3. Early Stopping - The aim is to avoid the problem of overfitting by training the model just enough to capture the basic structure of the dataset. The processing of stopping early is one solution of solving the problem of overfitting.
4. Adding Noise - The focus is to increase the dataset by injecting the noise into the existing dataset. The idea is to add noise to the dataset. This avoids the problem of overfitting as the model may not perform well for the dataset. Instead, it may increase the chance of capturing the underlying structure of the dataset.

16.4 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNN) are multilayer neural networks that are designed to recognize visual patterns directly from images. The CNN performs image classification, which means taking an image as input and specifying the class as output. Humans recognize objects in a fraction of time. Computers, on the other hand, could not carry out the recognition process till the arrival of deep neural networks. The CNN takes an image as an array of numbers and outputs the probability of the image belonging to a class.

Traditionally, features of objects are extracted (say triangle has three sides and squares have four sides) and used for classification. The CNN, however, automatically extracts the features such as edges and curves, and then, builds the higher concepts through a series of convolutional layers. The biological basis of this concept was provided by Hubel and Wiesel in 1962. The CNN is a special kind of neural network for image classification. It takes an image as input and performs various image related tasks such as image classification, object detection, object localization and image generation.

The CNN architecture is given in Figure 16.2.

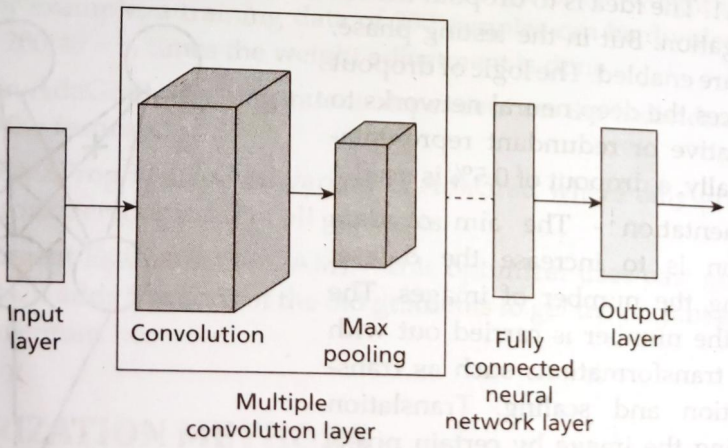

Figure 16.2: Design of a Convolutional Neural Network

The major layers in most of the CNN architectures are convolutional layers, pooling layers, ReLU layers and fully connected layers. Since images are the input to CNN, input layer is often two dimensional in nature. The output of CNN is the number of classes, and therefore, it is one-dimensional.

The CNN provides a hierarchical learning. The other types of learning are discussed in Box 16.1.

Box 16.1: Scalable Machine Learning (Online and Distributed)

Scalable machine learning techniques are evolving nowadays for analysing large amounts of data, that is, big data that powers the next generation of internet applications. In order to analyse such a large-scale, real-time unstructured data, the processing paradigms and framework and statistical analysis methods, machine learning algorithms for real-time data streams have to be explored and scaled for these aspects. Real-time, online distributed applications such as social recommender systems, real-time analytics of IoT data streams, spam filtering, topic models and document analysis pose a challenge in the development of scalable machine learning. These scalable machine learning algorithms when scaled to work for much larger sets of data, particularly for exabytes of data, also bring in another challenge of developing a system that can work across multiple nodes in a distributed manner. Hence, when machine learning processes interact among distributed multiple nodes, a different approach is required. Deep learning can be adopted for scalable machine learning processes and has to continuously evolve to accommodate large scale data in a distributed manner more efficiently. MapReduce is a programming model that provides a distributed computation framework to allow parallelization of computations. Popular machine learning algorithms such as Stochastic Gradient Descent (SGD), ensemble models or deep neural networks can be deployed on top of such parallel processing framework to account for large scale processing of real-time data.

One can implement CNN in many ways. Many representations lead to different CNN architectures. One simple architecture of the CNN is shown in Figure 16.3. This is called a LeNet CNN architecture.

Let us discuss about the individual components of CNN one by one.

Input Layer
Convolutional Layer
Average Pooling
Convolutional Layer
Average Pooling
Fully Connected Network
Output Layer

Figure 16.3: Implementation Detail of a Sample Convolutional Network

Input Layer

The input for CNN is the image. An image is a 2D signal that varies over the spatial coordinates x and y. This can be written mathematically as $f(x, y)$.

Scan for *'Additional Information on Input Layer'*

Convolutional Layers

Convolutional layer is the most important layer. It is named after the mathematical operation called convolution. It is also called as CONVNET. It has a set of filters, which are also known as masks or kernels. The filters are convolved with the input to give either activation map or feature map.

This layer reads an image and examines a small area using filters. The portion of the image examined is called receptive field. A filter is a small matrix of 3×3 or 5×5 size. One can visualize this matrix as a set of numbers called weights or parameters. Typically, the filter performs the process of convolution. The purpose of convolution operation is to identify the features of the image such as straight edges, curves and colours. So, one can visualize convolution operation as feature identifiers.

Based on the convolutional filters, feature maps are produced. Initially, a spatial mask called filter or kernel or receptive field, is designed and superimposed on the given image. Normally, the spatial mask is of a smaller size than the image. Traditional sizes of the mask size are 3×3, 5×5 and 7×7. The spatial masks or filters have fixed weights and these weights determine the kind of feature extracted. Once the spatial mask or filter is superimposed on the given image, convolution is done. The convolution operation is the multiplication of the weights of spatial masks or filters with the image pixels. Then, these multiplications are summed to give a single number in the output image. Then the spatial mask is moved and the convolution process is repeated throughout the image. The resultant image is called an activation map or feature map.

The algorithm for convolution is given below.

Algorithm 16.2: Convolution

Step 1: Superimpose a filter on the image.

Step 2: Perform element-wise multiplication of the weights of filters and image.

Step 3: Sum the element-wise products to produce the output value in the resultant image.

Step 4: Repeat the steps 1-3 for all the locations of the image.

The generated activation map or feature map is dependent on parameters of convolution operation. Some of the essential parameters are listed below:

1. Number of Filters - Choosing the number of filters is an important criterion, as filters can learn features. But more filters add to the complexity of the operation. The number of filters is called depth.
2. Filter Size - Filters are also known as kernels or receptive fields. Traditionally, a 3×3 filter is used. The other sizes are 5×5 or 7×7. Optimal size of the filters is important in learning features. If the kernel size is too high, then the complexity increases as more computations are required due to an increase in the competing features. On the contrary, if the kernel size is too low, the kernel may not be able to learn any useful features.
3. Stride - Stride is an important *hyperparameter* that modifies the behaviour of the layer by controlling the shifts of the filters. To apply convolution, the filter is just a 'slide' and by default the slide value is 1. Instead of sliding by one pixel, one can choose a larger slide with 2, 3 or 4 pixels. The parameter that defines the value of the slide is called a stride. Normally, a stride of 1 or 2 is used. By increasing the number of strides, the overlapping pixels can be avoided. It also increases the speed. But the disadvantage of more slides is the skipping of regions and an increase in many more features.

Example 16.1 illustrates the convolution of two 1D images.

Example 16.1: Illustrate the use of Stride with input data {1, 3, 3, 2, 1, 0, 1} and filter {1, 1, 1}.

Solution: The convolution operation with the stride of 1 is shown in Figure 16.4. The convolution operation with the stride of 2 is shown in Figure 16.5. It can be observed that with the increase in stride, the dimension of the output decreases.

The logic of stride can be extended for 2D too. The following Figure 16.6 illustrates the usage of stride for two dimensions.

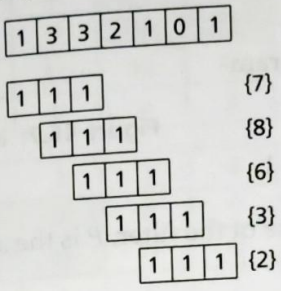

Figure 16.4: Convolution with Stride 1

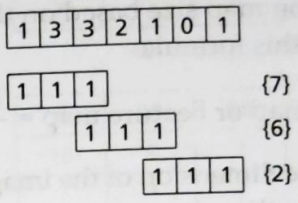

Figure 16.5: Convolution with Stride 2

4. Padding - Padding is another important hyperparameter for convolution operation. Applying convolutions reduces the spatial volume. It is often necessary to preserve as much information as possible. So, zero is padded around the image so that there is more space and the kernel can move. Padding is a technique of ensuring that the feature map has same dimension as input image. It adds a layer of zero to the edge of the image.

If one prefers to have an output image with the same dimension as the input, one can add zeros ınd the image so that the filter can move over a larger place. Zero padding is the addition of s around the edges of the image to ensure that the convolution results in a size with the same imension as the input. In other words, the output feature map is controlled by the padding size.

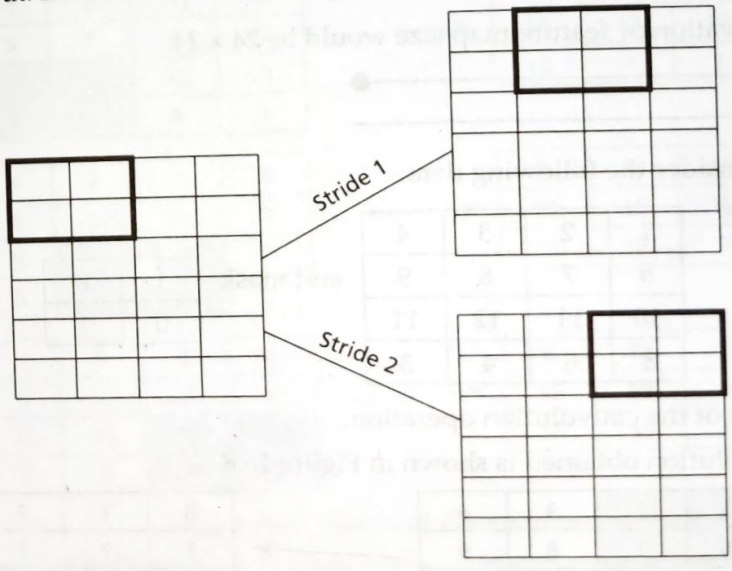

Figure 16.6: Positions of Masks After Stride with 1 and 2

The following Figure 16.7 illustrates the concept of zero padding. In general, zero padding is given as K–1/2. Here, K is the kernel size which ensures that the input and output have same spatial dimensions.

The depth of the output volume depends on the number of filters used as part of the convolutional layer. This is called as a hyperparameter as it is used to control the output volume. The other useful hyperparameters are stride and zero padding.

The activation map size based on these hyperparameters is given in this formula:

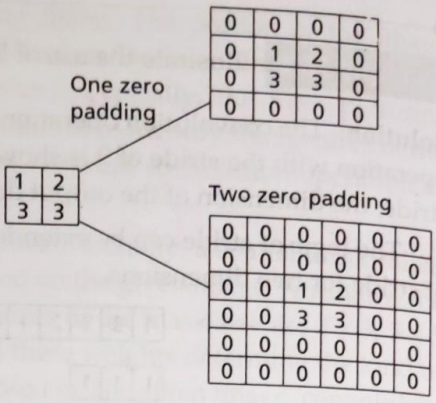

Figure 16.7: Illustration of Padding

$$\text{Activation map or Feature map} = \frac{D - F + 2P}{S} + 1 \tag{16.9}$$

Here, D is the dimension of the image, F is the size of the filter, P is the amount of padding and S is the stride length.

Scan for *'Additional Information on Padding'*

Example 16.2: Let us assume the input size is 28 × 28. If the filter size is 5 × 5, stride is 1 and padding is zero, what is the size of the feature map?

Solution: Using Eq. (16.9), one can observe that $D = 28$, $F = 5$, $S = 1$ and $P = 0$. Therefore,

$$\text{Activation map or Feature map} = \frac{28 - 5 + 2 \times 0}{1} + 1 = 24$$

The output activation or feature map size would be 24 × 24.

Example 16.3: Consider the following data:

1	2	3	4
5	7	8	9
10	11	12	11
8	6	4	3

and mask

1	0
0	1

Show the results of the convolution operation.

Solution: The convolution obtained is shown in Figure 16.8.

1	2	3	4
5	7	8	9
10	11	12	11
8	6	4	3

→

8	?	?
?	?	?
?	?	?

This value is obtained by multiplying the elements of the image and the mask. The mask is moved into the next position.

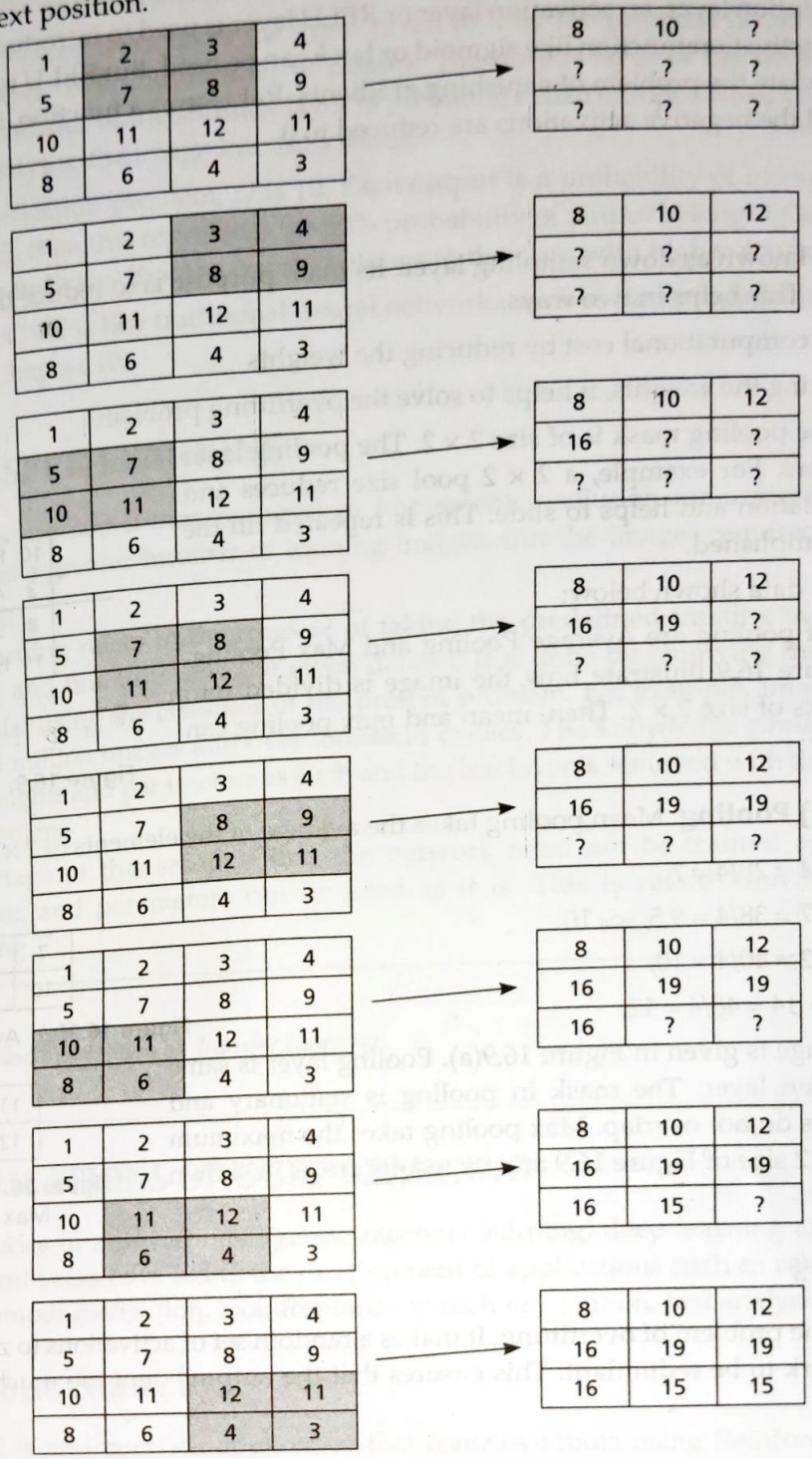

Figure 16.8: Stages of Convolution

Since, all locations of the image are processed, the convolution operation stops. This is the final feature or activation map.

16.6.7 Text Analysis

Text analysis is the automated process of automatically extracting and classifying unstructured text data such as social media comments, emails, reviews, surveys and responses with machine learning techniques to mine for useful information. Text classification is the process of assigning pre-defined tags or categories to unstructured text through statistical pattern learning. Common text classification tasks include language translation, sentiment analysis, topic modelling, language detection and intent detection. Language translation uses machine learning techniques to translate a sentence from one language to another. Sentiment analysis uses powerful machine learning algorithms to automatically identify opinion polarity (positive, negative, neutral) towards any topic or product and the emotions and sarcasm. Another common example of text classification is topic analysis (also called as topic modelling) that automatically organizes text by topic, subject or theme. Text classifiers can also be used to determine the intent of a text or more appropriately the reason behind customer feedback and so on. Text extraction is another widely used text analysis technique that extracts pieces of meaningful data that already exists within any given text. The most popular machine learning algorithms such as Support Vector Machines, Bayesian Networks, Conditional Random Field and Neural Networks/Deep Learning, etc. are used for all these text analysis applications.

16.7 RECURRENT NEURAL NETWORKS

The major problem of CNN is that its input should be fixed. Recurrent neural networks (RNN) are used for processing sequential data. What is sequence data? A sequence data imposes an explicit order on the observations and is one of the most important aspects and should not be changed as input or output.

Some of the sequence data are:

1. Time series data
2. Weather data that is observed over a time period
3. Sequence DNA data as a string of A, C, G and T
4. Sentiment analysis data as a sequence of text with sentiment such as happy or excellent

One example of sequence data is time series data. Some quantity of data that is measured sequentially in time over an interval of data is called time series data. A time series data is where each data is associated with a time stamp. For example, a stock exchange data is a time series data. Time series analysis is all about inferring the future prediction based on the past data. Time series analysis is used to understand the data. It can be used to understand past and future trends too.

The time series is called stationary if it does not have seasonal effects. It has consistent statistical properties such as mean, variants and covariance. The time series data needs to be made stationary if statistical computation should be done. The non-constant mean and variation at specific time frame are characteristics of non-stationary data. Hence, it is necessary to make it stationary for forecasting.

Baseline value of the time series is called level. A trend is a consistent directional movement that often is linearly increasing or decreasing and is present in the base data. This trend can be deterministic or stochastic. For example, one can use time data to analyse the price of a particular item.

Seasonal variation is about analysing with respect to the seasons or annual temperature variation. Seasoning is the optimal repeating pattern or cycles of behaviour over time. Serial dependence is where data is close together and tends to be correlated.

Data that cannot be described by the time series is called noise. Some of the time series analysis are forecasting/predicting the values, simulation of time series and inferring relationships among time series data and other values.

Data pre-processing needs to be done to remove noise, outliers and to adjust the data. The trend is adjusted using transformations such as log, square root and cube root. Trend is then modelled by smoothing. Some of the methods are listed below:

- Moving average method (MA models combine to give ARMA models)
- Exponential smoothing

In the moving average method, k consecutive values are averaged. The choice of k depends on the frequency of data. In some cases, exponentially waited moving average is used to make short term prediction of the future data. Methods like differencing and seasonal decomposition are used to find trend and seasonality. Seasonal trends displace the cyclical effects of seasonal components daily, weekly and yearly. ARIMA models are also used for time forecasting. ARIMA models use auto correlation in data. These methods use historical information to make predictions. ARIMA is used for finding varying mean and variance for a non-stationary signal.

Deep learning neural network can be used to predict trends in time series data as well. There may be many problems coined as sequence generation and sequence prediction. RNN has only one hidden layer. This hidden layer has memory buffer that stores and feeds it back into the hidden layer with the next input. This means it is processed in the context. Context is an important concept in RNN. For example, consider the sentence – Ramesh lives in France. French is the native language of France. He knows the native language of France. Here, 'he' refers to Ramesh that is given in the first sentence and 'native language' refers to French that is mentioned in the second sentence.

In general, natural text can have multiple words and sequence data many involve many time units where the word is given as input for RNN. A sample of RNN for 3 inputs is shown in Figure 16.11.

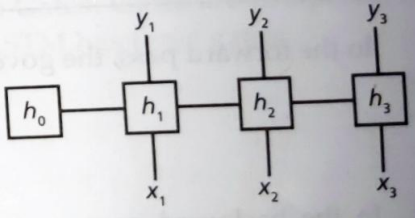

Figure 16.11: A RNN for Three Time Units Sequence Data

For example, at time step t_1, the input x_1 is processed to give output y_1. The hidden layer generates a set of activations that are passed from input to output along the arrow that connects the hidden state. In general, RNN is ideal for sequence data whose size is not known before. RNN contains directed cycles. A neuron in a layer can transmit a signal to another neuron in the same layer or to the connection of neuron in the next layer. The output of a layer may also serve as the input for the next input sequence. The RNN is shown in Figure 16.12.

A memory or state is added because of the recurrent connections. A state vector is used to represent the state. The input combined with the state vector produces the output vector. The decision at step 't' is influenced by the time step t–1. The hidden state can be calculated as follows:

The weight W_{xh} is used for all weights from input to hidden layers. W_{hh} is used for all weights from h_{t-1} to h_t links and W_{hy} is used for all links from hidden to output. b_h and b_h are the biases for hidden and output layers. Then, the governing equations are given as below:

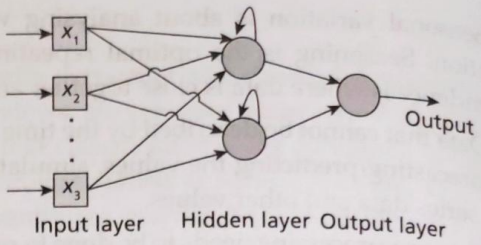

Figure 16.12: Model of Recurrent Neural Network

$$h_t = \tan h(W_{xh} x_t + W_{hh} h_{t-1} + b_h)$$
$$y_t = W_{hy} h_t + b_y \tag{16.10}$$

Training of RRN Networks

RNN is trained using the backpropagation algorithm as feed forward network. Since RNN has loops, backpropagation does not work in the same manner. So, a new algorithm called Backpropagation through time (BPTT) is used.

The outline of BPTT algorithm is given below.

Algorithm 16.3: BPTT

Step 1: The First step of BPTT algorithm is to unfold RNN.

Step 2: A copy of neurons that contains loops is created and cyclic graphs are converted into acyclic graphs.

Step 3: RNN is converted to feed-forward network.

Step 4: RNN is constructed as a chain of functions and derivatives are computed and chain rule is applied.

In the forward pass, the governing equations are implemented as:

$$h_t = \tan h(W_{xh} x_t + W_{hh} h_{t-1} + b_h)$$
$$y_t = W_{hy} h_t + b_y \tag{16.11}$$

In the backward pass, cross entropy loss function is used. It can be recollected that the cross-entropy loss function is given as:

$$L = -\ln(p_c) \tag{16.12}$$

Here, p_c is the number of correct classifications. The partial derivatives of L, W_{hy} and W_{xh} are obtained in the backward pass and are backpropagated for all time stamps. This is being done in BPTT algorithm. A variant of BPTT algorithm is Truncation Backpropagation Through Time algorithm. It is used as an approximation to RNN that cannot be unfolded.

Major Problems of RNN

The major problems of RNN are vanishing gradient and exploding gradients as discussed below.

Vanishing Gradients The training of RNN requires the back propagation of error throughout the length of the sequence. This requires back propagation through all hidden layers. It is possible through multiplication of error with the weights. Since the weights are same, K time steps are required for multiplication of weights k-times. This leads to a weight raised to the power of K. If the weight is less, it leads to diminishing at an exponential rate and ultimately vanishes when length is large. When weights are multiplied with gradients, it becomes small and closer to zero. This problem is called vanishing gradients. The problem of vanishing gradients can be minimized to some extent by initializing the network weights to identity matrix.

Exploding Gradients Exploding gradients is an issue of deep learning where the model's weights grow exceptionally large during training. Some of the weights grow and become NaN (Not a Number) during training. The problem of exploding gradients is solved by rescaling the values. If the value crosses the threshold, then its value is clipped or reduced by applying a penalty to the network loss function for larger weight values.

The problem of vanishing gradients can be solved by ReLU activation functions. Another way to solve these problems is to use long short-term memory (LSTM) and gated recurrent unit (GRU) variants of RNN.

16.8 LSTM AND GRU

The LSTM and GRU variants of RTU are discussed in the section below.

LSTM

LSTM is a variant of RNN. It solves the problem of vanishing gradients. LSTM holds information in a gated cell. A cell can accumulate information, and can read and write into its memory. The core idea of LSTM is a cell state that stores information. LSTM can add or remove information to the cell state using a regulated structure called Gates. Gates are composed of sigmoidal function and a pointwise multiplication operation. LSTM has three gates:

1. Forget gate
2. Input gate
3. Output gate

Forget Gate Forget gate is responsible for determining the activations of a cell that need to be forgotten. The input vector x_t is concatenated with the hidden state vector that has been propagated through step h_{t-1} and passes through a sigmoid activation function. The sigmoid function outputs value in the range 0 to 1. The value 0 represents the state of completely removing the information and the value 1 represents the state of keeping the complete information. The output of the sigmoidal function is multiplied with the cell state. Thus, sigmoidal function can be used as a filter to keep the relevant information and get rid of all unnecessary information.

For example, Ram is in France. All people in France know French. So, he knows French. Here, he refers to Ram and must be remembered. This information will be useful for subsequent predictions.

Input Gate or Update Gate Update gate or Input gate determines when to update and what to update in response to any new input. The input x_t is concatenated with h_{t-1} and passed through a sigmoidal function to generate a vector. The sigmoidal function is called input gate as its output decides which values need to be updated. A value of zero indicates that the values need not be updated and a value of 1 indicates that the values should be updated. The concatenated input also passes through tanh activation function. The tanh function creates another tensor that needs to be added to the state. The outputs of sigmoidal function and tanh function are multiplied and then added to the cell state using vector addition.

Output Gate Output gate determines what needs to be sent as output. The concatenated input of x_t and h_{t-1} is taken to a sigmoid activation function. The output gate decides what part should be the output. The cell state is sent as input for tanh function. Its value is in the range –1 and +1. Through the outputs of sigmoidal function and tanh function, parts of the output that need to be sent is decided.

Gated Recurrent Unit (GRU)

GRU is another variant of LSTM. The advantages of GRU are its ability to handle vanishing gradient problem, its ability to get quickly trained and fewer calculations.

There are two gates:

1. Reset Gate
2. Update Gate

Reset Gate Reset gate is used to merge the new input with the previous memory.

Update Gate Update gate determines the amount of previous memory that needs to be merged. It determines how much previous memory is maintained. Thus, update gate is a combination of input and forget gates.

RNN can use this LSTM cell or GRU cell as a neuron and can act as a neural network. Some of the cases where RNN can be useful are in applications such as machine translation, spoken word recognition and sentiment analysis.

Scan for information on, *'Restricted Boltzmann Machines and Deep Belief Networks, Auto Encoders, and Deep Reinforcement Learning'*

Summary

1. A neural network that has only one hidden layer is called a shallow network. If a neural network has two or more hidden layers, it is called as deep neural network.
2. Deep neural network extracts features automatically, enabling automatic machine learning.
3. Activation functions create non-linearity. ReLU and Softmax are some of the examples of activation functions.
4. Stochastic Gradient descent (SGD) computes error for every data and adjusts the we