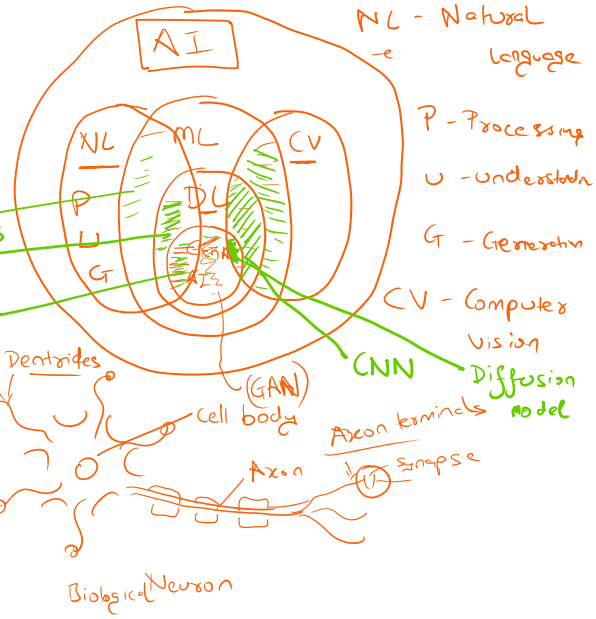
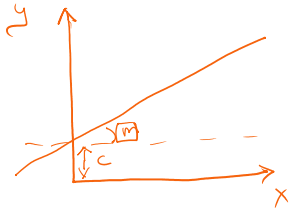
Gen AI – Generate Content

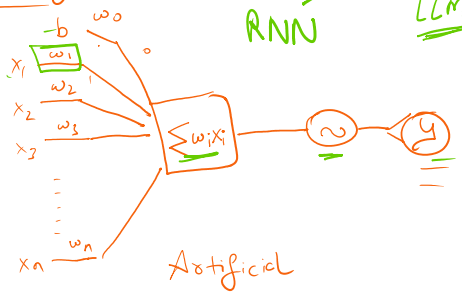
AI – Artificial Intelligence

ML – Machine learning → understand data

DL – Deep learning → processing power







Artificial Neuron

Biological Neuron

$y = mx + c$

Linear reg.

$y = m_1x_1 + m_2x_2 + m_3x_3 + c$

$y = m_1x_1 + m_2z_2$

Data collection

ω = weights

b = bias, c = constant, ω0

x_1 = feature

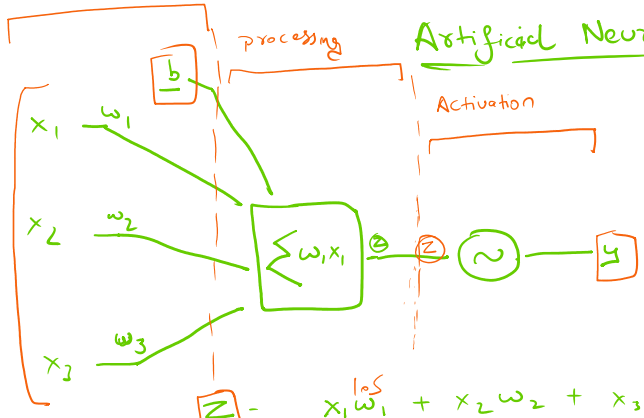
y = output

⊙ - Activation function

Activation function

- Step func.
- Sigmoid func = $\frac{1}{1+e^{-z}} = [0, 1]$
- Relu
- tanh

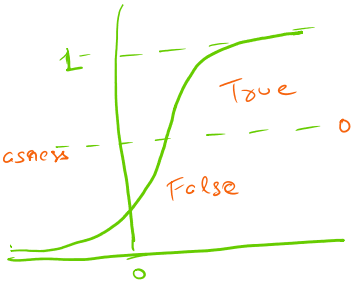
Artificial Neuron



x = feature

ω = weights

b = (helper value) biasness



$z = x_1\omega_1 + x_2\omega_2 + x_3\omega_3 + b$

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

Building block of DL

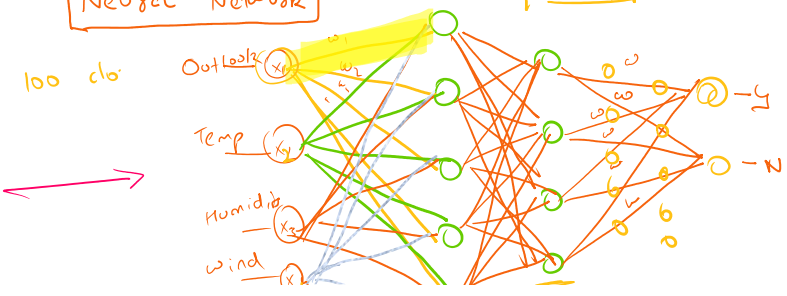
Artificial Neuron

Preceptrone

Deep Neural Network

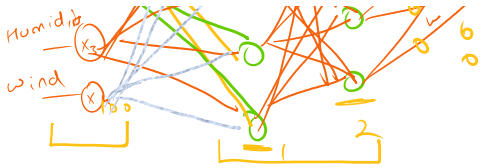
Neural Network

100 cl.

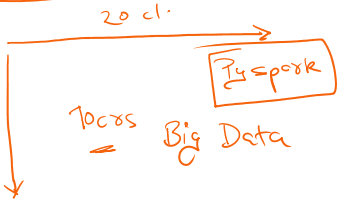


Outlook	Tc	H	ω	Y/N
S	H	L	H	Y
R	L	H	H	N
S	G	L	L	N
R	L	H	L	Y

20 cl.



Shallow Neural Network



Why we need Deep learning instead Machine learning ?

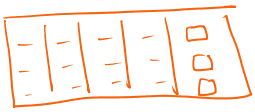
- Machine
  - Structured data
  - Feature Engg.
  - Not good high dimensional data
- Deep learning
  - can work unstructed data - audio, text, img. _ High Dimensiond data
  - High dimensiond data
  - Automatic Feature selection
  - High computation cost

100,000

loss

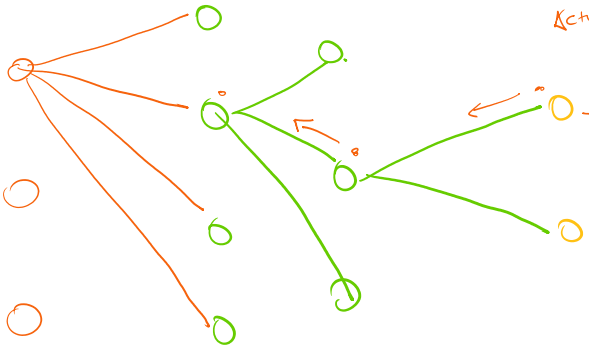
How to train a neural Networks :-

Feed Forward



Step 1

[Fully connected Feed forward neural network]



Actual - y

Step-2

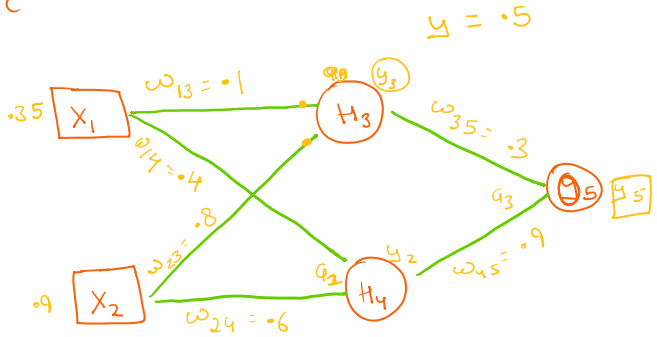
Error = $y - \hat{y}$

Backpropogation

Feed Forward

$$z = \sum x_i w_i$$

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



Step - 1

$a_1 = (w_{13} \times x_1) + (w_{23} \times x_2)$

$(.1 \times .35) + (.8 \times .9) = .755$

$y_3 = \frac{1}{1+e^{-.755}} = .68$

$a_2 = (.4 \times .35) + (.9 \times .6) = .68$

$y_4 = \frac{1}{1+e^{-.68}} = .6637$

$a_3 = (.68 \times .3) + (.6637 \times .9) = .8013$

$y_5 = \frac{1}{1+e^{-.8013}} = .6902$

Step - 2

Error = $y - \hat{y}$

$.5 - .69 = -.19$

Step-3 Backpropogation

η = learning rate

$\Delta w_{ji} = \eta \cdot \delta_j O_i$

$\delta_j = O_j(1-O_j)(t_j - O_j)$ if j is output

$\delta_j = O_j(1-O_j) \sum \delta_k w_{kj}$ if j is hidden neuron

δ_j = error measure of j^{th}

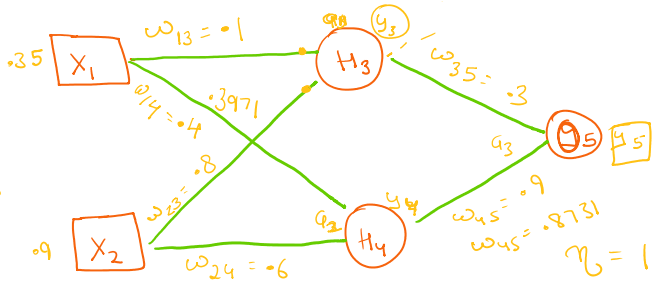
t_j = correct output

O = output of neuron

For output neuron

$\delta_5 = \hat{y}(1-\hat{y})(y-\hat{y})$

$= .6902(1-.6902)(.5-.6902) = -0.0406$



$\delta_3 = y_3(1-y_3)(w_{35} \times \delta_5)$

$= .68(1-.68)(.3 \times -.0406) = -0.00265$

$\delta_4 = y_4(1-y_4)(w_{45} \times \delta_5) =$

$.6637(1-.6637)(.9 \times -.0406) = -.00815$

updating weights

$\Delta w_{ji} = \eta \delta_j y_i$

$\Delta w_{45} = 1 \times -.0406 \times .6637$

$= -0.0269$

(new) $w_{45} = \Delta w_{45} + w_{45}$ (old)

$-0.0269 + .9$

$= .8731$

$\Delta w_{14} = \eta \delta_4 y_1$

$= 1 \times -.00082 \times .35$

$= -.00287$

$w_{14} = .3971$

-.19

2nd Iteration

→ Feed Forward

→ Error - .15

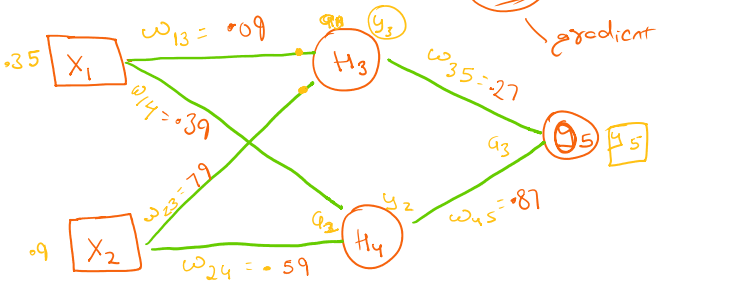
→ Backpropogation

3rd Iteration

- Feed Forward
- Error = .09 .149 ↓

4th

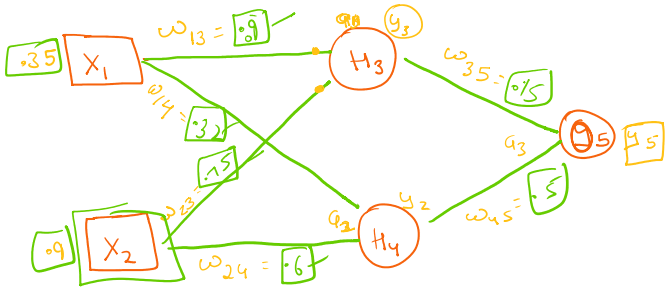
Error = .1485



(2) $\Delta w_{45} = -.0026$

(3rd) $\Delta w_{45} = -.000004$

gradient



- Different types of Neural Networks:

Day-2

Saturday, September 7, 2024 4:20 PM

Type of Neural Networks

- Fully connected Neural Network : (FCNN)
- Convolutional Neural Network : (CNN)
- Recurrent Neural Network (RNN)
- Long Short term memory : (LSTM)

} Supervised

- GAN - Generative Adversarial Network
- Autoencoder

} Unsupervised

★ Interview question

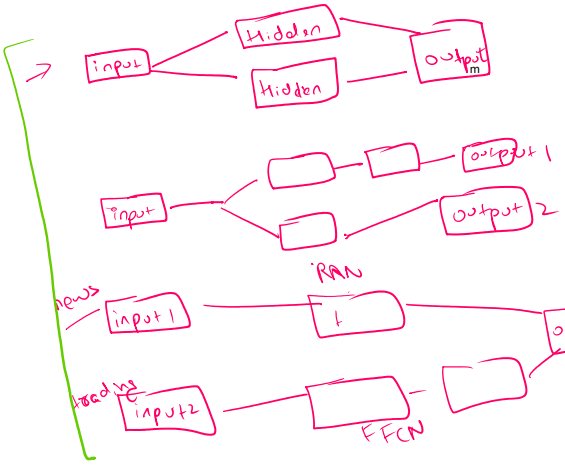
- Limitations of FCNN
  - No. of connection
  - Not possible for sequential data

37 connection

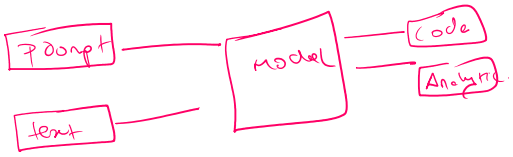
→ Different Archecture of Neural Network :-

- one input one output :-

- One input multiple output :-



- multiple input one output :-
- multiple input multiple output :-



$\Delta w = -.0026$

$\Delta w = 1$

→ Activation function :-

- Sigmoid ; $\frac{1}{1+e^{-z}}$

Range :- (0 to 1)

→ Vanishing Gradient

→ Exploding Gradient

→ ◦ Use case:
- binary classification

→ ◦ Pros:
probability based values which are interpretable

→ ◦ Cons:
- Vanishing Gradient

◦ Hyperbolic tan → tanh = $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Range : [-1, 1]

Use Cases:
- Generally used in hidden layer

Pros:
- Zero centered output helps optimizer
- less likely to saturate

Cons:
Vanishing gradient

→ Relu $ReLU(x) = max(0, x)$ Rectified Linear Unit

Range = [0, ∞)

Use Cases:
Default activation for hidden layer

Pros:
- Computationally efficient
- Mitigate vanishing gradient

Cons:
Dying Relu problem, Not zero centered

4) Leaky ReLU [0, ∞) Parametric ReLU PReLU · ELU - Exponential Linear unit

LR = $\begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \le 0 \end{cases}$ ⟶ $\begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \le 0 \end{cases}$ ⟶ $\begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \le 0 \end{cases}$

◦ α = is trained

LR - αx if $x \le 0$ ⟶ αx if $x \le 0$ ⟶ $\alpha(e^x - 1)$ if $x \le 0$

α = 0.01 — very small number

α = is trained

Range = $(-\infty, \infty)$ ⟶ $(-\alpha, \infty)$

Use case:

to solve Dying Relu Problem.

in deep networks to improve learning process

Pros:

Prevent dead neurons

α is Adaptive

Pros: Dying Relu Vanishing

Cons: not zero centered, α is tricky

Cons: increase model complexity

Cons: more complex

- Swish :

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

$x \times 0 = 0$

-100000×0.0001

$-10 \times (0, 1)$ — ve

Range : $(-\infty, \infty)$

Use:

deep network

Pros:

- Avoid zero gradient
- Smooth & non-monotonic

Cons:

- Computationally expensive

Softmax :

$$= \frac{e^{x_i}}{\sum_{j}^{n} e^{x_j}}$$

example

normal

$\frac{e^{10}}{e^{10} + e^{20} + e^{30}}$ 3 class

output (softmax)

10 — 0.2

20 — 0.3

30 — 0.5

1 1

Range = [0, 1]

use case

- multiclass classification

Pros

Convert logits to prob. distribution

Cons

- outlier data
- class are mutually exclusive

Cons
- outlier data
- Class are mutually exclusive

→ Hard Sigmoid

$$= \min\left(1, \max\left(0, 0.2x + 0.5\right)\right)$$

Range = (0, 1)

$x = -10$

$0.2 \times 10 + 0.5$

$\max(0, -1.5)$

$\min(1, 0)$

$1 \quad 0$

Use:
Faster approx. value

Pros:
computationally faster

Cons
less smooth than regular sigmoid

→ Softplust

$$= \log(1 + e^x)$$

Range: $(0, \infty)$

(40)

Use
smooth approx of Relu

Pros
No dead neuron

Cons
Slow

- Optimizer :-

- Gradient Descent : (GD)

$$\theta_{new} = \theta_{old} - \eta \nabla J(\theta)$$

θ = weight
η = learning rate
$\nabla J(\theta)$ = gradient of loss function

- Use
small datasets

- Pros:
Simple & effective on convex problem

- Cons:
slow convergences & computationally expensive on large dataset

- Cons: slow convergences & computationally expensive on large dataset

- Stochastic GD : (SGD)

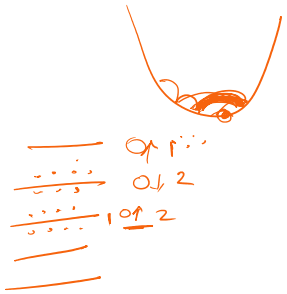
$$\theta = \theta - \eta \nabla J(\theta; x; y)$$

(x_i, y_i) – individual sample

Use Case :
Suitable for large datasets

Pros:
Fast processing & efficient on large dataset

Cons:
Noisy update, may not reach global minima.



- Mini-batch GD :

batch – (mini-batch)

$$\theta = \theta - \eta \nabla J(\theta; \text{batch})$$

use:
hybrid SGD & GD for large dataset, more stable than SGD

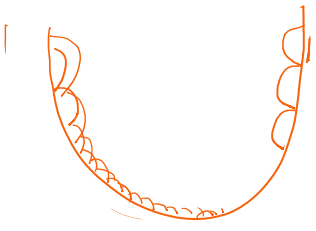
Pros:
- Reduce Noise & variance SGD
- Faster than GD

Cons:
Very large dataset

→ Momentum

$$v = \gamma v + \eta \nabla J(\theta)$$
$$\theta = \theta - v$$

γ = momentum factor = 0.9



use :
useful in training deep network

Pros :
Speedy the convergences

Pros:
- Speedy the convergence
- Reducing the oscillation

Const
- overshoot if momentum is high
- Requires tuning of γ

→ NAG – Nesterov Accelerated Gradient

$$v = \gamma v + \eta \nabla J(\theta - \gamma v)$$
$$\theta = \theta - v$$

Use
it looks ahead before computing the gradient

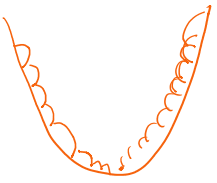
Pros:
better than momentum
prevent overshooting

Const
more complex.

→ Adagrad

$$\theta = \theta - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla J(\theta)$$

G_t = sum of sq of past gradient
ϵ = small constant to avoid zero division.



Use:
Suitable for sparse data like NLP, image

Pros:
Adapting learning rate based on freq of parameter

Cons:
Learning rate keeps decaying.

– RMSprop : Root Mean Sq propagation

β = decay rate = (.9)

$$G_t = \beta G_t + (1-\beta)(\nabla J(\theta))^2$$
(.1)

$$\theta = \theta - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla J(\theta)$$

— ↑.9
— ↑.99
— 1

Pros:
Addressing the adagrad decay prob.

Use :-
Good for non-stationary prob. like in RNN's

Cons :
Sensitive to β value

- Adam :- Adaptive moment Estimation :

First moment Estimation :
$$m_t = \beta_1 m_t + (1-\beta)\nabla J(\theta) \qquad \beta = .9 - .99$$

second moment Estimation :
$$v_t = \beta_2 v_t + (1-\beta)\nabla J(\theta)$$

Bias correction :
$$\hat{m}_t = \frac{m_t}{1-\beta_1}, \quad \hat{v}_t = \frac{v_t}{1+\beta_2}$$

Update :-
$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

ouse coset
- widely used in almost senario.

Pros :
- Combining adaptive learning & momentum learning
- Efficiently works with large datasets & noisy data.
- Faster convergence

Cons :
- multiple tunable parameter
- it can overshoot sometimes

Adamax :
$$v_t = \max\left(\beta_2 v_{t-1}, |\nabla J(\theta)|\right)$$
$$\theta = \theta - \frac{\eta}{v_t} \cdot m_t$$

Use
very large dataset

Pros :
more stable updates

Cons

- Nadam – Nesterov - accelerated adam
$$m_t = \beta_1 m_1 + (1-\beta)\nabla J(\theta)$$

$$\theta = \theta - \frac{\eta}{\sqrt{v_t + \epsilon}} \cdot m_t$$

Use + Pros

adaptive learning of adam + fast convergence nature of NAG

Cons

More sensitive to tuning of parameters

- Adamax

$$m_t = \beta m_{t-1} + (1-\beta)\nabla J(\theta)$$

$$v_t = \max\left(\beta_2 v_{t-1}, |\nabla J(\theta)|\right)$$

$$= \theta - \frac{\eta}{v_t} \cdot m_t$$

Use

β) Better for model that sparse updates

Pros

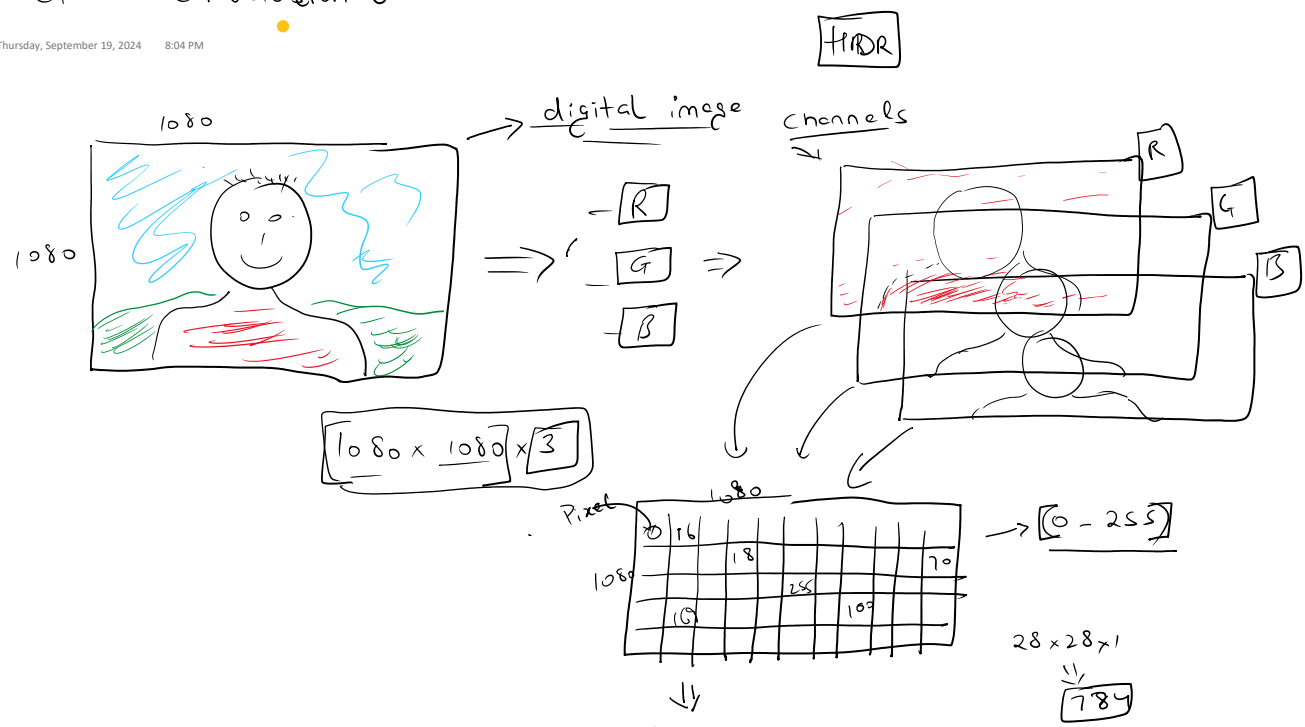
more stable than Adam

Cons

slightly slower than Adam.

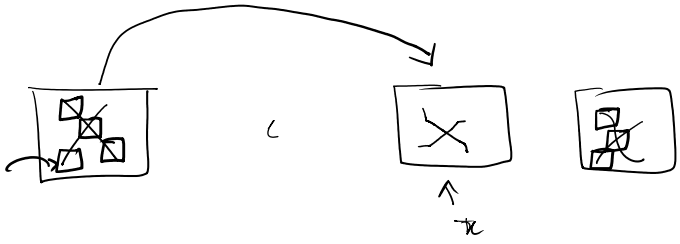
CNN - Convolutional Neural Networks

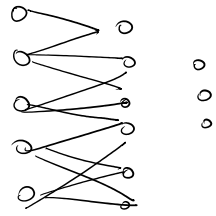
Thursday, September 19, 2024 8:04 PM



1) FCNN :- Computationally Expensive | 1080 x 1080 x 3 = 3,000,000

2)





3,000,000 x 10,000

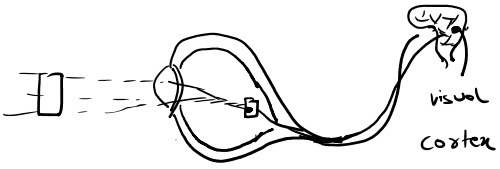
3,000,000,000,000

CNN - ∘ Local area Network → computationally less expensive

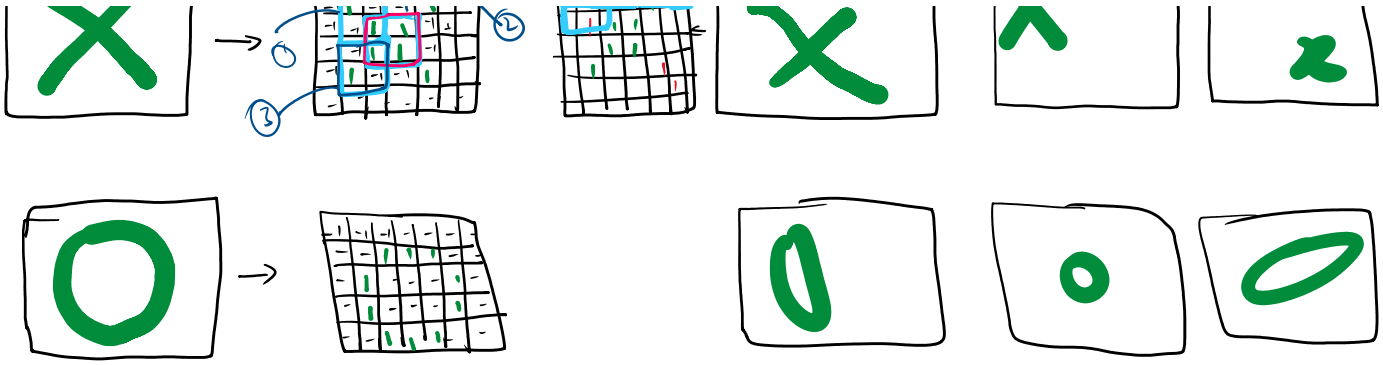
∘ automatically learns spatial features in data

How CNN works

3000000 → 10,000



- ∘ Convolutional layer
- ∘ Relu layer
- ∘ Pooling
- ∘ FCNN



Convolution:

feature

n - size of image

f - size of kernel

(n - f + 1)

(5 - 2 + 1) (7 - 2 + 1)

(4, 6, 1)

Convolution layer

(Kernel size)

(2, 2)

(5, 7, 1)

f1 f2 f3

stride

(4, 6, 3)

Relu - making all ave to zero

Relu()

Max Pooling

Pooling layer :

(Dimension red^n)

(3, 4)

(1, 2)

Max - to get sharpe edge

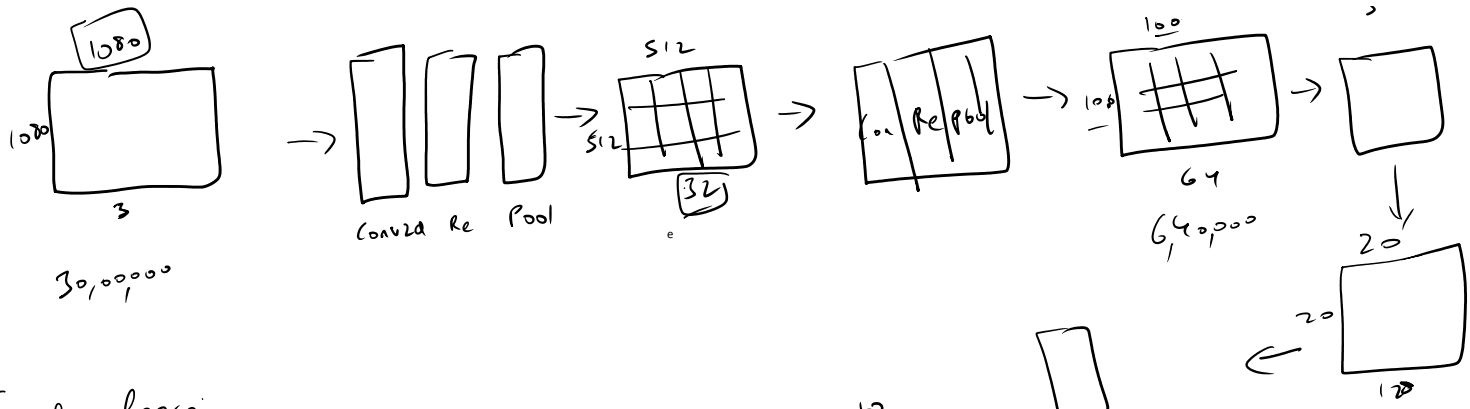
Avgpool when we want to retain max info.

Flatten

1080

512

100



Transfer learning:

- Fine tuning :- (Full / Partial)
- feature Extractor :-

when to use FCNN transfer learning

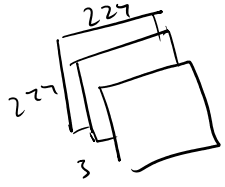
→ Why we need transfer learning :-?

- Reduce training time :-
- Avoid overfitting
- Data req. are less
- Performance

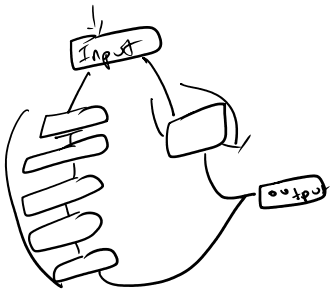
→ CNN-based Architectures :-

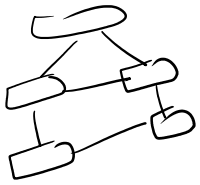
Application :- (32, 32, 1)

- LeNet-5 (1998) ↕
  - 2 Conv layes (5,5), tanh
  - 2 Avg. pooling
  - 2 fully CNN with softmax
- AlexNet (2012) – ImageNet Challenge :-
  - 5 conv. layer – relu (224, 224, 3)
  - 3 maxpooling –
  - 3 FCNN
  - 2 dropout



- VGG Net (2014) — 19 layers
- GoogleNet (Inception)
- ResNet 2015 — 152 layer
- DenseNet 2016
- MobileNet 2017
- EfficientNet





RCNN :- Region based CNN

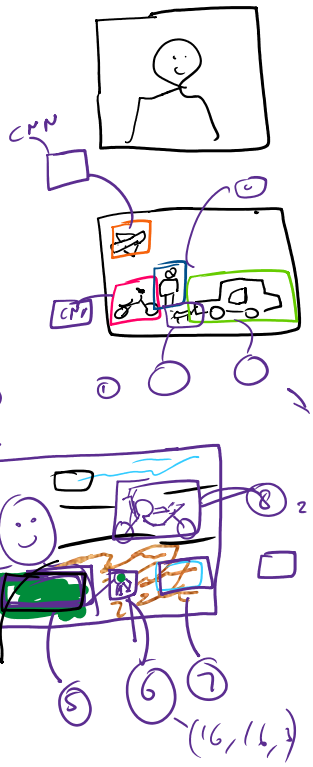
CNN - image cla

4 Layers

→ input shape = (32, 32, 3)

1080 x
240
28

→ Region Proposal :-
→ Classification



Step : 1

- Selective Search : 2000

2000
- color
- texture
- size
- shape

Step 2

- cropping / reshaping to desired input of CNN

Step 3 :

Classifier learning — Feature Extractor
— Classification

Step 4 :- Bounding Box Regression :-

Dist. of RCNN

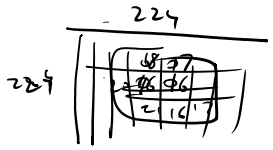
- Slow
- Training time
- Complex pipeline

- Fast RCNN → 2 ↑
- Faster RCNN — 20 ↑
- Mashed RCNN

- Selective Search : 2000
  - Superpixel Segmentation

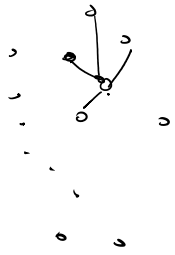
224

- Superpixel Segmentation

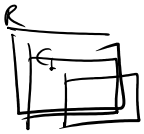


Math intuition :-

- Similarity weights $\omega_{ij} = |I_i - I_j|$



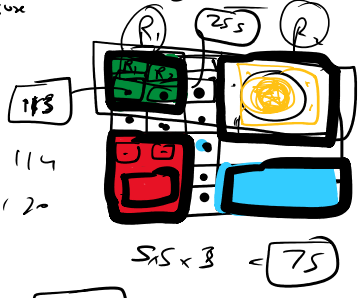
- Color Histogram
- Texture Histogram
- Size
- fill



2000

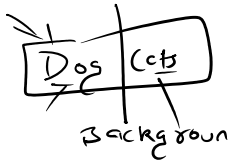
Similarity Score :-

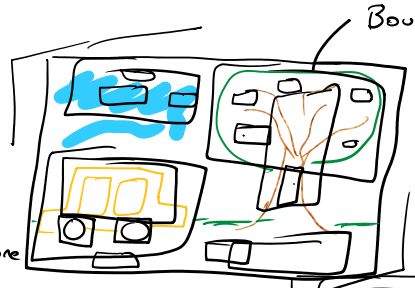
$S(R_i, R_j) = \omega_1 S_{color}(R_i, R_j) + \omega_2 S(R_i, R_j)_{texture} + \omega_3 S_{size}(R_i, R_j)$

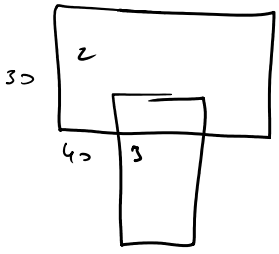


s = similarity score

R = Pixel value







Step 4 :- Bounding Box Regression :-

Proposed $\Rightarrow (x, y, w, h)$ — Pred.. (100, 120, 80, 60)

Actual $\Rightarrow$ ground truth $\rightarrow [(x_{gt}, y_{gt}, w_{gt}, h_{gt})]$

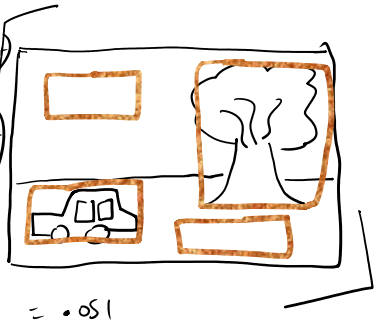
Actual ground truth → $[x_{gt}, y_{gt}, w_{gt}, h_{gt}]$

110 130 90 65

(x, y)

$\Delta x = \frac{x_{gt} - x}{w}$ $\quad \Delta w = \log\left(\frac{w_{gt}}{w}\right)$

$\Delta y = \frac{y_{gt} - y}{h}$ $\quad \Delta h = \log\left(\frac{h_{gt}}{h}\right)$



$x_{new} = x + \Delta x \cdot w$

$100 + 0.125 \times$

$y_{new} = y + \Delta y \cdot h$

$w = w \cdot e^{\left(\frac{w_{gt}}{w}\right)}$

$\Delta x = \frac{110 - 100}{80} = .125$ $\quad \Delta w = \log\left(\frac{90}{80}\right) = .051$

$\Delta y = \frac{130 - 120}{60} = .167$ $\quad \Delta h = \log\left(\frac{65}{60}\right) = .034$

RCNN

- Region Propose
- Feature extract
- Classification
- Bounding Box Regress.

Fast RCNN

- Single CNN
- Region of Interest Pooling
  - Selective Search / R
- Classification
- Bounding Box Regression

Faster RCNN

RPN

1 – CNN – 1000

2 – feature map → RPN

sliding to get region 9000

↓ objectiveness score 4500

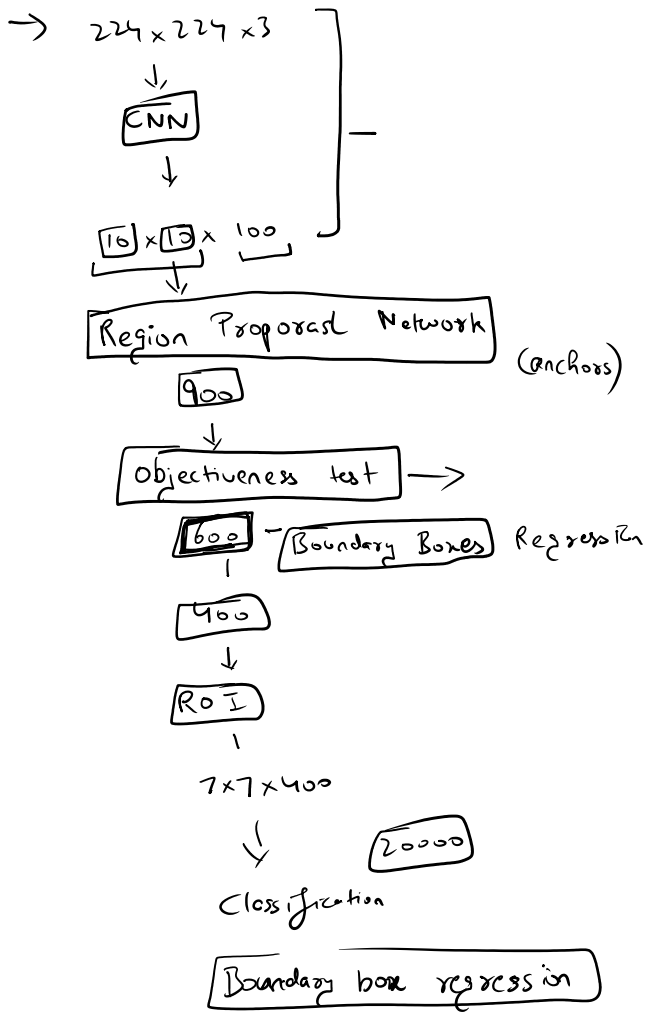
↓ Bounding box

↓ Non-maximum Suppression 400

3 RoI Pooling (7×7)

4 Classification

5) Bounding box regression



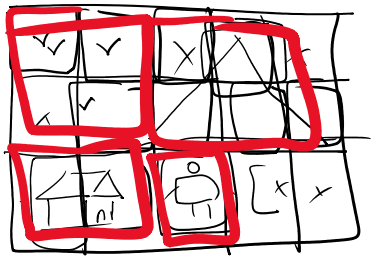
Faster

End - End trainable

Yolo :- You Only Look Once

How Yolo works

- Create grid of SxS (7x7, 13x13)
- Bounding box & Confidence Score
- Class probabilities
- NMS- Non Maximum Suppression
- Speed & Accuracy Tarde off



Limitations:

Variants:

- Yolov2: 2016
- Yolov2:
  (Yolo9000)
  - batch normalization
  - 416x416
  - 9000
- Yolov3:
- Yolov4:
  - CSP-Darknet 53
- Yolo v5
  Ultralytic
- Yolov6
  deployment
- Yolov7 – 2022
- Yolov8 – 2023

-> Object detection

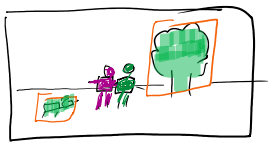
-> Image segmentation

-> Object Tracking

Masked R-CNN :- Instance Segmentation

- boundary box - pixel level mask

- Key Concepts
  - Object detection
  - Segmentation
  - Instance Segmentation



Object Detection

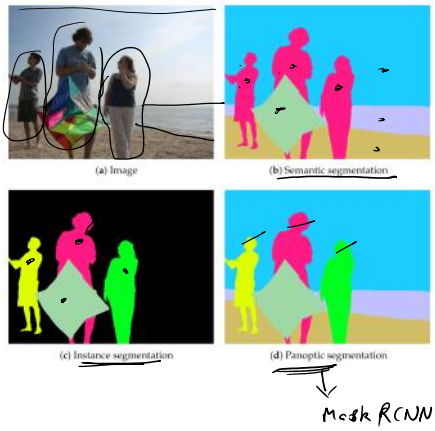
- Faster RCNN
  - CNN
  - RPN
  - RoI Pooling
  - FCNN/Regression

- Segmentation
- Instance Segmentation

- ROI Pooling
- FCNN/Regression

How Does it work?

- Feature Extraction: ResNet
- RPN - Regional Purposal Network
- ROI Align
- Classification / Bounding box Prediction
- Mask Prediction



Mask RCNN

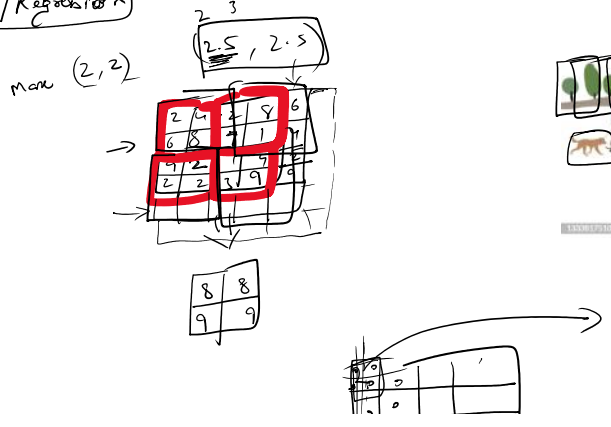
Use Cases:

- Autonomous Vehicles
- Medical Imaging

Advantages:

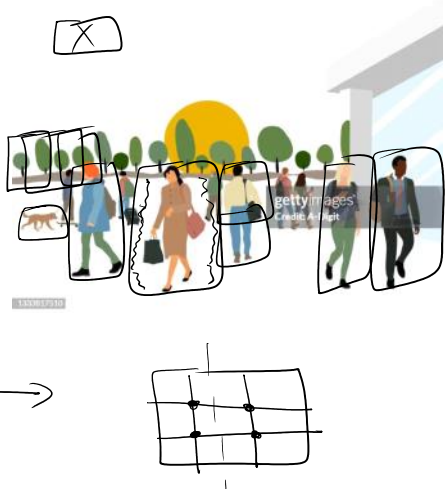
Faster RCNN

- Feature Extraction (CNN)
- RPN
- ROI Pooling - Region of Interest
- FCNN/Regression

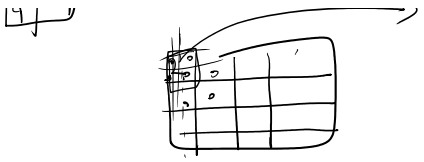


Masked R-CNN

- Feature Extraction (CNN)
- RPN
- ROI Align
- Classification / Regression / Masked Head



ROI Align

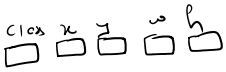


- bilinear interpolation

$$V_{(x,y)} = (1-\alpha)(1-\beta)\,V_{(x_0,y_0)} + \alpha(1-\beta)V_{(x_1,y_0)} + (1-\alpha)\beta\,V_{(x_0,y_1)} + \alpha\beta\,V_{(x_1,y_1)}$$

Fine tuning :-

Yolo | Faster RCNN



- Setup Enviroment
- Annotated dataset — major time consumer
- Prepare configuration file
- Start training
- Evaluati

Masked RCNN

- Setup Enviroment
- Annoted dataset
- Registering dataset
- Start train

Detectron2

Object Tracking :-

- Single object tracking :- (SOT)
- multiple object tracking :- (MOT)

→ Basic Components

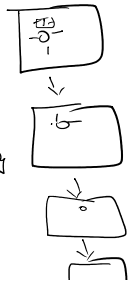
- Detection :-
- Feature Extraction
- Association
- Motion Prediction

→ Challenges of Object tracking

- Occlusion :-

:- Common techniques :-

- Correlation - based tracking :-
- optical Flow :-
- → Kalman Filter :-
- Deep Learning based tracking

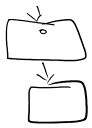


Applications :-

- Occlusion :-
- Illumination Changes :-
- Scale Variation :-
- Deformation :-
- Background Clutter :-

Applications :-

- Surveillance
- Robotics
- object tracking
- etc



- Working of OT :-
  - Detection :- CNN, Faster RCNN, Masked RCNN
  - Motion Modelling :- Linear modelling :- $x_{t+1} = x_t + V_t \cdot \Delta t$
  - Kalman Filter :-
    - predict - update - cycle

→ Prediction Step :-

- State transition :-

$$x_t = F \cdot x_{t-1} + v_{t-1}$$

- Covariance Update :-

$$P_t = F \cdot P_{t-1} \cdot F^T + Q$$

F = transition matrix

P = covariance matrix for uncertainty

Q = process noise

→ Update Step :-

- Kalman Gain :-

$$K_t = P_t \cdot H^T \cdot (H \cdot P_t \cdot H^T + R)^{-1}$$

- State Update :-

$$x_t = x_t + K_t \cdot (z_t - H \cdot x_t)$$

H = measurement matrix

R = measurement noise

K_t = Kalman Gain

z_t = actual measurement at time t

Optical Flow :-

- pixel wise motion detection :-

→ brightness constancy eqn :-

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

Particle Filter: (Sequential Monte Carlo Method)

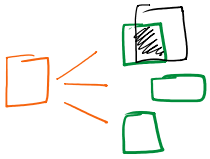
improved version of kalman filter, can work on non-linear & non-gaussian process

- Prediction
- Update
- Resampling = $w_t^i = p(z_t | x_t^i)$

Deep learning modelling methods:

- SORT (Simple Online & Realtime Tracking)

  bounding box detection + kalman filter



Steps: 1. Kalman filter predicts its next location

2. Detection in new frame & compared to predictions

   (Intersection over union) = IOU = $\frac{\text{Area of Overlap}}{\text{Area of Union}}$

3. Hungarian Algorithm minimize the total cost

- Deep SORT:

  (Appearence based features)

  - mahalanobis distance = $d(x,y) = \sqrt{(x-y)^T S^{-1} (x-y)}$

Challenge Handling:

→ Occlusion Handling:

- Appearence based feature & Particle Filter

→ Data Association:

Hangarian Algorithm & GRU (Gated Recurrent Unit)

→ 3D object tracking:

Stereo cameras, LiDAR

Additions:

- Localization:
- Motion:

- Motion Vector

- Tracking features :
  - SIFT (Scale invariant Feature Transform)
  - Harris Corner Detector
  - ORB (Oriented Fast & Rotated BRIEF)

- Byte Tracking :-