Unit-V

Introduction to Deep Learning: Multilayer perceptron. Backpropagation. Loss functions. Hyperparameter tuning, Overview of RNN, CNN and LSTM.

Overview of Data Science Models: Applications to text, images, videos, recommender systems, image classification, Social network graphs.

***

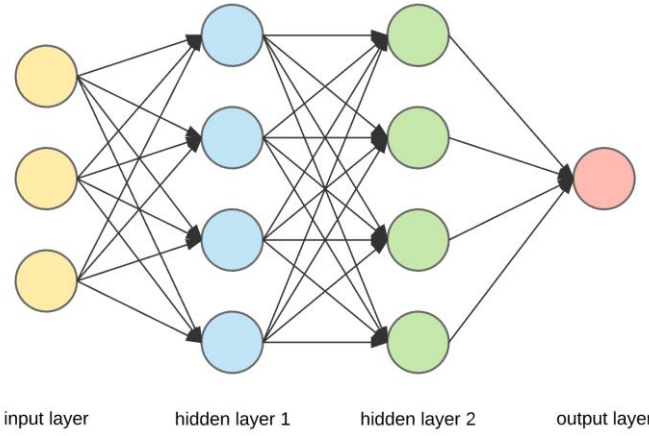
Multilayer perceptron

A multi-layered perceptron (MLP) is one of the most common neural network models used in the field of deep learning. Often referred to as a "vanilla" neural network, an MLP is simpler than the complex models of today's era. However, the techniques it introduced have paved the way for further advanced neural networks.

The multilayer perceptron (MLP) is used for a variety of tasks, such as stock analysis, image identification, spam detection, and election voting predictions.

The Basic Structure

A multi-layered perceptron consists of interconnected neurons transferring information to each other, much like the human brain. Each neuron is assigned a value. The network can be divided into three main layers.



Input Layer

This is the initial layer of the network which takes in an input which will be used to produce an output.

Hidden Layer(s)

The network needs to have at least one hidden layer. The hidden layer(s) perform computations and operations on the input data to produce something meaningful.

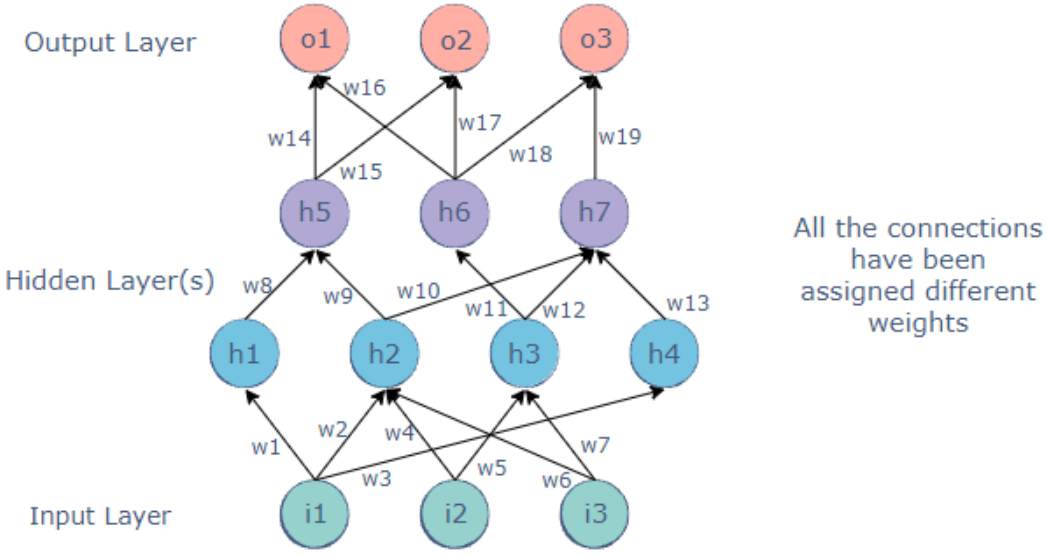
Output Layer

The neurons in this layer display a meaningful output.

Connections

The MLP is a feedforward neural network, which means that the data is transmitted from the input layer to the output layer in the forward direction.

The connections between the layers are assigned weights. The weight of a connection specifies its importance. This concept is the backbone of an MLP's learning process.



While the inputs take their values from the surroundings, the values of all the other neurons are calculated through a mathematical function involving the weights and values of the layer before it.

For example, the value of the h5 node could be:

h5 = h1.w8 + h2.w9h5=h1.w8+h2.w9

Backpropagation

Backpropagation is a technique used to optimize the weights of an MLP using the outputs as inputs.

In a conventional MLP, random weights are assigned to all the connections. These random weights propagate values through the network to produce the actual output. Naturally, this output would differ from the expected output. The difference between the two values is called the error.

Backpropagation refers to the process of sending this error back through the network, readjusting the weights automatically so that eventually, the error between the actual and expected output is minimized.

In this way, the output of the current iteration becomes the input and affects the next output. This is repeated until the correct output is produced. The weights at the end of the process would be the ones on which the neural network works correctly.

***

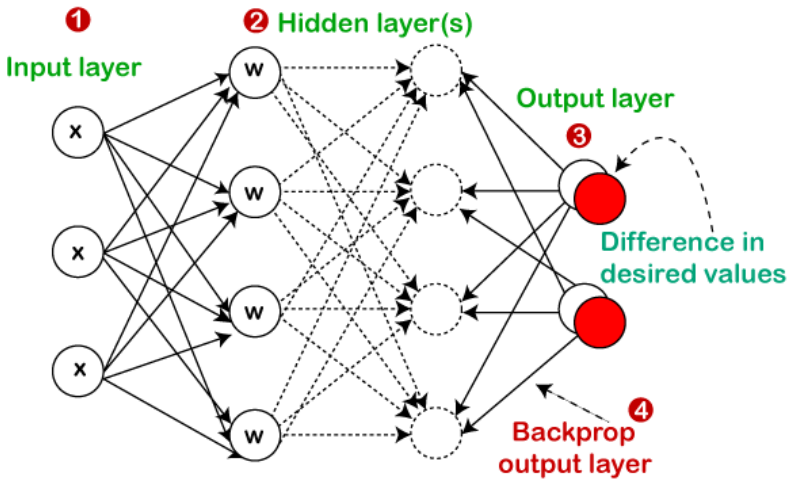
Backpropagation

The backpropagation consists of an input layer of neurons, an output layer, and at least one hidden layer. The neurons perform a weighted sum upon the input layer, which is then used by the activation function as an input, especially by the sigmoid activation function. It also makes use of supervised learning to teach the network. It constantly updates the weights of the network until the desired output is met by the network. It includes the following factors that are responsible for the training and performance of the network:

- Random (initial) values of weights.
- A number of training cycles.
- A number of hidden neurons.
- The training set.
- Teaching parameter values such as learning rate and momentum.

Working of Backpropagation

Consider the diagram given below.



1. The preconnected paths transfer the inputs X.
2. Then the weights W are randomly selected, which are used to model the input.
3. After then, the output is calculated for every individual neuron that passes from the input layer to the hidden layer and then to the output layer.
4. Lastly, the errors are evaluated in the outputs. ErrorB= Actual Output - Desired Output
5. The errors are sent back to the hidden layer from the output layer for adjusting the weights to lessen the error.
6. Until the desired result is achieved, keep iterating all of the processes.

Need of Backpropagation

- Since it is fast as well as simple, it is very easy to implement.
- Apart from no of inputs, it does not encompass of any other parameter to perform tuning.
- As it does not necessitate any kind of prior knowledge, so it tends out to be more flexible.
- It is a standard method that results well.

***

Deep Learning:

Deep learning is based on the branch of machine learning, which is a subset of artificial intelligence. Since neural networks imitate the human brain and so deep learning will do. In deep learning, nothing is programmed explicitly. Basically, it is a machine learning class that makes use of numerous nonlinear processing units so as to perform feature extraction as well as transformation. The output from each preceding layer is taken as input by each one of the successive layers.

Deep learning models are capable enough to focus on the accurate features themselves by requiring a little guidance from the programmer and are very helpful in solving out the problem of dimensionality. Deep learning algorithms are used, especially when we have a huge no of inputs and outputs.

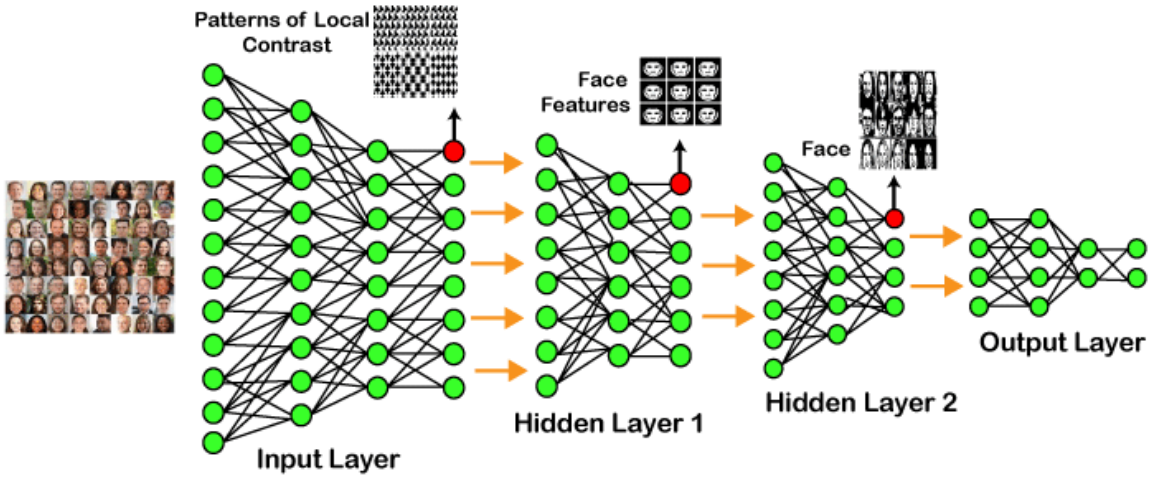
Since deep learning has been evolved by the machine learning, which itself is a subset of artificial intelligence and as the idea behind the artificial intelligence is to mimic the human behavior, so same is "the idea of deep learning to build such algorithm that can mimic the brain".

Deep learning is implemented with the help of Neural Networks, and the idea behind the motivation of Neural Network is the biological neurons, which is nothing but a brain cell.

Deep learning is a collection of statistical techniques of machine learning for learning feature hierarchies that are actually based on artificial neural networks.

So basically, deep learning is implemented by the help of deep networks, which are nothing but neural networks with multiple hidden layers.

Example of Deep Learning



In the example given above, we provide the raw data of images to the first layer of the input layer. After then, these input layer will determine the patterns of local contrast that means it will differentiate on the basis of colors, luminosity, etc. Then the 1st hidden layer will determine the face feature, i.e., it will fixate on eyes, nose, and lips, etc. And then, it will fixate those face features on the correct face template. So, in the 2nd hidden layer, it will actually determine the correct face here as it can be seen in the above image, after which it will be sent to the output layer. Likewise, more hidden layers can be added to solve more complex problems, for example, if you want to find out a particular kind of face having large or light complexions. So, as and when the hidden layers increase, we are able to solve complex problems.

Types of Deep Learning Networks:

- Convolutional Neural Network (CNN)
- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)

- Stacked Auto-Encoders.
- Deep Boltzmann Machine (DBM)
- Deep Belief Networks (DBN)

Deep learning applications

- Self-Driving Cars
- Voice Controlled Assistance
- Automatic Image Caption Generation
- Automatic Machine Translation

Limitations

- It only learns through the observations.
- It comprises of biases issues.

Advantages

- It lessens the need for feature engineering.
- It eradicates all those costs that are needless.
- It easily identifies difficult defects.
- It results in the best-in-class performance on problems.

Disadvantages

- It requires an ample amount of data.
- It is quite expensive to train.
- It does not have strong theoretical groundwork.

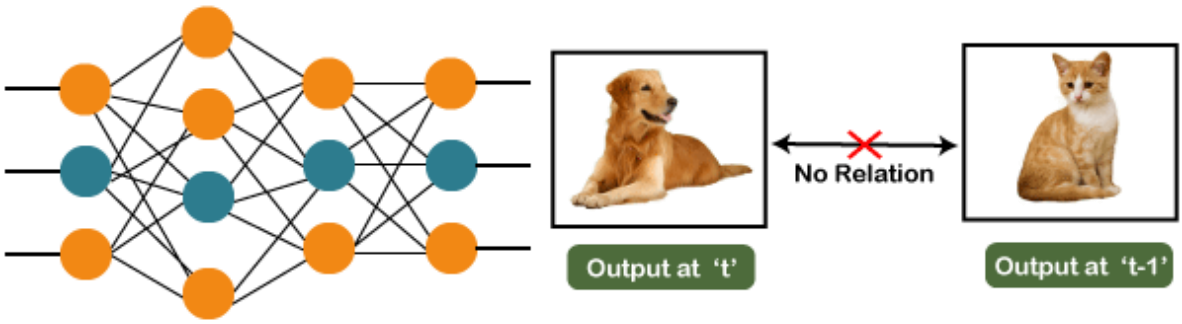
***

Recurrent Neural Networks

Why not Feedforward Networks?

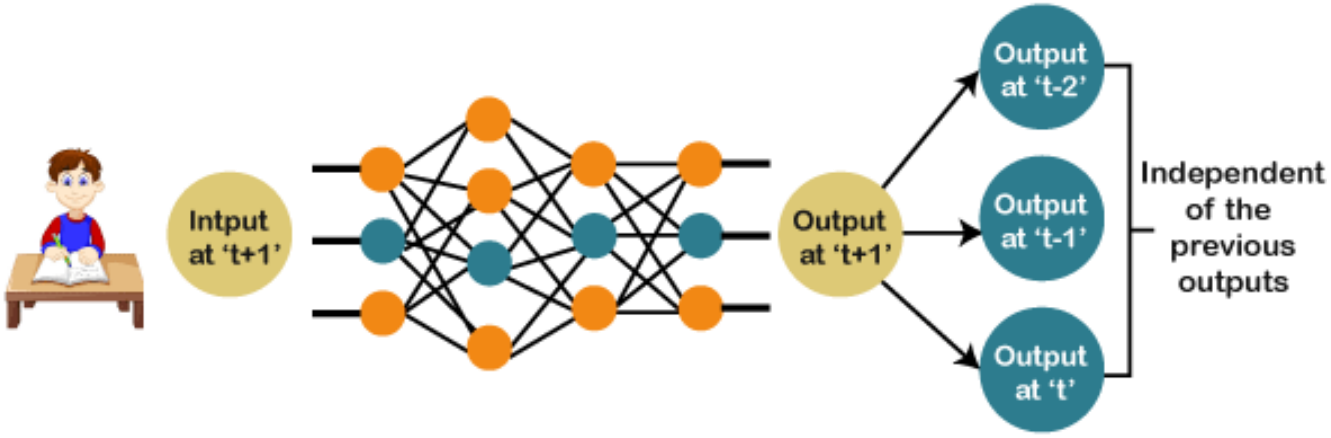
Feedforward networks are used to classify images. Let us understand the concept of a feedforward network with an example given below in which we trained our network for classifying various images of animals. If we feed an image of a cat, it will identify that image and provide a relevant label to that particular image. Similarly, if you feed an image of a dog, it will provide a relevant label to that image a particular image as well.

Consider the following diagram:



And if you notice the new output that we have got is classifying, a dog has no relation to the previous output that is of a cat, or you can say that the output at the time 't' is independent of output at a time 't-1'. It can be clearly seen that there is no relation between the new output and the previous output. So, we can say that in feedforward networks, the outputs are independent of each other.

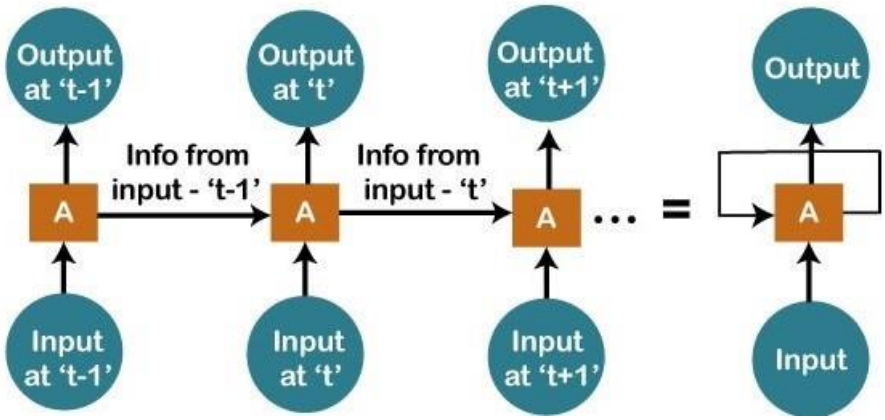
There are a few scenarios where we will actually need the previous output to get the new output. Let us discuss one such scenario where we will necessitate using the output that has been previously obtained.



Now, what happens when you read a book. You will understand that book only on the understanding of the previous words. So, if we use a feedforward network and try to predict the next word in the sentence, then in such a case, we will not be able to do that because our output will actually depend on previous outputs. But in the feedforward network, the new output is independent of the previous outputs, i.e., output at 't+1' has no relation with the output at 't-2', 't-1', and 't.' Therefore, it can be concluded that we cannot use feedforward networks for predicting the next word in the sentence. Similarly, many other examples can also be taken where we need the previous output or some information from the previous output, so as to infer the new output.

How to overcome this challenge?

Consider the following diagram:

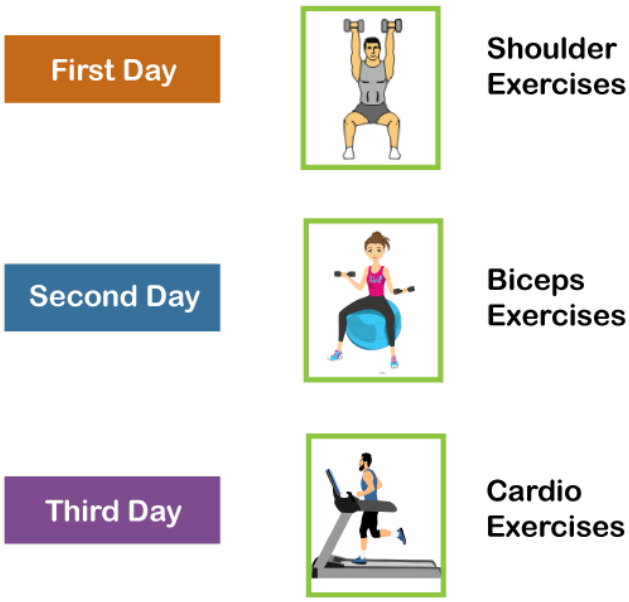


We have input at 't-1', which we will feed to the network, and then we will get the output at 't-1'. Then at the next timestamp that is at a time 't', we have an input at a time 't', which will be again given to the network along with the information from the previous timestamp, i.e., 't-1' and that will further help us to get the output at 't'. Similarly, at the output for 't+1', we have two inputs; one is the new input that we give, and the other is the information coming from the previous timestamps, i.e., 't' in order to get the output at a time 't+1'. In the same way, it will go on further like this. Here we have embodied in a more generalized way to represent it. There is a loop where the information from the previous timestamp is flowing, and this is how we can solve a particular challenge.

What are Recurrent Neural Networks?

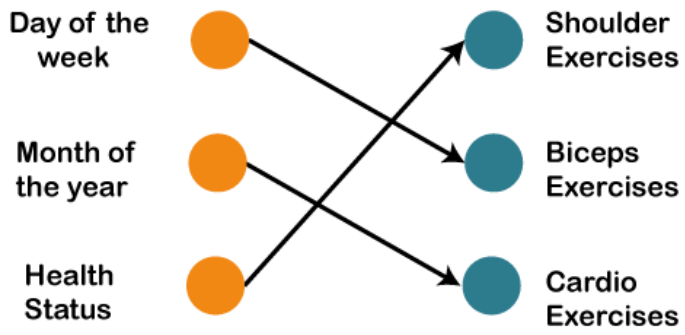
"Recurrent Networks are one such kind of artificial neural network that are mainly intended to identify patterns in data sequences, such as text, genomes, handwriting, the spoken word, numerical times series data emanating from sensors, stock markets, and government agencies".

In order to understand the concept of Recurrent Neural Networks, let's consider the following analogy.

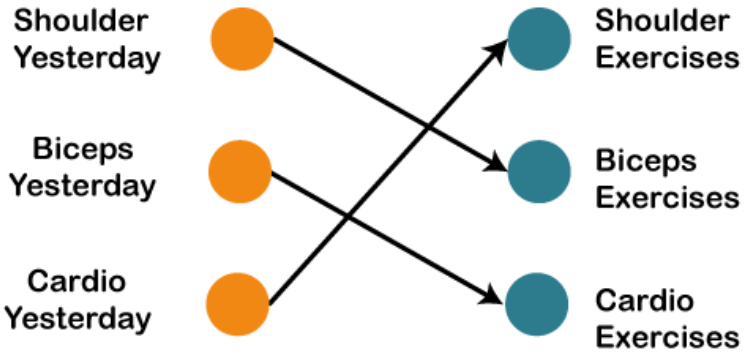


Suppose that your gym trainer has made a schedule for you. The exercises are repeated every third day. The above image includes the order of your exercises; on your very first day, you will be doing shoulders, the second day you will be doing biceps, the third day you will be doing cardio, and all these exercises are repeated in proper order.

Let's see what happens if we use a feedforward network for predicting the exercises today.

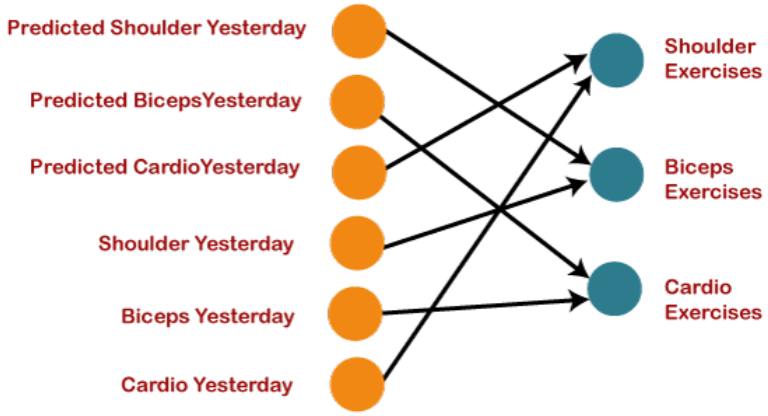


We have provided in the input such as day of the week, the month of the year, and health status. Also, we need to train our model or the network on the basis of the exercises that we have done in the past. After that, there will be a complex voting procedure involved, which will predict the exercises for us, and that procedure won't be that accurate. In that case, whatever output we will get would be as accurate as we want it to be. Now, what if the inputs get changed, and we make the inputs as the exercises that we have done the previous day.



Therefore, if shoulders were done yesterday, then definitely today will be biceps day. Similarly, if biceps were done yesterday, then today will be the cardio day, and if yesterday was the cardio day, then today, we will need to undergo shoulder.

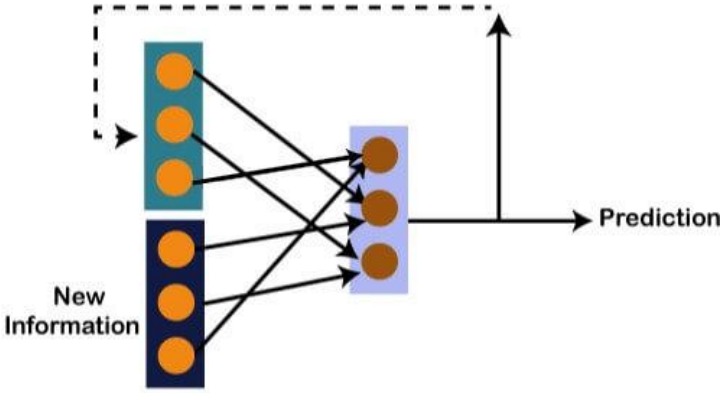
Now there can be one such scenario, where you are unable to go to the gym for one day due to some personal reasons, then it that case, we will go one timestamp back and will feed in what exercise happened day before yesterday as shown below.



So, if the exercise that happened the day before yesterday was the shoulder, then yesterday there were biceps exercises. Similarly, if biceps happened the day before yesterday, then yesterday would have been cardio exercises, and if cardio would have happened the day before yesterday, then yesterday would have been shoulder exercises. And this prediction for the exercises that happened yesterday will be fed back to our network so that these predictions can be used as inputs in order to predict what exercise will happen

today. Similarly, if you have missed your gym say for two days, three days or even one week, you will actually need to roll back, which means that you will need to go to the last day when you went to the gym, you need to figure out what exercises you did on that day and then only you will be getting the relevant output as to what exercises will happen today.

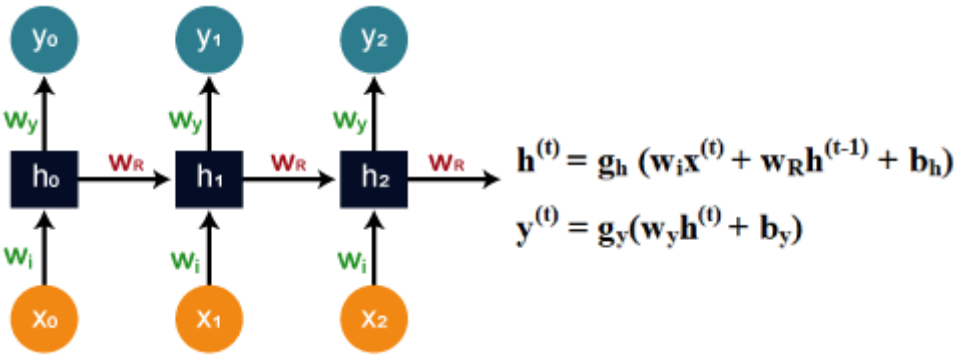
Next, we will convert all these things into a vector, which is nothing but a list of numbers.



So, there is new information along with the information which we got from the prediction at the previous timestamp because we need all of these in order to get the prediction at a time 't'. Imagine that you did shoulder exercises yesterday, then, in that case, the prediction will be biceps exercise because if the shoulder was done yesterday, then today it will definitely be biceps and output will be 0, 1, and 0, which is actually the work of our vectors.

Let's understand the math behind the Recurrent Neural Network

by simply having a look at the image given below.



Assume that 'w' is the weight matrix, and 'b' is the bias. Consider at time t=0, our input is 'xo', and we need to figure out what exactly is the 'ho'. We will substitute t=0 in the equation, as shown in the image, so as to procure the function ht value.

After that, we will find out the value of 'yo' by using values that were previously calculated when we applied it to the new formula.

The same process is repeated again and again through all the timestamps within the model so as to train it. So, this how a Recurrent Neural Networks works.

***

Long Short-Term Memory Networks (LSTMs)

Long Short-Term Memory networks, which are commonly known as "LSTMs," are a special kind of Recurrent Neural Networks that are capable enough of learning long-term dependencies.

What are long-term dependencies?

It has happened many times that we only require recent data in order to perform questions in a model. But at the same time, we may also need data that has been previously obtained.

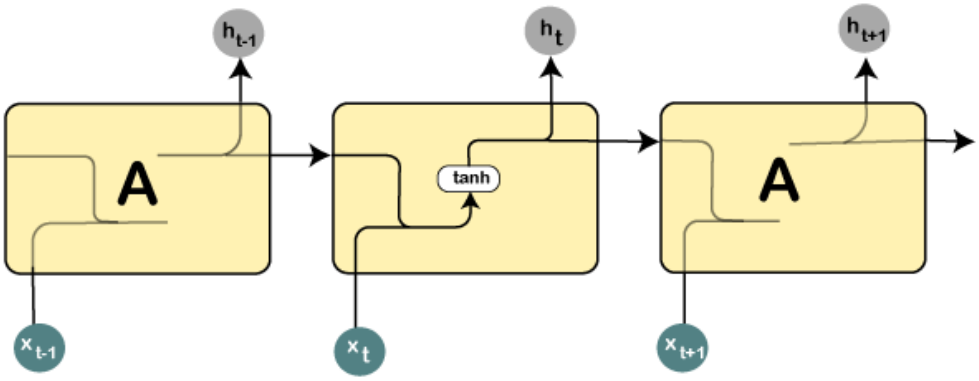
Consider the following **example** to have a better understanding of it.

Let's suppose there is a language model, which is trying to predict the next word on the basis of the previous ones. Assume that we are trying to predict the last word in the sentence say, "The car runs on the road".

Here the context is quite simple because the last word always ends up being a road. By incorporating Recurrent Neural Networks, the gap present between the former information and the existing necessities can be easily associated.

That is the reason why Vanishing and Exploding Gradient problems do not exist, followed by making this LSTM networks to easily handle long-term dependencies.

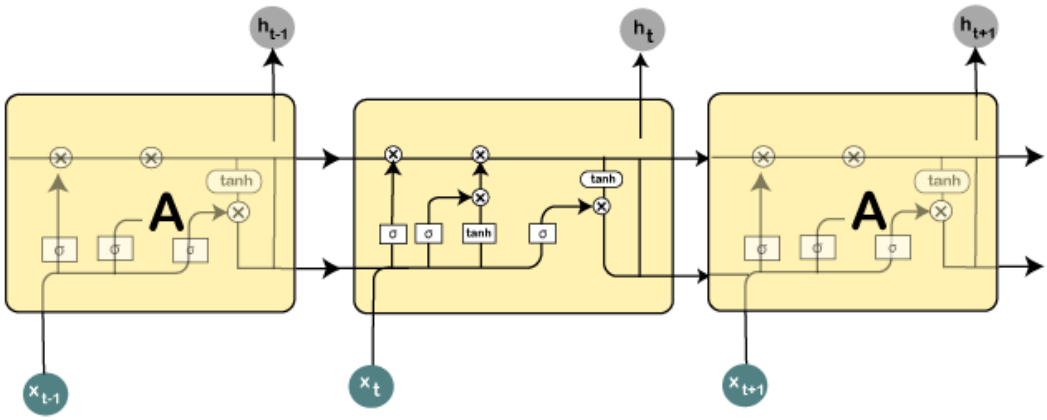
LSTM encompasses a layer of neural network in the form of a chain. In a standard recurrent neural network, the repeating module consists of one single function as shown in the image given below:



From the image given above, it can be seen that there is a tanh function in the layer, which is called as squashing function. So, what is a squashing function?

The squashing function is mainly used in between the range of -1 to +1 so that the values can be manipulated on the basis of inputs.

Now, let us consider the structure of an LSTM network:

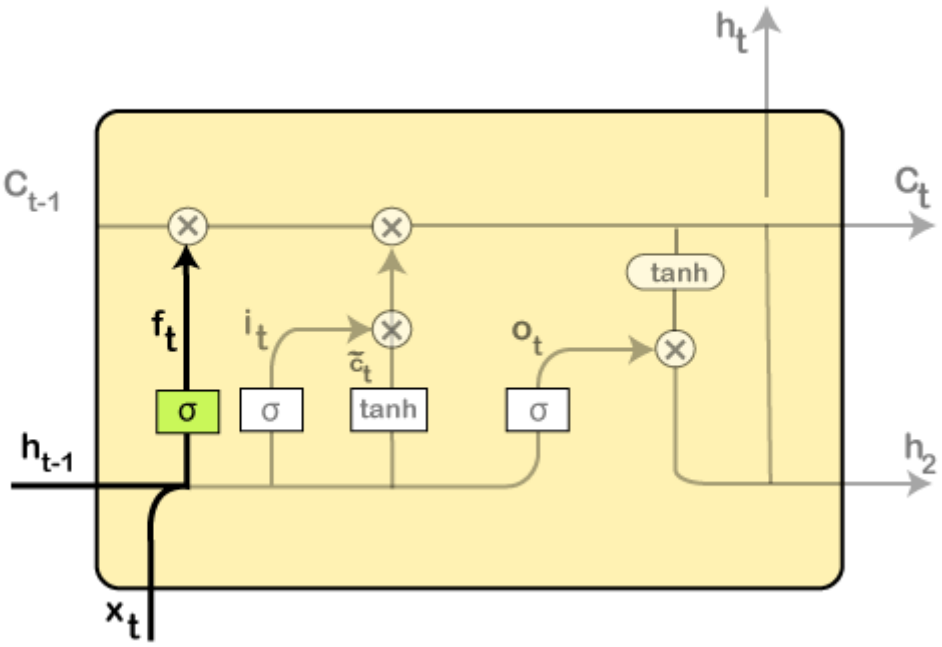


Here all those functions that are present in the layers have their own structures as and when it comes to LSTM networks. The cell state is represented by the horizontal line, acts as a conveyer belt that carries the data linearly crossways the data channel.

Let us consider a step-by-step approach to understand LSTM networks better.

Step 1:

The first step in the LSTM is to identify that information which is not required and will be thrown away from the cell state. This decision is made by a sigmoid layer, which called the forget gate layer.



The highlighted layer in the above is the sigmoid layer which is previously mentioned.

The calculation is done by considering the new input, and the previous timestamp is, which eventually leads to the output of a number between 0 and 1 for each number in that cell state.

As a typical binary, 1 represents to keep the cell state while 0 represents to trash it.

ft = σ(wf [ht-1, xt] + bf)

where, wf = Weight

ht-1 = Output from previous timestamp

xt = New input

bf = Bias

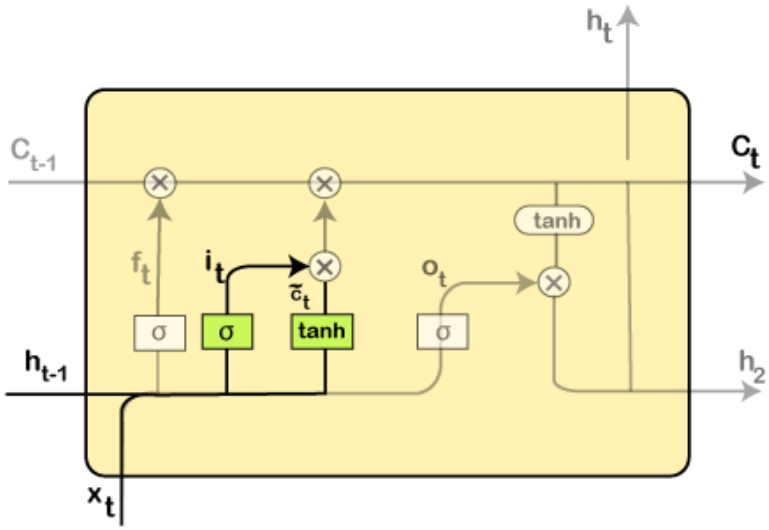
Considering a gender classification problem, it necessitates observing the correct gender when we are using the network.

Step 2:

Next, we will decide which information we will store in the cell state. It further consists of the following steps:

The sigmoid layer, which is also known as the "input gate layer," will make decisions about those values that are needed to be updated.

A vector of new candidate values is created so that they can be added to the state by the tanh layer.



$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$$

$$c\tilde{}_t = tanh(w_c[h_{t-1}, x_t] + b_c)$$

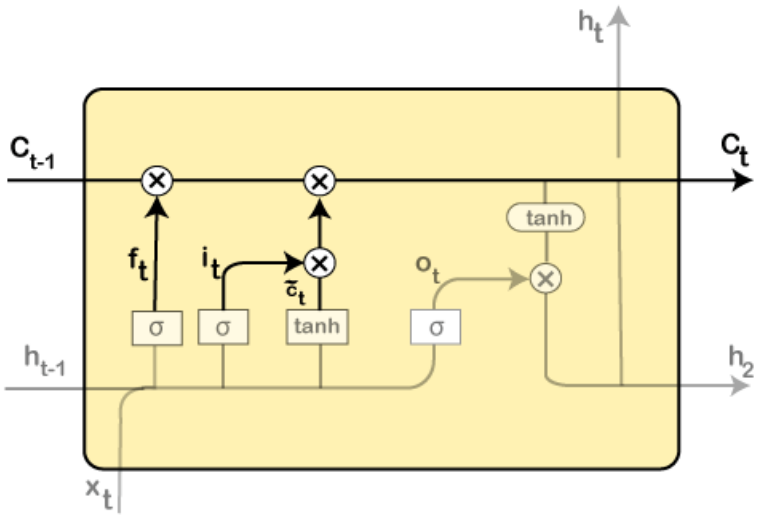
Then the new input, as well as the preceding timestamp's input, will get passed through a sigmoid function that will result in the value i(t), which will then be multiplied by c(t) followed by adding it to the cell state.

In the next step, we will combine both of them so as to update the state.

Step 3:

In the 3rd step, the previous cell state Ct-1 will get updated into the new cell state Ct.

And for that, we will need to multiply the old state (Ct-1) by f(t), keeping the things aside that we thought that we earlier decided to leave.



$$c_t = f_t * c_{t-1} + i_t{}^* \, c\tilde{}_t$$

Next, we will add i_t* c˜_t, which is the new candidate values. It has been actually scaled by how much we wanted to update each state value.

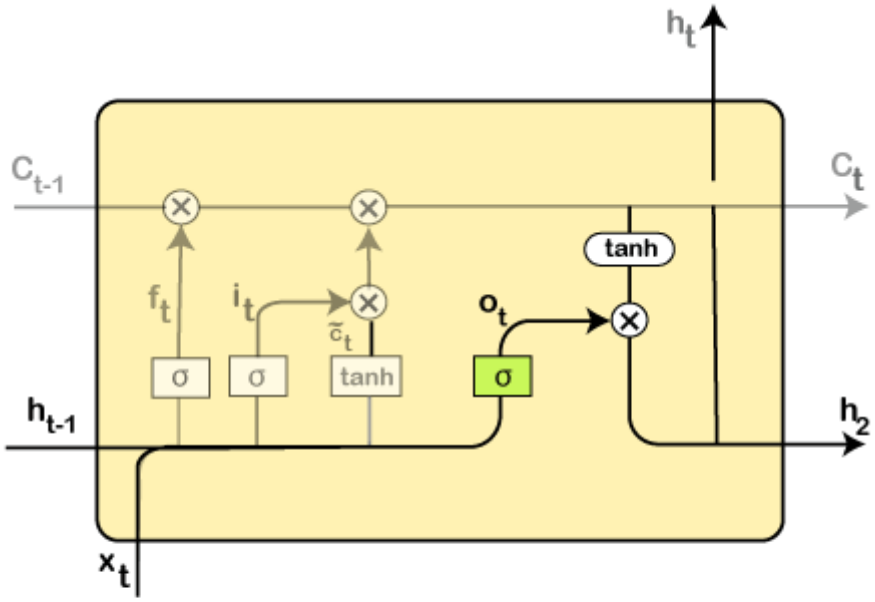
In the second step, we decided to do make use of the data, which is only required at that stage. However, in the third step, we have executed it.

Step 4:

In the 4th step, we will run the sigmoid layer that will decide for those parts of the cell state that will result in the output.

Next, we will put the cell state through tanh, which means we will be pushing the values in between the range of -1 and 1.

And then, further, we will multiply it with the sigmoid gate's output so that only the decided parts results in the output.



$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(c_t)$$

In this step, we will be doing some calculations that will result in the output.

However, the output consists of only the outputs there were decided to be carry forwarded in the previous steps and not all the outputs at once.

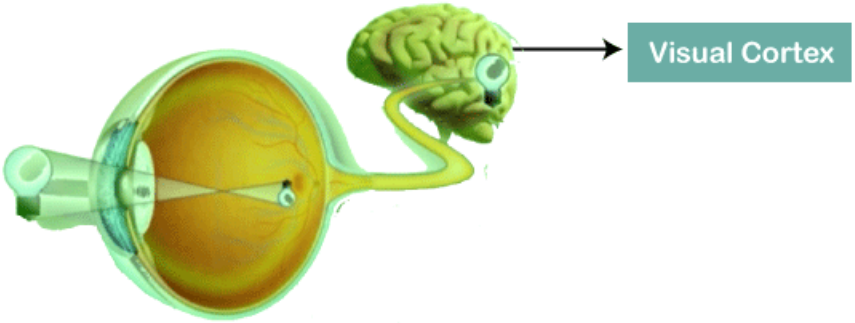
A Quick Recap:

- At first, we find out what is to be dropped.
- Then in the second step, it included newly added inputs to the network.
- In the third step, the earlier obtained inputs get combined in order to produce the new cell states.
- Lastly, we arrived at the output as per requirement.

***

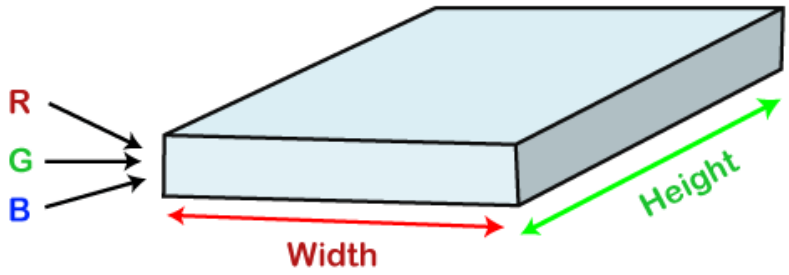
Convolutional Neural Network:

Convolutional Neural Networks are a special type of feed-forward artificial neural network in which the connectivity pattern between its neuron is inspired by the visual cortex.

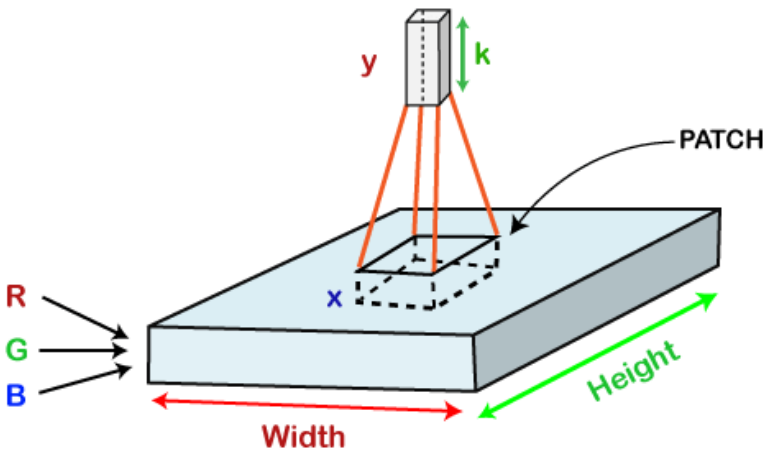


The visual cortex encompasses a small region of cells that are region sensitive to visual fields. In case some certain orientation edges are present then only some individual neuronal cells get fired inside the brain such as some neurons responds as and when they get exposed to the vertical edges, however some responds when they are shown to horizontal or diagonal edges, which is nothing but the motivation behind Convolutional Neural Networks.

The Convolutional Neural Networks, which are also called as covnets, are nothing but neural networks, sharing their parameters. Suppose that there is an image, which is embodied as a cuboid, such that it encompasses length, width, and height. Here the dimensions of the image are represented by the Red, Green, and Blue channels, as shown in the image given below.



Now assume that we have taken a small patch of the same image, followed by running a small neural network on it, having k number of outputs, which is represented in a vertical manner. Now when we slide our small neural network all over the image, it will result in another image constituting different width, height as well as depth. We will notice that rather than having R, G, B channels, we have come across some more channels that, too, with less width and height, which is actually the concept of Convolution. In case, if we accomplished in having similar patch size as that of the image, then it would have been a regular neural network. We have some wights due to this small patch.



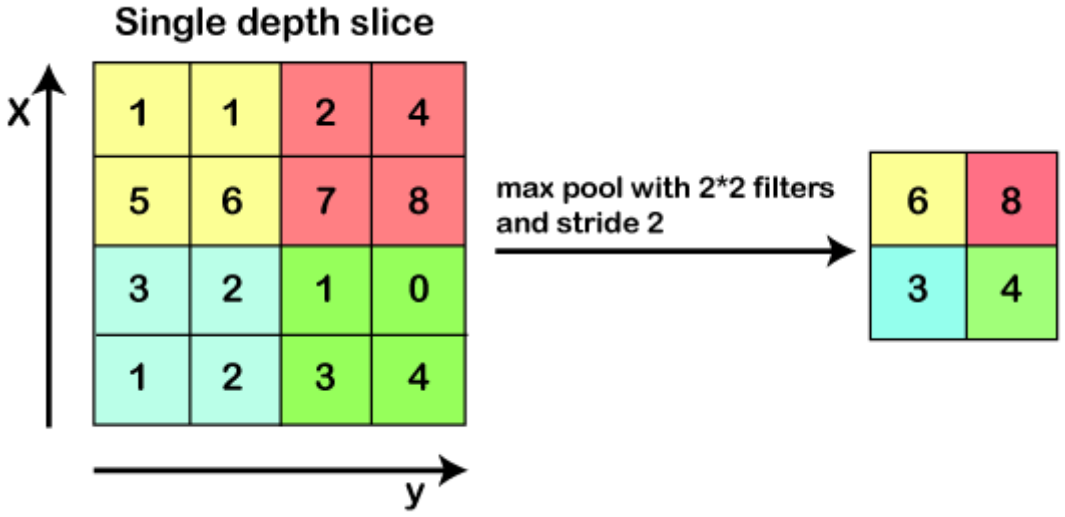
Mathematically it could be understood as follows:

- The Convolutional layers encompass a set of learnable filters, such that each filter embraces small width, height as well as depth as that of the provided input volume (if the image is the input layer then probably it would be 3).
- Suppose that we want to run the convolution over the image that comprises of 34x34x3 dimension, such that the size of a filter can be axax3. Here a can be any of the above 3, 5, 7, etc. It must be small in comparison to the dimension of the image.
- Each filter gets slide all over the input volume during the forward pass. It slides step by step, calling each individual step as a stride that encompasses a value of 2 or 3 or 4 for higher-dimensional images, followed by calculating a dot product in between filter's weights and patch from input volume.
- It will result in 2-Dimensional output for each filter as and when we slide our filters followed by stacking them together so as to achieve an output volume to have a similar depth value as that of the number of filters. And then, the network will learn all the filters.

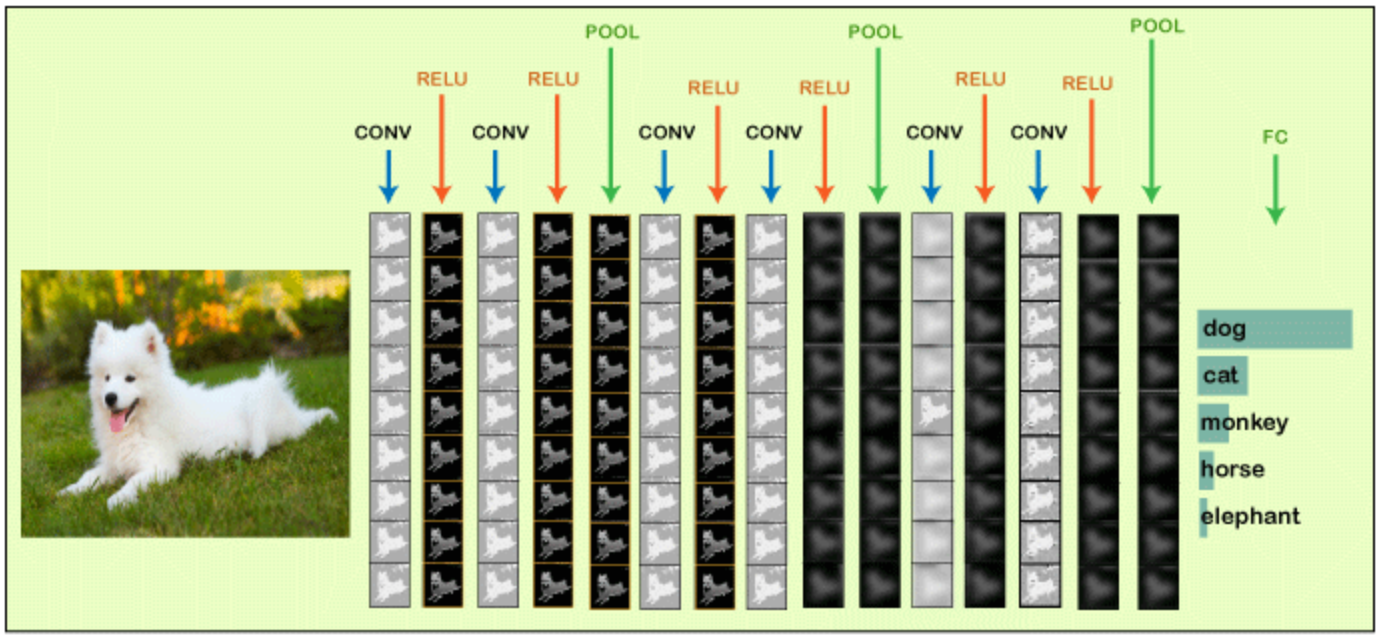
Working of CNN

Generally, a Convolutional Neural Network has three layers, which are as follows;

- **Input**: If the image consists of 32 widths, 32 height encompassing three R, G, B channels, then it will hold the raw pixel([32x32x3]) values of an image.
- **Convolution**: It computes the output of those neurons, which are associated with input's local regions, such that each neuron will calculate a dot product in between weights and a small region to which they are actually linked to in the input volume. For example, if we choose to incorporate 12 filters, then it will result in a volume of [32x32x12].
- **ReLU Layer**: It is specially used to apply an activation function elementwise, like as max (0, x) thresholding at zero. It results in ([32x32x12]), which relates to an unchanged size of the volume.
- **Pooling**: This layer is used to perform a downsampling operation along the spatial dimensions (width, height) that results in [16x16x12] volume.

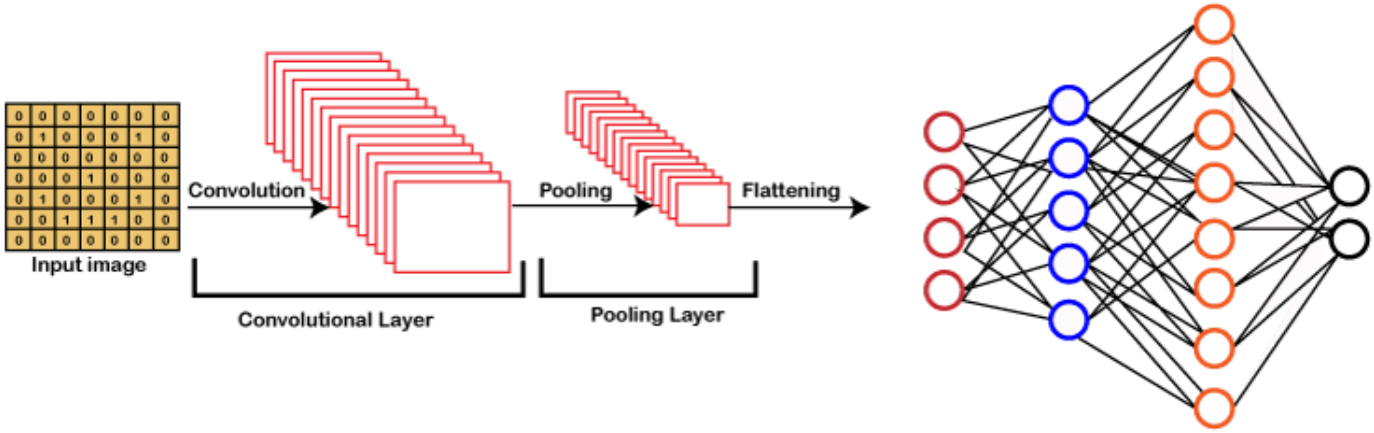


- **Locally Connected**: It can be defined as a regular neural network layer that receives an input from the preceding layer followed by computing the class scores and results in a 1-Dimensional array that has the equal size to that of the number of classes.



We will start with an input image to which we will be applying multiple feature detectors, which are also called as filters to create the feature maps that comprises of a Convolution layer. Then on the top of that layer, we will be applying the ReLU or Rectified Linear Unit to remove any linearity or increase non-linearity in our images.

Next, we will apply a Pooling layer to our Convolutional layer, so that from every feature map we create a Pooled feature map as the main purpose of the pooling layer is to make sure that we have spatial invariance in our images. It also helps to reduce the size of our images as well as avoid any kind of overfitting of our data. After that, we will flatten all of our pooled images into one long vector or column of all of these values, followed by inputting these values into our artificial neural network. Lastly, we will feed it into the locally connected layer to achieve the final output.



***

Hyperparameter:

Hyperparameters are the variables which determines the network structure(Eg: Number of Hidden Units) and the variables which determine how the network is trained(Eg: Learning Rate).

Hyperparameters are set before training(before optimizing the weights and bias).

Hyperparameters related to Network structure

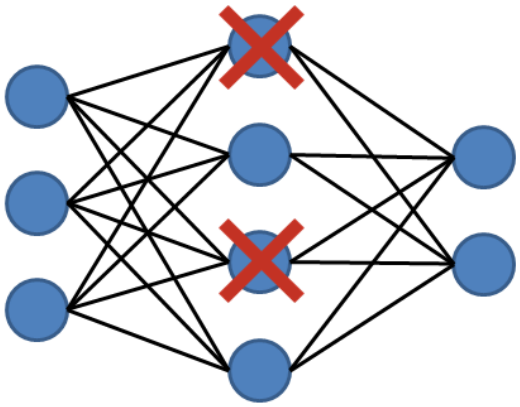
Number of Hidden Layers and units

Hidden layers are the layers between input layer and output layer.

“Very simple. Just keep adding layers until the test error does not improve anymore.”

Many hidden units within a layer with regularization techniques can increase accuracy. Smaller number of units may cause underfitting.

Dropout



Random neurons are cancelled

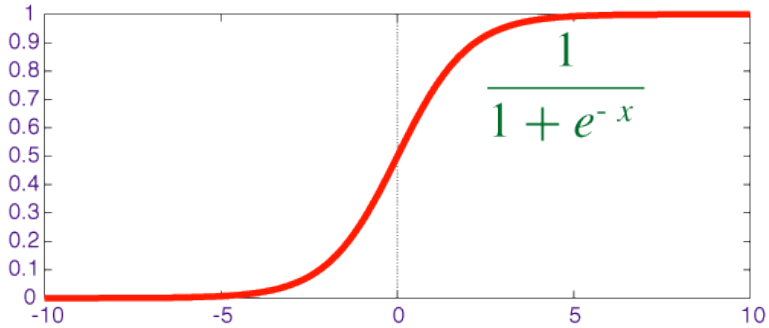
Dropout is regularization technique to avoid overfitting (increase the validation accuracy) thus increasing the generalizing power.

- Generally, use a small dropout value of 20%-50% of neurons with 20% providing a good starting point. A probability too low has minimal effect and a value too high results in under-learning by the network.
- Use a larger network. You are likely to get better performance when dropout is used on a larger network, giving the model more of an opportunity to learn independent representations.

Network Weight Initialization

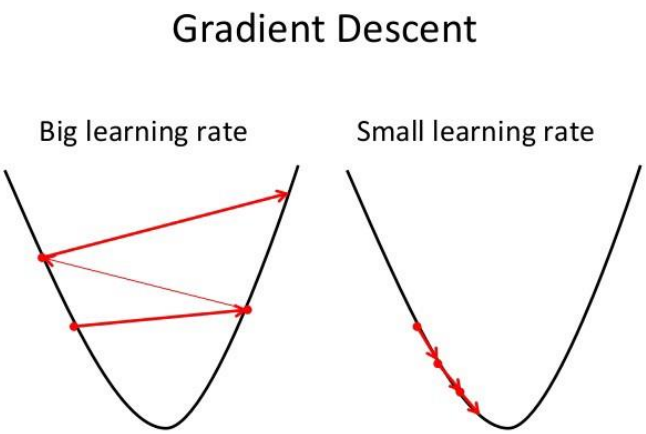
- Ideally, it may be better to use different weight initialization schemes according to the activation function used on each layer.
- Mostly **uniform distribution** is used.

Activation function



Hyperparameters related to Training Algorithm

Learning Rate



- The learning rate defines how quickly a network updates its parameters.
- Low learning rate slows down the learning process but converges smoothly. Larger learning rate speeds up the learning but may not converge.
- Usually a decaying Learning rate is preferred.

Momentum

- Momentum helps to know the direction of the next step with the knowledge of the previous steps. It helps to prevent oscillations. A typical choice of momentum is between 0.5 to 0.9.

Number of epochs

- Number of epochs is the number of times the whole training data is shown to the network while training.
- Increase the number of epochs until the validation accuracy starts decreasing even when training accuracy is increasing(overfitting).

Batch size

- Mini batch size is the number of sub samples given to the network after which parameter update happens.
- A good default for batch size might be 32. Also try 32, 64, 128, 256, and so on.

***