Unit-5

Challenges in Neural Network Optimization

Neural Networks face challenges in terms of data requirements, overfitting, explainability, security, and bias etc. They require large amounts of labeled data for effective training, can suffer from overfitting where they perform well on training data but fail to generalize. etc in their prediction.

(1) Training Data Requirements

require a large amount of labeled data to learn effectively. and also annotation such datasets take lot time.

costly even impractical in some cases.

(2) Overfitting

occurs when a neural network learns to perform exceptionally well on the trainins data but fail to generalize to new, unseen data.

(3) computational complexity

Neural Networks can be computationally demanding, particularly as the model size and complexity increase. it need high Performans GPU's or specialized hardware like TPU (Tensor Processin, units).

(4) Hyperparameter tunins

Networks have several hyperparamets such as learnins rate, number of layes, activation function). selectin them is molt important.

(5) Data privacy

→ Neural Networks often require large amount of data to train effectively this raises concerns about data privacy and security, especially when dealing with sensitive information.

optimizer	complexity	Computation expense
Gradient descent	low	high
Stochastic gradient descent	low	low
Stochastic gradient descent + Momentum	medium	low

Adagrad	High	medium
Adam	High	High

(6) Learning Rate

→ is too high, the model overshoots the best solution.

→ is too low, training becomes very slow.

(7) Batch size problems

→ small batches can be noisy, while large batches need more memory.

→ so we should try each to find which works best for us

Applications of large-scale Deep learning

is to use of very large Neural Networks trained on massive datasets, often using powerful computational resources like GPU or TPU.

① Computer vision

used to analyze and understand visual data like images and videos.

Applications

① Image Recognition
② object Detection
③ medical imaging
④ Video Analysis.

(2) Natural language processing

Deep learning model process and understand human language.

Applications

(1) Language Translation.
(2) chatbots and virtual Assistants
(3) Sentiment Analysis
(4) Text Summarization.

(3) Speech Recognition

a) converts spoken language into text or commands.

Applications

(1) Voice Assistants.
(2) Transcription Services
(3) Voice-controlled systems.

(4) Recommendation systems

↳ predicts user preferences and recommends products or content.

Applications

(1) E-commerce
(2) streaming services
(3) social media.

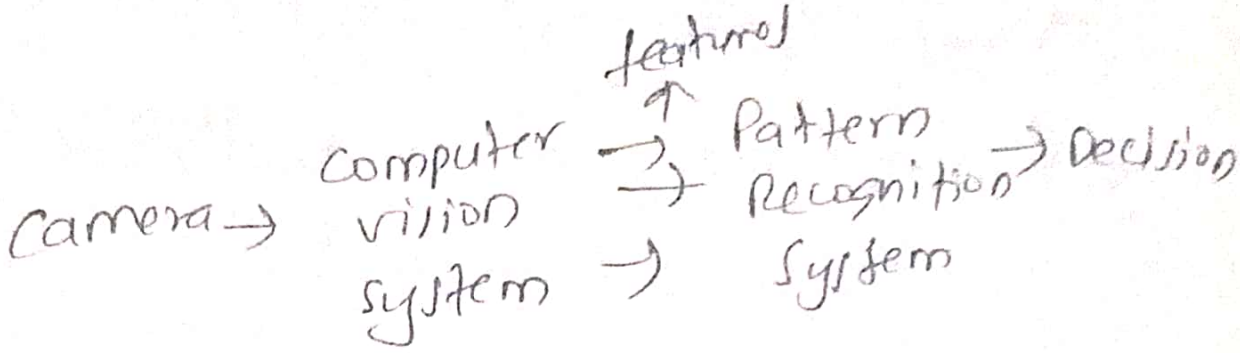
(5) Autonomous vechicles

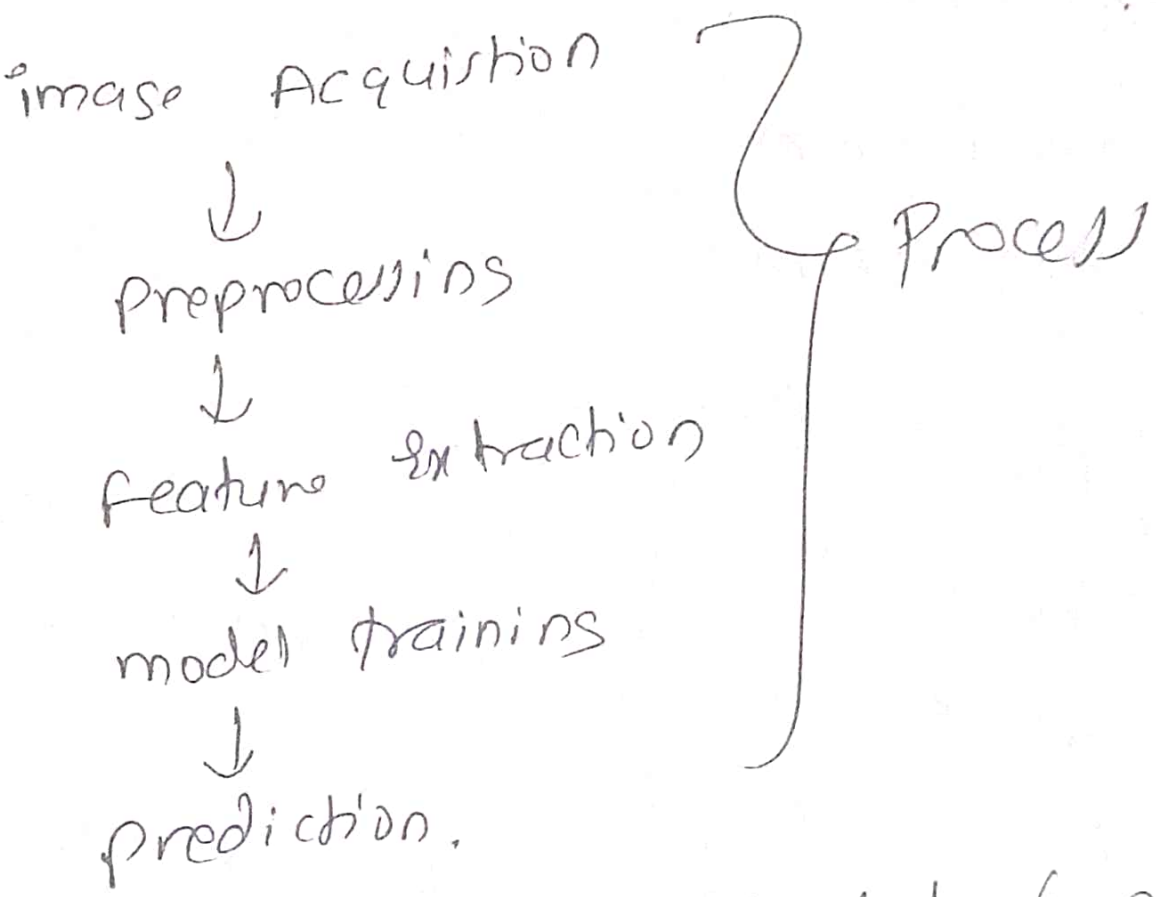
↳ Enables self-driving cars to perceive and navigate their environment.

Applications

(1) object detection
(2) path planning
(3) control systems.
(6) Healthcare
(7) Gaming
(8) finance.
(9) Robotics
(10) Generative models

Computer vision





we can get Dataset from

→ MNIST
→ IMAGENET
→ COCO (common objects in Context)

Computer vision models

1. YOLO (You only Look Once)
2. SSD (Single Shot MultiBox Detector)
3. Transformers in vision.
4. U-Net.
5. GAN.

Speech Recognition

→ It is the process of converting spoken language into text using AI and machine learning techniques. It is widely used in applications like voice assistants (Google assistant, Siri, Alexa), hands-free control.

How it works

(1) Audio input

(2) Preprocessing

(3) Acoustic modeling
↳ maps audio features

(4) Language modeling
↳ predicts the most likely word sequences.

(5) Decoding
↳ converts phonems into readable text.

Popular models

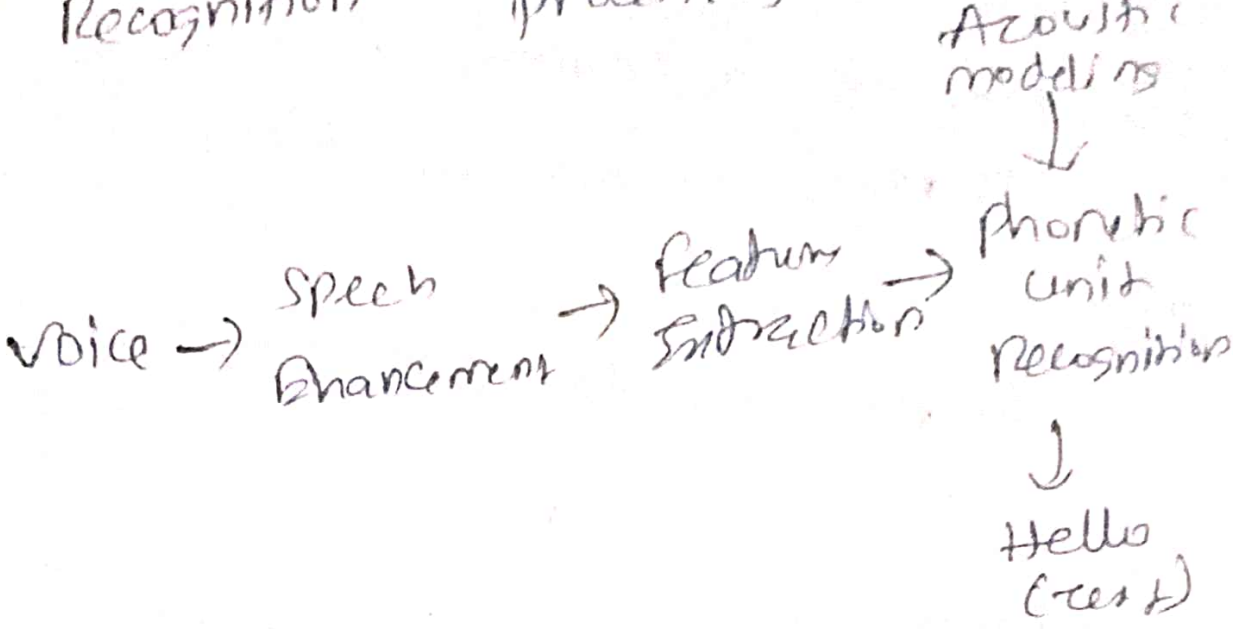
(1) DeepSpeech.

(2) Google Speech-to-Text API

(3) Kaldi (for researchers & enterprise use)

(4) Vosk (lightweight & offline support)

Automatic Speech Recognition → Natural language processing → text to speech



Natural Languase processing

it is a branch of artificial intelligence that enable computers to understand, interpret, generate, and respond to human language

key componends

1. text processing
2. syntactic Analysis. (syntax)
3. Semantic Analysis (meaning)

(4) Speech & Conversational NLP.

Popular NLP libraries

Python libraries

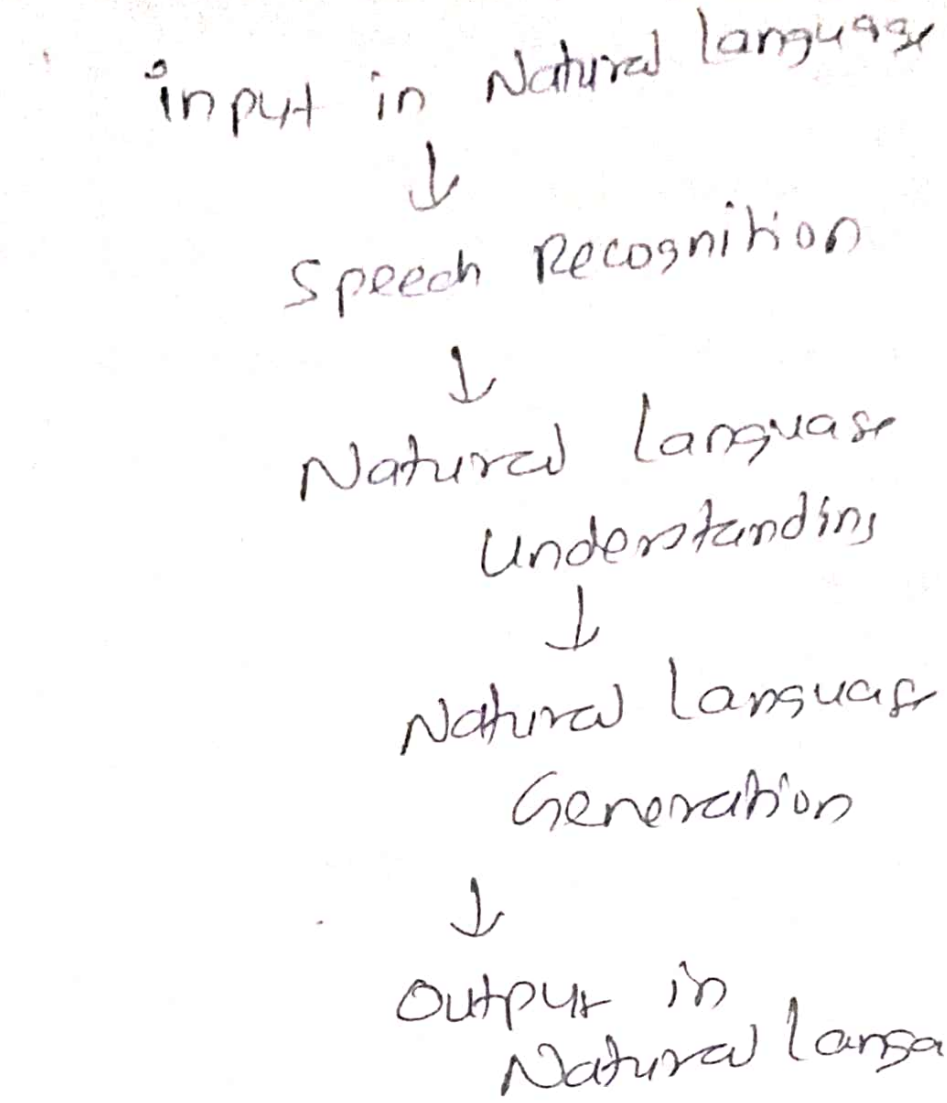
→ NLTK
→ spacy
→ Transformers
→ Gensim

Deep Learning NLP models

→ BERT (Google)
→ GPT (OpenAI)
→ T5 & BART
→ XLNet

Applications

① Chatbots & virtual assistants
② Machine translation.
③ Text Summarization.
④ Search Ranking
⑤ Spam Detection
⑥ Customer support Automation
⑦ Healthcare.



Optimization for Train Deep Models

Basic Algorithms

① Gradient Descent:-

The foundation of most optimiza-tion Algorithms. it minimize the loss function by iteratively updating the model parameter in the direction of the negative gradient.

Update rule

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$$

- $\nabla_\theta L(\theta_t)$: Gradient of the loss function w.r.t θ
- θ_t: model parameter at step t
- η: learning rate

Variants

(1) Batch Gradient Descent
- uses the entire dataset to compute the gradient.

(2) Stochastic Gradient Descent (SGD)
- uses a single data point to compute the gradient.

(3) mini-Batch Gradient Descent
- uses a small subset of the data to compute the gradient.

2. Momentum

→ Accelerates SGD by adding a fraction of the previous update vector to the current update. This helps in smoothing the optimization path and escaping local minima.

Update rule

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta L(\theta_t)$$

momentum term.

$$\theta_{t+1} = \theta_t - v_{t+1}$$

↳ velocity vector

3. Nesterov Accelerated Gradient

→ A variant of momentum that "looks ahead" by computing the gradient at the estimated future position.

update Rule

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta L(\theta_t - \gamma v_t)$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

4. Adaptive learning Rate Method

These methods adjust the learning rate for each parameter based on the history of gradients.

a. AdaGrad

↳ Adapts the learning rate for each parameter based on the sum of squared gradients.

Update rule

$$G_t = G_{t-1} + (\nabla_\theta \mathcal{L}(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta \mathcal{L}(\theta_t)$$

G_t: Accumulated Squared gradients

ϵ: small constant for stability

b. Adam (Adaptive moment estimation)

Combines the benefits of momentum and RMSProp. maintain both a moving average of gradients and a moving average of squared

gradient.

Update rule

$\beta \rightarrow$ Exponential decay rates.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad ; \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$\hat{m}_t$ ↙ first moment

$\hat{v}_t$ ↙ second moment

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

GD

uses the entire dataset to compute the gradient

updates paramter once per epoch

slower

High memory usage

Best for small datasets

SGD

uses one data point to compute the gradient.

updates parameter after every data point.

faster.

Low memory usage.

Best for large datasets.

Parameter initialization strategies

Parameter initialization is a critical step in training deep learning models. Poor initialization can lead to issue like vanishing gradients, exploding gradients or slow convergence.

① zero initialization

→ initialize all weights to zero.

→ Rarely used in practice.

② Random initialization.

→ initialize weights with small random values.

→ It allows neurons to learn different features.

→ if the weights are too large or too small it can lead to exploding or vanishing gradients.

3. Glorot initialization

→ Scales the random initialization based on the number of input and output neurons to maintain consistent variance across layer.

4. He initialization

A varient of Glorat initialization designed for ReLU activation functions. it scales the weights by the number of input neurons only.

5. lecun initalization

Similar to Glorat initialization but scaled specifically for tanh activation functions:

6. Orthogonal initialization

initializes weights as an orthogonal matrix, ensuring that the input and output have the same norm.

commonly used in RNN and LSTM.

7. Pre trained. initialization (Transfer learning)

* Use weights from a pretrained model as the starting point.

8. Biases initialization

Biases are typically initialized to zero or a small constant value (e.g 0.01).

Algorithms with Adaptive learning rates

gt is used to dynamically adjust the learning rate during the training to improve convergence, prevent diversance and enhance overall model performance.

① Adagrad (Adaptive Gradient Algorithm)

→ Adjust learning rate individually for each parameter based on past gradients.

Update Rule

$$g_t = g_{t-1} + \nabla \theta_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{g_t + \epsilon}} \nabla \theta_t$$

2. RMSprop (Root Mean Square propagation)

→ improves Adagrad by introducing a moving average of squared gradients.

→ well-suited for non-stationary problems like deep networks.

Update rule

$$g_t = \gamma g_{t-1} + (1-\gamma) \nabla \theta_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{g_t + \epsilon}} \nabla \theta_t$$

typical choice $\gamma = 0.9$

3. Adadelta

→ Extension of Adagrad, addressing its learning rate decay problem.

→ no need for manual learning rate tuning.

4. Adam (Adaptive moment Estimation)

→ combines the benefits of Adagrad and RMS prop.

→ maintains both first-moment (mean) and second-moment (uncentered variance) estimates of gradients.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla\theta_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla\theta_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

$\beta_1 = 0.9$, $\beta_2 = 0.999$ Default values

5. AdamW

→ A variation of Adam with decoupled weight decay.

→ Helps improve generalization in large models.

	Best for	Weakness
Adagrad	Sparse data (NLP, recommender systems)	Learning rate keeps decreasing
RMSprop	RNNs, non-stationary problems.	Needs tuning of decay factor.
Adadelta	Adaptive learning rate without manual tuning.	Can still be sensitive to hyperparameters
Adam	most deep learning tasks (CNNs, RNNs, transform).	Can gen-eralize poorly on some tasks.
AdamW	Large-scale models. regularization	Requires careful weight decay tuning.

Second-order methods

It leverage curvature information (Hessian matrix) of the loss function to improve optimization.

Unlike first-order methods, which relay only on gradients, second-order methods use the Hessian to guide updates.

offer

(1) Better direction

(2) Better step-size

~~Network~~ Newton's method

it is mostly used second-order method. it uses a second-order Taylor expansion to approximate the loss function $J(\theta)$ near a point θ_0

Newton update Rule

Solving for the critical point of the quadratic approximation gives the ~~Netw~~ newton update.

$$\theta^* = \theta_0 - H^{-1} \nabla_\theta J(\theta_0)$$

for a quadratic function with positive definite H, ~~Netw~~ Newtons method jumps directly to the minimum. For non-quadratic functions, the update is iterated.

Trainins Algorithm

1. Compute the gradient.
2. Compute the Hessian.
3. update parameters.

3. conjugate Gradients

is an iterative method to avoid computing H^{-1} explicitly. it addresses the inefficiency of steepest descent, which often follows a zig-zag path.

Alg

1. initialize α_0 and d_0
2. For each iteration t:
   → compute step size ϵ_t via line search.
   → update parameters.
   → compute new gradient.
   → update conjugate direction.

4. BFGS and L-BFGS

BFGS is a quasi-Newton method that approximates the inverse Hessian H^{-1} iteratively without explict computation.

→ Limited-memory BFGS reduces memory wase by storing only a few vectors instead of the full matrix.

→ Suitable for laise scale optimization problems.

use)

Better Direction

Better Step-size

Challenges

computational cost

scalability.

Optimization strategies and Meta Algorithms

① Gradient-Based optimization.

② Adaptive Learning Rate methods.

③ Second-order methods.

→ Newton method

→ quasi-Newton method (L-BFGS)

④ Learning Rate scheduling.

Meta Algorithms

① Ensemble methods

→ Bagging → Trains multiple models independently on different subsets of the data and averages their predictions.

Boosting → Sequentially trains models with each new model focusing on the errors of the previous one

Stacking → combines predictions from multiple models using a meta-model.

Transfer learning

Reuses pre-trained models and fine tunes them for a new task. Reduces training time and data requirements.

Neural Architecture Search (NAS)

Automates the design of neural network architectures using reinforcement learning, evolutionary algs, or gradient based methods.

Curriculum learning

↳ Trains the model on easier examples first and gradually introduces more complex examples.

self-supervised learning

↳ uses unlabeled data to pre-train models by defining pretext tasks

meta-learning (learning to learn)

Trains models to quickly adapt to new tasks with minimal data.