CBCS SCHEME

USN | | | | | | | | | | |

21CS743

Seventh Semester B.E./B.Tech. Degree Examination, Dec.2024/Jan.2025
Deep Learning

Time: 3 hrs. Max. Marks: 100

Note: ***Answer any FIVE full questions, choosing ONE full question from each module.***

Module-1

1 a. Explain the concept of tasks(T), Performance (P) and Experience (E). Describe the following with respect to tasks performance and experience.
i) Checker learning problem
ii) Handwriting recognition learning problem **(12 Marks)**

b. Explain the concept of supervised and unsupervised learning with example. **(08 Marks)**

OR

2 a. Explain the historical trends in deep learning. **(10 Marks)**

b. Define supervised and unsupervised learning algorithm. Describe KNN and K means algorithm. **(10 Marks)**

Module-2

3 a. Explain about gradient based learning. **(10 Marks)**

b. Explain the concept of Back propagation and how it helps in a Neural network. **(10 Marks)**

OR

4 a. Define Regularization. Describe L^1 and L^2 regularization. **(10 Marks)**

b. Given $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $C = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $W = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ and $b = 0$ draw feed forward network and evaluate XOR function. **(10 Marks)**

Module-3

5 a. What are the challenges in neural network optimization? **(10 Marks)**

b. Explain the following algorithms
i) RMSProp
ii) RMSProp with momentum. **(10 Marks)**

OR

6 a. Explain stochastic gradient descent and momentum algorithms **(10 Marks)**

b. Give the list of adaptive learning rates algorithms. Write the Ada Grad algorithm. **(10 Marks)**

Module-4

7 a. Explain the following with suitable diagram.
i) Sparse interactions ii) Parameter sharing. **(10 Marks)**

b. Explain briefly variant of the CNN models. **(10 Marks)**

OR

8 a. Differentiate locally connected layers, tiled convolution and standard convolution with suitable example and diagram. **(10 Marks)**

b. Explain the different layers in CNN models and its function with a neat diagram. **(10 Marks)**

Module-5

9 a. Discuss about Bidirectional Recurrent neural networks. **(10 Marks)**

b. What is speech recognition? Explain the different types of speech recognition systems. **(10 Marks)**

OR

10 a. Explain Long Short-Team Memory (LSTM) working principles along with all the equations. **(10 Marks)**

b. What is Natural language processing? Explain different steps involved in NLP. **(10 Marks)**

* * * * *

Seventh Semester B.E./B.Tech. Degree Examination, Dec.2024/Jan.2025

21CS743 - Deep Learning

Note: Answer any FIVE full questions, choosing ONE full question from each module.

Module -1

1. **a. Explain the concept of tasks(T), Performance (P) and Experience (E). Describe the following with respect to tasks performance and experience. i)Checker learning problem ii) Handwriting recognition learning problem (12 Marks)**

   Machine learning **tasks** are usually described in terms of how the machine learning system should process an example. An example is a collection of features that have been quantitatively measured from some object or event that we want the machine learning system to process. We typically represent an example as a vector $x \in Rn$ where each entry xi of the vector is another feature.

   We often refer to the error rate as the expected 0-1 loss. The 0-1 loss on a particular example is 0 if it is correctly classified and 1 if it is not. For tasks such as density estimation, it does not make sense to measure accuracy, error rate, or any other kind of 0-1 loss. Instead, we must use a different **performance** metric that gives the model a continuous-valued score for each example. The most common approach is to report the average log-probability the model assigns to some examples.

   Experience (E): This is the information used to learn or improve the task. It can include historical data, labeled examples, or interactive feedback. The more diverse and extensive the experience, the better the model can generalize.

   Checker Learning Problem with Respect to T, P, and E:
   The Checker Learning Problem is a classic example from game AI, where a program learns to play the game of checkers:

   1. Task (T): The task is to play checkers at an expert level, aiming to win games against opponents.
   2. Performance (P**):** Performance is measured by the percentage of games won against a range of opponents, including other AI and human players. Success is reflected in the win rate and the quality of moves.

3. Experience (E): Experience comes from playing games of checkers, either by playing against itself (self-play) or by learning from historical games. The AI can analyze past moves, outcomes, and strategies to refine its decision-making.

The Handwriting Recognition Learning Problem with Respect to T, P, and E: (focuses on enabling a machine to recognize handwritten text accurately)

1. Task (T):
   The task is to recognize and interpret handwritten characters or words from images or digital input. This involves classifying characters based on their shape and style.
2. Performance (P):
   Performance is evaluated using accuracy or error rate—the percentage of correctly recognized characters or words out of the total input. Higher accuracy indicates better performance.
3. Experience (E):
   Experience comes from a labeled dataset of handwritten samples. For example, in the MNIST dataset, images of handwritten digits (0-9) are labeled with their respective values. The more diverse the dataset, the better the model generalizes to different handwriting styles**.**

b) Explain the concept of supervised and unsupervised learning with example. (08 Marks)

Supervised learning algorithms are, roughly speaking, learning algorithms that learn to associate some input with some output, given a training set of examples of inputs x and outputs y.

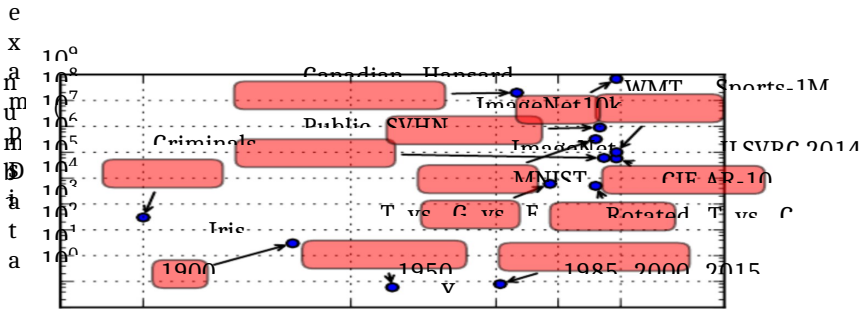
Example : k-nearest neighbors is not restricted to a fixed number of parameters.
We usually think of the k-nearest neighbors algorithm as not having any parameters, but rather implementing a simple function of the training data. In fact, there is not even really a training stage or learning process. Instead, at test time, when we want to produce an output y for a new test input x, we find the k-nearest neighbors to x in the training data X. We then return the average of the corresponding y values in the training set. This works for essentially any kind of supervised learning where we can define an average over y values.

Unsupervised Learning Algorithms : Informally, unsupervised learning refers to most attempts to extract information from a distribution that do not require human labor to annotate examples.

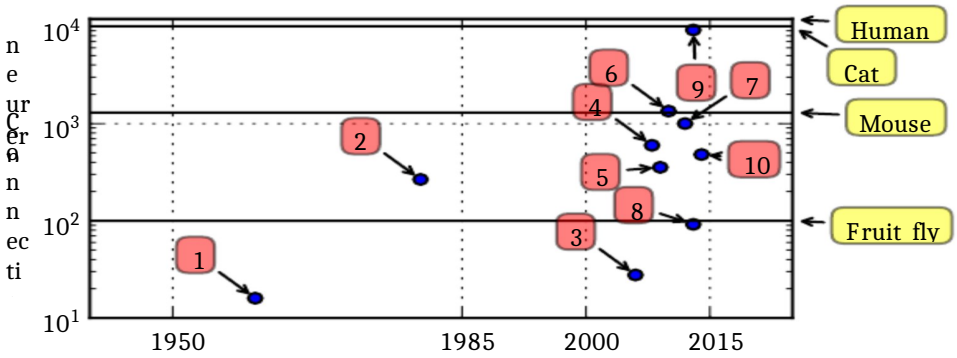
Example ; **Principal Component Analysis (PCA)** is a **dimensionality reduction** technique used in machine learning and statistics. It simplifies large datasets by transforming them into a new coordinate system with fewer dimensions, while retaining as much variability (information) as possible.

2 a. Explain the historical trends in deep learning. (10 Marks)

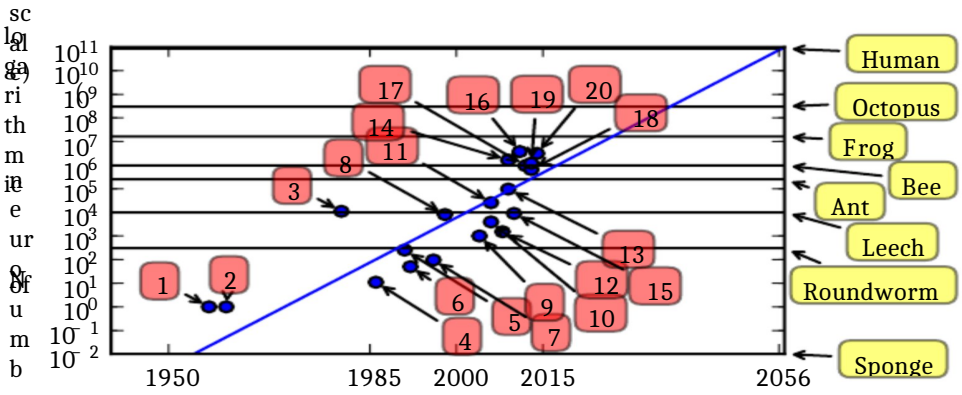
1. Deep learning has had a long and rich history but has gone by many names reflecting different philosophical viewpoints and has waxed and waned in popularity.
2.Deep learning has become more useful as the amount of available training data has increased.



3. Deep learning models have grown in size over time as computer infrastructure (both hardware and software) for deep learning has improved.



4.Deep learning has solved increasingly complicated applications with increasing accuracy over time.



b. Define supervised and unsupervised learning algorithm. Describe KNN and K means algorithm. (10 Marks)

Supervised learning algorithms are, roughly speaking, learning algorithms that learn to associate some input with some output, given a training set of examples of inputs x and outputs y.

Unsupervised Learning Algorithms : Informally, unsupervised learning refers to most attempts to extract information from a distribution that do not require human labor to annotate examples.

k-nearest neighbors is not restricted to a fixed number of parameters.
We usually think of the k-nearest neighbors algorithm as not having any parameters, but rather implementing a simple function of the training data. In fact, there is not even really a training stage or learning process. Instead, at test time, when we want to produce an output y for a new test input x, we find the k-nearest neighbors to x in the training data X. We then return the average of the corresponding y values in the training set. This works for essentially any kind of supervised learning where we can define an average over y values
When there is infinite training data, all test points x will have infinitely many training set neighbors at distance zero. If we allow the algorithm to use all of these neighbors to vote, rather than randomly choosing one of them, the procedure converges to the Bayes error rate. The high capacity of k-nearest neighbors allows it to obtain high accuracy given a large training set. However, it does so at high computational cost, and it may generalize very badly given a small, finite training set. One weakness of k-nearest neighbors is that it cannot learn that one feature is more discriminative than another.

The **k-means** clustering algorithm divides the training set into different clusters of examples that are near each other. We can thus think of the algorithm as providing a k-dimensional one-hot code vector h representing an input x. If x belongs to cluster i, then h i = 1 and all other entries of the representation h are zero. The k-means algorithm works by initializing k different centroids $\{\mu(1), \ldots, \mu(k)\}$ to different values, then alternating between two different steps until convergence. In one step, each training example is assigned to clusteri, where i is the index of the nearest centroid $\mu(i)$. In the other step, each centroid $\mu(i)$ is updated tothe mean of all training examples x(j) assigned to cluster i.

Module -2

3 a. Explain about gradient based learning. (10 Marks)

For feedforward neural networks, it is important to initialize all weights to small random values. The biases may be initialized to zero or to small positive values. The iterative gradient-based optimization algorithms used to train feedforward networks and almost all other deep models.

We can of course, train models such as linear regression and support vector machines with gradient descent too, and in fact this is common when the training set is extremely large. From this point of view, training a neural network is not much different from training any other model. Computing the gradient is slightly more complicated for a neural network, but can still be done efficiently and exactly.

The cost functions for neural networks are more or less the same as those for other parametric models, such as linear models. In most cases, our parametric model defines a distribution $p(y \mid x;\theta)$ and we simply use the principle of maximum likelihood. This means we use the cross-entropy between the training data and the model's predictions as the cost function. Sometimes, we take a simpler approach, where rather than predicting a complete probability distribution over y, we merely predict some statistics of y conditioned on x. Specialized loss functions allow us to train a predictor of these estimates. The total cost function used to train a neural network will often combine one of the primary cost functions described here with a regularization term. The weight decay approach used for linear models is also directly applicable to deep neural networks and is among the most popular regularization strategies.

b. Explain the concept of Back propagation and how it helps in a Neural network. (10 Marks)

The back-propagation algorithm , often simply called backprop, allows the information from the cost to then flow backwards through the network, in order to compute the gradient. The term back-propagation is often misunderstood as meaning the whole learning algorithm for multi-layer neural networks. Actually, back-propagation refers only to the method for computing the gradient, while another algorithm, such as stochastic gradient descent, is used to perform learning using this gradient. Furthermore, back-propagation is often misunderstood as being specific to multi-layer neural networks, but in principle it can compute derivatives of any function(for some functions, the correct response is to report that the derivative of the function is undefined). Specifically, we will describe how to compute the gradient $\nabla x f(x, y)$ for an arbitrary function f , wherex is a set of variables whose derivatives are desired, and y is an additional set of variables that are inputs to the function but whose derivatives are not required. In learning algorithms, the gradient we most often require is the gradient of the cost function with respect to the parameters, $\nabla\theta\ J(\theta)$. Many machine learning tasks involve computing other derivatives, either
as part of the learning process, or to analyze the learned model.

The back-propagation algorithm can be applied to these tasks as well, and is not restricted to computing the gradient of the cost function with respect to the parameters. The idea of computing derivatives by propagating information through a network is very general, and can be used to compute values such as the Jacobian of a function f with multiple outputs. We restrict our description here to the most commonly used case where f has a single output.

Steps :

1.Calculate Error**:** Compute the loss (error) between predicted and actual values.
2.Compute Gradients: Using the chain rule of calculus, calculate the derivative of the loss function with respect to each weight.
3.Update Weights**:** Adjust weights and biases using Gradient Descent:

$w=w-\alpha \cdot \partial J / \partial w$

Where

w = Weight

α = Learning rate

$\partial J / \partial w$ - Gradient of the loss function with respect to the weight

4.Repeat the process iteratively until the error is minimized.

4 a. Define Regularization. Describe Land L2 regularization. (10 Marks)

Regularization is a technique used in machine learning to prevent overfitting by adding a penalty term to the loss function. This encourages the model to maintain simplicity, improving generalization on unseen data.

Regularization helps by:

1. Reducing model complexity.
2. Improving model performance on unseen data.
3. Preventing overfitting by discouraging large coefficients.
4.

L1 Regularization (Lasso Regression): as the sum of absolute values of the individual parameters.

Cost Function: $J(\theta)=\text{Loss Function}+\lambda \sum |\theta j|$

In comparison to L2 regularization, L1 regularization results in a solution that is more sparse. The sparsity property induced by L1 regularization has been used extensively as a feature selection mechanism. Feature selection simplifies a machine learning problem by choosing which subset of the available features should be used. The L1 penalty causes a subset of the weights to become zero, suggesting that the corresponding features may safely be discarded.

L2 Parameter Regularization

L2 regularization is also known as ridge regression or Tikhonov regularization.

L2 Regularization, also known as Ridge Regularization in the context of linear regression, is a regularization technique used to prevent overfitting by adding a penalty term to the cost function. This penalty discourages large weights, leading to a simpler and more generalized model.

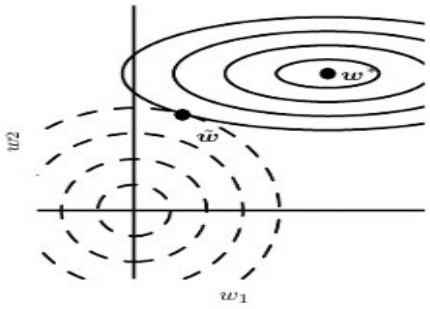


Figure 7.1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal $\boldsymbol{w}$. The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L^2 regularizer. At the point $\tilde{\boldsymbol{w}}$, these competing objectives reach an equilibrium. In the first dimension, the eigenvalue of the Hessian of J is small. The objective function does not increase much when moving horizontally away from $\boldsymbol{w}^*$. Because the objective function does not express a strong preference along this direction, the regularizer has a strong effect on this axis. The regularizer pulls w_1 close to zero. In the second dimension, the objective function is very sensitive to movements away from $\boldsymbol{w}^*$. The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w_2 relatively little.

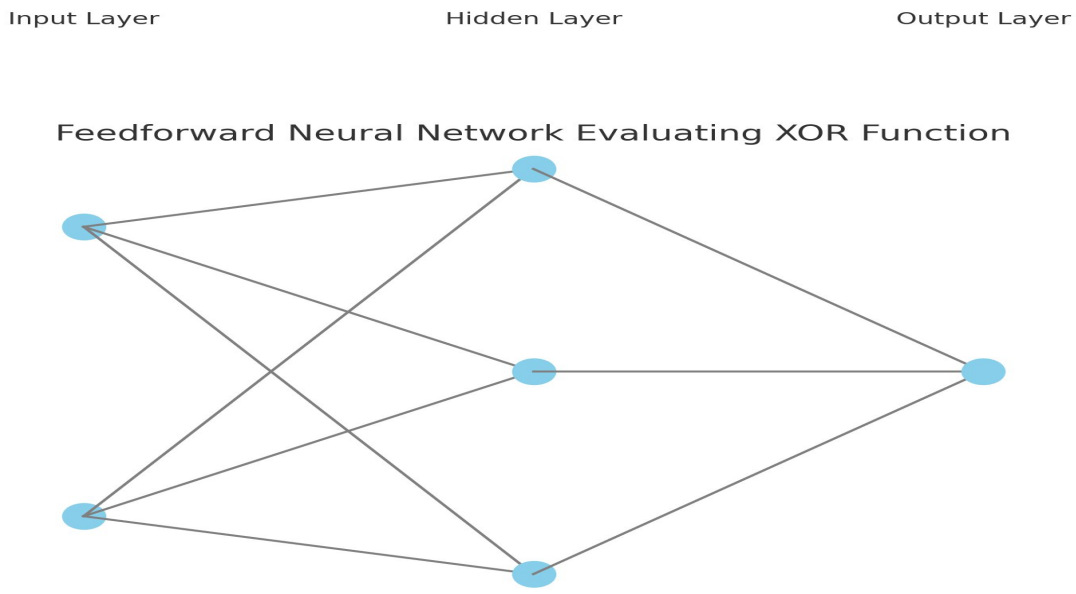
During **Gradient Descent**, L2 Regularization modifies the weight update rule:
$\theta j = \theta j - \alpha(\partial J/\partial \theta j + \lambda/m\theta j)$

$\lambda/m\theta j$= Regularization term
α = Learning rate

b. Given $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $C = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $W = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ and b = 0 draw feed forward network and evaluate XOR function. **(10 Marks)**

The visualization of the feedforward neural network using the given weights.
The predicted outputs for the XOR function are:

- (0 XOR 0) = 0
- (0 XOR 1) = 0
- (1 XOR 0) = 0
- (1 XOR 1) = 0



These results indicate that the network isn't correctly solving the XOR function. This setup might be unsuitable due to the choice of weights or activation function. XOR is a nonlinear problem, requiring a hidden layer with nonlinear activation functions.

Module-3

5 a. What are the challenges in neural network optimization? (10 Marks)

Ill-Conditioning

The ill-conditioning problem is generally believed to be present in neuralnetwork training problems. Ill-conditioning can manifest by causing SGD to get "stuck" in the sense that even very small steps increase the cost function.

$$\frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g} - \epsilon \boldsymbol{g}^\top \boldsymbol{g}$$

to the cost. Ill-conditioning of the gradient becomes a problem when $\frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}$ exceeds $\epsilon \boldsymbol{g}^\top \boldsymbol{g}$. To determine whether ill-conditioning is detrimental to a neural network training task, one can monitor the squared gradient norm $\boldsymbol{g}^\top \boldsymbol{g}$ and the $\boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}$ term. In many cases, the gradient norm does not shrink significantly throughout learning, but the $\boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}$ term grows by more than an order of magnitude. The result is that learning becomes very slow despite the presence of a strong gradient because the learning rate must be shrunk to compensate for even stronger

Local Minima

In the context of neural networks, the **local minima problem** occurs when the optimization algorithm (usually gradient descent) finds a point where the error is low, but not the lowest possible. This point is called a "local minimum," as opposed to the "global minimum," where the error is the lowest across the entire function.

To Avoid Local Minima:

1. Use activation functions like ReLU or Sigmoid instead of step functions.
2. Apply a multilayer neural network (at least one hidden layer) for non-linear problems.
3. Use optimization techniques like momentum, Adam optimizer, or stochastic gradient descent (SGD).
4. Randomly initialize weights multiple times (weight initialization strategies).

Plateaus, Saddle Points and Other Flat Regions

A plateau is a region in the loss function landscape where the gradient is nearly zero or very small over a large area. When optimization algorithms (e.g., gradient descent) encounter a plateau, updates become tiny or nonexistent.The network can get "stuck" in these regions for a long time, leading to slow or stalled training.

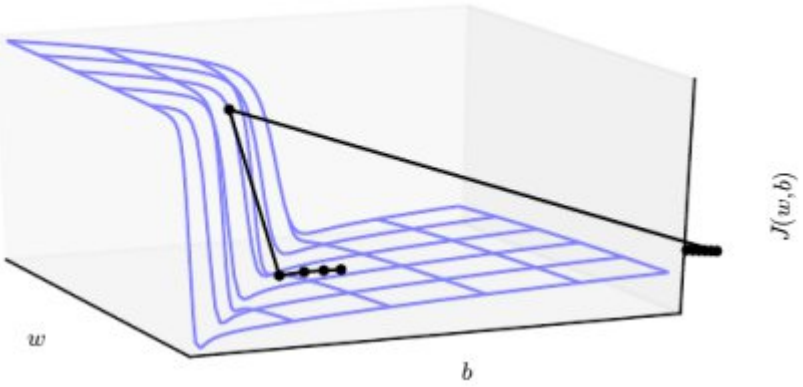
A saddle point is a point in the loss landscape where the gradient is zero, but the point is neither a minimum nor a maximum. In a saddle point, the loss function decreases in some directions but increases in others. High-dimensional functions often have numerous saddle points, especially in deep networks.

Flat regions refer to areas where the gradient is consistently close to zero, causing extremely slow convergence. These regions are common when using activation functions that saturate, such as Sigmoid or Tanh.

The network struggles to learn since gradient updates become negligible and Training time increases significantly.

Exploding Gradients

Neural networks with many layers often have extremely steep regions resembling cliffs. These result from the multiplication of several large weights together. On the face of an extremely steep cliff structure, the gradient update step can move the parameters extremely far, usually jumping off the cliff structure altogether. The cliff can be dangerous whether we approach it from above or from below, but fortunately its most serious consequences can be avoided using the gradient clipping The basic idea is to recall that the gradient does not specify the optimal step size, but only the optimal direction within an infinitesimal region.



Long-Term Dependencies

For example, suppose that a computational graph contains a path that consists of repeatedly multiplying by a matrix $\boldsymbol{W}$. After t steps, this is equivalent to multiplying by $\boldsymbol{W}^t$. Suppose that $\boldsymbol{W}$ has an eigendecomposition $\boldsymbol{W} = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}$. In this simple case, it is straightforward to see that

$$\boldsymbol{W}^t = \left(\boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^t = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1}. \tag{8.11}$$

Any eigenvalues λ_i that are not near an absolute value of 1 will either explode if they are greater than 1 in magnitude or vanish if they are less than 1 in magnitude. The **vanishing and exploding gradient problem** refers to the fact that gradients through such a graph are also scaled according to $\text{diag}(\boldsymbol{\lambda})^t$. Vanishing gradients make it difficult to know which direction the parameters should move to improve the cost function, while exploding gradients can make learning unstable. The cliff structures described earlier that motivate gradient clipping are an example of the exploding gradient phenomenon.

The repeated multiplication by $\boldsymbol{W}$ at each time step described here is very similar to the **power method** algorithm used to find the largest eigenvalue of a matrix $\boldsymbol{W}$ and the corresponding eigenvector. From this point of view it is not surprising that $\boldsymbol{x}^\top\boldsymbol{W}^t$ will eventually discard all components of $\boldsymbol{x}$ that are orthogonal to the principal eigenvector of $\boldsymbol{W}$.

Recurrent networks use the same matrix $\boldsymbol{W}$ at each time step, but feedforward networks do not, so even very deep feedforward networks can largely avoid the vanishing and exploding gradient problem (Sussillo, 2014).

Inexact Gradients

The inexact gradients problem arises when the gradients calculated during the optimization process are not accurate, leading to unreliable updates to the network's parameters. This can cause slow convergence, divergence, or getting stuck in local minima**.** Due to Non-Smooth or Discontinuous Functions , Numerical Precision Errors , Approximations in Optimization Algorithms , Vanishing or Exploding Gradients , Poor Weight Initialization.

Theoretical Limits of Optimization

Some theoretical results apply only to the case where the units of a neural network output discrete values. However, most neural network units output smoothly increasing values that make optimization via local search feasible. Some theoretical results show that there exist problem classes that are intractable, but it can be difficult to tell whether a particular problem falls into that class. Other results show that finding a solution for a network of a given size is intractable, but in practice we can find a solution easily by using a larger network for which many more parameter settings correspond to an acceptable solution. Moreover, in the context of neural network training, we usually do not care about finding the exact minimum of a function, but seek only to reduce its value sufficiently to obtain good generalization error. Theoretical analysis of whether an optimization algorithm can accomplish this goal is extremely difficult. Developing more realistic bounds on the performance of optimization algorithms therefore remains an important goal for machine learning research.

b. **Explain the following algorithms i) RMSProp ii) RMSProp with momentum.(10 Marks)**

i) **RMSProp :** RMSProp is an optimization algorithm that adjusts the learning rate for each parameter based on the magnitude of recent gradients. It keeps a moving average of the squared gradients to normalize the gradient descent step, preventing oscillations and speeding up convergence**.**

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

ii) RMSProp with momentum

RMSProp with momentum incorporates the concept of momentum into the RMSProp algorithm. It leverages both the running average of squared gradients (RMSProp) and the running average of past updates (momentum) to smooth out the optimization process.

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

6a. Explain stochastic gradient descent and momentum algorithms (10 Marks)

Stochastic Gradient Descent :

It is possible to obtain an unbiased estimate of the gradient by taking the average gradient on a minibatch of m. A crucial parameter for the SGD algorithm is the learning rate. Previously, we have described SGD as using a fixed learning rate In practice, it is necessary to gradually decrease the learning rate over time. The learning rate may be chosen by trial and error, but it is usually best to choose it by monitoring learning curves that plot the objective function as a function of time.

The most important property of SGD and related minibatch or online gradient-based optimization is that computation time per update does not grow with the number of training examples. This allows convergence even when the number of training examples becomes very large. For a large enough dataset, SGD may converge to within some fixed tolerance of its final test set error before it has processed the entire training set.

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

Formally, the momentum algorithm introduces a variable v that plays the role of velocity—it is the direction and speed at which the parameters move through parameter space. The velocity is set to an exponentially decaying average of the negative gradient.

Previously, the size of the step was simply the norm of the gradient multiplied by the learning rate. Now, the size of the step depends on how large and how aligned a sequence of gradients are. The step size is largest when many successive gradients point in exactly the same direction. If the momentum algorithm always observes gradient g, then it will accelerate in the direction of −g, until reaching a terminal velocity where the size of each step is

$$\frac{\epsilon||\boldsymbol{g}||}{1-\alpha}.$$

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

b. Give the list of adaptive learning rates algorithms. Write the Ada Grad algorithm. (10 Marks)

Adaptive learning rate algorithms are a class of optimization methods that adjust the learning rate dynamically during training to improve convergence, stability, and performance
Adagrad, Adadelta , RMSProp , Adam , AdamW (Adam with Weight Decay)

Ada Grad algorithm

The Adagrad algorithm, individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values

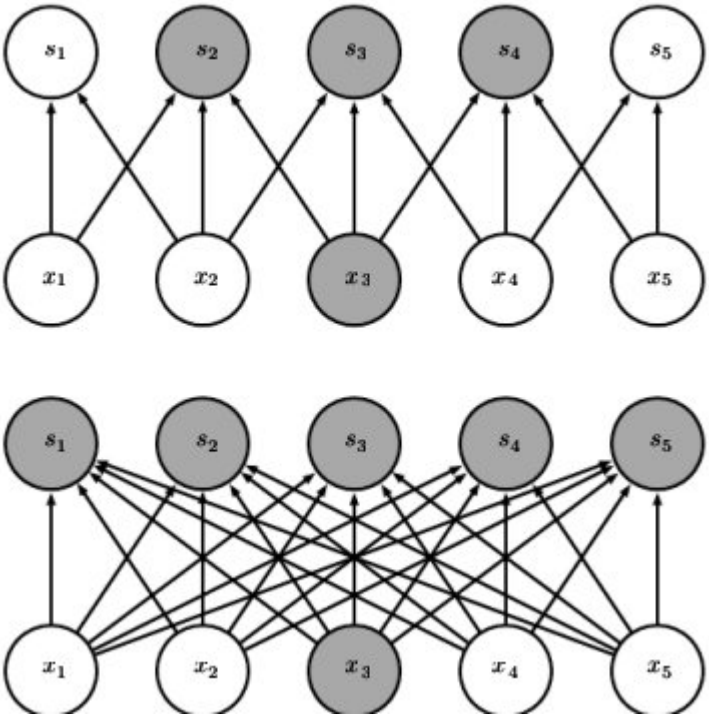
The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate. The net effect is greater progress in the more gently sloped directions of parameter space.

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

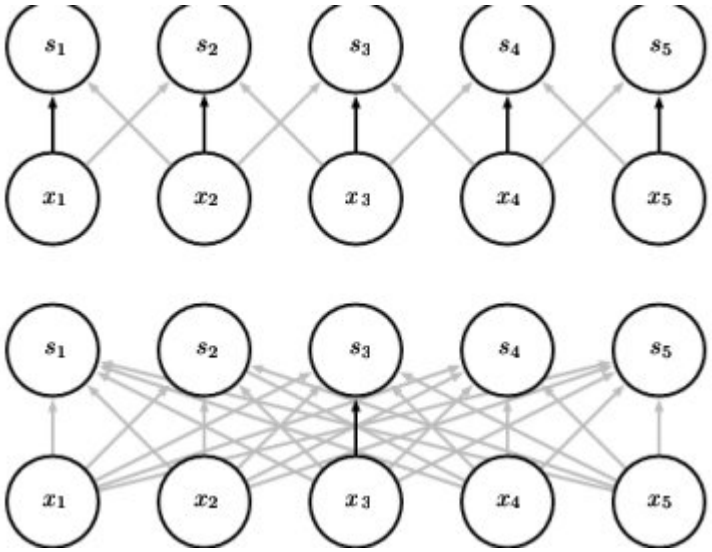
7 a. Explain the following with suitable diagram. i) Sparse interactions ii) Parameter sharing (10 Marks)

Sparse interactions : Sparse Interactions operation enables the extraction of hierarchical features from input data. Sparse Interactions: Sparse interactions within convolution optimize computation by focusing on essential connections, reducing computational
Sparse Interaction is a concept in deep learning and neural networks where only a subset of neurons or parameters interact at any given time, rather than all neurons being fully connected. This approach reduces computational complexity, memory usage, and improves efficiency, especially in large-scale mode



Sparse connectivity, viewed from below: We highlight one input unit, x3, and also highlight the output units in s that are affected by this unit. (Top)When s is formed by convolution with a kernel of width 3, only three outputs are affected by x. (Bottom)When s is formed by matrix multiplication, connectivity is no longer sparse, so all of the outputs are affected by x3.

Parameter sharing refers to using the same parameter for more than one function in a model. In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited. As a synonym for parameter sharing, one can say that a network has tied weights, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere.



Parameter sharing: Black arrows indicate the connections that use a particular parameter in two different models. (Top)The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Due to parameter sharing, this single parameter is used at all input locations. (Bottom)The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing so the parameter is used only once.

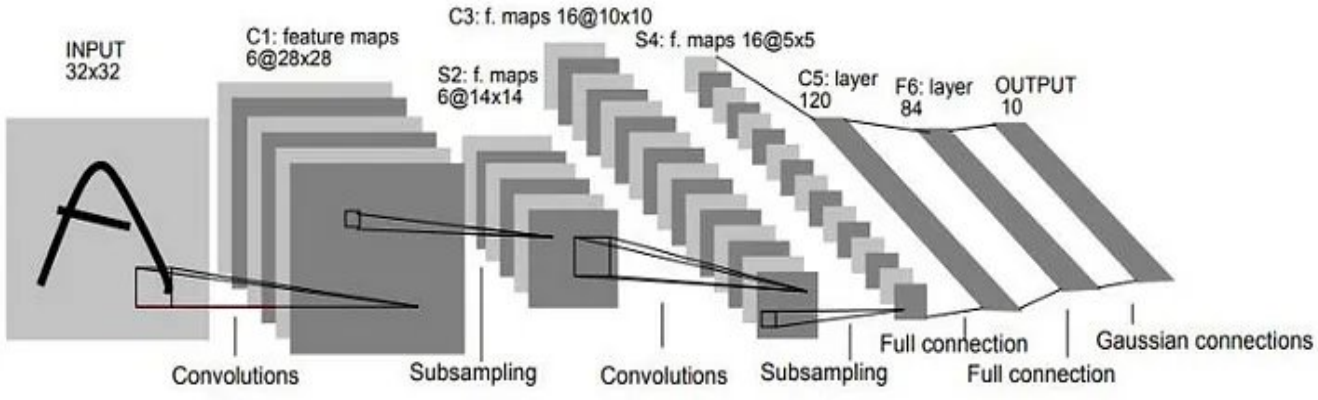
b. Explain briefly variant of the CNN models. (10 Marks)

LeNet-5:

Architecture:

- 2 convolutional layers + subsampling (pooling) layers
- Fully connected layers + SoftMax for classification

Purpose: Handwritten digit recognition (MNIST dataset)

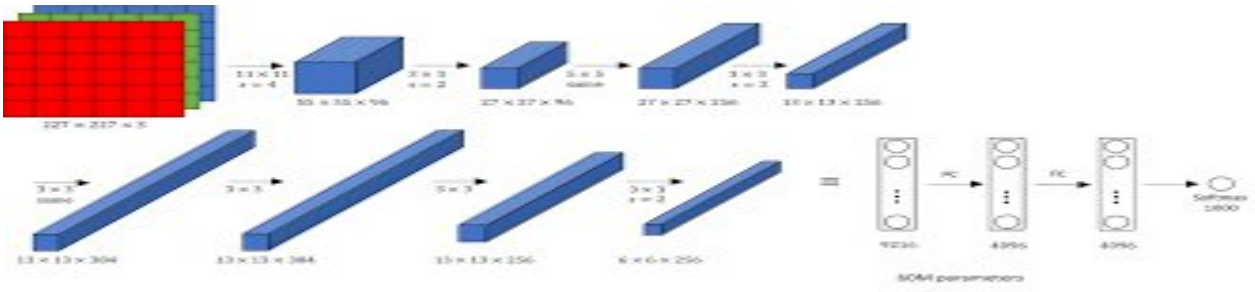


AlexNet :

Architecture:

- 5 convolutional layers + pooling layers
- ReLU activation, dropout for regularization
- Overlapping pooling and large kernels

Purpose**:** Image classification



VGGNet:

Architecture:

- Deep network (16 to 19 layers)
- Stacks of 3x3 convolutions with small receptive fields
- Fully connected layers at the end

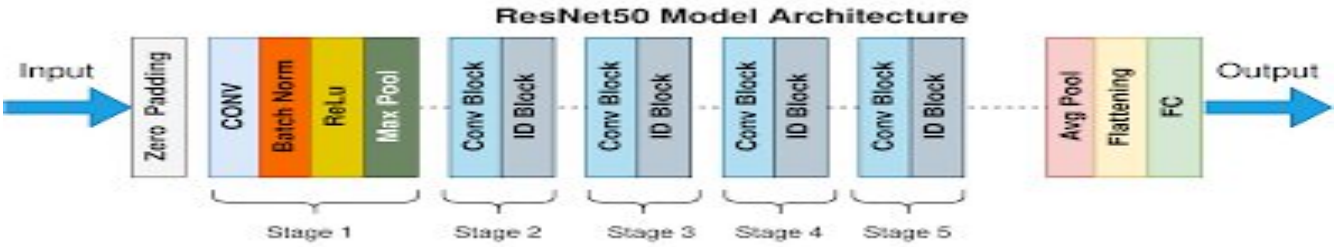
Purpose**:** Image classification, feature extraction

ResNet :

Architecture:

- Residual connections (skip connections)
- Identity mapping to preserve information
- 50, 101, 152-layer versions (very deep networks)

Purpose: Solve vanishing gradient problem in deep networks

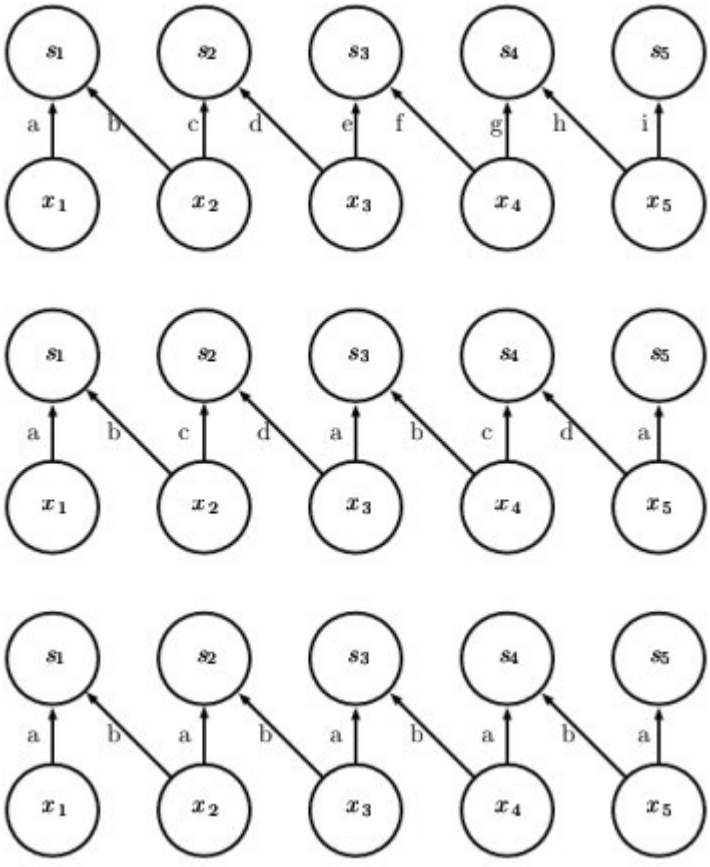


8 a. Differentiate locally connected layers, tiled convolution and standard convolution with suitable example and diagram.(10 Marks)

A locally connected layer is similar to a convolutional layer but without weight sharing. Each filter has unique weights for each location, making it more specialized to local features.

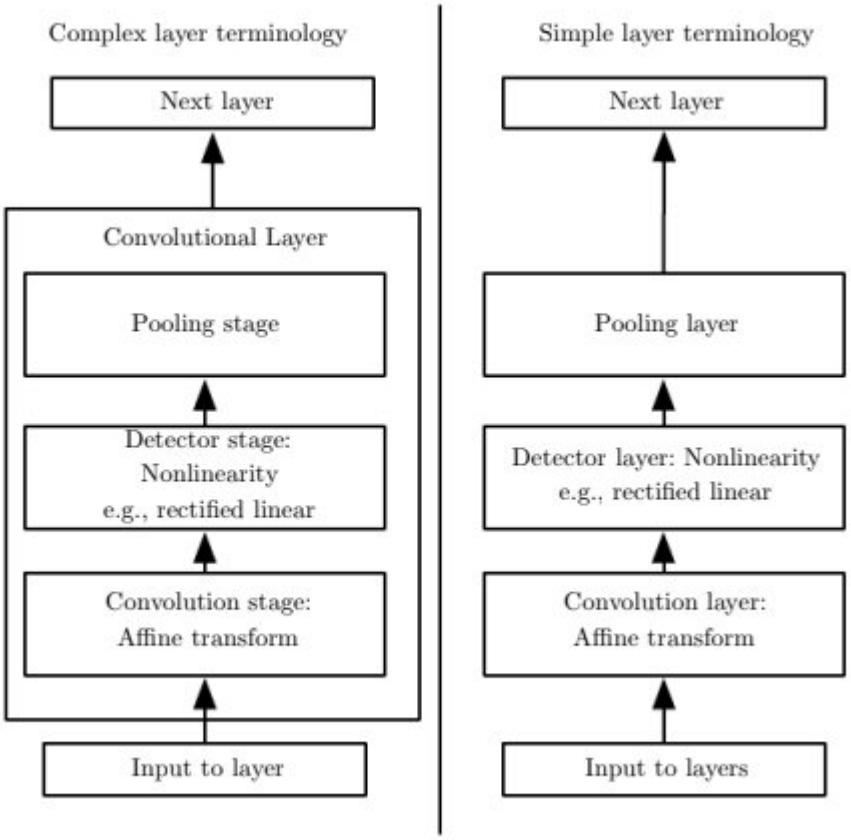
Tiled convolution is a hybrid approach between standard convolution and locally connected layers. It partially shares weights across spatial dimensions, creating periodic patterns.

A standard convolutional layer in CNNs applies the same filter (kernel) across the entire input image. This ensures translation invariance, meaning the same features can be detected anywhere in the image.



A comparison of locally connected layers, tiled convolution, and standard convolution. All three have the same sets of connections between units, when the same size of kernel is used. This diagram illustrates the use of a kernel that is two pixels wide. The differences between the methods lies in how they share parameters. (Top)A locally connected layer has no sharing at all. We indicate that each connection has its own weight by labeling each connection with a unique letter. (Center)Tiled convolution has a set of t different kernels. Here we illustrate the case oft = 2. One of these kernels has edges labeled “a” and “b,” while the other has edges labeled “c” and “d.” Each time we move one pixel to the right in the output, we move on to using a different kernel. This means that, like the locally connected layer, neighboring units in the output have different parameters. Unlike the locally connected layer, after we have gone through all available kernels, we cycle back to the first kernel. If two output units are separated by multiple t steps, then they share parameters. (Bottom)Traditional convolution is equivalent to tiled convolution with t = 1. There is only one kernel, and it is applied everywhere, as indicated in the diagram by using the kernel with weights labeled “a” and “b” everywhere.

b) . Explain the different layers in CNN models and its function with a neat diagram. (10 Marks)



The convolution layer is the core building block of CNNs. It detects patterns, edges, and features from input images by applying filters (kernels). Extracts local features from input data. Preserves spatial relationships by learning image features. Reduces dimensionality without losing important information.

pooling layer reduces the spatial dimensions of the input, retaining the most significant features. It's often used after convolution layers to down sample the feature maps. Reduces the size of feature maps (dimensionality reduction). Minimizes computational load and memory usage. Helps in making the network invariant to small translations.

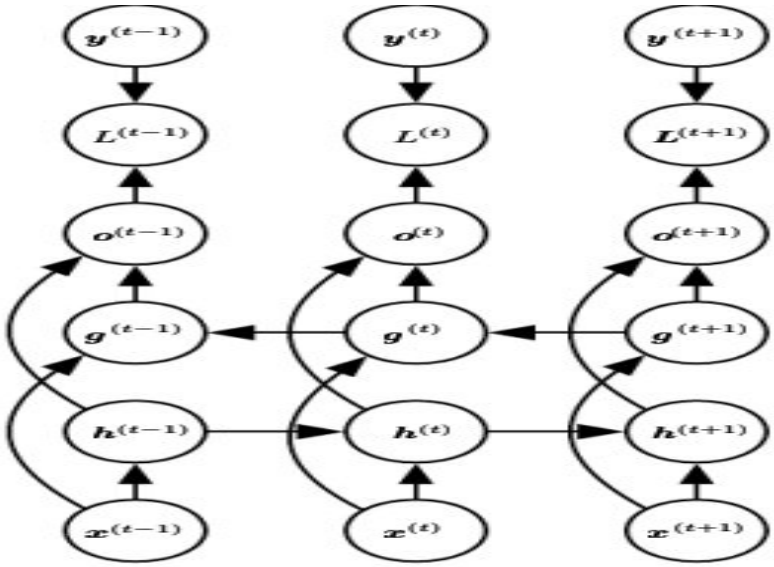
The detector layer applies a non-linear activation function to the output of convolutional layers. This introduces non-linearity to the network, enabling it to learn complex patterns. Activates neurons based on input strength. Introduces non-linearity to the network. Helps model complex relationships in data.

Module-5

9 a. Discuss about Bidirectional Recurrent neural networks. (10 Marks)

The recurrent networks we have considered up to now have a "causal" structure, meaning that the state at time t only captures information from the past(1), . . . , x(t−1), and the present input x(t).

For example, in speech recognition, the correct interpretation of the current sound as a phoneme may depend on the next few phonemes because of co-articulation and potentially may even depend on the next few words because of the linguistic dependencies between nearby words: if there are two interpretations of the current word that are both acoustically plausible, we may have to look far into the future (and the past) to disambiguate them. typical bidirectional RNN, with h(t) standing for the state of the sub-RNN that moves forward through time and g(t) standing for the state of the sub-RNN that moves backward through time. This allows the output units o(t) to compute a representation that depends on both the past and the future but is most sensitive to the input values around time t, without having to specify a fixed-size window around t



Computation of a typical bidirectional recurrent neural network, meant to learn to map input sequences x to target sequences y, with loss L(t) at each step t. The h recurrence propagates information forward in time (towards the right) while the g recurrence propagates information backward in time (towards the left). Thus at each point t, the output units o(t) can benefit from a relevant summary of the past in its h(t) input and from a relevant summary of the future in its g(t) input.

b. What is speech recognition? Explain the different types of speech recognition systems.(10 Marks)

The task of speech recognition is to map an acoustic signal containing a spoken natural language utterance into the corresponding sequence of words intended by the speaker.

The **GMM-HMM model** family treats acoustic waveforms as being generated by the following process: first an HMM generates a sequence of phonemes and discrete sub-phonemic states then a GMM transforms each discrete symbol into a brief segment of audio waveform. Although GMM-HMM systems dominated ASR until recently. Later, with much larger and deeper models and much larger datasets, recognition accuracy was dramatically improved by using neural networks to replace GMMs for the task of associating acoustic features to phonemes (or sub-phonemic states). Starting in 2009, speech researchers applied a form of deep learning based on unsupervised learning to speech recognition. This approach to deep learning was based on training undirected probabilistic models called restricted Boltzmann machines (RBMs) to model the input data.

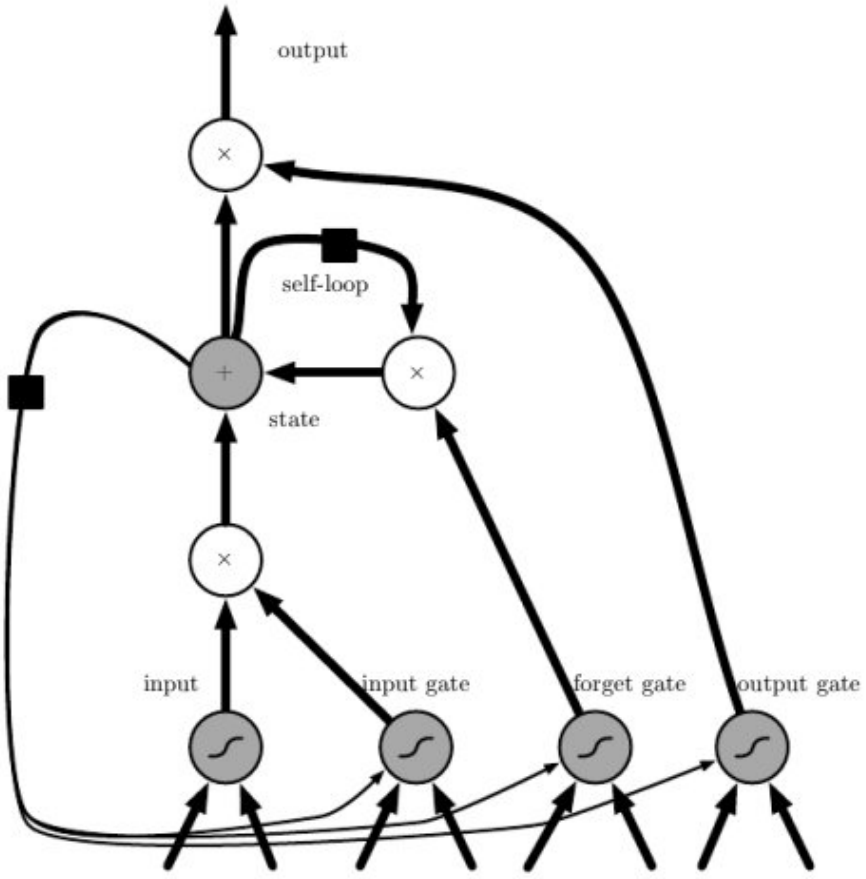
These networks take spectral acoustic representations in a fixed-size input window (around a center frame) and predict the conditional probabilities of HMM state for that center frame.

Training such deep networks helped to significantly improve the recognition rate on TIMIT), bringing down the phoneme error rate from about 26% to 20.7%.

Different types of speech recognition systems: i) Based on Speaker Dependency: Speaker-Dependent Systems , Speaker-Independent Systems ii) Based on Vocabulary Size: Small Vocabulary Systems, Medium Vocabulary Systems , Large Vocabulary Systems iii) Based on Utterance Type : Isolated Word Recognition , Connected Word Recognition iv) Based on Environment and Application : Embedded Speech Recognition Systems , Cloud-Based Speech Recognition Systems, Hybrid Speech Recognition Systems v) Based on Acoustic Models and Language Models: Acoustic Model , Language Model

10.a. Explain Long Short-Team Memory (LSTM) working principles along with all the equations.(10 Marks)

Long Short-Term Memory, is a type of recurrent neural network (RNN) that uses gates to capture both short-term and long-term memory. LSTMs are designed to process and retain information over multiple time steps. They are widely used in deep learning and are ideal for sequence prediction tasks



Block diagram of the LSTM recurrent network "cell." Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal input gate allows it. The state unit has a linear self-loop whose weight is controlled by the forget gate. The output of the cell can be shut off by the output gate. All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity. The state unit can also be used as a extra input to the gating units. The black square indicates a delay of a single time step.

Forget Gate Equation : $f_t=\sigma(W_f\cdot[h_{t-1},x_t]+b_f)$

Input Gate Equation : $i_t=\sigma(W_i\cdot[h_{t-1}, x_t]+b_i)$

Hidden State Equation : $h_t=o_t\odot\tanh(C_t)$

Cell State Equation : $C_t=f_t\odot C_{t-1}+i_t\odot C'_t$

Updates the current cell state using the forget gate, input gate, and cell state candidate.

Output Gate Equation : $o_t=\sigma(W_o\cdot[h_{t-1},x_t]+b_o)$

W_i: Weight matrix for input gate
b_i: Bias term
h_{t-1}: Previous hidden state
x_t: Current input
W_f: Weight matrix for forget gate
σ: Sigmoid activation function
C_{t-1}: Previous cell state
W_o: Weight matrix for output gate
b_o: Bias term

b. What is Natural language processing? Explain different steps involved in NLP. (10 Marks)

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that focuses on enabling computers to understand, interpret, and generate human language. It bridges the gap between human communication and computer understanding. NLP combines linguistics, computer science, and machine learning to process and analyze large amounts of natural language data.

steps in Natural Language Processing

1. Text Acquisition / Data Collection:

Gather raw textual data from various sources:
Websites, documents, social media, emails, transcripts, etc.

Determine the scope and domain of the data (e.g., news articles, product reviews)

2. Text Preprocessing (Data Cleaning):

This step prepares the raw text for analysis by standardizing and simplifying it.

1. Tokenization:
    - Splitting text into smaller units called tokens (words, phrases, or symbols).
    - Example:
      Input: "Natural Language Processing is exciting."
      Output: ["Natural", "Language", "Processing", "is", "exciting", "."]
2. Normalization / Text Lowercasing:
    - Converts all text to lowercase to avoid case sensitivity.
    - Example: "Hello" → "hello"
3. Stopword Removal:
    - Removes common words with little semantic value (e.g., "is", "and", "the").
    - Example: "This is a good movie." → "good movie"
4. Punctuation Removal:
    - Eliminates punctuation marks from text.
    - Example: "Hello, world!" → "Hello world"
5. Stemming and Lemmatization:
    - Reduces words to their root form.
    - Stemming: Chops off word endings (e.g., "playing" → "play")
    - Lemmatization: Maps words to dictionary form using grammar (e.g., "better" → "good")
6. Handling Special Characters / Noise Removal:
    - Removes unwanted symbols, numbers, and noise.
    - Example: "I l0ve NLP!!!" → "I love NLP"
7. Spelling Correction:
    - Corrects misspelled words using algorithms.
    - Example: "teh" → "the"
8. Text Segmentation (Sentence Splitting):
    - Splits text into sentences.

3. Text Representation (Feature Extraction):

Transform cleaned text into numerical data for machine learning models.

1. Bag of Words (BoW):
    - Represents text as word frequency vectors.
    - Ignores word order and context.
2. TF-IDF (Term Frequency-Inverse Document Frequency):
    - Measures the importance of words relative to the document.
    - High value for rare but important words.
3. Word Embeddings:

- Generates dense vector representations of words.
- Captures semantic meaning and relationships.
- Examples: Word2Vec, GloVe, FastText

4. Contextual Embeddings:
    - Generates dynamic vectors based on context.
    - Examples: BERT, GPT-3, RoBERTa

4. Model Building / Training:

Use machine learning or deep learning algorithms to train NLP models.

1. Select the Algorithm:
    - Traditional ML: Naive Bayes, SVM, Decision Trees, KNN
    - Deep Learning: RNN, LSTM, GRU, Transformer
    - Pre-trained Models: BERT, GPT, RoBERTa, T5
2. Train the Model:
    - Fit the model to the training data.
    - Optimize parameters to minimize errors.
3. Validation and Hyperparameter Tuning:
    - Evaluate model performance on validation data.
    - Fine-tune hyperparameters.

5. Evaluation:

Assess the model's performance using evaluation metrics:

1. Classification Tasks (e.g., Sentiment Analysis):
    - Accuracy, Precision, Recall, F1-Score, ROC-AUC
2. Sequence Prediction / Translation:
    - BLEU Score, Perplexity, Word Error Rate (WER)
3. Regression Tasks (e.g., Sentiment Scoring):
    - Mean Absolute Error (MAE), Mean Squared Error (MSE)
4. Language Models / Text Generation:
    - Perplexity, Human Evaluation

6. Post-Processing:

Refines output before presenting results.

1. Decoding and Formatting:
    - Converts predictions into a readable format.
    - For speech-to-text, use punctuation and capitalization.
2. Error Correction / Fine-Tuning:
    - Corrects errors and refines predictions.
    - Handle edge cases and context-specific corrections.

7. Deployment:

Make the NLP model available for real-world applications.

1. Deployment Platforms:

- Cloud services (AWS, Azure, Google Cloud)
- APIs, Microservices, Edge Devices

2. Scalability and Monitoring:
    - Track performance and handle varying loads.
    - Monitor for concept drift (changing patterns over time).

8. Maintenance and Optimization:

Continuously improve model performance.

1. Model Updates:
    - Retrain on fresh data to maintain accuracy.
2. Performance Monitoring:
    - Regularly assess model performance.
3. Handling Concept Drift:
    - Adapt to changing data patterns.