2. MACHINE LEARNING

Machine learning helps in analysis of massive data, which is really a treasure in this digital era generating huge volume of data. It generally provides faster and accurate results to uncover the more profitable situations and dangerous risks lying in the future in spite of consuming more time and resources to train the model correctly. Combination of machine learning, artificial intelligence and cognitive technology bring more powerful and unimaginable system to process large volume of data. This chapter elucidates machine learning concepts and its relationship with pattern classification by describing few machine learning algorithms used for pattern classification like decision tree, SVM, ANN, etc. This chapter also covers deep learning and various architectures of deep learning like CNN, RNN, DBN, etc. The needs of parameter optimization techniques that can be used to optimally initialize the model parameters are also presented.

2.1 MACHINE LEARNING AND PATTERN CLASSIFICATION

Machine learning aims in providing systems the capability to self-learn and gain knowledge from the data available and to improve its performance from experience without requiring to be explicitly programmed. It is a powerful application of Artificial Intelligence. The learning process in these algorithms starts with the set of observations or data which are the examples, instructions or direct experience happened so far and proceeds to discover the patterns in the underlying data and help in the process of efficient decision making in the future based on the earlier seen examples. This helps in making the computers to learn automatically, with nil human intervention or support. Generally machine learning algorithms fall in one of the four categories namely, unsupervised, supervised, semi-supervised and reinforcement machine learning algorithms.

Supervised machine learning algorithms are methods that learn from the past. They use labelled data called training set, where the system learns from the past data available and tries to model the function by mapping the available input and corresponding output. The system built is used to classify the new data in the future to the correct label or output value called target. The learning algorithm also verifies if the target produced is the intended output and also fine tunes the model accordingly, if the output is erroneous.

Unsupervised machine learning algorithms are used when the data in the training set is not labelled or classified. These algorithms learn the hidden commonalities or characteristics of groups of data available in the training data set and infer a function that is capable enough to describe the hidden structures in the data.

Semi-supervised machine learning algorithms lie in the intermediate of supervised and unsupervised learning approaches and are used when only a limited amount of data in the training set is labelled and the remaining are unlabelled. These types of systems considerably improve their accuracy by learning. This type of learning is suitable for cases where collecting more unlabelled data requires no additional resources or collecting labelled data requires more skill, time and additional relevant resources.

Reinforcement machine learning algorithms learns by interacting with its environment through actions and getting feedback for error and rewards. Theses comprise of machine and software agents, where the agents learn itself from the feedback received for the machine's action. The feedback is otherwise called ad reinforcement signal that helps in improving the performance of the system. The characteristics of these systems are trial and error search, and delayed reward.

There are various machine learning algorithms in the literature that are used for pattern classification out of which few are discussed here.

Support Vector Machines

A Support Vector Machine is discriminative classifier and a supervised learning algorithm. The algorithm creates a discriminative classifier by formulating optimal hyper plane capable enough to classify the new samples to the respective classes in an n-dimensional data space. The hyper plane or line that separates the two classes in a two-dimensional feature space can be visualized as shown in the Fig. 2.1. The figure shows that any new sample can be classified to either class 1 i.e. represented as circles or class 2 represented as squares. It is obvious that there is more possible number of hyper planes to separate the examples of the two classes. The aim of classifier is to find an optimal one that separates the samples to the maximum. This requires finding a hyper plane with the maximum margin. A margin defines the perpendicular distance between the hyper plane and the nearest sample to the hyper plane. Larger margins reflect the classes are highly separable. The hyper plane's dimension is dependent on the dimension of the feature space. For a two dimensional feature space, the hyper plane is a line, whereas for a three dimensional feature space the hyper plane is a two dimensional plane, and so on. The hyper plane for m-dimensional feature space is represented as follows,

$$B_0 + B_1X_1 + B_2X_2 + \cdots + B_mX_m = 0 \tag{2.1}$$

With the hyper plane defined, the data points are divided into two classes, one in either side of the hyper plane and defined as follows,

$$B_0 + B_1X_1 + B_2X_2 + \cdots + B_mX_m < 0, \text{ and} \quad (2.2)$$

$$B_0 + B_1X_1 + B_2X_2 + \cdots + B_mX_m > 0 \quad (2.3)$$

The datapoints that are lying closer to the hyper plane are chosen as support vectors. The support vectors are chosen to maximize the margin. When the data points in the training set are linearly separable as shown in Fig. 2.1, then the discriminative function for predicting the new input vector (x) and support vector (X^i) as follows,

$$f(x) = B_0 + \sum a_i * (x, X^i) \quad (2.4)$$

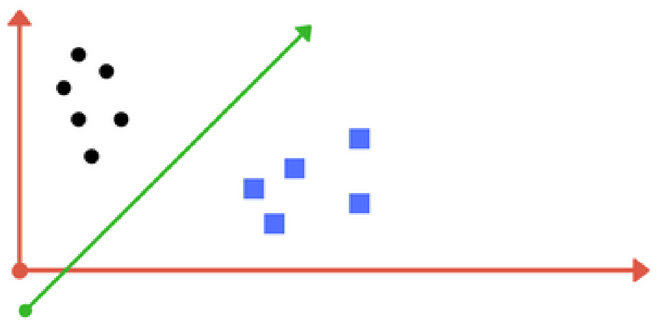


Fig. 2.1 Line Separating the Samples of Two Classes

where, the parameters B_0 *and* a_i are learnt through SVM algorithm from the training set. The second term of the equation represents the kernel function, linear in nature and is evaluated as a dot-product of the new data point and the support vectors. Polynomial kernel and radial kernel are the other types of kernels used popularly in SVM that divides the datapoints using curved line and polygon respectively. A polynomial kernel is defined as follows,

$$K(x, X^i) = 1 + \sum(x, X^i)^d \quad (2.5)$$

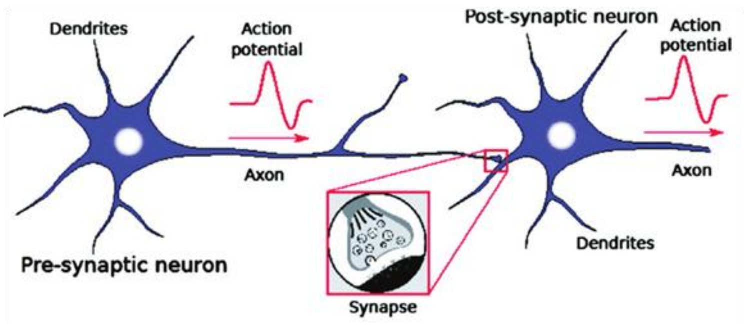
and a radial kernel is defined as follows,

$$\mathrm{K}(\mathrm{x}, \mathrm{X}^{\mathrm{i}}) = \exp\,(-\mathrm{gamma} + \sum(\mathrm{x}, \mathrm{X}^{\mathrm{i}})^2) \quad (2.6)$$

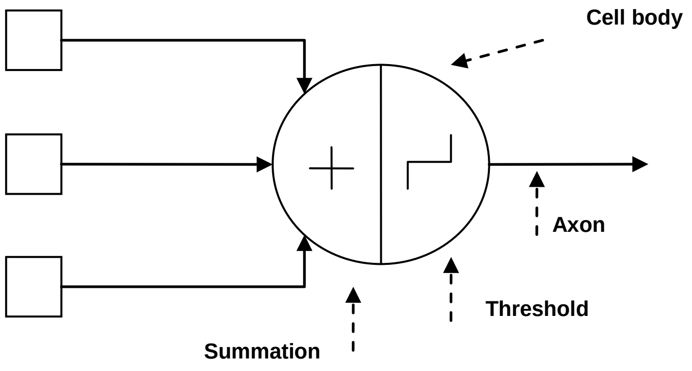
where, the gamma parameter defines the influence of an individual example on the plausible line. The value of gamma lies between 0 and 1. Low gamma values greater distance from the hyperplane and high values representing closer to the hyperplane.

Artificial Neural Networks

An Artificial Neural Network (ANN) is an idea proposed for information processing inspired by the working of brain, the biological nervous systems for process information. The core of this paradigm lies in the way the information is processed – the neurons and interconnection of huge number of neurons in the network to solve a specific problem. ANNs is an effort to make machine learn like human. ANNs are notable for their ability to learn even complex and imprecise data. ANNs are self-organised, support adaptive learning and real-time operation with false tolerance through redundant information coding. Fig. 2.2 shows the resemblance of the artificial neuron to the human neuron. A typical neuron in the human brain, collects signals from others through a horde of fine structures called dendrites. The neuron outputs spikes of electrical activity all the way through a long, thin stand called axon, which further splits into thousands of branches. A structure called a synapse lying in the end of each



(a)



(b)

Fig. 2.2 (a) Human Neuron with Synapse (b) Artificial Neuron

branch converts the action from the axon to electrical effects that induces activity in the connected neurons. Effectiveness of the synapses is changed through learning, thus influencing one neuron by the changes in other neurons. An artificial neuron is an entity with several inputs and one output. The neuron in operation can be in either the training mode or the using mode.

The neuron are trained to fire or not for a given pattern in training mode, where as it fires when taught input is received or decides whether to fire nor not if untaught input pattern is receive in the using mode.

Firing rule is a key concept in neural networks. It identifies if a neuron should be fired or not for the given input pattern. It defines the relationship between the inputs and the output not only for the inputs it has seen but even for the unseen ones. For example, a simple firing rule can be defined using hamming distance for a classification problem. For a collection of patterns in the training set for a given node, some pattern causes it to fire the true set of patterns and others prevent it from firing the false set patterns. Further the patterns not in the collection cause the node to fire if, on comparison, their hamming distance to the true set of pattern is lesser or un-fire if they are closest to false set of patterns. The output remains undefined in case of a tie. Frank Rosenblatt coined the term perceptron which was later highly influential in the area of ANNs [43]. Perceptrons are neurons with weighted input in addition to some fixed pre-processing that emits a binary output.

Architecture of Neural Networks

A neural network is composed of three layers, namely the input layer, hidden layer, and output layer. Each layer is formed using of one or more nodes. The connection lines between the nodes designate the information flow from one node to the other. A fully connected neural network is shown in Fig 2.3. The input layer nodes are passive in nature, implying that they do not change, where as the node of hidden and output layers are active that tend to change. They receive a value from the input and broadcast to multiple nodes through their outputs to the hidden nodes. The values in variables $X1_1$, $X1_{2...}X1_{15}$ denote the data to be evaluated. The outputs from the previous layer when sent as input to the next layer is multiplied by predetermined weights. The sum of weighted inputs of a neuron produces a single value and is passed on to a non-linear mathematical function defined in the neuron to produce the output of that neuron, called the activation function. The outputs of the hidden layer is represented as $X2_1$,$X2_2$...$X2_4$ as seen in the Fig. 2.3. The output of hidden layer is further passed to the output layer for activating the respective neurons. A number of hidden layers in the neural network may vary and the number of nodes in each layer may vary based of the requirement. A wide range of non-linear function options like sigmoid function, binary function, tanh function, scaled exponential function, rectified linear units are available to choose from for building a neural network model. Two basic categories of neural networks architecture based on the direction of information flow namely, feed forward neural network and feedback neural network are defined below.

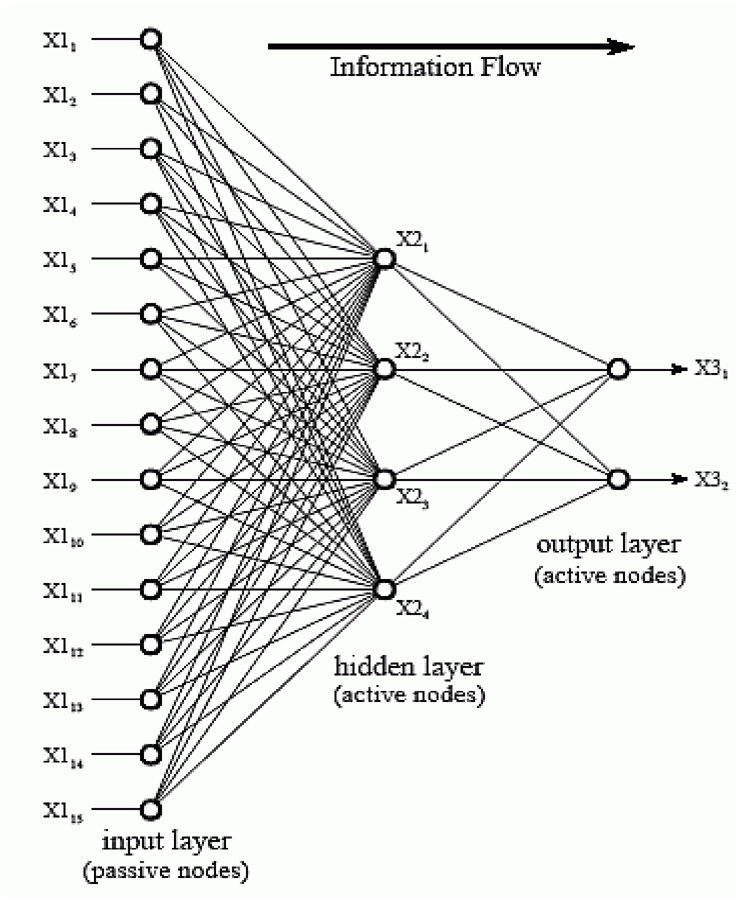


Fig. 2.3 Neural Network Architecture

Feed-forward Neural Networks: These networks as shown in Fig. 2.4 (a), allow the signals to travel only in one direction, i.e. from the input layer to the output layer. It does not have any backward connections tending to change or update the neurons through the layers from which it reached. This network tries to map the input values to their corresponding outputs. They are popularly used in the field of pattern recognition.

Feedback Neural Networks: These networks have backward connections to the neurons lying in the layers before. These make the network more complicated thus able to solve complex problems. The states of the neurons in this networks changes continuously until a stable network is built.

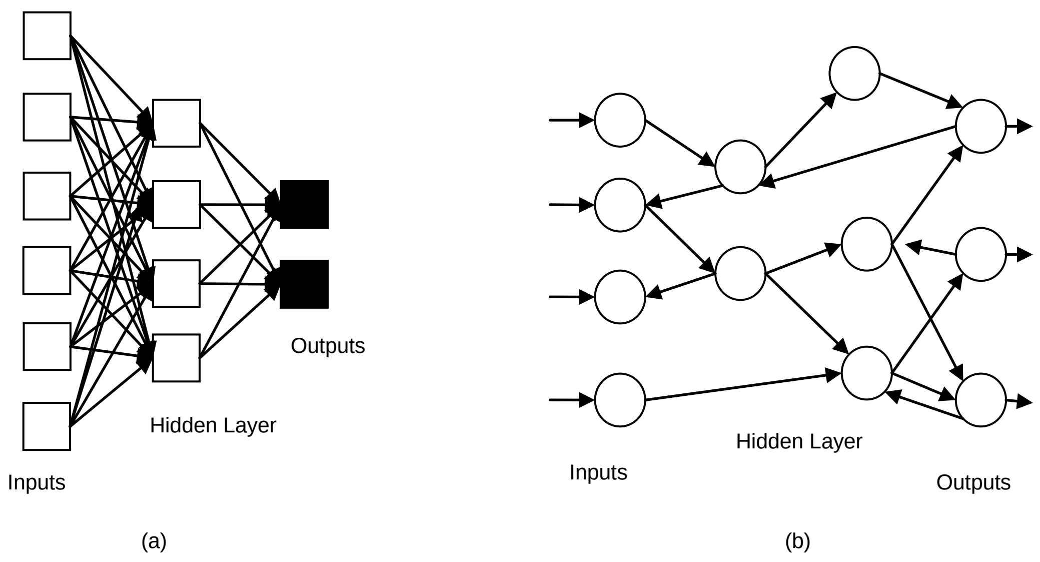


Fig. 2.4 (a) A Feed-forward Network (b) A Complicated Network with Feedbacks

The advantages of using ANNs include the capability of handling non-linear problems, parallel learning, reprogramming is not required for newer data and adaptive to solve complex problems. ANNs do have few setbacks which include requiring intensive training and high processing time to emulate. ANNs have been applied to various real-time problems like solar steam generating plant modeling, estimating heating-loads of the buildings, parabolic trough collector's intercept factor and local concentration ratio. Further ANNs has shown its prevalence in the domains including control, robotics, manufacturing, optimization, pattern recognition, forecasting, medicine, signal processing, power systems and social/psychological sciences.

Genetic Algorithm

Charles Darwin, in 'The origin of species' has mentioned the power of the planet undergoing cycles in accordance to a fixed law of gravity leading from the beginning to the endless of most beautiful and most wonderful evolution of life in this blue planet. Genetic algorithms approach mathematical optimization as a metaphor of nature's evolution, as artificial neural networks mimic the human brain. These algorithms are derived from the idea of biology namely natural selection, reproduction and genetic mutation. The process of natural selection is nothing but 'survival of fit' which means the living organisms unfit for the environment die and the one that are fit survive and reproduce in more prolific way. Reproduction and crossover happens when animals breed and produce a new one that inherits the properties from both its parents. The successful breeds or children then pass over their characteristics to their descendants. Mutation is nothing but the process of copying the gene from the parent and relaying it to the next generation. During such process the order of some genes' might be misread or a piece might be substituted by other leading to a change are some characteristics. This may lead to dead or impairment in one direction where as to explore more possible genetic combinations in other direction.

Genetic algorithms have as its backbone a set of operators that are operated on the samples in current generation to produce new samples in the next generation, out of which few key operators are discussed here. The first operator discussed here is selection. It is inspired by the simple idea of ‘survival of the fittest’, this type of genetic algorithm stochastically selects an individual from the current generation to create a basis for the next generation. The fittest individuals in the current population enter the matting pool for next generation, which does not mean that the weaker ones are useless; sometimes they prove themselves useful for the future generations.

The next is crossover genetic operator and is popularly used in genetic algorithms. In crossover, the genetic information of two parents is combined to form a new offspring. Traditional genetic algorithms represent genetic information of chromosomes in the form of bit arrays. There are various types of crossover available which are briefed as follows,

Single point crossover: A crossover point is selected for the chromosomes selected, the first part in the binary string of one chromosome is taken as the first part of the offspring and the second part i.e. lying after the crossover point of the other parent chromosome is taken as the remaining part of the new offspring.

Two point crossover: In this crossover, two crossover points are selected. The binary string from starting till the first crossover point of chromosome is copied from one parent, the substring of chromosome from the first crossover point till the second crossover point is copied from the other parent and the remaining part is copied from the first parent to form a new offspring.

Arithmetic crossover: It uses some arithmetic operation to produce new offspring.

The last one here is mutation operator, which uses a single parent contrasting to two parents as in crossover, to generate a new offspring by mutating or change some genetic information in a chromosome. One popular way of mutation is bit inversion, i.e. flipping of bits. The process flow of genetic algorithm is as shown below:

1. Create a population and initialize randomly.
2. Evaluate the fitness of all members in the population.
3. Repeat till the convergence is reached.
    a. Choose appropriate fittest parents from the available population.

b. Perform crossover operation to generate new population.

c. Apply mutation on the new population.

d. Evaluate the fitness of the present population.

Decision Tree

A decision tree is a directional graph with series of nodes which starts at the base with a single node called a root node and extends to leaf nodes that represent the categories in the solution for the tree to classify. It is a decision support tool which allows performing a classification based on a series of minor decisions. The classification process starts with the root node, deciding the path to take out of the available branching options based on the result of evaluation in the current node. It uses a tree-like graph to identify the predictions for the given problem that result through a series of feature-based splits at each tree node. The following Fig. 2.5 shows the structure of decision tree.

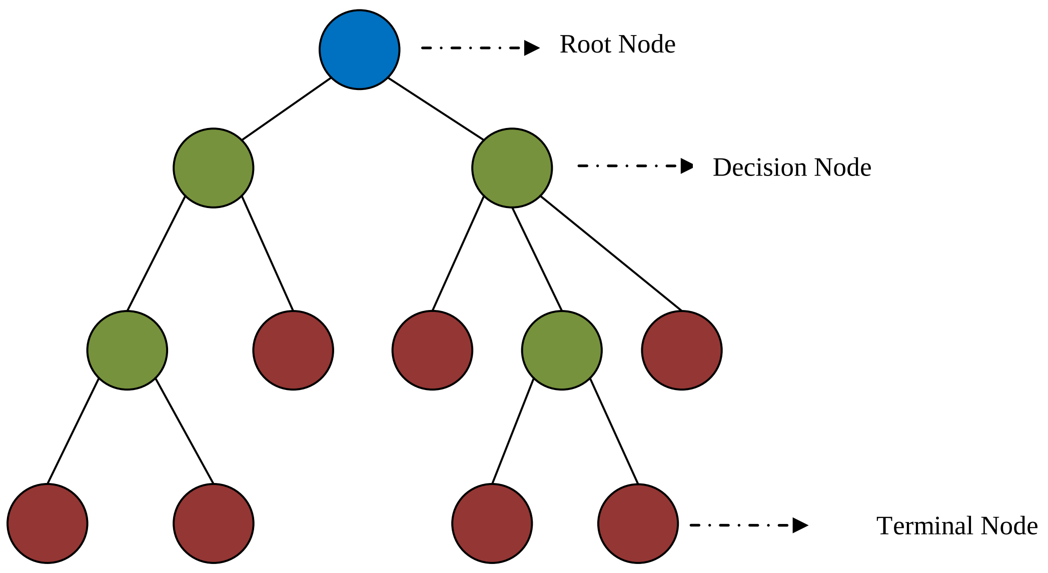


Fig. 2.5 Structure of Decision Tree

Popular algorithms to build decision trees are Classification and Regression Trees (CART) that uses Gini Index as classification metric and Iterative Dichotomiser 3 (ID3) that uses entropy function and information gain as decision metrics. CART is used for predictive modelling problems catering to classification and regression types developed by Breiman et al. in 1984. The CART algorithm generates a binary decision tree which means that the number of spits at each node is fixed as 2. It uses Gini Index as the classification metric and is defined as a sum of squared probability of each class in the dataset given as follows,

$$Gini\ Index = 1 - \sum_{i=1}^{no.\ of\ classes} (P_i)^2 \qquad (2.7)$$

where P_i is the probability of ith class. The following lists the steps involved in the CART algorithm.

1. A set of labelled data is taken as input.
2. For each variable in the given input space, find the best split.
3. Choose the best variable for split.
4. The input dataset is split into sub-datasets representing left and right nodes of the current node.
5. Repeat steps 2 to 4 till the required split is achieved.
6. Perform pruning operation on decision tree.

The CART is driven by three main elements namely, the splitting condition at each node, the stopping rule that helps to decide when to stop the branching in tree and the prediction process for the target variable at the terminal node.

For example, if the dataset considered has two variables var1 and var2, the Gini Index of var1 and var2 are calculated. The variable with the lowest Gini Index is chosen for split and a best split of dataset based on the selected variable is identified using greedy approach. The process is recursively applied to the sub-datasets derived until the specified stop criterion is met. Generally the stop criterion is set to some number of samples in the sub-dataset derived.

In other hand, ID3 is a decision tree building algorithm used for classification developed by J. Ross Quinlan in 1975. It uses greedy top-down search to test the attributes at every node in the tree. The tree is used to classify the incoming unseen data. This algorithm uses entropy and information gain metrics to identify the split. The algorithm first calculates the entropy of each attribute using the dataset. Then it splits the dataset into sub-datasets based on the attribute with minimum entropy or maximum information gain to make a split node. The process involves evaluating the entropy and information gain on splits of each attribute in the dataset. The entropy and information gain are calculated as follows,

$$\mathrm{Entropy(S)} = \sum - \mathrm{p(I)} - \log_2 \mathrm{p(I)} \qquad (2.8)$$

$$\mathrm{Gain(S, A)} = \mathrm{Entropy(S)} - \sum \mathrm{p[S|A]}.\mathrm{Entropy(\ S|A)} \qquad (2.9)$$

where $Entropy(S)$ is entropy of dataset before split, $Gain(S, A)$is information gain of splitting dataset S with attribute A, $p(I)$is the proportion of class I to number of samples in the dataset.

The decision tree algorithms are used to build the rule set from the dataset. It takes a long training time to build the rule set and also suffers from the problem of over-fitting. Several versions of decision tree algorithms have evolved, in which the version named random forest solves the over-fitting issue in addition to reduced time complexity.

2.2 DEEP LEARNING

Deep learning is a facet of Artificial Intelligence (AI), which automates the process of learning though predictive analytics by simulating the learning mechanism in human. The deep learning algorithms are considered to be more complex when compared to the traditional machine learning algorithms that supports in handling problems with even diverse, inter-connected or unstructured data. Also these algorithms tend to give a more abstract representation of the model being dealt. These algorithms are successful enough in building clustering or classification models for huge amount of data generated in this fast growing internet era which needs powerful problem solving techniques.

Deep learning is a representational learning that learns more and more from data. It gains knowledge from experience as human do. They learn from the data representation automatically when training the deep models either through supervised, unsupervised or semi-supervised learning approaches. The deepness in their architecture enables them to capture complex interrelationship underlying the training data in the computational units that form the skeleton of the deep architectures. The aspects that govern deep learning are multi-levels of hierarchical representation, training, multiple non-linear transformations, pattern recognition, feature extraction and high level data abstractions. The evolution of deep learning basically has been driven existence of various catalysts like multi-layered learning networks, big data, powerful graphical processing units, huge amount of data, performance of neural networks and parallel computing.

Currently, deep learning paradigm has been applied on to solve various problems and has achieved in designing efficient systems few of which are explained below.

Recommendation Engine: The two broad categories namely the content-based and collaborative filtering methods use deep networks to learn from massive datasets and delivers system with greater productivity and performance through efficient classification and regression analysis. Spotify, Amazon and Netflix are few such apps that have successfully implemented recommendation engine using deep networks.

Text Sentiment Analysis: Research in natural language processing have succeeded to greater extents in parallel with the studies of recurrent neural networks in developing models capable of extracting higher-level information. Current applications include comments section where comment-based reviews are generated and presented through deep networks. Also meaningful topics are extracted from social media using named-entity recognition models that uses deep learning in core to explore the reaction of public on the hot topics.

Chatbots: They are trained with a huge collection of dialogues to act as a conversational agent on websites and applications which uses deep structures like recurrent neural networks. They are one of the newest tools acting as virtual assistants, in many huge sectors like baking and telephony.

Self-driving cars: The technology lying behind self-driving cars has become true because of deep learning. Deep learning algorithms used in these cars are trained vigorously with millions of data collected on driving videos and then tested in restricted environment. The Uber AI Labs at Pittsburg is one of the organizations working acutely on driverless cars. In addition, they are trying to integrate smart features that include delivery options, which on completion can be used for food delivery, courier, etc. Handling unprecedented scenarios if one of the major concern focused by the developers of self-autonomous vehicles. Frequent testing and implementation cycles exposing varied scenarios for strengthen the models trained on deep learning algorithms ensures safety. Data from cameras, geomapping and sensors help to develop brief and complex models for navigating through traffic, identifying paths, signs, pedestrian-only routes and real-time components such as traffic volumes and road blocks.

Visual Recognition: Several tasks like search images or photographs linked to a particular person, location, group of faces, occasion fall in the plethora of visual recognition problems. Searching a photo from a digital library requires a state-of-art recognition system that is powerful enough to capture the intricate elements spanning from basic to advance. This has become true with the advent of deep neural networks making digital media management simple.

Fraud Detection: Banking and financial sector is another domain that makes use of deep learning to add excessive benefits to its user providing fraud detection functionalities on digital money transactions. Credit card fraudulent actions are detected by developing systems using autoencoder modules in Keras and Tensorflow helping to save billions and billions of money through recovery and insurance for financial institutions. The patterns of customer transactions and their credit score support in determining the fraudulent action leading to fraud prevention and detection. Outliers lend a hand in identifying anomalous behaviour of the customers. Fraud detection can be handled by many machine learning techniques like classification and regression. Eventhough machine learning have shown its predominance in this area, it require human assistance to an extent which can be even minimized in case deep learning techniques.

Healthcare: Computer assisted diagnosis and treatment have penetrated to each and every division of healthcare from medical imaging, genome analysis to new drug discovery. As a whole the healthcare industry is transforming with the heart as GPU. Powerful applications today are GPU-accelerated delivering efficiencies and possibilities to unseen levels. It thus empowers clinicians, physicians, and researchers to deliver the ultimate goals towards improving and protecting lives. Deep learning in health care support for early and accurate diagnosis even for life threatening diseases by providing augmented diagnosis and treatments. They also provide a means of standardized approach in treatment. Further, they help in understanding genetics and to fore say the risk of genetic disorders.

Deep learning has come out with varied architectures to handle variety of problem domains. Generally deep learning architecture is composed of three types of layers as in neural networks namely the input layer which is the first layer, the output layer which is the last layer, and a collection of hidden layers lying in between the input and output layers. The term 'deep' in deep learning represents more than two hidden layers. Various architectures of deep networks that support deep learning are Convolution Neural Networks (CNN), Recurrent Neural Networks (RNN), deep autoencoders, deep belief networks, Long-Short Term Memory (LSTM), etc. Few of which are detailed below.

Convolution Neural Networks

Convolution Neural Networks (CNNs) is a Deep Learning algorithm that requires less pre-processing when compared to the other classification algorithms. These networks even have the ability to self build the filters used in the convolution layer of the network while in other algorithms need to be handcrafted and trained. The CNN architecture is comparable to the connectivity pattern of neurons in the Human Brain and was inspired by the organization of the Visual Cortex [44]. Individual cortical neurons respond to stimuli only in a restricted region of the visual field called as the Receptive Field. Further the receptive fields of such neurons partially overlap to cover the entire visual area. They enable machines to view and perceive the world as humans do and can be used to perform multitude of tasks like image and video recognition, image analysis and classification, natural language processing, media recreation, recommendation systems, etc. [45-48]. Fig. 2.6 shows typical example architecture of CNN.

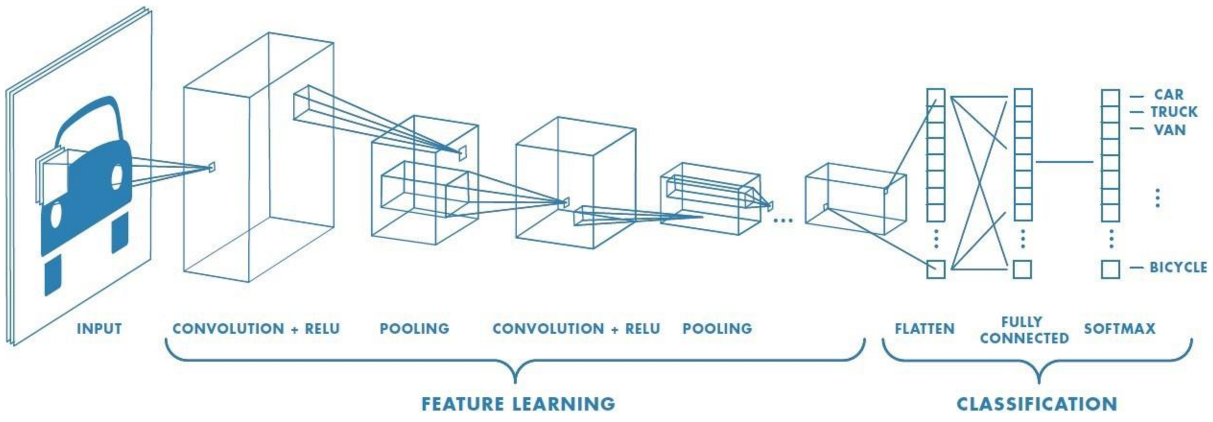


Fig. 2.6 Architecture of CNN

A CNN composed of an input layer, multiple hidden layers and an output layer. The hidden layers are arranged as interleaved convolution and pooling layers, which is then followed by a fully connected neural network leading to the output layer as illustrated in Fig.2.6.

The convolution layers are made up of a collection kernels, otherwise called as learnable filters. They have a small receptive field that are extending for the complete depth of input dimensions. During learning, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and input, producing a two-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position of the input.

Stacking the activation maps for all the filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares the parameters with neurons in the same activation map.

Rectified Linear Units (ReLU) layer applies the non-saturating activation function like max, hyperbolic tangent or sigmoid function and increases the non-linear properties of the objective function and the network as a whole but does not impact the receptive fields in the convolution layer. It helps to train the neural network faster without significant penalty to generalization accuracy.

The pooling layer is accountable for reducing the spatial size of the convolved feature which reduces computational power required and to extract dominant features which are rotational and positional invariant. It helps to maintain the effectiveness in the model training process. Max pooling and average pooling are the two types of pooling. Max pooling gives the maximum value from the portion of the image covered by the kernel, where as average pooling gives the average of values from the portion of the image covered by the kernel. The convolution layer and the pooling layer, together form the i^{th} layer of a CNN.

The non-linear function hidden in the available solution space is explored and learnt by fully-connected layer. The output of the last convolution layer is flattened and fed to the multi-level perceptron, where back-propagation training is applied for a lump sum of epochs. The model thus gets ready to classify the incoming image finally using softmax classification. LeNet, AlexNet, VGGNet, GoogLeNet, ResNet and ZFNet are various powerful architectures of CNNs powering AI. Several enhancements like Multiview Convolution Neural Network (Multiview CNN) [49] and attention pooling [50] for pooling layer help improve the productivity of CNNs.

CNNs have its own pros and cons as any other model. At the positive side, it automatically extracts the relevant features for the given task if the input can be defined as a tensor which comprises of correlated elements locally. The sparse connectivity and weights shared locally makes the CNNs have highly restricted prior, which when tailored properly and appropriately yield a more efficient models turning out with best accuracies. The weight sharing in CNNs reduces the complexity in terms of both time and memory to a greater extent. It does not work for highly uncorrelated input and also requires careful pre-training to initialize the parameters of the network, which is really a challenging task.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are deep neural network architectures where the output of previous step is fed as input to the current step. The network memories the previous state of the network, thus having the ability to pass information from one step to the next. Even though training such networks are difficult, the modern research approaches on optimization, parallelism, network architectures and graphics processing units have made them amenable. RNNs take one input vector at a time from the sequence of input vectors, allows the network to retain state through hidden state while modeling the next. The trademark property of RNNs is to model time dimensional problems like predicting the words of a sentence, music and speech signal modeling, handwriting generation and synthesis, etc. [51-53]. It uses the same parameters on all the inputs or hidden layers for each input as it performs the same task to generate the output.

RNN converts the independent activations seen in deeper networks into dependent activations with same weights and biases to all the layers instead of individual weights and biases for each layer as defined in traditional network. This reduces the complexity faced in increasing parameters and memorizing each of the previous outputs by providing each output as input to the next hidden layer. Hence all the hidden layers can be coupled together into a single recurrent layer such that the weights and bias of all the hidden layers is the same. Training RNN can be listed through the following steps:

1. An input vector representing a single time step is given as input to the network at a time.
2. The current state is evaluated with set of current input and the previous state as,

   $h_t = f(h_{t-1}, x_t)$, where h_t, defines the current state, h_{t-1}, the previous state and x_t, the input state. The activation function is given by, $\tanh(W_{hh}h_{t-1} + W_{xh}x_t)$, where W_{hh} and $W_{xh}$are weights of recurrent and input neurons respectively.
3. The number of time steps depends on the problem which joins the information from all the previous states.
4. The final current state is used to calculate the output once all the time steps are completed and is calculated as, $y_t = W_{hy}h_t$, where y_t is the output and $W_{hy}$is the weight at output layer.
5. The actual output and target output is then compared to evaluate the error.
6. The RNN is trained by back-propagating the error to update the weights of the network.

Even though RNNs are best suited for prediction problems, they suffer from gradient vanishing and exploding problems.

RNNs have two important points in their positive side. One is the size of input is constant regardless of the length of the input sequence. This is due to the fact that the input size depends on the transition from one state to the next state rather than depending on the history of states. The next is, at every time step, the transition function and their parameters can be set to same values. At the same time, the issues concerned with RNNs are vanishing gradient problem and exploding gradients problem. These problems prohibits RNNs in processing very long sequences when using tanh as activation function and becomes unstable when using ReLU function for activation. Also RNNs are not suitable to build deep models because of the saturated activation function.

Deep Autoencoders

A deep autoencoder is a neural network that uses unsupervised machine learning algorithms to transform the inputs to outputs where the targets are equal to inputs with minimum possible error. A deep autoencoder is a deep learning architecture as given in Fig. 2.7, composed of two symmetrical deep belief networks, one for encoding and the other for decoding. Both the encoding and decoding parts of the autoencoders individually have four or five shallow layers. The layers are restricted boltzmann machines that form the building blocks of deep belief networks. The layers in the encoder compress the features in the input space to lower dimension latent space, where as the decoding layers regenerates the original input from the features represented in latent space. In between the encoder and decoder layers of the autoencoder lies the code layer holding the compressed latent feature vector of the input.

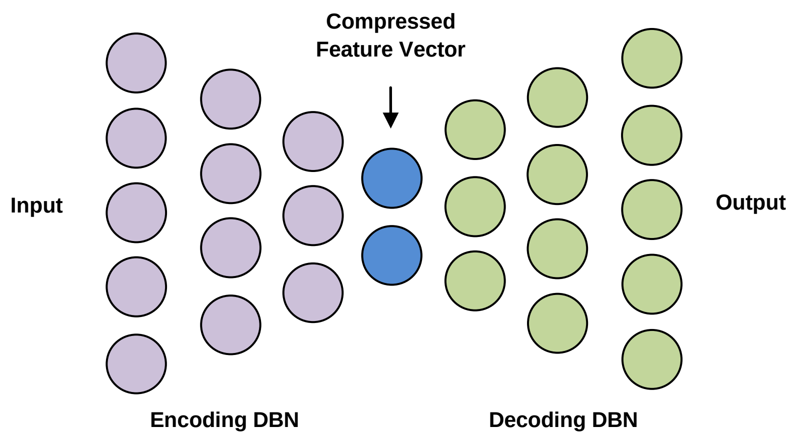


Fig. 2.7 Deep Autoencoder

Training autoencoders requires four parameters to be set prior to training, namely, code size, number of layers, number of node per layer and loss function. The code size represents the number of nodes in the middle code layer, a less number implies more compression or more dimensionality reduction. The number of layers specifies the number of layers in encoding and decoding phases of autoencoder. The choice of loss function to evaluate the error based on the inputs may be either mean square error or binary cross entropy. Generally, the number of nodes in the encoder layers keeps decreasing where it keeps increasing in the decoder layers.

The bottleneck approach used here discriminates the relevant and irrelevant features. This is achieved through two criteria, one is the compactness of input representation, which defines the level of compression. The other is related to the selection variables that retain the vital behavioural information about the input data.

Consider an input of dimension x is given to the auto encoder, the encoder compresses x-dimensional vector to a lower z-dimensional vector, which is the output of the encoder. The decoder then takes this z-dimensional input and outputs the parameters using the probability distribution of the data to reconstruct the input with x dimensions. The loss function that is composed of the reconstruction loss and the regularization is evaluated and the error is backpropagated to train the autoencoders.

Deep autoencoders provide a non-linear transformation of linear or non-linear data with the use of convolution layer rather than using dense layers and produces multiple transformations, one from each layer of the autoencoder. The autoencoders are unsupervised learners, data-specific and lossy in nature. Deep autoencoders are successfully used in image search, data compression, speech recognition, speech enhancement, topic modelling and information retrieval and more [54-59]. These models are more data specific, i.e. autoencoders trained on one specific type of data do not work well on another type. Also the decompressed output seems to be lossy to the actual input. But the model trained with appropriate and specific type of data learns automatically and outperforms for that type of input without any need of additional engineering.

Deep Belief Networks

Before getting deep into deep belief networks, understanding belief network and restricted boltzmann machines becomes mandatory. A belief network is composed of layers with stochastic binary units. These layers are when connected hold weights with each unit biased. The stochastic binary units in these layers are either in 0 state or 1 state. The probability of the output to be 1 is dependent on the bias and weighted inputs to that unit. They are intern directed acyclic graph. An RBM is a bipartite network having two layers, the visible layer and the hidden layer. In RBMs, the connections are restricted to visible-hidden connections. RBM differs from a boltzmann machine (BM) from the fact that RBM does not allow inter-layer connections where as BM does. Further, the hidden units in RBM are conditionally dependent on the visible units. This helps to generate unbiased sample from the posterior distribution for the available data vector.

A DBN is an artificial neural network which comprises of many hidden layers. A DBN is a multi-layer perceptron architecture, which is built as a stack of RBMs as shown in Fig. 2.8. In other words, DBN is composed as a collection of RBM layers during the pre-training phase, where as acts like a feed forward network in the fine-tune phase. The visible layer of the first RBM is fed with the feature vectors, which is passed on to the output layer of that RBM modelling the posterior probabilities of the hidden units of DBN. The activation outputs of one RBM acts as input to the succeeding RBM in the stack. Finally, the DBN as a whole is trained once the learning of final hidden layer is accomplished. The initial layer by layer learning of RBMs is unsupervised and its mathematical modelling is elucidated below before discussing the algorithm to train DBN.

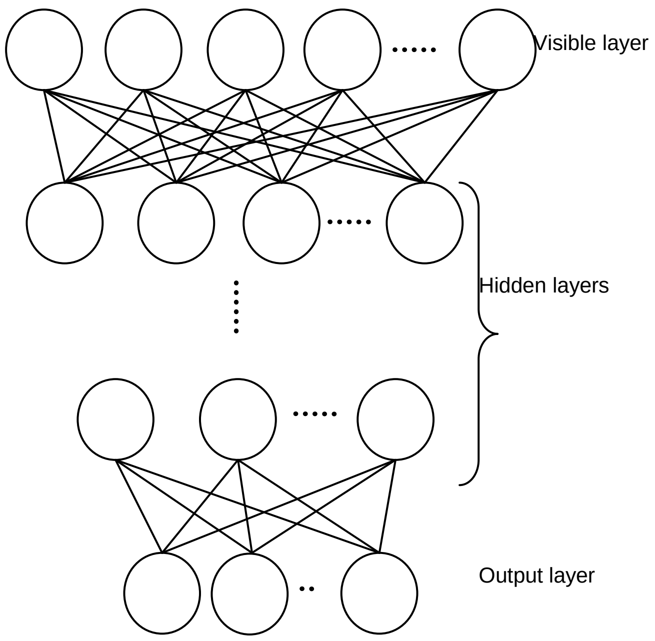


Fig. 2.8 Deep Belief Network with a Visible layer, n-Hidden layers and an Output Layer (Undirected DBN)

Mathematical Modeling of RBM

Based on the distribution of vectors in observations, the RBMs can be modelled either bernoulli-bernoulli or gaussian-bernoulli. In simple binary RBM or bernoulli-bernoulli RBM, both the visible and hidden units are binary and stochastic in nature. gaussian-bernoulli RBMs are usually used to model real-valued data. Thus, the DBN acts as a non-linear classifier with each of the hidden layer expressed as posterior probabilities. Each neuron in the hidden layer uses the logistic function to convert its input received from its lower layer to a scalar value which is passed on to the next layer.

Learning in Bernoulli-Bernoulli RBM (BBRBM)

In BBRBM, the connection weights and the biases of the neural units define the probability distribution over the joint states of the visible and hidden units through energy function:

$$E(v,h|\theta) = -\sum_{i=1}^{V}\sum_{j=1}^{H} w_{ij}v_ih_j - \sum_{i=1}^{V} b_iv_i - \sum_{j=1}^{H} a_jh_j \qquad (2.10)$$

where $\theta = (w,b,a)$ are model parameters and w_{ij} is the connection weight between the ith visible unit and the jth hidden node. V and H are the number of visible and hidden units respectively. The probability of the visible vector v for the given model parameters θ is given by:

$$p(v|\theta) = \frac{1}{Z}\sum_h e^{-E(v,h)} \qquad (2.11)$$

The conditional probability distribution of hidden units, given the model parameters and visible units is represented as:

$$p(h_j = 1|v,\theta) = \sigma(a_j + \sum_{i=1}^{V} w_{ij}v_i) \qquad (2.12)$$

Where, σ is the sigmoid function, given by, $\sigma(x) = (1+e^{-x})^{-1}$.

The conditional probability distribution of visible units, given the model parameters and hidden units is represented as:

$$\mathrm{p}(\mathrm{v_i} = 1|\mathrm{h},\theta) = \sigma(\mathrm{b_j} + \sum_{\mathrm{j}=1}^{\mathrm{H}} \mathrm{w_{ij}h_j}) \qquad (2.13)$$

Learning in Gaussian-Bernoulli RBM (GBRBM)

Similar to BBRBM, the connection weights and the biases of the neural units define the probability distribution over the joint states of the visible and hidden units through energy function which is stated as follows:

$$\mathrm{E}(\mathrm{v},\mathrm{h}|\theta) = -\sum_{\mathrm{i}=1}^{\mathrm{V}}\sum_{\mathrm{j}=1}^{\mathrm{H}} \mathrm{w_{ij}v_ih_j} - \sum_{\mathrm{i}=1}^{\mathrm{V}} \frac{(\mathrm{v_i}-\mathrm{b_i})^2}{2} - \sum_{\mathrm{j}=1}^{\mathrm{H}} \mathrm{a_jh_j} \qquad (2.14)$$

The conditional probability distribution of visible units, given the model and hidden units are represented as a Gaussian function ($\mathcal{N}$):

$$\mathrm{p}(\mathrm{v_i}|\mathrm{h},\theta) = \mathcal{N}(\mathrm{b_i} + \sum_{\mathrm{j}=1}^{\mathrm{H}} \mathrm{w_{ij}h_j}, 1) \qquad (2.15)$$

The learning algorithm detailed above to train the RBMs is called contrastive divergence. With the knowledge of the mathematical models of RBM, the greedy learning strategy used to in initialize a DBN model is detailed below.

Modeling DBN

DBNs are graphical structures capable enough to extracts the deeper hidden features from the input data as hierarchical representations. They are modelled as joint distribution of x, the observed vector l, hidden layers h^k and is defined as below,

` $$P(x, h^1, \ldots. h^{i=l}) = \left(\prod_{k=0}^{l-2} P(h^k | h^{k+1})\right) P(h^{l-1} | h^l) \qquad (2.16)$$

where $x = h^0$ and $P(h^k/h^{k+1})$ is a conditional distribution level k visible units on level $(k + 1)$ RBM's hidden unit, and $P(h^{l-1}/h^l)$ denotes the visible to hidden joint distribution of the top-level RBM. The following is the process flow of the greedy layer-wise training of DBN.

1. Train the first RBM to model the raw input $\boldsymbol{x} = \boldsymbol{h}^0$ as its visible layer.
2. The first layer is used to obtain a representation of the input, to be used as data for the second layer. The mean activation can be represented either as $P(\boldsymbol{h}^1 = 1/\boldsymbol{h}^0)$ or samples of $P(\boldsymbol{h}^1/\boldsymbol{h}^0)$.
3. The next RBM is trained with the transformed data, either mean activation or samples as input to its visible layer.
4. Repeat steps 2 and 3 for all the layers of RBM.
5. Fine-tune DBN either with respect to a surrogate for the DBN log- likelihood, or with respect to an objective function for supervised training to get an optimal model.

There are two most important advantages of greedy pretraining algorithm. First, it helps to select appropriate set of parameters to train and ensuring appropriate initialization process. Next, it uses unsupervised data. The deepness in DBNs are efficient in the sense to capture complex hierarchical relationships hidden in the features by learning low level statistical features in the shallow layers, and learning complex and abstract feature representations in deeper layers. At the same time using back propagation for fine tuning the deep network model parameters fail to produce more efficient models. It can be handled with smart pretraining algorithms. Thus the computational cost incurred in pretraining and additional training required through maximum likelihood for optimizing the network are the shortcomings concerned with DBNs.

DBNs have been successfully applied to various problem domains like Natural Language Processing (NLP), precision control, image compression, speech recognition, etc. This research uses DBN to build phoneme recognition models for Tamil continuous speech.

2.3 PARAMETER OPTIMIZATION

Parameter optimization is the process of identifying the optimal parameters that define any model like classification, regression or clustering that is developed to solve a specific task. This can be achieved by minimizing the loss function defined by the parameters of the model. For example the connection weights, biases, number of layers, number of neurons in each layer in a neural network forms the set of parameters defining the model. The values assigned to the parameters in the parameter set of a model play a vital role in controlling its efficiency. Even though several algorithms have been proposed in the literature to perform the learning more dynamic [61], there are popular optimization techniques available to optimize the parameters defining the model. Gradient descent, line search, conjugate gradient descent, Newton's method, Quasi-Newton method, genetic algorithms are few of such techniques. Optimization using gradient descent and particle swarm optimization are detailed below.

Gradient Descent

Gradient descent (GD) is an optimization method that helps us to identify the optimal parameters by following a steepest path in the solution space of the given objective function and reaching the valley. At each step the parameters are updated in the opposite direction to the gradient to reach the next best solution. The step size of the gradient which defines the distance to move is controlled by the learning rate while walking through iterations. Gradient descent can be defined as iterative evaluation of gradient followed by parameter updation. The complexity of gradient descent linearly increases with the number of parameters and the size of data used for a single parameter update. Based on how much data is used for evaluating the gradient, its variants are defined as follows.

Batch Gradient Descent, otherwise called as Vanilla GD uses the entire training dataset to evaluate a gradient and perform a parameter update defined by,

$$\theta = \theta - \gamma . \nabla_\theta L(\theta) \tag{2.17}$$

with θ, defining the parameters, γ, the learning rate and $\nabla_\theta L(\theta)$, the gradient of the loss function. So, the batch gradient descent calculates the error of the whole training dataset to undergo an update. The time complexity to perform a single update increases with the increase in number of samples in the training dataset, but the advantage here is, it promises to identify global minimum for problems with convex error surface in spite of identifying local minimum for problems with non-convex error surface.

Stochastic Gradient Descent, on the other hand, updates the parameter by evaluating gradient for each sample, x^i and label, y^i encountered in the training dataset and is given by,

$$\theta = \theta - \gamma . \nabla_\theta L(\theta; x^i;\ y^i\ ) \tag{2.18}$$

This ensures to be faster when compared to the batch gradient descent, as it updates for each encountered example, instead of updating after walking through clones of examples. The frequent updation of parameters leads to more fluctuating jumps in local minima, which can be controlled by reducing learning rate to achieve results similar to the stochastic GDs for both convex and non-convex problems.

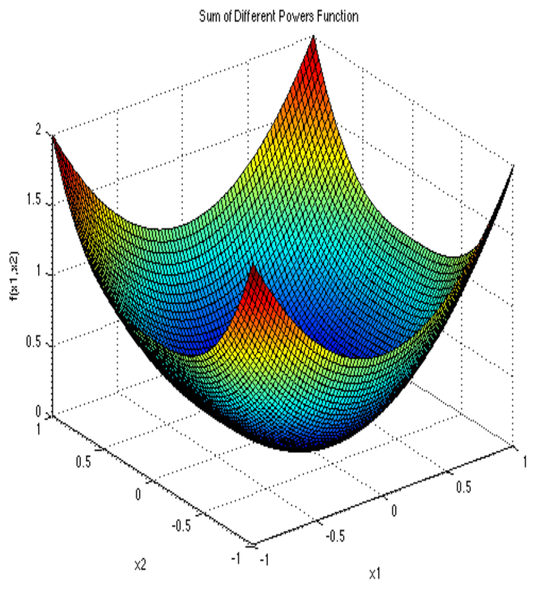
Mini-batch Gradient Descent, takes advantage of above two variants by performing gradient updates on parameters for every subset of samples of size *n* in training dataset and is defined as,

$$\theta = \theta - \gamma . \nabla_\theta L(\theta; x^{i:(i+n)};\ y^{i:(i+n)}\ ) \tag{2.19}$$

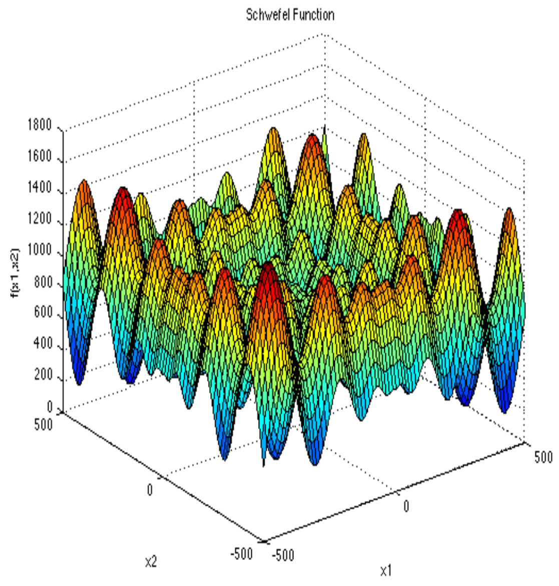
They tend to come out with more stable convergence in contrast to the fluctuations seen in stochastic GDs and support using matrix optimization which helps to speed up the calculations for highly complex structures in deep learning.

Particle Swarm Optimization

Particle swarm optimization is a technique for optimizing non-linear functions originally proposed by James Kennedy et al [62]. This algorithm binds the artificial life to fish schooling, bird flocking and swarm theory. This algorithm is inspired by the movement of the best member of the population of the bird flock or fish school that leads the other members [63]. Let $f(x_1, x_2, \ldots x_n) = f(X)$ is a n-dimensional function and is defined as a global optimization problem such that x_i is the solution variable defined by the independent variables. The objective is to find a value $x^{■}$ so that $f(x^{■})$ is maximum or minimum depending on the requirement. For example, consider the functions shown in Fig. 2.9. The function shown in Fig. 2.9(a) is the function of sum of different squares [64], is a unimodal function that has only one minimum which is the global minimum of the function lying at its origin and is straight forward to evaluate, where as the function in Fig. 2.9(b), the schwefel function [65], is a multimodal function which is complex to identify the global minimum. The multimodal function requires multiple agents for performing parallel search the search space to discover the global minimum and requires communication between the agents until the global minimum is identified. PSO is one such multi-agent parallel search method that maintains a population or swarm of particles, where each particle represents a potential solution to the problem. In crisp, all the particles of the swarm move in multidimensional solution space adjusting itself based on the experience if faced and based on its fellow mates in the swarm.



(a) unimodal



(b) multimodal

Fig. 2.9 Visualization of Unimodel and Multimodel Solution Space

The cumulus of particles is considered as a population, each represented through its position $x_i \in \mathbb{R}^D, i = 1, \ldots\ldots, M$. The optimal position in the search space is identified by repeatedly evaluating the fitness of the particles in the population. Initially the particles in the population are assigned with some random position and start moving with the random common velocity. All the particles travel in the search space according to the velocity function v_i. The velocity function depends on two components namely, social component and cognitive component. The best position of the best particle in a population is referred as social component, which can be otherwise called as global best, represented as $p_g \in \mathbb{R}^D$. The other component, which is cognitive represents the particle's own best position, represented as $p_i \in \mathbb{R}^D$, also known as local best. The local best of the personal best is updated at each time t as follows,

$$p_i^{t+1} = \begin{cases} p_i^t & if\ f(x_i^{t+1}) > p_i^t \\ x_i^{t+1} & if\ \ f(x_i^{t+1}) < p_i^t \end{cases} \tag{2.20}$$

and the global best is updated as follows,

$$p_g^t = \min\{p_i^t\}, where\ 1 < i < n \tag{2.21}$$

The velocity of each particle is updated using the following equation at each time t:

$$v_i(t+1) = \omega v_i(t) + c_1 r_1 \big(p_i(t) - x_i(t)\big) + c_2 r_2 \left(p_g(t) - x_i(t)\right) \tag{2.22}$$

where c_1, c_2 are local and global acceleration coefficients respectively, r_1, r_2 are uniformly distributed random variables in [0,1] and ω is the inertia weight which is set to linearly vary from 1 to 0 during the entire course of iteration. The velocity is generally limited to a range between v_{max} and v_{min}. The velocity of each individual particle is updated by adding its new velocity to its current position. Thus the population moves towards the best global optimum. The particle positions are then updated as follows:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.23}$$

Few parameters in PSO affect the performance of the algorithm to a greater extent. They are swarm size or population size, number of iterations, the accelerations coefficients and the velocity coefficients [66]. Swarm size denotes the number of particles in the population. The larger swarm size logically generates more parts of the search space and increases the parallel search, thus reduces the number of iterations to reach the optimal solution. This has the drawback of increasing the complexity namely, the time complexity, thus requiring more time for computations. The nominal swarm size is studied to be between 20 and 60 particles. Next is the number of iterations which is appropriately decided depending on the problem considered. Too lower value may result in premature completion of optimization while huge value may unnecessarily increase the computational and time complexity.

The velocity components of the PSO are $\omega v_i(t)$, $c_1 r_1 \big(p_i(t) - x_i(t)\big)$ and $c_2 r_2 \left(p_g(t) - x_i(t)\right)$. The first one denotes the inertia which helps to memorize the previous movement acts as a bias and control the direction of movement of the particle. The second one is the cognitive component of the respectively particle *i* and is relative to its previous performances. They act as a personal memory of particle that help to return back if required to its earlier better position. The last one is the social component which relatively compares a particle *i* to the group of particles in the swarm. It helps in the movement of the particles towards the best direction relatively to its neighbours.

The acceleration coefficient of PSO represented as c_1 an c_2 along with the random variables stochastically influence the velocity of the particles through the social and cognitive components of the particles. Some characteristics of c_1 and c_2 are:

- If $c_1 = c_2 = 0$, there is no change in the velocity until it reaches the boundary of the search space and is given by,

$$v_i(t+1) = \omega v_i(t) \tag{2.24}$$

- If $c_1 > 0$ and $c_2 = 0$, the velocity of particles are dependent only on itself and is given by,

$$v_i(t+1) = \omega v_i(t) + c_1 r_1 \big(p_i(t) - x_i(t)\big) \tag{2.25}$$

  On the other hand, if $c_1 = 0$ and $c_2 > 0$, the velocity of particles are updates so that all the particles are dragged toward the global best particle in the swarm and is given by,

$$v_i(t+1) = \omega v_i(t) + c_2 r_2 \left(p_g(t) - x_i(t)\right) \tag{2.26}$$

- If $c_1 = c_2$, all particles are attracted toward the mean of the global and local bests.
- If c_1 is very much greater than c_2, the particles are highly influence by itself, whereas the contrary case is highly influenced by the global best one, where both these cases may lead to premature optimal solution.

The advantages of using PSO include evaluation of multiple solutions in parallel, robustness, probability of discovering global optima is high, efficient enough to solve difficult mathematical problems and speedy convergence in comparatively shorter duration. PSO do have few limitations like difficulty in initializing PSO parameters, possibility of partial convergence in some complex problems and it is not suitable for scatter type problems. Throughout this research gradient descent is used while training the models using back propagation and some models built in this research are pretrained using particle swarm optimization or its variants.

SUMMARY

This chapter summarized the theories that were traversed throughout and those acted as a foundation for this research. The chapter started with the introduction to machine learning, discussed various pattern classification algorithms in machine learning like SVM, decision tree, ANN, etc. An introduction to deep learning and some deep learning architectures including CNN, RNN, Deep autoencoders and DBN were discussed in this chapter. Finally, parameter optimization, its need and methods were elucidated.