Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

Department of Computer Science and Engineering (AIML)

(R18)

Deep Learning and Neural Networks

Lecture Notes

B. Tech VI YEAR – I SEM

Prepared by

Mrs.Swapna
(Professor&HOD-CSM)
Dept. CSE(AIML)

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

SYLLABUS

Faculty Name : Mrs Swapna

Subject Name :DL

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

NEURAL NETWORKS AND DEEP LEARNING

B.Tech. IV Year I Sem. **L T P C**
3 0 0 3

Course Objectives:

- To introduce the foundations of Artificial Neural Networks
- To acquire the knowledge on Deep Learning Concepts
- To learn various types of Artificial Neural Networks
- To gain knowledge to apply optimization strategies

Course Outcomes:

- Ability to understand the concepts of Neural Networks
- Ability to select the Learning Networks in modeling real world systems
- Ability to use an efficient algorithm for Deep Models
- Ability to apply optimization strategies for large scale applications

UNIT-I

Artificial Neural Networks Introduction, Basic models of ANN, important terminologies, Supervised Learning Networks, Perceptron Networks, Adaptive Linear Neuron, Back-propagation Network. Associative Memory Networks. Training Algorithms for pattern association, BAM and Hopfield Networks.

UNIT-II

Unsupervised Learning Network- Introduction, Fixed Weight Competitive Nets, Maxnet, Hamming Network, Kohonen Self-Organizing Feature Maps, Learning Vector Quantization, Counter Propagation Networks, Adaptive Resonance Theory Networks. Special Networks-Introduction to various networks.

UNIT - III

Introduction to Deep Learning, Historical Trends in Deep learning, Deep Feed - forward networks, Gradient-Based learning, Hidden Units, Architecture Design, Back-Propagation and Other Differentiation Algorithms

UNIT - IV

Regularization for Deep Learning: Parameter norm Penalties, Norm Penalties as Constrained Optimization, Regularization and Under-Constrained Problems, Dataset Augmentation, Noise Robustness, Semi-Supervised learning, Multi-task learning, Early Stopping, Parameter Typing and Parameter Sharing, Sparse Representations, Bagging and other Ensemble Methods, Dropout, Adversarial Training, Tangent Distance, tangent Prop and Manifold, Tangent Classifier

UNIT - V

Optimization for Train Deep Models: Challenges in Neural Network Optimization, Basic Algorithms, Parameter Initialization Strategies, Algorithms with Adaptive Learning Rates, Approximate Second- Order Methods, Optimization Strategies and Meta-Algorithms

Applications: Large-Scale Deep Learning, Computer Vision, Speech Recognition, Natural Language Processing

TEXT BOOKS:

1. Deep Learning: An MIT Press Book By Ian Goodfellow and Yoshua Bengio and Aaron Courville

Neural Networks and Learning Machines, Simon Haykin, 3rd Edition, Pearson Prentice

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

DEEP LEARNING

UNIT 1

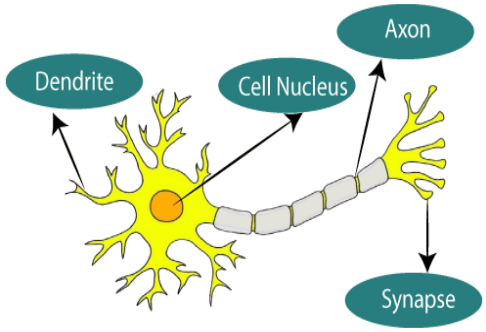
UNIT-I Artificial Neural Networks Introduction, Basic models of ANN, important terminologies, Supervised Learning Networks, Perceptron Networks, Adaptive Linear Neuron, Back-propagation Network. Associative Memory Networks. Training Algorithms for pattern association, BAM and Hopfield Networks.

ARTIFICIAL NEURAL NETWORK

The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modeled after the brain. An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.

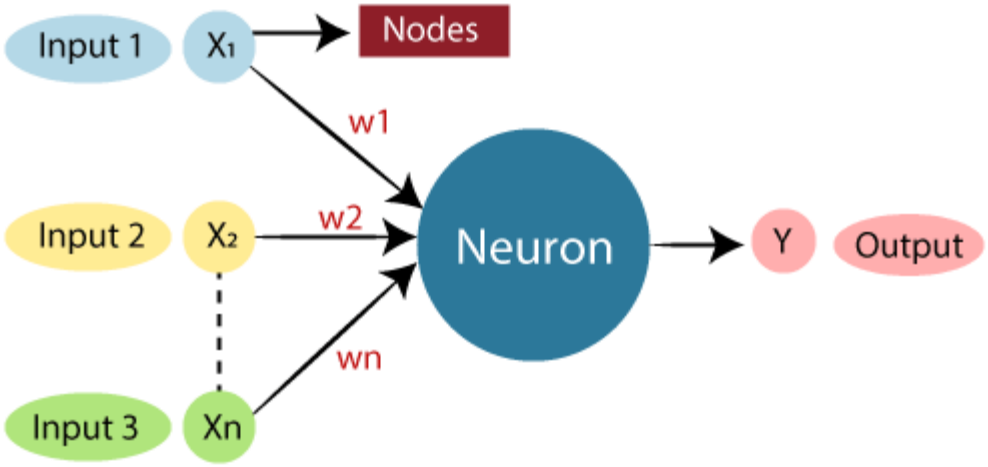
What is Artificial Neural Network

The term "**Artificial Neural Network**" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



The given figure illustrates the typical diagram of Biological Neural Network.

The typical Artificial Neural Network looks something like the given figure.



Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

Relationship between Biological neural network and artificial neural network:

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

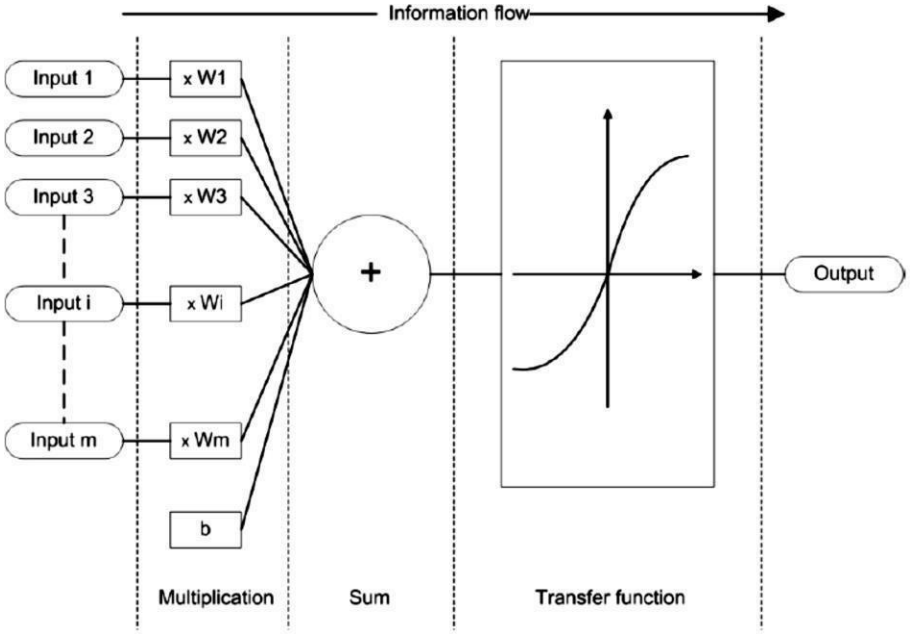
An **Artificial Neural Network** in the field of **Artificial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.

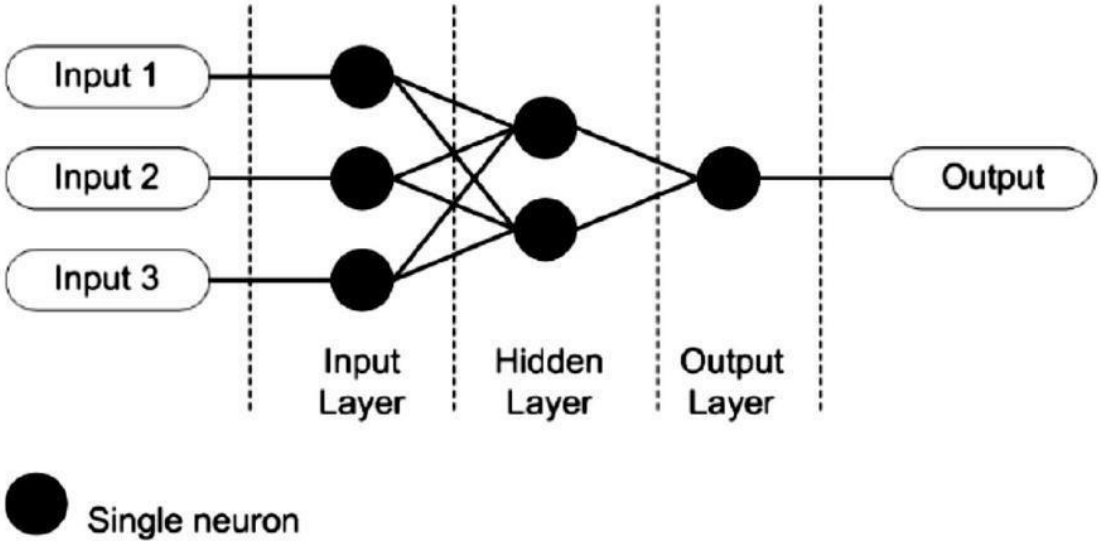
We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."

Architecture of Artificial neural Network

An Artificial Neural Network (ANN) is a mathematical model that tries to simulate the structure and functionalities of biological neural networks. Basic building block of every artificial neural network is artificial neuron, that is, a simple mathematical model (function). Such a model has three simple sets of rules: multiplication, summation and activation. At the entrance of artificial neuron the inputs are weighted what means that every input value is multiplied with individual weight. In the middle section of artificial neuron is sum function that sums all weighted inputs and bias. At the exit of artificial neuron the sum of previously weighted inputs and bias is passing through activation function that is



also called transfer function.



BIOLOGICAL NEURON STRUCTURE AND FUNCTIONS.

A neuron, or nerve cell, is an electrically excitable cell that communicates with other cells via specialized connections called synapses. It is the main component of nervous tissue. Neurons are typically classified into three types based on their function. Sensory neurons respond to stimuli such as touch, sound, or light that affect the cells of the sensory organs, and they send signals to the spinal cord or brain. Motor neurons receive signals from the brain and spinal cord to control everything from muscle contractions to glandular output. Interneurons connect neurons to other neurons within the same region of the brain or spinal cord.A group of connected neurons is called a neural circuit.

A typical neuron consists of a cell body (refered as soma), dendrites, and a single axon. The soma is usually compact. The axon and dendrites are filaments that extrude from it. Dendrites typically branch profusely and extend a few hundred micrometers from the soma. The axon leaves the soma at a swelling called the axon hillock, and travels for as far as 1 meter in humans or more in other species. It branches but usually maintains a constant diameter. At the farthest tip of the axon's branches are axon terminals, where the neuron can transmit a signal across the synapse to another cell. Neurons may lack dendrites or have no axon. The term neurite is used to describe either a dendrite or an axon, particularly when the cell is undifferentiated.

The soma is the body of the neuron. As it contains the nucleus, most protein synthesis occurs here. The

nucleus can range from 3 to 18 micrometers in diameter.

The dendrites of a neuron are cellular extensions with many branches. This overall shape and structure is referred to metaphorically as a dendritic tree. This is where the majority of input to the neuron occurs via the dendritic spine.

The axon is a finer, cable-like projection that can extend tens, hundreds, or even tens of thousands of times the diameter of the soma in length. The axon primarily carries nerve signals away from the soma, and carries some types of information back to it. Many neurons have only one axon, but this axon may—and usually will—undergo extensive branching, enabling communication with many target cells. The part of the axon where it emerges from the soma is called the axon hillock. Besides being an anatomical structure, the axon hillock also has the greatest density of voltage-dependent sodium channels. This makes it the most easily excited part of the neuron and the spike initiation zone for the axon. In electrophysiological terms, it has the most negative threshold potential.

While the axon and axon hillock are generally involved in information outflow, this region can also receive input from other neurons.

The axon terminal is found at the end of the axon farthest from the soma and contains synapses. Synaptic are specialized structures where neurotransmitter chemicals are released to communicate with target neurons. In addition to synaptic at the axon terminal, a neuron may have a synaptic bout, which are located along the length of the axon.

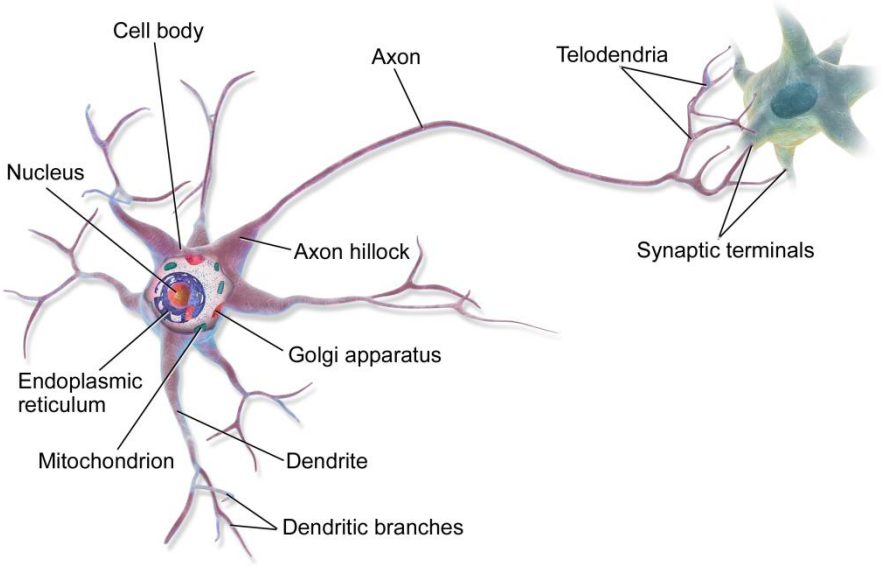
Most neurons receive signals via the dendrites and soma and send out signals down the axon. At the majority of synapses, signals cross from the axon of one neuron to a dendrite of another. However, synapses can connect an axon to another axon or a dendrite to another dendrite. The signaling process is partly electrical and partly chemical. Neurons are electrically excitable, due to maintenance of voltage gradients across their membranes. If the voltage changes by a large amount over a short interval, the neuron generates an all-or-nothing electrochemical pulse called an action potential.

This potential travels rapidly along the axon, and activates synaptic connections as it reaches them. Synaptic signals may be excitatory or inhibitory, increasing or reducing the net voltage that reaches the soma.

In most cases, neurons are generated by neural stem cells during brain development and childhood. Neurogenesis largely ceases during adulthood in most areas of the brain.

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001



A neuron consists of 3 main parts namely the dendrite, axon, and cell body. The dendrite is where a neuron receives input from other neurons. The axon is the output of a neuron it transmits the signal to other neurons. The cell body contains a nucleus and genetic material, which controls the cell's activities.

Neurons communicate with each other by sending signals, called neurotransmitters, across a narrow space, called a synapse, between the axons of the sender neuron and dendrites of the receiver neuron.

BRIEFLY EXPLAIN THE BASIC BUILDING BLOCKS OF ARTIFICIAL NEURAL NETWORKS.

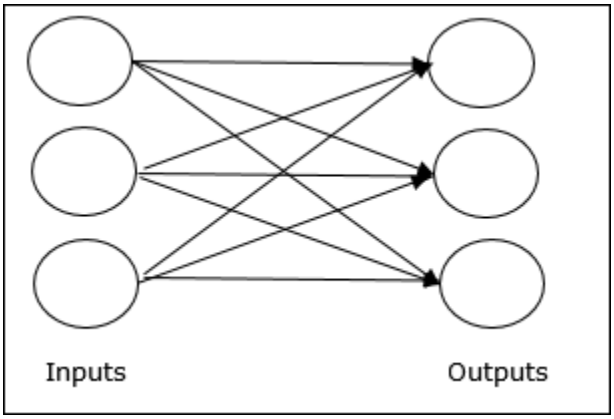
Processing of ANN depends upon the following three building blocks:

1. Network Topology
2. Adjustments of Weights or Learning
3. Activation Functions

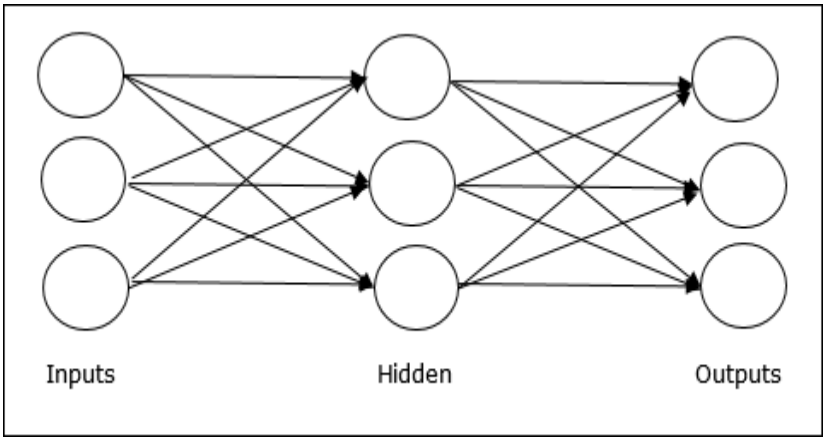
1. Network Topology: A network topology is the arrangement of a network along with its nodes and connecting lines. According to the topology, ANN can be classified as the following kinds:

A. Feed forward Network: It is a non-recurrent network having processing units/nodes in layers and all the nodes in a layer are connected with the nodes of the previous layers. The connection has different weights upon them. There is no feedback loop means the signal can only flow in one direction, from input to output. It may be divided into the following two types:

- **Single layer feed forward network:** The concept is of feed forward ANN having only one weighted layer. In other words, we can say the input layer is fully connected to the output layer.

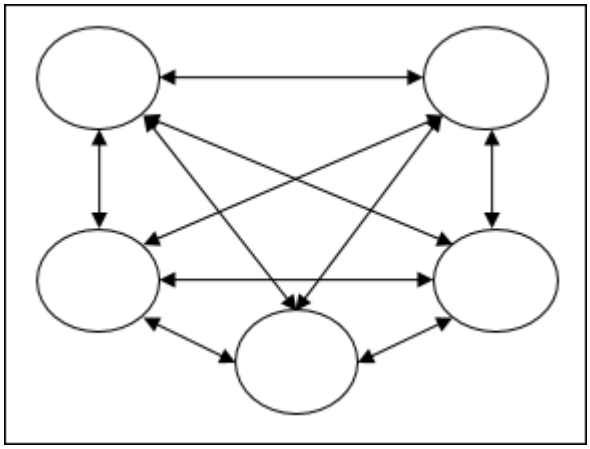


- **Multilayer feed forward network:** The concept is of feed forward ANN having more than one weighted layer. As this network has one or more layers between the input and the output layer, it is called hidden layers.

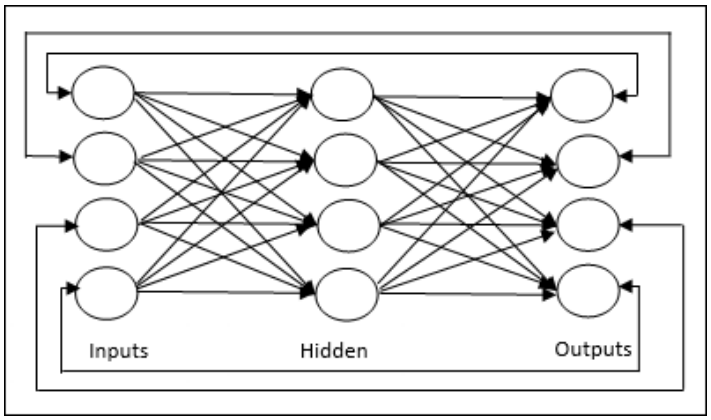


B. Feedback Network: As the name suggests, a feedback network has feedback paths, which means the signal can flow in both directions using loops. This makes it a non-linear dynamic system, which changes continuously until it reaches a state of equilibrium. It may be divided into the following types:

- **Recurrent networks:** They are feedback networks with closed loops. Following are the two types of recurrent networks.
- **Fully recurrent network:** It is the simplest neural network architecture because all nodes are connected to all other nodes and each node works as both input and output.



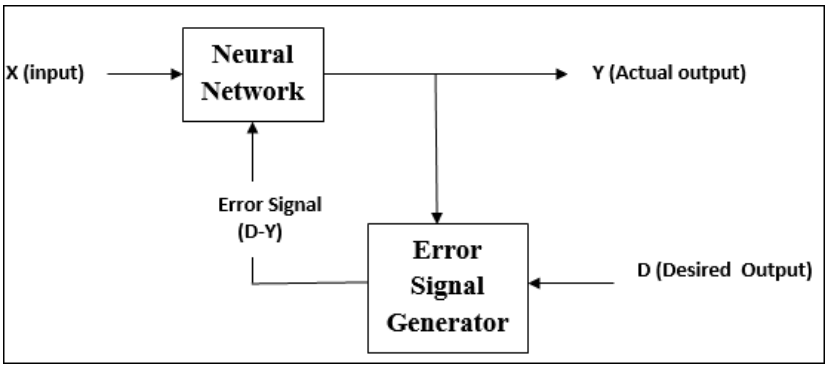
- **Jordan network** – It is a closed loop network in which the output will go to the input again as feedback as shown in the following diagram.



2. Adjustments of Weights or Learning: Learning, in artificial neural network, is the method of modifying the weights of connections between the neurons of a specified network. Learning in ANN can be classified into three categories namely supervised learning, unsupervised learning, and reinforcement learning.

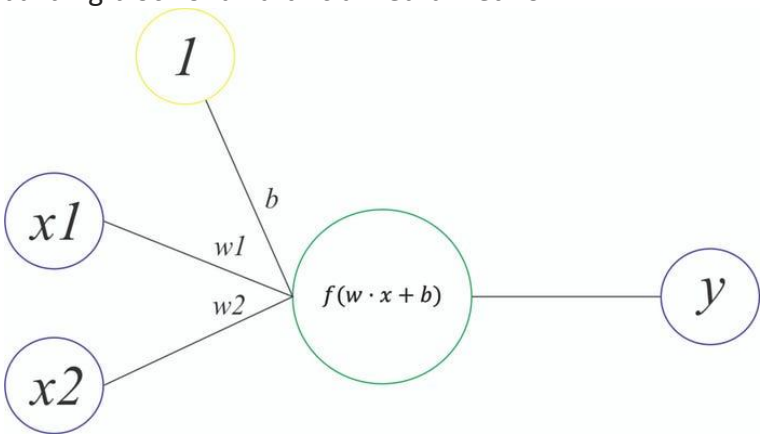
Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001



Artificial Neuron

The purpose of an artificial neural network is to mimic how the human brain works with the hope that we can build a machine that behaves like a human. An artificial neuron is the core building block of an artificial neural network.



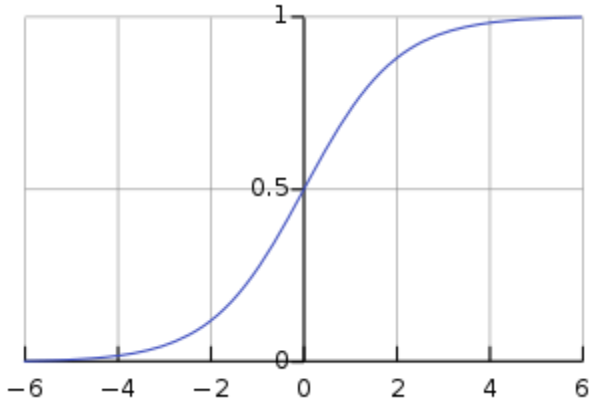
The structure of an artificial neuron is very similar to a biological neuron, it consists of 3 main parts, weight and bias as a dendrite denoted by w and b respectively, output as an axon denoted by y, and activation function as a cell body (nucleus) denoted by f(x). The x is the input signals received by the dendrite.

In artificial neurons, input and weight are represented as a vector while bias is represented as a scalar. Artificial neuron processes input signals by performing a dot product between the input vector and the weight vector, add the bias, then apply an activation function, and finally propagates the result to other neurons.

Activation Function

Activation functions are an essential part we should not underestimate, It's a function that an artificial neuron uses to get the output of a neuron, it is also known as Transfer Function. The

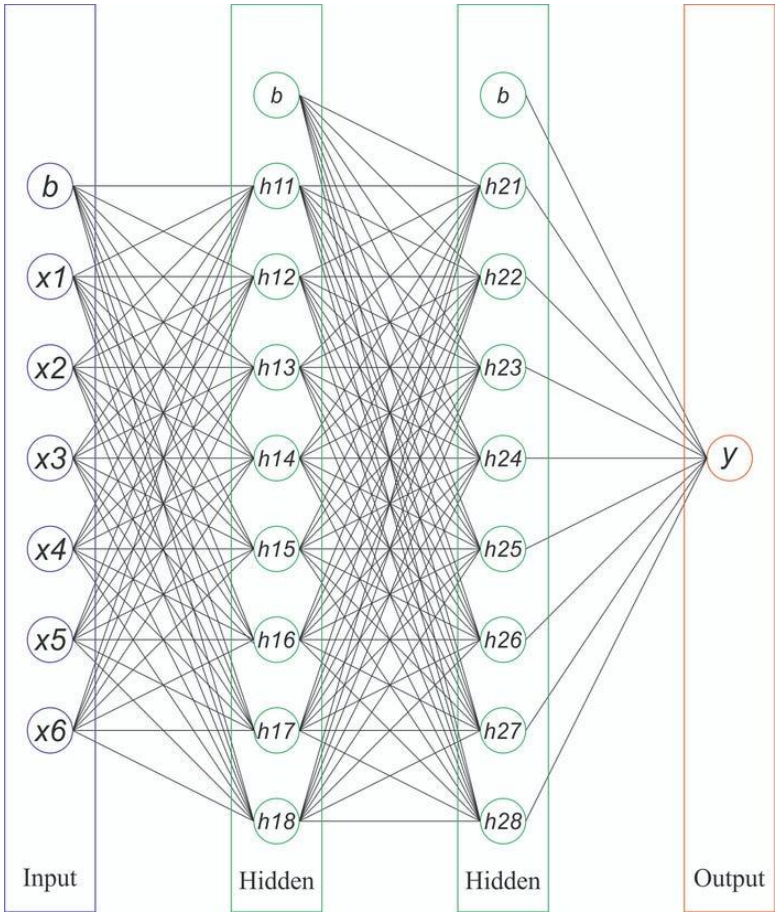
result of the dot product between weight and input plus bias is in the range of -inf and +inf, activation function aims to map the result into a certain range depending upon the function.



There are many activation functions out there, but the most important is the sigmoid activation function. It often used as activation in the output layer for binary classification tasks. Sigmoid bounds the result in the range of 0 until 1, it represents the probability of x whether x belongs to class 1 or 0. Sigmoid makes a decision by thresholding the result, if the result is ≥ 0.5 then x is classified as 1 otherwise, x is classified as 0.

Artificial Neural Network Structure

An artificial neural network is a bunch of artificial neurons interconnected to each other. Artificial neural networks (ANNs) learn to solve problems like a human brain. They process information by filtering it through densely connected artificial neurons. Each connection between neurons can transmit the signal to one another.



The neurons are structured into several sequential layers. Each neuron of a layer is connected to every neuron on the previous and the next layer. Every layer received input from the previous layer, processes it, and feeds it to the next layer. The first layer is called the input layer, it takes input and feeds them to the next layer. It doesn't have any weight, bias, and activation function. The last layer is called the output layer, it makes decisions about the data that is being fed. In between these two layers is called the hidden layer. This is where the computation takes place. We can stack hidden layers as many as we want to, but there will be a trade-off with the computation speed.

How Artificial Neural Network Works

An artificial neural network works by processing the input signals through the whole network and obtain the result on the output layer. This is also known as feedforward. The result then compared with the ground truth using a function. This function is called the loss function, it is a measurement that tells us how well our neural network models the data. The higher the loss, the worse the model.

How Artificial Neural Network Learns

An artificial neural network learns by nudging it's weight and bias so that it has a better prediction. The most popular algorithm is gradient descent, it is an iterative process that aims to minimize the loss function. In other words, it tries to find weights and biases in which if we use those weights and biases, the loss function will be minimum.

APPLICATIONS OF ANN

1. Data Mining: Discovery of meaningful patterns (knowledge) from large volumes of data.
2. Expert Systems: A computer program for decision making that simulates thought process of ahuman expert.
3. Fuzzy Logic: Theory of approximate reasoning.
4. Artificial Life: Evolutionary Computation, Swarm Intelligence.
5. Artificial Immune System: A computer program based on the biological immune system.
6. Medical: At the moment, the research is mostly on modelling parts of the human body and recognizing diseases from various scans (e.g. cardiograms, CAT scans, ultrasonic scans, etc.).Neural networks are ideal in recognizing diseases using scans since there is no need to provide a specific algorithm on how to identify the disease. Neural networks learn by example so the details of how to recognize the disease are not needed. What is needed is a set of examples that are representative of all the variations of the disease. The quantity of examples is not as important as the 'quantity'. The examples need to be selected very carefully if the system is to perform reliably and efficiently.

7. Computer Science: Researchers in quest of artificial intelligence have created spin offs like dynamic programming, object oriented programming, symbolic

programming, intelligent storage management systems and many more such tools. The primary goal of creating an artificial intelligence still remains a distant dream but people are getting an idea of the ultimate path, which could lead to it.

8. Aviation: Airlines use expert systems in planes to monitor atmospheric conditions and system status. The plane can be put on autopilot once a course is set for the destination.
9. Weather Forecast: Neural networks are used for predicting weather conditions. Previous data is fed to a neural network, which learns the pattern and uses that knowledge to predict weather patterns.
10. Neural Networks in business: Business is a diverted field with several general areas of specialization such as accounting or financial analysis. Almost any neural network application would fit into one business area or financial analysis.
11. There is some potential for using neural networks for business purposes, including resource allocation and scheduling.
12. There is also a strong potential for using neural networks for database mining, which is, searching for patterns implicit within the explicitly stored information in databases. Most of the funded work in this area is classified as proprietary. Thus, it is not possible to report on the full extent of the work going on. Most work is applying neural networks, such as the Hopfield-Tank network for optimization and scheduling.
13. Marketing: There is a marketing application which has been integrated with a neural network system. The Airline Marketing Tactician (a trademark abbreviated as AMT) is a computer system made of various intelligent technologies including expert systems. A feed forward neural network is integrated with the AMT and was trained using back-propagation to assist the marketing control of airline seat allocations. The adaptive neural approach was amenable to rule expression. Additionally, the application's environment changed rapidly and constantly, which required a continuously adaptive solution.
14. Credit Evaluation: The HNC company, founded by Robert Hecht-Nielsen, has developed several neural network applications. One of them is the Credit Scoring system which increases the profitability of the existing model up to 27%. The HNC neural systems were also applied to mortgage screening. A

neural network automated mortgage insurance under writing system was developed by the Nestor Company. This system was trained with 5048 applications of which 2597 were certified. The data related to property and borrower qualifications. In a conservative mode the system agreed on the under writers on 97% of the cases. In the liberal model the system agreed 84% of the cases. This is system run on an Apollo DN3000 and used 250K memory while processinga case file in approximately 1 sec.

ADVANTAGES OF ANN

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initialexperience.
2. Self-Organisation: An ANN can create its own organisation or representation of the information itreceives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Pattern recognition: is a powerful technique for harnessing the information in the data and generalizingabout it. Neural nets learn to recognize the patterns which exist in the data set.
5. The system is developed through learning rather than programming.. Neural nets teach themselves the patterns in the data freeing the analyst for more interesting work.
6. Neural networks are flexible in a changing environment. Although neural networks may take some time to learn a sudden drastic change they are excellent at adapting to constantly changing information.
7. Neural networks can build informative models whenever conventional approaches fail. Because neural networks can handle very complex interactions they can easily model data which is too difficult to model with traditional approaches such as inferential statistics or programming logic.
8. Performance of neural networks is at least as good as classical statistical modelling, and better on most problems. The neural networks build models that are more reflective of the structure of the data in significantly less time.

LIMITATIONS OF ANN

In this technological era everything has Merits and some Demerits in others words there is a Limitation with every system which makes this ANN technology weak in

some points. The various Limitations of ANN are:-

1) ANN is not a daily life general purpose problem solver.
2) There is no structured methodology available in ANN.
3) There is no single standardized paradigm for ANN development.
4) The Output Quality of an ANN may be unpredictable.
5) Many ANN Systems does not describe how they solve problems.
6) Greater computational burden.
7) Proneness to over fitting.
8) Empirical nature of model development.

ARTIFICIAL NEURAL NETWORK CONCEPTS/TERMINOLOGY

Here is a glossary of basic terms you should be familiar with before learning the details of neural networks.

Inputs: Source data fed into the neural network, with the goal of making a decision or prediction about the data. Inputs to a neural network are typically a set of real values; each value is fed into one of the neurons in the input layer.

Training Set: A set of inputs for which the correct outputs are known, used to train the neural network.

Outputs : Neural networks generate their predictions in the form of a set of real values or boolean decisions. Each output value is generated by one of the neurons in the output layer.

Neuron/perceptron: The basic unit of the neural network. Accepts an input and generates a prediction.

Each neuron accepts part of the input and passes it through the activation function. Common activation functions are sigmoid, TanH and ReLu. Activation functions help generate output values within an acceptable range, and their non-linear form is crucial for training the network.

Weight Space: Each neuron is given a numeric weight. The weights, together with the activation function, define each neuron's output. Neural networks are trained by fine-tuning weights, to discover the optimal set of weights that generates the most accurate prediction.

Forward Pass: The forward pass takes the inputs, passes them through the network and allows each neuron to react to a fraction of the input. Neurons generate their

outputs and pass them on to the next layer, until eventually the network generates an output.

Error Function: Defines how far the actual output of the current model is from the correct output. When training the model, the objective is to minimize the error function and bring output as close as possible to the correct value.

Backpropagation: In order to discover the optimal weights for the neurons, we perform a backward pass, moving back from the network's prediction to the neurons that generated that prediction. This is called backpropagation. Backpropagation tracks the derivatives of the activation functions in each successive neuron, to find weights that bring the loss function to a minimum, which will generate the best prediction. This is a mathematical process called *gradient descent*.

Bias and Variance: When training neural networks, like in other machine learning techniques, we try to balance between bias and variance. Bias measures how well the model fits the training set—able to correctly predict the known outputs of the training examples. Variance measures how well the model works with unknown inputs that were not available during training. Another meaning of bias is a " bias neuron" which is used in every layer of the neural network. The bias neuron holds the number 1, and makes it possible to move the activation function up, down, left and right on the number graph.

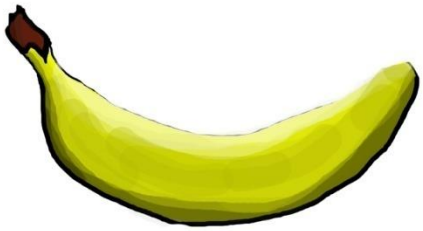
Hyperparameters: A hyper parameter is a setting that affects the structure or operation of the neural network. In real deep learning projects, tuning hyper parameters is the primary way to build a network that provides accurate predictions for a certain problem. Common hyper parameters include the number of hidden layers, the activation function, and how many times (epochs) training should be repeated.

Supervised learning: Supervised learning, as the name indicates, has the presence of a supervisor as a teacher. Basically supervised learning is when we teach or train the machine using data that is well-labelled. Which means some data is already tagged with the correct answer. After that, the machine is provided with a new set of examples(data) so that the supervised learning algorithm analyses the training data(set of training examples) and produces a correct outcome from labeled data.

For instance, suppose you are given a basket filled with different kinds of fruits. Now the first step is to train the machine with all the different fruits one by one like this:

- If the shape of the object is rounded and has a depression at the top, is red in color, then it will be labeled as –**Apple**.
- If the shape of the object is a long curving cylinder having Green-Yellow color, then it will be labeled as –**Banana**.

Now suppose after training the data, you have given a new separate fruit, say Banana from the basket, and asked to identify it.



Since the machine has already learned the things from previous data and this time has to use it wisely. It will first classify the fruit with its shape and color and would confirm the fruit name as BANANA and put it in the Banana category. Thus the machine learns the things from training data(basket containing fruits) and then applies the knowledge to test data(new fruit).

Supervised learning is classified into two categories of algorithms:

- **Classification**: A classification problem is when the output variable is a category, such as “Red” or “blue” , “disease” or “no disease”.
- **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Supervised learning deals with or learns with “labeled” data. This implies that some data is already tagged with the correct answer.

Types:-

- Regression
- Logistic Regression
- Classification
- Naive Bayes Classifiers

- K-NN (k nearest neighbors)
- Decision Trees
- Support Vector Machine

Advantages:-

- Supervised learning allows collecting data and produces data output from previous experiences.
- Helps to optimize performance criteria with the help of experience.
- Supervised machine learning helps to solve various types of real-world computation problems.
- It performs classification and regression tasks.
- It allows estimating or mapping the result to a new sample.
- We have complete control over choosing the number of classes we want in the training data.

Disadvantages:-

- Classifying big data can be challenging.
- Training for supervised learning needs a lot of computation time. So, it requires a lot of time.
- Supervised learning cannot handle all complex tasks in Machine Learning.
- Computation time is vast for supervised learning.
- It requires a labelled data set.
- It requires a training process.

Unsupervised learning

Unsupervised learning is the training of a machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of the machine is to group unsorted information according to similarities, patterns, and differences without any prior training of data.

Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore the machine is restricted to find the hidden structure in unlabeled data by itself.

For instance, suppose it is given an image having both dogs and cats which it has never seen.

Thus the machine has no idea about the features of dogs and cats so we can't categorize it as 'dogs and cats '. But it can categorize them according to their similarities, patterns, and differences, i.e., we can easily categorize the above picture into two parts. The first may contain all pics having **dogs** in them and the second part may contain all pics having **cats** in them. Here you didn't learn anything before, which means no training data or examples.
It allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with unlabelled data.

Unsupervised learning is classified into two categories of algorithms:

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Types of Unsupervised Learning:-

Clustering

1. Exclusive (partitioning)
2. Agglomerative
3. Overlapping
4. Probabilistic

Clustering Types:-

1. Hierarchical clustering
2. K-means clustering

Advantages of unsupervised learning:

- It does not require training data to be labeled.
- Dimensionality reduction can be easily accomplished using unsupervised learning.
- Capable of finding previously unknown patterns in data.
- **Flexibility**: Unsupervised learning is flexible in that it can be applied to a wide variety of problems, including clustering, anomaly detection, and association rule mining.
- **Exploration**: Unsupervised learning allows for the exploration of data and the discovery of novel and potentially useful patterns that may not be apparent from the outset.
- **Low cost**: Unsupervised learning is often less expensive than supervised learning because it doesn't require labeled data, which can be time-consuming and costly to obtain.

Disadvantages of unsupervised learning :

- Difficult to measure accuracy or effectiveness due to lack of predefined answers during training.
- The results often have lesser accuracy.
- The user needs to spend time interpreting and label the classes which follow that classification.

- **Lack of guidance**: Unsupervised learning lacks the guidance and feedback provided by labeled data, which can make it difficult to know whether the discovered patterns are relevant or useful.
- **Sensitivity to data quality**: Unsupervised learning can be sensitive to data quality, including missing values, outliers, and noisy data.
- **Scalability**: Unsupervised learning can be computationally expensive, particularly for large datasets or complex algorithms, which can limit its scalability.

Supervised vs. Unsupervised Machine Learning:

Parameters	**Supervised machine learning**	**Unsupervised machine learning**
Input Data	Algorithms are trained using labeled data.	Algorithms are used against data that is not labeled
Computational Complexity	Simpler method	Computationally complex
Accuracy	Highly accurate	Less accurate
No. of classes	No. of classes is known	No. of classes is not known
Data Analysis	Uses offline analysis	Uses real-time analysis of data
Algorithms used	Linear and Logistics regression, Random forest, Support Vector Machine, Neural Network, etc.	K-Means clustering, Hierarchical clustering, Apriori algorithm, etc.
Output	Desired output is given.	Desired output is not given.
Training data	Use training data to infer model.	No training data is used.
Complex model	It is not possible to learn larger and more complex models than with supervised learning.	It is possible to learn larger and more complex models with unsupervised learning.
Model	We can test our model.	We can not test our model.
Called as	Supervised learning is also called	Unsupervised learning is also called

	classification.	clustering.
Example	Example: Optical character recognition.	Example: Find a face in an image.

PERCEPTRON

- One type of ANN system is based on a unit called a perceptron. Perceptron is a single layer neural network.

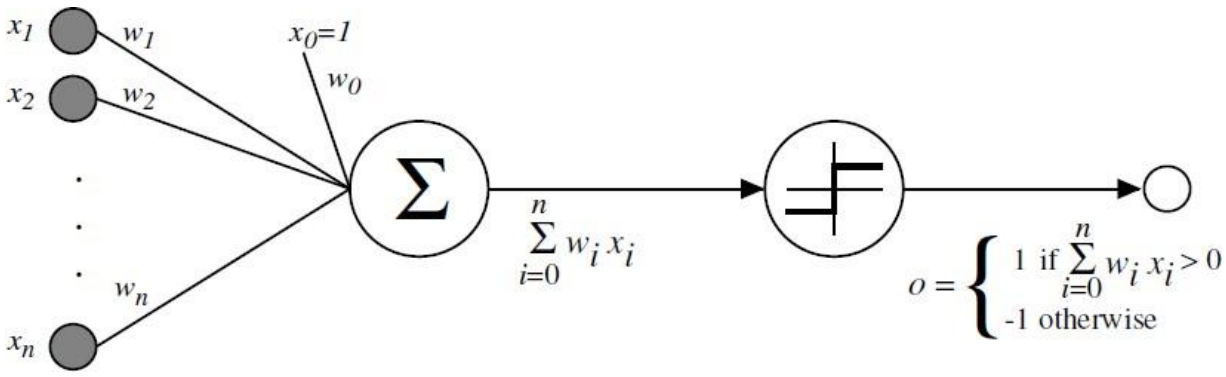


Figure: A perceptron

- A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.
- Given inputs ***x*** through ***x,*** the output ***O(x1,*** . . . , ***xn)*** computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

- Where, each w_i is a real-valued constant, or weight, that determines the contribution of input x_i to the perceptron output.
- $-w_0$ is a threshold that the weighted combination of inputs $w_1x_1 + \ldots + w_nx_n$ must surpass in order for the perceptron to output a 1.

Representational Power of Perceptrons

- The perceptron can be viewed as representing a hyperplane decision surface in the n- dimensional space of instances (i.e., points)
- The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in below figure

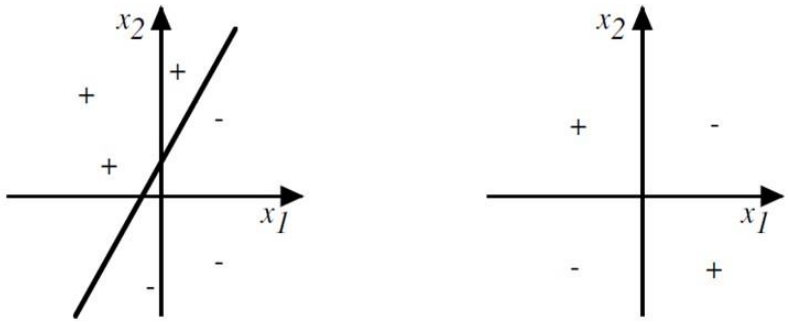
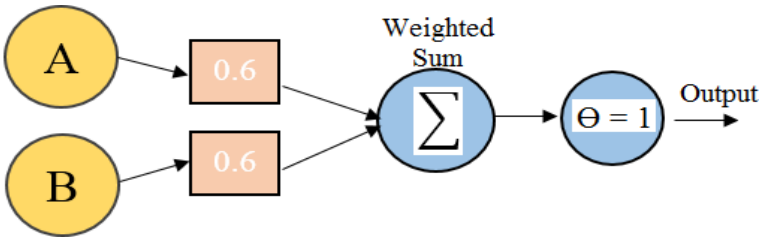


Figure : The decision surface represented by a two-input perceptron.
(a) A set of training examples and the decision surface of a perceptron that classifies them correctly. *(b)* A set of training examples that is not linearly separable.
x_1 and x_2 are the Perceptron inputs. Positive examples are indicated by "+", negative by "-".

Perceptrons can represent all of the primitive Boolean functions AND, OR, NAND (~ AND), and NOR (~OR)

Example: Representation of AND functions

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1



If A=0 & B=0 → 0*0.6 + 0*0.6 = 0.

This is not greater than the threshold of 1, so the output = 0.

If A=0 & B=1 → 0*0.6 + 1*0.6 = 0.6.

This is not greater than the threshold, so the output = 0.

If A=1 & B=0 → 1*0.6 + 0*0.6 = 0.6.

This is not greater than the threshold, so the output = 0.

If A=1 & B=1 → 1*0.6 + 1*0.6 = 1.2.

This exceeds the threshold, so the output = 1.

Drawback of perceptron

- The perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable

The Perceptron Training Rule

The learning problem is to determine a weight vector that causes the perceptron to produce the correct + 1 or - 1 output for each of the given training examples.

To learn an acceptable weight vector

- Begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
- This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to the perceptron training rule, which revises the weight w_i associated with input x_i according to the rule.

$$w_i \leftarrow w_i + \Delta w_i$$

Where,

$$\Delta w_i = \eta(t - o)x_i$$

Here,
t is the target output for the current training example
o is the output generated by the perceptron
η is a positive constant called the ***learning rate***

Drawback:

The perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable.

Problem 1

Truth Table of OR Logical GATE is,

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

Weights w1 = 0.6, w2 = 0.6, Threshold = 1 and Learning Rate n = 0.5 are given

For Training Instance 1: A=0, B=0 and Target = 0

wi.xi = 0*0.6 + 0*0.6 = 0

This is not greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 2: A=0, B=1 and Target = 1

wi.xi = 0*0.6 + 1*0.6 = 0.6

This is not greater than the threshold of 1, so the output = 0. Here the target does not match with calculated output.

$$wi = wi + n(t - o)xi$$

$$w1 = 0.6 + 0.5(1 - 0)0 = 0.6$$

$$w2 = 0.6 + 0.5(1 - 0)1 = 1.1$$

Now,

Weights w1 = 0.6, w2 = 1.1, Threshold = 1 and Learning Rate n = 0.5 are given

For Training Instance 1: A=0, B=0 and Target = 0

wi.xi = 0*0.6 + 0*1.1 = 0

This is not greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 2: A=0, B=1 and Target = 1

wi.xi = 0*0.6 + 1*1.1 = 1.1

This is greater than the threshold of 1, so the output = 1. Here the target is same as calculated output.

For Training Instance 3: A=1, B=0 and Target = 1

wi.xi = 1*0.6 + 0*1.1 = 0.6

This is not greater than the threshold of 1, so the output = 0. Here the target does not match with calculated output.

$$wi = wi + n(t - o)xi$$

$$w1 = 0.6 + 0.5(1 - 0)1 = 1.1$$

$$w2 = 1.1 + 0.5(1 - 0)0 = 1.1$$

Now,

Weights w1 = 1.1, w2 = 1.1, Threshold = 1 and Learning Rate n = 0.5 are given

For Training Instance 1: A=0, B=0 and Target = 0

wi.xi = 0*2.2 + 0*1.1 = 0

This is not greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 2: A=0, B=1 and Target = 1

wi.xi = 0*1.1 + 1*1.1 = 1.1

This is greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 3: A=1, B=0 and Target = 1

wi.xi = 1*1.1 + 0*1.1 = 1.1

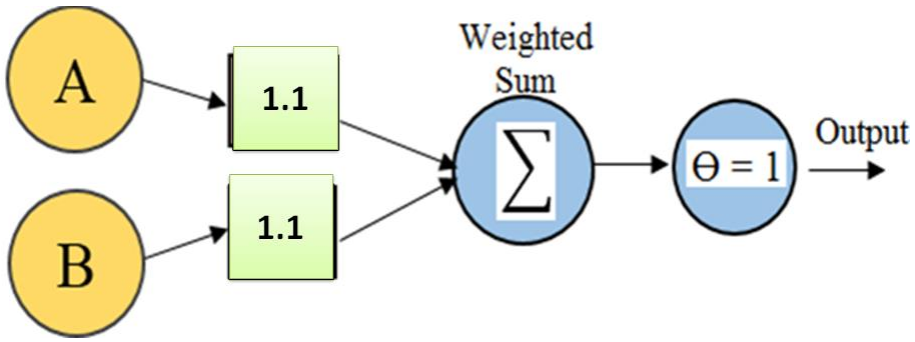
This is greater than the threshold of 1, so the output = 1. Here the target is same as calculated output.

For Training Instance 4: A=1, B=1 and Target = 1

wi.xi = 1*1.1 + 1*1.1 = 2.2

This is greater than the threshold of 1, so the output = 1. Here the target is same as calculated output.

Final wieghts w1 = 1.1, w2 = 1.1 Threshold = 1 and Learning Rate n = 0.5.



BRIEFLY EXPLAIN THE ADALINE MODEL OF ANN.

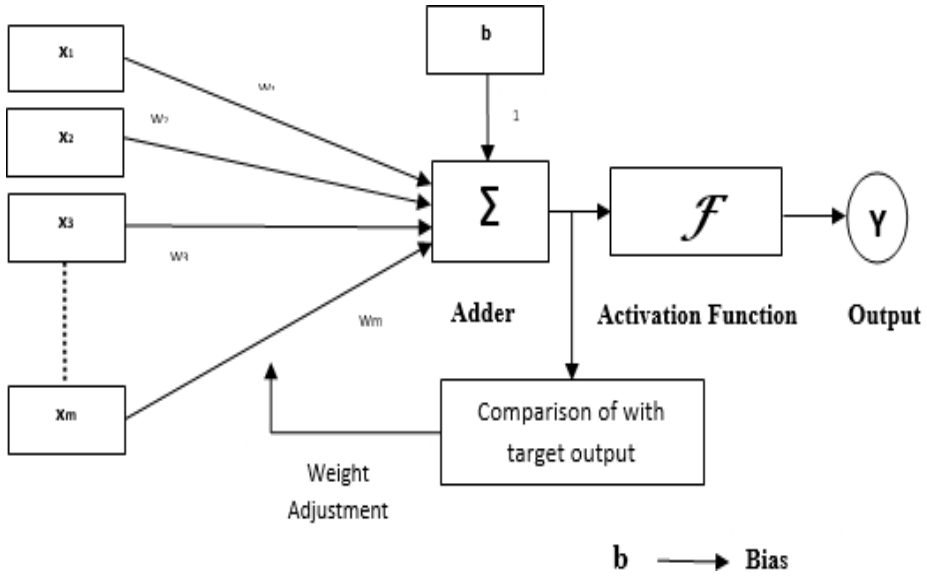
ADALINE (Adaptive Linear Neuron or later Adaptive Linear Element) is an early single-layer artificial neural network and the name of the physical device that implemented this network. The network uses memistors. It was developed by Professor Bernard Widrow and his graduate student Ted Hoff at Stanford University in 1960. It is based

on the McCulloch–Pitts neuron. It consists of a weight, a bias and a summation function. The difference between Adaline and the standard (McCulloch–Pitts) perceptron is that in the learning phase, the weights are adjusted according to the weighted sum of the inputs (the net). In the standard perceptron, the net is passed to the activation (transfer) function and the function's output is used for adjusting the weights. Some important points about Adaline are as follows:

- It uses bipolar activation function.
- It uses delta rule for training to minimize the Mean-Squared Error (MSE) between the actual outputand the desired/target output.
- The weights and the bias are adjustable.

Architecture of ADALINE network: The basic structure of Adaline is similar to perceptron having an extrafeedback loop with the help of which the actual output is compared with the desired/target output. After comparison on the basis of training algorithm, the weights and bias will be updated.

Architecture of ADALINE: The basic structure of Adaline is similar to perceptron having an extra feedback loop with the help of which the actual output is compared



with the desired/target output. After comparison on the basis of training algorithm, the weights and bias will be updated.

Training Algorithm of ADALINE:

Step 1 − Initialize the following to start the training:

- Weights
- Bias
- Learning rate α

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping

condition is not true.Step 3 − Continue step 4-6

for every bipolar training pair s : t.

Step 4 − Activate each input unit as follows:

$$x_i = s_i (i=1 \text{ to } n)$$

Step 5 − Obtain the net input with the following relation:

$$y_{in} = b + \sum_{i}^{n} x_i \, w_i$$

Here 'b' is bias and 'n' is the total number of input neurons.

Step 6 − Apply the following activation function to obtain the final output:

$$(f\ y)_n = \begin{cases} 1, \text{ if } y_{in} \geq 0 \\ -1, \text{ if } y_{in} < 0 \end{cases}$$

Step 7 − Adjust the weight and bias as follows :

Case 1 − if y ≠ t then, $w_i(new) = w_i(old) + \alpha(t - y_{in})x_i$

$$b(new) = b(old) + \alpha(t - y_{in})$$

Case 2 − if y = t then, $w_i(new) = w_i(old)$

b(new) = b(old)

Here 'y' is the actual output and 't' is the desired/target output. $(t-y_{in})$ is the computed error.

Step 8 − Test for the stopping condition, which will happen when there is no change in weight or thehighest weight change occurred during training is smaller than the specified tolerance.

EXPLAIN MULTIPLE ADAPTIVE LINEAR NEURONS (MADALINE).

Madaline which stands for Multiple Adaptive Linear Neuron, is a network which consists of many Adalines in parallel. It will have a single output unit.. training algorithm is based on a principle called "minimal disturbance". It proceeds by looping over training examples, then for each example, it:

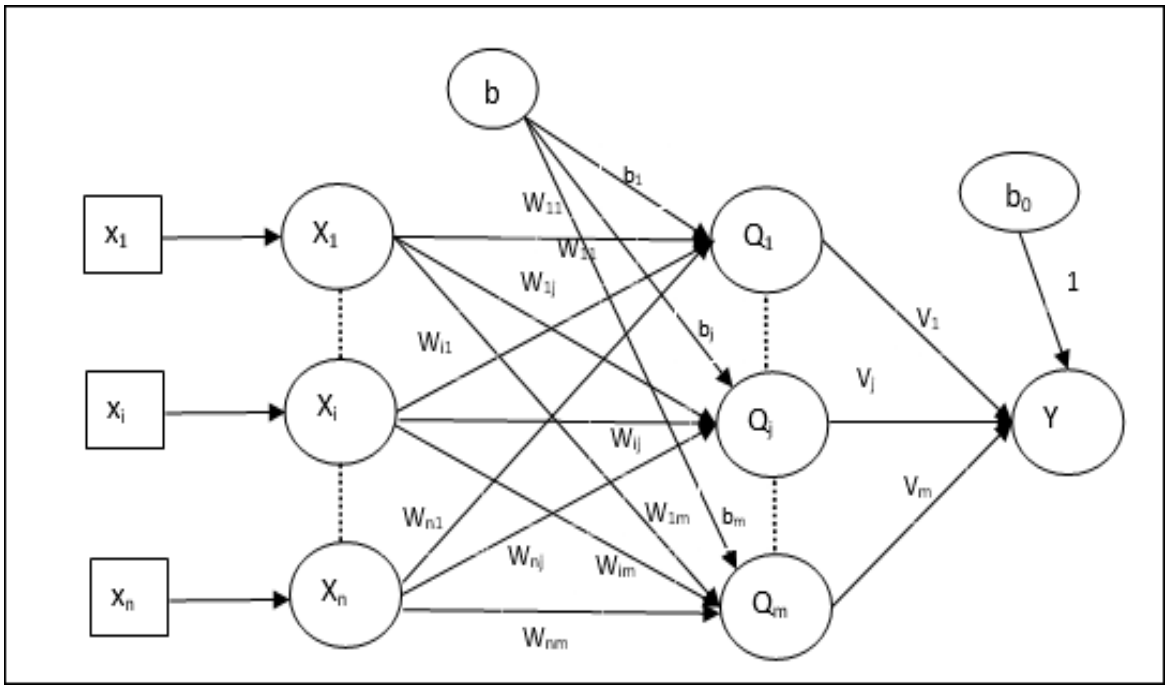
- finds the hidden layer unit (ADALINE classifier) with the lowest confidence in its prediction,tentatively flips the sign of the unit,
- accepts or rejects the change based on whether the network's error is reduced,
- stops when the error is zero.

Some important points about Madaline are as follows:

- It is just like a multilayer perceptron, where Adaline will act as a hidden unit between the input andthe Madaline layer.
- The weights and the bias between the input and Adaline layers, as in we see in the Adalinearchitecture, are adjustable.
- The Adaline and Madaline layers have fixed weights and bias of 1.
- Training can be done with the help of Delta rule.

BRIEFLY EXPLAIN THE ARCHITECTURE OF MADALINE

MADALINE (Many ADALINE) is a three-layer (input, hidden, output), fully connected, feed-forward artificial neural network architecture for classification that uses ADALINE units in its hidden and output layers, i.e. its activation function is the sign function. The three-layer network uses memistors. The architecture of Madaline consists of "n" neurons of the input layer, "m" neurons of the Adaline layer, and 1 neuron of the Madaline layer. The Adaline layer can be considered as the hidden layer as it is between the input layer and the output layer, i.e. the Madaline layer.



Training Algorithm of MADALINE

By now we know that only the weights and bias between the input and the Adaline layer are to beadjusted, and the weights and bias between the Adaline and the Madaline layer are fixed.

Step 1 − Initialize the following to start the training:

- Weights
- Bias
- Learning rate α

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.Step 3 − Continue step 4-6 for every bipolar training pair s:t.

Step 4 − Activate each input unit as follows:

$$x_i = s_i (I = 1 \text{ to } n)$$

Step 5 − Obtain the net input at each hidden layer, i.e. the Adaline layer with the following relation:

$$n$$

$$Oinj = bj + \sum_{i} xi \ \ wij \ j = 1 \ to \ m$$

Here 'b' is bias and 'n' is the total number of input neurons.

Step 6 − Apply the following activation function to obtain the final output at the Adaline and the Madaline Layer:

$(yi) =$ {
1, if x ≥ 0
−1, if x < 0}

Output at the hidden Adaline unit Qj=f(Qinj)

Final output of the network y = f(yin)

i.e.

$$yinj = b0 + \sum_{j=1}^{m} Qj \ vj$$

Step 7 − Calculate the error and adjust the

weights as follows –Case 1 − if y ≠ t

and t = 1 then,

wij(new) = wij(old)+α(1−Qinj)xi
bj(new) = bj(old)+α(1−Qinj)

In this case, the weights would be updated on Qj where the net input is close to 0 because t = 1.

Case 2 − if y ≠ t and t = -1 then,

wik(new) = wik(old)+α(−1 −Qink)xi
bk(new) = bk(old)+α(−1− Qink)

In this case, the weights would be updated on Qk where the net input is positive because t = -1.Here 'y' is the actual output and 't' is the desired/target output.

Case 3 − if y = t, then there would be no change in weights.

Step 8 − Test for the stopping condition, which will happen when there is no change in weight or thehighest weight change occurred during training is smaller than the specified tolerance.

Backpropagation:

Backpropagation is a widely used algorithm for training feedforward neural networks. It computes the gradient of the loss function with respect to the network weights. It is very efficient, rather than naively directly computing the gradient concerning each weight. This efficiency makes it possible to use gradient methods to train multi-layer networks and update weights to minimize loss; variants such as gradient descent or stochastic gradient descent are often used.

The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight via the chain rule, computing the gradient layer by layer, and iterating backward from the last layer to avoid redundant computation of intermediate terms in the chain rule.

Working of Backpropagation:

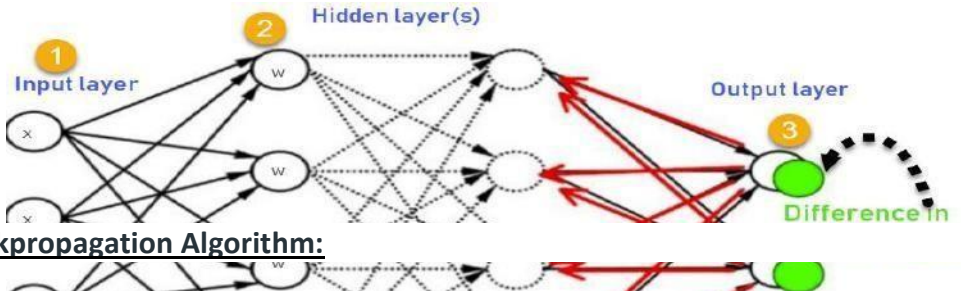
Neural networks use supervised learning to generate output vectors from input vectors that the network operates on. It Compares generated output to the desired output and generates an error report if the result does not match the generated output vector. Then it adjusts the weights according to the bug report to get your desired output.

HOW BACKPROPAGATION WORKS?

1. **Forward pass**—weights are initialized and inputs from the training set are fed into thenetwork. The forward pass is carried out and the model generates its initial prediction.
2. **Error function**—the error function is computed by checking how far away the prediction isfrom the known true value.
3. **Backpropagation with gradient descent**—the backpropagation algorithm calculates how much the output values are affected by each of the weights in the model. To do this, it calculates partial derivatives, going back from the error function to a specific neuron and its weight. This provides complete traceability from total errors, back to a specific weight whichcontributed to that error. The result of backpropagation is a set of weights that minimize theerror function.
4. **Weight update**—weights can be updated after every sample in the training set, but this is usually not practical. Typically, a batch of samples is run in one

big forward pass, and then backpropagation performed on the aggregate

5. result. The *batch size* and number of batchesused in training, called *iterations*, are important hyperparameters that are tuned to get thebest results. Running the entire training set through the backpropagation process is called an *epoch*.



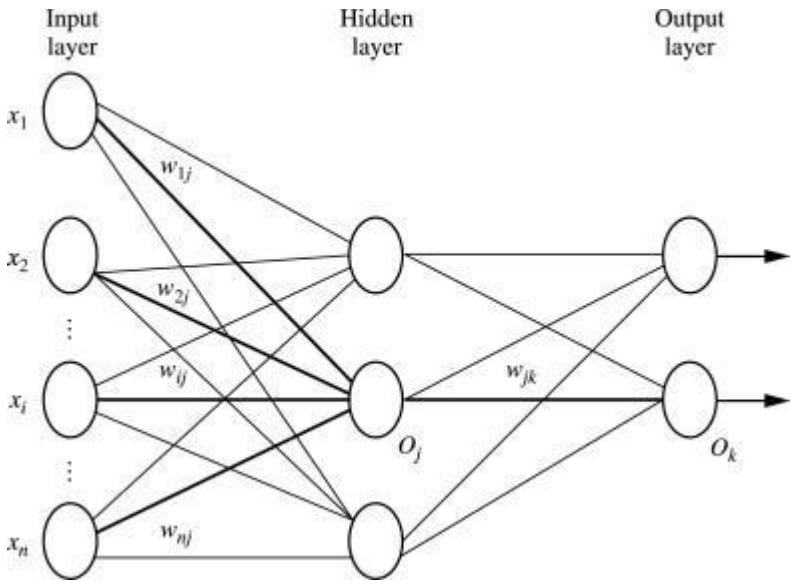
Backpropagation Algorithm:

Step 1: Inputs X, arrive through the preconnected path.
Step 2: The input is modeled using true weights W. Weights are usually chosen randomly.
Step 3: Calculate the output of each neuron from the input layer to the hidden layer to the output layer.

Step 4: Calculate the error in the outputs

Backpropagation Error= Actual Output – Desired Output

Step 5: From the output layer, go back to the hidden layer to adjust the weights to reduce the error.
Step 6: Repeat the process until the desired output is achieved.



Parameters :

- x = inputs training vector $x=(x_1,x_2,\ldots\ldots\ldots x_n)$.
- t = target vector $t=(t_1,t_2\ldots\ldots\ldots t_{n)}$.
- δ_k = error at output unit.
- δ_j = error at hidden layer.
- α = learning rate.
- V_{0j} = bias of hidden unit j.

Need for Backpropagation:

Backpropagation is "backpropagation of errors" and is very useful for training neural networks. It's fast, easy to implement, and simple. Backpropagation does not require any parameters to be set, except the number of inputs. Backpropagation is a flexible method because no prior knowledge of the network is required.

Advantages:

- It is simple, fast, and easy to program.
- Only numbers of the input are tuned, not any other parameter.
- It is Flexible and efficient.
- No need for users to learn any special functions.

Disadvantages:

- It is sensitive to noisy data and irregularities. Noisy data can lead to inaccurate results.
- Performance is highly dependent on input data.
- Spending too much time training.

Associative memory is also known as content addressable memory (CAM) or associative storage or associative array. It is a special type of memory that is optimized for performing searches through data, as opposed to providing a simple direct access to the data based on the address.
it can store the set of patterns as memories when the associative memory is being presented with a key pattern, it responds by producing one of the stored pattern which closely resembles or relates to the key pattern.

it can be viewed as **data correlation** here.
input data is correlated with that of stored data in the CAM.
it forms of two type:

1. auto associative memory network : An auto-associative memory network, also known as a recurrent neural network, is a type of associative memory that is used to recall a

pattern from partial or degraded inputs. In an auto-associative network, the output of the network is fed back into the input, allowing the network to learn and remember the patterns it has been trained on. This type of memory network is commonly used in applications such as speech and image recognition, where the input data may be incomplete or noisy.

2. hetero associative memory network : A hetero-associative memory network is a type of associative memory that is used to associate one set of patterns with another. In a hetero-associative network, the input pattern is associated with a different output pattern, allowing the network to learn and remember the associations between the two sets of patterns. This type of memory network is commonly used in applications such as data compression and data retrieval.

How Does Associative Memory Work?

In conventional memory, data is stored in specific locations, called addresses, and retrieved by referencing those addresses. In associative memory, data is stored together with additional tags or metadata that describe its content. When a search is performed, the associative memory compares the search query with the tags of all stored data, and retrieves the data that matches the query.

Associative memory is designed to quickly find matching data, even when the search query is incomplete or imprecise. This is achieved by using parallel processing techniques, where multiple search queries can be performed simultaneously. The search is also performed in a single step, as opposed to conventional memory where multiple steps are required to locate the data.

Applications of Associative memory :-

1. It can be only used in memory allocation format.
2. It is widely used in the database management systems, etc.
3. Networking: Associative memory is used in network routing tables to quickly find the path to a destination network based on its address.
4. Image processing: Associative memory is used in image processing applications to search for specific features or patterns within an image.
5. Artificial intelligence: Associative memory is used in artificial intelligence applications such as expert systems and pattern recognition.
6. Database management: Associative memory can be used in database management systems to quickly retrieve data based on its content.

Advantages of Associative memory :-

1. It is used where search time needs to be less or short.
2. It is suitable for parallel searches.
3. It is often used to speedup databases.
4. It is used in page tables used by the virtual memory and used in neural networks.

Disadvantages of Associative memory :-

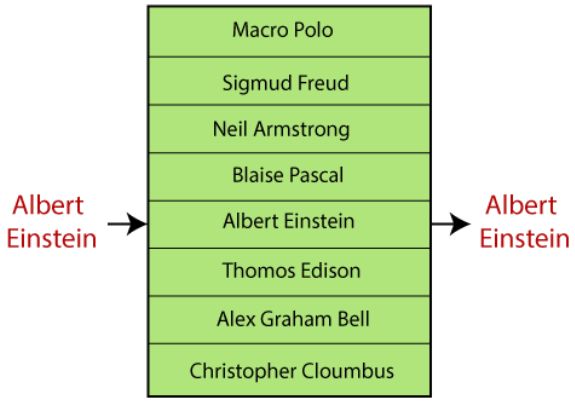
1. It is more expensive than RAM.

2. Each cell must have storage capability and logical circuits for matching its content with external argument.

Associate Memory Network EXAMPLES

An associate memory network refers to a content addressable memory structure that associates a relationship between the set of input patterns and output patterns. A content addressable memory structure is a kind of memory structure that enables the recollection of data based on the intensity of similarity between the input pattern and the patterns stored in the memory.

Understand this concept with an example:



Content-addressable memory, Input and Output

The figure given below illustrates a memory containing the names of various people. If the given memory is content addressable, the incorrect string "**Albert Einstein**" as a key is sufficient to recover the correct name "**Albert Einstein**."

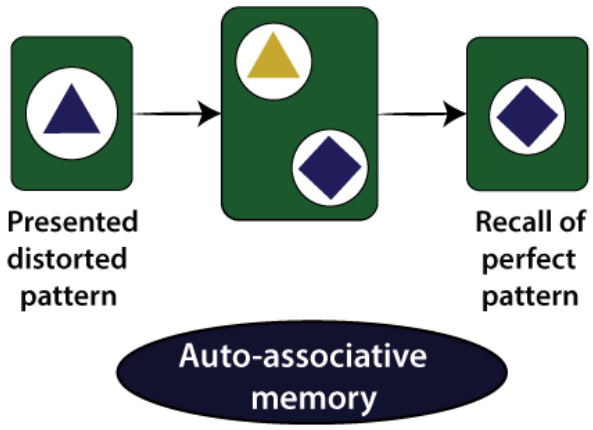
In this condition, this type of memory is robust and fault-tolerant because of this type of memory model, and some form of error-correction capability.

Note: An associate memory is obtained by its content, adjacent to an explicit address in the traditional computer memory system. The memory enables the recollection of information based on incomplete knowledge of its contents.

There are two types of associate memory- an auto-associative memory and hetero associative memory.

Auto-associative memory:

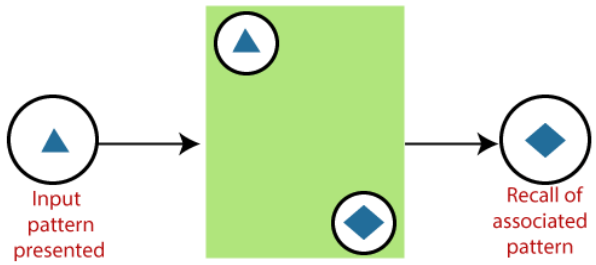
An auto-associative memory recovers a previously stored pattern that most closely relates to the current pattern. It is also known as an **auto-associative correlator**.



Consider **x[1], x[2], x[3],….. x[M]**, be the number of stored pattern vectors, and let **x[m]** be the element of these vectors, showing characteristics obtained from the patterns. The auto-associative memory will result in a pattern vector **x[m]** when putting a noisy or incomplete version of x[m].

Hetero-associative memory:

In a hetero-associate memory, the recovered pattern is generally different from the input pattern not only in type and format but also in content. It is also known as a **hetero-associative** correlator.



Hetero-associative memory

Consider we have a number of key response pairs **{a(1), x(1)}, {a(2),x(2)},.....,{a(M), x(M)}**. The hetero-associative memory will give a pattern vector **x(m)** when a noisy or incomplete version of the **a(m)** is given.

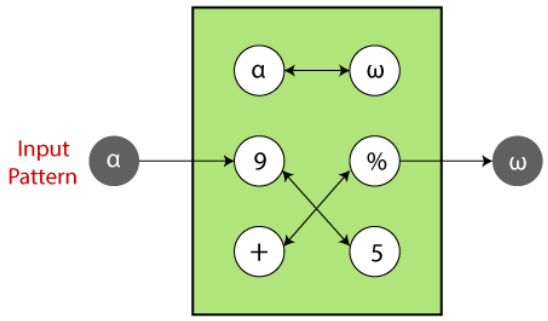
Neural networks are usually used to implement these associative memory models called **neural associative memory (NAM)**.

The linear associate is the easiest artificial neural associative memory.

These models follow distinct neural network architecture to memorize data.

Working of Associative Memory:

Associative memory is a depository of associated pattern which in some form. If the depository is triggered with a pattern, the associated pattern pair appear at the output. The input could be an exact or partial representation of a stored pattern.



Working of an associated memory

If the memory is produced with an input pattern, may say **α,** the associated pattern **ω** is recovered automatically.

Training Algorithms for pattern association

Learning rule is a method or a mathematical logic. It helps a Neural Network to learn from the existing conditions and improve its performance. Thus learning rules updates the weights and bias levels of a network when a network simulates in a specific data environment. Applying learning rule is an iterative process. It helps a neural network to learn from the existing conditions and improve its performance.

The different learning rules in the Neural network are:

Hebbian learning rule – It identifies, how to modify the weights of nodes of a network.

1. **Hebbian Learning Rule:** The Hebbian rule was the first learning rule. In 1949 Donald Hebb developed it aslearning algorithm of the unsupervised neural network. We can use it to identify how to improve the weights of nodes of a network. The Hebb learning rule assumes that – If two neighbor neurons activated and deactivated at the same time, then the weight connecting these neurons should increase. At the start, values of all weights are set to zero. This learning rule can be used for both soft- and hard-activation functions. Since desired responses of neurons are not used in the learning procedure, this is the unsupervised learning rule. The absolute values of the weights are usually proportional to the learning time, which is undesired.

$$W_{ij} = x_i * x_j$$

Mathematical Formula of Hebb Learning Rule.

Hebb Rule for Pattern Association:-The Hebb rule is the simplest and most common method of determining the weights for an associative memory neural net.-
we denote our training vector pairs (input training-target output vectors) as
s:t.

We then denote our testing input vector as x,which may or may not be the same as one of the training input vectors.
-In the training algorithm of hebb rule the weights initially adjusted to 0, then updated using the following formula:

In the training algorithm of hebb rule the weights initially adjusted to 0, then updated using the following

formula: wij(new) =wij(old)+ xiyj ; (i = 1, . . . ,n; j= 1, . . . , m): where, xi = si yj =t

2. Delta Learning Rule: Developed by Widrow and Hoff, the delta rule, is one of the most common learning rules. It depends on supervised learning. This rule states that the modification in sympatric weight of a node is equal to the multiplication of error and the input. In Mathematical form the delta rule is as follows:

$$\Delta w = \eta (t - y) x_i$$

Mathematical Formula of Delta Learning Rule

For a given input vector, compare the output vector is the correct answer. If the difference is zero, no learning takes place; otherwise, adjusts its weights to reduce this difference.

The change in weight from ui to uj is: dwij = r* ai * ej. where r is the learning rate, ai represents the activation of ui and ej is the differencebetween the expected output and the actual output of uj. If the set of input patterns form an independent set then learn arbitrary associations using the delta rule.

It has seen that for networks with linear activation functions and with no hidden units. The error squared vs. the weight graph is a paraboloid in n-space. Since the proportionality constant is negative, the graph of such a function is concave upward and has the least value. The vertex of this paraboloid represents the point where it reduces the error. The weight vector corresponding to this pointis then the ideal weight vector. We can use the delta learning rule with both single output unit and several output units. While applying the delta rule assume that the error can be directly measured. The aim of applying the delta rule is to reduce the difference between the actual and expected output that is the error.

BIDIRECTIONAL ASSOCIATIVE MEMORY

Bidirectional Associative Memory (BAM) is a recurrent neural network (RNN) of a special type, initially proposed by Bart Kosko, in the early 1980s,

Recurrent Neural Network (RNN)

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its **Hidden state**, which remembers some information about a sequence. The state is also referred to as *Memory State* since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output.

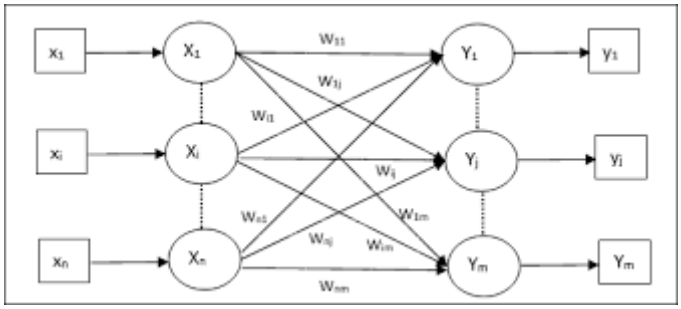
Bidirectional Associative Memory (BAM) is a supervised learning model in Artificial Neural Network. This is ***hetero-associative memory***, for an input pattern, it returns another pattern which is potentially of a different size. This phenomenon is very similar to the human brain. Human memory is necessarily associative. It uses a chain of mental associations to recover a lost memory like associations of faces with names, in exam questions with answers, etc.
In such memory associations for one type of object with another, a Recurrent Neural Network (RNN) is needed to receive a pattern of one set of neurons as an input and generate a related, but different, output pattern of another set of neurons.

Why BAM is required?

The main objective to introduce such a network model is to store hetero-associative pattern pairs. This is used to retrieve a pattern given a noisy or incomplete pattern.

BAM Architecture:

The architecture of BAM network consists of two layers of neurons which are connected by directed weighted pare interconnecrions. The network dynamics involve two layers of interaction. The BAM network iterates by sending the signals back and forth between the two layers until all the neurons reach equilibrium. The weights associated with the network are bidirectional. Thus, BAM can respond to the inputs in either layer.



The BAM models are rather efficient when deployed as a part of an AI-based decision-making process, inferring the solution to a specific data analysis problem, based on various associations of many interrelated data.

Generally, the BAM model can be used for a vast of applications, such as:

- **Classification and clustering data**
- **Incomplete data augmentation**
- **Recovering damaged or corrupted data**

The BAM models are very useful whenever the varieties of knowledge, acquired by the ANNs, are not enough for processing the data, introduced to an AI for analysis.

For example, the prediction of words missed out from incomplete texts with ANN, basically requires that the associations of words-to-sentences are stored in the ANN's memory. However, this would incur an incorrect prediction, because the same missing words might occur in more than one incomplete

sentence. In this case, using the BAM provides an ability to store and recall all possible associations of the data entities, such as *words-to-sentences*, *sentences-to-words*, *words-to-words*, and *sentences-to-sentences,* and vice versa. This, in turn, notably increases the quality of prediction.

Hopfield Network:

Hopfield network is a special kind of neural network whose response is different from other neural networks. It is calculated by converging iterative process. It has just one layer of neurons relating to the size of the input and output, which must be the same. When such a network recognizes, for example, digits, we present a list of correctly rendered digits to the network. Subsequently, the network can transform a noise input to the relating perfect output.

In 1982, **John Hopfield** introduced an artificial neural network to collect and retrieve memory like the human brain. Here, a neuron is either on or off the situation. The state of a neuron(on +1 or off 0) will be restored, relying on the input it receives from the other neuron. A Hopfield network is at first prepared to store various patterns or memories. Afterward, it is ready to recognize any of the learned patterns by uncovering partial or even some corrupted data about that pattern, i.e., it eventually settles down and restores the closest pattern. Thus, similar to the human brain, the Hopfield model has stability in pattern recognition.

Introduction to Hopfield Network

A Hopfield network is a particular type of single-layered neuron network. Dr. John J. Hopfield invented it in 1982. These networks were introduced to collect and retrieve memory and store various patterns. Also, auto-association and optimization of the task can be done using these networks. In this network, each node is fully connected(recurrent) to other nodes. These nodes exist only in two states: **ON (1)** or **OFF (0)**. These states can be restored based on the input received from other nodes. Unlike other neural networks, the output of the Hopfield network is finite. Also, the input and output sizes must be the same in these networks. .

The Hopfield network consists of associative memory. This memory allows the system to retrieve the memory using an incomplete portion. The network can restore the closest pattern using the data captured in associative memory. This feature of Hopfield networks makes it a good candidate for pattern recognition.
Associative memory is a content addressable memory that establishes a relation between the input vector and the output target vector. It enables the reallocation of data stored in the memory based on its similarity with the input vector.

The Hopfield networks are categorized into two categories. These are:

Discrete Networks

These networks give any of the two discrete outputs. Based on the output received, further two types:

1. **Binary:** In this type, the output is either 0 or 1.
2. **Bipolar:** In bipolar networks, the output is either -1 (When output < 0) or 1 (When output > 0)

Continuous Networks

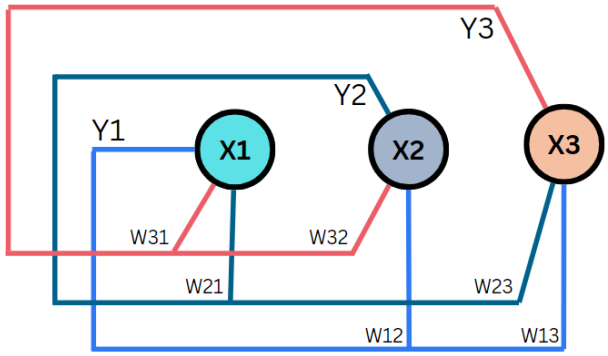
Instead of receiving binary or bipolar output, the output value lies between 0 and 1.

The Architecture of Hopfield Network

The architecture of the Hopfield network consists of the following elements:

- Individual nodes preserve their states until required an update.
- The node to be updated is selected randomly.
- Each node is connected to all other nodes except itself.
- The state of each node is either 0/1 or 1/-1.
- The Hopfield network structure is symmetric, i.e., Wij = Wji for all i's and j's.

The representation of the sample architecture of the Hopfield network having three nodes is as follows:



In the above diagram, each symbol represents:
x1, x2, x3 - represents the input.
y1, y2, y3 - represents output obtained from each node.
Wij - represents the weight associated with the connection from i to j.

Energy Function in Hopfield Network

In Hopfield networks, there are two different types of updations.

- **Synchronous:** Updating all the nodes simultaneously each time.
- **Asynchronous:** Updating only one node at a time. That node is selected randomly or based on specific rules.

In asynchronous updation, each state of Hopfield networks is associated with an energy value. The value is obtained from a function, and that function is named an energy function. This function can decrease or remain unchanged during updation. This energy function of the Hopfield network is defined as:

Where Wij = weight of the connection between i to j
Xi = value of input i
Xj = value of input j

A network is considered to be in a stable state if the energy function tends to be minimum. .

UNIT-II Unsupervised Learning Network- Introduction, Fixed Weight Competitive Nets, Maxnet, Hamming Network, Kohonen Self-Organizing Feature Maps, Learning Vector Quantization, Counter Propagation Networks, Adaptive Resonance Theory Networks. Special Networks-Introduction to various networks.

Unsupervised Learning Network

Unsupervised learning

Unsupervised learning is the training of a machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of the machine is to group unsorted information according to similarities, patterns, and differences without any prior training of data.

Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore the machine is restricted to find the hidden structure in unlabeled data by itself.

For instance, suppose it is given an image having both dogs and cats which it has never seen.

Thus the machine has no idea about the features of dogs and cats so we can't categorize it as 'dogs and cats '. But it can categorize them according to their similarities, patterns, and differences, i.e., we can easily categorize the above picture into two parts. The first may contain all pics having **dogs** in them and the second part may contain all pics having **cats** in them. Here you didn't learn anything before, which means no training data or examples.

It allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with unlabelled data.

Unsupervised learning is classified into two categories of algorithms:

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Types of Unsupervised Learning:-

Clustering

1. Exclusive (partitioning)
2. Agglomerative
3. Overlapping
4. Probabilistic

Clustering Types:-

1. Hierarchical clustering
2. K-means clustering

<u>Advantages of unsupervised learning:</u>

- It does not require training data to be labeled.
- Dimensionality reduction can be easily accomplished using unsupervised learning.
- Capable of finding previously unknown patterns in data.
- **Flexibility**: Unsupervised learning is flexible in that it can be applied to a wide variety of problems, including clustering, anomaly detection, and association rule mining.
- **Exploration**: Unsupervised learning allows for the exploration of data and the discovery of novel and potentially useful patterns that may not be apparent from the outset.
- **Low cost**: Unsupervised learning is often less expensive than supervised learning because it doesn't require labeled data, which can be time-consuming and costly to obtain.

<u>Disadvantages of unsupervised learning :</u>

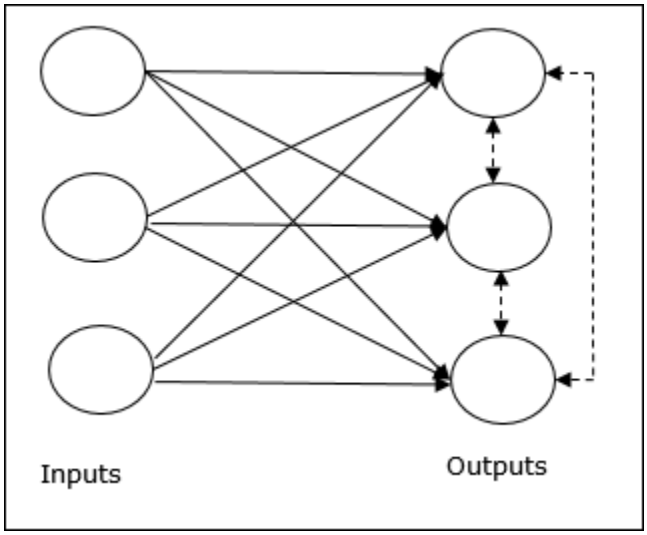
- Difficult to measure accuracy or effectiveness due to lack of predefined answers during training.
- The results often have lesser accuracy.
- The user needs to spend time interpreting and label the classes which follow that classification.
- **Lack of guidance**: Unsupervised learning lacks the guidance and feedback provided by labeled data, which can make it difficult to know whether the discovered patterns are relevant or useful.
- **Sensitivity to data quality**: Unsupervised learning can be sensitive to data quality, including missing values, outliers, and noisy data.

- **Scalability**: Unsupervised learning can be computationally expensive, particularly for large datasets or complex algorithms, which can limit its scalability.

This learning process is independent. During the training of ANN under unsupervised learning, the input vectors of similar type are combined to form clusters. When a new input pattern is applied, then the neural network gives an output response indicating the class to which input pattern belongs. In this, there would be no feedback from the environment as to what should be the desired output and whether it is correct or incorrect. Hence, in this type of learning the network itself must discover the patterns, features from the input data and the relation for the input data over the output.

Basic Concept of Competitive Network

This network is just like a single layer feed-forward network having feedback connection between the outputs. The connections between the outputs are inhibitory type, which is shown by dotted lines, which means the competitors never support themselves.



Basic Concept of Competitive Learning Rule

As said earlier, there would be competition among the output nodes so the main concept is - during training, the output unit that has the highest activation to a given input pattern, will be declared the winner. This rule is also called Winner-takes-all because only the winning neuron is updated and the rest of the neurons are left unchanged.

Fixed Weight Competitive Nets

During training process also the weights remains fixed in these competitive networks. The idea of competition is used among neurons for enhancement of contrast in their activation functions. In this, two networks- **Maxnet and Hamming networks**

In most of the neural networks using unsupervised learning, it is essential to compute the distance and perform comparisons.

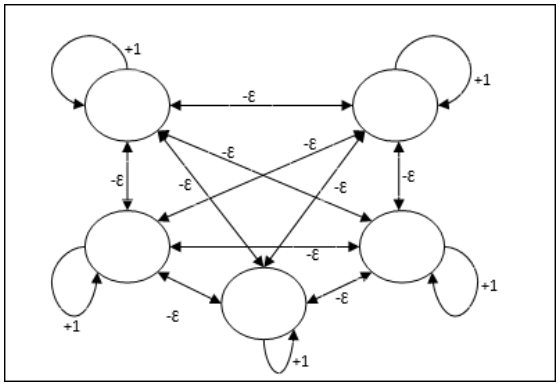
Max Net

This is also a fixed weight network, which serves as a subnet for selecting the node having the highest input. All the nodes are fully interconnected and there exists symmetrical weights in all these weighted interconnections.

When a net is trained to classify the input signal into one of the output categories, A, B, C, D, E, J, or K, the net sometimes responded that the signal was both a C and a K, or both an E and a K, or both a J and a K, due to similarities in these character pairs. In this case it will be better to include additional structure in the net to force it to make a definitive decision. The mechanism by which this can be accomplished is called competition.

The most extreme form of competition among a group of neurons is called Winner-Take-All, where only one neuron (the winner) in the group will have a nonzero output signal when the competition is completed. An example of that is the MAXNET.

Architecture



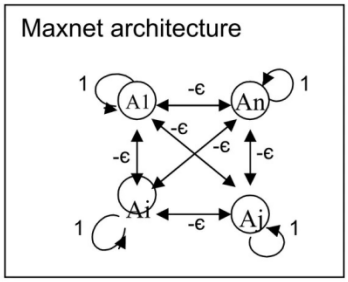
It uses the mechanism which is an iterative process and each node receives inhibitory inputs from all other nodes through connections. The single node whose value is maximum would be active or winner and the activations of all other nodes would be inactive.

- -There is no need for training the network, since the weights are fixed.
- -The Maxnet operates as a recurrent recall network that operates in an auxiliary mode.

Activation functions

$$f(net) = \begin{cases} net & \text{if } net > 0 \\ 0 & \text{otherwise} \end{cases}$$

Where Є is usually positive less than 1 number.



Maxnet Algorithm

Step 1: Set activations and weights,

aj (0) is the starting input value to node *Aj*

$$\omega ij = \begin{cases} 1 & \text{for } i = j \\ -\epsilon & i \neq j \end{cases}$$

Step 2: If more than one node has nonzero output, do step 3 to 5.

Step 3: Update the activation (output) at each node for

$j = 1, 2, 3\ldots\ldots, n$

$aj\ (t+1) = f\ [\ aj\ (t) - \epsilon \sum ai\ (t)]$ $\quad i \neq j$

$\epsilon < 1/m$ where m is the number of competitive neurons

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

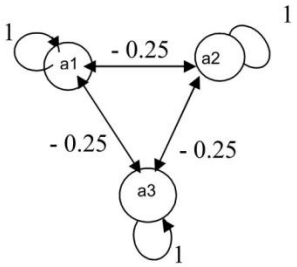
Step 4: Save activations for use in the next iteration.

$aj\ (t+1) \rightarrow aj\ (t)$

Step 5: Test for stopping condition. If more than one node has a nonzero output then Go To step 3, Else Stop.

***Example*:** A Maxnet has three inhibitory weights a 0.25 (ϵ = 0.25). The net is initially activated by the input signals [0.1 0.3 0.9]. The activation function of the neurons is:

$$f(net) = \begin{cases} net & \text{if net} > 0 \\ 0 & \text{otherwise} \end{cases}$$



Find the final winning neuron.

Solution:

First iteration: The net values are:

a**1** (1) = f [0.1 - 0.25(0.3+0.9)] = 0

a**2** (1) = f [0.3 - 0.25(0.1+0.9)] = 0.05

a**3** (1) = f [0.9 - 0.25(0.1+0.3)] = 0.8

Second iteration: a**1** (2) = f [0 - 0.25(0.05+0.8)] = 0

a**2** (2) = f [0.05 - 0.25(0 +0.8)] = 0

a**3** (2) = f [0.8 -0.25(0+0.05)] =0.7875

Then the 3rd neuron is the winner.

Hamming networks

This kind of network is Hamming network, where for every given input vectors, it would be clustered into different groups. Following are some important features of Hamming Networks −

- Lippmann started working on Hamming networks in 1987.
- It is a single layer network.
- The inputs can be either binary {0, 1} of bipolar {-1, 1}.
- The weights of the net are calculated by the exemplar vectors.
- It is a fixed weight network which means the weights would remain the same even during training.

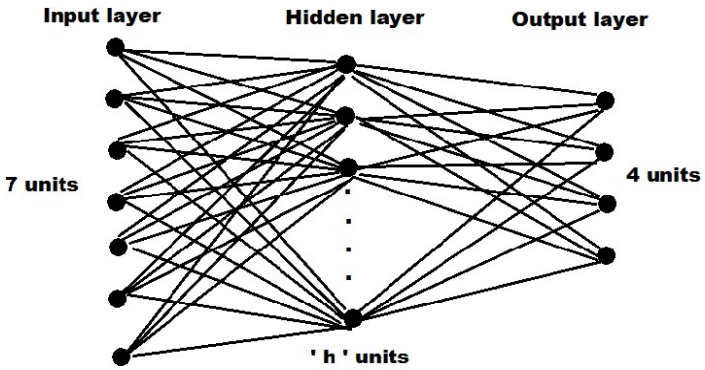


Fig. 1. Layout of (7, 4) Hamming Code ANN

Hamming Distance

Hamming distance of two vectors, x and y of dimension n

x.y = a - d

Where: a is number of bits in agreement in x & y(No. of Similarity bits in x & y), and d is number of bits different in x and y (No. of Dissimilarity bits in x & y).

The value "a - d" is the Hamming distance existing between two vectors. Since, the total number of components is n, we have,
n = a + d
i.e., d = n - a

On simplification, we get
x.y = a - (n - a)

x.y = 2a - n

2a = x.y + n

a = 1/2x.y + 1/2n
From the above equation, it is clearly understood that the weights can be set to one-half the exemplar vector and bias can be set initially to n/2

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

Design a Hamming network to find 3 vectors(min)
and when new vector x is given what
would be the o/p vector produced?
which vector is nearest to the new training
vector x. How do you calculate with Hamming
net

There are 3 vector with length 5 n=5.

$x_1 = [1, 1, 1, 1, 1]$ $x_2 = [-1, 1, 1, 1, -1]$

$x_3 = [1, -1, 1, -1, 1]$

$x = [1, 1, 1, -1, -1]$

Hamming N/W consist of 2 layers

1st layer = 5nodes
2nd layer = 3nodes.

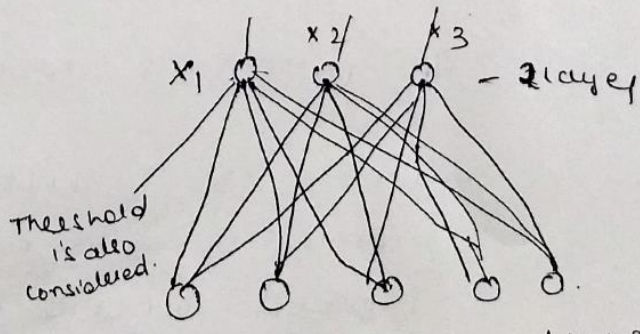
Now we calculate the weight matrix for
connection between 1st layer and 2nd layer

$$W = \frac{1}{2}\begin{bmatrix} x_1^{\text{Transpose}} \\ x_2^T \\ x_3^T \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 \end{bmatrix}$$

Threshold vector $= -\frac{n}{2} = \frac{5}{2}$

$$\theta = \begin{bmatrix} -\frac{5}{2} \\ -\frac{5}{2} \\ -\frac{5}{2} \end{bmatrix}$$

o/p ?



Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

when x is given at i/p new i/p node = [1, 1, 1, -1, -1]

$$0 = Wx + \theta$$

$$= \frac{1}{2}\begin{bmatrix} 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} -5 \\ -5 \\ -5 \end{bmatrix}$$

$$= \frac{1}{2}\begin{bmatrix} 1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} -5 \\ -5 \\ -5 \end{bmatrix}$$

$$= \frac{1}{2}\begin{bmatrix} -3 \\ -1 \\ 1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} -5 \\ -5 \\ -5 \end{bmatrix}$$

$$= \frac{1}{2}\begin{bmatrix} -8 \\ -6 \\ 4 \end{bmatrix} = \begin{bmatrix} -4 \\ -3 \\ -2 \end{bmatrix}$$

$$0 = \begin{bmatrix} -4 \\ -3 \\ -2 \end{bmatrix}$$

O/P provides negative Hamming distance and the lowest of the distance blw new vector X to X_3

i/p pattern is close to stored pattern X_3

and now X_3 is shortlisted for further processing and now by using maxnet fixed weight is calculated and

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

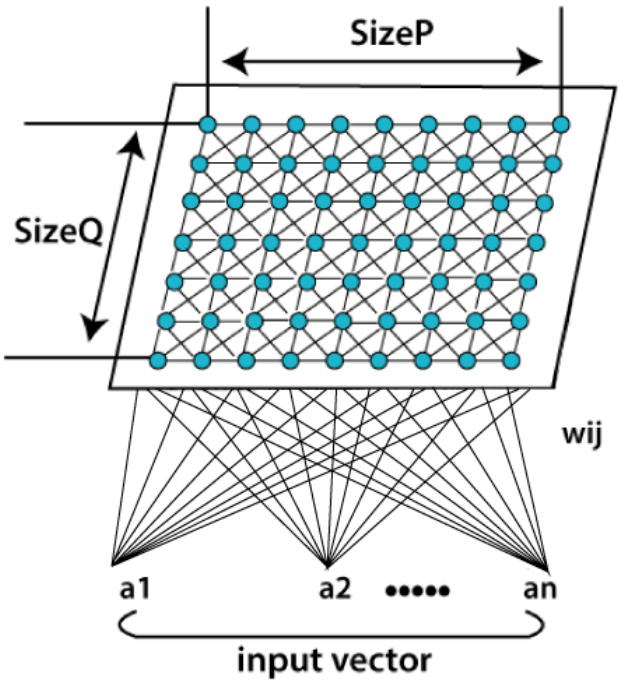
Kohonen Self- Organizing Feature Map

Kohonen Self-Organizing feature map (SOM) refers to a neural network, which is trained using competitive learning. Basic competitive learning implies that the competition process takes place before the cycle of learning. The competition process suggests that some criteria select a winning processing element.

The self-organizing map makes topologically ordered mappings between input data and processing elements of the map. Topological ordered implies that if two inputs are of similar characteristics, the most active processing elements answering to inputs that are located closed to each other on the map. The weight vectors of the processing elements are organized in ascending to descending order. $W_i < W_{i+1}$ for all values of i or W_{i+1} for all values of i (this definition is valid for one-dimensional self-organizing map only).

The self-organizing map is typically represented as a two-dimensional sheet of processing elements described in the figure given below. Each processing element has its own weight vector, and learning of SOM (self-organizing map) depends on the adaptation of these vectors. The processing elements of the network are made competitive in a self-organizing process, and specific criteria pick the winning processing element whose weights are updated. Generally, these criteria are used to limit the Euclidean distance between the input vector and the weight vector. SOM (self-organizing map) varies from basic competitive learning so that instead of adjusting only the weight vector of the winning processing element also weight vectors of neighboring processing elements are adjusted

It is discovered by Finnish professor and researcher Dr. Teuvo Kohonen in 1982. The self-organizing map refers to an unsupervised learning model proposed for applications in which maintaining a topology between input and output spaces



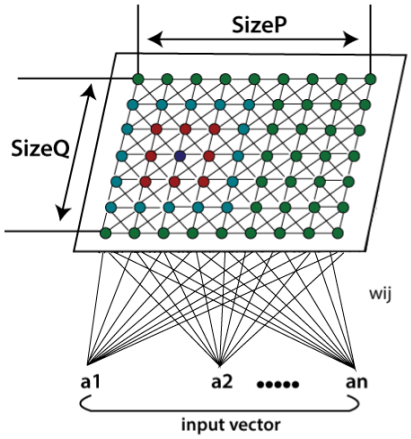
Kohonen Architecture

All the entire learning process occurs without supervision because the nodes are self-organizing. They are also known as feature maps, as they are basically retraining the features of the input data, and simply grouping themselves as indicated by the similarity between each other. It has practical value for visualizing complex or huge quantities of high dimensional data and showing the relationship between them into a low, usually two-dimensional field to check whether the given unlabeled data have any structure to it.

A self-Organizing Map (SOM) varies from typical artificial neural networks (ANNs) both in its architecture and algorithmic properties. Its structure consists of a single layer linear 2D grid of neurons, rather than a series of layers. All the nodes on this lattice are associated directly to the input vector, but not to each other. It means the nodes don't know the values of their neighbors, and only update the weight of their associations as a function of the given input. The grid itself is the map that coordinates itself at each iteration as a function of the input data. As such, after clustering, each node has its own coordinate (i,j), which enables one to calculate Euclidean distance between two nodes

A Self-Organizing Map utilizes competitive learning instead of error-correction learning, to modify its weights. It implies that only an individual node is activated at each cycle in which the features of an occurrence of the input vector are introduced to the neural network, as all nodes compete for the privilege to respond to the input.

The selected node- the Best Matching Unit (BMU) is selected according to the similarity between the current input values and all the other nodes in the network. The node with the fractional Euclidean difference between the input vector, all nodes, and its neighboring nodes is selected and within a specific radius, to have their position slightly adjusted to coordinate the input vector. By experiencing all the nodes present on the grid, the whole grid eventually matches the entire input dataset with connected nodes gathered towards one area, and dissimilar ones are isolated.



A Kohonen model with the BMW in blue, the layer inside the neighborhood radius in light blue and red, and the nodes outside are green.

Algorithm:

Step:1

Each node weight w_ij initialize to a random value.

Step:2

Choose a random input vector x_k.

Step:3

Repeat steps 4 and 5 for all nodes on the map.

Step:4

Calculate the Euclidean distance between weight vector w_{ij} and the input vector x(t) connected with the first node, where t, i, j =0.

Step:5

track the node that generates the smallest distance t.

Step:6

Calculate the overall Best Matching Unit (BMU). It means the node with the smallest distance from all calculated ones.

Step:7

Discover topological neighborhood *βij(t) its radius σ(t)* of BMU in Kohonen Map.

Step:8

Repeat for all nodes in the BMU neighborhood: Update the weight vector w_ij of the first node in the neighborhood of the BMU by including a fraction of the difference between the input vector x(t) and the weight w(t) of the neuron.

Wij(new)=wij(old)+alpha[xi-wij(old)]

Step:9

Repeat the complete iteration until reaching the selected iteration limit t=n.

Here, step 1 represents initialization phase, while step 2 to 9 represents the training phase.

Where;

t = current iteration.

i = row coordinate of the nodes grid.

J = column coordinate of the nodes grid.

W= weight vector

w_ij = association weight between the nodes i,j in the grid.

X = input vector

X(t)= the input vector instance at iteration t

β_ij = the neighborhood function, decreasing and representing node i,j distance from the BMU.

σ(t) = The radius of the neighborhood function, which calculates how far neighbor nodes are examined in the 2D grid when updating vectors.

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

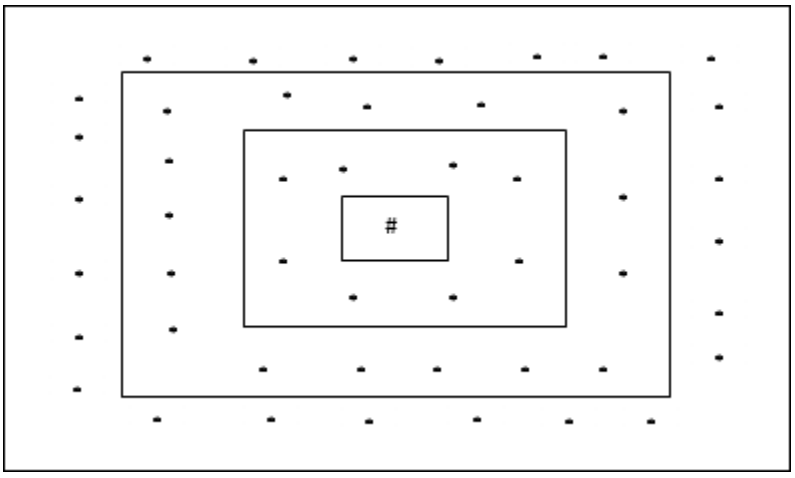
MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

Neighbor Topologies in Kohonen SOM

There can be various topologies, however the following two topologies are used the most −

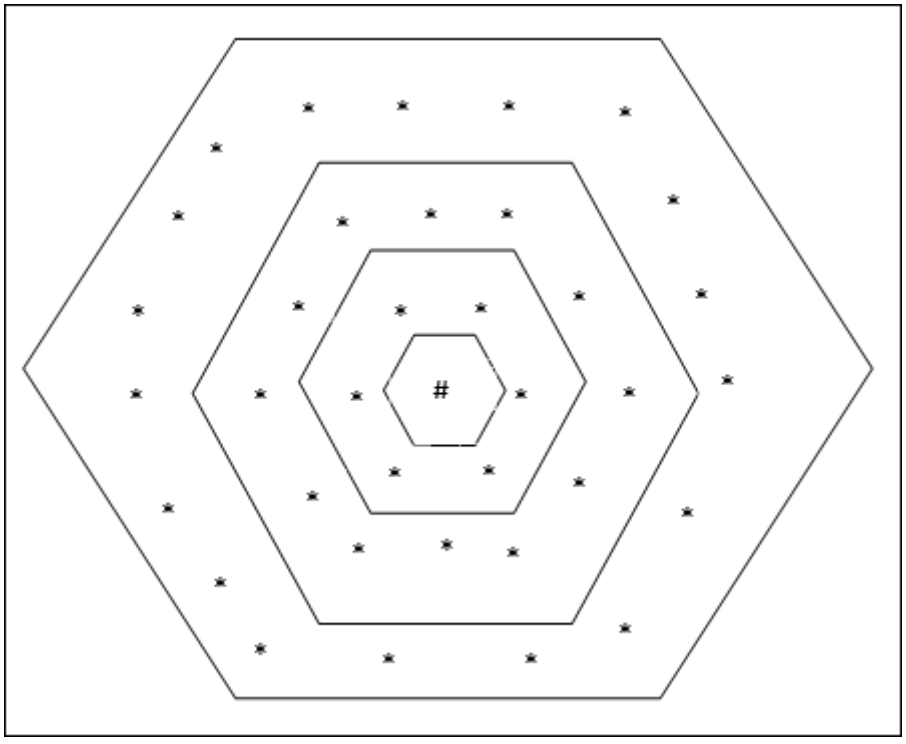
Rectangular Grid Topology

This topology has 24 nodes in the distance-2 grid, 16 nodes in the distance-1 grid, and 8 nodes in the distance-0 grid, which means the difference between each rectangular grid is 8 nodes. The winning unit is indicated by #.



Hexagonal Grid Topology

This topology has 18 nodes in the distance-2 grid, 12 nodes in the distance-1 grid, and 6 nodes in the distance-0 grid, which means the difference between each rectangular grid is 6 nodes. The winning unit is indicated by #.



Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

construct KSOFM to its clustel four given vectol

[0, 0, 1, 1] , [1, 0, 0, 0] [0, 1, 1, 0]

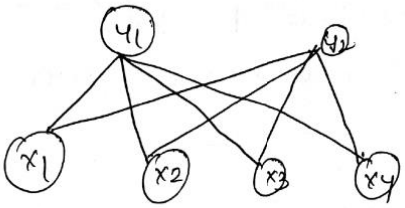
[0, 0, 0, 1]

No of clusters to be formed is 2 Assume an

initial learning rate of 0.5

No of input vector n = 4

No of clusters m = 2



Initialize the weight Randomly.

$$W_{ij} = \begin{matrix} & w_1 & w_2 \\ w_{11} & 0.2 & 0.9 \\ w_{12} & 0.4 & 0.7 \\ w_{13} & 0.6 & 0.5 \\ w_{14} & 0.8 & 0.3 \end{matrix} \begin{matrix} w_{21} \\ w_{22} \\ w_{23} \\ w_{24} \end{matrix}$$

Calculate Euclidean distance.

$$D = \sum (w_{ij} - x_i)^2$$

for the first input vector, [0, 0, 1, 1] for cluster (1)

$$D(1) = [0.2 - 0]^2 + [0.4 - 0]^2 + [0.6 - 1]^2 + [0.8 - 1]^2$$

$$= 0.04 + 0.16 + 0.36 + 0.04$$

$$= 0.4$$

for the first input vector [0, 0, 1, 1] for cluster (2)

$$D(2) = [0.9 - 0]^2 + [0.7 - 0]^2 + [0.5 - 1]^2 + [0.3 - 1]^2$$

$$0.81 + 0.49 + 0.25 + 0.49$$

$$= 2.04$$

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

$D(1) < D(2)$ is winning cluster $\Rightarrow J = 1$

first iterations.

update weights on winning cluster

$$w_{ij}(new) = w(old) + \alpha [x_i - w_{ij}(old)]$$

$$w_{11}(new) = w_{11}(old) + 0.5[0 - 0.2]$$
$$0.2 + 0.5[0 - 0.2)$$
$$= 0.1$$

$$w_{21} = w_{21}(old) + 0.5[0 - 0.4]$$
$$0.4 + 0.5(0 - 0.4)$$
$$= 0.2$$

$$w_{31}(n) = 0.6 + 0.5[1 - 0.6] = 0.8$$

$$w_{41}(n) = 0.8 + 0.5[1 - 0.8] = 0.9$$

updated weight matrix is

$$w_{ij} = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.7 \\ 0.8 & 0.5 \\ 0.9 & 0.3 \end{bmatrix}$$

Second input $X = [1\ 0\ 0\ 0]$

$$D(1) = (0.1 - 1)^2 + (0.2 - 0)^2 + (0.8 - 0)^2 + (0.9 - 0)^2$$
$$= 0.81 + 0.04 + 0.64 + 0.81 = 2.3$$

$$(D(2) = (0.9 - 1)^2 + (0.7 - 0)^2 + (0.5 - 0)^2 + (0.3 - 0)^2$$
$$= 0.01 + 0.49 + 0.25 + 0.09] = 0.84$$

$D(2) < D(1)$ $\quad J = 2$.

weight updation on cluster 2

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

$w_{12} = w_{12}(old) + \alpha(x - w_{12}(old))$

$= 0.9 + 0.5(1-0.9) = 0.45$

$w_{22} = 0.7 + 0.5(0-0.7) = 0.35$

$w_{32} = 0.5 + 0.5(0-0.5) = 0.2$

$w_{42} = 0.3 + 0.5(0-0.3) = 0.15$

Now update weight

$$\begin{bmatrix} 0.1 & 0.45 \\ 0.2 & 0.35 \\ 0.8 & 0.25 \\ 0.9 & 0.15 \end{bmatrix}$$

Now Third input vector (0, 1, 1, 0)

$D(1) = 1.5 \quad D(2) = 1.91$

Winning cluster D(1)

$w_{11} = 0.05$
$w_{21} = 0.6$
$w_{31} = 0.9$
$w_{41} = 0.45$

$\rightarrow$

$$\begin{bmatrix} 0.05 & 0.45 \\ 0.06 & 0.35 \\ 0.9 & 0.25 \\ 0.45 & 0.15 \end{bmatrix}$$

Updating the weight using formula

Now fourth input vector taken into consideration

[0, 0, 0, 1]

$D(1) = 1.475$

$D(2) = 1.81$

Winning cluster is D1

Update the weight using formula.

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

$w_{11} = 0.025$

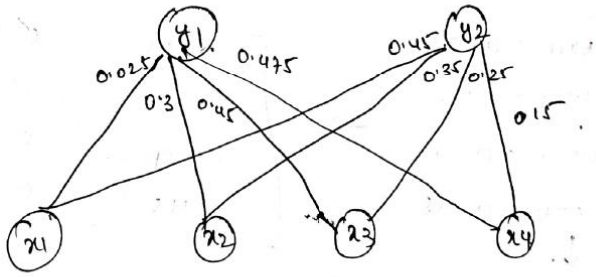
$w_{21} = 0.3$

$w_{31} = 0.45$

$w_{41} = 0.475$

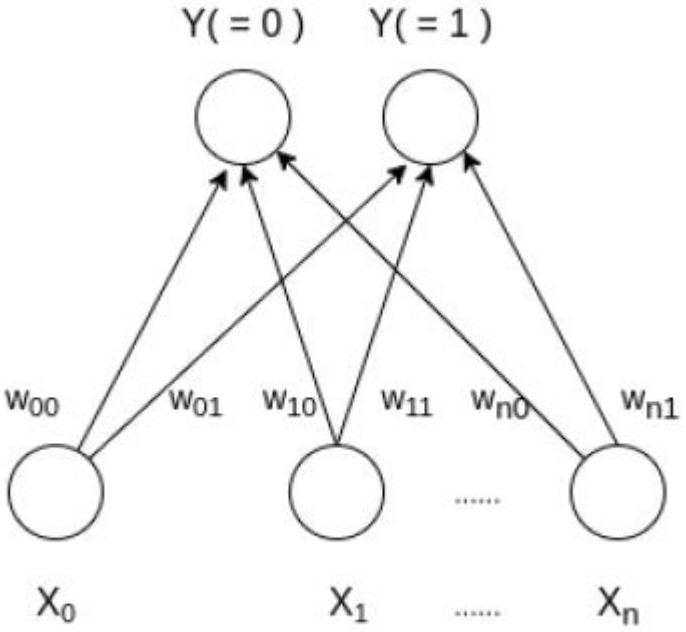
$$w_{15} = \begin{bmatrix} 0.025 & 0.45 \\ 0.3 & 0.35 \\ 0.45 & 0.25 \\ 0.475 & 0.15 \end{bmatrix}$$

Now for the all the input vectors the New weight are updated.



For all the 4 input vector given 2 clusters and updated associated weights are being mapped by using SOM (Self Organizing map).

Learning Vector Quantization (or LVQ) is a type of Artificial Neural Network which also inspired by biological models of neural systems. It is based on prototype supervised learning classification algorithm and trained its network through a competitive learning algorithm similar to Self Organizing Map. It can also deal with the multiclass classification problem. LVQ has two layers, one is the Input layer and the other one is the Output layer. The architecture of the Learning Vector Quantization with the number of classes in an input data and n number of input features for any sample is given



:

How Learning Vector Quantization works?

Let's say that an input data of size (m, n) where m is the number of training examples and n is the number of features in each example and a label vector of size (m, 1). First, it initializes the weights of size (n, c) from the first c number of training samples with different labels and should be discarded from all training samples. Here, c is the number of classes. Then iterate over the remaining input data, for each training example, it updates the winning vector (weight vector with the shortest distance (e.g Euclidean distance) from the training example).

The weight updation rule is given by:

```
if correctly_classified:
wij(new) = wij(old) + alpha(t) * (xi^k - wij(old))
else:
wij(new) = wij(old) - alpha(t) * (xik - wij(old))
```

where alpha is a learning rate at time t, j denotes the winning vector, i denotes the i^{th} feature of training example and k denotes the k^{th} training example from the input data. After training the LVQ network, trained weights are used for classifying new examples. A new example is labelled with the class of the winning vector.

Algorithm LVQ :

Step 1: Initialize reference vectors.
from a given set of training vectors, take the first "n" number of clusters training vectors and use them as weight vectors, the remaining vectors can be used for training.

Assign initial weights and classifications randomly

Step 2: Calculate Euclidean distance for i=1 to n and j=1 to m,

$D(j) = \Sigma\Sigma (x_i - W_{ij})^2$

find winning unit index j, where D(j) is minimum

Step 3: Update weights on the winning unit w_i using the following conditions:

if T = J then $w_i(new) = w_i(old) + \alpha[x - w_i(old)]$

if T ≠ J then $w_i(new) = w_i(old) - \alpha[x - w_i(old)]$

Step 4: Check for the stopping condition if false repeat the above steps.

Below is the implementation

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

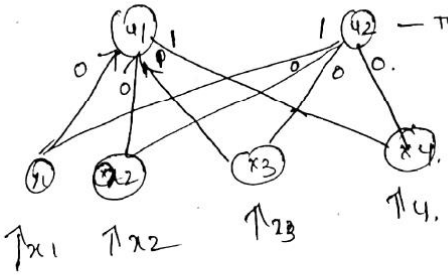
MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

construct an LVQ net with 3 vectors assigned to 2 classes. given vectors along with classes are shown

vector	class
[0 0 0 1]	2
[0 1 0 0]	1
[0 1 1 0]	1

Initial weight for 2 class of clusters are

[0 0 1 1]	1
[1 0 0 0]	2



Initial weight vectors as

$w_1 = [0\ 0\ 1\ 1]$ $\quad w_2 = [1\ 0\ 0\ 0]$

let $\alpha = 0.1$

$$w = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$

for the first input vect $x = [0\ 0\ 0\ 1]$ $T = 2$

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

Calculate Euclidean distance.

$$D(j) = \sum (w_{ij} - x_i)^2$$

$D(1) = (0-0)^2 + (0-0)^2 + (1-0)^2 + (1-1)^2 = 1$

$D(2) = (1-0)^2 + (0-0)^2 + (0-0)^2 + (0-1)^2 = 2$

$D(1) < D(2)$ $\Rightarrow$ $D(1)$ is minimum

the winning node is $D(1)$ which belongs to class 2 $\quad T = 2$

and weight of cluster $W1$ belong to class 1 $\quad J = 1$

$T \neq J$.

then according to the formula of LVQ.

$$w_j(new) = w_j(old) - \alpha(x - w_j(old))$$

$w_{11} = 0 - 0.1(0-0) = 0.$

$w_{21} = w_{21}(old) - \alpha(x - w_j(old))$

$= 0 - 0.1[0-0] = 0$

$w_{31} = 1 - 0.1(0-1) = 1.1$

$w_{41} = 1 - 0.1(0-1) = 1$

New weight updated

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1.1 & 0 \\ 1 & 0 \end{bmatrix}$$

take the second input $X = [1\ 1\ 0\ 0]$

belong to class 1 $T = 1$

$D(1) = (0-1)^2 + (0-1)^2 + (1.1-0)^2 + (1-0)^2 = 4.2$

$D(2) = (1-1)^2 + (0-1)^2 + (0-0)^2 + (0-0)^2 = 1$

$D2 < D1$ $\quad D2$ is winning node.

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

$T = 1$

$J = 2$.

D_2 belongs to class 2

Now consider $T \neq J$ second input $x = [1\ 1\ 0\ 0]$.

$w_{11} = 1 - 0.1[1-1] = 1$

$w_{22} = 0 - 0.1[1-0] = -0.1$

$w_{32} = 0 - 0.1[0-0] = 0$

$w_{42} = 0 - 0.1[0-0] = 0$

$$w_2 = \begin{bmatrix} 0 & 1 \\ 0 & -0.1 \\ 1.1 & 0 \\ 1 & 0 \end{bmatrix}$$

third input $= x = [0\ 1\ 1\ 0]$ $T = 1$.

$D(1) = (0-0)^2 + (0-1)^2 + (1.1-1)^2 + (1-0)^2 = 2.01$

$D(2) = (1-0)^2 + (-0.1-1)^2 + (0-1)^2 + (0-0)^2 = 3.21$

$D_2 > D_1$ Hence D_1 is winning node.

$T = 1$ $S = 1$

$T = J$.

then formula becomes $w_J = w_J(\text{old}) + \alpha[x - w_J(\text{old})]$

$w_{11}(n) = 0 + 0.1(0-0) = 0$

$w_{21}(n) = 0 + 0.1(1-0) = 0.1$

$w_{31}(n) = 1.1 + 0.1(1-1.1) = 1.09$

$w_{41}(n) = 1 + 0.1(0-1) = 0.9$

final weight are

$$\begin{bmatrix} 0 & 1 \\ 0.1 & -0.1 \\ 1.09 & 0 \\ 0.9 & 0 \end{bmatrix}$$

Counter propagation network

Counter propagation network (CPN) were proposed by Hecht Nielsen in 1987.They are multilayer network based on the combinations of the input, output, and clustering layers. The application of counter propagation net are data compression, function approximation and pattern association. The counter propagation network is basically constructed from an instar-outstar model. This model is three layer neural network that performs input-output data mapping, producing an output vector y in response to input vector x, on the basis of competitive learning. The three layer in an instar-outstar model are the input layer, the hidden(competitive) layer and the output layer.

There are two stages involved in the training process of a counter propagation net. The input vector are clustered in the first stage. In the second stage of training, the weights from the cluster layer units to the output units are tuned to obtain the desired response.

There are two types of counter propagation network:

1. **Full counter propagation network**
2. **Forward-only counter propagation network**

Full CPN

• The Full CPN allows to produce a correct output even when it is given an input vector that is partially incomplete or incorrect.

• In first phase, the training vector pairs are used to form clusters using either dot product or Euclidean distance.

• If dot product is used, normalization is a must.

• During second phase, the weights are adjusted between the cluster units and output units.

• The architecture of CPN resembles an instar and outstar model.

• The model which connects the input layers to the hidden layer is called Instar model and the model which connects the hidden layer to the output layer is called Outstar model.

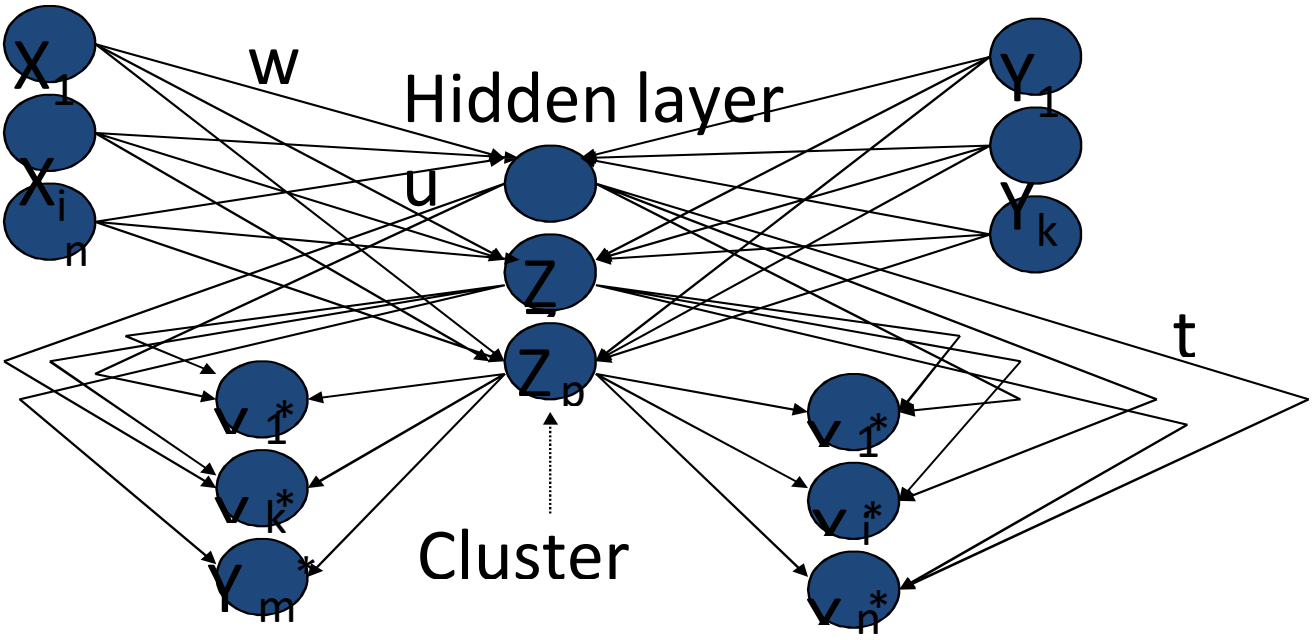
• The weights are updated in both the Instar (in first phase) and Outstar model (second phase).

• The network is fully interconnected network.

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

Architecture of Full Counter propagation



First phase of Full CPN

- This phase of training is called as In star modeled training.
- The active units here are the units in the x-input, z-cluster and y-input layers.
- The winning unit uses standard Kohonen learning rule for its weigh updation.
- The rule is: • v ij(new)= vij(old) + α(xi– vij (old)

  = (1- α)vij(old) + α.xi ;where i=1 to n

- w kj(new)= wkj(old) + β(yk– wi k (old)

  = (1- β)wkj(old) + β.yk;where k=1 to n

Second phase of full CPN

- In this phase, we can find only the J unit remaining active in the cluster layer.

• The weights from the winning cluster unit J to the output units are adjusted, so that vector of activation of units in the y ouput layer, y*, is approximation of input vector y; and x* is an approximation of input vector x.

• Here weight updation is done by Grossberg learning rule.

• Here no competition is assumed among units.

• The weight updation rule is given as:

• ujk(new)= ujk(old) + a(yk– ujk (old)

• = (1- a) ujk(old) + a.yk ;where k=1 to n

•tji(new)= tji(old) + b(xi– tji (old)

• = (1- b) tji(old) + b.xi;where i=1 to n

Training Algorithm

• The parameters used are:

• x – Input training vector x=(x1,…,xi,…,xn)

• y - Target output vector y=(y1,…,yk,…,ym)

• zj – activation of cluster unit Zj.

• x* - Approximation to vector x.

• y* - Approximation to vector y.

• vij – weight from x input layer to Z-cluster layer.

• wjk – weight from y input layer to Z-cluster layer.

• tji – weight from cluster layer to X-output layer.

• ujk – weight from cluster layer to Y-output layer.

• α, β – Learning rates during Kohonen learning.

• a, b – Learning rates during Grossberg learning.

Forward-only Counterpropagation network:

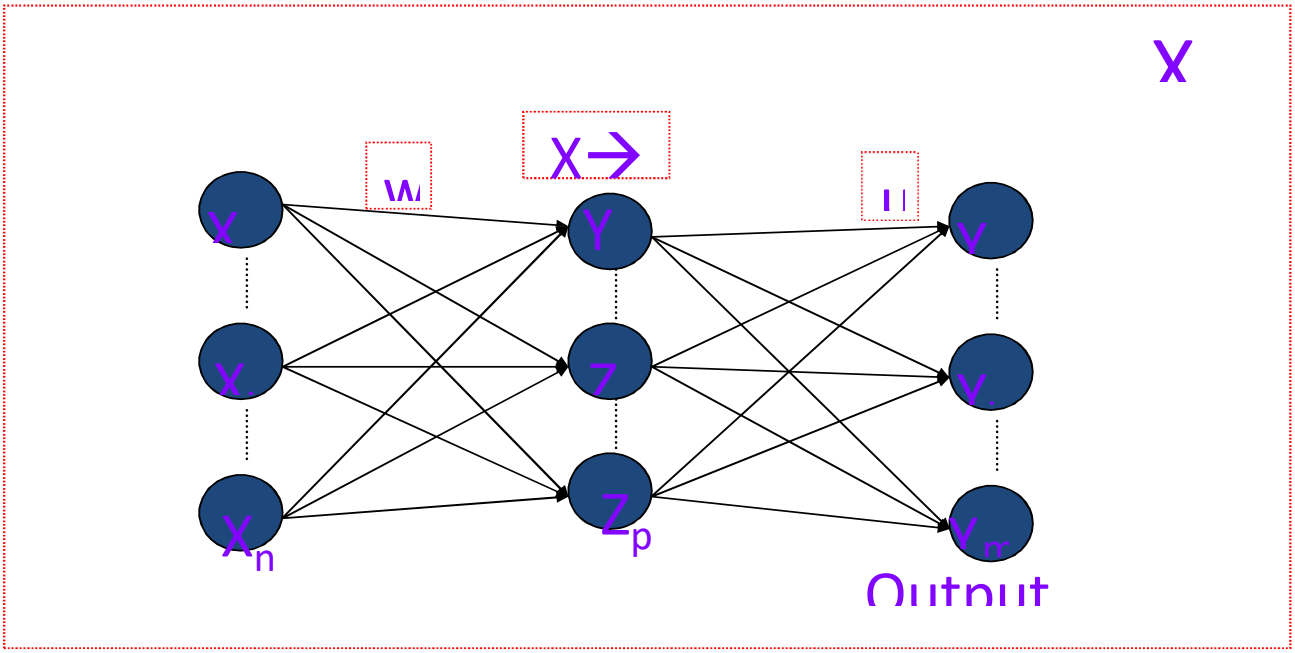
A simplified version of full CPN is the forward-only CPN. Forward-only CPN uses only the x vector to form the cluster on the Kohonen units during phase I training. In case of forward-only CPN, first input vectors are presented to the input units. First, the weights between the input layer and cluster layer are trained. Then the weights between the cluster layer and output layer are trained. This is a specific competitive network, with target known.

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified
ESTD : 2001

Architecture of forward-only CPN

It consists of three layers: input layer, cluster layer and output layer. Its architecture resembles the back-propagation network, but in CPN there exists interconnections between the units in the cluster layer.



The activation Function Will be similar to the Full Propagation

First phase formula from x to z

• The rule is: • v ij(new)= vij(old) + α(xi– vij (old)

= (1- α)vij(old) + α.xi ;where i=1 to n

• vij – weight from x input layer to Z-cluster layer.

The weight updation rule is given as:

• ujk(new)= ujk(old) + a(yk– ujk (old)

• = (1- a) ujk(old) + a.yk ;where k=1 to n

ujk – weight from cluster layer to Y-output layer

Adaptive Resonance Theory (ART) Adaptive resonance theory is a type of neural network technique developed by Stephen Grossberg and Gail Carpenter in 1987. The basic ART uses unsupervised learning technique. The term **"adaptive"** and **"resonance"** used in this suggests that they are open to new learning(i.e. adaptive) without discarding the previous or the old information(i.e. resonance). The ART networks are known to solve the stability-plasticity dilemma i.e., stability refers to their nature of memorizing the learning and plasticity refers to the fact that they are flexible to gain new information. Due to this the nature of ART they are always able to learn new input patterns without forgetting the past. ART networks implement a clustering algorithm. Input is presented to the network and the algorithm checks whether it fits into one of the already stored clusters. If it fits then the input is added to the cluster that matches the most else a new cluster is formed.

Types of Adaptive Resonance Theory(ART) Carpenter and Grossberg developed different ART architectures as a result of 20 years of research. The ARTs can be classified as follows:

- **ART1 –** It is the simplest and the basic ART architecture. It is capable of clustering binary input values.
- **ART2 –** It is extension of ART1 that is capable of clustering continuous-valued input data.
- **Fuzzy ART –** It is the augmentation of fuzzy logic and ART.
- **ARTMAP –** It is a supervised form of ART learning where one ART learns based on the previous ART module. It is also known as predictive ART.
- **FARTMAP –** This is a supervised ART architecture with Fuzzy logic included.

Basic of Adaptive Resonance Theory (ART) Architecture The adaptive resonant theory is a type of neural network that is self-organizing and competitive. It can be of both types, the unsupervised ones(ART1, ART2, ART3, etc) or the supervised ones(ARTMAP). Generally, the supervised algorithms are named with the suffix "MAP". But the basic ART model is unsupervised in nature and consists of :

- F1 layer or the comparison field(where the inputs are processed)
- F2 layer or the recognition field (which consists of the clustering units)
- The Reset Module (that acts as a control mechanism)

The **F1 layer** accepts the inputs and performs some processing and transfers it to the F2 layer that best matches with the classification factor. There exist **two sets of weighted interconnection** for controlling the degree of similarity between the units in the F1 and the F2 layer. The **F2 layer** is a competitive layer. The cluster unit with the large net input becomes the candidate to learn the input pattern first and the rest F2 units are ignored. The **reset unit** makes the decision whether or not the cluster unit is allowed to learn the input pattern depending on

how similar its top-down weight vector is to the input vector and to the decision. This is called the vigilance test.
Thus we can say that the **vigilance parameter** helps to incorporate new memories or new information. Higher vigilance produces more detailed memories, lower vigilance produces more general memories.

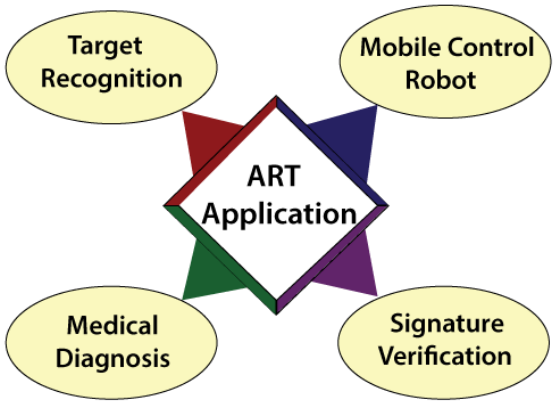
Generally **two types of learning** exists,slow learning and fast learning. In fast learning, weight update during resonance occurs rapidly. It is used in ART1.In slow learning, the weight change occurs slowly relative to the duration of the learning trial. It is used in ART2.

Advantage of Adaptive Resonance Theory (ART)

- It exhibits stability and is not disturbed by a wide variety of inputs provided to its network.
- It can be integrated and used with various other techniques to give more good results.
- It can be used for various fields such as mobile robot control, face recognition, land cover classification, target recognition, medical diagnosis, signature verification, clustering web users, etc.
- It has got advantages over competitive learning. The competitive learning lacks the capability to add new clusters when deemed necessary.
- It does not guarantee stability in forming clusters.

Application of ART:

ART stands for Adaptive Resonance Theory. ART neural networks used for fast, stable learning and prediction have been applied in different areas. The application incorporates target recognition, face recognition, medical diagnosis, signature verification, mobile control robot.



Target recognition:

Fuzzy ARTMAP neural network can be used for automatic classification of targets depend on their radar range profiles. Tests on synthetic data show the fuzzy ARTMAP can result in substantial savings in memory requirements when related to k nearest neighbor(kNN) classifiers. The utilization of multiwavelength profiles mainly improves the performance of both kinds of classifiers.

Medical diagnosis:

Medical databases present huge numbers of challenges found in general information management settings where speed, use, efficiency, and accuracy are the prime concerns. A direct objective of improved computer-assisted medicine is to help to deliver intensive care in situations that may be less than ideal. Working with these issues has stimulated several ART architecture developments, including ARTMAP-IC.

Signature verification:

Automatic signature verification is a well known and active area of research with various applications such as bank check confirmation, ATM access, etc. the training of the network is finished using ART1 that uses global features as input vector and the verification and recognition phase uses a two-step process. In the initial step, the input vector is coordinated with the stored reference vector, which was used as a training set, and in the second step, cluster formation takes place.

Mobile control robot:

Nowadays, we perceive a wide range of robotic devices. It is still a field of research in their program part, called artificial intelligence. The human brain is an interesting subject as a model for such an intelligent system. Inspired by the structure of the human brain, an artificial neural emerges. Similar to the brain, the artificial neural network contains numerous simple computational units, neurons that are interconnected mutually to allow the transfer of the signal from the neurons to neurons. Artificial neural networks are used to solve different issues with good outcomes compared to other decision algorithms.

Limitations of Adaptive Resonance Theory Some ART networks are inconsistent (like the Fuzzy ART and ART1) as they depend upon the order of training data, or upon the learning rate.

Special Networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the brain. ANNs, like people, learn by examples. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning largely involves adjustments to the synaptic connections that exist between the neurons.
Artificial Neural Networks (ANNs) are a type of machine learning model that are inspired by the structure and function of the human brain. They consist of layers of interconnected "neurons" that process and transmit information.

There are several different architectures for ANNs, each with their own strengths and weaknesses. Some of the most common architectures include:

Feedforward Neural Networks: This is the simplest type of ANN architecture, where the information flows in one direction from input to output. The layers are fully connected, meaning each neuron in a layer is connected to all the neurons in the next layer.

Recurrent Neural Networks (RNNs): These networks have a "memory" component, where information can flow in cycles through the network. This allows the network to process sequences of data, such as time series or speech.

Convolutional Neural Networks (CNNs): These networks are designed to process data with a grid-like topology, such as images. The layers consist of convolutional layers, which learn to detect specific features in the data, and pooling layers, which reduce the spatial dimensions of the data.

Autoencoders: These are neural networks that are used for unsupervised learning. They consist of an encoder that maps the input data to a lower-dimensional representation and a decoder that maps the representation back to the original data.

Generative Adversarial Networks (GANs): These are neural networks that are used for generative modeling. They consist of two parts: a generator that learns to generate new data samples, and a discriminator that learns to distinguish between real and generated data.

The model of an artificial neural network can be specified by three entities:

- **Interconnections**
- **Activation functions**
- **Learning rules**

Interconnections:

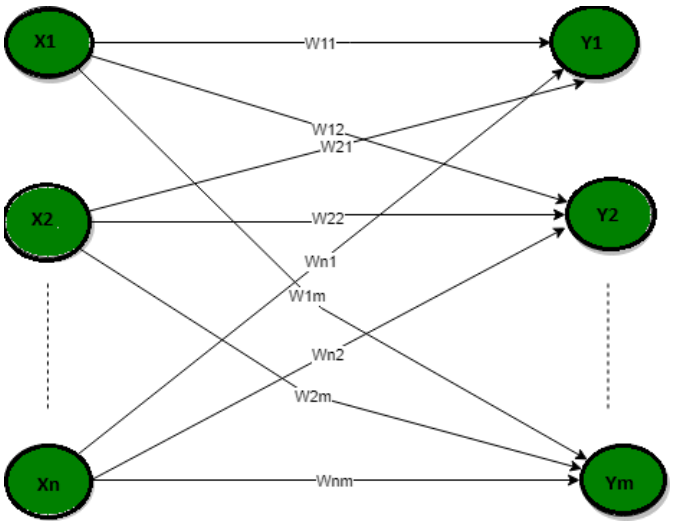
Interconnection can be defined as the way processing elements (Neuron) in ANN are connected to each other. Hence, the arrangements of these processing elements and geometry of interconnections are very essential in ANN.
These arrangements always have two layers that are common to all network architectures, the Input layer and output layer where the input layer buffers the input signal, and the output layer generates the output of the network. The third layer is the Hidden layer, in which neurons are neither kept in the input layer nor in the output layer. These neurons are hidden from the people

who are interfacing with the system and act as a black box to them. By increasing the hidden layers with neurons, the system's computational and processing power can be increased but the training phenomena of the system get more complex at the same time.

There exist five basic types of neuron connection architecture :

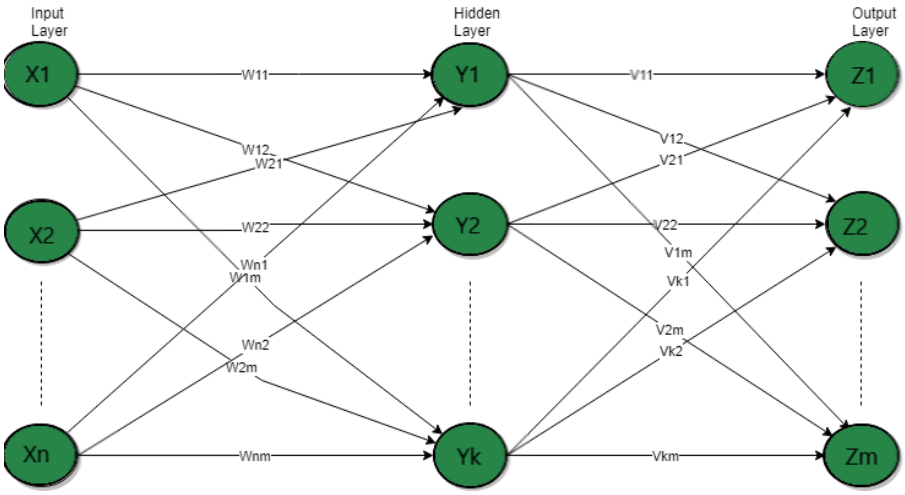
1. Single-layer feed-forward network
2. Multilayer feed-forward network
3. Single node with its own feedback
4. Single-layer recurrent network
5. Multilayer recurrent network

1. Single-layer feed-forward network



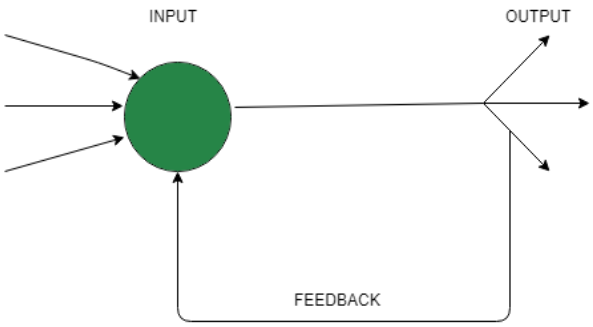
In this type of network, we have only two layers input layer and the output layer but the input layer does not count because no computation is performed in this layer. The output layer is formed when different weights are applied to input nodes and the cumulative effect per node is taken. After this, the neurons collectively give the output layer to compute the output signals.

2. Multilayer feed-forward network



This layer also has a hidden layer that is internal to the network and has no direct contact with the external layer. The existence of one or more hidden layers enables the network to be computationally stronger, a feed-forward network because of information flow through the input function, and the intermediate computations used to determine the output Z. There are no feedback connections in which outputs of the model are fed back into itself.

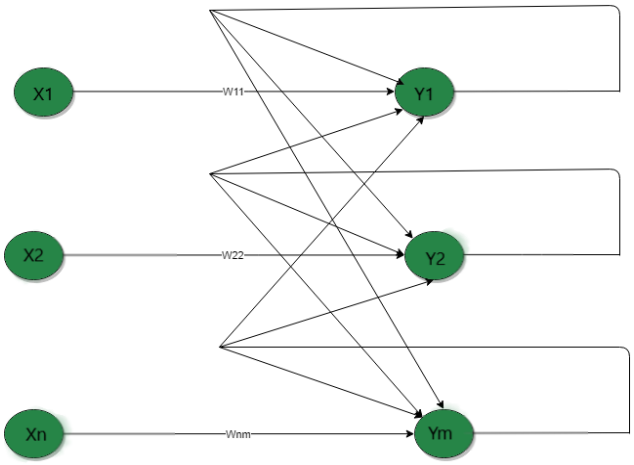
3. Single node with its own feedback



Single Node with own Feedback

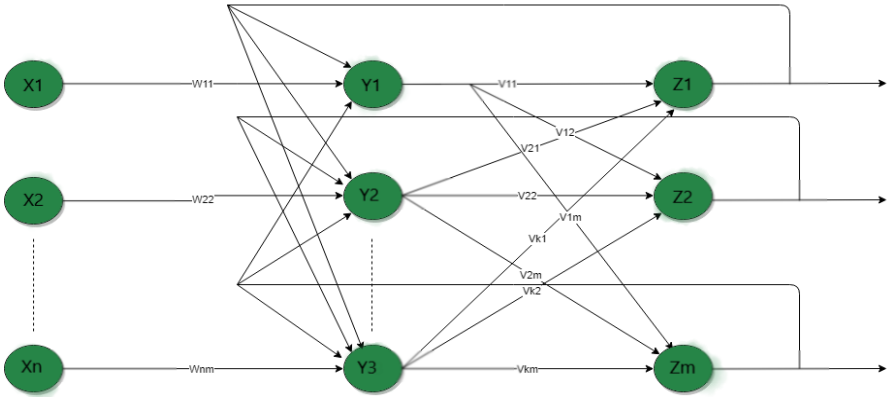
When outputs can be directed back as inputs to the same layer or preceding layer nodes, then it results in feedback networks. Recurrent networks are feedback networks with closed loops. The above figure shows a single recurrent network having a single neuron with feedback to itself.

4. Single-layer recurrent network



The above network is a single-layer network with a feedback connection in which the processing element's output can be directed back to itself or to another processing element or both. A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.

5. Multilayer recurrent network



In this type of network, processing element output can be directed to the processing element in the same layer and in the preceding layer forming a multilayer recurrent network. They perform the same task for every element of a sequence, with the output being dependent on the previous computations. Inputs are not needed at each time step. The main feature of a Recurrent Neural Network is its hidden state, which captures some information about a sequence.

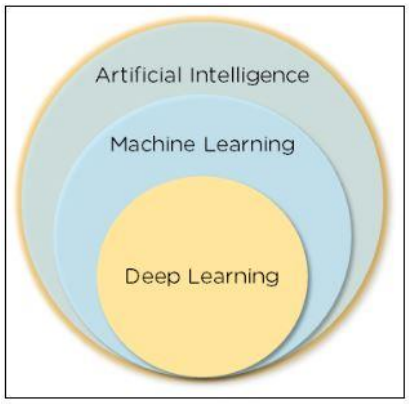
Types of Neural Networks

There are *seven* types of neural networks that can be used.

- Multilayer Perceptron (MLP): A type of feedforward neural network with three or more layers, including an input layer, one or more hidden layers, and an output layer. It uses nonlinear activation functions.
- Convolutional Neural Network (CNN): A neural network that is designed to process input data that has a grid-like structure, such as an image. It uses convolutional layers and pooling layers to extract features from the input data.
- Recursive Neural Network (RNN): A neural network that can operate on input sequences of variable length, such as text. It uses weights to make structured predictions.
- Recurrent Neural Network (RNN): A type of neural network that makes connections between the neurons in a directed cycle, allowing it to process sequential data.
- Long Short-Term Memory (LSTM): A type of RNN that is designed to overcome the vanishing gradient problem in training RNNs. It uses memory cells and gates to selectively read, write, and erase information.
- Sequence-to-Sequence (Seq2Seq): A type of neural network that uses two RNNs to map input sequences to output sequences, such as translating one language to another.
- Shallow Neural Network: A neural network with only one hidden layer, often used for simpler tasks or as a building block for larger networks

Unit 3

Deep learning is a subfield of machine learning that deals with algorithms inspired by the structure and function of the brain. Deep learning is a subset of machine learning, which is a part of artificial intelligence (AI).



Artificial intelligence is the ability of a machine to imitate intelligent human behavior. Machine learning allows a system to learn and improve from experience automatically. Deep learning is an application of machine learning that uses complex algorithms and deep neural nets to train a model

Importance of Deep Learning

- Machine learning works only with sets of structured and semi-structured data, while deep learning works with both structured and unstructured data
- Deep learning algorithms can perform complex operations efficiently, while machine learning algorithms cannot
- Machine learning algorithms use labeled sample data to extract patterns, while deep learning accepts large volumes of data as input and analyzes the input data to extract features out of an object
- The performance of machine learning algorithms decreases as the number of data increases; so to maintain the performance of the model, we need a deep learning

1. Virtual Assistants

Virtual Assistants are cloud-based applications that understand natural language voice commands and complete tasks for the user. Amazon Alexa, Cortana, Siri, and Google Assistant are typical examples of virtual assistants. They need internet-connected devices to work with their full capabilities. Each time a command is fed to the assistant, they tend to provide a better user experience based on past experiences using Deep Learning algorithms.

2. Chatbots

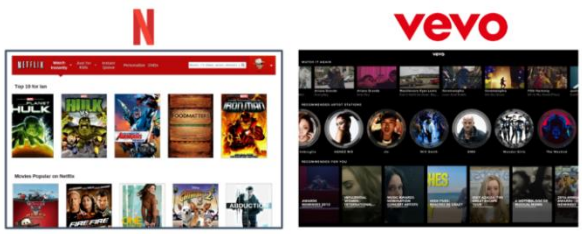
Chatbots can solve customer problems in seconds. A chatbot is an AI application to chat online via text or text-to-speech. It is capable of communicating and performing actions similar to a human. Chatbots are used a lot in customer interaction, marketing on social network sites, and instant messaging the client. It delivers automated responses to user inputs. It uses machine learning and deep learning algorithms to generate different types of reactions.

The next important deep learning application is related to Healthcare.

3. Healthcare

Deep Learning has found its application in the Healthcare sector. Computer-aided disease detection and computer-aided diagnosis have been possible using Deep Learning. It is widely used for medical research, drug discovery, and diagnosis of life-threatening diseases such as cancer and diabetic retinopathy through the process of medical imaging.

4. Entertainment



Companies such as Netflix, Amazon, YouTube, and Spotify give relevant movies, songs, and video recommendations to enhance their customer experience. This is all thanks to Deep Learning. Based on a person's browsing history, interest, and behavior, online streaming companies give suggestions to help them make product and service choices. Deep learning techniques are also used to add sound to silent movies and generate subtitles automatically.

Next, we have News Aggregation as our next important deep learning application.

5. News Aggregation and Fake News Detection

Deep Learning allows you to customize news depending on the readers' persona. You can aggregate and filter out news information as per social, geographical, and economic parameters and the individual preferences of a reader. Neural Networks help develop classifiers that can detect fake and biased news and remove it from your feed. They also warn you of possible privacy breaches.

6. Composing Music

A machine can learn the notes, structures, and patterns of music and start producing music independently. Deep Learning-based generative models such as WaveNet can be used to develop raw audio. Long Short Term Memory Network helps to generate music automatically. Music21 Python toolkit is used for computer-aided musicology. It allows us to train a system to develop music by teaching music theory fundamentals, generating music samples, and studying music.

Next in the list of deep learning applications, we have Image Coloring.

7. Image Coloring

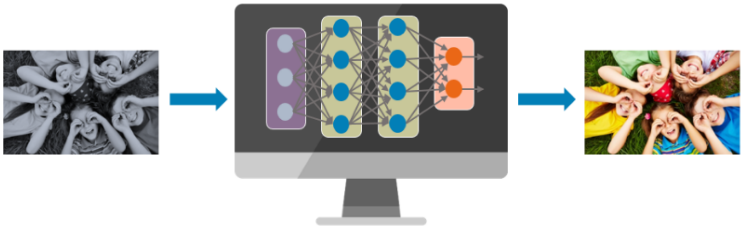


Image colorization has seen significant advancements using Deep Learning. Image colorization is taking an input of a grayscale image and then producing an output of a colorized image. ChromaGAN is an example of a picture colorization model. A generative network is framed in an adversarial model that learns to colorize by incorporating a perceptual and semantic understanding of both class distributions and color.

8. Robotics

Deep Learning is heavily used for building robots to perform human-like tasks. Robots powered by Deep Learning use real-time updates to sense obstacles in their path and pre-plan their journey instantly. It can be used to carry goods in hospitals, factories, warehouses, inventory management, manufacturing products, etc.

9. Image Captioning

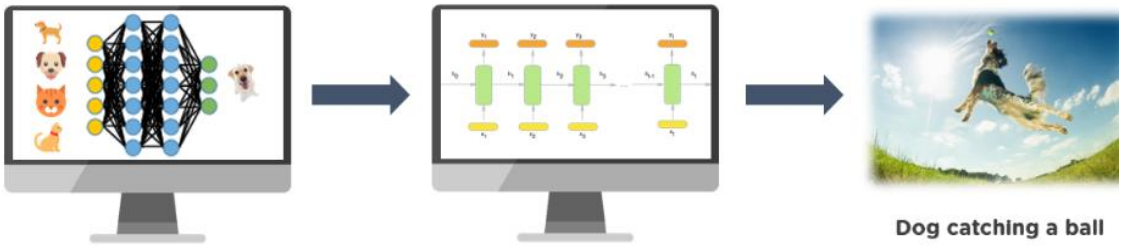


Image Captioning is the method of generating a textual description of an image. It uses computer vision to understand the image's content and a language model to turn the understanding of the image into words in the right order. A recurrent neural network such as an LSTM is used to turn the labels into a coherent sentence. Microsoft has built its caption bot where you can upload an image or the URL of any image, and it will display the textual description of the image. Another such application that suggests a perfect caption and best hashtags for a picture is Caption AI.

10. Advertising

In Advertising, Deep Learning allows optimizing a user's experience. Deep Learning helps publishers and advertisers to increase the significance of the ads and boosts the advertising campaigns.

11. Self Driving Cars

Deep Learning is the driving force behind the notion of self-driving automobiles that are autonomous. Deep Learning technologies are actually "learning machines" that learn how to act and respond using millions of data sets and training. To diversify its business infrastructure, Uber Artificial Intelligence laboratories are powering additional autonomous cars and developing self-driving cars for on-demand food delivery. Amazon, on the other hand, has delivered their merchandise using drones in select areas of the globe.

The perplexing problem about self-driving vehicles that the bulk of its designers are addressing is subjecting self-driving cars to a variety of scenarios to assure safe driving. They have operational sensors for calculating adjacent objects. Furthermore, they manoeuvre through traffic using data from its camera, sensors, geo-mapping, and sophisticated models. Tesla is one popular example.

12. Natural Language Processing

Another important field where Deep Learning is showing promising results is NLP, or Natural Language Processing. It is the procedure for allowing robots to study and comprehend human language.

However, keep in mind that human language is excruciatingly difficult for robots to understand. Machines are discouraged from correctly comprehending or creating human language not only because of the alphabet and words, but also because of context, accents, handwriting, and other factors.

Many of the challenges associated with comprehending human language are being addressed by Deep Learning-based NLP by teaching computers (Autoencoders and Distributed Representation) to provide suitable responses to linguistic inputs.

13. Visual Recognition

Just assume you're going through your old memories or photographs. You may choose to print some of these. In the lack of metadata, the only method to achieve this was through physical labour. The most you could do was order them by date, but downloaded photographs occasionally lack that metadata. Deep Learning, on the other hand, has made the job easier. Images may be sorted using it based on places recognised in pictures, faces, a mix of individuals, events, dates, and so on. To detect aspects when searching for a certain photo in a library, state-of-the-art visual recognition algorithms with various levels from basic to advanced are required.

14. Fraud Detection

Another attractive application for deep learning is fraud protection and detection; major companies in the payment system sector are already experimenting with it. PayPal, for example, uses predictive analytics technology to detect and prevent fraudulent activity. The business claimed that examining sequences of user behavior using neural networks' long short-term memory architecture increased anomaly identification by up to 10%. Sustainable fraud detection techniques are essential for every fintech firm, banking app, or insurance platform, as well as any organization that gathers and uses sensitive data. Deep learning has the ability to make fraud more predictable and hence avoidable.

15. Personalisations

Every platform is now attempting to leverage chatbots to create tailored experiences with a human touch for its users. Deep Learning is assisting e-commerce behemoths such as Amazon, E-Bay, and Alibaba in providing smooth tailored experiences such as product suggestions, customised packaging and discounts, and spotting huge income potential during the holiday season. Even in newer markets, reconnaissance is accomplished by providing goods, offers, or plans that are more likely to appeal to human psychology and contribute to growth in micro markets. Online self-service solutions are on the increase, and dependable procedures are bringing services to the internet that were previously only physically available.

16. Detecting Developmental Delay in Children

Early diagnosis of developmental impairments in children is critical since early intervention improves children's prognoses. Meanwhile, a growing body of research suggests a link between developmental impairment and motor competence, therefore motor skill is taken into account in the early diagnosis of developmental disability. However, because of the lack of professionals and time restrictions, testing motor skills in the diagnosis of the developmental problem is typically done through informal questionnaires or surveys to parents. This is progressively becoming achievable with deep learning technologies. Researchers at MIT's Computer Science and Artificial Intelligence Laboratory and the Institute of Health Professions at Massachusetts General Hospital have created a computer system that can detect language and speech impairments even before kindergarten.

17. Colourisation of Black and White images

The technique of taking grayscale photos in the form of input and creating colourized images for output that represent the semantic colours and tones of the input is known as image colourization. Given the intricacy of the work, this technique was traditionally done by hand using human labour. However, using today's Deep Learning Technology, it is now applied to objects and their context inside the shot - in order to colour the image, in the same way that a human operator would. In order to reproduce the picture with the addition of color, high-quality convolutional neural networks are utilized in supervised layers.

18. Adding Sounds to Silent Movies

In order to make a picture feel more genuine, sound effects that were not captured during production are frequently added. This is referred to as "Foley." Deep learning was used by researchers at the University of Texas to automate this procedure. They trained a neural network on 12 well-known film incidents in which filmmakers commonly used Foley effects.

19. Automatic Machine Translation

Deep learning has changed several disciplines in recent years. In response to these advancements, the field of Machine Translation has switched to the use of deep-learning neural-based methods, which have supplanted older approaches such as rule-based systems or statistical phrase-based methods. Neural MT (NMT) models can now access the whole information accessible anywhere in the source phrase and automatically learn which piece is important at which step of synthesising the output text, thanks to massive quantities of training data and unparalleled processing power. The elimination of previous independence assumptions is the primary cause for the remarkable improvement in translation quality. This resulted in neural translation closing the quality gap between human and neural translation.

20. Automatic Handwriting Generation

This Deep Learning application includes the creation of a new set of handwriting for a given corpus of a word or phrase. The handwriting is effectively presented as a series of coordinates utilised by a pen to make the samples. The link between pen movement and letter formation is discovered, and additional instances are developed.

21. Automatic Game Playing

A corpus of text is learned here, and fresh text has created word for word or character for character. Using deep learning algorithms, it is possible to learn how to spell, punctuate, and even identify the style of the text in corpus phrases. Large recurrent neural networks are typically employed to learn text production from objects in sequences of input strings. However, LSTM recurrent neural networks have lately shown remarkable success in this challenge by employing a character-based model that creates one character at a time.

22. Language Translations

Machine translation is receiving a lot of attention from technology businesses. This investment, along with recent advances in deep learning, has resulted in significant increases in translation quality. According to Google, transitioning to deep learning resulted in a 60% boost in translation accuracy over the prior phrase-based strategy employed in Google Translate. Google and Microsoft can now translate over 100 different languages with near-human accuracy in several of them.

23. Pixel Restoration

It was impossible to zoom into movies beyond their actual resolution until Deep Learning came along. Researchers at Google Brain created a Deep Learning network in 2017 to take very low-quality photos of faces and guess the person's face from them. Known as Pixel Recursive Super Resolution, this approach uses pixels to achieve super resolution. It dramatically improves photo resolution, highlighting salient characteristics just enough for personality recognition.

24. Demographic and Election Predictions

Gebru et al used 50 million Google Street View pictures to see what a Deep Learning network might accomplish with them. As usual, the outcomes were amazing. The computer learned to detect and pinpoint automobiles and their specs. It was able to identify approximately 22 million automobiles, as

well as their make, model, body style, and year. The explorations did not end there, inspired by the success story of these Deep Learning capabilities. The algorithm was shown to be capable of estimating the demographics of each location based just on the automobile makeup.

25. Deep Dreaming

DeepDream is an experiment that visualises neural network taught patterns. DeepDream, like a toddler watching clouds and attempting to decipher random forms, over-interprets and intensifies the patterns it finds in a picture.

It accomplishes this by sending an image across the network and then calculating the gradient of the picture in relation to the activations of a certain layer. The image is then altered to amplify these activations, improving the patterns perceived by the network and producing a dream-like visual. This method was named "Inceptionism" (a reference to InceptionNet, and the movie Inception).

Today, *artificial intelligence (AI)* is a thriving field with many practical applications and active research topics. We look to intelligent software to automate routine labor, understand speech or images, make diagnoses in medicine and support basic scientific research.

In the early days of artificial intelligence, the field rapidly tackled and solved problems that are intellectually difficult for human beings but relatively straight-forward for computers—problems that can be described by a list of formal, math-ematical rules. The true challenge to artificial intelligence proved to be solving the tasks that are easy for people to perform but hard for people to describe formally—problems that we solve intuitively, that feel automatic, like recognizingspoken words or faces in images.

This solution is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts. By gathering knowledge from experience, this approach avoids the need for human operators to formally specify all of the knowledge that the computer needs. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI *deep learning*.

Many of the early successes of AI took place in relatively sterile and formal environments and did not require computers to have much knowledge aboutthe world. For example, IBM's Deep Blue chess-playing system defeated world champion Garry Kasparov in 1997 (Hsu, 2002). Chess is of course a very simple world, containing only sixty-four locations and thirty-two pieces that can move in only rigidly circumscribed ways. Devising a successful chess strategy is a tremendous accomplishment, but the challenge is not due to the difficulty of describing the set of chess pieces and allowable moves to the computer. Chess can be completely described by a very brief list of completely formal rules, easily provided ahead of time by the programmer.

Ironically, abstract and formal tasks that are among the most difficult mental undertakings for a human being are among the easiest for a computer. Computers have long been able to defeat even the best human chess player, but are only recently matching some of the abilities of average human beings to recognize objectsor speech. A person's everyday life requires an immense amount of knowledge about the world. Much of this knowledge is subjective and intuitive, and therefore difficult to articulate in a formal way. Computers need to capture this same knowledge in order to behave in an intelligent way. One of the key challenges in artificial intelligence is how to get this informal knowledge into a computer.

Several artificial intelligence projects have sought to hard-code knowledge about the world in formal languages. A computer can reason about statements in these formal languages automatically using logical inference rules. This is known as the *knowledge base* approach to artificial intelligence.

The difficulties faced by systems relying on hard-coded knowledge suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data. This capability is known as *machine learning*. The introduction of machine learning allowed computers to tackle problems involving knowledge of the real world and make decisions that appear subjective. A simple machine learning algorithm called *logistic regression*

The performance of these simple machine learning algorithms depends heavily on the *representation* of the data they are given Instead, the doctor tells the system several pieces of relevant information, such as the presence or absence of a uterine scar. Each piece of information included in the representation of the patient is known as a *feature*.Logistic regression learns how each of these features of the patient correlates with various outcomes. However, it cannot influence the way that the features are defined in any way. If logistic regression was given an MRI scan of the patient,rather than the doctor's

formalized report, it would not be able to make usefulpredictions. Individual pixels in an MRI scan have negligible correlation with any complications that might occur during delivery.

This dependence on representations is a general phenomenon that appears throughout computer science and even daily life. In computer science, opera- tions such as searching a collection of data can proceed exponentially faster if the collection is structured and indexed intelligently that the choice of representation has anenormous effect on the performance of machine learning algorithms

One solution to this problem is to use machine learning to discover not only the mapping from representation to output but also the representation itself.This approach is known as *representation learning*. Learned representations often result in much better performance than can be obtained with hand-designedrepresentations. They also allow AI systems to rapidly adapt to new tasks, with minimal human intervention. A representation learning algorithm can discover a good set of features for a simple task in minutes, or a complex task in hours tomonths. Manually designing features for a complex task requires a great deal of human time and effort; it can take decades for an entire community of researchers.

The quintessential example of a representation learning algorithm is the *au- toencoder*. An autoencoder is the combination of an *encoder* function that converts the input data into a different representation, and a *decoder* function that converts the new representation back into the original format. Autoencoders are trained to preserve as much information as possible when an input is run through the encoder and then the decoder, but are also trained to make the new representation have various nice properties. Different kinds of autoencoders aim to achieve different kinds of properties.

When analyzing a speech recording, the factors of variation include the speaker'sage, their sex, their accent and the words that they are speaking. When analyzing an image of a car, the factors of variation include the position of the car, its color,and the angle and brightness of the sun.

A major source of difficulty in many real-world artificial intelligence applicationsis that many of the factors of variation influence every single piece of data we are able to observe. The individual pixels in an image of a red car might be very closeto black at night

Deep learning solves this central problem in representation learning by introduc- ing representations that are expressed in terms of other, simpler representations. Deep

learning allows the computer to build complex concepts out of simpler con-cepts. Fig. 1.2 shows how a deep learning system can represent the concept of an image of a person by combining simpler concepts, such as corners and contours, which are in turn defined in terms of edges.

The quintessential example of a deep learning model is the feedforward deep network or *multilayer perceptron* (MLP). A multilayer perceptron is just a mathe-matical function mapping some set of input values to output values. The function is formed by composing many simpler functions. We can think of each application of a different mathematical function as providing a new representation of the input.

The idea of learning the right representation for the data provides one perspec-tive on deep learning. Another perspective on deep learning is that depth allows the computer to learn a multi-step computer program. Each layer of the representation can be thought of as the state of the computer's memory after executing another set of instructions in parallel. Networks with greater depth can execute more instructions in sequence. Sequential instructions offer great power because later instructions can refer back to the results of earlier instructions. According to this

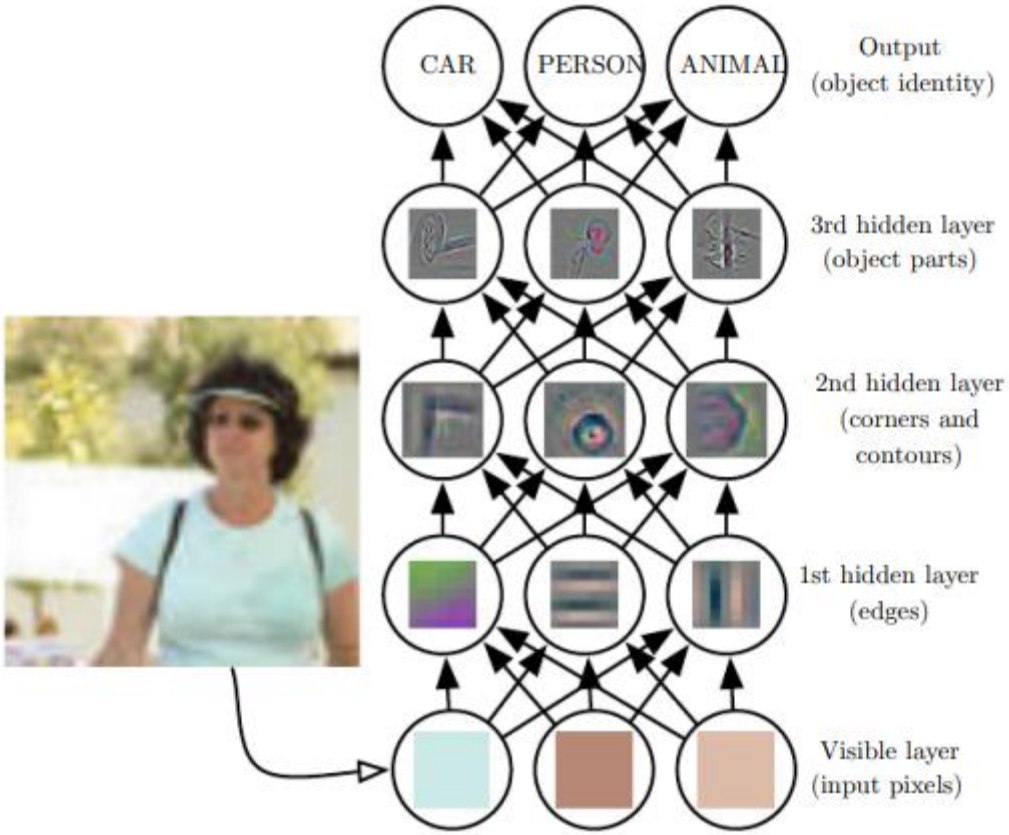


Figure 1.2: Illustration of a deep learning model. It is difficult for a computer to understand the meaning of raw sensory input data, such as this image represented as a collection of pixel values. The function mapping from a set of pixels to an object identity is very complicated. Learning or evaluating this mapping seems insurmountable if tackled directly. Deep learning resolves this difficulty by breaking the desired complicated mapping into a series of nested simple mappings, each described by a different layer of the model. Theinput is presented at the *visible layer*, so named because it contains the variables that we are able to observe. Then a series of *hidden layers* extracts increasingly abstract features from the image. These layers are called "hidden" because their values are not given in the data; instead the model must determine which concepts are useful for

explaining the relationships in the observed data. The images here are visualizations of the kind of feature represented by each hidden unit. Given the pixels, the first layer can easily identify edges, by comparing the brightness of neighboring pixels. Given the first hidden layer's description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges. Given the second hidden layer's description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by finding specific collections of contours and corners. Finally, this description of the image in terms of the object parts it contains can be used to recognize the objects present in the image.

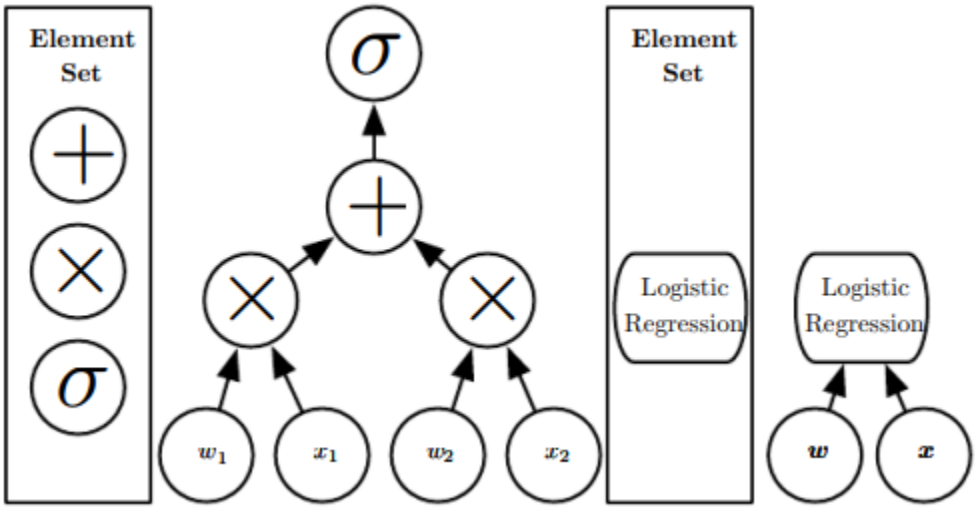


Figure 1.3: Illustration of computational graphs mapping an input to an output where each node performs an operation. Depth is the length of the longest path from input to output but depends on the definition of what constitutes a possible computational step. The computation depicted in these graphs is the output of a logistic regression model, $\sigma(\boldsymbol{w}^T \boldsymbol{x})$, where σ is the logistic sigmoid function. If we use addition, multiplication and logistic sigmoids as the elements of our computer language, then this model has depth three. If we view logistic regression as an element itself, then this model has depth one.

view of deep learning, not all of the information in a layer's activations necessarily encodes factors of variation that explain the input. The representation also stores state information that helps to execute a program that can make sense of the input.This state

information could be analogous to a counter or pointer in a traditional computer program. It has nothing to do with the content of the input specifically,but it helps the model to organize its processing.

There are two main ways of measuring the depth of a model. The first view is based on the number of sequential instructions that must be executed to evaluate the architecture. We can think of this as the length of the longest path through a flow chart that describes how to compute each of the model's outputs given its inputs. Just as two equivalent computer programs will have different lengths depending on which language the program is written in, the same function may be drawn as a flowchart with different depths depending on which functions we allow to be used as individual steps in the flowchart. Fig. 1.3 illustrates how this choice of language can give two different measurements for the same architecture.

Another approach, used by deep probabilistic models, regards the depth of a model as being not the depth of the computational graph but the depth of the graph describing how concepts are related to each other. In this case, the depth of the flowchart of the computations needed to compute the representation of

---

each concept may be much deeper than the graph of the concepts themselves. This is because the system's understanding of the simpler concepts can be refinedgiven information about the more complex concepts. For example, an AI system observing an image of a face with one eye in shadow may initially only see one eye.After detecting that a face is present, it can then infer that a second eye is probably present as well. In this case, the graph of concepts only includes two layers—a layer for eyes and a layer for faces—but the graph of computations includes 2n layers if we refine our estimate of each concept given the other n times.

Because it is not always clear which of these two views—the depth of the computational graph, or the depth of the probabilistic modeling graph—is most relevant, and because different people choose different sets of smallest elements from which to construct their graphs, there is no single correct value for the depth of an architecture, just as there is no single correct value for the length of a computer program. Nor is there a consensus about how much depth a model requires to qualify as "deep." However, deep learning can safely be regarded as the study of models that either involve a greater amount of composition of learned functions or learned concepts than traditional machine learning does.

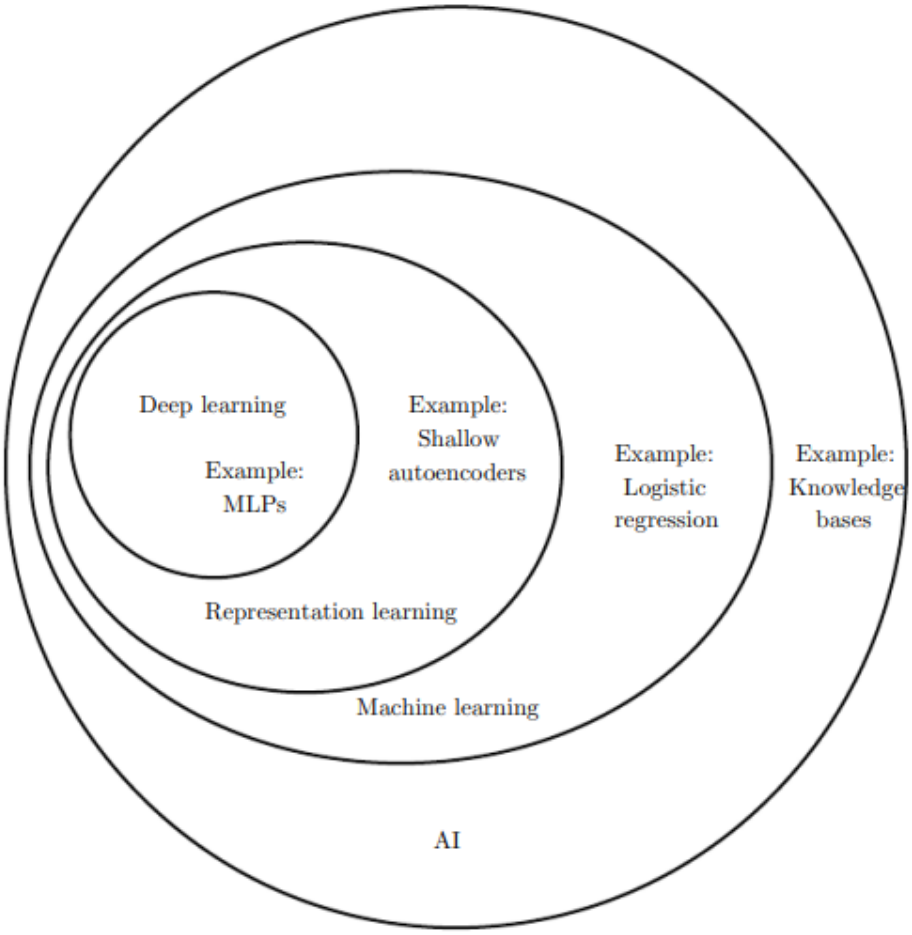


Figure 1.4: A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI. Each section of the Venn diagram includes an example of an AI technology.

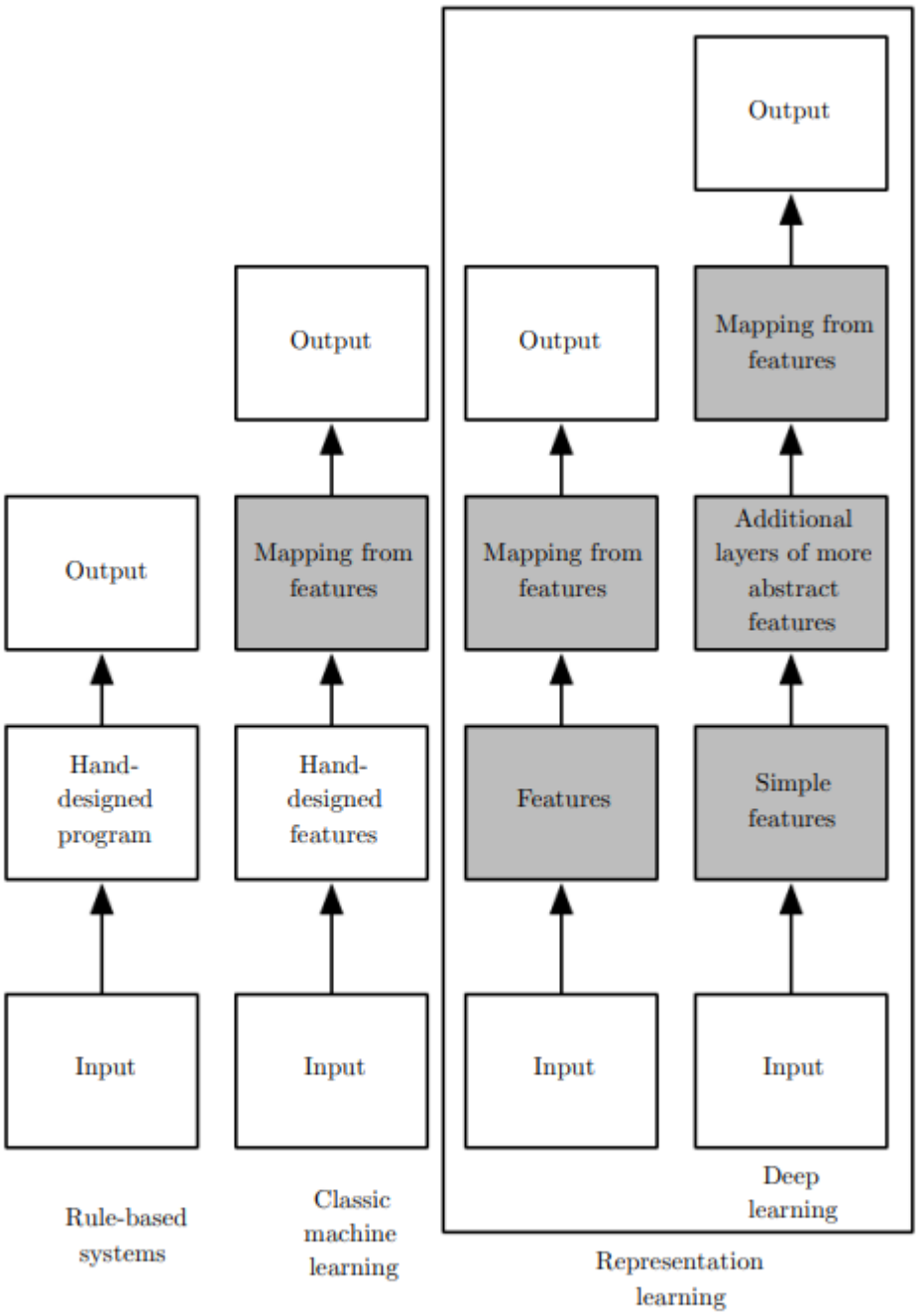


Figure 1.5: Flowcharts showing how the different parts of an AI system relate to each other within different AI disciplines. Shaded boxes indicate components that are able to learn from data.

of machine learning allowed computers to tackle problems involving knowledge of the real world and make decisions that appear subjective. A simple machine learning algorithm called *logistic regression* can determine whether to recommend cesarean delivery (Mor-Yosef *et al.*, 1990). A simple machine learning algorithm called *naive Bayes* can separate legitimate e-mail from spam e-mail.

The performance of these simple machine learning algorithms depends heavily on the *representation* of the data they are given. For example, when logisticregression is used to recommend cesarean delivery, the AI system does not examine the patient directly. Instead, the doctor tells the system several pieces of relevant information, such as the presence or absence of a uterine scar. Each piece of information included in the representation of the patient is known as a *feature*.Logistic regression learns how each of these features of the patient correlates with various outcomes. However, it cannot influence the way that the features are defined in any way. If logistic regression was given an MRI scan of the patient,rather than the doctor's formalized report, it would not be able to make usefulpredictions. Individual pixels in an MRI scan have negligible correlation with any complications that might occur during delivery.

This dependence on representations is a general phenomenon that appears throughout computer science and even daily life. In computer science, opera- tions such as searching a collection of data can proceed exponentially faster if the collection is structured and indexed intelligently. People can easily perform arithmetic on Arabic numerals, but find arithmetic on Roman numerals much more time-consuming. It is not surprising that the choice of representation has anenormous effect on the performance of machine learning algorithms. For a simple visual example, see Fig. 1.1.

Many artificial intelligence tasks can be solved by designing the right set of features to extract for that task, then providing these features to a simple machine learning algorithm. For example, a useful feature for speaker identification from sound is an estimate of the size of speaker's vocal tract. It therefore gives a strongclue as to whether the speaker is a man, woman, or child.

However, for many tasks, it is difficult to know what features should be extracted. For example, suppose that we would like to write a program to detect cars in photographs. We know that cars have wheels, so we might like to use the presence of a wheel as a feature. Unfortunately, it is difficult to describe exactly what a wheel looks like in terms of pixel values. A wheel has a simple geometric shape but its image may be complicated by shadows falling on the wheel, the sun glaring off the metal parts of the wheel, the fender of the car or an object in the foreground obscuring part of the wheel, and so on.

One solution to this problem is to use machine learning to discover not only the mapping from representation to output but also the representation itself. This approach is known as

representation learning. Learned representations often result in much better performance than can be obtained with hand-designed representations. They also allow AI systems to rapidly adapt to new tasks, with minimal human intervention. A representation learning algorithm can discover a good set of features for a simple task in minutes, or a complex task in hours to months. Manually designing features for a complex task requires a great deal of human time and effort; it can take decades for an entire community of researchers.

Historical Trends in Deep Learning

It is easiest to understand deep learning with some historical context. Rather thanproviding a detailed history of deep learning, we identify a few key trends:

- Deep learning has had a long and rich history, but has gone by many names reflecting different philosophical viewpoints, and has waxed and waned in popularity.
- Deep learning has become more useful as the amount of available training data has increased.
- Deep learning models have grown in size over time as computer hardware and software infrastructure for deep learning has improved.

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

The Many Names and Changing Fortunes of Neural Net-works

1 Deep learning dates back to the 1940s. Deep learning only *appears* to be new, because it was relatively unpopular for several years preceding its current popularity, and because it has gone through many different names, and has only recently become called “deep learning.” The field has been rebranded many times, reflecting the influence of different researchers and different perspectives.

2 However, some basic context is useful for understanding deep learning. Broadly speaking, there have been three waves of development of deep learning: deep learn- ing known as *cybernetics* in the 1940s–1960s, deep learning known as *connectionism* in the 1980s–1990s, and the current resurgence under the name deep learning beginning in 2006. This is quantitatively illustrated in Fig. 1.7.

3 Some of the earliest learning algorithms we recognize today were intended to be computational models of biological learning, i.e. models of how learning happens or could happen in the brain. As a result, one of the names that deep learning has gone by is *artificial neural networks* (ANNs). The corresponding perspective on deep learning models is that they are

engineered systems inspired by the biological brain (whether the human brain or the brain of another animal).

4 While the kinds of neural networks used for machine learning have sometimes been used to understand brain function (Hinton and Shallice, 1991), they are generally not designed to be realistic models of biological function. The neural perspective on deep learning is motivated by two main ideas. One idea is that the brain provides a proof by example that intelligent behavior is possible, and a conceptually straightforward path to building intelligence is to reverse engineer the computational principles behind the brain and duplicate its functionality. Another perspective is that it would be deeply interesting to understand the brain and the principles that underlie human intelligence, so machine learning models that shedlight on these basic scientific questions are useful apart from their ability to solve engineering applications.

5 The modern term "deep learning" goes beyond the neuroscientific perspectiveon the current breed of machine learning models. It appeals to a more general principle of learning *multiple levels of composition*, which can be applied in machine learning frameworks that are not necessarily neurally inspired.

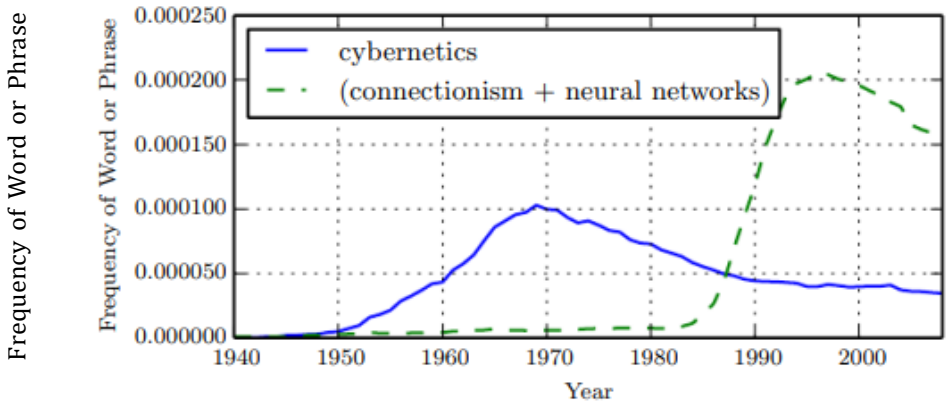


Figure 1.7: The figure shows two of the three historical waves of artificial neural nets research, as measured by the frequency of the phrases "cybernetics" and "connectionism" or "neural networks" according to Google Books (the third wave is too recent to appear). The first wave started with cybernetics in the 1940s–1960s, with the development of theories of biological learning (McCulloch and Pitts, 1943; Hebb, 1949) and implementations ofthe first models such as the perceptron (Rosenblatt, 1958) allowing the training of a single neuron. The second wave started with the connectionist approach of the 1980–1995 period, with back-propagation (Rumelhart *et al.*, 1986a) to train a neural network with one or two hidden layers. The current and third wave, deep learning, started around 2006 (Hinton*et al.*, 2006; Bengio *et al.*, 2007; Ranzato *et al.*, 2007a), and is

just now appearing in book form as of 2016. The other two waves similarly appeared in book form much later than the corresponding scientific activity occurred.

The earliest predecessors of modern deep learning were simple linear models motivated from a neuroscientific perspective. These models were designed to take a set of n input values $x_1, \ldots, x_n$ and associate them with an output y. These models would learn a set of weights $w_1, \ldots, w_n$ and compute their output $f(\boldsymbol{x}, \boldsymbol{w}) = x_1 w_1 + \cdots + x_n w_n$. This first wave of neural networks research was known as *cybernetics*, as illustrated in Fig. 1.7.

The McCulloch-Pitts Neuron (McCulloch and Pitts, 1943) was an early modelof brain function. This linear model could recognize two different categories of inputs by testing whether $f(\boldsymbol{x}, \boldsymbol{w})$ is positive or negative. Of course, for the model to correspond to the desired definition of the categories, the weights needed to be set correctly. These weights could be set by the human operator. In the 1950s, the perceptron (Rosenblatt, 1958, 1962) became the first model that could learn the weights defining the categories given examples of inputs from each category. The *adaptive linear element* (ADALINE), which dates from about the same time, simply returned the value of $f(\boldsymbol{x})$ itself to predict a real number (Widrow and Hoff, 1960), and could also learn to predict these numbers from data.

It is worth noting that the effort to understand how the brain works on an algorithmic level is alive and well. This endeavor is primarily known as "computational neuroscience" and is a separate field of study from deep learning. It is common for researchers to move back and forth between both fields. The field of deep learning is primarily concerned with how to build computer systems that are able to successfully solve tasks requiring intelligence, while the field of computational neuroscience is primarily concerned with building more accurate models of how the brain actually works.

In the 1980s, the second wave of neural network research emerged in great part via a movement called *connectionism* or *parallel distributed processing* (Rumelhart *et al.*, 1986c; McClelland *et al.*, 1995). Connectionism arose in the context of

cognitive science. Cognitive science is an interdisciplinary approach to understand- ing the mind, combining multiple different levels of analysis. During the early 1980s, most cognitive scientists studied models of symbolic reasoning. Despite their popularity, symbolic models were difficult to explain in terms of how the brain could actually implement them using neurons. The connectionists began to study models of cognition that could actually be grounded in neural implementations (Touretzky and Minton, 1985), reviving many ideas dating back to the work of psychologist Donald Hebb in the 1940s (Hebb, 1949).

The central idea in connectionism is that a large number of simple computational units can achieve intelligent behavior when networked together. This insight applies equally to neurons in biological nervous systems and to hidden units in computational models.

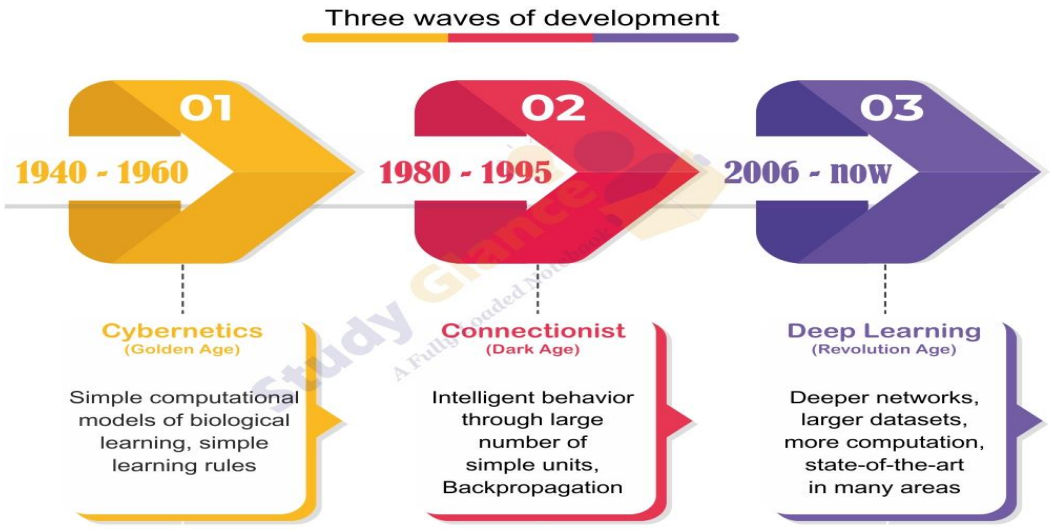
Several key concepts arose during the connectionism movement of the 1980s that remain central to today's deep learning.

One of these concepts is that of *distributed representation* (Hinton *et al.*, 1986). This is the idea that each input to a system should be represented by many features, and each feature should be involved in the representation of many possible inputs.

For example, suppose we have a vision system that can recognize cars, trucks, and birds and these objects can each be red, green, or blue. One way of representing these inputs would be to have a separate neuron or hidden unit that activates for each of the nine possible combinations: red truck, red car, red bird, green truck, and so on. This requires nine different neurons, and each neuron must independently learn the concept of color and object identity. One way to improve on this situation is to use a distributed representation, with three neurons describing the color and three neurons describing the object identity. This requires only six neurons total instead of nine, and the neuron describing redness is able to learn about redness from images of cars, trucks and birds, not only from images of one specific category of objects.

This third wave of popularity of neural networks continues to the time of this writing, though the focus of deep learning research has changed dramatically within the time of this wave. The third wave began with a focus on new unsupervised learning techniques and the ability of deep models to generalize well from small datasets, but today there is more interest in much older supervised learning algorithms and the ability of deep models to leverage large labeled datasets.

Deep Learning have been three waves of development: The first wave started with cybernetics in the 1940s-1960s, with the development of theories of biological learning and implementations of the first models such as the perceptron allowing the training of a single neuron. The second wave started with the connectionist approach of the 1980-1995 period, with back-propagation to train a neural network with one or two hidden layers. The current and third wave, deep learning, started around 2006.



Increasing Dataset Sizes

One may wonder why deep learning has only recently become recognized as a crucial technology though the first experiments with artificial neural networks were conducted in the 1950s. Deep learning has been successfully used in commercial applications since the 1990s, but was often regarded as being more of an art than a technology and something that only an expert could use, until recently. It is true that some skill is required to get good performance from a deep learning algorithm. Fortunately, the amount of skill required reduces as the amount of training data increases. The learning algorithms reaching human performance on complex tasks today are nearly identical to the learning algorithms that struggled to solve toy problems in the 1980s, though the models we train with these algorithms have undergone changes that simplify the training of very deep architectures. The most important new development is that today we can provide these algorithms with
the resources they need to succeed

The age of "Big Data" has made machine learning much easier because the key burden of statistical estimation—generalizing well to new data after observing only a small amount of data—has been considerably lightened. As of 2016, a rough rule of thumb is that a supervised deep learning algorithm will generally achieve acceptable performance with around 5,000 labeled examples per category, and will match or exceed human performance when trained with a dataset containing at least 10 million labeled examples. Working successfully with datasets smaller than this is an important research area, focusing in particular on how we can take advantage of large quantities of unlabeled examples, with unsupervised or semi-supervised learning.

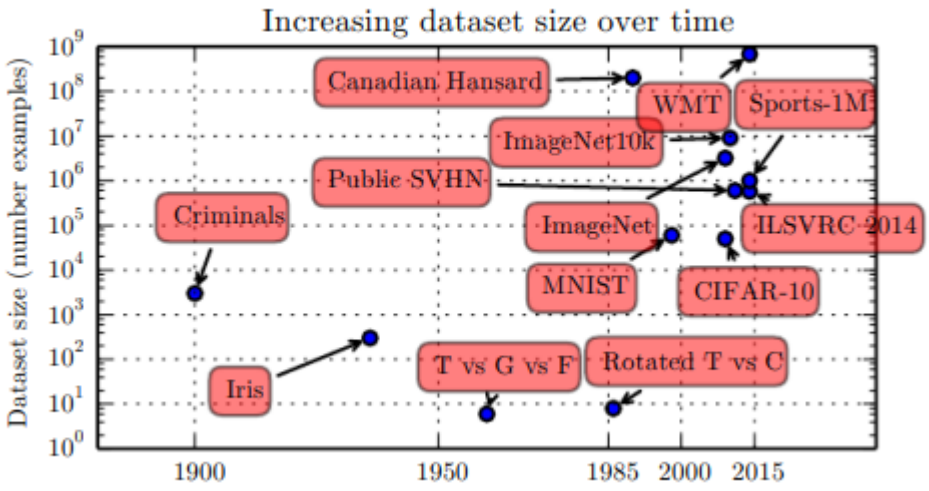


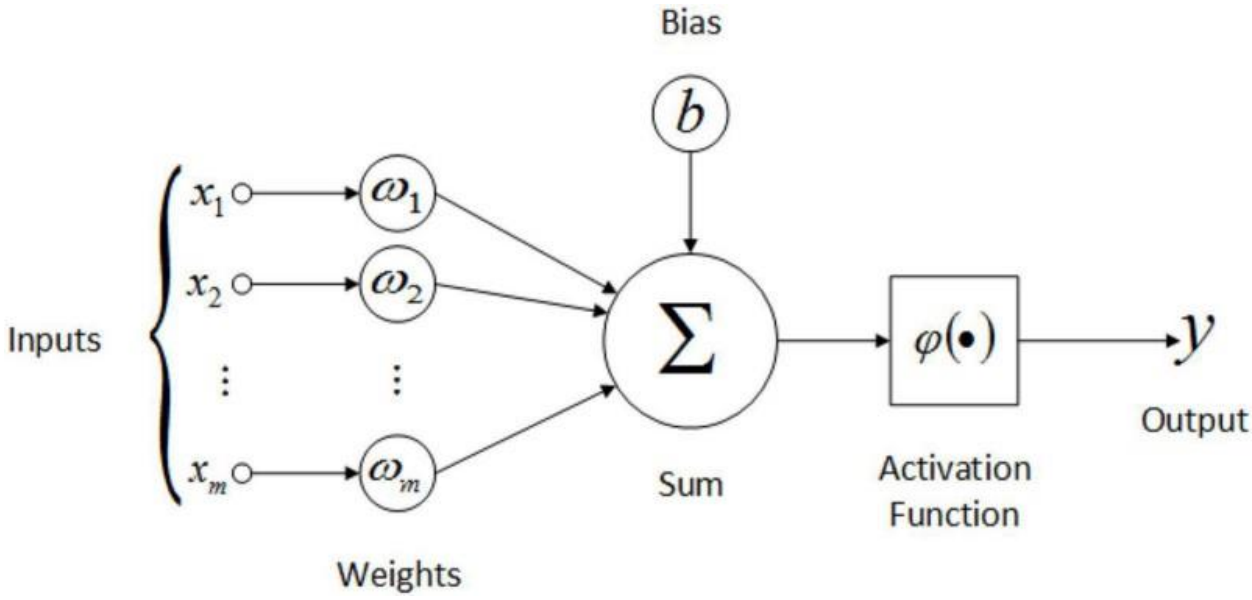
Figure 1.8: Dataset sizes have increased greatly over time. In the early 1900s, statisticians studied datasets using hundreds or thousands of manually compiled measurements (Garson, 1900; Gosset, 1908; Anderson, 1935; Fisher, 1936). In the 1950s through 1980s, the pioneers of biologically inspired machine learning often worked with small, synthetic datasets, such as low-resolution bitmaps of letters, that were designed to incur low computational cost and demonstrate that neural networks were able to learn specific kinds of functions (Widrow and Hoff, 1960; Rumelhart et al., 1986b). In the 1980s and 1990s, machine learning became more statistical in nature and began to leverage larger datasets containing tens of thousands of examples such as the MNIST dataset (shown in Fig. 1.9) of scans of handwritten numbers (LeCun et al., 1998b). In the first decade of the 2000s, more sophisticated datasets of this same size, such as the CIFAR-10 dataset (Krizhevsky and Hinton, 2009) continued to be produced.

Toward the end of that decade and throughout the first half of the 2010s, significantly larger datasets, containing hundreds of thousands to tens of millions of examples, completely changed what was possible with deep learning. These datasets included the public Street View House Numbers dataset (Netzer et al., 2011), various versions of the ImageNet dataset (Deng et al., 2009, 2010a; Russakovsky et al., 2014a), and the Sports-1M dataset (Karpathy et al., 2014). At the top of the graph, we see that datasets of translated sentences, such as IBM's dataset constructed from the Canadian Hansard (Brown et al., 1990) and the WMT 2014 English to French dataset (Schwenk, 2014) are typically far ahead of other dataset sizes.

Feed Forward Network

its most basic form, a Feed-Forward Neural Network is a single layer perceptron. A sequence of inputs enter the layer and are multiplied by the weights in this model. The weighted input values are then summed together to form a total. If the sum of the values is more than a predetermined threshold, which is normally set at zero, the output value is usually 1, and if the sum is less than the threshold, the output value is usually -1.

The single-layer perceptron is a popular feed-forward neural network model that is frequently used for classification. Single-layer perceptrons can also contain machine learning features.



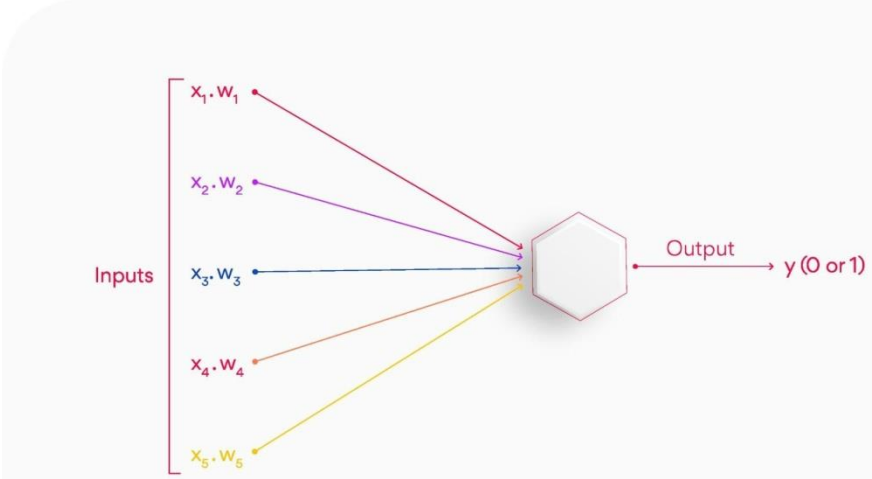
The neural network can compare the outputs of its nodes with the desired values using a property known as the delta rule, allowing the network to alter its weights through training to create more accurate output values. This training and learning procedure results in gradient descent. The technique of updating weights in multi-layered perceptrons is virtually the same, however, the process is referred to as back-propagation.

Feed forward neural networks are artificial neural networks in which nodes do not form loops. This type of neural network is also known as a multi-layer neural network as all information is only passed forward.

During data flow, input nodes receive data, which travel through hidden layers, and exit output nodes. No links exist in the network that could get used to by sending information back from the output node.

A feed forward neural network approximates functions in the following way:

- An algorithm calculates classifiers by using the formula y = f* (x).
- Input x is therefore assigned to category y.
- According to the feed forward model, y = f (x; θ). This value determines the closest approximation of the function.



when the feed forward neural network gets simplified, it can appear as a single layer perceptron.

This model multiplies inputs with weights as they enter the layer. Afterward, the weighted input values get added together to get the sum. As long as the sum of the values rises above a certain threshold, set at zero, the output value is usually 1, while if it falls below the threshold, it is usually -1.

As a feed forward neural network model, the single-layer perceptron often gets used for classification. Machine learning can also get integrated into single-layer perceptrons. Through training, neural networks can adjust their weights based on a property called the delta rule, which helps them compare their outputs with the intended values.

As a result of training and learning, gradient descent occurs. Similarly, multi-layered perceptrons update their weights. But, this process gets known as back-propagation. If this is the case, the network's hidden layers will get adjusted according to the output values produced by the final layer.

> *Deep feedforward networks*, also often called *feedforward neural networks*, or *multi-layer perceptrons* (*MLPs*), are the quintessential deep learning models. The goal of a feedforward network is to approximate some function f^*. For example, for a classifier, $y = f^*(\boldsymbol{x})$ maps an input $\boldsymbol{x}$ to a category y. A feedforward network
>
> defines a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\vartheta})$ and learns the value of the parameters $\boldsymbol{\vartheta}$ that result in the best function approximation.

These models are called *feedforward* because information flows through the function being evaluated from $\boldsymbol{x}$, through the intermediate computations used to define f, and finally to the output $\boldsymbol{y}$. There are no *feedback* connections in which outputs of the model are fed back into itself. When feedforward neural networks are extended to include feedback connections, they are called *recurrent neural networks*

Layers of feed forward neural network

- **input layer:**

The neurons of this layer receive input and pass it on to the other layers of the network. Feature or attribute numbers in the dataset must match the number of neurons in the input layer.

- **Output layer:**

According to the type of model getting built, this layer represents the forecasted feature.

- **Hidden layer:**

Input and output layers get separated by hidden layers. Depending on the type of model, there may be several hidden layers.

There are several neurons in hidden layers that transform the input before actually transferring it to the next layer. This network gets constantly updated with weights in order to make it easier to predict.

- **Neuron weights:**

Neurons get connected by a weight, which measures their strength or magnitude. Similar to linear regression coefficients, input weights can also get compared.

Weight is normally between 0 and 1, with a value between 0 and 1.

- **Neurons:**

Artificial neurons get used in feed forward networks, which later get adapted from biological neurons. A neural network consists of artificial neurons.

Neurons function in two ways: first, they create weighted input sums, and second, they activate the sums to make them normal.

Activation functions can either be linear or nonlinear. Neurons have weights based on their inputs. During the learning phase, the network studies these weights.

- **Activation Function:**

Neurons are responsible for making decisions in this area.

According to the activation function, the neurons determine whether to make a linear or nonlinear decision. Since it passes through so many layers, it prevents the cascading effect from increasing neuron outputs.

An activation function can be classified into three major categories: sigmoid, Tanh, and Rectified Linear Unit (ReLu).

- **Sigmoid(most Generally Used)**

Input values between 0 and 1 get mapped to the output valueso better understand how feedforward neural networks function, let's solve a simple problem — predicting if it's raining or not when given three inputs.

- x1 - day/night
- x2 - temperature
- x3 - month

Let's assume the threshold value to be 20, and if the output is higher than 20 then it will be raining, otherwise it's a sunny day. Given a data tuple with inputs (x1, x2, x3) as (0, 12, 11), initial weights of the feedforward network (w1, w2, w3) as (0.1, 1, 1) and biases as (1, 0, 0).

Here's how the neural network computes the data in three simple steps:

1. Multiplication of weights and inputs: The input is multiplied by the assigned weight values, which this case would be the following:

$$(x1^* w1) = (0 * 0.1) = 0$$

$$(x2^* w2) = (1 * 12) = 12$$

$$(x3^* w3) = (11 * 1) = 11$$

2. Adding the biases: In the next step, the product found in the previous step is added to their respective biases. The modified inputs are then summed up to a single value.

$$(x1^* w1) + b1 = 0 + 1$$

$$(x2^* w2) + b2 = 12 + 0$$

$$(x3^* w3) + b3 = 11 + 0$$

$$\text{weighted\_sum} = (x1^* w1) + b1 + (x2^* w2) + b2 + (x3^* w3) + b3 = 23$$

3. Activation: An activation function is the mapping of summed weighted input to the output of the neuron. It is called an activation/transfer function because it governs the inception at which the neuron is activated and the strength of the output signal.

4. Output signal: Finally, the weighted sum obtained is turned into an output signal by feeding the weighted sum into an activation function (also called transfer function). Since the weighted sum in our example is greater than 20, the perceptron predicts it to be a rainy day.

Calculating the Loss

In simple terms, a loss function quantifies how “good” or “bad” a given model is in classifying the input data. In most learning networks, the loss is calculated as the difference between the actual output and the predicted output.

Mathematically:

$$\text{loss} = y_{\{predicted\}} - y_{\{original\}}$$

The function that is used to compute this error is known as loss function J(.). Different loss functions will return different errors for the same prediction, having a considerable effect on the performance of the model.

Gradient Descent

Gradient descent is the most popular optimization technique for feedforward neural networks. The term “gradient” refers to the quantity change of output obtained from a neural network when the inputs change a little. Technically, it measures the updated weights concerning the change in error. The gradient can also be defined as the slope of a function. The higher the angle, the steeper the slope and the faster a model can learn.

delta training rule, consider the task of training a threshold perception.That is, a linear unit for which the output ***O*** is given by

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

$$O(\vec{x}) = (\vec{w} \cdot \vec{x}) \qquad \text{equ. (1)}$$

To derive a weight learning rule for linear units, specify a measure for the ***training error***

of ahypothesis (weight vector), relative to the training examples.

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \qquad \text{equ. (2)}$$

Where,

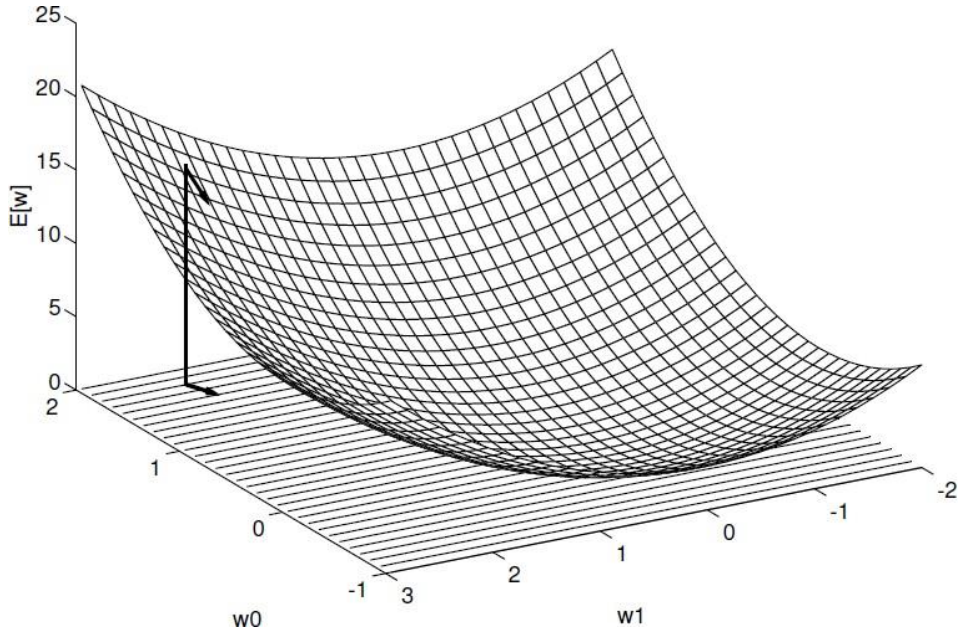
- D is the set of training examples,
- t_d is the target output for training example d,
- o_d is the output of the linear unit for training example d
- E ($\vec{w}$) is simply half the squared difference between the target output t_d and the linear unit output o_d, summed over all training examples.

Gradient Descent Error Estimation in 3 dimensional plane

A gradient simply measures the change in all weights with regard to the change in error. You can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops learning.

Visualizing the Hypothesis Space

- To understand the gradient descent algorithm, it is helpful to visualize the entire hypothesis space of possible weight vectors and their associated E values as shown in below figure.
- Here the axes w_0 and w_l represent possible values for the two weights of a simple linearunit. The w_0, w_l plane therefore represents the entire hypothesis space.
- The vertical axis indicates the error E relative to some fixed set of training examples.
- The arrow shows the negated gradient at one particular point, indicating the direction inthe w_0, w_l plane producing steepest descent along the error surface.
- The error surface shown in the figure thus summarizes the desirability of every weight vector in the hypothesis space



- Given the way in which we chose to define E, for linear units this error surface mustalways be parabolic with a single global minimum.

Gradient descent search determines a weight vector that minimizes E by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps.
At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface depicted in above figure. This process continues until the global minimum error is reached.

Types of Gradient Descent

There are three popular types of gradient descent that mainly differ in the amount of data they use:

BATCH GRADIENT DESCENT

Batch gradient descent, also called vanilla gradient descent, calculates the error for each example within the training dataset, but only after all training examples have been evaluated does the model get updated. This whole process is like a cycle and it's called a training epoch.

Some advantages of batch gradient descent are its computational efficient, it produces a stable error gradient and a stable convergence. Some disadvantages are the stable error gradient can sometimes result in a state of convergence that isn’t the best the model can achieve. It also requires the entire training dataset be in memory and available to the algorithm.

STOCHASTIC GRADIENT DESCENT

By contrast, stochastic gradient descent (SGD) does this for each training example within the dataset, meaning it updates the parameters for whole training data set example one by one. Depending on the problem, this can make SGD faster than batch gradient descent.

The frequent updates, however, are more computationally expensive than the batch gradient descent approach.

Additionally, the frequency of those updates can result in noisy gradients, which may cause the error rate to jump around instead of slowly decreasing.

MINI-BATCH GRADIENT DESCENT

Mini-batch gradient descent is the go-to method since it's a combination of the concepts of SGD and batch gradient descent. It simply splits the training dataset into small batches and performs an update for each of those batches.

This creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

Issues in Gradient Descent Algorithm

- Can veer off in the wrong direction due to frequent updates.
- Frequent updates are computationally expensive in process due to using all resources for processing one training sample at a time.

Backpropagation

The predicted value of the network is compared to the expected output, and an error is calculated using a function. This error is then propagated back within the whole network, one layer at a time, and the weights are updated according to the value that they contributed to the error. This clever bit of math is called a backpropagation algorithm. The process is repeated for all of the examples in the training data. One round of updating the network for the entire training dataset is called an epoch. A network may be trained for tens, hundreds or many thousands of epochs.

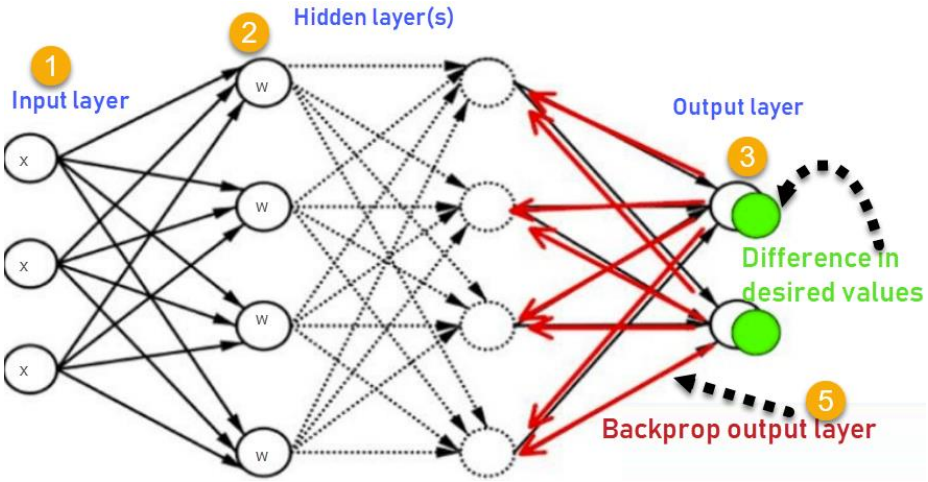
How does back propagation work?

Let us take a look at how back propagation works. It has four layers: input layer, hidden layer, hidden layer II and final output layer.

So, the main three layers are:

1. Input layer
2. Hidden layer
3. Output layer

Each layer has its own way of working and its own way to take action such that we are able to get the desired results and correlate these scenarios to our conditions. Let us discuss other details needed to help summarizing this algorithm.



This image summarizes the functioning of the backpropagation approach.

1. Input layer receives x
2. Input is modeled using weights w
3. Each hidden layer calculates the output and data is ready at the output layer
4. Difference between actual output and desired output is known as the error
5. Go back to the hidden layers and adjust the weights so that this error is reduced in future runs

This process is repeated till we get the desired output. The training phase is done with supervision. Once the model is stable, it is used in production.

Why do we need back propagation?

Back propagation has many advantages, some of the important ones are listed below-

- Back propagation is fast, simple and easy to implement
- There are no parameters to be tuned
- Prior knowledge about the network is not needed thus becoming a flexible method
- This approach works very well in most cases

Disadvantages of using Backpropagation

- The actual performance of backpropagation on a specific problem is dependent on the input data.

Back propagation algorithm can be quite sensitive to noisy data

BACKPROPAGATION AND ITS DIFFERENTIAL ALGORITHMS:

Backpropagation is implemented in deep learning frameworks like Tensorflow, Torch, Theano, etc., by using computational graphs. More significantly, understanding back propagation on computational graphs combines several different algorithms and its variations such as backprop through time and backprop with shared weights. Once everything is converted into a computational graph, they are still the same algorithm – just back propagation on computational graphs.

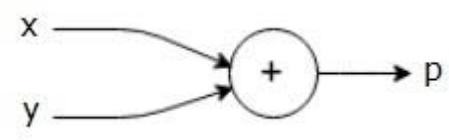
What is Computational Graph

A computational graph is defined as a directed graph where the nodes correspond to mathematical operations. Computational graphs are a way of expressing and evaluating a mathematical expression.

For example, here is a simple mathematical equation –

$$p=x+y$$

We can draw a computational graph of the above equation as follows.

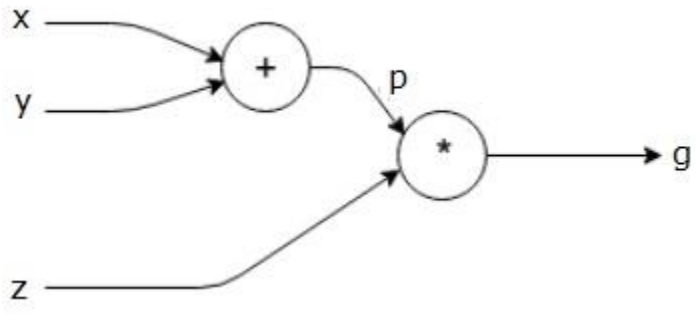


The above computational graph has an addition node (node with "+" sign) with two input variables x and y and one output q.

Let us take another example, slightly more complex. We have the following equation.

$$g=(x+y)*z$$

The above equation is represented by the following computational graph.



Computational Graphs and Backpropagation

Computational graphs and backpropagation, both are important core concepts in deep learning for training neural networks.

1.Forward Pass

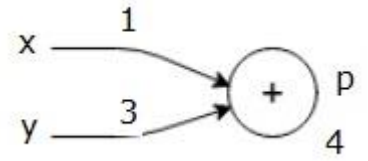
Forward pass is the procedure for evaluating the value of the mathematical expression represented by computational graphs. Doing forward pass means we are passing the value from variables in forward direction from the left (input) to the right where the output is.

Let us consider an example by giving some value to all of the inputs. Suppose, the following values are given to all of the inputs.

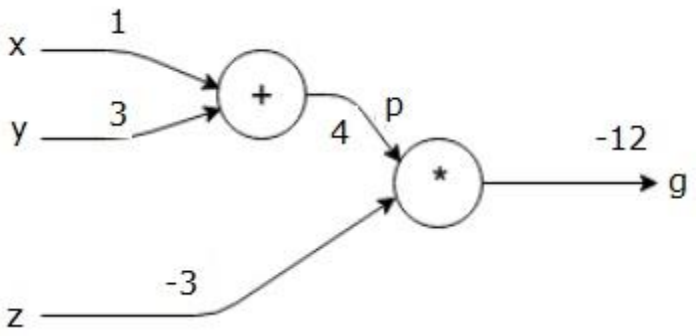
$$x=1,y=3,z=-3$$

By giving these values to the inputs, we can perform forward pass and get the following values for the outputs on each node.

First, we use the value of x = 1 and y = 3, to get p = 4.



Then we use p = 4 and z = -3 to get g = -12. We go from left to right, forwards.



To formalize our graphs, we also need to introduce the idea of an operation. An operation is a simple function of one or more variables. Our graph language is accompanied by a set of allowable operations. Functions more complicated than the operations in this set may be described by composing many operations together. Without loss of generality, we define an operation to return only a single output variable. This does not lose generality because the output variable can have multiple entries, such as a vector. Software implementations of back-propagation usually support operations with multiple outputs, but we avoid this case in our description because it introduces many extra details that are not important to conceptual understanding.

If a variable y is computed by applying an operation to a variable x, then we draw a directed edge from x to y. We sometimes annotate the output node with the name of the operation applied, and other times omit this label when the operation is clear from context.

Examples of computational graphs are shown in Fig. 6.8.

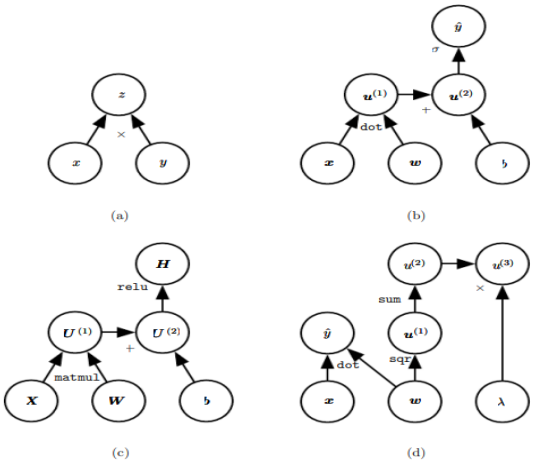
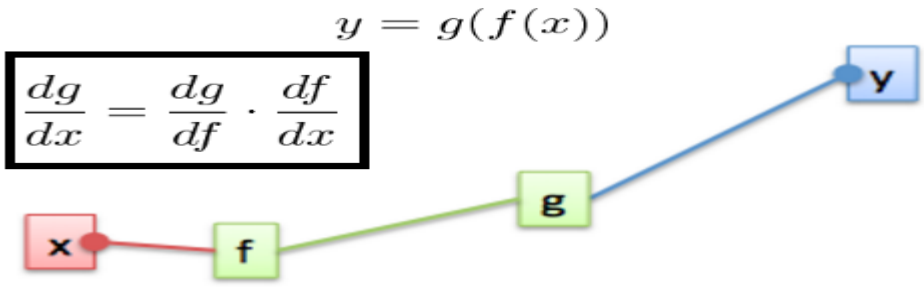


Figure 6.8: Examples of computational graphs. *(a)* The graph using the$\times$ operation to compute $z = xy$. *(b)* The graph for the logistic regression prediction $\hat{y} = \sigma\left(\boldsymbol{x}^\top \boldsymbol{w} + b\right)$. Some of the intermediate expressions do not have names in the algebraic expression but need names in the graph. We simply name the i-th such variable $\boldsymbol{u}^{(i)}$. *(c)* The computational graph for the expression $\boldsymbol{H} = \max\{0, \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}\}$, which computes a design matrix of rectified linear unit activations $\boldsymbol{H}$ given a design matrix containing a minibatch of inputs $\boldsymbol{X}$. *(d)* Examples a–c applied at most one operation to each variable, but it is possible to apply more than one operation. Here we show a computation graph that applies more than one operation to the weights $\boldsymbol{w}$ of a linear regression model. The weights are used to make the both the prediction $\hat{y}$ and the weight decay penalty $\lambda \sum_i w_i^2$.

2. Backpropagation and the chain rule used in backward pass

The backpropagation algorithm is really just an example of the trusty chain-rule from calculus. It states how to find the influence of a certain input, on systems that are composed of multiple functions. So for example in the image below, if you want to know the influence of *x* on the function *g*, we just multiply the influence of *f* on g by the influence of *x* on *f*:



Unit 4

Regularization for Deep Learning: Parameter norm Penalties, Norm Penalties as Constrained Optimization, Regularization and Under-Constrained Problems, Dataset Augmentation, Noise Robustness, Semi-Supervised learning, Multi-task learning, Early Stopping, Parameter Typing and Parameter Sharing, Sparse Representations, Bagging and other Ensemble Methods, Dropout, Adversarial Training, Tangent Distance, tangent Prop and Manifold, Tangent Classifier

Regularization is a set of techniques that can prevent overfitting in neural networks and thus improve the accuracy of a Deep Learning model when facing completely new data from the problem domain.

Regularization is a technique used in machine learning and deep learning to prevent overfitting and improve the generalization performance of a model. It involves adding a penalty term to the loss function during training.

This penalty discourages the model from becoming too complex or having large parameter values, which helps in controlling the model's ability to fit noise in the training data. Regularization methods include L1 and L2 regularization, dropout, early stopping, and more. By applying regularization, models become more robust and better at making accurate predictions on unseen data.

Underfitting in Machine Learning

A statistical model or a machine learning algorithm is said to have underfitting when a model is too simple to capture data complexities. It represents the inability of the model to learn the training data effectively result in poor performance both on the training and testing data. In simple terms, an underfit model's are inaccurate, especially when applied to new, unseen examples. It mainly happens when we uses very simple model with overly simplified assumptions. To address underfitting problem of the model, we need to use more complex models, with enhanced feature representation, and less regularization.

Note: The underfitting model has High bias and low variance.

Bias and Variance in Machine Learning

- **Bias**: Bias refers to the error due to overly simplistic assumptions in the learning algorithm. These assumptions make the model easier to comprehend and learn but might not capture the underlying complexities of the data. It is the error due to the model's inability to represent the true relationship between input and output accurately. When a model has poor performance both on the training and testing data means high bias because of the simple model, indicating underfitting.
- **Variance**: Variance, on the other hand, is the error due to the model's sensitivity to fluctuations in the training data. It's the variability of the model's predictions for different instances of training data. High variance occurs when a model learns the training data's noise and random fluctuations rather than the underlying pattern. As a result, the model performs well on the training data but poorly on the testing data, indicating overfitting.

Reasons for Underfitting

1. The model is too simple, So it may be not capable to represent the complexities in the data.
2. The input features which is used to train the model is not the adequate representations of underlying factors influencing the target variable.
3. The size of the training dataset used is not enough.
4. Excessive regularization are used to prevent the overfitting, which constraint the model to capture the data well.

Techniques to Reduce Underfitting

1. Increase model complexity.
2. Increase the number of features, performing feature engineering.
3. Remove noise from the data.
4. Increase the number of epochs or increase the duration of training to get better results.

Example An epoch is when all the training data is used at once and is defined as the total number of iterations of all the training data in one cycle for training the machine learning model. Another way to define an epoch is the number of passes a training dataset takes around an algorithm.

Overfitting in Machine Learning

A statistical model is said to be overfitted when the model does not make accurate predictions on testing data. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. And when testing with test data

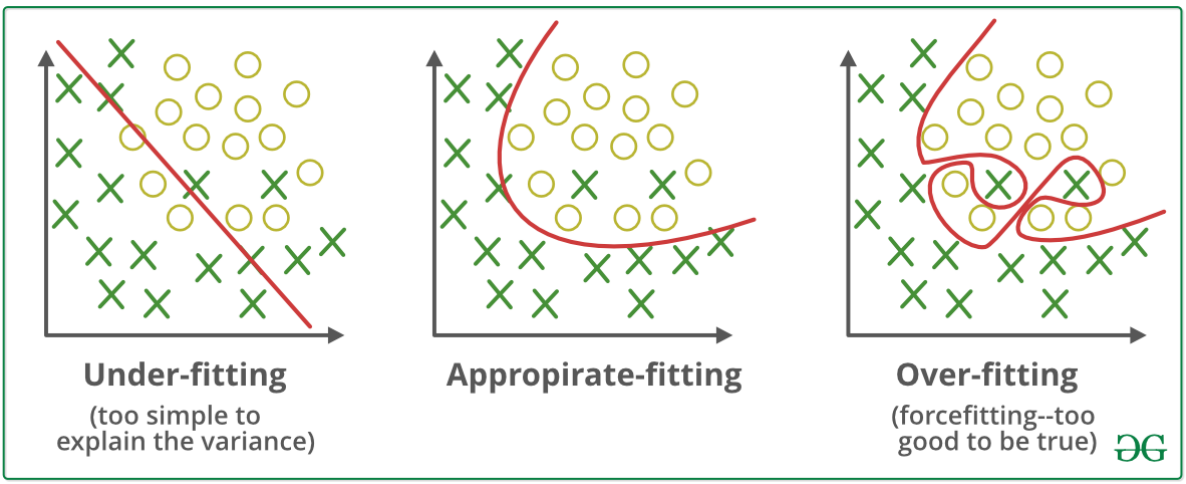
results in High variance. Then the model does not categorize the data correctly, because of too many details and noise. The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

Reasons for Overfitting:

1. High variance and low bias.
2. The model is too complex.
3. The size of the training data.

Techniques to Reduce Overfitting

1. Increase training data.
2. Reduce model complexity.
3. Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
4. Ridge Regularization and Lasso Regularization.
5. Use dropout for neural networks to tackle overfitting.



Good Fit in a Statistical Model

Ideally, the case when the model makes the predictions with 0 error, is said to have a good fit on the data. This situation is achievable at a spot between overfitting and

underfitting. In order to understand it, we will have to look at the performance of our model with the passage of time, while it is learning from the training dataset.
With the passage of time, our model will keep on learning, and thus the error for the model on the training and testing data will keep on decreasing. If it will learn for too long, the model will become more prone to overfitting due to the presence of noise and less useful details. Hence the performance of our model will decrease. In order to get a good fit, we will stop at a point just before where the error starts increasing. At this point, the model is said to have good skills in training datasets as well as our unseen testing dataset.

Parameter norm Penalties

Parameter Norm Penalties are regularization methods that apply a penalty to the norm of parameters in the objective function of a neural network.

Different Regularization Techniques in Deep Learning

Now that we have an understanding of how regularization helps in reducing overfitting, we'll learn a few different techniques in order to apply regularization in deep learning.

Lasso Regression

A regression model which uses the **L1 Regularization** technique is called **LASSO(Least Absolute Shrinkage and Selection Operator)** regression. **Lasso Regression** adds the *"absolute value of magnitude"* of the coefficient as a penalty term to the loss function(L). Lasso regression also helps us achieve feature selection by penalizing the weights to approximately equal to zero if that feature does not serve any purpose in the model.

$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m} |w_i|$$

where,

- ***m*** *– Number of Features*
- ***n*** *– Number of Examples*
- ***y_i*** *– Actual Target Value*
- ***y_i(hat)*** *– Predicted Target Value*

Ridge Regression

A regression model that uses the **L2 regularization** technique is called **Ridge regression**. **Ridge regression** adds the "*squared magnitude*" of the coefficient as a penalty term to the loss function(L).

$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{i=1}^{m} w_i^2$$

L2 & L1 regularization

L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.

Cost function = Loss (say, binary cross entropy) + Regularization term

Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent.

However, this regularization term differs in L1 and L2.

In L2, we have:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

Here, **lambda** is the regularization parameter. It is the hyperparameter whose value is optimized for better results. L2 regularization is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero).

In L1, we have:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

In this, we penalize the absolute value of the weights. Unlike L2, the weights may be reduced to zero here. **Hence, it is very useful when we are trying to compress our model. Otherwise, we usually prefer L2 over it.**

In keras, we can directly apply regularization to any layer using the regularizers. Below I have applied regularizer on dense layer having 500 neurons and relu activation function.

In [11]:

```
#creating sequential model
model=Sequential()
model.add(Conv2D(filters=16,kernel_size=2,padding="same",activation="relu",input_shape
=(50,50,3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
```

```
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
#l2 regularizer
model.add(Dense(500,kernel_regularizer=regularizers.l2(0.01),activation="relu"))
model.add(Dense(2,activation="softmax"))#2 represent output layer neurons
```

Note: Here the value 0.01 is the value of regularization parameter, i.e., lambda, which we need to optimize further

Similarly, we can also apply L1 regularization.

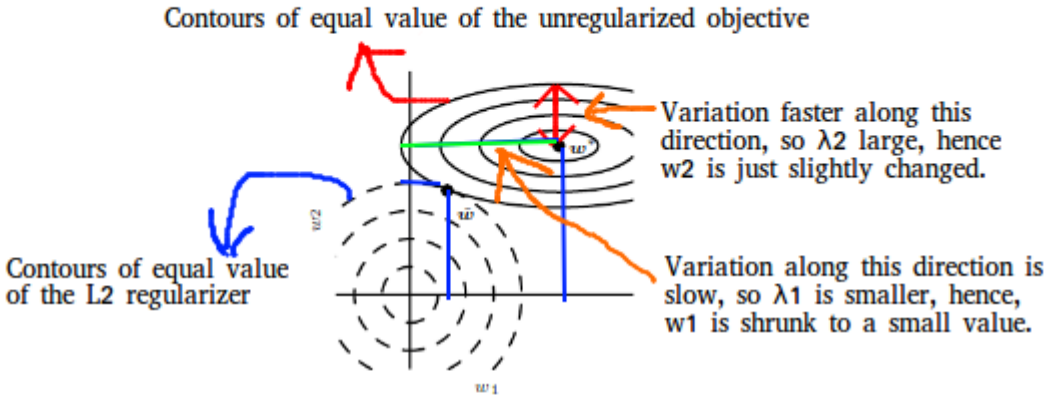


Figure 7.1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal $\boldsymbol{w}$. The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L^2 regularizer. At the point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

2. Norm penalties as constrained optimization

we can construct a generalized Lagrangian function containing the objective function along with the penalties can be increased or decreased. Suppose we wanted **Ω(θ) < k**, then we could construct the following Lagrangian equation proposed by author:

$$\mathcal{L}(\boldsymbol{\theta}, \alpha; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha(\Omega(\boldsymbol{\theta}) - k).$$

We get optimal θ by solving the Lagrangian. If Ω(θ) > k, then the weights need to be compensated highly and hence, α should be large to reduce its value below k.

Likewise, if Ω(θ)<k, then the norm shouldn't be reduced too much and hence, α should be small. This is now similar to the parameter norm penalty regularized objective function as both of them encourage lower values of the norm. Thus, parameter norm penalties naturally impose a constraint, like the L^2-regularization, defining a constrained L^2-ball.

Larger α implies a smaller constrained region as it pushes the values really low, hence, allowing a small radius and vice versa. The idea of constraints over penalties is important for several reasons. Large penalties might cause non-convex optimization algorithms to get stuck in local minima due to small values of **θ**, leading to the formation of so-called *dead cells*, as the weights entering and leaving them are too small to have an impact.

Constraints don't enforce the weights to be near zero, rather being confined to a constrained region.

3. Regularized & Under-constrained problems

Underdetermined problems are those problems that have infinitely many solutions. A logistic regression problem having linearly separable classes with as a solution, will always have *2w* as a solution and so on. In some machine learning problems, regularization is necessary. For e.g., many algorithms require the inversion of X' X, which might be singular. In such a case, we can use a regularized form instead. (X' X + αI) is guaranteed to be invertible.

Regularization can solve underdetermined problems. For e.g. the Moore-Pentose proposed pseudoinverse defined earlier as:

$$X^+ = lim_{\alpha \to 0}(X^T X + \alpha I)^{-1} X^T$$

This can be applied in performing a linear regression with L^2-regularization.

Many linear models in machine learning, including linear regression depend on inverting the matrix X⊺X. This is not possible whenever X⊺X is singular. This matrix can be singular whenever the data generating distribution truly has no variance in some direction, or when no variance in observed in some direction because there are fewer examples (rows of X) than input features (columns of X). In this case, many forms of regularization correspond to inverting X⊺X + αI instead. This regularized matrix is guaranteed to be invertible.

Data Augmentation

The simplest way to reduce overfitting is to increase the size of the training data. In machine learning, we were not able to increase the size of training data as the labeled data was too costly.

But, now let's consider we are dealing with images. In this case, there are a few ways of increasing the size of the training data – rotating the image, flipping, scaling, shifting, etc. In the below image, some transformation has been done on the handwritten digits dataset.

This technique is known as data augmentation. This usually provides a big leap in improving the accuracy of the model. *It can be considered as a mandatory trick in order to improve our predictions.*

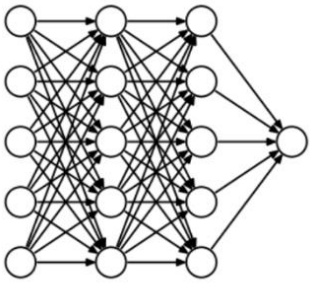
Below is the implementation code example

```
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
    zoom_range = 0.1, # Randomly zoom image
    width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=False,  # randomly flip images
    vertical_flip=False)  # randomly flip images

datagen.fit(x_train)
```

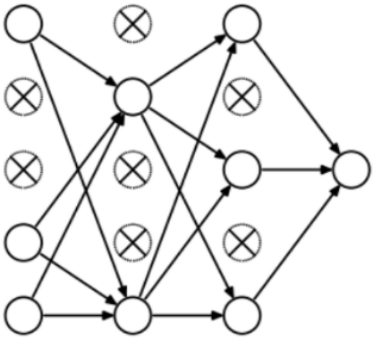
Dropout

This is the one of the most interesting types of regularization techniques. It also produces very good results and is consequently the most frequently used regularization technique in the field of deep learning.

To understand dropout, let's say our neural network structure is akin to the one shown



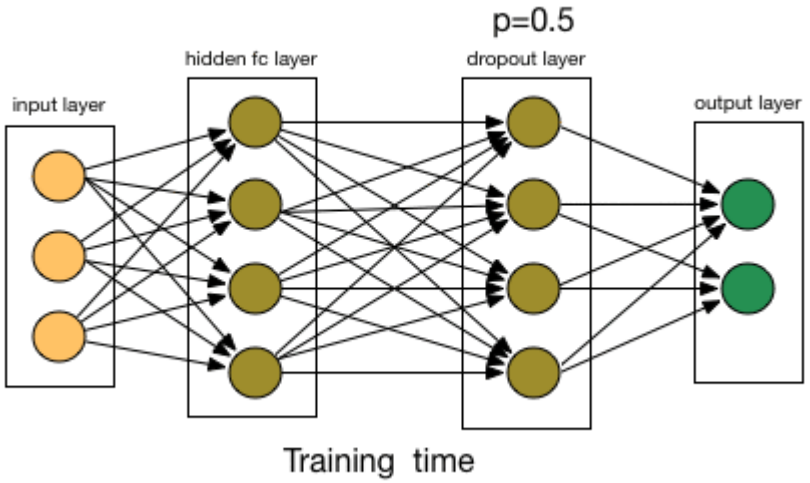
So what does dropout do? At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections as shown below.



So each iteration has a different set of nodes and this results in a different set of outputs. **It can also be thought of as an ensemble technique in machine learning.**

Ensemble models usually perform better than a single model as they capture more randomness. Similarly, dropout also performs better than a normal neural network model.

This probability of choosing how many nodes should be dropped is the hyperparameter of the dropout function. As seen in the image above, dropout can be applied to both the hidden layers as well as the input layers.



Due to these reasons, dropout is usually preferred when we have a large neural network structure in order to introduce more **randomness.**

In keras, we can implement dropout using the **keras layer**. Below is the Dropout Implementation. I have introduced dropout of 0.5 as the probability of dropping in my neural network architecture after last hidden layer having 64 kernels and after first dense layer having 500 neurons.

example
linkcode

```
#creating sequential model
model=Sequential()
model.add(Conv2D(filters=16,kernel_size=2,padding="same",activation="relu",input_shape
=(50,50,3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
# 1st dropout
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(500,activation="relu"))
# 2nd dropout
model.add(Dropout(0.2))
model.add(Dense(2,activation="softmax"))#2 represent output layer neurons
```

Early stopping

Early stopping is a kind of cross-validation strategy where we keep one part of the training set as the validation set. When we see that the performance on the validation set is getting worse, we immediately stop the training on the model. This is known as early stopping.

In the above image, we will stop training at the dotted line since after that our model will start overfitting on the training data.

in keras, we can apply early stopping using the **callbacks** function. Below is the implementation code for it.I have applied early stopping so that it will stop immendiately if validation error will not decreased after 3 epochs.

```
In [14]:
from keras.callbacks import EarlyStopping
earlystop= EarlyStopping(monitor='val_acc', patience=3)
epochs = 20 #
batch_size = 256
```

Here, **monitor** denotes the quantity that needs to be monitored and **'val_err'** denotes the validation error.

Patience denotes the number of epochs with no further improvement after which the training will be stopped. For better understanding, let's take a look at the above image again. After the dotted line, each epoch will result in a higher value of validation error.

Therefore, 5 epochs after the dotted line (since our patience is equal to 3), our model will stop because no further improvement is seen.

Noise Robustness

Noise applied to inputs is a data augmentation, For some models addition of noise with extremely small variance at the input is equivalent to imposing a penalty on the norm of the weights.

Noise applied to hidden units, Noise injection can be much more powerful than simply shrinking the parameters. Noise applied to hidden units is so important that Dropout is the main development of this approach.

Training a neural network with a small dataset can cause the network to memorize all training examples, in turn leading to overfitting and poor performance on a holdout dataset. One approach to making the input space smoother and easier to learn is to add noise to inputs during training.

- Small datasets can make learning challenging for neural nets and the examples can be memorized.
- Adding noise during training can make the training process more robust and reduce generalization error.
- Noise is traditionally added to the inputs, but can also be added to weights, gradients, and even activation functions.

random noise can be added to other parts of the network during training. Some examples include:

- **Add noise to activations**, i.e. the outputs of each layer.
- **Add noise to weights**, i.e. an alternative to the inputs.
- **Add noise to the gradients**, i.e. the direction to update weights.
- **Add noise to the outputs**, i.e. the labels or target variables.

The addition of noise to the layer activations allows noise to be used at any point in the network. This can be beneficial for very deep networks. Noise can be added to the layer outputs themselves, but this is more likely achieved via the use of a noisy activation function.

The addition of noise to weights allows the approach to be used throughout the network in a consistent way instead of adding noise to inputs and layer activations. This is particularly useful in recurrent neural networks.

The addition of noise to gradients focuses more on improving the robustness of the optimization process itself rather than the structure of the input domain. The amount of noise can start high at the beginning of training and decrease over time, much like a decaying learning rate. This approach has proven to be an effective method for very deep networks and for a variety of different network types

Adding noise to the activations, weights, or gradients all provide a more generic approach to adding noise that is invariant to the types of input variables provided to the model.

If the problem domain is believed or expected to have mislabeled examples, then the addition of noise to the class label can improve the model's robustness to this type of error. Although, it can be easy to derail the learning process.

Adding noise to a continuous target variable in the case of regression or time series forecasting is much like the addition of noise to the input variables and may be a better use case.

Semi-Supervised Learning

Semi-supervised learning is a type of machine learning that falls in between supervised and unsupervised learning. It is a method that uses a small amount of labeled data and a large amount of unlabeled data to train a model. The goal of semi-supervised learning is to learn a function that can accurately predict the output variable based on the input variables, similar to supervised learning. However, unlike supervised learning, the algorithm is trained on a dataset that contains both labeled and unlabeled data.
Semi-supervised learning is particularly useful when there is a large amount of unlabeled data available, but it's too expensive or difficult to label all of it.

Examples of Semi-Supervised Learning

- **Text classification**: In text classification, the goal is to classify a given text into one or more predefined categories. Semi-supervised learning can be used to train a text classification model using a small amount of labeled data and a large amount of unlabeled text data.
- **Image classification**: In image classification, the goal is to classify a given image into one or more predefined categories. Semi-supervised learning can be used to train an image classification model using a small amount of labeled data and a large amount of unlabeled image data.
- **Anomaly detection**: In anomaly detection, the goal is to detect patterns or observations that are unusual or different from the norm

Applications of Semi-Supervised Learning

1. **Speech Analysis:** Since labeling audio files is a very intensive task, Semi-Supervised learning is a very natural approach to solve this problem.
2. **Internet Content Classification:** Labeling each webpage is an impractical and unfeasible process and thus uses Semi-Supervised learning algorithms. Even the Google search algorithm uses a variant of Semi-Supervised learning to rank the relevance of a webpage for a given query.
3. **Protein Sequence Classification:** Since DNA strands are typically very large in size, the rise of Semi-Supervised learning has been imminent in this field.

Disadvantages of Semi-Supervised Learning

The most basic disadvantage of any **Supervised Learning** algorithm is that the dataset has to be hand-labeled either by a Machine Learning Engineer or a Data Scientist. This is a very *costly process*, especially when dealing with large volumes of data. The most basic disadvantage of any **Unsupervised Learning** is that its **application spectrum is limited**.

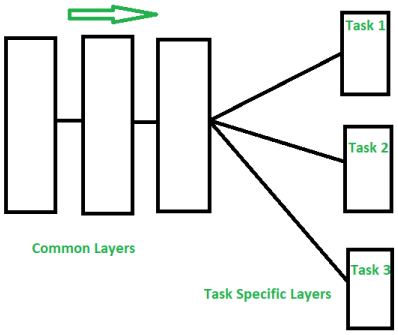
To counter these disadvantages, the concept of **Semi-Supervised Learning** was introduced. In this type of learning, the algorithm is trained upon a combination of labeled and unlabelled data. Typically, this combination will contain a very small amount of labeled data and a very large amount of unlabelled data. The basic procedure involved is that first, the programmer will cluster similar data using an unsupervised learning algorithm and then use the existing labeled data to label the rest of the unlabelled data.

Multi-Task Learning

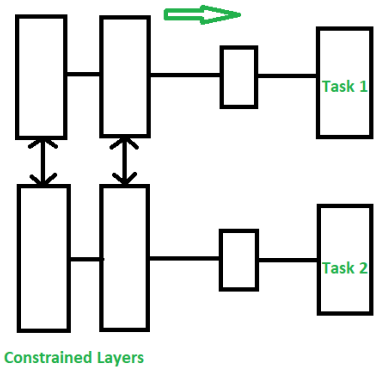
Multi-Task Learning (MTL) is a type of machine learning technique where a model is trained to perform multiple tasks simultaneously. In deep learning, MTL refers to training a neural network to perform multiple tasks by sharing some of the network's layers and parameters across tasks.

In MTL, the goal is to improve the generalization performance of the model by leveraging the information shared across tasks. By sharing some of the network's parameters, the model can learn a more efficient and compact representation of the data, which can be beneficial when the tasks are related or have some commonalities.

Hard Parameter Sharing – A common hidden layer is used for all tasks but several task specific layers are kept intact towards the end of the model. This technique is very useful as by learning a representation for various tasks by a common hidden layer, we reduce the risk of overfitting.



Soft Parameter Sharing – Each model has their own sets of weights and biases and the distance between these parameters in different models is regularized so that the parameters become similar and can represent all the tasks.



Assumptions and Considerations –
Using MTL to share knowledge among tasks are very useful only when the tasks are very similar, but when this assumption is violated, the performance will significantly decline.
Applications: MTL techniques have found various uses, some of the major applications are-

- Object detection and Facial recognition
- Self Driving Cars: Pedestrians, stop signs and other obstacles can be detected together
- Multi-domain collaborative filtering for web applications
- Stock Prediction
- Language Modelling and other NLP applications

Multi-Task Learning (MTL) for deep learning important observations :

1. Task relatedness: MTL is most effective when the tasks are related or have some commonalities, such as natural language processing, computer vision, and healthcare.

2. Data limitation: MTL can be useful when the data is limited, as it allows the model to leverage the information shared across tasks to improve the generalization performance.
3. Shared feature extractor: A common approach in MTL is to use a shared feature extractor, which is a part of the network that is shared across tasks and is used to extract features from the input data.
4. Task-specific heads: Task-specific heads are used to make predictions for each task and are typically connected to the shared feature extractor.
5. Shared decision-making layer: another approach is to use a shared decision-making layer, where the decision-making layer is shared across tasks, and the task-specific layers are connected to the shared decision-making layer.

Parameter Typing

Two models are doing the same classification task (with the same set of classes), but their input distributions are somewhat different.

- We have model **A** has the parameters
- Another model **B** has the parameters .

W(A)

and

W(B)

$$\hat{y}^{(A)} = f\left(\boldsymbol{w}^{(A)}, \boldsymbol{x}\right)$$

$$\hat{y}^{(B)} = g\left(\boldsymbol{w}^{(B)}, \boldsymbol{x}\right)$$

are the two models that transfer the input to two different but related outputs.

Assume the tasks are comparable enough (possibly with similar input and output distributions) that the model parameters should be near to each other: $\hat{y}^{(A)} = f\left(\boldsymbol{w}^{(A)}, \boldsymbol{x}\right)$ should be close to $\hat{y}^{(B)} = g\left(\boldsymbol{w}^{(B)}, \boldsymbol{x}\right)$.

We can take advantage of this data by regularising it. We can apply a parameter norm penalty of the following form We utilised an $\mathbf{L}^2$ penalty here, but there are other options.

Parameter Sharing

The parameters of one model, trained as a classifier in a supervised paradigm, were regularised to be close to the parameters of another model, trained in an unsupervised paradigm, using this method (to capture the distribution of the observed input data).

Many of the parameters in the classifier model might be linked with similar parameters in the unsupervised model thanks to the designs.

While a parameter norm penalty is one technique to require sets of parameters to be equal, constraints are a more prevalent way to regularise parameters to be close to one another. Because we view the numerous models or model components as sharing a unique set of parameters, this form of regularisation is commonly referred to as parameter sharing. The fact that only a subset of the parameters (the unique set) needs to be retained in memory is a significant advantage of parameter sharing over regularising the parameters to be close (through a norm penalty). This can result in a large reduction in the memory footprint of certain models, such as the convolutional neural network.

Example : Convolutional neural networks (CNNs) used in computer vision are by far the most widespread and extensive usage of parameter sharing. Many statistical features of natural images are translation insensitive. A shot of a cat, for example, can be translated one pixel to the right and still be a shot of a cat. By sharing parameters across several picture locations, CNNs take this property into account. Different locations in the input are computed with the same feature (a hidden unit with the same weights).

Sparse Representations

Sparse representation (SR) is used to represent data with as few atoms as possible in a given overcomplete dictionary. By using the SR, we can concisely represent the data and easily extract the valuable information from the data

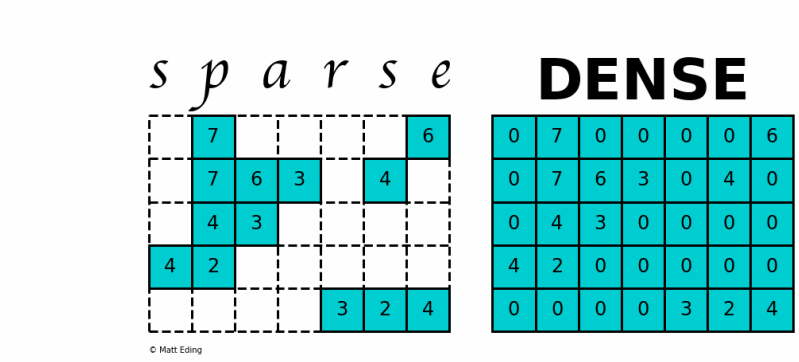
Sparse representations classification (SRC) is a powerful technique for pixelwise classification of images and it is increasingly being used for a wide variety of image analysis tasks. The method uses sparse representation and learned redundant dictionaries to classify image pixels.

sparse representation attracts great attention as it can significantly save computing resources and find the characteristics of data in a low-dimensional space. Thus, it can be widely applied in engineering fields such as dictionary learning, signal reconstruction, image clustering, feature selection, and extraction.

As real-world data becomes more diverse and complex, it becomes hard to completely reveal the intrinsic structure of data with commonly used approaches. This has led to the exploration of more practicable representation models and efficient optimization approaches. New formulations such as deep sparse representation, graph-based sparse representation, geometry-guided sparse representation, and group sparse representation have achieved remarkable success

the terms "sparse" and "dense" are commonly used to describe the distribution of zero and non-zero array members in machine learning (e.g. vector or matrix). Sparse matrices are those that primarily consist of zeros, while dense matrices have a large number of nonzero entries.

Machine learning makes use of sparse and dense representations due to their usefulness in efficient data representation. While dense representations are useful for capturing intricate interactions between data points, sparse representations can help minimize the amount of a dataset.



- **Sparse** representations *have the potential to be more resilient* to noise and produce more interpretable outcomes. For calculations, dense representations are typically more effective since they can be processed more quickly. On top of that, dense representations are useful for tasks like classification and regression because they can capture intricate connections between data points.
- **Sparse** representations ar*e helpful for reducing the dimensionality of the data in tasks* like natural language processing and picture recognition. Further, sparse representations can be utilized to capture only the most crucial elements of the data, which can greatly cut down on the time needed to train a model.

- **Dense** representations are able to *capture complicated interactions between data points*, they are frequently employed in machine learning and can be especially helpful for tasks like classification and regression. Because of their increased

computational efficiency, dense representations can also shorten the time it takes to train a model.

A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have **0 value**, then it is called a sparse matrix.

Why to use Sparse Matrix instead of simple matrix ?

- **Storage:** There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
- **Computing time:** Computing time can be saved by logically designing a data structure traversing only non-zero elements..

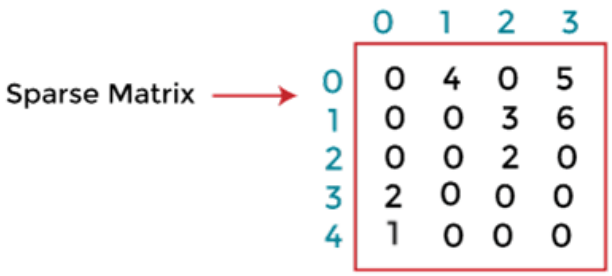
sparse Matrix Representations can be done in many ways following are two common representations:

1. Array representation
2. Linked list representation

Example -

Let's understand the array representation of sparse matrix with the help of the example given below -

Consider the sparse matrix -



In the above figure, we can observe a 5x4 sparse matrix containing 7 non-zero elements and 13 zero elements. The above matrix occupies 5x4 = 20 memory space. Increasing the size of matrix will increase the wastage space.

The tabular representation of the above matrix is given below -

Table Structure

Row	Column	Value
0	1	4
0	3	5
1	2	3
1	3	6
2	2	2
3	0	2
4	0	1
5	4	7

In the above structure, first column represents the rows, the second column represents the columns, and the third column represents the non-zero value. The first row of the table represents the triplets. The first triplet represents that the value 4 is stored at 0th row and 1st column. Similarly, the second triplet represents that the value 5 is stored at the 0th row and 3rd column. In a similar manner, all triplets represent the stored location of the non-zero elements in the matrix.

The size of the table depends upon the total number of non-zero elements in the given sparse matrix. Above table occupies 8x3 = 24 memory space which is more than the space occupied by the sparse matrix. So, what's the benefit of using the sparse matrix? Consider the case if the matrix is 8*8 and there are only 8 non-zero elements in the matrix, then the space occupied by the sparse matrix would be 8*8 = 64, whereas the space occupied by the table represented using triplets would be 8*3 = 24.

Example -

Let's understand the linked list representation of sparse matrix with the help of the example given below -

Consider the sparse matrix

Sparse Matrix →

	0	1	2	3
0	0	0	1	0
1	3	0	0	0
2	0	4	5	0
3	0	6	0	0

In the above figure, we can observe a 4x4 sparse matrix containing 5 non-zero elements and 11 zero elements. Above matrix occupies 4x4 = 16 memory space. Increasing the size of matrix will increase the wastage space.

The linked list representation of the above matrix is given below -

In the above figure, the sparse matrix is represented in the linked list form. In the node, the first field represents the index of the row, the second field represents the index of the column, the third field represents the value, and the fourth field contains the address of the next node.

In the above figure, the first field of the first node of the linked list contains 0, which means 0^{th} row, the second field contains 2, which means 2^{nd} column, and the third field contains 1 that is the non-zero element. So, the first node represents that element 1 is stored at the 0^{th} row-2^{nd} column in the given sparse matrix. In a similar manner, all of the nodes represent the non-zero elements of the sparse matrix.

Sparse Coding representation in Neural Networks

sparse code follows the more all-encompassing idea of neural code. Consider the case when you have binary neurons. So, basically:

- The neural networks will get some inputs and deliver outputs
- Some neurons in the neural network will be frequently activated while others won't be activated at all to calculate the outputs
- The average activity ratio refers to the number of activations on some data, whereas the neural code is the observation of those activations for a specific input
- Neural coding is the process of instructing your neurons to produce a reliable neural code

Now that we know what a neural code is, we can speculate on what it may be like. **Then, data will be encoded using a sparse code while taking into consideration the following scenarios:**

- No neurons are even activated
- One neuron alone is activated
- Half of the neurons are active

These are the methods which are being followed to represent image and its classifications

Ensemble Learning Methods: Bagging, Boosting

Ensemble learning is a machine learning technique combining multiple individual models to create a stronger, more accurate predictive model. By leveraging the diverse strengths of different models, ensemble learning aims to mitigate errors, enhance performance, and increase the overall robustness of predictions, leading to improved results across various tasks in machine learning and neural networks .

Bagging or Bootstrap Aggregating is an ensemble learning method that is used to reduce the error by training homogeneous weak learners on different random samples from the training set, in parallel. The results of these base learners are then combined through voting or averaging approach to produce an ensemble model that is more robust and accurate.

Bagging mainly focuses on obtaining an ensemble model with lower variance than the individual base models composing it. Hence, bagging techniques help avoid the overfitting of the model.

Benefits of Bagging

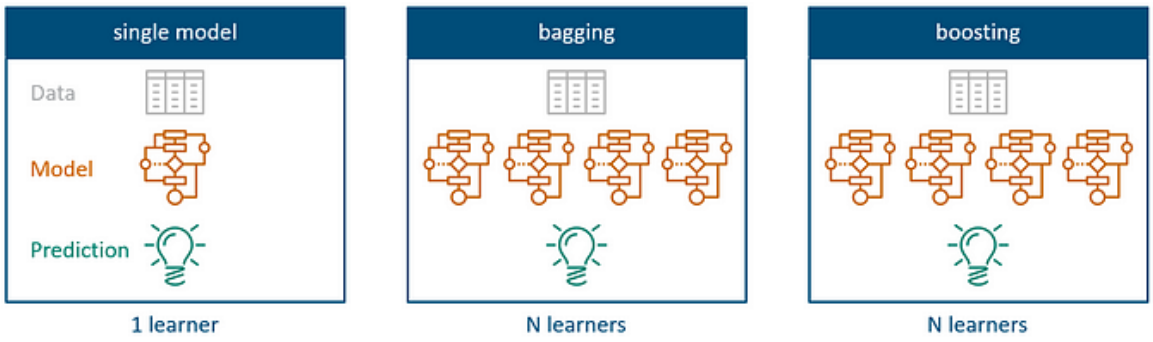
- Reduce Overfitting
- Improve Accuracy
- Handles Unstable Models

Note: Random Forest Algorithm is one of the most common Bagging Algorithm.

Steps of Bagging Technique

- Randomly select multiple bootstrap samples from the training data with replacement and train a separate model on each sample.
- For classification, combine predictions using majority voting. For regression, average the predictions.
- Assess the ensemble's performance on test data and use the aggregated models for predictions on new data.
- If needed, retrain the ensemble with new data or integrate new models into the existing ensemble.

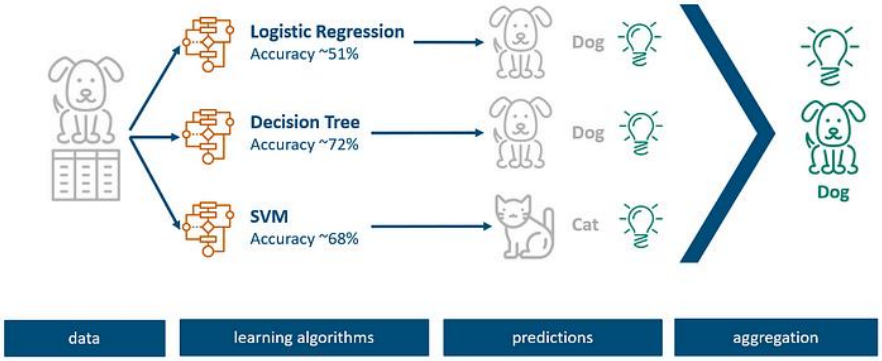
Example of Bagging and boosting



The main idea behind ensemble learning is the usage of multiple algorithms and models that are used together for the same task. While single models use only one algorithm to create prediction models, bagging and boosting methods aim to combine several of those to achieve better prediction with higher consistency compared to individual learnings.

Image classification

Supposing a collection of images, each accompanied by a categorical label corresponding to the kind of animal, is available for the purpose of training a model. In a traditional modeling approach, we would try several techniques and calculate the accuracy to choose one over the other. Imagine we used logistic regression, decision tree, and support vector machines here that perform differently on the given data set.



In the above example, it was observed that a specific record was predicted as a dog by the logistic regression and decision tree models, while a support vector machine identified it as a cat. As various models have their distinct advantages and disadvantages for particular

records, it is the key idea of ensemble learning to combine all three models instead of selecting only one approach that showed the highest accuracy.

The procedure is called *aggregation* or *voting* and combines the predictions of all underlying models, to come up with one prediction that is assumed to be more precise than any sub-model that would stay alone.

Boosting is an ensemble learning method that involves training homogenous weak learners **sequentially** such that a base model depends on the previously fitted base models. All these base learners are then combined in a very adaptive way to obtain an ensemble model.

In boosting, the ensemble model is the **weighted sum** of all constituent base learners. There are two meta-algorithms in boosting that differentiate how the base models are aggregated:

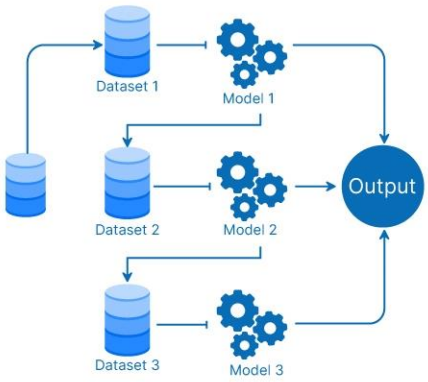
- Adaptive Boosting (AdaBoost)
- Gradient Boosting
- XGBoost

Benefits of Boosting Techniques

- High Accuracy
- Adaptive Learning
- Reduces Bias
- Flexibility

How is Boosting Model Trained to Make Predictions

- Samples generated from the training set are assigned the **same weight** to start with. These samples are used to train a homogeneous weak learner or base model.
- The prediction error for a sample is calculated – **the greater the error, the weight of the sample increases**. Hence, the sample becomes more important for training the next base model.
- The individual learner is weighted too – **does well on its predictions, gets a higher weight assigned to it**. So, a model that outputs good predictions will have a higher say in the final decision.
- The weighted data is then passed on to the following base model, and steps 2 and step 3 are repeated until the **data is fitted well enough to reduce the error below a certain threshold**.
- When new data is fed into the boosting model, it is passed through all individual base models, and **each model makes its own weighted prediction**.
- Weight of these models is used to generate the final prediction. The predictions are scaled and **aggregated to produce a final prediction**.



Boosting

Key Difference Between Bagging and Boosting

- The bagging technique combines multiple models trained on different subsets of data, whereas boosting trains models sequentially, focusing on the error made by the previous model.
- Bagging is best for high variance and low bias models while boosting is effective when the model must be adaptive to errors, suitable for bias and variance errors.
- Generally, boosting techniques are not prone to overfitting. Still, it can be if the number of models or iterations is high, whereas the Bagging technique is less prone to overfitting.
- Bagging improves accuracy by reducing variance, whereas boosting achieves accuracy by reducing bias and variance.
- Boosting is suitable for bias and variance, while bagging is suitable for high-variance and low-bias models.

About bias and variance used in bagging and boosting

Bias:*While making predictions, a difference occurs between prediction values made by the model and actual values/expected values, and this difference is known as bias errors or Errors due to bias*

- **Low Bias:** A low bias model will make fewer assumptions about the form of the target function.
- **High Bias:** A model with a high bias makes more assumptions, and the model becomes unable to capture the important features of our dataset. **A high bias model also cannot perform well on new data.**

Variance:the variance would specify the amount of variation in the prediction if the different training data was used. In simple words, ***variance tells that how much a random variable is different from its expected value.*** Ideally, a model should not vary too much from one training dataset to another, which means the algorithm should be good in understanding the hidden mapping between inputs and output variables. Variance errors are either of **low variance or high variance.**

- **Low variance** means there is a small variation in the prediction of the target function with changes in the training data set. At the same time, **High variance** shows a large variation in the prediction of the target function with changes in the training dataset.

Tangent Distance, Tangent Prop, and Manifold Tangent Classifier

Tangent propagation is a way of regularizing neural nets. It encourages the representation to be invariant by penalizing large changes in the representation when small transformations are applied to the inputs.

It combines this prior knowledge with observed training data, by minimizing an objective function that measures both the network's error with respect to the training example values (fitting the data) and its error with respect to the desired derivatives (fitting the prior knowledge).

Tangent propagation is closely related to dataset augmentation. In both cases, the user of the algorithm encodes his or her prior knowledge of the task by specifying a set of transformations that should not alter the output of the network.

The difference is that in the case of dataset augmentation, the network is explicitly trained to correctly classify distinct inputs that were created by applying more than an infinitesimal amount of these transformations.

tangent propagation does not require explicitly visiting a new input point. Instead, it analytically regularizes the model to resist perturbation in the directions corresponding to the specified transformation. While this analytical approach is intellectually elegant,

it has two major drawbacks. First, it only regularizes the model to resist infinitesimal perturbation. Explicit dataset augmentation confers resistance to larger perturbations(means changes in datasets) Second, the infinitesimal approach poses difficulties for models based on rectified linear units. These models can only shrink their derivatives by turning units off or shrinking their weights.

They are not able to shrink their derivatives by saturating at a high value with large weights, as sigmoid or tanh units can. Dataset augmentation works well with rectified linear units because different subsets of rectified units can activate for different transformed versions of each original input. Tangent propagation is also related to double backprop (Drucker and LeCun, 1992) and adversarial training

The TANGENTPROP Algorithm TANGENTPROP (Simard et al. 1992) accommodates domain knowledge expressed as derivatives of the target function with respect to transformations of its inputs. Consider a learning task involving an instance space X and target function f.

The TANGENTPROP algorithm assumes various training derivatives of the target function are also provided. For example, if each instance xi is described by a single real value, then each training example may be of the form (xi, f (xi), q lx,). Here lx, denotes the derivative of the target function f with respect to x, evaluated at the point x = xi.

To develop an intuition for the benefits of providing training derivatives as well as training values during learning, consider the simple learning task depicted in Figure

The task is to learn the target function f shown in the leftmost plot of the figure, based on the three training examples shown: (xl, f (xl)), (x2, f (x2)), and (xg, f (xg)).

Given these three training examples, the BACKPROPAGATION algorithm can be expected to hypothesize a smooth function, such as the function g depicted in the middle plot of the figure. The rightmost plot shows the effect of

providing training derivatives, or slopes, as additional information for each training example (e.g., (XI, f (XI), I,,)). By fitting both the training values f (xi) and these training derivatives PI,, the learner has a better chance to correctly generalize from the sparse training data.

To summarize, the impact of including the training derivatives is to override the usual syntactic inductive bias of BACKPROPAGATION that favors a smooth interpolation between points, replacing it by explicit input information about required derivatives. The resulting hypothesis h shown in the rightmost plot of the figure provides a much more accurate estimate of the true target function f.

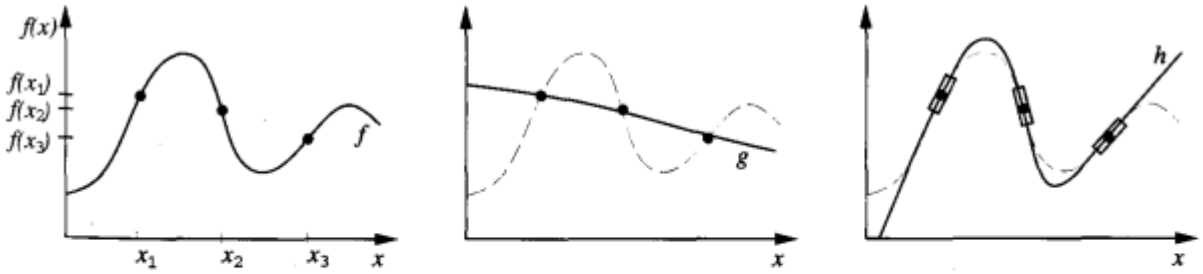


FIGURE 12.5
Fitting values and derivatives with TANGENTPROP. Let f be the target function for which three examples $\langle x_1, f(x_1)\rangle$, $\langle x_2, f(x_2)\rangle$, and $\langle x_3, f(x_3)\rangle$ are known. Based on these points the learner might generate the hypothesis g. If the derivatives are also known, the learner can generalize more accurately h.

Each transformation must be of the form sj(a, x) where aj is a continuous parameter, where sj is differentiable, and where sj(O, x) = x (e.g., for rotation of zero degrees the transformation is the identity function). For each such transformation, sj(a, x),

In the Figure one f(X) are the hypothesis and x1 , x2 ,x3 are the instances and these instances fit to proper hypothesis shown in first figure and in second fig we can see the instances classified and machine learns to fit to proper hypothesis by doing necessary modification by using

TANGEPROP considers the squared error between the specified training derivative and the actual derivative of the learned neural network. The modified error function is

$$E = \sum_i \left[(f(x_i) - \hat{f}(x_i))^2 + \mu \sum_j \left(\frac{\partial f(s_j(\alpha, x_i))}{\partial \alpha} - \frac{\partial \hat{f}(s_j(\alpha, x_i))}{\partial \alpha} \right)^2_{\alpha=0} \right]$$

where p is a constant provided by the user to determine the relative importance of fitting training values versus fitting training derivatives.

Notice the first term in this definition of E is the original squared error of the network versus training values, and the second term is the squared error in the network versus training derivatives.

In the third figure we can see the instances are classified properly and maintaining accuracy.

An Illustrative Example

Simard et al. (1992) present results comparing the generalization accuracy of TANGENTPROP and purely inductive BACKPROPAGATION for the problem of recognizing handwritten characters. More specifically, the task in this case is to label images containing a single digit between 0 and 9. In one experiment, both TANGENTPROP and BACKPROPAGATION were trained using training sets of varying size, then evaluated based on their performance over a separate test set of 160 examples. The prior knowledge given to TANGENTPROP was the fact that the classification of the digit is invariant of vertical and horizontal translation of the image (i.e., that the derivative of the target function was 0 with respect to these transformations). The results, shown in Table 12.4, demonstrate the ability of TANGENTPROP using this prior knowledge to generalize more accurately than purely inductive BACKPROPAGATION.

Training set size	Percent error on test set	
	TANGENTPROP	BACKPROPAGATION
10	34	48
20	17	33
40	7	18
80	4	10
160	0	3
320	0	0

TABLE 12.4
Generalization accuracy for TANGENTPROP and BACKPROPAGATION, for handwritten digit recognition. TANGENTPROP generalizes more accurately due to its prior knowledge that the identity of the digit is invariant of translation. These results are from Simard et al. (1992).

Remarks To summarize, TANGENTPROP uses prior knowledge in the form of desired derivatives of the target function with respect to transformations of its inputs.

It combines this prior knowledge with observed training data, by minimizing an objective function that measures both the network's error with respect to the training example values (fitting the data) and its error with respect to the desired derivatives (fitting the prior knowledge).

UNIT - V

Optimization for Train Deep Models: Challenges in Neural Network Optimization, Basic Algorithms, Parameter Initialization Strategies, Algorithms with Adaptive Learning Rates, Approximate Second Order Methods, Optimization Strategies and Meta-Algorithms

Applications: Large-Scale Deep Learning, Computer Vision, Speech Recognition, Natural Language Processing

The Challenges of Optimizing Deep Learning Models

There are several types of optimization in deep learning algorithms but the most interesting ones are focused on reducing the value of cost functions.

Some Basics of Optimization in Deep Learning Models

The core of deep learning optimization relies on trying to minimize the cost function of a model without affecting its training performance. That type of optimization problem contrasts with the general optimization problem in which the objective is to simply minimize a specific indicator without being constrained by the performance of other elements(ex:training).

Most optimization algorithms in deep learning are based on gradient estimations. In that context, optimization algorithms try to reduce the gradient of specific cost functions evaluated against the training dataset. There are different categories of optimization

algorithms depending on the way they interact with the training dataset. For instance, algorithms that use the entire training set at once are called deterministic. Other techniques that use one training example at a time has come to be known as online algorithms. Similarly, algorithms that use more than one but less than the entire training dataset during the optimization process are known as minibatch stochastic or simply stochastic.

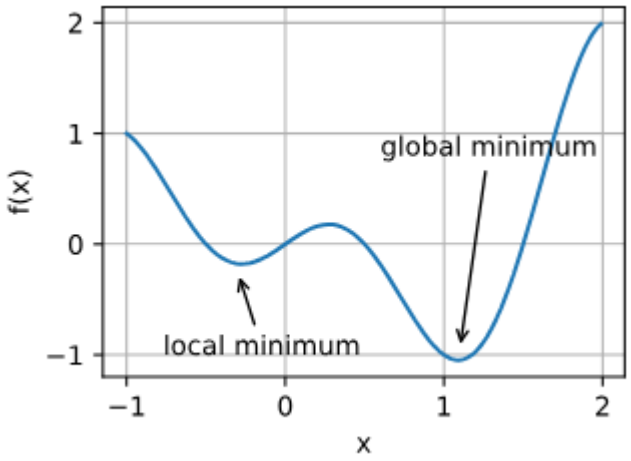
The most famous method of stochastic optimization which is also the most common algorithm in deep learning solution is known as stochastic gradient descent(SGD)(read my previous article about SGD).

Regardless of the type of optimization algorithm used, the process of optimizing a deep learning model is a careful path full of challenges.

Common Challenges in Deep Learning Optimization

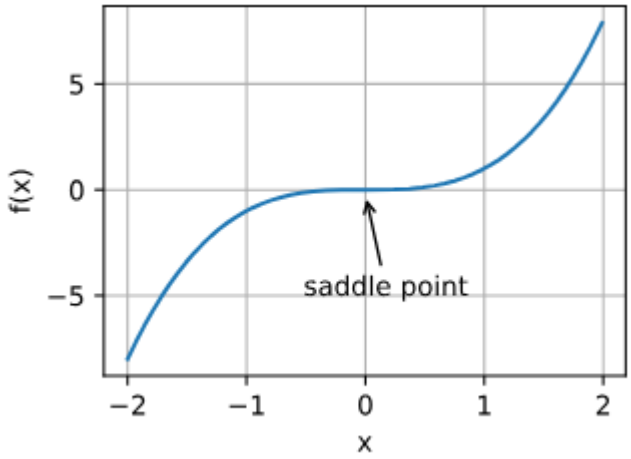
There are plenty of challenges in deep learning optimization but most of them are related to the nature of the gradient of the model. Below, I've listed some of the most common challenges in deep learning optimization that you are likely to run into:

a)Local Minima: local minima is a permanent challenge in the optimization of any deep learning algorithm. The local minima problem arises when the gradient encounters many local minimums that are different and not correlated to a global minimum for the cost function.



B.saddle points

saddle points are another reason for gradients to vanish. A *saddle point* is any location where all gradients of a function vanish but which is neither a global nor a local minimum.



Flat Regions: In deep learning optimization models, flat regions are common areas that represent both a local minimum for a sub-region and a local maximum for another. That duality often causes the gradient to get stuck.

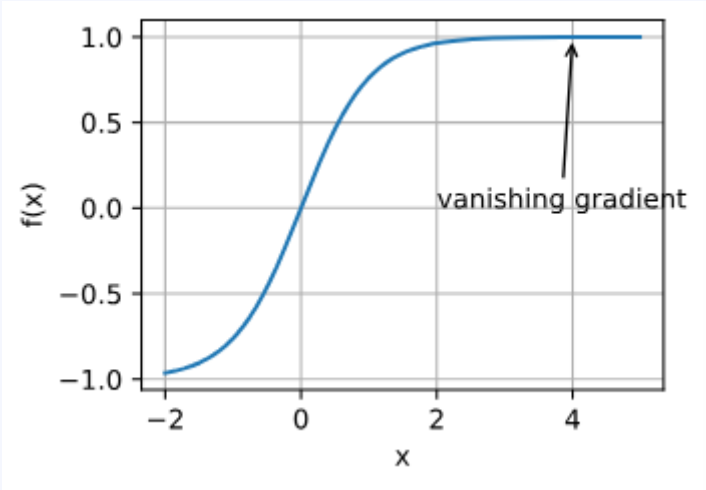
c) Inexact Gradients: There are many deep learning models in which the cost function is intractable which forces an inexact estimation of the gradient. In these cases, the inexact gradients introduce a second layer of uncertainty in the model.

d) Local vs. Global Structures: Another very common challenge in the optimization of deep leavening models is that local regions of the cost function don't correspond with its global structure producing a misleading gradient.

Vanishing and Exploding Gradients

Deep learning networks can be problematic when the numbers change too quickly or slowly through many layers. This can make it hard for the network to learn and stay stable. This can cause difficulties for the network in learning and remaining stable.

Solution: Gradient clipping, advanced weight initialization, and skip connections help a computer learn things accurately and consistently.



Overfitting

Overfitting happens when a model knows too much about the training data, so it can't make good predictions about new data. As a result, the model performs well on the training data but struggles to make accurate predictions on new, unseen data. It's essential to address overfitting by employing techniques like regularization, cross-validation, and more diverse datasets to ensure the model generalizes well to unseen examples.

Regularisation techniques help us ensure our models memorize the data and use what they've learned to make good predictions about new data. Techniques like dropout, L1/L2 regularisation, and early stopping can help us do this.

Data Augmentation and Preprocessing

Data augmentation and preprocessing are techniques used to provide better information to the model during training, enabling it to learn more effectively and make accurate predictions.

Solution: Apply data augmentation techniques like rotation, translation, and flipping alongside data normalization and proper handling of missing values.

Label Noise

Training data sometimes need to be corrected, making it hard for computers to do things well.

Solution: Using special kinds of math called "loss functions" can help ensure that the model you are using is not affected by label mistakes.

Imbalanced Datasets

Datasets can have too many of one type of thing and need more of another type. This can cause models not to work very well for things not represented as much.

Solution: Classes can sometimes be uneven, meaning more people are in one group than another. To fix this, we can use special techniques like class weighting, oversampling, or data synthesis to ensure that all the classes have the same number of people.

Computational Resource Constraints

Training deep neural networks can be very difficult and take a lot of computer power, especially if the model is very big.

Solution: Using multiple computers or special chips called GPUs and TPUs can help make learning faster and easier.

Hyperparameter Tuning

Deep neural networks have numerous hyperparameters that require careful tuning to achieve optimal performance.

Solution: To efficiently find the best hyperparameters, such as Bayesian optimization or genetic algorithms, utilize automated hyperparameter optimization methods.

Convergence Speed

It is important to ensure a model works quickly when using lots of data and complicated designs.

Solution: Adopt learning rate scheduling or adaptive algorithms like Adam or RMSprop to expedite convergence.

Memory Constraints

Computers need a lot of memory to train large models and datasets, but they can work properly if there is enough memory.

Solution: Reduce memory usage by applying model quantization, using mixed-precision training, or employing memory-efficient architectures like MobileNet or EfficientNet.

Transfer Learning and Domain Adaptation

Deep learning networks need lots of data to work well. If they don't get enough data or the data is different, they won't work as well.

Solution: Leverage transfer learning or domain adaptation techniques to transfer knowledge from pre-trained models or related domains.

.

Adversarial Attacks

Deep neural networks are unique ways of understanding data. But they can be tricked by minimal changes that we can't see. This can make them give wrong answers.

Solution: Employ adversarial training, defensive distillation, or certified robustness methods to enhance the model's robustness against adversarial attacks.

Interpretability and Explainability

Understanding the decisions made by deep neural networks is crucial in critical applications like healthcare and autonomous driving.

Solution: Adopt techniques such as LIME (Local Interpretable Model-Agnostic Explanations) or SHAP (SHapley Additive exPlanations) to explain model predictions.

Handling Sequential Data

Training deep neural networks on sequential data, such as time series or natural language sequences, presents unique challenges.

Solution: Utilize specialized architectures like recurrent neural networks (RNNs) or transformers to handle sequential data effectively.

Limited Data

Training deep neural networks with limited labeled data is a common challenge, especially in specialized domains.

Solution: Consider semi-supervised, transfer, or active learning to make the most of available data.

Catastrophic Forgetting

When a model forgets previously learned knowledge after training on new data, it encounters the issue of catastrophic forgetting.

Solution: Implement techniques like elastic weight consolidation (EWC) or knowledge distillation to retain old knowledge during continual learning.

Hardware and Deployment Constraints

Using trained models on devices with not much computing power can be hard.

Solution: Scientists use special techniques to make computer models run better on devices with limited resources.

Data Privacy and Security

When training computers to do complex tasks, it is essential to keep data private and ensure the computers are secure.

Solution: Employ federated learning, secure aggregation, or differential privacy techniques to protect data and model privacy.

Long Training Times

Training deep neural networks is like doing a challenging puzzle. It takes a lot of time to assemble the puzzle, especially if it is vast and has a lot of pieces.

Solution: Special tools like GPUs or TPUs can help us train our computers faster. We can also try using different computers simultaneously to make the training even quicker.

Exploding Memory Usage

Some models are too big and need a lot of space, so they are hard to use on regular computers.

Solution: Explore memory-efficient architectures, use gradient checkpointing, or consider model parallelism for training.

Learning Rate Scheduling

Setting an appropriate learning rate schedule can be challenging, affecting model convergence and performance.

Solution: Using special learning rate schedules can help make learning easier and faster. These schedules can be used to help teach things in a better way.

Avoiding Local Minima
Deep neural networks can get stuck in local minima during training, impacting the model's final performance.
Solution: Using unique strategies like simulated annealing, momentum-based optimization, and evolutionary algorithms can help us escape difficult spots.
Unstable Loss Surfaces
Finding the best way to do something can be very hard when there are many different options because the surface it is on is complicated and bumpy.
Solution: Utilize weight noise injection, curvature-based optimization, or geometric methods to stabilize loss surfaces.

Ill-Conditioned Matrix

In neural network the adjustments of weights computation and calculation in hidden layer when calculate in matrix form it simply tells us the characteristics of the matrix in terms of further computations and calculations, or formally it can be defined as a measure of how much the output value of the function can change for a small change in the input argument.

A matrix is said to be Ill-conditioned if the condition number is very high, so for a small change in the input function/the Hessian matrix (The Hessian Matrix is a square matrix of second ordered partial derivatives of a scalar function. It is of immense use in linear algebra as well as for determining points of local maxima or minima.) we will end up getting outputs with high variance

Basic Algorithms

Gradient Descent is an iterative optimization process that searches for an objective function's optimum value (Minimum/Maximum). It is one of the most used methods for changing a model's parameters in order to reduce a cost function in machine learning projects.

The primary goal of gradient descent is to identify the model parameters that provide the maximum accuracy on both training and test datasets

Stochastic Gradient Descent (SGD) is a variant of the Gradient Descent algorithm that is used for optimizing machine learning models. It addresses the computational inefficiency of traditional Gradient Descent methods when dealing with large datasets in machine learning projects.

In SGD, instead of using the entire dataset for each iteration, only a single random training example (or a small batch) is selected to calculate the gradient and update the model parameters. This random selection introduces randomness into the optimization process, hence the term “stochastic” in stochastic Gradient Descent

The advantage of using SGD is its computational efficiency, especially when dealing with large datasets. By using a single example or a small batch, the computational cost per iteration is significantly reduced compared to traditional Gradient Descent methods that require processing the entire dataset.

Stochastic Gradient Descent Algorithm

- **Initialization**: Randomly initialize the parameters of the model.
- **Set Parameters**: Determine the number of iterations and the learning rate (alpha) for updating the parameters.
- **Stochastic Gradient Descent Loop**: Repeat the following steps until the model converges or reaches the maximum number of iterations:

  a. Shuffle the training dataset to introduce randomness.

  b. Iterate over each training example (or a small batch) in the shuffled order.

  c. Compute the gradient of the cost function with respect to the model parameters using the current training example (or batch).

  d. Update the model parameters by taking a step in the direction of the negative gradient, scaled by the learning rate.

  e. Evaluate the convergence criteria, such as the difference in the cost function between iterations of the gradient.

- **Return Optimized Parameters**: Once the convergence criteria are met or the maximum number of iterations is reached, return the optimized model parameters.

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

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

Stochastic gradient descent (SGD) with momentum

the momentum algorithm introduces a variable v that plays the role of velocity—it is the direction and speed at which the parameters move through parameter space. The velocity is set to an exponentially decaying average of the negative gradient. The name momentum derives from a physical analogy, in which the negative gradient is a force moving a particle through parameter space, according to Newton's laws of motion. Momentum in physics is mass times velocity. In the momentum learning algorithm, we assume unit mass, so the velocity vector v may also be regarded as the momentum of the particle The algorithm is balanced with momentum and steps velocity is added as

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

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

SGD is generally noisier than typical Gradient Descent, it usually took a higher number of iterations to reach the minima, because of the randomness in its descent. Even though it requires a higher number of iterations to reach

the minima than typical Gradient Descent, it is still computationally much less expensive than typical Gradient Descent.

Parameter Initialization Strategies

Training algorithms for deep learning models are iterative in nature and require the specification of an initial point. This is extremely crucial as it often decides whether or not the algorithm converges and if it does, then does the algorithm converge to a point with high cost or low cost.

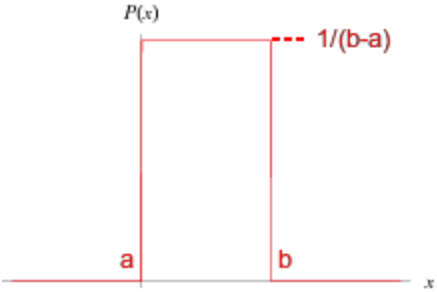
We have limited understanding of neural network optimization but the one property that we know with complete certainty is that the initialization should **break symmetry**. This means that if two hidden units are connected to the same input units, then these should have different initialization or else the gradient would update both the units in the same way and we don't learn anything new by using an additional unit. The idea of having each unit learn something different motivates random initialization of weights which is also computationally cheaper.

Biases are often chosen heuristically (zero mostly) and only the weights are randomly initialized, almost always from a Gaussian or uniform distribution. The scale of the distribution is of utmost concern. Large weights might have better symmetry-breaking effect but might lead to chaos (extreme sensitivity to small perturbations in the input) and exploding values during forward & back propagation. As an example of how large weights might lead to chaos, consider that there's a slight noise adding ϵ to the input. Now,

we if did just a simple linear transformation like `W * x`, the ϵ noise would add a factor of W * ϵ to the output. In case the weights are high, this ends up making a significant contribution to the output. SGD and its *variants* tend to halt in areas near the initial values, thereby expressing a prior that the path to the final parameters from the initial values is discoverable by steepest descent algorithms.

Various suggestions have been made for appropriate initialization of the parameters. The most commonly used ones include sampling the weights of each fully-connected layer having `m` inputs and `n` outputs uniformly from the following distributions:

- $U(-1 / \sqrt{m}, 1 / \sqrt{m})$
- $U(- \sqrt{6} / (m+n), \sqrt{6} / (m+n))$



$U(a, b)$ represents the uniform distribution where the probability of each value between a and b, a and b inclusive, is 1/(b-a). The probability of every other value is 0.

These initializations have already been incorporated into the most commonly used Deep Learning frameworks nowadays so that you can just specify which initializer to use and the framework takes care of sampling appropriately. For e.g. *Keras*, which is a very famous deep learning framework, has a module called *initializers*, where the

second distribution (among the 2 mentioned above) is implemented as `glorot_uniform`.

One drawback of using 1 / √m as the standard deviation is that the weights end up being small when a layer has too many input/output units. Motivated by the idea to have the total amount of input to each unit independent of the number of input units *m,* **Sparse initialization** sets each unit to have exactly *k* non-zero weights. However, it takes a long time for GD to correct *incorrect large values* and hence, this initialization might cause problems.

If the weights are too small, the range of activations across the mini-batch will shrink as the activations propagate forward through the network.By repeatedly identifying the first layer with unacceptably small activations and increasing its weights, it is possible to eventually obtain a network with reasonable initial activations throughout.

The biases are relatively easier to choose. Setting the biases to zero is compatible with most weight initialization schemes except for a few cases .

Algorithms with Adaptive Learning Rates

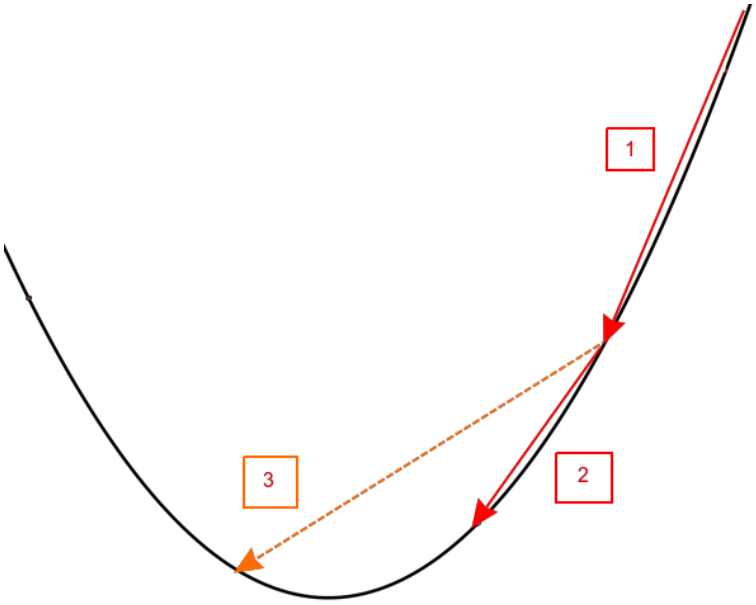
- **AdaGrad**: it is important to incrementally decrease the learning rate for faster convergence. Instead of manually reducing the learning rate after each (or several) epochs, a better approach is to adapt the learning rate as the training progresses. This can be done by scaling the learning rates

of *each model parameter* individually inversely proportional to the square root of the sum of historical squared values of the gradient. In the parameter update equation below, **r** is initialized with 0 and the multiplication in the update step happens element-wise as mentioned. Since the gradient value would be different for each parameter, the learning rate is scaled differently for each parameter too.

- Thus, those parameters having a large gradient have a large decrease in the learning rate as the learning rate might be too high leading to oscillations or it might be approaching the minima but having large learning rate might cause it to jump over the minima as explained in the figure below, because of which the learning rate should be decreased for better convergence, while those with small gradients have a small decrease in the learning rate as they might have already approached their respective minima and should not be pushed away from that. Even if they have not, reducing the learning rate too much would reduce the gradient even further leading to slower learning.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.

AdaGrad parameter update equation.



This figure illustrates the need to reduce the learning rate if gradient is large in case of a single parameter. 1) One step of gradient descent representing a large gradient value. 2) Result of reducing the learning rate — moves towards the minima 3) Scenario if the learning rate was not reduced — it would have jumped over the minima.

However, accumulation of squared gradients from the very beginning can lead to excessive and premature decrease in the learning rate. Consider that we had a model with only 2 parameters (for simplicity) and both the initial gradients are 1000.

After some iterations, the gradient of one of the

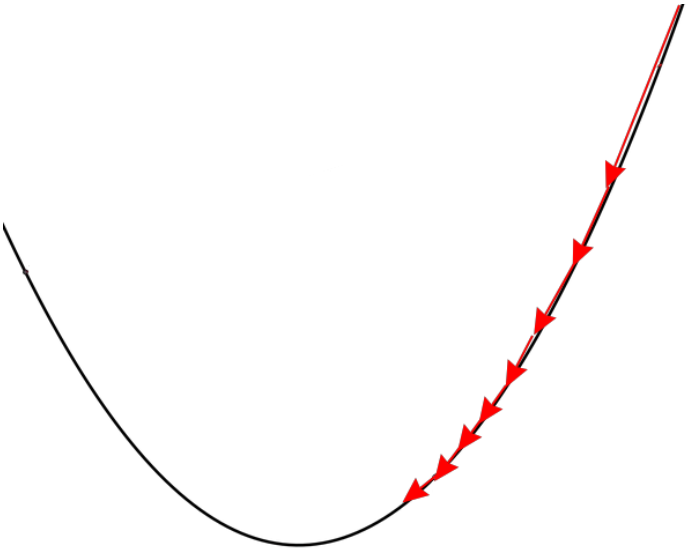


Figure explaining the problem with AdaGrad. Accumulated gradients can cause the learning rate to be reduced far too much in the later stages leading to slower learning.

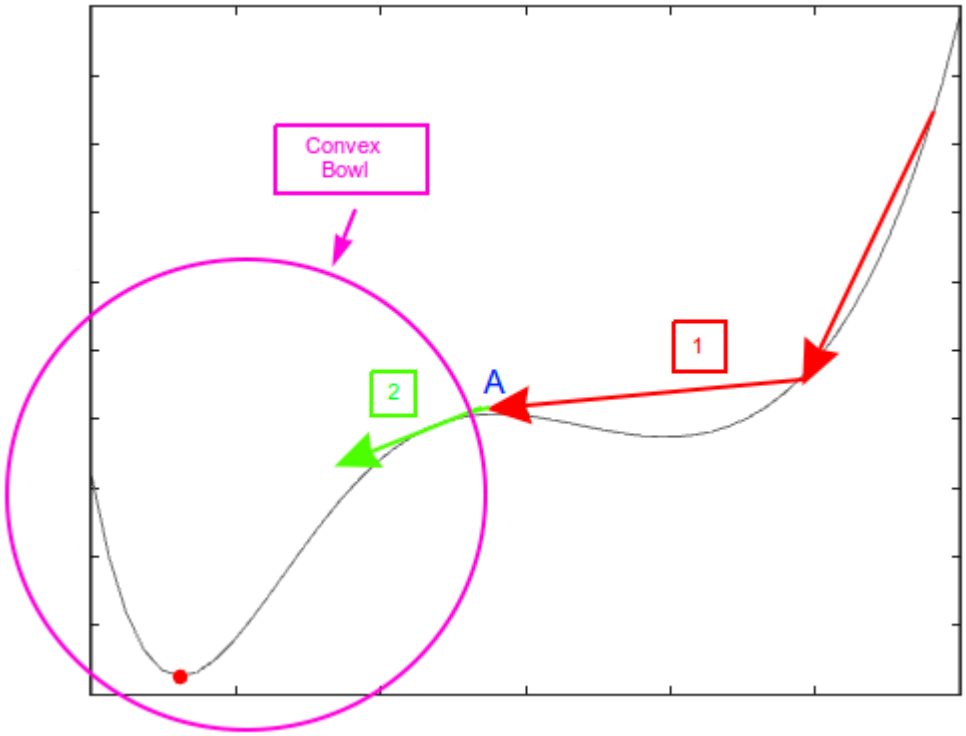
parameters has reduced to 100 but that of the other parameter is still around 750. However, because of the *accumulation at each update*, the accumulated gradient would still have almost the same value. For e.g. let the accumulated gradients at each step for the *Parameter 1* be `1000 + 900 + 700 + 400 + 100 = 3100, 1/3100=0.0003` and that for *Parameter 2* be: `1000 + 900 + 850 + 800 + 750 = 4300, 1/4300 = 0.0002`. This would lead to a similar decrease in the learning rates for both the parameters, even though the parameter having the lower gradient might have its learning rate reduced too much leading to slower learning.

- **RMSProp**: RMSProp addresses the problem caused by accumulated gradients in AdaGrad. It modifies the gradient accumulation step to an exponentially weighted moving average in order to discard history from the extreme past. The RMSProp update is given by:

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. $(\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)

ρ is the weighing used for exponential averaging. As more updates are made, the contribution of past gradient values are reduced since **ρ** < 1 and **ρ > ρ² >ρ³** …

This allows the algorithm to converge rapidly after finding a convex bowl, as if it were an instance of AdaGrad initialized within that bowl. . Consider the figure below. The region represented by **1** indicates usual RMSProp parameter updates as given by the update equation, which is nothing but exponentially averaged AdaGrad updates. Once the optimization process lands on A, it essentially lands at the top of a convex bowl. At this point, intuitively, all the updates before A can be seen to be forgotten due to the exponential averaging and it can be seen as if (exponentially averaged) AdaGrad updates start from point A onwards.



Intuition behind RMSProp. 1) Usual parameter updates 2) Once it reaches the convex bowl, exponentially weighted averaging would cause the effect of earlier gradients to reduce and to simplify, we can assume their contribution to be zero. This can be seen as if AdaGrad had been used with the training initiated inside the convex bowl

- **Adam**: Adapted from "*adaptive moments*", it focuses on combining RMSProp and Momentum. Firstly, it views Momentum as an estimate of the first-order moment and RMSProp as that of the second moment. The weight update for Adam is given by:

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
$t \leftarrow t + 1$
Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$ (operations applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

Secondly, since $\boldsymbol{s}$ and $\boldsymbol{r}$ are initialized as zeros, the authors observed a bias during the initial steps of training thereby adding a correction term for both the moments to account for their initialization near the origin. As an example of what the effect of this bias correction is, we'll look at the values of $\boldsymbol{s}$ and $\boldsymbol{r}$ for a single parameter (in which case everything is now represented as a scalar). Let's first understand what would happen if there was no bias correction. Since *s (*notice that this is not in bold as we are looking at the value for a single parameter and the *s* here is a scalar*)* is initialized as zero, after the first iteration, the value of *s* would be (1 — ρ1) * *g* and that

of r would be $(1 - \rho 2) * g^2$. The *preferred values* for $\rho 1$ and $\rho 2$ are 0.9 and 0.99 respectively. Thus, the initial values of s and r are pretty small and this gets compounded as the training progress. However, if we now use bias correction, after the first iteration, the value of s is just g and that of r is just g^2. This gets rid of the bias that occurs in the initial phase of training. A major advantage of Adam is that it's *fairly robust* to the choice of these hyperparameters, i.e. $\rho 1$ and $\rho 2$.

3. Approximate Second-Order Methods

The optimization algorithms that we've looked at till now involved computing only the first derivative. But there are many methods which involve higher order derivatives as well. The main problem with these algorithms are that they are not practically feasible in their vanilla form and so, certain methods are used to approximate the values of the derivatives. We explain three such methods, all of which use empirical risk as the objective function:

- **Newton's Method**: This is the most common higher-order derivative method used. It makes use of the curvature of the loss function via its second-order derivative to arrive at the optimal point. Using the second-order Taylor Series expansion to approximate $J(\theta)$ around a point θo and ignoring derivatives of order greater than 2 (this has already been discussed in previous chapters), we get:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

We know that we get a critical point for any function *f*(*x*) by solving for `f'(x) = 0`. We get the following critical point of the above equation (refer to the *Appendix* for proof):

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

For quadratic surfaces (i.e. where cost function is quadratic), this directly gives the optimal result in one step whereas gradient descent would still need to iterate. However, for surfaces that are not quadratic, *as long as the Hessian remains positive definite*, we can obtain the optimal point through a 2-step iterative process — 1) Get the inverse of the Hessian and 2) update the parameters.

Saddle points are problematic for Newton's method. If all the eigenvalues are not positive, Newton's method might cause the updates to move in the wrong direction. A way to avoid this is to add regularization:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - [H\left(f(\boldsymbol{\theta}_0)\right) + \alpha \boldsymbol{I}]^{-1} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_0)$$

However, if there is a strong negative curvature i.e. the eigenvalues are largely negative, α needs to be sufficiently high to offset the negative eigenvalues in which case the Hessian becomes dominated

by the diagonal matrix. This leads to an update which becomes the standard gradient divided by α:

$$\mathcal{H} + \alpha I \approx \alpha I$$

$$\Rightarrow \theta^* \approx \theta_0 - [\alpha I]^{-1} \nabla_\theta f(\theta_0)$$

$$\Rightarrow \theta^* \approx \theta_0 - \frac{\nabla_\theta f(\theta_0)}{\alpha}$$

Another problem restricting the use of Newton's method is the computational cost. It takes $O(k^3)$ time to calculate the inverse of the Hessian where k is the number of parameters. It's not uncommon for Deep Neural Networks to have about a million parameters and since the parameters are updated every iteration, this inverse needs to be calculated at every iteration, which is not computationally feasible.

- **Conjugate Gradients**: One weakness of the method of steepest descent (i.e. GD) is that *line searches* happen along the direction of the gradient. Suppose the previous search direction is *d*(t-1). Once the search terminates (which it does when the gradient along the current gradient direction vanishes) at the minimum, the next search direction, d(t) is given by the gradient at that point, which is orthogonal to d(t-1) (because if it's not orthogonal, it'll have some component along d(t-1) which cannot be true as at the minimum, the gradient along d(t-1) has vanished).
  Upon getting the minimum along the current search direction, the minimum along the previous search direction is not

preserved, undoing, in a sense, the progress made in previous search direction.

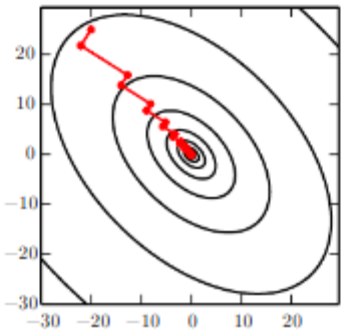


Figure 8.6: The method of steepest descent applied to a quadratic cost surface. The method of steepest descent involves jumping to the point of lowest cost along the line defined by the gradient at the initial point on each step. This resolves some of the problems seen with using a fixed learning rate in figure 4.6, but even with the optimal step size the algorithm still makes back-and-forth progress toward the optimum. By definition, at the minimum of the objective along a given direction, the gradient at the final point is orthogonal to that direction.

In the method of conjugate gradients, we seek a search direction that is conjugate to the previous line search direction:

$$\boldsymbol{d}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \beta_t \boldsymbol{d}_{t-1},$$

Now, the previous search direction contributes towards finding the next search direction.

with *d*(t) and *d*(t-1) being **conjugates** if `d(t)' H d(t-1) = 0`. βt decides how much of *d*(t-1) is added back to the current search direction. There are two popular choices for βt — Fletcher-Reeves and Polak-Ribière. These discussions assumed the cost function to be quadratic where the conjugate directions ensure that the gradient along the previous direction does not increase in magnitude. To extend the concept to work for training neural networks, there is one additional change. Since it's no longer quadratic, there's no guarantee anymore than the conjugate direction would preserve the minimum in the previous search directions. Thus, the algorithm

includes occasional resets where the method of conjugate gradients is restarted with line search along the unaltered gradient.

- **BFGS:** This algorithm tries to bring the advantages of Newton's method without the additional computational burden by approximating the inverse of H by $M(t)$, which is iteratively refined using low-rank updates. Finally, line search is conducted along the direction $M(t)g(t)$. However, BFGS requires storing the matrix $M(t)$ which takes $O(n^2)$ memory making it infeasible. An approach called Limited Memory BFGS (L-BFGS) has been proposed to tackle this infeasibility by computing the matrix $M(t)$ using the same method as BFGS but assuming that $M(t-1)$ is the identity matrix.

4. Optimization Strategies and Meta-Algorithms

- **Batch Normalization:** Batch normalization (BN) is one of the most exciting innovations in Deep learning that has significantly stabilized the learning process and allowed faster convergence rates. The intuition behind batch normalization is as follows: Most of the Deep Learning networks are compositions of many layers (or functions) and the gradient with respect to one layer is taken considering the other layers to be constant. However, in practise all the layers are updated simultaneously and this can lead to unexpected results. For example, let $y^* = x\, W^1\, W^2 \ldots W^{10}$. Here, y^* is a linear function of x but not a linear function of the weights. Suppose the gradient is given by $\boldsymbol{g}$ and we now intend to reduce y^* by 0.1. Using first-order Taylor Series approximation, taking a step

of $\epsilon \mathbf{g}$ would reduce y* by $\epsilon \mathbf{g}' \mathbf{g}$. Thus, ϵ should be $\mathbf{0.1/(g' g)}$ just using the first-order information. However, higher order effects also creep in as the updated y* is given by:

$$x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2) \dots (w_l - \epsilon g_l)$$

An example of a second-order term would be $\epsilon^2 \mathbf{g}_1 \mathbf{g}_2 \prod \mathbf{w}_i$. $\prod \mathbf{w}_i$ can be negligibly small or exponentially high depending on whether the individual weights are less than or greater than 1. Since the updates to one layer is so strongly dependent on the other layers, choosing an appropriate learning rate is tough. Batch normalization takes care of this problem by using an efficient reparameterization of almost any deep network. Given a matrix of activations, **H**, the normalization is given by: $\mathbf{H' = (H-\mu) / \sigma}$, where the subtraction and division is *broadcasted*.

$$\boldsymbol{\mu} = \frac{1}{m} \sum_i \boldsymbol{H}_{i,:}$$

$$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m} \sum_i (\boldsymbol{H} - \boldsymbol{\mu})_i^2},$$

δ is added to ensure that σ is not equal to 0.

Going back to the earlier example of y*, let the activations of layer l be given by $\mathbf{h}(l\text{-}1)$. Then $\mathbf{h}(l\text{-}1) = x\, \mathbf{W}_1 \mathbf{W}_2 \dots \mathbf{W}(l\text{-}1)$. Now, if x is drawn from a unit Gaussian, then $\mathbf{h}(l\text{-}1)$ also comes from a Gaussian, however, not of zero mean and unit variance, as it is a linear transformation of x. BN makes it zero mean and unit variance. Therefore, y* = $\mathbf{W}l\, \mathbf{h}(l\text{-}1)$ and thus, the learning now becomes much

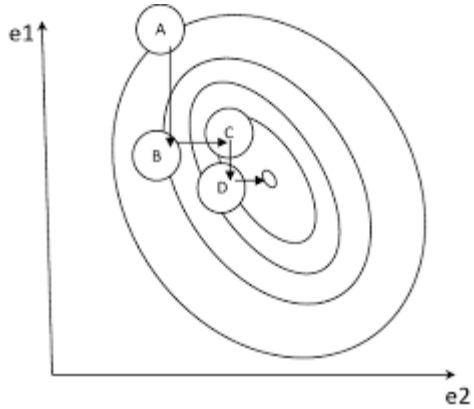
simpler as the parameters at the lower layers mostly do not have any effect. This simplicity was definitely achieved by rendering the lower layers useless. However, in a realistic deep network with non-linearities, the lower layers remain useful. Finally, the complete reparameterization of BN is given by replacing **H** with **γH' + β**. This is done to retain its expressive power and the fact that the mean is solely determined by **XW**. Also, among the choice of normalizating **X** or **XW + B**, the authors recommend the latter, specifically **XW**, since **B** becomes redundant because of **β**. Practically, this means that when we are using the Batch Normalization layer, the biases should be turned off. In a deep learning framework like *Keras*, this can be done by setting the parameter *use_bias=False* in the Convolutional layer.

- **Coordinate Descent:** Generally, a single weight update is made by taking the gradient with respect to every parameter. However, in cases where some of the parameters might be independent (discussed below) of the remaining, it might be more efficient to take the gradient with respect to those independent sets of parameters separately for making updates. Let me clarify that with an example. Suppose we have the following cost function:

$$J(\boldsymbol{H}, \boldsymbol{W}) = \sum_{i,j} |H_{i,j}| + \sum_{i,j} \left(\boldsymbol{X} - \boldsymbol{W}^\top \boldsymbol{H} \right)^2_{i,j}$$

This cost function describes the learning problem called sparse coding. Here, $\boldsymbol{H}$ refers to the sparse representation of $\boldsymbol{X}$ and $\boldsymbol{W}$ is

the set of weights used to linearly decode $\boldsymbol{H}$ to retrieve $\boldsymbol{X}$. An explanation of why this cost function enforces the learning of a sparse representation of $\boldsymbol{X}$ follows. The first term of the cost function penalizes values far from 0 (positive or negative because of the modulus, $|\mathbf{H}|$, operator. This enforces most of the values to be 0, thereby *sparse*. The second term is pretty self-explanatory in that it compensates the difference between $\boldsymbol{X}$ and $\boldsymbol{H}$ being linearly transformed by $\boldsymbol{W}$, thereby enforcing them to take the same value. In this way, $\boldsymbol{H}$ is now learned as a sparse "representation" of $\boldsymbol{X}$. The cost function generally consists of additionally a *regularization* term like *weight decay*, which has been avoided for simplicity. Here, we can divide the entire list of parameters into two sets, $\boldsymbol{W}$ and $\boldsymbol{H}$. Minimizing the cost function with respect to any of these sets of parameters is a convex problem. **Coordinate Descent (CD)** refers to minimizing the cost function with respect to only 1 parameter at a time. It has been shown that repeatedly cycling through all the parameters, we are guaranteed to arrive at a local minima. If instead of 1 parameter, we take a set of parameters as we did before with $\boldsymbol{W}$ and $\boldsymbol{H}$, it is called **block coordinate descent** (the interested reader should explore *Alternating Minimization*). CD makes sense if either the parameters are clearly separable into independent groups or if optimizing with respect to certain set of parameters is more efficient than with respect to others.



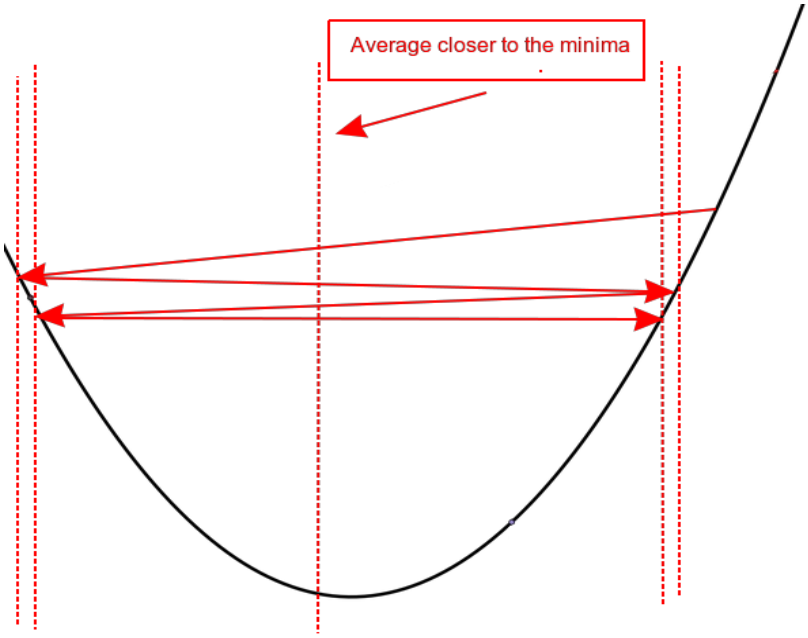
The points A, B, C and D indicates the locations in the parameter space where coordinate descent landed after each gradient step.

Coordinate descent may *fail terribly* when one variable influences the optimal value of another variable.

- **Polyak Averaging:** Polyak averaging consists of averaging several points in the parameter space that the optimization algorithm traverses through. So, if the algorithm encounters the points $\theta(1)$, $\theta(2)$, ... during optimization, the output of Polyak averaging is:

$$\hat{\boldsymbol{\theta}}^{(t)} = \frac{1}{t}\sum_i \boldsymbol{\theta}^{(i)}$$

The figure below explains the intuition behind Polyak averaging:



The optimization algorithm might oscillate back and forth across a valley without ever reaching the minima. However, the average of those points should be closer to the bottom of the valley.

Most optimization problems in deep learning are non-convex where the path taken by the optimization algorithm is quite complicated and it might happen that a point visited in the distant past might be quite far from the current point in the parameter space. Thus, including such a point in the distant past might not be useful, which is why an exponentially decaying running average is taken. This scheme where the recent iterates are weighted more than the past ones is called **Polyak-Ruppert Averaging**:

$$\hat{\boldsymbol{\theta}}^{(t)} = \alpha \hat{\boldsymbol{\theta}}^{(t-1)} + (1-\alpha)\boldsymbol{\theta}^{(t)}$$

- **Supervised Pre-training:** Sometimes it's hard to directly train to solve for a specific task. Instead it might be better to

train for solving a simpler task and use that as an initialization point for training to solve the more challenging task.

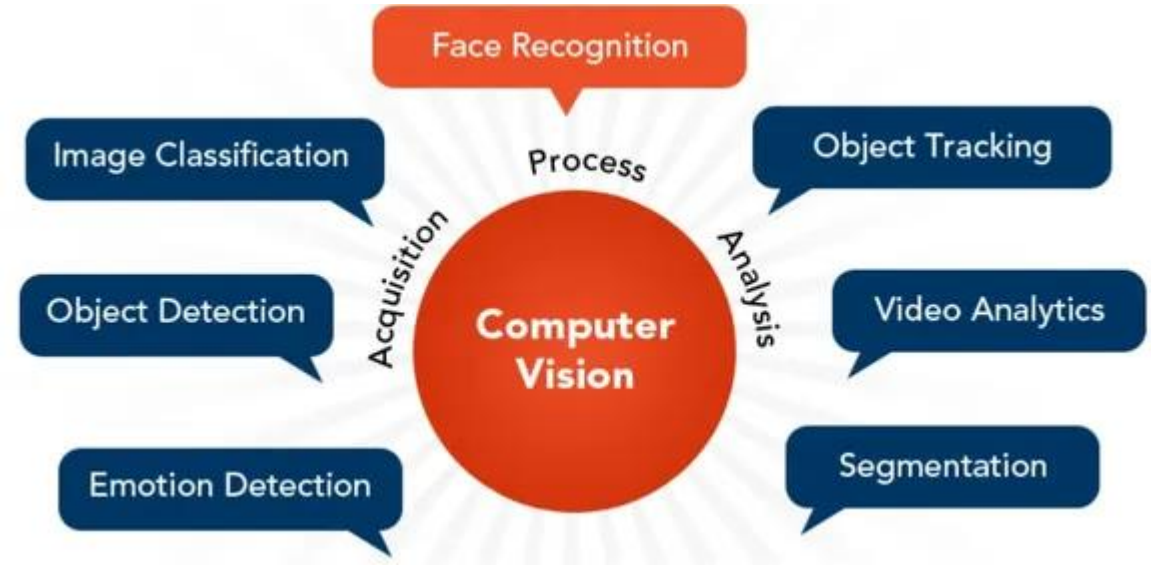
Applications: Large-Scale Deep Learning : Computer Vision, Speech Recognition, Natural Language Processing

Common Applications of Deep Learning

Deep learning has many uses in many fields, and its potential grows. Let's analyze a few of artificial intelligence's widespread profound learning uses.

- Image Recognition and Computer Vision
- Natural Language Processing (NLP)
- Speech Recognition and Voice Assistants
- Recommendation Systems
- Autonomous Vehicles
- Healthcare and Medical Imaging
- Fraud Detection and Cybersecurity
- Gaming and Virtual Reality

Image Recognition and Computer Vision

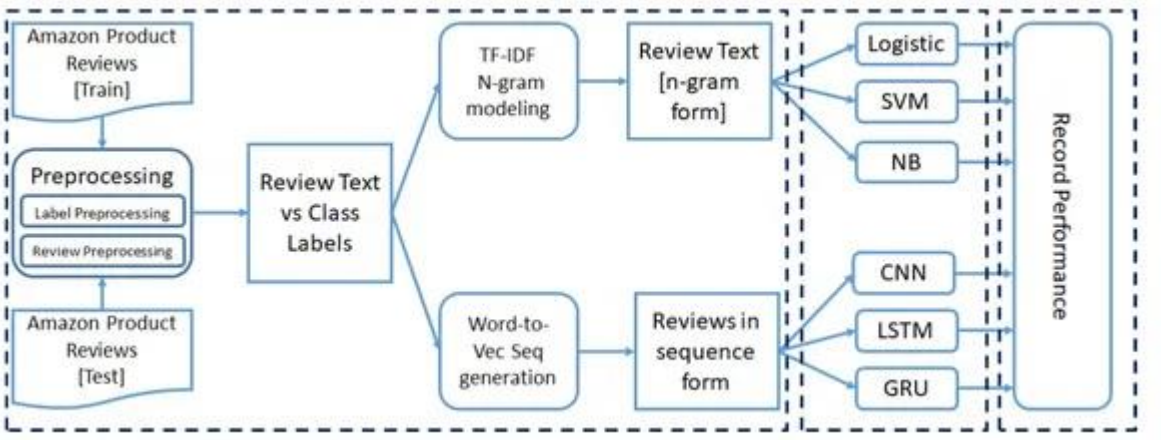


The performance of image recognition and computer vision tasks has significantly improved due to deep learning. Computers can now reliably classify and comprehend images owing to training deep neural networks on enormous datasets, opening up a wide range of applications.

A smartphone app that can instantaneously determine a dog's breed from a photo and self-driving cars that employ computer vision algorithms to detect pedestrians, traffic signs, and other roadblocks for safe navigation are two examples of this in practice.

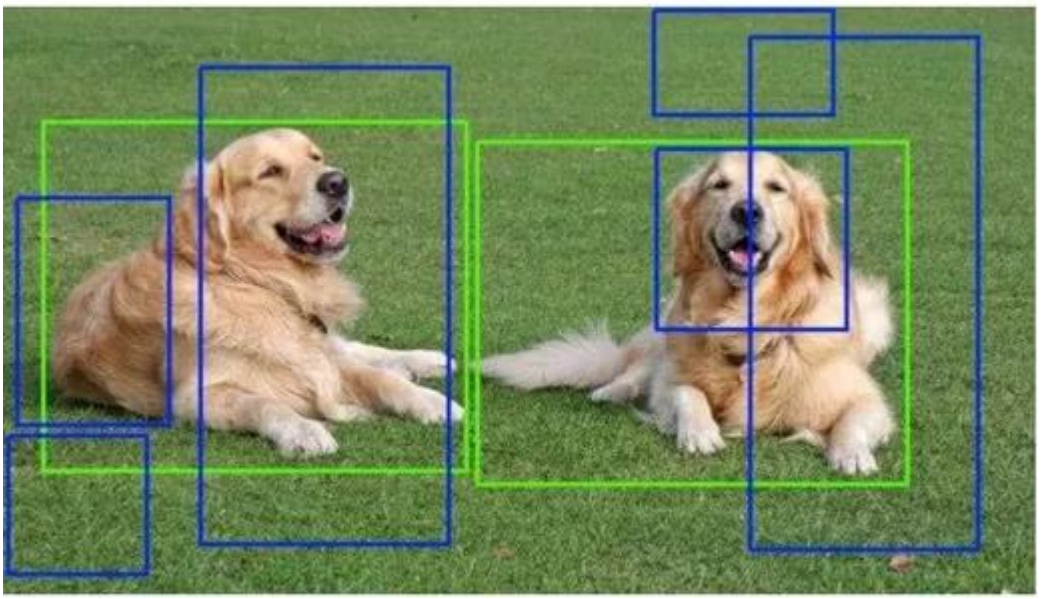
Deep Learning Models for Image Classification

The process of classifying photos entails giving them labels based on the content of the images. Convolutional neural networks (CNNs), one type of deep learning model, have performed exceptionally well in this context. They can categorize objects, situations, or even specific properties within an image by learning to recognize patterns and features in visual representations.



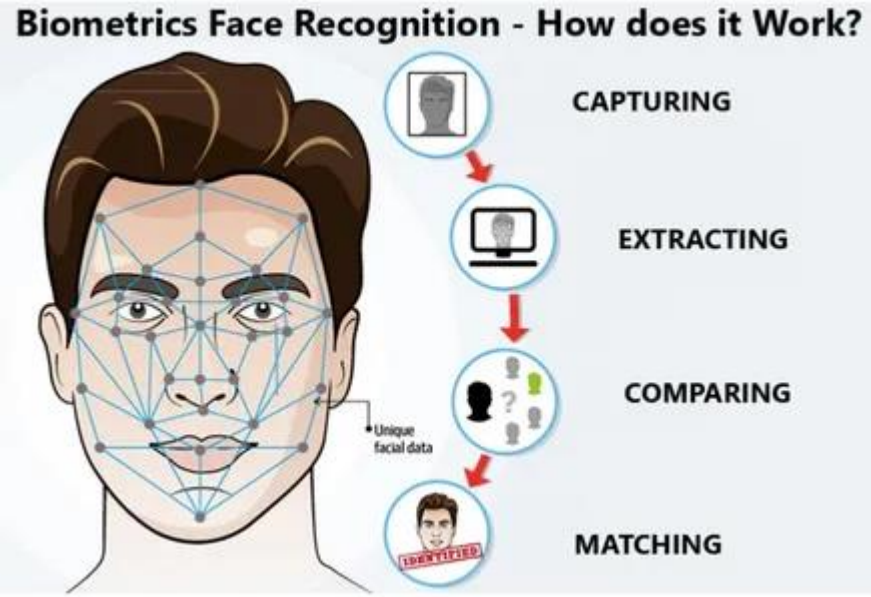
Object Detection and Localization using Deep Learning

Object detection and localization go beyond image categorization by identifying and locating various things inside an image. Deep learning methods have recognized and localized objects in real-time, such as You Only Look Once (YOLO) and region-based convolutional neural networks (R-CNNs). This has uses in robotics, autonomous cars, and surveillance systems, among other areas.



Applications in Facial Recognition and Biometrics

Deep learning has completely changed the field of facial recognition. Hence, allowing for the precise identification of people using their facial features. Security systems, access control, monitoring, and law enforcement use facial recognition technology. Deep learning methods have also been applied in biometrics for functions including voice recognition, iris scanning, and fingerprint recognition.



Natural Language Processing (NLP)

Natural language processing (NLP) aims to make it possible for computers to comprehend, translate, and create human language. NLP has substantially advanced primarily to deep learning, making strides in several language-related activities. Virtual voice assistants like Apple's Siri and Amazon's Alexa, who can comprehend spoken orders and questions, are a practical illustration of this.

Deep Learning for Text Classification and Sentiment Analysis

Text classification entails classifying text materials into several groups or divisions. Deep learning models like recurrent neural networks (RNNs) and long short-term memory (LSTM) networks have been frequently used for text categorization tasks. To ascertain the sentiment or opinion expressed in a text, whether good, negative, or neutral, sentiment analysis is a widespread use of text categorization.

Language Translation and Generation with Deep Learning

Machine translation systems have considerably improved because of deep learning. Deep learning-based neural machine translation (NMT) models have been shown to perform better when converting text across multiple languages. These algorithms can gather contextual data and generate more precise and fluid translations. Deep learning models have also been applied to creating news stories, poetry, and other types of text, including coherent paragraphs.

Question Answering and Chatbot Systems Using Deep Learning

Deep learning is used by chatbots and question-answering programs to recognize and reply to human inquiries. Transformers and attention mechanisms, among other deep learning models, have made tremendous progress in understanding the context and semantics of questions and producing pertinent answers. Information retrieval systems, virtual assistants, and customer service all use this technology.

Speech Recognition and Voice Assistants

The creation of voice assistants that can comprehend and respond to human speech and the advancement of speech recognition systems have significantly benefited from deep learning. A real-world example is using your smartphone's voice recognition feature to dictate messages rather than typing them and asking a smart speaker to play your favorite tunes or provide the weather forecast.

Deep Learning Models for Automatic Speech Recognition

Systems for automatic speech recognition (ASR) translate spoken words into written text. Recurrent neural networks and attention-based models, in particular, have substantially improved ASR accuracy. Better voice commands, transcription services, and accessibility tools for those with speech difficulties are the outcome. Some examples are voice search features in search engines like Google, Bing, etc.

Voice Assistants Powered by Deep Learning Algorithms

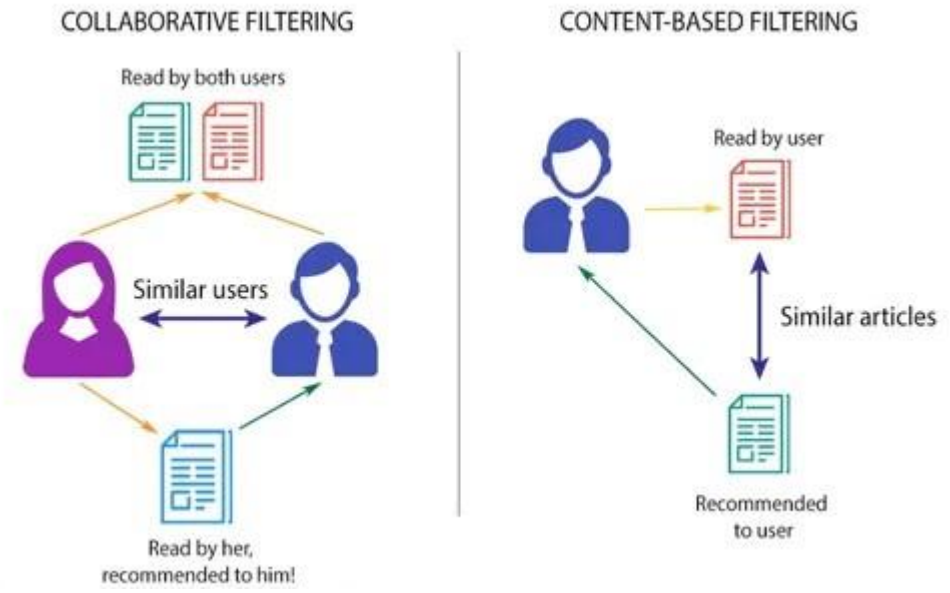
Daily, we rely heavily on voice assistants like Siri, Google Assistant, and Amazon Alexa. Guess what drives them? Deep learning it is. Deep learning

techniques are used by these intelligent devices to recognize and carry out spoken requests. The technology also enables voice assistants to recognize speech, decipher user intent, and deliver precise and pertinent responses thanks to deep learning models.

Applications in Transcription and Voice-Controlled Systems

Deep learning-based speech recognition has applications in transcription services, where large volumes of audio content must be accurately converted into text. Voice-controlled systems, such as smart homes and in-car infotainment systems, utilize deep learning algorithms to enable hands-free control and interaction through voice commands.

Recommendation Systems



Recommendation systems use deep learning algorithms to offer people personalized recommendations based on their tastes and behavior.

Deep Learning-Based Collaborative Filtering

A standard method used in recommendation systems to suggest products/services to users based on how they are similar to other users is collaborative filtering. Collaborative filtering has improved accuracy and performance thanks to deep learning models like matrix factorization and deep autoencoders, which have produced more precise and individualized recommendations.

Personalized Recommendations Using Deep Neural Networks

Deep neural networks have been used to identify intricate links and patterns in user behavior data, allowing for more precise and individualized suggestions. Deep learning algorithms can forecast user preferences and make relevant product, movie, or content recommendations by looking at user interactions, purchase history, and demographic data. An instance of this is when streaming services recommend films or TV shows based on your interests and history.

Applications in E-Commerce and Content Streaming Platforms

Deep learning algorithms are widely employed to fuel recommendation systems in e-commerce platforms and video streaming services like Netflix and Spotify. These programs increase user pleasure and engagement by assisting users in finding new goods, entertainment, or music that suits their tastes and preferences.

Autonomous Vehicles

Deep learning has significantly impacted how well autonomous vehicles can understand and navigate their surroundings. These vehicles can analyze enormous volumes of sensor data in real-time using powerful deep learning algorithms. Thus, enabling them to make wise decisions, navigate challenging routes, and guarantee the safety of passengers and pedestrians. This game-changing technology has prepared the path for a time when driverless vehicles will completely change how we travel.

Deep Learning Algorithms for Object Detection and Tracking

Autonomous vehicles must perform crucial tasks, including object identification and tracking, to recognize and monitor objects like pedestrians, cars, and traffic signals. Convolutional and recurrent neural networks (CNNs) and other deep learning algorithms have proved essential in obtaining high accuracy and real-time performance in object detection and tracking.

Deep Reinforcement Learning for Decision-Making in Self-Driving Cars

Autonomous vehicles are designed to make complex decisions and navigate various traffic circumstances using deep reinforcement learning. This technology is profoundly used in self-driving cars manufactured by companies like Tesla. These vehicles can learn from historical driving data and adjust to changing road conditions using deep neural networks. Self-driving cars demonstrate this in practice, which uses cutting-edge sensors and artificial intelligence algorithms to navigate traffic, identify impediments, and make judgments in real time.

Applications in Autonomous Navigation and Safety Systems

The development of autonomous navigation systems that decipher sensor data, map routes, and make judgments in real time depends heavily on deep learning techniques. These systems focus on collision avoidance, generate lane departure warnings, and offer adaptive cruise control to enhance the general safety and dependability of the vehicles.

Healthcare and Medical Imaging

Deep learning has shown tremendous potential in revolutionizing healthcare and medical imaging by assisting in diagnosis, disease detection, and patient care. Revolutionizing diagnostics using AI-powered algorithms that can precisely identify early-stage tumors from medical imaging is an example of how to do this. This will help with prompt treatment decisions and improve patient outcomes.

Deep Learning for Medical Image Analysis and Diagnosis

Deep learning algorithms can glean essential insights from the enormous volumes of data that medical imaging systems produce. Convolutional neural networks (CNNs) and generative adversarial networks (GANs) are examples of deep learning algorithms. They can be effectively used for tasks like tumor identification, radiology image processing, and histopathology interpretation.

Predictive Models for Disease Detection and Prognosis

Deep learning models can analyze electronic health records, patient data, and medical pictures to create predictive models for disease detection, prognosis, and treatment planning.

Applications in Medical Research and Patient Care

Deep learning can revolutionize medical research by expediting the development of new drugs, forecasting the results of treatments, and assisting clinical decision-making. Additionally, deep learning-based systems can also improve medical care by helping with diagnosis, keeping track of patients' vital signs, and making unique suggestions for dietary changes and preventative actions.

Fraud Detection and Cybersecurity

Deep learning has become essential in detecting anomalies, identifying fraud patterns, and strengthening cybersecurity systems.

Deep Learning Models for Anomaly Detection

These systems shine when finding anomalies or outliers in large datasets. By learning from typical patterns, deep learning models may recognize unexpected behaviors, network intrusions, and fraudulent operations. These methods are used in network monitoring, cybersecurity systems, and financial transactions. JP Morgan Chase, PayPal, and other businesses are just a few that use these techniques.

Deep Neural Networks in Fraud Prevention and Cybersecurity

In fraud prevention systems, deep neural networks have been used to recognize and stop fraudulent transactions, credit card fraud, and identity theft. These algorithms examine user behavior, transaction data, and historical patterns to spot irregularities and notify security staff. This enables proactive fraud prevention and shields customers and organizations from financial loss. Organizations like Visa, Mastercard, and PayPal use deep neural networks. It helps improve their fraud detection systems and guarantees secure customer transactions.

Applications in Financial Transactions and Network Security

Deep learning algorithms are essential for preserving sensitive data, safeguarding financial transactions, and thwarting online threats. Deep learning-based cybersecurity systems can proactively identify and reduce potential hazards, protecting vital data and infrastructure by learning and adapting to changing attack vectors over time.

Gaming and Virtual Reality

Deep learning has significantly improved game AI, character animation, and immersive surroundings, benefiting the gaming industry and virtual reality experiences. A virtual reality game, for instance, can adjust and customize its gameplay experience based on the player's real-time motions and reactions by using deep learning algorithms.

Deep Learning in Game Development and Character Animation

Deep learning algorithms have produced more intelligent and lifelike video game characters. Game makers may create realistic animations, enhance character behaviors, and make more immersive gaming experiences by training deep neural networks on enormous datasets of motion capture data.

Deep Reinforcement Learning for Game AI and Decision-Making

Deep reinforcement learning has changed game AI by letting agents learn and enhance their gameplay through contact with the environment. Using

deep learning algorithms in game AI enables understanding optimal strategies, adaptation to various game circumstances, and challenging and captivating gaming.

Applications in Virtual Reality and Augmented Reality Experiences

Experiences in augmented reality (AR) and virtual reality (VR) have been improved mainly due to deep learning. Deep neural networks are used by VR and AR systems to correctly track and identify objects, detect movements and facial expressions, and build real virtual worlds, enhancing the immersiveness and interactivity of the user experience.

Conclusion

In artificial intelligence, deep learning has become a powerful technology that allows robots to learn and make wise decisions. Deep learning in AI has many uses, from image identification and NLP to cybersecurity and healthcare. It has substantially improved the capabilities of AI systems, resulting in innovations across various fields and the disruption of entire sectors. Common applications of deep learning in AI Accenture leverages deep learning within its AI initiatives to enhance data analytics, customer experience, and operational efficiency.