DE GRUYTER

# DEEP LEARNING

## RESEARCH AND APPLICATIONS

*Edited by Siddhartha Bhattacharyya, Vaclav Snasel, Aboul Ella Hassanien, Satadal Saha, B. K. Tripathy*

FRONTIERS IN COMPUTATIONAL INTELLIGENCE

Siddhartha Bhattacharyya, Vaclav Snasel, Aboul Ella Hassanien, Satadal Saha,
B. K. Tripathy (Eds.)

**Deep Learning**

# De Gruyter Frontiers in Computational Intelligence

Edited by
Siddhartha Bhattacharyya

## Volume 7

# Deep Learning

Research and Applications

Edited by
Siddhartha Bhattacharyya, Vaclav Snasel,
Aboul Ella Hassanien, Satadal Saha, B. K. Tripathy

DE GRUYTER

**Editors**

Prof. (Dr.) Siddhartha Bhattacharyya
CHRIST (Deemed to be University)
Hosur Road, Bhavani Nagar
S. G. Palya 560 029 Bangalore, India
dr.siddhartha.bhattacharyya@gmail.com

Dr. Satadal Saha
MCKV Institute of Engineering
G T Road North 243
711204 Liluah, West Bengal, India
satadalsaha@yahoo.com

Vaclav Snasel
VSB Technical University of Ostrava
17 Listopadu 2172/15
708 00 Ostrava, Czech Republic
vaclav.snasel@vsb.cz

B. K. Tripathy
VIT University
Near Katpadi Road
632014 Vellore, Tamil Nadu, India
tripathybk@rediffmail.com

Dr. Aboul Ella Hassanien
Cairo University
Information Technology Department
Ahmed Zewail
12613 Giza Governorate, Ad Doqi, Dokki, Egypt
aboitcairo@gmail.com

ISBN 978-3-11-067079-0
e-ISBN (PDF) 978-3-11-067090-5
e-ISBN (EPUB) 978-3-11-067092-9
ISSN 2512-8868

**Library of Congress Control Number: 2020938478**

**Bibliographic information published by the Deutsche Nationalbibliothek**
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available on the Internet at http://dnb.dnb.de.

Cover image: shulz/E+/getty images
Typesetting: Integra Software Services Pvt. Ltd.
Printing and binding: CPI books GmbH, Leck

www.degruyter.com

Siddhartha Bhattacharyya would like to dedicate this book to his late father Ajit Kumar Bhattacharyya, his late mother Hashi Bhattacharyya, his beloved wife Rashni, his colleagues Abhishek, Arpan, Pampa, Soham, and Srijibendu.

Vaclav Snasel would like to dedicate this book to his wife Božena Snášelová.

Aboul Ella Hassanien would like to dedicate this book to his wife Nazaha Hassan Elsaman.

Balakrushna Tripathy would like to dedicate this book to his beloved son Anurag.

Satadal Saha would like to dedicate this book to his little daughter Koushiki.

# Preface

Deep learning (DL) is a new area of machine learning (ML) research, which has been introduced with the objective of moving ML closer to one of its original goals, that is, artificial intelligence (AI). DL was developed as an ML approach to deal with complex input–output mappings. While traditional ML methods successfully solve problems where final value is a simple function of input data, DL techniques are able to capture composite relations between air pressure recordings and English words, millions of pixels and textual description, brand-related news and future stock prices, and almost all real-world problems. DL framework uses a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. The learning may be supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis). These algorithms learn multiple levels of representations that correspond to different levels of abstraction by using some form of gradient descent for training via backpropagation. The main components include multiple hidden layers of an artificial neural network and sets of propositional formulas. Other variations are also in vogue where latent variables are organized layerwise in deep generative models. Representative examples include deep belief networks and deep Boltzmann machines. DL is a state-of-the-art system in various disciplines, particularly computer vision, automatic speech recognition, and human action recognition.

This volume comprises seven well-versed chapters reporting latest trends in DL research and applications.

Chapter 1 presents the various cloud platforms available in market offerings from different vendors. IBM provided ML platform – Data Science Experience is considered here for the field of study. Prior to this section, an overview of AI, ML, and DL with their relationship is deliberated. Discussion on popular DL architectures with elementary comparison is also considered in this chapter.

Unlike conventional ML algorithms, where humans learn more of feature extraction than machines, which just crunches numbers, DL algorithms are quite promising as they have revolutionized the way data is dealt with. Based on the powerful notion of artificial neural networks, these learning algorithms learn to represent data or, if we rephrase, can extract features from raw data. DL has paved a way for end-to-end systems, where one learning algorithm does all the tasks. Convolutional neural networks (CNNs) have earned a lot of fame lately, especially in the domain of computer vision where in some cases its performance has beaten that of humans. In Chapter 2, the authors try to demystify how CNNs work and illustrate using one novel application in the field of astronomy. In addition, the chapter investigates what features the network is learning.

Optical character recognition systems have been used for extraction of text contained in scanned documents or images. This system consists of two steps: character detection and recognition. One classification algorithm is required for character

https://doi.org/10.1515/9783110670905-202

recognition by their features. Character can be recognized using neural networks. The multilayer perceptron (MLP) provides acceptable recognition accuracy for character classification. Moreover, the CNN and the recurrent neural network (RNN) are providing character recognition with high accuracy. MLP, RNN, and CNN may suffer from the large amount of computation in the training phase. In Chapter 3, a CNN is implemented for recognition of digits from MNIST database, and a comparative study is established between MLP, RNN, and CNN.

DL techniques have had a huge impact on AI research. They have improved on the traditional ML techniques where human expertise was required for feature engineering. By removing one human factor, they have moved us one step forward in the field of AI. They have not entirely removed humans though they are required for designing the architectures and cleaning the data. DL techniques have managed to achieve breakthrough results in domains such as speech recognition, machine translation, image recognition, and object detection. Chapter 4 gives a brief overview of various DL techniques being used today. Techniques that make DL more effective have been described. Some interesting applications have also been covered.

Handwritten document analysis using intelligent computing technology is a demanding research area nowadays considering its usefulness in identifying a person and human characteristics, particularly that of persons having typical disabilities such as dyslexia, dysgraphia, and Parkinson's disease. Analysis of handwriting, falling under the broad purview of graphology, helps us understand the writer's psychology, emotional outlays, and noticeable disorders as well. Since there prevails a broad spectrum of cursive nature and high inconsistency of handwriting styles, the techniques for modern handwriting analysis need to be more robust and sensitive to different patterns compared to the traditional graphological techniques. Herein lies the necessity of computing technology which intelligently analyzes handwritten texts to find out the similarity of finer aspects of handwritings of children or adult with some kind of learning/writing disability. DL technology is chosen as the technical tool to identify and classify common features of handwriting of children with developmental dysgraphia. Variational autoencoder, a deep unsupervised learning technique, is purported to be used here. Chapter 5 reports successful extraction of significant number of distinguishable handwriting characteristics that are clinically proved to be symptoms of dysgraphia. Audio signal processing and its classification dates back to the past century. From speech recognition to speaker recognition and from speech to text conversion to music generation, a lot of advances have been made in this field using algorithms such as hidden Markov models, RNNs with long–short-term memory layers, deep convolutional neural networks, and the recent state-of-the-art model for music and speech generation using WaveNets. These algorithms are applied after the audio signals are processed and effective feature extraction techniques are applied on them. Chapter 6 gives a detailed explanation about what an audio signal is and how it is processed. It also covers the various feature extraction techniques and the classification algorithms. Finally, the present-day applications and the potentials of DL in this field are explored.

In DL, data is transmitted through a number of layers in the feedforward network between input and output layers. In a recurrent network, data may propagate through a layer several times. Backpropagation through time (BPTT) technique is used to train recurrent networks (RNN). The underlying idea of BPTT is to transform a recurrent network into an unfolded feedforward network (multilayer network), where conventional backpropagation learning is used for gradient calculation. Here, each layer of the unfolded network represents a time step. The objective of Chapter 7 is to integrate the concept of BPTT in the framework of fuzzy time series prediction. The model takes a sequence of previous values as input (fuzzy inputs) to the different layer of the unfolded network and produces fuzzy output. Temperature dataset is used to evaluate the performance of the model, and prediction accuracy of BPTT is better than that of backpropagation neural network model.

This volume is intended for the students of computer science, electrical engineering, and information sciences of different universities to cater to a part of their curriculum. The editors would find this venture fruitful if this volume comes to use for the academic and scientific fraternity as well. The editors would like to take this opportunity to render their heartfelt gratitude to all the contributory authors and reviewers for their cooperation during the book project. Thanks are also due to Cambridge Scholar Publishing Ltd. for consenting to publish this volume.

India, Czech Republic and Egypt
September, 2019

Siddhartha Bhattacharyya
Vaclav Snasel
Aboul Ella Hassanien
Satadal Saha
B. K. Tripathy

# Contents

# List of Contributors

**Amit Adate**
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, Tamil Nadu, India
email2amitadate@gmail.com

**Dhruv Arya**
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, Tamil Nadu, India
aryadhruv@gmail.com

**Siddhartha Bhattacharyya**
CHRIST (Deemed to be University)
Bangalore, Karnataka, India
dr.siddhartha.bhattacharyya@gmail.com

**Ankita Bose**
VIT University, Vellore, Tamil Nadu, India
a1997bose@gmail.com

**Mahua Bose**
Department of Computer Science and Engineering, University of Kalyani
Nadia, West Bengal, India
e_cithi@yahoo.com

**Swagatam Das**
Indian Statistical Institute
Kolkata, West Bengal, India
swagatamdas19@yahoo.co.in

**Shatabhisa Dey**
RCC Institute of Information Technology
Kolkata, West Bengal, India
shatabhisa.97@gmail.com

**Soumyajit Goswami**
IBM India Private Limited, Salt Lake, Sector V
Kolkata 700091, West Bengal, India
soumyajit_goswami@yahoo.com

**Ranjan Jana**
RCC Institute of Information Technology
Kolkata, West Bengal, India
ranjan.rcciit@gmail.com

**Karan Maheshwari**
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, Tamil Nadu, India
karan@cognet.ai

**Kalyani Mali**
Department of Computer Science and Engineering, University of Kalyani
Nadia, West Bengal, India
kalyanimali1992@gmail.com

**Anirban Mukherjee**
RCC Institute of Information Technology
Kolkata, West Bengal, India
anirbanm.rcciit@gmail.com

**Rajib Saha**
RCC Institute of Information Technology
Kolkata, West Bengal, India
rajibsaha_4u@yahoo.co.in

**Aditya Shaha**
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, Tamil Nadu, India
aditya.shaha.p@gmail.com

**Rajkumar Rajasekaran**
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, Tamil Nadu, India
rrajkumar@vit.ac.in

**Avik Sarkar**
RCC Institute of Information Technology
Kolkata, West Bengal, India
avik.ron20@gmail.com

**B. K. Tripathy**
School of Information Technology and Engineering, Vellore Institute of Technology
Vellore, Tamil Nadu, India
tripathybk@rediffmail.com

https://doi.org/10.1515/9783110670905-204

Soumyajit Goswami

# 1 Deep Learning – A State-of-the-Art Approach to Artificial Intelligence

**Abstract:** This chapter presents various cloud platforms that are available in market offerings from different vendors. IBM provided a machine learning (ML) platform "IBM Watson Studio" (formerly "Data Science Experience"), and this is considered here for the field of study. An overview of artificial intelligence, ML, and deep learning (DL) with their relationship is deliberated. Discussion on popular DL architectures with elementary comparison is also considered.

**Keywords:** Artificial intelligence, machine learning, deep learning, IBM Watson Studio (formerly, Data Science Experience or DSX)

## 1.1 Introduction

Deep learning (DL), the subfield of artificial intelligence (AI), is the most promising area considered for research and industry. Although DL is a very modern topic, it is already being used by multiple technology giants to fulfill their needs. Few examples are voice and image recognition algorithms of Google [1]: Netflix and Amazon use it to decide [2] which video a person desires to watch or purchase in near future, upcoming forecast by MIT researchers [3], and Facebook uses it to predict future actions for advertisers [4]. UCLA researchers have manufactured an advanced microscope that produces a high-dimensional dataset used to train a DL network in identifying cancer cells in tissue samples [5]. In a nutshell, it has been used nowadays everywhere whenever automation comes into picture.

In the following section of this chapter, the relationship between DL, machine learning (ML), and AI has been discussed. A brief introduction to artificial neural network (ANN) with its classification and its different learning techniques has been specified in Section 1.3. As part of classification of ANN, feedforward neural networks (FFNNs) and recurrent neural networks (RNNs) with their uses have been discussed. While in the section of learning techniques, supervised, unsupervised, and reinforcement learning are briefly considered. Section 1.4 has been reserved to discuss about DL. It has been stated clearly in this section why the term "deep" has been used. Multiple points have been identified, which makes DL as state of the art. In Section 1.6, different activation functions such as sigmoid activation function, hyperbolic tangent activation function, rectified linear unit (ReLU) activation function, and

**Soumyajit Goswami,** IBM India Private Limited, Salt Lake, Sector V, Kolkata, West Bengal, India

https://doi.org/10.1515/9783110670905-001

softmax activation function are described in detail. There are many DL architectures available in literature. Few of them became very popular and offers high accuracy resulting in better performance. The concepts of restricted Boltzmann machine (RBM), deep belief network (DBN), autoencoder (AE), and convolutional neural network (CNN) are deliberated in this section. In Section 1.6, multiple ML platforms from different organizations have been furnished. All of them provide cloud infrastructures with high-performance graphics processing units (GPUs) to quicken the training of DL network with the huge volumes of data, which lessens the training time from weeks to hours. The last section of this chapter is dedicated for describing different steps of using IBM ML platform – IBM Watson Studio (formerly, Data Science Experience or DSX).

## 1.2 AI versus ML versus DL

AI is a subcategory of computer science that handles the simulation of intelligent activities in computers. AI is a computer system, which can accomplish responsibilities that usually need human acumen. Generally, a rule engine leads the AI system and a good AI system should have an intelligent rule engine, which is based on a series of meaningful IF–THEN statements. Since the 1950s, AI has been successfully used in visual perception, speech recognition, decision-making, and translation between languages. AI and ML are often used interchangeably, especially in the realm of big data.

As shown in Figure 1.1, DL is considered as a subcategory of ML and again ML is a subcategory of AI. In other words, all DL is ML, but not all ML is DL, and so forth.

ML, flourished since the 1980s, is a subset of AI but dynamic in nature and does not require human intervention to make certain changes. Symbolic logic is an example of AI but not considered as ML. As per the definition given by Tom Mitchel [6], "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." In ML programs, there is no need of following IF–THEN rule engine, rather they can adjust themselves in response to the data they are exposed to, which has been termed as learning. Learning can be of three types, as explained in the subsequent section.

As DL is the subset of both AI and ML, all the characteristics of AI and ML are propagated to DL inherently. It deals with the large amount of data applying on ANN. Statistics propose that the accrued volume of data will grow from 4.4 zettabytes to roughly 44 zettabytes in 2020. That is why sometimes it has been called as deep neural network.

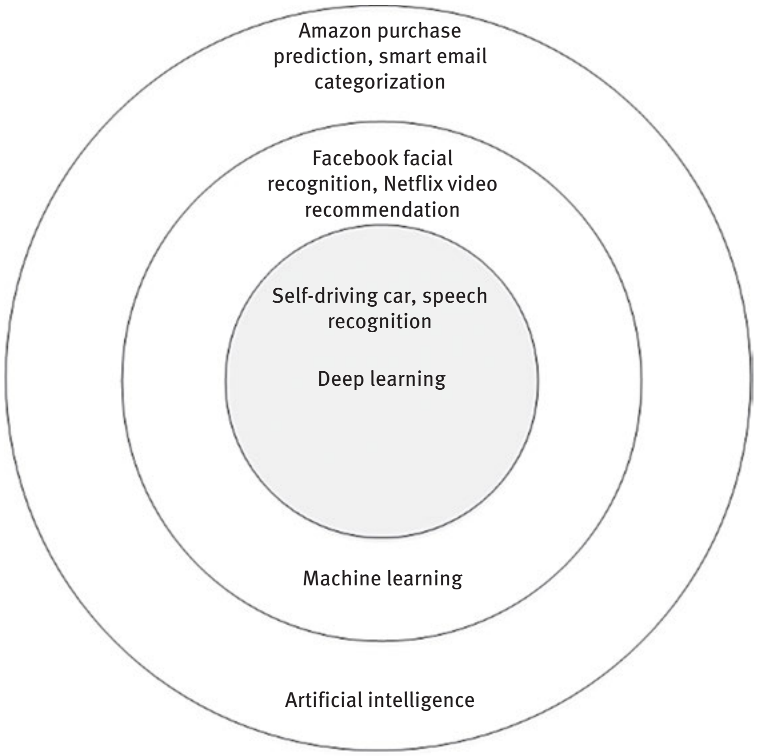

Amazon purchase prediction, smart email categorization
Facebook facial recognition, Netflix video recommendation
Self-driving car, speech recognition
Deep learning
Machine learning
Artificial intelligence

**Figure 1.1:** Relationship between AI, ML, and DL.

## 1.3 Artificial neural network

An ANN is an information processing system that has several performance features in common with biological neurons. An ANN is characterized by [7]
(a) its design pattern between the neurons, called architecture;
(b) its technique of determining the weights on the connections, called learning algorithm;
(c) its activation functions.

The artificial neuron as shown in Figure 1.2 is the basic building block or processing unit of an ANN.

### 1.3.1 Classification of ANN

Primarily, ANNs can be classified into two categories [7]: FFNNs and RNNs.

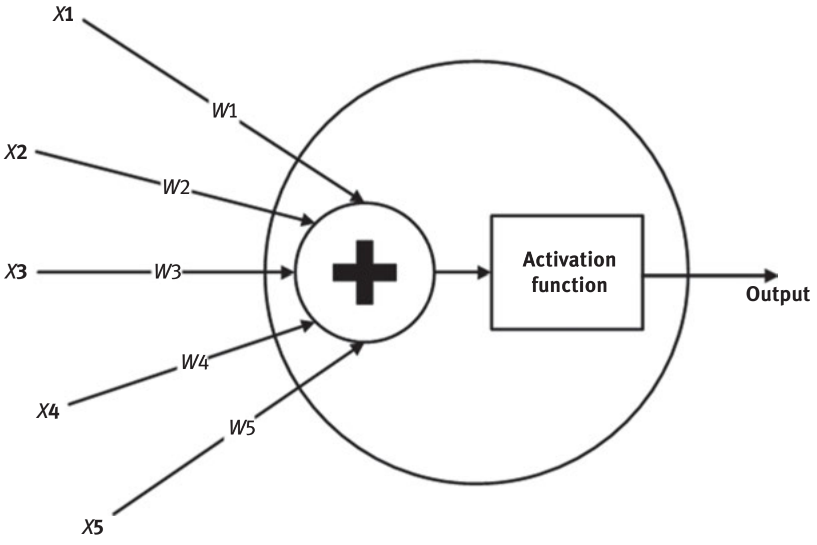


**Figure 1.2:** An artificial neuron.

#### 1.3.1.1 Feedforward neural network

The movement of signals in FFNN is in the forward pathway only. As there is no feedback loop, its behavior is independent of historical input and responds only to its current involvement. There are primarily two types of FFNN, as depicted below.

##### 1.3.1.1.1 Single-layer feedforward network

This is the simplest form of a layered network (Figure 1.3) which has only one input layer of source nodes that portrays onto an output layer of computation nodes. Its inputs are associated with the outputs through a sequence of single-layer weights. Adaptive linear neuron, Hopfield network, and learning vector quantization are common examples of single-layer feedforward network.

##### 1.3.1.1.2 Multilayer feedforward network

This type of feedforward network has one input layer, one output layer, and one or more hidden layers. By connecting these neurons as shown in Figure 1.4, the network is empowered to evaluate higher order information.

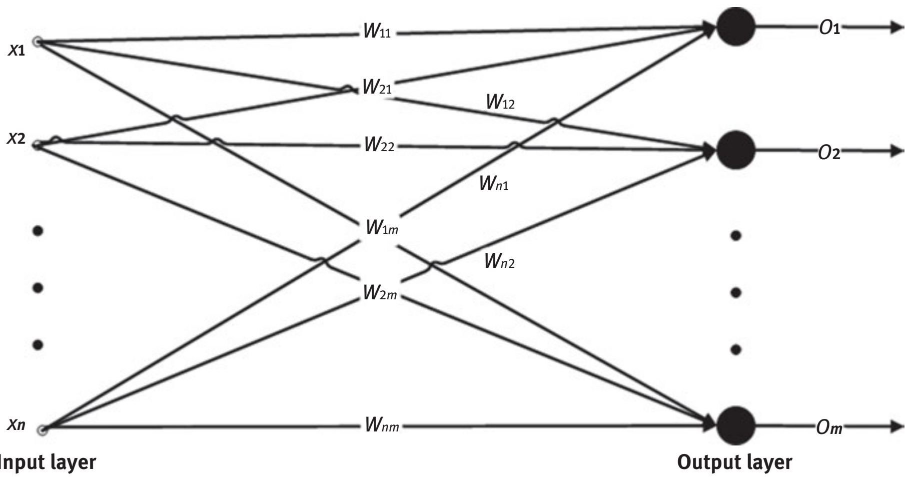


**Figure 1.3:** Single-layer feedforward network.

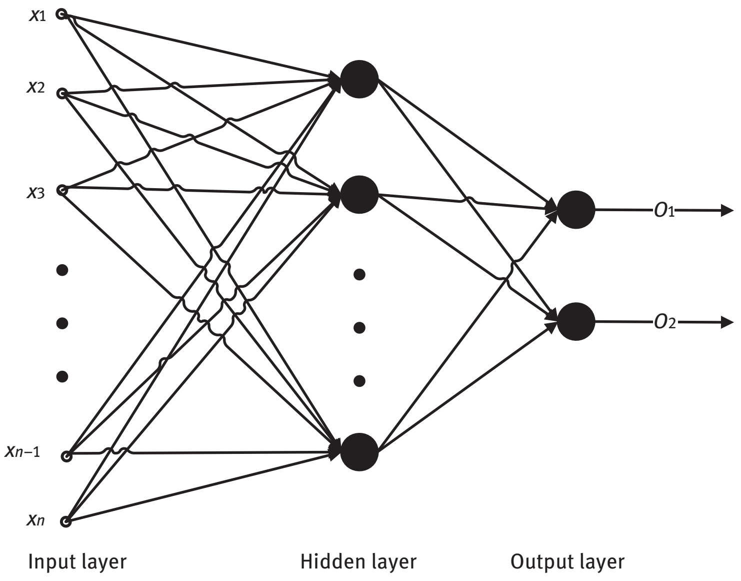


**Figure 1.4:** Multilayer feedforward network.

### 1.3.1.2 Recurrent neural network

In the architecture of an RNN (Figure 1.5), it should have minimum one feedback loop (can also be self-feedback). There are numerous varieties of recurrent networks based on the pattern of the feedback. In a specific situation, the network can have a single layer of neurons with each neuron feeding its output signal back to the inputs of all other neurons. Other types of recurrent networks may have self-feedback loops with hidden neurons. Hopfield network, bidirectional associative memory, brain state in a box, and adaptive resonance theory are few examples of RNN.

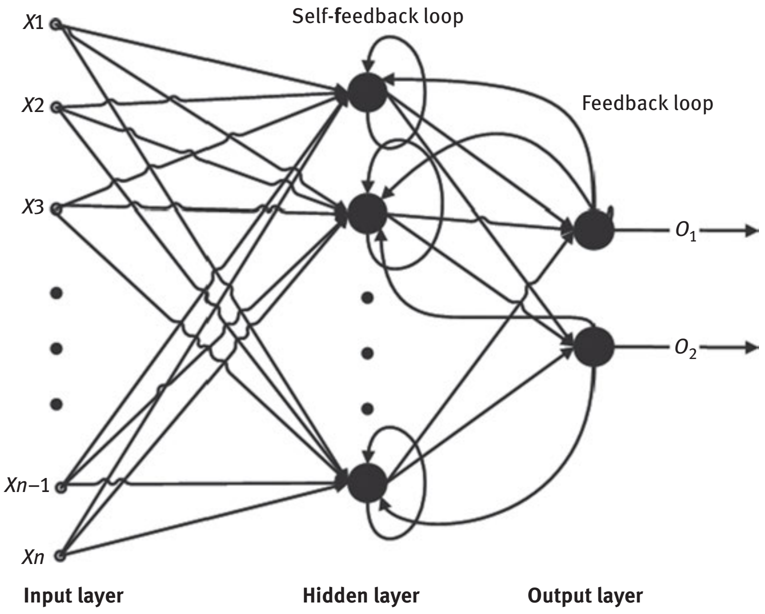


**Figure 1.5:** Recurrent network.

## 1.3.2 ANN learning

To acquire knowledge from environment and to improve the network's performance through learning is the most significant property of a neural network (NN). This process indicates the subsequent order of actions [7]:

- Stimulation by an environment
- Changing free parameters as an outcome of stimulation
- Responding in a better means to the environment

#### 1.3.2.1 Unsupervised learning

In unsupervised learning (Figure 1.6), no external trainer is required to accomplish the learning procedure. Inputs are provided to the network but without the desired output(s). The network must decide itself what characteristics are used to assemble the input data. Competitive learning rule is one of the ways to perform unsupervised learning. If an NN has two layers – an input layer and a competitive layer, the input layer neurons collect the feed data, whereas the competitive layer consists of neurons participating among themselves and only the winners of the competition survive and the other neurons switch off according to the "winner-takes-all" approach. Another type of unsupervised learning is Hebbian, that is, correlate weight adjustment.

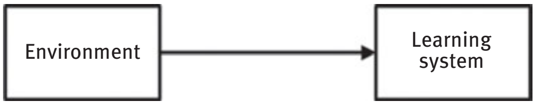


**Figure 1.6:** Block diagram of unsupervised learning.

### 1.3.2.2 Supervised learning

If the desired output of an NN is already known and is feed to the network, the learning method of the network is called supervised learning. The network then processes the inputs and relates its resulting outputs against the intended outputs. Differential errors are then calculated, producing the system to regulate the weights to control the network. This method happens repeatedly as the weights are continuously being fine-tuned.

Supervised learning technique (Figure 1.7) is of two types – stochastic (weight adjustment by probability) and through error correction. Error correction learning rule is of two types, as stated below.

– **Delta rule** – It was designed by Widrow and Hoff in 1960 and is also known as the Widrow and Hoff learning rule or the least mean square rule. The delta rule uses the difference between target activation (i.e., target output values) and obtained activation to undergo learning. The values of the weights are fine-tuned to minimize the difference between target and actual output activation (i.e., error).
– **Gradient descent rule** – The prime characteristic of gradient descent is to gradually but consistently reduce the output error by regulating the weights. If a change in a weight ($\delta w$) increases (decreases) the error ($\delta E$), then the respective weight decreases (increases) and can be represented as follows:

$$\Delta w_{ij} = -\eta \frac{\delta E_i}{\delta w_{ij}} \tag{1.1}$$

The distance, $\eta$, is a standard parameter in NNs and is termed learning rate.

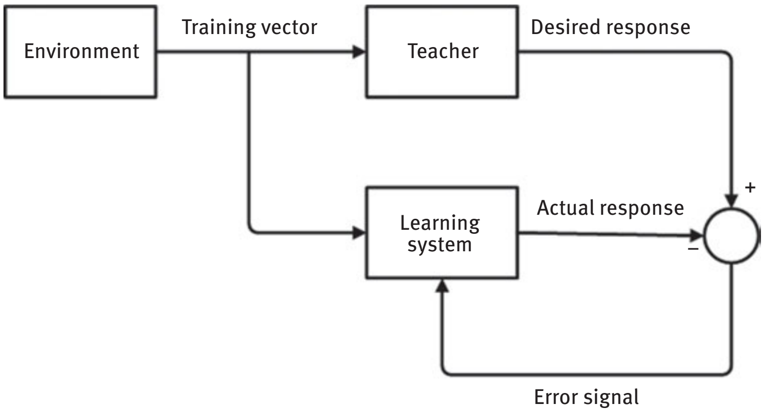


**Figure 1.7:** Block diagram of supervised learning.

#### 1.3.2.3 Reinforcement learning

In reinforcement learning, the learning of the network is achieved through constant collaboration with the situation or environment. Figure 1.8 shows the block diagram of a typical reinforcement learning network constructed around a trainer that transforms a reinforcement signal collected from environment into an advanced signal termed as the heuristic reinforcement signal.

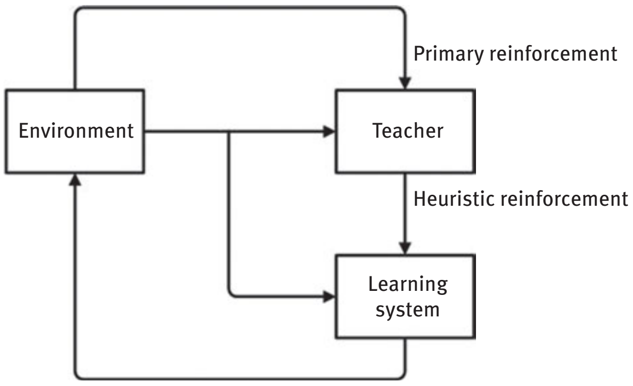


**Figure 1.8:** Block diagram of reinforcement learning.

## 1.4 Deep learning

DL is a category of ML in which a network or model learns directly from dataset [8]. The word “deep” denotes the number of layers in the network – the deeper network consists of many layers. Traditional NNs comprise only two or three layers, whereas deep networks can have hundreds of such layers as shown in Figure 1.9. There is no specific definition beyond how many layers we can categorize the network as "deep" [9]. Learning techniques are similar to ML and ANN, as depicted earlier. DL is particularly suitable to identify modern smart applications [10] such as face recognition, text translation, voice recognition, and advanced driver assistance systems including lane classification and traffic sign recognition.

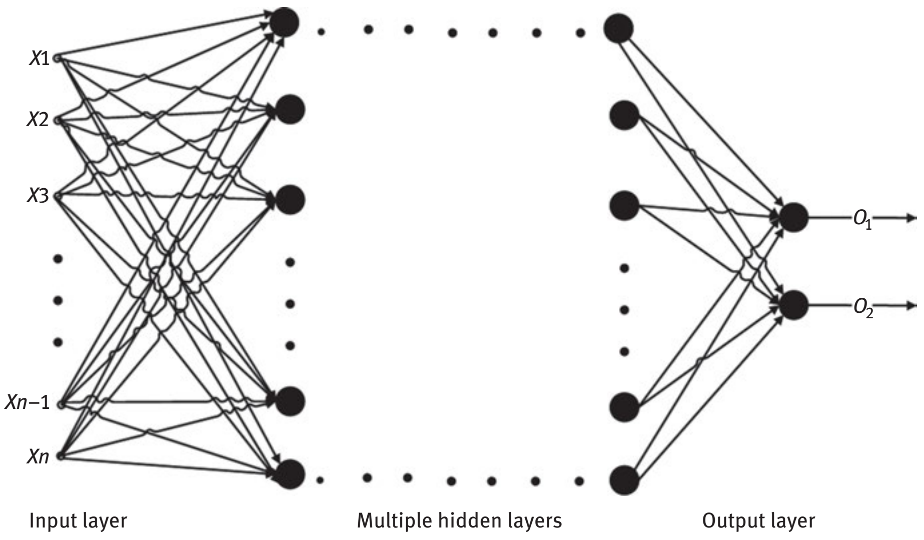


**Figure 1.9:** Block diagram of deep learning network.

## 1.4.1 What makes DL state of the art?

- Advanced tools and techniques have deeply improved the accuracy of DL algorithms. By comparing with ML in the same problem with same dataset, it has been shown that DL minimizes the % of error drastically.
- DL can deal with massive dataset, which purposefully improvises the training of the network. In today's world, there is no scarcity of data. As shown in Figure 1.10, DL performs far better than ML at the presence of huge amount of data.
- High-performance GPUs quicken the training of DL network with the huge volumes of data, which lessens the training time from weeks to hours. GPUs are available in any DL cloud platform.

Many standard models, developed and agreed by experts, are available in cloud platforms to reuse in any DL network for any purpose.

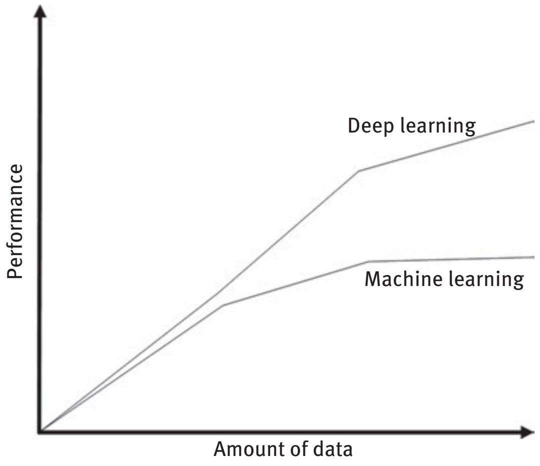


**Figure 1.10:** Performance comparison between ML and DL.

### 1.4.2 Activation functions

Activation function or transfer function [11, 12] converts an input signal of a node of an NN to an output signal, which is considered as an input to the next layer of the network. It introduces nonlinearity into the network, which makes the network more powerful and adds ability to learn something complex and complicated form data. So, it is very significant to choose the appropriate activation function for any network because it can considerably change the behavior of the network. The popular activation functions are described below. In all equations, $f'(x)$ is considered as the derivative of $f(x)$, where $x$ is the input signal.

#### 1.4.2.1 Sigmoid activation function

Sigmoid function is defined as

$$\begin{aligned} f(x) &= \frac{1}{1+e^{-x}} \\ f'(x) &= f(x)(1-f(x)) \end{aligned} \tag{1.2}$$

The sigmoid function curve looks like an S-shape, as illustrated in Figure 1.11. This function exists between 0 and 1. We can find the slope of the sigmoid curve at any two points; hence, it is differentiable.

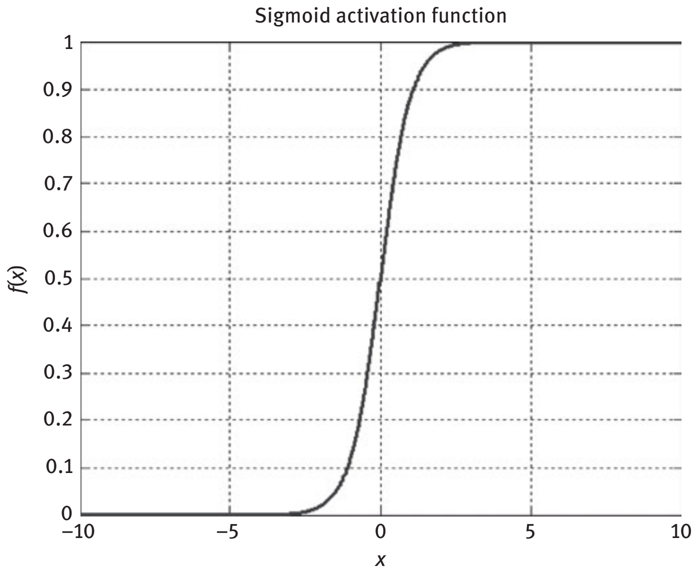


**Figure 1.11:** Sigmoid activation function.

#### 1.4.2.2 Hyperbolic tangent activation function

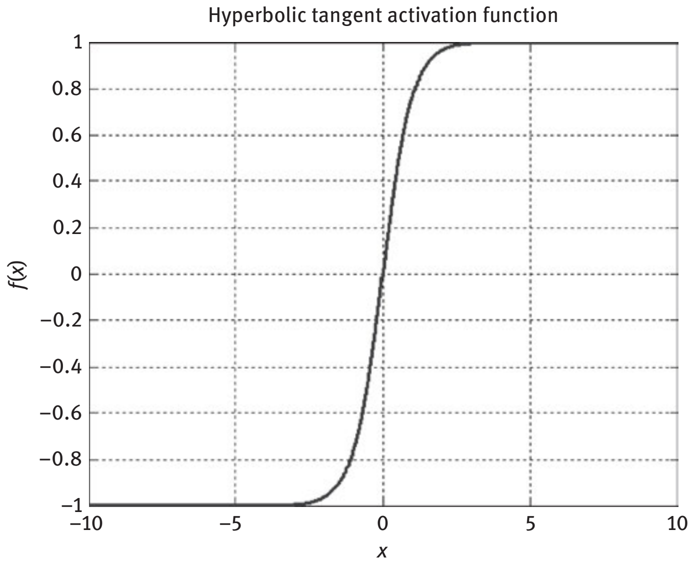


**Figure 1.12:** “tanh” activation function.

Hyperbolic tangent function is defined as

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$
$$f'(x) = 1 - f(x)^2 \tag{1.3}$$

As shown in Figure 1.12, "tanh" function also looks like an S-shape. This function exists between –1 and 1. It is differentiable too.

#### 1.4.2.3 ReLU activation function

At present, ReLU is the furthermost used activation function. It is shown in Figure 1.13 and defined as

$$\begin{aligned} f(x) &= 0, \quad \text{for } x < 0 \\ x&, \quad \text{otherwise} \\ f'(x) &= 0, \quad \text{for } x < 0 \\ 1&, \quad \text{otherwise} \end{aligned} \tag{1.4}$$

This function exists between 0 and infinity. It is differentiable too.

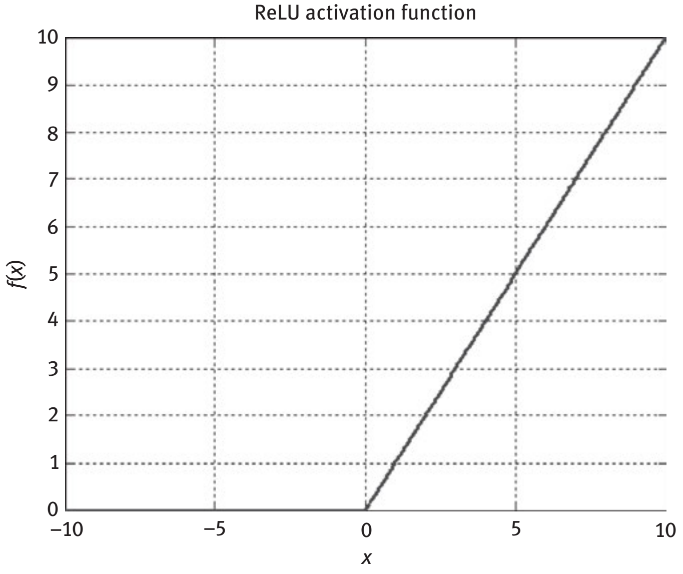


**Figure 1.13:** Rectified linear unit activation function.

#### 1.4.2.4 Softmax activation function

Softmax function is a type of sigmoid function, which can handle multiple classes in the classification problem. The output of the softmax function is equivalent to a probability distribution. It can be defined as

$$f(x)|_{j=1}^{k} = \frac{e^{x_j}}{\sum_{k=1}^{k} e^{x_k}} \tag{1.5}$$

### 1.4.3 Popular DL architectures

There are many DL architectures available in the literature. Few of them become very popular and offer high accuracy with better performance [12].

#### 1.4.3.1 Restricted Boltzmann machine

RBMs are extensively used in DL networks because of their historical importance and comparative straightforwardness. It was invented by Geoff Hinton and is used for dimensionality reduction, classification, regression, collaborative filtering, feature learning, and topic modeling. Boltzmann machines (BMs) are the primary building blocks of RBMs. BMs can be designed as NNs with bidirectionally associated stochastic processing components. RBMs are shallow, two-layer NNs that establish the DBN. As shown in Figure 1.14, It has just two layers – visible layer (or input layer) and hidden layer. Each node of visible layer is connected to each of the hidden layer, but there is no connection between any node of any particular layer. This is the restriction in RBM network.

In recent years, few modified RBMs introduced by different researchers are discriminative RBM, conditional RBM, and temperature-based RBM.

#### 1.4.3.2 Deep belief network

DBN also falls under the same stack of RBM, because here also the nodes of any single layer do not interconnect. DBNs, constructed by Hinton in 2006, are used to recognize, cluster, and generate images, video sequences, and motion-capture data.

To maximize the flexibility of DBNs, a novel prototype of convolutional DBN was presented by Arel in 2010. It can extract the features of high-dimensional images. The structure of DBN has been demonstrated in Figure 1.15.

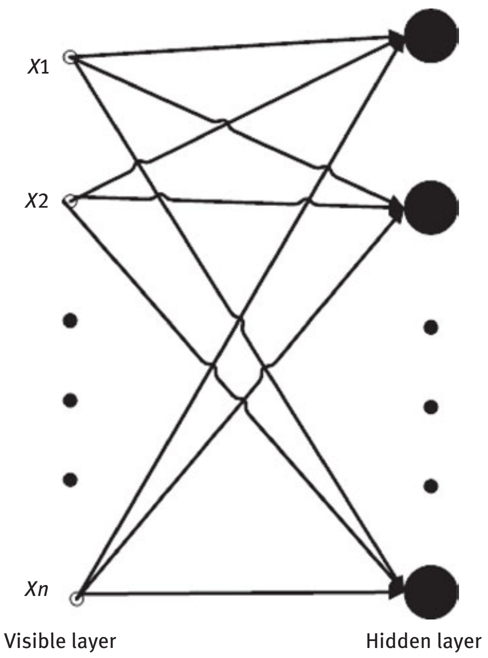


Figure 1.14: Structure of RBM.

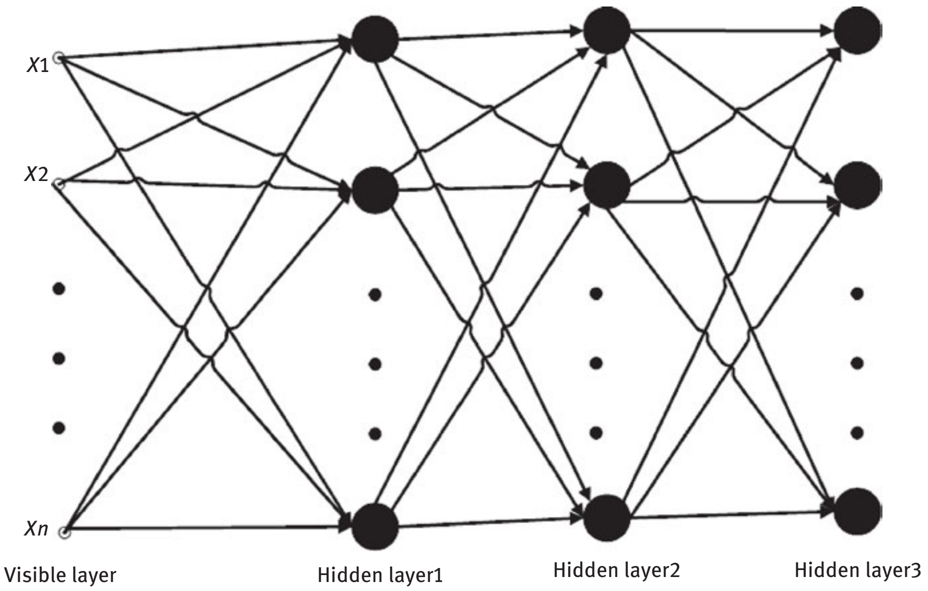


Figure 1.15: Structure of deep belief network.

### 1.4.3.3 Autoencoder

An AE is comprised of two symmetrical DBNs, as illustrated in Figure 1.16. The first DBN does the encoding operation and the second one does the decoding. It has been successfully used in image search, data compression, information retrieval, and so on.

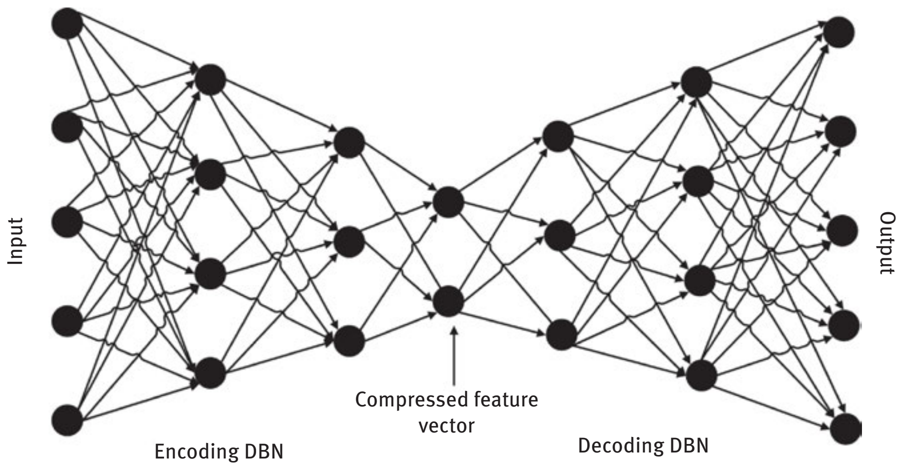


**Figure 1.16:** Structure of autoencoder.

As a practical example, 28 × 28 pixel (i.e., 784 pixels) image dataset can be taken as input to an AE. The first layer of the encoding-half is generally kept a slightly larger set of parameters. In essence, the encoder will contain the below set of parameters:

784 (input) ----> 1000 ----> 500 ----> 250 ----> 100 ----> 30 (compressed vector)

Similarly, the decoder will contain the below set of parameters:

30 (compressed vector) ----> 100 ----> 250 ----> 500 ----> 1000 ----> 784 (output)

To improve the ability of AE, there exist various types of AE, such as denoising autoencoder, sparse autoencoder, variational autoencoder, and contractive autoencoder.

#### 1.4.3.4 Convolutional neural network

The convolution operator of two functions $f(t)$and $g(t)$can be defined as follows:

$$f(t)^*g(t) \overset{\Delta}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau \tag{1.6}$$

where $\tau$ is a dummy variable representing time.

This mathematical operation has been used in CNNs and has suitable performance in processing 2D data with grid-like topology. The primary use of CNN is in the areas of image classification, clustering, object recognition, behavior recognition, sound representation over spectrogram, and natural language processing on hand-written documents. Image recognition functionality of CNN is majorly being applied in self-driving cars, robotics, drones, security, medical diagnoses, and treatments for the visually impaired.

Convolutional nets analyze images in a different way than RBMs. While RBMs acquire to recreate and recognize the features of each image entirely, convolutional nets analyze images in pieces. A convolutional network receives a normal color image as a rectangular box whose width and height are measured by the number of pixels and whose depth is three layers deep, one for each letter in red–green–blue. In other words, a convolutional net treats an image as a 3D object.

There are few modified versions of CNN available in the literature, which was shown to have better performance while comparing with conventional CNN. Combination of RBM and CNN, termed as convolutional RBMs, provides an advanced convergence rate with a reduced value of the negative likelihood function.

## 1.5 ML platforms of multiple organizations

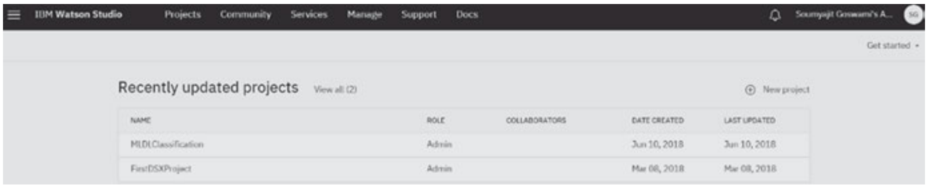
| Company | ML platform | URL |
|---|---|---|
| **IBM** | IBM Watson Studio (formerly, Data Science Experience) | https://www.ibm.com/cloud/watson-studio |
| **Amazon** | SageMaker | https://aws.amazon.com/sagemaker/ |
| **Microsoft** | Azure Machine Learning Studio | https://studio.azureml.net/ |
| **SKIL** | Skymind Intelligence Layer | https://docs.skymind.ai/docs |
| **Google** | Cloud AI | https://cloud.google.com/products/machine-learning/ |
| **Cloudera** | Data Science Workbench | https://www.cloudera.com/products/data-science-and-engineering/data-science-workbench.html |
| **Databricks** | MLlib | https://docs.databricks.com/spark/latest/mllib/index.html |
| **Dataiku** | Collaborative Data Science Platform | https://www.dataiku.com/ |

## 1.6 IBM Watson Studio (formerly, Data Science Experience)

In 2016, IBM launched DSX [13], which is modified as Watson Studio, to address each stage of the data science lifecycle, offering notebooks, collaboration spaces, tutorials, and ML models by provisioning Spark, R, Python, Scala, and other ML languages. DSX also helps data scientists by providing support to their familiar tools such as Jupyter, RStudio, Notebooks, and Apache Spark to create complex ML models and deploy these models to production. Figure 1.17 shows few snapshots from IBM Watson Studio on projects and services subsections. A project is used to organize resources to achieve a precise goal. It can include data, collaborators, and analytical assets like notebooks and models. An admin can associate any number of services with the project. The types of services that can be associated are

- IBM Apache Spark compute service
- Amazon Elastic MapReduce compute service
- IBM Analytics Engine compute service
- Cognos Dashboard Embedded
- IBM Watson Machine Learning
- IBM Streaming Analytics
- Watson Natural Language Classifier
- Visual Recognition
- Watson Assistant
- Watson Discovery
- Watson Knowledge Studio
- Watson Language Translator
- Natural Language Understanding
- Watson Personality Insights
- Watson Speech to Text
- Watson Text to Speech
- Watson Tone Analyzer

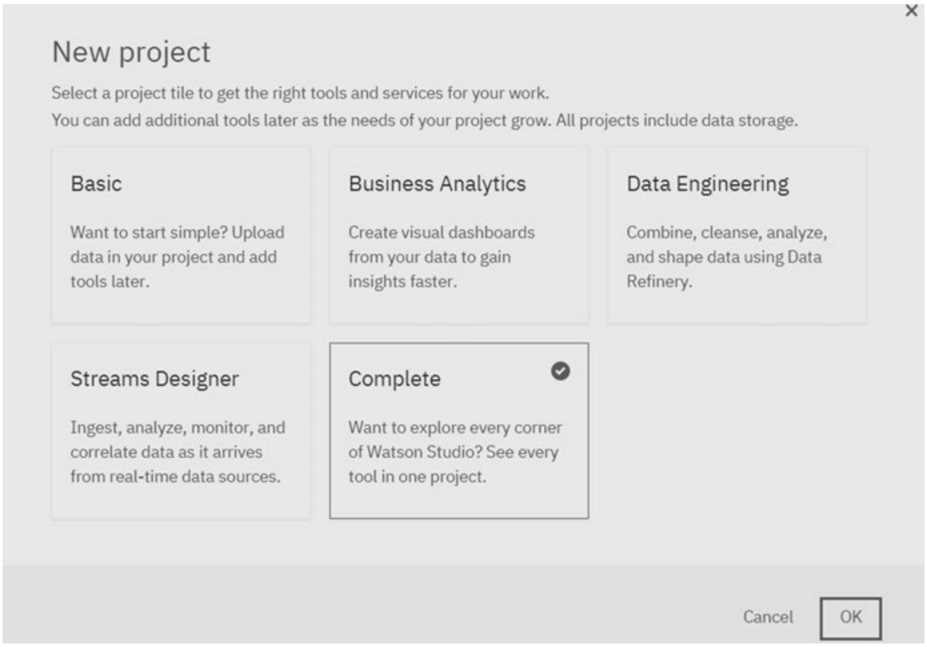
Once a project is created, an admin can add data assets to it for working with that data. Data assets can be added from local files, community, and database connections. All the collaborators in the project are automatically authorized to access the data in the project. After adding raw data into project, data needs to be refined, which consists of cleansing and shaping. Cleansing can be done by removing data that is incorrect, incomplete, improperly formatted, or duplicated. Whereas shaping data can be done by filtering, sorting, combining or removing columns, and performing operations. The refined data is ready for analyzing and visualizing [14].



Recently updated projects

| NAME | ROLE | COLLABORATORS | DATE CREATED | LAST UPDATED |
|---|---|---|---|---|
| MLDLClassification | Admin | | Jun 10, 2018 | Jun 10, 2018 |
| FirstDSXProject | Admin | | Mar 08, 2018 | Mar 08, 2018 |

(a) Projects (both for new and existing) Subsection



New project

Select a project tile to get the right tools and services for your work.
You can add additional tools later as the needs of your project grow. All projects include data storage.

Basic

Want to start simple? Upload data in your project and add tools later.

Business Analytics

Create visual dashboards from your data to gain insights faster.

Data Engineering

Combine, cleanse, analyze, and shape data using Data Refinery.

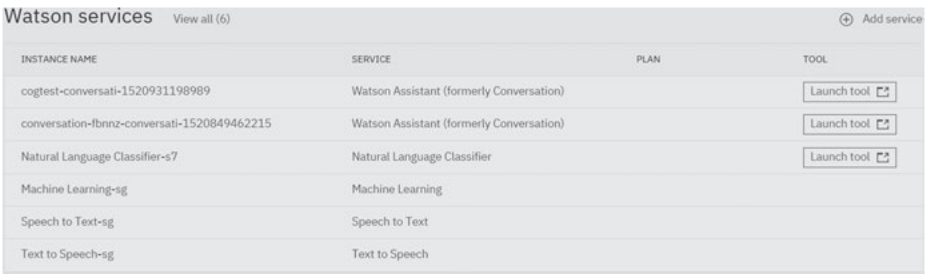
Streams Designer

Ingest, analyze, monitor, and correlate data as it arrives from real-time data sources.

Complete ✓

Want to explore every corner of Watson Studio? See every tool in one project.

(b) Creating New Project



Watson services

| INSTANCE NAME | SERVICE | PLAN | TOOL |
|---|---|---|---|
| cogtest-conversati-1520931198989 | Watson Assistant (formerly Conversation) | | Launch tool |
| conversation-fbnnz-conversati-1520849462215 | Watson Assistant (formerly Conversation) | | Launch tool |
| Natural Language Classifier-s7 | Natural Language Classifier | | Launch tool |
| Machine Learning-sg | Machine Learning | | |
| Speech to Text-sg | Speech to Text | | |
| Text to Speech-sg | Text to Speech | | |

(c) Services (both for new and existing) Subsection

**Figure 1.17:** IBM Watson Studio subsections. (a) Projects (both for new and existing) subsection, (b) creating new project, and (c) services (both for new and existing) subsection.

## 1.7 Conclusion

In this chapter, an overview with their relationship between AI, ML, and DL has been specified. At the same time, the reason for considering DL as state of the art in modern era has been discussed. The main focus is to highlight different cloud platforms available in the market by different vendors. In Section 1.6, different steps of using ML with IBM Watson Studio – have been described, including data set management and cleansing raw data. Comparative discussion in terms of performance, accuracy, 24 × 7 availability, and user-friendliness between the ML platform offerings by multiple organizations can be considered for future study.

## References

[1] https://www.forbes.com/sites/bernardmarr/2017/08/08/the-amazing-ways-how-google-uses-deep-learning-ai/#509e1a7b3204
[2] https://www.techtimes.com/articles/3442/20140213/netflix-deep-learning-based-ai-will-tell-you-what-you-want-to-watch-but-do-we-need-one.htm
[3] https://www.digitaltrends.com/computing/mit-future-video/
[4] https://theintercept.com/2018/04/13/facebook-advertising-data-artificial-intelligence-ai/
[5] https://www.systematics.co.il/wp-content/uploads/Deep_Learning_ebook.pdf
[6] https://www.cs.ubbcluj.ro/~gabis/ml/ml-books/McGrawHill%20-%20Machine%20Learning%20-Tom%20Mitchell.pdf
[7] Haykin Simon S. Neural networks and learning machines. Pearson, Upper Saddle River, 2009, 3.
[8] Deng L. A tutorial survey of architectures, algorithms, and applications for deep learning. APSIPA transactions on signal and information processing, 2014, 3.
[9] Deng L., Dong Y. Deep learning: Methods and applications. Foundations and Trends® in Signal Processing, 2014, 7, 3–4, 197–387.
[10] Liu W., Zidong W., Xiaohui L., Nianyin Z., Yurong L., & Fuad E. A. A survey of deep neural network architectures and their applications. Neurocomputing, 2017, 234, 11–26.
[11] Schmidhuber J. Deep learning in neural networks: An overview. Neural networks, 2015,61, 85–117.
[12] Bishop Ch. M. Pattern recognition and machine learning (information science and statistics). 2006.
[13] https://hortonworks.com/blog/exciting-data-science-experience-hdp/
[14] Kanit W., Smilkov D., Wexler J., Wilson J., Mané D., Fritz D., Krishnan D., Viégas F.B., & Wattenberg M. Visualizing dataflow graphs of deep learning models in TensorFlow. IEEE transactions on visualization and computer graphics, 2018, 24, 1, 1–12.

Karan Maheshwari, Aditya Shaha, Dhruv Arya,
Rajkumar Rajasekaran, and B.K. Tripathy

# 2 Convolutional Neural Networks: A Bottom-Up Approach

**Abstract:** A lot has changed in the world since the inception of the so-called deep learning (DL) era. Unlike conventional machine learning algorithms, where human learns more of feature extraction than machine, which just crunches numbers, DL algorithms are quite promising as they have revolutionized the way we deal with data and have become adept in taking humanlike decisions. Based on the powerful notion of artificial neural networks, these learning algorithms learn to represent data or, if we rephrase, can extract features from raw data. DL has paved way for end-to-end systems where one learning algorithm does all the tasks. Convolutional neural networks have earned a lot of fame lately, especially in the domain of computer vision where in some cases its performance has beaten that of humans. A lot of work has been done on convNets and in this chapter we will demystify how convolutional neural networks work and will illustrate using one novel application in the field of astronomy where we will do galaxy classification using raw images as input and classify them based on its shape. In addition to this we will investigate what features the network is learning. We will also discuss how DL superseded other forms of learning and some recent algorithmic innovations centered on convolutional neural nets.

**Keywords:** deep learning (DL), convolutional neural networks, galaxy classification, artificial neural networks, deconvolution

## 2.1 Introduction

We as humans have always wondered about our position in the universe. Since the existence of mankind, we have always looked up into the sky and asked ourselves qualitative questions about the Sun, the Moon, the stars, and the Universe. How the universe came into existence? What's the future of our universe? These are two of the many questions, yet to be answered clearly. There are many theories but there is no consensus about one in the scientific community. The astronomers and scientists in

**Karan Maheshwari, Aditya Shaha, Dhruv Arya, Rajkumar Rajasekaran,** School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India.
**B. K. Tripathy,** School of Information Technology and Engineering, Vellore Institute of Technology, Vellore, India.

https://doi.org/10.1515/9783110670905-002

early twentieth century valued the data highly. For them collecting data meant hand recording observations from the Earth, it certainly was a time-consuming and costly process, but the only means to study the universe. They thought with more data they'll be able to answer all the impending questions. Today scientists have more data than they can process. Everyday petabytes (1 Petabyte = 106 Gigabytes = 1,012 kilobytes) of data is downloaded and archived into data tombs and we believe why it is called data tomb is self-explanatory. Today what scientists agree upon is that we can't answer the questions relating to our existence straight away but with more data we will get more information about the different astronomical phenomena and events occurring out there. Based upon that information we can frame new theories which may lead us to the path of knowing the origin of our universe. Before the existence of modern computing techniques and collecting data from space telescopes, cosmologists used to rely on crude techniques for analysis of hand recorded data.

It all began in 1800s when Harvard College Observatory's director Edward Pickering wanted to survey the whole sky using the newly discovered technique (stellar spectroscopy) of that era, with a dedicated and determined team of human machines who worked on classification of stars based upon presence of different elements in the spectral lines. After several hit and trial combinations they finally found the OBAFGKM star classification scheme which is still used till date. During 1911–15, these human computers were used to classify around 50,000 stars annually, which resulted in the Henry Draper catalogue (HDR). This amount of manual classification done was big feat in itself but despite the hard work, no manual method can cope with the current data rates.

With the launch of first space telescope (Hubble Space Telescope, 1990) and later on followed by many, as the technology developed, scientists and engineers crafted new state of the art techniques for recording data from the sky. Thus, now there is an abundance of data.

Today when the space telescopes scan out the sky with around a trillion stars, even at the incredible rate of classification by those human computers, it would cost us around 16,000 years to classify a billion stars or merely 0.1% of total.

In short, we would like to summarize the whole era in terms of two problems and their solutions.

1. The first problem was that data was recorded in crude way by hand or photographs from earth so it was highly distorted and inaccurate because of dust and pollution in atmosphere. So, the solution to this problem was the launch of space telescopes and it resulted in the second problem.
2. With the launch of space telescopes, the data capturing rates sky rocketed, and then for humans it is just not possible to classify this much data manually as it will take thousands of years to classify. The solution to this problem is what this chapter is focused on.

To solve the second problem, we need tools or rather we need programs which can predict anomalies if they occur or programs that can classify and cluster our data by taking decisions or finding patterns in data, for example, classifying the shape of the galaxy. In half a decade we will be marking 200 years of computation beginning from the invention of difference engines to this date where computational power has grown to such an extent that we have computers that can take decisions autonomously. As said by Plato, necessity is the mother of invention so is the field of machine learning (ML) as we needed programs which can learn to predict/classify/find patterns on its own or rather we'd say programs which can do the tasks performed by the so-called human machines on large amounts of data, for example, galaxy classification. Thus, formally we can say that ML is a program which improves on its performance measure ($p$) over task ($t$) with more experience (e) of (t) [1]. ML is a set of different techniques based on principles of statistics and calculus that allow us to create programs, which without being explicitly coded can do tasks, which requires human-level decision making and intelligence. Examples of different ML-based algorithms are regression, logistic regression, artificial neural networks (ANNs), support vector machines, clustering, and so on. Using these techniques, we are either trying to predict value of some entity (e.g., house price prediction) or we are trying to classify some entity (e.g., tumor class prediction), based on certain factors which we call features of the data. For instance, let's say we want to predict the price/category of a house. This price/category will depend on a number of factors (features), such as size of house ($x_1$), locality ($x_2$), number of bedrooms ($x_3$), and number of floors ($x_4$). All these features become our input data and contribute differently toward the house price or category; hence, their contributions can be represented as $w_1$, $w_2$, $w_3$, and $w_4$, respectively. The actual price/category of the house can be represented as y and the predicted price/category of the house can be represented as y'.

We can group these algorithms based upon the way they learn.

1. Supervised ML algorithms: Here the data comprises of $x_1$, $x_2, ..., x_n$ and $y$. The goal of algorithm is to predict $y$' correctly by learning the $w_1$,$w_2$,,$w_n$, which map the relationship between $x$ and $y$. The algorithm learns under the supervision provided by labeled y as it optimizes the weights based on error (y-y'). This error is used to tune $w_1$, $w_n$ based on different learning strategies (e.g., backpropagation) and weight updating rules (e.g., gradient descent).
2. Unsupervised ML algorithms: Here the data comprises of only $x_1$,$x_2$, ..., $x_n$. The goal of algorithm is to cluster or group the data into different number of categories ($y_1$,$y_2$..) prespecified by user. These algorithms make internal representations or cluster the data based on the data itself ($x_1$,$x_2$, ..., $x_n$). Here no supervision is provided as we don't have the values of label y.

So, it can be understood that the features representing the data are very important because if we use incorrect features which cannot represent the data properly then our predictions *y*’ will be incorrect. Therefore, in traditional ML algorithms, it just becomes the task of optimizing weights of features that are extracted or designed by human experts in the field and can be seen in Figure 2.1.

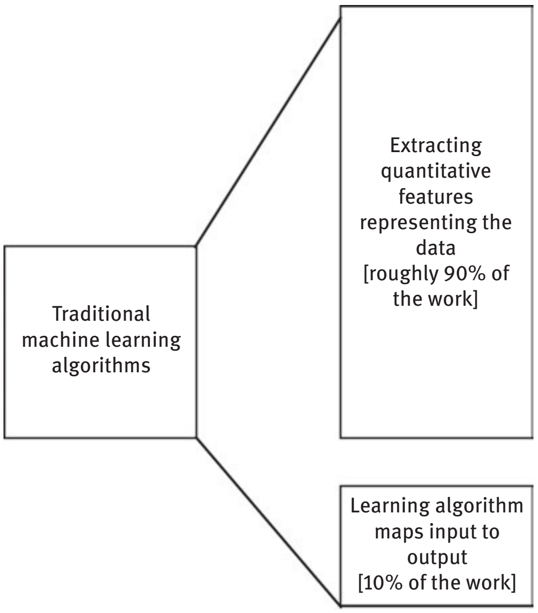

Traditional machine learning algorithms
Extracting quantitative features representing the data [roughly 90% of the work]
Learning algorithm maps input to output [10% of the work]

**Figure 2.1:** Machine learning in practice.

So, in these approaches human learns more about the problem rather than machine. In our opinion, in traditional ML almost 90% of work is done by humans to figure out features which define the data, the rest 10 % of work is done by computers, that is numeric optimization. Computers are good in numeric optimization. ML should not just mean numeric optimization of the given weights, we want computers to automatically draw features from raw data. The current deep learning (DL) algorithms exactly does this. DL algorithms are based on concept of ANNs which is inspired by human cognition.

We know that brains process information, compute a lot of data, and retain short-term and long-term memory. The human brain takes raw input through the five sensory organs and process information. This is where the concept of ANNs (ANNs) fit in. In ANNs the scientific community has tried to model how human brain works. The brain has its fundamental building blocks which are responsible for taking the input from the five senses (from the outside world) and sending instructions to the body. Analogously in the ANN model where computation happens, the building blocks are nerve cells or neurons. Neurons are responsible for taking input signals and mapping these inputs to a targeted response. Similarly ANNs have a dense network of interconnected neurons where each connection has synaptic weights. The input signals are propagated through this dense network and mapped to a target response. Table 2.1 and (Figure 2.2) show the comparison between biological neuron and artificial neuron.

**Table 2.1:** Biological neuron versus artificial neural networks.

| Biological neuron | Artificial neuron |
|---|---|
| Nerve cell | Neuron |
| Dendrites | Input signals |
| Synapses | Weights |
| Soma | Net input |
| Axon | Output |

In an ordinary neural network, we update the parameters $w^{[L]}$ using weight updating rules like gradient descent which minimizes the loss after each iteration by backpropagating the error.

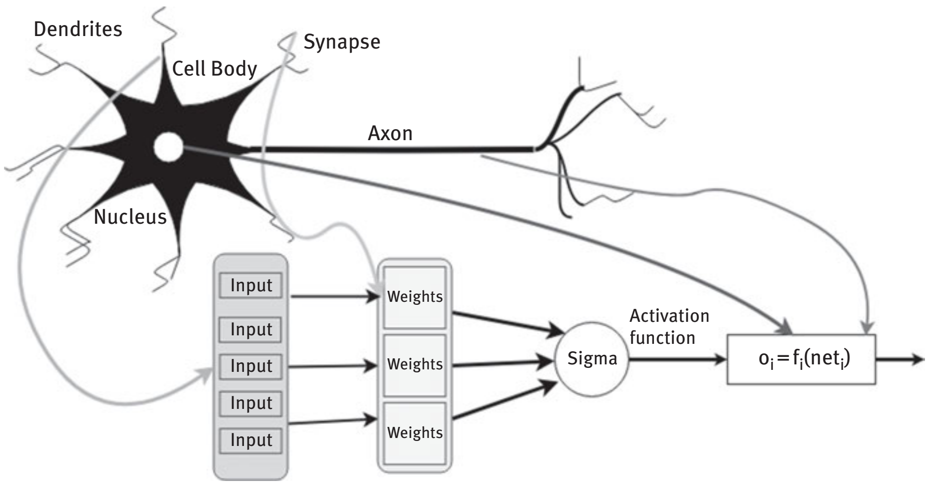


**Figure 2.2:** Neuron versus artificial neural networks.

In DL we have a dense network of layers of neurons where raw inputs are processed linearly and nonlinearly layer after layer in a hierarchical order (Figure 2.3). DL is a subfield of representation learning. Neural networks attempt to automatically learn

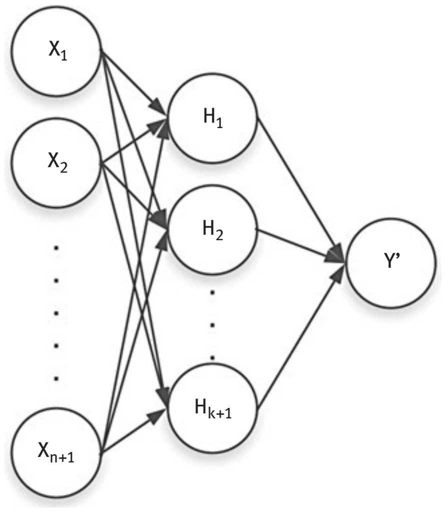


**Figure 2.3:** Artificial neural network.

good features or representations and deep neural networks (DNNs) or DL algorithms (based on ANNs) attempt to learn (multiple levels of) representation and output from raw inputs x. DL just does not mean using neural networks solely, it means we have multiple layers of representation. Convolutional neural networks (CNNs) (Le Cunn 1989) are one such DL architecture which is known to work on data that is structured in a gridlike manner (Figure 2.3).

$$Z[L] = W[L]A[L-1] + B[L]$$
$$A[L] = g(Z[L])$$

$L$ is some layer $1 \leq L \leq N$

Now coming back to our task of replacing human effort of classifying galaxy images using a DL algorithm, we need to know on what basis images are recognized. Image is a two-dimensional function $f(x,y)$, where $x$ and $y$ are spatial coordinates, and the amplitude of $f$ at any pair of coordinates $(x,y)$ is called intensity of the pixel at that point in the image. So, in layman's term we can say that an image is a $m \times n$-dimensional matrix which can be considered as a two-dimensional grid. When we are talking about an image, we consider all the raw pixels as the features of the image. For the task of object/image classification, researchers have long relied on the edges of the object. Edge map of the image gives us the general outline or shape of the object and can be really useful in object classification. In a lot of computer vision tasks, we start by passing a filter across the image which detects edges in the image. Figure 2.4 shows how a filter is passed through an image.

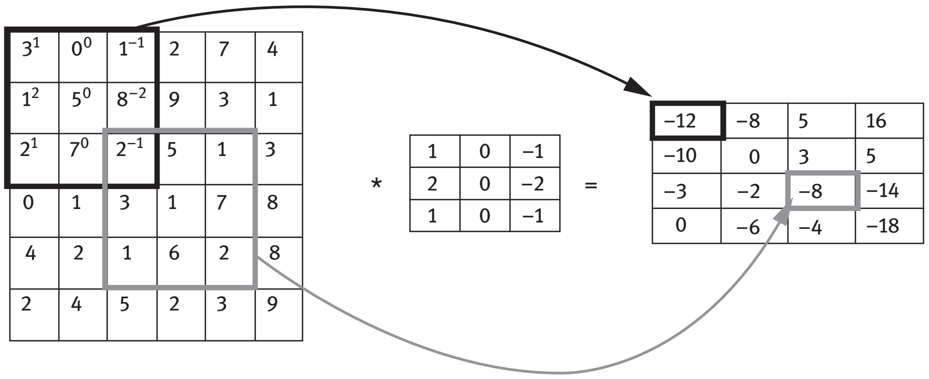


**Figure 2.4:** Application of Sobel filter on an image.

The filter used here is a Sobel filter. Here we can say that the filter can be seen as set of weights which maps input $A^{[L-1]}$ to output $Z^{[L]}$. The effect of applying a filter across an image is shown in Figure 2.5, where edges of a cat are detected using a Sobel filter.

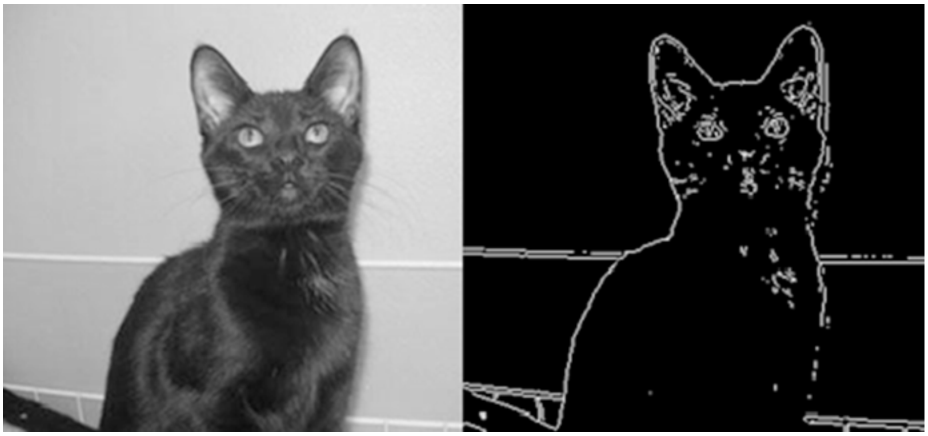


**Figure 2.5:** Output of the cat image after applying Sobel filter.

Like Sobel filter there are many different types of filters which can be used in various settings. One question which should arise in readers mind is how to select the optimum filter. As we stated before a ML algorithm learns parameters $W$ which maps input $X$ to output $Y$. Therefore, we could consider the task of object recognition as a supervised ML problem and learn the optimum filter that gives us right features (edges), which can be further used to classify the object. CNNs learn the right set of filters that extract correct features to classify the object. In the next section, we have given a mathematical overview of CNNs. The name suggests that CNNs use the mathematical operator called convolution. This convolution operation is different from what is defined in mathematics or signal processing textbooks. The DL community actually uses cross correlation (sum of element-wise product) but by convention it is being called CNN and in this chapter, we will be following that convention. As already mentioned, convolution operation can be performed on any data that is oriented in a grid format. For example, audio data (one-dimensional grid where amplitude is measured over time or sequences of texts) and images (two-dimensional grid, where each position $x$, $y$ represents a pixel). Thus, using the right filter, which can be learnt using supervised ML, we can learn to extract the right features.

Let us see how the convolution operation works.

## 2.2 The convolutional operation

In cricket, we judge a player's merit by his/her average performance. This is nothing but an average of runs that a player scores in all the matches he/she has played. Now suppose we as the coach of a cricket team have been assigned the task of selecting the best players from a bunch of players, we can use the convolution operation to help us out. The convolution operation is just a weighted sum of the runs of the player in last $n$ matches.

So, our responsibility is to choose the players who have not only performed consistently well but also performed well in the recent past. So, the runs scored by a player in the recent past should be given more weightage than the runs scored by the player in distant past. Hence, we use a weighted sum to find the performance of a player.

Let the weight of matches played in the past be less as compared to the recent matches. Then the weights of the matches are given in Table 2.2:

**Table 2.2:** Weights assigned to set of seven matches played in past.

| Match 1 ($w_1$) | Match 2 ($w_2$) | Match 3 ($w_3$) | Match 4 ($w_4$) | Match 5 ($w_5$) | Match 6 ($w_6$) | Match 7 ($w_7$) |
|---|---|---|---|---|---|---|
| 0.01 | 0.01 | 0.02 | 0.02 | 0.04 | 0.4 | 0.5 |

The runs scored by the player are given in Table 2.3.

**Table 2.3:** Runs scored by the player in the past.

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $X_7$ | $X_8$ | $X_9$ | $X_{10}$ | $X_{11}$ | $X_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 110 | 120 | 140 | 170 | 180 | 190 | 210 | 220 | 240 | 250 | 270 |

After applying the convolution operation on these scores we get the values as in Table 2.4.

**Table 2.4:** Result of the convolution operation.

| $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ |
|---|---|---|---|---|---|
| 181.1 | 196.7 | 211.2 | 226.9 | 241.3 | 256.9 |

For example, $S_6$ was obtained by

$$S_6 = X_{12}w_7 + X_{11}w_6 + X_{10}w_5 + X_9w_4 + X_8w_3 + X_7w_2 + X_6w_1$$

Thus, the convolution operation now gave us a smaller space which extracted important performance details from a larger space of run. Now, this space can be used for a more thorough comparison of performance of players.

In the above convolution operation, the kernel (weights) that we used was one-dimensional as our input was 1-dimensional. The same concept of convolution can be extended for a two-dimensional Input (like Images) by using a two-dimensional kernel:

$$S(i,j) = (K^{*}I)(i,j) = \sum_{m} \sum_{n} I(i+m, j+n)\, K(m,n)$$

In the above equation we are applying the convolution operation on the image $I$ by applying a kernel $K$ of size $(m \times n)$. Figure 2.6 [2] shows the convolution operation by a kernel of size (2 × 2) on an image of size (3 × 4). From the above example we can see that by applying a one-dimensional kernel on 1-dimensional input we get a one-dimensional output.

Similarly, by applying a two-dimensional kernel on two-dimensional input (generally grayscale images) we get a two-dimensional output. So, what will be the case when we have a three-dimensional input (like RGB images)? In RGB images having three channels ($\text{image}_{\text{width}} \times \text{image}_{\text{height}} \times 3$) we need a kernel with the same number of channels. It will be convolved with a filter having three channels. The reader should note that each individual filter channel is convolved with the corresponding

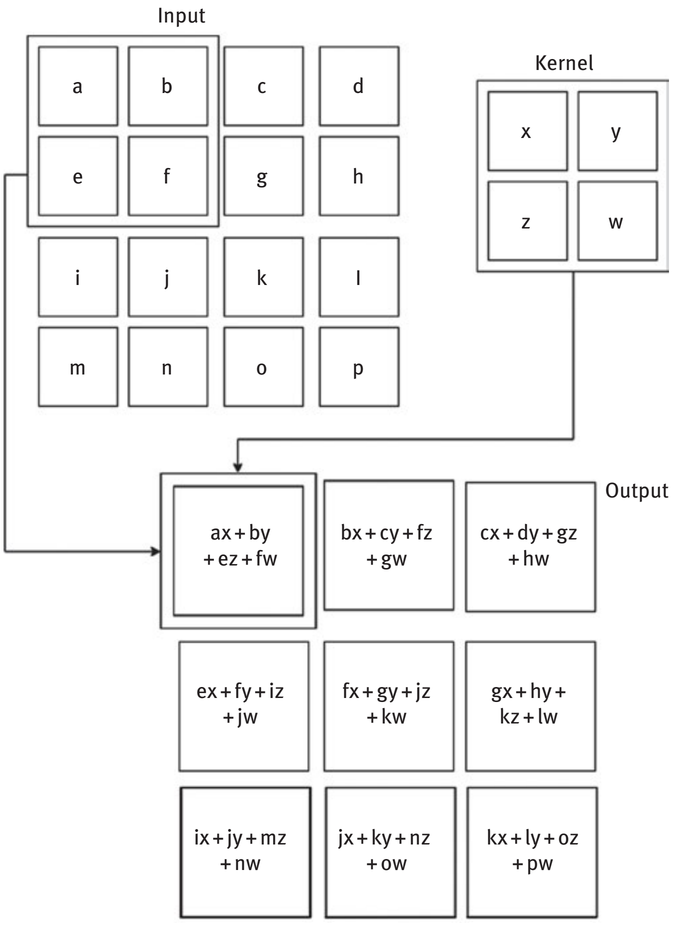


**Figure 2.6:** Convolution operation on image of size (4 × 4) with kernel (2 × 2).

input channel and the output of all channels is summed to a single output. Thus, we get a two-dimensional output. In most of the CNNs, we use *K* number of three-dimensional filters and we get a three-dimensional output where the third dimension would be *K*, the number of filters. This is illustrated in Figure 2.7.

## 2.3 Relationships between input dimension, output dimension, and the filter dimension

Now one question that can come in our mind is how we can predict the dimension of the output without actually applying the filter. The answer to this question is

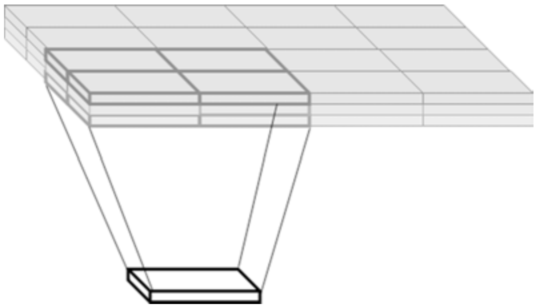


**Figure 2.7:** Convolution operation on three-dimensional images.

through a mathematical relation that exists between the input dimension, output dimension and the filter dimension. So let us start deriving this relation by answering a simple question. Why is the output size less than the input size? The answer is that there are pixels on the image where we cannot place our kernel because we will go out of the image boundary (Figures 2.8 and 2.9).

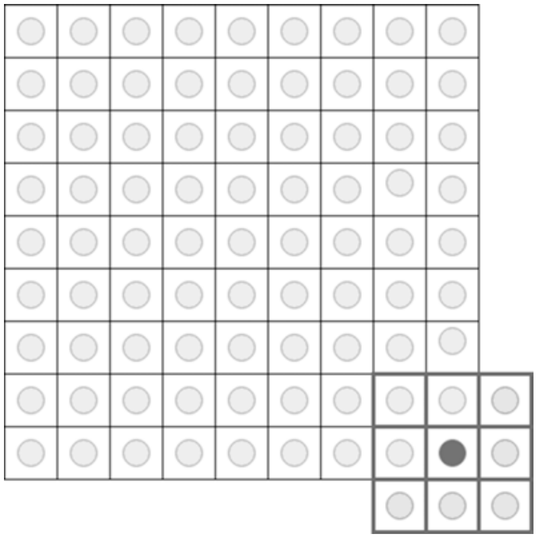


**Figure 2.8:** Convolution operation on image with kernel size (3 × 3).

To avoid this, we can pad the image before applying the convolution operation, but it will still increase the image dimension. So padding will increase the dimension of the image by (2 × 2) for (3 × 3) kernel. But what if we increase the size of the kernel? For example, for a (5 × 5) kernel we have to increase the size of the image by

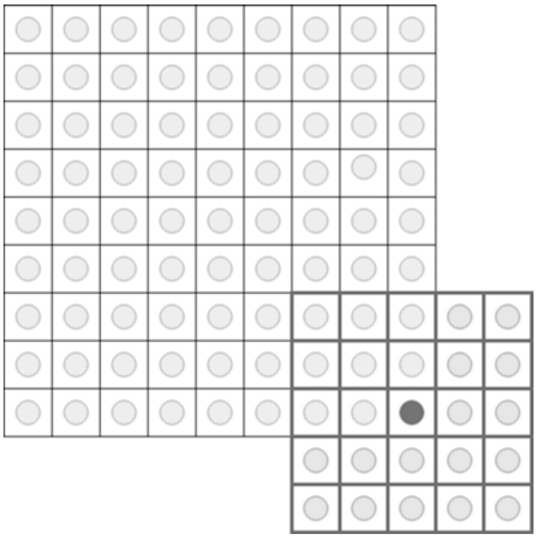


**Figure 2.9:** Convolution operation on image with kernel size (5 × 5).

padding (4 × 4). So in general we can see that by using the kernel of size *F* for a convolution operation we need a padding of *F*–1 on either sides of the image. In other words, we cannot calculate the convolution of *F*–1 pixels on either side of the image. So, the size of the image will decrease by *F*–1 So let the height and width of the image be *H* and *W*, respectively. The height and width of the output be $H_o$ *and* $W_o$, respectively. So the output dimension of the image will be

$$H_{\mathrm{o}} = H - F + 1$$

$$\mathrm{W}_{\mathrm{o}} = \mathrm{W} - F + 1$$

Now as we saw in the above example that in order to calculate the convolution of all pixels we need to pad the image and padding an image increases the dimensions of the image by twice the padding size(above and below) So, after padding the dimensions of the output will be

$$H_{\mathrm{o}} = H + 2P - F + 1$$

$$\mathrm{W}_{\mathrm{o}} = \mathrm{W} + 2P - F + 1$$

If we pad the image around with a pixel of value zero then it is called zero padding. Many a times we can do padding by adding layers of pixels which are present at the boundary. As padding is used to control the size of the output there are two commonly used types based on the output size. If the output size is equal to the input size then we call it as same padding. If the input is not padded then we call that scheme as valid padding. In addition to these two types we can always define our own custom pad size.

Reason for padding are (1) information on corner as filter passes only once and (2) output will become very small if there is no padding. So, to control the size of output we go for padding.

Now what if we don't want to calculate the convolution operation for all the pixels in our image but we want to calculate the convolution for alternate pixels or maybe two alternate pixels or k-alternate pixels. In order to solve this we introduce the concept of strides. A stride is nothing but a step that we take while applying the convolution operation. In the normal case we have a stride of size 1, in which we calculate the convolution for each pixel. To calculate the convolution of alternate pixel we can have a stride of 2 and so on.

So how will the application of stride affect our output dimensions? Because we are now calculating the alternate pixel convolution, the output size is bound to reduce by half of the input size. So, when we apply a stride of $k$, the output size will decrease by $(1/K)$th the input size. So the formula to calculate the output dimensions when we use a padding of size $P$ and Stride of $S$ is

$$Ho = \frac{H + 2P - F}{\mathrm{S}} + 1$$

$$\mathrm{Wo} = \frac{\mathrm{W} + 2P - F}{\mathrm{S}} + 1$$

## 2.4 Properties of CNNs

Now that we know what a convolutional operator is and how the convolutional operation changes the output dimension it is now time to ask a very fundamental question. What are CNNs? Also why do we use CNNs and how are they different from the normal feed forward neural networks?

Let us answer these questions one by one. *Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers* [2]. But why should we use the convolution operation? Why are we moving from matrix multiplication operation used in ANNs to the convolution operation?

The convolution operation has the following benefits which help us improve the ML process:
- Sparse interactions
- Parameter sharing
- Equivariant representations

## 2.4.1 Sparse connections

In the above example, we used kernels of size less than the image. By using a kernel of smaller size we are able to capture meaningful information in that neighborhood and only store this meaningful information for further processing. It is similar to what we saw in the cricket team selection example. We capture meaningful performance details from the array of 12 matches and store them in an array of size 6 which was significantly smaller than the original data and also helped us understand how the player has performed in his last 6 matches.

Similarly in case of images the neighboring pixels contribute more when it comes to detecting features such as edges. As we saw in the example above, a (3 × 3) Sobel filter was able to capture the edges of high-dimensional images. But how we are achieving sparse connectivity by using smaller kernels?

Consider the following case. We have to design a digit recognition software in which each image has size (4 × 4). So there are a total of 16 pixels. Let us say that we use an ANN to solve this problem. So the input layer has 16 values. Let the hidden layer have 4 neurons. Therefore, the number of connections required to connect hidden layer to the input layer is 16 × 4 = 48 (Figure 2.12). Now if we use a CNN with a kernel size of (2 × 2), the numbers of connections required are 4 × 9 = 36 (Figure 2.13).

To calculate the convolution operation for the above image, we have to multiply the kernel with the image.

So, in a CNN, with a (2 × 2) kernel the output will be nine-neuron hidden layer with neurons $\{K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9\}$ (Figure 2.10).

$$K_1 = 1 \times w_1 + 2 \times w_2 + 5 \times w_3 + 6 \times w_4$$

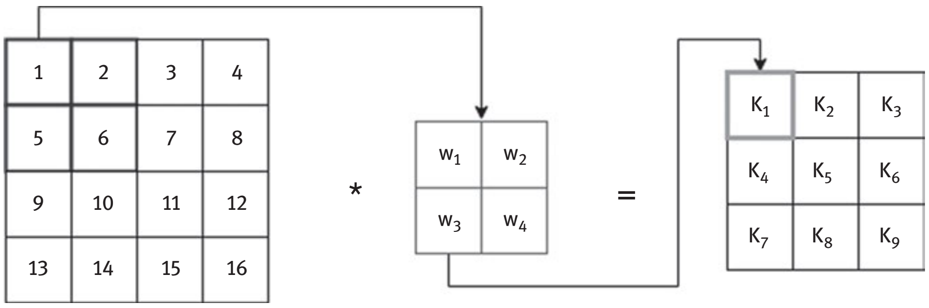


**Figure 2.10:** Using a (2 × 2) kernel on (4 × 4) image.

Thus, sparse connectivity reduces significantly the number of calculations required to calculate the output of the hidden layer. Another concern of sparse connectivity is that "Are we losing information by not letting some of the neurons interact?" The answer to this question can be explained with the help of Figure 2.11 from [2].

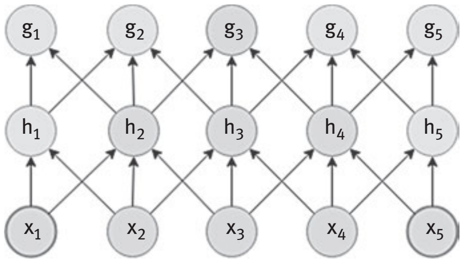


**Figure 2.11:** Interaction between sparsely connected neurons.

The neurons $\{x_1,x_2\}$ and $\{x_4,x_5\}$ in the input layer did not interact in the first hidden layer because of the sparse connections. But indirectly they interacted at the second hidden layer at the neuron $g_3$ through the neurons $\{h_2,h_3,h_4\}$.Thus deep CNNs can efficiently describe the interactions between large portions of the inputs with the help of kernels.

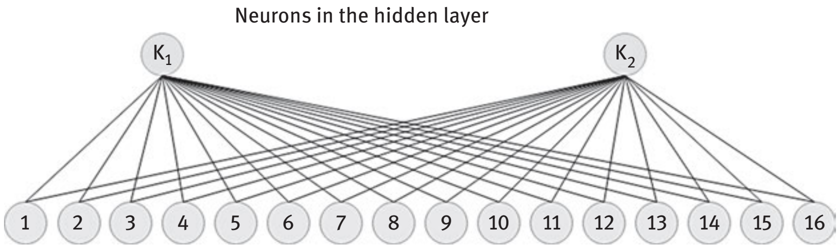


**Figure 2.12:** Dense connections in case of an artificial neural network.

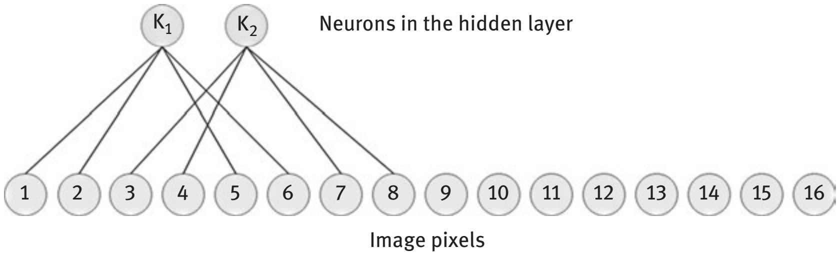


**Figure 2.13:** Sparse connection in case of convolutional neural network.

### 2.4.2 Receptive fields

The nodes which are directly affecting the highlighted node s3 $(x_1,x_2,x_3,x_4,x_5)$ are called as receptive field of s3. The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This means that even though direct connections in a convolutional net are very sparse, units in the deeper layers are getting input from all or most of the input. This is why deeper layer learns complex features (high level) because their receptive field is across the whole of input image (larger than receptive field of lower layers) (Figure 2.14).

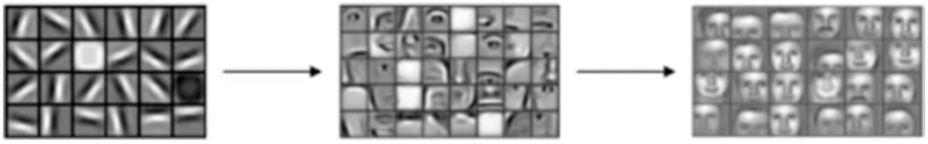


**Figure 2.14:** An illustration showing that deep layers learn high-level features and shallow layers learn low-level features.

### 2.4.3 Parameter sharing

In the example above if we see, the parameters required to find the output for every neuron in the hidden layer are same. For calculating the output of the neuron $K_{11}$ the parameters required were $\{W_1,W_2,W_3,W_4\}$. Similarly, for the $K_{12}$,$K_{13}$, and $K_{14}$ (Figure 2.15) . But in case of fully connected ANN each neuron had a different set of parameters attached to it.

An intuitive explanation for parameter sharing can be illustrated using an example. Let us, assume that we want to detect all the edges in the images. So the filter that we will use to detect all edges will be the same. We won’t use Sobel filter for one part of the image and Canny edge detector for others.

By sharing parameters there is a significant reduction in the number of parameters of the network. So there exists a possibility of underfitting. So we can use multiple filters at a single layer to avoid underfitting. An intuitive explanation of the same will be only by using Sobel filter we cannot accurately determine similar objects because mostly all cars have same shapes. So, in order to determine different cars, we need to incorporate some other filters that will learn some other distinguishing features about the images.

### 2.4.4 Equivariance to translation

To understand how the convolution operator is equivariant to translation we first need to understand what is translation. Translation means that each point/pixel in

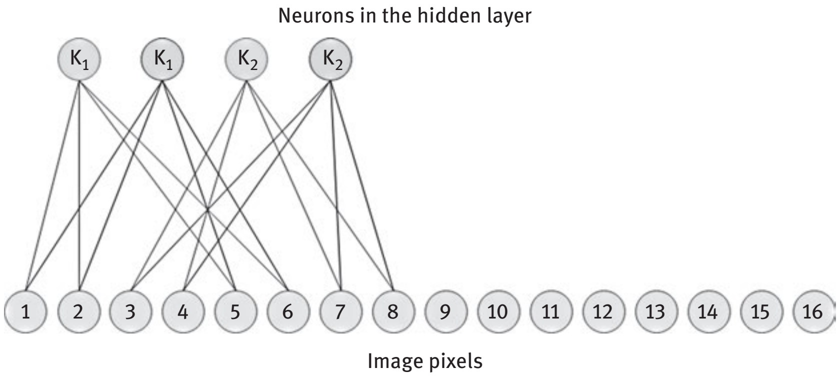


**Figure 2.15:** Parameter sharing with multiple filters.

the image has been moved the same magnitude in the same direction. Translation can be depicted by this simple Figure 2.16.

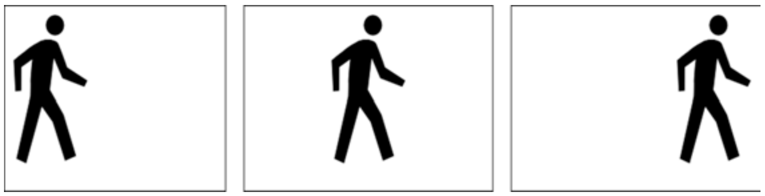


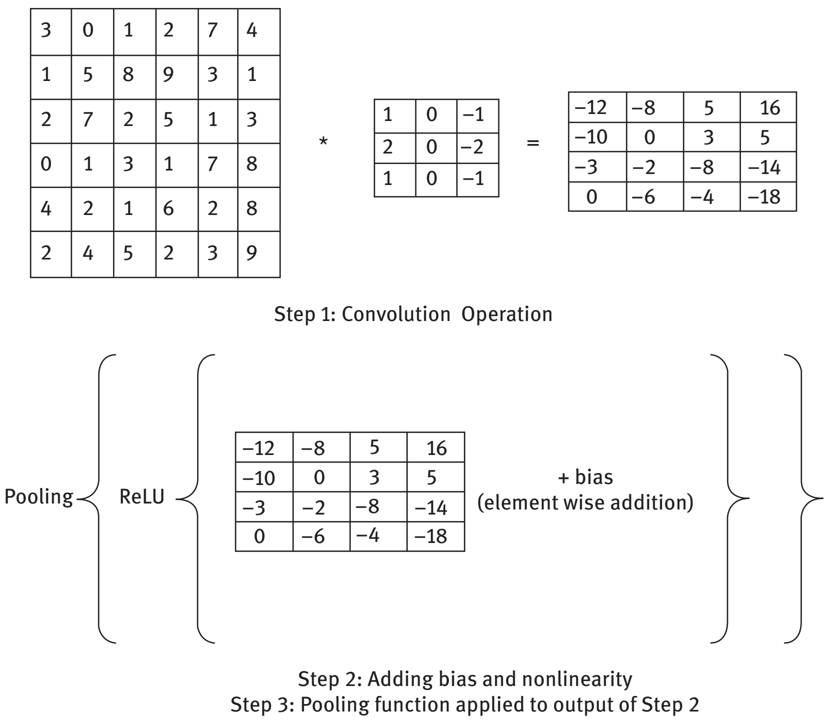
**Figure 2.16:** Translation in an image.

An operator is said to be equivariant to a transformation when the effect of the transformation is visible in the operator output. Equivariance should not be confused with invariance. An operator is invariant to a transformation if the effects of the transformation on the input do not have any impact on the operator output. The convolution operator is translation equivariant, however there is no definitive explanation for the mechanism by which the translation invariance is obtained in CNN [3]. In the paper [4] translation-invariance was attributed to the gradual increase in the receptive field size of neurons in successive convolution layer. However, paper [5] suggested that the pooling operation is responsible for the translation-invariance. In [3], the authors developed a tool called the translation-sensitivity map which was used to visualize and quantify the translation-invariance in various architectures. It was observed that the architectural choices have less effect on the translation-invariance of the network as compared to the data augmentation.

Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations.

## 2.5 Building a CNN

We've seen how convolution operation works, but in a CNN we don't just convolve the image. In every CNN, Convolution is the first step. In the second step we add bias (constant) to the convolved image, which then passes through a nonlinear function like ReLU. This stage is also known as the detector stage as it detects the features of the image. In the third step the nonlinear output undergoes a pooling function. In Figure 2.17, we have illustrated the steps involved in the fundamental block of the CNN.



$$\begin{array}{|c|c|c|c|c|c|}\hline 3&0&1&2&7&4\\\hline 1&5&8&9&3&1\\\hline 2&7&2&5&1&3\\\hline 0&1&3&1&7&8\\\hline 4&2&1&6&2&8\\\hline 2&4&5&2&3&9\\\hline\end{array} * \begin{array}{|c|c|c|}\hline 1&0&-1\\\hline 2&0&-2\\\hline 1&0&-1\\\hline\end{array} = \begin{array}{|c|c|c|c|}\hline -12&-8&5&16\\\hline -10&0&3&5\\\hline -3&-2&-8&-14\\\hline 0&-6&-4&-18\\\hline\end{array}$$

Step 1: Convolution Operation

$$\text{Pooling}\left\{\text{ReLU}\left\{\begin{array}{|c|c|c|c|}\hline -12&-8&5&16\\\hline -10&0&3&5\\\hline -3&-2&-8&-14\\\hline 0&-6&-4&-18\\\hline\end{array}\ \begin{array}{c}+\text{ bias}\\\text{(element wise addition)}\end{array}\right\}\right\}$$

Step 2: Adding bias and nonlinearity
Step 3: Pooling function applied to output of Step 2

**Figure 2.17:** An illustration of the steps involved in the fundamental block of the CNN.

### 2.5.1 Pooling in CNN

The role of the pooling function is to summarize the information the detected features hold. Obviously, there have been architectures using different permutations of these steps where certain times multiple convolution operations are performed without using pooling. Computer vision community is highly active and comes up with different state of the art architectures, and these topics are out of the scope of this chapter.

Now let's see how exactly these pooling functions work. As mentioned earlier, the task of these pooling functions is to give summary of the detected features and there are different variants of these functions. Commonly used forms are max pooling and average pooling. In max pooling as the name suggests we pass the maximum value of the feature to the output. In average pooling the average of all the features is passed to the output. Pooling is performed by passing a pooling filter across the image and producing the output according to the type of function defined. Example of max pooling can be illustrated in the following Figure 2.18.

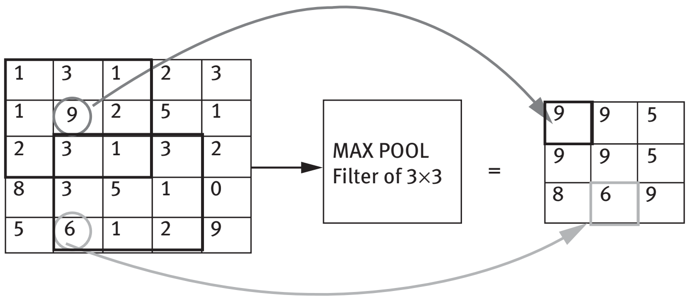


**Figure 2.18:** Illustration of max pooling.

We saw that convolution operation follows the property of equivariance to translation. Likewise, the pooling function helps the output become invariant to changes in translation in the input. All variations of pooling help make the network invariant to minute translations of the input. Researchers use combination of convolution and pooling functions to make the networks robust to small changes in images. One point the reader should note that is pooling operations are sensitive to changes in scale and rotation. Figures 2.19 shows how max pooling is invariant to translation. As observed in the figure below all the entities in

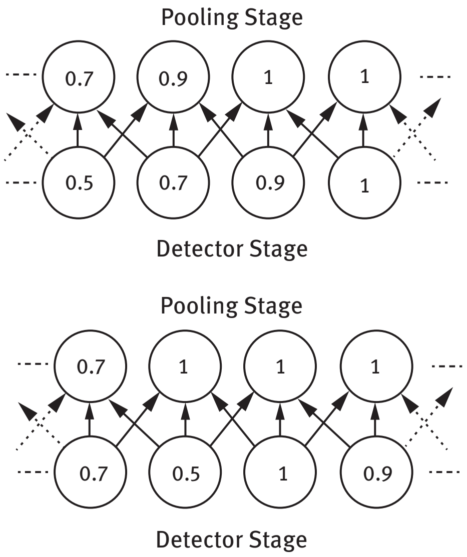


**Figure 2.19:** Pooling stage.

the input is translated to right by 1 unit, but only half of the entities in the output are changed.

## 2.5.2 Building block of CNN

The building blocks of a CNN are explained with help of Figure 2.20–2.21 and the table 2.5 provides a detailed layer-wise description of the Le-Net Architecture.

$$A^{L-1} = \mathrm{Image}_{\mathrm{width}} \times \mathrm{Image}_{\mathrm{height}} \times \mathrm{Image}_{\mathrm{channel}}$$

$$W^{L} = \mathrm{Filter}_{\mathrm{width}} \times \mathrm{Filter}_{\mathrm{height}} \times \ I_{\mathrm{channel}} \times \ F_{\mathrm{number\ of\ filters}}$$

$$Z^{L} = A^{L-1} * W^{L} + B^{L}$$

$$A^{L} = \mathrm{Pool}\left(\mathrm{ReLU}\left(Z^{L}\right)\right)$$

$$A^{L} \text{ is of dimension } \mathrm{Output}_{\mathrm{width}} \times \mathrm{Output}_{\mathrm{height}} \times \mathrm{Output}_{\mathrm{channel}}$$

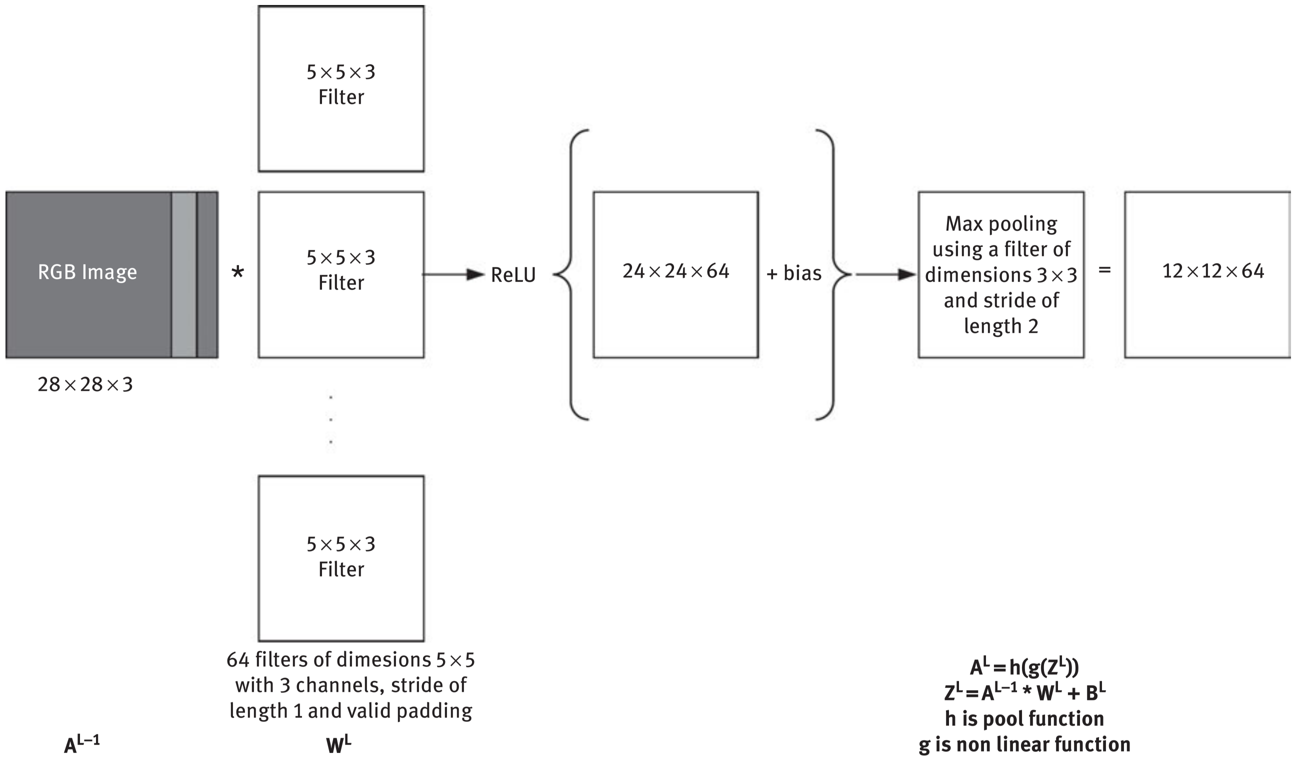

RGB Image
28×28×3
*
5×5×3 Filter
5×5×3 Filter
5×5×3 Filter
64 filters of dimesions 5×5 with 3 channels, stride of length 1 and valid padding
ReLU
24×24×64
+ bias
Max pooling using a filter of dimensions 3×3 and stride of length 2
=
12×12×64
$A^{L-1}$
$W^L$
$A^L = h(g(Z^L))$
$Z^L = A^{L-1} * W^L + B^L$
h is pool function
g is non linear function

**Figure 2.20:** A fundamental building block of the convolutional neural network.

## 2.6 Complete CNN

### 2.6.1 Le-Net architecture

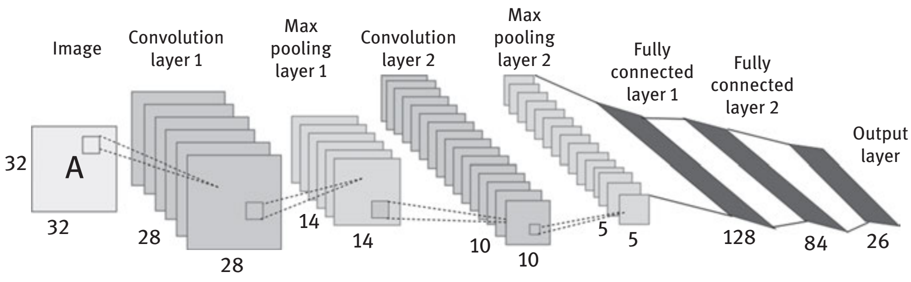


**Figure 2.21:** The Le-Net architecture.

**Table 2.5:** Detailed layer-wise description of Le-Net.

| Input layer | | | |
|---|---|---|---|
| Convolution layer | Stride = 1<br>Kernel size = 5,<br>Number of kernels = 6,<br>Padding = 0 | Output dimension $= \frac{32-5}{1} + 1$<br>= 28 × 28 × 6 | Number of parameters = 5 × 5 × 6 × 1 = 150 |
| Max pooling layer | Stride = 2,<br>kernel size = 2,<br>Number of kernels = 6,<br>Padding = 0 | Output dimension $= \frac{28-1}{2} + 1 = 14$<br>= 14 × 14 × 6 | Number of parameters = 0 |
| Convolution layer | Stride = 1,<br>Kernel size = 5<br>Number of kernels = 16,<br>Padding = 0 | Output dimension $= \frac{14-5}{1} + 1$<br>= 10 × 10 × 16 | Number of parameters = 5× 5 × 16 × 6 = 2,400 |
| Max pooling layer | Stride = 2,<br>Kernel size = 2,<br>Number of kernels = 16,<br>Padding = 0 | Output dimension $= \frac{10-1}{2} + 1 = 5$<br>= 5 × 5 × 16 | Number of parameters = 0 |

**Table 2.5** (continued)

| **Input layer** | | | |
|---|---|---|---|
| Fully connected layer 1 | Size = 120 | Output dimension = 120 × 1 | Number of parameters = 120 × 5 × 5 × 16 = 48,120 |
| Fully connected layer 2 | Size = 84 | Output dimension = 84 × 1 | Number of parameters = 84 × 120 + 84 = 10,164 |
| Output layer | Size = 26 | Output dimension = 26 × 1 | Number of parameters = 84 × 26 + 26 = 2,210 |

## 2.7 What is the network learning?

Now that we know how CNNs work, let’s try applying them. We have trained a CNN for classifying images of cats and dogs. The network seems to be performing poorly on the test set and we want to find out the reason behind it. We will explore some techniques in this section which will help us in understanding what the CNN has actually learned about a given problem.

Practitioners in the field of image processing have traditionally relied on using filters designed specifically for the task. We have seen how Sobel filters can be used to detect edges. Similarly, there are many filters which detect specific features. In the previous decade, in most image categorization pipelines, multiple such filters were used to extract features from images. The practitioner usually knew what specific feature that they were getting. For example, one might use a vertical edge detecting filter to detect road markings. Being able to understand what features are being extracted makes the decision-making process easy to understand.

In CNNs, we have thousands of filters. Furthermore, these filters are generated by the training procedure. Only the dimensionality of the filters is fixed manually. With so many filters being used, one very natural question comes to mind – what specific features from the image are being used while categorizing the image? The answer to this question helps us in debugging our CNN.

### 2.7.1 Visualizing the first layer

One simple way of discerning the types of features being extracted is to look at the filters being used. We can understand these filters by visualizing them. Simply plotting the weights gives us good insights into what features of the input image are being looked at by the filter. The filters from the first convolution layer of our cats and dogs classifier have been visualized in Figure 2.22. The first layer of this network used 64 7 × 7 by three (for the three channels) filters. The 64 filters can be seen in the figure. Note how many filters are performing the task of edge detection. In particular, the third last filter seems to be looking at oriented edges at a 135° angle horizontally. Some filters are looking for circles in specific channels.

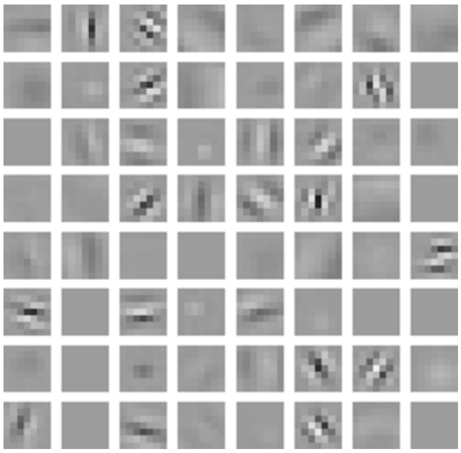


**Figure 2.22:** The first-layer filters of the cats and dogs classifier.

It is not surprising that the network is looking for such primitive features in the first layer. Experiments by neurophysiologists Torsten Wiesel and David Hubel in the 1960s on the working of brain had some similar results. They found that the cat brain also looks for primitive features such as oriented bars of light in the first few layers of the visual cortex.

Visualizing the weights of other layers of the network would not give us many insights into the decision-making process. This is because the layers after the first layer are looking at the output of the previous layer not the input image. Even if we visualize these layers, the images would not be intuitive.

Another way using which we can debug the network is to find out the areas in which the network is looking at in the input image. That is, for any input image, we want to find out the regions in the image that result in a particular classification. This might help us in gaining more insights into our classifier.

## 2.7.2 Saliency maps

Saliency maps [6] are used for finding the parts of the image that lead to a particular classification. They work by computing the gradients of the input image with respect to the target class. Note that this is different from our training procedure in which we find the gradients of the weights with respect to the loss function. When we compute the gradients of weights with respect to a target class, we are trying to find out how much and in which direction should we change them to increase the probability of predicting that class. The weights that are being increased are important for that particular target class. Similarly, when we take the gradients of an image with respect to a target class, we find out the pixels that are important for that particular target class. The gradients of our network for the cat image with respect to the target class cat have been shown in Figure 2.23. The pixels of high importance in the original image are represented by white dots in the saliency maps. Note how the collar of the cat is not considered to be important for classification by the network. This might be because the input dataset contains images of both dogs and cats with a collar. Thus, a collar is not a distinguishing feature.

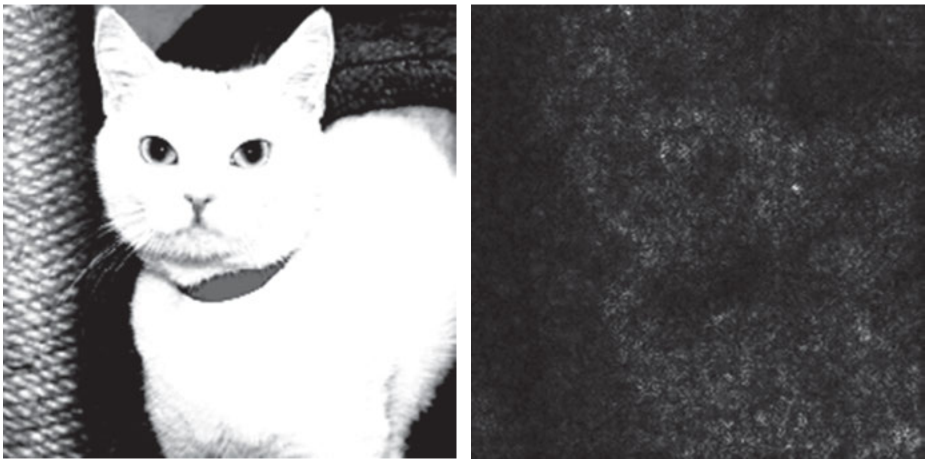

(a) The input cat image (b) The saliency map for the input image

**Figure 2.23:** The input image and its saliency map: (a) the input cat image and (b) the saliency map for the output image.

Saliency maps are particularly useful when debugging erroneous classification. They can help us spot false correlations. If in the training dataset all the dog images have a traffic light in them, the network might learn to classify all images with a traffic light as dogs. This can be easily spotted by looking at a saliency map of an image with an erroneous classification. A saliency map will clearly show that the traffic light plays a role in the classification of the image as a dog.

## 2.8 Transposed convolution

We now know how to map a 128 × 128 × 3 image into a 64 × 64 × 12 feature map. We can easily achieve this by adding unit padding along the edges of the input feature map and then using 12 kernels each of size 3 × 3 by 3 over it. However, our convolution operation cannot map a 64 × 64 × 12 feature map to a 128 × 128 × 3 feature map. That is, it cannot perform the inverse operation. Let's take the case of a television which has the resolution 2,000 × 1,000. We want to display an image of resolution 1,000 × 500 on this television. To do a full-screen display, we will have to upscale the image from 1,000 × 500 × 3 to 2,000 × 1,000 × 3. There are many algorithms for performing this upscaling operation. We can use the many variants of interpolation operation such as linear and cubic to achieve upscaling. In this section, we introduce an operation that will allow neural networks to perform upscaling.

Let's consider a feature map of size 4 × 4. When we use a kernel of shape 3 × 3 on it with a stride of one, we get an output feature map of shape 2×2. The same is shown in Figure 2.24. To recover the 4 × 4 input volume, we use a kernel of size 3 × 3 after applying zero padding of size 2 on the 2×2 feature map. The corresponding transposed convolution operation has been depicted in Figure 2.25. Let *s* be the stride

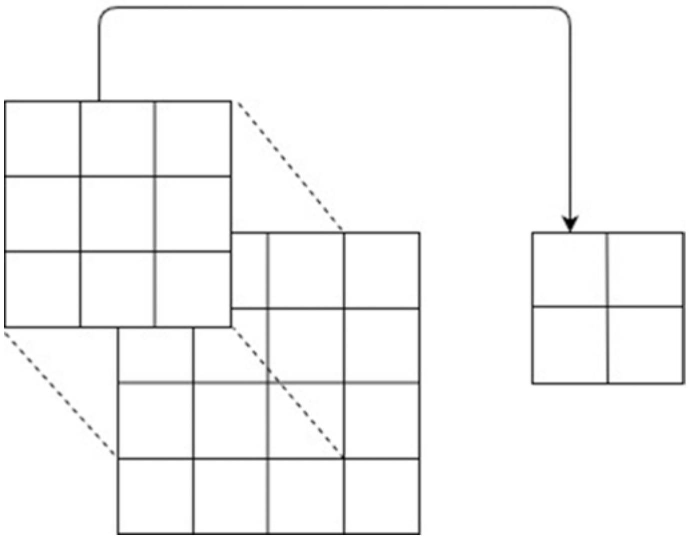


**Figure 2.24:** 3 × 3 kernel convolving on a 4 × 4 feature map.

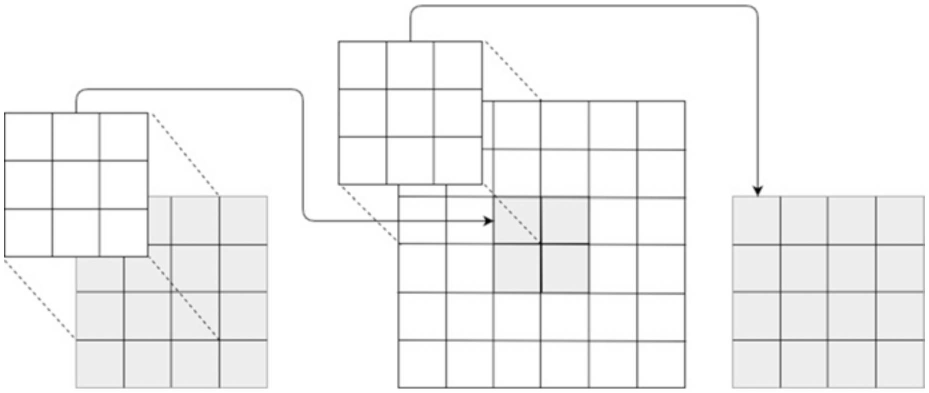


**Figure 2.25:** The transposed deconvolution operation.

length, $k$ be the kernel size (assuming a square kernel), and $p$ be the zero padding in the input image. Then to recover the original input image, when we have $s = 1$, and $p = 0$, we set $s' = s$, $k' = k$, and $p' = p$ for the transposed convolution. [7] can be referred for a more in-depth discussion of the transposed convolution arithmetic. The transposed convolution operation has been used for visualizing the feature maps of the later layers in CNNs [8]. Transposed convolution layers are also being used in applications such as super resolution [9] and semantic segmentation [10].

## 2.9 Applications of CNN

In this section we will go through the recent progress in application of CNNs in different domains.

1. Localization: Localization is the task of finding the location of an object in an image given that we know what kind of an object we are looking for. This is different from detection which involves finding all objects of interest in an image and labelling them. Localization typically involves predicting a bounding box around the object. Bounding box regression, which was developed by Girschik et al. [11], has been used with some success for localization. This involves predicting the coordinates of the bounding box in the last layer of the CNN. If the number of objects and the type of each object is known before hand, this technique can be extended by having multiple parallel fully connected layer to predict the bound box for each object.
2. Detection: In most scenarios such as in obstacle detection for self-driving cars, the number of objects that are to be detected are not known beforehand. Detection involves both localization of objects in the images and labelling them. There has

been a lot of progress in this field. In [11], the authors have proposed a method called region-based convolutional neural networks (R-CNNs). In this method, a network is used to propose regions of interest in the image. These regions of interest are then passed to a CNN classifier which predicts the class of the object in the region. A faster version of R-CNNs was proposed in [12] where the region proposal is done after a feature map of the whole image has been generated using a CNN. Other approaches such as you only look once [13] and single shot detection [14] have been developed. These approaches are significantly faster as they involve passing the image through a CNN only once.

3. Semantic segmentation: Semantic segmentation involves assigning a class to every pixel in the image. Long et al., presented a model in [10] called the fully convolutional networks which achieved breakthrough results in semantic segmentation. The model first uses the convolution operation to map the image to a feature map and then the transposed convolution operation is used to map this feature map back to a map with height and width the same as the input image. The depth of the final output is equal to the number of classes. Thus, the value of the point at row $i$, column $j$, and depth $d$ in the output feature map helps us compute the probability of the pixel $I_{ij}$ in the input image $I$ belonging to class $d$. Figures 2.15–2.26 illustrate localization, detection, and segmentation.
4. Speech recognition: The application of CNNs is not limited to just visual data. CNNs are also being applied in speech and text processing. Speech recognition is the task of identifying words and phrases in speech and storing them in machine-readable format. Most approaches such as in [15] pass frames from the audio signal to a CNN. The CNN uses a pooling operation over time. CNNs have been shown to be comparable to recurrent neural networks (RNNs) for speech recognition.
5. Visual Question Answering (VQA): In VQA, our agent has to answer questions related to an image. These tasks require a deeper understanding of the image; simple captions are not that useful. The agent also needs to perform complex reasoning. Many modern approaches such as [16] use a CNN to map the image to a feature vector using a CNN and using a RNN for generating an answer for the given textual question. The advantage of such a system is that it can be trained end to end.

## 2.10 Conclusion and future work

In this chapter we have introduced the powerful CNNs (CNN). CNNs allow us to overcome the many problems faced when making DNNs. They have found application in many domains such as image classification, object detection, segmentation, text classification, and speech recognition. The recently developed generative adversarial networks (GANs) [17] have opened up new possibilities for applications of CNNs. Radford et al. [18] presented an architecture that combines both GANs and

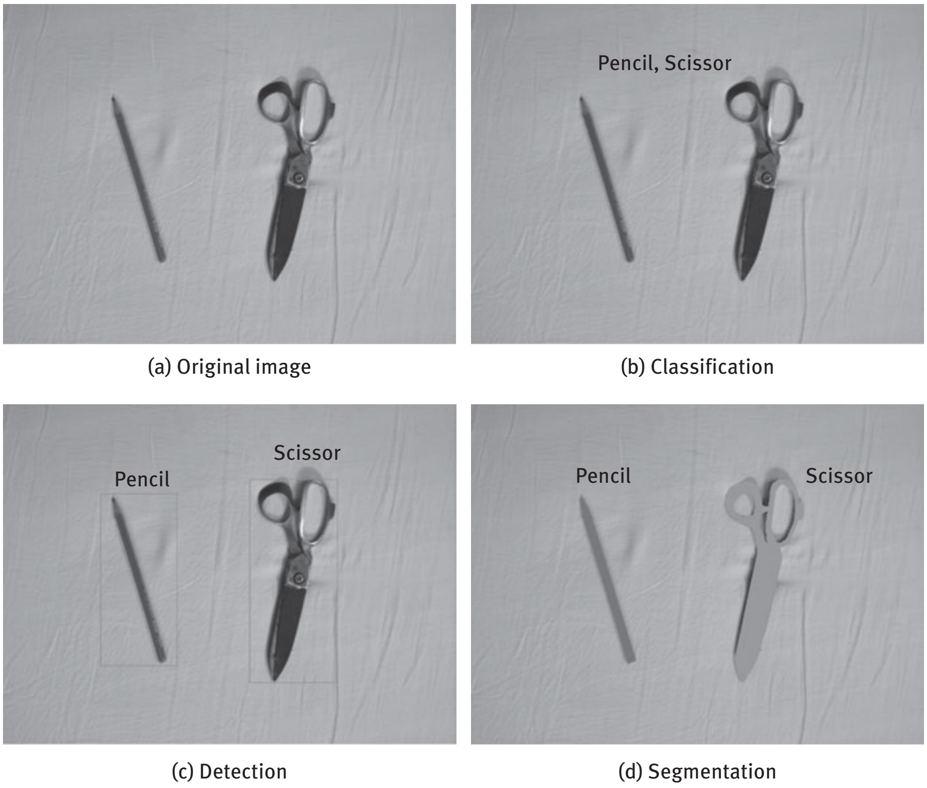


**Figure 2.26:** Illustration of the differences between classification, detection, and segmentation.

CNNs. This has resulted in many impressive breakthroughs in generative tasks involving images. In [9], the authors have used this combination of CNNs and GANs for upscaling images to higher resolutions. A lot of progress has also taken place in mapping images from one domain to another even without paired datasets [19]. Capsule networks [20] have built upon CNNs and are an attempt at extracting equivariant features. CNNs are an active area of research and new applications with breakthrough results are consistently being found.

## References

[1] Mitchell T. M. Machine learning. WCB, 1997.
[2] Goodfellow I., Bengio Y., & Courville A. Deep Learning. MIT Press, 2016.
[3] Kauderer-Abrams E. Quantifying Translation-Invariance in Convolutional Neural Networks. 2017.
[4] Gens R., Pedro M. D. Deep symmetry networks. Advances in Neural Information Processing Systems, 2014.

[5] Jaderberg M., Simonyan K., & Zisserman A. Spatial transformer networks. Advances in Neural Information Processing Systems, 2015.

[6] Simonyan K., Vedaldi A., Zisserman A. Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034, 2013.

[7] Dumoulin V., Visin F. A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285, 2016.

[8] Zeiler M. D., & Fergus R. Visualizing and understanding convolutional networks. European conference on computer vision, Springer, Cham, September 2014, 818–833.

[9] Ledig C., Theis L., Huszr F., Caballero J., Cunningham A., Acosta A., & Shi W. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. CVPR, July 2017, 2(3), 4.

[10] Long J., Shelhamer E., & Darrell T. Fully convolutional networks for semantic segmentation. Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, 3431–3440.

[11] Girshick R., Donahue J., Darrell T., & Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, 580–587.

[12] Girshick R. Fast R-CNN. Proceedings of the IEEE international conference on computer vision, 2015, 1440–1448.

[13] Redmon J., Divvala S., Girshick R., & Farhadi A. You only look once: Unified, real-time object detection. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, 779–788.

[14] Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C. Y., & Berg A. C. Ssd: Single shot multibox detector. European conference on computer vision, Springer, Cham, October 2016, 21–37.

[15] Palaz D., Collobert R. Analysis of cnn-based speech recognition system using raw speech as input. Proceedings of INTERSPEECH, 2015, EPFL-CONF-210029.

[16] Malinowski M., Rohrbach M., & Fritz M. Ask your neurons: A neural-based approach to answering questions about images. Proceedings of the IEEE international conference on computer vision, 2015, 1–9.

[17] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., & Bengio Y. Generative adversarial nets. Advances in Neural Information Processing Systems, 2014, 2672–2680.

[18] Radford A., Metz L., Chintala S. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.

[19] Zhu J. Y., Park T., Isola P., & Efros A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. arXiv preprint, 2017.

[20] Sabour S., Frosst N., & Hinton G. E. Dynamic routing between capsules. Advances in Neural Information Processing Systems, 2017, 3856–3866.

Ranjan Jana, Siddhartha Bhattacharyya, and Swagatam Das

# 3 Handwritten Digit Recognition Using Convolutional Neural Networks

**Abstract:** Optical character recognition (OCR) systems have been used for extraction of text contained in scanned documents or images. This system consists of two steps: character detection and recognition. One classification algorithm is required for character recognition by their features. Character can be recognized using neural networks. The multilayer perceptron (MLP) provides acceptable recognition accuracy for character classification. Moreover, the convolutional neural network (CNN) and the recurrent neural network (RNN) are providing character recognition with high accuracy. MLP, RNN, and CNN may suffer from the large amount of computation in the training phase. MLP solves different types of problems with good accuracy but it takes huge amount of time due to its dense network connection. RNNs are suitable for sequence data, while CNNs are suitable for spatial data. In this chapter, a CNN is implemented for recognition of digits from MNIST database and a comparative study is established between MLP, RNN, and CNN. The CNN provides the higher accuracy for digit recognition and takes lowest amount of time for training the system with respect to MLP and RNN. The CNN gives better result with accuracy up to 98.92% as the MNIST digit dataset is used, which is spatial data.

**Keywords:** convolutional neural network, deep neural network, handwritten digit recognition, multilayer perceptron, optical character recognition, recurrent neural network

## 3.1 Introduction

Optical character recognition (OCR) translates the handwritten text image or printed text image into normal text. The text images are generally from images of a text document [1]. OCR is a significant area in many computer vision problems like automatic number plate recognition, extracting business card information, and data entry for passport, cheque, and invoice documents. OCR is mostly an offline process, which converts text from a static document. Movement of handwriting can be used for handwriting character recognition [2]. In online character recognition, instead of using the shape of the characters or words, the direction and pressure on pen can be used for

**Ranjan Jana,** RCC Institute of Information Technology, Kolkata, West Bengal, India.
**Siddhartha Bhattacharyya,** CHRIST (Deemed to be University), Bangalore, Karnataka, India.
**Swagatam Das,** Indian Statistical Institute, Kolkata, West Bengal, India.

https://doi.org/10.1515/9783110670905-003

extra information to make the process more efficient. OCR is a mechanism that classifies the character or word belonging in a text image. The character recognition from text image is processed through character segmentation, features extraction, and features classification. Introduction of multilayer perceptron (MLP) has been an outbreak for data classification in computer vision [3], although the performance of MLP fully depended on appropriate features of the object for classification [4, 5]. The classical machine leaning technique has been revolutionized by deep neural networks (DNNs). This network works on the raw pixel data to extract appropriate features for classification [6]. DNN is designed with a huge number of hidden layers and connections. So, the network has many numbers of parameters and it takes huge numbers of training samples to prevent overfitting. DNN architectures are mainly classified into convolutional neural network (CNN) and recurrent neural network (RNN) to solve the problems of various computer vision tasks. For extraction of position-invariant features, CNN is generally used, where RNN is used for modeling the network in sequence data [7]. CNN has fewer numbers of parameters compared to a fully connected neural network [8]. CNN is able to model the data by changing the number of hidden layers and the number of trainable parameters of each layer [9]. CNN can design versatile nonlinear relationship between input and output with local receptive field and temporal subsampling to provide a degree of shift, rotation, and distortion invariance [10]. In a neural network, CNN is formed by a series of convolution and pooling layers between input and output layers. A fully connected layer is performed for generating unique representation of the input data in the last layer of the network. In this chapter, the comparison of performances is made for MLP, RNN, and CNN methods on the MNIST dataset. CNN provides better results compared to MLP and RNN.

This chapteris organized as follows. The motivation for handwritten digit recognition is discussed in Section 3.2. Related works are presented in Section 3.3 and the overview of MLP, CNN, and RNN architecture are discussed in Section 3.4. The implementation details of the system are described in Section 3.5. Experimental results are shown in Section 3.6. Finally, the conclusions are derived in Section 3.7.

## 3.2 Motivation for handwritten digit recognition

The handwriting characters recognition technique was started around 1980. The task of handwritten digit recognition using a classifier has a great importance in machine learning. Handwriting character recognition is generally important for zip code recognition for postal mail sorting, processing of bank check, extraction of numeric entries in handwritten filled-up forms, and number plate recognition of any vehicle. The handwritten digits are different in size, alignment, and thickness for different time even for the same writer. Handwriting of different individuals for same digit are different that influences the appearance of the digits. So, there are many challenges that need to be faced to implement handwritten character recognition. The common problem in digit recognition is to predict the digits in spite of their similarity like 3 and 5, 9 and 4, 8 and 9, 5 and 6 as shown in Figure 3.1. The aim of this chapter is to implement a method for recognition of handwritten digits with varying size, thickness, and orientation.

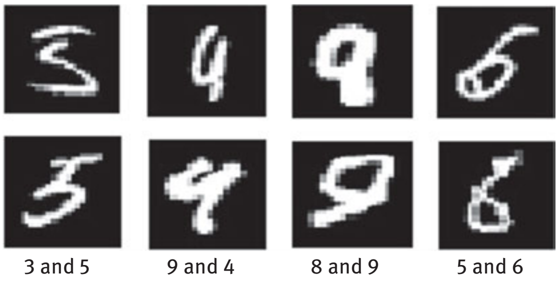


**Figure 3.1:** Similarity of digits.

## 3.3 Related work

Arica and Vural proposed an offline character recognition system by using minimal preprocessing operations to restrict loss of essential information [11]. They used an efficient segmentation algorithm for detecting character boundary, slant angle, lower and upper baseline, stroke width, and height from a grayscale image. Hidden Markov model (HMM) was used to estimate the system parameters as well as some feature space parameters. HMM was used to determine the rank of the character based on the features of character shape. Khedher et al. described that selection of good features provides better accuracy for any character recognition system [12]. They discussed how different good features were selected for recognizing Arabic handwritten characters. They used a real sample of handwritten characters. The average recognition accuracy was 88% for the Arabic numbers and 70% for the Arabic letters. Hanmandlu and Murthy implemented a fuzzy model in the form of membership values for recognition of handwritten English and Hindi numbers [13]. The membership function in the fuzzy set was adapted by optimization of entropy. The accuracy of the model was 95% for Hindi digits and 98.4% for English digits. Arora et al. implemented a system for the recognition of Devanagari character [14]. Four features were extracted from the shadow, straight line fitting, intersection, and chain code histogram using box approach. The features of shadow were extracted from all portion of the image and the features of chain code histogram, intersection, and straight-line-fitting were extracted by partitioning the image into subimage. The system was tested using 4,900 Devanagari characters and the accuracy was 92.8%. Kimura implemented a system for character recognition that used features selection using a genetic algorithm [15]. He selected the genes in such a way that the recognition rate of training data should not exceed the predefined threshold. So, the system was adopted as a reduction ratio for selecting the number of features. Graves and Schmidhuber developed a multidimensional RNN model for Arabic handwriting character recognition. They participated in the ICDAR word recognition competition in 2007 and the accuracy was 91% for the IFN/ENIT database [16]. MLP was used by Pal and Singh for recognizing handwritten English characters [17]. They extracted the features using character boundary tracing and Fourier descriptors. The system classified each character by comparing features and shape of the character. They did an investigation to get better accuracy for character classification by using the optimized number of hidden layers. They reported 94% recognition accuracy for handwritten English characters. Pradeep et al. extracted the diagonal features for offline character recognition [18]. The system was based on a neural network model and they used two approaches by using 54 features and 69 features to implement this recognition system. They reported that the accuracy of offline character recognition was 97.8% for using 54 features and 98.5% for using 69 features. Neves et al. proposed an offline handwritten digit recognition system using a support vector machine and declared that it provided better accuracy compared to MLP [19]. The experiment was done on an NIST SD19 standard dataset. They reported that, MLP is capable of distinguishing nonlinear separable classes, but it could not achieve an optimal solution due to being stuck in local minima. So, an MPL might not be able to provide higher accuracy for digit recognition. Gaurav and Bhatia proposed a technique that deals with the various preprocessing techniques which are detection and correction of skew, contrast stretching, conversion into a binary image, removal of noise, segmentation, and morphological operation for character recognition [20]. The technique was used for the handwritten document containing a color background with different intensities. They concluded that a single technique is not sufficient for character recognition from the varied intensities background image. Deep learning was used in a broad range of applications for its consistent performance in computer vision and machine learning [21]. Hu et al. implemented an OCR system and a face recognition system using a DNN and a CNN, which gave better accuracy than other systems. DNNs and CNNs provided a great opportunity for growing the applications of deep learning. They used the system for solving the problem of face recognition for security problems in Jordan. They achieved great success for face recognition. Ghazi and Ekenel demonstrated the model of VGG-Face and lightened CNN on five benchmark datasets. They reported that preprocessing can improve the performance in spite of having a small alignment of face or character [22]. The method showed favorable performance with accuracy up to 98.46% on MNIST dataset. Using CNN, preprocessing of image/data and handcrafted features extraction was not required. Younis and Alkhateeb implemented a face recognition system using a deep CNN on a face dataset, which was captured from different persons in University of Jordan [23]. They implemented the system in the environment of Tensorflow and Keras. The accuracy of face classification was acceptable after 80 epochs which took a very low amount of time.

## 3.4 Overview of MLP, CNN, and RNN

MLP network has been used for a long time in pattern recognition, which used the hand-crafted features for pattern classification. A huge amount of time is required for selection of best features for a particular application. The best features are also called hand-crafted features. The hand-crafted features are fully dependent on some applications but it may not be applicable for other applications. Deep learning (DL) was a revolution in the field of neural networks, which uses huge number of layers for training the network to achieve better accuracy. DL consists of one input layer, a series of hidden layers, and one output layer for features extraction and features classification. DL tries to extract more features automatically through multiple stage of feature learning process. Researchers have proposed numbers of neural networks as modifications of existing model or a new model. They have generally recommended MLP, CNN, and RNN for consistent performance in broad range of applications.

### 3.4.1 Multilayer perceptron

A perceptron is a simple neural network model. It is the pioneer of complex neural networks. It tries to solve difficult problems like biological brains. The power of neural networks comes from their ability to model the system correctly from the training data and it predicts the output for data classification. In this sense, neural networks learn mapping scientifically. These types of networks are capable of learning any mapping function. The capability for prediction of any neural networks comes from the multilayered structure of the networks. The architecture of MLP consist of one input layer, one output layer, and multiple hidden layers as shown in Figure 3.2. The architecture learns the features at different scales and combines them into higher-order features. For example, the architecture learns lines, then from collections of lines to shape. MLP is an efficient classifier. During training, the inputs are assigned with a class for the network to train the system. The network learns mapping very efficiently from input dataset to output dataset for data classification.

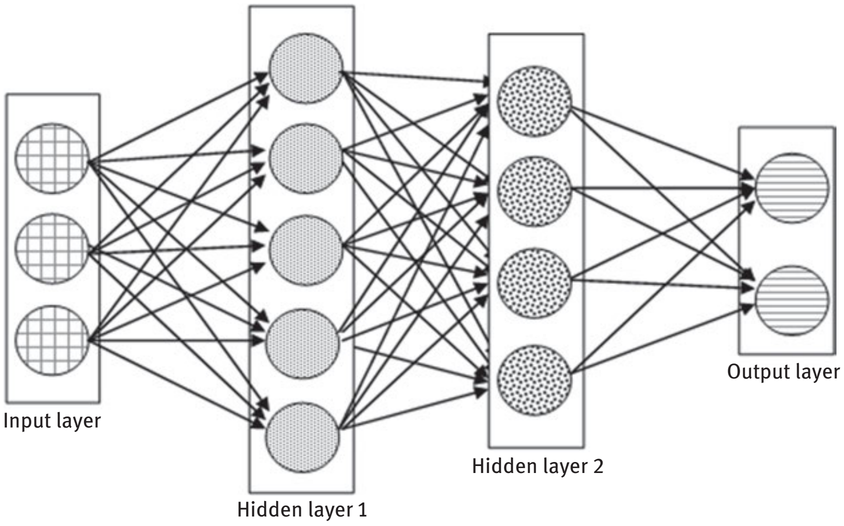


**Figure 3.2:** Basic architecture of MLP.

### 3.4.2 Convolutional neural network

CNN is an efficient neural network architecture in computer vision. It is used to extract the features automatically, and then classify the extracted features. The architecture uses a sequence of filters on raw pixels of an image to extract high-level, middle-level, and low-level features and then classifies the image based on extracted features. CNN maintains the spatial relationship between pixels using small squares of input data to extract features. Features are extracted across the entire image to allow the objects in the image to be translated or shifted. CNN architecture consists of a sequence of different layers that transform an input data into an output data. This architecture consists of input layer, output layer, and hidden layers as shown in Figure 3.3. The hidden layers are divided into two modules. First module is used for feature extraction and the second module is used for features classification. The first module consists of repetitive convolution layers and pooling layers. The second module consists of a number of dense layers. The number of nodes in the last dense layer consists of the number of target classes. This layer is called output layer.

#### 3.4.2.1 Convolution layer

Convolution layer is the main layer of a CNN. This layer consists of a set of convolution filters to perform some mathematical operations to generate one value in the output map. In the forward pass, every filter is convoluted throughout the entire

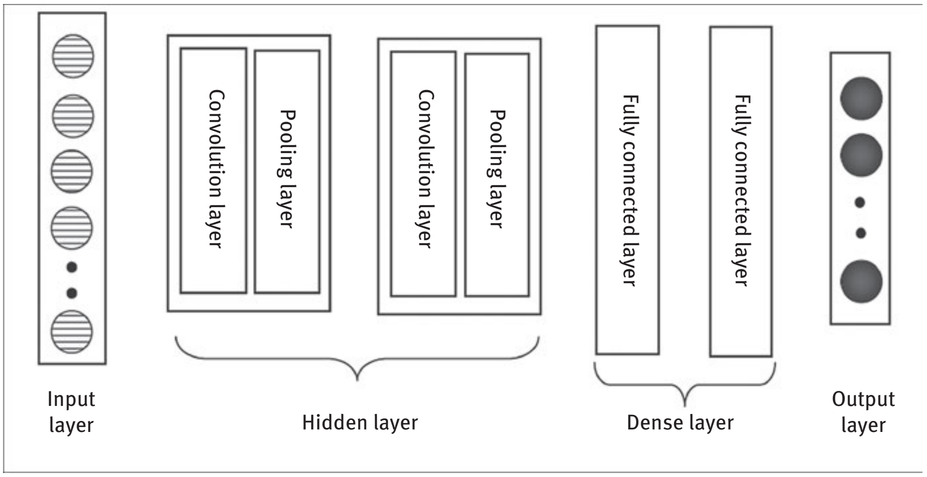


**Figure 3.3:** Basic architecture of CNN.

image subregions. It is simply the dot product between the values of the subregion of input image/data and the values of the convolution filter. It produces a two-dimension activation map of the convolution filter to detect some specific type of features. The output data size of the convolution layer depends on the size of the filter, the value of stride, and zero-padding. The number of neurons in a convolution layer that connect to the same subregion of the input data is controlled by the size of the filter. These neurons learn to activate different features in the input data. For the first convolution layer, the input data is the raw pixels of the input image and the neurons learn to activate the presences of edges in the input image. The step of convolution operation is called stride, which controls the size of the output data of the convolution layer. If the stride is n then the convolution filter moves $n$ pixels at a time. For low value of stride, the filter is overlapping the receptive fields and produce large output data size. Zero-padding is used to pad the border of the input data by the zero. It is required to maintain the output data size same as input data size. Each convolution layer applies a rectified linear unit (ReLU) to the output map which establishes nonlinearities into the architecture. ReLU applies element wise activation without affecting the receptive fields of the convolution layer. The most frequently used nonlinear activation functions in this layer are $f(x) = \tanh(x)$, $f(x) = \max(0,x)$, and $f(x) = (1 + e^{-x})^{-1}$.

### 3.4.2.2 Pooling layer

The pooling layer performs a nonlinear down-sampling operation on the input data generated by previous convolution layer. This layer performs between two successive convolution layers. It is a procedure to compress the extracted features and reduce the overfitting of the training data. Most frequently, pooling layer uses 2 × 2 filter with a stride of 2 for down-sampling. So, this technique is discarding 75% of the activations. Although, several nonlinear functions are available, max pooling is the most frequently used nonlinear function for pooling operation. In this layer, the input data is partitioned into a number of square subregions and generates the highest value of each subregion. The logic behind pooling is that the actual location of a feature in the image/data is less significant. This layer provides the architecture to another form of translation invariance.

#### 3.4.2.3 Fully connected network (FCN) layer

After the repetitive convolution and pooling layers, one or more FCN layer performs features classification. Here, every node of current layer is connected to every node of previous layer. The number of nodes in the final FCN layer consists of the number of target classes. The softmax activation function is used to generate a value between 0 and 1 for each node of the target class. The softmax function generates the probability of an image/data belonging to each target class. The total of all softmax values is 1.

### 3.4.3 Recurrent neural network

RNN is capable to memorize the input into output mapping for its internal memory. The network contains minimum one feedback connection for memorization as shown in Figure 3.4. A delay unit is used to introduce the output of RNN to loop back into the network for memorization. RNN inserts the recent past data to the current state. So, RNN has a present input data as well as recent past input data. The network is an exceptional neural network that is proposed for sequence data. This network allows demonstrating the dynamic behavior for a time sequence data. RNN architecture uses its memory to process sequences of input data. RNN is a feed-forward MLP with addition of loops in the architecture. So that, each neuron of a layer can transfer its signal to the next layer. The output of a layer may feedback as an input to the layer for the next input data. The recursive connections in RNN permit to gain knowledge of broad abstractions for any sequence of input data. RNN is more applicable for connected handwriting character recognition or speech recognition.

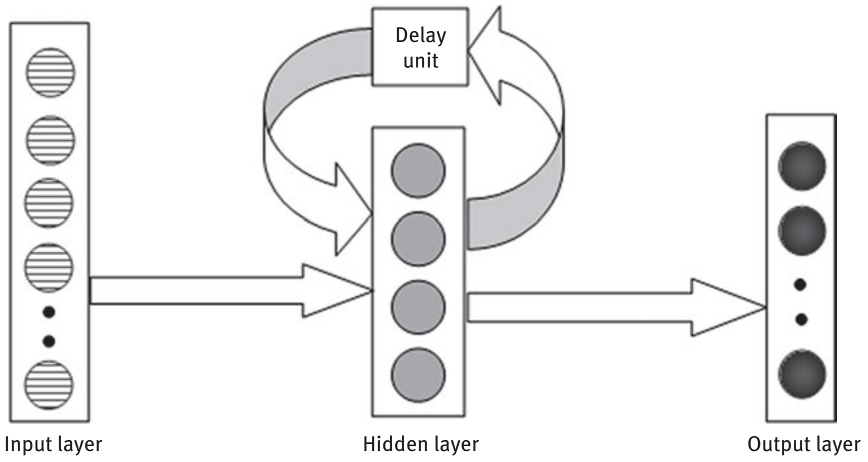


**Figure 3.4:** Basic architecture of RNN.

## 3.5 Implementation

Handwritten digit recognition has been applied for extraction of text contained in scanned documents or images. One classification algorithm is required for character recognition by their patterns. DNN has proved its excellent performance in the domain of machine learning [22, 23]. DNN consists of a huge number of hidden layers and connections. So, the number of trainable parameters in DNN is enormous. A massive number of samples are required for training the system to prevent overfitting. Whereas, CNN consists of a few hidden layers and lesser number of parameters compared to DNN. So, CNN takes lesser number of samples for training the system. CNN can change number of hidden layers and number of trainable parameters in every layer during formation of the architecture. Accordingly, the CNN architecture is planned for handwritten digits recognition. Here, the MNIST real world database is used to check the performance of the system.

### 3.5.1 Database used

In this chapter, MNIST handwritten digits database is used to train and test the system. The MNIST handwritten digit images have been normalized in size and are preprocessed. So, a user is not required for preprocessing the image. The MNIST training set and test set examples contained from approximately 250 writers. The MNIST database consists of 60,000 images of handwritten digit used for training and 10,000 images of handwritten digit used for testing [24]. Each handwritten digit image consists

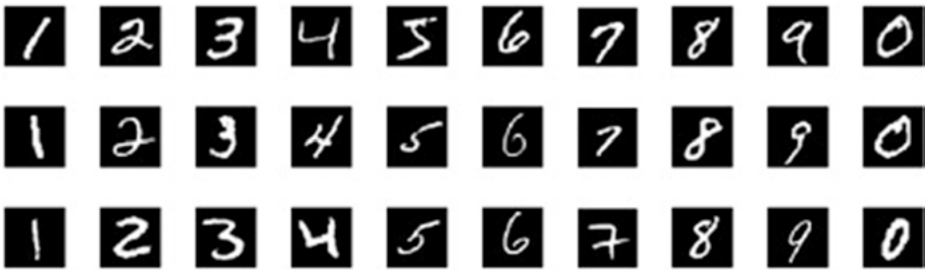


**Figure 3.5:** Examples of MNIST dataset.

of 28 by 28 pixels and each pixel has a value between 0 (background) and 255 (foreground) as shown in Figure 3.5. The database consists of training set image file, training set label file, test set image file, and test set label file. Pixel values are arranged row wise in every image file. In the image file, all the pixel values of an image are arranged in a row. So, each image file consists of 784 columns for 28 by 28 pixels. In the label files, the label values consist of 10 columns. For a particular instance of any digit, the label value is set by 1 for an appropriate column and set by 0 for the other columns.

### 3.5.2 Design of proposed CNN architecture

In this work, two CNN architectures are implemented for MNIST digit recognition. Each architecture consists of two hidden layers or convolution layers. The ReLU activation function is applied after every convolution operation for nonlinearity of the system. The first CNN architecture is named as “CNN Method 1” and it is implemented with the following details. The CNN Method 1 is implemented with two hidden layers or convolution layers for digit recognition of MNIST dataset. Input image size is 28 × 28, means total 784 numbers of pixels. So, the input layer has 784 numbers of inputs. Initially, a convolution operation is performed with 32 filters of size 5 × 5 without padding and the stride is equal to 1. The convolution operation gives the output of 32 layers with size 24 × 24. Then, a subsampling operation is done using a 2 × 2 max filter. This subsampling reduces 75% volume of the data and the generated output consists of 32 layers of size 12 × 12. After that, another convolution operation is performed with 64 filters of size 7 × 7 with zero-padding and the stride is equal to 1. The convolution operations produce the output of 64 layers with size 12 × 12. After that, another subsampling operation is done using a 2 × 2 max filter. This subsampling again reduces 75% volume of the data and the generated output consists of 64 layers of size 6 × 6. Then, an FCN layer is attached with 2,304 input nodes (64 × 6 × 6 nodes) and 1024 output nodes. Then, another FCN layer with 10 output nodes is attached for 10 digits.

The second CNN architecture is named as “CNN Method 2” and it consists of the following details. The CNN Method 2 is implemented with two hidden layers or convolution layers for digit recognition of MNIST dataset as shown in Figure 3.6.

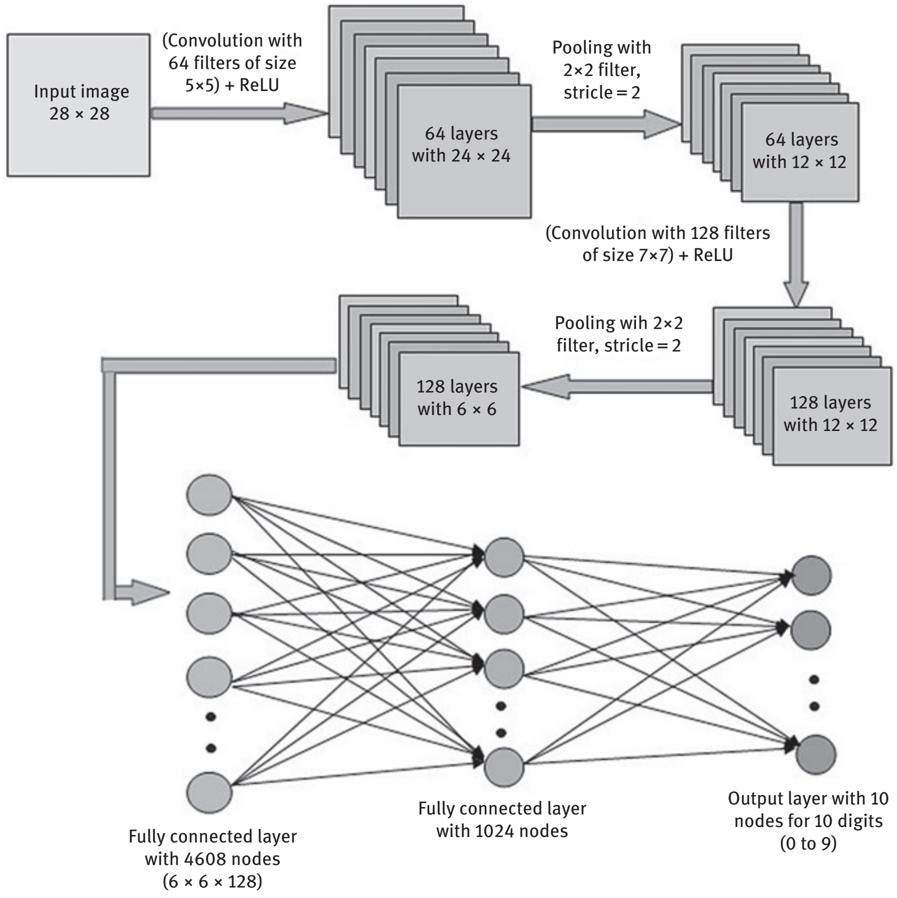

Input image 28 × 28
(Convolution with 64 filters of size 5×5) + ReLU
64 layers with 24 × 24
Pooling with 2×2 filter, stricle = 2
64 layers with 12 × 12
(Convolution with 128 filters of size 7×7) + ReLU
128 layers with 12 × 12
Pooling wih 2×2 filter, stricle = 2
128 layers with 6 × 6
Fully connected layer with 4608 nodes (6 × 6 × 128)
Fully connected layer with 1024 nodes
Output layer with 10 nodes for 10 digits (0 to 9)

**Figure 3.6:** Architecture of proposed CNN Method 2.

Input image size is 28 × 28, means a total 784 numbers of pixels. So, the input layer has 784 numbers of inputs. Initially, a convolution operation is performed with 64 filters of size 5 × 5 without padding and the stride is equal to 1. The convolution operation gives the output of 64 layers with size 24 × 24. Then, a subsampling operation is done using a 2 × 2 max filter. This subsampling reduces 75% volume of the data and the generated output consists of 64 layers of size 12 × 12. After that, another convolution operation is performed with 128 filters of size 7 × 7 with zero-padding and the stride is equal to 1. The convolution operations produce the output of 128 layers with size 12 × 12. After that, another subsampling operation is done using a 2 × 2 max filter. This subsampling again reduces 75% volume of the data and the generated output consists of 128 layers of size 6 × 6. Then, an FCN layer is attached with 4608 input nodes (128 × 6 × 6 nodes) and 1024 output nodes. Then, another FCN layer with 10 output nodes is attached for 10 digits.

## 3.6 Experimental result

Initially, an MLP is implemented for digit recognition. The MLP has 784 nodes in input layer as each image of MNIST has 784 pixels (28 × 28 pixels). The Xeon processor of 2.2 gigahertz with 128 GB RAM is used to train and test the system. MLP takes huge amount of time as compared to CNN and RNN as shown in Table 3.1.

**Table 3.1:** Accuracy of RNN and CNN.

| Steps | CNN implemented by Younis and Alkhateeb [23] | | RNN | |
|---|---|---|---|---|
| | Accuracy | Training time (in seconds) | Accuracy | Training time (in seconds) |
| 10,000 | 94.97% | 1354 | 86.67% | 148 |
| 20,000 | 97.03% | 2698 | 93.55% | 295 |
| 30,000 | 97.60% | 4050 | 95.99% | 443 |
| 40,000 | 97.99% | 5391 | 96.55% | 589 |
| 50,000 | 98.25% | 6738 | 97.01% | 736 |
| 60,000 | 98.41% | 8083 | 97.21% | 883 |
| 70,000 | 98.53% | 9436 | 97.30% | 1029 |
| 80,000 | 98.59% | 10782 | 97.55% | 1178 |
| 90,000 | 98.67% | 12132 | 97.79% | 1322 |
| 100,000 | 98.80% | 13,502 | 97.76% | 1,476 |
| 110,000 | 98.85% | 14,821 | 97.96% | 1,620 |

The experimental result of the MLP takes 5371 seconds to achieve 92.44% accuracy for MNIST digit classification. To achieve the same accuracy, 295 s is sufficient for training the RNN and 1354 s is sufficient for training the CNN implemented by

Younis and Alkhateeb. The comparative study for accuracy with respect to time shows that, RNN train the system faster compared to CNN as shown in Figure 3.7. But, CNN has the capability to train more and more if the training time is very high.

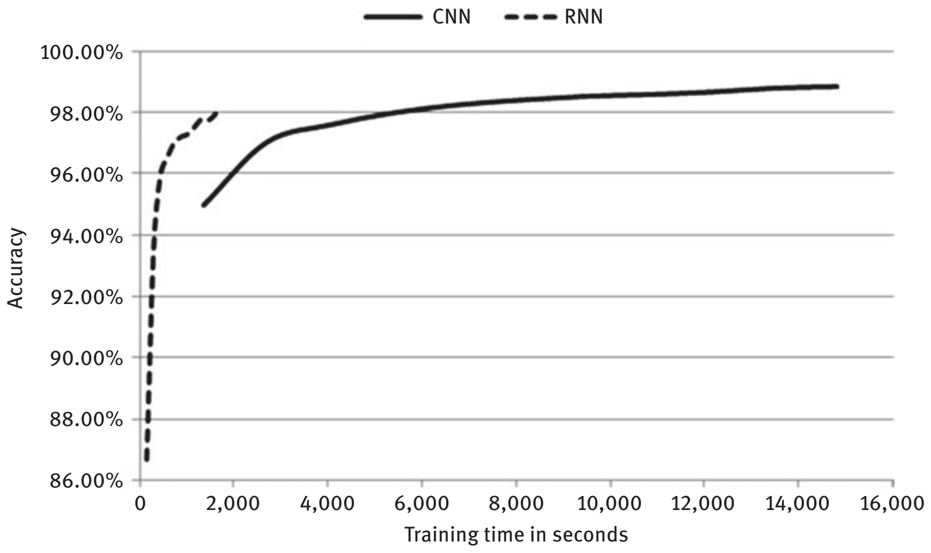


**Figure 3.7:** Accuracy vs. training time for CNN and RNN.

Experimental result shows the accuracy and training time with respect to number of steps for two proposed CNN methods with previous CNN method as shown in Table 3.2. The proposed “CNN Method 2” gives the higher accuracy (98.92%)

**Table 3.2:** Accuracy of previous CNN and proposed CNN.

| Steps | CNN Method 2 | | CNN Method 1 | | CNN implemented by Younis and Alkhateeb [23] | |
|---|---|---|---|---|---|---|
| | Accuracy | Training time (in seconds) | Accuracy | Training time (in seconds) | Accuracy | Training time (in seconds) |
| 10,000 | 95.37% | 2,337 | 95.20% | 796 | 94.97% | 1,354 |
| 20,000 | 97.13% | 4,613 | 96.99% | 1,571 | 97.03% | 2,698 |
| 30,000 | 97.92% | 6,867 | 97.70% | 2,342 | 97.60% | 4,050 |
| 40,000 | 98.23% | 9,196 | 98.05% | 3,132 | 97.99% | 5,391 |
| 50,000 | 98.42% | 11,456 | 98.29% | 3,907 | 98.25% | 6,738 |

**Table 3.2** (continued)

| Steps | CNN Method 2 | | CNN Method 1 | | CNN implemented by Younis and Alkhateeb [23] | |
|---|---|---|---|---|---|---|
| | Accuracy | Training time (in seconds) | Accuracy | Training time (in seconds) | Accuracy | Training time (in seconds) |
| 60,000 | 98.50% | 13,706 | 98.39% | 4,671 | 98.41% | 8,083 |
| 70,000 | 98.66% | 16,037 | 98.51% | 5,469 | 98.53% | 9,436 |
| 80,000 | 98.79% | 18,276 | 98.61% | 6,228 | 98.59% | 10,782 |
| 90,000 | 98.79% | 20,632 | 98.72% | 7,036 | 98.67% | 12,132 |
| 100,000 | 98.84% | 22,852 | 98.78% | 7,789 | 98.80% | 13,502 |
| 110,000 | 98.92% | 25,127 | 98.85% | 8,564 | 98.85% | 14,821 |

compared to previous CNN method and proposed “CNN Method 1” after 110,000 numbers of steps. But, the performance of proposed “CNN Method 1” is the fastest compared to others by considering training time as shown in Figure 3.8. Although, the parameters are different in previous CNN and proposed two CNNs, but the accuracy of all the systems are similar with respect to number of steps for training the system as shown in Figure 3.9. The prediction of the system appears incorrect for some cases due to the improper style of writing as shown in Table 3.3.

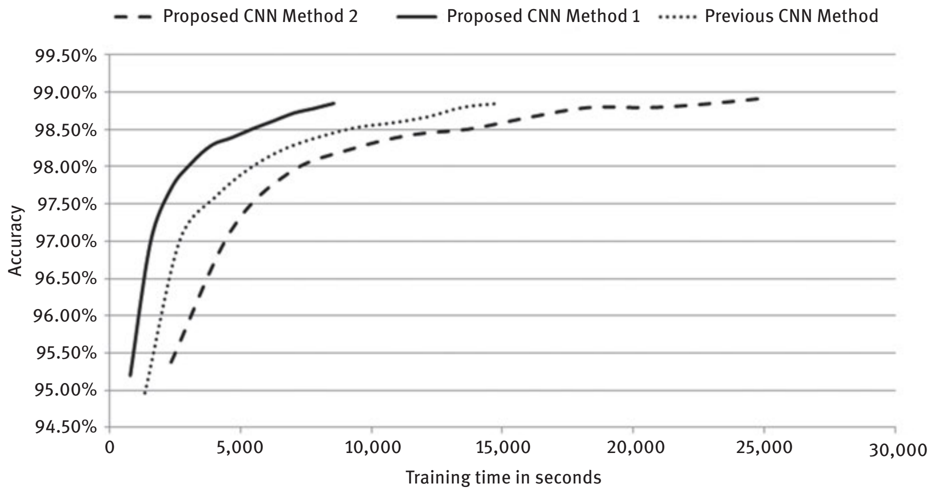


**Figure 3.8:** Accuracy versus training time.

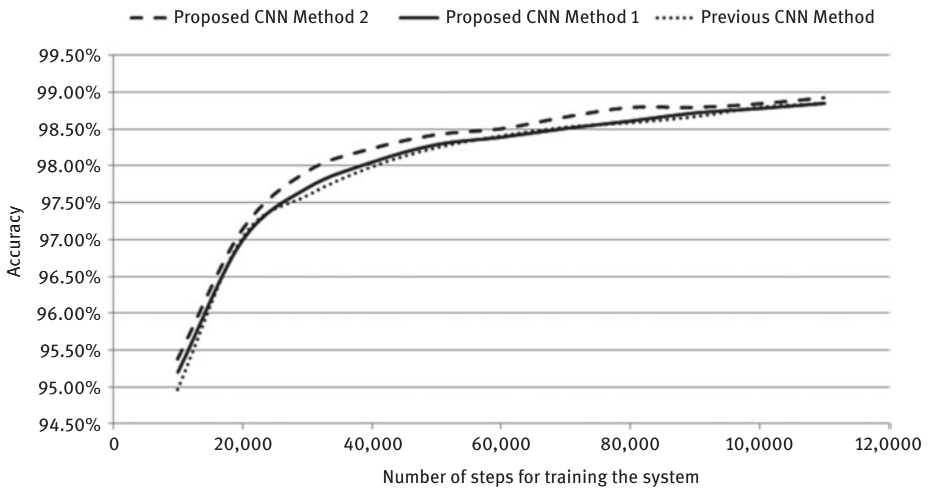
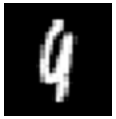
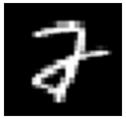
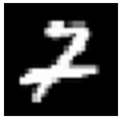
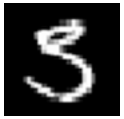
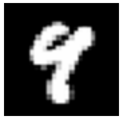
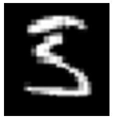
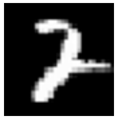
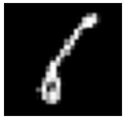


**Figure 3.9:** Accuracy vs. number of steps for training the CNN.

## 3.7 Conclusion

This chapter deals with MLP, RNN, and CNN for recognizing handwritten characters of MNIST character dataset. Implementations of handwritten digit recognition using two CNN methods are implemented using different parameters. From the experimental results, it is proved that CNN is more accurate for recognizing of handwritten characters. The proposed “CNN Method 2” provides the accuracy of 98.92% whereas the proposed “CNN Method 1” provides the accuracy of 98.85% for MNIST character dataset. The experimental result shows that, the proposed “CNN Method 2” is better in terms of accuracy and the proposed “CNN Method 1” is better in terms of training time as compared to others for the MNIST dataset. The accuracy of the system will be better by using more number of filters. But, the training time will be very high. For further improvement, it is required to increase the number steps for training the system.

**Table 3.3:** Some misclassified images.

| **Sample MNIST images** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| |  |  |  |  |  |  |  |  |
| Actual digit | 9 | 2 | 7 | 3 | 4 | 5 | 2 | 6 |
| Predicted digit | 4 | 7 | 2 | 5 | 9 | 3 | 7 | 1 |

## References

[1] Arica N. & Vural F.T.Y. An overview of character recognition focused on offline handwriting. IEEE Transactions on Systems, Man and Cybernetics – Part C: Applications and Reviews, 2001, 31(2), 216–233.

[2] Tappert C.C., Suen C.Y., & Wakahara T. The state of the art in online handwriting recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1990, 12(8), 787–808.

[3] Quiles M.G. & Romero R.A.F. A computer vision system based on multi-layer perceptrons for controlling mobile robots. International Congress of Mechanical Engineering, OuroPreto, Brazil, November, 2005.

[4] Ruck D.W., Rogers S.K., & Kabrisky M. Feature selection using a multilayer perceptron. Journal of Neural Network Computing, 1990, 2(2), 40–48.

[5] Yang J.B., Shen K.Q., Ong C.J., & Li X.P. Feature selection for MLP neural network: The use of random permutation of probabilistic outputs. IEEE Transactions on Neural Networks, 2009, 20(12), 1911–1922.

[6] Lee H., Grosse R., Ranganath R., & Ng A.Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations., International Conference on Machine Learning, ACM, 2009, 609–616.

[7] Vu N.T., Adel H., Gupta P., & Schutze H. Combining recurrent and convolutional neural networks for relation classification. NAACL HLT, 2016,534–539.

[8] Le Cun Y., Bottou L., Bengio Y., & Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998, 86(11),2278–2324.

[9] Krizhevsky A., Sutskever I., & Hinton G.E. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012,1097–1105.

[10] Le Cun B.B., Denker J.S., Henderson D., Howard R.E., Hubbard W., & Jackel L.D. Handwritten digit recognition with a backpropagation network. Advances in neural information processing systems, Citeseer, 1990.

[11] Arica N. & Vural F.T.Y. Optical character recognition for cursive handwriting. IEEE Transactions on Pattern Analysis and Machine Intelligence, June, 2002, 24(6), 801–113.

[12] Khedher M.Z., Abandah G.A., & Al-Khawaldeh A.M. Optimizing feature selection for recognizing handwritten Arabic characters. Proceedings of World Academy of Science Engineering and Technology, February 2005, 4.

[13] Hanmandlu M. & Ramana Murthy O.V. Fuzzy model based recognition of handwritten numerals. Pattern Recognition, 2007, 40, 1840–1854.

[14] Arora S., Bhattacharjee D., Nasipuri M., Basu D.K., & Kundu M. Combining multiple feature extraction techniques for handwritten Devnagari character recognition. IEEE Region 10 and the Third international Conference on Industrial and Information Systems, Kharagpur, India, December 2008.

[15] Kimura Y. Feature selection for character recognition using genetic algorithm. Fourth International Conference on Innovative Computing, Information and Control, IEEE, 2009.

[16] Graves A. & Schmidhuber J. Offline handwriting recognition with multidimensional recurrent neural networks. Advances in Neural Information Processing Systems 22, NIPS'22, Vancouver, MIT Press, 2009, 545–552.

[17] Pal A. & Singh D. Handwritten English character recognition using neural. Network International Journal of Computer Science & Communication, July–December, 2010, 1(2), 141–144.

[18] Pradeep J., Srinivasan E., & Himavathi S. Diagonal based feature extraction for handwritten alphabets recognition system using neural network. International Journal of Computer Science & Information Technology (IJCSIT), February 2011, 3, 1.

[19] Neves R.F.P., Filho A.N.G.L., Mello C.A.B., & Zanchettin C. A SVM based off-line handwritten digit recognizer. International Conference on Systems, Man and Cybernetics, IEEE, Brazil, October 2011, 510–515.
[20] Gaurav K. & Bhatia P.K. Analytical review of preprocessing techniques for offline handwritten character recognition. International Conference on Emerging Trends in Engineering & Management, 2013.
[21] Hu G., Yang Y., Yi D., Kittler J., Christmas W., Li S.Z., & Hospedales T. When face recognition meets with deep learning: An evaluation of convolutional neural networks for face recognition. IEEE International Conference on Computer Vision Workshops, Santiago, Chile, 13–16 Dec 2015, 142–150.
[22] Ghazi M.M. & Ekenel H.K. A comprehensive analysis of deep learning based representation for face recognition., IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2016, 34–41.
[23] Younis K.S. & Alkhateeb A.A. A new implementation of deep neural networks for optical character recognition and face recognition., Proceedings of the New Trends in Information Technology, Jordan, 25–27 April 2017.
[24] LeCun Y., Cortes C., & Burges C.J.C. http://yann.lecun.com/exdb/mnist/

Amit Adate, Dhruv Arya, Aditya Shaha, and B.K. Tripathy

# 4 Impact of Deep Neural Learning on Artificial Intelligence Research

**Abstract:** Deep learning techniques have had a huge impact on artificial intelligence research. They have improved upon the traditional machine learning techniques where human expertise was required for feature engineering. By removing one human factor, they have moved us one step forward in the field of artificial intelligence. They have not entirely removed humans, though. They are required for designing the architectures and cleaning the data. Deep learning techniques have managed to achieve breakthrough results in domains such as speech recognition, machine translation, image recognition, and object detection. This chapter gives a brief overview of various deep learning techniques being used today. Techniques that make deep learning more effective have been described. Some interesting applications have also been covered.

**Keywords:** deep learning, artificial intelligence, artificial neural networks, deep learning applications

## 4.1 Introduction

Modeling the functions of the human brain is the core challenge of artificial intelligence research. The human brain is able to create complex models of the world that it observes and extrapolates from it. Furthermore, it is very robust in its functioning. It only takes the brain a few experiences to learn about a new entity. By looking at a banana once, humans are able to recognize bananas in different shapes and sizes. Artificial intelligence researchers have been trying to tackle such problems for decades now. The increase in computational and storage capacity has allowed researchers to come up with methods that deal with very high dimensional inputs. Deep learning research has also been helped by the availability of high-quality datasets. This chapter provides a brief overview of deep learning approaches. These have been discussed keeping in mind the impact of deep learning on artificial intelligence research and applications. First, some deep learning models are briefly described, and then some techniques that have led to the success of deep learning have been discussed. Next, some frameworks that aid the development of deep learning models have been presented,

**Amit Adate, Dhruv Arya, Aditya Shaha,** School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India.
**B. K. Tripathy,** School of Information Technology and Engineering, Vellore Institute of Technology, Vellore, India.

https://doi.org/10.1515/9783110670905-004

after which some interesting applications are covered. Finally, some areas of research of deep learning are presented which might become the focus of the coming years.

## 4.2 Convolutional neural models

Convolutional neural networks (CNN) are a class of deep neural networks (DNN) which were built with the motive of achieving minimal preprocessing. Although they have performed exceptionally well in the domains of image processing they can be used to analyze any time-series data. The success of the CNNs can be attributed to the comparatively lesser preprocessing required by them which reduces human effort and any requirement of the domain knowledge to hand-engineer the features. The connectivity pattern between the neurons in a CNN is inspired from animal visual cortex in which a neuron only responds to a stimuli in a fixed region of the visual field known as the receptive field. In the convolution operation, the first argument is the input (in the formula the function *x*) and the second argument is a kernel (in the formula *w*). The output is referred to as a feature map. Instead of handcrafting features for an input space, we learn the convolution operation weights that work as feature extractors:

$$P_{\text{new}} s(\mathrm{t}) = \sum_{a=0}^{n} x(a) w(\mathrm{t} - a) \tag{4.1}$$

which can be equivalently written as

$$s(\mathrm{t}) = (x^* w)(\mathrm{t}) \tag{4.2}$$

where * is the convolution operator.

CNNs differ from the conventional neural networks in a way that they have the properties of sparse connectivity, weight sharing and translation equivariance [1]. Recently CNNs have been successfully applied in many domains such as object detection, image classification, segmentation, text classification, speech recognition, and so on.

## 4.3 Recurrent neural networks

Most machine learning models have not been designed with sequential data in mind. That is not to say that they do not work with sequential data at all. Support vector machines, fully connected feedforward neural networks, and so on can be trained to work with sequential data of fixed size. However, that is where the problem lies, it makes it difficult for us to deal with data that has long-term dependencies. Most texts have such dependencies in them. Neural networks trained for classifying texts of length 100 would not work with texts of length 200. Furthermore, most existing models have been designed while assuming that the data points are independent of each other. Clearly, this is not a valid assumption for valid data where dependencies exist between all units.

Recurrent neural networks (RNNs) have managed to overcome these problems to an extent. As the name implies, the computational units in these networks have a recurrent connection. These networks process sequential data one unit at a time. They maintain a set of activations for each step and are able to selectively pass information across these steps. They have been designed while keeping sequential data in mind.

Let the input sequence of length $T$ be represented by $X = \{x1, x2, \ldots, xt, \ldots, xT\}$. A general RNN is represented by the following equation:

$$s_{\mathrm{t}} = F(s_{\mathrm{t}-1}, x_{\mathrm{t}}, \theta) \tag{4.3}$$

Here $s_t$ is the internal state of the RNN at time $t$ and $\theta$ is the set of weights of the RNN. A more specific representation is given by

$$s_t = W_{\mathrm{rec}}\sigma(s_{\mathrm{t}-1}) + W_{\mathrm{in}}x_{\mathrm{t}} + b \tag{4.4}$$

where $\sigma$ is a nonlinear function that operates elementwise over the input. The parameter $\theta$ from the general case is made up of the weight matrices $W_{\mathrm{rec}}$, $W_{\mathrm{in}}$, and the bias $b$. Generally, $s_0$ is initialized to zeros. It is clear from the equation how the current state is dependent on both the current input $x_t$ and the state from the previous time-step $s_{t-1}$. The amount by which these two influence the current state is directly dependent on the weights $W_{\mathrm{in}}$ and $W_{\mathrm{rec}}$ respectively.

While RNNs worked well, it was shown in [2] that they did not capture long-term dependencies well. Furthermore, RNNs are tricky to train. They are also trained using backpropagation. In their case, we backpropagate the gradients over the sequence, thus we call it backpropagation through time. The gradients are computed by unrolling the network into a multilayered network. They suffer from the exploding and vanishing gradients problems where the gradient either becomes too large or too small for the initial layers in the unrolled network. The exploding gradients problem can be dealt with by clipping the gradients which go over a certain threshold [3]. However, vanishing gradients are more difficult to deal with. This is mainly because the gradients for short-term dependencies are high while only the gradients for long-term dependencies are low. This is the reason that classic RNNs cannot model long-term dependencies. A variant of RNNs called long short-term dependency networks were introduced in [4]. This variant worked by explicitly adding operations to allow hidden state modification, retention, and deletion. This makes it particularly suitable for handling long-term dependencies, thus resolving the vanishing gradients problem.

Traditionally, n-gram models have been used for text processing. Hidden Markov models (HMMs) have been used for the same. However, HMMs work on an underlying assumption that text sequences can be modeled as Markov models, that is, the next state that we can land is only dependent on the previous state. In most applications, an *n*-gram is taken as one state. However, this model fails to capture long-term dependencies in texts. Furthermore, HMMs become computationally difficult to solve as the number of hidden states increase. RNNs are more general than HMMs and their variants, LSTMs, can capture long-term dependencies. This has made them the go-to model for tasks such as text classification.

## 4.4 Generative adversarial networks

Generative models, in general, learn the underlying probability distribution of the task. Generation is simply the sampling of data points from the learned distribution. Suppose that the underlying probability distribution of some data $D$ is $p_D$, the task of a generative model $M$ is to estimate this distribution in the form of $p_M$.

One recently introduced generative model is the generative adversarial network (GAN) [5]. The model works by using two competing neural networks, the generator, $G$, and the discriminator, $D$. The generator tries to model the probability distribution of the training data $p_D$ as $p_G$ while the discriminator learns to distinguish between samples generated by the generator and samples from the training data. Both the networks are initialized randomly and perform very poorly at their respective tasks in the beginning. Both networks become better at their tasks over time, however, as the modeled distribution $p_G$ becomes very similar to $p_D$, the discriminator starts randomly predicting genuine or fake with equal probability. This analogous to an amateur painter learning to imitate the painter Van Gogh while another person simultaneously learns to spot fake Van Goghs. When the amateur painter becomes as good as Van Gogh, the other person will not have any criteria to distinguish these paintings.

Formally, the generator is a differentiable function $G$ and the discriminator is a differentiable function $D$. Let $z$ be some noise sampled from probability distribution $p_z$ and $p_D$ is the probability distribution of the training data. The generator and discriminator minimax game can be represented by the following equation:

$$\min_{\mathrm{G}} \max_{\mathrm{D}} V(D, G) = \mathbb{E}_{x \sim PD}(x)[\log D(x)] + \mathbb{E}_{x \sim p_{(x)}}[\log(1 - D(G(z)))] \tag{4.5}$$

Radford et al. [6] have shown how CNNs can be used as the generator and discriminator functions. They removed all pooling layers and used transposed convolution for upsampling in the generator. This resulted in impressive results in generative tasks related to images. The use of CNNs has become very common in the last few years in GAN research.

GANs suffer from mode collapse. The loss function that we used earlier does not ensure that the GAN is able to generate a variety of samples. It might learn to generate a few samples that satisfy its loss function in that it beats the discriminator. This is highly undesirable. GANs are very tricky to train in other ways too. Sometimes they fail to converge and the model parameters keep on oscillating. If the discriminator becomes too good at its job, the gradient for the generator diminishes and it stops learning. It is essential to balance the number of times the discriminator is trained for each iteration of training the generator. Wasserstein GANs, introduced in [7], managed to overcome some of these problems by using a different loss function. Conditional GAN, introduced in [8], use the data class in-formation in the discriminator. This allows them to capture better multimodal data representations. Zhu et al. introduced Cycle GANs in [9] which allow mapping from one image domain to another even when no paired datasets exist for those domains. For example, this allows us to convert the image of a red car to that of a blue car even when we do not have an image of that particular car model in blue. A thorough overview of GANs can be found in [10].

## 4.5 Variational autoencoders

Variational autoencoders (VAEs) [11] are another type of a generative model. Autoencoders are networks where we have an encoder–decoder structure. The encoder takes a data point and maps it to a smaller vector which is called the latent representation of the data. Then, a decoder takes this smaller vector and maps tries to map it back to the original data point. Both the encoder and the decoder are trained together by minimizing the mean squared error between the original input and the output of the decoder. The encoder learns to compress that data and the decoder learns to decompress it.

Now, if we take any random vector of the same dimensions as the latent representation and pass it to the decoder, we should get a data point from the space of the original data. We are effectively performing a generative task. However, the latent space is not continuous, and the data reconstructed by the decoder for many random vectors will be unrealistic. Essentially, variational autoencoders are autoencoders that ensure that the activations of latent representation (hidden layer) are continuous. This is done by making the encoder output two vectors of dimension $d$ instead of one vector of dimension $d$. These two vectors represent means and standard deviations and are represented by $\mu$ and $\sigma$, respectively. We then sample a random vector $H$ of dimension $d$, and perform the following operation on each element $h_{\mathrm{i}}$:

$$z_{\mathrm{i}} = \sigma_{\mathrm{i}} {}^{*} h_{\mathrm{i}} + \mu_{\mathrm{i}} \tag{4.6}$$

We get a vector $Z$ which is then decoded back by the decoder. Note that this vector $Z$ will not be the same even when the same input is given at different times. This is because the sampled vector $H$ will differ. For the training, the Kullback–Leibler divergence between the latent representation and unit Gaussian is minimized, the mean squared error between the output and the input is also minimized simultaneously. The overall flow has been visualized in Figure 4.1. A good tutorial of VAEs can be found in [12].

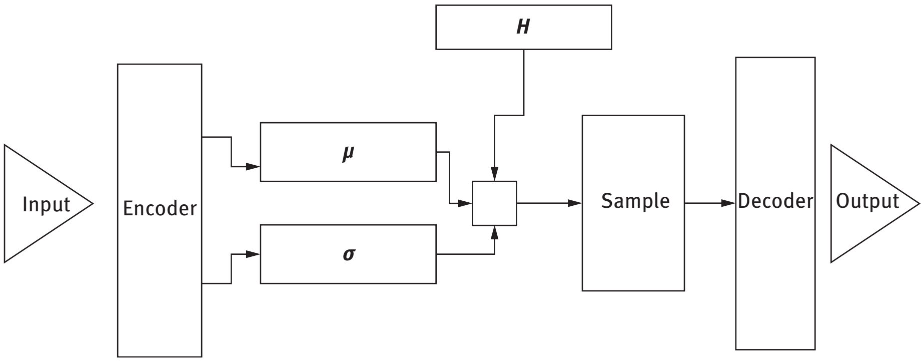


**Figure 4.1:** Variational autoencoder.

## 4.6 Deep reinforcement learning

The reinforcement learning literature is rife with algorithms that guarantee convergence to the optimal solution when they are used with look-up tables. These reinforcement learning algorithms are analyzed for the cases when the underlying Markov decision processes (MDP) have finite number of states and actions. However, the look-up tables do not scale well for high-dimensional MDP which has continuous (infinite) states or actions because of the curse of dimensionality. So, function approximators like sigmoidal, multilayer perceptron, a radial basis function, or a memory based learning network must be used.

In [53], Widrow et al. introduced a reinforcement learning variant of ADALINE Algorithm, in which an adaptive logic unit learned from the feedback the decision functions that are not necessarily linearly separable. Barto, Andrew G. et al. proposed in [54], that decision-making tasks with delayed consequences can be accurately formulated as sequential decision problems and solved by dynamic programming. They showed that the adaptive neural networks (ANN) will be successfully used for approximating the functions required for solving such problems. The dynamic programming methods, in case when the states are finite and discrete, depend on the look-up table representation of the evaluation functions to find the value of the states in successive iterations. But when the state space is continuous, the discretization of state space results in exponential-states in the state space. However, with the advancement in the knowledge of Deep learning, ANNs are being used for generalized functional approximation.

## 4.6.1 Deep Q-networks

One such important learning algorithm that is successfully used to learn control policies from images using reinforcement learning and neural networks as function approximators was deep Q-learning. It used a CNN with a variant of Q-learning. The input to this model was images in the form of raw pixels and, as an output, it predicted the value function which was used to estimate the future rewards. The problem with applying deep learning directly as functional approximators is that:

1. Deep learning requires large amounts of hand-labeled data while reinforcement learning uses scalar rewards that are usually sparse, noisy, and delayed.
2. Most deep learning algorithms assume that the data is being sampled from independent and identical distributions, however in reinforcement learning the rewards received are highly correlated and the data distribution changes as the agent learns new behaviors.

Deep Q-networks (DQN) used the experience replay mechanism that randomly sampled from previous transitions to solve the problem of correlated data and nonstationary distribution. The experience replay stores the agents experience at each time step, $\epsilon_t = (s_t, a_t, r_t, s_{t+1})$ in a dataset $D = \epsilon_1, \ldots \epsilon_N$ which is collected over many episodes into the replay memory[13].So, instead of performing updates after every reward signal, deep Q-learning uses mini-batch updates drawn at random from the experience replay. Once the updates are performed the agent selects and performs actions according to the $\epsilon - greedy$ policy.

Performing the batch updates addresses many of the problems faced by directly applying Q-learning with CNNs as functional approximators.

1. It is efficient to perform batch updates.
2. As there exists a strong correlation between data, it is inefficient to learn from consecutive samples. We are able to break the correlations by sampling randomly and hence the variance of the updates is reduced.
3. Traditional methods use on-policy learning in which the current parameters determine the next data sample on which the parameters are trained on. However, when the training distribution changes while learning, there is a higher risk of the parameters being stuck in the local minima. Experience replay averages the behavior distribution over many of the previous states which smoothens learning and avoids a suboptimal solution. Experience replay uses off-policy learning.

Deep Q-learning algorithm uniformly samples from the experience replay. With this method of sampling, transitions are replayed at the same frequency with which they were experienced irrespective of their significance. In [14], the authors developed a framework in which the important transitions were replayed more frequently which made the learning more efficient. One of the important concept that needs to be addressed in this update would be how we can decide the importance of the transition. The importance of the replay can be defined by how much the agent learnt by taking the transition from its current state. This can be measured using magnitude of the transition's TD error $\delta$ which indicates how much the value differed from the expected value. However, the TD error can be poor estimates when the underlying rewards are noisy.

Policy search methods are reinforcement learning that aims at directly finding the underlying policies by means of gradient-free or gradient-based methods. Traditionally, evolutionary algorithms were used to solve these problems which are gradient-free policy search algorithms. In evolutionary methods, the performance of a population of agents is considered for evaluating the performance. These methods give successful results in case of small population but become intractable when there are large numbers of agents or agents have multiple parameters. However, evolutionary models can be used to train large networks if the neural networks weight representation can be compressed. In [15], the authors represented the networks weights as Fourier-type coefficients, thus, using a compressed encoding for weights. This method was successfully used for training for tasks that have very large networks owing to the high dimensional input space.

## 4.7 Making deep learning work

### 4.7.1 Regularization

The quality and quantity of the crops are always important for the countries where a large percentage of its population depends on agriculture. As the models are becoming more complex, overfitting has become more common. Overfitting happens when the model fails to generalize to test data even though it gives good performance on the test data. If we keep other things the same, overfitting increases with increase in complexity of the model. While $L1/L2$ regularization has been used a lot, two other techniques that are now being used are covered below:

1. Dropout: Dropout [16] is a regularization technique in which network connections are dropped with a certain probability $p$. This ensures that the network does not become too reliant on those connections.
2. Batch normalization: Batch normalization [17] works by first calculating the mean and the standard deviation of the activations of hidden layers in the network for the batch. The activations of the batch are then normalized by subtracting the mean from them and dividing them by the calculated standard deviation. Two new parameters are introduced in these layers $y$ and $\beta$. These are used to scale and shift the normalized activations $\hat{x}$:

$$y = \hat{x}\gamma + \beta \tag{4.7}$$

3. This also has a small regularizing effect on the network.

### 4.7.2 Transfer learning

The success of deep learning has been possible partly due to the availability of huge datasets and massive amounts of computational power. Deep neural architectures such as VGG [18] are useless without a large dataset to train on. However, large datasets are often not available in domains where data collection is difficult or expensive. Transfer learning has enabled deep learning to work with small datasets. It works on the assumption that deep learning models learn to extract reusable features implicitly when trained on a task. When two domains are similar, a model trained on the dataset of one domain may be used for the other domain with some fine-tuning. Common transfer learning approaches involve first training a model, typically a deep CNN, on a large dataset such as ImageNet [19]. This enables the CNN to learn filters that can extract features that are common across many image-oriented tasks. The last fully connected layer of the network is then changed with another one with the number of output neurons equal to the number of classes in the target dataset. Then, the network is retrained on the target dataset. During the retraining phase, the number of layers that are trained can vary. One may choose to just train last layer and use the rest of the network as a feature extractor. The whole network can be retrained too. Transfer learning has given some impressive results in the last decade.

## 4.8 Deep learning frameworks

Deep learning research has been made easier with the rise of many deep learning frameworks. The network designer does not have to explicitly specify how the gradients are to be computed, these frameworks automate this task for them. Most frameworks covered here are open source. Furthermore, most of these rely on the CUDA [20] for GPU processing. All these frameworks use some form of automatic differentiation. Some popular deep learning frameworks have been briefly described below:

1. Theano: Theano [21] is an open source project which was developed at the Montreal Institute for Learning Algorithms. It was under active development since 2008 till 2017 when its final version 1.0.0 was released. Essentially, the library allows definition and execution of mathematical expressions on both GPU and CPU. Like most other libraries covered here, it has been implemented

in C++. It provides an API that is similar to the popular linear algebra framework NumPy for Python.

2. Tensorflow: Tensorflow [22] was designed while keeping deployment of machine learning models in mind. It has inbuilt support for distributing the workload over multiple computational nodes. In Tensorflow, a computational graph must first be defined and compiled. Later, the data can be fed and the graph is executed.
3. Keras: Keras [23] was built to make it easier to experiment with deep learning models. It has several deep learning architectures inbuilt. It provides a higher layer of abstraction than frameworks like Tensorflow or Theano. Keras is only a wrapper, and the actual computation is delegated to the underlying computational library. The users can select either Theano or Tensorflow for the backend.
4. PyTorch: PyTorch [24] is a relatively new framework which was developed at Facebook AI research. It is similar to Keras in that it was designed to be easy to experiment with. Unlike Tensorflow, PyTorch does not require the user to compile a computational graph first before execution. While PyTorch was not developed with deployment to production in mind, recent releases have added features to make PyTorch models more suitable for production. These additions have come in the form of a just-in-time (JIT) compiler that allows computational graphs to be compiled and optimized before execution, just like Tensorflow. Another important feature of PyTorch is that it allows neural networks to be defined in an object-oriented manner.
5. Others: Many other deep learning frameworks have been developed in the last few years. The computational network toolkit [25] was developed by Microsoft. Some other notable frameworks are MXNet [26 27].

## 4.9 Engineering applications of deep learning

Success of deep learning techniques in a range of engineering applications has been well documented. DNNs are being tailor-made for natural language processing, speech recognition, object detection, and so on. In this section, we have covered some interesting engineering applications that have experienced advances due to deep learning.

There remains a scope of research on finding the optimal number of clusters from a true color image and in future the quantum inspired algorithms will open up the door to resolve the multiobjective optimization problem efficiently within a short time frame.

1. Style transfer: Style transfer involves repainting an image in the style of another painting. While this was the domain of trained artists in the past, CNNs have made it more accessible. Before deep learning, algorithms had to be constructed

for every specific style. Gatys et al. [28] showed in their seminal work how the style of an image can be captured separately from the content of the image. In particular, they use CNNs as feature extractors where the gram matrix of the activations of any particular layer represents the style of the input image with respect to that layer. It is understood that as we go deeper into a CNN, more semantic features are captured. They used activations of the earlier layers of network to use as a reference for content. Style transfer in this method involves optimizing a randomly initialized image to have a similar activation gram matrix to the style image and similar activations in the earlier layers to a desired content image. Note that the network was only used as a feature extractor and was not optimized. One shortcoming of this approach is that it takes a lot of time, as for each content image, the optimization process has to be carried out again. Johnson et al. [29] have used the methods presented by Gatys et al. to train neural networks to do style transfer. In this approach, one CNN is used for feature extraction and is not optimized while the CNN is used to convert a content image to a stylized image. The loss is computed by using the feature extraction network and the style transfer network is optimized. This approach is much faster than [28] as style transfer only requires one passage through a CNN.

2. Super resolution: It is the task of obtaining a higher resolution (HR) image for a given set of low resolution input images. Single image super resolution is an interesting problem where we only have one input image. The interesting thing is that there is no correct answer as such because the information regarding the missing pixels is mostly not captured in the input. One input image can correspond to many high-resolution output images. In [30], the authors have presented a model called super-resolution CNN that can be trained end to end for Single image super resolution. They first upscale the input image using bicubic interpolation and then pass it to a CNN which gives us a feature map in the final layer that has the same dimensions as that of the desired HR image. The CNN is trained to minimize the mean squared error between the output and the target. The authors improved their model in [31] in which they processed all three channels of the input image simultaneously. Kim et al. [32] proposed the usage of the same convolutional layer repeatedly multiple times. They show that although such a network is difficult to train, the performance benefits are significant.
3. Automatic speech recognition: Automatic speech recognition is used to automatically map audio to text. Graves et al. [33] have used a deep RNN architecture to accomplish this task. They have used a connectionist temporal classification loss [34] which is useful as it allows us to train RNNs where the input and the output sequence alignment are not known. This happens in speech recognition where multiple timesteps of an audio clip correspond to the same letter. Zhao et al. [35] have presented an architecture that combines RNNs and CNNs for automatic speech recognition. This enables the network to capture both frequency and temporal dependence of the input. The network

is trained on spectrograms of the speech which makes CNNs suitable for the job.

4. Image recognition, object detection, and segmentation: CNNs have been used as feature extractors to find similarities between faces in [36]. This has been used to do face recognition. The network was trained to map an image to a vector such that images of the same person should have vectors near each other while images of different people should have vectors that are far away from each other. When deployed, the network maps the input image to the vectorized representation and the distance is computed between it and the stored vectors. One important thing to note here is that network does not need to be trained specifically for faces that are to be recognized. Availability of high quality datasets such as COCO [37], ImageNet [19], and PASCAL VOC [38] have been an impetus for object detection and segmentation research. Fully CNNs presented in [39] have shown breakthrough results for object segmentation. Region-based convolutional network [40] was seminal in the field of object detection. YOLO [41] and Fast R-CNN [42] are both faster during inference. Single shot detection (SSD) [43] is another faster approach that builds upon the others and has shown impressive results.
5. Text to speech synthesis: Deep learning has made synthesis of natural sounding audio from text a reality. Earlier, to generate somewhat natural sounding audio, a big database for each type of sound had to be painstakingly maintained. Wavenet [44] was a breakthrough work that used a fully CNN to synthesize audio from text. The network was conditioned on the identity of the speaker whom it was being trained on which allowed a single network to have multiple types of voices. In [45], the authors have presented another audio synthesis approach that uses RNNs instead of convolutional networks. They use a layered architecture where the networks at different layers are sampling the input at a different temporal resolution. Tacotron [46] is another neural audio synthesis model that uses encoder–decoder architecture.
6. Machine translation (sequence to sequence): The encoder–decoder approach for mapping a sequence of inputs to another sequence using recurrent networks presented in [47] has been seminal in the field of machine translation. Neural machine translation approaches have become common. In [47], two LSTM, a variant type of RNN, networks are used – an encoder and a decoder. The encoder takes a text sequence and iteratively maps it to a vector. Then, the decoder takes this vectorized representation of the input sequence and generates a new sequence iteratively. The proposed method performs well for English to French translation tasks. One important benefit of this technique is that the model can be trained end to end. This approach is not perfect and it fails to work for long sequences. In [48], the authors reason that a fixed-length vector might not be able to capture all the information needed for translation. They propose that activations generated for each input unit in the input sequence be

stored and referred when decoding. They use an attention mechanism that allows the decoder prioritize activations that are more relevant to the word being generated. This model can also be trained end to end.

7. Playing Video Games (Atari games): In [13], Minh et al. showed that DQN model was successfully able to achieve human-level performance across 41 out of 49 Atari games. DQN uses deep Q-learning algorithm a variant of Q-learning algorithm which uses experience replay to solve the problems faced by neural Q-learning. In [14], the authors extended DQN with prioritized replay in which the frequency of replay in updates depended upon its TD error.
8. Work in and reasoning: Most of the literature on reasoning and knowledge representation used symbolic logic-based methods. Currently, research is being done to train a DNN model that will be able to perform logical reasoning in the form of basic ontology reasoning [49].

## 4.10 Conclusion

A lot of research is being done in the field of deep learning. This has mostly been encouraged by the impressive results deep learning is achieving in all the domains it is being applied to.

With the increase in the usage of DNNs for various applications, there is an increased need for practitioners who can design task-specific architectures. This has also generated interest in neural architecture search and hyper parameter optimization algorithms which automate this process. Most of neural architecture search techniques use either evolutionary algorithms or reinforcement learning. In particular, [50] presents how neural architectures can be searched using reinforcement learning. One exciting approach in this field is differentiable architecture search [51] which as the name implies assumes that the architecture space is continuous. This allows the usage of gradient-based techniques.

One recent advance in the field has been the introduction of capsule networks [52]. This neural network architecture that builds upon CNNs manages to learn equivariant representations. That is, they can easily deal with the input even when the image is scaled, rotated, or skewed. Research is being done to make the neural decision-making process more understandable. Deployment of black boxes such as neural networks in safety-critical applications raises several concerns. The field of deep learning is very exciting due to the rapid advances being made in it. Breakthrough results are being achieved very consistently due to new architectures, training techniques, and datasets.

## References

[1] Goodfellow I., Bengio Y., Courville A., & Bengio Y. Deep learning, Vol. 1, Cambridge, MIT press, 2016.

[2] Bengio Y., Simard P., & Frasconi P. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 1994, 5(2), 157–166.

[3] Pascanu R., Mikolov T., & Bengio Y. Understanding the exploding gradient problem. CoRR, abs/1211.5063, 2012.

[4] Hochreiter S. & Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8), 1735–1780.

[5] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., & Bengio J. Generative adversarial nets. Advances in Neural Information Processing Systems, 2014, 2672–2680.

[6] Radford A., Metz L., & Chintala S. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.

[7] Arjovsky M., Chintala S., & Bottou L. Wasserstein gan. arXiv preprint arXiv:1701.07875, 2017.

[8] Mirza M. & Osindero S. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784, 2014.

[9] Zhu J.-Y., Park T., Isola P., & Efros A.A. Un-paired image-to-image translation using cycle-consistent adversarial networks. arXiv preprint, 2017.

[10] Creswell A., White T., Dumoulin V., Arulkumaran K., Sengupta B., & Bharath A.A. Generative adversarial networks: An overview. IEEE Signal Processing Magazine, 2018, 35(1), 53–65.

[11] Kingma D.P. & Welling M. Auto-encoding variational bayes. arXiv preprint. arXiv:1312.6114, 2013.

[12] Doersch C. Tutorial on variational autoencoders. arXiv preprint arXiv:1606.05908, 2016.

[13] Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., & Riedmiller M. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[14] Schaul T., Quan J., Antonoglou I., and Silver D. Prioritized experience replay. In the International Conference on Learning Representations (ICLR), 2016.

[15] Koutnk J., Cuccu G., Schmidhuber J., & Gomez F. Evolving large-scale neural networks for vision-based reinforcement learning. Proceedings of the 15th annual conference on Genetic and evolutionary computation. ACM, July 2013, 1061–1068.

[16] Hochreiter S. & Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8), 1735–1780.

[17] Ioffe, S., & Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32nd International Conference on International Conference on Machine Learning, 2015, 37, 448–456.

[18] Simonyan K. & Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint, arXiv:1409.1556, 2014.

[19] Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z. et al. Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 2015, 115(3), 211–252.

[20] Nickolls J., Buck I., Garland M., & Skadron K. Scalable parallel programming with CUDA, ACM SIGGRAPH 2008 classes, ACM, 2008, 16.

[21] Bergstra J., Breuleux O., Bastien F., Lamblin P., Pascanu R., Desjardins G., & Bengio Y. Theano: A CPU and GPU math compiler. Python. In Proc. 9th Python in Science Conf, June 2010, 1.

[22] Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M. et al. Tensorflow: a system for large-scale machine learning. OSDI, 2016, 16, 265–283.

[23] Chollet F. Keras, 2015.
[24] Paszke A., Gross S., Chintala S., & Chanan G. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. 2017.
[25] Yu D. & Huang X. Microsoft Computational Network Toolkit (CNTK). A Tutorial Given at NIPS. 2015.
[26] Chen T., Li M., Li Y., Lin M., Wang N., Wang M., Xiao T., Xu B., Zhang C., & Zhang Z. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274, 2015.
[27] Jia Y., Shelhamer E., Donahue J., Karayev S., Long J., Girshick R., Guadarrama S., & Darrell T. Caffe: Convolutional architecture for fast feature embedding. Proceedings of the 22nd ACM international conference on Multimedia. ACM, 2014, 675–678.
[28] Gatys L.A., Ecker A.S., & Bethge M. A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576, 2015.
[29] Johnson J., Alahi A., & Fei-Fei L. Perceptual losses for real-time style transfer and super-resolution. European Conference on Computer Vision. Springer, Cham, 2016, 694–711.
[30] Dong C., Change Loy C., He K., & Tang X. Learning a deep convolutional network for image super-resolution. European conference on computer vision. Springer, Cham, 2014, 184–199.
[31] Dong C., Change Loy C., He K., & Tang X. Image super-resolution using deep convolutional networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016, 38(2), 295–307.
[32] Kim J., Kwon Lee J., & Mu Lee K. Deeply-recursive convolutional network for image super-resolution. Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, 1637–1645.
[33] Graves A., Mohamed A., & Hinton G. Speech recognition with deep recurrent neural networks. Acoustics, Speech and Signal Processing. IEEE, 2013, 6645–6649.
[34] Graves A., Fernández S., Gomez F., & Schmidhuber J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. Proceedings of the 23rd international conference on Machine learning. ACM, 2006, 369–376.
[35] Zhao Y., Jin X., & Hu X. Recurrent convolutional neural network for speech processing. Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference. March 2017, 5300–5304.
[36] Koch G., Zemel R., & Salakhutdinov R. Siamese neural networks for one-shot image recognition, ICML Deep Learning Workshop, 2015, 2.
[37] Lin T.-Y., Maire M., Belongie S., Hays J., Perona P., Ramanan D., Dollár P., & Zitnick C.L. Microsoft coco: Common objects in context. European conference on computer vision, Springer, Cham, 2014, 740–755.
[38] Everingham M., Van Gool L., Williams C.K., Winn J., & Zisserman A. The pascal visual object classes (voc) challenge. International Journal of Computer Vision, 2010, 88(2), 303–338.
[39] Long J., Shelhamer E., & Darrell T. Fully convolutional networks for semantic segmentation. Proceedings of the IEEE conference on computer vision and pattern recognition. 2015, 3431–3440.
[40] Girshick R., Donahue J., Darrell T., & Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, 580–587.
[41] Redmon J., Divvala S., Girshick R., & Farhadi A. You only look once: Unified, real-time object detection. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, 779–788.
[42] Girshick R. Fast r-cnn. Proceedings of the IEEE international conference on computer vision. 2015, 1440–1448.

[43] Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C., & Berg A.C. Ssd: Single shot multibox detector. European conference on computer vision. Springer, Cham, 2016, 21–37.
[44] Van Den Oord A., Dieleman S., Zen H., Simonyan K., Vinyals O., Graves A., Kalchbrenner N., Senior A., & Kavukcuoglu K. Wavenet: A generative model for raw audio, CoRR abs/1609.03499, 2016.
[45] Mehri S., Kumar K., Gulrajani I., Kumar R., Jain S., Sotelo J., Courville A., & Bengio Y. SampleRNN: An unconditional end-to-end neural audio generation model. arXiv preprint, arXiv:1612.07837, 2016.
[46] Wang Y., Skerry-Ryan R.J., Stanton D., Wu Y., Weiss R.J., Jaitly N., Yang Z. et al. Tacotron: Towards end-to-end speech synthesis. arXiv preprint arXiv:1703.10135, 2017.
[47] Sutskever I., Vinyals O., & Le Q.V. Sequence to sequence learning with neural networks. Advances in Neural Information Processing Systems, 2014, 3104–3112.
[48] Bahdanau D., Cho K., & Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.
[49] Hohenecker P. &Lukasiewicz T. Ontology Reasoning with Deep Neural Networks. arXiv preprint arXiv:1808.07980, 2018.
[50] Bello, I., Zoph, B., Vasudevan, V., & Le, Q. V. Neural optimizer search with reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning. 2017, 70, 459–468.
[51] Liu H., Simonyan K., & Yang Y. Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055, 2018.
[52] Sabour S., Frosst F., & Hinton G.E. Dynamic routing between capsules. Advances in Neural Information Processing Systems, 2017, 3856–3866.
[53] Widrow, Bernard, Narendra K. Gupta, and Sidhartha Maitra. "Punish/reward: Learning with a critic in adaptive threshold systems." IEEE Transactions on Systems, Man, and Cybernetics 5, 1973: 455–465.
[54] Barto, Andrew G., Richard S. Sutton, and Christopher JCH Watkins. "Sequential decision problems and neural networks." Advances in neural information processing systems. 1990.

Rajib Saha, Anirban Mukherjee, Avik Sarkar, and Shatabhisa Dey

# 5 Extraction of Common Feature of Dysgraphia Patients by Handwriting Analysis Using Variational Autoencoder

**Abstract:** Nowadays, handwritten document analysis using intelligent computing technology is a demanding research area, considering its usefulness in identifying a person and human characteristics, particularly that of persons having typical disabilities such as dyslexia, dysgraphia, and Parkinson's disease. Analysis of handwriting, falling under the broad purview of graphology, helps us understand the writer's psychology, emotional outlays, and noticeable disorders as well. Since there prevails a broad spectrum of cursive nature and high inconsistency of handwriting styles, the techniques for modern handwriting analysis need to be more robust and sensitive to different patterns compared to the traditional graphological techniques. Herein lies the necessity of computing technology, which should intelligently analyze handwritten texts to find out the similarity of finer aspects of handwritings of children or adult with some kind of learning/writing disability. Deep learning technology is chosen as the technical tool to identify and classify common features of handwriting of children with developmental dysgraphia. Variational autoencoder, a deep unsupervised learning technique, is presently used for this purpose. This chapter reports successful extraction and interpretation of significant number of distinguishable handwriting characteristics that are clinically proved to be symptoms of dysgraphia.

**Keywords:** feature extraction, dysgraphia, handwriting analysis, variational autoencoder

## 5.1 Introduction

There are so many electronic gadgets/tools available today, which are commonly used as digital platform for reading and writing, and as a medium of communication. Yet today, traditional medium like printed books, study materials, and handwritten documents are widely used and quite popular as learning or communication medium. Communication through writing on paper by hand remains the most basic mode though writing on digital screen by stylus is another useful mode. Therefore, handwriting skill needs to be acquired by one and all in their tender age as a part of

**Rajib Saha, Anirban Mukherjee, Avik Sarkar, Shatabhisa Dey,** RCC Institute of Information Technology, Kolkata, West Bengal, India.

https://doi.org/10.1515/9783110670905-005

the basic education. Handwriting is a complex task involving perceptual, attention-related, linguistic, and finer motor skills. Supposedly, handwriting is the foundation of all other educational errands such as jot notes, self-expression, and composition. The academic and behavioral progress of children is closely associated with prompt and significant development of reading and writing skills. Handwriting of every individual evolves with time on both qualitative and quantitative terms. Despite proper handwriting training, a significant percentage of children in the world, fail to grasp the skill and encounter difficulties in establishing their handwriting. This results in diminution of cognitive performance, weakening of self-esteem and so on. Therefore, detection and remediation of any kind of difficulty in handwriting at the earliest possible stage holds crucial importance.

Most individuals facing conspicuous handwriting challenges owing to motor or sensory-motor weakness or disorder are considered to have dysgraphia [1], a neurological disorder. The prefix "dys" implies the presence of impairment while the root word "graph" refers to the hand's function in writing the letters similarly, the suffix "ia" means having a condition. Hence, dysgraphia can be defined as the unique condition of impaired letter writing by hand, that is, disabled handwriting.

The term dysgraphia is often attributed to individuals having deficits in the following aspects in their writings:

I. Accuracy in spellings
II. Grammatical and punctuation accuracy
III. Clarity or organization of written expression

This neurodegenerative disorder commonly develops when children are in the initial stage of their introduction to handwriting. It can also occur following severe neurological trauma or may be found in a patient with physical impairments like ADHD, Tourette syndrome, learning disabilities, or an autism spectrum disorder such as Asperger's syndrome. Apart from these, an individual can be suffering from dysgraphia without having any other disabilities. They are believed to be the victims of the cursed genes.

Some of the general symptoms of dysgraphia are as follows:

I. Lack of consistency in the case (upper/lower) of letters
II. Sizes and shapes of letters are irregular and often they are incomplete
III. Odd writing grip
IV. Several mistakes in spellings (sometimes)
V. Significant change in speed of writing and copying
VI. Poor use of lines and spaces
VII. Reluctance or refusal to complete writing tasks
VIII. Experience of physical pain in hand and/or arm while writing

Dysgraphia can be widely classified into five types namely, dyslexic dysgraphia, motor dysgraphia, spatial dysgraphia, phonological dysgraphia, and lexical dysgraphia. As far as the present study is concerned, dyslexic, motor, and spatial dysgraphia are of primary interest as these are characterized by poor handwriting. Dyslexic dysgraphia affects an individual's spontaneous writing and spelling. Motor dysgraphia is characterized by poor writing (both spontaneous and copied), bordering on illegible and slanted writing which may be attributed to lack of fine motor skills. Spatial dysgraphic patients often face difficulty in maintaining uniform skew for lines and uniform gap between words. However, patients may suffer from multiple type of dysgraphia and the symptoms may differ from individual to individual.

VAE (variational autoencoder) has been used in this chapter. VAE [2] is nothing but a neural network model that immediately adapts and adopts an inference model for deduction of imperceptible features of image data and initiates a generative model to produce images from those deduced features. Earlier, authors [3] have successfully generated different clusters of characters having similar handwriting features using MNIST, a handwritten numeric dataset. Moreover, VAE can produce characters when alphabet dataset is used having more classes than MNIST. Hence, the same process has been applied on the current dataset to generate the generative model. However, the features of the handwritings in the dataset could not be perfectly captured since they are more complex when compared with the handwritten numerals of MNIST, as the character image was mapped into a one-dimensional vector.

## 5.2 Background

Except a few unfortunate children, majority of children are able to master the handwriting skills essential for copying or penning down their own thoughts. Their handwriting is intelligible and the handwriting process [4, 5] is carried out with comfort and ease. It is by the age of six a child begins to get habituated with the form of the letters and an approximate 2 years of formal education provides handwriting experience, which leads to stable and effective handwriting by the age of 8 [6]. Now children who fail to adopt and develop handwriting proficiency by that age faces developmental dysgraphia [7, 8].

According to prior studies, it is believed that the presence of dysgraphia is common among children with average IQ level and is also found among those who have not been diagnosed with any neurological or perceptual motor disorders.

The risk of developmental dysgraphia in children with neurodevelopmental disabilities is very high [9]. In school-going children the risk is comparatively low and the percentage lies between 10 and 30 [8]. It is clinically proved that handwriting difficulty is capable of making serious impact on an individual's overall academic performance, psychology, and behavior [10]. Thus, it is important to detect developmental dysgraphia at the earliest, to prevent uncontrollable and undesired manifestation in the adult life [11].

Scientific studies substantiate the utility of an automated system for recognizing the handwriting process and proceed with assessing the same to distinguish handwriting features of children suffering from dysgraphia [8, 12–14]. However, as perceived, concepts of deep learning will throw more light on the aspects of features and would be more effective to diagnose and identify handwriting defects among patients of dysgraphia.

Previously, a system of automatic rating of dysgraphia based on HPSQ total score estimation was developed using the concept of machine learning which implemented handwriting parameterization techniques and intrawriter normalization approach.

For the first time, in this chapter, VAE, a deep learning technique to compress data, is used to identify common handwriting features in dysgraphia patients. Deep learning is a subset of unsupervised learning, also termed as hierarchical learning which has created huge impact on the field of machine learning [15]. AE typically comprises dense, fully connected layers of a feedforward neural network; it consists of an encoder network and a decoder network. The encoder is compared to a convolution neural network (CNN). The input is processed using several hidden layers "h" inside an autoencoder. The input is encoded into a much smaller representation as an output. The output received from the particular layer consists of sufficient suitable information to be processed in the next layer of the network to produce necessary output format. It simultaneously assimilates an inference model to deduce the latent features of image data and comprehends a generative model for generating images from those deduced latent features. The input image is encoded and decoded by VAE. Following this, clustering is implemented which may lead to loss of information. The similarities and dissimilarities among the various characters contained in the dataset are depicted as the output of clustering of characters. Identification of handwritings manually is comparatively easier than of identification done by machines since handwriting varies from individual to individual.

VAE is considered as a neural network which can encrypt image into a direction in the latent space of $z$ real numbers. For the sample that has been collected from a $z$-dimensional normal distribution, the direction is assumed to be arbitrary. Then, the decoder network decodes the encoded vector representation and obtains the original image. Random samples can be drawn from the distribution and fetch them into decoder network for which the latent space is $z$-dimensional normal distribution. From this new images can be obtained that are absent in the dataset we trained on. The architecture of the encoder is represented in Figure 5.1.

The pivotal facet of the VAE lies in the fact that its latent representation, $y \in \mathbb{P}^{K}$ is derived from a particular Gaussian distribution (i.e., $p(y) = N(y| \mu, \Sigma)$, where $\mu$ denotes mean and $\Sigma$ denotes covariance). The VAE is trained in an unsupervised manner as the output is basically the reconstructed version of the input. Let us

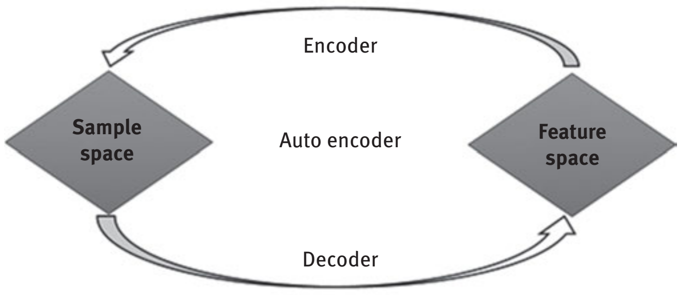


**Figure 5.1:** The simplest form of representation of the encoder architecture.

consider the given set of training data is $S = \{s_n\}_{n=1}{}^{\boldsymbol{N}}$, the VAE comprises of a probabilistic encoder $q\theta(y|s)$, that finds the latent representation $y$ for the given set of input data $s$. It also consists of a probabilistic decoder $p\phi(s|y)$ which reconstructs the input data for the specific latent representation, where $\theta$ represents the network parameter for encoder and $\phi$ represents the network parameter for decoder.

Optimization of the variational bound $L_{\text{VAE}}(\theta, \varnothing)$ with respect to the parameters $\theta$ and $\phi$ in the encoder and decoder occurs learning VAE.

$$L_{\text{VAE}}(\theta, \varnothing) = \sum_{n-1}^{n} -E_{y\sim}\, q\theta(y|s_n)[\log p\varnothing(s_n|y)] + KL(q\theta(y|s_n)||p(y)) \tag{5.1}$$

Here, the first term is the reconstruction error that has been computed by taking the expectation with respect to the distribution of $y$ while the second term denotes regularizer which is the Kullback–Leibler (KL) divergence between the estimated distribution $q\theta(y|s_n)$ and the true distribution $p(y)$. This divergence is the measurement of the amount of information lost while using $q$ to represent $p$. The estimation of parameters can be done using an autoencoding variational Bayes algorithm [16].

The pictorial representation of an encoder and decoder is given in Figure 5.2(a) and (b), respectively.

The encoder is said to be a neural network whose input is a data point s while output is a hidden representation $y$, having weights and biases $\theta$. Specifically saying, let us consider $y$ to be a 15 by 15 pixel photo of a handwritten digit. The encoder "encodes" the data which is 225-dimensional into a latent (hidden) representation space y, which is much less than 225 dimensions. This is conventionally termed to be a "bottleneck" because the encoder must be capable of efficient compression of the data into this lower-dimensional space.

From Figure 5.2, we can consider or assume the encoder to be denoted as $q\theta(y|s)$. We know that the lower-dimensional space is arbitrary: the parameters to $q\theta(y|s)$, which is a Gaussian probability density, is the output of the encoder. Sampling of this distribution can be done to obtain the noisy values of the representations $y$.

y
Encoder
$q\theta(y|s)$
Data: $S$
(a)
y
Decoder
$p\phi(s|y)$
Reconstruction: $\hat{S}$
(b)

**Figure 5.2:** (a) The encoder is used to compress data into latent space ($y$). (b) The decoder is used for reconstruction of given data which is a hidden representation.

The decoder is also a neural network having representation y as the input, while the parameters to the probability distribution of the data is its output, and has weights and biases $\phi$. The decoder has been denoted by $p\phi(s|y)$. On execution with the handwritten digit as example, let us assume that the images are black and white and representation of each pixel is in the form of 0 or 1 (Figure 5.3).

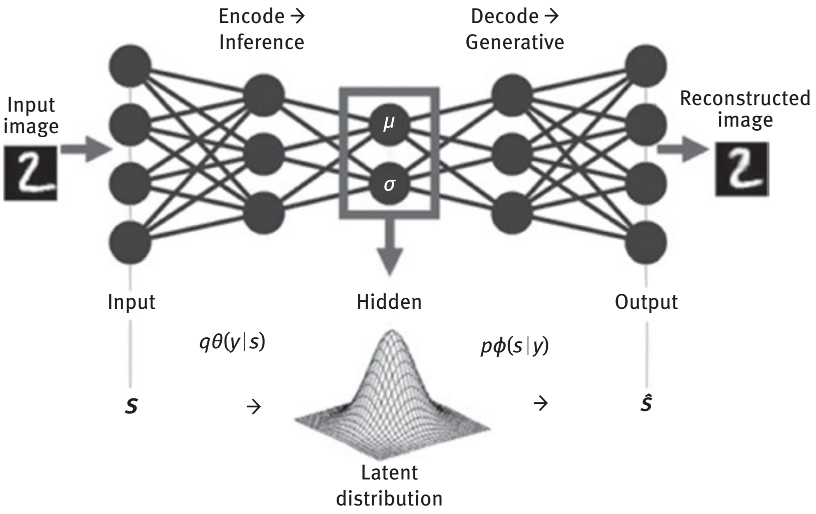


**Figure 5.3:** Illustration of the cited example.

The probability distribution of a single pixel can be represented using a Bernoulli distribution. The latent representation of a digit y is given as input to the decoder and the output, thus, obtained is 225 Bernoulli parameters, one for each of the 225 pixels in the image. Decoding the real-valued numbers in y into 225 real-valued numbers between 0 and 1 is done by the decoder. There is the occurrence of loss of information as it goes from a smaller to a larger dimensionality [17].

The requirements of VAE are as follows:

I. It is essential to use two tractable distributions:
   a. The initial distribution $p(y)$ must be easy to be sampled from.
   b. The conditional likelihood $p(s|y,\theta)$ should be computable.
II. In practice, this implies that the two distributions are seldom complex, for example, uniform, Gaussian, or even isotropic Gaussian.

Once the image is encoded and decoded in the VAE, it involves reconstruction of the images. For extraction of similar characteristics/features, clustering needs to be performed.

Hierarchical clustering is a statistical cluster analysis process which involves building of clusters in a hierarchical form. Here, splitting and merging are determined in a greedy manner. After hierarchical clustering the results thus obtained are represented in a dendrogram. In case a set of $M$ items with an $M \times M$ distance (or similarity) matrix needs to be clustered, then the following needs to be performed for the process of hierarchical clustering:

1) First, each M item is assigned to its own cluster; that is, in case of M items there would be M clusters formed, each containing single item. Let us assume, that the distances (with similarities) between the clusters is equivalent to the distances (similarities) between the items they have.
2) Next, the closest (most similar) pair of clusters needs to be calculated and merged into a single cluster, so that there is a decrease in the number of clusters by 1.
3) Next, the distances (similarities) between the newly formed clusters and each of the old clusters need to be calculated.
4) Steps 2 and 3 are repeated until all items are clustered into a single cluster of size M.

The uniqueness between VAE and a standard autoencoder is that they have continuous latent spaces allowing easy random sampling and interpolation. This property makes them effective generative models. This avoids producing unrealistic output due to discontinuous latent space.

## 5.3 Proposed work and methodology

The chapter purports to thoroughly study the handwritings that have been taken from children suffering from dysgraphia. With the help of deep learning as the technical tool, a number of characteristics or patterns uniquely present in the samples collected were deduced. These distinguishable characteristics are further recognized as symptoms of dysgraphia in the field of medicine. Hence, dysgraphia can be detected at an early stage, which would in turn be helpful in taking up remedial measures like therapeutic care.

The very initial step of this study was collection of handwriting of children suffering from dysgraphia and then those collected documents were scanned to proceed further. The assumptions taken into consideration are:

- Scanning has been performed perfectly, hence the introduced skew is the responsibility of the writer
- Salt and pepper noise [18, 19] and background noise [18, 19]might be present, which may have been caused before or after the scanning process

The steps of the proposed work are as follows:

### 5.3.1 Preprocessing

To maximize the efficiency of processing and to reduce complicacy in the terminal stages of the code, image preprocessing [20, 21] is done to obtain image of better quality. Preprocessing steps performed for handwriting analysis includes binarization [22, 23] and noise removal. Median filter technique [24] is adopted for salt and pepper noise removal [18, 19, 24–26] while image binarization is done using Otsu's [22] thresholding technique. In binarization, binary image is obtained from grayscale image.

### 5.3.2 Line segmentation

The binary image obtained is used for line segmentation. The present work implements conversion of image into grayscale which is then binarized inversely, resulting in a dark background with the text in a light shade. Resultant image is dilated and **findContours()** function of OpenCV library is used to find the contours. Each of the contours found is stored as a vector of points. Function RETR_EXTERNAL returns only the extreme outer contours, which is the implemented mode of retrieval. In CHAIN_APPROX_SIMPLE function, the horizontal, vertical, and diagonal segments are compressed and reduced to their end points only, as done in the adopted approximation method. For example, an upright triangular contour is encoded with 3 points.

**Algorithm for line segmentation**

STEP 1: Load the sample writings as images.
STEP 2: Convert image into grayscale.
STEP 3: Convert the grayscale image into binary image using proper threshold and invert the image.
STEP 4: Dilate the binary image using kernel of 5 × 150 matrix of ones.
STEP 5: Find contours from the diluted image and consider only the extreme outer flags.
STEP 6: Extract the desired region of the image using those contours and save them as images which are basically lines in the writings.

### 5.3.3 Word segmentation

Word segmentation is performed by applying the abovementioned method on the segmented lines and the characters are thereby segmented using each individual word.

**Algorithm for word segmentation**

STEP 1: Load the sample lines as images.
STEP 2: Convert image into grayscale.
STEP 3: Convert the grayscale image into binary image using proper threshold and invert the image.
STEP 4: Dilate the binary image using kernel of 5 x 35 matrix of ones.
STEP 5: Find contours from the diluted image and consider only the extreme outer flags.
STEP 6: Extract the desired region of the image using those contours and save them as images which are basically words in the line.

### 5.3.4 Character segmentation

The characters are thereby segmented using each individual word using the same method as that of line segmentation.

**Algorithm for character segmentation**

STEP 1: Load the sample words as images.
STEP 2: Resize the contour containing the word using bicubic interpolation over 4 × 4 pixel neighborhoods.
STEP 3: Convert image into grayscale.
STEP 4: Convert the grayscale image into binary image using proper threshold and invert the image.

STEP 5: Dilate the binary image using kernel of 5 x 5 matrix of ones.
STEP 6: Find contours from the diluted image and consider only the extreme outer flags.
STEP 7: Extract the region of interest from the image using those contours and save them as images which are basically characters in the words.

### 5.3.5 Implementation of VAE

The images thus obtained after character segmentation is fed to the VAE where the image is reconstructed after encoding and decoding.

The steps that are followed are mentioned below:

STEP 1: The image is read.
STEP 2: The image is converted to desirable format for required channels and size.
STEP 3: The image pixel intensity value is stored in an array.
STEP 4: The array is shaped to coincide with the input shape of the VAE model.
STEP 5: The data is split into training and test set.
STEP 6: The encoder model is constructed using fully connected layers are used in the model to calculate mean and variance.
STEP 7: The decoder, a sequential model with fully connected layers is constructed.
STEP 8: The input and output shapes for the encoder and decoder are used to construct the autoencoder.
STEP 9: The model is compiled using negative log normal loss function.
STEP 10: The middle-hidden layer represents the mean and variance layer using KL. The model is trained as a whole and the mean and variance is updated for every batch of data in each epoch using back propagation.

Once the result is obtained from the VAE, clustering is done. Clustering facilitates grouping of similar attributes within the same cluster. The resultant cluster would, thus, hold the clear picture that would be beneficial for deriving important conclusions out of it. In an ideal scenario, it is expected that if we form clusters of the images of all the English alphabets, we would obtain 26 different clusters in all. But in case of handwriting analysis of children suffering from dysgraphia, it is certain and predictable that the number of clusters for all 26 alphabets would be less than 26. This significant decrease in the number of clusters formed would pave the path to conclude the obvious defective characteristics that is present in the handwriting.

The entire methodology is illustrated in the flowchart shown in Figure 5.4.

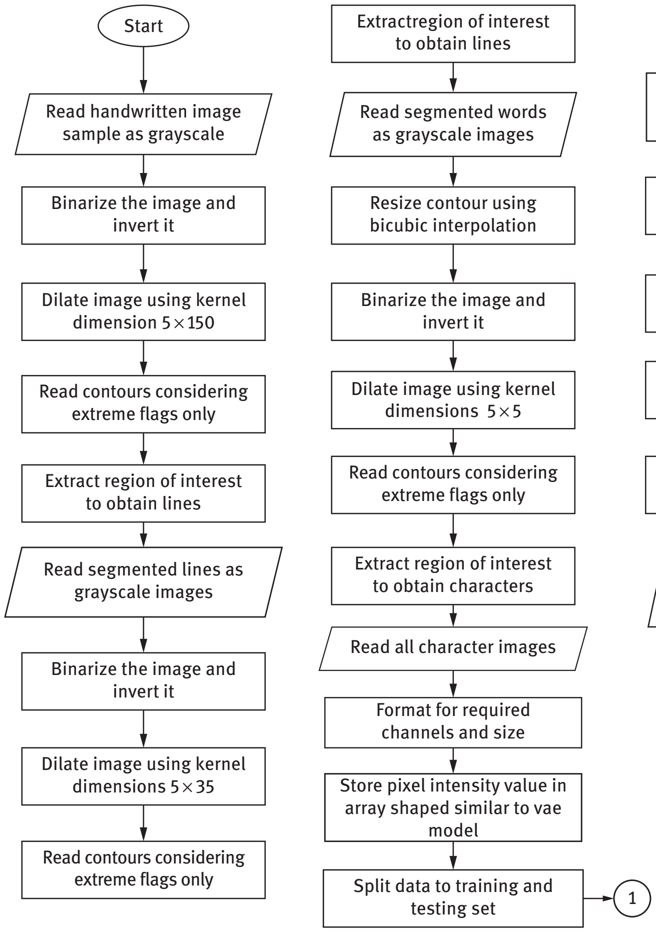

Start
Read handwritten image sample as grayscale
Binarize the image and invert it
Dilate image using kernel dimension 5×150
Read contours considering extreme flags only
Extract region of interest to obtain lines
Read segmented lines as grayscale images
Binarize the image and invert it
Dilate image using kernel dimensions 5×35
Read contours considering extreme flags only
Extractregion of interest to obtain lines
Read segmented words as grayscale images
Resize contour using bicubic interpolation
Binarize the image and invert it
Dilate image using kernel dimensions 5×5
Read contours considering extreme flags only
Extract region of interest to obtain characters
Read all character images
Format for required channels and size
Store pixel intensity value in array shaped similar to vae model
Split data to training and testing set
1
1
Construct sequential encoder and decoder models
Merge models to create vae model
Compile using negative log normal loss function
Fit the model for training and testing set
Use recons tructed image for clustering
Plot the clustered images
Stop

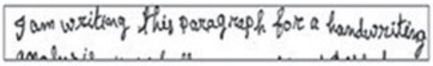
**Figure 5.4:** Flowchart of the entire process.

## 5.4 Results and discussion

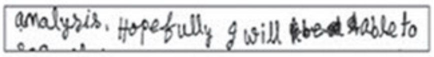
The outputs of each step involved in this chapter are represented with sample outputs followed by discussion on inferences.

### 5.4.1 Line segmentation output

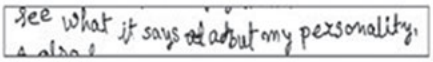
As discussed previously, line segmentation is performed as the first major step. Few of the segmented lines obtained from handwriting samples of dysgraphia patients are shown in Figure 5.5(a)–(h).


I am writing this paragraph for a handwriting

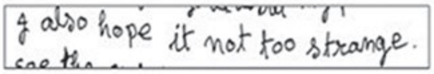
5.5(a)


analysis. Hopefully I will ~~be~~ able to

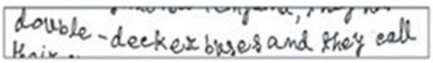
5.5(b)


see what it says ~~ab~~ about my personality.

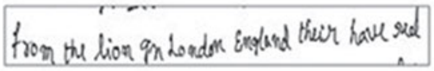
5.5(c)


I also hope it not too strange.

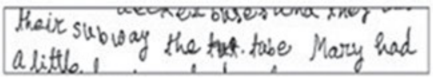
5.5(d)


double-decker buses and they call

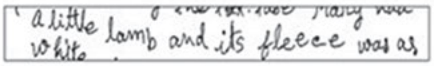
5.5(e)


from the lion in London England their have red

5.5(f)


their subway the ~~tut~~ tube Mary had
a little

5.5(g)


a little lamb and its fleece was as
white

5.5(h)

**Figure 5.5:** Snapshot of line segments (a)–(h).

### 5.4.2 Word segmentation output

After the lines are segmented, it is used as input for the next step which is word segmentation. The word segmentation algorithm processes the segmented lines and divides it further into smaller segments. The segmented words for each patient are stored, few samples of which are shown in Figure 5.6.

### 5.4.3 Character segmentation output

Each segmented word is further segmented to obtain each character present in the handwritten samples. Figure 5.7 shows a snapshot of character segmentation output containing few segmented handwritten characters.

### 5.4.4 Output of reconstruction using VAE

The segmented characters were passed through the VAE to extract the common features. The VAE takes the segmented characters as input and reconstructs the images using only relevant information, by intelligent selective discarding of irrelevant information. This reconstruction as shown in Figure 5.8 is done using an encoder and decoder network. After 500 epochs, the result is obtained over a training sample set of 1,050 characters; the loss is 0.2573 and validation loss is 0.2648.

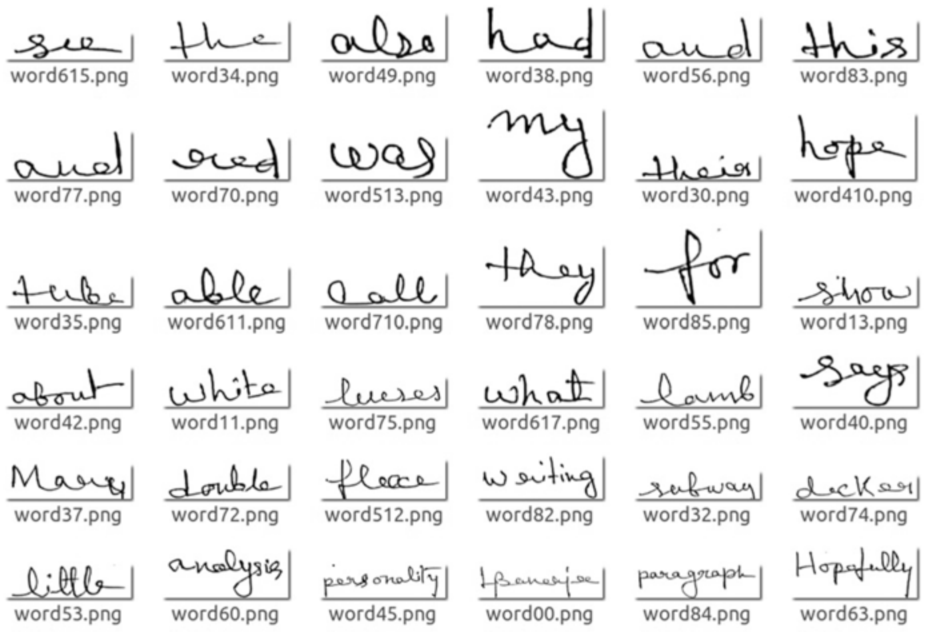


**Figure 5.6:** Snapshot of word segments.

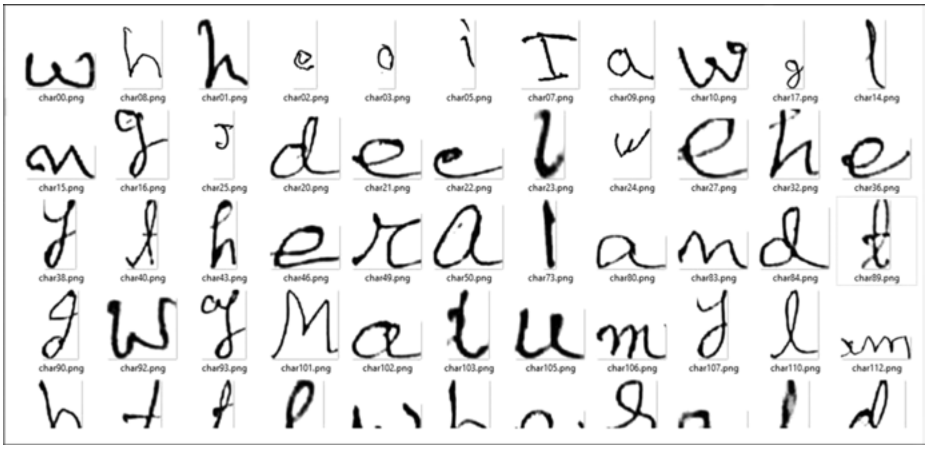


**Figure 5.7:** Snapshot of character segmentation output.

The images after reconstruction are clustered in order to loosely group them such that the images have close similarities with each other when depicted as data points in a two-dimensional plane. Such clustering samples are shown in Figures 5.9–5.12.

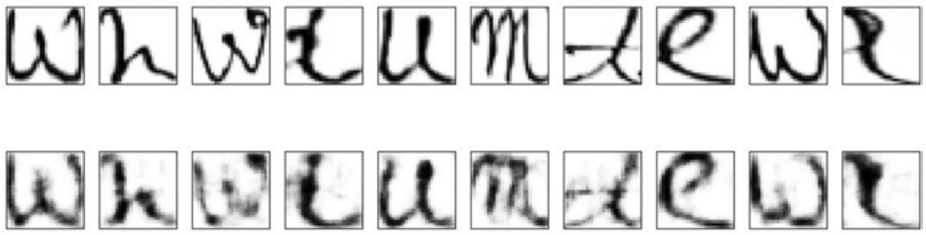


**Figure 5.8:** Reconstruction of segmented characters using VAE.

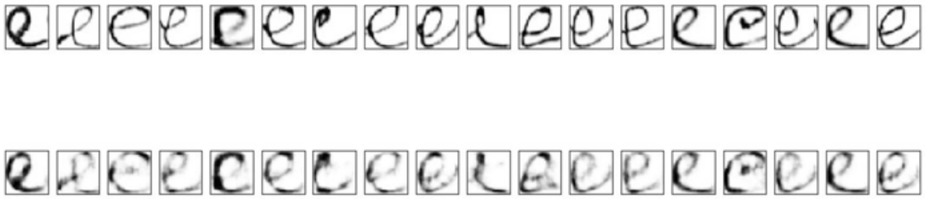


**Figure 5.9:** Representation of the cluster containing mostly character “e” written by patients.

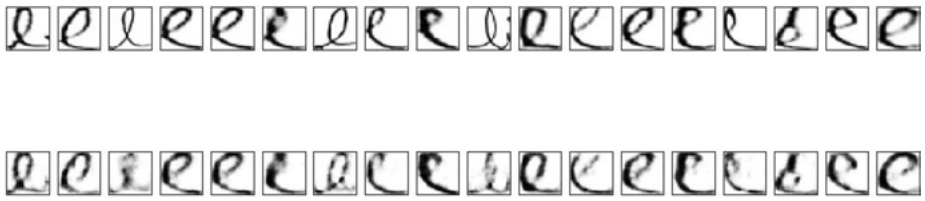


**Figure 5.10:** Representation of the cluster containing mostly character “” written by patients.

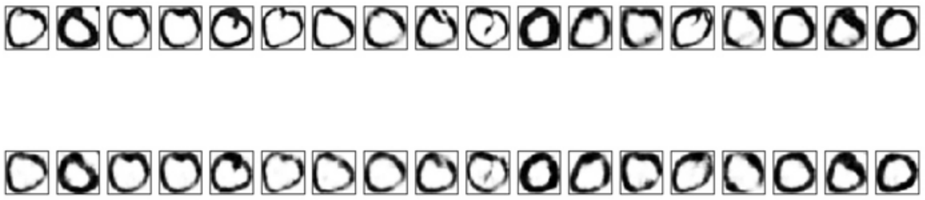


**Figure 5.11:** Representation of the cluster containing mostly character “o” written by patients.

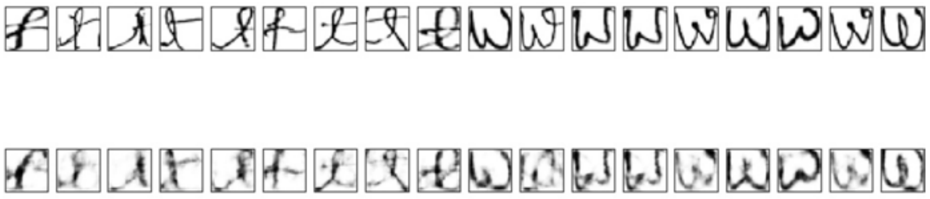


**Figure 5.12:** Representation of the cluster containing characters “t” and “w.”.

Now the inferences that can be drawn from the output obtained are as follows:

1. The three images of handwritten alphabets in Figure 5.13 (very few in number among all the images), substantially prove that children with dysgraphia have an odd gripping, they certainly face difficulty in holding a pen/pencil while writing and as a result the formation of the alphabets or characters is starkly peculiar and improper in comparison to that of a normal individual’s handwriting.

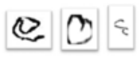


**Figure 5.13:** Snapshot of distorted characters.

This in turn makes the handwriting illegible to understand. The outcome of an odd holding grip is that it results in varying pressure imposed by the writer while writing. Thus, pressure exerted by the person having handwriting disability is far more than the fortunate ones and noticeably inconsistent throughout the handwritten sample. In fact, one can easily realize why speed of writing is very less for a dysgraphia patient. During data collection, it was noticed that a paragraph which a normal individual takes an average of 50 s to complete was actually being completed by dysgraphia patient in an average of 2.5 min.

2. Ambiguity in the handwriting:

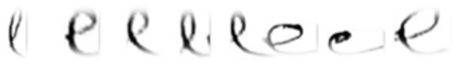


**Figure 5.14:** Snapshot representing ambiguous writing.

The clusters obtained consisted of similar looking alphabets together; for example, let us consider Figure 5.14 where “l,” “e,” and “c” were confused and jumbled up by the patients. To state it specifically, they fail to figure out and identify the specific character and as a result while copying they try to draw the same that they visualize instead of understanding what the character is and then penning it down.

3. Failure to distinguish between similar characters present adjacent to each other:

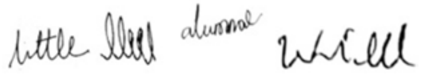


**Figure 5.15:** Snapshot representing failure to distinguish between similar characters.

The original words in Figure 5.15 were "little," "lull," "alumnae," "will;" since "le," "ul," "mn," and "il" are similar and placed adjacent, it was beyond recognition for them and as a result it lead to confusion and word formed by them was difficult to be deciphered.

4. Similar alphabets contained in the same word gets interchanged often:

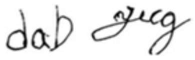


**Figure 5.16:** Snapshot representing interchanged letters.

In the example shown in Figure 5.16 the original words were "bad" and "guy." Since, "b" and "d," "y," and "g" are, though, not exactly same but look quite same, their position got interchanged when written by a dysgraphia patient. Though such error is not the same for all the patients it is quite common among majority.

5. Unclear and disconnected writing with errors in spellings:

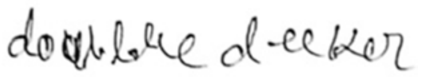


**Figure 5.17:** Snapshot representing disconnected writing.

This is one common facet for most; often the characters of words are disconnected and written far apart making it two different words (as in Figure 5.17) with no meaning and resulting in spelling mistakes.

Though handwriting can be a tangible tool to rely upon for detecting dysgraphia, certain characters can mislead too. Not all patients having illegible handwriting is a patient of dysgraphia or not all dysgraphia patients have extremely illegible handwriting. It may so happen that a normal person possesses very poor handwriting as compared to a dysgraphia patient.

It is also observed that alphabets like "i," "s," and few more were written perfectly and the clusters were same throughout as shown in Figure 5.18.

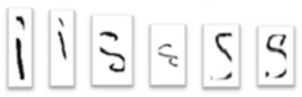


**Figure 5.18:** Snapshot of "i" and "s" present in the collected samples.

After studying so many handwriting samples, it is noted that patients with severe dysgraphia fail to write even a complete sentence and it's almost impossible to construe what is exactly written, as in Figure 5.19.

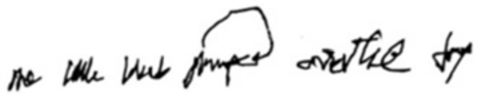


**Figure 5.19:** Handwriting of a patient with severe dysgraphia.

While collecting the handwriting samples, a child was found suffering from the disorder at a very serious and severe stage. Unfortunately, the ailment was all the more complicated as the subject was affected with another neurodegenerative disease as well. He was reluctant to write at the very beginning; though after a while of coaxing, he agreed to write but he failed to complete a single sentence and began to cry with panic.

The psychologies common to majority of these patients are that they are extremely reluctant and unwilling to write. While writing they become extra cautious about their handwriting and try to provide a suitable justification for their bad and unclear handwriting. Their tendency is to minutely observe the given sample and write the same which looked exactly similar. Not all but most of the patient observed was in a state of agitation while writing. Few complained of pain in their hands, as well. This endorses the theory which was read about the psychology of dysgraphia patient.

As already mentioned at the beginning of the chapter, the type of dysgraphia and symptoms present in a child varies uniquely and one may be suffering from more than one type. Analysis of the types by the machine seems to be beyond reach as of now.

In the field of agriculture, loss of production due to crop diseases is one of the crucial challenges. Hence, plant disease detection has established a thoughtfulness that production quality can be improved if the diseases are detected earlier. Different machine learning techniques are prescribed in related literature and these are very helpful in plant disease detection. Examples of these various machine learning methods are ANN, SVM, k-means clustering, K-NN and so on. Individually, these machine learning method based applications are articulated for segmentation, classification, and clustering. The experimental results in disease recognition show that the projected method is a primarily appreciated approach that can support disease detection in a tiny computational effort.

A postponement of this chapter will emphasize on the importance of incorporating decision support system along with other machine learning techniques, which may provide more accurate early detection of crop diseases. Future work also targets the area where machine learning is developed only to understand and handle various agriculture-related natural phenomenon. Also, the decision which is deduced by the machine should be converted into human-readable text or in the local language so that the farmer can understand and cooperate with the machine.

## 5.5 Conclusion

In this chapter, extraction of common features of dysgraphia patients by handwriting analysis using VAE was proposed. This method can be used to diagnose the disease at the early stage. This is a cost-effective and noninvasive method or diagnostic tool. Though no particular pattern was identified as an irrefutable sign of dysgraphia, the results from the study can be utilized to test the cognitive skills of the patient. The study could have been more effective had the samples been collected on an equipped electronic tablet so that pressure, time, and other factors could be precisely recorded. The study can be modified in future to yield better and more accurate result by adding more functionality like monitoring, tracking the gradual progression of the disease, and also identifying the stage in which the patient is at the time of acquiring the sample so that the improvement or deterioration of the condition can be detected at the earliest. This can only continue up to the stage where the patient retains normal functional ability. Dysgraphia patients in the terminal stages suffer from impaired functional ability and its certain that they would be unable to provide sufficient data to analyze using this model.

The study can also be applied to extract features for other neurodegenerative diseases with slight modifications in the implementation to determine whether handwriting analysis can be used as an early diagnostic method or not. This study will also benefit from the availability of a large training set and dynamic data collection. This will help in better image reconstruction. The dynamic collection will provide more precise parameters for more accurate clustering.

## References

[1] www.nature.com/npjdigitalmed

[2] Saha P., Das S.K., & Nandy S. Variational autoencoder coupled with deep generative neural network for the identification of handwritten digits. International Journal of Applied Engineering Research, ISSN 0973-4562, 2018, 13(10), 8014–8017.

[3] Kingma D.P. & Welling M. Auto-encoding variational Bayes, arXiv:1312.6114v10, 2014.

[4] Rosenblum S., Weiss P., & Parush S. Product and process evaluation of handwriting difficulties. Educational Psychology Review, 2003, 15(1), 41–81.

[5] Erhardt R.P. & Meade V. Improving handwriting without teaching handwriting: The consultative clinical reasoning process. Australian Occupational Therapy Journal, 2005, 52(3), 199–210.

[6] Rosenblum S. & Gafni-Lachter L. Handwriting proficiency screening questionnaire for children (HPSQ-C): Development, reliability, and validity. The American Journal of Occupational Therapy: official publication of the American Occupational Therapy Association, 2015, 69(3), 6903220030.

[7] O'Hare A. Hands up for handwriting. Developmental Medicine & Child Neurology, 2004, 46 (10), 651–651.

[8] Kushki A., Schwellnus H., Ilyas F., & Chau T. Changes in kinetics and kinematics of writing during a prolonged writing task in children with and without dysgraphia. Research in Developmental Disabilities, 2011, 32(3), 1058–1064.

[9] Fuentes C.T., Mostofsky S.H., & Bastian A.J. Perceptual reasoning predicts handwriting impairments in adolescents with autism. Neurology, 2010, 75(20), 1825–1829.

[10] Peverly S.T., Vekaria P.C., Reddington L.A., Sumowski J.F., Johnson K.R., & Ramsay C.M. The relationship of handwriting speed, working memory, language comprehension and outlines to lecture notetaking and test-taking among college students. Applied Cognitive Psychology, 2013, 27(1), 115–126.

[11] Martins M.R.I., Bastos J.A., Cecato A.T., de Lourdes Souza Araujo M., Magro R., & Alaminos V. Screening for motor dysgraphia in public schools. Jornal de Pediatria, 2013, 89(1), 70–74.

[12] Rosenblum S., Dvorkin A.Y., & Weiss P.L. Automatic segmentation as a tool for examining the handwriting process of children with dysgraphic and proficient handwriting. Human Movement Science, 2006, 25(45), 608–621.

[13] Rosenblum S., Goldstand S., & Parush S. Relationships among biomechanical ergonomic factors, handwriting product quality, handwriting efficiency, and computerized handwriting process measures in children with and without handwriting difficulties. The American Journal of Occupational Therapy, 2006, 60(1), 28–39.

[14] Rosenblum S., Parush S., & Weiss P. The in air phenomenon: Temporal and spatial correlates of handwriting process. Perceptual Motor Skills, 2003, 96, 933–954.

[15] Deng L. & Yu D. Deep learning: Methods and applications. Foundations and Trends® in Signal Processing, 2014, 7(3–4), 197–387.

[16] Kingma D.P. & Welling M. Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114v10, 2013.

[17] https://jaan.io/what-is-variational-autoencoder-vae-tuto

[18] Farahmand A., Sarrafzadeh A., & Shanbehzadeh J. Document image noise and removal methods. Proceedings of the International Multi Conference of Engineers and Computer Scientists, Hong Kong, 2013, I.

[19] Story G., O'Gorman L., Fox D., Schaper L., & Jagadish H. The rightpages image-based electronic library for alerting and browsing. IEEE Computer Society Press Los Alamitos, CA, USA, 1991, 25(9), 17–26.

[20] Nicolas S., Paquet T., & Heutte L. Text line segmentation in handwritten document using a production system. Proceedings of the 9th Int'l Workshop on Frontiers in Handwriting Recognition, 2004, IEEE, 3.

[21] Bin Abdl K.M. & Mohd Hashim S.Z., Handwriting identification: A direction review. IEEE International Conference on Signal and Image Processing Applications, 2009, 978-1-4244-5561-4/09.

[22] Otsu N. A threshold selection method from Gray level histogram. IEEE Transaction on system, Man, Cybernetics, 1979, SMC-9, 62–66.

[23] Niblack W. An Introduction to Digital Image Processing. New Jersey Prentice-Hall, Englewood Cliffs, 1986.

[24] Premchaiswadi N., Yimgnagm S., & Premchaiswadi W. A scheme for salt and pepper noise reduction and its application for OCR systems. WSEAS Transactions on Computers, 2010, 9, 351–360.
[25] Plamondon R. & Srihari S.N. On-line and off-line handwriting recognition. A Comprehensive Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000, 22, 1.
[26] Solihin Y. & Leedham C.G. Noise and Background Removal from Handwriting Images, IEEE, 1997.

Ankita Bose and B. K. Tripathy

# 6 Deep Learning for Audio Signal Classification

**Abstract:** Audio signal processing and its classification dates back to the past century. From speech recognition to speaker recognition and from speech to text conversion to music generation, a lot of advances has been made in this field using algorithms such as hidden Markov models, recurrent neural networks with long short-term memory layers (LSTM), deep convolutional neural networks (DCNNs), and the recent state-of-the-art model for music and speech generation using WaveNets. These algorithms are applied after the audio signals are processed and effective feature extraction techniques are applied on them. Nowadays, devices come up with personal assistants with which they can interact either through text inputs or voice inputs. Most applications have also come up with voice search features, while some can generate transcripts from videos and recognize the song title when played. The constant urge for attaining perfection has also led to hybrid models combining supervised and unsupervised learning techniques for better feature extraction. The ability to deal with spectral and temporal data makes DCNNs different from deep neural networks and also makes it the appropriate choice to deal with speech data because correlation between words and phonemes are a characteristic of such data. The potentials of convolution neural networks are huge and being extended in areas like environmental sound classification, music and instrumental sound processing and classification, and large vocabulary continuous speech recognition. Therefore, this chapter gives detailed explanation about what an audio signal is and how it is processed. It will also cover the various feature extraction techniques and the classification algorithms. Finally, the present-day applications and the potentials of deep learning in this field will be explored.

**Keywords:** audio signal, feature extraction, speech processing, deep learning, applications of audio signal processing

## 6.1 Introduction

Language is one of the most important inventions of mankind without which communication would not have been possible. Speech using any language is the most effective and fast mode of communication because no other method can convey the accurate emotions along with the information, parallelly. Computers have been competing with humans in all aspects, be it processing speed or accuracy. The revolution

**Ankita Bose, B. K. Tripathy,** VIT University, Vellore, Tamil Nadu, India.

https://doi.org/10.1515/9783110670905-006

brought about by artificial intelligence allows computers to think rationally and act by keeping a balance between rational and human behavior. To fulfil this aim, a computer needs to be equipped with the senses that a human is gifted with – vision, hearing, taste, smell, and touch. Vision has already been incorporated in computers as what is popularly known to be computer vision [1]. To allow taste, artificial tongues are being created [2] and certain kind of sensors can detect the presence of gas which can indicate the smell. For touch, scientists have already developed artificial skin [3], which leaves us with the question of what has been done to give computers the ability to hear and speak like humans. Thus, we arrive at the discussion of processing audio signals to enable the computers to understand audio data and respond like humans do. The context of audio is not restricted to just speech, it could be birds chirping, the sound of rain, the grunting of an engine, a baby crying, music from a flute, or any of the more limitless possibilities. The applications of identifying what kind of audio was processed holds good in multiple arenas like chatbots [4], surveillance, environmental sound classification [5](which could help detect the kind of scene from the last phone call of a victim), classification of musical instruments [6] or musical genres [7], speech to text converters (STT), [8] and the like. Learning to process audio data also implants the ability to generate music for people who cannot play instruments or generate speech for people who have difficulty in speaking (text to speech converters) [9]. The learning is accomplished in two major steps: signal processing for feature extraction and classification. The best features for classification happen to be the Mel frequency cepstral coefficients (MFCCs) [10], and the classification algorithms used has improved over time from hidden Markov models (HMMs) to deep convolutional neural networks (DCNNs). The earliest used model was HMM [11] but it had limitations [12], the most striking of which was the assumption which states that the probability of being in a given state at time $t$ only depends on the state at time $t$–1. This was taken care by using long short-term memory layers (LSTMs) in conjunction with recurrent neural networks (RNNs) [13]and also restricted Boltzman machines (RBMs) were used for the same purpose [14] but DCNNs [5] proved to give best results. While all these techniques served for classification, the best model for generation of speech or music proved to be WaveNets. With the help of these models, applications such as voice search, transcript generation, song search (Shazm, tells which song is being played by recognizing the audio), dictation, and personal assistants or audio books are made. With more and more data being available in corpuses in written or audio form, language recognition and real time translation over video chats are being made possible. Combining labelled video and audio classification in the training data can make annotations for sound in videos possible (helpful for deaf people). Detecting emotions from speech and context recognition are complex problems which can be solved by the combined use of both audio data classification and natural language processing (NLP).

## 6.2 Audio signal processing and its importance

The fifth generation of computers are expected to think, analyze, and act on their own. We prefer to give commands and get fast responses. Who could imagine that saying "OK Google, play me some JAZZ" could actually play some jazz music for you, without you exclusively having to search for the genre. You could also turn on the fan without compromising the comfort of your sofa by telling Alexa (personal assistant launched by Amazon) to do it for you. These fascinating advancements could be made possible by audio signal processing. Not only can it be used for fancy things, environmental sound classification [15] has made it possible to know the kind of environment in which a signal has been recorded. This could help in locating a victim from their last phone call or in annotating videos. There have been experiments on classifying sounds from different birds, sounds of babies crying [16] to differentiate normal and pathological cries, classification of musical instruments, identifying the speaker, and much more. The study of various characteristics of an audio signal, along with feature extraction and proper training using machine learning algorithms makes this possible. The next section gives a detailed insight into the nitty-gritties of an audio signal. Figure 6.1 shows an audio waveform.

**Figure 6.1:** An audio waveform.

### 6.2.1 Audio signals

The basic definition of sound says it is a vibration that propagates essentially through any medium longitudinally and as both longitudinal and transverse waves in solids. In case of *longitudinal waves,* the particles of the medium vibrate parallel to the direction of wave propagation which might either be in the same or opposite direction. This motion causes regions of compression and rarefaction which indeed causes regions of varying pressure and hence these waves are also called *Pressure waves or Compression waves* as shown in Figure 6.2.

On the other hand, particles in the medium travel in a direction perpendicular to the wave propagation in case of *transverse waves* as shown in Figure 6.3. Waves are defined in terms of various characteristics – for longitudinal waves, the regions of highest particle concentration are known as *compressions* whereas the regions of lowest particle concentration are known as *rarefactions*; for transverse waves, *crests* are regions where maximum upward movement of particles occur and *troughs* are

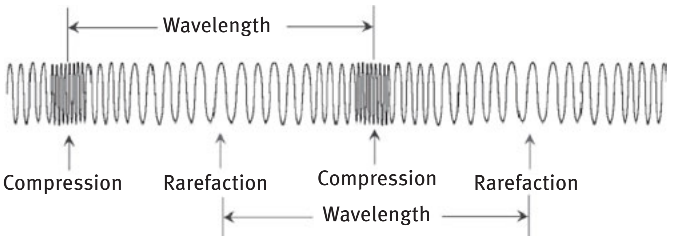


**Figure 6.2:** Depiction of a longitudinal wave.

regions where maximum downward movement occurs. Movements are relative to the rest position of the particle. For any wave, the distance between two consecutive crests or troughs, or between two compressions or rarefactions, is known as the wavelength. *Frequency* of a wave is the number of waves that pass through a fixed point in the medium, per second. Shorter the wavelength, more is the frequency and hence more is the *pitch* of the audio. A high-pitched audio appears shriller than a low pitched one. The *amplitude* of a wave is the maximum displacement of the particle from its mean or rest position and is a measure of *loudness* and *intensity*. The more the amplitude, louder the sound appears. What tells apart a *note* from a *noise* is the orderly repetitive nature of its wave. Waves that repeat itself at intervals are more soothing as compared to waves that have irregular waveforms that do not repeat and constitute noise which is very unpleasant. The *quality* or *timbre* of a sound wave is determined by the waveform. Keeping the frequency constant, if the waveform is changed, different sounds of the same pitch will be perceived; this aspect is explored more in electronic music where varied sounds are produced by modulating the waveforms. The earlier description is a small outlook on the anatomy of a wave. For details on the mechanism by which a human being is able to hear please refer to [17]. The subjective nature of volume, noise, and how sound perceived varies among different individuals, and is immaterial to the context since we are to discuss various audio signal processing techniques for feature extraction and classification. The next section gives details on the various features of an audio wave and the methods by which they can be extracted.

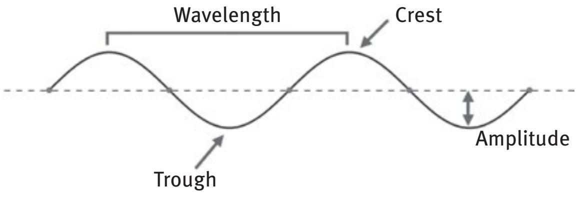


**Figure 6.3:** Depiction of a transverse wave.

### 6.2.2 Feature extraction methods

Machines learn through training. They are first made to get acquainted with the kind of input they need to process and are told what exactly it is. This is the simplest way in which supervised learning can be described. Therefore, when it comes to audio classification, the classification algorithm is fed with a set of relevant features corresponding to each audio piece, and a label, indicating either the words in the original audio or a sentence or a musical note. When a new sample is to be classified, the program extracts features from the new sample and outputs the nearest class to which these features belong. Audio features can be broadly divided into two heads – *physical* and *perceptual* features [18]. Each of these broad categories can be further subdivided into time, frequency, wavelet, image, and cepstral domains. In many audio classification systems, there is a segment where relevance feedback is taken from human beings, therefore, it becomes extremely important to take into consideration the perceptual features, because the way humans perceive sound is quite different from the physics of it. Therefore, either perceptual features are extracted or signal features, that bear a correlation to the perceptual features, are extracted [19].

A thorough research in psychoacoustics and the rudimentary stages of auditory signal processing in humans forms the basis of perceptual features. The cochlea in the inner ear serves as intersecting bandpass filters. To model it in real world, proposed methods include gammatone filters [20–22], and simple implementation using discrete Fourier transforms, among others and the features thus extracted are known as *cochleagrams*. Another such feature is the *correlogram* which is responsible for capturing the essence of the fundamental frequency and higher harmonics from the signal [23] that helps in segmentation of audio. *Modular Spectrums* on the other hand are able to detect variances from extended durations which help in preserving phonemic or syllabic structure in case of speech signals; this is done by applying some kind of transform on every frequency component along the time axis [24]. The aforementioned features model the outer ear. To model the complexities of the inner ear *multiscale representations* are used [25, 26]. Therefore, perceptual features can be thought of as simple minor features that are able to convey notable impressions of sound synonymity. Loudness, sharpness, and spectral features, such as spectral centroid or spectral flux and roughness, are few among many such features; MFCC being one of the most important and widely used ones, which will be elaborated later in the chapter.

Physical features are more mathematical in nature and extraction techniques are mostly applied on the crude signal without transforming it to other domains or applying filters, as opposed to perceptual features. Such features include the magnitude of energy, alterations in the amplitude of a waveform within every cycle [27], the duration of such alterations, or the root-mean square of the magnitude of a waveform. These features could also be *rhythm-based* and computed using different methods including finding the peaks of the output envelopes in frequency bands or by finding pulse clarity or band periodicity, to state a few. Another set of physical features are *frequency-based* which constitute spectral flux, line spectral frequencies, spectral peaks, and the spectral centroid which is defined as the center of gravity of spectral energy leading to the mean of the frequency values. Frequency features also include pitch profiles, harmonicity, jitter, and a lot more spectral features which will be elaborated later in the chapter. *Wavelet transforms* have an upper-hand over Fourier transforms (FT) when representing discontinuous functions, because they are able to deconstruct and reconstruct finite, non-periodic and non-stationary signals precisely. Therefore, wavelet-based features such as Hurst parameter features and matching pursuit-based Gabor features serve as an appropriate feature-set for vocal and musical audio signals. Image processing techniques are used to extract features from time–frequency illustration of audio signals in the form of image. These features are known as image spectrograms. Among other physical features are *cepstral features* and features from the eigen space, phase space, and acoustic environment features.

The taxonomy of audio signal features branches out to great details with various kinds of features under each head, interested readers can refer to [18] for further details. In this chapter, only the most common and widely used features for audio classification and learning have been mentioned and detailed.

### 6.2.3 Mel frequency cepstral coefficients (MFCCs)

MFCCs are the most widely used perceptual features because of their success in embodying the speech amplitude spectrum concisely. These features are obtained after applying a series of functions on the audio signal as described [10, 28].

It starts with applying a filter that amplifies the higher frequencies in the signal, which elevates the energy at these parts. After amplification, the entire signal is fragmented into small frames ranging typically between 20ms and 40ms. This is done because an audio signal changes constantly, but not much statistical variations are observed over short time frames, so the chosen range is appropriate as ranges smaller than that would not give sufficient samples for decent spectral estimation. The next process in the line is applying the Hamming window function which eliminates the edge effects and assimilates all the frequency lines that are placed close together. This is done to imitate the cochlea of the human ear that is unable to discriminate between frequencies that are very near to each other, thus maintaining the perceptual nature of this technique. The output of the Hamming window is expressed in eq. (6.1).

$$y(n) = x(n) \times w(n) \tag{6.1}$$

$$w(n) = 0.54 - 0.46 \times \left(\cos\frac{2\pi n}{N-1}\right),\ 0 \le n \le N-1 \tag{6.2}$$

In the eq. (6.2), *N* represents the number of samples in each frame and in eq. (6.1), *y(n)* is the output signal. The input signal is represented by *x(n)* and *w(n)* represents the Hamming window operation. A time signal is composed of its constituent frequencies and a FT on the time signals gives us the corresponding complex-valued function in the frequency domain. This transform is represented by eq. (6.3).

$$F(\xi) = \int_{-\infty}^{\infty} f(x)e^{-2\pi i x\xi}dx \tag{6.3}$$

In the above equation ξ is any real number and Ƒ(ξ) represents the FT of the function f(x). Therefore, after windowing, FT is applied on the resultant signal so now the subject worked upon is a frequency representation. The resultant signal after applying FT is represented using eq. (6.4). The periodogram-based power spectral estimate for the resultant frame, represented by $P_i(k)$, is given by eq. (6.5).

$$S_i(k) = \sum_{n=1}^{N} s_i(n)y(n)e^{-j2\pi kn/N} \quad 1 \le \mathrm{k} \le \mathrm{K} \tag{6.4}$$

$$P_i(k) = \frac{1}{N}|S_i(k)|^{2} \tag{6.5}$$

In the above equation *i* is the total number of frames, *n* represents the samples in each frame, *N* is the total number of samples in a frame and *K* represents the length of the FT. *y(n)* is the output signal obtained after windowing. Let us assume that first **C** coefficients are used for further processing (use shown later). The inability of the cochlea to distinguish between near adjacent frequency lines becomes more pronounced towards the higher frequencies. The Mel scale helps to incorporate this behavior into the extracted features so that it is analogous to how humans process audio. The Mel transform is given by eq. (6.6).

$$M(f) = 1125 \times \ln\left(1 + \frac{f}{700}\right) \tag{6.6}$$

In the above equation, *f* is the frequency expressed in hertz. To represent the power spectrum on the Mel scale, filterbanks are applied to it. Figure 6.4 shows a demonstration of the filterbanks and the windowed segments.

The number of these triangular filterbanks usually range from 20–40 and the length of each filterbank is equal to **C**. Each filterbank is multiplied with the power spectrum and the weighted sum of the coefficients are taken. So, if the number of filterbanks taken is equal to say *b* such that $20 \le b \le 40$, we get *b* numbers that denote the energy constituent in each filter. Next step in line is to perform the logarithm of the numbers obtained. This is done to imitate the fact that human beings do not

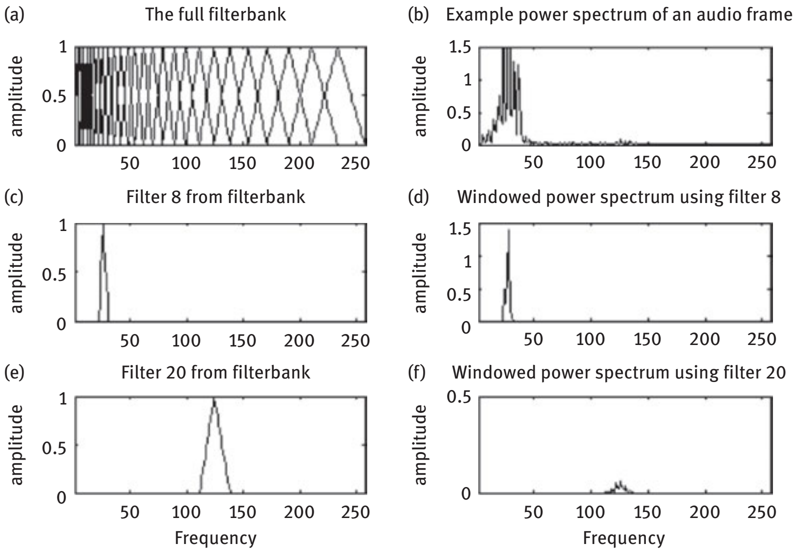


**Figure 6.4:** Plot of Mel filterbanks.

perceive loudness on a linear scale. Variations in sound is not very evident when the sound is too loud. At this stage, the numbers obtained are highly correlated. Consequently, dimensionality reduction can be considered to make further computations faster and easier. Hence, discrete cosine transform (DCT) is applied to the log filterbanks. This retains nearly half the coefficients, eliminating the higher valued ones which result in improved performance. These coefficients are known as **MFCCs**. They are commonly known as *acoustic feature vectors*. The most common representation of a DCT is given in eq. (6.7).

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)\times k\right],\ \mathrm{k}=0,\ \ldots,\mathrm{N}-1 \tag{6.7}$$

The eq. (6.7) represents a DCT, where numbers in one domain – $x_0, x_1, \ldots, x_{N\text{-}1}$ are transformed into another domain as $X_0, X_1, \ldots, X_{N\text{-}1}$.In case of speech processing, improvements in the results can also be made by adding features related to the dynamics of speech. The changes in the trajectory of the MFCCs are given by the Delta function as shown in eq. (6.8). This would give us an equal number of delta coefficients as the MFCCs and appending these to the original features gives an extended feature set that is shown to improve performances in automatic speech recognition.

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2} \tag{6.8}$$

In the above equation, $d_t$ represents the delta coefficient of the $t^{th}$ frame, $c_{t+n}$ and $c_{t-n}$ are static coefficients and the value of $N$ is usually taken to be 2.

## 6.3 Timeline of algorithms used for audio signal processing and classification

Research in audio signal processing and classification has been going on since a long time starting around the beginning of the twentieth century. Speech recognition (SR) was the main area of interest among other audio processing techniques. Environmental sound classification and music processing has gained interest more recently. It took years of research to reach to the neural network approach for audio classification that outperformed all the other then known techniques. Prior to that HMMs and other probabilistic models would be used both for classification and generation. These models are elaborated in this section along with their limitations.

### 6.3.1 Hidden Markov models

HMMs are stochastic models that work on sequences. The main functions of HMMs include part-of-speech (POS) tagging, SR, and named entity tagging, among others. HMMs are derived from the Markov Chain model (MCM) which consists of a set of states $Q = \{q_0, q_1, q_2, \ldots, q_N\}$, a transition matrix A as shown in eq. (6.9) and a start state ($q_O$) and final state($q_N$).

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{bmatrix} \tag{6.9}$$

In the above equation $a_{ij,}$ that is, the element in the $i^{th}$ row and $j^{th}$ column represents the probability of transition from state i to state j. The summation of probabilities from a state to all the other states is 1, as a result the summation of each row in the matrix is 1. The main property of the MCM is that the probability of being in a particular state is dependent solely on the previous state; as demonstrated by eq. (6.10).

$$P[X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_0 = x_0] = P[X_{t+1} = x_{t+1} | X_t = x_t] \tag{6.10}$$

MCMs are useful when all the states in a model are completely visible to us and we are required to calculate the probability of a sequence of events. Figure 6.5 shows a simple Markov model.

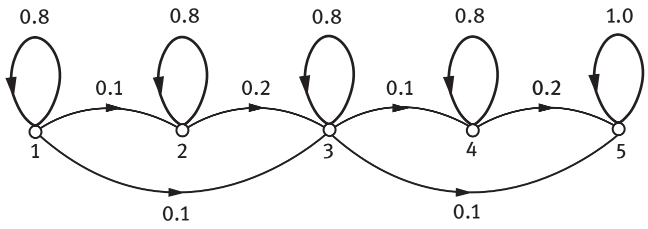


**Figure 6.5:** A five state Markov model with state transition probabilities.

But there are quite many events that are not completely visible to us, for example, if we are to predict the weather outside from a room with closed windows and doors or we have to predict the color of a ball drawn from an urn containing balls of many colors. To be able to predict these events HMMs are introduced. There are $N$ hidden states – $\{S_1, S_2, \ldots, S_N\}$, in a HMM and in most cases, each state can be reached from the other state. The state at time $t$ is represented by $q_t$. Each state contains $M$ observations – $\{v_1, v_2, \ldots, v_M\}$. The state transition matrix denoted by eq. (6.9) is common for both MCMs and HMMs. In addition to that, HMMs have an observation symbol probability distribution, $B = \{b_y(\text{x})\}$ that is indicative of the possibility of observation $v_x$ in state $q_y$, where $1 \leq \text{x} \leq M$ and $1 \leq y \leq N$ as shown in eq. (6.11).

$$b_y(x) = P\left[v_x \text{ at } t | q_t = S_y\right] \tag{6.11}$$

Lastly, it is composed of the initial state probabilities that indicates the chances of being in a particular state at time $t = 1$, that is, the starting point. This is denoted by $\pi$ and explained through eq. (6.12).

$$\pi_i = \mathrm{P}[q_1 = S_i]\,; \text{where } 1 \leq i \leq N \tag{6.12}$$

This entire model including N, M, A, B and $\pi$ is denoted by $\lambda$ and is shown in Figure 6.6. It is used to explain or account for any set of observation sequences, for example, given λ, HMMs can be used to calculate the probability of three consecutive rainy days starting from tomorrow. HMMs address three main problems:

**Problem 1:**
To compute the probability of an observation sequence $O = \{O_1, O_2, \ldots, \mathrm{O}_T\}$ over duration $T$, given $\lambda$. The most efficient solution to this problem that requires the least computation has been given by the *forward–backward algorithm* [29, 30].

**Problem 2:**
Choosing a state sequence $Q = \{q_1, q_2, \ldots, q_T\}$, that best explains an observation sequence $O$, given a model $\lambda$. The Viterbi algorithm best solves this problem [31, 32].

**Problem 3:**
Adjusting the model parameters to maximize the probability of a given observation, that is $P(O|\lambda)$. This problem is solved by the Baum–Welch method [33] or gradient techniques [34].

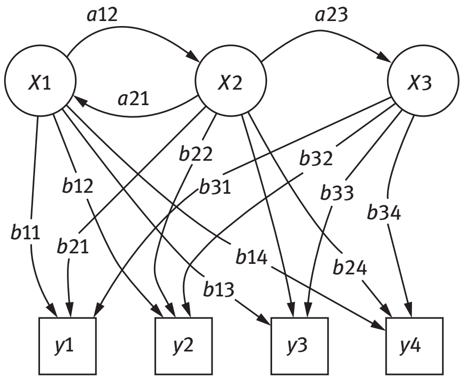


**Figure 6.6:** An hidden Markov model with hidden states depicted in *X* and visible states depicted in *Y*. The hidden state probabilities are given in *a* and probability of a visible state from a hidden state is given in *b*.

HMMs can be constrained or unconstrained. Ergodic HMMs are the ones where transitions are possible from every state to the other state, hence they are unconstrained. Other variations of HMMs are left–right(LR) models where transitions are constrained from state *j* to state *i*, that is, $a_{ij} = 0$ if j<i. Sometimes, the number of jumps between the states is also constrained, that is, $a_{ij} = 0$ if j<i and $j \geq i+\mathcal{H}$ where $\mathcal{H}$ is the number of hops. Constrained HMMs also include parallel LR models which is formed by the cross-coupled linkage between 2 parallel LR HMMs. They are more relaxed than the typical LR models as they allow parallel paths.

In this chapter, we will discuss more on the use of HMMs in SR. Every classification algorithm for speech processing works on features extracted. HMMs are shown to work well with features extracted using linear predictive coding (LPC). A series of steps are followed [35] to calculate the LPC coefficients that model the vocal track and captures utterances. Consequently, they are also known as speech attributes. These coefficients can also be used later to regenerate the original audio, hence it serves very well as a speech compression algorithm. LPCs give coefficients that form vectors with continuous values; but to use these as features for HMM, discrete values are required for which a vector quantization (VQ) step is performed [36, 37]. First, the functioning of HMMs in identifying individual words will be explained followed by continuous SR [37].

In terms of HMMs, any word uttered can be thought of as a set of observations, O, which corresponds to the outputs from the VQ. Since SR is independent of the speaker, audio samples are taken from around $K$ (usually K=100) speakers. Then the probability of the observation sequence $O^{(k)}$ given the model is calculated and focus lies on maximizing the product as shown in eq. (6.13). This is done using the Baum–Welch algorithm [33].

$$\mathrm{P} = \prod_{k=1}^{K} P(O^{(k)}|\lambda) \tag{6.13}$$

Finding out the probability of the observed sequence then resembles problem 1. This process confides in two segments – training and identification. During training, a model is built for every word present in the vocabulary and while *identifying* a test observation, each model's probability to generate the given observation sequence is assessed and the test case is classified as the word whose corresponding model gives the highest probability. This probability *P(O|λ)* is calculated as shown in eqs. (6.14)–(6.17) (all symbols and notations are in accordance with the standard HMM model as already described above).

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, q_t = S_i|\lambda) \tag{6.14}$$

$$\therefore \alpha_1(i) = \pi_i b_i(O_1); 1 \le \mathrm{i} \le \mathrm{N} \tag{6.15}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1});\ 1 \le \mathrm{j} \le \mathrm{N},\ 1 \le \mathrm{t} \le \mathrm{T}-1 \tag{6.16}$$

$$\mathrm{P(O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{6.17}$$

In (14) an expression is given, that finds out the probability of an observation sequence that ends in a particular state $S_i$ at time $t$ = $T$, that is, the total duration of observation. This probability is calculated for every state in the model. *Training* involves adjusting the model parameters to maximize the probability of every model for each word that they represent for which the solutions to problems 2 and 3 are used. Training is terminated after no significant increase in probability is obtained or after a certain number of iterations. To better each model, parameters like the number of states, the initial state transition probability matrix, and the probability of an observation in a state has to be tuned in order to reach perfect accuracy. In every iteration of the training phase, these values are modified.

Along with individual word recognition, HMMs also contribute to continuous SR. Any sentence, $\mathfrak{S}$, is a continuous speech consisting of words. The features in this case is a sequence of phones, $\wp$, output by an acoustic processor (AP) (Jelinek 1976). To decide which sentence most probably caused $\wp$ is on the linguistic decoder and this is done by calculating the probability P(ς |$\wp$) as shown in eq. (6.18).

$$\text{By Bayes Theorem:} P(\varsigma|\wp) = \frac{P(\wp|\varsigma)P(\varsigma)}{P(\wp)} \tag{6.18}$$

In the above equation P(ς) is an a-priori probability that is supposed to be calculated by the language model (LM). Here is where HMMs come to play because it serves as the LM just mentioned. The Raleigh language is an implementation of the procedure which serves for relatively simple language models. It is executed as a graph with every branch having a state transition probability while word lists contained in boxes on the edges serve as choices for the output of transitions. Features for each of the words is then extracted using an AP and a corresponding HMM is constructed which would either give 1 output or nothing at all. Next, the Viterbi algorithm is used to find the most probable state path beginning from the start. The word corresponding to this path is taken as the output.

HMMs had ruled for quite some time around when it was introduced due to its commendable performance in certain types of SR. But it turns out that some of the basic assumptions made to build up such models are the causes of its limitations. The main Markov assumption that the probability of being in a particular state is solely dependent only on its previous state is inaccurate because, in case of speech signals, the reliance usually extends through more than one state. Also, the assumptions that succeeding observations are independent and their distributions can be well represented by autoregressive or Gaussian densities, often leads to incorrect outcomes.

### 6.3.2 Long short-term memory-recurrent neural networks

In language, the smallest unit is a phoneme. Phonemes combine to form words which in turn combines to form a sentence and many sentences together form a context. Therefore, one cannot simply determine the entire word by listening to the phoneme at a particular time frame, as the utterances before and after together makes a meaningful word. Similarly, to determine the context of a given sentence or a piece of text, which is an important application of NLP, it is necessary to take into account the entire data before and after that sentence. RNNs were used widely for the task of SR, but they were not good at modelling the dependencies between phonemes or words. This is due to the fact that whenever the frame of consideration is increased, the vanishing and exploding gradient descent problem arises. Long short-term memory-recurrent neural networks (LSTM-RNNs) served as the solution which gave impressive results in SR by capturing dependencies over extended time frames. The main divisions of a LSTM unit are the input gate(i), forget gate(f), and the output gate(o). Since the network is recurrent in nature, the connections from the output gates are directed back to the input layer as shown in Figure 6.7. In essence, the forget gate controls the flow of data to avoid redundancy or repetition, given the previous inputs, that is, it is required to prevent similar words from reoccurring so that meaning is preserved. Memory(c) is

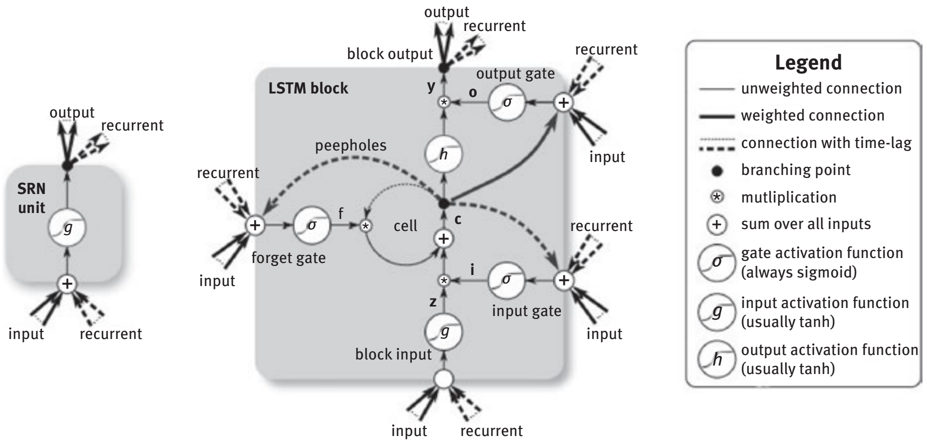


**Figure 6.7:** Detailed representation of a simple recurrent network unit (left) and long short-term memory unit (right).

also incorporated in LSTMs as shown in the Figure 6.7, so that it is able to remember previous words encountered which helps to stick to the context. A single layer comprising LSTM units is not very good at the task of SR, so to make it better multiple hidden layers of LSTMs are stacked together which is known as deep-LSTMs [38]. In case of SR, commendable accuracy is obtained by using a combination of an acoustic model(AM) and a LM both of which are built using LSTM-RNNs. The set of inputs to any LSTM-RNN model, represented by $x$={$x_1$,$x_2$, . . ., $x_T$}, can either be a set of discrete words in case of a LM or a vector of MFCCs in case of an AM. The hidden sequence calculated through $t$={1,2, . . ., $T$} is denoted as $h$ = {$h_1$,$h_2$, . . ., $h_T$} and the outputs as $y$ = {$y_1$,$y_2$, . . ., $y_T$}.

The calculations of the hidden and output units are shown in eqs. (6.19) and (6.20) where Ƕ(.) is a mere simplified representation of the more complex operations used to calculate the value of each hidden LSTM unit and $W$ represents the entire set of weights; at any point $W_{ab}$ represents the weights between the $a$ and $b$ units which can one among $i, f, o, c, x$, and $h$, and $b$ is a bias term.

$$h_t = Ƕ(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{6.19}$$

$$y_t = (W_{hy}h_t + b_y) \tag{6.20}$$

The breakdown of Ƕ into the functions of the previous mentioned gates are shown in eqs. (6.21)–(6.25) which also explains the flow of data within the LSTM unit.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{6.21}$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \tag{6.22}$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{6.23}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{6.24}$$

$$h_t = o_t \tanh(c_t) \tag{6.25}$$

Various architectures involving LSTMs have been proposed [13, 38] and a detailed comparative study on the proposed architectures have been carried out by [39] which shows that bidirectional LSTM-RNNs(BLSTM-RNNs) give the best results when it comes to phoneme recognition, therefore we will elaborate more on the training and framework of BLSTMs which take into consideration the future context as well which is necessary as explained before. Equations (6.26) and (6.27) show the calculation of $h^{\leftarrow}$ and $h^{\rightarrow}$, which represents the backward and forward sequences respectively.

$$h_t^{\rightarrow} = \mathcal{H}\left(W_{xh^{\rightarrow}}x_t + W_{h^{\rightarrow}h^{\rightarrow}}h_{t-1}^{\rightarrow} + b_{h^{\rightarrow}}\right) \tag{6.26}$$

$$h_t^{\leftarrow} = \mathcal{H}\left(W_{xh^{\leftarrow}}x_t + W_{h^{\leftarrow}h^{\leftarrow}}h_{t-1}^{\leftarrow} + b_{h^{\leftarrow}}\right) \tag{6.27}$$

$$y_t = W_{h^{\rightarrow}y}h_t^{\rightarrow} + W_{h^{\leftarrow}y}h_t^{\leftarrow} + b_{\mathrm{y}} \tag{6.28}$$

In deep LSTM-RNNs, the calculation for the hidden units are kept the same for all the $N$ layers as shown in (29), where $h^0$ = x, and the output depends on the inputs from the final hidden layer as shown in (30). In deep BLSTMs, every hidden layer comprises a forward and a backward sequence: $h^{\mathrm{n}\leftarrow}$ and $h^{\mathrm{n}\rightarrow}$and every layer receives inputs from both the sequences in the previous layer.

$$h_t^n = \mathcal{H}\left({}_{h^{n-1}h^n}h_t^{n-1} + W_{h^n h^n}h_{t-1}^n + b_h^n\right) \tag{6.29}$$

$$y_t = W_{h^N y}h_t^N + b_{\mathrm{y}} \tag{6.30}$$

The network is trained to directly predict the phonemes from the acoustic input $x$. The conditional probability, Pr($y$/$x$), of all the possible phonemes given the input sequence is calculated and the log probability of the target output sequence $z$, log Pr($z$/$x$), is then differentiated with respect to the network weights using backpropagation through time and optimization is done using gradient descent. In [38] an improvement over the normal probability calculation is suggested by conditioning the output phoneme on both the acoustic input and the phoneme at the previous time step.

## 6.3.3 Restricted Boltzman machines with deep belief networks

RBMs are probabilistic neural network models that have proved to be good at object recognition, phone recognition, and also learn better representations of images and sound data. Since we are focused on audio signal processing, we will detail more on the applications of RBMs in speech processing by phone recognition [40]. An RBM comprises of a single layer of binary visible units, υ, and another layer of binary hidden units, ɧ. It is called restricted because of its bipartite nature, that is, there are no intralayer connections. Sampling and fragmentation are a pre-requisite for any classification technique. Similarly, *υ* embodies a particular fragment containing *ȵ* samples. This entire model can be represented as Θ = {ƍ,ẟ,ω}. The energy of the joint architecture is given in eq. (6.31), where $\upsilon_i$ and $\text{ɧ}_j$ are the units in the visible and hidden layer respectively, $\text{ƍ}_i$ is the bias for the ith unit in the visible layer whereas $\text{ẟ}_j$ is the bias for the jth unit in the hidden layer and $\omega_{ij}$ is the weight between the ith unit in the visible layer and the jth unit in the hidden layer.

$$E(\upsilon, \text{ɧ}) = -\sum_{i\,\epsilon\,visible} \text{ƍ}_i v_i - \sum_{j\,\epsilon\,hidden} \text{ẟ}_j \text{ɧ}_j - \sum_{i,j} \upsilon_i \text{ɧ}_j \omega_{ij} \tag{6.31}$$

The probabilities assigned to visible units and to all possible pairs of visible and hidden units, using the energy function in (31), is given in eqs. (6.32) and (6.33) respectively.

$$P(\upsilon|\theta) = \frac{\sum_{\text{ɧ}} e^{-E(\upsilon,\text{ɧ})}}{\sum_{v,\text{ɧ}} e^{-E(\upsilon,\text{ɧ})}} \tag{6.32}$$

$$P(\upsilon, \text{ɧ}|\theta) = \frac{e^{-E(\upsilon,\text{ɧ})}}{\sum_{v,\text{ɧ}} e^{-E(\upsilon,\text{ɧ})}} \tag{6.33}$$

Since the units contain binary values, the probability with which any of the hidden and visible unit will be 1 is given by eq. (6.34) and (6.35), where $\sigma(\varkappa) = (1 + e^{-})^{-1}$.

$$P\left(\text{ɧ}_j = 1|\upsilon;\theta\right) = \sigma\left(\sum\nolimits_{i\,\epsilon\,visible} \upsilon_i\, \omega_{ij} + \text{ẟ}_j\right) \tag{6.34}$$

$$P\left(\upsilon_i = 1|\text{ɧ};\theta\right) = \sigma\left(\sum\nolimits_{j\,\epsilon\,hidden} \text{ɧ}_i\, \omega_{ij} + \text{ƍ}_j\right) \tag{6.35}$$

In case, the RBM is required to replicate the joint distribution of the class labels along with the data, the vector of binary representation of the class labels, *ȴ*, are concatenated with the vector of visible units and the energy function taking into consideration the labels is given in eq. (6.36).

$$E(\upsilon, \text{ɧ}, \text{ȴ}) = -\sum_{i\,\epsilon\,visible} \text{ƍ}_i \upsilon_i - \sum_{j\,\epsilon\,hidden} \text{ẟ}_j \text{ɧ}_j - \sum_{y\,\epsilon\,labels} \text{ɕ}_y \text{ȴ}_y - \sum_{i,j} v_i \text{ɧ}_j \omega_{ij} - \sum_{j,y\,\epsilon\,labels} \text{ȴ}_y \text{ɧ}_j \omega_{ij} \tag{6.36}$$

The probability with which the label corresponding to a particular unit is turned on is given by eq. (6.25) and P(*ȴ*/*υ*) is given by eq. (6.38). In eq. (6.36) and eq. (6.37), ɕ is the bias associated with each label.

$$P(\text{ɭ}_y = 1|\text{ɧ};\theta) = \text{softmax}\left(\sum_{j\,\epsilon\, hidden} \text{ɧ}_j\omega_{yj} + \text{ȼ}_y\right) \tag{6.37}$$

$$\text{P}(\text{ɭ}|\upsilon) = \frac{\sum_{\text{ɧ}} e^{-E(\upsilon,\text{ɧ},\text{ɭ})}}{\sum_{l,\text{ɧ}} e^{-E(\upsilon,\text{ɧ},\text{ɭ})}} \tag{6.38}$$

Next, the RBMs are trained using either a generative or a discriminative approach as shown in Figure 6.8 where *V-H* and *v-h* correspond to *υ-ɧ* of the discriminative and generative model respectively and *W* corresponds to *ω*. The *generative* approach is more of a short-cut method that speeds up the training by a margin. It is done using the contrastive divergence (CD) approximation only once on the gradient, as shown in eq. (6.39), where $< . >_{data}$ represents the expectation that denotes the number of times $\upsilon_i$ and $\text{ɧ}_j$ are both equal to 1 in the training set and $< . >_1$ represents the expectation after applying $CD_1$.

$$\omega_{ij} = <\upsilon_i\text{ɧ}_j>_{data} - <\upsilon_i\text{ɧ}_j>_1 \tag{6.39}$$

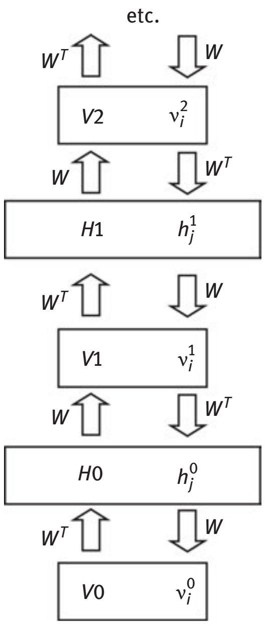


**Figure 6.8:** An Infinite logistic belief net where the generative part of the model is represented by downward arrows. The upward arrows represent parameters belonging to the discriminative part of the model.

The discriminative approach is used to train the RBM weights, so they act as initial values for the weights of a deep belief network. For models depicted by (36), the weight update takes place as shown in eq. (6.40); but to generalize the model well and prevent over fitting, the gradient of a function of ɭ and υ, f(ɭ,υ) is used to update the weights. ƌ in (41), is a parameter that controls the influence of the term it is associated with.

$$\Delta\omega_{\mathrm{ij}} = \sum_{j\in hidden} \sigma\left( ẟ_j + \omega_{jy} + \sum_{i\in visible} \upsilon_i\omega_{ij}\right)\upsilon_i - \sum_{\substack{y\in labels,\\ j\in hidden}} \sigma\left( ẟ_j + \omega_{jy} + \sum_{i\in visible} \upsilon_i\omega_{ij}\right)\mathrm{P}(ƀ|\upsilon)\upsilon_i \tag{6.40}$$

$$\mathrm{f}(\upsilon, ƀ) = ƌ\mathrm{P}(ƀ|\upsilon) + \mathrm{P}(\upsilon|ƀ) \tag{6.41}$$

Another variation of RBMs are the mean-covariance RBMs (mcRBMs), that gives one of the lowest error rates on the standard TIMIT dataset which is most widely used for the purpose of phone recognition. mcRBMs are composed of two categories of hidden units – precision and mean units. The precision units are required to impose smoothness constraints, consequently, they are active for samples that deviate beyond constrained limits. Therefore, the active precision units define the covariance matrix, particular to the sample. More about the structure of mcRBMs can be found in [41].

One way to look at deep belief networks **(*DBNs*)** is that they are layers of RBMs stacked up together as shown in Figure 6.9. Such models, when used for phone recognition [40], give very accurate results which turn out to be even better than RNNs. It is not easy to train DBNs with many hidden layers because it is hard to determine the posterior distribution for the variables in the hidden layer. After assessing algorithms like Markov chain Monte Carlo [42] and CD, the most optimal training for DBNs turned out to be the complementary priors algorithm coupled with the wake sleep algorithm [43]. A DBN consists of multiple hidden layers out of which two of the top-most layers have undirected connections whereas the rest of the layers have generative weights and recognition weights, that is, specific directed weights in the upward and downward directions, respectively. The greedy algorithm (GA) used to train it works on the principle that each model in the sequence learns from its previous model. The input of a layer is the output from the previous layer which is obtained by performing a nonlinear transform on the vector of data representations. As a result, individual models in the stack have varying representations of the actual data. All the layers in the model have an equal number of units. Figure 6.9(b) shows a model of the DBN as described so far. The weights between the first two layers are learned based on the assumption that the parameters in the higher layers will formulate a complementary prior [43] for $W_0$. To construct the vector input for $\mathrm{H}_1$, that is, the first hidden layer, the data in $V_0$ is mapped through $W_0^T$. This is a generative model which can be perfected by using a GA that states – $\mathrm{W}_0$ is learnt with the assumption that all other weight matrices are cinched after which $W_0$ is fixed and $W_0^T$ is used to generate vectors for $H_1$ even if consecutive changes in the weights of the higher layers indicate that this assumption is incorrect. The higher-level weight matrices are then locked to each other but kept unbound with $W_0$ and an RBM model is learnt for the higher-level data that was generated using $W_0^T$ to reconstruct the original input data vector. Any changes made to the higher-level weight matrices by the GA results in an improvement in the generative model. The GA is tested recursively and CD is used for learning the tied weights while being separated from the weights of the bottom layers. In this manner, weights are determined layer-wise efficiently though not optimally as the model might suffer from under fitting. To solve this problem, back-fitting is used which is a variation of the wake-sleep algorithm [44]. After learning the initial weights from GA, the recognition weights used for deduction and the generative weights used for defining the model are separated for further training. In the bottom-up pass, the recognition weights help determine the states of the hidden units stochastically and the generative weights are modified as per the maximum likelihood algorithm while the weights between the two top-most layers of the RBM is adjusted by fitting it according to the penultimate layer's posterior distribution. In the top-down pass, mapping is done through the generative weights to activate units in the bottom layers stochastically while only modifying the recognition weights. The RBM at the top can either settle at its equilibrium distribution or undergo few stages of CD before attaining equilibrium, before commencing the top-down phase. The later proves to be better at generating varying modes of the data.

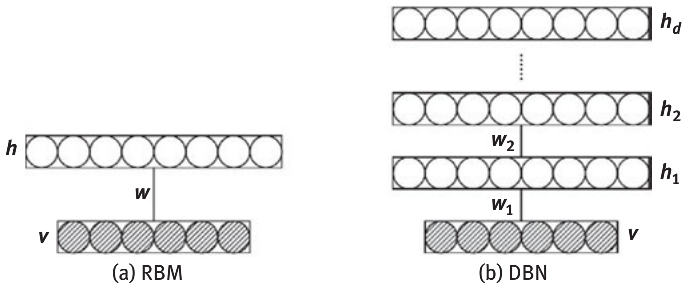


**Figure 6.9:** Deep belief networks as stack of restricted Boltzman machines .

In case of phone recognition, Backpropagating DBNs gave the best results, that is, the least phone error rate (PER) on the TIMIT dataset. As an input to the first visual layer, MFCCs along with their first and second temporal derivatives are used for each frame. The initial weight training for BP-DBNs follow the layer-wise CD approach with the addition of a final layer of outputs and then fine-tune the parameters of the model discriminatively using backpropagation. The number of output units equals the dimension of the binary representation of class labels which is decided arbitrarily. The probability assigned to each class, ʒ, given the ɖ-dimensional representation, ɚ, of an input speech fragment generated by the model, ϴ, and the vector of class ɮ’s transformed posterior probability distribution, $\text{ɔ}_{\text{ɮ}}$, is given in eq. (6.42).

$$P(\text{ɮ}|\theta, \text{ɚ}) = \frac{e^{\text{ɔ}\text{ɮ}\text{ɚ}}}{\sum_{\text{ƈ} \in classes} e^{\text{ɔ}_{\text{ƈ}}\text{ɚ}}} \tag{6.42}$$

The best results were given by a four-layer BP-DBN each containing 2048 units, with an additional output layer of 128 units.

### 6.3.4 WaveNets

WaveNets are probabilistic models [9] that give state-of-the-art performances in speech and music generation while being commendable at discriminative tasks like phone recognition as well. Audio generated by WaveNets sound more natural than present day concatenative and parametric methods, as a single model can successfully capture the characteristics of different speakers and generate speech conditioned on any particular speaker. Unlike other models where training is performed on extracted features, WaveNets are trained on the raw audio waveform. The combined probability of generating a particular waveform is expressed as products of probabilities of each sample conditioned on all previous samples as shown in eq. (6.43).

$$p(\psi) = \prod_{t=1}^{T} p(\psi_t|\psi_1, \ldots, \psi_{t-1}) \tag{6.43}$$

The above principle is implemented using many layers of causal convolution neural networks (CaCNNs) without the pooling layers and equal time dimension of input and output layers. The output of the model is calculated using a softmax layer that gives a categorical distribution indicating the probability of a sample, $\psi_t$, being generated. During training, the parameters are tuned to achieve the maximum log-likelihood of the data. Following (31), it is evident that during training the conditional probability for predicting the samples at each time step can be calculated parallelly as ψ is already known. Generation of audio is a sequential phenomenon where every sample generated is fed back into the network to make predictions for the subsequent time step. Though the training time for CaCNNs is lesser than RNNs due to the absence of recurrent connections, it takes a greater number of hidden layers or larger filters to increase the receptive field. An increased receptive field length is very important for capturing the dependencies in speech and musical signals during training to ensure naturalistic speech and harmonious music generation. To reduce computations resulting from additional hidden layers, a dilation factor (Df) is introduced in each layer, keeping the total number of layers in the model as constant. In dilated CaCNNs, the length of the filter is smaller than the area it is applied on and it works by skipping a specific number of intermediate input values as shown in Figure 6.10. The Df indicates the number of values to be skipped and helps increase the receptive field without the addition of more hidden layers.

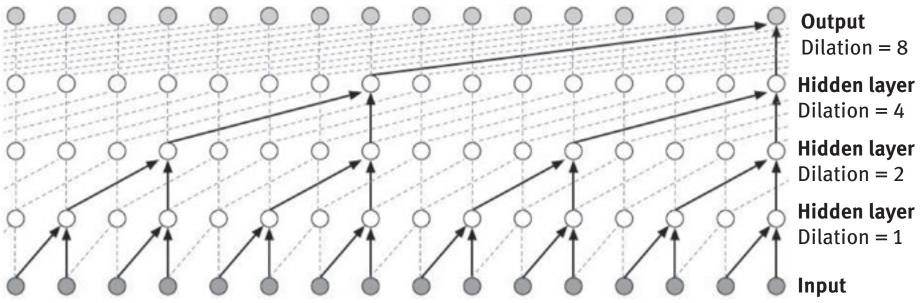


**Figure 6.10:** causal convolution neural networks with dilation.

A standard CaCNN has Df equal to 1 as shown in Figure 6.11. When comparing the CaCNNs of Figures 6.10 and 6.11, it can be observed that keeping the number of hidden layers constant, the receptive field of CaCNN in Figure 6.10 is more than thrice the receptive field of CaCNN in Figure 6.11 which results from the introduction of dilations in each layer in the former. Usually the Df is increased exponentially over the layers up to a certain point and the cycle is repeated again. A block of CaCNN with a Df of $1, F, F^2, F^3, \ldots, F^{L-1}$ over $L$ layers, respectively, has a receptive field of $2F^{L-1}$ and turns out to be better than a $(1 \times F^{L-1})$ convolutional model due to the increased receptive field and its capacity to capture the nonlinearities due to the hidden layers in between.

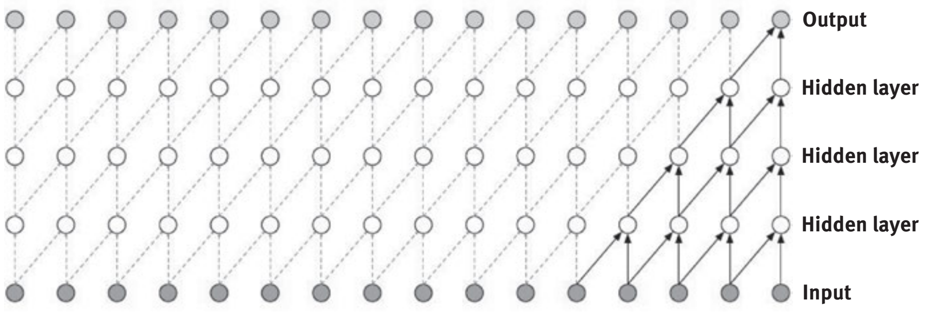


**Figure 6.11:** Standard causal convolution neural networks without dilation.

Raw audio is represented using 16 bits giving 65,536 possibilities which makes it a computation-intensive task so a μ-law transform as shown in eq. (6.44), is applied to the data to scale it down to 256 possibilities which can now be represented using 8 bits without trading the originality of the signal. The conditional probabilities in the output is then represented using a softmax distribution which works better than Gaussian scale mixture because the former makes no prior assumptions about the shape of the data.

$$f(\psi_t) = sign(\psi_t)\frac{\ln(1+\ \mu|\psi_t|)}{\ln(1+\ \mu)},\ where -1 < \psi_t < 1\ and\ \mu = 255 \tag{6.44}$$

The activations of each neuron in the kth layer is given by eq. (6.45), where * is the convolution operator, Θ is the element-wise product operator, *f* denotes the filter and *g* denotes the gate. The weight matrices *W*, are the learnable parameters also known as the convolution filter and *σ*(.) is the sigmoid function. This form of activation works significantly better for audio signals as compared to the linear ReLU activation layers.

$$z = \tanh(W_{f,k} {}^{*}\psi)\,\Theta\,\sigma(W_{g,k} {}^{*}\psi) \tag{6.45}$$

Figure 6.12 shows the overall representation of the above model. From (43), it is evident that WaveNets can be conditioned on external factors, $\mathcal{E}$, such as speaker identity, text, and musical tags, such as genre or type of instruments as shown in eq. (6.46). Two types of conditioning can be done with WaveNets – global in which case one external factor has its influence entirely over all time steps; and local conditioning where the external factor is time dependent and has to undergo a transformation

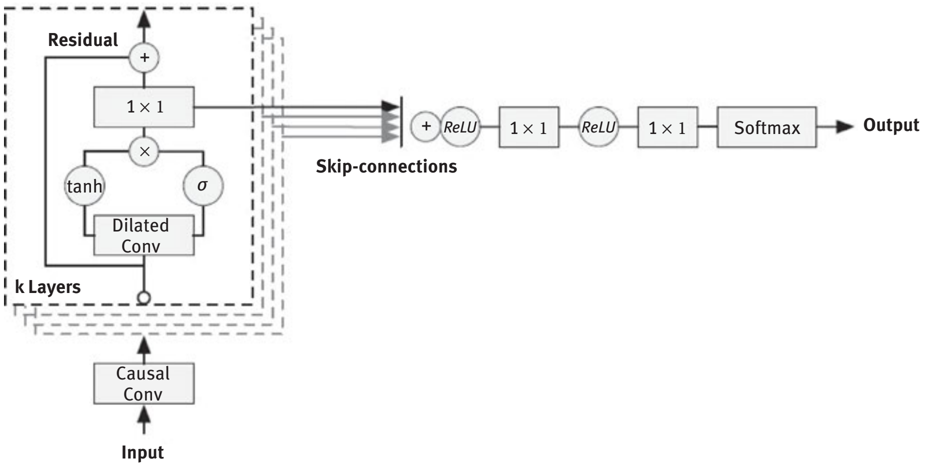


**Figure 6.12:** Overall representation of waveNet architecture.

for upsampling to make up for its lower sampling rate, that is, y = f($\mathcal{E}$). The activations for global and local conditioning is shown in eqs. (6.47) and (6.48), respectively, where $V_{*,k}$ is a trainable parameter and $V_{*,k}^{T}\mathcal{E}$ is a vector used over the entire time duration.

$$p(\psi|\mathcal{E}) = \prod_{t=1}^{T} p(\psi_t|\psi_1, \ldots, \psi_{t-1}, \mathcal{E}) \tag{6.46}$$

$$z = \tanh\left(W_{f,k}*\psi + V_{f,k}^{T}\mathcal{E}\right) \Theta\, \sigma\left(W_{g,k}*\psi + V_{g,k}^{T}\mathcal{E}\right) \tag{6.47}$$

$$z = \tanh\left(W_{f,k}*\psi + V_{f,k}^{T}*y\right) \Theta\, \sigma\left(W_{g,k}*\psi + V_{g,k}^{T}*\mathrm{y}\right) \tag{6.48}$$

When used for speech generation without conditioning it on text, it generates audio which sounds like speech but does not contain any actual words as it does not know what to say. But this model could be used to generate language-like sounds conditioned on different speakers based on their identity which is one-hot encoded and fed into the model during training as well as during generation. The accuracy with which a single WaveNet model can generate voices of different speakers shows that its internal architecture is very efficient in capturing the characteristics of multiple speakers. For text to speech (TTS), the WaveNet is trained on the log of the fundamental frequencies as well as linguistic features extracted from the input texts. This gave significantly better mean opinion score results (which indicates how natural the generated speech sounds) than LSTM-RNN parametric and HMM-driven concatenative models, when tested on North-American English and Mandarin Chinese. For music generation, the WaveNets can be trained on music datasets which are annotated with relevant information like genre, mood, volume, and tempo that can be used as conditioning factors during generation. A general observation with music generation is that longer receptive fields are essential to capture musical dependencies between notes and result in more melodious pieces. Experiments have also been performed to apply WaveNets in the domain of SR, where raw audio is fed as input and the model is slightly modified by adding a pooling layer followed by normal convolutional layers to the existing model and trained to perfect two tasks predicting the next frame and classifying the present one. WaveNets give the least PER rates when compared to any model trained on the TIMIT dataset that takes raw audio as input. The time required to train WaveNets can be optimized by using parallel architectures as demonstrated in [45].

## 6.4 Current applications

Audio signal processing, as mentioned already, is an integral part of AI and has huge contributions towards its advancements. It has a wide range of applications – chatbots that serve as personal assistants in banks, tourism websites, messaging applications;

labelling different regions in videos as per the sound emitted using environmental sound classification, or detecting the surrounding by processing an audio clip; music genre classification can help classify any new untagged music, the same models can be used to classify different types of cries of babies in order to identify their needs, or predict the different types of animals or birds by the sounds emitted by them. Speech or music generation can also be used for producing context specific text or genre specific music which can be used in story making applications or music applications. In this section, a little brief on few of the applications is provided.

## 6.4.1 Speech to text converter

STTs are used in almost all applications these days as a voice search utility or video transcript generation or subtitles. It can also be used by disabled people who cannot write using their hands as a dictation tool. For a long time the HMMs trained using expectation–maximization algorithm along with Gaussian mixture models were unbeatable [46].With the invention of increasingly efficient machine learning (ML) algorithms and hardware over the years, deep neural networks (DNNs) vanquished all of the prevailing methods [8]. The best STTs so far have been built by Google using parametric [47] and concatenative [48] models. The most elementary operation for STT is phoneme detection and a particular word is then formed by a concatenated phoneme sequence. The commencement of any SR starts with feature extraction which can either be MFCCs or LPCs after which various classification algorithms are applied. Detailed descriptions of models such as HMMs and LSTM-RNNs used to build STTs have been given in the previous sections, waveNets being the one to give the best performances. Languages usually have specific linguistic features like grammar and statistics like frequencies of bigrams, trigrams, and so on. These features are characteristic of languages and can be learnt hence known as linguistic or language model. To enhance STT, sometimes the acoustic probabilities are combined with the

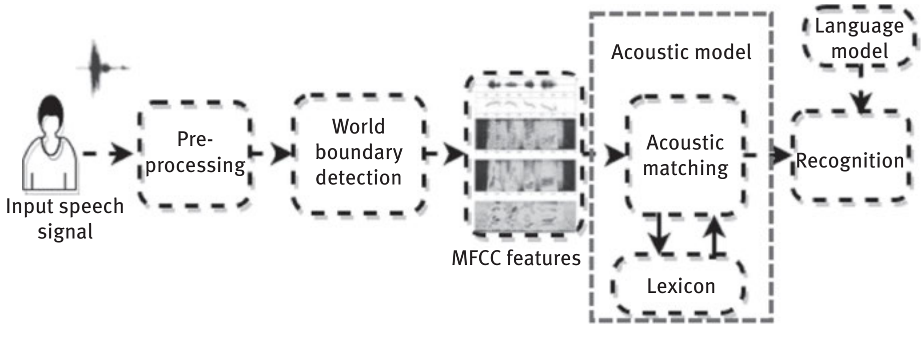


**Figure 6.13:** Speech recognition module.

linguistic probabilities and the product of both the results are combined to determine the word said. Though existing models give good enough results, efforts are being made to better the performances in continuous SR. Figure 6.13 shows a comprehensive view of a speech recognition module.

### 6.4.2 Audio classification

Tasks like predicting the type of surroundings from an indistinct clip like that of a busy road or an airport or the inside of a bus, predicting the bird species given their chirping sound, the genre of a given song, or any task where the outputs are required to be distinct labels, belong to this domain. It is helpful in various arenas like surveillance [49] and contextwise computing [50]. Be it songs or any sound, every audio signal belonging to a specific category have common features that might not be perceived by the human ear but can be detected by the feature extraction algorithms. In case of music genre classification, MFCCs are extracted and fed into deep feed forward neural networks and trained against genres which are fed as class tags after one-hot encoding. MFCCs combined with timbral features also give very good classification accuracies [7]. In [51], a new feature extraction method known as Daubechies wavelet coefficient histograms was devised and tested with classification algorithms like linear discriminant analysis and support vector machines; the classification results thus obtained were more accurate than the former mentioned approach. In case of environmental sound classification, once again MFCCs are extracted and this time DCNNs [5] are chosen as they give the best results for this kind of classification where once again the different classes such as airport, bus, busy street, shopping mall, and railway station are one-hot encoded and act as output labels. Similar approaches can be taken for any other type of classification for instance musical instrument classification, where features are extracted and fed into a neural network model to perform supervised training.

### 6.4.3 Speech and music generation

AI has made huge advancements in the fields of music generation where a model is trained against huge musical datasets so that it can generate new music on its own. WaveNets vanquished in the competition and give appreciable results when it comes to either music or speech generation. Speech generation conditioned on text is used in applications like audio books and it is flexible enough to generate different voices. It also helps in communication where one does not know the correct pronunciation but knows how to spell any word in that language. It is also helpful for clarification of pronunciations or learning a new language. Till date, TTS has been done mainly by playing pre-recorded sound which has been manually uttered. Storing such audio files on servers and fetching them on user request requires both huge amount of storage and increases the response time, the former being the bigger concern. Substituting the mechanism with a sound generating algorithm conditioned on text is quite a solution. Similarly, music generation can be conditioned on multiple factors such as tempo, genre, and instruments, which is of help to people who cannot play instruments but have a taste in music. Some music softwares which have incorporated AI and are running successfully are Orb Composer, IBM Watson Beat, OrchExtra, Google Magenta's NSynthSuper, ChucK, and Jukedeck.

### 6.4.4 Language translation

Language Translation becomes extremely important when intercultural meetings happen or anytime when travelling to another destination whose language is unknown. As a result, most often people find themselves using Google translate. Artificial intelligence's take on language translation is done by sequence to sequence learning [52] where deep LSTM-RNNs were used and gave excellent results comparable to the state of the art. Any sentence or word to be translated is considered to be a query sequence which is to be mapped to the target sequence in the other language to which the translation is required. Another solution to the translation problem has been proposed using RNN encoder–decoder [53]. They are capable enough to grasp sensible phrase and sentence representations.

### 6.4.5 Personal assistants

Ever since Apple introduced its personal assistant, Siri, there has been huge competition among the top organizations to introduce their own efficient chatbots. Consequently, Amazon's Alexa and Google's Google Home have entered the battlefield. These bots are not just mere question–answer systems, but their extended functionalities allow them to be linked to other applications such as electricity controls that allows them to collaborate with smart home architectures. Users can command them to play music, switch on/off electrical appliances, ask them questions, interact with them, have friendly conversations, make phone calls or appointments, and a ton of other things. To be able to provide such services, the personal assistant devices have to be linked to the external modules using application program interface. Some of the commonly used Personal Assistants are depicted in Figure 6.14.

These days even banks, hospitals, or travel websites have their own chatbots with which any user can interact to clear their own doubts. In such cases, the chatbots act as question–answer models [54–57]. In such models there are multiple components that work together – the input device which acquires either a voice input or a text input, a processing unit that converts the input into computer understandable form, a conversion unit, and a database. In the processing unit if the input is in the form of

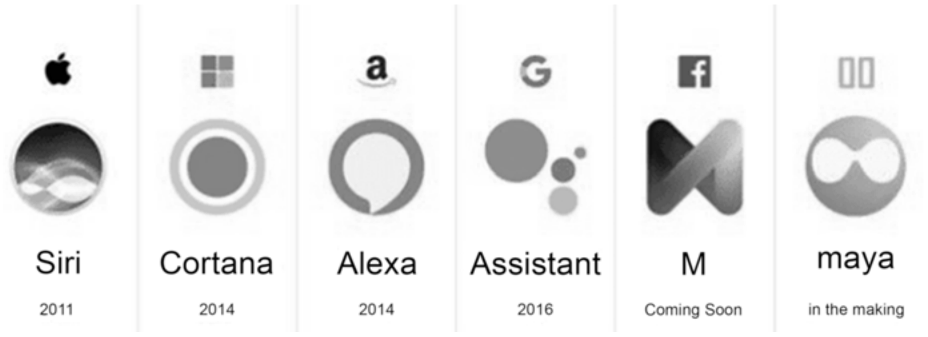


**Figure 6.14:** Personal assistants that made it big.

text then there are mechanisms to understand the language used which is done using information about the character encoding and other metadata. In case of voice inputs, the processing is done using speech recognizers mentioned earlier. After the input is processed, algorithms are deployed that perform domain classification which help to understand the query and the context to which it belongs [58, 59]. The natural language question is then converted to a query in a language that the database will respond to, for example Structured Query Language. Information is then retrieved from the database as answers to the query which is then output to the user interface either as voice using TTS conversion [60] or text output. This is summarized in Figure 6.15.

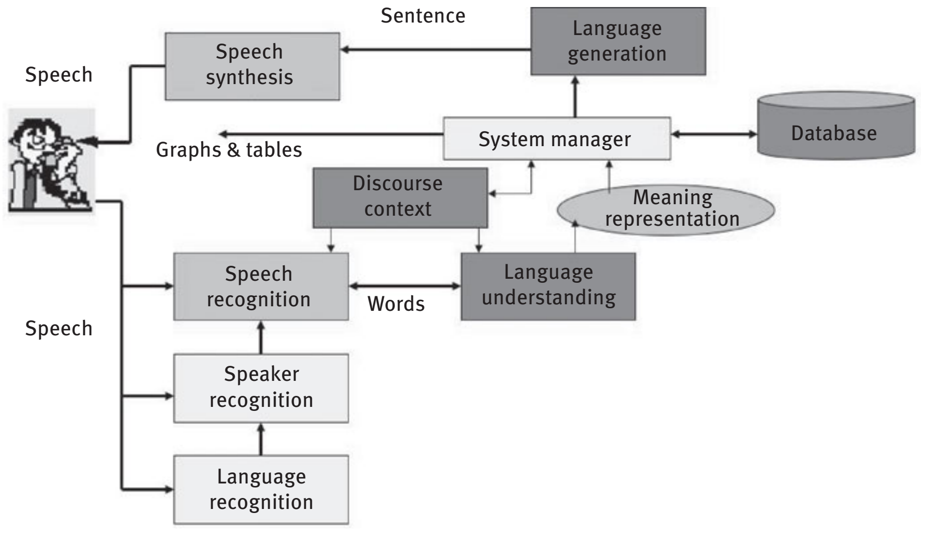


**Figure 6.15:** Summary of a question–answer model.

## 6.5 Future scope

With the amount of research going on in this domain, we can expect to see quite miraculous applications in the future. The existing architectures though good, still have a long way to go in terms of cognitive skills and accuracy. Most of the work related to speech and translations still need to be extended to different languages and various accents of the same language. More varieties of deep architectures are being explored to perfect the art of conversations with machines. As new and improved hardware is made available, problems related to storage are resolved and parallel architectures are allowing incredibly fast processing time. Therefore, the training of ML architectures will be more perfect with time. With availability of more information, NLP is reaching new heights; soon people will find friends in chatbots and there will be commercialization of personal assistants that can assist teachers, doctors, or even give a tough competition to them.

## 6.6 Conclusion

This chapter mainly focusses on drawing a clear picture about an audio signal and methods to process it and finally understand it. Physics has put forth the nitty-gritties about sound ages ago and only recently with the advent of AI, effort is made to make the computer understand sound – be it speech, music, or noise. Many algorithms and models have been tried in the past but due to constraints in hardware and availability of labelled data, efficient algorithms could not be built to recognize or understand sound. Given that an audio signal is quite complex, simple algorithms were not good enough to capture the characteristics or complexities. Today, the availability of huge amount of data has sparked research in innumerable arenas and it is rightly said that “Data is the new Currency.” New and improved hardware along with data has allowed researches to devise efficient algorithms that can process audio signals better. Deep architectures have always given better performances as they are able to study the higher order dependencies between data – this is proved by countless research in this field, a fraction of which has been mentioned and detailed in this chapter. As is seen from studies, DNNs are beating other pre-existing models in terms of accuracy and are able to model AI better. Currently, they are the best in image classification and used in multiple arenas like self-driving cars, healthcare, and giving pretty good performance in fields such as brain-computer interfaces and audio classification. Audio processing is one of the fundamental human senses and with the help of DNNs it is being successfully incorporated in machines, contributing to a more complete AI.

## References

[1] Bradski, G., Kaehler, A. Learning OpenCV. 1st Edition. Learning. 2008. https://doi.org/10.1109/MRA.2009.933612.

[2] Wide, P., Winquist, W., Bergsten, P., & Petriu, E.M. The human-based multisensor fusion method for artificial nose and tongue sensor data. IEEE Transactions on Instrumentation and Measurement, 1998. https://doi.org/10.1109/19.746559.

[3] Efimenko, K., Rackaitis, M., Manias, E., Vaziri, A., Mahadevan, L., & Genzer, J. Nested self-similar wrinkling patterns in skins. Nature Materials, 2005. https://doi.org/10.1038/nmat1342.

[4] Li, J., Monroe, W., Ritter, A., Jurafsky, D., Galley, M., & Gao, J. Deep Reinforcement Learning for Dialogue Generation. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 2016. https://doi.org/10.18653/v1/D16-1127.

[5] Salamon, J., & Bello J.P. Deep convolutional neural networks and data augmentation for environmental sound classification. IEEE Signal Processing Letters, 2017. https://doi.org/10.1109/LSP.2017.2657381.

[6] Martin, K.D., & Kim, Y.E. Musical instrument identification: a pattern-recognition approach. The Journal of the Acoustical Society of America, 1998. https://doi.org/10.1121/1.424083.

[7] Tzanetakis, G., and Cook, P. Musical genre classification of audio signals. IEEE Transactions on Speech and Audio Processing, 2002. https://doi.org/10.1109/TSA.2002.800560.

[8] Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A., Jaitly, N., Senior, A. et al. Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Processing Magazine, 2012. https://doi.org/10.1109/MSP.2012.2205597.

[9] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. WaveNet: a generative model for raw audio. Conference on Neural Information Processing Systems 2016, 1–15. https://doi.org/10.1109/ICASSP.2009.4960364.

[10] Muda, L., Begam, M., & Elamvazuthi, I. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. JOURNAL OF COMPUTING, VOLUME 2, ISSUE 3, 2010, 2(3), 138–43. https://doi.org/10.5815/ijigsp.2016.09.03.

[11] Juang, B.H., & Rabiner, L.R. Hidden Markov models for speech recognition. Technometrics, 1991. https://doi.org/10.1080/00401706.1991.10484833.

[12] Chakraborty, Ch., & Talukdar, P.H. Issues and limitations of HMM in speech processing: a survey. International Journal of Computer Applications, 2016.

[13] Sak, H., Senior, A., & Beaufays, F. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. Interspeech. 2014. https://doi.org/arXiv:1402.1128.

[14] Lee, H., Pham, P., Largman, Y., & Ng, A. Unsupervised feature learning for audio classification using convolutional deep belief networks. Nips, 2009. https://doi.org/10.1145/1553374.1553453.

[15] Piczak, K.J. Environmental Sound Classification with Convolutional Neural Networks. IEEE International Workshop on Machine Learning for Signal Processing, MLSP, 2015. https://doi.org/10.1109/MLSP.2015.7324337.

[16] Reyes-Galaviz, O., Arch-Tirado, E., & Reyes-Garcia, C.A. Classification of Infant Crying to Identify Pathologies in Recently Born Babies with ANFIS. Computers Helping People with Special Needs. 2004, vol. 3118 of Lecture Notes in Computer Science, 408–415. Doi: 10.1007/978-3-540-27817-7_60.

[17] Alberti, P.W. The Anatomy and Physiology of the Ear and Hearing. Occupational Exposure to Noise: Evaluation, Prevention and Control, 2001.
[18] Alías, F., Socoró, J., & Sevillano, X. A Review of Physical and Perceptual Feature Extraction Techniques for Speech, Music and Environmental Sounds. Applied Sciences, 2016. https://doi.org/10.3390/app6050143.
[19] Richard, G., Sundaram, S., & Narayanan, S. An Overview on Perceptually Motivated Audio Indexing and Classification. Proceedings of the IEEE, 2013. https://doi.org/10.1109/JPROC.2013.2251591.
[20] Katsiamis, A.G., Drakakis, E.M. & Lyon, R.F. Practical gammatone-like filters for auditory processing. Eurasip Journal on Audio, Speech, and Music Processing, 2007. https://doi.org/10.1155/2007/63685.
[21] Patterson, R.D., Robinson, K., Holdsworth, J., McKeown, D., Zhang, C., & Allerhand, M., Complex sounds and auditory images. Simulation. 1992. https://doi.org/10.1016/B978-0-08-041847-6.50054-X.
[22] Slaney, M. An Efficient Implementation of the Patterson-Holdsworth Auditory Filter Bank. Apple Computer Perception Group Tech Report. 1993.
[23] Wang, D., & Brown, G.J. Computational auditory scene analysis: principles, algorithms, and applications. Neural Networks, IEEE Transactions, 2008. https://doi.org/10.1109/TNN.2007.913988.
[24] Thompson, J.K., & Atlas, L.E. A Non-Uniform Modulation Transform for Audio Coding with Increased Time Resolution. 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. ICASSP ’03. https://doi.org/10.1109/ICASSP.2003.1199990.
[25] Mesgarani, N., Slaney, M., & Shamma, S.A. Discrimination of speech from nonspeech based on multiscale spectro-temporal modulations. IEEE Transactions on Audio, Speech and Language Processing, 2006. https://doi.org/10.1109/TSA.2005.858055.
[26] Yang, X., Wang, K., & Shamma, S.A. Auditory representations of acoustic signals. IEEE Transactions on Information Theory, 1992. https://doi.org/10.1109/18.119739.
[27] Mitrović, D., Zeppelzauer, M., & Breiteneder, C. Features for content-based audio retrieval. Advances in Computers, 2010. https://doi.org/10.1016/S0065-2458(10)78003-7.
[28] Logan, B. Mel Frequency Cepstral Coefficients for Music Modeling. International Symposium on Music Information Retrieval, 2000. https://doi.org/10.1.1.11.9216.
[29] Baum, L.E., Eagon, J.A. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. Bulletin of the American Mathematical Society, 1967. https://doi.org/10.1090/S0002-9904-1967-11751-8.
[30] Baum, L.E., Sell, G.R. Growth transformations for functions on manifolds. Pacific Journal of Mathematics, 1968. https://doi.org/10.2140/pjm.1968.27.211.
[31] Forney G.D. Jr. The Viterbi Algorithm. Proceedings of the IEEE, 1973. https://doi.org/10.1109/PROC.1973.9030.
[32] Viterbi, A.J. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory, 1967. https://doi.org/10.1109/TIT.1967.1054010.
[33] Dempster, A.P., Laird, N.M., & Rubin, D.B. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society. Series B, 1977. https://doi.org/10.2307/2984875.
[34] Levinson, S.E., Rabiner, L.R., & Sondhi, M.M. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. Bell System Technical Journal, 1983. https://doi.org/10.1002/j.1538-7305.1983.tb03114.x.

[35] O'Shaughnessy, D. Linear predictive coding. IEEE Potentials, 1988. https://doi.org/10.1109/45.1890.

[36] Makhoul, J., Roucos, S., & Gish, H. Vector Quantization in Speech Coding. Proceedings of the IEEE, 1985. https://doi.org/10.1109/PROC.1985.13340.

[37] Rabiner, L.R., Levinson, S.E., & Sondhi, M.M. On the application of vector quantization and hidden Markov models to speaker-independent, isolated word recognition. Bell System Technical Journal, 1983. https://doi.org/10.1002/j.1538-7305.1983.tb03115.x.

[38] Graves, A., Mohamed, A., & Hinton, G. Speech Recognition with Deep Recurrent Neural Networks. ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing – Proceedings, 2013. https://doi.org/10.1109/ICASSP.2013.6638947.

[39] Graves, A., & Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks, 2005. https://doi.org/10.1016/j.neunet.2005.06.042.

[40] Mohamed, A., Dahl, G., & Hinton, G. Deep Belief Networks for Phone Recognition. NIPS 22 Workshop on Deep Learning for Speech Recognition, 2009. https://doi.org/10.4249/scholarpedia.5947.

[41] Dahl, G., Abdel-Rahman M., & Hinton, G. Phone recognition with the mean-covariance restricted boltzmann machine. NIPS, 2010. https://doi.org/10.1586/ern.12.52.

[42] Neal, R.M. Connectionist learning of belief networks. Artificial Intelligence. 1992. https://doi.org/10.1016/0004-3702(92)90065-6.

[43] Hinton, G.E., Osindero, S., & The, Y.W. A fast learning algorithm for deep belief nets. Neural Computation, 2006. https://doi.org/10.1162/neco.2006.18.7.1527.

[44] Hinton, G.E., Dayan, P., Frey, B.J., & Neal, R.M. The 'Wake-Sleep' algorithm for unsupervised neural networks. Science, 1995. https://doi.org/10.1126/science.7761831.

[45] van den Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., van den Driessche, G., Lockhart, E. et al. Parallel WaveNet: Fast High-Fidelity Speech Synthesis, 28 November 2017. ArXiv: 1711.10433v1[Cs.LG]

[46] Juang, B.H., Levinson, S.E., Sondhi, M.M. Maximum likelihood estimation for multivariate mixture observations of Markov chains. IEEE Transactions on Information Theory, 1986. https://doi.org/10.1109/TIT.1986.1057145.

[47] Zen, H., Tokuda, K., & Black, A.W. Statistical parametric speech synthesis. Speech Communication, 2009. https://doi.org/10.1016/j.specom.2009.04.004.

[48] Godlewski jun, E. Der Eireifungsporze\ im Lichte der Untersuchung der Kernplasmarelation bei Echinodermenkeimen. Archiv für Entwicklungsmechanik der Organismen, 1918, 44(3–4), 499–529. https://doi.org/10.1007/BF02554390.

[49] Radhakrishnan, R., Divakaran, A., & Smaragdis, P. Audio analysis for surveillance applications. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 2005. https://doi.org/10.1109/ASPAA.2005.1540194.

[50] Chu, S., Narayanan, S., & Kuo, C.C.J. Environmental sound recognition with time-frequency audio features. IEEE Transactions on Audio, Speech and Language Processing, 2009. https://doi.org/10.1109/TASL.2009.2017438.

[51] Li, T., Ogihara, M., & and Li, Q. A comparative study on content-based music genre classification. Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval – SIGIR '03, 2003. https://doi.org/10.1145/860435.860487.

[52] Sutskever, I., Vinyals, O., Le, Q.V. Sequence to sequence learning with neural networks. NIPS'14: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, 2014, 1–9. arXiv:1409.3215 [cs.CL]

[53] Huang, S., Cheng, C., Chiang, C., & Chang, C. Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation Kyunghyun. Circuits and Systems, APCCAS 2006. IEEE Asia Pacific Conference On, 2006. https://doi.org/10.1109/APCCAS.2006.342179.

[54] Hermjakob, U. Parsing and Question Classification for Question Answering. Proceedings of the Workshop on ARABIC Language Processing Status and Prospects, 2011. https://doi.org/10.3115/1117856.1117859.

[55] Li, X., & Roth, D. Learning Question Classifiers. Proceedings of the 19th International Conference on Computational Linguistics, 2002. https://doi.org/10.3115/1072228.1072378.

[56] Ravichandran, D., & Hovy, E. Learning Surface Text Patterns for a Question Answering System. Proceedings of the 40th Annual Meeting on Association for Computational Linguistics – ACL '02, 2001. https://doi.org/10.3115/1073083.1073092.

[57] Zhang, D., & Lee, W.S. Question Classification Using Support Vector Machines. Proceedings of the 26th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2003. https://doi.org/10.1145/860435.860443.

[58] Ravuri, S., & Stolcke, A. A Comparative Study of Neural Network Models for Lexical Intent Classification. 2015 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2015 – Proceedings, 2016. https://doi.org/10.1109/ASRU.2015.7404818.

[59] Ravuri, S., & Stolcke, A. A Comparative Study of Recurrent Neural Network Models for Lexical Domain Classification. ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing – Proceedings, 2016. https://doi.org/10.1109/ICASSP.2016.7472844.

[60] Capes, T., Coles, P., Conkie, A., Golipour, L., Hadjitarkhani, A., Hu, Q., Huddleston, N. et al. Siri On-Device Deep Learning-Guided Unit Selection Text-To-Speech System. Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH, 2017. https://doi.org/10.21437/Interspeech.2017-1798.

[61] Jelinek, F. Continuous Speech Recognition by Statistical Methods. Proceedings of the IEEE, 1976. https://doi.org/10.1109/PROC.1976.10159.

Mahua Bose and Kalyani Mali

# 7 Backpropagation Through Time Algorithm in Temperature Prediction

**Abstract:** In deep learning, data is transmitted through a number of layers in the feedforward network between input and output layers. In a recurrent network, data may propagate through a layer several times. Backpropagation through time (BPTT) technique is used to train recurrent networks (RNN). The underlying idea of BPTT is to transform a recurrent network into an unfolded feedforward network (multilayer network) where conventional backpropagation learning is used for gradient calculation. Here, each layer of the unfolded network represents a time step. The objective of this study is to integrate concept of BPTT in the framework of fuzzy time series prediction. The model takes sequence of previous values as input (fuzzy inputs) to the different layer of the unfolded network and produces fuzzy output. Temperature dataset is used to evaluate the performance of the model and prediction accuracy of BPTT is better than that of backpropagation neural network model.

**Keywords:** fuzzy set, layer, neural network, order, unfolding

## 7.1 Introduction

Deep learning technique is applicable to different fields such as image and speech processing, pattern recognition, bioinformatics, and so on. Recently, it is employed in monsoon rainfall prediction [1, 2] also. Objective of this research is to apply deep learning in the context of fuzzy time series (FTS) forecasting.

FTS forecasting is a branch of traditional time series forecasting where observations are fuzzy sets [3] and a relationship is established between the current observation and previous observation. The topic [4–6] was first introduced in 1993 and 1994. Then it is presented in the simplified form [7] in the year 1996. In the first-order forecasting model, present state is dependent on previous state only. In high order forecasting model present state depends on more than one previous state. FTS model has already been applied in various application areas such as student enrolment [5–10], temperature [10–11], unemployment [8], stock exchange [8–14], and so on.

Backpropagation neural network (BPNN) is integrated successfully in fuzzy forecast models by many researchers [9–14]. In the past, for the high order models

**Mahua Bose, Kalyani Mali,** Department of Computer Science and Engineering, University of Kalyani, Nadia, West Bengal, India

https://doi.org/10.1515/9783110670905-007

using BPNN, values of previous time periods are fed together at the input layer at a time and there is a single hidden layer. But in this work, inputs for the previous time periods are fed into different layer of the network instead of same layer. This is the main difference between the proposed approach and the existing neural network based high order FTS models. There are many variations of BPTT algorithm. Description of different types of BPTT algorithms is found in the published work by Williams and Zipser in the year 1995[15]. In epochwise BPTT algorithm, weight updations are done only at the end of each training example sequence (which is called an epoch). Computation of gradient value is same in epochwise BPTT model and real-time BPTT model. But the basic difference between them is that, in epochwise BPTT in addition to the $\partial$ values, the error at each intermediate step is also considered. In truncated BPTT algorithm, denoted as BPTT ($h$), data values of previous $h$ time steps are used only.

This chapter presents epochwise BPTT model which is truncated in nature. The model takes sequence of previous values as input (fuzzy inputs) to the different layer of the unfolded network. Number of order is same as the number of time steps (or layers) in the network. This model is called TBPTT ($h1$, $h2$). Here, $h1$ is the number of the steps in the forward pass and $h2$ is the number of the steps in the backward pass. In this study, $h1 = h2$.

The work is presented in the following sections: In Section 7.2, description of BPTT model is provided. Section 7.3 describes the entire methodology. Comparison of the experimental results is shown in Section 7.4. Conclusion is presented in Section 7.5.

## 7.2 Basic concept of backpropagation through time

The basic principle of back propagation through time [16–17] is to unfold the network at the time steps 1, 2, . . ., $T$ to store longer history of information. The unfolding technique converts a recurrent network into an equivalent feedforward network with multiple steps of computation. The unfolded network is trained with back propagation and all copies of each equivalent connection weight should be identical. Updations of weights are done using the sum of the gradients calculated for weights in each layer [Figure 7.1].

Basic algorithm:

1. Unroll the recurrent network for an arbitrary number of levels (in forward pass).
2. Back propagate the error (in backward pass).
3. Weight changes at each unrolled stages (connection weights of the equivalent connection on each fold) are added to get a single set of weight changes.

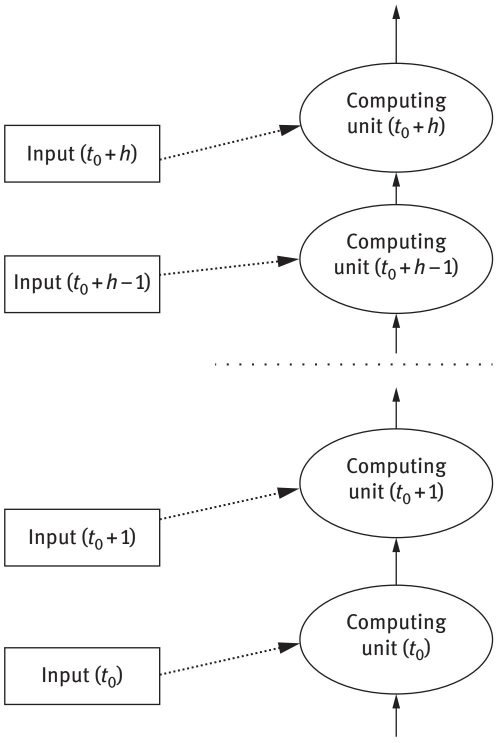


**Figure 7.1:** Example of unrolling.

There are many variations of BPTT algorithm. In this study, Epochwise BPTT [15] will be employed for prediction. Let us consider an unfolded network. The output of $n$-th processing node at time $t + 1$ is expressed as,

$$S_n(t+1) = \sum_{l \in u} w_{nl} y_l(t) + \sum_{l \in I} w_{nl} x_l(t+1) \tag{7.1}$$

In each node, output at time $t + 1$ is a function of the current input $x$ $(t + 1)$ and previous output value $Y$ $(t)$. $x_1$ $(t)$ denotes the external input signals to the $n$th node at time $t$. The element $w_{\text{nl}}$ represents the weight on the link from the $l$th node to the $n$th node. Set of input nodes represented by $I$ and $u$ denotes the set of nodes in the recurrent layer. Each processing node applies the logistic function:

$$Y_n(t+1) = f_n(S_n(t+1)) \tag{7.2}$$

$$f'_n(S_n(t+1)) = Y_n(t+1)[1 - Y_n(t+1)] \tag{7.3}$$

Epochwise BPTT process proceeds by repeating eqs. (7.4) and (7.5).

$$\in_n(t;F) = \begin{cases} e_n(t;F) & \text{if } t = T \\ e_n(t;F) + \sum_{l \in u} w_{ln} \partial_l(t+1;F) & \text{if } t < T \end{cases} \tag{7.4}$$

$$\partial_n(t;F) = f'_n(S_n(t)) \in_n(t;F) \tag{7.5}$$

For *n*th node, error "*e*" at time "*t*" is calculated by subtracting actual output from the target.

Each connection weight $w_{ij}$ is updated as follows:

$$\left(\partial F / \partial w_{ij}\right) = (\text{learning rate})^* \sum_{t=0}^{T-1} x_j(t) \partial_i(t+1, F) \tag{7.6}$$

"*x*" terms in the equation represent either the output of a processing node or an external input node of the network at time *t*. Final time step is denoted by *T*.

## 7.3 Proposed method

In this study, epochwise BPTT algorithm [15] is applied for forecasting high-order FTS. Computation is done by truncating the backward propagation of information to a predefined number of time steps. So this model is a combination of epochwise BPTT and truncated BPTT [Figure 7.2]. Steps of the proposed BPTT-based approach are as follows.

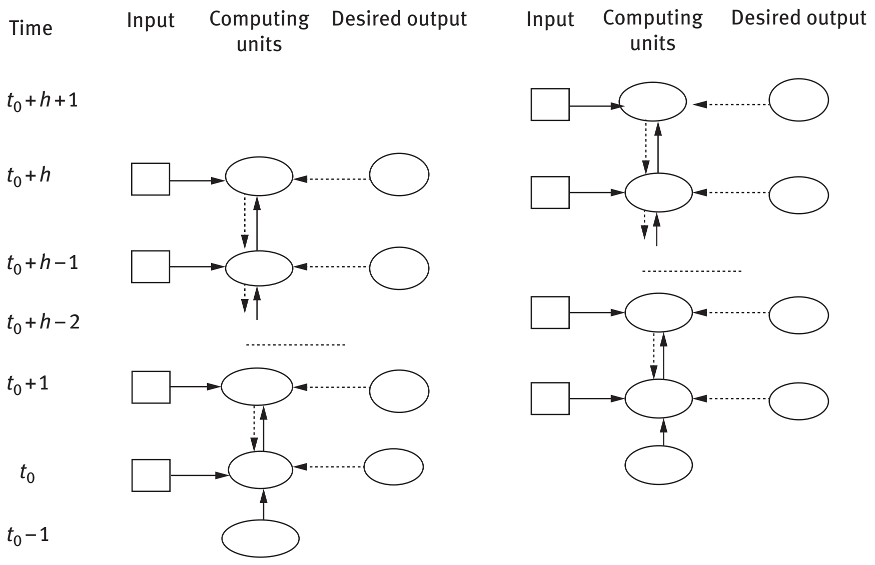


**Figure 7.2:** Epochwise BPTT (truncated).

Step 1. Apply fuzzy c-means (FCM) algorithm [18] to create "$k$" clusters and obtain fuzzy membership value of each datapoint in each cluster.

Step 2. Choose number of prior time steps "$h$." Here, this is equal to the number of order of the model.

Step 3. Apply truncated version of epochwise BPTT as described below.
Let us consider, total number of datapoints (total time period) is "MAX" and number of prior time steps is "h." Each layer of the unfolded network takes fuzzy membership values as input.

**Algorithm:**

```
3.1. Initial activation = 0; iteration= 0.
3.2. $t = 1$;
3.3. Repeat steps 3.4–3.9 until $t \le$ MAX-$h$
3.4. $t_0 = t$; repeat steps 3.5–3.6 until $t_0 < t + h$
3.5. Calculate output values in the forward pass using eqs. (7.1)–(7.3).
3.6. $t_0 = t_0 + 1$.
3.7. Proceed in the backward direction repeating eqs. (7.4) and (7.5).
3.8. Update each connection weight using eq. (7.6).
3.9. $t = t + 1$.
3.10. Iteration = Iteration + 1
3.11. If Iteration < maximum iteration, go to step 3.2, else stop.
```

Step 4. Defuzzification:
Let us consider that $m$ clusters are created in the dataset. If the output membership values are $\mu_1, \mu_2, \dots\dots\dots, \mu_m$. and corresponding centers of the clusters are $c_1, c_2, \dots\dots\dots, c_m$. Then,

$$\text{Output} = \frac{\sum_{i=1}^{m} c_i \mu_i}{\sum_{i=1}^{m} \mu_i} \tag{7.7}$$

**Example:**
Let us consider temperature (August) data set. Fuzzy C-means algorithm is applied to create three clusters. Membership values are shown in Table 7.1. Number of input units and processing units are three in this case. Now for the fifth order model, number of prior time steps $h = 5$.

Now, the following sequence of membership values is fed into the system. Each layer of the unfolded network represents a time step. Each of three membership values is fed into the input layer of the network through three nodes.

| Time | External input |
|---|---|
| $t = 1$ | (0.71 0.25 0.04) |
| $t = 2$ | (0.01 0.05 0.95) |
| $t = 3$ | (0.01 0.05 0.95) |
| $t = 4$ | (0 0.02 0.98) |
| $t = 5$ | (0.02 0.11 0.87) |

**Table 7.1:** A fraction of August data set.

| **Days** | **Data** | **Membership values in cluster** | | |
|---|---|---|---|---|
| | | **1** | **2** | **3** |
| 1 | 27.1 | 0.71 | 0.25 | 0.04 |
| 2 | 28.9 | 0.01 | 0.05 | 0.95 |
| 3 | 28.9 | 0.01 | 0.05 | 0.95 |
| 4 | 29.3 | 0 | 0.02 | 0.98 |
| 5 | 28.8 | 0.02 | 0.11 | 0.87 |
| 6 | 28.7 | 0.03 | 0.22 | 0.75 |
| 7 | 29 | 0 | 0.01 | 0.99 |
| 8 | 28.2 | 0.02 | 0.91 | 0.07 |
| 9 | 27 | 0.82 | 0.15 | 0.03 |
| 10 | 28.3 | 0.03 | 0.81 | 0.15 |
| 11 | 28.9 | 0.01 | 0.05 | 0.95 |
| 12 | 28.1 | 0.01 | 0.97 | 0.02 |
| 13 | 29.9 | 0.05 | 0.13 | 0.82 |
| 14 | 27.6 | 0.1 | 0.85 | 0.05 |
| 15 | 26.8 | 0.96 | 0.03 | 0.01 |
| 16 | 27.6 | 0.1 | 0.85 | 0.05 |
| – | – | – | – | – |

Final output is compared with the values at $T = 6$, that is, (0.03 0.22 0.75). Back propagation is done and weights are updated. Then, the next sequence is fed into the system.

| Time | External input |
|---|---|
| $t = 2$ | (0.01 0.05 0.95) |
| $t = 3$ | (0.01 0.05 0.95) |
| $t = 4$ | (0 0.02 0.98) |
| $t = 5$ | (0.02 0.11 0.87) |
| $t = 6$ | (0.03 0.22 0.75) |

Final output is compared with the values at $T = 7$, that is, (0 0.01 0.99). Back propagation is done and weights are updated. Then, the next sequence is fed into the system and so on.

Let the center of three clusters be 26.59, 27.95, and 29.1 and the output memberships in the corresponding clusters are 0.035354, 0.388622, and 0.626119. Then, the output is calculated by eq. (7.7).

Output = ((26.59*0.035354) + (27.95*0.388622) + (29.1*0.626119)) / ((0.035354 + 0.388622 + 0.626119)) = 28.59.

## 7.4 Experimental results and discussion

### 7.4.1 Description of datasets

Experiments are carried out using four real-world data sets: daily average temperature dataset (June, July, August, and September) of Taipei, Taiwan (Tables 7.2–7.5).

**Table 7.2:** Actual and predicted values in °C for June dataset.

| Day | Actual | Predicted | Day | Actual | Predicted |
|---|---|---|---|---|---|
| 1 | 26.1 | | 16 | 28.8 | 28.09 |
| 2 | 27.6 | – | 17 | 29 | 29.12 |
| 3 | 29 | – | 18 | 30.3 | 28.2 |
| 4 | 30.5 | – | 19 | 30.2 | 28.37 |
| 5 | 30 | – | 20 | 30.9 | 29.77 |
| 6 | 29.5 | – | 21 | 30.8 | 29.42 |
| 7 | 29.7 | 29.68 | 22 | 28.7 | 29.51 |
| 8 | 29.4 | 29.42 | 23 | 27.8 | 29.82 |
| 9 | 28.8 | 29.07 | 24 | 27.4 | 28.19 |
| 10 | 29.4 | 28.87 | 25 | 27.7 | 27.84 |
| 11 | 29.3 | 28.97 | 26 | 27.1 | 28.07 |
| 12 | 28.5 | 29.097 | 27 | 28.4 | 28.44 |
| 13 | 28.7 | 29.04 | 28 | 27.8 | 28.86 |
| 14 | 27.5 | 28.41 | 29 | 29 | 28.94 |
| 15 | 29.5 | 28.34 | 30 | 30.2 | 28.94 |

### 7.4.2 Performance evaluation

FTS model of order fifth, sixth, seventh, and eighth are used in this study Here, number of prior time steps is equivalent to the number of order taken, that is, for fifth-order model, $h = 5$. In the first step, fuzzy-c-means algorithm is applied to create different number of clusters for each dataset and fuzzy logical relationships are defined using BPTT model. Fuzziness index $m = 2$; for temperature datasets error = 0.005. Number of clusters used in each dataset is shown in the Tables 7.6–7.9 with RMSE for in-sample data.

**Table 7.3:** Actual and predicted values in °C for July dataset.

| Day | Actual | Predicted | Day | Actual | Predicted |
|---|---|---|---|---|---|
| 1 | 29.9 | | 17 | 28.7 | 28.37 |
| 2 | 28.4 | – | 18 | 29.9 | 28.82 |
| 3 | 29.2 | – | 19 | 30.8 | 29.26 |
| 4 | 29.4 | – | 20 | 31.6 | 29.59 |
| 5 | 29.9 | – | 21 | 31.4 | 30.64 |
| 6 | 29.6 | – | 22 | 31.3 | 30.76 |
| 7 | 30.1 | 29.61 | 23 | 31.3 | 30.76 |
| 8 | 29.3 | 29.46 | 24 | 31.3 | 30.77 |
| 9 | 28.1 | 29.71 | 25 | 28.9 | 30.72 |
| 10 | 28.9 | 28.48 | 26 | 28 | 30.71 |
| 11 | 28.4 | 28.98 | 27 | 28.6 | 28.11 |
| 12 | 29.6 | 29.49 | 28 | 28 | 28.55 |
| 13 | 27.8 | 29.1 | 29 | 29.3 | 29.35 |
| 14 | 29.1 | 29.6 | 30 | 27.9 | 29.09 |
| 15 | 27.7 | 29.08 | 31 | 26.9 | 28.57 |
| 16 | 28.1 | 28.61 | – | – | – |

**Table 7.4:** Actual and predicted values in °C for August dataset.

| Day | Actual | Predicted | Day | Actual | Predicted |
|---|---|---|---|---|---|
| 1 | 27.1 | | 17 | 27.9 | 27.79 |
| 2 | 28.9 | – | 18 | 29 | 28.39 |
| 3 | 28.9 | – | 19 | 29.2 | 28.09 |
| 4 | 29.3 | – | 20 | 29.8 | 28.6 |
| 5 | 28.8 | – | 21 | 29.6 | 28.58 |
| 6 | 28.7 | 28.59 | 22 | 29.3 | 28.53 |
| 7 | 29 | 28.54 | 23 | 28 | 28.56 |
| 8 | 28.2 | 28.51 | 24 | 28.3 | 28.58 |
| 9 | 27 | 28.56 | 25 | 28.6 | 27.91 |

**Table 7.4** (continued)

| Day | Actual | Predicted | Day | Actual | Predicted |
|---|---|---|---|---|---|
| 10 | 28.3 | 27.93 | 26 | 28.7 | 28.47 |
| 11 | 28.9 | 27.79 | 27 | 29 | 28.45 |
| 12 | 28.1 | 28.46 | 28 | 27.7 | 28.53 |
| 13 | 29.9 | 28.59 | 29 | 26.2 | 28.6 |
| 14 | 27.6 | 27.94 | 30 | 26 | 27.87 |
| 15 | 26.8 | 28.57 | 31 | 27.7 | 27.76 |
| 16 | 27.6 | 27.77 | – | – | – |

**Table 7.5:** Actual and predicted values in °C for September dataset.

| Day | Actual | Predicted | Day | Actual | Predicted |
|---|---|---|---|---|---|
| 1 | 27.5 | | 16 | 28.3 | 29.81 |
| 2 | 26.8 | – | 17 | 28.6 | 28.5 |
| 3 | 26.4 | – | 18 | 28.1 | 27.89 |
| 4 | 27.5 | – | 19 | 28.4 | 28.05 |
| 5 | 26.6 | – | 20 | 28.3 | 27.26 |
| 6 | 28.2 | – | 21 | 26.4 | 27.39 |
| 7 | 29.2 | – | 22 | 25.7 | 28.3 |
| 8 | 29 | 27.78 | 23 | 25 | 26.95 |
| 9 | 30.3 | 29.31 | 24 | 27 | 25.74 |
| 10 | 29.9 | 29.6 | 25 | 25.8 | 26.75 |
| 11 | 29.9 | 29.55 | 26 | 26.4 | 26.44 |
| 12 | 30.5 | 29.73 | 27 | 25.6 | 26.21 |
| 13 | 30.2 | 29.62 | 28 | 24.2 | 25.76 |
| 14 | 30.3 | 29.74 | 29 | 23.3 | 25.64 |
| 15 | 29.5 | 29.74 | 30 | 23.5 | 25.73 |

**Table 7.6:** Error estimation for June data set using BPTT model.

| Error | Number of clusters | Order | | | |
|---|---|---|---|---|---|
| | | 5th | 6th | 7th | 8th |
| RMSE | 4 | 1.17 | 0.99 | 1.03 | 1.07 |
| | 5 | 1.08 | 1.1 | 1.4 | 1.02 |
| | 6 | 1.09 | 1.15 | 1.27 | 1.13 |
| Theil's *U* statistic | 4 | 0.021 | 0.02 | 0.02 | 0.02 |
| | 5 | 0.019 | 0.019 | 0.025 | 0.018 |
| | 6 | 0.019 | 0.02 | 0.022 | 0.02 |

**Table 7.7:** Error estimation for July data set using BPTT model.

| Error | Number of clusters | Order | | | |
|---|---|---|---|---|---|
| | | 5th | 6th | 7th | 8th |
| RMSE | 4 | 1.18 | 1.19 | 1.23 | 1.31 |
| | 5 | 1.15 | 1.14 | 1.14 | 1.15 |
| | 6 | 1.18 | 1.13 | 1.34 | 1.34 |
| Theil's *U* statistic | 4 | 0.02 | 0.02 | 0.021 | 0.022 |
| | 5 | 0.021 | 0.02 | 0.02 | 0.02 |
| | 6 | 0.02 | 0.019 | 0.023 | 0.023 |

**Table 7.8:** Error estimation for August data set using BPTT model.

| Error | Number of clusters | Order | | | |
|---|---|---|---|---|---|
| | | 5th | 6th | 7th | 8th |
| RMSE | 3 | 0.98 | 1.01 | 1.02 | 1.06 |
| | 4 | 1.13 | 1.18 | 1.16 | 1.12 |
| | 5 | 1.05 | 1.09 | 1.13 | 1.14 |
| Theil's *U* statistic | 3 | 0.017 | 0.018 | 0.018 | 0.019 |
| | 4 | 0.02 | 0.021 | 0.021 | 0.02 |
| | 5 | 0.019 | 0.019 | 0.02 | 0.02 |

**Table 7.9:** Error estimation for September data set using BPTT model.

| Error | Number of clusters | Order | | | |
|---|---|---|---|---|---|
| | | 5th | 6th | 7th | 8th |
| RMSE | 6 | 1.44 | 1.38 | 1.3 | 1.28 |
| | 7 | 1.32 | 1.36 | 1.23 | 1.3 |
| | 8 | 1.32 | 1.48 | 1.23 | 1.23 |
| Theil's *U* statistic | 6 | 0.025 | 0.026 | 0.023 | 0.023 |
| | 7 | 0.024 | 0.024 | 0.022 | 0.022 |
| | 8 | 0.024 | 0.027 | 0.022 | 0.022 |

Number of external input nodes is equal to the number of clusters taken. Number of external input nodes and number of computing units are equal. Number of iteration in each of the dataset is 100, learning rate = 0.5.

Predicted values are computed taking different combination of sequence length (prior time steps) and clusters. Then, prediction error is computed by using root mean square error (RMSE) and Theil's U statistic [19]. It is observed that the accuracy of forecast is dependent on the number of clusters created and the order of the model (number of layer used). For each dataset, only predicted values (corresponding to minimum RMSE) are shown (Tables 7.2–7.5).

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(A-P)^2}{n}} \tag{7.8}$$

$$U = \left(\sqrt{\sum_{i=1}^{n}(A-P)^2}\right) \Big/ \left(\left(\sqrt{\sum_{i=1}^{n}A^2}\right) + \left(\sqrt{\sum_{i=1}^{n}P^2}\right)\right) \tag{7.9}$$

where $A$ and $P$ represent actual and predicted values, respectively. Number of observations is $n$.

Error estimation for June data set using BPTT model is shown in Table 7.6. Using four clusters, there are four membership values and these are fed into the system as four external inputs. If we are using fifth order model then the number of layers in the unfolded network is five (i.e., $h = 5$) and membership values of previous five periods are fed into each layer of the network. In this case, RMSE value is 1.17. Similarly, other values are computed using different combination of sequence length (prior time steps) and clusters.

June dataset (Table 7.6) produces minimum RMSE (0.99) in sixth order ($h = 6$) using four clusters. Using July dataset (Table 7.7), minimum RMSE (1.13) is obtained in sixth order with six clusters. For August dataset (Table 7.8), minimum RMSE (0.98) is in fifth order ($h = 5$) using three clusters. Using September dataset (Table 7.9) minimum RMSE (1.23) is in seventh order with seven clusters and eight clusters. Using eight clusters in 8th order RMSE is 1.23 also.

It is revealed from the experimental results that the accuracy of forecast is dependent on the number of clusters created and the number of layers (i.e., order) of the model. Good forecasting accuracy is achieved if the value of $U$ (eq. (7.9)) is close to zero.

Performance of high-order BPNN model [11] using temperature dataset is also provided in Table 7.10. In this model, there is one input layer and hidden layer. Values of previous time periods are fed together at a time (number of input nodes is equal to the number of time periods). Middle points of the corresponding intervals are used as input. In this case, datasets for the months of June and July are partitioned into 17 intervals. Datasets for the months of August and September are partitioned into 20 and 21 intervals, respectively.

**Table 7.10:** Error estimation for BPNN model on temperature data [11].

| Error | Month | Order | | | |
|---|---|---|---|---|---|
| | | 5th | 6th | 7th | 8th |
| RMSE | June | 1.23 | 1.27 | 1.22 | 1.25 |
| | July | 1.33 | 1.36 | 1.37 | 1.47 |
| | August | 1.05 | 1.03 | 1.02 | 1.03 |
| | September | 1.35 | 1.43 | 1.39 | 1.57 |
| Theil's $U$ statistic | June | 0.021 | 0.022 | 0.021 | 0.022 |
| | July | 0.023 | 0.024 | 0.024 | 0.026 |
| | August | 0.019 | 0.019 | 0.018 | 0.019 |
| | September | 0.019 | 0.026 | 0.025 | 0.028 |

From the comparative study (using RMSE values) with BPNN model, it is seen that, BPTT model using fuzzy input produces better result for three of the datasets (June, July, and September). For August dataset improvement is marginal (Figure 7.3).

Actual and predicted values in Tables 7.2–7.5 are in °C. Predicted values in Tables 7.2–7.5 are based on in-sample data. So, first few days' values are not present. For example in case of June dataset, temperature values (membership) of previous six days (sixth order model) are used to predict the seventh value. It is possible to get forecast for the first few days of the month (July, August, and September) from its previous month's data. These out-of-sample observations are shown separately in Tables 7.11 and 7.12. Estimated RMSE values for July, August,

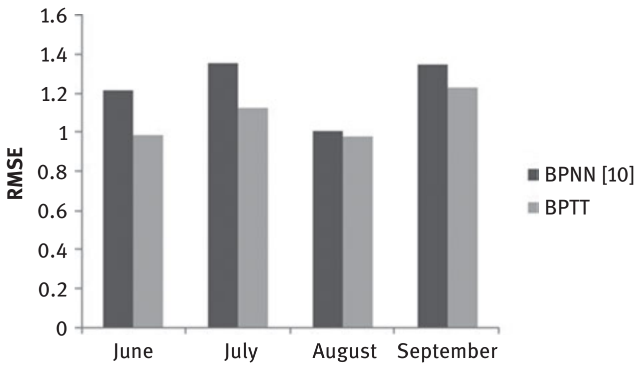


**Figure 7.3:** RMSE values: BPNN versus BPTT.

and September dataset are 1.01, 0.71, and 1.81, respectively. Actual and predicted values in Tables 7.11–7.12 are in °C.

**Table 7.11:** Performance on out-of-sample data (first few days of July and August) using proposed model.

| Day | July | | August | |
|---|---|---|---|---|
| | Actual | Predicted | Actual | Predicted |
| 1 | 29.9 | 28.86 | 27.1 | 28.4 |
| 2 | 28.4 | 29.51 | 28.9 | 28.89 |
| 3 | 29.2 | 29.24 | 28.9 | 28.53 |
| 4 | 29.4 | 27.93 | 29.3 | 28.97 |
| 5 | 29.9 | 28.64 | 28.8 | 29.56 |
| 6 | 29.6 | 29.43 | 28.7 | 28.77 |
| 7 | 30.1 | 29.18 | 29 | 29.24 |
| 8 | 29.3 | 29.22 | 28.2 | 29.43 |
| 9 | 28.1 | 29.14 | 27 | 29.08 |
| 10 | 28.9 | 29.25 | 28.3 | 29.29 |

This work deals with univariate model. In each of the temperature dataset (Tables 7.2–7.5) the difference between the maximum value and minimum value is small, that is, data range is small. But for the applications where such difference is large, preprocessing of data is required before clustering. In this case, differences

**Table 7.12:** Performance on out-of-sample data (first few days of September) using proposed model.

| Day | September | |
|---|---|---|
| | Actual | Predicted |
| 1 | 27.5 | 27.71 |
| 2 | 26.8 | 28.41 |
| 3 | 26.4 | 28.14 |
| 4 | 27.5 | 27.78 |
| 5 | 26.6 | 27.94 |
| 6 | 28.2 | 28.36 |
| 7 | 29.2 | 27.56 |
| 8 | 29 | 28.56 |
| 9 | 30.3 | 28.61 |
| 10 | 29.9 | 28.59 |

between consecutive values are to be calculated before clustering. Differencing helps in reducing the number of clusters. Preprocessing of data is necessary if the time series contains patterns like trend, seasonal effect and so on.

Drawback of using BPTT is the memory requirement for the large sequence. Let us discuss the issues related to time complexity and space complexity of both of the models. For a BPNN model with $n$ input layer nodes, $h$ hidden layer nodes, and $m$ output layer nodes, there are $h(n + 1)$ and $m(h + 1)$ link weights in the first and second layer, respectively. For each epoch, both the space and time complexity is $O(h \cdot (m + n))$ [20].

Let us consider that the number of hidden layer nodes and input nodes are $n$ and $m$, respectively. Epoch length is represented by $h$. In case of fully connected network, number of adaptable weights = $n\ (n + m)$ and number of nonzero weights between units = $\mathrm{n}^2$. If the desired output values at every time period is considered, then epochwise BPTT has space complexity in $O(nh)$ and average time complexity (per time step) is $O(n^2)$ [15].

## 7.5 Conclusion

The proposed model can work with different number of clusters and orders. In addition to temperature data, this forecasting model is applicable to other datasets. In future, the technique can also be extended for forecasting multiple observations. Applications using variable/dynamic sequence length (prior time step) [21] instead of fixed sequence length can also be investigated in future. This approach is also applicable for solving complex and dynamic real-world problems like sequence classification in bioinformatics where large databases with multiple variables are required. Other signal analysis applications are in the field of seismology and speech processing.

## References

[1] Misra S., Sarkar S., & Mitra P. Statistical downscaling of precipitation using long short-term memory recurrent neural networks. Theoretical and applied climatology, 2018, 134(3–4), 1179–1196.

[2] Saha M., Mitra P., & Ravi N. Deep learning for predicting the monsoon over the homogeneous regions of India. Journal of earth system science, 2017, 126(54), 1–18.

[3] Zadeh L.A. Fuzzy set. Information and control, 1965, 8, 338–353.

[4] Song Q., & Chissom B.S. Fuzzy time series and its models. Fuzzy sets and systems, 1993a, 54, 269–277.

[5] Song Q., & Chissom B.S. Forecasting enrollments with fuzzy time series –Part I. Fuzzy sets and systems, 1993b, 54, 1–9.

[6] Song Q., & Chissom B.S. Forecasting enrollments with fuzzy time series –Part II. Fuzzy sets and systems, 1994, 64, 1–8.

[7] Chen S.M. Forecasting enrollments based on fuzzy time series. Fuzzy sets and systems, 1996, 81, 311–319.

[8] Lu W., Chen X., Pedrycz W., Liu X., & Yang J. Using interval information granules to improve forecasting in fuzzy time series. International journal of approximate reasoning, 2015, 57, 1–18.

[9] Chen M.Y. A high-order fuzzy time series forecasting model for internet stock trading. Future generation of computer system, 2014, 37, 461–467.

[10] Bose M., & Mali K. Fuzzy time series forecasting model using particle swarm optimization and neural network. Soft Computing for Problem Solving, Advances in Intelligent Systems and Computing, Springer, Singapore, 2019, 816.

[11] Singh P., & Borah B. High-order fuzzy-neuro expert system for daily temperature Forecasting. Knowledge-based systems, 2013, 46, 12–21.

[12] Huarng K., & Yu H.K. The application of neural networks to forecast fuzzy time series. Physica A, 2006, 363(2), 481–491.

[13] Aladag C.H. Using multiplicative neuron model to establish fuzzy logic relationships. Expert systems with applications, 2013, 40(3), 850–853.

[14] Aladag C.H., Basaran M.A., Egrioglu E., Yolcu U., & Uslu V.R. Forecasting in high order fuzzy time series by using neural networks to define fuzzy elations. Expert with applications, 2009, 36, 4228–4231.

[15] William R.J., & Zipser D. Gradient-based learning algorithms for recurrent networks and their computational complexity, Chauvin & Rumelhart, eds., Backpropagation: Theory, Architectures and Applications, New York, 1995, 433–486.

[16] Rumelhart D.E., & McClelland J.D. ed. Parallel Distributed Processing, MIT press, Cambridge, USA, 1986.

[17] Werbos P.J. Backpropagation through time: What it does and how to do it. Proceedings of the IEEE, 1990, 78(10), 1550–1560.
[18] Bezdek J.C. Pattern recognition with fuzzy objective function algorithms, Plenum Press, New York, USA, 1981.
[19] Theil H. Applied Economic Forecasting, Rand McNally, 1966.
[20] Alpaydin E. Introduction to Machine Learning, 2nd, The MIT Press, 2010.
[21] Grau, I., Nápoles, G., Bonet, I., & García, M. M. Backpropagation through Time Algorithm for Training Recurrent Neural Networks using Variable Length Instances. Computación y Sistemas, 2013, 17(1), 15–24.

# Index

https://doi.org/10.1515/9783110670905-008