Computational Intelligence

Topics:

- Learning. Describe the difference between unsupervised and supervised learning including differences in training datasets. Explain the principle of the supervised gradient learning, how the gradient is calculated and hyperparameters of the algorithm.
- Shallow and deep neural networks (DNNs). Typical neurons, their activations and typical activation functions. Describe convolution, pooling and dense layers and their parameters. Application of DNNs for object detection in images.
- Recurrent neural networks (RNN). Explain what the recurrent connections and necessity of their specific evaluation are. Explain fully recurrent and LSTM neural architecture.
- **Reinforcement learning. Explain the principle. What is the environmental state, observation, reward, and policy? Describe the actor-critic architecture.**
- Nature-inspired optimization. Explain the principle of the genetic algorithm and the particle swarm optimization algorithm

① **Learning**: Process by which AI systems improve its performance with experience and data

Supervised	Unsupervised
- Labelled data (I/O pair)	Unlabelled data
- Predict Outputs for new I/p	finds patterns from I/p
- Types- Reg (continuous o/p)	- Types - Clustering
- Classificatn (class labels)	(grouping similar/ related data pts)
- Algorithm: Linear, LI/L2, Polynomial Regression, KNN, SVM, decision trees.	- Algo: PCA, K means clustering
- Use: House Price Forecasting, Spam detection	Use: Customer segmentatn

Supervised Gradient learning: Optimizing model's parameters to minimize the cost function and ↑ model's accuracy.

- Optimizatn by → Iteratively adjusting model's params using gradient (∇f) of a loss ()

Gradient: vector of partial derivatives of loss function w.r.t model parameters $\nabla L(\Theta) = \left[\frac{\partial L}{\partial \theta_0}, \frac{\partial L}{\partial \theta_1}, \dots \frac{\partial L}{\partial \theta_n}\right]$

points in direction of steepest ascent of function

In GD, for loss minimisatn, we move opposite dir of gradient

Gradient Descent Algo: Optimizatn Algorithm

hypothesis function → linear Reg: $h_\theta(x) = \theta_0 + \theta_1 x$

Cost function → (MSE for LR): $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x)_i - y_i)^2$

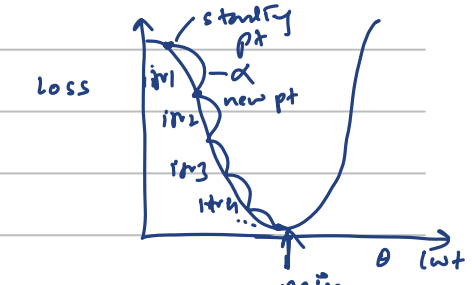
MSE loss is quadratic

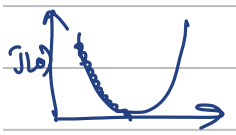
Goal: minimize $J(\theta_0, \theta_1)$ over θ_0, θ_1

Algorithm:

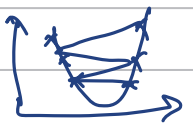
1. Initialize parameters θ
2. Calculate gradient of cost() w.r.t parameters $\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$
3. Update Gradient: $\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$

   α = learning rate / size of step
4. Repeat 2 & 3 until cost() converges (ie. change in $J(\theta)$ becomes very small b/w itr) or until preset values of iterations





→ longtime to converge
α = small



→ Overshooting the minimum might not converge.
α = big

hyperparameters: α, Epochs, Batchsize

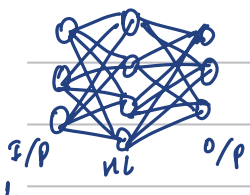
α = determines step size in GD while moving in dir of loss ()

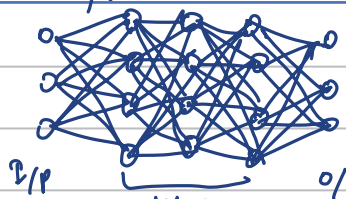
Epoch: No. of complete pass through dataset

Batch size: Training examples in 1 Iteratn

(2)

Shallow Neural N/w:

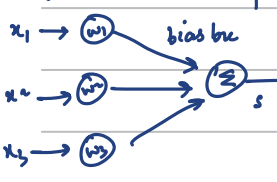




- Simple tasks
- Relation b/w I/O is straight forward
- example: Logistic Regression, SL Perceptron

- Complex tasks
- example - Image Recognition, NLP, etc
- learns hierarchical features (low level features - early, high level (complex) features in late layers)
- ex: CNN, RNN

Neuron - Building block of Neural Network



$$s = \sum_{i=1}^{n} x_i w_i + b$$

$$z = \sigma(s)$$

Activation function: Mathematical function applied to the summation of weight & I/p to introduce non linearity into the O/p

It helps in learning complex patterns & relationships

Hidden layer AF:

RELU: $f(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$ Range $(0, \infty)$

use: CNN
issue: dying neuron problem because -ve I/p get 0

Leaky ReLU: $f(z) = \begin{cases} 0.01z, & z < 0 \\ z, & z \geq 0 \end{cases}$ Range $(-0.01, \infty)$

: CNN
: solves dying neuron problem

Tanh: For zero centered activation, RNN

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

Range $(-1, 1)$ & mean comes around 0: helpful in hidden layers

Output:

Sigmoid: $f(z) = \frac{e^x}{1 + e^x}$ Range $(0, 1)$

for Binary classification
O/P → probabilities

Softmax: Used in Multiclass classification

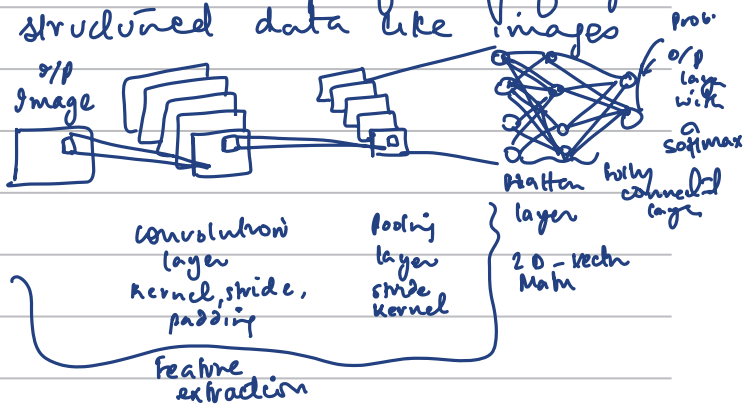
$$f(z_i) = \frac{e^{z_i}}{\sum e^{z_j}}$$

softmax applied to final layer where values are exponentiated & then normalised by dividing with sum of all e^{z_j}
∴ all positive, sum of prob = 1
Range $(0, 1)$

CNN: Why? • Fewer params
• efficient
• Reduced overfitting (↓ params)
• complexity Reduced.

→ Convolutional Neural N/w

↳ DL Model specifically for grid structured data like images



S layer: convolutional layer: extract features using Kernel/filters from I/P

Pooling layer (max, avg): Control dimensions & computational load

Flatten layer: 2D feature map - 1D feature vector

Fully connected: perform further processing & then classify

O/P layer: provides final prediction

Kernel / matrices: Small matrices (3×3, 5×5) that slide over the input matrix and perform element wise multiplication & summation to produce feature map.

Stride: No. of pixels by which filter moves across input matrix

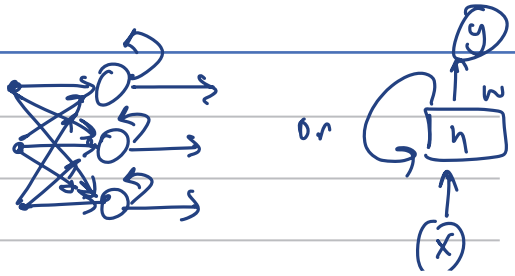
Padding: No. of pixels added to I/p matrix to prevent info loss at the corners along with to control the size of O/p feature map

$$\text{Output size} = \left(\frac{\text{Input size} - K + 2P}{S}\right) + 1$$

Pooling: Reduce dimension of I/p and preserves important features.
↳ Max & Average

(3) RNN, LSTM

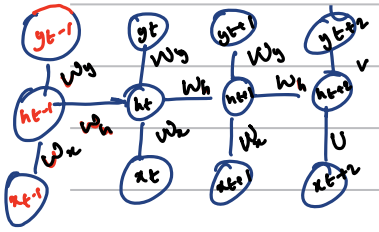
RNN:



- designed for processing sequential data where current inputs are dependent on the previous I/Ps (NLP, speech, time series)

Hidden state: core captures the info from previous time stamps. HS is updated at each time stamp as new I/P are processed allowing N/w to maintain a form of memory ∴ capturing sequential dependencies.

Formula for hidden state update:



$$H_t = \sigma(W_h h_{t-1} + W_x x_t + b)$$

tanh, Relu

Necessity: • Sequential data
• Contextual Understanding (short memory) to understand context & dependencies within the sequence.

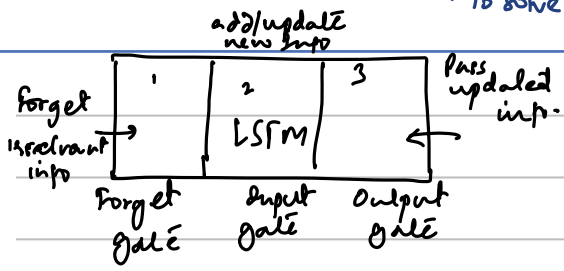
Issues with RNN: Vanishing gradient (Of small dere to BP)
Exploding gradient: (Of - large)
difficulty with long sequences → LTD.
limited memory (retaining diff over long sequences)

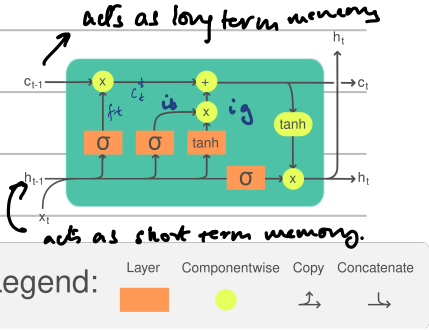
Uses: NLP, speech recognition, time series prediction, music generation.

RNN types

1:1	1: many	many: 1	many: many
I/P O/P	text generation	Sentiment analysis	Translation

LSTM: designed to handle long term dependencies in sequential data.
– To solve vanishing gradient problem in RNN





Cell State: carries information across time stamps - acts as LTM of LSTM

Hidden state: for making prediction at each step. (memory - short term)

At each time step t,

Forget gate: Use sigmoid (σ) to decide how much previous cell state info to retain

$$f_t = \sigma(x_t W_{fx} + W_{fh} h_{t-1} + b_f)$$

$$C_t = C_{t-1} \odot f_t$$

I/P gate: determines which new info is added to cell state

$$i_t = \sigma(W_{ih} h_{t-1} + x_t W_{ix} + b_i)$$

$$C_t = \tanh(W_c h_{t-1} + x_t W_{ic} + b_c)$$

O/P gate: controls what part of cell state should O/P as hidden state.

σ = determine O/P 0 to 1
tanh = scale O/P -1 to 1

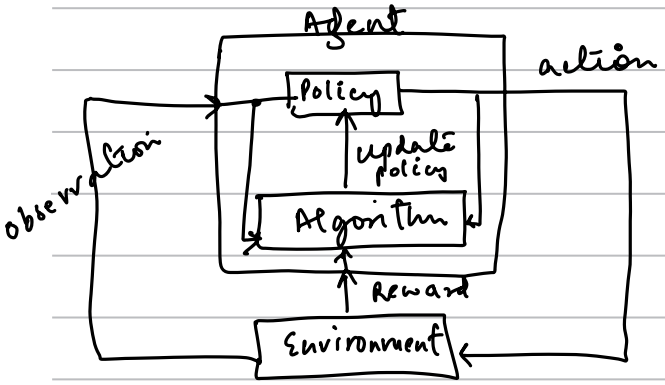
$$O_t = \sigma(W_{ox} x_t + W_{oh} h_{t-1} + b_o)$$

$$h_t = O_t \odot \tanh(C_t)$$

⊙ → insures only relevant part of cell state contributes to O/P

Reinforcement Learning:

Involves agent learning to make decisions by interacting with environment to maximize its cumulative reward w/o human interaction



Agent → decision maker [car] interacts with environment, receives observations takes actions, gets rewards.

Environment: world agent interacts in (road, traffic)

Observatn: Inf agent gets from env. sensor, image camera, GPS.

Action: Choice agent makes [turn left, brake]

Reward: how well agent is doing
+ reward – Repeat actn
– reward :– discourages actn

Policy: Rule / Strategy agent follows to map observations to actions
It is learned over time.
eg: Turn right when board ↱
Stop when red.

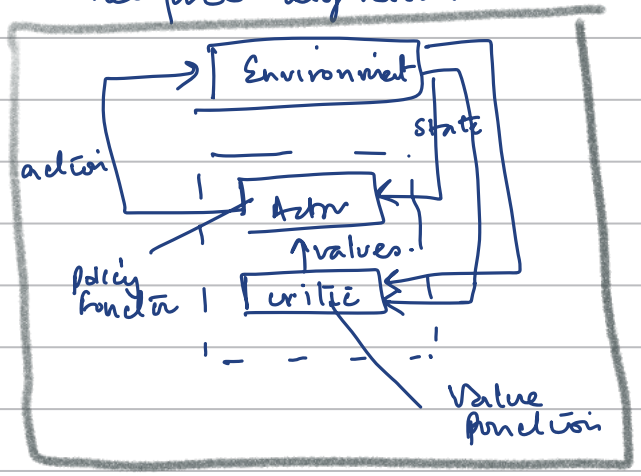
Applicatns of RL: self driving cars, Robotics, Recommendatn system, Drones

Actor critic Architecture

Type of RL algorithm combines [policy] + [value] based methods

- Value based methods: learns value() that maps each action pair to a value
  $S \rightarrow V$ works well when finite actns
  example → Q learning ✗ → continuous action
- Policy based methods: For continuous action
  directly optimal policy (rule) is found w/o value function. ✗ optimal value function diff, how good policy is.

Slow learning
Reinforce algorithm



Actor: Chooses action based on policy
– Policy based Method is used
isr neural Net $\pi(s, a, \theta)$

Critic: Evaluates actions by actor & and provide fb to improve policy
and – Value based method
neural nw $\hat{q}(s, a, w)$

When Actor & Critic participate in game, with time, they start playing better :. learn & become efficient.

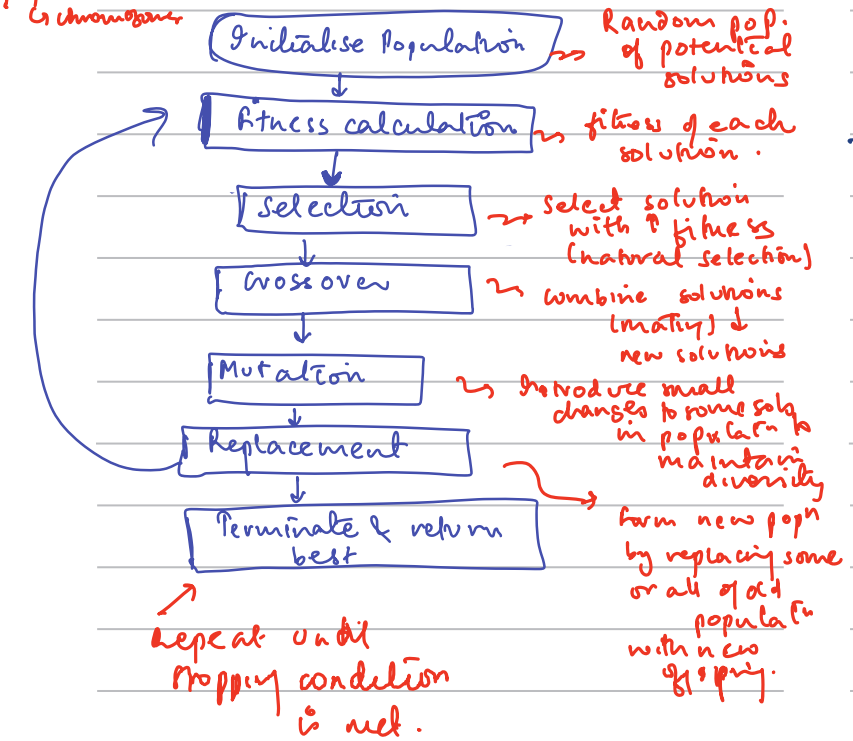
Nature Inspired algorithm

Computational method to solve complex optimisation & search problem by taking inspiration from nature (evolution, natural selection, genetics etc).

Genetic Algorithm:

principle: inspired by Darwin's theory of natural selection & genetics. Algorithm operates on population of potential solutions, evolving them over generations to find optimum solution to a given problem.

Algorithm:

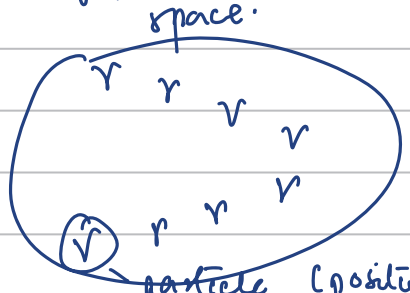


Used to solve: Travel Salesman Problem & Knapsack Problem (NP hard problems) → no efficient method exists

particle Swarm Optimisation Algorithm.

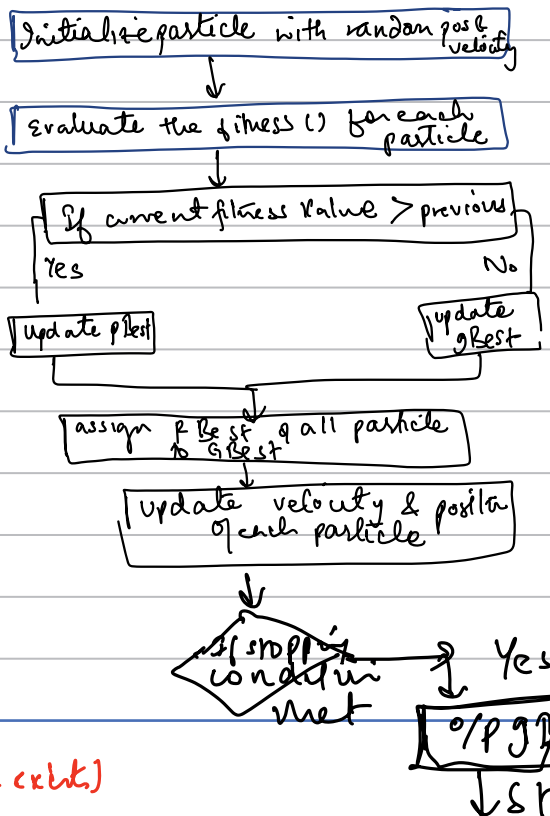
Optimization algo inspired by social behaviour of bird flocking & fish schooling.

Used: To find optimal solution to a problem by simulating swarm of particles moving through search space.



position → potential solution
velocity → direction & speed of movement
fitness value → how good solution is.

particle → keep track of pBest & gBest guiding swarm movement to optimal solution.



Velocity Update formula:

$$v_i(t+1) = w\, v_i(t) + c_1 r_1 (pBest - x_i(t)) + c_2 r_2 (gBest - x_i(t))$$

- $w\, v_i(t)$ → Diversification
- $c_1 r_1 (pBest - x_i(t))$ → personal influence
- $c_2 r_2 (gBest - x_i(t))$ → social influence

w → inertia weight constant $0 < w < 1$
controls influence of prev. velocity.

c_1 & c_2 → cognitive & social coefficient
control the tradeoff b/w exploration & exploitation

Swarm → Set of particles (solutions) (S)
Particles → a potential solution
Position : $x_i = (x_{i1}, x_{i2}, x_{i3} \dots x_{in}) \in R^n$
Velocity : $v_i = (v_{i1}, v_{i2}, v_{i3} \dots v_{in}) \in R^n$

Each particle maintains : pBest (Individual Best)
Swarm maintains it global best : gBest

Position Update:

$$pos_i(t+1) = pos_i(t) + v_i(t+1)$$