

**Cyclic Adversarial Framework with Implicit Autoencoder and Wasserstein Loss
(CAFIAWL)**

by,

EHSAN MOBARAKI

Submitted to the Graduate School of Engineering and Natural Sciences

In Partial Fulfillment of the Requirements for the Degree of

M.Sc

in

Computer Science and Engineering

Sabanci University

Summer 2020

**Cyclic Adversarial Framework with Implicit Autoencoder and Wasserstein Loss
(CAFIWL)**

APPROVED BY:

Assoc. Prof. Dr. Kemal Kilic
(Thesis Supervisor)



Prof. Dr. Berrin Yanikoglu



Asst. Prof. Dr. Alper Ozpinar



DATE OF APPROVAL: 07/09/2020

ABSTRACT

Cyclic Adversarial Framework with Implicit Autoencoder and Wasserstein Loss (CAFIAWL)

Ehsan Mobaraki

M.Sc in Computer Science and Engineering

Thesis Supervisor: Dr. Kemal Kılıç

Keywords: Auto-Encoder, GAN, Bi-GAN, Wasserstein loss, Cycle-GAN, VGH, Mode Collapse

Since the day that the Simple Perceptron was invented, Artificial Neural Networks (ANNs) attracted many researchers. Technological improvements in computers and the internet paved the way for unseen computational power and an immense amount of data that boosted the interest (therefore the advance), particularly in the last decade. As of today, NNs seem to take a vital role in all different types of machine learning research and the main engine of many applications. Not only learning from the data with machines in order to make informed decisions but also “creating” something new, unseen, novel with machines is also a very appealing area of research. The generative models are among the most promising models that can address this goal and eventually lead to “computational creativity”. Recently the Variational Autoencoders (VAE) and the Generative Adversarial Networks (GAN) have shown tremendous success in terms of their generative performance. However, the conventional forms of VAEs had problems in terms of the quality of the outputs and GANs suffered hard from a problem that limited the diversity of the generated outputs, i.e., the mode collapse problem. One line of research that targets to eliminate these weaknesses of both algorithms is developing hybrid models which capture the strengths of these algorithms but avoiding their weaknesses. In this research, we propose a novel generative model. The proposed model is composed of four adversarial networks. Two of the adversarial networks are very similar to conventional GANs and the remaining two are basically WGAN that is based on the Wasserstein loss function. The way that these adversarial networks are put together also incorporates two implicit autoencoders to the proposed model and provides a cyclic framework that addresses the mode collapse problem. The performance of the proposed model is evaluated in various aspects by using the MNIST data. The analysis suggests that the proposed model generates good quality output meanwhile avoids the mode collapse problem.

ÖZET

Örtük Otokodlayıcı ve Wasserstein Kaybı İçeren Döngüsel Çekişmeli Çerçeve (CAFIAWL)

Ehsan Mobaraki

Bilgisayar Bilimi ve mühendisliği Lisansüstü Programı

Tez danışmanı: Dr. Kemal Kılıç

Anahtar Sözcükler: Otomatik Kodlayıcılar, GAN, Çift Yönlü, Wasserstein Ölçütü, Cycle-GAN, VGH, Mod Çöküşü Sorunu

Perceptronun icadından bu yana Yapay Sinir Ağları (NN) birçok araştırmacının ilgisini çekmektedir. Teknolojik gelişmeler özellikle son on yılda ilgiyi (dolayısıyla ilerlemeyi) daha da hızlandırmış olan yüksek hesaplama gücünün ve muazzam miktarda verinin önünü açtı. Bugün itibarıyla NN'ler, farklı makine öğrenmesi uygulamasının ana motoru olarak merkezi bir rol oynamaktadır. Makine öğrenmesiyle bilgiye dayalı karar vermek değil, aynı zamanda daha önce hiç görülmemiş, yeni bir şey “yaratmak” da çok çekici bir araştırma alanıdır. Üretken (Generative) modeller, bu hedefe hitap edebilecek ve sonunda "hesaplamalı yaratıcılığa" yol açabilecek en umut verici modeller arasındadır. Son zamanlarda Varyasyonel Otomatik Kodlayıcılar (VAE) ve Generative Adversarial Networks (GAN), yüksek performanslarıyla ilgi çekmektedirler. Bununla birlikte, VAE'lerin geleneksel biçimleri, çıktılarının kalitesi açısından sorunlar yaşarken, GAN'lar üretilen çıktılarının çeşitliliğini sınırlayan bir sorun, yani mod çöküşü sorunu, açısından sıkıntılar yaşamaktadır. Her iki algoritmanın bu zayıf yönlerini ortadan kaldırmayı hedefleyen bir bakış açısı, bu algoritmaların güçlü yönlerini alırken zayıflıklarından kaçınacak hibrit modeller geliştirmektir. Bu çalışmada, yeni bir üretken model öneriyoruz. Önerilen model, dört adet çekişmeli (adversarial) ağıdan oluşmaktadır. Çekişmeli ağlardan ikisi geleneksel GANs'a çok benzerken, diğer ikisi temelinde Wasserstein ölçütünün hesaplanmasının yer aldığı WGAN'dir. Bu dört adet çekişmeli ağın bir araya getirilme şekli, önerilen modele iki tane de örtük otomatik kodlayıcıyı dâhil ederek mod çöküşü sorununu da en aza indirecek şekilde döngüsel bir çerçeve sağlamaktadır. Önerilen modelin performansı MNIST verileri kullanılarak çeşitli yönlerden değerlendirilmiştir. Analiz, önerilen modelin iyi kalitede çıktı ürettiğini ve bu arada mod çöküşü sorununu ortadan kaldırdığını göstermektedir.

Acknowledgments

I would rather appreciate my supervisor Dr. Kemal Kilic whose encouragements and advice were the key points in the completion of this thesis. We faced many problems over the time of doing this job, many ups and downs. It goes without saying that his role as a leader was determinative for me, thank you, professor.

I also want to thank Prof. Berrin Yanikoglu and Dr. Alper Ozpinar to be part of the thesis jury and impress me with their feedbacks.

Finally, my warmest gratitude goes for my father Hassan Mobaraki, my mother Asmar Nourani, and my sister Samane Mobaraki for their kindest favor, nice, and encouragements during this research work.

Thank you

E.

© Ehsan Mobaraki 2020

All Rights Reserved

TABLE OF CONTENTS

1. INTRODUCTION	1
2. BACKGROUND.....	5
2.1 TYPES OF MACHINE LEARNING	5
2.1.1 Supervised Learning	5
2.1.2 Unsupervised Learning	5
2.1.3 Reinforcement Learning	6
2.1.4 Semi-Supervised and Self-Supervised Learning	7
2.2 OVERFITTING, GENERALIZATION, AND UNDERFITTING	7
2.3 ENSEMBLE LEARNING TECHNIQUES	8
2.3.1 Bootstrap Aggregation	8
2.3.2 Boosting	10
2.3.3 Stacking.....	11
2.3.4 Multi-Layered Stacking	12
2.4 NEURAL NETWORKS	13
2.4.1 Activation Functions	15
2.5 MAIN TOPOLOGIES FOR NEURAL NETWORKS.....	16
2.5.1 Convolutional Neural Networks	16
2.5.2 Recurrent Neural Network.....	17
2.5.3 Long Short-Term Memory.....	18
2.5.4 Residual Neural Networks	21
3. RELATED LITERATURE	23
3.1 AUTO ENCODERS	23
3.1.1 Under Complete Auto-Encoders.....	24
3.1.2 Complete Auto-Encoders.....	25
3.1.3 Over Complete Auto-Encoders.....	26

3.1.4	Regularized Auto Encoders	27
3.1.5	Variational Auto-Encoders	32
3.2	GENERATIVE ADVERSARIAL MODELS	33
3.2.1	Adversarial Auto-Encoder – AAE	36
3.2.2	Bidirectional Generative Adversarial Networks	37
3.2.3	The main drawback of Bi-GANs	39
3.2.4	Boosting Generative Adversarial Networks	40
3.2.5	Cycle-GANs.....	40
3.2.6	Mode Collapse and Missing Mode	41
3.2.7	Variational Encoder Enhancement to Generative Adversarial Networks.....	42
3.2.8	Wasserstein Generative Adversarial Networks	45
3.2.9	Variational Auto-Encoder Generative Adversarial Networks VAE-GAN or VGH/VGH++	48
4.	PROPOSED MODEL	51
4.1	MULTIPLE CHANCE ENHANCEMENT TECHNIQUE (MCET).....	54
5.	EXPERIMENTAL ANALYSIS AND DISCUSSIONS	56
6.	CONCLUDING REMARKS AND FUTURE WORK.....	62
	BIBLIOGRAPHY	64

LIST OF TABLES

TABLE 1: ACTIVATION FUNCTION (TABLE IS FROM [26]).....	16
TABLE 2: FRÉCHET INCEPTION DISTANCE SCORES	61

LIST OF FIGURES

FIGURE 1: OODA-LOOP INTRODUCED BY JOHN BOYD (FIGURE IS ADAPTED FROM [1])	1
FIGURE 2: BOOTSTRAP AGGREGATION (FIGURE IS ADAPTED FROM [9])	9
FIGURE 3: BOOSTING – SEQUENTIAL ENSEMBLE LEARNING (FIGURE IS TAKEN FROM [14])	11
FIGURE 4: STACKING ENSEMBLE LEARNING (FIGURE IS TAKEN FROM [23])	12
FIGURE 5: MULTI-LAYERED STACKING (FIGURE IS TAKEN FROM [23]).....	13
FIGURE 6: SINGLE LAYER PERCEPTRON (SLP) (FIGURE IS TAKEN FROM [24]).....	15
FIGURE 7: MULTI-LAYER PERCEPTRON (MLP) (FIGURE IS TAKEN FROM [25]).....	15
FIGURE 8: CNN vs FCNN (FIGURE IS TAKEN FROM [28]), COMPREHENSIVE GUIDE TO CNNs (FIGURE IS TAKEN FROM [29])	17
FIGURE 9: RNN vs FFNN (FIGURE IS TAKEN FROM [31]).....	18
FIGURE 10: LONG SHORT-TERM MEMORY (FIGURE IS TAKEN FROM [33]).....	19
FIGURE 11: GATED RECURRENT UNIT (FIGURE IS TAKEN FROM [34]).....	20
FIGURE 12: RNN vs LSTM vs GRU (FIGURE IS TAKEN FROM [35])	20
FIGURE 13: RESIDUAL VS PLAIN NETS (FIGURE IS TAKEN FROM [37]).....	22
FIGURE 14: THE GENERAL STRUCTURE OF AUTO ENCODERS (AE) (FIGURE IS TAKEN FROM [39]).	24
FIGURE 15: UNDERCOMPLETE AUTO-ENCODER (UCAE) (FIGURE IS TAKEN FROM [46]).....	25
FIGURE 16: COMPLETE AUTO-ENCODERS (FIGURE IS TAKEN FROM [46])	26
FIGURE 17: OVER COMPLETE AUTO-ENCODERS (FIGURE IS TAKEN FROM [47])	26
FIGURE 18: SPARSE AUTO-ENCODER (FIGURE IS TAKEN FROM [46]).....	29
FIGURE 19: DENOISING AUTO-ENCODER (FIGURE IS TAKEN FROM [50])	30
FIGURE 20: STACKED AUTO-ENCODER (FIGURE IS TAKEN FROM [52])	31
FIGURE 21 VARIATIONAL AUTO-ENCODER (FIGURE IS ADAPTED FROM [54]).....	32
FIGURE 22: GENERATIVE ADVERSARIAL NETWORKS (GAN) (FIGURE IS ADAPTED FROM [57]).....	34
FIGURE 23: GENERATIVE ADVERSARIAL NETWORK (FIGURE IS TAKEN FROM [58]).....	35
FIGURE 24: ADVERSARIAL AUTO-ENCODER(FIGURE IS TAKEN FROM [59]).....	36

FIGURE 25: SCHEMATIC VIEW OF AN ADVERSARIAL AUTO-ENCODER (FIGURE IS ADAPTED FROM [60])	37
FIGURE 26: BIDIRECTIONAL GENERATIVE ADVERSARIAL NETWORKS (FIGURE IS ADAPTED FROM [61])	38
FIGURE 27: VARIATIONAL ENCODER ENHANCEMENT GENERATIVE ADVERSARIAL NETWORKS (FIGURE IS ADAPTED FROM [67])	43
FIGURE 28: HOW THE RECONSTRUCTOR NETWORK IN VEEGANs BEHAVES IN SCENARIO 1 (FIGURE IS TAKEN FROM [67])	44
FIGURE 29: HOW THE RECONSTRUCTOR NETWORK IN VEE-GAN BEHAVES IN SCENARIO 2 (FIGURE IS TAKEN FROM [67])	45
FIGURE 30: VARIATIONAL AUTO-ENCODER GENERATIVE ADVERSARIAL NETWORK – VAEGAN AND VGH/++ (FIGURE IS ADAPTED FROM [3]).....	49
FIGURE 31: OUR PROPOSED MODEL	54
FIGURE 32: GENERATED DATA FOR THE THREE SCENARIOS IN THE FIRST ANALYSIS	58
FIGURE 33: LEFT PLOT WHERE MCET IS NOT APPLIED, THE RIGHT PLOT IS WHERE WE HAVE APPLIED MCET.....	59
FIGURE 34: RESPECTIVELY, FROM THE LEFT A COLLECTION OF GENERATED SAMPLES BY OUR PROPOSED MODEL, VGH++, WGAN, AND VAE (THE LAST THREE FIGURES ARE TAKEN FROM [3])	60

LIST OF ABBREVIATIONS

AAE	Adversarial Auto-Encoder
AE	Auto-Encoder
AI	Artificial Intelligence
BAgging	Bootstrap Aggregation
Bi-GAN	Bidirectional Generative Adversarial Networks
CAE	Contractive Auto-Encoder
CE	Cross Entropy
CNN	Convolutional Neural Networks
CoAE	Complete Auto-Encoder
DAE	Denoising Auto-Encoder
DM	Data Mining
ELT	Ensemble Learning Technique
FcNN	Fully Connected Neural Networks
FCNN	Fully Convolutional Neural Networks
FID	Fréchet Inception Distance Scores
GAN	Generative Adversarial Networks
GRU	Gated Recurrent Unit
IoT	Internet of Things
LSTM	Long Short-Term Memory
MCET	Multiple Chance Enhancement Technique
ML	Machine Learning
MLP	Multi Layered Perceptron
MNIST	Modified National Institute of Standards and Technology
MSE	Mean Square Error
NN	Artificial Neural Networks

OCAE	Over Complete Auto-Encoder
OODA	Observe, Orient, Decide, Act
RAE	Regularized Auto-Encoder
ResNet	Residual Networks
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SAE	Sparse Auto-Encoder
SLP	Single Layer Perceptron
StAE	Stacked Auto-Encoder
UCAE	Under Complete Auto-Encoder
VAE	Variational Auto-Encoder
VAEGAN	Variational Auto-Encoder Generative Adversarial Networks
VEEGAN	Variational Encoder Enhancement Generative Adversarial Networks
VGH	<u>V</u> ariational Auto-Encoder <u>G</u> enerative Adversarial Networks <u>H</u> ybrid
WGAN	Wasserstein Generative Adversarial Networks

1. INTRODUCTION

OODA-Loop which is introduced by Boyd [1] models the human decision-making process. According to this model, the process has four stages namely, *Observe* (O), *Orient* (O), *Decide* (D), and *Act* (A). Every moment we *observe* (i.e., receive data) by means of our five senses (i.e., hear, see, touch, smell, taste). The data passes through our neural system as an electrical signal and enters to our connectome (i.e., brain) where due to the synaptic gaps between the neurons the communication is done by means of electrochemical signals. In our connectome, we give meaning to the data that was sensed which corresponds to the *orientation* stage in the OODA-Loop. The orientation stage provides the necessary input for the third stage, namely *decide*. After the decision is made, it is sent to our motor system so that we start to *act* (e.g., move, talk, walk, etc.).

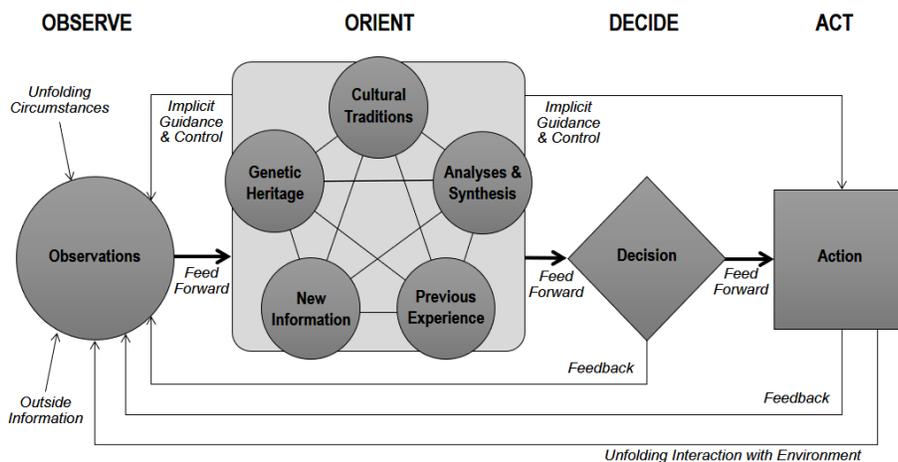


Figure 1: OODA-Loop introduced by John Boyd (Figure is adapted from [1])

Digital transformation in a nutshell aims to take the human out of the OODA-Loop. In the ecosystem that will be created by digital transformation, the sensors will gather data, via the internet the data will be transferred to the cloud and/or edge systems where the patterns hidden inside the data will be identified. That is to say, meaning will be associated with otherwise *meaningless* data. Next by utilizing various decision-making techniques such as optimization,

simulation, etc. the decisions will be made by intelligent agents and the action would be carried out by smart robots, chatbots, cyber-physical systems, etc. Since the orientation and decision stages are basically what we consider as “intelligence” in humans, in the context of digital transformation, the activities regarding giving meaning to data and decision making correspond to what we call Artificial Intelligence (AI).

AI has a central role in the process and is positioned at the core of digital transformation. The orientation is usually addressed by descriptive data analytic techniques (such as data visualization, descriptive statistics, etc.) and by predictive data analytic techniques (such as predictive statistics, machine learning, etc.). Thus, as states, organizations, companies, researchers increased their attention towards digital transformation activities, the attention towards machine learning also rocketed.

Research on neither AI nor ML is new. Both concepts were introduced in the era of the post-second world war. After the infamous question posed by Alan Turing, i.e., “Can Machines Think”, researchers started working to develop machines that could think like a human. That was also the era when neuroscience was flourishing, and the secrets of the human brain were being unveiled. In 1959, based on the postulate of Donald Hebb which was addressing the learning mechanism in the human brain (mostly known as Hebbian Learning), Rosenblatt [2] introduced the *perceptron*, i.e., the first algorithm which was mimicking the human neural networks. Subsequently, the research on *artificial neural networks* (NN) started to gain momentum.

For several decades (with ups-and-downs) the NN received a lot of attention. However, the pace was lost due to various weaknesses in particular the need for a lot of data, huge computational requirements to process the data and determine a virtually endless number of parameters during the training phase. Technological advances of computers and sensors as well as the wide availability of the internet kind of resolved these weaknesses and another wave of attention started during the last decade. Thanks to the Internet of Things (IoT) devices the immense amount of data was being gathered with subtle of no cost and various technologies such as cloud computing, parallel processing, etc. provided the computational power required to train the NNs.

Besides the availability of sufficient computational power and data, the introduction of new algorithms also became an impulsive force to the realm of NN research. One such example was the introduction of the Generative Adversarial Networks. As the name implies GANs is a generative model that is trained based on an adversarial game between two of its component networks, namely the generator and the discriminator.

Even though other generative models outdate GANs, the potential of the algorithm created much attention from the research and practitioner’s community. In an interview, Chief AI scientist of Facebook Yann LeCun states that the “Generative Adversarial Networks (GANs) is the most interesting idea for the last decade in machine learning”. What makes generative models very appealing for the researchers is crossing the borderline between *learning* something that exists and

creating something that does not exist. GANs in that sense is a step towards the machines that create based on their learnings.

GANs learn from a huge set of data that supposedly represents the population well. The data usually consists of many modes (i.e., subgroups with similar characteristics). What is expected from a good generative model is also generating (i.e., creating) new data which represents these modes in a balanced manner. That is to say, missing some of the modes and generating new data only from some modes are not desired and referred to as the missing mode problem. Sometimes the problem with the generator is so severed, it starts generating only from one of the modes that are available in the whole data set. This problem is referred to as the mode collapse problem.

A major problem with GANs is soon to be realized as its weakness to find a solution to the mode collapse problem. A solution for the mode collapse problem of the GANs that is suggested in the literature is developing hybrid architectures that borrow the strength of each architecture and overcome the challenges that are faced. Recently Rosca et al. [3] proposed a hybrid architecture where GANs and Variational Auto Encoders (VAE) are combined.

Auto-Encoders (AE) are also a NN architecture that consists of three components namely the encoder, decoder, and the code layer which is basically the interface between the encoder and decoder networks. As the name implies, the overall goal of AEs is having an output that is very close to the input data (thus the name has *auto*). In order to avoid the learning of the trivial identity function, various regularization techniques are proposed in the literature. One such regularization was based on the restriction of the code layer to represent the parameters vector of a multidimensional Gaussian distribution. This regularization was referred to as the VAEs. Note that, since the code itself was learned from the training data, later it could be used to start *generating* totally new data as well. Thus, VAEs can also be used as generators. Unlike the GANs the weakness is not the mode collapse but the quality of the output.

The framework proposed by [3], mainly targets to solve the mode collapse of GANs by augmenting a VAE to the architecture. In this thesis, we are inspired by [3] and proposed a novel architecture. Unlike the framework of [3], the proposed architecture doesn't have an *explicit* AE but due to its structure (i.e., how the adversarial networks are put together) has *two implicit* AEs. Various other differences such as the utilization of four adversarial networks and separating the code discriminator from the data discriminator, the introduction of Wasserstein GANs rather than the simple $\mathcal{L}_1 - norm$ used by [3], and the novel Multiple Change Enhancement Technique (MCET) used in order to boost the training process are some other modifications introduced in the proposed model. **The proposed model is the main contribution of the thesis.**

The structure of the thesis is as follows. In Chapter 2 we are going to provide basic information regarding the background of machine learning that is needed to be understood to follow some of the further discussions. We will discuss the types of machine learning paradigms such as supervised/unsupervised/reinforcement learning and introduce some of the concepts such as

overfitting, generalization as well. Basic knowledge related to NNs (e.g., historical evolution, various activation functions, commonly used architectures will also be discussed in this chapter. Next in Chapter 3, the details of AEs and GANs will be provided. Strengths and weaknesses of both algorithms, various solutions that address these problems as well as recent research on the hybrid frameworks will also be presented in Chapter 3.

These discussions will help the readers to understand the motivation behind the proposed methodology which will be presented in Chapter 4. The experimental analysis, the results, and discussions are available in Chapter 5. The experimental analysis in the thesis is in two folds. First, more subjective and anecdotal analysis is conducted to demonstrate the performance of the proposed architecture. Secondly, the performance of the proposed method is compared with various other techniques available in the literature in terms of the Fréchet Inception Distance Scores (FID). In both analyses, we have utilized the MNIST dataset. The thesis will be finalized with our concluding remarks and suggested future research topics in Chapter 6.

2. BACKGROUND

In this chapter, we will briefly summarize various concepts in the context of machine learning. We will start by providing a rough taxonomy of machine learning approaches. Next, we will elaborate on the generalization of the training models and discuss briefly over- and under-fitting. The focus of our thesis is on the realm of neural networks (NN). Thus, we will spend more time on NN and provide some background information including various activation functions and finalize the chapter with a section where some NN topologies are covered.

2.1 Types of Machine Learning

Machine Learning (ML) plays a significant role in AI. Generally, it is considered that there are three main types of ML which are: supervised learning, unsupervised, and reinforcement learning. We can consider other types of ML such as semi-supervised and self-supervised learning as well, however, they are combinations of the main three types of ML that are mentioned above.

2.1.1 Supervised Learning

Most of the ML problems and models belong to this group. What distinguishes the supervised learning problems from the others is the availability of the output. These output(s) can be numeric which have values in ratio or interval scale or classes which have values in ordinal or nominal scales. The former case usually is referred to as regression problems and the latter one is referred to as the classification problems. The output acts as a supervisor, like someone who watches everything to inform the model which data point (i.e., an input vector with various features) is related to which type of outputs. That is why this type of problem is referred to as the Supervised Learning problems.

2.1.2 Unsupervised Learning

The second most common type of ML is unsupervised learning. In unsupervised learning problems, the data does not have an output. Thus, the supervision of the output is out of the

question. On the other, the data still have patterns that might be hidden in the input feature space and unsupervised learning algorithms tries to bring that hidden patterns to the surface.

A major type of problem in the realm of unsupervised learning is the clustering problem. Clustering basically tries to determine the “*natural*” groupings in the data based on the attributes (i.e., the features) associated with each sample. Depending on how one defines the “*natural*” groupings, there are various unsupervised learning approaches, such as distance-based (e.g., K-Means Clustering and variants, Hierarchical Clustering, etc.), model-based (Expectation-Maximization, etc.), connectivity/density-based (e.g., DBSCAN, CLIQUE, etc.). Even though it is also possible to end up with soft clusters by using fuzzy clustering algorithms (e.g., FCM, PCM, etc. which assigns a membership degree to the data from the interval $[0,1]$), usually in practice hard clustering techniques are used which assigns a particular sample top a cluster or not (i.e., membership is from set $\{0,1\}$).

The distance-based clustering algorithms, instead of determining a class which a given sample belongs to, try to group the samples based on distance function - similarity/proximity or in some cases dissimilarity-. As a result, finding out all similar samples based on the given attributes is the task which unsupervised learning is responsible for. One of the most well-known and frequently used unsupervised learning algorithms is K-means Clustering. In the K-Means Clustering algorithm, the distance between a sample and the centroid (i.e., the mean) of a cluster determines its similarity to that cluster. Thereafter, all clusters are determined by executing two steps of K-Means (reassign and recalculate the centroid) iteratively.

2.1.3 Reinforcement Learning

The third type of ML is Reinforcement Learning (RL). RL is the problem faced by an agent that must learn behavior through trial-and-error interactions in a dynamic environment [4]. As opposed to supervised learning, there are no input-output pairs in the RL paradigm. There are agents, which are in a state in the dynamic environment, as they take actions, they immediately receive payoffs so based on those payoffs the agent adopts its behavior.

In RL, the learner is not assumed to take any action, instead it is supposed to find out which action to take in order to maximize the reward. Despite the supervised learning algorithms and unsupervised learning algorithms, RL is supposed to react to a dynamic environment. Therefore, the goal of RL algorithms is to maximize reward in its interaction with the environment. Omelet swapping machine is a well-known example of reinforcement learning algorithms.

2.1.4 Semi-Supervised and Self-Supervised Learning

The three types of ML models explained formerly (Supervised Learning, Unsupervised Learning, and Reinforcement Learning) are the main pieces of training frames; nevertheless, we have other types as well.

One of the secondary types of ML models is called Semi-Supervised Learning [5]. Sometimes to fasten convergence of an unsupervised hypothesis, we add few labeled data into the input set to feed the algorithm. Practically the reasons behind it are data shortage, regularizations on parameters, or some other undesired cases in ML problems to be handled appropriately and consequently to avoid Overfitting and Underfitting. So, the algorithm is trained on partially supervised and unsupervised data, but the point behind this partial combination is that the quantity of labeled data is much less than the quantity of unlabeled data. This type of data arrangement for a model training is known as semi-supervised learning.

Another secondary ML frame is called Self-Supervised Learning. Self-Supervised Learning is a form of Unsupervised Learning where data provides its own supervision. Generally, by taking some part of data, Self-Supervised Learning enforces the algorithm to predict it. One of the reasons for introducing this form of learning is that labeling large datasets for Supervised Learning practically is not feasible, however, training Self-Supervised Learning sounds more practical. Thus, Self-Supervised Learning enables training without explicit supervision which is the case for both the supervised learning and the semi-supervised learning. Zhai et al. [6], demonstrate the efficiency and effectiveness of both Self-Supervised Learning and Semi-Supervised Learning in their research.

2.2 Overfitting, Generalization, and Underfitting

Generally, the training phase of a model can yield three possible situations: underfitting, generalization, and overfitting. When a model is trained too accurate on a training dataset, in other words, it perfectly matches what it should match, even though it might have extremely small training error when we test our model with an unseen dataset (i.e., the test data), it might yield an unacceptable error. This situation is what is called overfitting. Unlike the training error, it is the test error that would give us a good estimate of the performance of the model to predict the population parameters. Thus, overfitting is undesirable and should be avoided during the training phase.

On the other extreme, we have underfitting which as its name indicates, it is the situation that occurs when the model is not trained enough on the dataset. The model cannot extract as good as it should the latent structure from the training dataset, so, cannot make accurate predictions of the population parameters as well.

The third possibility is the most desired situation for most of ML models, where we have neither underfitting nor overfitting. In this case, our model neither misses the important latent structure in the training data samples nor memorizes a specific features map and it would lead to an acceptable test error, compared to underfitting and overfitting. In fact, the generalization of a model is a major goal for ML projects.

The main goal in any classification problem is to achieve the configuration which yields a generalized hypothesis (neither Overfitting nor Underfitting). i.e., the desired hypothesis avoids Overfitting and Underfitting.

A popular approach that is used to achieve this goal is to use the validation set. Practically, 70% of the data set is taken as a training set, and the algorithm runs for some iterations on it, 15% usually is taken as a validation set i.e., a dataset that is fed to the algorithm to prevent Underfitting and Overfitting. Thus, we analyze the performance of the learner throughout the validation set during the training phase to avoid Overfitting and Underfitting. The last 5% dataset remains as the test set, which we would use to check our model final accuracy rates.

In order to take a dataset into three training, validation, and testing, we have two main techniques i.e., bootstrap and secondly, k-fold. Bootstrap has replacement while the k-fold has no replacement while making any subset.

2.3 Ensemble Learning Techniques

Ensemble Learning Techniques (ELTs) are some meta-algorithms, that can have positive impacts on the performance of basic classifiers, and it is demonstrated that an ensemble is more accurate than any single classifier (learner) in the ensemble [7]. ELTs use multiple learning algorithms that could have better predictive performance compared to separately using basic learning algorithms (i.e., weak learners) [7]. In this section, we will cover the most frequently used three ELTs, namely, BAgging, Boosting, and Stacking as well as the Multi-Tier Stacking which is an extension to the stacking.

2.3.1 Bootstrap Aggregation

The word Bagging stands for Bootstrap Aggregation (BAGging). It uses the Bootstrap technique as a type of data segmentation which is introduced by Efron in 1979 [8]. Each generated subset is given into a base learner, to be trained on.

Replacement of the bootstrap technique enables a sample to be selected again, while the replacement is not permitted in the K-fold cross-validation technique. Thus, Bootstrap is widely perceived by the scientific community as a sound way to determine the empirical distribution that

represents the population distribution. So, each sample is fed as input to the base learners to be trained on, consequently, accuracy (or any other performance metric) based on predictions per base learner is calculated. The next phase is called the aggregation phase where for all samples which have been passed through any base learner, a voting process determines its final class of membership in a classification problem (and averaging in a regression). Figure 2 depicts the BAGging meta-algorithm.

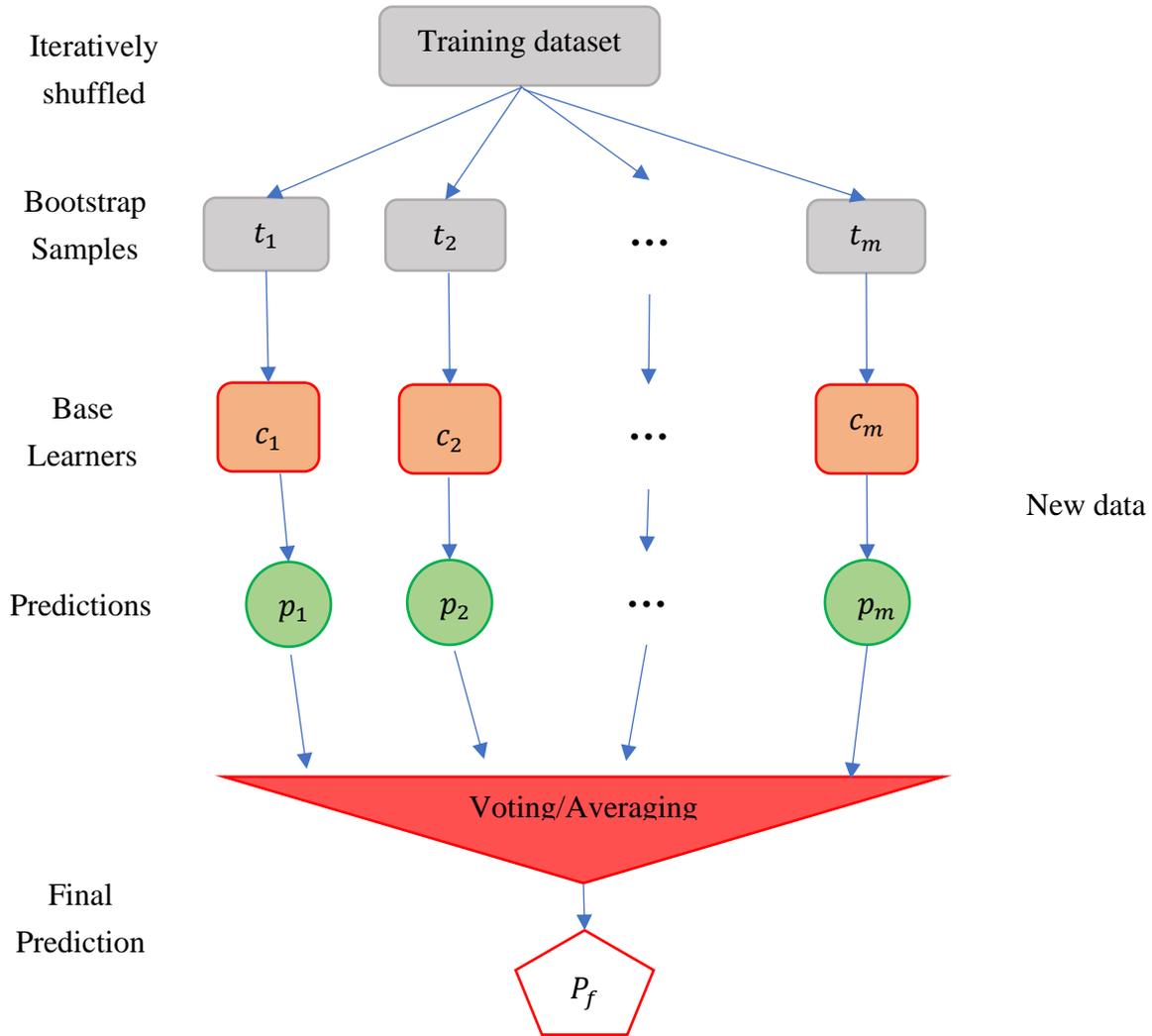


Figure 2: Bootstrap Aggregation (Figure is Adapted from [9])

The BAGging technique could be used in regression i.e., we can use the average of all base learners' outputs and make decisions for the super classifier based on the average calculated on predictions of the basic classifiers. Thus, we have the Equation (1) for the aggregation part to make the final decision.

$$F(x) = \frac{1}{M} \sum_{m=1}^M f_m(x) \quad (1)$$

The M on the aggregation formula stands for the number of the base learners. In the case of classification problem voting is used in the aggregation process to finalize the prediction of the ensemble. Figure 2 depicts the aggregation stage of the BAGging with the node voting/average.

Obviously, the idea behind this meta-algorithm is a generalization of the base hypotheses (i.e., base classifiers). Since the classification is not only dependent on one classifier, but it is based on multiple models, the result would be more generalized compared to a single base classifier. Roughly speaking, utilizing multiple samples during any analysis is a common approach to reduce the variance (and referred to as variance reduction techniques in general). Consequently, the output prediction is less susceptible to either overfitting or underfitting. As a result, it is expected that Bagging to be robust from the generalization viewpoint compared to a single base learner.

2.3.2 Boosting

Boosting is another meta-algorithm which supposed to turn weak base classifiers to strong learners [10], [11], and it is primarily for reducing bias and variance [12]. It focuses on incorrect predictions of the base learners and enforces them to fix their incorrect predictions. Boosting performs it via weighting each sample, such that when an incorrect label is predicted for a sample, the sample would have higher weights compared to those which have been matched to correct labels [12].

In boosting misclassification would increase weights and correct classification would decrease the weights [12]. An important point is that weighting is a hypothetical approach in order to converge base learners as much as possible which it leads to a sequential way of learning per sample, and that's why in some resources Boosting technique is considered as a sequential ensemble learning technique, and the others like Bagging are called committee methods [13].

Note that if a sample is misclassified its weight would increase for the next iteration since an error rate factor would be multiplied for the corresponding sample's weight per each misclassification. In a similar vein, we are distributing higher weights among misclassified samples until all become well-classified or any other threshold is achieved [12]. Figure 3 provides a snapshot of a Boosting technique in a more detailed way.

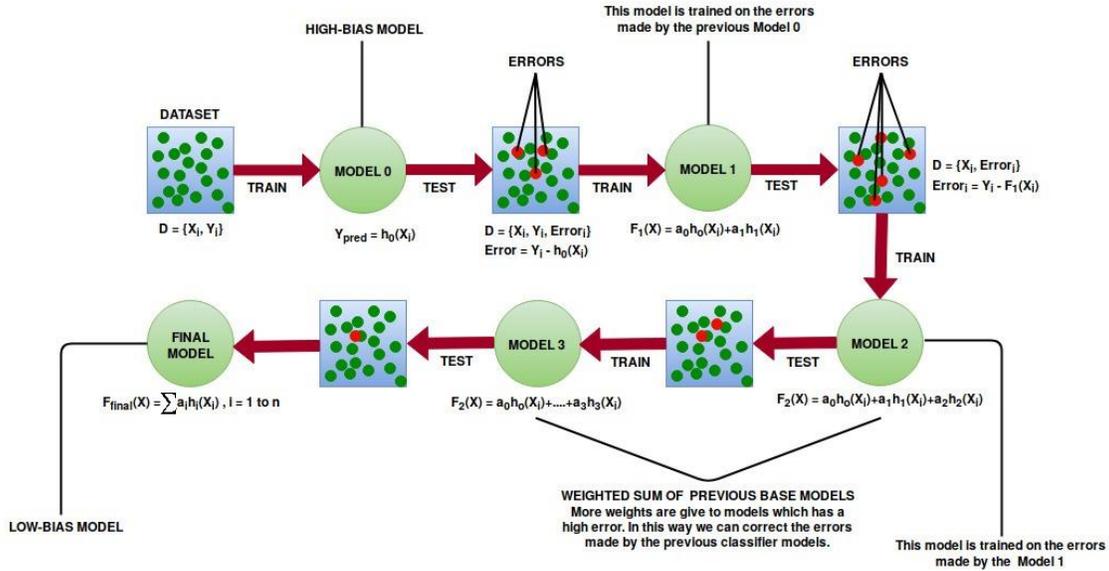


Figure 3: Boosting – Sequential Ensemble Learning (Figure is taken from [14])

Besides the Adaptive Boosting technique (AdaBoost [15]) as the commonly used example of Boosting techniques, we have other boosting based techniques like:

- Gradient Boosting Tree [16]
- GentleBoost [17]
- LPBoost [18]
- Brown Boost [15]
- XGBoost [19]
- CatBoost [20]
- LightGBM [21]

2.3.3 Stacking

Stacking as a form of ELTs, involves training of a learner algorithm to combine predictions of multiple base learning algorithms [22]. As the first stage of training a stacking algorithm, all base learning algorithms are trained on the desired dataset, then a combiner learning algorithm (higher level) is trained to make the final predictions based on the predictions of the base learners along with the dataset (i.e., jointly training). That is to say, the combiner learner deals with the desired dataset and predictions of base learners as additional inputs. The main difference of the Stacking with Boosting and BAGging is that in the case Stacking the combiner is also a learning algorithm, while the Boosting uses a statistical function to arrange weights for misclassifications and the BAGging uses voting/averaging to predict the final predictions. A research done to evaluate

performance of using the stacking meta training algorithm demonstrates that using stacking performs better than using any base learning algorithms [22].

The main contribution of this technique is its dependency on type of data. Such that, if a specific model performs well on a specific subset of data, the prime classifier would be trained to rely primarily on the specific base learner rather than any other basic learner when dealing with that subset of data. Figure 4 depicts general structure of a Stacking ELT on a network of basic learners.

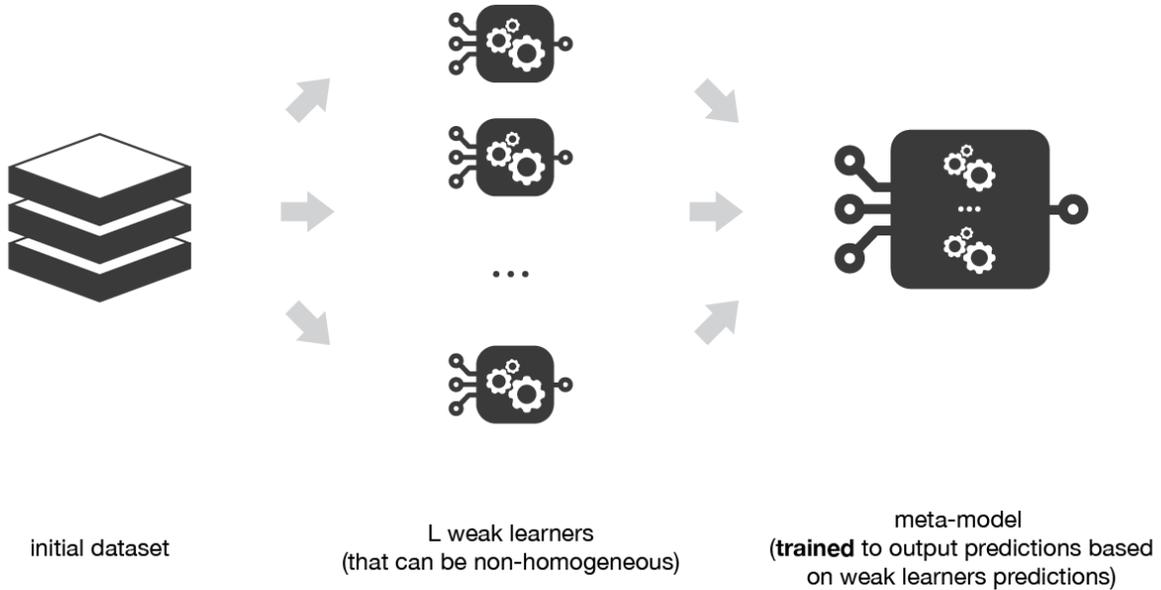


Figure 4: Stacking ensemble learning (Figure is taken from [23])

So, the last learner (meta-model) is trained on the performance result of base learners.

2.3.4 Multi-Layered Stacking

A creative extension for Stacking, as a type of ensemble learning techniques, is multi-layered Stacking. Accordingly, instead of one meta-model which supervises some base learners, we have multiple (at least two) meta-models which all together supervise the base learners.

Each meta-model in the second layer (the first layer consists of base learners) is connected to all base learners in the first layer and their inputs are the outputs of the base learners in the first layer. Then, all meta-models are connected to a single meta-model in the third layer, which in our example, the third layer is the final meta-classifier to make the final decision. Note that, we might have multiple layers of meta-layers. Figure 5 presents the scheme three leveled Stacking in detail.

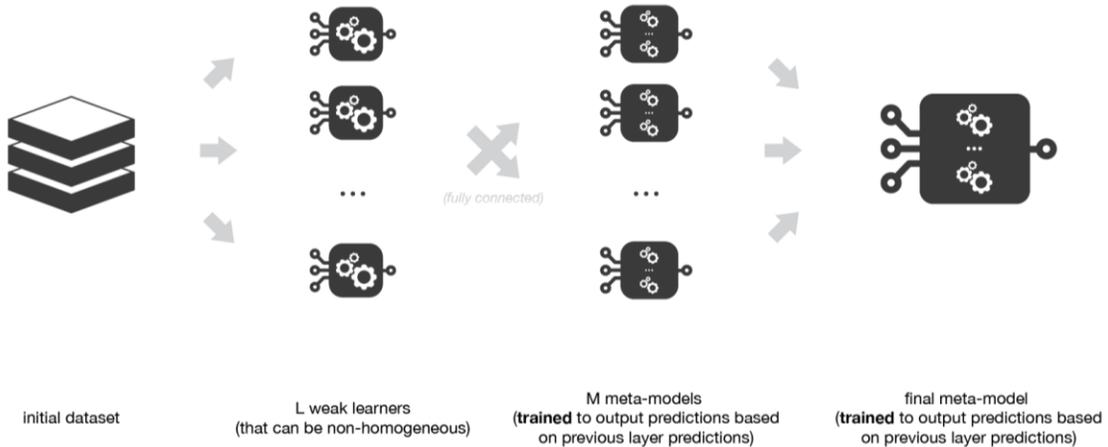


Figure 5: Multi-Layered Stacking (Figure is taken from [23])

As presented in Figure 5, predictions of L base learners are taken as inputs throughout M meta-models in the second layer, and the outputs of M meta-learners are given as inputs for the final meta-model in the third layer as the final classifier in our case.

2.4 Neural Networks

Even though Artificial Neural Networks (from now on which will be referred to as Neural Networks in short and denoted as NN) was available for many decades after WWII, due to the various limitations (such as the need for extensive data to train them, computational power, etc.) many other basic algorithms were introduced and preferred as an alternative in machine learning problems (e.g., supervised, unsupervised, and reinforcement learning). However, the recent technological developments made an immense amount of data as well as computing power cheap and reachable, a new tide of attention has developed towards NN.

Perceptron was the first NN algorithm that was invented by Rosenblatt [2]. It has been inspired by human brain neurology, in particular from the Hebbian learning paradigm. The simplest perceptron is known to be the Single Layer Perceptron (SLP) which was developed for supervised learning in particular for binary classification. Figure 6 depicts an SLP. In SLP the input data is fed into a “neuron” and binary output is obtained through an activation function. The activation function uses the weighted sum of the input vector and bias to determine the binary output. Later it was shown that SLP could not handle Boolean XOR operation thus research on NN was decreased for a while.

After the introduction of Multi-Layered Perceptron (MLP) which enabled the NN to handle nonlinear problems as well, NNs again popularized and started being used in complex problems. Each node in MLP is identical to a node in SLP, such that an input vector is summed up together in a net function and the output is given into an activation function (or sometimes referred to as

the transfer function) to calculate output value for the node. However, architecturally they have many differences, such that in MLP as clearly indicated we have at least one more layer of neurons compared to SLP which consists of a single perceptron.

Figure 7 depicts an MLP. As demonstrated in Figure 7, we can see that we may have many layers of weights, and that's why it's called Multi Layers perceptron. As a result, the singularity and multiplicity ensue from the number of layers of weights. When we talk about SLP, it means we would have a single layer of weights for input set cases, and when we talk about MLP, it means we have at least one hidden layer in our network.

Later, MLP was commonly used in many nonlinear ML problems, in fact, MLP resolved NN limits to be able to handle any spherical data. Schematically, the number of the neurons is the number of linear borders which the model sets on to make an appropriate classification. Obviously, as the number of lines increases, it may become accurate in training, however, there is no rule to get the best architecture for any problem, rather than try and check.

Generally, backpropagation plays a principal role in the training of NNs. Backpropagation is the practice of weight tuning in NNs, which is done based on the loss value (i.e., error) calculated in the last epoch. Normally, there is a consequential relation between weight tuning and error rate. As properly as weight tuning is performed, the lower error rate will come up and the learner is more generalized.

An error gradient is a direction and magnitude calculated during the training of a neural network and it is used to be backpropagated on the network parameters to update them based on the direction and value of it for the next iterations. In deep NNs with a large number of layers and nodes, the gradients might accumulate and build up huge values (larger than one). Thus, due to the backpropagation process the gradients grow even more and cause the training process to be unstable. The fast growth of gradients by repeatedly multiplying is known as gradient explosion.

Oppositely, the gradients might decrease as they propagate back to the first layers in a deep NN, as a result, the closer gradients become to the first layers, the lower they become and parameters of the network in the first layers do not change as the parameters in the last layers change. Therefore, gradients vanish during propagating back to the first layers. This problem is known as the vanishing gradient which is a reason for unstable training of a network.

One of the major concerns in the NN design process is deciding the topology as well as the architecture. According to the Encyclopedia of Machine Learning (Springer) "*Topology of a neural network refers to the way the neurons are connected, and it is an important factor in network functioning and learning. The most common topology in supervised learning is the fully connected, three-layer, feedforward network.*". In this case, the "fully connectedness" implies that each neuron is a layer that is connected to all the neurons in the next later. These type of NN are referred to as the Plain Nets. On the other hand, it's not enough just to decide the topology of the

neural networks, but one also has to decide various other things such as (*at least the initial*) number of nodes at each layer, etc. as part of the architecture of the NN.

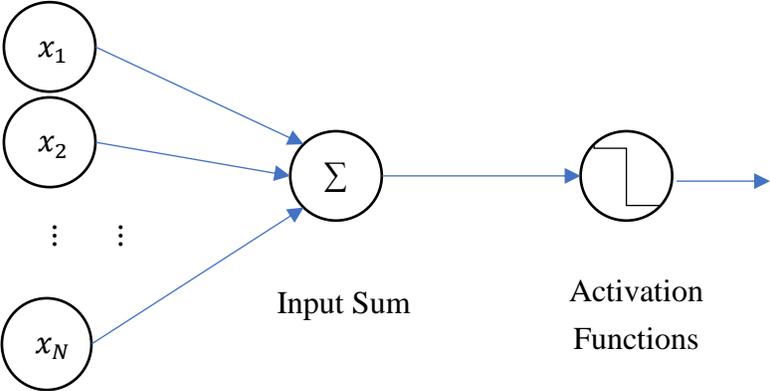


Figure 6: Single Layer Perceptron (SLP) (Figure is taken from [24])

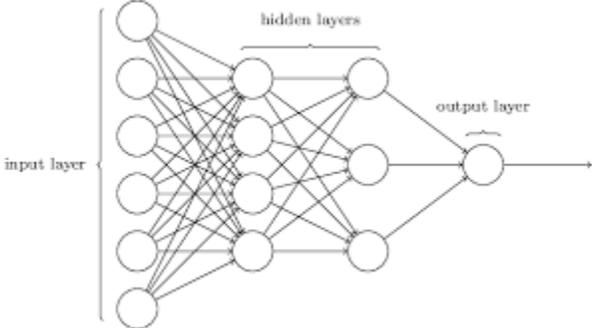
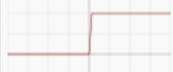


Figure 7: Multi-Layer Perceptron (MLP) (Figure is taken from [25])

2.4.1 Activation Functions

After applying the net function through our input set and weight set, the output is given to an activation function to generate the output value for the node. In the literature, there are various types of activation functions. Some of the popular activation functions are tabulated in Table 1.

Table 1: Activation Function (Table is from [26])

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

2.5 Main Topologies for Neural Networks

For many years NNs were used in their simple formats - fully connected -, and the only parameters which were used to make distinctions between them were architectural parameters like number of neurons, number of layers, and epoch value; until Convolutional Neural Networks and Recurrent Neural Networks carried NN into a new era. Despite the new topologies had small changes compared to basic NNs, both led valuable contributions.

2.5.1 Convolutional Neural Networks

The idea behind Convolutional Neural Networks (CNN) [27] was to reduce the number of parameters. Generally, the number of parameters in fully connected NNs is much higher than the number of parameters in CNNs. CNNs are locally connected networks, in other words, instead of

connectivity between any single neuron with all neurons in the next layers and/or previous layers, connections are local, not global. CNNs pass input images through series of convolutional layers with some filters. Convolution preserves the relations between pixels by learning images features through using small squares of input data.

CNN stands for local connectivity in a network, that is while if a network is called as Fully Convolutional Neural Network, it means even in the classification layers of the network we do not have any fully connected layer. i.e., FCNN is a network full of locally connected nodes and it lacks any fully connected neuron. Figure 8 shows CNN in a schematic picture compared to a fully connected NN and downsampling which happens in a CNN.

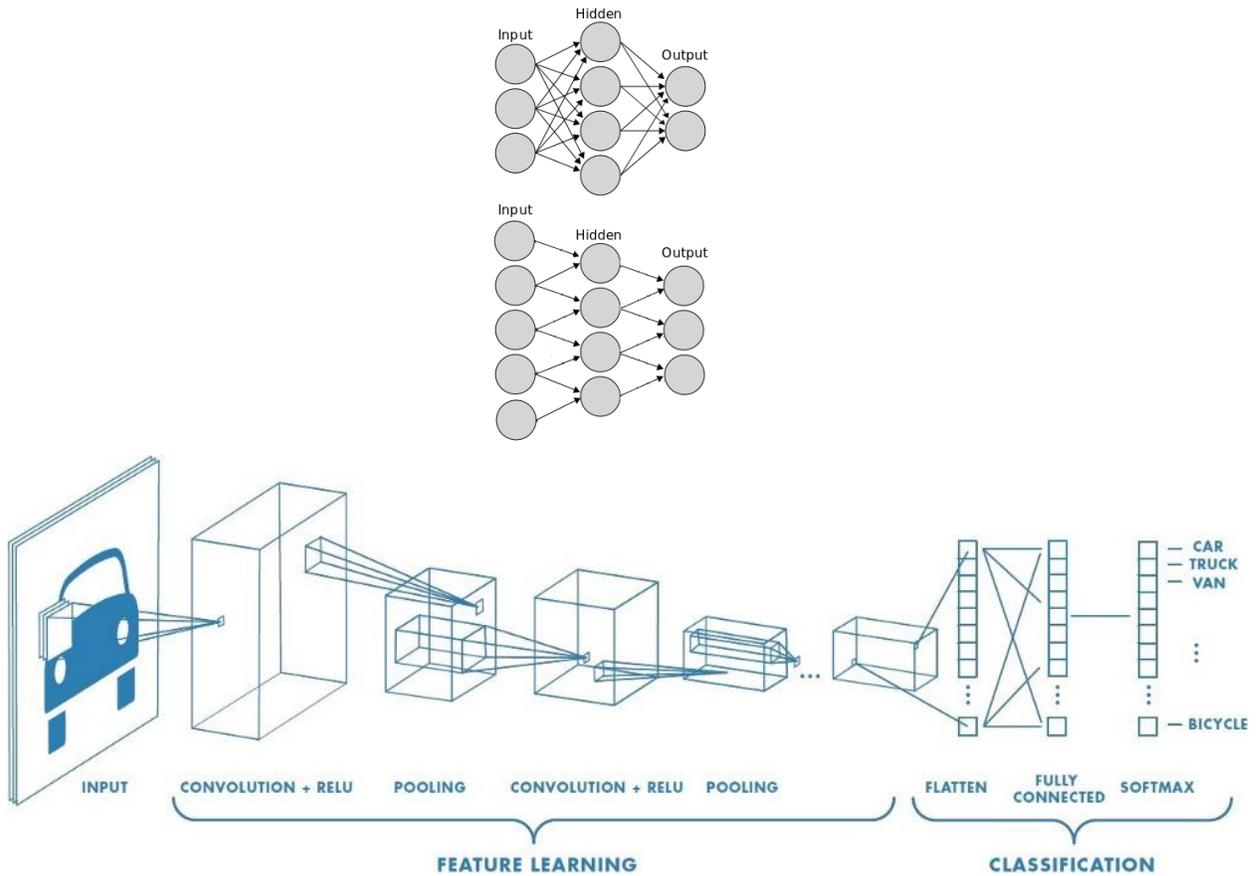


Figure 8: CNN vs FcNN (Figure is taken from [28]), Comprehensive Guide to CNNs (Figure is taken from [29])

2.5.2 Recurrent Neural Network

Recurrent Neural Networks (RNNs) [30] are introduced by John J. Hopfield [30] and are considered as an important type of ANNs. RNNs use their internal memory to process variable-length sequences of inputs. i.e., RNNs use the output of each node as one of its own input

dimensions in the next iteration, which clearly indicates the current output would be effective on the consecutive output of the same node.

The performance of RNN is not significantly stable and as a result, it is not reliable in the problems which need long-lasting states. As an example, after applying an RNN network on plenty of samples belonging to one target, it's almost impossible to save adaptability of the hypothesis on the samples from any other classes, in other words, the scope of memory in RNNs is very short which enforces them to preserve adaptability to a couple of last states. That is why RNNs are called short term memories. Besides their effectivity and applicability on many practical and theoretical projects, their weakness in handling long span memorial states brought up the new forms to deal with sequential problems. Figure 9 shows basic feedforward neural networks and basic recurrent neural networks.

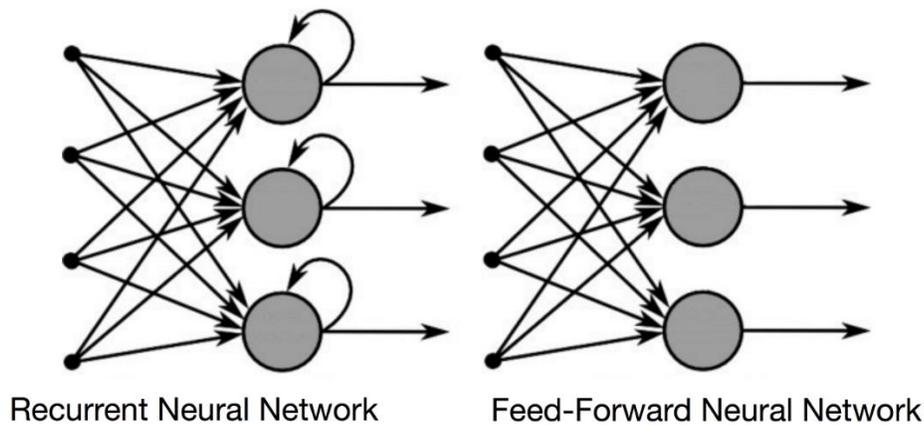


Figure 9: RNN vs FFNN (Figure is taken from [31])

2.5.3 Long Short-Term Memory

Long Short-Term Memories [32] were introduced to address problems that were related to RNNs. The introduction of LSTM was a turning point in the history of the sequential data learning process. It is because LSTMs have higher stability compared to basic RNNs. LSTMs have long and stable state preserving capability compared to RNNs and that is why these networks are called Long short-term memories.

Based on the capability of LSTMs, they can extract sequential patterns accurately. LSTMs could solve one of the main problems regarding RNNs which is the vanishing gradient; however, the exploding gradient is kept on.

Structurally, LSTM consists of three gates which respectively are: forget gate, memory gate, and output gate. The introduction of using the three gates was the idea to improve the scope of memory in any deal with a sequential learning problem. The three gates leverage LSTMs to handle dependencies in sequential events.

In summary, LSTMs have a memory cell, which determines how much we are intended to intervene in former events on the next sequences through time. Applying memory cells is the way that LSTMs use to prevent the vanishing gradient problem. Despite clearly defined details, a couple of structures are introduced for LSTMs which they have small differences. The differences rely on the connections among the gates in each structure. Figure 10 shows the structure of an LSTM.

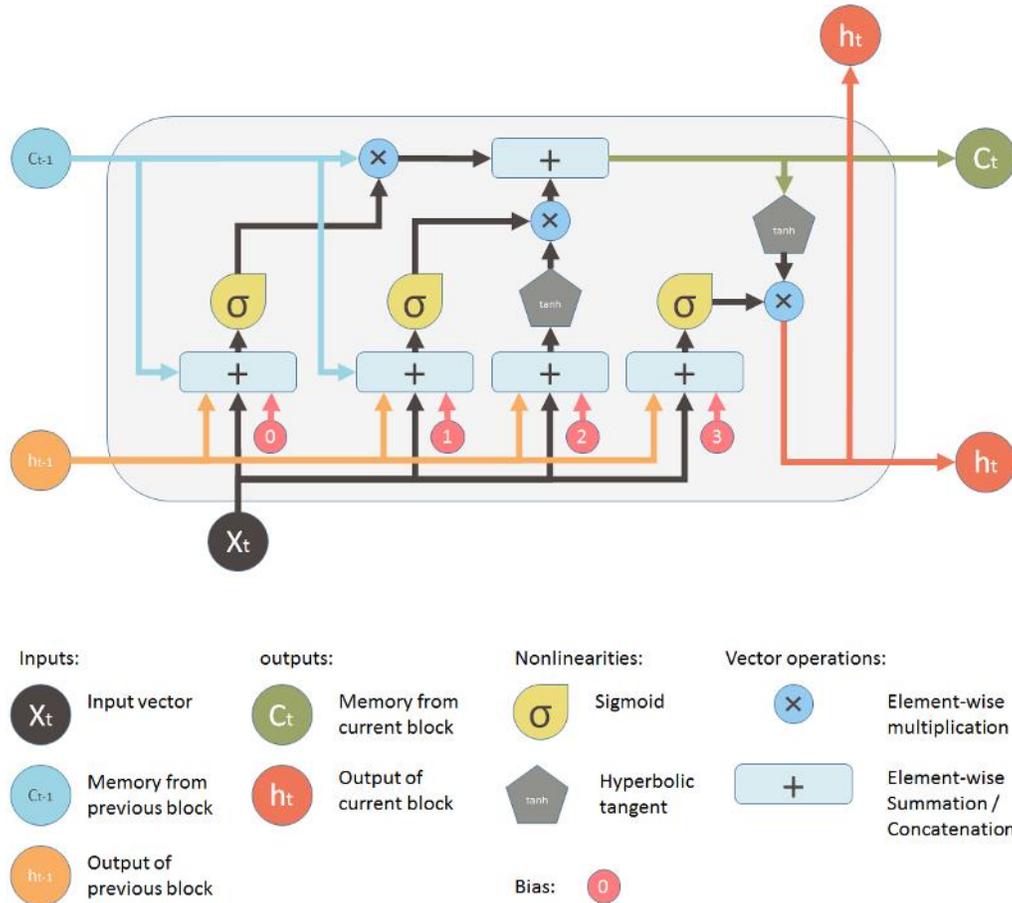


Figure 10: Long Short-Term Memory (Figure is taken from [33])

Despite the strong capability of LSTMs in dealing with sequential problems, a couple of problems are related to them. One of the most important problems was their latency. i.e., LSTMs are not that much fast which they are desired to be. The latency problem was the motivation for more research and discoveries which led to a new Recurrent network called Gated Recurrent Unit (GRU).

Even though GRUs were fast enough compared to LSTMs, (because GRUs have two gates: forget gate and memory gate) their accuracy and performance in terms of sequential inception factors were not as high as LSTMs were. Therefore, LSTMs are known to cover a longer scope of memory in comparison with GRUs. All in all, both are the most well-known recurrent networks for sequential problems.

Practically, when we are talking about RNNs, either we are referring to LSTMs, or GRUs (practically mostly it is referred to LSTMs) because both networks outperform basic RNNs, obviously. Figure 11 depicts the network of a GRU.

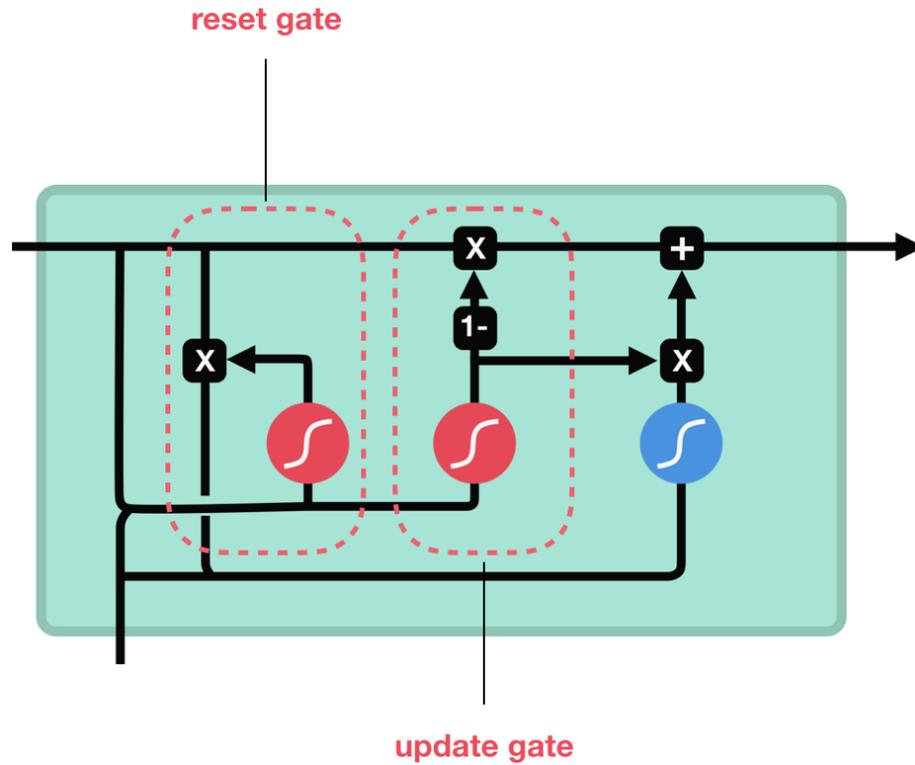


Figure 11: Gated Recurrent Unit (Figure is taken from [34])

Figure 12 illustrates three sequential learning networks including RNNs, LSTMs, and GRUs respectively.

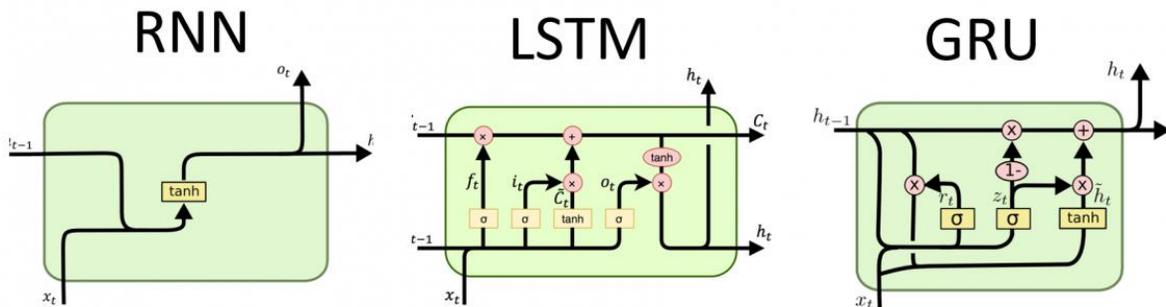


Figure 12: RNN vs LSTM vs GRU (Figure is taken from [35])

2.5.4 Residual Neural Networks

For a long time, consistent connections between neurons of different layers were the only connection form in neural networks. Residual NNs gave a new topological form to NNs which was a solution for vanishing gradient, as well [36].

In fact, Residual-based networks could be helpful to avoid continuous reduction of gradients during propagating back. Thus, we connect a neuron to neuron/s in the next layers, while we skip at least one layer. As a result, we prevent at least one step of fractional decrease on loss value, consequently, the gradients would decrease on backpropagation with a lower acceleration compared to a normally connected NN (consistent connections among neurons in different layers).

Residual NNs are categorized into three types, that is while the three types differ only from topological view. Respectively, the three types of residual networks are Plain NNs, Highway NNs, and Dense NNs [36].

Accordingly, if we have not ignored any connection/s, i.e., if each neuron is exactly connected to all/partially of nodes in the next and previous layer, the network is called Plain NN, and it's the typically used topology before the context of the Residual NNs, so connectivity to neurons in exactly next and previous layers is expected and no skipping over any layer happens.

If the network has any neuron which skips one layer and merges any node in the followed layer after the next layer nodes, it's called Residual Net (ResNet) [36]. Thus, in ResNet, we have at least one connection between a node in layer X to another node in layer $X+2$ (i.e., the connectivity jumps over the layer $X+1$). Moreover, if a network has a connection between two nodes with more than one skipping layer (i.e., not one, and it is two or three layers), the corresponding NN is called Highway Network [36]. Finally, networks that have nodes with more than three – several - parallel skipping layers are known as Dense Networks (DenseNets). Figure 13 demonstrates the Residual Nets compared to Plain Nets in detail.

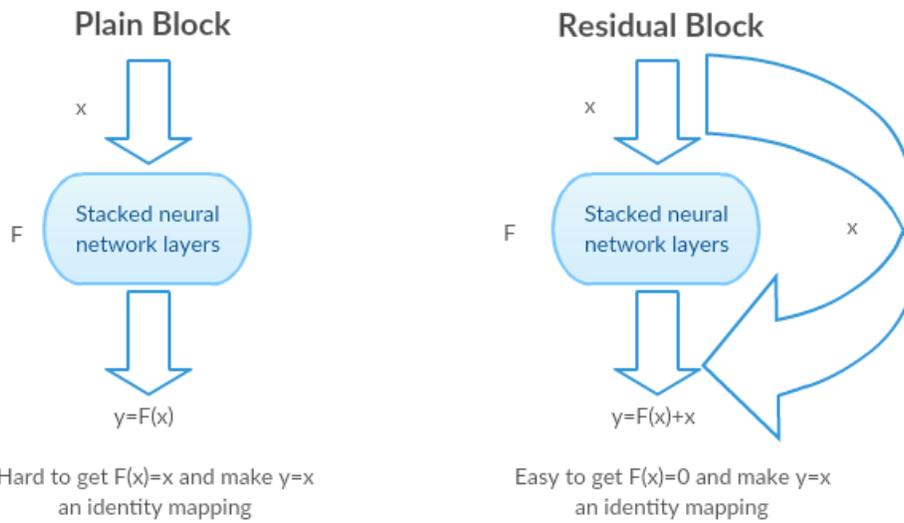


Figure 13: Residual vs Plain nets (Figure is taken from [37])

As the number of skipped layers increases, the network is less susceptible to encounter a vanishing gradient [36]. Respectively, ResNets, Highway Nets, and Dense Nets are considered as solutions for vanishing gradient, since the amount of loss value which is propagated back to the earlier layers would increase compared to plain networks.

3. RELATED LITERATURE

As we already explained, one of the major goals in unsupervised learning methods is determining the latent structure hidden inside the data, and the significant characteristic of neural networks is their salient capability in this task. Generally speaking, there is not any perfect network, such that, that addresses all of the problems associated with the unsupervised learning and outperforms all of the other alternatives in any benchmark or real-life datasets. Therefore, the field is still very fertile, and many different proposals are made to handle various problems that is faced in applications. Among these proposals, particularly Auto Encoders (AE) and Generative Adversarial Networks (GANs) received much attention lately, partially due to their (and some of their variants') high success as generative models. In this chapter, we are going to discuss various aspects of AE as well as GANs in more detail since they constitute the basic components of the proposed model.

3.1 Auto Encoders

Auto Encoders (AE) are well-known NN algorithms that are used for unsupervised learning problems introduced by (Rumelhart et al., 1986; [38] Baldi and Hornik, 1989) as a technique for Dimensionality Reduction. The overall goal of AE is to regenerate the input data that is fed to it. That is to say, AE basically learns the feature maps of the input and uses that feature maps to yield an output, which supposedly should be equal to the input.

AEs have two independent networks respectively named: encoder and decoder. An encoder, as implied by the name, encodes a given sample to a latent feature map, on the other hand, a decoder decodes a latent feature map to an output which is supposedly is same as the given input.

In order to regenerate the same input data, the encoder and the decoder networks should have compatible configurations. For example, the size of the input data vector and the size of the output data vector should be the same, also the number of hidden layers in both networks should be the same as well. Figure 14 illustrates the general structure of an Auto-Encoder:

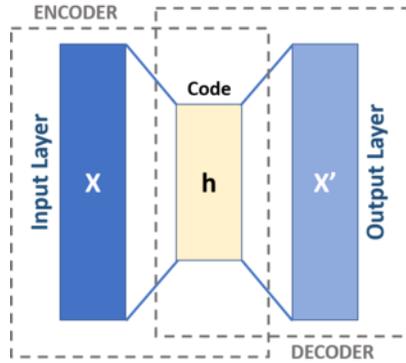


Figure 14: The general structure of Auto Encoders (AE) (Figure is taken from [39])

The layer that corresponds to the intersection between the encoder and the decoder networks are referred to as the code (i.e., the bottleneck). The output of the code layer which is realized for a particular input becomes the input for the decoder network. The output of the code layer corresponds to the representation of the input data in the identified latent feature map (i.e., latent space). In the context of image processing, the input vector size corresponds to the bits of the image and the size of the code corresponds to the dimension of the latent space.

In the literature many different types of AEs are available, and each has been purposed toward a specific type of problem. According to the [40], Dimensionality reduction, Feature fusion, Anomaly Detection [41], Drug Discovery [42], Machine Translation [43], Popularity prediction [44], and Information Retrieval [45] are practical applications of AEs. Next, we will introduce the most popular AEs and provide relevant information regarding their structure and use in practice.

3.1.1 Under Complete Auto-Encoders

If the number of nodes in the core layer is less than the number of input nodes, the network is called Under Complete Auto Encoders (UCAE). UCAEs are the simplest types of AEs, which we restrain the code distributions to catch significant features of the data. That is to say, UCAEs are designed in order to catch the most important and salient features of the input data. Therefore, dimensionality reduction is one of the major goals of UCAEs.

UCAEs can be considered as the advanced version of Principal Component Analysis (PCA). While PCA just captures the linear hypothesis of data, UCAE could catch any hyperdimensional scheme available in the data. In fact, NNs' capability on feature extraction enables UCAEs to find any higher-dimensional manifold for data presentation and encode it to lower dimensionality. In the [40], it is demonstrated the dimensionality reduction as the practical application of UCAEs.

An example of a UCAE is presented in Figure 15. As shown in the figure, the number of nodes in the code layer (i.e., the intersection layer of the encoder and decoder networks) is less than the number of input nodes.

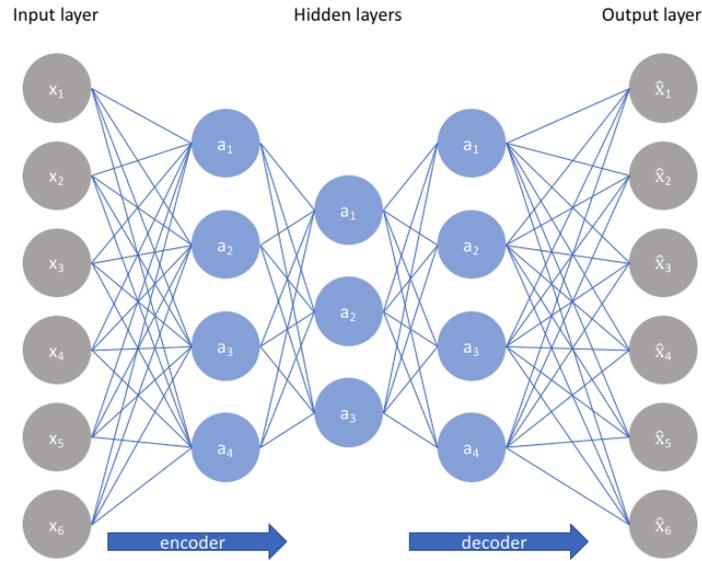


Figure 15: Undercomplete Auto-Encoder (UCAE) (Figure is taken from [46])

3.1.2 Complete Auto-Encoders

If the number of nodes in the code layer is the *same* as the number of nodes in the input layer, the network is referred to as the Complete Auto Encoder (CoAE). Thus, the number of nodes in the code layer is equal to the size of the input vector. Thus, in the case of CoAE, we do not determine the size of the code layer, instead, it is equal to the size of the input layer. Figure 16, depicts details of complete AEs.

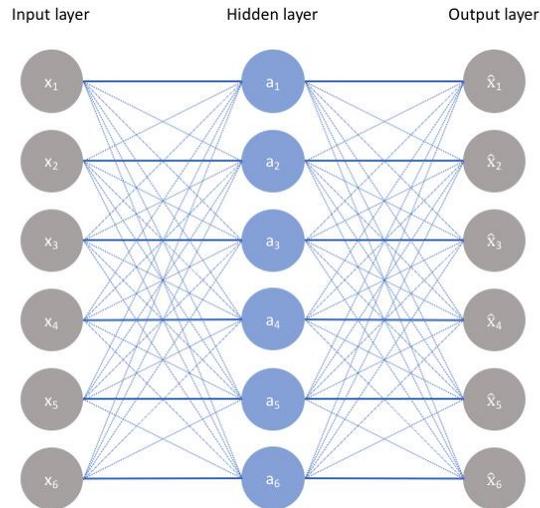


Figure 16: Complete Auto-Encoders (Figure is taken from [46])

3.1.3 Over Complete Auto-Encoders

If relations among attributes could not be summarized by a code layer that has the same number of nodes with the input layer, obviously, the model would need to add more nodes to the code layer in order to extract hidden features maps. If the size of the code is greater than the number of nodes in the input/output layer, then the network is referred to as Over Complete Auto Encoders (OCAE). In OCAE the code layer is not referred to as the bottleneck anymore since it is not limited to values smaller than input layer size. Figure 17 depicts the architecture of an OCAEs.

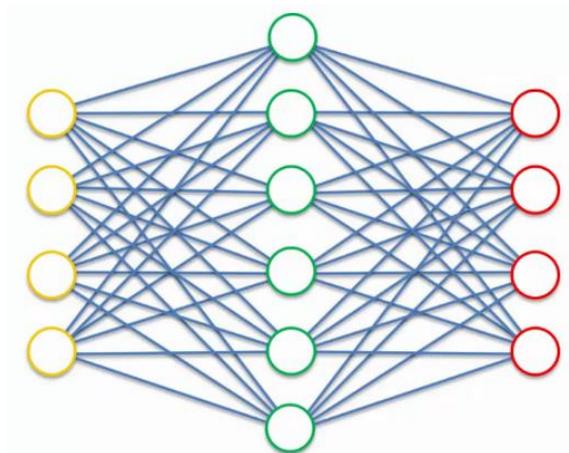


Figure 17: Over Complete Auto-Encoders (Figure is taken from [47])

3.1.4 Regularized Auto Encoders

The above discussed three models are usually considered as the basic AEs. Note that, since AEs are targeting to reconstruct (or regenerate) the input as its output, overfitting (e.g., learning a trivial identity function) is a major risk for AEs. Even though UCAE, by choosing the size (i.e., the number of nodes) of the code layer less than the number of nodes in the input layer which imposes a constraint to avoid overfitting, this might not be sufficient and overfitting is still possible. Therefore, one might need to incorporate further regularizations for generalization purposes and avoid ending up with overfitting. Regularized AEs basically introduces further regularizations in various forms. Next, we are going to discuss various RAEs such as Sparse AEs (SAE), Denoising AEs (DAE), Contractive AEs (CAE), Stacked AEs (StAEs), and Variational AEs (VAEs).

3.1.4.1 Sparse Auto Encoders

Sparse AEs (SAEs) [48], basically adds a regularization cost (like in general regularization) to the overall loss function of the NN, and intentionally tries to silence (i.e., deactivate or yield a very weak output) some of the nodes in the code layer and allows only a few of the nodes of the code layer becomes active at any given pass. Since not all the nodes are activated at once, sparsity is attained, thus the AEs are referred to as SAEs. In this way, even if the size of the code layer is greater than the size of the input layer since at any moment only some of them are allowed to be activated, the number of activated nodes of the code layer practically becomes less than the number of input layers. Thus, overfitting becomes less likely.

The desired objective would be at any moment the set of activated nodes represents a particular characteristic that exists in the latent feature space. If this is attained, the more common a pattern is throughout data samples, the more active would be the nodes which are representing those patterns. Note that, activation/deactivation of a neuron solely depends on weight vectors throughout a regularization process. The way that regularization is handled enables SAE to be architecturally data dependent.

In order to ensure the sparsity, frequently L_1 -Regularization and KL-Divergence are used as regularization constraints in SAEs.

L_1 -Regularization: where we use a tuning parameter (λ) product on the absolute sum of vector activations $-a_i^{(h)}$ - present in layer h for observation i_{th} , so, our final objective function is the Equation (2)

$$\mathcal{L}(x, \hat{x}) + \lambda \sum_i |a_i^{(h)}| \quad (2)$$

KL-Divergence: The Kullback-Leibler divergence (KL-divergence) is an asymmetric measure that indicates how a probability distribution is different from another. It is also referred to as the relative entropy. Let the ρ be the distribution of the (*desired*) average activations of the nodes in the code layer. At each epoch, we calculate the sparsity values, denoted as $\hat{\rho}$, which is the average of activations for the j^{th} neuron (in the code layer) throughout all samples. Equation (3) is used in this calculation where i is the index that corresponds to the sample and $a_i^{(h)}(x)$ denotes the activation obtained for the node in the h^{th} layer (i.e., the code layer) when input x is fed to the network.

$$\hat{\rho}_j = \frac{1}{m} \sum_i [a_i^{(h)}(x)] \quad (3)$$

Note that the $KL(\rho \parallel \hat{\rho})$ is used as the regularization and added to the loss function in the objective. Thus, the goal is forcing the realization of the activations to have a distribution to the desired distribution (i.e., ground truth) which is usually close to zero so that the scarcity is ensured.

$$\mathcal{L}(x, \hat{x}) + \sum_j KL(\rho \parallel \hat{\rho}) \quad (4)$$

So, we just aim to restrain activations in the hidden layer by applying either of the constraints on each neuron to enforce sparsity. The Equation (4) demonstrates loss function for SAEs. In Figure 18 we have depicted the architecture of a sparse AE in detail.

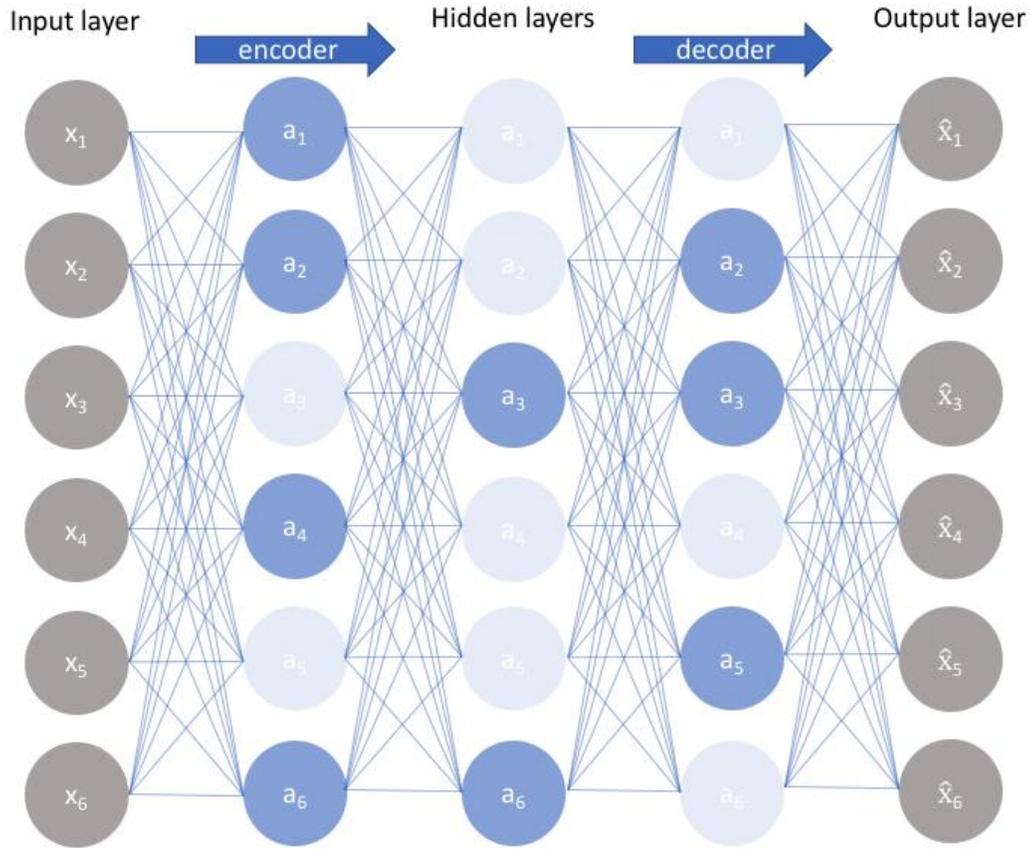


Figure 18: Sparse Auto-Encoder (Figure is taken from [46])

3.1.4.2 Denoising Auto-Encoders

The word “Denoising” refers to removing added random noise to the raw input data before feeding it to the AE. So, a Denoising AE (DAE) receives intentionally corrupted data, and it is expected to denoise it and produce noise-free data [49]. The output of a DAE is compared with the real data sample, not the noisy one while calculating the loss function during the backpropagation. Thus, a DAE is responsible for removing corruption out of a noisy input and its output is compared with the corresponding real sample - free of noise - to evaluate the model performance.

Equation (5) is used as the loss function where $f(x)$ assumes the noisy input x .

$$\mathcal{L}(x, \hat{x}) = |\hat{x} - g(f(x))| \quad (5)$$

In fact, denoising helps the AE to learn the patterns and representations in the real input data. So, the decoder will resist to small perturbations. In other words, a DAE tries to make the decoder to have the locality properties (i.e., the main features in the latent space), thus, small changes in the hidden layer lead to as small as possible changes in the output layer. Consequently, the decoder

would try to underestimate any small outlier with the corresponding distribution. In other words, a *robust* AE is trained. Figure 19 depicts the configuration of a Denoising Auto-Encoder.

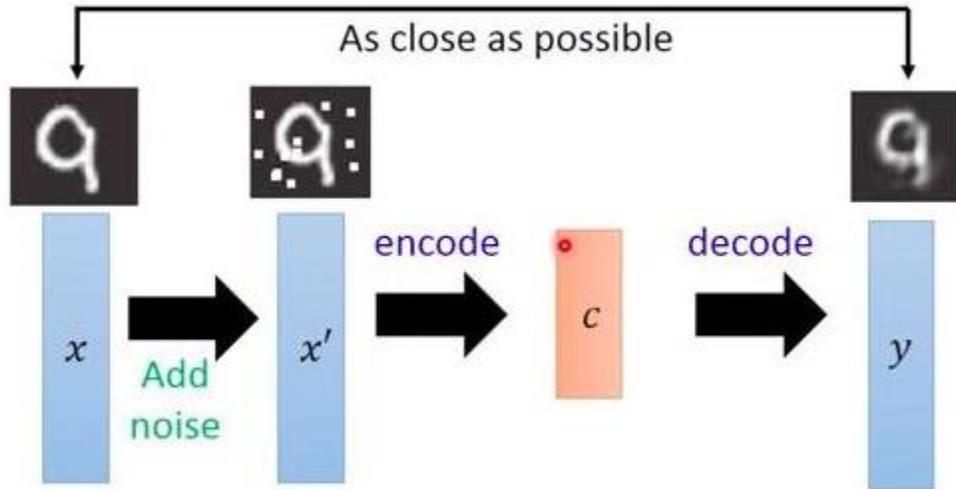


Figure 19: Denoising Auto-Encoder (Figure is taken from [50])

3.1.4.3 Contractive AEs

Rifai et al. (2011) [51] introduced Contractive AEs (CAEs) which try to minimize the effects of any tiny variations on input data i.e., develop a robust NN just like DAEs but with a different approach. CAEs try to underestimate any small (local) mismatching in input space and utilizes a penalty term based on the Jacobian Matrix, for this purpose. In summary, the idea behind CAEs is that we are looking for a less sensitive model or more robust learned representation on small variations in the input data samples. CAEs enforce the neighborhood of input data to a smaller neighborhood of the output space.

So, to make the model more robust CAEs adds a penalty term on loss function which is the square of the Frobenius norm of the Jacobian matrix in the code layer. The Jacobian matrix of the code layer (say layer h) is basically the derivative of the node j^{th} in the layer (i.e., $h_j(x)$) with respect to each input (x_i). Note that the Frobenius norm is basically the square root of the sum of the squares of all the elements in a matrix. The objective function of CAEs and the Frobenius norm is provided in Equations (6) and (7).

$$\mathcal{L}(x, \hat{x}) = |x - g(f(x))| + \lambda \|J_f(x)\|_F^2 \quad (6)$$

$$\|J_f(x)\|_F^2 = \sum_{i,j} \left(\frac{\partial h_j(x)}{\partial x_i} \right)^2 \quad (7)$$

Both DAE and CAE aims to come up with a NN model that is robust and less sensitive to perturbations in the input data so that the model’s generalization capability is enhanced. DAE addresses this objective by intentionally adding noise to the input data, so the weights of the NN is determined to handle such small perturbations. On the other hand, CAE penalizes deviations at the code layer due to such perturbations. Son in a sense, the focus of CAE is “directly” to the code layer, on the other hand, DAE *dilutes* the focus to the whole encoder-decoder NN.

3.1.4.4 Stacked Auto-Encoders

The main goal behind the introduction of the Stacked Auto-Encoder is to obtain much detailed and summarized version of the raw data, while the promising feature information is preserved. i.e., summarization aims to find the best feature representation of the input samples.

Generally, StAE consists of several layers of SAEs such that the output of each hidden layer is connected to the input of the following hidden layer. Thus, this hierarchical form of dimensionality reduction will enforce the hidden layer representation to be as detailed as possible. While stacking will continue summarization if the quality of reconstructed data is not lowered. Representation of online advertisement strategies is a practical application of StAEs, research in predicting the popularity of social media posts demonstrates promising potential of StAEs [44]. Figure 20 depicts the scheme of a Stacked (Denoising) Auto-Encoder.

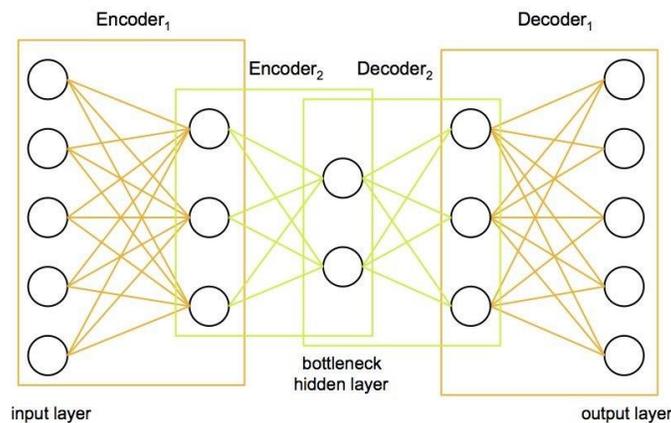


Figure 20: Stacked Auto-Encoder (Figure is taken from [52])

3.1.5 Variational Auto-Encoders

AEs have a significant problem which is the possibility of memorizing despite the remarkable capabilities they have in unsupervised learning in particular feature extraction. Even if the latent layer or the bottleneck has one node, memorizing is possible for the network. This problem triggered a turning point in the history of AEs toward a new idea to overcome it.

The new idea opened a simple and reasonable new branch in AEs which was adding a constraint on the encoder network, such that the encoder generates codes that follow a unique Gaussian distribution. So, now the AE not only does not memorize input data but also it can generate new samples as similar as possible to the real samples by learning their main features. This frame of AE is called Variational Auto-Encoder (VAEs) [53], and in some others called Generative Auto-Encoders (GAEs) due to its generative capabilities.

Thus, VAEs have constraints to limit the given data set into a gaussian distribution in order to avoid overfitting- memorizing, which is accomplished via adding two layers, namely, one layer for the mean vector and the other layer for the standard deviation vector. The network in Figure 21 depicts a general structure of a VAE.

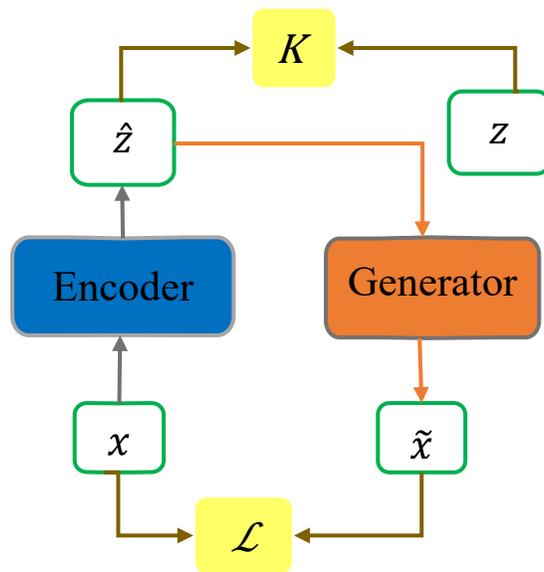


Figure 21 Variational Auto-Encoder
(Figure is adapted from [54])

Despite the fact that the VAEs could resolve the main problem of AEs which was to avoid memorizing, VAEs also faced a new and important problem. Since VAEs are dealing with data distributions and parametrical feature extractions, as expected, samples that are generated

throughout a VAE are somewhat blurred. Thereafter, there was a new problem to address, and it was generating more qualified data samples.

VAEs performance in terms of the quality mostly depends on the standard deviations, and as the number of samples deviated from mean increases, expectedly, generated samples will be more qualified. According to a group work performed on population synthesis by approximating high-dimensional survey data, VAEs have demonstrated valuable capability to be applied [55].

Basically, AEs catch dominant features of samples, however, unintentionally it would lead to many interrelations among samples which consequently would result in training to be more complicated.

Practically, VAEs categorize any interrelations among samples via two parameters: mean and standard deviation. In other words, distributions of classes match data distribution, thus the model behaves separately for different categories. That is while the model has no label since it is dealing with unsupervised data. This characteristic of VAEs enables them to seem more realistic on new data generation process due to their distributional information access rather than normal AEs.

VAEs besides their designation and goal were used in generative tasks as well, such that VAEs could generate new data samples and that is why in some resources VAEs are called generative AEs. By using distributional parameters, VAEs could obtain enough info about the whole dataset which enables them to generate new data which is a combination of whole dataset status regarding each attribute.

Artificial data was the topic that ensued from the capability offered by VAEs to generated new samples i.e. we are interested in models that could generate new data, incredibly similar and undetectable compared to real samples, and finally, the motion led to what today is called Generative Models. We strongly believe variational models and generative models are in a sequence and latterly we would talk about this topic in detail.

3.2 Generative Adversarial Models

VAEs performing well in terms of generating *new* (i.e., artificial) data, however, they miss a model to evaluate the quality of generated data. In other words, VAEs generate new data while no judgment is made on new data to verify them in terms of similarity and realism perceptuality in comparison with visible data. This point gave a reason for the introduction of a new model to generate new (artificial) data.

The idea behind GANs is adding a new *adversarial* network as the judge for the generative network. It was initially proposed by Goodfellow et al. [2014] [56]. So, GANs are composed of two networks targeting against each other i.e. they are both pitting each other. The first network is a generator that aims to generate artificial data that fools the other network which is called the discriminator. As its name clearly indicates, the discriminator network tries to make a distinction

between artificial data and the real samples as accurately as possible. Figure 22 captures the overall structure of a GAN.

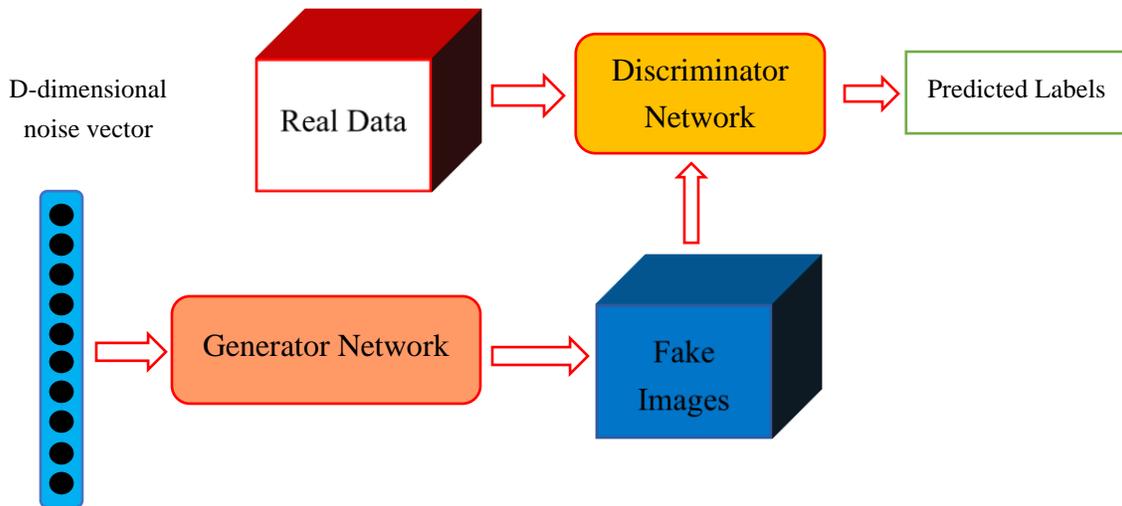


Figure 22: Generative Adversarial Networks (GAN) (Figure is adapted from [57])

An important point about GANs and VAEs is that both are dealing with data rather than labels, i.e., they are in the realm of unsupervised learning. In fact, when we take a model as a generative model, we are referring to the capability of the model to produce new data (i.e., artificial data). Basically, variational models were the starting point which led to the introduction of GANs. We can conclude that the three words Variational, Generative, and Creative are in a nice sequence, respectively. All are leading to the definition of basics for Machine Consciousness in the close future.

As we described the network of a GAN previously, it consists of two networks which are involved in a competition- while the generator tries to fool the discriminator by generating artificial data as similar as possible to the real samples, the discriminator tries to differentiate the generated data and real samples as accurately as possible. Thus, an adversarial min-max game between two networks takes place. Two objectives are corresponding to two networks respectively, which are contradicting each other inherently. The discriminator's objective is maximizing the accuracy of itself; the generator's objective is minimizing the accuracy of the discriminator while undertaking the discrimination task. The training of new samples throughout GANs is an iterative process that is explained in detail as follows:

- Train the Discriminator: Artificial data are generated by feeding some random vectors to the generator network. These artificial data and the real samples are fed to the discriminator which specifies a label as the output. The output of the discriminator is a probability of being a real sample or artificial data that can be used to specify a label. By using the loss

function and backpropagation the discriminator is trained so that it can defeat (i.e., catch) the generator.

- Train Generator: The generator is trained based on the loss values produced by the discriminator for artificial data only. Again, backpropagation is used during the training of the generator.

The two stages described above are iteratively conducted until an equilibrium is reached. Ideally in the equilibrium, the probability of being artificial data or real sample is $\frac{1}{2}$ for each input to the discriminator, no matter if it is a real sample or artificial data. The best result for the generator is the worst result for the discriminator. That is why this framework is referred to as the adversarial networks.

A competition between two networks against each other eventually yields a good performing generator network. Therefore, this generator network can be used to generate artificial data that cannot be distinguished from the real samples. After the training of the GANS, the generator network can thus be used to generate new artificial data. Figure 23 demonstrates the scheme of a GAN.

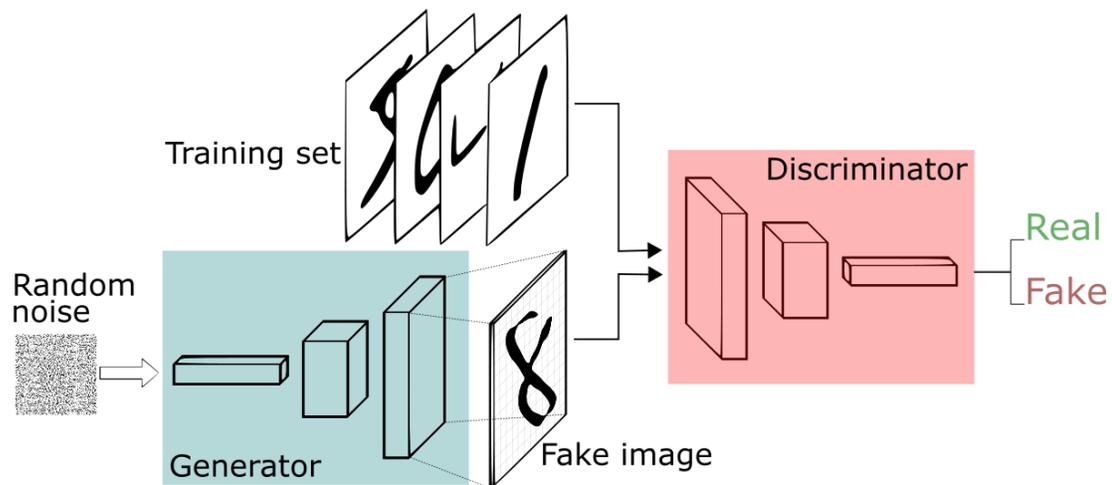


Figure 23: Generative Adversarial Network (Figure is taken from [58])

The architectures and topologies used per each network are independent of each other and the overall features of the two networks are problem dependent.

GANs were a turning point for generative models. The introduction of GANs brought up many successes over the quality of new data generation. Despite their large successes and advantages, GANs have some basic problems which affect their performance negatively.

One of the main problems related to GANs is referred to as the Mode Collapse (i.e., the Helvetica scenario) problem. If not dealt appropriately, the generator starts generating artificial data only

similar to a specific mode of the real samples, and consequently, generated artificial data do not vary enough. In other words, we would have limited variations in the generated artificial data. Mode Collapse is one of the main reasons for another problem which is referred to as the Missing Mode problem. The Missing Mode happens when the generator misses some of the modes of the data and insists on specific modes.

The other problem with GANs is referred to as the diminished gradient, which takes place when the discriminator network outperforms the generator network. As a result, discriminator becomes dominant on the generator over time. The diminished gradient would lessen the learning capability of a generator. The problem can be addressed by changing the architectures of the generator and discriminator.

All these problems associated with GANs have been addressed in various ways and new problems have been discovered as well. One of the proposed solutions is a new model called Bidirectional GANs- Bi-GANs- which we will discuss later in this chapter.

3.2.1 Adversarial Auto-Encoder – AAE

Before discussing the details of the Bi-GANs, we just go over a new type of Auto-Encoders which is based on the adversarial game of generative models. It is called Adversarial Auto-Encoders (AAEs), [59]. The idea behind AAEs is that we can use the latent distribution as the input for a discriminator to discriminate between a real latent distribution and the fake one. Figure 24 depicts the general form of an AAE.

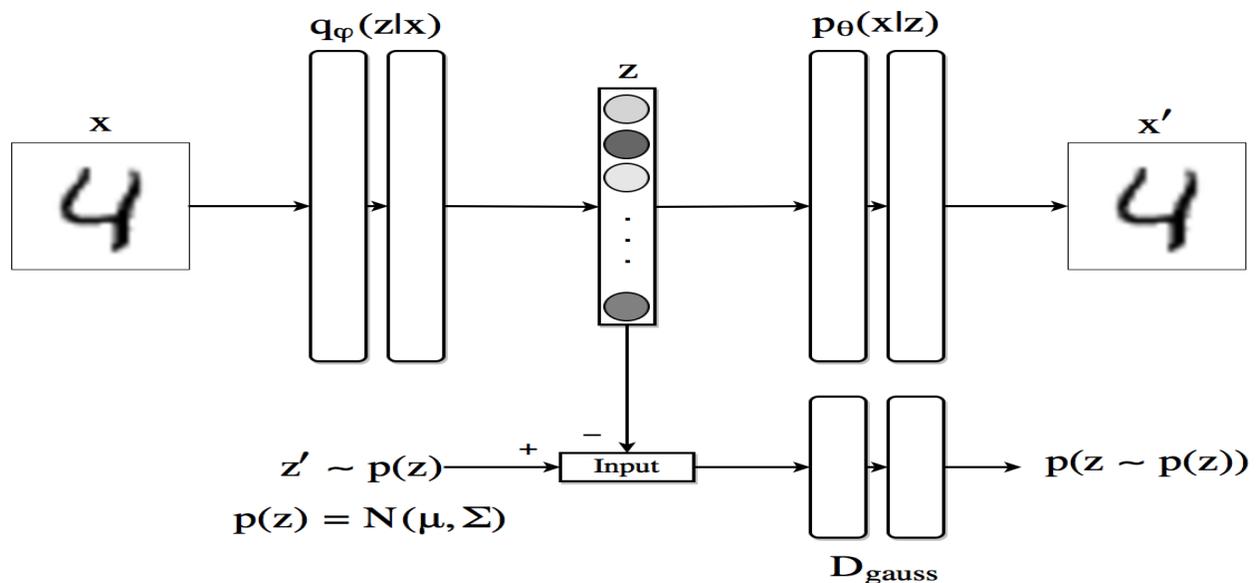


Figure 24: Adversarial Auto-Encoder(Figure is taken from [59])

As demonstrated in Figure 24, an AAE is an AE (where the code is restricted to represent a generated latent distribution) along with a discriminator. The discriminator deals with the latent distributions (i.e., the generated distribution by the encoder and the “real” distribution that is imposed on the code). The training of an AAE is done in two phases. The first phase is training the AE (i.e., encoder and decoder) based on the reconstruction error. The second phase is done via the adversarial game between the encoder (i.e., the *generator*) network and the discriminator. During the first phase, the Discriminator is frozen, and during the second phase, the decoder of the AE is frozen. When the AE is trained well, it is the time to ignore the decoder part of the network, consequently, we have one encoder (i.e., generator) and one discriminator. Subsequently, the adversarial game starts between these two networks. During the adversarial game, the generator (i.e., the encoder of AE) generates code and tries to fool the discriminator. At this stage, both the generated code and the “real” samples from the “real” distribution are used to train the discriminator with backpropagation. As the training process of GANs next the generator (i.e., the encoder) is trained.

After the training of AAE, samples from the imposed distribution can be fed to the trained decoder network which generates artificial data in the visible space. Figure 25 provides a snapshot of how an AAE works.

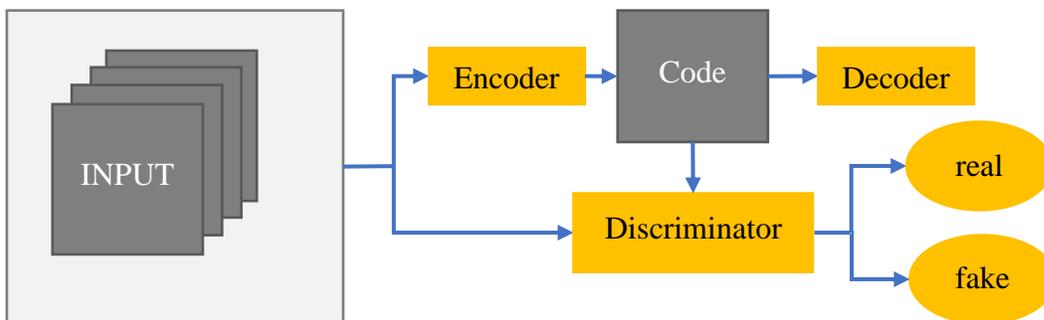


Figure 25: Schematic view of an Adversarial Auto-Encoder (Figure is adapted from [60])

3.2.2 Bidirectional Generative Adversarial Networks

Despite GANs were a great success at the new data generation process, various problems which were already mentioned above were associated with them. One of the main weaknesses of GANs was their unidirectional feature extraction, such that GANs just try to map latent space on real data samples. It means that GANs can cover all modes on latent space, but there is no guarantee that for GANs to cover all modes of real data (recall the mode collapse problem).

Due to the mode collapse problem (and the missing mode problem associated with it), GANs performance in an unsupervised learning context (i.e., as a generative model) is questionable. If GANs are trained with unsupervised data where the samples all belong to a single mode in that case GANs generally perform well and generate artificial data that is indistinguishable from the training data. However, if the training data has multiple modes in that case it is possible that for some of the modes no artificial data would be generated after the training. In other words, the generator will not generate artificial data with equal quality and quantity, because it may not cover equally all the modes.

So, an alternative component is needed to equip GANs to cover the whole of the data distribution. The extra component which Bi-GANs [61] brought it up to be used was an Encoder. Bi-GANs had attached an encoder onto the generator part to add an inverse mapping- real data to latent space. Thus, Bi-GANs not only map latent space on real data but also, they do the map in inverse direction- real samples to latent space (e.g. desired distribution). Figure 26 presents the structure of a Bi-GAN.

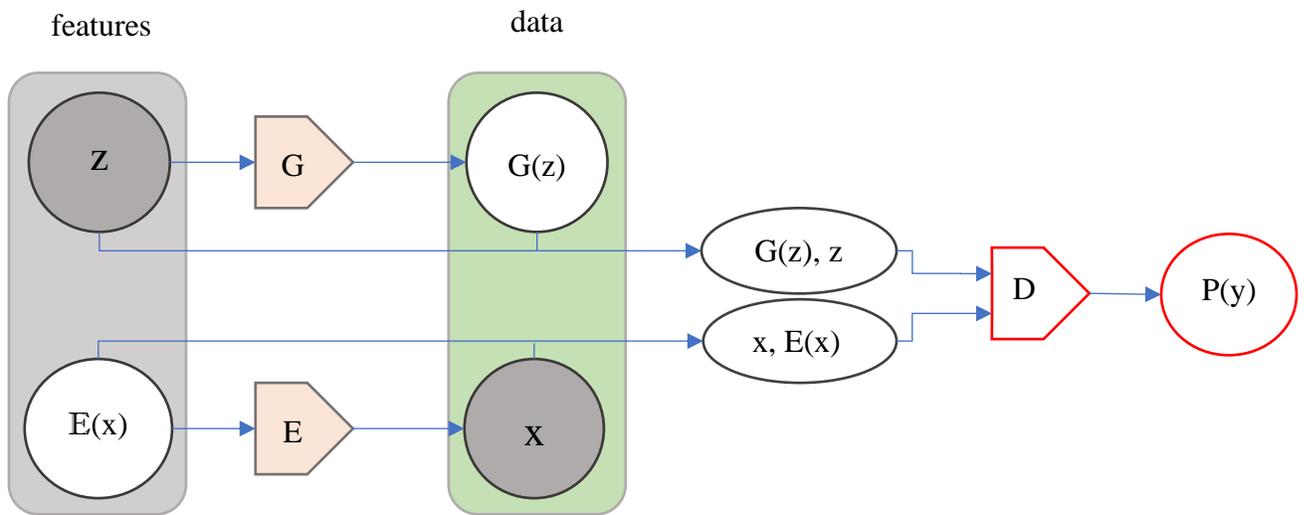


Figure 26: Bidirectional Generative Adversarial Networks (Figure is Adapted from [61])

An important point about the architecture of a Bi-GAN is that the encoder and the generator have no interaction with each other. The generator takes a random vector (z) which is sampled from the desired distribution and generates an artificial data $G(z)$ (say, an image). At the same time, the encoder takes real data (x) (say an image) and generates an artificial data ($E(x)$) supposedly from the desired distribution. Thus, two tuples are obtained from the generator and encoder, respectively. These tuples are next fed to the discriminator. Discriminator compares the real data (x) with the artificial data ($G(z)$) and it also compares artificial data sampled from the desired distribution with the random vector sampled from the desired distribution and gives feedback.

The main contribution of Bi-GANs is the encoder which is added to the generator part of the model. Thus, in Bi-GANs we have two objective functions.

The first objective function is related to the quality of the generated samples, the same as basic GANs. (x vs. $G(z)$). The second one is the objective function for comparing the generated code vs. and real sample from the desired distributions. Such a set up tries to avoid the generator network to collapse any specific mode of the real data space.

A modified version of Bi-GANs is referred to as the latent regressor in which we can give the output of the generator to the encoder as well. As a result, the encoder could take either of the real data and generated visible data as inputs to output the corresponding latent data. The latent regressor is advised in the feature learning context.

One of the main problems referred to the Bi-GANs is the non-discriminative loss between the generator and the encoder. In other words, the invalid latent space mapping caused by imperfect encoder misleads the generator's update in backpropagation. So, in a Bi-GAN if the discriminator could find out differences between the real data and the generated visible data, both generator and the encoder are needed to be backpropagated. That is while one of them maybe is converged enough to its perfect condition. Thus, its convergence obviously would decrease after backpropagation which in turn may lead to endless oscillations and creates problems in terms of the stability of the network.

3.2.3 The main drawback of Bi-GANs

Bi-GANs came up to address the mode collapse problem in GANs. The generator part of the Bi-GAN models comprises an encoder besides the generator. The encoder maps the real data to the latent space which is gaussian. The outputs of the generator and the encoder are fed into the discriminator, the discriminator gives a loss based on the result of the two networks.

The main problem of the Bi-GAN model is that the single additive loss would be backpropagated on two independent networks (i.e., the generator and the encoder). In other words, the discriminator does not give two losses for the two networks independently, instead, it gives a single loss and it will be backpropagated on the networks, separately. This manner of backpropagating is problematic because we have two objectives, one quality of generated data and the other modes coverage. Even though GANs performed well on the quality of generated data, mode collapsing was strongly possible in GANs. Bi-GANs were introduced to address the mode collapsing problem in GANs. Bi-GANs outperformed GANs in terms of covering all modes of data, but the goal was not achieved perfectly due to the additive loss backpropagation which is explained above.

3.2.4 Boosting Generative Adversarial Networks

Formerly, we talked about Bidirectional Generative Adversarial Networks. The aftermath of using an encoder on the generator side, the discriminator was equipped to make stronger and more reliable results. The reason was because of extra information which was provided via passing real samples throughout the encoder. Despite many contributions to improving accuracy and realism perceptuality generated samples by Bi-GANs, still they had some problems which could be improved in terms of artificial data quality.

The researchers from the Max Planck Institute offered a new model that helped Bi-GANs to improve more the realization of the new sample generation process [62]. The goal was achieved by implementing the discriminator network based on the Boosting technique as a type of Ensemble Learning techniques, thus, if a generated sample was misclassified as real one according to any discriminator, the discriminator would give higher weights for it to pass again through the discriminator until it could catch the features of the generated sample, such that it should not be misclassified. As a result, the Ada-GAN model takes advantage of the AdaBoost technique to improve the accuracy and performance of a discriminator.

When we claim a discriminator is improved more and it is reliable more, it would lead to the point that better performance of the discriminator would help the generator as well. As elaborative our discriminator is, the more curious and comprehensive the generator would be on sample generation and data distribution support. Note that usage of the AdaBoost technique on any discriminator is possible and independent of architecture. The only point behind it is turning a single discriminative model to a meta-model that discriminates between the artificially generated samples and the real samples. In the [62], it is demonstrated mathematically and empirically that using the boosting technique on the discriminator network would result in more qualified and real-looking artificially generated data.

3.2.5 Cycle-GANs

In 2018 Jun-Yan Zhu et al. [63], from Berkley Artificial Intelligence Research (BAIR) Lab introduced a new type of generative model which aimed to generate new data via a cyclic architecture which is referred to as Cycle-GANs. Unlike conventional generative models which generate new artificial data from scratch (e.g. based on the desired distribution, etc.) Cycle-GANs transform existing real data into an artificially generated one. In their proposed model, an adversarial network is used to map X to Y (F) meanwhile another adversarial network is used to map Y to X (G). Their basic premise is if the generator which transforms X to Y and the other generator which transforms Y to X are performing well, in that case, $X \approx G(F(X))$ and $Y \approx F(G(X))$. In order to achieve this goal, they utilize two types of loss functions (i.e., adversarial loss, and cycle-consistency loss.)

Adversarial loss is calculated for both the F and G networks as explained earlier. That is to say, from the adversarial game between the two networks (i.e., generator and discriminator), such that the generator network aims to generate samples as similar as possible to the real samples, and the discriminator targets to differentiate the generated samples from the real data. The sum of the adversarial loss from both networks is the adversarial loss component of the cost function in the Cycle-GANs.

Despite adversarial loss alone can bring up outputs which have a distribution close to the distribution of the target domains (i.e., from X to Y and from Y to X), a generator (transformer) network per iteration can map the same input data to some different permutations with the same loss. That is to say that adversarial loss has no intuition to avoid this arbitrary transforming, and the Cycle Consistency Loss (CCL) is proposed as a solution to address this arbitrary matching between data. Thus, a CCL is introduced as another component to the loss function of Cycle-GANs which is $[F(G(X))-X] + [G(F(Y))-Y]$, so that better-performing F and G can be determined.

In our proposed model, we also used a similar idea, however in with a different approach which will be clarified in Chapter 4.

3.2.6 Mode Collapse and Missing Mode

In basic GANs, when a latent set(i.e., a sample from the desired distribution) is given to the generator in order to generate new data, feasibly, the generator would adapt to a specific mode of the data distribution and it would try to generate new samples specifically from those specific modes- the generator is adapted to some modes more than others.

Insisting only on few modes will cause the discriminator to become enough adept on distinguishing generated and real samples corresponding to the specific modes. Subsequently, the discriminator will be less susceptible to be fooled in that mode of data. Consequently, the generated data may have a perfect quality, but the GAN suffers from distribution mismatching on real data because there is no guarantee that the generator will not repeat to map different code modes to an identical mode on real data. This problem is known as Mode Collapse. As a result, the generator may give up.

Using boosting technique on the discriminator part [62], Guided Latent Spaces technique which performs the weighting in reverse direction on data distribution [64], Pacing technique on GANs which is referred to as PacGAN that segment data into smaller parts to feed into GANs [65], using some specific loss functions like Earth Mover distance [66] are offered as solutions to overcome mode collapse.

Mode collapse takes a role in another problem referred to as the Missing Mode problem. Missing Mode happens when the generator deals with some modes of data while misses others. In other

words, the generator does not generate new samples from different modes of data consistently, we would have the missing mode problem.

3.2.7 Variational Encoder Enhancement to Generative Adversarial Networks

Mode Collapse is a problem associated with the generators of the GANs and can only be treated by setting the parameters of the generator so that the generated output does not miss a mode of the real data space (i.e., cause a more general missing mode problem). Variational Encoder Enhancement to Generative Adversarial Network is one of the proposed models to address the Mode Collapse problem.

The solution which VEEGANs [67] offer to address the missing mode problem is adding another network which is referred to as the *reconstructor network*. The Reconstructor network is basically an encoder that encodes the visible data to the desired distribution (i.e., the output *code* of the encoder). One of the objectives of the reconstructor network is ensuring the code of the encoder matches the desired distribution that is used by the generator to generate artificial data. The second objective of the reconstructor network is ensuring that the generator and the encoder work in tandem like an auto-encoder, but in this case, the generator acts as the encoder and the encoder (i.e., reconstructor network) acts as a decoder. That is to say, the second objective of the reconstructor network is minimizing the difference between the desired distribution which is fed to the generator, and the corresponding reconstructed distribution by the reconstructor. VEEGANs measures this difference by means of a L_2 - *norm*. The total loss function that trains the encoder (i.e., generator network) is the sum of these objectives which is presented in Equation (8). The first term in Equation (8) corresponds to the first objective mentioned above (where H is a measure that represents how a distribution is different from another, e.g., KL-divergence) and the second term corresponds to the second objective.

$$\mathcal{O}(\gamma, \theta) = H(Z, F_{\theta}(x)) + E \left[\|z - F_{\theta}(G_{\gamma}(z))\|_2^2 \right] \quad (8)$$

The generator is trained in the same way as the conventional GANs. That is to say, the output of the discriminator trains the generator with a subtle but significant difference though. The discriminator in VEEGAN tries to differentiate a vector which jointly consists of the real data and reconstructed code and another vector which jointly consists of artificial data and sample from the desired distribution. Note that in conventional GANS the discriminator was trying to differentiate only the real data from the artificial data. That is to say, in VEEGAN the discriminator does not only check the performance of the generator but also the reconstructor network (i.e., encoder) as well. Furthermore, since the input vector of the discriminator also involves the reconstructed

code/sample from the desired distribution the generator training is different from the conventional GANs and also has to take care of the match between the reconstructed code of the encoder and the sample from the desired distribution. This implies a training that avoids mode collapse to a degree. Figure 27 provides the details of the VEE-GAN structure.

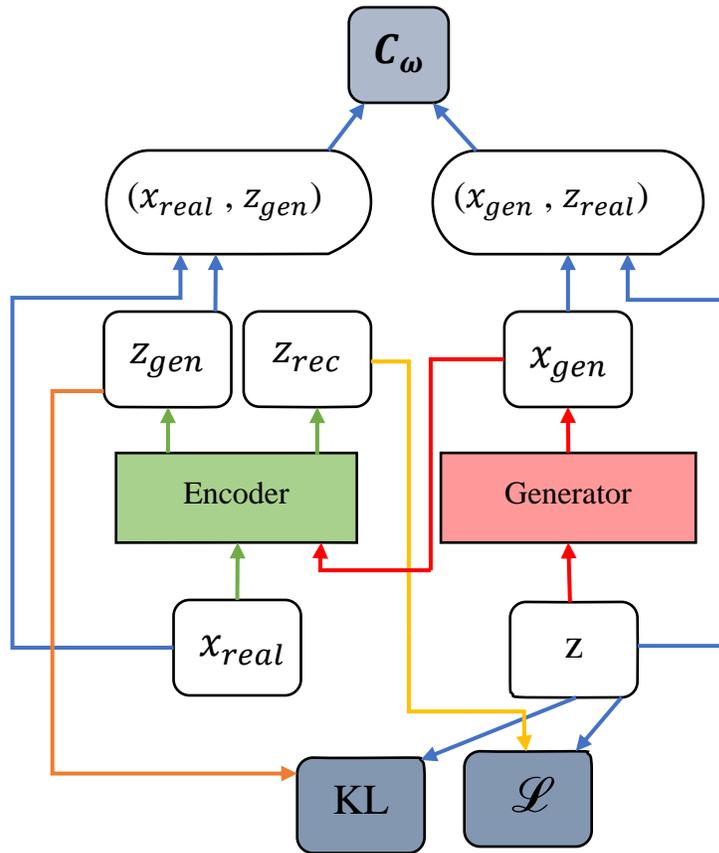


Figure 27: Variational Encoder Enhancement Generative Adversarial Networks (Figure is adapted from [67])

Now, let us discuss how VEEGANs deal with the Mode Collapse problem in detail. Recall that the performance of the model from the encoder point of view is based on achieving two objectives simultaneously; ensuring the reconstructor network - encoder- is a good approximate inverse of the generator, and be trained to match the true data distribution to a desired (e.g. Gaussian) distribution appropriately.

The encoder takes the real input data and spits output supposedly from the desired distribution, say Gaussian. Consider the first objective, i.e., the encoder is trained to be inverse of the generator approximately (i.e., the second term in Equation (8)). Let us assume a case where the generator-

encoder is doing a good job in this aspect (that is to say encoder is a good inverse of the generator) yet the mode collapse happens in the generator (Scenario 1). Consequently, due to its objective function, the encoder will be trained to spit out the desired distribution even for this situation when the input is from the generated data with specific modes are missing. That is to say, in order to maintain its performance in terms of the first objective, the encoder should start sacrificing from the second one and trains itself accordingly. Thus the output of the reconstructor network (i.e., encoder) will be distorted (due to this training) and start missing some modes of the desired distribution in the reconstructed distribution (i.e., the modes which the generator missed) when fed with the real data. That is to say, the output of the encoder will not be from the desired distribution anymore when fed with the real data.

Figure 28 depicts the representation of this case where real data is a mixture of two Gaussian distributions, but the generator has mode collapse and covers only one (i.e., not complete coverage of all modes) of the modes in the real data. Therefore, the output of the encoder (i.e., since perfectly inverses the generator) will also collapse in the collapsed modes of the generator and start spitting out a distorted distribution when fed with the real data. Thus, through the feedbacks of the discriminator (which is based on the comparison of the joint input vectors, i.e., [real data; encoder output code] vs [generated data; desired distribution]), the generator starts behaving appropriately due to the adversarial game (i.e., avoids the mode collapse problem) among the two networks.

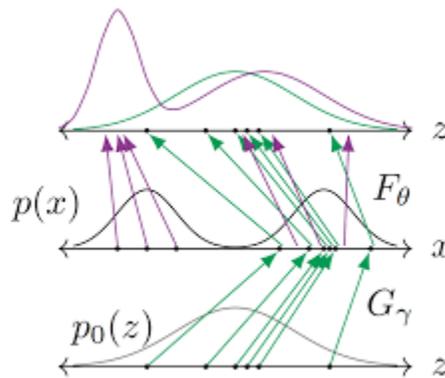


Figure 28: How the reconstructor network in VEEGANs behaves in Scenario 1 (Figure is taken from [67])

Now let's focus to the second objective (i.e., the first term in Equation (8)) and consider a scenario in which the reconstructor network (no matter what the input is) matches its output to the desired distribution say, Gaussian distribution, very well yet there is a mode collapse in the generator (Scenario 2). When the generator collapses in a specific mode/s, this time encoder should start

sacrificing from the first objective (i.e., to ensure that the encoder to be approximately inverse of the generator) in order to maintain its excellence in terms of the second objective. That is to say, it is trained so that, when fed with the output of the generator (i.e., the generated/artificial data), the output of the encoder doesn't match with the input (desired) distribution of the generator. As a result, eventually, the second objective forces the encoder to change, which in turn distorts the output of the encoder like Scenario 1 and the discriminator starts sending a signal to the generator to change its behavior and avoid mode collapse. Figure 29 demonstrates how VEEGAN behaves in Scenario 2.

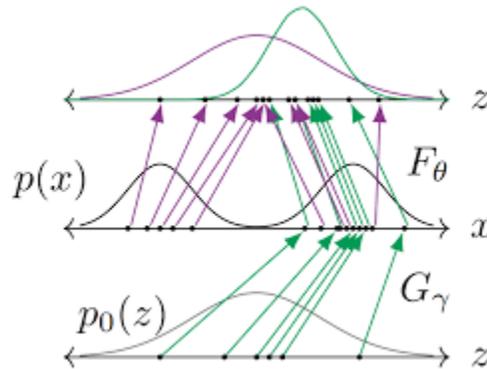


Figure 29: How the reconstructor network in VEE-GAN behaves in Scenario 2 (Figure is taken from [67])

3.2.8 Wasserstein Generative Adversarial Networks

Arjovsky et al. [66], introduced a new GAN based on a distance metric called Wasserstein distance. The Wasserstein metric is introduced by Russian American mathematician Leonid Vaserstein in 1969 [68], such that it is a metric to compare probability distributions of two variables X and Y , where one variable is derived from the other by some perturbations. In Computer Science literature it is also referred to as the *Earth Mover's distance* (EM). If one considers each probability distributions as different heaps of a certain amount of dirt, EM distance corresponds to the *minimum* total amount of work moving a pile of dirt from a heap to the other distribution, so that the former distribution eventually matches with the latter one. The work is calculated by the product of the amount of dirt and the moving distance. Since many possible transport plans can be used to match the distributions, the minimum one is chosen as the EM, a.k.a. the Wasserstein distance between the two distributions.

In the case of continuous probability distributions, the Wasserstein distance is determined as in Equation (9).

$$W(P_r, P_g) = \inf_{\gamma \in \mathbb{I}(P_r, P_g)} E_{(x,y) \sim \gamma} [\|x - y\|] \quad (9)$$

Where in the Equation (9), $\mathbb{I}(P_r, P_g)$ denotes the all possible joint probability distributions $\gamma(x, y)$, whose marginals are P_r and P_g . One can interpret each $\gamma \in \mathbb{I}(P_r, P_g)$ as one dirt transport plan in the framework of the Earth Mover's distance. In this context, $\gamma(x, y)$, corresponds to the amount of dirt that should be transported from the source x to the destination y which have a distance $\|x - y\|$ between them. Thus, the cost of such a move would be equal to $\gamma(x, y) \cdot \|x - y\|$. If one double sums such costs for all (x, y) pairs, can calculate the total cost associated with the transport plan $\gamma(x, y)$. Therefore, the expectation term in Equation (9), i.e., the probability of $\gamma(x, y)$ multiplied by the distance of $\|x - y\|$, corresponds to the total cost calculation for each possible γ . Finally, we need to find the minimum one among the costs of all transportation plans. That is to say, the cost of the *optimal plan* equals to the distance between the two distributions, i.e., $W(P_r, P_g)$.

Calculating the Wasserstein distance itself is a constrained optimization problem and practically, it is not feasible to calculate all possible joint distributions $\gamma \in \mathbb{I}(P_r, P_g)$. These intractable computations on all possible joint probability distributions can be transformed into tractable ones, based on Kantorovich-Rubinstein duality. This transformation in a sense decomposes Equation (9) and leads to Equation (10) where the intractable joint distributions are eliminated.

$$W(P_r, P_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} E_{x \sim p_r} [f(x)] - E_{x \sim p_g} [f(x)] \quad (10)$$

In Equation (10) a condition is imposed on the f , i.e., $\|f\|_L \leq K$, which implies that the functions are K-Lipschitz continuous. Note that, a real-valued function f is called to be K-Lipschitz continuous if there exists a real constant $K \geq 0$ such that, for all $x_1, x_2 \in \mathbb{R}$,

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2| \quad (11)$$

Therefore, the optimization problem now becomes finding the optimal f that is K-Lipschitz continuous.

Arjovsky et al. [66], considers using a NN to determine the optimal f with the following loss function:

$$L(P_r, P_g) = W(P_r, P_g) = \max_{\omega \in W} E_{x \sim P_r} [f_{\omega}(x)] - E_{z \sim P_r(z)} [f_{\omega}(g_{\theta}(x))] \quad (12)$$

Where $g_{\theta}(x)$ is considered as the output of the generator network and the second network (i.e., f_{ω} , or the *critic*) calculates the distance between the generated distribution and the real distribution by means of the Wasserstein distance metric. Recall that in conventional GANs the discriminator spits out a probability that could be used to differentiate the real and the fake data. However, the output of the second network in WGAN is not a probability but rather a Wasserstein distance between two distributions. Therefore, in the context of WGAN, the second network is not referred to as discriminator (since it does not discriminate) but rather as a *critic*.

As the loss function of the *critic* decreases, this means that the Wasserstein distance between the real distribution and the generated data decreases. Likewise, as the generator starts generating outputs that are closer to the real distribution, the Wasserstein distance, i.e., the loss function of the *critic* decreases.

While the objective of the critic is learning the Wasserstein distance between the two probability distributions, P_r and P_g (which is finding the f_{ω} that *maximizes* the distance between the real distribution and the distribution of the output of the generator due to the loss function), the objective of the generator is the opposite, i.e., *minimizing* the distance. Thus, there is an adversarial game with the overall loss function is as follows:

$$\min_{g_{\theta}} \max_{\omega \in W} E_{x \sim P_r} [f_{\omega}(x)] - E_{z \sim P_r(z)} [f_{\omega}(g_{\theta}(x))] \quad (15)$$

One of the issues that should be addressed in WGANs is preserving the K-Lipschitz continuity of the network during the training process. One way of achieving it is to clip weight vectors of the network f_{ω} (i.e., the discriminator network in WGAN) to a small window like $[-0.01, +0.01]$.

As we explained in previous chapters, one of the main problems of the GANs is the mode collapse problem. According to Arjovsky et al. [66], as a result of applying the Wasserstein loss function, there is a significant degree of decrease in mode collapse. In addition, they have indicated that WGAN is more robust against the vanishing gradient problem [66].

Despite the fact that the WGAN made valuable contributions to conventional GANs, it has some problems like unstable training, slow convergence (especially when the weight clipping window is too large), and still to a degree vanishing gradient (particularly when the weight clipping window is too small). [69]

3.2.9 Variational Auto-Encoder Generative Adversarial Networks VAE-GAN or VGH/VGH++

In terms of the quality of the generated data, GANs have demonstrated superior potential compared to other models. However, GANs do not perform well at covering all modes of the real data; consequently, they might generate only specific mode/s of data. On the other hand, VAEs performing well on covering all modes, but their outputs do not have the same quality as generated data by GANs [70]. Therefore, hybrid approaches (i.e., hybrid of GANs and VAEs) that exploit the strength of both algorithms received attention from researchers. Among them, Rosca et al. (2019) recently introduced two hybrid models namely, VGH and VGH++ which are slightly different from each other.

VGH/VGH++ are similar to the VEE-GANs. However, the generator of the VGH/VGH++ is trained both by feedbacks of the discriminator and the autoencoder networks, unlike VEE-GANs where the generator directly depends solely on the feedback of the discriminator. Furthermore, in the frame of VGH/VGH++, there are two separate discriminators (one as code discriminator and the other one as data discriminator), but in VEE-GANs one discriminator handles both the tasks jointly.

In terms of the encoder network, there are also differences between VGH/VGH++ and VEE-GANs. In VEEGANs we have a L_1 - *norm* reconstruction loss to capture any possible code diversity between the real code and the reconstructed code (i.e., reconstruction on the generator's output) in addition to using a KL-divergence on the distribution of real code space and generated code space. However, VGH/VGH++ has a generation loss in the code space which is handled via a code discriminator along with reconstruction loss in the visible space (i.e., the cost associated with autoencoder network) to train the encoder. That is to say, the VGH/VGH++ has an independent discriminator to evaluate the equality of generated latent space and the real latent space (i.e., say Gaussian distribution) as part of its training process. Note that the reconstruction loss is separately fed to the encoder and generator (which acts as a decoder), unlike the conventional autoencoder which is fed only to the decoder during the training.

Finally, the generator in VEE-GANs is trained only with the adversarial game with the discriminator. However, in VGH/VGH++ the reconstruction loss of the autoencoder is also used in addition to the generation loss during the training which is handled via the data discriminator. The subtle difference between VGH and VGH++ is at the generation loss training stage. Figure 30 presents a VGH++ structure.

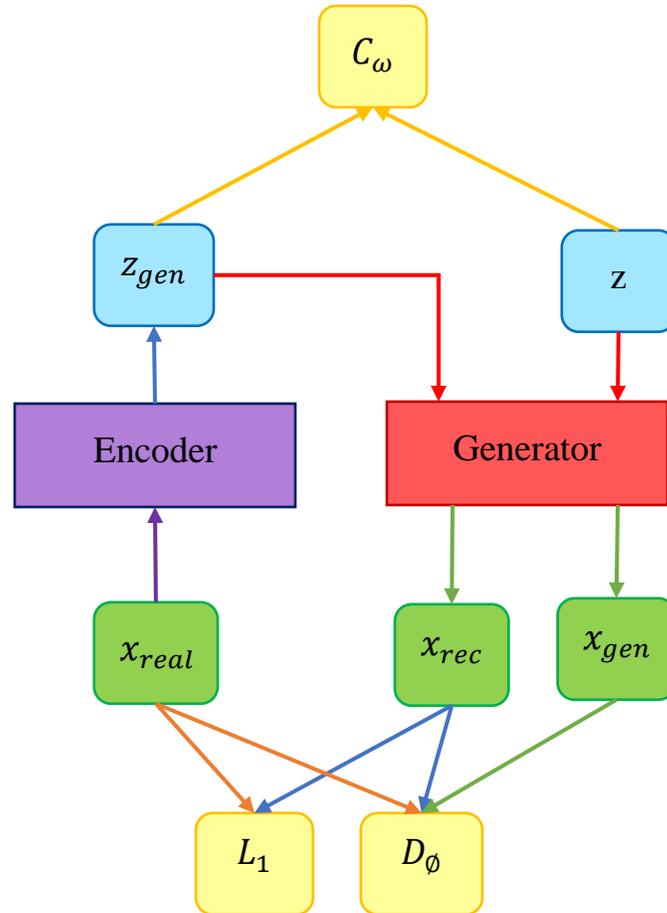


Figure 30: Variational Auto-Encoder Generative Adversarial Network – VAEGAN and VGH++ (Figure is adapted from [3])

Two variants namely VGH and VGH++ differ from each other in generation loss calculation. The generation loss in VGH is determined by the data discriminator which tries to differentiate the reconstructed data (i.e., data generated based on the encoder’s output) and the real data. However, the generation loss in VGH++ is a joint term of the reconstructed data with the real data and the generated data with the real data. That is to say, the VGH does not pass any generated data to the data discriminator, while the VGH++ feeds it to the discriminator as well. Consequently, Figure 30 belongs to VGH++, and if we remove the bottom right-most arrow from generated data toward the data discriminator, it will represent the VGH.

Variational Auto-Encoder and Generative Adversarial Network Hybrid model could not achieve expected goals in terms of the generated data quality and likelihood (according to the Rosca et al. [3]). In fact, basic GANs are performing better than the hybrid models in terms of the quality of the generated data. However, the motivation behind the hybrid models was to address the mode collapse problem along with preserving the quality of generated data. Meanwhile, the hybrid model

performed better than basic GANs in terms of various modes covering, however, it was achieved by the cost of decreasing the quality of the generated data.

A recent variant of VGH/VGH++ is proposed which is referred to as Best of Many Samples. According to their point of view, the encoder is restrained and any divergence from the desired distribution in generated latent space is penalized (as of the configuration in the VAE-GAN model). Consequently, the encoder is supposed to make a balance between the quality of the generated code likelihood and divergence in the code space (i.e., supposedly to cover all modes in code space). According to their proposed solution, for one real data fed into the encoder, multiple codes will be generated (i.e., multiple times fed to the encoder) and the one with the best convergence to the desired distribution will be fed into the generator to reconstruct the visible data. Additionally, any change in the output of the encoder could have a large impact on the performance of the generator and discriminator. That is because the discriminator is supposed to produce likelihood for both the generated and reconstructed data which in the case of any instability (i.e., un-convergence of the encoder's output to the desired distribution), it can cause the generator and discriminator to deviate from their appropriate form.

In Chapter 4 we have proposed a new model, to address both problems (i.e., mode collapse problem and quality of generated data). We have taken the strengths of the Variational Auto-Encoder and Generative Adversarial Networks model into account, and also we have used a cyclic way of generating new data (i.e., like Cycle-GAN proposed by Zhu et al. [63]) to generate new data with perfect quality and covering of modes in both latent and visible space.

4. PROPOSED MODEL

In this study, a new hybrid model is proposed to address the two issues, namely, covering all modes and improving the quality of the generated data. The proposed hybrid model is a modified version of the Rosca et al. model [3] with some major differences. Our point of view is that by applying visible data reconstruction loss on both the generator and the encoder networks as suggested by Rosca et al. [3], it is possible that an underperforming generator would *directly* penalize the encoder network inappropriately due to the architecture of VGH/VGH++. Note that in a conventional AE, backpropagation is initiated from the decoder, however, in VGH/VGH++ it is performed on both the decoder (i.e., generator) and encoder separately.

Furthermore, in VGH/VGH++ there is a single autoencoder (i.e., the encoder-generator network, where the generator acts as the decoder) and the reconstruction loss is between real data and reconstructed data. In our model, there are two *implicit* autoencoders based on two coupled adversarial networks. The first implicit autoencoder acts like the autoencoder of VGH/VGH++ and the second one is where the generator acts as an encoder and the encoder acts as the decoder. In the second one, the reconstruction loss this time is determined based on the difference between the code of the encoder and the sample from the desired distribution. That is to say, in our architecture, the code of the encoder represents a sample from the desired distribution. Note that in our architecture the reconstruction loss for both of the *implicit* AEs is obtained from the adversarial network rather than the L_1 - norm as suggested by [3]. That is why the AEs are *implicit* in the proposed model.

Both the encoder and the generator are trained partially based on the separate adversarial network that they are part of (i.e., the first adversarial network consists of the encoder and the code discriminator and the second one consists of the generator and the data discriminator), and partially based on *reconstruction loss* which is determined separately as we discussed above. For both the encoder and the decoder, the overall loss function is the weighted sum of these two components.

For the encoder, the adversarial component of the loss function (i.e., *adversarial loss*) is obtained from a code discriminator, where a sample from the desired distribution (z), and the output of the

encoder (\hat{z}) when fed by a sample from real data (x) to it (i.e., $\hat{z} = En(x)$). The discriminator job is basically differentiating the z from the \hat{z} just like the case in GANs and it is trained based on its performance on this task. We use the loss function of the code discriminator as the adversarial component of the loss function that is used to train the encoder.

For the generator, the adversarial component of the loss function is obtained from a data discriminator, where a sample from the real data (x), and the output of the generator (\hat{x}) when fed by a sample from desired distribution (z) to it (i.e., $\hat{x} = G(z)$). The data discriminator job is basically differentiating the x from the \hat{x} . We use the loss function of the data discriminator as the adversarial component of the loss function that is used to train the generator.

As it was explained earlier in Chapter 3, the conventional GANs suffer from mode collapse problem. WGANs which are based on the Wasserstein distance between two distributions shown to be prone to the mode collapse problem. In order to determine the *reconstruction loss* component of the total loss function, we used the WGANs loss function in the proposed model. That is to say, both the encoder and generator networks are part of a second adversarial network.

These latter two adversarial networks differ from the former two adversarial networks (which are used to determine the adversarial loss functions as described above) not only due to the usage of the Wasserstein distance as the loss function (i.e., as of WGAN) but also due to the inputs of their *critics* (recall the discussion on WGANs for choosing *critic* as opposed to *discriminator*). Despite the former adversarial networks which are based on the generated data (the generated code in the case of the encoder) and the real data (and the real code in the case of the encoder) comparisons, the input to the *critics* of these latter adversarial networks are the real data (and the real code in the case of *code critic* that is associated with the encoder) and the *reconstructed* data, i.e., $\tilde{x} = G(En(x))$, (and the *reconstructed* code in the case of *code critic*, i.e., $\tilde{z} = En(G(z))$). Figure 31 demonstrates the details of the model.

To sum up, for the encoder, the second component of the loss function (i.e., the reconstruction loss component) is obtained from a Wasserstein code critic, which receives a sample from the desired distribution (z), and the output of the encoder (\hat{z}) when fed by a sample from generated data (\hat{x}) to it as input. The critic's job is basically determining the Wasserstein distance between the desired distribution and the distribution of the code of the encoder based on these inputs. Next, we use the loss function of the code critic as the reconstruction loss component of the total loss function that is used to train the encoder.

For the generator, the reconstruction loss component of the loss function is obtained from a Wasserstein data critic, which receives a sample from the real data (x), and the output of the generator (\hat{x}) when fed by a sample from generated code (\hat{z}) to it as input. The data critic's job is

basically determining the Wasserstein distance between the real data distribution and the distribution of the reconstructed data based on these inputs. We use the loss function of the data critic as the reconstruction loss component of the total loss function that is used to train the generator.

The objective functions for all the six networks are presented below. In the equations, D_w corresponds to the output of the code discriminator, D_r corresponds to the output of the data discriminator, C_w corresponds to the code WGAN, C_r stands for the output of the data WGAN, En refers to the output of the encoder, and G refers to the output of the generator.

Objective Function for Data Discriminator

$$[-\log D_r(x) + \log(1 - D_r(G(z)))] \quad (13)$$

Objective Function for Code Discriminator

$$[-\log D_w(z) + \log(1 - D_w(En(x)))] \quad (14)$$

Objective Function for Data WGAN

$$\left[E_{p(z)} [C_r(x) - C_r(G(En(x)))] \right] \quad (15)$$

Objective Function for Code WGAN

$$\left[E_{p(x)} [C_w(z) - C_w(En(G(z)))] \right] \quad (16)$$

Objective Function for Generator

$$E_{p(z)} \left[\alpha \left[[-\log D_r(x) + \log(1 - D_r(G(z)))] \right] + (1 - \alpha) \left[E_{p(z)} [C_r(x) - C_r(G(En(x)))] \right] \right] \quad (17)$$

Objective Function for Encoder

$$E_{p(x)} \left[\alpha \left[[-\log D_w(z) + \log(1 - D_w(En_w(x)))] \right] + (1 - \alpha) \left[E_{p(x)} [C_w(z) - C_w(En(G(z)))] \right] \right] \quad (18)$$

The proposed model aims to achieve the generation of high-quality data with complete coverage of the modes. Until now, we have explained the details of our model in terms of the component networks as well as the objective functions used to train the corresponding components. Another point that needs to be mentioned regarding the training of the data discriminator is the utilization

of Multiple Chance Enhancement Technique (MCET) on misclassified samples. To the best of our knowledge, MCET is a novel approach that we introduced in this research which basically trains the data discriminator by giving more chance on the samples which are misclassified. Next, we will explain MCET in detail.

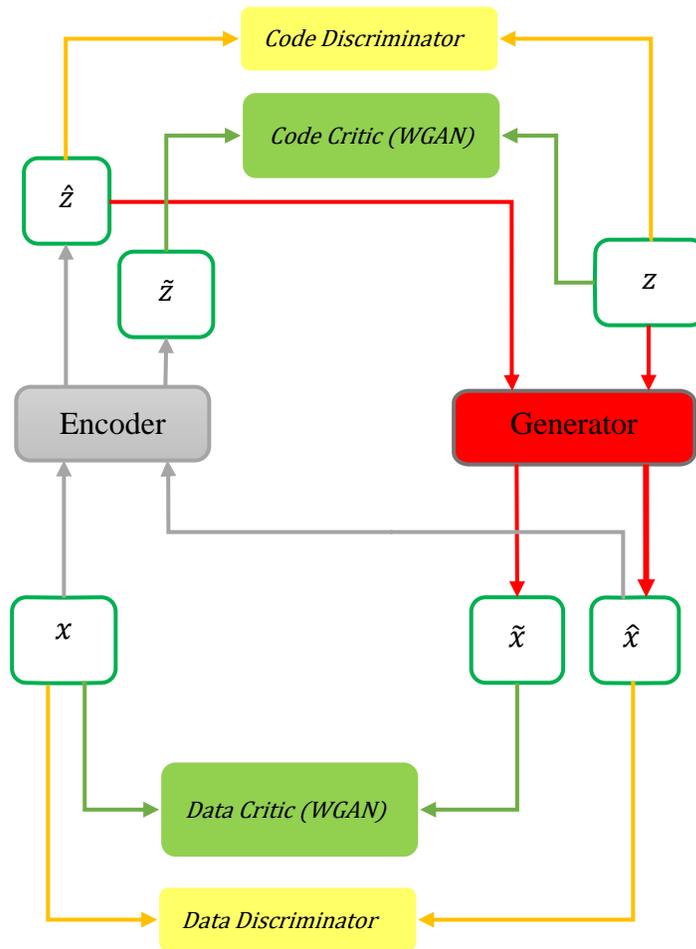


Figure 31: our proposed model

4.1 Multiple Chance Enhancement Technique (MCET)

After some pilot tests, we observed that the proposed model had some problems regarding generating data that are kind of at the boundaries among different classes (e.g., labels) of the real data. For example, in the case of the MNIST dataset, the generator could generate amorphous

hybrid outcomes which were neither 1 nor 7 but to a degree both. In order to address this problem, we focused on the training of the *data discriminator* of the proposed model.

The basic intuition is the classes that are similar in the real data also (most likely) have similar latent data. That's why the generator can't generate outputs that have sharp boundaries between different classes and might end up with the amorphous hybrid outputs which are not resembling any class that is available in the real data set. In order to force the generator not to generate such outputs, we targeted the data discriminator so that it can learn to differentiate these cases better and penalize the generator for such outputs.

As a matter of fact, these cases are also the ones where the discriminator had a hard time to differentiate one class from the other one even for the real data since they are indeed similar to each other and particularly *the outliers* of either class might start overlapping (e.g., 1 and 7). That is to say, in these cases, the discriminator is in doubt and cannot generate an "almost certain" probability of being real data (or fake data in the other case) and start making mistakes. MCET exploits this observation and addresses the problem of generating amorphous hybrid outputs.

In a nutshell, MCET determines the data samples (real or fake) which the discriminator assigns probabilities that are less than the batch average and feeds the discriminator again with the same data and enhances the training of the discriminator for these cases.

Initially, an approach that was inspired by AdaBoost was developed which aimed to penalize the misclassified cases in the loss function by increasing their weights and considered different epochs as different classifiers. However, quickly we have realized that this was not the right way to take. Next, we decided to enhance the learning of the discriminator by feeding the misclassified cases once again to the discriminator. In a sense, multiple chances were offered to those cases which were contributing to ending up with amorphous hybrid outputs from the generator. Thus, we referred to this approach as the Multiple Chance Enhancement Technique (MCET).

Selection of the real or fake images that would be given a second (or multiple) chances can be based on setting a threshold to the output of the discriminator which is basically a probability (typically 0.5). However, such a static threshold would not be effective very soon since the discriminator starts learning quickly. Therefore, we decided to use a dynamic approach in which those cases that have outputs less than the average of the batch are considered as those that are used once again for the training of the discriminator. Note that the task of finding the average is executed separately for the real and fake data. The effect of introducing MCET to the training of the data discriminator is presented in Chapter 5.

5. EXPERIMENTAL ANALYSIS AND DISCUSSIONS

We have applied our proposed model on the MNIST data set. As discussed earlier in Chapter 4, the overall objective of the proposed model is to generate data with high quality, which has a similar distribution with the real data and covers all modes of the real data distribution.

In our experimental analysis, we have used 1xTesla GPU with 12GB DDR5 RAM. Our objective function involves outputs of six networks, respectively: generator, encoder, data discriminator, data critic, code discriminator, and code critic.

The generator network has a 100-dimensional input vector size per sample from a Gaussian distribution. We have applied one fully connected layer as an input layer and four convolutional layers along with some batch normalizations. Leaky ReLU is the only activation function used in all layers except the last layer which we have used the Tanh function. The output of the generator network is a 28x28x1 image in the case where we have greyscale data.

We have applied four convolutional layers in the encoder with two dense layers (the first and the last layers). All convolutional layers have Leaky ReLU as activation functions, however, the first layer has sigmoid as its transfer function and the last layer applies a linear activation function. The output layer of the encoder is the same as latent data, in terms of dimensionality and batch size as well.

Data discriminator and data critic networks are all in the same form except the activation function applied in the last layers; the critic network applies a linear transfer function, but the discriminator has sigmoid as its activation function in the final layer. Both networks have five convolutional layers and one dense layer as the last layer all layers except the last layer use Leaky ReLU. Either of the networks takes an image as input and gives back a score for the realism perceptuality of the generator network. Note that, as it is explained in Chapter 3 to preserve Lipschitz continuity in NNs is required. Clipping weight vectors of the network is one way of doing that. We limit weight vectors of the two critic networks (i.e., data critic and code critic) to the domain of $[-0.01, +0.01]$.

Finally, the code discriminator and code critic networks taking generated and real latent space as inputs and giving feedbacks to the encoder network. Both networks have three convolutional layers

and two fully connected layers as the first and last layers. Like the case of real data, the code critic network has a linear activation function in the final layer while the discriminator applies sigmoid, and the remaining layers have a Leaky ReLU transfer function.

Thus, the encoder and generator networks are receiving feedbacks from the two networks. We have weighted them with α and $1 - \alpha$. Change in the weight combinations can make the generator/encoder to overestimate/underestimate one over the other network (i.e., discriminator and critic). In a sense by assuming α to be equal to 0.5, the same significance for both quality and mode coverage is given in the experiments.

Evaluating the quality of the outputs of GANs is not an easy task since it does not rely on an explicit model. Developing an objective, quantitative metric is still a fertile field of research. Generally, two proposed metrics that are widely used in the literature for measuring the performance of GANs are the Inception Score (IS) and the Fréchet Inception Distance (FID). Both of these metrics are obtained from a pre-trained image classifier (namely, Inception v3 [71]) which is used as a deep feature extractor. Activations of the output of the coding layer (which has a size of 2048) of the pre-trained image classifier are assumed to be multivariate Gaussian. Both the real and the generated images are fed to Inceptionv3 and by means of obtained activations, the parameters of the multivariate Gaussians are estimated for both real and generated images separately (i.e., respectively, $N(\mu_R, C_R)$ and $N(\mu_G, C_G)$, where μ is the mean vector with size 2048 and C is the Covariance matrix with size 2048×2048). Next, the inception scores are calculated from these estimates. IS calculation uses K-L Divergence and FID calculation is based on the Fréchet Distance [72] (which is also known as the Wasserstein-2 distance [68]).

Out of the two metrics FID is considered to be more reliable since in various applications optimizing the IS leads to adversarial examples and sensitive to minor perturbations in the network [73]. Therefore, we will consider FID as the metric to quantify the performance in our analysis. Yet we will also rely on a more conventional approach, that is to say, subjective visual analysis will also be considered.

Using FID as an objective, quantitative measure for GANs was introduced by Heusel et al. [71] in 2017. As explained above both for the generated and real data, Inception v3 network is used to determine the parameters of the multivariate Gaussian distribution that is assumed for the output of the coding layer. As a result, the FID of the generated outputs is calculated as follows:

$$d^2((\mu_G, C_G), (\mu_R, C_R)) = \|\mu_G - \mu_R\|^2 + Tr\left(C_G + C_R - 2(C_G C_R)^{\frac{1}{2}}\right) \quad (19)$$

In Equation (19), Tr stands for the trace of the matrix, i.e., the sum of the diagonal elements of the resulting matrix.

We conducted four different analyses. The first analysis questions the performance of the WGANs as the critics in the proposed model. We considered three possible scenarios: (1) Using WGANs as critics (i.e., the proposed model), (2) Model that use two extra discriminators rather than WGANs (i.e., four discriminators in total). The extra discriminators both have Cross-Entropy (CE) as the loss function, and (3) Model that use again two extra discriminators instead of WGANs (i.e., four discriminators in total), but this time the extra discriminators both have Mean Square Error (MSE) as the loss function. The second analysis focuses on the performance of the MCET algorithm. Two scenarios are considered: (1) Model without using MCET in the training of the data discriminator (2) Model that uses MCET. The third analysis is where the proposed method output is compared with various other alternatives (namely, VGH++, WGAN, and VAE) in terms of subjective visual analysis. Finally, the fourth analysis again compares the generated output of the proposed method with various other algorithms but this time the performance is quantified with the FID. Below we will discuss each one of the experiments as well as the results of the analysis in more detail.

In the first analysis, we tried to understand the performance of the WGANs as the critic in the model. In all the three scenarios, the Cross-Entropy (CE) is used as the loss function for the two original discriminators. Firstly, we introduced WGANs as the critics (both data and code critics) as proposed in Chapter 4. In the second scenario instead of WGANs, conventional GANs are used as the additional discriminator networks (i.e., both data and code critic) with CE loss functions. The third scenario is similar to the second scenario, but this time MSE loss function is used for the extra discriminators introduced instead of the WGANs. In all of the scenarios, the loss functions for the generator and the encoder are the weighted averages of the output of the associated discriminator and the loss function of the associated critics (or additional discriminators in the first and the third scenarios). In all the three scenarios the *implicit* autoencoders are part of the models, however, in each case, they have different loss functions. Figure 32 depicts the output of this analysis.

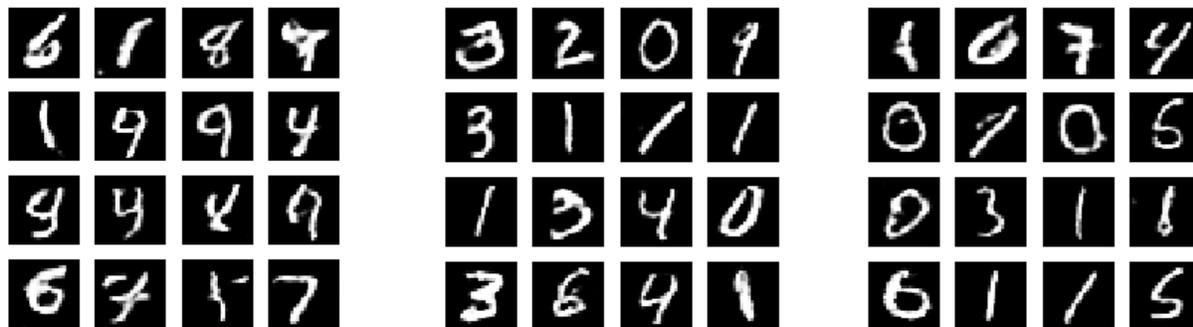


Figure 32: Generated Data for the Three Scenarios in the First Analysis

The left image in Figure 32 corresponds to Scenario 2 (i.e., all discriminators use CE as the loss functions). The middle image depicts the generated images for the case where we had the critics that used the Wasserstein distance as the loss function (i.e., Scenario 1). The right image depicts the generated images that corresponds to the case where the additional discriminators use MSE as the loss function (i.e., Scenario 3). A visual analysis indicates that the case where WGANs is used as critic networks have better quality compared to the other two cases. However, we should also note that the examples used in this analysis were the result of the generators that were trained only 50 epochs which are typically considered a very low value for the training process to reveal the actual performance of the generators. Even though, we applied FID on the results of the second and third scenarios and achieved the FID scores of (2nd scenario: 1094.080 and 3rd scenario: 3991.305) for the corresponding scenarios which convinced us to stick with the CE loss function for the discriminators.

The second analysis tries to capture the influence of the MCET during the training of the data discriminator. In Chapter 4, we explained details of MCET on the data discriminator network. Figure 33, presents a subset of the outputs generated by the proposed model without applying MCET (left) and with MCET (right).

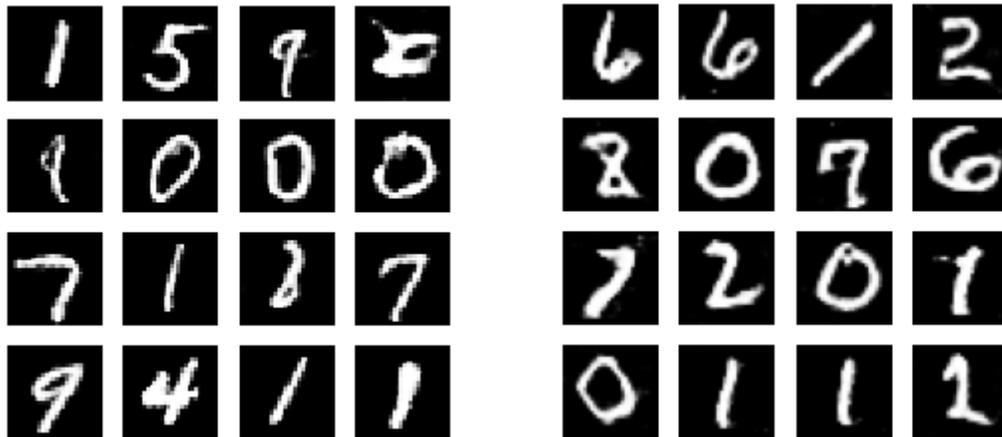


Figure 33: Left plot where MCET is not applied, the right plot is where we have applied MCET.

As we can observe in Figure 33, even in the case of a low number of epochs applying MCET results better quality for the generated data compared to the case where the MCET is not applied. In the case where we have MCET, not only the quality of generated samples is better than the other case, but also applying MCET affects the modes covering as well.

In the third analysis, we compared the output performance of the proposed model with the three other recently introduced models, namely, VGH++ introduced by Rosca et al. [3], the WGAN introduced by the Facebook Machine Learning Group [66], and the VAE [53].

In our proposed model the generator and encoder play a cycle simultaneously with the adversarial games that enable them to take care of quality and modes covering at the same time. In fact, the distribution analysis and quality analysis are performed in a cycle by the discriminators and the critics once on data and once on code. In other words, the discriminators and critics are set as the quality analyzers and modes covering, respectively. Despite the model contains a basic GAN configuration, it also applies the generated data into the cycle for the reconstruction of the inputs which aims to cover as many as possible modes of data, and the same process is done on code space.

To analyze our model based on the quality of generated data and various modes coverage, we compare with three different configurations which are VGH++, WGAN, and VAE models. Figure 34 presents the experimental results.



Figure 34: Respectively, from the left a collection of generated samples by our proposed model, VGH++, WGAN, and VAE (The last three figures are taken from [3])

As clearly demonstrated in Figure 34, our proposed model outperforms the WGAN, VGH++, and VAE models in terms of visual expression. The generated samples by our model have higher quality in comparison with the generated samples by three models. A worthwhile noting point is that our proposed model generates samples from various modes of data, which demonstrates that it is opposing mode collapsing.

We have compared our model with state of the arts on the MNIST dataset in terms of FID metric as well. Table 2 presents the results of comparisons in detail. In Table 2, FID values of the algorithms which are marked with * are taken from [74].

Table 2: Fréchet Inception Distance scores

Method	FID
Fréchet-GAN*	21.22
OT-GAN*	13.13
SWG*	17.16
Our model	8.52
WGAN*	18.86
WGAN-GP*	20.35
Max-SWG*	38.63

Our experiments clearly show that our proposed model could pass both factors very well, such that, quality of generated data is similar to the real data, and also our model could generate new samples from all modes of data (i.e., avoid mode collapsing problem and consequently missing mode problem). One thing that might be worth to be mentioned in this analysis is how FID of the proposed method decreases with respect to the number of epochs. For 5k generated images FID values after the first epoch was equal to 821.677 which later dropped to 8.52 after the 3000th epoch.

6. CONCLUDING REMARKS AND FUTURE WORK

For decades ML was limited to the tasks in which the main idea of training a learner was to enable it to act in a way that humankind refers to it as *intellectual behaving*. The introduction of GANs entered the realm of ML into a new era, which enabled ML to create data that it does not exist in the real world. From the day GANs are introduced, their significant capability in generating new data has given a significant capacity to them almost in all branches of computer science. However, generativity was not defined by GANs for the first. Before GANs, AEs and VAEs also gave remarkable developments to the sovereign state of ML.

In terms of the quality of the generated data and realism perceptuality, GANs outperformed VAEs. In fact, generated data through the VAEs are blurred. However, this problem of VAEs could not lower the amount of attention on them after the introduction of GANs. Since their capability in generating enough distributive data coincided with the weakness of GANs in covering various modes on generated data. As a result, scientists saw their (i.e., GANs and VAEs) weaknesses and strengths as a complementary way of generating new data, which in turn led to VAEs and GANs hybrid models.

Recently, new hybrid models are introduced which aimed to address the problems of VAEs and GANs simultaneously. Variational Encoder Enhancement Generative Adversarial Networks (VEEGANs) were introduced to address the problems which the GANs suffer. In order to increase the diversity of the generated data, VEEGANs aim to solve the mode collapse problem by combining strengths of Variational Auto-Encoders (VAEs) (i.e., *diversity*) with GANs (i.e., *quality*). However, VEEGANs have also some weaknesses. One of the most important problems related to VEEGANs is the backpropagation of error rate in visible space to both the generator and the encoder networks which in turn may lower the various modes covering capability of VEEGANs.

The problems related to VEEGANs were motivations toward introducing VAEGANs. Despite VEEGANs, VAEGANs have two discriminators: i.e., the main discriminator as VEEGANs, secondly a discriminator in the generator side to discriminate the generated latent space with real latent space which is called as code discriminator instead of the linear distance function that VEEGANs use in the code space. The code discriminator is involved in an adversarial game with

the encoder such that the encoder tries to fool the code discriminator by generating latent space which is as similar as possible to the desired distribution, and at the same time, the discriminator tries to be as precise as possible to find out any differences between the two latent spaces.

In this research, we proposed a new model that consists of two adversarial networks which are in both the visible space and the latent space. Our experimental analyses and comparisons demonstrated that our proposed model could address the two problems in a worthwhile way. A comprehensive analysis on the effectivity of weight clipping as a way of preserving K-Lipschitz continuity, and its impact on the critic networks performance is left as the future work.

Bibliography

- [1] J. Boyd, "DESTRUCTION AND CREATION," *US Army Comand and General Staff College*, September 1987.
- [2] F. Rosenblatt, "The Perceptron—a perceiving and recognizing automaton," 1957.
- [3] M. Rosca, B. Lakshminarayanan and S. Mohamed, "Distribution Matching in Variational Inference," *arXiv preprint arXiv:1802.06847*, 2019.
- [4] L. P. Kaelbling, M. L. Littman and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of artificial intelligence research*, vol. 4, pp. 237--285, 1996.
- [5] X. Zhu, "Semi-Supervised Learning Literature Survey," 2005.
- [6] X. Zhai, A. Oliver, A. Kolesnikov and L. Beyer, "S4L: Self-Supervised Semi-Supervised Learning," 2019.
- [7] D. Opitz and R. Maclin, "Popular Ensemble Methods: An Empirical Study," *Journal of artificial intelligence research*, vol. 11, pp. 169--198, 1999.
- [8] E. B. Bootstrap, "Methods: Another look at the Jackknife," *Ann. Statist*, vol. 7, pp. 1--26, 1979.
- [9] "Bagging – building an ensemble of classifiers from bootstrap samples".
- [10] M. Kearns, "Thoughts on Hypothesis Boosting.," 1988.
- [11] M. Kearns and L. G. Valiant, "Cryptographic limitations on learning Boolean formulae and finite automata," *Journal of the ACM (JACM)*, pp. 67--95, 1994.
- [12] L. Breiman, "Bias, variance, and arcing classifiers," Tech. Rep. 460, Statistics Department, University of California, Berkeley, 1996.

- [13] W. W. Cohen and V. R. Carvalho, "Stacked Sequential Learning," 2005.
- [14] S. Paul, "Ensemble Learning — Bagging, Boosting, Stacking and Cascading Classifiers in Machine Learning using SKLEARN and MLEXTEND libraries," Nov 2018. [Online]. Available: <https://medium.com/@saugata.paul1010/ensemble-learning-bagging-boosting-stacking-and-cascading-classifiers-in-machine-learning-9c66cb271674>.
- [15] Y. Freund, "An adaptive version of the boost by majority algorithm," *Machine learning*, vol. 43, pp. 293--318, 2001.
- [16] L. Breiman, "Arcing the edge," Technical Report 486, Statistics Department, University of California, 1997.
- [17] A. Torralba, K. P. Murphy and W. T. Freeman, "Sharing visual features for multiclass and multiview object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, pp. 854--869, 2007.
- [18] A. Demiriz, K. P. Bennett and J. Shawe-Taylor, "Linear programming boosting via column generation," *Machine Learning*, vol. 46, pp. 225--254, 2002.
- [19] T. Chen and C. Guestrin, "Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining," 2016.
- [20] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush and A. Gulin, "CatBoost: unbiased boosting with categorical features," 2018.
- [21] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in neural information processing systems*, 2017, pp. 3146--3154.
- [22] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, pp. 241--259, 1992.
- [23] J. Rocca, "Ensemble methods: bagging, boosting and stacking," Apr 2019. [Online]. Available: <https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205>.
- [24] "Tensorflow Single Layer Perceptron," 2020. [Online]. Available: https://www.tutorialspoint.com/tensorflow/tensorflow_single_layer_perceptron.htm.
- [25] R. Cassani, "Multi-Layer Perceptron," [Online]. Available: <https://github.com/rcassani/mlp-example>.

- [26] I. Changhau, "Activation functions and when to use them," [Online]. Available: <https://patrickhoo.wixsite.com/diveindatascience/single-post/2019/06/13/Activation-functions-and-when-to-use-them>.
- [27] Y. LeCun, P. Haffner, L. Bottou and Y. Bengio, Object recognition with gradient-based learning, Springer, 1999.
- [28] V. Stojov, N. Koteli, P. Lameski and E. Zdravevski, "Application of machine learning and time-series," April 2018.
- [29] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks," [Online]. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.
- [30] J. Hopfield, "Neural networks and physical systems with emergent collective," vol. 79, pp. 2554--2558, 1982.
- [31] nerdthecoder, "Neural Networks," 28 January 2019. [Online]. Available: <https://nerdthecoder.wordpress.com/2019/01/28/neural-networks-simplified-version/>.
- [32] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, pp. 1735--1780, 1997.
- [33] X. Liang, X. Shen, J. Feng, L. Lin and S. Yan, Semantic Object Parsing with Graph LSTM, Singapore: Springer, 2016.
- [34] M. Phi, "Illustrated Guide to LSTM's and GRU's: A step by step explanation," 2018. [Online]. Available: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>.
- [35] S. Rathor, "Simple RNN vs GRU vs LSTM :- Difference lies in More Flexible control," June 2018. [Online]. Available: <https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57>.
- [36] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," 2016.
- [37] S. Fan, "Understanding Deep Residual Networks," November 2018. [Online]. Available: <https://shuzhanfan.github.io/2018/11/ResNet/>.

- [38] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "LEARNING INTERNAL REPRESENTATIONS," *INSTITUTE FOR COGNITIVE SCIENCE*, vol. 323, pp. 533--536, September 1985.
- [39] "Auto-Encoder, Basic Architecture," July 2020. [Online]. Available: <https://en.wikipedia.org/wiki/Autoencoder>.
- [40] D. Charte, F. Charte, S. Garcia, M. J. del Jesus and F. Herrera, "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines," *Information Fusion*, vol. 44, 2018.
- [41] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction. In Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis," 2014.
- [42] A. Zhavoronkov, Y. A. Ivanenkov, A. Aliper, M. S. Veselov, V. A. Aladinskiy, A. V. Aladinskaya, V. A. Terentiev, D. A. Polykovskiy, M. D. Kuznetsov, A. Asadulaev and others, "Deep learning enables rapid identification of potent DDR1 kinase inhibitors," *Nature biotechnology*, vol. 37, pp. 1038--1040, 2019.
- [43] K. Cho, B. v. Merriënboer, D. Bahdanau and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches," *arXiv preprint arXiv:1409.1259*, 2014.
- [44] S. De, A. Maity, V. Goel, S. Shitole and A. Bhattacharya, "Predicting the Popularity of Instagram Posts for a Lifestyle Magazine Using Deep Learning," 2017.
- [45] R. Salakhutdinov and G. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, vol. 50, pp. 969--978, 2009.
- [46] J. Jordan, "Introduction to Auto-Encoders," March 2018. [Online]. Available: <https://www.jeremyjordan.me/autoencoders/>.
- [47] "Autoencoders with PyTorch," May 2018. [Online]. Available: <https://mc.ai/autoencoders-with-pytorch/>.
- [48] A. Makhzani and B. Frey, "k-Sparse Autoencoders," *arXiv preprint arXiv:1312.5663*, 2013.
- [49] P. Vincent, H. Larochelle, Y. Bengio and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," 2008.

- [50] "Programmersought.com," 24 08 2018. [Online]. Available: <https://www.programmersought.com/article/28301988742/>.
- [51] S. Rifai, P. Vincent, X. Muller, X. Glorot and Y. Bengio, "Contractive Auto-Encoders: Explicit Invariance During Feature Extraction," in *Icml*, 2011.
- [52] V. Palaniappan, "Stacked Neural Networks for Prediction," October 2018. [Online]. Available: <https://towardsdatascience.com/stacked-neural-networks-for-prediction-415ef3b04826>.
- [53] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv preprint arXiv:1312.6114*, 2013.
- [54] S. Harel, "Structured data Vision Team," Feb 2017. [Online]. Available: <https://www.slideshare.net/ShaiHarel/variational-autoencoder-talk..>
- [55] S. S. Borysov, J. Rich and F. C. Pereira, "Scalable Population Synthesis with Deep Generative Modeling," *arXiv preprint arXiv:1808.06910*, 2018.
- [56] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets," 2014.
- [57] C. Nicholson, "Generative Adversarial Network Definition, A.I.Wiki," [Online]. Available: <https://pathmind.com/wiki/generative-adversarial-network-gan>.
- [58] M. Santos, "Learning Generative Adversarial Networks (GANs)," Mar 2020. [Online]. Available: [https://mc.ai/learning-generative-adversarial-networks-gans/..](https://mc.ai/learning-generative-adversarial-networks-gans/)
- [59] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow and B. Frey, "Adversarial Autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.
- [60] s. selim, "Introduction to Adversarial Autoencoders," Jan 2019. [Online]. Available: <https://rubikscore.net/2019/01/14/introduction-to-adversarial-autoencoders/>.
- [61] J. Donahue, P. Krahenbuhl and T. Darrell, "ADVERSARIAL FEATURE LEARNING," *Published as a conference paper at ICLR 2017*, 2017.
- [62] I. O. Tolstikhin, S. Gelly, O. Bousquet, C.-J. Simon-Gabriel and B. Scholkopf, "AdaGAN: Boosting Generative Models," 2017.

- [63] J.-Y. Zhu, T. Park, P. Isola and A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks," 2017.
- [64] K. Kato, J. Zhou, S. Tomotake and N. Akira, "Rate-Distortion Optimization Guided Autoencoder for Isometric Embedding in Euclidean Latent Space," *arXiv preprint arXiv:1910.04329*, 2019.
- [65] Z. Lin, A. Khetan, G. Fanti and S. Oh, "PacGAN: The power of two samples in generative adversarial networks," in *Advances in neural information processing systems*, 2018.
- [66] M. Arjovsky, S. Chintala and L. Bottou, "Wasserstein GAN," *arXiv preprint arXiv:1701.07875*, 2017.
- [67] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann and C. Sutton, "VEEGAN: Reducing Mode Collapse in GANs using," 2017.
- [68] L. N. Vaserstein, "Markov processes over denumerable products of spaces, describing large systems of automata," *Problemy Peredachi Informatsii*, vol. 5, pp. 64--72, 1969.
- [69] L. Weng, "From GAN to WGAN," *arXiv preprint arXiv:1904.08994*, 2019.
- [70] A. Bhattacharyya, M. Fritz and B. Schiele, "'BEST-OF-MANY-SAMPLES' DISTRIBUTION MATCHING," *arXiv preprint arXiv:1909.12598*, 2019.
- [71] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in neural information processing systems*, 2017, pp. 6626--6637.
- [72] M. Frechet, "Sur la distance de deux lois de probabilite," *COMPTEs RENDUS HEBDOMADAIRES DES SEANCES DE L'ACADEMIE DES SCIENCES*, vol. 244, pp. 689-692, 1957.
- [73] P. Dimitrakopoulos, G. Sfikas and C. Nikou, "Wind: Wasserstein Inception Distance For Evaluating Generative Adversarial Network Performance," 2020.
- [74] K. D. Doan, S. Manchanda, F. Wang, S. Keerthi, A. Bhowmik and C. K. Reddy, "Image Generation Via Minimizing Frechet Distance in Discriminator Feature Space," *arXiv preprint arXiv:2003.11774*, 2020.
- [75] "Neural Network Toolbox The MathWorks," 1994-2005. [Online]. Available: <http://matlab.izmiran.ru/help/toolbox/nnet/percept4.html>.

- [76] T. Miyato, T. Kataoka, M. Koyama and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.
- [77] R. Khandelwal, "Deep Learning — Different Types of Autoencoders," December 2018. [Online]. Available: <https://medium.com/datadriveninvestor/deep-learning-different-types-of-autoencoders-41d4fa5f7570>.
- [78] S. C. Larson, "The shrinkage of the coefficient of multiple correlation.," *Journal of Educational Psychology*, vol. 22, 1931.
- [79] T. Lucas, K. Shmelkov, K. Alahari, C. Schmid and J. Verbeek, "Adaptive Density Estimation for Generative Models," in *Advances in Neural Information Processing Systems*, 2019, pp. 12016--12026.
- [80] V. Fortin, J. Bernier and B. Bobee, "Simulation, Bayes, and bootstrap in statistical hydrology," *Water Resources Research*, pp. 439--448, 1997.
- [81] Y. Freund, R. Schapire and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, p. 1612, 1999.