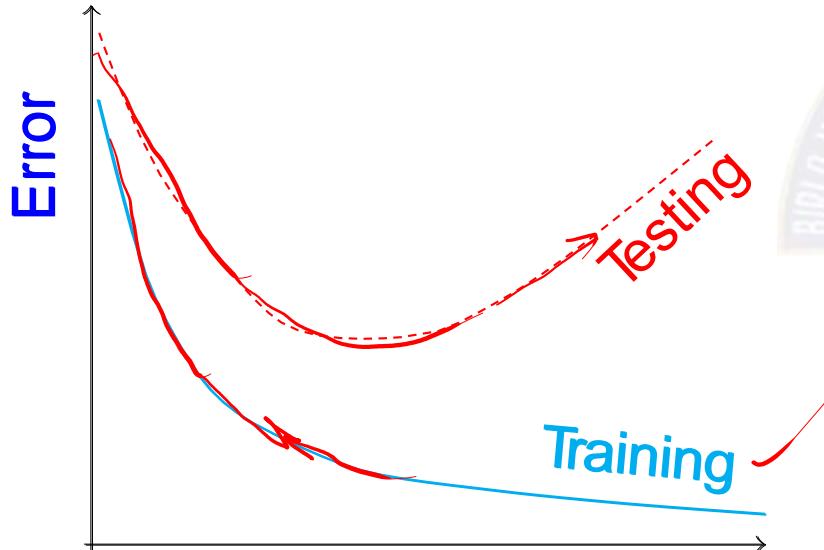
# Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



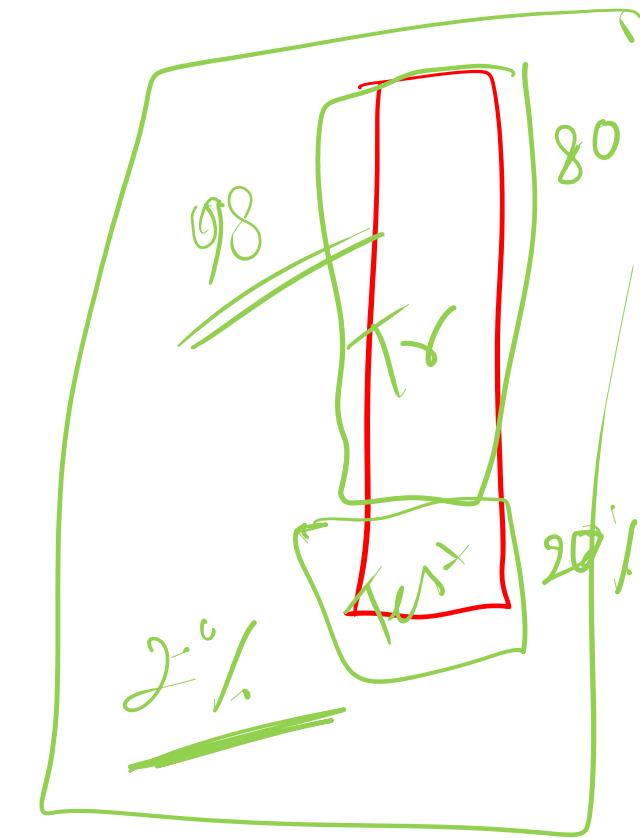
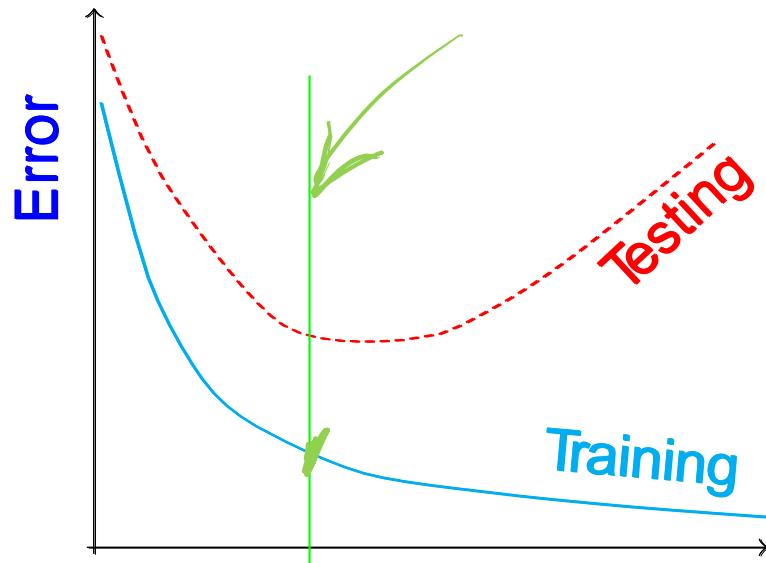
# Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



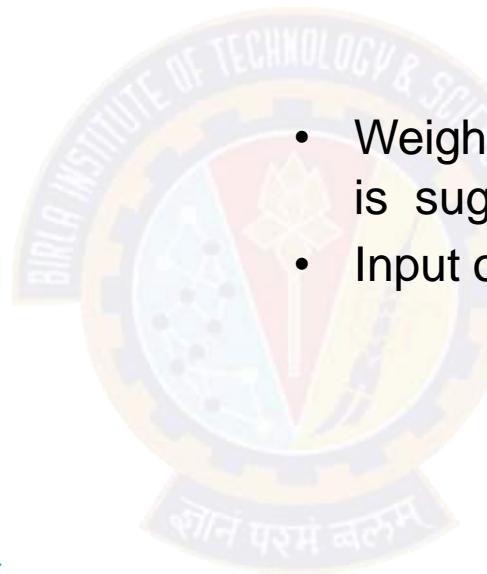
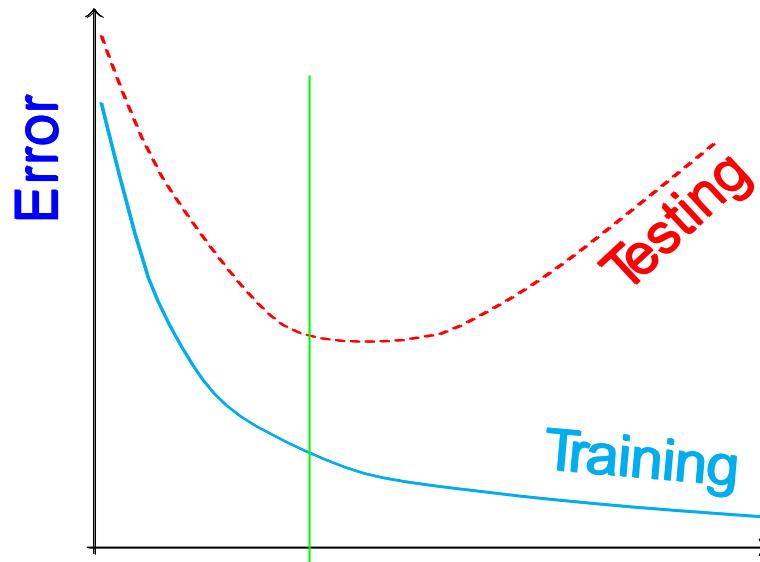
# Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



# Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy

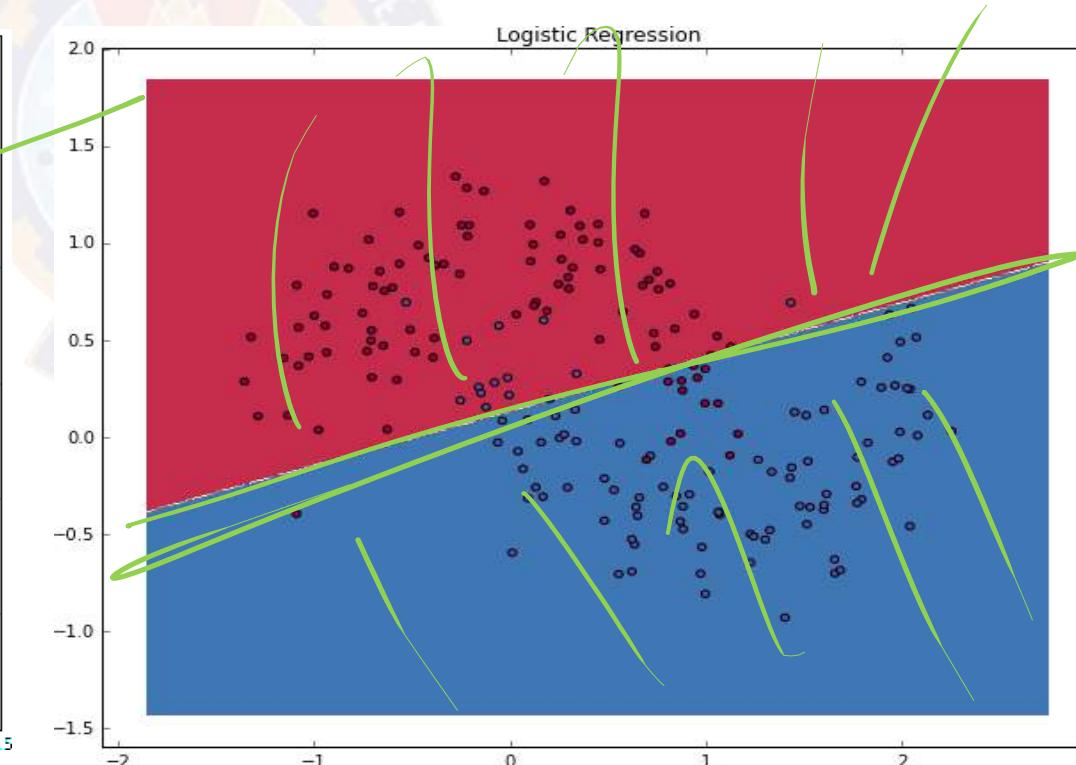
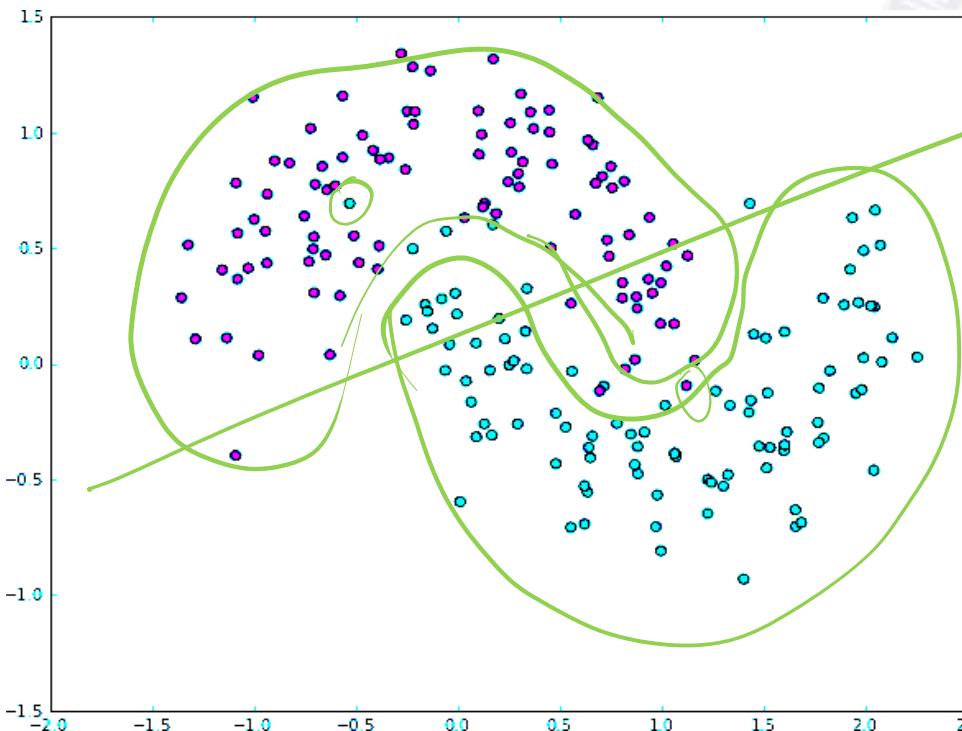


- Weight decay or use of validation set ( $k$ -fold ?) is suggested
- Input or output encoding can be used

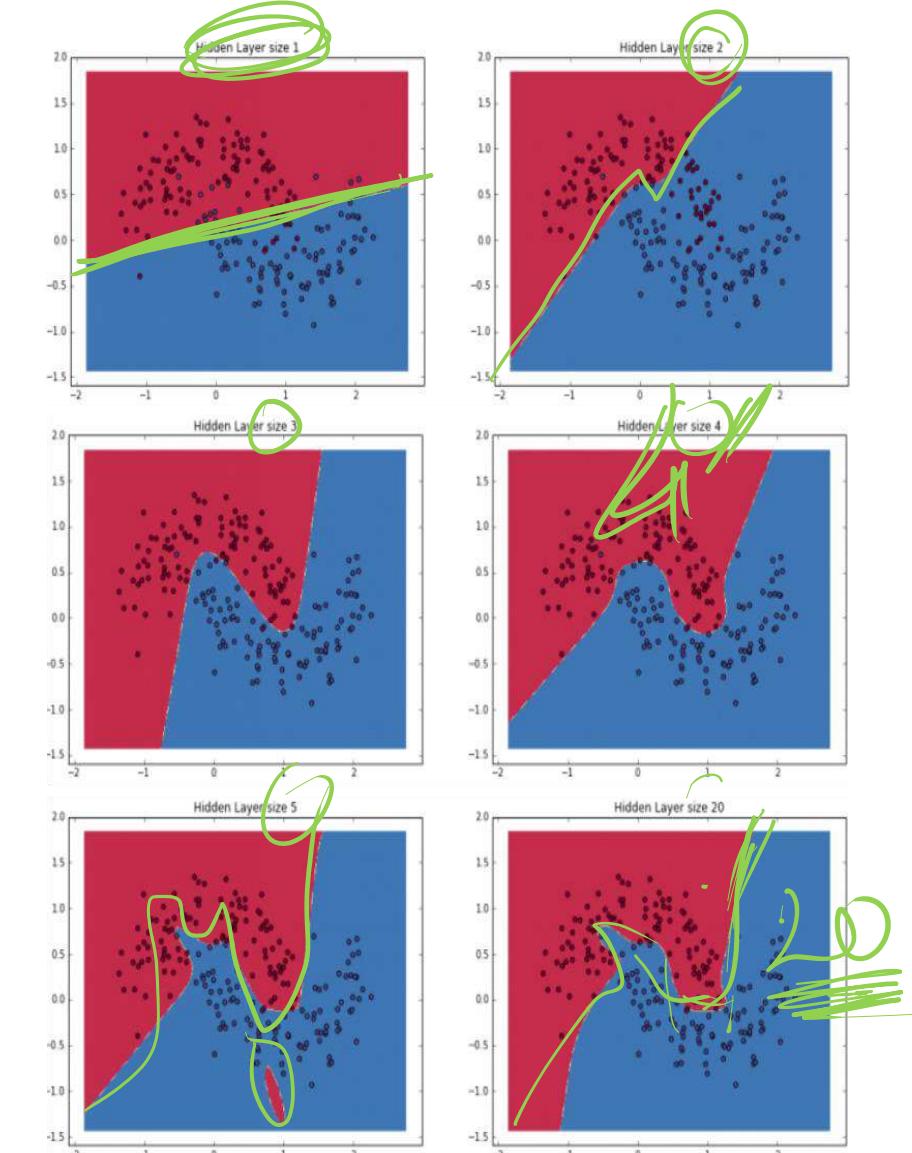
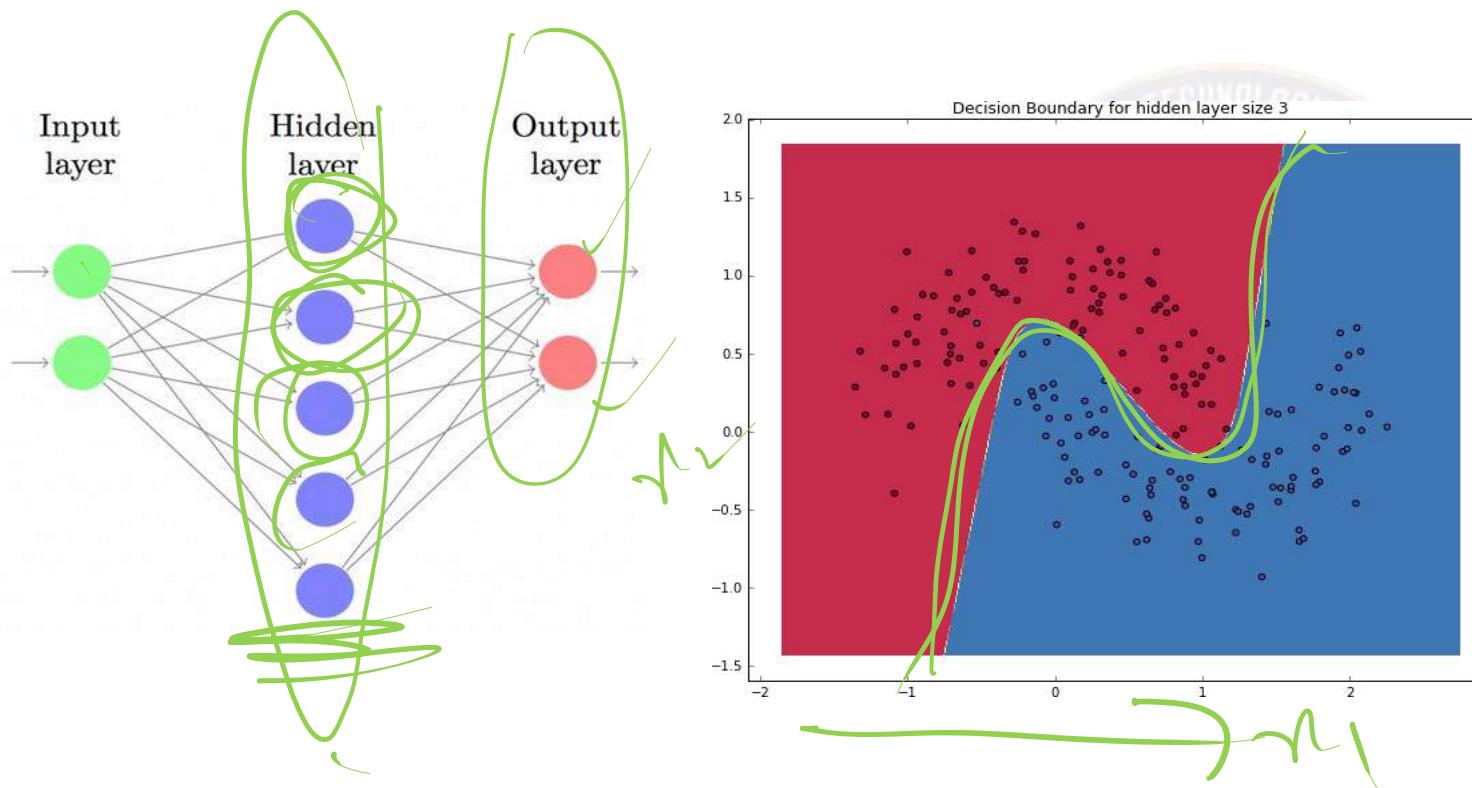


# Coding Example 1

Consider two class data  
Logistic Regression  
Neural Network  
Decision Boundary  
Bigger hidden layer



# Coding Example 1





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# ANN and Deep Learning

**Dr. Sugata Ghosal**

---

[sugata.ghosal@pilani.bits-pilani.ac.in](mailto:sugata.ghosal@pilani.bits-pilani.ac.in)

# Syllabus

M1	<b>Artificial Neural Network:</b> Introduction and Background, Discrimination power of single neuron, Training a single perceptron (delta rule) , Multilayer Neural Networks, Activation functions and Loss functions, Backpropagation
M2	<b>Deep Learning:</b> Introduction to end to end learning, Abstractions of features using deep layers, Hyper parameter tuning, Regularization for Deep Learning, Dropout
M3	<b>Convolution Networks with Deep Learning:</b> CNN, Pooling, Variants of pooling functions, CNN with Fully connected Networks, RCNN, Faster RCNN
M4	<b>Sequence Modeling in Neural Network:</b> Architecture of RNN , Unfolding of RNN, Training RNN, LSTM and its applications
M5	<b>Autoencoders with Deep Learning:</b> Undercomplete Autoencoders, Regularized Autoencoders, Variational autoencoders, Manifold learning with Autoencoders, Applications of Autoencoders
M6	<b>Generative deep learning models:</b> Boltzmann Machine, Restricted Boltzmann Machine, Deep Belief Machines, GAN, Applications of GAN



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

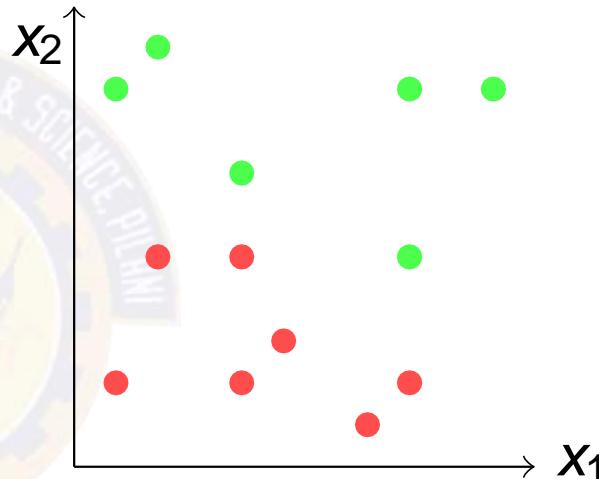
## Background of NN

# Linear Classification

Consider Following data

$x_1$	$x_2$	y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red

Data is in 2D, so let us visualize

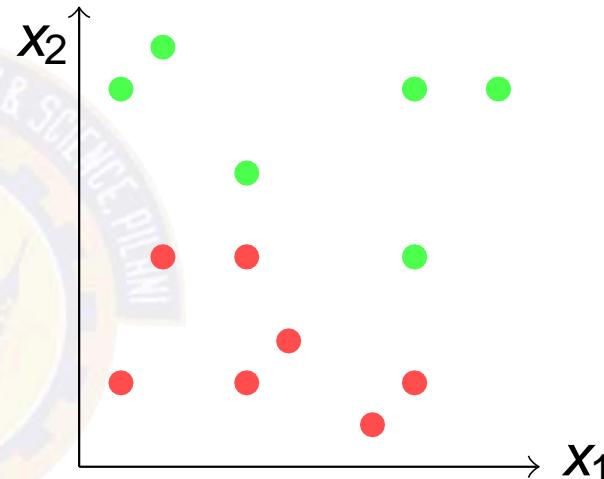


# Linear Classification

Consider Following data

$x_1$	$x_2$	y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red

Data is in 2D, so let us visualize



- Data looks **linearly separable**
- What is the decision boundary?

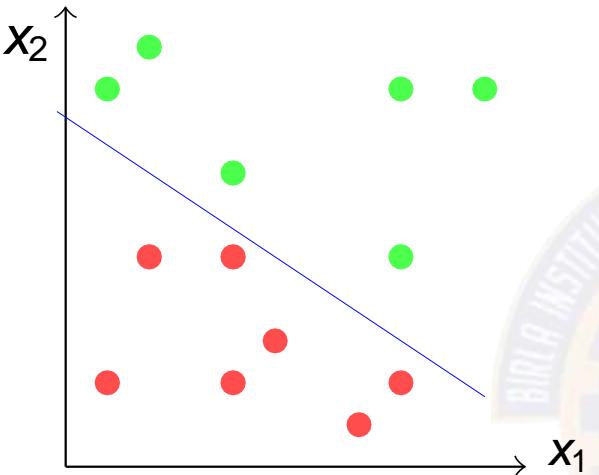
Many Possibilities, such as

if  $(2x_1 + 3x_2 - 25 > 0)$  it is **green**  
otherwise **red**

# What about this arrangement?

With chosen *decision boundary*

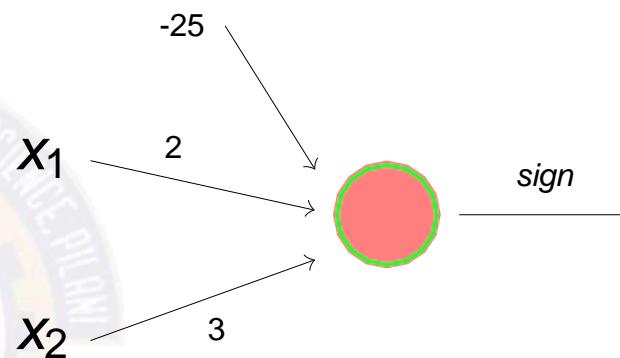
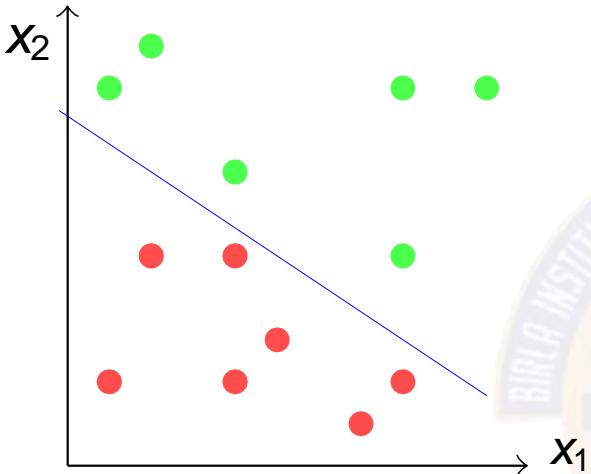
$$2x_1 + 3x_2 - 25 = 0$$



# What about this arrangement?

With chosen *decision boundary*

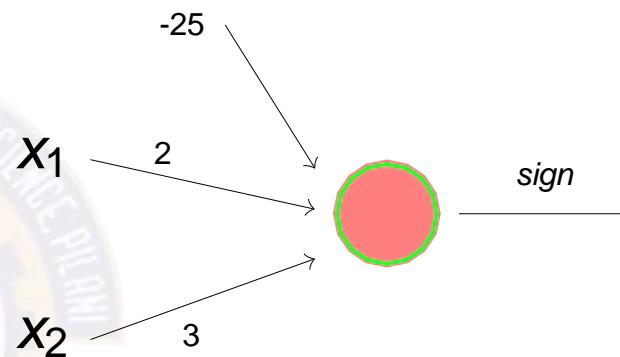
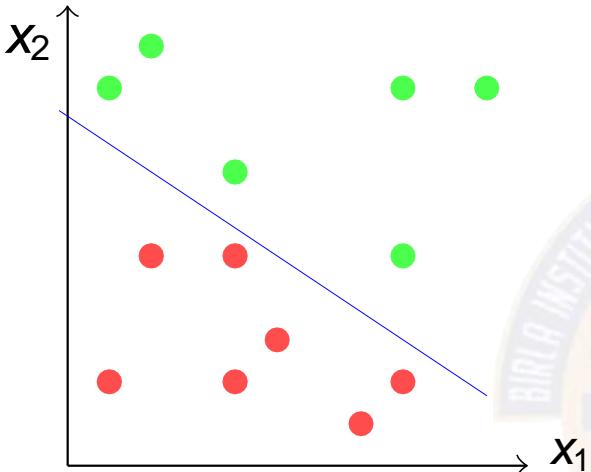
$$2x_1 + 3x_2 - 25 = 0$$



# What about this arrangement?

With chosen *decision boundary*

$$2x_1 + 3x_2 - 25 = 0$$



- This illustration is called as **perceptron**
- Provides a graphical way to represent the linear boundary
- Values **3, 2, -25** are its parameters or weights

Given a data

“How to find appropriate parameters?” is an important **issue**

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm: Perceptron training rule

---

- 1 Begin with **random** weights  $w$
  - 2 **repeat**
  - 3     **for each** *misclassified* example **do**
  - 4          $w_i = w_i + \eta(t - o)x_i$
  - 5 **until** *all training examples are correctly classified*;
  - 6 **return**  $w$
-

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?
  - Weight does not change when classification is correct

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?
  - Weight does not change when classification is correct
  - If perceptron outputs -1 when target is +1:

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- **Why would this strategy converge?**
  - Weight does not change when classification is correct
  - If perceptron outputs -1 when target is +1: weight increases ↑

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?
  - Weight does not change when classification is correct
  - If perceptron outputs -1 when target is +1: weight increases ↑
  - If perceptron outputs +1 when target is -1:

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?
  - Weight does not change when classification is correct
  - If perceptron outputs -1 when target is +1: weight increases ↑
  - If perceptron outputs +1 when target is -1: weight decreases ↓

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- **Why would this strategy converge?**

- Weight does not change when classification is correct
- If perceptron outputs -1 when target is +1: weight increases ↑
- If perceptron outputs +1 when target is -1: weight decreases ↓

Conversion with perceptron training rule is subject to linear separability of training example and appropriate  $\eta$

# Example

Consider the same data

X <sub>1</sub>	X <sub>2</sub>	Y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red



# Example

Consider the same data

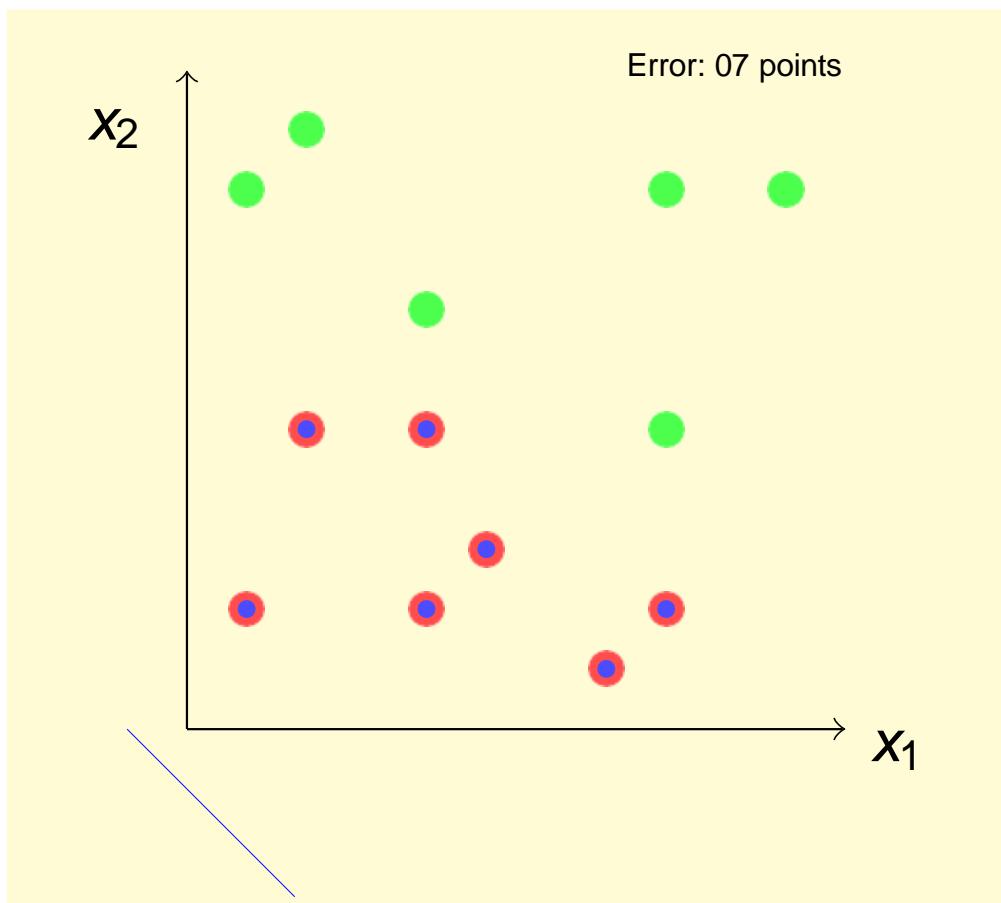
X <sub>1</sub>	X <sub>2</sub>	Y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red

$$\eta = 0.01$$

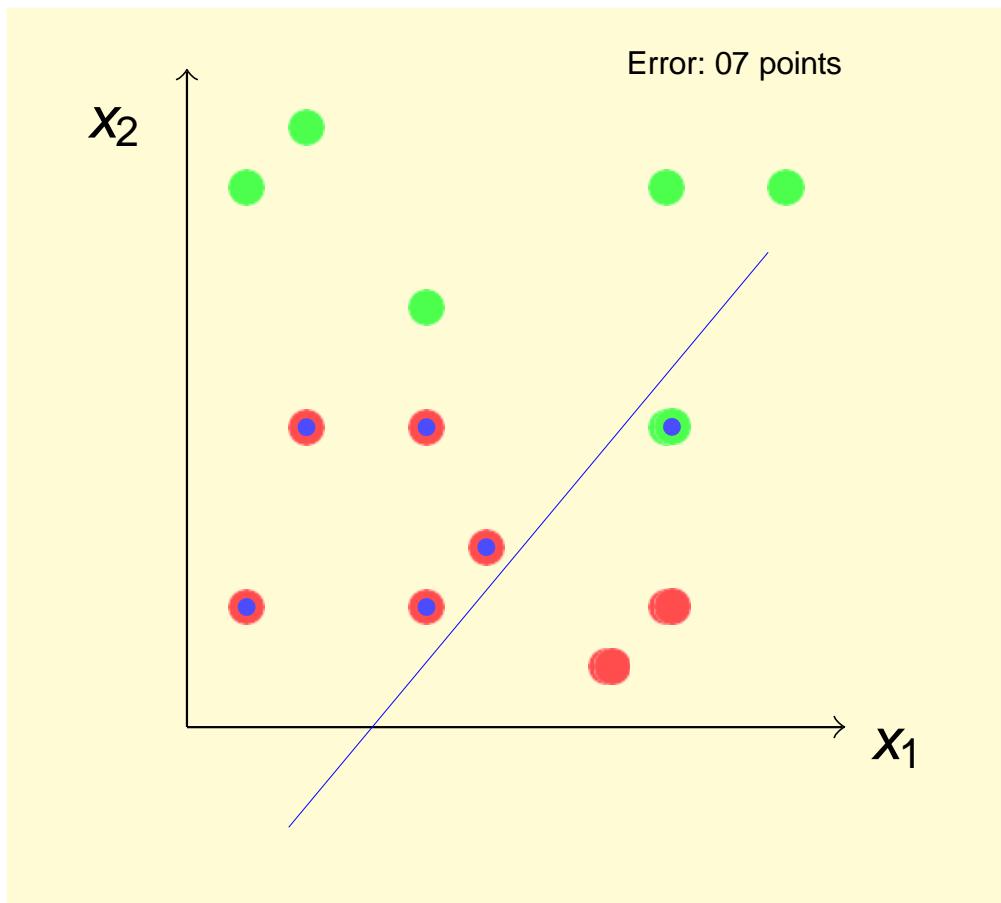
w0=0.500, w1=0.500, w2=0.500	err=7
w0=0.360, w1=-0.120, w2=0.100	err=6
w0=0.300, w1=-0.180, w2=0.060	err=5
w0=0.240, w1=-0.140, w2=0.140	err=4
w0=0.180, w1=-0.200, w2=0.100	err=5
w0=0.120, w1=-0.160, w2=0.180	err=4
w0=0.080, w1=-0.060, w2=0.180	err=5
w0=0.020, w1=-0.120, w2=0.140	err=4
w0=-0.040, w1=-0.180, w2=0.100	err=5
w0=-0.100, w1=-0.140, w2=0.180	err=4
w0=-0.140, w1=-0.040, w2=0.180	err=5
w0=-0.200, w1=-0.100, w2=0.140	err=3
w0=-0.260, w1=-0.160, w2=0.100	err=4
w0=-0.320, w1=-0.120, w2=0.180	err=3
w0=-0.360, w1=-0.020, w2=0.180	err=3
w0=-0.420, w1=-0.080, w2=0.140	err=2
w0=-0.420, w1=-0.080, w2=0.240	err=2
Fourteen more iterations	
w0=-0.900, w1=-0.020, w2=0.180	err=1
w0=-0.900, w1=-0.020, w2=0.240	err=2
w0=-0.920, w1=0.020, w2=0.220	err=2
w0=-0.960, w1=-0.020, w2=0.220	err=3
w0=-0.980, w1=0.020, w2=0.200	err=2
w0=-1.000, w1=0.060, w2=0.180	err=2
w0=-1.040, w1=0.020, w2=0.180	err=0



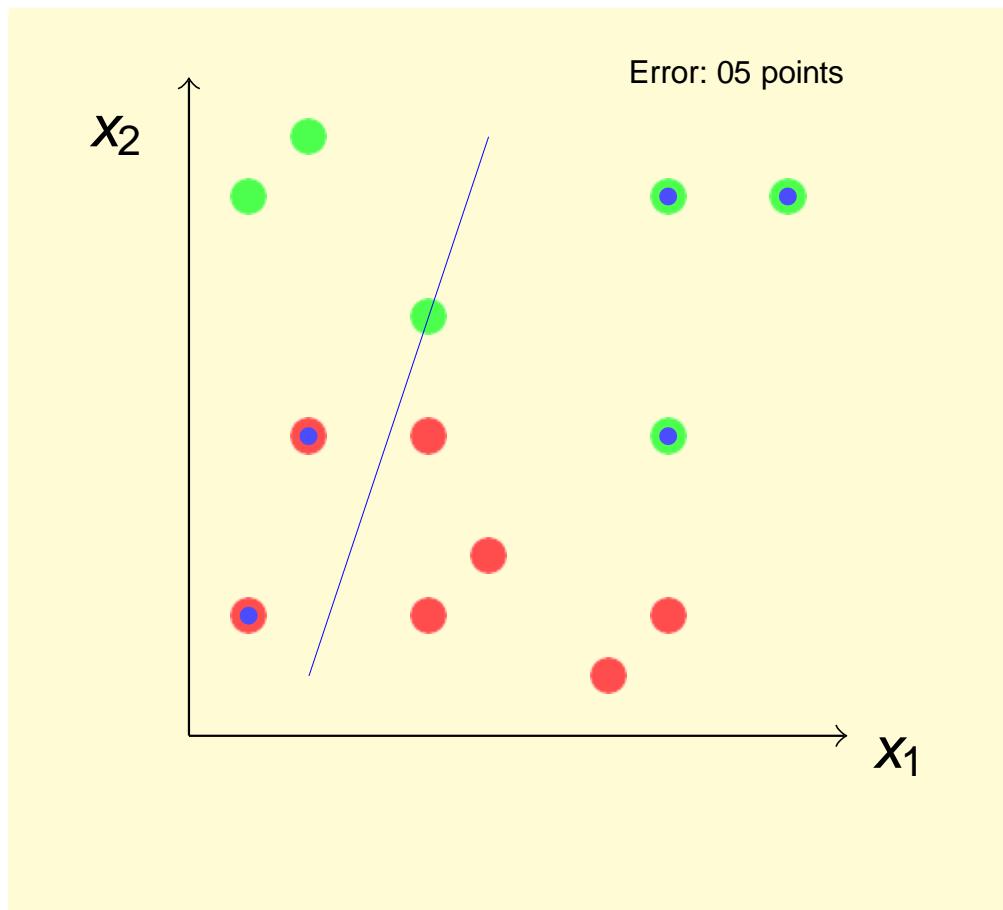
# Visual Interpretation



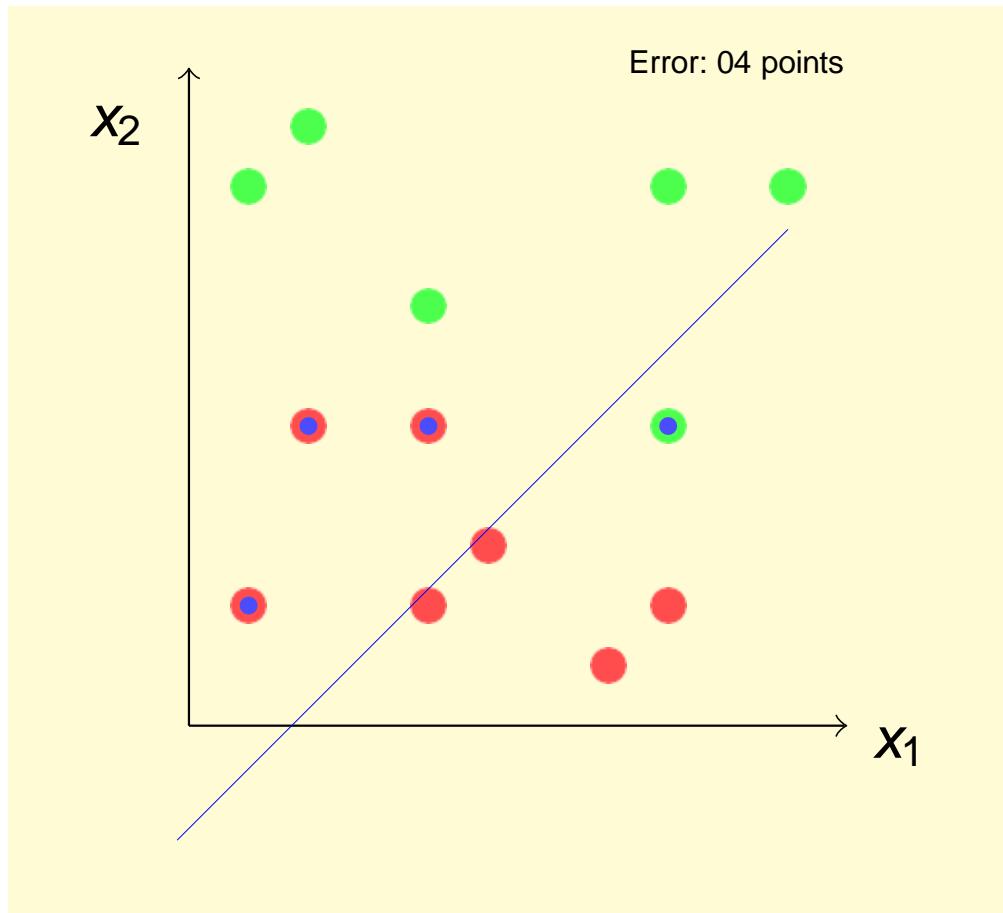
# Visual Interpretation



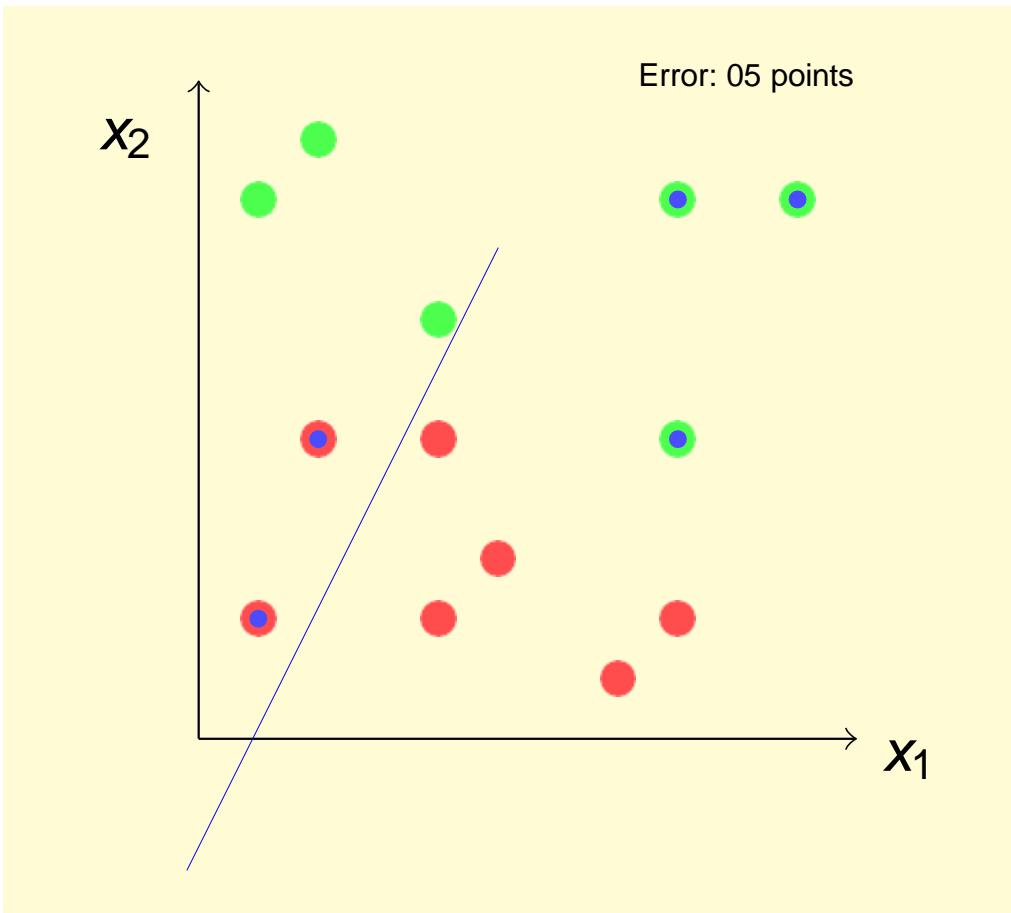
# Visual Interpretation



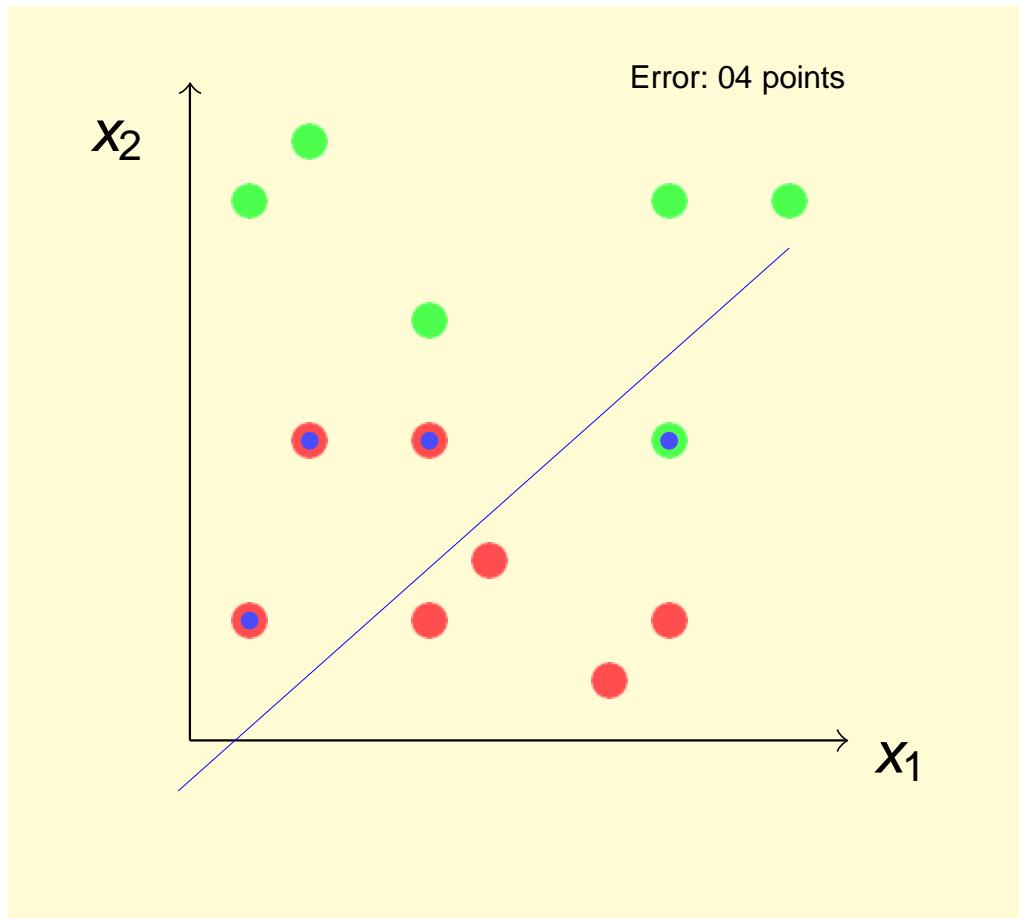
# Visual Interpretation



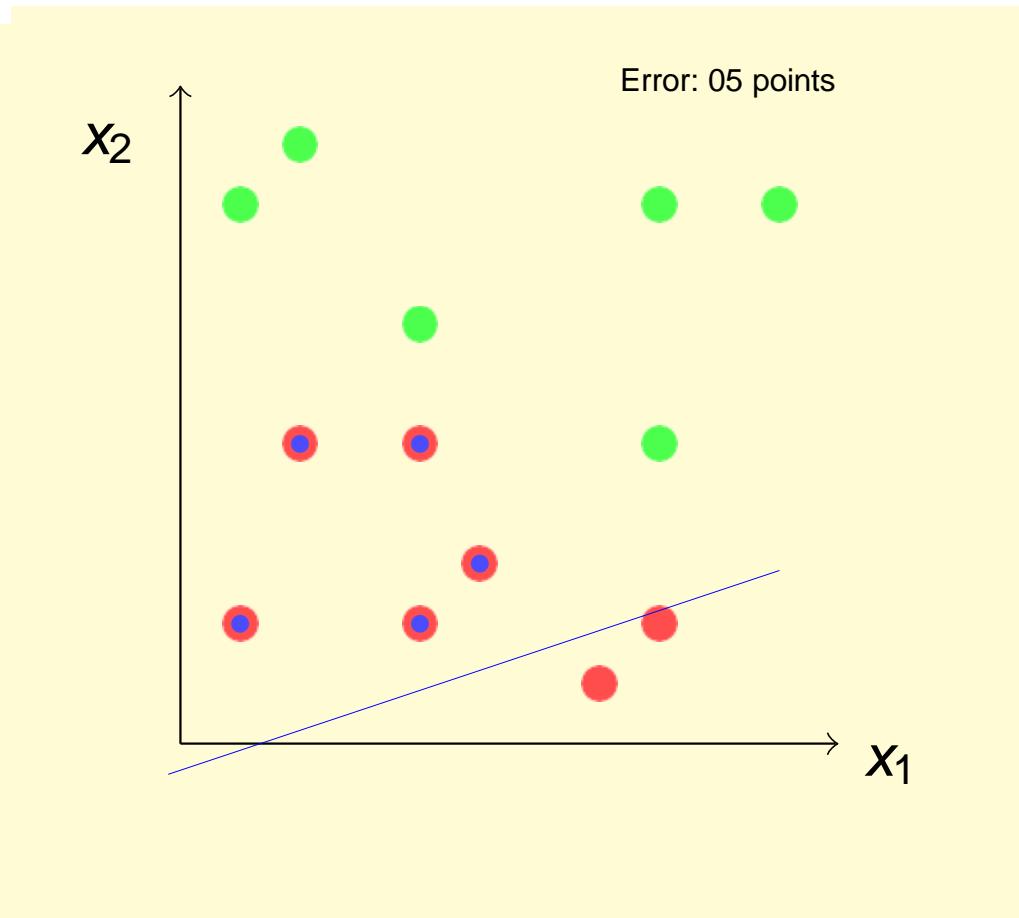
# Visual Interpretation



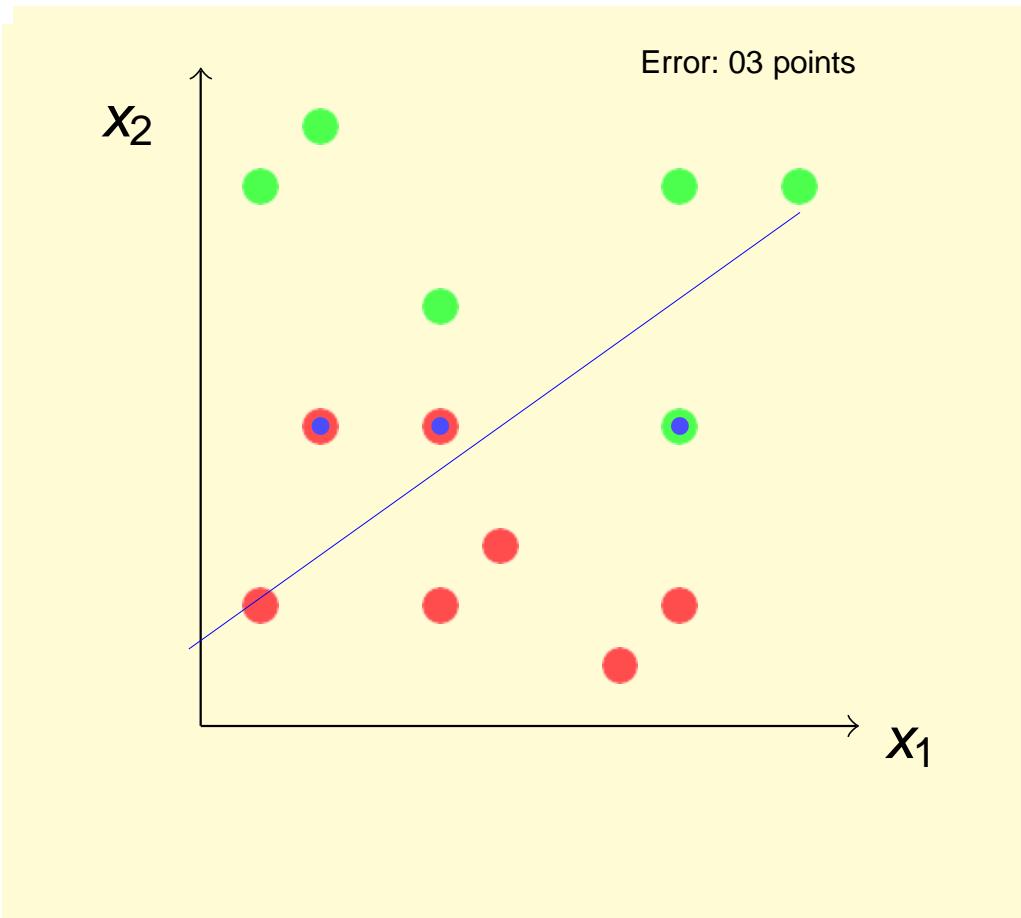
# Visual Interpretation



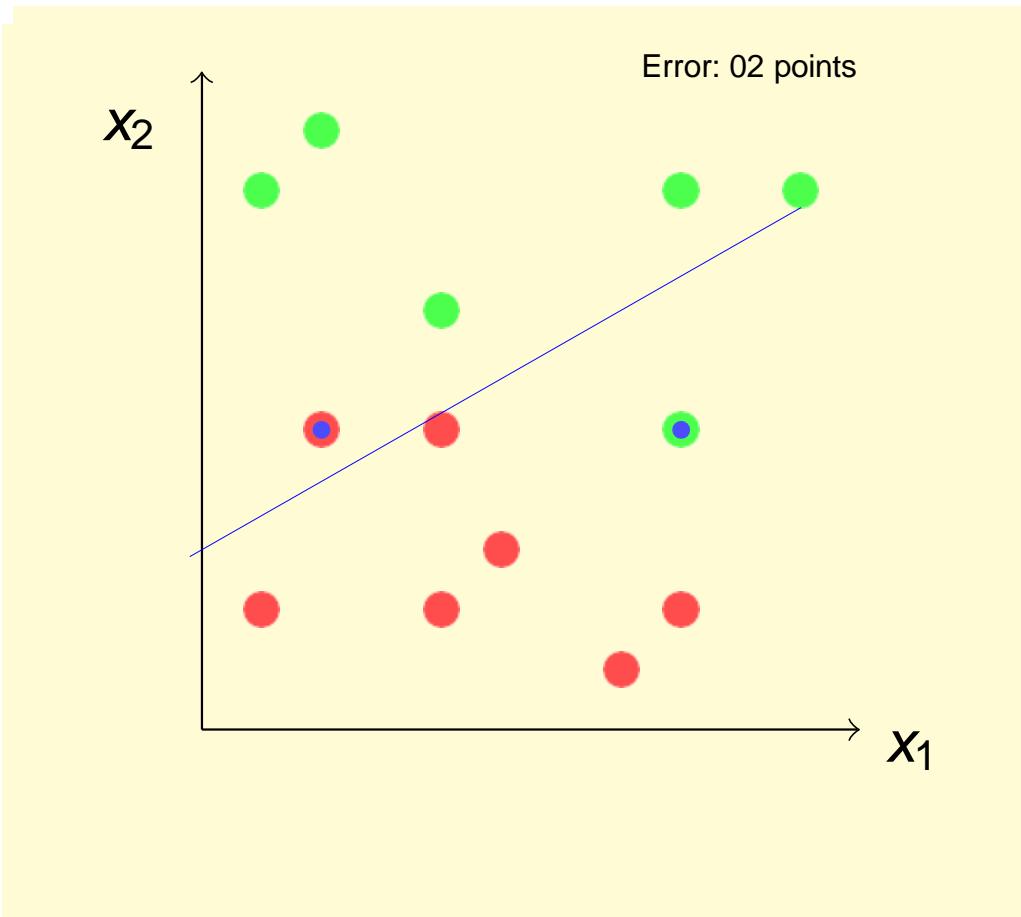
# Visual Interpretation



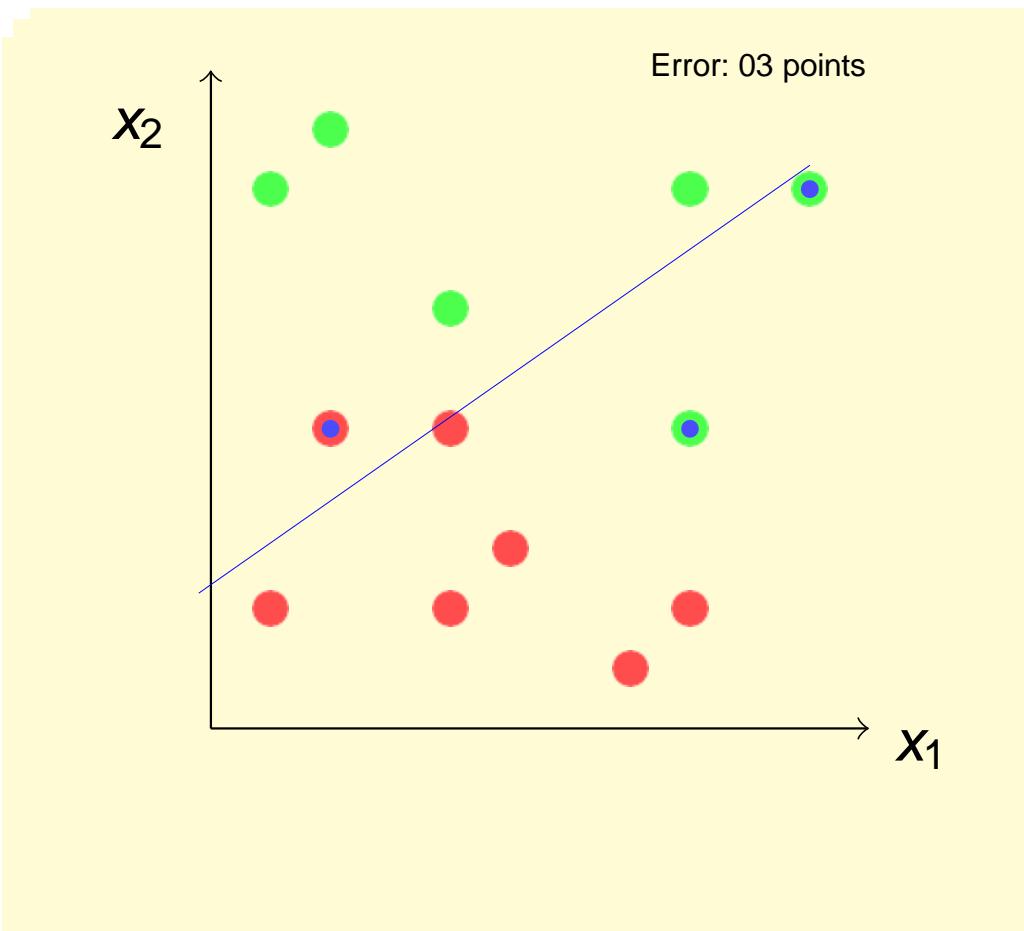
# Visual Interpretation



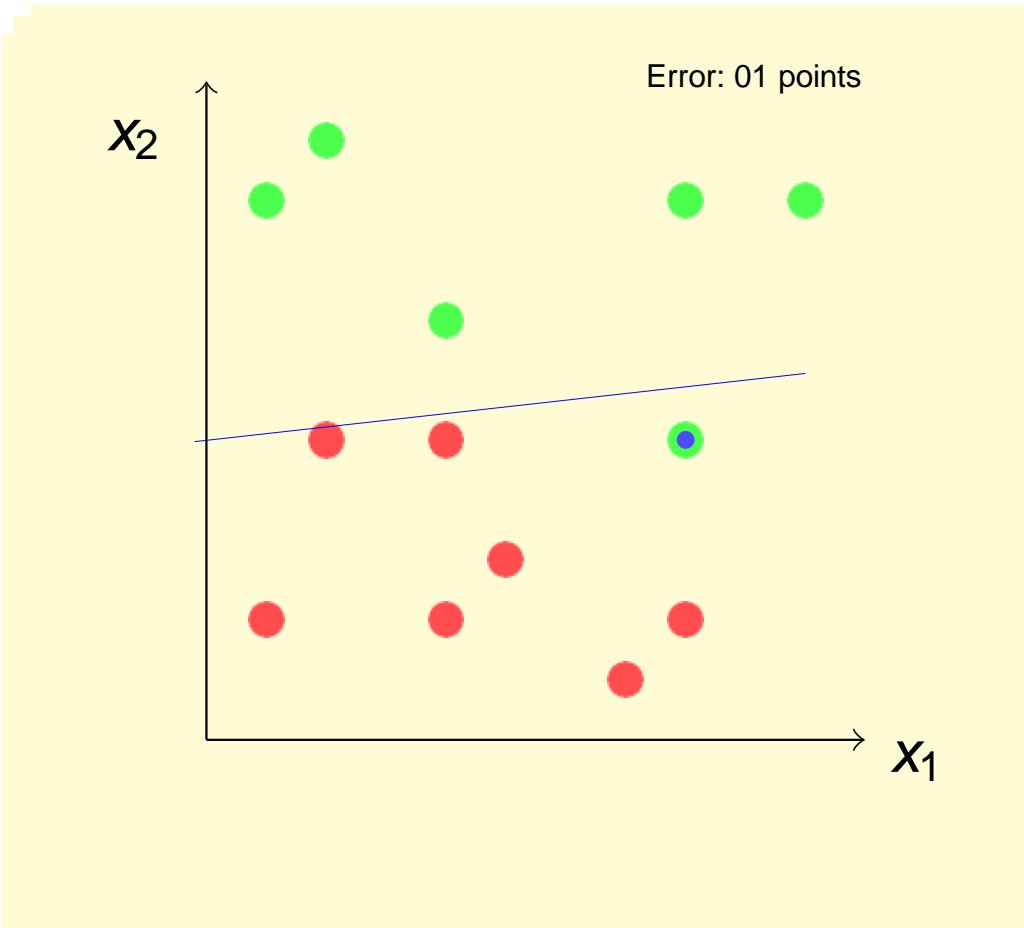
# Visual Interpretation



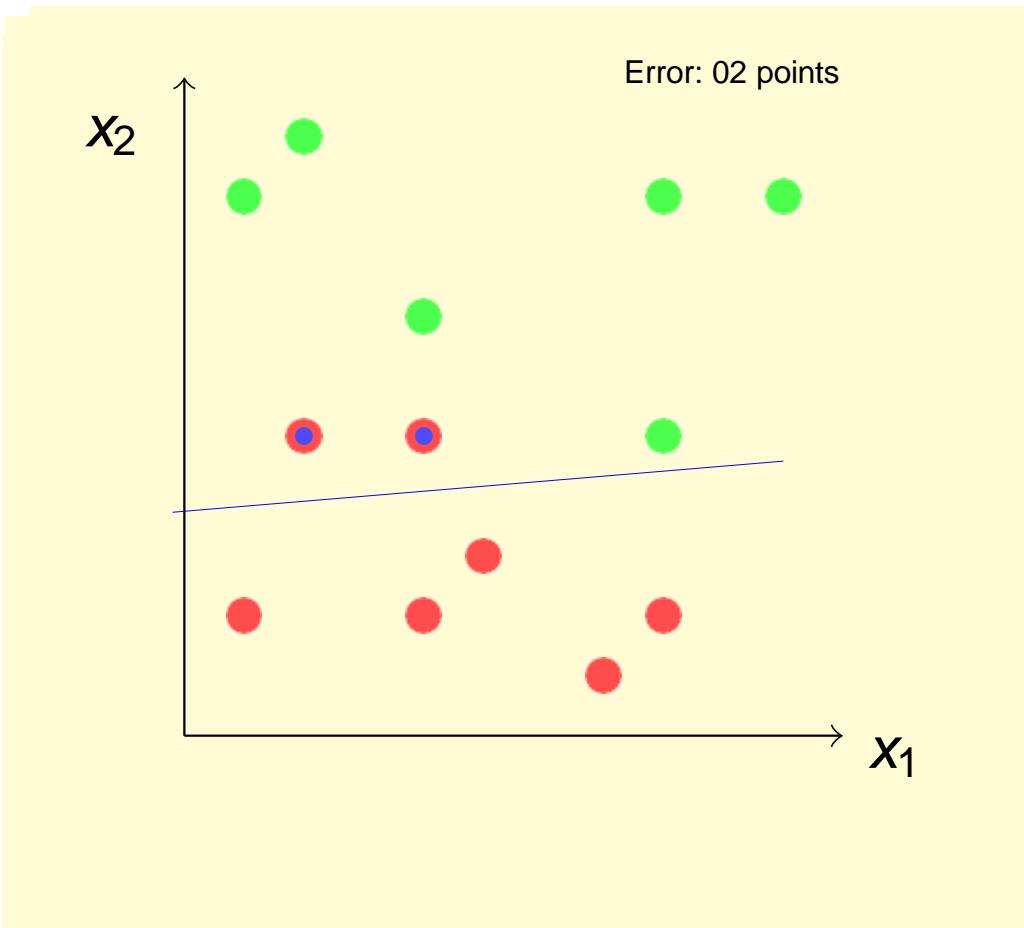
# Visual Interpretation



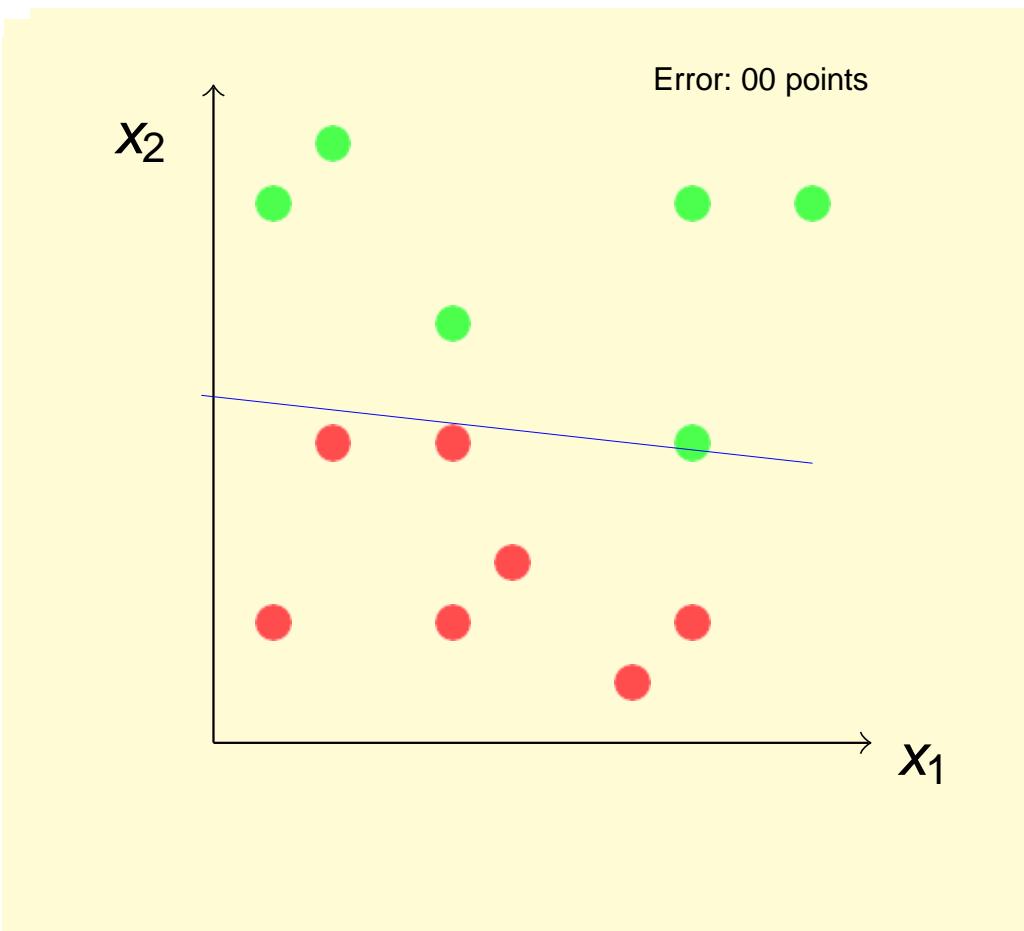
# Visual Interpretation



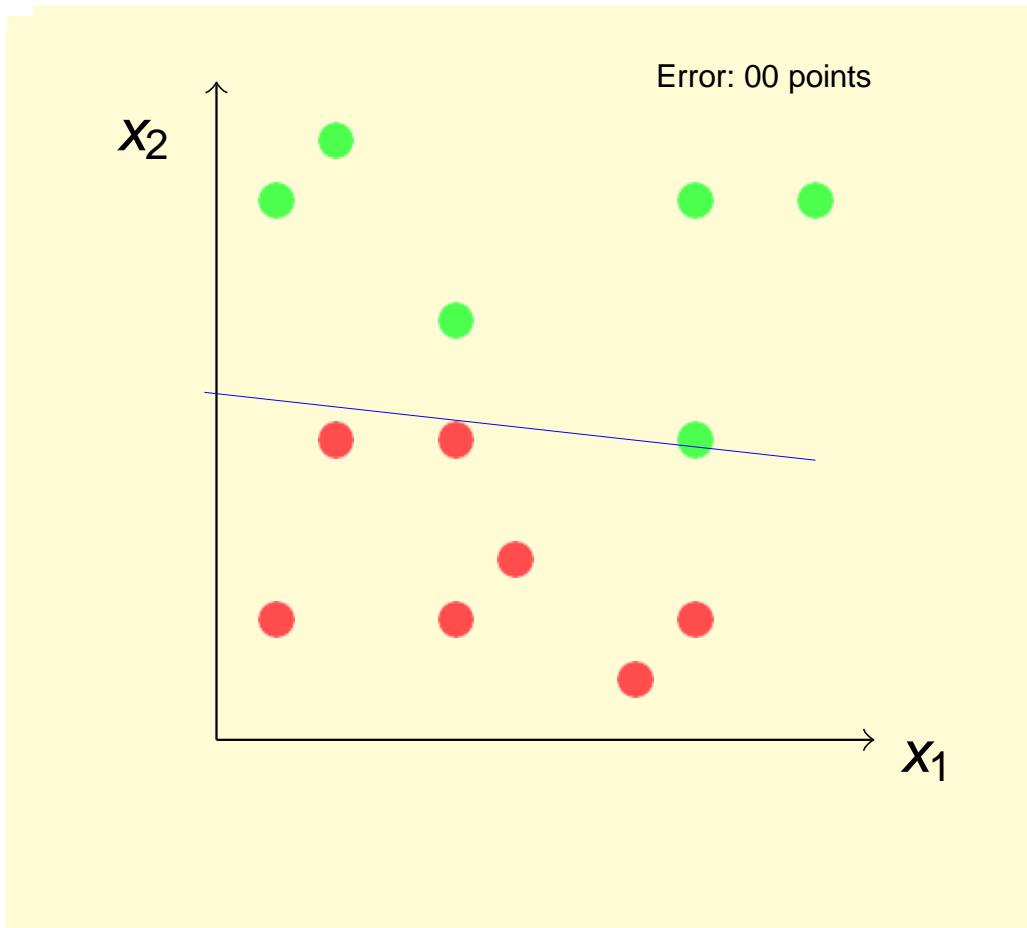
# Visual Interpretation



# Visual Interpretation



# Visual Interpretation



- Conversion is not gradual. (Error is NOT reducing monotonically)
- It is difficult to decide when to stop if data is not linearly separable



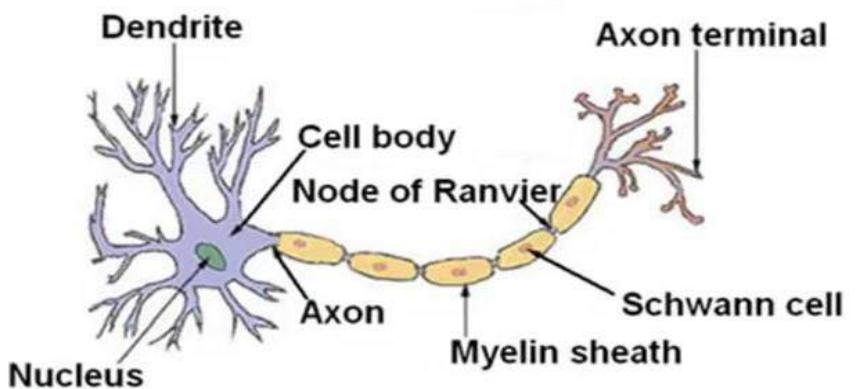
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to Perceptron

# Neural Network (NN)

NN is biologically motivated learning model that mimic human brain

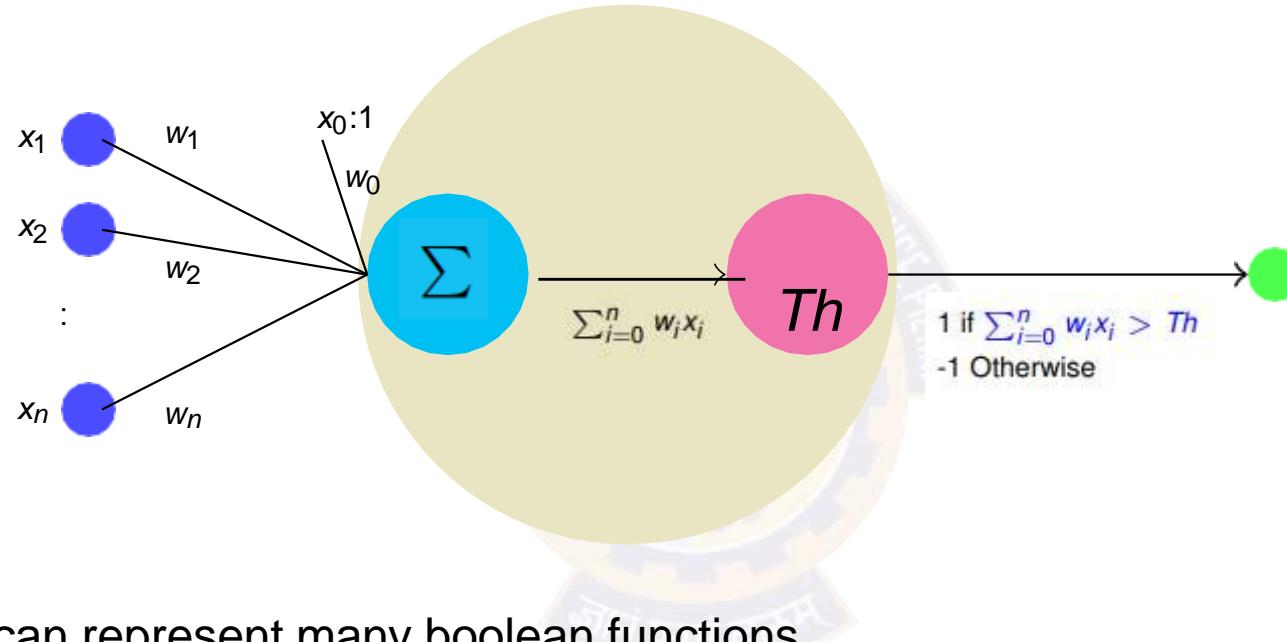
- Started by *W. McCulloch* study on working of neurons in 1943
- MADALINE (1959), an adaptive filter that eliminates echoes on phone lines was the first neural network
- Popularity of Neural Network diminished in 90's but, due to advances in **processing power** and availability of **large data** it again became state-of-the-art



- Cell, Axon, Synapses, Molecules, and Dendrites
- Humans have  $10^{11}$  neurons, each connected to  $10^4$  others, switches in  $10^{-3}$  sec

# A Single Perceptron

## Perceptron representation

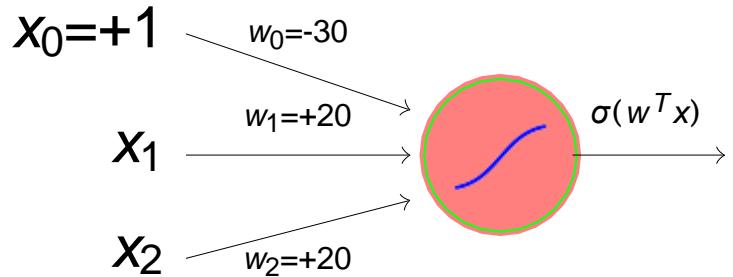


- A single perceptron can represent many boolean functions
- Any  $m$ -of- $n$  function (at least  $m$  of the  $n$  inputs must be true) can be represented by perceptron. OR ( $m=1$ ) and AND ( $m=n$ )

Two layer NN can represent any boolean function (Consider SOP)

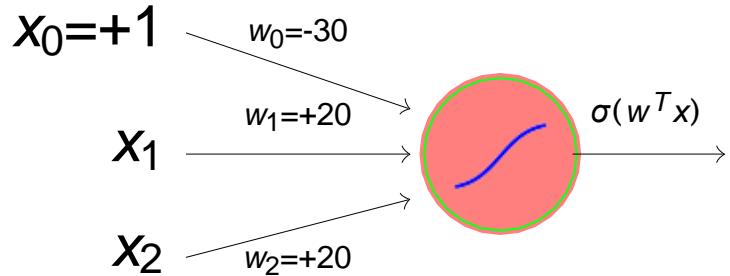
# An Example

Consider a perceptron with output 0/1 as below



# An Example

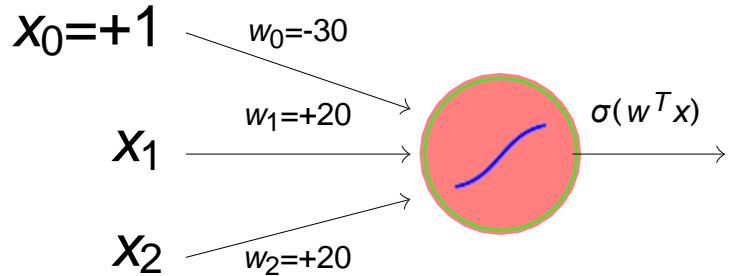
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	

# An Example

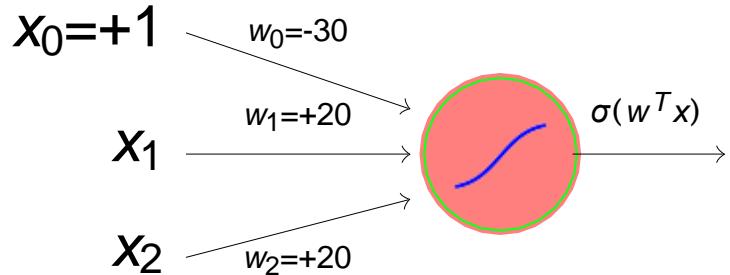
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	

# An Example

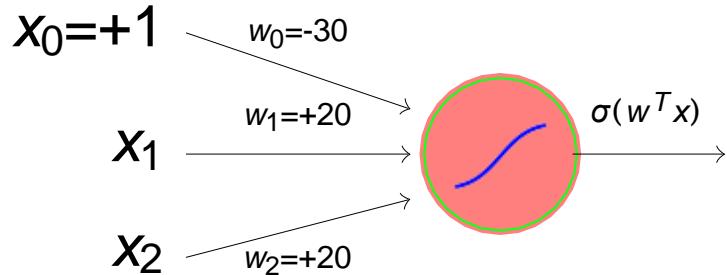
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	

# An Example

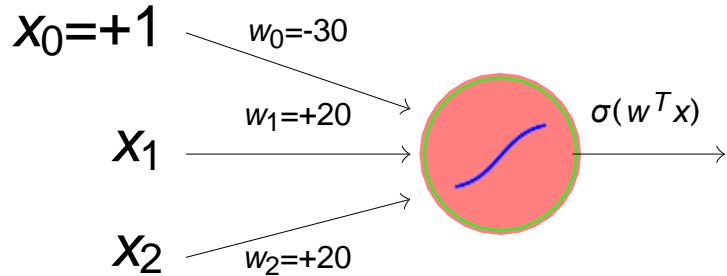
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	0
1	1	

# An Example

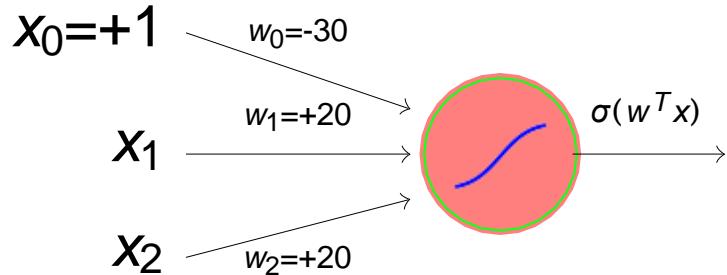
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	0
1	1	1

# An Example

Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	0
1	1	1

This perceptron computes logical AND

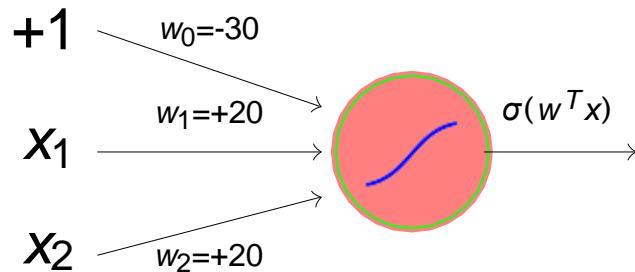
- $w_0 = -10$  gives logical OR
- $w_0 = 10$ ,  $w_1 = -20$  with single input gives logical NOT
- XOR is not possible



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Perceptron in Layer and Network

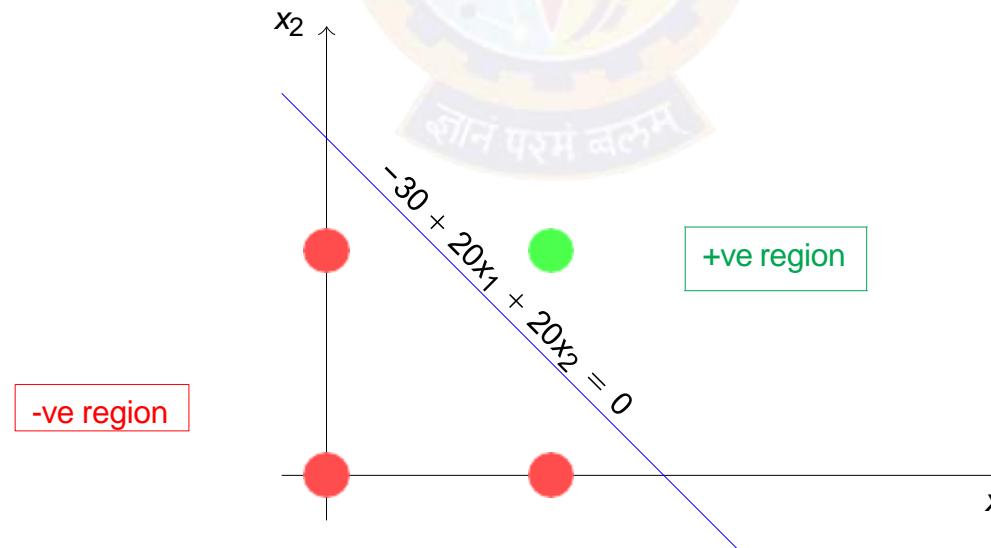
# Essentially it Represents A Decision Boundary



Provides **positive** classification if

$$-30 + 20x_1 + 20x_2 \geq 0$$

Represents a linear decision boundary

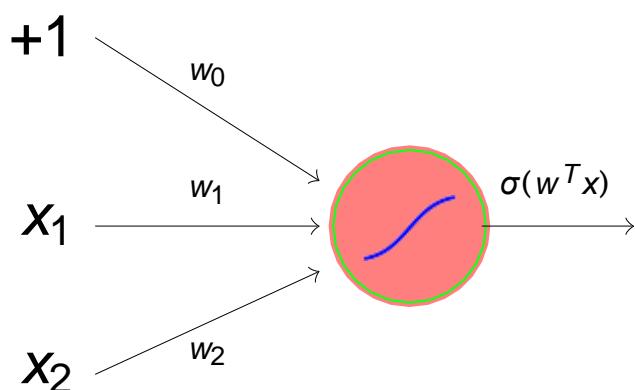


# An Example

Design a perceptron for

$x_1$	$x_2$	Classification
0	0	0
0	1	0
1	0	1
1	1	0

Let us assume following



We have following four equations

$$w_0 + w_1 \times (0) + w_2 \times (0) < 0 \quad (1)$$

$$w_0 + w_1 \times (0) + w_2 \times (1) < 0 \quad (2)$$

$$w_0 + w_1 \times (1) + w_2 \times (0) \geq 0 \quad (3)$$

$$w_0 + w_1 \times (1) + w_2 \times (1) < 0 \quad (4)$$

By (1)  $w_0 < 0$  so let  $w_0 = -1$

By (2)  $w_0 + w_2 < 0$  so let  $w_2 = -1$

By (3)  $w_0 + w_1 \geq 0$  so let  $w_1 = 1.5$

By (4)  $w_0 + w_1 + w_2 < 0$  that is valid

So  $(w_0, w_1, w_2) = (-1, -1, 1.5)$

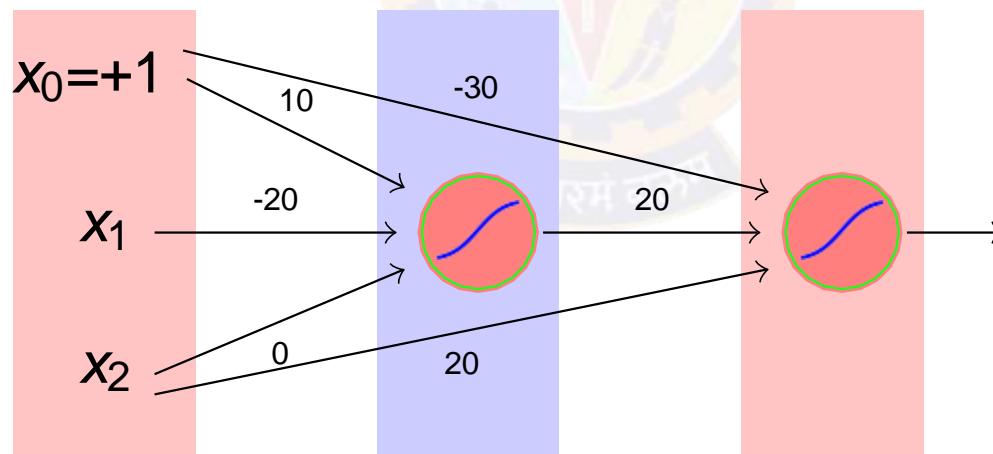
Other possibilities are also there

# An Example

Design a neural network for

$x_1$	$X_2$	Classification
0	0	0
0	1	0
1	0	1
1	1	0

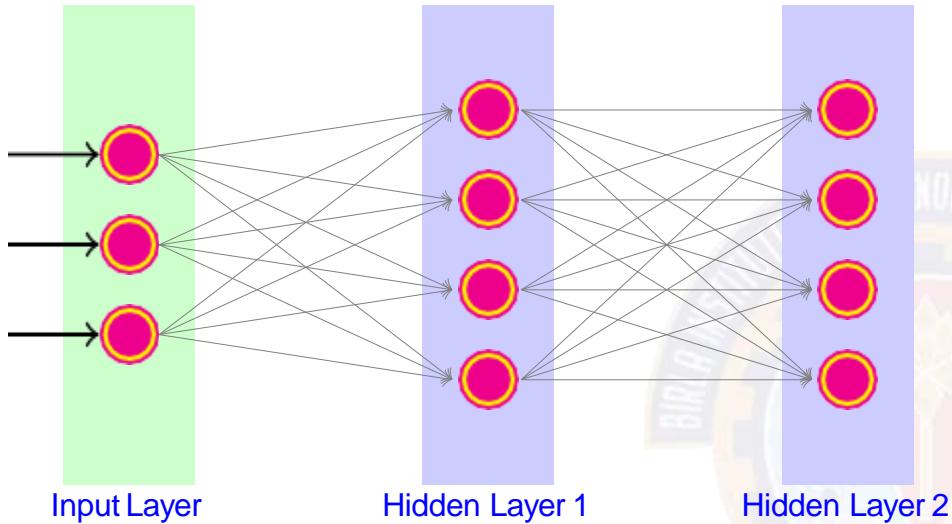
$x_1$	$X_2$	$\bar{x}_2$	$(x_1) \text{AND} (\bar{x}_2)$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0



This arrangement is mostly avoided, as training is more challenging

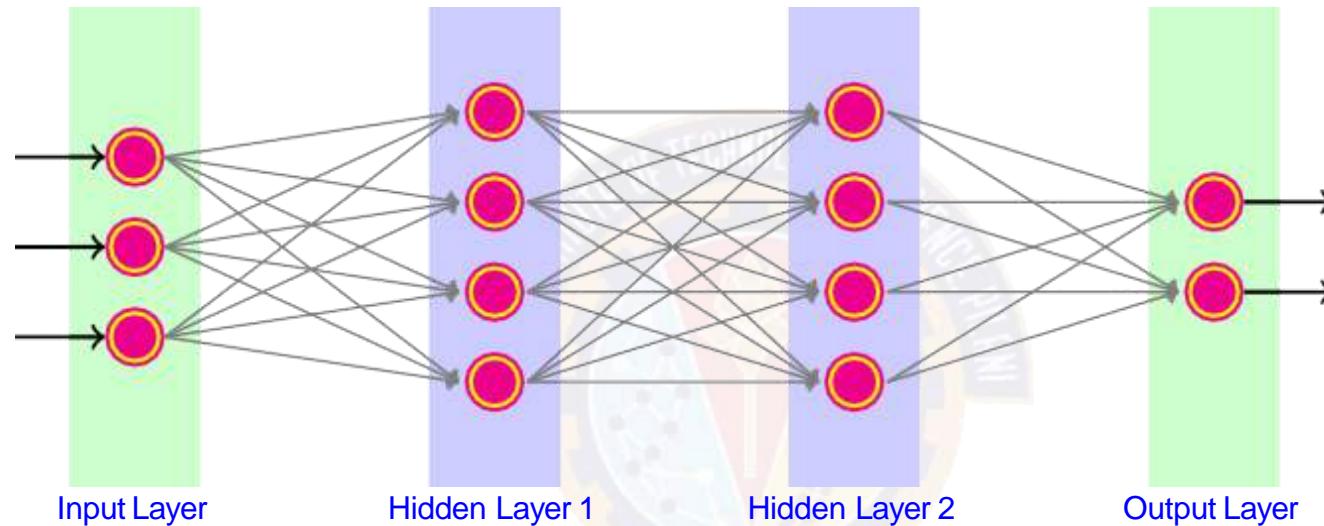
# Neural Network

When neurons are interconnected in layers



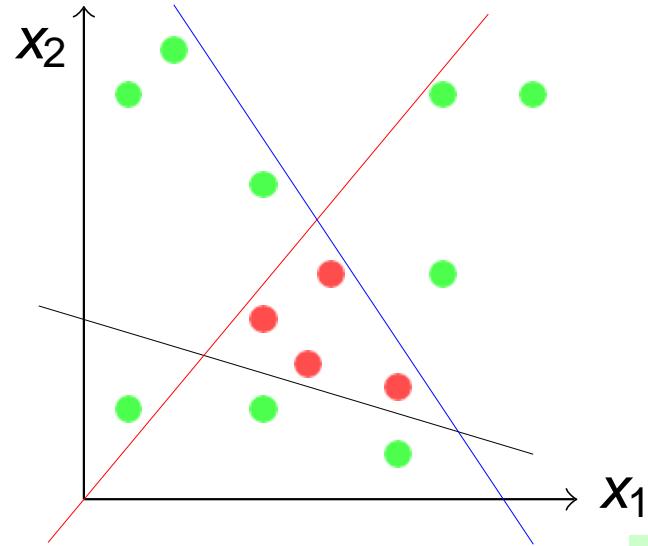
# Neural Network

When neurons are interconnected in layers



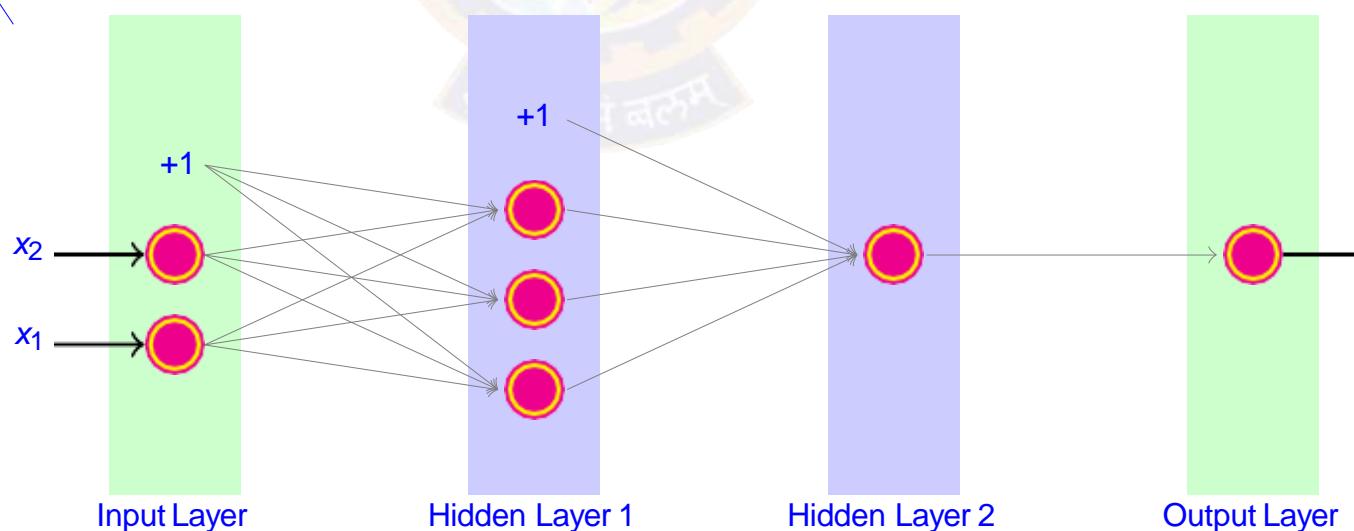
- Number of layers may differ
- Nodes in each intermediate layers may also differ
- Multiple output neurons are used for different class
- **Two levels deep** NN can represent any boolean function

# More Example: Design NN for the following data



Whether it is green?

Red-line	Blue-line	Black-line	Color
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



# Neural Network Applications

- NN is appropriate for problems with the following characteristics:
  - Instances are provided by many attribute-value pairs (more data)
  - The target function output may be discrete-valued, real-valued, or a vector of several real or discrete valued attributes
  - The training examples may contain errors
  - Long training times are acceptable
  - Fast evaluation of the target function may be required
  - The ability of humans to understand the learned target function is not important



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# Perceptron Training (Delta Rule)

# Perceptron Training (delta rule)

- Delta rule converges to a best-fit approximation of the target
- Uses gradient descent
- Consider un-thresholded perceptron, i.e., output  $o(\vec{x}) = \vec{w} \cdot \vec{x}$
- Training error is defined as

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- Gradient would specify direction of steepest increase

$$\nabla E(\vec{w}) = \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- Weights can be learned as  $w_i = w_i - \eta \frac{\partial E}{\partial w_i}$

- It can be seen that  $\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$

# Perceptron Training (delta rule)

---

## Algorithm: Gradient Descent ( $D, \eta$ )

---

```
1 Initialize  $w_i$  with random weights
2 repeat
3   For each  $w_i$ , initialize  $\delta w_i = 0$ 
4   for each training example  $d \in D$  do
5     Compute output  $o$  using model for  $d$  whose target is  $t$ 
6     For each  $w_i$ , update  $\delta w_i = \delta w_i + \eta(t-o)x_i$ 
7   For each  $w_i$ , set  $w_i = w_i + \delta w_i$ 
8 until termination condition is met,
9 return  $w$ 
```

---

# Perceptron Training (delta rule)

---

## Algorithm: Gradient Descent ( $D, \eta$ )

---

```
1 Initialize  $w_i$  with random weights
2 repeat
3   For each  $w_i$ , initialize  $\delta w_i = 0$ 
4   for each training example  $d \in D$  do
5     Compute output  $o$  using model for  $d$  whose target is  $t$ 
6     For each  $w_i$ , update  $\delta w_i = \delta w_i + \eta(t-o)x_i$ 
7   For each  $w_i$ , set  $w_i = w_i + \delta w_i$ 
8 until termination condition is met,
9 return  $w$ 
```

---

- A date item  $d \in D$ , is supposed to be multidimensional  $d = (x_1, x_2, \dots, x_n, t)$
- Algorithm converges toward the minimum error hypothesis.

# Perceptron Training (delta rule)

---

## Algorithm: Gradient Descent ( $D, \eta$ )

---

```
1 Initialize  $w_i$  with random weights
2 repeat
3   For each  $w_i$ , initialize  $\delta w_i = 0$ 
4   for each training example  $d \in D$  do
5     Compute output  $o$  using model for  $d$  whose target is  $t$ 
6     For each  $w_i$ , update  $\delta w_i = \delta w_i + \eta(t-o)x_i$ 
7   For each  $w_i$ , set  $w_i = w_i + \delta w_i$ 
8 until termination condition is met,
9 return  $w$ 
```

---

- A date item  $d \in D$ , is supposed to be multidimensional  $d = (x_1, x_2, \dots, x_n, t)$
- Algorithm converges toward the minimum error hypothesis.
- Linear programming can also be an approach

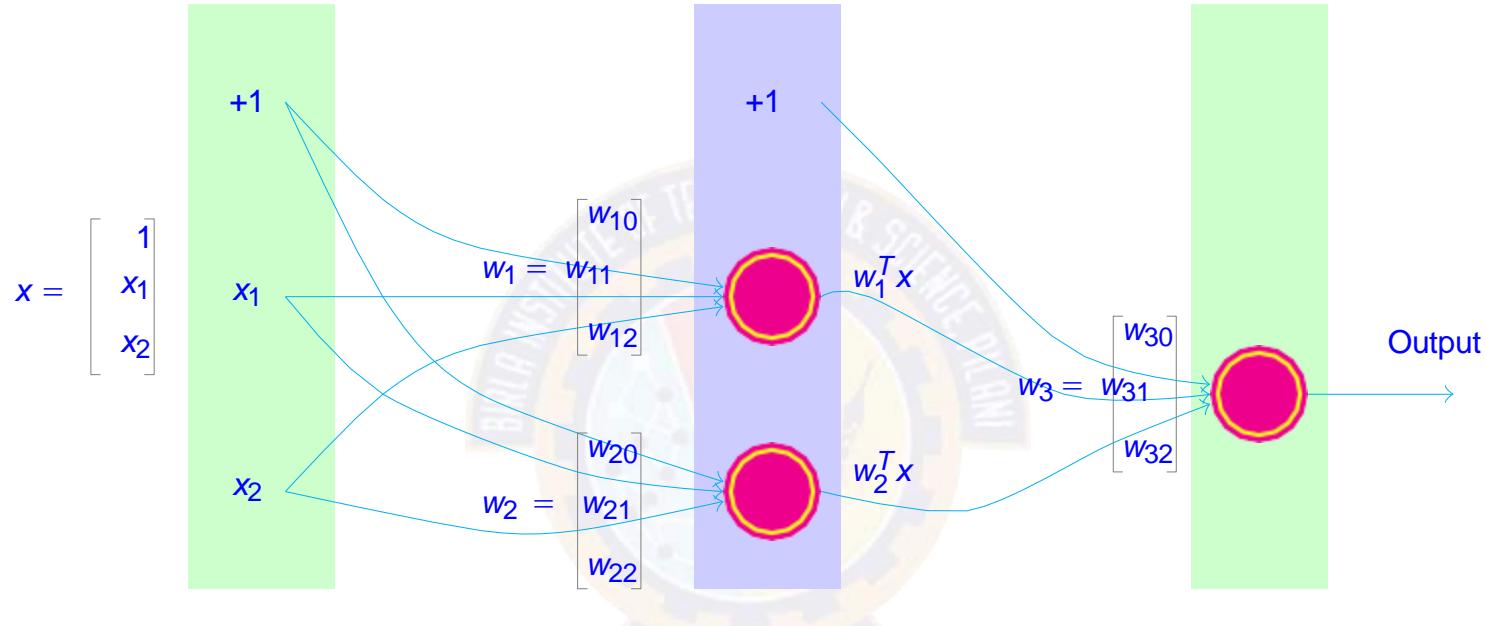


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Importance of Non-Linearity

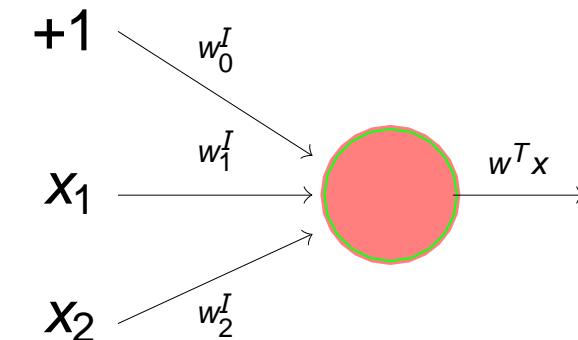
# Linear Activation is Not Much Interesting

NN with perceptrons have limited capability, even with many layers



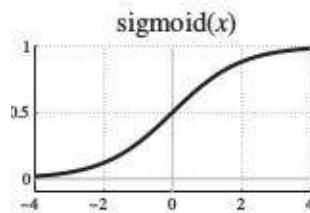
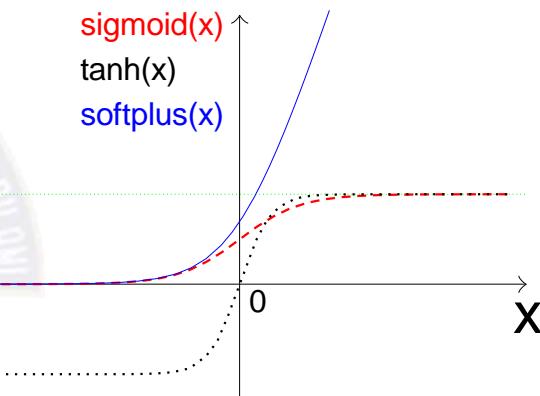
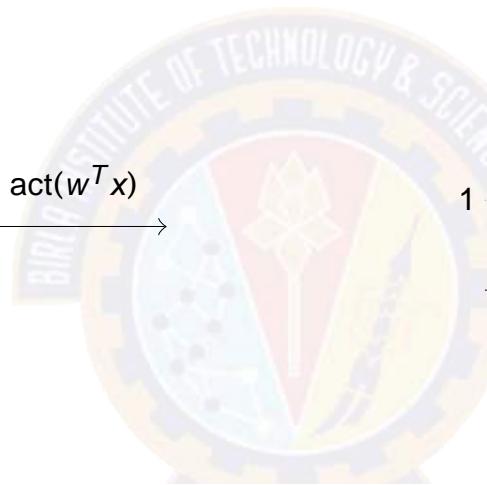
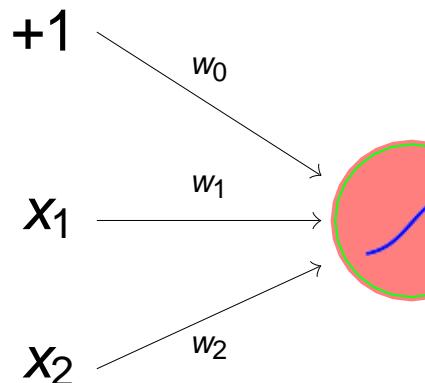
$$\begin{aligned} \text{Output} &= w_{30} \times 1 + w_{31} \times (w_1^T x) + w_{32} \times (w_2^T x) \\ &= w_{30} \times 1 + w_{31} \times [w_{10} \times 1 + w_{11} \times x_1 + w_{12} \times x_2] \\ &\quad + w_{32} \times [w_{20} \times 1 + w_{21} \times x_1 + w_{22} \times x_2] \\ &= (w_{30} + w_{31}w_{10} + w_{32}w_{20}) + (w_{31}w_{11} + w_{32}w_{21}) \times x_1 \\ &\quad + (w_{31}w_{12} + w_{32}w_{22}) \times x_2 \\ &= w'_0 + w'_1 \times x_1 + w'_2 \times x_2 \end{aligned}$$

Expression of single perceptron

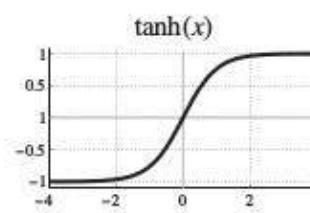


# Neuron

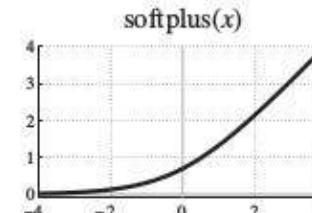
Neuron uses nonlinear **activation functions** (**sigmoid, tanh, ReLU, softplus etc.**) at the place of thresholding



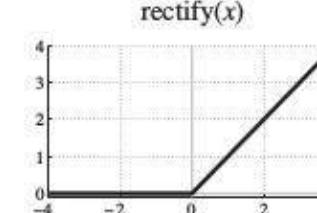
$$h(x) = \frac{1}{1 + \exp(-x)}$$



$$h(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



$$h(x) = \log(1 + \exp(x))$$



$$h(x) = \max(0, x)$$



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Backpropagation

# Multilayer Networks and Backpropagation

- Single perceptron can only express linear decision surface
- We need units whose output is a nonlinear function of its inputs AND is also differentiable (Use Neuron not Perceptron)

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x})$$

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

- **Backpropagation** algorithm learns the weights for a fixed set of units and interconnections
- It employs **gradient descent** to minimize the error between the network output values and the target values for these outputs
- Let Error function is redefined as

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

# Backpropagation (for 2 layers)

---

**Algorithm:** Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

---

- 1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers
  - 2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$
  - 3 **repeat**
  - 4     **for each**  $\langle \vec{x}, \vec{t} \rangle \in D$  **do**
  - 5          $o_u = \text{get output from network } \forall \text{unit } u$
  - 6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**
  - 7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**
  - 8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$
  - 9 **until** converge;
- 

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$

# Backpropagation (for 2 layers)

---

**Algorithm:** Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

---

- 1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers
- 2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$
- 3 **repeat**
- 4     **for each**  $\langle \vec{x}, \vec{t} \rangle \in D$  **do**
- 5          $o_u = \text{get output from network } \forall \text{unit } u$
- 6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**
- 7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**
- 8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$
- 9 **until** converge;

---

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$
- For a single training example  $E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$

# Backpropagation (for 2 layers)

---

**Algorithm:** Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

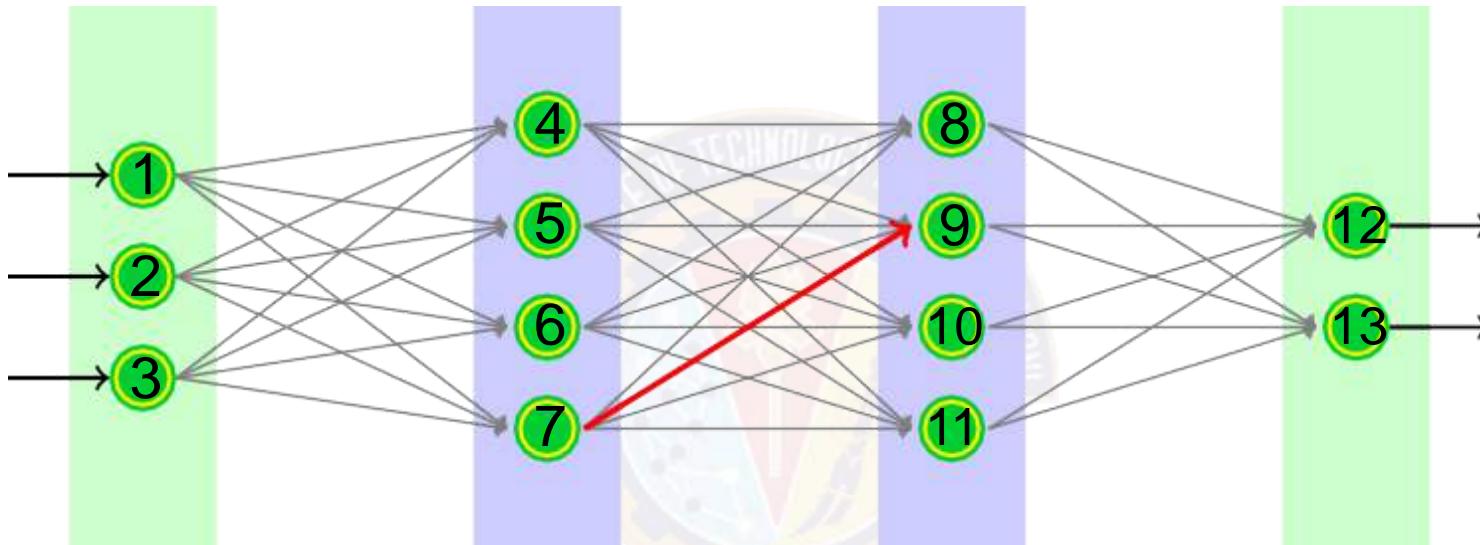
---

- 1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers
- 2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$
- 3 **repeat**
- 4     **for each**  $\langle \vec{x}, \vec{t} \rangle \in D$  **do**
- 5          $o_u = \text{get output from network } \forall \text{unit } u$
- 6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**
- 7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**
- 8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$
- 9 **until** converge;

---

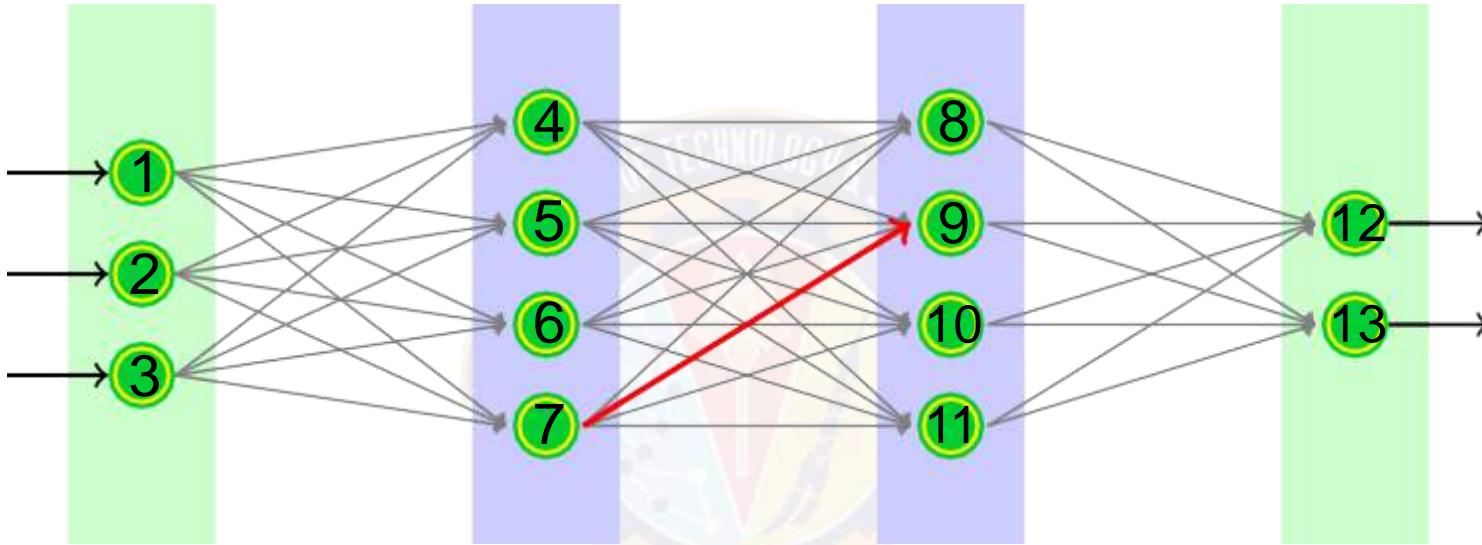
- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$
- For a single training example  $E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$
- Weight  $w_{ji}$  is updated by adding  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$

# Conventions Over The Network



$x_{ji}$     $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

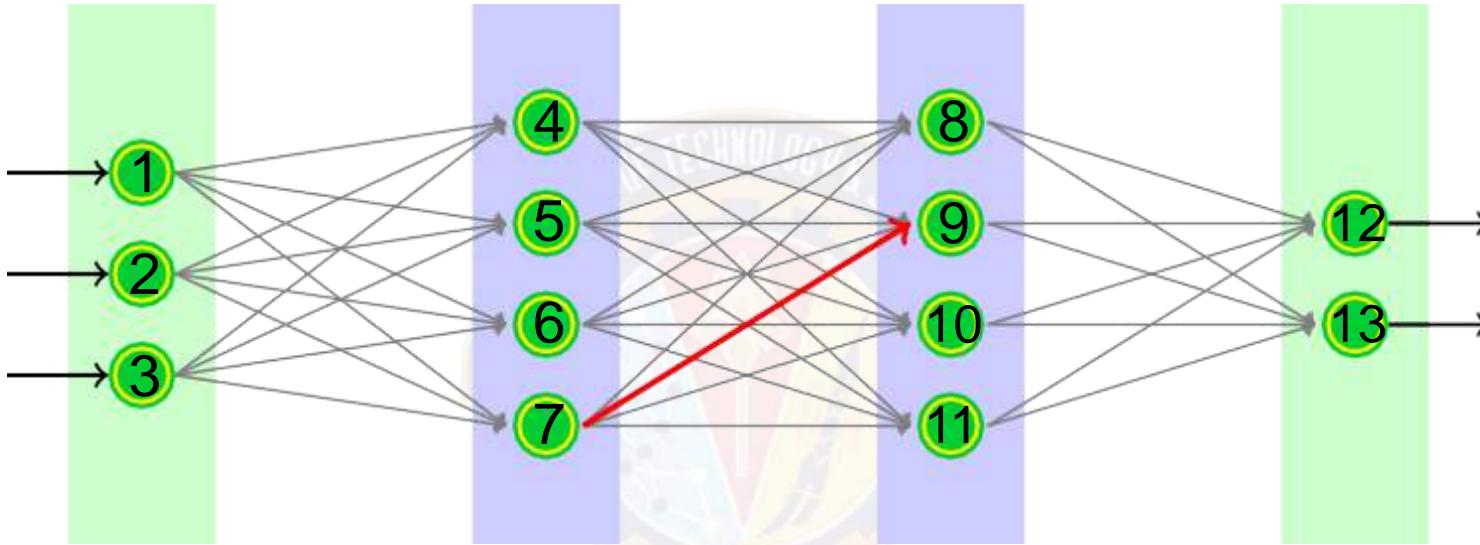
# Conventions Over The Network



$x_{ji}$     $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$    weight associated with  $i$  th input to unit  $j$

# Conventions Over The Network

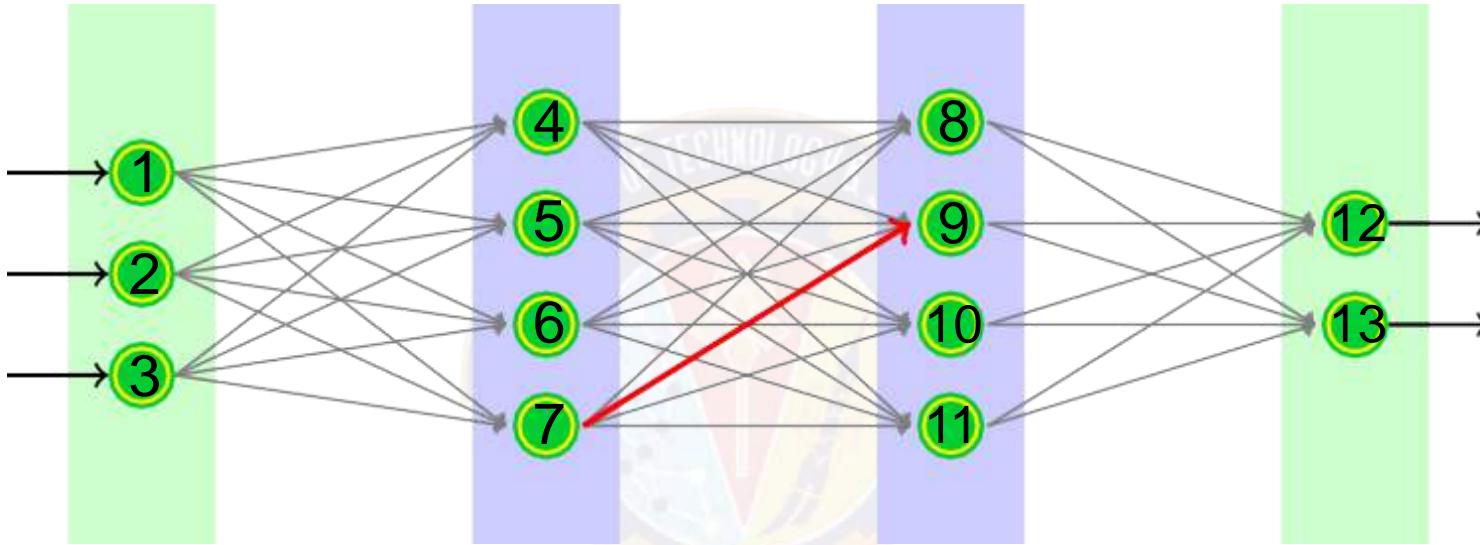


$x_{ji}$   $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$  weight associated with  $i$  th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

# Conventions Over The Network



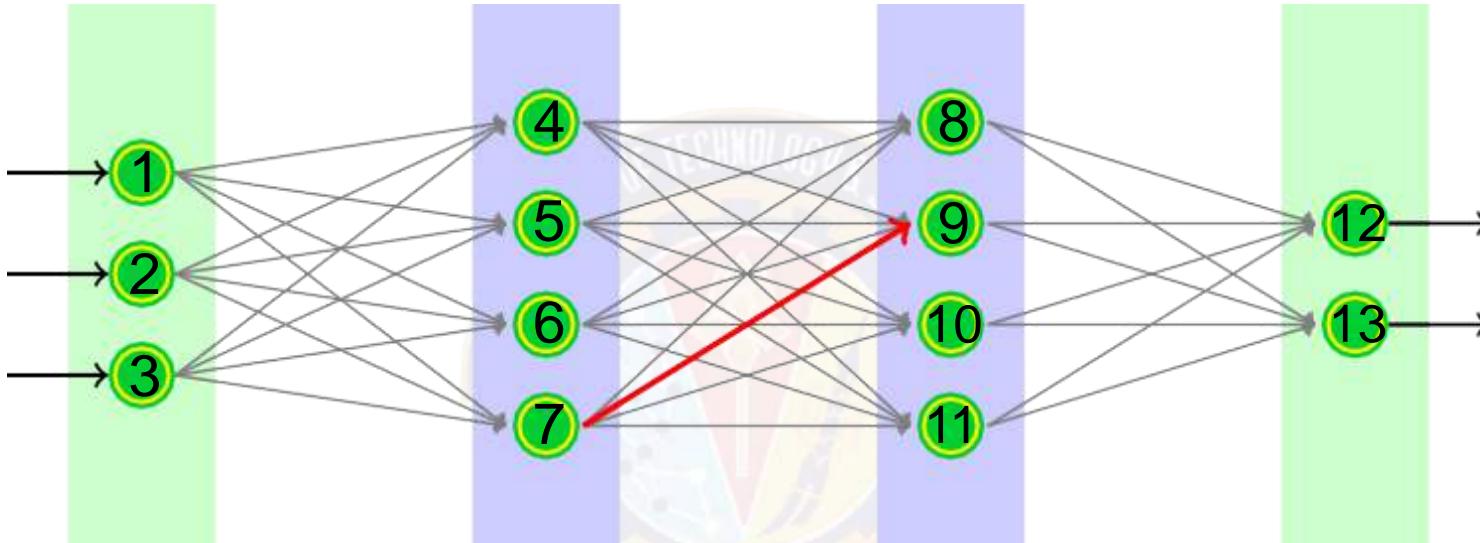
$x_{ji}$   $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$  weight associated with  $i$  th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

# Conventions Over The Network



$x_{ji}$   $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

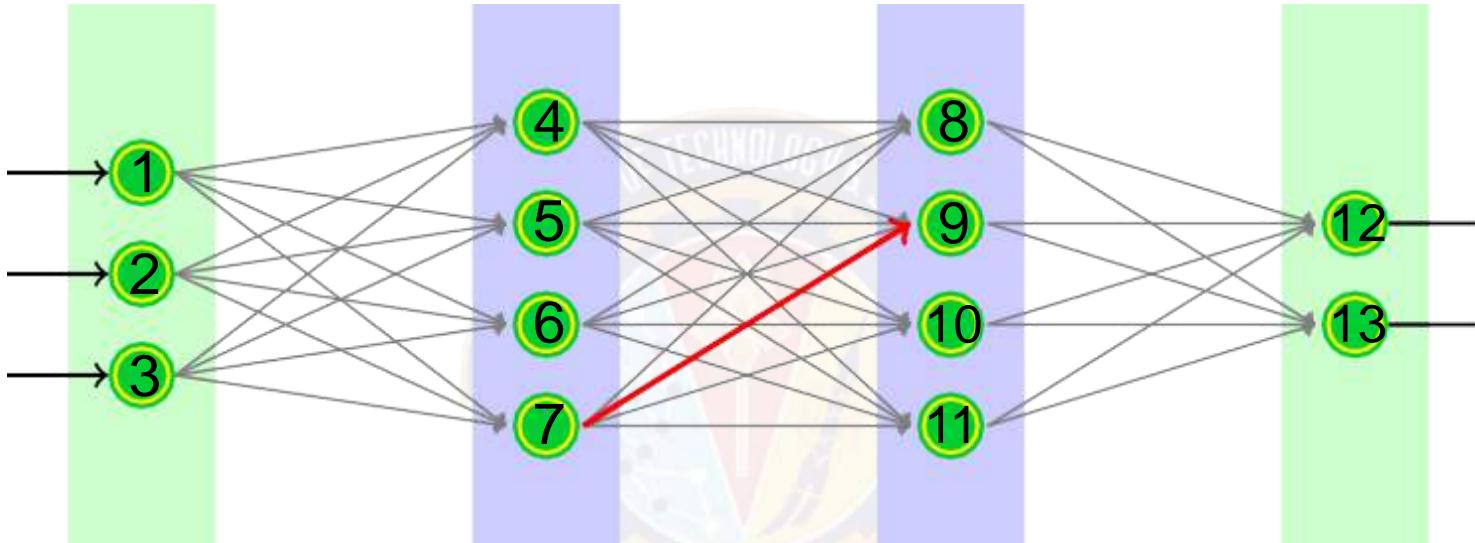
$w_{ji}$  weight associated with  $i$  th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

$t_j$  target output for unit  $j$

# Conventions Over The Network



$x_{ji}$   $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$  weight associated with  $i$  th input to unit  $j$

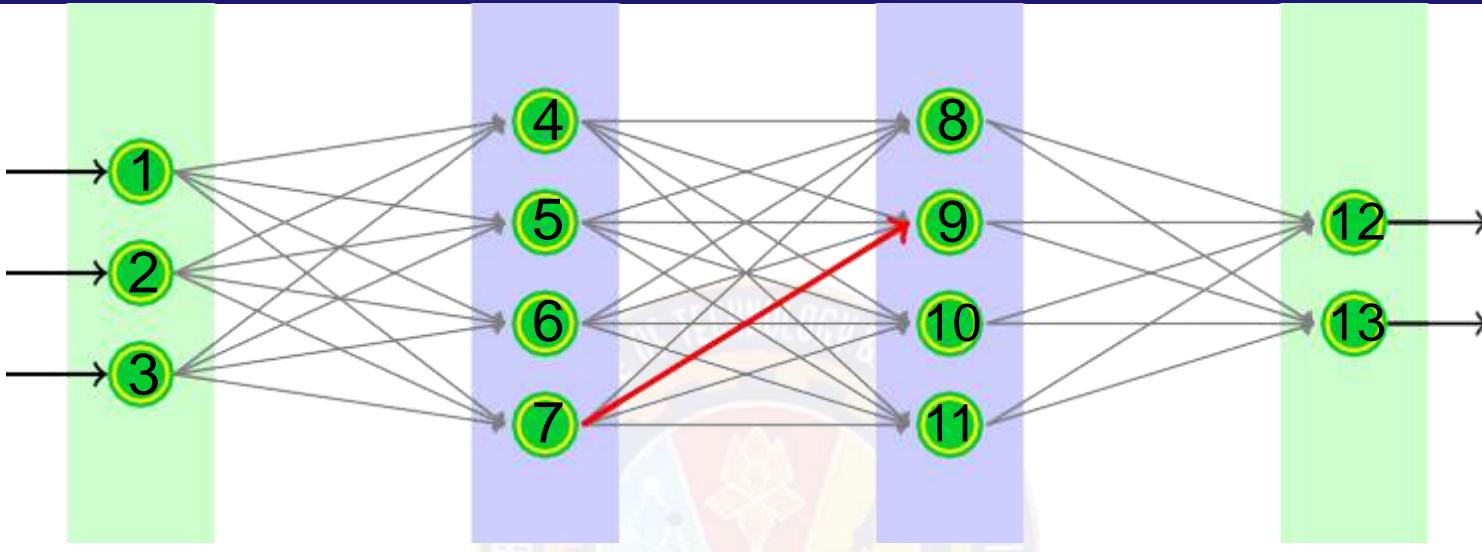
$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

$t_j$  target output for unit  $j$

$outputs$  set of units in final layer ( $\{12, 13\}$  in our case)

# Conventions Over The Network



$x_{ji}$   $i$ th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$  weight associated with  $i$ th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

$t_j$  target output for unit  $j$

*outputs* set of units in final layer ( $\{12, 13\}$  in our case)

*Downstream( $j$ )* units whose immediate input is the output of unit  $j$

We are interested in  $\frac{\partial E_d}{\partial w_{ji}}$  it is  $\frac{\partial E_d}{\partial net_j} \times \frac{\partial net_j}{\partial w_{ji}}$  and therefore,  $\frac{\partial E_d}{\partial net_j} \times x_{ji}$

# Value of $\frac{\partial E_d}{\partial net_j}$ for output units

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial net_j}$$



# Value of $\frac{\partial E_d}{\partial net_j}$ for output units

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$



# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2$$

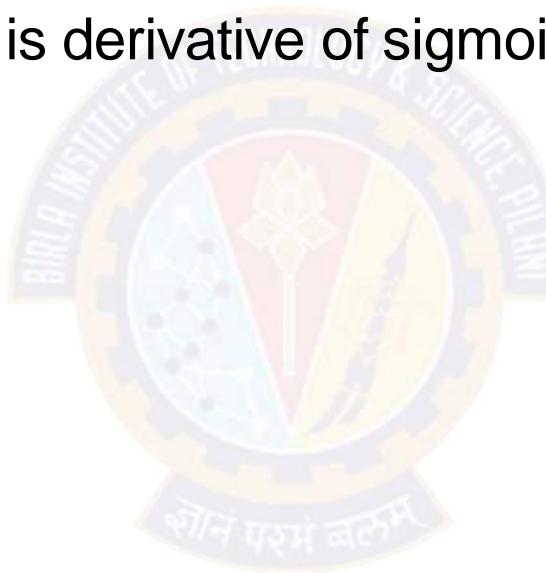


# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid



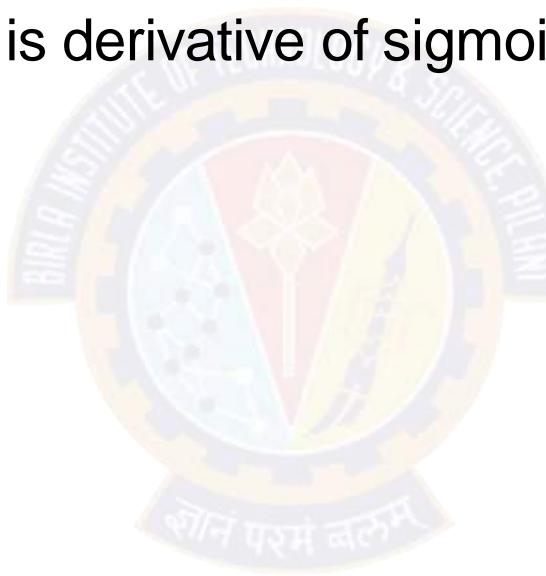
# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\frac{d}{dx} \sigma(x) = \frac{d}{dx} \frac{1}{1 + e^{-x}}$$



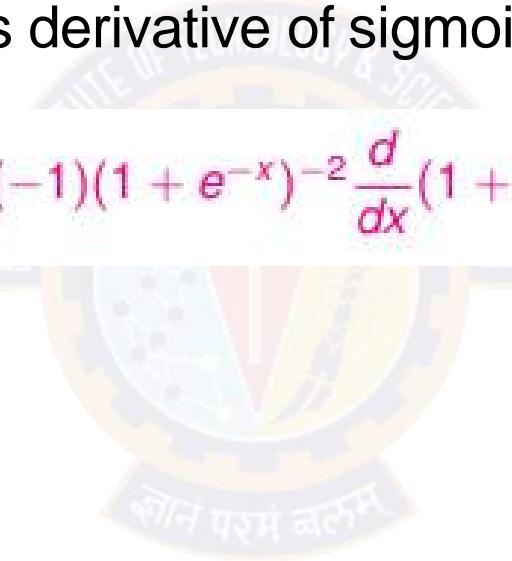
# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\frac{d}{dx} \sigma(x) = \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x})$$



# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x})\end{aligned}$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2}(0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}}\end{aligned}$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j))$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

Therefore,  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \text{net}_j} \times x_{ji}$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

Therefore,  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \text{net}_j} \times x_{ji} = \boxed{\eta(t_j - o_j)o_j(1 - o_j)x_{ji}}$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for hidden units

$$\begin{aligned}\frac{\partial E_d}{\partial \text{net}_j} &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E_d}{\partial \text{net}_k} \times \frac{\partial \text{net}_k}{\partial \text{net}_j} \\&= \sum_{k \in \text{Downstream}(j)} -\delta_k \times \frac{\partial \text{net}_k}{\partial \text{net}_j} \\&= \sum_{k \in \text{Downstream}(j)} -\delta_k \times \frac{\partial \text{net}_k}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j} \\&= \sum_{k \in \text{Downstream}(j)} -\delta_k \times w_{kj} \times o_j(1 - o_j)\end{aligned}$$

$\delta_j$  being  $-\frac{\partial E_d}{\partial \text{net}_j} = o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \delta_k \times w_{kj}$

Therefore,  $\Delta w_{ji} = \eta \delta_j x_{ji} = \boxed{\eta(o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \delta_k \times w_{kj}) x_{ji}}$

# Backpropagation (for 2 layers)

---

**Algorithm:** Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

---

- 1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers
  - 2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$
  - 3 **repeat**
  - 4     **for each**  $\langle \vec{x}, \vec{t} \rangle \in D$  **do**
  - 5          $o_u = \text{get output from network } \forall \text{unit } u$
  - 6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**
  - 7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**
  - 8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$
  - 9 **until** converge;
-

# Backpropagation

- **Adding Momentum:** weight update during  $n^{th}$  iteration depend partially on the update that occurred during the  $(n - 1)^{th}$  iteration

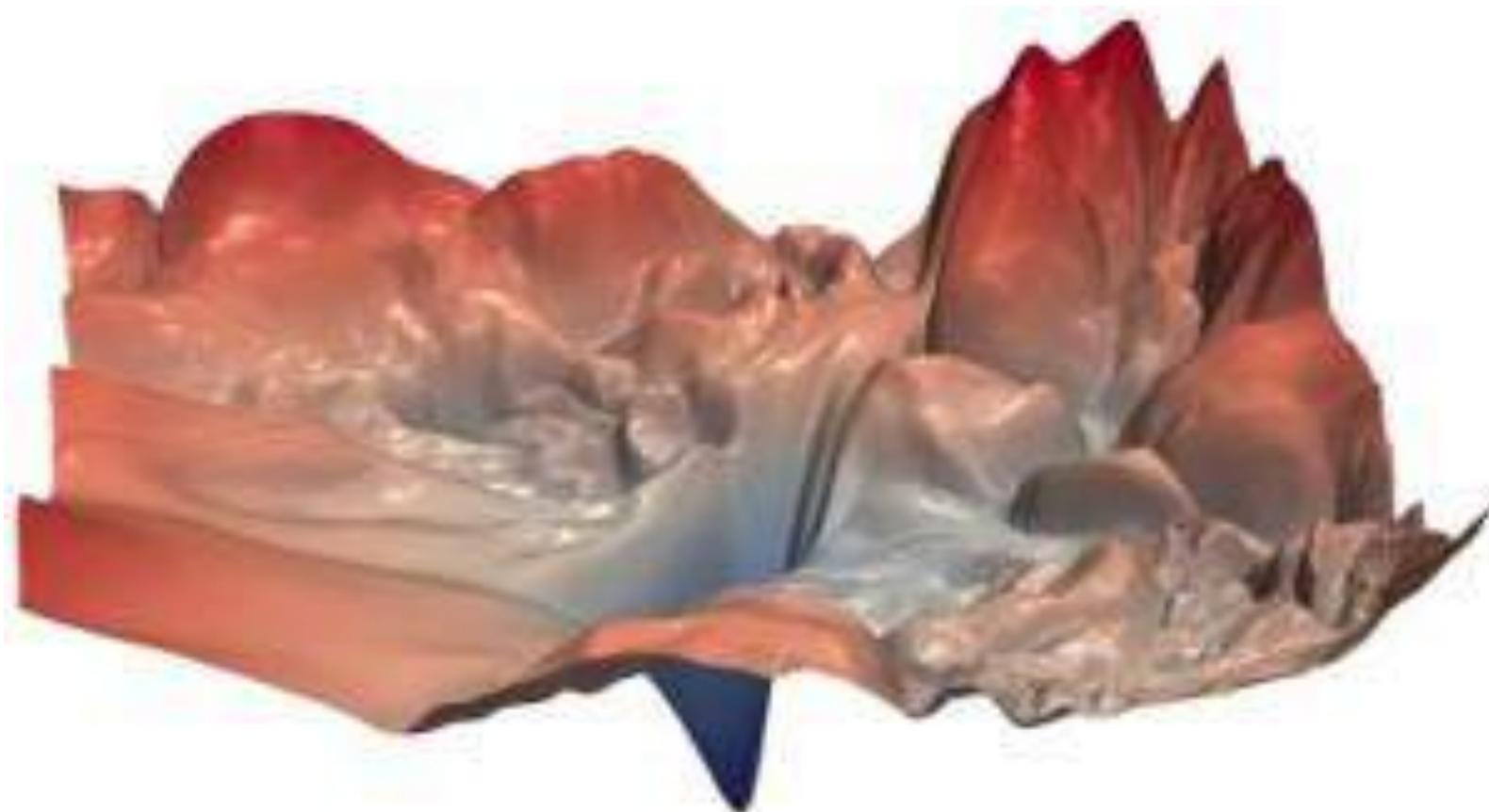
$$\Delta w_{jl}(n) = \eta \delta_j x_{jl} + \alpha \Delta w_{jl}(n-1)$$

- **Learning in arbitrary acyclic network:** for feed-forward networks of arbitrary depth,  $\delta_r$  value for a unit in hidden layer is determined as

$$\delta_r = o_r(1 - o_r) \sum_{s \in \text{Downstream}(r)} w_{sr} \times \delta_s$$

# Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error



# Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error
- No methods are known to predict with certainty when local minima will cause difficulties
- Suggested to use momentum, true gradient descent or multiple networks (initialized with different random weights)
- Any boolean function can precisely be represented by some network having only **two** layers of units (Cybenko 1989; Hornik et al. 1989)



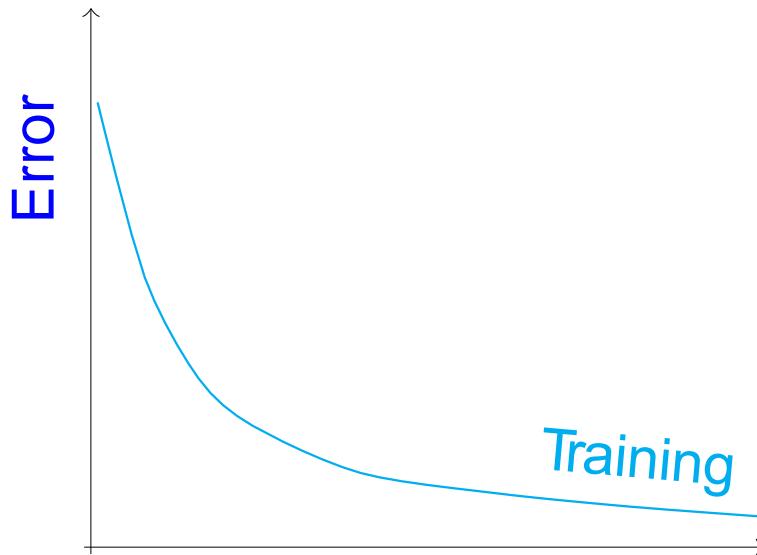
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Generalization, Overfitting and Stopping Criteria

Kamlesh Tiwari

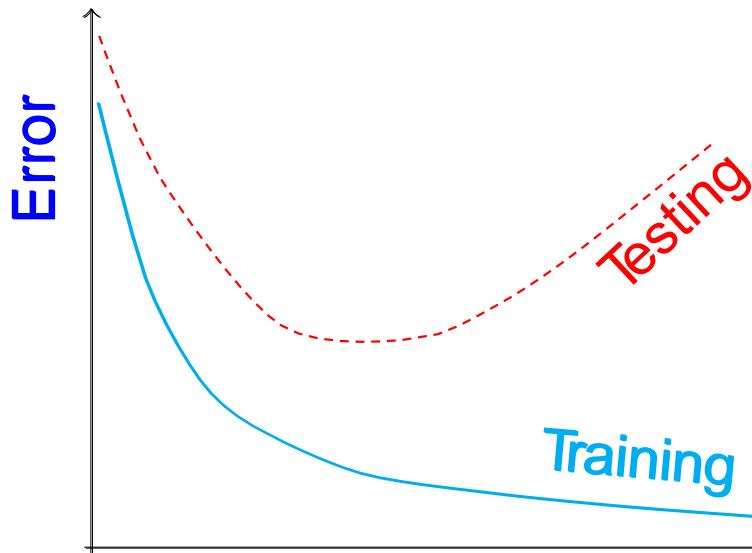
# Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



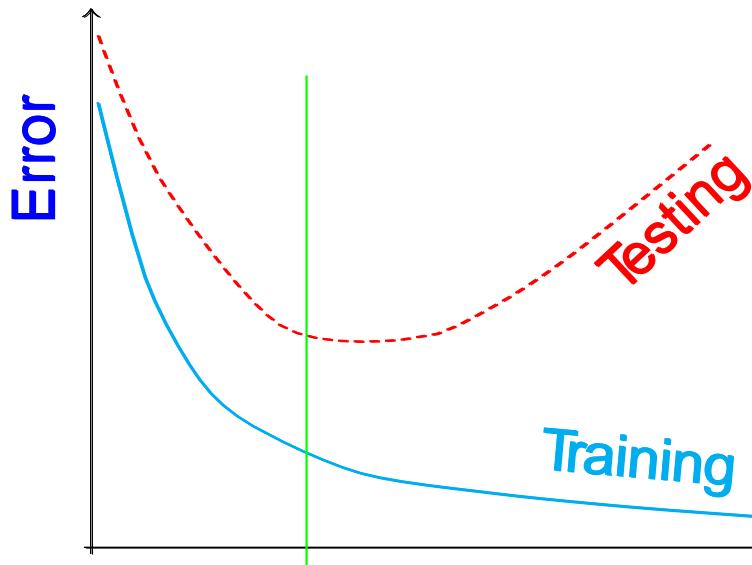
# Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



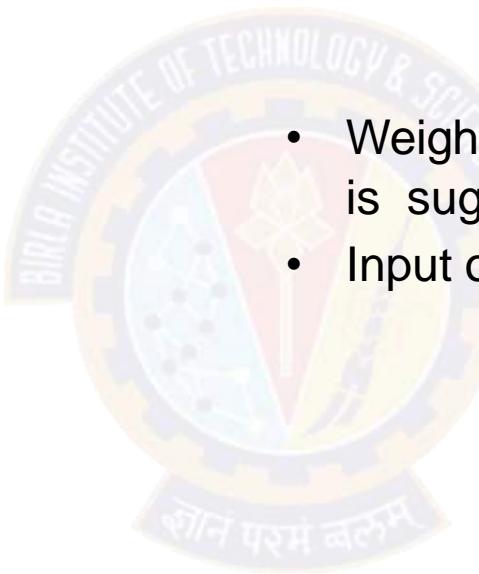
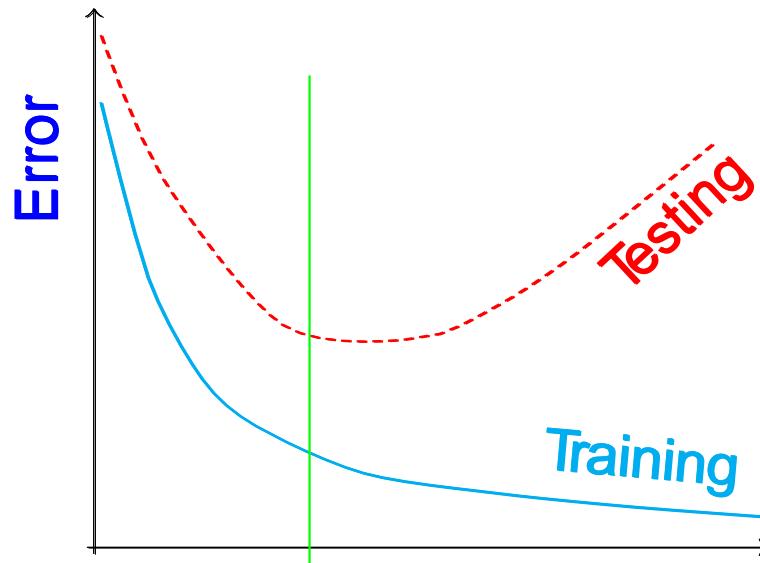
# Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



# Generalization, Overfitting, and Stopping Criterion

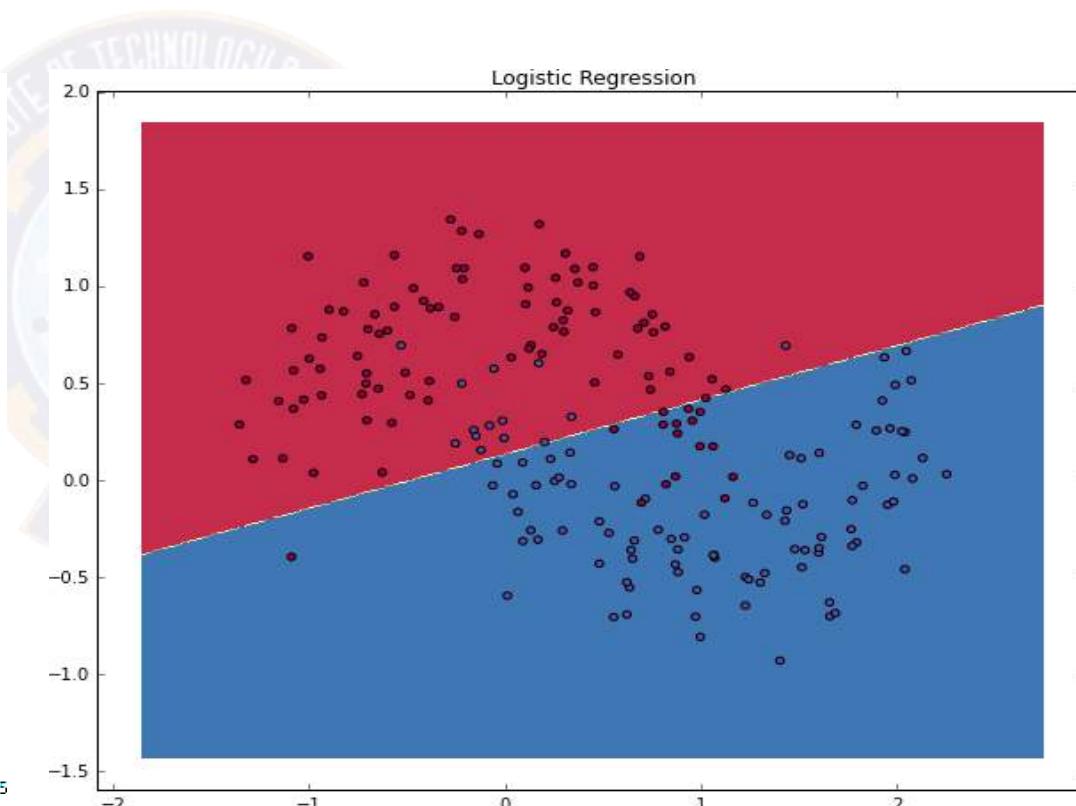
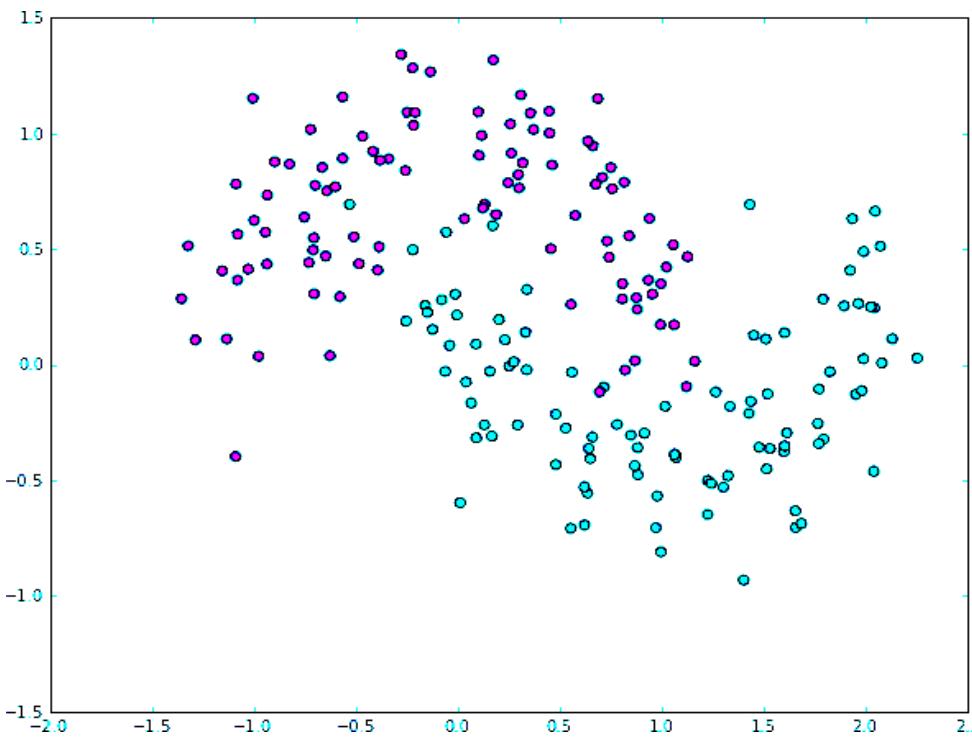
Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



- Weight decay or use of validation set ( $k$ -fold ?) is suggested
- Input or output encoding can be used

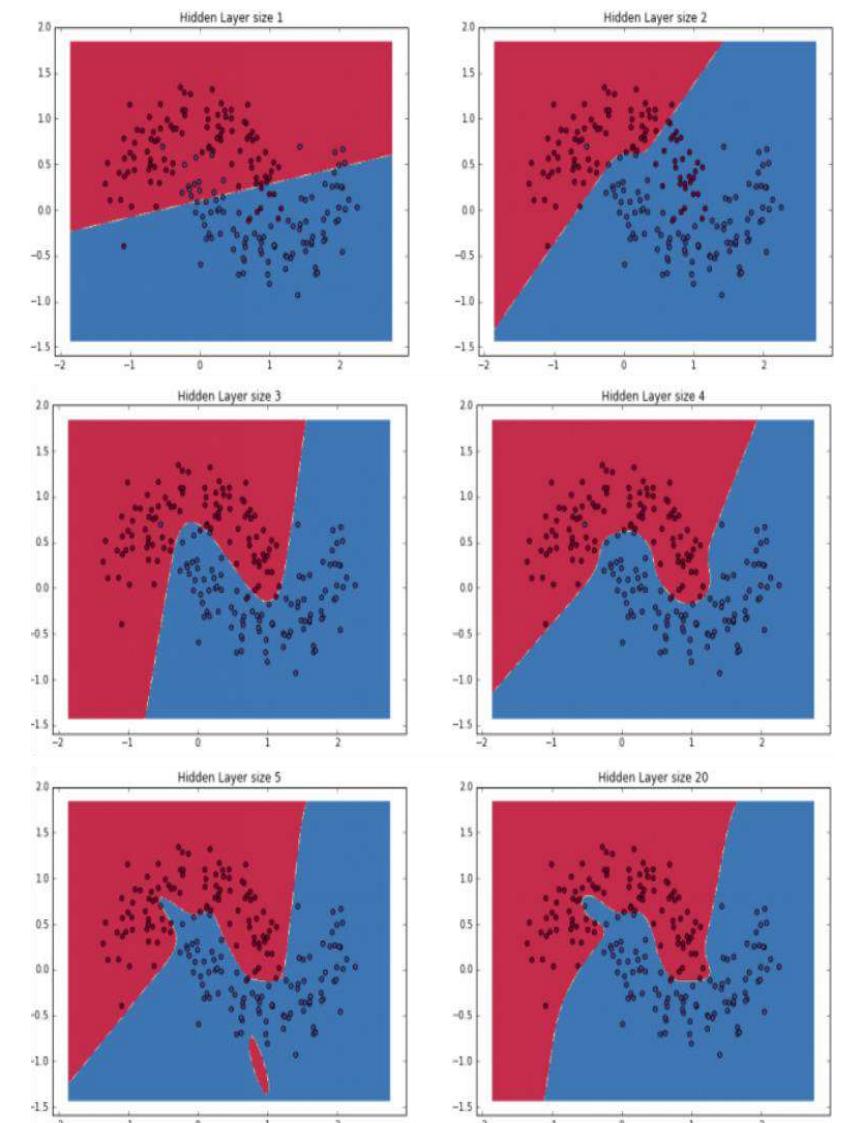
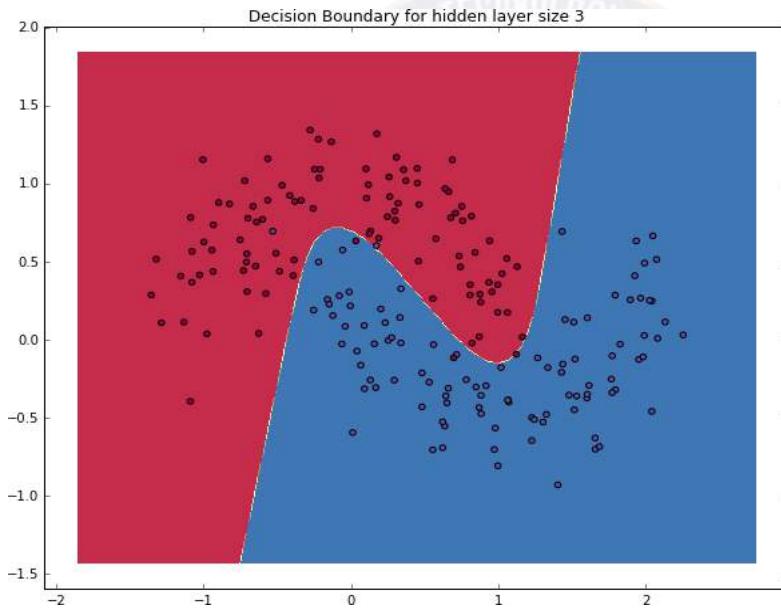
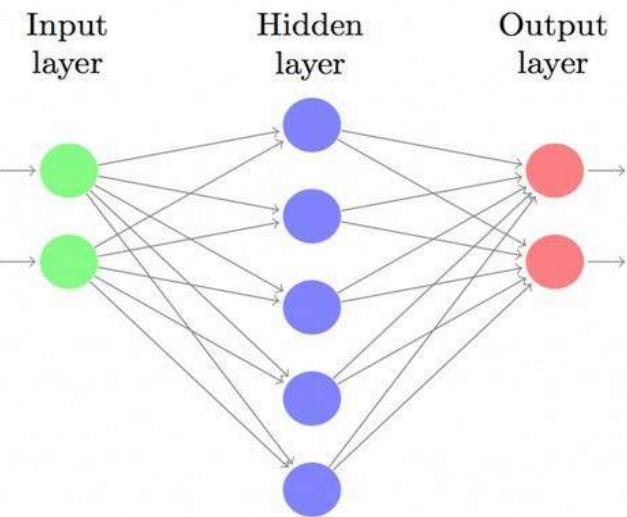
# Coding Example 1

Consider two class data  
Logistic Regression  
Neural Network  
Decision Boundary  
Bigger hidden layer



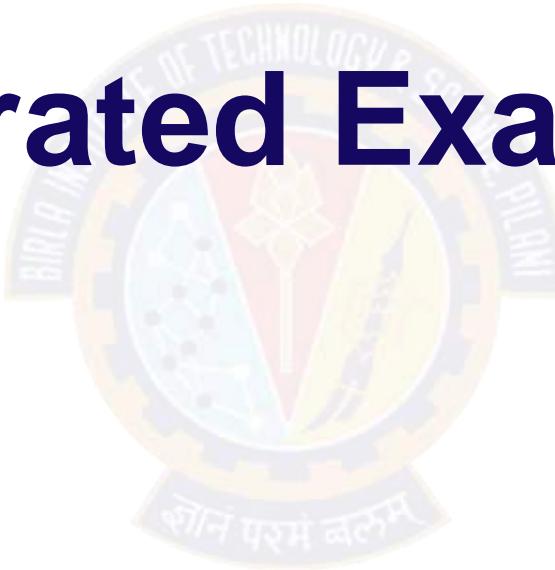
<sup>1</sup><https://github.com/dennybritz/nan-from-scratch/blob/master/nan-from-scratch.ipynb>

# Coding Example 1



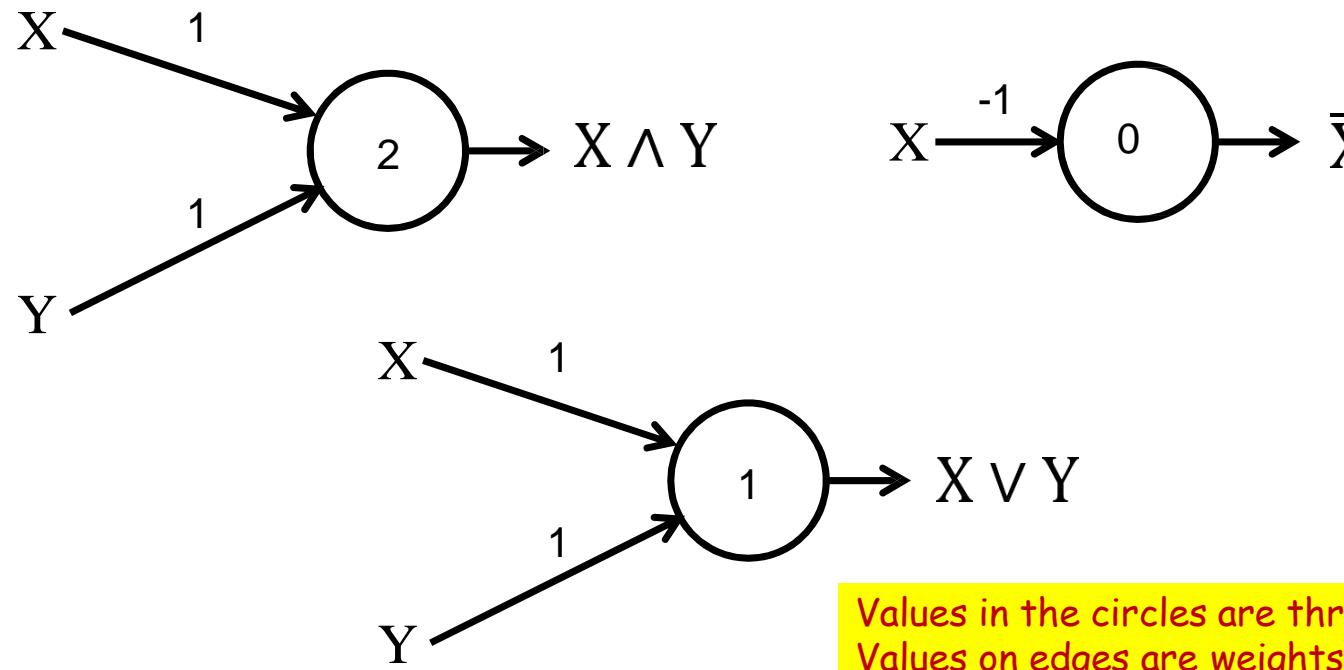
<sup>1</sup><https://github.com/dennybritz/nn-from-scratch/blob/master/nn-from-scratch.ipynb>

# Illustrated Examples



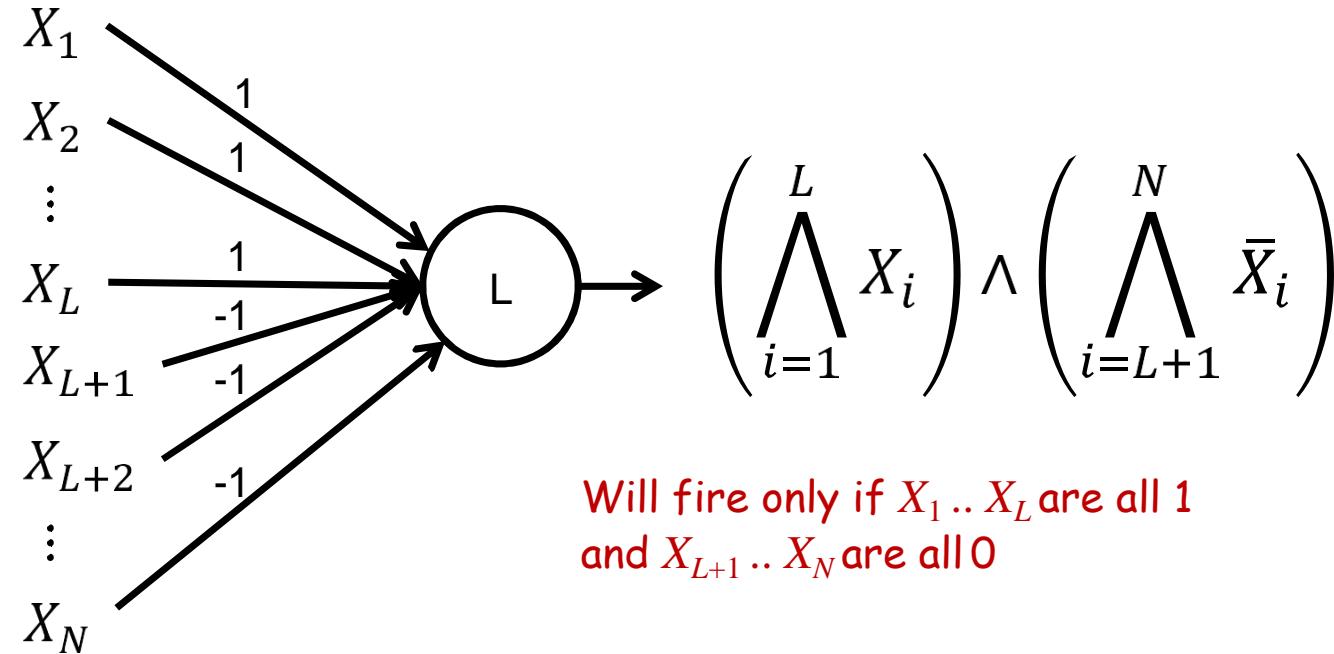
- Multi-layer Perceptrons as universal Boolean functions
- MLPs as universal classifiers

# The perceptron as a Boolean gate



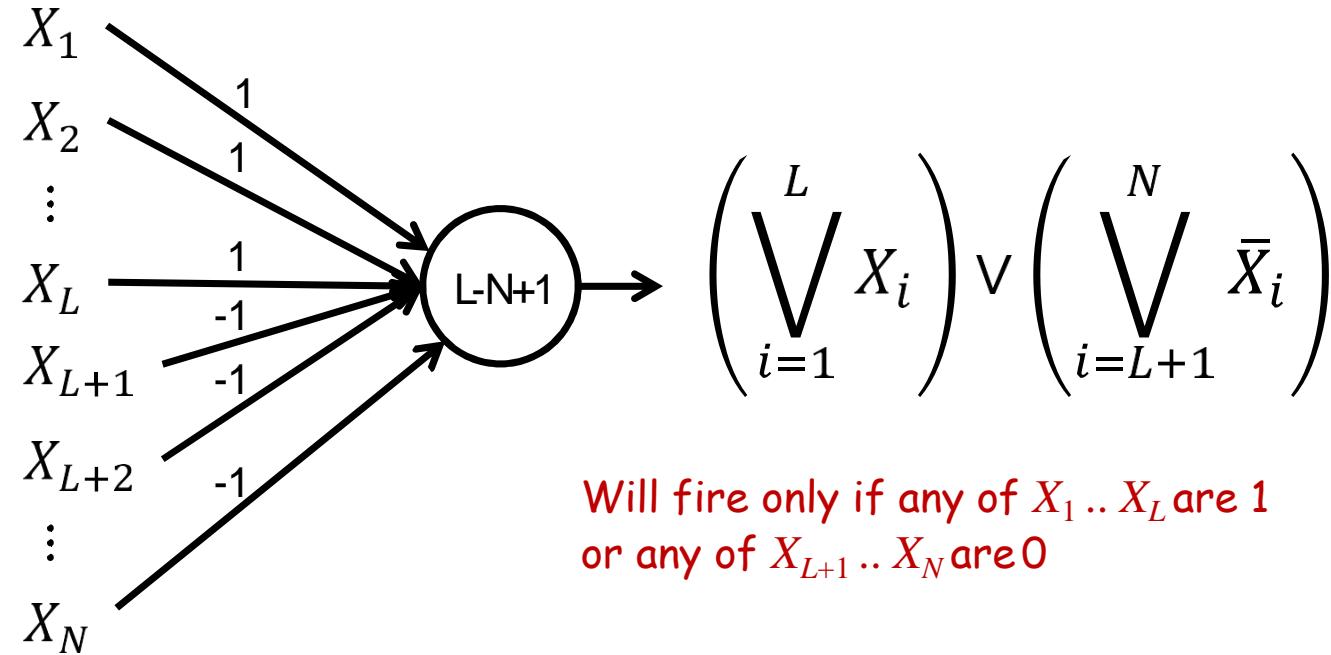
- A perceptron can model any simple binary Boolean gate
- Output = 1 if total input  $\geq$  threshold

# Perceptron as a Boolean gate



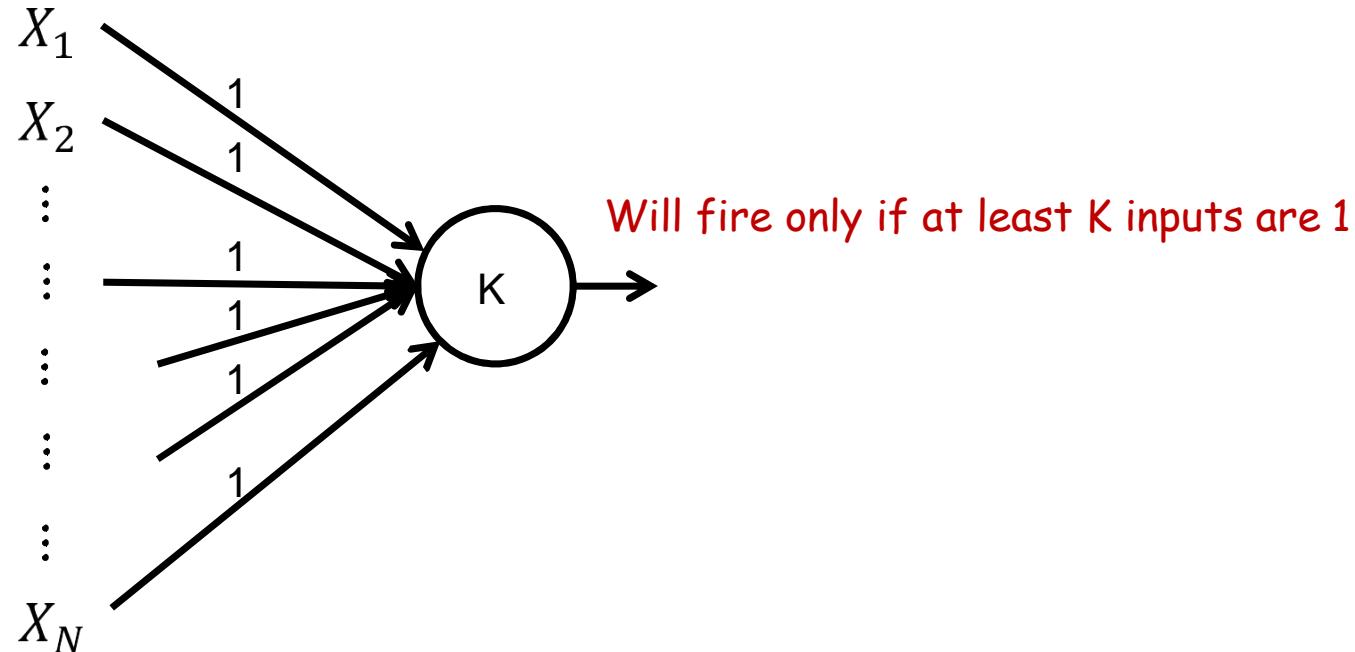
- The universal AND gate
  - AND any number of inputs
    - Any subset of who may be negated

# Perceptron as a Boolean gate



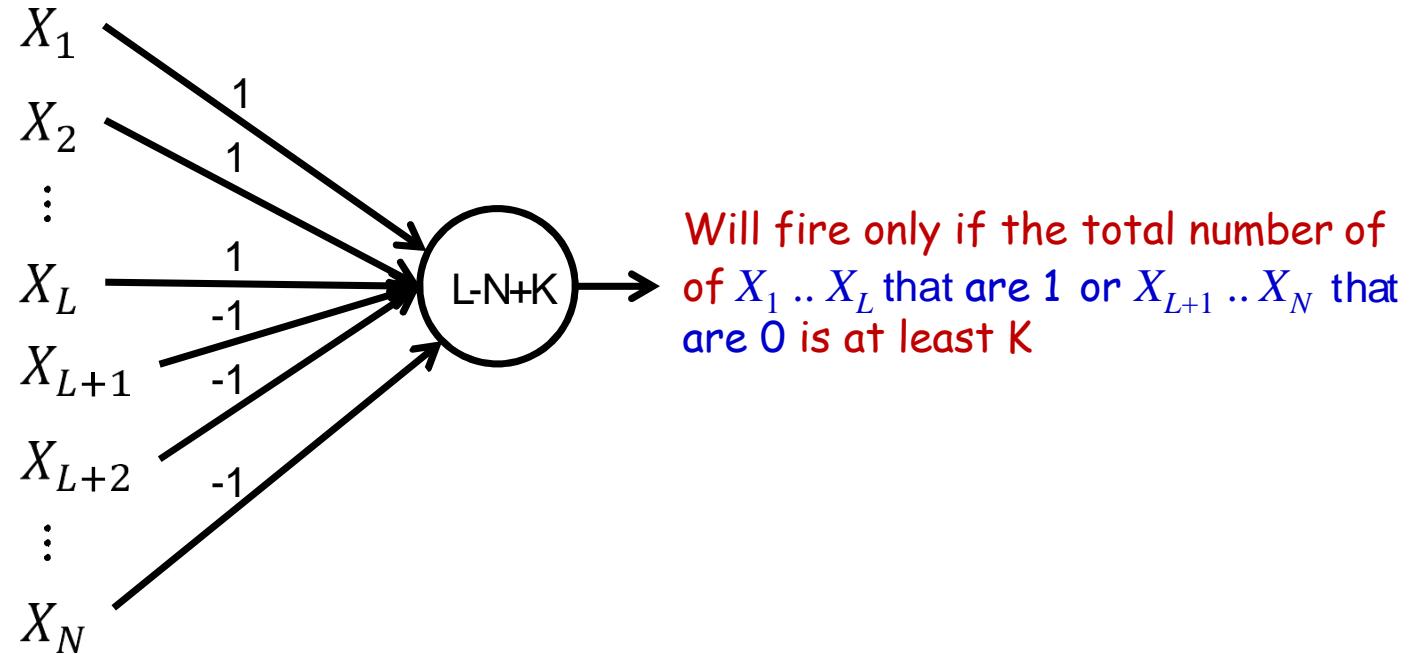
- The universal OR gate
  - OR any number of inputs
    - Any subset of who may be negated

# Perceptron as a Boolean Gate



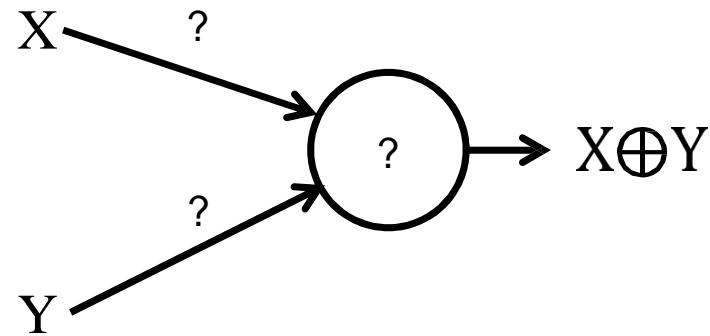
- Generalized *majority* gate
  - Fire if at least  $K$  inputs are of the desired polarity

# Perceptron as a Boolean Gate



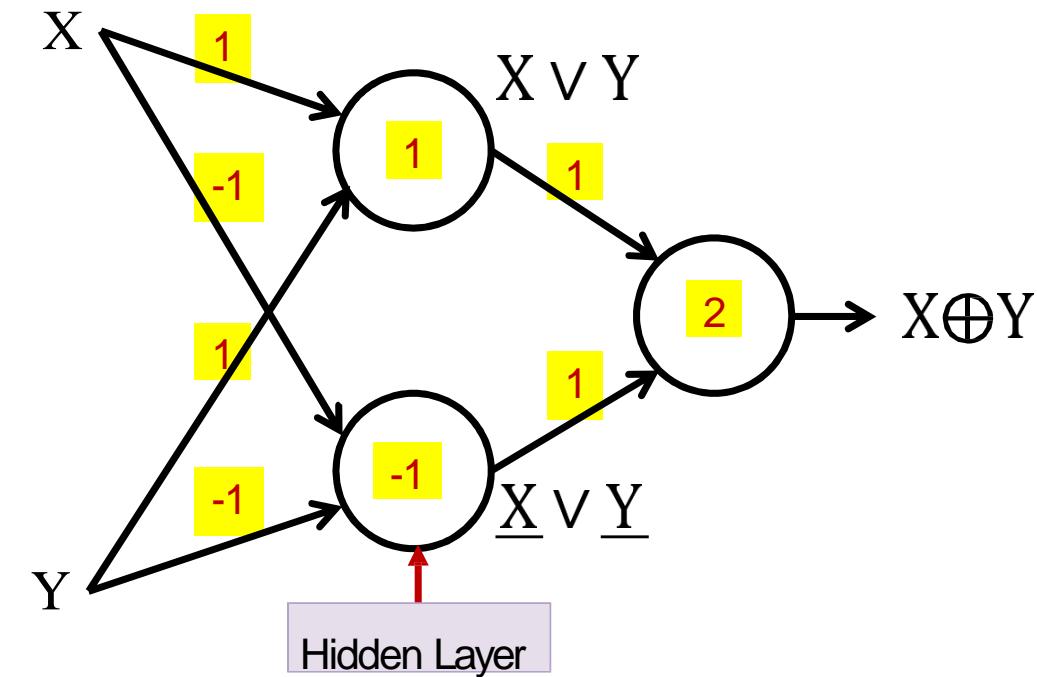
- Generalized *majority* gate
  - Fire if at least  $K$  inputs are of the desired polarity

# The perceptron is not enough



- Cannot compute an XOR

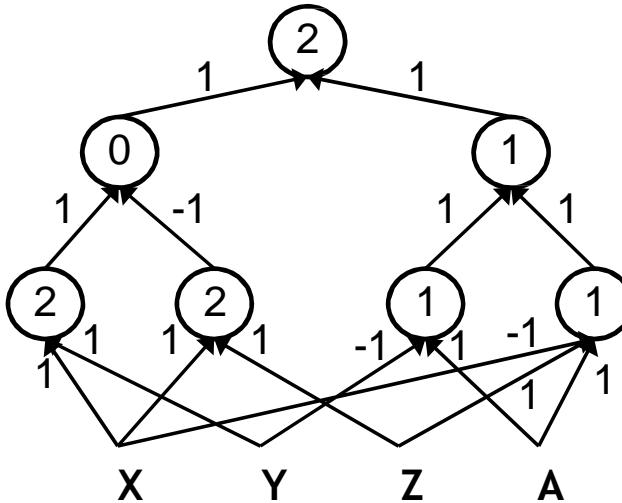
# Multi-layer perceptron



- MLPs can compute the XOR

# Multi-layer perceptron

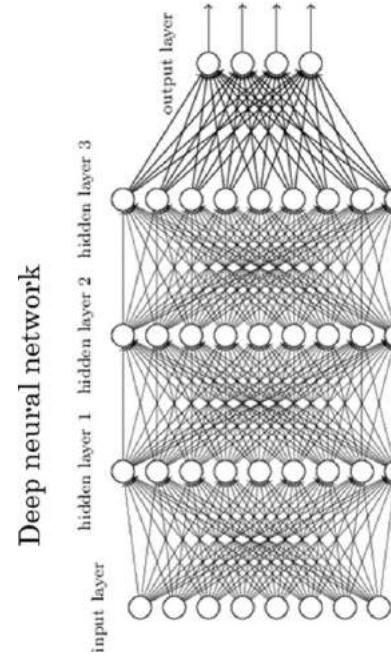
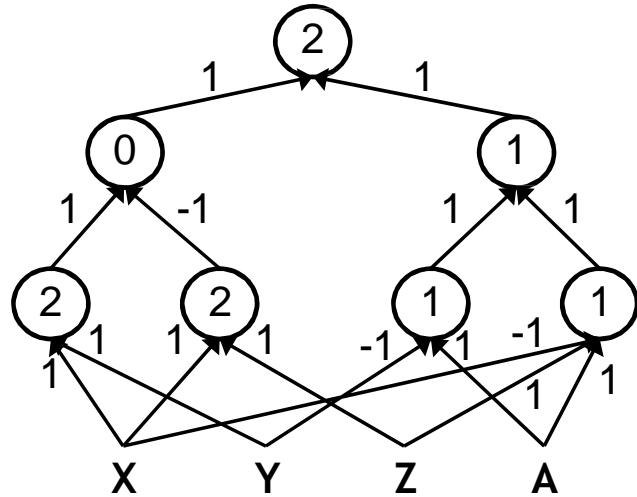
$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | (\bar{X} \& \bar{Z}))$$



- MLPs can compute more complex Boolean functions
- MLPs can compute *any* Boolean function
  - Since they can emulate individual gates
- **MLPs are *universal Boolean functions***

# MLP as Boolean Functions

$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | (\bar{X} \& \bar{Z}))$$



- MLPs are universal Boolean functions
  - Any function over any number of inputs and any number of outputs
- But how many “layers” will they need?

# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations  
for which output is 1

- *A Boolean function is just a truth table*

# How many layers for a Boolean MLP?

Truth Table

X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>	X <sub>4</sub>	X <sub>5</sub>	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations  
for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + \\ X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$

- Expressed in disjunctive normal form (DNF)

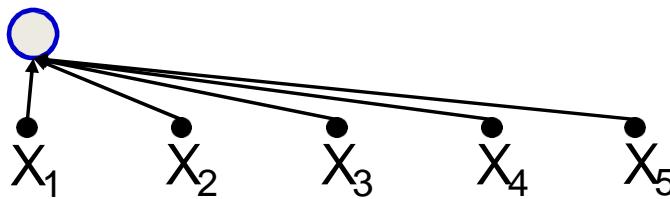
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 X_2 X_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

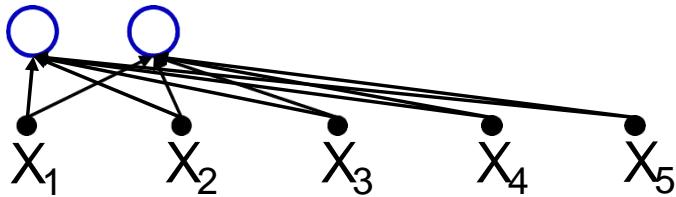
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations  
for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \textcircled{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + \\ X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

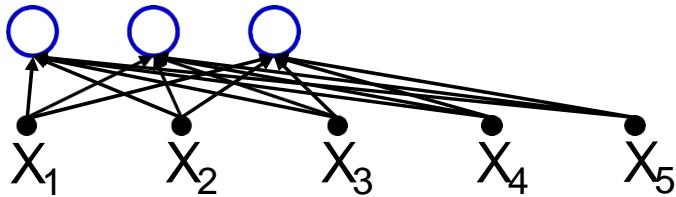
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \cancel{\bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5} + \\ X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

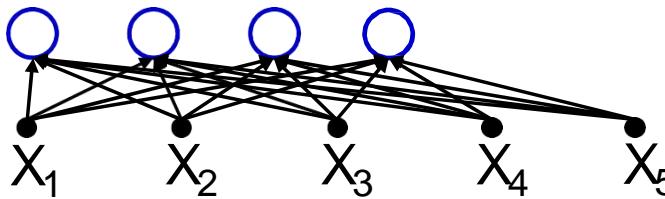
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations  
for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 +$$
  
$$\textcircled{X}_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + \blacksquare X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

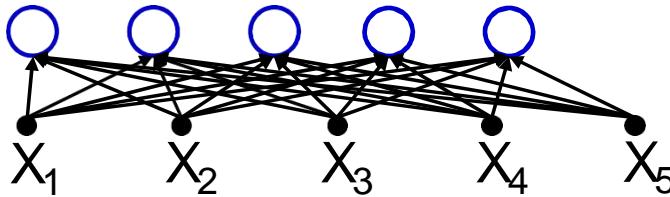
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations  
for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + \\ X_1 \bar{X}_2 X_3 \bar{X}_4 X_5 + \textcircled{X}_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

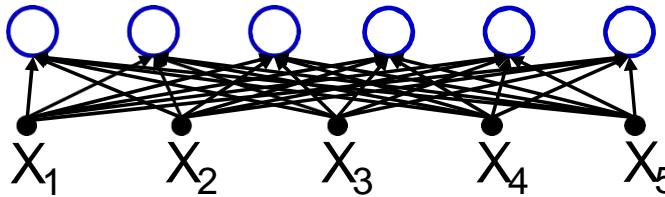
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + \\ X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + \textcircled{X}_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

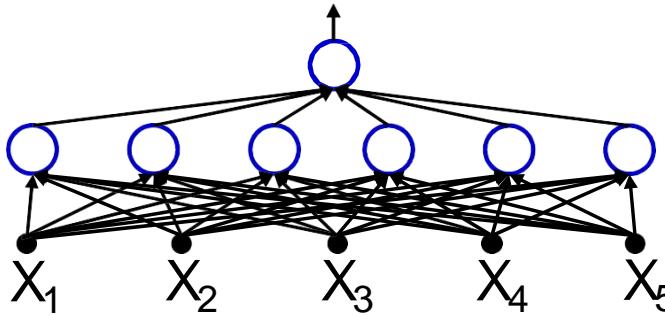
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

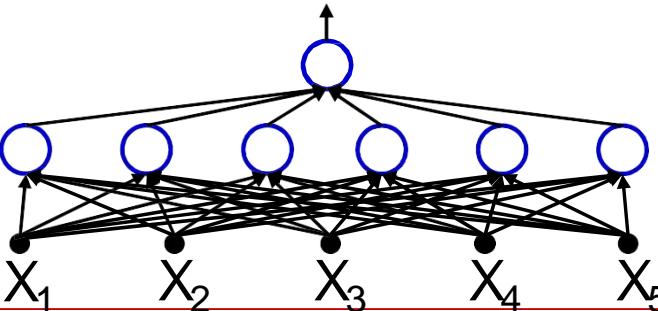
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



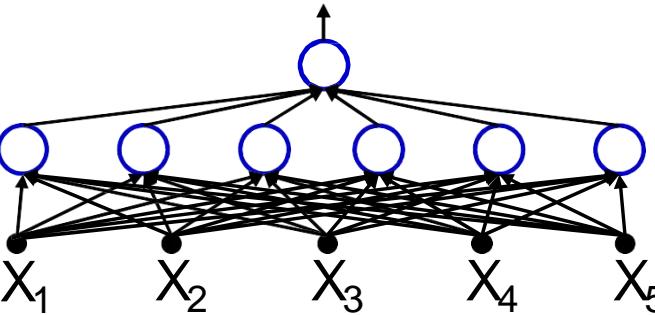
- Any truth table can be expressed in this manner!
- A one-hidden-layer MLP is a Universal Boolean Function

# How many layers for a Boolean MLP?

Truth Table					
$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

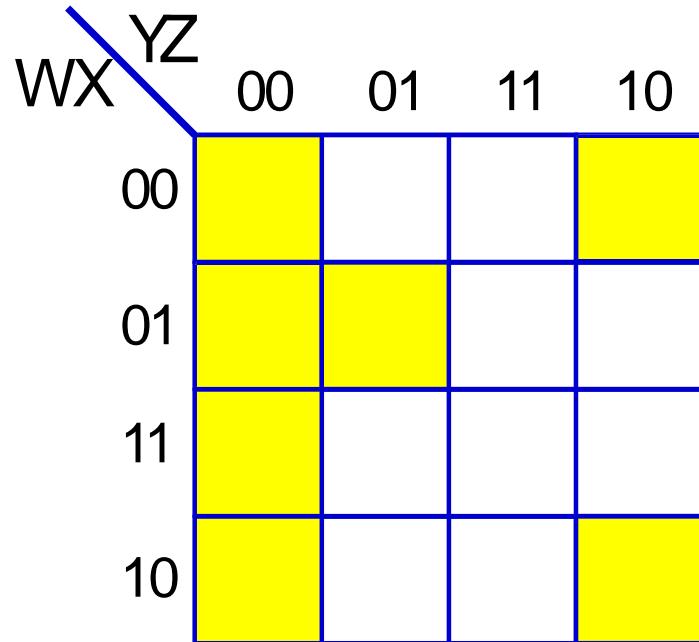
$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Any truth table can be expressed in this manner!
- A one-hidden-layer MLP is a Universal Boolean Function

But what is the largest number of perceptrons required in the single hidden layer for an N-input-variable function?

# Reducing a Boolean Function



This is a "Karnaugh Map"

It represents a truth table as a grid  
Filled boxes represent input combinations  
for which output is 1; blank boxes have  
output 0

Adjacent boxes can be "grouped" to  
reduce the complexity of the DNF formula  
for the table

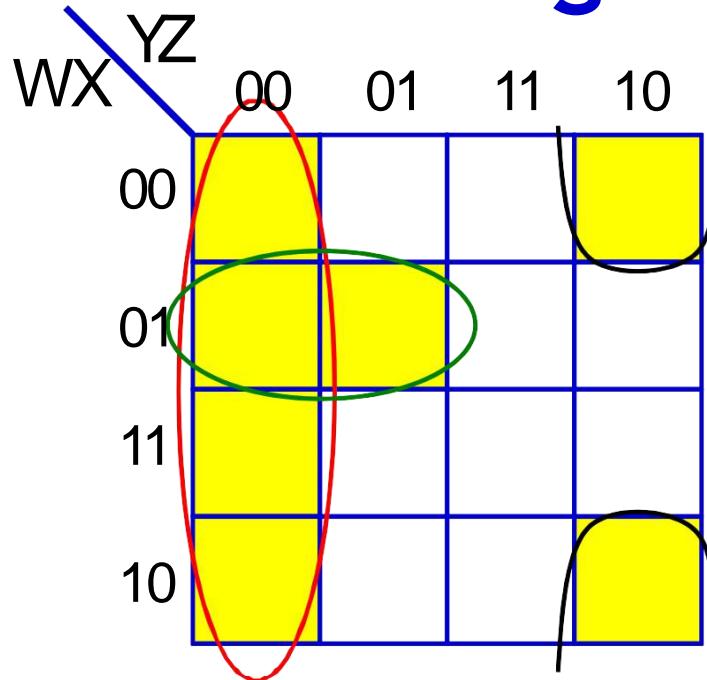
- DNF form:
  - Find groups
  - Express as reduced DNF

# Reducing a Boolean Function

	WX	YZ	00	01	11	10
00	Y	Y	Y			Y
01	Y	Y				
11	Y					
10	Y					Y

Basic DNF formula will require 7 terms

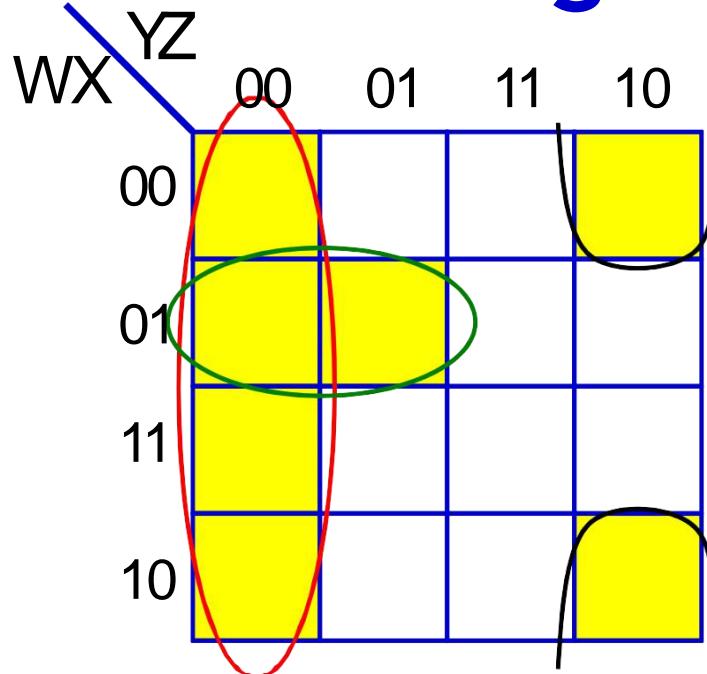
# Reducing a Boolean Function



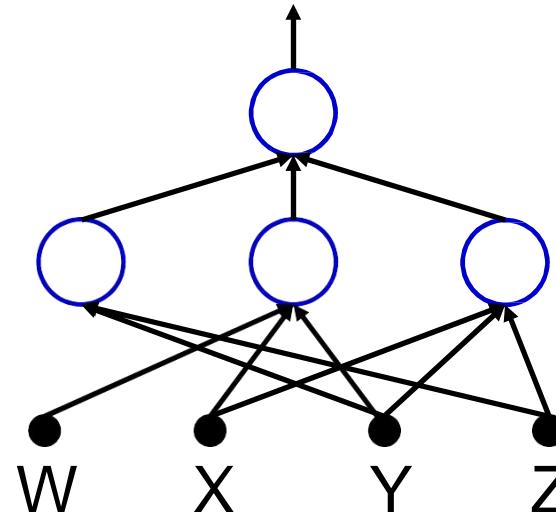
$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$

- *Reduced DNF form:*
  - Find groups
  - Express as reduced DNF

# Reducing a Boolean Function



$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$



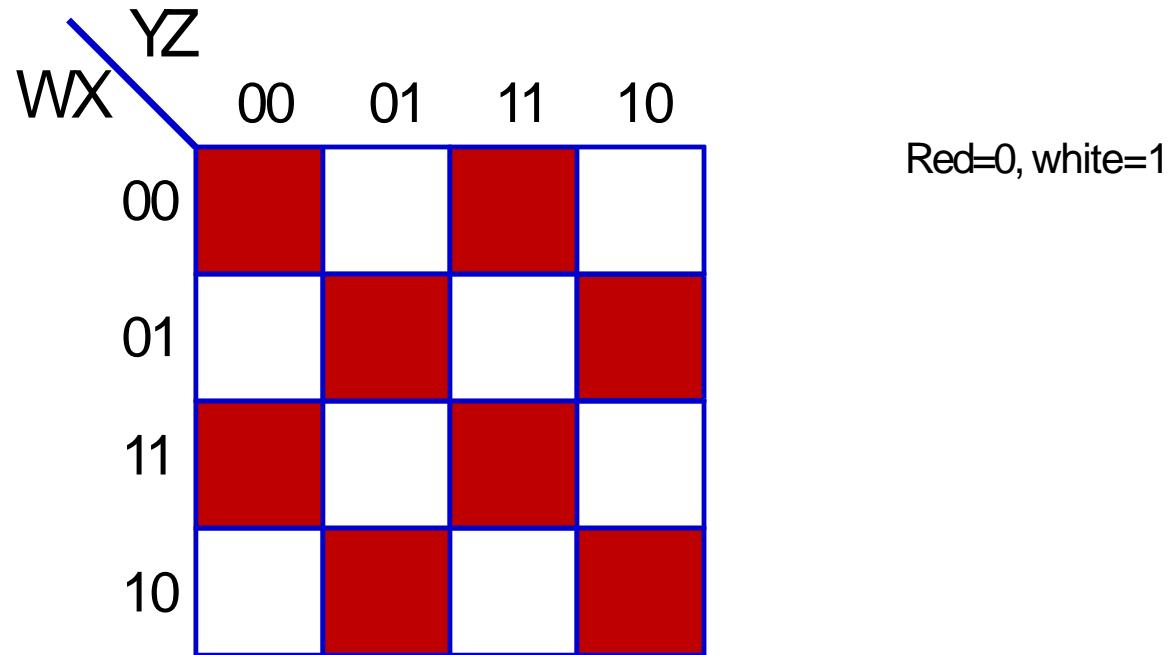
- *Reduced DNF form:*
  - Find groups
  - Express as *reduced* DNF
  - Boolean network for this function needs only 3 hidden units
    - Reduction of the DNF reduces the size of the one-hidden-layer network

# Largest irreducible DNF?

	YZ	00	01	11	10
WX	00				
	01				
	11				
	10				

- What arrangement of ones and zeros simply cannot be reduced further?

# Largest irreducible DNF?



- What arrangement of ones and zeros simply cannot be reduced further?

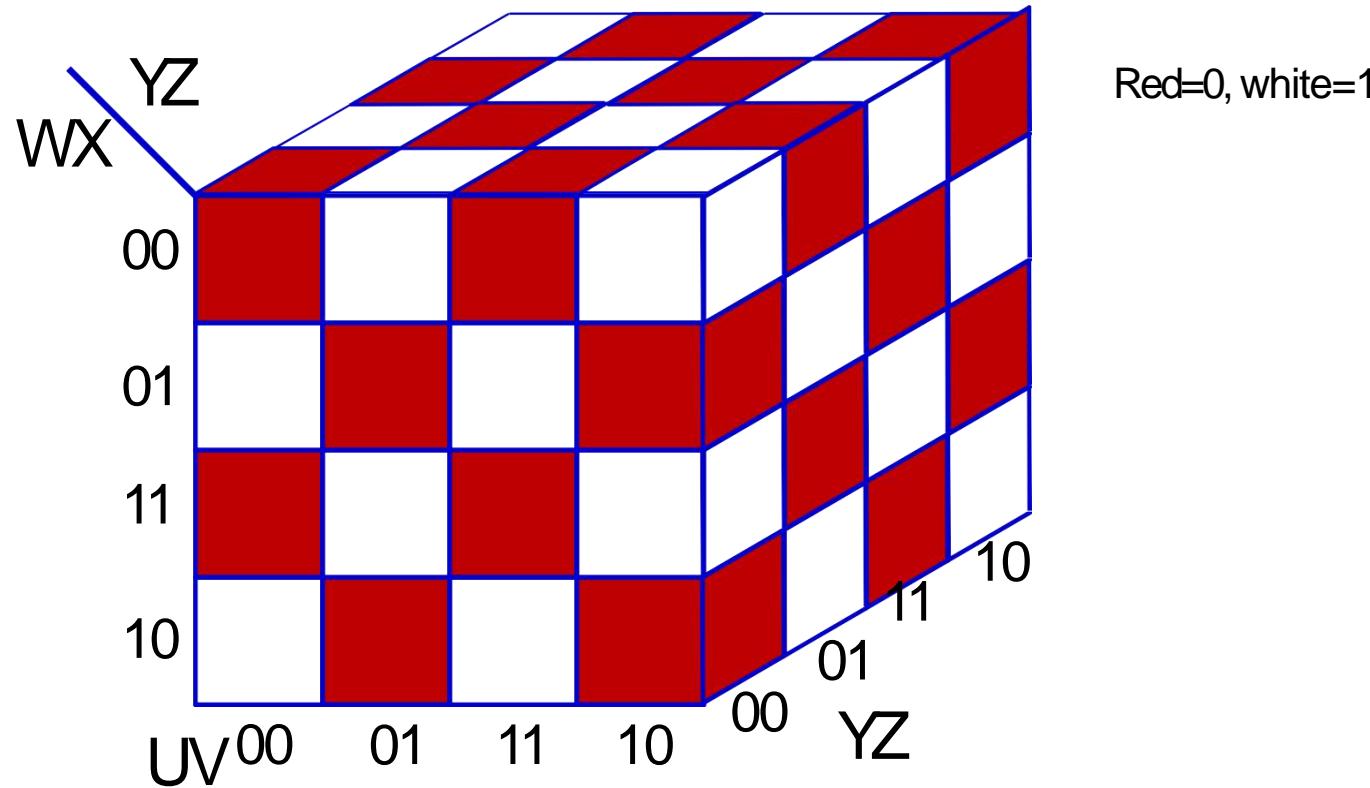
# Largest irreducible DNF?

WX	YZ	00	01	11	10
00		■		■	
01			■		■
11		■		■	
10			■		■

How many neurons  
in a DNF (one-  
hidden-layer) MLP  
for this Boolean  
function?

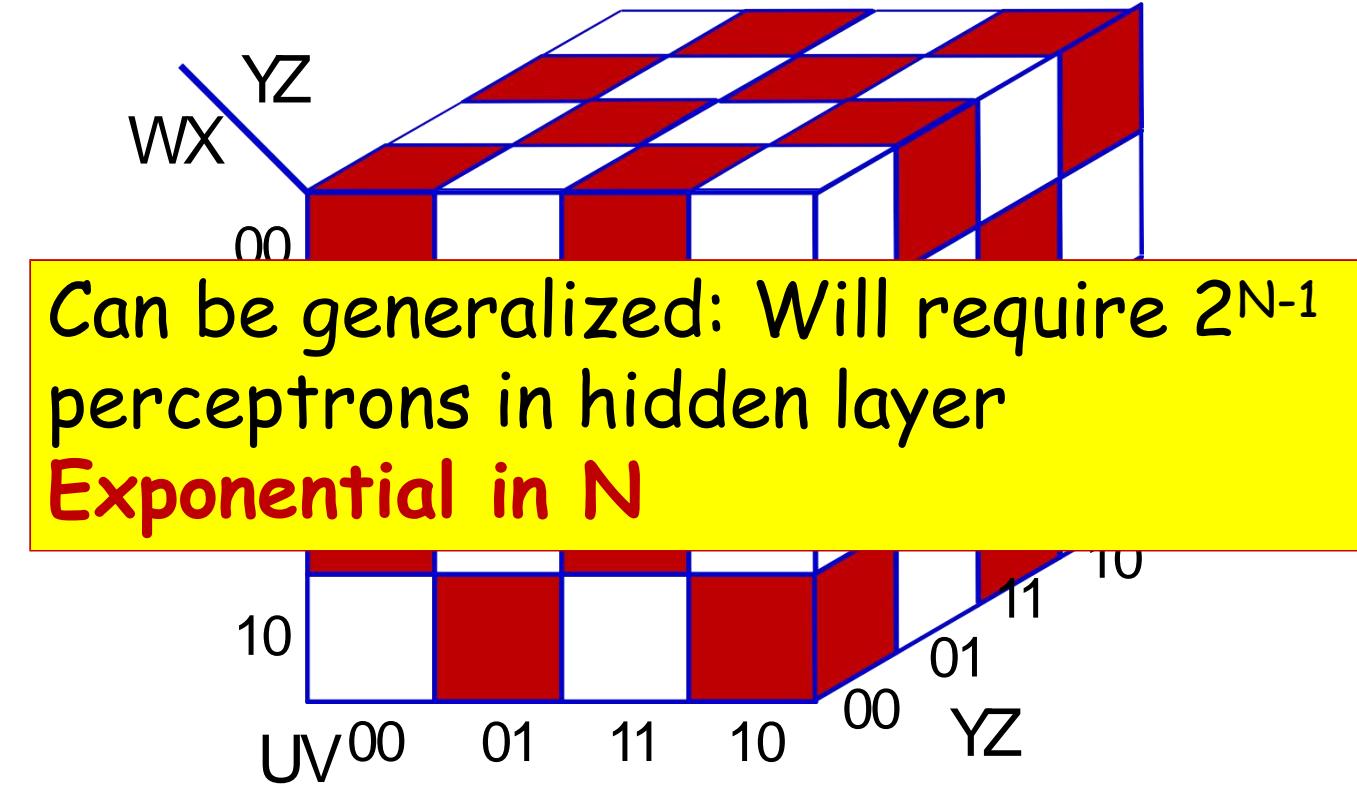
- What arrangement of ones and zeros simply cannot be reduced further?

# Width of a single-layer Boolean MLP



- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function of 6 variables?

# Width of a single-layer Boolean MLP



- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function

# Width of a single-layer Boolean MLP

WX  
YZ

00

Can be generalized: Will require  
 $2^{N-1}$  perceptrons in hidden layer  
**Exponential in N**

10

UV  
00

01

11

10

00

Y  
Z

10

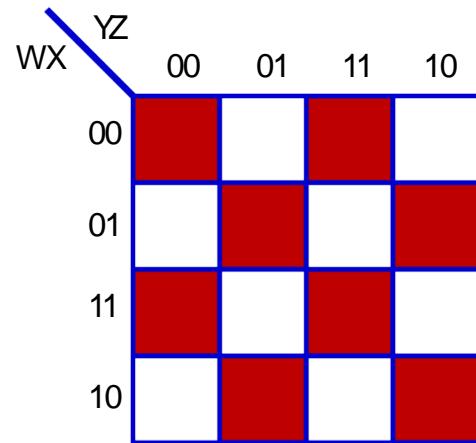
11

01

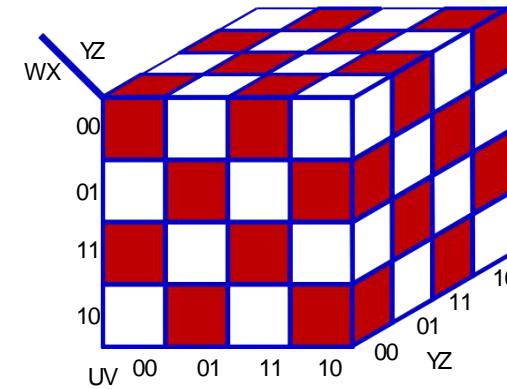
How many units if we use *multiple layers*?

layer) MLP for this Boolean function

# Width of a deepMLP

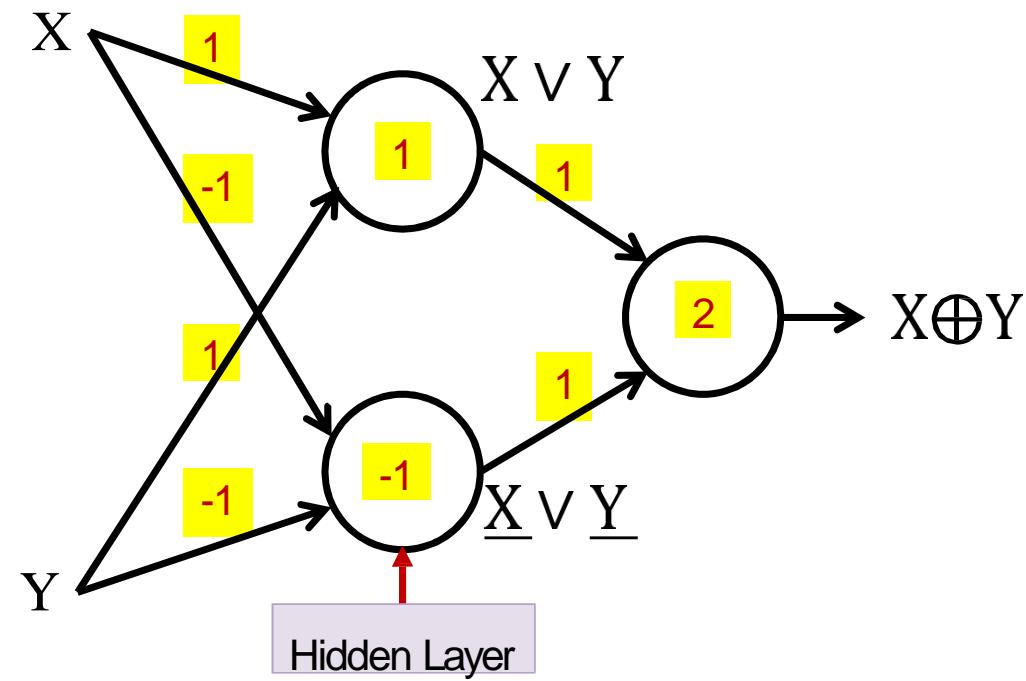


$$O = W \oplus X \oplus Y \oplus Z$$



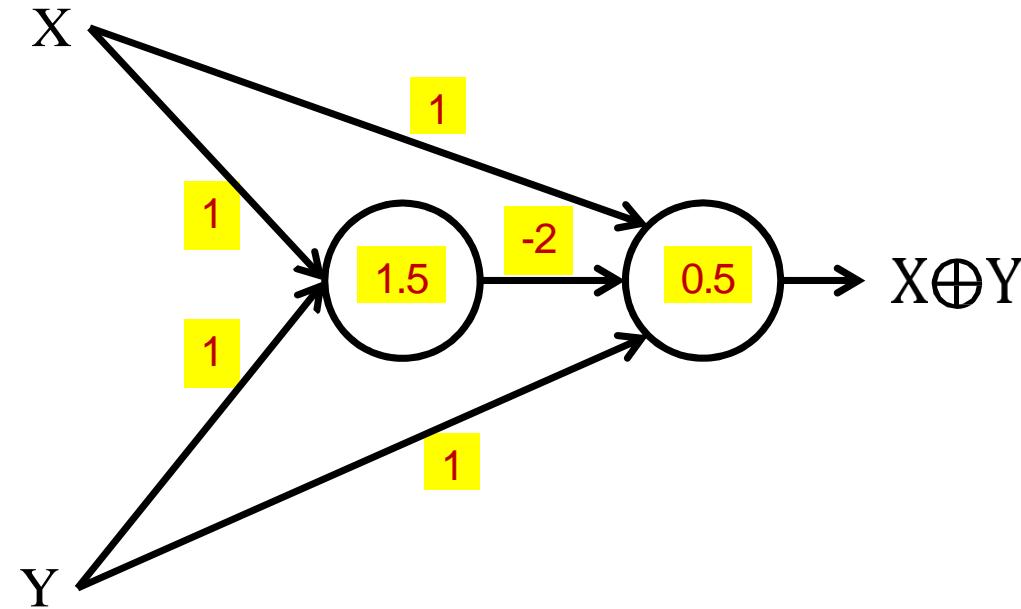
$$O = U \oplus V \oplus W \oplus X \oplus Y \oplus Z$$

# Multi-layer perceptron XOR



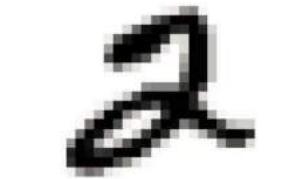
- An XOR takes three perceptrons

# Multi-layer perceptron XOR

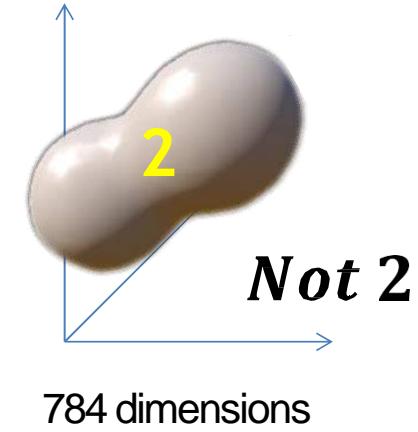
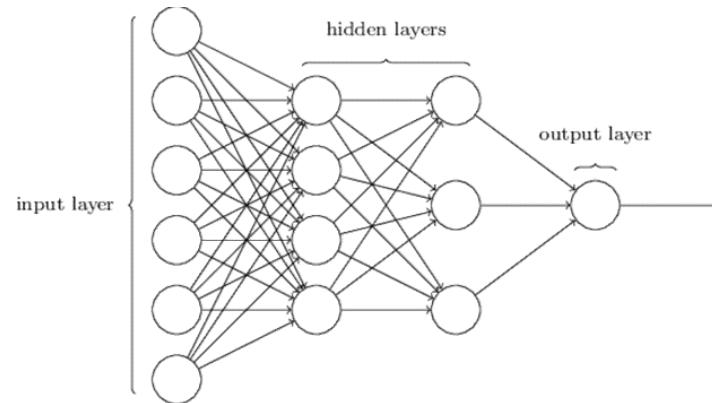


- With 2 neurons
  - 5 weights and two thresholds

# The MLP as a classifier

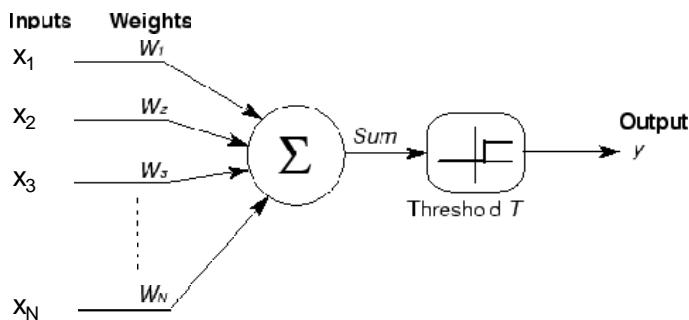


784 dimensions  
(MNIST)



- MLP as a function over real inputs
- MLP as a function that finds a complex “decision boundary” over a space of *reals*

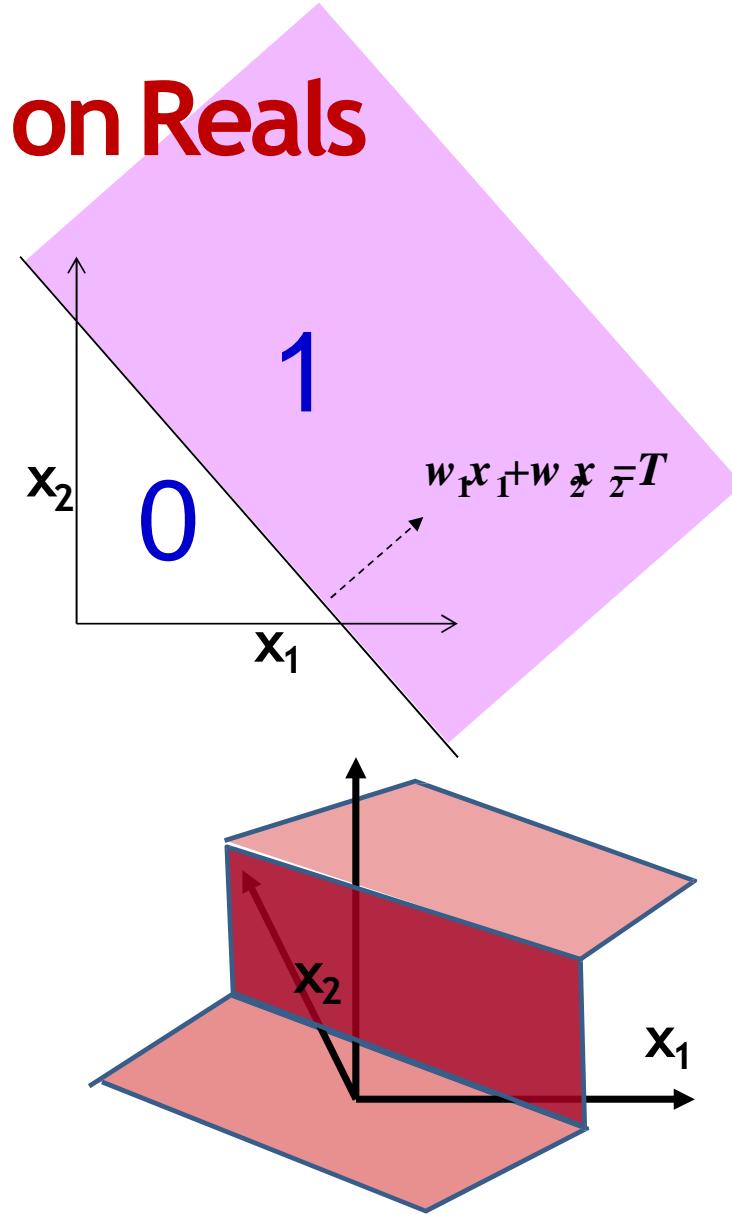
# A Perceptron on Reals



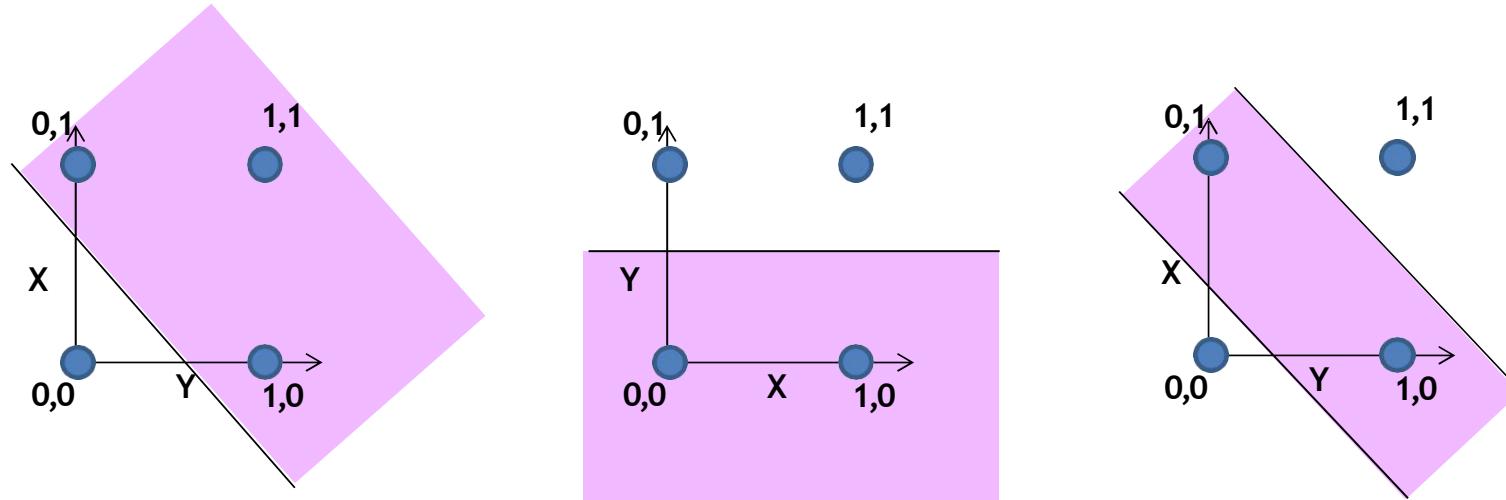
$$y = \begin{cases} 1 & \text{if } \sum_i w_i x_i \geq T \\ 0 & \text{else} \end{cases}$$

- A perceptron operates on *real-valued* vectors

– This is a *linear classifier*

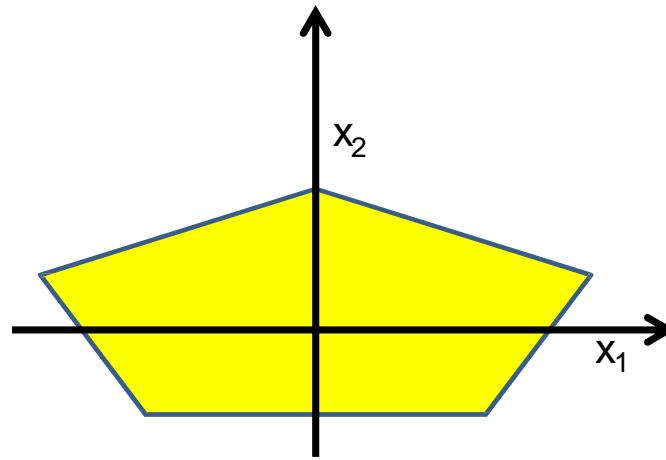


# Boolean functions with a real perceptron



- Boolean perceptrons are also linear classifiers
  - Purple regions are 1

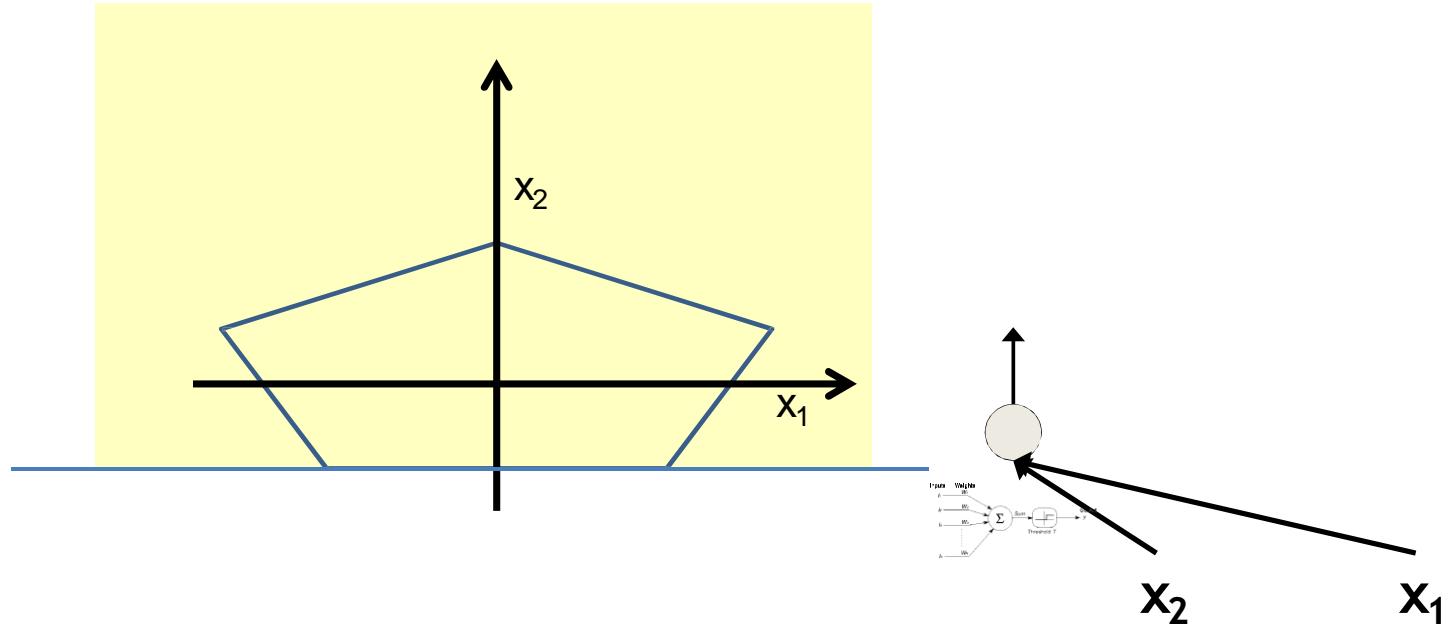
# Composing complicated “decision” boundaries



Can now be composed into  
“networks” to compute arbitrary  
classification “boundaries”

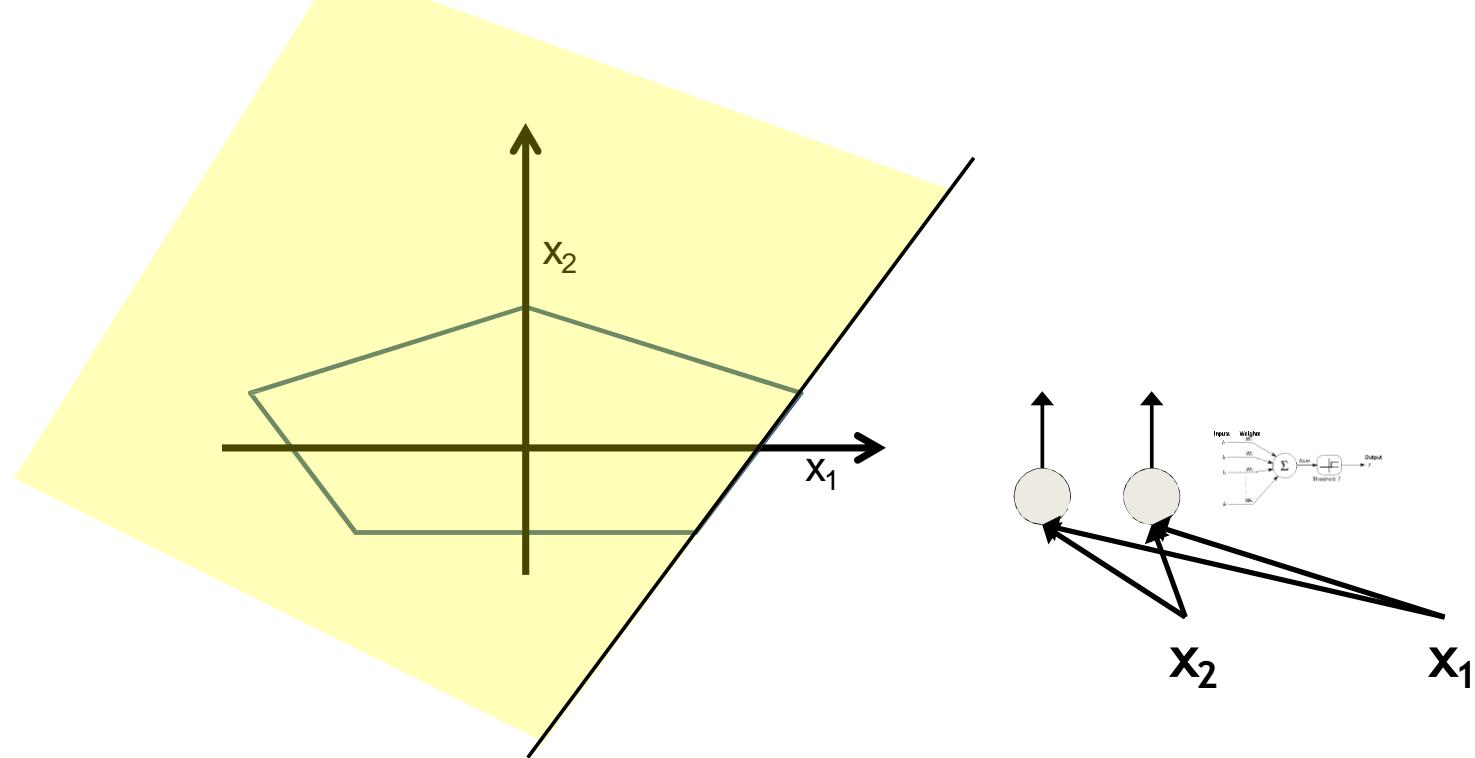
- Build a network of units with a single output that fires if the input is in the coloured area

# Booleans over the reals



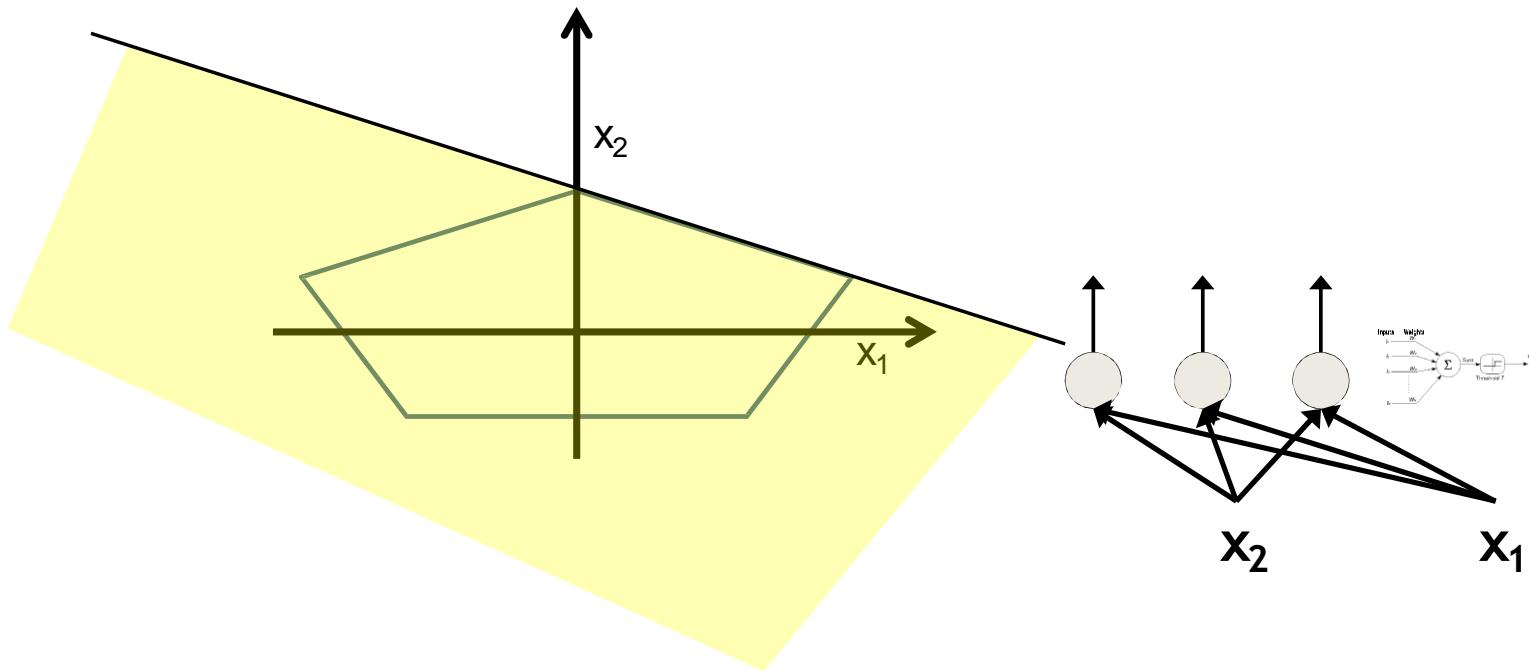
- The network must fire if the input is in the coloured area

# Booleans over the reals



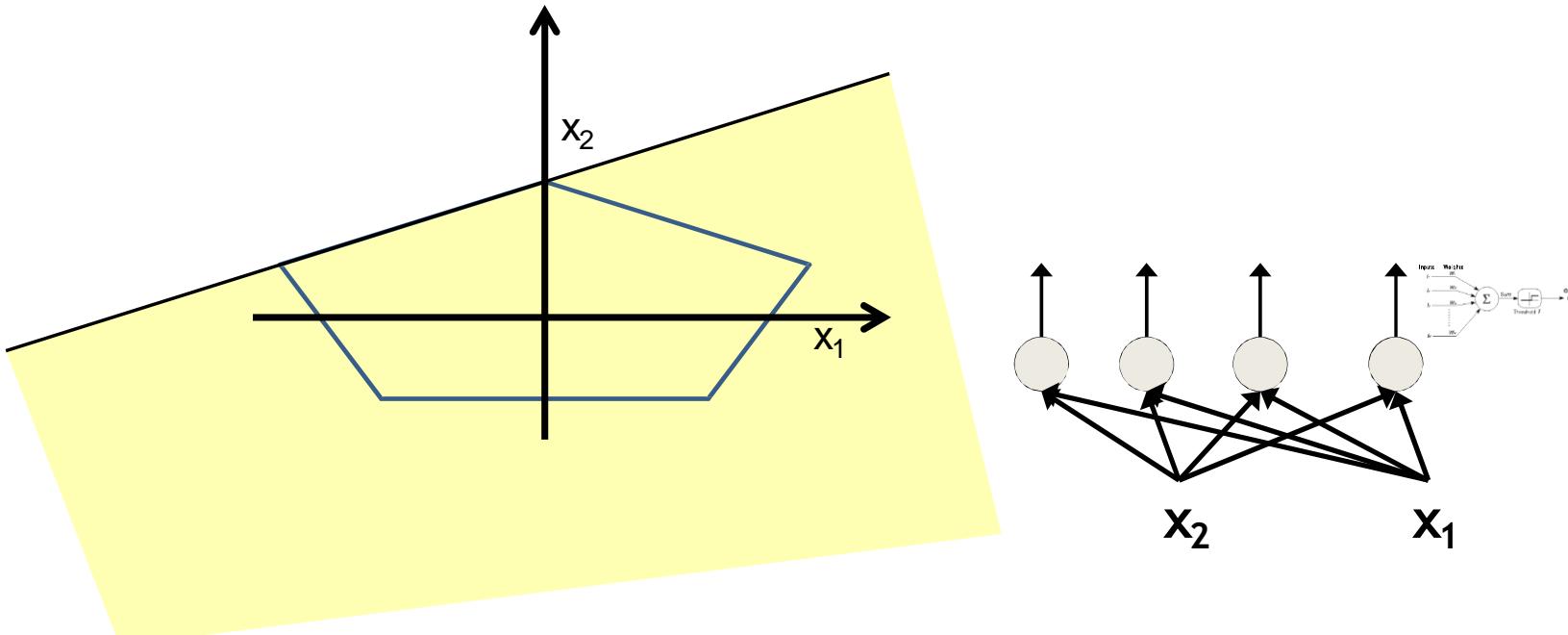
- The network must fire if the input is in the coloured area

# Booleans over the reals



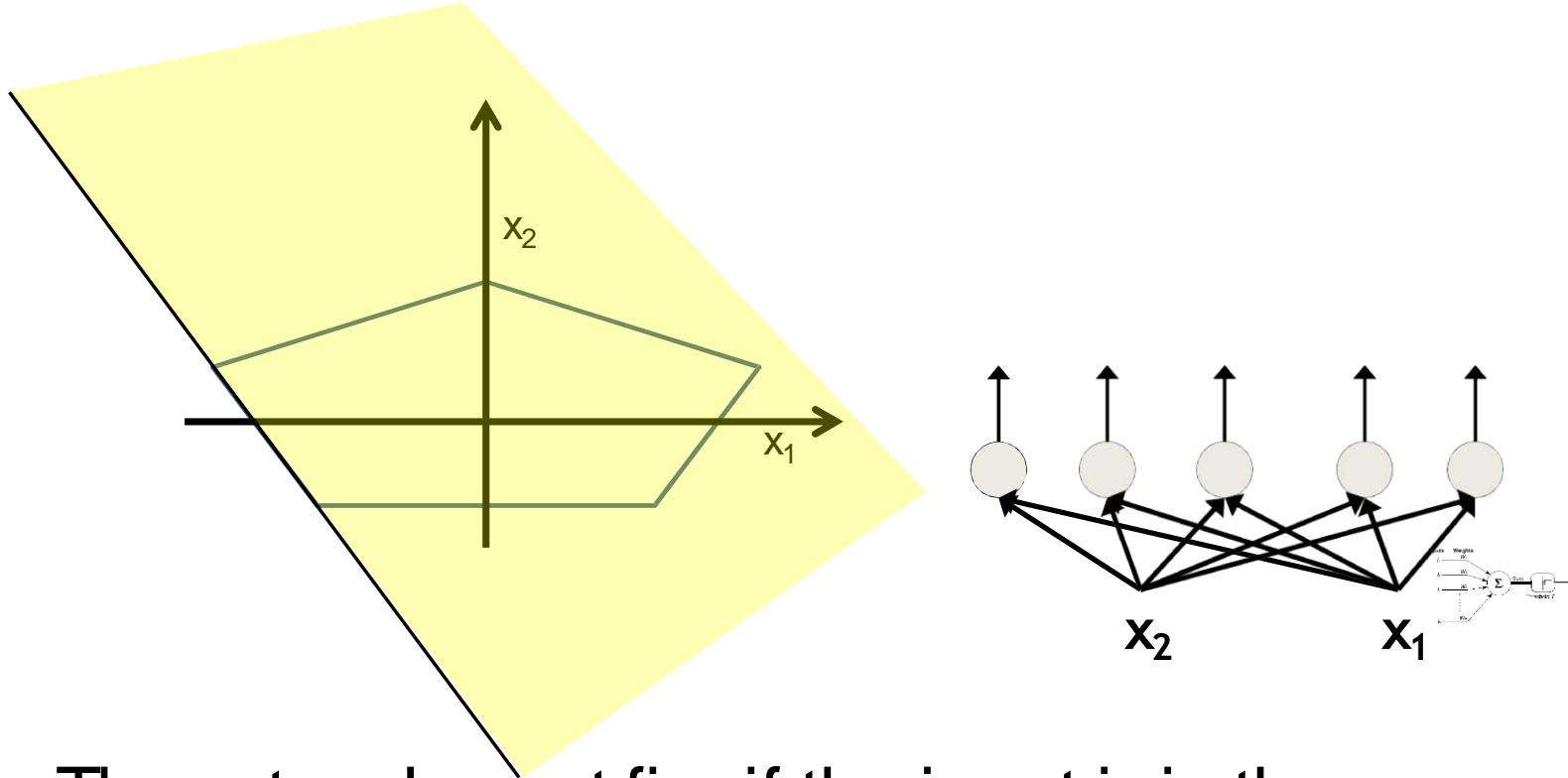
- The network must fire if the input is in the coloured area

# Booleans over the reals



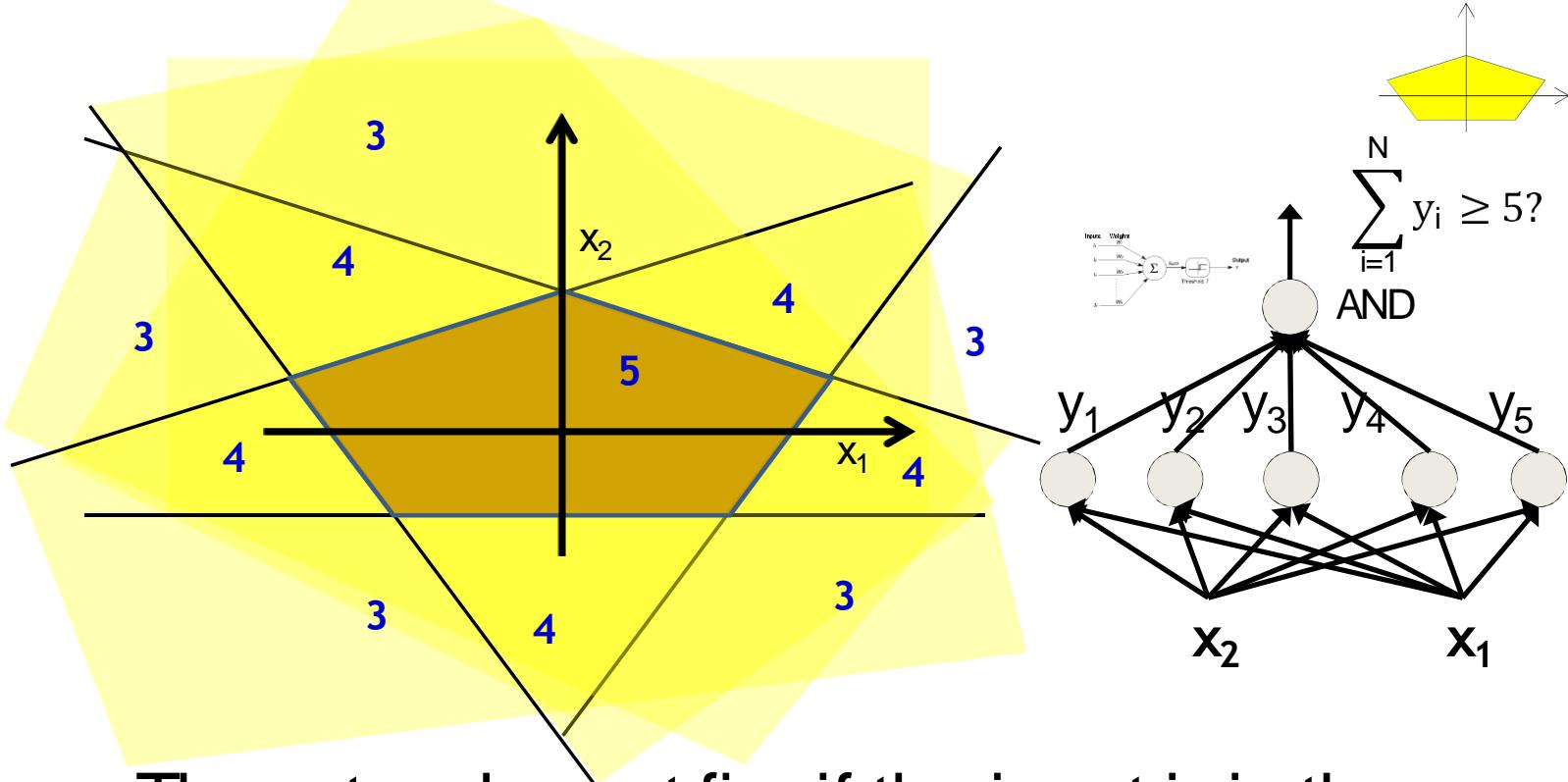
- The network must fire if the input is in the coloured area

# Booleans over the reals



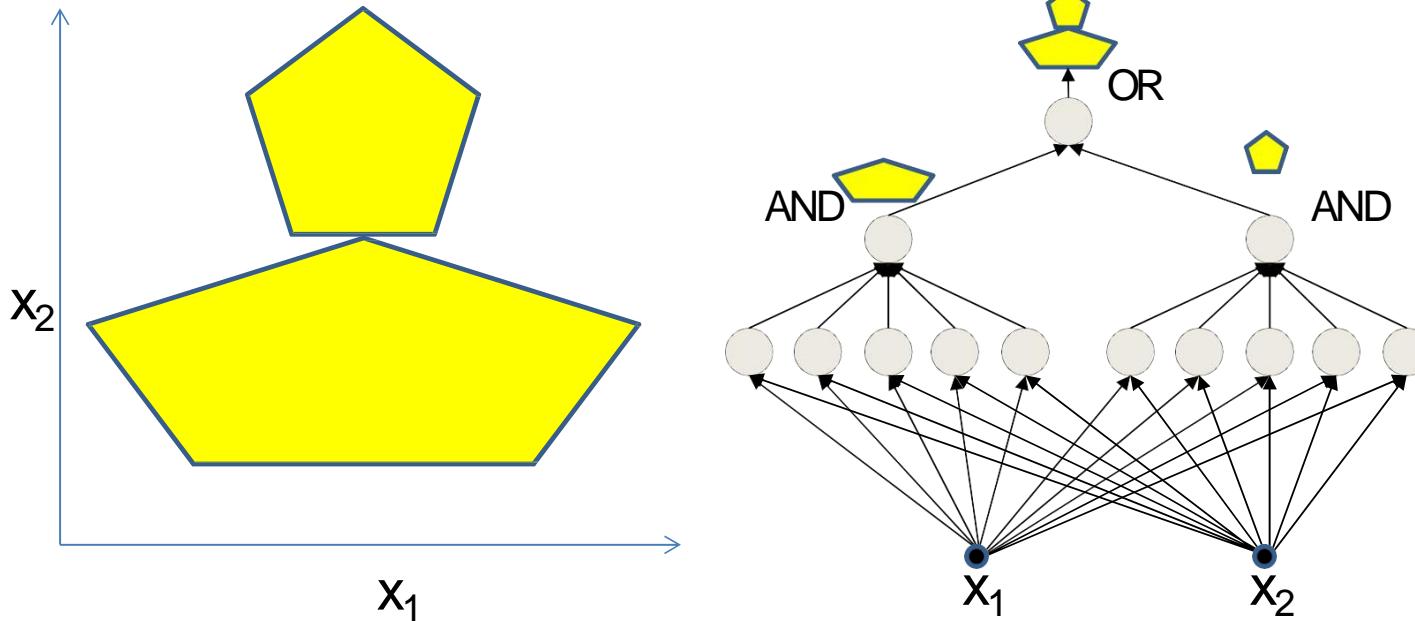
- The network must fire if the input is in the coloured area

# Booleans over the reals



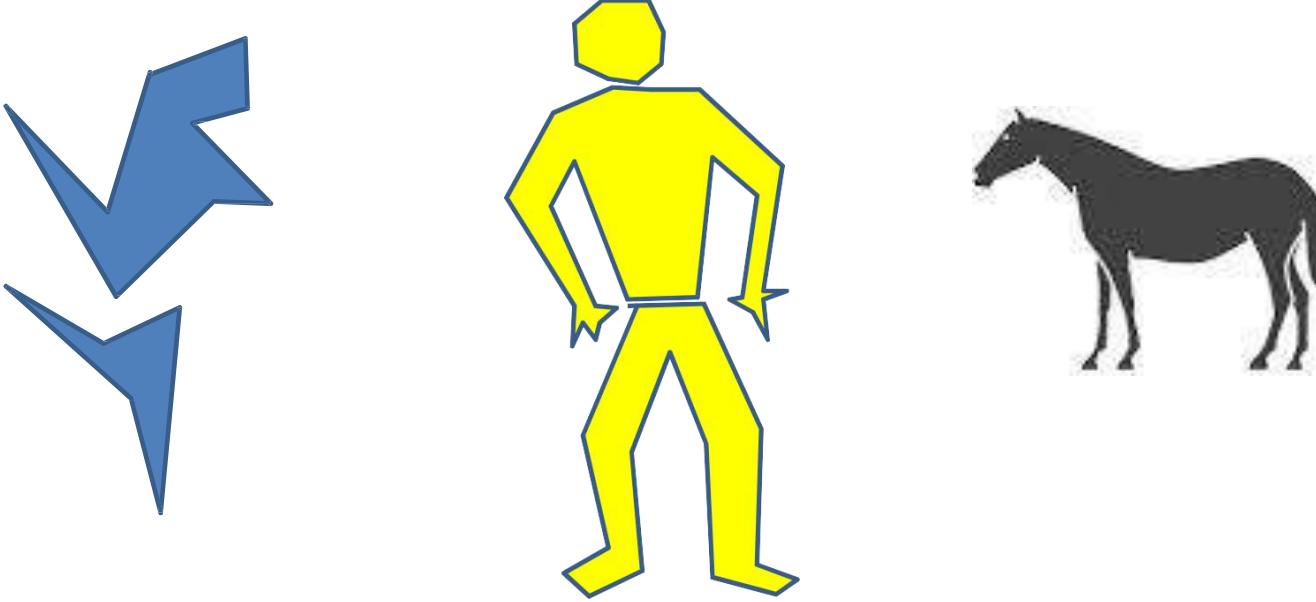
- The network must fire if the input is in the coloured area

# More complex decision boundaries



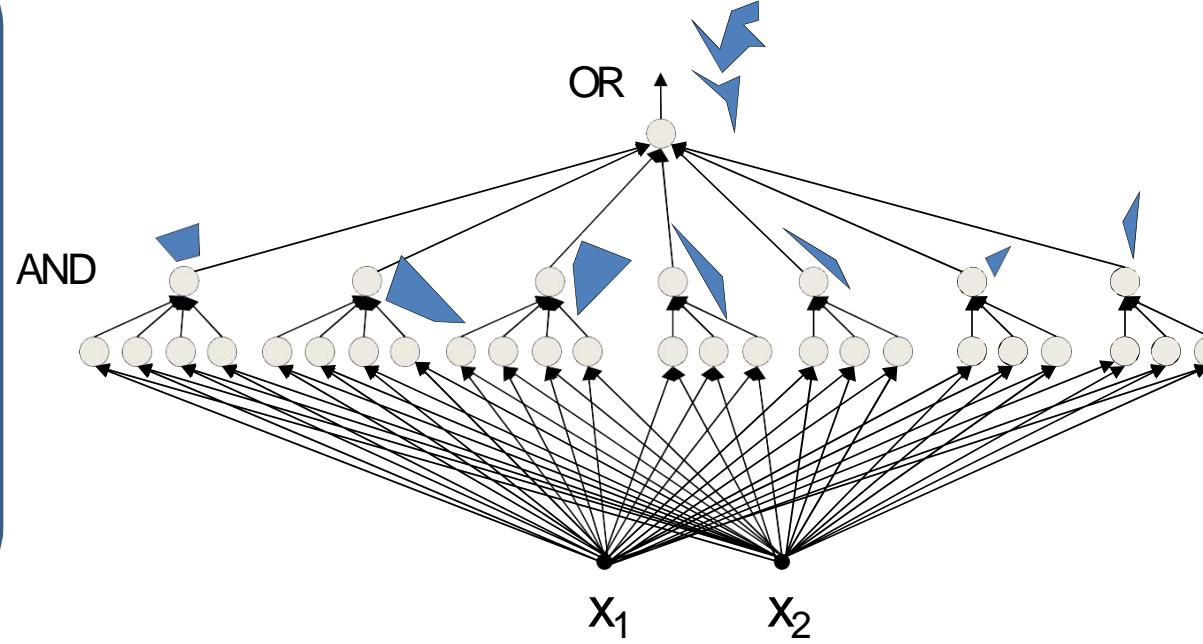
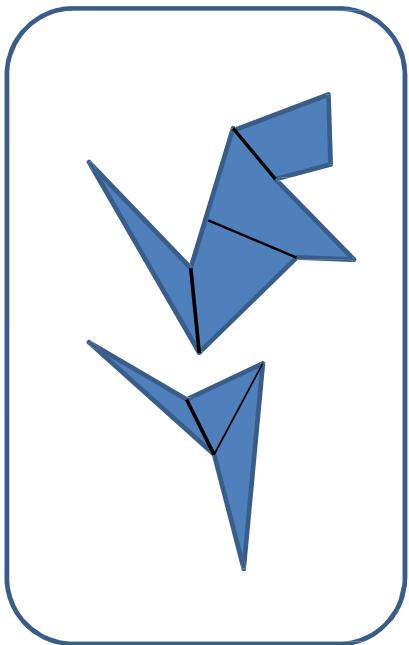
- Network to fire if the input is in the yellow area
  - “OR” two polygons
  - A third layer is required

# Complex decision boundaries



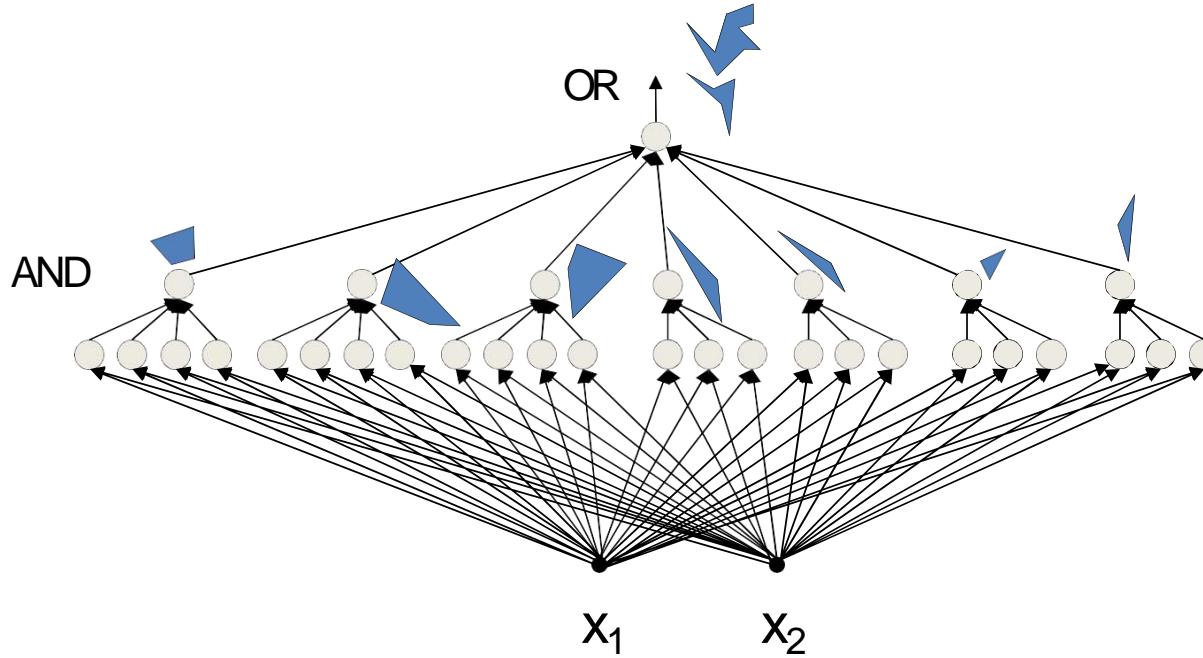
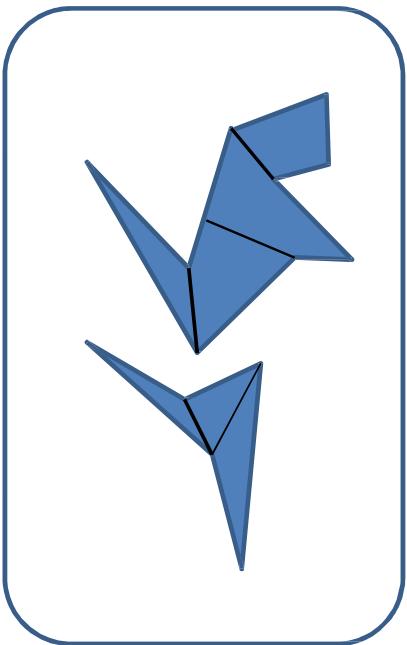
- Can compose *arbitrarily* complex decision boundaries

# Complex decision boundaries



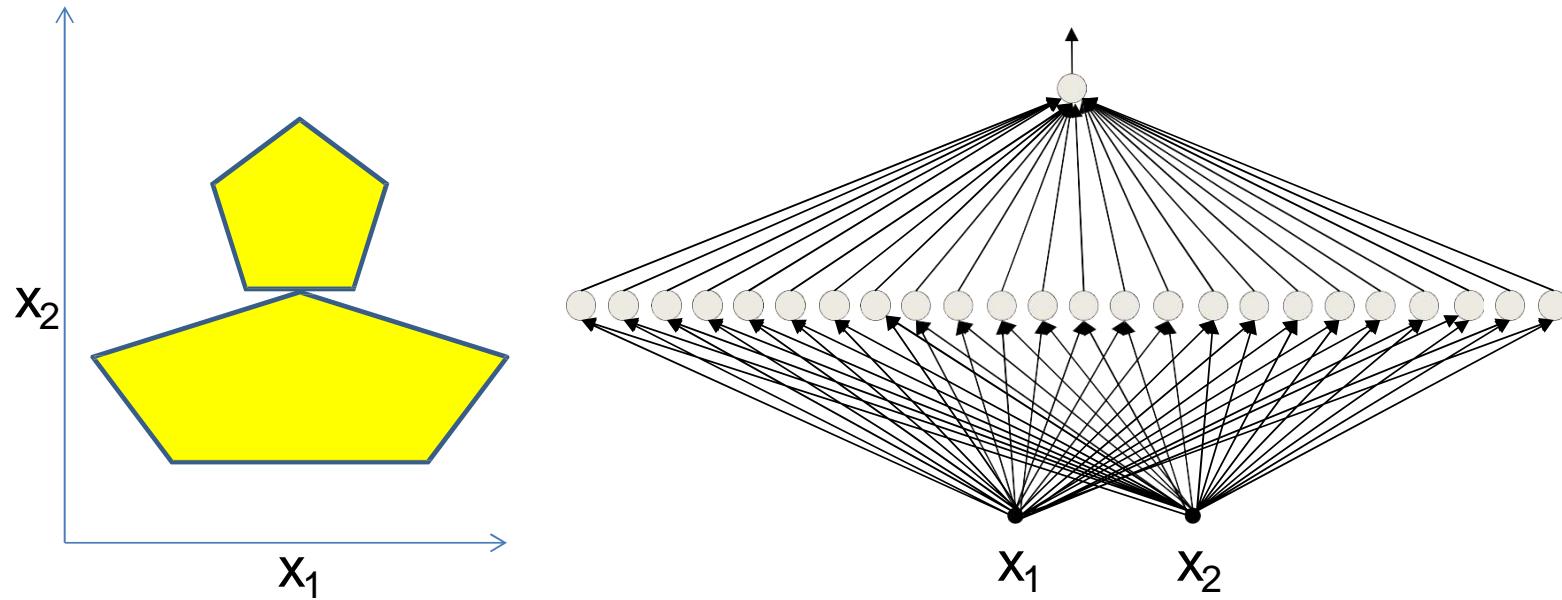
- Can compose *arbitrarily* complex decision boundaries

# Complex decision boundaries



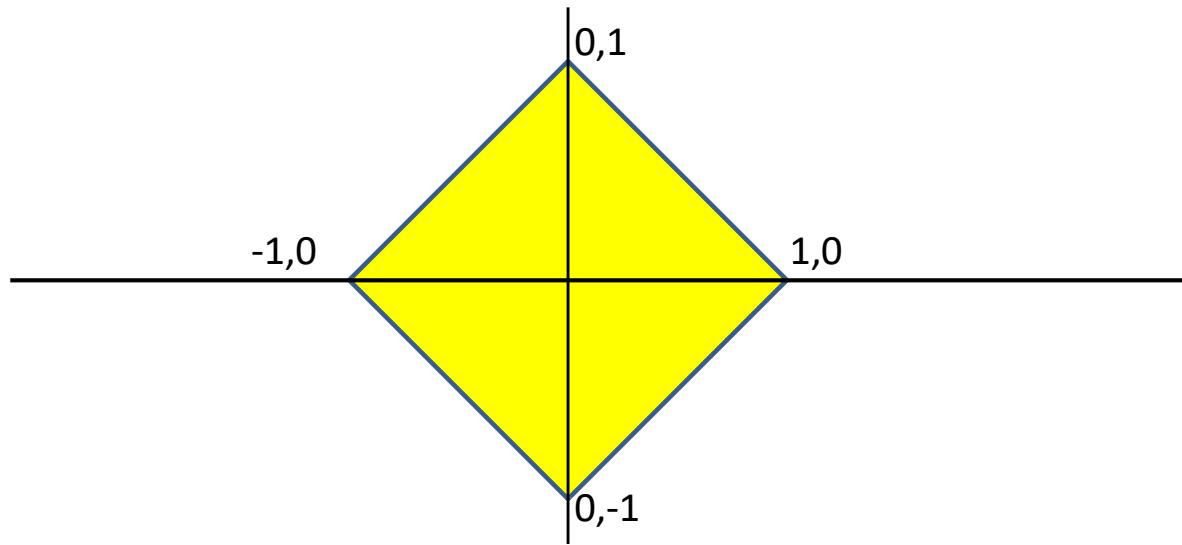
- Can compose *arbitrarily* complex decision boundaries
  - With *only one hidden layer!*
  - **How?**

## Exercise: compose this with one hidden layer



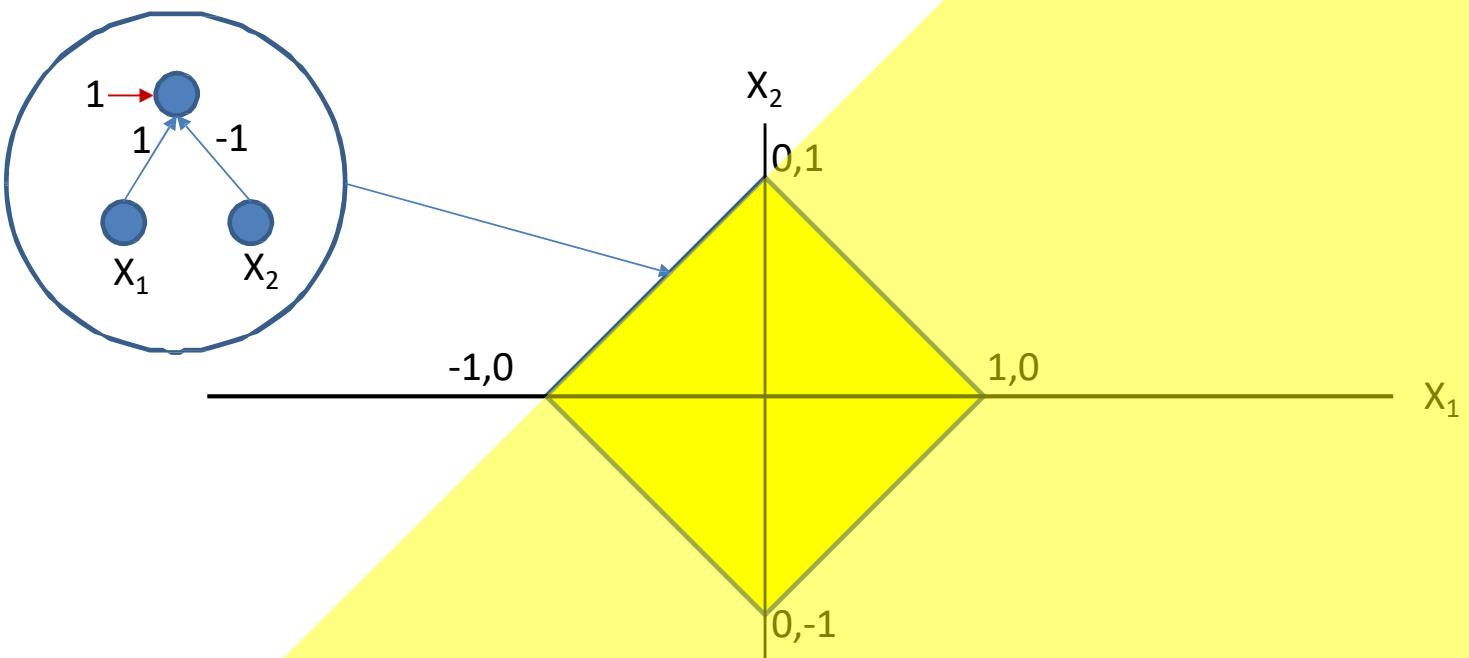
- How would you compose the decision boundary to the left with only *one* hidden layer?

# Construct by hand



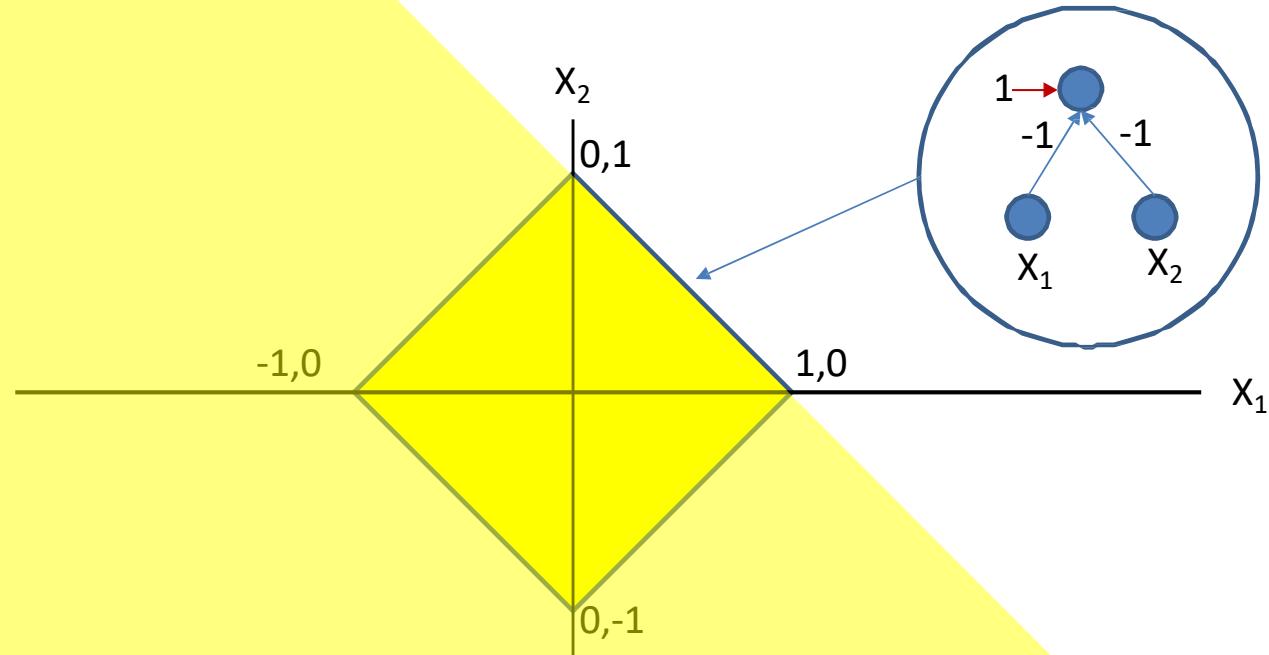
- Given a function, *handcraft* a network to satisfy it
- E.g.: Build an MLP to classify this decision boundary

# Construct by hand



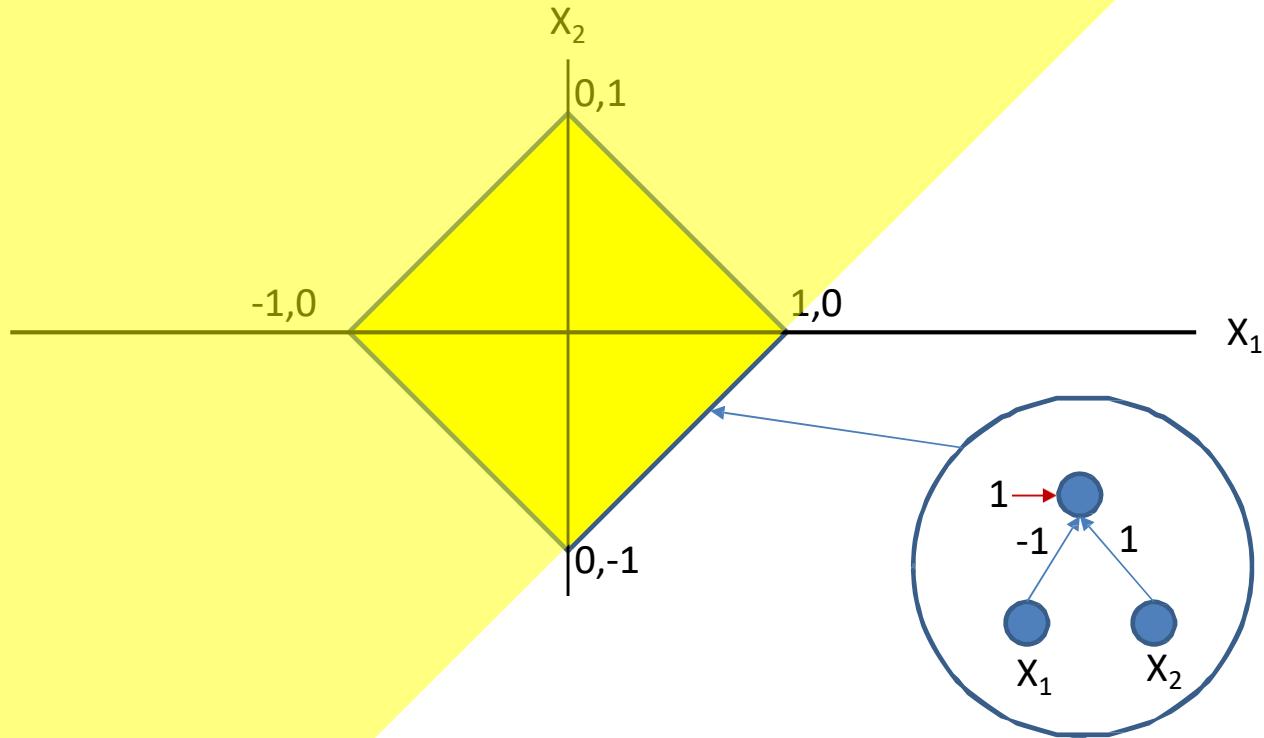
Assuming simple perceptrons:  
 $\text{output} = 1 \text{ if } \sum_i w_i x_i + b_i \geq 0, \text{ else } 0$

# Construct by hand



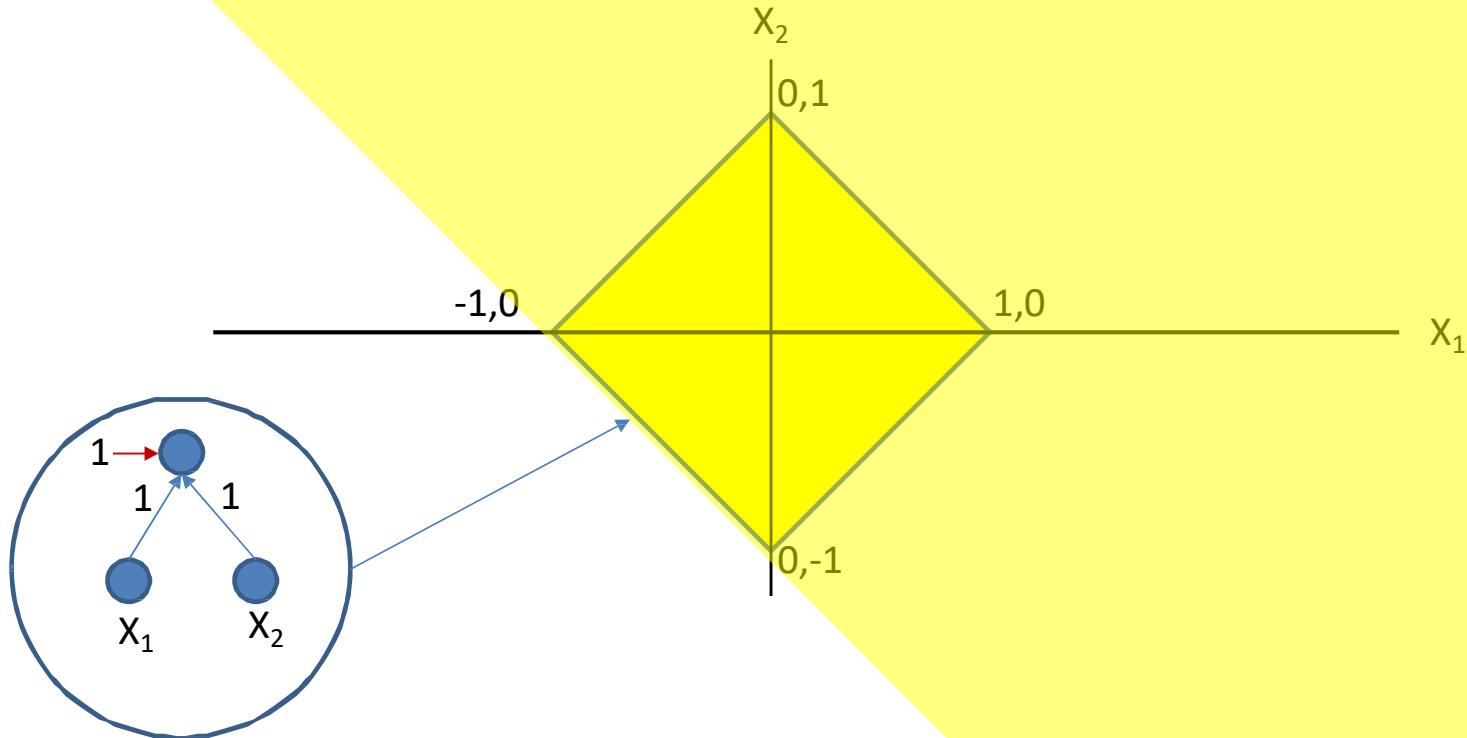
Assuming simple perceptrons:  
 $\text{output} = 1 \text{ if } \sum_i w_i x_i + b_i \geq 0, \text{ else } 0$

# Construct by hand



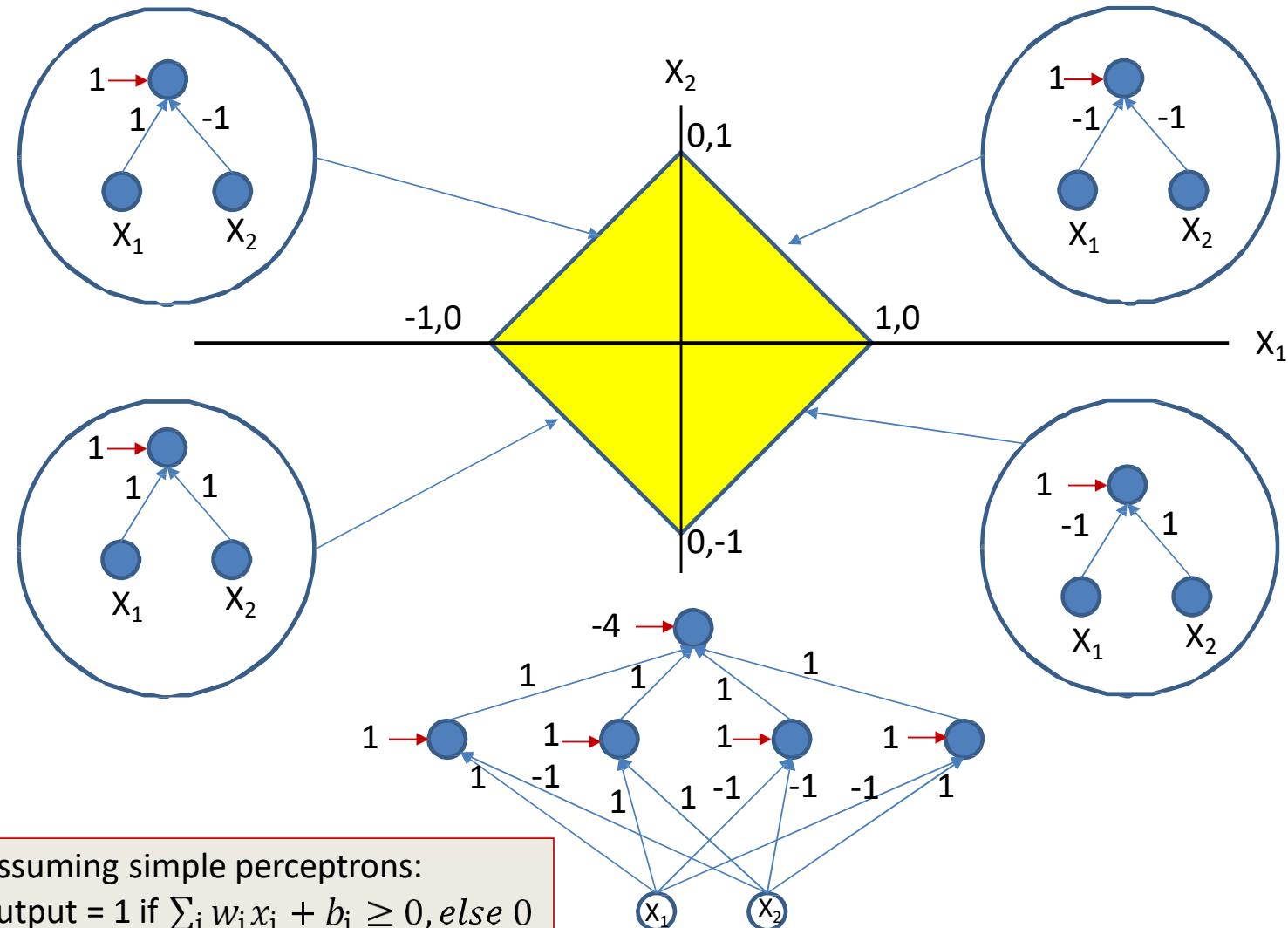
Assuming simple perceptrons:  
 $\text{output} = 1 \text{ if } \sum_i w_i x_i + b_i \geq 0, \text{ else } 0$

# Construct by hand

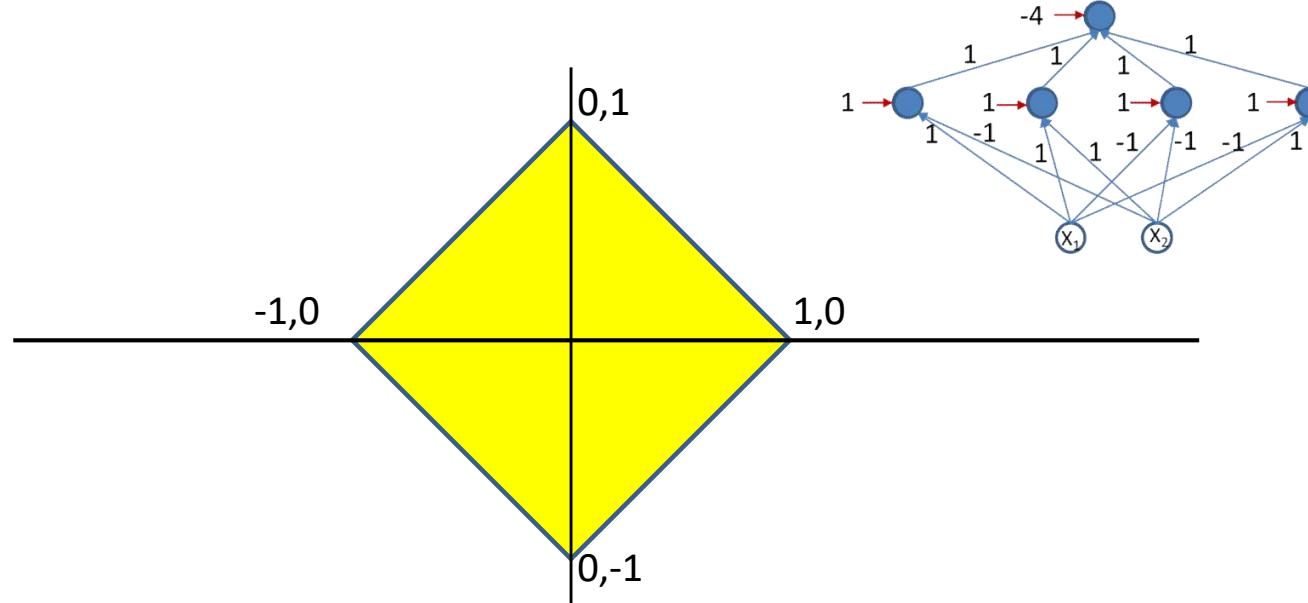


Assuming simple perceptrons:  
 $\text{output} = 1 \text{ if } \sum_i w_i x_i + b_i \geq 0, \text{ else } 0$

# Construct by hand



# Construct by hand



- Given a function, *handcraft* a network to satisfy it
- E.g.: Build an MLP to classify this decision boundary
- Not possible for all but the simplest problems..

# How to Specify/Design Perceptron Parameters

# $x_1$ OR $x_2$ with Perceptron (Step Threshold)

Perceptron with hard/step threshold

Output = 0 if input  $w_1*x_1 + w_2*x_2 < \text{threshold } T$   
= 1 if input  $\geq T$

From OR truth table,

$w_1*0 + w_2*0 < T$  implies  $T > 0$

$w_1*1+w_2*0 \geq T$  implies  $w_1 \geq T$

$w_1*0+w_2*1 \geq T$  implies  $w_2 \geq T$

$w_1*1+w_2*1 \geq T$  implies  $w_1+w_2 \geq T$

Choose,  $T=1$  (you can choose any  $T > 0$ )

Then  $w_1=1$  (you can choose any value  $\geq T$ )

Similarly choose  $w_2=1$  (you can choose any value  $\geq T$ )

$w_1+w_2 \geq T$  is automatically satisfied for  $w_1=w_2=1$  and  $T=1$

X1 AND x2 Truth Table

x1	x2	Output
0	0	0
1	0	1
0	1	1
1	1	1

Thus, AND can be realized with  $w_1=1$ ,  $w_2=1$ ,  $T=1$ . Important to note that other values of  $w_1$ ,  $w_2$ , and  $T$  can also implement OR, as long as the about 4 inequalities are satisfied

# $x_1$ AND $x_2$ with Perceptron (Step Threshold)

Perceptron with hard/step threshold

Output = 0 if input  $w_1 \cdot x_1 + w_2 \cdot x_2 < \text{threshold } T$   
= 1 if input  $\geq T$

From OR truth table,

$w_1 \cdot 0 + w_2 \cdot 0 < T$  implies  $T > 0$

$w_1 \cdot 1 + w_2 \cdot 0 < T$  implies  $w_1 < T$

$w_1 \cdot 0 + w_2 \cdot 1 < T$  implies  $w_2 < T$

$w_1 \cdot 1 + w_2 \cdot 1 \geq T$  implies  $w_1 + w_2 \geq T$

Choose,  $T = 2$  (you can choose any  $T > 0$ )

Then  $w_1 = 1$  (you can choose any value  $< T$ )

Similarly choose  $w_2 = 1$  (you can choose any value  $< T$ )

$w_1 + w_2 \geq T$  is automatically satisfied for  $w_1 = w_2 = 1$  and  $T = 2$

X1 AND x2 Truth Table

x1	x2	Output
0	0	0
1	0	0
0	1	0
1	1	1

Thus, AND can be realized with  $w_1 = 1$ ,  $w_2 = 1$ ,  $T = 2$ . Important to note that other values of  $w_1$ ,  $w_2$ , and  $T$  can also implement AND, as long as the about 4 inequalities are satisfied

# Linear Classification with Perceptron

Perceptron with hard/step threshold

Output = -1 if input  $w_1*x_1 + w_2*x_2 < \text{threshold } T$   
= 1 if input  $\geq T$

From training data,

$$w_1*(-2) + w_2*0 < T \text{ implies } -2w_1 < T$$

$$w_1*1 + w_2*(-1) < T \text{ implies } w_1 - w_2 < T$$

$$w_1*0 + w_2*1 \geq T \text{ implies } w_2 \geq T$$

$$w_1*1 + w_2*1 \geq T \text{ implies } w_1 + w_2 \geq T$$

Choose,  $w_1=1$

Then  $T > -2$ , choose  $T = -1$  (you can choose any value  $> -2$ )

Then,  $1 - w_2 < -1$  or  $w_2 > 2$ . Choose,  $w_2=3$ . (you can choose any value  $> 2$ )

Then,  $w_1 + w_2 \geq T$  is automatically satisfied

Training Dataset

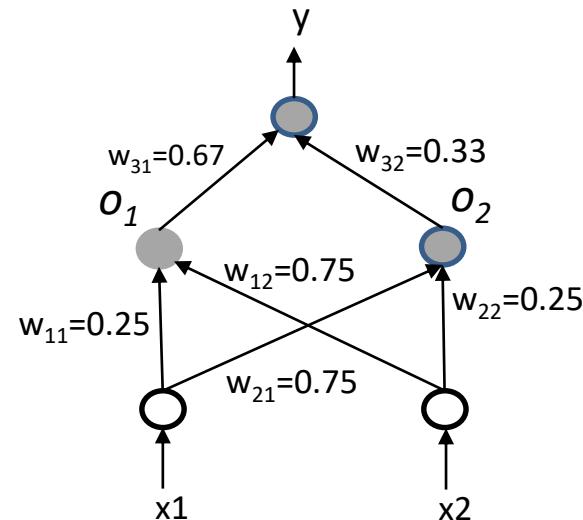
x1	x2	Class
-2	0	-1
1	-1	-1
0	1	1
1	1	1

Thus, classification can be realized with  $w_1=1$ ,  $w_2=3$ ,  $T=-1$ . Important to note that other values of  $w_1$ ,  $w_2$ , and  $T$  can also classify the training dataset, as long as the about 4 inequalities are satisfied

# Parameter Updates in ANN

Leaky ReLU activation function that generates output = input, if input  $\geq 0$ , and  $0.1 * \text{input}$  if output  $< 0$ . What will be the weights  $w_{31}$  and  $w_{12}$  in the next iteration with learning rate = 0.1,  $x_1=x_2=1$ , and target output 0?

Assume, squared difference between the actual and target output is used as the loss function, derivative of activation function = 0 at input = 0, and zero bias at all nodes.



. Assume, error  $E = 0.5*(t-y)^2$ . For  $x_1=x_2=1$ , actual output  $y = 1$  and output of hidden nodes are  $o_1=o_2=1$ . Target output  $t = 0$  as specified in the question. Note  $y = w_{31}o_1 + w_{32}o_2$

$$\Delta w_{31} = -\frac{\eta \delta E}{\delta w_{31}} = -\frac{\eta \delta E}{\delta y} * \frac{\delta y}{\delta w_{31}} = \eta(t - y) * o_1 = -0.1.$$

Thus, value of  $w_{31}$  in the next iteration will be

$$w_{31} + \Delta w_{31} = 0.67 - 0.1 = 0.57.$$

$$\Delta w_{12} = -\frac{\eta \delta E}{\delta w_{12}} = -\frac{\eta \delta E}{\delta y} * \frac{\delta y}{\delta w_{12}} = -0.1 * \frac{\delta y}{\delta o_1} * \frac{\delta o_1}{\delta w_{12}} = -0.1 * w_{31} * x_2 = -0.067.$$

Thus, value of  $w_{12}$  in the next iteration will be

$$w_{12} + \Delta w_{12} = 0.75 - 0.067 = 0.683.$$



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# Introduction to Deep Learning

Kamlesh Tiwari

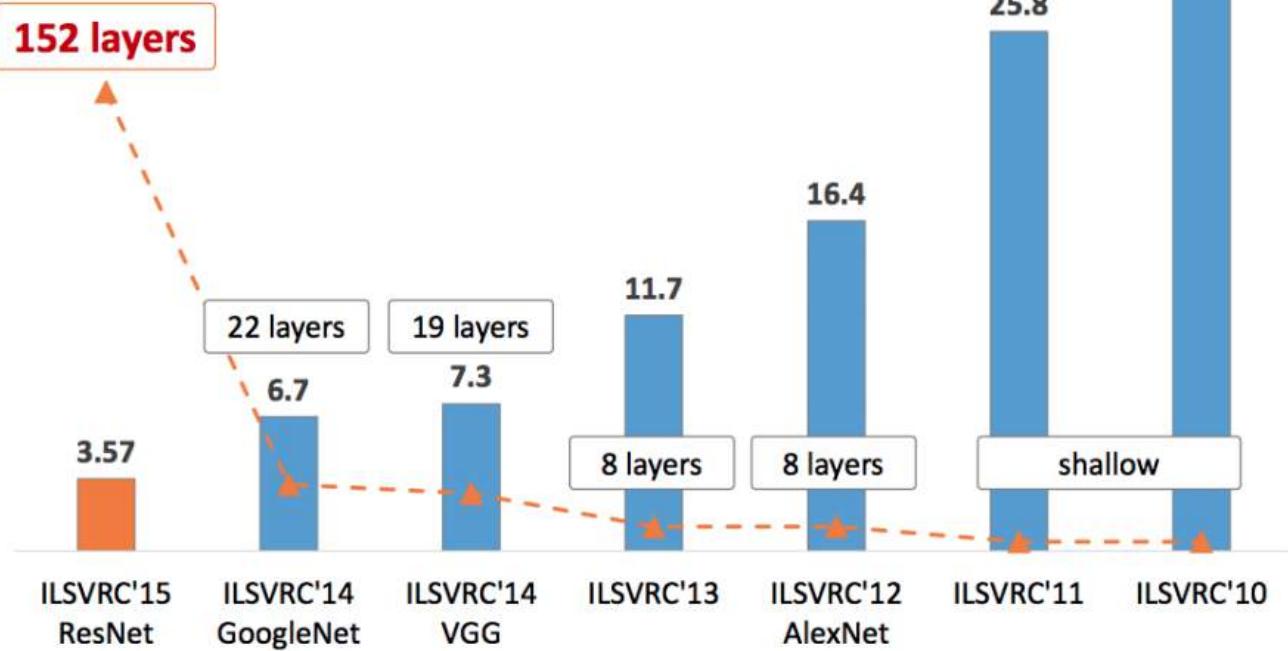
# Deep Learning (DL)

Deep neural networks are capable of solving very complex problems

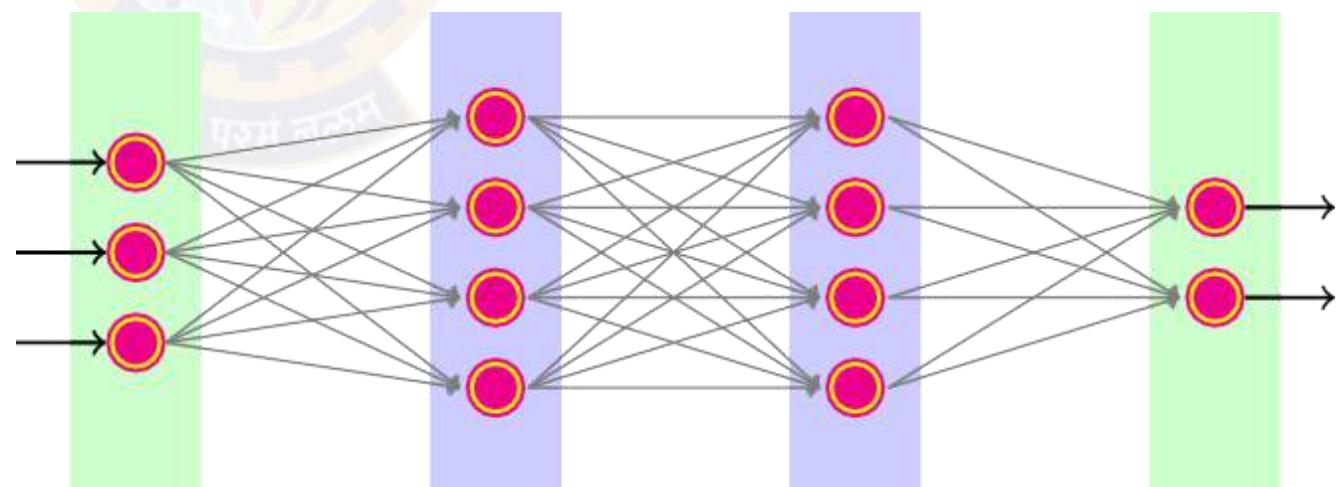
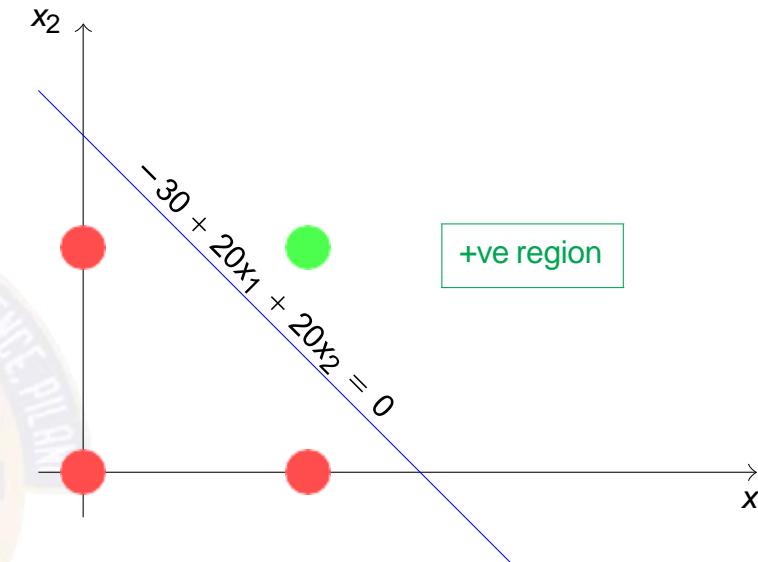
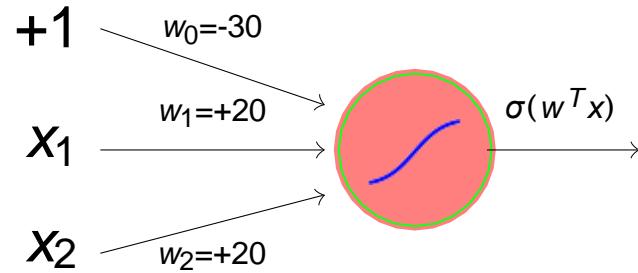
- ImageNet Challenge (2011): identify objects in images  
(4 million images in 21841 categories [Challenge 1000 categories])



Classification: ImageNet Challenge top-5 error

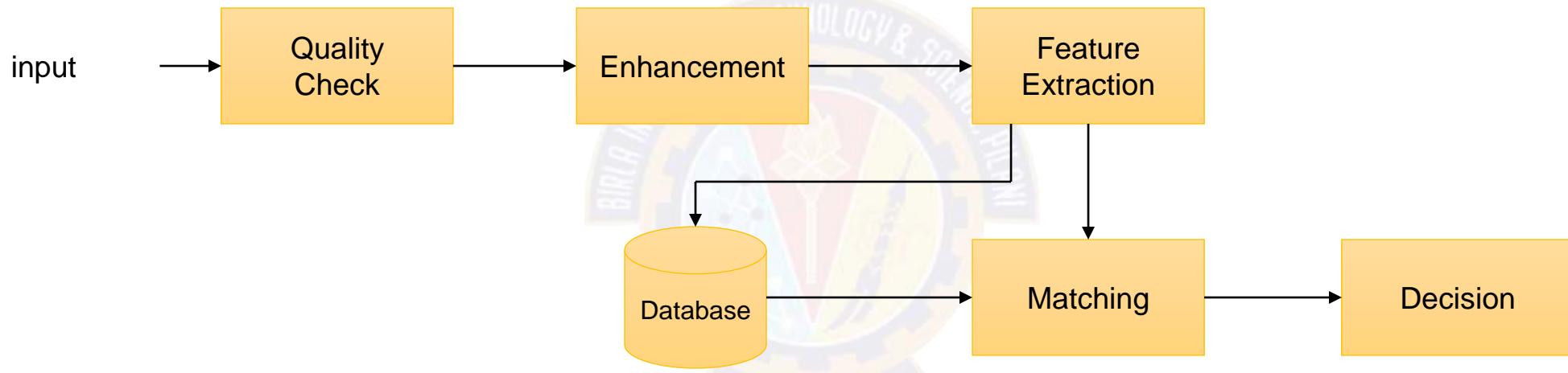


# Essentially it Represents A Decision Boundary



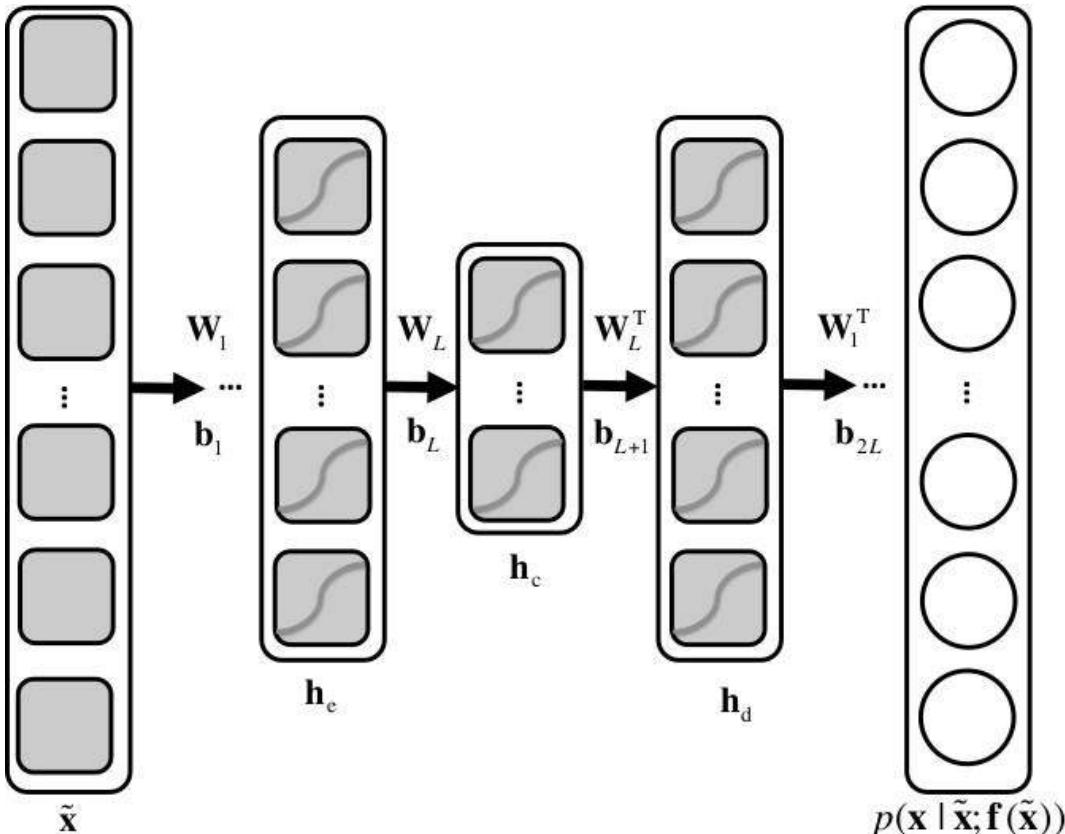
However, complex boundaries could be made using multiple neurons and stacking them in a layer.

# General ML Pipeline



Deep Learning can provided end-to-end learning

# Autoencoders



- An unsupervised learning algorithm, setting the target values to be equal to the inputs
- Learns an approximation

# Deep Learning

- **Shallow to Deep:** is linear combination to composition

$$\sum_i g_i \rightarrow g_1(g_2(g_3(\dots)))$$

- Why?



- Optimization is difficult for a non-convex, non-linear systems
- Freezing some layers makes it shallow.

# Key Points

- Traditional way to hand engineer feature is brittle and not scalable
- **Why now?** because we have data (imageNet), compute power (GPUs) and software (tensorFlow)
- DL learns underlying features/representations directly from data.
- DL have been found useful to many real life problems. Capable of solving very complex problems.
- As the data increases algorithm DL becomes better
- Deep learning is **NOT** just adding more layers to a neural network.



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN

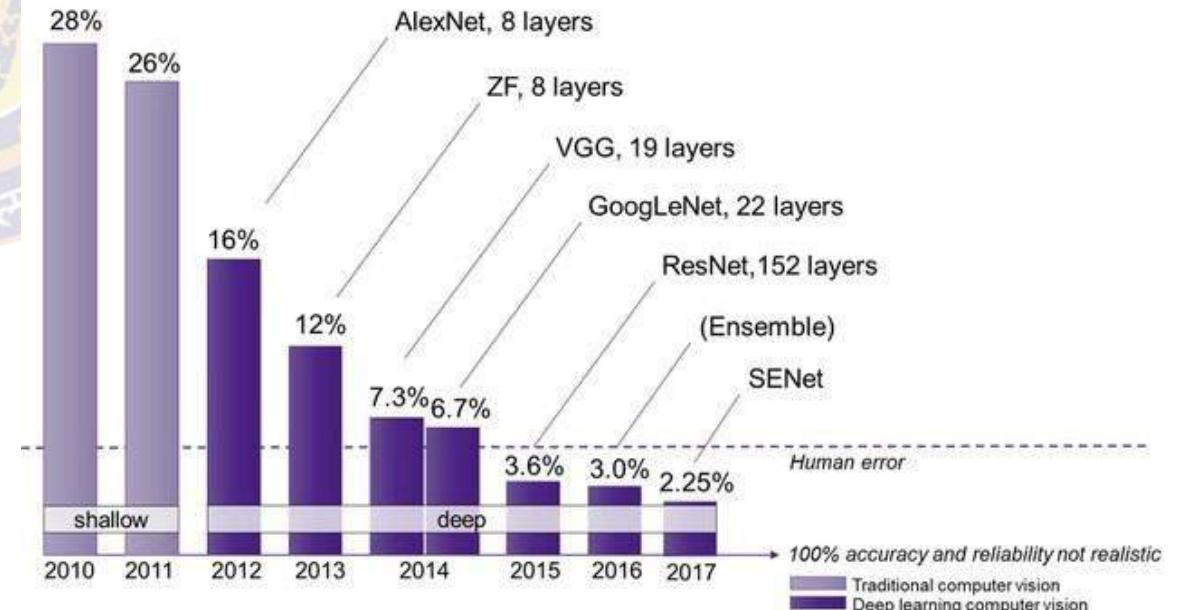
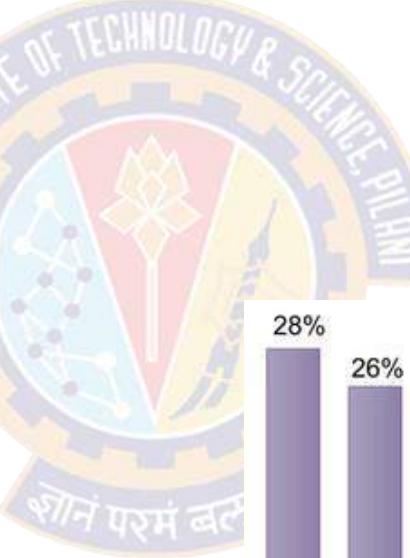
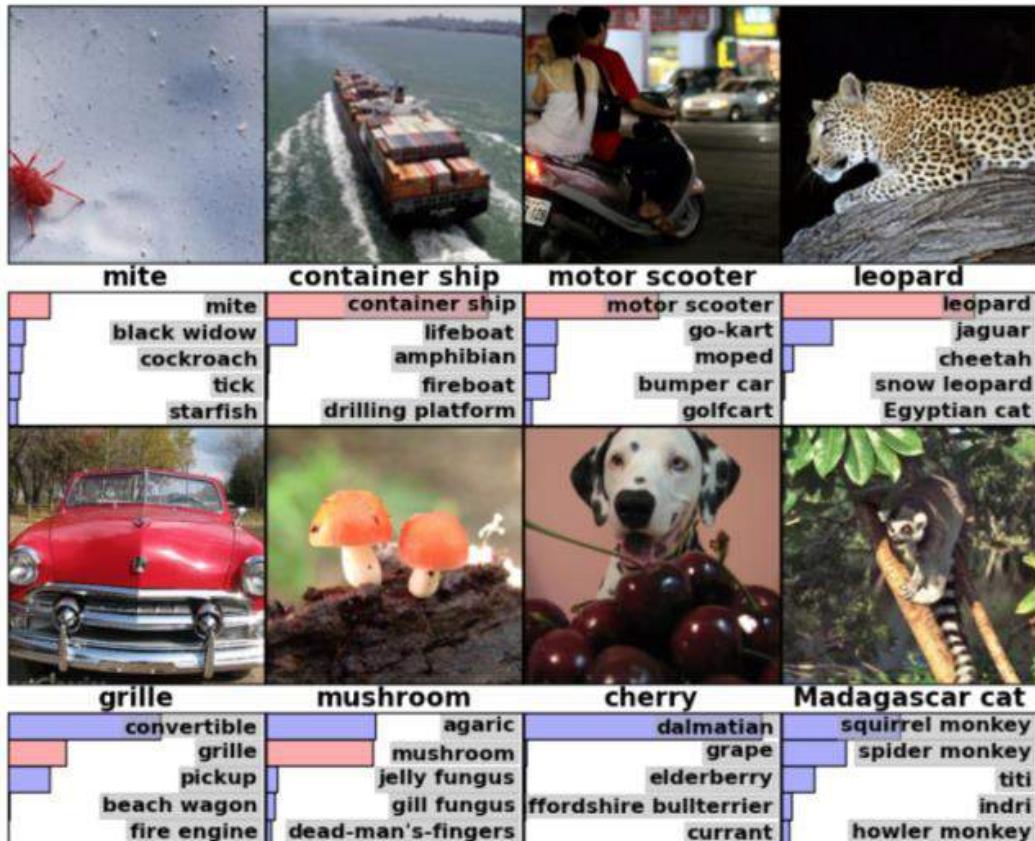
**Dr. Kamlesh Tiwari**

# Deep Learning (DL)

An approach to learn data using neural network

Capable of solving very complex problems

- **ImageNet Challenge** (2011): identify objects, 4 million images in 21841 categories (Challenge 1000 categories)

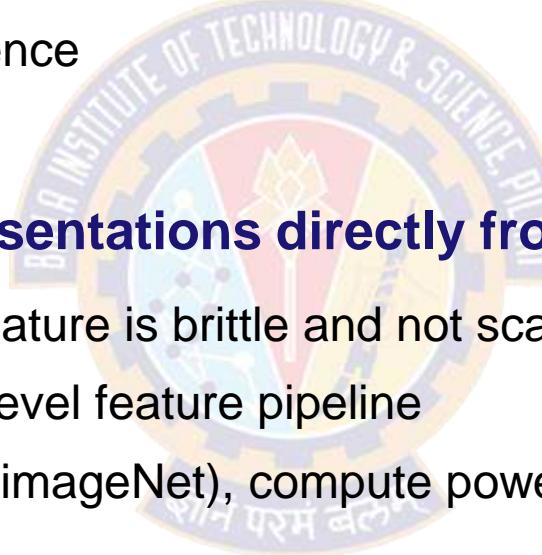


# Deep Learning (DL)

## An approach to learn data using neural network

Capable of solving very complex problems

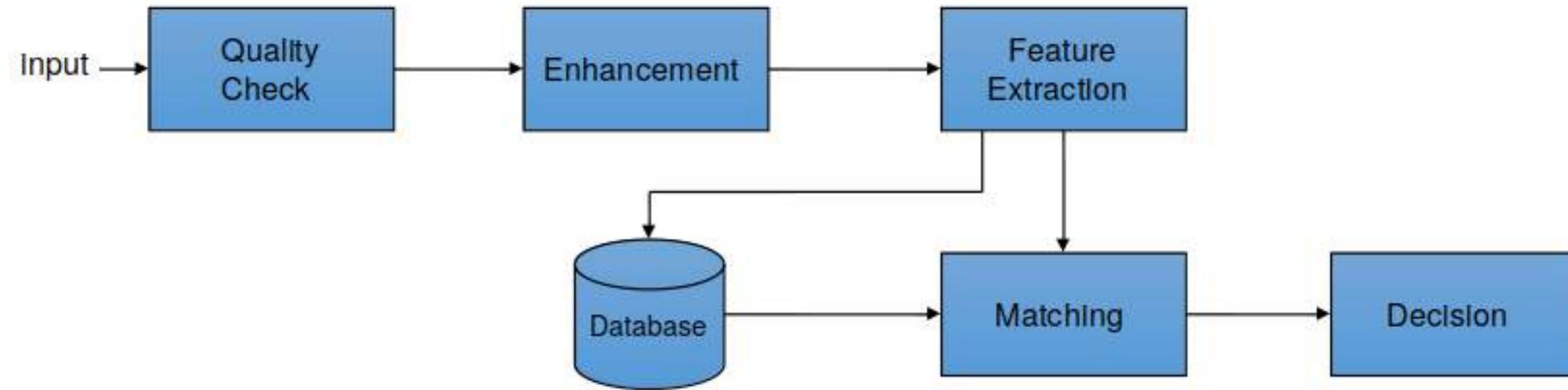
- **ImageNet Challenge** (2011): identify objects in images (millions)
- Generate audio: temporal dependence



## DL learns underlying features/representations directly from data

- Traditional way to hand engineer feature is brittle and not scalable
- DL involves NO low, mid and high level feature pipeline
- Why now? because we have data (imageNet), compute power  
(GPUs) and software (tensorFlow)
- Deep learning is NOT just adding more layers to a neural network

# General ML Pipeline



**Various choices are there at all levels**

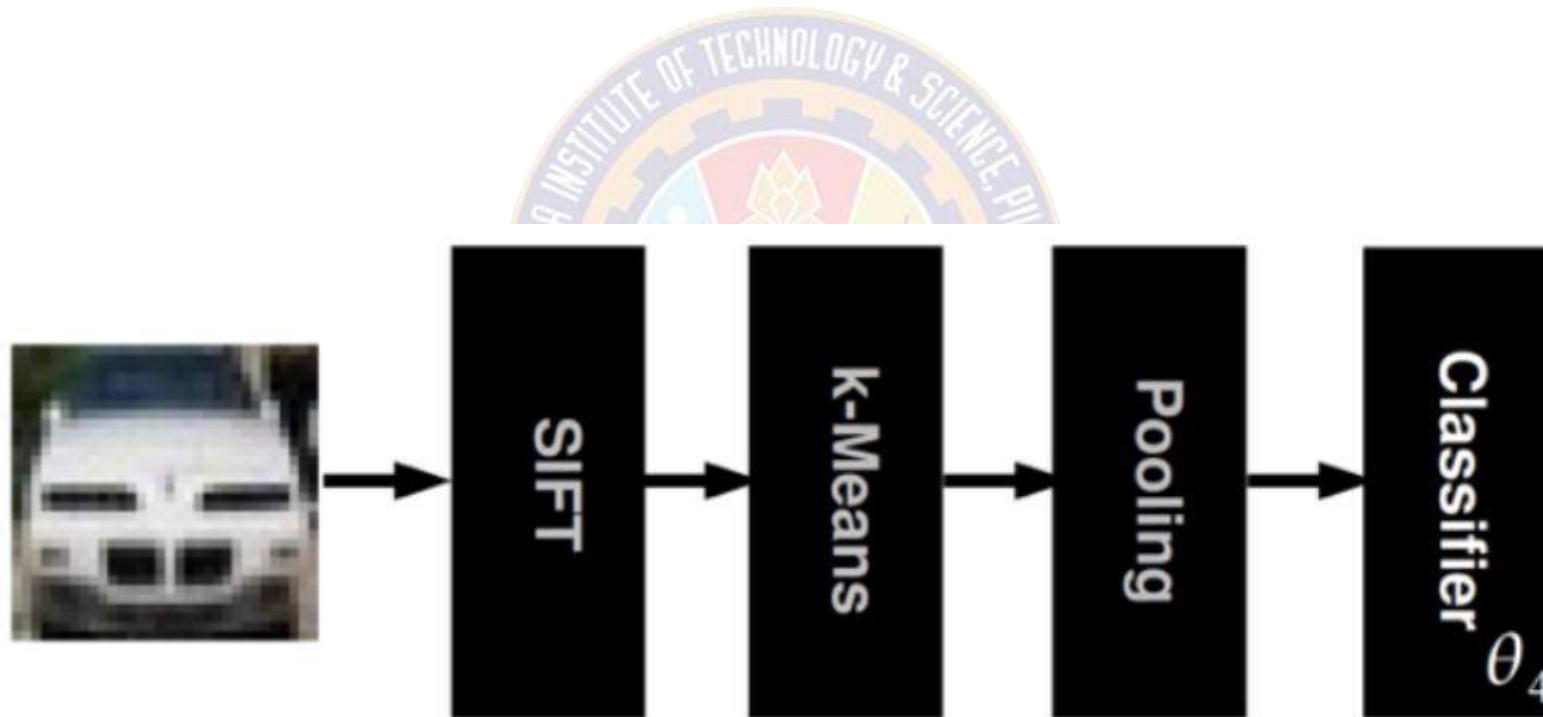
DL provides end-to-end learning

# Deep Learning

- Shallow to Deep: is linear combination to composition

$$\sum_i g_i \rightarrow g_1(g_2(g_3(\dots)))$$

- Why



- Optimization is difficult for a non-convex, non-linear systems
- Freezing some layers makes it shallow



# Thank You!

Next Session: DL



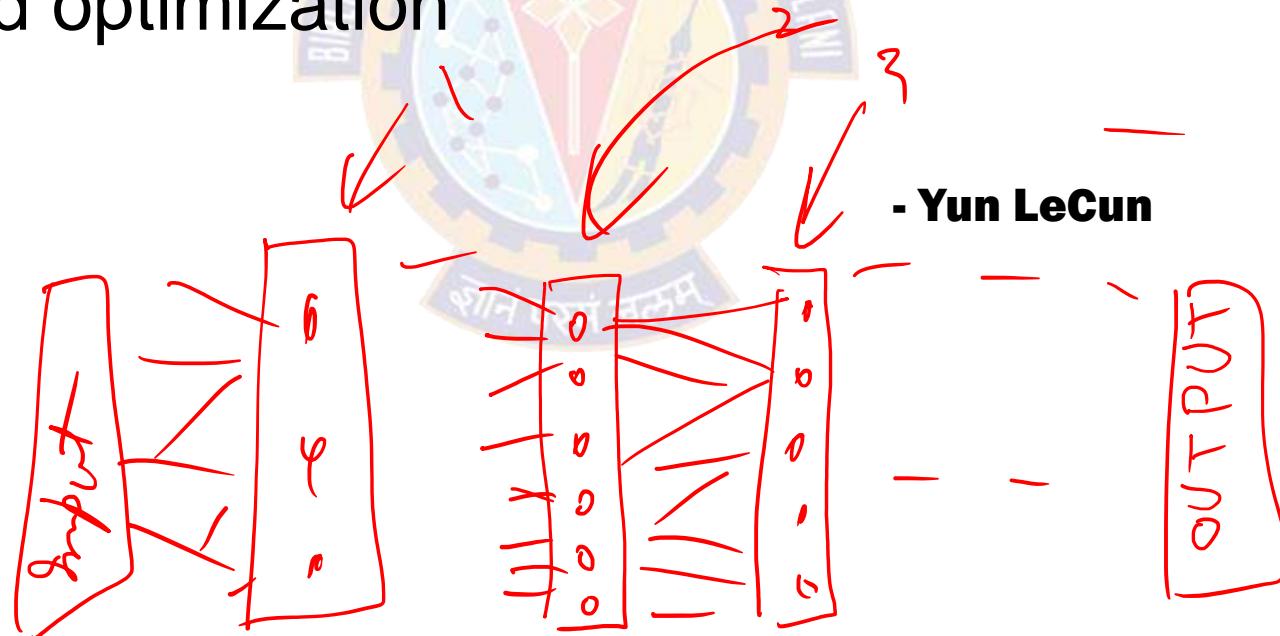
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN

**Dr. Kamlesh Tiwari**

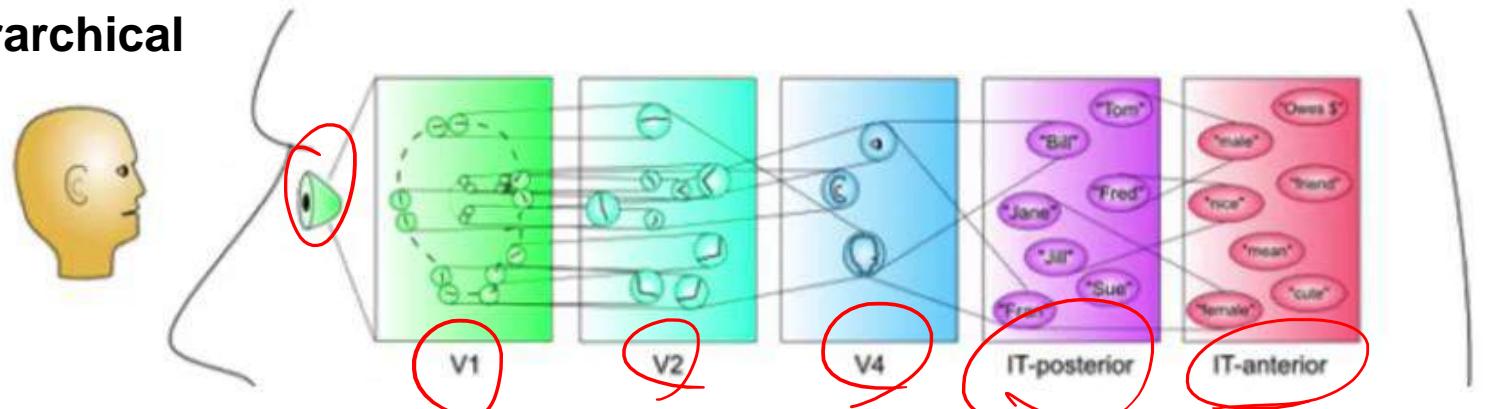
# Deep Learning

DL is constructing network of parameterized functional modules and training them from examples using gradient based optimization

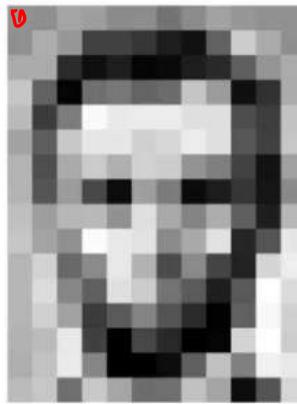


# Visual Cortex is Hierarchical

Visual Cortex is Hierarchical



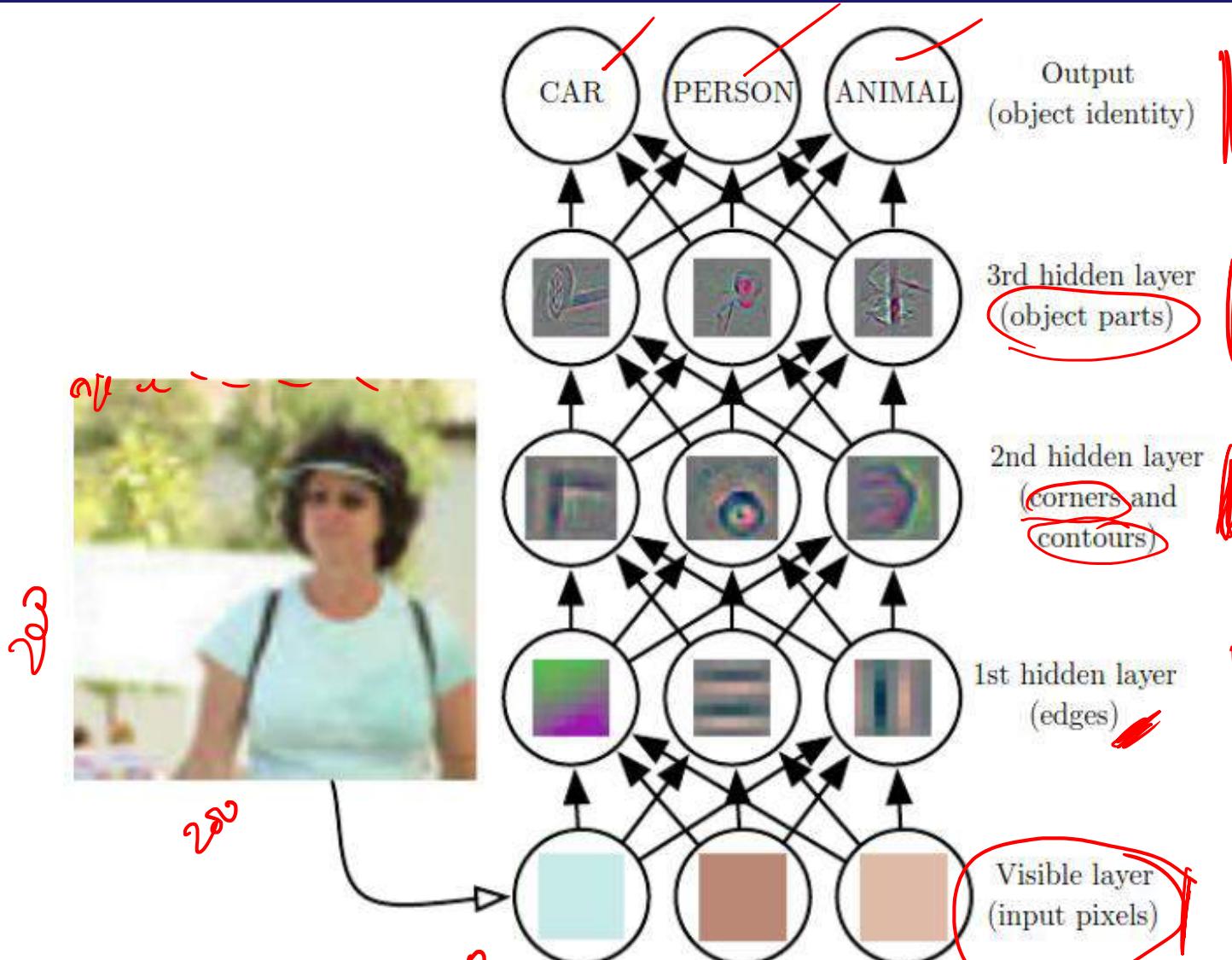
Images are numbers (so determine feature)



A 16x16 grid of numerical values representing the image features, labeled with a red circled 'a'.

187	183	174	164	150	162	129	151	172	161	166	156	188	182	168	179	171	116	210	180	154
188	182	168	154	142	152	129	151	172	161	166	156	189	183	169	177	170	117	211	181	155
189	185	170	154	140	150	126	148	169	158	163	153	190	184	170	178	171	118	212	182	156
190	186	171	155	141	151	127	149	170	159	164	154	191	185	171	179	172	119	213	183	157
191	187	172	156	142	152	128	150	171	160	165	155	192	186	172	178	173	119	214	184	158
192	188	173	157	143	153	129	151	172	161	166	156	193	187	173	179	174	120	215	185	159
193	189	174	158	144	154	130	152	173	162	167	157	194	188	174	180	175	121	216	186	160
194	190	175	159	145	155	131	153	174	163	168	158	195	189	175	181	176	122	217	187	161
195	191	176	160	146	156	132	154	175	164	169	159	196	190	176	182	177	123	218	188	162
196	192	177	161	147	157	133	155	176	165	170	160	197	191	177	183	178	124	219	189	163
197	193	178	162	148	158	134	156	177	166	171	161	198	192	178	184	179	125	220	190	164
198	194	179	163	149	159	135	157	178	167	172	162	199	193	179	185	180	126	221	191	165
199	195	180	164	150	160	136	158	179	168	173	163	200	194	180	186	181	127	222	192	166
200	196	181	165	151	161	137	159	180	169	174	164	201	195	181	187	182	128	223	193	167
201	197	182	166	152	162	138	160	181	170	175	165	202	196	182	188	183	129	224	194	168
202	198	183	167	153	163	139	161	182	171	176	166	203	197	183	190	184	130	225	195	169
203	199	184	168	154	164	140	162	183	172	177	167	204	198	184	191	185	131	226	196	170
204	200	185	169	155	165	141	163	184	173	178	168	205	199	185	192	186	132	227	197	171
205	201	186	170	156	166	142	164	185	174	179	169	206	200	186	193	187	133	228	198	172
206	202	187	171	157	167	143	165	186	175	180	170	207	201	187	194	188	134	229	199	173
207	203	188	172	158	168	144	166	187	176	181	171	208	202	188	195	190	135	230	200	174
208	204	189	173	159	169	145	167	188	177	182	172	209	203	189	196	191	136	231	201	175
209	205	190	174	160	170	146	168	189	178	183	173	210	204	190	197	192	137	232	202	176
210	206	191	175	161	171	147	169	190	179	184	174	211	205	191	198	193	138	233	203	177
211	207	192	176	162	172	148	170	191	180	185	175	212	206	192	199	194	139	234	204	178
212	208	193	177	163	173	149	171	192	181	186	176	213	207	193	200	195	140	235	205	179
213	209	194	178	164	174	150	172	193	182	187	177	214	208	194	201	196	141	236	206	180
214	210	195	179	165	175	151	173	194	183	188	178	215	210	195	202	197	142	237	207	181
215	211	196	180	166	176	152	174	195	184	189	179	216	211	196	203	198	143	238	208	182
216	212	197	181	167	177	153	175	196	185	190	180	217	212	197	204	199	144	239	209	183
217	213	198	182	168	178	154	176	197	186	192	181	218	213	198	205	200	145	240	210	184
218	214	199	183	169	179	155	177	198	187	193	182	219	214	199	206	201	146	241	211	185
219	215	200	184	170	180	156	178	199	188	194	183	220	215	200	207	202	147	242	212	186
220	216	201	185	171	181	157	179	200	189	195	184	221	216	201	208	203	148	243	213	187
221	217	202	186	172	182	158	180	201	190	196	185	222	217	202	209	204	149	244	214	188
222	218	203	187	173	183	159	181	202	191	197	186	223	218	203	210	205	150	245	215	189
223	219	204	188	174	184	160	182	203	192	198	187	224	219	204	211	206	151	246	216	190
224	220	205	189	175	185	161	183	204	193	199	188	225	220	205	212	207	152	247	217	191
225	221	206	190	176	186	162	184	205	194	200	193	226	221	206	213	208	153	248	218	192
226	222	207	191	177	187	163	185	206	195	201	194	227	222	207	214	209	154	249	219	193
227	223	208	192	178	188	164	186	207	196	202	195	228	223	208	215	210	155	250	220	194
228	224	209	193	179	189	165	187	208	197	203	196	229	224	209	216	211	156	251	221	195
229	225	210	194	180	190	166	188	209	198	204	197	230	225	210	217	212	157	252	222	196
230	226	211	195	181	191	167	189	210	199	205	198	231	226	211	218	213	158	253	223	197
231	227	212	196	182	192	168	190	211	200	206	199	232	227	212	219	214	159	254	224	198
232	228	213	197	183	193	169	191	212	201	207	200	233	228	213	220	215	160	255	225	199
233	229	214	198	184	194	170	192	213	202	208	201	234	229	214	221	216	161	256	226	200
234	230	215	199	185	195	171	193	214	203	209	202	235	230	215	222	217	162	257	227	201
235	231	216	200	186	196	172	194	215	204	210	203	236	231	216	223	218	163	258	228	202
236	232	217	201	187	197	173	195	216	205	211	204	237	232	217	224	219	164	259	229	203
237	233	218	202	188	198	174	196	217	206	212	205	238	233	218	225	220	165	260	230	204
238	234	219	203	189	199	175	197	218	207	213	206	239	234	219	226	221	166	261	231	205
239	235	220	204	190	200	176	198	219	208	214	207	240	235	220	227	222	167	262	232	206
240	236	221	205	191	201	177	199	220	209	215	208	241	236	221	228	223	168	263	233	207
241	237	222	206	192	202	178	200	221	210	216	209	242	237	222	229	224	169	264	234	208
242	238	223	207	193	203	179	201	222	211	217	210	243	238	223	230	225	170	265	235	209
243	239	224	208	194	204	180	202	223	212	218	211	244	239	224	231	226	171	266	236	210
244	240	225	209	195	205	181	203	224	213	219	212	245	240	225	232	227	172	267	237	211
245	241	226	210	196	206	182	204	225	214	220	213	246	241	226	233	228	173	268	238	212
246	242	227	211	197	207	183	205													

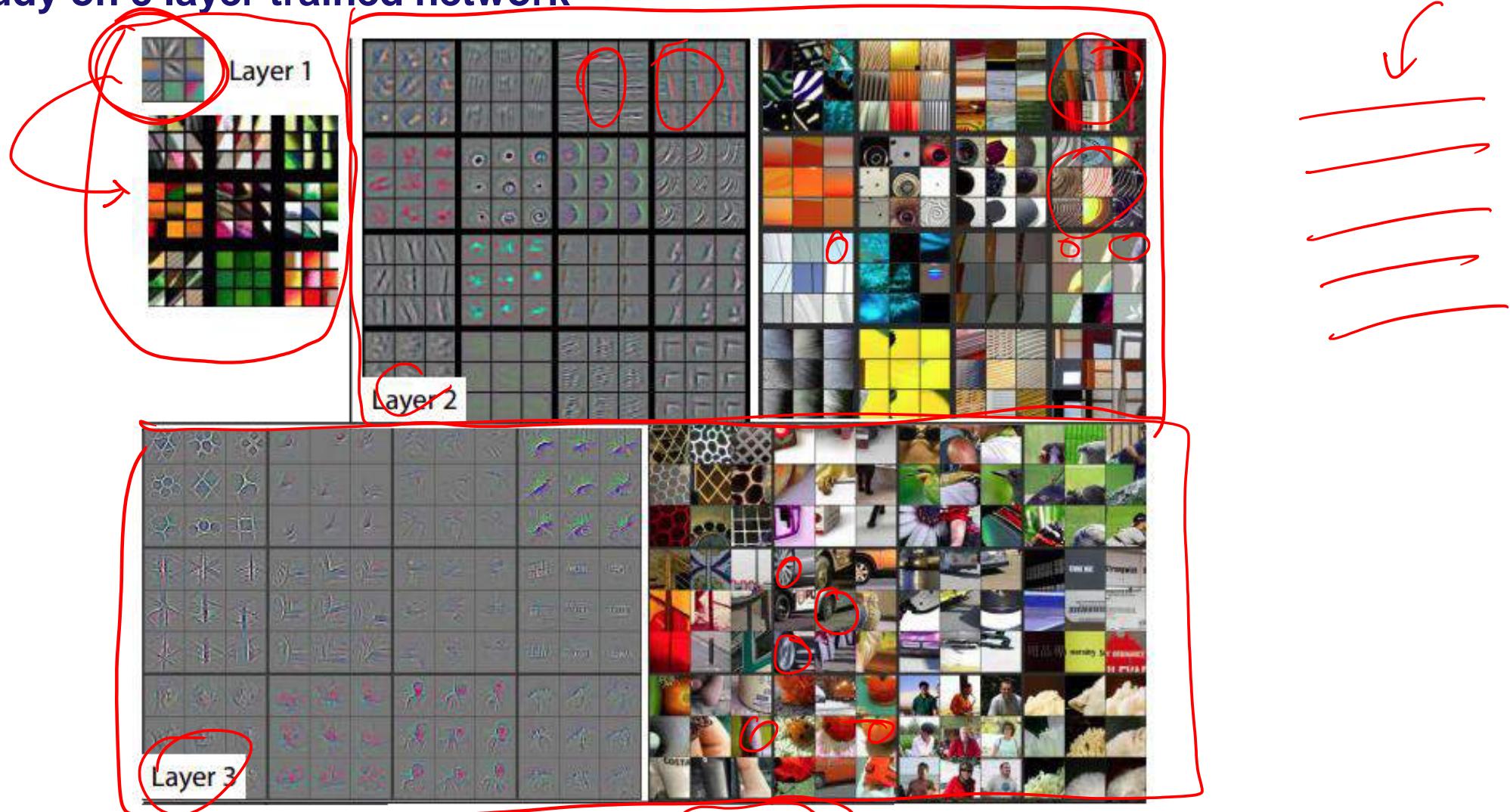
# DL Builds Hierarchy of Features



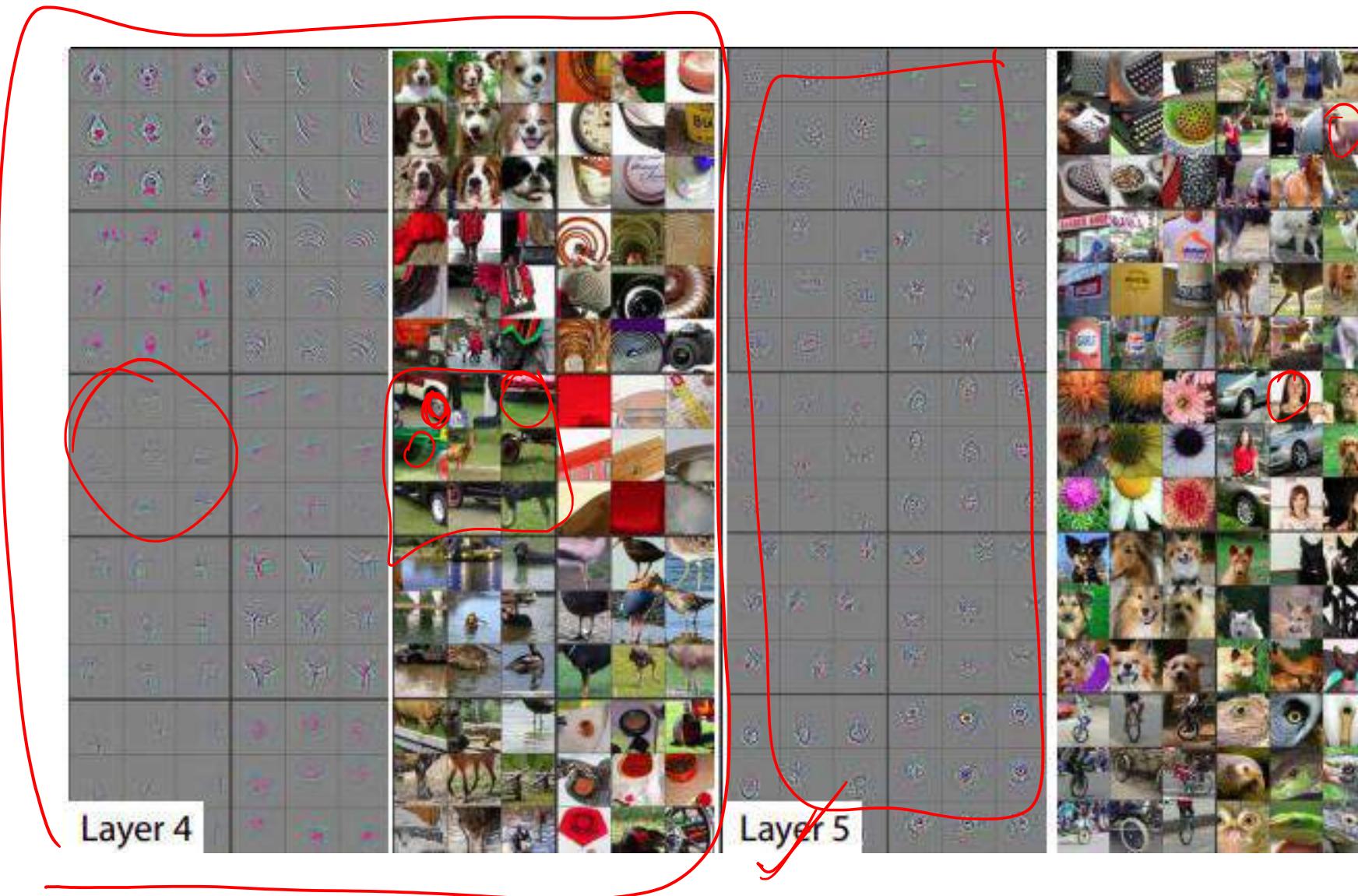
Higher (or deeper) layers represent abstraction of the features

# ECCV2014 - Visualizing and Understanding CNN

A case study on 5 layer trained network



# ECCV2014 - Visualizing and Understanding CNN





# Thank You!

Next Session: Hyperparameter Tuning



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN (hyperparameters)

**Dr. Kamlesh Tiwari**

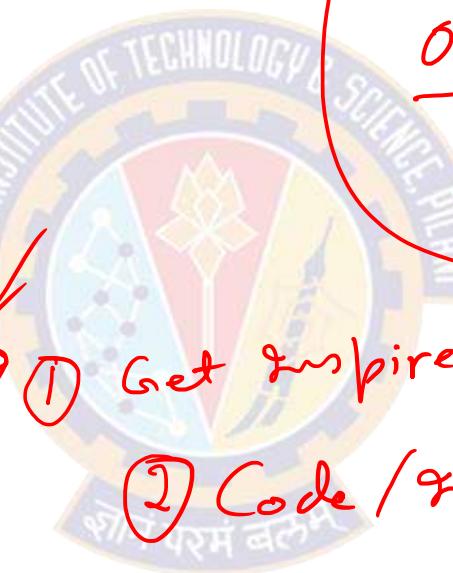
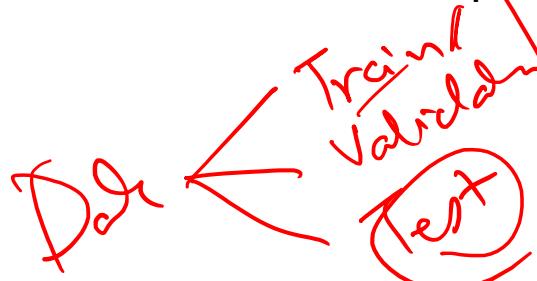
# Hyperparameter Tuning

There are many interesting questions for the network

- How many layers?
- How many units in each layer?
- What should be the learning rate?
- What is right activation function? etc

Difficult to answer in the beginning

It is an iterative process



- ① Get inspired from some thought
- ② Code / implement
- ③ Evaluate / Train - accuracy

Underfitting (Bias) <sup>Int.</sup>  $\rightarrow$  more complex

Overfitting (Varina) Increases training data

Regulenzed  $\leftarrow$  Reduce complexity



# Thank You!

Next Session: Loss Landscape

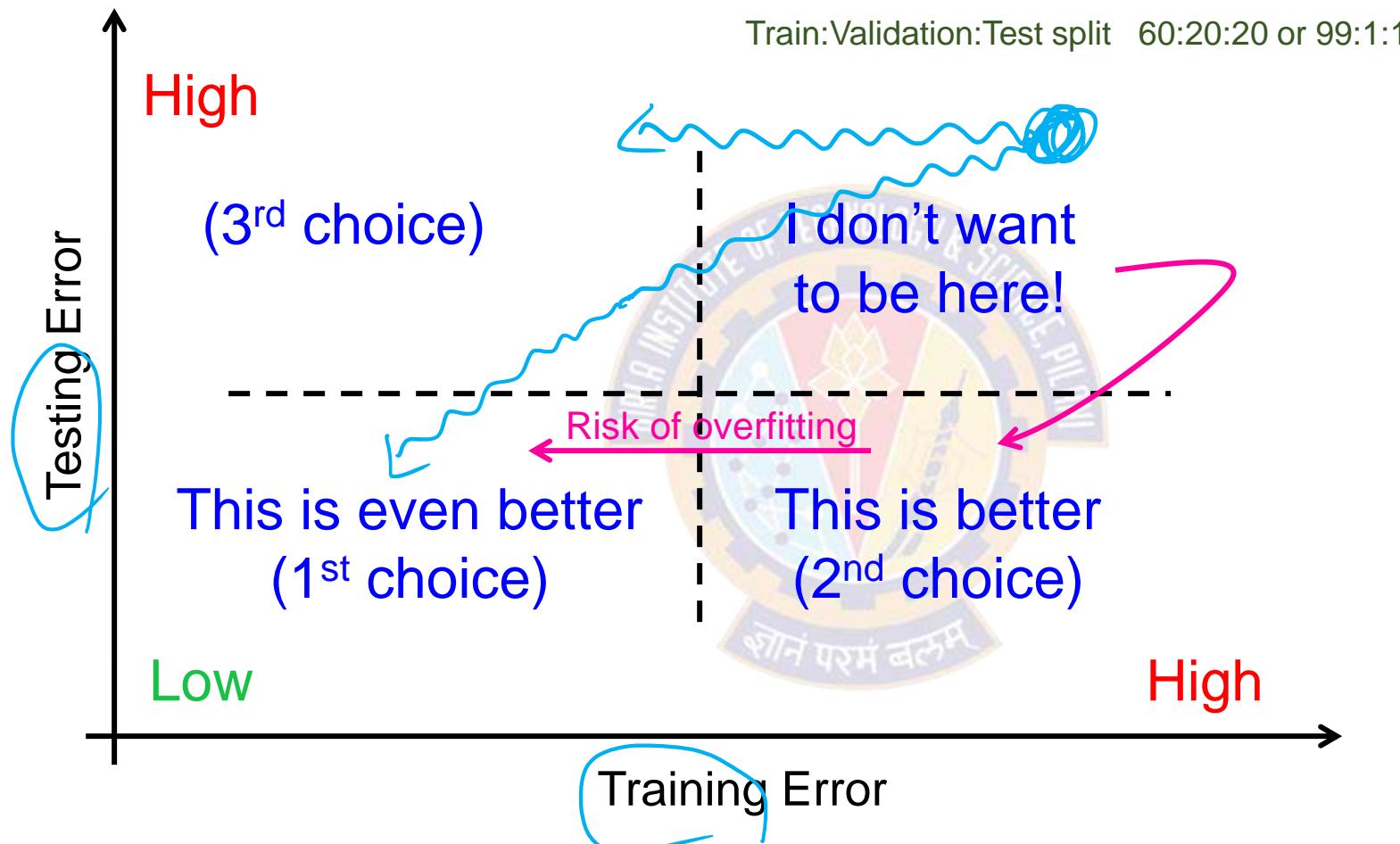


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN

**Dr. Kamlesh Tiwari**

# Which side do you want to be?



Low training error comes with a risk of overfitting (high variance)



# Thank You!

Next Session: Weight Decay



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN Weight Decay

**Dr. Kamlesh Tiwari**

# Regularization for logistic regression

- Optimization minimize loss  $\min_w J(w)$  by adjusting  $w$  where

$$J(w) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$$

- Regularization penalizes the large values of  $w$  by

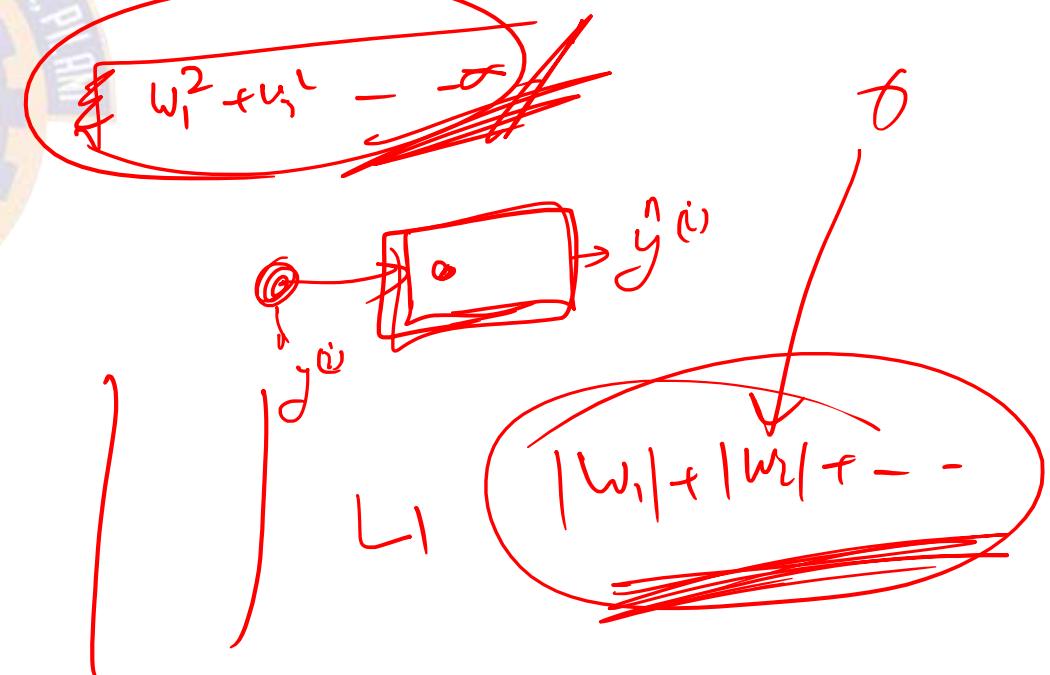
$$J(w) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2$$

where

$$\|w\|_2^2 = \sum_j w_j^2 = w^T w$$

$\lambda$  being regularization parameter

$$\min_w J(w)$$
$$J(w) = \frac{1}{m} \sum_{i=1}^m (y^{(i)}, \hat{y}^{(i)})$$



# Regularization for NN

- As there could be L layers each with their parameters so

$$J(w^{[1]}, w^{[2]}, \dots, w^{[L]}) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|w^{[l]}\|_F^2$$

where **frobenius** norm is

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} w_{ij}^2$$



- How you update the parameter earlier?

Get

$dw^{[l]}$  = (from backpropagation) that is

$$\frac{\partial J}{\partial w^{[l]}}$$

then

$$w^{[l]} = w^{[l]} - \alpha \cdot dw^{[l]}$$

# Regularization for NN (contd...)

- With regularization term  $d\mathbf{w}^{[l]} = (\text{from backpropagation}) + \frac{\lambda}{m} \mathbf{w}^{[l]}$

- Therefore the update is modified to

$$\mathbf{w}^{[l]} = \mathbf{w}^{[l]} - \alpha \cdot (\text{from backpropagation}) + \frac{\lambda}{m} \mathbf{w}^{[l]}$$

Which is

$$\mathbf{w}^{[l]} = \left(1 - \frac{\alpha\lambda}{m}\right) \mathbf{w}^{[l]} - \alpha \cdot (\text{from backpropagation})$$

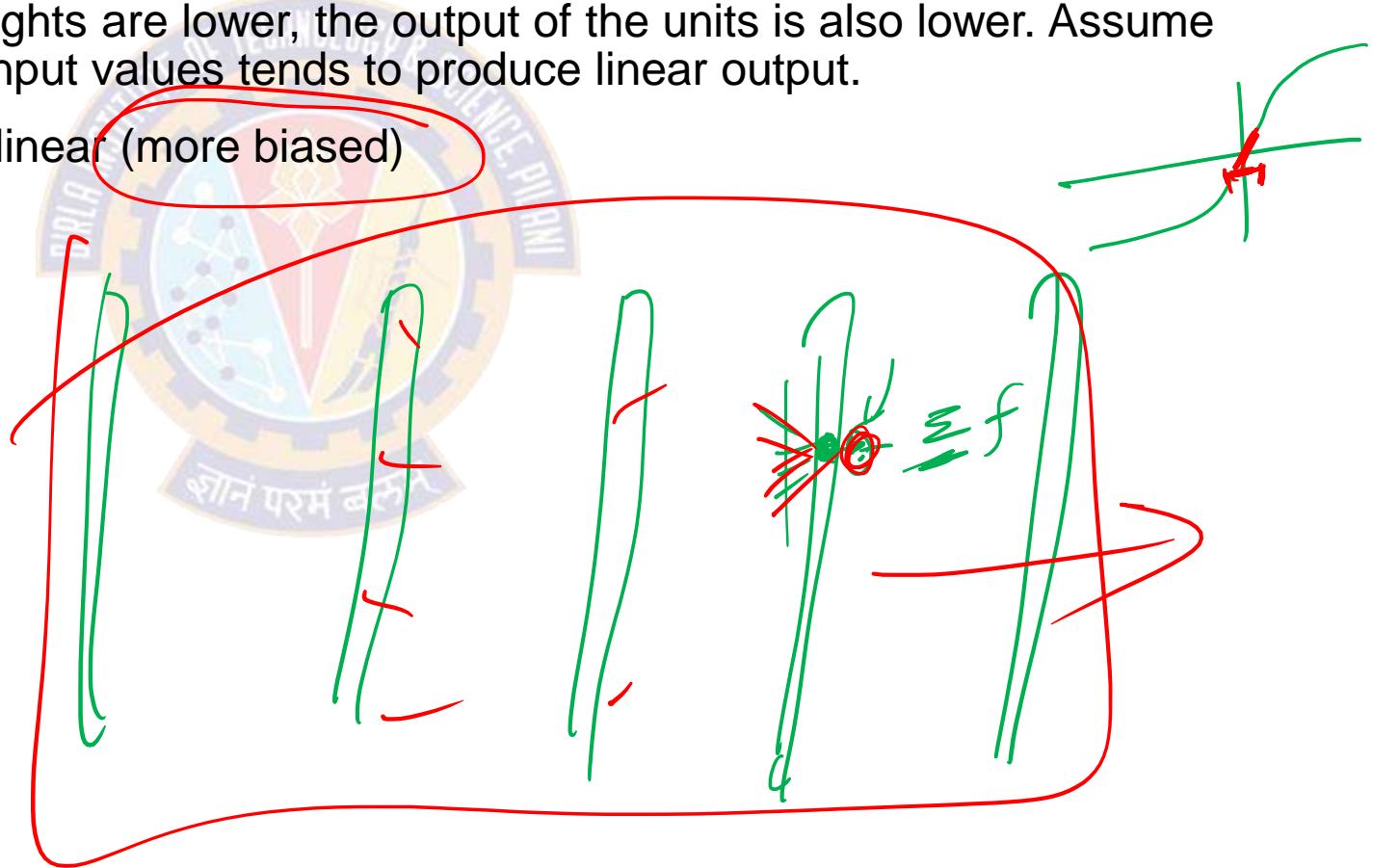
$$\left(\mathbf{w}^{[l]} - \frac{\lambda}{m}\right) \mathbf{w}^{[l]} - \alpha \cdot (\text{from backpropagation})$$

- Due to the  $\left(1 - \frac{\alpha\lambda}{m}\right)$  factor, this update method is also called **weight decay**

Our objective here is to penalize the weight matrices being too large

# Penalize the weight matrices from being too large

- With low weight, the connection get weakened so network has effectively less connections and become simpler.
- Another intuition is that, when weights are lower, the output of the units is also lower. Assume activation function be tanh small input values tends to produce linear output.
- So overall n/w tends to becomes linear (more biased)





# Thank You!

Next Session: Dropout



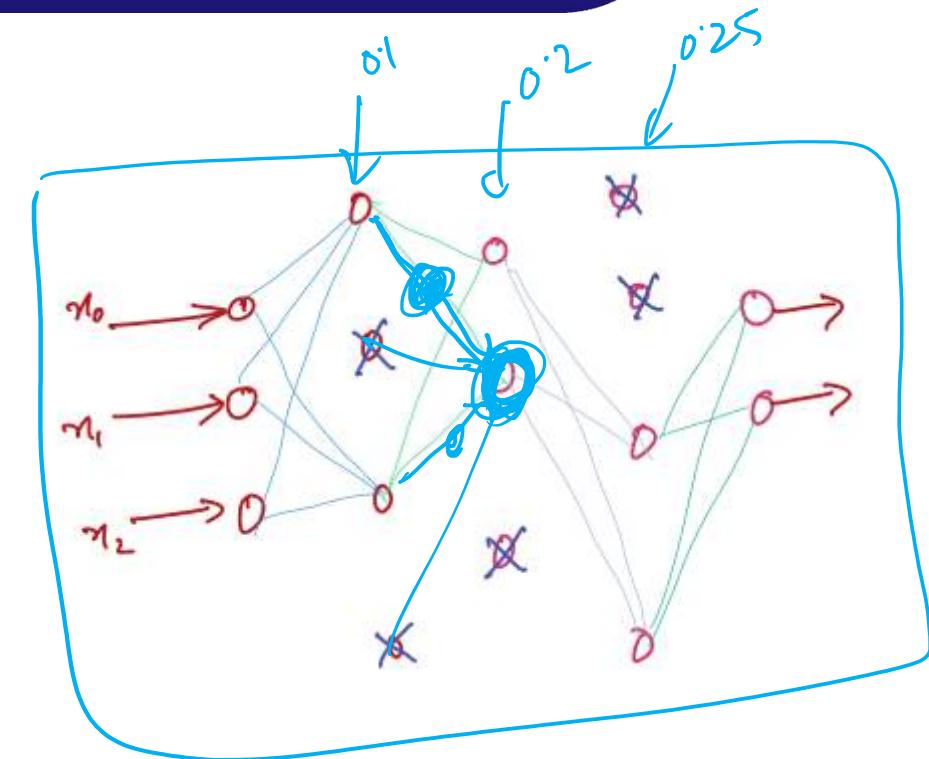
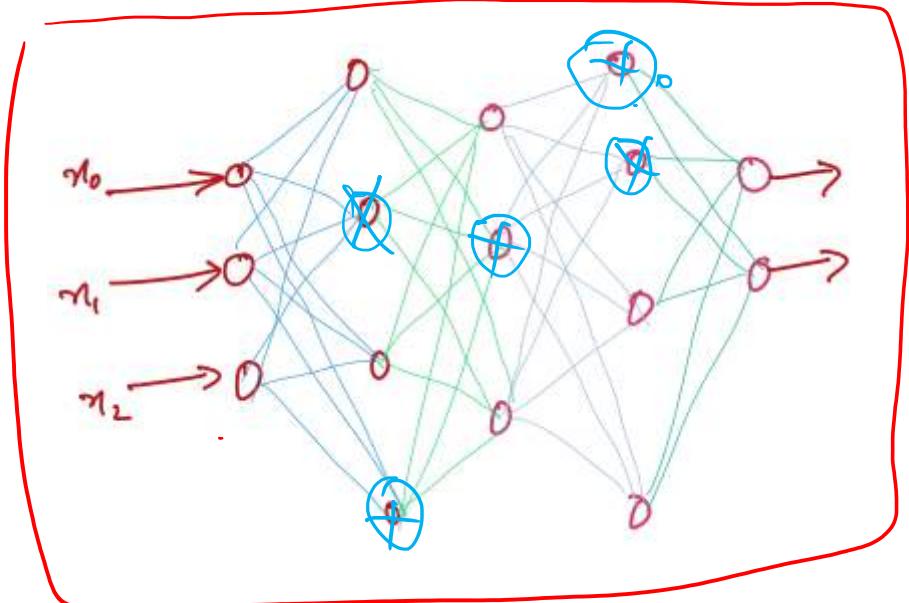
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN (Dropout)

**Dr. Kamlesh Tiwari**

# Dropout Regularization

Randomly shutdown some of the units

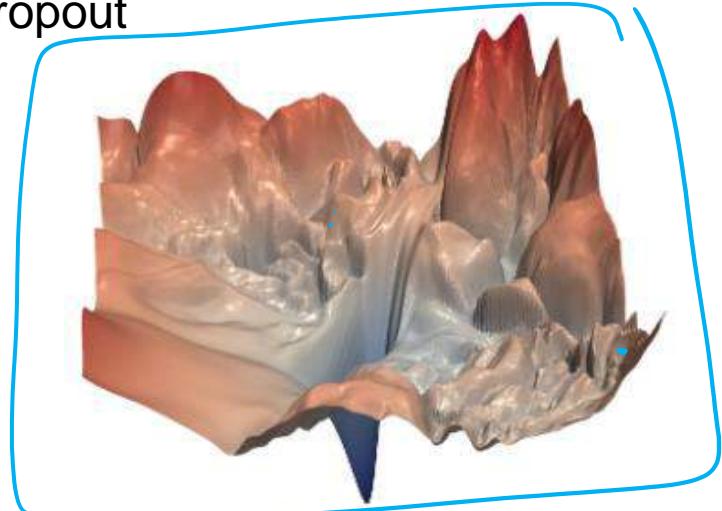


- Drop probabilities for different layers may be different
- Networks can not rely on single connection. So have to give importance to others also
- Issue is that cost function is now not well defined



# Regularization

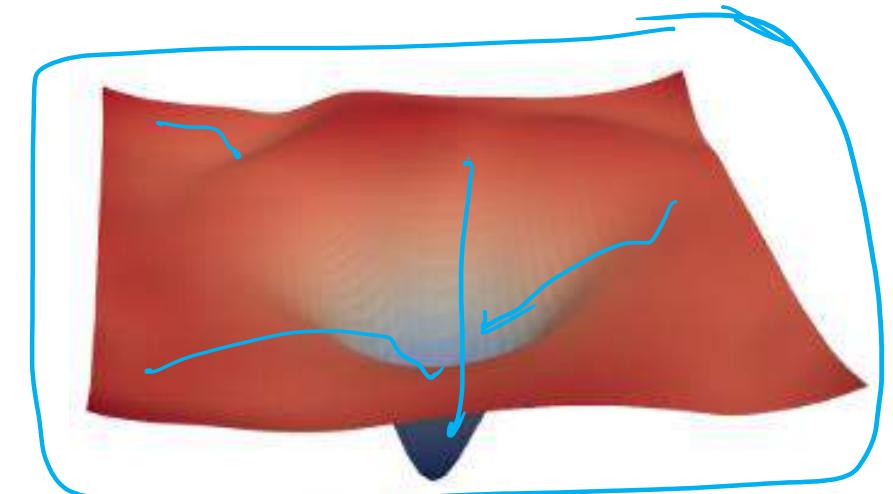
- Dropout



(a) without skip connections



Visualizing the Loss Landscape of Neural Nets, Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein, 2018



(b) with skip connections

- Early stopping





# Thank You!

Next Session: Other Methods



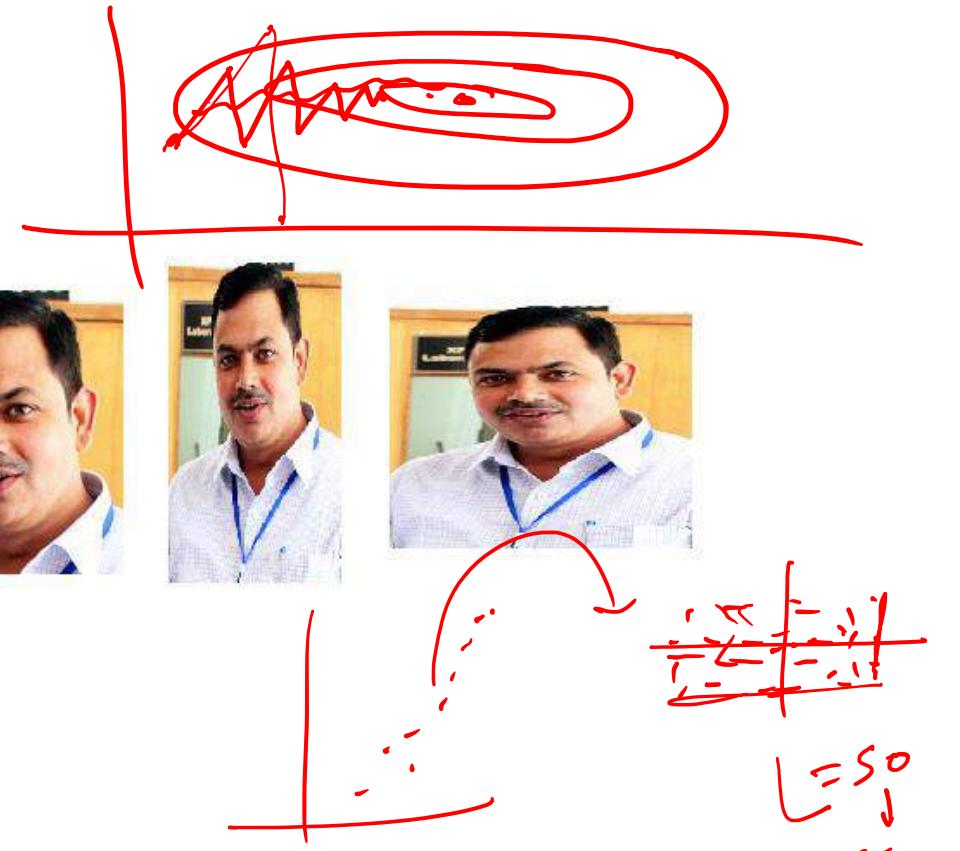
**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN (Other methods)

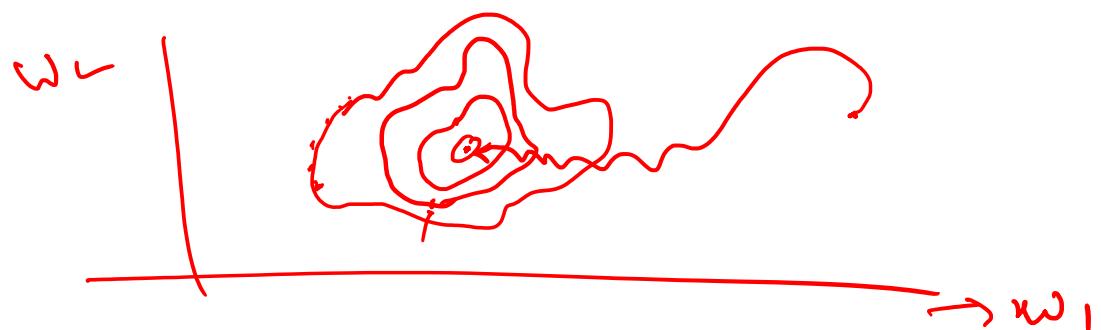
**Dr. Kamlesh Tiwari**

# Other Regularization Methods

- Increase the data by **data augmentation**
- Flip, rotate, scale, translate, deform ....



• Normalize training examples to **speed up your training**





# Thank You!

Next Session: Wt Initialization



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN (Weight Initialization and Training)

**Dr. Kamlesh Tiwari**

# Weight initialization and training

- Exploding and vanishing gradient is an issue
- Set the variance of weights to be  $1/n$
- For ReLu it is better to use variance as  $2/n$
- For tanh use variance as  $\sqrt{1/n}$  called **xavier** initialization
- Adam** optimization, uses momentum and RMSProp simultaneously

$$\beta_1, \beta_2, \epsilon = 0.00001$$

iteration  $t$   
mini batch

$$w, b$$

$$V_{dw}, V_{db}, S_{dw}, S_{db} = 0$$

$$dw, db$$

$$\begin{aligned} V_{dw} &= \beta_1 V_{dw} + (1 - \beta_1) dw \\ V_{db} &= \beta_1 V_{db} + (1 - \beta_1) db \\ S_{dw} &= \beta_2 S_{dw} + (1 - \beta_2) dw^2 \\ S_{db} &= \beta_2 S_{db} + (1 - \beta_2) db^2 \end{aligned}$$

$$\begin{aligned} S_{dw}^{cor} &= \frac{S_{dw}}{(1 + \beta_2^t)} \\ S_{db}^{cor} &= \frac{S_{db}}{(1 + \beta_2^t)} \end{aligned}$$

$$V_{dw}^{cor} = \frac{V_{dw}}{(1 + \beta_1^t)}$$

$$V_{db}^{cor} = \frac{V_{db}}{(1 + \beta_1^t)}$$

$$w = w - \alpha \frac{V_{dw}^{cor}}{\sqrt{S_{dw}^{cor} + \epsilon}}$$

$$b = b - \alpha \frac{V_{db}^{cor}}{\sqrt{S_{db}^{cor} + \epsilon}}$$



# Thank You!

Next Session: (Module-4) CONVOLUTIONAL NEURAL NETWORKS



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Backpropagation

**Dr. Sugata Ghosal**  

---

**Contact Session 2**

# Queries Posted

On what basis the number of deep layer and number of units selected ? Do we keep increasing layer and number of unit based on bias-variance ? Or are there any specific recommendation of trial and error method ?

Out of 4-5 gradient optimization techniques present , is there any specific rule for each technique use case ? Adam optimization has any advantages over other gradient optimization techniques?

Does dropout method of optimization is equivalent to L1(Lasso Regularization) since in L1 we get sparse weights ,keeping some of the weights to zero which is equivalent to dropping those weights so does L1 and dropout is same ?

Can we please take an example to illustrate an iteration of weights updates in back-propagation.

In dropout method, we are not updating the weights associated to some random nodes.

Are we completely ignoring these nodes or we are using previously assigned weights?

Is there a parameter in python neural network libraries which we can use to drop random nodes for dropout method?

What is the role of momentum and RMSprop in Adam optimization?

Please take an example(preferably in python) to show comparison of different optimization technique for a problem.

# Multilayer Networks and Backpropagation

- Single perceptron can only express linear decision surface
- We need units whose output is a nonlinear function of its inputs AND is also differentiable (Use Neuron not Perceptron)

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x})$$

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

- **Backpropagation** algorithm learns the weights for a fixed set of units and interconnections
- It employs **gradient descent** to minimize the error between the network output values and the target values for these outputs
- Let Error function is redefined as

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

# Backpropagation (for 2 layers)

---

**Algorithm:** Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

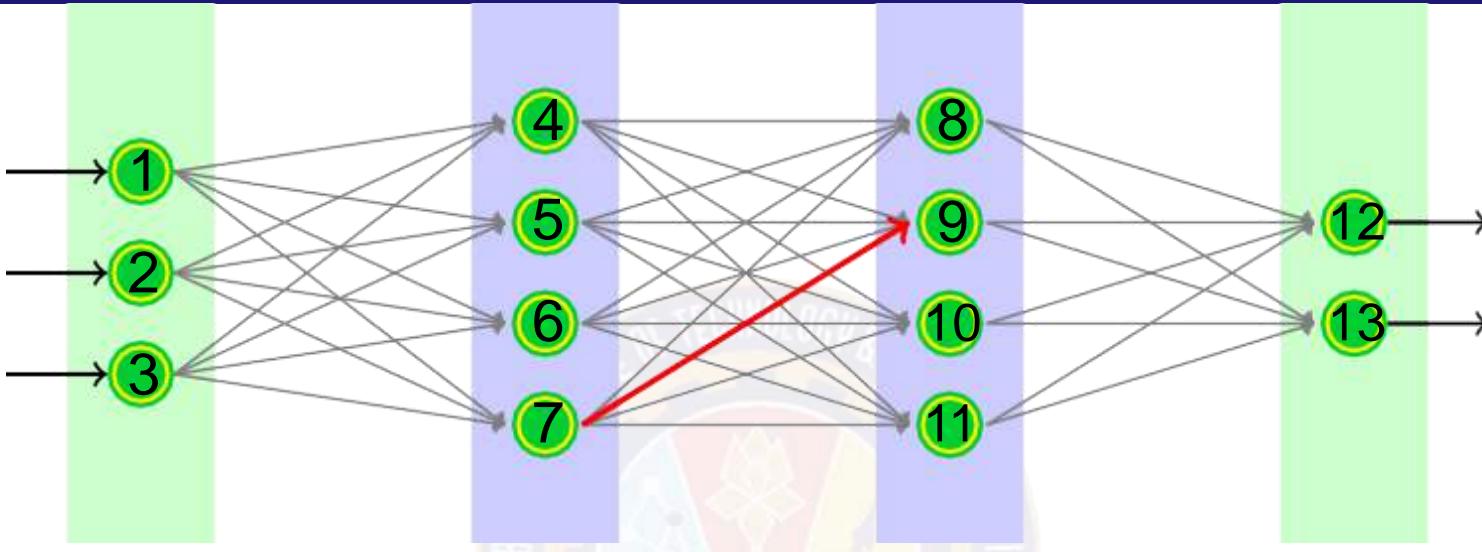
---

1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers  
2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$   
3 **repeat**  
4     **for each**  $\langle \vec{x}, \vec{t} \rangle \in D$  **do**  
5          $o_u = \text{get output from network } \forall \text{unit } u$   
6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**   
7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**   
8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$   
9 **until** converge;

---

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$
- For a single training example  $E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$
- Weight  $w_{ji}$  is updated by adding  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$

# Conventions Over The Network



$x_{ji}$   $i$ th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$  weight associated with  $i$ th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

$t_j$  target output for unit  $j$

*outputs* set of units in final layer ( $\{12, 13\}$  in our case)

*Downstream( $j$ )* units whose immediate input is the output of unit  $j$

We are interested in  $\frac{\partial E_d}{\partial w_{ji}}$  it is  $\frac{\partial E_d}{\partial net_j} \times \frac{\partial net_j}{\partial w_{ji}}$  and therefore,  $\frac{\partial E_d}{\partial net_j} \times x_{ji}$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

Therefore,  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \text{net}_j} \times x_{ji}$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

Therefore,  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \text{net}_j} \times x_{ji} = \boxed{\eta(t_j - o_j)o_j(1 - o_j)x_{ji}}$



# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for hidden units

$$\begin{aligned}\frac{\partial E_d}{\partial \text{net}_j} &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E_d}{\partial \text{net}_k} \times \frac{\partial \text{net}_k}{\partial \text{net}_j} \\&= \sum_{k \in \text{Downstream}(j)} -\delta_k \times \frac{\partial \text{net}_k}{\partial \text{net}_j} \\&= \sum_{k \in \text{Downstream}(j)} -\delta_k \times \frac{\partial \text{net}_k}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j} \\&= \sum_{k \in \text{Downstream}(j)} -\delta_k \times w_{kj} \times o_j(1 - o_j)\end{aligned}$$

$\delta_j$  being  $-\frac{\partial E_d}{\partial \text{net}_j} = o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \delta_k \times w_{kj}$

Therefore,  $\Delta w_{ji} = \eta \delta_j x_{ji} = \boxed{\eta(o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \delta_k \times w_{kj}) x_{ji}}$

# Backpropagation

---

**Algorithm:** Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

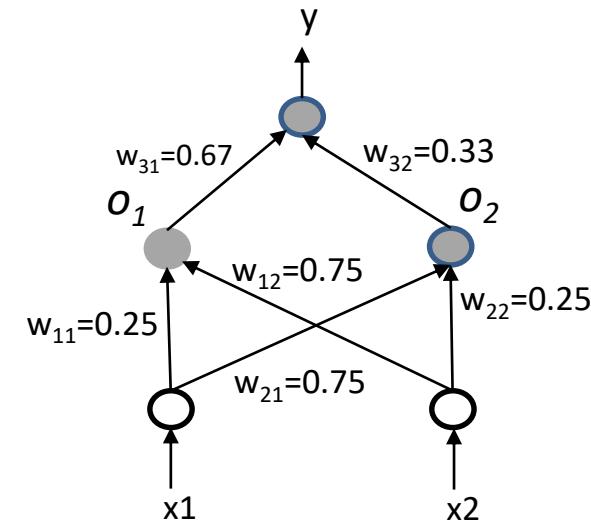
---

- 1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers
  - 2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$
  - 3 **repeat**
  - 4     **for each**  $\langle \vec{x}, \vec{t} \rangle \in D$  **do**
  - 5          $o_u = \text{get output from network } \forall \text{unit } u$
  - 6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**
  - 7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh}\delta_k)$  for all **hidden unit  $h$**
  - 8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$
  - 9 **until** converge;
-

# Parameter Updates in ANN (ReLU activation)

Leaky ReLU activation function that generates output = input, if input  $\geq 0$ , and  $0.1 * \text{input}$  if output  $< 0$ . What will be the weights  $w_{31}$  and  $w_{12}$  in the next iteration with learning rate = 0.1,  $x_1=x_2=1$ , and target output 0?

Assume, squared difference between the actual and target output is used as the loss function, derivative of activation function = 0 at input = 0, and zero bias at all nodes.



. Assume, error  $E = 0.5*(t-y)^2$ . For  $x_1=x_2=1$ , actual output  $y = 1$  and output of hidden nodes are  $o_1=o_2=1$ . Target output  $t = 0$  as specified in the question. Note  $y = w_{31}o_1 + w_{32}o_2$

$$\Delta w_{31} = -\frac{\eta \delta E}{\delta w_{31}} = -\frac{\eta \delta E}{\delta y} * \frac{\delta y}{\delta w_{31}} = \eta(t - y) * o_1 = -0.1.$$

Thus, value of  $w_{31}$  in the next iteration will be

$$w_{31} + \Delta w_{31} = 0.67 - 0.1 = 0.57.$$

$$\Delta w_{12} = -\frac{\eta \delta E}{\delta w_{12}} = -\frac{\eta \delta E}{\delta y} * \frac{\delta y}{\delta w_{12}} = -0.1 * \frac{\delta y}{\delta o_1} * \frac{\delta o_1}{\delta w_{12}} = -0.1 * w_{31} * x_2 = -0.067.$$

Thus, value of  $w_{12}$  in the next iteration will be

$$w_{12} + \Delta w_{12} = 0.75 - 0.067 = 0.683.$$



# Backpropagation

- **Adding Momentum:** weight update during  $n^{th}$  iteration depend partially on the update that occurred during the  $(n - 1)^{th}$  iteration

$$\Delta w_{jl}(n) = \eta \delta_j x_{jl} + \alpha \Delta w_{jl}(n-1)$$

- **Learning in arbitrary acyclic network:** for feed-forward networks of arbitrary depth,  $\delta_r$  value for a unit in hidden layer is determined as

$$\delta_r = o_r(1 - o_r) \sum_{s \in \text{Downstream}(r)} w_{sr} \times \delta_s$$

# Deep MLP: Code Snippet

## Specify Data

```
# Normalize the data
# 60000 input images are in the train set.
# 10000 input images are in the test set.

Xtrain = Xtrain.reshape((60000, 28*28))      # reshape the input set to size 28*28.
Xtrain = Xtrain.astype('float32')/255          # normalize to grayscale; set datatype as float32

Xtest = Xtest.reshape((10000, 28*28))        # reshape the input set to size 28*28.
Xtest = Xtest.astype('float32')/255            # normalize to grayscale; set datatype as float32

Ytrain = tf.keras.utils.to_categorical(Ytrain)
Ytest = tf.keras.utils.to_categorical(Ytest)
```

## Specify Network Architecture

```
# Layer 1 = input layer
# specify the input size in the first layer.

dnnModel.add(layers.Dense(50, activation='relu', input_shape= (28*28,)))

# Layer 2 = hidden layer
dnnModel.add(layers.Dense(60, activation='relu'))

# Layer 3 = hidden layer
dnnModel.add(layers.Dense(30, activation='relu'))

# Layer 4 = output layer
dnnModel.add(layers.Dense(10, activation='softmax'))

dnnModel.summary()
```

# Deep MLP: Code Snippet

## Configure Training Parameters

```
# Configure the model for training, by using appropriate optimizers and regularizations
# Available optimizer: adam, rmsprop, adagrad, sgd
# loss: objective that the model will try to minimize.
# Available loss: categorical_crossentropy, binary_crossentropy, mean_squared_error
# metrics: List of metrics to be evaluated by the model during training and testing.

dnnModel.compile( optimizer = 'adam', loss = 'categorical_crossentropy', metrics=['accuracy'] )
```

## Train the Model

```
# train the model

h = dnnModel.fit( Xtrain, Ytrain, epochs=25, batch_size=64)
```

# Deep MLP: L2 Regularization

## Specify the Architecture

```
# Layer 1 = input layer
# specify the input size for in the first layer.

dnnModel.add(layers.Dense(50, activation='relu', input_shape= (28*28,)))

# Layer 2 = hidden Layer with Regularizers
dnnModel.add(layers.Dense(60, activation='relu', kernel_regularizer=regularizers.l2(0.01),
                        activity_regularizer=regularizers.l2(0.01)) )

# Layer 3 = hidden Layer
dnnModel.add(layers.Dense(30, activation='relu'))

# Layer 4 = output Layer
dnnModel.add(layers.Dense(10, activation='softmax'))

dnnModel.summary()
```

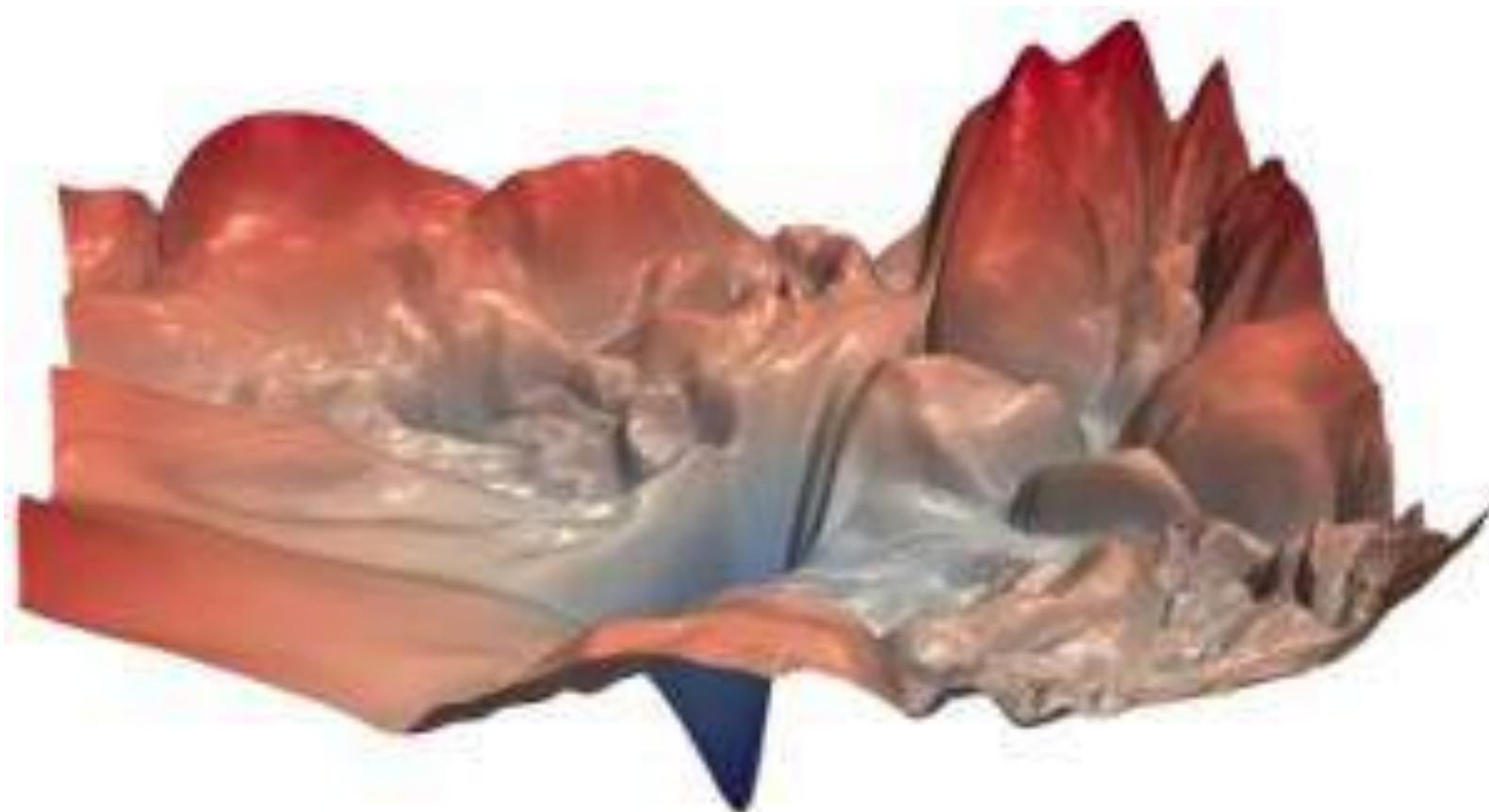
# Deep MLP: Dropout

## Specify the Architecture

```
# Layer 1 = input layer  
# specify the input size for in the first layer.  
  
dnnModel.add(layers.Dense(50, activation='relu', input_shape= (28*28,)))  
  
# Layer 2 = hidden layer |  
dnnModel.add(layers.Dense(60, activation='relu'))  
  
# Add dropout of 50% to Layer 2  
dnnModel.add(layers.Dropout(0.5))  
  
# Layer 3 = hidden layer  
dnnModel.add(layers.Dense(30, activation='relu'))  
  
# Add dropout of 50% to Layer 3  
dnnModel.add(layers.Dropout(0.5))  
  
# Layer 4 = output layer  
dnnModel.add(layers.Dense(10, activation='softmax'))  
  
dnnModel.summary()
```

# Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error

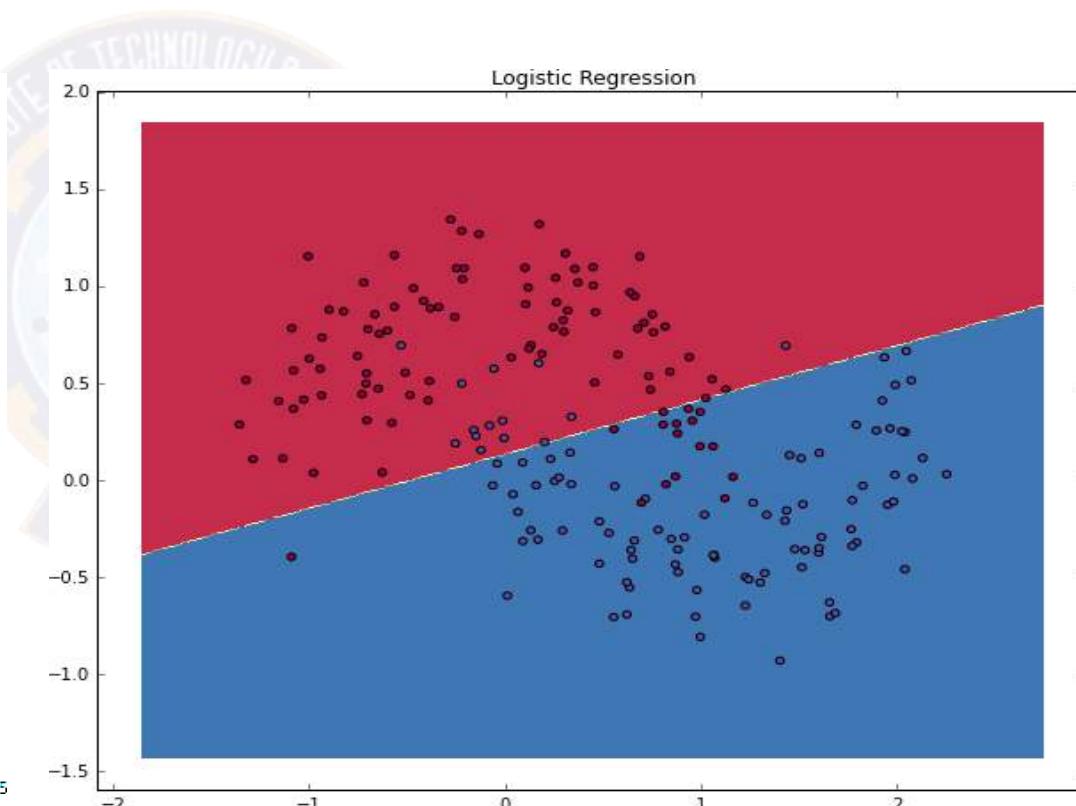
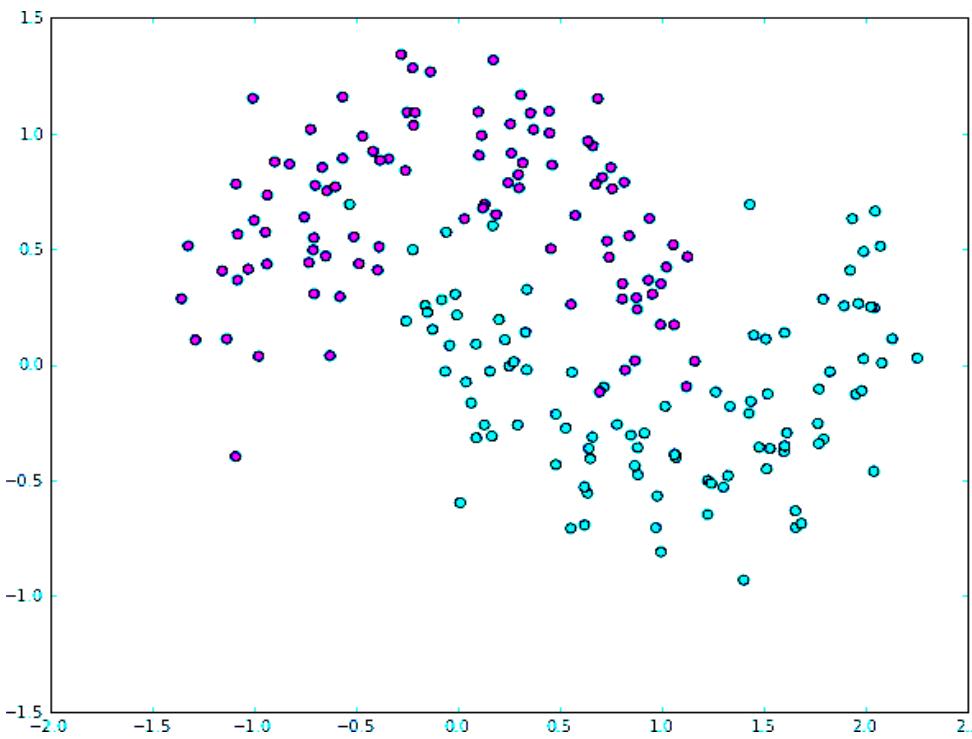


# Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error
- No methods are known to predict with certainty when local minima will cause difficulties
- Suggested to use momentum, true gradient descent or multiple networks (initialized with different random weights)
- Any boolean function can precisely be represented by some network having only **two** layers of units (Cybenko 1989; Hornik et al. 1989)

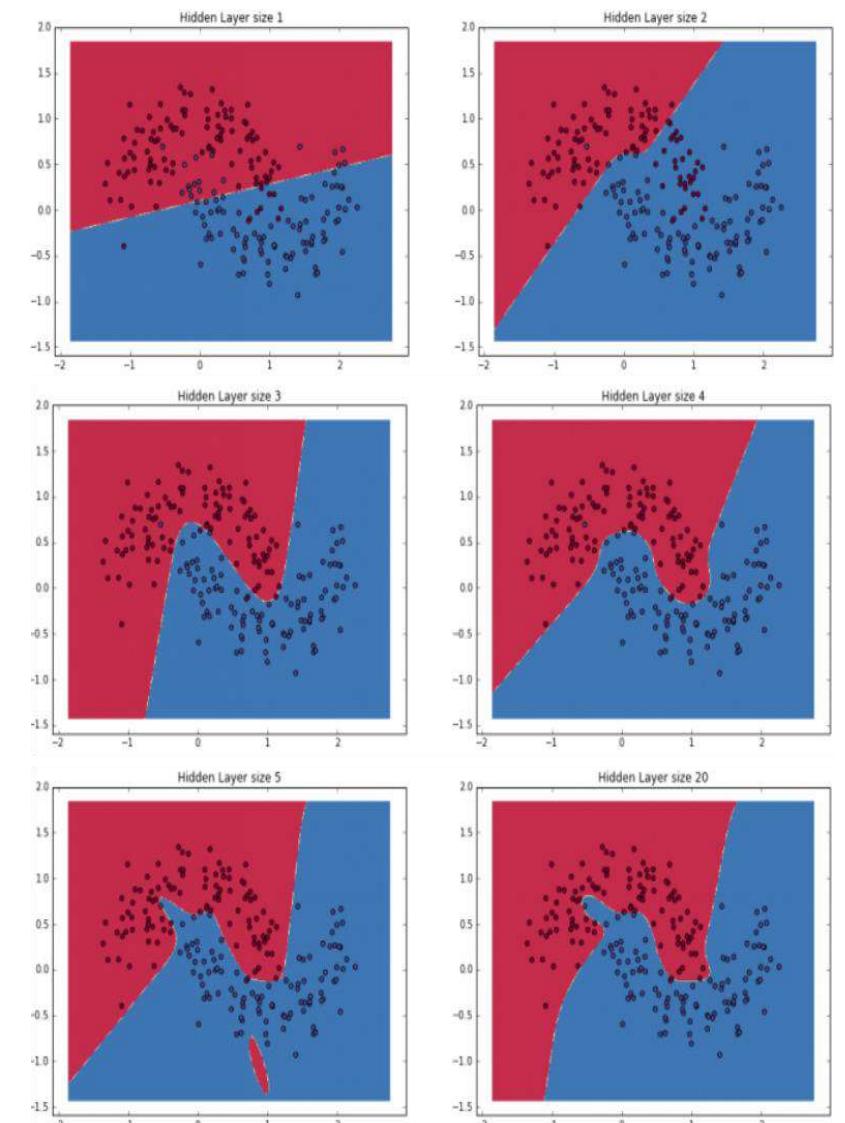
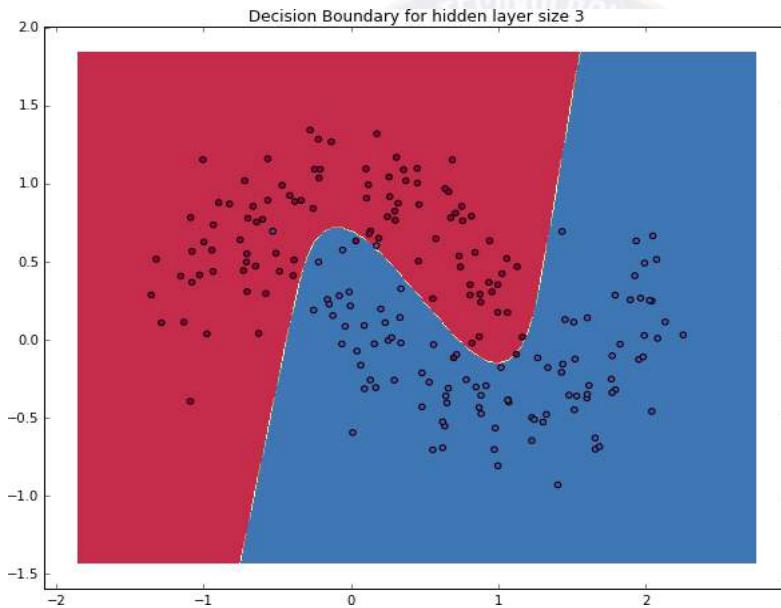
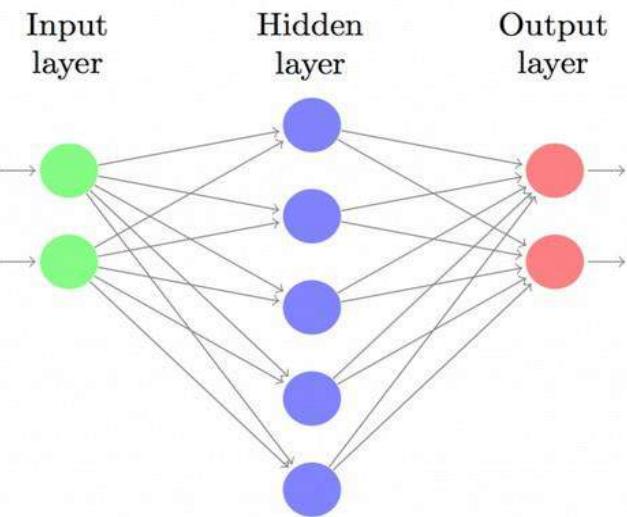
# Coding Example 1

Consider two class data  
Logistic Regression  
Neural Network  
Decision Boundary  
Bigger hidden layer



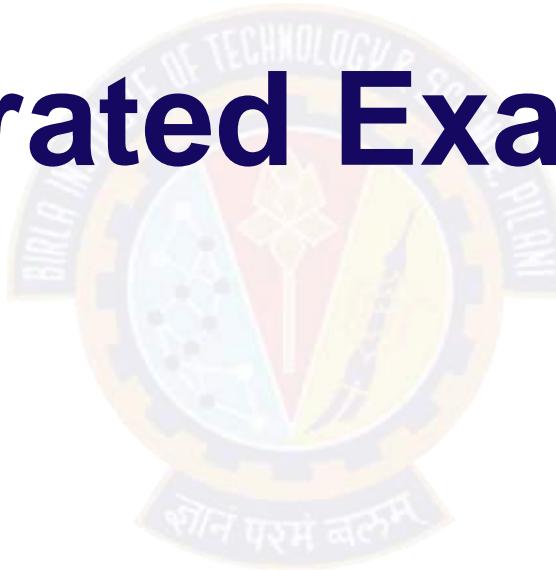
<sup>1</sup><https://github.com/dennybritz/nn-from-scratch/blob/master/nn-from-scratch.ipynb>

# Coding Example 1



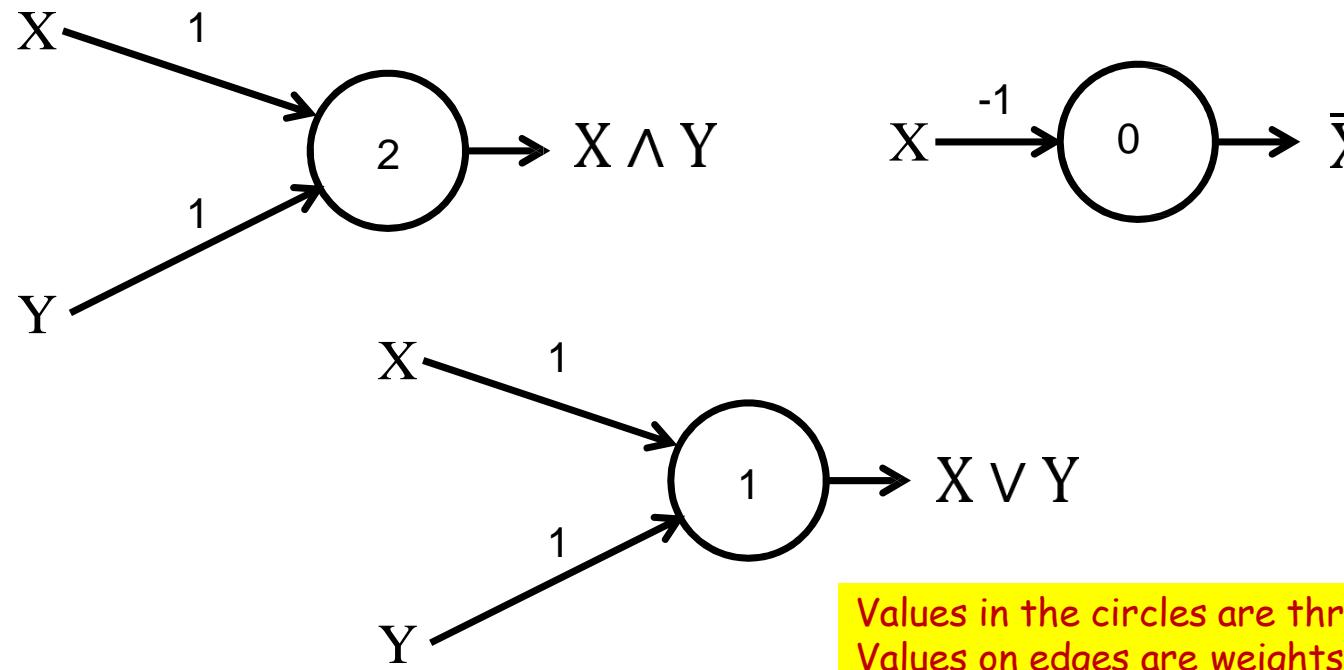
<sup>1</sup><https://github.com/dennybritz/nn-from-scratch/blob/master/nn-from-scratch.ipynb>

# Illustrated Examples



- Multi-layer Perceptrons as universal Boolean functions
- MLPs as universal classifiers

# The perceptron as a Boolean gate



- A perceptron can model any simple binary Boolean gate
- Output = 1 if total input  $\geq$  threshold

# $x_1$ OR $x_2$ with Perceptron (Step Threshold)

Perceptron with hard/step threshold

Output = 0 if input  $w_1*x_1 + w_2*x_2 < \text{threshold } T$   
= 1 if input  $\geq T$

From OR truth table,

$w_1*0 + w_2*0 < T$  implies  $T > 0$

$w_1*1+w_2*0 \geq T$  implies  $w_1 \geq T$

$w_1*0+w_2*1 \geq T$  implies  $w_2 \geq T$

$w_1*1+w_2*1 \geq T$  implies  $w_1+w_2 \geq T$

Choose,  $T=1$  (you can choose any  $T > 0$ )

Then  $w_1=1$  (you can choose any value  $\geq T$ )

Similarly choose  $w_2=1$  (you can choose any value  $\geq T$ )

$w_1+w_2 \geq T$  is automatically satisfied for  $w_1=w_2=1$  and  $T=1$

X1 AND x2 Truth Table

x1	x2	Output
0	0	0
1	0	1
0	1	1
1	1	1

Thus, AND can be realized with  $w_1=1$ ,  $w_2=1$ ,  $T=1$ . Important to note that other values of  $w_1$ ,  $w_2$ , and  $T$  can also implement OR, as long as the about 4 inequalities are satisfied

# $x_1$ AND $x_2$ with Perceptron (Step Threshold)

Perceptron with hard/step threshold

Output = 0 if input  $w_1 \cdot x_1 + w_2 \cdot x_2 < \text{threshold } T$   
= 1 if input  $\geq T$

From OR truth table,

$w_1 \cdot 0 + w_2 \cdot 0 < T$  implies  $T > 0$

$w_1 \cdot 1 + w_2 \cdot 0 < T$  implies  $w_1 < T$

$w_1 \cdot 0 + w_2 \cdot 1 < T$  implies  $w_2 < T$

$w_1 \cdot 1 + w_2 \cdot 1 \geq T$  implies  $w_1 + w_2 \geq T$

Choose,  $T = 2$  (you can choose any  $T > 0$ )

Then  $w_1 = 1$  (you can choose any value  $< T$ )

Similarly choose  $w_2 = 1$  (you can choose any value  $< T$ )

$w_1 + w_2 \geq T$  is automatically satisfied for  $w_1 = w_2 = 1$  and  $T = 2$

X1 AND x2 Truth Table

x1	x2	Output
0	0	0
1	0	0
0	1	0
1	1	1

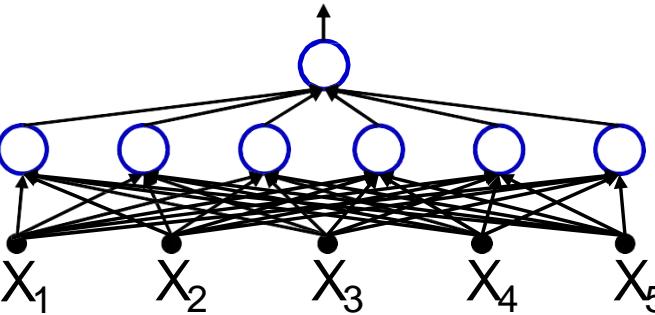
Thus, AND can be realized with  $w_1 = 1$ ,  $w_2 = 1$ ,  $T = 2$ . Important to note that other values of  $w_1$ ,  $w_2$ , and  $T$  can also implement AND, as long as the about 4 inequalities are satisfied

# How many layers for a Boolean MLP?

Truth Table					
$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

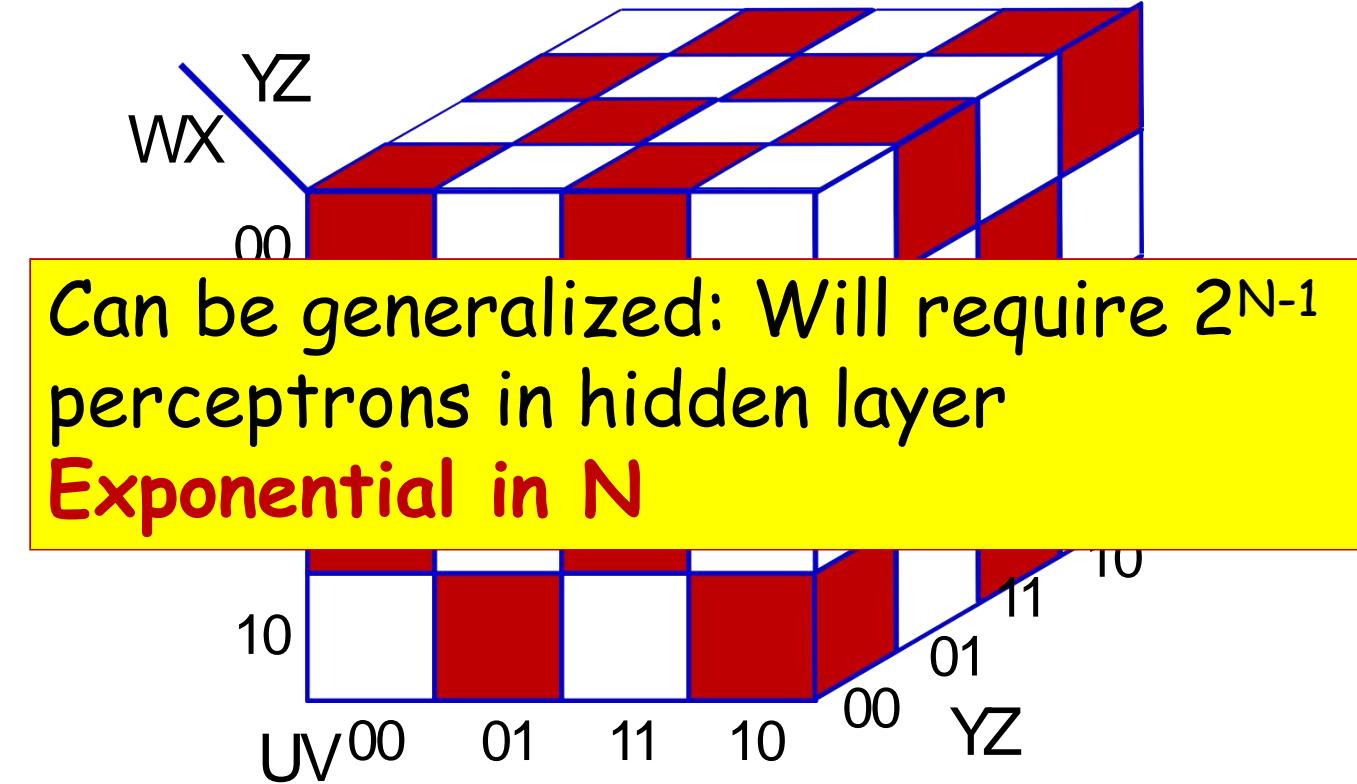
$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Any truth table can be expressed in this manner!
- A one-hidden-layer MLP is a Universal Boolean Function

But what is the largest number of perceptrons required in the single hidden layer for an N-input-variable function?

# Width of a single-layer Boolean MLP



- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function

# Width of a single-layer Boolean MLP

WX  
YZ

00

Can be generalized: Will require  
 $2^{N-1}$  perceptrons in hidden layer  
**Exponential in N**

10

UV  
00

01

11

10

00

Y  
Z

10

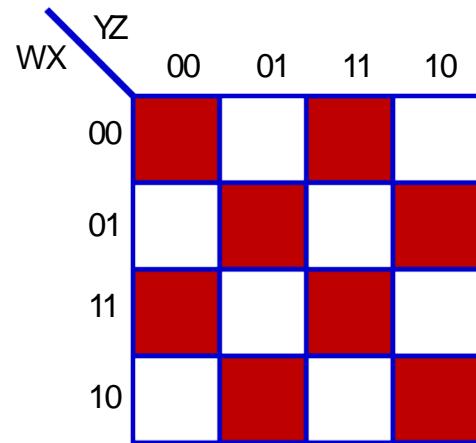
11

01

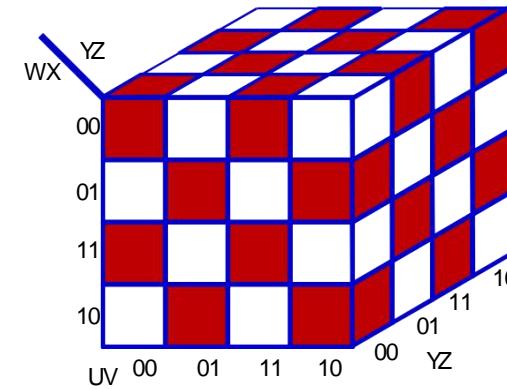
How many units if we use *multiple layers*?

layer) MLP for this Boolean function

# Width of a deepMLP

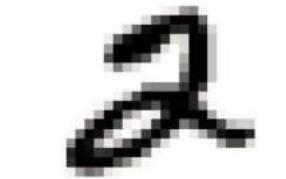


$$O = W \oplus X \oplus Y \oplus Z$$

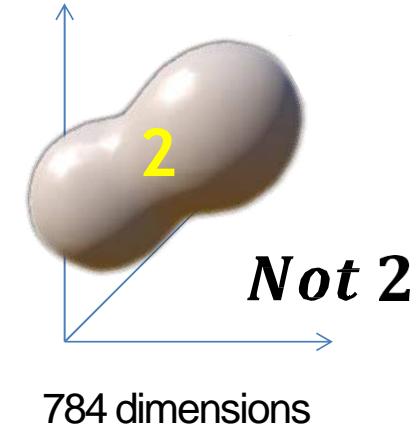
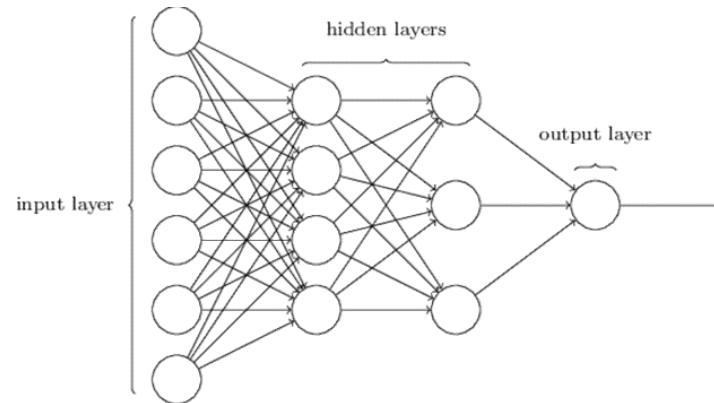


$$O = U \oplus V \oplus W \oplus X \oplus Y \oplus Z$$

# The MLP as a classifier



784 dimensions  
(MNIST)



- MLP as a function over real inputs
- MLP as a function that finds a complex “decision boundary” over a space of *reals*

# Linear Classification with Perceptron

Perceptron with hard/step threshold

Output = -1 if input  $w_1*x_1 + w_2*x_2 < \text{threshold } T$   
= 1 if input  $\geq T$

From training data,

$$w_1*(-2) + w_2*0 < T \text{ implies } -2w_1 < T$$

$$w_1*1 + w_2*(-1) < T \text{ implies } w_1 - w_2 < T$$

$$w_1*0 + w_2*1 \geq T \text{ implies } w_2 \geq T$$

$$w_1*1 + w_2*1 \geq T \text{ implies } w_1 + w_2 \geq T$$

Choose,  $w_1=1$

Then  $T > -2$ , choose  $T = -1$  (you can choose any value  $> -2$ )

Then,  $1 - w_2 < -1$  or  $w_2 > 2$ . Choose,  $w_2=3$ . (you can choose any value  $> 2$ )

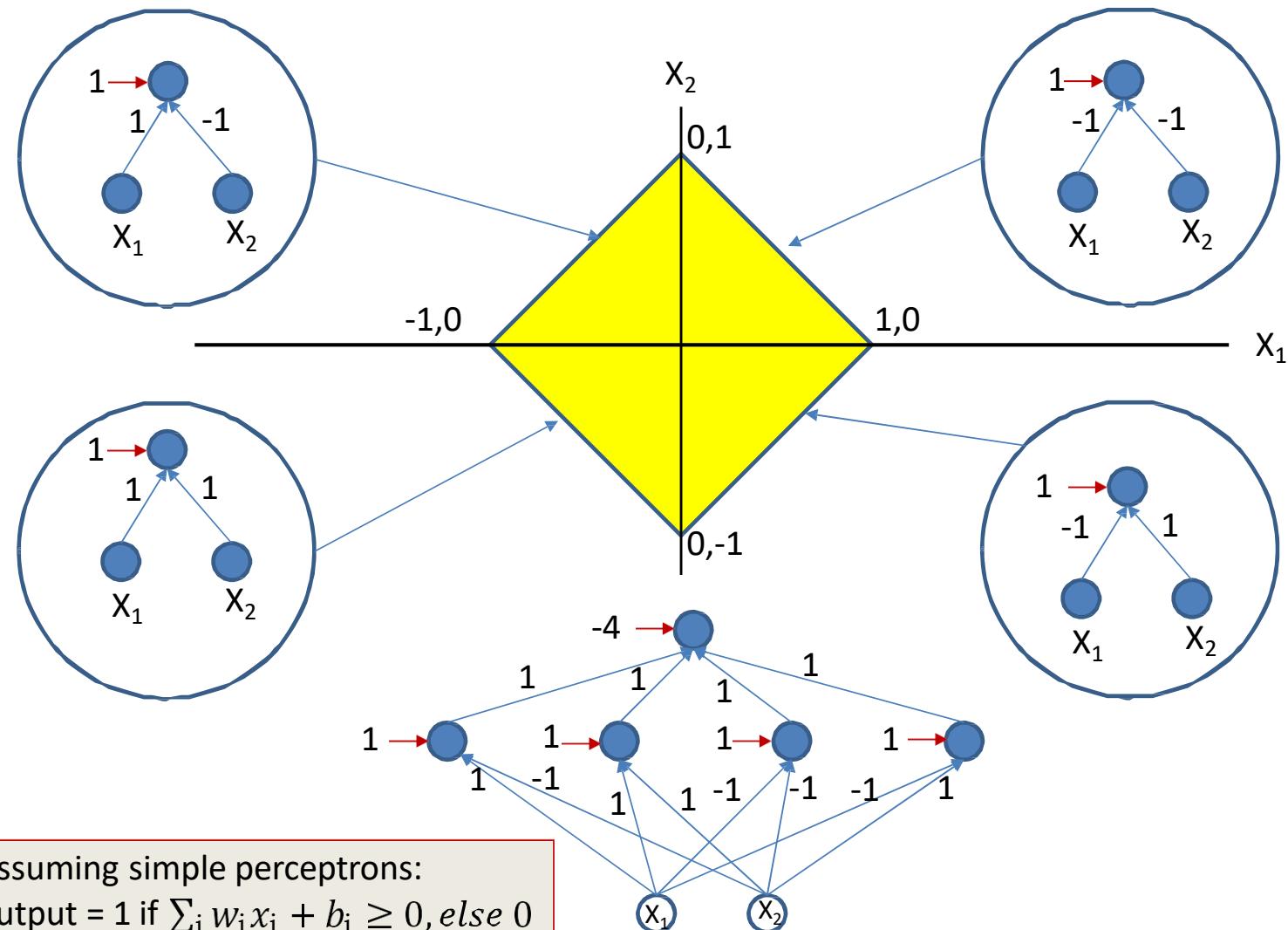
Then,  $w_1 + w_2 \geq T$  is automatically satisfied

Training Dataset

x1	x2	Class
-2	0	-1
1	-1	-1
0	1	1
1	1	1

Thus, classification can be realized with  $w_1=1$ ,  $w_2=3$ ,  $T=-1$ . Important to note that other values of  $w_1$ ,  $w_2$ , and  $T$  can also classify the training dataset, as long as the about 4 inequalities are satisfied

# Construct by hand





**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to Deep Learning

**Dr. Sugata Ghosal**

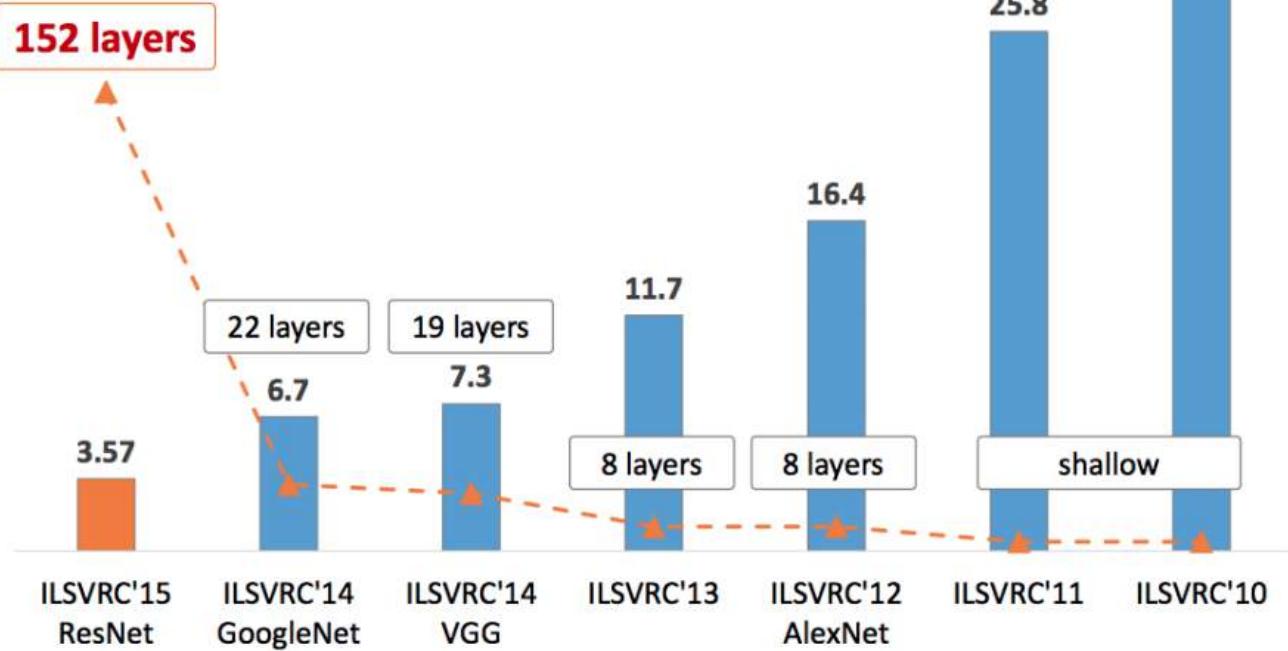
# Deep Learning (DL)

Deep neural networks are capable of solving very complex problems

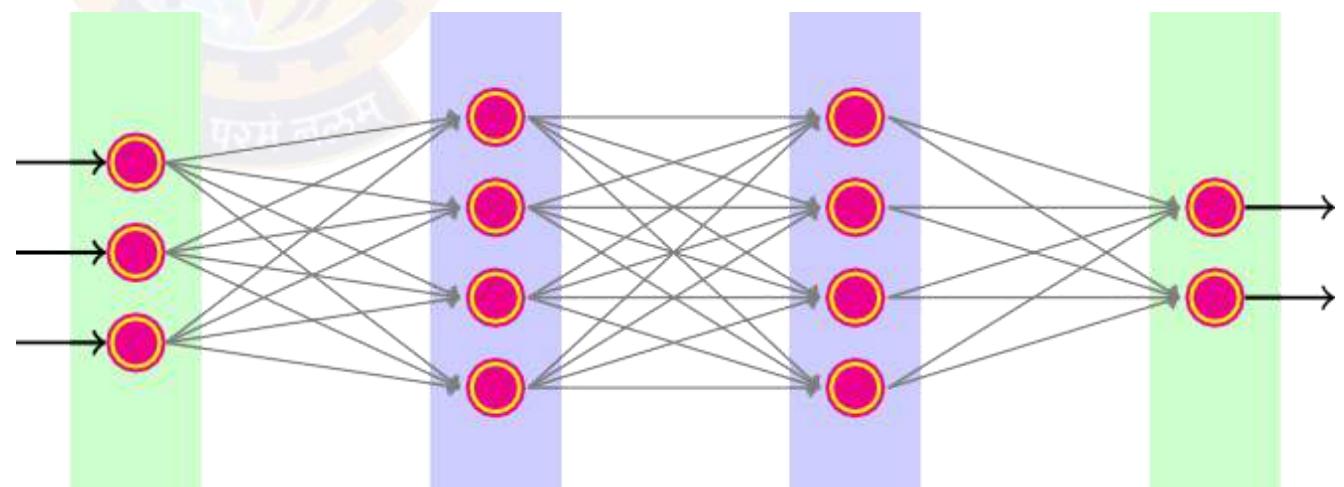
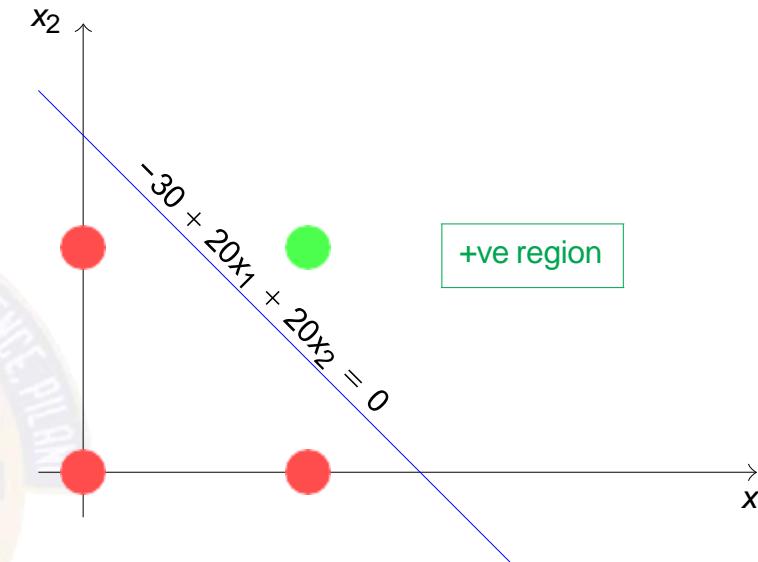
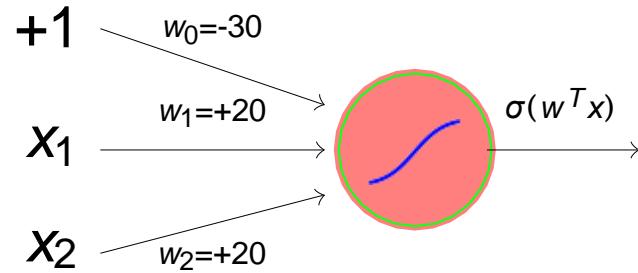
- ImageNet Challenge (2011): identify objects in images  
(4 million images in 21841 categories [Challenge 1000 categories])



Classification: ImageNet Challenge top-5 error

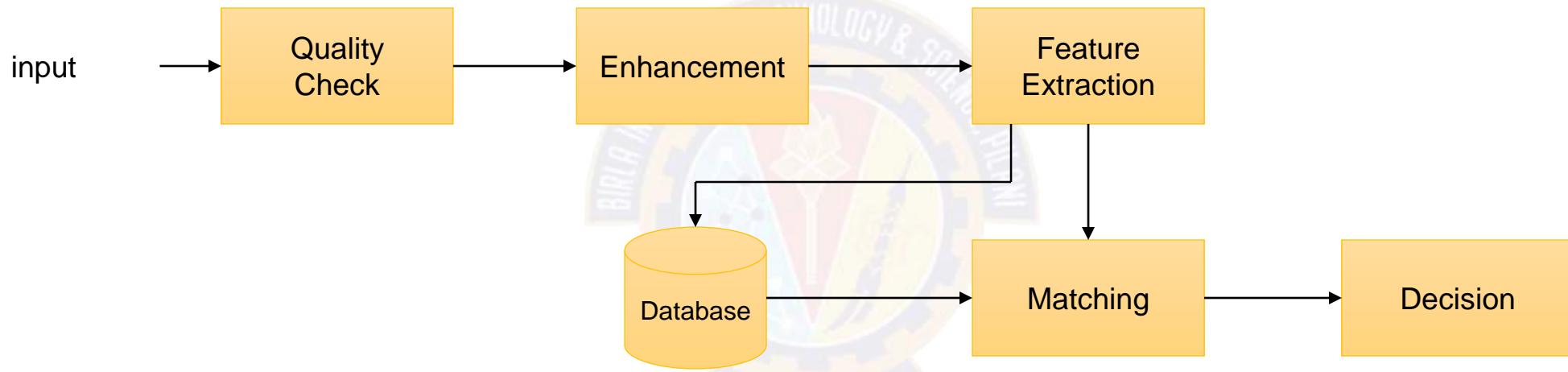


# Essentially it Represents A Decision Boundary



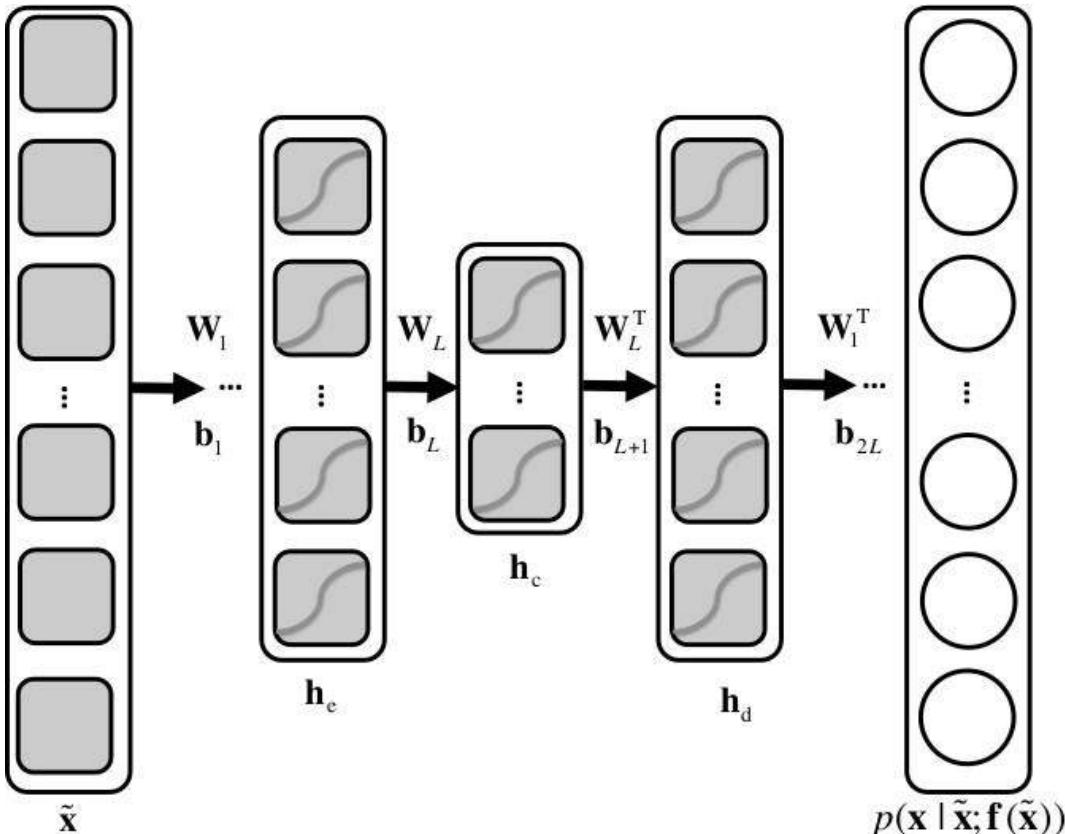
However, complex boundaries could be made using multiple neurons and stacking them in a layer.

# General ML Pipeline



Deep Learning can provided end-to-end learning

# Autoencoders



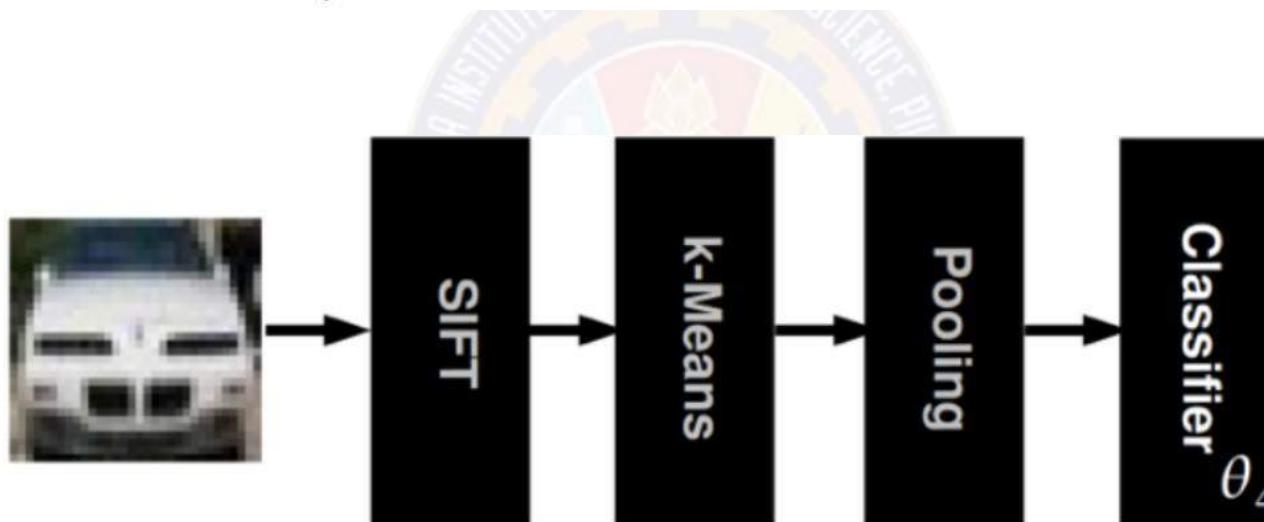
- An unsupervised learning algorithm, setting the target values to be equal to the inputs
- Learns an approximation

# Deep Learning

- **Shallow to Deep:** is linear combination to composition

$$\sum_i g_i \rightarrow g_1(g_2(g_3(\dots)))$$

- Why?



- Optimization is difficult for a non-convex, non-linear systems
- Freezing some layers makes it shallow.

# Key Points

- Traditional way to hand engineer feature is brittle and not scalable
- **Why now?** because we have data (imageNet), compute power (GPUs) and software (tensorFlow)
- DL learns underlying features/representations directly from data.
- DL have been found useful to many real life problems. Capable of solving very complex problems.
- As the data increases algorithm DL becomes better
- Deep learning is **NOT** just adding more layers to a neural network.



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN

---

---

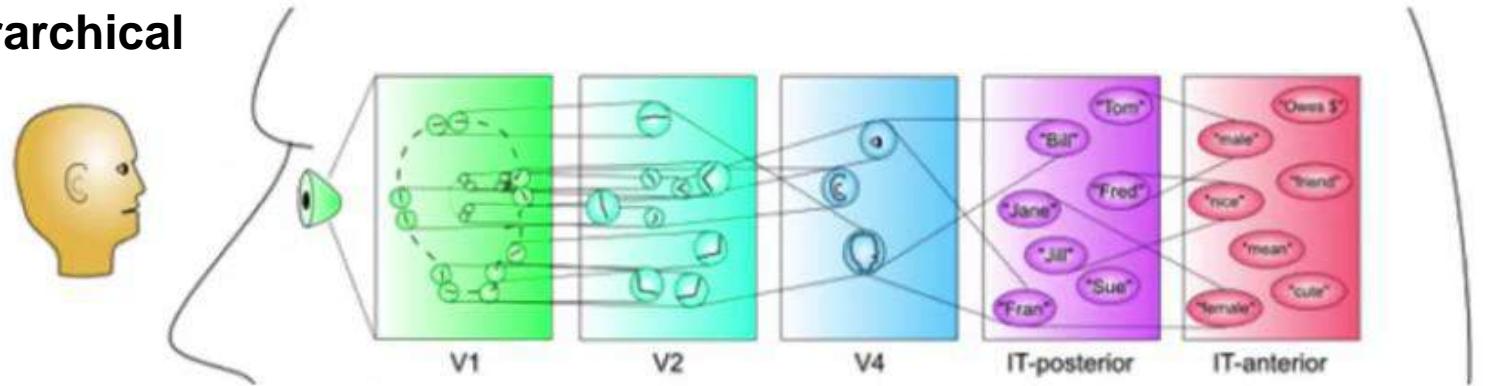
# Deep Learning

DL is constructing network of parameterized functional modules and training them from examples using gradient based optimization

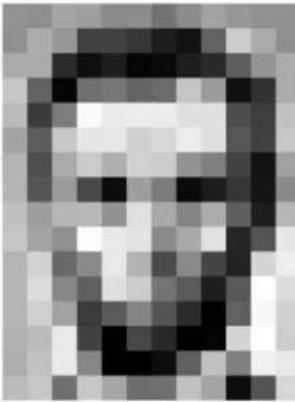
- Yun LeCun

# Visual Cortex is Hierarchical

Visual Cortex is Hierarchical



Images are numbers (so determine feature)



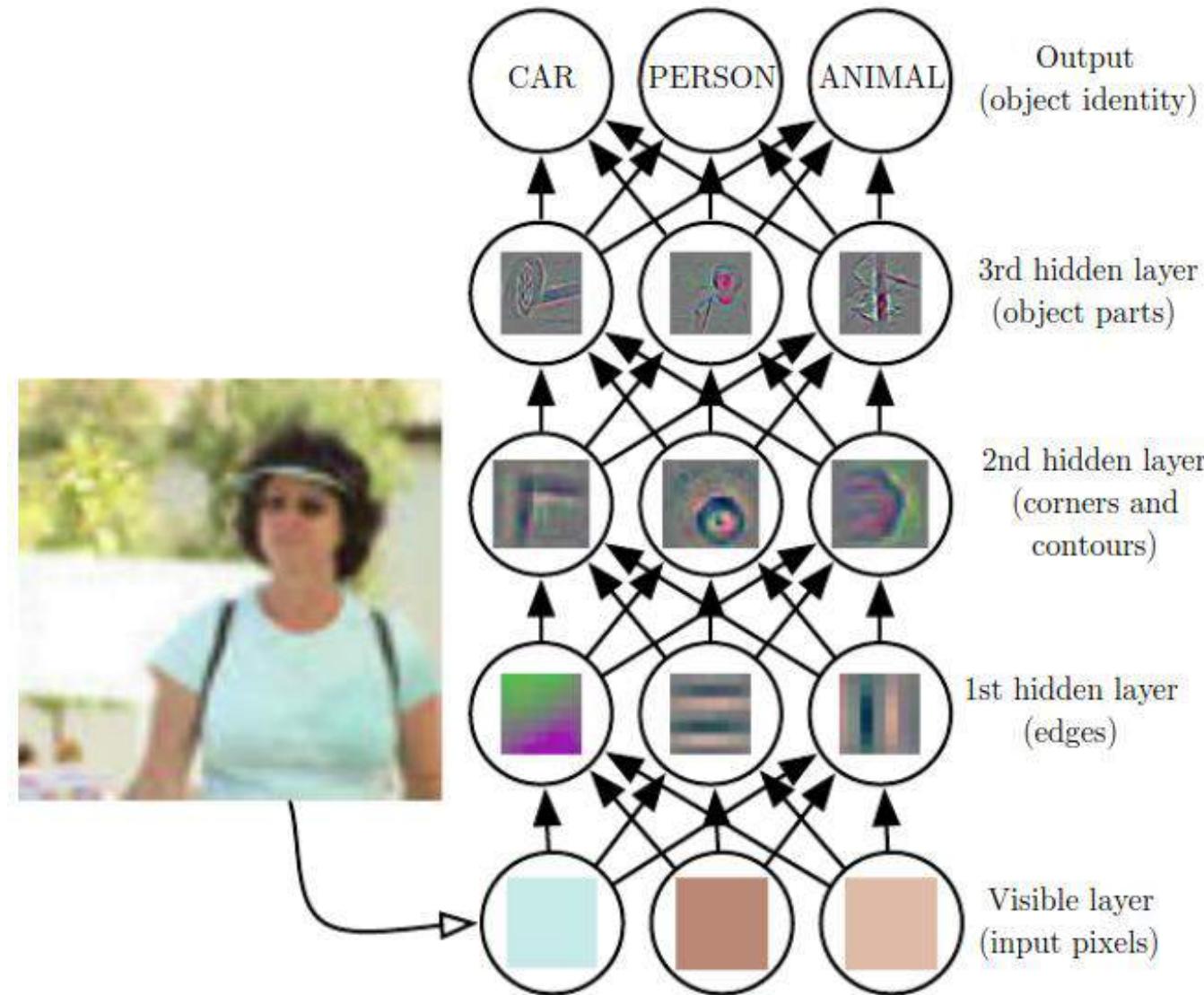
197	193	174	164	150	162	129	151	172	161	166	156
188	192	163	79	73	62	39	17	110	210	180	154
180	190	85	34	54	6	10	33	48	196	159	181
206	198	5	134	191	111	130	204	165	15	56	180
194	188	187	281	237	239	239	228	227	87	71	201
172	156	267	238	239	214	230	239	228	88	74	204
188	188	179	209	186	216	217	188	178	75	35	149
148	187	168	64	50	168	134	11	31	62	22	148
199	198	191	193	168	227	173	143	182	186	36	190
205	174	186	282	236	231	169	176	228	43	95	234
190	215	116	149	236	187	96	150	78	38	218	241
190	224	147	168	227	210	127	103	36	184	29	224
190	214	173	56	103	143	98	90	2	109	249	216
187	196	236	71	1	81	47	0	6	217	285	211
183	202	237	145	0	0	12	106	209	138	243	236
195	206	123	207	177	121	128	200	179	13	96	218

What the computer sees
197
193
174
164
150
162
129
151
172
161
166
156
188
192
163
79
73
62
39
17
110
210
180
154
180
190
85
34
54
6
10
33
48
196
159
181
206
198
5
134
191
111
130
204
165
15
56
180
194
188
187
281
237
239
239
228
227
87
71
201
172
195
207
239
239
214
229
229
228
98
74
206
188
179
209
186
215
217
158
139
75
30
148
189
165
64
10
188
134
11
31
62
32
148
199
168
191
193
158
227
178
143
182
106
36
190
205
185
232
236
231
149
178
228
43
95
234
190
216
146
236
187
86
150
79
38
218
241
190
224
147
198
227
210
127
103
36
101
255
224
190
214
173
56
103
143
98
90
2
109
249
216
187
196
236
71
1
81
47
0
6
217
285
211
183
202
237
145
0
0
12
106
209
138
243
236
195
206
123
207
177
121
128
200
179
13
96
218

There could be occlusion, viewpoint variation, scale variation, illumination variation, deformation, background clutter, noise, intra-class variation and much more.

DL helps to get hierarchy of features

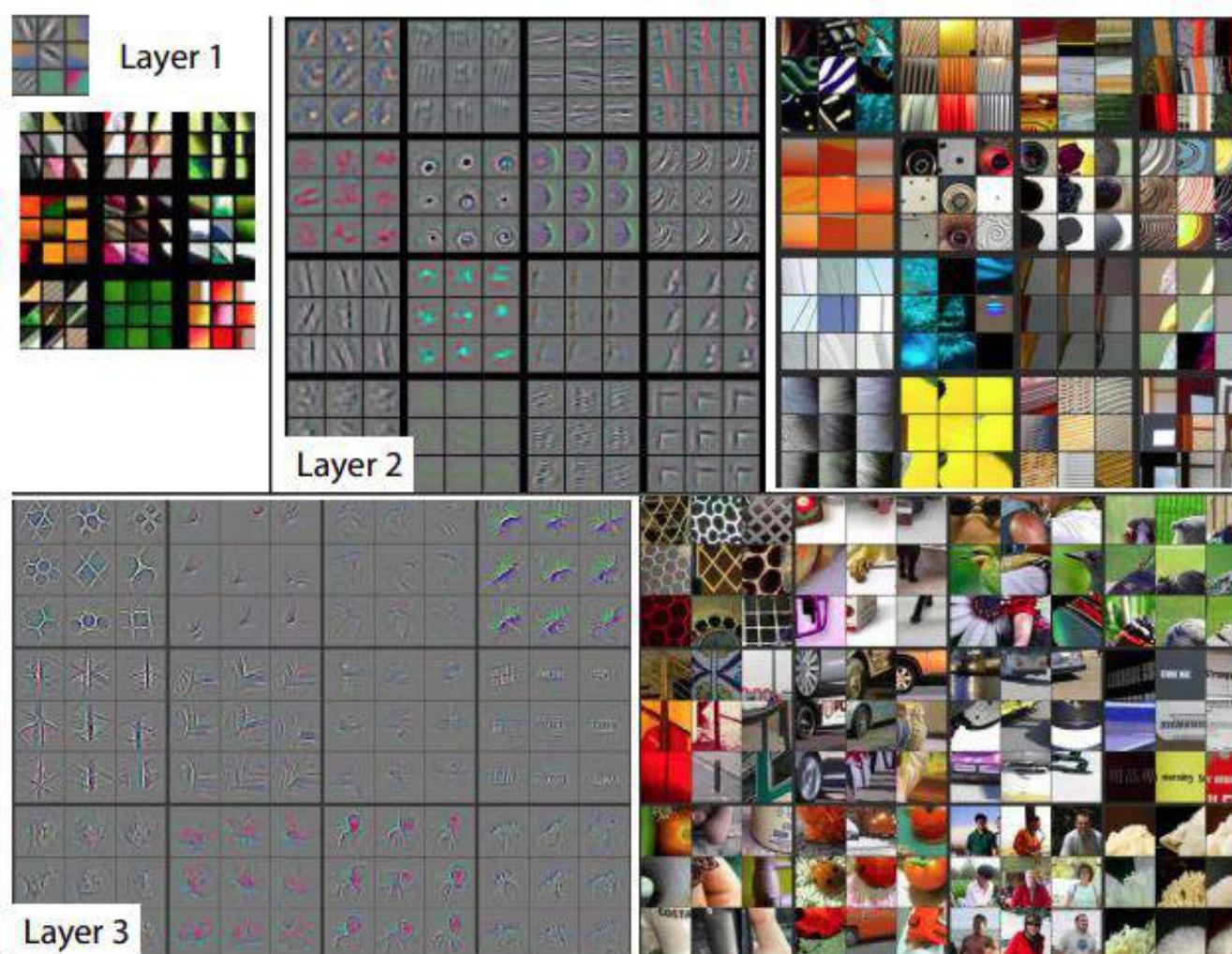
# DL Builds Hierarchy of Features



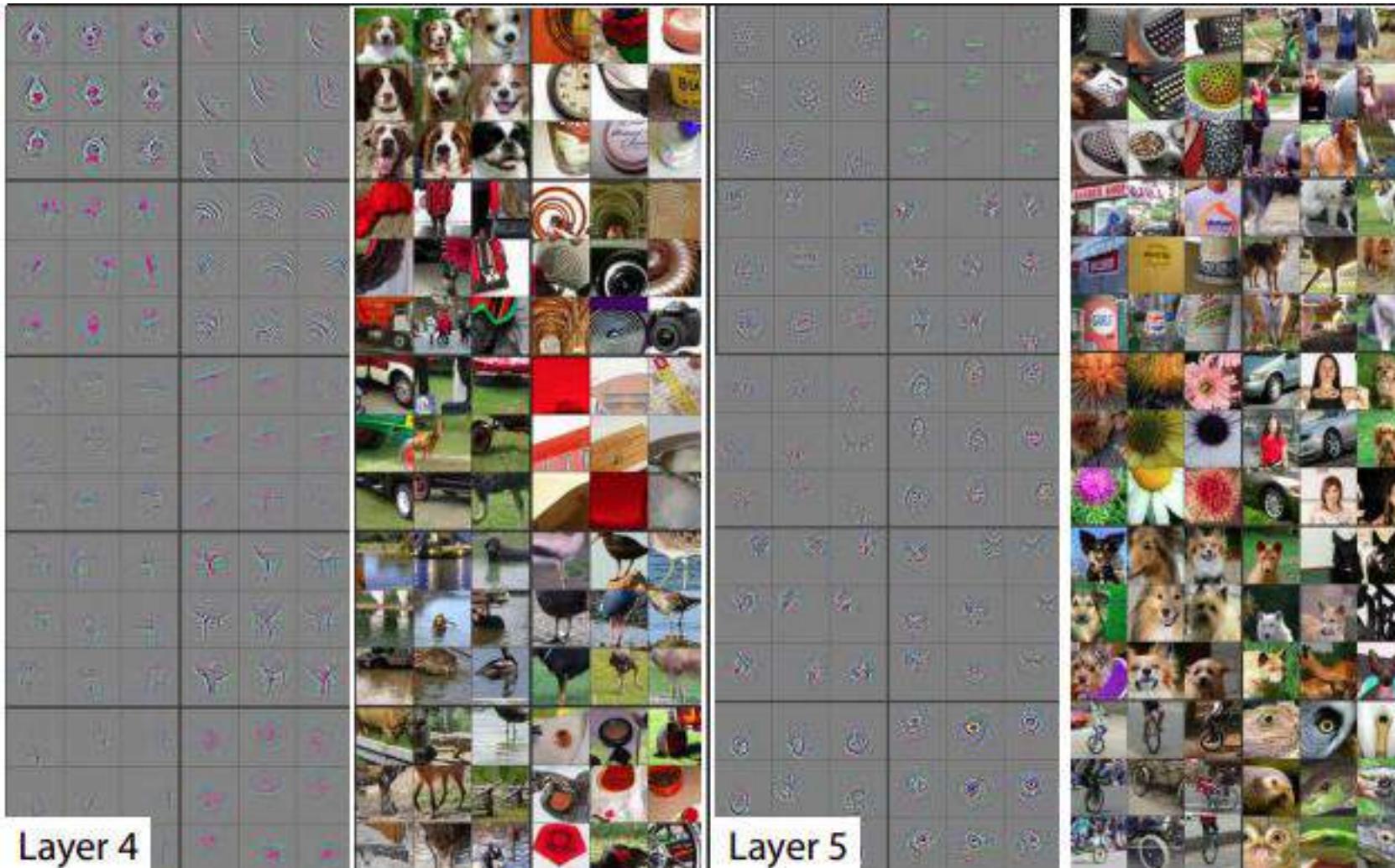
**Higher (or deeper) layers represents abstraction of the the features**

# ECCV2014 - Visualizing and Understanding CNN

A case study on 5 layer trained network



# ECCV2014 - Visualizing and Understanding CNN





**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN (hyperparameters)

# Hyperparameter Tuning

**There are many interesting questions for the network**

- How many layers?
- How many units in each layer?
- What should be the learning rate?
- What is right activation function? etc.

**Difficult to answer in the beginning**

It is an iterative process



[Demo](#)



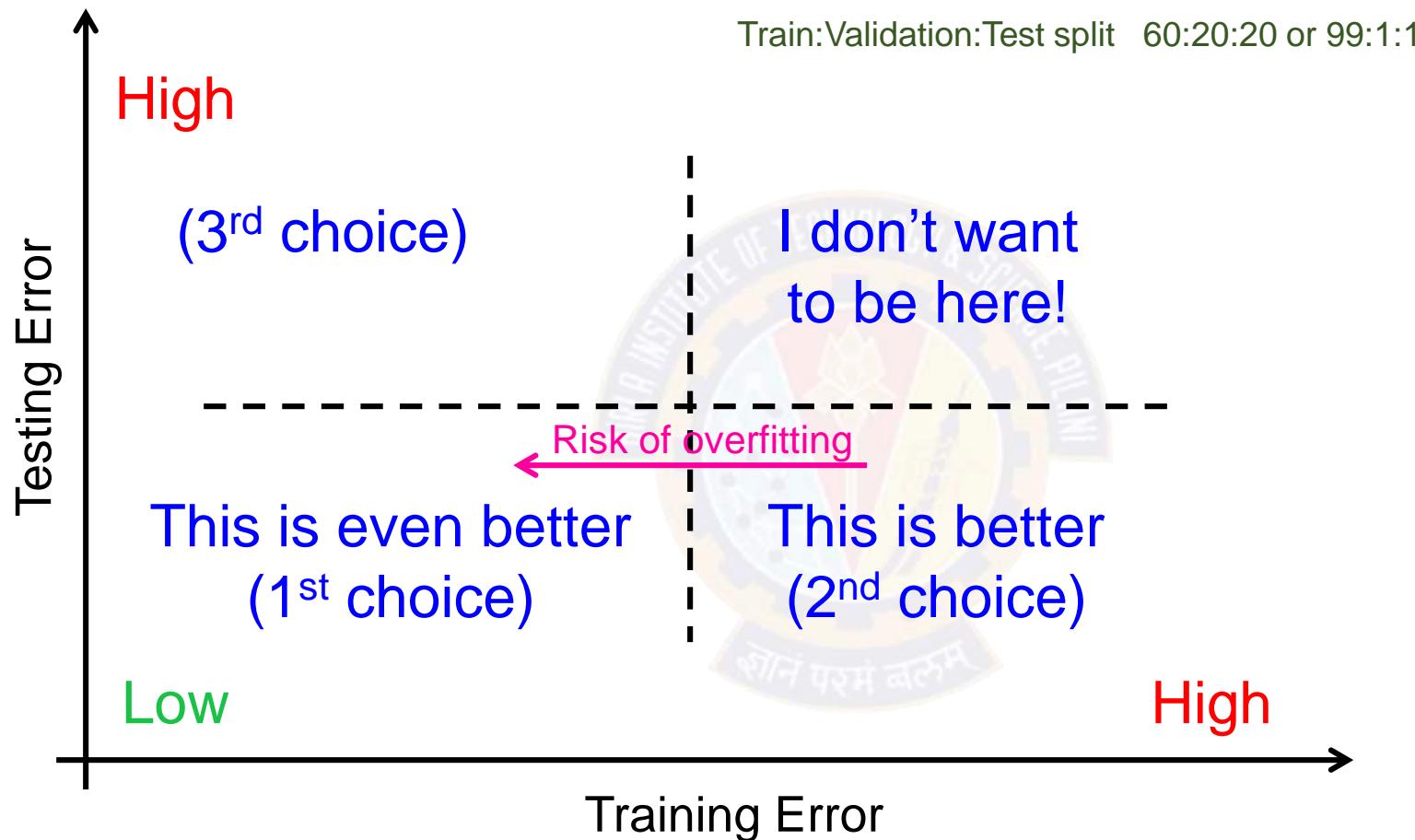
**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN

---

---

# Which side do you want to be?



**Low training error comes with a risk of overfitting (high variance)**



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN Weight Decay

# Regularization for logistic regression

- Optimization minimize loss  $\min_w J(w)$  by adjusting  $w$  where

$$J(w) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$$

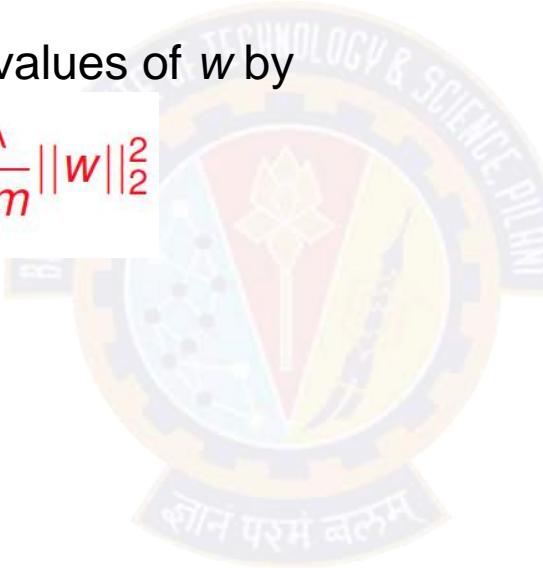
- Regularization penalizes the large values of  $w$  by

$$J(w) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2$$

where

$$\|w\|_2^2 = \sum_j w_j^2 = w^T w$$

$\lambda$  being regularization parameter



# Regularization for NN

- As there could be L layers each with their parameters so

$$J(w^{[1]}, w^{[2]}, \dots, w^{[L]}) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|w^{[l]}\|_F^2$$

where **frobenius** norm is

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} w_{ij}^2$$

- How you update the parameter earlier?

Get

$$dw^{[l]} = (\text{from backpropagation}) \text{ that is } \frac{\partial J}{\partial w^{[l]}}$$

then

$$w^{[l]} = w^{[l]} - \alpha \cdot dw^{[l]}$$

# Regularization for NN (contd...)

- With regularization term  $dw^{[l]} = (\text{from backpropagation}) + \frac{\lambda}{m}w^{[l]}$
- Therefore the update is modified to

$$w^{[l]} = w^{[l]} - \alpha.(\text{(from backpropagation)} + \frac{\lambda}{m}w^{[l]})$$

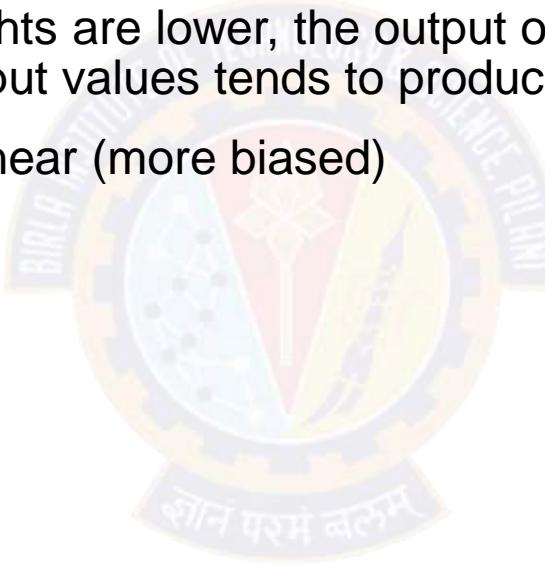
Which is

$$w^{[l]} = \left(1 - \frac{\alpha\lambda}{m}\right)w^{[l]} - \alpha.(\text{from backpropagation})$$

- Due to the  $\left(1 - \frac{\alpha\lambda}{m}\right)$  factor, this update method is also called **weight decay**
- Our objective here is to penalize the weight matrices being too large

# Penalize the weight matrices from being too large

- With low weight, the connection get weakened so network has effectively less connections and become simpler.
- Another intuition is that, when weights are lower, the output of the units is also lower. Assume activation function be tanh small input values tends to produce linear output.
- So overall n/w tends to becomes linear (more biased)



# Deep MLP: L2 Regularization

## Specify the Architecture

```
# Layer 1 = input layer
# specify the input size for in the first layer.

dnnModel.add(layers.Dense(50, activation='relu', input_shape= (28*28,)))

# Layer 2 = hidden Layer with Regularizers
dnnModel.add(layers.Dense(60, activation='relu', kernel_regularizer=regularizers.l2(0.01),
                        activity_regularizer=regularizers.l2(0.01)) )

# Layer 3 = hidden Layer
dnnModel.add(layers.Dense(30, activation='relu'))

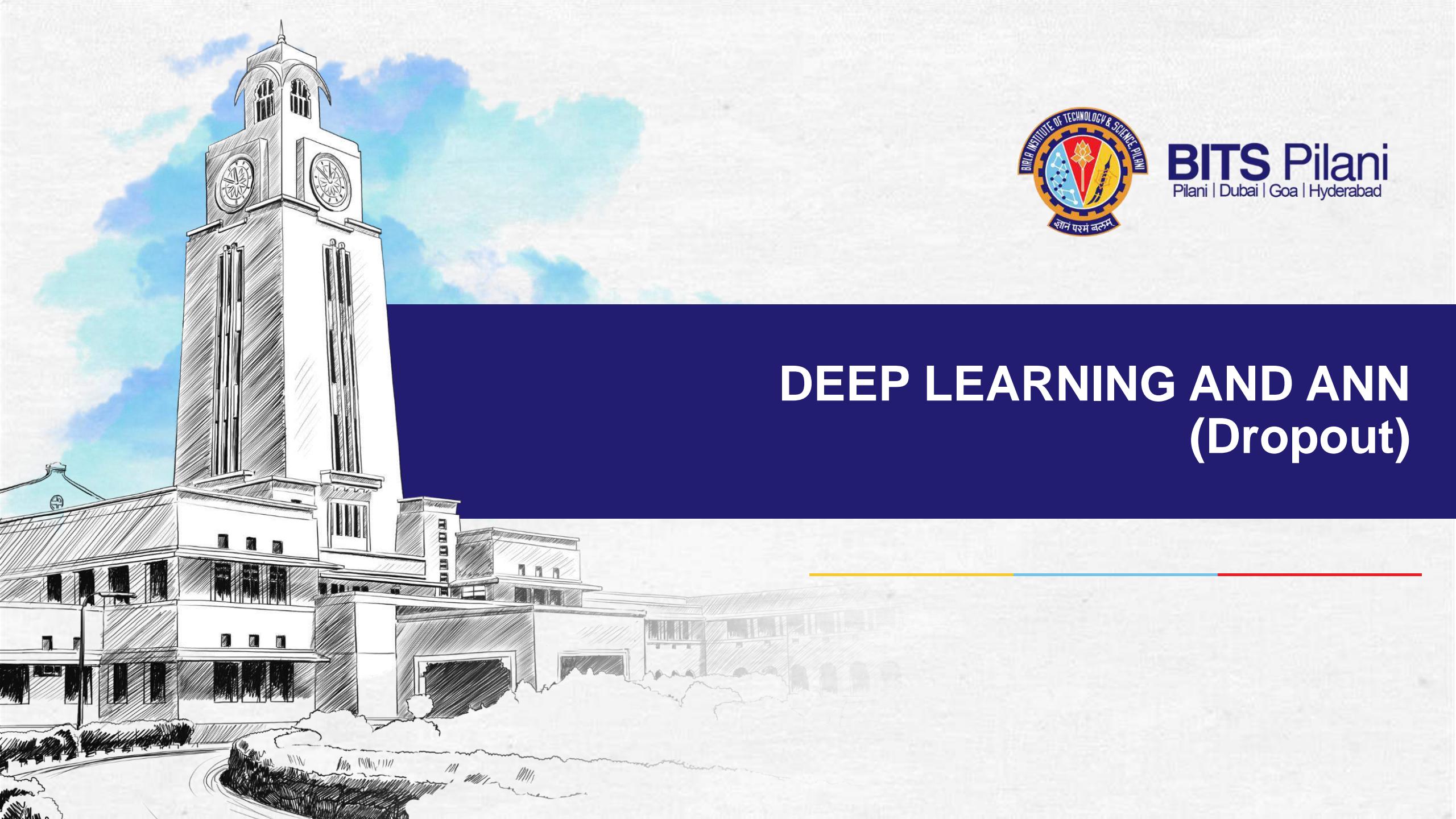
# Layer 4 = output Layer
dnnModel.add(layers.Dense(10, activation='softmax'))

dnnModel.summary()
```



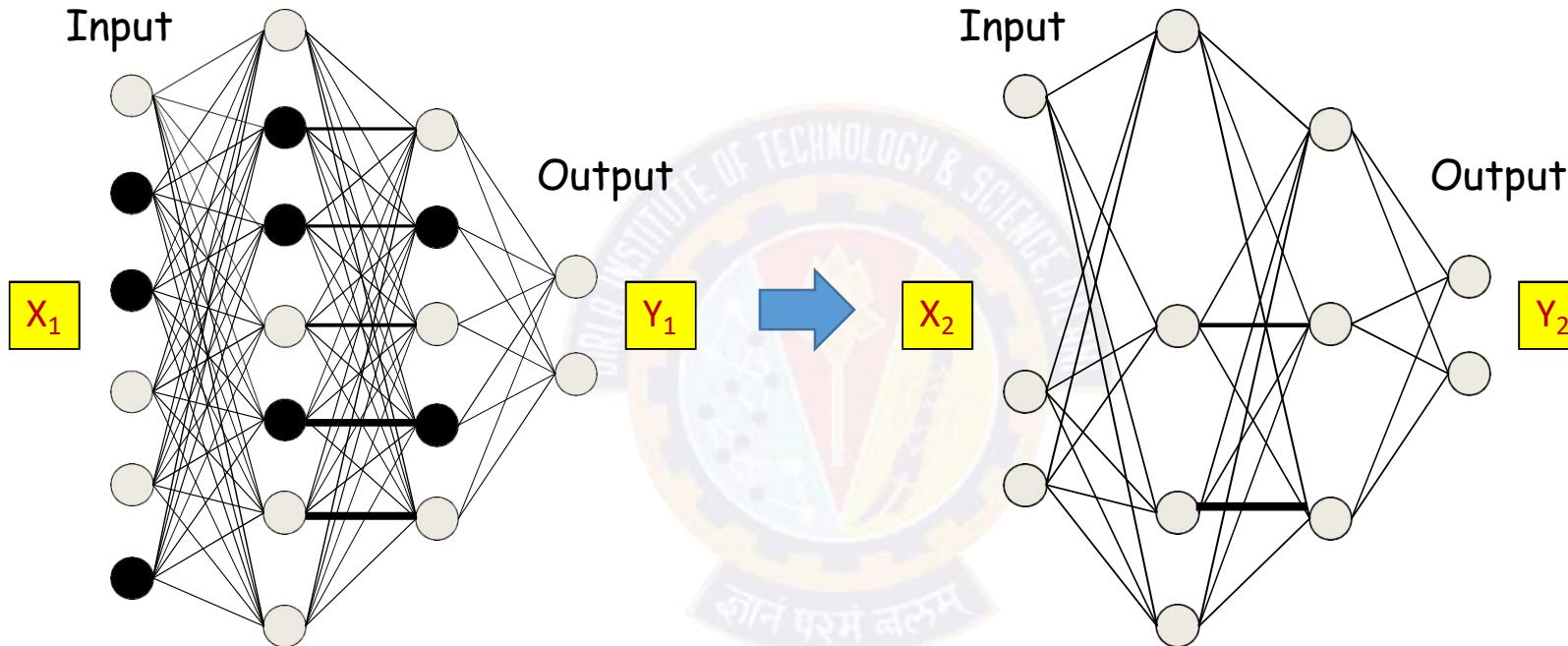
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN (Dropout)



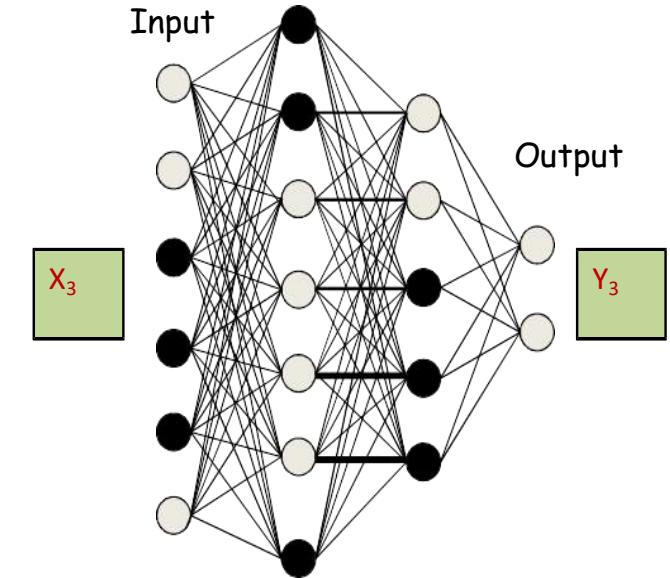
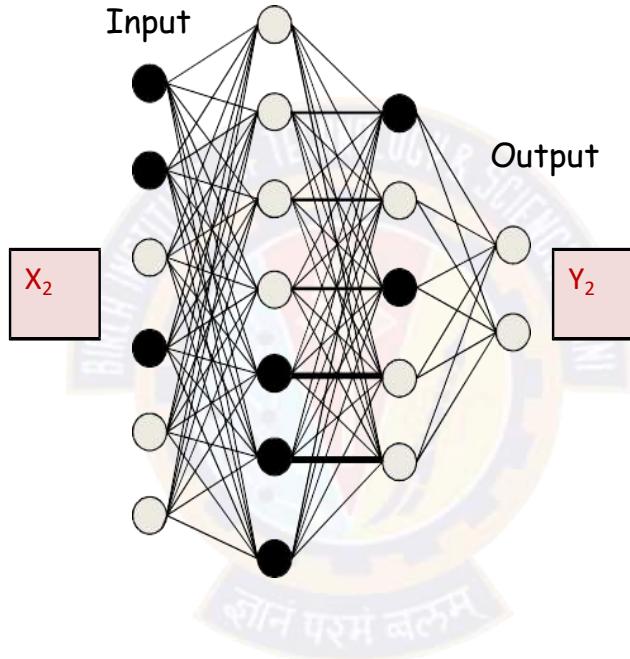
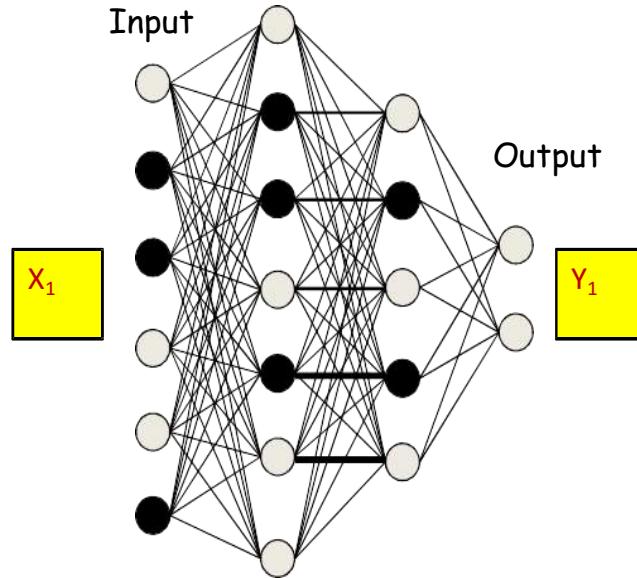
# Dropout Regularization

Randomly shutdown some of the units



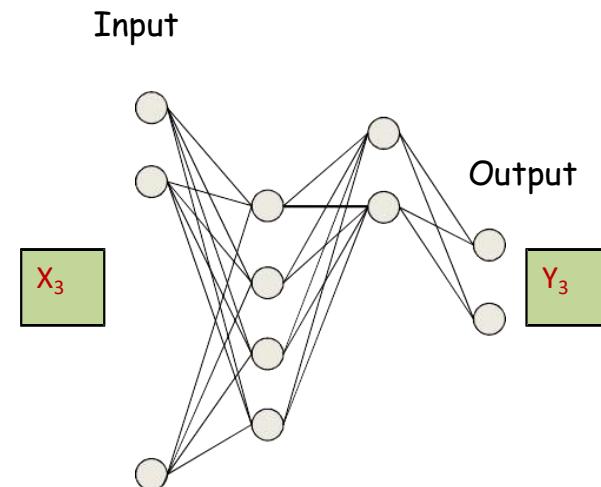
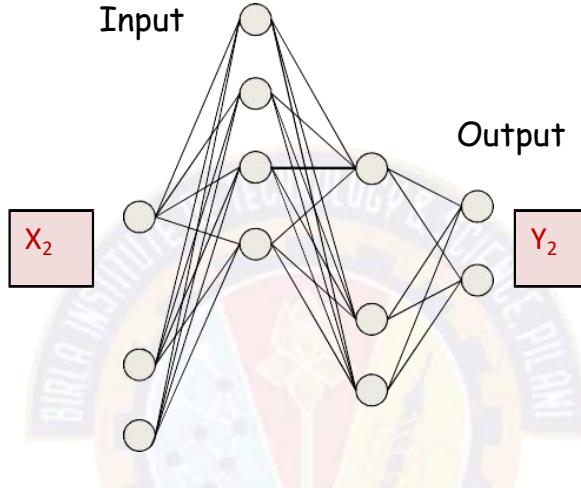
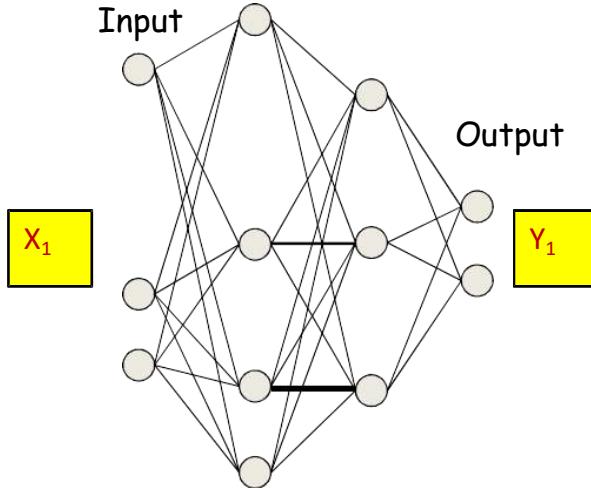
- Drop probabilities for different layers may be different
- Networks can not rely on single connection. So have to give importance to others also
- Issue is that cost function is now not well defined

# Dropout



- The pattern of dropped nodes changes for each input, i.e., in every pass through the net

# Dropout



- **During training:** Backpropagation is effectively performed only over the remaining network
  - The effective network is different for different inputs
  - Gradients are obtained only for the weights and biases *from “On” nodes to “On” nodes*
    - For the remaining, the gradient is just 0

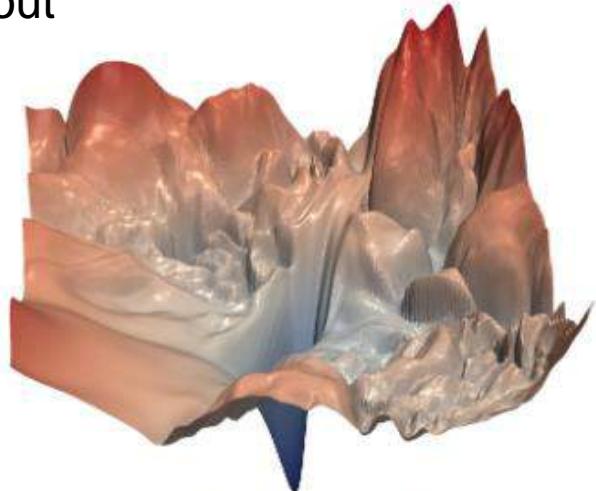
# Deep MLP: Dropout

## Specify the Architecture

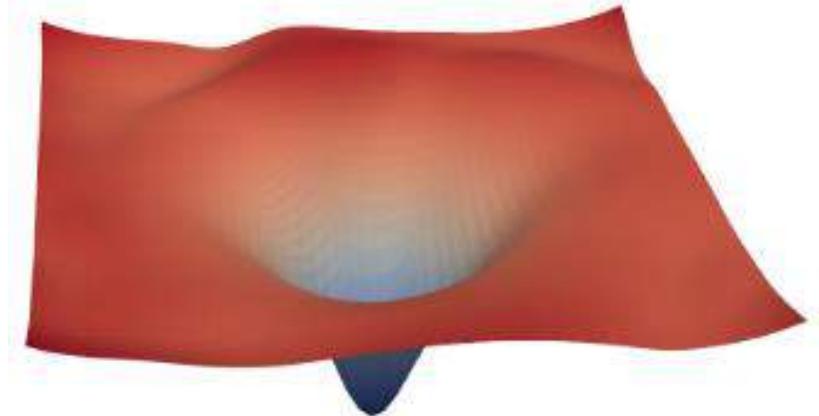
```
# Layer 1 = input layer  
# specify the input size for in the first layer.  
  
dnnModel.add(layers.Dense(50, activation='relu', input_shape= (28*28,)))  
  
# Layer 2 = hidden layer |  
dnnModel.add(layers.Dense(60, activation='relu'))  
  
# Add dropout of 50% to Layer 2  
dnnModel.add(layers.Dropout(0.5))  
  
# Layer 3 = hidden layer  
dnnModel.add(layers.Dense(30, activation='relu'))  
  
# Add dropout of 50% to Layer 3  
dnnModel.add(layers.Dropout(0.5))  
  
# Layer 4 = output layer  
dnnModel.add(layers.Dense(10, activation='softmax'))  
  
dnnModel.summary()
```

# Regularization

- Dropout



(a) without skip connections



(b) with skip connections

Visualizing the Loss Landscape of Neural Nets, Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein, 2018

- Early stopping





**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN (Other methods)



# Other Regularization Methods

- Increase the data by **data augmentation**
- Flip, rotate, scale, translate, deform ....



CocaColaZero1\_1.png



CocaColaZero1\_2.png



CocaColaZero1\_3.png



CocaColaZero1\_4.png



CocaColaZero1\_5.png



CocaColaZero1\_6.png



CocaColaZero1\_7.png



CocaColaZero1\_8.png

- Normalize training examples to **speed up your training**

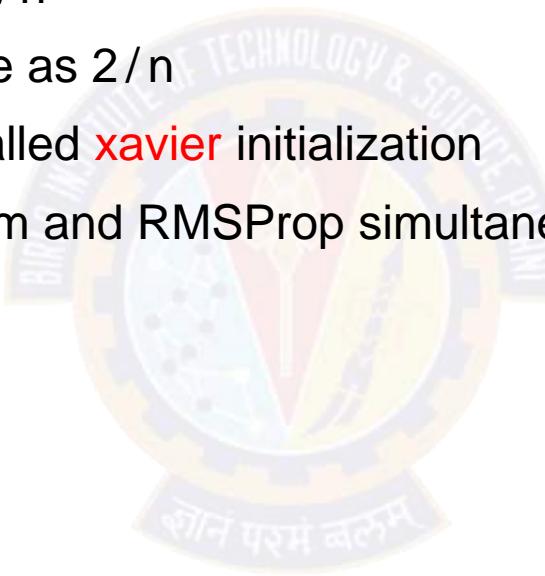


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# DEEP LEARNING AND ANN (Weight Initialization and Training)

# Weight initialization and training

- Exploding and vanishing gradient is an issue
- Set the variance of weights to be  $1/n$
- For ReLu it is better to use variance as  $2/n$
- For tanh use variance as  $\sqrt{1/n}$  called **xavier** initialization
- **Adam** optimization, uses momentum and RMSProp simultaneously



# Deep MLP: Code Snippet

## Configure Training Parameters

```
# Configure the model for training, by using appropriate optimizers and regularizations
# Available optimizer: adam, rmsprop, adagrad, sgd
# loss: objective that the model will try to minimize.
# Available loss: categorical_crossentropy, binary_crossentropy, mean_squared_error
# metrics: List of metrics to be evaluated by the model during training and testing.

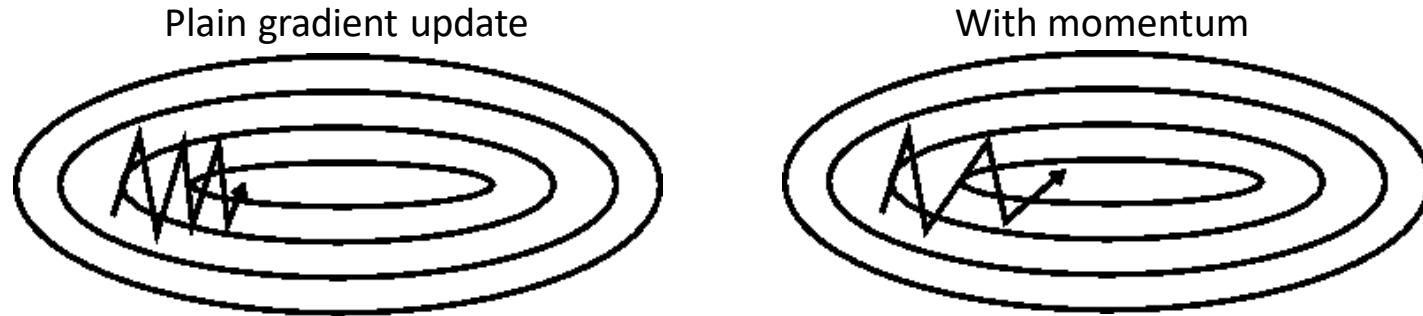
dnnModel.compile( optimizer = 'adam', loss = 'categorical_crossentropy', metrics=['accuracy'] )
```

## Train the Model

```
# train the model

h = dnnModel.fit( Xtrain, Ytrain, epochs=25, batch_size=64)
```

# Momentum Update



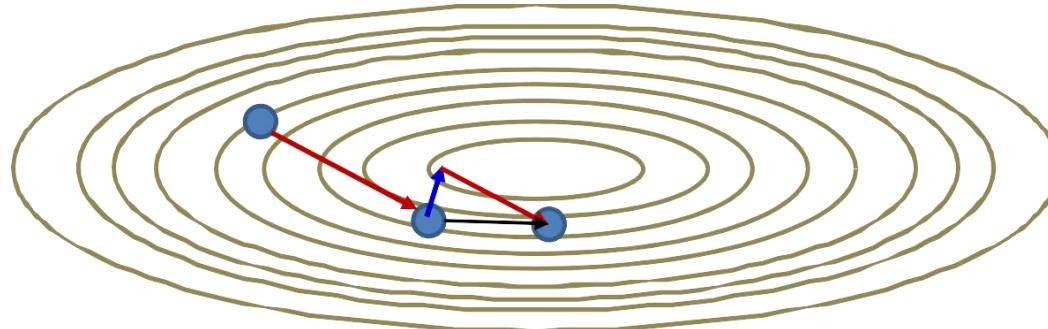
- The momentum method maintains a running average of all gradients until the *current* step

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Loss(W^{(k-1)})^T$$

$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

- Typical  $\beta$  value is 0.9
- The running average steps
  - Get longer in directions where gradient stays in the same sign
  - Become shorter in directions where the sign keeps flipping

# Momentum Update

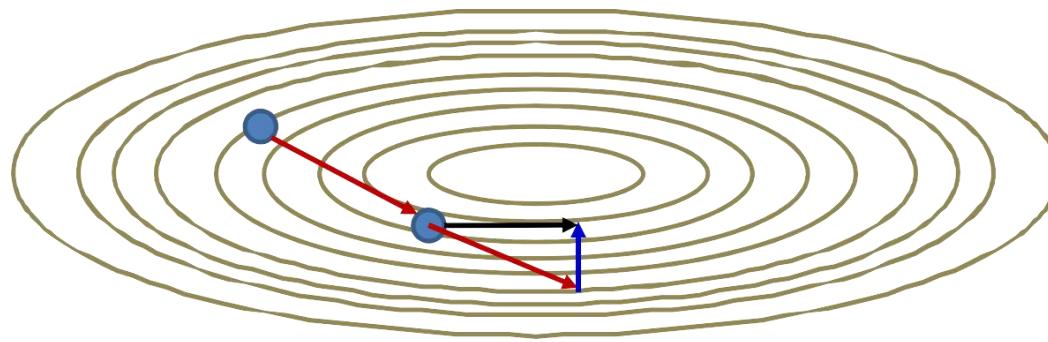


- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Loss(W^{(k-1)})^T$$

- At any iteration, to compute the current step:
  - First computes the gradient step at the current location
  - Then adds in the scaled *previous* step
    - Which is actually a running average
  - To get the final step

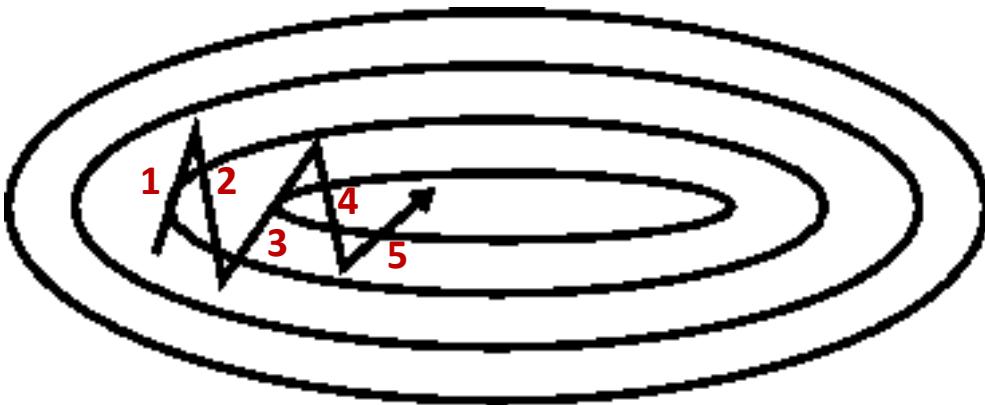
# Nestorov's Accelerated Gradient



- Nestorov's method

$$\begin{aligned}\Delta W^{(k)} &= \beta \Delta W^{(k-1)} - \eta \nabla_W \text{Loss}(W^{(k-1)} + \beta \Delta W^{(k-1)})^T \\ W^{(k)} &= W^{(k-1)} + \Delta W^{(k)}\end{aligned}$$

# Smoothing the trajectory



Step	X component	Y component
1	1	+2.5
2	1	-3
3	3	+2.5
4	1	-2
5	2	1.5

- Simple gradient and acceleration methods still demonstrate oscillatory behavior in some directions
  - Depends on magic step size parameters
- Observation: Steps in “oscillatory” directions show large total movement
  - In the example, total motion in the vertical direction is much greater than in the horizontal direction
- Improvement: Dampen step size in directions with high motion
  - ***Second order term***

# RMS Prop

- Notation:
  - Updates are *by parameter*
  - Sum derivative of divergence w.r.t any individual parameter  $w$  is shown as  $\partial_w D$
  - The *squared* derivative is  $\partial_w^2 D = (\partial_w D)^2$ 
    - Short-hand notation represents the squared derivative, not the second derivative
  - The *mean squared* derivative is a running estimate of the average squared derivative. We will show this as  $E[\partial_w^2 D]$
- Modified update rule: We want to
  - scale down updates with large mean squared derivatives
  - scale up updates with small mean squared derivatives

# RMS Prop

- This is a variant on the *basic* mini-batch SGD algorithm
- **Procedure:**
  - Maintain a running estimate of the mean squared value of derivatives for each parameter
  - Scale update of the parameter by the *inverse* of the *root mean squared* derivative

$$E[\partial_w^2 D]_k = \gamma E[\partial_w^2 D]_{k-1} + (1 - \gamma)(\partial_w^2 D)_k$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{E[\partial_w^2 D]_k + \epsilon}} \partial_w D$$

# ADAM: RMSprop with momentum

- RMS prop only considers a second-moment normalized version of the current gradient
- ADAM utilizes a smoothed version of the *momentum-augmented* gradient
  - Considers both first and second moments
- **Procedure:**
  - Maintain a running estimate of the mean derivative for each parameter
  - Maintain a running estimate of the mean squared value of derivatives for each parameter
  - Scale update of the parameter by the *inverse of the root mean squared* derivative

$$m_k = \delta m_{k-1} + (1 - \delta)(\partial_w D)_k$$

$$v_k = \gamma v_{k-1} + (1 - \gamma)(\partial_w^2 D)_k$$

$$\hat{m}_k = \frac{m_k}{1 - \delta^k}, \quad \hat{v}_k = \frac{v_k}{1 - \gamma^k}$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{\hat{v}_k + \epsilon}} \hat{m}_k$$

# Other variants of the same theme

- Many:
  - Adagrad
  - AdaDelta
  - ADAM
  - AdaMax
  - ...
- Generally no explicit learning rate to optimize
  - But come with other hyper parameters to be optimized
  - Typical params:
    - RMSProp:  $\eta = 0.001, \gamma = 0.9$
    - ADAM:  $\eta = 0.001, \delta = 0.9, \gamma = 0.999$



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# ANN and Deep Learning

**Dr. Sugata Ghosal**

---

[sugata.ghosal@pilani.bits-pilani.ac.in](mailto:sugata.ghosal@pilani.bits-pilani.ac.in)

# Syllabus

M1	<b>Artificial Neural Network:</b> Introduction and Background, Discrimination power of single neuron, Training a single perceptron (delta rule) , Multilayer Neural Networks, Activation functions and Loss functions, Backpropagation
M2	<b>Deep Learning:</b> Introduction to end to end learning, Abstractions of features using deep layers, Hyper parameter tuning, Regularization for Deep Learning, Dropout
M3	<b>Convolution Networks with Deep Learning:</b> CNN, Pooling, Variants of pooling functions, CNN with Fully connected Networks, RCNN, Faster RCNN
M4	<b>Sequence Modeling in Neural Network:</b> Architecture of RNN , Unfolding of RNN, Training RNN, LSTM and its applications
M5	<b>Autoencoders with Deep Learning:</b> Undercomplete Autoencoders, Regularized Autoencoders, Variational autoencoders, Manifold learning with Autoencoders, Applications of Autoencoders
M6	<b>Generative deep learning models:</b> Boltzmann Machine, Restricted Boltzmann Machine, Deep Belief Machines, GAN, Applications of GAN



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

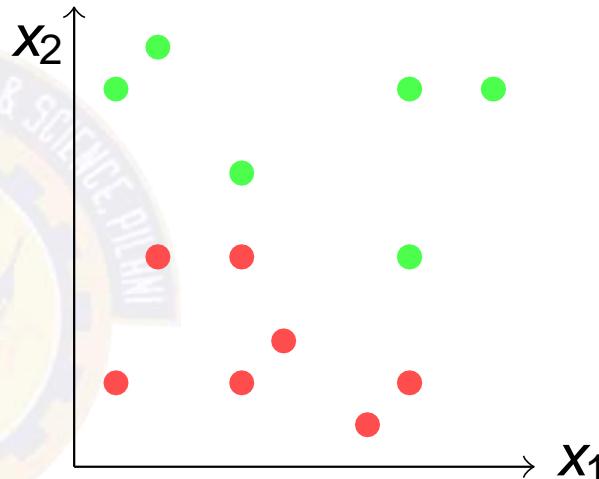
## Background of NN

# Linear Classification

Consider Following data

$x_1$	$x_2$	y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red

Data is in 2D, so let us visualize

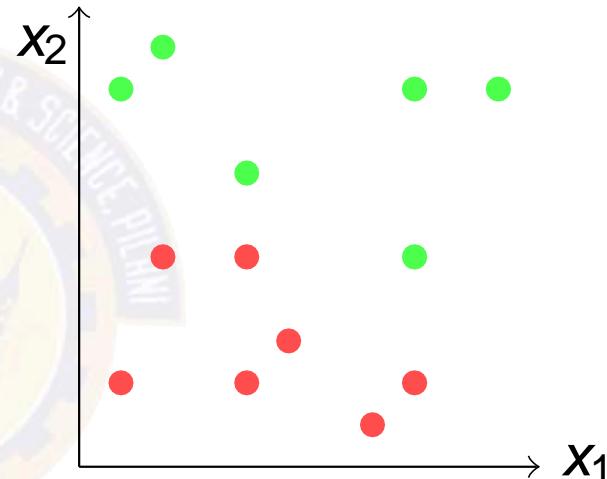


# Linear Classification

Consider Following data

$x_1$	$x_2$	y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red

Data is in 2D, so let us visualize



- Data looks **linearly separable**
- What is the decision boundary?

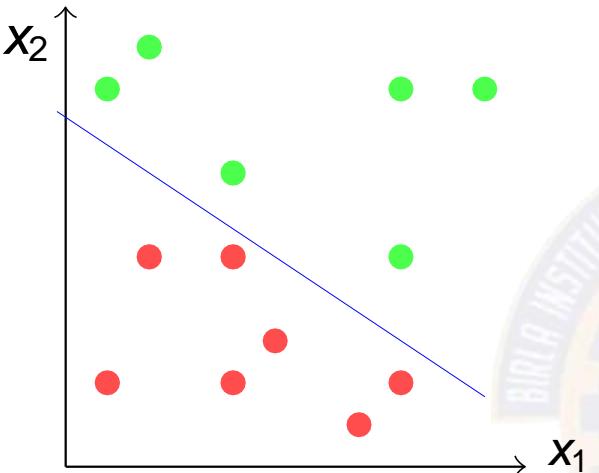
Many Possibilities, such as

if  $(2x_1 + 3x_2 - 25 > 0)$  it is **green**  
otherwise **red**

# What about this arrangement?

With chosen *decision boundary*

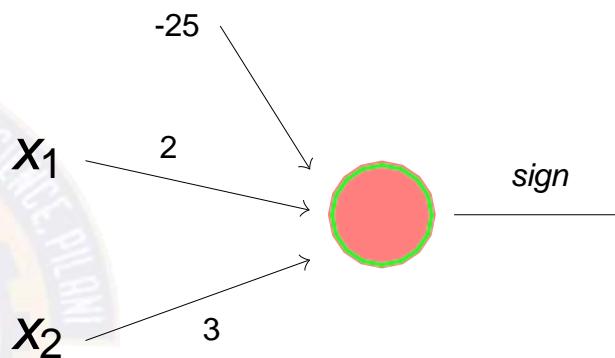
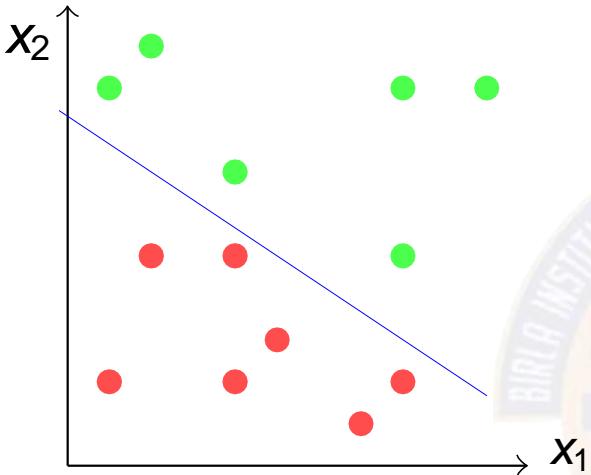
$$2x_1 + 3x_2 - 25 = 0$$



# What about this arrangement?

With chosen *decision boundary*

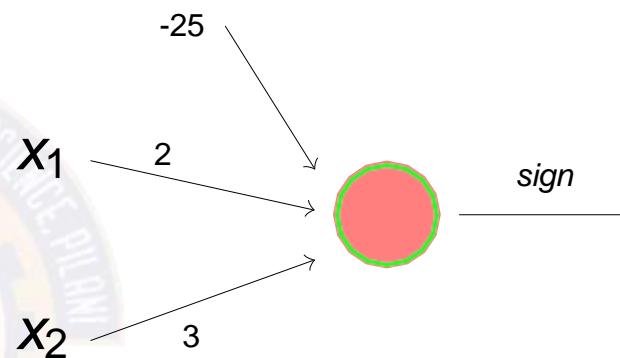
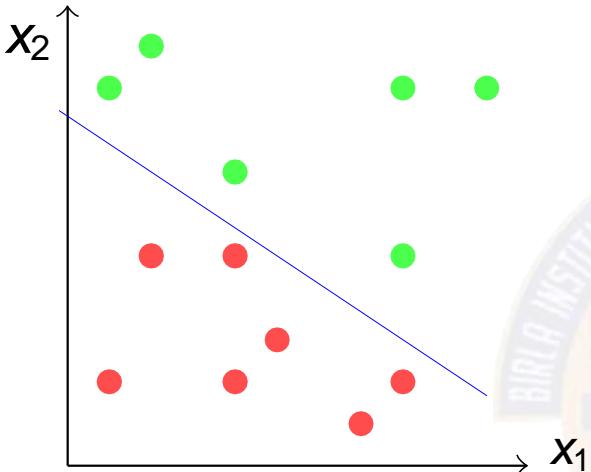
$$2x_1 + 3x_2 - 25 = 0$$



# What about this arrangement?

With chosen *decision boundary*

$$2x_1 + 3x_2 - 25 = 0$$



- This illustration is called as **perceptron**
- Provides a graphical way to represent the linear boundary
- Values  $3, 2, -25$  are its parameters or weights

Given a data

“How to find appropriate parameters?” is an important **issue**

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm: Perceptron training rule

---

- 1 Begin with **random** weights  $w$
  - 2 **repeat**
  - 3     **for each** *misclassified* example **do**
  - 4          $w_i = w_i + \eta(t - o)x_i$
  - 5 **until** *all training examples are correctly classified*;
  - 6 **return**  $w$
-

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?
  - Weight does not change when classification is correct

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?
  - Weight does not change when classification is correct
  - If perceptron outputs -1 when target is +1:

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- **Why would this strategy converge?**
  - Weight does not change when classification is correct
  - If perceptron outputs -1 when target is +1: weight increases ↑

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?
  - Weight does not change when classification is correct
  - If perceptron outputs -1 when target is +1: weight increases ↑
  - If perceptron outputs +1 when target is -1:

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- Why would this strategy converge?
  - Weight does not change when classification is correct
  - If perceptron outputs -1 when target is +1: weight increases ↑
  - If perceptron outputs +1 when target is -1: weight decreases ↓

# Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

---

## Algorithm 2: Perceptron training rule

---

```
1 Begin with random weights  $w$ 
2 repeat
3   for each misclassified example do
4      $w_i = w_i + \eta(t - o)x_i$ 
5 until all training examples are correctly classified;
6 return  $w$ 
```

---

- **Why would this strategy converge?**

- Weight does not change when classification is correct
- If perceptron outputs -1 when target is +1: weight increases ↑
- If perceptron outputs +1 when target is -1: weight decreases ↓

Conversion with perceptron training rule is subject to linear separability of training example and appropriate  $\eta$

# Example

Consider the same data

X <sub>1</sub>	X <sub>2</sub>	Y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red



# Example

Consider the same data

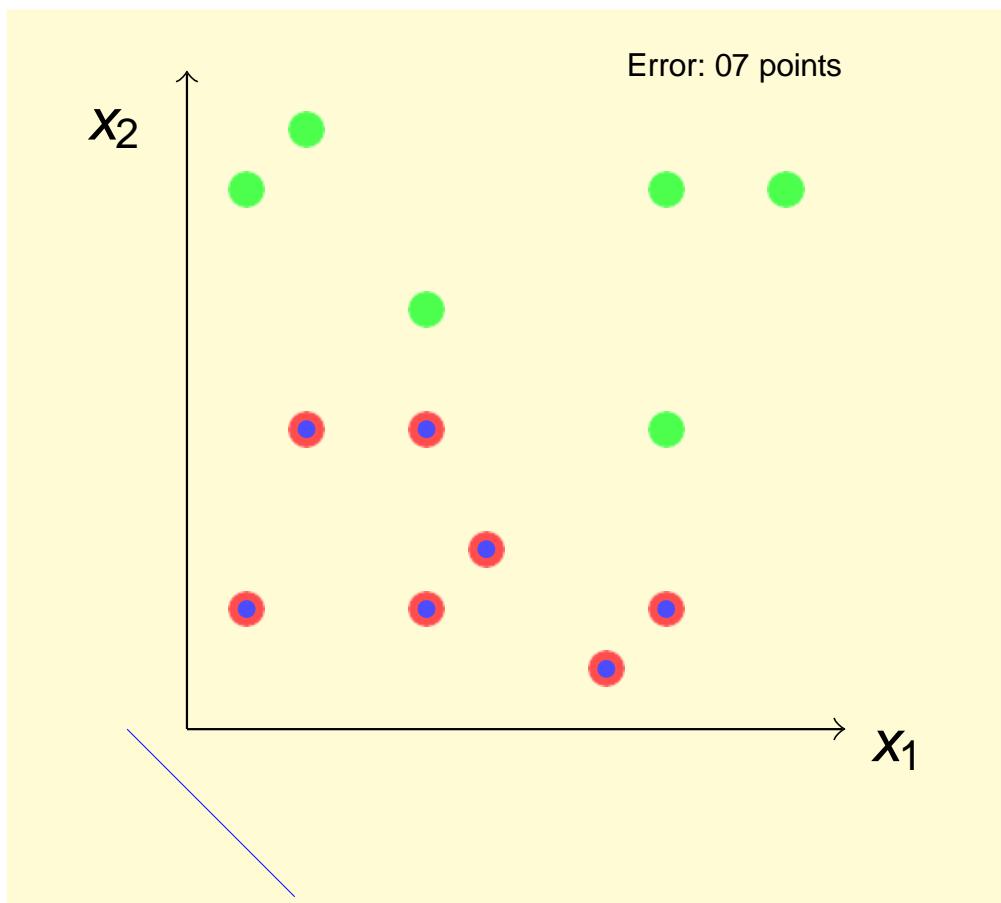
X <sub>1</sub>	X <sub>2</sub>	Y
1	9	Green
10	9	Green
4	7	Green
4	5	Red
5	3	Red
8	9	Green
4	2	Red
2	5	Red
7	1	Red
2	10	Green
8	5	Green
1	2	Red
8	2	Red

$$\eta = 0.01$$

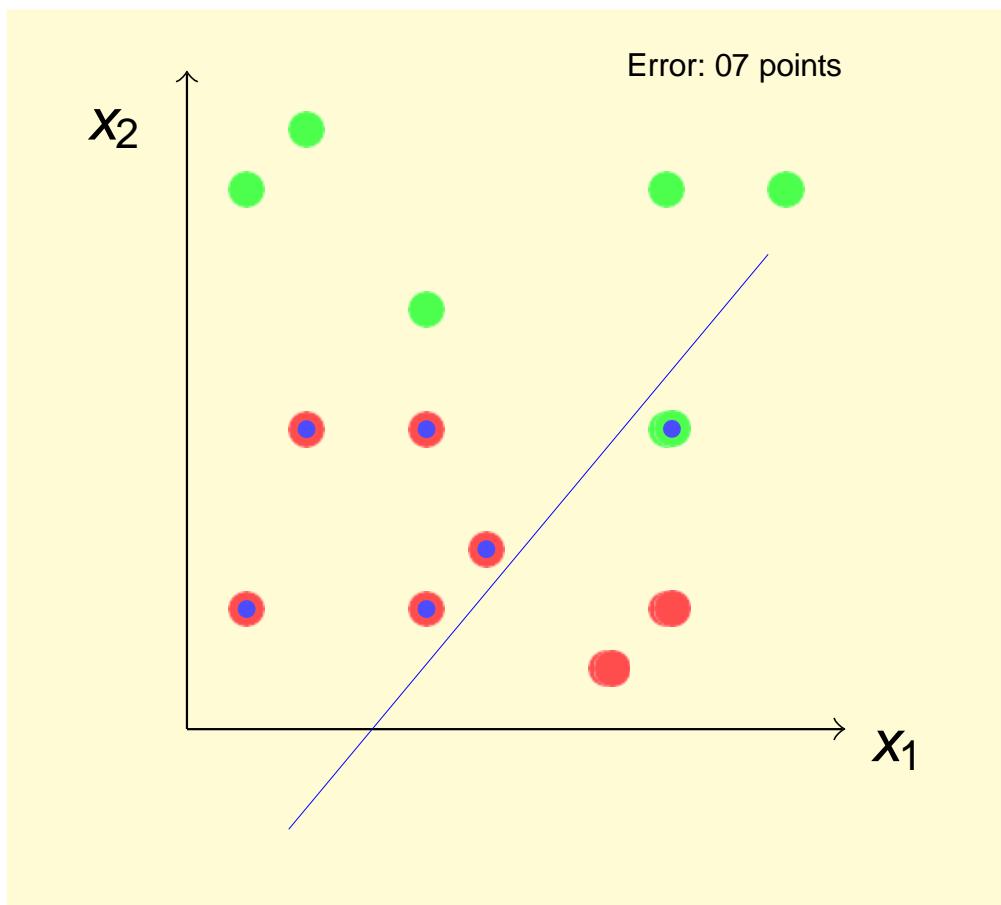
w0=0.500, w1=0.500, w2=0.500	err=7
w0=0.360, w1=-0.120, w2=0.100	err=6
w0=0.300, w1=-0.180, w2=0.060	err=5
w0=0.240, w1=-0.140, w2=0.140	err=4
w0=0.180, w1=-0.200, w2=0.100	err=5
w0=0.120, w1=-0.160, w2=0.180	err=4
w0=0.080, w1=-0.060, w2=0.180	err=5
w0=0.020, w1=-0.120, w2=0.140	err=4
w0=-0.040, w1=-0.180, w2=0.100	err=5
w0=-0.100, w1=-0.140, w2=0.180	err=4
w0=-0.140, w1=-0.040, w2=0.180	err=5
w0=-0.200, w1=-0.100, w2=0.140	err=3
w0=-0.260, w1=-0.160, w2=0.100	err=4
w0=-0.320, w1=-0.120, w2=0.180	err=3
w0=-0.360, w1=-0.020, w2=0.180	err=3
w0=-0.420, w1=-0.080, w2=0.140	err=2
w0=-0.420, w1=-0.080, w2=0.240	err=2
Fourteen more iterations	
w0=-0.900, w1=-0.020, w2=0.180	err=1
w0=-0.900, w1=-0.020, w2=0.240	err=2
w0=-0.920, w1=0.020, w2=0.220	err=2
w0=-0.960, w1=-0.020, w2=0.220	err=3
w0=-0.980, w1=0.020, w2=0.200	err=2
w0=-1.000, w1=0.060, w2=0.180	err=2
w0=-1.040, w1=0.020, w2=0.180	err=0



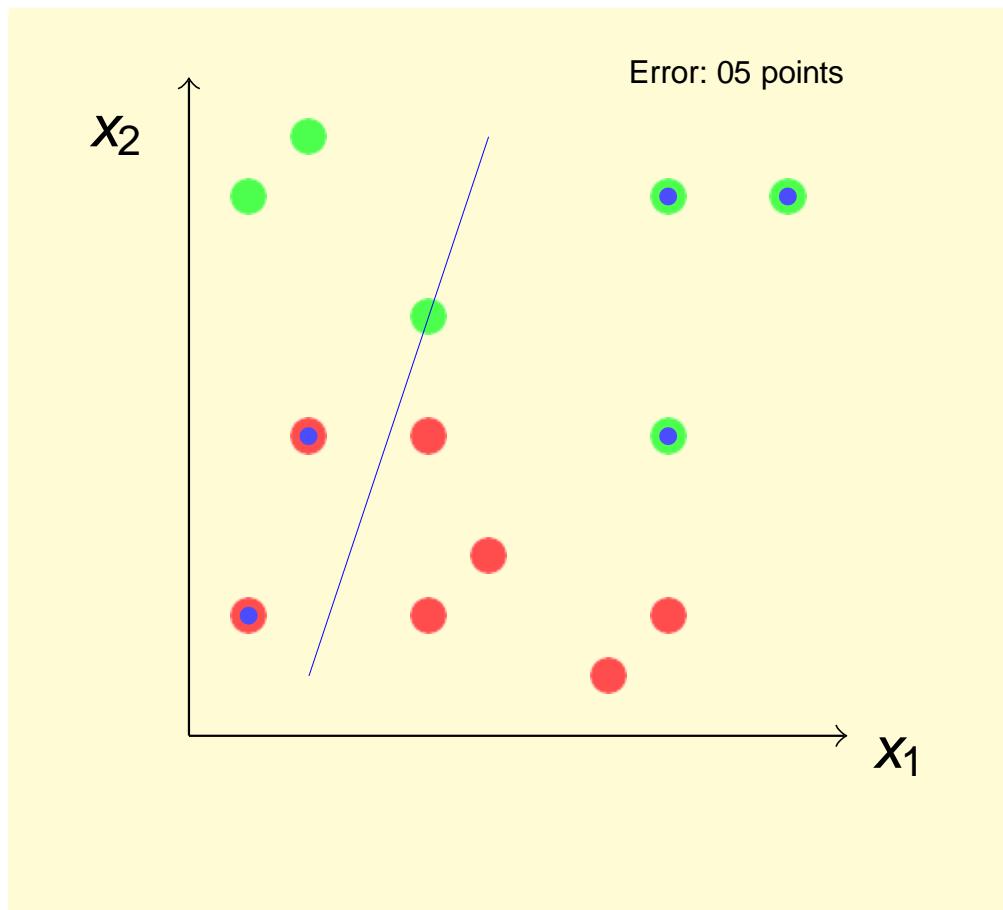
# Visual Interpretation



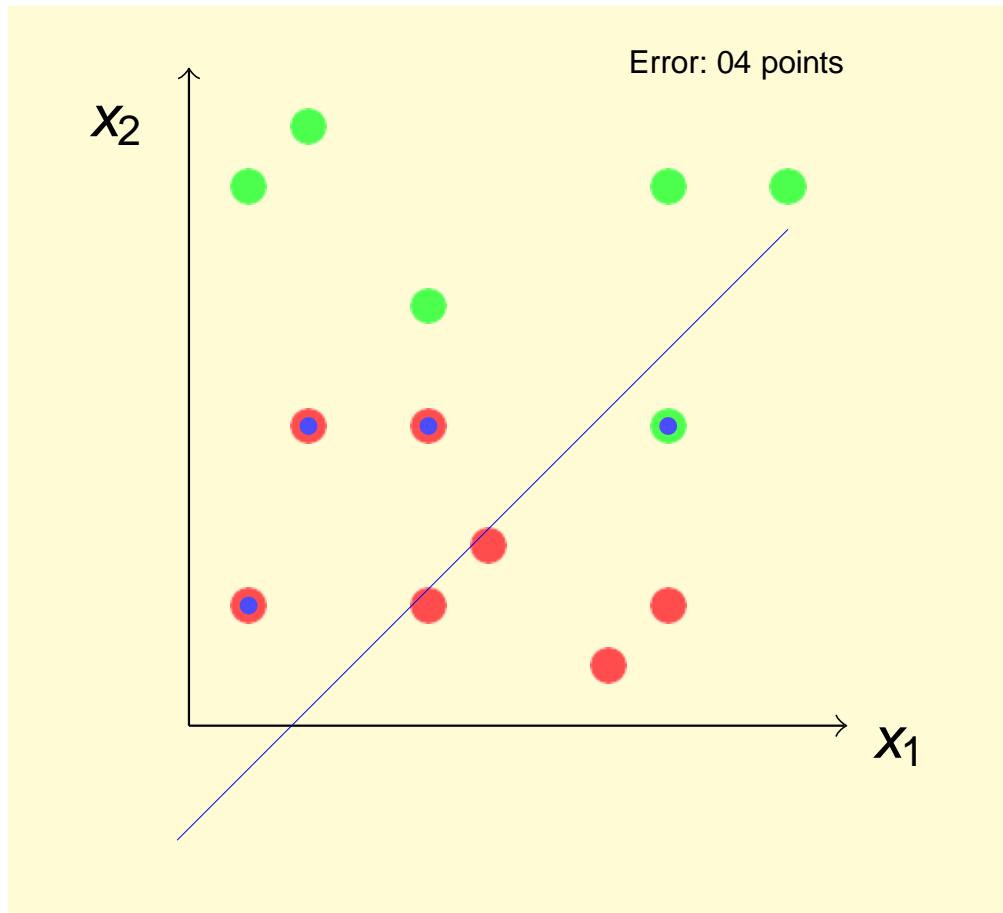
# Visual Interpretation



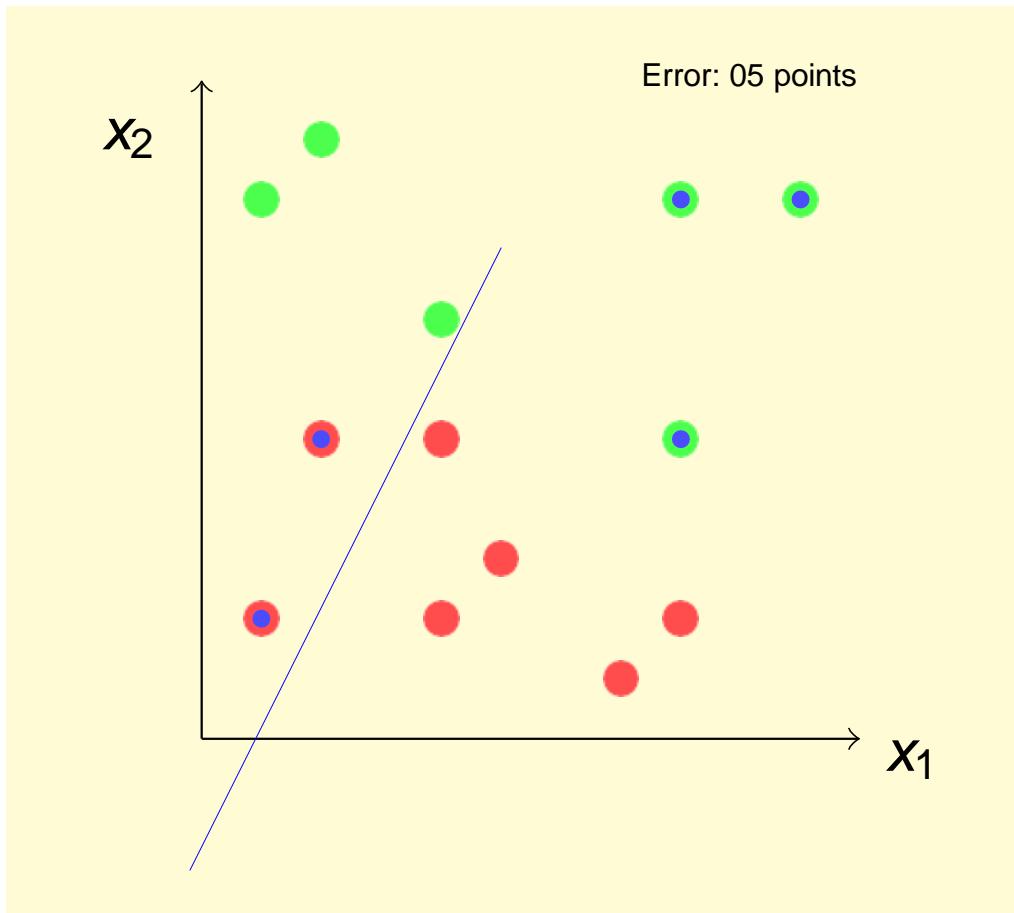
# Visual Interpretation



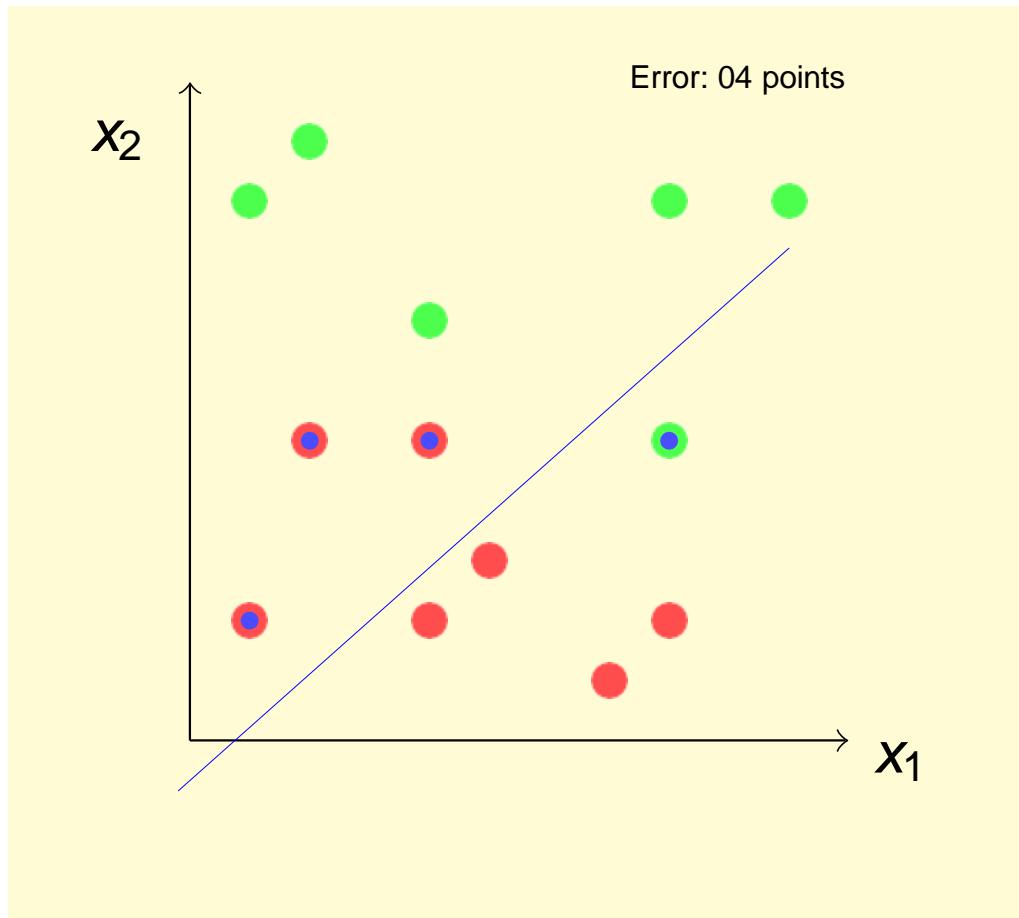
# Visual Interpretation



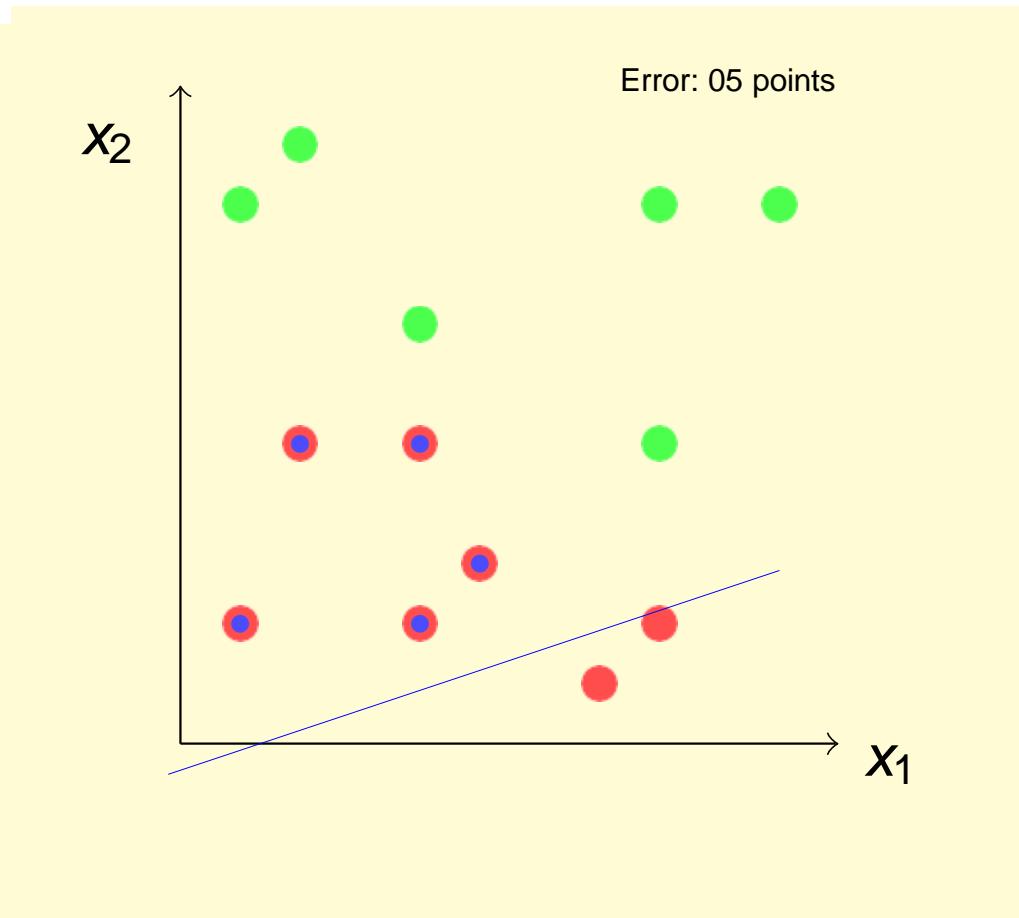
# Visual Interpretation



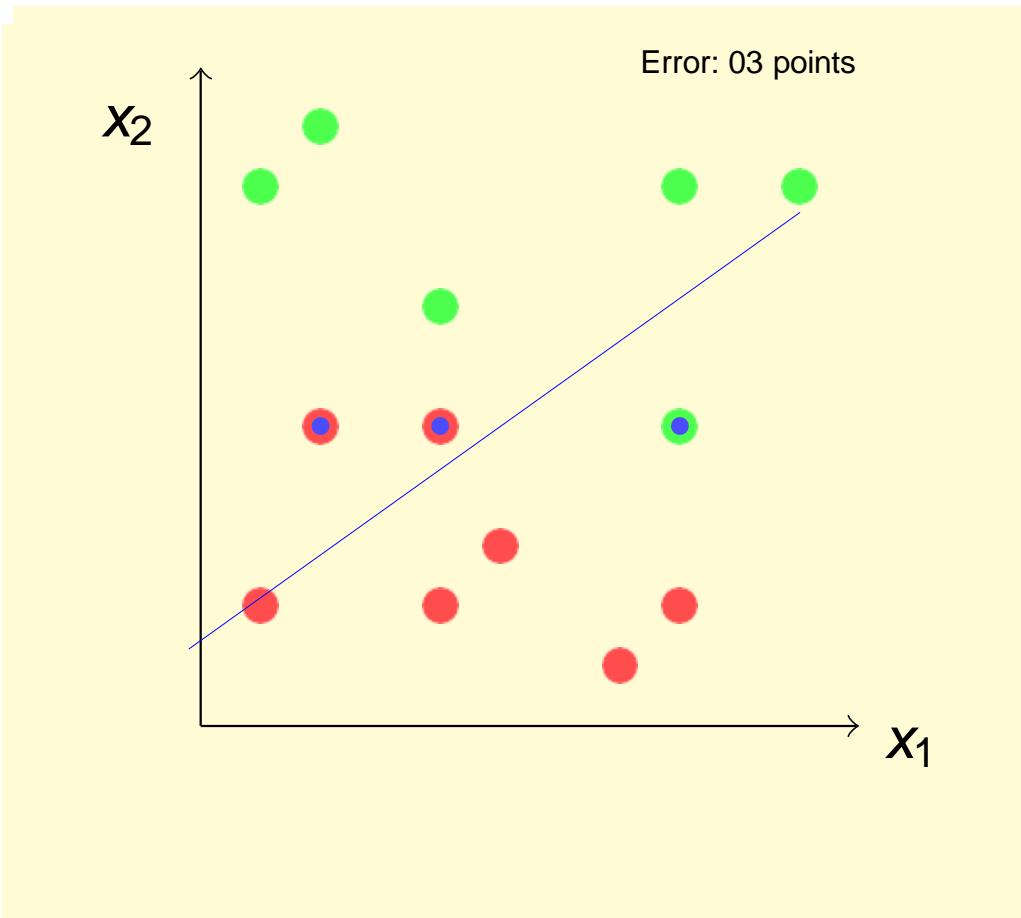
# Visual Interpretation



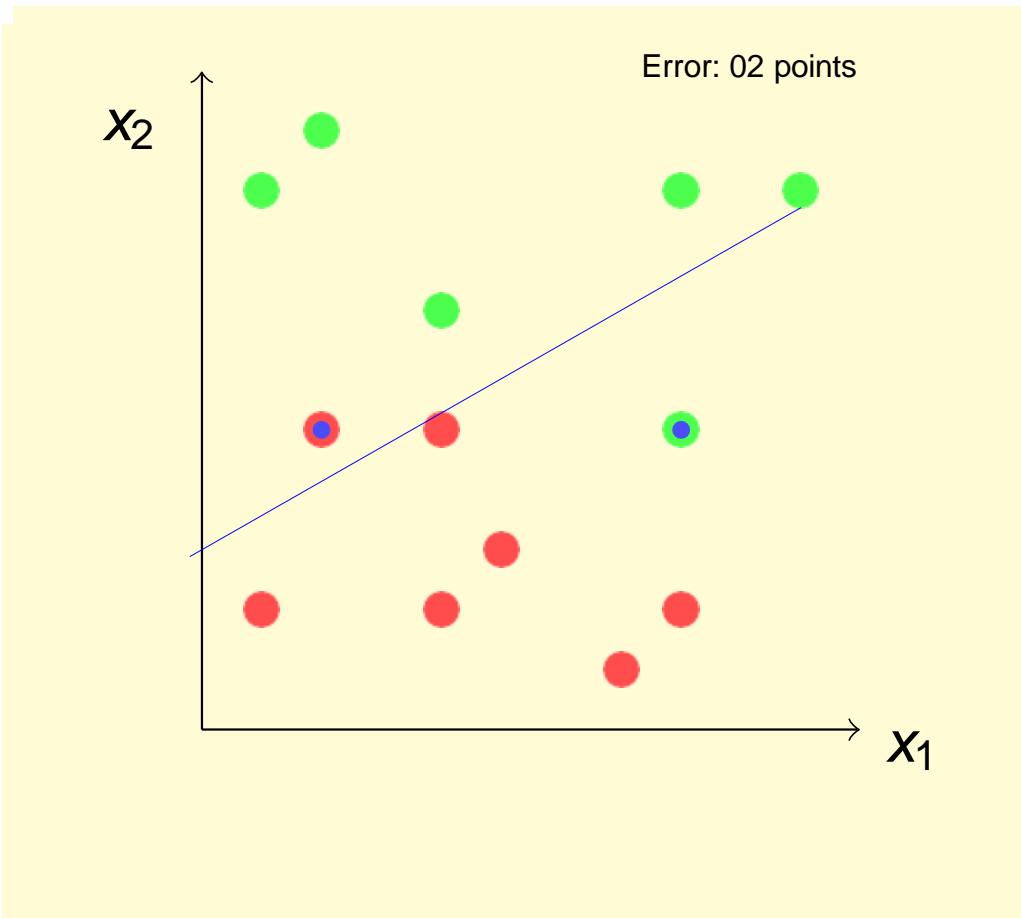
# Visual Interpretation



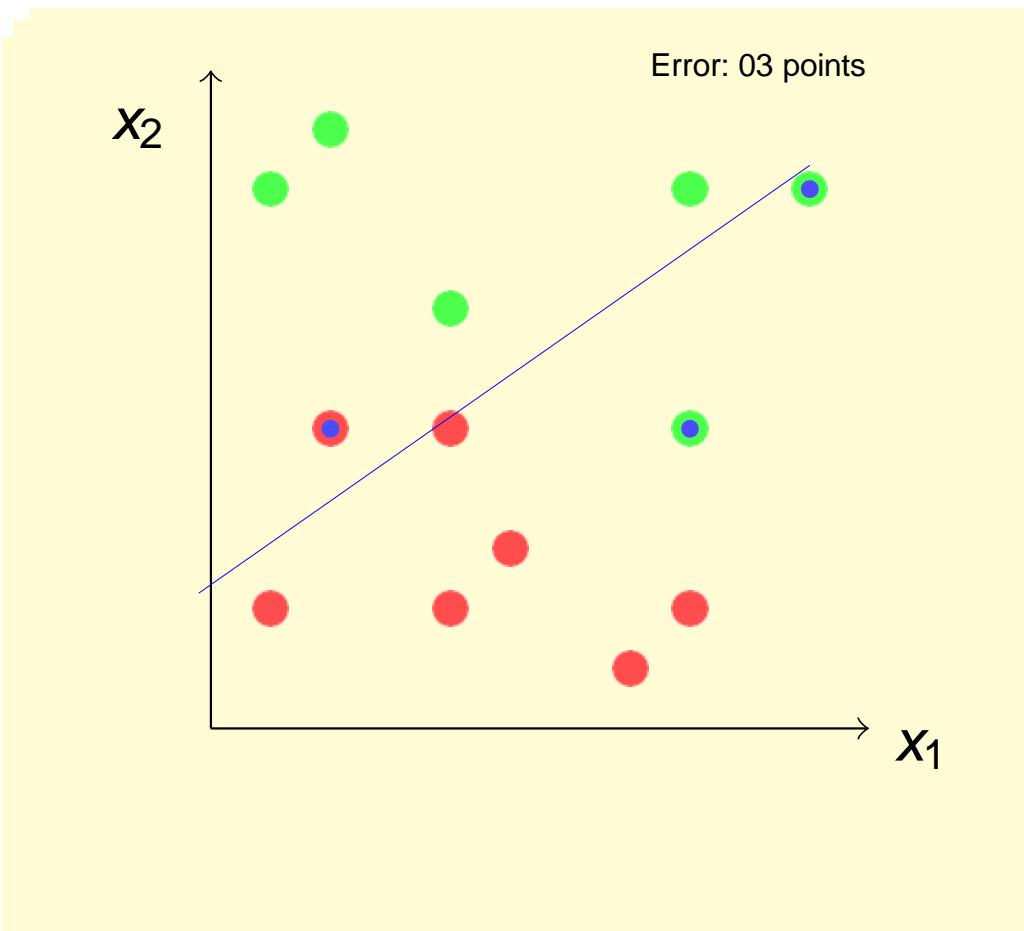
# Visual Interpretation



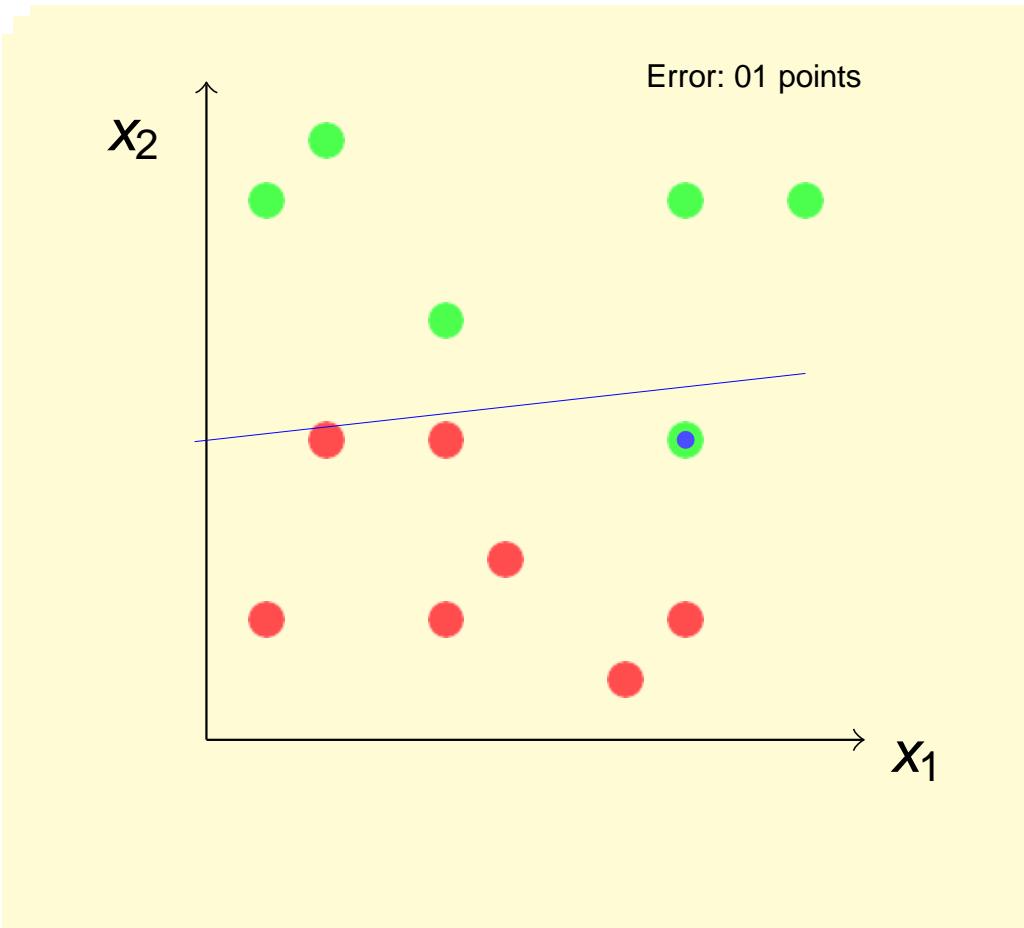
# Visual Interpretation



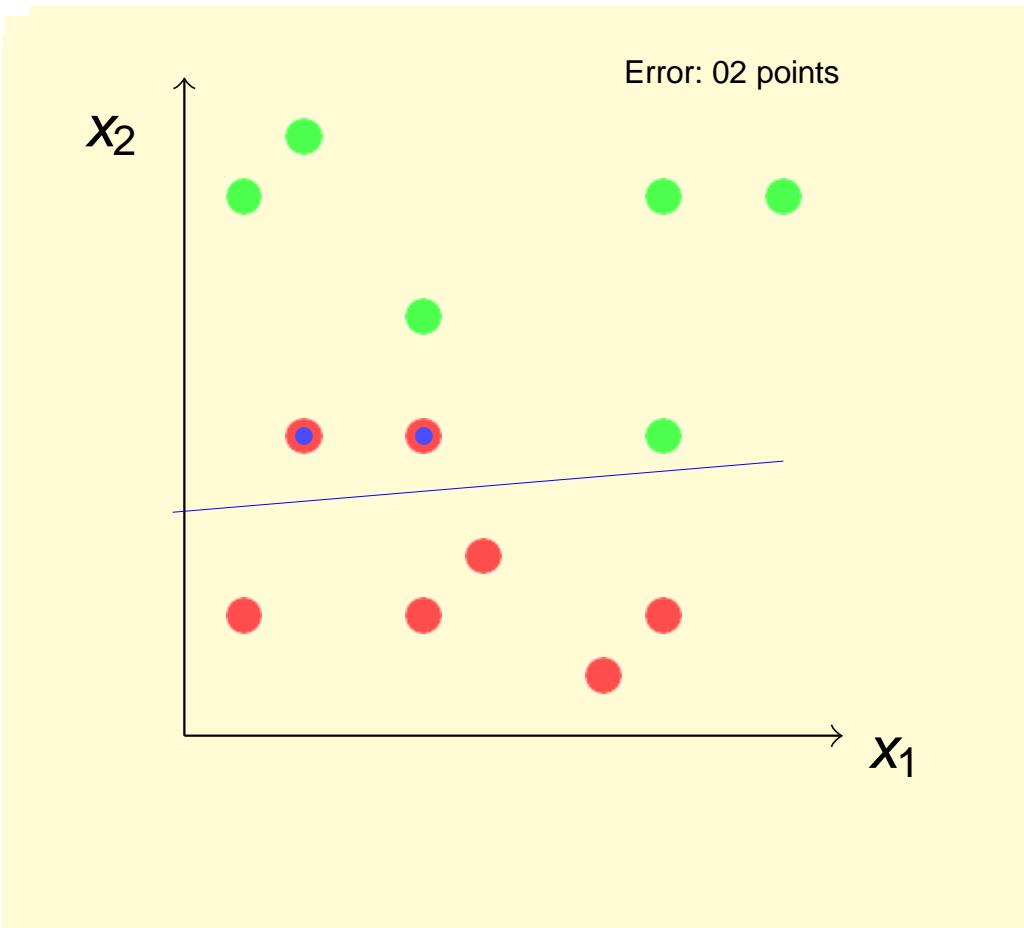
# Visual Interpretation



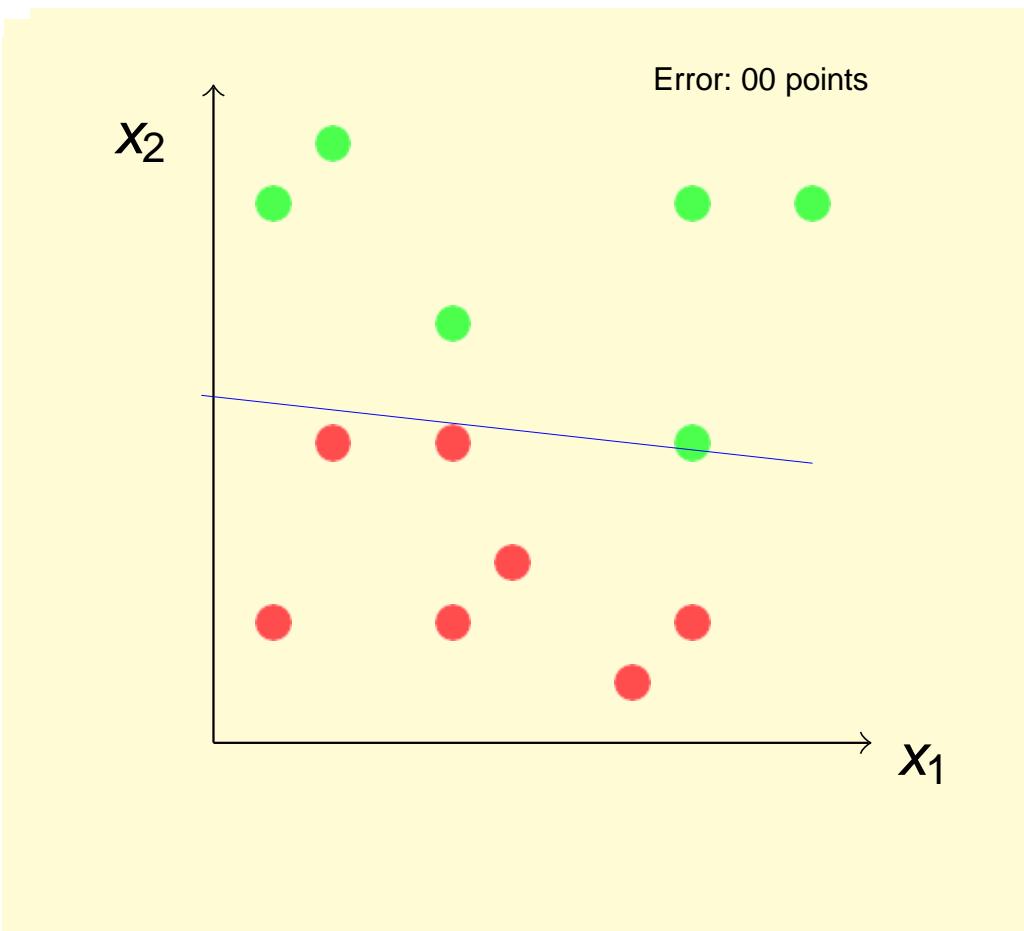
# Visual Interpretation



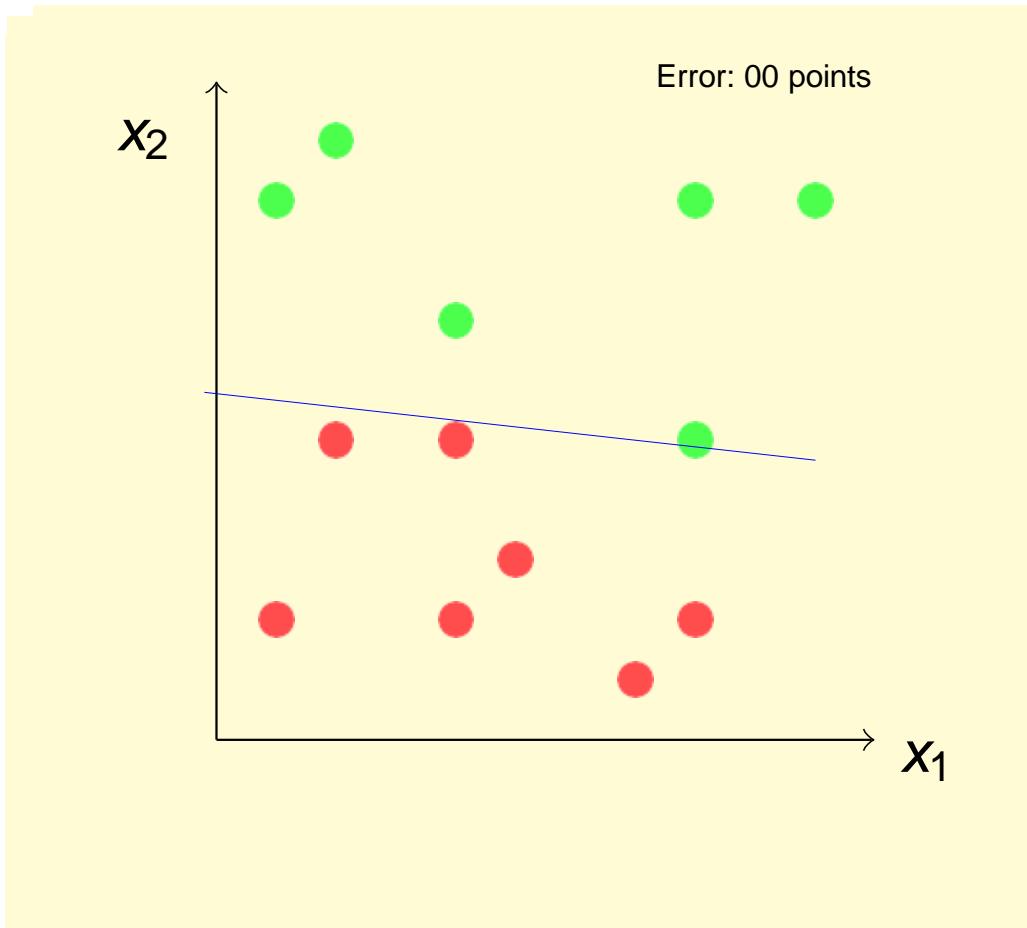
# Visual Interpretation



# Visual Interpretation



# Visual Interpretation



- Conversion is not gradual. (Error is NOT reducing monotonically)
- It is difficult to decide when to stop if data is not linearly separable



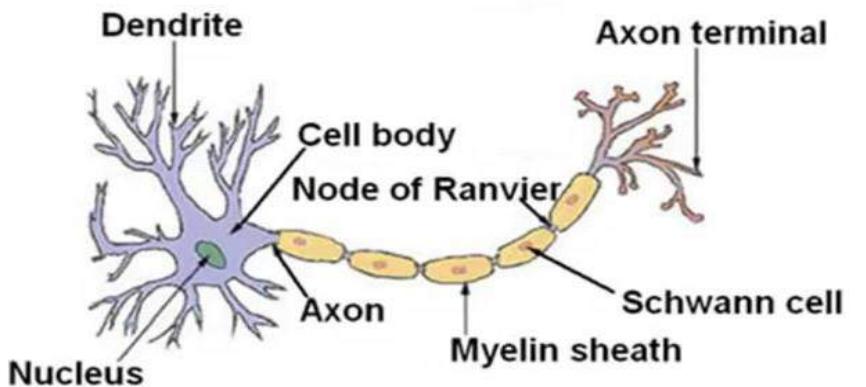
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to Perceptron

# Neural Network (NN)

NN is biologically motivated learning model that mimic human brain

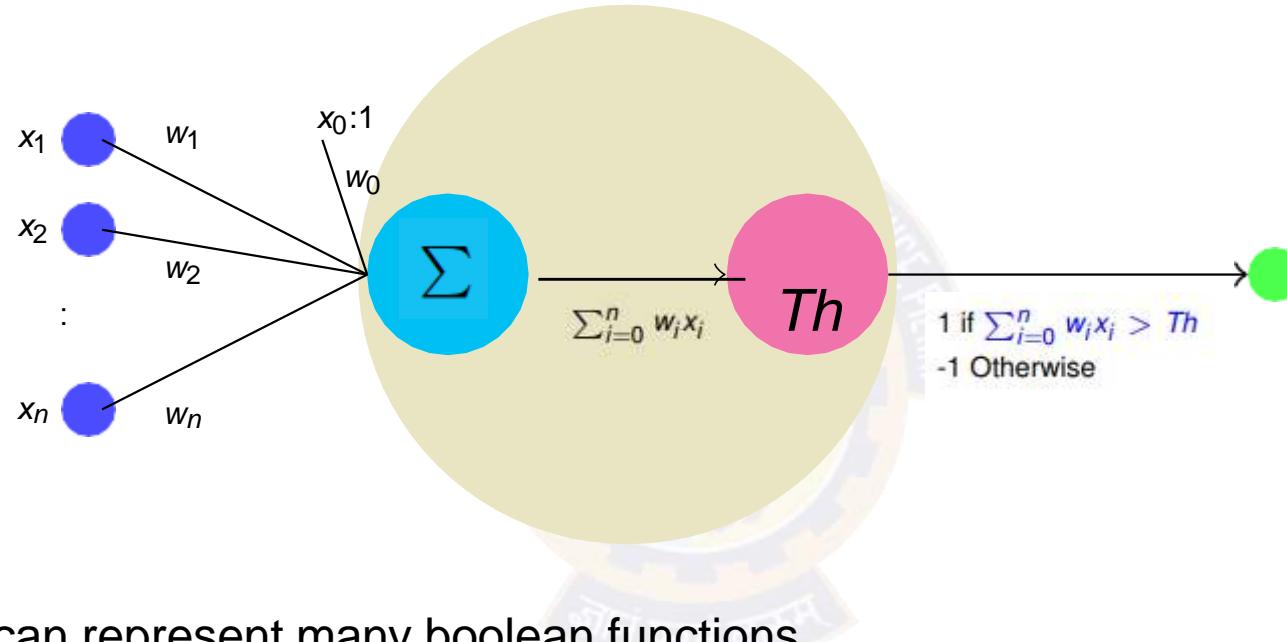
- Started by *W. McCulloch* study on working of neurons in 1943
- MADALINE (1959), an adaptive filter that eliminates echoes on phone lines was the first neural network
- Popularity of Neural Network diminished in 90's but, due to advances in **processing power** and availability of **large data** it again became state-of-the-art



- Cell, Axon, Synapses, Molecules, and Dendrites
- Humans have  $10^{11}$  neurons, each connected to  $10^4$  others, switches in  $10^{-3}$  sec

# A Single Perceptron

## Perceptron representation

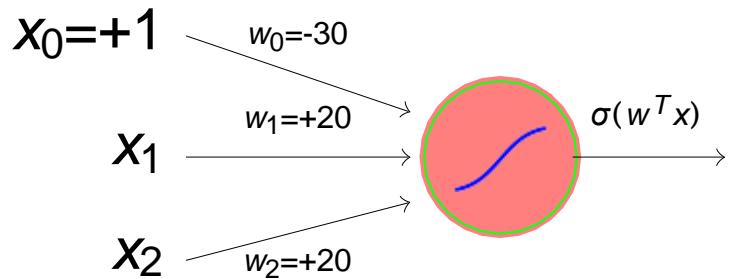


- A single perceptron can represent many boolean functions
- Any  $m$ -of- $n$  function (at least  $m$  of the  $n$  inputs must be true) can be represented by perceptron. OR ( $m=1$ ) and AND ( $m=n$ )

Two layer NN can represent any boolean function (Consider SOP)

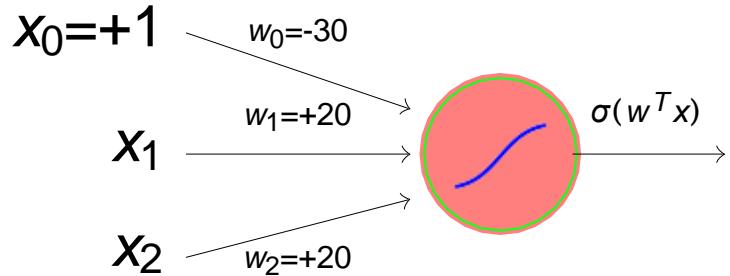
# An Example

Consider a perceptron with output 0/1 as below



# An Example

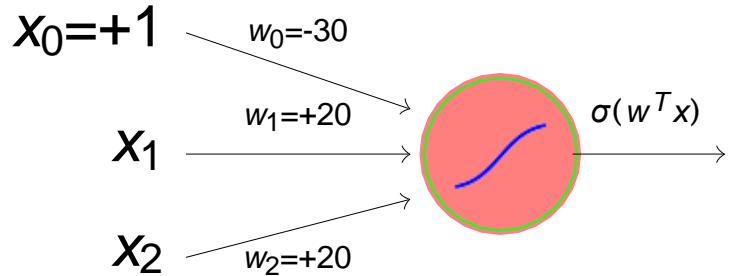
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	

# An Example

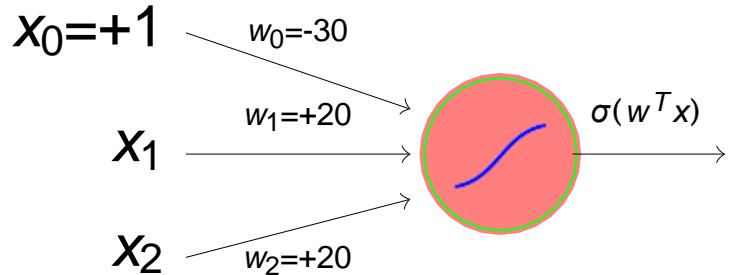
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	

# An Example

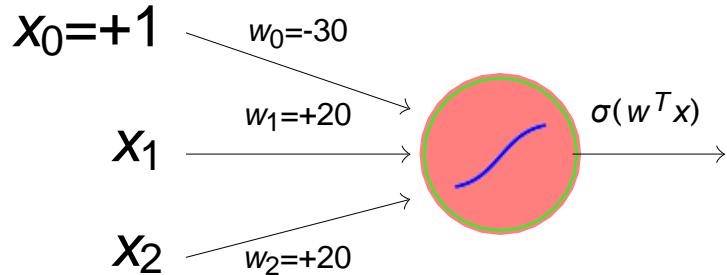
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	

# An Example

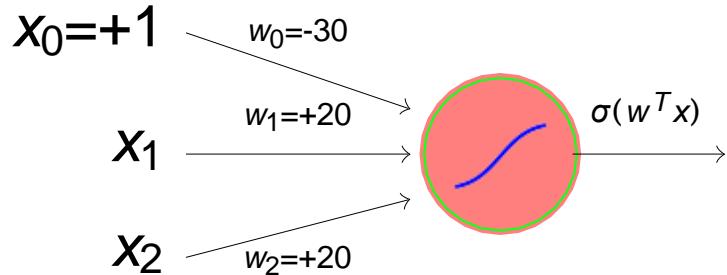
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	0
1	1	1

# An Example

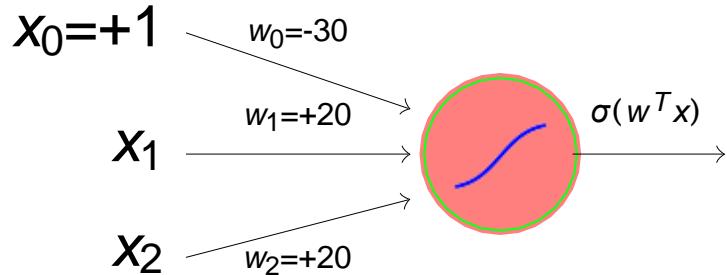
Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	0
1	1	1

# An Example

Consider a perceptron with output 0/1 as below



$x_1$	$x_2$	Output
0	0	0
0	1	0
1	0	0
1	1	1

This perceptron computes logical AND

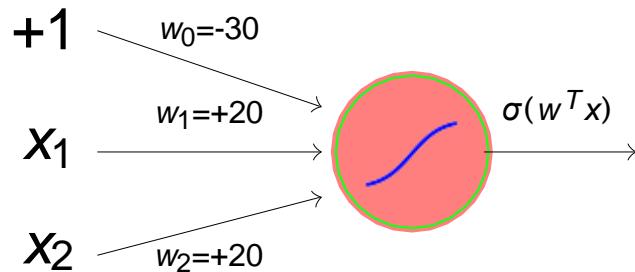
- $w_0=-10$  gives logical OR
- $w_0=10$ ,  $w_1=-20$  with single input gives logical NOT
- XOR is not possible



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Perceptron in Layer and Network

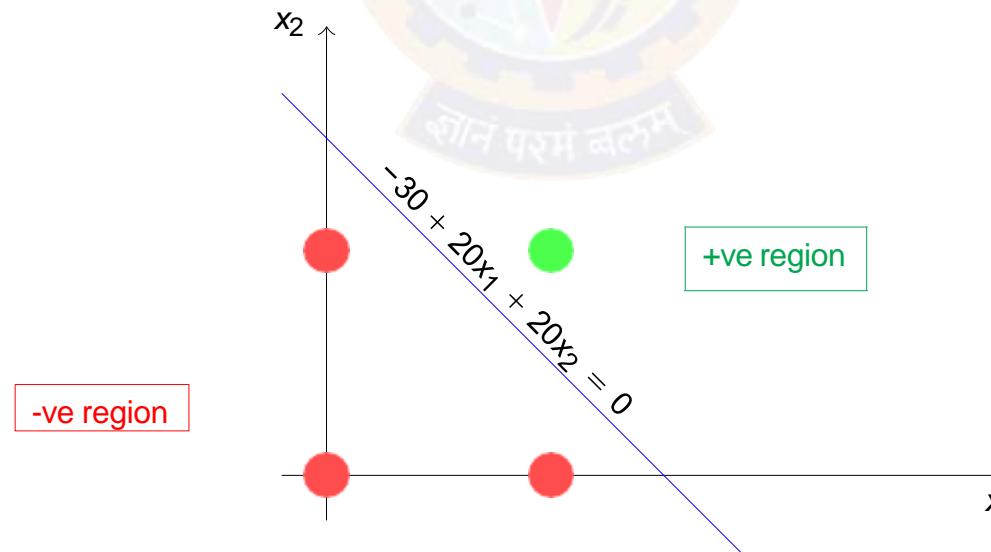
# Essentially it Represents A Decision Boundary



Provides **positive** classification if

$$-30 + 20x_1 + 20x_2 \geq 0$$

Represents a linear decision boundary

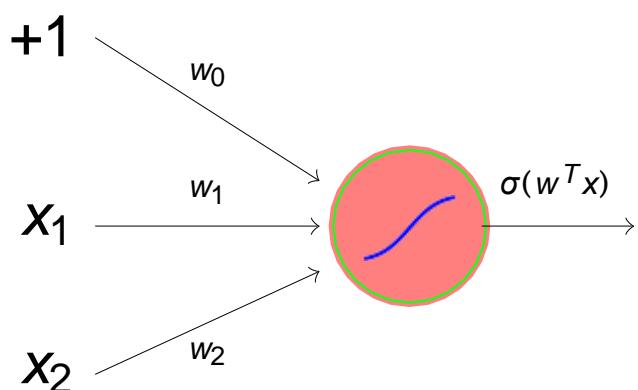


# An Example

Design a perceptron for

$x_1$	$x_2$	Classification
0	0	0
0	1	0
1	0	1
1	1	0

Let us assume following



We have following four equations

$$w_0 + w_1 \times (0) + w_2 \times (0) < 0 \quad (1)$$

$$w_0 + w_1 \times (0) + w_2 \times (1) < 0 \quad (2)$$

$$w_0 + w_1 \times (1) + w_2 \times (0) \geq 0 \quad (3)$$

$$w_0 + w_1 \times (1) + w_2 \times (1) < 0 \quad (4)$$

By (1)  $w_0 < 0$  so let  $w_0 = -1$

By (2)  $w_0 + w_2 < 0$  so let  $w_2 = -1$

By (3)  $w_0 + w_1 \geq 0$  so let  $w_1 = 1.5$

By (4)  $w_0 + w_1 + w_2 < 0$  that is valid

So  $(w_0, w_1, w_2) = (-1, -1, 1.5)$

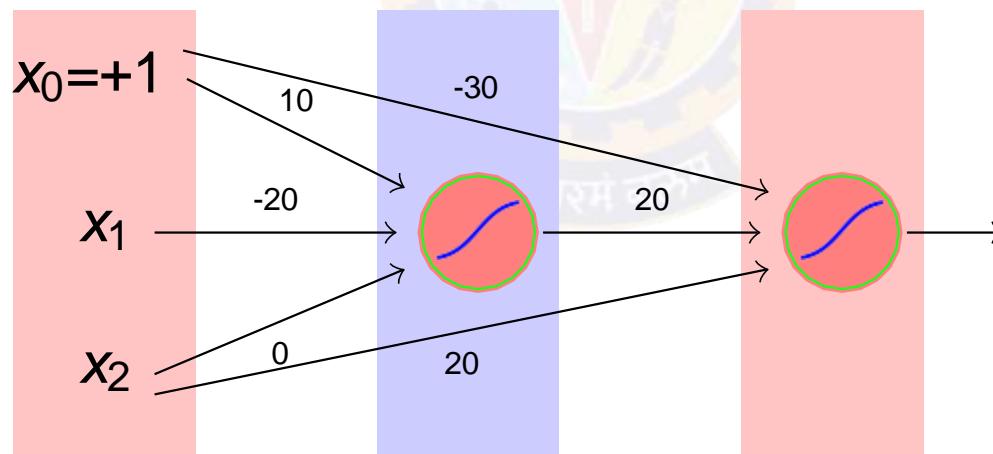
Other possibilities are also there

# An Example

Design a neural network for

$x_1$	$X_2$	Classification
0	0	0
0	1	0
1	0	1
1	1	0

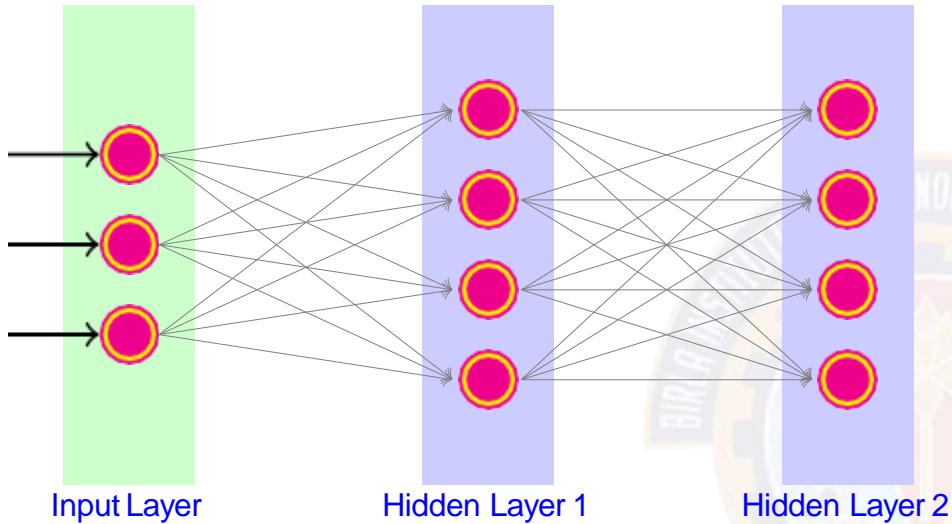
$x_1$	$X_2$	$\bar{x}_2$	$(x_1) \text{AND} (\bar{x}_2)$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0



This arrangement is mostly avoided, as training is more challenging

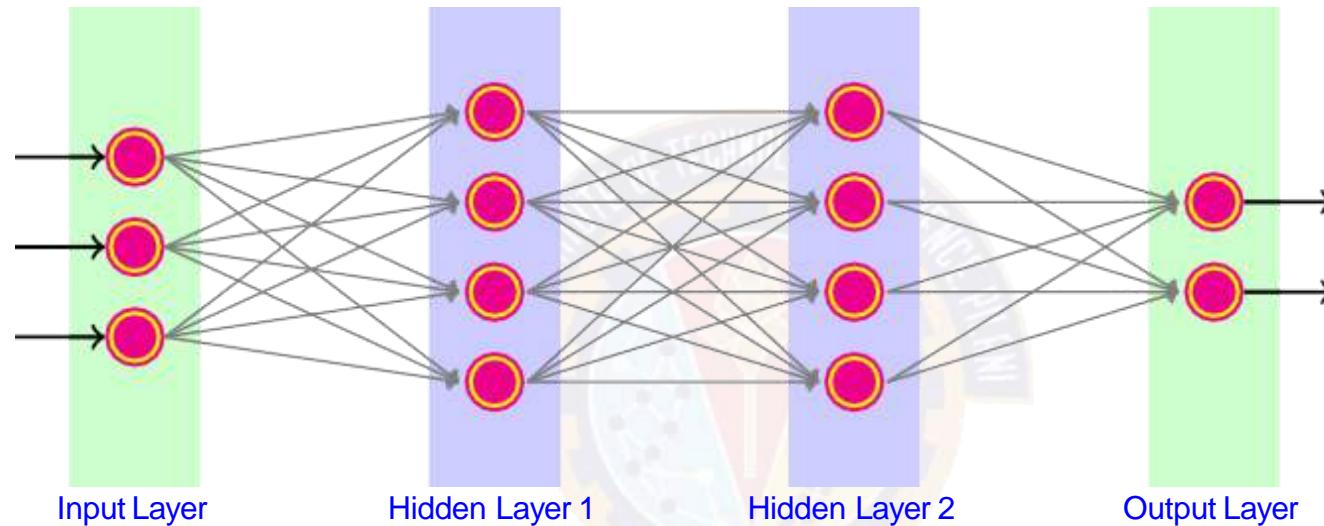
# Neural Network

When neurons are interconnected in layers



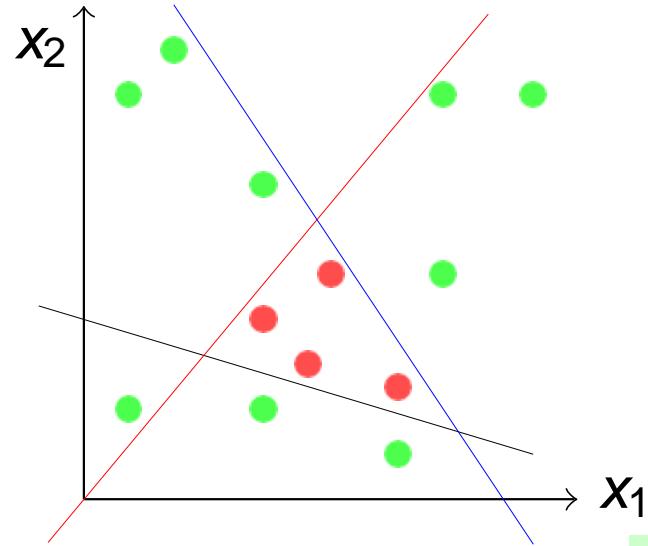
# Neural Network

When neurons are interconnected in layers



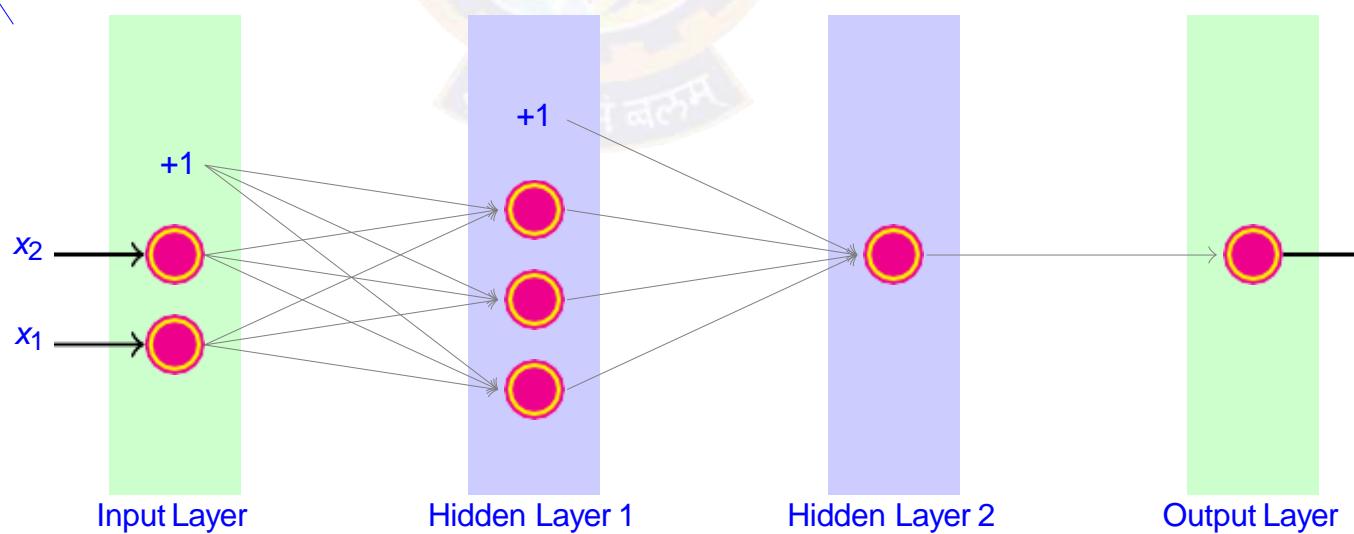
- Number of layers may differ
- Nodes in each intermediate layers may also differ
- Multiple output neurons are used for different class
- **Two levels deep** NN can represent any boolean function

# More Example: Design NN for the following data



Whether it is green?

Red-line	Blue-line	Black-line	Color
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



# Neural Network Applications

- NN is appropriate for problems with the following characteristics:
  - Instances are provided by many attribute-value pairs (more data)
  - The target function output may be discrete-valued, real-valued, or a vector of several real or discrete valued attributes
  - The training examples may contain errors
  - Long training times are acceptable
  - Fast evaluation of the target function may be required
  - The ability of humans to understand the learned target function is not important



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# Perceptron Training (Delta Rule)

# Perceptron Training (delta rule)

- Delta rule converges to a best-fit approximation of the target
- Uses gradient descent
- Consider un-thresholded perceptron, i.e., output  $o(\vec{x}) = \vec{w} \cdot \vec{x}$
- Training error is defined as

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- Gradient would specify direction of steepest increase

$$\nabla E(\vec{w}) = \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- Weights can be learned as  $w_i = w_i - \eta \frac{\partial E}{\partial w_i}$

- It can be seen that  $\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$

# Perceptron Training (delta rule)

---

## Algorithm: Gradient Descent ( $D, \eta$ )

---

```
1 Initialize  $w_i$  with random weights
2 repeat
3   For each  $w_i$ , initialize  $\delta w_i = 0$ 
4   for each training example  $d \in D$  do
5     Compute output  $o$  using model for  $d$  whose target is  $t$ 
6     For each  $w_i$ , update  $\delta w_i = \delta w_i + \eta(t-o)x_i$ 
7   For each  $w_i$ , set  $w_i = w_i + \delta w_i$ 
8 until termination condition is met,
9 return  $w$ 
```

---

# Perceptron Training (delta rule)

---

## Algorithm: Gradient Descent ( $D, \eta$ )

---

```
1 Initialize  $w_i$  with random weights
2 repeat
3   For each  $w_i$ , initialize  $\delta w_i = 0$ 
4   for each training example  $d \in D$  do
5     Compute output  $o$  using model for  $d$  whose target is  $t$ 
6     For each  $w_i$ , update  $\delta w_i = \delta w_i + \eta(t-o)x_i$ 
7   For each  $w_i$ , set  $w_i = w_i + \delta w_i$ 
8 until termination condition is met,
9 return  $w$ 
```

---

- A date item  $d \in D$ , is supposed to be multidimensional  $d = (x_1, x_2, \dots, x_n, t)$
- Algorithm converges toward the minimum error hypothesis.

# Perceptron Training (delta rule)

---

## Algorithm: Gradient Descent ( $D, \eta$ )

---

```
1 Initialize  $w_i$  with random weights
2 repeat
3   For each  $w_i$ , initialize  $\delta w_i = 0$ 
4   for each training example  $d \in D$  do
5     Compute output  $o$  using model for  $d$  whose target is  $t$ 
6     For each  $w_i$ , update  $\delta w_i = \delta w_i + \eta(t-o)x_i$ 
7   For each  $w_i$ , set  $w_i = w_i + \delta w_i$ 
8 until termination condition is met,
9 return  $w$ 
```

---

- A date item  $d \in D$ , is supposed to be multidimensional  $d = (x_1, x_2, \dots, x_n, t)$
- Algorithm converges toward the minimum error hypothesis.
- Linear programming can also be an approach

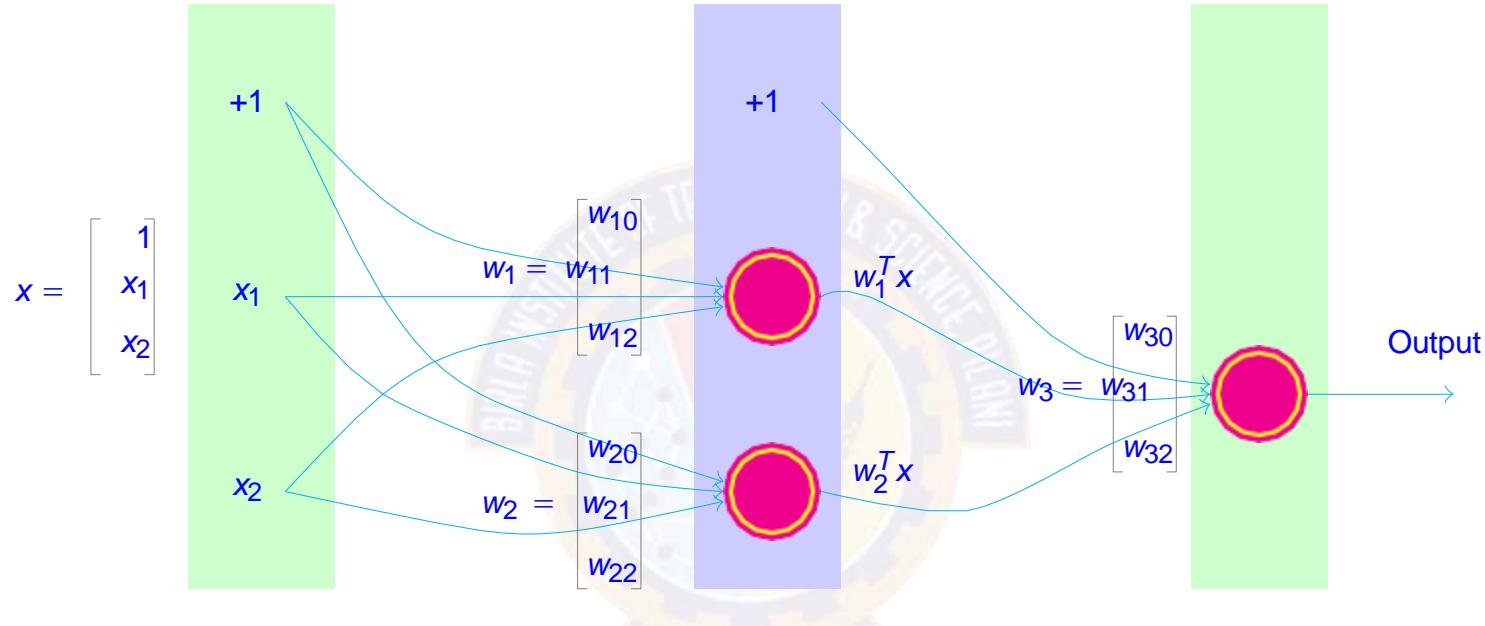


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Importance of Non-Linearity

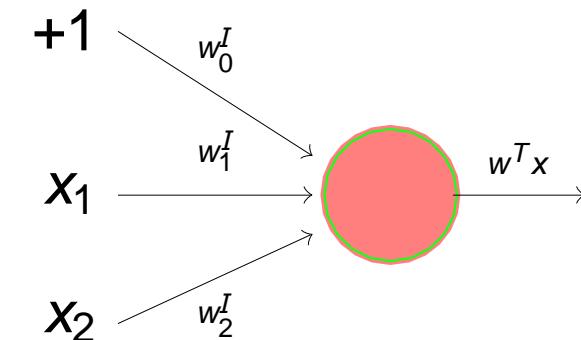
# Linear Activation is Not Much Interesting

NN with perceptrons have limited capability, even with many layers



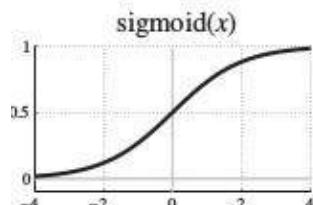
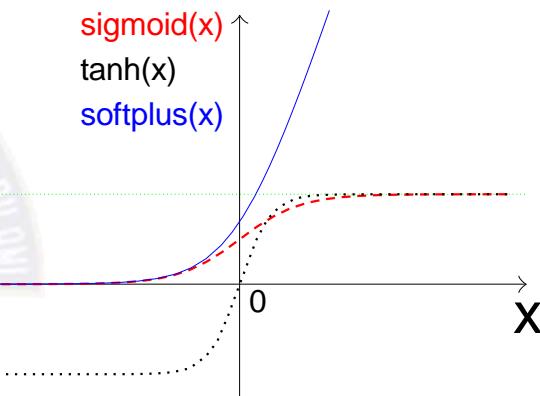
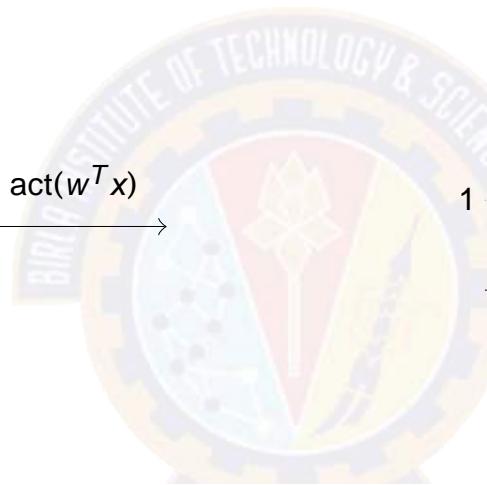
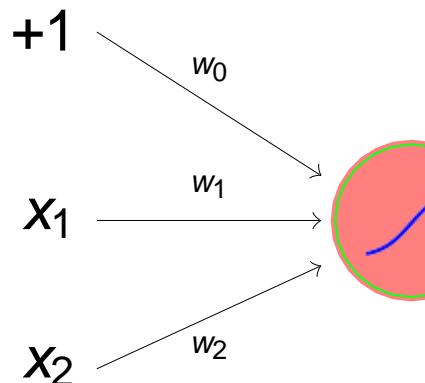
$$\begin{aligned} \text{Output} &= w_{30} \times 1 + w_{31} \times (w_1^T x) + w_{32} \times (w_2^T x) \\ &= w_{30} \times 1 + w_{31} \times [w_{10} \times 1 + w_{11} \times x_1 + w_{12} \times x_2] \\ &\quad + w_{32} \times [w_{20} \times 1 + w_{21} \times x_1 + w_{22} \times x_2] \\ &= (w_{30} + w_{31}w_{10} + w_{32}w_{20}) + (w_{31}w_{11} + w_{32}w_{21}) \times x_1 \\ &\quad + (w_{31}w_{12} + w_{32}w_{22}) \times x_2 \\ &= w'_0 + w'_1 \times x_1 + w'_2 \times x_2 \end{aligned}$$

Expression of single perceptron

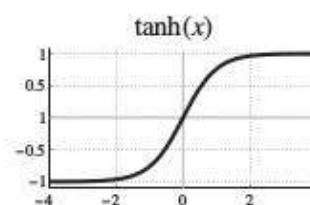


# Neuron

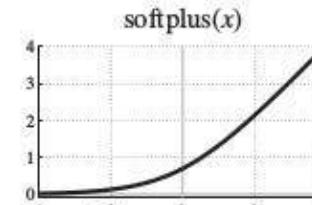
Neuron uses nonlinear **activation functions** (**sigmoid, tanh, ReLU, softplus etc.**) at the place of thresholding



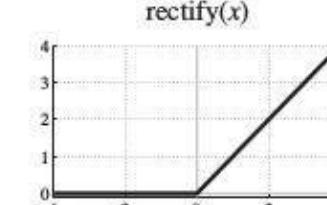
$$h(x) = \frac{1}{1 + \exp(-x)}$$



$$h(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



$$h(x) = \log(1 + \exp(x))$$



$$h(x) = \max(0, x)$$



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# Backpropagation

# Multilayer Networks and Backpropagation

- Single perceptron can only express linear decision surface
- We need units whose output is a nonlinear function of its inputs AND is also differentiable (Use Neuron not Perceptron)

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x})$$

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

- **Backpropagation** algorithm learns the weights for a fixed set of units and interconnections
- It employs **gradient descent** to minimize the error between the network output values and the target values for these outputs
- Let Error function is redefined as

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

# Backpropagation (for 2 layers)

---

**Algorithm:** Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

---

- 1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers
  - 2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$
  - 3 **repeat**
  - 4     **for each**  $\langle \vec{x}, \vec{t} \rangle \in D$  **do**
  - 5          $o_u = \text{get output from network } \forall \text{unit } u$
  - 6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**
  - 7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**
  - 8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$
  - 9 **until** converge;
- 

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$

# Backpropagation (for 2 layers)

---

**Algorithm:** Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

---

- 1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers
- 2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$
- 3 **repeat**
- 4     **for each**  $\langle \vec{x}, \vec{t} \rangle \in D$  **do**
- 5          $o_u = \text{get output from network } \forall \text{unit } u$
- 6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**
- 7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**
- 8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$
- 9 **until** converge;

---

- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$
- For a single training example  $E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$

# Backpropagation (for 2 layers)

---

**Algorithm:** Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

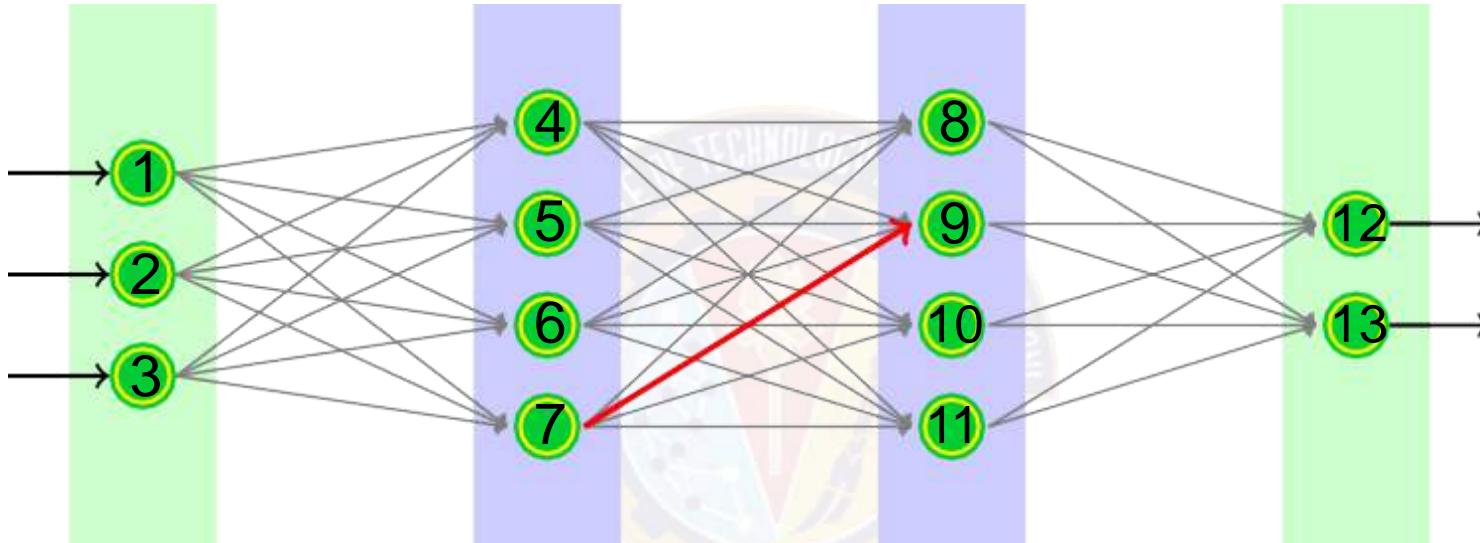
---

1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers  
2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$   
3 **repeat**  
4     **for each**  $\langle \vec{x}, \vec{t} \rangle \in D$  **do**  
5          $o_u = \text{get output from network } \forall \text{unit } u$   
6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**   
7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**   
8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$   
9 **until** converge;

---

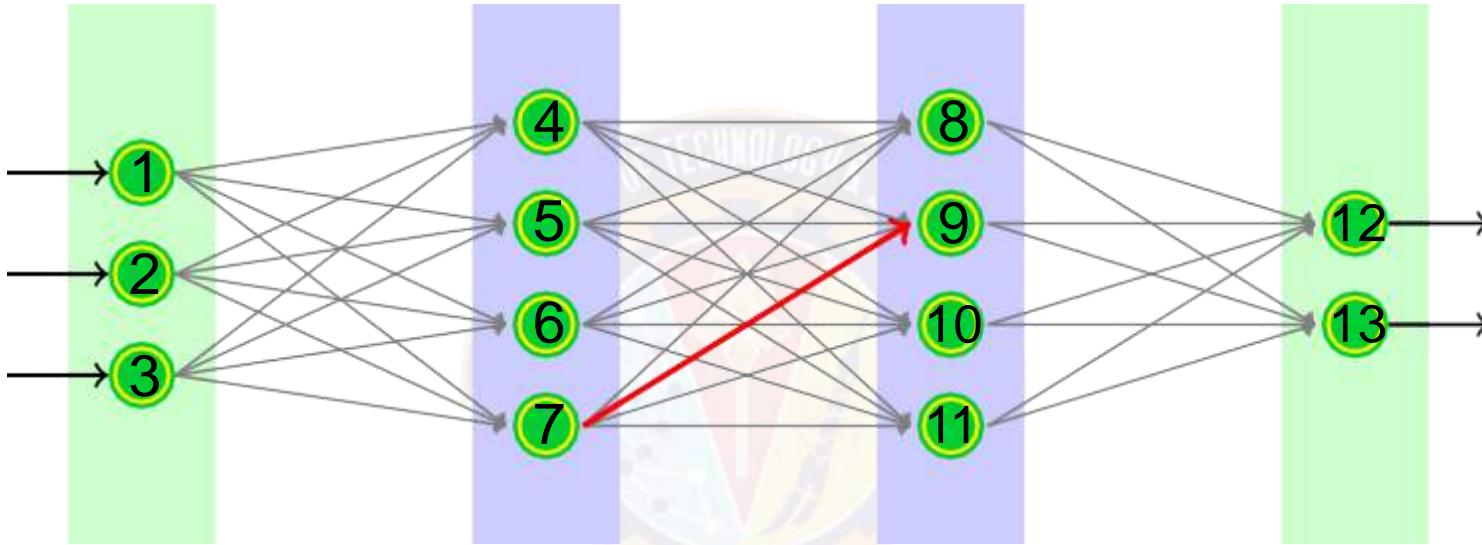
- Recall error function is  $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$
- For a single training example  $E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$
- Weight  $w_{ji}$  is updated by adding  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$

# Conventions Over The Network



$x_{ji}$     $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

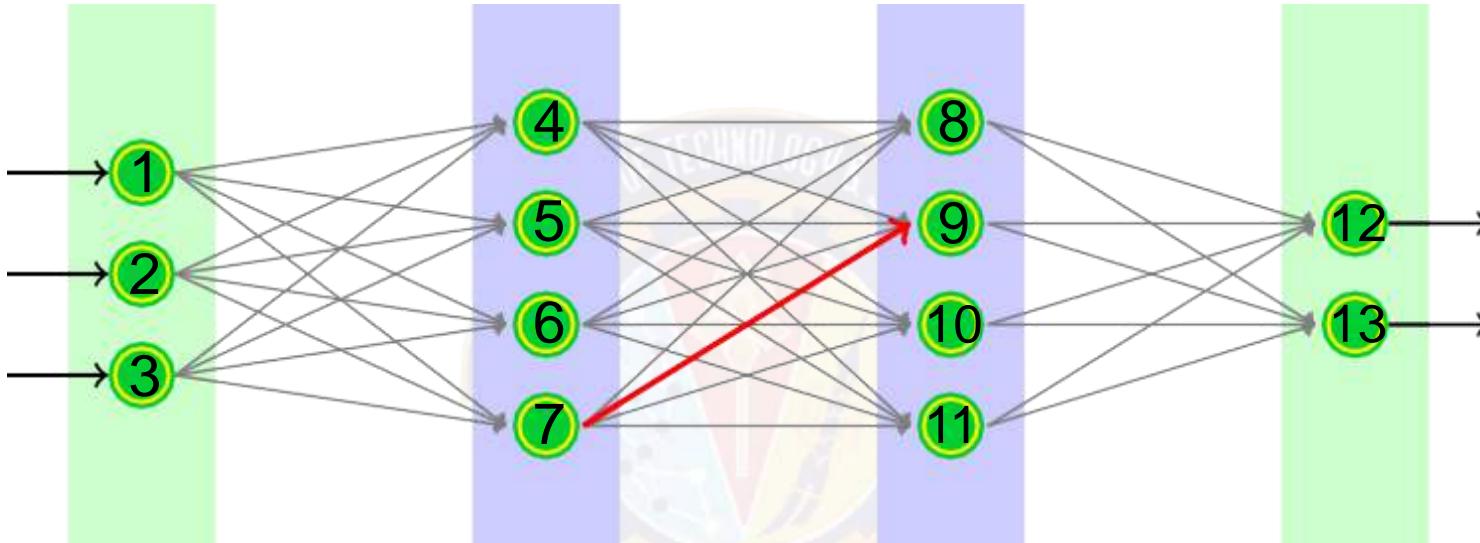
# Conventions Over The Network



$x_{ji}$     $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$    weight associated with  $i$  th input to unit  $j$

# Conventions Over The Network

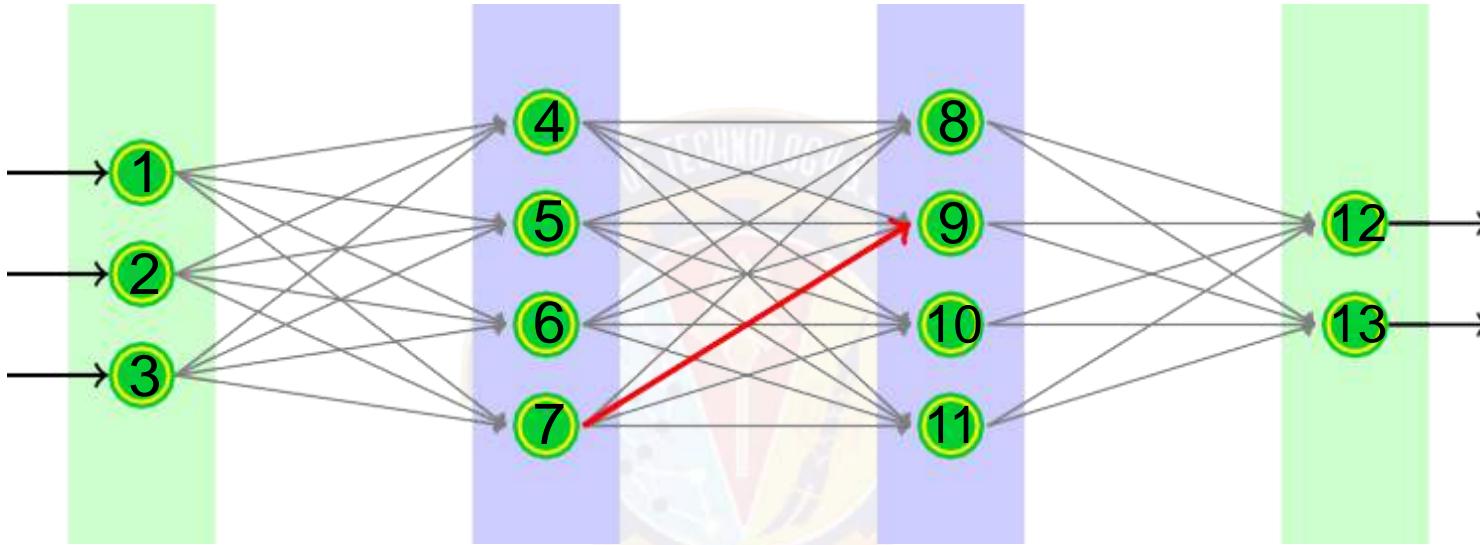


$x_{ji}$   $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$  weight associated with  $i$  th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

# Conventions Over The Network



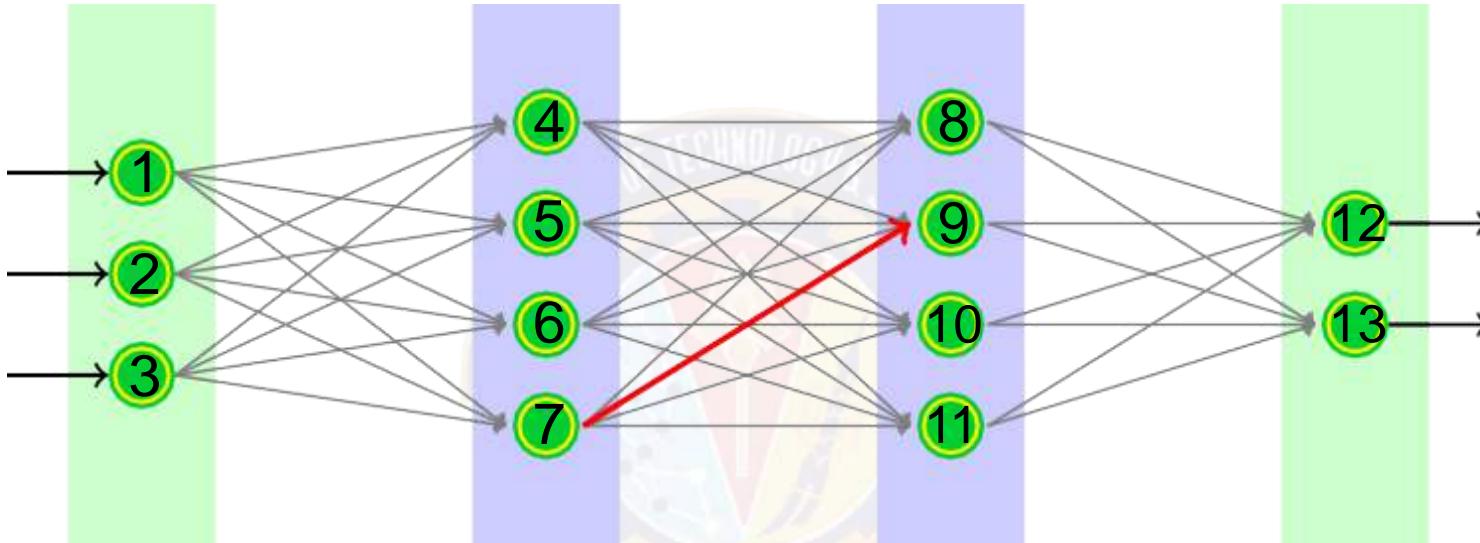
$x_{ji}$   $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$  weight associated with  $i$  th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

# Conventions Over The Network



$x_{ji}$   $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

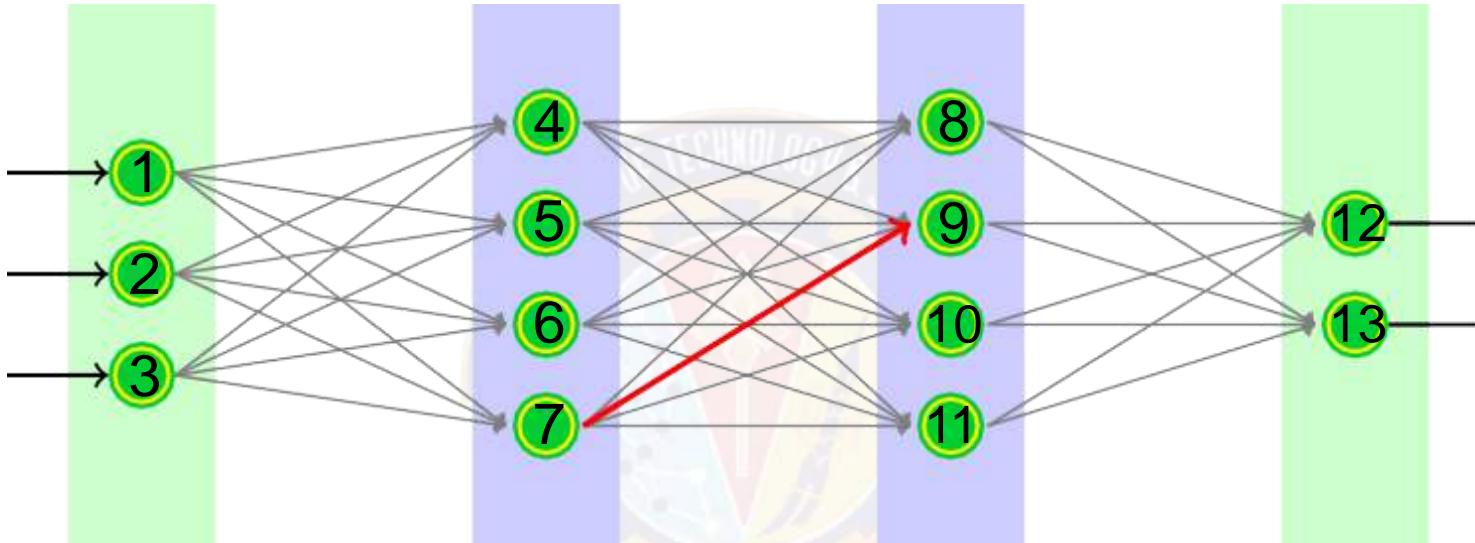
$w_{ji}$  weight associated with  $i$  th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

$t_j$  target output for unit  $j$

# Conventions Over The Network



$x_{ji}$   $i$  th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$  weight associated with  $i$  th input to unit  $j$

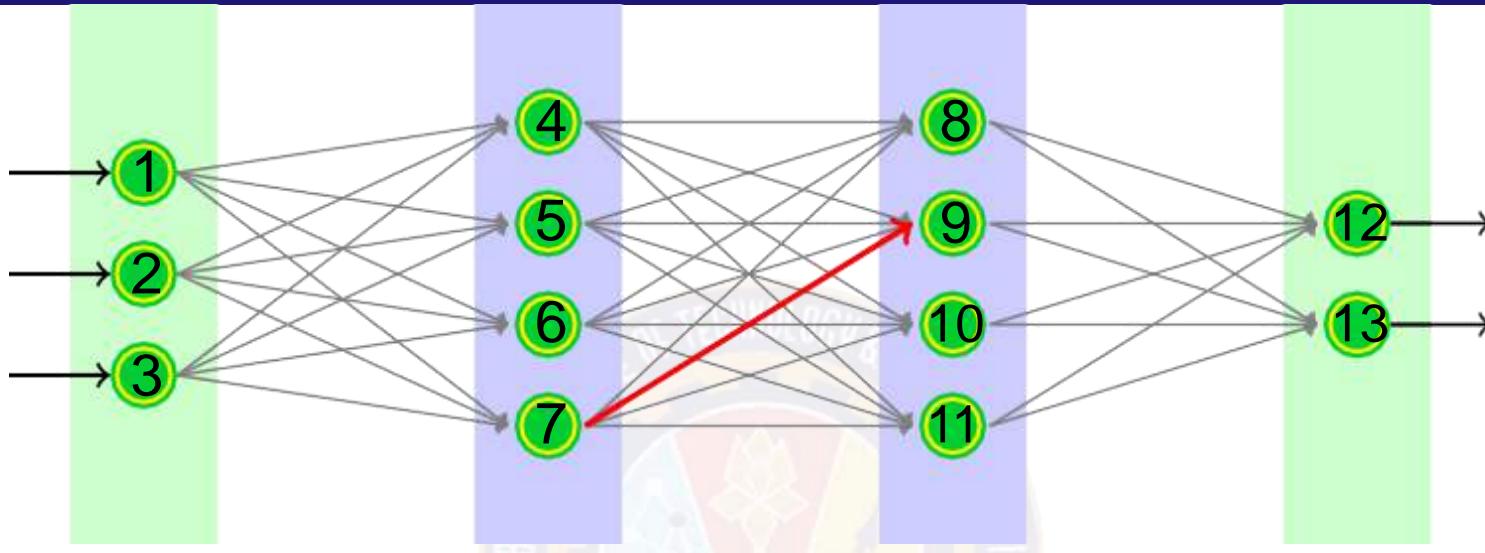
$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

$t_j$  target output for unit  $j$

$outputs$  set of units in final layer ( $\{12, 13\}$  in our case)

# Conventions Over The Network



$x_{ji}$   $i$ th input to unit  $j$  ( $x_{97}$  is highlighted)

$w_{ji}$  weight associated with  $i$ th input to unit  $j$

$net_j$  be  $\sum_i w_{ji}x_{ji}$  the weighted sum of input for unit  $j$

$o_j$  output computed by unit  $j$ . Let it be  $\sigma(net_j)$

$t_j$  target output for unit  $j$

*outputs* set of units in final layer ( $\{12, 13\}$  in our case)

*Downstream( $j$ )* units whose immediate input is the output of unit  $j$

We are interested in  $\frac{\partial E_d}{\partial w_{ji}}$  it is  $\frac{\partial E_d}{\partial net_j} \times \frac{\partial net_j}{\partial w_{ji}}$  and therefore,  $\frac{\partial E_d}{\partial net_j} \times x_{ji}$

# Value of $\frac{\partial E_d}{\partial net_j}$ for output units

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial net_j}$$



# Value of $\frac{\partial E_d}{\partial net_j}$ for output units

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$



# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2$$

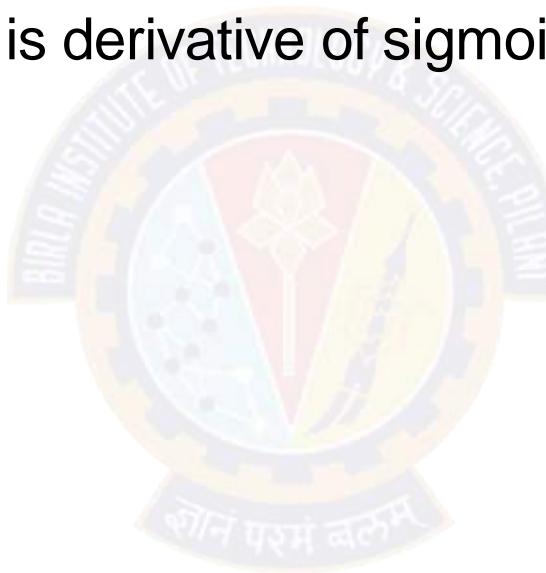


# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid



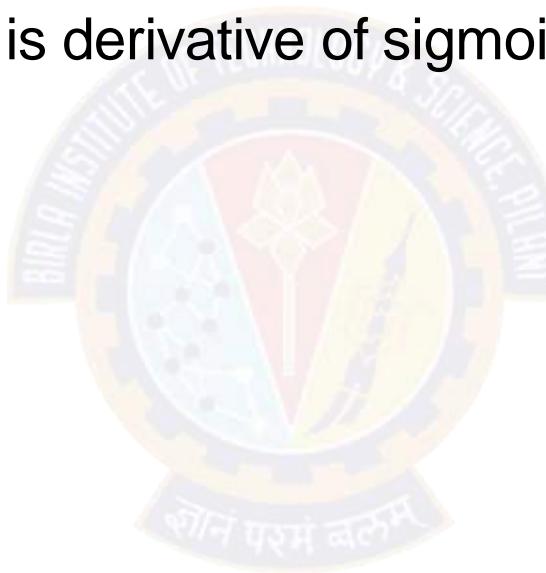
# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\frac{d}{dx} \sigma(x) = \frac{d}{dx} \frac{1}{1 + e^{-x}}$$



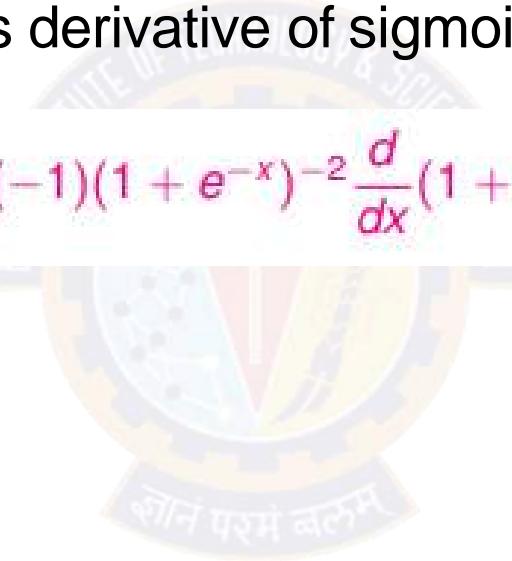
# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\frac{d}{dx} \sigma(x) = \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x})$$



# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x})\end{aligned}$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2}(0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}}\end{aligned}$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j))$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

Therefore,  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \text{net}_j} \times x_{ji}$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for output units

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -(t_j - o_j)$$

Note that  $o_j = \sigma(\text{net}_j)$  therefore  $\frac{\partial o_j}{\partial \text{net}_j}$  is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} = (-1)(1 + e^{-x})^{-2} \frac{d}{dx} (1 + e^{-x}) \\ &= (-1)(1 + e^{-x})^{-2} (0 - e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \times \frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x))\end{aligned}$$

As a result  $\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \sigma(\text{net}_j)}{\partial \text{net}_j} = \sigma(\text{net}_j)(1 - \sigma(\text{net}_j)) = o_j(1 - o_j)$

$$\frac{\partial E_d}{\partial \text{net}_j} = -(t_j - o_j)o_j(1 - o_j)$$

Term  $(t_j - o_j)o_j(1 - o_j)$  is treated as  $\delta_j$

Therefore,  $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial \text{net}_j} \times x_{ji} = \boxed{\eta(t_j - o_j)o_j(1 - o_j)x_{ji}}$

# Value of $\frac{\partial E_d}{\partial \text{net}_j}$ for hidden units

$$\begin{aligned}\frac{\partial E_d}{\partial \text{net}_j} &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E_d}{\partial \text{net}_k} \times \frac{\partial \text{net}_k}{\partial \text{net}_j} \\&= \sum_{k \in \text{Downstream}(j)} -\delta_k \times \frac{\partial \text{net}_k}{\partial \text{net}_j} \\&= \sum_{k \in \text{Downstream}(j)} -\delta_k \times \frac{\partial \text{net}_k}{\partial o_j} \times \frac{\partial o_j}{\partial \text{net}_j} \\&= \sum_{k \in \text{Downstream}(j)} -\delta_k \times w_{kj} \times o_j(1 - o_j)\end{aligned}$$

$\delta_j$  being  $-\frac{\partial E_d}{\partial \text{net}_j} = o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \delta_k \times w_{kj}$

Therefore,  $\Delta w_{ji} = \eta \delta_j x_{ji} = \boxed{\eta(o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \delta_k \times w_{kj}) x_{ji}}$

# Backpropagation (for 2 layers)

---

**Algorithm:** Backpropagation( $D, \eta, n_{in}, n_{out}, n_{hidden}$ )

---

- 1 Create the feedforward network with  $n_{in}$ ,  $n_{out}$ ,  $n_{hidden}$  layers
  - 2 Randomly initialize weights to small values  $\in [-0.05, +0.05]$
  - 3 **repeat**
  - 4     **for each**  $\langle \vec{x}, \vec{t} \rangle \in D$  **do**
  - 5          $o_u = \text{get output from network } \forall \text{unit } u$
  - 6          $\delta_k = o_k(1 - o_k)(t_k - o_k)$  for all **output unit  $k$**
  - 7          $\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} (w_{kh} \delta_k)$  for all **hidden unit  $h$**
  - 8          $w_{ji} = w_{ji} + \Delta w_{ji}$      where  $\Delta w_{ji} = \eta \delta_j x_{ji}$
  - 9 **until** converge;
-

# Backpropagation

- **Adding Momentum:** weight update during  $n^{th}$  iteration depend partially on the update that occurred during the  $(n - 1)^{th}$  iteration

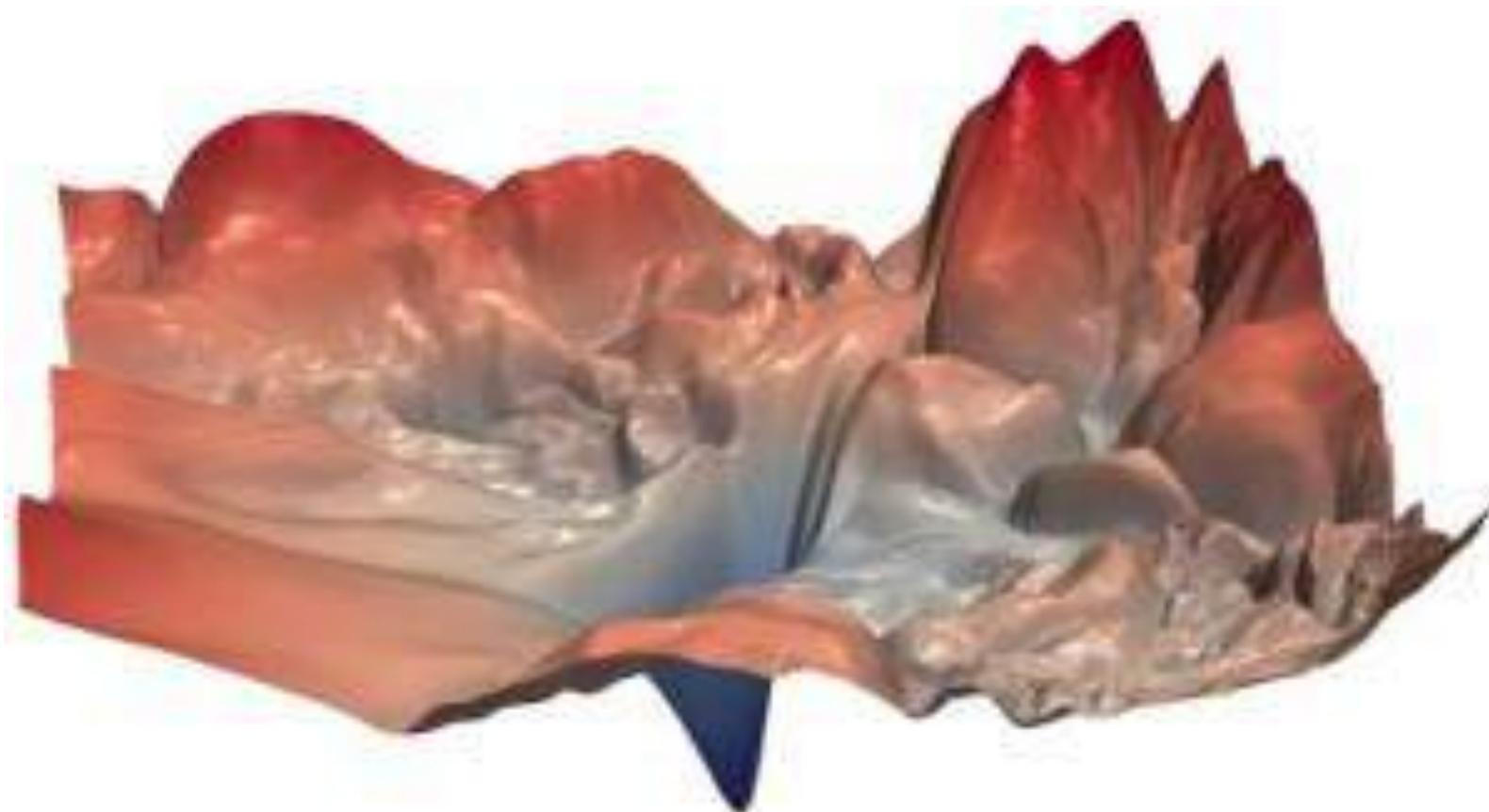
$$\Delta w_{jl}(n) = \eta \delta_j x_{jl} + \alpha \Delta w_{jl}(n-1)$$

- **Learning in arbitrary acyclic network:** for feed-forward networks of arbitrary depth,  $\delta_r$  value for a unit in hidden layer is determined as

$$\delta_r = o_r(1 - o_r) \sum_{s \in \text{Downstream}(r)} w_{sr} \times \delta_s$$

# Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error



# Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error
- No methods are known to predict with certainty when local minima will cause difficulties
- Suggested to use momentum, true gradient descent or multiple networks (initialized with different random weights)
- Any boolean function can precisely be represented by some network having only **two** layers of units (Cybenko 1989; Hornik et al. 1989)



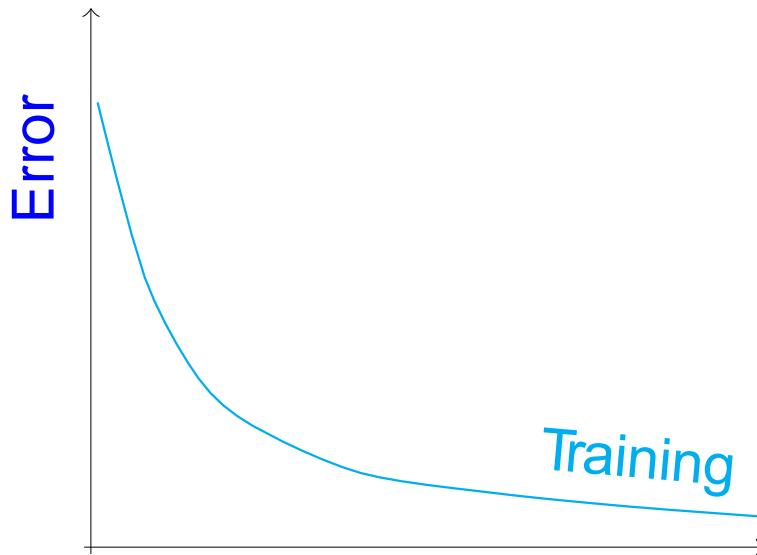
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Generalization, Overfitting and Stopping Criteria

Kamlesh Tiwari

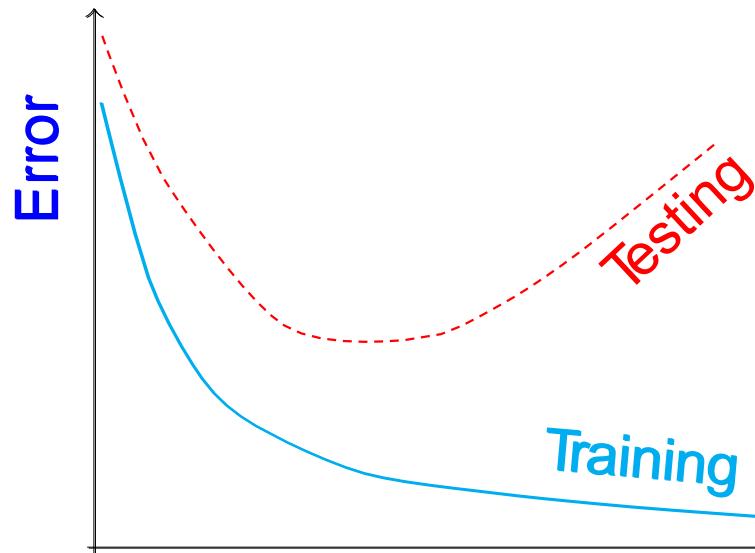
# Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



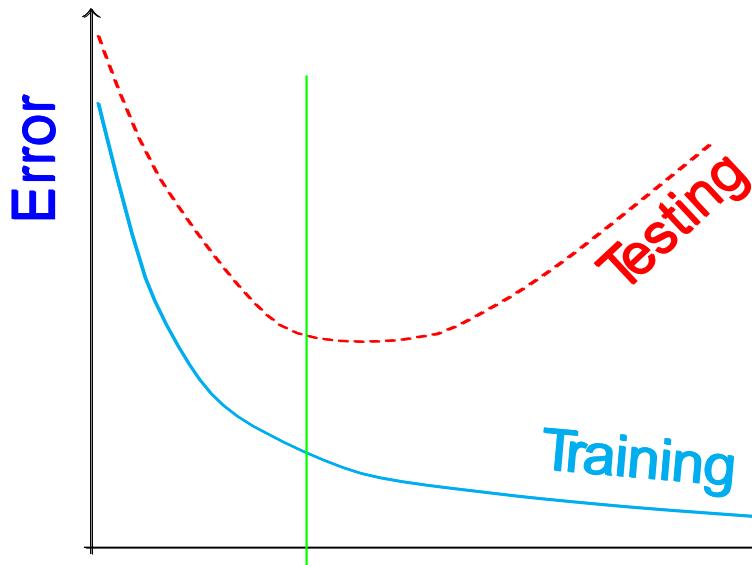
# Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



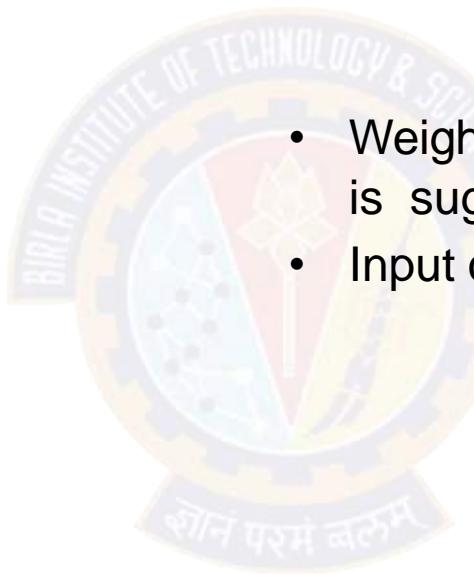
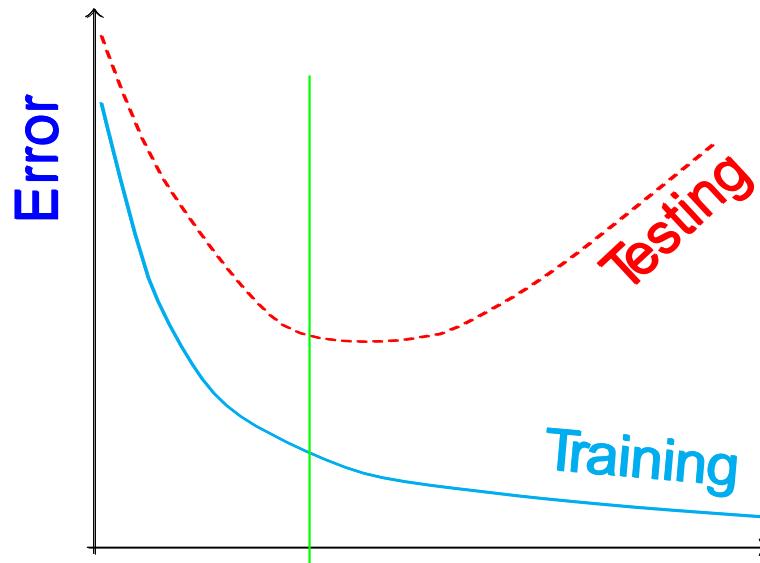
# Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



# Generalization, Overfitting, and Stopping Criterion

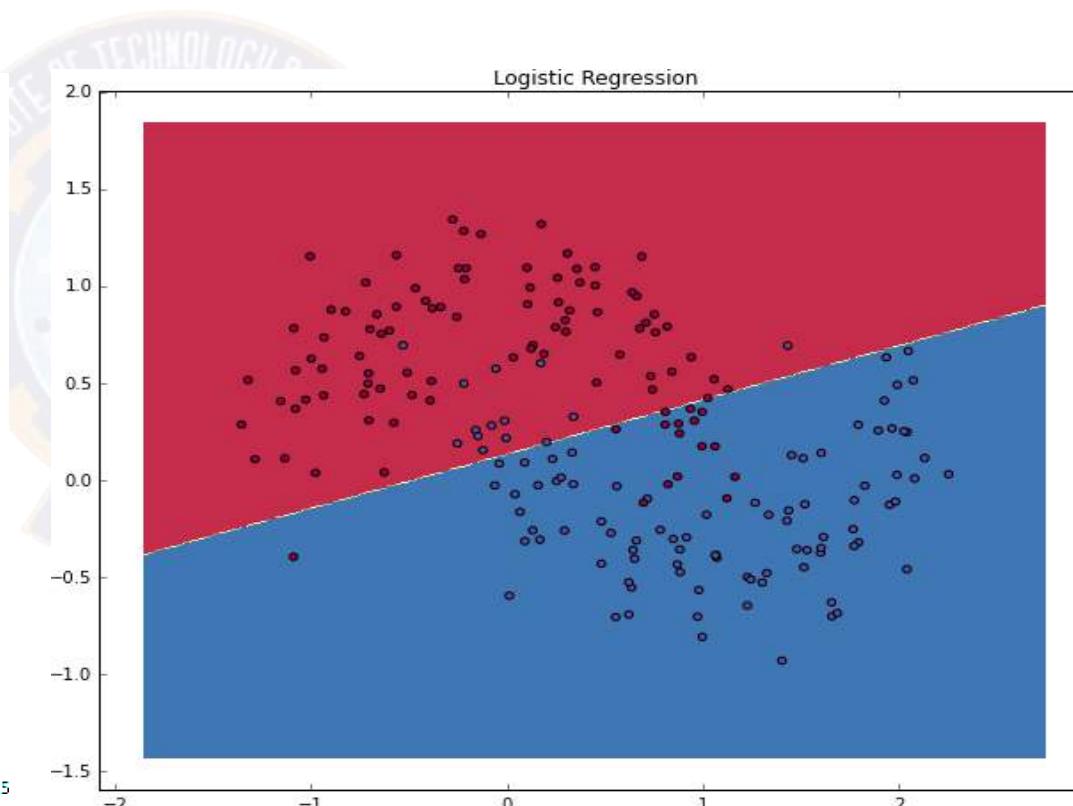
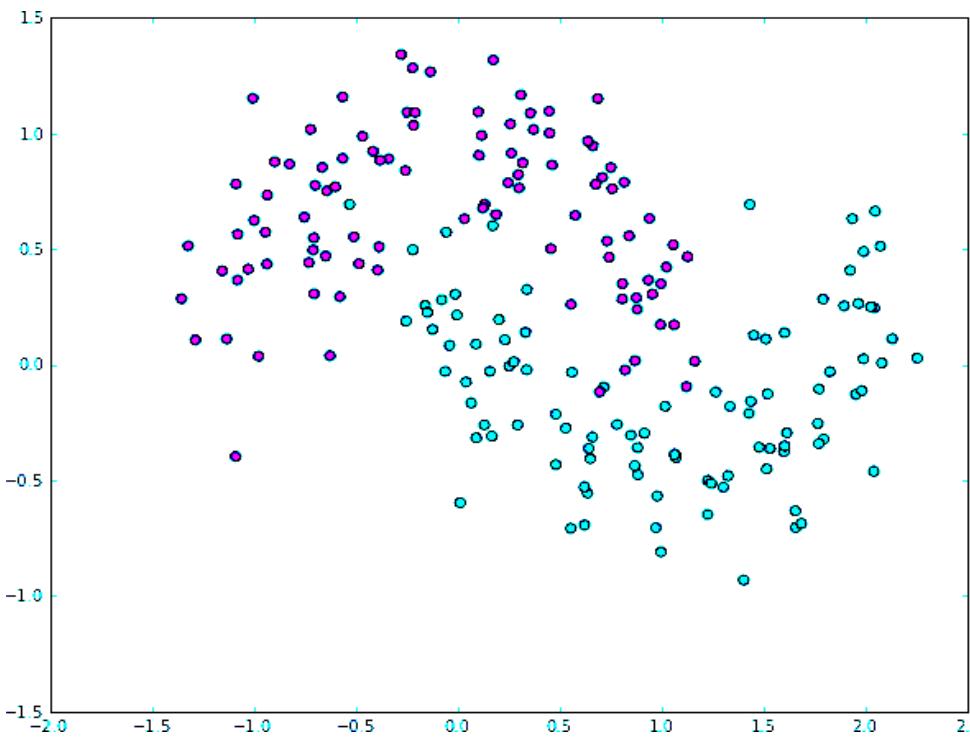
Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



- Weight decay or use of validation set ( $k$ -fold ?) is suggested
- Input or output encoding can be used

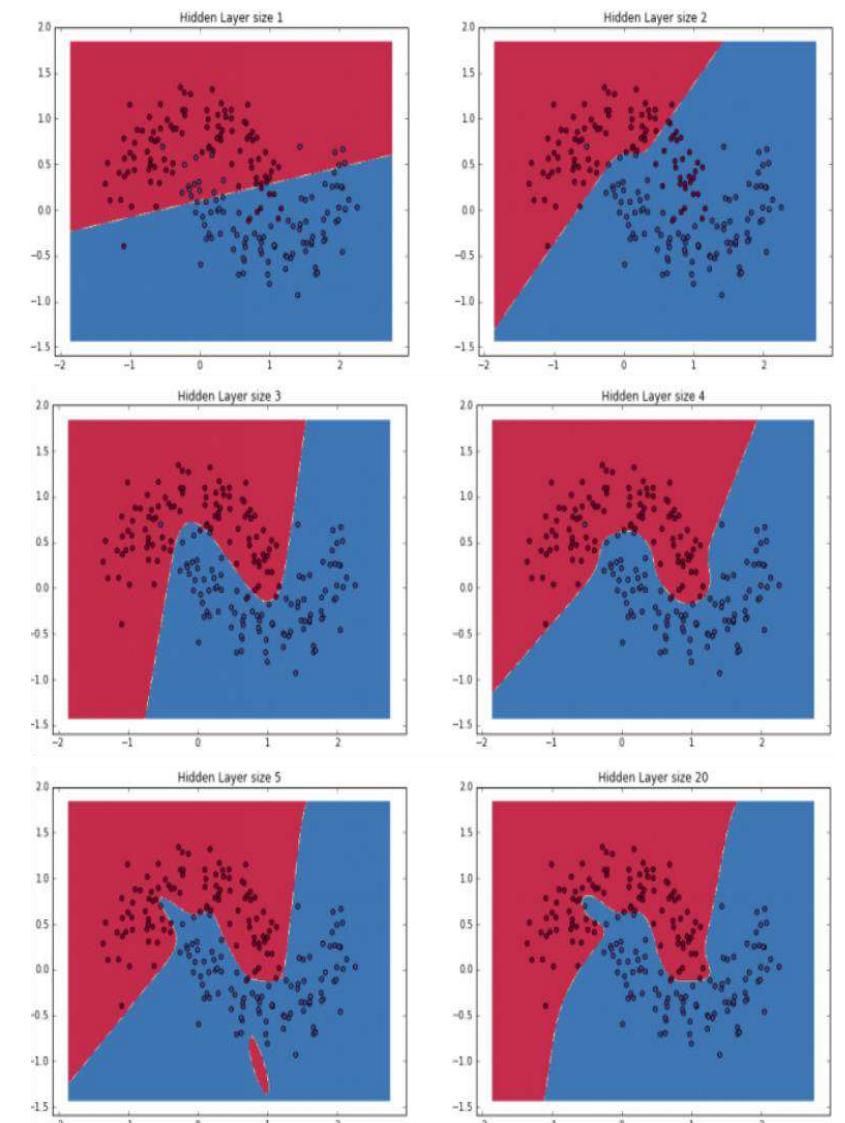
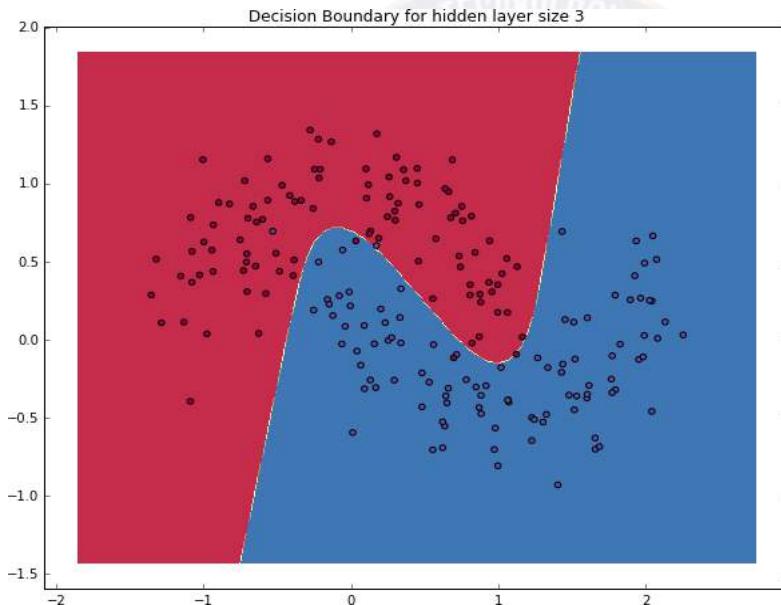
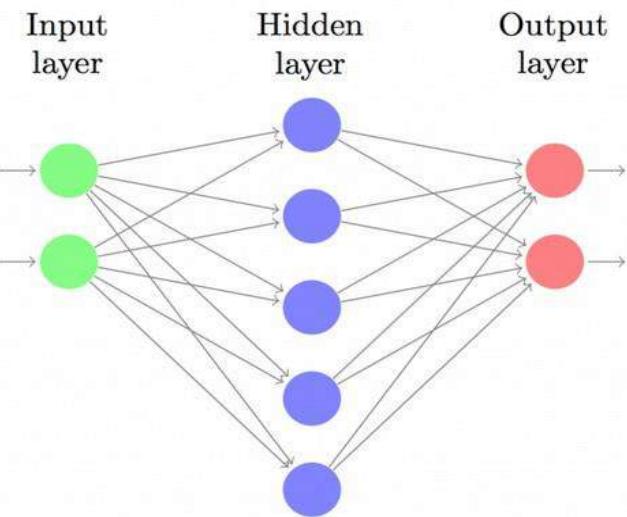
# Coding Example 1

Consider two class data  
Logistic Regression  
Neural Network  
Decision Boundary  
Bigger hidden layer



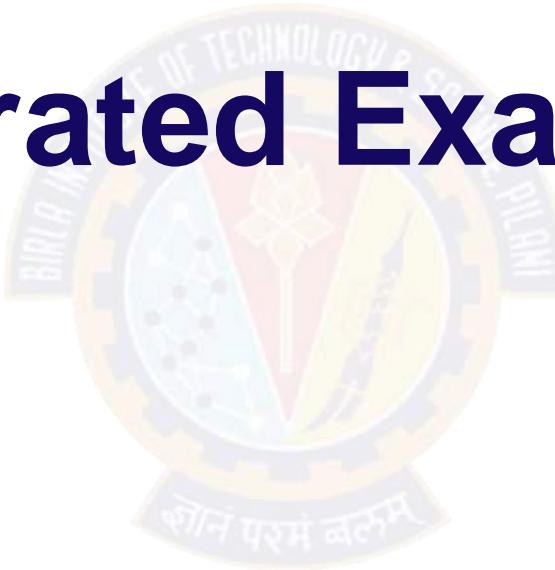
<sup>1</sup><https://github.com/dennybritz/nan-from-scratch/blob/master/nan-from-scratch.ipynb>

# Coding Example 1



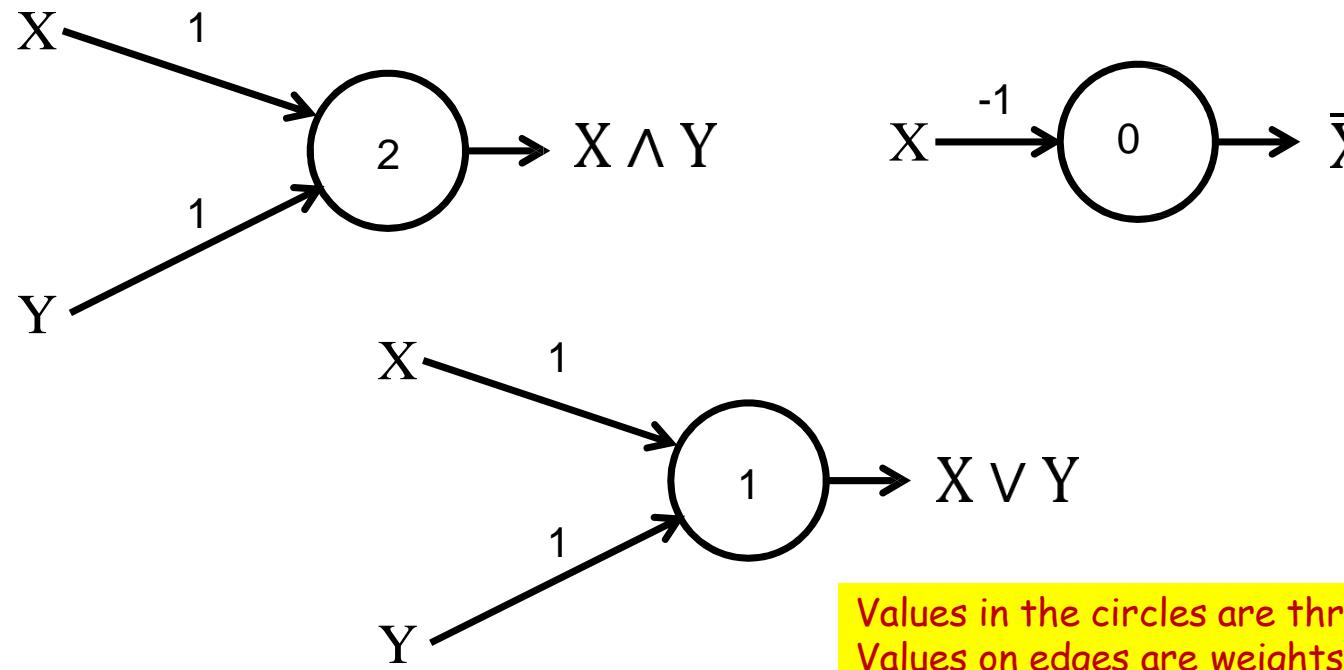
<sup>1</sup><https://github.com/dennybritz/nn-from-scratch/blob/master/nn-from-scratch.ipynb>

# Illustrated Examples



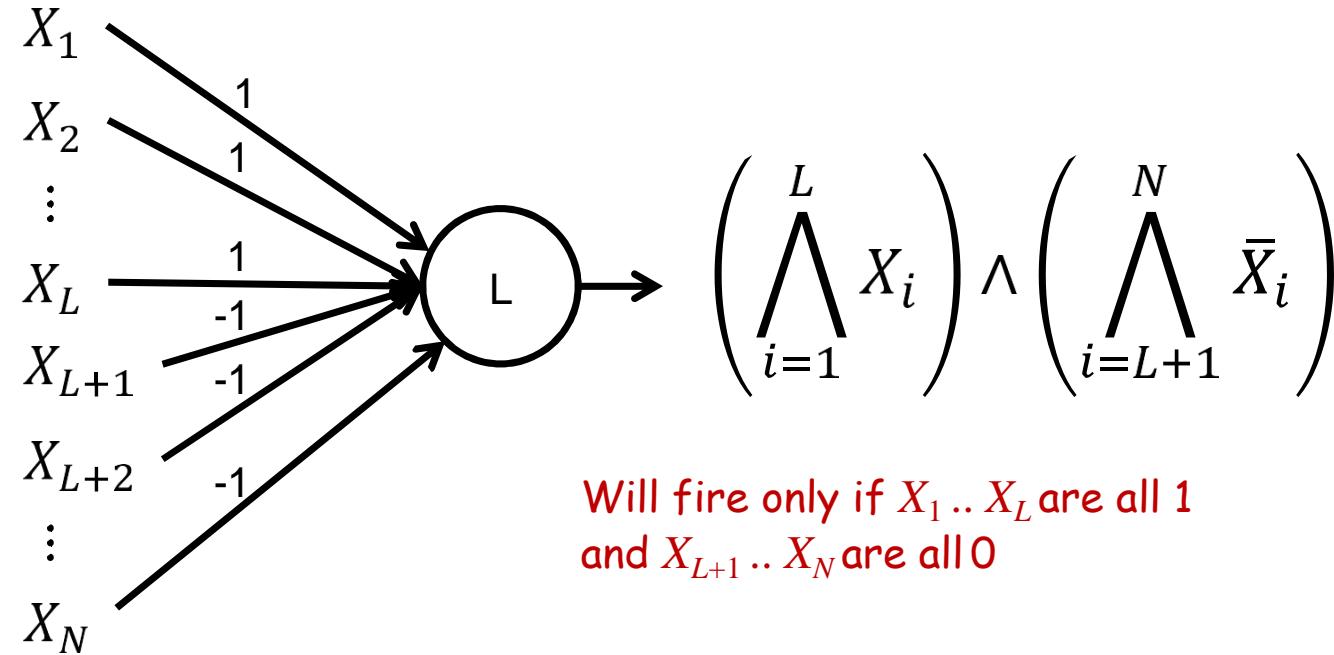
- Multi-layer Perceptrons as universal Boolean functions
- MLPs as universal classifiers

# The perceptron as a Boolean gate



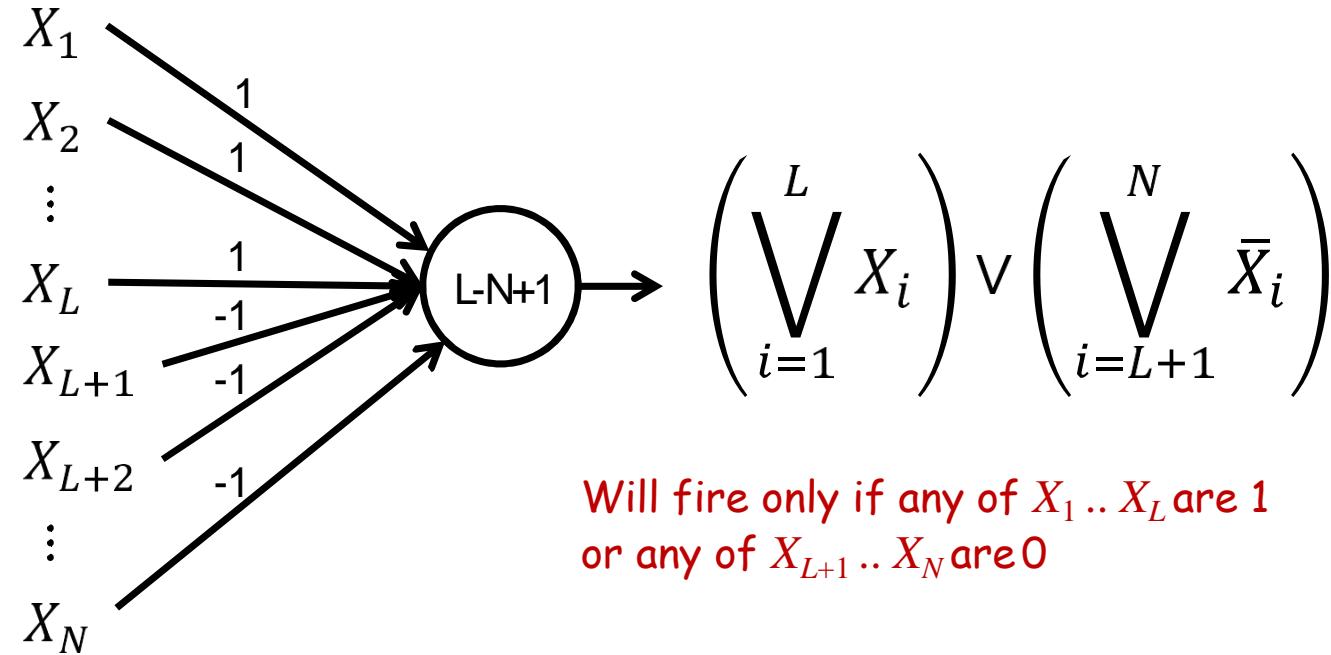
- A perceptron can model any simple binary Boolean gate
- Output = 1 if total input  $\geq$  threshold

# Perceptron as a Boolean gate



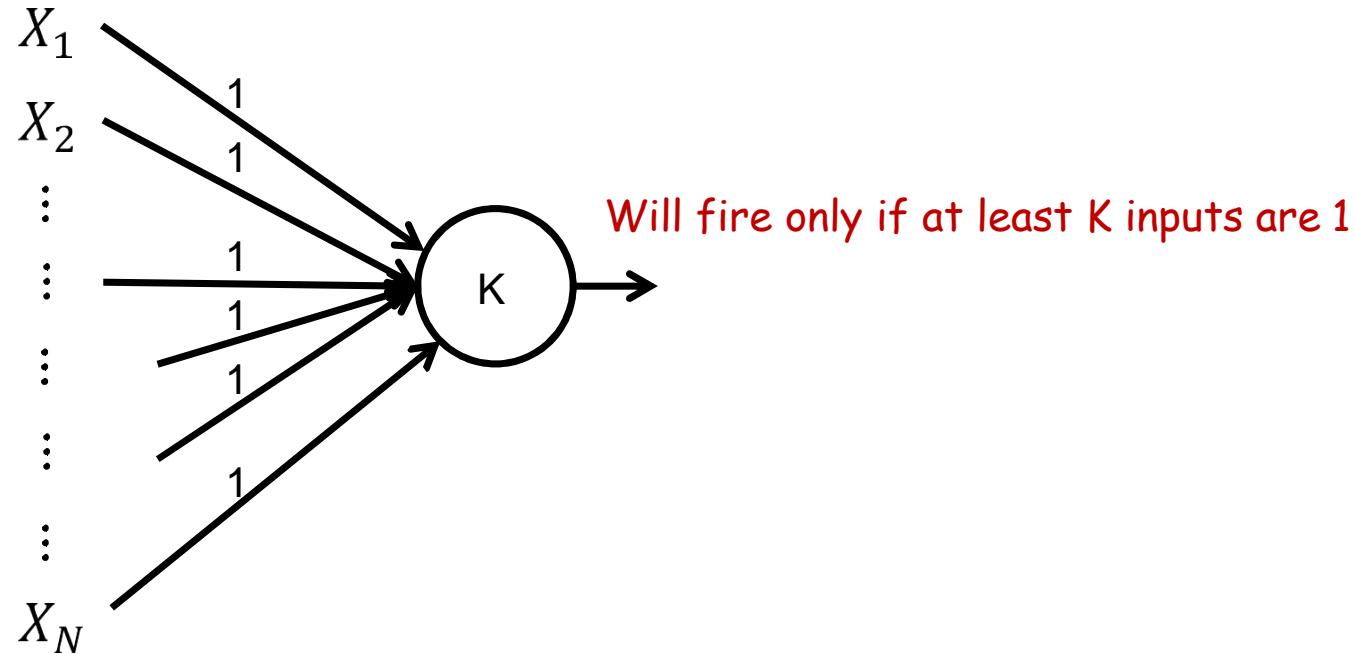
- The universal AND gate
  - AND any number of inputs
    - Any subset of who may be negated

# Perceptron as a Boolean gate



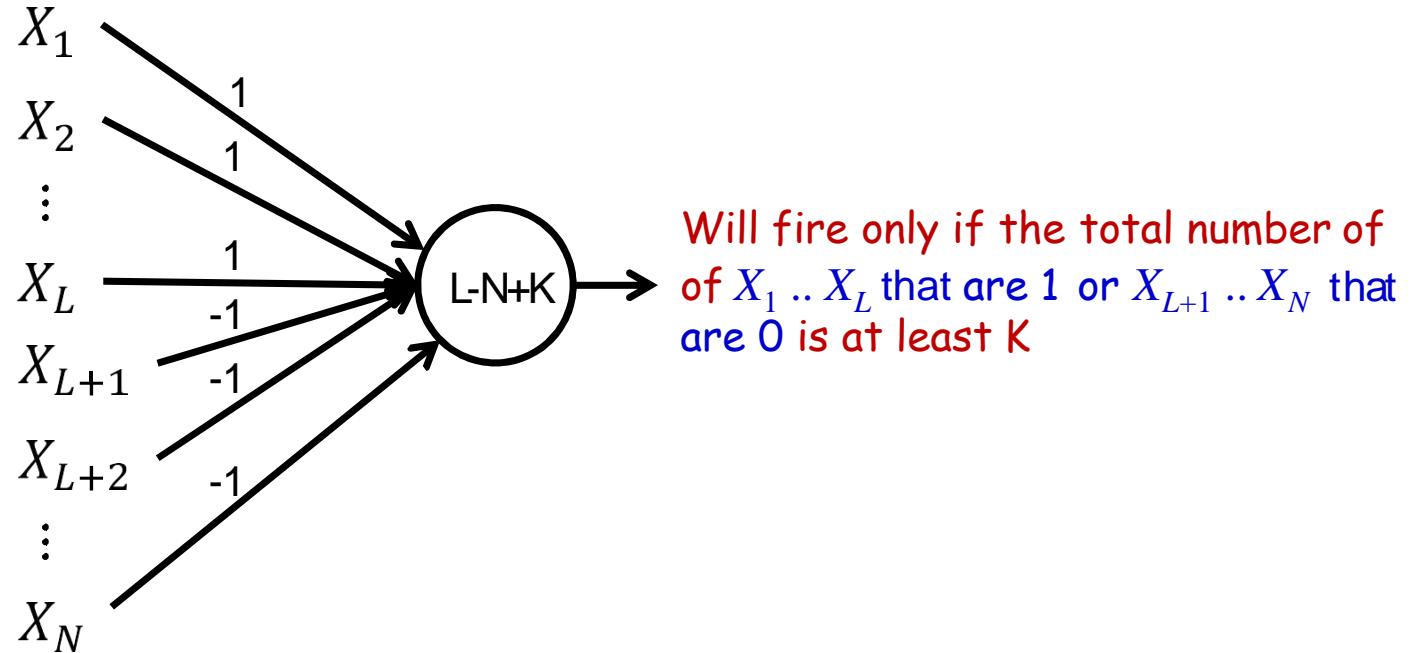
- The universal OR gate
  - OR any number of inputs
    - Any subset of who may be negated

# Perceptron as a Boolean Gate



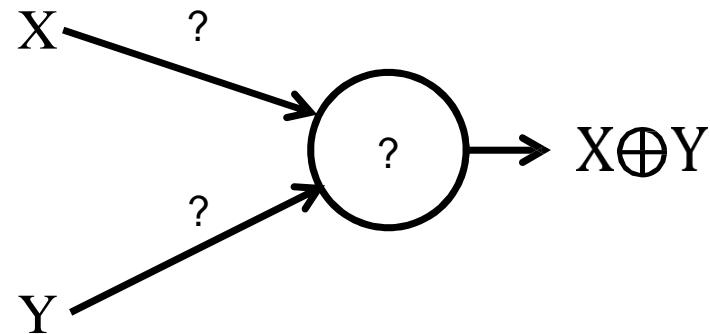
- Generalized *majority* gate
  - Fire if at least  $K$  inputs are of the desired polarity

# Perceptron as a Boolean Gate



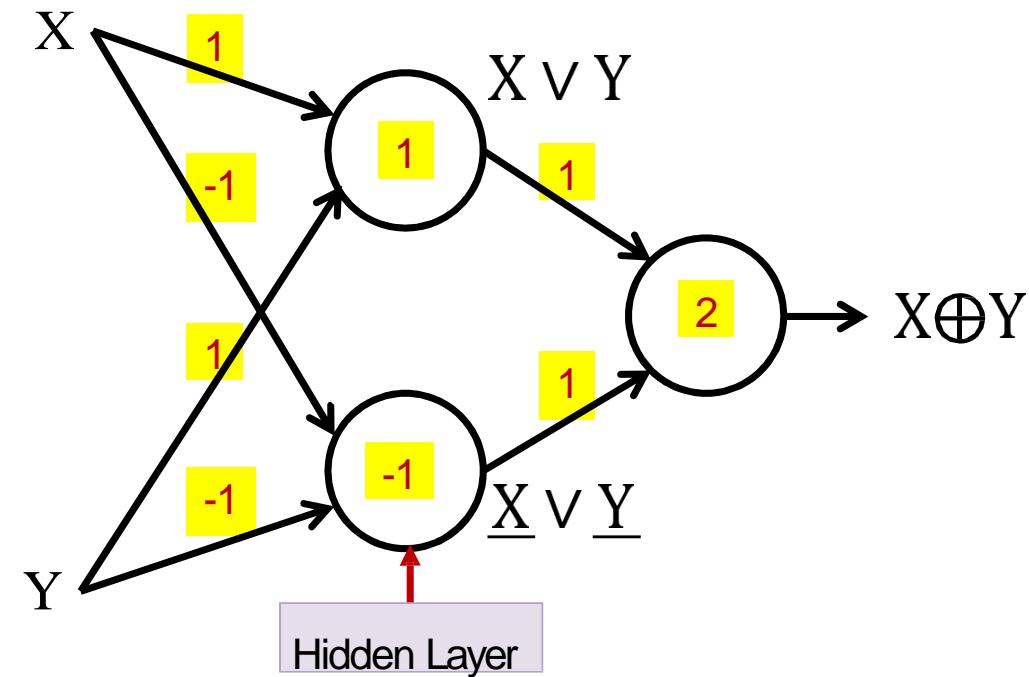
- Generalized *majority* gate
  - Fire if at least  $K$  inputs are of the desired polarity

# The perceptron is not enough



- Cannot compute an XOR

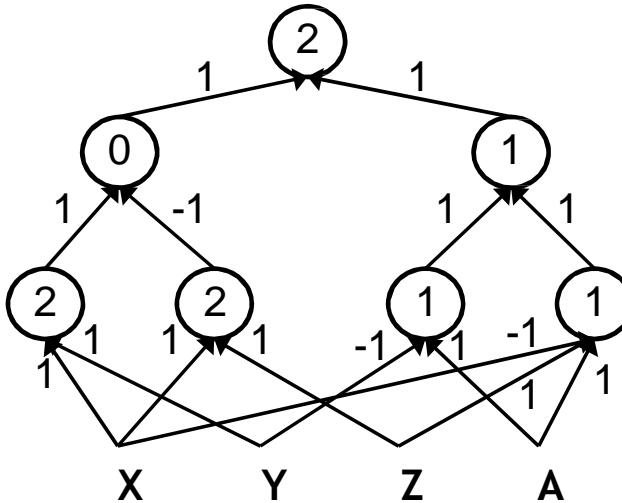
# Multi-layer perceptron



- MLPs can compute the XOR

# Multi-layer perceptron

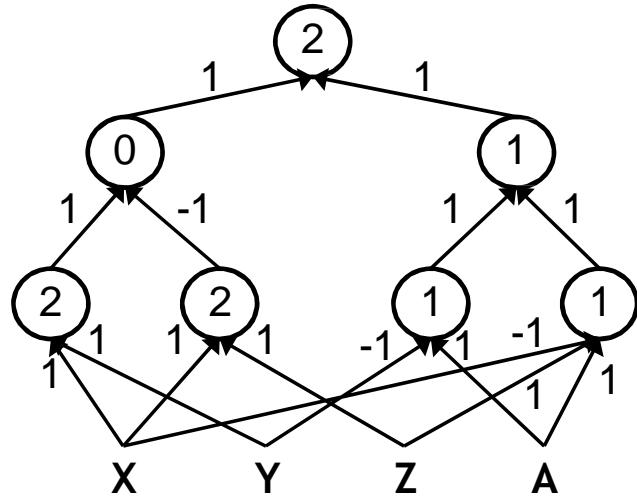
$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | (\bar{X} \& \bar{Z}))$$



- MLPs can compute more complex Boolean functions
- MLPs can compute *any* Boolean function
  - Since they can emulate individual gates
- **MLPs are *universal Boolean functions***

# MLP as Boolean Functions

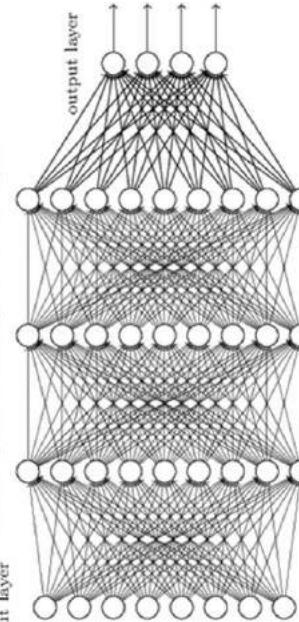
$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | \overline{(X \& Z)})$$



```

graph TD
    A[Input layer] --> B[hidden layer 1]
    B --> C[hidden layer 2]
    C --> D[hidden layer 3]

```



- MLPs are universal Boolean functions
    - Any function over any number of inputs and any number of outputs
  - But how many “layers” will they need?

# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations  
for which output is 1

- *A Boolean function is just a truth table*

# How many layers for a Boolean MLP?

Truth Table

X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>	X <sub>4</sub>	X <sub>5</sub>	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations  
for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + \\ X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$

- Expressed in disjunctive normal form (DNF)

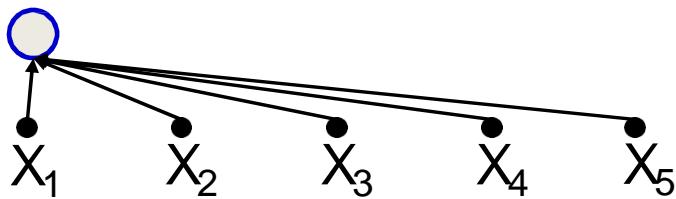
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 X_2 X_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

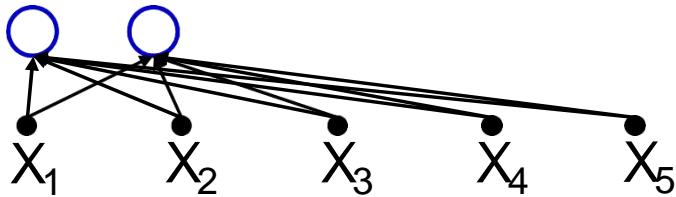
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \textcircled{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 X_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

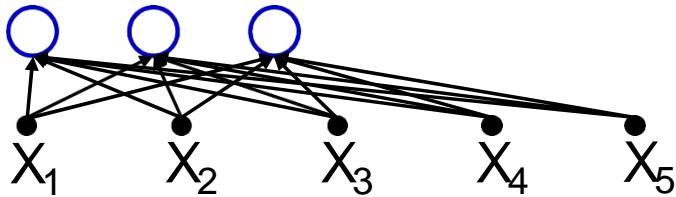
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations  
for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \textcircled{\bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5} + \\ X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

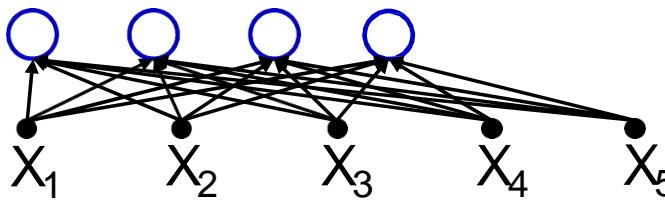
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations  
for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 +$$
  
$$\textcircled{X}_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 \blacksquare X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

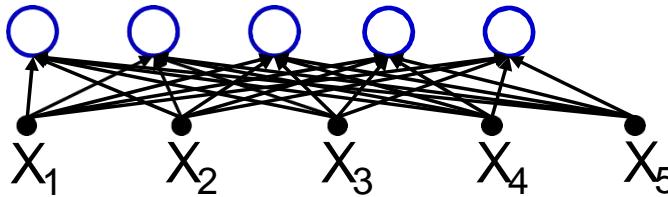
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations  
for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + \\ X_1 \bar{X}_2 X_3 \bar{X}_4 X_5 + \textcircled{X}_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

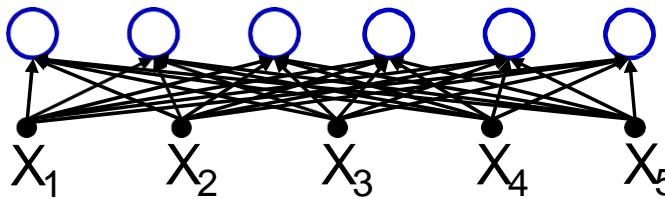
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + \\ X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + \textcircled{X}_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

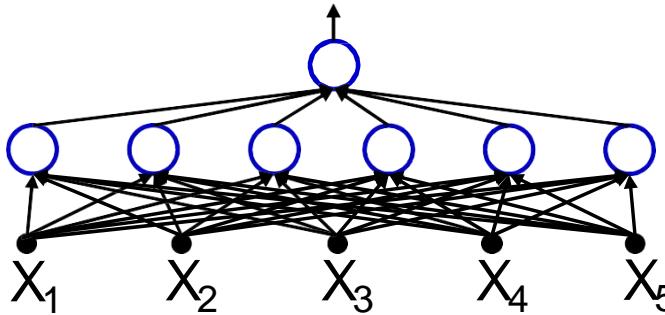
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

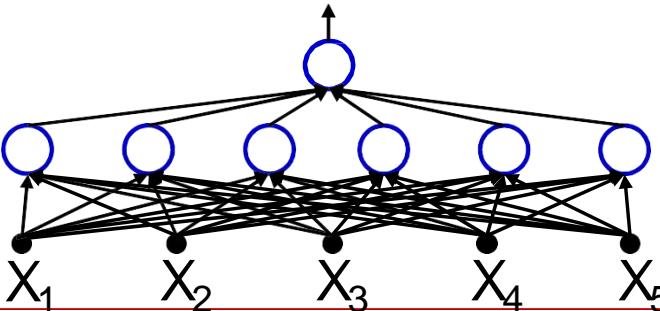
# How many layers for a Boolean MLP?

Truth Table

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y$
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Any truth table can be expressed in this manner!
- A one-hidden-layer MLP is a Universal Boolean Function

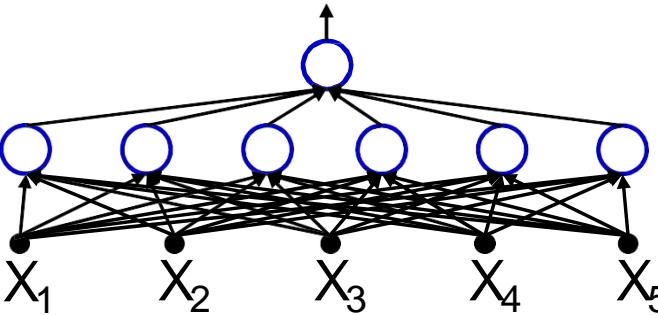
# How many layers for a Boolean MLP?

Truth Table

X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>	X <sub>4</sub>	X <sub>5</sub>	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

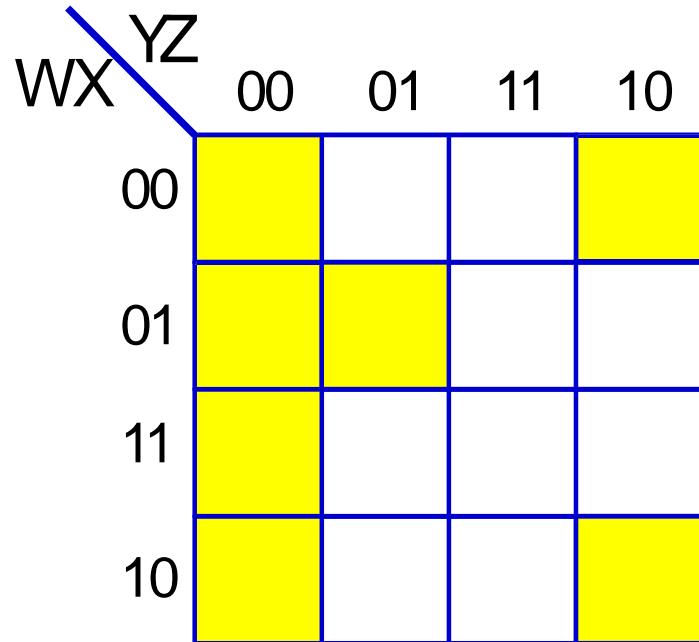
$$Y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Any truth table can be expressed in this manner!
- A one-hidden-layer MLP is a Universal Boolean Function

But what is the largest number of perceptrons required in the single hidden layer for an N-input-variable function?

# Reducing a Boolean Function



This is a "Karnaugh Map"

It represents a truth table as a grid  
Filled boxes represent input combinations  
for which output is 1; blank boxes have  
output 0

Adjacent boxes can be "grouped" to  
reduce the complexity of the DNF formula  
for the table

- DNF form:
  - Find groups
  - Express as reduced DNF

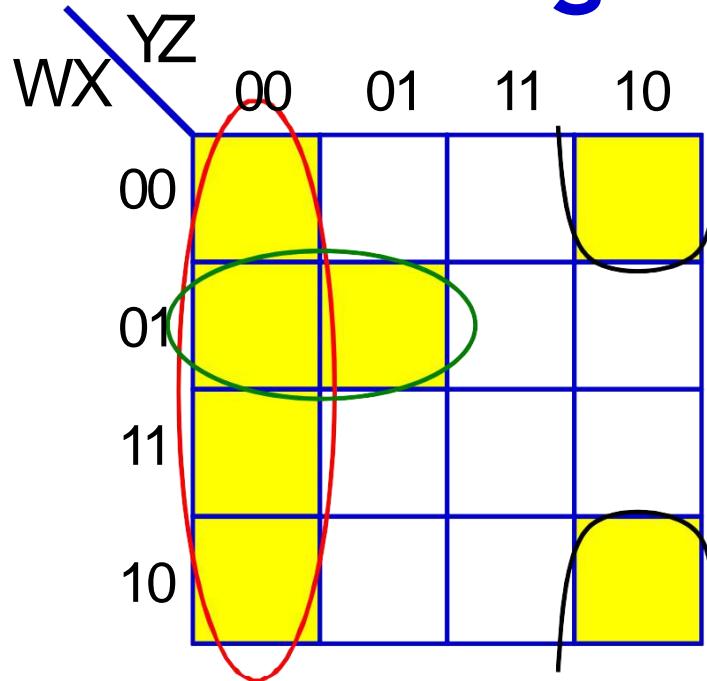
# Reducing a Boolean Function

	WX	YZ	00	01	11	10
00						
01						
11						
10						

A Karnaugh map for a Boolean function with variables  $WX$  and  $YZ$ . The rows are labeled 00, 01, 11, 10 and the columns are labeled 00, 01, 11, 10. Yellow cells indicate where the function value is 1. There are yellow cells at (00,00), (00,10), (01,00), (01,10), (11,00), and (10,00).

Basic DNF formula will require 7 terms

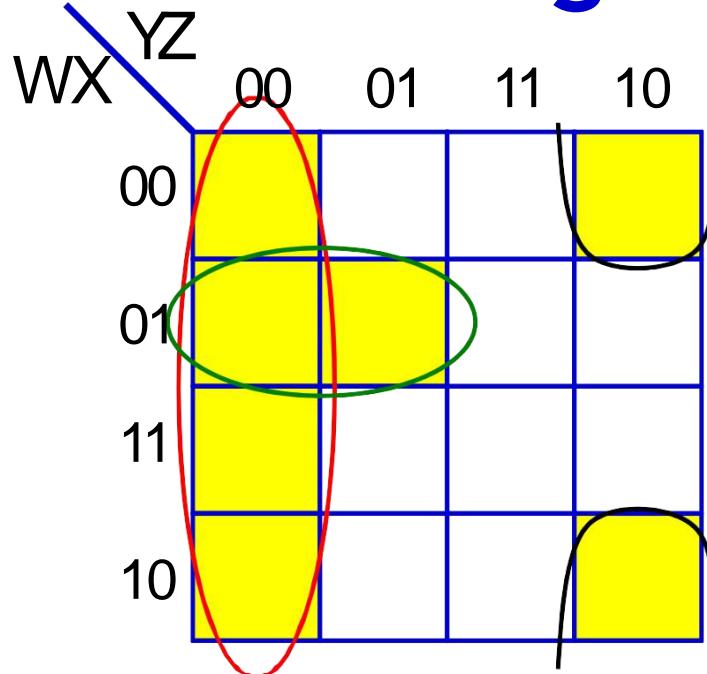
# Reducing a Boolean Function



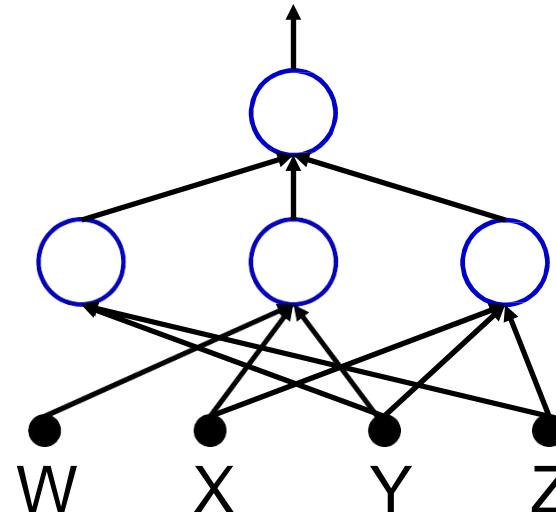
$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$

- *Reduced DNF form:*
  - Find groups
  - Express as reduced DNF

# Reducing a Boolean Function



$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$



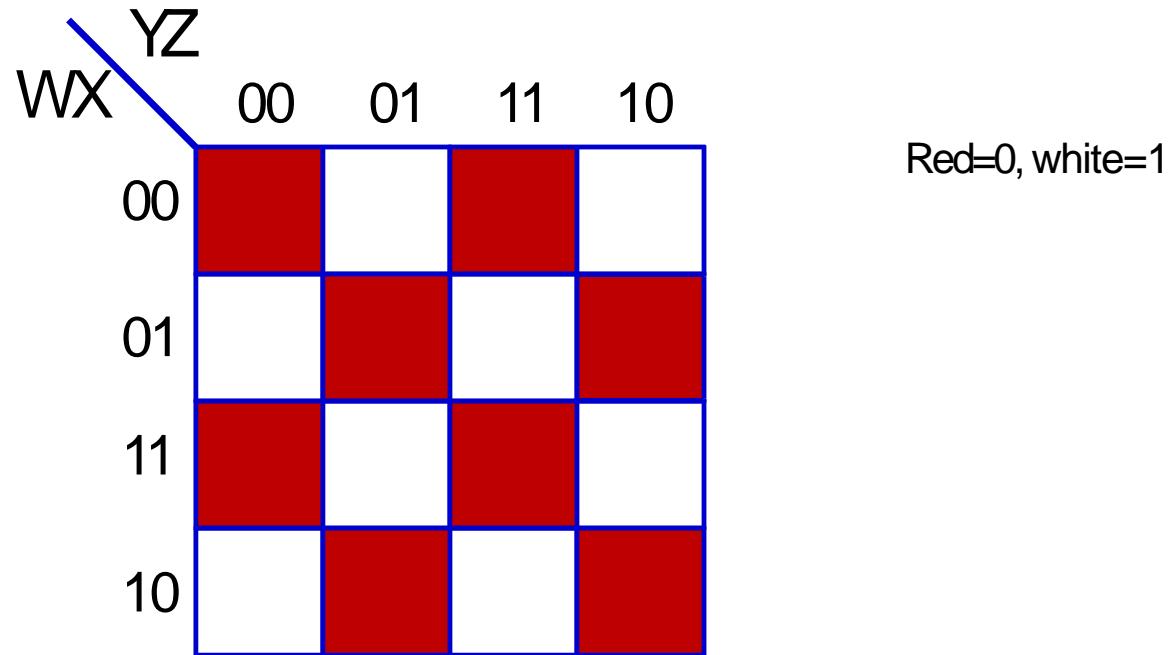
- *Reduced DNF form:*
  - Find groups
  - Express as *reduced* DNF
  - Boolean network for this function needs only 3 hidden units
    - Reduction of the DNF reduces the size of the one-hidden-layer network

# Largest irreducible DNF?

	YZ	00	01	11	10
WX	00				
	01				
	11				
	10				

- What arrangement of ones and zeros simply cannot be reduced further?

# Largest irreducible DNF?



- What arrangement of ones and zeros simply cannot be reduced further?

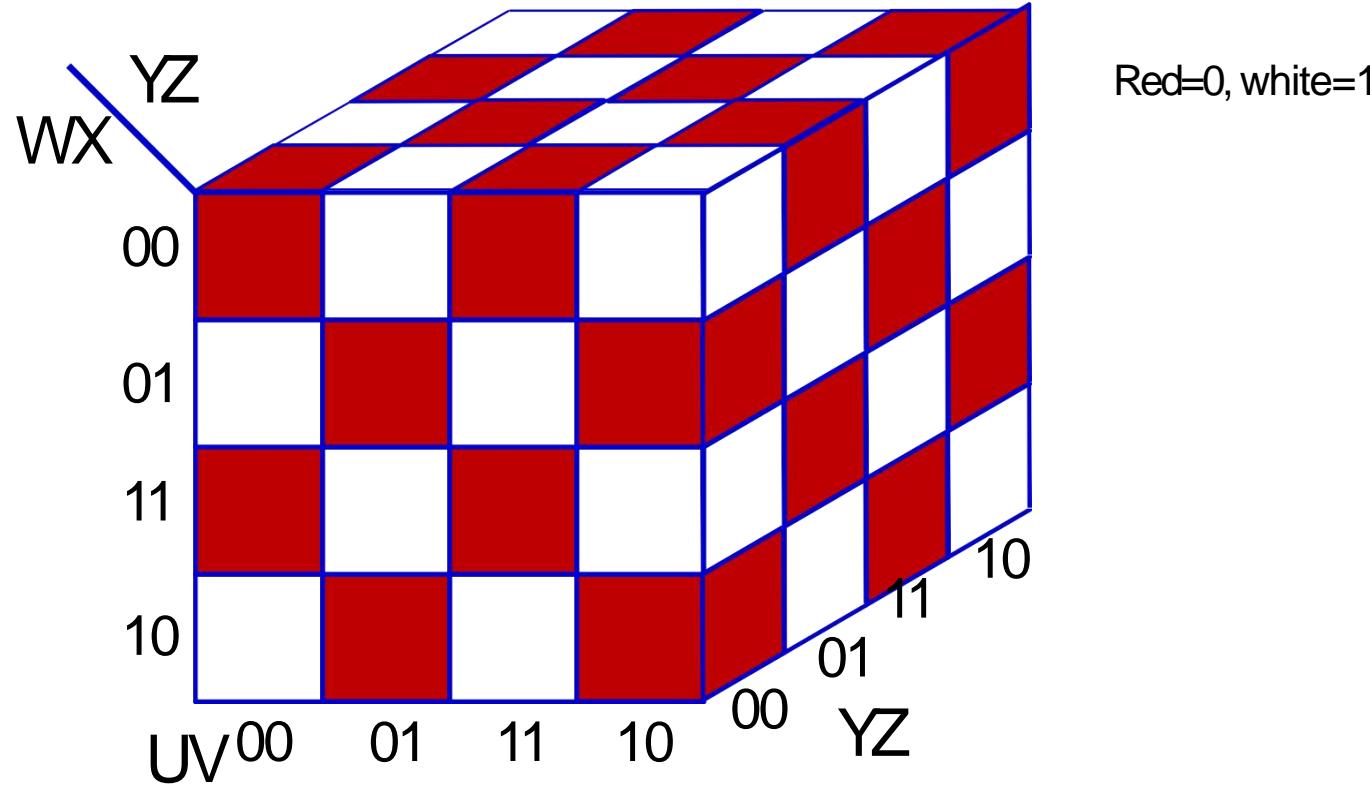
# Largest irreducible DNF?

WX	YZ	00	01	11	10
00		■		■	
01			■		■
11		■		■	
10			■		■

How many neurons  
in a DNF (one-  
hidden-layer) MLP  
for this Boolean  
function?

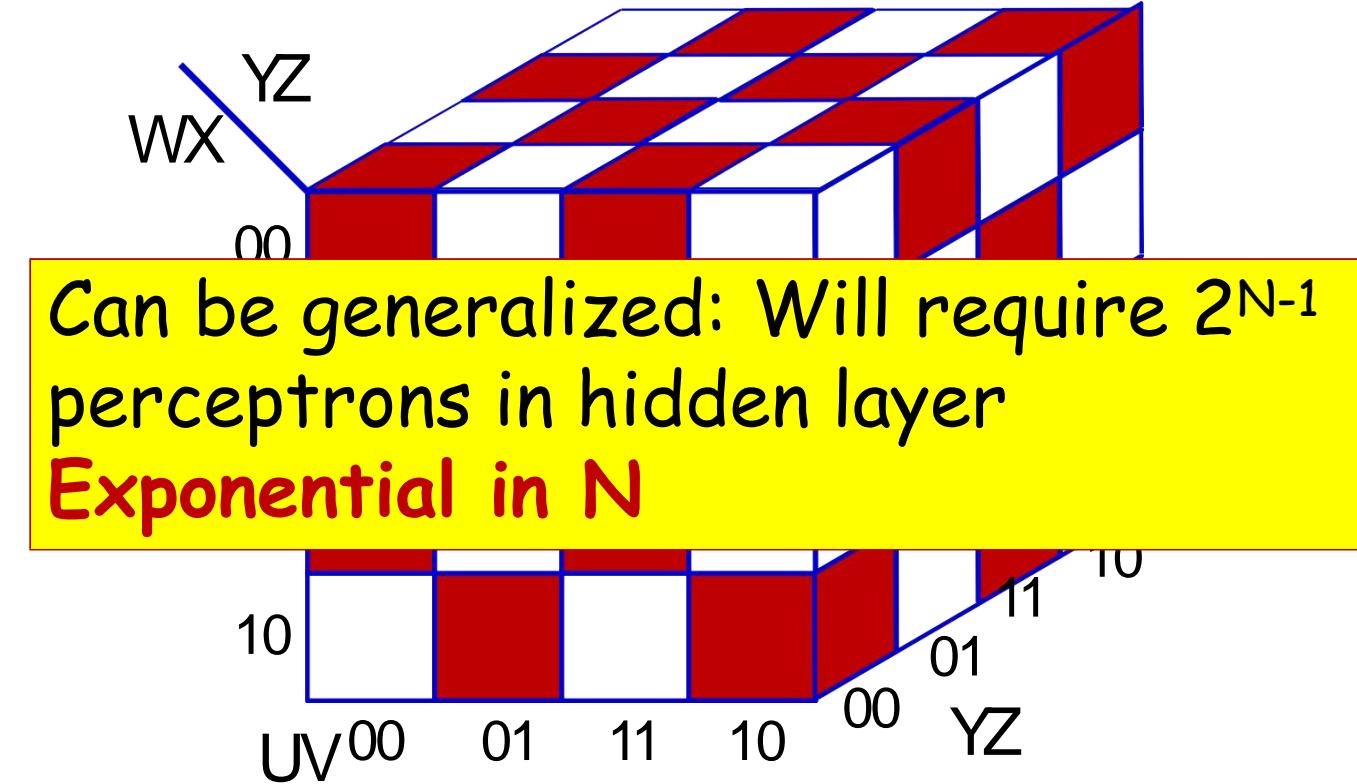
- What arrangement of ones and zeros simply cannot be reduced further?

# Width of a single-layer Boolean MLP



- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function of 6 variables?

# Width of a single-layer Boolean MLP



- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function

# Width of a single-layer Boolean MLP

WX  
YZ

00

Can be generalized: Will require  
 $2^{N-1}$  perceptrons in hidden layer  
**Exponential in N**

10

UV  
00

01

11

10

00

Y  
Z

10

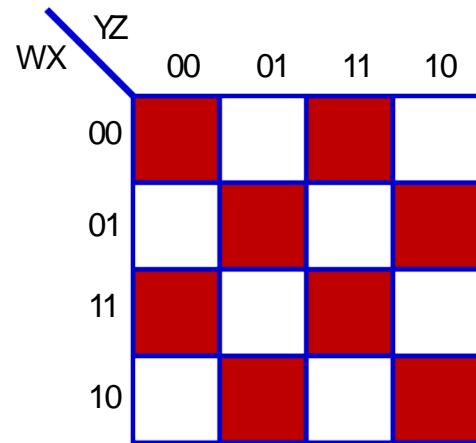
11

01

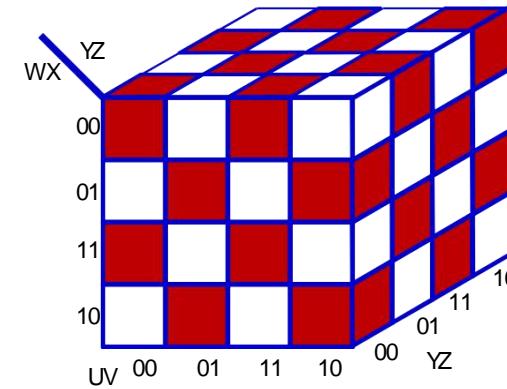
How many units if we use *multiple layers*?

layer) MLP for this Boolean function

# Width of a deepMLP

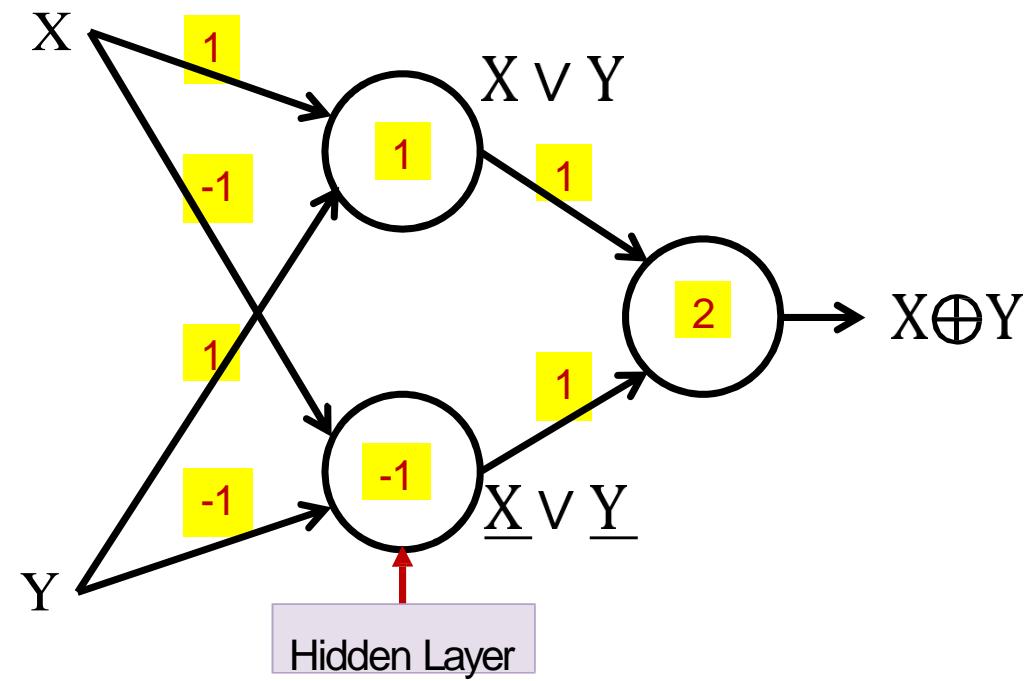


$$O = W \oplus X \oplus Y \oplus Z$$



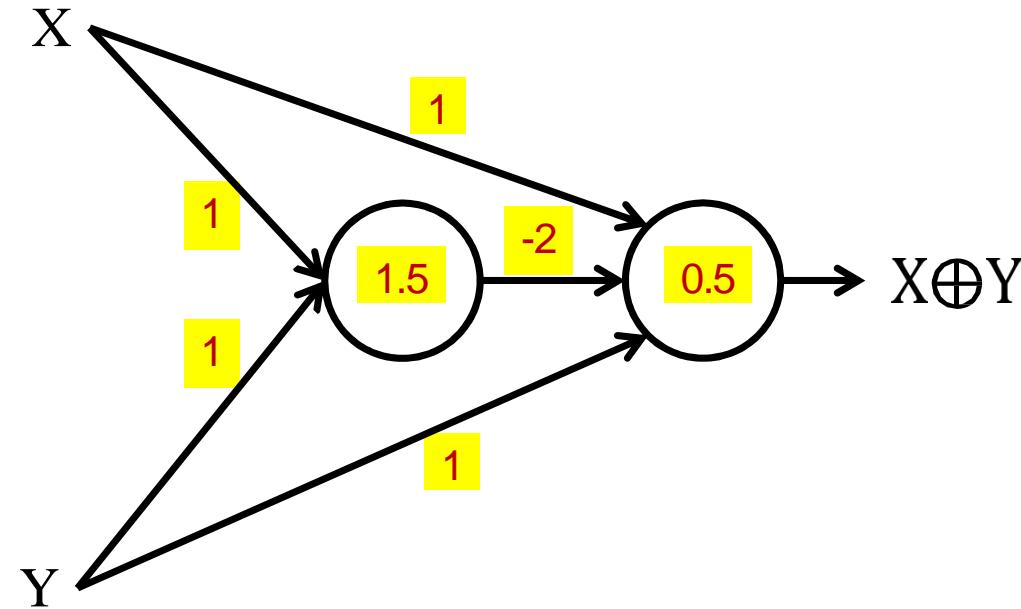
$$O = U \oplus V \oplus W \oplus X \oplus Y \oplus Z$$

# Multi-layer perceptron XOR



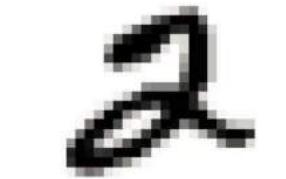
- An XOR takes three perceptrons

# Multi-layer perceptron XOR

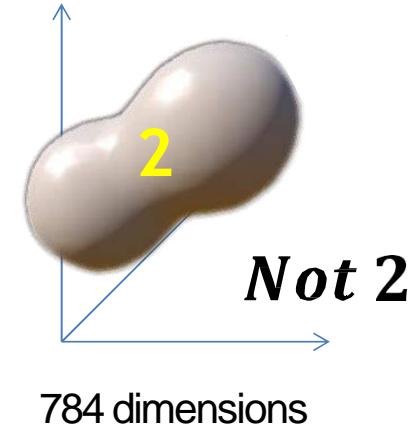
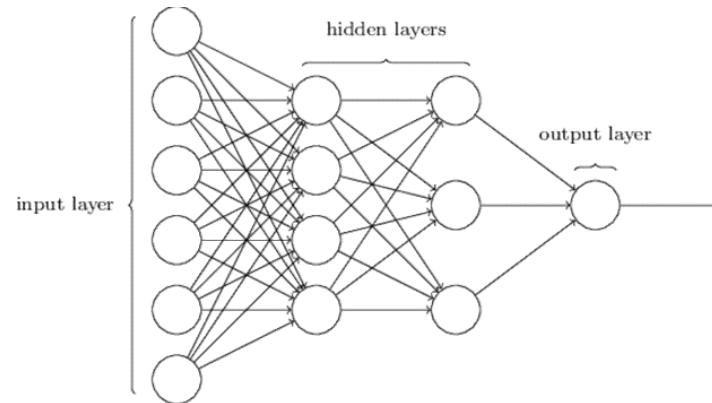


- With 2 neurons
  - 5 weights and two thresholds

# The MLP as a classifier

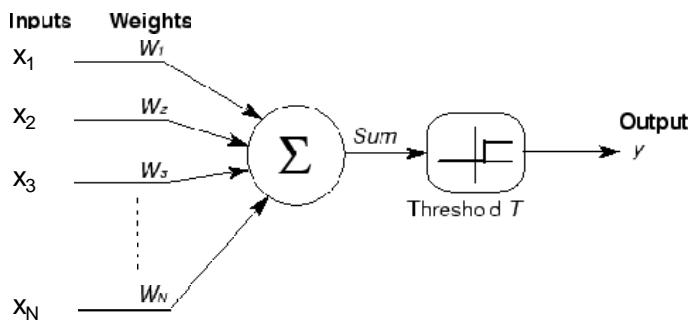


784 dimensions  
(MNIST)



- MLP as a function over real inputs
- MLP as a function that finds a complex “decision boundary” over a space of *reals*

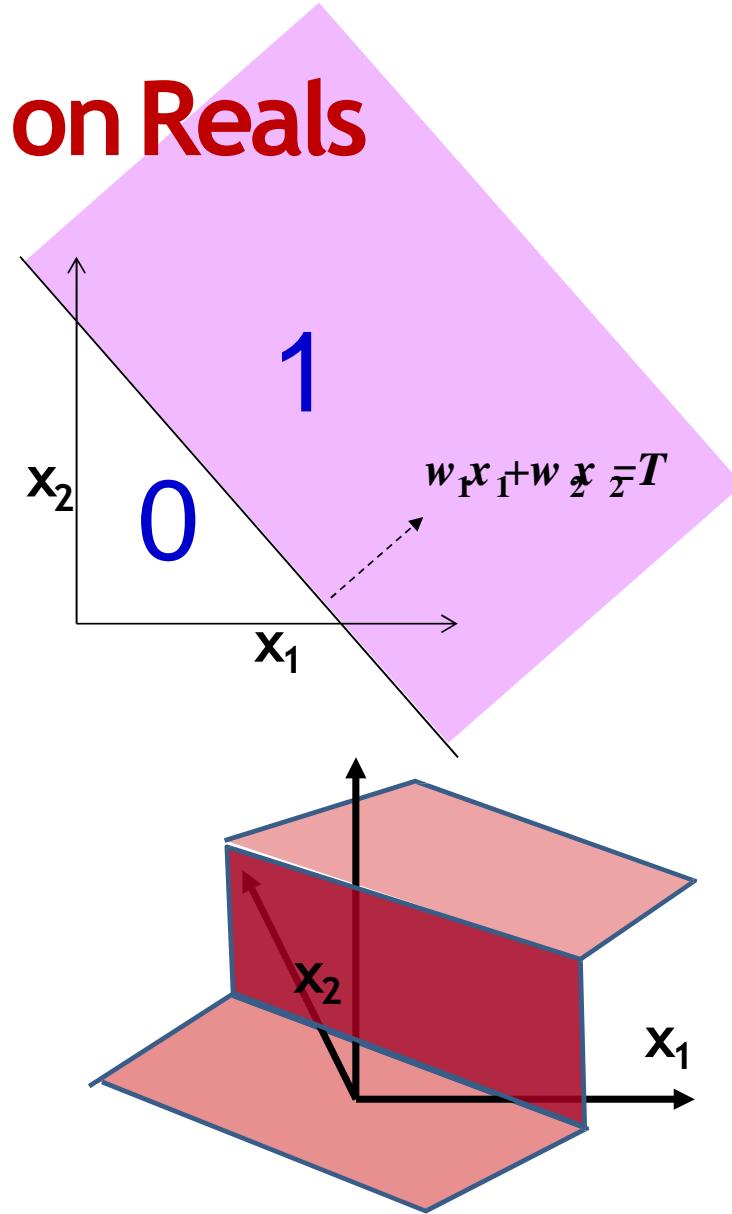
# A Perceptron on Reals



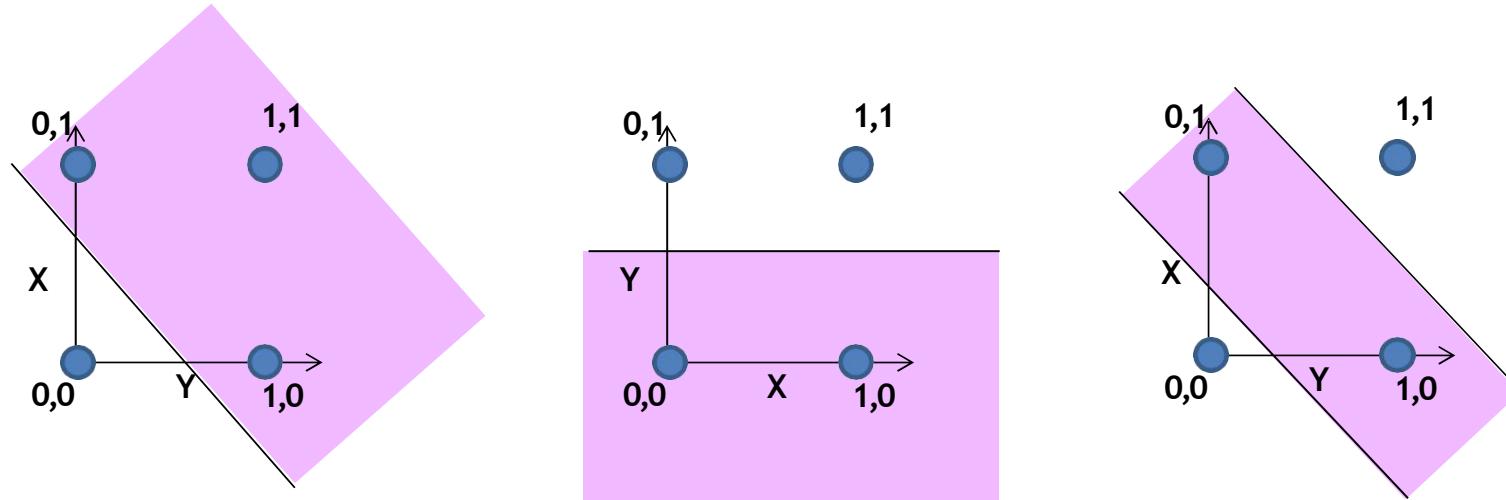
$$y = \begin{cases} 1 & \text{if } \sum_i w_i x_i \geq T \\ 0 & \text{else} \end{cases}$$

- A perceptron operates on *real-valued* vectors

– This is a *linear classifier*

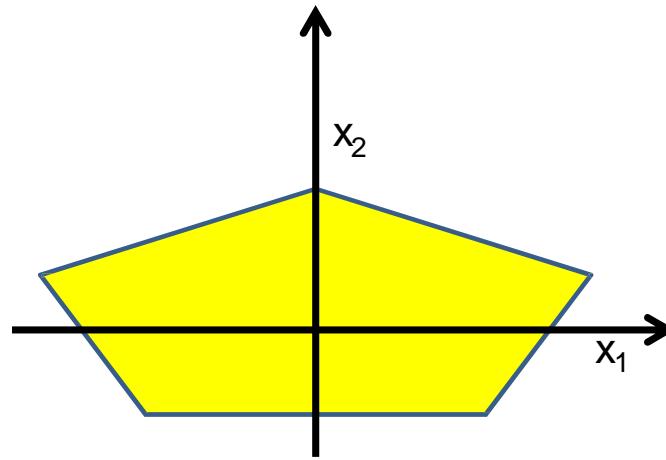


# Boolean functions with a real perceptron



- Boolean perceptrons are also linear classifiers
  - Purple regions are 1

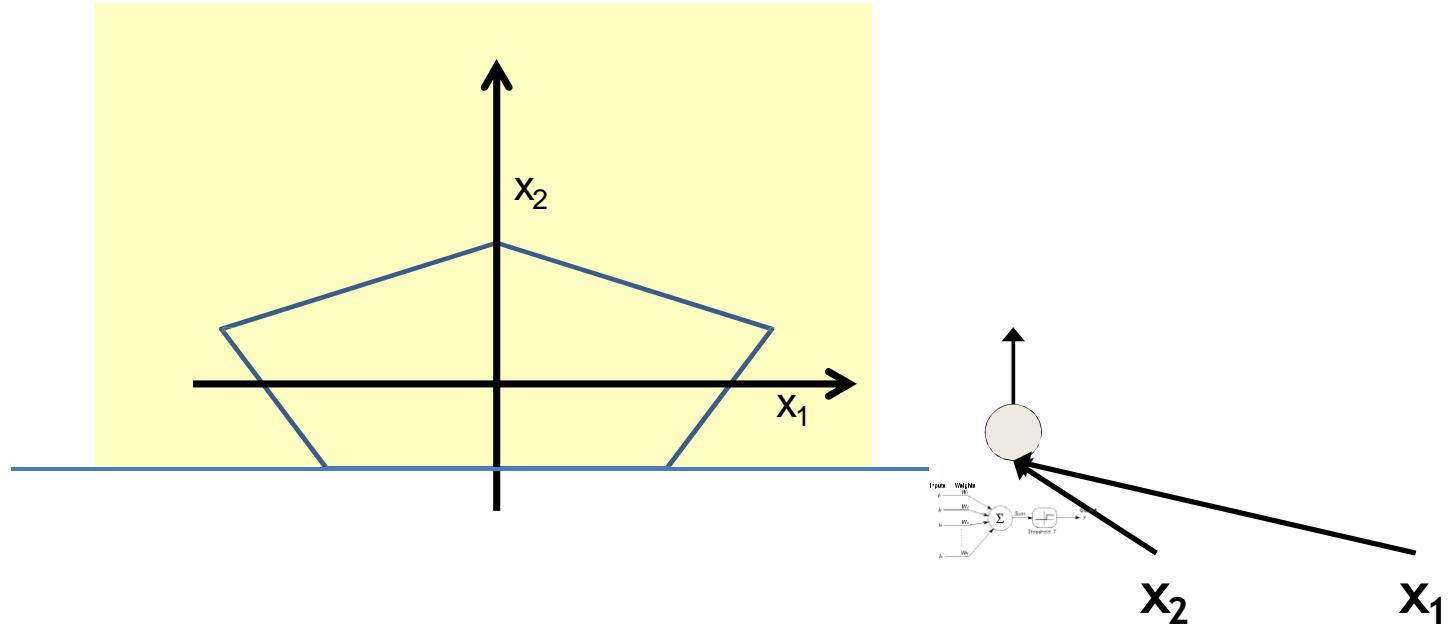
# Composing complicated “decision” boundaries



Can now be composed into  
“networks” to compute arbitrary  
classification “boundaries”

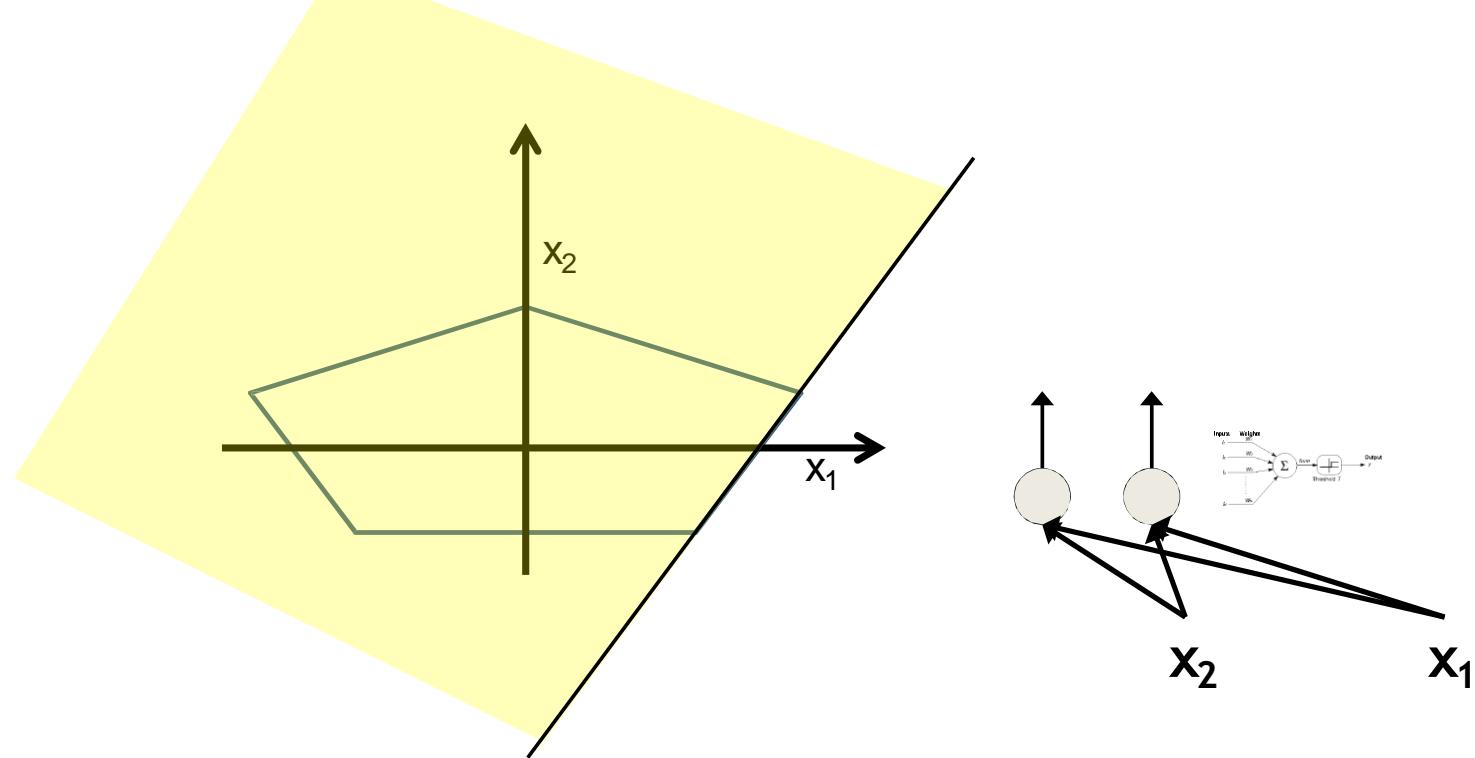
- Build a network of units with a single output that fires if the input is in the coloured area

# Booleans over the reals



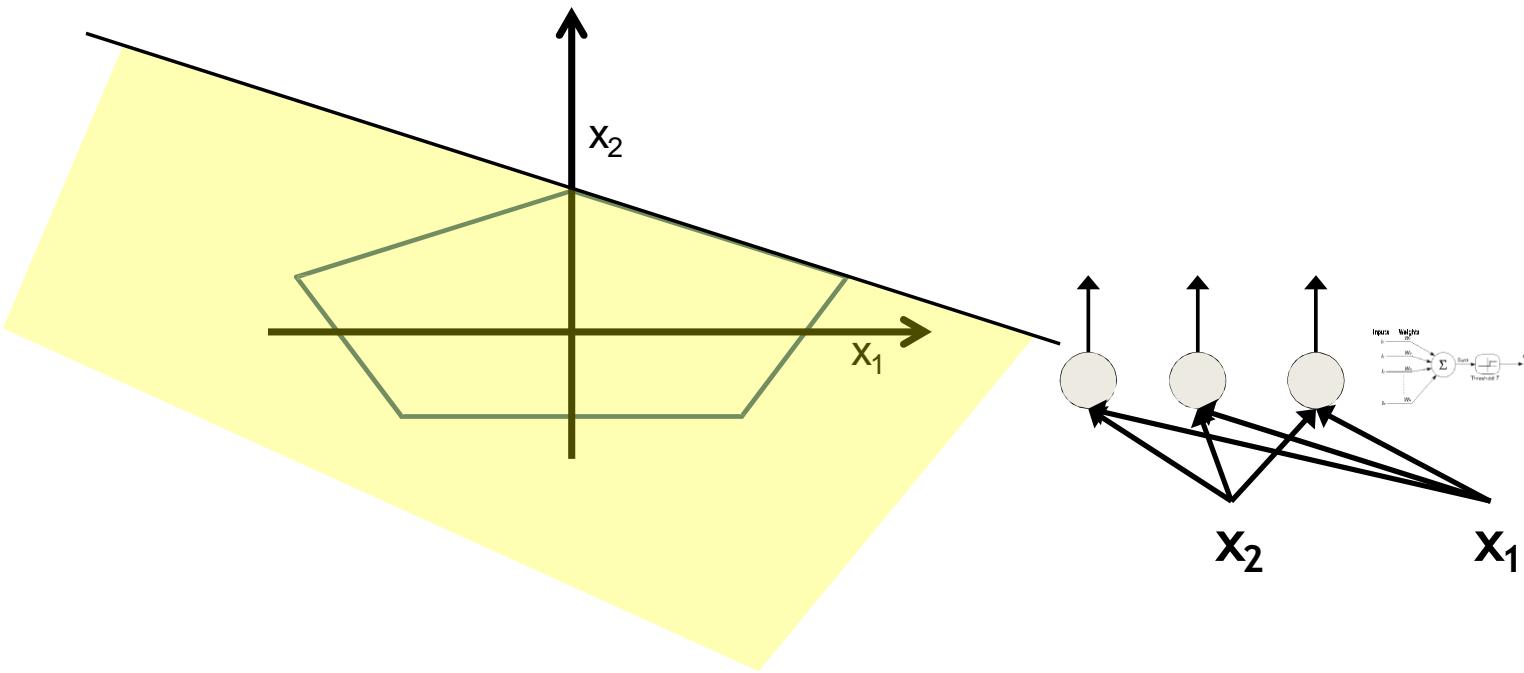
- The network must fire if the input is in the coloured area

# Booleans over the reals



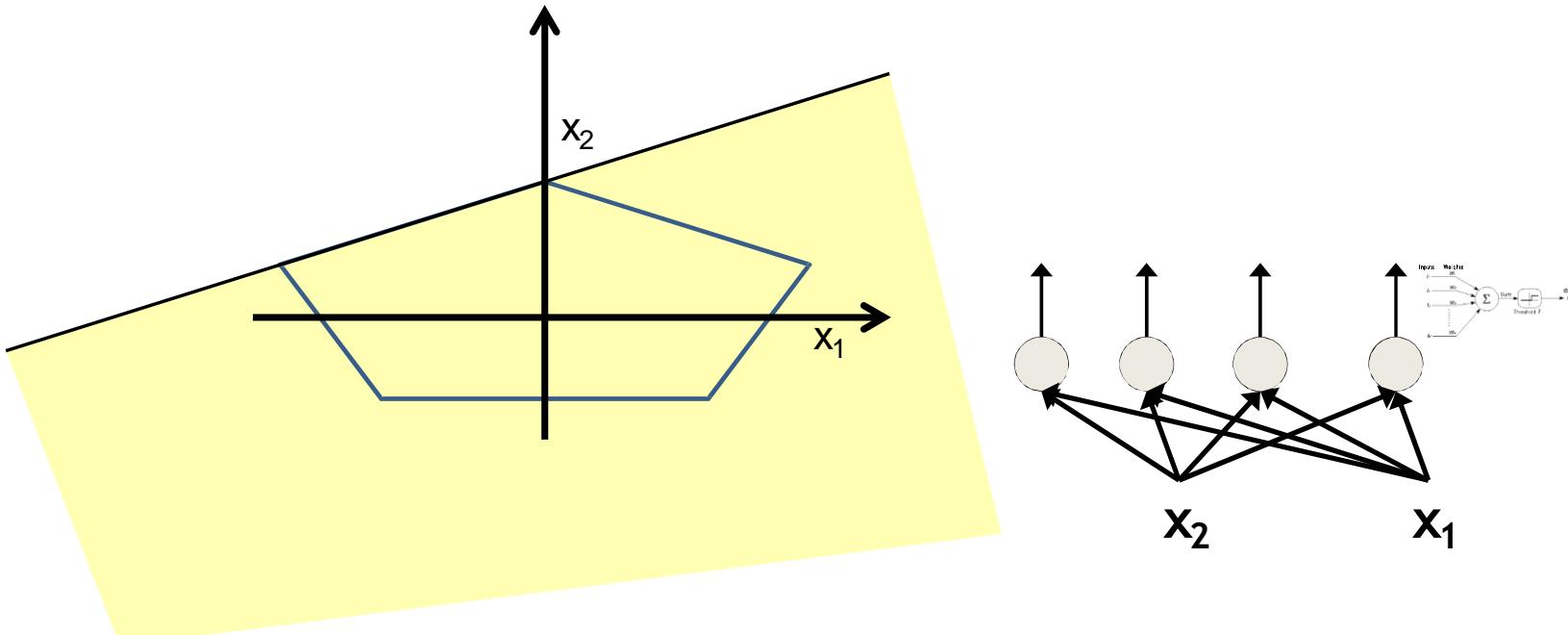
- The network must fire if the input is in the coloured area

# Booleans over the reals



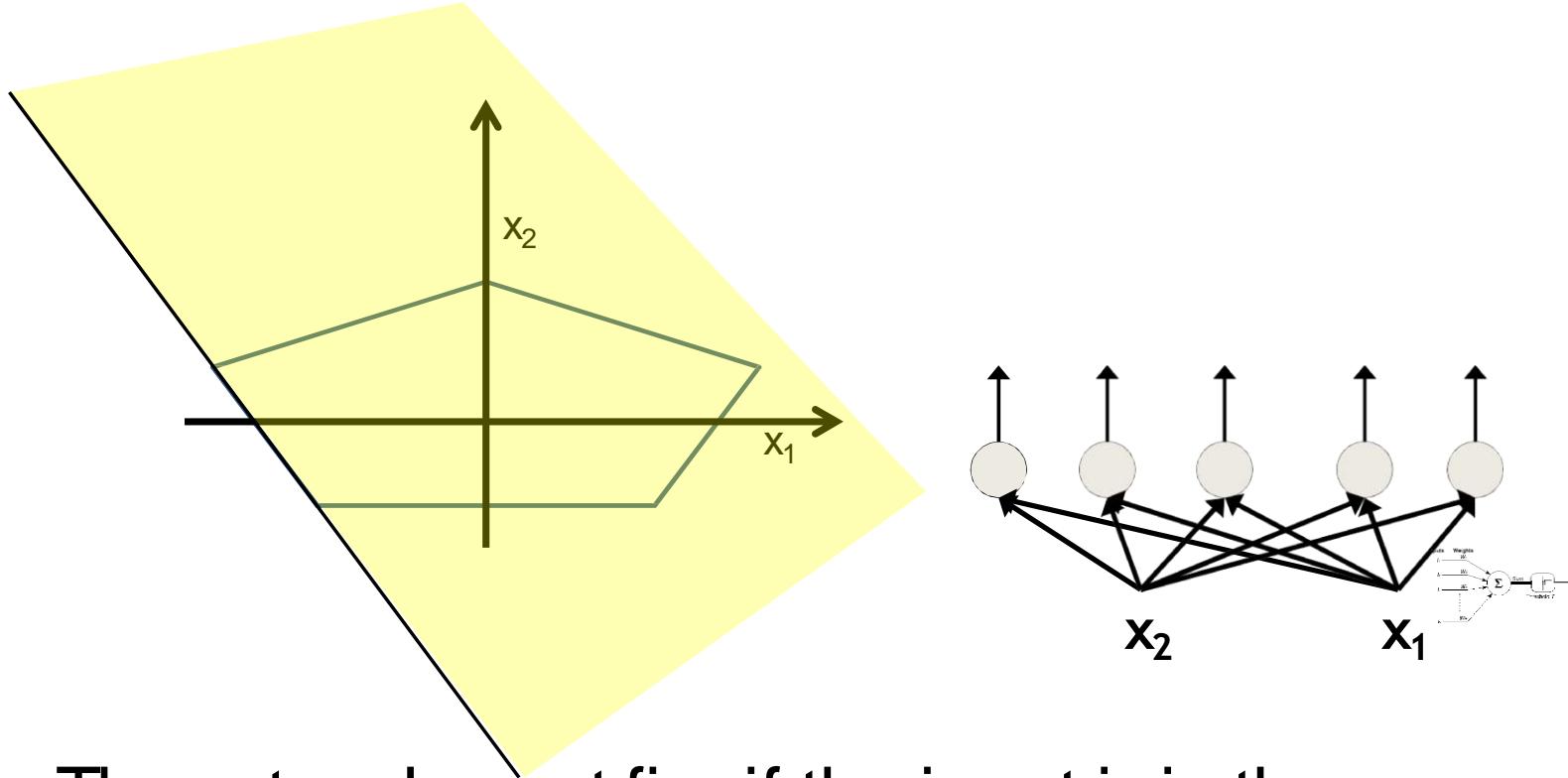
- The network must fire if the input is in the coloured area

# Booleans over the reals



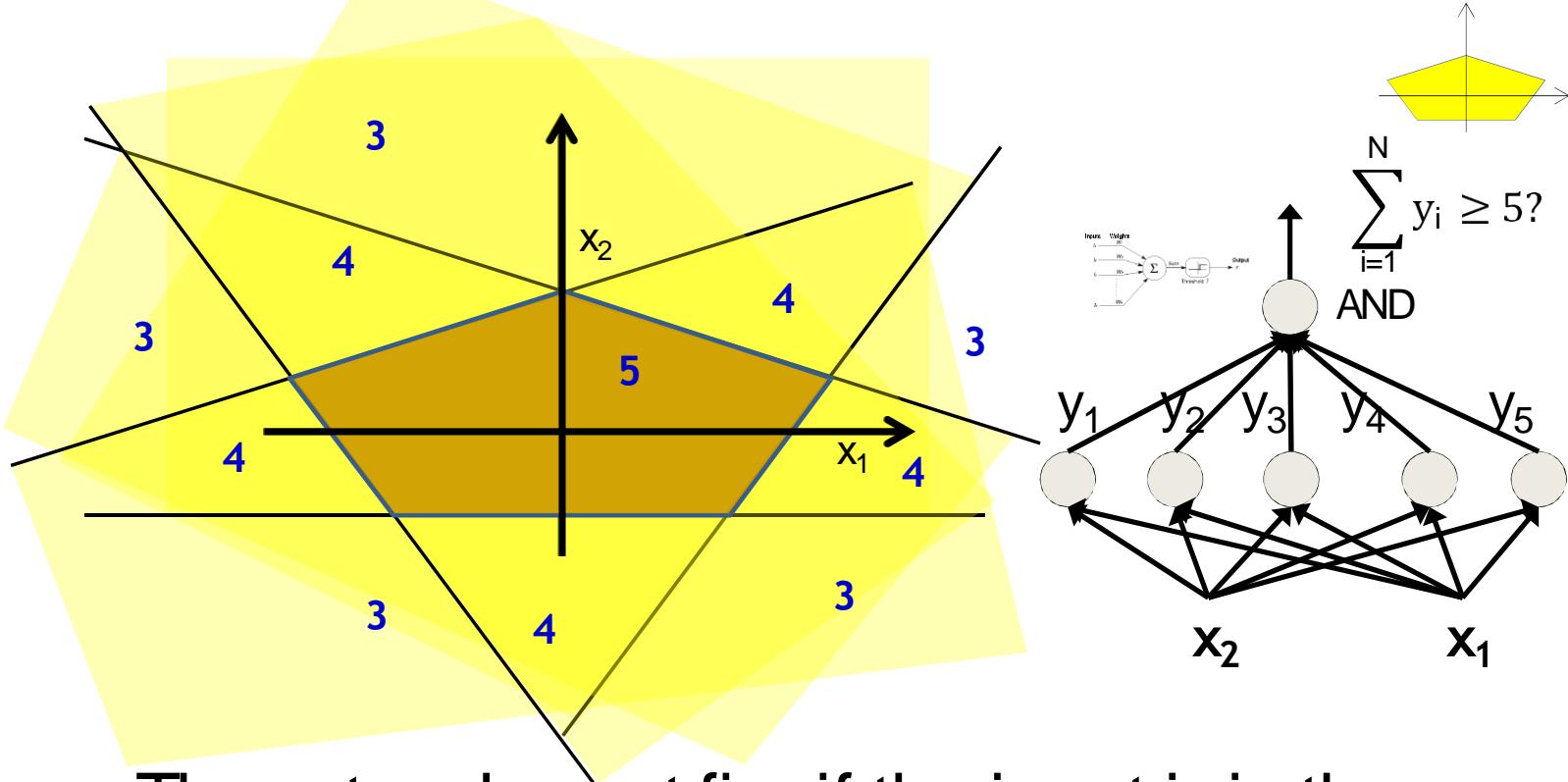
- The network must fire if the input is in the coloured area

# Booleans over the reals



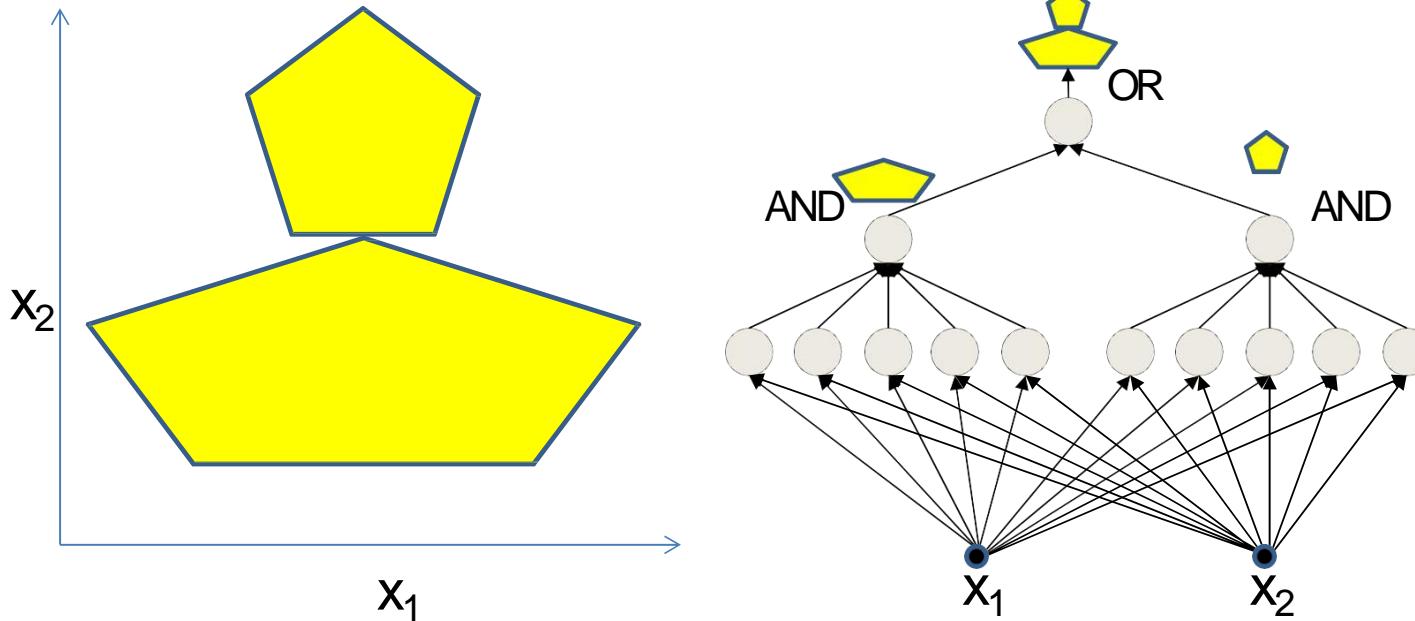
- The network must fire if the input is in the coloured area

# Booleans over the reals



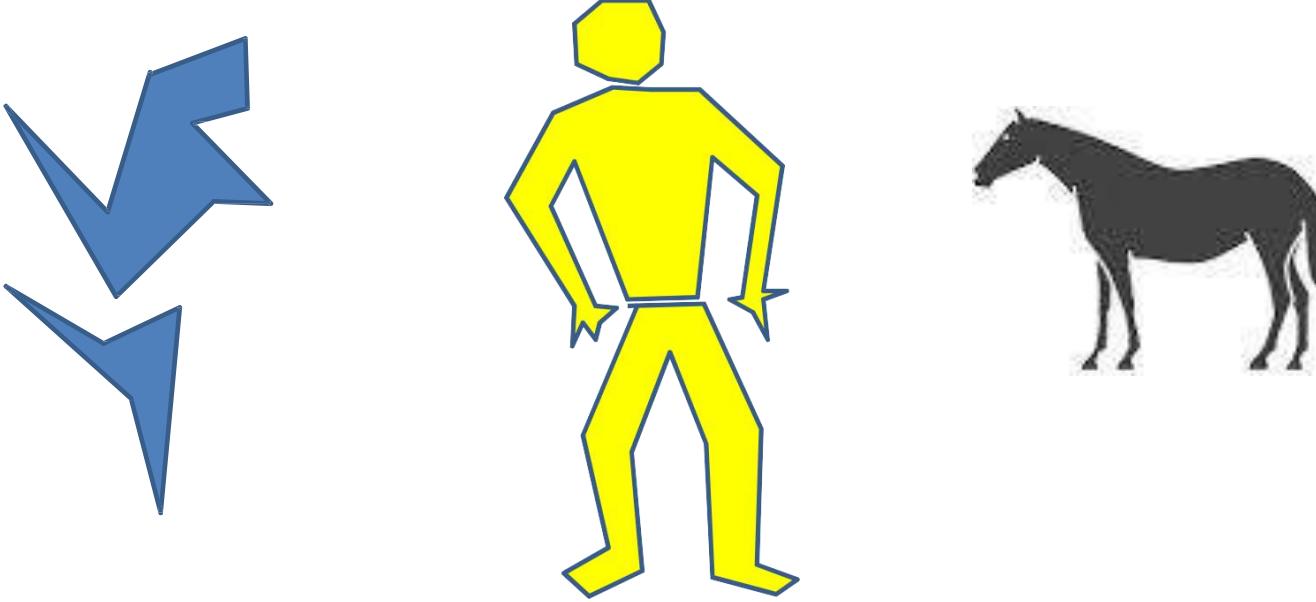
- The network must fire if the input is in the coloured area

# More complex decision boundaries



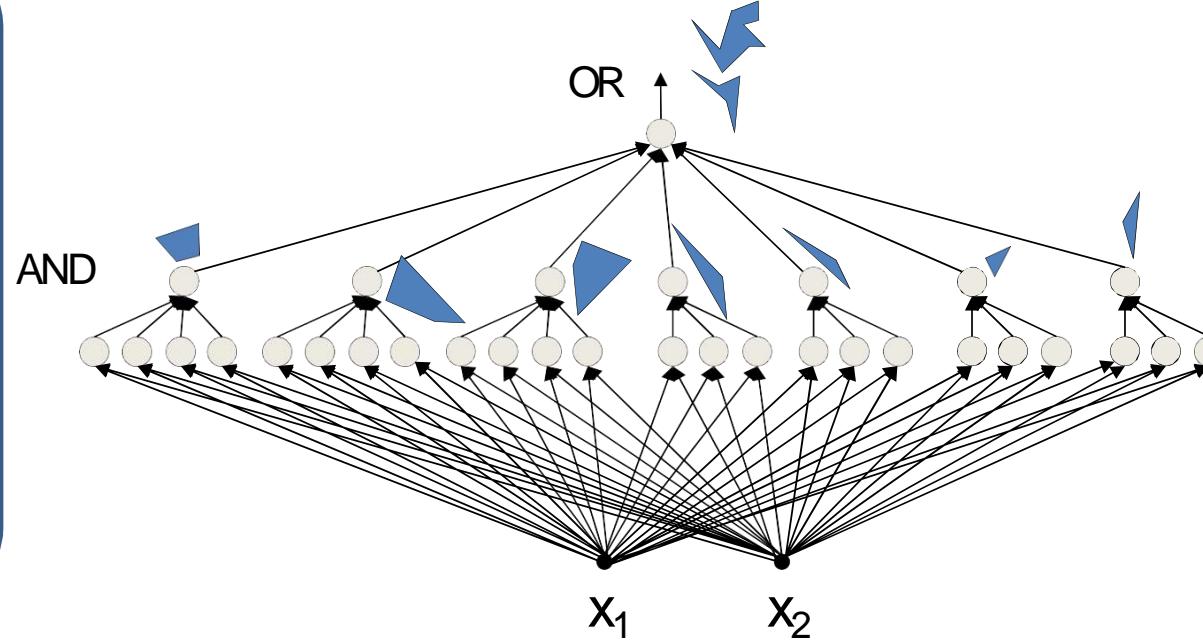
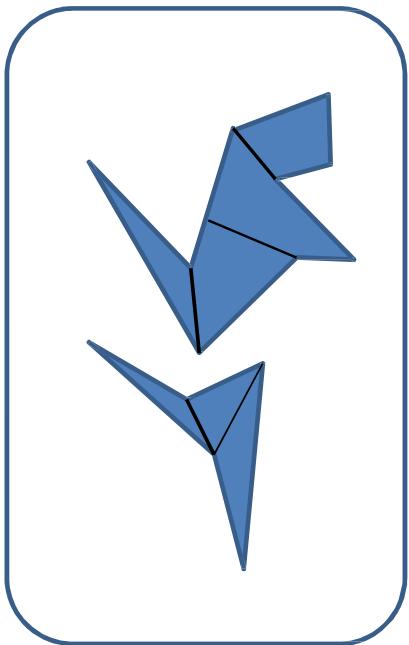
- Network to fire if the input is in the yellow area
  - “OR” two polygons
  - A third layer is required

# Complex decision boundaries



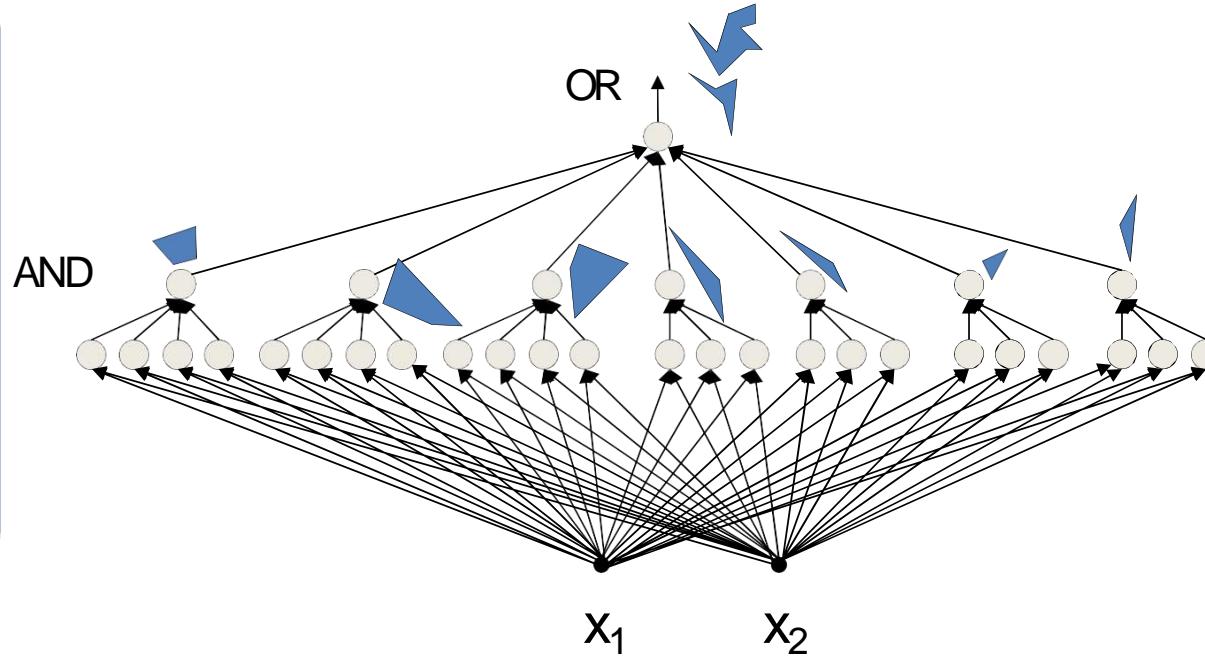
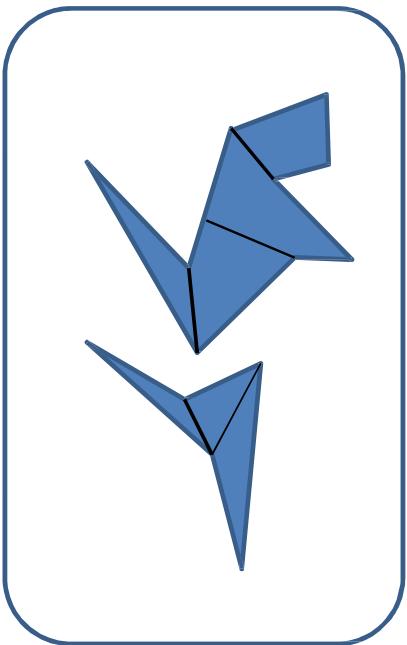
- Can compose *arbitrarily* complex decision boundaries

# Complex decision boundaries



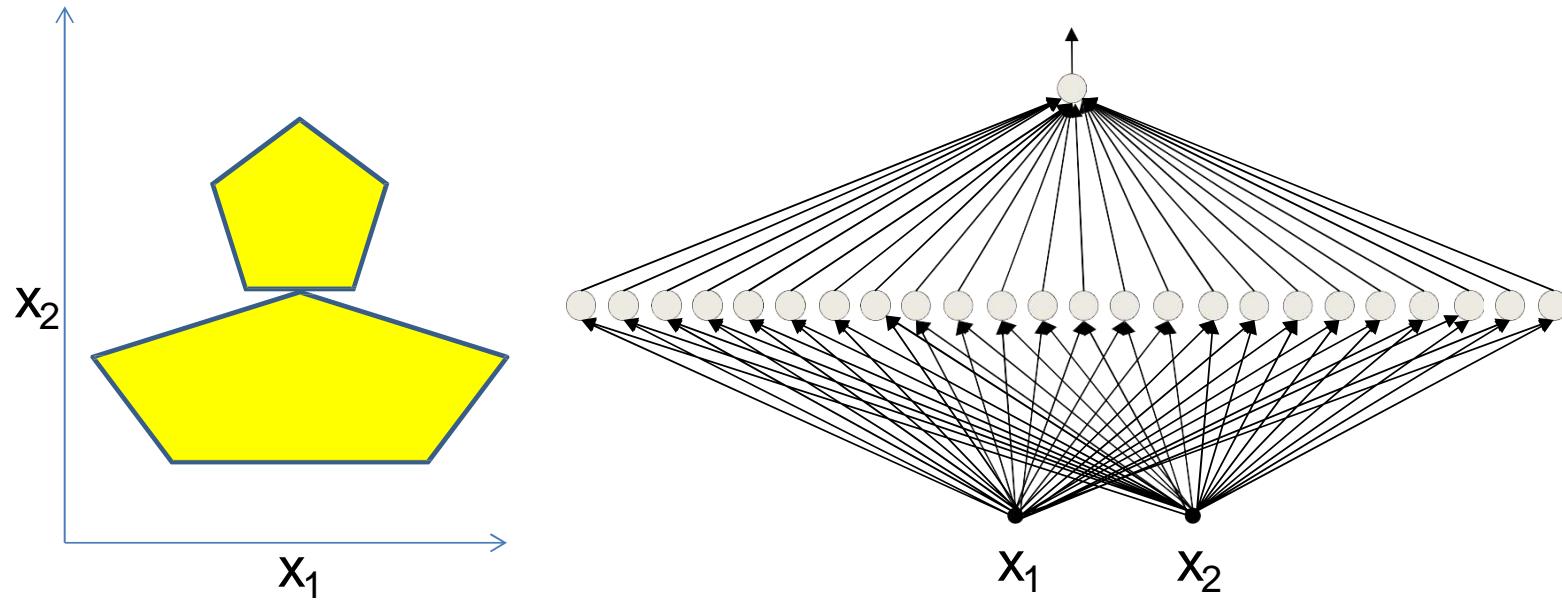
- Can compose *arbitrarily* complex decision boundaries

# Complex decision boundaries



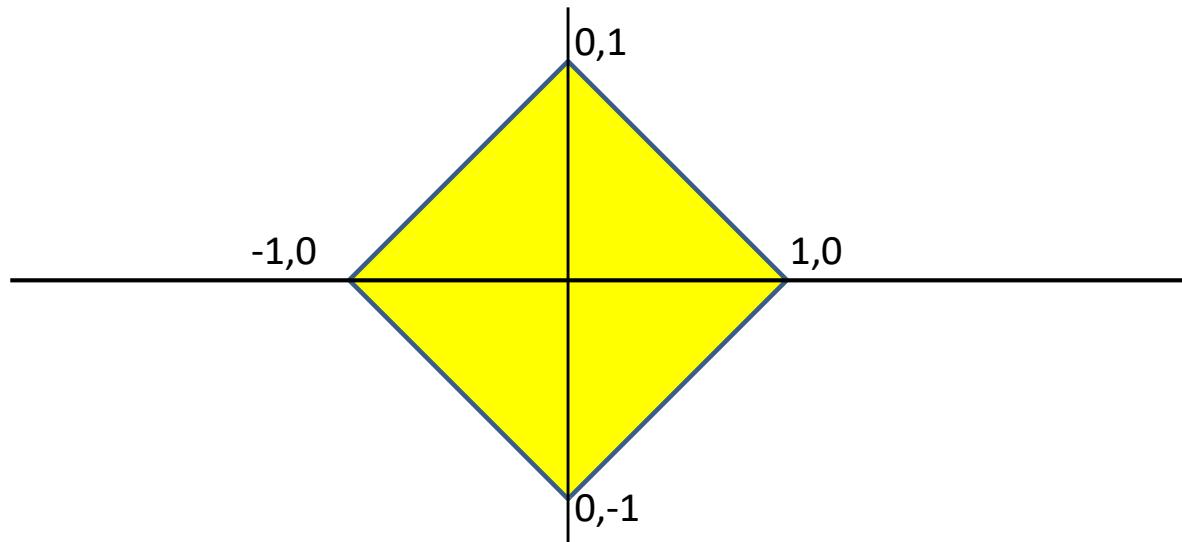
- Can compose *arbitrarily* complex decision boundaries
  - With *only one hidden layer!*
  - **How?**

## Exercise: compose this with one hidden layer



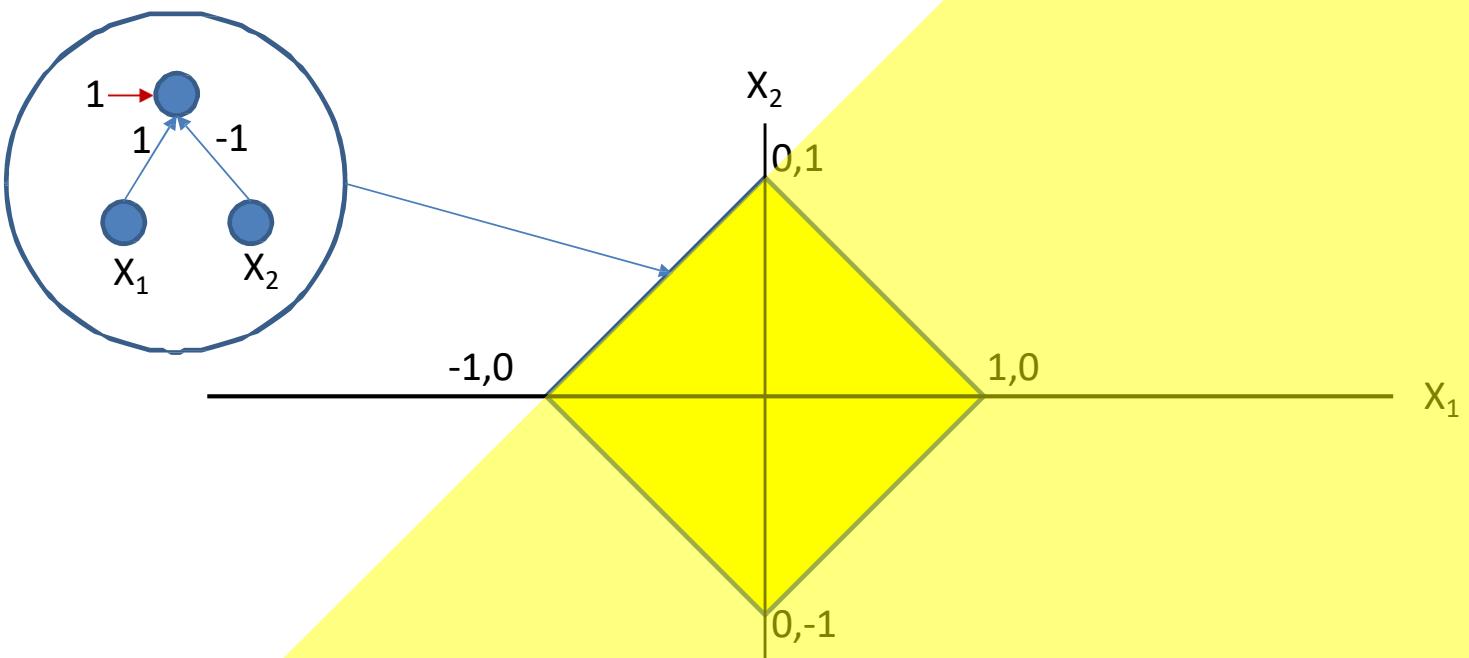
- How would you compose the decision boundary to the left with only *one* hidden layer?

# Construct by hand



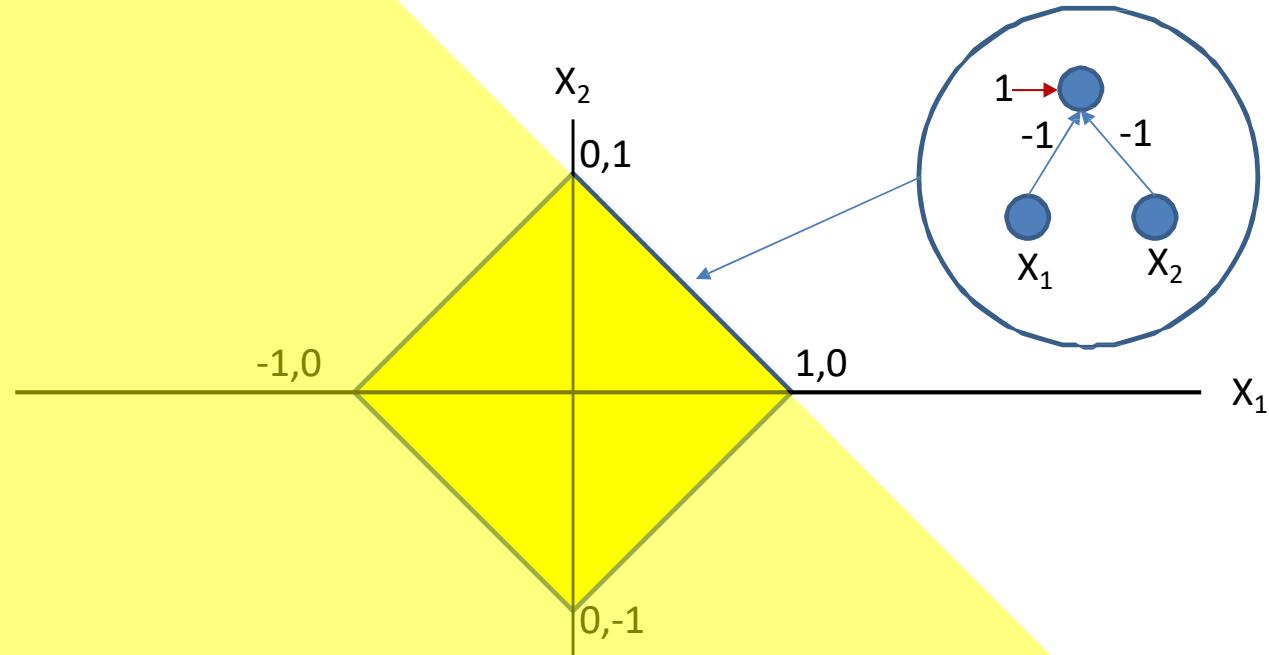
- Given a function, *handcraft* a network to satisfy it
- E.g.: Build an MLP to classify this decision boundary

# Construct by hand



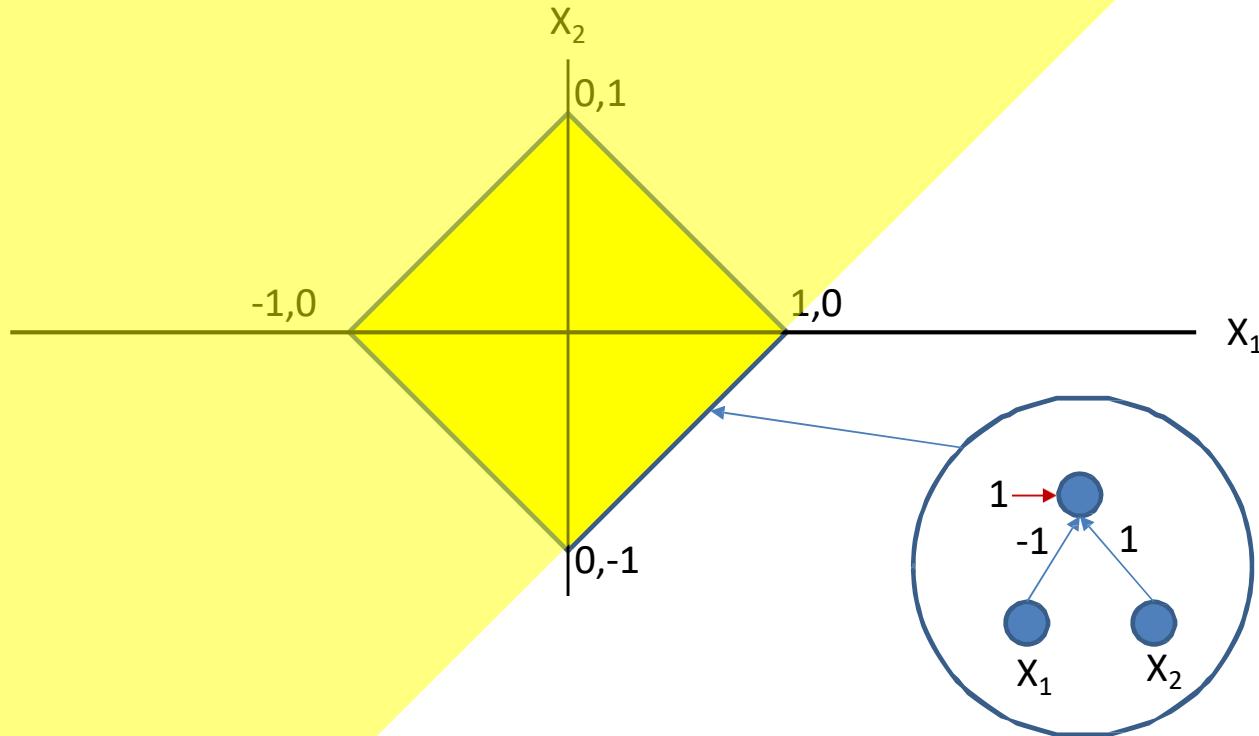
Assuming simple perceptrons:  
 $\text{output} = 1 \text{ if } \sum_i w_i x_i + b_i \geq 0, \text{ else } 0$

# Construct by hand



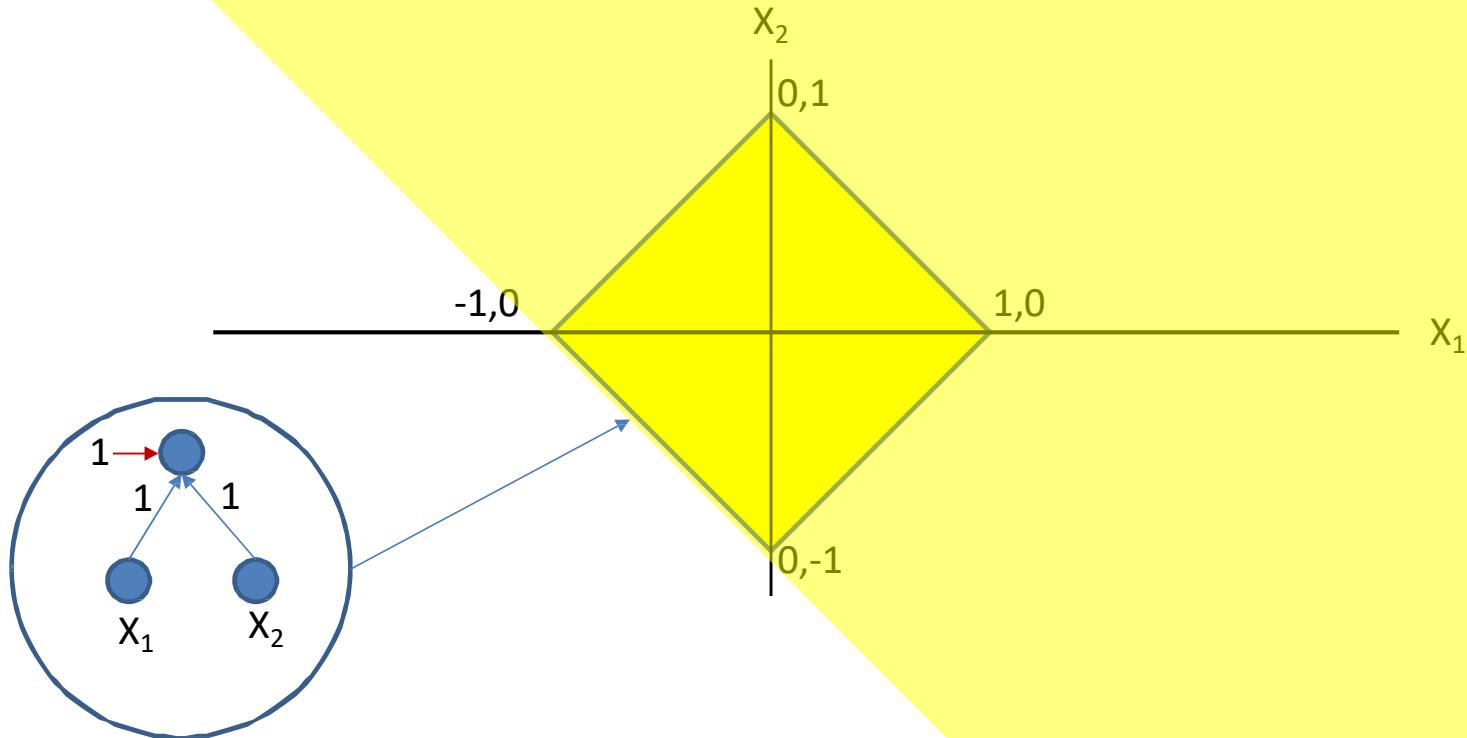
Assuming simple perceptrons:  
 $\text{output} = 1 \text{ if } \sum_i w_i x_i + b_i \geq 0, \text{ else } 0$

# Construct by hand



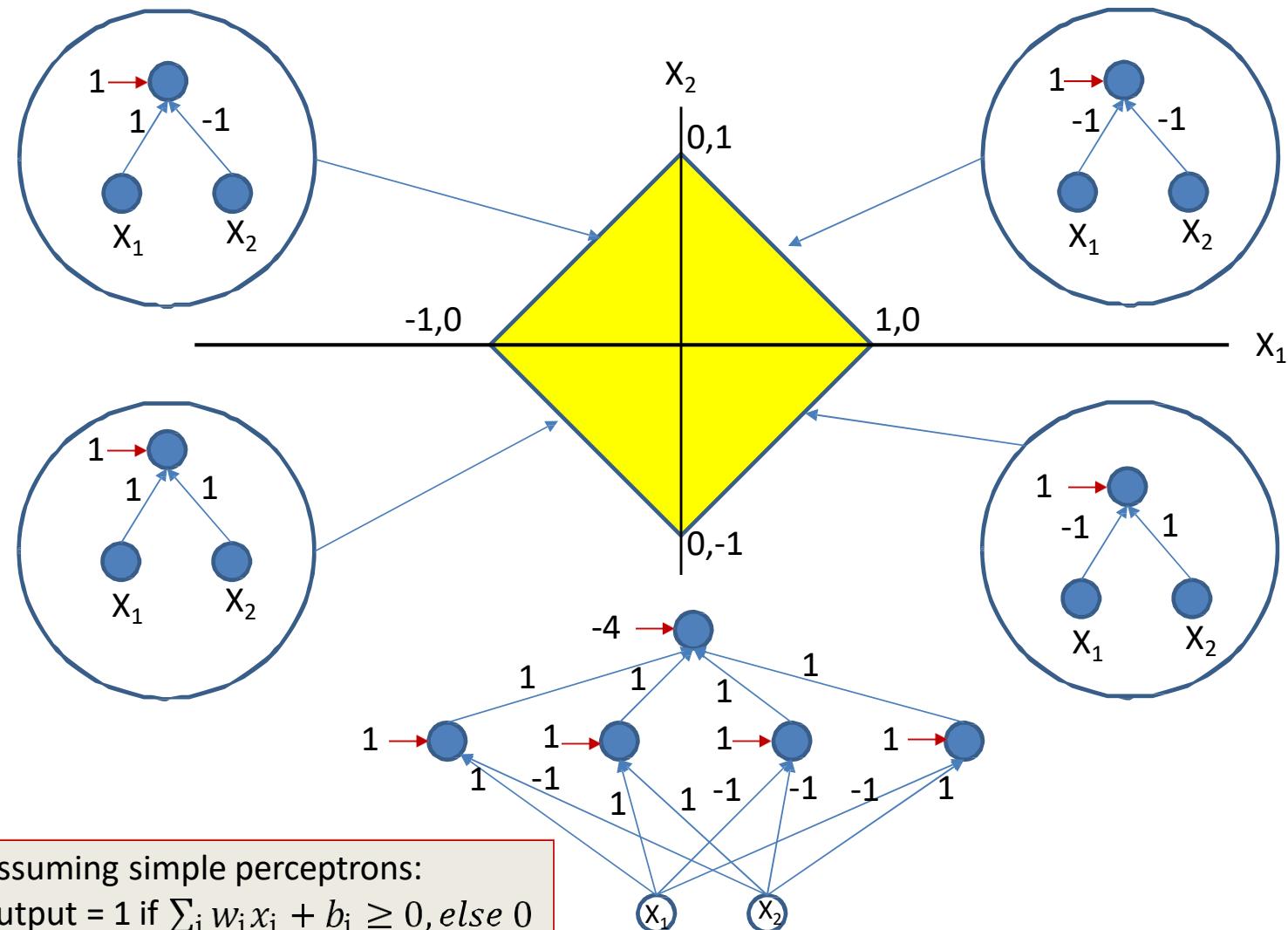
Assuming simple perceptrons:  
 $\text{output} = 1 \text{ if } \sum_i w_i x_i + b_i \geq 0, \text{ else } 0$

# Construct by hand

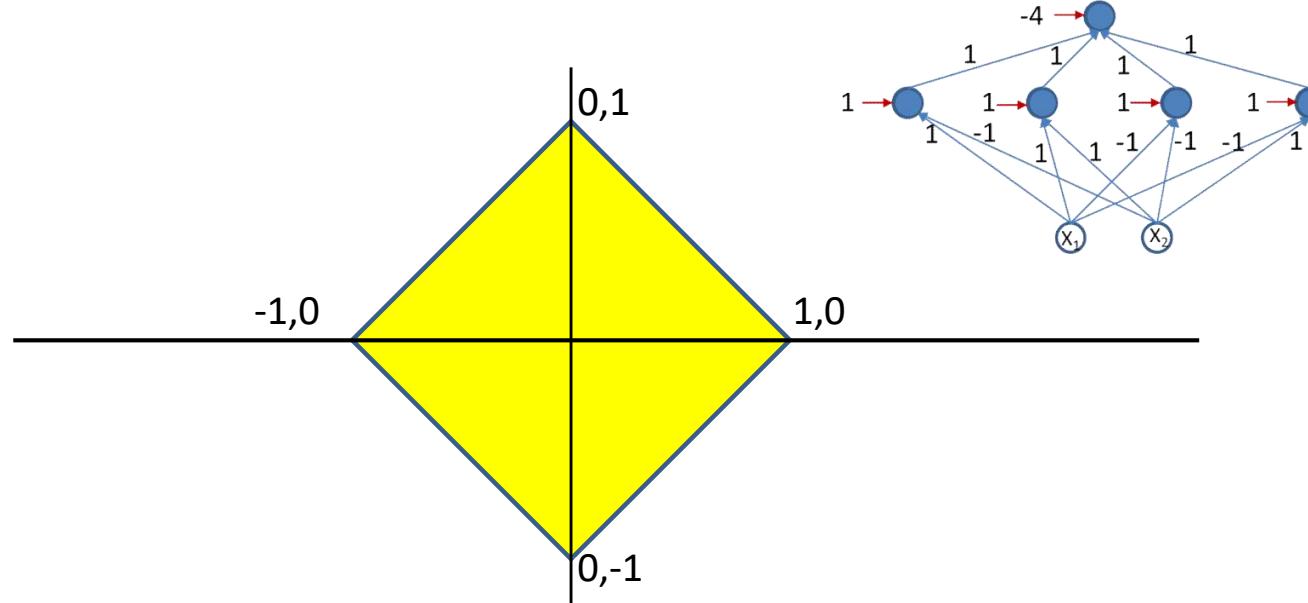


Assuming simple perceptrons:  
 $\text{output} = 1 \text{ if } \sum_i w_i x_i + b_i \geq 0, \text{ else } 0$

# Construct by hand



# Construct by hand



- Given a function, *handcraft* a network to satisfy it
- E.g.: Build an MLP to classify this decision boundary
- Not possible for all but the simplest problems..

# How to Specify/Design Perceptron Parameters

# $x_1$ OR $x_2$ with Perceptron (Step Threshold)

Perceptron with hard/step threshold

Output = 0 if input  $w_1*x_1 + w_2*x_2 < \text{threshold } T$   
= 1 if input  $\geq T$

From OR truth table,

$w_1*0 + w_2*0 < T$  implies  $T > 0$

$w_1*1+w_2*0 \geq T$  implies  $w_1 \geq T$

$w_1*0+w_2*1 \geq T$  implies  $w_2 \geq T$

$w_1*1+w_2*1 \geq T$  implies  $w_1+w_2 \geq T$

Choose,  $T=1$  (you can choose any  $T > 0$ )

Then  $w_1=1$  (you can choose any value  $\geq T$ )

Similarly choose  $w_2=1$  (you can choose any value  $\geq T$ )

$w_1+w_2 \geq T$  is automatically satisfied for  $w_1=w_2=1$  and  $T=1$

X1 AND x2 Truth Table

x1	x2	Output
0	0	0
1	0	1
0	1	1
1	1	1

Thus, AND can be realized with  $w_1=1$ ,  $w_2=1$ ,  $T=1$ . Important to note that other values of  $w_1$ ,  $w_2$ , and  $T$  can also implement OR, as long as the about 4 inequalities are satisfied

# $x_1$ AND $x_2$ with Perceptron (Step Threshold)

Perceptron with hard/step threshold

Output = 0 if input  $w_1 \cdot x_1 + w_2 \cdot x_2 < \text{threshold } T$   
= 1 if input  $\geq T$

From OR truth table,

$w_1 \cdot 0 + w_2 \cdot 0 < T$  implies  $T > 0$

$w_1 \cdot 1 + w_2 \cdot 0 < T$  implies  $w_1 < T$

$w_1 \cdot 0 + w_2 \cdot 1 < T$  implies  $w_2 < T$

$w_1 \cdot 1 + w_2 \cdot 1 \geq T$  implies  $w_1 + w_2 \geq T$

Choose,  $T = 2$  (you can choose any  $T > 0$ )

Then  $w_1 = 1$  (you can choose any value  $< T$ )

Similarly choose  $w_2 = 1$  (you can choose any value  $< T$ )

$w_1 + w_2 \geq T$  is automatically satisfied for  $w_1 = w_2 = 1$  and  $T = 2$

X1 AND x2 Truth Table

x1	x2	Output
0	0	0
1	0	0
0	1	0
1	1	1

Thus, AND can be realized with  $w_1 = 1$ ,  $w_2 = 1$ ,  $T = 2$ . Important to note that other values of  $w_1$ ,  $w_2$ , and  $T$  can also implement AND, as long as the about 4 inequalities are satisfied

# Linear Classification with Perceptron

Perceptron with hard/step threshold

Output = -1 if input  $w_1*x_1 + w_2*x_2 < \text{threshold } T$   
= 1 if input  $\geq T$

From training data,

$$w_1*(-2) + w_2*0 < T \text{ implies } -2w_1 < T$$

$$w_1*1 + w_2*(-1) < T \text{ implies } w_1 - w_2 < T$$

$$w_1*0 + w_2*1 \geq T \text{ implies } w_2 \geq T$$

$$w_1*1 + w_2*1 \geq T \text{ implies } w_1 + w_2 \geq T$$

Choose,  $w_1=1$

Then  $T > -2$ , choose  $T = -1$  (you can choose any value  $> -2$ )

Then,  $1 - w_2 < -1$  or  $w_2 > 2$ . Choose,  $w_2=3$ . (you can choose any value  $> 2$ )

Then,  $w_1 + w_2 \geq T$  is automatically satisfied

Training Dataset

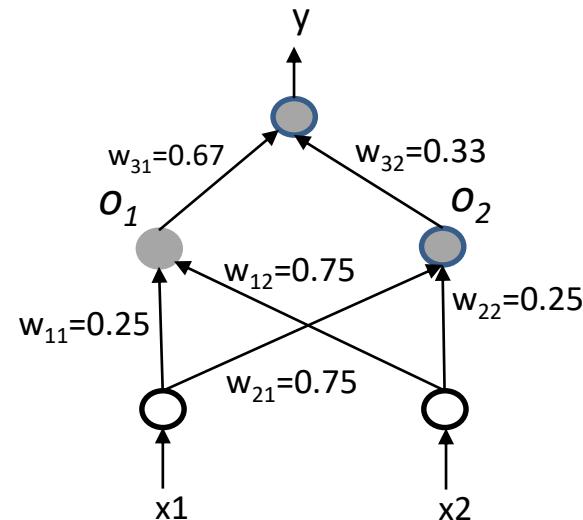
x1	x2	Class
-2	0	-1
1	-1	-1
0	1	1
1	1	1

Thus, classification can be realized with  $w_1=1$ ,  $w_2=3$ ,  $T=-1$ . Important to note that other values of  $w_1$ ,  $w_2$ , and  $T$  can also classify the training dataset, as long as the about 4 inequalities are satisfied

# Parameter Updates in ANN

Leaky ReLU activation function that generates output = input, if input  $\geq 0$ , and  $0.1 * \text{input}$  if output  $< 0$ . What will be the weights  $w_{31}$  and  $w_{12}$  in the next iteration with learning rate = 0.1,  $x_1=x_2=1$ , and target output 0?

Assume, squared difference between the actual and target output is used as the loss function, derivative of activation function = 0 at input = 0, and zero bias at all nodes.



. Assume, error  $E = 0.5*(t-y)^2$ . For  $x_1=x_2=1$ , actual output  $y = 1$  and output of hidden nodes are  $o_1=o_2=1$ . Target output  $t = 0$  as specified in the question. Note  $y = w_{31}o_1 + w_{32}o_2$

$$\Delta w_{31} = -\frac{\eta \delta E}{\delta w_{31}} = -\frac{\eta \delta E}{\delta y} * \frac{\delta y}{\delta w_{31}} = \eta(t - y) * o_1 = -0.1.$$

Thus, value of  $w_{31}$  in the next iteration will be

$$w_{31} + \Delta w_{31} = 0.67 - 0.1 = 0.57.$$

$$\Delta w_{12} = -\frac{\eta \delta E}{\delta w_{12}} = -\frac{\eta \delta E}{\delta y} * \frac{\delta y}{\delta w_{12}} = -0.1 * \frac{\delta y}{\delta o_1} * \frac{\delta o_1}{\delta w_{12}} = -0.1 * w_{31} * x_2 = -0.067.$$

Thus, value of  $w_{12}$  in the next iteration will be

$$w_{12} + \Delta w_{12} = 0.75 - 0.067 = 0.683.$$