Shree SATHYAM COLLEGE OF ENGG & TECH, Sankari

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

AD3501 / DEEPLEARNING - NOTES

AD3501	DEEP LEARNING	L T P C
		3 0 0 3

COURSE OBJECTIVES:

• To understand and need and principles of deep neural networks
• To understand CNN and RNN architectures of deep neural networks
• To comprehend advanced deep learning models
• To learn the evaluation metrics for deep learning models

UNIT I DEEP NETWORKS BASICS

Linear Algebra: Scalars -- Vectors -- Matrices and tensors; Probability Distributions -- Gradient-based Optimization – Machine Learning Basics: Capacity -- Overfitting and underfitting --Hyperparameters and validation sets -- Estimators -- Bias and variance -- Stochastic gradient descent -- Challenges motivating deep learning; Deep Networks: Deep feedforward networks; Regularization -- Optimization.

UNIT II CONVOLUTIONAL NEURAL NETWORKS

Convolution Operation -- Sparse Interactions -- Parameter Sharing -- Equivariance -- Pooling -- Convolution Variants: Strided -- Tiled -- Transposed and dilated convolutions; CNN Learning: Nonlinearity Functions -- Loss Functions -- Regularization -- Optimizers --Gradient Computation.

UNIT III RECURRENT NEURAL NETWORKS

Unfolding Graphs -- RNN Design Patterns: Acceptor -- Encoder --Transducer; Gradient Computation -- Sequence Modeling Conditioned on Contexts -- Bidirectional RNN -- Sequence to Sequence RNN – Deep Recurrent Networks -- Recursive Neural Networks -- Long Term Dependencies; Leaky Units: Skip connections and dropouts; Gated Architecture: LSTM.

UNIT IV MODEL EVALUATION

Performance metrics -- Baseline Models -- Hyperparameters: Manual Hyperparameter -- Automatic Hyperparameter -- Grid search -- Random search -- Debugging strategies.

UNIT V AUTOENCODERS AND GENERATIVE MODELS

Autoencoders: Undercomplete autoencoders -- Regularized autoencoders -- Stochastic encoders and decoders -- Learning with autoencoders; Deep Generative Models: Variational autoencoders – Generative adversarial networks.

TEXT BOOK

1. Ian Goodfellow, Yoshua Bengio, Aaron Courville, ``Deep Learning", MIT Press, 2016.
2. Andrew Glassner, "Deep Learning: A Visual Approach", No Starch Press, 2021.

REFERENCES

1. Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, Mohammed Bennamoun, ``A Guide to Convolutional Neural Networks for Computer Vision", Synthesis Lectures on Computer Vision, Morgan & Claypool publishers, 2018. 2. Yoav Goldberg, ``Neural Network Methods for Natural Language Processing", Synthesis Lectures on Human Language Technologies, Morgan & Claypool publishers, 2017.
3. Francois Chollet, ``Deep Learning with Python", Manning Publications Co, 2018.
4. Charu C. Aggarwal, ``Neural Networks and Deep Learning: A Textbook", Springer International Publishing, 2018.
5. Josh Patterson, Adam Gibson, ``Deep Learning: A Practitioner's Approach", O'Reilly Media, 2017.

Shree SATHYAM COLLEGE OF ENGG & TECH, Sankari

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

AD3501 / DEEPLEARNING - NOTES

UNIT - I

DEEP NETWORKS BASICS

Linear Algebra: Scalars -- Vectors -- Matrices and tensors; Probability Distributions -- Gradientbased Optimization – Machine Learning Basics: Capacity -- Overfitting and underfitting -- Hyperparameters and validation sets -- Estimators -- Bias and variance -- Stochastic gradient descent -- Challenges motivating deep learning; Deep Networks: Deep feedforward networks; Regularization -- Optimization.

Two Marks / Part - A

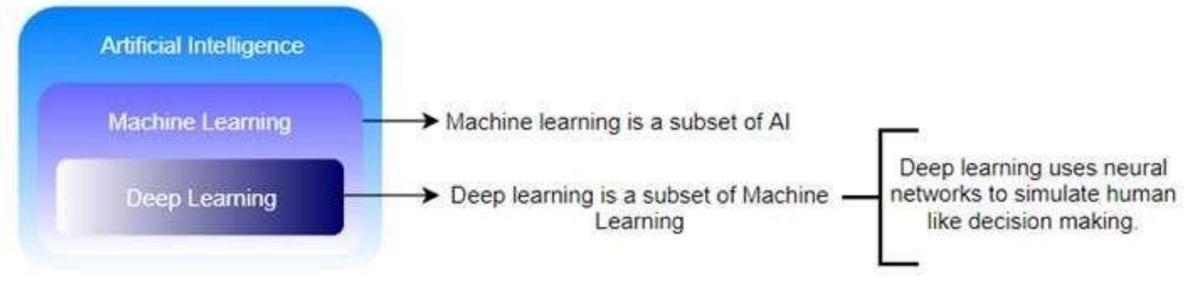
1. **What is Deep Learning?**

   Deep learning is a part of machine learning with an algorithm inspired by the structure and function of the brain, which is called an artificial neural network. In the mid-1960s, Alexey Grigorevich Ivakhnenko published the first general, while working on deep learning network. Deep learning is suited over a range of fields such as computer vision, speech recognition, natural language processing, etc

2. **What are the main differences between AI, Machine Learning, and Deep Learning?**

   AI stands for Artificial Intelligence. It is a technique which enables machines to mimic human behavior.

   Machine Learning is a subset of AI which uses statistical methods to enable machines to improve with experiences.


Artificial Intelligence
Machine Learning
Deep Learning
Machine learning is a subset of AI
Deep learning is a subset of Machine Learning
Deep learning uses neural networks to simulate human like decision making.

   Deep learning is a part of Machine learning, which makes the computation of multi-layer neural networks feasible. It takes advantage of neural networks to simulate human-like decision making.

3. **Differentiate supervised and unsupervised deep learning procedures.**

   Supervised learning is a system in which both input and desired output data are provided. Input and output data are labeled to provide a learning basis for future data processing.

   Unsupervised procedure does not need labeling information explicitly, and the operations can be carried out without the same. The common unsupervised learning method is cluster analysis. It is used for exploratory data analysis to find hidden patterns or grouping in data.

4. **What are the applications of deep learning?**

   There are various applications of deep learning:

   Computer vision

   Natural language processing and pattern recognition

   Image recognition and processing

   Machine translation

   Sentiment analysis

   Question answering system

   Object Classification and Detection

   Automatic Handwriting Generation

   Automatic Text Generation.

5. **What is scalar and vector?**

   A scalar is just a single number, in contrast to most of the other objects like Vectors, which are usually arrays of multiple numbers.

6. **What are matrices and tensors?**

   Matrices: A matrix is a 2D array of numbers, so each element is identified by two subscripts instead of just one. We usually give matrices uppercase variable names with bold characters, such as A.

   We usually identify the elements of a matrix by using its name in italics but not in bold, and the subscripts are listed with separating commas.

   Tensors: In some cases, we'll need an array with more than two axes. In the general case, an array of numbers arranged on a regular grid with a varying number of axes is called a tensor. We note a tensor named "A" with this font: A.

7. **Why probability is important in deep learning?**

   Probability is the science of quantifying uncertain things. Most of machine learning and deep learning systems utilize a lot of data to learn about patterns in the data. Whenever data is utilized in a system rather than sole logic, uncertainty grows up and whenever uncertainty grows up, probability becomes relevant.

   By introducing probability to a deep learning system, we introduce common sense to the system. Otherwise the system would be very brittle and will not be useful.In deep learning, several models like Bayesian models, probabilistic graphical models, hidden markov models are used. They depend entirely on probability concepts.

8. **Define Random Variable.**

   A random variable is a variable that can take on different values randomly. We typically denote the random variable itself with a lower case letter in plain typeface, and the values it can take on with lower case script letters. For example, x1 and x2 are both possible values that the random variable x can take on.

9. **Do random variables is discrete or continuous?**

Random variables may be discrete or continuous. A discrete random variable is one that has a finite or countably infinite number of states. Note that these states are not necessarily the integers; they can also just be named states that are not considered to have any numerical value. A continuous random variable is associated with a real value.

10. **What are probability distributions?**

A probability distribution is a description of how likely a random variable or set of random variables is to take on each of its possible states. The way we describe probability distributions depends on whether the variables are discrete or continuous.

11. **Define Probability mass function?**

A probability distribution over discrete variables may be described using a probability mass function (PMF). We typically denote probability mass functions with a capital P. Often we associate each random variable with a different probability mass function and the reader must infer which probability mass function to use based on the identity of the random variable, rather than the name of the function; P(x) is usually not the same as P(y).

12. **List the properties that probability mass function satisfies?**

• The domain of P must be the set of all possible states of x.

• $\forall x \in x, 0 \leq P(x) \leq 1$. An impossible event has probability 0 and no state can be less probable than that. Likewise, an event that is guaranteed to happen has probability 1, and no state can have a greater chance of occurring.

• $\sum_{x \in x} P(x) = 1$. We refer to this property as being normalized. Without this property, we could obtain probabilities greater than one by computing the probability of one of many events occurring.

13. **List the properties that probability density function satisfies?**

When working with continuous random variables, we describe probability distributions using a probability density function (PDF) rather than a probability mass function.

To be a probability density function, a function p must satisfy the following properties:
• The domain of p must be the set of all possible states of x.
• $\forall x \in x, p(x) \geq 0$. Note that we do not require $p(x) \leq 1$.
• $\int p(x)dx = 1$.

14. **What is Gradient based optimizer?**

Gradient descent is an optimization algorithm that's used when training deep learning models. It's based on a convex function and updates its parameters iteratively to minimize a given function to its local minimum.

The notation used in the above Formula is given below,
In the above formula,

- α is the learning rate,
- J is the cost function, and
- ϴ is the parameter to be updated.

As you can see, the gradient represents the partial derivative of J(cost function) with respect to ϴj

15. Why overfitting and underfitting in ML?

Factors determining how well an ML algorithm will perform are its ability to:
1. Make the training error small
2. Make gap between training and test errors small

• They correspond to two ML challenges
Underfitting - Inability to obtain low enough error rate on the training set
Overfitting - Gap between training error and testing error is too large

We can control whether a model is more likely to overfit or underfit by altering its capacity

16. What is capacity of a model?

Model capacity is ability to fit variety of functions
– Model with Low capacity struggles to fit training set
– A High capacity model can overfit by memorizing properties of training set not useful on test set
• When model has higher capacity, it overfits – One way to control capacity of a learning algorithm is by choosing the hypothesis space
• i.e., set of functions that the learning algorithm is allowed to select as being the solution

17. How to control the capacity of learning algorithm?

One way to control the capacity of a learning algorithm is by choosing its hypothesis space, the set of functions that the learning algorithm is allowed to select as being the solution. For example, the linear regression algorithm has the set of all linear functions of its input as its hypothesis space. We can generalize linear regression to include polynomials, rather than just linear functions, in its hypothesis space. Doing so increases the model's capacity

18. Define Bayes error.

Ideal model is an oracle that knows the true probability distributions that generate the data • Even such a model incurs some error due to noise/overlap in the distributions • The error incurred by an oracle making predictions from the true distribution p(x,y) is called the Bayes error

19. Why hyperparameters in ML?

Most ML algorithms have hyperparameters
– We can use to control algorithm behavior
– Values of hyperparameters are not adapted by learning algorithm itself
• Although, we can design nested learning where one learning algorithm
– Which learns best hyperparameters for another learning algorithm.

20. How to solve overfitting problem caused by learning hyperparameters on training dataset?

• To solve the problem, we use a validation set

– Examples that training algorithm does not observe

• Test examples should not be used to make choices about the model hyperparameters • Training data is split into two disjoint parts

– First to learn the parameters

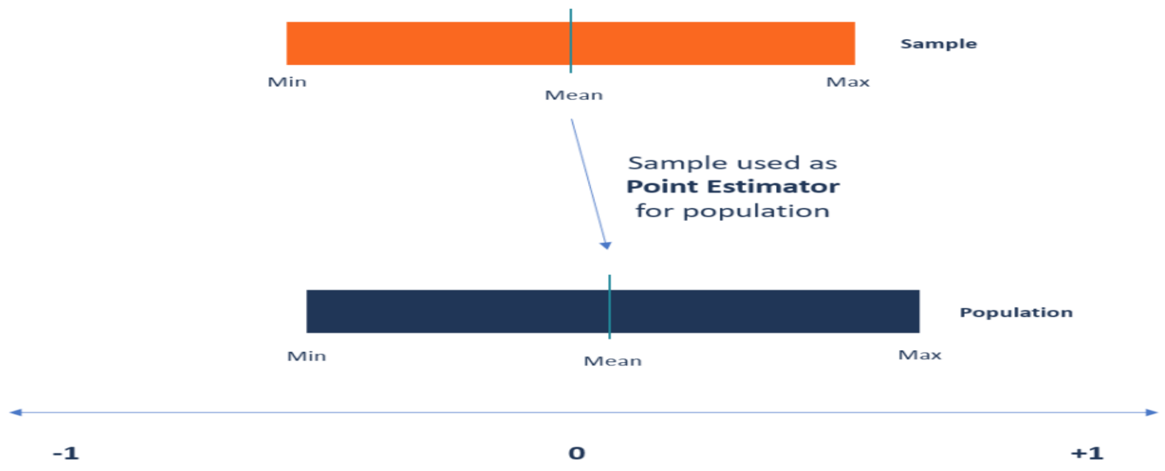
– Other is the validation set to estimate generalization error during or after training

• allowing for the hyperparameters to be updated

– Typically, 80% of training data for training and 20% for validation

21. What are point estimators?

Point estimators are functions that are used to find an approximate value of a population parameter from random samples of the population. They use the sample data of a population to calculate a point estimate or a statistic that serves as the best estimate of an unknown parameter of a population.



Most often, the existing methods of finding the parameters of large populations are unrealistic. For example, when finding the average age of kids attending kindergarten, it will be impossible to collect the exact age of every kindergarten kid in the world. Instead, a statistician can use the point estimator to make an estimate of the population parameter.

22. List the characteristics or Properties of Point Estimators?

The following are the main characteristics of point estimators:

1. Bias

The bias of a point estimator is defined as the difference between the expected value of the estimator and the value of the parameter being estimated. When the estimated value of the parameter and the value of the parameter being estimated are equal, the estimator is considered unbiased.

Also, the closer the expected value of a parameter is to the value of the parameter being measured, the lesser the bias is.

2. Consistency

Consistency tells us how close the point estimator stays to the value of the parameter as it increases in size. The point estimator requires a large sample size for it to be more consistent and accurate.
You can also check if a point estimator is consistent by looking at its corresponding expected value and variance. For the point estimator to be consistent, the expected value should move toward the true value of the parameter.
3. Most efficient or unbiased
The most efficient point estimator is the one with the smallest variance of all the unbiased and consistent estimators. The variance measures the level of dispersion from the estimate, and the smallest variance should vary the least from one sample to the other.

23. Define Stochastic Gradient Descent with merits and demerits.

Stochastic Gradient Descent (SGD) is a variant of the Gradient Descent algorithm used for optimizing machine learning models. In this variant, only one random training example is used to calculate the gradient and update the parameters at each iteration. Here are some of the advantages and disadvantages of using SGD:

Advantages of Gradient Descent

Speed: SGD is faster than other variants of Gradient Descent such as Batch Gradient Descent and Mini-Batch Gradient Descent since it uses only one example to update the parameters.

Memory Efficiency: Since SGD updates the parameters for each training example one at a time, it is memory-efficient and can handle large datasets that cannot fit into memory.

Avoidance of Local Minima: Due to the noisy updates in SGD, it has the ability to escape from local minima and converges to a global minimum.

Disadvantages of Gradient Descent

Noisy updates: The updates in SGD are noisy and have a high variance, which can make the optimization process less stable and lead to oscillations around the minimum.

Slow Convergence: SGD may require more iterations to converge to the minimum since it updates the parameters for each training example one at a time.

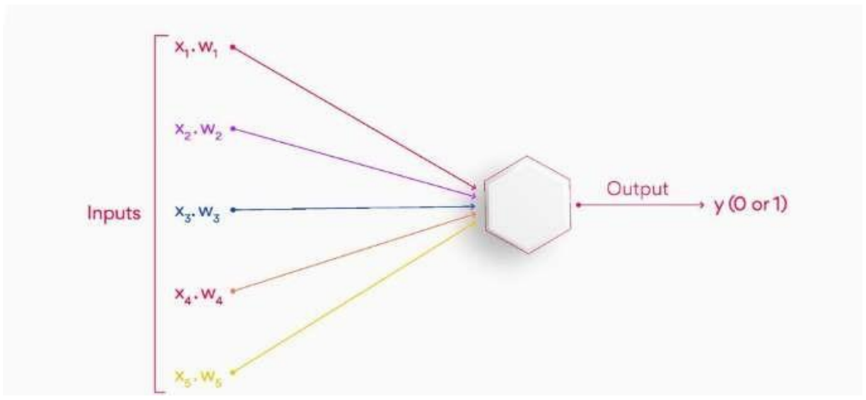
Sensitivity to Learning Rate: The choice of learning rate can be critical in SGD since using a high learning rate can cause the algorithm to overshoot the minimum, while a low learning rate can make the algorithm converge slowly.

Less Accurate: Due to the noisy updates, SGD may not converge to the exact global minimum and can result in a suboptimal solution. This can be mitigated by using techniques such as learning rate scheduling and momentum-based updates

24. What is a deep feedforward network?

In a feedforward network, the information moves only in the forward direction, from the input layer, through the hidden layers (if they exist), and to the output layer. There are no cycles or loops in this network. Feedforward neural networks are sometimes ambiguously called multilayer perceptron's.

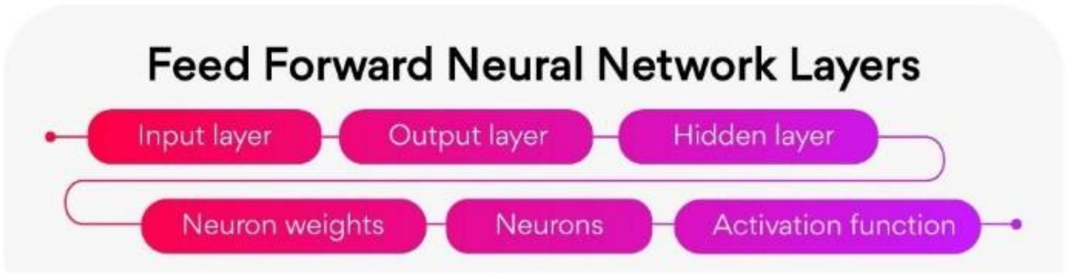
25. What is the working principle of a feed forward neural network?



When the feed forward neural network gets simplified, it can appear as a single layer perceptron.

This model multiplies inputs with weights as they enter the layer. Afterward, the weighted input values get added together to get the sum. As long as the sum of the values rises above a certain threshold, set at zero, the output value is usually 1, while if it falls below the threshold, it is usually -1.

26. What are the Layers of feed forward neural network?



27. **Brief on classification of activation function.**

An activation function can be classified into three major categories: sigmoid, Tanh, and Rectified Linear Unit (ReLu).

- Sigmoid:

Input values between 0 and 1 get mapped to the output values.

- Tanh:

A value between -1 and 1 gets mapped to the input values.

- Rectified linear Unit:

Only positive values are allowed to flow through this function. Negative values get mapped to 0.

28. **What is Regularization?**

Regularization is a technique used in machine learning and deep learning to prevent overfitting and improve the generalization performance of a model. It involves adding a penalty term to the loss function during training. This penalty discourages the model from becoming too complex or having large parameter values, which helps in controlling the model’s ability to fit noise in the training data. Regularization methods include L1 and L2 regularization, dropout, early stopping, and more.

29. **What is dropout in neural network?**

Dropout is a regularization technique used in neural networks to prevent overfitting. During training, a random subset of neurons is “dropped out” by setting their outputs to zero with a certain probability. This forces the network to learn more robust and independent features, as it cannot rely on specific neurons. Dropout improves generalization and reduces the risk of overfitting.

30. **Difference between regularization and optimization.**

The main conceptual difference is that optimization is about finding the set of parameters/weights that maximizes/minimizes some objective function (which can also include a regularization term), while regularization is about limiting the values that your parameters can take during the optimization/learning/training, so optimization with regularisation (especially, with L1 and L2 regularization) can be thought of as constrained optimization, but, in some cases, such as dropout, it can also be thought of as a way of introducing noise in the training process.

UNIT II

CONVOLUTIONAL NEURAL NETWORKS

Convolution Operation -- Sparse Interactions -- Parameter Sharing -- Equivariance -- Pooling -- Convolution Variants: Strided -- Tiled -- Transposed and dilated convolutions; CNN Learning: Nonlinearity Functions -- Loss Functions -- Regularization -- Optimizers -- Gradient Computation.

Part - A

1. **What is convolutional neural network?**
   A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interprehe image or visual data.

2. **What are the three types of layers in neural network?**
   In a regular Neural Network there are three types of layers:
   Input Layers: It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).
   Hidden Layer: The input from the Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.
   Output Layer: The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

3. **Define feed forward and Back propagation.**
   The data is fed into the model and output from each layer is obtained from the above step is called feed forward, we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. The error function measures how well the network is performing. After that, we backpropagate into the

model by calculating the derivatives. This step is called Backpropagation which basically is used to minimize the loss.

4. **What is convolution operation with the representation of equation?**

Applying a weighted average operation at every moment with respective to time , a new estimated function s is obtained

$$s(t) = \int x(a)w(t - a)da$$

This operation is called convolution. The convolution operation is denoted as asterick.

$$s(t) = (x * w)(t)$$

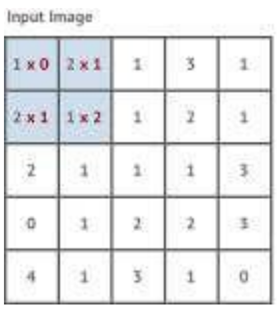
Where

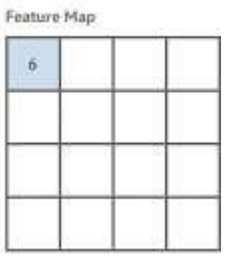
w= valid probability density function and w needs to 0 for all negative arguments. In general, convolution is defined for any functions for which the above integral is defined, and may be used for other purposes besides taking weighted averages.

In convolutional network terminology, the first argument (in this example, the function x) to the convolution is often referred to as the input and the second argument (in this example, the function w) as the kernel. The output is sometimes referred to as the feature map.

5. **What are the motivation of convolution?**
   1. Sparse interactions
   2. Parameter Sharing
   3. Equivariant representations.
6. **Define Sparse Connectivity or Sparse interactions.**

A Convolution layer defines a window or filter or kernel by which they examine a subset of the data, and subsequently scans the data looking through this window. his is what we call **sparse connectivity** or **sparse interactions** or **sparse weights.** Actually, it limits the activated connections at each layer. In the example below an 5x5 input with a 2x2 filter produces a reduced 4x4 output. The first element of feature map is calculated by the convolution of the input area with the filter i.e.





Apply 2x2 filter to the input and get the first convolutional layer (a feature map)

$$1 \times 0 + 2 \times 1 + 2 \times 1 + 1 \times 2 = 6$$

First element of the feature map

7. **How many filters use at each layer?**
    1) hyperparameter which is called the depth of the output volume.

    2) Another hyperparameter is the stride that defines how much we slide the filter over the data. For example if stride is 1 then we move the window by 1 pixel at a time over the image, when our input is an image. When we use larger values of stride 2 or 3 we allow jumping 2 or pixels at a time. This reduces significantly the output size.

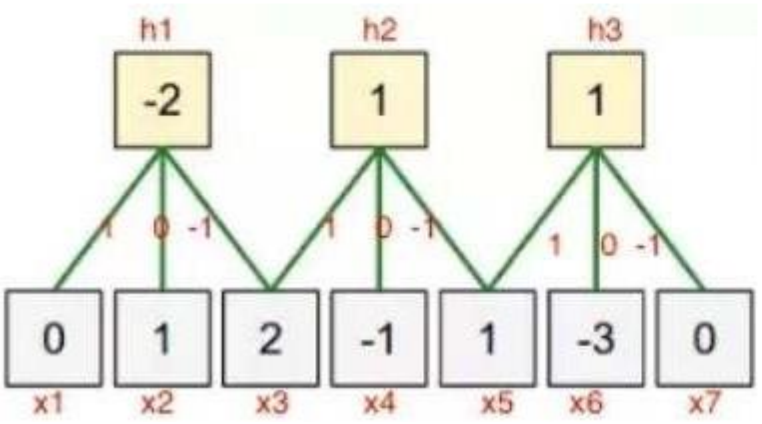
    3) The last hyperparameter is the size of zero-padding, when sometimes is convenient to pad the input volume with zeros around the border.

8. **Write the formula to find how many neurons fit for a network?**

To compute the spatial size of the output volume as a function of the input volume size (W), the receptive field size of the Conv Layer neurons (F), the stride with which they are applied (S), and the amount of zero padding used (P) on the border. The formula for calculating how many neurons "fit" is given by

$$\frac{W - F + 2P}{S} + 1$$

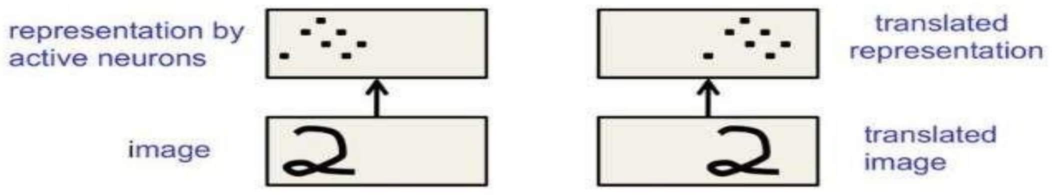
9. **Why parameter sharing is used in CNN?**
   Parameter sharing is used in the convolutional layers to reduce the number of parameters in the network. For example in the first convolutional layer let's say we have an output of 15x15x4 where 15 is the size of the output and 4 the number of filters used in this layer. For each output node in that layer we have the same filter, thus reducing dramatically the storage requirements of the model to the size of the filter.



The same filter (weights) (1, 0, -1) are used for that layer.

10. What is equivariant representations?

Equivariant means varying in the similar or equivalent proportion. Equivariant to translation means that a translation of input features results in an equivalent translation of outputs. It makes the CNN understand the rotation or proportion change. The equivariance allows the network to generalize edge, texture, shape, detection in different locations.



11. Why pooling layer in CNN?

A pooling layer is another building block of a CNN. Pooling Its function is to progressively reduce the spatial size of the representation to reduce the network complexity and computational cost.

12. What are the two types of pooling widely used?

There are two types of widely used pooling in CNN layer:

Max Pooling
Average Pooling

13. Outline the problem arise due to convolution.

1. Every time after convolution operation, original image size get shrinks.
2. The second issue is that, when kernel moves over original images, it touches the edge of the image less number of times and touches the middle of the image more number of times and it overlaps also in the middle. So, the corner features of any image or on the edges aren't used much in the output.

14. Define Padding and Stride.

- Padding preserves the size of the original image.
- Stride is the number of pixels shifts over the input matrix. For padding p, filter size $*$ and input image size $*$ and stride '' our output image dimension will be[{(+ 2 − + 1) / } + 1] $*$ [{(+ 2 − + 1) / } + 1].

15. What is the difference between normal convolution and transposed convolution?

Traditional convolution determines the output value as the dot product between filter and input, by moving the filter kernel for two pixels in every step, the input is down sampled by factor two. For transposed convolution, the input value determines thefilter values that will be written to the output.

16. What is a transposed convolution?

Transposed convolutions are standard convolutions but with a modified input feature map. The stride and padding do not correspond to the number of zeros added around the image and the amount of shift in the kernel when sliding it across the input, as they would in a standard convolution operation.

17. **What does non-linearity mean?**
    It means that the neural network can successfully approximate functions that do not follow linearity or it can successfully predict the class of a function that is divided by a decision boundary which is not linear.

18. **What is linear and non-linear in deep learning?**
    Linearity refers to the property of a system or model where the output is directly proportional to the input, while nonlinearity implies that the relationship between input and output is more complex and cannot be expressed as a simple linear function.

19. **What is the loss function in CNN machine learning?**

A loss function is a function that compares the target and predicted output values; measures how well the neural network models the training data. When training, we aim to minimize this loss between the predicted and target outputs.

20. **List any two loss function.**
    1. Regression

    MSE(Mean Squared Error)
    MAE(Mean Absolute Error)
    Hubber loss

    2. Classification

    Binary cross-entropy
    Categorical cross-entropy

21. **Differentiate loss function and cost function.**
    Loss Function:

    A loss function/error function is for a single training example/input.

    Cost Function:

    A cost function, on the other hand, is the average loss over the entire training dataset.

22. **Give details on some of regularization techniques used in CNN?**
    L1 and L2 Regularization (Weight Decay), Dropout, Batch Normalization, Data Augmentation and Early Stopping.

23. Define Mean Absolute Error (MAE)

Mean absolute error (MAE) also called **L1 Loss** is a loss function used for regressionproblems. It represents the difference between the original and predicted values extracted by averaging the absolute difference over the data set.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}|$$

24. What is regularization in CNN?

Regularization is a technique that helps prevent overfitting, which occurs when a neuralnetwork learns too much from the training data and fails to generalize well to new data.

25. What are the commonly used non linearity function using CNN?

1. Rectified Linear Unit (ReLU)
2. Leaky ReLU
3. Sigmoid
4. Hyperbolic Tangent (Tanh)
5. Softmax

26. Why is it important to place non-linearities between the layers of neural networks?

Non-linearity introduces more degrees of freedom to the model. It lets it capture more complex representations which can be used towards the task at hand. A deep neural network without non-linearities is essentially a linear regression.

Shree SATHYAM COLLEGE OF ENGG & TECH

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

AD3501 / DEEPLEARNING - NOTES

UNIT – I DEEP NETWORKS BASICS
Linear Algebra: Scalars -- Vectors -- Matrices and tensors; Probability Distributions -- Gradient based Optimization – Machine Learning Basics: Capacity -- Overfitting and underfitting -- Hyperparameters and validation sets -- Estimators -- Bias and variance -- Stochastic gradient descent -- Challenges motivating deep learning; Deep Networks: Deep feed forward networks; Regularization -- Optimization.

1. Introduction:

Today, *Artificial Intelligence (AI)* is a thriving field with many practical applications and active research topics. We look to intelligent software to automate routine labor, understand speech or images, make diagnoses in medicine and support basic scientific research. In the early days of artificial intelligence, the field rapidly tackled and solved problems that are intellectually difficult for human beings but relatively straightforward for computers—problems that can be described by a list of formal, mathematical rules. The true challenge of AI lies in solving more intuitive problems. The solution is to allow computers to learnfrom experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts. By gathering knowledge from experience, this approach avoids the need for human operators to formally specify all of the knowledge that the computer needs. Thehierarchyof concepts allows the computer to learn complicated concepts by building them out of simpler ones. If one draws a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, this approach is called as *deep learning*.
A computer can reason about statements in these formal languages automatically using logical inference rules. This is known as the *knowledge base* approach to artificial intelligence. The difficulties faced by systems relying on hard-coded knowledge suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data. This capability is known as *machine learning*. The introduction of machine learning allowed computers to tackle problems involving knowledge of the real world and make decisions that appear subjective. A simple machine learning algorithm called *logistic regression* can determine whether to recommend cesarean delivery. A simple machine learning algorithm called *naive Bayes* can separate legitimate e-mail from spam e-mail.
The performance of these simple machine learning algorithms depends heavily on the *representation* of the data they are given. For example, when logistic regression is used to recommend cesarean delivery, the AI system does not examine the patient directly. Instead, the doctor tells the system several pieces of relevant information, such as the presence or absence of a uterine scar. Each piece of information included in the representationofthepatientisknownasa*feature*.Logisticregressionlearnshoweachofthesefeaturesof

the patient correlates with various outcomes. However, it cannot influence the way that the features are defined in any way. If logistic regression was given an MRI scan of the patient, rather than the doctor‘s formalized report, it would not be able to make useful predictions. Individual pixels in an MRI scan have negligible correlation with any complications that might occur during delivery.

This dependence on representations is a general phenomenon that appears throughout computer science and even daily life. In computer science, operations such as searching a collection of data can proceed exponentially faster if the collection is structured and indexed intelligently. People can easily perform arithmetic on Arabic numerals, but find arithmetic on Roman numerals much more time-consuming. It is not surprising that the choice of representation has an enormous effect on the performance of machine learning algorithms. Many artificial intelligence tasks can be solved by designing the right set of features to extract for that task, then providing these features to a simple machine learning algorithm. However, for many tasks, it is difficult to know what features should be extracted. For example, suppose that we would like to write a program to detect cars in photographs. We know that cars have wheels, so we might like to use the presence of a wheel as a feature. Unfortunately, it is difficult to describe exactly what a wheel looks like in terms of pixel values.

One solution to this problem is to use machine learning to discover not only the mapping from representation to output but also the representation itself. This approach is known as *representation learning*. Learned representations often result in much better performance than can be obtained with hand-designed representations. They also allow AI systems to rapidly adapt to new tasks, with minimal human intervention. A representation learning algorithm can discover a good set of features for a simple task in minutes, or a complex task in hours to months.

The quintessential example of a representation learning algorithm is the *autoencoder*. An autoencoder is the combination of an *encoder* function that converts the input data into a different representation, and a*decoder* function that converts the new representation back into the original format. Autoencoders are trained to preserve as much information as possible when an input is run through the encoder and then the decoder, but are also trained to make the new representation have various nice properties. Different kinds of autoencoders aim to achieve different kinds of properties. When designing features or algorithms for learning features, our goal is usually to separate the *factors of variation* that explain the observed data. A major source of difficulty in many real-world artificial intelligence applications is that many of the factors of variation influence every single pieceofdata weareable to observe. The individual pixels in animageof a red car might be very close to black at night. The shape of the car‘s silhouette depends on the viewing angle. It can be very difficult to extract such high-level, abstract features from raw data. *Deep learning* solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations.

Deep learning allows the computer to build complex concepts out of simpler concepts. Fig. 1.1showshowa deep learning system can represent the concept of an image of a person by combining simpler concepts, such as corners and contours, which are in turn defined in terms of edges. The quintessential example of a deep learning model is the feed forward deep network or *multilayer perceptron* (MLP). A multilayer perceptron is just a mathematical function mapping some set of input values to output values. The function is formed by composing many simpler functions. The idea of learning the right representation for the data provides one perspective on deep learning. Another perspective on deep learning is that depth allows the computer to learn a multi-step computer program. Each layer of the representation can be thought of as the state of the computer‘s memory after executing another set of instructions in parallel. Networks with greater

depth can execute more instructions in sequence. Sequential instructions offer great power because later instructions can refer back to the results of earlier instructions.
The input is presented at the, so named because it contains *visible layer* the variables that we are able to observe. Then a series of *hidden layers* extracts increasingly abstract features from the image. These layers are called ―hidden‖ because their values are not given in the data; instead the model must determine which concepts are useful for explaining the relationships in the observed data. The images here are visualizations of the kind of feature represented by each hidden unit. Given the pixels, the first layer can easily identify edges, by comparing the brightness of neighboring pixels. Given the first hidden layer‗s description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges. Given the second hidden layer‗s description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by finding specific collections of contours and corners. Finally, this description of the image in terms of the object parts it contains can be used to recognize the objects present in the image.

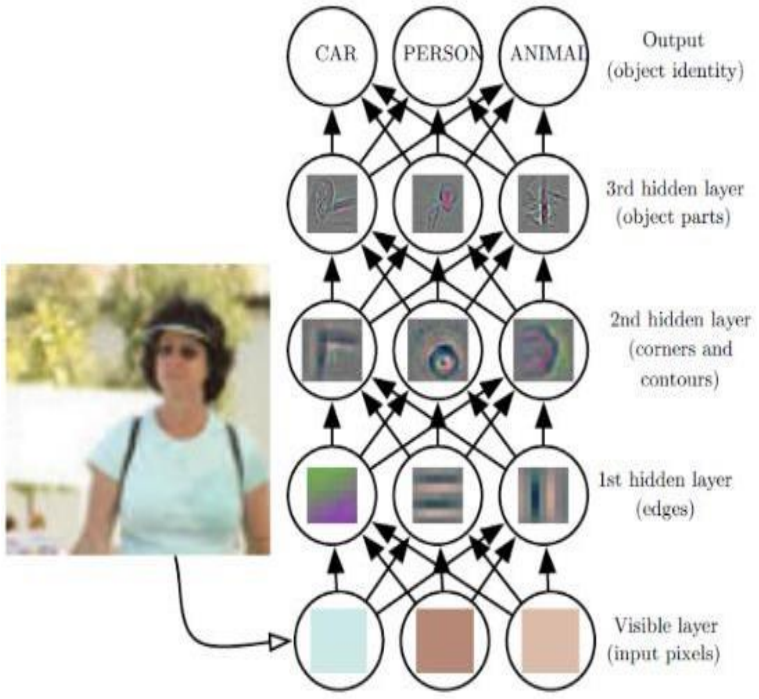


Fig.1.1 Illustration of deep learning model

There are two main ways of measuring the depth of a model. The first view is based on the number of sequential instructions that must be executed to evaluate the architecture. Another approach, used by deep probabilistic models, regards the depth of a model as being not the depth of the computational graph but the depth of the graph describing how concepts are related to each other. Machine learning is the only viable approach to building AI systems that can operate in complicated, real-world environments. Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts,with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. Fig. 1.2 illustrates the relationship between these different AI disciplines. Fig. 1.3 gives a high-level schematic of how each works.

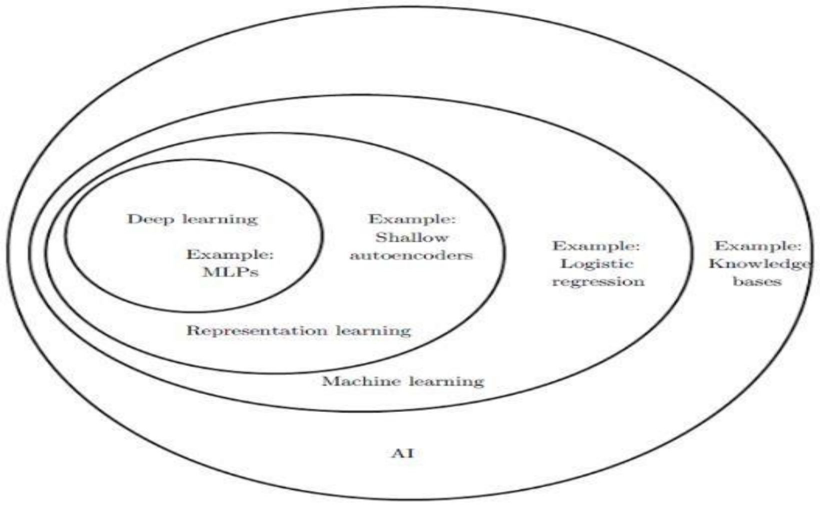


Fig.1.2 Venn diagram representing relationship between AI disciplines

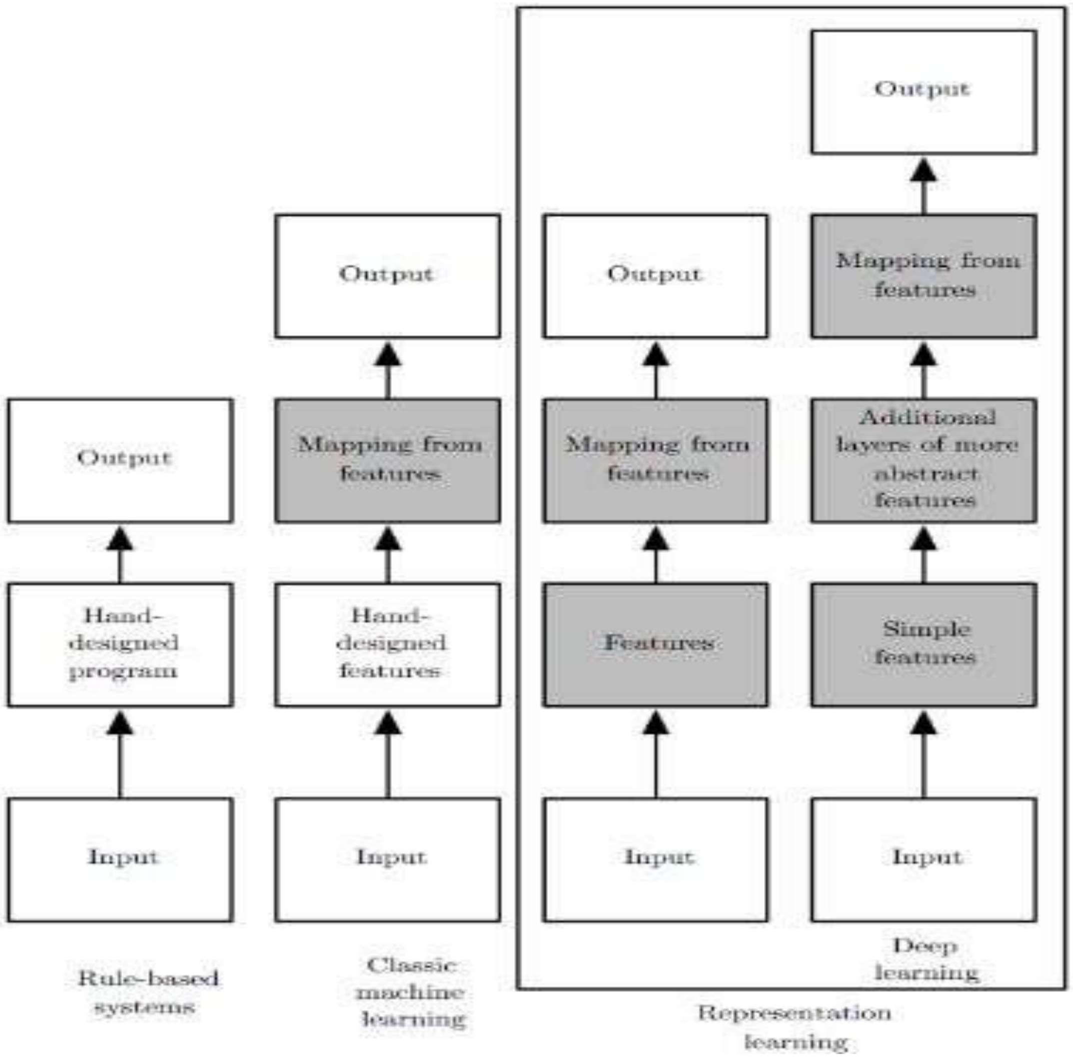


Fig.1.3 High level schematics representing relationship between AI disciplines

AI is basically the study of training your machine (computers) to mimic a human brain and its thinking capabilities.AI focuses on three major aspects (skills): learning, reasoning, and self-correction to obtain the maximum efficiency possible. **Machine Learning (ML) is an application or subset of AI. The major aim of ML is to allow the systems to learn by themselves through experience withoutany kind of human intervention or assistance.** Deep Learning(DL)is basically a sub-part of the broader familyof Machine Learning which makes use of **Neural Networks** (similar to the neurons working in our brain) to mimic human brain-like behavior. DL algorithms focus on **information processing patterns** mechanism to possibly identify the patterns just like our human brain does and classifies the information accordingly. DL works on larger sets of data when compared to ML and the **prediction mechanism is self-administered by machines**. The differences between AI, ML and DL are presented as Table 1 as below.

Table1.Difference between Artificial Intelligence, Machine Learning & Deep Learning

Artificial Intelligence	**Machine Learning**	**Deep Learning**
AI stands for Artificial Intelligence, and is basically the study/process which enables machines to mimic human behaviour through particular algorithm.	ML stands for Machine Learning, and is the study that uses statistical methods enabling machines to improve with experience.	DL stands for Deep Learning, and is the study that makes use of Neural Networks (similar to neuronspresentinhumanbrain) toimitatefunctionalityjustlike ahuman brain.
AI is the broader family consisting of ML and DL asit's components.	ML is the subset of AI.	DL is the subset of ML.
AI is a computer algorithm which exhibits intelligence through decision making.	ML is an AI algorithm which allows system to learn from data.	DL is a ML algorithm that uses deep (more than one layer) neuralnetworkstoanalyzedata andprovideoutput accordingly.
Search Trees and much complex math are involved in AI.	Having a clear idea about the logic (math) involved in behind and can visualize the complex functionalities like K-Mean, Support Vector Machines, etc., then it defines the ML aspect.	With clear about the math involved in it but don't have idea about the features, so one break the complex functionalities into linear/lower dimension features by adding

		morelayers,thenitdefinesth e DLaspect.
The aim is to basically increase chances of success and not accuracy.	Theaimistoincreaseaccuracyno t caring much about the success ratio.	It attains the highest rank in terms of accuracy when it is trainedwithlargeamountof data.
The efficiency of AI is basicallythe efficiencyprovidedbyMLandDL respectively.	LessefficientthanDLasitcan't workforlongerdimensionsor higher amount of data.	MorepowerfulthanMLasit caneasilyworkforlargersets of data.

Artificial Intelligence	**Machine Learning**	**Deep Learning**
Threebroadcategories/typesOf AI are: Artificial Narrow Intelligence (ANI),ArtificialGeneralIntelligence (AGI)andArtificialSuper Intelligence (ASI)	Threebroadcategories/typesof ML are: Supervised Learning, Unsupervised Learning and Reinforcement Learning	DLcanbeconsideredasneural networkswithalargenumber ofparameterslayerslyingin oneofthefourfundamental network architectures: Unsupervised Pre-trained Networks, Convolutional Neural Networks, Recurrent NeuralNetworksandRecursive NeuralNetworks
Examples of AIapplicationsinclude: Google's AI-Powered Predictions, RidesharingAppsLikeUberandLyft, Commercial Flights Use an AI Autopilot,etc.	Examples of ML applications include: Virtual Personal Assistants:Siri,Alexa,Google, etc.,EmailSpamandMalware Filtering.	ExamplesofDLapplications include:Sentimentbasednews aggregation,Imageanalysisand captiongeneration,etc.

2. Linear Algebra:

Agoodunderstandingoflinearalgebraisessentialforunderstandingandworkingwithmanymachine learning algorithms, especially deep learning algorithms.

Scalars,Vectors,Matrices and Tensors

- *Scalars*: A scalar is just a single number, in contrast to most of the other objects studied in linear algebra, which are usually arrays of multiple numbers. We write scalars in italics. We usually give scalars lower-case variable names. When we introduce them, we specify what kind of number they are. For example, we might say ―Let s ∈R be the slope of the line,‖ while defining a real-valued scalar, or ―Let n ∈N be the number of units,‖ while defining a natural number scalar.

- *Vectors*: A vector is an arrayof numbers. The numbers are arranged in order. We can identifyeach individual number by its index in that ordering. Typically we give vectors lower case names written in bold typeface, such as x. The elements of the vector are identified by writing its name in italic typeface, with a subscript. Thefirst element of x is x1, the second element is x2 and so on. We also need to saywhat kinds of numbers are stored in the vector. If each element is in R, and the vector has n elements, then the vector lies in the set formed bytakingthe Cartesian product of R n times, denoted as Rn. Whenweneed to explicitlyidentifythe elements of a vector, we write them as a column enclosed in square brackets:

We can think of vectors as identifying points in space, with each element giving the coordinate along a different axis. Sometimes we need to index a set of elements of a vector. In this case, we define a set containing the indices and write the set as a subscript. For example, to access x1, x3 and x6 , we define the set S = {1, 3, 6} and write x_S . We use the − sign to index the complement of a set. For example x_{-1} is the vector containing all elements of x except for x_1, and x_{-S} is the vector containing all of the elements of x except for x1, x3 and x6.

- *Matrices*: A matrix is a 2-D arrayof numbers, so each element is identified by two indices instead of just one. We usually give matrices upper-case variable names with bold typeface, such as A. If a real-valued matrix A has a height of m and a width of n, then we say that A $\in \mathbb{R}^{m\times n}$. We usually identify the elements of a matrix using its name in italic but not bold font, and the indices are listed with separating commas. For example, $A_{1,1}$ is the upper left entry of A and $A_{m,n}$ is the bottom right entry. We can identify all of the numbers with vertical coordinate i by writing a ―:‖ for the horizontal coordinate. For example, $A_{i,:}$ denotes the horizontal cross section of A with vertical coordinate i.This is known as the i-th*row*ofA. Likewise, $A_{:,i}$ is the i-th *column* of A. When we need to explicitly identify the elements of a matrix, we write them as an array enclosed in square brackets:

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}.$$

Sometimes we mayneed to index matrix-valued expressions that are not just a single letter. In this case, we use subscripts after the expression, but do not convert anything to lower case. For example, f $(A)_{i,j}$ gives element (i, j) of the matrix computed by applying the function f to A.

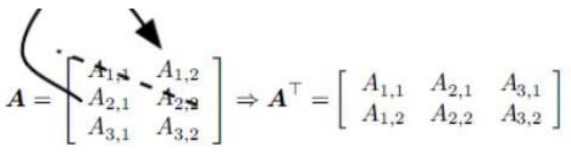


Figure 2.1: The transpose of the matrix can be thought of as a mirror image across the main diagonal.

- *Tensors*: In some cases we will need an array with more than two axes. In the general case, an array of numbers arranged on aregular grid with avariable numberofaxes is knownas a *tensor*. Wedenoteatensor named ―A‖ with this typeface: **A**. We identifythe element of **A** at coordinates (i, j, k) bywriting $A_{i,j,k}$. One important operation on matrices is the *transpose*.Thetransposeofamatrix is themirrorimageofthematrix across a diagonal line, called the *main diagonal*, running down and to the right, starting from its upper left corner.SeeFig. 2.1 fora graphical depiction ofthis operation. Wedenotethetransposeofamatrix Aas A^T, and it is defined such that

$$(\boldsymbol{A}^\top)_{i,j} = A_{j,i}.$$

Vectors can be thought of as matrices that contain onlyone column. The transpose of a vector is therefore a matrixwithonlyonerow.Sometimeswedefineavectorbywritingoutitselementsinthetextinlineasa rowmatrix,thenusingthetransposeoperatortoturnitintoastandardcolumnvector,e.g.,x=[x1,x2,x3 $]^T$.

A scalar can be thought of as a matrix with only a single entry. From this, we can see that a scalar is its own transpose: $a = a^T$. We can add matrices to each other, as long as they have the same shape, just by adding their corresponding elements: C = A +B where $C_{i,j} = A_{i,j} + B_{i,j}$.We can also add a scalar to a matrix or multiply a matrix by a scalar, just by performing that operation on each element of a matrix: $D = a \cdot B + c$ where $D_{i,j}= a \cdot B_{i,j}+ c$.

In the context of deep learning, we also use some less conventional notation. We allow the addition of matrix and a vector, yielding another matrix: C = A + b, where $C_{i,j}= A_{i,j}+ b_j$. In other words, the vector b is added to each row of the matrix. This shorthand eliminates the need to define a matrix with b copied into each row before doing the addition. This implicit copying of b to many locations is called *broadcasting*.

Probability Distributions

Probability can be seen as the extension of logic to deal with uncertainty. Logic provides a set of formal rules for determining what propositions are implied to true or false given the assumption that someother set of propositions is true or false. Probability theory provides a set of formal rules for determining the likelihood of a proposition being true given the likelihood of other propositions.

A *random variable* is a variable that can take on different values randomly. Random variables may be discrete or continuous. A discrete random variable is one that has a finite or countably infinite number of states. Note that these states are not necessarily the integers; they can also just be named states that are not considered to have any numerical value. A continuous random variable is associated with a real value.

A *probability distribution* is a description of how likely a random variable or set of random variables is to take on each of its possible states. The way we describe probability distributions depends on whether the variables are discrete or continuous.

Discrete Variables and Probability Mass Functions

Aprobabilitydistributionoverdiscretevariablesmaybedescribedusinga *probabilitymassfunction*(PMF). We typically denote probability mass functions with a capital P. Often we associate each random variable with a different probability mass function and the reader must infer which probability mass function to use based on the identity of the random variable, rather than the name of the function; P(x) is usually not the same as P(y).

The probability mass function maps from a state of a random variable to the probability of that random variable taking on that state. The probability that x = x is denoted as P (x),with a probabilityof 1 indicating that x = x is certain and a probability of 0 indicating that x = x is impossible. Sometimes to disambiguate which PMF to use, we write the name of the random variable explicitly: P (x = x). Sometimes we define a variable first, then use ~notation to specify which distribution it follows later: x ~ P(x).

Probability mass functions can act on many variables at the same time. Such a probability distribution over many variables is known as a*joint probability distribution*. P (x = x, y =y) denotes the probability that x =x and y = y simultaneously. We may also write P(x, y) for brevity. To be a probability mass function on a random variable x, a function P must satisfy the following properties:

- □$\forall x \in X$, $0 \leq P(x) \leq 1$. An impossible event has probability 0 and no state can be less probablethan that. Likewise,an event that is guaranteed to happen has probability1, and no state can have a greater chance of occurring.

- □$\sum_{x \in X} P(x) = 1$.We refers to this property as being *normalized*. Without this property, we could obtain probabilities greater than one by computing the probability of one of many events occurring.

Forexample, considerasingle discreterandom variable xwith k different states.We canplacea*uniform distribution* on x—that is, make each of its states equally likely—by setting its probabilitymass function to

$$P(\mathrm{x} = x_i) = \frac{1}{k}$$

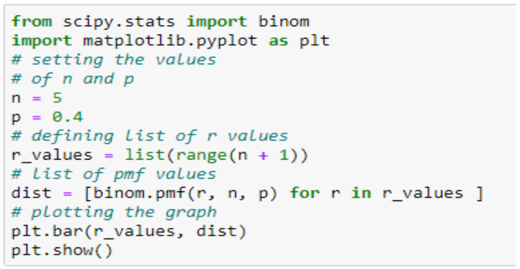
foralli.Wecanseethatthisfitstherequirementsforaprobabilitymassfunction.Thevalue$\frac{1}{k}$ ispositive becausekis apositiveinteger.Wealsoseethat

$$\sum_i P(\mathrm{x} = x_i) = \sum_i \frac{1}{k} = \frac{k}{k} = 1,$$

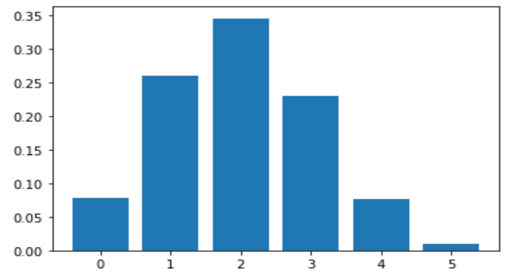
sothedistributionisproperlynormalized.Let‘sdiscussfewdiscreteprobabilitydistributionsas follows:

Binomial Distribution

The binomial distribution is a discrete distribution with a finite number of possibilities. When observing a series of what are known as Bernoulli trials, the binomial distribution emerges. A Bernoulli trial is a scientific experiment with only two outcomes: success or failure.
Consider a random experiment in which you toss a biased coin six times with a 0.4 chance of getting head.If'gettingahead'isconsidereda_success‘,thebinomialdistributionwillshowtheprobabilityofrsuccesses for each value of r.
The binomial random variable represents the number of successes (r) in n consecutive independentBernoulli trials.

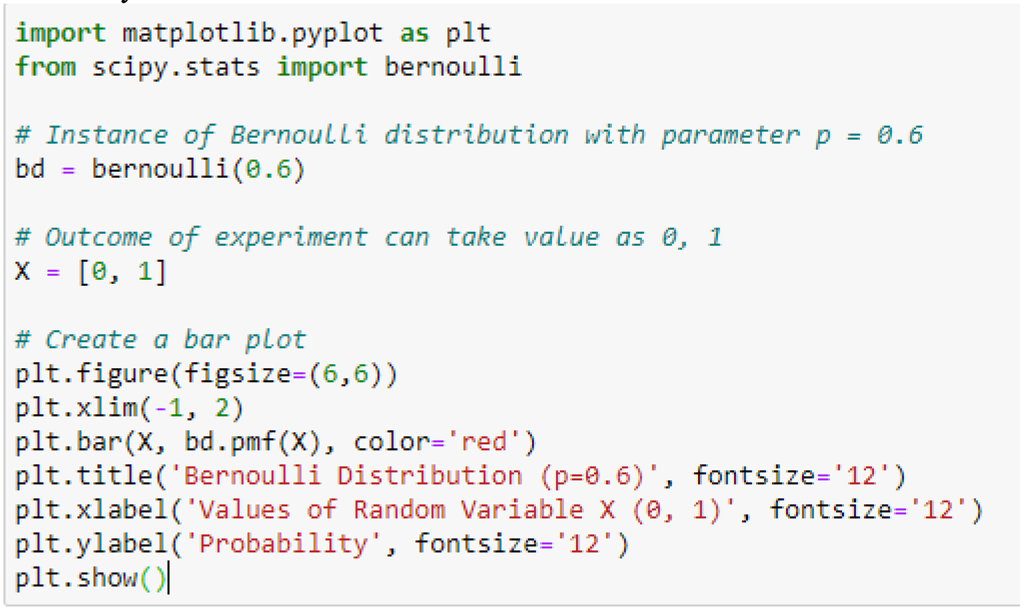


```python
from scipy.stats import binom
import matplotlib.pyplot as plt
# setting the values
# of n and p
n = 5
p = 0.4
# defining list of r values
r_values = list(range(n + 1))
# list of pmf values
dist = [binom.pmf(r, n, p) for r in r_values ]
# plotting the graph
plt.bar(r_values, dist)
plt.show()
```



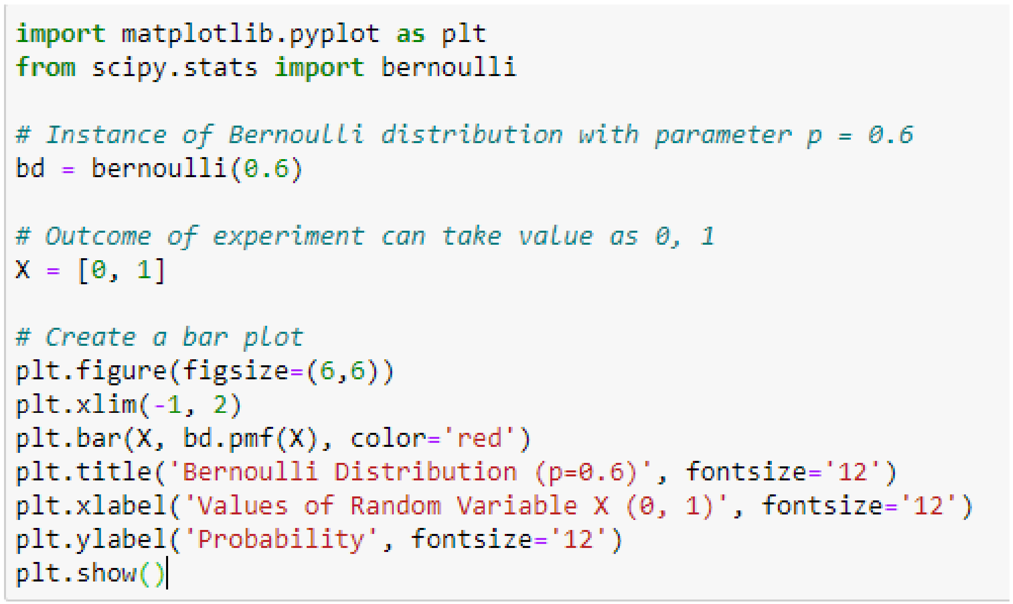
Bernoulli's Distribution

The Bernoulli distribution is a variant of the Binomial distribution in which only one experiment is conducted,resultinginasingleobservation.Asaresult, theBernoullidistributiondescribeseventsthathave exactly two outcomes.
Here‘saPythonCodetoshowBernoullidistribution:



```
import matplotlib.pyplot as plt
from scipy.stats import bernoulli

# Instance of Bernoulli distribution with parameter p = 0.6
bd = bernoulli(0.6)

# Outcome of experiment can take value as 0, 1
X = [0, 1]

# Create a bar plot
plt.figure(figsize=(6,6))
plt.xlim(-1, 2)
plt.bar(X, bd.pmf(X), color='red')
plt.title('Bernoulli Distribution (p=0.6)', fontsize='12')
plt.xlabel('Values of Random Variable X (0, 1)', fontsize='12')
plt.ylabel('Probability', fontsize='12')
plt.show()
```

TheBernoullirandomvariable'sexpectedvalueisp,whichisalsoknownastheBernoullidistribution's parameter. Theexperiment'soutcomecanbeavalueof0or1.Bernoullirandomvariablescanhavevaluesof0 or1. The pmf function is used to calculate the probability of various random variable values.



```
import matplotlib.pyplot as plt
from scipy.stats import bernoulli

# Instance of Bernoulli distribution with parameter p = 0.6
bd = bernoulli(0.6)

# Outcome of experiment can take value as 0, 1
X = [0, 1]

# Create a bar plot
plt.figure(figsize=(6,6))
plt.xlim(-1, 2)
plt.bar(X, bd.pmf(X), color='red')
plt.title('Bernoulli Distribution (p=0.6)', fontsize='12')
plt.xlabel('Values of Random Variable X (0, 1)', fontsize='12')
plt.ylabel('Probability', fontsize='12')
plt.show()
```

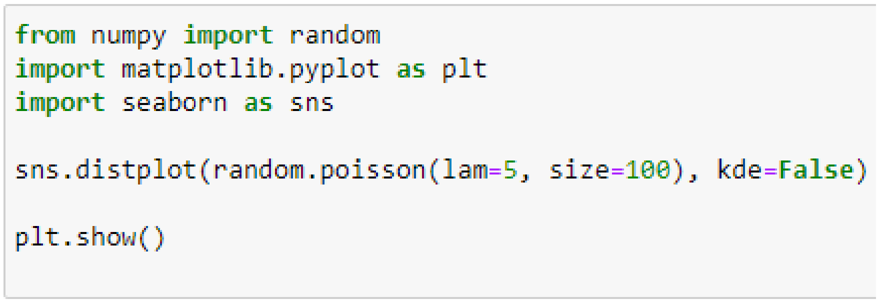
Poisson Distribution

A Poisson distribution is a probability distribution used in statistics to show how many times an event is likely to happen over a given period of time. To put it another way, it's a count distribution. Poisson distributions are frequently used to comprehend independent events at a constant rate over a given time interval. Siméon Denis Poisson, a French mathematician, was the inspiration for the name.

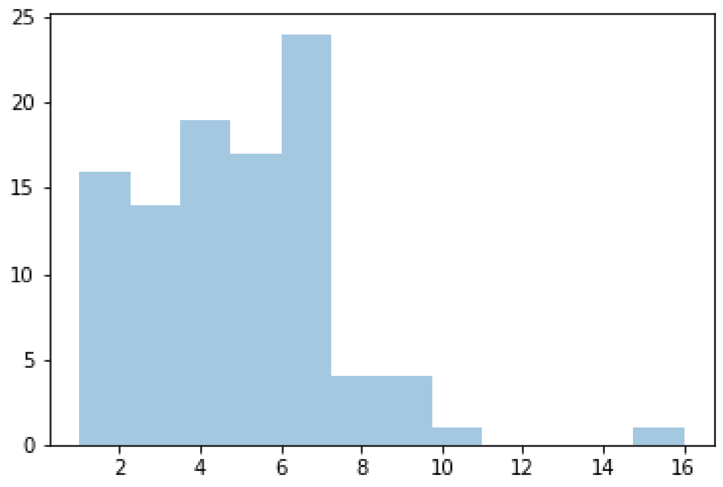
ThePythoncodebelowshowsasimpleexampleofPoissondistribution.Ithastwo parameters:

1. Lam:Knownnumberofoccurrences
2. Size:Theshapeofthereturnedarray

Thebelow-givenPythoncodegeneratesthe1x100distributionforoccurrence5.



```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.poisson(lam=5, size=100), kde=False)

plt.show()
```



Continuous Variables and Probability Density Functions

When working with continuous random variables, we describe probability distributions using a *probability density function (PDF)*rather than a probability mass function. To be a probability density function, a function p must satisfy the following properties:

• Thedomainofmustbe the setof pallpossible statesofx.

• $\forall x \in X$,P(x)$\geq$ 0. Notethatwedonot requirep(x) $\leq$ 1.

• $\int p(x)dx=1$.

A probability density function p(x) does not give the probability of a specific state directly, instead the probability of landing inside an infinitesimal region with volume δx is given by p(x)δx.

We can integrate the density function to find the actual probability mass of a set of points. Specifically, the probabilitythatxliesinsomesetSisgivenbytheintegralofp(x)overthatset.Intheunivariateexample, theprobabilitythat xliesin theinterval[a, b] isgiven by $\int_{[a,b]} p(x)dx$.

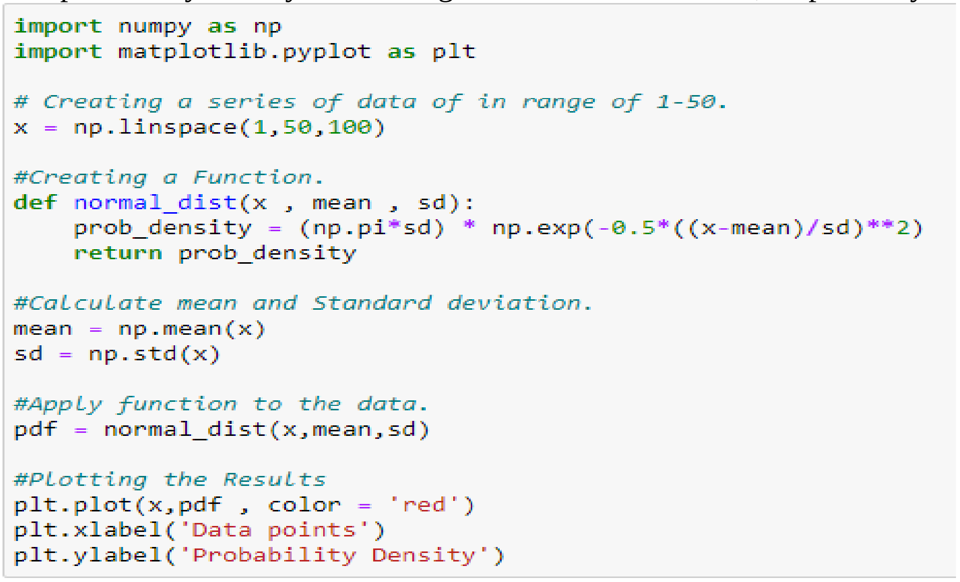
For an example of a probability density function corresponding to a specific probability density over a continuous random variable, consider a uniform distribution on an interval of the real numbers. We can do this with a function u(x; a, b), where a and b are the endpoints of the interval, with b > a. The ―;‖ notation means―parametrizedby‖;weconsiderxtobetheargumentofthefunction,whileaandbareparameters

.

Normal Distribution

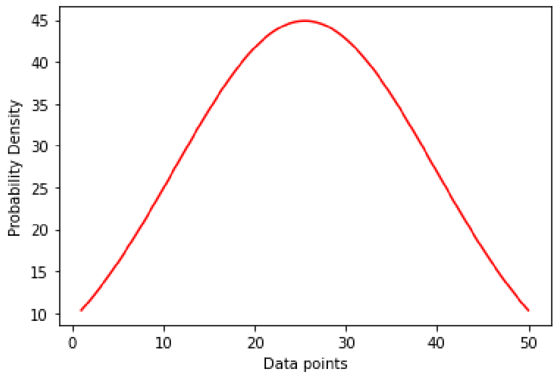
Normal Distribution is one of the most basic continuous distribution types. Gaussian distribution is another name for it. Around its mean value, this probability distribution is symmetrical. It also demonstrates that data close to the mean occurs more frequently than data far from it. Here, the mean is 0, and the variance isa finite value.

In the example, you generated100 random variables rangingfrom 1 to 50. Afterthat, you created a function to define the normal distribution formula to calculate the probability density function. Then, you have plotted the data points and probability density function against X-axis and Y-axis, respectively.



```
import numpy as np
import matplotlib.pyplot as plt

# Creating a series of data of in range of 1-50.
x = np.linspace(1,50,100)

#Creating a Function.
def normal_dist(x , mean , sd):
    prob_density = (np.pi*sd) * np.exp(-0.5*((x-mean)/sd)**2)
    return prob_density

#Calculate mean and Standard deviation.
mean = np.mean(x)
sd = np.std(x)

#Apply function to the data.
pdf = normal_dist(x,mean,sd)

#Plotting the Results
plt.plot(x,pdf , color = 'red')
plt.xlabel('Data points')
plt.ylabel('Probability Density')
```

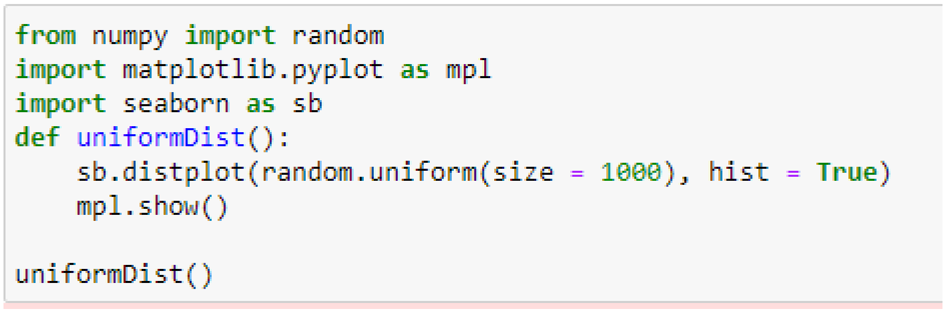
```
Text(0, 0.5, 'Probability Density')
```



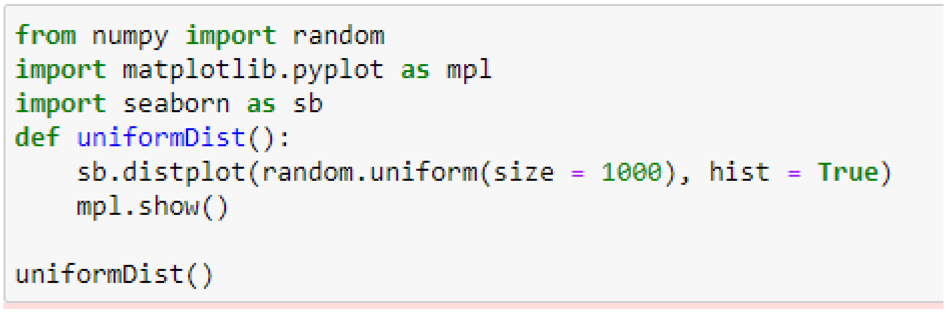
Continuous Uniform Distribution

In continuous uniform distribution, all outcomes are equally possible. Each variable has the same chance of being hit as a result. Random variables are spaced evenly in this symmetric probabilistic distribution, with a 1/ (b-a) probability.

The below Python code is a simple example of continuous distribution taking 1000 samples of random variables.



```python
from numpy import random
import matplotlib.pyplot as mpl
import seaborn as sb
def uniformDist():
    sb.distplot(random.uniform(size = 1000), hist = True)
    mpl.show()

uniformDist()
```



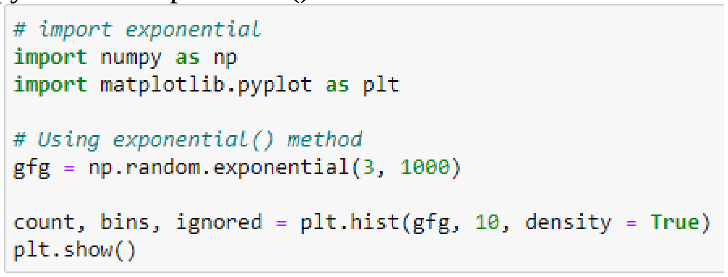
```python
from numpy import random
import matplotlib.pyplot as mpl
import seaborn as sb
def uniformDist():
    sb.distplot(random.uniform(size = 1000), hist = True)
    mpl.show()

uniformDist()
```

Log-Normal Distribution

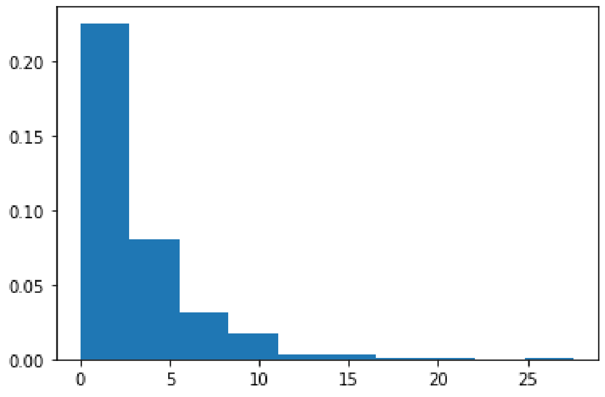
Therandomvariableswhoselogarithmvaluesfollowanormaldistributionareplottedusingthis distribution.TakealookattherandomvariablesXandY.ThevariablerepresentedinthisdistributionisY =ln(X),wherelndenotesthenaturallogarithmofXvalues.
Thesizedistributionofraindroplets canbeplottedusinglognormal distribution.

Exponential Distribution

In a Poisson process, an exponential distribution is a continuous probability distribution that describes thetime between events (success, failure, arrival, etc.).
Youcan seeinthebelow examplehowto getrandomsamplesofexponentialdistributionandreturnNumpy array samples by using the numpy.random.exponential() method.



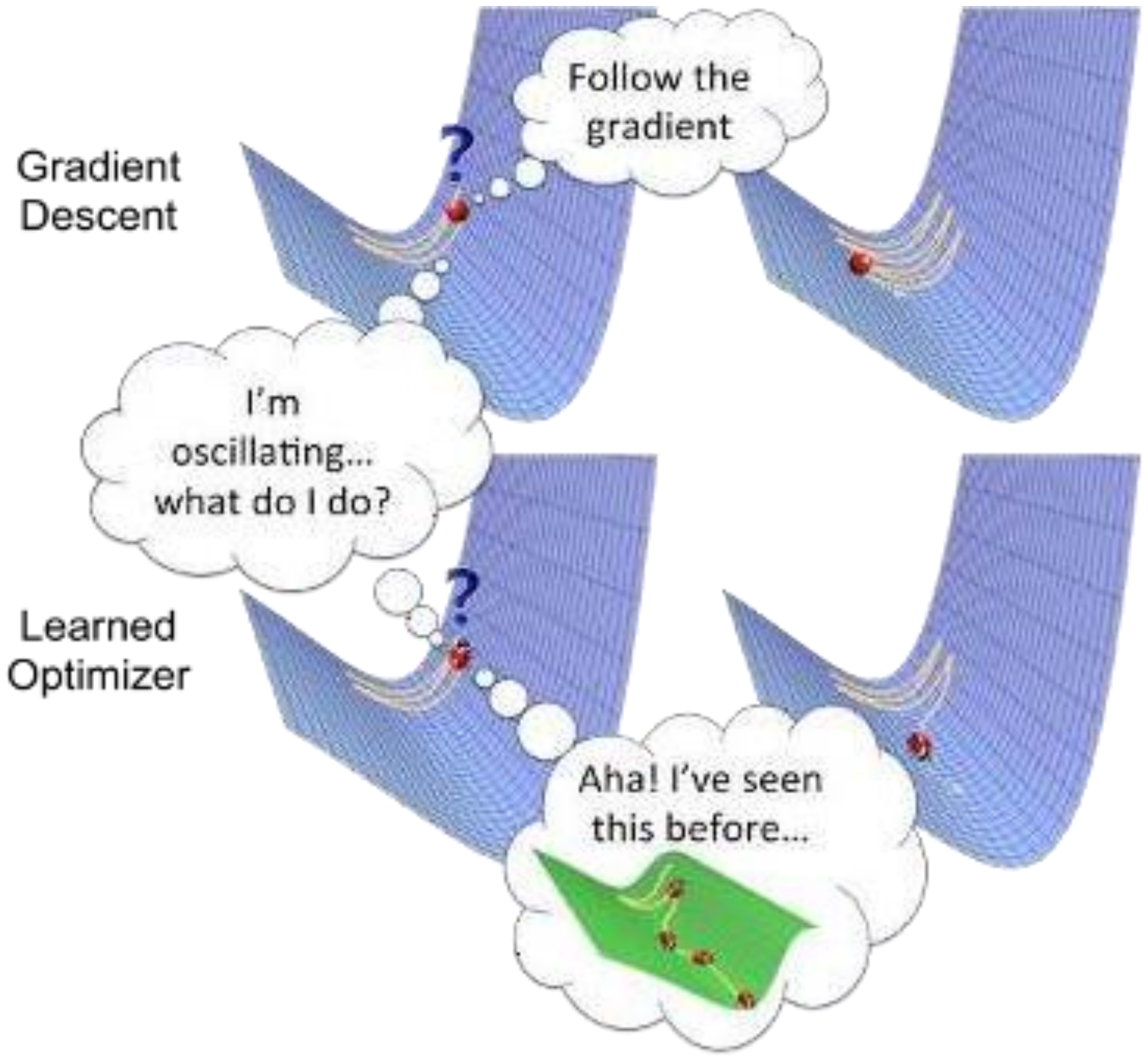
```python
# import exponential
import numpy as np
import matplotlib.pyplot as plt

# Using exponential() method
gfg = np.random.exponential(3, 1000)

count, bins, ignored = plt.hist(gfg, 10, density = True)
plt.show()
```



4. Gradient based Optimization

Optimization means minimizing or maximizing any mathematical expression. Optimizers are algorithms or methods used to update the parameters of the network such as weights, biases, etc. to minimize the losses. Therefore, Optimizers are used to solve optimization problems byminimizing the function i.e, loss function in the case of neural networks.

Here, we‘re going to explore and deep dive into the world of optimizers for deep learning models. We will also discuss the foundational mathematics behind these optimizers and discuss their advantages, and disadvantages.


Gradient Descent
Follow the gradient
I'm oscillating... what do I do?
Learned Optimizer
Aha! I've seen this before...

Roleofan Optimization

As discussed above, optimizers update the parameters of neural networks such as weights and learning rate to minimize the loss function. Here, the loss function acts as a guide to the terrain telling optimizer if it is moving in the right direction to reach the bottom of the valley, the global minimum.

The Intuition behind Optimization

Let us imagine a climber hiking down the hill with no sense of direction. He doesn‘t know the right way to reach the valley in the hills, but, he can understand whether he is moving closer (going downhill) or further away (uphill) from his final destination. If he keeps taking steps in the correct direction, he will reach to his aim i.,e the valley.

Batch Gradient Descent

Gradient descent algorithm is an **optimization algorithm**which is used to minimize the function. The functionwhichissettobeminimizediscalledasan **objectivefunction**.For **machinelearning**, the**objective function**is also termed as the**cost function** or**loss function**. It is the loss function which is optimized (minimized) and gradient descent is used to find the most optimal value of parameters / weights which minimizes the loss function. **Loss function, simply speaking, is the measure of the squareddifference between actual values and predictions.** In order to minimize the objective function, the most optimal value of **the parameters of the function** from large or infinite parameter space are found.

Gradient of a function at any point is the direction of steepest increase or ascent of the function atthat point.

Based on above,**the gradient descent of a function at any point, thus, represent the direction ofsteepest decrease or descent of function at that point**.

Inorderto **findthegradient**ofthefunctionwithrespecttoxdimension,takethe **derivativeof thefunction**with respect to x , then substitute the x-coordinate of the point of interest in for the x values in the derivative. Once gradient of the function at any point is calculated, the gradient descent can becalculated by multiplying the gradient with -1. Here are the steps of finding minimum of the function using gradient descent:

- Calculate the gradient by taking the derivative of the function with respect to the specific parameter. In case, there are multiple parameters, take the partial derivatives with respect to different parameters.
- Calculate the descent value for different parameters by multiplying the value of derivatives with learning or descent rate (step size) and -1.
- Update the value of parameter by adding up the existing value of parameter and the descent value. The diagram below represents the updation of parameter [latex]\theta[/latex] with the value of gradient in the opposite direction while taking small steps.

Gradientdescentisanoptimizationalgorithmthat‘susedwhentrainingdeeplearningmodels.It‘s basedon aconvexfunctionandupdatesitsparametersiterativelytominimizeagiven functiontoitslocalminimum.

The notation used in the above Formula is given below,

- α isthelearningrate,
- J isthecostfunction,and
- Ɵistheparameterto beupdated.

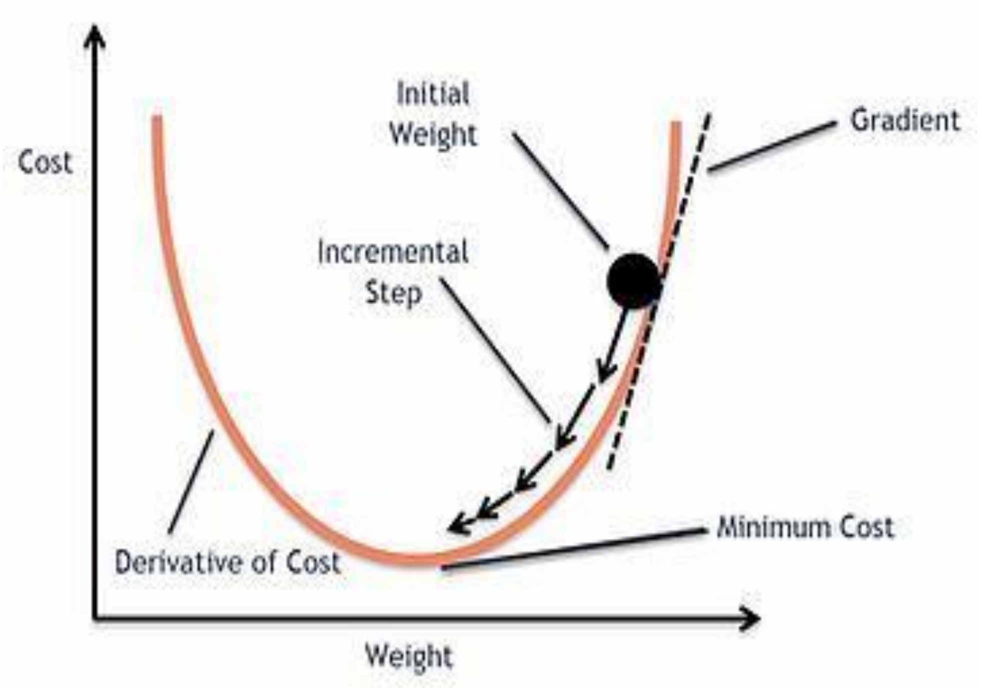
Aswesee,thegradientrepresentsthepartialderivativeofJ(costfunction) withrespecttoƟ$_j$

Note that, as we reach closer to the global minima, the slope(gradient) of the curve becomes less and less steep, which results in a smaller value of derivative, which in turn reduces the step size(learning rate) automatically.

It is themost basicbut most usedoptimizerthat directlyusesthederivative oftheloss function and learning rate to reduce the loss function and tries to reach the global minimum.

- Linear Regression,
- Classification Algorithms,
- Back-propagation in Neural Networks,etc.

The above-described equation calculates the gradient of the cost function J(θ) with respect to the networkparameters θ for the entire training dataset:



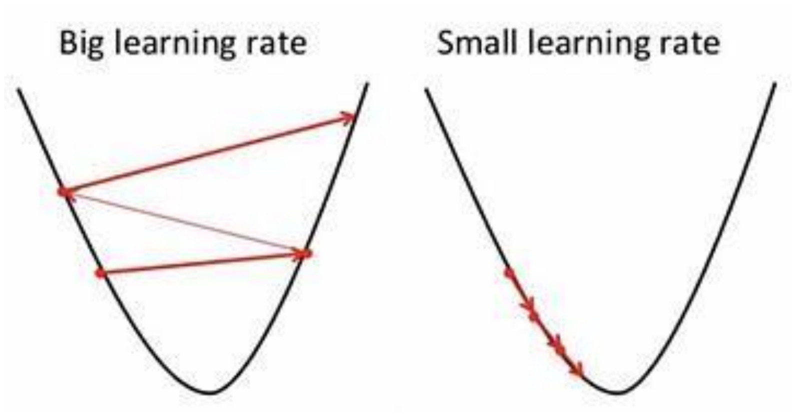
Our aim is to reach at the bottom of the graph (Cost vs weight), or to a point where we can no longer move downhill–a local minimum.

➢ Role of Gradient

In general, Gradient represents the slope of the equation while gradients are partial derivatives and they describe the change reflected in the loss function with respect to the small change in parameters of the function. Now, this slight change in loss functions can tell us about the next step to reduce the output of the loss function.

➢ Role of Learning Rate

Learning rate represents the size of the steps our optimization algorithm takes to reach the global minima.To ensure that the gradient descent algorithm reaches the local minimum we must set the learning rate to an appropriate value, which is neither too low nor too high.
Takingverylargestepsi.e,alarge value ofthelearningrate mayskipthe global minima,andthemodelwill never reach the optimal value for the loss function. On the contrary, taking very small steps i.e, a smallvalue of learning rate will take forever to converge.



.

As we discussed, the gradient represents the direction of increase. But our aim is to find the minimum point in the valley so we have to go in the opposite direction of the gradient. Therefore, we update parameters in the negative gradient direction to minimize the loss.

Algorithm:θ=θ−α⋅∇J(θ)
Incode,BatchGradientDescentlookssomethinglikethis:
forx in range(epochs):
params_gradient=find_gradient(loss_function,data,parameters)
parameters=parameters-learning_rate*params_gradient

➢ Advantages of Batch GradientDescent

- Easycomputation
- Easyto implement
- Easyto understand

➢ Disadvantage of Batch Gradient Descent

- Maytrapatlocal minima
- Weightsarechangedaftercalculatingthegradientonthewholedataset.So,ifthedatasetistoo large then this may take years to converge to the minima

Stochastic Gradient Descent

To overcome some of the disadvantages of the GD algorithm, the SGD algorithm comes into the picture as an extension of the Gradient Descent. One of the disadvantages of the Gradient Descent algorithm is that it requiresalotofmemorytoloadtheentiredatasetatatimetocomputethederivativeofthelossfunction.

So, In the SGD algorithm, we compute the derivative by taking one data point at a time i.e, tries to update the model‘s parameters more frequently. Therefore, the model parameters areupdated after the computation of loss on each training example.
So, let‘s have a dataset that contains 1000 rows, and when we apply SGD it will update the model parameters 1000 times in one complete cycle of a dataset instead of one time as in Gradient Descent.

Wewantthetraining,evenmore,faster,sowetakeaGradientDescentstepforeachtrainingexample. Let‘s seetheimplicationsintheimage below:

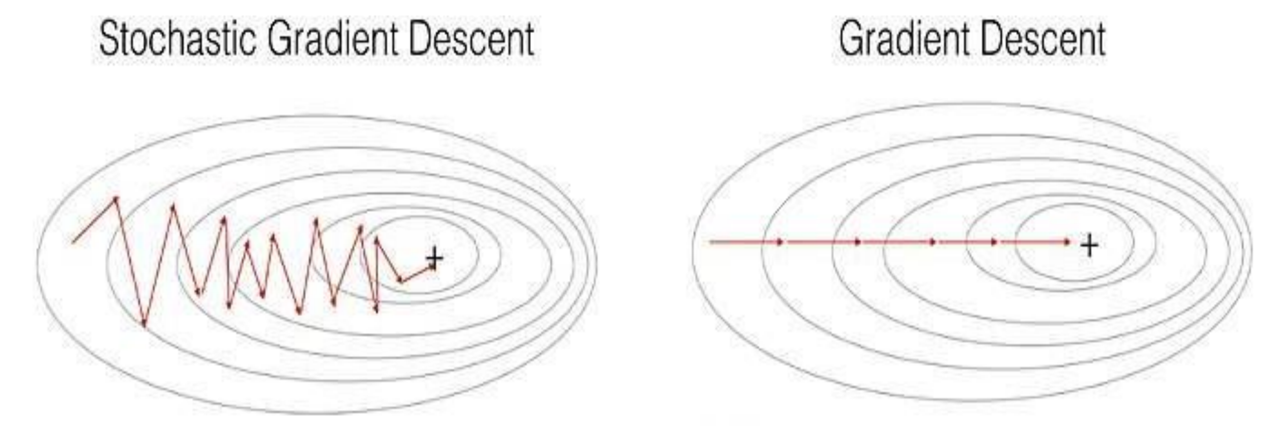


Figure 1 : **SGD vs GD**
"+" denotes a minimum of the cost. SGD leads to many oscillations to reach convergence. But each step is a lot faster to compute for SGD than for GD, as it uses only one training example (vs. the whole batch for GD).

- In the left diagram of the above picture, we have SGD (where 1 per step time) we take a Gradient Descent step for each example and on the right diagram is GD(1 step per entire training set).

- SGD seems to be quite noisy, but at the same time it is much faster than others and also it might be possible that it not converges to a minimum.

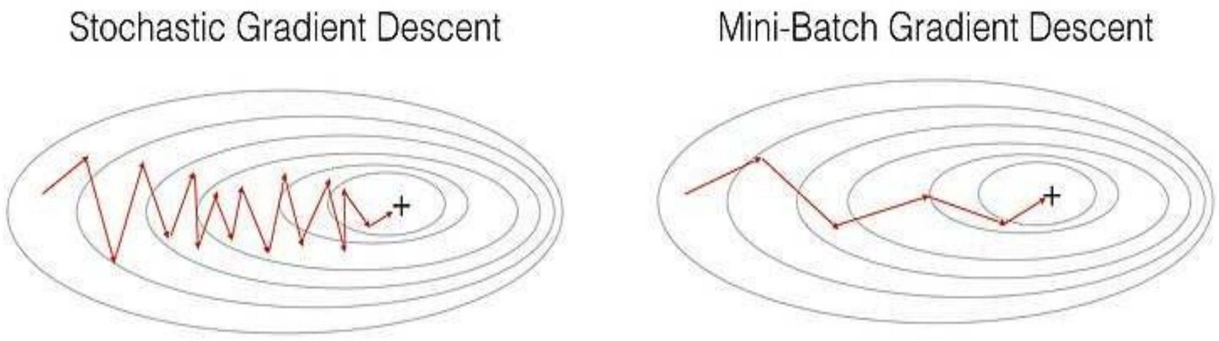
It is observed that in SGD the updates take more iteration compared to GD to reach minima. On the contrary, the GD takes fewer steps to reach minima but the SGD algorithm is noisier and takes more iterations as the model parameters are frequently updated parameters having high variance and fluctuations in loss functions at different values of intensities.
Its code snippet simplyadds a loop over thetraining examples and finds the gradient with respect to each of the training examples.

```
forx in range(epochs):
np.random.shuffle(data)
forexampleindata:
    params_gradient=find_gradient(loss_function,example,parameters)
parameters=parameters-learning_rate*params_gradient
```

- To reach the same convergence as that of gradient descent, we need to slowlyreduce the value of the learning rate

Mini-Batch Gradient Descent

To overcome the problem of large time complexity in the case of the SGD algorithm. MB-GD algorithm comes into the picture as an extension of the SGD algorithm. It‘s not all but it also overcomes the problem of Gradient descent. Therefore, It‘s considered the best among all the variations of gradient descent algorithms. MB-GD algorithm takes a batch of points or subset of points from the dataset to compute derivate.



It is observed that the derivative of the loss function for MB-GD is almost the same as a derivate of the loss function for GD after some numberofiterations. But the numberof iterations to achieve minima is large for MB-GD compared to GD and the cost of computation is also large.
Therefore, the weight updation is dependent on the derivate of loss for a batch of points. The updates in the case of MB-GD are much noisy because the derivative is not always towards minima.
It updatesthe model parametersafter everybatch. So, this algorithm divides thedataset into various batches and after every batch, it updates the parameters.

Algorithm:θ=θ−α⋅∇J(θ;B(i)),where{B(i)}arethebatchesoftrainingexamples

nthecodesnippet,insteadofiteratingoverexamples,wenowiterateovermini-batchesofsize30:

forx in range(epochs):
np.random.shuffle(data)
forbatchinget_batches(data,batch_size=30):
params_gradient=find_gradient(loss_function,batch,parameters)
parameters=parameters-learning_rate*params_gradient

Disadvantages of Mini Batch Gradient Descent

- The parameter updating in MB-SGD is much noisy compared to the weight updating in the GD algorithm
- ComparedtotheGDalgorithm,ittakesalongertimetoconverge
- Maygetstuckatlocal minima

4.3.5 Challenges with all types of Gradient-based Optimizers

Optimum Learning Rate: If we choose the learning rate as a too-small value, then gradient descent may takeaverylongtimetoconverge. Formoreabout thischallenge,refertotheabovesectionof LearningRate which we discussed in the Gradient Descent Algorithm.

Constant Learning Rate: For all the parameters, they have a constant learning rate but there may be some parameters that we may not want to change at the same rate.

Localminimum: Maygetstuckatlocalminimai.e.,notreachuptothelocal minimum.

5. Basics in Machine Learning

Need for machine learning:

Machine learning is important because it allows computers to learn from data and improve theirperformance on specific tasks without being explicitly programmed. This ability to learn from data and adapttonewsituationsmakesmachinelearningparticularlyusefulfortasksthatinvolvelargeamounts of data, complex decision-making, and dynamic environments.

- **Predictive modeling:** Machine learning can be used to build predictive models that can help businesses make better decisions. For example, machine learning can be used to predict which customersaremostlikelytobuyaparticularproduct,orwhichpatientsaremostlikelytodevelop a certain disease.
- **Natural language processing:** Machine learning is used to build systems that can understand and interpret human language. This isimportant for applications such as voice recognition, chatbots,and language translation.
- **Computervision:** Machinelearningisusedtobuildsystemsthatcanrecognizeandinterpret imagesandvideos.Thisisimportantforapplicationssuchasself-drivingcars,surveillance systems, and medical imaging.
- **Fraud detection:** Machine learning can be used to detect fraudulent behavior in financialtransactions, online advertising, and other areas.
- **Recommendation systems:**Machine learning can be used to build recommendation systems that suggestproducts,services,orcontenttousersbasedontheir pastbehaviorandpreferences.

Overall, machine learning has become an essential tool for many businesses and industries, as it enables them to make better use of data, improve their decision-making processes, and deliver more personalized experiences to their customers.

Definition and Workflow:

Machine Learning is a branch of artificial intelligence that develops algorithms by learning the hidden patterns of the datasets used it to make predictions on new similar type data, without being explicitly programmed for each task.

MachineLearningworksinthefollowingmanner.

- **Forward Pass:** IntheForwardPass,themachinelearningalgorithmtakesininputdataand producesanoutput. Depending onthe modelalgorithm itcomputes the predictions.
- **Loss Function:** The loss function, also known as the error or cost function, is used to evaluate the accuracy of the predictions made by the model. The function compares the predicted output of the modeltotheactualoutputandcalculatesthedifferencebetweenthem.Thisdifferenceisknown

as error or loss. The goal of the model is to minimize the error or loss function by adjusting its internal parameters.

- **Model Optimization Process:** The model optimization process is the iterative process of adjustingthe internal parameters of the model to minimize the error or loss function. This is done using an optimization algorithm, such as **gradient descent**. The optimization algorithm calculates thegradient of the error function with respect to the model‘s parameters and uses this information to adjust the parameters to reduce the error. The algorithm repeats this process until the error is minimized to a satisfactory level.

Oncethemodelhasbeentrainedandoptimizedonthetrainingdata,itcanbeusedtomakepredictions onnew,unseendata.Theaccuracyofthemodel‘spredictionscanbeevaluatedusingvarious performance metrics, such as accuracy, precision, recall, and F1-score.

Machine Learning life cycle:

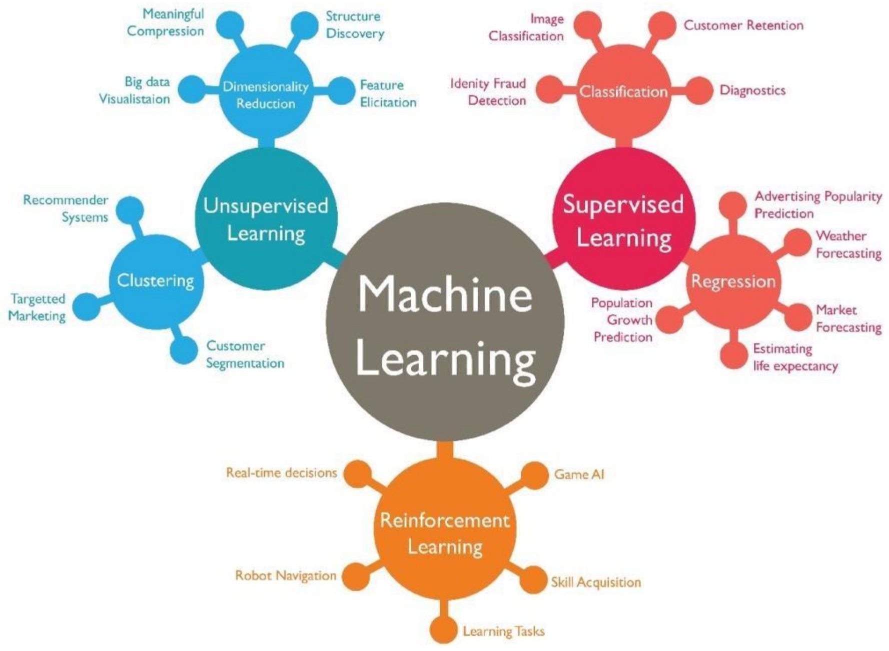
Thelifecycleofamachinelearningprojectinvolvesaseriesofstepsthatinclude:

1. **Study the Problems:** The first step is to study the problem. This step involves understanding the business problem and defining the objectives of the model.
2. **Data Collection:** When the problem is well-defined, we can collect the relevant data required forthe model.Thedatacouldcomefromvarioussourcessuchasdatabases,APIs,orwebscraping.
3. **Data Preparation:** When our problem-related data is collected. then it is a good idea to check the data properly andmakeitinthe desiredformatsothatitcanbe usedby the modeltofindthe hidden patterns. This can be done in the following steps:
   - Data cleaning
   - Data Transformation
   - Explanatory Data Analysis and Feature Engineering
   - Splitthedatasetfortrainingandtesting.
4. **Model Selection:** The next step is to select the appropriate machine learning algorithm that is suitableforourproblem.Thissteprequiresknowledgeofthestrengthsandweaknessesof different algorithms. Sometimes we use multiple models and compare their results and select thebest model as per our requirements.
5. **ModelbuildingandTraining:**Afterselectingthealgorithm,wehavetobuildthemodel.
   a. Inthecaseoftraditionalmachinelearningbuildingmodeiseasyitisjustafew hyperparameter tunings.
   b. Inthecaseofdeeplearning,wehavetodefinelayer-wisearchitecturealongwithinputand outputsize,numberofnodesineachlayer, lossfunction,gradient descent optimizer,etc.
   c. Afterthatmodelistrainedusingthepreprocesseddataset.
6. **Model Evaluation:** Once the model is trained, it can be evaluated on the test dataset to determineits accuracy and performance using different techniques like classification report, F1 score, precision, recall, ROC Curve, Mean Square error, absolute error, etc.
7. **Model Tuning:** Based on the evaluation results, the model may need to be tuned or optimized to improve its performance. Thisinvolvestweaking the hyperparameters ofthe model.
8. **Deployment:** Oncethemodelistrainedandtuned,itcanbedeployedinaproduction environment to make predictions on new data. This step requires integrating the model into an existing software system or creating a new system for the model.

9. **Monitoring and Maintenance:** Finally, it is essential to monitor the model‘s performance in the production environment and perform maintenance tasks as required. This involves monitoring for datadrift,retraining themodelasneeded,andupdatingthemodelasnewdatabecomesavailable.

Types of Machine Learning

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Reinforcement Machine Learning



Supervised Machine Learning:
Supervisedlearningisatypeofmachinelearninginwhichthealgorithmistrainedonthelabeled dataset.Itlearnstomapinputfeaturestotargetsbasedonlabeledtrainingdata.Insupervisedlearning, the algorithm is provided with input features and corresponding output labels, and it learns to generalize from this data to make predictions on new, unseen data.

- **Regression**: Regression is a type of supervised learning where the algorithm learns to predict continuous values based on input features. The output labels in regression are continuous values, such as stock prices, and housing prices. The different regression algorithms in machine learningare: Linear Regression, Polynomial Regression, Ridge Regression, Decision Tree Regression, Random Forest Regression, Support Vector Regression, etc
- **Classification**: Classification is a type of supervised learning where the algorithm learns to assign inputdatatoaspecificcategoryorclassbasedoninputfeatures.Theoutputlabelsin classificationarediscretevalues.Classificationalgorithmscanbebinary,wheretheoutputisone oftwopossibleclasses,ormulticlass,wheretheoutputcanbeoneofseveralclasses.The different Classification algorithms in machine learning are: Logistic Regression, Naive Bayes, Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), etc

Unsupervised Machine Learning:

Unsupervised learning is a typeof machine learning where the algorithm learns to recognize patterns indata without being explicitly trained using labeled examples. The goal of unsupervised learning is to discover the underlying structure or distribution in the data.

- **Clustering**: Clustering algorithms group similar data points together based on their characteristics.Thegoalistoidentifygroups,orclusters,ofdatapointsthataresimilartoeachother,while beingdistinctfromothergroups.SomepopularclusteringalgorithmsincludeK-means, Hierarchical clustering, and DBSCAN.
- **Dimensionality reduction:**Dimensionalityreductionalgorithmsreducethenumberofinput variables ina dataset while preserving as muchof the original information as possible. This isusefulforreducingthecomplexityofadatasetandmakingiteasiertovisualizeandanalyze. SomepopulardimensionalityreductionalgorithmsincludePrincipalComponentAnalysis(PCA), t-SNE, and Autoencoders.

Reinforcement Machine Learning

Reinforcementlearningisatypeofmachinelearningwhereanagentlearnstointeractwithan environment by performing actions and receiving rewards or penalties based on its actions. The goal of reinforcement learning is to learn a policy, which is a mapping from states to actions, that maximizes the expected cumulative reward over time.

- **Model-based reinforcement learning:**In model-based reinforcement learning, the agent learns a model of the environment, including the transition probabilities between states and the rewards associatedwitheachstate-actionpair.Theagentthenusesthismodeltoplanitsactionsinorder to maximize its expected reward. Some popular model-based reinforcement learning algorithms include Value Iteration and Policy Iteration.
- **Model-freereinforcementlearning**:Inmodel-freereinforcementlearning,theagentlearnsa policydirectlyfromexperiencewithoutexplicitlybuildingamodeloftheenvironment.The agentinteractswiththeenvironmentandupdatesitspolicybasedontherewardsitreceives. Some popular model-free reinforcement learning algorithms include Q-Learning, SARSA, andDeep Reinforcement Learning.

Capacity

The capacity of a network refers to the range of the types of functions that the model can approximate. Informally, a model‘s capacity is its ability to fit a wide variety of functions. A model with less capacitymay not be able to sufficiently learn the training dataset.

A model with more capacity can model more different types of functions and may be able to learn afunction to sufficiently map inputs to outputs in the training dataset. Whereas a model with too much capacity may memorize the training dataset and fail to generalize or get lost or stuck in the search for a suitablemappingfunction.Generally,wecanthinkofmodel capacityas a controloverwhetherthe modelis likely to underfit or overfit a training dataset.

Thecapacityof aneuralnetworkcanbecontrolled bytwo aspects ofthe model:

- Number of Nodes
- Number of Layers

A model with more nodes or more layers has a greater capacity and, in turn, is potentially capable of learning a larger set of mapping functions. A model with more layers and more hidden units per layer has higher **representational capacity**; it is capable of representing more complicated functions.

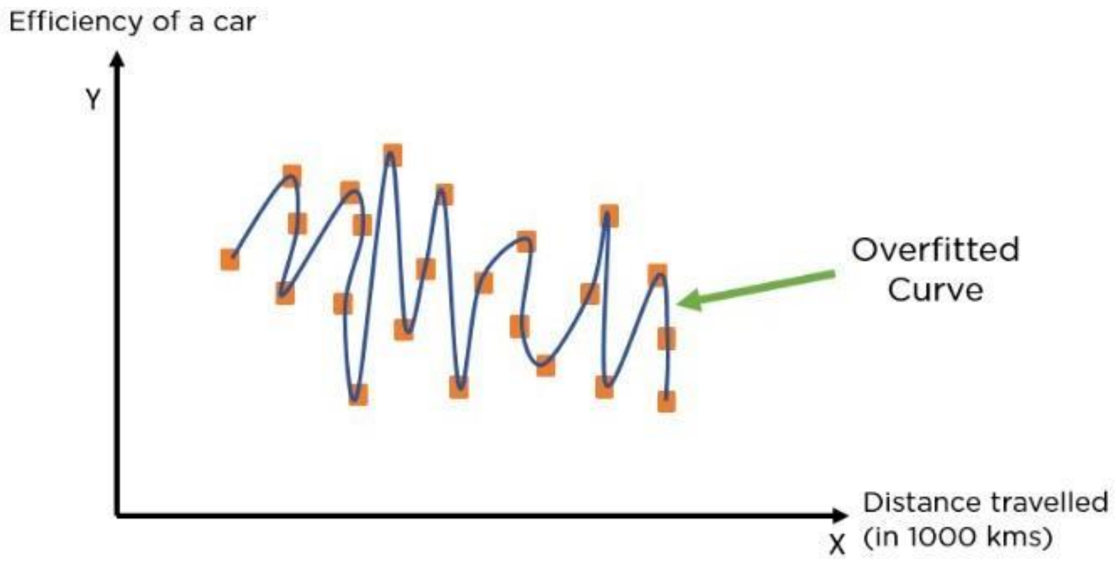
The number of nodes in a layer is referred to as the **width** and the number of layers in a model is referred to as its**depth**. Increasing the depth increases the capacityof the model. Training deep models, e.g. those with many hidden layers, can be computationally more efficient than training a single layer network with a vast number of nodes.

Over-fittingandunder-fitting

Over-fitting and under-fitting are two crucial concepts in machine learning and are the prevalent causes for the poor performance of a machine learning model. In this topic we will explore over-fitting and under-fitting in machine learning.

Over-fitting

When a model performs very well for trainingdatabut has poor performance with test data (new data), it is known as over-fitting. In this case, the machine learning model learns the details and noise in the training data such that it negativelyaffects the performance of the model on test data. Over-fittingcan happen due to low bias and high variance.



Reasons for over-fitting

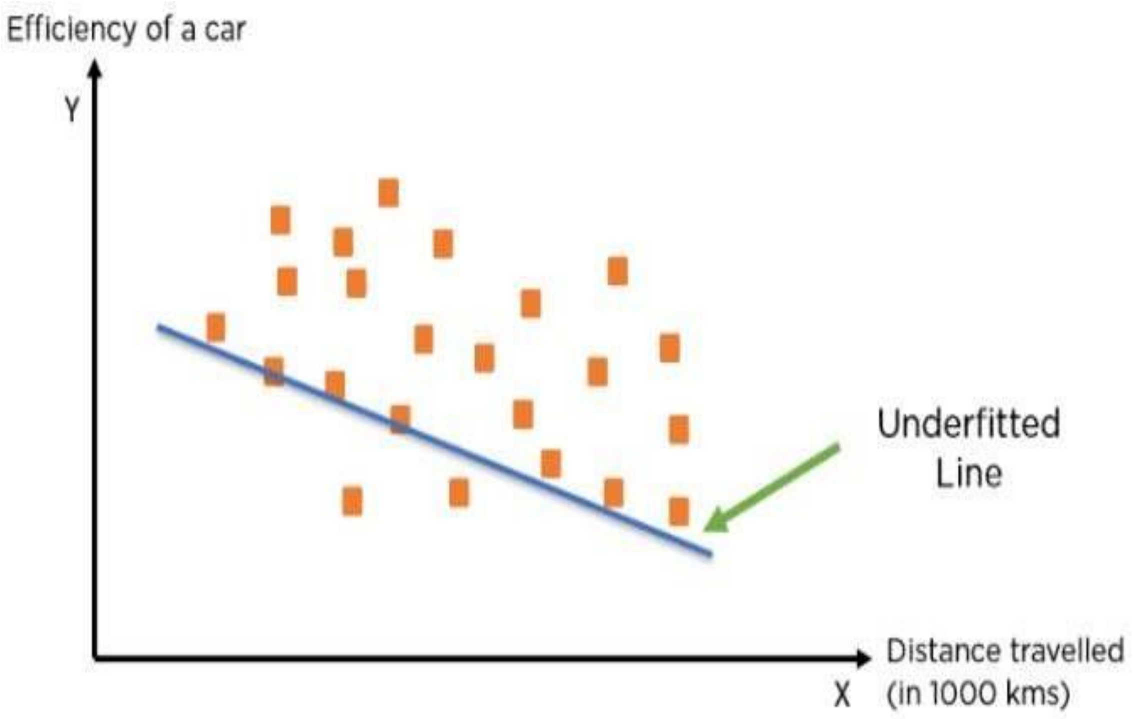
- Datausedfortrainingisnotcleanedandcontainsnoise(garbagevalues)init
- Themodelhasahigh variance
- Thesizeofthetrainingdataset usedisnot enough
- Themodel is too complex

Methods to tackleover-fitting

- Using K-fold cross-validation
- Using Regularization technique such as Lasso and Ridge
- Trainingmodelwithsufficientdata
- Adoptingensemblingtechniques

Under-fitting

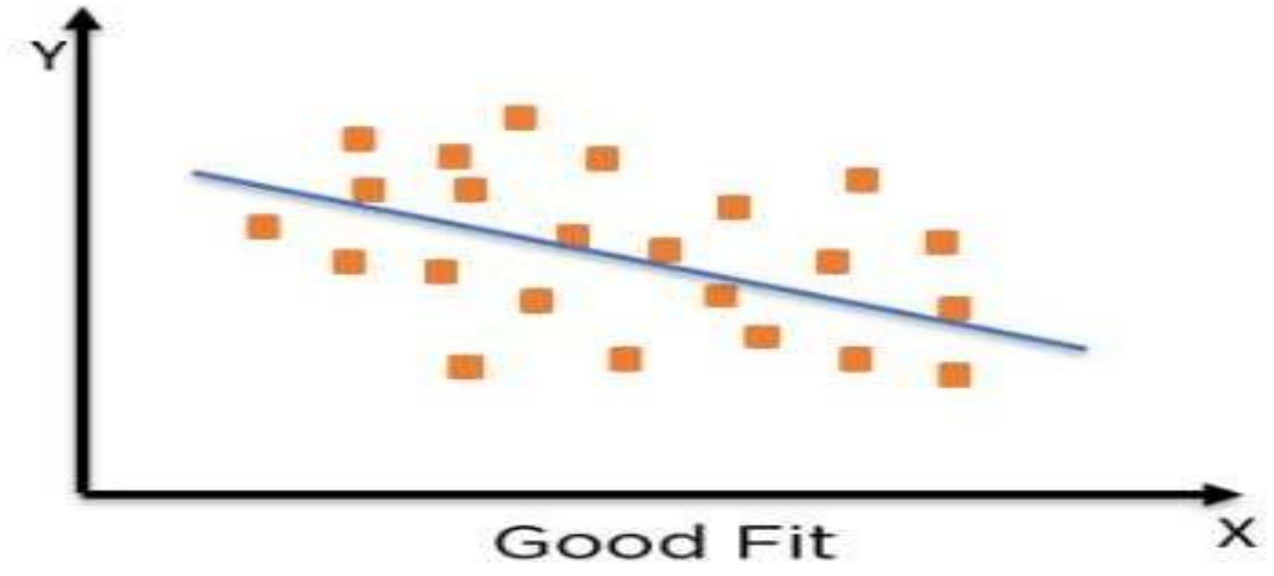
When a model has not learned the patterns in the training data well and is unable to generalize well on the new data, it is known as under-fitting. An under-fit model has poor performance on the training data andwill result in unreliable predictions. Under-fitting occurs due to high bias and low variance.



Nowthatwehaveunderstoodwhatover-fittingandunder-fittingare,let‘sseewhatagoodfitmodelisin thistutorialonover-fittingandunder-fittinginmachinelearning.

- **Goodfitinmachinelearning**

To find the good fit model, we need to look at the performance of a machine learning model over time with the training data. As the algorithm learns over time, the error for the model on the training data reduces, as well as the error on the test dataset. If we train the model for too long, the model maylearn the unnecessary details and thenoisein thetrainingset and hencelead to over-fitting. In orderto achievea good fit, weneed to stop training at a point where the error starts to increase.



Hyper-parameter

Hyper-parameters are defined as the parameters that are explicitly defined by the user to control the learning process. The value of the Hyper-parameter is selected and set by the machine learning engineer before the learning algorithm begins training the model. These parameters are tunable and can directlyaffect how well a model trains. **Hence, these are external to the model, and their values cannot be changed during the training process**. Some examples of hyper-parameters in machine learning:

- Learning Rate
- Number of Epochs
- Momentum
- Regularizationconstant
- Numberofbranchesinadecisiontree
- Numberofclustersinaclusteringalgorithm(likek-means)

Model Parameters:

Model parameters are configuration variables that are internal to the model, and a model learns them on its own. For example**, Weights or Coefficients of dependent variables in the linear regression model**. **Weights or Coefficients of independent variables in SVM, weight, and biases of a neural network, cluster centroid in clustering.** Some key points for model parameters are as follows:

- They are used by the model for making predictions
- Theyarelearnedbythe model from thedata itself

Model Hyper-parameters:

- Hyper-parameters are those parameters that are explicitly defined by the user tocontrol the learningprocess. Some key points for model parameters are as follows:
- Theseareusuallydefined manuallybythe machine learningengineer.
- One cannot know the exact best value for hyper-parameters for the given problem. The best value can be determined either by the rule of thumb or by trial and error.
- Some examples of Hyper-parameters are**the learning rate for training a neural network, K in the KNN algorithm**

Difference between ModelandHyperparameters

Thedifferenceisastabulatedbelow.

MODELPARAMETERS	HYPER-PARAMETERS
Theyarerequiredformakingpredictions	Theyarerequiredforestimatingthemodel parameters
Theyareestimatedbyoptimization algorithms(GradientDescent,Adam,Adagrad)	Theyareestimatedbyhyperparametertuning
They are not setmanually	They are setmanually
The final parameters found after training will decide how the model will perform on unseen data	The choice of hyperparameters decide how efficient the training is. In gradient descent the learning rate decide how efficient and accurate the optimization process is in estimating the parameters

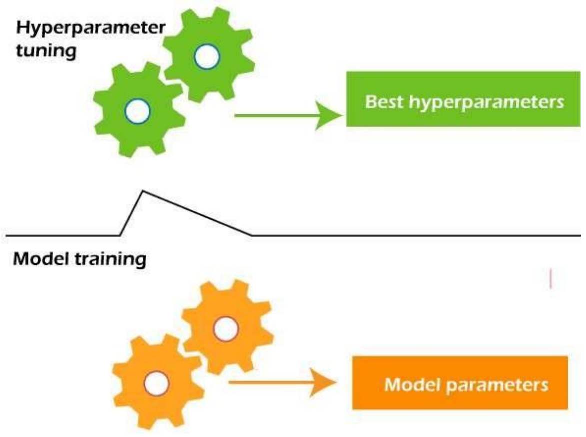
Categories of Hyper-parameters

Broadlyhyper-parameterscanbedividedintotwocategories,whicharegiven below:

- **Hyper-parameter for Optimization**
- **Hyper-parameter for Specific Models**

Hyper-parameter for optimization

The process of selecting the best hyper-parameters to use is known as hyper-parameter tuning, and the tuning process is also known as hyper-parameter optimization. Optimization parameters are used for optimizing the model.



- **Learning Rate:**The learning rate is the hyper-parameter in optimization algorithms that controls how much the model needs to change in response to the estimated error for each time when the model's weights are updated. It is one of the crucial parameters while building a neural network, and also it determines the frequencyofcross-checkingwith model parameters. Selectingthe optimized learning rate is a challenging task because if the learning rate is very less, then it may slow down the training process. On the other hand, if the learning rate is too large, then it may not optimize the model properly.
- **Batch Size:**To enhance the speed of the learning process, the training set is divided into different subsets, which are known as a batch.
- **Number of Epochs:**An epoch can be defined as the complete cycle for training the machine learning model. Epoch represents an iterative learning process. The number of epochs varies from model to model, and various models are created with more than one epoch. To determine the right number of epochs, a validation error is taken into account. The number of epochs is increased until there is a reduction in a validation error. If there is no improvement in reduction error for the consecutive epochs, then it indicates to stop increasing the number of epochs.

5.7.1.2Hyper-parameterforSpecificModels

Hyper-parametersthat areinvolved in thestructureofthe model areknown as hyper-parametersforspecific models. These are given below:

- **A number of Hidden Units:**Hidden units are part of neural networks, which refer to the components comprising the layers of processors between input and output units in a neural network.
- **Number of Layers:**A neural network is made up of verticallyarranged components, which are called layers. There are mainly**input layers, hidden layers, and output layers**. A 3-layered neural networkgivesabetterperformancethana2-layerednetwork.ForaConvolutionalNeuralnetwork,a greater number of layers make a better model.

Validation Sets

A validation set is a set of data used to train artificial intelligence (AI) with the goal of finding and optimizing the best model to solve a given problem. Validation sets are also known as dev sets. Asupervised AI is trained on a corpus of training data.

Training,tuning,modelselectionandtestingareperformedwiththreedifferentdatasets: thetrainingset,the validation set and the testing set. Validation sets are used to select and tune the final AI model.

Training sets make up the majority of the total data, averaging 60%. Most of the training data sets are collected from several resources and then pre-processed and organized to provide proper performance ofthe model. Type of training data sets determines the ability of the model to generalize .i.e. the better the quality and diversity of training data sets, the better will be the performance of the model.

V**alidation set** makes up about 20% of the bulk of data used. The validation set contrasts with training sets and test sets is an intermediate phase used for choosing the best model and optimizing it. Validation is sometimes considered a part of the training phase. In this phase that parameter tuning occurs for optimizing the selected model.Over-fittingis checked and avoided in the validation set to eliminate errors that can be caused for future predictions and observations to a specific dataset.

Testing sets make up 20% of the bulk of the data. These sets are ideal data and results with which to verify correct operation of an AI. The test set is ensured to be the input data grouped together with verified correct outputs, generally by human verification. This ideal setis used to test results and assess the performance of the final model.

Cross Validation

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. Hence the basic steps of cross-validations are:

- Reserveasubsetofthedatasetasavalidationset.
- Providethetrainingtothemodelusingthetrainingdataset.
- Now, evaluatemodel performanceusingthevalidation set. Ifthemodel performs well with the validation set, perform the further step, else check for the issues.

Methods used for Cross-Validation

Therearesomecommonmethodsthatareusedforcross-validation.These methodsaregiven below:

1. ValidationSetApproach
2. Leave-P-outcross-validation
3. Leaveoneoutcross-validation
4. K-foldcross-validation
5. Stratifiedk-foldcross-validation

Validation Set Approach

We divide our input dataset into a training set and test or validation set in the validation set approach. Both the subsets are given 50% of the dataset.

But it has one of the big disadvantages that we are just using a 50% dataset to train our model, so the model may miss out to capture important information of the dataset. It also tends to give the underfitted model.

Leave-P-outcross-validation

In this approach, the p datasets are left out of the training data. It means, if there are total n datapoints in the original input dataset, then n-p data points will be used as the training dataset and the p data points as the validation set. This complete process is repeated for all the samples, and the average error is calculated to know the effectiveness of the model.

Leave one outcross-validation

This method is similar to the leave-p-out cross-validation, but instead of p, we need to take 1 dataset out of training. It means, in this approach, for each learning set, only one datapoint is reserved, and the remaining dataset is used to train the model. This process repeats for each datapoint. Hence for n samples, we get n different training set and n test set. It has the following features:

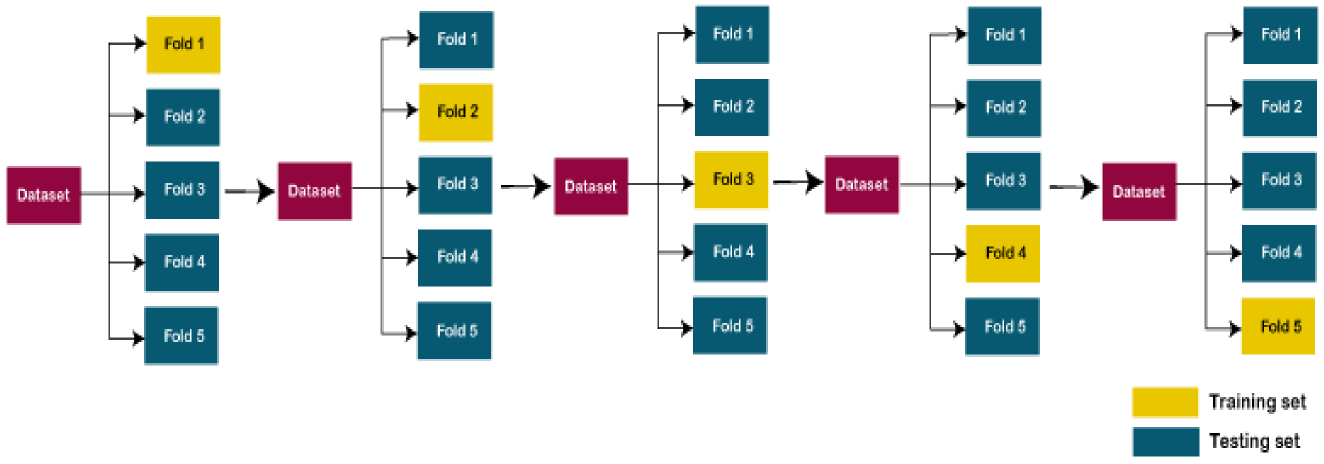
- This approach leads to high variation in testing the effectiveness of the model as we iteratively check against one data point.

K-Fold Cross-Validation

K-fold cross-validation approach divides the input dataset into K groups of samples of equal sizes. These samples are called**folds**. For each learning set, the prediction function uses k-1 folds, and the rest of the folds are used forthe test set. This approach is a verypopular CV approach because it is easyto understand, and the output is less biased than other methods.
Thestepsfork-foldcross-validationare:

- SplittheinputdatasetintoKgroups
- Foreach group:
- Takeone groupasthereserveortestdataset.
- Useremaininggroupsas thetrainingdataset
- Fitthemodelon thetrainingsetandevaluatetheperformanceof themodelusingthe testset.

Let's take an example of 5-folds cross-validation. So, the dataset is grouped into 5 folds. On 1stiteration, the first fold is reserved for test the model, and rest are used to train the model. On 2nditeration, the second fold is used to test the model, and rest are used to train the model. This process will continue until each fold isnot used for the test fold.
Considerthebelow diagram:



Stratifiedk-fold cross-validation

This technique is similar to k-fold cross-validation with some little changes. This approach works on stratification concept, it is a process of rearranging the data to ensure that each fold or group is a good representative of the complete dataset. To deal with the bias and variance, it is one of the best approaches.
It can be understood with an example of housing prices, such that the price of some houses can be much high than other houses. To tackle such situations, a stratified k-fold cross-validation technique is useful.

Holdout Method

This method is the simplest cross-validation technique among all. In this method, we need to remove a subset of the training data and use it to get prediction results by training it on the rest part of the dataset.
The error that occurs in this process tells how well our model will perform with the unknown dataset. Although this approach is simple to perform, it still faces the issue of high variance, and it also produces misleading results sometimes.

Comparison of Cross-validationtotrain/testsplitinMachineLearning

- **Train/test split:**The input data is divided into two parts, that are training set and test set on a ratio of 70:30, 80:20, etc. It provides a high variance, which is one of the biggest disadvantages.
- **Training Data:**The training data is used to train the model, and the dependent variable is known.
- **Test Data:**The test data is used to make the predictions from the model that is already trainedon the training data. This has the same features as training data but not the part of that.
- **Cross-Validation dataset:**It is used to overcome the disadvantage of train/test split bysplitting the dataset into groups of train/test splits, and averaging the result. It can be used if we want to optimize our model that has been trained on the training dataset for the best performance. It is more efficient as compared to train/test split as every observation is used for the training and testing both.
-

Limitations of Cross-Validation

- For the ideal conditions, it provides the optimum output. But for the inconsistent data, it may produce a drastic result. So, it is one of the big disadvantages of cross-validation, as there is no certainty of the type of data in machine learning.
- In predictive modeling, the data evolves over a period, due to which, it may face the differences between the training set and validation sets. Such as if we create a model for the prediction of stock market values, and the data is trained on the previous 5 years stock values, but the realistic future values for the next 5 years may drastically different, so it is difficult to expect the correct output for such situations.

Applications of Cross-Validation

- This technique can be used to compare the performance of different predictive modeling methods.
- Ithasgreatscopeinthemedicalresearch field.
- It can also be used for the meta-analysis, as it is already being used by the data scientists in the field of medical statistics.

Estimators

Inmachine learning, an **estimator** is an equation for picking the ―best,‖ or most likely accurate, data model based upon observations in realty. The estimator is the formula that evaluates a given quantity and generates an estimate. This estimate is then inserted into thedeep learningclassifiersystem to determine what action to take. **Estimation**isastatisticaltermforfindingsomeestimateofunknownparameter,given somedata.PointEstimation istheattemptto providethe singlebestpredictionofsomequantityofinterest.
Quantity of interest can be:

- A single parameter
- A vector of parameters—e.g.,weightsinlinearregression
- A whole function

➢ Point estimator

Todistinguishestimatesofparametersfromtheirtruevalue,apointestimateofaparameter **θ**is represented by **θˆ.** Let **{x(1) , x(2) ,..x(m)}**be **m** independent and identically distributed data points. Then a point estimator is any function of the data:
Thisdefinitionof apointestimator isverygeneralandallows the designerof anestimatorgreatflexibility.

$$\hat{\theta}_m = g(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$$

While almost any function thus qualifies as an estimator, a good estimator isa function whose output is close to the true underlying θ that generated the training data.
Pointestimationcanalsorefertoestimationofrelationshipbetweeninputandtargetvariablesreferred to as function estimation.

➢ Function Estimator

Herewearetryingtopredictavariableygivenaninputvectorx.Weassumethatthereisafunction**f(x)** thatdescribestheapproximaterelationshipbetweenyandx.Forexample,
wemayassumethat **y= f(x)+ε**, where **ε**standsfor the partof **y**thatisnot predictablefrom **x**.In function estimation, we are interested in approximating **f**with a model or estimate **fˆ**. Function estimation is really just the same as estimating a parameter **θ**; the function estimator **fˆ**is simplya point estimator in function space. Ex: in polynomial regression we areeither estimating a parameter***w***or estimating a function mapping from ***x*** to **y**.

Uses of Estimators

By quantifying guesses, estimators are how machine learning in theory is implemented in practice. Without the abilityto estimate the parameters of a dataset (such as the layers in a neural networkor the bandwidth in a kernel), there would be no wayfor an AI system to ―learn.‖
A simple example of estimators and estimation in practice is the so-called ―German Tank Problem‖ from WorldWarTwo.The Allieshadno waytoknowforsurehow manytankstheGermans werebuildingevery month. By counting the serial numbers of captured or destroyed tanks, allied statisticians created an estimator rule. This equation calculated the maximum possible number of tanks based upon the sequential serial numbers, and applies minimum varianceanalysis to generate the most likely estimate for how many new tanks German was building.

Types of Estimators

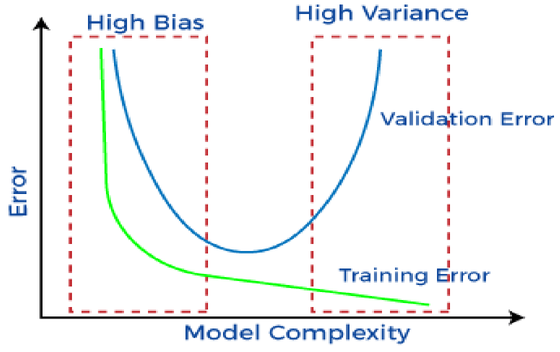
Estimators come in two broad categories, **point and interval**. Point equations generate single value results, suchasstandarddeviation, that can beplugged into adeep learningalgorithm‘s classifier functions. Interval equations generate a range of likely values, such as a confidence interval, for analysis.
Inaddition,each estimatorrulecanbetailoredto generatedifferenttypesof estimates:

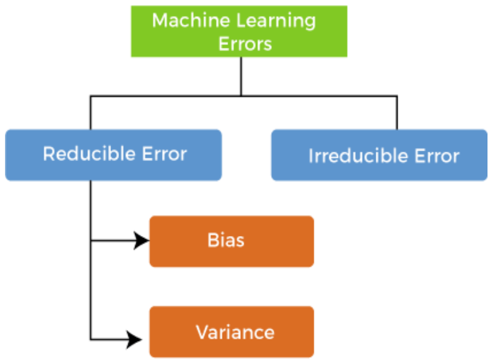
- **Biased:**Eitheranoverestimateoranunderestimate.
- **Efficient:**Smallestvarianceanalysis.Thesmallestpossiblevarianceisreferredtoasthe―best‖ estimate.
- **Invariant:**Lessflexibleestimatesthataren‘teasilychangedbydatatransformations.
- **Shrinkage:**Anunprocessedestimatethat‘scombinedwithothervariablestocreatecomplex estimates.
- **Sufficient:**Estimatingthetotalpopulation‘sparameterfromalimiteddataset.
- **Unbiased:**Anexact-matchestimatevaluethatneitherunderestimatesnoroverestimates.

Bias and Variance

Errorsin Machine Learning

In machine learning, an error is a measure of how accurately an algorithm can make predictions for the previously unknown dataset. On the basis of these errors, the machine learning model is selected that can perform best on the particular dataset. There are mainly two types of errors in machine learning, which are:





Reducible errors: These errors can be reduced to improve the model accuracy. Such errors can further be classified into bias and Variance.
Irreducible errors: These errors will always be present in the model regardless of which algorithm hasbeen used. The cause of these errors is unknown variables whose value can't be reduced.

Bias

In general, a machine learning model analyses the data, find patterns in it and make predictions. While training, the model learns these patterns in the dataset and applies them to test data for prediction. **While making predictions, a difference occurs between prediction values made by the model and actual values/expected values,and this difference is known as bias errors or Errors due to bias**. It can be defined as an inability of machine learning algorithms such as Linear Regression to capture the true relationship between the data points. Each algorithm begins with some amount of bias because bias occurs from assumptions in the model, which makes the target function simple to learn. A model has either:

- **Low Bias:**A low bias model will make fewer assumptions about the form of thetarget function.
- **High Bias:**A model with a high bias makes more assumptions, and the model becomes unable to capture the important features of our dataset. **A high bias model also cannot perform well on new data.**

Generally, a linear algorithm has a high bias, as it makes them learn fast. The simpler the algorithm, the higher the bias it has likely to be introduced. Whereas a nonlinear algorithm often has low bias.
Some examples of machine learning algorithms with low bias **are Decision Trees, k-Nearest Neighbours and Support Vector Machines**. At the same time, an algorithm with high bias is **Linear Regression, Linear Discriminant Analysis and Logistic Regression.**

Ways to reduce High Bias:

- Increase the input features as the model is under-fitted.
- Decreasetheregularizationterm.
- Usemorecomplex models,suchasincludingsomepolynomialfeatures.

Variance

The variance would specify the amount of variation in the prediction if the different training data was used. Insimplewords,**variancetellsthathowmucharandomvariableisdifferentfromitsexpected value.** Ideally, a model should not vary too much from one training dataset to another, which means the algorithm should be good in understanding the hidden mapping between inputs and output variables. Variance errors are either of **low variance or high variance.**

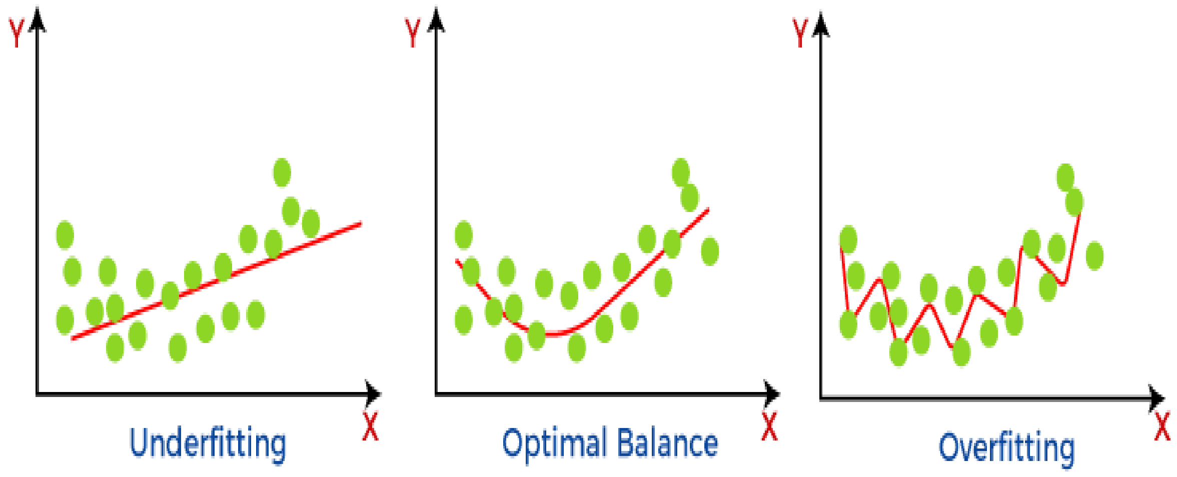
- **Low variance**means there is a small variation in the prediction of the target function with changes in the training data set.
- **High variance**shows a large variation in the prediction of the target function with changes in the training dataset.

A model that shows high variance learns a lot and performs well with the training dataset, and does not generalize well with the unseen dataset. As a result, such a model gives good results with the trainingdataset but shows high error rates on the test dataset.

Since, with high variance, the model learns too much from the dataset, it leads to over-fitting of the model. A model with high variance has the below problems:

- Ahighvariancemodelleadstoover-fitting.
- Increasemodel complexities.

Usually,nonlinearalgorithmshavealot offlexibilitytofitthemodel,have high variance.

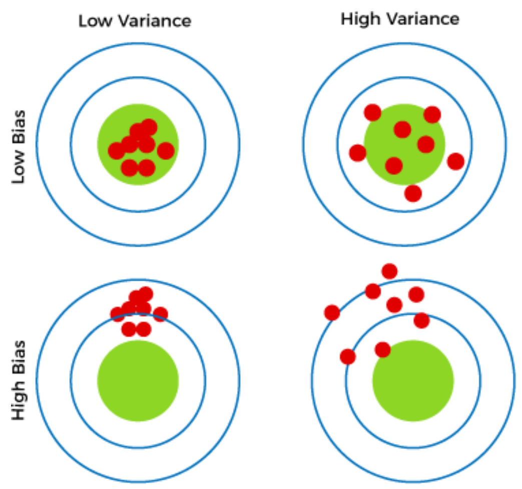


Some examples of machine learning algorithms with low variance are, **Linear Regression, Logistic Regression,andLineardiscriminantanalysis**.Atthesametime,algorithmswithhighvariance are **decision tree, Support Vector Machine, and K-nearest neighbours.**

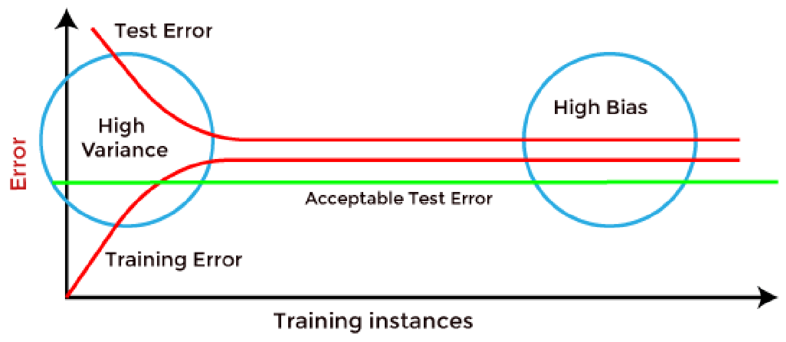
.

Different Combinations of Bias-Variance

Therearefourpossiblecombinationsofbiasandvariances,whicharerepresentedbythebelow diagram:

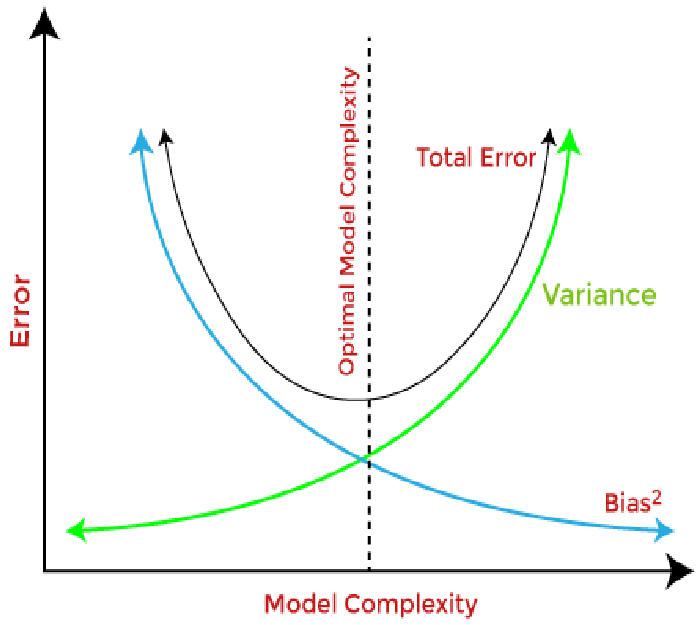


1. **Low-Bias, Low-Variance:**The combination of low bias and low variance shows an ideal machine learning model. However, it is not possible practically.
2. **Low-Bias, High-Variance:**With low bias and high variance, model predictions are inconsistent and accurate on average. This case occurs when the model learns with a large number of parameters and hence leads to an **over-fitting**
3. **High-Bias, Low-Variance:**With High bias and low variance, predictions are consistent but inaccurate on average. This case occurs when a model does not learn well with the training dataset or uses few numbers of the parameter. It leads to **under-fitting** problems in the model.
4. **High-Bias, High-Variance:**With high bias and high variance, predictions are inconsistent and also inaccurate on average.



Bias-Variance Trade-Off

While building the machine learning model, it is really important to take care of bias and variance in orderto avoid over-fitting and under-fitting in the model. If the model is very simple with fewer parameters, it mayhavelowvarianceandhighbias.Whereas,ifthemodelhasalargenumberofparameters,itwillhave high variance and low bias. So, it is required to make a balance between bias and variance errors, and thisbalance between the bias error and variance error is known as **the Bias-Variance trade-off.**



For an accurate prediction of the model, algorithms need a low variance and low bias. But this is not possible because bias and variance are related to each other:

Bias-Variance trade-off is a central issue in supervised learning. Ideally, we need a model that accurately captures the regularities in training data and simultaneously generalizes well with the unseen dataset. Unfortunately, doing this is not possible simultaneously. Because a high variance algorithm may perform well with training data, but it maylead to over-fitting to noisy data. Whereas, high bias algorithm generates a much simple model that may not even capture important regularities in the data. So, we need to find a sweet spot between bias and variance to make an optimal model.
Hence, the**Bias-Variancetrade-off is aboutfinding thesweetspot to makea balancebetween bias and variance errors.**

5.10.Challenges Motivating Deep Learning

1. Learning without Supervision
Deep learning models are one of, if not the most data-hungry models of the Machine Learning world. They need huge amounts of data to reach their optimal performance and serve us with the excellence we expect from them.
However, having this much data is not always easy. Additionally, while we can have large amounts of data on some topic, many times it is not labeled so we cannot use it to train any kind of supervised learning algorithm.

One of the main challenges of Deep Learning derived from this is being able to deliver great performances with a lot less training data. As we will see later, recent advances like transfer learning or semi-supervised learning are already taking steps in this direction, but still it is not enough.

2. Coping with data from outside the training distribution

Data is dynamic, it changes through different drivers like time, location, and many other conditions. However,MachineLearningmodels,includingDeepLearningones,arebuiltusingadefinedsetofdata (*the training set*) and perform well as long as the data that is later used to make predictions once the system is built comes from the same distribution as the data the system was built with.

This makes them perform poorly when data that is not entirelydifferent, but that does have some variations from the training data is fed to them. Another challenge of Deep Learning in the future will be to overcome thisproblem,andstillperformreasonablywellwhendatathatdoesnotexactlymatchthe trainingdataisfed to them.

3. Incorporating Logic

Incorporating some sort of rule based knowledge, so that logical procedures can be implemented and sequential reasoning used to formalize knowledge.

While these cases can be covered in code, Machine Learning algorithms don't usually incorporate sets or rules into their knowledge. Kind of like a prior data distribution used in Bayesian learning, sets of pre-defined rules could assist Deep Learning systems in their reasoning and live side by side with the _learning from data' based approach.

4. The Need for less data and higher efficiency

Althoughwekindofcoveredthisinourfirsttwosections,thispointisreallyworth highlighting.

The success of Deep Learning comes from the possibility to incorporate many layers into our models, allowing them to try an insane number of linear and non-linear parameter combinations. However, with more layers comes more model complexity and we need more data for this model to function correctly.

When the amount of data that we have is effectivelysmaller than the complexityof the neural network then we need to resort to a different approach like the aforementioned Transfer Learning.

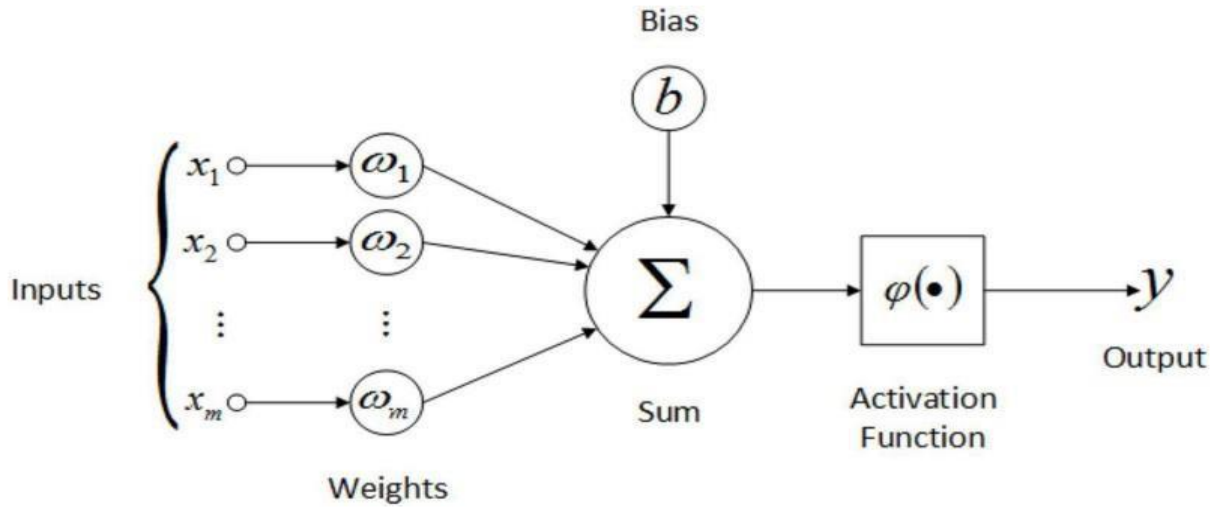
Also, too big Deep Learningmodels, aside from needingcrazyamounts of data to be trained on, use a lot of computational resources and can take a very long while to train. Advances on the field should also be oriented towards making the training process more efficient and cost effective

6. Deep Neural Network

Deep neural networks (DNN) is a class of machine learning algorithms similar to the artificial neural network and aims to mimic the information processing of the brain. Deep neural networks, or deep learning networks,have several hidden layers with millions of artificial neurons linked together. A number, called weight, represents the connections between one node and another. The weight is a positive number if one node excites another, or negative if one node suppresses the other.

Feed-Forward Neural Network

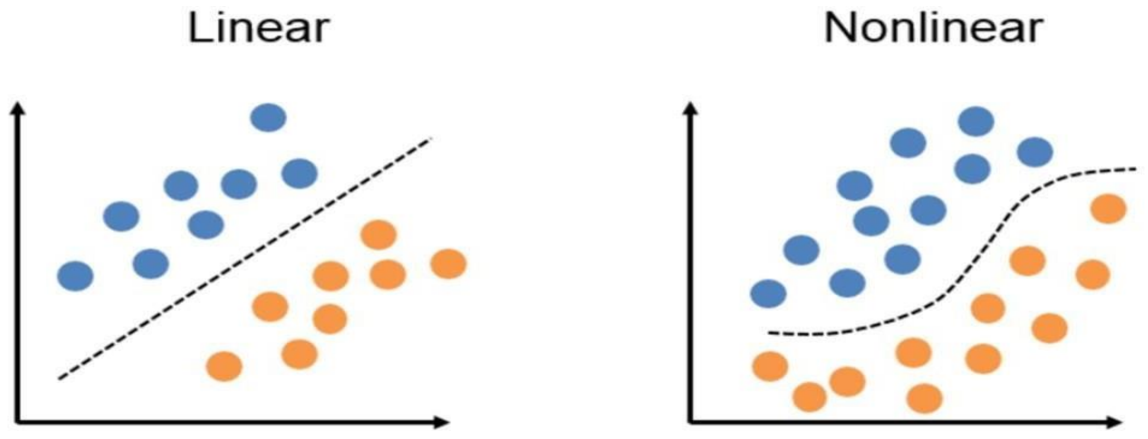
In its most basic form, a Feed-Forward Neural Network is a single layer perceptron. A sequence ofinputs enter the layer and aremultiplied bythe weights in this model. Theweightedinput values are then summed together to form a total. If the sum of the values is more than a predetermined threshold, which is normally set at zero, the output value is usually1, and if the sum is less than the threshold, the output value is usually -1. The single-layer perceptron is a popular feed-forward neural network model that is frequently used for classification. Single-layer perceptrons can also contain machine learning features.



The neural network can compare the outputs of its nodes with the desired values using a property known as the delta rule, allowing the network to alter its weights through training to create more accurateoutput values.This training and learning procedure results in gradient descent. The technique of updating weights in multi-layered perceptrons is virtually the same, however, the process is referred to as back- propagation. In such circumstances, the output values provided by the final layer are used to alter each hidden layer inside the network.

Work Strategy

The function of each neuron in the network is similar to that of linear regression. The neuron also hasan activation function at the end, and each neuron has its weight vector.



6.1..2 Importance of the Non-Linearity

When*two or more linear objects, such as a line, plane, or hyperplane, are combined, the outcome is also a linear object: line, plane, or hyperplane*. No matter how many of these linear things we add, we‘ll still end up with a linear object.

However, this is not the case when adding non-linear objects. When two separate curves are combined, the result is likely to be a more complex curve.

We're introducing non-linearity at every layer using these activation functions, in addition to just adding non-linear objects or hyper-curves like hyperplanes. In other words, we‘re applying a nonlinear function on an already nonlinear object.

Supposeifneuralnetworksdidn‘thaveanactivationfunction,they‘djustbeahugelinearunitthatasingle
= m*x + d
Z=k*a+t=>k*(m*x+d)+t=>k*m*x+k*d+t=>(k*m)*x+ (k*c+t)

6.1.3ApplicationsoftheFeedForwardNeural Networks

A Feed Forward Neural Network is an artificial neural network in which the nodes are connected circularly. A feed-forward neural network, in which some routes are cycled, is the polar opposite of a recurrent neuralnetwork. The feed-forward model is the simplest type of neural network because the input is only processed in one direction. The data always flows in one direction and never backwards, regardless of how many buried nodes it passes through.

Regularization in Machine Learning

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it.
Sometimes themachine learningmodel performs well with the training data but does not perform well with thetestdata. Itmeansthemodelisnotabletopredicttheoutputwhen deals withunseendatabyintroducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.
This technique can be used in such a waythat it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model.
It mainly regularizes or reduces the coefficient of features toward zero. In simple words, "*In regularization technique, we reduce the magnitude of the features by keeping the same number of features."*
HowdoesRegularizationWork?
Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:
y=β0+β1x1+β2x2+β3x3+⋯+βnxn+b
Inthe aboveequation,Y representsthevaluetobepredicted
X1,X2,…XnarethefeaturesforY.
β0,β1,…..βnaretheweightsormagnitudeattachedtothefeatures,respectively.Hererepresentsthebiasof themodel,and brepresentstheintercept.
Linearregressionmodelstrytooptimizetheβ0andbtominimizethecostfunction.Theequationforthe costfunctionforthelinearmodelisgivenbelow:

$$\sum_{i=1}^{M}\left(y_i - y'_i\right)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n} \beta_j * Xij\right)^2$$

Now,we willaddalossfunctionandoptimizeparameter tomake themodelthatcanpredicttheaccurate value of Y. The loss function for the linear regression is called as **RSS or Residual sum of squares.**
TechniquesofRegularization
Therearemainlytwotypesofregularizationtechniques,whicharegivenbelow:

- **Ridge Regression**
- **Lasso Regression**

Ridge Regression

Ridge regression is one of the types of linear regression in which a small amount of bias is introduced sothat we can get better long-term predictions.
Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It isalso called as **L2 regularization**.
In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called**Ridge Regression penalty**. We can calculate it by multiplying with the lambda to the squared weight of each individual feature.

In the above equation, the penaltyterm regularizes the coefficients of the model, and hence ridge regression reduces the amplitudes of the coefficients that decreases the complexity of the model.
As we can see from the above equation, if the values of **λ tend to zero, the equation becomes the cost function of the linear regression model.** Hence, for the minimum value of λ, the model will resemble the linear regression model.
A general linear or polynomial regression will fail if there is high collinearity between the independent variables, so to solve such problems, Ridge regression can be used.

Lasso Regression

Lassoregressionisanotherregularizationtechniquetoreducethecomplexityofthemodel.Itstandsfor **Least Absolute and Selection Operator.**
It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.
Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
Itisalsocalledas **L1regularization.**TheequationforthecostfunctionofLasso regressionwillbe:

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{n}|\beta_j|$$

Someofthefeaturesinthistechniquearecompletelyneglectedformodelevaluation.
Hence,theLassoregressioncanhelpustoreducetheoverfittinginthemodelaswellasthefeature selection.
KeyDifferencebetween RidgeRegressionandLassoRegression
Ridgeregression ismostlyusedtoreducetheoverfittinginthemodel,anditincludesallthefeatures present in the model. It reduces the complexity of the model by shrinking the coefficients.
Lassoregression helpstoreducetheoverfittinginthemodelaswellasfeatureselection.

Optimizationin Machine Learning

In machine learning, optimization is the procedure of identifying the ideal set of model parameters that minimize a loss function. For a particular set of inputs, the loss function calculates the discrepancy between the predicted and actual outputs. For the model to successfully forecast the output for fresh inputs, optimization seeks to minimize the loss function.

A method for finding a function's minimum or maximum is called an optimization algorithm, which is used in optimization. Up until the minimum or maximum of the loss function is reached, the optimization algorithm iteratively modifies the model parameters. Gradient descent, stochastic gradient descent, Adam, Adagrad, and RMSProp are a few optimization methods that can be utilised in machine learning.

- **Gradient Descent**

  In machine learning, gradient descent is a popular optimization approach. It is a first-order optimization algorithm that works by repeatedly changing the model's parameters in the opposite direction of the loss function's negative gradient. The loss function lowers most quickly in that direction because the negative gradient leads in the direction of the greatest descent.

  The gradient descent algorithm operates by computing the gradient of the loss function with respect to each parameter starting with an initial set of parameters. The partial derivatives of the loss function with respect to each parameter are contained in a vector known as the gradient. After that, the algorithm modifies the parameters by deducting a small multiple of the gradient from their existing values.

- **Stochastic Gradient Descent**

  A part of the training data is randomly chosen for each iteration of the stochastic gradient descent process, which is a varianton the gradient descent technique. This makes the algorithm's computations simpler and speeds up its convergence. For big datasets when it is not practical to compute the gradient of the loss function for all of the training data, stochastic gradient descent is especially helpful.

  The primary distinction between stochastic gradient descent and gradient descent is that stochastic gradient descent changes the parameters based on the gradient obtained for a single example rather than the full dataset. Due to the stochasticity introduced by this, each iteration of the algorithm may result in a different local minimum.

- **Adam**

  Adam is an optimization algorithm that combines the advantages of momentum-based techniques and stochastic gradient descent. The learning rate during training is adaptively adjusted using the firstandsecond momentsofthegradient.Adamisfrequentlyusedindeep learningsinceitisknown to converge more quickly than other optimization techniques.

- **Adagrad**

  An optimization algorithm called Adagrad adjusts the learning rate for each parameter based on previous gradient data. It is especially beneficial for sparse datasets with sporadic occurrences of specific attributes. Adagrad can converge more quickly than other optimization methods because it uses separate learning rates for each parameter.

- **RMSProp**

  An optimization method called RMSProp deals with the issue of deep neural network gradients that vanishandexplode. Itemploysthemovingaverage ofthesquaredgradienttonormalizethelearning rate for each parameter. Popular deep learning optimization algorithm RMSProp is well known for converging more quickly than some other optimization algorithms.

Importance of Optimization in Machine Learning

Machine learning depends heavily on optimization since it gives the model the ability to learn from dataand generate precise predictions. Model parameters are estimated using machine learning techniques using the observed data. Finding the parameters' ideal values to minimize the discrepancy between the predicted and actual results for a given set of inputs is the process of optimization. Without optimization, the model's parameters would be chosen at random, making it impossible to correctly forecast the outcome for brand- new inputs.

Optimization is highlyvalued in deep learning models, which have multiple levels of layers and millions of parameters. Deep neural networks need a lot of data to be trained, and optimizing the parameters of the model in which they are used requires a lot of processing power. The optimization algorithm chosen can have a big impact on the training process's accuracy and speed.

New machine learning algorithms are also implemented solely through optimization. Researchers are constantly looking for novel optimization techniques to boost the accuracy and speed of machine learning systems. These techniques include normalization, optimization strategies that account for knowledge of the underlying structure of the data, and adaptive learning rates.

Challenges in Optimization

There are difficulties with machine learning optimization. One of the most difficult issues is overfitting, which happens when the model learns the training data too well and is unable to generalize to new data. When the model is overly intricate or the training set is insufficient, overfitting might happen.

When the optimization process converges to a local minimum rather than the global optimum, it poses the problem of local minima, which is another obstacle in optimization. Deep neural networks, which contain many parameters and may have multiple local minima, are highly prone to local minima.

Shree SATHYAM COLLEGE OF ENGG & TECH

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

AD3501 / DEEPLEARNING - NOTES

UNIT – II CONVOLUTIONAL NEURAL NETWORKS
Convolution Operation -- Sparse Interactions -- Parameter Sharing -- Equivariance -- Pooling -- Convolution Variants: Strided -- Tiled -- Transposed and dilated convolutions; CNN Learning: Nonlinearity Functions -- Loss Functions -- Regularization -- Optimizers --Gradient Computation.

1. Introduction to Convolutional Neural Networks:

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.In a regular Neural Network the reare three types of layers:

- Hidden Layer: The input from the Input layer is then feed into the hidden layer. There can be many hidden layersdepending uponour modeland datasize. Each hidden layer can havedifferent numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplicationofoutput oftheprevious layer with learnable weightsofthat layer and then bytheadditionoflearnable biases followed byactivation functionwhich makesthe network nonlinear.
- Output Layer: The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

The data is fed into the model and output from each layer is obtained from the above step is called feed forward, wethencalculatetheerror using anerrorfunction, somecommonerrorfunctionsarecross-entropy, square loss error, etc. The error function measures how well the network is performing. After that, we back propagateinto the modelbycalculating thederivatives. Thisstep iscalled Back propagationwhich basically is used to minimize the loss.

Convolution Neural Network

Convolutional Neural Network(CNN)is the extended version of artificial neural networks(ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example visual datasets like images or videos where data patterns play an extensive role.

Around the 1980s, CNNs were developed and deployed for the first time. A CNN could only detect handwritten digits at the time. CNN was primarily used in various areas to read zip and pin codes etc. The most common aspect of any AImodel is thatitrequires amassiveamountof data totrain. Thiswas one of

The biggest problemsthatCNNfacedatthetime,andduetothis,theywereonlyusedinthepostal industry. Yann LeCun was the first to introduce convolutional neural networks.

Convolutional Neural Networks, commonly referred to as CNNs, are a specialized kind of neural network architecture that is designed to process data with a grid-like topology. This makes them particularly well-suited for dealing with spatial and temporal data, like images and videos that maintain a high degree of correlation between adjacent elements.

CNNs are similar to other neural networks, but they have an added layer of complexity due to the fact that they use a series of convolutional layers. Convolutional layers perform a mathematical operation called convolution,a sortof specialized matrix multiplication, on theinputdata. The convolution operation helps to preserve the spatial relationship between pixels by learning image features using small squares of input data. . The picture below represents a typical CNN architecture.

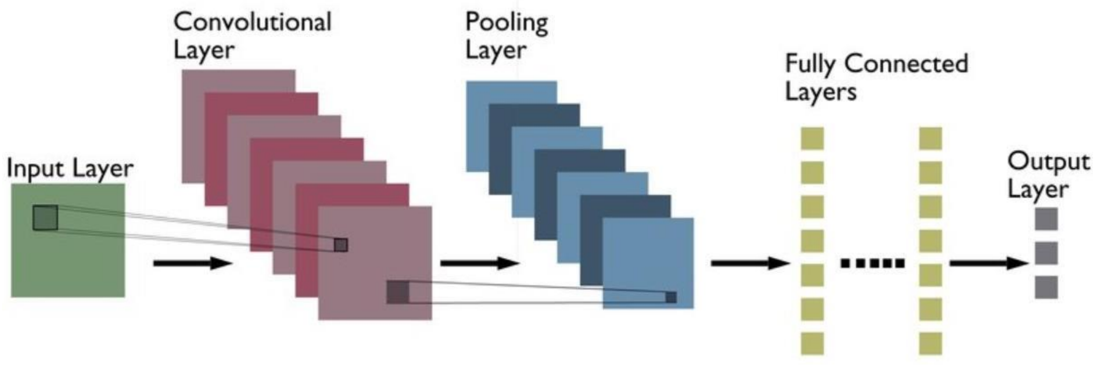


Fig.1 Typical CNN architecture

The following are definitions of different layers shown in the above architecture:

- **Convolutional layers**

Convolutional layers operate by sliding a set of 'filters' or 'kernels' across the input data. Each filter is designed todetecta specificfeature or pattern, such as edges,corners, ormore complexshapes in the case of deeper layers. As these filters move across the image, they generate a map that signifies the areas where thosefeatureswerefound.Theoutputoftheconvolutionallayerisafeaturemap,whichisa representation of theinputimagewith thefilters applied.Convolutionallayerscanbestackedtocreate morecomplexmodels,whichcanlearnmoreintricatefeaturesfromimages.Simplyspeaking, convolutional layers are responsible for extracting features from the input images. These features might include edges, corners, textures, or more complex patterns.

- **Pooling layers**

Pooling layers follow the convolutional layers and are used to reduce the spatial dimension of the input, makingiteasier toprocess and requiringless memory. In the contextof images,"spatial dimensions" refer tothewidth andheightof theimage.Animageismadeupof pixels,andyoucan thinkof itlikeagrid, with rows and columns of tiny squares (pixels). By reducing the spatial dimensions, pooling layers help reducethenumberofparametersorweightsinthenetwork.Thishelpstocombatover-fittingandhelp train the model in a fast manner. Max pooling helps in reducing computational complexity, owing to reductioninsizeoffeaturemap,andmakingthemodelinvarianttosmalltransitions.Withoutmax pooling,thenetworkwouldnotgaintheabilitytorecognizefeaturesirrespectiveofsmallshiftsor rotations. This would make the model less robust to variations in object positioning within the image, possibly affecting accuracy.

There are two main types of pooling: max pooling and average pooling. Max pooling takes the maximum valuefromeachfeaturemap.Forexample,ifthepoolingwindowsizeis2×2,itwillpickthepixelwith the highest value in that 2×2 region. Max pooling effectively captures the most prominent feature or characteristic within the pooling window. Average pooling calculates the average of all values within the pooling window. It provides a smooth, average feature representation.

- **Fully connected layers**

Fully-connectedlayers are one of themostbasictypes of layers in a convolutional neural network (CNN). As the name suggests, each neuron in a fully-connected layerisFully connected- to every otherneuron in the previous layer. Fully connected layers are typically used towards the end of a CNN- when the goal is to take thefeatureslearnedby the convolutional andmax poolinglayers and use them tomake predictions suchasclassifyingtheinputtoalabel.Forexample,ifwewereusingaCNNtoclassifyimagesof animals,thefinalFullyconnectedlayermighttakethefeatureslearnedbythepreviouslayersanduse them to classify an image as containing a dog, cat, bird, etc. Fullyconnectedlayerstakethehigh-dimensionaloutputfromthepreviousconvolutionalandpooling layers and flatten it into a one-dimensional vector. This allows the network to combine and integrate all the extracted features across the entire image, rather than considering localized features. It helps inunderstanding the global context of the image. The fully connected layers are responsible for mapping the integrated features to the desired output, such as class labels in classification tasks. They act as the final decision-making part of the network, determining what the extracted features mean in the context of the specific problem (e.g., recognizing a cat or a dog).

The combination of Convolution layer followed by max-pooling layer and then similar sets creates a hierarchy of features. The first layer detects simple patterns, and subsequent layers build on those to detect more complex patterns.

CNNs are often used for image recognition and classification tasks. For example, CNNs can be used to identifyobjects in an image or to classifyan image as being a cat or a dog. CNNs can also be used for more complex tasks, such as generating descriptions of an image or identifying the points of interest in an image. Beyond image data, CNNs can also handle time-series data, such as audio data or even text data, although other types of networks like Recurrent Neural Networks (RNNs) or transformers are often preferred forthese scenarios. CNNs are a powerful tool for deep learning, and they have been used to achieve state-of- the-art results in many different applications.

CNN architecture

ConvolutionalNeuralNetworkconsistsofmultiplelayersliketheinputlayer,Convolutionallayer, Pooling layer, and fully connected layers.

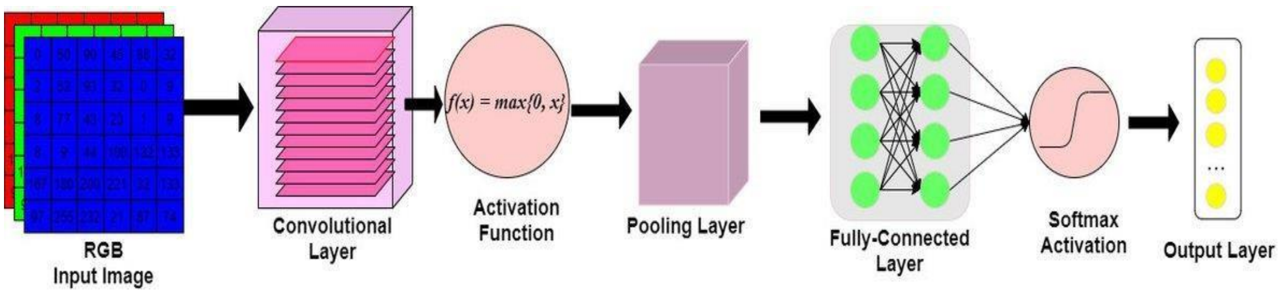


Fig.2 Simple CNN architecture

The Convolutional layer appliesfilters to the inputimage to extractfeatures, the Poolinglayerdown samples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through back propagation and gradientdescentas detailed in Fig.3.

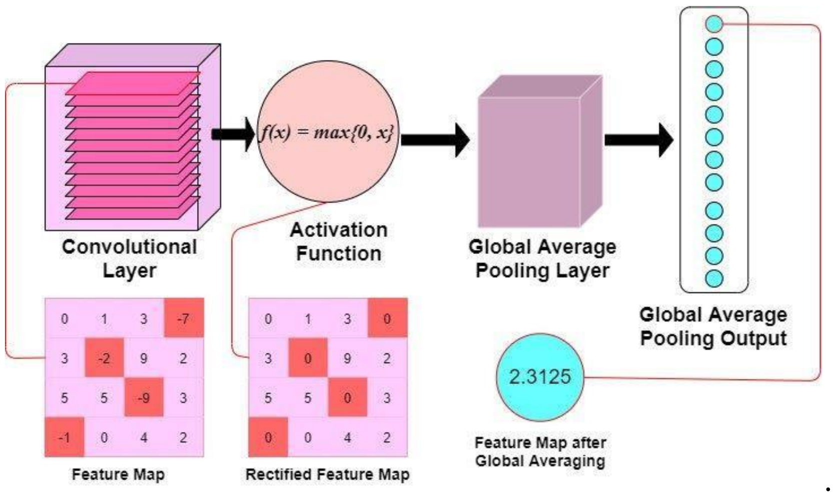


Fig.3 Functions of CNN Layers

Different types of CNN Architectures

LeNet: LeNet is the first CNN architecture. It was developed in 1998 byYann LeCun, Corinna Cortes, and Christopher Burges for handwritten digit recognition problems. LeNet was one ofthe first successful CNNs and is often considered the "Hello World" of deep learning. It is one of the earliest and most widely-used CNN architectures and has been successfully applied to tasks such as handwritten digit recognition. The LeNet architecture consists of multiple convolutional and pooling layers, followed by a fully-connected layer. The model has five convolution layers followed by two fully connected layers. LeNet was the beginningofCNNs indeep learning for computervisionproblems. However, LeNet couldnottrainwelldue tothe vanishing gradientsproblem. To solve this issue, a shortcut connection layer knownasmax-pooling is used between convolutional layers to reduce the spatial size of images which helps prevent overfitting and allows CNNs to train more effectively. The diagram below represents LeNet-5 architecture.

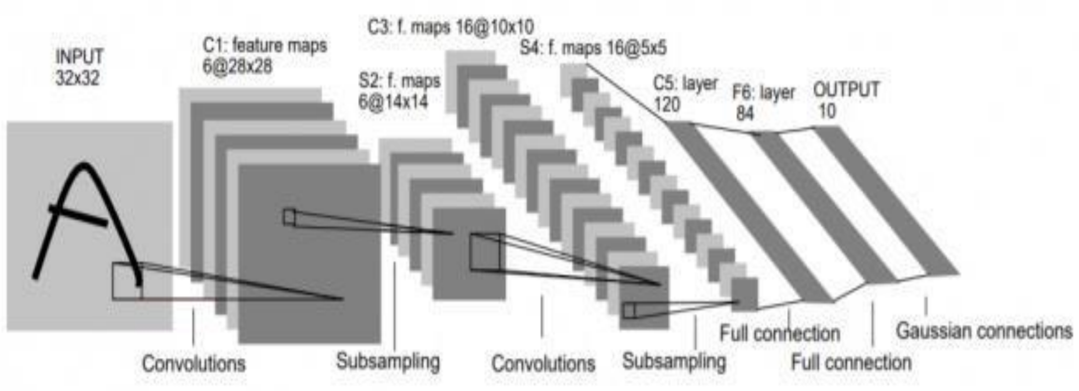


Fig.4 LeNet Architecture

The LeNet CNN is a simple yet powerful model that has been used for various tasks such as handwritten digit recognition, traffic sign recognition, and face detection. Although LeNet was developed more than 20 years ago, its architecture is still relevant today and continues to be used.

AlexNet: AlexNet is the deep learning architecture that popularized CNN. It was developed by Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. AlexNet network had a very similar architecture to LeNet,but was deeper, bigger, and featured Convolutional Layers stacked on top of each other. AlexNet was the first large-scale CNN and was used to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in2012. The AlexNet architecture wasdesigned to be used with large-scale image datasetsand it achieved state-of-the-art results atthetime ofits publication. AlexNet is composed of5convolutionallayers with a combination of max-pooling layers, 3 fully connected layers, and 2 dropout layers. The activation function used in all layers is Relu. The activation function used in the output layer is Softmax. The total number of parameters in this architecture is around 60 million.

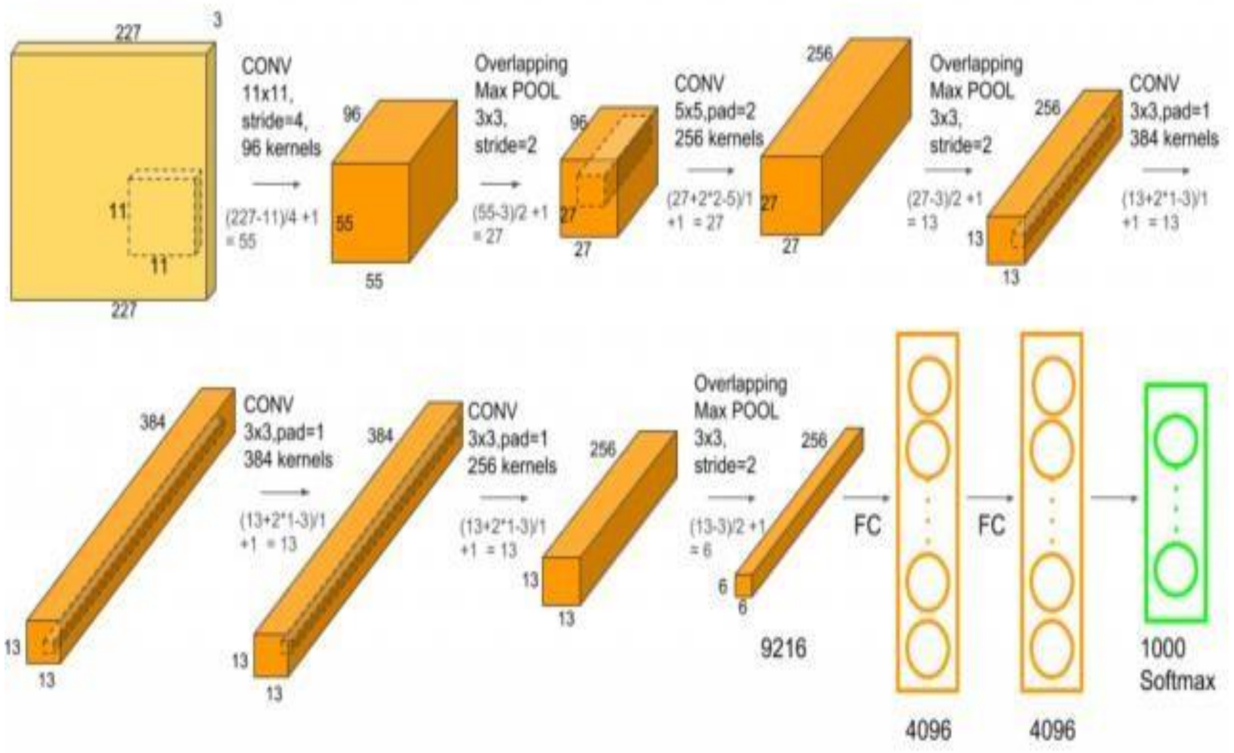


Fig.5 AlexNet Architecture

ZF Net: ZFnet is the CNN architecturethat uses acombinationoffully-connected layers and CNNs. ZFNet was developed by Matthew Zeiler and Rob Fergus. It was the ILSVRC 2013 winner. The network has relatively fewer parameters than AlexNet, but still outperforms it on ILSVRC 2012 classification task by achieving top accuracywithonly1000 imagesper class. It wasan improvement onAlexNet bytweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller. It is based onthe Zeiler and Fergus model, which was trained on the ImageNet dataset. ZF Net CNN architecture consists of a total of seven layers: Convolutional layer, max-pooling layer (downscaling), concatenation layer, convolutional layer with linear activation function, and stride one, dropout for regularization purposes applied before the fully connected output. This CNN model is computationally more efficient than AlexNet by introducing an approximate inference stage through deconvolutional layers in the middle of CNNs.

GoogLeNet: GoogLeNet is the CNN architecture used by Google to win ILSVRC 2014 classification task.It was developed by Jeff Dean, Christian Szegedy, Alexandro Szegedy et al.. It has been shown to have a notably reduced error rate in comparison with previous winners AlexNet (Ilsvrc 2012 winner) and ZF-Net (Ilsvrc 2013 winner). In terms of error rate, the error is significantly lesser than VGG (2014 runner up). It achieves deeperarchitecture by employinga numberof distincttechniques,including1×1convolution and global average pooling. GoogleNet CNN architecture is computationally expensive. To reduce the parameters that must be learned, it uses heavy unpooling layers on top of CNNs to remove spatial redundancyduring training and also featuresshortcut connections betweenthe first two convolutionallayers before adding new filters in later CNN layers. Real-world applications/examples of GoogLeNet CNN architecture include Street View House Number (SVHN) digit recognition task, which is often used as a proxy for roadside object detection. Below is the simplified block diagram representing GoogLeNet CNN architecture:

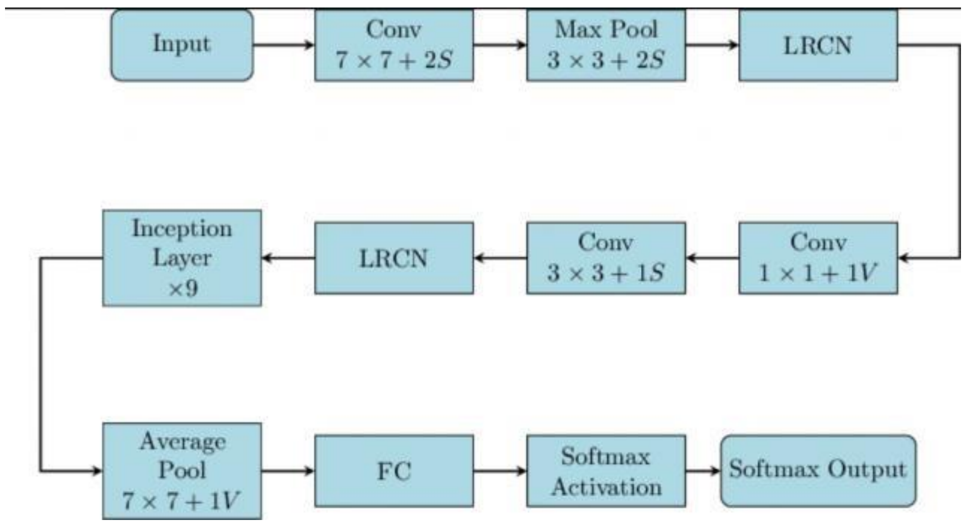


Fig.6 GoogLeNet Architecture

VGGNet: VGGNet is the CNN architecture that was developed by Karen Simonyan, Andrew Zisserman et al. at Oxford University. VGGNet is a 16-layer CNN with up to 95 million parameters and trained on over one billion images (1000 classes). It can take large inputimages of 224 x 224-pixel size for which it has 4096 convolutional features. CNNs with such large filters are expensive to train and require a lot of data, which is the main reason why CNN architectures like GoogLeNet (AlexNet architecture) work better than VGGNet for most image classification tasks where input images have a size between 100 x 100-pixel and 350 x 350 pixels. Real-world applications/examples of VGGNet CNN architecture include the ILSVRC2014 classification task, which was also won by GoogleNet CNN architecture. The VGG CNN model is computationallyefficient and serves as a strong baseline for manyapplications in computer visiondue to its applicability for numerous tasks including object detection. Its deep feature representations are used across multiple neural network architectures like YOLO, SSD, etc. The diagram below represents the standard VGG16 network architecture diagram:

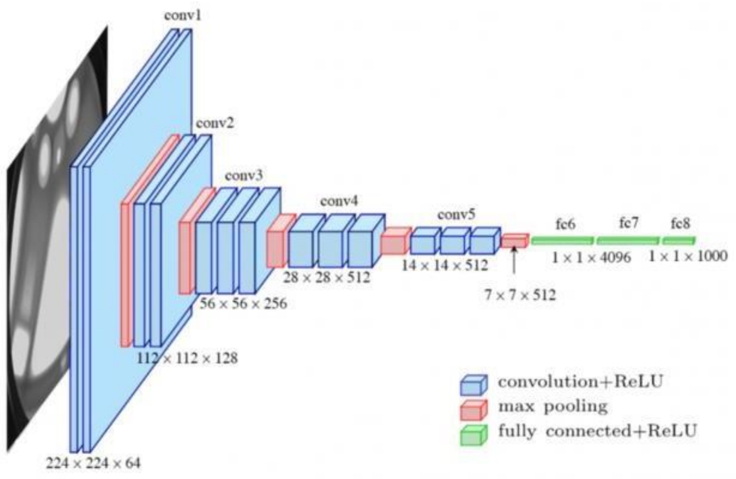


Fig.7 GGNet Architectre

ResNet: ResNet is the CNN architecture hat was developed by Kaiming He et al. to win the ILSVRC 2015 classification task with a top-five error of only 15.43%. The network has 152 layers and over one million parameters, which is considered deep even for CNNs because it would have taken more than 40 days on 32 GPUs to train the network on the ILSVRC 2015 dataset. CNNs are mostly used for image classificationtasks with 1000 classes, but ResNet proves that CNNs can also be used successfully to solve natural language processing problems like sentence completion or machine comprehension, where it was used bythe Microsoft Research Asia teamin2016 and 2017 respectively. Real-life applications/examples of ResNet CNN architecture include Microsoft's machine comprehension system, which has used CNNs to generatethe answers for more than 100k questions in over 20 categories. The CNN architecture ResNet is computationally efficient and can be scaled up or downto match the computational power of GPUs.

MobileNets: MobileNets are CNNs that can be fit on a mobile device to classify images or detect objects with low latency. MobileNets have been developed by Andrew G Trillion et al.. Theyare usually verysmall CNN architectures, which makes them easy to run in real-time using embedded devices like smartphonesand drones. The architecture is also flexible so it has been tested on CNNs with 100-300 layers and it still works better than other architectures like VGGNet. Real-life examples of MobileNets CNN architecture includeCNNsthat isbuilt into AndroidphonestorunGoogle'sMobileVisionAPI, whichcanautomatically identify labels of popular objects in images.

GoogLeNet_DeepDream: GoogLeNet_DeepDream is a deep dream CNN architecture that was developedbyAlexanderMordvintsev,ChristopherOlah,etal..ItusestheInceptionnetworktogenerateimages based on CNN features. The architecture is often used with the ImageNet dataset to generate psychedelic imagesorcreateabstractartworksusinghumanimagination attheICLR2017workshopby DavidHa,et al.

To summarize the different types of CNN architectures described above in an easy to remember form, you can use the following:

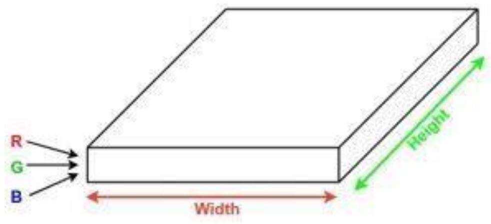
Table1.Different Types of CNN Architectures

Architecture	**Year**	**Key Features**	**Use Case**
LeNet	1998	First successful applications of CNNs, 5 layers (alternating between convolutional and pooling), Used tanh/sigmoid activation functions	Recognizing handwritten and machine-printed characters
AlexNet	2012	DeeperandwiderthanLeNet,UsedReLU activationfunction,Implementeddropoutlayers, UsedGPUsfortraining	Large-scale image recognition tasks
ZFNet	2013	Similar architecture to AlexNet, but with different filter sizes and numbers of filters, Visualizationtechniquesforunderstandingthe network	ImageNetclassification
VGGNet	2014	Deepernetworkswithsmallerfilters(3×3),All convolutional layers have the same depth, Multipleconfigurations(VGG16,VGG19)	Large-scale image recognition
ResNet	2015	Introduced "skip connections" or "shortcuts" to enable training of deeper networks, Multiple configurations(ResNet-50,ResNet-101,ResNet-152)	Large-scale image recognition, won 1st place in the ILSVRC 2015
GoogleLeNet	2014	Introduced Inception module, which allows for moreefficient computationanddeeper networks, multiple versions (Inception v1, v2, v3, v4)	Large-scale image recognition, won 1st place in the ILSVRC 2014

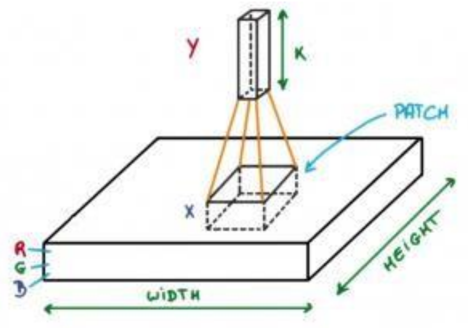
Architecture	Year	Key Features	UseCase
MobileNets	2017	Designed for mobile and embedded vision applications, Uses depthwise separable	Mobile and embedded vision applications, real-time object detection
LeNet	1998	First successful applications of CNNs, 5 layers (alternating between convolutional and pooling), Used tanh/sigmoid activation functions	Recognizing handwritten and machine-printed characters

Working of Convolutional Layers

Convolution Neural Networks or convnets areneural networks thatshare their parameters.Imagineyou have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e the channel as images generally have red, green, and blue channels).



Nowimaginetakingasmallpatchof thisimageandrunningasmallneuralnetwork,calledafilteror kernel on it, with say, K outputs and representing them vertically. Now slide that neural network across the wholeimage, as a result,we will getanotherimage with differentwidths,heights, and depths. Instead of justR,G,andBchannelsnowwehavemorechannelsbutlesserwidthandheight.Thisoperationis called **Convolution**. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.



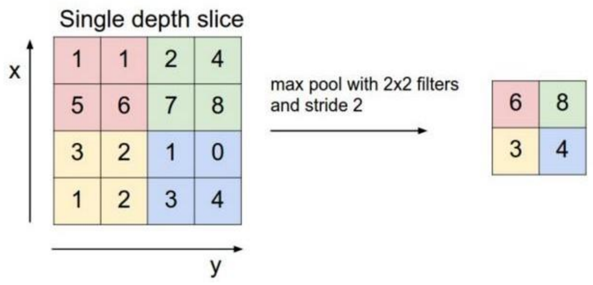
Nowlet’stalkaboutabitofmathematicsthatisinvolvedinthewholeconvolutionprocess.

Convolution layers consist of a set of learnable filters (or kernels) having small widths and heights and the same depth as that of input volume (3 if theinput layer is image input).
For example, if we have to run convolution on an image with dimensions 34x34x3. The possible size of filters can be axax3, where ‘a’ can be anything like 3, 5, or 7 but smaller as compared to the image dimension.
During the forward pass, we slide each filter across the whole input volume step by step where each step is called **stride** (which can have a value of 2, 3, or even 4 for high-dimensional images) and compute the dot product between the kernel weights and patch from input volume.
Asweslideourfilterswe’llgeta2-Doutputforeachfilterandwe’llstackthemtogetherasaresult,we’ll getoutputvolumehavingadepthequaltothenumberoffilters.Thenetworkwilllearn allthefilters.

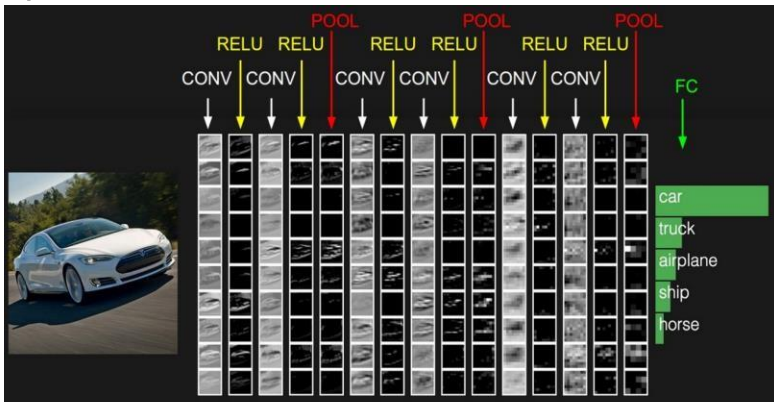
Layers used to build ConvNets

AcompleteConvolutionNeuralNetworksarchitectureisalsoknownasconvNets.AconvNetsisa sequenceoflayers,andeverylayertransformsonevolumetoanotherthroughadifferentiablefunction. Let's take an exampleby running a convNets on of image of dimension 32 x 32 x 3.

- **Input Layers:** It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.
- **Convolutional Layers:** Thisis the layer, which is used to extract the feature fromthe input dataset.It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smallermatricesusually 2×2, 3×3, or 5×5 shape.itslides over theinputimage data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layeris referred adfeaturemaps. Suppose we usea total of 12filters for thislayer we'll getan output volume of dimension 32 x 32 x 12.
- **Activation Layer:** By adding an activation function to the output of the preceding layer, activation layersaddnonlinearity tothenetwork.itwillapplyanelement-wiseactivationfunction tothe output of theconvolution layer. Some common activation functionsare **RELU**,**Tanh**, **Leaky RELU**, etc. Thevolume remainsunchangedhence outputvolumewill havedimensions 32x 32x 12.
- **Pooling layer:**This layer is periodically inserted in the convnets and its main function is to reducethe size of volume which makes the computation fast reduces memory and alsoprevents over-fitting.Twocommontypesofpoolinglayersare **maxpooling** and **averagepooling**.Ifweusea max pool with 2 x 2 filters and stride 2, the resultantvolume will be of dimension 16x16x12.



- **Flattening:** The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer forcategorization or regression.
- **Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.



- **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

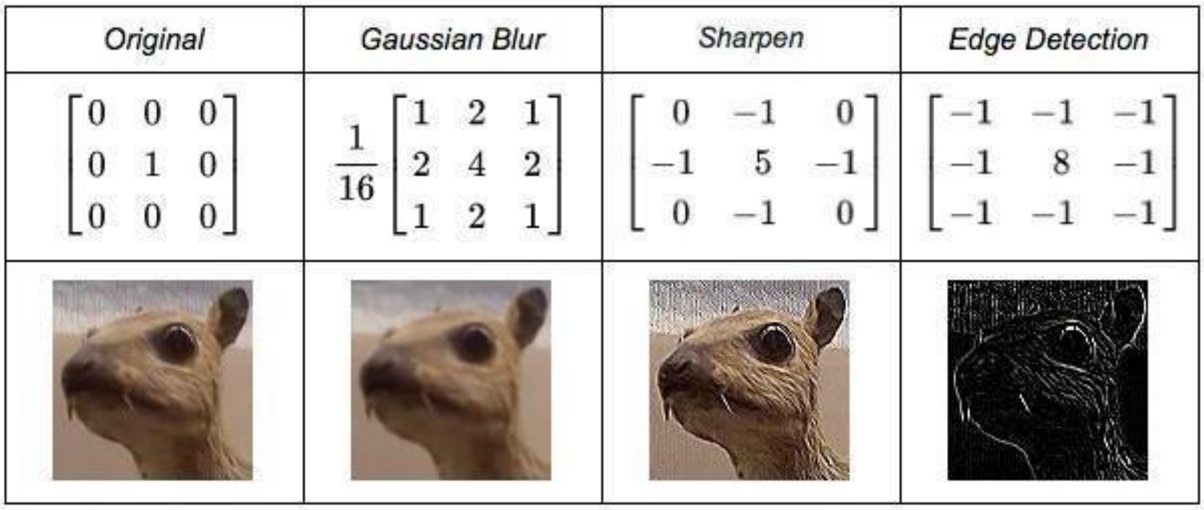
Applications of CNN

- Semantic segmentation: CNNs can classify every pixel in an imageinto differentclasses,for e.g.- different types of vegetation in satellite images.
- Object detection: CNNs can detect objects within an image, for e.g.- identifying the location & the type of vehicle on the road.
- Image classification: CNNs can classify images into different categories, for e.g. identifying objects in a photograph.
- Image captioning: CNNs can generate natural language descriptions of images, for e.g.- describing the objects in a photograph.
- Face recognition - CNNs can recognize & verify the identityof different individuals in images, such as finding people's faces in security footages.
- Medical image analysis -CNNs can identifytumors in medical scans, or indetecting abnormalities in X rays.
- Autonomousvehicles-CNNscanidentify&trackobjects-suchaspedestrians&other vehicles.

2. Convolution Operation:

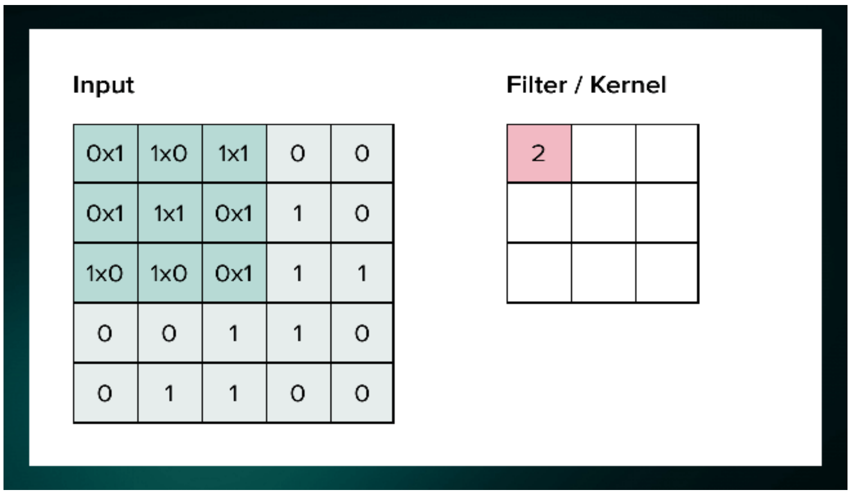
A convolutional neural network, or ConvNet, is just a neural network that uses convolution. To understand the principle, we are going to work with a 2-dimensional convolution first.
Convolution is a mathematical operation that allows the merging of two sets of information. Convolution betweentwo functions in mathematics produces athird function expressing howthe shape ofone function is modified by other.In the case of CNN, convolution is applied to the input data to filter the information and produce a feature map.

This filter isalso calledakernel, orfeaturedetector, and itsdimensionscanbe, forexample, 3x3.Akernelis asmall2D matrixwhosecontentsarebased upontheoperationsto beperformed. Akernelmapsonthe input image bysimple matrix multiplicationandaddition, theoutputobtained is oflowerdimensionsandtherefore easier to work with.

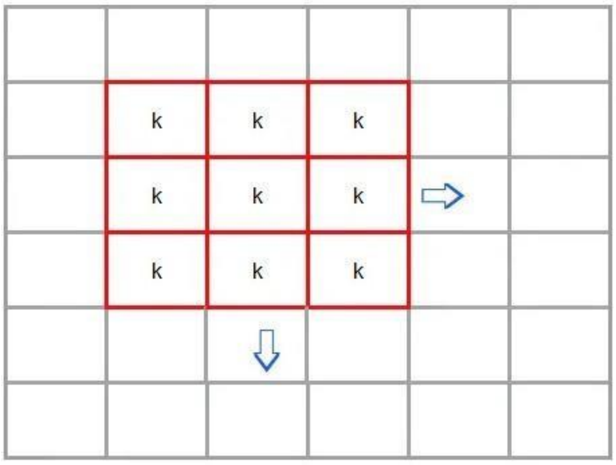


Original	Gaussian Blur	Sharpen	Edge Detection
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

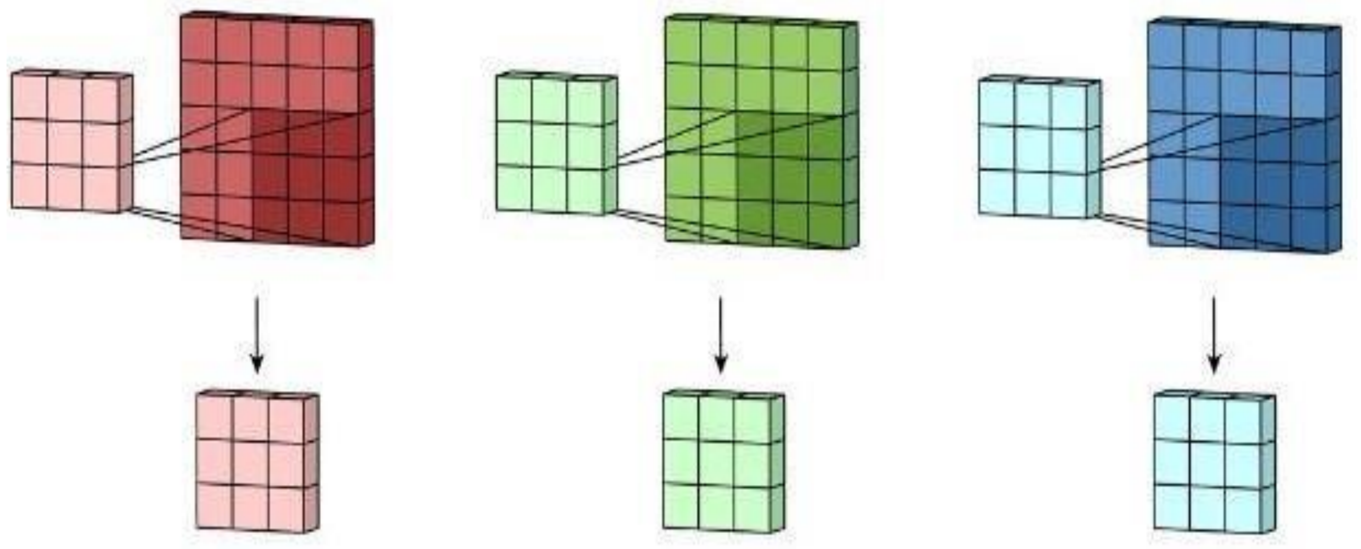
Above is an example of a kernel for applying Gaussian blur (to smoothen the image before processing), Sharpen image (enhance the depth of edges) and edge detection.To perform convolution, the kernel goesover the input image, doing matrix multiplication element after element. The result for each receptive field (the area where convolution takes place) is written down in the feature map.



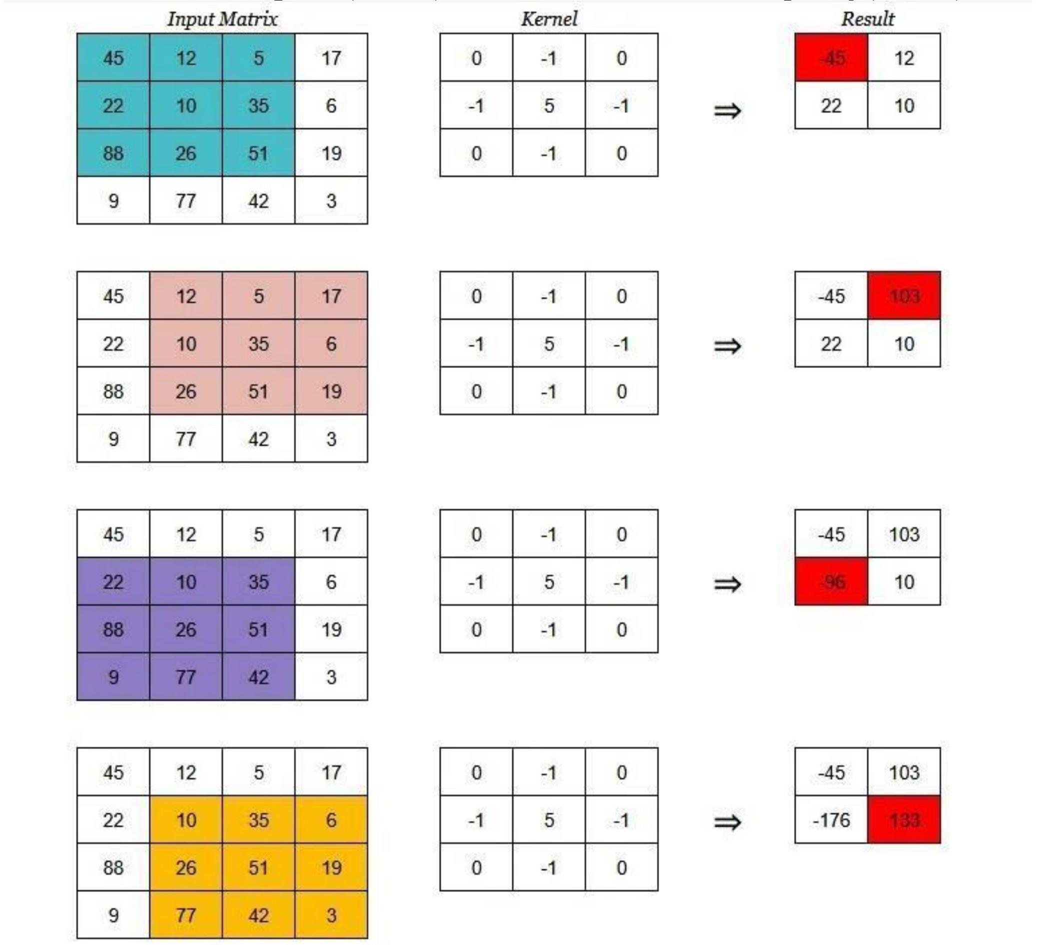
The shape of a kernel is heavily dependent on the input shape of the image and architecture of the entire network, mostly the size of kernels is **(MxM)** i.e., a square matrix. The movement of a kernel is always from left to right and top to bottom.



Stridedefinesbywhatstepdoestokernelmove,forexamplestrideof1makeskernelslidebyone row/columnat atimeandstrideof2 moveskernelby2rows/columns. Wecontinueslidingthe filteruntilthe feature map is complete.



For input images with 3 or more channels such as RGB a filter is applied. Filters are one dimension higher than kernels and can be seen as multiple kernels stacked on each other where everykernel is for a particular channel. Therefore for an RGB image of (32x32) we have a filter of the shape say(5x5x3).



45	12	5	17
22	10	35	6
88	26	51	19
9	77	42	3

0	-1	0
-1	5	-1
0	-1	0

-45	12
22	10

-45	103
22	10

-45	103
-96	10

-45	103
-176	133

Herethe input matrixhas shape 4x4x1 andthe kernelis ofsize 3x3 since the shape ofinput is largerthanthe kernel, weareabletoimplement aslidingwindowprotocolandapplythekerneloverentire input. First entry in the convoluted result is calculated as:
45*0+12*(-1)+5*0+22*(-1)+10*5+35*(-1)+88*0+26*(-1)+51*0 =-45
Wecontinueslidingthefilteruntilthefeaturemapiscomplete.

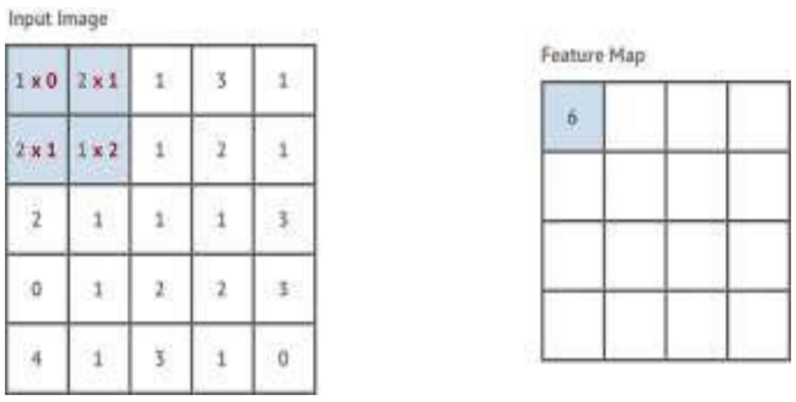
Slidingwindowprotocol:

1. Thenit startsmoving left toright,calculatingthedotproduct andsaving it toanew matrixuntilit has reached the last column.
2. Next,kernelresetsitspositionatfirstcolumnbutnowitslidesonerowtothebottom.Thus following the fashion left-right and top-bottom.
3. Steps2and 3arerepeatedtilltheentireinputhasbeenprocessed.

For a3Dinputmatrixthemovementofthekernelwillbefromfronttoback,lefttorightandtopto bottom.

Sparse Interactions(Connectivity)

Convolutional neural networks are more efficient than simple neural networks — in applications where they apply, because theysignificantlyreduce the number ofparameters whichreduces the required memoryofthe network and improves its statistical efficiency. Theyexploit **feature locality**. Theytryto find patterns in the input data. They stack them to make abstract concepts by their convolution layers. A **Convolution layer** defines a window or filter or kernel by which they examine a subset of the data, and subsequently scans the data looking through this window. We can parameterize the window to look for specific features (e.g. edges within an image). The output they produce focuses solely on the regions of the data which exhibited the feature it was searching for. This is what we call **sparse connectivity**or**sparse interactions** or **sparse weights.** Actually it limits the activated connections at each layer. Inthe example below an5x5 input with a 2x2 filter producesareduced 4x4 output.The first element offeaturemap iscalculated bytheconvolutionof the input area with the filter i.e.



$$1 \times 0 + 2 \times 1 + 2 \times 1 + 1 \times 2 = 6$$

In practice, we don’t explicitly define the filters that our convolutional layer will use; we instead parameterize the filters and let the network learn the best filters to use during training. We do, however, define howmanyfilters, we’lluseat eachlayer— ahyperparameter whichiscalledthe **depth**oftheoutput volume.

Another hyperparameter is the **stride** that defines how much we slide the filter over the data. For example if stride is 1 then we move the window by 1 pixel at a time over the image, when our input is an image. When we use larger values of stride 2 or 3 we allow jumping 2 or pixels at a time. This reduces significantly the output size.

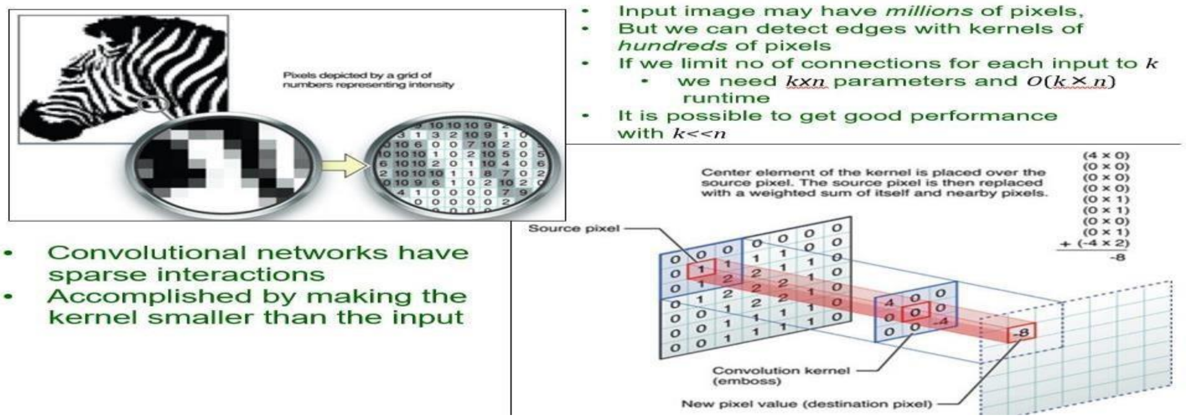
The last hyperparameter is the size of **zero-padding**, when sometimes is convenient to pad the input volume with zeros around the border.

So now we can compute the spatial size of the output volume as a function of the input volume size (W), thereceptivefieldsizeof theConv Layerneurons(F), thestridewith which they areapplied(S),andthe amountofzeropaddingused(P)ontheborder.Theformulaforcalculatinghowmanyneurons“fit”is givenby

$$\frac{W-F+2P}{S}+1$$

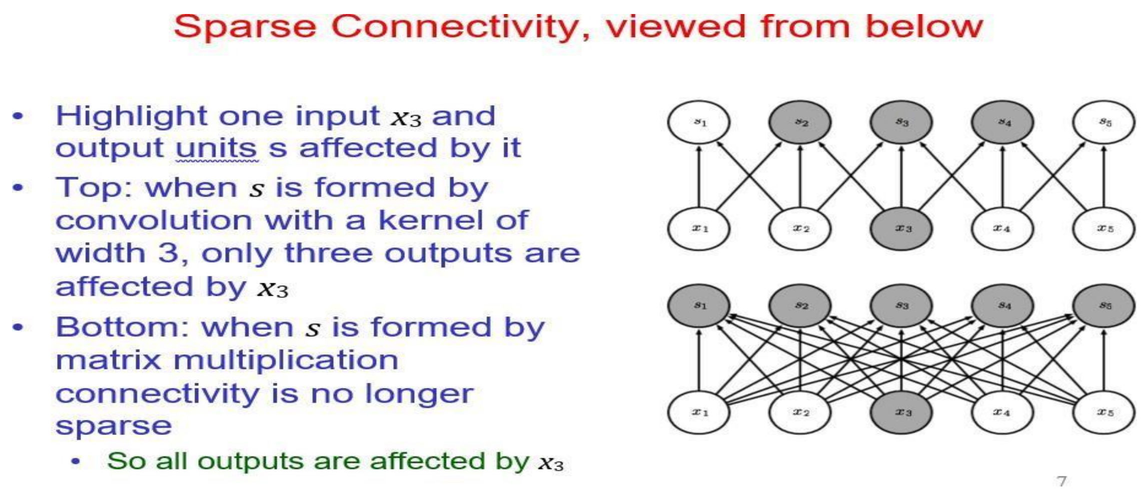
In our previous example for the 5x5 input (W=5) and the 2x2 filter (F=2) with stride 1(S=1) and pad 0 (P=0) we would get a *4x4x (number of filters)* output for each network node.

Trivial neural network layers use matrix multiplication by a matrix of parameters describing the interactionbetweenthe input andoutput unit. This means that everyoutput unit interactswithevery input unit. However, convolution neural networks have ***sparse interaction***. This is achieved by making kernel smaller than the input e.g., an image can have millions or thousands of pixels, but while processing it using kernel we can detect meaningful information that is of tens or hundreds of pixels. This means that we need to store fewer parameters that not only reduces the memory requirement of the model but also improves the statistical efficiency of the model.


Pixels depicted by a grid of numbers representing intensity
Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)
(4 × 0)
(0 × 0)
(0 × 0)
(0 × 0)
(0 × 1)
(0 × 1)
(0 × 0)
(0 × 1)
+ (-4 × 2)
-8

- Input image may have *millions* of pixels,
- But we can detect edges with kernels of *hundreds* of pixels
- If we limit no of connections for each input to k
  - we need kxn parameters and $O(k \times n)$ runtime
- It is possible to get good performance with $k<<n$

- Convolutional networks have sparse interactions
- Accomplished by making the kernel smaller than the input



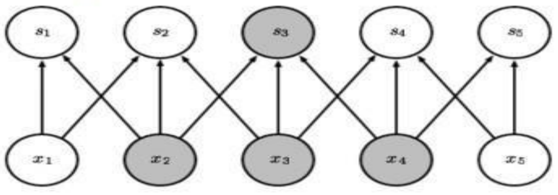
Sparse Connectivity, viewed from below

- Highlight one input x_3 and output units s affected by it
- Top: when s is formed by convolution with a kernel of width 3, only three outputs are affected by x_3
- Bottom: when s is formed by matrix multiplication connectivity is no longer sparse
  - So all outputs are affected by x_3

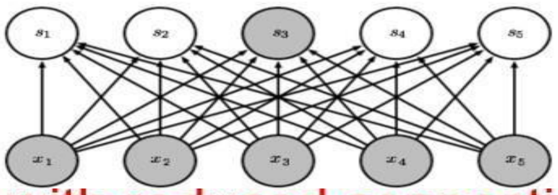
Sparse Connectivity, viewed from above

- Highlight one output s_3 and inputs x that affect this unit
  - These units are known as the receptive field of s_3

When s_3 is formed by convolution with a kernel of width 3

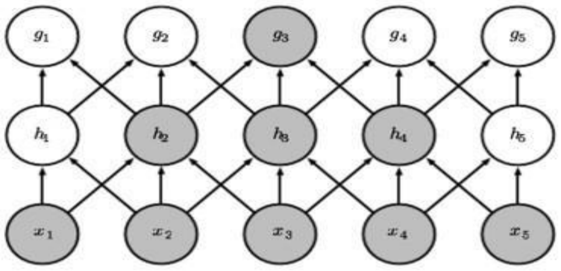


When s_3 is formed by matrix multiplication



Keeping up performance with reduced connections

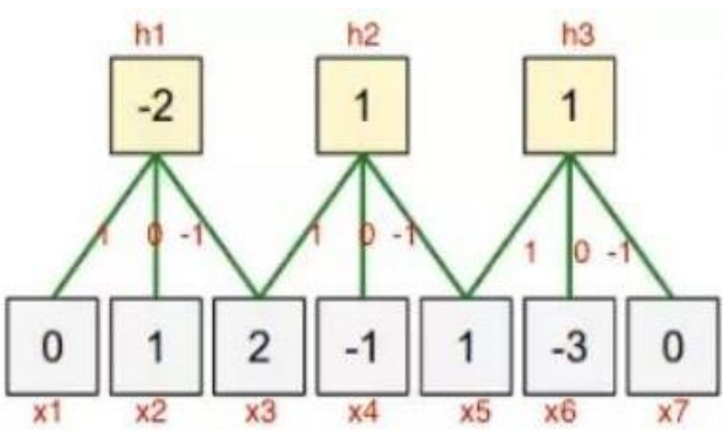
- It is possible to obtain good performance while keeping k several magnitudes lower than m
- In a deep neural network, units in deeper layers may indirectly interact with a larger portion of the input
  - Receptive Field in Deeper layers is larger than the receptive field of units in shallow layers



- This allows the network to efficiently describe complicated interactions between many variables from simple building blocks that only describe sparse interactions

Parameter(Weight)Sharing

Ifcomputing one feature at a spatialpoint (x1, y1) is usefulthen it should also be usefulat some other spatial point say (x2, y2). It means that for a single two-dimensional slice i.e., for creating one activation map,neurons are constrained to use the same set of weights. In a traditional neural network, each element of the weight matrixisusedonceandthenneverrevisited,whileconvolutionnetworkhas ***sharedparameters*** i.e., for getting output,weightsapplied toone input arethe same asthe weight applied elsewhere. Parametersharing is used in the convolutional layers to reduce the number of parameters in the network. For example in the first convolutional layer let'ssaywe haveanoutputof15x15x4 where15 isthesizeoftheoutput and 4the number of filters used in this layer. For each output node in that layer we have the same filter, thus reducing dramatically the storage requirements of the model to the size of the filter.



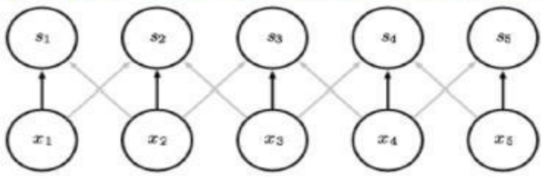
Thesamefilter(weights)(1,0,-1)areusedforthat layer.

Parameter Sharing

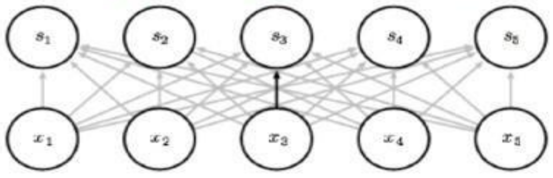
- *Parameter sharing* refers to using the same parameter for more than one function in a model
- In a traditional neural net each element of the weight matrix is used exactly once when computing the output of a layer
  - It is multiplied by one element of the input and never revisited
- Parameter sharing is synonymous with *tied weights*
  - Value of the weight applied to one input is tied to a weight applied elsewhere
- In a Convolutional net, each member of the kernel is used in every position of the input (except at the boundary– subject to design decisions)

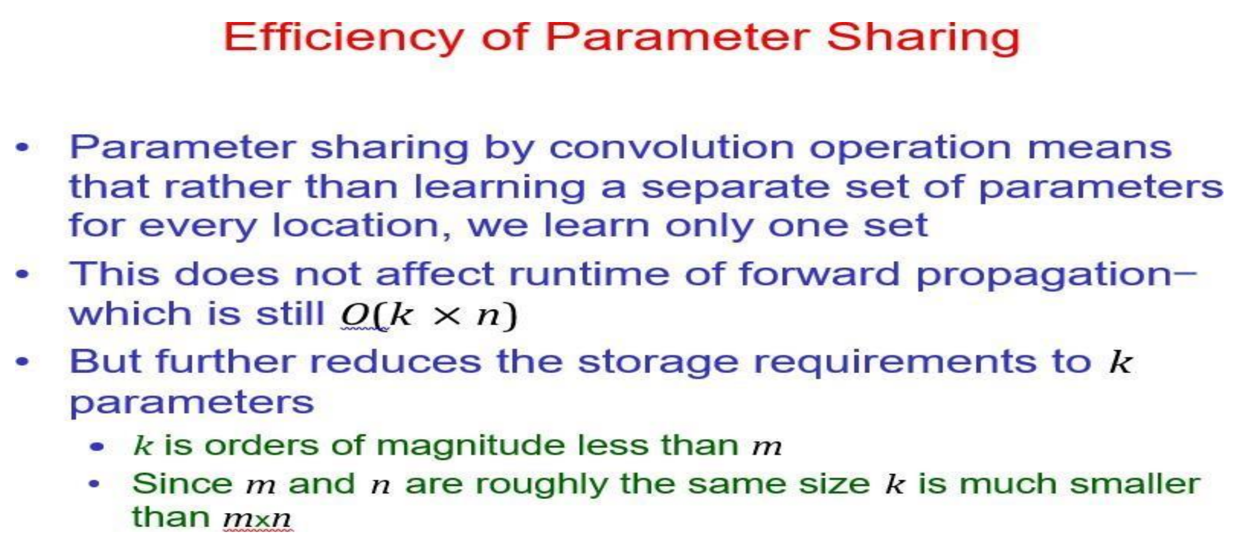
How parameter sharing works

- Black arrrows: connections that use a particular parameter
  1. Convolutional model: Black arrows indicate uses of the central element of a 3-element kernel



  2. Fully connected model: Single black arrow indicates use of the central element of the weight matrix
     - Model has no parameter sharing, so the parameter is used only once





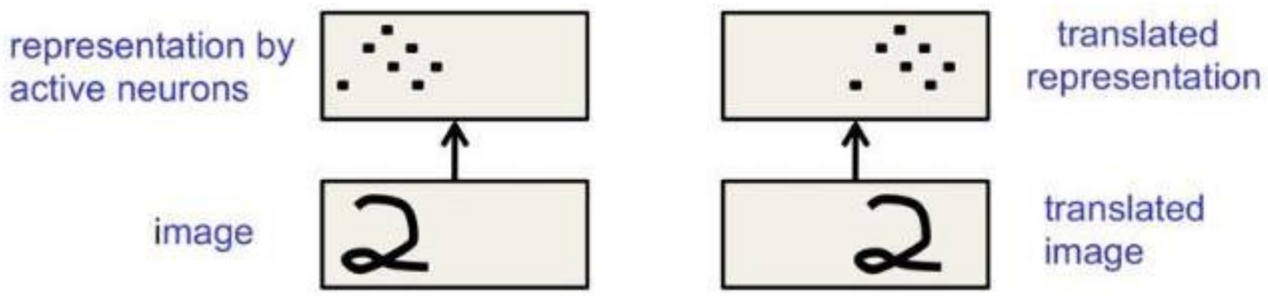
Efficiency of Parameter Sharing

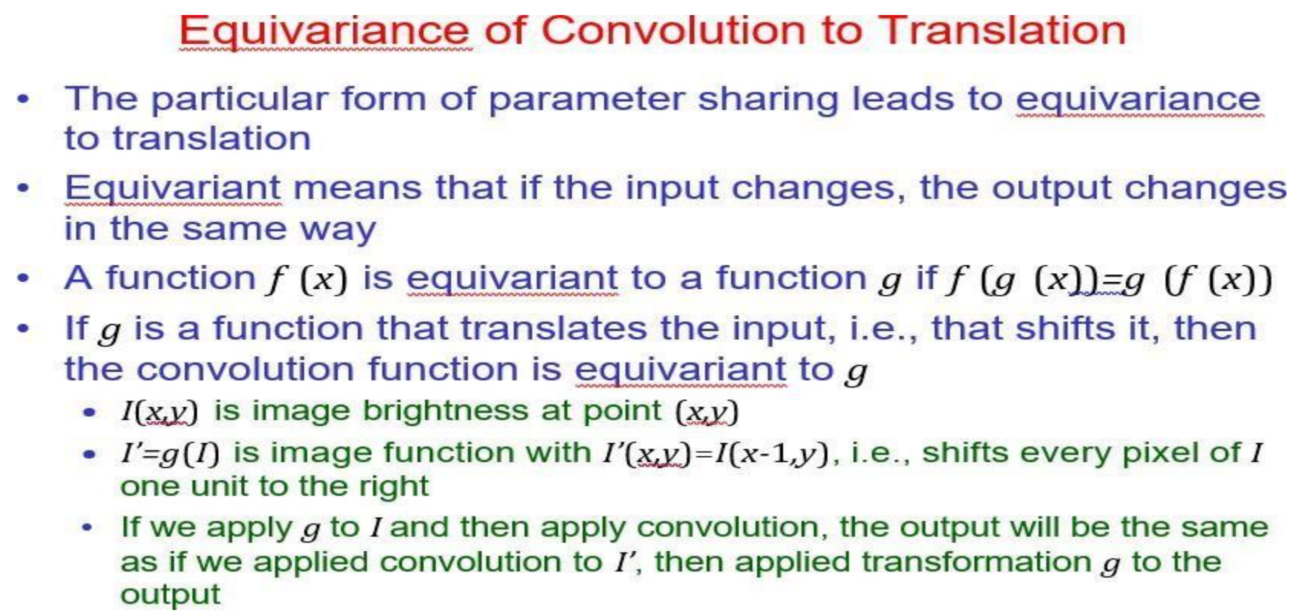
- Parameter sharing by convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set
- This does not affect runtime of forward propagation– which is still $O(k \times n)$
- But further reduces the storage requirements to k parameters
  - k is orders of magnitude less than m
  - Since m and n are roughly the same size k is much smaller than $m \times n$

Equivariant Representations

Equivariant means varying in the similar or equivalent proportion. Due to parameter sharing, the layers of convolution neural network will have a property of ***equivariance to translation***. It says that if we changed the input in a way, the output will also get changed in the same way.

Equivariant to translation means that a translation of input features results in an equivalent translation of outputs. It makes the CNN understand the rotation or proportion change. The equivariance allows the networkto generalize edge, texture, shape, detection in different locations.





Equivariance of Convolution to Translation

- The particular form of parameter sharing leads to equivariance to translation
- Equivariant means that if the input changes, the output changes in the same way
- A function $f(x)$ is equivariant to a function g if $f(g(x))=g(f(x))$
- If g is a function that translates the input, i.e., that shifts it, then the convolution function is equivariant to g
  - $I(x,y)$ is image brightness at point (x,y)
  - $I'=g(I)$ is image function with $I'(x,y)=I(x-1,y)$, i.e., shifts every pixel of I one unit to the right
  - If we apply g to I and then apply convolution, the output will be the same as if we applied convolution to I', then applied transformation g to the output

Pooling

Thepoolingoperationinvolvesslidingatwo-dimensionalfilteroverеachchanneloffeaturemapand summarizing the features lying within the region covered by the filter.
Forafeaturemaphavingdimensionsnhxnwxnc,thedimensionsofoutputobtainedafterapoolinglayeris

(nh–f+1)/sx(nw-f+1)/sx nc

where,

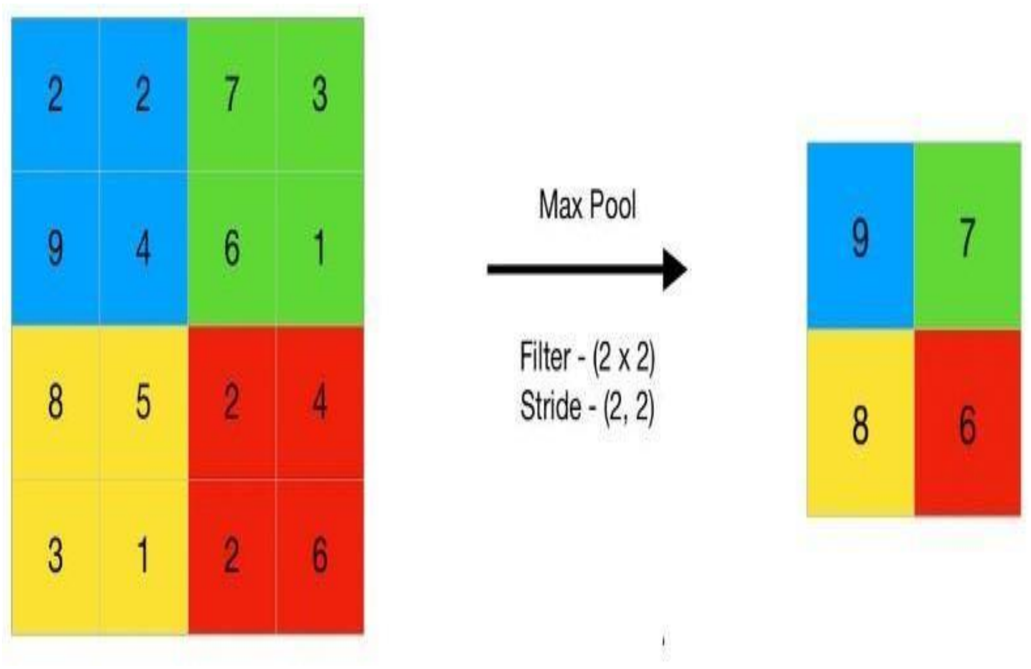
- nh-heightoffeaturemap
- nw–widthoffeaturemap
- nc–number ofchannelsinthefeaturemap
- f-sizeoffilter
- s-stridelength

The pooling layer summarizes the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarized features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

Types of Pooling Layers:

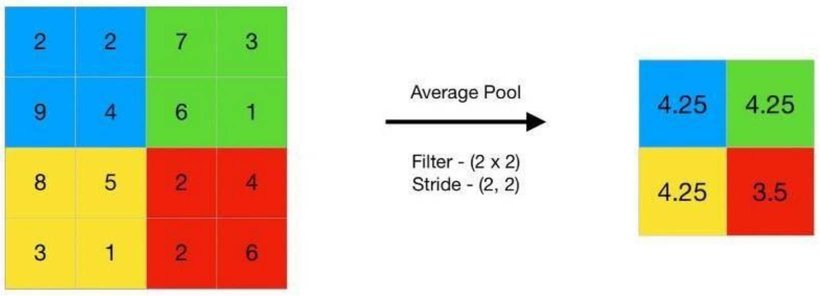
- **Max Pooling**

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.



- ## Average Pooling

Average pooling computes the average of the elements present in the region of feature map covered by thefilter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.



- ## Global Pooling

Global pooling reduces each channel in the feature map to a single value. Thus, an $\mathbf{n_h x\ n_w x\ n_c}$feature map is reduced to**1 x 1 x** $\mathbf{n_c}$feature map. This is equivalent to using a filter ofdimensions$\mathbf{n_h x\ n_w}$i.e. the dimensions of the feature map. Further, it can be either global max pooling or global average pooling.

- ## Global Average Pooling

Considering a tensor of shape **h*w*n**, the output of the Global Average Pooling layer is a single value across **h*w** that summarizes the presence of the feature. Instead of downsizing the patches of the input feature map,the Global Average Pooling layer downsizes the whole **h*w** into 1 value by taking the average.

- ## Global MaxPooling

With the tensor of shape **h*w*n**, the output of the Global Max Pooling layer is a single value across **h*w** that summarizes the presence of a feature. Instead of downsizing the patches of the input feature map, the Global Max Pooling layer downsizes the whole **h*w** into 1 value by taking the maximum.
In convolutional neural networks (CNNs), the pooling layer is a common type of layer that is typically added after convolutionallayers. The pooling layer isusedto reducethe spatialdimensions(i.e.,the widthand height) of the feature maps, while preserving the depth (i.e., the number of channels).

- The pooling layer works bydividing the input feature map into a set of non-overlapping regions, called poolingregions.Eachpoolingregionisthentransformed into asingleoutput value,whichrepresentsthe presence of a particular feature in that region. The most common types of pooling operations are max pooling and average pooling.
- In max pooling, the output value for each pooling region is simply the maximum value of the input values within that region. This has the effect of preserving the most salient features in each pooling region, while discarding less relevant information. Max pooling is often used in CNNs for object recognition tasks, as it helps to identify the most distinctive features of an object, such as its edges and corners.
- In average pooling, the output value for each pooling region is the average of the input values withinthatregion. This hastheeffect ofpreserving moreinformationthanmaxpooling, but mayalso dilutethe most salient features. Average pooling is often used in CNNs for tasks such as image segmentation and object detection, where a more fine-grained representation of the input is required.

Pooling layers are typically used in conjunction with convolutional layers in a CNN, with each pooling layer reducing the spatial dimensions of the feature maps, while the convolutional layers extract increasinglycomplex features from the input. The resulting feature maps are then passed to a fully connected layer, which performs the final classification or regression task.

Advantages of Pooling Layer

Dimensionality reduction: The main advantage of pooling layers is that they help in reducing the spatial dimensions of the feature maps. This reduces the computational cost and also helps in avoiding over-fitting by reducing the number of parameters in the model.

Translation invariance: Pooling layers are also useful in achieving translation invariance in the feature maps. This means that the position of an object in the image does not affect the classification result, as the same features are detected regardless of the position of the object.

Feature selection: Pooling layers can also help in selecting the most important features from the input, as max pooling selects the most salient features and average pooling preserves more information.

Disadvantages of Pooling Layer

- Information loss: One ofthe main disadvantages ofpooling layers is that theydiscard some information fromthe input feature maps, which can be important for the final classification or regression task.
- Over-smoothing: Pooling layers can also cause over-smoothing of the feature maps, whichcanresultinthelossofsomefine-graineddetailsthatareimportantforthefinalclassificationorregression task.
- Hyperparameter tuning: Pooling layers also introduce hyperparameters such as the size of the pooling regions and the stride, which need to be tuned in order to achieve optimalperformance.

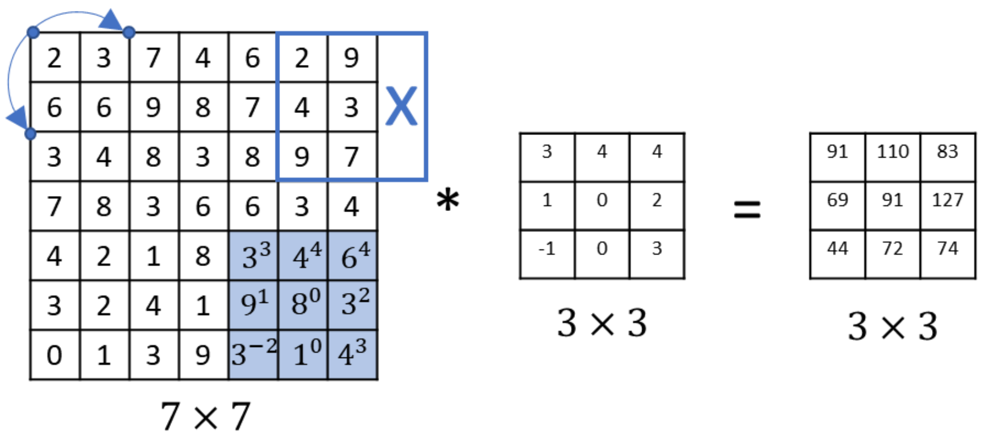
- ConvolutionVariants

3.

The goal of a CNN is to transform the input image into concise abstract representations of the original input. The individual convolutional layers try to find more complex patterns from the previous layer’s observations. The logic is that 10 curved lines would formtwo elipses, which would make an eye.

StridedConvolution

A strided convolution is another basic building block of convolution that is used in Convolutional Neural Networks. Let’s saywe want to convolve this (7 *7) image withthis (3 *3) filter, except, that instead ofdoing it the usual way, we’re going to do it with a stride of (2).



Convolutions with astride of two

This means that we take the element-wise product as usual in this upper left (3 times 3) region, and then multiply and sum elements. That gives us (91). But then instead of stepping the blue box over by one step,we're going to step it over by two steps. It's illustrated how the upper left corner has gone from one dot to anotherjumpingoveroneposition.

In this example we convolve (7 times 7) matrix with a (3 times 3) matrix and we get a (3 times 3) output. The input and output dimensions turns out to be governed bythe following formula:

$$\frac{n-f+2p}{s+1}$$

Ifwe have (n times n) image convolved with an (ftimes f) filter and ifwe use a padding (p) and a stride (s), in this example (s=2), then we end up with an output that is (n-f+2p). Because we're stepping (s) steps at the time instead ofjustonestep atatime,wenowdivideby(s) andadd (1).Inourexample,wehave((7-3+0)/2+1=4/2 +1=3),thatiswhyweendupwiththis(3times 3) output.Noticethat inthis formulaabove, weroundthevalue of this fraction, which generally might not be an integer value, downto the nearest integer.

Tiled Convolution

Tiled Convolutional Neural Networks are an extension to Convolutional Neural Networks that learn k separate convolutionkernels withinthe same layer. These convolutionoperations are applied over every k'th unit (hence the "tiling"). Even k=2 has been shown to give good results. The advantage of this is that through the pooling operation(where layers are "downsampled" bytaking the max, average, or evenstochastic combinationofeach pxp window in the output of a convolutional layer, across many tiles-- where k=p has been shown to give good performance), the tiled layers can provide rotational and scale invariance as well as the translational invariance that comes from having convolutional layers in the first place.

Moreover, each convolution operation is effectively learning an additional feature (or map), which is a learned representation of the training data, and the tiled layers, like convolutional layers, also still have a relatively small numberoflearned parameters.Inessence, it isthe pooling operationoverthese multiple "tiled" mapsthat allows the network to learn invariances over scaling and rotation.

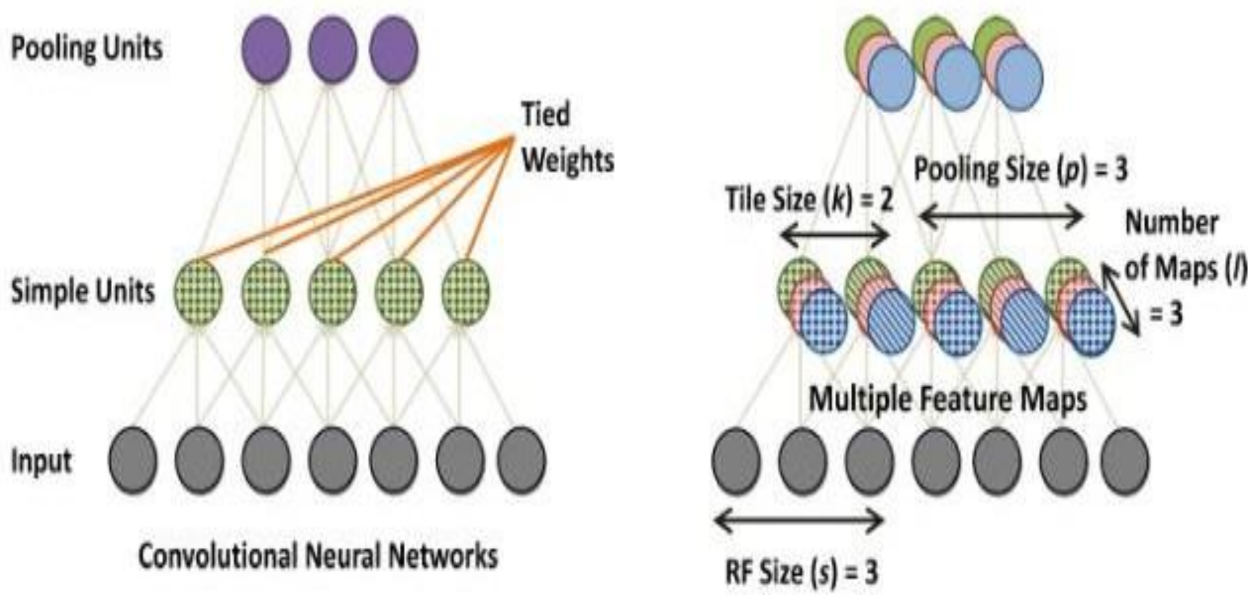
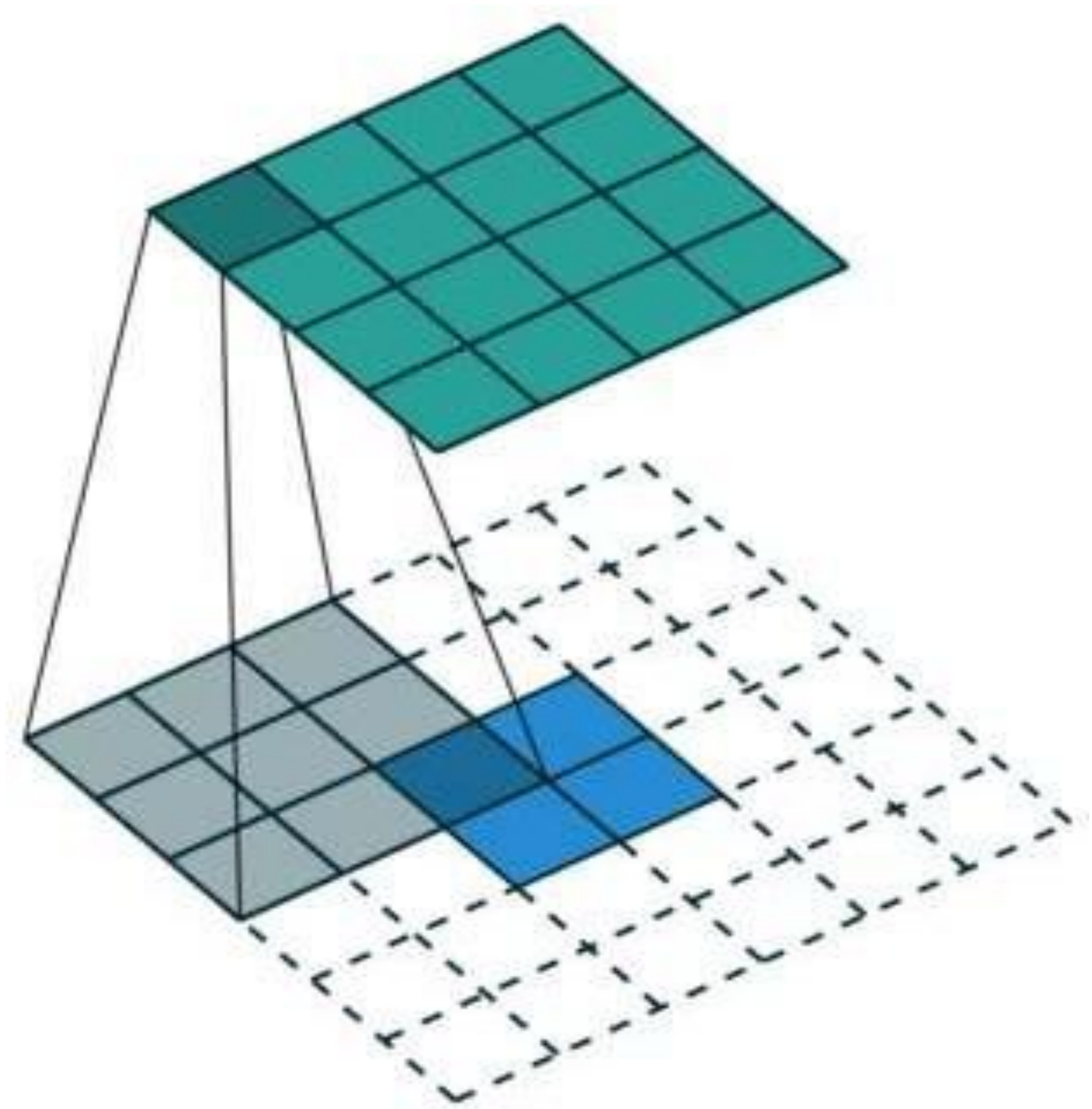
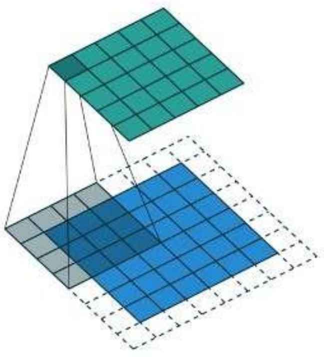


Fig.8 :CNN vs Tiled CNN

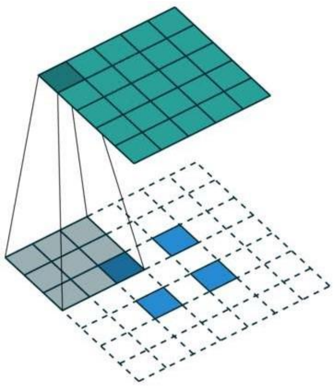
The transposed convolutional layer, unlike the convolutional layer, is up sampling in nature. Transposed convolutions are usually used in auto-encoders and GANs, or generally any network that must reconstruct an image.
The word **transpose** means to cause two or more things to switch places with each other, and in the context of convolutional neural networks, this causes the input and the output dimensions to switch.
In a transposed convolution, instead ofthe input being larger than the output, the output is larger. An easyway to think of it is to picture the input being padded until the corner kernel can just barely reach the corner ofthe input.



When down sampling and up sampling techniques are applied to transposed convolutional layers, their effects are reversed. The reason for this is for a network to be able to use convolutional layers to compress the image, then transposed convolutional layers with the exact same down sampling and up sampling techniques to reconstruct the image.
When padding is 'added' to the transposed convolutional layer, it**seems** as if padding is removed from the input, and the resulting output becomes smaller.



Without padding, the output is 7x7, but with padding on both sides, it is 5x5. When strides are used, theyinstead affect the input, instead of the output.



Strides(2,2)increasestheoutputdimensionfrom3x3to5x5.

- **Transposed Convolution vs Deconvolution**

Deconvolution is a term floating around next to transposed convolutions, and the two are often confused for each other. Many sources use the two interchangeably, and while deconvolutions do exist, they are not very popular in the field of machine learning.

A deconvolution is a mathematical operation that reverses the effect ofconvolution. Imagine throwing an input through a convolutional layer, and collecting the output. Now throw the output through the deconvolutional layer, and you get back the exact same input. It is the inverse ofthe multivariate convolutional function.

On the other hand, a transposed convolutional layer only reconstructs the spatial dimensions of the input. In theory, this is finein deep learning, as it can learn its own parameters through gradient descent, however, it does not give the same output as the input.

Dilated Convolution

Dilated convolution, also knownas atrousconvolution, is atype ofconvolutionoperationused inconvolutional neuralnetworks(CNNs)that enables the networkto have a larger receptive field without increasing the number of parameters. It is a technique that expands the kernel (input) by inserting holes between its consecutive elements. In simpler terms, it is the same as convolution but it involves pixel skipping, so as to cover a larger area of the input.

Inaregular convolutionoperation, a filter ofa fixedsizeslidesover the input featuremap, andthe values inthe filter are multiplied with the corresponding values in the inputfeature map to produce a single output value. The receptive field ofa neuron inthe output feature map is defined as the area in the input feature map that the filter can “see”. The size of the receptive field is determined by the size of the filter and the stride of the convolution.

In contrast, in a dilated convolution operation, the filter is “dilated” by inserting gaps between the filter values. The dilation rate determines the size of the gaps, and it is a hyperparameter that can be adjusted. When the dilation rate is 1, the dilated convolution reduces to a regular convolution.

The dilation rate effectively increases the receptive field of the filter without increasing the number of parameters, because the filter is still the same size, but with gaps between the values. This can be useful in situations where a larger receptive field is needed, but increasing the size ofthe filter would lead to an increase in the number of parameters and computational complexity.
Dilatedconvolutionshavebeenusedsuccessfullyinvariousapplications,suchassemanticsegmentation,where a larger context is needed to classify each pixel, and audio processing, where the network needs to learnpatterns with longer time dependencies.
An additional parameter ***l*** (dilation factor) tells how much the input is expanded. In other words, based on the value of this parameter, ***(l-1)*** pixels are skippedin the kernel.**Figure 9** depicts the differencebetween normal vs dilated convolution. In essence, normal convolution is just a **1-dilated convolution**.

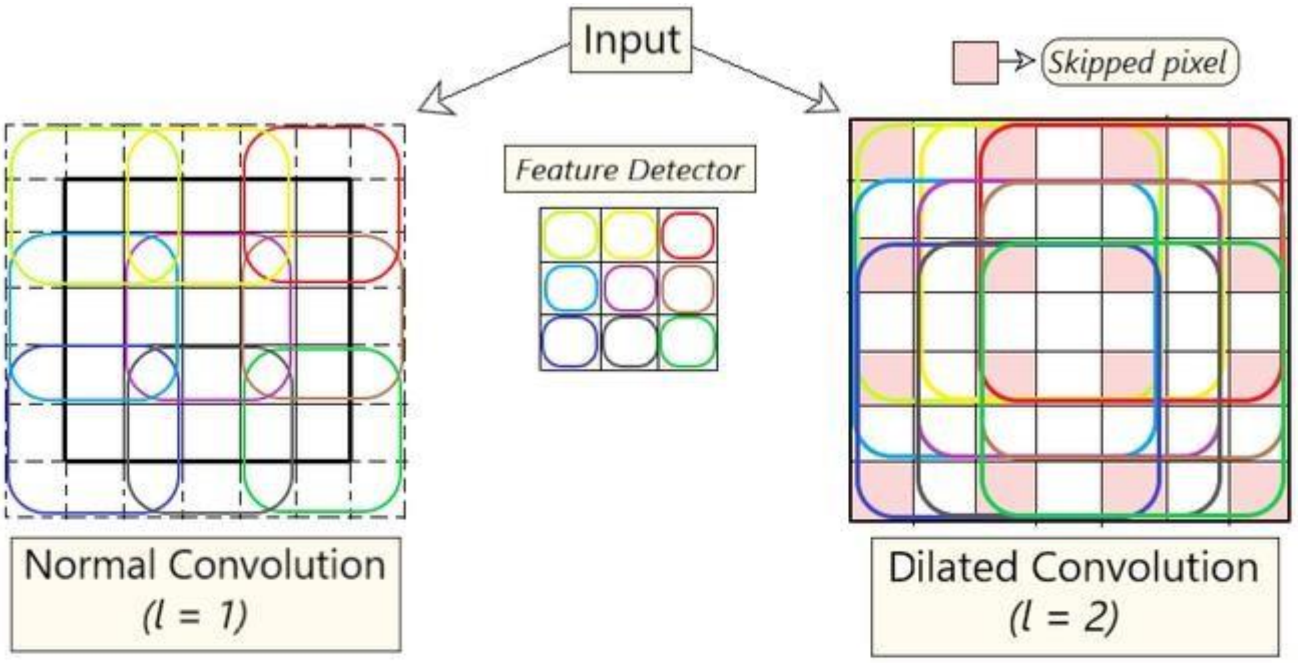


Fig9:NormalConvolutionvsDilated Convolution

Dilatedconvolutionhelpsexpandtheareaofthe inputimagecoveredwithout pooling.Theobjective istocover more information fromthe output obtained with everyconvolution operation. This method offers a wider field of view at the same computational cost. We determine the value of the dilation factor ***(l)*** byseeing how much information is obtained with each convolution on varying values of ***l***.
Byusing this method, we are able to obtain more information without increasing the number ofkernel parameters. In**Fig9**,the imageontheleft depictsdilatedconvolution. Onkeepingthevalueof $l=2$,weskip1 pixel($l-1$ pixel) while mapping the filter onto the input, thus covering more information in each step.

- **Formula Involved:**

$$(F_{*lk})(p)=\sum_{(s+lt=p)}F(s)k(t)$$

```
where,
F(s)=Input
k(t)=AppliedFilter
*l=l-dilatedconvolution
(F*lk)(p)=Output
```

4. CNN Learning

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

NonLinearity Functions

Nonlinear functions play a crucial role in Convolutional Neural Networks (CNNs) by introducing complex transformationsthat allowthe networktocaptureintricatepatternsandrelationships inthedata. InCNNs, these nonlinear functions are typically applied after convolutional and pooling layers to introduce nonlinearity intothe network architecture. The most commonly used nonlinear function in CNNs is the Rectified Linear Unit (ReLU), but there are other options as well. Here are some common nonlinear activation functions used in CNNs:

- **Rectified Linear Unit (ReLU):**The ReLU activation function is defined as f(x) = max(0, x). It replaces all negative values with zero and keeps positive values unchanged. ReLU is computationally efficient and helps mitigate the vanishing gradient problem, allowing deeper networks to be trained effectively.
- **Leaky ReLU:** The Leaky ReLU is an extension of the ReLU function that allows a small gradient for negative values to prevent neurons frombecoming inactive. It's defined as f(x) = x if x> 0, and f(x) = αx if x < 0, where α is a small positive constant.
- **Parametric ReLU (PReLU):** PReLU is similar to Leaky ReLU, but the slope for negative valuesis learned during training rather than being a fixed constant. This can lead to improvedperformance, especially on large datasets.
- **Exponential LinearUnit (ELU):** The ELU activation function is defined as f(x) = xfor x> 0, and f(x) = α * (exp(x) - 1) for x < 0, where α is a positive constant. ELU can help alleviate thevanishing gradient problem and produce smoother gradients.
- **Scaled Exponential Linear Unit (SELU):** SELU is a variant of ELU that aims to maintain mean and variance stability in neural networks. It's designed to automatically adjust its parameters to achieve this stability, making it particularly useful in deeper architectures.
- **Hyperbolic Tangent (tanh):** Thetanhactivation functionsquashes valuestotherangeof -1 to 1.It is symmetric around the origin, so it can produce both positive and negative values.

Here we will look into the ReLU activation function, more specificallyabout it’s non - linear behaviour. ReLU is anacronym for Rectified Linear Unit. It is the most commonlyused activation function. The functionreturns 0 if it receives any negative input, but for anypositive value x it returns that value back. So, Mathematically it can be expressed as:- f(x) = max(0,x) Basically, it setsanything lessthanorequalto 0(negative numbers) to be 0. Andkeepsallthesamevaluesfor anyvalues>0.GraphicalrepresentationofReLUfunctionis:

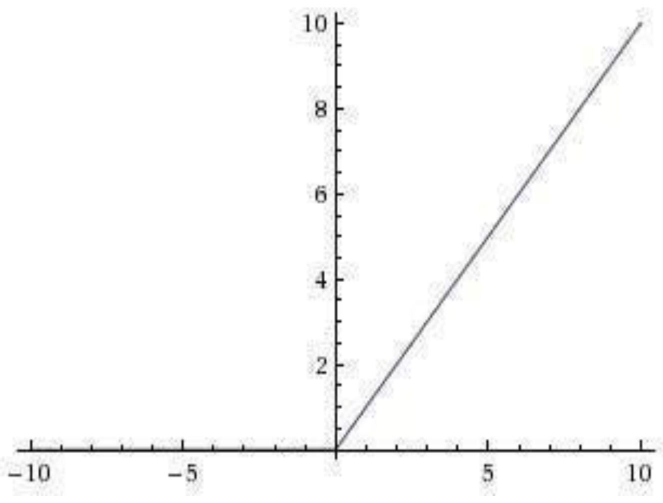


Fig.10RELUActivationFunction

From the graphical representation, we observe that it is a very simple function. This means, it is composed of two pieces of straight lines only which are separated by y-axis of the graph. Also it includes very simple mathematical operations that are why it is less computationally expensive than other activation functions. Derivative of ReLU function By just looking into the equation of ReLU function it's not clear that what the derivative will be, However let's look into the graph so that it mayget clear to me about it's derivatives. Let's draw a graph of ReLU function where x is ranging from -4 to +4, and increment by 1 unit. Similarly y axis is labelled as f(x), value of function at x.

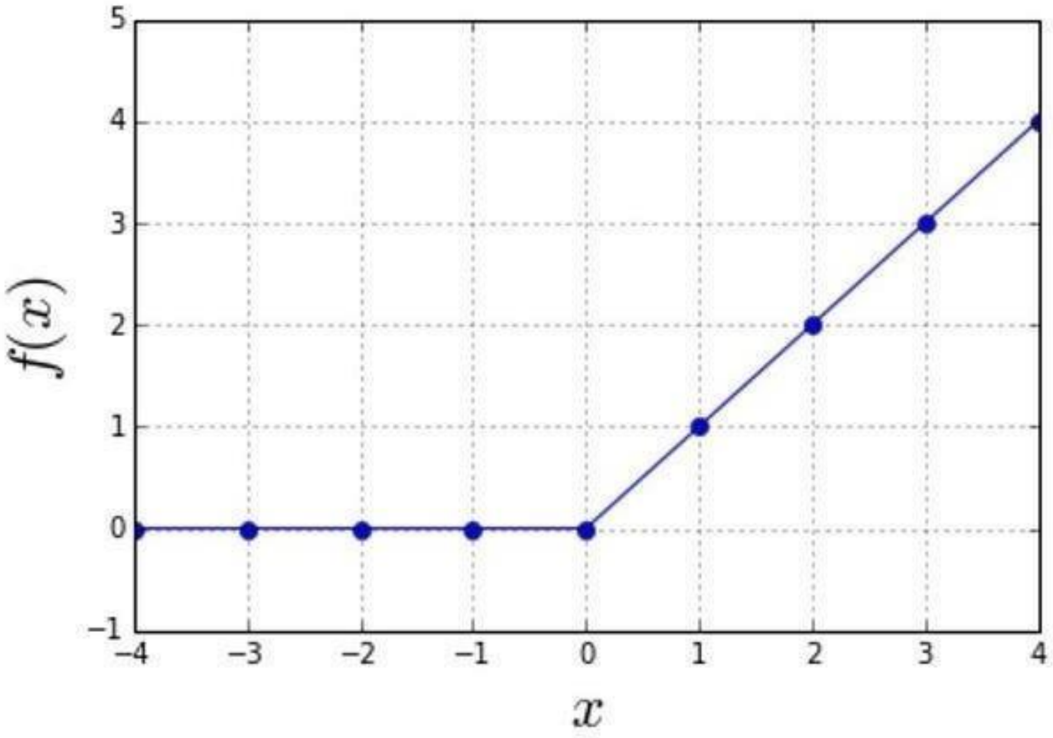


Fig.11 RELU function

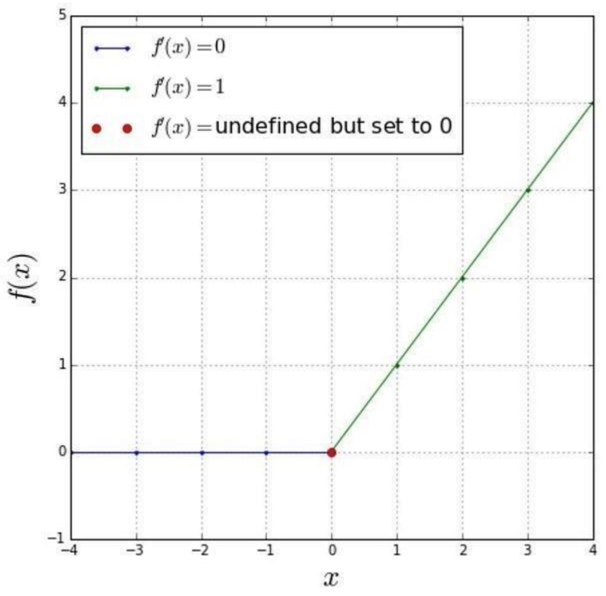


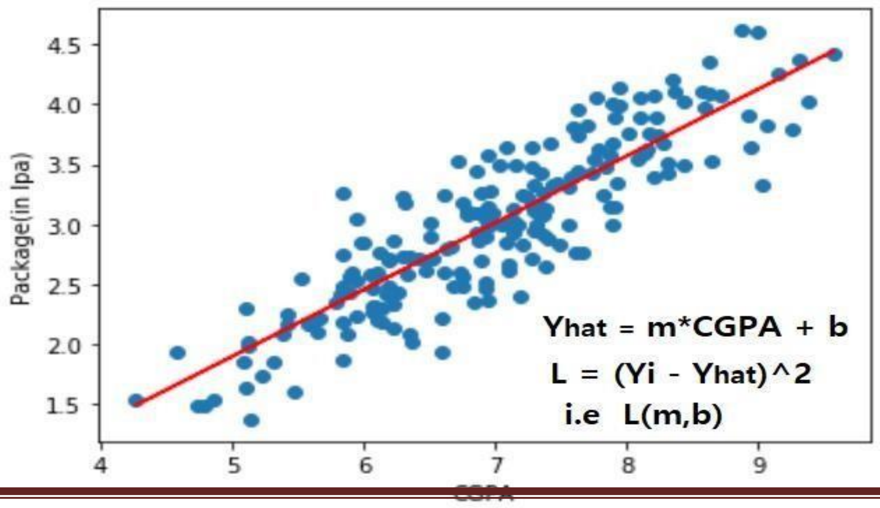
Fig.12DerivativefunctionofRELU

As we know the derivative of function is defined as the slope of the function at certain point.So you can see that the function is mostly differentiable. If x is greater than 0 the derivative is 1 and if x is less than zero the derivative is 0. But when x = 0, the derivative does not exist. There are two ways to deal with this.First, you can just arbitrarily assign a value for the derivative of y = f(x) when x = 0. A second alternative is, instead of using the actual y = f(x) function, use an approximation to ReLU which is differentiable for all values of x. Anyway, Tillnow wewere getting confused that actuallywhat the ReLU function is non-differentiable around 0, but the slope is always either 0 (for negative values) or 1 (for positive values). That's why the ReLU function is Non-Linear. Intuitively, we can understand that as The ReLU is an activation function and the purpose of activation function is to introduce non-linearity in the neural network.

Loss Functions

The loss function is very important in machine learning or deep learning. In mathematical optimization and or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. In simple terms, the Loss function is a method of evaluating how well your algorithm is modeling your dataset. It is a mathematical function of the parameters of the machine learning algorithm.

In simple linear regression, prediction iscalculated using slope(m) and intercept(b).The loss function forthis is the (Yi – Yihat)^2 i.e loss function is the function of slope and intercept.



Loss Function in Deep Learning

- Regression
  - MSE(Mean Squared Error)
  - MAE(Mean Absolute Error)
  - Hubber loss
- Classification
  - Binary cross-entropy
  - Categorical cross-entropy

A. Regression Loss

1. Mean Squared Error/Squared loss/L2loss

The Mean Squared Error (MSE) is the simplest and most common loss function. To calculate them MSE, you take the difference between the actual value and model prediction, square it, and average it across the whole dataset.

$$\text{MSE} = \frac{1}{N}\sum_{i}^{N}(Yi - \hat{Y}i)^2$$

Shree SATHYAM COLLEGE OF ENGG & TECH

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

AD3501 / DEEPLEARNING - NOTES

AD3501- DEEP LEARNING

UNIT III RECURRENT NEURAL NETWORKS

Unfolding Graphs – RNN- Design Patterns: Acceptor -- Encoder --Transducer; Gradient Computation -- Sequence Modeling Conditioned on Contexts -- Bidirectional RNN -- Sequence to Sequence RNN – Deep Recurrent Networks -- Recursive Neural Networks -- Long Term Dependencies; Leaky Units: Skip connections and dropouts; Gated Architecture: LSTM.

RECURRENT NEURAL NETWORKS

Introduction to RNN

Traditional neural networks mainly have independent input and output layers, which make them inefficient when dealing with sequential data. Hence, a new neural network called **Recurrent Neural Network**, introduced to store results of previous outputs in the internal memory. These results are then fed into the network inputs in order to predict the output of the layer. This allows it to be used in applications like pattern detection, speech and voice recognition, natural language processing, and time series prediction.
Below is how we can convert a Feed-Forward Neural Network into a Recurrent Neural Network:

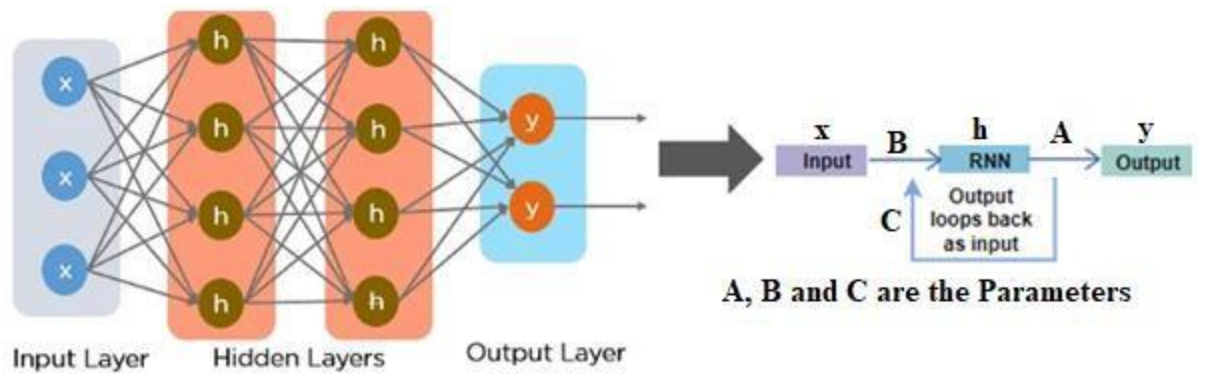
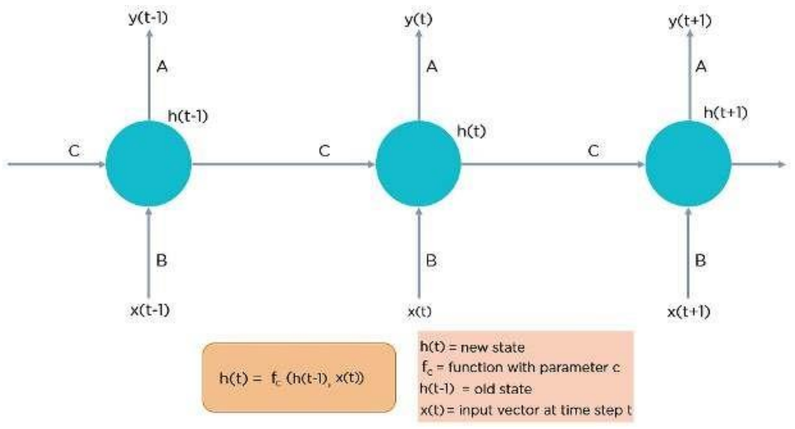


Fig: Simple Recurrent Neural Network
RNN has hidden layers that act as memory locations to store the outputs of a layer in a loop.
Here, "x" is the input layer, "h" is the hidden layer (act as memory locations to store the outputs of a layer in a loop), and "y" is the output layer. A, B, and C are the network parameters used to improve the output of the model. At any given time t, the current input is a combination of input at x(t) and x(t-1). The output at any given time is fetched back to the network to improve on the output.



y(t-1)
y(t)
y(t+1)
A
h(t-1)
h(t)
h(t+1)
C
B
x(t-1)
x(t)
x(t+1)
h(t) = f_c (h(t-1), x(t))
h(t) = new state
f_c = function with parameter c
h(t-1) = old state
x(t) = input vector at time step t

Why Recurrent Neural Networks?

RNN were created because there were a few issues in the feed-forward neural network:

- Cannot handle sequential data
- Considers only the current input
- Cannot memorize previous inputs

The solution to these issues is the RNN. An RNN can handle sequential data, accepting the current input data, and previously received inputs. RNNs can memorize previous inputs due to their internal memory.

How Does Recurrent Neural Networks Work?

In Recurrent Neural networks, the information cycles through a loop to the middle hidden layer.

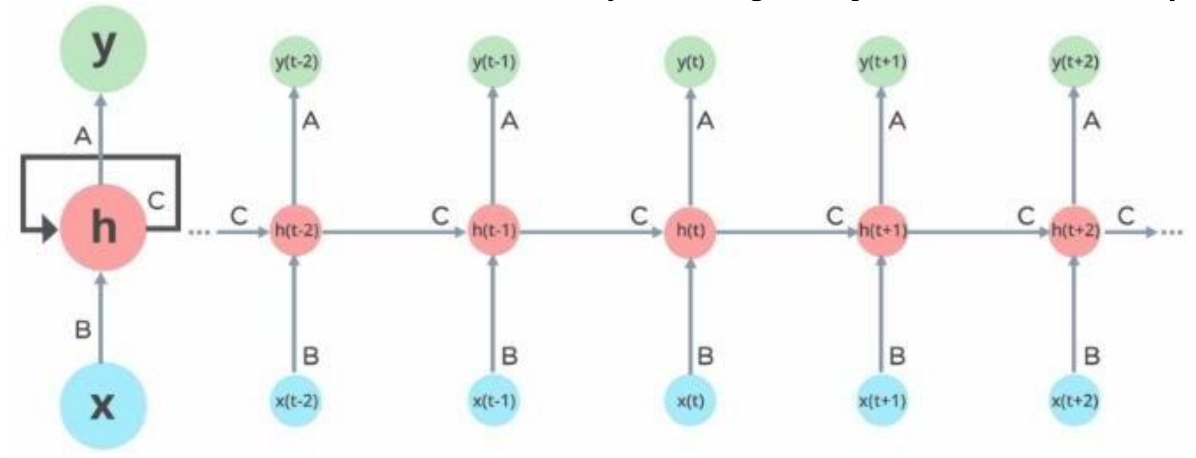


Fig: Working of Recurrent Neural Network

The input layer ‘x’ takes in the input to the neural network and processes it and passes it onto the middle layer.

The middle layer ‘h’ can consist of multiple hidden layers, each with its own activation functions and weights and biases. If we have a neural network where the various parameters of different hidden layers are not affected by the previous layer, ie: the neural network does not have memory, then we can use a recurrent neural network.

The Recurrent Neural Network will standardize the different activation functions and weights and biases so that each hidden layer has the same parameters. Then, instead of creating multiple hidden layers, it will create one and loop over it as many times as required.

Feed-Forward Neural Networks vs Recurrent Neural Networks

A feed-forward neural network allows information to flow only in the forward direction, from the input nodes, through the hidden layers, and to the output nodes. There are no cycles or loops in the network. Below is how a simplified presentation of a feed-forward neural network looks like:

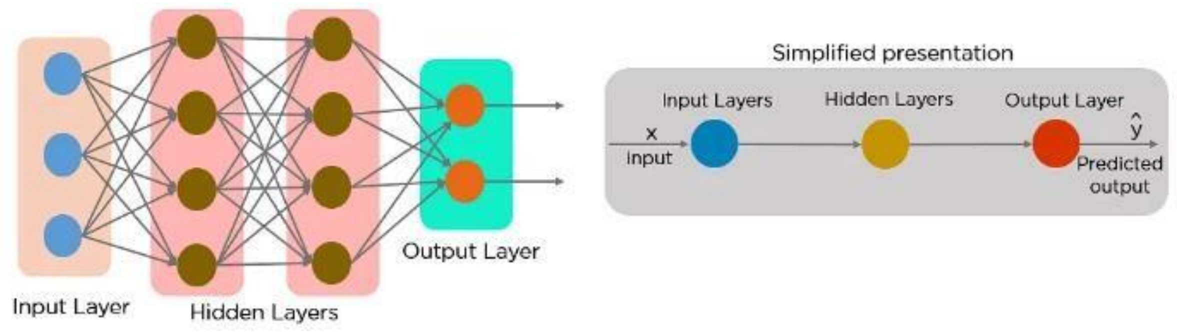


Fig: Feed-forward Neural Network

In a feed-forward neural network, the decisions are based on the current input. It doesn't memorize the past data, and there's no future scope. Feed-forward neural networks are used in general regression and classification problems.

Applications of Recurrent Neural Networks

Image Captioning: RNNs are used to caption an image by analysing the activities present.

<u>Time Series Prediction</u>: Any time series problem, like predicting the prices of stocks in a particular month, can be solved using an RNN.

<u>Natural Language Processing</u>: Text mining and Sentiment analysis can be carried out using an RNN for Natural Language Processing (NLP).

NATURAL
LANGUAGE
PROCESSING
When it rains, look for rainbows.
When it's dark, look for stars.
Positive Sentiment
Natural Language Processing

Machine Translation: Given an input in one language, RNNs can be used to translate the input into different languages as output.

Here the person is speaking in English and it is getting translated into Chinese, Italian, French, German and Spanish languages
Machine Translation

Advantages of Recurrent Neural Network

Recurrent Neural Networks (RNNs) have several advantages over other types of neural networks, including:

Ability to Handle Variable-Length Sequences: RNNs are designed to handle input sequences of variable length, which makes them well-suited for tasks such as speech recognition, natural language processing, and time series analysis.

Memory of Past Inputs:RNNs have a memory of past inputs, which allows them to capture information about the context of the input sequence. This makes them useful for tasks such as language modelling, where the meaning of a word depends on the context in which it appears.

Parameter Sharing: RNNs share the same set of parameters across all time steps, which reduce the number of parameters that need to be learned and can lead to better generalization.

Non-Linear Mapping: RNNs use non-linear activation functions, which allow them to learn complex, non-linear mappings between inputs and outputs.

Sequential Processing: RNNs process input sequences sequentially, which makes them computationally efficient and easy to parallelize.

Flexibility: RNNs can be adapted to a wide range of tasks and input types, including text, speech, and image sequences.

Improved Accuracy:RNNs have been shown to achieve state-of-the-art performance on a variety of sequence modeling tasks, including language modeling, speech recognition, and machine translation.

These advantages make RNNs a powerful tool for sequence modelling and analysis, and have led to their widespread use in a variety of applications, including natural language processing, speech recognition, and time series analysis.

Disadvantages of Recurrent Neural Network

Although Recurrent Neural Networks (RNNs) have several advantages, they also have some disadvantages. Here are some of the main disadvantages of RNNs:

Vanishing and Exploding Gradients:RNNs can suffer from the problem of vanishing or exploding gradients, which can make it difficult to train the network effectively. This occurs when the gradients of the loss function with respect to the parameters become very small or very large as they propagate through time.

Computational Complexity:RNNs can be computationally expensive to train, especially when dealing with long sequences. This is because the network has to process each input in sequence, which can be slow.

Difficulty in Capturing Long-Term Dependencies: Although RNNs are designed to capture information about past inputs; they can struggle to capture long-term dependencies in the input sequence. This is because the gradients can become very small as they propagate through time, which can cause the network to forget important information.

Lack of Parallelism:RNNs are inherently sequential, which makes it difficult to parallelize the computation. This can limit the speed and scalability of the network.

Difficulty in Choosing the Right Architecture: There are many different variants of RNNs, each with its own advantages and disadvantages. Choosing the right architecture for a given task can be challenging, and may require extensive experimentation and tuning.

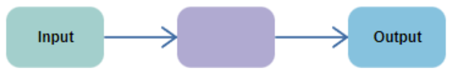
Difficulty in Interpreting the Output: The output of an RNN can be difficult to interpret, especially when dealing with complex inputs such as natural language or audio. This can make it difficult to understand how the network is making its predictions.

These disadvantages are important when deciding whether to use an RNN for a given task. However, many of these issues can be addressed through careful design and training of the network and through techniques such as regularization and attention mechanisms.

The four commonly used types of Recurrent Neural Networks are:

1. One-to-One

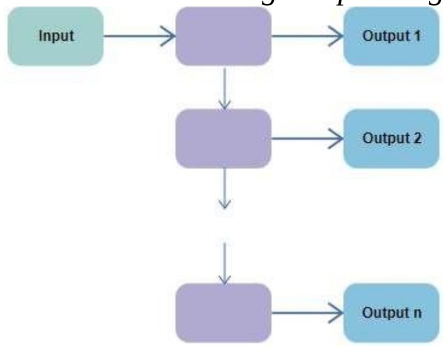
The simplest type of RNN is **One-to-One**, which allows a single input and a single output. It has fixed input and output sizes and acts as a traditional neural network. The One-to-One application can be found in *Image Classification*.



One-to One

2. One-to-Many

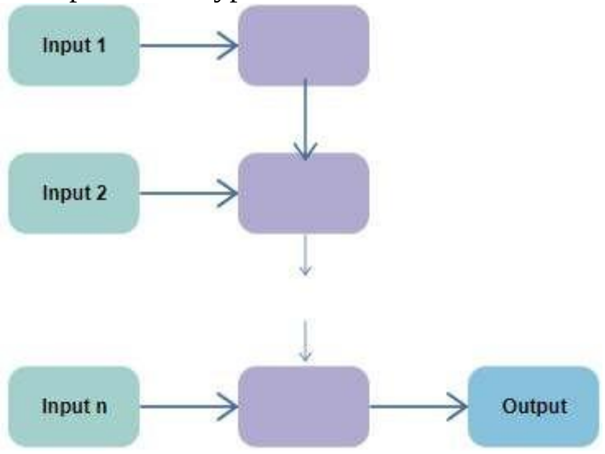
One-to-Many is a type of RNN that gives multiple outputs when given a single input. It takes a fixed input size and gives a sequence of data outputs. Its applications can be found in *Music Generation* and *Image Captioning*.



One-to-Many

3. Many-to-One

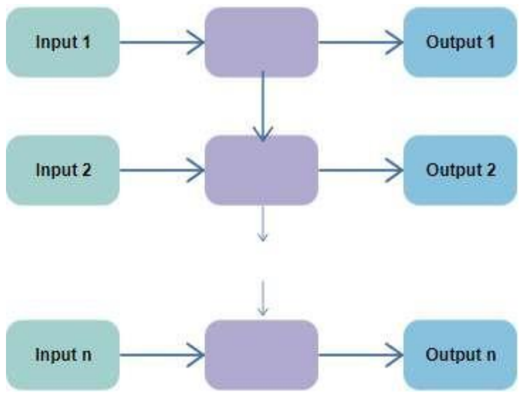
Many-to-One is used when a single output is required from multiple input units or a sequence of them. It takes a sequence of inputs to display a fixed output. *Sentiment Analysis* is a common example of this type of Recurrent Neural Network.



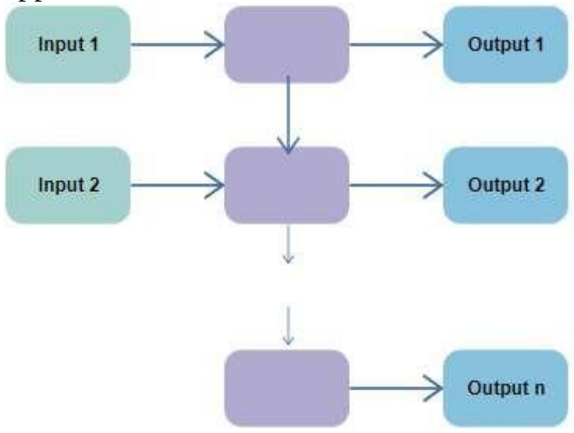
4. Many-to-Many

Many-to-Many are used to generate a sequence of output data from a sequence of input units. This type of RNN is further divided into the following two subcategories:

1. Equal Unit Size: In this case, the number of both the input and output units is the same. A common application can be found in *Name-Entity Recognition*.



2. Unequal Unit Size: In this case, inputs and outputs have different numbers of units. Its application can be found in *Machine Translation*.

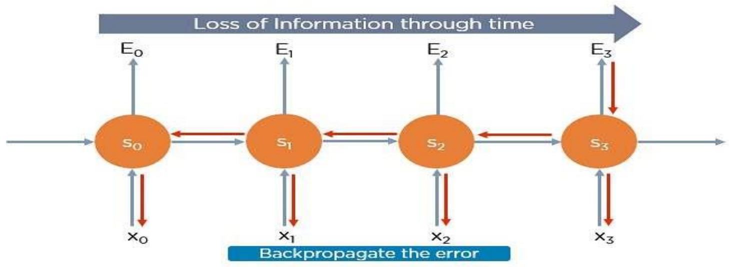


Two Issues of Standard RNNs

1**. Vanishing Gradient Problem**

Recurrent Neural Networks enable us to model time-dependent and sequential data problems, such as stock market prediction, machine translation, and text generation. We will find, however, RNN is hard to train because of the gradient problem.

RNNs suffer from the problem of vanishing gradients. The gradients carry information used in the RNN, and when the gradient becomes too small, the parameter updates become insignificant. This makes the learning of long data sequences difficult.



1. **Exploding Gradient Problem**

While training a neural network, if the slope tends to grow exponentially instead of decaying, this is called an Exploding Gradient. This problem arises when large error gradients accumulate, resulting in very large updates to the neural network model weights during the training process.Long training time, poor performance, and bad accuracy are the major issues in gradient problems.

Feed-Forward Neural Networks vs Recurrent Neural Networks

A feed-forward neural network allows information to flow only in the forward direction, from the input nodes, through the hidden layers, and to the output nodes. There are no cycles or loops in the network. Below is how a simplified presentation of a feed-forward neural network looks like:

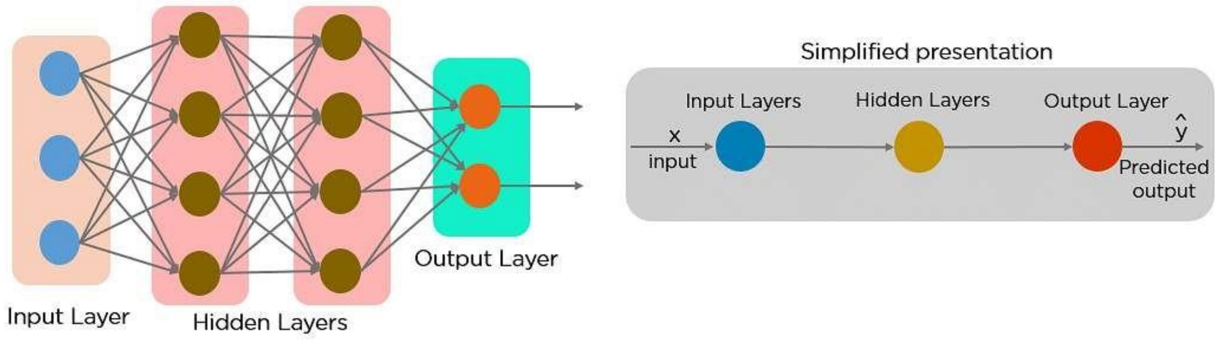


Fig: Feed-forward Neural Network

In a feed-forward neural network, the decisions are based on the current input. It doesn't memorize the past data, and there's no future scope. Feed-forward neural networks are used in general regression and classification problems.

Variant RNN Architectures

There are several variant RNN architectures that have been developed over the years to address the limitations of the standard RNN architecture. Here are a few examples:

Long Short-Term Memory (LSTM) Networks

LSTM is a type of RNN that is designed to handle the vanishing gradient problem that can occur in standard RNNs. It does this by introducing three gating mechanisms that control the flow of

information through the network: the input gate, the forget gate, and the output gate. These gates allow the LSTM network to selectively remember or forget information from the input sequence, which makes it more effective for long-term dependencies.

Gated Recurrent Unit (GRU) Networks

GRU is another type of RNN that is designed to address the vanishing gradient problem. It has two gates: the reset gate and the update gate. The reset gate determines how much of the previous state should be forgotten, while the update gate determines how much of the new state should be remembered. This allows the GRU network to selectively update its internal state based on the input sequence.

Bidirectional RNNs:

Bidirectional RNNs are designed to process input sequences in both forward and backward directions. This allows the network to capture both past and future context, which can be useful for speech recognition and natural language processing tasks.

Encoder-Decoder RNNs:

Encoder-decoder RNNs consist of two RNNs: an encoder network that processes the input sequence and produces a fixed-length vector representation of the input and a decoder network that generates the output sequence based on the encoder's representation. This architecture is commonly used for sequence-to-sequence tasks such as machine translation.

Attention Mechanisms

Attention mechanisms are a technique that can be used to improve the performance of RNNs on tasks that involve long input sequences. They work by allowing the network to attend to different parts of the input sequence selectively rather than treating all parts of the input sequence equally. This can help the network focus on the input sequence's most relevant parts and ignore irrelevant information.

These are just a few examples of the many variant RNN architectures that have been developed over the years. The choice of architecture depends on the specific task and the characteristics of the input and output sequences.

Encoder-Decoder Model

There are three main blocks in the encoder-decoder model,

- Encoder
- Hidden Vector
- Decoder

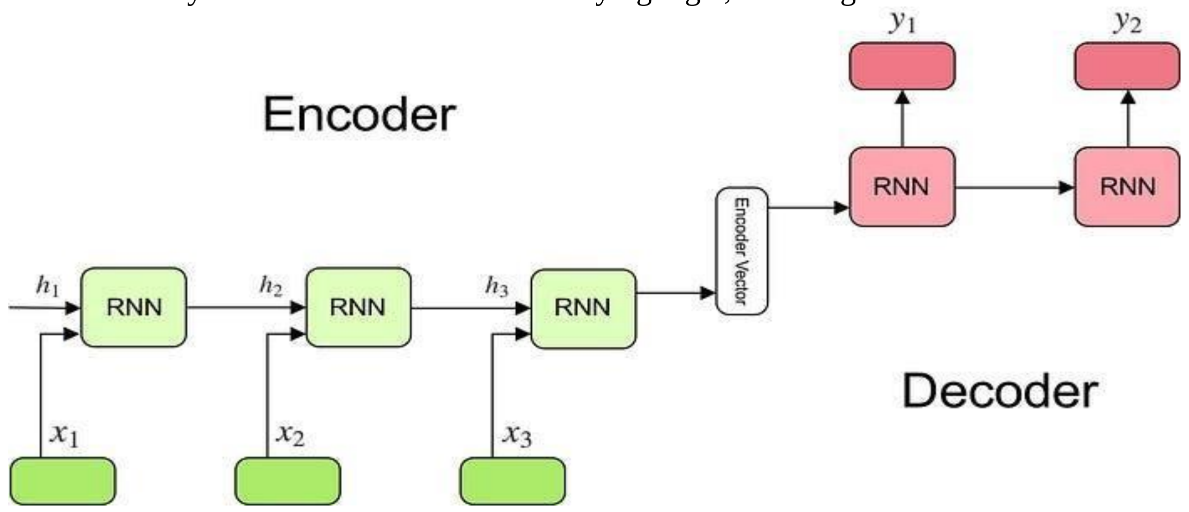
The Encoder will convert the input sequence into a single-dimensional vector (hidden vector). The decoder will convert the hidden vector into the output sequence.

Encoder-Decoder models are jointly trained to maximize the conditional probabilities of the target sequence given the input sequence.

SEQUENCE TO SEQUENCE RNN

How the Sequence to Sequence Model works?

In order to fully understand the model's underlying logic, we will go over the below illustration:

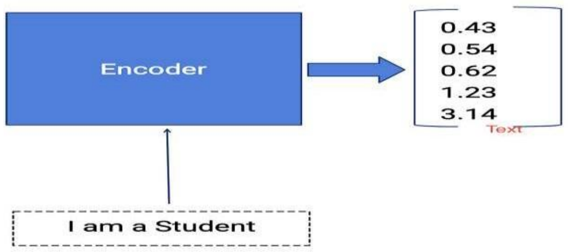


Encoder-decoder sequence to sequence model

Encoder

- Multiple RNN cells can be stacked together to form the encoder. RNN reads each inputs sequentially
- For every timestep (each input) t, the hidden state (hidden vector) h is updated according to the input at that timestep X[i].
- After all the inputs are read by encoder model, the final hidden state of the model represents the context/summary of the whole input sequence.

- Example: Consider the input sequence "I am a Student" to be encoded. There will be totally 4 timesteps (4 tokens) for the Encoder model. At each time step, the hidden state h will be updated using the previous hidden state and the current input.



Example: Encoder

- At the first timestep t1, the previous hidden state h0 will be considered as zero or randomly chosen. So the first RNN cell will update the current hidden state with the first input and h0. Each layer outputs two things — updated hidden state and the output for each stage. The outputs at each stage are rejected and only the hidden states will be propagated to the next layer.

- The hidden states *h_i* are computed using the formula:

$$h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

- At second timestep t2, the hidden state h1 and the second input X[2] will be given as input , and the hidden state h2 will be updated according to both inputs. Then the hidden state h1 will be updated with the new input and will produce the hidden state h2. This happens for all the four stages wrt example taken.

- A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element, and propagates it forward.

- In the question-answering problem, the input sequence is a collection of all words from the question. Each word is represented as *x_i* where *i* is the order of that word.

This simple formula represents the result of an ordinary recurrent neural network. As you can see, we just apply the appropriate weights to the previously hidden state h_(t-1) and the input vector x_t.

Encoder Vector

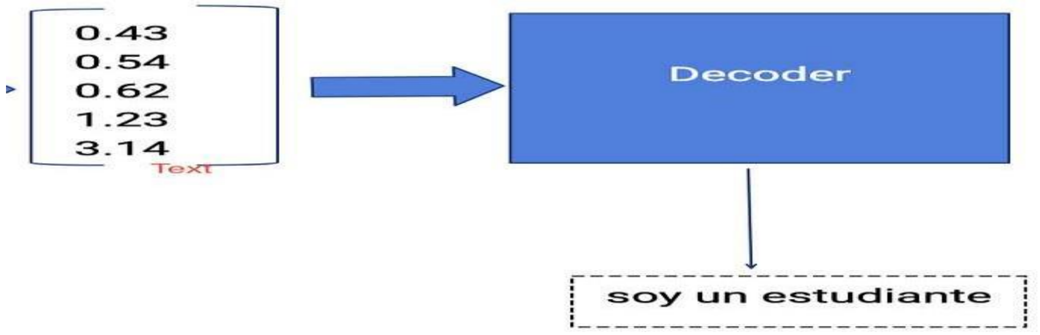
- This is the final hidden state produced from the encoder part of the model. It is calculated using the formula above.
- This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.
- It acts as the initial hidden state of the decoder part of the model.

Decoder

- The Decoder generates the output sequence by predicting the next output Yt given the hidden state ht.
- The input for the decoder is the final hidden vector obtained at the end of encoder model.
- Each layer will have three inputs, hidden vector from previous layer ht-1 and the previous layer output yt-1, original hidden vector h.
- At the first layer, the output vector of encoder and the random symbol START, empty hidden state ht-1 will be given as input, the outputs obtained will be y1 and updated hidden state h1 (the information of the output will be subtracted from the hidden vector).
- The second layer will have the updated hidden state h1 and the previous output y1 and original hidden vector h as current inputs, produces the hidden vector h2 and output y2.
- The outputs occurred at each timestep of decoder is the actual output. The model will predict the output until the END symbol occurs.
- A stack of several recurrent units where each predicts an output *y_t* at a time step *t*.

- Each recurrent unit accepts a hidden state from the previous unit and produces an output as well as its own hidden state.

- In the question-answering problem, the output sequence is a collection of all words from the answer. Each word is represented as *y_i* where *i* is the order of that word.



Example: Decoder.

- Any hidden state *h_i* is computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1})$$

As you can see, we are just using the previous hidden state to compute the next one.

Output Layer

- We use Softmax activation function at the output layer.

- It is used to produce the probability distribution from a vector of values with the target class of high probability.

- The output *y_t* at time step *t* is computed using the formula:

$$y_t = softmax(W^S h_t)$$

We calculate the outputs using the hidden state at the current time step together with the respective weight W(S). Softmax is used to create a probability vector that will help us determine the final output (e.g. word in the question-answering problem).

The power of this model lies in the fact that it can map sequences of different lengths to each other**.** As you can see the inputs and outputs are not correlated and their lengths can differ. This opens a whole new range of problems that can now be solved using such architecture.

Applications

It possesses many applications such as

- Google's Machine Translation
- Question answering chatbots
- Speech recognition
- Time Series Application etc.,

BIDIRECTIONAL RNN

A bi-directional recurrent neural network (Bi-RNN) is a type of recurrent neural network (RNN) that processes input data in both forward and backward directions. The goal of a Bi-RNN is to capture the contextual dependencies in the input data by processing it in both directions, which can be useful in a variety of natural language processing (NLP) tasks.

In a Bi-RNN, the input data is passed through two separate RNNs: one processes the data in the forward direction, while the other processes it in the reverse direction. The outputs of these two RNNs are then combined in some way to produce the final output.

One common way to combine the outputs of the forward and reverse RNNs is to concatenate them, but other methods, such as element-wise addition or multiplication can also be used. The choice of combination method can depend on the specific task and the desired properties of the final output.

Need for Bi-directional RNNs

- A uni-directional recurrent neural network (RNN) processes input sequences in a single direction, either from left to right or right to left.

- This means that the network can only use information from earlier time steps when making predictions at later time steps.

- This can be limiting, as the network may not capture important contextual information relevant to the output prediction.

- For example, in natural language processing tasks, a uni-directional RNN may not accurately predict the next word in a sentence if the previous words provide important context for the current word.

Consider an example where we could use the recurrent network to predict the masked word in a sentence.

1. Apple is my favorite______.
2. Apple is my favourite______, and I work there.
3. Apple is my favorite______, and I am going to buy one.

In the first sentence, the answer could be fruit, company, or phone. But in the second and third sentences, it cannot be a fruit.

A Recurrent Neural Network that can only process the inputs from left to right might not be able to accurately predict the right answer for sentences discussed above.

To perform well on natural language tasks, the model must be able to process the sequence in both directions.

Bi-directional RNNs

- A bidirectional recurrent neural network (RNN) is a type of recurrent neural network (RNN) that processes input sequences in both forward and backward directions.

- This allows the RNN to capture information from the input sequence that may be relevant to the output prediction, but the same could be lost in a traditional RNN that only processes the input sequence in one direction.

- This allows the network to consider information from the past and future when making predictions rather than just relying on the input data at the current time step.

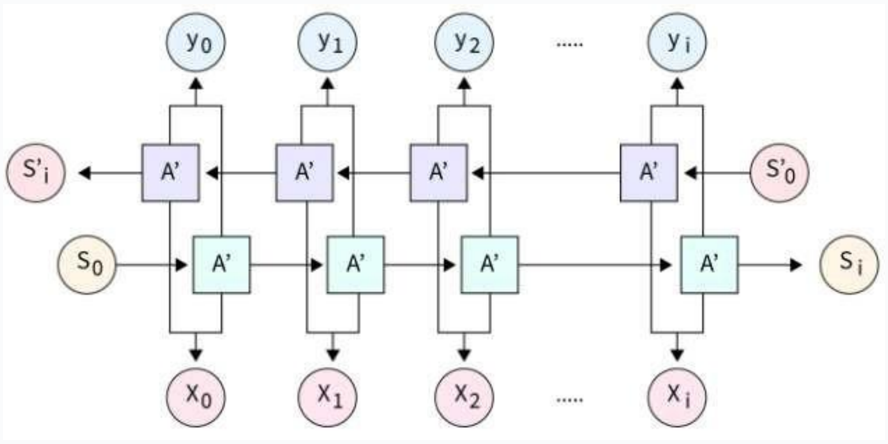
- This can be useful for tasks such as language processing, where understanding the context of a word or phrase can be important for making accurate predictions.

- In general, bidirectional RNNs can help improve the performance of a model on a variety of sequence-based tasks.

This means that the network has **two separate RNNs**:

1. One that processes the input sequence from left to right
2. Another one that processes the input sequence from right to left.

These two RNNs are typically referred to as the forward and backward RNNs, respectively.



During the forward pass of the RNN, the forward RNN processes the input sequence in the usual way by taking the input at each time step and using it to update the hidden state. The updated hidden state is then used to predict the output at that time step.

Back-propagation through time (BPTT) is a widely used algorithm for training recurrent neural networks (RNNs). It is a variant of the back-propagation algorithm specifically designed to handle the temporal nature of RNNs, where the output at each time step depends on the inputs and outputs at previous time steps.

In the case of a bidirectional RNN, BPTT involves two separate Back-propagation passes: one for the forward RNN and one for the backward RNN. During the forward pass, the forward RNN processes the input sequence in the usual way and makes predictions for the output sequence. These predictions are then compared to the target output sequence, and the error is back-propagated through the network to update the weights of the forward RNN.

During the backward pass, the backward RNN processes the input sequence in reverse order and makes predictions for the output sequence. These predictions are then compared to the target output sequence in reverse order, and the error is back-propagated through the network to update the weights of the backward RNN.

Once both passes are complete, the weights of the forward and backward RNNs are updated based on the errors computed during the forward and backward passes, respectively. This process is repeated for multiple iterations until the model converges and the predictions of the bidirectional RNN are accurate.

This allows the bidirectional RNN to consider information from past and future time steps when making predictions, which can significantly improve the model's accuracy.

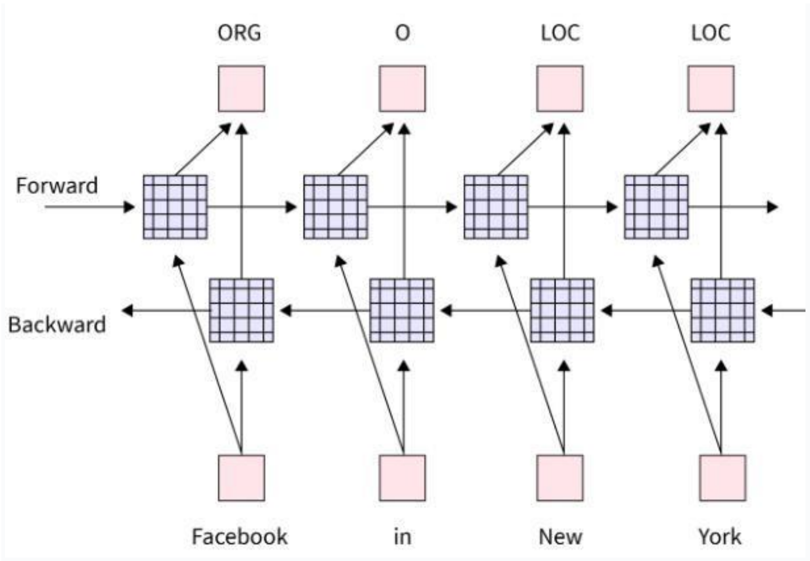
Applications of Bi-directional RNNs

Bidirectional recurrent neural networks (RNNs) can outperform traditional RNNs on various tasks, particularly those involving sequential data processing. Some examples of tasks where bidirectional RNNs have been shown to outperform traditional RNNs include:

- Natural languages processing tasks, such as language translation and sentiment analysis, where understanding the context of a word or phrase can be important for making accurate predictions.

- Time series forecasting tasks, such as predicting stock prices or weather patterns, where the sequence of past data can provide important clues about future trends.

- Audio processing tasks, such as speech recognition or music generation, where the information in the audio signal can be complex and non-linear.



In general, bidirectional RNNs can be useful for any task where the input data has a temporal structure and where understanding the context of the data is important for making accurate predictions.

Advantages and Disadvantages of Bi-directional RNNs

Advantages:

Bidirectional Recurrent Neural Networks (RNNs) have several advantages over traditional RNNs. Some of the key advantages of bidirectional RNNs include the following:

- Improved performance on tasks that involve processing sequential data. Because bidirectional RNNs can consider information from both past and future time steps when making predictions, they can outperform traditional RNNs on tasks such as natural language processing, time series forecasting, and audio processing.

Disadvantages:

However, Bidirectional RNNs also have some disadvantages. Some of the key disadvantages of bidirectional RNNs include the following:

- Increased computational complexity. Because bidirectional RNNs have two separate RNNs (one for the forward pass and one for the backward pass), they can require more computational resources to train and evaluate than traditional RNNs. This can make them more difficult to implement and less efficient in terms of runtime performance.

- More difficult to optimize. Because bidirectional RNNs have more parameters (due to the two separate RNNs), they can be more difficult to optimize. This can make finding the right set of weights for the model challenging and lead to slower convergence during training.

- The need for longer input sequences. For a bidirectional RNN to capture long-term dependencies in the data, it typically requires longer input sequences than a traditional RNN. This can be a disadvantage in situations where the input data is limited or noisy, as it may not be possible to generate enough input data to train the model effectively.

RECURSIVE NEURAL NETWORKS

Recursive Neural Networks (RvNNs) are a class of deep neural networks that can learn detailed and structured information. With RvNN, you can get a structured prediction by recursively applying the same set of weights on structured inputs. The word recursive indicates that the neural network is applied to its output.

Due to their deep tree-like structure, Recursive Neural Networks can handle hierarchical data. The tree structure means combining child nodes and producing parent nodes. Each child-parent bond has a weight matrix, and similar children have the same weights. The number of children for every node in the tree is fixed to enable it to perform recursive operations and use the same weights. RvNNs are used when there's a need to parse an entire sentence.

To calculate the parent node's representation, we add the products of the weight matrices (W_i) and the children's representations (C_i) and apply the transformation f:

\[h = f \left(\sum_{i=1}^{i=c} W_i C_i \right) \], where c is the number of children.

Recurrent Neural Network vs. Recursive Neural Networks

- Recurrent Neural Networks (RNNs) are another well-known class of neural networks used for processing sequential data. They are closely related to the Recursive Neural Network.
- Recurrent Neural Networks represent temporal sequences, which they find application in Natural language Processing (NLP) since language-related data like sentences and

paragraphs are sequential in nature. Recurrent networks are usually chain structures. The weights are shared across the chain length, keeping the dimensionality constant.

- On the other hand, Recursive Neural Networks operate on hierarchical data models due to their tree structure. There are a fixed number of children for each node in the tree so that it can execute recursive operations and use the same weights for each step. Child representations are combined into parent representations.
- The efficiency of a recursive network is higher than a feed-forward network.
- Recurrent Networks are recurrent over time, meaning recursive networks are just a generalization of the recurrent network.

Recursive Neural Network Implementation

A Recursive Neural Network is used for sentiment analysis in natural language sentences. It is one of the most important tasks of Natural language Processing (NLP), which identifies the writing tone and sentiments of the writer in a particular sentence. If a writer expresses any sentiment, basic labels about the writing tone are recognized. We want to identify the smaller components like nouns or verb phrases and order them in a syntactic hierarchy. For example, it identifies whether the sentence showcases a constructive form of writing or negative word choices.

A variable called 'score' is calculated at each traversal of nodes, telling us which pair of phrases and words we must combine to form the perfect syntactic tree for a given sentence.

Let us consider the representation of the phrase -- "a lot of fun" in the following sentence.

Programming is a lot of fun.

An RNN representation of this phrase would not be suitable because it considers only sequential relations. Each state varies with the preceding words' representation. So, a subsequence that doesn't occur at the beginning of the sentence can't be represented. With RNN, when processing the word 'fun,' the hidden state will represent the whole sentence.

However, with a Recursive Neural Network (RvNN), the hierarchical architecture can store the representation of the exact phrase. It lies in the hidden state of the node R_{a\ lot\ of\ fun}. Thus, Syntactic parsing is completely implemented with the help of Recursive Neural Networks.

Benefits of RvNNs for Natural Language Processing

- The two significant advantages of Recursive Neural Networks for Natural Language Processing are their structure and reduction in network depth.
- As already explained, the tree structure of Recursive Neural Networks can manage hierarchical data like in parsing problems.

- Another benefit of RvNN is that the trees can have a logarithmic height. When there are O(n) input words, a Recursive Neural Network can represent a binary tree with height O(log\ n). This lessens the distance between the first and last input elements. Hence, the long-term dependency turns shorter and easier to grab.

Disadvantages of RvNNs for Natural Language Processing

- The main disadvantage of recursive neural networks can be the tree structure. Using the tree structure indicates introducing a unique inductive bias to our model. The bias corresponds to the assumption that the data follow a tree hierarchy structure. But that is not the truth. Thus, the network may not be able to learn the existing patterns.
- Another disadvantage of the Recursive Neural Network is that sentence parsing can be slow and ambiguous. Interestingly, there can be many parse trees for a single sentence.
- Also, it is more time-consuming and labor-intensive to label the training data for recursive neural networks than to construct recurrent neural networks. Manually parsing a sentence into short components is more time-consuming and tedious than assigning a label to a sentence.

Gated Architecture

LONG SHORT TERM MEMORY NETWORK (LSTM).

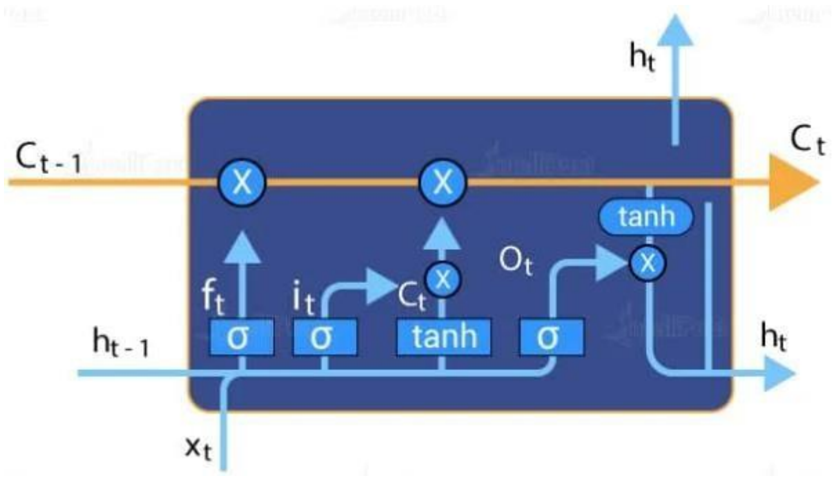
LSTM used in the field of Deep Learning. It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems.

LSTMs are predominantly used to learn, process, and classify sequential data because these networks can learn long-term dependencies between time steps of data. Common LSTM applications include sentiment analysis, language modelling, speech recognition, and video analysis.

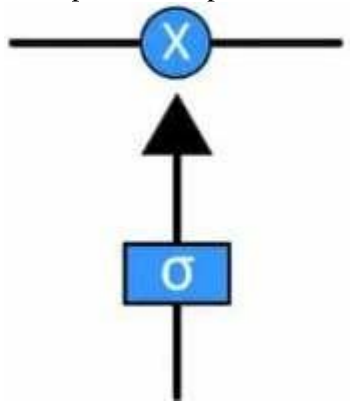
LSTM has feedback connections, i.e., it is capable of processing the entire sequence of data, apart from single data points such as images. This finds application in speech recognition, machine translation, etc. LSTM is a special kind of RNN, which shows outstanding performance on a large variety of problems.

The Logic behind LSTM

The central role of an LSTM model is held by a memory cell known as a ‘cell state’ that maintains its state over time. The cell state is the horizontal line that runs through the top of the below diagram. It can be visualized as a conveyor belt through which information just flows, unchanged.



Information can be added to or removed from the cell state in LSTM and is regulated by gates. These gates optionally let the information flow in and out of the cell. It contains a point wise multiplication operation and a sigmoid neural net layer that assist the mechanism.



The sigmoid layer gives out numbers between zero and one, where zero means 'nothing should be let through,' and one means 'everything should be let through.'

1. **Forget Gate(f):** At forget gate the input is combined with the previous output to generate a fraction between 0 and 1, that determines how much of the previous state need to be preserved (or in other words, how much of the state should be forgotten). This output is then multiplied with the previous state. Note: An activation output of 1.0 means "remember everything" and activation output of 0.0 means "forget everything." From a different perspective, a better name for the forget gate might be the "remember gate"
2. **Input Gate(i):** Input gate operates on the same signals as the forget gate, but here the objective is to decide which new information is going to enter the state of LSTM. The output of the input gate (again a fraction between 0 and 1) is multiplied with the output of tan h block that produces the new values that must be added to previous state. This gated vector is then added to previous state to generate current state
3. **Input Modulation Gate(g):** It is often considered as a sub-part of the input gate and much literature on LSTM's does not even mention it and assume it is inside the Input gate. It is used to modulate the information that the Input gate will write onto the Internal State Cell by adding non-linearity to the information and making the information **Zero-mean**. This is done to reduce the learning time as Zero-mean input has faster convergence. Although this gate's actions are less important than the others and are often treated as a finesse-

providing concept, it is good practice to include this gate in the structure of the LSTM unit.

4. **Output Gate(o):** At output gate, the input and previous state are gated as before to generate another scaling fraction that is combined with the output of tanh block that brings the current state. This output is then given out. The output and state are fed back into the LSTM block.

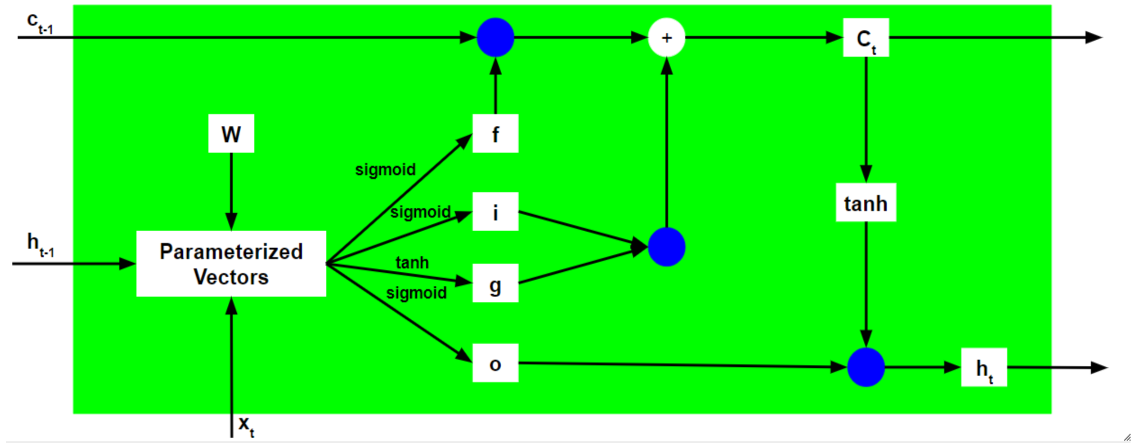
The basic workflow of a Long Short Term Memory Network is similar to the workflow of a Recurrent Neural Network with the only difference being that the Internal Cell State is also passed forward along with the Hidden State.

Working of an LSTM recurrent unit:

1. Take input the current input, the previous hidden state, and the previous internal cell state.
2. Calculate the values of the four different gates by following the below steps:-
   - For each gate, calculate the parameterized vectors for the current input and the previous hidden state by element-wise multiplication with the concerned vector with the respective weights for each gate.
   - Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.
3. Calculate the current internal cell state by first calculating the element-wise multiplication vector of the input gate and the input modulation gate, then calculate the element-wise multiplication vector of the forget gate and the previous internal cell state and then add the two vectors.

4. Calculate the current hidden state by first taking the element-wise hyperbolic tangent of the current internal cell state vector and then performing element-wise multiplication with the output gate.

The above-stated working is illustrated as below:-



Note that the blue circles denote element-wise multiplication. The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.

LSTMs work in a 3-step process.

Step 1: Decide How Much Past Data It Should Remember

The first step in the LSTM is to decide which information should be omitted from the cell in that particular time step. The sigmoid function determines this. It looks at the previous state (ht-1) along with the current input xt and computes the function.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

f_t – forget gate. Decides which information to delete that is not important from previous time step.

Consider the following two sentences:

1. Let the output of h(t-1) be "Alice is good in Physics. John, on the other hand, is good at Chemistry."

2. Let the current input at x(t) be "John plays football well. He told me yesterday over the phone that he had served as the captain of his college football team."

The forget gate realizes there might be a change in context after encountering the first full stop. It compares with the current input sentence at x(t). The next sentence talks about John, so the information on Alice is deleted. The position of the subject is vacated and assigned to John.

Step 2: Decide How Much This Unit Adds to the Current State

In the second layer, there are two parts. One is the sigmoid function, and the other is the tanh function. In the sigmoid function, it decides which values to let through (0 or1). tanh function gives weightage to the values which are passed, deciding their level of importance (-1 to 1).

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

i_t - input gate.Determines which information to let through based on its significance in the current time step.

With the current input at x(t), the input gate analyses the important information John plays football, and the fact that he was the captain of his college team is important.

"He told me yesterday over the phone" is less important; hence it's forgotten. This process of adding some new information can be done via the input gate.

Step 3: Decide What Part of the Current Cell State Makes It to the Output

The third step is to decide what the output will be. First, we run a sigmoid layer, which decides what parts of the cell state make it to the output. Then, we put the cell state through tanh to push the values to be between -1 and 1 and multiply it by the output of the sigmoid gate.

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

$$h_t = o_t * \tanh\left(C_t\right)$$

O_t-output gate.Allows the passed in information to impact the output in the current time step

Let’s consider this example to predict the next word in the sentence: “John played tremendously well against the opponent and won for his team. For his contributions, brave___was awarded player of the match.”There could be many choices for the empty space. The current input brave is an adjective, and adjectives describe a noun. So, “John” could be the best output after brave.

LSTM Applications

LSTM networks find useful applications in the following areas:

- Language modelling
- Machine translation
- Handwriting recognition
- Image captioning
- Image generation using attention models
- Question answering
- Video-to-text conversion
- Polymorphic music modelling
- Speech synthesis
- Protein secondary structure prediction

connections

Skip connections are a type of shortcut that connects the output of one layer to the input of another layer that is not adjacent to it. For example, in a CNN with four layers, A, B, C, and D, a skip connection could connect layer A to layer C, or layer B to layer D, or both.

Skip connection is a standard module in much convolutional architecture. By using a skip connection, we provide an alternative path for the gradient (with back-propagation). It is experimentally validated that this additional paths are often beneficial for the model convergence. **Skip connections** in deep architectures, as the name suggests, **skip some layer in the neural network and feeds the output of one layer as the input to the next layers** (instead of only the next one).

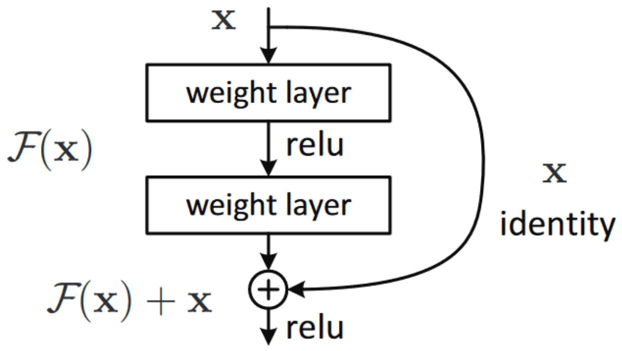
As previously explained, using the chain rule, we must keep multiplying terms with the error gradient as we go backwards. However, in the long chain of multiplication, if we multiply many things together that are less than one, then the resulting gradient will be very small. Thus, **the gradient becomes very small as we approach the earlier layers in a deep architecture**. In some cases, the gradient becomes zero, meaning that **we do not update the early layers at all**.

In general, there are two fundamental ways that one could use skip connections through different non-sequential layers:

a) **Addition** as in residual architectures,
b) **Concatenation** as in densely connected architectures.
We will first describe addition which is commonly referred as residual skip connections.

Skip connections via addition

The core idea is to back-propagate through the identity function, by just using a vector addition. Then the gradient would simply be multiplied by one and its value will be maintained in the earlier layers. This is the main idea behind Residual Networks (ResNets): they stack these skip residual blocks together. We use an identity function to **preserve the gradient**.



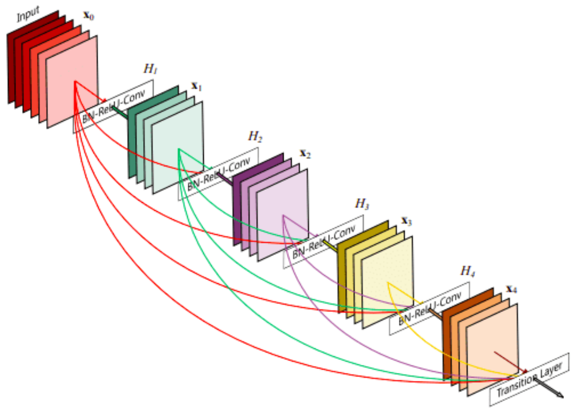
Mathematically, we can represent the residual block, and calculate its partial derivative (gradient), given the loss function like this:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial H}\frac{\partial H}{\partial x} = \frac{\partial L}{\partial H}(\frac{\partial F}{\partial x} + 1) = \frac{\partial L}{\partial H}\frac{\partial F}{\partial x} + \frac{\partial L}{\partial H}$$

Apart from the vanishing gradients, there is another reason that we commonly use them. For a plethora of tasks (such as semantic segmentation, optical flow estimation, etc.) there is some information that was captured in the initial layers and we would like to allow the later layers to also learn from them. **It has been observed that in earlier layers the learned features correspond to lower semantic information that is extracted from the input.** If we had not used the skip connection that information would have turned too abstract.

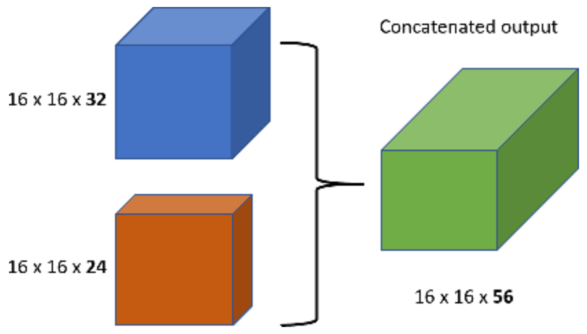
Skip connections via concatenation

As stated, for many dense prediction problems, **there is low-level information shared between the input and output, and it would be desirable to pass this information directly across the net**. The alternative way that we can achieve skip connections is by concatenation of previous feature maps. The most famous deep learning architecture is DenseNet. Below we can see an example of feature reusability by concatenation with 5 convolutional layers:



This architecture heavily uses feature concatenation so as to ensure maximum information flow between layers in the network. This is achieved by **connecting via concatenation all layers directly with each other**, as opposed to ResNets. Practically, what we basically do is to concatenate the feature channel dimension. This leads to

a) An **enormous amount of feature channels on the last layers** of the network,
b) To more **compact** models, and
c) Extreme **feature reusability**.



Short and long skip connections in Deep Learning

In more practical terms, we have to be careful when introducing additive skip connections in our deep learning model. **The dimensionality has to be the same in addition and also in concatenation apart from the chosen channel dimension**. That is the reason why we see that additive skip connections are used in two kinds of setups:

a) **Short** skip connections
b) **Long** skip connections.

Short skip connections are used along with consecutive convolutional layers that do not change the input dimension (see Res-Net), while long skip connections usually exist in encoder-decoder architectures. It is known that the **global information** (shape of the image and other statistics) **resolves what**, **while local information resolves where** (small details in an image patch).

Long skip connections often exist in architectures that are symmetrical, where the **spatial dimensionality is reduced in the encoder part** and is gradually increased in the decoder part as illustrated below. In the decoder part, one can increase the dimensionality of a feature map via transpose convolutional layers. The transposed convolution operation forms the same connectivity as the normal convolution but in the backward direction.

Benefits of skip connections

Skip connections can provide several benefits for CNNs, such as improving accuracy and generalization, solving the vanishing gradient problem, and enabling deeper networks. Skip connections can help the network to learn more complex and diverse patterns from the data and reduce the number of parameters and operations needed by the network. Additionally, skip connections can help to alleviate the problem of vanishing gradients by providing alternative paths for the gradients to flow. Furthermore, they can make it easier and faster to train deeper networks, which have more expressive power and can capture more features from the data.

Drawbacks of skip connections

Skip connections are a popular and powerful technique for improving the performance and efficiency of CNNs, but they are not a panacea. They can help preserve information and gradients, combine features, solve the vanishing gradient problem, and enable deeper networks. However, they can also increase complexity and memory requirements, introduce redundancy and noise, and require careful design and tuning to match the network architecture and data domain. Different types and locations of skip connections can have different impacts on the network performance, with some being more beneficial or harmful than others. Thus, it is essential to understand how skip connections work and how to use them wisely and effectively for CNNs.

Dropouts

Dropout refers to data, or noise, that's intentionally dropped from a neural network to improve processing and time to results. A neural network is software attempting to emulate the actions of the human brain.

Neural networks are the building blocks of any machine-learning architecture. They consist of one input layer, one or more hidden layers, and an output layer.

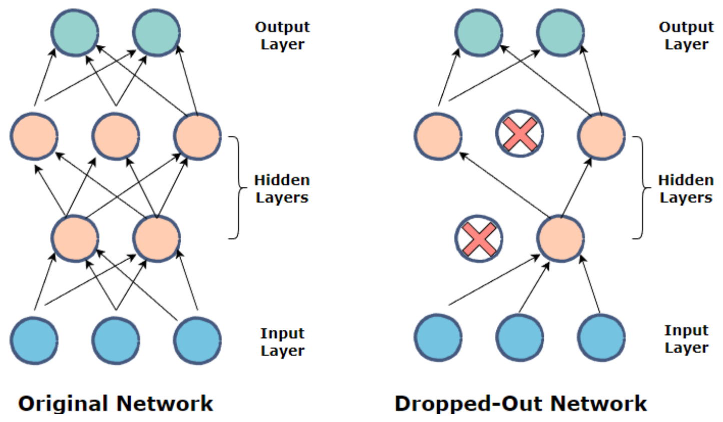
When we training our neural network (or model) by updating each of its weights, it might become too dependent on the dataset we are using. Therefore, when this model has to make a prediction or classification, it will not give satisfactory results. This is known as **over-fitting**. We might understand this problem through a real-world example: If a student of mathematics learns **only** one chapter of a book and then takes a test on the **whole** syllabus, he will probably fail.

To overcome this problem, we use a technique that was introduced by Geoffrey Hinton in 2012. This technique is known as **dropout**.

The basic idea of this method is to, based on probability, temporarily *“drop out”* neurons from our original network. Doing this for every training example gives us different models for each one. Afterwards, when we want to test our model, we take the average of each model to get our answer/prediction.

Dropout during training

We assign ‘p’ to represent the probability of a neuron, in the hidden layer, being excluded from the network; this probability value is usually equal to 0.5. We do the same process for the input layer whose probability value is usually lower than 0.5 (e.g. 0.2). Remember, we delete the connections going into, and out of, the neuron when we drop it.



Dropout during testing

An output, given from a model trained using the dropout technique, is a bit different: We can take a sample of many dropped-out models and compute the geometric mean of their output neurons by multiplying all the numbers together and taking the product’s square root. However, since this is computationally expensive, we use the original model instead by simply cutting all of the hidden units’ weights in half. This will give us a good approximation of the average for each of the different dropped-out models.

Shree SATHYAM COLLEGE OF ENGG & TECH

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

AD3501 / DEEPLEARNING - NOTES

UNIT - IV MODEL EVALUATION

Performance metrics—Base line Models--Hyperparameters: Manual Hyperparameter -- Automatic Hyperparameter -- Grid search -- Random search -- Debugging strategies.

PERFORMANCE METRICS

To evaluate the performance or quality of the model, different metrics are used, and these metrics are known as performance metrics or evaluation metrics. *These performance metrics help us understand* how well our model has performed for the given data. In this way, we can improve the model's performance by tuning the hyper-parameters. Each ML model aims to generalize well on unseen/new data, and performance metrics help determine how well the model generalizes on the new dataset.

In machine learning, each task or problem is divided into **classification** and **Regression**. Not all metrics can be used for all types of problems;

Performance Metrics for Classification

In a classification problem, the category or classes of data is identified based on training data. The model learns from the given dataset and then classifies the new data into classes or groups based on the training. It predicts class labels as the output, such as *Yes or No, 0 or 1, Spam or Not Spam,* etc. To evaluate the performance of a classification model, different metrics are used, and some of them are as follows:

- Accuracy
- Confusion Matrix
- Precision
- Recall
- F-Score
- AUC(Area Under the Curve)-ROC

I. Accuracy

The accuracy metric is one of the simplest Classification metrics to implement, and it can be determined as the number of correct predictions to the total number of predictions

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ predictions}$$

When to Use Accuracy?

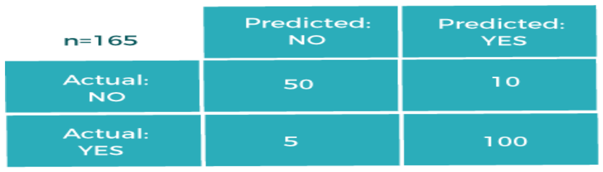
It is good to use the Accuracy metric when the target variable classes in data are approximately balanced. For example, if 60% of classes in a fruit image dataset are of Apple, 40% are Mango. In this case, if the model is asked to predict whether the image is of Apple or Mango, it will give a prediction with 97% of accuracy.

When not to use Accuracy?

It is recommended not to use the Accuracy measure when the target variable majorly belongs to one class. For example, Suppose there is a model for a disease prediction in which, out of 100 people, only five people havea disease, and95 people don't have one. Inthis case, if our model predicts everyperson with no disease (which means a bad prediction), the Accuracy measure will be 95%, which is not correct.

II. Confusion Matrix

A confusion matrix is a tabular representation of prediction outcomes of any binary classifier, which is used to describe the performance of the classification model on a set of test data when true values are known.



n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

- In the matrix, columns are for the prediction values, and rows specify the Actual values. Here Actual and prediction give two possible classes, Yes or No. So, if we are predicting the presence of a disease in a patient, the Prediction column with Yes means, Patient has the disease, and for NO, the Patient doesn't have the disease.
- In this example, the total number of predictions are 165, out of which 110 time predicted yes, whereas 55 times predicted No.However, in reality, 60 cases in which patients don't have the disease, whereas 105 cases in which patients have the disease.

III. Precision

The precision metric is used to overcome the limitation of Accuracy. The precision determines the proportion of positive prediction that was actually correct. It can be calculated as the True Positive or predictions that are actually true to the total positive predictions (True Positive and False Positive).

$$Precision = \frac{TP}{(TP + FP)}$$

IV. Recall or Sensitivity

It is alsosimilar tothePrecision metric; however, it aims tocalculatetheproportion ofactualpositivethat was identified incorrectly. It can be calculated as TruePositive or predictions that are actually true to the total number of positives, either correctly predicted as positive or incorrectly predicted as negative (true Positive and false negative).

$$\text{Recall}=\frac{TP}{TP+FN}$$

When to use Precision and Recall?

From the above definitions of Precision and Recall, we can say that recall determines the performance ofa classifier with respect to a false negative, whereas precision gives information about the performance of a classifier with respect to a false positive.

So, if we want to minimize the false negative, then, Recall should be as near to 100%, and if we want to minimize the false positive, then precision should be close to 100% as possible.

V. F-Scores

F-score or F1 Score is a metric to evaluate a binary classification model on the basis of predictions that are made for the positive class. It is calculated with the help of Precision and Recall. It is a type of single score that represents both Precision and Recall. So, ***the F1 Score can be calculated as the harmonic mean of both precision and Recall, assigning equal weight to each of them.***

VI. AUC-ROC

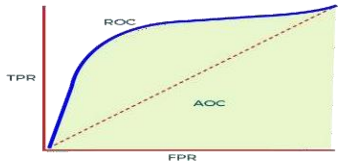
Sometimes we need to visualize the performance of the classification model on charts; then, we can use the AUC-ROC curve. It is one of the popular and important metrics for evaluating the performance of the classification model.

Firstly, let's understand ROC (Receiver Operating Characteristic curve) curve. ***ROC represents a graphto show the performance of a classification model at different threshold levels***. The curve is plotted between two parameters, which are:

- **True Positive Rate**
- **False Positive Rate**

$$TPR = \frac{TP}{TP + FN}$$

$$TPR = \frac{FP}{FP + TN}$$



AUC calculates the performance across all the thresholds and provides an aggregate measure. The value of AUC ranges from 0 to 1. It means a model with 100% wrong prediction will have an AUC of 0.0, whereas models with 100% correct predictions will have an AUC of 1.0.

Performance Metrics for Regression

Regression is a supervised learning technique that aims to find the relationships between the dependent and independent variables. A predictive regression model predicts a numeric or discrete value. The metrics used for regression are different from the classification metrics.

- Mean Absolute Error
- Mean Squared Error
- R2Score
- AdjustedR2

I. Mean Absolute Error(MAE)

MeanAbsoluteErrororMAEisoneofthesimplestmetrics,whichmeasurestheabsolutedifference between actual and predicted values, where absolute means taking a number as Positive.

The below formula is used to calculate MAE:

$$MAE = 1/N \sum |Y - Y'|$$

Here,

Y is the Actual outcome, Y'isthepredictedoutcome,andNisthetotalnumberofdatapoints.

II. Mean Squared Error

MeanSquarederror or MSEisoneofthemost suitable metricsfor Regressionevaluation. It measuresthe average of the Squared difference between predicted values and the actual value given by the model.

Moreover, due to squared differences, it penalizes small errors also, and hence it leads to over-estimation of how bad the model is.

$$MSE = 1/N \sum (Y - Y')^2$$

III. RSquared Score

R squared error is also known as Coefficient of Determination, The R-squared metric enables us to compare our model with a constant baseline to determine the performance of the model. To select the constant baseline, we need to take the mean of the data and draw the line at the mean.

The R squared score will always be less than or equal to 1 without concerning if the values are too largeor small.

$$R^2 = 1 - \frac{MSE(Model)}{MSE(Baseline)}$$

IV. Adjusted RSquared

Adjusted R squared, as the name suggests, is the improved version of R squared error. R square has a limitation of improvement of a score on increasing the terms, even though the model is not improving,and it may mislead the data scientists.

To overcome the issue of R square, adjusted R squared is used, which will always show a lower value thanR^2. Itis becauseit adjusts thevalues of increasing predictors andonlyshows improvement if thereis a real improvement.

$$R_a^2 = 1-\left[\left(\frac{n-1}{n-k-1}\right) \times \left(1 - R^2\right)\right]$$

R_a^2denotes the adjusted R^2

BASELINEMODELS

``What is a Baseline Model?

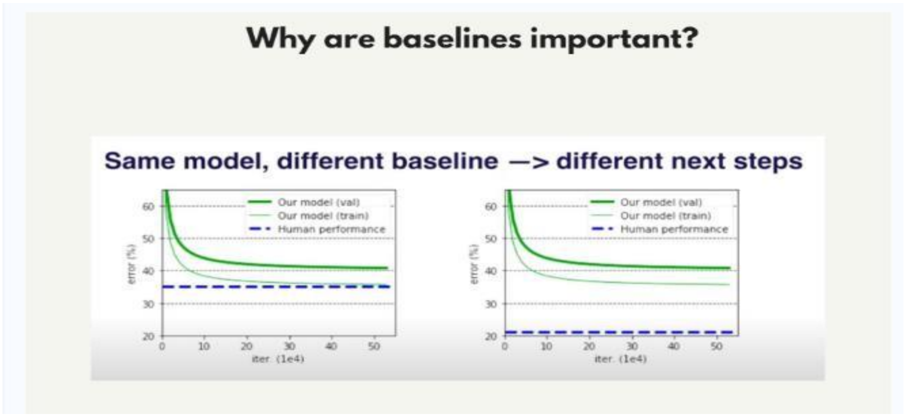
Baseline models**serve as a benchmark in an ML application. Their main goal is to put the results of trained models into context.**

Assume you begin working on a problem statement and complete all of the steps, including EDA, data cleansing, and feature engineering. You now begin working on your model. During model training, you discoverthatyourmodel'saccuracyis54%.So,withoutmakingmucheffort,younowhavea54%accuracy level, which is now your base value.You can now tag this as a baseline model, indicating that you will enhance this number after this. If your model's accuracy level goes below 54% in the future, it means the model requires improvements.

Types of baseline models

Base line models are divided into three main categories:

- **Random Baseline Models:**Data in the actual world isn't always reliable. A dummy classifier or regressoristheoptimalbaselinemodelfortheseissues.Thisbaselinemodelwillinformyouifyour machine learning model is learning or not.
- **ML Baseline Modes:**Now, ifthedataispredictable, youcancreateabaselinemodelwhichhelps us analyze which features are critical for prediction and which are not. The baseline models are commonly used with feature engineering.
- **Automated ML Baseline Models:**It is the ultimate baseline model. It's an excellent model for comparing your ML model. If your ML model outperforms the automated baseline model, it's a strong indication that themodel has thepotentialtobecomea product.



Faster iteration

Baselinemodelsalsohelpimprovetheefficiencywithwhichyoucanbuildthemodels.

- **Increase speed and performance:**With a baseline model in place, you will have detailed informationonwhattoimproveanddevelop.Thismakesiteasytoseeifthechangesyou'remaking toyourmodelareimprovingmetricsornot.Thisenablesyoutoquicklydiscoverinitiativesthatcan enhance your KPIs.
- **Efficiency:** If you builda baseline model, theamount of work you haveto do on current projects mayreduce, allowingyou tofocusonother projects.Thebaseline modelfacilitates efficiencyand productivity.

Performance benchmark

Baselinemodelsprovideasuitablestandardagainstwhichyoucanevaluateyourrealmodels.

- Someperformancemeasures,suchaslogarithmicloss,arehelpfultoevaluateamongstmodelsthan toassessindividually.Thisisduetothefactthatmanyperformancemeasurementslackaspecified scaleandinsteadtakeonvaryingvaluesbasedontheresultvariable'srange.Thiscanassistyou in determiningwhenasophisticatedmodelisrequiredvswhensimplebusiness logicisadequate.
- Calculate the impact on key business parameters. Creating a simple baseline model can also help you see what type of influence you might have on company indicators. This is particularly true if your baseline model is stochastic as well.

HYPERPARAMETERS

In neural networks, parameters are used to train the model and make predictions. There are two types of parameters:

Model parameters are internal to the neural network – for example, neuron weights. They are estimated or learned automatically from training samples. These parameters are also used to make predictions in a production model.
Hyperparameters are external parameters set by the operator of the neural network – for example, selecting whichactivation functionto useor thebatchsizeused in training. Hyperparameters havea huge impact on the accuracy of a neural network, there may be different optimal values for different values,and it is non-trivial to discover those values.
The simplest way to select hyperparameters for a neural network model is "manual search" – in other words, trial and error. New methods are evolving which use algorithms and optimization methods to discover the best hyperparameters.

What is Hyperparameter Tuning?

A hyperparameter is a parameter of the model whose value influences the learning process and whose value cannot be estimated from the training data. Hyperparameters are configured externally before startingthemodellearning/trainingprocess.

List of Common Hyperparameters

1. Number of hidden layers – adding more hidden layers of neurons generally improves accuracy, to a certain limit which can differ depending on the problem.
2. Dropout–whatpercentageofneuronsshouldberandomly"killed"duringeachepochtoprevent overfitting.
3. Activation function– which function should be used to process the inputs flowing into each neuron. The activation function can impact the network's ability to converge and learn for different ranges ofinput values, and also its training speed.
4. Weights initialization– it is necessary to set initial weights for the first forward pass. Two basic options are to set weights to zero or to randomize them. However, this can result in a vanishing or exploding gradient, which will make it difficult to train the model. To mitigate this problem, you can usea heuristic (a formula tied to the number of neuron layers) to determine the weights. A common heuristic used for the Tanh activation is called Xavier initialization.

Hyperparameters related to training algorithm

1. Learning rate– how fast the back propagation algorithm performs gradient descent. A lower learning rate makes the network train faster but might result in missing the minimum of the loss function.
2. Epoch, iterations and batch size – theseparameters determinetherateat whichsamples arefedtothe modelfor training. An epochis a group of samples whicharepassedthroughthe modeltogether (forward pass)andthenrunthroughbackpropagation(backwardpass)todeterminetheiroptimalweights.Ifthe

epoch cannot be run all together due the size of the sample or complexity of the network, it is split into batches, and the epoch is run in two or more iterations. The number of epochs and batches per epoch can significantly affect model fit, as shown below.

3. Optimizer algorithm– when a neural network trains, it uses an algorithm to determine the optimal weights for the model, called an optimizer. Thebasic option is Stochastic Gradient Descent, but thereare other options.

4. Momentum— Another common algorithm is Momentum, which works by waiting after a weight is updated, and updating it a second time using a delta amount. This speeds up training gradually, with a reduced risk of oscillation. Other algorithms are Nesterov Accelerated Gradient, AdaDelta and Adam.

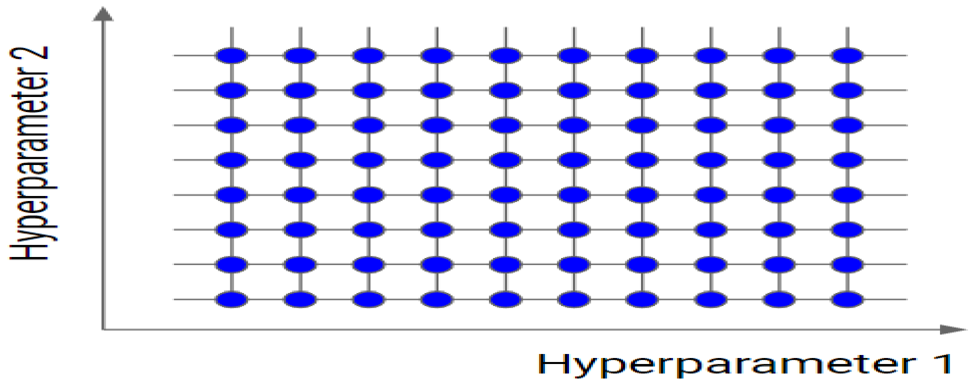
Manual Hyperparameter Tuning

Traditionally, hyperparameters were tuned manually by trial and error. This is still commonly done, and experienced operators can "guess" parameter values that will achieve very high accuracy for deeplearning models. However, there is a constant search for better, faster and more automatic methods to optimize hyperparameters. **Pros:**Very simple and effective with skilled operators **Cons:**Not scientific, unknown if you have fully optimized hyperparameters

1. Grid Search

Grid search is slightly more sophisticated than manual tuning. It involves systematicallytesting multiple values of each hyperparameter, by automatically retraining the model for each value of the parameter. For example, you can perform a grid search for the optimal batch size by automatically trainingthemodel for batchsizes between10-100 samples, insteps of20. Themodel willrun5 times and the batch size selected will be the one which yields highest accuracy. **Pros:**Maps out the problem space and provides more opportunity for optimization **Cons:**Can be slow to run for large numbers of hyperparameter values

*Grid-search is used to find the optimalhyperparametersof a model which results in the most 'accurate' predictions.*Grid search is the simplest algorithm for hyperparameter tuning. Basically, we divide the domain of the hyperparameters into a discrete grid. Then, we try every combinationof values of this grid, calculating some performance metrics using cross-validation. The point of the grid that maximizes the average value in cross-validation, is the optimal combination of values for the hyperparameters.



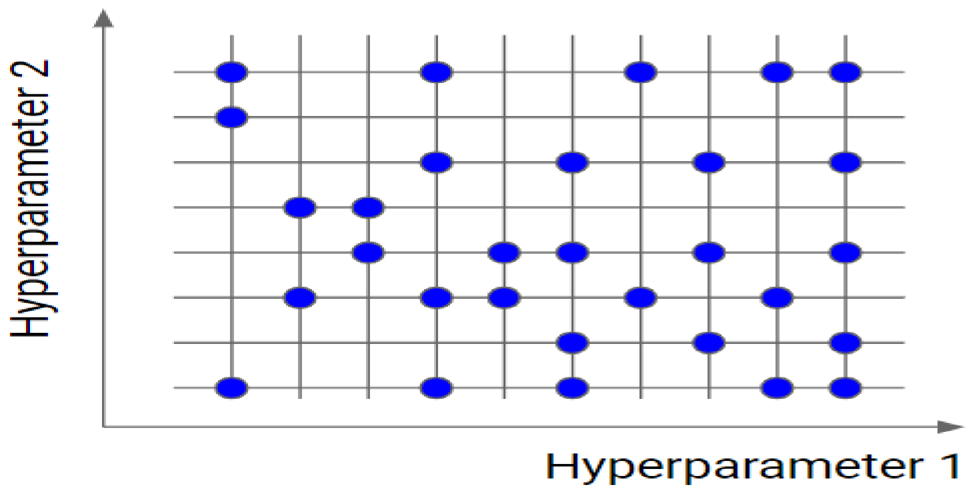
Exampleofagridsearch

Grid search is an exhaustive algorithm that spans all the combinations, so it can actually find the bestpoint in the domain. The great drawback is that it's very slow. Checking every combination of the space requires a lot of time that, sometimes, is not available. Don't forget that every point in the grid needs k- fold cross-validation, which requires *k*training steps. So, tuning the hyperparameters of a model in this way can be quite complex and expensive. However, if we look for the best combination of values of the hyperparameters, grid search is a very good idea.

2. Random Search

According to a 2012 research study by James Bergstra and Yoshua Bengio, testing randomized values of hyperparameters is actually more effective than manual search or grid search. In other words, instead of testing systematically to cover "promising areas" of the problem space, it is preferable to test random values drawn from the entire problem space. **Pros:**According to the study, provides higher accuracy with less training cycles, for problems with high dimensionality**Cons:**Results are unintuitive, difficult to understand "why" hyperparameter values were chosen

Randomsearch is similar togridsearch, but instead of usingall thepoints inthe grid, ittests only a randomly selected subset of these points. The smaller this subset, the faster but less accurate the optimization. The larger this dataset, the more accuratethe optimization but the closer to a grid search.



Example of random search

Randomsearch is a veryusefuloption when you haveseveralhyperparameters witha fine-grained gridof values. Using a subset made by 5-100 randomly selected points, we are able to get a reasonably good set of values of thehyperparameters. Itwillnot likelybe thebest point, butit canstillbea goodset of values that gives us a good model.

3. Bayesian Optimization

Bayesian optimization is a technique which tries to approximate the trained model with different possible hyperparameter values. To simplify, bayesian optimization trains the model with different hyperparameter values, and observes the function generated for the model by each set of parameter values. It does this over and over again, each time selecting hyperparameter values that are slightly different and can help plot the next relevant segment of the problem space. Similar to sampling methods in statistics, the algorithm ends up with a list of possible hyperparameter value sets and model functions, from which it predicts the optimal function across the entire problem set. **Pros:**The original study and practical experience from the industry shows that bayesian optimization results in significantly higher accuracy compared to random search. **Cons:**Like random search, results are not intuitive and difficult to improve on, even by trained operators

The Bayesian optimization method takes a different approach. This method treats the search for the optimal hyperparameters as an optimization problem. When choosing the next hyperparameter combination, this methodconsiders theprevious evaluationresults. Itthenapplies a probabilistic function to select the combination that will probably yield the best results. This method discovers a fairly good hyperparameter combination in relatively few iterations. Data scientists choose a probabilistic modelwhen the objective function is unknown. That is, there is no analytical expression to maximize or minimize. The data scientists apply the learning algorithm to a data set, use the algorithm’s results to define the objective function, and take the various hyperparameter combinations as the input domain.The probabilistic model is based on past evaluation results. It estimates the probability of a hyperparameter combination’s objective function result:

$$P(\text{result}|\text{hyperparameters})$$

Shree SATHYAM COLLEGE OF ENGG & TECH

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

AD3501 / DEEPLEARNING - NOTES

UNIT - V AUTOENCODERS AND GENERATIVE MODELS

Autoencoders: Undercomplete autoencoders -- Regularized autoencoders -- Stochastic encoders and decoders -- Learning with autoencoders; Deep Generative Models: Variational autoencoders – Generative adversarial networks.

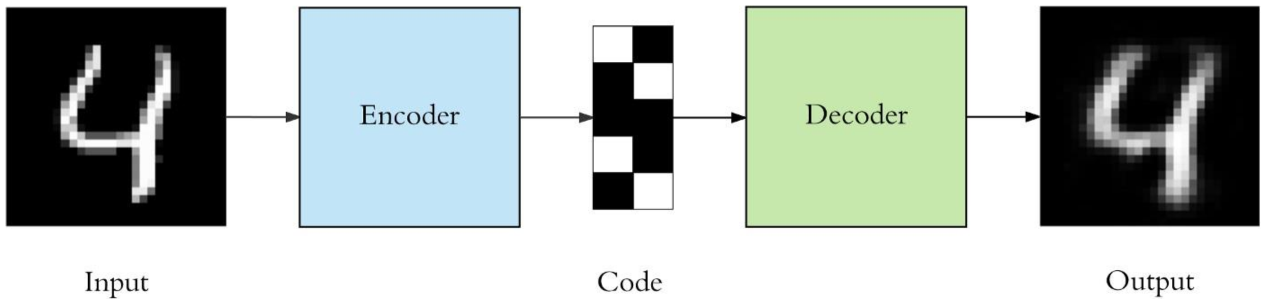
--=====

1. Introduction to Autoencoders

- Autoencoders are a specific type of feed forward neural networks where the input is thesame as the output.
- They compress the input into a lower-dimensional *code* and then reconstruct the output from this representation.
- The code is a compact “summary” or “compression” of the input, also called the *latent-space representation.*

Components

- An autoencoder consists of 3 components: encoder, code and decoder.
- The encoder compresses the input and produces the code,
- the decoder then reconstructs the input only using this code.



To build an autoencoder we need 3 things:

- an encoding method, decoding method, and a loss function to compare the output with the target.

Properties of Autoencoders

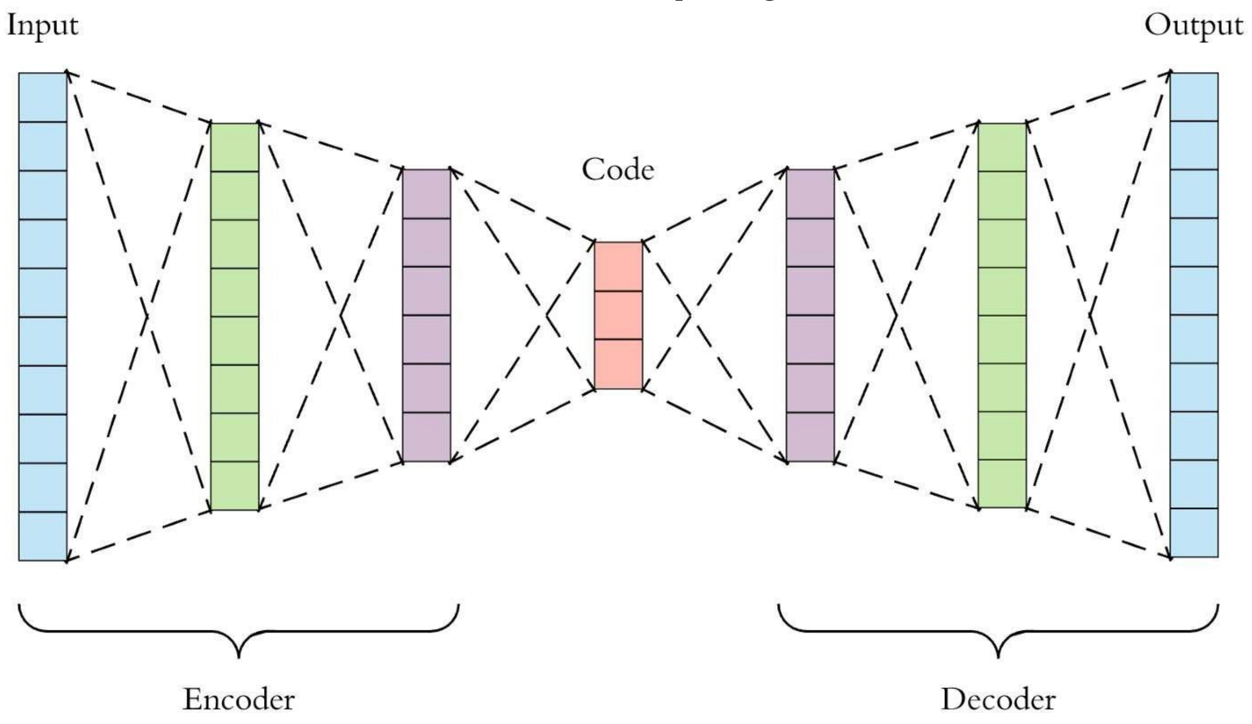
Autoencoders are mainly a dimensionality reduction (or compression) algorithm with a couple of important properties:

- **Data-specific:** Autoencoders are only able to meaningfully compress data similar to what they have been trained on. Since they learn features specific for the given training data,
- **Lossy:** The output of the autoencoder will not be exactly the same as the input, it will bea close but degraded representation. If you want lossless compression they are not the way to go.
- **Unsupervised:** To train an autoencoder we don’t need to do anything fancy, just throw the raw input data at it. Autoencoders are considered an *unsupervised* learning technique

since they don’t need explicit labels to train on. But to be more precise they are *self-supervised* because they generate their own labels from the training data.

Autoencoder Architecture:

The network architecture for autoencoders can vary between a simple FeedForward network, LSTM network or Convolutional Neural Network depending on the use case.

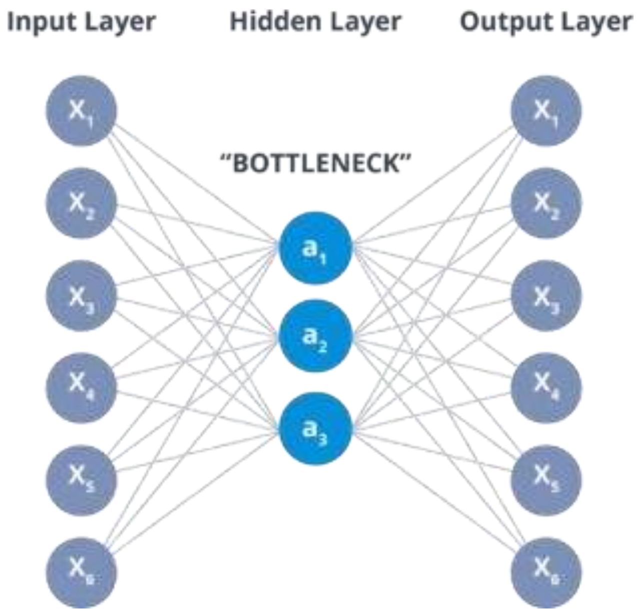


- This is a more detailed visualization of an autoencoder.
- First the input passes through the encoder, which is a fully-connected ANN, to produce the code.
- The decoder, which has the similar ANN structure, then produces the output only using the code.
- The goal is to get an output identical with the input.
- Note that the decoder architecture is the mirror image of the encoder.
- This is not a requirement but it’s typically the case.
- The only requirement is the dimensionality of the input and output needs to be the same. Anything in the middle can be played with.

- The layer between the encoder and decoder, ie. the code is also known as ***Bottleneck / latent-space representation.***
- This is a well-designed approach to decide which aspects of observed data are relevant information and what aspects can be discarded.

It does this by balancing two criteria :

- Compactness of representation, measured as the compressibility.
- It retains some behaviourally relevant variables from the input.



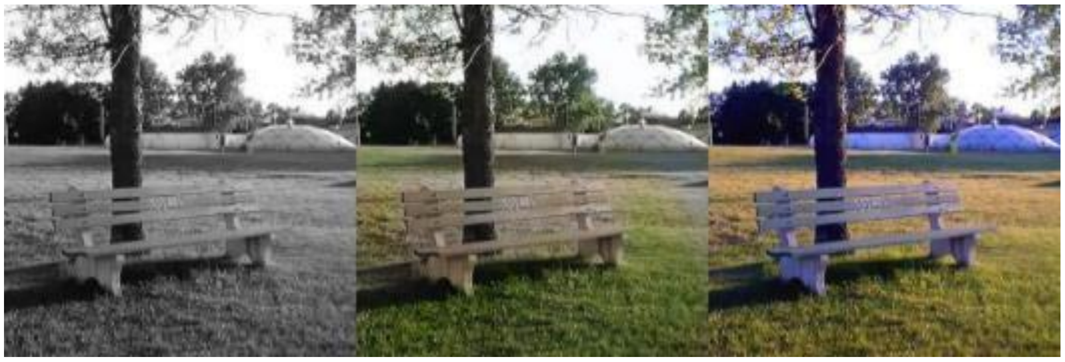
There **are 4 hyperparameters that we need to set before training an autoencoder**:

- Code size: number of nodes in the middle layer. Smaller size results in more compression.
- Number of layers: the autoencoder can be as deep as we like. In the figure above we have 2 layers in both the encoder and decoder, without considering the input and output.
- Number of nodes per layer: the autoencoder architecture we're working on is called a *stacked autoencoder* since the layers are stacked one after another. Usually stacked autoencoders look like a "sandwitch". The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. Also the decoder is symmetric to the encoder in terms of layer structure. As noted above this is not necessary and we have total control over these parameters.
- Loss function: we either use *mean squared error (mse)* or *binary crossentropy*. If the input values are in the range [0, 1] then we typically use crossentropy, otherwise we use the mean squared error.

Autoencoders are trained the same way as ANNs via backpropagation.

Applications of Autoencoders

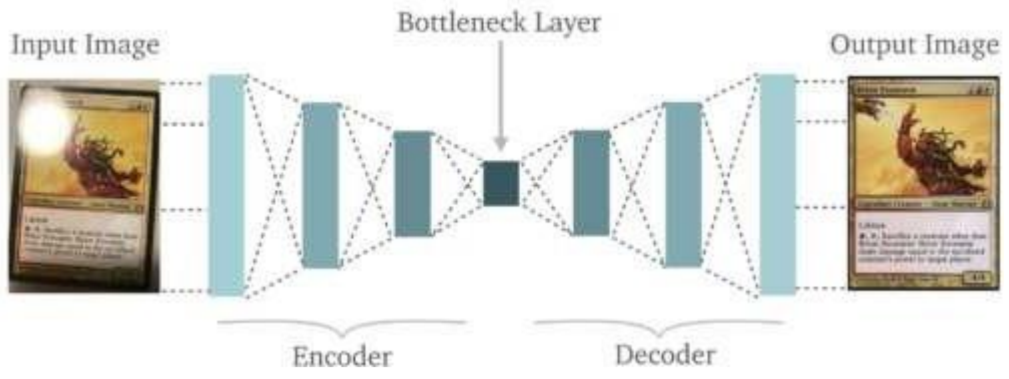
Image Coloring



Autoencoders are used for converting any black and white picture into a colored image. Depending on what is in the picture, it is possible to tell what the color should be.

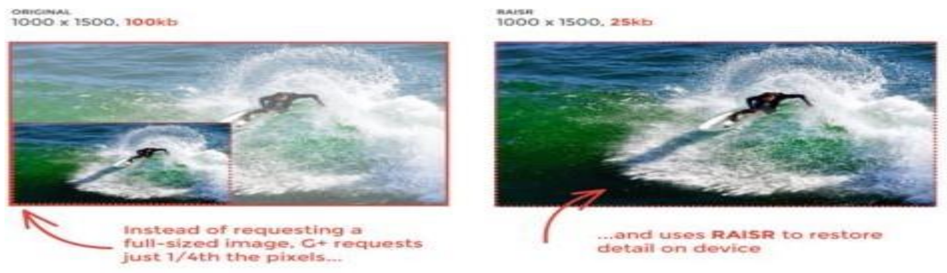
Feature variation

It extracts only the required features of an image and generates the output by removing any noise or unnecessary interruption.



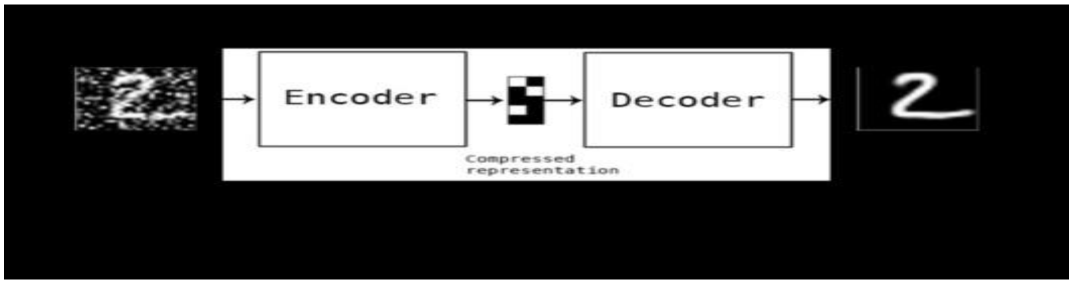
Dimensionality Reduction

The reconstructed image is the same as our input but with reduced dimensions. It helps in providing the similar image with a reduced pixel value.


ORIGINAL
1000 x 1500, 100kb
RAISR
1000 x 1500, 25kb
Instead of requesting a full-sized image, G+ requests just 1/4th the pixels...
...and uses RAISR to restore detail on device

Denoising Image

The input seen by the autoencoder is not the raw input but a stochastically corrupted version. A denoising autoencoder is thus trained to reconstruct the original input from the noisy version.



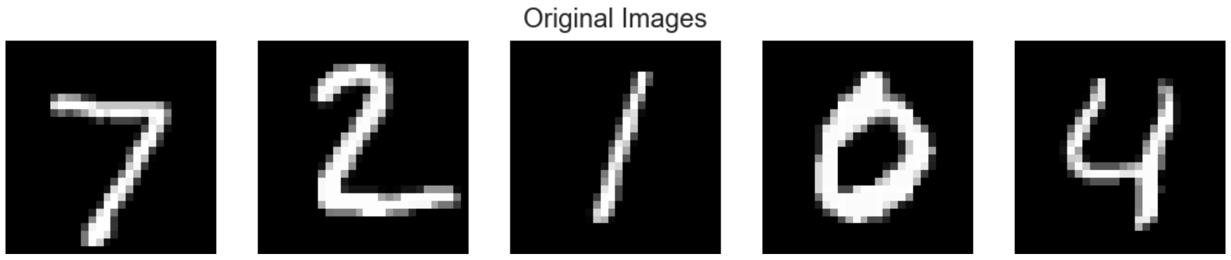
Watermark Removal

It is also used for removing watermarks from images or to remove any object while filming a video or a movie.

Implementation

Now let’s implement an autoencoder for the following architecture, 1 hidden layer in the encoder and decoder.

We will use the extremely popular MNIST dataset as input. It contains black-and-white images of handwritten digits.



They’re of size 28x28 and we use them as a vector of 784 numbers between [0, 1]

We will now implement the autoencoder with Keras. The hyperparameters are: 128 nodes in the hidden layer, code size is 32, and binary crossentropy is the loss function.

Code:

Let’s import the required libraries

```
import numpy as np
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
import matplotlib.pyplot as plt
```

Declaration of Hidden Layers and Variables

```
# this is the size of our encoded representations
encoding_dim = 32 # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# this is our input placeholder
input_img = Input(shape=(784,))
```

```
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)

# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)

# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)

# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)

# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))

# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]

# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))

# configure our model to use a per-pixel binary crossentropy loss, and the Adadelta optimizer:
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
```

Preparing the input data (MNIST Dataset)

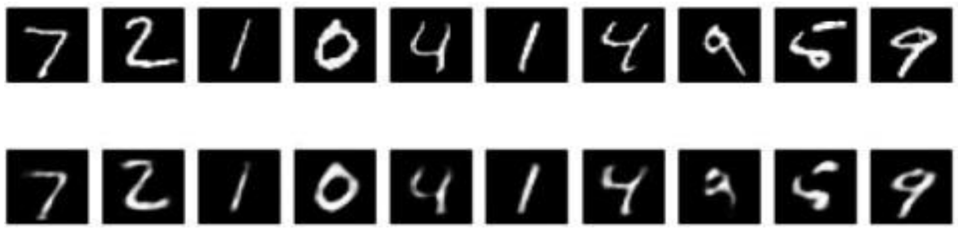
```
(x_train, _), (x_test, _) = mnist.load_data()
# normalize all values between 0 and 1 and we will flatten the 28x28 images
into vectors of size 784.

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

print x_train.shape
print x_test.shape
```

Training Autoencoders for 50 epochs

```
autoencoder.fit(x_train, x_train,
epochs=50,
batch_size=256,
shuffle=True,
validation_data=(x_test, x_test))
# encode and decode some digits
# note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

Visualizing the reconstructed inputs and the encoded representations using Matplot lib

```
n = 20 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
# display original
ax = plt.subplot(2, n, i + 1)
plt.imshow(x_test[i].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)

 # display reconstruction
ax = plt.subplot(2, n, i + 1 + n)
plt.imshow(decoded_imgs[i].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
plt.show()
```



2. Under complete autoencoders

- Under complete autoencoders is an unsupervised neural network that you can use to generate a compressed version of the input data.
- It is done by taking in an image and trying to predict the same image as output, thus reconstructing the image from its compressed bottleneck region.
- The primary use for autoencoders like these is generating a latent space or bottleneck, which forms a compressed substitute of the input data and can be easily decompressed back with the help of the network when needed.

Under complete autoencoders learn features by minimizing the same loss function:

$$L(x, g(f(x))),$$

Where L is the loss function penalizing $g(f(x))$ from diverging from the original input x. L can be a mean squared error or even a mean absolute error.

- Goal of the Autoencoder is to capture the most important features present in the data.
- Undercomplete autoencoders have a smaller dimension for hidden layer compared to the input layer. This helps to obtain important features from the data.
- Objective is to minimize the loss function by penalizing the $g(f(x))$ for being different from the input x.
- When decoder is linear and we use a mean squared error loss function then undercomplete autoencoder generates a reduced feature space similar to PCA
- We get a powerful nonlinear generalization of PCA when encoder function f and decoder function g are non linear.
- Undercomplete autoencoders do not need any regularization as they maximize the probability of data rather than copying the input to the output.

Advantages

- Undercomplete autoencoders, with code dimension less than the input dimension, can learn the most salient features of the data distribution.

Disadvantages

- We have seen that these autoencoders fail to learn anything useful if the encoder and decoder are given too much capacity.
- A similar problem occurs if the hidden code is allowed to have dimension equal to the input, and in the overcomplete case in which the hidden code has dimension greater than the input. In these cases, even a linear encoder and a linear decoder can learn to copy the input to the output without learning anything useful about the data distribution.
- Ideally, one could train any architecture of autoencoder successfully, choosing the code dimension and the capacity of the encoder and decoder based on the complexity of distribution to be modelled.

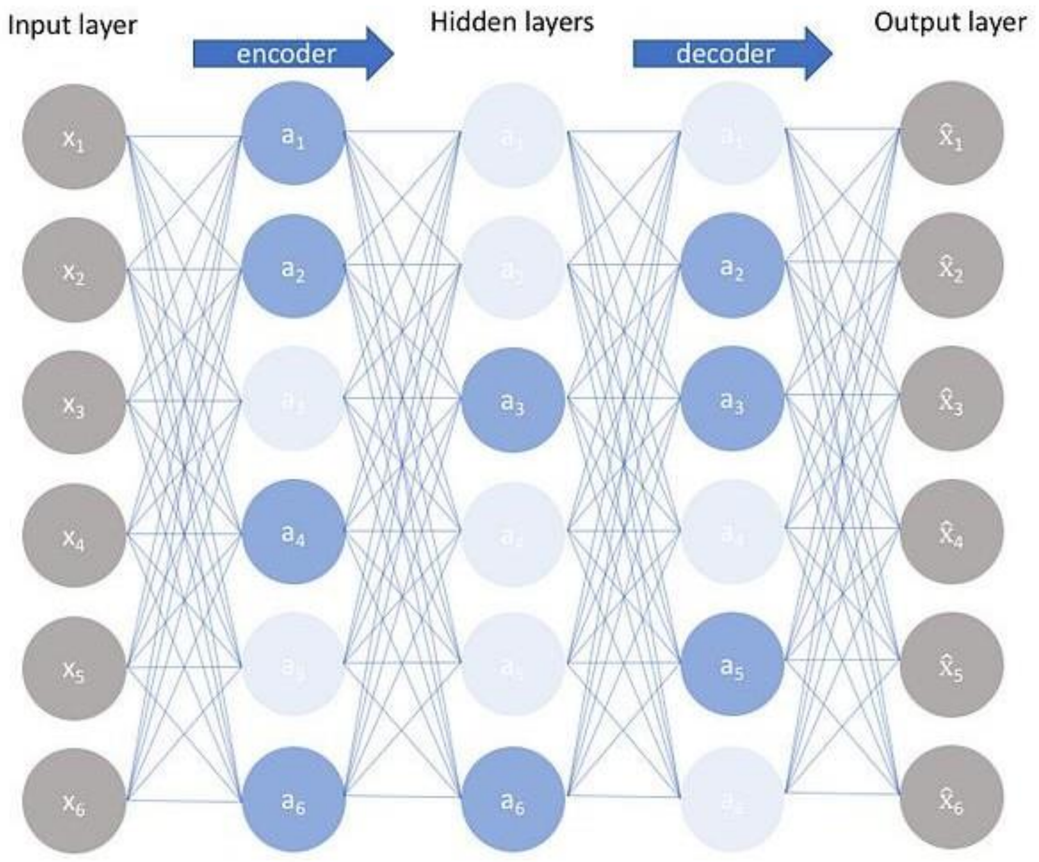
---===

3. Regularized autoencoders

Regularized autoencoders provide the ability to do so. Rather than limiting the model capacity by keeping the encoder and decoder shallow and the code size small, regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output. These other properties include sparsity of the representation, smallness of the derivative of the representation, and robustness to noise or to missing inputs. A regularized autoencoder can be nonlinear and overcomplete but still learn something useful about the data distribution, even if the model capacity is great enough to learn a trivial identity function.

In addition to the methods described here, which are most naturally interpreted as regularized autoencoders, nearly any generative model with latent variables and equipped with an inference procedure (for computing latent representations given input) may be viewed as a particular form of autoencoder.
In practice, we usually find two types of regularized autoencoder: the **sparse** autoencoder and the **denoising** autoencoder.

(i) Sparse autoencoder : Sparse autoencoders are typically used to learn features for another task such as classification. An autoencoder that has been regularized to be sparse must respond to unique statistical features of the dataset it has been trained on, rather than simply acting as an identity function. In this way, training to perform the copying task with a sparsity penalty can yield a model that has learned useful features as a byproduct.
Another way we can constraint the reconstruction of autoencoder is to impose a constraint in its loss. We could, for example, add a regularization term in the loss function. Doing this will make our autoencoder learn sparse representation of data.



There are actually two different ways to construct our sparsity penalty: L1 regularization and KL-divergence.

Why L1 Regularization Sparse

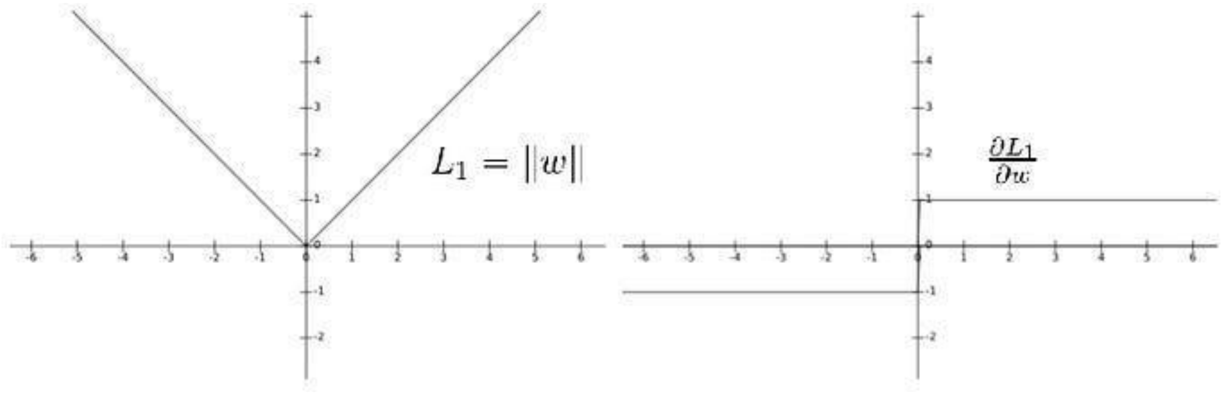
L1 regularization and L2 regularization are widely used in machine learning and deep learning. L1 regularization adds "absolute value of magnitude" of coefficients as penalty term while L2 regularization adds "squared magnitude" of coefficient as a penalty term.

Although L1 and L2 can both be used as regularization term, the key difference between them is that L1 regularization tends to shrink the penalty coefficient to zero while L2 regularization would move coefficients towards zero but they will never reach. Thus L1 regularization is often used as a method of feature extraction. But why L1 regularization leads to sparsity?

Consider that we have two loss functions L1 and L2 which represent L1 regularization and L2 regularization respectively.

$$L_1 = \|w\| \, , L_2 = w^2$$

Gradient descent is always used in optimizing neural networks. If we plot these two loss functions and their derivatives, it looks like this:



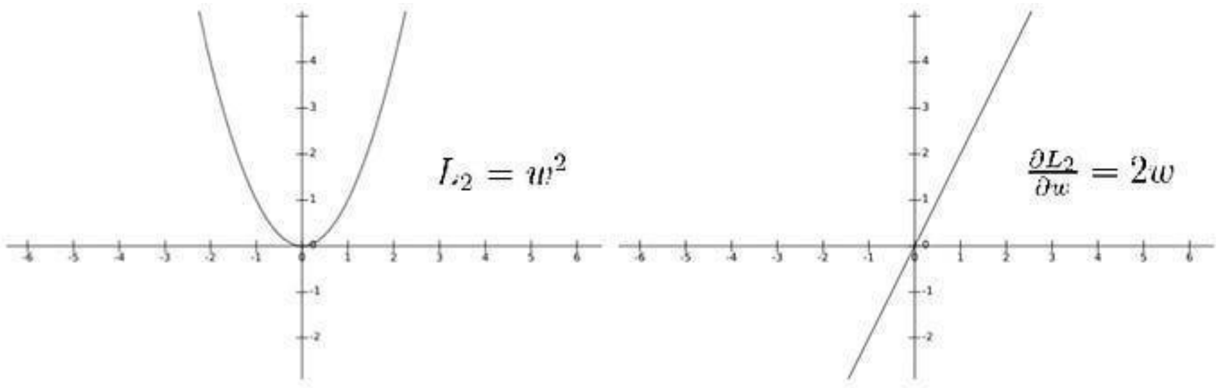
L1 regularization and its derivative

Loss Function

Finally, after the above analysis, we get the idea of using L1 regularization in sparse autoencoder and the loss function is as below:

$$Obj = L(x, \hat{x}) + regularization + \lambda \sum_i |a_i^{(h)}|$$

Except for the first two terms, we add the third term which penalizes the absolute value of the vector of activations a in layer h for sample i. Then we use a hyperparameter to control its effect on the whole loss function. And in this way, we do build a sparse autoencoder.

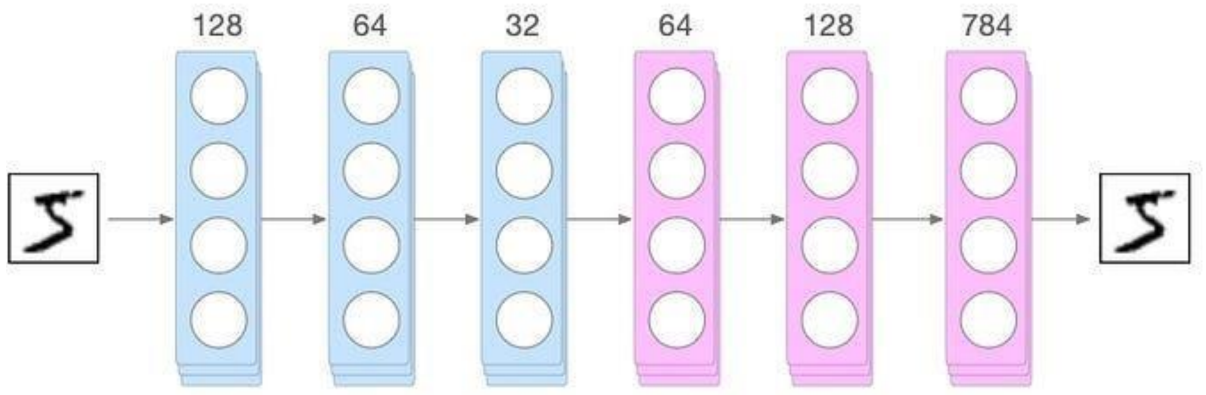


L2 regularization and its derivative

We can notice that for L1 regularization, the gradient is either 1 or -1 except when w=0, which means that L1 regularization will always move w towards zero with same step size (1 or -1) regardless of the value of w. And when w=0, the gradient becomes zero and no update will be made anymore. However, for L2 regularization things are different. L2 regularization will also move w towards zero but the step size becomes smaller and smaller which means that w will never reach zero.

Visualization

We tried to build a deep autoencoder and train it on MNIST dataset without L1 regularization and with regularization. The structure of this deep autoencoder is plotted as below:

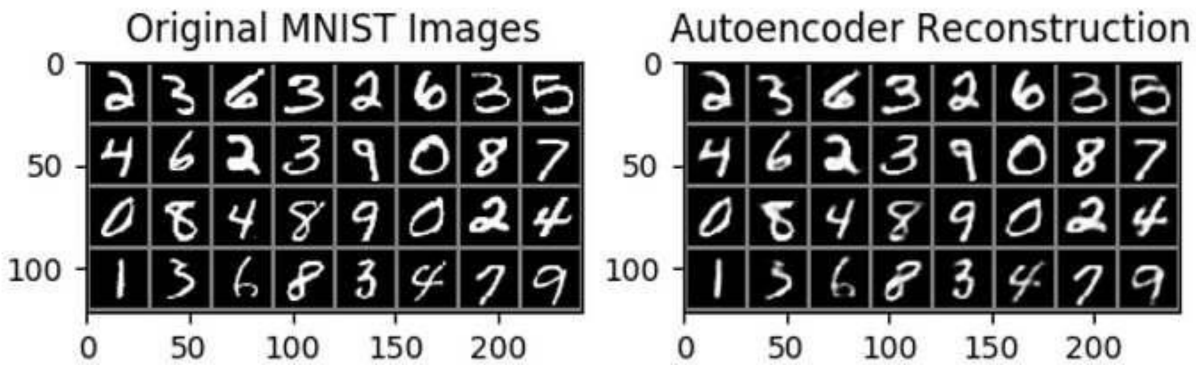


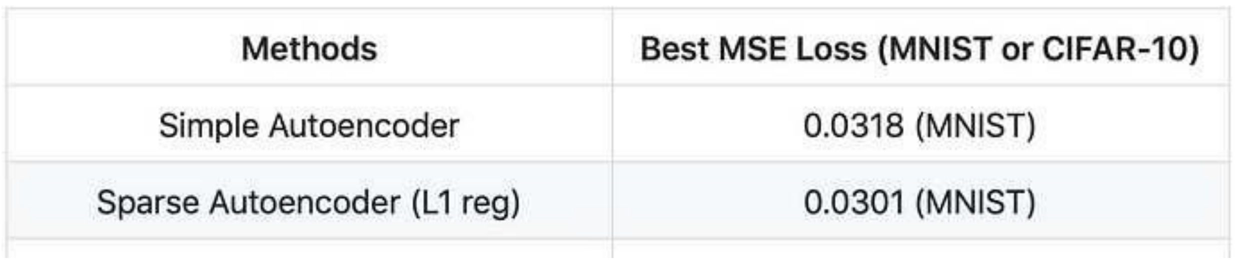
Code:

```
input_size = 784
hidden_size = 64
output_size = 784

x = Input(shape=(input_size,))

# Encoder
```

```
h = Dense(hidden_size, activation='relu', activity_regularizer=regularizers.l1(10e-5))(x)

# Decoder
r = Dense(output_size, activation='sigmoid')(h)

autoencoder = Model(input=x, output=r)
autoencoder.compile(optimizer='adam', loss='mse')
```

Notice in our hidden layer, we added an l1 activity regularizer, that will apply a penalty to the loss function during the optimization phase. As a result, the representation is now sparser compared to the vanilla autoencoder.
And after 100 epochs of training using 128 batch size and Adam as the optimizer, we got below results:





Methods	Best MSE Loss (MNIST or CIFAR-10)
Simple Autoencoder	0.0318 (MNIST)
Sparse Autoencoder (L1 reg)	0.0301 (MNIST)

(ii) Denoising autoencoder :
Denoising autoencoders are a specific type of neural network that enables unsupervised learning of data representations or encodings. Their primary objective is to reconstruct the original version of the input signal corrupted by noise. This capability proves valuable in problems such as image recognition or fraud detection, where the goal is to recover the original signal from its noisy form.
An autoencoder consists of two main components:

- **Encoder:** This component maps the input data into a low-dimensional representation or encoding.

- **Decoder:** This component returns the encoding to the original data space.

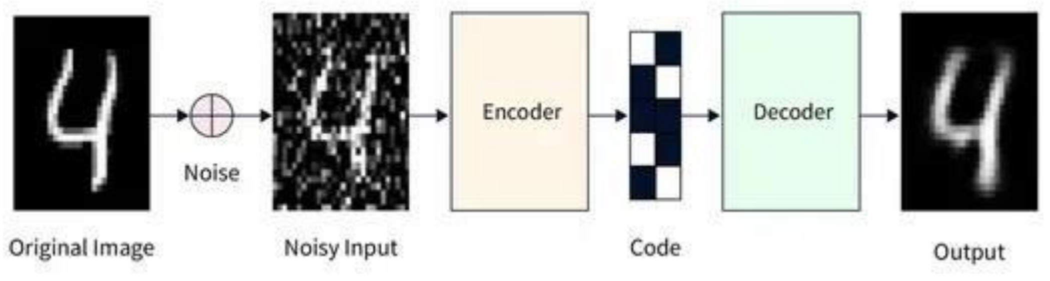
During the training phase, present the autoencoder with a set of clean input examples along with their corresponding noisy versions. The objective is to learn a task using an encoder-decoder architecture that efficiently transforms noisy input into clean output.
Architecture of DAE
The denoising autoencoder (DAE) architecture is similar to a standard autoencoder. It consists of two main components:

Encoder

- The encoder creates a neural network equipped with one or more hidden layers.
- Its purpose is to receive noisy input data and generate an encoding, which represents a low-dimensional representation of the data.
- Understand an encoder as a compression function because the encoding has fewer parameters than the input data.

Decoder

- Decoder acts as an expansion function, which is responsible for reconstructing the original data from the compressed encoding.
- It takes as input the encoding generated by the encoder and reconstructs the original data.
- Like encoders, decoders are implemented as neural networks featuring one or more hidden layers.

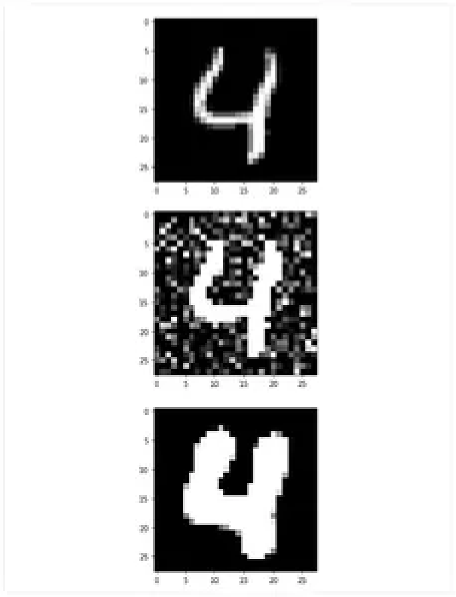


During the training phase, present the denoising autoencoder (DAE) with a collection of clean input examples along with their respective noisy counterparts. The objective is to acquire a function that maps a noisy input to a relatively clean output using an encoder-decoder architecture. To achieve this, a reconstruction loss function is typically employed to evaluate the disparity between the clean input and the reconstructed output. A DAE is trained by minimizing this loss through the use of backpropagation, which involves updating the weights of both encoder and decoder components.

Code:
```
x = Input(shape=(28, 28, 1))
```

```
# Encoder
conv1_1 = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
pool1 = MaxPooling2D((2, 2), padding='same')(conv1_1)
conv1_2 = Conv2D(32, (3, 3), activation='relu', padding='same')(pool1)
h = MaxPooling2D((2, 2), padding='same')(conv1_2)


# Decoder
conv2_1 = Conv2D(32, (3, 3), activation='relu', padding='same')(h)
up1 = UpSampling2D((2, 2))(conv2_1)
conv2_2 = Conv2D(32, (3, 3), activation='relu', padding='same')(up1)
up2 = UpSampling2D((2, 2))(conv2_2)
r = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(up2)

autoencoder = Model(input=x, output=r)
autoencoder.compile(optimizer='adam', loss='mse')
```

OUTPUT:



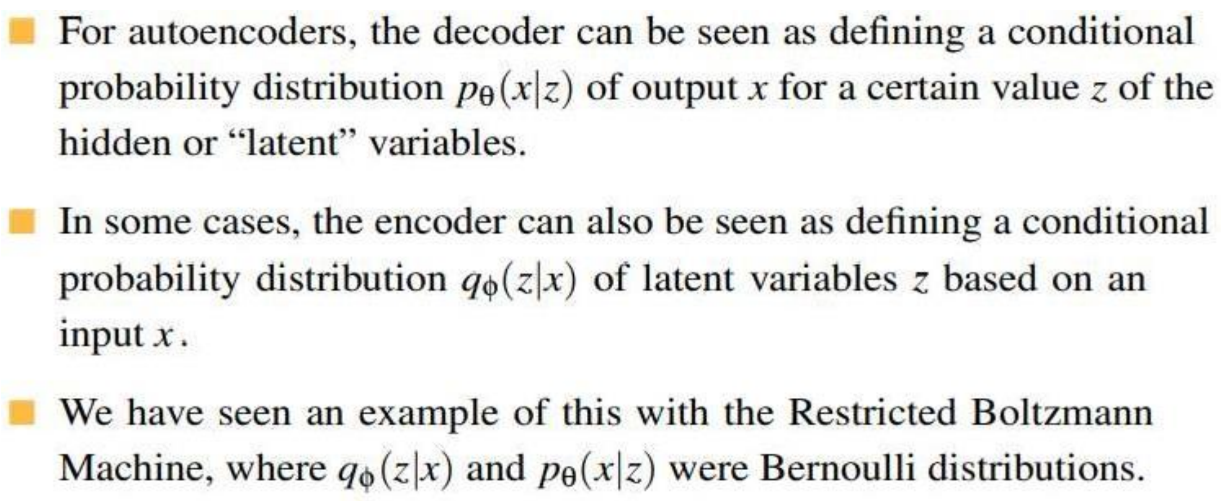
Applications of Denoising Autoencoders (DAEs) span a variety of domains, including computer vision, speech processing, and natural language processing.

Examples

- **Image Denoising**: DAEs are effective in removing noise from images, such as Gaussian noise or salt-and-pepper noise.
- **Fraud Detection**: DAEs can contribute to identifying fraudulent transactions by learning to reconstruct common transactions from their noisy counterparts.

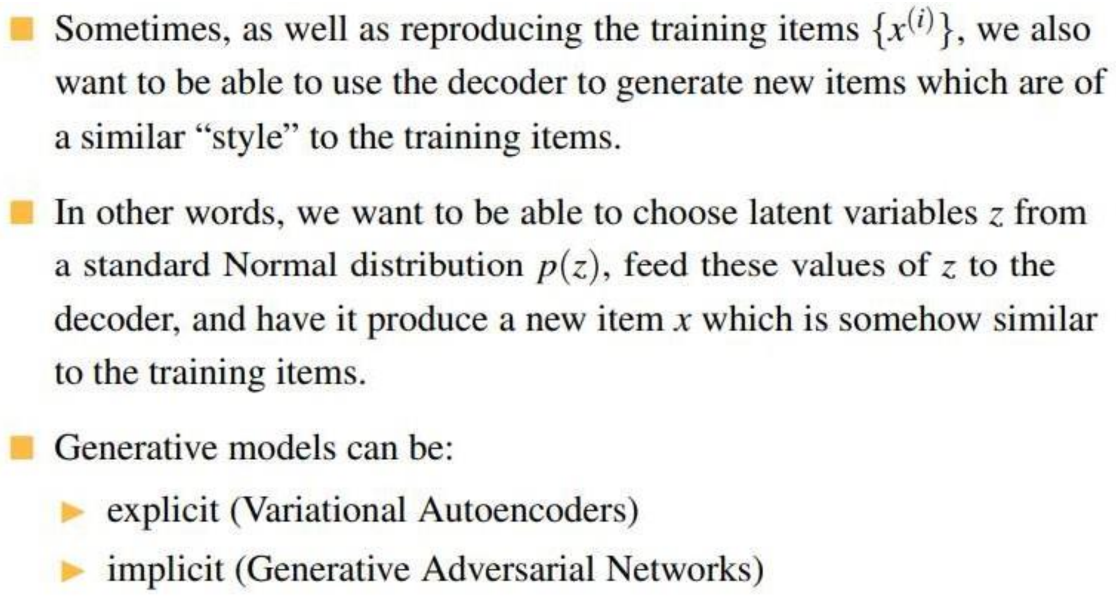
- **Data Imputation**: To reconstruct missing values from available data by learning, DAEs can facilitate data imputation in datasets with incomplete information.
- **Data Compression**: DAEs can compress data by obtaining a concise representation of the data in the encoding space.
- **Anomaly Detection**: Using DAEs, anomalies in a dataset can be detected by training a model to reconstruct normal data and then flag challenging inputs as potentially abnormal.

4.Stochastic Encoders and Decoders



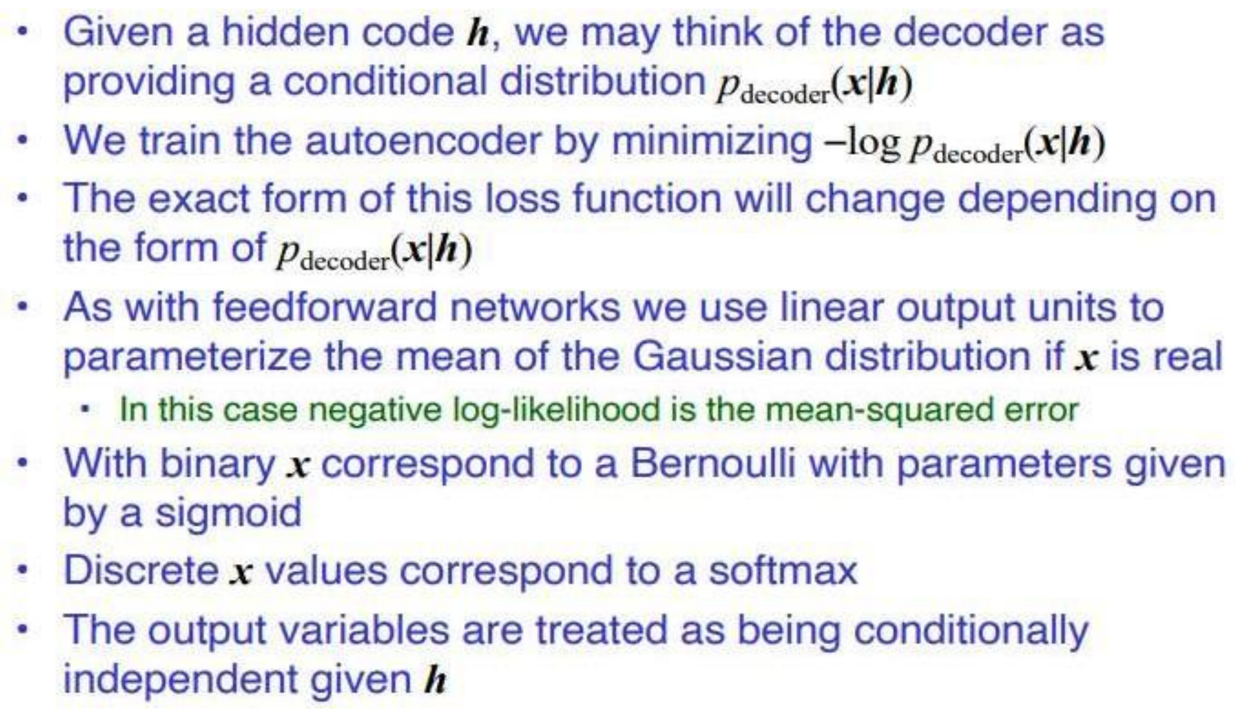
- For autoencoders, the decoder can be seen as defining a conditional probability distribution $p_\theta(x|z)$ of output x for a certain value z of the hidden or "latent" variables.
- In some cases, the encoder can also be seen as defining a conditional probability distribution $q_\phi(z|x)$ of latent variables z based on an input x.
- We have seen an example of this with the Restricted Boltzmann Machine, where $q_\phi(z|x)$ and $p_\theta(x|z)$ were Bernoulli distributions.

Generative Models

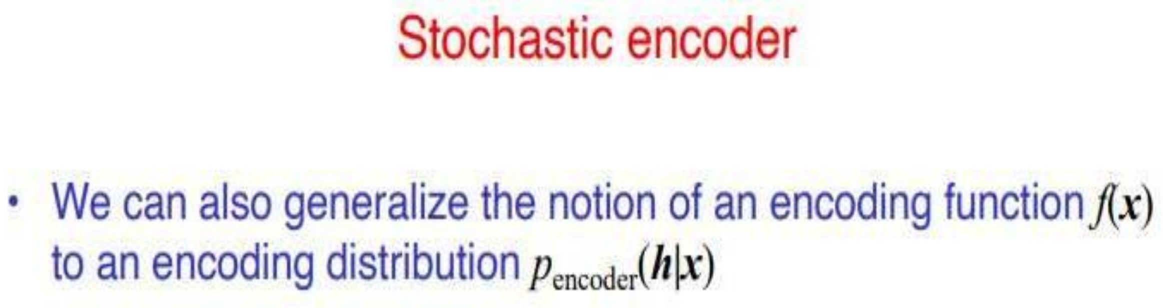


- Sometimes, as well as reproducing the training items $\{x^{(i)}\}$, we also want to be able to use the decoder to generate new items which are of a similar "style" to the training items.
- In other words, we want to be able to choose latent variables z from a standard Normal distribution $p(z)$, feed these values of z to the decoder, and have it produce a new item x which is somehow similar to the training items.
- Generative models can be:
  - explicit (Variational Autoencoders)
  - implicit (Generative Adversarial Networks)

Loss function for Stochastic Decoder

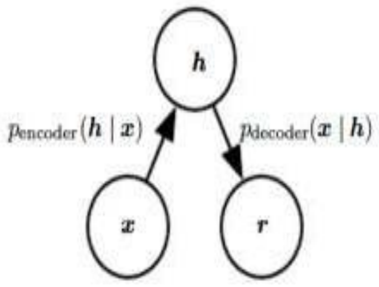


- Given a hidden code $\boldsymbol{h}$, we may think of the decoder as providing a conditional distribution $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$
- We train the autoencoder by minimizing $-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$
- The exact form of this loss function will change depending on the form of $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$
- As with feedforward networks we use linear output units to parameterize the mean of the Gaussian distribution if $\boldsymbol{x}$ is real
    - In this case negative log-likelihood is the mean-squared error
- With binary $\boldsymbol{x}$ correspond to a Bernoulli with parameters given by a sigmoid
- Discrete $\boldsymbol{x}$ values correspond to a softmax
- The output variables are treated as being conditionally independent given $\boldsymbol{h}$



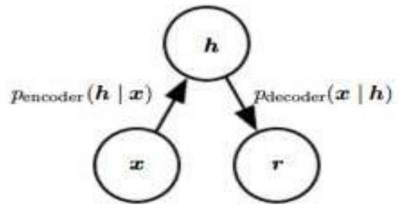
Stochastic encoder

- We can also generalize the notion of an encoding function $f(\boldsymbol{x})$ to an encoding distribution $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$

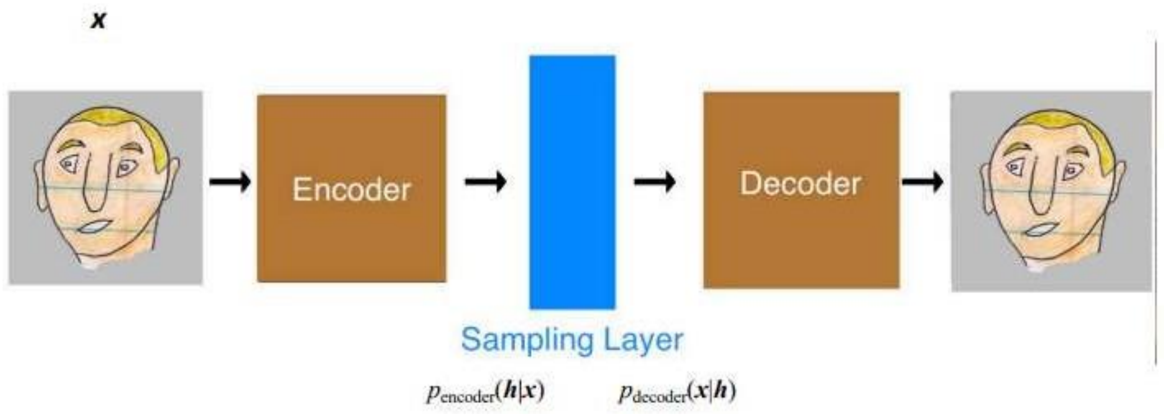

h
$p_{\text{encoder}}(h \mid x)$
$p_{\text{decoder}}(x \mid h)$
x
r

Structure of stochastic autoencoder

- Both the encoder and decoder are not simple functions but involve a distribution
- The output is sampled from a distribution $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ for the encoder and $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ for the decoder


$\boldsymbol{h}$
$p_{\text{encoder}}(\boldsymbol{h} \mid \boldsymbol{x})$
$p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$
$\boldsymbol{x}$
$\boldsymbol{r}$

Sampling $p_{\text{model}}(\boldsymbol{h}|\boldsymbol{x})$


x
Encoder
Decoder
Sampling Layer
$p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$
$p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$

--==

5. Learning with autoencoders; Deep Generative Models: Variational autoencoders

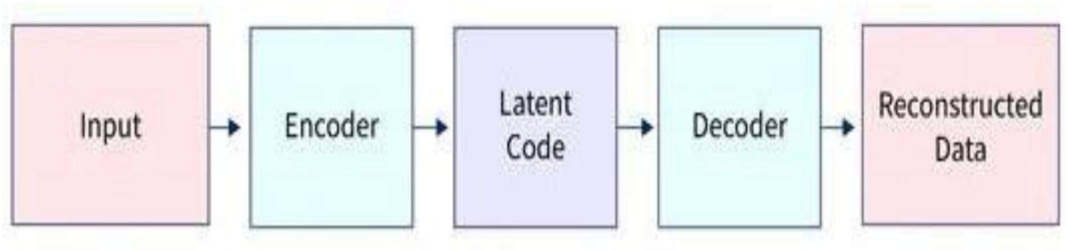
Variational Autoencoders (VAEs) are generative models explicitly designed to capture the underlying probability distribution of a given dataset and generate novel samples. They utilize an architecture that comprises an encoder-decoder structure. The encoder transforms input data into a latent form, and the decoder aims to reconstruct the original data based on this latent representation. The VAE is programmed to minimize the dissimilarity between the original and reconstructed data, enabling it to comprehend the underlying data distribution and generate new samples that conform to the same distribution.

One notable advantage of VAEs is their ability to generate new data samples resembling the training data. Because the VAE's latent space is continuous, the decoder can generate new data points that seamlessly interpolate among the training data points. VAEs find applications in various domains like density estimation and text generation.

The Architecture of Variational Autoencoder

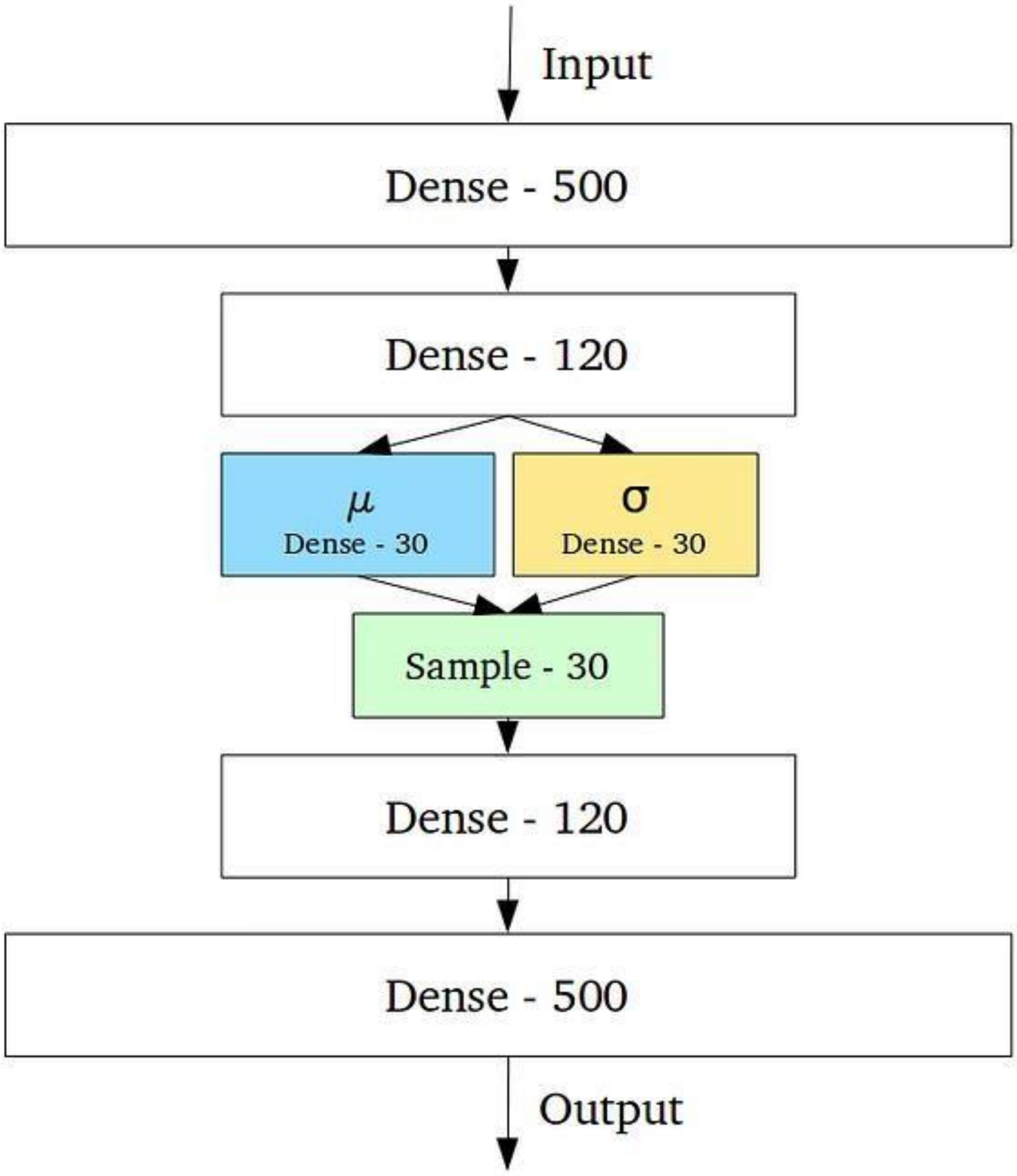
A VAE typically has two major components: An encoder connection and a decoder connection. An encoder network transforms The input data into a low-dimensional secret space, often called a "secret code".

Various neural network topologies, such as fully connected or convolutional neural networks, can be investigated for implementing encoder networks. The architecture chosen is based on the characteristics of the data. The encoder network produces essential parameters, such as the mean and variance of a Gaussian distribution, necessary for sampling and generating the latent code.



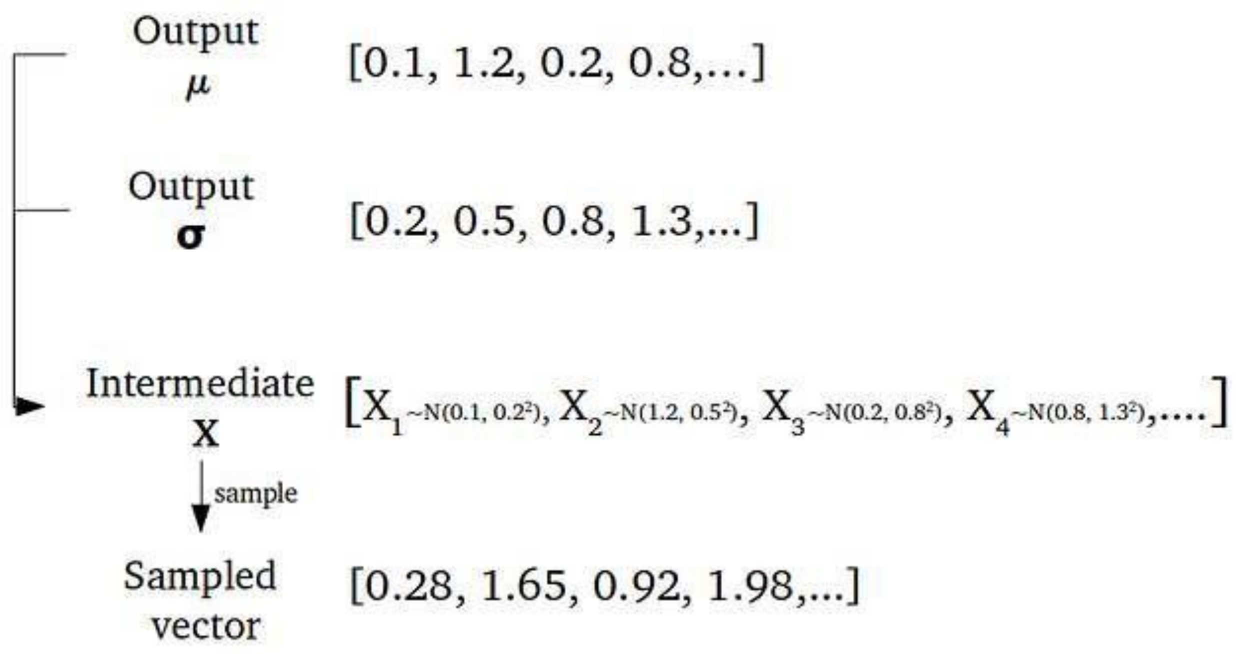
A VAE comprises an encoder network that maps input data to a latent code and a decoder network that conducts the inverse operation by translating the latent code back to the reconstruction data. By undergoing this training process, the VAE learns an optimized latent representation that captures the fundamental characteristics of the data, enabling precise reconstruction.

It achieves this by doing something that seems rather surprising at first: making its encoder not output an encoding vector of size *n*, rather, outputting two vectors of size n: a vector of means**, μ**, and another vector of standard deviations, **σ**.

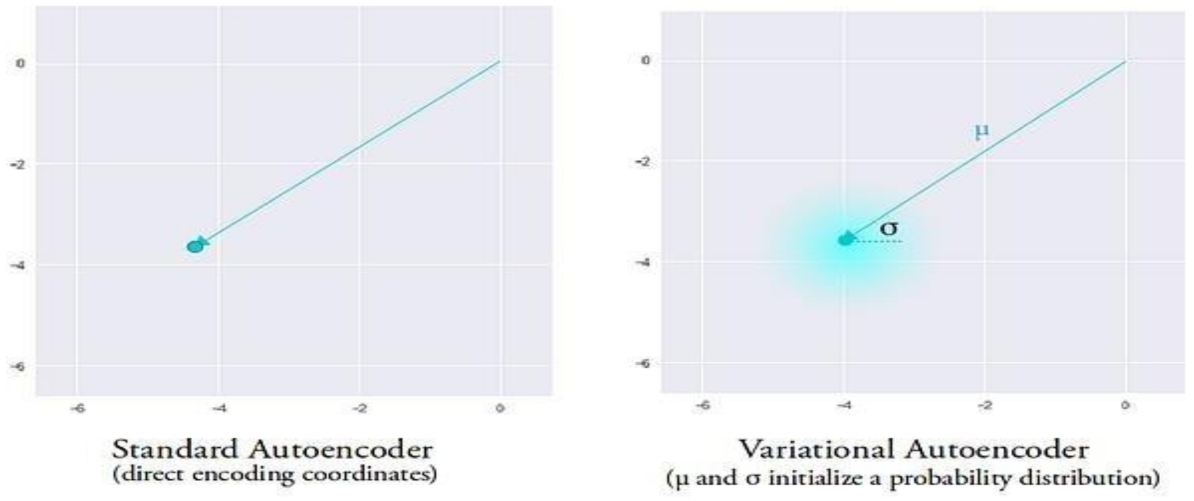


Variational Autoencoder

They form the parameters of a vector of random variables of length n, with the *i* th element of **μ** and **σ** being the mean and standard deviation of the *i* th random variable, **X** i, from which we sample, to obtain the sampled encoding which we pass onward to the decoder:



Stochastically generating encoding vectors
This stochastic generation means, that even for the same input, while the mean and standard deviations remain the same, the actual encoding will somewhat vary on every single pass simply due to sampling.



Code:

```
# build your encoder upto here. It can simply be a series of dense layers, a convolutional network
# or even an LSTM decoder. Once made, flatten out the final layer of the encoder, call it hidden.

# we use Keras to build the graph

latent_size = 5
```

```
mean = Dense(latent_size)(hidden)

# we usually don't directly compute the stddev σ
# but the log of the stddev instead, which is log(σ)
# the reasoning is similar to why we use softmax, instead of directly outputting
# numbers in fixed range [0, 1], the network can output a wider range of numbers which we can
later compress down
log_stddev = Dense(latent_size)(hidden)

def sampler(mean, log_stddev):
   # we sample from the standard normal a matrix of batch_size * latent_size (taking into
account minibatches)
   std_norm = K.random_normal(shape=(K.shape(mean)[0], latent_size), mean=0, stddev=1)
   # sampling from Z~N(μ, σ^2) is the same as sampling from μ + σX, X~N(0,1)
   return mean + K.exp(log_stddev) * std_norm

latent_vector = Lambda(sampler)([mean, log_stddev])
```

Output

---=======

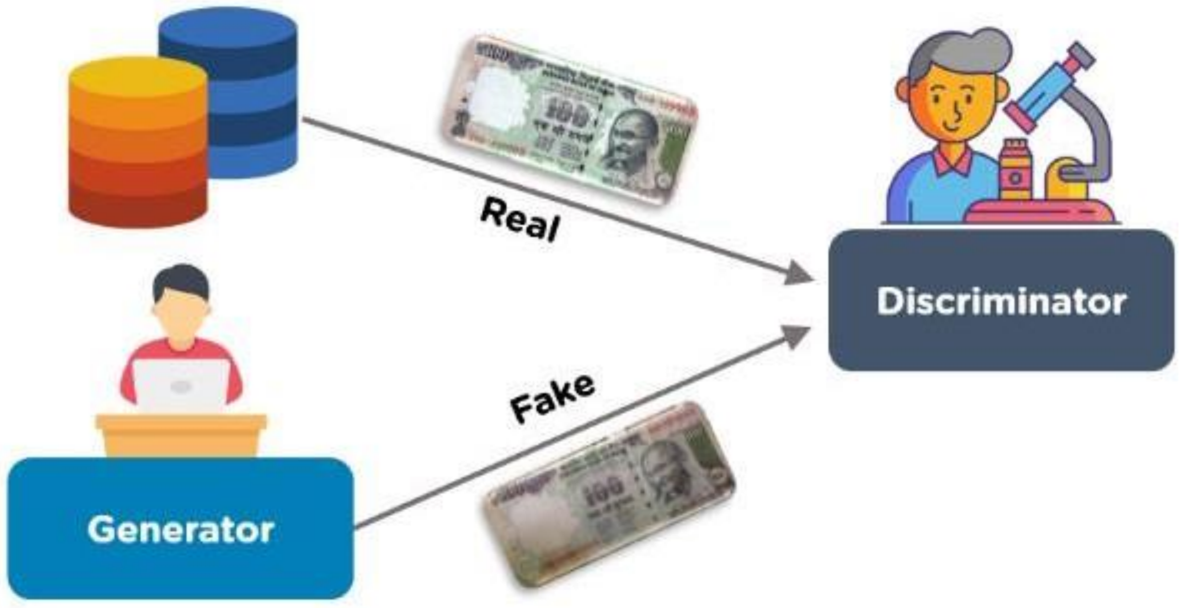
6. Learning with autoencoders; Deep Generative Models: Generative Adversarial Networks

Generative Adversarial Networks (GANs) were introduced in 2014 by Ian J. Goodfellow and co-authors. GANs perform unsupervised learning tasks in machine learning. It consists of 2 models that automatically discover and learn the patterns in input data.

The two models are known as Generator and Discriminator.

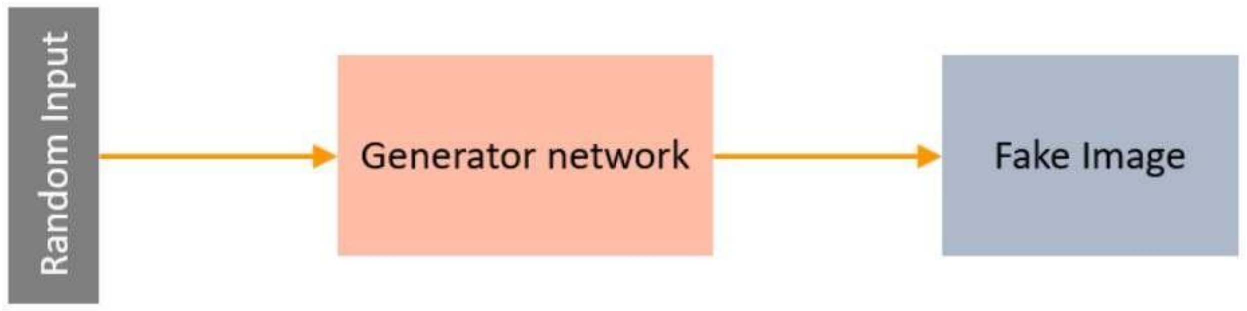
They compete with each other to scrutinize, capture, and replicate the variations within a dataset. GANs can be used to generate new examples that plausibly could have been drawn from the original dataset.

Shown below is an example of a GAN. There is a database that has real 100 rupee notes. The generator neural network generates fake 100 rupee notes. The discriminator network will help identify the real and fake notes.



What is a Generator?

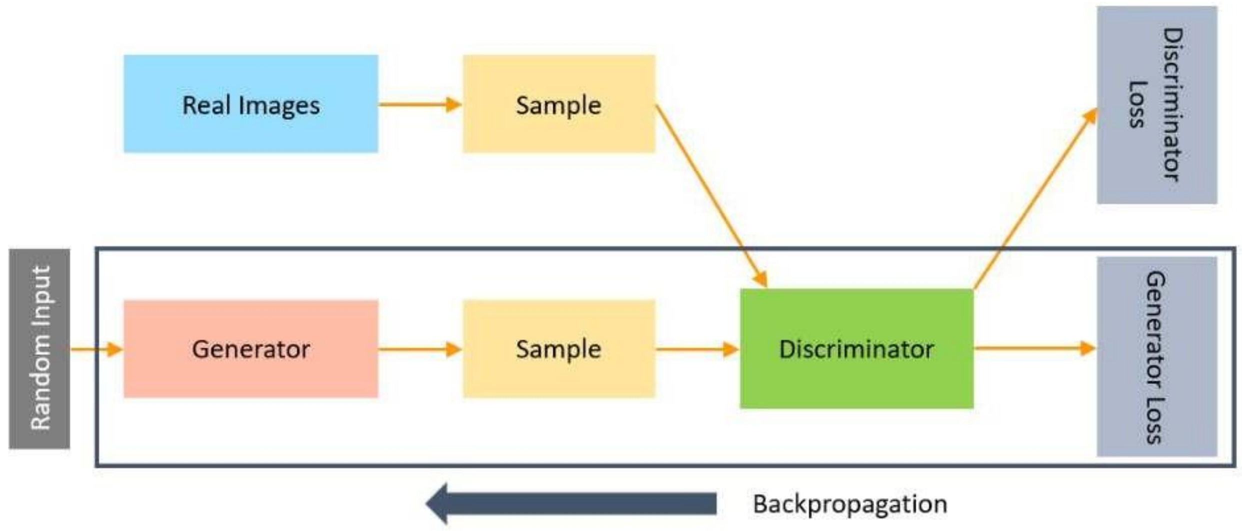
A Generator in GANs is a neural network that creates fake data to be trained on the discriminator. It learns to generate plausible data. The generated examples/instances become negative training examples for the discriminator. It takes a fixed-length random vector carrying noise as input and generates a sample.



The main aim of the Generator is to make the discriminator classify its output as real. The part of the GAN that trains the Generator includes:

- noisy input vector
- generator network, which transforms the random input into a data instance
- discriminator network, which classifies the generated data
- generator loss, which penalizes the Generator for failing to dolt the discriminator

The backpropagation method is used to adjust each weight in the right direction by calculating the weight's impact on the output. It is also used to obtain gradients and these gradients can help change the generator weights.

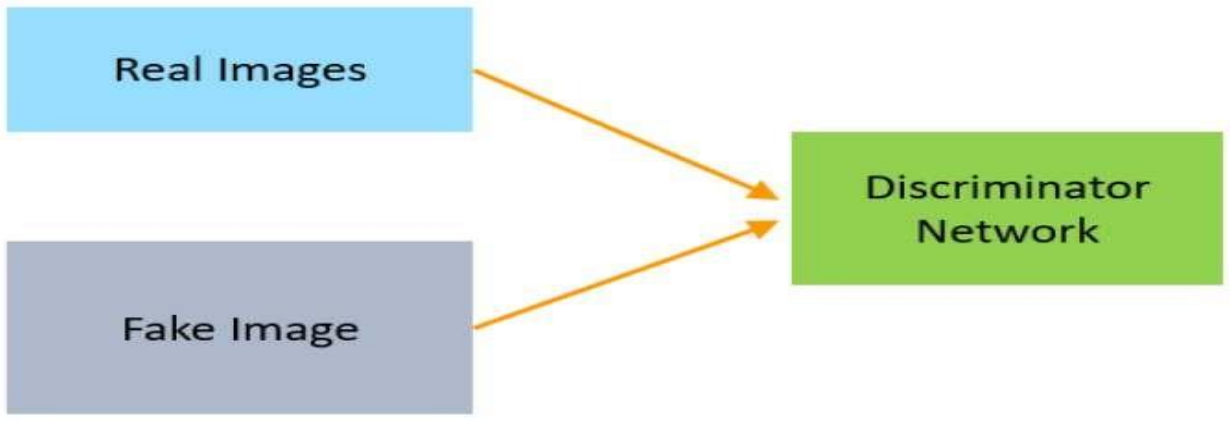


Let's see the next topic in this article on what GANs are, i.e., a Discriminator.

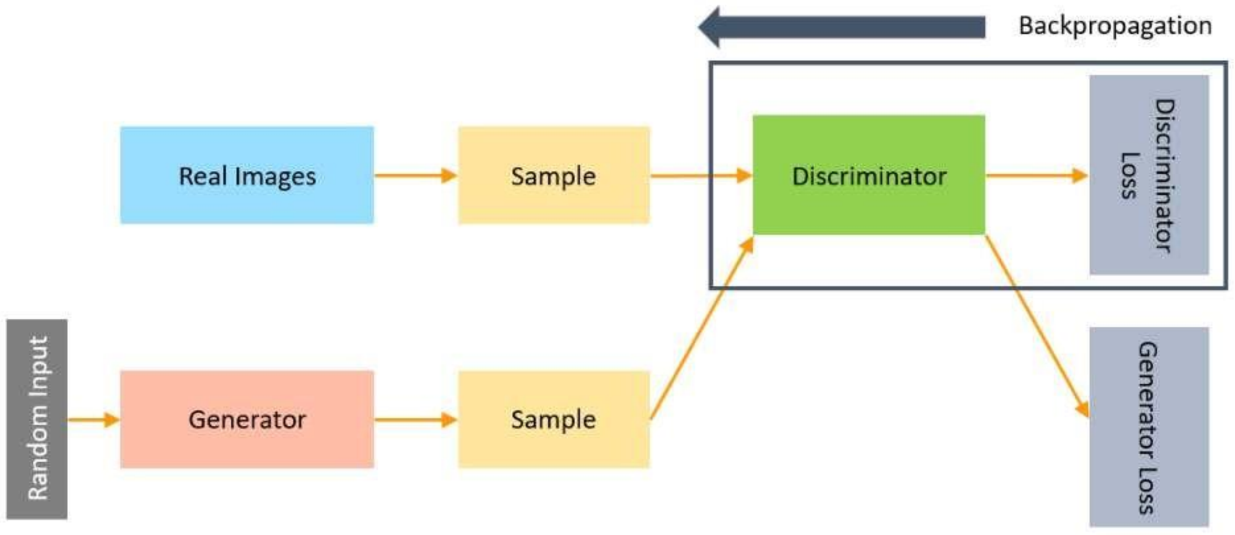
What is a Discriminator?

The Discriminator is a neural network that identifies real data from the fake data created by the Generator. The discriminator's training data comes from different two sources:

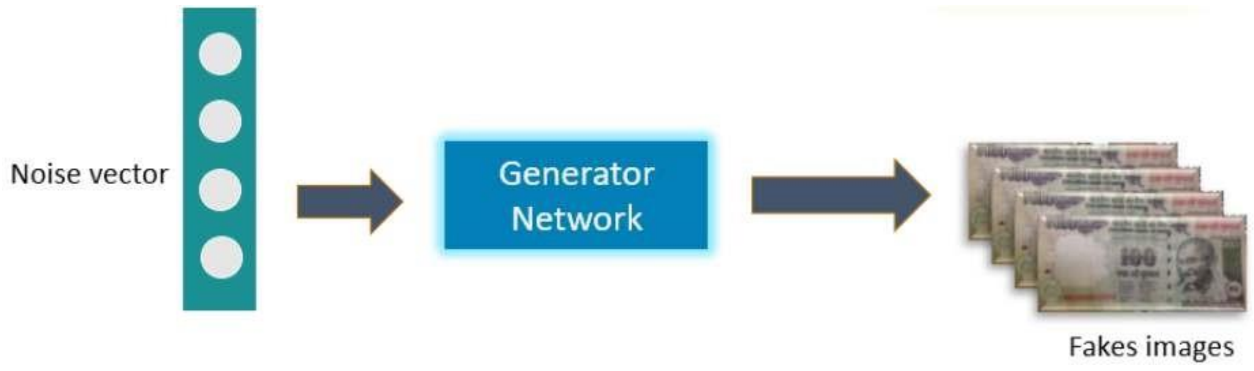
- The real data instances, such as real pictures of birds, humans, currency notes, etc., are used by the Discriminator as positive samples during training.
- The fake data instances created by the Generator are used as negative examples during the training process.



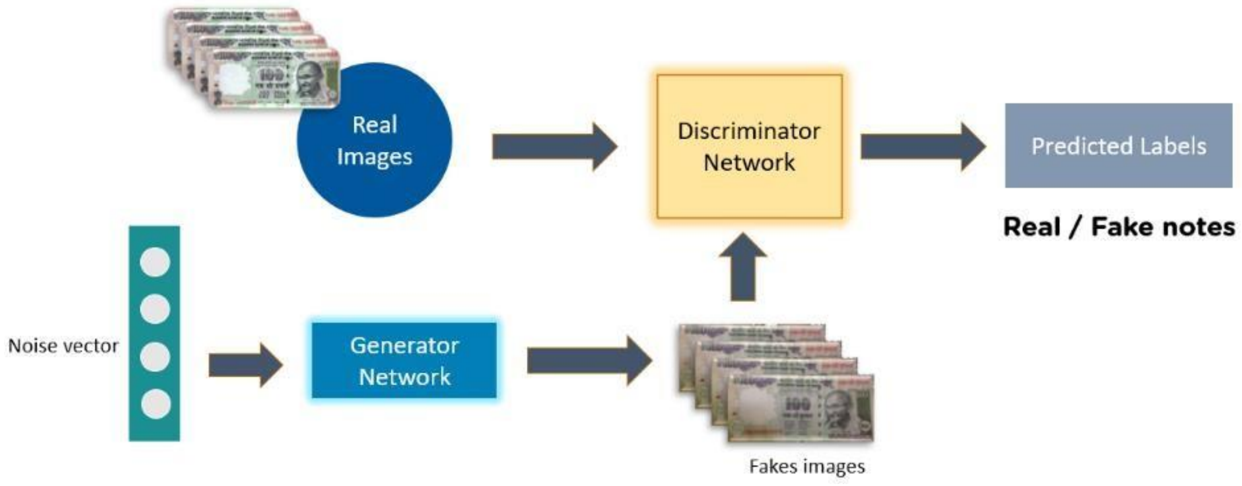
While training the discriminator, it connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss.
In the process of training the discriminator, the discriminator classifies both real data and fake data from the generator. The discriminator loss penalizes the discriminator for misclassifying a real data instance as fake or a fake data instance as real.
The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.



How Do GANs Work?
GANs consists of two neural networks. There is a Generator G(x) and a Discriminator D(x). Both of them play an adversarial game. The generator's aim is to fool the discriminator by producing data that are similar to those in the training set. The discriminator will try not to be fooled by identifying fake data from real data. Both of them work simultaneously to learn and train complex data like audio, video, or image files.
The Generator network takes a sample and generates a fake sample of data. The Generator is trained to increase the Discriminator network's probability of making mistakes.



Below is an example of a GAN trying to identify if the 100 rupee notes are real or fake. So, first, a noise vector or the input vector is fed to the Generator network. The generator creates fake 100 rupee notes. The real images of 100 rupee notes stored in a database are passed to the discriminator along with the fake notes. The Discriminator then identifies the notes as classifying them as real or fake.
We train the model, calculate the loss function at the end of the discriminator network, and backpropagate the loss into both discriminator and generator models.



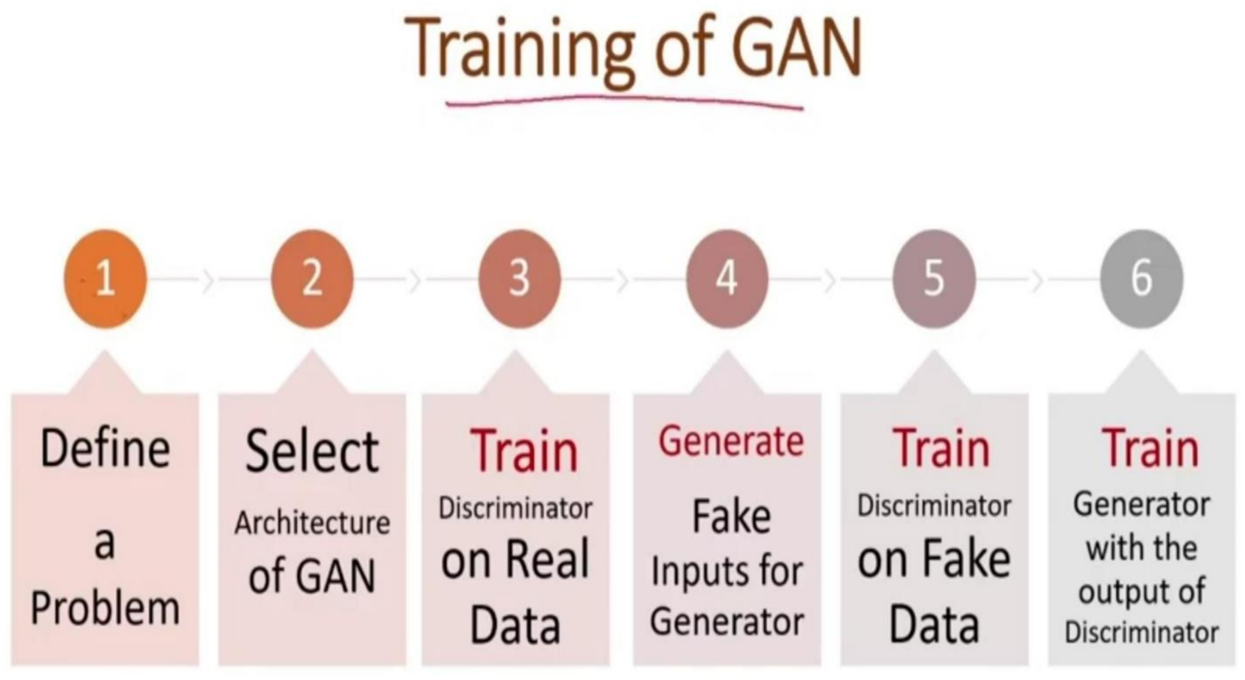
Mathematical Equation

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

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

The mathematical equation for training a GAN can be represented as:
Here,
G = Generator
D = Discriminator
Pdata(x) = distribution of real data
p(z) = distribution of generator
x = sample from Pdata(x)

z = sample from P(z)
D(x) = Discriminator network
G(z) = Generator network


Training of GAN
1
2
3
4
5
6
Define a Problem
Select Architecture of GAN
Train Discriminator on Real Data
Generate Fake Inputs for Generator
Train Discriminator on Fake Data
Train Generator with the output of Discriminator

Code:

Building the Generative Adversarial Network

Python3

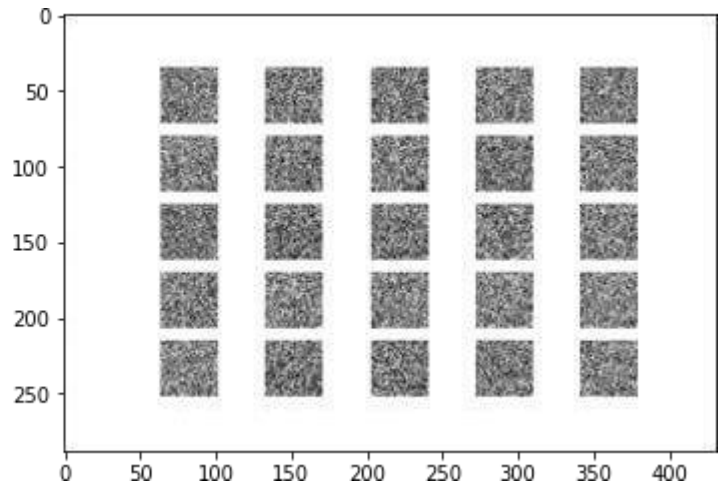
```
# Define the generator and discriminator
# Initialize generator and discriminator
generator = Generator(latent_dim).to(device)
discriminator = Discriminator().to(device)

# Loss function
adversarial_loss = nn.BCELoss()

# Optimizers
optimizer_G = optim.Adam(generator.parameters()\
                         , lr=lr, betas=(beta1, beta2))
optimizer_D = optim.Adam(discriminator.parameters()\
                         , lr=lr, betas=(beta1, beta2))
```

Check the results

Let's plot the generated images at different epochs to see that after how many epochs the generator was capable to extract some information.

Plot the generated Image at zero epoch

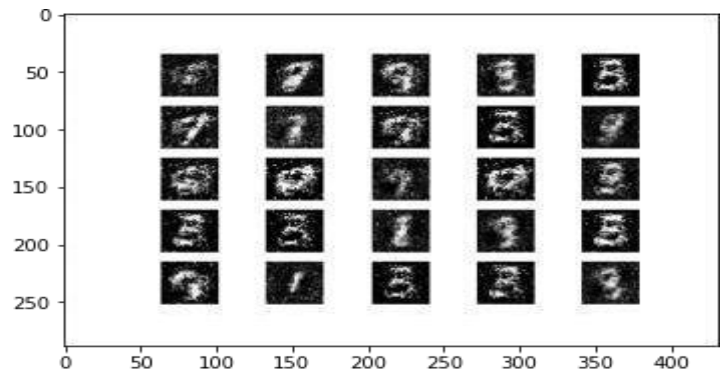
```
from skimage.io import imread
a = imread('gan_images/0.png')
plt.imshow(a)
```



No information is extracted from the generator and the discriminator is intelligent enough to identify it as fake.

Plot Image Generated after training on 1000 epoch

```
from skimage.io import imread
a = imread('gan_images/10000.png')
plt.imshow(a)
```



Now Generator is slowly being capable to extract some information that can be observed.

Plot Image Generated after training on 10000 Epochs

.