Vivekanand Education Society's Institute of Technology

Department of IT
AIDS-2
BE-SEM VII

AI : **Artificial Intelligence** (AI) works by simulating human intelligence through the use of algorithms, data, and computational power.

DS : Data science is the study of data to extract meaningful insights for business. It is a multidisciplinary approach that combines principles and practices from the fields of mathematics, statistics, artificial intelligence, and computer engineering to analyze large amounts of data.

Module 4
Introduction to Deep Learning

- In the fast-evolving era of artificial intelligence, Deep Learning stands as a cornerstone technology, revolutionizing how **machines understand, learn, and interact with complex data.**

- At its essence, Deep Learning AI mimics the intricate **neural networks of the human brain**, enabling computers to **autonomously** discover patterns and make decisions from vast amounts of unstructured data.

- This transformative field has propelled breakthroughs across various domains, from **computer vision and natural language processing to healthcare diagnostics and autonomous driving, and Reinforcement learning.**

- Deep Learning : branch of machine learning that is based on artificial neural network architecture.
- An artificial neural network or ANN uses layers of interconnected nodes called neurons that work together to process and learn from the input data.

- In a fully connected Deep neural network, there is an input layer and one or more hidden layers connected one after the other.
- Each neuron receives input from the previous layer neurons or the input layer.
- The output of one neuron becomes the input to other neurons in the next layer of the network, and this process continues until the final layer produces the output of the network.
- **The layers of the neural network transform the input data through a series of nonlinear transformations, allowing the network to learn complex representations of the input data.**

Deep learning AI can be used for supervised, unsupervised as well as reinforcement machine learning. it uses a variety of ways to process these.

•**Supervised Machine Learning:** Supervised machine learning is the machine learning technique in which the neural network learns to make predictions or classify data based on the labeled datasets. Here we input both input features along with the target variables. the neural network learns to make predictions based on the cost or error that comes from the difference between the predicted and the actual target, this process is known as backpropagation. Deep learning algorithms like Convolutional neural networks, Recurrent neural networks are used for many supervised tasks like image classifications and recognition, sentiment analysis, language translations, etc.

•**Unsupervised Machine Learning:** Unsupervised machine learning is the machine learning technique in which the neural network learns to discover the patterns or to cluster the dataset based on unlabeled datasets. Here there are no target variables. while the machine has to self-determined the hidden patterns or relationships within the datasets. Deep learning algorithms like autoencoders and generative models are used for unsupervised tasks like clustering, dimensionality reduction, and anomaly detection.

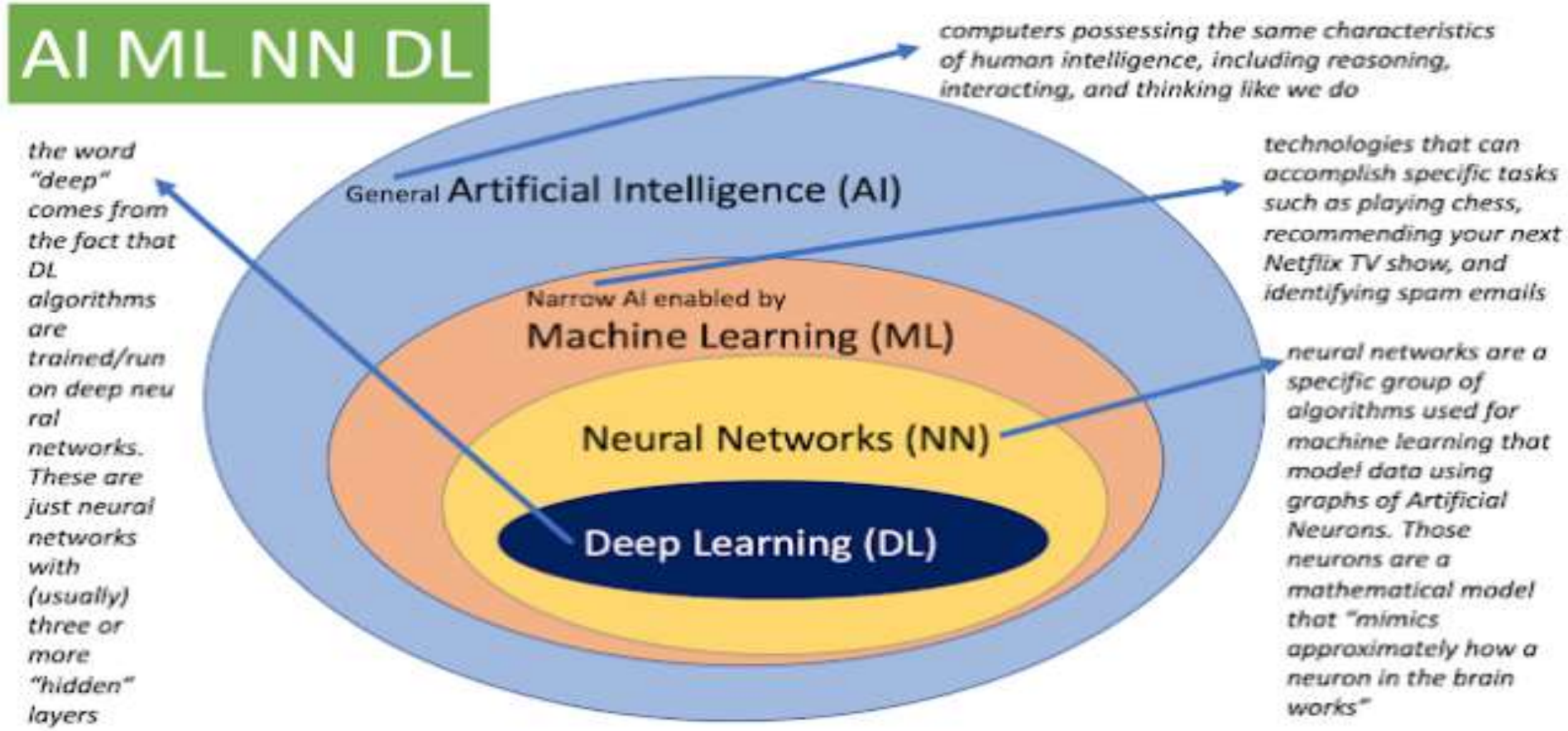
•**Reinforcement Machine Learning**: Reinforcement Machine Learning is the machine learning technique in which an agent learns to make decisions in an environment to maximize a reward signal. The agent interacts with the environment by taking action and observing the resulting rewards. Deep learning can be used to learn policies, or a set of actions, that maximizes the cumulative reward over time. Deep reinforcement learning algorithms like Deep Q networks and Deep Deterministic Policy Gradient (DDPG) are used to reinforce tasks like robotics and game playing etc.

Difference between Machine Learning and Deep Learning :

machine learning and deep learning AI both are subsets of artificial intelligence but there are many similarities and differences between them.

Machine Learning	Deep Learning
Apply statistical algorithms to learn the hidden patterns and relationships in the dataset.	Uses artificial neural network architecture to learn the hidden patterns and relationships in the dataset.
Can work on the smaller amount of dataset	Requires the larger volume of dataset compared to machine learning
Better for the low-label task.	Better for complex task like image processing, natural language processing, etc.
Takes less time to train the model.	Takes more time to train the model.
A model is created by relevant features which are manually extracted from images to detect an object in the image.	Relevant features are automatically extracted from images. It is an end-to-end learning process.
Less complex and easy to interpret the result.	More complex, it works like the black box interpretations of the result are not easy.
It can work on the CPU or requires less computing power as compared to deep learning.	It requires a high-performance computer with GPU.

Relation between AI, ML and DL



Challenges in Deep Learning

Deep learning has made significant advancements in various fields, but there are still some challenges that need to be addressed. Here are some of the main challenges in deep learning:

1.Data availability: It requires large amounts of data to learn from. For using deep learning it's a big concern to gather as much data for training.

2.Computational Resources: For training the deep learning model, it is computationally expensive because it requires specialized hardware like GPUs(Graphics Processing Unit) and TPUs(Tensor Processing Unit).

3.Time-consuming: While working on sequential data depending on the computational resource it can take very large even in days or months.

4.I**nterpretability:** Deep learning models are complex, it works like a black box. it is very difficult to interpret the result.

5.Overfitting: when the model is trained again and again, it becomes too specialized for the training data, leading to overfitting and poor performance on new data.

Advantages of Deep Learning:

1.High accuracy: Deep Learning algorithms can achieve state-of-the-art performance in various tasks, such as image recognition and natural language processing.

2.Automated feature engineering: Deep Learning algorithms can automatically discover and learn relevant features from data without the need for manual feature engineering.

3.Scalability: Deep Learning models can scale to handle large and complex datasets, and can learn from massive amounts of data.

4.Flexibility: Deep Learning models can be applied to a wide range of tasks and can handle various types of data, such as images, text, and speech.

5.Continual improvement: Deep Learning models can continually improve their performance as more data becomes available.

Disadvantages of Deep Learning:

1.High computational requirements: Deep Learning AI models require large amounts of data and computational resources to train and optimize.

2.Requires large amounts of labeled data: Deep Learning models often require a large amount of labeled data for training, which can be expensive and time- consuming to acquire.

3.Interpretability: Deep Learning models can be challenging to interpret, making it difficult to understand how they make decisions.

Overfitting: Deep Learning models can sometimes overfit to the training data, resulting in poor performance on new and unseen data.

4.Black-box nature: Deep Learning models are often treated as black boxes, making it difficult to understand how they work and how they arrived at their predictions.

Deep Learning Applications:

The main applications of deep learning AI can be divided into computer vision, natural language processing (NLP), and reinforcement learning.

1. Computer vision

The first Deep Learning applications is Computer vision. In computer vision, Deep learning AI models can enable machines to identify and understand visual data. Some of the main applications of deep learning in computer vision include:

- **Object detection and recognition:** Deep learning model can be used to identify and locate objects within images and videos, making it possible for machines to perform tasks such as self-driving cars, surveillance, and robotics.
- **Image classification:** Deep learning models can be used to classify images into categories such as animals, plants, and buildings. This is used in applications such as medical imaging, quality control, and image retrieval.
- **Image segmentation:** Deep learning models can be used for image segmentation into different regions, making it possible to identify specific features within images.

2. Natural language processing (NLP)**:**

In Deep learning applications, second application is NLP. NLP, the Deep learning model can enable machines to understand and generate human language. Some of the main applications of deep learning in NLP include:

•**Automatic Text Generation** – Deep learning model can learn the corpus of text and new text like summaries, essays can be automatically generated using these trained models.

•**Language translation:** Deep learning models can translate text from one language to another, making it possible to communicate with people from different linguistic backgrounds.

•**Sentiment analysis:** Deep learning models can analyze the sentiment of a piece of text, making it possible to determine whether the text is positive, negative, or neutral. This is used in applications such as customer service, social media monitoring, and political analysis.

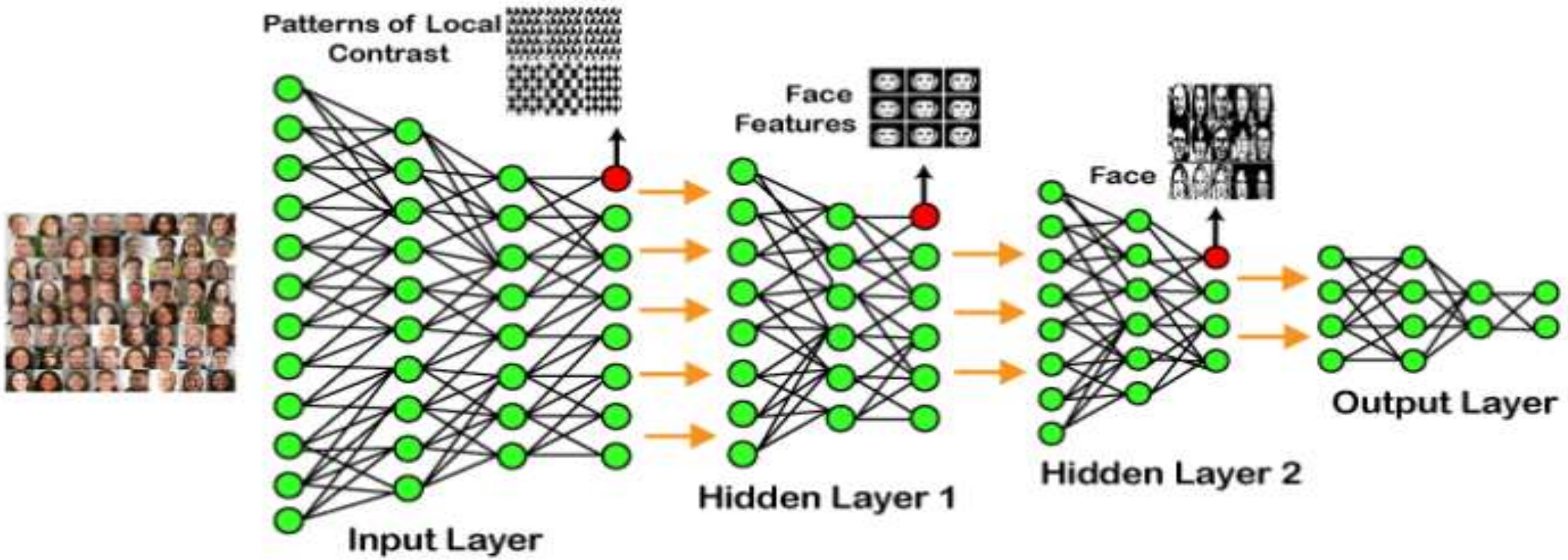
•**Speech recognition:** Deep learning models can recognize and transcribe spoken words, making it possible to perform tasks such as speech-to-text conversion, voice search, and voice-controlled devices.

3. Reinforcement learning:

In reinforcement learning, deep learning works as training agents to take action in an environment to maximize a reward. Some of the main applications of deep learning in reinforcement learning include:

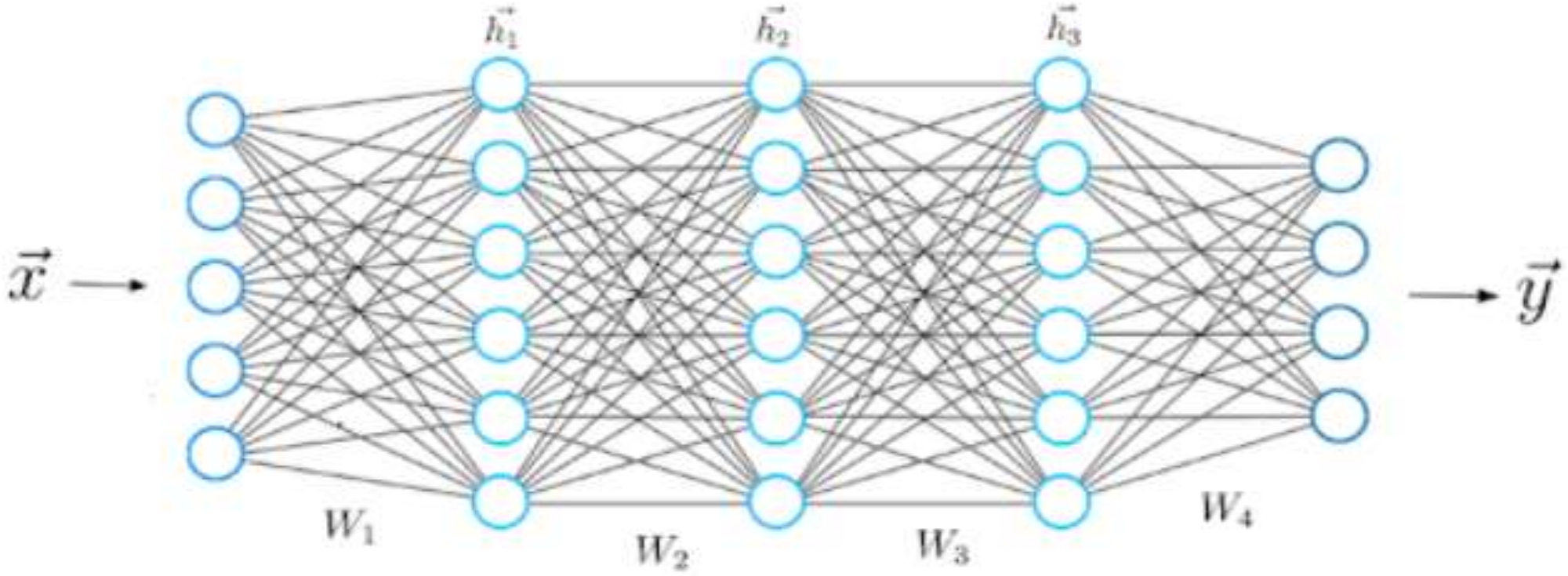
- **Game playing:** Deep reinforcement learning models have been able to beat human experts at games such as Go, Chess, and Atari.
- **Robotics:** Deep reinforcement learning models can be used to train robots to perform complex tasks such as grasping objects, navigation, and manipulation.
- **Control systems:** Deep reinforcement learning models can be used to control complex systems such as power grids, traffic management, and supply chain optimization.

Architecture Of Deep Learning….



How Does Deep Learning Work?

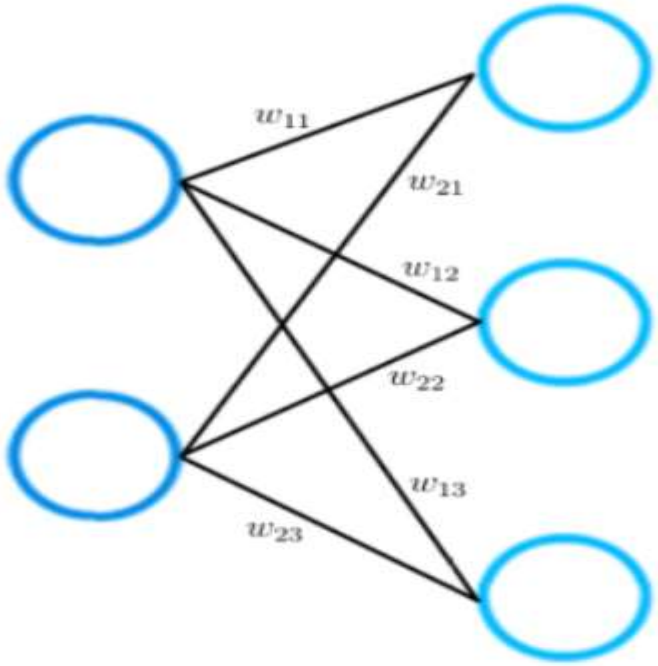
Deep learning algorithms attempt to draw similar conclusions as humans would by constantly analyzing data with a given logical structure. To achieve this, deep learning uses a multi-layered structure of algorithms called neural networks.



A typical neural network

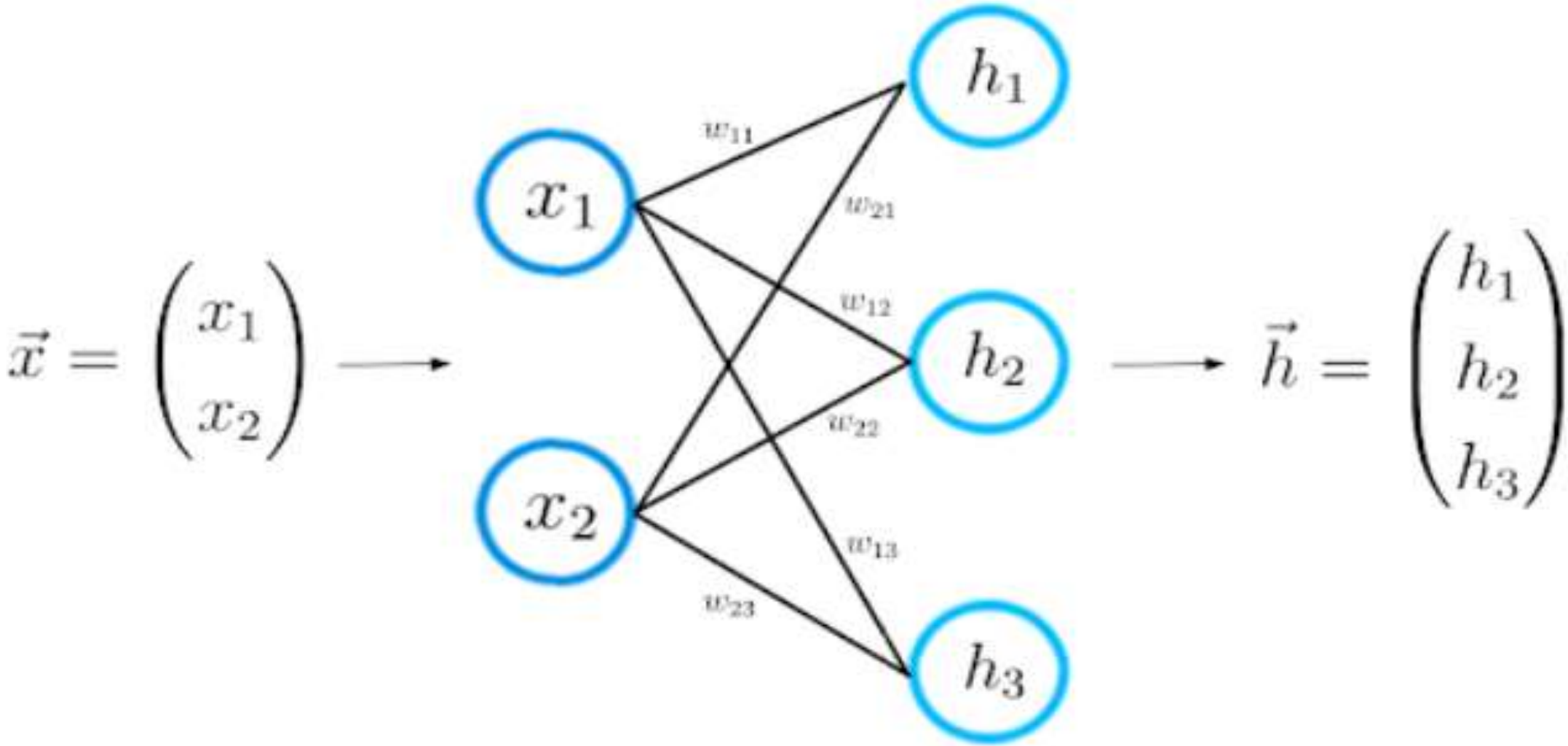
- The design of the neural network is based on the structure of the human brain. Just as **we use our brains to identify patterns and classify different types of information**, we can teach neural networks to perform the same tasks on data.
- The **individual layers of neural networks can also be thought of as a sort of filter** that works from gross to subtle, **which increases the likelihood of detecting and outputting a correct result.** The human brain works similarly. Whenever we receive new information, the brain tries to compare it with known objects. The same concept is also used by deep neural networks.
- Neural networks enable us to perform many tasks, such as clustering, classification or regression(to determine relationship between a dependent variable and one or more independent variables).
- With neural networks, we can group or sort unlabeled data according to similarities among samples in the data. Or, in the case of classification, we can train the network on a labeled data set in order to classify the samples in the data set into different categories.
- In other words, artificial neural networks have unique capabilities that enable deep learning models to solve tasks that machine learning models can never solve.
- All recent advances in artificial intelligence in recent years are due to deep learning.
- self-driving cars, chatbots or personal assistants like Alexa and Siri. Google Translate would continue to be as primitive as it was before Google switched to neural networks and Netflix would have no idea which movies to suggest. Neural networks are behind all of these deep learning applications and technologies.

- Please consider a smaller neural network that consists of only two layers. The input layer has two input neurons, while the output layer consists of three neurons.
- As mentioned earlier, each connection between two neurons is represented by a numerical value, which we call weight.
- The number of rows corresponds to the number of neurons in the layer from which the connections originate and the number of columns corresponds to the number of neurons in the layer to which the connections lead.
- In this particular example, the number of rows of the weight matrix corresponds to the size of the input layer, which is two, and the number of columns to the size of the output layer, which is three.



$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

A weight matrix



Forward propagation

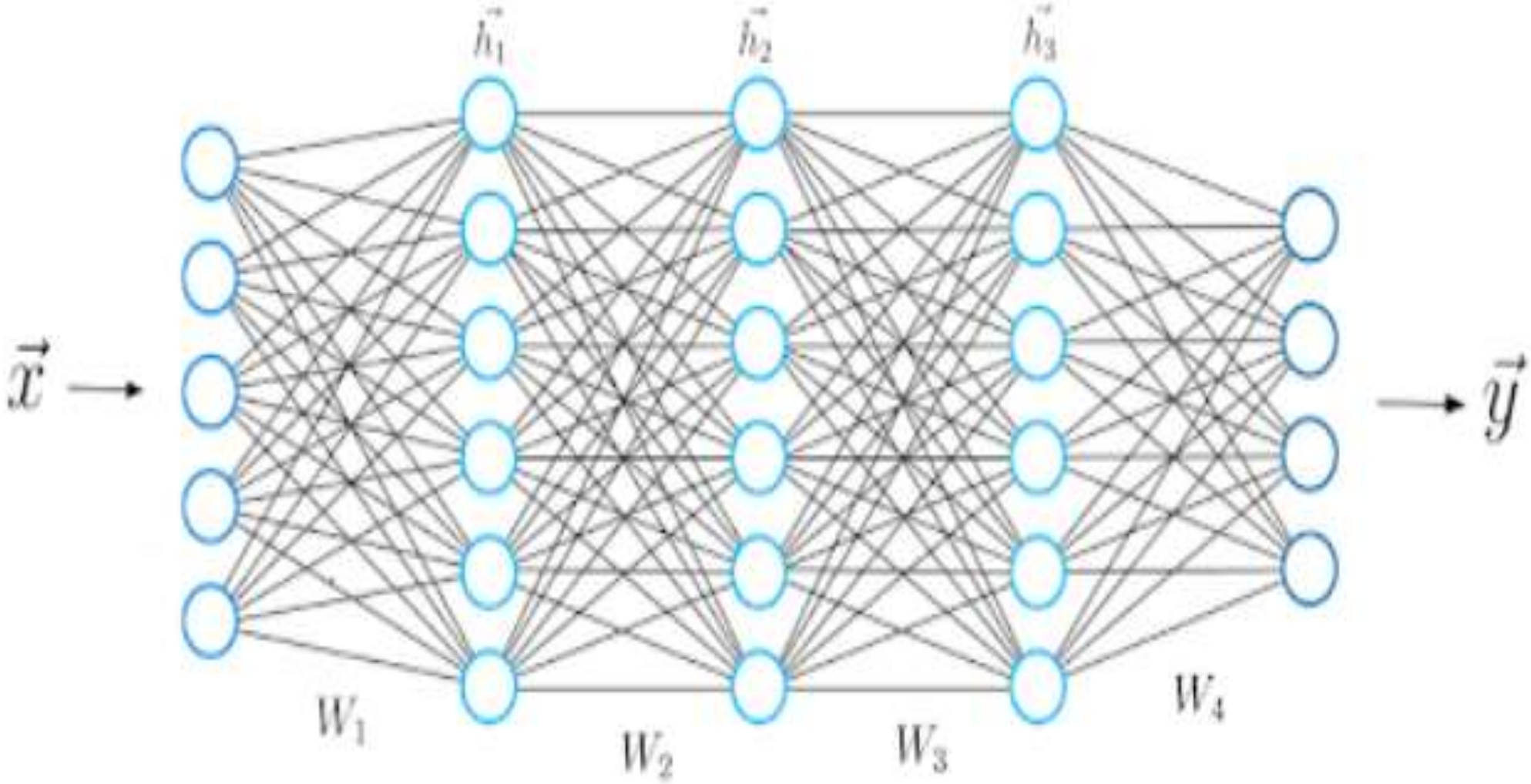
$$\vec{x}^T \cdot W = \begin{pmatrix} x_1, & x_2 \end{pmatrix} \cdot \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

$$= \begin{pmatrix} x_1 w_{11} + x_2 w_{21}, & x_1 w_{12} + x_2 w_{22}, & x_1 w_{13} + x_2 w_{23} \end{pmatrix}$$

$$= \begin{pmatrix} z_1, z_2, z_3 \end{pmatrix} = \vec{z}, \quad \vec{h} = \sigma(\vec{z})$$

Equations for forward propagation

we can extend our knowledge to a deeper architecture that consists of five layers.



As before, we calculate the dot product between the input `x` and the first weight matrix `W1`, and apply an activation function to the resulting vector to obtain the first hidden vector `h1`. We now consider `h1` the input for the upcoming third layer. We repeat the whole procedure from before until we obtain the final output `y`:

$$\vec{h_1} = \sigma(\vec{x}^T \cdot W_1)$$
$$\vec{h_2} = \sigma(\vec{h_1} \cdot W_2)$$
$$\vec{h_3} = \sigma(\vec{h_2} \cdot W_3)$$
$$\vec{y} = \sigma(\vec{h_3} \cdot W_4)$$

Equations for forward propagation

Loss Functions in Deep Learning

After we get the prediction of the neural network, we must compare this prediction vector to the actual ground truth label. We call the ground truth label vector `y_hat`.

While the vector `y` contains predictions that the neural network has computed during the forward propagation (which may, in fact, be very different from the actual values), the vector `y_hat` contains the actual values.

Mathematically, we can measure the difference between `y` and `y_hat` by defining a loss function, whose value depends on this difference.

An example of a general loss function is the quadratic loss:

$$\mathcal{L}(\theta) = \frac{1}{2}(\vec{y} - \hat{\vec{y}})^2$$

Quadratic loss

- A higher difference means a higher loss value and a smaller difference means a smaller loss value.
- Minimizing the loss function directly leads to more accurate predictions of the neural network, as the difference between the prediction and the label decreases.
- Minimizing the loss function automatically causes the neural network model to make better predictions regardless of the exact characteristics of the task at hand. You only have to select the right loss function for the task.
- Fortunately, there are only two loss functions that you should know about to solve almost any problem that you encounter in practice: the cross-entropy loss and the mean squared error (MSE) loss.

The Cross-Entropy Loss

$$\mathcal{L}(\theta) = -\sum_{i=0}^{N} \hat{y}_i \cdot log(y_i)$$

y_i : entries in the prediction vector $\vec{y}$

$\hat{y}_i$: entries in the ground truth label $\hat{\vec{y}}$

Cross-entropy loss function

Mean Squared Error Loss

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=0}^{N} (y_i - \hat{y}_i)^2$$

y_i : entries in the prediction vector $\vec{y}$

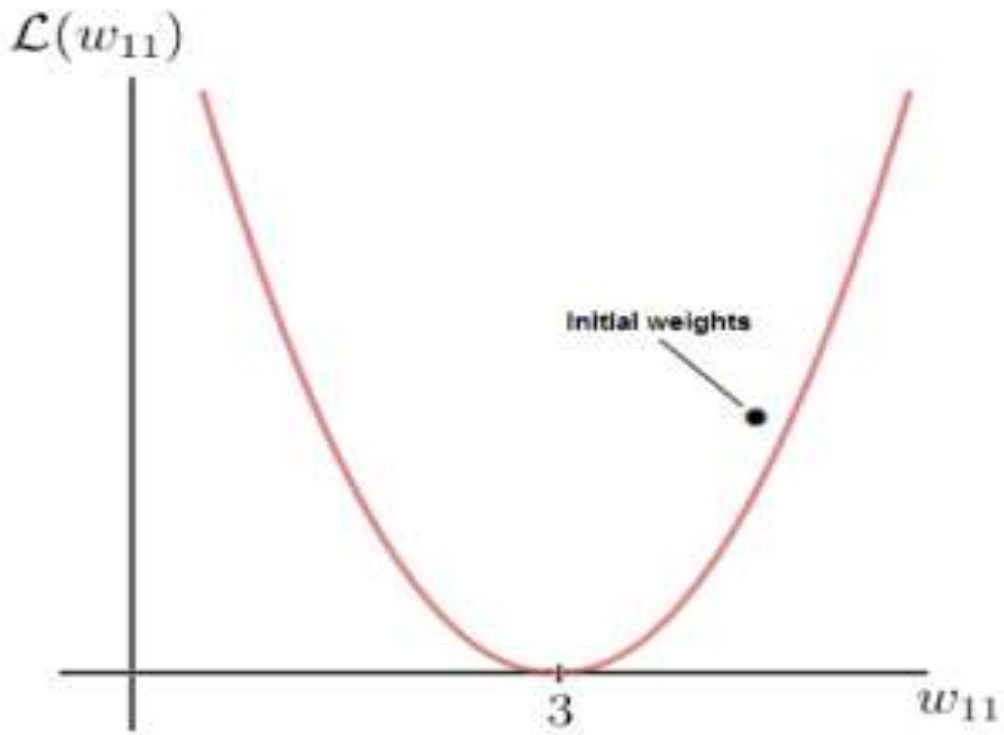
$\hat{y}_i$: entries in the ground truth label $\hat{\vec{y}}$

Mean squared error loss function

Since the loss depends on the weight, we must find a certain set of weights for which the value of the loss function is as small as possible. The method of minimizing the loss function is achieved mathematically by a method called gradient descent.

- **3. Mean Absolute Percentage Error**
- Finally, we come to the Mean Absolute Percentage Error (MAPE) loss function. This loss function doesn’t get much attention in deep learning. For the most part, we use it to measure the performance of a neural network during demand forecasting tasks.

As you can see, there is a certain weight `w` for which the loss function reaches a global minimum. This value is the optimal weight parameter that would cause the neural network to make the correct prediction (which is `6`). In this case, the value for the optimal weight is `3`:



Initial weight value

On the other hand, our initial weight is `5`, which leads to a fairly high loss. The goal now is to repeatedly update the weight parameter until we reach the optimal value for that particular weight. This is the time when we need to use the gradient of the loss function.

On the other hand, our initial weight is `5`, which leads to a fairly high loss. The goal now is to repeatedly update the weight parameter until we reach the optimal value for that particular weight. This is the time when we need to use the gradient of the loss function.

Fortunately, in this case, the loss function is a function of one single variable, the weight `w`:

$$\mathcal{L}(w_{11}) = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(6 - w_{11} \cdot x_1)^2$$

Loss function

In the next step, we calculate the derivative of the loss function with respect to this parameter:

$$\nabla_{w_{11}} \mathcal{L}(w_{11}) = \frac{\partial \mathcal{L}(w_{11})}{\partial w_{11}} = -x_1(6 - w_{11} \cdot x_1) = 8$$

Gradient of the loss function

In the end, we get `8`, which gives us the value of the slope or the tangent of the loss function for the corresponding point on the x-axis, at which point our initial weight lies.

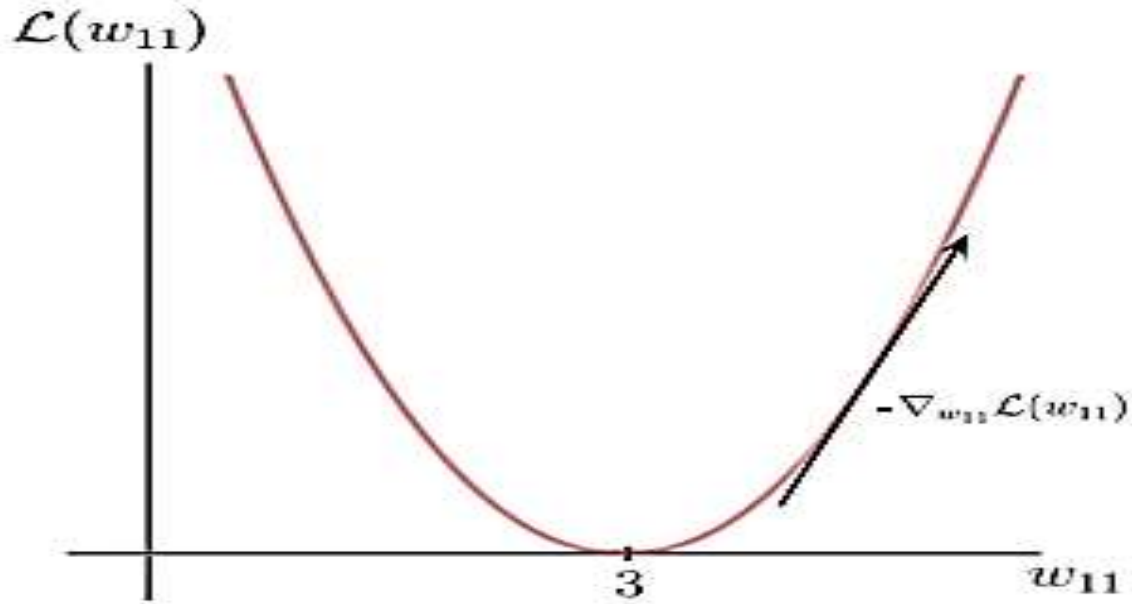
This tangent points toward the highest rate of increase of the loss function and the corresponding weight parameters on the x-axis.

$$\nabla_{w_{11}} \mathcal{L}(w_{11}) = \frac{\partial \mathcal{L}(w_{11})}{\partial w_{11}}$$

Gradient of the loss function

This means that we have just used the gradient of the loss function to find out which weight parameters would result in an even higher loss value. What we really want to know is the exact opposite. We can get what we want if we multiply the gradient by `-1` and, in this way, obtain the opposite direction of the gradient.

This is how we get the direction of the loss function's highest rate of decrease and the corresponding parameters on the x-axis that cause this decrease:



Finally, we perform one gradient descent step as an attempt to improve our weights. We use this negative gradient to update your current weight in the direction of the weights for which the value of the loss function decreases, according to the negative gradient:

$$w_{11_{new}} = w_{11_{old}} - \epsilon \cdot \nabla_{w_{11}} \mathcal{L}(w_{11})$$

Gradient descent step

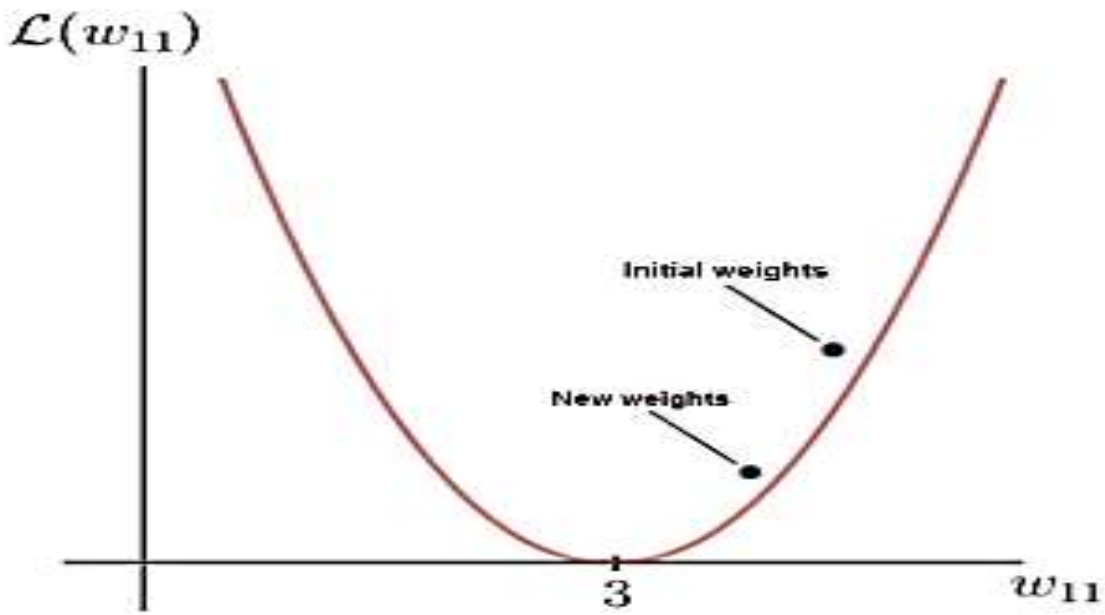
$$w_{11_{new}} = w_{11_{old}} - 0.1 \cdot 8 = 4.2$$

Gradient descent step

The factor `epsilon` in this equation is a hyper-parameter called the learning rate. The learning rate determines how quickly or how slowly you want to update the parameters.

Please keep in mind that the learning rate is the factor with which we have to multiply the negative gradient and that the learning rate is usually quite small. In our case, the learning rate is `0.1`.

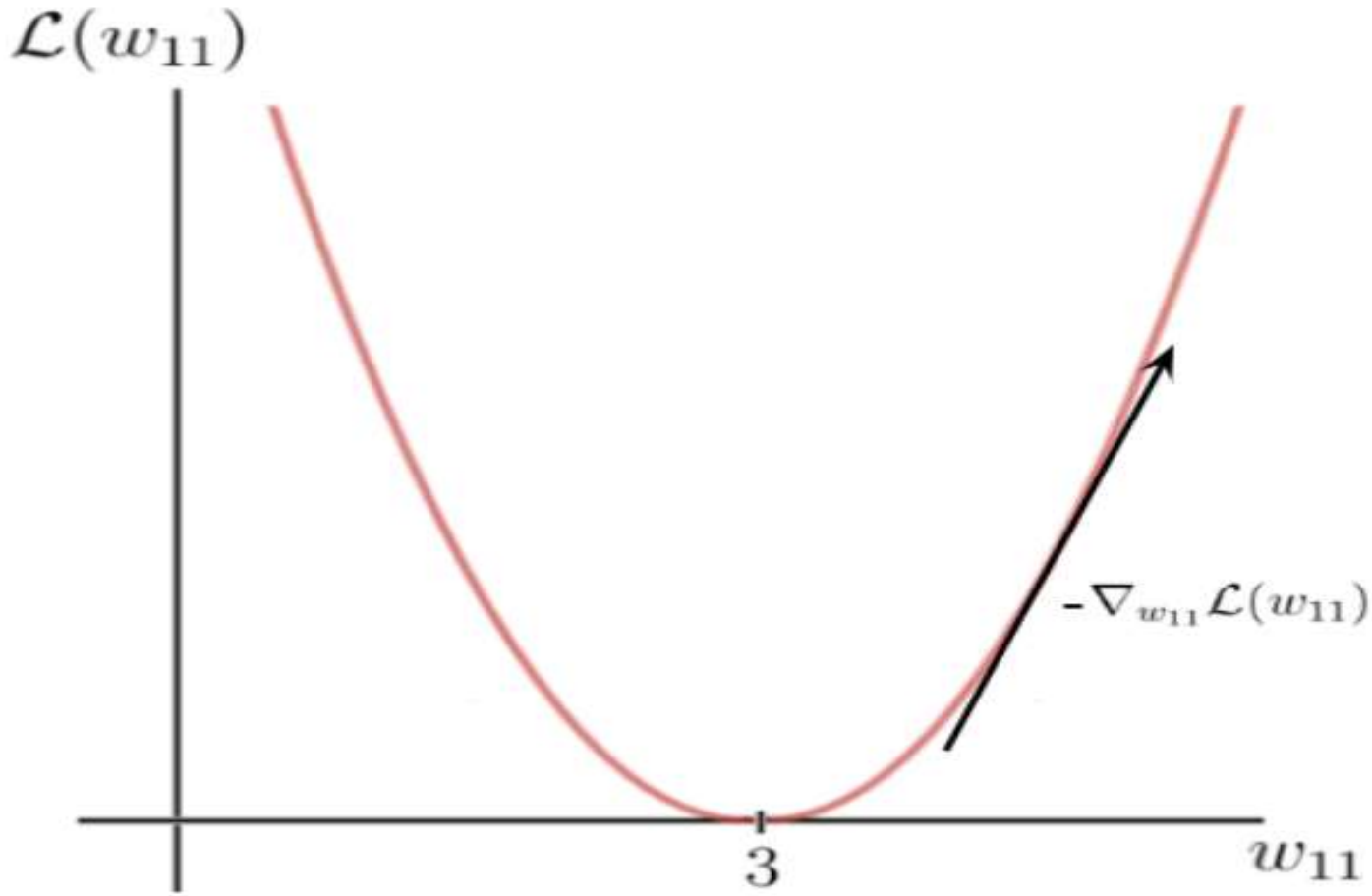
As you can see, our weight `w` after the gradient descent is now `4.2` and closer to the optimal weight than it was before the gradient step.



New weights after gradient descent

The value of the loss function for the new weight value is also smaller, which means that the neural network is now capable of making better predictions. You can do the calculation in your head and see that the new prediction is, in fact, closer to the label than before.

Each time we update the weights, we move down the negative gradient towards the optimal weights.



Finally, we perform one gradient descent step as an attempt to improve our weights. We use this negative gradient to update your current weight in the direction of the weights for which the value of the loss function decreases, according to the negative gradient:

$$w_{11_{new}} = w_{11_{old}} - \epsilon \cdot \nabla_{w_{11}} \mathcal{L}(w_{11})$$

Gradient descent step

$$w_{11_{new}} = w_{11_{old}} - 0.1 \cdot 8 = 4.2$$

Gradient descent step

The factor `epsilon` in this equation is a hyper-parameter called the learning rate. The learning rate determines how quickly or how slowly you want to update the parameters.

Please keep in mind that the learning rate is the factor with which we have to multiply the negative gradient and that the learning rate is usually quite small. In our case, the learning rate is `0.1`.

As you can see, our weight `w` after the gradient descent is now `4.2` and closer to the optimal weight than it was before the gradient step.

Types of Deep Learning Networks

1> Feed forward neural network(1)

2> Recurrent neural network (RNN)

3> Convolution neural network (CNN)

4> Restricted Boltzman Machine

5 > Autoencoders

Types of neural networks

Deep Learning models are able to automatically learn features from the data, which makes them well-suited for tasks such as image recognition, speech recognition, and natural language processing. The most widely used architectures in deep learning are feedforward neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs).

1.Feedforward neural networks (FNNs) are the simplest type of ANN, with a linear flow of information through the network. FNNs have been widely used for tasks such as image classification, speech recognition, and natural language processing.

2.Convolutional Neural Networks (CNNs) are specifically for image and video recognition tasks. CNNs are able to automatically learn features from the images, which makes them well-suited for tasks such as image classification, object detection, and image segmentation.

3.Recurrent Neural Networks (RNNs) are a type of neural network that is able to process sequential data, such as time series and natural language. RNNs are able to maintain an internal state that captures information about the previous inputs, which makes them well-suited for tasks such as speech recognition, natural language processing, and language translation.

Feed forward neural network

1> nodes do not form a cycle

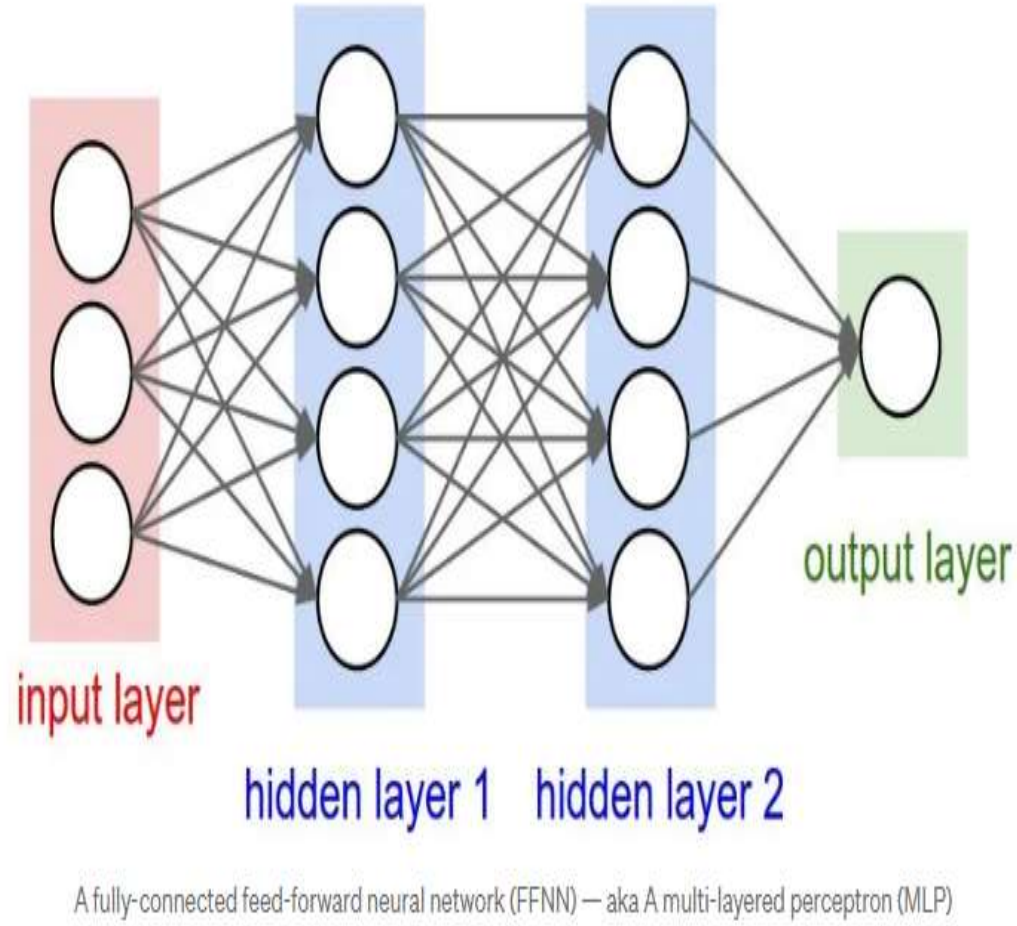
2> no back-loops in the feed-forward network.

3> the hidden lavers do not link with the outside world

4> all of the nodes are fully connected.

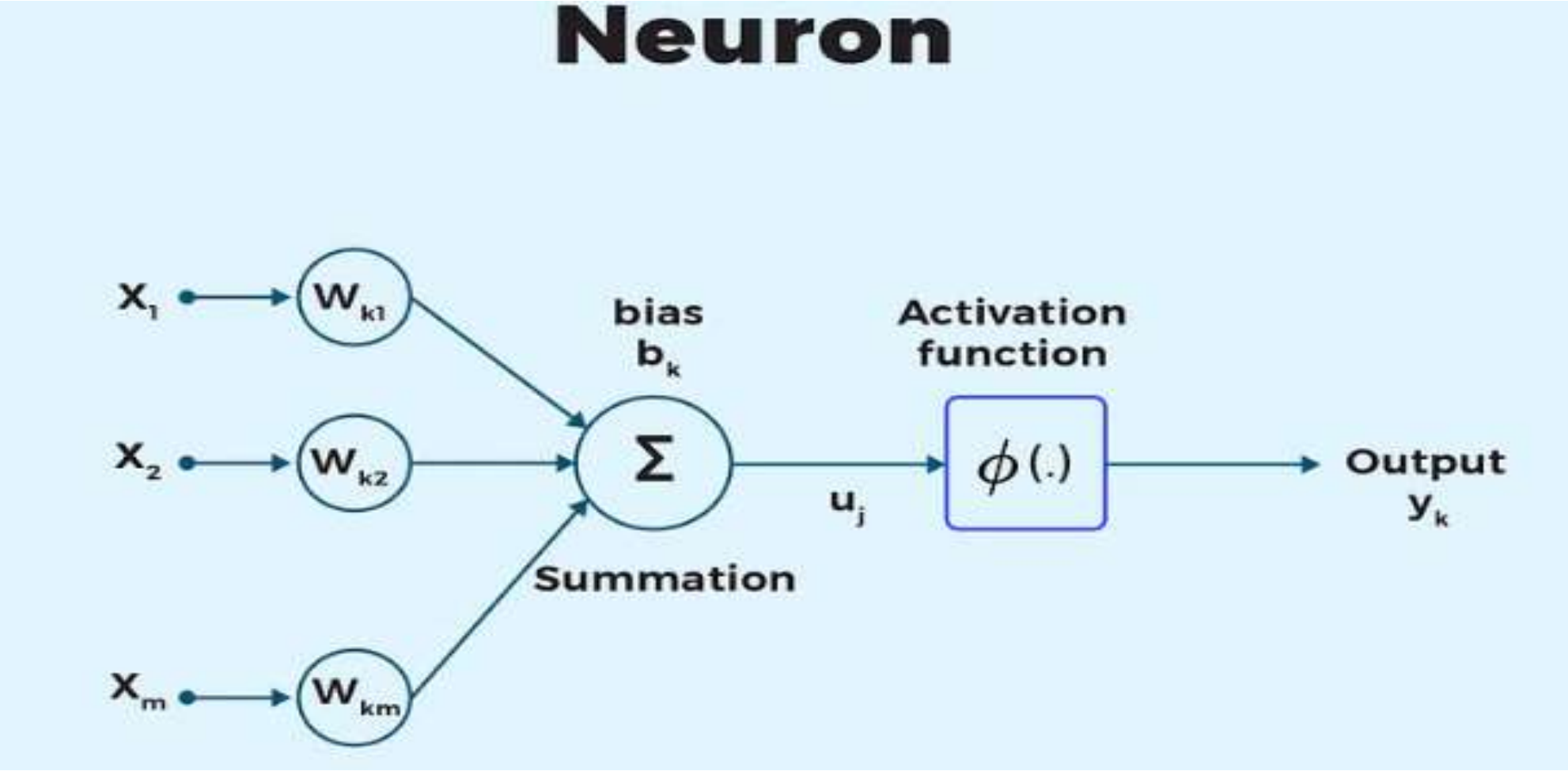
https://medium.com/@b.terryjack/introduction-to-deep-learning-feed-forward-neural-networks-ffnns-a-k-a-c688d83a309d

Architecture



Key Components of Deep Learning Architectures

The simplest form of a neural network: perceptron. A perceptron makes binary decisions – a simple "yes" or "no" based on the weighted sum of input features.



Bias

(1) The bias is included in the network.

(2) It has impact in calculating the net input.

(3) The input vector becomes, $X = \{1, X_1, X_2, \ldots, X_n\}$

(4) The bias is another weight, say $w = b_1$.

(5) Bias plays a major role in determining the output of the network.

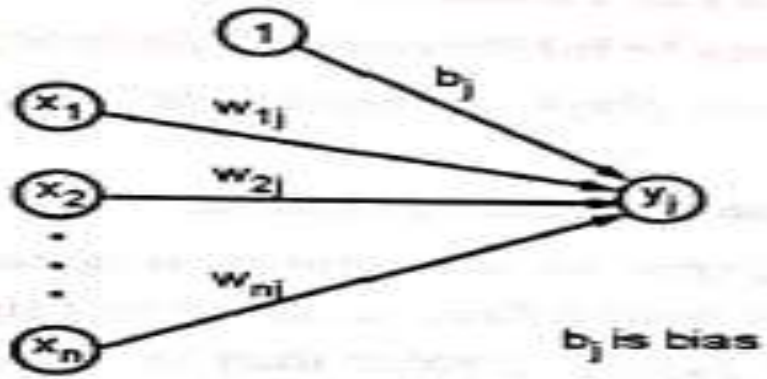


Fig. Net with bias

(6) The bias is of two types :

(a) Positive bias increases the net input of the network and

(b) The negative bias decreases the net input of the network.

(7) Due to bias-effect, the output of the network can be varied.

McCulloch Pitts Neuron
(assuming no inhibitory inputs)

$$y = 1 \quad if \sum_{i=0}^{n} x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} x_i < 0$$

Perceptron

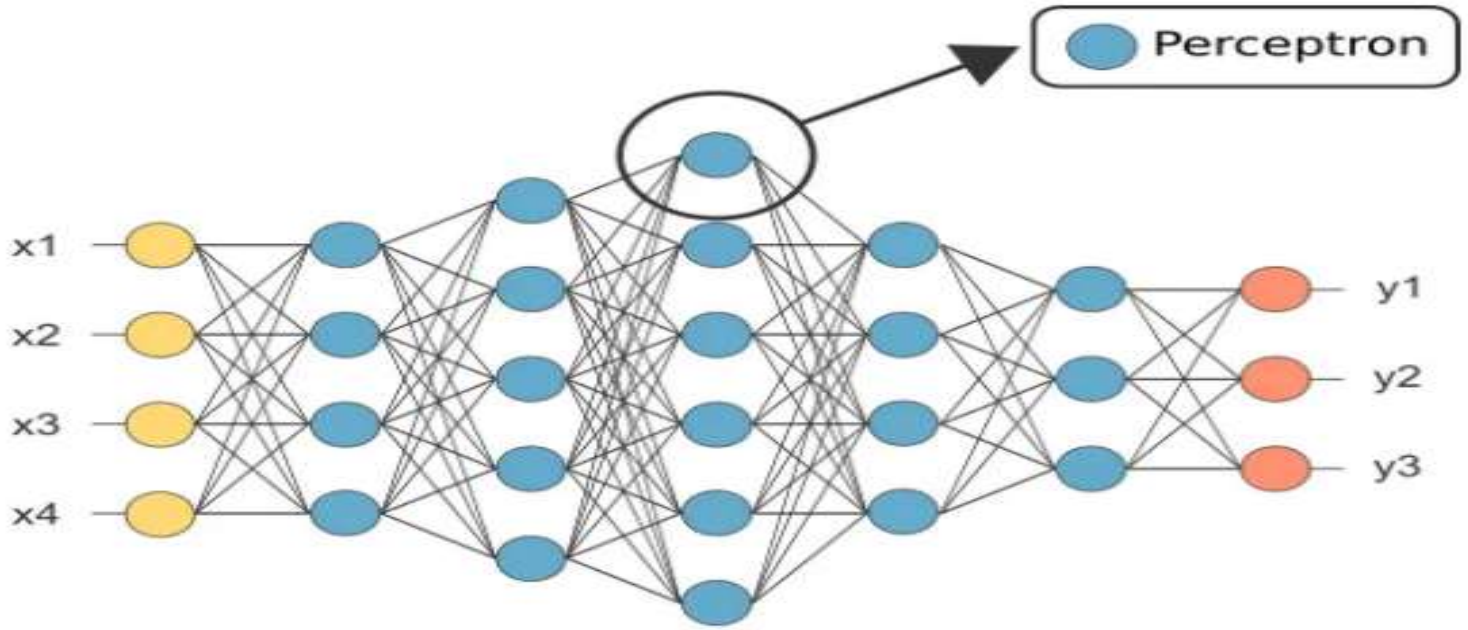
$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

The difference between the perceptron and neuron is that the perceptron is a type of artificial neural network that is used to classify patterns, while a neuron is a cell in the brain that processes and transmits information.

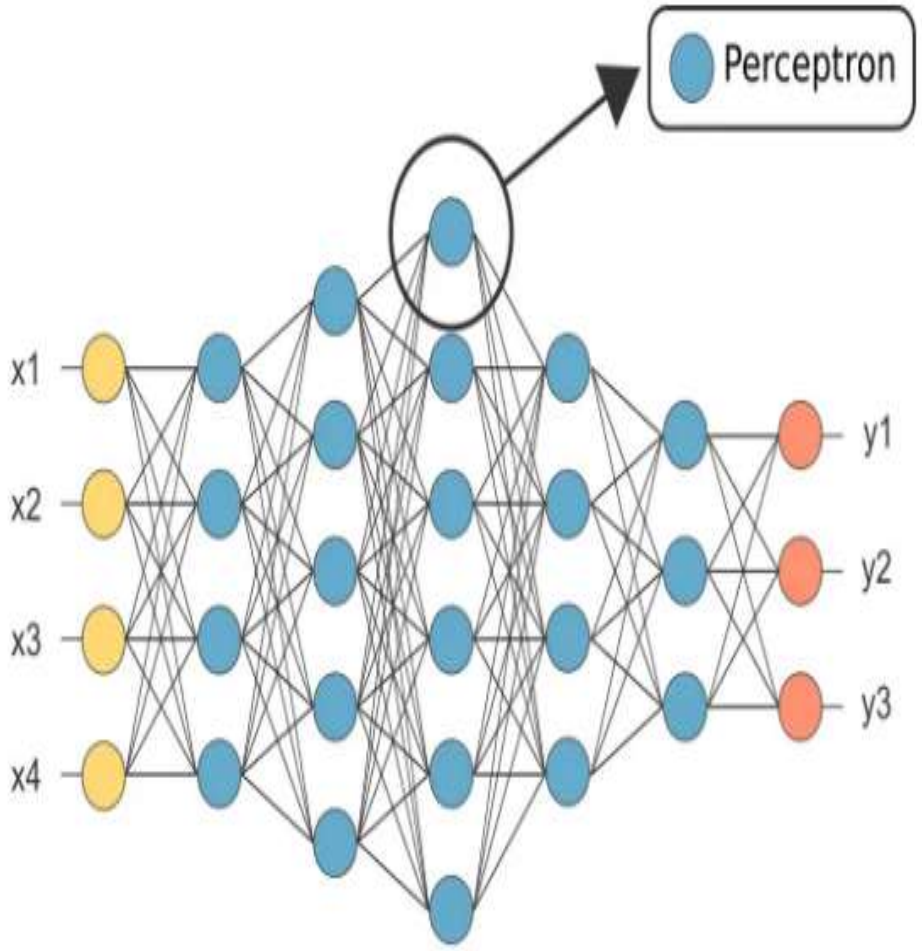
Definitions :

1> Perceptron : A perceptron takes several inputs, processes them using weights and a bias, and produces an output.

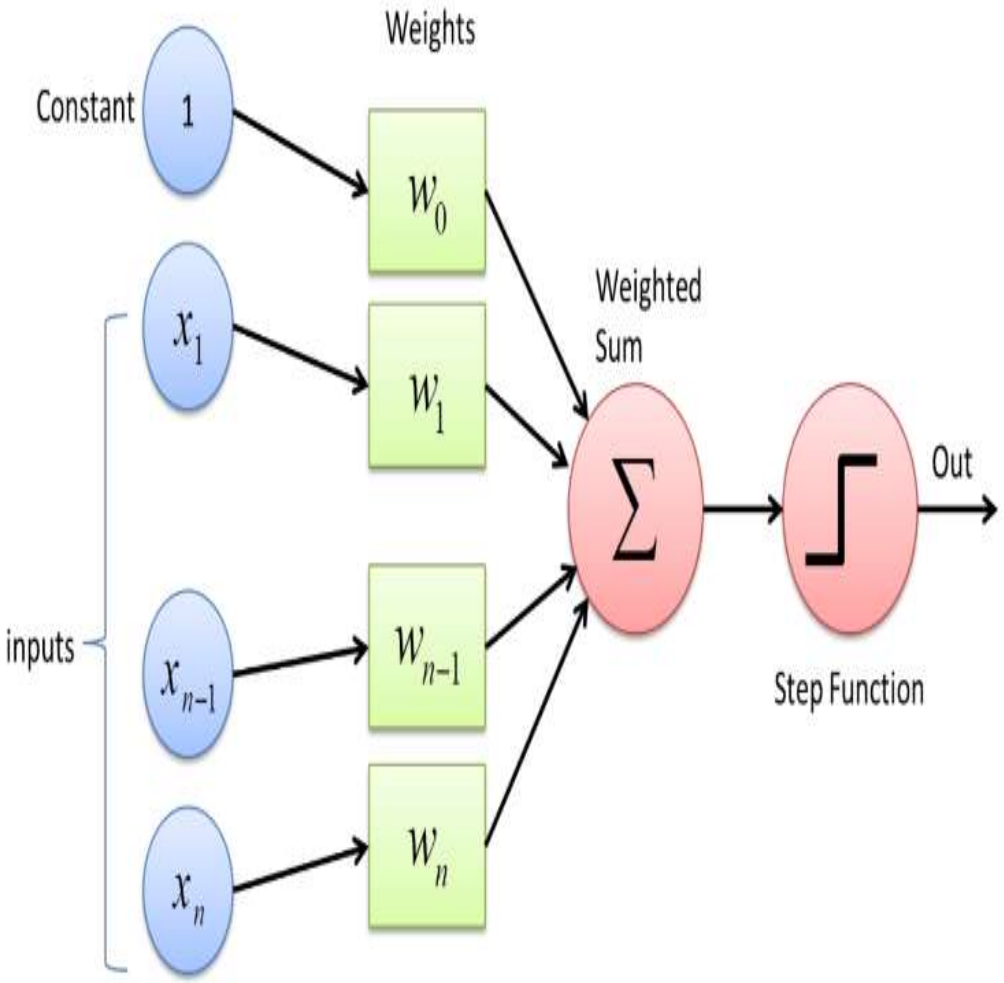


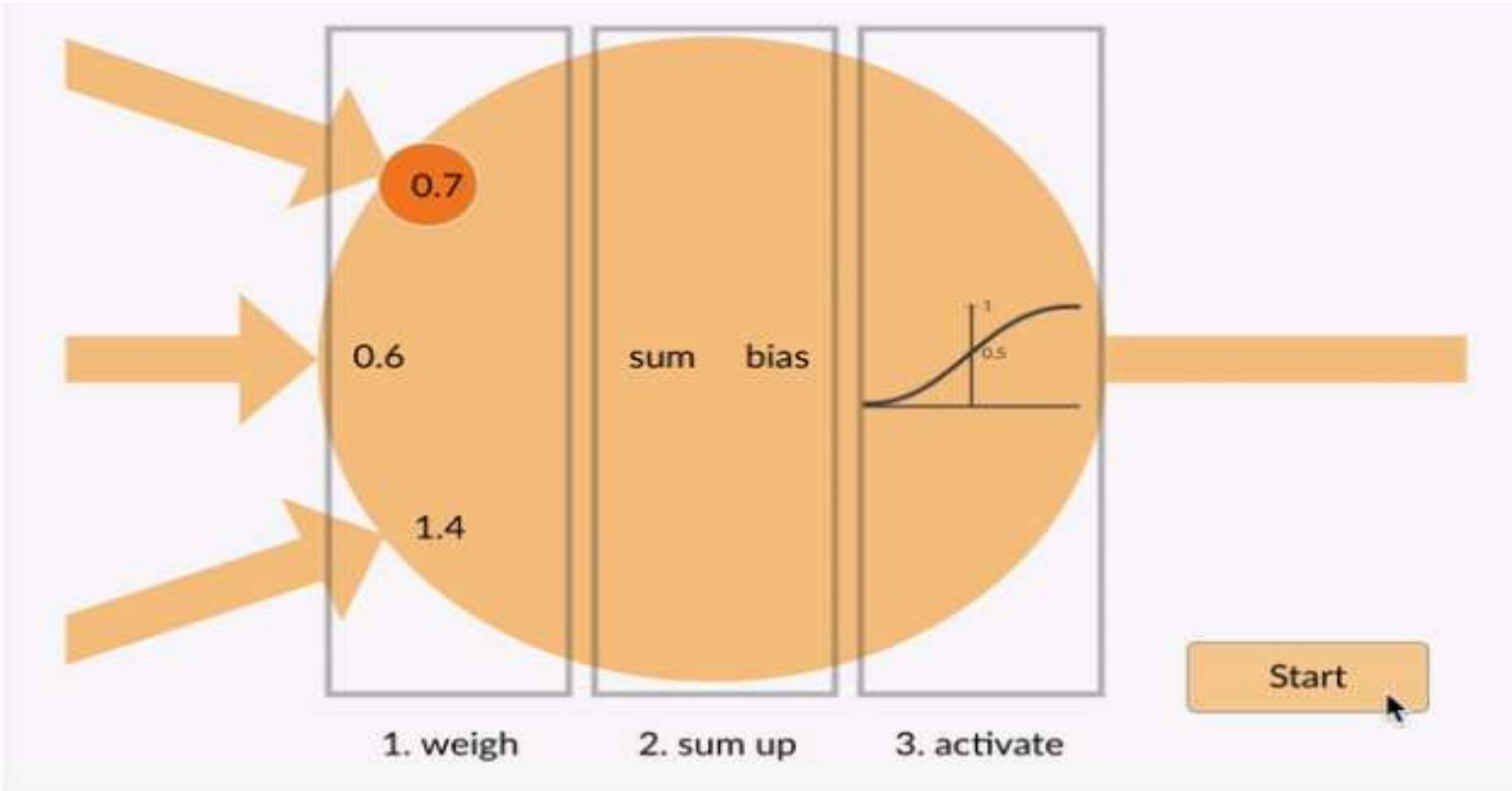
The Perceptron, one of the first Neural Networks, is made of just a single Neuron

Perceptron : A perceptron takes several inputs, processes them using weights and a bias, and produces an output.



The Perceptron, one of the first Neural Networks, is made of just a single Neuron





A single Neuron from an Artificial Neural Network (ANN)

Neurons

Neurons are at the heart of any neural network, including the perceptron. These digital entities receive inputs, apply weights, and produce an output. Neurons are the building blocks through which information flows in a neural network. Just as neurons in our brains communicate, these fundamental blocks communicate in the language of numbers.

Activation Functions

Activation functions, such as the step function in the perceptron, determine whether the neuron fires. In other words, they decide whether the information flowing through should be allowed to contribute to the output. Picture it as a threshold—if the incoming data is above a certain level, the perceptron 'fires' or produces an output; otherwise, it remains silent. This binary decision-making process showcases the essence of activation functions in shaping the output of our digital neurons.

Weights and Biases

Weights and biases are the adjustable parameters in the network that influence the importance of input features and establish a threshold for activation. Each input data feature is given a certain weight, indicating its importance in the decision. Biases act as the minimum requirement for a feature to contribute. Adjusting weights and biases during model training refines its ability to make accurate predictions.

Loss Functions

Loss functions quantify the difference between the predicted output and the actual target. The objective is to minimize this difference during the training process. In the context of the perceptron, the concept of error or loss becomes evident. The perceptron learns by reducing the difference between its prediction and the actual target, laying the groundwork for understanding more sophisticated loss functions in advanced neural networks. Mean Squared Error (MSE) and Cross-Entropy Loss are examples of loss functions commonly used in different use cases..

Optimizers

optimizers are crucial for adjusting weights and biases based on the computed loss. They fine-tune the model parameters to minimize the loss and improve overall performance. This mechanism hints at the broader optimization techniques in more complex deep learning architectures. Popular optimizing techniques include Stochastic Gradient Descent (SGD), Adam, and RMSprop.

Types of Neural Networks Activation Functions

1. Binary step function (Threshold Function)
2. Linear Activation Function
3. Sigmoid/Logistic Activation Function
4. The derivative of the sigmoid Activation Function
5. Tanh Function (Hyperbolic Tangent)
6. Gradient of the Tanh Activation function
7. ReLu Activation Function

Lets discuss few of them here.

1. **Linear function :** It is defined as

$$f(x) = x \qquad \text{for all } x$$

Here input = output

2. **Bipolar Step function :** The function is defined as :

$$f(x) = \begin{cases} 1, & \text{if } x \geq q \\ -1, & \text{if } x < q \end{cases}$$

3. **Binary step function :** The function is defined as :

$$f(x) = \begin{cases} 1, & \text{if } x \geq q \\ 0, & \text{if } x < q \end{cases}$$

This function is used in single-layer nets to convert the net input to an output that is binary (0 or 1).

4. **Sigmoidal function :** This function is used in back-propagation nets. They are of two types :

   (i) **Binary sigmoidal function :** It is also called as unipolar sigmoid function or a logistic sigmoid function, and defined as

$$f(x) = \frac{1}{1 + e^{-\lambda x}}, \quad \text{where } \lambda \text{ is steepness parameter}$$

The derivative of f(x) is :

$$f'(x) = \frac{-1}{[1 + e^{-\lambda x}]^2} \cdot (-\lambda e^{-\lambda x}) = \frac{\lambda (e^{-\lambda x})}{[1 + e^{-\lambda x}]^2}$$

$$= \frac{\lambda [1 + e^{-\lambda x} - 1]}{[1 + e^{-\lambda x}]^2}$$

$$= \lambda \left\{ \frac{1}{(1 + e^{-\lambda x})} - \frac{1}{(1 + e^{-\lambda x})^2} \right\}$$

$$= \lambda \left[\frac{1}{(1 + e^{-\lambda x})} \left\{ 1 - \frac{1}{(1 + e^{-\lambda x})} \right\} \right]$$

$$= \lambda [f(x) (1 - f(x))] = \lambda f(x) [1 - f(x)]$$

Range of this sigmoid function is 0 to 1

[∵ denominator is always greater than or equal to 1]

(ii) Bipolar sigmoid function :

The function is given by

$$f(x) = \frac{2}{1+e^{-\lambda x}} - 1 = \frac{2-1-e^{-\lambda x}}{1+e^{-\lambda x}} = \frac{1-e^{-\lambda x}}{1+e^{-\lambda x}}$$

where again, λ is steepress parameter and the range is between – 1 and + 1.

Derivative of f(x) is :

$$f'(x) = \frac{[1+e^{-\lambda x}][\lambda e^{-\lambda x}] - (1-e^{-\lambda x})(-\lambda e^{-\lambda x})}{(1+e^{-\lambda x})^2}$$

$$= \frac{\lambda e^{-\lambda x}[1+e^{-\lambda x}+1-e^{-\lambda x}]}{(1+e^{-\lambda x})^2}$$

(5) Ramp function

It is defined as $$f(x) = \begin{cases} 1, \text{ if } x > 1 \\ x, \ 0 \le x \le 1 \\ 0, \ x < 0 \end{cases}$$

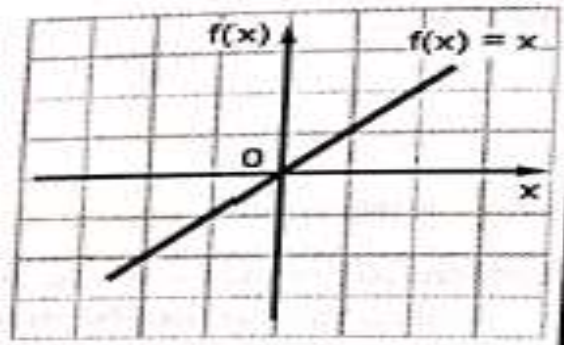
(6) ReLU

ReLU stands for Rectified Linear Unit. Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient.

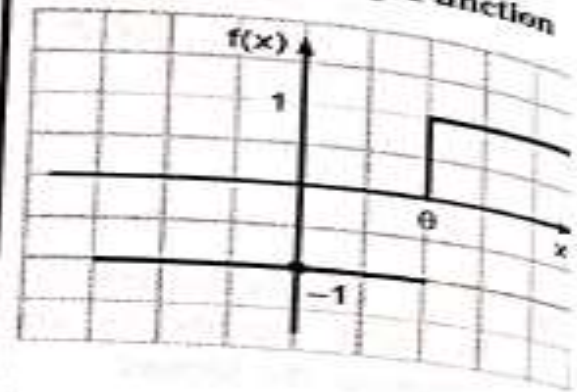
The ReLU function does not activate all the neurons at the same time. The neurons will only be deactivated if the output of the linear transformation is less than 0.

Graphs of activation functions

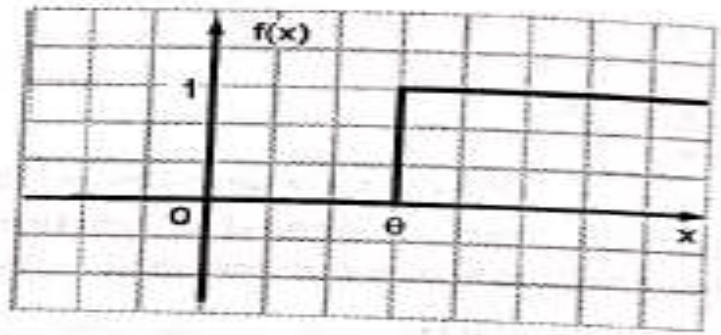
(1) Linear function



(2) Bipolar Step Function

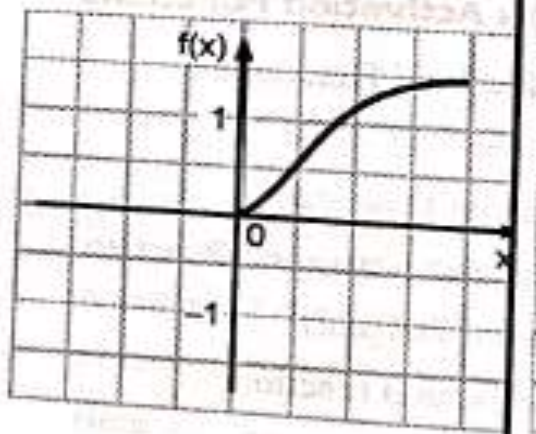


3) Binary-step function

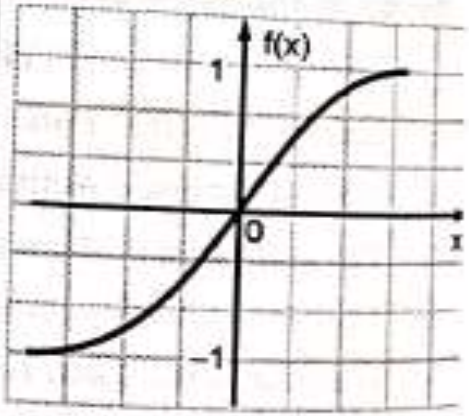


(4)

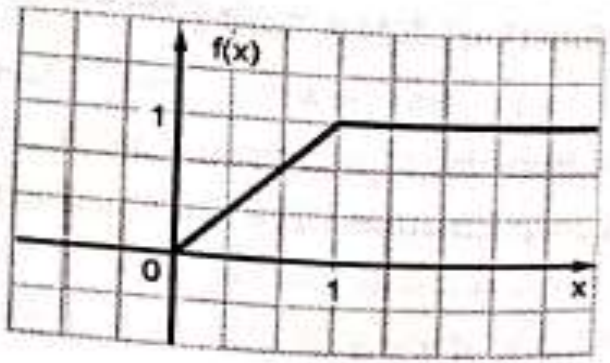
(i) Binary sigmoidal function



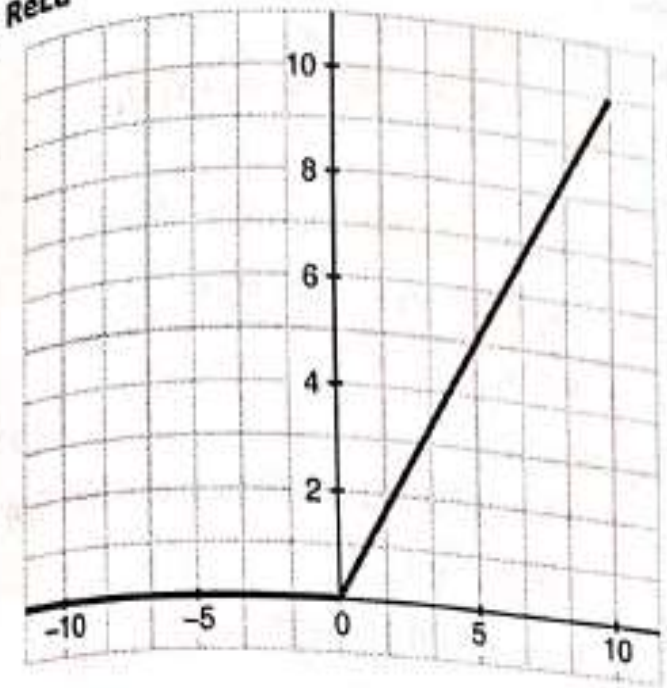
(ii) Bipolar sigmoidal function



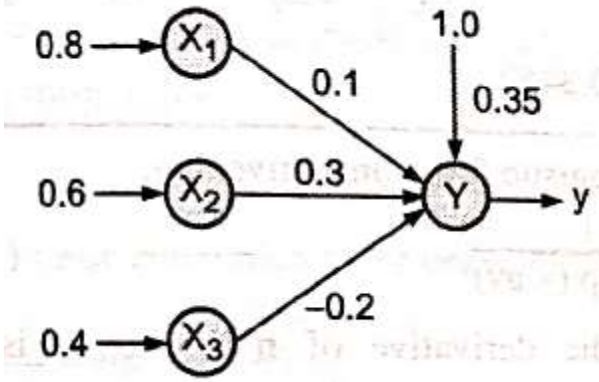
5) Ramp function



(6) ReLu



Obtain the output of the neuron Y for the network shown in the Fig. using activation functions as: (i) binary sigmoidal and (ii) bipolar sigmoidal.



The inputs are : $x_1 = 0.8$; $x_2 = 0.6$; $x_3 = 0.4$

and weights are $w_1 = 0.1$; $w_2 = 0.3$; $w_3 = -0.2$

and bias is $b = 0.35$ (its input is always 1)

: Now the net input to the output neuron is

$$y_{in} = b + \sum_{i=1}^{3} x_i w_i;$$

(∵ only 3 input neurons are given)

$$= b + x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$= 0.35 + (0.8)(0.1) + (0.6)(0.3) + (0.4)(-0.2)$$

$$= 0.35 + 0.08 + 0.18 - 0.08 = 0.53$$

: (i) Binary sigmoidal activation function given by,

$$y = f(y_{in}) = \left(\frac{1}{1 + e^{-y_{in}}}\right) = \frac{1}{1 + e^{-0.53}} = 0.625$$

(ii) bipolar sigmoidal activation functions is given by,

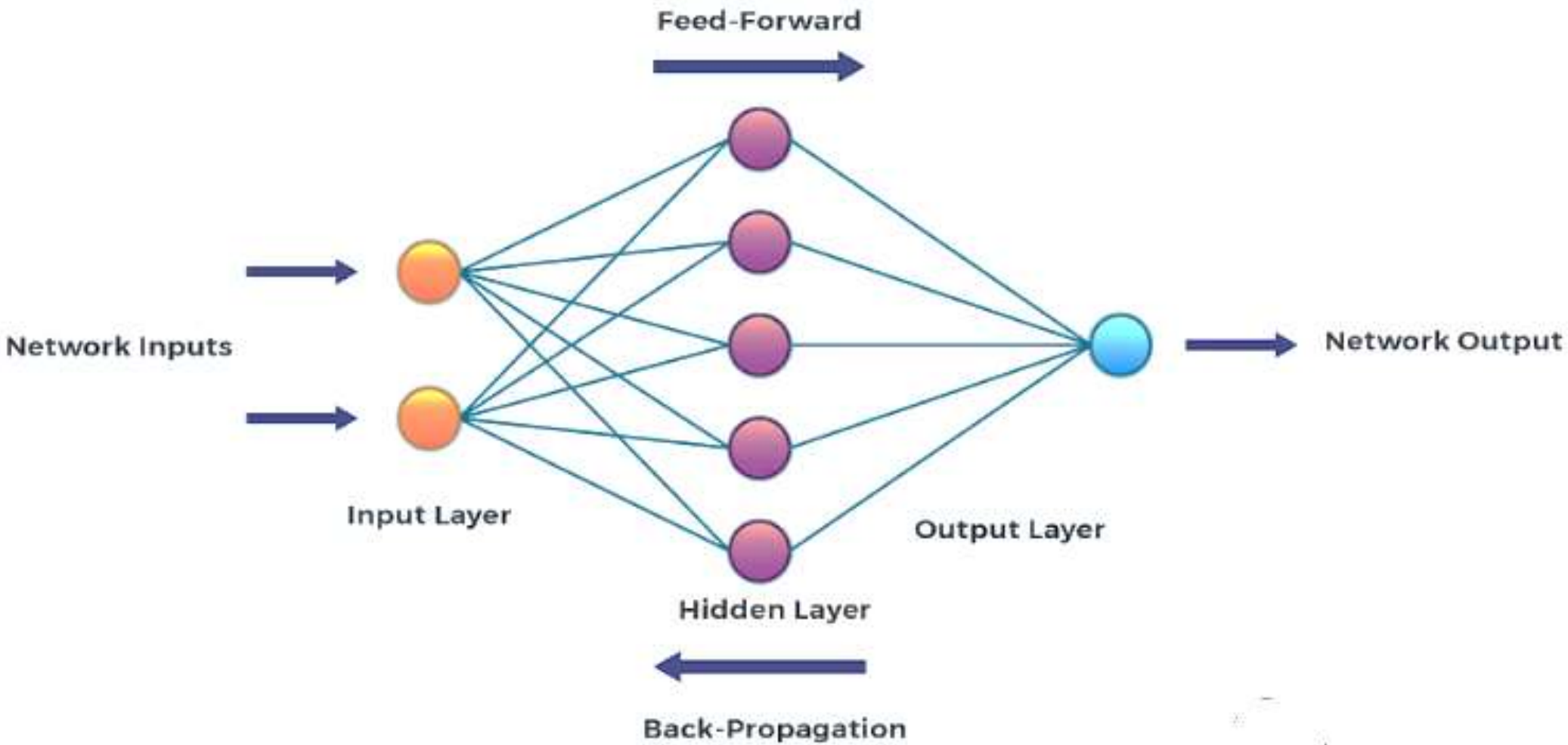
$$y = f(y_{in}) = \left(\frac{2}{1 + e^{-y_{in}}}\right) - 1$$

$$= \left(\frac{2}{1 + e^{-0.53}}\right) - 1 = 0.259$$

1.Artificial Neural Networks

Artificial Neural Networks (ANNs) serve as the foundational architecture in deep learning. At its core, an ANN is characterized by layers of interconnected nodes comprising an input layer, one or more hidden layers, and an output layer. Information flows in a unidirectional manner, progressing from the input layer through the hidden layers to generate the final output.

Introduction to Artificial Neural Networks

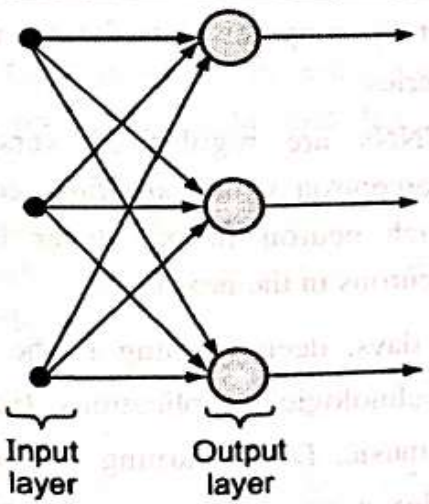
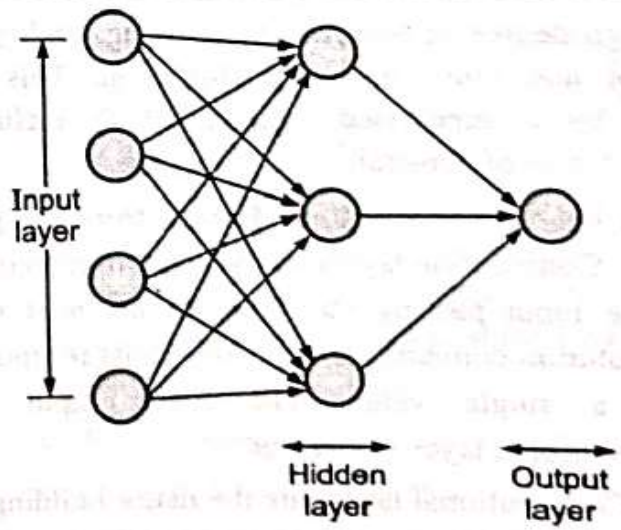


- Each node in the network represents a neuron and is connected to nodes in the adjacent layers.
- These connections are associated with weights, which determine the strength of the connection.
- The input data is fed into the network, and as it passes through each layer, the weighted sum is computed, and an activation function determines the output of each node.
- This process continues until the final layer produces the network's output.
- Next, the backpropagation technique improves the model's performance and accuracy.
- Backpropagation, a vital concept, facilitates learning by adjusting these weights based on prediction errors.
- Input data traverses the network, accumulating weighted sums at each layer, where activation functions govern node outputs. This iterative process reaches its peak in the final output layer, generating the network's output.

- The advantages of ANNs lie in their simplicity, ease of implementation, and the ability to model complex relationships within data.
- Their capacity to learn from large datasets enables them to generalize well to new, unseen data, making them a preferred choice in various real-world scenarios.

Applications

- ANNs find applications in diverse fields, showcasing their adaptability and effectiveness. They excel in pattern recognition systems like image and speech recognition.
- The layered structure allows them to learn hierarchical representations of features, making them adept at discerning intricate patterns within data.
- These networks are widely employed in classification problems, where the goal is to assign inputs to specific categories, and regression problems, where the network predicts numerical values.
- Applications range from predicting stock prices to classifying emails as spam or non-spam.
- One of the key strengths is their capability to approximate any continuous function. This makes ANNs versatile, enabling them to model complex relationships in diverse domains.

Sr. No.	Single layer feed forward Neural Network	Multi-layer feed forward Neural Network
1.	Formation of layer is done by processing and combining elements with other processing elements.	Multi-layer is formed by interconnection of several layers.
2.	There is linking between input and output layer.	Between input and output layers, there are multiplelayers, which are known as hidden layers.
3.	Inputs are connected to the processing nodes with various weights resulting series of output one for node.	Input layers receive input and stores input signal and output layer generates output.
4.	There is nohidden layer.	There are several hidden layers in a network.
5.	In certain applications, it is not efficient.	More hidden layers create complexity of network, but the output is efficient.
		

- 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.
- When it comes to Machine Learning, Artificial Neural Networks perform really well.
- Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use **Recurrent Neural Networks** more precisely an LSTM,
- similarly for image classification we use Convolution Neural networks.

- CNNs are distinguished from classic machine learning algorithms such as **SVMs** and **decision trees** by their ability to autonomously extract features at a large scale, bypassing the need for manual feature engineering and thereby enhancing efficiency.
- The convolutional layers grant CNNs their translation-invariant characteristics, empowering them to identify and extract patterns and features from data irrespective of variations in position, orientation, scale, or translation.
- A variety of pre-trained CNN architectures, including VGG-16, ResNet50, Inceptionv3, and EfficientNet, have demonstrated top-tier performance. These models can be adapted to new tasks with relatively little data through a process known as fine-tuning.
- Beyond image classification tasks, CNNs are versatile and can be applied to a range of other domains, such as natural language processing, time series analysis, and speech recognition.

In a regular Neural Network there are three types of layers:

1.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).

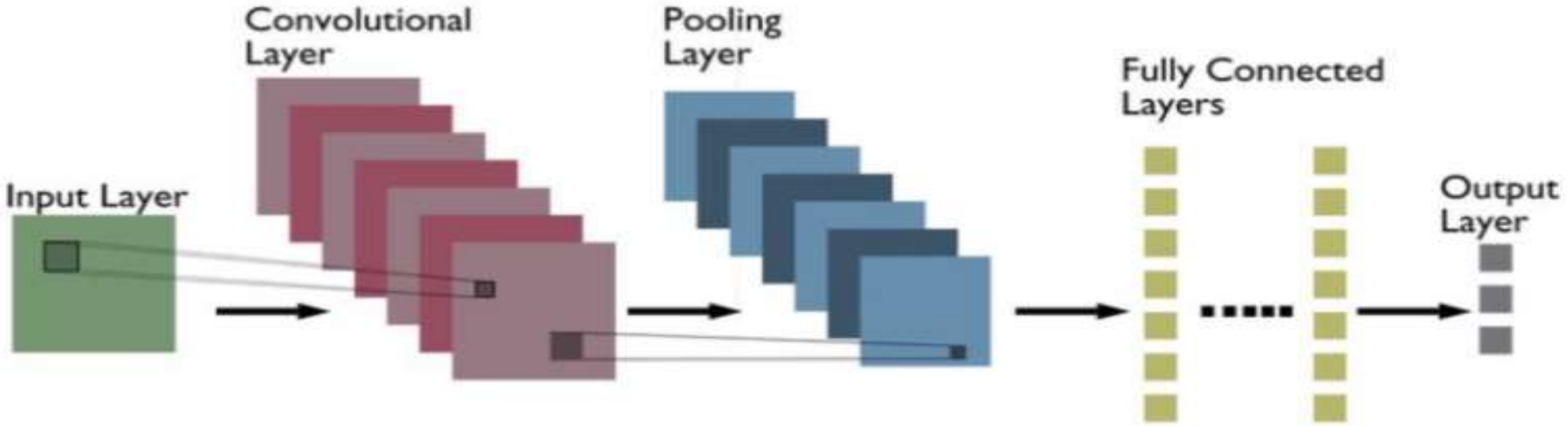
2.Hidden Layer: The input from the Input layer is then fed into the hidden layer. There can be many hidden layers depending on our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of the 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.

3.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 **feedforward**, 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.

- **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.
- **Key Components of a CNN**
- They help the CNNs mimic how the human brain operates to recognize patterns and features in images:
- Convolutional layers
- Rectified Linear Unit (ReLU for short)
- Pooling layers
- Fully connected layers

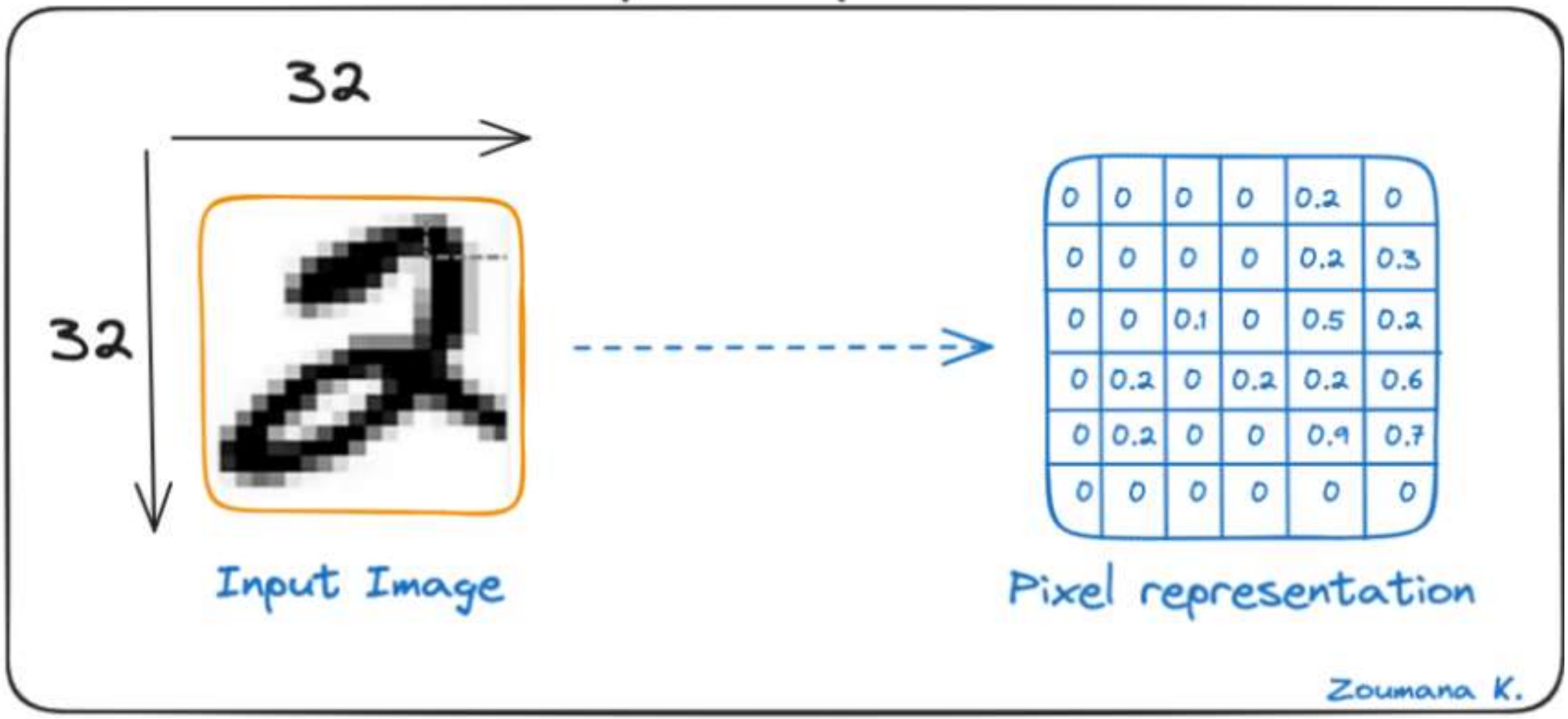
picture below represents a typical CNN architecture



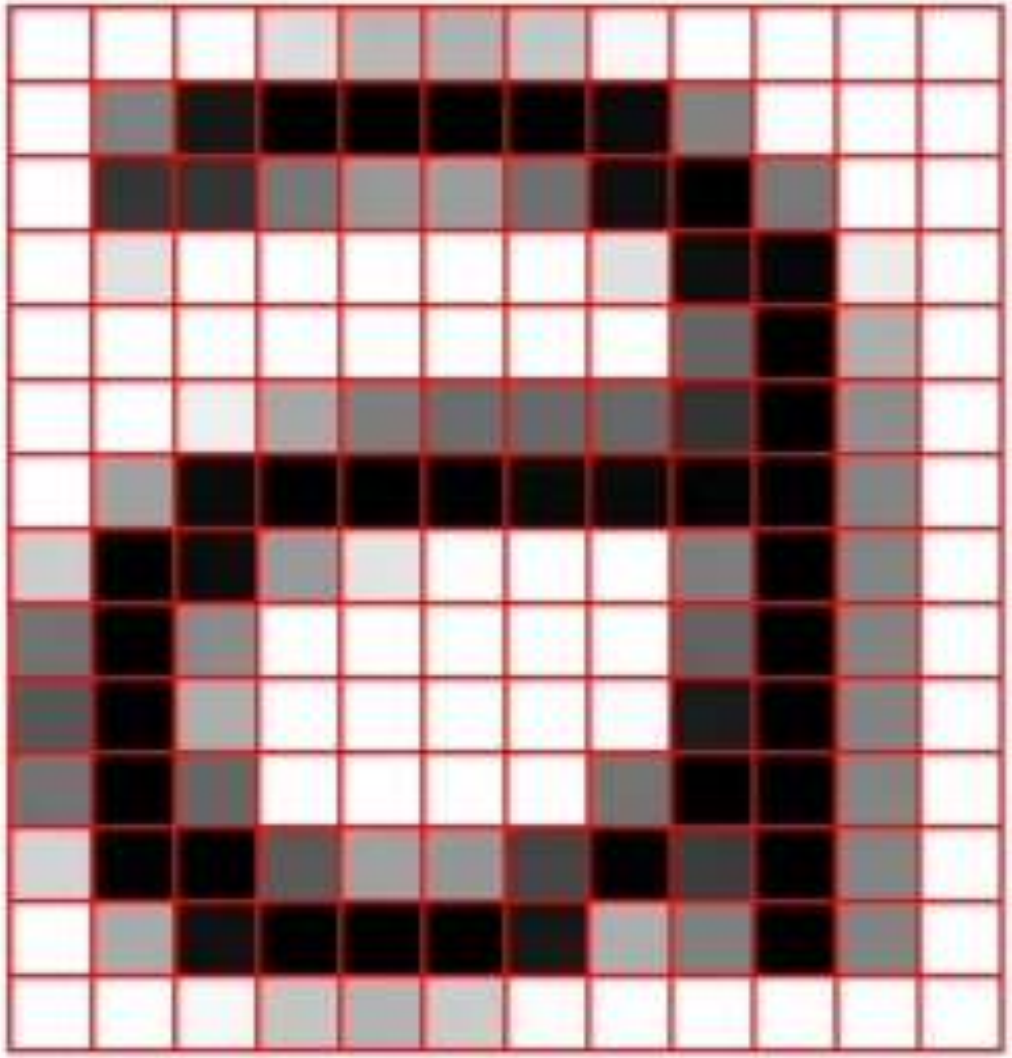
- The Convolutional layer applies filters to the input image to extract features,
- the Pooling layer down samples the image to reduce computation,
- The fully connected layer makes the final prediction.
- The network learns the optimal filters through backpropagation and gradient descent.

- **Convolution layers**
- This is the first building block of a CNN. As the name suggests, the main mathematical task performed is called convolution, which is the application of a sliding window function to a matrix of pixels representing an image. The sliding function applied to the matrix is called kernel or filter, and both can be used interchangeably.
- In the convolution layer, several filters of equal size are applied, and each filter is used to recognize a specific pattern from the image, such as the curving of the digits, the edges, the whole shape of the digits, and more.
- Put simply, in the convolution layer, we use small grids (called filters or kernels) that move over the image. Each small grid is like a mini magnifying glass that looks for specific patterns in the photo, like lines, curves, or shapes. As it moves across the photo, it creates a new grid that highlights where it found these patterns.

Illustration of the input image and its pixel representation



a



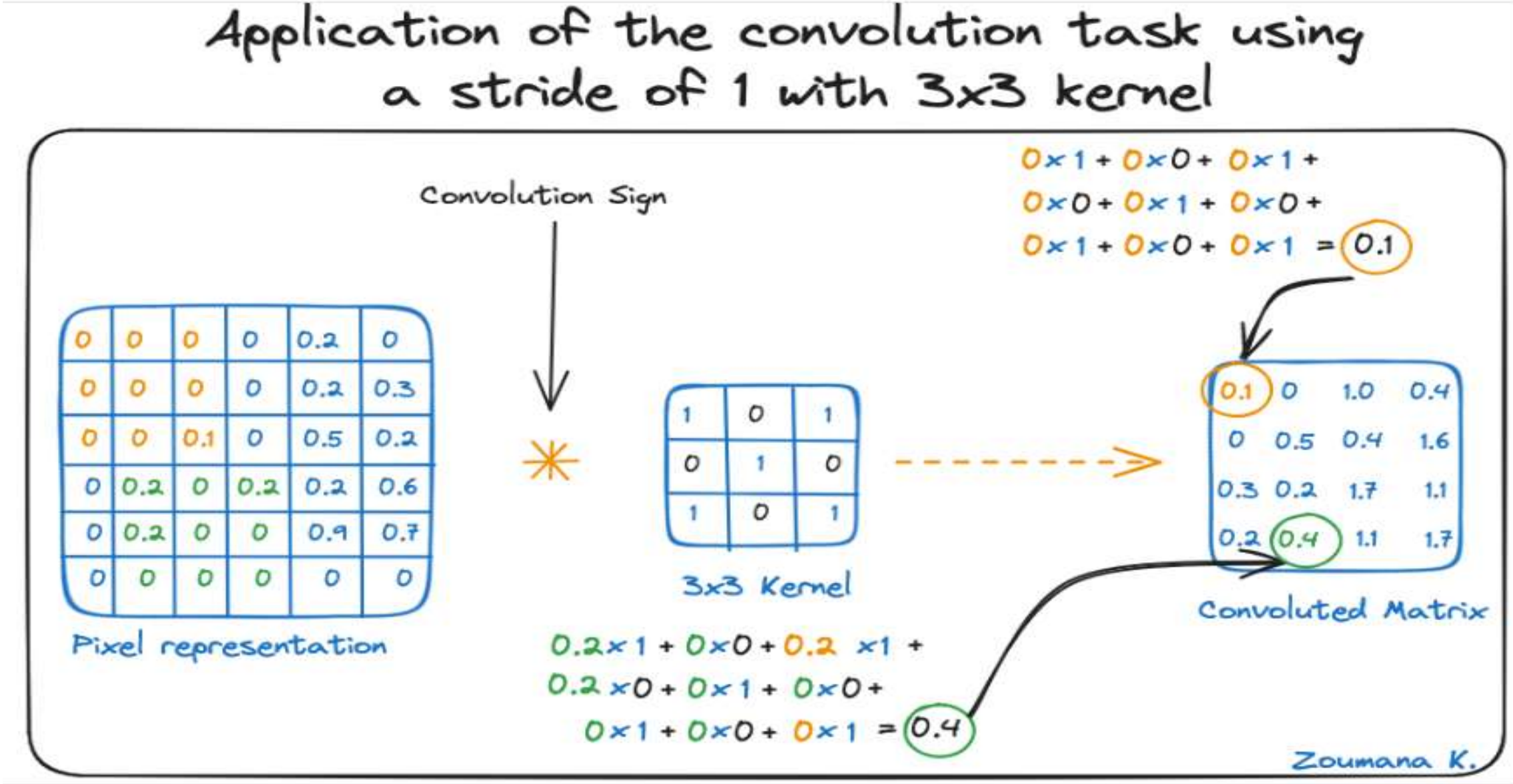
1.0	1.0	1.0	0.9	0.6	0.6	0.6	1.0	1.0	1.0	1.0	1.0
1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	1.0	1.0
1.0	0.2	0.2	0.5	0.6	0.6	0.5	0.0	0.0	0.5	1.0	1.0
1.0	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.9	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
1.0	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.4	0.0	0.5	1.0
1.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0
0.9	0.0	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.6	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.7	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.5	1.0
0.6	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.0	0.5	1.0
0.9	0.1	0.0	0.6	0.7	0.7	0.5	0.0	0.5	0.0	0.5	1.0
1.0	0.7	0.1	0.0	0.0	0.0	0.1	0.9	0.8	0.0	0.5	1.0
1.0	1.0	1.0	0.8	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0

- Also, let’s consider the kernel used for the convolution. It is a matrix with a dimension of 3x3. The weights of each element of the kernel is represented in the grid. Zero weights are represented in the black grids and ones in the white grid.

Using these two matrices, we can perform the convolution operation by applying the dot product, and work as follows:

1. Apply the kernel matrix from the top-left corner to the right.
2. Perform element-wise multiplication.
3. Sum the values of the products.
4. The resulting value corresponds to the first value (top-left corner) in the convoluted matrix.
5. Move the kernel down with respect to the size of the sliding window.
6. Repeat steps 1 to 5 until the image matrix is fully covered.

- The dimension of the convoluted matrix depends on the size of the sliding window. The higher the sliding window, the smaller the dimension.



Application of the convolution task using a stride of 1 with 3×3 kernel

- Also, the dimension of the feature map becomes smaller as the pooling function is applied.
- The last pooling layer flattens its feature map so that it can be processed by the fully connected layer.
- **Fully connected layers**
- These layers are in the last layer of the convolutional neural network, and their inputs correspond to the flattened one-dimensional matrix generated by the last pooling layer.
- ReLU activations functions are applied to them for non-linearity.
- Finally, a softmax prediction layer is used to generate probability values for each of the possible output labels, and the final label predicted is the one with the highest probability score.

- **Activation function**
- A ReLU activation function is applied after each convolution operation.
- This function helps the network learn non-linear relationships between the features in the image,
- hence making the network more robust for identifying different patterns.
- It also helps to mitigate the vanishing gradient problems.

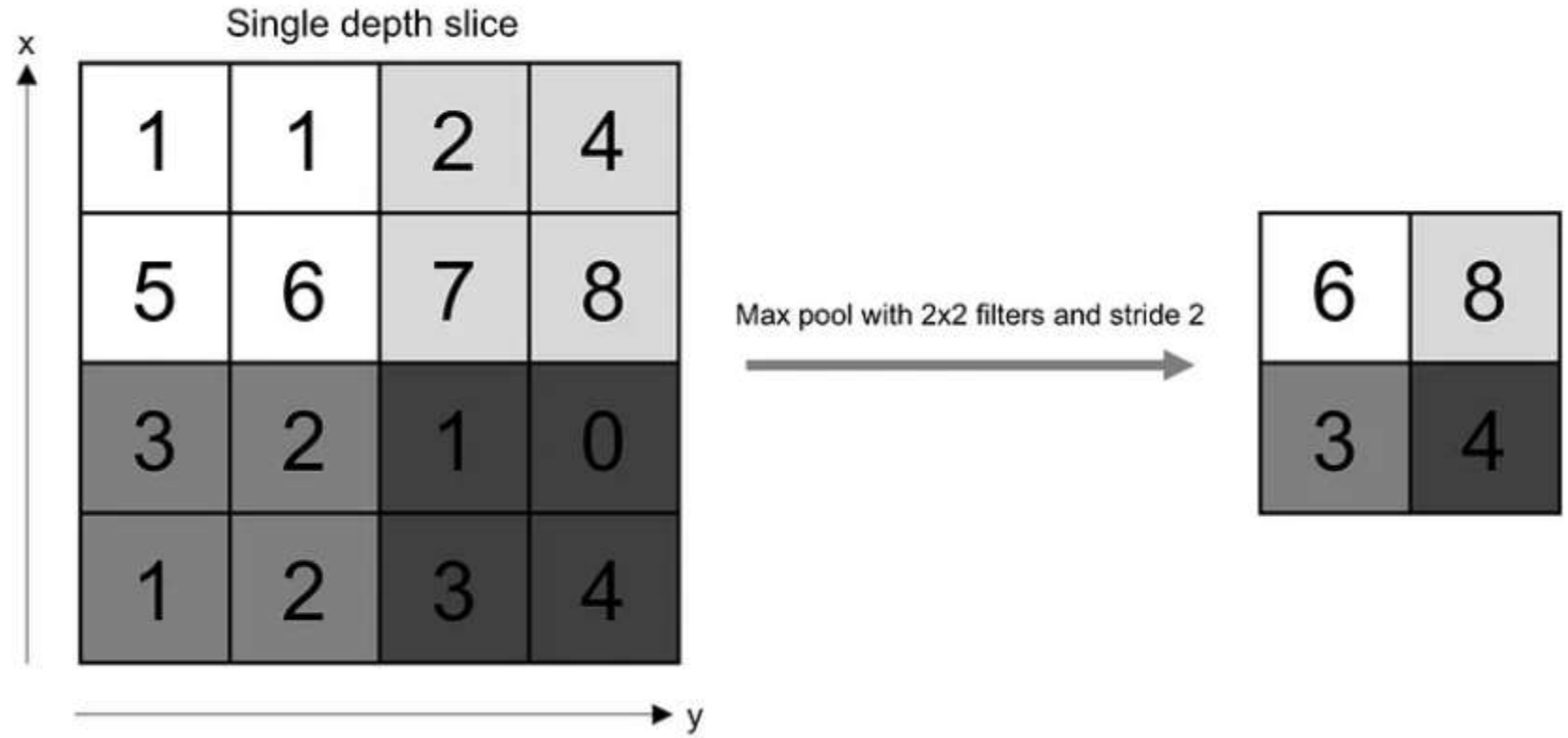
- **Pooling layer**
- The goal of the pooling layer is to pull the most significant features from the convoluted matrix.
- This is done by applying some aggregation operations, which reduce the dimension of the feature map (convoluted matrix), hence reducing the memory used while training the network.
- Pooling is also relevant for mitigating overfitting.
- The most common aggregation functions that can be applied are:
- Max pooling, which is the maximum value of the feature map
- Sum pooling corresponds to the sum of all the values of the feature map
- Average pooling is the average of all the values.

The popular kind of pooling is **Max-pooling**. Suppose we want to pool by a ratio of 2. It implies that the height and width of your image will be half of its original value.

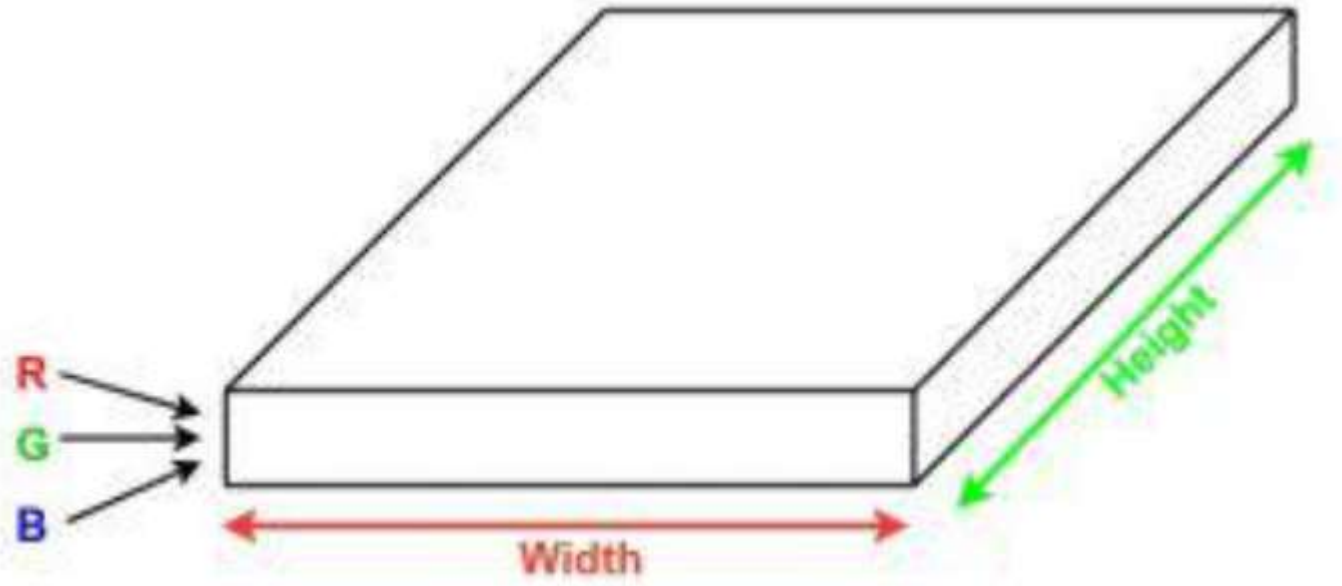
So we have to compress every 4 pixels (a 2×2 gri(d) and map it to a new single pixel with **losing** the **important** data from the **missing** pixels.

Max pooling is done by taking the largest value of these 4 pixels. Thus one new pixel represents 4 old pixels by using the largest value of these 4 pixels. This is repeated for every group of 4 pixels throughout the whole image.

Application of max pooling with a stride of 2 using 2x2 filter


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
Max pool with 2x2 filters and stride 2
6 8
3 4

- **How Convolutional Layers works**
- Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you 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).

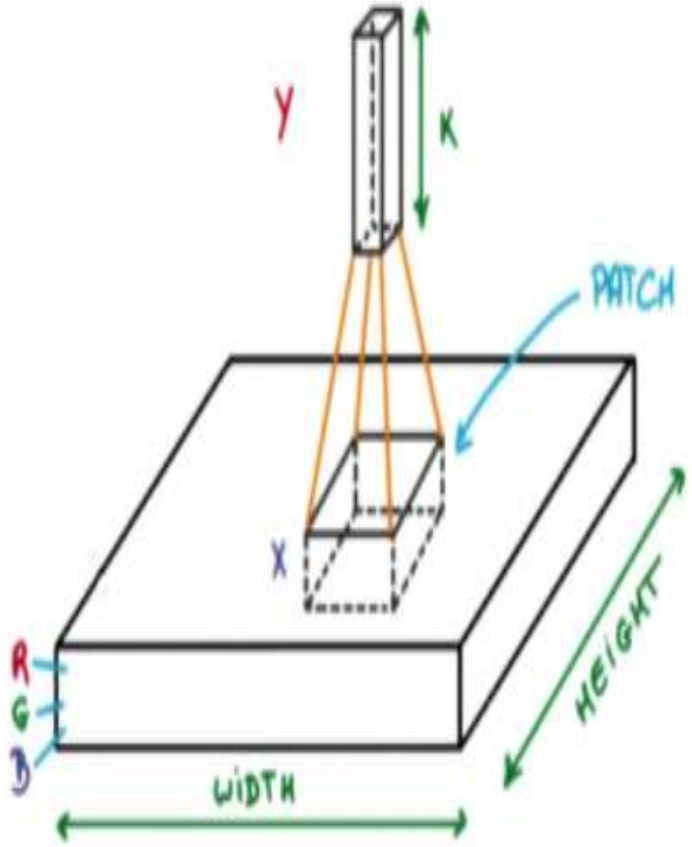


The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.

Output Layer :

The output layer performs final stage of convolution. Each neuron is assigned a receptive field and is assigned the possible characters.

- Now imagine taking a small patch of this image and running a small neural network, called a filter or kernel on it, with say, K outputs and representing them vertically.
- Now slide that neural network across the whole image, as a result, we will get another image with different widths, heights, and depths.
- Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is 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.

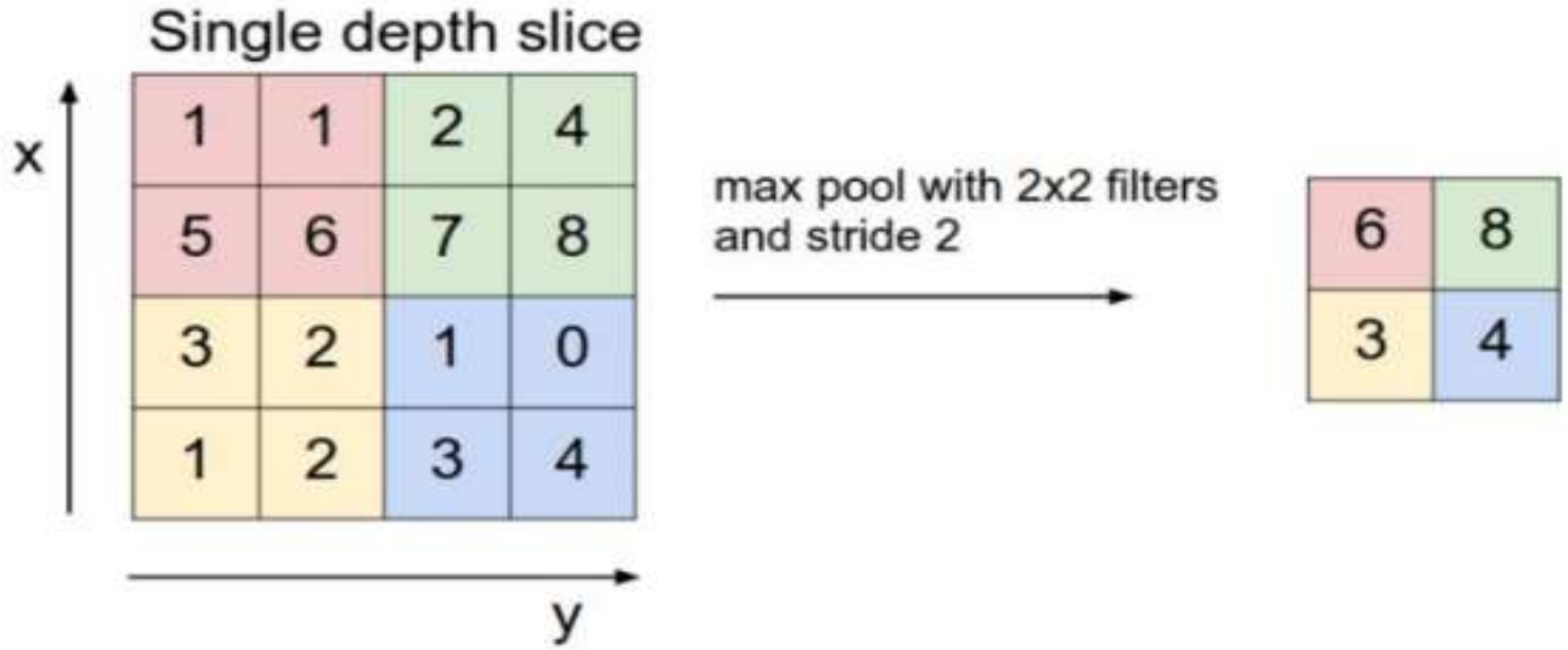


- Convolution layers consist of a set of learnable filters (or kernels) having small widths and heights and the same depth as that of input volume (3 if the input layer is image input).
- For example, if we have to run convolution on an image with dimensions 34x34x3. The possible size of filters can 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.
- As we slide our filters we’ll get a 2-D output for each filter and we’ll stack them together as a result, we’ll get output volume having a depth equal to the number of filters. The network will learn all the filters.

- **Layers used to build ConvNets**
- A complete Convolution Neural Networks architecture is also known as covnets.
- A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

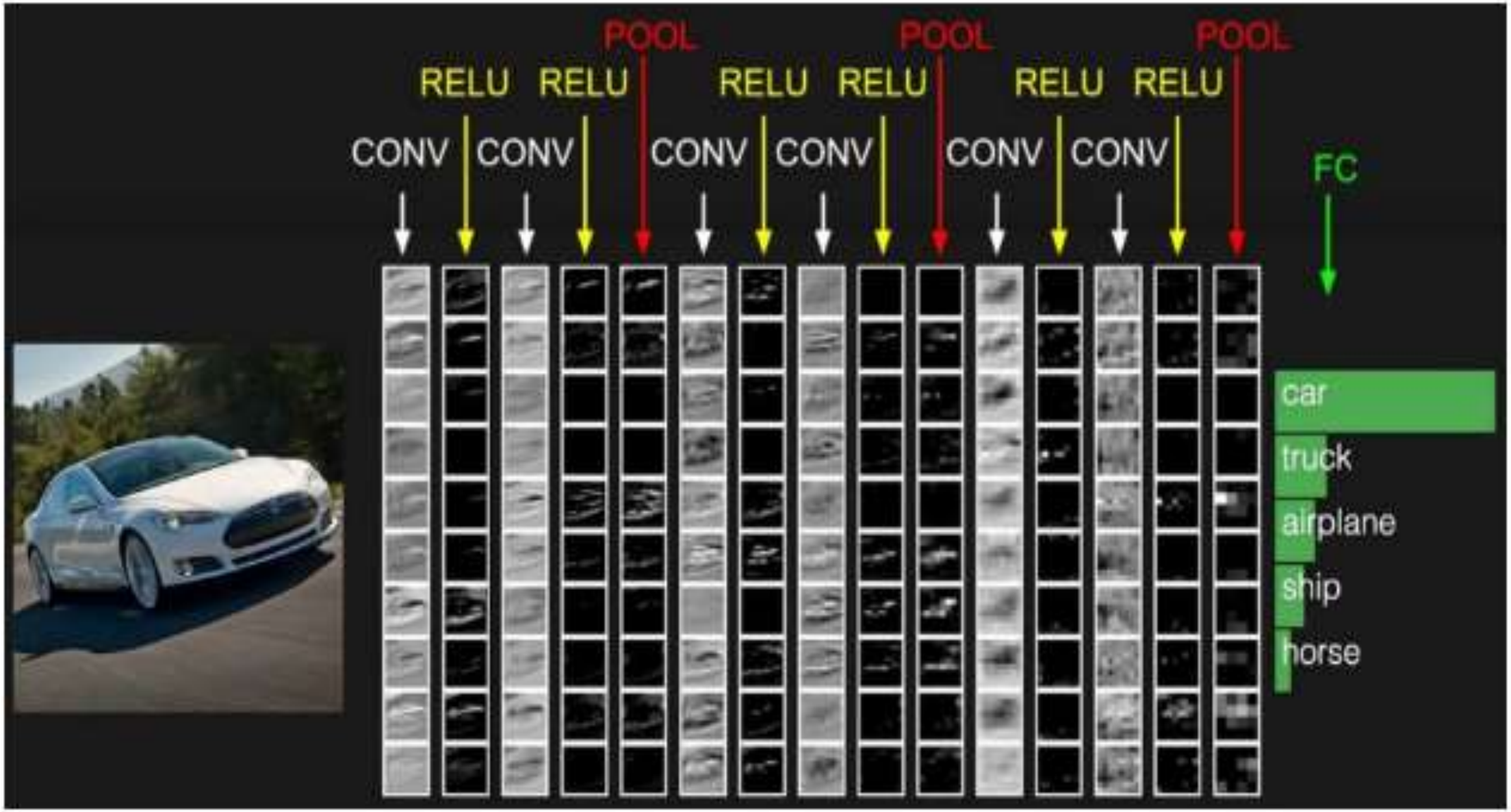
- **Types of layers:** datasets
  Let's take an example by running a covnets on of image of dimension 32 x 32 x 3.
- **Input Layers:** It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.
- **Convolutional Layers:**
- This is the layer, which is used to extract the feature from the input dataset.
- It applies a set of learnable filters known as the kernels to the input images.
- The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape.
- it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch.
- The output of this layer is referred as feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12.

- **Activation Layer:** By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network.
- it will apply an element-wise activation function to the output of the convolution layer.
- Some common activation functions are **RELU**: max(0, x), **Tanh**, **Leaky RELU**, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.
- **Pooling layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting.
- Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume 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 for categorization or regression.

•**Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

- **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.

- **How do CNNs work?**
- *CNNs work by applying a series of convolution and pooling layers to an input image or video.*
- *Convolution layers extract features from the input by sliding a small filter, or kernel, over the image or video and computing the dot product between the filter and the input.*
- *Pooling layers then downsample the output of the convolution layers to reduce the dimensionality of the data and make it more computationally efficient.*

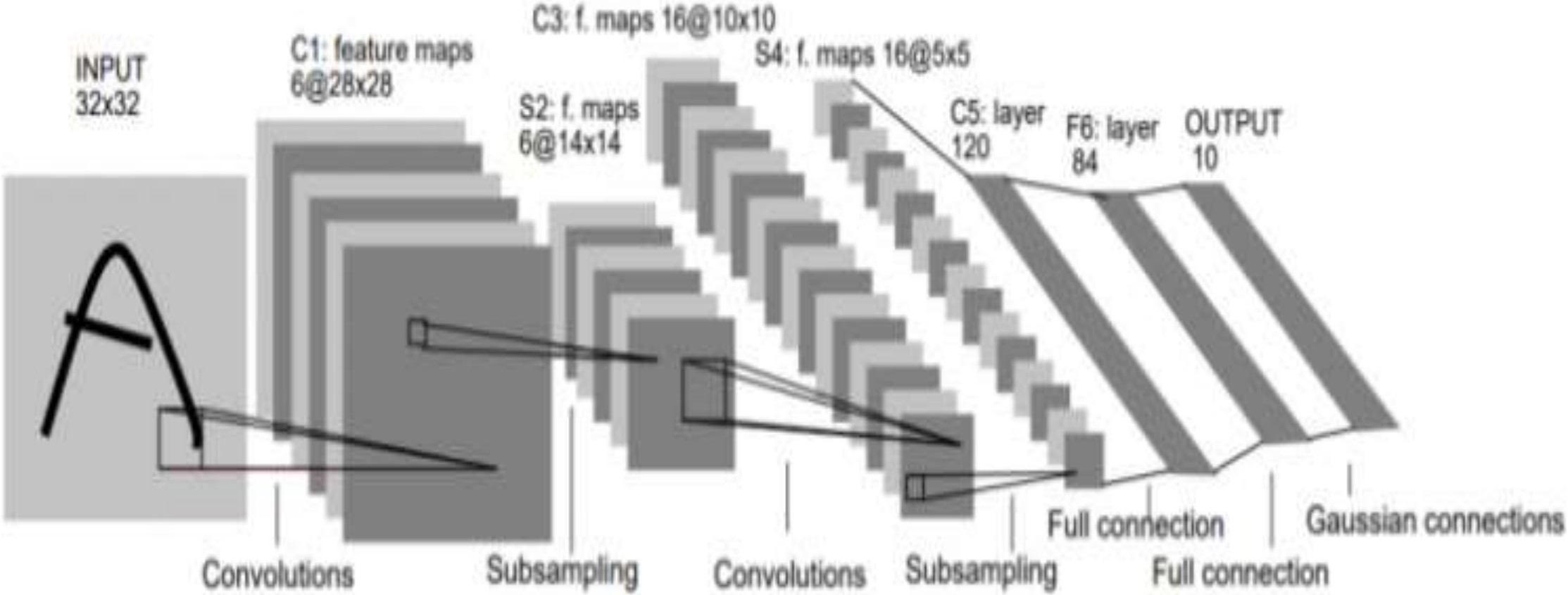
- **Applications of CNN**
- **Image classification:** CNNs are the state-of-the-art models for image classification. They can be used to classify images into different categories, such as cats and dogs, cars and trucks, and flowers and animals.
- **Object detection:** CNNs can be used to detect objects in images, such as people, cars, and buildings. They can also be used to localize objects in images, which means that they can identify the location of an object in an image.
- **Image segmentation:** CNNs can be used to segment images, which means that they can identify and label different objects in an image. This is useful for applications such as medical imaging and robotics.
- **Video analysis:** CNNs can be used to analyze videos, such as tracking objects in a video or detecting events in a video. This is useful for applications such as video surveillance and traffic monitoring.
- **Advantages of CNN**
- CNNs can achieve state-of-the-art accuracy on a variety of image recognition tasks, such as image classification, object detection, and image segmentation.
- CNNs can be very efficient, especially when implemented on specialized hardware such as GPUs.
- CNNs are relatively robust to noise and variations in the input data.
- CNNs can be adapted to a variety of different tasks by simply changing the architecture of the network.
- **Disadvantages of CNN**
- CNNs can be complex and difficult to train, especially for large datasets.
- CNNs can require a lot of computational resources to train and deploy.
- CNNs require a large amount of labeled data to train.
- CNNs can be difficult to interpret, making it difficult to understand why they make the predictions they do.

different types of CNN architectures in deep learning

LeNet – First CNN Architecture

- LeNet was developed in 1998 by Yann LeCun, Corinna Cortes, and Christopher Burges for handwritten digit recognition problems.
- LeNet was one of the first successful CNNs.
- It is one of the earliest and most widely-used
- 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 beginning of CNNs in deep learning for computer vision problems.
- However, LeNet could not train well due to the vanishing gradients problem.
- To solve this issue, a shortcut connection layer known as max-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.
- 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. It was developed more than 20 years ago, its architecture is still relevant today and continues to be used.

The diagram below represents LeNet-5 architecture.



(1) **First Layer :** The input for LeNet-5 is a 32 × 32 grayscale image which passes through the first convolutional layer with 6 feature maps or filters having size 5 × 5 and a stride of one. The image dimensions changes from 32 × 32 × 1 to 28 × 28 × 6.

(2) **Second Layer :** Then the LeNet-5 applies average pooling layer or sub-sampling layer with a filter size 2 × 2 and a stride of two. The resulting image dimensions will be reduced to 14 × 14 × 6.

(3) **Third Layer :** Next, there is a second convolutional layer with 16 feature maps having size 5 × 5 and a stride of 1. In this layer, only 10 out of 16 feature maps are connected to 6 feature maps of the previous layer.

(4) **Fourth Layer :** The fourth layer (S4) is again an average pooling layer with filter size 2×2 and a stride of 2. This layer is the same as the second layer (S2) except it has 16 feature maps so the output will be reduced to 5 × 5 × 16.

(5) **Fifth Layer :** The fifth layer (C5) is a fully connected convolutional layer with 120 feature maps each of size 1 ×
1. Each of the 120 units in C5 is connected to all the 400 nodes (5x5x16) in the fourth layer S4.

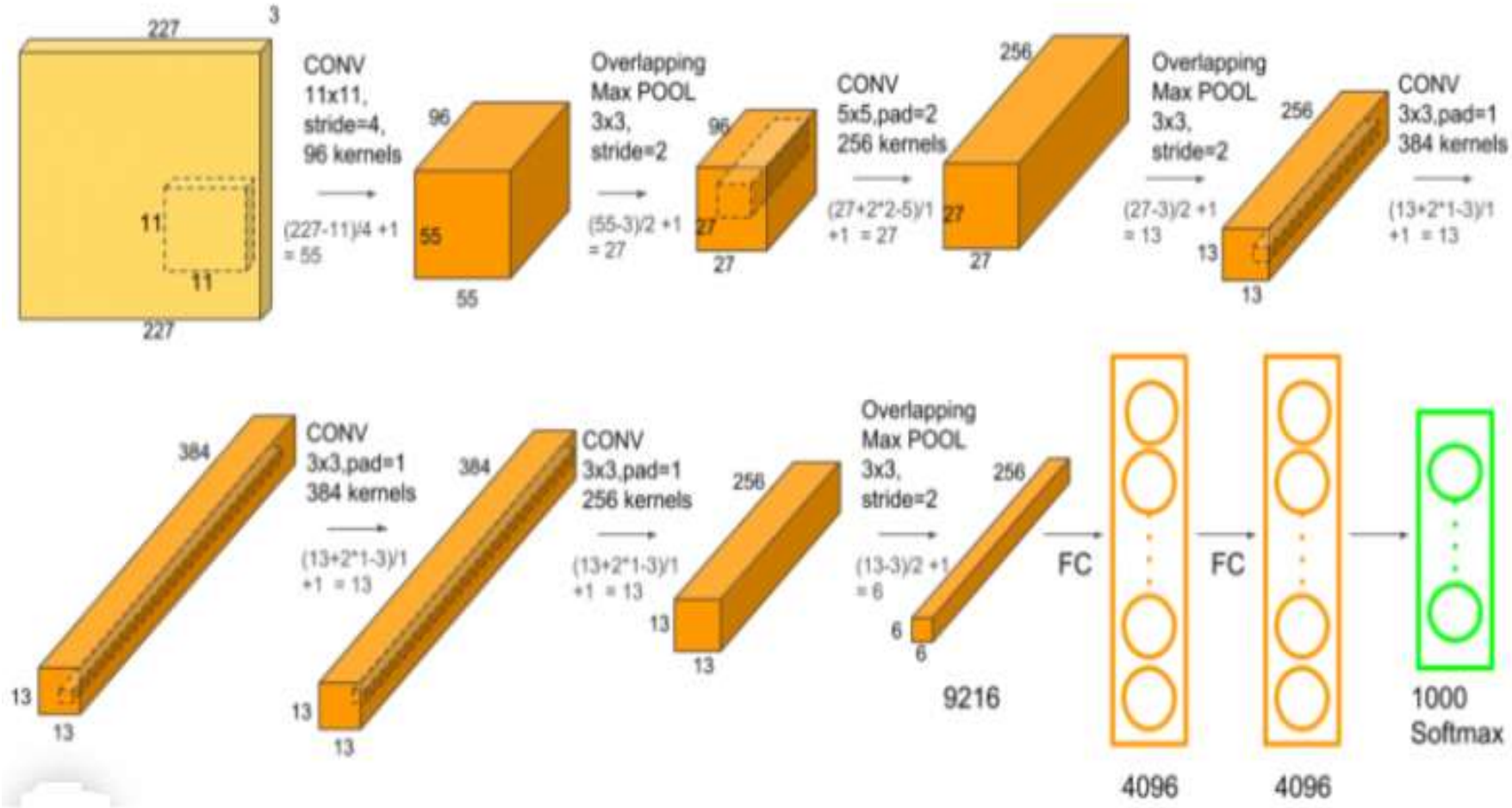
(6) **Sixth Layer :** The sixth layer is a fully connected layer (F6) with 84 units.

(7) **Output Layer :** Finally, there is a fully connected softmax output layer $\hat{y}$ with 10 possible values corresponding to the digits from 0 to 9.

Layer		Feature map	Size	Kernel size	Stride	Activation
Input	Image	1	32 × 32	–	–	–
1	Convolution	6	28 × 28	5 × 5	1	tanh
2	Average pooling	6	14 × 14	2 × 2	2	tanh
3	Convolution	16	10 × 10	5 × 5	1	tanh
4	Average Pooling	16	5 × 5	2 × 2	2	tanh
5	Convolution	120	1 × 1	5 × 5	1	tanh
6	FC	–	84	–	–	tanh
Output	FC	–	10	–	–	softmax

Summary of LeNet-5 architecture

- **AlexNet – Deep Learning Architecture that popularized CNN**
- AlexNet 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) in 2012.
- The AlexNet architecture was designed to be used with large-scale image datasets and it achieved state-of-the-art results at the time of its publication.
- AlexNet is composed of 5 convolutional layers 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.


227
227
3
11
11
CONV
11x11,
stride=4,
96 kernels
(227-11)/4 +1
= 55
96
55
55
Overlapping
Max POOL
3x3,
stride=2
(55-3)/2 +1
= 27
96
27
27
CONV
5x5,pad=2
256 kernels
(27+2*2-5)/1
+1 = 27
256
27
27
Overlapping
Max POOL
3x3,
stride=2
(27-3)/2 +1
= 13
256
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
256 kernels
(13+2*1-3)/1
+1 = 13
256
13
13
Overlapping
Max POOL
3x3,
stride=2
(13-3)/2 +1
= 6
256
6
6
9216
FC
4096
FC
4096
1000
Softmax

2. AlexNet

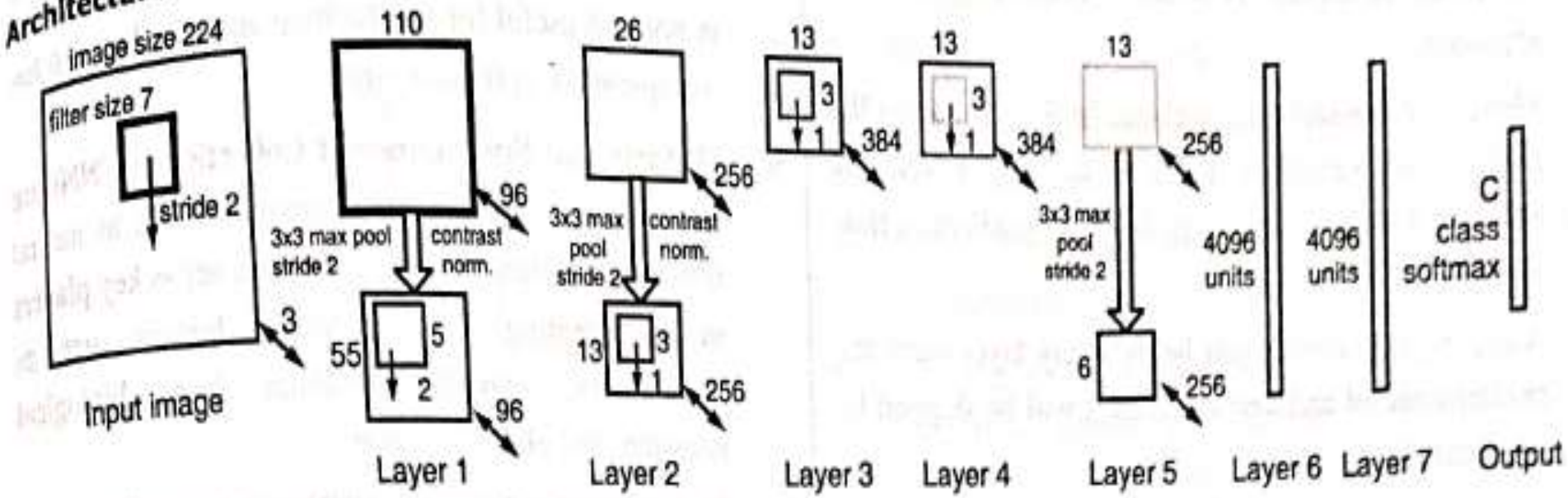
- **AlexNet** was the first convolutional network which used GPU to boost performance. the authors introduced padding to prevent the size of the feature maps from reducing drastically.
- AlexNet won the 2012 ImageNet competition with a top-5 error rate of 15.3% compared to second place top-5 error rate of 26.2%.
- The input to this model is the images of size 227 * 227 * 3.
- AlexNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer.
- Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU.
- The pooling layers are used to perform max pooling.
- Input size is fixed due to the presence of fully connected layers.
- The input size is mentioned at most of the places as $224 \times 224 \times 3$ but due to some padding which happens it works out to be $227 \times 227 \times 3$
- AlexNet overall has 60 million parameters.
- AlexNet was trained on a GTX 580 GPU with only 3 GB of memory which couldn't fit the entire network. So the network was split across 2 GPUs, with half of the neurons(feature maps) on each GPU.

Layer	8 filters /Filter size neurons	Stride	Padding	Size of feature rump	Activation function
Dropout 1	rate = 0.5	–	–	6 x 6 x 256	–
Fully Connected 1	–	–	–	4096	ReLU
Dropout 2	rate = 0.5	–	–	4096	–
Fully Connected 2	–	–	–	4096	ReLU
Fully Connected 3	–	–	–	1004	Soltmax

: Summary of Fully Connected and Dropout layers in AlexNet

ZF Net: ZFnet is the CNN architecture that uses a combination of fully-connected layers and CNNs. ZF Net 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 accuracy with only 1000 images per class. It was an improvement on AlexNet by tweaking 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 on the 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. Here is the paper on ZFNet.

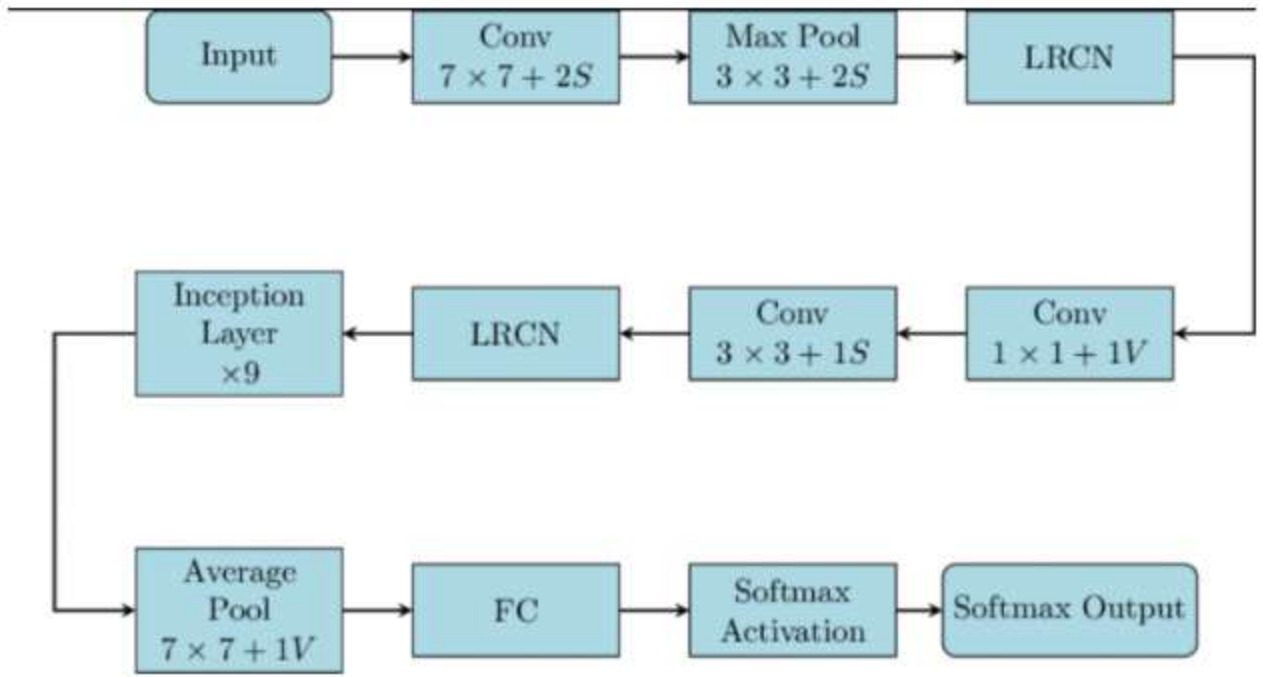
Architecture



Architecture of ZFNet

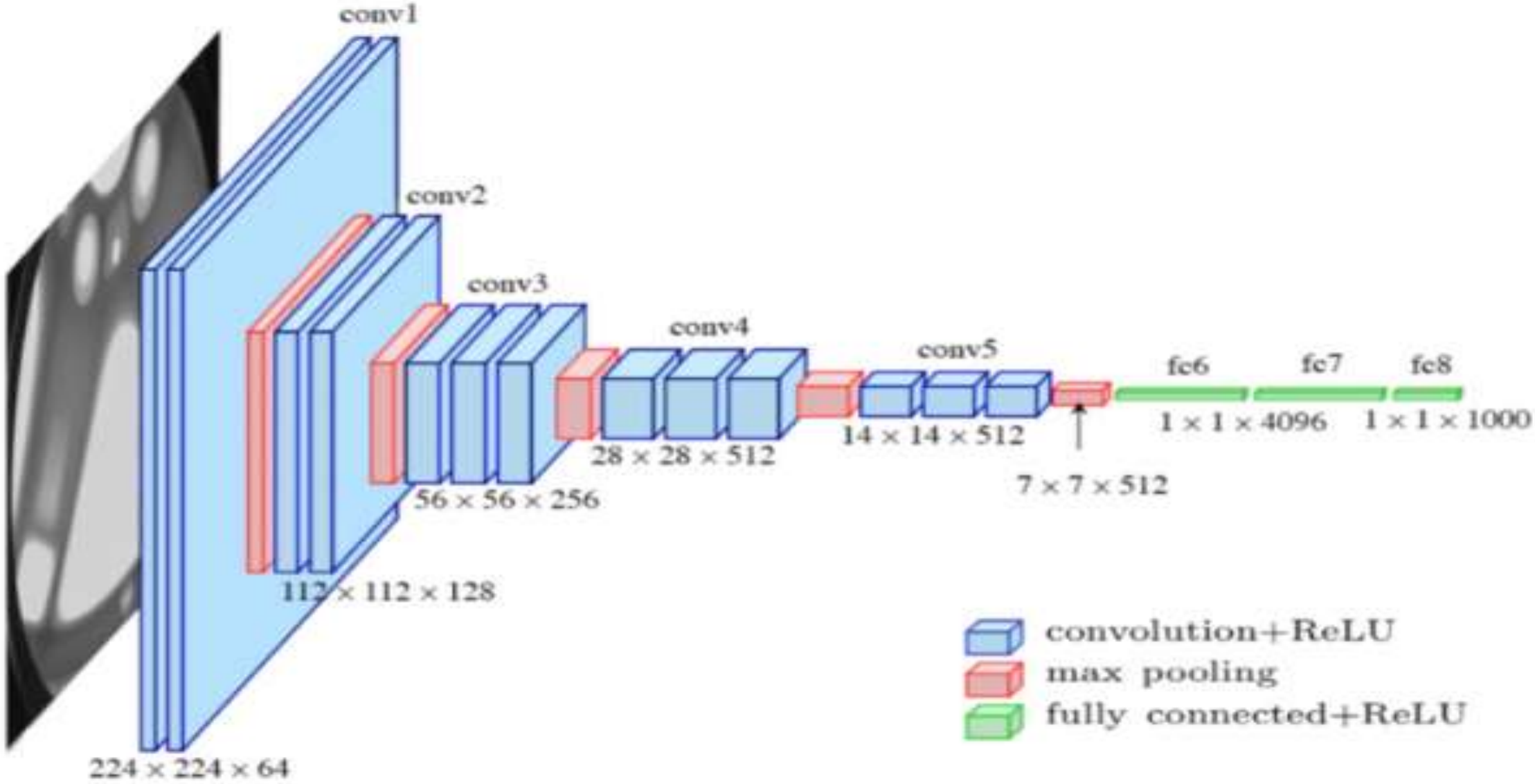
- Our input is 224x224x3 images.
- Next, 96 convolutions of 7x7 with a stride of 2 are performed, followed by ReLU activation, 3x3 max pooling with stride 2 and local contrast normalization.
- Followed by it are 256 filters of 3x3 each which are then again local contrast normalized and pooled.
- The third and fourth layers are identical with 384 kernels of 3x3 each.
- The fifth layer has 256 filters of 3x3, followed by 3x3 max pooling with stride 2 and local contrast normalization.
- The sixth and seventhlayers house 4096 dense units each.
- Finally, we feed into a Dense layer of 1000 neurons i.e., the number of classes in ImageNet.

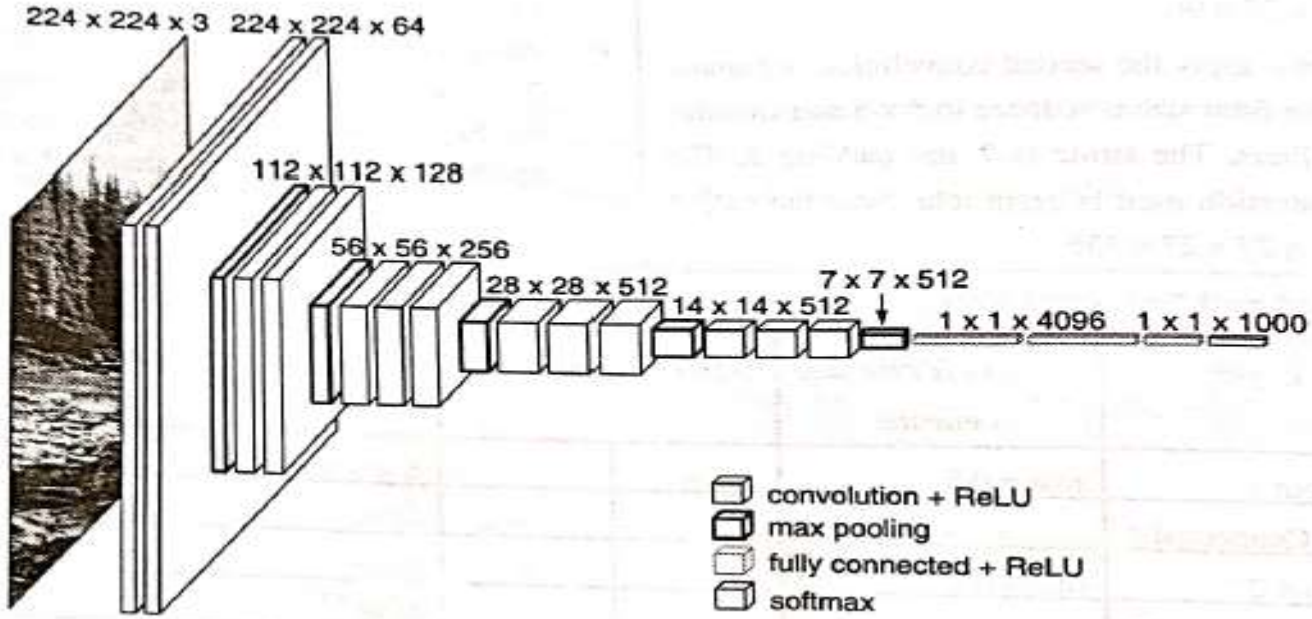
- GoogLeNet – CNN Architecture used by Google
- GoogLeNet is the CNN architecture used by Google to win ILSVRC 2014 classification task.
- It was developed by Jeff Dean, Christian Szegedy, Alexandro Szegedy et al..
- It has been shown to have a notably reduced error rate in comparison with previous winners 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 deeper architecture by employing a number of distinct techniques, including 1×1 convolution 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 redundancy during training and also features shortcut connections between the first two convolutional layers 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:


Input
Conv
7 × 7 + 2S
Max Pool
3 × 3 + 2S
LRCN
Conv
1 × 1 + 1V
Conv
3 × 3 + 1S
LRCN
Inception
Layer
×9
Average
Pool
7 × 7 + 1V
FC
Softmax
Activation
Softmax Output

GoogLeNet

- VGGNet – CNN Architecture with Large Filters
- VGGNet is the CNN architecture that was developed by Karen Simonyan, Andrew Zisserman et al. at Oxford University. The **full form of "VGG16"** stands for "**Visual Geometry Group 16**".
- This name comes from the Visual Geometry Group at the University of Oxford, where this neural network architecture was developed.
- The "16" in the name indicates that the model contains 16 layers that have weights; this includes convolutional layers as well as fully connected layers.
- 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 input images 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 ILSVRC 2014 classification task, which was also won by GoogleNet CNN architecture.
- The VGG CNN model is computationally efficient and serves as a strong baseline for many applications in computer vision due 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:


conv1
conv2
conv3
conv4
conv5
fc6
fc7
fc8
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU



Architecture of VGGNet

- The first two layers are convolutional layers with 3×3 filters, and first two layers use 64 filters that results in $224 \times 224 \times 64$ volume as same convolutions are used.
- The filters are always 3×3 with stride of 1.
- After this, pooling layer was used with max-pool of 2×2 size and stride 2 which reduces height and width of a volume from $224 \times 224 \times 64$ to $112 \times 112 \times 64$.
- This is followed by 2 more convolution layers with 128 filters.
- This results in the new dimension of $112 \times 112 \times 128$.
- After pooling layer is used, volume is reduced to $56 \times 56 \times 128$.
- Two more convolution layers are added with 256 filters each followed by down sampling layer that reduces the size to $28 \times 28 \times 256$.
- Two more stack each with 3 convolution layer is separated by a max-pool layer.
- After the final pooling layer, $7 \times 7 \times 512$ volume is flattened into Fully Connected (F(C) layer with 4096 channels and softmax output of 1000 classes.

- **ResNet – CNN architecture that also got used for NLP tasks apart from Image Classification**
- ResNet is the CNN architecture that 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 classification tasks 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 by the Microsoft Research Asia team in 2016 and 2017 respectively.
- Real-life applications/examples of ResNet CNN architecture include Microsoft's machine comprehension system, which has used CNNs to generate the answers for more than 100k questions in over 20 categories.
- The CNN architecture ResNet is computationally efficient and can be scaled up or down to match the computational power of GPUs.

- MobileNets – CNN Architecture for Mobile Devices
- 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.. They are usually very small CNN architectures, which makes them easy to run in real-time using embedded devices like smartphones and 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 include CNNs that is built into Android phones to run Google's Mobile Vision API, which can automatically identify labels of popular objects in images.

- GoogLeNet_DeepDream – Generate images based on CNN features
- GoogLeNet_DeepDream is a deep dream CNN architecture that was developed by Alexander Mordvintsev, Christopher Olah, et al..
- It uses the Inception network to generate images based on CNN features.
- The architecture is often used with the ImageNet dataset to generate psychedelic images or create abstract artworks using human imagination at the ICLR 2017 workshop by David Ha, et al.

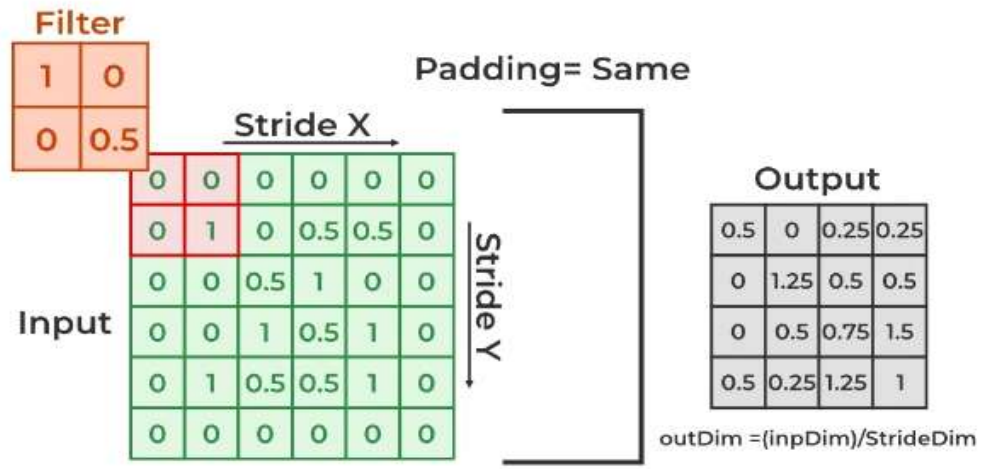
Architecture	Year	Key Features	Use Case
LeNet	1998	First successful applications of CNNs, 5 layers (alternating between convolutional and pooling), Used tanh/sigmoid activation functions	Recognizing handwritten and machine-printed characters
AlexNet	2012	Deeper and wider than LeNet, Used ReLU activation function, Implemented dropout layers, Used GPUs for training	Large-scale image recognition tasks
ZFNet	2013	Similar architecture to AlexNet, but with different filter sizes and numbers of filters, Visualization techniques for understanding the network	ImageNet classification
VGGNet	2014	Deeper networks with smaller filters (3×3), All convolutional layers have the same depth, Multiple configurations (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 more efficient computation and deeper networks, multiple versions (Inception v1, v2, v3, v4)	Large-scale image recognition, won 1st place in the ILSVRC 2014
MobileNets	2017	Designed for mobile and embedded vision applications, Uses depthwise separable convolutions to reduce the model size and complexity	Mobile and embedded vision applications, real-time object detection

- During convolution, the size of the output feature map is determined by the size of the input feature map, the size of the kernel, and the stride. if we simply apply the kernel on the input feature map, then the output feature map will be smaller than the input. This can result in the loss of information at the borders of the input feature map. In Order to preserve the border information we use padding.
- **What Is Padding**
- padding is a technique used to preserve the spatial dimensions of the input image after convolution operations on a feature map. Padding involves adding extra pixels around the border of the input feature map before convolution.
- **This can be done in two ways:**
- **Valid Padding**: In the valid padding, no padding is added to the input feature map, and the output feature map is smaller than the input feature map. This is useful when we want to reduce the spatial dimensions of the feature maps.
- **Same Padding**: In the same padding, padding is added to the input feature map such that the size of the output feature map is the same as the input feature map. This is useful when we want to preserve the spatial dimensions of the feature maps.
- The number of pixels to be added for padding can be calculated based on the size of the kernel and the desired output of the feature map size.
- The most common padding value is zero-padding, which involves adding zeros to the borders of the input feature map.
- Padding can help in reducing the loss of information at the borders of the input feature map and can improve the performance of the model.
- However, it also increases the computational cost of the convolution operation.
- Overall, padding is an important technique in CNNs that helps in preserving the spatial dimensions of the feature maps and can improve the performance of the model.

- **Problem With Convolution Layers Without Padding**
- For a grayscale (n x n) image and (f x f) filter/kernel, the dimensions of the image resulting from a convolution operation is
- **(n – f + 1) x (n – f + 1)**.
- For example, for an (8 x 8) image and (3 x 3) filter, the output resulting after the convolution operation would be of size (6 x 6).
- Thus, the image shrinks every time a convolution operation is performed.
- This places an upper limit to the number of times such an operation could be performed before the image reduces to nothing thereby precluding us from building deeper networks.

Effect Of Padding On Input Images

Padding is simply a process of adding layers of zeros to our input images so as to avoid the problems mentioned above through the following changes to the input image.



padding in convolutional network

Padding prevents the shrinking of the input image.

p = number of layers of zeros added to the border of the image,

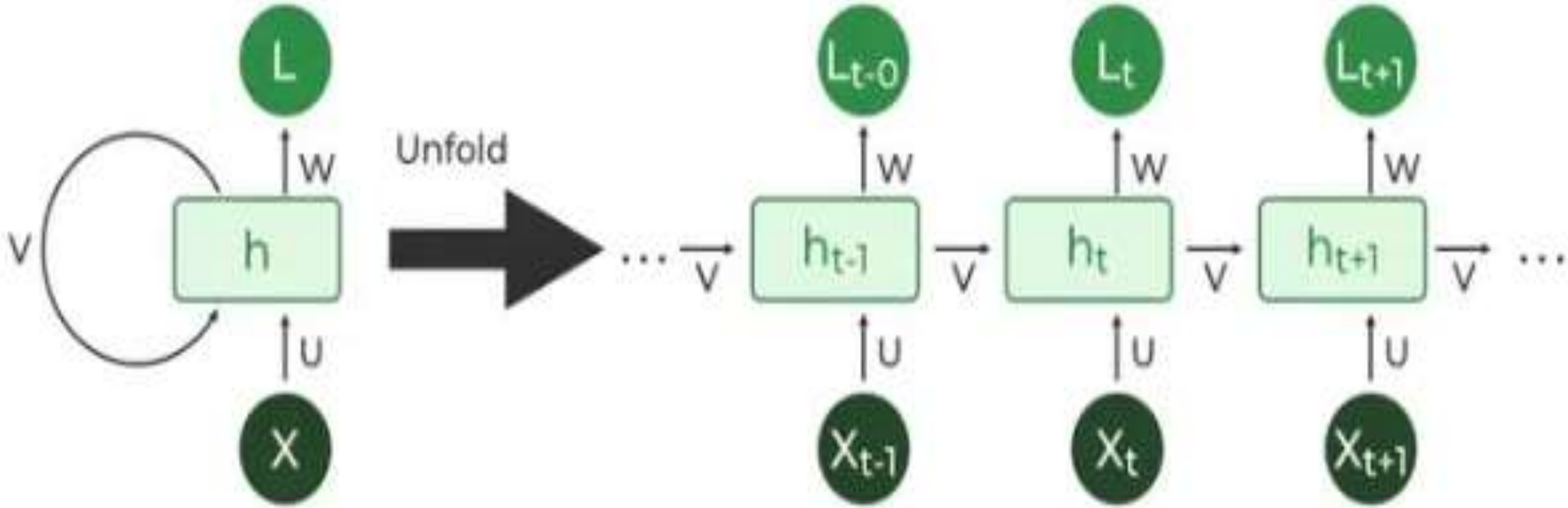
then ***(n x n)*** *image —> **(n + 2p) x (n + 2p)** image after padding.*

***(n + 2p) x (n + 2p) * (f x f) —> outputs (n + 2p – f + 1) x (n + 2p – f + 1)** images*

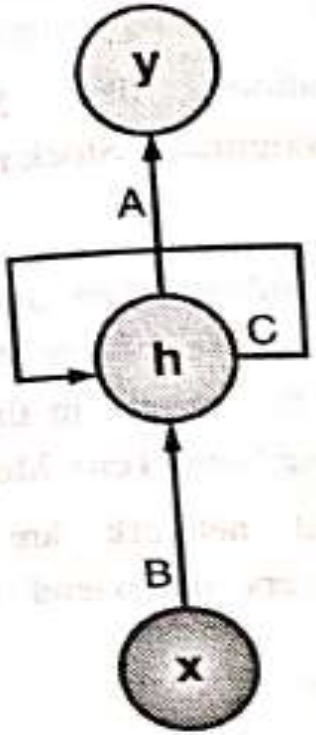
For example, by adding one layer of padding to an (8 x 8) image and using a (3 x 3) filter we would get an (8 x 8) output after performing a convolution operation.

This increases the contribution of the pixels at the border of the original image by bringing them into the middle of the padded image. Thus, information on the borders is preserved as well as the information in the middle of the image.

- **What is Recurrent Neural Network (RNN)?**
- Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step.
- In traditional neural networks, all the inputs and outputs are independent of each other.
- Still, in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words.
- Thus RNN came into existence, which solved this issue with the help of a Hidden Layer.
- The main and most important feature of RNN is its **Hidden state**, which remembers some information about a sequence.
- The state is also referred to as *Memory State* since it remembers the previous input to the network.
- It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output.
- This reduces the complexity of parameters, unlike other neural networks.

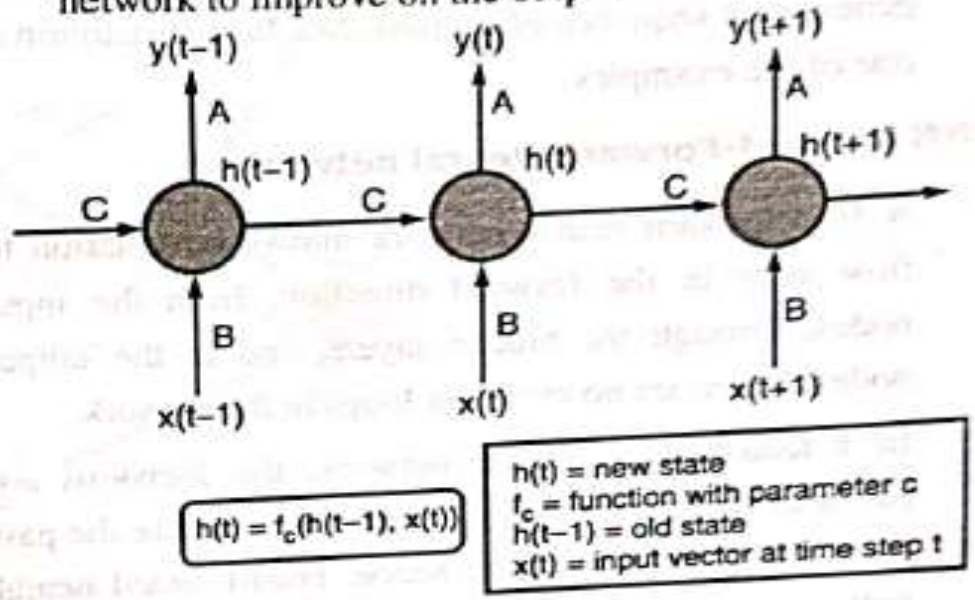


Recurrent Neural Network



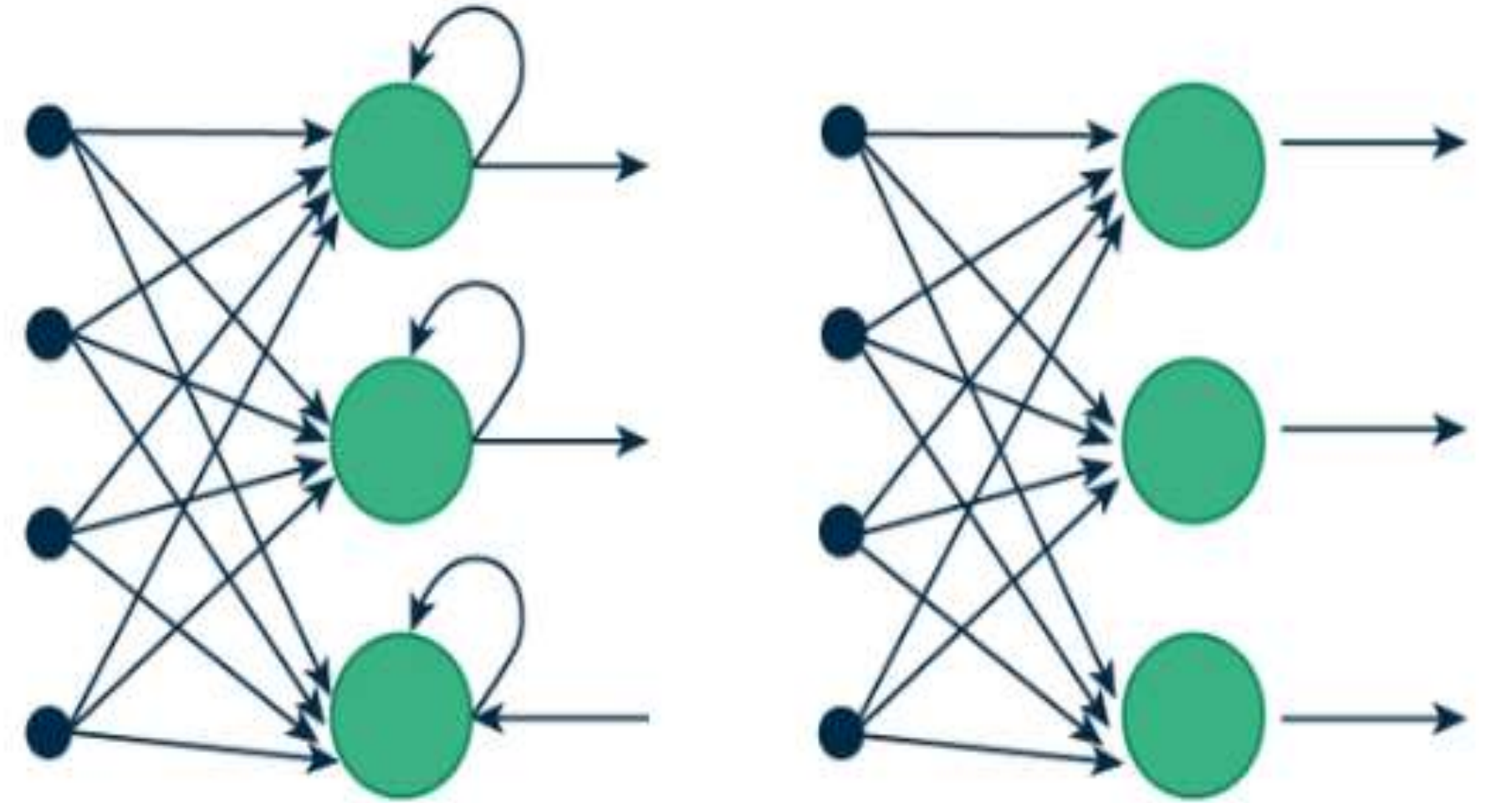
Simple RNN

- As indicated in fig. the nodes in different layers of the neural network are compressed to form a single layer of recurrent neural networks.
- Here, "x" is the input layer, "h" is the hidden layer, and "y" is the output layer. A, B, and C are the network parameters used to improve the output of the model.
- RNN uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output.
- This reduces the complexity of parameters, unlike other neural networks.
- 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.



: Fully connected RNN

- **How RNN differs from Feedforward Neural Network?**
- Artificial neural networks that do not have looping nodes are called feed forward neural networks. Because all information is only passed forward, this kind of neural network is also referred to as a multi-layer neural network.
- Information moves from the input layer to the output layer – if any hidden layers are present – unidirectionally in a feedforward neural network.
- These networks are appropriate for image classification tasks, for example, where input and output are independent.
- Nevertheless, their inability to retain previous inputs automatically renders them less useful for sequential data analysis.



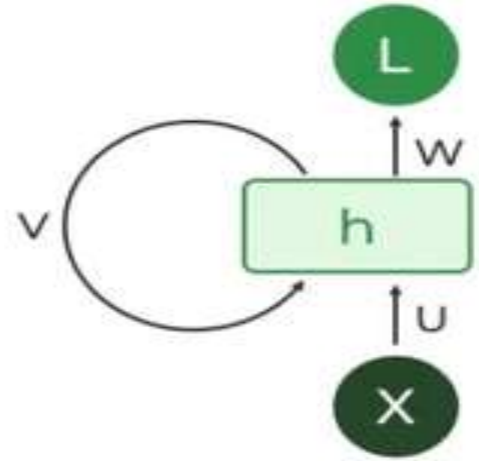
Recurrent Vs Feedforward networks

Recurrent Neuron and RNN Unfolding

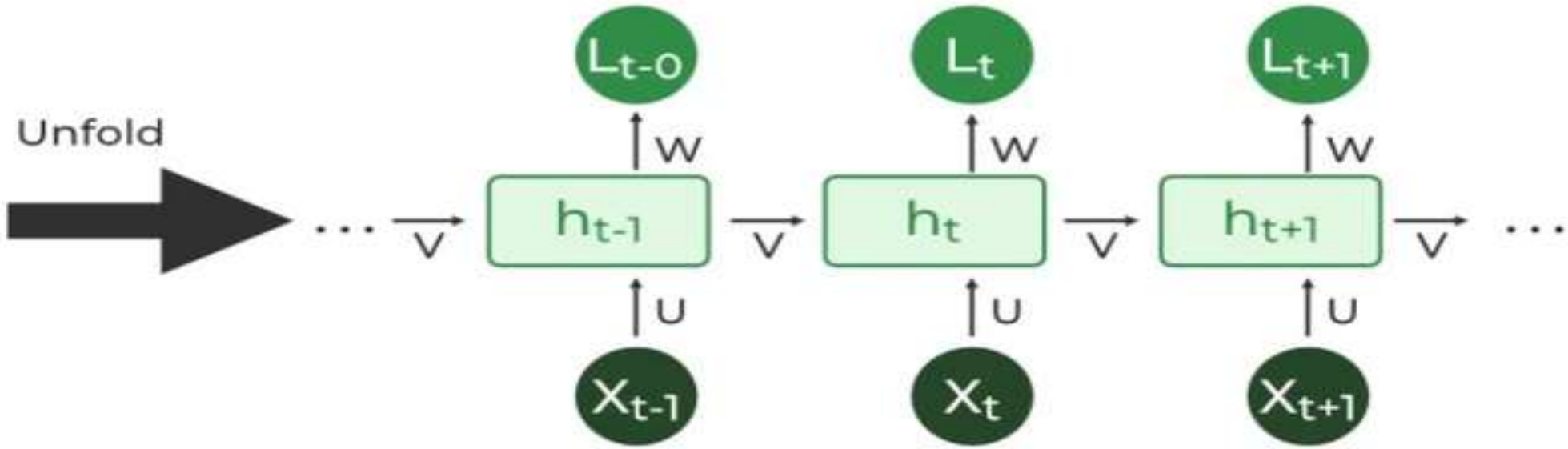
The fundamental processing unit in a Recurrent Neural Network (RNN) is a Recurrent Unit, which is not explicitly called a “Recurrent Neuron.”

This unit has the unique ability to maintain a hidden state, allowing the network to capture sequential dependencies by remembering previous inputs while processing.

Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) versions improve the RNN’s ability to handle long-term dependencies.

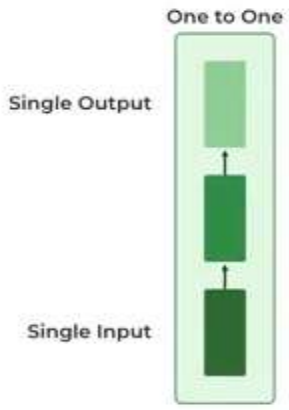


Recurrent Neuron



RNN Unfolding

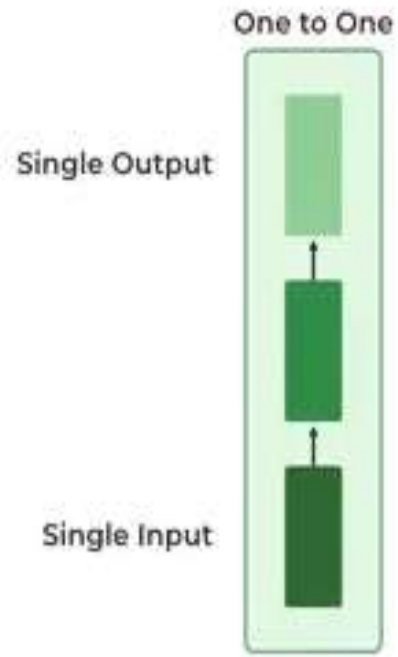
- **Types Of RNN**
- There are four types of RNNs based on the number of inputs and outputs in the network.
- One to One
- One to Many
- Many to One
- Many to Many
- **One to One**
- This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output.



One to One RNN

One to One

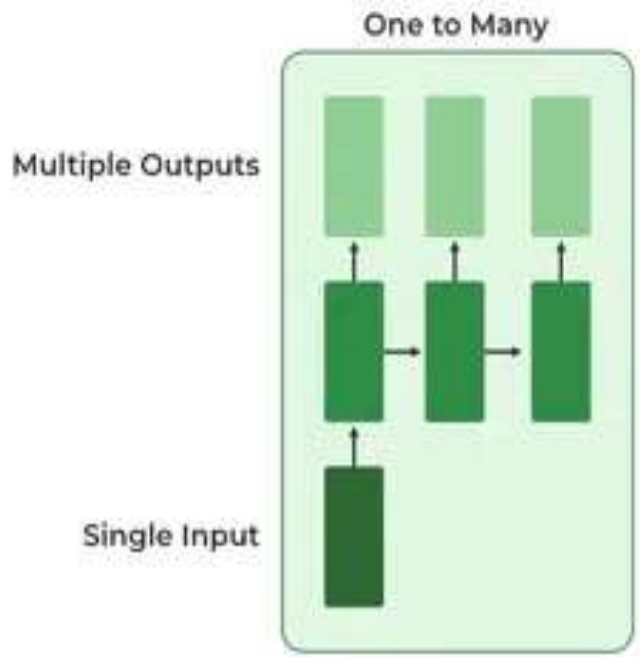
This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output.



One to One RNN

One To Many

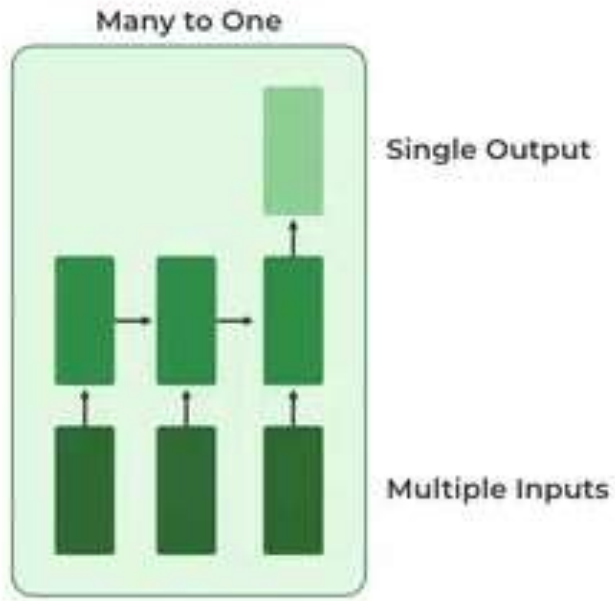
In this type of RNN, there is one input and many outputs associated with it. One of the most used examples of this network is Image captioning where given an image we predict a sentence having Multiple words.



One to Many RNN

Many to One

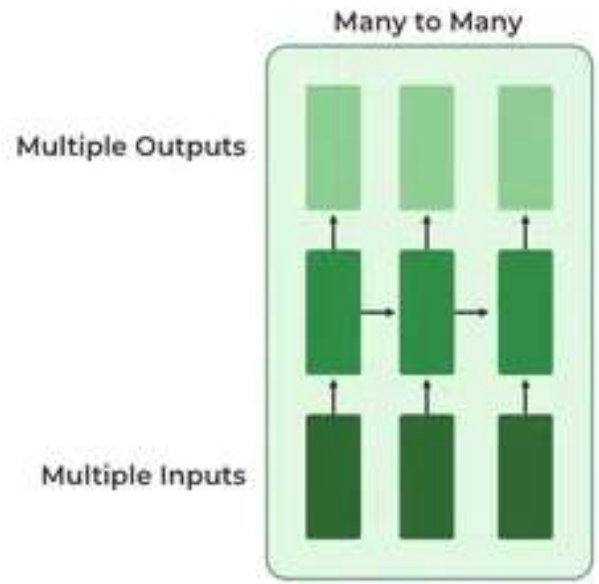
In this type of network, Many inputs are fed to the network at several states of the network generating only one output. This type of network is used in the problems like sentimental analysis. Where we give multiple words as input and predict only the sentiment of the sentence as output.



Many to One RNN

Many to Many

In this type of neural network, there are multiple inputs and multiple outputs corresponding to a problem. One Example of this Problem will be language translation. In language translation, we provide multiple words from one language as input and predict multiple words from the second language as output.



Many to Many RNN

Recurrent Neural Network Architecture

RNNs have the same input and output architecture as any other deep neural architecture. However, differences arise in the way information flows from input to output. Unlike Deep neural networks where we have different weight matrices for each Dense network in RNN, the weight across the network remains the same. It calculates state hidden state H_i for every input X_i . **By using the following formulas:**

$$h = \sigma(UX + Wh_{-1} + B)$$

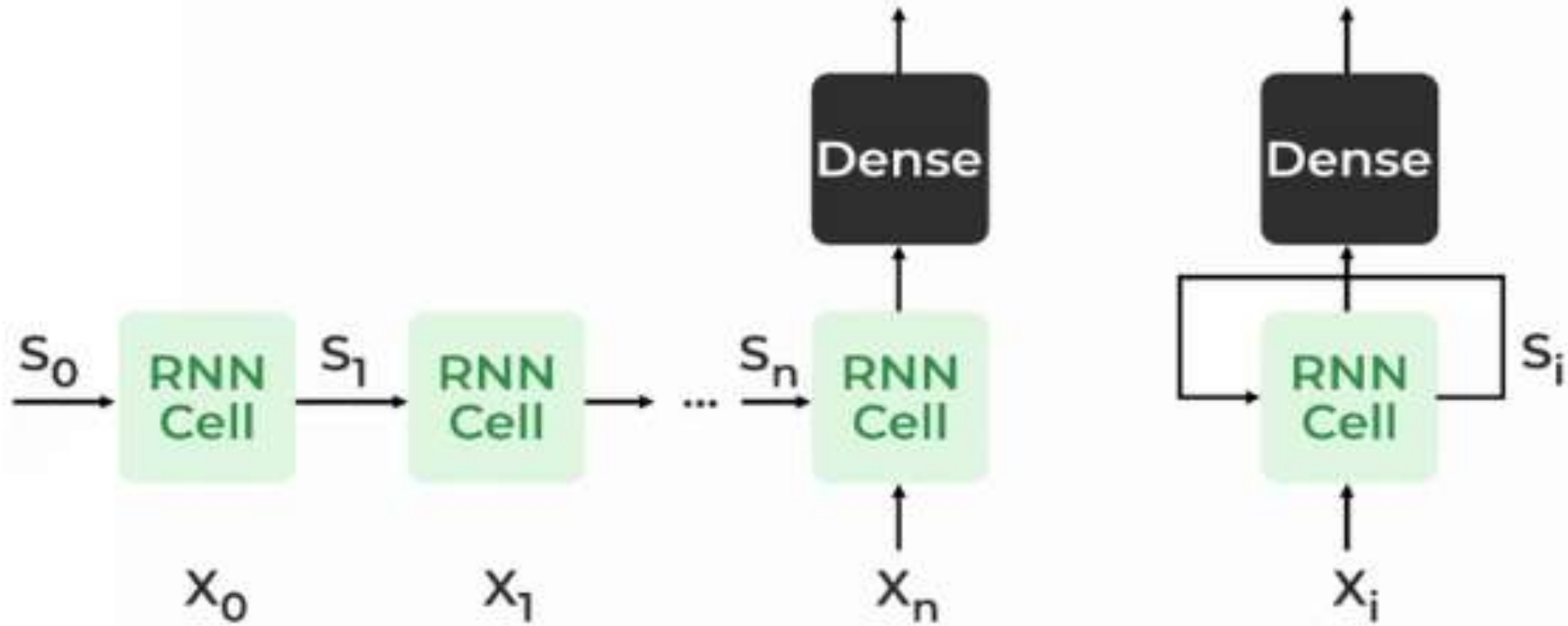
$$Y = O(Vh + C)$$

Hence

$$Y = f(X, h, W, U, V, B, C)$$

Here S is the State matrix which has element si as the state of the network at timestep i
The parameters in the network are W, U, V, c, b which are shared across timestep

RECURRENT NEURAL NETWORKS



Recurrent Neural Architecture

How does RNN work?

The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step. Each unit has an internal state which is called the hidden state of the unit. This hidden state signifies the past knowledge that the network currently holds at a given time step. This hidden state is updated at every time step to signify the change in the knowledge of the network about the past. The hidden state is updated using the following recurrence relation:-

The formula for calculating the current state:

$$h_t = f(h_{t-1}, x_t)$$

where,

- h_t -> current state
- h_{t-1} -> previous state
- x_t -> input state

Formula for applying Activation function(tanh)

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

where,

- w_{hh} -> weight at recurrent neuron
- w_{xh} -> weight at input neuron

The formula for calculating output:

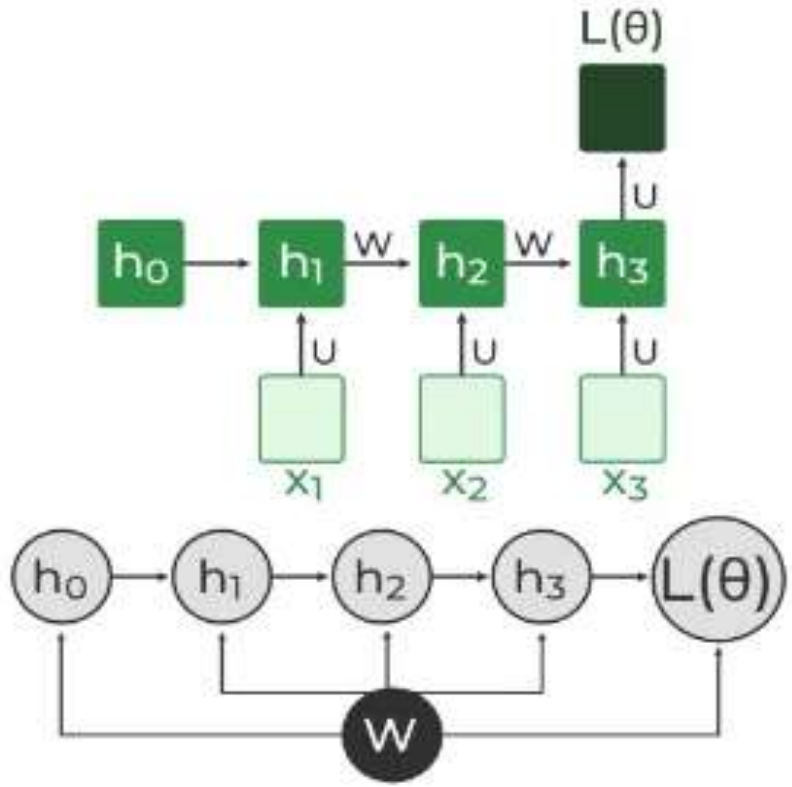
$$y_t = W_{hy}h_t$$

- Y_t -> output
- W_{hy} -> weight at output layer

These parameters are updated using Backpropagation. However, since RNN works on sequential data here we use an updated backpropagation which is known as Backpropagation through time.

Backpropagation Through Time (BPTT)

In RNN the neural network is in an ordered fashion and since in the ordered network each variable is computed one at a time in a specified order like first h1 then h2 then h3 so on. Hence we will apply backpropagation throughout all these hidden time states sequentially.



Backpropagation Through Time (BPTT) In RNN

- L(θ)(loss function) depends on h3
- h3 in turn depends on h2 and W
- h2 in turn depends on h1 and W
- h1 in turn depends on h0 and W
- where h0 is a constant starting state.

$$\frac{\partial \mathbf{L}(\theta)}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathbf{L}(\theta)}{\partial W}$$

For simplicity of this equation, we will apply backpropagation on only one row

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

We already know how to compute this one as it is the same as any simple deep neural network backpropagation.

$$\frac{\partial L(\theta)}{\partial h_3}$$

.However, we will see how to apply backpropagation to this term $\frac{\partial h_3}{\partial W}$

As we know h3 = σ(Wh2 + b)

And In such an ordered network, we can't compute $\frac{\partial h_3}{\partial W}$ by simply treating h3 as a constant because as it also depends on W. the total derivative $\frac{\partial h_3}{\partial W}$ has two parts:

1. **Explicit:** $\frac{\partial h_{3^+}}{\partial W}$ treating all other inputs as constant
2. **Implicit:** Summing over all indirect paths from h_3 to W

Let us see how to do this

$$\begin{aligned}\frac{\partial h_3}{\partial W} &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} \\ &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\left[\frac{\partial h_2^+}{\partial W} + \frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W}\right] \\ &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\left[\frac{\partial h_1^+}{\partial W}\right]\end{aligned}$$

For simplicity, we will short-circuit some of the paths

$$\frac{\partial h_3}{\partial W} = \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2^+}{\partial W} + \frac{\partial h_3}{\partial h_1}\frac{\partial h_1^+}{\partial W}$$

Finally, we have

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \cdot \frac{\partial h_3}{\partial W}$$

Where

$$\frac{\partial h_3}{\partial W} = \sum_{k=1}^{3} \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

Hence,

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \sum_{k=1}^{3} \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

This algorithm is called backpropagation through time (BPTT) as we backpropagate over all

- **Advantages and Disadvantages of Recurrent Neural Network**
- **Advantages**
- An RNN remembers each and every piece of information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
- Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighborhood.
- **Disadvantages**
- Gradient vanishing and exploding problems.
- Training an RNN is a very difficult task.
- It cannot process very long sequences if using tanh or relu as an activation function.
- **Applications of Recurrent Neural Network**
- Language Modelling and Generating Text
- Speech Recognition
- Machine Translation
- Image Recognition, Face detection
- Time series Forecasting

- **Exploding gradient**
- An RNN can wrongly predict the output in the initial training. You need several iterations to adjust the model's parameters to reduce the error rate. You can describe the sensitivity of the error rate corresponding to the model's parameter as a gradient. You can imagine a gradient as a slope that you take to descend from a hill. A steeper gradient enables the model to learn faster, and a shallow gradient decreases the learning rate.
- Exploding gradient happens when the gradient increases exponentially until the RNN becomes unstable. When gradients become infinitely large, the RNN behaves erratically, resulting in performance issues such as overfitting. Overfitting is a phenomenon where the model can predict accurately with training data but can't do the same with real-world data.
- **Vanishing gradient**
- The vanishing gradient problem is a condition where the model's gradient approaches zero in training. When the gradient vanishes, the RNN fails to learn effectively from the training data, resulting in underfitting. An underfit model can't perform well in real-life applications because its weights weren't adjusted appropriately. RNNs are at risk of vanishing and exploding gradient issues when they process long data sequences.
- **Slow training time**
- An RNN processes data sequentially, which limits its ability to process a large number of texts efficiently. For example, an RNN model can analyze a buyer's sentiment from a couple of sentences. However, it requires massive computing power, memory space, and time to summarize a page of an essay.

Recurrent Neural Network	Deep Neural Network
Weights are same across all the layers number of a Recurrent Neural Network	Weights are different for each layer of the network
Recurrent Neural Networks are used when the data is sequential and the number of inputs is not predefined.	A Simple Deep Neural network does not have any special method for sequential data also here the number of inputs is fixed
The Numbers of parameter in the RNN are higher than in simple DNN	The Numbers of Parameter are lower than RNN
Exploding and vanishing gradients is the the major drawback of RNN	These problems also occur in DNN but these are not the major problem with DNN

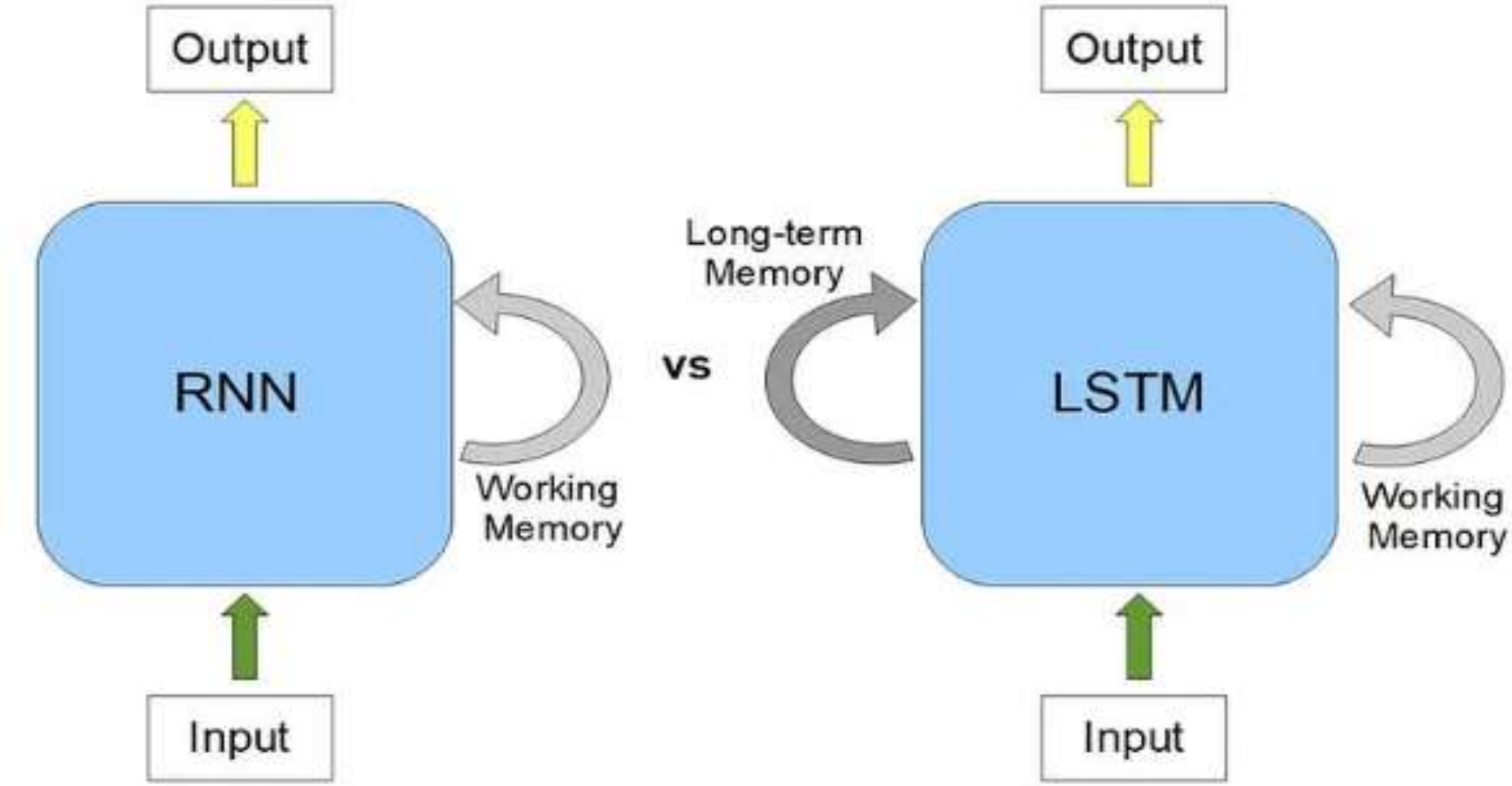
LSTM

> A long short-term memory network is a type of recurrent neural network (RNN).

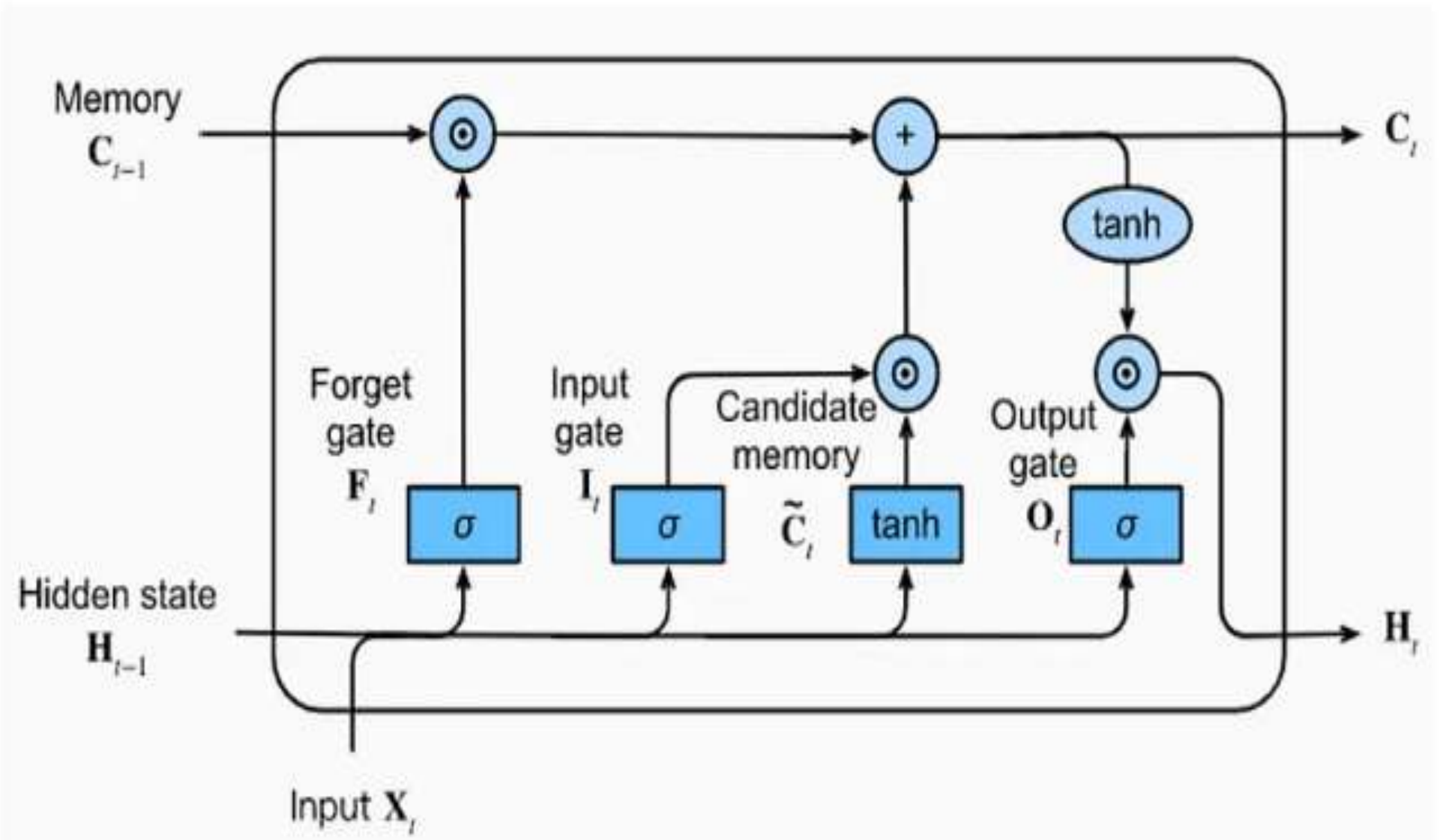
> LSTMs are predominantly used to learn, process, and **classify sequential data** because these networks can learn long-term dependencies between time steps of data.

>applications include sentiment analysis, language modeling, speech recognition, and video analysis, time series forecasting.

Implementation of Long Short-Term Memory(LSTM)



- Depending on the task at hand, RNNs can be utilised for both classification and regression tasks.
- while performing classification tasks, the RNN generates a probability distribution over all potential classes for each input in a sequence.
- In regression tasks, the RNN predicts a continuous value for each input in the sequence.
- However, one major problem to tackle with in the case of RNNs is the vanishing gradient.
- That is when LSTMs come into the picture.
- Long Short-Term Memory(LSTM) is a type of RNN architecture designed to address the vanishing gradient problem in RNNs.
- The vanishing gradient problem occurs when the gradients become too small during backpropagation and cause the weights to stop updating effectively.
- This can lead to poor performance and difficulty in learning long-term dependencies in the data.


Memory
C_{t-1}
C_t
tanh
Forget gate
F_t
Input gate
I_t
Candidate memory
$\tilde{C}_t$
Output gate
O_t
σ
σ
tanh
σ
Hidden state
H_{t-1}
H_t
Input X_t

LSTM

- In LSTM, the concept of memory cells is introduced. These cells can store information over time while also allowing the network to selectively forget/remember certain pieces of information.
- The three gates (input, output, and forget) in each memory cell regulate the flow of data into and out of the cell.
- The quantity of information that can travel through these gates is controlled by sigmoid activation functions.
- The input gate regulates how much new information is permitted into the cell.
- The output gate regulates the amount of information that is output from the cell to the rest of the network.
- And the forget gate regulates how much information is discarded from the cell.
- These gates, along with the memory cell, allow LSTM to learn and remember long-term dependencies in sequential data.

Tanh Function

The tanh function, short for hyperbolic tangent, outputs values between -1 and 1. It's defined mathematically as:

```
tanh(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x))
```

In LSTM networks, the tanh function is primarily used in two places:

1. Cell State Updates: When new information is added to the cell state, tanh helps in scaling the values, ensuring that they remain normalized between -1 and 1. This normalization is crucial for stabilizing the network over time.
2. Output Generation: Before producing the final output, the cell state is passed through a tanh function to normalize it, which is then used to create the final output in conjunction with the output gate.

In LSTMs, the sigmoid function is used in the gates (forget, input, and output gates) for the following reasons:

1. **Decision Making:** Sigmoid's output range of 0 to 1 is perfect for making binary decisions. In the context of LSTM gates, a value close to 0 means "forget this" or "don't let it through," while a value close to 1 means "retain this" or "let it through."

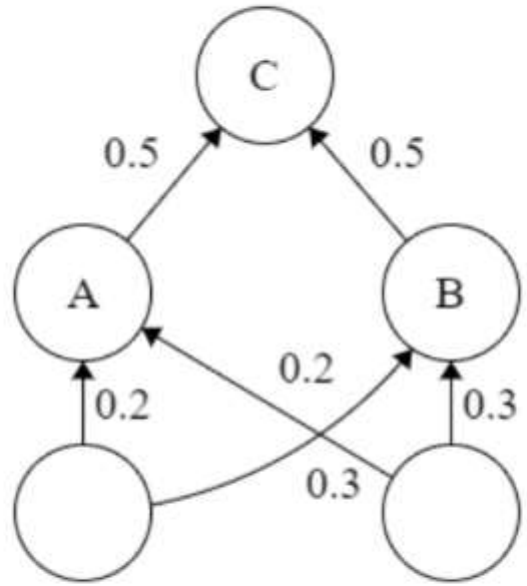
2. **Gate Control:** By outputting values between 0 and 1, the sigmoid function effectively controls how much of the previous state and current input should be allowed to influence the cell state and the final output.

- **Advantages:**
- **Handling Long Sequences:** LSTMs are well-suited for processing sequences of data with long-range dependencies. They can capture information from earlier time steps and remember it for a more extended period, making them effective for tasks like natural language processing (NLP) and time series analysis.
- **Avoiding Vanishing Gradient Problem:** LSTMs address the vanishing gradient problem, which is a common issue in training deep networks, particularly RNNs. The architecture of LSTMs includes gating mechanisms (such as the forget gate) that allow them to control the flow of information and gradients through the network, preventing the gradients from becoming too small during training.
- **Handling Variable-Length Sequences:** LSTMs can handle variable-length input sequences by dynamically adjusting their internal state. This is useful in many real-world applications where the length of the input data varies.
- **Memory Cell:** LSTMs have a memory cell that can store and retrieve information over long sequences. This memory cell allows LSTMs to maintain important information while discarding irrelevant information, making them suitable for tasks that involve remembering past context.
- **Gradient Flow Control:** LSTMs are equipped with mechanisms that allow them to control the flow of gradients during backpropagation. The forget gate, for example, can prevent gradients from vanishing when they need to be propagated back in time. This enables LSTMs to capture information from earlier time steps effectively.

- Disadvantages:
- **Computational Complexity:** LSTMs are computationally more intensive compared to other neural network architectures like feedforward networks or simple RNNs. Training LSTMs can be slower and may require more resources.
- **Overfitting:** Like other deep learning models, LSTMs are susceptible to overfitting when there is insufficient training data. Regularization techniques like dropout can help mitigate this issue.
- **Hyperparameter Tuning:** LSTMs have several hyperparameters to tune, such as the number of LSTM units, the learning rate, and the sequence length. Finding the right set of hyperparameters for a specific problem can be a challenging and time-consuming process.
- **Limited Interpretability:** LSTMs are often considered as “black-box” models, making it challenging to interpret how they arrive at a particular decision. This can be a drawback in applications where interpretability is crucial.
- **Long Training Times:** Training deep LSTM models on large datasets can be time-consuming and may require powerful hardware, such as GPUs or TPUs.

- Despite their disadvantages, LSTMs have been instrumental in many fields, including NLP, speech recognition, and time series forecasting, due to their ability to capture complex sequential patterns.

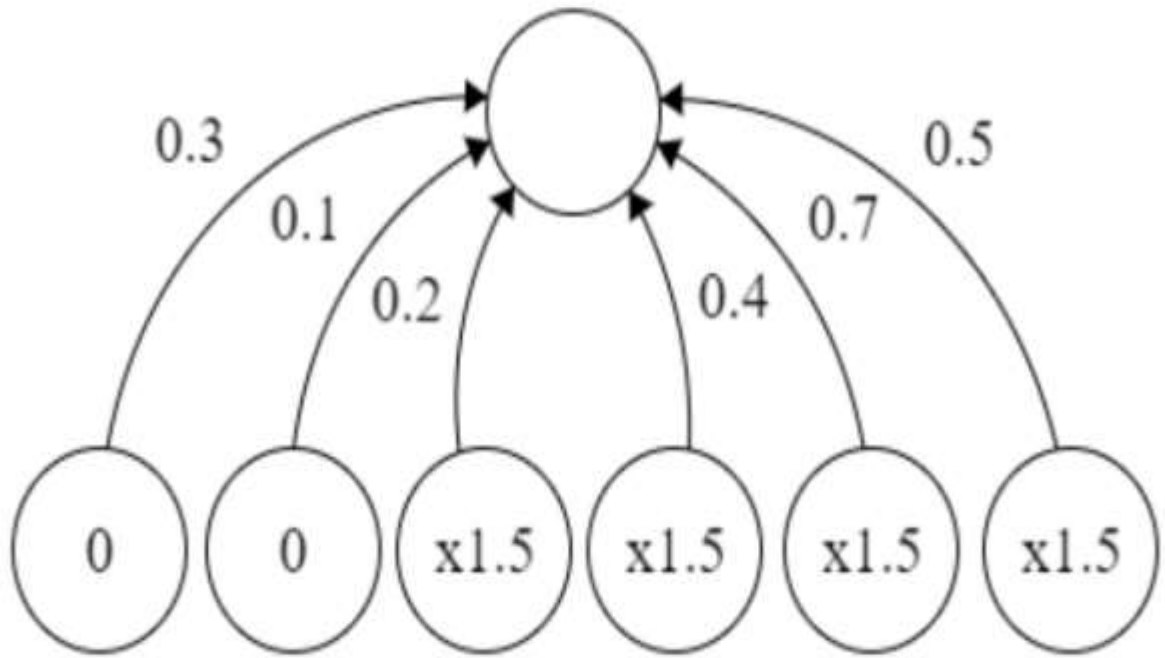
Problem: When a fully-connected layer has a large number of neurons, co-adaptation is more likely to happen. Co-adaptation refers to when multiple neurons in a layer extract the same, or very similar, hidden features from the input data. This can happen when the connection weights for two different neurons are nearly identical.

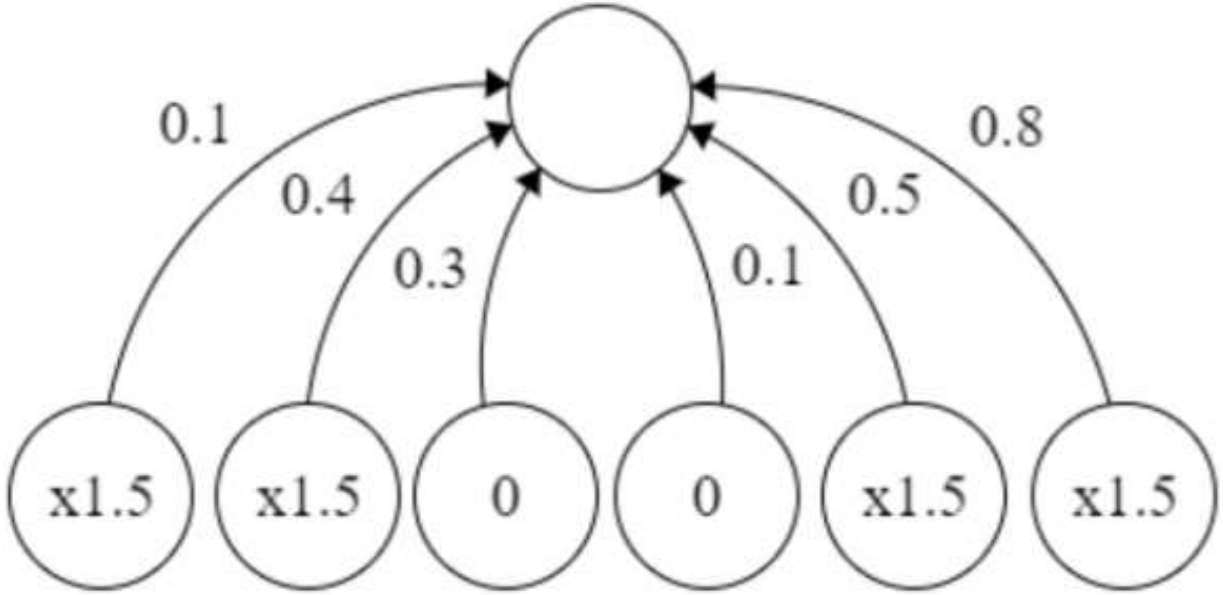


This poses two different problems to our model:

- Wastage of machine's resources when computing the same output.
- If many neurons are extracting the same features, it adds more significance to those features for our model. This leads to overfitting if the duplicate extracted features are specific to only the training set.

Solution to the problem: As the title suggests, we use dropout while training the NN to minimize co-adaptation. In dropout, we randomly shut down some fraction of a layer's neurons at each training step by zeroing out the neuron values. The fraction of neurons to be zeroed out is known as the dropout rate, r_d. The remaining neurons have their values multiplied by $\frac{1}{1-r_d}$ so that the overall sum of the neuron values remains the same.





The two images represent dropout applied to a layer of 6 units, shown at multiple training steps. The dropout rate is 1/3, and the remaining 4 neurons at each training step have their value scaled by x1.5. Thereby, we are choosing a random sample of neurons rather than training the whole network at once. This ensures that the co-adaptation is solved and they learn the hidden features better.

Why dropout works?

•By using dropout, in every iteration, you will work on a smaller neural network than the previous one and therefore, it approaches regularization.

•Dropout helps in shrinking the squared norm of the weights and this tends to a reduction in overfitting.

- **Deep Recurrent Neural Networks**
- we stack the RNNs on top of each other.
- Given a sequence of length T, the first RNN produces a sequence of outputs, also of length T.
- These, in turn, constitute the inputs to the next RNN layer.
- we illustrate a deep RNN with L hidden layers.
- Each hidden state operates on a sequential input and produces a sequential output.
- Moreover, any RNN cell (white box) at each time step depends on both the same layer's value at the previous time step and the previous layer's value at the same time step.

- Deep RNN is a type of computer program that can learn to recognize patterns in data that occur in a sequence, like words in a sentence or musical notes in a song.
- It works by processing information in layers, building up a more complete understanding of the data with each layer.
- This helps it capture complex relationships between the different pieces of information and make better predictions about what might come next.
- Deep RNNs are used in many real-life applications, such as speech recognition systems like Siri or Alexa, language translation software, and even self-driving cars.
- They’re particularly useful in situations where there’s a lot of sequential data to process, like when you’re trying to teach a computer to understand human language.
- Deep RNNs, with their ability to handle sequential data and capture complex relationships between input and output sequences, have become a powerful tool in various real-life applications, ranging from speech recognition and natural language processing to music generation and autonomous driving.

- Deep RNNs have been successfully applied in various real-life applications. Here are a few examples:
- Speech Recognition: Deep RNNs have been used to build speech recognition systems, such as Google's Speech API, Amazon's Alexa, and Apple's Siri. These systems use deep RNNs to convert speech signals into text.
- Natural Language Processing (NLP): Deep RNNs are used in various NLP applications, such as language translation, sentiment analysis, and text classification. For example, Google Translate uses a deep RNN to translate text from one language to another.
- Music Generation: Deep RNNs have been used to generate music, such as Magenta's MusicVAE, which uses a deep RNN to generate melodies and harmonies.
- Image Captioning: Deep RNNs are used in image captioning systems, such as Google's Show and Tell, which uses a deep RNN to generate captions for images.
- Autonomous Driving: Deep RNNs have been used in autonomous driving systems to predict the behaviour of other vehicles on the road, such as the work done by Waymo.

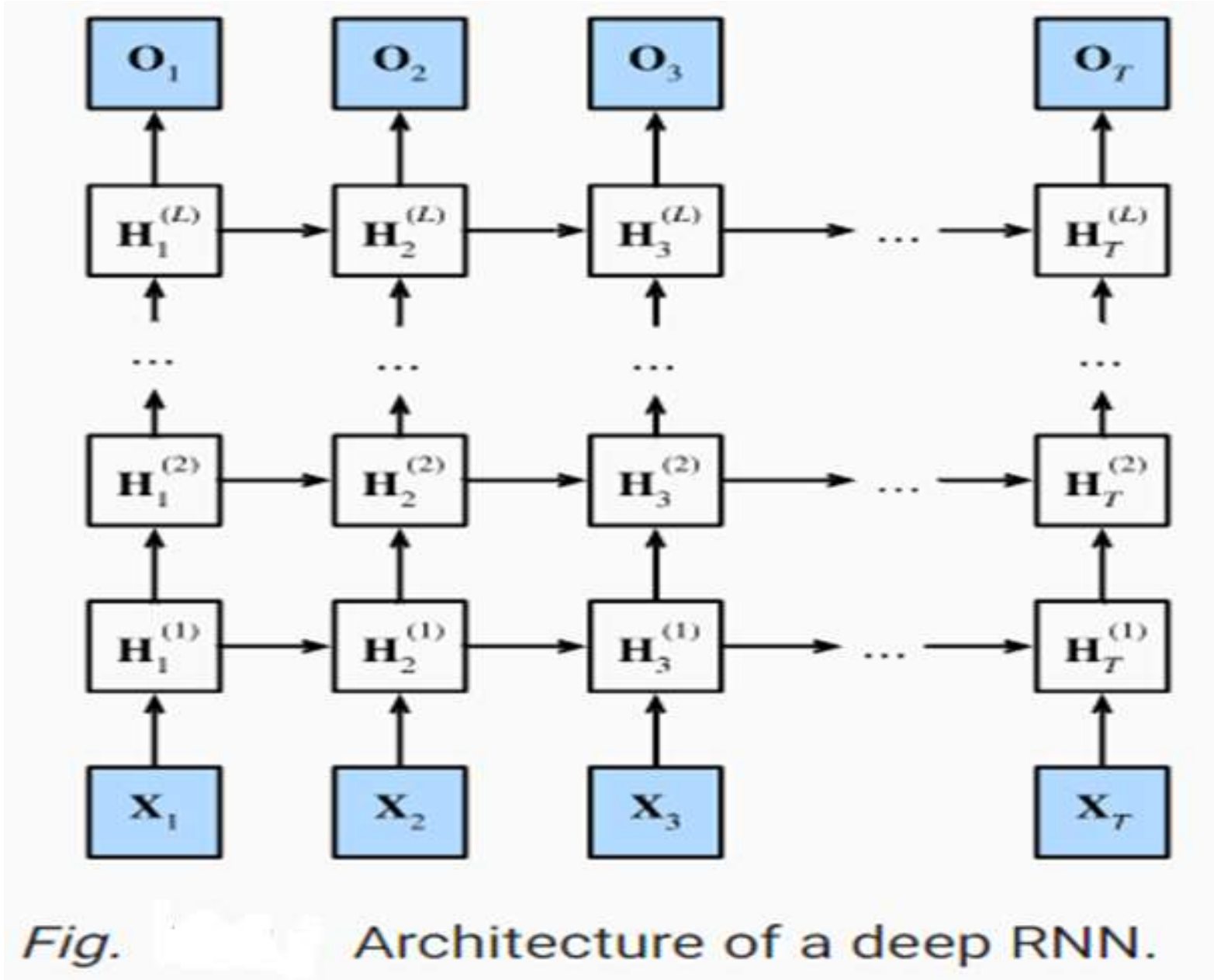


Fig. Architecture of a deep RNN.

Formally, suppose that we have a minibatch input $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ (number of examples $= n$; number of inputs in each example $= d$) at time step t. At the same time step, let the hidden state of the l^{th} hidden layer ($l = 1, \ldots, L$) be $\mathbf{H}_t^{(l)} \in \mathbb{R}^{n \times h}$ (number of hidden units $= h$) and the output layer variable be $\mathbf{O}_t \in \mathbb{R}^{n \times q}$ (number of outputs: q). Setting $\mathbf{H}_t^{(0)} = \mathbf{X}_t$, the hidden state of the l^{th} hidden layer that uses the activation function ϕ_l is calculated as follows:

$$\mathbf{H}_t^{(l)} = \phi_l(\mathbf{H}_t^{(l-1)}\mathbf{W}_{\text{xh}}^{(l)} + \mathbf{H}_{t-1}^{(l)}\mathbf{W}_{\text{hh}}^{(l)} + \mathbf{b}_{\text{h}}^{(l)}),$$

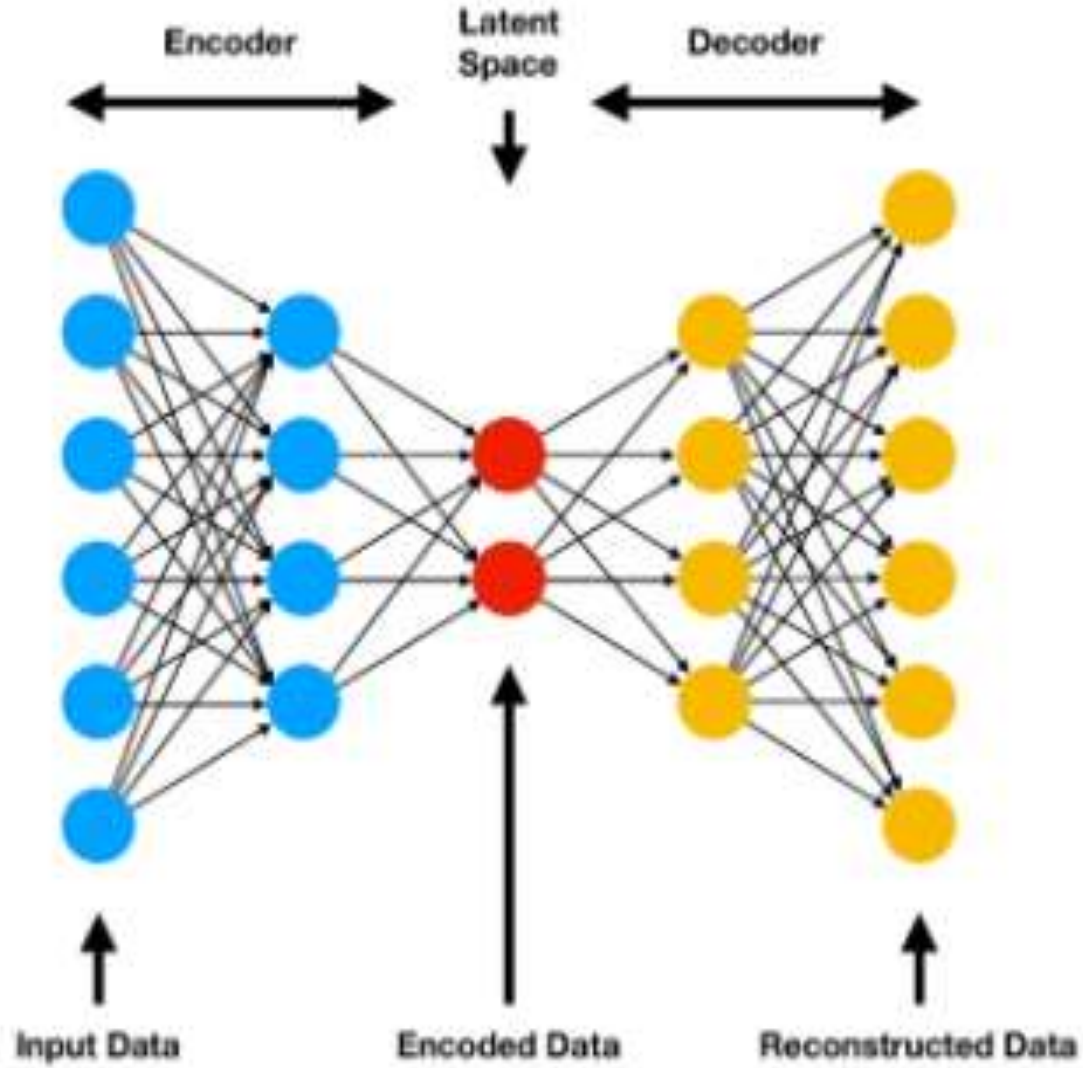
where the weights $\mathbf{W}_{\text{xh}}^{(l)} \in \mathbb{R}^{h \times h}$ and $\mathbf{W}_{\text{hh}}^{(l)} \in \mathbb{R}^{h \times h}$, together with the bias $\mathbf{b}_{\text{h}}^{(l)} \in \mathbb{R}^{1 \times h}$, are the model parameters of the l^{th} hidden layer.

At the end, the calculation of the output layer is only based on the hidden state of the final L^{th} hidden layer:

$$\mathbf{O}_t = \mathbf{H}_t^{(L)}\mathbf{W}_{\text{hq}} + \mathbf{b}_{\text{q}},$$

where the weight $\mathbf{W}_{\text{hq}} \in \mathbb{R}^{h \times q}$ and the bias $\mathbf{b}_{\text{q}} \in \mathbb{R}^{1 \times q}$ are the model parameters of the output layer.

- **What are Autoencoders?**
- ***"Autoencoders are a type of artificial neural network used in unsupervised machine learning and deep learning. They are primarily used for dimensionality reduction, feature learning, and data compression tasks."***
- Autoencoders are a kind of artificial neural network that is utilized for unsupervised learning in order to create effective data representations.
- The objective of an autoencoder is to acquire an encoding for a dataset, often used for reducing dimensionality, cleaning data, or learning features.
- **This is how an autoencoder operates:**
- **Encoding**: The input information undergoes encoding by a neural network, which condenses the information into a latent-space representation that is typically of a smaller dimension compared to the initial input.
- **Latent Representation**: The most crucial characteristics of the input data are captured by the latent representation. It is a compact, concentrated depiction that ideally retains the most important information.
- **Decoding**: The hidden representation is later fed through a decoder neural network that tries to recreate the initial input data from the hidden representation.



Basic architecture of autoencoders

- Autoencoders are a type of artificial neural network used primarily for unsupervised learning tasks. Here are some key features and characteristics of autoencoders:
- **Architecture**:
  - **Encoder**: Compresses the input into a lower-dimensional representation (latent space).
  - **Decoder**: Reconstructs the input from the compressed representation.
- **Dimensionality Reduction**: Autoencoders can reduce the dimensionality of data while preserving essential features, making them useful for tasks like data compression and noise reduction.
- **Reconstruction Loss**: They typically use a loss function (like Mean Squared Error) to measure the difference between the input and the reconstructed output, guiding the learning process.
- **Variability**: There are several types of autoencoders, including:
  - **Denoising Autoencoders, Sparse Autoencoders, Variational Autoencoders (VAEs)**.
- **Training**: Autoencoders are usually trained using backpropagation, similar to other neural networks, adjusting weights to minimize reconstruction loss.
- **Latent Space Visualization**: The compressed representations can often be visualized, helping in understanding data distributions and relationships.
- **Transfer Learning**: Pre-trained autoencoders can be fine-tuned on related tasks, leveraging learned representations.
- **Generative Capabilities**: Variational autoencoders, in particular, can generate new data samples that resemble the training data, making them valuable for tasks like image generation.
- **Applications**: Common applications include image denoising, anomaly detection, data compression, and as feature extractors for supervised tasks.
- **Regularization**: Techniques like dropout or weight decay can be incorporated to prevent overfitting and improve generalization.

- **Types of Autoencoders**
- are as follows:
- **Vanilla Autoencoder**
- Vanilla Autoencoder is a simple yet powerful framework for unsupervised learning tasks.
- It comprises of two primary components: an encoder and a decoder. Together, these components function together compress input data into a lower-dimensional representation and then reconstruct it.
- Training a Vanilla Autoencoder involves minimizing the reconstruction error, which quantifies the disparity between the input data and the reconstructed output.
- This is typically achieved through a process known as backpropagation, where gradients of the reconstruction error with respect to the model parameters are computed and used to update the weights of the neural network layers.
- The objective is to optimize the parameters such that the reconstructed output closely matches the original input.

- **Applications of Vanilla Autoencoders:**
- **Data Compression:** By learning a compressed representation of the input data, Vanilla Autoencoders excel in data compression tasks, facilitating efficient storage and transmission of information.
- **Feature Learning:** Vanilla Autoencoders are adept at capturing salient features of the input data, making them valuable for feature learning tasks in domains such as computer vision, natural language processing, and sensor data analysis.
- **Anomaly Detection:** The ability of Vanilla Autoencoders to reconstruct input data accurately makes them well-suited for anomaly detection tasks. Deviations between the original input and the reconstructed output can signal anomalies or outliers in the data.

Vanilla Autoencoder

Advantages:

1.Simplicity:
Easy to understand and implement, making it a great starting point for learning about autoencoders and neural networks.

1.Unsupervised Learning:
Requires no labeled data, which is beneficial for tasks where labeling is expensive or impractical.

1.Dimensionality Reduction:
Effective for reducing the dimensionality of data, which can help in tasks like data compression or feature extraction.

1.Feature Learning:
Learns useful representations of the input data that can be utilized in downstream tasks, such as classification or clustering.

1.Flexible Architecture:
Can be adapted with different architectures (e.g., changing the number of layers or neurons) to suit specific needs

Disadvantages:

Overfitting:
Prone to memorizing the training data instead of learning generalizable features, especially with small datasets.

Limited Generative Capability:
Does not generate diverse new samples effectively, as it primarily focuses on reconstruction.

Inefficient Representation:
May not learn optimal representations, especially for complex data structures, leading to subpar performance compared to more advanced autoencoders (like VAEs or GANs).

Sensitive to Hyperparameters:
Performance can be highly dependent on the choice of hyperparameters (e.g., learning rate, number of layers, size of the latent space), which may require extensive tuning.

Loss Function Limitations:
Typically uses simple loss functions (like Mean Squared Error), which may not capture complex relationships in the data, leading to less effective learning.

- **Sparse Autoencoder**
- In neural networks, Sparse Autoencoders emerge as a remarkable variant, distinguished by their emphasis on sparsity within the latent representation.
- While traditional autoencoders aim to faithfully reconstruct input data in a lower-dimensional space,
- Sparse Autoencoders introduce constraints that encourage only a small subset of neurons to activate, leading to a sparse and efficient representation.
- **Regularization:** Introducing penalties or constraints on the activation of neurons in the latent space encourages sparsity. Common regularization techniques include L1 regularization, which penalizes the absolute values of the weights, and dropout, which randomly deactivates neurons during training.
- **Kullback-Leibler (KL) Divergence:** KL divergence is a measure of the difference between two probability distributions. In the context of Sparse Autoencoders, it can be used as an additional loss term to encourage the latent distribution to match a predefined sparse target distribution.

- **Sparse Autoencoder**
- **Advantages:**
- **Sparse Representation**:
  - Encourages sparsity in the hidden layer, which can lead to more interpretable and meaningful features, highlighting only the most important aspects of the input data.
- **Improved Generalization**:
  - Reduces the risk of overfitting by limiting the number of active neurons during training, helping the model to learn more robust features.
- **Feature Extraction**:
  - Effectively learns a compact representation of the input data, which can be useful for various tasks like classification or clustering.
- **Flexibility**:
  - Can be combined with other techniques (e.g., denoising or convolutional layers) to enhance its performance in specific applications.
- **Reduced Redundancy**:
  - Promotes a representation that reduces redundancy, capturing only essential information from the data.
- **Disadvantages:**
- **Training Complexity**:
  - Requires careful tuning of the sparsity parameter, which can complicate the training process and may necessitate extensive experimentation.
- **Sensitivity to Hyperparameters**:
  - Performance can significantly depend on the choice of hyperparameters, such as the sparsity level and learning rate, making it challenging to optimize.
- **Potential for Incomplete Learning**:
  - If sparsity constraints are too strong, the model may fail to capture important features, leading to underfitting.
- **Computational Cost**:
  - Depending on the architecture, sparse autoencoders can be computationally intensive, especially when implementing sparsity constraints.
- **Limited Interpretability**:
  - While the sparse representations can be interpretable, understanding the specific contributions of individual neurons can still be complex.

- **Denoising Autoencoder**
- Denoising Autoencoders emerge as a formidable solution for handling noisy input data.
- Unlike Denoising Autoencoders offer a powerful solution for handling noisy input data, enabling robust feature learning and data reconstruction in the presence of noise.
- By intentionally corrupting input data with noise and training the autoencoder to recover the clean underlying representation, Denoising Autoencoders effectively filter out noise and enhance the quality of reconstructed data.
- Their applications span diverse domains, from image and signal processing to data preprocessing and beyond, making them invaluable tools in the arsenal of machine learning practitioners striving for robust and reliable solutions in the face of noisy data.
- Training Denoising Autoencoders involves optimizing the model parameters to minimize the reconstruction error between the clean input data and the output reconstructed by the decoder.
- However, since the input data is intentionally corrupted during training, the autoencoder learns to filter out the noise and recover the underlying clean representation. This process encourages the autoencoder to focus on capturing meaningful features while disregarding the noise present in the input data.

- **Applications of Denoising Autoencoders:**
- **Image Denoising:** In computer vision tasks, Denoising Autoencoders are used to remove noise from images, enhancing image quality and improving the performance of subsequent image processing algorithms.
- **Signal Denoising:** In signal processing applications such as audio processing and sensor data analysis, Denoising Autoencoders can effectively filter out noise from signals, improving the accuracy of signal detection and analysis.
- **Data Preprocessing:** Denoising Autoencoders can be employed as a preprocessing step in machine learning pipelines to clean and denoise input data before feeding it into downstream models.
- This helps improve the robustness and generalization performance of the overall system.

- **Denoising Autoencoder**
- **Advantages:**
- **Robustness to Noise**:
  - Trains the model to reconstruct clean data from noisy inputs, making it effective in real-world applications where noise is prevalent.
- **Improved Feature Learning**:
  - By focusing on learning to extract relevant features from corrupted data, denoising autoencoders can develop more robust and informative representations.
- **Better Generalization**:
  - The noise introduced during training can help the model generalize better to unseen data, as it learns to ignore irrelevant variations.
- **Data Augmentation**:
  - Acts as a form of data augmentation, as the model is trained on various noisy versions of the input data, which can improve performance on downstream tasks.
- **Versatile Applications**:
  - Useful in various domains such as image denoising, speech enhancement, and any task that benefits from noise robustness.
- **Disadvantages:**
- **Increased Complexity**:
  - Training requires careful selection of noise types and levels, which can complicate the design and training process.
- **Computational Resources**:
  - May require more computational power and time compared to standard autoencoders due to the additional complexity of denoising tasks.
- **Limited Performance on Clean Data**:
  - Performance may be less optimal when directly applied to clean data without noise, as the model is specifically tuned to handle corrupted inputs.
- **Sensitivity to Noise Levels**:
  - If the noise is too strong or not representative of real-world conditions, the model may struggle to learn effectively, leading to poor reconstruction.
- **Hyperparameter Tuning**:
  - Performance can be sensitive to the choice of hyperparameters, including the type and amount of noise, necessitating careful tuning.

- **Undercomplete Autoencoder**
- Undercomplete Autoencoders are a fascinating subset of autoencoder models that focus on reducing the dimensionality of the input data.
- Unlike traditional autoencoders, which aim to match the input and output, undercomplete autoencoders intentionally constrain the size of the hidden layer to be smaller than the input layer.
- This constraint forces the model to learn a compressed representation of the input data, capturing only its most essential features.
- Training undercomplete autoencoders involves optimizing the model parameters to minimize the reconstruction error between the input and output data.
- The objective is to learn a compressed representation of the input data that captures its essential features while discarding redundant information.
- This is typically achieved through backpropagation and gradient descent, where the weights of the neural network layers are adjusted iteratively to minimize the reconstruction error.

- **Applications of Undercomplete Autoencoders:**
- **Anomaly Detection:** The compressed representation learned by undercomplete autoencoders can highlight anomalies or outliers in the data by capturing deviations from the normal patterns.
- **Feature Learning:** By focusing on the most important features of the input data, undercomplete autoencoders aid in feature learning tasks, facilitating subsequent analysis and classification tasks.
- **Data Compression:** The compressed representation generated by undercomplete autoencoders serves as an efficient encoding of the input data, enabling data compression and storage optimization.

- **Undercomplete Autoencoder**
- **Advantages:**
- **Dimensionality Reduction**:
  - Forces the model to learn a compact representation of the input data, effectively reducing dimensionality and emphasizing essential features.
- **Regularization Effect**:
  - The constraint of having fewer neurons in the hidden layer helps prevent overfitting, encouraging the model to generalize better to unseen data.
- **Feature Learning**:
  - Promotes the discovery of useful patterns and structures in the data, making it valuable for tasks like feature extraction.
- **Interpretability**:
  - The reduced representation can often be more interpretable, as it condenses information into fewer dimensions that can highlight key characteristics.
- **Simplicity**:
  - The architecture is straightforward, making it easy to implement and understand, serving as a good starting point for experimentation.
- **Disadvantages:**
- **Information Loss**:
  - The undercomplete nature can lead to the loss of important information, particularly if the latent space is too small to capture the complexity of the data.
- **Potential Underfitting**:
  - If the dimensionality reduction is too aggressive, the model may fail to learn adequately, resulting in underfitting and poor reconstruction quality.
- **Training Challenges**:
  - May require careful tuning of hyperparameters (like the size of the latent space) to find the right balance between compression and reconstruction fidelity.
- **Limited Generative Capabilities**:
  - Unlike some other types of autoencoders (like Variational Autoencoders), undercomplete autoencoders are not typically designed for generating new data samples.
- **Dependence on Architecture**:
  - The effectiveness of an undercomplete autoencoder can heavily depend on the specific architecture choices, such as the number of layers and the activation functions used

- **Contractive Autoencoder**
- Contractive Autoencoders are a specialized type of autoencoder designed to learn robust and stable representations of input data.
- Unlike traditional autoencoders that focus solely on reconstructing input data, contractive autoencoders incorporate an additional regularization term in the training objective.
- This term penalizes the model for producing representations that are sensitive to small variations in the input data, encouraging the learning of more stable and invariant features.
- Training contractive autoencoders involves optimizing the model parameters to minimize both the reconstruction error and the additional regularization term.
- The regularization term penalizes the model for producing representations that are sensitive to small changes in the input data, encouraging the learning of stable and invariant features.
- This is typically achieved through backpropagation and gradient descent, where the weights of the neural network layers are adjusted iteratively to minimize the combined loss function.

- **Applications of Contractive Autoencoders:**
- **Representation Learning:** Contractive autoencoders excel in learning stable and invariant representations of input data, making them valuable for tasks such as classification and clustering.
- **Transfer Learning:** The stable representations learned by contractive autoencoders can be transferred to downstream tasks, where they serve as effective feature vectors for tasks with limited labeled data.
- **Data Augmentation:** Contractive autoencoders can be used to generate augmented data by perturbing the input data and reconstructing it with stable representations, thereby increasing the diversity of the training dataset.

- **Contractive Autoencoder**
- **Advantages:**
- **Robust Feature Learning**:
  - Encourages the model to learn features that are invariant to small perturbations in the input, making it effective for noise robustness.
- **Regularization**:
  - The contractive penalty (which encourages the model to keep the latent representations close together) helps prevent overfitting and improves generalization.
- **Useful Representations**:
  - Can capture complex data structures and relationships, making it suitable for various applications such as classification and clustering.
- **Dimensionality Reduction**:
  - Similar to other autoencoders, it can compress data into a lower-dimensional representation while maintaining essential information.
- **Invariance to Input Noise**:
  - By focusing on learning invariant features, it can be beneficial in tasks involving noisy or variable input data.
- **Disadvantages:**
- **Training Complexity**:
  - The addition of the contractive penalty complicates the training process, which may require more careful tuning of hyperparameters.
- **Computational Cost**:
  - Can be more computationally intensive than standard autoencoders due to the need for calculating the Jacobian matrix for the contractive penalty.
- **Potential for Over-regularization**:
  - If the contractive penalty is too strong, it may lead to underfitting by overly constraining the learned representations.
- **Sensitivity to Hyperparameters**:
  - Performance can be highly dependent on the choice of hyperparameters, such as the weight of the contractive penalty, requiring thorough experimentation.
- **Limited Interpretability**:
  - While it learns robust features, understanding the specific contributions of different neurons can be challenging, similar to other complex neural networks.

- **Convolutional Autoencoder**
- Convolutional Autoencoders represent a powerful variant of autoencoder models specifically designed for handling high-dimensional data with spatial structure, such as images.
- Unlike traditional autoencoders, convolutional autoencoders leverage convolutional layers to capture spatial dependencies and hierarchical features within the input data.
- This architecture enables them to efficiently encode and decode complex patterns, making them particularly well-suited for tasks such as image reconstruction, denoising, and feature extraction.
- Training convolutional autoencoders involves optimizing the model parameters to minimize the reconstruction error between the input and output data.
- This is typically achieved through backpropagation and gradient descent, where the weights of the convolutional and fully connected layers are adjusted iteratively to minimize the loss function.
- Common loss functions used include mean squared error (MSE) or binary cross-entropy, depending on the nature of the input data.

- **Applications of Convolutional Autoencoders:**
- **Image Reconstruction:** Convolutional autoencoders excel in reconstructing high-resolution images from compressed latent representations, enabling applications in image compression and restoration.
- **Image Denoising:** By learning robust representations of clean image patches, convolutional autoencoders can effectively remove noise from input images, enhancing image quality and improving the performance of subsequent image processing tasks.
- **Feature Extraction:** Convolutional autoencoders serve as powerful feature extractors, capturing hierarchical features at multiple spatial scales. These features can be used for tasks such as image classification, object detection, and image segmentation.

- **Convolutional Autoencoder**
- **Advantages:**
- **Spatial Hierarchy**:
  - Leverages convolutional layers to capture spatial hierarchies in data, making it especially effective for image data where local patterns and features are crucial.
- **Reduced Parameters**:
  - Convolutional layers share weights, leading to fewer parameters compared to fully connected layers, which reduces the risk of overfitting and improves computational efficiency.
- **Robust Feature Extraction**: Automatically learns relevant features from raw image data without the need for extensive manual feature engineering.
- **Effective Denoising**:
  - Particularly effective for tasks like image denoising, as it can learn to reconstruct images from corrupted inputs by focusing on local structures.
- **Generative Capabilities**: Can be used for generating new images or variations of input images, making them suitable for applications in generative modeling.
- **Disadvantages:**
- **Computationally Intensive**:
  - May require significant computational resources, especially for large datasets or complex architectures, due to the complexity of convolution operations.
- **Training Complexity**: Requires careful tuning of hyperparameters (e.g., number of filters, filter sizes, and stride), which can complicate the training process.
- **Limited Applicability to Non-Image Data**:
  - While powerful for images, they may not perform as well on non-image data types without significant adaptation.
- **Overfitting Risks**:Despite fewer parameters, they can still overfit, particularly if the model is too complex relative to the amount of training data available.
- **Interpreting Features**:
  - While they learn effective features, understanding the specific contributions of the different filters and layers can be challenging.

- **Variational Autoencoder**
- Variational Autoencoders (VAEs) represent a groundbreaking advancement in the field of deep learning, seamlessly integrating probabilistic modeling and neural network architectures.
- Unlike traditional autoencoders, VAEs introduce a probabilistic framework that enables them to not only reconstruct input data but also generate new data samples from a learned latent space.
- This dual capability of reconstruction and generation makes VAEs invaluable for tasks such as image generation, data synthesis, and representation learning.
- Training VAEs involves optimizing the model parameters to maximize the evidence lower bound (ELBO), a lower bound on the log-likelihood of the data.
- The ELBO consists of two terms: the reconstruction loss, which measures the discrepancy between the input data and the reconstructed output, and the KL divergence, which measures the difference between the learned latent distribution and a predefined prior distribution (usually a standard Gaussian).

- **Applications of Variational Autoencoders**
- **Image Generation**: VAEs can generate realistic images by learning the underlying distribution of the training data and sampling from it to create new images. This has applications in creative fields like art and design, as well as in data augmentation for training image recognition models.
- **Anomaly Detection**: VAEs can learn the normal patterns in data and identify anomalies by measuring the reconstruction error. This is useful in various domains such as cybersecurity (detecting unusual network activity), manufacturing (detecting defective products), and finance (detecting fraudulent transactions).
- **Dimensionality Reduction**: VAEs can learn low-dimensional representations of high-dimensional data, which is useful for visualization and downstream tasks such as clustering and classification. They can capture the underlying structure of the data in a compact form, making it easier to analyze and interpret.

- **Variational Autoencoder (VAE)**
- **Advantages:**
- **Generative Modeling**:VAEs can generate new, unseen data samples by sampling from the learned latent space, making them powerful for tasks like image and text generation.
- **Probabilistic Framework**:Provides a probabilistic approach to encoding and decoding data, allowing for the modeling of uncertainty and better capturing of data distributions.
- **Smooth Latent Space**:The latent space is continuous and structured, which facilitates interpolation between data points and makes it easier to generate diverse outputs.
- **Robustness to Overfitting**:The inclusion of a regularization term (Kullback-Leibler divergence) helps prevent overfitting, improving generalization on unseen data.
- **Flexible Architecture**:
  - Can be combined with various neural network architectures (e.g., convolutional or recurrent layers) to adapt to different types of data.
- **Disadvantages:**
- **Complex Training**:Training can be more complex than traditional autoencoders due to the need for variational inference and careful balancing of loss components.
- **Potential for Blurriness**:Generated samples may sometimes be less sharp or detailed than those produced by other generative models (like GANs), particularly in image generation tasks.
- **Sensitivity to Hyperparameters**:Performance can be sensitive to the choice of hyperparameters, such as the weight of the KL divergence term, requiring thorough tuning.
- **Limited Expressiveness**:The assumption of a Gaussian prior may not adequately capture complex data distributions, potentially limiting the model's performance in certain applications.
- **Computational Cost**: Training can be computationally intensive, particularly with larger models or datasets, requiring substantial resources.

- **Autoencoders applications**
- **Image Compression**: Autoencoders are used in image compression techniques, such as JPEG 2000 and WebP. By encoding images into a compact representation and then decoding them back, autoencoders can significantly reduce file sizes while preserving image quality. This is crucial for efficient image storage and transmission in applications like web browsing and video conferencing.
- **Anomaly Detection in Cybersecurity:** Autoencoders are employed to detect abnormal network traffic patterns and security threats. During training, they learn the normal behavior of a network. When presented with new data, deviations from this learned normalcy can be flagged as potential security breaches or anomalies.
- **Healthcare and Medical Imaging:** Autoencoders play a vital role in medical image analysis. They are used for tasks like tumor detection in MRI or CT scans, organ segmentation, and denoising of medical images. Autoencoders can learn to extract relevant features and enhance the diagnostic capabilities of medical imaging systems.
- **Natural Language Processing (NLP):** Variational Autoencoders (VAEs) have applications in generating text and other language-related tasks. For instance, they can be used to generate coherent and contextually meaningful text, making them valuable in chatbots, text summarization, and language translation.
- **Recommendation Systems**: Autoencoders are employed in recommendation engines to learn user and item embeddings. By understanding user preferences and item characteristics in a latent space, they can provide personalized recommendations in e-commerce, streaming services, and content platforms.
- **Autonomous Vehicles:** Autoencoders can be applied in the perception systems of autonomous vehicles. They help process sensor data (e.g., LiDAR, camera) to detect objects, lane markings, and obstacles on the road.
- **Manufacturing Quality Control**: Autoencoders can analyze sensor data from manufacturing processes to identify defects or anomalies in real-time. This ensures product quality and reduces production downtime.
- **Content-Based Image Retrieval**: Autoencoders can be used to create content-based image retrieval systems. By encoding the features of images in a latent space, they enable users to search for visually similar images in large databases.

- Deep Learning Applications
- 1. Virtual Assistants
- Virtual Assistants are cloud-based applications that understand natural language voice commands and complete tasks for the user. Amazon Alexa, Cortana, Siri, and Google Assistant are typical examples of virtual assistants. They need internet-connected devices to work with their full capabilities. Each time a command is fed to the assistant, they tend to provide a better user experience based on past experiences using Deep Learning algorithms.
- 2. Chatbots
- Chatbots can solve customer problems in seconds. A chatbot is an AI application to chat online via text or text-to-speech. It is capable of communicating and performing actions similar to a human. Chatbots are used a lot in customer interaction, marketing on social network sites, and instant messaging the client. It delivers automated responses to user inputs. It uses machine learning and deep learning algorithms to generate different types of reactions.
- The next important deep learning application is related to Healthcare.
- 3. Healthcare
- Deep Learning has found its application in the Healthcare sector. Computer-aided disease detection and computer-aided diagnosis have been possible using Deep Learning. It is widely used for medical research, drug discovery, and diagnosis of life-threatening diseases such as cancer and diabetic retinopathy through the process of medical imaging.

4. Entertainment

Companies such as Netflix, Amazon, YouTube, and Spotify give relevant movies, songs, and video recommendations to enhance their customer experience. This is all thanks to Deep Learning. Based on a person's browsing history, interest, and behavior, online streaming companies give suggestions to help them make product and service choices. Deep learning techniques are also used to add sound to silent movies and generate subtitles automatically.

5. News Aggregation and Fake News Detection

Deep Learning allows you to customize news depending on the readers' persona. You can aggregate and filter out news information as per social, geographical, and economic parameters and the individual preferences of a reader. Neural Networks help develop classifiers that can detect fake and biased news and remove it from your feed. They also warn you of possible privacy breaches.

6. Composing Music

A machine can learn the notes, structures, and patterns of music and start producing music independently. Deep Learning-based generative models such as WaveNet can be used to develop raw audio. Long Short Term Memory Network helps to generate music automatically. Music21 Python toolkit is used for computer-aided musicology. It allows us to train a system to develop music by teaching music theory fundamentals, generating music samples, and studying music.

- 7. Image Coloring
- Image colorization has seen significant advancements using Deep Learning. Image colorization is taking an input of a grayscale image and then producing an output of a colorized image. ChromaGAN is an example of a picture colorization model. A generative network is framed in an adversarial model that learns to colorize by incorporating a perceptual and semantic understanding of both class distributions and color.
- 8. Robotics
- Deep Learning is heavily used for building robots to perform human-like tasks. Robots powered by Deep Learning use real-time updates to sense obstacles in their path and pre-plan their journey instantly. It can be used to carry goods in hospitals, factories, warehouses, inventory management, manufacturing products, etc.
- Boston Dynamics robots react to people when someone pushes them around, they can unload a dishwasher, get up when they fall, and do other tasks as well.
- 9. Image Captioning
- Image Captioning is the method of generating a textual description of an image. It uses computer vision to understand the image's content and a language model to turn the understanding of the image into words in the right order. A recurrent neural network such as an LSTM is used to turn the labels into a coherent sentence. Microsoft has built its caption bot where you can upload an image or the URL of any image, and it will display the textual description of the image. Another such application that suggests a perfect caption and best hashtags for a picture is Caption AI.

- 10. Advertising
- In Advertising, Deep Learning allows optimizing a user's experience. Deep Learning helps publishers and advertisers to increase the significance of the ads and boosts the advertising campaigns. It will enable ad networks to reduce costs by dropping the cost per acquisition of a campaign from $60 to $30. You can create data-driven predictive advertising, real-time bidding of ads, and target display advertising.
- 11. Self Driving Cars
- Deep Learning is the driving force behind the notion of self-driving automobiles that are autonomous. Deep Learning technologies are actually "learning machines" that learn how to act and respond using millions of data sets and training. To diversify its business infrastructure, Uber Artificial Intelligence laboratories are powering additional autonomous cars and developing self-driving cars for on-demand food delivery. Amazon, on the other hand, has delivered their merchandise using drones in select areas of the globe.
- 12. Natural Language Processing
- Another important field where Deep Learning is showing promising results is NLP, or Natural Language Processing. It is the procedure for allowing robots to study and comprehend human language.
- However, keep in mind that human language is excruciatingly difficult for robots to understand. Machines are discouraged from correctly comprehending or creating human language not only because of the alphabet and words, but also because of context, accents, handwriting, and other factors.

- 13. Visual Recognition
- Just assume you're going through your old memories or photographs. You may choose to print some of these. In the lack of metadata, the only method to achieve this was through physical labour. The most you could do was order them by date, but downloaded photographs occasionally lack that metadata. Deep Learning, on the other hand, has made the job easier. Images may be sorted using it based on places recognised in pictures, faces, a mix of individuals, events, dates, and so on. To detect aspects when searching for a certain photo in a library, state-of-the-art visual recognition algorithms with various levels from basic to advanced are required.
- 14. Fraud Detection
- Another attractive application for deep learning is fraud protection and detection; major companies in the payment system sector are already experimenting with it. PayPal, for example, uses predictive analytics technology to detect and prevent fraudulent activity. The business claimed that examining sequences of user behaviour using neural networks' long short-term memory architecture increased anomaly identification by up to 10%. Sustainable fraud detection techniques are essential for every fintech firm, banking app, or insurance platform, as well as any organisation that gathers and uses sensitive data. Deep learning has the ability to make fraud more predictable and hence avoidable.
- 15. Personalisations
- Every platform is now attempting to leverage chatbots to create tailored experiences with a human touch for its users. Deep Learning is assisting e-commerce behemoths such as Amazon, E-Bay, and Alibaba in providing smooth tailored experiences such as product suggestions, customised packaging and discounts, and spotting huge income potential during the holiday season. Even in newer markets, reconnaissance is accomplished by providing goods, offers, or plans that are more likely to appeal to human psychology and contribute to growth in micro markets. Online self-service solutions are on the increase, and dependable procedures are bringing services to the internet that were previously only physically available.

- 16. Detecting Developmental Delay in Children
- Early diagnosis of developmental impairments in children is critical since early intervention improves children's prognoses. Meanwhile, a growing body of research suggests a link between developmental impairment and motor competence, therefore motor skill is taken into account in the early diagnosis of developmental disability. However, because of the lack of professionals and time restrictions, testing motor skills in the diagnosis of the developmental problem is typically done through informal questionnaires or surveys to parents. This is progressively becoming achievable with deep learning technologies. Researchers at MIT's Computer Science and Artificial Intelligence Laboratory and the Institute of Health Professions at Massachusetts General Hospital have created a computer system that can detect language and speech impairments even before kindergarten.
- 17. Colourisation of Black and White images
- The technique of taking grayscale photos in the form of input and creating colourized images for output that represent the semantic colours and tones of the input is known as image colourization. Given the intricacy of the work, this technique was traditionally done by hand using human labour. However, using today's Deep Learning Technology, it is now applied to objects and their context inside the shot - in order to colour the image, in the same way that a human operator would. In order to reproduce the picture with the addition of color, high-quality convolutional neural networks are utilized in supervised layers.
- 18. Adding Sounds to Silent Movies
- In order to make a picture feel more genuine, sound effects that were not captured during production are frequently added. This is referred to as "Foley." Deep learning was used by researchers at the University of Texas to automate this procedure. They trained a neural network on 12 well-known film incidents in which filmmakers commonly used Foley effects. Their neural network identifies the sound to be generated, and they also have a sequential network that produces the sound. They employed neural networks to transition from temporally matched visuals to sound creation, a completely another medium!

- 19. Automatic Machine Translation
- Deep learning has changed several disciplines in recent years. In response to these advancements, the field of Machine Translation has switched to the use of deep-learning neural-based methods, which have supplanted older approaches such as rule-based systems or statistical phrase-based methods. Neural MT (NMT) models can now access the whole information accessible anywhere in the source phrase and automatically learn which piece is important at which step of synthesising the output text, thanks to massive quantities of training data and unparalleled processing power. The elimination of previous independence assumptions is the primary cause for the remarkable improvement in translation quality. This resulted in neural translation closing the quality gap between human and neural translation.
- 20. Automatic Handwriting Generation
- This Deep Learning application includes the creation of a new set of handwriting for a given corpus of a word or phrase. The handwriting is effectively presented as a series of coordinates utilised by a pen to make the samples. The link between pen movement and letter formation is discovered, and additional instances are developed.
- 21. Automatic Game Playing
- A corpus of text is learned here, and fresh text has created word for word or character for character. Using deep learning algorithms, it is possible to learn how to spell, punctuate, and even identify the style of the text in corpus phrases. Large recurrent neural networks are typically employed to learn text production from objects in sequences of input strings. However, LSTM recurrent neural networks have lately shown remarkable success in this challenge by employing a character-based model that creates one character at a time.

- 22. Language Translations
- Machine translation is receiving a lot of attention from technology businesses. This investment, along with recent advances in deep learning, has resulted in significant increases in translation quality. According to Google, transitioning to deep learning resulted in a 60% boost in translation accuracy over the prior phrase-based strategy employed in Google Translate. Google and Microsoft can now translate over 100 different languages with near-human accuracy in several of them.
- 23. Pixel Restoration
- It was impossible to zoom into movies beyond their actual resolution until Deep Learning came along. Researchers at Google Brain created a Deep Learning network in 2017 to take very low-quality photos of faces and guess the person's face from them. Known as Pixel Recursive Super Resolution, this approach uses pixels to achieve super resolution. It dramatically improves photo resolution, highlighting salient characteristics just enough for personality recognition.
- 24. Demographic and Election Predictions
- Gebru et al used 50 million Google Street View pictures to see what a Deep Learning network might accomplish with them. As usual, the outcomes were amazing. The computer learned to detect and pinpoint automobiles and their specs. It was able to identify approximately 22 million automobiles, as well as their make, model, body style, and year. The explorations did not end there, inspired by the success story of these Deep Learning capabilities. The algorithm was shown to be capable of estimating the demographics of each location based just on the automobile makeup.
- 25. Deep Dreaming
- DeepDream is an experiment that visualises neural network taught patterns. DeepDream, like a toddler watching clouds and attempting to decipher random forms, over-interprets and intensifies the patterns it finds in a picture.
- It accomplishes this by sending an image across the network and then calculating the gradient of the picture in relation to the activations of a certain layer. The image is then altered to amplify these activations, improving the patterns perceived by the network and producing a dream-like visual. This method was named "Inceptionism" (a reference to InceptionNet, and the movie Inception).