



NN&DL

JOURNAL

POONAM BADHE

ROLL NO 04
CST 3

INDEX

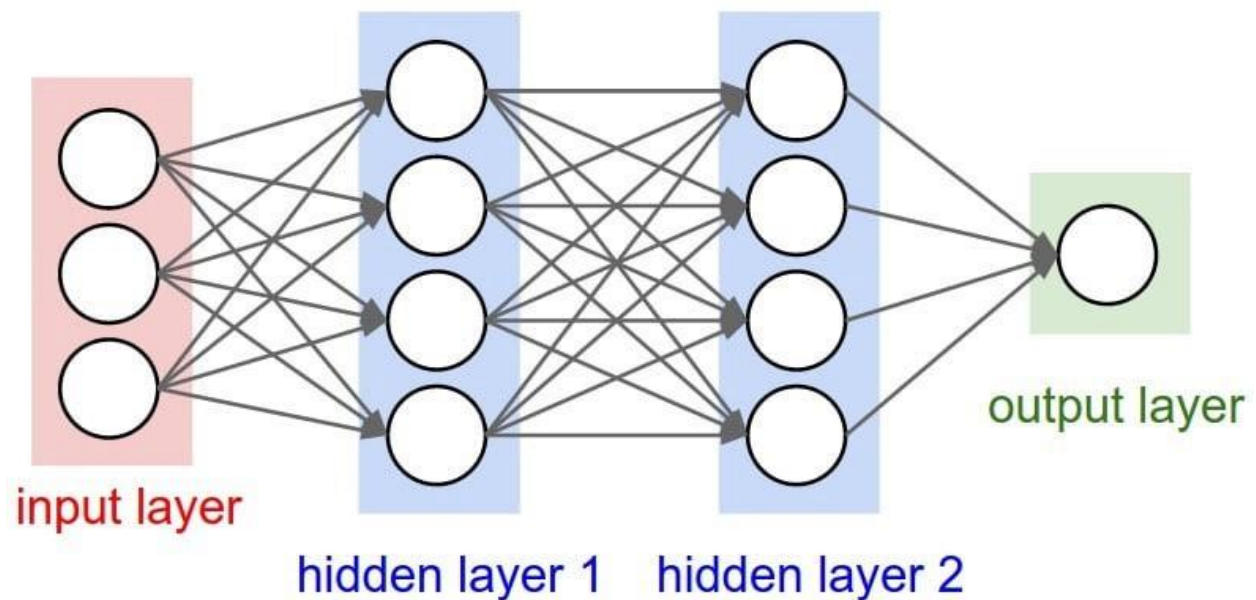
Sr No.	Contents	Pg. No.	Remarks
1	Implementation of ANN:- a) Implementation of XOR gate b) Implementation of Customer churn prediction using dataset for Telecom company. c) Implementation of Customer churn prediction using dataset for Bank.	2	
2	Implementation of CNN: a) Image classification using CIFAR-10 Dataset. b) Flower image classification c) Handwritten digits classification	6	
3	Implementation of RNN/LSTM	15	
4	Implementation of GAN	24	
5	Implementation of Autoencoder	28	

Experiment 1: Artificial Neural Networks (ANN) (Theory N Hands-on)

Aim:

To study about ANN

Theory:



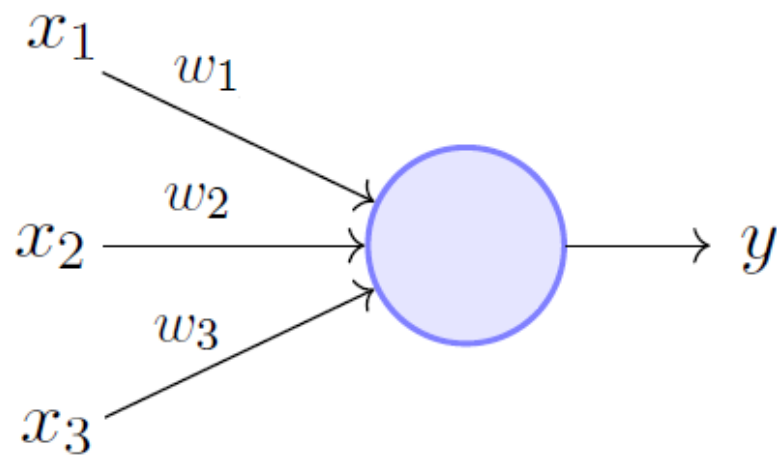
Artificial neural networks (ANN) are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems “learn” to perform tasks by considering examples, generally without being programmed with task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as “cat” or “no cat” and using the results to identify cats in other images.

Perceptron Model

A perceptron model was a form of neural network introduced in 1958 by Frank Rosenblatt.

Perceptron is a single layer neural network and a *multi-layer perceptron* is called *Neural Networks*.

Perceptron is a **linear classifier** (binary). Also, it is used in supervised learning. It helps to classify the given input data.

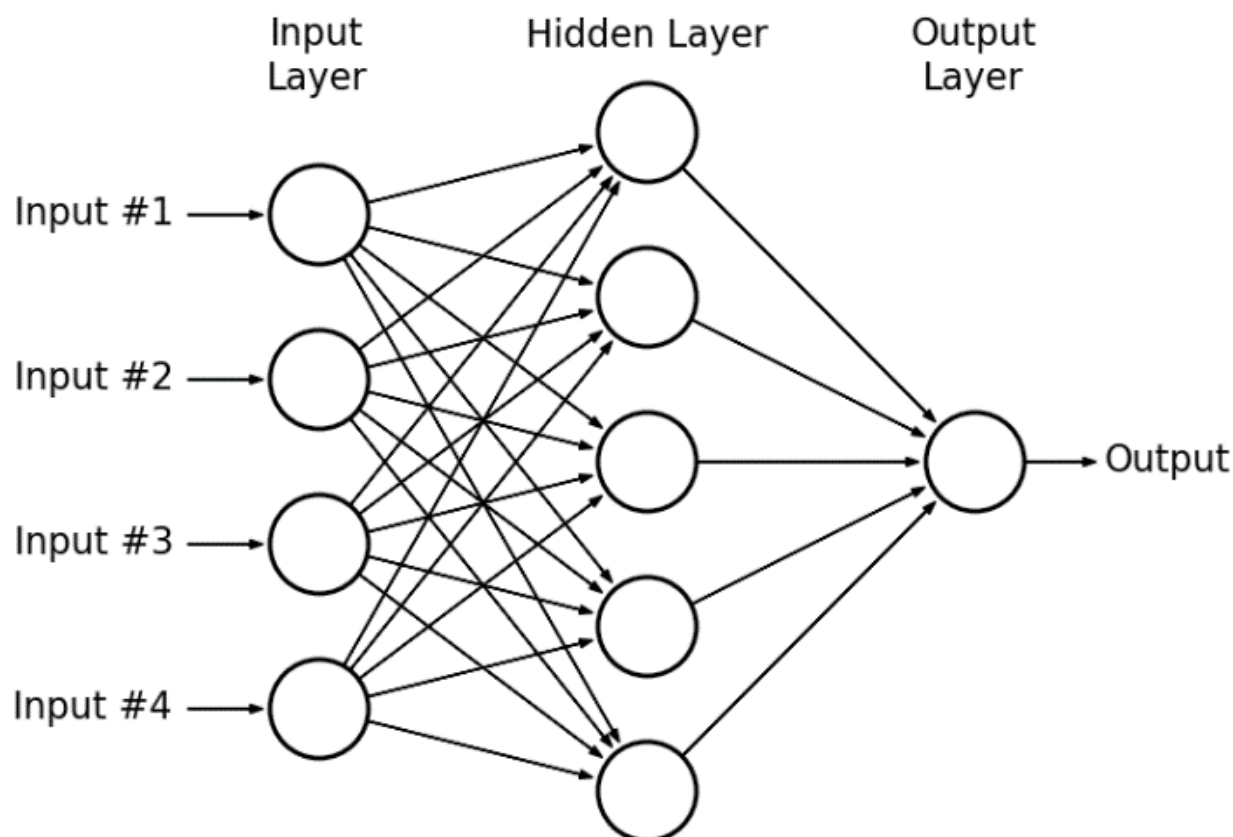


Perceptron Model (Minsky-Papert in 1969)

If $f(y)$ is just a sum, then $y = x_1 w_1 + x_2 w_2 + x_3 w_3$.

But a single perceptron won't be enough to learn complicated systems. Fortunately, we need to expand the single perceptron, to create a multi-layer perceptron model. We'll also introduce the idea of the activation function.

To build a network of perceptrons, we can connect layers of perceptrons, using a multi-layer perceptron model.



Input Layer- receives the input as data from the dataset.

Output Layer (it can be more than one also)- Give the output as a prediction from the model.

Hidden Layers- Any layer between Input and Output layer.

Activation Function

It is used to set boundaries for the overall output values of:

$$x*w+b$$

we can also state as:

$$z=x*w+b$$

And then pass z through some activation function to limit its values. Keep in mind that you will find variables $f(z)$ or X to denote a tensor input consisting of multiple values.

Sigmoid — The sigmoid function is applicable for outputs having 0 or 1.

Rectified Linear Unit (ReLU)- ReLu has been found to have very good performance, especially when dealing with the issues of vanishing gradient. It's always used as default because it has

overall good performance.

Cost Functions

The cost function is always referred to as a loss function. We can keep track of our loss/cost during training to monitor network performance. A **cost function** is a measure of “how good” a **neural network** did with respect to its given training sample and the expected output. It also may depend on variables such as weights and biases. A **cost function** is a single value, not a vector because it rates how good the **neural network** did as a whole.

BackPropogation

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

Backpropagation is a short form for “backward propagation of errors.” It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respect to all the weights in the network.

Why Do We Need Backpropagation?

Most prominent advantages of Backpropagation are:

- Backpropagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

Dropout Layer

Dropout refers to ignoring units (i.e. neurons) during the training phase of a certain set of neurons which is chosen at random. By “ignoring”, I mean these units are not considered during a particular forward or backward pass. More technically, At each training stage, individual nodes are either dropped out of the net with probability $1-p$ or kept with probability p , so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.

Conclusion

Studied about ANN and implemented a) Implementation of XOR gate
b) Implementation of Customer churn prediction using dataset for Telecom company.
c) Implementation of Customer churn prediction using dataset for Bank. in google collab.

Experiment 2: Understanding of Convolutional Neural Network (CNN) – Deep Learning

Aim:

To study about CNN and implement

- a) Image classification using CIFAR-10 Dataset.
- b) Flower image classification
- c) Handwritten digits classification

Theory:

In neural networks, Convolutional neural networks (ConvNets or CNNs) is one of the main categories to do image recognition, images classification. Object detections, recognition faces etc., are some of the areas where CNNs are widely used.

CNN image classifications take an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see $h \times w \times d$ (h = Height, w = Width, d = Dimension). Eg., An image of $6 \times 6 \times 3$ array of matrix of RGB (3 refers to RGB values) and an image of $4 \times 4 \times 1$ array of matrix of grayscale image.

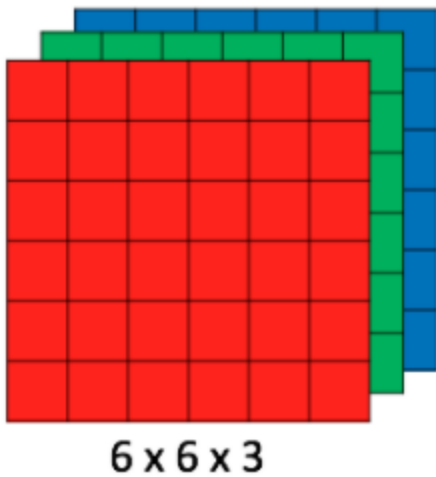


Figure 1 : Array of RGB Matrix

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

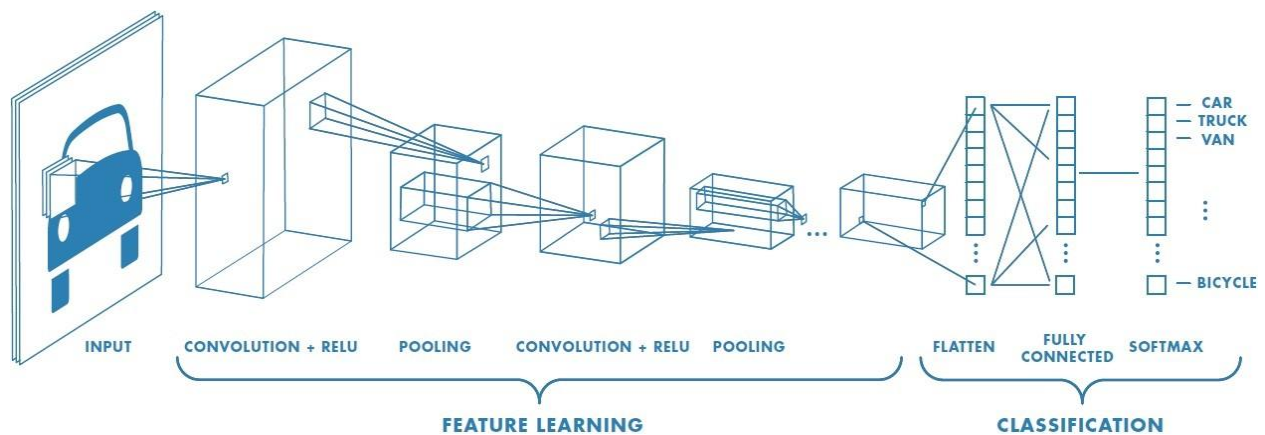


Figure 2 : Neural network with many convolutional layers

Convolution Layer

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

- An image matrix (volume) of dimension **($h \times w \times d$)**
- A filter (**$f_h \times f_w \times d$**)
- Outputs a volume dimension **($h - f_h + 1 \times w - f_w + 1 \times 1$)**



Figure 3: Image matrix multiplies kernel or filter matrix

Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

*

1	0	1
0	1	0
1	0	1

5 x 5 – Image Matrix

3 x 3 – Filter Matrix

Figure 4: Image matrix multiplies kernel or filter matrix

Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called

“**Feature Map**” as output shown in below

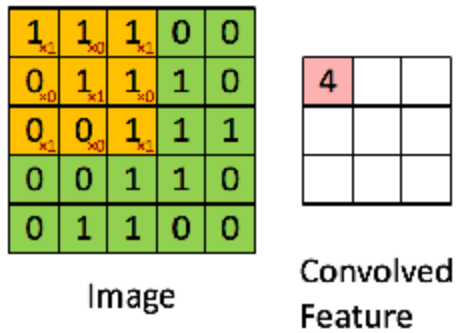


Figure 5: 3 x 3 Output matrix

Convolution of an image with different filters can perform operations such as edge detection,

blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).






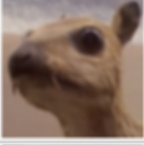

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 7 : Some common filters

Strides

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

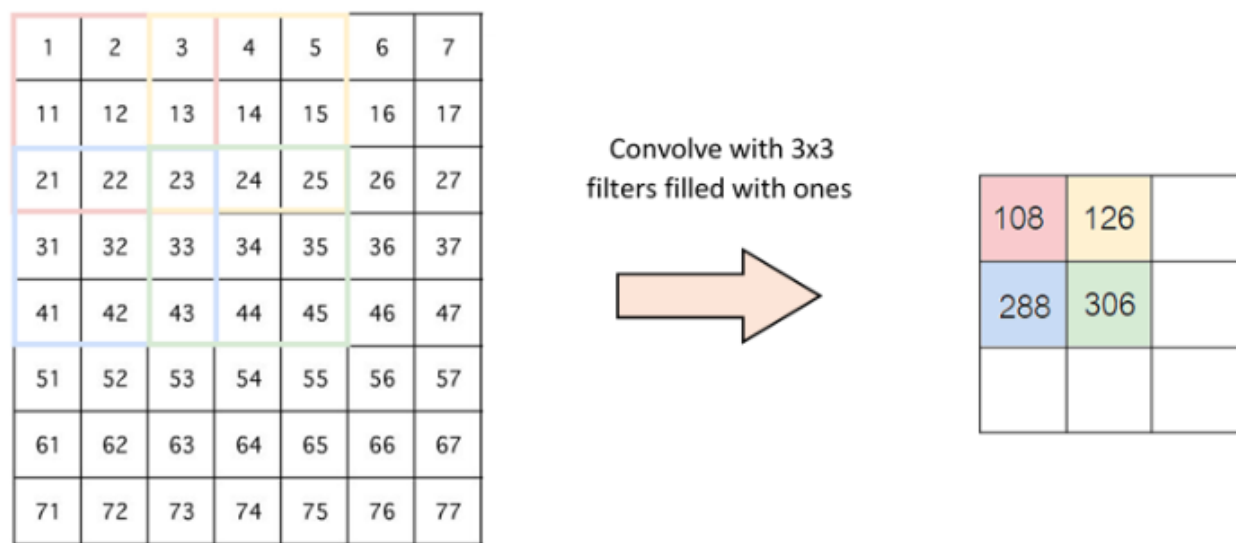


Figure 6 : Stride of 2 pixels

Padding

Sometimes filter does not fit perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = \max(0, x)$.

Why ReLU is important : ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.

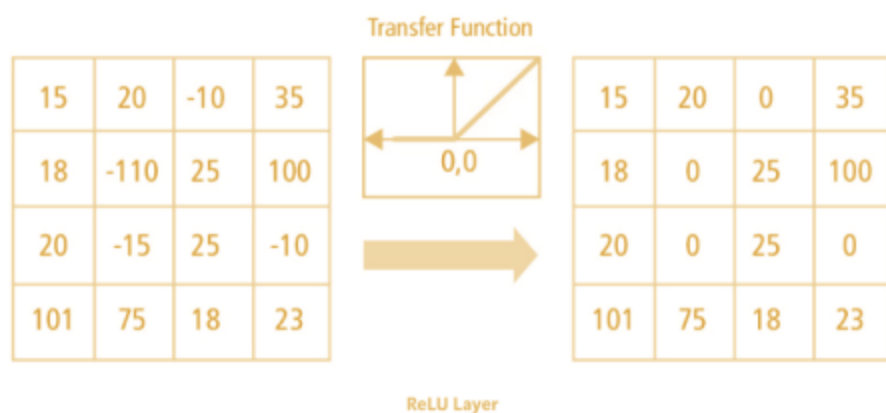


Figure 7 : ReLU operation

There are other non linear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance wise ReLU is better than the other two.

Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large.

Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

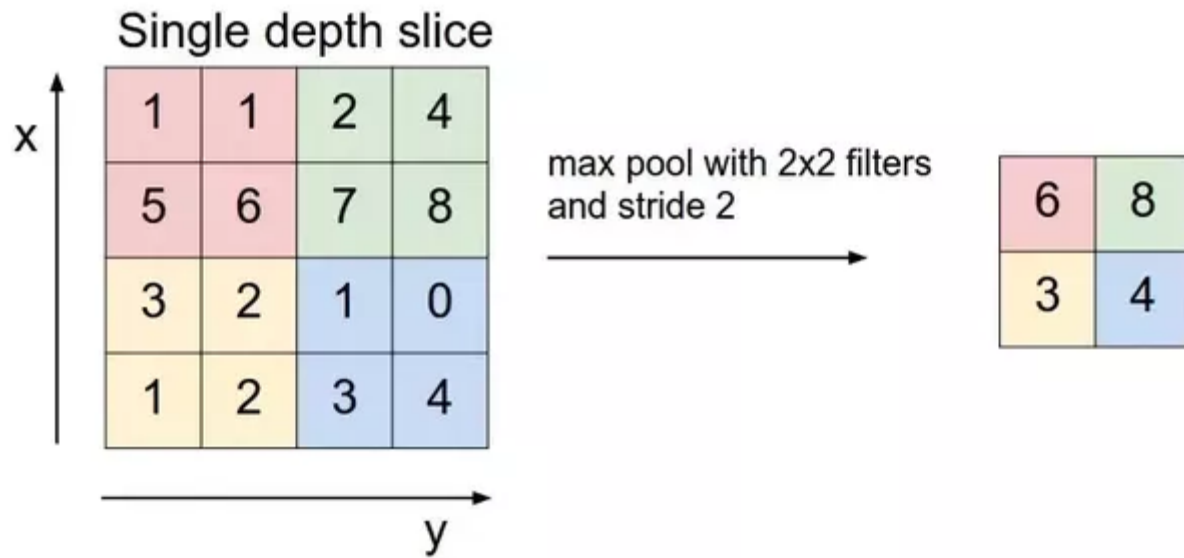


Figure 8 : Max Pooling

Fully Connected Layer

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.

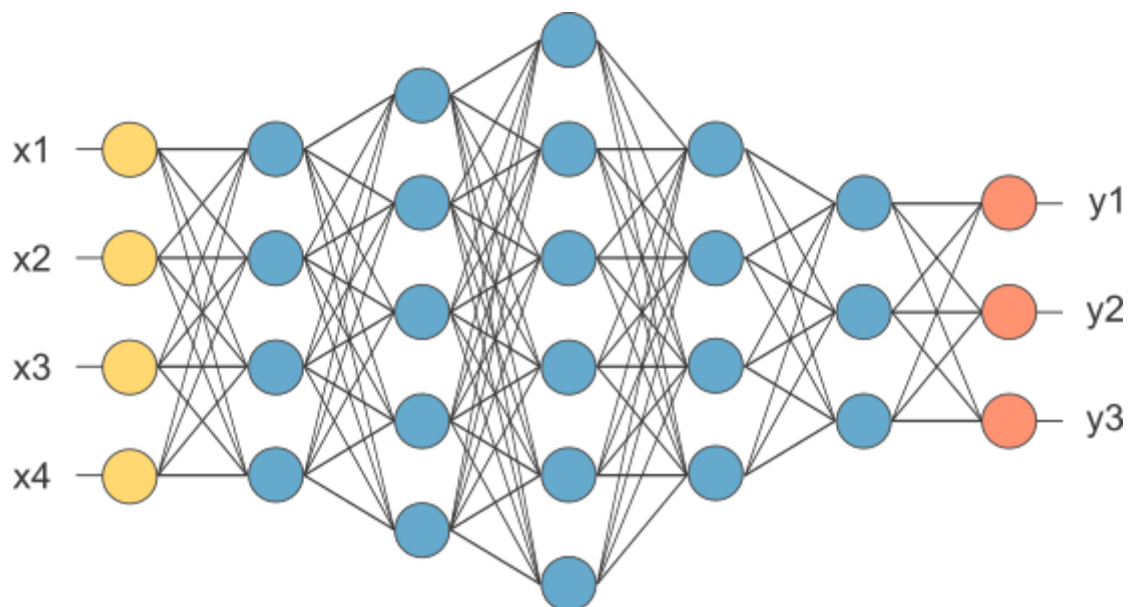


Figure 9 : After pooling layer, flattened as FC layer

In the above diagram, the feature map matrix will be converted as vector (x1, x2, x3, ...). With the fully connected layers, we combined these features together to create a model. Finally, we

have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc.,

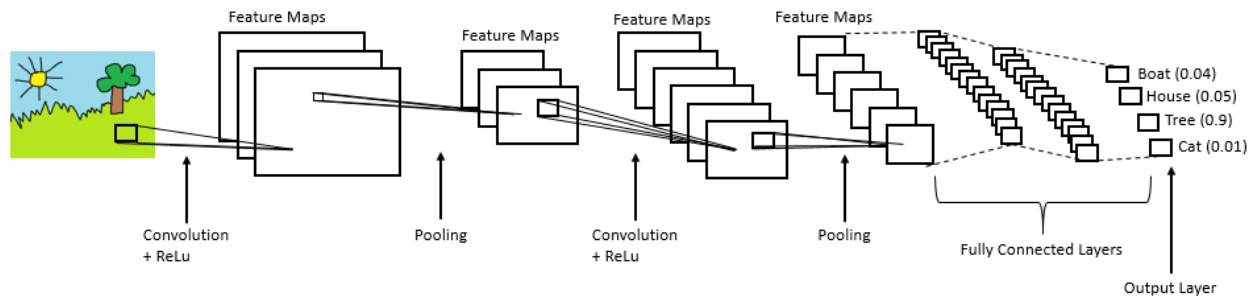


Figure 10 : Complete CNN architecture

Summary

- Provide input image into convolution layer
- Choose parameters, apply filters with strides, padding if requires. Perform convolution on the image and apply ReLU activation to the matrix.
- Perform pooling to reduce dimensionality size
- Add as many convolutional layers until satisfied
- Flatten the output and feed into a fully connected layer (FC Layer)
- Output the class using an activation function (Logistic Regression with cost functions) and classifies images.

In the next post, I would like to talk about some popular CNN architectures such as AlexNet, VGGNet, GoogLeNet, and ResNet.

Conclusion:

Implemented:

- Image classification using CIFAR-10 Dataset.
- Handwritten digits classification

Experiment 3: Introduction to Recurrent Neural Networks (RNN)

Aim:

To study about RNN

Theory:

RNNs are a powerful and robust type of neural network, and belong to the most promising algorithms in use because it is the only one with an internal memory.

Like many other deep learning algorithms, recurrent neural networks are relatively old. They were initially created in the 1980's, but only in recent years have we seen their true potential. An increase in computational power along with the massive amounts of data that we now have to work with, and the invention of long short-term memory (LSTM) in the 1990s, has really brought RNNs to the foreground.

Because of their internal memory, RNN's can remember important things about the input they received, which allows them to be very precise in predicting what's coming next. This is why they're the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more. Recurrent neural networks can form a much deeper understanding of a sequence and its context compared to other algorithms.

WHAT IS A RECURRENT NEURAL NETWORK (RNN)?

Recurrent neural networks (RNN) are a class of neural networks that are helpful in modeling sequence data. Derived from feedforward networks, RNNs exhibit similar behavior to how human brains function. Simply put: recurrent neural networks produce predictive results in sequential data that other algorithms can't.

But when do you need to use a RNN?

“Whenever there is a sequence of data and that temporal dynamics that connects the data is more important than the spatial content of each individual frame.” – Lex Fridman (MIT)

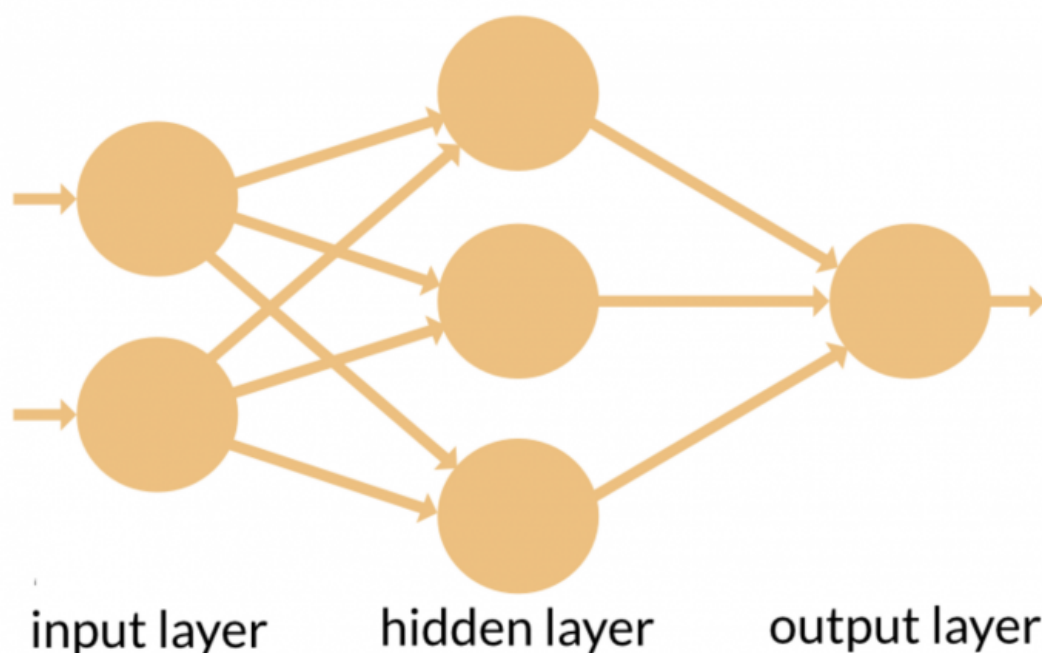
Since RNNs are being used in the software behind Siri and Google Translate, recurrent neural networks show up a lot in everyday life.

How Recurrent Neural Networks Work

To understand RNNs properly, you'll need a working knowledge of "normal" feed-forward neural networks and sequential data.

Sequential data is basically just ordered data in which related things follow each other. Examples are financial data or the DNA sequence. The most popular type of sequential data is perhaps time series data, which is just a series of data points that are listed in time order.

RNN VS. FEED-FORWARD NEURAL NETWORKS



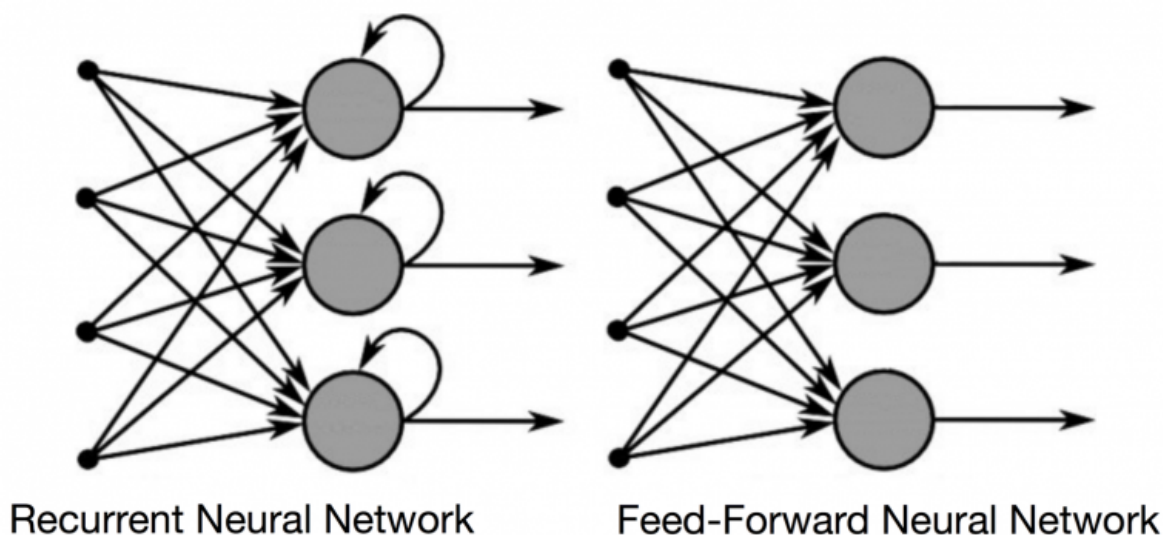
RNN's and feed-forward neural networks get their names from the way they channel information.

In a feed-forward neural network, the information only moves in one direction — from the input layer, through the hidden layers, to the output layer. The information moves straight through the network and never touches a node twice.

Feed-forward neural networks have no memory of the input they receive and are bad at predicting what's coming next. Because a feed-forward network only considers the current input, it has no notion of order in time. It simply can't remember anything about what happened in the past except its training.

In a RNN the information cycles through a loop. When it makes a decision, it considers the current input and also what it has learned from the inputs it received previously.

The two images below illustrate the difference in information flow between a RNN and a feed-forward neural network.



A usual RNN has a short-term memory. In combination with a LSTM they also have a long-term memory (more on that later).

Another good way to illustrate the concept of a recurrent neural network's memory is to explain it with an example:

Imagine you have a normal feed-forward neural network and give it the word "neuron" as an input and it processes the word character by character. By the time it reaches the character "r," it has already forgotten about "n," "e" and "u," which makes it almost impossible for this type of neural network to predict which character would come next.

A recurrent neural network, however, is able to remember those characters because of its internal memory. It produces output, copies that output and loops it back into the network.

Simply put: recurrent neural networks add the immediate past to the present.

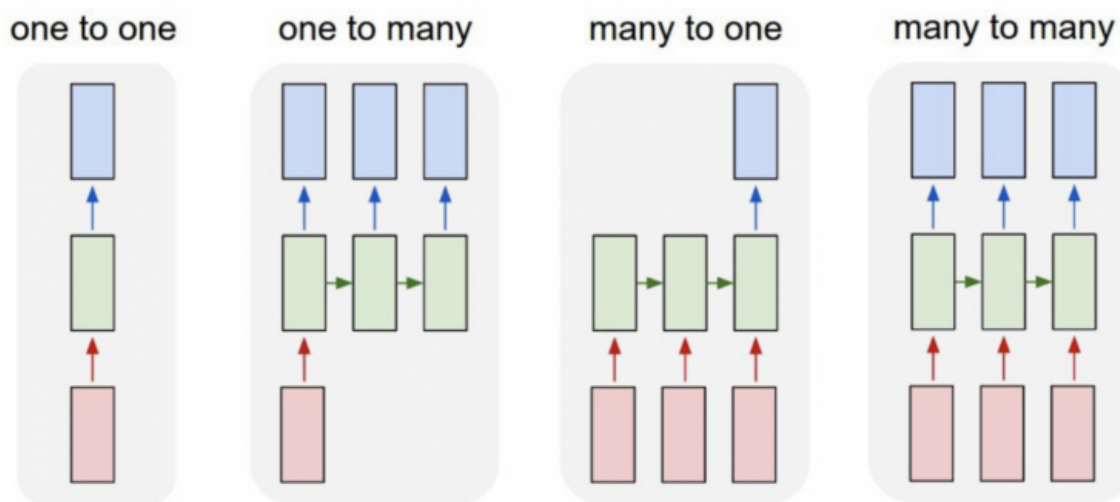
Therefore, a RNN has two inputs: the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.

A feed-forward neural network assigns, like all other deep learning algorithms, a weight matrix to its inputs and then produces the output. Note that RNNs apply weights to the current and also to the previous input. Furthermore, a recurrent neural network will also tweak the weights for both through gradient descent and backpropagation through time (BPTT).

TYPES OF RNNs

- One to One
- One to Many
- Many to One
- Many to Many

Also note that while feed-forward neural networks map one input to one output, RNNs can map one to many, many to many (translation) and many to one (classifying a voice).



Backpropagation Through Time

To understand the concept of backpropagation through time you'll need to understand the concepts of forward and backpropagation first. We could spend an entire article discussing these concepts, so I will attempt to provide as simple a definition as possible.

WHAT IS BACKPROPAGATION?

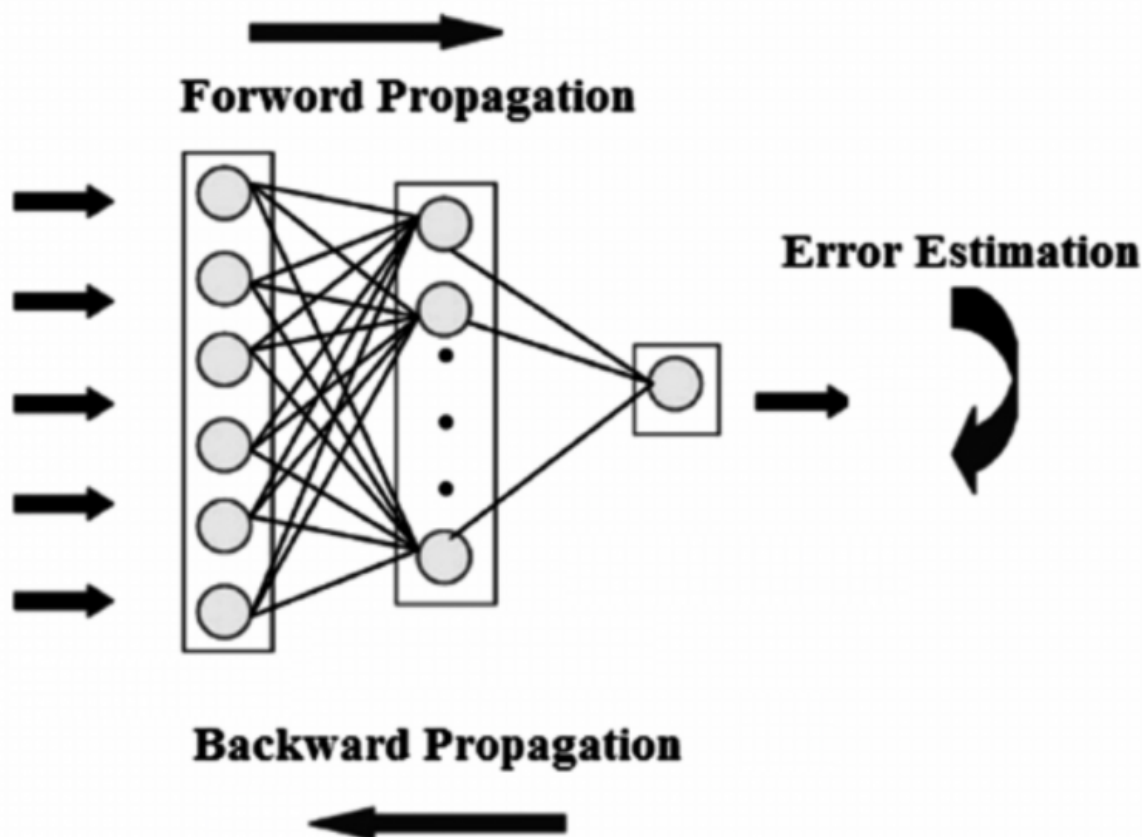
Backpropagation (BP or backprop, for short) is known as a workhorse algorithm in machine learning. Backpropagation is used for calculating the gradient of an error function with respect to a neural network's weights. The algorithm works its way backwards through the various layers of gradients to find the partial derivative of the errors with respect to the weights. Backprop then uses these weights to decrease error margins when training.

In neural networks, you basically do forward-propagation to get the output of your model and check if this output is correct or incorrect, to get the error. Backpropagation is nothing but going backwards through your neural network to find the partial derivatives of the error with respect to the weights, which enables you to subtract this value from the weights.

Those derivatives are then used by gradient descent, an algorithm that can iteratively minimize a given function. Then it adjusts the weights up or down, depending on which decreases the error. That is exactly how a neural network learns during the training process.

So, with backpropagation you basically try to tweak the weights of your model while training.

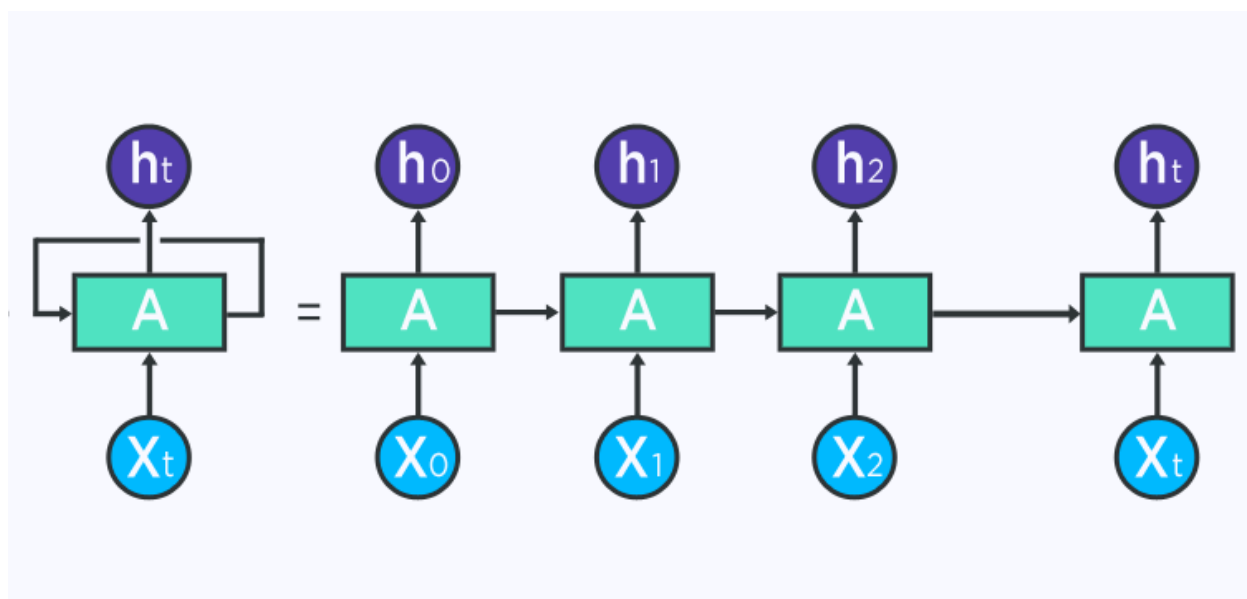
The image below illustrates the concept of forward propagation and backpropagation in a feed-forward neural network:



BPTT is basically just a fancy buzz word for doing backpropagation on an unrolled RNN. Unrolling is a visualization and conceptual tool, which helps you understand what's going on within the network. Most of the time when implementing a recurrent neural network in the common programming frameworks, backpropagation is automatically taken care of, but you need to understand how it works to troubleshoot problems that may arise during the development process.

You can view a RNN as a sequence of neural networks that you train one after another with backpropagation.

The image below illustrates an unrolled RNN. On the left, the RNN is unrolled after the equal sign. Note there is no cycle after the equal sign since the different time steps are visualized and information is passed from one time step to the next. This illustration also shows why a RNN can be seen as a sequence of neural networks.



An unrolled version of RNN

If you do BPTT, the conceptualization of unrolling is required since the error of a given timestep depends on the previous time step.

Within BPTT the error is backpropagated from the last to the first timestep, while unrolling all the timesteps. This allows calculating the error for each timestep, which allows updating the weights. Note that BPTT can be computationally expensive when you have a high number of timesteps.


Two issues of standard RNN's

There are two major obstacles RNN's have had to deal with, but to understand them, you first need to know what a gradient is.

A gradient is a partial derivative with respect to its inputs. If you don't know what that means, just think of it like this: a gradient measures how much the output of a function changes if you change the inputs a little bit.

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

EXPLODING GRADIENTS



Exploding gradients are when the algorithm, without much reason, assigns a stupidly high importance to the weights. Fortunately, this problem can be easily solved by truncating or squashing the gradients.

VANISHING GRADIENTS

Vanishing gradients occur when the values of a gradient are too small and the model stops learning or takes way too long as a result. This was a major problem in the 1990s and much harder to solve than the exploding gradients. Fortunately, it was solved through the concept of LSTM by Sepp Hochreiter and Juergen Schmidhuber.

Long Short-Term Memory (LSTM)

Long short-term memory networks (LSTMs) are an extension for recurrent neural networks, which basically extends the memory. Therefore it is well suited to learn from important experiences that have very long time lags in between.

WHAT IS LONG SHORT-TERM MEMORY (LSTM)?

Long Short-Term Memory (LSTM) networks are an extension of RNN that extend the memory. LSTM are used as the building blocks for the layers of a RNN. LSTMs assign data “weights” which helps RNNs to either let new information in, forget information or give it importance enough to impact the output.

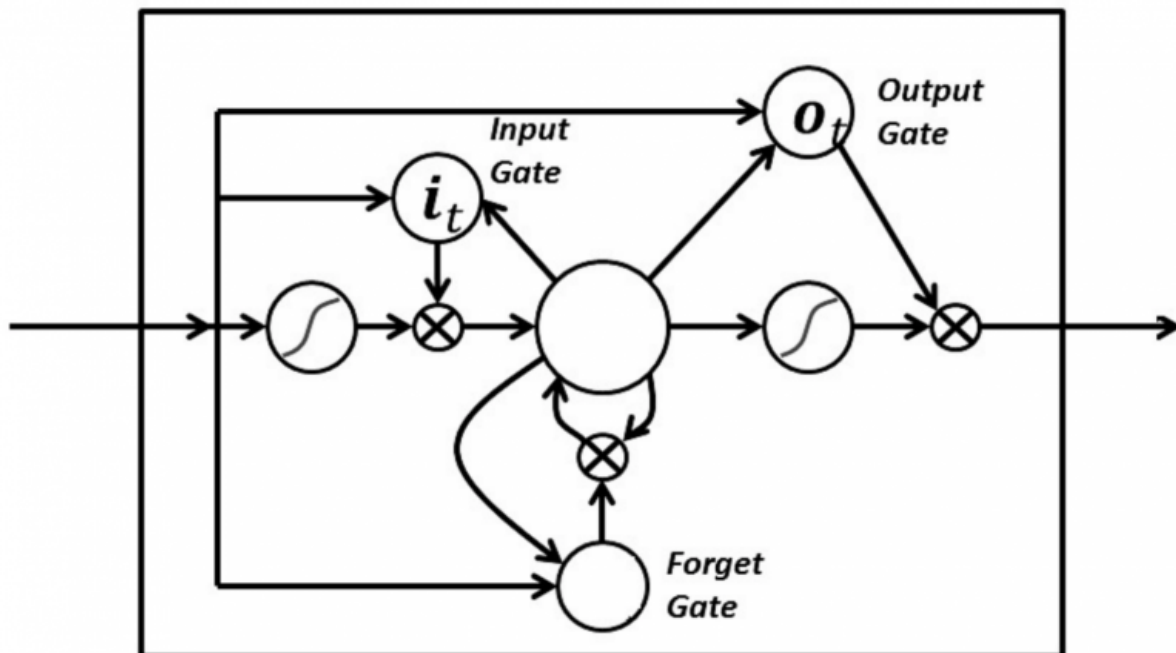
The units of an LSTM are used as building units for the layers of a RNN, often called an LSTM network.

LSTMs enable RNNs to remember inputs over a long period of time. This is because LSTMs contain information in a memory, much like the memory of a computer. The LSTM can read, write and delete information from its memory.

This memory can be seen as a gated cell, with gated meaning the cell decides whether or not to store or delete information (i.e., if it opens the gates or not), based on the importance it assigns to the information. The assigning of importance happens through weights, which are also learned by the algorithm. This simply means that it learns over time what information is important and what is not.

In an LSTM you have three gates: input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn't important (forget

gate), or let it impact the output at the current timestep (output gate). Below is an illustration of a RNN with its three gates:



The gates in an LSTM are analog in the form of sigmoids, meaning they range from zero to one. The fact that they are analog enables them to do backpropagation.

The problematic issue of vanishing gradients is solved through LSTM because it keeps the gradients steep enough, which keeps the training relatively short and the accuracy high.

Conclusion

Studied and implemented RNN in google collab

Experiment 4: Implementation of GAN

Aim:

To study about GAN

Theory:

Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning. It was developed and introduced by Ian J. Goodfellow in 2014. GANs are basically made up of a system of two competing neural network models which compete with each other and are able to analyze, capture and copy the variations within a dataset.

Why were GANs developed in the first place?

It has been noticed most of the mainstream neural nets can be easily fooled into misclassifying things by adding only a small amount of noise into the original data. Surprisingly, the model after adding noise has higher confidence in the wrong prediction than when it predicted correctly. The reason for such adversary is that most machine learning models learn from a limited amount of data, which is a huge drawback, as it is prone to overfitting. Also, the mapping between the input and the output is almost linear. Although, it may seem that the boundaries of separation between the various classes are linear, but in reality, they are composed of linearities and even a small change in a point in the feature space might lead to misclassification of data.

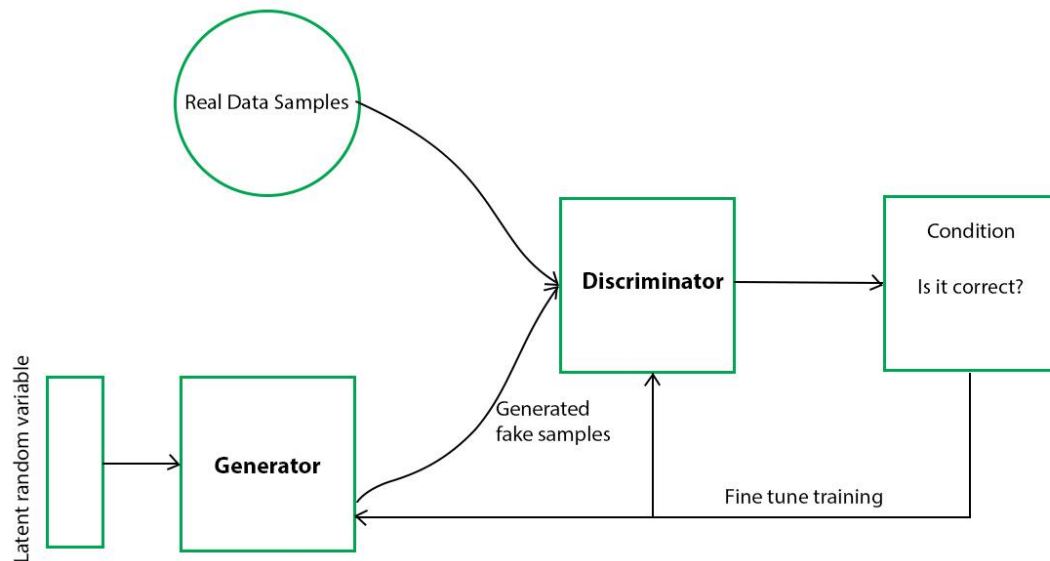
How does GANs work?

Generative Adversarial Networks (GANs) can be broken down into three parts:

- **Generative:** To learn a generative model, which describes how data is generated in terms of a probabilistic model.
- **Adversarial:** The training of a model is done in an adversarial setting.
- **Networks:** Use deep neural networks as the artificial intelligence (AI) algorithms for training purpose.

In GANs, there is a **generator** and a **discriminator**. The Generator generates fake samples of data (be it an image, audio, etc.) and tries to fool the Discriminator. The Discriminator, on the other hand, tries to distinguish between the real and fake samples. The Generator and the Discriminator are both Neural

Networks and they both run in competition with each other in the training phase. The steps are repeated several times and in this, the Generator and Discriminator get better and better in their respective jobs after each repetition. The working can be visualized by the diagram given below:



Here, the generative model captures the distribution of data and is trained in such a manner that it tries to maximize the probability of the Discriminator in making a mistake. The Discriminator, on the other hand, is based on a model that estimates the probability that the sample that it got is received from the training data and not from the Generator.

The GANs are formulated as a minimax game, where the Discriminator is trying to minimize its reward $V(\mathbf{D}, \mathbf{G})$ and the Generator is trying to minimize the Discriminator's reward or in other words, maximize its loss. It can be mathematically described by the formula below:

$$\min_G \max_D V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

where,

G = Generator

D = Discriminator

$P_{data}(x)$ = distribution of real data

$P(z)$ = distribution of generator

x = sample from $P_{data}(x)$

z = sample from $P(z)$

$D(x)$ = Discriminator network

$G(z)$ = Generator network

So, basically, training a GAN has two parts:

- **Part 1:** The Discriminator is trained while the Generator is idle. In this phase, the network is only forward propagated and no back-propagation is done. The Discriminator is trained on real data for n epochs, and see if it can correctly predict them as real. Also, in this phase, the Discriminator is also trained on the fake generated data from the Generator and see if it can correctly predict them as fake.
- **Part 2:** The Generator is trained while the Discriminator is idle. After the Discriminator is trained by the generated fake data of the Generator, we can get its predictions and use the results for training the Generator and get better from the previous state to try and fool the Discriminator. The above method is repeated for a few epochs and then manually check the fake data if it seems genuine. If it seems acceptable, then the training is stopped, otherwise, its allowed to continue for few more epochs.

Different types of GANs:

GANs are now a very active topic of research and there have been many different types of GAN implementation. Some of the important ones that are actively being used currently are described below:

1. **Vanilla GAN:** This is the simplest type GAN. Here, the Generator and the Discriminator are simple multi-layer perceptrons. In vanilla GAN, the algorithm is really simple, it tries to optimize the mathematical equation using stochastic gradient descent.
2. **Conditional GAN (CGAN):** CGAN can be described as a deep learning method in which some conditional parameters are put into place. In CGAN, an additional parameter 'y' is added to the Generator for generating the corresponding data. Labels are also put into the input to the Discriminator in order for the Discriminator to help distinguish the real data from the fake generated data.

3. **Deep Convolutional GAN (DCGAN):** DCGAN is one of the most popular also the most successful implementation of GAN. It is composed of ConvNets in place of multi-layer perceptrons. The ConvNets are implemented without max pooling, which is in fact replaced by convolutional stride. Also, the layers are not fully connected.
4. **Laplacian Pyramid GAN (LAPGAN):** The Laplacian pyramid is a linear invertible image representation consisting of a set of band-pass images, spaced an octave apart, plus a low-frequency residual. This approach uses multiple numbers of Generator and Discriminator networks and different levels of the Laplacian Pyramid. This approach is mainly used because it produces very high-quality images. The image is down-sampled at first at each layer of the pyramid and then it is again up-scaled at each layer in a backward pass where the image acquires some noise from the Conditional GAN at these layers until it reaches its original size.
5. **Super Resolution GAN (SRGAN):** SRGAN as the name suggests is a way of designing a GAN in which a deep neural network is used along with an adversarial network in order to produce higher resolution images. This type of GAN is particularly useful in optimally up-scaling native low-resolution images to enhance its details, minimizing errors while doing so.

Conclusion

Studied and implemented GAN in google collab

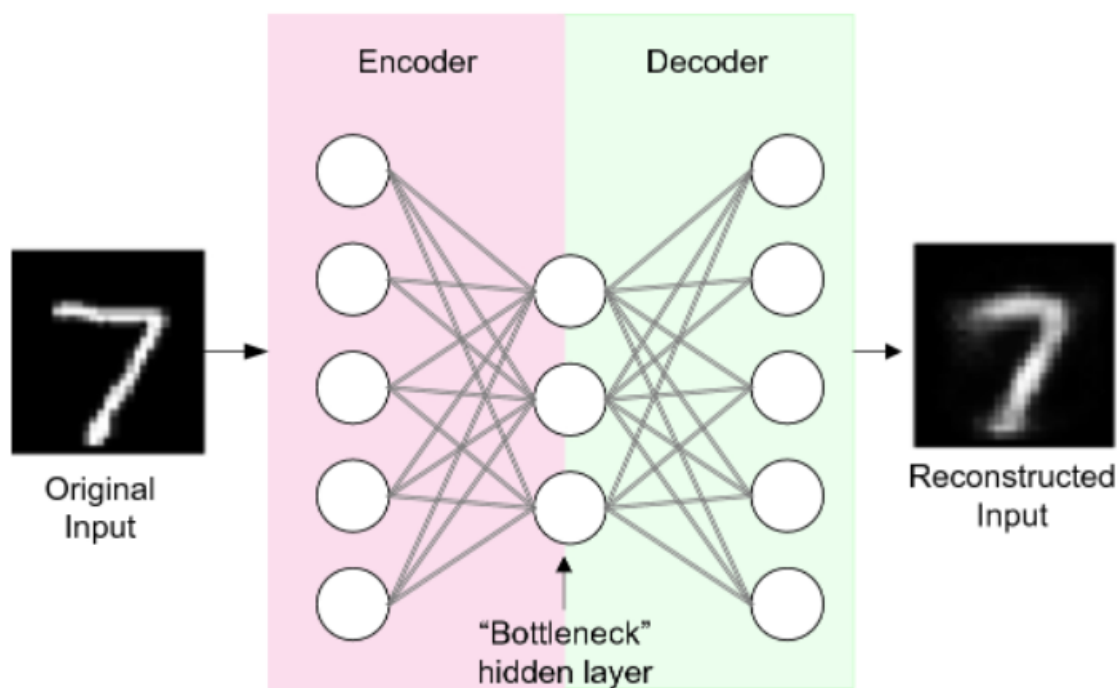
Experiment 5: Implementation of Autoencoder

Aim:

To study about Autoencoder

Theory:

Auto-encoder is a complex mathematical model which trains on unlabeled as well as unclassified data and is used to map the input data to another compressed feature representation and from that feature representation reconstructing back the input data.



Where Auto-Encoders are used ?

Autoencoders can be used to remove noise, perform image colourisation and various other purposes like -

1. **Dimensionality Reduction** : Dimension Reduction refers to the process of converting a set of data having vast dimensions into data with lesser dimensions ensuring that it conveys similar information concisely.
2. **Image -Denoising** : A noisy image can be given as input to the autoencoder and a denoised image can be provided as output. The autoencoder will try **de-noise the image by learning the latent features of the image and using that to reconstruct an image without noise**. The reconstruction error can be calculated as a measure of distance between the pixel values of the output image and ground truth image.
3. **Feature Extraction** : Once the model is fit on a training dataset, the **reconstruction (decoding) aspect of the model can be discarded** and the model up to the point of the bottleneck can be used (only the **encoding part is required**). The output of the model at the bottleneck is a fixed-length vector that provides a compressed representation of the input data.
4. **Data Compression** : It is a process to reduce the number of bits needed to represent data. Compressing data can save storage capacity, speed up file transfer, and decrease costs for storage hardware and network bandwidth. Auto-encoders are able to generate reduced representation of input data.
5. **Removing Watermarks from Images**

Drawbacks of Auto-Encoders :

1. An autoencoder learns to capture as much information as possible rather than as much *relevant* information as possible.
2. To train an autoencoder there is a need for lots of data, processing time, hyperparameter tuning, and model validation before even starting building the real model.
3. Trained with “back-propagation technique” using loss-metric, there are chances of crucial information loss during reconstruction of input.

Conclusion

Studied and implemented Autoencoders in google collab.

