

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

MASTER'S THESIS

Efficient Approaches for Data Augmentation by Using Generative Adversarial Network

Author:

Pretom Kumar SAHA
Matriculation Nr. 1276545

Supervisor:

Prof. Dr. Doina LOGOFATU
Prof. Dr. Eicke GODEHARDT

*A thesis submitted in fulfillment of the requirements
for the degree of Master of Engineering (M. Eng.)
in the*

November 10, 2021

Declaration of Authorship

I, Pretom Kumar SAHA, declare that this thesis titled, Efficient Approaches for Data Augmentation by Using Generative Adversarial Network and the work presented in it are my own.

- I certify that neither I nor anyone else has ever submitted this work or sections of it for credit elsewhere.
- All citations or allusions from other people's works have been carefully recognized and acknowledged.
- The bibliography includes all secondary literature and other sources that have been noted and listed. Any charts, graphs, drawings, and Internet sources are subject to the same rules.
- Furthermore, I agree to have my work stored electronically and delivered anonymously to be checked for plagiarism. I am aware that if the declaration is not made, the paper will not be reviewed and marked as failed.

Signed: 

Date: November 10, 2021

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Abstract

Faculty 2: Computer Science and Engineering
Information Technology (M.Eng.)

Master of Engineering (M. Eng.)

Efficient Approaches for Data Augmentation by Using Generative Adversarial Network

by Pretom Kumar SAHA

In the present and future, data is the most valuable thing in the world. Therefore, it is now a challenge for everyone in every sector to work with data. Collecting data to predict or collecting data to analyze is a very valuable task. And every new research, new machine learning method, and algorithms testing depends on a massive amount of different data. And it's also a security issue for many fields to share actual data. It's always hard to find the perfect data set. It's not just about figuring out huge amounts of data. Many other data analysis processes need to be performed on that dataset to make it worthwhile. To overcome this problem, data augmentation is one of the suitable solutions. There has a lot of suitable ways which is used in the field of data enhancement. Statistical, mathematical, and adding noise are very common in the sector of data augmentation. On the other hand, deep learning has the quality to gain state-of-the-art performance in the computer vision sector. So, it's a suitable machine learning method that can be used to produce synthetic real-world data.

If we can introduce one machine learning method in the field of data augmentation, then we can reduce the cost and energy to collect real-world data from a different source. To collect the data and then inspect, cleanse, transform, and model data to find out the valuable data from the data set. We can skip all the processes to successfully train a deep learning model to generate synthesized data. The idea behind data augmentation is to create a new dataset that depends on some existing dataset features. Generative Adversarial Networks (GANs) are a class of machine learning frameworks introduced by Ian Goodfellow in 2014. They are a game-like way to learn and generate new datasets.

In most cases, GANs are used in image data enhancement. The result is very high. This work aims to implant GANs to generate tabular datasets based on some defined input data with features. The process of GANs is very interesting as a game. GANs have two parts, one is the generator, and the second is the discriminator. They play against each other to win the game. This is the process of generating a new data set. We will use our data set on the GAN model using some specific hyperparameter value and optimizer, which we will find out through our experiment. Finally, we will produce a CSV file with model-generated synthesis data and visualize the performance statistic of our model in the graph. This article will explain the different facts related to deep learning, ANN, CNN, Data Augmentation, and GANs.

Keywords: Data Augmentation , Generative Adversarial Networks , Deep Learning , Tabular Data

Acknowledgements

In the beginning, It's a genuine pleasure to express my deep sense of thanks and gratitude to my supervisor **Prof. Dr. Doina LOGOFATU** allow me to do my thesis under her excellent supervision. It's quite impossible for me to complete my thesis work and writing without her support. She gives me her valuable advice and suggestion whenever I encounter problems or difficulties. I learn a lot about writing a paper and explaining my experiments in a way that will be eye-catching and valuable to others.

I would also like to acknowledge and give my warmest thanks to **Prof. Dr. Eicke GODEHARDT**, for accepting my proposal and for being the second supervisor. It is my privilege to thank my best friend Chandrima Biswas, who has been very supportive mentally throughout the completion of my master's degree. I am also grateful for the support I received from my parents and sister. And finally, I am thankful to the Almighty for giving me the opportunity that allowed me to conduct this thesis.

Contents

Declaration of Authorship	iii
Abstract	v
Acknowledgements	vii
1 Introduction	1
1.1 Motivation	2
1.2 Problem Statement	2
1.3 Research Question:	3
1.4 Scope of the work	3
1.5 contribution	4
1.6 Outline	4
2 Literature Overview	5
2.1 Neural Networks	6
2.2 Artificial Neurons	8
2.3 Feedforward Neural Networks	9
2.4 Optimizing Neural Networks	11
2.4.1 Gradient Descent	12
2.4.2 Adaptive Learning Rate Optimizers	12
Adagrad	13
RMSProp	13
Adam	14
2.5 Batch Normalization	14
2.6 Activation Functions	15
2.6.1 Linear Activation	15
2.6.2 Sigmoid or Logistic Activation Function	16
2.6.3 Tanh (Hyperbolic Tan)	16
2.6.4 ReLU	17
2.6.5 Leaky ReLU	17
2.7 Convolutional Neural Networks	18
2.7.1 convolutional layer	20
2.7.2 Pooling layer	21
2.7.3 Dropout layer	21
3 Data Augmentation	23
3.1 Existing Methods of Data generation	24
3.2 Data Augmentation Techniques	24
3.2.1 Image Data Augmentation	25
3.2.2 Text Data Augmentation	26
3.2.3 Speech Data Augmentation	28
3.2.4 Tabular Data Augmentation	29

4	Generative Adversarial Network	31
4.1	Preliminaries	31
4.1.1	Terminology	31
4.1.2	Notation	32
4.1.3	Statistic distributions	32
4.2	Generative Adversarial Nets	33
4.2.1	Fully Connected	33
4.2.2	Convolutional GANs	34
4.2.3	Conditional Adversarial Nets	35
4.2.4	GANs with inference models	35
4.2.5	Adversarial Auto Encoder (AAE)	35
4.3	Proposed GAN Model	37
4.3.1	Generator	38
4.3.2	Discriminator	39
4.3.3	Loss Function	40
	Original Loss	41
	Information Loss	41
	Classification Loss	42
4.4	Training GANs	42
4.4.1	Training tricks	44
5	Experiment and Result Evaluation	47
5.1	Experimental Environment	47
5.2	Python Librarys	47
5.3	Dataset	48
5.4	Data Analysis	49
5.5	TGAN Model Setup	50
5.6	Performance Analysis	54
6	Conclusion and Future work	59
6.1	Conclusion	59
6.2	Future work	60

List of Figures

2.1	Schamane of human neuron network [61]	7
2.2	Representation of Neural Network.	7
2.3	A single Neuron of ANN[7]	8
2.4	A single-layer Feedforward Neural Network	10
2.5	A multilayer Feed-forward network with one hidden layer	11
2.6	Linear Activation Functions	15
2.7	The sigmoid activation function	16
2.8	tanh activation function	16
2.9	ReLU activation function	17
2.10	Leaky ReLU activation function	18
2.11	A typical ML system block diagram	19
2.12	A CNN with one convolutional,pooling layer and Dropout	20
3.1	The Classification of Data Augmentation	23
4.1	The Architecture of GANs	31
4.2	Architecture of Adversarial Auto Encoder (AAE)	37
4.3	Architecture of the generator with 5 deconvolutional layers	38
4.4	Architecture of the discriminator with 5 convolutional layers	39
5.1	The flowchart of proposed method	50
5.2	The flowchart of the building generator model	51
5.3	The flowchart of the building discriminator model	52
5.4	The flowchart of the building GAN model	53
5.5	From Table 1 Numerical data train with GAN	54
5.6	From Table 2 Numerical data train with GAN	55
5.7	Dummy value of gender feature from table	56
5.8	From Table 1 One feature Categorical data train with GAN	56
5.9	From Table 2 One feature Categorical data train with GAN	56
5.10	From Table 1 two Categorical feature and numeric data train with GAN	57
5.11	From Table 2 two Categorical feature and numeric data train with GAN	57
5.12	GAN generated synthesized including categorical and numeric data	58

List of Tables

5.1	Configaretion details of Google Colab	47
5.2	A dataset about the European automobile industry	49
5.3	A dataset of student test scores	49
5.4	Actual numerical data used as Input	54
5.5	GAN generated synthesized numerical data	55
5.6	Actual data and GAN generated synthesized numerical data	55

Chapter 1

Introduction

In the world of data to aggregate a large amount of different fields data for implementing and testing applications. To evaluate the application and perform in specific scenarios using accurate data is essential for applications. In every area of data sciences, it's tough to collect a significant amount of pertinent data. Gathering valuable data will be highly cost-consuming in terms of money, time, staffing, and applications [25]. Feather more, many other related methodologies are connected for data analysis, data cleaning, and so on . It's essential to arrange proper study, guidelines, and training to fulfill all these above factors. To overcome all this complexity and cost by creating artificial new data containing all the features and property like actual data [25].

A set of detached, objective facts, which explain a set of features, is defined as data [2]. In an accessible word, data is a set of quantitative or qualitative values that represent the characteristics of components [31]. Data can be defined in three parts depending on the way of data organizing Structured, Semi-structured, and Unstructured . For different use cases, data may summarize in a tabular style in a variety of ways. A table that displays all of the rows of a data collection is the most basic type [58]. A tabular database is organized in a tabular format, as the name suggests [59].

The data science research literature frequently assumes that the input data for various data analytics procedures is already clean and accurate for data consumption [31]. However, cleansing data of issues in its values or structure consumes a significant portion of data professionals time and effort. According to recent polls, data cleansing takes up to 60% of data scientists work [31].

The danger of poor metadata quality impacts a dataset's discovery and consumption both inside and across portals. On the one hand, missing metadata directly impacts the ability of search and discovery services to find relevant and related datasets for specific customer requirements [20]. On the other hand, incorrect dataset descriptions present several challenges in terms of processing and integration with other datasets. These dangers are present in any search and data integration scenario[20]. There are no comprehensive, quantitative, or objective reports on the actual quality of Open Data portals that we are aware of.

Naturally, most of the information in the business is private and secret. While some databases are freely accessible, new datasets are difficult to come by. Even if data owners are ready to provide anonymized datasets, the time and effort required to compile the data and through bureaucracy may deter them [24]. We believe that if data owners can quickly generate "fake" or synthesized data comparable to actual data, friction in data disclosure will be reduced.

1.1 Motivation

Access to data is becoming increasingly crucial as interest in using machine learning (ML) and predictive modeling approaches across all disciplines of actuarial science has grown in recent years [47]. Researchers can address more practical challenges and verify newly established techniques when they have access to actual data. If the data is also made public, academics and corporations can open source their processes, allowing others to build on previous work. Furthermore, having a centralized repository of datasets for each research issue helps the community to describe the current state of the art, assess new procedures, and track progress [47].

Other papers on the scientific approach of Big data development and deployment employ three key levels from a scientific standpoint [30]. The writers of most publications emphasized storage, processing, and analysis of data as layers of Big data, and they all agreed on the relevance of these levels and their roles in corporate success [30]. Almost every article regarding Big data implementation focuses on Storage, Process, and Analysis in some way. In terms of storage, it can argue that the corporate sector now has a greater need to store and handle data than in the past due to the significant rise in structured and unstructured data (Chen, Mao, and Liu, 2014). Furthermore, large data storage systems require adequate and trustworthy interfaces to provide real-time data for real-time applications (Kambatla, Kollias, Kumar, and Grama, 2014). After data is gathered and stored from numerous sources that create diverse data types (Bolón-Canedo et al., 2015), it is categorized, integrated, and noise and redundancy are removed to reduce redundant storage space and accommodate important information data (Chen et al., 2014). We need business software and information systems to analyze (Wang and Hajli, 2017). After then, they should explore to make the best use of the information [30]. The data then should be interpreted to make the best use of it. Data analysts and scientists who possess the abilities of a database designer, software programmer, and statistician should be prepared for this purpose by information technology instructors [30]. By Introduce an automated synthesized data generation can resolve all this manual process of research data.

1.2 Problem Statement

For applications in many areas, the availability of a massive quantity of data is a critical problem. Appropriate data is required to comprehend particular situations (for example, the relevant information is retrieved for forecasting the development of systems or environments) and create successful applications [24]. Preliminary case studies are available to create big enough data banks to educate doctors, experts, or artificial models. In certain areas, artificially creating new data may be a solution to overcome restricted availability[24]. This may be accomplished for various jobs by altering previously accessible data.

Some basic problems we faced in this thesis are summarised as follow:

1. Generate synthesized tabular form given Dataset. The selection of a suitable GAN model is one of the biggest challenges in the thesis.
2. Train a GAN model with generator and discriminator neuron. Find out suitable hyper-parameter values for the GAN model.

3. Select a more appreciated optimizer for the GAN model.
4. Visualized and analyzed the performance of the model.

1.3 Research Question:

There have lots of machine learning algorithms like data classification, Anomaly detection, missing value imputation, and so on. It's difficult for the researcher to find relevant data set that they can use in their research. Besides that, many models need to train with vast amounts of data to become more precious. To solve all these problems, the deep learning framework GAN is a perfect option. Nowadays, GAN performs with great success in the image data field. So, we are planning to use the same model with some changes to produce the tabular data. Because tabular data is more usable in the research background. As we explain about data, tabular data, and the importance of data in the above.

In this thesis, some points or questions which are elaborately explained or answered are listed here:

- RQ_1: What is the basic structure of the GAN model with Generator and Discriminator neurons?
- RQ_2: What is the importance of the Activation function, Batch normalization, optimizer, and CNN in GAN?
- RQ_3: What is Data Augmentation, and what is the present situation of Data Augmentation in different data fields?
- RQ_4: How does the model perform with different data sets and features, including the success of data generation compared with original data?
- RQ_5: What are the limitations and future work can be for the GAN model with respect to tabular data?

1.4 Scope of the work

We recognize the significance of real-world data and its issues as a result of the discourse described above. As a result, the approach is to develop a deep learning model and train it to improve the data quality. This is the most crucial topic in this thesis, and we will apply GAN to solve the problem. GAN is already a well-established model in the field of picture data, and it has achieved considerable success. The many forms of GANs are distinguished by their mode of operation and the significance of the characters they provide. As a result, choosing the most appropriate GAN model from among the available options isn't easy. There is currently a significant amount of active research on using the GAN model for the generation of tabular data. The majority of the study was confronted with the challenge of utilizing categorical data in addition to numerical data sets. Finding adequate optimizers and hyperparameters is one of the most difficult intractable challenges that any researcher is currently confronting. Currently, there is no solution. We want to make an effort to address all of the issues that have been raised. It also focuses on the theoretical foundations of GAN and its associated elements, which might help elucidate this understanding further.

1.5 contribution

GAN-based data augmentation and traditional data augmentation will be studied in depth in this thesis to assess the performance of GANs in data augmentation. With the help of a unique GAN architecture, we can produce synthesized tabular data. It's possible to train a deep neural network for data segmentation and assess the efficacy of GAN-based data augmentation in a quantifiable way by using these synthetic data and their related synthetic annotations. On two separate datasets, one from 2011-2021 cars information in Germany, and one from students' performance in exams. To further analyze the influence of GANs samples on the training process, we compare segmentation performance while training with various ratios of actual and generated data.

In short form, the following contributions can be counted from this thesis work:

- We presented all the related issues that one can face when implementing the GAN model for data augmentation.
- We proposed one of the suitable optimizers that can be used to improve the model's learning rates.
- The preference can vary depending on the data set's features, which guides the other researcher on further development.
- The augmented data can be used for data classification, anomaly detection, and other data mining research to improve performance accuracy.
- We show that our proposed method overcomes the classical data enhancement problems, achieving state-of-the-art performance.

1.6 Outline

The influence of data augmentation on tabular data is investigated in this study using a deep convolutional neural network. Additionally, a generative adversarial network-based and novel augmentation method are suggested. On a publically available dataset, the performance of the proposed technique is evaluated. The rest of the paper is progress in the following manner. The literature overviews are related to deep learning and GAN such as NN, ANN, FNN, ONN, BN, AF, and CNN in Section 2. In part 3, the related developments on Data Augmentation that are already used or in ongoing research are explained. Section 4 is presented the overview of GAN with mathematical explanation and algorithms related to the proposed methods. The experiments and evolution results are discussed in section 5. Finally, A summarization of the thesis with gain and loss is presented, including the idea that we can work in the future.

Chapter 2

Literature Overview

Selecting an adequate feature space where input instances have desirable characteristics for solving a specific problem is a significant challenge in statistical machine learning. In supervised learning for binary classification, for example, it is often necessary that the two classes be separable by a hyperplane [10]. When this characteristic is not directly fulfilled in the input space, the user is given the option of mapping instances into an intermediate feature space with linearly separable classes. This middle space may be created directly with hand-coded characteristics, implicitly using a so-called kernel function, or learned automatically [10]. The user is responsible for designing the feature space in both of the initial instances. This may come at a high cost in computing effort or expert knowledge, mainly when working with high-dimensional input fields like pictures [10].

Deep structured learning, familiar as deep learning or hierarchical learning, has been a growing field of study in machine learning since 2006 [14]. Deep learning techniques have already influenced a wide range of signal and information processing work, traditional and new, widened scopes, including crucial aspects of machine learning and artificial intelligence. Several seminars, tutorials, special issues, and unique conference sessions have focused on deep learning and its applicability to different signal and information processing fields [14].

However, human information processing processes indicate that deep architectures must extract complex structures and construct internal representation from various sensory inputs. Human speech production and perception systems, for example, both use layered hierarchical structures to convert data from the waveform to the language level [14]. In a similar vein, the human visual system is hierarchical in structure, primarily in terms of perception and, intriguingly, in terms of "creation". If efficient and effective deep learning algorithms can be created, it is reasonable to expect that the state-of-the-art in processing these kinds of natural signals can be advanced. Artificial neural network research is where the idea of deep learning came from in the past [14]. Models with a deep architecture include feed-forward neural networks, MLPs with multiple hidden layers, and deep neural networks (DNNs) [14].

Representation learning is a collection of techniques that enables a computer to be given raw data and automatically learn the representations required for detection or classification. Deep-learning techniques are representation-learning methods with many layers of expression, achieved by assembling simple but non-linear modules that convert the model at one level into a higher, somewhat more abstract level [21]. Very complicated functions may be learned by composing a large number of such

changes. Higher layers of representation enhance elements of the input that are essential for discrimination and reduce irrelevant variations in classification tasks. For example, a picture comprises an array of pixel values. The first layer of representation usually indicates the presence or absence of edges at particular orientations and positions in the image [21]. The second layer identifies motifs by detecting specific edge configurations, despite minor differences in edge locations. The third layer may combine motifs into bigger combinations that relate to pieces of known things, with succeeding levels detecting items as a mixture of these parts. The essential element of deep learning is that these layers of features are learned from data using a general-purpose learning process rather than being developed by human engineers[21].

Deep learning is making significant progress in addressing issues that have eluded the artificial intelligence industry for many years. It has shown to be highly effective at detecting complex structures in high-dimensional data, making it suitable for a wide range of scientific, commercial, and government applications [21]. It has beaten other machine-learning techniques at predicting the activity of potential drug molecules, analyzing particle accelerator data, reconstructing brain circuits, and predicting the effects of mutations in non-coding DNA on gene expression and disease, in addition to breaking records in image recognition and speech recognition [21]. Perhaps more unexpectedly, deep learning has shown great promise for various natural language comprehension tasks, including subject categorization, sentiment analysis, question answering, and language translation [21]. It can predict that deep learning will have more success soon since it needs minimal manual engineering and can readily improve computing and data access [21]. New deep neural network learning methods and architectures are being developed, which will only accelerate this development.

2.1 Neural Networks

Following the principle "divide and conquer" is one effective method of tackling complex issues [7]. It may be broken down into smaller components to comprehend a complicated system. Simple components may also be combined to create an elaborate scheme (Bar Yam, 1997) [7]. One way to do this is via networks in figure 2.2. The human brain demonstrates the presence of vast neural networks capable of performing cognitive, perceptual, and control tasks similar to those performed by humans [8]. The brain can do computationally intensive perceptual actions and control tasks. The brain's advantage is its efficient utilization of massive parallelism, highly parallel computing architecture, and insufficient information-processing capacity [8]. More than 10 billion linked neurons make up the human brain [8]. Each neuron is a network, which works as a cell that receives, processes, and transmits information via biochemical reactions as shown in figure 2.1. There are many different networks like the human neuron network, but they always have the same components: a collection of nodes and node connections [7].

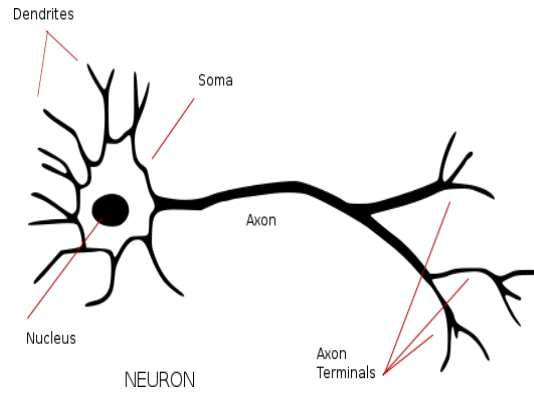


FIGURE 2.1: Schamane of human neuron network [61]

Nodes may be thought of as computing units. They take in information and process it to get a result. This processing may be primary or sophisticated [7]. The connections determine information flow between nodes. They may be unidirectional or bidirectional. In the human body, Dendrites are tree-like nerve fibers that link the cell nucleus to the cell body or soma [8]. A single long thread called an axon extends from the cell body and is linked to neighboring neurons through synaptic terminals or synapses [8]. Signal transmission at synapses is a complicated chemical process in which particular transmitter chemicals are released from the sending end of the junction [8]. The electrical potential within the receiving cell's body is raised or decreased due to this action. A pulse is transmitted down the axon, and the cell is 'fired' if the voltage exceeds a threshold [8].

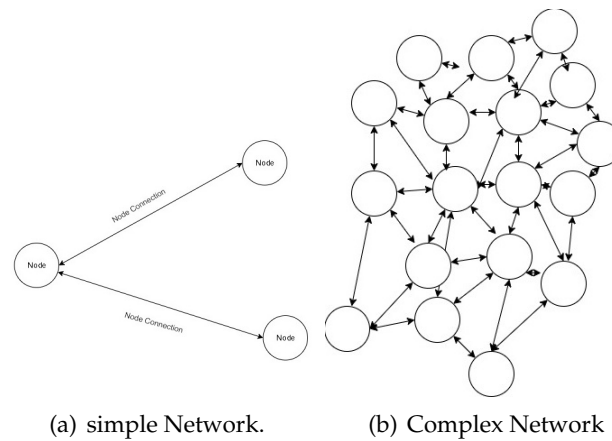


FIGURE 2.2: Representation of Neural Network.

According to this idea of a human brain network, the interactions of nodes via the connections lead to a network's global behavior, which cannot be seen in the network's components [7]. Emergent behavior is the term for this kind of global behavior. This implies that the network's capabilities outweigh its constituents, making networks a very effective tool. In physics, computer science, biology, ethology, mathematics, sociology, economics, and telecommunications, among other fields, networks play a broad range of influences [7].

2.2 Artificial Neurons

In theory, a neural network has the power of a universal approximator, which means it can perform any arbitrary mapping from one vector space to another [4]. The nodes of one kind of network are referred to as 'artificial neurons.' Artificial neural networks are what is referred to as (ANNs) [7]. An artificial neuron is a computer model that is based on natural neurons. The primary benefit of neural networks is that they may utilize previously undiscovered information buried in data, but they cannot extract it [4].

When modeling artificial neurons, the intricacy of actual neurons is extensively abstracted. These comprise inputs (such as synapses) multiplied by weights (signal intensity) and then calculated by a mathematical function that determines the neuron's activity [7]. 'Learning of neural network' or 'training of neural network' is the process of 'capturing' unknown knowledge. Learning involves adjusting the weight coefficients in such a manner that specific criteria are met in mathematical formalism [4]. Another function computes the artificial neuron's output, sometimes depending on a certain threshold. Artificial neuron networks (ANNs) integrate artificial neurons to analyze data [7].

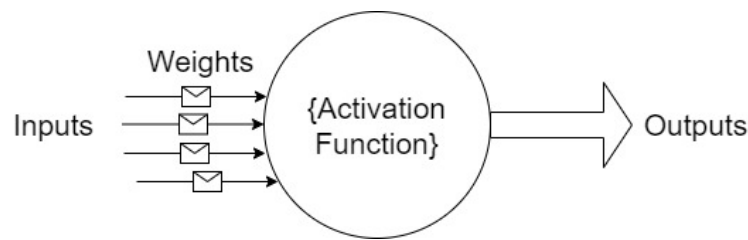


FIGURE 2.3: A single Neuron of ANN[7]

The greater the input amplified by an artificial neuron, the larger its weight. We may argue that the negative weight blocks the signal since it can be negative[7]. The calculation of the neuron will vary depending on the weights. We can get the desired output for particular inputs by changing the weights of an artificial neuron [7]. However, finding all the required weights by hand in an ANN with hundreds or thousands of neurons would be difficult. However, some methods can change the ANN's weights to get the required output from the network [7]. Learning or training is the term used to describe the process of changing the weights. The two major kinds of training processes are supervised and unsupervised. Supervised training for example, using a multi-layer feed-forward (MLF) neural network, implies that the neural network is aware of the intended output and that the weight coefficients are adjusted so that the calculated and desired outputs are calculated near as feasible [4]. Unsupervised training implies that the intended output is unknown, and the system is given a collection of facts to settle down to a stable state after several repetitions [4].

The number of different kinds of ANNs and their applications is enormous. Hundreds of distinct neural models have been created since McCulloch and Pitts (1943) established the first [7]. Functions, acceptable values, topology, learning methods, and so forth may be the distinctions between them. There are also numerous hybrid

models in which each neuron has more characteristics than the ones we've seen so far [7]. Many artificial neural models are available, ranging in complexity from basic integrate-and-fire neurons to compartmental models and beyond [32]. Between biological precision and computer efficiency, there is a trade-off. Leaky-integrate-and-fire (LIF) models are extensively employed in biological brain modeling, and they may be found on neuromorphic devices [32]. Unlike conventional Von Neumann designs, Neuromorphic hardware is not general-purpose and instead emulates neurons, allowing for a high level of parallelism[32].

Because ANNs are used to handle information, they are mostly utilized in areas that deal with it. There are several ANNs used to mimic actual neural networks to investigate behavior and control in animals and machines, as well as ANNs used for technical applications, including pattern recognition, forecasting, and data compression [7]. LIF models gather activity from other neurons, to put it simply. When a neuron receives enough activation to exceed its threshold, it fires and transmits the stimulation to the neurons to which it is linked [32]. These can be simulated in real-time, although most systems operate in discrete time increments. Some of a neuron's activity leaks away if it doesn't fire in a time step [32]. In simulations, utilize a 1 ms time step. Weighted unidirectional synapses link neurons. These weights may be positive or negative[32].

2.3 Feedforward Neural Networks

Artificial neural networks are based on biological neural networks, as the name suggests. The biological brain (or, in the most extreme case, the human brain) is much more sophisticated and superior than traditional computers. The capacity to "learn" and "adapt" is the most significant distinguishing characteristic of a biological brain, lacking in a typical computer [9]. Traditional computers carry out particular tasks depending on the instructions, often known as "programs" or "software," placed into them [9]. Layered neural networks have layers of neurons. An input layer and an output layer are required. Hidden layers are between the input and output layers (if any), with hidden neurons or units corresponding to their processing nodes. The input layer of the network's source nodes provides specific components of the activation pattern (input vector), which are used to generate the input signals for the second layer's neurons (computation nodes) (i.e., the first hidden layer) [6]. The second layer's output signals are fed into the third layer and throughout the network. The following layer of neurons (computation nodes) projects onto a layer of nodes, but not the other way around.

"Feedforward neural networks"(FNN) and "recurrent neural networks" (RNN) are the two major types of network designs that rely on the kind of connections between the neurons [9]. The network is referred to as a "feedforward neural network" if there is no "feedback" from the neurons' outputs to the inputs. Otherwise, the network is known as a "recurrent neural network", if such feedback, i.e., a synaptic link from the outputs to the inputs[9].

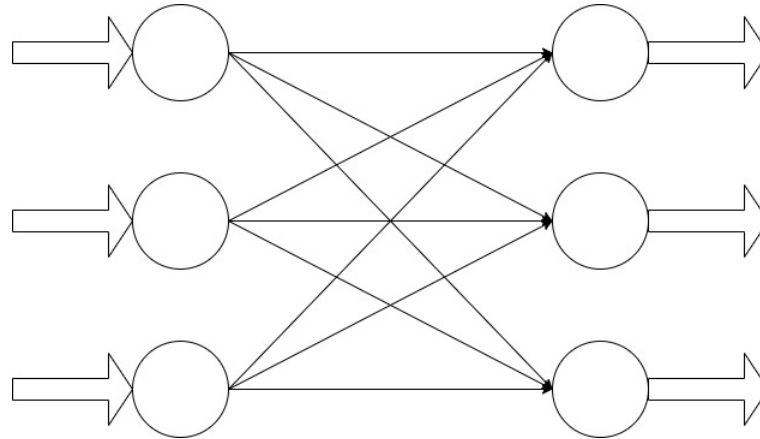


FIGURE 2.4: A single-layer Feedforward Neural Network

In a feedforward or acyclic network, only the previous layer's output signals are used as inputs to the neurons in each network layer [6]. Figure 2.4 illustrates a fully linked single-layer feedforward neural network. There are two levels in this structure, including the input layer. However, since no calculation is done in the input layer, it is not counted. The output layer receives input signals through the weights, and the neurons in the output layer compute the output signals [9].

The total response of the network to the activation pattern provided by the source nodes in the input (first) layer is composed of the collection of output signals of the neurons in the output (final) layer of the network [6]. The multilayer perceptron is the name given to neural networks with this type of design. A multilayer feedforward neural network is shown in Figure 2.5 with one "hidden layer." There is at least one layer of "hidden neurons" between the input and output layers instead of a single-layer network [9]. The purpose of hidden neurons, according to Haykin [5], is to act as a helpful intermediary between external input and network output. The network can retrieve higher-order statistics thanks to the presence of one or more hidden layers. Because there are three input neurons, four hidden neurons, and two output neurons in the example shown in Figure 2.5, there is only one hidden layer, and the network is referred to as a 3-4-2 network [9]. Because every neuron in each layer is linked to every other neuron in the next forward layer, networks in both Figure 2.4 and Figure 2.5 are "completely connected."

Hidden neurons' job is to act as a helpful intermediary between external input and network output [6]. The network is possible to retrieve high-order statistics by adding one or more hidden layers. Because of the additional synaptic connections and the added dimension of brain interactions, the network gains a global viewpoint despite its local connectedness in a loose sense [6]. In each network layer, every node is linked to every other node in the next forward layer, indicating that the neural network is completely connected. The network is referred to be partly connected if it is not fully linked [6].

The capacity of a neural network to "learn" is the most significant characteristic that sets it apart from a traditional computer [9]. A neural network may pick up information from its surroundings and use it to enhance its performance. "With the Learning process, the free parameters of a neural network are modified via a process of stimulation," Haykin described learning in the context of neural networks in the literature

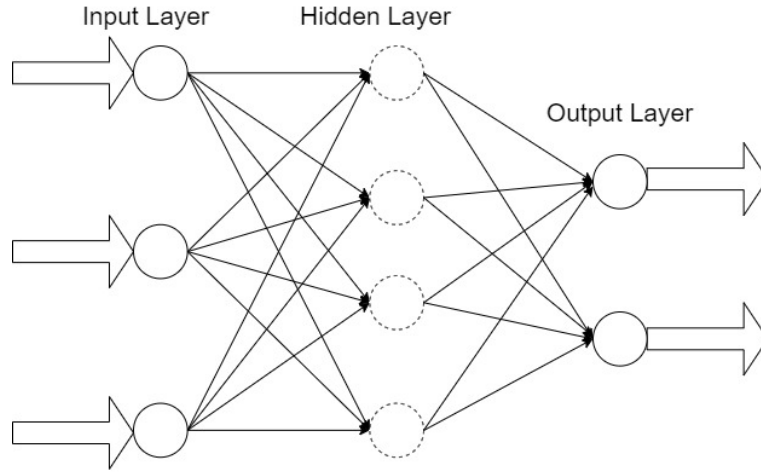


FIGURE 2.5: A multilayer Feed-forward network with one hidden layer

[5]. The way in which parameter changes occur [5] determines the kind of learning .

2.4 Optimizing Neural Networks

It's a big challenge for an FNN to find the best parameter (weight) suitable for the dataset. Depending on the expected output and input data, the values of the parameters will be changed. An FNN with input X_0 to output $X_i = f(x, \theta)$, where i is donated as a number of layer[19]. defines as a vector contain all the parameters. If W_i is consider as the weighted vector, then the final output,

$$\alpha = \delta(\sum_i (x_i * w_i) + b) \quad (2.1)$$

Here, in equation 2.1 δ is the activation function, and b is the bias.

Loss functions are how an algorithm fits data in the first place, and they're more than simply a static depiction of how they're doing. Most machine learning algorithms employ some kind of loss function in optimizing or determining the optimum parameters (weights) for the dataset.

Consider linear regression for a basic example. MSE ("Mean Squared Error") is used to calculate the line of most incredible fit in conventional "least squares" regression. The MSE is computed over all input instances for every set of weights that the model attempts. The model then uses an optimizer technique like Gradient Descent to optimize the MSE functions—in other words, to reduce them to the smallest possible value.

If the target is y and the prediction of the FNN is z then the loss faction can be define as $L(y, z)$. As we consider $f(x, \theta)$ as the output then mean of loss function:

$$L_m = \frac{1}{N} \sum_{i=1}^N (L(f(x_i, \theta), z_i)) \quad (2.2)$$

Where,

$$L = \| f(x_i, \theta) - z_i \|^2 \quad (2.3)$$

This is the function of popular MSE ("Mean Squared Error") [33]. Because a lower loss function value usually equals a better FNN function approximation, the training process for FNNs may be framed as an optimization problem to minimize the loss function L_m concerning the network parameters[33]. A version of the gradient descent method is frequently used for this.

2.4.1 Gradient Descent

One of the powerful optimization algorithms is Gradient descent, and it is by far the most frequent method of optimizing neural networks [11]. Gradient descent (GD) has been suggested many times (e.g., [1]) to reduce the empirical risk $E_r(f(x))$. Gradient descent is a method of minimizing an objective function L_m equation 2.2 parameterized by the parameters $\theta \in \mathbb{R}^d$ of a model by updating the parameters in the opposite direction as the gradient of the objective function L_m [23]. Each iteration updates the weights w using the gradient of $E_r(f(x))$,

$$w_{t+1} = w_t - \lambda \frac{1}{N} \sum_{i=1}^N \Delta_w Q(z_i, w_t) \quad (2.4)$$

where, λ is a well-selected learning rate. This method achieves linear convergence [3], that is, $\log t$, where t denotes the residual error, under appropriate regularity assumptions, when the starting estimate w_0 is close enough to the optimum, and when the learning rate is modest enough [11]. The learning rate find out the size of the steps it takes to achieve a (local) minimum. Put another way, we follow the slope of the surface produced by the objective function downwards until we arrive at a valley[23]. By substituting the scalar learning rate with a positive definite matrix Γ_t that approaches the inverse of the Hessian of the cost at the optimum [11], far better optimization algorithms may be created:

$$w_{t+1} = w_t + \Gamma_t \frac{1}{N} \sum_{i=1}^N \Delta_w Q(z_i, w_t) \quad (2.5)$$

This second order gradient descent (2GD) method is the well-known Newton algorithm spin-off. Second-order gradient descent provides quadratic convergence under suitably optimistic regularity assumptions and given that w_0 is near enough to the optimum[11]. The method achieves the optimum in a single iteration when the cost is quadratic and the precise scaling matrix. Otherwise, we get $\log t$ as assuming adequate smoothness [11].

Meanwhile, each state-of-the-art Deep Learning library includes implementations of different methods for optimizing gradient descent[23]. However, since practical descriptions of their benefits and shortcomings are hard to come by, these algorithms are often employed as black-box optimizers [23].

2.4.2 Adaptive Learning Rate Optimizers

It is challenging to use learning rate schedules because their hyperparameters must be established in advance, and they are very dependent on the kind of model and issue being addressed[60]. A further issue is that all parameter updates use the same

learning rate, which is inefficient. Sparse data may need updating the settings to a different extent.

Training may fluctuate, not converge or skip crucial local minima if the learning rate is too high. With a small threshold, the convergence will be greatly delayed[33]. Learner rate decay is a systematic method for getting around this problem. For example, by employing step decay, the learning rate may be lowered by a certain percentage every few epochs, allowing for a high learning rate at the beginning of training and a lower learning rate towards the conclusion of training[33]. As a result, the decay mechanism must be treated as a hyperparameter on its own and tailored to the specific application.

Traditional SGD may be replaced with adaptive gradient descent algorithms like RMSprop, Adam, Adagrad, and Adadelata [60]. A heuristic approach is provided by per-parameter learning rate approaches, which do not need time-consuming manual tweaking of hyperparameters for determining learning rate schedules[60].

Adagrad

It achieves this by adapting the learning rate to the parameters and making more minor updates for parameters associated with often occurring features and more significant updates for uncommon features[60]. Adagrad is a gradient-based optimization method. Thus, sparse data may be handled with ease because of its flexibility and adaptability[60].

The approach may now be vectorized using a matrix-vector product \odot between G_t and g_t since G_t includes all previous gradient squares along its diagonal concerning all parameters:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \quad (2.6)$$

Adagrad has the advantage of not requiring human adjustment of the learning rate. To keep things simple, most implementations utilize the default value of 0.01 [60].

RMSProp

RMSProp [12] changes the matrix G from the Adagrad update. It does not include the cumulative sum of squared gradients of all parameters in its diagonals to lessen the aggressive, monotonically declining learning rates supplied by Adagrad [33]. The moving average in its diagonals is instead stated as:

$$G_{ii} = \lambda G_{ii} + (1 - \lambda) \left(\frac{1}{m} \sum_{i=1}^m g\theta_i \right)^2 \quad (2.7)$$

With the inclusion of the decay rate λ hyperparameter, the default value of 0.9 has been recommended [33]. In other words, this strategy still modifies each parameter's learning rate according to the size of its gradients, but the moving average prevents updates from growing monotonically smaller [33].

Adam

Another technique for calculating adaptive learning rates is Adaptive Moment Estimation (Adam) [29]. When storing previously squared gradients, Adam uses both Adadelta and RMSprop [22]. Still, it also uses an exponentially decaying average called m_t , the same as momentum but for past gradients.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \quad (2.8)$$

The method's name is derived from estimations of the gradients' first and second moments, referred to as m_t and v_t , respectively. The authors of Adam have found that m_t and v_t are biased towards zero since they are started as vectors of 0s [22]. This is particularly true during the early time steps and when the decay rates are low. As a result, they provide first and second moment estimates that are bias-corrected.

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \quad (2.9)$$

As with the Adadelta and RMSprop updates, they utilize these to change the parameters, resulting in the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (2.10)$$

It is suggested that β_1 and β_2 be set to default values of 0.9 and 0.999, respectively, and ϵ is 1.10^{-8} . Using real-world data, they demonstrate that Adam is an effective adaptive learning strategy[22].

2.5 Batch Normalization

Normalization is a pre-processing procedure for numerical data to bring it to a similar scale without altering its form [53]. Batch normalization is a method for increasing the speed and stability of a deep neural network by adding more layers[53]. It's the job of the new layer to standardize and normalize the input from a prior layer.

A batch of input data is used to train a standard neural network. In batch normalization, the normalizing procedure also occurs in batches rather than as a single input[53]. When data are normalized, the mean and standard deviation are set to zero.

$$\mu = \frac{1}{m} \sum h_i \quad (2.11)$$

Layer h has a total of m neurons. Once the concealed activations have been calculated, the standard deviation will be determined.

$$\sigma = \left[\frac{1}{m} \sum (h_i - \mu)^2 \right]^{1/2} \quad (2.12)$$

To prevent division by zero, the smoothing term (ϵ) ensures numerical stability inside the operation.

$$h_{i(norm)} = \frac{(h_i - \mu)}{\sigma + \epsilon} \quad (2.13)$$

2.6 Activation Functions

In the field of artificial neural networks, an activation function is a function that outputs a small value for small inputs and a more significant value if its inputs surpass a threshold [44]. There is no action taken until enough data is provided to trigger it, in which case it "fires" [44]. So, an activation function is like a gate that checks whether an incoming value exceeds a critical threshold [44].

In conventional matrix notation,

$$\alpha_i^l = w_i^l * k^{l-1} + b_i \quad (2.14)$$

where, w_i^l is the weight column vector associated with the layer l neuron i and k^{l-1} is the column vector related to the output of the layer $(l - 1)$ neurons [52]. The input variables are represented by the vector k^{l-1} when $l = 1$. A differentiable, non linear function $f()$ is often used to calculate the neuron output $f() : k_i^l = f(\alpha_i^l)$. The nonlinear functions $f()$ in this network model are often simple fixed functions like the logistic sigmoid or the tanh functions, and they are referred to as activation functions[52].

2.6.1 Linear Activation

Typically, linear neural networks refer to neural networks that have Linear Activation Functions [46]. A great deal of research has been done on them to understand better how deep learning works. To better comprehend neural networks, these models are essential from an analytical standpoint [46]. Multiple hidden layers in linear neural networks are comparable to a single hidden layer in neural networks.

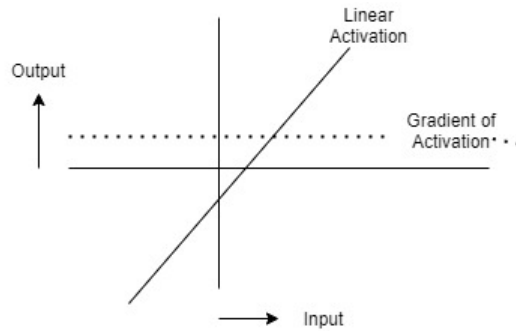


FIGURE 2.6: Linear Activation Functions

A closed-form solution exists for these networks since they are employed for linear regression applications [46]. In the case of linear activation functions,

$$y = X \quad (2.15)$$

where x is the input and y is the output. The gradient of linear activation functions is unity, as shown by Eq. 2.15. Figure 2.6 depicts the activation function and gradient in graphical form.

2.6.2 Sigmoid or Logistic Activation Function

Even though it was created for binary classification, Sigmoid has found use in many other areas, including attention models and limited output regression [44]. The mathematical definition of a sigmoid curve is as follows:

$$\sigma(y) = \frac{1}{1 + e^{-y}} \quad (2.16)$$

Sigmoid gives a value between $[0, 1]$, which may be read as a probability.

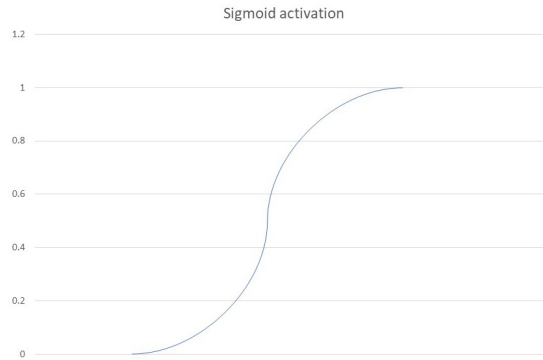


FIGURE 2.7: The sigmoid activation function

The vanishing gradient issue is one of the most significant drawbacks of sigmoids [44]. As shown in Figure 2.7, the sigmoid activation function is constrained by a value of 0 to 1. When used in neural networks, this function can cause a neural network to become stuck during training [44]. It is a generalized logistic activation function used for multiclass classification, called the softmax function [44].

2.6.3 Tanh (Hyperbolic Tan)

Another activation function known as symmetric sigmoid is hyperbolic tan, also known as tanh [46]. A zero-centered, doubly-saturating activation function describes this signal's characteristics.

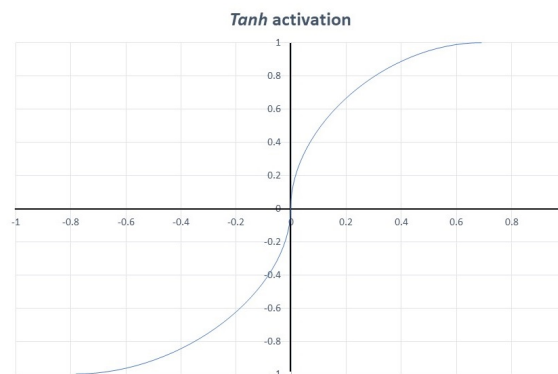


FIGURE 2.8: tanh activation function

Tanh resembles the logistic sigmoid, but it's much superior. The tanh function's parameter range is from $(-1 \text{ to } 1)$ [44]. The benefit is that the tanh graph will map

the negative inputs firmly negative and the zero inputs near zero [44]. It may be expressed mathematically as follows:

$$\tanh(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}} \quad (2.17)$$

From the equations, the tanh has an upper and lower limit of 1 and -1, respectively. Figure 2.8 depicts the symmetric sigmoids. The tanh function is mainly used for separating data into two groups based on similarities [46]. In feed-forward networks, the activation functions tanh and logistic sigmoid are both utilized [46].

2.6.4 ReLU

Feedback stabilization in analog circuits was shown by ReLU in [46]. They demonstrated that networks with ReLU non-linearity always converge to a stable state under certain circumstances [46]. ReLU may be defined mathematically as follows:

$$\text{ReLU}(y) = \max(0, y) = \begin{cases} y, & \text{if } y \geq 0 \\ 0, & \text{if } y < 0 \end{cases} \quad (2.18)$$

At any location other than zero, ReLU is a continuous function that can be differentiated [46]. Stochastic gradient descent and ReLU may seem incompatible; however, given random initialization of structures and the inclusion of a bias factor before applying activation, there is only an infinitesimally tiny likelihood of ending up with an actual zero input to this activation function [46].

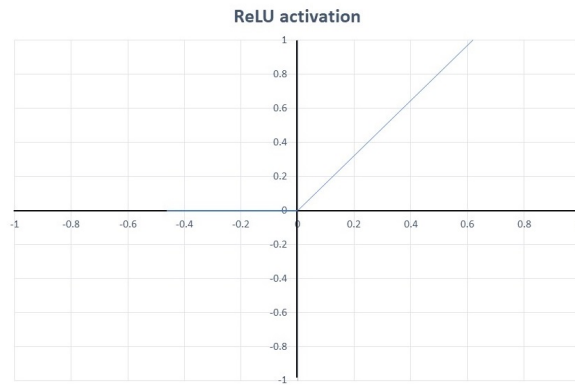


FIGURE 2.9: ReLU activation function

ReLU is by far the most often used activation function in real life [46]. Numerous benefits may be reaped via activating ReLU Fig. 2.9. The ReLU is partially repaired. When y is less than zero, $f(y)$ is zero, and when y is more than or equal to zero, $f(y)$ is equal to y [46].

2.6.5 Leaky ReLU

A novel activation function dubbed leaky ReLU is utilized to circumvent the dying issue in ReLU [46]. The main downside of ReLU is that when it is not enabled, it saturates at zero. This results in a zero gradient and slower training. In order to address these difficulties, a leak is introduced to the activation rather than a complex

zero [46]. LReLU is defined mathematically as:

$$\text{ReLU}(y) = \max(0, y) = \begin{cases} y, & \text{if } y \geq 0 \\ 0.01y, & \text{if } y < 0 \end{cases} \quad (2.19)$$

This leak contributes to the ReLU's increased range.

Figure 2.10 depicts the properties of Leaky ReLU and its gradient. The leak contributes to the ReLU function's increased range. The value of α is usually around 0.01 [46].

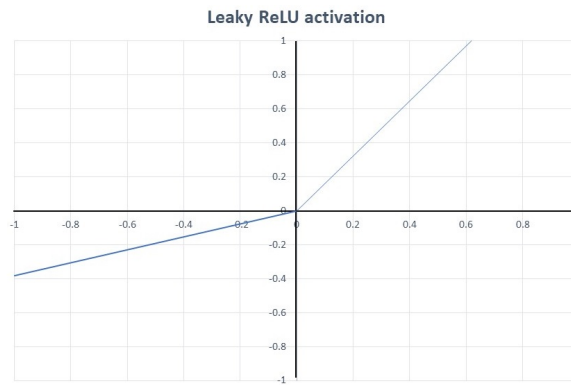


FIGURE 2.10: Leaky ReLU activation function

It's termed Randomized ReLU when α isn't 0.01. As a result, the Leaky ReLU's range is $(-\infty \text{ to } \infty)$. The nature of both the Leaky and Randomized ReLU functions is monotonic. Furthermore, the character of their derivatives is monotone [46].

2.7 Convolutional Neural Networks

Currently, CNN is regarded as one of the most commonly utilized ML techniques, particularly for vision-related tasks. CNN can learn representations from grid-like data, and it's lately demonstrated significant gains in several machine learning applications [39]. Figure 2.11 depicts a typical ML system block diagram. In a typical ML system, CNN capabilities are utilized for feature creation and classification since it has strong feature production and discriminating skills[39].

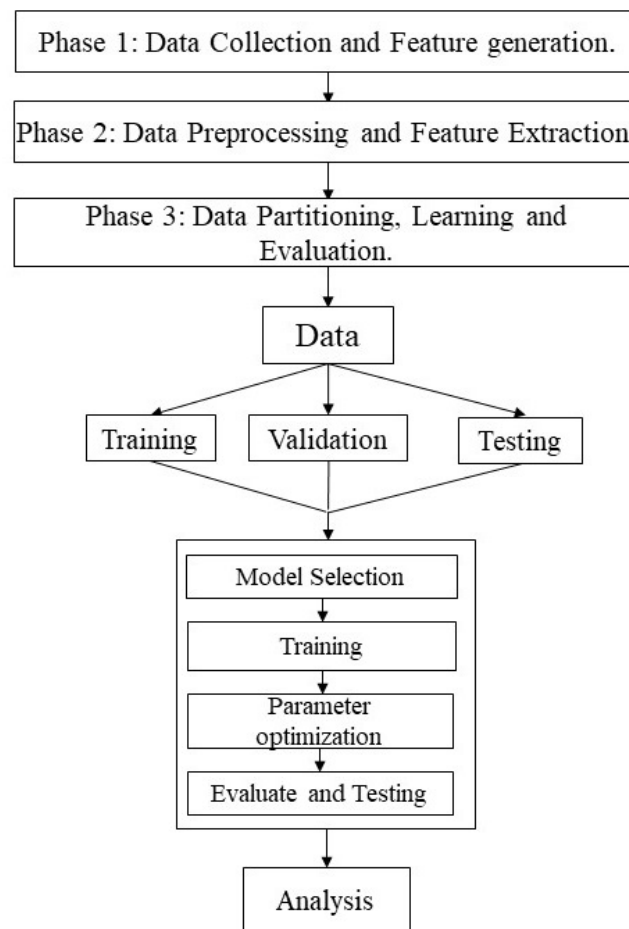


FIGURE 2.11: A typical ML system block diagram

When it comes to machine learning, Deep Learning (DL) is one of the most recent breakthroughs. DL has shown initially near-human and now superhuman abilities in a wide range of applications, such as voice-to-text translations, object recognition, anomaly detection, and recognizing emotions in audio and video recordings, to name a few[55]. In this part, we'll get started with the most basic DL tool: deep (and conventional) CNNs. We'll go over their essential characteristics and blocks, as well as how they work show in figure 2.12.

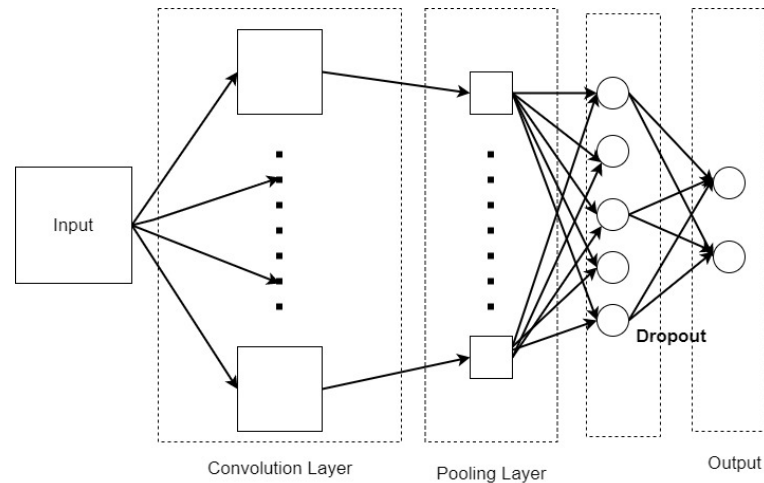


FIGURE 2.12: A CNN with one convolutional, pooling layer and Dropout

A feedforward neural network with convolution components is a convolutional neural network capable of extracting features from data. Different from conventional approaches, CNN does not need manual extraction of features [48]. As a result of this inspiration, CNN's design incorporates elements of visual perception. Biological neurons are equivalent to artificial neurons, and CNN kernels represent multiple types of receptors, each of which may react to a distinct set of properties [48]. Activation functions replicate the way neurons only send electrical impulses that reach a particular threshold. People created loss functions and optimizers to train the whole CNN system to learn the anticipated information [48]. There are several benefits to using CNNs over other types of artificial neural networks.

1. Relationships based on the immediate vicinity. Because each neuron is no longer linked to every other neuron in the preceding layer, but just a few, the parameters may be reduced, and convergence can be accelerated [48].
2. The sharing of weight. The weights of many connections may be shared, thus reducing the number of parameters [48].
3. Dimensionality decrease by downsampling. Downsampling an image using a pooling layer uses local image correlation to minimize the quantity of data while maintaining important information [48].

Pooling layers are very beneficial. By deleting redundant features, the number of parameters may be reduced even more. Because of its three enticing features, CNN has grown to be a leading algorithm in the area of deep learning.

2.7.1 convolutional layer

Each neuron works as a kernel in the convolutional layer, which is made up of convolutional kernels. Convolution becomes a correlation operation when the kernel is symmetric (Ian Goodfellow et al. 2017) [39]. Receptive fields in a convolutional kernel are tiny sections of an image divided into smaller ones by the kernel. It's easier to find feature themes when a picture has been divided into smaller chunks. Kernel concatenates with the pictures by multiplying its components by the corresponding elements of the receptive field according to a predetermined set of weights (Bouvrie

2006) [39]. The following is an example of how to represent a convolution operation:

$$f_l^k(p, q) = \sum_c \sum_{x, y} i_c(x, y) \cdot e_l^k(u, v) \quad (2.20)$$

The convolutional kernel of the k^{th} layer, k_l , is multiplied element-wise by the $e_l^k(u, v)$ index of the input image tensor I_C , and the result is an element of the image tensor [39]. A feature-map of k^{th} convolutional operation's output, on the other hand, maybe described in the following way:

$$F_l^k = [f_l^k(1, 1), \dots, f_l^k(p, q), \dots, f_l^k(P, Q)]$$

Different features inside a picture may be retrieved by sliding kernel with the same weights using convolutional operations due to their ability to share weights [39]. This makes CNN parameters more efficient than fully connected networks because of their shared weights. The kind and amount of filters, type of padding, and direction of convolution all affect the type of convolution operation performed (LeCun et al. 2015) [39].

2.7.2 Pooling layer

The pooling layer downsamples the convolved features nonlinearly. Dimensionality reduction reduces the amount of processing power needed [56]. Overfitting is controlled, and picture translation and rotational variance are eliminated. Performing a pooling process causes the input to be divided into several rectangular patches [56]. No matter what kind of pooling method is employed, each patch is replaced with a single value. Maximum pooling and average pooling are the two distinct approaches [56].

2.7.3 Dropout layer

These prevent overfitting when the trained network significantly depends on characteristics of the training data that do not generalize well to a new, unknown input [28]. By putting hidden units values to zero with a probability $p > (0, 1)$, dropout layers randomly exclude them so other hidden units cannot wholly depend on them [28]. Such techniques are also used in standard artificial neural networks. To reduce dependency on noise or artefact, these layers make the network use more units.

As a result, convolutional layers seek to extract characteristics that are slightly translation invariant[28]. Note that such convolutional layers also have substantially fewer weights / connections. Similar to pooling layers, the network becomes immune to tiny transitions, and its nodes are reduced. Compared to fully linked artificial neural networks, these two adjustments frequently improve classification performance substantially more quickly [28].

Chapter 3

Data Augmentation

The term "Data Augmentation" (DA) refers to techniques for expanding the variety of training data without gathering any extra data in the traditional sense. When training ML models, most approaches either add slightly changed copies of existing data or produce synthetic data to have the augmented data work as a regularizer and prevent overfitting [54]. Many computer vision methods, including cropping, flipping, and color jittering, involve DA as an essential part of the training process. A less obvious way to construct augmented instances that capture desirable invariances when the input space is discrete is in natural language processing (NLP) [54].

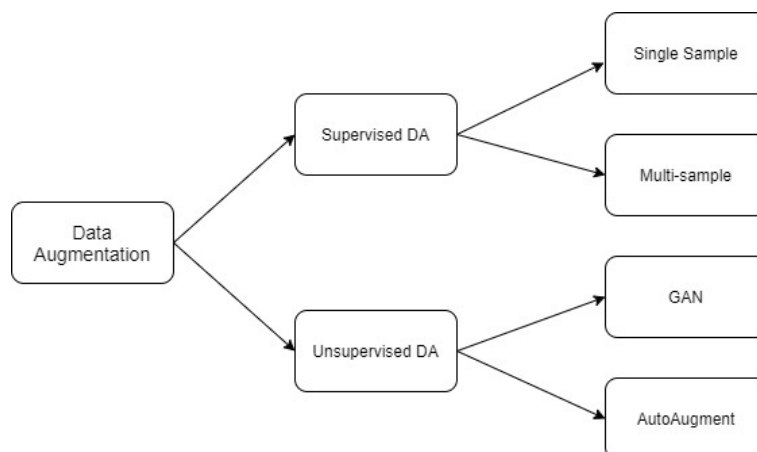


FIGURE 3.1: The Classification of Data Augmentation

Supervised, semi-supervised, and unsupervised data augmentation are all types of data augmentation methods[49]. Figure 3.1 shows the data augmentation categorization. There are two types of Supervised data augmentation: single sample and multi-sample [49]. Supervised data enhancement is based on current data, and it uses pre-set data transformation algorithms to make the changes. It's possible to enrich unsupervised data in two different ways. Data distribution may be learned by models that randomly generate pictures that are consistent with the distribution of the training data set [49]. This approach is known as GAN[15]. Another option is to use the model to learn a data improvement strategy appropriate for the present job. AutoAugment is an example of a symbolic approach.

The initial applications of data augmentation were in computer vision and voice recognition. Despite its late start, text data improvement has made significant progress in recent years, particularly in the domains of text categorization, question answering, and multi-turn discussion [49]. Text data improvement techniques include back

translation, random word replacement, non-core word replacement, text generation model-based data augmentation, and so on [49].

3.1 Existing Methods of Data generation

In the subject of synthesizing tabular data from existing distributions, significant progress has been accomplished. Synthetic data may be created by statistical sampling techniques, deep learning methods, or both to understand actual data distribution [51]. Statistical methods employ a set of pre-defined probability distributions to fit a new tabular dataset. A Gaussian mixture model, for example, may be used to represent the joint distribution of a few continuous columns, whereas Bayesian networks can model the joint distribution of discrete columns [51].

However, because the available probability distributions constrain these models, they are insufficiently generic for various datasets, mainly when they include discrete and continuous columns [51]. People discretize endless queues in survey data so that Bayesian networks or decision trees can model the data. Even if the statistical models are capable, training them is costly, and they can't handle Big datasets with thousands of columns and millions of rows [51]. Another critical group of data synthesizers is deep learning techniques.

The success of such models in computer vision and natural language processing provides the impetus for creating a high-quality deep neural network for tabular data. Variational autoencoders (VAEs) [13] and generative adversarial networks (GANs)[45] are two Deep Generative models, which have two new capabilities:

- The ability to learn a complex high-dimensional probability distribution.
- The ability to generate high-quality samples from pictures or text [51].

These capabilities have enabled several significant applications in vision and language[34].

It's also feasible to create synthetic tabular data using comparable high-quality models: the implicit joint distribution of columns may be learned from actual data, and synthetic rows can be sampled from that distribution. A few models (MedGAN [26], TableGAN [34], PATE-GAN [38]) have been presented that work by applying fully connected networks or convolutional neural networks directly to tabular data without taking into account the unique situation of modeling tabular data. Although these models perform well on datasets, they have not been thoroughly compared against other statistical models [51].

3.2 Data Augmentation Techniques

Techniques for augmenting data may be helpful in the fight against the problems that the artificial intelligence industry is facing today. In order to enhance the performance and results of machine learning models, data augmentation may be used to provide fresh and diverse instances for training datasets [42]. A machine learning model's performance and accuracy improve if the dataset is rich and adequate. Here, we briefly explain some data augmentations that are hugely used in different fields of data [42].

3.2.1 Image Data Augmentation

Computer vision tasks, including picture classification, object recognition, and segmentation, have been particularly influential among popular deep learning applications[42]. In these situations, data augmentation may be a valuable tool for training DL models. Geometric transformations such as Flipping, Rotation, Translation, Cropping, and Scaling, as well as color space transformations such as Color Casting, Varying Brightness, and Noise Injection, may all be applied to a picture with ease [42].

Flipping: It is considerably more frequent to fiddle with the horizontal axis than the vertical axis. This augmentation is simplest to employ and has shown promise on large datasets like ImageNet and CIFAR-10 [42].

Color space: Isolating a single color channel like R, G, or B is a simple way to enhance colors. A picture may be easily turned into a representation in a single color channel by isolating that matrix and adding two zero matrices from the other color channels [42]. Additionally, the brightness of a picture may be adjusted by modifying the RGB values using simple matrix operations.

Cropping: Cropping an image's center patch may be utilized as a helpful processing step for photos with mixed height and width dimensions[42]. Random cropping may also be employed to give a similar effect to translations.

Rotation: To add rotational augmentations, rotate the picture by a value between 1° and 359° on an axis. Increasing rotational safety relies significantly on the degree parameter[42]. For applications like MNIST's digit identification, minor rotations like the ones between 1 and 20 might be advantageous.

Translation: Avoiding positional bias in the data may be accomplished by transforming pictures in either direction. It's possible to fill the space left after translation with a fixed number like 0 seconds or 255 seconds or random noise like Gaussian noise[42].

Noise injection: It is possible to do noise injection by injecting a random value matrix, typically generated from a normal distribution. Using nine datasets from the UCI collection, Moreno-Barea et al. investigate the impact of noise injection [42]. Increasing the amount of noise in pictures may aid in the training of CNNs to recognize more complex patterns.

Color space transformations: Each of the three stacked matrices has a height-width size, which is used to encode image data. In this case, the matrices represent the individual RGB color values as pixel values [42]. Image identification difficulties are typically complicated by lighting biases. As a result, the efficiency of color space transformations, sometimes called photometric transformations, is easy to grasp. Increasing or decreasing the pixel values by a constant number is a simple repair for too bright or dark photographs [42]. Splicing separate RGB color matrices is another rapid color space operation. Pixel values may also be restricted to a minimum or maximum value as another transformation.

Mixing images: Data Augmentation takes a surprising method by combining photos by averaging their pixel values[42]. To the human eye, the pictures created by performing this alteration will not seem to be helpful. The pairing of samples was shown to be a successful enhancement approach by Ionue [42]. The RGB channels' pixel values are averaged to create a new picture. As a consequence, you'll have a messy picture for training a classifier.

Kernel filters: Images may be sharpened or blurred using kernel filters, a standard image processing method. A Gaussian blur filter or a high contrast vertical or horizontal edge filter, which will produce a sharper picture around the edges, are used to slide a $n \times n$ matrix over an image to effect the desired outcome [42]. Visually blurring pictures for Data Augmentation may lead to better test results because of the increased resistance to motion blur. Sharper photos for Data Augmentation may also capture more information about the items of interest because of the additional features [42].

Random erasing: It's noteworthy that Zhong et al. invented a new Data Augmentation approach called random erasing [42]. When it comes to random erasing, it may compare to dropout regularization since it occurs in the input data space rather than inside the network's design as dropout does. This method was developed specifically to deal with problems caused by occlusion in picture recognition [42]. Occlusion occurs when a portion of an item is obscured. As the model learns about a picture, random erasure will prevent it from overfitting to a single visual characteristic [42].

3.2.2 Text Data Augmentation

Due to the significant level of linguistic complexity, enhancing text in the NLP discipline is difficult [57]. Not every word can be replaced with a, an, or the like. There aren't synonyms for every term, either. Context is everything and altering a single word changes everything. However, creating an enhanced picture in the computer vision field is much simpler [57]. Even if someone adds noise to the picture or clip off a piece of it, the model will still identify what it is.

Symbolic augmentation: There is a significant distinction between employing auxiliary neural networks and applying symbolic rules to enhance data while generating data [57]. Human designers gain enormously from symbolic augmentation since it is easily interpretable. Supplemental examples function better with symbolic augmentations, such as substituting words or phrases. Longer inputs, such as answering questions or summarizing, are used in information-heavy applications[57]. The use of symbolic rules to enhance whole phrases or paragraphs is limited.

Rule based augmentation: These augmentations use rules to generate enhanced versions of the original instances. If-else scripts and symbolic templates are used to add new data and reorganize old data as part of this procedure [57]. Wei et al. provide four augmentations in Easy Data Augmentation. Using Easy Data Augmentation off-the-shelf is pretty simple,

which should delight you. Many of the augmentations are still in the research stage, waiting for large-scale testing and acceptance [57]. Random swapping, random deletion, random insertion, and random synonym substitution are methods for quickly enhancing data, with little to no programming knowledge required. Many of the NLP adversarial attacks may be included in rule-based augmentation programs [57]. Using Regular Expression Augmentation is another rule-based option. Enhancement methods based on syntactic heuristics are proposed by Min et al.

Graph structured augmentation: Incorporating graph-structured representations of text data offers an intriguing option for text data enhancement. For example, relation and entity encodings may be found in knowledge graphs, grammatical structures in syntax trees, and language-specific information in citation networks [57]. These enhancements include explicit structural information, which is a novel addition to Deep Learning systems. Finding label-preserving transformations and representation analysis may be made more accessible with the inclusion of structure[57].

MixUp augmentation: New examples are created by meshing together old instances; the labels are occasionally blended in as well. This is known as MixUp Augmentation. MixUp, as an illustration, may concatenate half of one text sequence with half of another sequence in the dataset to produce a new example [57]. In terms of connecting distant spots and illuminating an interpolation route, MixUp is perhaps one of the better interfaces available. The interpolation layer used by most MixUp implementations differs. Word and sentence level MixUp tests are performed by Guo et al. [57].

Feature space augmentation: The phrase "Feature Space Augmentation" refers to enhancing Deep Neural Networks' intermediate representation space with additional data. In most Deep Neural Networks, incoming data is processed into distributed representations and then task-specific predictions using a sequential processing framework [57]. Intermediary features are isolated using Feature Space Augments, and noise is used to create new data instances. This noise may be generated using adversarial controllers or sampled from typical uniform or Gaussian distributions. Feature space augmentations are discussed in MODALS [57].

Neural augmentation: The following enhancements produce new training data using auxiliary neural networks. Using supervised Neural Machine Translation datasets to translate from one language to another and back to sample new instances, or generative language modeling to substitute mask out words or phrases to make new data, is what this implies [57].

Back translation augmentation: It is possible to translate text from one language to another and then back to the original language using a technique known as back-translation. Consider translating 1,000 IMDB movie reviews from English to French, Chinese, or Arabic, then translating those reviews back to English [57]. Machine translation has garnered a lot of attention. As a consequence, massive tagged datasets of parallel phrases have been curated for analysis. Other text datasets, such as translations between computer languages or writing styles, might also be used [57].

Style augmentation: A Picasso-themed dog picture, for example, may serve as an OOD augmentation in a Negative Data Augmentation architecture. In order to avoid overfitting to high-frequency variables or blurring out the shape of language, this is a novel method [57]. The transfer of writing style for applications such as abstractive summarization or context for extractive question answering might be used in the text data domain.

Generative data augmentation: Deep Learning’s Generative Data Augmentation is one of the most intriguing new concepts. Create photorealistic face photos or indistinct text portions are examples of this. In terms of Transfer Learning, these models have been quite beneficial, but the issue remains: What is the killer application for Generative Tasks[57]. This generation’s usage for representation learning and Data Augmentation is vital, but aesthetic applications are just a tiny part of its appeal.

Label augmentation: Supervised learning takes an input (x) and outputs (y), both of which are labeled. We’ve discussed many methods for bringing the x values into line throughout the study. Here, we’ll take a look at studies that aim to disprove their class designations. Knowledge Distillation has been the most effective use of this strategy [57]. With Knowledge Distillation, y labels are transformed from a hard to a soft distribution by re-labeling them with another neural network’s prediction logits. This technology has played a critical role in compressions, such as using DistilBERT, information retrieval, and computer vision classification.

3.2.3 Speech Data Augmentation

SpecAugment was developed by Park et al. to improve speech recognition by augmenting input data[40]. Time warping, frequency masking, and time masking are the three most used methods of enhancing data. LibriSpeech basic (LB), LibriSpeech double (LD), Switchboard moderate (SM), and Switchboard strong are the four possible combinations that they introduce in their experiment (SS)[40].

Time Warping: Tensorflow’s `sparse_image_warp` function uses time warping. It’s like looking at a picture where time is horizontal, and frequency is vertical since it consists of logarithmic τ time steps[40]. Within each time step ($W, \tau - W$), a random point on the horizontal line running through the center of the picture is randomly picked from a uniform distribution ranging from 0 to the time warp parameter W and then warped left-right by a distance w . Each of the boundary’s six anchor points are set in stone: its four corners and the midpoints of its vertical edges[40].

Frequency Masking: Frequency masking is used to mask f consecutive mel frequency channels $[f_0, f_0 + f]$, where f is initially selected from a uniform distribution from 0 to the frequency mask parameter F , and f_0 is selected from $[0, v - f]$. In mathematics, it denotes the number of mel frequency channels available [40].

Time Masking: Time masking is used to mask t successive time steps $[t_0, t_0 + t]$, where t is selected from 0 to the time mask parameter τ , which is a uniform distribution range, and t_0 is selected from $[0, \tau - t]$ [40]. A time mask with an upper limit of p times the number of time steps.

3.2.4 Tabular Data Augmentation

Tabular Data Augmentation (TDA) is a new moniker for a technique of modular feature engineering and observation engineering for tabular data that emphasizes the sequence of augmentation to get the best-projected result for a given information set [50]. It solves the issue of how to best represent data to a learning model for the highest prediction accuracy. An augmentation method is any process that alters the underlying data and results in an increase in the amount or quality of the data [50].

Transformation: Any procedure in which a single input feature creates a new feature or feature is referred to as transformation. Cross-sectional and time-series data may both benefit from transformations[50]. Some transformations are only applicable to time-series data, although a few others are applicable to both. Operations, smoothing functions, select filters, and curve-fitting algorithms may all be used in transformation [50].

Interaction: Any approach that involves more than one feature to produce an extra feature or feature is referred to as an interaction[50]. There are also normalization and discretization strategies, which use many characteristics to complete the computation. Almost majority of these techniques are also applicable to cross-sections [50]. Technical elements in the specialized section and the autoregression model are the only techniques that are time-specific.

Mapping: Mappers are methods for assisting in summarising characteristics by remapping them to meet a goal such as maximizing variability or class separability [50]. Unsupervised techniques are standard, but they may also be supervised. Mapping improves the quality of data by reducing noise from raw data and emphasizing signals [50]. When added to existing data, mappers may enhance the dataset size to take advantage of model interaction effects.

Extraction: When extracting a collection of walk-forward features, someone determined the amount of the time-series history to utilize. Some transformation and interaction approaches are intimately linked with extraction[50]. The distinction is that extraction produces a single value rather than a new series every iteration. As a result, extraction is time-consuming, and reconstructing a whole new series requires numerous iterations. Since extraction is simply a time-series approach, it cannot be used in any cross-sectional transformation or extraction procedures [50].

Statistical model: Classification and regression trees and Bayesian networks are two statistical methods for generating this kind of synthetic data[36]. Ping et al. provide a web-based Data Synthesizer that models correlation using a Bayesian network [36]. Patki et al. present a framework for using copulas to create a relational database recursively.

GANs: In addition to statistical models that generate completely synthetic data, neural models are used to impute missing values in datasets; for example, Gondara and Wang employ deep de-noising autoencoders, while Yoon et al. utilize GAN [36]. Several GAN models have recently evolved to handle tabular data, particularly for the generation of medical records.

Real-valued time-series data may be generated using RGAN and RC-GAN [36]. medGAN , corrGAN , and numerous enhanced models can create discrete medical records but not continuous multimodal information[36]. ehrGAN creates enhanced medical records but does not create synthetic data deliberately[36].

Chapter 4

Generative Adversarial Network

In machine learning, a generative model is a subset of unsupervised learning methods. Machine learning is to create models and algorithms that evaluate and comprehend real-world data [43]. In contrast to conventional classification problems, generative models are one of the most promising methods to accomplish this objective [43]. GAN, a novel framework for estimating generative models through an adversarial process, has shown tremendous promise in producing actual data. This section introduces GAN and its variants, which are relevant to this study.

4.1 Preliminaries

4.1.1 Terminology

Goodfellow et al. (2014) presented the generative adversarial network (GAN) as a solid generative model for a wide variety of applications for the first time[41]. As shown in Figure 4.1, a simple GAN comprises two neural networks: a discriminator D conducts classification between genuine and fake samples, and a generator G generates samples from a data distribution, which is often a low dimensional random noise [41]. The generator has been tuned to deceive the discriminator, and the discriminator has been taught to differentiate between false and genuine samples.

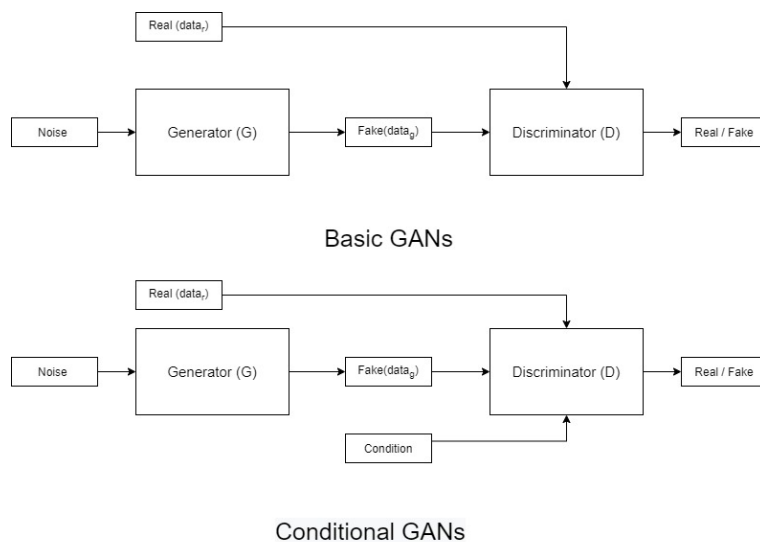


FIGURE 4.1: The Architecture of GANs

Generic models learn to capture the statistical distribution of training data, enabling us to synthesize samples from it. We're interested in utilizing the representations that these models know for tasks like classification and image retrieval in addition to synthesizing new data samples that may be utilized for downstream activities like semantic picture editing, data augmentation, and style transfer [27].

Deep networks with fully connected and convolutional layers are extensions of perceptrons or spatial filter banks with nonlinear postprocessing, which we refer to occasionally [27].

Backpropagation is used to learn the network weights in every instance.

4.1.2 Notation

In general, the GAN literature works with multidimensional vectors, and italics(K) are often used to denote vectors in probability spaces. Vectors are often represented using bold, lowercase symbols in signal processing, and we follow this practice to highlight the multidimensional character of variables [27]. As a result, we'll call $p_d(data)$ the probability density function over a random vector $data$. The distribution of vectors generated by the GAN's generator network will be denoted by the function $g(data)$. The generator and discriminator networks are represented by the characters G and D . W_D and W_G are two sets of weights parameters learned during training via optimization [27].

We define the following notations.

- $data_1, data_2, data_3, \dots$ are the concatenate vectors
- $gumbel_\tau(data)$: apply Gumbel softmax with parameter τ on a vector $data$
- $leaky_\lambda(data)$: apply a leaky ReLU activation on $data$ with leaky ratio λ
- $LT_{a \rightarrow b}(data)$: apply a linear transformation on a input to get b output [51].

We also use \tanh , ReLU , softmax , BN for batch normalization [53], and drop for dropout.

Training, like all deep-learning systems, requires the existence of a specific goal function. The objective functions of the generator and discriminator are referred to as $f_g(W_G, W_D)$ and $f_d(W_G, W_D)$, respectively, using the standard notation [27]. The notation reminds us that the two-goal functions are in some ways reliant on the networks repeatedly updated parameter sets W_G and W_D . In the section "Training GANs," we'll go over this in more depth. Finally, multidimensional gradients are utilized in the updates; we use Δ_{W_G} to indicate the gradient operator concerning the generator parameter weights and Δ_{W_D} to signify the gradient operator for the discriminator parameter weights [27]. $\mathbb{A}\Delta$ is a notation that denotes the anticipated gradients.

4.1.3 Statistic distributions

The issue of density estimation is fundamental in signal processing and statistics: establishing an implicit or explicit representation of data in the actual world, whether

parametric or nonparametric[27]. This incentive mainly drives GANs. The probability mass function of observation data is often referred to as the data generating distribution in the GAN literature. GANs learn by inferring a degree of similarity between a candidate model's distribution and the distribution corresponding to actual data [27].

The trouble of estimating density in the first place is at the core of many visual inference issues, including picture classification, visible object identification and recognition, object tracking, and object registration, to name a few[27]. In theory, Bayes theorem allows all computer vision inference issues to be solved by estimating conditional density functions, which may be done indirectly in a model that learns the joint distribution of variables of interest and observed data[27]. We have a problem in that constructing likelihood functions for high-dimensional, real-world picture data is challenging. While GANs do not explicitly evaluate density functions, the generator implicitly captures the data distribution for a generator-discriminator pair of sufficient capacity[27].

4.2 Generative Adversarial Nets

As a new method to train a generative model, generative adversarial networks were recently presented. They are made up of two 'adversarial' models:

- a generative model G that captures the data distribution
- a discriminative model D that assesses the likelihood that a sample originated from the training data rather than G [35].

Both G and D may be a multi-layer perceptron or another non-linear mapping function.

The generator creates a mapping function from a previous noise distribution $p_n(k)$ to data space as $G(k, \theta_g)$ in order to learn a generator distribution p_{gd} over *data*. And the discriminator, $D(k, \theta_d)$, returns a single scalar indicating the likelihood that *data* originated from training data rather than p_{gd} . We change settings for G to minimize $\log(1 - D(G(k)))$ and for D to minimize $\log D(\text{data})$ as though they were playing a two-player min-max game with value function $f_v(G, D)$:

$$\min_G \max_D f_v(G, D) = \mathbb{E} [\log D(\text{data})] + \mathbb{E} [\log(1 - D(G(k)))] \quad (4.1)$$

4.2.1 Fully Connected

Both the generator and discriminator in the earliest GAN designs were Fully Connected (FC) neural networks [16]. The MNIST (handwritten digits), CIFAR-10 (natural pictures), and the Toronto Face Data Set were all subjected to this kind of design (TFD).

The GAN generator and discriminator networks each include many ultimately linked layers, resulting in a GAN architecture[16]. FC GAN architecture model for producing 32*32*3 images. A sequence of fully connected (FC) layers precedes a series of convolution (CONV) layers in the generator[16]. The FC section takes a low-dimensional (for example, 100) noise vector k and transforms it to a high-dimensional

(4096 features) intermediate representation of image features. A convolution block is created by reshaping this representation ($4 \times 4 \times 256$). The intermediate characteristics are converted to an output picture ($32 \times 32 \times 3$) via a sequence of transposed convolution layers[16]. A stack of convolution layers is followed by an accumulation of FC layers in the construction of the discriminator. An FC component gradually maps them to a lower-dimensional space, allowing an output layer to classify them[16]. For downsampling, average pooling layers are employed instead of stride convolutions in the discriminator.

The Fully Connected layer contains:

- Before convolution layers in the generator, several deep, fully connected layers transform the low-dimensional noise vector to a high-dimensional representation of image features[37].
- In the discriminator, several deep, fully connected layers transfer high-dimensional characteristics retrieved by convolution layers to a lower-dimensional space before classification[37].

By adequately changing the shape and depth of the convolution stack, the architecture illustrated may be modified for pictures of varying resolutions [37]. This research demonstrates the method on four standard picture datasets with three different resolutions (28, 32, and 64 pixels).

4.2.2 Convolutional GANs

Since CNN's are very well adapted to picture data, moving from fully connected to convolutional neural networks (CNNs) is a logical progression[27]. Early tests on CIFAR-10 indicated that training generator and discriminator networks using CNNs of the same capacity and representational power as those used for supervised learning were more challenging[27].

By decomposing a ground truth picture into a Laplacian pyramid and training a conditional, convolutional GAN to generate each successive layer given the one above, the LAPGAN suggested solution proposes to address this issue decomposing the generation process utilizing different scales[27]. Radford et al. have introduced the deep convolutional GAN (DCGAN) family of network designs, which allows for training a pair of deep convolutional generators and discriminator networks[27]. Stride and fractionally strode DCGANs use convolutions to learn the spatial down-sampling and upsampling operators during training. These operators deal with changes in sample rates and locations, which are essential for mapping from image space to potentially lower-dimensional latent space and from image space to the discriminator[27].

Wu et al. demonstrated GANs that could generate three-dimensional (3-D) data samples using volumetric convolutions as an extension of developing pictures in two dimensions [27]. Wu et al. created new items such as chairs, tables, and automobiles, as well as a technique for mapping two-dimensional (2-D) pictures to three-dimensional (3-D) representations of the things.

4.2.3 Conditional Adversarial Nets

Generative adversarial networks may be extended to a conditional model by making the generator and the discriminator dependent on some extra y information, such as class labels or input from other modalities[16]. Conditioning may be accomplished by adding an input layer of y to the discriminator and generator.

The previous input noise $p_k(k)$ and y are merged in a joint hidden representation in the generator[16]. The adversarial training framework allows for significant flexibility in the composition of this confidential representation[16].

In the discriminator, $data$ and y are input to a discriminative function. In a two-player minimax game, the goal function would be Eq 4.2.

$$\min_G \max_D f_v(G, D) = \mathbb{E} [\log D(data|y)] + \mathbb{E} [\log(1 - D(G(k|y)))] \quad (4.2)$$

4.2.4 GANs with inference models

GANs didn't have a method to map a given observation, $data$, to a vector in latent space when they were first developed, which is an inference mechanism in the GAN literature. Several methods have been suggested to invert the generator of trained GANs [27]. Adverbially learned inference (ALI) and bidirectional GANs (BiGANs), both separately developed, provide an inference network in which the discriminators evaluate joint (data, latent) pairs, providing simple but effective expansions[27].

The "encoder" and the "decoder" are the two networks that make up the generator in this formulation. They've been taught how to deceive the discriminator by working together[27]. The discriminator must decide whether pair of $(data, k)$ vectors represents a legitimate tuple consisting of an actual picture sample and its encoding or a false image sample and the matching latent-space input to the generator[27].

The result referred to as a reconstruction of an encoding-decoding model should ideally be comparable to the input. Reconstructed data samples generated using an ALI/BiGAN often have low fidelity[27]. With an extra adversarial penalty on the distribution of data samples and their reconstructions, samples' fidelity may be enhanced [27].

4.2.5 Adversarial Auto Encoder (AAE)

The AAE generator creates a latent code and attempts to deceive the discriminator into thinking that the latent code is drawn from the specified distribution[62]. On the other hand, the discriminator will predict whether a particular latent code was produced by the autoencoder (fake) or by a random vector drawn from the normal distribution (real)[62].

An autoencoder with deep and decoder has x as the input and k as the latent code vector[18]. Let $p(k)$ represent the prior distribution we wish to impose on the codes,

$q(k|x)$ denote the encoding distribution, and $p(x|k)$ denotes the decoding distribution. Let $p_d(x)$ and $p(x)$ represent the data and model distributions, respectively [18]. The autoencoder's encoding function, $q(k|x)$, defines the following aggregated posterior distribution of $q(k)$ on the autoencoder's hidden code vector:

$$q(k) = \int_x q(k|x)p_d(x)dx \quad (4.3)$$

The adversarial autoencoder is a regularized autoencoder that matches an arbitrary prior, $p(k)$, with the aggregated posterior, $q(k)$. As seen in Figure 4.2, an adversarial network is placed on top of the autoencoder's concealed code vector to do this [18]. $q(k)$ is guided to match $p(k)$ by the adversarial network. Meanwhile, the autoencoder tries to reduce the reconstruction error as much as possible. The adversarial network's generator encodes the autoencoder $q(k|x)$. The encoder guarantees that the aggregated posterior distribution may deceive the discriminative adversarial network into believing that the concealed code $q(k)$ originates from the genuine prior distribution $p(k)$ [18].

Stochastic Gradient Descent (SGD) trains both the adversarial network and the autoencoder in two phases: reconstruction and regularization [18]. The autoencoder adjusts the encoder and decoder during the reconstruction phase to reduce the inputs reconstruction error. In the regularization phase, the adversarial network changes its discriminative network to discern the difference between actual and created samples [18]. The adversarial network changes its generator.

There are three different types of encoders:

Deterministic: The encoder will attempt to compress the input into specified characteristics expressed as vector k . Deterministic is the same encoder used in autoencoder [62].

Gaussian Posterior: Gaussian Instead of encoding each feature as a single value, the encoder will record the gaussian distribution of each component with two variables, mean and variance. This is the same encoder used in VAE [62].

Universal Approximator Posterior: This also encodes the characteristics as distribution as Universal Approximator Posterior. Apart from that, we don't assume the feature distribution is gaussian [62]. The encoder in this instance will be a function $f(x, n)$, where x is the input and n is random noise with any distribution.

As a result, the components of the AAE architecture are as follows:

- The encoder transforms the input into a lower dimension (latent code k)
- The decoder will convert the latent code k into the produced picture.

The discriminator receives the autoencoder's encoded latent code k (fake) and a random vector k sampled from the specified distribution (actual) [62]. It will determine whether or not the input is valid.

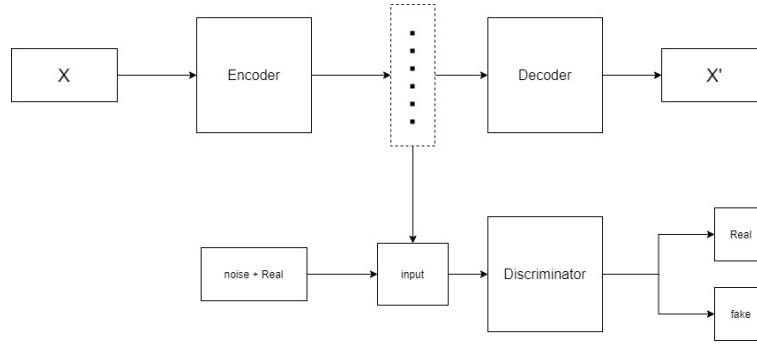


FIGURE 4.2: Architecture of Adversarial Auto Encoder (AAE)

As shown in the architecture above 4.2, the encoder and discriminator are the two major distinctions between AAE and GAN[62]. Instead of generating a random vector k like GAN, AAE uses an dataset as input. This is accomplished by beginning with an encoder. AAE also makes an effort to have the latent code follow a normal distribution[62]. This is accomplished by altering the discriminator jobs to anticipate whether a latent code k is produced by the autoencoder or drawn from a normal distribution[62]. Unlike in GAN, where the discriminator's job is to figure out whether a given data is genuine or fake.

As previously stated, it is common to want a helpful structure of the latent space. A feed-forward, ancestral sampling from an autoencoder may also be beneficial[27]. These two objectives may be met via adversarial training. Adversarial training may be used between the latent space and a desired prior distribution on the latent space in particular (latent-space GAN)[27]. Consequently, a combined loss function is generated, representing both the reconstruction error and a measure of how different the prior distribution is from that provided by a candidate encoding network. This method is similar to that of a variational autoencoder (VAE), in which the latent-space GAN serves as the loss function's Kullback–Leibler (KL)-divergence term[27].

In the guise of the adversarial variational Bayes (AVB) paradigm, Mescheder et al. combined VAEs with adversarial training[27]. offered a similar viewpoint. AVB utilizes an adversarial training goal rather than the KL divergence to maximize the same criteria as VAEs[27].

4.3 Proposed GAN Model

An adversarial game is set up using the GAN framework, with a generator and a discriminator as the two participants. The discriminator's job is to tell apart samples from the model from samples from the training data, whereas the generator's job is to confuse the discriminator as much as possible [17]. Using a minimax value function, we may express the goal.

It won't be long until the random variables and distributions needed to understand Equation 4.1 are defined. The following is a simple explanation of the two terms in the equation in prose for the time being:

- Train the discriminator to maximize the training data's probability.

- Train the discriminator to reduce the likelihood of the generator's data being sampled. The generator should also focus on an opposing goal: to maximize the discriminator's probability to its samples [17].

SGD may be used to train the two players in alternation when they are represented as MLPs by a mathematical model[17].

This paper's contribution is to provide the framework of a conditioning capability. As long as the generator's output and the discriminator's anticipated input are both constrained, we can set up any arbitrary condition y on generation[17]. In other words, we might say that this condition y engages both the generator and the discriminator in a process of generation or prediction.

4.3.1 Generator

There are many deconvolutional layers in the generator G , which makes it a neural network as well. The discriminator's procedure is reversed when G is involved, as illustrated in Figure 4.3 [34]. It takes as input a uniformly sampled latent vector k from the unit hypercube space [34]. The input k is transformed into a 2-dimensional matrix, corresponding to a synthetic table row, using several de-convolutional layers.

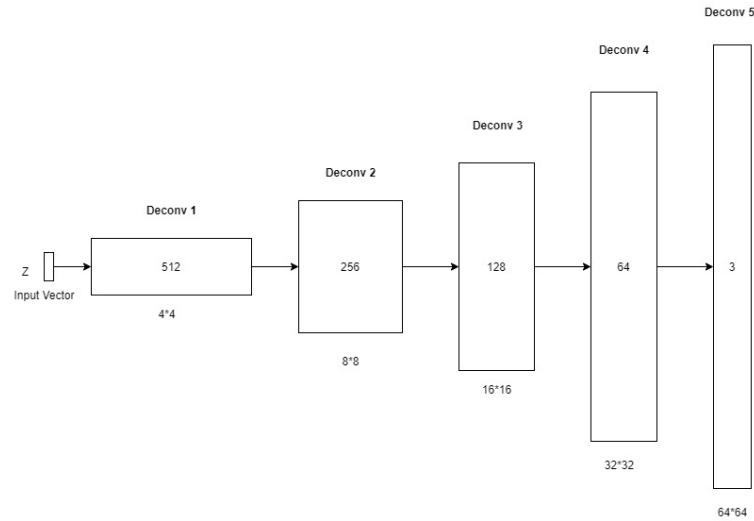


FIGURE 4.3: Architecture of the generator with 5 deconvolutional layers

The generation of numerical data takes place in two stages. We begin by creating the value scalar s_i , and then the cluster vector c_i follows suit. We produce a probability distribution over all potential labels to represent a category characteristic in a single step [36].

LSTM has a hidden state and output size of n_h . Depending on the previous output, the prior hidden vector k_{t-1} or an embedding vector k_{t-1}^i are used as input to the LSTM in each step[36]. The weighted context vector w_{t-1} is also an input. The random variable k is composed of n random elements. There are N dimensions in all, and each one is sampled $N(0, 1)$. Attention-based context vector w_t weighs all previous LSTM outputs $L_{1:t}$ in equal proportion. There are n_h dimensions to this vector[36]. We create an attention weight vector $w_t \in \mathbb{R}^t$ and use it to calculate context

as

$$w_t = \sum_{i=1}^t \frac{\exp w_{t,i}}{\sum_j \exp w_{t,j}} L_i \quad (4.4)$$

Set w_0 to be equal to 0. Its output is L_t , which we project to a hidden vector $f_t = \tanh(LP_L L_t)$, where LP_L is an input parameter that the network has learned over time. f_t has a radius of n_f . The concealed vector is then turned into an output variable.

- $b_i = \tanh(LP_t f_t)$ is the formula for calculating the value portion of a continuous variable. f_t is the value of the hidden vector at time $t + 1$.
- The result is computed as $a_i = \text{softmax}(LP_t f_t)$ if a continuous variable cluster component. f_t is the feature vector for $t+1$.
- The output is a discrete variable if it is $d_i = \text{softmax}(LP_t f_t)$. $f'_t = E_i [\arg \max d_i]$ represents the discrete variable D_i as an embedding matrix $E \in \mathbb{R}^{|D_i| \times n_f}$ for time step $t+1$.
- f_0 is a particular vector that we learn throughout training.

The discriminator's prediction results may be used to train the generator. That's the idea behind it: the several options for training the generator in our table-GAN [34]. It is possible to use back-propagation to execute this training procedure efficiently.

4.3.2 Discriminator

Using Discriminator, it will be possible to tell if data is legitimate or malicious. More hostile instances will be generated by G based on the detection result, just as it would with regular data[43].

The network that can distinguish one thing from another, D is a neural network taught to differentiate between synthetic records and genuine ones in a table [34]. D is a convolutional neural network (CNN) with many layers. In each layer, the whole input matrix is subjected to a set of trainable filters. Remember that in our approach, the records are transformed into square matrices[34]. As a result, the output layer size is inversely proportional to the number of filters included inside it.

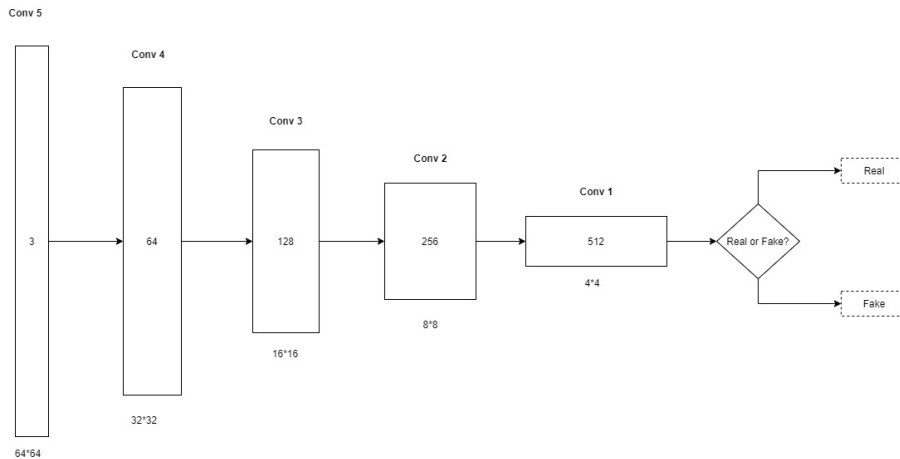


FIGURE 4.4: Architecture of the discriminator with 5 convolutional layers

According to Figure 4.4, a layer's output is its following layer's input. After the final sigmoid activation layer, the depth of intermediate tensors decreases, and the likelihood of being genuine or synthetic increases[34]. In addition to batch normalization and LeakyReLU, additional intermediary layers impact the network's functioning[34].

For this project, the Discriminator is a fully connected neural network of l layers. This function takes three arguments, all of which form a hash $b_{1:n_c}$, $a_{1:n_d}$ and $d_{1:n_d}$. This is how we arrived at the internal layers:

$$f_1^D = \text{LeakyReLU}(\text{BN}(lP_1^D(b_{1:n_c} \oplus a_{1:n_d} \oplus d_{1:n_d}))) \quad (4.5)$$

$$f_i^D = \text{LeakyReLU}(\text{BN}(lP_i^D(f_i^D \oplus \text{diversity}(f_i^D)))), i = 2 : l, \quad (4.6)$$

When the symbol \oplus represents the concatenation operation, the mini-batch discrimination vector is $\text{diversity}(f_i^D)$. Using a previously learned distance metric, the diversity vector's dimensions are the total distances between each sample and the other samples in the mini-batch[36]. There is a leaky reflect linear activation function, $\text{LeakyReLU}()$, and $\text{BN}()$ is batch normalization(BN). Further, the discriminator's output is calculated as a scalar, $lP_i^D(f_i^D \oplus \text{diversity}(f_i^D))$ [36].

One actual or synthetic record is sent into the first layer, which creates a $x * x$ matrices input. By training it on real data, the Discriminator can predict 1 for actual data while predicting 0 for fake data[34].

Any records with zero padding and are reshaped into a square matrix have more characteristics in the original table than the input size can hold[34]. It is possible to customize our model and let it learn from a large table of data.

4.3.3 Loss Function

Training neural networks are based on the loss function. In general, neural networks are taught by minimizing a loss function, and poorly constructed loss functions degrade the training process and malfunction to neural networks[34]. As a result, the choice of loss functions for training the three suggested neural networks is critical to the success of table syntheses[34]. Using DCGAN's loss function as a starting point, we create two additional loss functions:

- To begin, we'll use the initial loss from DCGAN, which is given in Equation 1.
- If two synthetic and actual records statistics differ by a certain amount, there is an information loss.
- Discrepancies between the anticipated and synthesized labels are referred to as classification loss[34].

The classifier uses the classification loss to train, whereas the discriminator uses the DCGAN loss to train[34]. We train the generator with all three loss functions since it's the essential one in our system. Individual loss functions are discussed in detail in this section.

Original Loss

Equation 1 depicts the initial GAN loss function. When it comes to accuracy, both the discriminator and the generator are taught to increase it and reduce it, respectively. Here we see the GANs training methodology in action[34]. While the generator attempts to deceive the discriminator, it can significantly improve its synthesis capability. As L_{orig}^D and L_{orig}^G , we use the original loss definitions in our approach instead. They are in charge of instructing the discriminator and the generator, respectively, on how to work[34].

Information Loss

Before the discriminator network's sigmoid activation, we extract features to measure information loss[34]. The discriminator determines the authenticity of the inputs based on these characteristics.

In other words, the retrieved features should include essential properties of your input data. After flattening, retrieved features often form very high-dimensional matrices[34]. When referring to them as vectors, we'll use the strong typeface f . The following is the most basic example of information loss:

$$L_{mean} = \| \mathbb{E}[f_x] - \mathbb{E}[f_{G_k}] \|_2 \quad (4.7)$$

where f represents features retrieved from the discriminator's final layer, and $\mathbb{E}[f_x]$ represents the dataset's average feature. Note that we quantify the difference between two mean features by using the Euclidean norm[34]. Hence, the purpose of L_{mean} is to compare the characteristics of actual and synthetic records based on their first-order statistics[34]. Second-order statistics, such as the standard deviation, are also used as follows:

$$L_{sd} = \| \text{SD}[f_x] - \text{SD}[f_{G_k}] \|_2 \quad (4.8)$$

Standard deviation (SD) is defined SD . if L_{mean} and L_{sd} are both zero, then real and synthetic data have the same statistical properties when seen through the extracted features are from the last layer of the discriminator[34]. As a result, it's possible the discriminator won't be able to tell them apart.

The synthesis process quality should be within your control. If your companions are untrustworthy, you may not want to share a synthetic table that closely resembles the actual table[34]. If that's the case, creating a low-quality table may be a good idea. As a result, we construct the following loss to train the generator:

$$L_{info}^G = \max(0, L_{mean} - \delta_{mean}) + \max(0, L_{sd} - \delta_{sd}) \quad (4.9)$$

where $\max()$ is used to execute the loss until a specified quality deterioration threshold is reached. As long as L_{mean} or L_{sd} is below a threshold value δ_{mean} or δ_{sd} , there will be no information loss due to L_{info}^G . Mean, and standard deviation are two variables used to regulate the degree of privacy[34]. The privacy level will be reduced, and the synthetic table will resemble the real one if these settings are kept modest.

Classification Loss

Values in synthetic records don't always line up with labels. Classification loss as L^C is another loss function we designed to prevent this problem.

$$L_{info}^C = \mathbb{E} [|\ell(x) - C(rmv(x))|], L_{info}^G = \mathbb{E} [|\ell(k) - G(rmv(k))|] \quad (4.10)$$

Where, $\ell()$ and $rmv()$ are function that returns and removes the label attribute from an input record, whereas $C()$ is the label predicted by the classifier neural network. A record's label and the classifier's projected label differ, and this loss measures the difference between the two[34].

Classifier neural networks may be extended to do multi-task learning if the labels are numerous, and the classifier includes several distinct final sigmoid activations that share many intermediate layers[34]. For each sigmoid activation, the typical intermediate layers are used to train it to predict a label.

We also discovered that synthetic records do not always remember all of the data from the original table, even when mean = 0 and sd = 0 are used as filter parameters. In our tests, we were surprised to find that the suggested classification loss may, in many instances, solve this issue[34]. This is another benefit of using categorization loss techniques.

4.4 Training GANs

First, to train GANs, identify the discriminator parameters that optimize classification accuracy and then identify the generator parameters that confuse the discriminator the most. Figure 5 depicts the training procedure.

$f(G, D)$ is used to assess the training costs since it is dependent on both the generator and the discriminator to work. It is necessary to solve

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

where

$$f(G, D) = \mathbb{E}_{p_{data}(data)} \log D(data) + \mathbb{E}_{p_g(data)} \log(1 - D(data)).$$

One model's parameters are updated while the second model's parameters remain constant throughout training. A static generator is shown to have unique $D^*(data) = p_{data}(data) / (p_{data}(data) + p_g(data))$. For $p_g(data) = p_{data}(data)$, G is equal to the optimum discriminator predicating 0.5 for all samples taken from $data$, as shown by the researchers. The generator is at its best when the discriminator D is completely confused and unable to distinguish between actual samples and false ones.

To get the best results, train the discriminator until it's as good as possible with the current source, and then update the original again [27]. When training the discriminator for a limited number of trials, the discriminator may only be taught for a few iterations before being used with a newer generation. For the generator, this means that instead of using $\min_G \log(1 - D(G(k)))$, an alternative nonsaturating training criteria is employed, such as $\max_G \log D(G(z))$.

No matter how many unique solutions exist in theory, GAN training is difficult and unstable for various reasons[27]. Assessing empirical "symptoms" that may be encountered during training may help improve GAN training. Among the signs are:

- problems with model convergence
- As various inputs are used, the generative model "collapses" to produce extremely similar examples
- The discriminator loss rapidly reduces to zero, leaving no route for the generator's gradient to be updated reliably.

As a result, several writers proposed heuristic methods to deal with the problems [27].

To begin, researchers Goodfellow and Salimans et al. hypothesized that GAN training is unstable since the answer to the optimization issue it poses is a saddle point, which is why gradient descent techniques are usually employed to update both the generator's and discriminator's parameters. The example is given by Salimans et al. demonstrates this [27]. However, stochastic gradient descent is often used to update neural networks, and well-developed machine-learning programming environments make it simple to build and update networks using stochastic gradient descent.

Despite an early theoretical approach showing that the generator is optimum when $p_g(data)$ are equal to $p_{data}(data)$, a very tidy conclusion with a strong intuition behind it, actual data samples are found on a manifold in high-dimensional space representations[27]. Consider a data table sample with the values range $[0, \mathbb{R}^+]$ of size $N * N$, which has a dimensionality of N^2 . Each dimension accepts values between zero and the highest detectable intensity of the cell value. This is referred to as the \mathbb{X} space. When using p_{data} , the data samples provide assistance for a specific issue. Still, they usually only take up the tiniest fraction of the total available space, \mathbb{X} . Likewise, the generator's samples should only take up a tiny part of \mathbb{X} .

According to Arjovsky et al. [26], the support functions $p_g(data)$ and $p_{data}(data)$ are located in a lower-dimensional space than the one corresponding to the \mathbb{X} coordinate. The two functions has no overlap, which means that the discriminator can accurately differentiate between genuine and false samples with a near-trivial amount of effort[27].The discriminator error will rapidly converge to zero in this situation.

The generator's parameters can only be updated via the discriminator, which means that any gradients used to update the generator's parameters would eventually converge to zero, making them useless for future generator updates. Several of the symptoms associated with GAN training are explained by Arjovsky et al. [26].

In addition, Goodfellow et al. demonstrated that training G to minimize the Jensen-Shannon (JS) divergence between $p_g(data)$ and $p_{data}(data)$ is equal to reducing D , when D is optimum. The update may be less accurate or less significant if D is not optimum[27]. This theoretical discovery has sparked research into alternative distance-based cost functions.

4.4.1 Training tricks

For example, Lei Xu and Kalyan Veeramachaneni introduced the TGAN architectures for training GANs to generate tabular data as one of the first significant advances in this area [36]. CNN architectures have been extensively explored in computer vision before, and this study produced a set of recommendations on how to build and train the generator and discriminator. We discussed the significance of stride and fractionally stride convolutions in the section on "Convolutional GANs" [27]. These are essential structural elements [36]. In this way, the discriminator and the generator may enhance the quality of data synthesis by learning about appropriate upsampling and downsampling procedures. Batch normalization was suggested for usage in both networks to stabilize training in more complex models.

Another idea was to reduce the number of wholly linked layers to train more complex models more efficiently. Finally, Radford et al. demonstrated that leaky rectifying linear units (ReLUs) activation functions across the discriminator's intermediate layers outperformed conventional ReLUs in terms of discrimination quality.

Different heuristic methods for stabilizing GAN training were later suggested by Salimans et al. [36]. Features matching alters the goal of the generator significantly to provide more information. It's important to note that the discriminator is still taught to recognize genuine from fake samples. Still, the generator is now trained to match the anticipated intermediate activations of its false samples to the expected intermediate activations of the actual samples. An additional input is added to the discriminator using minibatch discrimination, encoding the distance between a particular minibatch sample and the other samples [36]. The discriminator can quickly determine whether the generator is generating the same outputs, which is meant to avoid mode collapse.

Algorithm 1 GANs Training Algorithm [34]

Input: Real Sample: $\{data_1, data_2, \dots, data_n\}$

Output: a Generative Model G

- 1: $G \leftarrow \text{Generator}$
- 2: $D \leftarrow \text{Discriminator}$
- 3: **while** upto loss values synchronization **do**
- 4: Create a mini-batch of real sample $data = \{data_1, data_2, \dots, data_n\}$
- 5: Create a set of latent vector inputs $k = k_1, k_2, \dots, k_n$
- 6: Train the discriminator D by

$$\Delta_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D(data_i) + \log(1 - D(G(k_i)))]$$

- 7: Train the generator G by

$$\Delta_{\theta_g} \frac{1}{m} \sum_{i=1}^m [\log(1 - D(G(k_i)))]$$

- 8: **end while**
 - 9: **return** G
-

Iterative optimization is another technique that penalizes network parameters that vary from an average of initial values, which aids in convergence to equilibrium. Four, virtual batch normalization minimizes a sample's reliance on other samples in a minibatch by generating batch statistics for normalizing using a sample taken from a fixed reference minibatch at the start of training—virtual batch normalization [36]. By smoothing a discriminator's classification border, one-sided label smoothing prevents an overconfident discriminator from providing weak gradients for the generator, which would lead to an overconfident discriminator providing gradients of 0.9 instead of 1. Snderby et al. proposed that samples be subjected to noise before being fed to the discriminator to put the discriminator to the test. According to Snderby et al., the optimal discriminator is biased by one-sided label smoothing. In contrast, instance noise moves the manifolds of the natural and fake samples closer together, preventing the discriminator from quickly finding a discrimination boundary that separates the real and fake samples entirely [36]. By annealing the standard deviation over time, it is possible to introduce Gaussian noise to synthetic and genuine data. Arjovsky et al. developed a similar method separately.

Algorithm 2 TGANs Training Algorithm with classifier [34]

Input: Rea Sample: $\{data_1, data_2, \dots, data_n\}$

Output: a Generative Model G

- 1: $G \leftarrow \text{Generator}$
 - 2: $D \leftarrow \text{Discriminator}$
 - 3: $C \leftarrow \text{Classifier}$
 - 4: **while** upto loss values synchronization **do**
 - 5: Create a mini-batch of real sample $data = \{data_1, data_2, \dots, data_n\}$
 - 6: Create a set of latent vector inputs for G $k = k_1, k_2, \dots, k_n$
 - 7: Update the discriminator D by SGD with L_{orig}^D
 - 8: Update the classifier C by SGD with L_{class}^C
 - 9: $f_{mean}^{data} = w * f_{mean}^{data} + (1 - w) * \mathbb{E}[f_{data}]$
 - 10: $f_{sd}^{data} = w * f_{mean}^{sd} + (1 - w) * \text{SD}[f_{data}]$
 - 11: $f_{mean}^k = w * f_{mean}^k + (1 - w) * \mathbb{E}[f_{G(k)}]$
 - 12: $f_{sd}^k = w * f_{mean}^{sd} + (1 - w) * \text{SD}[f_{G(k)}]$
 - 13: Update the discriminator D by SGD with

$$L_{orig}^G + L_{info}^G + L_{class}^G$$
 - 14: **end while**
 - 15: **return** G
-

The algorithm 1 illustrates the basic GAN training idea. Neuronal networks may take on any shape or size since G and D are not restricted. Both D and G aim to maximize and minimize the object respectfully, but only D and G successfully do this [34]. Instead of telling actual samples apart from created ones, the discriminator D uses a generator G to create convincing false samples that the discriminator

can't tell apart from real ones. The discriminator may be seen as a mentor and the generator as a pupil. The instructor gives the pupil comments on the quality of their work. Because it is the most developed model and its neural network designs rely upon many other GANs, we based our table-GAN techniques on the DCGAN [34].

The stochastic gradient descent (SGD) update based on mini-batches is used by deep learning algorithms that deal with big datasets [34]. This is also how we do things since it's more scalable [34]. Yet another issue with mini-batch training is the inability to compute the global mean and standard deviation for actual and synthetic data. As a result, we estimate them using an exponentially weighted moving average in Algorithm 2. The mean feature of a genuine or synthetic mini-batch, for example, is denoted by $\mathbb{E}[f_{data}]$ or $\mathbb{E}[f_{G(k)}]$. Thereby, it calculates f_{mean}^{data} and f_{mean}^k as global mean features. It is recommended that the weight w in the moving average computation should be near 1 to maintain a steady global mean and standard deviation [34].

To train the classifier,

1. start with L_{orig}^D
2. then go on to L_{class}^C
3. finally to $L_{orig}^G + L_{info}^G + L_{class}^G$

Making theoretical complexity calculations for deep learning algorithms is time-consuming and pointless since the training process includes many sophisticated neural network operators [34]. Deep learning algorithms are difficult to understand. In our tests, our algorithm takes no more than 30 minutes to learn. It's easy to create fake records after you've been taught.

Figure 2 depicts the generator's input, which is a latent vector named k . The generator takes a random sample of k from the unit hypercube space and uses it [34]. It creates a single synthetic record as a result of its operation. In comparison to the training process, the generating process is far more manageable.

Chapter 5

Experiment and Result Evaluation

The goal of the implementation is to produce a bunch of new data with the help of the Generative Adversarial Networks. To produce a new bunch of data with similar characteristics like existing data is data augmentation. Using GAN to produce image data augmentation is widespread in the research group. In this thesis, our motive is to use the same algorithm for numerical and categorical tabular data to data augmentation. Preparing the dataset is one of the essential parts of any implementation.

5.1 Experimental Environment

For the experiment, we chose to use the cloud system that Google provides. It's a hugely used platform for researchers and scientists to do their research using the python programming language. The platform is known as Google Colab. As a regular version of Google Colab, provide the following configuration to the users.

System name	System Value
vendor_id	GenuineIntel
model name	Intel(R) Xeon(R) CPU @ 2.20GHz
cpu cores	1
cpu MHz	2199.998
cache size	56320 KB

TABLE 5.1: Configaretion details of Google Colab

With that configuration, our project takes more than 30 minutes to train and generate tabular data.

5.2 Python Librarys

Some supported library is required to implement the algorithm. Python is the programming language we use to put the algorithm into action. Panda, NumPy, Matplotlib, Seaborn, TensorFlow, Keras, Sklearn, Warning, Missingno, and Scipy are examples of Python libraries we used in implantation.

Panda: Data manipulation and analysis are two of the most common uses of this module in Python. Data from a variety of sources may be read and modified using this package.

Numpy: With this python library, huge, multi-dimensional matrices may be formatted using a wide variety of high-level mathematical operations.

Matplotlib: Matplotlib is a powerful python toolkit for charting data and information. It is used extensively.

Seaborn: Python's Seaborn module is mainly used to create graphical representations of statistical data. Based on Matplotlib, Seaborn's visualization library is tightly linked with the panda's data structures in Python and uses panda's data for plotting.

Tensorflow: Machine learning and artificial intelligence software library TensorFlow is free and open source. It may be used for various tasks, although its primary emphasis is on deep neural network training and inference. Tensorflow is a toolkit for symbolic math that uses dataflow and differentiable programming to make computations more efficient.

Keras: Keras is a Python-based deep learning API built on top of TensorFlow's machine learning framework. It was created to facilitate quick experiments. The ability to quickly go from a concept to a result is essential for effective research.

Sklearn: For the Python programming language, there is scikit-learn/Sklearn, a free machine learning package. It includes support vector machines, random forests, gradient boosting, k-means, and DBSCAN, as well as a variety of classification, regression, and clustering algorithms, and it's built to work with Python's NumPy and SciPy numerical and scientific libraries.

Warning: When a warning message is used to inform the user of a condition in a program, but raising an exception and stopping the program isn't necessary (usually), the warning message is sent. When software uses an out-of-date module, for instance, you may wish to send a warning.

Missingno: Missingno is a Python module that makes it possible to see how many values are missing from a dataset.

SciPy: In Python, SciPy is a free, open-source library for tackling mathematics, science, engineering, and technology issues. It offers a wide variety of high-level Python commands for data manipulation and visualization. Based on Python NumPy, SciPy was created. SciPy may be pronounced as "Sigh Pi".

5.3 Dataset

We'll use a new dataset this time to see what the model's predictions are. Finding a good dataset for the experiment was a difficult task. For category and numerical data, we are implementing our model. The dataset, including both sorts of data, thus, is difficult to locate. We began by using a dataset about the European automobile industry. From 2011 until 2021, you'll find details on both old and new cars. A small sample of the whole data set is shown below in table 5.2.

	mileage	make	model	fuel	gear	offerType	price	hp	year
0	23500	BMW	316	Diesel	Manual	Used	6800	116.0	2011
1	92800	Volkswagen	Golf	Gasoline	Manual	Used	6877	122.0	2011
2	149300	SEAT	Exeo	Gasoline	Manual	Used	6900	160.0	2011
3	96200	Renault	Megane	Gasoline	Manual	Used	6950	110.0	2011
4	156000	Peugeot	308	Gasoline	Manual	Used	6950	156.0	2011

TABLE 5.2: A dataset about the European automobile industry

Another collection of data that we utilized in the experiment is made up of student test scores. Categorical and numerical data are both included in this collection. The dataset is shown in the following figure 5.4.

	gender	race/ ethnicity	parental level of education	lunch	test prepa- ration course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

TABLE 5.3: A dataset of student test scores

5.4 Data Analysis

Preparing data for using them in the proper algorithm, many processes are needed to fulfill the dataset. Developing the dataset with accurate data and rearranging them in the data frame is part of data analysis. Data cleaning, inspection, transformation, and modeling are the main factors considered in this section. I also integrated all the required processes of data analysis for my proposed algorithm.

Data Cleaning:

The first step is to read the CSV file from the local machine through the panda and convert the data into a panda frame. Using the `panda.head()` makes it easier to check the dataset, including the index, which helps to get an overview of the dataset. We reduce all unnecessary columns and data rows by selecting the panda's column name and drop methods. It's an easier way of data cleaning.

Data Inspection:

We have done several types of inspections on our dataset in terms of data inspection. To find out the missing data is the most important one. We have applied to fill the missing value with the mean of the column. If, on the other hand, the missing value exceeds 70%, the column should be removed. Another inspection is to separate the columns which contain numeric values and categorical values.

Data Transforming:

In the dataset, there have some columns which contain categorical data. It's challenging to use this type of data in different mathematical calculations. So, the easiest way is to convert those data into corresponding numerical values. There have lots of methods to do this type of conversion. We used a dummy data method for this transformation. In this method, we assignee is converted actual data into indicator numeric value.

Data Normalization:

As part of the data preparation process, normalization is often used. Numerical columns in a dataset may be normalized to utilize a common scale without distorting or losing information by changing their values to match this scale. Some algorithms need the data to be normalized in order to represent it properly. The `MinMaxScaler()` function is used to standardize the data.

5.5 TGAN Model Setup

We use the processed data set to test our TGAN model. Furthermore, since we employed both category and numerical data, we tested the performance in three distinct methods. One is for numerical data, another is for categorical data, and the third is for a mixture of categorical and numerical data. In addition, we run the model on two distinct datasets to see how they vary. The initial step in the model setup is to choose specific hyperparameters for the model. We fixed the hyperparameter as follows:

$$\begin{aligned} latent_{dim} &= 100, \\ n_{epochs} &= 5000, \\ n_{batch} &= 8000, \\ n_{eval} &= 100. \end{aligned}$$

We also experiment with other hyper-parameter values. However, with that parameter value, we eventually receive a satisfactory outcome. The flowchart in figure 5.1 below depicts how the procedure works.

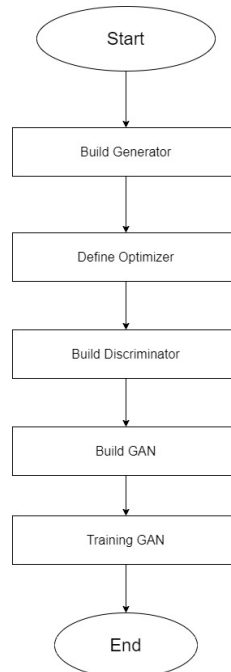


FIGURE 5.1: The flowchart of proposed method

For various sorts of data sets, we utilize the same flow. As shown in the diagram, the initial step in this GAN model is to construct the generator. As we can see from

the theoretical discussion, the generator plays a vital role in the generation of new data. The flowchart of the building generator model is shown below in figure 5.2.

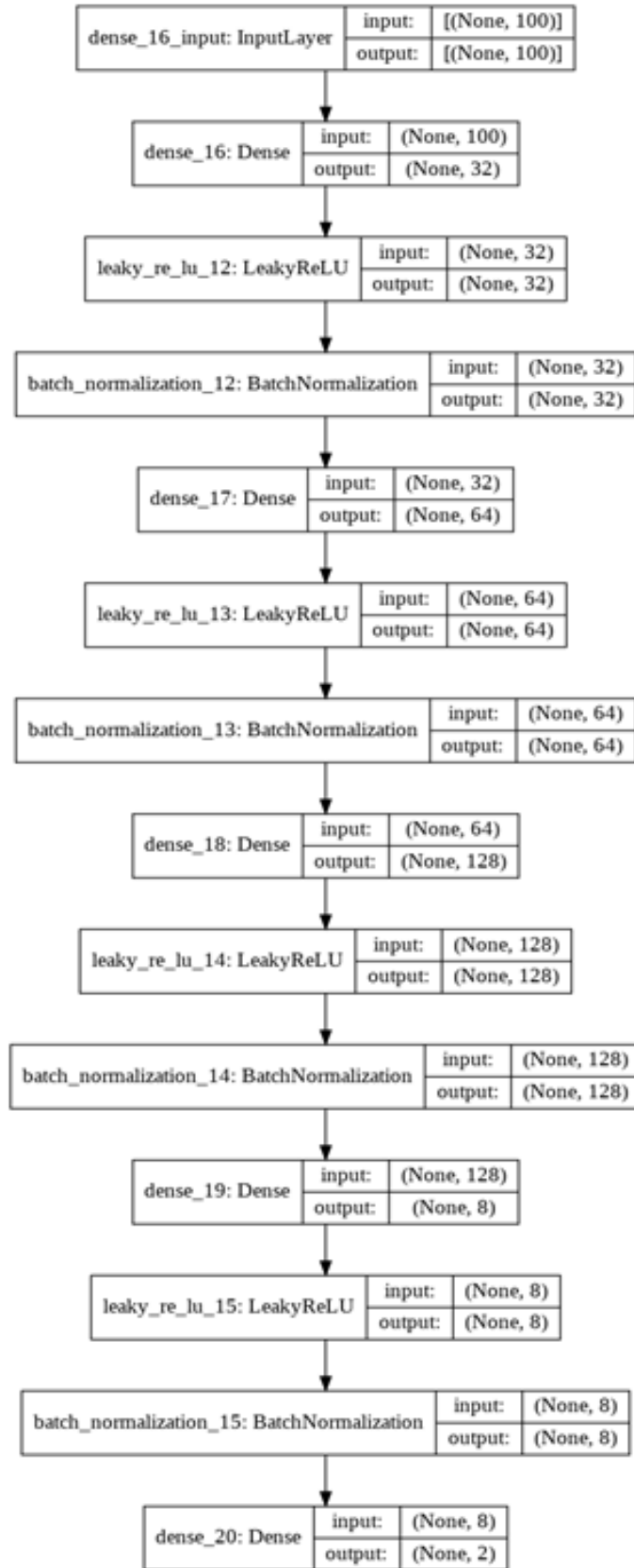


FIGURE 5.2: The flowchart of the building generator model

The following stage is the discriminator, which is used to examine and determine the correctness of created data. This discriminator is responsible for the whole performance. Because it is an essential rule for our GAN model, this model's generator and discriminator are two mutually dependent terms. The process flow in discriminator pieces is shown in in figure 5.3 below.

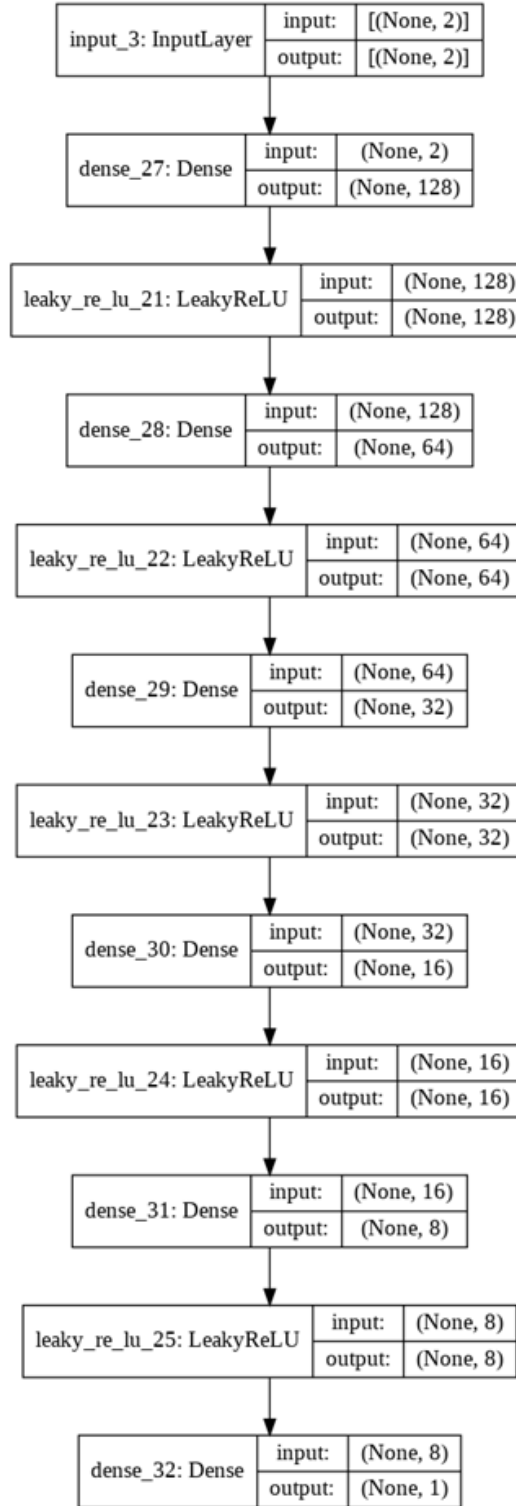


FIGURE 5.3: The flowchart of the building discriminator model

The final model, which we designate as GAN, combines the generator and discriminator models. We'll train our model with our supplied data set when we finish building this final model. As we explained in the data analysis section, we prepared our data via a series of steps. The flow chart that is used in the GAN model is also shown here in figure 5.4.

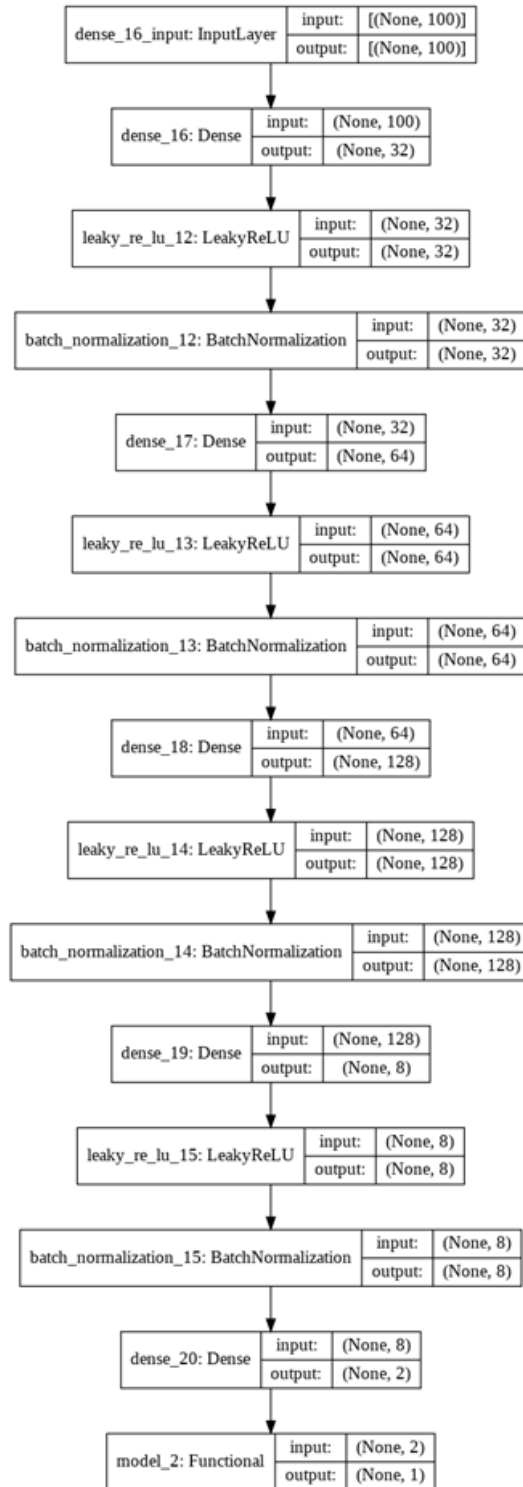


FIGURE 5.4: The flowchart of the building GAN model

We obtain our output after training the model with the desired input and hyperparameter. We can extract some useful information from our output and summarize our trials if we analyze it.

5.6 Performance Analysis

We encountered several differences in experience when we incorporated numerical and categorical data for various datasets. It is highly typical for all computations to be performed mathematically, making it simple to use numerical data. And there's a chance you'll receive a more accurate result in the output if you use numerical data. We start with a dataset in table 5.4 that contains information on new and used cars from 2011 through 2021.

	mileage	price	hp	year
0	23500	6800	116.0	2011
1	92800	6877	122.0	2011
2	149300	6900	160.0	2011
3	96200	6950	110.0	2011
4	156000	6950	156.0	2011

TABLE 5.4: Actual numerical data used as Input

As illustrated in the diagram 5.5, we obtain the following result. As we can see from the diagram, the loss of the generator and discriminator was quite different. The loss of the generator is brought to the same place to loss of discriminator by raising the epochs. As a consequence, it seems that they are learning to provide more accurate results.

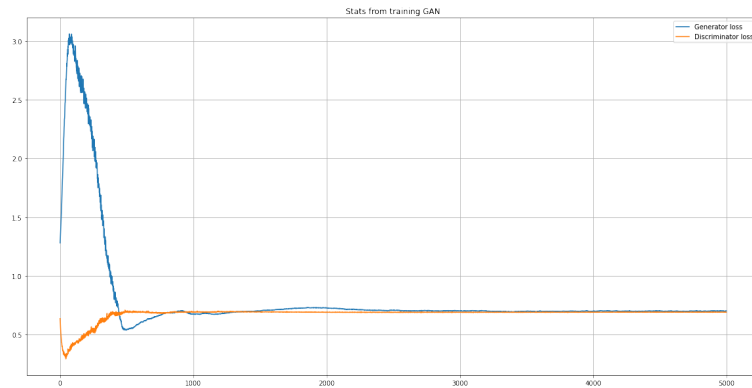


FIGURE 5.5: From Table 1 Numerical data train with GAN

If we take a look on the data that are generated from TGAN in the table, then we understand its not same as the input data but the values are symmetrical of table 5.4.

	mileage	price	hp	year
0	5.714193e+10	1.990812e+10	84972.0	22164.0
1	3.290286e+09	5.752948e+10	183475.0	22220.0
2	1.680598e+11	2.000913e+09	108931.0	22121.0
3	1.796668e+11	7.170141e+09	136219.0	22162.0
4	5.176544e+10	3.075806e+09	79564.0	22169.0

TABLE 5.5: GAN generated synthesized numerical data

On the other hand, if we observe the identical graph for a student's performance, that is measured in numerical terms. It's also the same performance as the one before it. And in this cases, we obtain the table 5.6 below as a result the left side is the input data and the right is the synthesized data.

	math score	reading score	writing score		math score	reading score	writing score
0	72	72	74		79.0	74.0	76.0
1	69	90	88		78.0	78.0	85.0
2	90	95	93		64.0	75.0	73.0
3	47	57	44		86.0	87.0	88.0
4	76	78	75		51.0	60.0	57.0

TABLE 5.6: Actual data and GAN generated synthesized numerical data

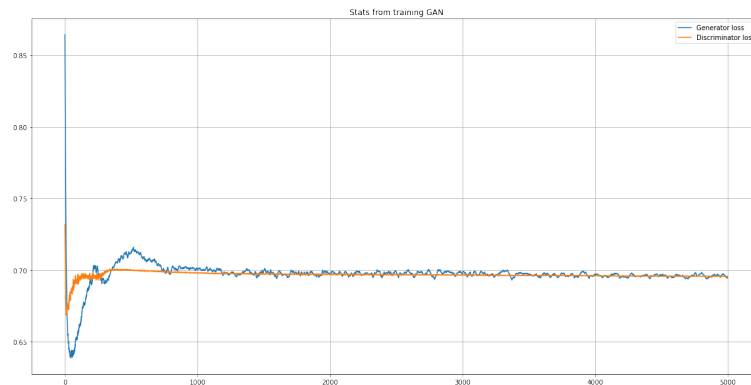


FIGURE 5.6: From Table 2 Numerical data train with GAN

The model for categorical data for both datasets is the next plane. We must transform categorical data to dummy data since it is not feasible to utilize categorical data directly. When we utilize this dummy data, it increases the table's feature based on the unique values. For example, if a table has gender information, there are two distinct values (male, female).

	gender_female	gender_male
0	1	0
1	1	0
2	1	0
3	0	1
4	0	1

FIGURE 5.7: Dummy value of gender feature from table

Then, based on the unique value, separate that table feature in figure 5.7. Then apply the dataset to the GAN model. The graphical depiction in the figure 5.6 is for the automobile information data. From the figure we understand that the model gives quite good accuracy on categorical data values.

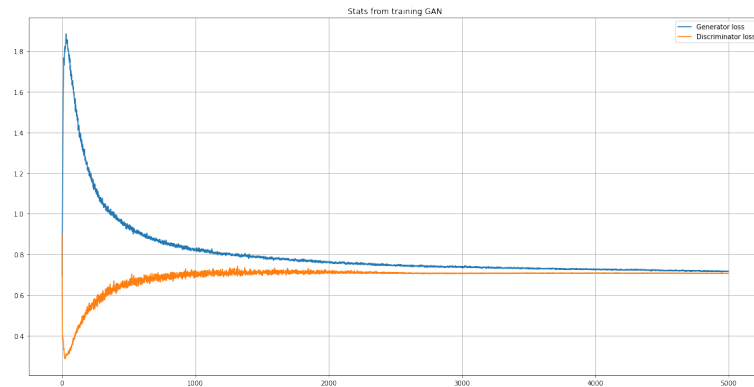


FIGURE 5.8: From Table 1 One feature Categorical data train with GAN

We obtain the following graph in the figure 5.8 when we utilize student performance. We can observe that the generator and discriminator losses are approaching equilibrium in both cases when the period increases fast.

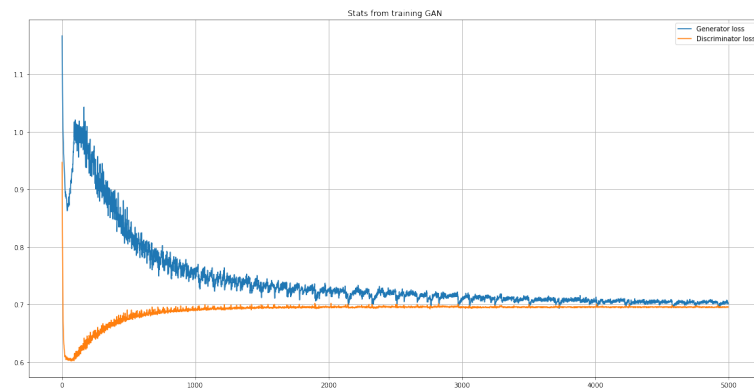


FIGURE 5.9: From Table 2 One feature Categorical data train with GAN

Furthermore, as we employ that feature with more unique values like 'model' data shape: (46071, 835). that means 'model' data feature has 835 unique values. So, the number of dummy data for 'model' have 835 features. And we are going to use this 'model' data in the GAN. We obtain the following result in the figure 5.9, when we apply it to the GAN. It shows that the loss for the generator and discriminator are moving in opposite directions.

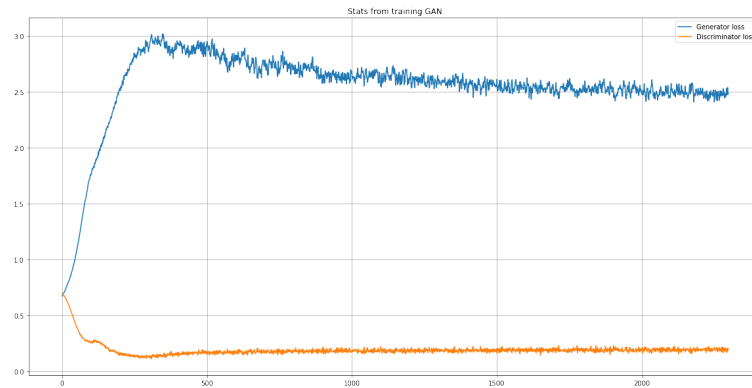


FIGURE 5.10: From Table 1 two Categorical feature and numeric data train with GAN

For the car information dataset, When both categorical and numerical features are used together, the resultant graph is not the same as trained individually. Because the number of the feature is huge for the GAN. As a result we will get vary distorted output shown in the figure 5.9. But if we compare the same scenario for student test score data we get the result in the figure 5.11, which is more suitable then the car information result. Because, It has only 7 feature in total in the input data.

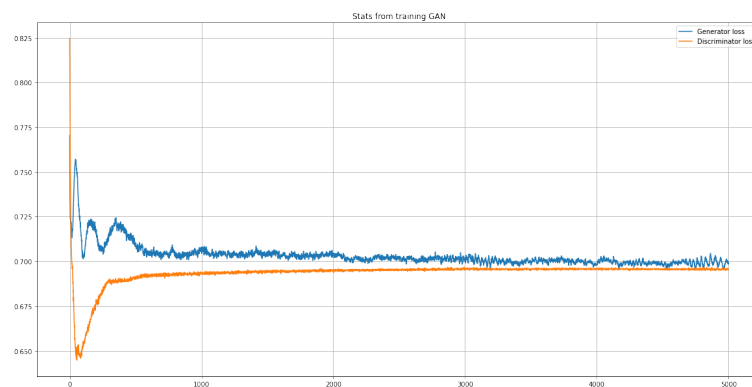



FIGURE 5.11: From Table 2 two Categorical feature and numeric data train with GAN

So, based on the conversation and the accompanying graph, we can conclude that if the data table has more unique values in categorical features, the GAN model's performance suffers and produce a very distorted result at the end.



	gender_female	gender_male	test preparation course_completed	test preparation course_none	math score	reading score	writing score
0	1.0	0.0	0.0	1.0	64.0	67.0	74.0
1	1.0	0.0	1.0	0.0	84.0	96.0	98.0
2	0.0	1.0	0.0	1.0	82.0	63.0	65.0
3	0.0	1.0	0.0	1.0	78.0	79.0	69.0
4	0.0	1.0	0.0	1.0	62.0	60.0	48.0
...
9995	1.0	0.0	1.0	0.0	93.0	98.0	99.0
9996	1.0	0.0	0.0	1.0	60.0	77.0	72.0
9997	0.0	1.0	0.0	1.0	88.0	88.0	86.0
9998	1.0	0.0	1.0	0.0	75.0	83.0	84.0
9999	0.0	1.0	1.0	0.0	47.0	46.0	43.0

10000 rows × 7 columns

FIGURE 5.12: GAN generated synthesized including categorical and numeric data

In the figure 5.12 is represented the final output of the GAN generated synthesized data, containing the categorical and numerical features. As we can see that the categorical features are shown in dummy data format. And we need to apply the process of work to retrieve the actual data from the dummy value. On the other hand, if we look at the numerical value, it seems pretty perfect and similar to the input data.

Chapter 6

Conclusion and Future work

6.1 Conclusion

In the field of data augmentation, the generative adversarial network is a beneficial and potent technique. In 2014, the first GANs were proposed. After then, GAN evolved into a valuable tool for data augmentation due to several adjustments and various types of a study conducted on it. We may infer from the findings of the research that GANs are extensively employed in image data augmentation. GAN is quite successful in this area. As of now, we plan to include the GAN-based augmentation into the tabular data set. It's a never-ending quest for better ways to supplement data. To generate relational tables with both continuous and discrete data, the TGAN model utilizes a GAN-based algorithm. On GAN-based technology, image augmentation is more straightforward than tabular data augmentation.

GAN has a lot of promise for relational data synthesis applications. In terms of classification, clustering, and AQP, it's advantageous since it creates synthetic data. Additionally, it delivers competitive performance when it comes to safeguarding personal information from being reidentified. Higher data utility cannot be produced with the existing method while maintaining differential privacy, which is a drawback of GAN.

Transforming data from original records into GAN-recognized input impacts overall performance, demonstrating the importance of representing relational data. A hybrid optimization methodology that co-trains GAN and record representation might lead to some intriguing future work.

We find that GANs are better suited to massive datasets because they can successfully capture correlations between features. That is, we demonstrate that using our approach, we can produce useful synthetic data. The usage of relational databases is widespread, yet modeling them is very challenging. Our architecture is capable of supporting a single table with both numerical and categorical data elements. Eventually, we'll look at utilizing GAN to model sequential data and many tables. Some approaches have difficulty combining tabular data with noise or anomaly. Such instances are inaccessible to the TGAN.

For instance, the model picks up on the basics of data produced from a clean and analyzed data collection. Our method's capabilities may be summed up as follows:

1. Production of high-quality tabular data in real-time, based on specified characteristics.
2. Collection and analysis of any size dataset.

3. The processing time required increases linearly with the quantity of output data.
4. Numeric and categorical data may be processed successfully.

6.2 Future work

Machine learning hyper-parameters are parameters that influence how well a system learns. Other parameter's values (such as node weights) are obtained by experimentation. It is possible to classify hyper-parameters as either model hyper-parameters, which cannot be inferred from the training data because they are related to the model selection task. On the other hand, algorithm hyper-parameters do not affect the model's performance but how quickly and well it learns. Although hyper-parameter selection may have a considerable impact on the model's performance, finding optimal values can be difficult.

Choosing the proper collection of values while developing a machine learning model, in our instance, the machine learning model is GAN, may be termed Hyperparameter Optimization or Hyper-parameter Tuning.

Bibliography

- [1] David E. Rumelhart and James L. McClelland. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.
- [2] Thomas H. Davenport and Laurence Prusak. *Working Knowledge: How Organizations Manage What They Know*. Harvard Business School Press, Jan. 1992.
- [3] J. E. Dennis and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, Jan. 1996. ISBN: 9780898713640. DOI: 10.1137/1.9781611971200. URL: <https://epubs.siam.org/doi/book/10.1137/1.9781611971200> (visited on 11/03/2021).
- [4] Daniel Svozil, Vladimír Kvasnicka, and Jirí Pospichal. "Introduction to multi-layer feed-forward neural networks". In: *Chemometrics and Intelligent Laboratory Systems* 39.1 (1997), pp. 43–62. ISSN: 0169-7439. DOI: [https://doi.org/10.1016/S0169-7439\(97\)00061-0](https://doi.org/10.1016/S0169-7439(97)00061-0). URL: <https://www.sciencedirect.com/science/article/pii/S0169743997000610>.
- [5] S.S. Haykin. *Neural Networks: A Comprehensive Foundation*. International edition. Prentice Hall, 1999. ISBN: 9780139083853. URL: <https://books.google.de/books?id=M5abQgAACAAJ>.
- [6] Balázs Csanád Csáji. "Approximation with Artificial Neural Networks". MA thesis. Eindhoven University of Technology, June 2001.
- [7] Carlos Gershenson. "Artificial Neural Networks for Beginners". In: *CoRR* cs.NE/0308031 (2003). URL: <http://arxiv.org/abs/cs/0308031>.
- [8] Ajith Abraham. "Artificial Neural Networks". In: *Handbook of Measuring System Design*. American Cancer Society, 2005. Chap. 129. ISBN: 9780471497394. DOI: <https://doi.org/10.1002/0471497398.mm421>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471497398.mm421>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/0471497398.mm421>.
- [9] Murat Hüsnü Sazli. "A brief review of feed-forward neural networks". In: 2006, pp. 11–17.
- [10] Ludovic Arnold et al. "An Introduction to Deep Learning". In: *European Symposium on Artificial Neural Networks (ESANN)*. Proceedings of the European Symposium on Artificial Neural Networks (ESANN). Bruges, Belgium, Apr. 2011. URL: <https://hal.archives-ouvertes.fr/hal-01352061>.
- [11] Léon Bottou. "Stochastic Gradient Descent Tricks". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 421–436. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_25. URL: https://doi.org/10.1007/978-3-642-35289-8_25.
- [12] Srivastava N. Hinton G. and K. Swersky. "Lecture 6a: Overview of minibatch gradient descent." In: *COURSERA: Neural Networks for Machine Learning*. 2012.

- [13] Diederik P Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *arXiv:1312.6114* (Dec. 2013).
- [14] L. Deng and D. Yu. *Deep Learning: Methods and Applications*. Foundations and trends in signal processing. Now Publishers, 2014. ISBN: 9781601988140. URL: <https://books.google.de/books?id=46qNoAEACAAJ>.
- [15] Ian J. Goodfellow et al. "Generative Adversarial Nets". In: *Machine Learning* (June 2014). URL: <https://arxiv.org/abs/1406.2661v1>.
- [16] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *CoRR abs/1411.1784* (2014). arXiv: 1411.1784. URL: <http://arxiv.org/abs/1411.1784>.
- [17] Jon Gauthier. "Conditional generative adversarial nets for convolutional face generation". en. In: (2015).
- [18] Alireza Makhzani et al. "Adversarial Autoencoders". In: *CoRR abs/1511.05644* (2015). arXiv: 1511.05644. URL: <http://arxiv.org/abs/1511.05644>.
- [19] James Martens and Roger B. Grosse. "Optimizing Neural Networks with Kronecker-factored Approximate Curvature". In: *CoRR abs/1503.05671* (2015). arXiv: 1503.05671. URL: <http://arxiv.org/abs/1503.05671>.
- [20] Jürgen Umbrich, Sebastian Neumaier, and Axel Polleres. "Quality Assessment and Evolution of Open Data Portals". In: *2015 3rd International Conference on Future Internet of Things and Cloud*. Aug. 2015, pp. 404–411. DOI: 10.1109/FiCloud.2015.82.
- [21] LeCun Y, Bengio Y, and Hinton G. "Deep learning". In: *Nature* 521 (2015), 436–444. DOI: 10.1038/nature14539. URL: <https://pubmed.ncbi.nlm.nih.gov/26017442/>.
- [22] *An overview of gradient descent optimization algorithms*. en. Jan. 2016. URL: <https://ruder.io/optimizing-gradient-descent/> (visited on 11/08/2021).
- [23] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR abs/1609.04747* (2016). arXiv: 1609.04747. URL: <http://arxiv.org/abs/1609.04747>.
- [24] Francesco Calimeri et al. "Biomedical Data Augmentation Using Generative Adversarial Neural Networks". In: *Artificial Neural Networks and Machine Learning – ICANN 2017*. Cham: Springer International Publishing, 2017, pp. 626–634. ISBN: 978-3-319-68612-7.
- [25] Francesco Calimeri et al. "Biomedical Data Augmentation Using Generative Adversarial Neural Networks". In: *Artificial Neural Networks and Machine Learning – ICANN 2017* 10614 (2017), pp. 626–634. URL: https://doi.org/10.1007/978-3-319-68612-7_71.
- [26] Edward Choi et al. "Generating Multi-label Discrete Patient Records using Generative Adversarial Networks". In: *arXiv:1703.06490* (Mar. 2017).
- [27] Antonia Creswell et al. "Generative Adversarial Networks: An Overview". In: *CoRR abs/1710.07035* (2017). arXiv: 1710.07035. URL: <http://arxiv.org/abs/1710.07035>.
- [28] Fabian Gieseke et al. "Convolutional neural networks for transient candidate vetting in large-scale surveys". In: *Monthly Notices of the Royal Astronomical Society* 472.3 (2017), 3101–3114. ISSN: 1365-2966. DOI: 10.1093/mnras/stx2161. URL: <http://dx.doi.org/10.1093/mnras/stx2161>.

- [29] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (2017). arXiv: 1412.6980 [cs.LG].
- [30] Saeed Rouhani, Sayna Rotbei, and Homa Hamidi. "What do we know about the big data researches? A systematic review from 2011 to 2017". In: *Journal of Decision Systems* 26.4 (2017), pp. 368–393. DOI: 10.1080/12460125.2018.1437654. URL: <https://doi.org/10.1080/12460125.2018.1437654>.
- [31] Dina Sukhobok, Nikolay Nikolov, and Dumitru Roman. "Tabular Data Anomaly Patterns". In: *2017 International Conference on Big Data Innovations and Applications (Innovate-Data)* (Aug. 2017), pp. 25–34. URL: <http://link.aip.org/link/?RSI/69/1236/1>.
- [32] Christian Huyck and Dainius Kreivenas. "Implementing Rules with Artificial Neurons". In: *Artificial Intelligence XXXV*. Ed. by Max Bramer and Miltos Petridis. Cham: Springer International Publishing, 2018, pp. 21–33. ISBN: 978-3-030-04191-5.
- [33] Thomas Neff. "Data Augmentation in Deep Learning using Generative Adversarial Networks". MA thesis. Graz University of Technology, Mar. 2018.
- [34] Noseong Park et al. "Data synthesis based on generative adversarial networks". In: *Proceedings of the VLDB Endowment* 11.10 (2018), 1071–1083. ISSN: 2150-8097. DOI: 10.14778/3231751.3231757. URL: <http://dx.doi.org/10.14778/3231751.3231757>.
- [35] Giorgia Ramponi et al. "T-CGAN: Conditional Generative Adversarial Network for Data Augmentation in Noisy Time Series with Irregular Sampling". In: *CoRR abs/1811.08295* (2018). arXiv: 1811.08295. URL: <http://arxiv.org/abs/1811.08295>.
- [36] Lei Xu and Kalyan Veeramachaneni. "Synthesizing Tabular Data using Generative Adversarial Networks". In: *CoRR abs/1811.11264* (2018). arXiv: 1811.11264. URL: <http://arxiv.org/abs/1811.11264>.
- [37] Sukarna Barua, Sarah Monazam Erfani, and James Bailey. "FCC-GAN: A Fully Connected and Convolutional Net Architecture for GANs". In: *CoRR abs/1905.02417* (2019). arXiv: 1905.02417. URL: <http://arxiv.org/abs/1905.02417>.
- [38] James Jordon, Jinsung Yoon, and Mihaela van der Schaar. "PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: <https://openreview.net/forum?id=S1zk9iRqF7>.
- [39] Asifullah Khan et al. "A Survey of the Recent Architectures of Deep Convolutional Neural Networks". In: *CoRR abs/1901.06032* (2019). arXiv: 1901.06032. URL: <http://arxiv.org/abs/1901.06032>.
- [40] Daniel S. Park et al. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition". In: *Interspeech 2019* (2019). DOI: 10.21437/interspeech.2019-2680. URL: <http://dx.doi.org/10.21437/Interspeech.2019-2680>.
- [41] Yanmin Qian, Hu Hu, and Tian Tan. "Data augmentation using generative adversarial networks for robust speech recognition". In: *Speech Communication* 114 (2019), pp. 1–9. ISSN: 0167-6393. DOI: <https://doi.org/10.1016/j.specom.2019.08.006>. URL: <https://www.sciencedirect.com/science/article/pii/S0167639319300044>.

- [42] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (July 2019), p. 60. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: <https://doi.org/10.1186/s40537-019-0197-0> (visited on 11/04/2021).
- [43] Qiao Yan et al. "Automatically synthesizing DoS attack traces using generative adversarial networks". In: *International Journal of Machine Learning and Cybernetics* 10 (Feb. 2019), pp. 3387–3396. URL: <https://doi.org/10.1007/s13042-019-00925-6>.
- [44] *Activation Function*. Sept. 2020. URL: <https://deeptai.org/machine-learning-glossary-and-terms/activation-function> (visited on 11/03/2021).
- [45] Ian J. Goodfellow et al. "Generative adversarial networks". In: *Review of Scientific Instruments* 63.11 (Nov. 2020), pp. 139–144. URL: <https://doi.org/10.1145/3422622>.
- [46] Mohit Goyal et al. "Activation Functions". en. In: *Deep Learning: Algorithms and Applications*. Ed. by Witold Pedrycz and Shyi-Ming Chen. Studies in Computational Intelligence. Cham: Springer International Publishing, 2020, pp. 1–30. ISBN: 9783030317607. DOI: 10.1007/978-3-030-31760-7_1. URL: https://doi.org/10.1007/978-3-030-31760-7_1 (visited on 11/03/2021).
- [47] Kevin Kuo. "Generative Synthesis of Insurance Datasets". In: 1912.02423, *arXiv.org* (Aug. 2020).
- [48] Zewen Li et al. "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects". In: CoRR abs/2004.02806 (2020). arXiv: 2004.02806. URL: <https://arxiv.org/abs/2004.02806>.
- [49] Pei Liu et al. "A Survey of Text Data Augmentation". In: 2020 *International Conference on Computer Communication and Network Security (CCNS)*. 2020, pp. 191–195. DOI: 10.1109/CCNS50731.2020.00049.
- [50] Derek Snow. *DeltaPy: A Framework for Tabular Data Augmentation in Python*. en. SSRN Scholarly Paper ID 3582219. Rochester, NY: Social Science Research Network, Apr. 2020. URL: <https://papers.ssrn.com/abstract=3582219> (visited on 11/04/2021).
- [51] Lei Xu. "Synthesizing tabular data using conditional GAN". eng. Thesis. Massachusetts Institute of Technology, 2020. URL: <https://dspace.mit.edu/handle/1721.1/128349> (visited on 11/05/2021).
- [52] Andrea Apicella et al. "A survey on modern trainable activation functions". In: *Neural Networks* 138 (June 2021). arXiv: 2005.00817, pp. 14–32. ISSN: 08936080. DOI: 10.1016/j.neunet.2021.01.026. URL: <http://arxiv.org/abs/2005.00817> (visited on 11/03/2021).
- [53] *Batch Normalization | What is Batch Normalization in Deep Learning*. en. Mar. 2021. URL: <https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/> (visited on 11/08/2021).
- [54] Steven Y. Feng et al. "A Survey of Data Augmentation Approaches for NLP". In: CoRR abs/2105.03075 (2021). arXiv: 2105.03075. URL: <https://arxiv.org/abs/2105.03075>.
- [55] Serkan Kiranyaz et al. "1D convolutional neural networks and applications: A survey". In: *Mechanical Systems and Signal Processing* 151 (2021), p. 107398. ISSN: 0888-3270. DOI: <https://doi.org/10.1016/j.ymssp.2020.107398>. URL: <https://www.sciencedirect.com/science/article/pii/S0888327020307846>.

- [56] D. R. Sarvamangala and Raghavendra V. Kulkarni. "Convolutional neural networks in medical image understanding: a survey". en. In: *Evolutionary Intelligence* (Jan. 2021). ISSN: 1864-5917. DOI: 10.1007/s12065-020-00540-3. URL: <https://doi.org/10.1007/s12065-020-00540-3> (visited on 11/04/2021).
- [57] Connor Shorten, Taghi M. Khoshgoftaar, and Borko Furht. "Text Data Augmentation for Deep Learning". In: *Journal of Big Data* 8.1 (July 2021), p. 101. ISSN: 2196-1115. DOI: 10.1186/s40537-021-00492-0. URL: <https://doi.org/10.1186/s40537-021-00492-0> (visited on 11/04/2021).
- [58] "Orbit Analytics". "Tabular Data". URL: "<https://www.orbitanalytics.com/tabular-data/>".
- [59] "Techopedia Inc.". "Tabular Database". URL: "<https://www.techopedia.com/definition/26181/tabular-database>". "(accessed: 06.12.2018)".
- [60] Suki Lau. *Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning*. URL: "<https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1>". "(accessed: 29.07.2017)".
- [61] derivative work: Notjim. *Diagram of a en:neuron*. URL: "https://commons.wikimedia.org/wiki/File:Neuron_-_annotated.svg". "(accessed: 20.09.2008)".
- [62] Fathy Rashad. *Adversarial Auto Encoder (AAE)*. URL: "<https://medium.com/vitrox-publication/adversarial-auto-encoder-aae-a3fc86f71758>". "(accessed: 29.12.2020)".