CBCS SCHEME

USN [redacted] BAI701

Seventh Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026
Deep Learning and Reinforcement Learning

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	What is Deep Learning? Write a program to demonstrate the working of a deep neural network for classification task.	07	L1	CO1
	b.	Explain the prominent challenges involved in optimization during the training deep neural networks.	07	L2	CO1
	c.	Explain with an example how maximum likelihood estimation decomposes into a sum over training examples. List the key factors that influence the choice of mini batch size in deep learning.	06	L2	CO1
		OR			
Q.2	a.	Differentiate between conventional Machine learning and Deep Learning approaches in terms of feature extraction and representation learning. Illustrate your answer with a suitable diagram.? List the challenges associated with deep learning.	10	L2	CO1
	b.	Define surrogate Loss function and early stopping. What are the key factors that influence the choice of minibatch size in deep learning?	10	L1	CO1
		Module – 2			
Q.3	a.	What is convolution layer? Explain the convolution neural network layers in detail.	10	L1	CO1
	b.	Explain the activation function used in Artificial Neural Networks: i) RLU (Rectified Linear Unit) ii) Logistic / Sigmoid Function iii) Tanh Function. iv) Softmax Function	10	L2	CO1
		OR			
Q.4	a.	With the help of an example, explain the convolution Operation. List the applications of Deep Learning.	10	L2	CO2
	b.	List and explain the evolution of Convolution Neural Network.	10	L2	CO2
		Module – 3			
Q.5	a.	Outline the CNN architecture in detail including its mathematical operations.	10	L2	CO2
	b.	Explain the working of Alex Net, Highlighting its key layers and features, with a suitable architecture diagram. Design and implement a Convolutional Neural Network for classification of image data set.	10	L2	CO2

		OR			
Q.6	a.	Illustrate the feature map transformations in LeNet – 5 for an input image of size 32 × 32. Develop and implement a deep learning network for forecasting time series data.	10	L2,3	CO2
	b.	List and explain the Gradient Descent Variants. What are the challenges in Training Deep Networks?	10	L3	CO2
		Module – 4			
Q.7	a.	How does unfolding a Recurrent Neural Network (RNN) represent recurrence as a computational graph? State any two advantages of unfolding.	10	L1	CO3
	b.	Explain Gated Recurrent Neural Networks (RNNs) with reference of LSTM and GRU. Describe their purpose and working.	10	L2	CO3
		OR			
Q.8	a.	List and explain the three parameters transformations in a RNN. What roles do skip connections play in deep RNNs?	10	L1	CO3
	b.	Explain the architecture and working of Bidirectional Recurrent Neural Networks (BRNN). How do they address the limitations of causal RNNs? List the applications in which they are most effective.	10	L2	CO3
		Module – 5			
Q.9	a.	Explain with an example, how Reinforcement Learning uses reward – driven trail and error in two environments, such as video games and self-driving cars.	10	L2	CO3
	b.	Compare traditional table – based Reinforcement Learning and Deep Reinforcement Learning using examples of Tic – Tac – Toe and chess.	10	L2	CO3
		OR			
Q.10	a.	Explain how a Mouse learns to find cheese in maze through interaction with its environment using the Reinforcement Learning framework?	10	L2	CO3
	b.	Explain how Reinforcement Learning was applied in Focebook's negotiation chatbot. How did self – play and reward – based learning improve negotiation behavior?	10	L1	CO3

* * * * *

Module 1

Q1(a) What is Deep Learning? Write a program to demonstrate a deep neural network for classification

Deep Learning is a subfield of machine learning that uses multi-layered artificial neural networks to automatically learn hierarchical feature representations from data.

Key characteristics

- Multiple hidden layers
- Automatic feature learning
- Trained using backpropagation and gradient descent

Simple Python Program (DNN for Classification)

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
   Dense(64, activation='relu', input_shape=(20,)),
   Dense(32, activation='relu'),
   Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
          loss='binary_crossentropy',
          metrics=['accuracy'])

model.summary()
```

Q1(b) Challenges in optimization during training deep neural networks

Initial drawbacks of deep learning included slow training and lack of computational power, making other methods like kernel approaches more favorable.

The revival of deep learning was driven by advances in GPU computing and access to large labeled datasets.

Training deep networks is still complex due to the large number of parameters and the difficulty of optimizing stacked nonlinear layers.

Researchers have improved training through:

- Better optimizers
- Smarter initialization
- Advanced activation functions
- Skip connections

Data dependency remains a limitation—deep learning requires large datasets, which are not available in all domains.

Limited flexibility: Deep models excel at single tasks, but struggle with multitasking and require retraining even for similar problems.
There is an ongoing need to develop more adaptable, data-efficient, and general-purpose deep learning models.

Data dependency remains a limitation—deep learning requires large datasets, which are not available in all domains.

Limited flexibility: Deep models excel at single tasks, but struggle with multitasking and require retraining even for similar problems

There is an ongoing need to develop more adaptable, data-efficient, and general-purpose deep learning models.

Q1(c) Explain Maximum Likelihood Estimation (MLE) and mini-batch size factors

Maximum Likelihood Estimation (MLE)

Maximum Likelihood Estimation (MLE) is a **statistical method** used to estimate the parameters of a model by **maximizing the probability of observing the given training data** under the assumed model.

Problem Setting

Let:

- $\{(x_i, y_i)\}_{i=1}^{N}$ be the training dataset
- θ be the model parameters
- $p(y|x;\theta)$ be the model's conditional probability

The likelihood function is:

$$\mathcal{L}(\theta) = \prod_{i=1}^{N} p(y_i|x_i;\theta)$$

Since products are difficult to optimize numerically, we take the logarithm:

$$\log \mathcal{L}(\theta) = \sum_{i=1}^{N} \log p(y_i|x_i;\theta)$$

Key Observation (Decomposition)

The total log-likelihood **decomposes into a sum over individual training examples**, which allows:

- Efficient optimization
- Gradient computation per sample
- Mini-batch and stochastic training

Relation to Loss Function

In deep learning, **negative log-likelihood (NLL)** is used as the loss:

$$\mathcal{J}(\theta) = -\sum_{i=1}^{N} \log p(y_i|x_i;\theta)$$

Examples:

- **Regression** → Mean Squared Error (Gaussian assumption)
- **Binary classification** → Binary Cross-Entropy
- **Multi-class classification** → Categorical Cross-Entropy

Thus, **training a neural network using backpropagation is equivalent to performing MLE** under suitable probability assumptions.

Mini-Batch Gradient Descent in MLE

Instead of computing the loss over all N samples, the dataset is divided into **mini-batches** of size B:

$$\mathcal{J}_{batch}(\theta) = -\frac{1}{B}\sum_{i=1}^{B} \log p(y_i|x_i;\theta)$$

This significantly reduces computation while preserving good gradient estimates.

Q2(a) Differentiate Conventional ML vs Deep Learning

Aspect	Conventional ML	Deep Learning
Feature extraction	Manual	Automatic
Data requirement	Small	Large
Model depth	Shallow	Deep
Performance	Limited	High

Figure compares traditional machine learning approach based on handcrafted features to deep learning approach based on hierarchical representation learning.

- The word “deep” refers to learning successive layers of increasingly meaningful representations of input data.
- The number of layers used to model the data determines the depth of the model. Current deep learning often involves learning tens or even hundreds of successive layers of representation from the training data automatically. The conventional approaches to machine learning often focus on learning only one or two layers of representations of data; such approaches are often categorized as shallow learning.

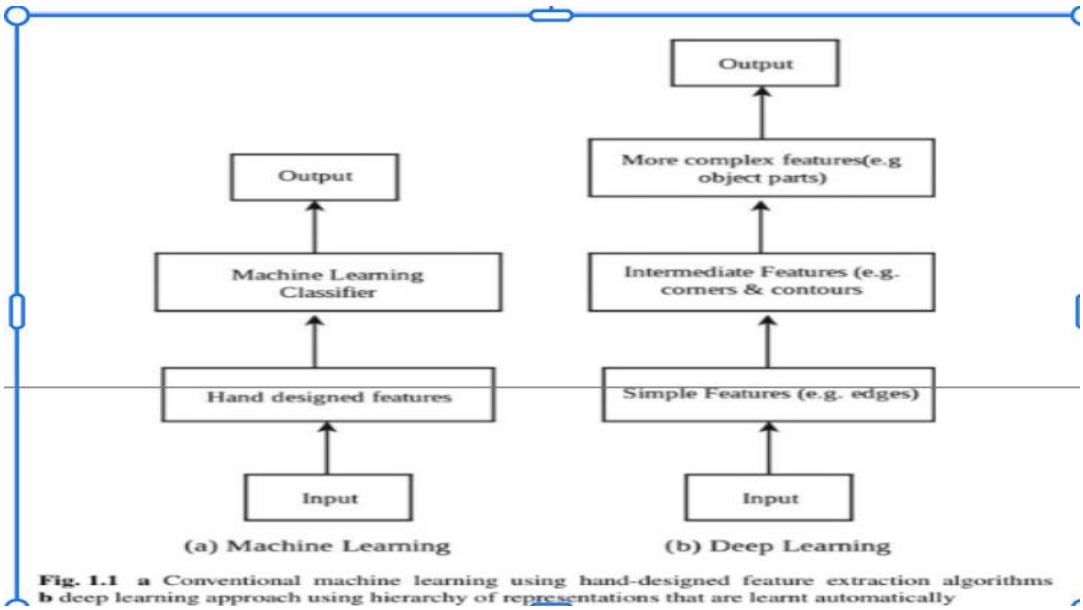
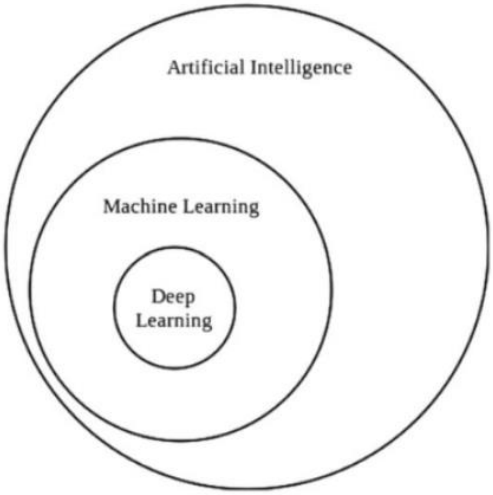


Fig. 1.1 a Conventional machine learning using hand-designed feature extraction algorithms b deep learning approach using hierarchy of representations that are learnt automatically

Deep learning and machine learning are subfields of Artificial Intelligence (AI).

Figure 1.2 illustrates the relationship between AI, machine learning, and deep learning.



Some of the aspects that helped in the evolution of deep networks are listed below:

- Improved computational resources for processing massive amounts of data and training much larger models.

- Automatic feature extraction.

The learning mechanisms used by deep learning models are in no way comparable to the human brain, but can be described as a mathematical framework for learning representations from data.

Q2(b) Surrogate loss function and Early stopping

Sometimes, the loss function we actually care about (say classification error) is not one that can be optimized efficiently. For example, exactly minimizing expected 0-1 loss is typically intractable (exponential in the input dimension), even for a linear classifier (Marcotte and Savard, 1992). In such situations, one typically optimizes a surrogate loss function instead, which acts as a proxy but has advantages. For example, the negative log-likelihood of the correct class is typically used as a surrogate for the 0-1 loss. The negative log-likelihood allows the model to estimate the conditional probability of the classes, given the input, and if the model can do that well, then it can pick the classes that yield the least classification error in expectation.

One aspect of machine learning algorithms that separates them from general optimization algorithms is that the objective function usually decomposes as a sum over the training examples. Optimization algorithms for machine learning typically compute each update to the parameters based on an expected value of the cost function estimated using only a subset of the terms of the full cost function.

For example, maximum likelihood estimation problems, when viewed in log space, decompose into a sum over each example:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}). \tag{8.4}$$

Maximizing this sum is equivalent to maximizing the expectation over the empirical distribution defined by the training set:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}, y; \boldsymbol{\theta}). \tag{8.5}$$

Most of the properties of the objective function J used by most of our optimization algorithms are also expectations over the training set. For example, the most commonly used property is the gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} \nabla_{\boldsymbol{\theta}} \log p_{\mathrm{model}}(\boldsymbol{x}, y; \boldsymbol{\theta}). \tag{8.6}$$

Computing this expectation exactly is very expensive because it requires evaluating the model on every example in the entire dataset. In practice, we can compute these expectations by randomly sampling a small number of examples from the dataset, then taking the average over only those examples.

Recall that the standard error of the mean (equation 5.46) estimated from n samples is given by $\sigma/\sqrt{n}$, where σ is the true standard deviation of the value of the samples. The denominator of $\sqrt{n}$ shows that there are less than linear returns to using more examples to estimate the gradient. Compare two hypothetical estimates of the gradient, one based on 100 examples and another based on 10,000 examples. The latter requires 100 times more computation than the former, but reduces the standard error of the mean only by a factor of 10. Most optimization algorithms converge much faster (in terms of total computation, not in terms of number of updates) if they are allowed to rapidly compute approximate estimates of the gradient rather than slowly computing the exact gradient.

Module 2

Q3(a) Convolution layer and CNN layers

Convolutional Neural Network also known as ConvNet or CNN is a deep learning technique that consists of multiple numbers of layers. ConvNets are inspired by the biological visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. Different neurons in the brain respond to different features. For example, certain neurons fire only in the presence of lines of a certain orientation, some neurons fire when exposed to vertical edges and some when shown horizontal or diagonal edges. This idea of certain neurons having a specific task is the basis behind ConvNets. ConvNets have shown excellent performance on several applications such as image classification, object detection, speech recognition, natural language process-ng, and medical image analysis. Convolutional neural networks are powering core of computer vision that has many applications which include self-driving cars, robotics, and treatments for the visually impaired. The main concept of ConvNets is to obtain local features from input (usually an image) at higher layers and combine them into more complex features at the lower layers. However, due to its multilayered architecture, it is computationally exorbitant and training such networks on a large dataset

Convolution Layer: Convolution layer is the core building block of a convolutional neural network which uses convolution operation (represented by *) in place of general matrix multiplication. Its parameters consist of a set of learnable filters also known as kernels. The main task of the convolutional layer is to detect features found within local regions of the input image that are common throughout the dataset and mapping their appearance to a feature map. A feature map is obtained for each filter in the layer by repeated application of the filter across subregions of the complete image, i.e., convolving

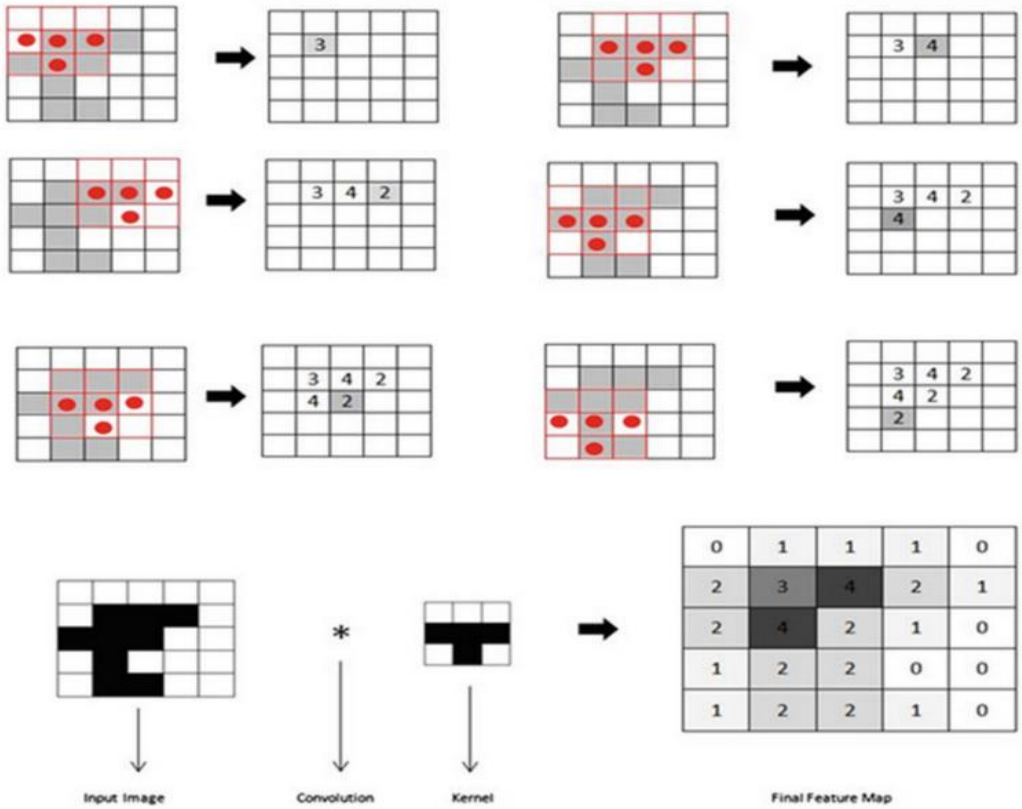


Fig. 2.5 Example of convolution operation

the filter with the input image, adding a bias term, and then applying an activation function. The input area on which a filter is applied is called local receptive field. The size of the receptive field is same as the size of the filter.

Figure 2.5 shows how a filter (T-shaped) is convolved with the input to get the feature map. Feature map is obtained after adding a bias term and then applying a nonlinear function to the output of the convolution operation. The purpose of nonlinearity function is to introduce nonlinearity in the ConvNet model, and there are a number of nonlinearity functions available which are briefly explained in the next section. Filters/Kernels The weights in each convolutional layer specify the convolution filters and there may be multiple filters in each convolutional layer. Every filter contains some feature like edge, corner, etc. and during forward pass, each filter is slid ac oss the width and height of the input generating feature map of that filter. Hyperparameters Convolutional neural network architecture has many hyperparameters that are used to control the behavior of the model. Some of these hyperparameters control the size

Basics of Supervised Deep Learning of the output while some are used to tune the running time and memory cost of the model. The four important hyperparameters in the convolution layer of the ConvNet are given below:

Example

a. Filter Size: Filters can be of any size greater than 2 × 2 and less than the size of the input but the conventional size varies from 11 × 11 to 3 × 3. The size of a filter is independent of the size of input.

b. Number of Filters: There can be any reasonable number of filters. AlexNet used 96 filters of size 11 × 11 in the first convolution layer. VGGNet used 96 filters of size 7 × 7, and another variant of VGGNet used 64 filters of size 11 × 11 in first convolution layer.

c. Stride: It is the number of pixels to move at a time to define the local receptive field for a filter. Stride of one means to move across and down a single pixel. The value of stride should not be too small or too large. Too small stride will lead to heavily overlapping receptive fields and too large value will overlap less and the resulting output volume will have smaller dimensions spatially.

d. Zero Padding: This hyperparameter describes the number of pixels to pad the input image with zeros. Zero padding is used to control the spatial size of the output volume. Each filter in the convolution layer produces a feature map of size ([A − K + 2P]/S) + 1 where A is the

input volume size, K is the size of the filter, P is the number of padding applied and S is the stride. Suppose the input image has size 128 × 128, and
5 filters of size 5 × 5 are applied, with single stride and zero padding, i.e., A 128, F 5, P 0 and S 1. The number of feature maps produced will be equal to the number of filters applied, i.e., 5 and the size of each feature map will be ([128 – 5 + 0]/1) + 1 124. Therefore, the output volume will be 124 × 124 × 5.

Q3(b) Activation Functions

- **Logistic/Sigmoid Activation Function**: The sigmoid function is mathematically represented as

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

It is an S-shaped curve as shown in Fig. 2.6. Sigmoid function squashes the input into the range [0, 1].

- **Tanh Activation Function**: The hyperbolic tangent function is similar to sigmoid function but its output lies in the range [−1, 1]. The advantage of tanh over *sigmoid* is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph as shown in Fig. 2.7.
- **Softmax Function** (**Exponential Function**): It is often used in the output layer of a neural network for classification. It is mathematically represented as

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}}$$

The softmax function is a more generalized logistic activation function which is used for multiclass classification.

- **ReLU Activation Function**: Rectified Linear Unit (ReLU) has gained some importance in recent years and currently is the most popular activation function for deep neural networks. Neural networks with ReLU train much faster than other

Q4(a) Convolution operation with example

Convolution is a mathematical operation performed on two functions and is written as $(f * g)$, where f and g are two functions. The output of the convolution operation for domain n is defined as

$$(f * g)(n) = \sum_m f(m)g(n-m)$$

For time-domain functions, n is replaced by t. The convolution operation is commutative in nature, so it can also be written as

$$(f * g)(n) = \sum_m f(n-m)g(m)$$

Convolution operation is one of the important operations used in digital signal processing and is used in many areas which includes statistics, probability, natural language processing, computer vision, and image processing.

Convolution operation can be applied to higher dimensional functions as well. It can be applied to a two-dimensional function by sliding one function on top of another, multiplying and adding. Convolution operation can be applied to images to perform various transformations; here, images are treated as two-dimensional functions. An example of a two-dimensional filter, a two-dimensional input, and a two-dimensional feature map is shown in Fig. 2.4. Let the 2D input (i.e., 2D image) be denoted by A, the 2D filter of size $m \times n$ be denoted by K, and the 2D feature map be denoted by F. Here, the image A is convolved with the filter K and produces the feature map F. This convolution operation is denoted by $A*K$ and is mathematically given as

$$F(i,j) = (A * K)(i,j) = \sum_m \sum_n A(m,n)K(i-m, j-n) \tag{2.1}$$

The convolution operation is commutative in nature, so we can write Eq. 2.1 as

$$F(i,j) = (A * K)(i,j) = \sum_m \sum_n A(i-m, j-n)K(m,n) \tag{2.2}$$

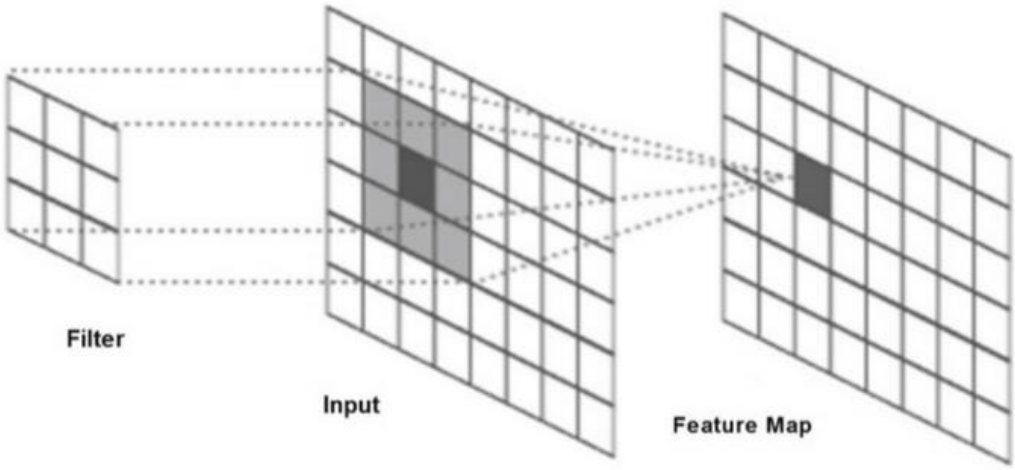


Fig. 2.4 Convolution operation

The kernel K is flipped relative to the input. If the kernel is not flipped, then convolution operation will be same as cross-correlation operation that is given below:

$$F(i, j) = (A * K)(i, j) = \sum_m \sum_n A(i + m, j + n) K(m, n) \qquad (2.3)$$

Many CNN libraries use cross-correlation function as convolution function because cross-correlation is more convenient to implement than convolution operation itself. According to Eq. 2.3, the operation computes the inner product (element-wise multiplication) of the filter at every location in the image.

Q4(b) Evolution of CNNs

LeNet: The first practical convolution-based architecture was LeNet which used backpropagation for training the network. LeNet was designed to classify handwritten digits (MNIST), and it was adopted to read large numbers of handwritten checks in the United States. Unfortunately, the approach did not get much success as it did not scale well to larger problems. The main reasons for this limitation were as follows:

a. Small labeled datasets.
b. Slow computers.
c. Use of wrong nonlinearity (activation) function.

The use of appropriate activation function in a neural network has huge impact on the final performance. Any deep neural network that uses a nonlinear activation function like sigmoid or tanh and is trained using backpropagation suffers from vanishing gradient. Vanishing gradient is a problem found in training the neural networks with gradient-based training methods. Vanishing gradient makes it hard to train and tune the parameters of the top layers in a neural network. The problem worsens as the total number of layers in the network increases.

AlexNet: The first breakthrough came in 2012 when the convolutional model which was named AlexNet significantly outperformed all other conventional methods in ImageNet

Large-Scale Visual Recognition Competition (ILSVRC) 2012 that featured the ImageNet dataset. The AlexNet brought down classification error rate from 26 to 15%, a significant improvement at that time. AlexNet was simple but much more efficient than LeNet. The improvements to overcome the above mentioned problems were due to the following reasons:

a. Large labeled image database (ImageNet), which contained around 15 million labeled images from a total of over 22,000 categories, was used.
b. The model was trained on high-speed GTX 580 GPUs for 5 to 6 days.
c. ReLU (Rectified Linear Unit) f (x) max(x, 0) activation function was used.

This activation function is several times faster than the conventional activation functions like sigmoid and tanh. The ReLU activation function does not experience the vanishing gradient problem.

AlexNet consists of five convolutional layers, three pooling layers, three fully connected layers, and a 1000-way softmax classifier.

ZFNet: In 2013, an improved version of CNN architecture called ZFNet was introduced. ZFNet reduced the filter size in the first layer from 11 × 11 to 7 × 7 and used a stride of 2 instead of 4 which resulted in more distinctive features and fewer

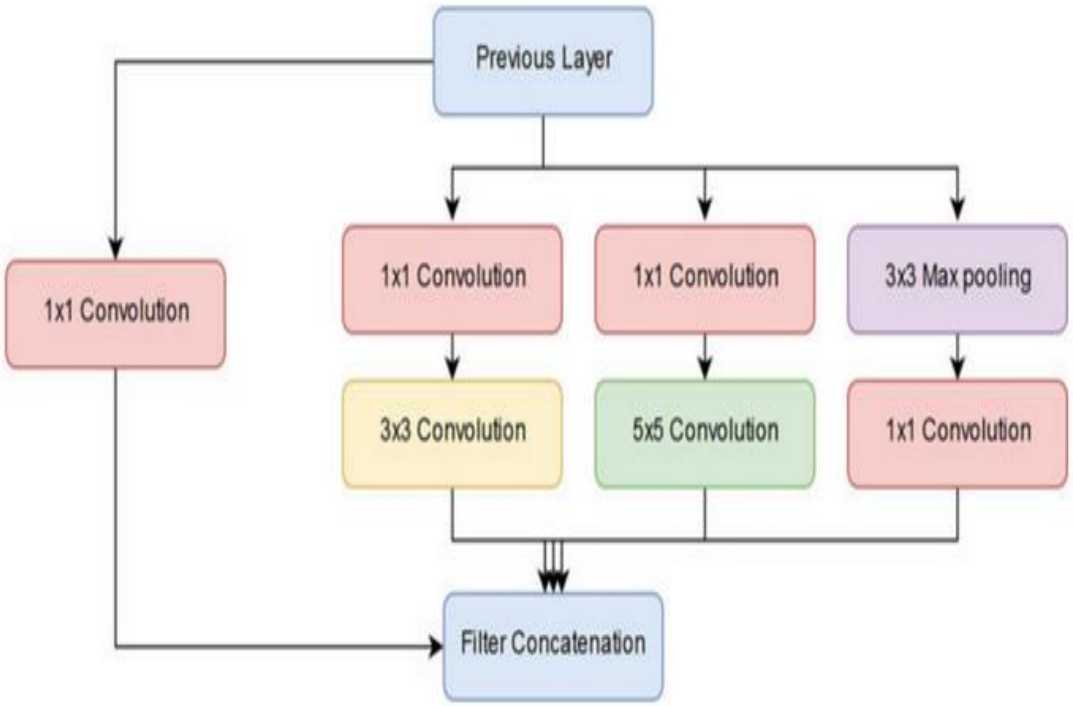


Fig. 2.1 Inception module in GoogLeNet

Table 2.1 Classification accuracy of AlexNet, VGG-16, ResNet-152, Inception and Xception on ImageNet

Model	Top-1 accuracy	Top-5 accuracy
AlexNet	0.625	0.86
VGG-16	0.715	0.901
Inception	0.782	0.941
ResNet-152	0.870	0.963
Xception	0.790	0.945

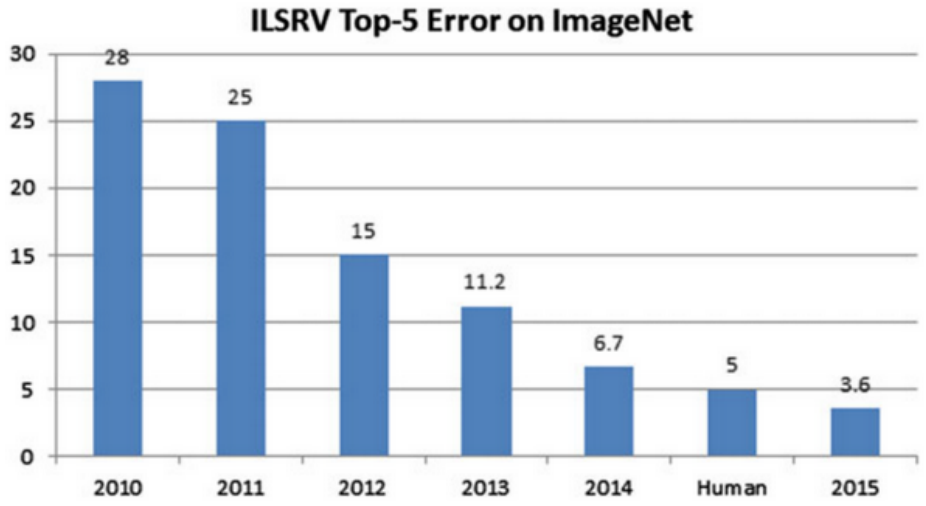


Fig. 2.3 ILSRV top-5 error on ImageNet since 2010

Module 3

Q5(a) CNN architecture with mathematical operations

Convolution is a mathematical operation performed on two functions and is written as $(f * g)$, where f and g are two functions. The output of the convolution operation for domain n is defined as

$$(f * g)(n) = \sum_m f(m)g(n-m)$$

For time-domain functions, n is replaced by t. The convolution operation is commutative in nature, so it can also be written as

$$(f * g)(n) = \sum_m f(n-m)g(m)$$

m

Convolution operation is one of the important operations used in digital signal processing and is used in many areas which includes statistics, probability, natural language processing, computer vision, and image processing.

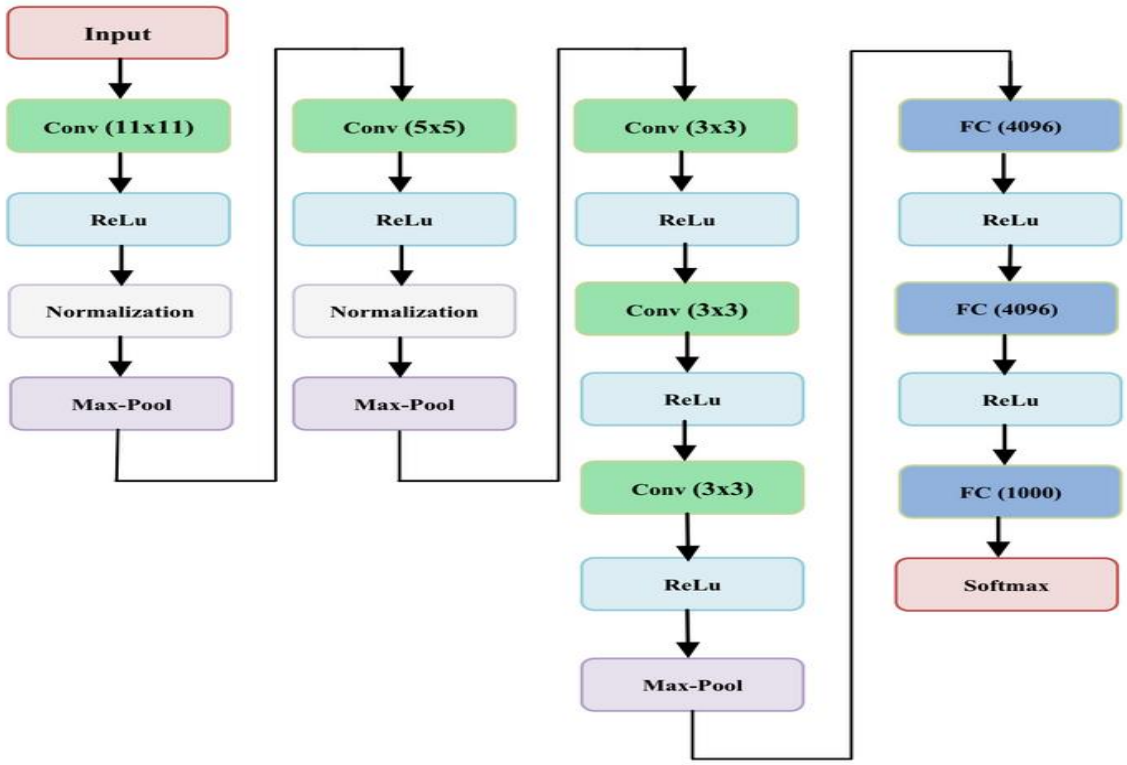
Convolution operation can be applied to higher dimensional functions as well. It can be applied to a two-dimensional function by sliding one function on top of another, multiplying and adding. Convolution operation can be applied to images to perform various transformations; here, images are treated as two-dimensional functions. An example of a two-dimensional filter, a two-dimensional input, and a two-dimensional feature map is shown in Fig. 2.4. Let the 2D input (i.e., 2D image) be denoted by A, the 2D filter of size $m \times n$ be denoted by K, and the 2D feature map be denoted by F. Here, the image A is convolved with the filter K and produces the feature map F. This convolution operation is denoted by $A*K$ and is mathematically given as

$$F(i, j) = (A * K)(i, j) = \sum_m \sum_n A(m, n) K(i - m, j - n) \tag{2.1}$$

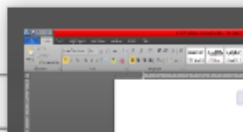
The convolution operation is commutative in nature, so we can write Eq. 2.1 as

$$F(i, j) = (A * K)(i, j) = \sum_m \sum_n A(i - m, j - n) K(m, n) \tag{2.2}$$

Q5(b) AlexNet architecture



name		size	size	Filters			size	Feature maps
Conv 1	224 × 224	11 × 11	–	96	4	1	55 × 55	96
Max-pooling 1	55 × 55	–	3 × 3	–	2	0	27 × 27	96
Conv 2	27 × 27	5 × 5	–	256	1	2	27 × 27	256
Max-pooling 2	27 × 27	–	3 × 3	–	2	0	13 × 13	256
Conv 3	13 × 13	3 × 3	–	384	1	1	13 × 13	384
Conv 4	13 × 13	3 × 3	–	384	1	1	13 × 13	384
Conv 5	13 × 13	3 × 3	–	256	1	1	13 × 13	256
Max-pooling 3	13 × 13	–	3 × 3	–	2	0	6 × 6	256
Fully con-nected 1	4096 neurons							
Fully con-nected 2	4096 neurons							
Fully con-nected 3	1000 neurons							
Softmax	1000 Classes							



Q6(a) LeNet-5 feature map transformation (32×32 input)

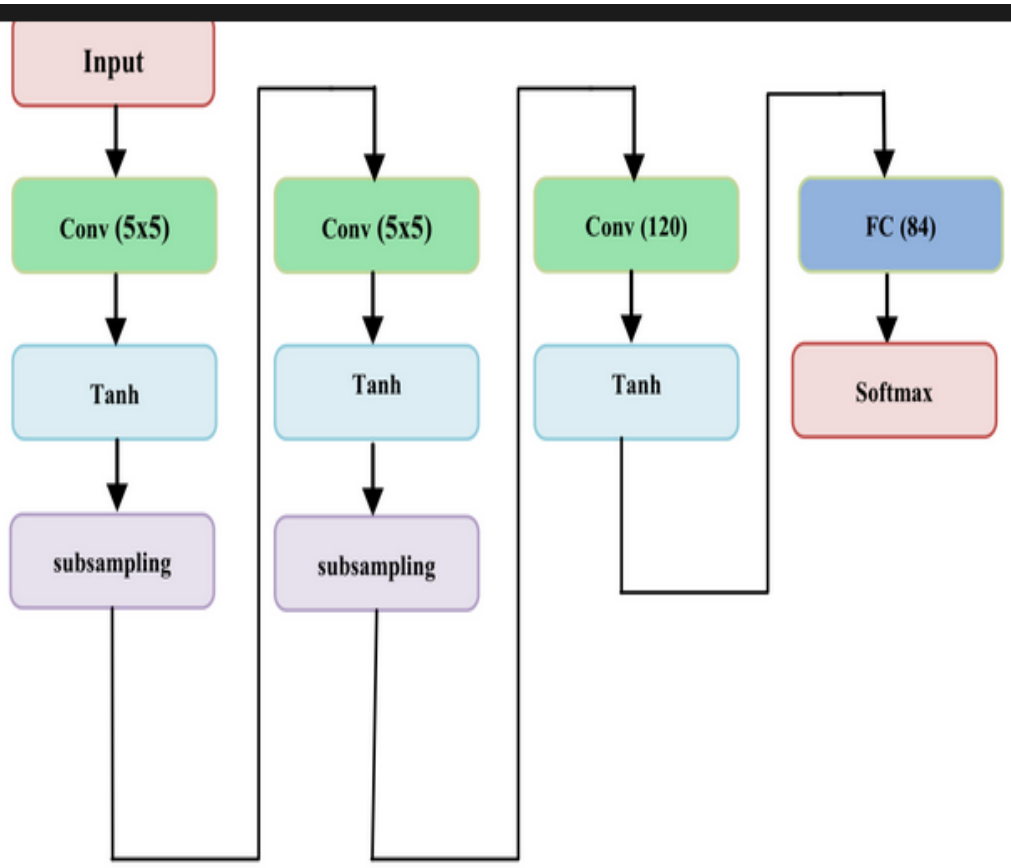


Fig. 4.1 Architecture diagram of LeNet-5

Table 4.1 Details of various layers of LeNet-5

Layer name	Input size	Filter size	Window size	# Filters	Stride	Padding	Output size	# Feature maps
Conv 1	32 × 32	5 × 5	–	6	1	0	28 × 28	6
Subsampling-1	28 × 28	–	2 × 2	–	2	0	14 × 14	6
Conv 2	14 × 14	5 × 5	–	16	1	0	10 × 10	16
Subsampling-2	10 × 10		2 × 2	16	2	0	5 × 5	16
Conv 3	5 × 5	5 × 5	–	120	1	0	1 × 1	120
Fully connected	120	–	–	–	–	–	1 × 1	84
Softmax	84	–	–	–	–	–	1 × 1	10

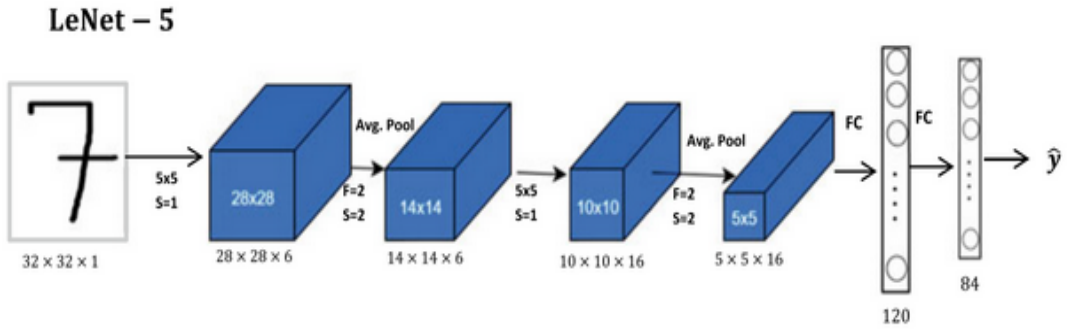


Fig. 4.2 Feature maps at various layers of LeNet-5

Training of LeNet-5:

Step 1: Random values are used for initialization of all filter parameters and weights.
Step 2: During this step, the input image goes through various layers, i.e., convolutional layers, subsampling layers, and fully connected layers. This step performs forward propagation which finds the output probabilities for all the classes in the network.
Step 3: The total error between output probabilities and target probabilities is calculated at the output layer of the network during this step.
Step 4: During this step, the error gradients with respect to weights in the network are calculated and gradient descent algorithm is used to update all weights and filter parameters to minimize the output error. The weights are adjusted proportionately depending on their contribution to the total error. Only the values of the connection weights and filter matrix are updated. The hyper-parameters like filter sizes and number of filters of the network are fixed and do not change during the training process.
Step 5: Steps 2–4 are repeated with all images present in the training set.

Q6(b) Gradient Descent Variants

3.4.1.1 Batch Gradient Descent (GD)

In traditional Gradient Descent (GD), also known as batch gradient descent, error gradient with respect to weight parameter w is computed for the entire training set followed by updating the weight parameter as shown below:

$$w = w - \mu \cdot \nabla \mathbf{E}(\mathbf{w})$$

where $\nabla \mathbf{E}(\mathbf{w})$ is the error gradient with respect to weight w and $\boldsymbol{\mu}$ is the learning rate that defines the step size to take along the gradient. The learning rate is a hyperparameter which cannot be too high or too low. Large value of learning rate can miss the optimum value, and too low learning rate will result in slow training time.

The training set often contains hundreds and thousands of examples that may demand huge memory which makes it difficult to fit in the memory. As a result, computing the error gradient can be very slow.

3.4.1.2 Stochastic Gradient Descent (SGD)

The above problem can be rectified by using Stochastic Gradient Descent (SGD), also known as incremental gradient descent, where gradient is computed for one training example at a time followed by updating of parameter values. It is usually much faster than standard gradient descent as it performs one update at a time.

$$w = w - \mu . \nabla \mathbf{E}(\mathbf{w}; \mathbf{x(i)}; \mathbf{y(i)})$$

where $\nabla \mathbf{E}(\mathbf{w}; \mathbf{x(i)}; \mathbf{y(i)})$ is the gradient of loss function—$E(w)$ w.r.t parameters w, for the training example $\{x(i), y(i)\}$.

In SGD, the one example based on updation of the parameter values causes the loss function to fluctuate frequently.

Mini-batch Gradient Descent Mini-batch gradient descent also known as mini-batch SGD is a combination of both standard gradient descent and SGD techniques. Mini-batch SGD divides the entire training set into mini-batches of n training examples and performs the updating of parameter values for each mini-batch. This type of gradient descent technique takes advantage of both standard gradient descent and SGD techniques, and is commonly used optimization technique in deep learning.

$$w = w - \mu \cdot \nabla \mathbf{E}(\mathbf{w}; \mathbf{x}(\mathbf{i} : \mathbf{i}+\mathbf{n}); \mathbf{y}(\mathbf{i} : \mathbf{i}+\mathbf{n}))$$

Typical mini-batch size varies from 50 to 256 and should also be chosen sensibly according to the following factors:

- Large batch sizes provide more accurate gradients but have high memory requirements.
- Small batch sizes can offer a regularizing effect but require a small learning rate to maintain stability owing to the high variance in the estimate of the gradient. This in turn increases the training time because of the reduced learning rate.

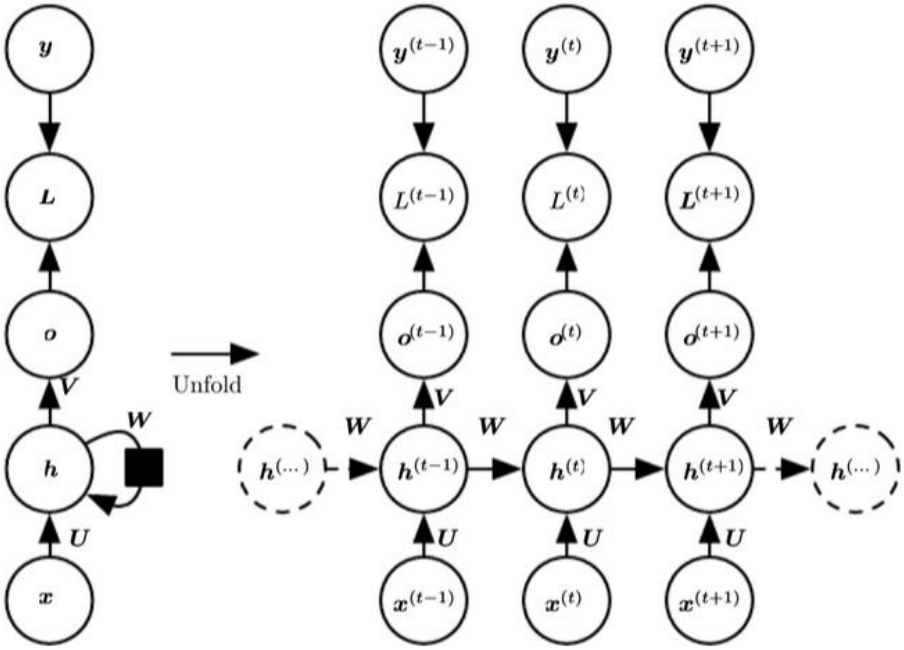
3.4.2 Improving Gradient Descent for Faster Convergence

The main objective of optimization is to minimize the cost/loss or objective function. There are many methods available that help an optimization algorithm to converge faster. Some of the commonly used methods are discussed below.

Module 4

Q7(a) Unfolding RNN

Armed with the graph unrolling and parameter sharing ideas of section 10.1, we can design a wide variety of recurrent neural networks.



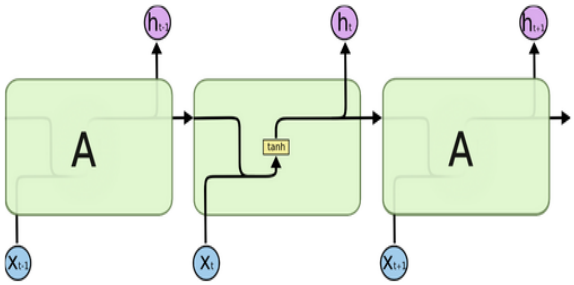
Recurrent networks that produce an output at each time step and have recurrent connections between hidden units, illustrated in figure

• Recurrent networks that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step,

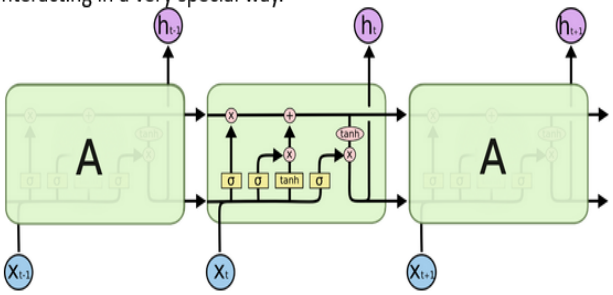
• Recurrent networks with recurrent connections between hidden units, that read an entire sequence and then produce a single output,

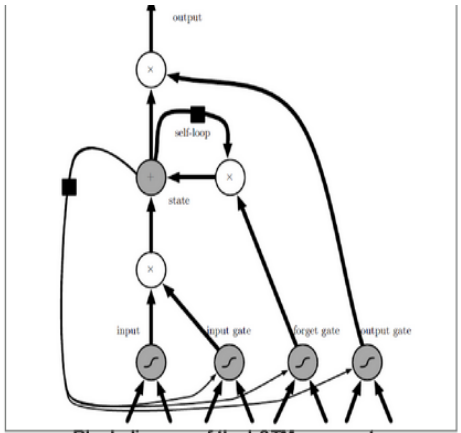
Q7(b) LSTM and GRU

- Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies.
- LSTMs are explicitly designed to avoid the long-term dependency problem.
- Remembering information for long periods of time is practically their default behavior.
- All recurrent neural networks have the form of a chain of repeating modules of neural network.
- In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



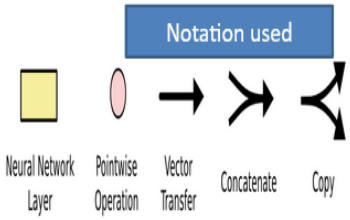
- LSTMs also have this chain like structure, but the repeating module has a different structure.
- Instead of having a single neural network layer, there are four, interacting in a very special way.





Block diagram of the LSTM recurrent network "cell."

In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.



Q8(a) Three parameter transformations in RNN

1. Input-to-Hidden Transformation

Transforms the **current input** into the hidden state space.

Mathematical Representation

WxhxtW_{xh} x_tWxhxt

Where:

- xtx_txt: input at time step ttt
- WxhW_{xh}Wxh: input-to-hidden weight matrix

Explanation

- Maps input features to hidden units
- Captures information from the **current observation**
- Helps encode raw input into a latent representation

Example

In speech recognition, this transformation converts the **current audio frame** into a hidden representation.

2. Hidden-to-Hidden (Recurrent) Transformation

Incorporates **past information** by transforming the previous hidden state.

Mathematical Representation

Whhht−1W_{hh} h_{t-1}Whhht−1

Where:

- ht−1h_{t-1}ht−1: hidden state at previous time step
- WhhW_{hh}Whh: recurrent weight matrix

Explanation

- Enables memory in RNNs
- Models temporal dependencies
- Same weights reused across all time steps
- Main reason RNNs can process sequences

Significance

- Responsible for **sequence modeling**
- Can cause **vanishing or exploding gradients** during training

3. Hidden-to-Output Transformation

Transforms the hidden state into the **network output**.

Mathematical Representation

WhyhtW_{hy} h_tWhyht

Where:

- hth_tht: current hidden state
- WhyW_{hy}Why: hidden-to-output weight matrix

Explanation

- Maps internal representation to output space
- Followed by activation functions such as:
  - Softmax (classification)
  - Linear (regression)

Example

In language modeling, this layer predicts the **next word** in the sequence.

Q8(b) Bidirectional RNN

Bidirectional RNNs

- All of 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, $x^{(1)}, \ldots, x^{(t-1)}$, and the present input $x^{(t)}$.
- Some of the models we have discussed also allow information from past y values to affect the current state when the y values are available.
- However, in many applications we want to output a prediction of $y^{(t)}$ which may depend on the whole input sequence. For example, in speech recognition, 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.
- Bidirectional recurrent neural networks (or bidirectional RNNs) were invented to address that need.
- They have been extremely successful in applications such as handwriting recognition, speech recognition and bioinformatics.
- **Bidirectional RNNs combine an RNN that moves forward through time beginning from the start of the sequence with another RNN that moves backward through time beginning from the end of the sequence.**

- Figure 10.11 illustrates the 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.*
- This idea can be naturally extended to 2-dimensional input, such as images, by having *four* RNNs, each one going in one of the four directions: up, down, left, right.
- At each point (i, j) of a 2-D grid, an output $O_{i,j}$ could then compute a representation that would capture mostly local information but could also depend on long-range inputs, if the RNN is able to learn to carry that information.

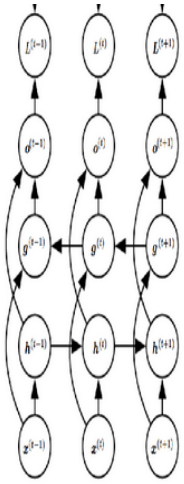


Figure 10.11: Computation of a typical bidirectional recurrent neural network, meant to learn to map input sequences $\boldsymbol{x}$ to target sequences $\boldsymbol{y}$, with loss $L^{(t)}$ at each step t. The $\boldsymbol{h}$ recurrence propagates information forward in time (towards the right) while the $\boldsymbol{g}$ recurrence propagates information backward in time (towards the left). Thus at each point t, the output units $\boldsymbol{o}^{(t)}$ can benefit from a relevant summary of the past in its $\boldsymbol{h}^{(t)}$ input and from a relevant summary of the future in its $\boldsymbol{g}^{(t)}$ input.

Module 5

Q9(a) Reinforcement Learning with example

Reinforcement Learning is a type of machine learning in which an **agent learns to make decisions by interacting with an environment**, taking actions, and receiving **rewards or penalties**.
The objective of the agent is to **learn an optimal policy** that maximizes the **cumulative reward** over time.

Key Components of Reinforcement Learning

1. **Agent** – Learner or decision maker
2. **Environment** – External system the agent interacts with
3. **State (S)** – Current situation of the agent
4. **Action (A)** – Choices available to the agent
5. **Reward (R)** – Feedback signal from the environment
6. **Policy (π)** – Strategy that maps states to actions
7. **Value Function** – Expected cumulative reward
8. **Episode** – Sequence of states, actions, and rewards until termination

Reinforcement Learning Process

1. Agent observes the current state StS_tSt
2. Agent selects an action AtA_tAt based on policy π\piπ
3. Environment returns reward RtR_tRt and next state St+1S_{t+1}St+1
4. Agent updates its policy to improve future rewards
5. Process repeats until goal is achieved

Mathematical Objective

The goal is to maximize the **expected return**:

Gt=∑k=0∞γkRt+k+1G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}Gt=k=0∑∞γkRt+k+1

Where:

- γ\gammaγ = discount factor (0≤γ≤10 \le \gamma \le 10≤γ≤1)

Q9(b) Traditional RL vs Deep RL

Aspect	**Traditional RL**	**Deep RL**
State space	Small	Large
Representation	Tabular	Neural networks
Example	Tic-Tac-Toe	Chess, Go

Q10(a) Mouse learning maze problem

Example 1: Reinforcement Learning in a Maze (Mouse and Cheese Problem)

Problem Description

A mouse must find cheese in a maze.

RL Formulation

RL Element	**Description**
Agent	Mouse
Environment	Maze
States	Mouse positions
Actions	Move up, down, left, right
Reward	+10 for cheese, −1 per step
Policy	Path selection strategy

Learning Behavior

- Initially, the mouse explores randomly
- Receives positive reward on reaching cheese
- Over time, it learns the **shortest path** to maximize reward