# 2. DEEP LEARNING ARCHITECTURES

Deep Learning along with Artificial Intelligence is the most popular technologies recently. Deep learning is a subset of machine learning coming under the sphere of artificial intelligence and works by gathering huge datasets to make machines act like humans. Deep architectures comprise of multiple levels of non-linear operations where each level of the architecture represents features at a different level of abstraction, defined as a composition of lower-level features. Deep learning methods have the potential to tackle new complex problems like speech, language and image recognition by learning features in the data that combine into increasingly higher level, abstract forms. The deployment of neural networks has supported deep learning to produce optimized results.

This chapter deals with the major architectures of deep learning used in this research work. Section 2.1 begins with an outline of the deep neural network architecture and highlights its applications in real life. It introduces the basic terminologies used in deep neural networks and analyses the pros and cons of deep neural networks. Section 2.2 presents the basic architecture of Convolutional Neural Networks and its architecture. Restricted Boltzman Machines and Deep Belief Networks are introduced in Section 2.3 and 2.4. The detailed description of the variant forms of RNN, i.e., LSTM and GRU are presented in Section 2.6 and 2.7 respectively.

## 2.1 DEEP NEURAL NETWORKS

A deep neural network includes a hierarchy of layers, in which each layer alters the input data into more abstract depictions which are then combined by the output layer to make predictions. DNN aims to learn feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Much attention has recently been dedicated to them due to their theoretical appeal, inspiration from biology, human cognition, empirical success in vision and natural language processing. Deep architectures are needed to learn the kind of complex functions which can represent high-level abstractions in vision, language and other AI-level tasks.

Deep Neural Networks consist of several intermediate layers that are used to build up multiple layers of abstraction. For instance, in visual pattern recognition, the neurons in the first layer might learn to recognize edges, the neurons in the second layer could learn to identify more complex shapes, like rectangle or triangle, which is built up from edges. The next layer would then distinguish still more complex shapes. These numerous layers of abstraction give deep networks a convincing lead in learning to resolve complex pattern recognition problems. Moreover there are theoretical results suggesting that deep networks are intrinsically more powerful than shallow networks. A shallow neural network has only three layers of neurons:

- An input layer that accepts the independent variables or inputs of the model
- One hidden layer
- An output layer that generates predictions

A DNN has a structure similar to shallow networks, but it has two or more hidden layers of neurons that process inputs. Goodfellow et al. [53] showed that while shallow neural networks are able to tackle complex problems, deep learning networks are more accurate, and improve in accuracy as more neuron layers are added. Additional layers are useful up to a limit of nine to ten, after which their predictive power starts to decline. Most neural network models and implementations use a deep network with three to ten neuron layers.

As shown in Fig 2.1, the input layer in a DNN receives the input data. The inputs are passed by the input layer to the first hidden layer which performs mathematical computations on the inputs. A neuron in the network contains a nonlinear activation function and has several incoming and outgoing weighted connections. Neurons receive weighted inputs and are trained to detect precise patterns by transforming it with the activation function und passing it to the outgoing connections. Most deep networks use Rectified Linear Units (ReLU) for hidden layers, since it trains much faster, is more expressive than logistic function and prevents the vanishing gradient problem [54]. Deep neural networks end in an output layer which is a logistic or softmax classifier that assigns a likelihood to a particular outcome or label. An error signal is generated by the network to determine the difference between the predictions of the network and the desired values. The network uses this error signal to change the weights so that predictions are more accurate.

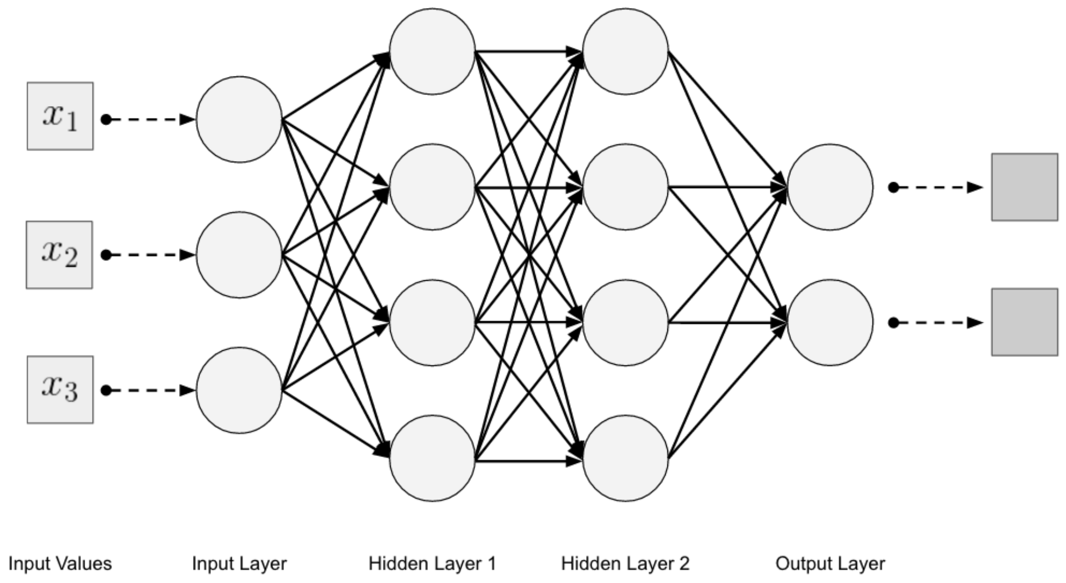

**Fig. 2.1 Basic Architecture of Deep Neural Networks**

Some basic terms used in deep neural networks are given below.

*Input* - 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 essential unit of the neural network that accepts an input and generates a prediction.

*Activation Function* - Each neuron allows part of the input which is then passed through the activation function. Common activation functions are sigmoid, tanH and ReLu (Rectifier Linear Units). Activation functions help to generate output values within an acceptable range, and their non-linear form is crucial for training the network.

*Weight Space* - Each neuron is assigned a numeric weight which is combined with the activation function to define each neuron’s output. Neural networks are trained by fine-tuning weights, to determine 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 the difference between the actual output of the current model and 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 learn the optimal weights for the neurons, a backward pass is performed 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, bias variance tradeoff is significant. Bias measures how well the model fits the training set and is capable of correctly predicting the known outputs of the training samples. Variance measures how well the model works with unfamiliar inputs that were not available during training.

*Hyperparameters* - A hyperparameter is a setting that affects the structure or operation of the neural network [55]. In real deep learning projects, tuning hyperparameters is the primary way to build a network that provides accurate predictions for a certain problem. Common hyperparameters include the number of hidden layers, the activation function, and number of epochs the training should be repeated.

*Overfitting* - Overfitting happens when the neural network is good at learning its training set, but is not able to generalize its predictions to additional, unseen examples. This is characterized by low bias and high variance.

*Underfitting* **-** Underfitting happens when the neural network is not able to accurately predict the training set and also the validation set. This is characterized by high bias and high variance.

***Methods to Avoid Overfitting***

A few common methods to avoid overfitting in deep neural networks are listed below.

- *Retraining neural networks* - running the same model on the same training set but with different initial weights, and selecting the network with the best performance.
- *Multiple neural networks* - training several neural network models in parallel, with the same structure but different weights, and averaging their outputs.
- *Early stopping* - training the network, monitoring the error on the validation set after each iteration and stopping training when the network starts to overfit the data.
- *Regularization* **-** adding a term to the error function equation, intended to decrease the weights and biases, smooth outputs and make the network less likely to overfit.
- *Tuning performance ratio* – it is similar to regularization and a parameter is used to define the required level of regularization

The neural network structure and the training algorithm is determined by variables called hyperparameters. Some of the hyperparameters related to neural network structure are given below.

*Number of hidden layers* - A hidden layer is present in between input layers and output layers, where artificial neurons generate an output through an activation function taking in a set of weighted inputs. Using too few neurons in the hidden layers will not help to detect the signals in a complicated data set. An exceedingly large number of neurons in the hidden layers increase the time taken to train the network. They are fine-tuned and calibrated through a process called backpropagation.

*Dropout* - It is a regularization technique for neural network model. In this technique arbitrarily chosen neurons are ignored during training. They are dropped-out randomly. Their contribution to the activation of downstream neurons is removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

*Activation function* - In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input. The rectified linear activation function is the default activation function for many types of neural networks which is a piecewise linear function. It is easier to train and often achieves better performance. The sigmoid and hyperbolic tangent activation functions are other methods available.

*Weights initialization* - A proper initialization of the weights in a neural network is critical to its convergence. Initially the weights of artificial neural networks are set to small random numbers.

The hyperparameters related to the training algorithm are given below.

*Learning rate* - The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. It is challenging to choose the learning rate as a too small value may end in a long training process that could get stuck, whereas a large value may result in learning a sub-optimal set of weights too fast or an unsteady training process

*Epoch, batch size* - The number of epochs is a hyperparameter that defines the number of times the learning algorithm will work through the entire training dataset. One epoch indicates that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. Before updating the internal model parameters, the number of samples to work through is defined using the batch size hyperparameter. Batch gradient descent, is the learning algorithm when all training samples are used to create one batch [56]. In the stochastic gradient descent, the batch is the size of one sample. Mini-batch gradient descent is the learning algorithm used when the batch size is more than one sample and less than the size of the training dataset. In the case of mini-batch gradient descent, popular batch sizes are 32, 64, and 128 samples.

*Optimizer algorithm* - Gradient descent is one of the most popular algorithms to perform optimization and the most common way to optimize neural networks. Every Deep Learning library contains implementations of various algorithms to optimize gradient descent. Stochastic Gradient Descent (SGD), Adam, AdaGrad, Adamax, Adadelta, RMSprop are some optimizers available.

The number of architectures and algorithms used in deep learning is extensive and broad. Recently there are several deep learning architectures developed that greatly expanded the number and type of problems neural networks can address. Some of the popular deep learning architectures, Convolutional Neural Networks, Restricted Boltzman Machines, Deep Belief Networks, Recurrent Neural Networks, Long Short Term Memory units and Gated Recurrent Units are elaborated in the following sections.

**Applications of Deep Neural Networks**

Deep learning models have repeatedly demonstrated its superior performance on a wide variety of tasks including speech, natural language, vision and playing games. DNNs are used for Content editing, face recognition, market segmentation, marketing campaign analysis etc. A DNN associates the video frames with a database of pre-rerecorded sounds in order to select a sound that best matches what is happening in the scene. DNNs are good at text translation which is done without any preprocessing and the algorithm discovers the dependencies between words and their mapping to a new language.  DNN is applied for automatic handwriting generations where a corpus of handwriting examples is given and the model generates new handwriting for a given word or phrase [57].  Image classification, object detection, image restoration and image segmentation are the various applications of DNN in computer vision. Some of the popular DNN applications are listed below

*Automatic Speech Recognition:* The success of deep learning started with automatic speech recognition. DNN models are competitive with traditional speech recognizers on selected tasks.

*Visual Art Processing:* Increasing application of deep learning techniques has been developed in various visual art tasks. DNNs have proven themselves capable of identifying the style period of a given painting, capturing the style of a given artwork and applying it in a visually pleasing manner to an arbitrary photograph or video, and generating striking imagery based on random visual input fields. Deep learning-based image recognition has become excellent, producing more accurate results than human contestants.

*Image Recognition:* Deep learning-trained vehicles now interpret 360° camera views. DNNs are also applied in facial dysmorphology novel analysis that is used to analyze cases of human deformity linked to a huge database of genetic syndromes.

*Recommendation Systems:* Deep learning is used in recommendation systems to extract significant features for a latent factor model for content-based music recommendations. Multiple view deep learning has been applied for learning user preferences from numerous domains. The model enhances recommendations in multiple tasks with the help of a hybrid collaborative and content-based approach.

*Bioinformatics:* Prediction of gene ontology annotations and gene-function relationships is done by an autoencoder ANN. In medical informatics, sleep quality is predicted using deep learning using wearable data and health complications are predicted from electronic health record data. Deep learning has also showed efficacy in healthcare.

**Benefits of DNN**

In deep neural networks, features are automatically deduced and optimally tuned for desired outcome. Deep networks do not require feature extraction manually and takes input directly. Features are not required to be extracted ahead of time thus avoiding time consuming machine learning. Classical machine learning algorithms often require complex feature engineering in which a deep dive exploratory data analysis is first performed on the dataset. In conclusion the finest features must be cautiously selected to pass over to the machine learning algorithm. This can be skipped when using a deep network as one can just pass the data directly to the network and usually achieve good performance. This totally gets rid of the big and challenging feature engineering stage of the whole process.

The performance of deep networks is much better with more data than classical machine learning algorithms. Multifaceted methods are often required to improve accuracy with classical machine learning algorithms but with a deep network it can be done using more data. Deep networks are also robust to natural variations in the data and automatically learn them. Massive parallel computations can be performed using GPUs and are scalable for large volumes of data. Also it delivers better performance results when amount of data are huge.

Deep learning architecture is adaptable to novel problems in the future. Deep learning techniques can be adapted to different domains and applications far more easily than classical machine learning algorithms. Transfer learning has made it possible to use pre-trained deep networks for different applications within the same domain [58]. In computer vision, pre-trained image classification networks are often used as a feature extraction front-end to object detection networks.

**Limitations of DNN**

In a deep neural network it is not easy to comprehend output using mere learning and it needs classifiers to do so. The results of a deep network are not interpretable. The underlying process of what is happening at the neuronal level is not clear. Deep networks are very black box such that researchers do not fully understand the inside of deep networks. Classical ML algorithms are quite easy to interpret and understand as direct feature engineering is involved. The design of the network and hyper-parameters are also challenging due to the lacking theoretical foundation. Deep learning requires lot of tuning and proper parameter configuration is often difficult to be done. The presence of noisy labels affects the learning of the algorithms. It is extremely expensive to train deep networks due to complex data models. They require high-end GPUs to be trained in a reasonable amount of time with big data. GPUs are very expensive and training deep networks to high performance for large amount of data incurs high computational cost.

## 2.2 CONVOLUTIONAL NEURAL NETWORKS (CNN)

CNN is a neural network with multiple layers and its exclusive architecture is designed to capture increasingly multifaceted features of the data at each layer to decide the output [59]. CNNs are proven very effective at tasks involving data that is closely knitted together, primarily in the field of computer vision. CNNs are proved to work fine for visual recognition. Once a segment within a particular sector of an image is learned, the CNN can recognize that segment present anywhere else in the image [60]. On the other side CNN is highly dependent on the size and quality of the training data and is highly susceptible to noise. There are 4 primary steps or stages in designing a CNN namely

- *Convolution:* The input signal is received at this stage
- *Subsampling*: Inputs received from the convolution layer are smoothened to reduce the sensitivity of the filters to noise or any other variation
- *Activation:* This layer controls the flow of signals from one layer to the other

- *Fully connected:* In this stage, all the layers of the network are connected with every neuron from a preceding layer to the neurons from the subsequent layer

The fully connected neural network structure, where neurons in one layer communicate with all the neurons in the next layer, is incompetent when it comes to analyzing large images. As shown in Fig.2.2, CNN uses a three-dimensional structure in which neurons in one layer do not connect to all the neurons in the next layer; instead, each set of neurons analyzes a small region or feature of the image.

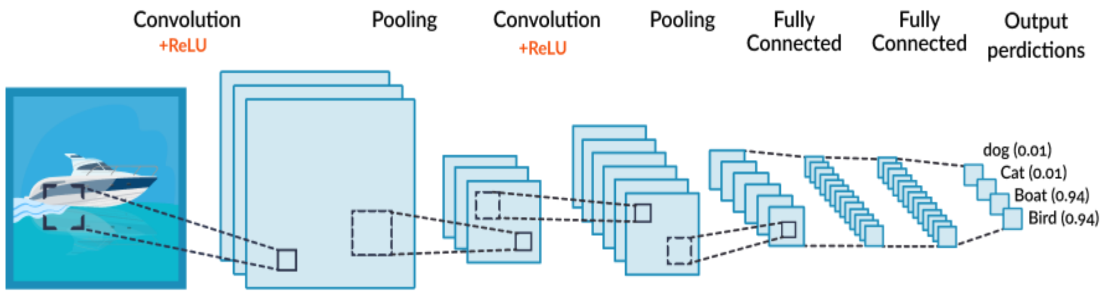

**Fig. 2.2 Architecture of CNN**

The final output of this structure is a single vector of probability scores. A CNN first performs a convolution, which involves scanning the image, analyzing a small part of it each time, and creating a feature map with probabilities that each feature belongs to the required class. The second step is pooling, which reduces the dimensionality of each feature while maintaining its most important information. The pooling function in CNN is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently.

CNNs are able to carry out numerous rounds of convolution and then pooling. Eventually, when the features are at the precise level of granularity, it creates a fully-connected network that analyzes the ultimate probabilities and decides the class of the image. The concluding step may be used for more intricate tasks, such as generating a caption for the image.

**Benefits and Limitations**

CNNs minimize computation compared to a regular neural network They simplify computation to a great extent without losing the essence of the data. They are great at handling image classification. CNNs depend on the initial parameter tuning to avoid local optima. CNNs require a substantial amount of work to initialize according to the problem at hand which is a limitation and it requires specialist knowledge in the domain.

## 2.3 RESTRICTED BOLTZMANN MACHINES (RBM)

RBM is a two layer undirected neural network consisting of visible layer and hidden layer [61]. There are no connections within each layer, but connections run visible to hidden. It is trained to maximize the expected log probability of the data. The inputs are binary vectors as it learns Bernoulli distributions over each input. The activation function is computed the same way as in a regular neural network and the logistic function usually used is between 0-1. The output is treated as a probability and each neuron is activated if activation is greater than random variable. The hidden layer neurons take visible units as inputs. Visible neurons take binary input vectors as initial input and then hidden layer probabilities. The architecture of an RBM is illustrated in Fig.2.3.

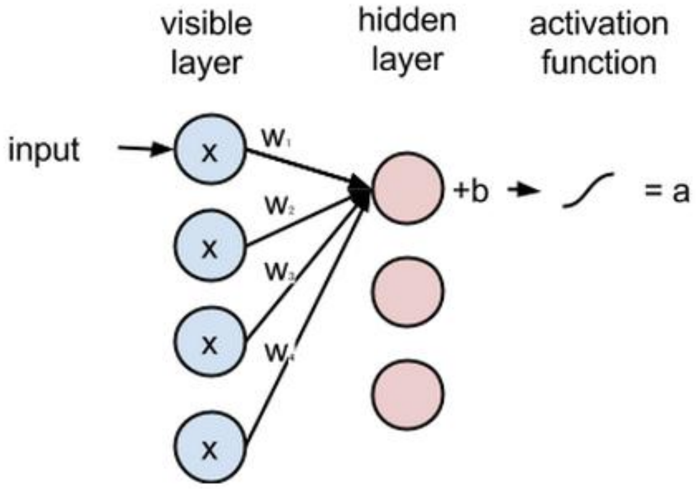

**Fig. 2.3 RBM Architecture**

In the training phase Gibbs Sampling is performed and is equated to computing a probability distribution using a Markov Chain Monte Carlo approach. In pass 1 hidden layer probabilities h is computed from inputs v whereas in pass 2 these probabilities back down to the visible layer, and to get v and h they are backed up to the hidden layer. The weights are updated using the differences in the outer products of the hidden and visible activations between the first and second passes [62]. To approach the optimal model, a vast number of passes are needed, so this approach provides proximate inference, but works well in practice. After training, the hidden layer activations of an RBM can be used as learned features.

**Benefits and Limitations**

RBM is a generative model, and is used to model the unknown distribution of data like images, text, etc. The advantage of this model is that it can create samples that look like they come from the distribution of the data [63]. RBMs can be used as feature extractors that could be trained with some other models on top or stack them to pre-train a deeper feed forward neural model. The primary disadvantage is that RBMs are difficult to train.

## 2.4 DEEP BELIEF NETWORKS (DBN)

A deep-belief network is composed of a stack of Restricted Boltzmann machines, and each RBM layer communicates with both the preceding and successive layers. The nodes of any single layer do not communicate with each other laterally. The stack of RBMs concludes with a softmax layer to create a classifier. Except the first and final layers, each layer in a DBN has a double role [64]. It acts as the hidden layer to the nodes that arrive before it, and as the input or visible layer to the nodes that appear after. It is a network built of single-layer networks. Fig.2.4 illustrates the architecture of DBN which is composed of layers of Restricted Boltzmann Machines (RBMs) for the pretrain phase and then a feed-forward network for the fine-tune phase.

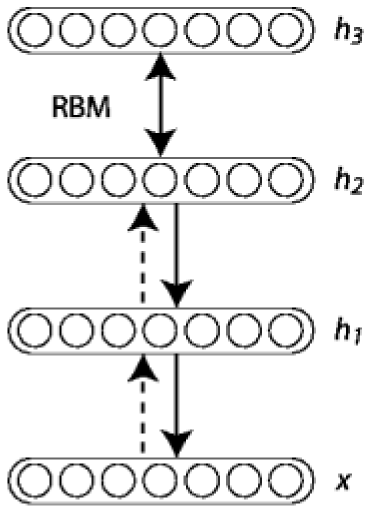

**Fig. 2.4 DBN Architecture**

The fundamental purpose of RBMs in the context of deep learning and DBNs is to learn these higher-level features of a dataset in an unsupervised training fashion. In DBNs, the building blocks for each layer are RBMs, and the principle of greedy layer-wise unsupervised training is applied. The process is as follows:

1. The first layer is trained as an RBM and the raw input is modeled as its visible layer
2. First layer is used to obtain a representation of the input that will be used as data for the second layer
3. The second layer is trained as an RBM by taking the transformed data as training examples
4. Iterate 2 and 3 for the desired number of layers, each time propagating upward either samples or mean values
5. All the parameters of this deep architecture are fine-tuned with respect to a supervised training criterion

**Benefits of DBN**

DBN has the benefit that each layer learns more complex features than layers before it. DBN can be used as a feature extraction method and also used as neural network with initially learned weights. DBNs have a better a performance than the traditional neural network due the initialization of the connecting weights rather than just using random weights in neural networks [65]. Each layer in DBN depends on Contrastive Divergence method for input reconstruction which increases the performance of the network. The greedy layer-by layer learning algorithm can find a good set of model parameters fairly quickly, even for models that contain many layers of nonlinearities and millions of parameters. Also the learning algorithm can make efficient use of very large sets of unlabeled data, and so the model can be pre-trained in a completely unsupervised fashion.

**Limitations of DBN**

The disadvantage of DBN is that the approximate inference procedure is limited to a single bottom-up pass. The model can fail to sufficiently account for ambiguity when interpreting uncertain sensory inputs as a consequence of ignoring top-down influences on the inference process. Further it learns a layer of features at a time and does not re-adjusts its lower-level parameters.

## 2.5 RECURRENT NEURAL NETWORKS (RNN)

Recurrent Neural Networks (RNNs) are artificial neural network models that are well-suited for pattern classification tasks whose inputs and outputs are sequences. The importance of developing methods for mapping sequences to sequences is illustrated by tasks such as speech recognition, speech synthesis, named-entity recognition, language modeling, and machine translation. An RNN symbolizes a sequence with a high-dimensional vector of a preset dimensionality that includes new observations using a complex nonlinear function. RNNs can implement random memory-bounded computation and are extremely expressive and as a result, they can be configured to accomplish nontrivial performance on complex sequence tasks [66]. RNNs have turned out to be difficult to train, especially on problems with complicated long-range temporal structure, exactly the setting where RNNs have to be most useful.

In a traditional neural network the assumption is that all inputs and outputs are self-regulating and not dependent on each other. As depicted in Fig 2.5 RNNs consider both the current input and a framework unit built upon previously seen data, whereas the only input ANNs take into concern is the present example they are being fed. An obvious limitation of neural networks is that they accept a fixed-sized vector as input and produce a fixed-sized vector as output performing this mapping using a fixed amount of computational steps.

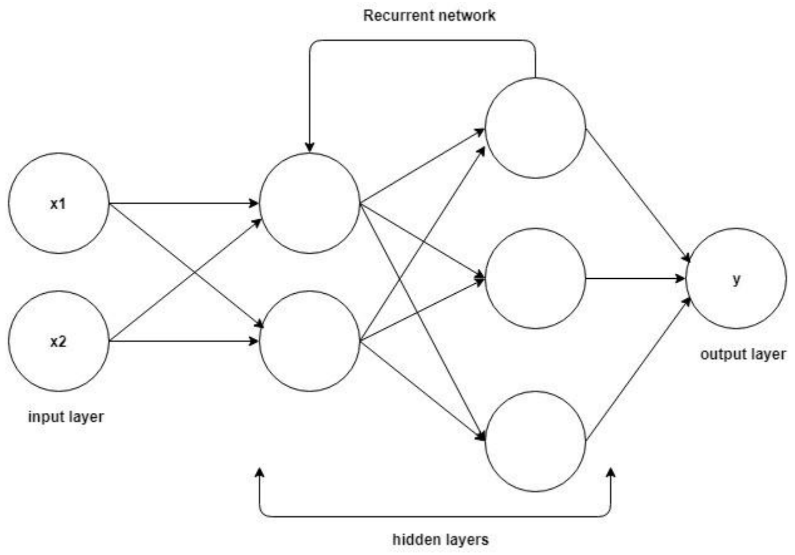

**Fig. 2.5 Architecture of RNN**

Recurrent nets are more stimulating as they operate over sequences of vectors either in the input or output, or in both. RNNs are defined as recurrent as they carry out the same task for every element of a sequence, with the output being reliant on the earlier computations. RNNs have a memory that captures information about what has been calculated so far and hence make use of information in long sequences. Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters across all steps. This reflects the fact that RNN performs the same task at each step, just with different inputs. This significantly reduces the total number of parameters needed to learn. Training a RNN is done using the back propagation algorithm. As the parameters are shared by all time steps in the network, the gradient at each output depends on the calculations of the current time step and also the previous time steps.

A recurrent neural network can be thought of as the addition of loops to the standard feed-forward multilayer perceptron architecture. For example, in a given layer, each neuron may pass its signal latterly in addition to forward to the next layer. The output of the network is given as an input to the network with the next input vector. The standard RNN is a nonlinear dynamical system that maps sequences to sequences. It is parameterized with three weight matrices and three bias vectors whose concatenation θ completely describes the RNN [67]. Given an input sequence ($x_1,...,x_t$), the RNN computes a sequence of hidden states $h_t$ and a sequence of outputs $z_t$.

Recurrent neural networks (RNNs) have recently shown attractive methods to solve machine learning tasks. RNN is extension of a traditional neural network, which is able to handle a variable-length sequence input. The variable-length sequence is solved by recurrent hidden layer whose activation is done at each time in RNN. In an RNN as shown in Fig.2.6, given input layer of sequence $x=\{x_1,x_2,\dots x_n\}$, $h_t$ is hidden layer, output layer $y=\{y_1,y_2\dots y_n\}$ the update of the recurrent hidden layer is given in equation 2.1.

$$h_t= \sigma(W_{xh}x_t+W_{hh}h_{t-1}+b_n) \qquad (2.1)$$

where σ is activation function, commonly used sigmoid function or hyperbolic tangent function, $w_{hh}$ is the weight at recurrent neuron, $w_{xh}$ is the weight at input neuron and b denotes bias vector. The output layer is computed using equation 2.2.

$$Y_t=W_{hy}h_t+b_y \qquad (2.2)$$

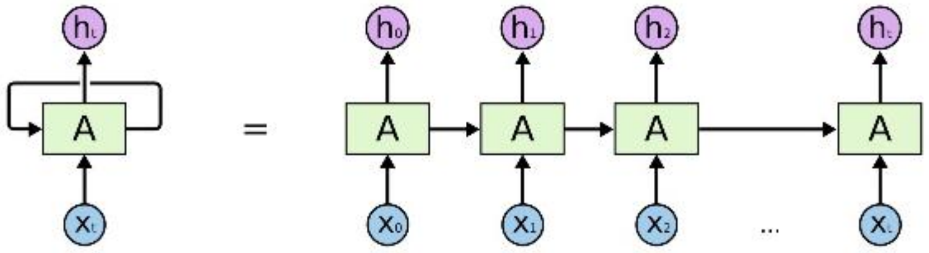

**Fig. 2.6 RNN Unfolded**

In RNN model, if the sequence consists of 3 words, the network would be unrolled into a 3-layer neural network, one layer for each word. $X_t$ is the input at time step t, $s_t$ the hidden state at time step t which is the memory of the network. The hidden state is calculated based on the preceding hidden state and the input at the present step $s_t = f\ (Ux_t+Ws_{t-1})$. The function f is a nonlinearity like tanh or Relu. $S_{-1}$ required to calculate the first hidden state, is normally initialized to zero. $O_t$ is the output at step t. $S_t$ obtains information about the earlier time steps.

The output at step $O_t$ is calculated exclusively based on the memory at time t. The recurrent connections include state or memory to the network and let it to learn broader generalizations from the input sequences. The staple technique for training feed forward neural networks is to back propagate error and update the network weights. Due to the recurrent or loop connections backpropagation breaks down in a recurrent neural network. Backpropagation through Time or BPTT is the technique used to address this issue. The configuration of the network is unrolled, and copies of the neurons that have recurrent connections are created. For example a single neuron with recurrent connection (A->A) could be characterized as two neurons with the same weight values (A->B).This allows the cyclic graph of a recurrent neural network to be turned into an acyclic graph like a classic feed-forward neural network and backpropagation can be applied [68]. The vanishing gradient problem is alleviated in deep multilayer perceptron networks through the use of the rectifier transfer function, and even more exotic but now less popular approaches of using unsupervised pre-training of layers.

RNNs are a promising solution to tackle the problem of learning sequences of information. There are many forms of sequence prediction problems depending on the types of inputs and outputs supported. RNNs allow sequences of vectors to be used in the input, output, or in the most general case both [69]. The different types of RNN based on the inputs and outputs are given in Fig.2.7. Each rectangle represents vectors and arrows represent functions. Some examples of sequence prediction problems using RNN include:

- *One-to-one*: It deals with fixed size of input to fixed size of output where they are independent of previous information. Example: Image classification.
- *One-to-Many*: An input observation is mapped to a sequence with multiple steps as an output. Example: Image captioning
- *Many-to-One*: A sequence of multiple steps as input mapped to class or quantity prediction. Example: sentiment classification
- *Many-to-Many*: An input sequence of multiple steps is mapped to a sequence with multiple steps as output. Example: Machine translation

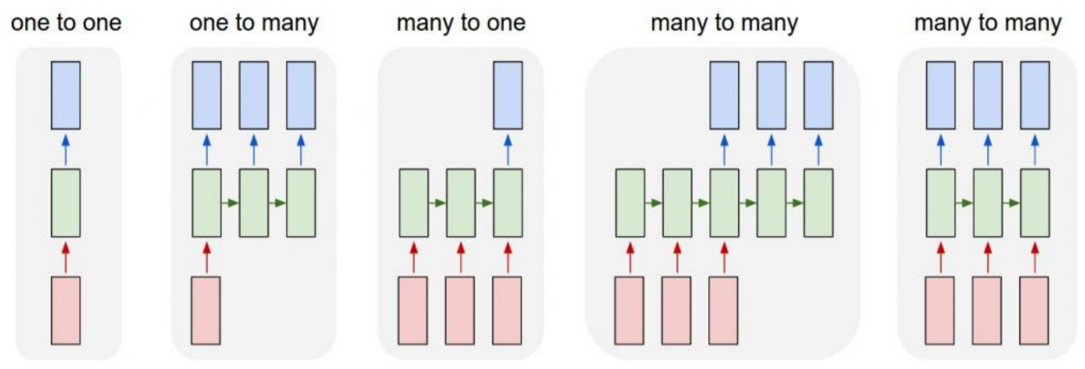

**Fig. 2.7 RNN Types Based on Input / Output**

An example for one to many networks is labelling an image with a sentence. The many to one approach could handle a sequence of image and produce one sentence for it. The many to many approaches is used for language translations. Other use cases for the many to many approaches could be to label each image of a video sequence.

**Training through RNN**

There are various steps involved in training an RNN and the foremost one is providing single time step of the input to the network. Then its current state is calculated using set of current input and the previous state. At the next time step, the current $h_t$ becomes $h_{t-1}$. One can go as many time steps according to the problem and join the information from all the previous states. Once all the time steps are completed the final current state is used to calculate the output. The output is then compared to the actual output i.e the target output and the error is generated. The error is then back-propagated to the network to update the weights and hence the network is trained.

Recurrent Neural Network model suffers from a drawback due to the diminishing effect of the inputs, which are farther into the past, causing short term memory effect. Although the gradients of the RNN are easy to compute, RNNs are fundamentally difficult to train, especially on problems with long-range temporal dependencies [70], due to their nonlinear iterative nature. Even a little change to an iterative process can multiply and result in great effects many iterations later. The inference is that in an RNN, at a particular time, the derivative of the loss function can be exponentially large with respect to the hidden activations at a much earlier time. Thus the loss function is very sensitive to small changes, so it becomes effectively discontinuous. RNNs also experience the vanishing gradient problem which was first described by Hochreiter [71]. RNN has its variant forms like Long Short Term Memory units (LSTM) and Gated Recurrent Units (GRU) which are developed to overcome the vanishing gradient issue and they are discussed in detail in Sections 2.3 and 2.4 respectively.

**Benefits of RNN**

The main advantage of RNN over ANN is that RNN can model sequence of data so that each sample can be assumed to be dependent on previous ones. The default feed forward network can just compute one fixed-size input to one fixed size output. With the recurrent approach one to many, many to one and many to many inputs to outputs are possible. RNNs can handle sequential data of arbitrary length and are used for handwriting recognition and speech recognition. The ability to work with sequences makes RNN applications versatile as they accept variable sized inputs and can also produce variable size outputs. They have a built-in feedback loop, which enables them to remember current and past inputs while arriving at a decision making them good at forecasting.

In a neural network there is no memory associated with the model, which is a problem for sequential data. RNN addresses that issue by including a feedback loop which serves as a kind of memory. Hence a footprint is left out by the past inputs to the model. The RNN uses the feedback connection to store information over time in form of activations. This ability is significant for many applications involving sequential data because each neuron or unit can use its internal memory to maintain information about the previous input. These little memory units allow RNNs to be much more accurate and are the reason behind the popularity of this model.

**Limitations of RNN**

The downside of RNN is that it can be much more difficult to train and has multiple issues with convergence. In unrolled RNN, the gradients that are calculated in order to update the weights can become unstable when backpropagation is used. They can turn out to be very large numbers called exploding gradients or very small numbers called the vanishing gradients [72]. When these large numbers are used to update the network weights, training becomes unstable and the network is unreliable. Memorization in RNNs continues to pose a challenge in many applications. RNNs are required to store information over many timesteps and retrieve it when it becomes relevant but vanilla RNNs often struggle to do this. RNNs find it complex to process very long sequences if using tanh or relu as an activation function.

## 2.6 LONG SHORT-TERM MEMORY UNITS (LSTM)

The Long Short-Term Memory or LSTM network is a recurrent neural network introduced [73] to address the vanishing gradient problem. It can be used to deal with difficult sequence problems in machine learning and achieve state-of-the-art results. LSTM networks have memory cell instead of neurons that are connected into layers.

A LSTM memory cell as shown in Fig.2.8 has components that make it smarter than a classical neuron and a memory for recent sequences. A block consists of gates that manage its state and output. A unit works upon an input sequence and sigmoid activation function is used by each gate within a unit to control whether they are triggered or not, making the change of state. A linear summing unit is at the core and at a given time step it aggregates the combined input it receives. Its self recurrent connection has a fixed weight of 1.0 that prevents exponential decay of activations from way back in the past, thereby solving the drawback of conventional RNNs. There are three types of gates within a memory unit namely the forget gate that conditionally decides what information to discard from the unit, the input gate that conditionally decides which values from the input to update the memory state and the output gate that conditionally decides what to output based on input and the memory of the unit.

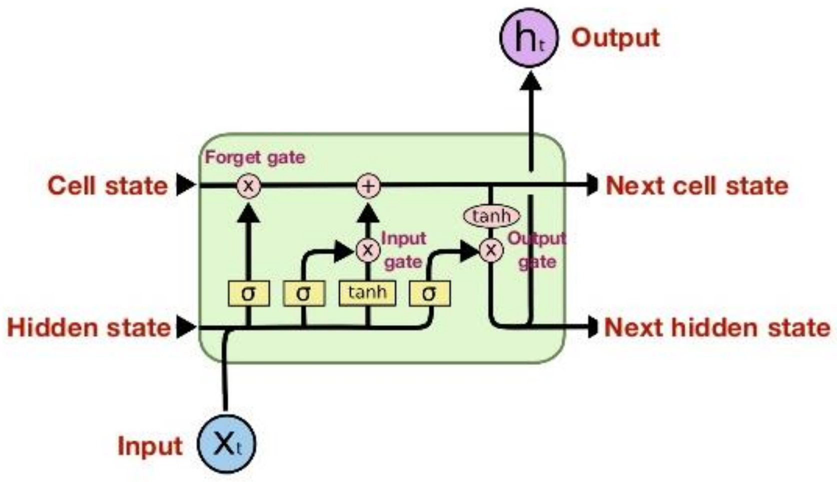

**Fig. 2.8 LSTM Cell**

The input gate modulates whether the input from other cells or layer of cells are to be fed to the linear unit at the core, thereby modulating the effect of particular input vectors on the state of the cell. The output gate modulates whether the output of the linear unit is to be broadcasted to the network and hence modulates the effect of this cell on the output of the network. The forget gate sets the fixed weight of self recurrent connection to 0 thereby resetting the state of the cell at appropriate times, and helps the cell behave like a counter for certain application. This also helps it from being influenced by inputs very long back in the past that does not have any bearings on the present input.

Gates consist of a sigmoid neural net layer and a pointwise multiplication operation. The output of sigmoid layer is numbers between zero and one, relating how much of each component should be let through. A value of zero means to allow nothing while a value of one means allow everything. The repeating module in an LSTM has a different structure than an RNN and has four, interacting components as shown in Fig.2.9. The forget gate layer receives the values $h_{t-1}$ and $x_t$, and then outputs a number between 0 and 1 for each number in the cell state $C_{t-1}$. The values to be updated are decided by the input gate. A vector of new candidate values, $C_t$ that could be added to the state is created by a tanh layer.  These two values are combined in the next step and the old cell state, $C_{t-1}$ is updated into the new cell state $C_t$.

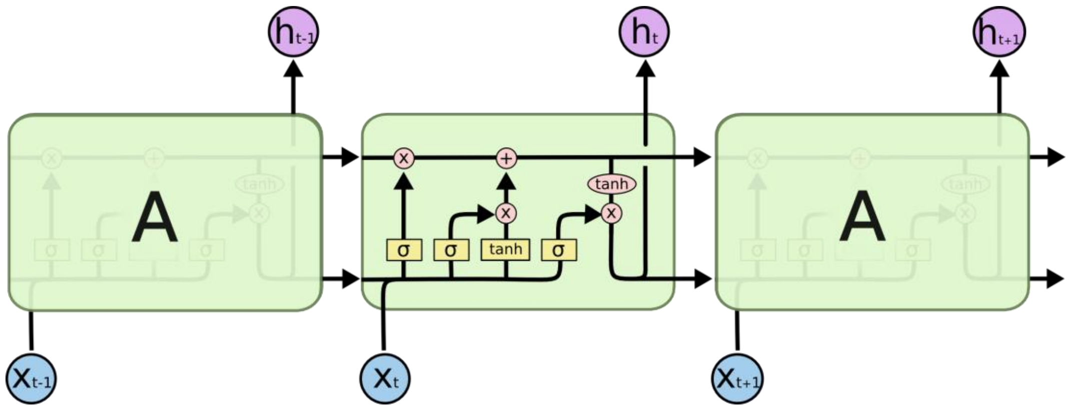

**Fig. 2.9 Repeating Module in LSTM**

A detailed discussion of the three gates namely forget gate, input gate and output gate of the LSTM cell is presented below.

***Forget gate***

A forget gate is accountable for removing information from the cell state. The information that is of less importance or the information that is no longer required for the LSTM to comprehend things is removed by multiplication of a filter. This is necessary for optimizing the performance of the LSTM network [74]. This gate takes in two inputs $h_{t-1}$ and $x_{t,}$, where $h_{t-1}$ is the hidden state from the previous cell or the output of the previous cell and $x_t$ is the input at that particular time step. Multiplication of the given inputs by the weight matrices is done and a bias is added. Next the sigmoid function is applied to this value. The sigmoid function outputs a vector, with values ranging from 0 to 1, which corresponds to each number in the cell state and is multiplied to the cell state.

***Input Gate***

The input gate is accountable for the addition of information to the cell state, which is essentially a three-step process

a. Regulating what values need to be added to the cell state by involving a sigmoid function. This is similar to the forget gate and acts as a filter for l the information from $h_{t-1}$ and $x_t$.
b. Creating a vector containing all possible values that can be added to the cell state by using the tanh function, which outputs values from -1 to +1.
c. Multiplying the value of the regulatory filter to the created vector and then adding this useful information to the cell state via addition operation.

***Output Gate***

There are three steps in the functioning of an output gate

a. Creating a vector after applying tanh function to the cell state, thereby scaling the values to the range -1 to +1.
b. Making a filter using the values of h_t-1 and x_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function.
c. Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as a output and also to the hidden state of the next cell.

### Benefits of LSTM

RNNs are prone to vanishing gradients as they can remember information for a short duration of time only. This is addressed with a better architecture like a Long Short-Term Memory (LSTM) that resolves issues of vanishing gradients. LSTMs are clearly designed to evade the long-term dependency crisis and can retain information for long periods of time [75]. Since the output is based on previous computations, it is vital to remember previous inputs. A standard RNN can look back a few time steps only, which is not enough to produce the most accurate results when more parameters are added. An LSTM, on the other hand, can look back many time steps and are considered best suited for applications like sequence predictions, language translation, speech to text and for any information rooted in time like video. An LSTM can forget and remember patterns selectively for a long duration of time. The LSTM has the ability to remove or add information to the cell state, carefully regulated by structures called gates. It understands which data is important and should be retained in the network and this makes LSTM competitive over simple RNNs and feed forward neural networks.

### Limitations of LSTM

LSTMs have an update gate and forget gate which makes them more sophisticated but at the same time more complex as well. It is more costly to compute the network output and apply back propagation. The explicit memory adds several more weight to each node, all of which must be trained which increases the dimensionality of the problem, and potentially makes it harder to find an optimal solution. It takes much time and resources to run this model.

## 2.7 GATED RECURRENT UNITS (GRU)

GRU addresses the vanishing problem by replacing hidden nodes in traditional RNN by GRU node. The core idea of GRUs is that the gradient chains do not disappear due to the length of sequences as values are passed completely through the cells. Fig.2.10 shows the architecture of GRU. Each GRU node consists of two gates, update gate $Z_t$ and reset gate $r_t$ . GRU has two gates, a reset gate and an update gate.  The reset gate decides how to merge the new input with the previous memory, and the update gate identifies how much of the previous memory to keep around [76]. When reset is set to all 1's and update gate to all 0's again arrive at our plain RNN model.

**Reset Gate:** Essentially, this gate is used from the model to decide how much of the past information to forget while calculating present information.

**Update Gate:** The update gate $z_t$ helps the model to determine how much of the past information from previous time steps need to be passed to the future. It is really influential as the model can decide to copy all the information from the past and eliminate the risk of vanishing gradient problem.

Update gate determines how much the unit modifies its activation, or content. It is computed in equation 2.3. Reset gate allows the unit to forget the previously computed state and is calculated by equation 2.4. The hidden layer is computed by equation 2.6 using $h_t$ which in turn is calculated by equation 2.5. The network computes $h_t$ that holds information for the current unit and sends it down to the network. The update gate determines what to collect from the current memory content and previous memory content. The model keeps a majority of the previous information by learning to set the vector $z_t$ close to 1. At this time step $z_t$ will be close to 1 and $1 - z_t$ will be close to 0 and so big portion of the current content and vice versa will be ignored.

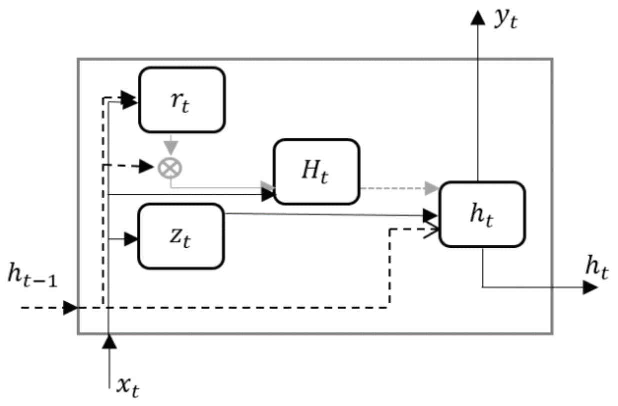

**Fig. 2.10 Gated Recurrent Unit**

The update functions are calculated as follows:

$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$ (2.3)

$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$ (2.4)

$h_t = \tanh(W x_t + U(r_t\ h_{t-1}) + b)$ (2.5)

$h_t = (1 - z_t)\ h_{t-1} + z_t .\ h_t$ (2.6)

where matrices $W_z$, $W_r$, $W$, $U_z$, $U_r$, $U$ and vectors $b_z$, $b_r$, $b$ are model parameters.

**Difference between LSTM and GRU**

GRU is a simplified version of LSTM where both state vectors are merged into a single vector $h_t$. Both the forget gate and the input gate are controlled by a single gate controller. The forget gate is open and the input gate is closed when the gate controller outputs a 1. It is vice versa if it outputs a 0. There is no output gate in GRU and the full state vector is output at every time step. But there is a new gate controller that controls which part of the previous state will be shown to the main layer. Though there are a few key differences, the primary idea of using a gating mechanism to learn long-term dependencies similar to LSTM. There are two gates in a GRU whereas an LSTM has three gates. GRUs do not possess an internal memory that is different from the exposed hidden state [77]. The output gate that is present in LSTMs is not there in GRU. The input and forget gates are coupled by an update gate z and the reset gate r is applied directly to the previous hidden state. Thus, the responsibility of the reset gate in a LSTM is really split up into both r and z.

**Benefits of GRU**

GRUs with their internal memory capability are valuable to store and filter information using the update and reset gates. The issues faced by RNNs are eliminated and GRU offers a powerful tool to handle sequence data. The gates allow a GRU to carry forward information over many time periods in order to influence a future time period [78]. In other words, the value is stored in memory for a certain amount of time and at a critical point is pulled out and used with the current state to update at future.

The success of GRU is due to the gating network signals that control how the present input and previous memory are used to update the current activation and produce the current state. These gates possess sets of weights that are adaptively recomputed in the learning phase. The GRU unit controls the flow of information without having to use a memory unit. GRU is relatively new, and the performance is on par with LSTM, but computationally more efficient. Compared to LSTMs, GRUs train faster and perform better on less training data. They are simpler and thus easier to modify, for example adding new gates in case of additional input to the network. It is computationally easier than LSTM since it has only two gates.

**Limitations of GRU**

Though these models ensure learning in RNN, they introduce an increase in parameterization through their gated networks. GRUs expose the entire cell state to other units in the network and are difficult to train.

## SUMMARY

Deep architectures have the potential to integrate diverse data sets across heterogeneous data types because of their hierarchical learning structure. They also provide greater generalization given the focus on representation learning and not simply on classification accuracy. Various deep learning architectures, the concepts behind deep learning approaches, their benefits and limitations are presented in detail in this chapter. The rapid developments in the design of deep architecture models have opened avenues for the application of these models in research. These deep learning architectures are able to extract automatically useful structures from input patterns. The different architectures of deep learning namely DNN, RNN variants BRNN, LSTM and GRU which are elaborated in this chapter have been used to build ASD predictive models.