

MCA 402B: Deep Learning

UNIT I

Basics Of Neural Networks: Basic concept of Neurons – Perceptron Algorithm – Feed Forward and Back Propagation Networks.

UNIT II

Introduction To Deep Learning: :Feed Forward Neural Networks – Gradient Descent – Back Propagation Algorithm – Vanishing Gradient problem – Mitigation – ReLU Heuristics for Avoiding Bad Local Minima – Heuristics for Faster Training – Nestors Accelerated Gradient Descent – Regularization – Dropout.

UNIT III

Convolutional Neural Networks: : CNN Architectures – Convolution – Pooling Layers – Transfer Learning – Image Classification using Transfer Learning

UNIT IV

More Deep Learning Architectures\;LSTM, GRU, Encoder/Decoder Architectures – Autoencoders – Standard- Sparse – Denoising – Contractive- Variational Autoencoders – Adversarial Generative Networks – Autoencoder and DBM

UNIT V

Applications Of Deep Learning: Image Segmentation – Object Detection – Automatic Image Captioning – Image generation with Generative Adversarial Networks – Video to Text with LSTM Models – Attention Models for Computer Vision – Case Study: Named Entity Recognition – Opinion Mining using Recurrent Neural Networks – Parsing and Sentiment Analysis using Recursive Neural Networks – Sentence Classification using Convolutional Neural Networks – Dialogue Generation with LSTMs.

Text Books:

1. Ian Good Fellow, Yoshua Bengio, Aaron Courville, “Deep Learning”, MIT Press, 2017.
2. Navin Kumar Manaswi, “Deep Learning with Applications Using Python”, Apress, 2018.

Reference Books

1. Francois Chollet, “Deep Learning with Python”, Manning Publications, 2018.
2. Phil Kim, “Matlab Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence”, Apress , 2017.
3. Ragav Venkatesan, Baoxin Li, “Convolutional Neural Networks in Visual Computing”, CRC Press, 2018.

Lecture Notes

Unit-1

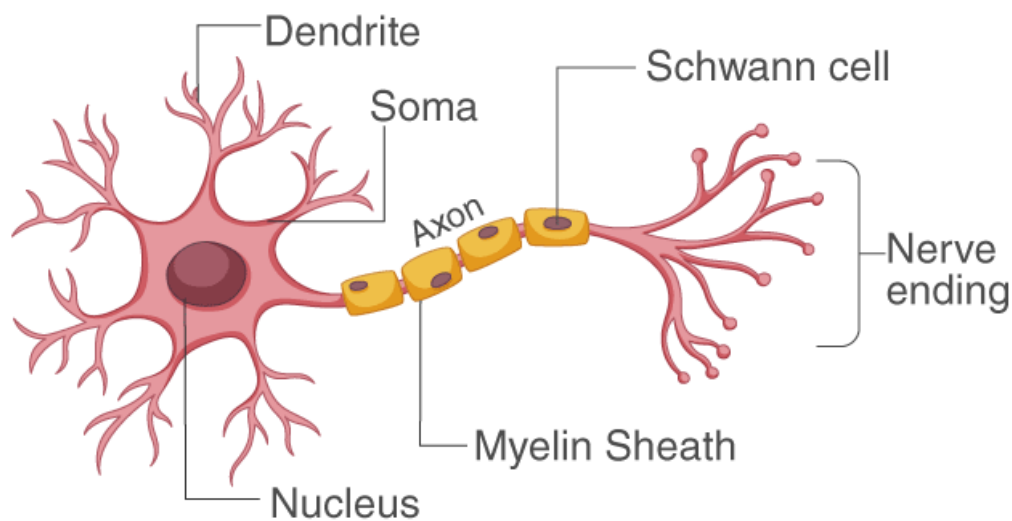
Basic Concept of neurons

Neurons are the building blocks of the nervous system. They receive and transmit signals to different parts of the body. This is carried out in both physical and electrical forms. There are several different types of neurons that facilitate the transmission of information.

The sensory neurons carry information from the sensory receptor cells present throughout the body to the brain. Whereas, the motor neurons transmit information from the brain to the muscles. The interneurons transmit information between different neurons in the body.

Also Read: Nervous System

STRUCTURE OF NEURON



Neuron Structure

A neuron varies in shape and size depending on its function and location. All neurons have three different parts – dendrites, cell body and axon.

Parts of Neuron

Following are the different parts of a neuron:

Dendrites

These are branch-like structures that receive messages from other neurons and allow the transmission of messages to the cell body.

Cell Body

Each neuron has a cell body with a nucleus, Golgi body, endoplasmic reticulum, mitochondria and other components.

Axon

Axon is a tube-like structure that carries electrical impulse from the cell body to the axon terminals that pass the impulse to another neuron.

Synapse

It is the chemical junction between the terminal of one neuron and the dendrites of another neuron.

Also Read: Difference between neurons and neuroglia

Neuron Types

There are three different types of neurons:

Sensory Neurons

The sensory neurons convert signals from the external environment into corresponding internal stimuli. The sensory inputs activate the sensory neurons and carry sensory information to the brain and spinal cord. They are pseudounipolar in structure.

Motor Neurons

These are multipolar and are located in the central nervous system extending their axons outside the central nervous system. This is the most common type of neuron and transmits information from the brain to the muscles of the body.

Interneurons

They are multipolar in structure. Their axons connect only to the nearby sensory and motor neurons. They help in passing signals between two neurons.

Also Read: Nerves

Neuron Functions

The important functions of a neuron are:

Chemical Synapse

In chemical synapses, the action potential affects other neurons through a gap present between two neurons known as the synapse. The action potential is carried along the axon to a postsynaptic ending that initiates the release of chemical messengers known as neurotransmitters. These neurotransmitters excite the postsynaptic neurons that generate an action potential of their own.

Electrical Synapse

When two neurons are connected by a gap junction, it results in an electrical synapse. These gaps include ion channels that help in the direct transmission of a positive electrical signal. These are much faster than chemical synapses.

Perceptron algorithm

As you know a perceptron serves as a basic building block for creating a deep neural network therefore, it is quite obvious that we should begin our journey with perceptron and learn how to implement it using TensorFlow to solve different problems. In case you are completely new to deep learning, I would suggest you to go through of this Deep Learning Tutorial series to avoid any confusion. Following are the topics that will be covered in this blog on Perceptron Learning Algorithm:

- **Perceptron as a Linear Classifier**
- **Implementation of a Perceptron using TensorFlow Library**
- **SONAR Data Classification Using a Single Layer Perceptron**

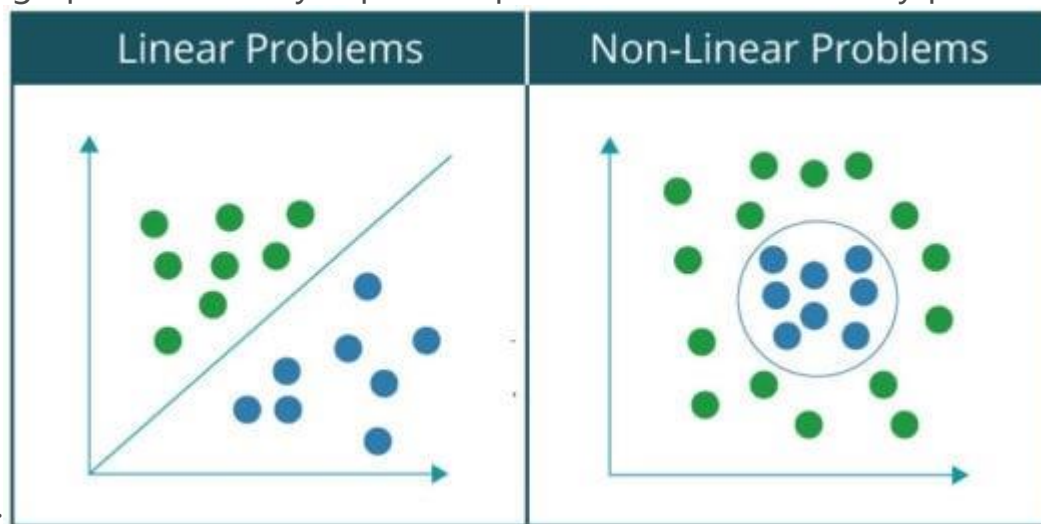
Types of Classification Problems

One can categorize all kinds of classification problems that can be solved using neural networks into two broad categories:

- **Linearly Separable Problems**
- **Non-Linearly Separable Problems**

Basically, a problem is said to be linearly separable if you can classify the data set into two categories or classes using a single line. *For example, separating cats from a group of cats and dogs.* On the contrary, in case of a non-linearly separable problems, the data set contains multiple classes

and requires non-linear line for separating them into their respective classes. For Let us visualize the difference between the two by plotting the graph of a linearly separable problem and non-linearly problem data



set:

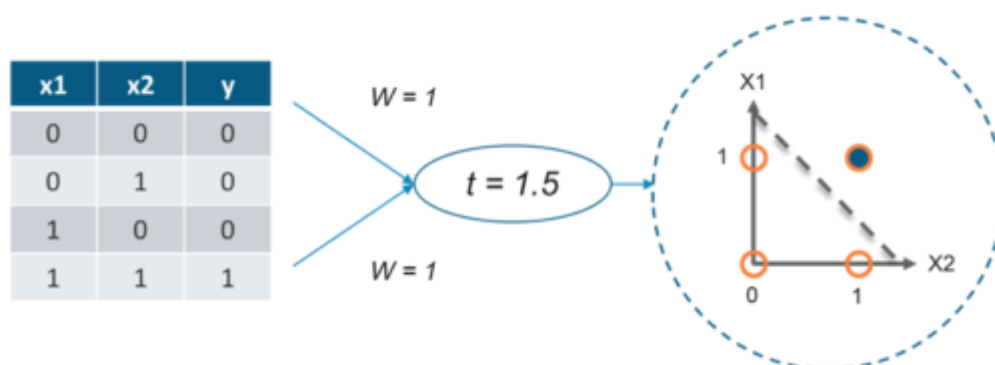
Since, you all are familiar with AND Gates, I will be using it as an example to explain how a perceptron works as a linear classifier.

Perceptron as AND Gate

As you know that AND gate produces an output as 1 if both the inputs are 1 and 0 in all other cases. Therefore, a perceptron can be used as a separator or a decision line that divides the input set of AND Gate, into two classes:

- **Class 1:** Inputs having output as 0 that lies below the decision line.
- **Class 2:** Inputs having output as 1 that lies above the decision line or separator.

The below diagram shows the above idea of classifying the inputs of AND Gate using a perceptron:



Till now, you understood that a linear perceptron can be used to classify the input data set into two classes. But, how does it actually classify the data?

Mathematically, one can represent a perceptron as a function of weights,

The diagram shows the equation $f(x) = w \cdot x + b$. Brackets and arrows point from parts of the equation to labels on the right: a bracket under $f(x)$ points to 'Transfer Function', a bracket under w points to 'Weight Vector', a bracket under x points to 'Input Vector', and a bracket under b points to 'Bias'.

inputs and bias (vertical offset):

- Each of the input received by the perceptron has been weighted based on the amount of its contribution for obtaining the final output.
- Bias allows us to shift the decision line so that it can best separate the inputs into two classes.

Enough of the theory, let us look at the first example of this blog on Perceptron Learning Algorithm where I will implement AND Gate using a perceptron from scratch.

Perceptron Learning Algorithm: Implementation of AND Gate

1. Import all the required library

I will begin with importing all the required libraries. In this case, I need to import one library only i.e. TensorFlow:

Feed Forward and Back Propagation networks

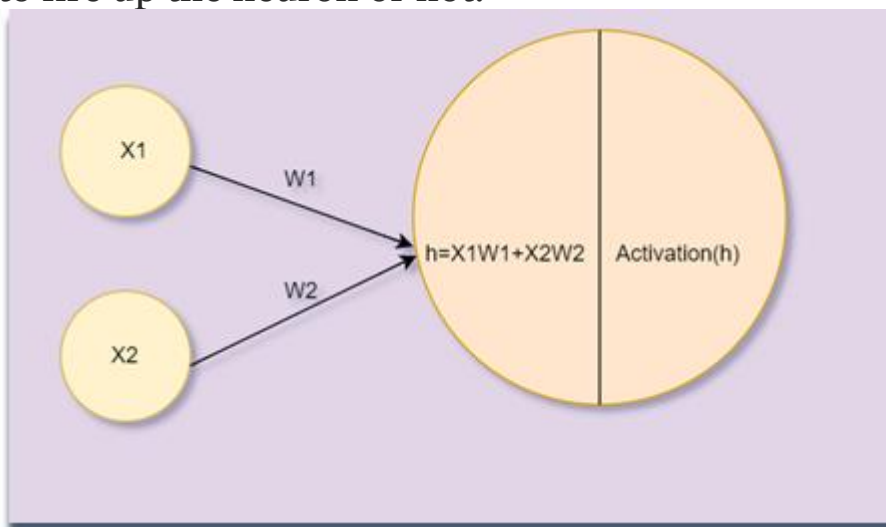
Feed Forward Propagation

In this write up a technical explanation and functioning of a fully connected neural network which involves bi direction flow, first a forward direction known as Feed forward and a backward direction known as back propagation.

Take the below example of a fully connected neural network which has two inputs, one hidden layer with 2 neurons and an output layer where 2 neurons represent the two outputs so it can be deemed as a binary class classification.

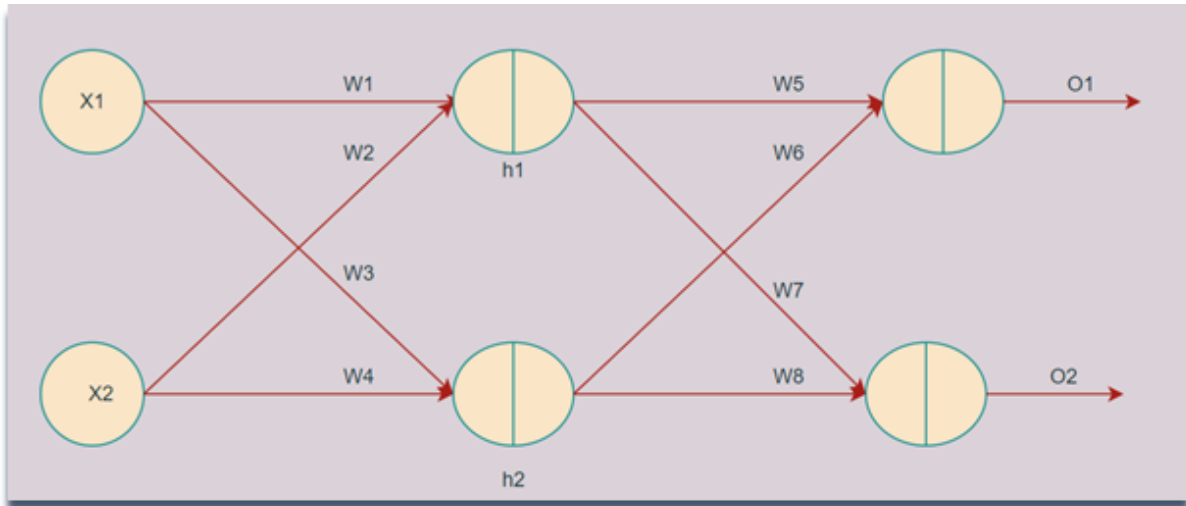
Hidden layer consists of a summation and an activation function to it, activation functions are used to introduce non linearity other than acting as a filter for neurons to pass and the commonly used activation functions are Relu, Tanh and Sigmoid and Softmax.

A typical neuron cell looks like below. As can be seen the initial weights which may be random gets multiplied by the feature vector and gets added up in a neuron, the activation then decides whether to fire up the neuron or not.



A typical neuron with inputs weights and internal assembly containing summation and activation

A fully connected network looks like below where each input is connected to each neuron in the hidden layer by weights associated with each connection. The relation is many to many.



A Fully connected NN with one hidden having two neuron along with output layer

Consider a scenario where the problem is a binary class classification and the expected accuracy is .99 and 0.01 resp for both the classes. You can also try with the label outcome as 1 and 0.

let's have a look below at the assumed values which are required initially for the feed fwd and back prop. The hidden layer activation function is assumed to be sigmoid and the weights are random initially.

X1	X2	W1	W2	W3	W4	W5	W6	W7	W8
.2	.3	.15	.25	.35	.45	.55	.65	.75	.85

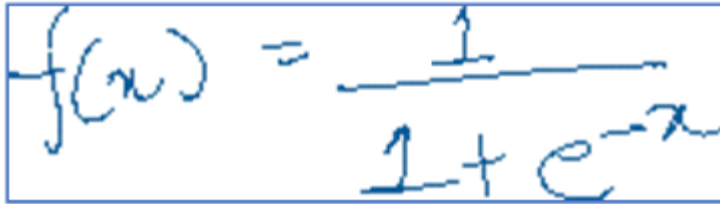
Weight matrix, weights are randomly initialized

The summation equation

$$\sum h1 = X1W1 + X2W2 = 0.2 * 0.15 + 0.3 * 0.25 = 0.105 .$$

$$\sum h2 = X1W3 + X2W4 = 0.2 * 0.35 + 0.3 * 0.45 = 0.205$$

These values which are the sum at hidden nodes then go for a non-linear transformation (sigmoid in this case) the sigmoid transformed values are the output of the hidden neurons which is given by.

A handwritten formula for the sigmoid function, $f(x) = \frac{1}{1 + e^{-x}}$, enclosed in a blue rectangular box.

Sigmoid expression

After feeding the summed up values (0.105 and 0.205) the output of the neuron becomes.

$$Oh1=0.526$$

$$Oh2=0.551$$

These outputs are further multiplied by the weight matrix for the next layer the equation becomes.

$$\Sigma OutH1 = Oh1*W5+Oh2*W6 = 0.526*0.55+0.551*0.65 = 0.647$$

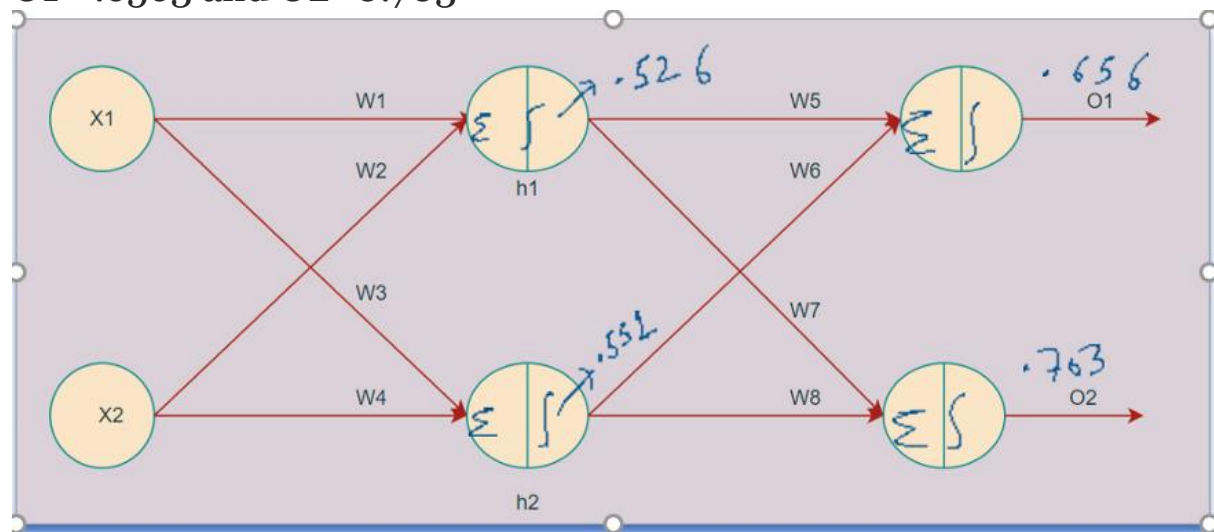
$$\Sigma OutH2 = Oh1*W7+Oh2*W8 = 0.526*0.75+0.551*0.85 = 0.862$$

We have taken the sigmoid function at the output node but for in practice Softmax is more appropriate especially when it comes to multiclass classification.

The Output will finally come to

$$\text{Sigmoid}(\Sigma OutH1) \text{ and } \text{Sigmoid}(\Sigma OutH2)$$

$O_1 = .6563$ and $O_2 = 0.703$



Feed Fwd calculation

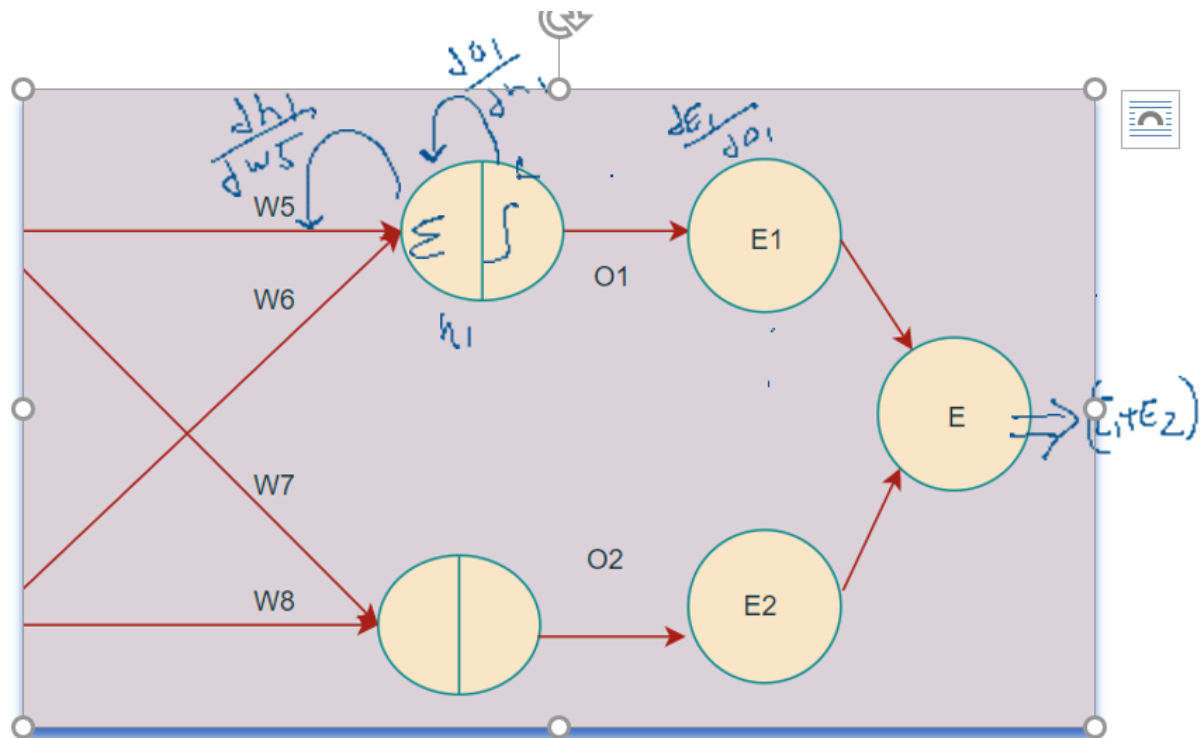
Backpropagation.

Backpropagation (BP) is a mechanism by which an error is distributed across the neural network to update the weights, till now this is clear that each weight has different amount of say in the total error hence in order to reduce the total error the weights needs to be updated accordingly, how much each individual weight changes with respect to one unit of change in error is the crux of backprop.

This is also called as the gradient, so finding gradient wrt total error for each weight is what Backprop does.

Mathematically gradient is expressed as partial derivation or differentiation.

Let's take one of the weights as an example and see how to understand the change in weight wrt change in the error.



Backprop representation

The total Error term **E** does not relate to any of the weight **W** directly and hence in other words if we want to calculate the change in **W5** wrt **E** we have to decode sequences in reverse order.

The Change in Weight **W5** depends on how the summation changes wrt activation which in turn depends on how activation changes wrt Error done by the first output neuron which in turn depends on the total Error , so now expressing mathematically the gradients, the gradient of **w5** wrt total Error will unfold to.

Chain rule of partial derivation

The right-hand Side of the equation which expands the dimensions on how **W5** is related to **E** unfolds a chain and hence known as “**The Chain Rule**” so that’s why in technical parlance it is said that in a Neural Network backpropagation happens through the Chain rule.

Short Questions

- 1.What is Neural Network?
- 2.What are the applications of Neurons?
- 3.Explain Feed Forward Briefly?
- 4.Explain Back Propagation Networks?

Long Questions:

- 1.Explain Concept of Neuron?
- 2.Explain Perceptron algorithm with example?
- 3.Difference between Feed Forward and Back Propagation Networks?

Unit-2

Deep learning

Deep learning is a branch of m, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don't need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now.

It's on hype nowadays because earlier we did not have that much processing power and a lot of data. As in the last 20 years, the processing power increases exponentially, deep learning and machine learning came in the picture. A formal definition of deep learning is- neurons

Deep Learning is a subset of Machine Learning that is based on artificial neural networks (ANNs) with multiple layers, also known as deep neural networks (DNNs). These neural networks are inspired by the structure and function of the human brain, and they are designed to learn from large amounts of data in an unsupervised or semi-supervised manner.

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).

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.

Convolutional Neural Networks (CNNs) are a special type of FNNs designed 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.

Recurrent Neural Networks (RNNs) are a type of neural networks that are 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.

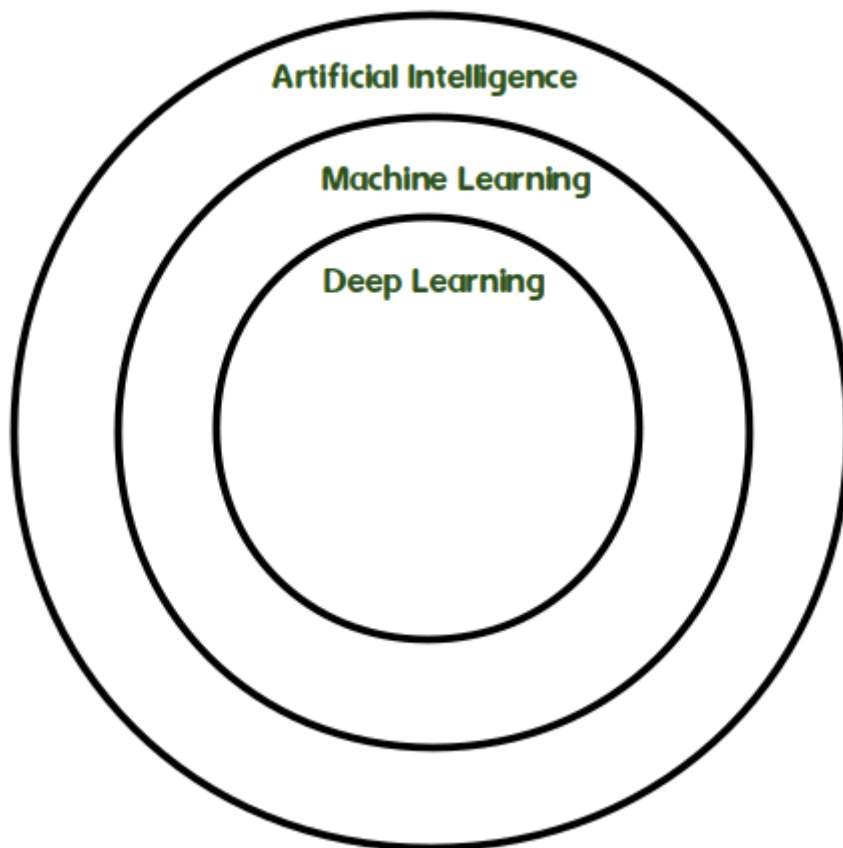
Deep Learning models are trained using large amounts of labeled data and require significant computational resources. With the increasing availability of large amounts of data and computational resources, deep learning has been able to achieve state-of-the-art performance in a wide range of applications such as image and speech recognition, natural language processing, and more.

Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

In human brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousand of their neighbours. The question here is how do we recreate these neurons in a computer. So, we create an artificial structure called an artificial neural net where we have nodes or neurons. We have some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer. **Architectures :**

1. **Deep Neural Network** – It is a neural network with a certain level of complexity (having multiple hidden layers in between input and output layers). They are capable of modeling and processing non-linear relationships.

2. **Deep Belief Network(DBN)** – It is a class of Deep Neural Network. It is multi-layer belief networks. **Steps for performing DBN** : a. Learn a layer of features from visible units using Contrastive Divergence algorithm. b. Treat activations of previously trained features as visible units and then learn features of features. c. Finally, the whole DBN is trained when the learning for the final hidden layer is achieved.
3. **Recurrent (perform same task for every element of a sequence) Neural Network** – Allows for parallel and sequential computation. Similar to the human brain (large feedback network of connected neurons). They are able to remember important things about the input they received and hence enables them to be more precise.



Difference

between Machine Learning and Deep Learning :

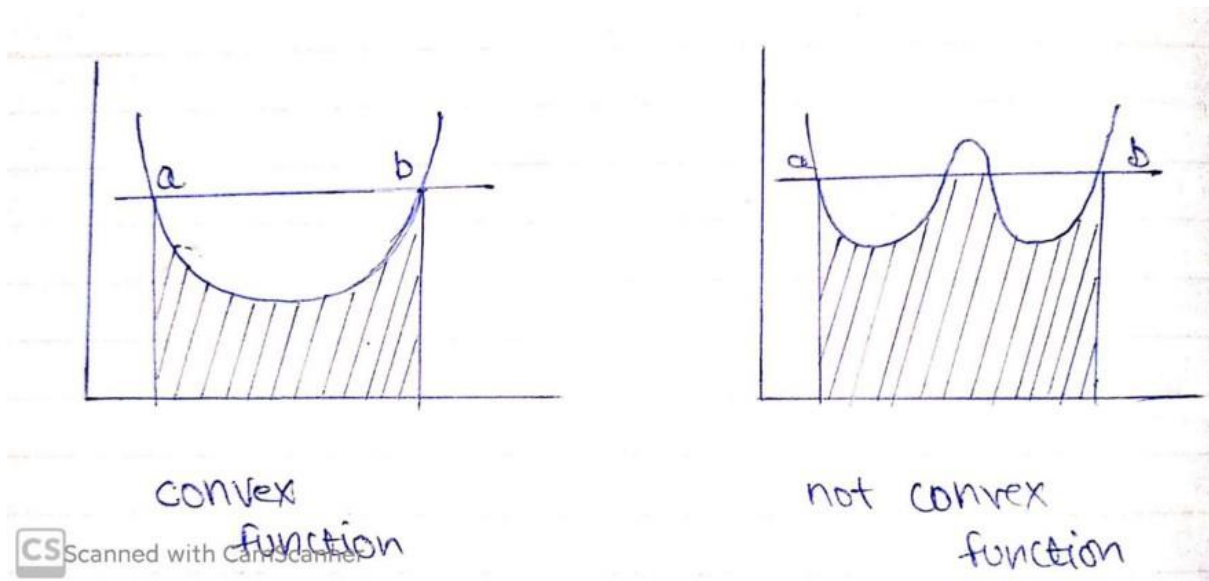
Machine Learning	Deep Learning
Works on small amount of Dataset for accuracy.	Works on Large amount

Machine Learning	Deep Learning
	of Dataset.
Dependent on Low-end Machine.	Heavily dependent on High-end Machine.
Divides the tasks into sub-tasks, solves them individually and finally combine the results.	Solves problem end to end.
Takes less time to train.	Takes longer time to train.
Testing time may increase.	Less time to test the data.

Working : First, we need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). Second, we need to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Third, Choose the Deep Learning Algorithm appropriately. Fourth, Algorithm should be used while training the dataset. Fifth, Final testing should be done on dataset.

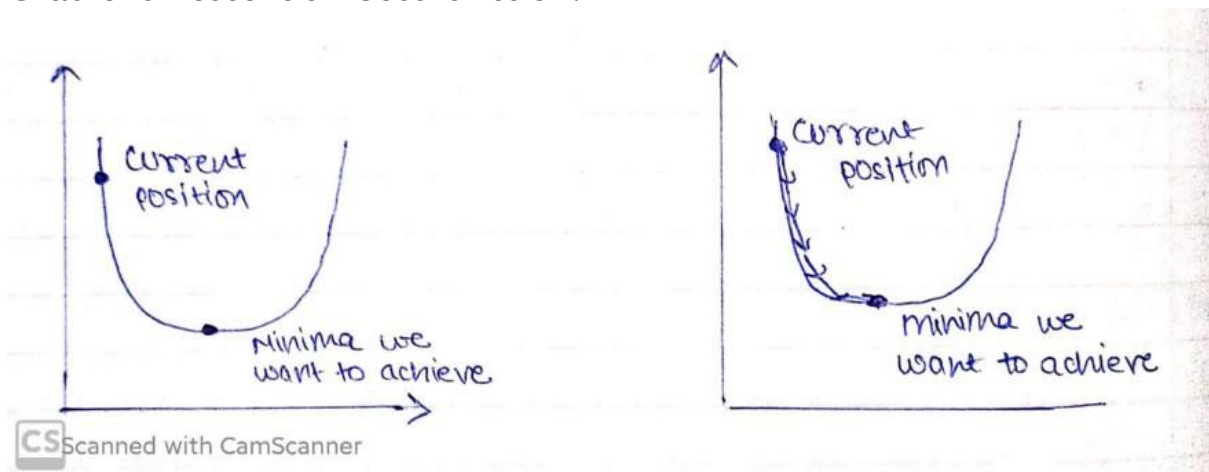
Gradient Descent

Gradient Descent is an optimizing algorithm used in Machine/ Deep Learning algorithms. The goal of Gradient Descent is to minimize the objective convex function $f(x)$ using iteration.



Convex function v/s Not Convex function

Gradient Descent on Cost function.



Intuition behind Gradient Descent

For ease, let's take a simple linear model.

$$\text{Error} = Y(\text{Predicted}) - Y(\text{Actual})$$

A machine learning model always wants low error with maximum accuracy, in order to decrease error we will intuit our algorithm that

you're doing something wrong that is needed to be rectified, that would be done through Gradient Descent.

We need to minimize our error, in order to get pointer to minima we need to walk some steps that are known as alpha(learning rate).

Steps to implement Gradient Descent

1. Randomly initialize values.

2. Update values.

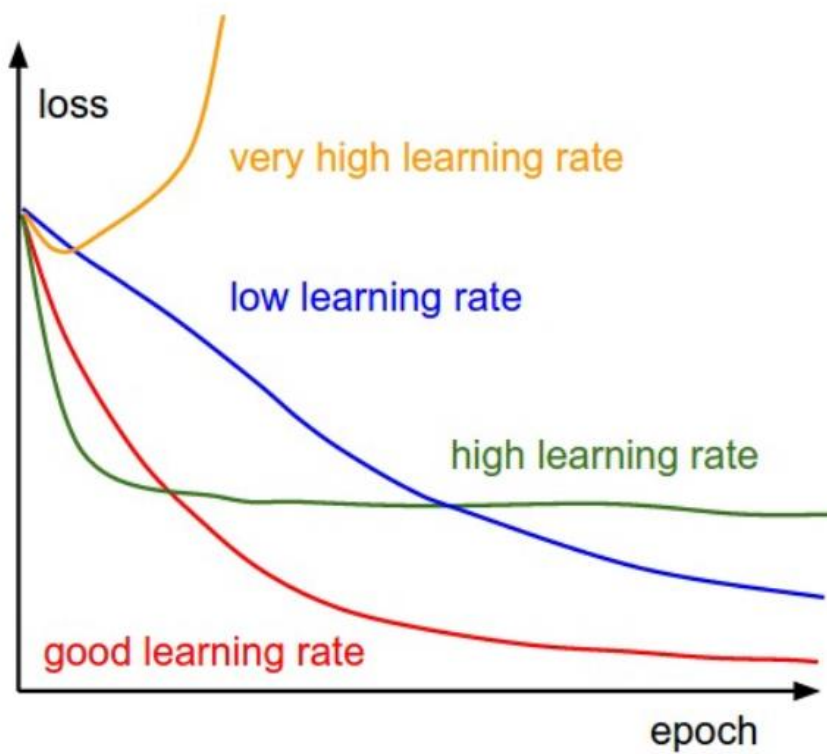
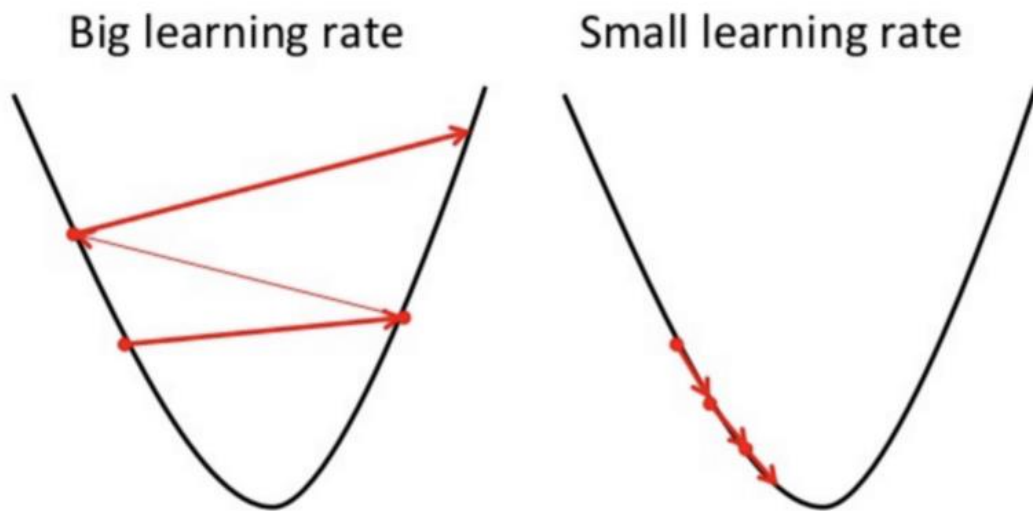
$$weight^{(new)} = weight^{(old)} - constant \frac{\partial J(\Theta)}{\partial weight}$$

3. Repeat until slope =0

A derivative is a term that comes from calculus and is calculated as the slope of the graph at a particular point. The slope is described by drawing a tangent line to the graph at the point. So, if we are able to compute this tangent line, we might be able to compute the desired direction to reach the minima.

Learning rate must be chosen wisely as:

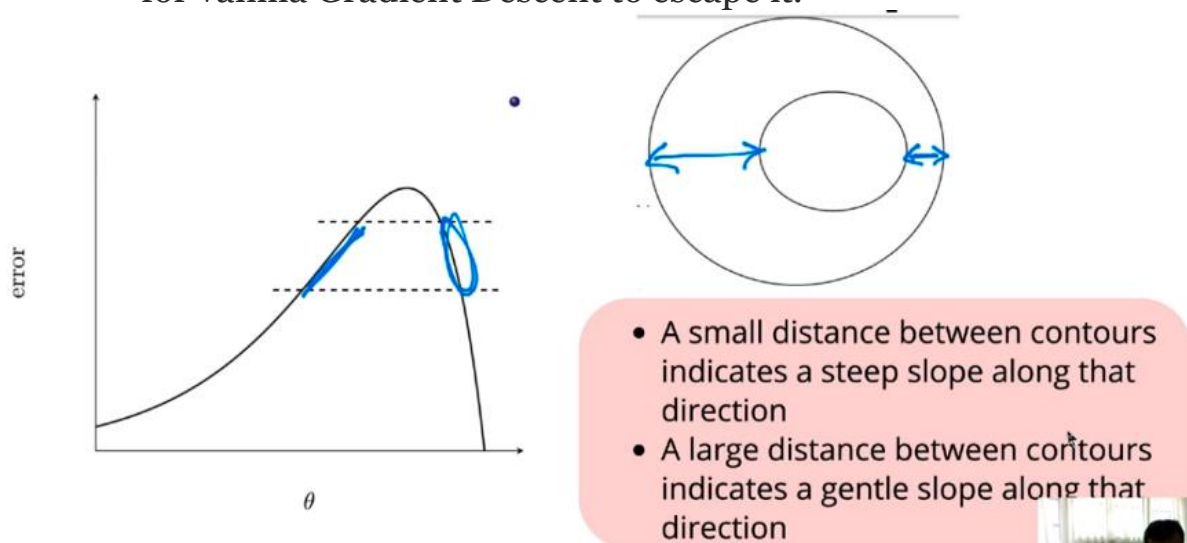
1. if it is too small, then the model will take some time to learn.
2. if it is too large, model will converge as our pointer will shoot and we'll not be able to get to minima.



Gradient Descent with different learning rates, [Source](#)

Vanilla gradient descent, however, can't guarantee good convergence, due to following reasons:

- Picking an appropriate learning rate can be troublesome. A learning rate that is too low will lead to slow training and a higher learning rate will lead to overshooting of slope.
- Another key hurdle faced by Vanilla Gradient Descent is it avoid getting trapped in local minima; these local minimas are surrounded by hills of same error, which makes it really hard for vanilla Gradient Descent to escape it.



Contour maps visualizing gentle and steep region of curve, [Source](#)

In simple words, every step we take towards minima tends to decrease our slope, now if we visualize, in steep region of curve derivative is going to be large therefore steps taken by our model too would be large but as we will enter gentle region of slope our derivative will decrease and so will the time to reach minima.

Momentum Based Gradient Descent

If we consider, Simple Gradient Descent completely relies only on calculation i.e. if there are 10000 steps, then our model would try to implement Simple Gradient Descent for 10000 times that would be obviously too much time consuming and computationally expensive.

In laymen language, suppose a man is walking towards his home but he don't know the way so he ask for direction from by passer, now we expect him to walk some distance and then ask for direction but man is asking for direction at every step he takes, that is obviously more time consuming, now compare man with Simple Gradient Descent and his goal with minima.

In order to avoid drawbacks of vanilla Gradient Descent, we introduced momentum based Gradient Descent where the goal is to lower the computation time and that can be achieved when we introduce the concept of experience i.e. the confidence using previous steps.

In this way rather than computing new steps again and again we are averaging the decay and as decay increases its effect in decision making decreases and thus the older the step less effect on decision making. More the history more bigger steps will be taken.

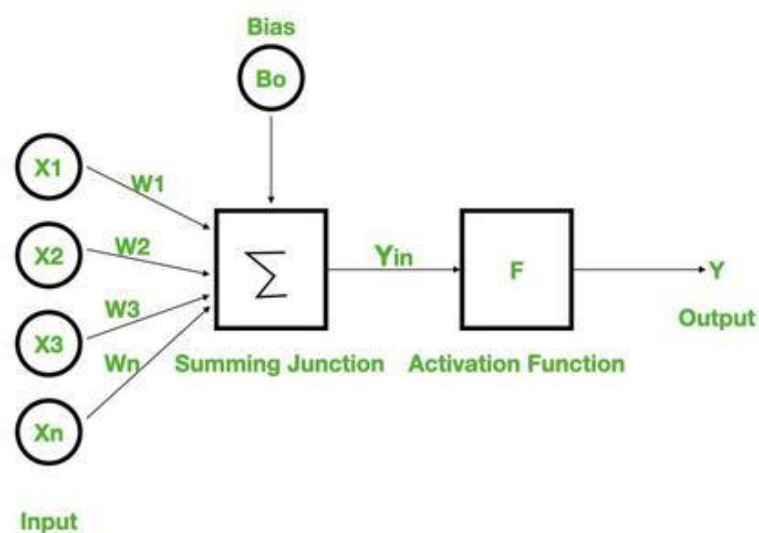
Even in the gentle region, momentum based Gradient Descent is taking large steps due to the momentum it is burdening along.

Back Propagation Algorithm

Backpropagation is an algorithm that backpropagates the errors from the output nodes to the input nodes. Therefore, it is simply referred to as the backward propagation of errors. It uses in the vast applications of neural networks in data mining like Character recognition, Signature verification, etc.

Neural Network:

Neural networks are an information processing paradigm inspired by the human nervous system. Just like in the human nervous system, we have biological neurons in the same way in neural networks we have artificial neurons, artificial neurons are mathematical functions derived from biological neurons. The human brain is estimated to have about 10 billion neurons, each connected to an average of 10,000 other neurons. Each neuron receives a signal through a synapse, which controls the effect of the signal concerning on the neuron.



Backpropagation:

Backpropagation is a widely used algorithm for training feedforward neural networks. It computes the gradient of the loss function with respect to the network weights. It is very efficient, rather than naively directly computing the gradient concerning each weight. This efficiency makes it possible to use gradient methods to train multi-layer networks.

and update weights to minimize loss; variants such as gradient descent or stochastic gradient descent are often used.

The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight via the chain rule, computing the gradient layer by layer, and iterating backward from the last layer to avoid redundant computation of intermediate terms in the chain rule.

Features of Backpropagation:

1. it is the gradient descent method as used in the case of simple perceptron network with the differentiable unit.
2. it is different from other networks in respect to the process by which the weights are calculated during the learning period of the network.
3. training is done in the three stages :
 - the feed-forward of input training pattern
 - the calculation and backpropagation of the error
 - updation of the weight

Working of Backpropagation:

Neural networks use supervised learning to generate output vectors from input vectors that the network operates on. It Compares generated output to the desired output and generates an error report if the result does not match the generated output vector. Then it adjusts the weights according to the bug report to get your desired output.

Backpropagation Algorithm:

Step 1: Inputs X, arrive through the preconnected path.

Step 2: The input is modeled using true weights W. Weights are usually chosen randomly.

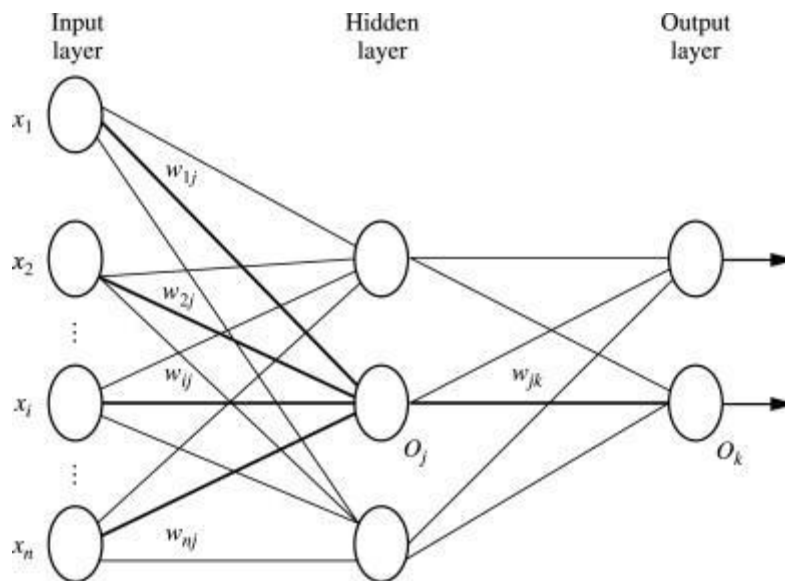
Step 3: Calculate the output of each neuron from the input layer to the hidden layer to the output layer.

Step 4: Calculate the error in the outputs

Backpropagation Error= Actual Output – Desired Output

Step 5: From the output layer, go back to the hidden layer to adjust the weights to reduce the error.

Step 6: Repeat the process until the desired output is achieved.



Parameters :

- x = inputs training vector $x=(x_1, x_2, \dots, x_n)$.
- t = target vector $t=(t_1, t_2, \dots, t_n)$.
- δ_k = error at output unit.
- δ_i = error at hidden layer.
- α = learning rate.
- V_{0j} = bias of hidden unit j .

Training Algorithm :

Step 1: Initialize weight to small random values.

Step 2: While the stopping condition is to be false do step 3 to 10.

Step 3: For each training pair do step 4 to 9 (Feed-Forward).

Step 4: Each input unit receives the signal unit and transmits the signal x_i signal to all the units.

Step 5 : Each hidden unit Z_j ($z=1$ to a) sums its weighted input signal to calculate its net input

$$z_{inij} = v_{0j} + \sum x_i v_{ij} \quad (i=1 \text{ to } n)$$

Applying activation function $z_i = f(z_{inij})$ and sends this signals to all units in the layer about $i.e$ output units

Vanishing Gradient problem

In Machine Learning, the Vanishing Gradient Problem is encountered while training Neural Networks with methods (example, Back Propagation). This problem makes it hard to learn and tune the parameters of the earlier layers in the network.

The vanishing gradients problem is one example of unstable behaviour that you may encounter when training a deep neural network.

It describes the situation where a deep multilayer feed-forward network or a recurrent neural network is unable to propagate useful gradient information from the output end of the model back to the layers near the input end of the model.

The result is the general inability of models with many layers to learn on a given dataset, or for models with many layers to prematurely converge to a poor solution.

Methods proposed to overcome vanishing gradient problem

1. Multi-level hierarchy
2. Long short – term memory
3. Faster hardware
4. Residual neural networks (ResNets)
5. ReLU

Residual neural networks (ResNets)

One of the newest and most effective **ways** to resolve the **vanishing gradient** problem is with residual neural networks, or ResNets (not to be confused with recurrent neural networks). It was noted before ResNets that a deeper network would have higher training error than the shallow network.

The weights of a neural network are updated using the backpropagation algorithm. The backpropagation algorithm makes a small change to each weight in such a way that the loss of the model decreases. How does this happen? It updates each weight such that it takes a step in the direction along which the loss decreases. This direction is nothing but the gradient of this weight (concerning the loss).

Using the chain rule, we can find this gradient for each weight. It is equal to (local gradient) x (gradient flowing from ahead),

Here comes the problem. As this gradient keeps flowing backwards to the initial layers, this value keeps getting multiplied by each local gradient. Hence, the gradient becomes smaller and smaller, making the updates to the initial layers very small, increasing the training time considerably. We can solve our problem if the local gradient somehow became 1.

ResNet

How can the local gradient be 1, i.e, the derivative of which function would always be 1? The Identity function!

As this gradient is back propagated, it does not decrease in value because the local gradient is 1.

The ResNet architecture, shown below, should now make perfect sense as to how it would not allow the vanishing gradient problem to occur. ResNet stands for Residual Network.

These *skip connections* act as gradient *superhighways*, allowing the gradient to flow unhindered. And now you can understand why ResNet comes in flavours like ResNet50, ResNet101 and ResNet152.

In the TensorFlow code that I am about to show you, we'll be creating a seven-layer densely connected network (including the input and output layers) and using the TensorFlow summary operations and TensorBoard visualization to see what is going on with the gradients. The code uses the TensorFlow layers (tf.layers) framework, which allows quick and easy building of networks. The data we will be training the network on is the MNIST hand-written digit recognition dataset that comes packaged up with the TensorFlow installation. To create the dataset, we can run the following:

Mitigation

There are several attacks against deep learning models in the literature, including **fast-gradient sign method** (FGSM), **basic iterative method** (BIM) or **momentum iterative method** (MIM) attacks. These attacks are the purest form of the gradient-based evading technique that is used by attackers to evade the classification model.

Cite The Code

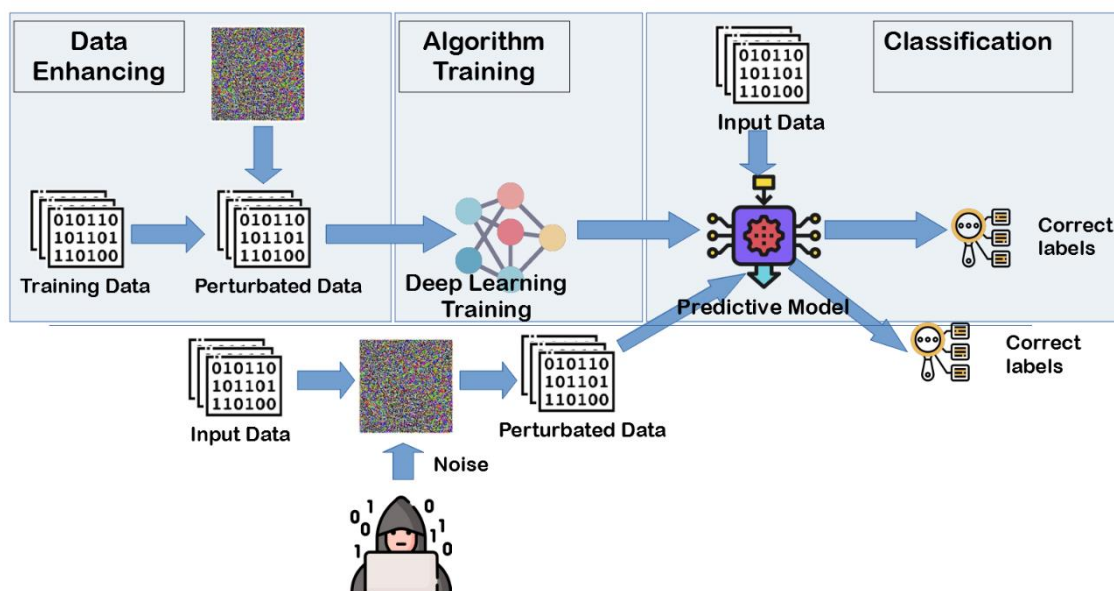
In this work, I will present a new approach to protect a malicious activity detection model from the several adversarial machine learning attacks. Hence, we explore the power of applying adversarial training to build a robust model against FGSM attacks. Accordingly, (1) dataset enhanced with the adversarial examples; (2) deep neural network-based detection model is trained using the KDDCUP99 dataset to learn the FGSM based attack patterns. We applied this training model to the benchmark cybersecurity dataset.

The adversarial machine learning has been used to describe the attacks to machine learning models, which tries to mislead models by malicious input instances. The figure shows the typical adversarial machine leaning attack.

A tyical machine learning model basically consists of two stages as training time and decision time. Thus, the adversarial machine learning attacks occur in either training time or decision time. The techniques used by hackers for adversarial machine learning can be divided into two, according to the time of the attack:

- **Data Poisoning:** The attacker changes some labels of training input instances to mislead the output model.
- **Model Poisoning:** The hacker drives model to produce false labelling using some perturbed instance after the model is created.

Our model is able to respond to the model attacks by hackers who use the adversarial machine learning methods. The figure illustrates the system architecture used to protect the model and to classify correctly.



In this work, we will use standard KDDCUP'99 intrusion detection dataset to show the results. We need to extract the numerical features from the dataset. I created a new method to load and extract the KDDCUP'99 dataset.

confusion matrix

10874	15	0	0
16	17530	5	17
6	24	400	11
3	23	5	178

Adversarial trained model's confusion matrix

10878	11	0	0
12	17537	4	15
1	16	420	4
0	21	2	186

ReLU Heuristics for avoiding bad local minima

Activation functions are mathematical equations that define how the weighted sum of the input of a neural node is transformed into an output, and they are key parts of an artificial neural network (ANN) architecture.

Activation functions add **non-linearity** to a neural network, allowing the network to learn complex patterns in the data. The choice of activation function has a significant impact on an ANN's performance, and one of the most popular choices is the **Rectified Linear Unit (ReLU)**.

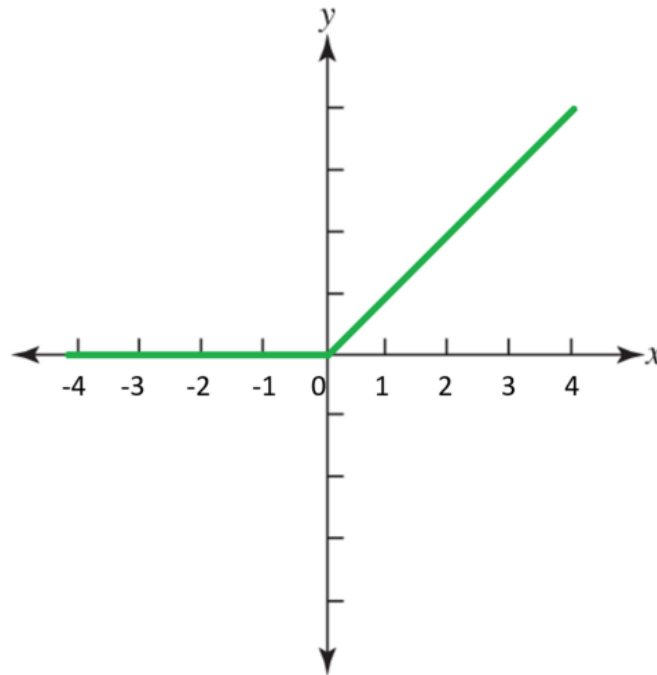
What is ReLU, and what are its advantages?

The Rectified Linear Unit (ReLU) activation function can be described as:

$$f(x) = \max(0, x)$$

What it does is:

- (i) For negative input values, output = 0
- (ii) For positive input values, output = original input value



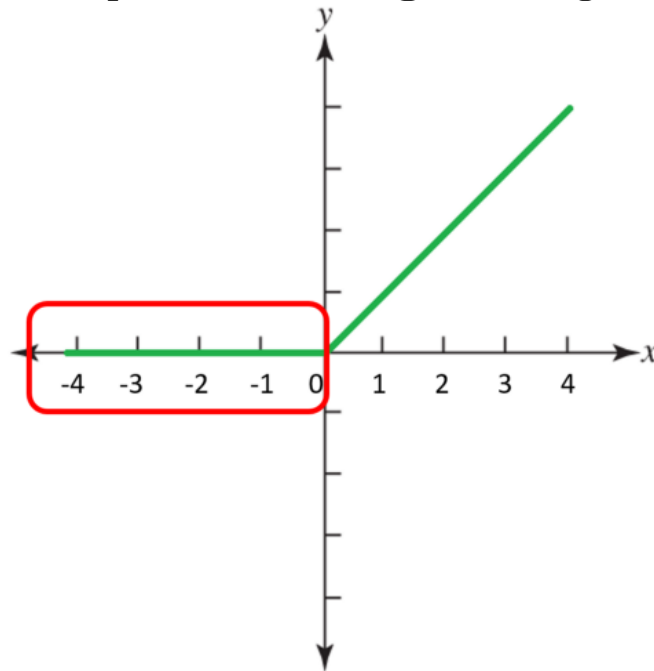
Graphic representation of ReLU activation function

ReLU has gained massive popularity because of several key **advantages**:

- ReLU takes less time to learn and is computationally less expensive than other common activation functions (e.g., *tanh*, *sigmoid*). Because it outputs 0 whenever its input is negative, fewer neurons will be activated, leading to **network sparsity** and thus higher **computational efficiency**.
- ReLU involves **simpler mathematical operations** compared to *tanh* and *sigmoid*, thereby boosting its computational performance further.

What's the Dying ReLU problem?

The dying ReLU problem refers to the scenario when many ReLU neurons only output values of 0. The red outline below shows that this happens when the inputs are in the **negative** range.



Red outline (in the negative x range) demarcating the horizontal segment where ReLU outputs 0

While this characteristic gives ReLU its strengths (through network sparsity), it becomes a problem when most of the inputs to these ReLU neurons are in the negative range. The worst-case scenario is when the entire network dies, meaning that it becomes just a constant function.

When most of these neurons return output zero, the gradients fail to flow during backpropagation, and the weights are not updated. Ultimately a large part of the network becomes inactive, and it is unable to learn further.

Because the slope of ReLU in the negative input range is also zero, once it becomes dead (i.e., stuck in negative range and giving output 0), it is likely to remain unrecoverable.

However, the dying ReLU problem does not happen all the time since the optimizer (e.g., stochastic gradient descent) considers multiple input values each time. **As long as NOT all the inputs** push ReLU to the negative segment (i.e., some inputs are in the positive range), the neurons can stay active, the weights can get updated, and the network can continue learning.

What causes the Dying ReLU problem?

The dying ReLU problem is commonly driven by these two factors:

(i) High learning rate

Let us first look at the equation for the update step in backpropagation:

Equation for update rule (Image by author)

If our learning rate (α) is set too high, there is a significant chance that our new weights will end up in the highly negative value range since our old weights will be subtracted by a large number. These negative weights result in negative inputs for ReLU, thereby causing the dying ReLU problem to happen.

*Note: Recall that input to activation function is $(W*x) + b$.*

(ii) Large negative bias

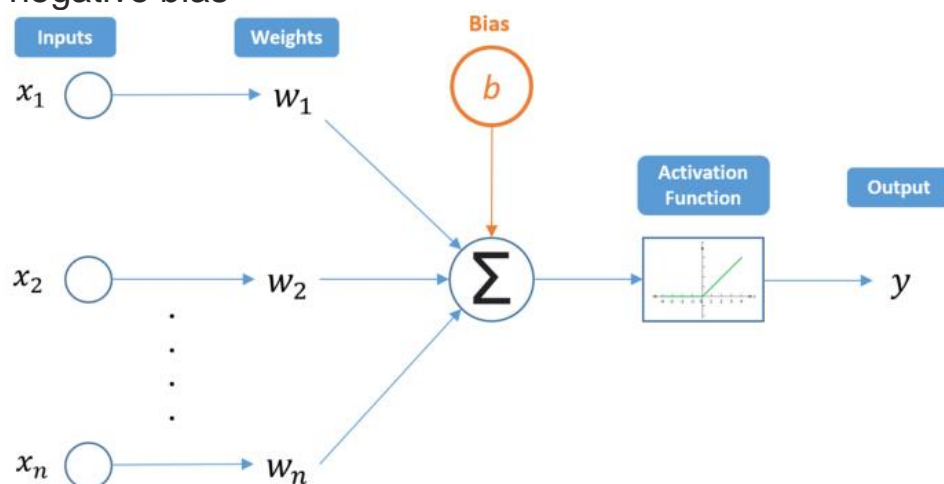


Illustration of a simple neural network (Image by author)

While we have mostly talked about weights so far, we must not forget that the bias term is also passed along with the weights into the activation function.

Bias is a constant value added to the product of inputs and weights. Given its involvement, a large negative bias term can cause the ReLU activation inputs to become negative. This, as already described, causes the neurons to consistently output 0, leading to the dying ReLU problem.

How to solve the Dying ReLU problem?

There are several ways to tackle the dying ReLU problem:

(i) Use of a lower learning rate

Since a large learning rate results in a higher likelihood of negative weights (thereby increasing chances of dying ReLU), it can be a good idea to decrease the learning rate during the training process.

Heuristics For Faster Training

with unseen data A heuristic is, simply put, a shortcut. Heuristics are strategies often used to find a solution that is not perfect, but is within an acceptable degree of accuracy for the needs of the process. In computing, heuristics are especially useful when finding an optimal solution to a problem is impractical because of slow speed or processing power limitations.

In the example above, the heuristic we used was based on some consistent context (the fact that sales tax happens to be between around 10%). You should be wary when taking this heuristic into new territory where your assumptions may not be accurate. For example, if you went out to dinner in Saskatchewan, Canada where the provincial sales tax rate is 6%, and applied this heuristic, you may find yourself with a very dissatisfied waiter. The point is, heuristics are great when the assumptions you've based your heuristic on are sufficiently consistent.

8. Nesterov's Accelerated Gradient Descent

Nesterov's Accelerated Gradient Descent

In this lecture we consider the same setting than in the previous post (that is we want to minimize a smooth convex function over \mathbb{R}^n). Previously we saw that the plain Gradient Descent algorithm has a rate of convergence of order $1/t$ after t steps, while the lower bound that we proved is of order $1/t^2$.

We present now a beautiful algorithm due to Nesterov, called Nesterov's Accelerated Gradient Descent, which attains a rate of order $1/t^2$. First we define the following sequences:

$$\lambda_0 = 0, \quad \lambda_s = \frac{1 + \sqrt{1 + 4\lambda_{s-1}^2}}{2}, \quad \text{and} \quad \gamma_s = \frac{1 - \lambda_s}{\lambda_{s+1}}.$$

(Note that $\gamma_s \leq 0$.) Now the algorithm is simply defined by the following equations, with an arbitrary initial point $x_1 = y_1$,

$$\begin{aligned}
y_{s+1} &= x_s - \frac{1}{\beta} \nabla f(x_s), \\
x_{s+1} &= (1 - \gamma_s) y_{s+1} + \gamma_s y_s.
\end{aligned}$$

In other words, Nesterov's Accelerated Gradient Descent performs a simple step of gradient descent to go from x_s to y_{s+1} , and then it 'slides' a little bit further than y_{s+1} in the direction given by the previous point y_s .

$$\begin{aligned}
& f\left(x - \frac{1}{\beta} \nabla f(x)\right) - f(y) \\
& \leq f\left(x - \frac{1}{\beta} \nabla f(x)\right) - f(x) + \nabla f(x)^\top (x - y) \\
& \leq \nabla f(x)^\top \left(x - \frac{1}{\beta} \nabla f(x) - x\right) + \frac{\beta}{2} \left\|x - \frac{1}{\beta} \nabla f(x) - x\right\|^2 + \nabla f(x)^\top (x - y) \\
& = -\frac{1}{2\beta} \|\nabla f(x)\|^2 + \nabla f(x)^\top (x - y).
\end{aligned}$$

Now let us apply this inequality to $x = x_s$ and $y = y_s$, which gives

$$\begin{aligned}
f(y_{s+1}) - f(y_s) &= f\left(x_s - \frac{1}{\beta} \nabla f(x_s)\right) - f(y_s) \\
&\leq -\frac{1}{2\beta} \|\nabla f(x_s)\|^2 + \nabla f(x_s)^\top (x_s - y_s) \\
&= -\frac{\beta}{2} \|y_{s+1} - x_s\|^2 - \beta(y_{s+1} - x_s)^\top (x_s - y_s).
\end{aligned}
\tag{1}$$

Similarly we apply it to $x = x_s$ and $y = x^*$ which gives

$$(2) \quad f(y_{s+1}) - f(x^*) \leq -\frac{\beta}{2} \|y_{s+1} - x_s\|^2 - \beta(y_{s+1} - x_s)^\top (x_s - x^*).$$

Now multiplying (1) by $(\lambda_s - 1)$ and adding the result to (2), one obtains with $\delta_s = f(y_s) - f(x^*)$,

$$\lambda_s \delta_{s+1} - (\lambda_s - 1) \delta_s \leq -\frac{\beta}{2} \lambda_s \|y_{s+1} - x_s\|^2 - \beta(y_{s+1} - x_s)^\top (\lambda_s x_s - (\lambda_s - 1)y_s - x^*).$$

Multiplying this inequality by λ_s and using that by definition $\lambda_{s-1}^2 = \lambda_s^2 - \lambda_s$ one obtains

$$\begin{aligned}
& \lambda_s^2 \delta_{s+1} - \lambda_{s-1}^2 \delta_s \\
(3) \quad & \leq -\frac{\beta}{2} \left(\|\lambda_s (y_{s+1} - x_s)\|^2 + 2\lambda_s (y_{s+1} - x_s)^\top (\lambda_s x_s - (\lambda_s - 1)y_s - x^*) \right).
\end{aligned}$$

Now one can verify that

$$\begin{aligned} & \|\lambda_s(y_{s+1} - x_s)\|^2 + 2\lambda_s(y_{s+1} - x_s)^\top (\lambda_s x_s - (\lambda_s - 1)y_s - x^*) \\ (4) &= \|\lambda_s y_{s+1} - (\lambda_s - 1)y_s - x^*\|^2 - \|\lambda_s x_s - (\lambda_s - 1)y_s - x^*\|^2. \end{aligned}$$

Next remark that, by definition, one has

$$\begin{aligned} x_{s+1} &= y_{s+1} + \gamma_s(y_s - y_{s+1}) \\ &\Leftrightarrow \lambda_{s+1}x_{s+1} = \lambda_{s+1}y_{s+1} + (1 - \lambda_s)(y_s - y_{s+1}) \\ (5) &\Leftrightarrow \lambda_{s+1}x_{s+1} - (\lambda_{s+1} - 1)y_{s+1} = \lambda_s y_{s+1} - (\lambda_s - 1)y_s. \end{aligned}$$

Putting together (3), (4) and (5) one gets with $u_s = \lambda_s x_s - (\lambda_s - 1)y_s - x^*$,

$$\lambda_s^2 \delta_{s+1} - \lambda_{s-1}^2 \delta_s^2 \leq \frac{\beta}{2} \left(\|u_s\|^2 - \|u_{s+1}\|^2 \right).$$

Summing these inequalities from $s = 1$ to $s = t - 1$ one obtains:

$$\delta_t \leq \frac{\beta}{2\lambda_{t-1}^2} \|u_1\|^2.$$

By induction it is easy to see that $\lambda_{t-1} \geq \frac{t}{2}$ which concludes the proof

Regularization

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it.

Sometimes the [machine learning](#) model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals by introducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.

This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model.

It mainly regularizes or reduces the coefficient of features toward zero. In simple words, *"In regularization technique, we reduce the magnitude of the features by keeping the same number of features."*

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_n x_n + b$$

In the above equation, Y represents the value to be predicted

X1, X2, ...Xn are the features for Y.

$\beta_0, \beta_1, \dots, \beta_n$ are the weights or magnitude attached to the features, respectively. Here represents the bias of the model, and b represents the intercept.

Linear regression models try to optimize the β_0 and b to minimize the cost function. The equation for the cost function for the linear model is given below:

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

Now, we will add a loss function and optimize parameter to make the model that can predict the accurate value of Y. The loss function for the linear regression is called as **RSS or Residual sum of squares**.

Techniques of Regularization

There are mainly two types of regularization techniques, which are given below:

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

Ridge Regression

- Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions.
- Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as **L2 regularization**.

- In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called **Ridge Regression penalty**. We can calculate it by multiplying with the lambda to the squared weight of each individual feature.
- The equation for the cost function in ridge regression will be:

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

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

Lasso Regression:

- Lasso regression is another regularization technique to reduce the complexity of the model.
- It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.
- Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
- It is also called as **L1 regularization**. The equation for the cost function of Lasso regression will be:

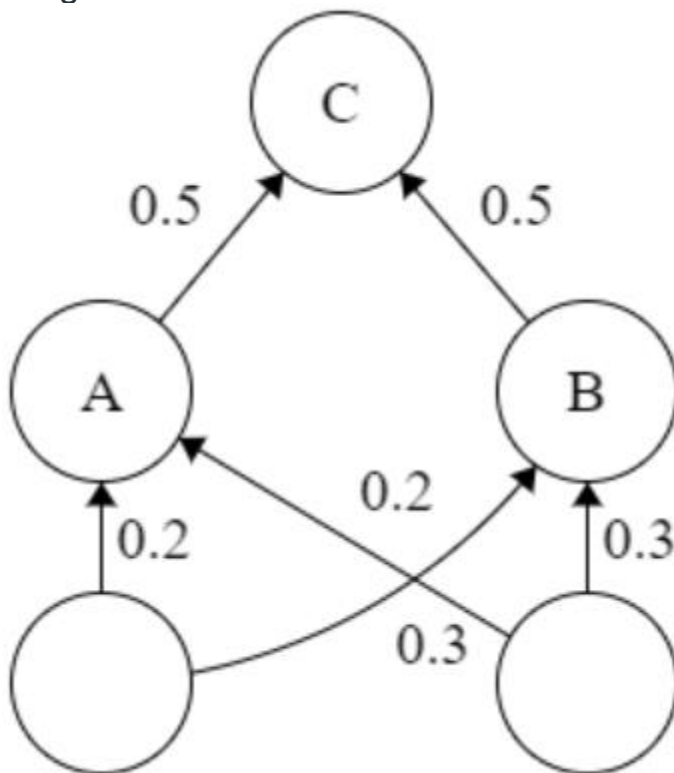
$$\sum_{i=1}^M (y_i - y'_i)^2 = \sum_{i=1}^M \left(y_i - \sum_{j=0}^n \beta_j * x_{ij} \right)^2 + \lambda \sum_{j=0}^n |\beta_j|$$

- Some of the features in this technique are completely neglected for model evaluation.
- Hence, the Lasso regression can help us to reduce the overfitting in the model as well as the feature selection.

Drop Out

The concept of Neural Networks is inspired by the neurons in the human brain and scientists wanted a machine to replicate the same process. This craved a path to one of the most important topics in Artificial Intelligence. A Neural Network (NN) is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Since such a network is created artificially in machines, we refer to that as Artificial Neural Networks (ANN). This article assumes that you have a decent knowledge of ANN. More about ANN can be found Now, let us go narrower into the details of **Dropout** in ANN.

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.

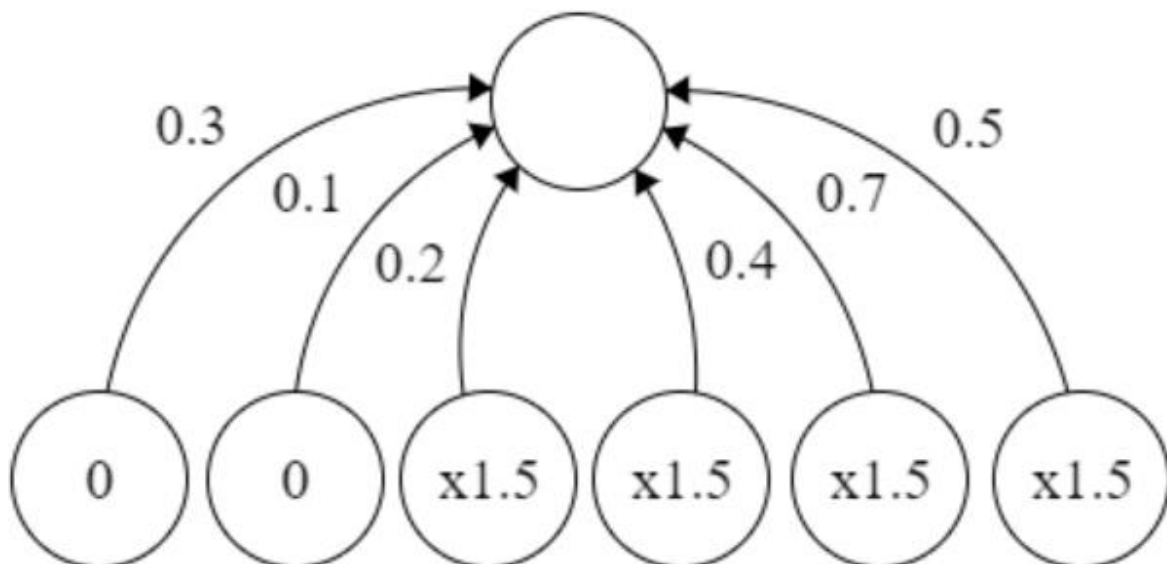


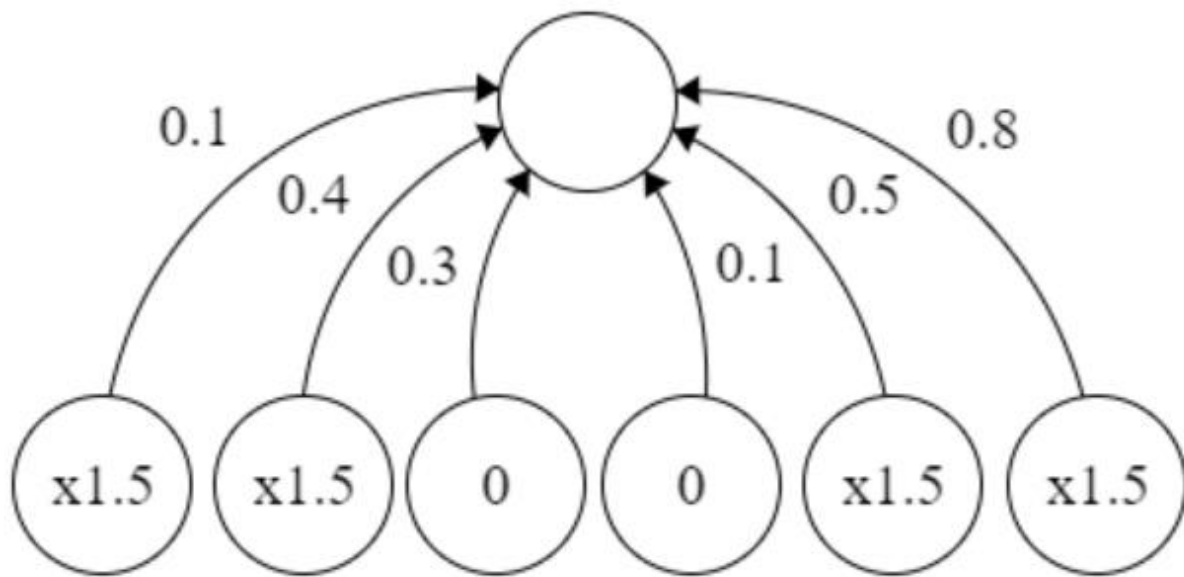
This poses two different problems to our model:

- 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, p . The remaining neurons have their

values multiplied by $\frac{1}{1-p}$ 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 $\frac{1}{3}$, and the remaining 4 neurons at each training step have their value scaled by $\times 1.5$. 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.

Short Questions

1. What is deep learning?
2. Explain Gradient Descent?
3. Explain Briefly about Mitigation?
4. What is Regularization?
5. Explain about Drop out in deep learning?

Long Questions

1. Explain Vanishing Gradient Problem?
2. Explain about ReLu Heuristics for avoiding bad local minimum?
3. What is Nestors Accelerated Gradient Descent with Example

4.Explain heuristics for faster training?

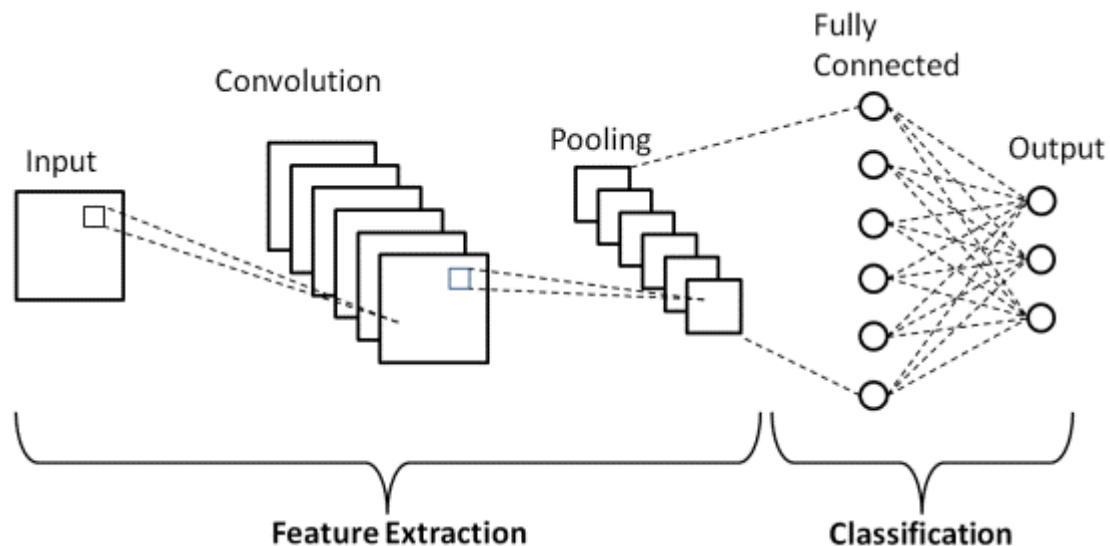
5.Explain Backpropagation algorithm in detail?

Unit-3

CNNArchitecture:

There are two main parts to a CNN architecture

- A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction.
- The network of feature extraction consists of many pairs of convolutional or pooling layers.
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.
- This CNN model of feature extraction aims to reduce the number of features present in a dataset. It creates new features which summarizes the existing features contained in an original set of features. There are many CNN layers as shown in the CNN architecture diagram.



Convolution Layers

There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully-connected (FC) layers. When these layers are stacked, a CNN architecture will be formed. In addition to

these three layers, there are two more important parameters which are the dropout layer and the activation function which are defined be

1. Convolutional Layer

This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size $M \times M$. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter ($M \times M$).

The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

The convolution layer in CNN passes the result to the next layer once applying the convolution operation in the input. Convolutional layers in CNN benefit a lot as they ensure the spatial relationship between the pixels is intact.

2. Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer.

The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations. It basically summarises the features generated by a convolution layer.

In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

This CNN model generalises the features extracted by the convolution layer, and helps the networks to recognise the features independently. With the help of this, the computations are also reduced in a network.

3. Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

In this, the input image from the previous layers are flattened and fed to the FC layer. 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. The reason two layers are connected is that two fully connected layers will perform better than a single connected layer. These layers in CNN reduce the human supervision

4. Dropout

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on a new data.

To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

Dropout results in improving the performance of a machine learning model as it prevents overfitting by making the network simpler. It drops neurons from the neural networks during train.

5. Activation Functions

Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network. It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred and for a multi-class classification, generally softmax is used. In simple terms, activation functions in a CNN model determine whether a neuron should be activated or not. It decides whether the input to the work is important or not to predict using mathematical operation

Convolution

Convolution is a mathematical operation used to express the relation between input and output of an LTI system. It relates input, output and impulse response of an LTI system as

$$y(t) = x(t) * h(t) \quad \text{or} \quad Y(\omega) = X(\omega) * H(\omega)$$

Where $y(t)$ = output of LTI

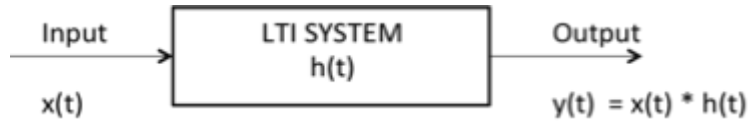
$x(t)$ = input of LTI

$h(t)$ = impulse response of LTI

There are two types of convolutions:

- Continuous convolution
- Discrete convolution

Continuous Convolution

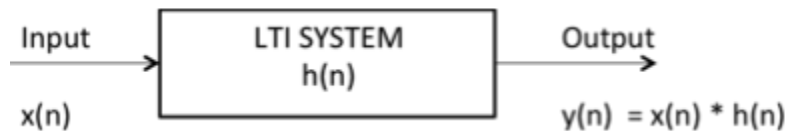


$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau) h(t - \tau) d\tau$$

(or)

$$y(t) = \int_{-\infty}^{\infty} x(t - \tau) h(\tau) d\tau$$

Discrete Convolution



$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k) h(n - k)$$

(or)

$$y(n) = \sum_{k=-\infty}^{\infty} x(n - k) h(k)$$

By using convolution we can find zero state response of the system.

Deconvolution

Deconvolution is reverse process to convolution widely used in signal and image processing.

Properties of Convolution

Commutative Property

$$x_1(t) * x_2(t) = x_2(t) * x_1(t)$$

Distributive Property

$$x_1(t) * [x_2(t) + x_3(t)] = [x_1(t) * x_2(t)] + [x_1(t) * x_3(t)]$$

Associative Property

$$x_1(t) * [x_2(t) * x_3(t)] = [x_1(t) * x_2(t)] * x_3(t)$$

Shifting Property

$$x_1(t) * x_2(t) = y(t)$$

$$x_1(t) * x_2(t - t_0) = y(t - t_0)$$

$$x_1(t - t_0) * x_2(t) = y(t - t_0)$$

$$x_1(t-t_0) * x_2(t-t_1) = y(t-t_0-t_1) \quad \diamond 1(\diamond - \diamond 0) * \diamond 2(\diamond - \diamond 1) = \diamond(\diamond - \diamond 0 - \diamond 1)$$

Convolution with Impulse

$$x_1(t) * \delta(t) = x(t) \quad \diamond 1(\diamond) * \diamond(\diamond) = \diamond(\diamond)$$

$$x_1(t) * \delta(t-t_0) = x(t-t_0) \quad \diamond 1(\diamond) * \diamond(\diamond - \diamond 0) = \diamond(\diamond - \diamond 0)$$

Convolution of Unit Steps

$$u(t) * u(t) = r(t) \quad \diamond(\diamond) * \diamond(\diamond) = \diamond(\diamond)$$

$$u(t-T_1) * u(t-T_2) = r(t-T_1-T_2) \quad \diamond(\diamond - \diamond 1) * \diamond(\diamond - \diamond 2) = \diamond(\diamond - \diamond 1 - \diamond 2)$$

$$u[n] * u[n] = [n+1]u[n] \quad \diamond(\diamond) * \diamond(\diamond) = [\diamond+1] \diamond(\diamond)$$

Scaling Property

$$\text{If } x(t) * h(t) = y(t) \quad \diamond(\diamond) * h(\diamond) = \diamond(\diamond)$$

$$\text{then } x(at) * h(at) = \frac{1}{|a|} y(at) \quad \diamond(\diamond \diamond) * h(\diamond \diamond) = \frac{1}{| \diamond |} \diamond(\diamond \diamond)$$

Differentiation of Output

$$\text{if } y(t) = x(t) * h(t) \quad \diamond(\diamond) = \diamond(\diamond) * h(\diamond)$$

$$\text{then } \frac{dy(t)}{dt} = \frac{dx(t)}{dt} * h(t) \quad \diamond \diamond(\diamond) \diamond \diamond = \diamond \diamond(\diamond) \diamond \diamond * h(\diamond)$$

or

$$\frac{dy(t)}{dt} = x(t) * \frac{dh(t)}{dt} \quad \diamond \diamond(\diamond) \diamond \diamond = \diamond(\diamond) * \diamond h(\diamond) \diamond \diamond$$

Note:

- Convolution of two causal sequences is causal.
- Convolution of two anti causal sequences is anti causal.
- Convolution of two unequal length rectangles results a trapezium.
- Convolution of two equal le

Pooling Layers

The pooling operation involves sliding a two-dimensional filter over each channel of feature map and summarising the features lying within the region covered by the filter.

For a feature map having dimensions $n_h \times n_w \times n_c$, the dimensions of output obtained after a pooling layer is

$$(n_h - f + 1) / s \times (n_w - f + 1) / s \times n_c$$

where,

- > n_h - height of feature map
- > n_w - width of feature map
- > n_c - number of channels in the feature map
- > f - size of filter
- > s - stride length

A common CNN model architecture is to have a number of convolution and pooling layers stacked one after the other.

Why to use Pooling Layers?

- Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.
- The pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

Types of Pooling Layers:

Max Pooling

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

Transfer Learning

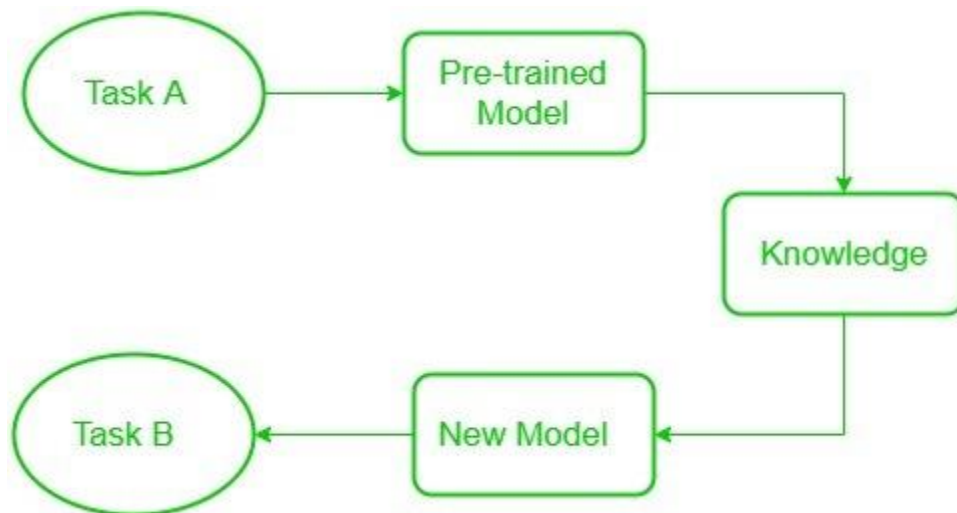
Many deep neural networks trained on images have a curious phenomenon in common: in the early layers of the network, a deep learning model tries to learn a low level of features, like detecting edges, colours, variations of intensities, etc. Such kind of features appears not to be specific to a particular dataset or a task because no matter what type of image we are processing either for detecting a lion or car. In both cases, we have to detect these low-level features. All these features occur regardless of the exact cost function or image dataset. Thus learning these features in one task of detecting lion can be used in other tasks like detecting humans.

Necessity for transfer learning: Low-level features learned for task A should be beneficial for learning of model for task B.

This is what transfer learning is. Nowadays, it is very hard to see people training a whole convolutional neural networks from scratch, and it is common to use a pre-trained model trained on a variety of

images in a similar task, e.g models trained on ImageNet (1.2 million images with 1000.

The Block diagram is shown below as follows:



categories), and use features from them to solve a new task. When dealing with transfer learning, we come across a phenomenon called freezing of layers. A layer, it can be a CNN layer, hidden layer, a block of layers, or any subset of a set of all layers, is said to be fixed when it is no longer available to train. Hence, the weights of freezed layers will not be updated during training. While layers that are not freezed follows regular training procedure. When we use transfer learning in solving a problem, we select a pre-trained model as our base model. Now, there are two possible approaches to use knowledge from the pre-trained model. First way is to freeze a few layers of pre-trained model and train other layers on our new dataset for the new task. Second way is to make a new model, but also take out some features from the layers in the pre-trained model and use them in a newly created model. In both cases, we take out some of the learned features and try to train the rest of the model. This makes sure that the only feature that may be same in both of the tasks is taken out from the pre-trained model, and the rest of the model is changed to fit new dataset by training.

5. Image classification using transfer learning

Trans

Image classification is one of the supervised machine learning problems which aims to categorize the images of a dataset into their respective categories or labels. Classification of images of various dog breeds is a classic image classification problem. So, we have to classify more than one class that's why the name multi-class classification, and in this article, we will be doing the same **fer**

learning: Transfer learning is a popular deep learning method that follows the approach of using the knowledge that was learned in some task and applying it to solve the problem of the related target task. So, instead of creating a neural network from scratch we “**transfer**” the learned features which are basically the “**weights**” of the network. To implement the concept of transfer learning, we make use of “**pre-trained models**”.

Necessities for transfer learning: Low-level features from model A (task A) should be helpful for learning model B (task B).

Pre-trained model: Pre-trained models are the deep learning models which are trained on very large datasets, developed, and are made available by other developers who want to contribute to this machine learning community to solve similar types of problems. It contains the biases and weights of the neural network representing the features of the dataset it was trained on. The features learned are always transferrable. For example, a model trained on a large dataset of flower images will contain learned features such as corners, edges, shape, color, etc.

InceptionResNetV2: InceptionResNetV2 is a convolutional neural network that is 164 layers deep, trained on millions of images from the ImageNet database, and can classify images into more than 1000 categories such as flowers, animals, etc. The input size of the images is 299-by-299.

short questions

- 1.what is Convolutional neural networks?
- 2.Explain Convolution?
- 3.Explain Transfer Learning?

Long Questions

- 1.Explain about CNN Architectures?
- 2.Explain about pooling Layers?
- 3.Explain Image Classification using Transfer Learning?

Unit-4

LSTM

LSTM networks are an extension of recurrent neural networks (RNNs) mainly introduced to handle situations where RNNs fail. Talking about

RNN, it is a network that works on the present input by taking into consideration the previous output (feedback) and storing in its memory for a short period of time (short-term memory). Out of its various applications, the most popular ones are in the fields of speech processing, non-Markovian control, and music composition. Nevertheless, there are drawbacks to RNNs. First, it fails to store information for a longer period of time. At times, a reference to certain information stored quite a long time ago is required to predict the current output. But RNNs are absolutely incapable of handling such “long-term dependencies”. Second, there is no finer control over which part of the context needs to be carried forward and how much of the past needs to be ‘forgotten’. Other issues with RNNs are exploding and vanishing gradients (explained later) which occur during the training process of a network through backtracking. Thus, Long Short-Term Memory (LSTM) was brought into the picture. It has been so designed that the vanishing gradient problem is almost completely removed, while the training model is left unaltered. Long time lags in certain problems are bridged using LSTMs where they also handle noise, distributed representations, and continuous values. With LSTMs, there is no need to keep a finite number of states from beforehand as required in the hidden Markov model (HMM). LSTMs provide us with a large range of parameters such as learning rates, and input and output biases. Hence, no need for fine adjustments. The complexity to update each weight is reduced to $O(1)$ with LSTMs, similar to that of Back Propagation Through Time (BPTT), which is an advantage.

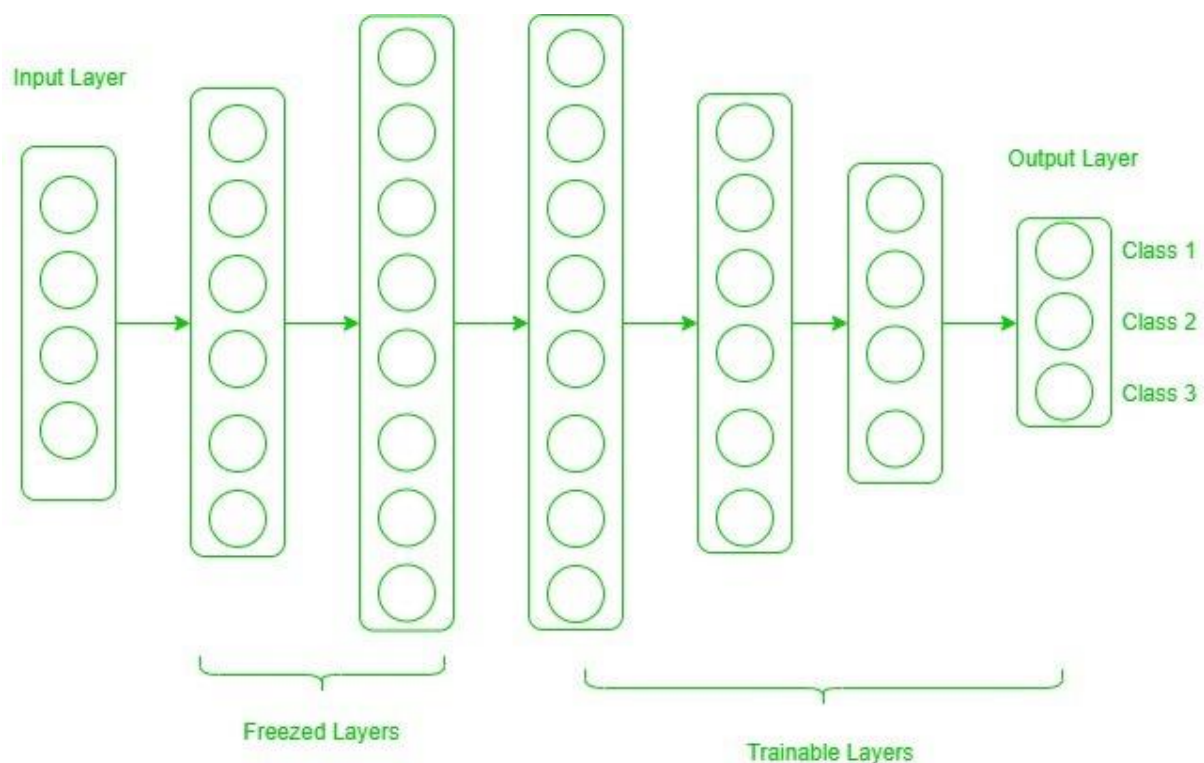
Exploding and Vanishing Gradients:

During the training process of a network, the main goal is to minimize loss (in terms of error or cost) observed in the output when training data is sent through it. We calculate the gradient, that is, loss with respect to a particular set of weights, adjust the weights accordingly and repeat this process until we get an optimal set of weights for which loss is minimum. This is the concept of backtracking. Sometimes, it so happens that the gradient is almost negligible. It must be noted that the gradient of a layer depends on certain components in the successive layers. If some of these components are small (less than 1), the result obtained, which is the gradient, will be even smaller. This is known as the scaling effect. When this gradient is multiplied with the learning rate which is in itself a small value ranging between 0.1-0.001, it results in a smaller value. As a consequence, the alteration in weights is quite small, producing almost the same output as before. Similarly, if the gradients are quite large in value due to the large values of components, the weights get updated to a value beyond the optimal

value. This is known as the problem of exploding gradients. To avoid this scaling effect, the neural network unit was re-built in such a way that the scaling factor was fixed to one. The cell was then enriched by several gating units and was called LSTM.

Architecture:

The basic difference between the architectures of RNNs and LSTMs is that the hidden layer of LSTM is a gated unit or gated cell. It consists of four layers that interact with one another in a way to produce the output of that cell along with the cell state. These two things are then passed onto the next hidden layer. Unlike RNNs which have got the only single neural net layer of tanh, LSTMs comprises of three logistic sigmoid gates and one tanh layer. Gates have been introduced in order to limit the information that is passed through the cell. They determine which part of the information will be needed by the next cell and which part is to be discarded. The output is usually in the range of 0-1 where '0' means 'reject all' and '1' means 'include all'.



Now, one may ask how to determine which layers we need to freeze and which layers need to train. The answer is simple, the more you want to inherit features from a pre-trained model, the more you have to

freeze layers. For instance, if the pre-trained model detects some flower species and we need to detect some new species. In such a case, a new dataset with new species contains a lot of features similar to the pre-trained model. Thus, we freeze less number of layers so that we can use most of its knowledge in a new model. Now, consider another case, if there is a pre-trained model which detects humans in images, and we want to use that knowledge to detect cars, in such a case where dataset is entirely different, it is not good to freeze lots of layers because freezing a large number of layers will not only give low level features but also give high-level features like nose, eyes, etc which are useless for new dataset (car detection). Thus, we only copy low-level features from the base network and train the entire network on a new dataset. Let's consider all situations where the size and dataset of the target task vary from the base network.

Target dataset is small and similar to the base network

dataset: Since the target dataset is small, that means we can fine-tune the pre-trained network with target dataset. But this may lead to a problem of overfitting. Also, there may be some changes in the number of classes in the target task. So, in such a case we remove the fully connected layers from the end, maybe one or two, and add a new fully-connected layer satisfying the number of new classes. Now, we freeze the rest of the model and only

GRU

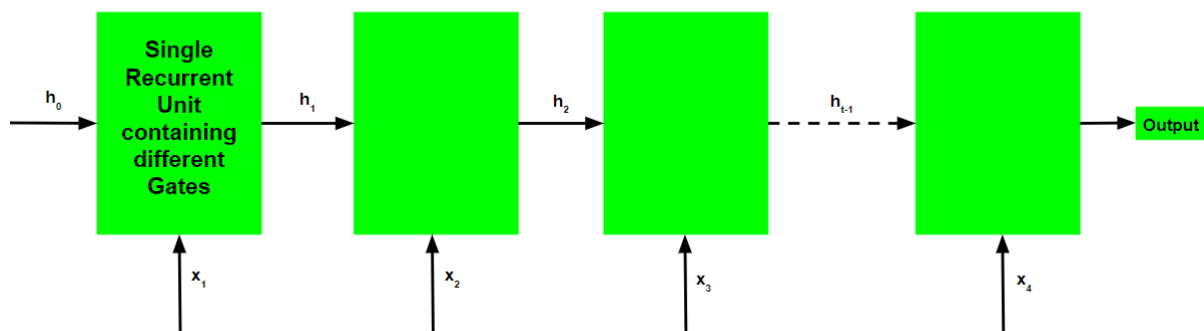
One of the lesser-known but equally effective variations is the **Gated Recurrent Unit Network (GRU)**.

Unlike LSTM, it consists of only three gates and does not maintain an Internal Cell State. The information which is stored in the Internal Cell State in an LSTM recurrent unit is incorporated into the hidden state of the Gated Recurrent Unit. This collective information is passed onto the next Gated Recurrent Unit. The different gates of a GRU are as described below:-

1. **Update Gate(z):** It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the Output Gate in an LSTM recurrent unit.
2. **Reset Gate(r):** It determines how much of the past knowledge to forget. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.
3. **Current Memory Gate():** It is often overlooked during a typical discussion on Gated Recurrent Unit Network. It is incorporated into the Reset Gate just like the Input Modulation

Gate is a sub-part of the Input Gate and is used to introduce some non-linearity into the input and to also make the input Zero-mean. Another reason to make it a sub-part of the Reset gate is to reduce the effect that previous information has on the current information that is being passed into the future.

The basic work-flow of a Gated Recurrent Unit Network is similar to that of a basic Recurrent Neural Network when illustrated, the main difference between the two is in the internal working within each recurrent unit as Gated Recurrent Unit networks consist of gates which modulate the current input and the previous hidden state.



Working of a Gated Recurrent Unit:

- Take input the current input and the previous hidden state as vectors.
- Calculate the values of the three different gates by following the steps given below:-
 1. For each gate, calculate the parameterized current input and previously hidden state vectors by performing element-wise multiplication (Hadamard Product) between the concerned vector and the respective weights for each gate.
 2. Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.
- The process of calculating the Current Memory Gate is a little different. First, the Hadamard product of the Reset Gate and the previously hidden state vector is calculated. Then this vector is parameterized and then added to the parameterized current input vector.

- To calculate the current hidden state, first, a vector of ones and the same dimensions as that of the input is defined. This vector will be called ones and mathematically be denoted by $\mathbf{1}$. First, calculate the Hadamard Product of the update gate and the previously hidden state vector. Then generate a new vector by subtracting the update gate from ones and then calculate the Hadamard Product of the newly generated vector with the current memory gate. Finally, add the two vectors to get the currently hidden state vector.

Just like Recurrent Neural Networks, a GRU network also generates an output at each time step and this output is used to train the network using gradient descent.

Note that just like the workflow, the training process for a GRU network is also diagrammatically similar to that of a basic Recurrent Neural Network and differs only in the internal working of each recurrent unit.

The Back-Propagation Through Time Algorithm for a Gated Recurrent Unit Network is similar to that of a Long Short Term Memory Network and differs only in the differential chain formation.

Encoder/Decoder Architecture

The encoder-decoder architecture for recurrent neural networks is the standard neural machine translation method that rivals and in some cases outperforms classical statistical machine translation methods. This architecture is very new, having only been pioneered in 2014, although, has been adopted as the core technology inside Google's translate service.

In this post, you will discover the two seminal examples of the encoder-decoder model for neural machine translation.

After reading this post, you will know:

- The encoder-decoder recurrent neural network architecture is the core technology inside Google's translate service.
- The so-called "*Sutskever model*" for direct end-to-end machine translation.
- The so-called "*Cho model*" that extends the architecture with GRU units and an attention mechanism.

Encoder-Decoder Architecture for NMT

The Encoder-Decoder architecture with recurrent neural networks has become an effective and standard approach for both neural machine translation (NMT) and sequence-to-sequence (seq2seq) prediction in general.

The key benefits of the approach are the ability to train a single end-to-end model directly on source and target sentences and the ability to handle variable length input and output sequences of text.

As evidence of the success of the method, the architecture is the *Our model follows the common sequence-to-sequence learning framework with attention. It has three components: an encoder network, a decoder network, and an attention network.*

, 2016

In this post, we will take a closer look at two different research projects that developed the same Encoder-Decoder architecture at the same time in 2014 and achieved results that put the spotlight on the approach. They are:

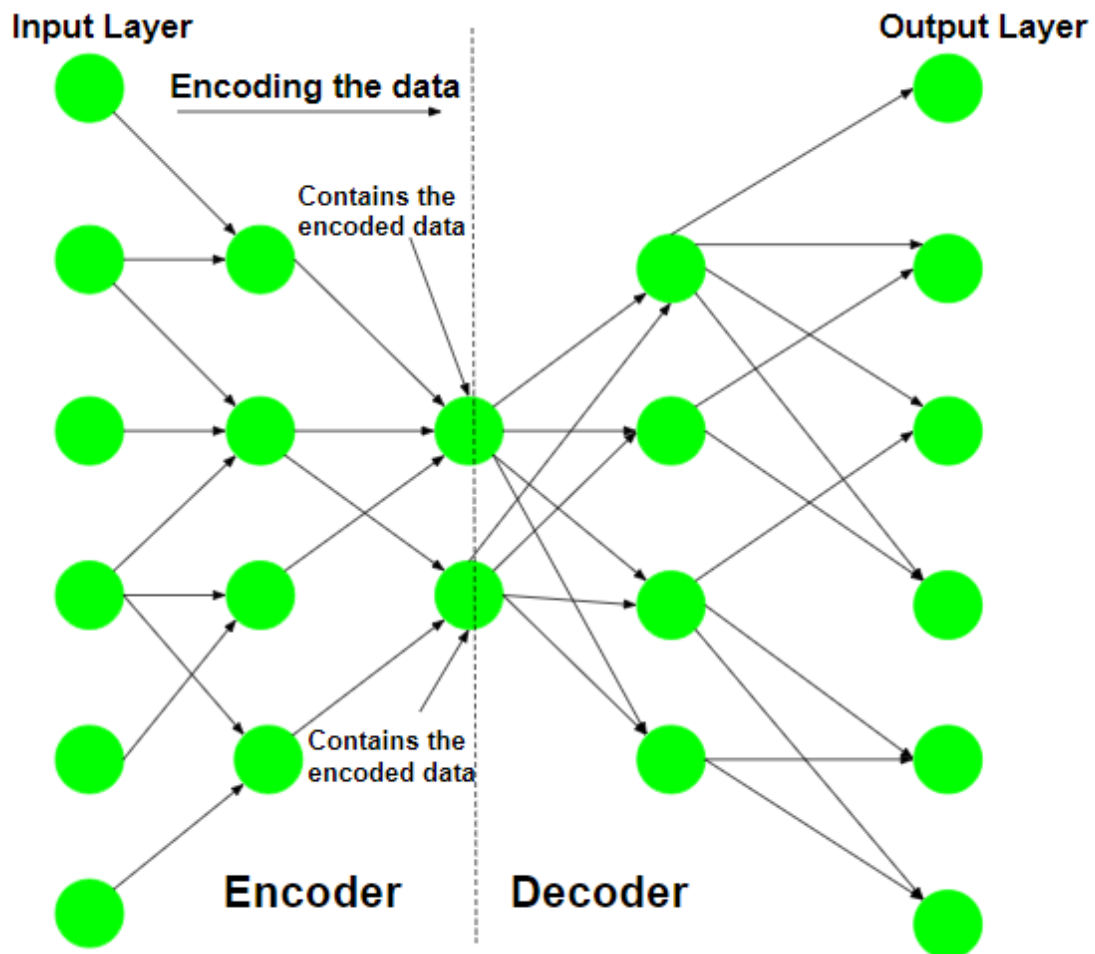
Autoencoders

The encoder part of the network is used for **encoding and sometimes even for data compression** purposes although it is **not very effective as compared to other general compression techniques like JPEG**. Encoding is achieved by the encoder part of the network which has a **decreasing number of hidden units** in each layer. Thus this part is forced to pick up only the most significant and representative features of the data. The second half of the network performs the **Decoding function**. This part has **an increasing number of hidden units** in each layer and thus tries to reconstruct the original input from the encoded data. Thus Auto-encoders are an **unsupervised learning technique**.

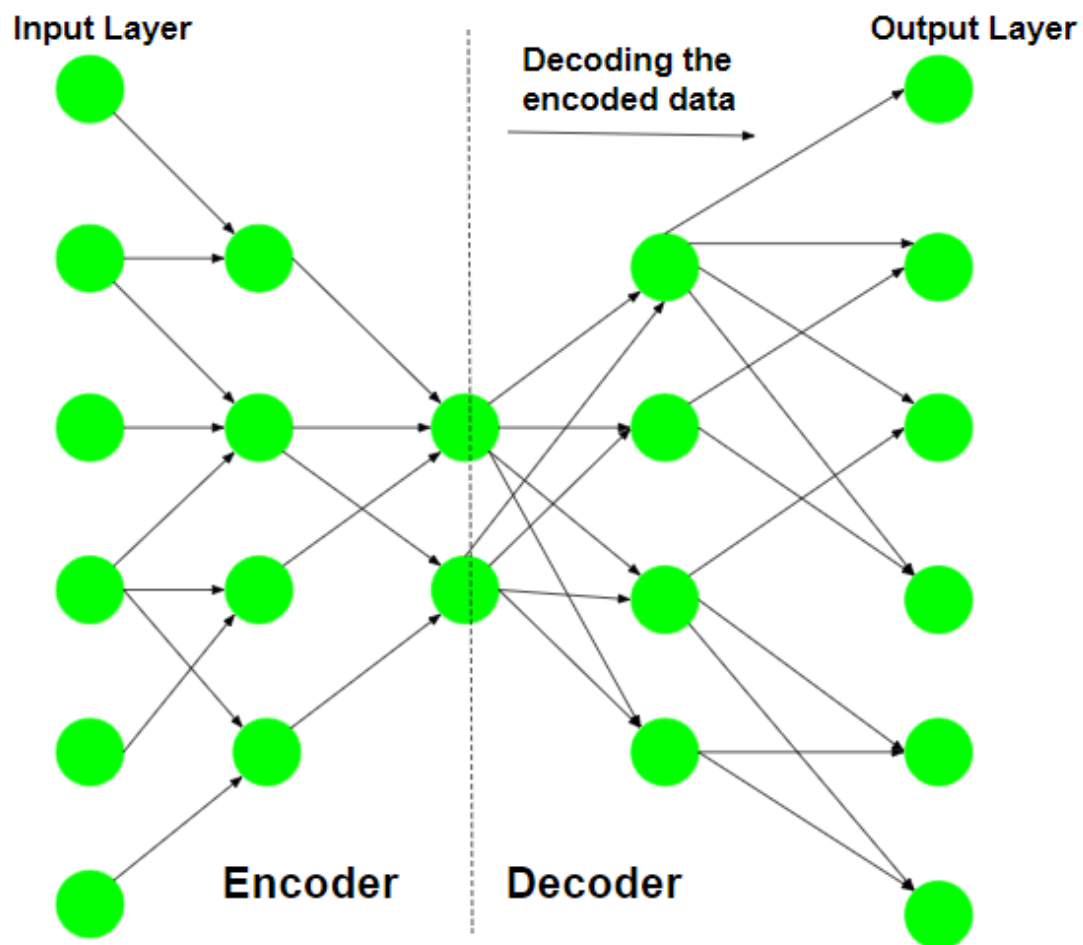
Example: See the below code, in autoencoder training data, is fitted to itself. That's why instead of fitting `X_train` to `Y_train` we have used `X_train` in both places.

Training of an Auto-encoder for data compression: For a data compression procedure, the most important aspect of the compression

is the reliability of the reconstruction of the compressed data. This requirement dictates the structure of the Auto-encoder as a bottleneck. **Step 1: Encoding the input data** The Auto-encoder first tries to encode the data using the initialized weights and biases.



Step 2: Decoding the input data The Auto-encoder tries to reconstruct the original input from the encoded data to test the reliability of the encoding.



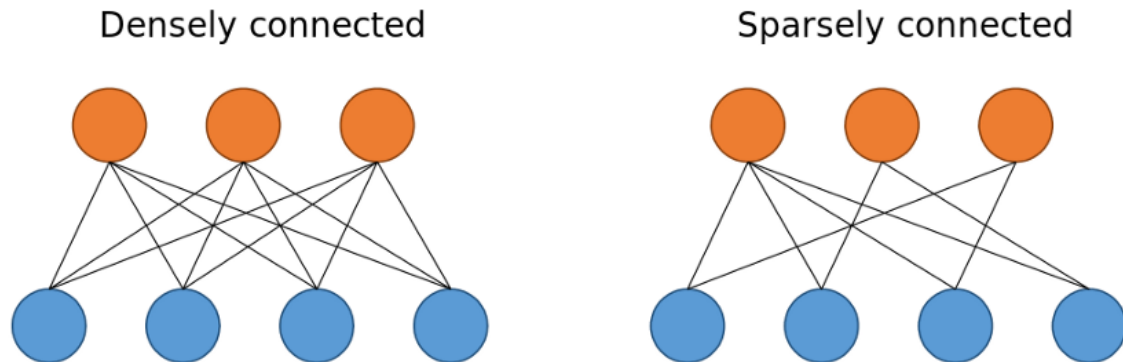
Step 3: Backpropagating the error After the reconstruction, the loss function is computed to determine the reliability of the encoding. The error generated is backpropagated.

sparse

Sparse expert models are a thirty-year old concept re-emerging as a popular architecture in deep learning. This class of architecture encompasses Mixture-of-Experts, Switch Transformers, Routing Networks, BASE layers, and others, all with the unifying idea that each example is acted on by a subset of the parameter

Is it absolutely necessary that each node in one layer be connected to each layer in the next? The answer turns out to be “no.”

We can easily imagine a layer that does not connect every node exhaustively. In the research, this sort of layer is often called a Sparsely Connected Layer (SCL).



lot more like this. For example, returning to the original inspiration for artificial neural networks (the brain), neurons (analogous to nodes) are only connected to handful of other neurons.

The above-described training process is reiterated several times until an acceptable level of reconstruction is reached.

After the training process, only the encoder part of the Auto-encoder is retained to encode a similar type of data used in the training process. The different ways to constrain the network are:-

- **Keep small Hidden Layers:** If the size of each hidden layer is kept as small as possible, then the network will be forced to pick up only the representative features of the data thus encoding the data.
- **Regularization:** In this method, **a loss term is added** to the cost function which encourages the network to train in ways other than copying the input.
- **Denoising:** Another way of constraining the network is to **add noise to the input** and teach the network how to remove the noise from the data.
- **Tuning the Activation Functions:** This method involves **changing the activation functions of various nodes so that a majority of the nodes are dormant** thus effectively reducing the size of the hidden layers.

The different variations of Auto-encoders are:-

- **Denoising Auto-encoder:** This type of auto-encoder works on a partially corrupted input and trains to recover the original undistorted image. As mentioned above, this method is an effective way to constrain the network from simply copying the input.
- **Sparse Auto-encoder:** This type of auto-encoder typically contains more hidden units than the input but only a few are allowed to be active at once. This property is called the sparsity of the network. The sparsity of the network can be controlled by either manually zeroing the required hidden units, tuning the activation functions or by adding a loss term to the cost function.
- **Variational Auto-encoder:** This type of auto-encoder makes strong assumptions about the distribution of latent variables and uses the **Stochastic Gradient Variational Bayes** estimator in the training process. It assumes that the data is generated by a **Directed Graphical Model** and tries to learn an approximation to $p(x)$ to the conditional property $p(x|z)$ where θ and ϕ are the parameters of the encoder and the decoder respectively.

7.Denoising

Denoising an image is a classical problem that researchers are trying to solve for decades. In earlier times, researchers used *filters* to reduce the noise in the images. They used to work fairly well for images with a reasonable level of noise. However, applying those filters would add a blur to the image. And if the image is too noisy, then the resultant image would be so blurry that most of the critical details in the image are lost.

There has to be a better way to solve this problem. As a result, I have implemented several deep learning architectures that far surpass the traditional denoising filters. In this blog, I will explain my approach step-by-step as a case study, starting from the problem formulation to implementing the state-of-the-art deep learning models, and then finally see the results.

Contents Summary

1. What is noise in images?

2. Problem Formulation
3. Machine Learning Problem Formulation
4. Source of Data
5. Exploratory Data Analysis
6. An Overview on Traditional Filters for Image Denoising
7. Deep Learning Models for Image Denoising
8. Results Comparison
9. Deployment
10. Future Work and Scope for Improvement
11. References

Contractive

Contractive Autoencoder was proposed by the researchers at the University of Toronto in 2011 in the paper Contractive auto-encoders: Explicit invariance during feature extraction. The idea behind that is to make the autoencoders robust of small changes in the training dataset.

To deal with the above challenge that is posed in basic autoencoders, the authors proposed to add another penalty term to the loss function of autoencoders. We will discuss this loss function in details.

The Loss function:

Contractive autoencoder adds an extra term in the loss function of autoencoder, it is given as:

Variational AutoEncoders

Variational autoencoder was proposed in 2013 by Knigam and Welling at Google and Qualcomm. A variational autoencoder (VAE) provides a probabilistic manner for describing an observation in latent space. Thus, rather than building an encoder that outputs a single value to describe each latent state attribute, we'll formulate our encoder to describe a probability distribution for each latent attribute.

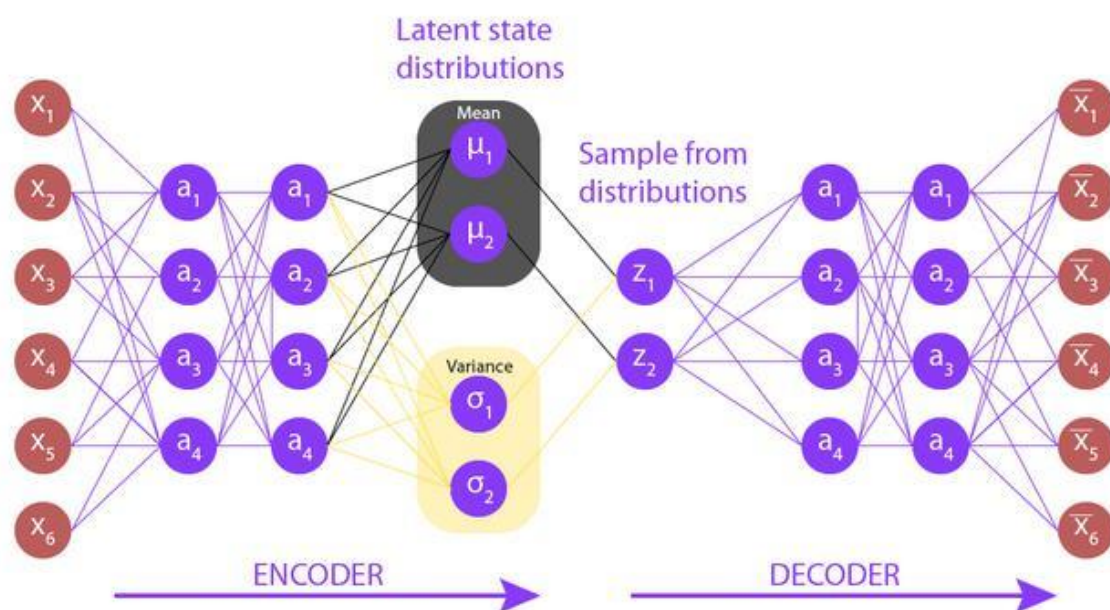
It has many applications such as data compression, synthetic data creation etc.

Architecture:

Autoencoders are a type of neural network that learns the data encodings from the dataset in an unsupervised way. It basically contains two parts: the first one is an encoder which is similar to the

convolution neural network except for the last layer. The aim of the encoder to learn efficient data encoding from the dataset and pass it into a bottleneck architecture. The other part of the autoencoder is a decoder that uses latent space in the bottleneck layer to regenerate the images similar to the dataset. These results backpropagate from the neural network in the form of the loss function.

Variational autoencoder is different from autoencoder in a way such that it provides a statistic manner for describing the samples of the dataset in latent space. Therefore, in variational autoencoder, the encoder outputs a probability distribution in the bottleneck layer instead of a single output value.



Variational autoencoder uses KL-divergence as its loss function, the goal of this is to minimize the difference between a supposed distribution and original distribution of dataset.

Suppose we have a distribution z and we want to generate the observation x from it. In other words, we want to calculate

We can do it by following way:

But, the calculation of $p(x)$ can be quite difficult

This usually makes it an intractable distribution. Hence, we need to approximate $p(z|x)$ to $q(z|x)$ to make it a tractable distribution. To better approximate $p(z|x)$ to $q(z|x)$, we will minimize the KL-divergence loss which calculates how similar two distributions are:

By simplifying, the above minimization problem is equivalent to the following maximization problem :

The first term represents the reconstruction likelihood and the other term ensures that our learned distribution q is similar to the true prior distribution p .

Thus our total loss consists of two terms, one is reconstruction error and other is KL-divergence loss:

Implementation:

In this implementation, we will be using the Fashion-MNIST dataset, this dataset is already available in keras.datasets API, so we don't need to add or upload manually

10. Adversarial Generative Networks

A Generative Adversarial Network (GAN) is a deep learning architecture that consists of two neural networks competing against each other in a zero-sum game framework. The goal of GANs is to generate new, synthetic data that resembles some known data distribution.

1.Components:

- Generator network: creates synthetic data
- Discriminator network: evaluates the synthetic data and tries to determine if it's real or fake

2.Training:

- The generator network produces synthetic data and the discriminator network evaluates it.
- The generator is trained to fool the discriminator and the discriminator is trained to correctly identify real and fake data.
- This process continues until the generator produces data that is indistinguishable from real data.

3.Applications:

- Image synthesis
- Text-to-Image synthesis
- Image-to-Image translation
- Anomaly detection
- Data augmentation

4.Limitations:

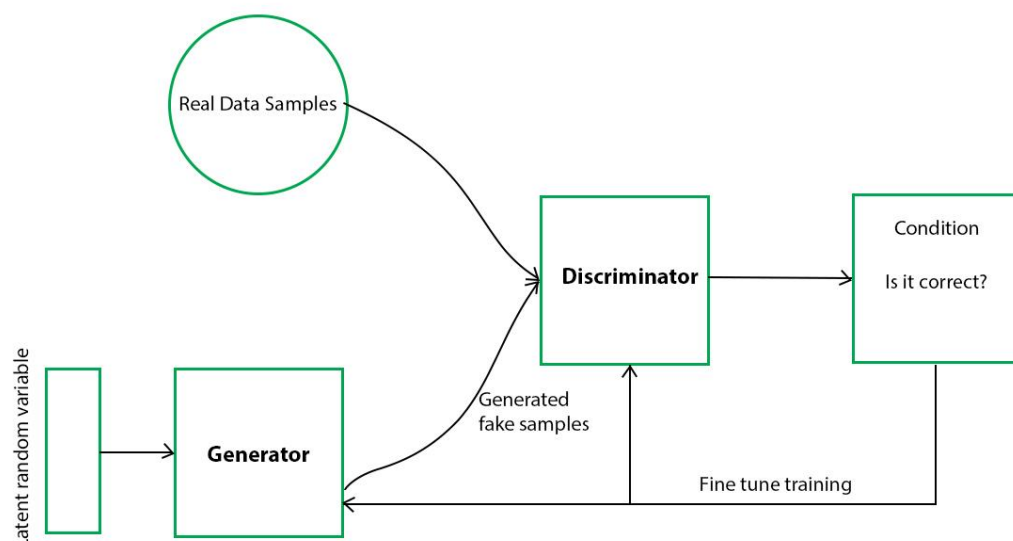
- Training can be unstable and prone to mode collapse, where the generator produces limited variations of synthetic data.
- GANs can be difficult to train and require a lot of computational resources.
- GANs can generate unrealistic or irrelevant synthetic data if the generator and discriminator are not properly trained.

Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used. It was developed and introduced by Ian J. Goodfellow in 2014. GANs are basically made up of a system of two competing neural network models which compete with each other and are able to analyze, capture and copy the variations within a dataset. **Why were GANs developed in the first place?** It has been noticed most of the mainstream neural nets can be easily fooled into misclassifying things by adding only a small amount of noise into the original data. Surprisingly, the model after adding noise has higher confidence in the wrong prediction than when it predicted correctly. The reason for such adversary is that most machine learning models learn

from a limited amount of data, which is a huge drawback, as it is prone to overfitting. Also, the mapping between the input and the output is almost linear. Although, it may seem that the boundaries of separation between the various classes are linear, but in reality, they are composed of linearities and even a small change in a point in the feature space might lead to misclassification of data. **How does GANs work?** Generative Adversarial Networks (GANs) can be broken down into three parts:

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

In GANs, there is a **generator** and a **discriminator**. The Generator generates fake samples of data (be it an image, audio, etc.) and tries to fool the Discriminator. The Discriminator, on the other hand, tries to distinguish between the real and fake samples. The Generator and the Discriminator are both Neural Networks and they both run in competition with each other in the training phase. The steps are repeated several times and in this, the Generator and Discriminator get better and better in their respective jobs after each repetition. The working can be visualized by the diagram given below:



Here, the generative model captures the distribution of data and is trained in such a manner that it tries to maximize the probability of the Discriminator in making a mistake. The Discriminator, on the other

hand, is based on a model that estimates the probability that the sample that it got is received from the training data and not from the Generator. The GANs are formulated as a minimax game, where the Discriminator is trying to minimize its reward $V(\mathbf{D}, \mathbf{G})$ and the Generator is trying to minimize the Discriminator's reward or in other words, maximize its loss. It can be mathematically described by the formula below:

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

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

where, G = Generator D = Discriminator $P_{data}(x)$ = distribution of real data $P(z)$ = distribution of generator x = sample from $P_{data}(x)$ z = sample from $P(z)$ $D(x)$ = Discriminator network $G(z)$ = Generator network So, basically, training a GAN has two parts:

Autoencoder and DBM

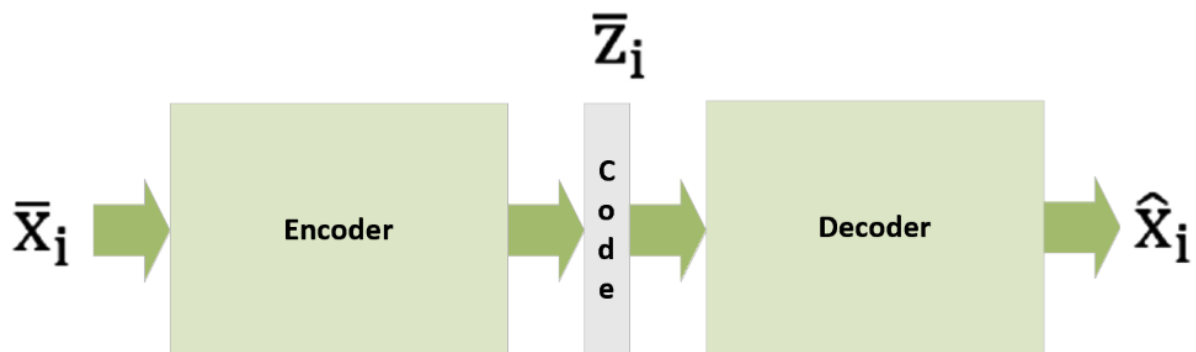
Autoencoders are the models in a dataset that find low-dimensional representations by exploiting the extreme non-linearity of neural networks. An autoencoder is made up of two parts:

Encoder – This transforms the input (high-dimensional into a code that is crisp and short.

Decoder – This transforms the shortcode into a high-dimensional input.

Assume that from a data-generating process, $p_{data}(x)$, if X is a set of samples drawn. Suppose $x_i \gg n$; however, do not keep any restrictions on the support structure. An example of this is, for RGB images, $x_i \gg n \times m \times 3$.

Here is a simple illustration of a generic autoencoder:



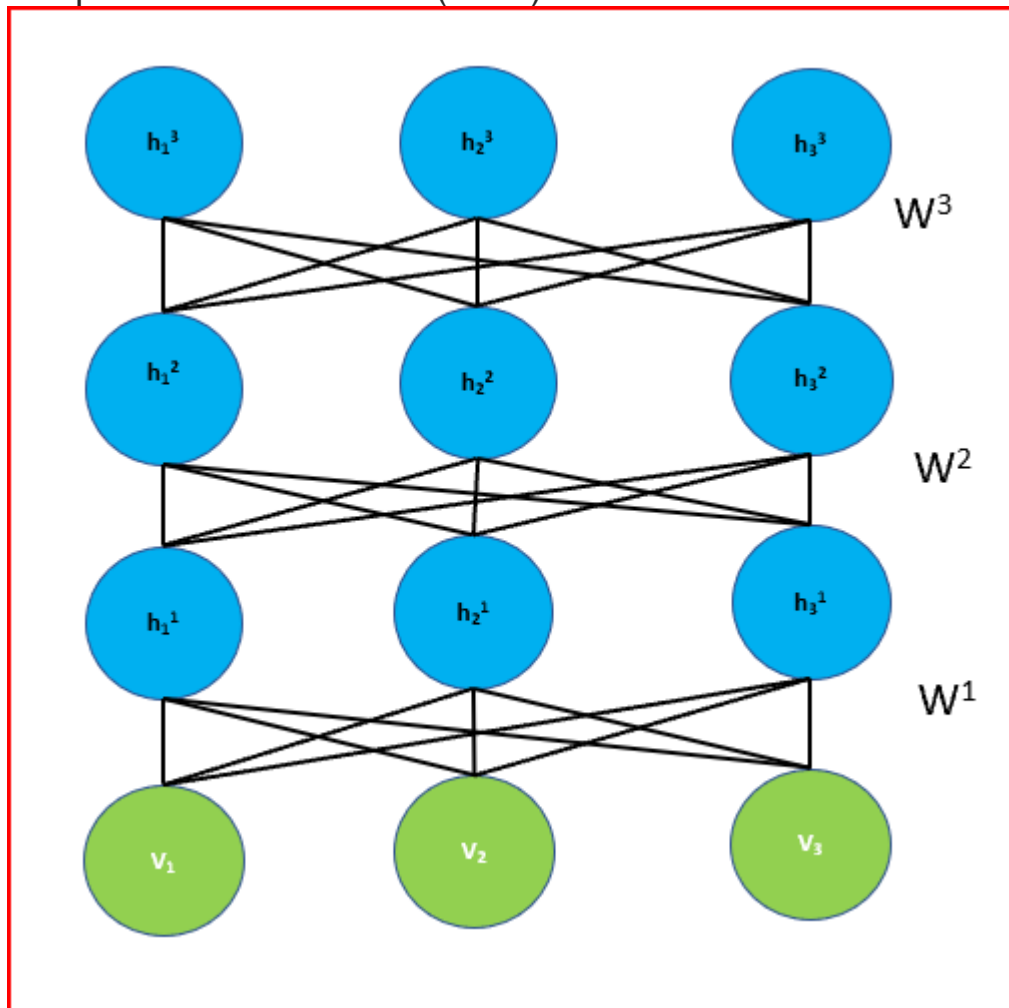
For a p -dimensional vector code, a parameterized function, $e(\bullet)$ is the definition of the encoder:

$$\bar{z}_i = e(\bar{x}_i, \bar{\theta}_e) \text{ where } \bar{x}_i \in \mathbb{R}^n \text{ and } \bar{z}_i \in \mathbb{R}^p$$

In an analogous way, the decoder is another parameterized function, $d(\cdot)$:

$$\hat{x}_i = d(\bar{z}_i, \bar{\theta}_d) \text{ where } \hat{x}_i \in \mathbb{R}^n \text{ and } \bar{z}_i \in \mathbb{R}^p$$

Deep Boltzmann Machine(DBM)?



Deep Boltzmann Machine

- Unsupervised, probabilistic, generative model with entirely undirected connections between different layers
- Contains visible units and multiple layers of hidden units
- Like RBM, no intralayer connection exists in DBM. Connections exists only between units of the neighboring layers

- Network of symmetrically connected stochastic binary units
- DBM can be organized as bipartite graph with odd layers on one side and even layers on one side
- Units within the layers are independent of each other but are dependent on neighboring layers
- Learning is made efficient by layer by layer pre training — Greedy layer wise pre training slightly different than done in DBM
- After learning the binary features in each layer, DBM is fine tuned by back propagation.

Sounds similar to DBN so what is the difference between Deep Belief networks(DBN) and Deep Boltzmann Machine(DBM)?

Let's talk first about similarity between DBN and DBM and then difference between DBN and DBM

Short questions

- 1.what is LSTM?
- 2.Difference between encoder and decoder?
- 3.Explain Sparse?
- 4.What is Denoising?
- 5.Explain Contractive in Deep Learning?

Long Questions

- 1.Explain Briefly about Variational Autoencoders?
- 2.What is Adversarial Generative Networks?
- 3.Define GRU in detail?
- 4.Explain Autoencoder and DBM?
- 5.Explain Encoder - Decoder Architecture?

Unit-5

Object detection

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos.^[1] Well-researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance.

Detection of objects on a road

It is widely used in computer vision tasks such as image annotation,^[2] vehicle counting,^[3] activity recognition,^[4] face detection, face recognition, video object co-segmentation. It is also used in tracking objects, for example tracking a ball during a football match, tracking movement of a cricket bat, or tracking a person in a video.

Often, the test images are sampled from a different data distribution, making the object detection task significantly more difficult.^[5] To address the challenges caused by the domain gap between training and test data, many unsupervised domain adaptation approaches have been proposed.^{[5][6][7][8][9]} A simple and straightforward solution of reducing the domain gap is to apply an image-to-image translation approach, such as cycle-GAN.^[10] Among other uses, cross-domain object detection is applied in autonomous driving, where models can be trained on a vast

amount of video game scenes, since the labels can be generated without manual labor.

Concept[edit]

Every object class has its own special features that help in classifying the class – for example all circles are round. Object class detection uses these special features. For example, when looking for circles, objects that are at a particular distance from a point (i.e. the center) are sought. Similarly, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed. A similar approach is used for face identification

Automatic Image captioning

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos.^[1] Well-researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval.

Detection of objects on a road

It is widely used in computer vision tasks such as image annotation,^[2] vehicle counting,^[3] activity recognition,^[4] face detection, face recognition, video object co-segmentation. It is also used in tracking objects, for example tracking a ball during a football match, tracking movement of a cricket bat, or tracking a person in a video.

Often, the test images are sampled from a different data distribution, making the object detection task significantly more difficult.^[5] To address the challenges caused by the domain gap between training and test data, many unsupervised domain adaptation approaches have been proposed.^{[5][6][7][8][9]} A simple and straightforward solution of reducing the domain gap is to apply an image-to-image translation approach, such as cycle-GAN.^[10] Among other uses, cross-domain object detection is applied in autonomous driving, where models can be trained on a vast amount of video game scenes, since the labels can be generated without manual labor.

Every object class has its own special features that help in classifying the class – for example all circles are round. Object class detection uses these special features. For example, when looking for circles, objects

that are at a particular distance from a point (i.e. the center) are sought. Similarly, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed. A similar approach is used for face identification

Image Generation with generative adversarial networks

Generative Adversarial Network which is popularly known as GANs is a deep learning, unsupervised machine learning technique which is proposed in year 2014 through this research paper. The main blocks of this architecture are ;

1. **Generator** : This block tries to generate the images which are very similar to that of original dataset by taking noise as input. It tries to learn the joint probability of the input data (X) and output data(Y); $P(X|Y)$.
2. **Discriminator** : This block tries to accept two inputs, one from main dataset and other from images generated from Generator, and bifurcates them as Real or Fake.

To make this Generative and Adversarial process simple, both these blocks are made from Deep Neural Network based architecture which can be trained through forward and backward propagation techniques.

From the time GANs were introduced, there has been tremendous advancement in the GANs. There are GAN architectures which are specifically made for some tasks.

- Image Generation using Deep Convolution GANs.
- Generation of Anime Characters using GANs.

- Sketch to Color photograph generation using GANs.
- Unpaired Image-to-Image translation using CycleGANs.
- Text-to-Image Synthesis with Stacked GAN.
- Generation of new Human Poses using GANs.
- Single Image Super Resolution using GANs.
- GAN based Inpainting of photographs.

To understand this concept in-depth, we will implement GAN architectures through tensorflow-keras. We will be focusing on generation of MNIST images through **simple GANs** and also through **Deep Convolution GANs**

Video to Text with LSTM Models

Generative Adversarial Network which is popularly known as GANs is a deep learning, unsupervised machine learning technique which is proposed in year 2014 through [this research paper](#). The main blocks of this architecture are ;

1. **Generator :** This block tries to generate the images which are very similar to that of original dataset by taking noise as input. It tries to learn the joint probability of the input data (X) and output data(Y); $P(X|Y)$.
2. **Discriminator :** This block tries to accept two inputs, one from main dataset and other from images generated from Generator, and bifurcates them as Real or Fake.

To make this Generative and Adversarial process simple, both these block are made from Deep Neural Network based architecture which can be trained through forward and backward propagation techniques.

From the time GANs were introduced, there has been tremendous advancement in the GANs. There are GAN architecture which are specifically made for some tasks.

-

To understand this concept in-depth, we will implement GAN architectures through tensorflow-keras. We will be focusing on generation of MNIST images through **simple GANs** and also through **Deep Convolution GANs** and also the **Super Resolution GANs** with working example.

Attention models for ComputerVision

Attention models, or attention mechanisms, are input processing techniques for neural networks that allows the network to focus on specific aspects of a complex input, one at a time until the entire dataset is categorized. The goal is to break down complicated tasks into smaller areas of attention that are processed sequentially. Similar to how the human mind solves a new problem by dividing it into simpler tasks and solving them one by one.

Attention models require continuous reinforcement or backpropagation training to be effective.

How do Attention Models work?

In broad strokes, attention is expressed as a function that maps a query and “s set” of key value pairs to an output. One in which the query, keys, values, and final output are all vectors. The output is then calculated as a weighted sum of the values, with the weight assigned to each value expressed by a compatibility function of the query with the corresponding key value.

In practice, attention allows neural networks to approximate the visual attention mechanism humans use. Like people processing a new scene, the model studies a certain point of an image with intense, “high resolution” focus, while perceiving the surrounding areas in “low resolution,” then adjusts the focal point as the network begins to understand the scene.

What Types of Problems Do Attention Models Solve?

While this is a powerful technique for improving computer vision, the most work so far with attention mechanisms has focused on Neural Machine Translation (NMT). Traditional automated translation systems rely on massive libraries of data labeled with complex functions mapping each word’s statistical properties.

Using attention mechanisms in NMT is a much simpler approach. Here, the meaning of a sentence is mapped into a fixed-length vector, which then generates a translation based on that vector as a whole. The goal isn’t to translate the sentence word for word, but rather pay attention to the general, “high level” overall sentiment. Besides drastically improving accuracy, this attention-driven learning approach is much easier to construct and faster to train.

Named Entity Recognition

The named entity recognition (NER) is one of the most popular data preprocessing task. It involves the identification of key information in the text and classification into a set of predefined categories. An entity is basically the thing that is consistently talked about or refer to in the text.

NER is the form of NLP.

At its core, NLP is just a two-step process, below are the two steps that are involved:

- Detecting the entities from the text
- Classifying them into different categories

Some of the categories that are the most important architecture in NER such that:

- Person
- Organization
- Place/ location

Other common tasks include classifying of the following:

- date/time.
- expression
- Numeral measurement (money, percent, weight, etc)
- E-mail address

Ambiguity in NE

- For a person, the category definition is intuitively quite clear, but for computers, there is some ambiguity in classification. Let's look at some ambiguous example:
 - *England (Organisation)* won the 2019 world cup vs The 2019 world cup happened in *England(Location)*.
 - *Washington(Location)* is the capital of the US vs The first president of the US was *Washington(Person)*.

Methods of NER

- One way is to train the model for multi-class classification using different machine learning algorithms, but it requires a lot of labelling. In addition to labelling the model also requires a deep understanding of context to deal with the ambiguity of the sentences. This makes it a challenging task for a simple machine learning algorithm.
- Another way is that Conditional random field that is implemented by both NLP Speech Tagger and NLTK. It is a probabilistic model that can be used to model sequential data such as words. The CRF can capture a deep understanding of the context of the sentence. In this model, the input

- **Deep Learning Based NER:** deep learning NER is much more accurate than previous method, as it is capable to assemble words. This is due to the fact that it used a method called word embedding, that is capable of understanding the semantic and syntactic relationship between various words. It is also able to learn analyses topic-specific as well as high level words automatically. This makes deep learning NER applicable for performing multiple tasks. Deep learning can do most of the repetitive work itself, hence researchers for example can use their time more efficiently.

Opinion mining using Recurrent

.. The purpose of ASC is to see if user opinions stated in reviews/tweets are positive, negative, or neutral on particular aspects (aspect categories or aspect keywords). Aspect-based SA [5], [11] is a core task in SA research that is broken down into several subtasks: aspect extraction [12][13][14], opinion identification [15][16] and ASC [15][16][17][18][19]. Previous research [20][21] attempted to tackle these sub-tasks at the same time, with each sub-task receiving the majority of the research time. This research focuses on deep learning approaches to solving the ASC problem. ...

... The purpose of ASC is to see if user opinions stated in reviews/tweets are positive, negative, or neutral on particular aspects (aspect categories or aspect keywords). Aspect-based SA [5], [11] is a core task in SA research that is broken down into several subtasks: aspect extraction [12][13][14], opinion identification [15][16] and ASC [15][16][17][18][19]. Previous research [20][21] attempted to tackle these sub-tasks at the same time, with each sub-task receiving the majority of the research time. ...

... Given a phrase and an aspect, ASC attempts to infer the polarity/orientation of a statement's sentiment toward that aspect. Lexicons and syntactic characteristics-based Machine learning models are used in most traditional ASC techniques [15], [18], and [22]. The labour-intensive hand-crafted feature qu

7. Parsing and Sentement analysis using Recursive Neural Networks

The purpose of ASC is to see if user opinions stated in reviews/tweets are positive, negative, or neutral on particular aspects (aspect categories or aspect keywords). Aspect-based SA [5], [11] is a core task in SA research that is broken down into several subtasks: aspect extraction [12][13][14], opinion identification [15][16] and ASC [15][16][17][18][19]. Previous research [20][21] attempted to tackle these sub-tasks at the

same time, with each sub-task receiving the majority of the research time. This research focuses on deep learning approaches to solving the ASC problem. ...

... The purpose of ASC is to see if user opinions stated in reviews/tweets are positive, negative, or neutral on particular aspects (aspect categories or aspect keywords). Aspect-based SA [5], [11] is a core task in SA research that is broken down into several subtasks: aspect extraction [12][13][14], opinion identification [15][16] and ASC [15][16][17][18][19]. Previous research [20][21] attempted to tackle these sub-tasks at the same time, with each sub-task receiving the majority of the research time. ...

... Given a phrase and an aspect, ASC attempts to infer the polarity/orientation of a statement's sentiment toward that aspect. Lexicons and syntactic characteristics-based Machine learning models are used in most traditional ASC techniques [15], [18], and [22]. The labour-intensive hand-crafted feature quality is critical to the models' performance. ...

Sentence Classification Using Convolutional Neural Networks

In deep learning, a **convolutional neural network (CNN, or ConvNet)** is a class of artificial neural network (ANN), most commonly applied to analyze visual imagery.^[1] CNNs are also known as **Shift Invariant** or **Space Invariant Artificial Neural Networks (SIANN)**, based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation-equivariant responses known as feature maps.^{[2][3]} Counter-intuitively, most convolutional neural networks are not invariant to translation, due to the downsampling operation they apply to the input.^[4] They have applications in image and video recognition, recommender systems,^[5] image classification, image segmentation, medical image analysis, natural language processing,^[6] brain-computer interfaces,^[7] and financial time series.^[8]

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a

different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increase

ng complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes^{[9][10][11][12]} in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

Dialogue Generation with LSTMs

One of the principal problems of human-computer interaction is miscommunication. Occurring mainly on behalf of the dialogue system, miscommunication can lead to dialogue breakdown, i.e., a point when the dialogue cannot be continued. Detecting breakdown can facilitate its prevention or recovery after breakdown occurred. In the paper, we propose a multinomial sequence classifier for dialogue breakdown detection. We explore several LSTM models each different in terms of model type and word embedding models they use. We select our best performing model and compare it with the performance of the best model and with the majority baseline from the previous challenge. We conclude that our detector outperforms the baselines during the offline testing.

Short Questions

- 1.What are the applications of deep learning?
- 2.Explain Briefly about object detection?
- 3.Explain Name entity recognition?
- 4.Explain Automatic Image Captioning?

5.Explain dialogue Generation with LSTMs?

Long Questions

1.Explain about Image Generation with Generative adversarial networks?

2.Explain briefly about Video to Text with LSTMs Models?

3.Explain parsing and Sentiment Analysis using Recursive Neural Networks?

4.What is opinion mining using recurrent in detail?

5.Explain sentence Classification using Convolutional Neural Networks?

SVU COLLEGE OF CM AND CS

DEPARTMENT OF COMPUTER SCIENCE

Time:10 A.M Internal Examination-1 max.Marks:30

Deep Learning I-Internal Question paper

Part-A

[Answer any 4 Questions]

5*2=10

1.What is Neural Network?

2.What are the applications of Neurons?

3.Explain Feed Forward Briefly?

4.What is deep learning?

5.Explain Gradient Descent?

6.Explain Briefly about Mitigation?

7.What is Regularization?

8.Explain about Drop out in deep learning?

Part-B

Answer any One Question from each unit

10*2=20

Unit-1

9.Explain Perceptron algorithm with example?

(or)

10.Difference between Feed Forward and Back Propagation Networks?

Unit-II

11.Explain about Relu Heuristics for avoiding bad local minimum?

(or)

12.What is nestors Accelerated Gradient Descent with Example?

SVU COLLEGE OF CM AND CS

DEPARTMENT OF COMPUTER SCIENCE

Time:10 A.M

Internal Examination-1

max.Marks:30

Deep Learning I-Internal Question paper

Part-A

[Answer any 4 Questions]

5*2=10

1.what is Convolutional neural networks?

2.Explain Convolution?

3.Explain Transfer Learning?

4.what is LSTM?

5.Difference between encoder and decoder?

6.Explain Sparse?

7.What is Denoising?

8.Explain Contractive in Deep Learning?

Part-B

Answer any One Question from each unit

10*2=20

Unit-1

9.Explain about CNN Architectures?

(or)

10.Explain about pooling Layers?

Unit-2

11.Explain Briefly about Variational Autoencoders?

(or)

12.Explain sentence Classification using Convolutional Neural Networks?

MASTER OF COMPUTER APPLICATIONS DEGREE EXAMINATION

SECOND SEMESTER

MCA 402 - DEEP LEARNING

(UNDER C.B.C.S REVISED REGULATIONS W.e.f. 2020-2021)

(common paper to university and all affiliated colleges)

Time: 3 hours

Max.Marks:70

Part-A

(Compulsory)

Answer any FIVE of the following questions.

Each question carries 4 marks

5*4=20

1.

- a). What is Neural Network?
- b). What are the applications of Neurons?
- c). Explain Feed Forward Briefly?
- d) What is deep learning?
- e) Explain Gradient Descent?
- f) Explain Briefly about Mitigation?
- g). What is Regularization?
- h). Explain about Drop out in deep learning?

Part-B

**Answer FIVE Questions , choosing ONE question from each unit
each question carries 10 marks**

5*10=50

Unit-1

2.

- i). Explain Perceptron algorithm with example?
(or)
- ii). Difference between Feed Forward and Back Propagation Networks?

Unit-II

3.

- i) Explain about ReLu Heuristics for avoiding bad local minimum?
(or)
- ii) What is Nestors Accelerated Gradient Descent

Part-B

Answer any One Question from each unit

10*2=20

Unit-1

1.

i) Explain Perceptron algorithm with example?

(or)

ii) Difference between Feed Forward and Back Propagation Networks?

Unit-II

2.

i) Explain about Relu Heuristics for avoiding bad local minimum?

(or)

ii) What is Nesterovs Accelerated Gradient Descent with Example

Unit-III

3.

i) Explain about CNN Architectures?

(or)

ii) Explain about pooling Layers?

Unit-IV

4.

i) Explain Briefly about Variational Autoencoders?

(or)

ii) Explain sentence Classification using Convolutional Neural Networks?

Unit-V

5.

i) Explain about Image Generation with Generative adversarial networks?

(or)

ii). Explain briefly about Video to Text with LSTMs Models?