P.V.P Siddhartha Institute of Technology (Autonomous)

Department of Computer Science and Engineering

DEEP LEARNING IV CSE SEM-I

MID-II Answers Academic Year: 2025-26

Subject Code: 20CS4701A | Regulation: PVP20

1(a). Identify the efficient convolutional algorithms and explain with examples. (CO3-L3) (5M)

Ans:

Definition

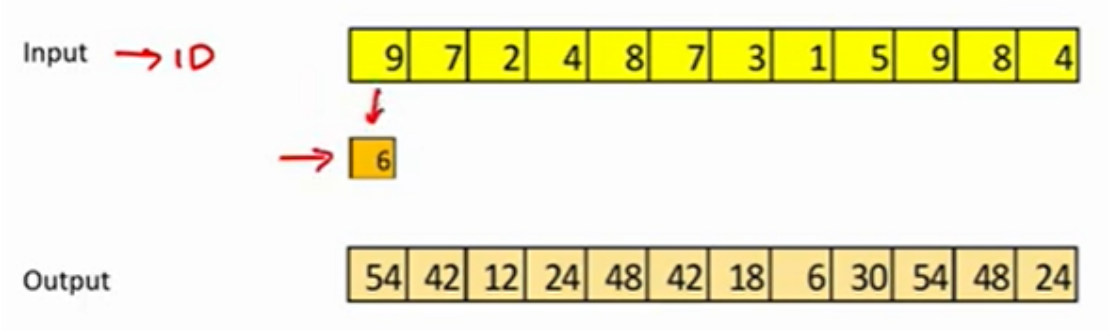
Convolution applies a small filter or kernel to local regions of an input to produce a feature map. In CNNs, the filter coefficients are weights that detect local patterns. Following the Unit-III slides, the main topics are **1D convolution, 2D convolution, Fourier-transform-based computation, and separable kernels**.

The 1D and 2D examples below use the CNN convention of sliding dot products, technically cross-correlation. Filter flipping is required for mathematical convolution. This distinction keeps the numerical examples consistent with the slides.

1. One-dimensional (1D) convolution

A 1D filter moves along one spatial or time axis. It is useful for sequential signals such as accelerometer readings, audio, and time-series data. Input values form a feature vector, filter coefficients are weights, and the resulting values form a feature map.

Original PPT diagram: 1D filtering (Unit-III, slide 4)



Example A: one-coefficient filter from the slides

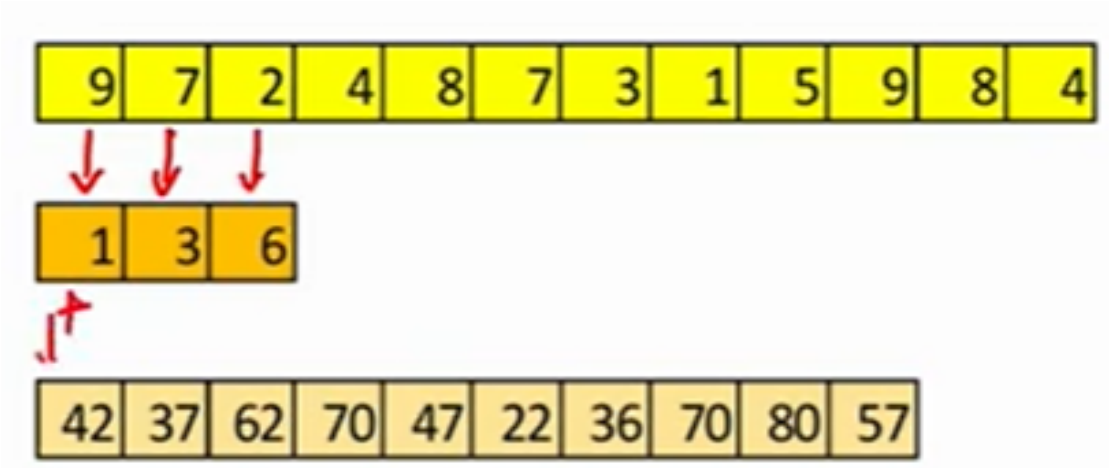
Input x = [9,7,2,4,8,7,3,1,5,9,8,4]; filter w = [6]

Step 1: multiply the first value by the coefficient: 9 × 6 = 54.
Step 2: move one position: 7 × 6 = 42.
Step 3: repeat over the full input.

Output = [54,42,12,24,48,42,18,6,30,54,48,24]

There are 12 input positions and a filter length of 1. With stride 1 and no padding, the output length remains 12.

Original PPT diagram: three-coefficient filter (slide 7)



Example B: three-coefficient filter from the slides

Filter w = [1,3,6]; stride S = 1; padding P = 0

Step 1: select the first patch [9,7,2].
Step 2: multiply corresponding values and sum: 9×1 + 7×3 + 2×6 = **42**.
Step 3: move right to [7,2,4]: 7×1 + 2×3 + 4×6 = **37**.
Step 4: continue until the last complete patch [9,8,4], giving 9 + 24 + 24 = **57**.

Output = [42,37,62,70,47,22,36,70,80,57]

L_out = floor((L_in + 2P - K)/S) + 1

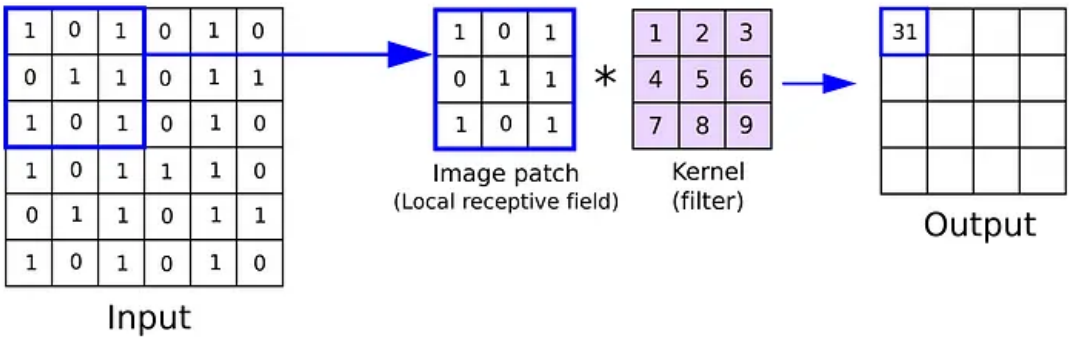
L_out = floor((12 + 0 - 3)/1) + 1 = 10

The spatial length decreases because only complete patches are used. Adding one zero at each end with K = 3 and S = 1 preserves length 12, although padding changes boundary values and does not restore information already discarded.

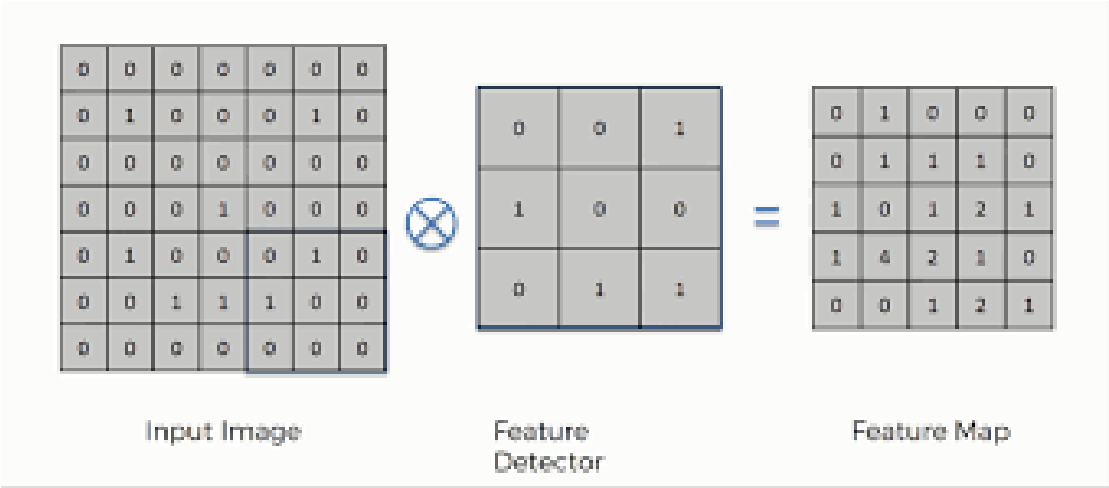
2. Two-dimensional (2D) convolution

A 2D kernel moves horizontally and vertically over spatial input, usually an image. At each location, multiply the filter-sized patch element by element with the kernel and sum the products. Repeating this operation gives a feature map that can highlight edges, lines, or other patterns. An image tensor also has a channel axis; each output filter normally combines all input channels.

Original PPT diagram: image patch and kernel (slide 8)



Original PPT diagram: edge detector and feature map (slide 9)



Step-by-step 2D example

X = [[1,2,3], [4,5,6], [7,8,9]]; W = [[1,0], [0,1]]

Use stride 1 and no padding.

Top-left: 1×1 + 2×0 + 4×0 + 5×1 = 6.

Top-right: 2×1 + 3×0 + 5×0 + 6×1 = 8.

Bottom-left: 4×1 + 5×0 + 7×0 + 8×1 = 12.

Bottom-right: 5×1 + 6×0 + 8×0 + 9×1 = 14.

Feature map Y = [[6,8], [12,14]]

The output height and width are each (3 - 2)/1 + 1 = 2. Different learned or hand-designed kernels respond to different image patterns.

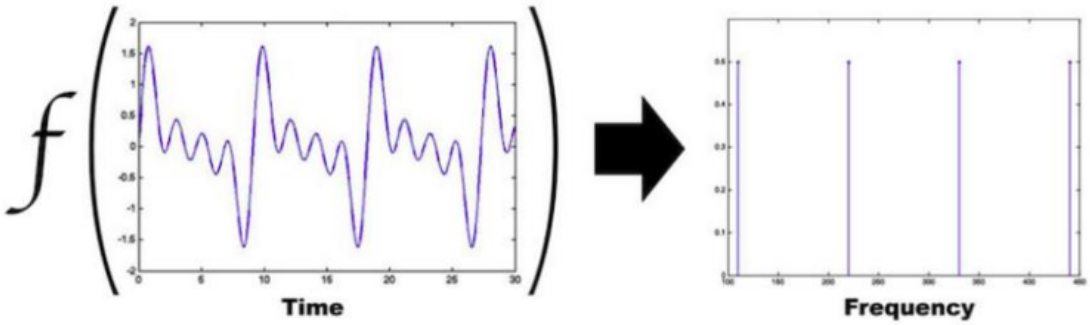
3. Fourier-transform-based convolution

A Fourier transform converts a signal from a time or spatial representation into a frequency representation. By the convolution theorem, convolution in the original domain corresponds to pointwise multiplication in the frequency domain. An inverse transform converts the result back.

$F(x * k) = F(x) \times F(k)$

$x * k = F^{-1}(F(x) \times F(k))$

Original PPT diagram: time to frequency domain (slide 11)



Original PPT formula: convolution theorem (slide 12)

$$f \star g = F^{-1}\{F\{f \star g\}\} = F^{-1}\{F\{f\} \times F\{g\}\}$$

Step 1: choose x = [1,2] and kernel k = [1,1].
Step 2: pad both to length at least 2 + 2 - 1 = 3.
Step 3: compute their discrete Fourier transforms and multiply corresponding frequency values.
Step 4: apply the inverse transform. The linear convolution is **[1,3,2]**. Its direct check is [1×1, 1×1 + 2×1, 2×1].

Zero-padding prevents circular wraparound. FFT computation can be advantageous for sufficiently large signals or kernels, although transforms and memory operations add overhead. Exact arithmetic preserves the complete signal under an invertible transform; numerical implementations have finite-precision errors.

4. Separable kernels

A spatially separable 2D kernel can be expressed as the outer product of a column vector and a row vector. Its 2D filtering can then be performed as two 1D passes, reducing arithmetic. Spatial separability is distinct from frequency-domain transformation.

$a = [1,2,1]^T$; $b = [1,2,1]$

$K = a\ b = [[1,2,1], [2,4,2], [1,2,1]]$

Step 1: filter horizontally with [1,2,1].
Step 2: filter the intermediate result vertically with [1,2,1].
Step 3: use consistent padding and boundary conventions to obtain the same result as the full separable 3 × 3 kernel.

Direct cost per output $\approx K^2 = 9$ multiplications

Two 1D passes $\approx 2K = 6$ multiplications per output

For a 5 × 5 separable kernel, the analogous count decreases from 25 to 10 multiplications per output, apart from boundary and implementation overhead. Only kernels with the required factorization can use this exact two-pass representation.

2(a). Model the structure of an LSTM component. (CO3) (3M)

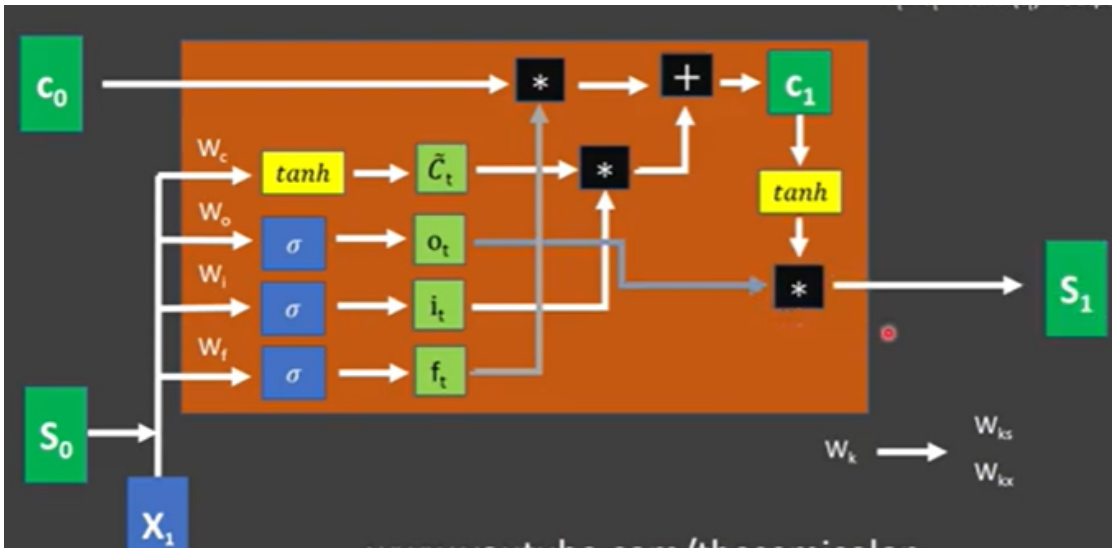
Ans:

Definition

Long Short-Term Memory (LSTM), as described in the Unit-IV slides, is a recurrent neural network component that maintains a cell state for long-term information and a hidden state for the current output. Forget, input, and output gates control information flow, helping the network learn dependencies across longer sequences.

Architecture of a standard LSTM cell

Original PPT diagram: complete LSTM cell (Unit-IV, slide 33)



The PPT labels the previous hidden state as S0 and the updated state as S1. These correspond to h(t-1) and h(t) below. The cell state follows the upper memory path, while sigmoid and tanh blocks compute the gates and candidate update. The equations below make the diagram operations explicit.

Equations

$z_t = [h_{t-1} ; x_t]$

$f_t = \sigma(W_f z_t + b_f)$ $i_t = \sigma(W_i z_t + b_i)$

$g_t = \tanh(W_g z_t + b_g)$ $o_t = \sigma(W_o z_t + b_o)$

$c_t = f_t \odot c_{t-1} + i_t \odot g_t$

$h_t = o_t \odot \tanh(c_t)$

σ is the sigmoid, whose output lies between 0 and 1; tanh lies between -1 and 1; and the circled-dot symbol denotes elementwise multiplication. x is the input, h is the hidden state, c is the cell state, and W and b are learned parameters.

Step-by-step working, following Unit-IV

Step 1: Receive the inputs. Each cell receives the current input x(t), previous hidden state h(t-1), and previous cell state c(t-1). It produces updated states h(t) and c(t). In a normal network these are vectors, not single scalar values.

Step 2: Compute the forget gate. Apply a learned affine transformation to [h(t-1); x(t)] and then a sigmoid. The gate scales the old cell state: values near 0 suppress information and values near 1 retain it. The sigmoid produces continuous values between 0 and 1, rather than only binary decisions.

Step 3: Compute the input gate and candidate memory. A sigmoid determines how strongly each candidate value should enter memory. A separate tanh transformation constructs candidate values between -1 and 1. Their elementwise product is the proposed memory addition.

Step 4: Update cell memory. Multiply the old cell state by the forget gate, then add the input-gated candidate. This combines retained information with useful new information.

Step 5: Compute the output gate. Another sigmoid controls which parts of the updated memory contribute to the hidden output.

Step 6: Produce the hidden state. Apply tanh to the new cell state and multiply by the output gate. Pass h(t) and c(t) to the next time step; h(t) can also feed an output layer.

This organization supports long-term dependencies by maintaining a gated memory path. A conventional RNN can in principle represent information from earlier steps, but learning long dependencies can be difficult because of vanishing or exploding gradients. LSTM gates improve that learning behavior.

Step-by-step scalar example

For a one-unit LSTM, assume the gate preactivations are a_f = 0, a_i = 0, a_g = 0.549306, and a_o = 0, with previous cell state c(t-1) = 0.8. These preactivations are illustrative values produced by the learned affine transformations.

$f = \sigma(0) = 0.5;\ i = \sigma(0) = 0.5;\ o = \sigma(0) = 0.5$

$g = \tanh(0.549306) \approx 0.5$

$c = 0.5 \times 0.8 + 0.5 \times 0.5 = 0.4 + 0.25 = 0.65$

$h = 0.5 \times \tanh(0.65) \approx 0.5 \times 0.57167 = 0.28584$

The cell retains 0.4 units of old memory and adds 0.25 units of candidate memory. It then exposes part of that updated memory as the hidden state. LSTMs support language modeling, sequence labeling, speech processing, and time-series prediction; they mitigate rather than completely eliminate gradient difficulties.

2(b). Compare recurrent networks and recursive networks. (CO4-L4) (2M)

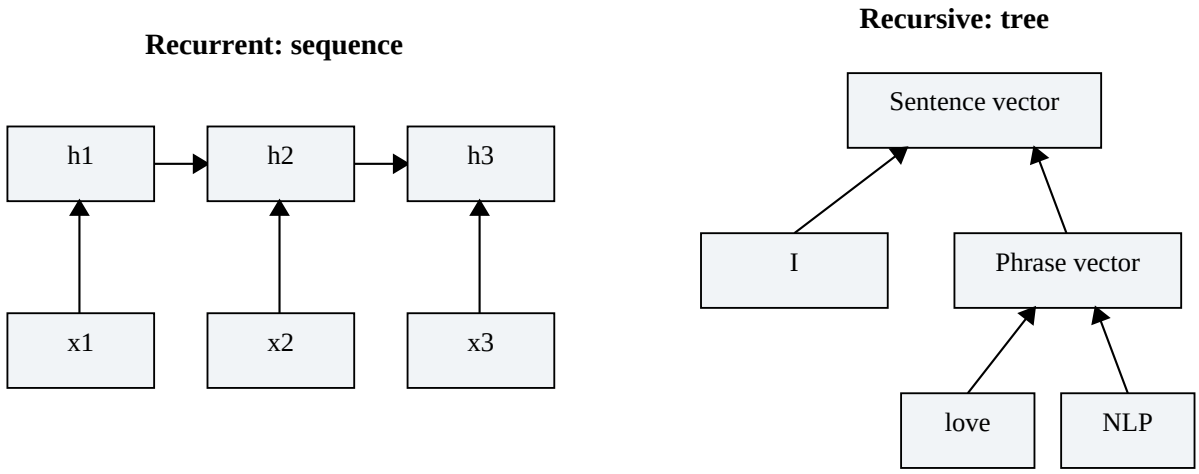
Ans:

Definitions

Recurrent networks apply a shared state-update function along a sequence. Each hidden state depends on the current input and the preceding hidden state.

Recursive networks apply a shared composition function over a hierarchy, usually a tree. Child representations combine to form parent representations.

Architectures: sequence versus tree



Aspect	Recurrent	Recursive
Input structure	Ordered sequence	Tree or hierarchy
Computation	Previous state plus current input	Compose child-node vectors
Parameter sharing	Across sequence positions	Across tree nodes
Typical application	Time series, language modeling	Tree-based sentence representation

Step-by-step example: "I love NLP"

Recurrent: process "I" to produce h1, then "love" with h1 to produce h2, then "NLP" with h2 to produce h3. A final-state classifier can use h3 as a sentence representation.

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

Recursive: use the illustrative tree [I [love NLP]]. First combine the vectors for "love" and "NLP" into a phrase vector p; then combine "I" and p into a sentence vector s.

$p = \tanh(W[v_{love} ; v_{NLP}] + b)$

$s = \tanh(W[v_I ; p] + b)$

Both learn representations with shared parameters. The main difference is sequential versus hierarchical computation. The supplied PPTs do not include this comparison, so this answer and its diagram supplement the slides.

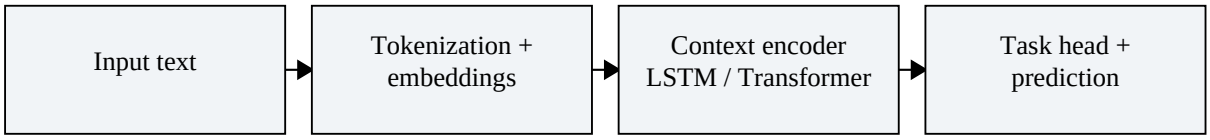
3(a). Examine the primary objective of Deep Learning in NLP. How do deep neural networks handle complex language understanding tasks? (CO4-L4) (5M)

Ans:

Definition and objective

Deep learning in Natural Language Processing learns numerical representations of text and maps them to task outputs. Its primary objective is to learn contextual and semantic patterns that support applications such as translation, sentiment analysis, question answering, and text generation.

Architecture: NLP classification pipeline



A CNN can detect local word patterns, an LSTM can accumulate sequential information, and a Transformer can combine token information through attention. The task head converts these representations into labels, answer spans, or generated tokens. This supplementary pipeline is provided because the uploaded PPTs do not cover Question 3.

Step 1: Tokenize and represent the input

Example: **"The movie was not good."** Tokenize into [The, movie, was, not, good, .] or corresponding subwords. Map tokens into learned vectors. Attention-based encoders also receive position information so they can represent word order.

Step 2: Learn the context

The encoder combines information across tokens. It can learn that "not" changes the sentiment associated with "good". Contextual representations also help distinguish "bank" in "deposit money at the bank" from "sit on the river bank". Training examples teach these relationships rather than requiring a separate manually written rule for every sentence.

Step 3: Connect relevant words with attention

$$\text{Attention}(Q,K,V) = \text{softmax}(QK^T / \sqrt{d_k})V$$

Queries and keys determine relevance scores, and values provide the content combined by attention weights. For an illustrative pair of scores [0, ln(3)], softmax gives weights [0.25, 0.75]. The combined vector is 0.25v1 + 0.75v2. These numbers demonstrate the operation and are not attention measurements from a trained model.

Step 4: Predict sentiment

A classification head receives a pooled or task-specific sentence representation. Suppose its logits for [negative, positive] are [1.5, 0.5].

$$P(\text{negative}) = e^{1.5} / (e^{1.5} + e^{0.5}) \approx 0.7311$$

$$P(\text{positive}) \approx 0.2689$$

The model predicts **negative**. It can use the contextual interaction of "not" and "good" rather than treating "good" alone as proof of positive sentiment.

Step 5: Train and evaluate

If the true label is negative, the cross-entropy loss is $-\ln(0.7311) \approx 0.3133$. Backpropagation computes gradients, and an optimizer updates the trainable parameters. General language pretraining followed by suitable task adaptation can reduce the amount of task-specific labeled data needed. Evaluate performance on held-out data.

Language challenge	How a deep model can help
Contextual meaning	Embeddings updated by surrounding text support disambiguation.
Long dependencies	Recurrent memory or attention links relevant distant information.
Complex patterns	Stacked layers learn features useful for task predictions.
Different tasks	Task-specific output heads produce labels, spans, or token sequences.

Source material

Question 1: supplied Unit-III, Efficient Convolution Algorithms, slides 3-13. Question 2(a): supplied Unit-IV, Long Short Term Memory, including its original architecture diagram, gate equations, and working stages. Questions 2(b) and 3 are supplementary answers because those topics are not covered in the supplied PPTs. Numerical examples are shown with explicit assumptions. The LSTM equations are also consistent with the standard LSTMCell formulation.