Novel Architectures and Optimization Algorithms for Training Neural Networks and Applications

Vasily I. Zadorozhnyy
University of Kentucky, vasily.zadorozhnyy@gmail.com
Author ORCID Identifier:
https://orcid.org/0000-0002-0379-7348
Digital Object Identifier: https://doi.org/10.13023/etd.2023.125

Novel Architectures and Optimization Algorithms for Training Neural Networks and Applications

DISSERTATION

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the College of Arts and Sciences at the University of Kentucky

By
Vasily I Zadorozhnyy
Lexington, Kentucky

Director: Dr. Qiang Ye, Professor of Mathematics
Lexington, Kentucky
2023

ABSTRACT OF DISSERTATION

Novel Architectures and Optimization Algorithms for Training Neural Networks and Applications

The two main areas of Deep Learning are Unsupervised and Supervised Learning. Unsupervised Learning studies a class of data processing problems in which only descriptions of objects are known, without label information. Generative Adversarial Networks (GANs) have become among the most widely used unsupervised neural net models. GAN combines two neural nets, generative and discriminative, that work simultaneously. We introduce a new family of discriminator loss functions that adopts a weighted sum of real and fake parts, which we call adaptive weighted loss functions. Using the gradient information, we can adaptively choose weights to train a discriminator in the direction that benefits the GAN's stability. Also, we propose several improvements to the GAN training schemes. One is self-correcting optimization for training a GAN discriminator on Speech Enhancement tasks, which helps avoid "harmful" training directions for parts of the discriminator loss. The other improvement is a consistency loss, which targets the inconsistency in time and time-frequency domains caused by Fourier Transforms. Contrary to Unsupervised Learning, Supervised Learning uses labels for each object, and it is required to find the relationship between objects and labels. Building computing methods to interpret and represent human language automatically is known as Natural Language Processing which includes tasks such as word prediction, machine translation, etc. In this area, we propose a novel Neumann-Cayley Gated Recurrent Unit (NC-GRU) architecture based on a Neumann series-based Scaled Cayley transformation. The NC-GRU uses orthogonal matrices to prevent exploding gradient problems and enhance long-term memory on various prediction tasks. In addition, we propose using our newly introduced NC-GRU unit inside Neural Nets model to create neural molecular fingerprints. Integrating novel NC-GRU fingerprints and Multi-Task Deep Neural Networks schematics help to improve the performance of several molecular-related tasks. We also introduce a new normalization method - Assorted-Time Normalization, that helps to preserve information from multiple consecutive time steps and normalize using them in Recurrent Nets like architectures. Finally, we propose a Symmetry Structured Convolutional Neural Network (SCNN), an architecture with 2D structured symmetric features over spatial dimensions, that generates and preserves the symmetry structure in the network's convolutional layers.

KEYWORDS: Deep Learning, Adaptively Weighted Discriminator Generative Adversarial Network, Self-Correcting Optimization, Orthogonal Gated Recurrent Unit with Neumann-Cayley Transformation, Assorted-Time Normalization, Symmetric Convolutional Neural Network

Vasily I Zadorozhnyy

May 1, 2023

Novel Architectures and Optimization Algorithms for Training Neural Networks and Applications

By
Vasily I Zadorozhnyy

Dr. Qiang Ye
Director of Dissertation

Dr. Benjamin Braun
Director of Graduate Studies

May 1, 2023
Date

Dedicated to my parents, grandparents, and my wife.

Посвящается моим родителям, бабушке с дедушкой и жене.

ACKNOWLEDGMENTS

Without the help of many people, the development of this thesis would not have been possible. Firstly, I would like to thank my advisor Dr. Qiang Ye for all his help, support, and wisdom. I would also like to thank my committee members Dr. Brown, Dr. Chen, Dr. Cheng, and Dr. Lu. Thank you for agreeing to serve on my committee and reading this manuscript.

To my parents, I have no words to describe my gratitude for your support, love, understanding, and willingness to help me in any situation. Your encouragement and guidance have been instrumental in shaping the person I am today. Thank you for your sacrifices, wisdom, and unwavering faith in me. Thank you Mom and Dad. I love you very much and miss you always.

To my wife, Becca, and my in-law family, I am grateful for your constant love, support, and understanding. I am grateful for your patience, as you have been there for me, even when I could not give you the time and attention you deserved. I am fortunate to have you in my life and blessed to be a part of your family.

To my host parents, Dr. Norma, Kevin, Wendy, and Dr. Ken, I would like to express my gratitude for your kindness, hospitality, and friendship. Your generosity and warmth made me feel welcome and supported, and I am blessed for the opportunity to be a part of your wonderful families.

To my friends, I am thankful for your love, encouragement, and friendship. Your unwavering support has been a source of motivation and strength, and I am fortunate to have had you by my side during this journey. Particularly I want to thank and mention Derek, Cole, Edison, Susanna, Ben, Will, Chase, Daniel, Samir, Gayan, Dr. Brennan, Marat, and Vladislav.

I cannot forget to thank my amazing puppy - Teddy, for always being willing to be my stress reliever during my Ph.D. journey.

Lastly, I want to thank the University of Kentucky Center for Computational Sciences and Information Technology Services Research Computing for their support and use of the Lipscomb Compute Cluster and associated research computing resources.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 Deep Learning Introduction

Deep Learning is a subset of machine learning, a field of Artificial Intelligence (AI) that trains machines (computers) to learn from data. Deep Learning does this by using artificial neural networks with structures and operations inspired by those of the human brain. These neural networks can evaluate enormous volumes of complex data and draw out patterns and insights, enabling machines to carry out previously thought to be the sole preserve of humans.

Deep learning has seen substantial progress in recent years, including innovations in areas such as speech and picture recognition, gaming, video rendering, computer vision, autonomous vehicles, natural language processing, etc. The availability of copious amounts of data and the processing capacity necessary to evaluate it have been some of the leading forces behind this development. Big Data and cloud computing have made it feasible to train neural networks that are more intricate and vast, which has increased the accuracy of the developed model and advanced and broader applications.

By enabling machines to automate operations and deliver insights that were difficult to gain, deep learning can potentially transform many industries, including healthcare, banking, transportation, communication, and advertising. Despite its achievements, Deep Learning is still an active field of study, and there are several issues that scientists are attempting to resolve, including creating more effective algorithms, addressing bias and ethical issues, and enhancing the interpretability of models. Deep Learning is developing quickly, and over the next several years, its influence on society will only increase.

1.1 Neural Networks

A form of machine learning model that mimics the human brain's composition and operation is called a neural network. They are made up of networked nodes called neurons that analyze data and provide predictions. Each neuron takes information from other neurons and then applies a nonlinear function to create an output sent back to those neurons. As a result, the network may run intricate calculations on the incoming data to provide predictions.

Neural networks can perform various tasks, including speech and picture recognition, natural language processing, and gameplay. They work particularly effectively for complex, multidimensional data applications, like audio or pictures. In recent years, neural networks have achieved amazing success, outperforming humans in numerous tasks and reshaping industries like computer vision and natural language processing.

The capacity of neural networks to learn from data is one of its main advantages. The network learns to generate predictions based on the patterns and correlations it finds in a massive dataset while being exposed to it during training. As a result, the network can generalize to novel, unexplored data and provide precise forecasts regarding current issues.

Neural networks, however, may also be computationally expensive and challenging to train. When a neural network is being trained, its weights are adjusted depending on the discrepancy between expected output and actual output. This procedure frequently uses the backpropagation technique, which may be slow and computationally expensive for big datasets or intricate networks.

Despite these difficulties, neural networks are still a hot topic for study and development. New network designs that are more suited for particular tasks are being investigated by researchers. For example, convolutional neural networks are well suited for images; however, recurrent neural networks are better for time-series data. Additionally, they are looking into approaches like transfer learning and pruning that improve the effectiveness and simplicity of neural network training. With these developments, neural networks will play an essential part in developing machine learning and artificial intelligence.

1.1.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a type of generative neural network trained and used for generative purposes, that is, the generation of "new" data similar to an original dataset. Since their first introduction in 2014 by Ian Goodfellow [72], they have become one of the most popular and active fields in deep learning research.

The original Generative Adversarial Network [72] consists of two models: a generative model that produces new data, such as images, text, or audio, from input that consists of random noise and a discriminative model that attempts to differentiate between actual data from the training dataset and produced data from the generator network when given both as inputs. Both models are trained in tandem to resemble a game, with the generator trying to trick the discriminator by producing realistic (similar to the original dataset) data and the discriminator attempting to accurately distinguish the real data from the one produced by the generator. The generator network learns to produce more and more realistic output that the discriminator cannot tell apart from actual data through the adversarial training process. Consequently, the discriminator network learns to differentiate between real and fake data.

Realistic pictures like people, landscapes, and animals, as well as other data like music and text, may be produced with astonishing accuracy using these networks. GANs have also been utilized for tasks like movie creation, image-to-image translation, and speech enhancement.

Unfortunately, GANs are known to be very sensitive to hyperparameters and subject to mode collapse, which occurs when the generator only generates a small fraction of all potential outputs [215]. Training GANs may be challenging. Researchers are actively researching to overcome these difficulties and increase the capabilities of GANs. Generally speaking, GANs represent an interesting and exciting area of deep learning research that can transform industries like the creative arts, design, entertainment, and communication, as well as influence domains like drug development and medical imaging.

1.1.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a particular class of neural networks that excel at tasks involving sequential input, such as audio and natural language processing, including prediction and translation tasks. Contrary to the feedforward neural networks, which process each input independently, RNNs process each input sequentially where the order matters. RNNs transfer information using a feedback loop from one stage in the sequence to the next. The network receives an input at each time step, combines it with data from the previous time step, and generates an output. As a result, the network can preserve an internal state that describes the sequence's history up until that moment.

The capacity of RNNs to process input sequences of different lengths is one of its main advantages. In jobs like sentiment analysis or machine translation, where the length of the input fluctuates based on the input text, they are ideally suited for such tasks. Unfortunately, the vanishing gradient problem, which occurs when the gradients used to update the network's weights becomes practically zero, might make it challenging for RNNs to learn long-term relationships. Researchers have created numerous RNN variations that are intended to capture long-term dependencies better to overcome the above issue, including Long Short-Term Memory (LSTM) [88] and Gated Recurrent Unit (GRU) [37].

RNNs have excelled in many natural language processing tasks, including text generating, machine translation, and character and word prediction. They have also been used for handwriting and speech recognition. Researchers are also looking at novel RNN uses, such as creating music, analyzing videos, and making stock value predictions. RNNs are a formidable tool for managing sequential data and might change a wide range of disciplines that use sequence processing.

1.1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a particular class of neural networks that excel at computer vision and image recognition applications. They were initially presented in the 1980s-1990s [61, 114] but only took off in the advent of potent graphics processing units (GPUs) and sizable datasets like ImageNet [46]. Convolutional filters are applied to input data, such as an image, by CNNs in order to function. These filters look for specific features in the image, such as edges or textures, and produce a feature map that shows where these features are in the image. Several convolutional filters can be used to create distinct feature maps that each highlight certain aspects of the picture.

The network often contains one or more pooling layers [113, 114, 160] after the convolutional layers, which downsample the feature maps to lessen their dimensionality and produce spatial invariance. This makes the network more resistant to input modifications, such as adjustments to the size or position of objects in the picture. A fully connected layer or layers, the network's last component, employ the features extracted from the convolutional and pooling layers to predict outcomes. These final (also known as output) layers can be customized to fit the specific task, for example, classifying images into different categories or identifying the object in the picture.

In certain circumstances, CNNs have outperformed humans in image identification tests, which is impressive. They have been used for various purposes, including face and object identification, object detection, and medical imaging. CNNs have been used in numerous fields outside image identification, including speech recognition and natural language processing. CNNs still need help despite their success. Overfitting, which happens when the network grows too complex and begins to match the training data too closely, is one of the critical problems and results in worse performance on new data. Researchers are looking for solutions to this problem, including data augmentation and regularization methods. CNNs are anticipated to keep playing a crucial role in image identification and computer vision jobs with further improvements in the area.

1.2 Thesis Outline

This manuscript is broken into three main parts.

In Chapter 2, we study Generative Adversarial Networks. Particularly, we look into the Discriminative model and issues that can cause training to harm parts of the model. The original Discriminative model loss function is an equally weighted linear combination of real and fake losses that only depend on the actual and generated data. However, with an equally weighted loss, the training may benefit one part of the loss but harm the other, which can cause instability and mode collapse. We introduce an adaptively weighted discriminator loss function(s) that adopts a weighted sum of real and fake parts using the gradients of the real and fake parts of the loss. We can adaptively choose weights to train a discriminator in the direction that benefits the GAN's stability. Later, we extend this idea to the speech enhancement domain, where the discriminator loss function can have two or three parts and has a very different behavior from the image-generating domain.

In Chapter 3, we study Recurrent Neural Networks like architecture and propose several improvements. We introduce orthogonal weights into Gate Recurrent Unit, where orthogonality is preserved via the newly introduced Neumann-Cayley Transformation and helps to capture long-term behavior in sequential data. Moreover, we show that such orthogonal weights help to derive a more desirable molecular descriptor and help to create more reliable prediction models for drug design. In addition, we propose a new normalization technique for the Long Short-Term Memory model, which helps to incorporate temporal dependencies into normalizations.

In Chapter 4, we study Convolutional Neural Networks and applications with underlying symmetric structures. In order to get a symmetrical structure and maintain it during the training phases, we introduce symmetry-generating and symmetry-preserving kernels and use convolutional kernel reparameterization to help enforce the symmetrical structure into the convolutional layer output.

Chapter 2 Generative Adversarial Networks

2.1 Adaptive Weighted Discriminator for Training Generative Adversarial Networks

Generative adversarial network (GAN) has become one of the most important neural network models for classical unsupervised machine learning. A variety of discriminator loss functions have been developed to train GAN's discriminators and they all have a common structure: a sum of real and fake losses that only depends on the actual and generated data, respectively. One challenge associated with an equally weighted sum of two losses is that the training may benefit one loss but harm the other, which we show causes instability and mode collapse. This paper introduces a new family of discriminator loss functions that adopts a weighted sum of real and fake parts, which we call adaptive weighted loss functions or aw-loss functions. Using the gradients of the real and fake parts of the loss, we can adaptively choose weights to train a discriminator in the direction that benefits the GAN's stability. Our method can be potentially applied to any discriminator model with a loss that is a sum of the real and fake parts. Experiments validated the effectiveness of our loss functions on unconditional and conditional image generation tasks, improving the baseline results by a significant margin on CIFAR-10, STL-10, and CIFAR-100 datasets in Inception Scores (IS) and Fréchet Inception Distance (FID) metrics.

2.1.1 Introduction

Generative Adversarial Network (GAN) [72] has become one of the most important neural network models for unsupervised machine learning. The origin of this idea lies in the combination of two neural networks, one generative and one discriminative, that work simultaneously. The task of the generator is to generate data of a given distribution, while the discriminator's purpose is to try to recognize which data are created by the generative model and which are the original ones. While a variety of GAN models have been developed, many of them are prone to issues with training, such as instability, where model parameters might destabilize and not converge, mode collapse, where the generative model produces a limited number of different samples, diminishing gradients where the generator gradient vanishes and training does not occur, and high sensitivity to hyperparameters.

We focus on the discriminative model to rectify the issues of instability and mode collapse in training GAN. In the GAN architecture, the discriminator model takes samples from the original dataset and the output from the generator as input and tries to classify whether a particular element in those samples is *real* or *fake data* [72]. The discriminator updates its parameters by maximizing a discriminator loss function via backpropagation through the discriminator network. In many of the proposed models [72, 74, 121, 130], the discriminator loss function consists of two equally weighted parts: the "real part" that purely relies on the original dataset and the "fake part" that depends on the generator network and its output; for simplicity, we will call them $\mathcal{L}_r$ and $\mathcal{L}_f$ for *real* and *fake* losses, respectively. For example, in the original GAN paper [72], the discriminator loss function $\mathcal{L}_D$ is written as

$$\mathcal{L}_D = \mathcal{L}_r + \mathcal{L}_f, \tag{2.1}$$

with $\mathcal{L}_r = \mathbb{E}_{x\sim p_d}\big[\log D(x)\big]$ and $\mathcal{L}_f = \mathbb{E}_{z\sim p_z}\big[\log(1 - D(G(z)))\big]$, where D and G are the discriminative and generative models, respectively, p_d is the probability distribution of the real data, and p_z is the probability distribution of the generator parameter z.

The GAN discriminator training aims to increase both $\mathcal{L}_r$ and $\mathcal{L}_f$ so that the discriminator $D(\cdot)$ assigns high scores to real data and low scores to fake data. This is done in (2.1) by placing equal weights on $\mathcal{L}_r$ and $\mathcal{L}_f$ [72]. However, the training with $\mathcal{L}_D$ is not performed equally on $\mathcal{L}_r$ and $\mathcal{L}_f$. Indeed, a gradient ascent training step along the $\nabla\mathcal{L}_D$ may decrease $\mathcal{L}_r$ (or $\mathcal{L}_f$), depending on the angle between $\nabla\mathcal{L}_D$ and $\nabla\mathcal{L}_r$ (or $\nabla\mathcal{L}_f$). For example, if we have a large obtuse angle between $\nabla\mathcal{L}_r$ and $\nabla\mathcal{L}_f$, which is the case in most training steps (see Section 2.1.5.1), training along the direction of $\nabla\mathcal{L}_D$ may potentially decrease either $\mathcal{L}_r$ or $\mathcal{L}_f$ by going in the opposite direction to $\nabla\mathcal{L}_r$ or $\nabla\mathcal{L}_f$ (see Section 2.1.3 and Section 2.1.5.2). We suggest that this reduction on the real loss may destabilize training and cause mode collapses. Specifically, if a generator is converging with its generated samples close to the data distribution (or a particular mode), a training step that increases the fake loss will reduce the discriminator scores on the fake data and, by the continuity of $D(\cdot)$, reduce the scores on the nearby real data as well. With the updated discriminator now assigning lower scores to the regions of data where the generator previously approximated well, the generator update is likely to move away from that region and to the regions with higher discriminator scores (possibly a different mode). Hence, we see instability or mode collapse. See Section 2.1.5.3 for experimental results.

We propose a new approach for training the discriminative model by modifying the discriminator loss function and introducing adaptive weights in the following way,

$$\mathcal{L}_D^{aw} = w_r \cdot \mathcal{L}_r + w_f \cdot \mathcal{L}_f. \tag{2.2}$$

We adaptively choose w_r and w_f weights to calibrate the training in the real and fake losses. Using the information of $\nabla\mathcal{L}_r$ and $\nabla\mathcal{L}_f$, we can control the gradient direction, $\nabla\mathcal{L}_D^{aw}$, by either training in the direction that benefits both $\mathcal{L}_r$ and $\mathcal{L}_f$ or increasing one loss while not changing the other. This attempts to avoid a situation where training may benefit one loss but significantly harm the other. Section 2.1.3 presents a more detailed mathematical approach.

Our proposed method can be applied to any GAN model with a discriminator loss function composed of two parts as in (2.1). For our experiments, we have applied adaptive weights to the SN-GAN [142], AutoGAN [71], and BigGAN [18] models for unconditional as well as a conditional image generating tasks. We have achieved significant improvements on them for CIFAR-10, STL-10 and CIFAR-100 datasets in both Inception Scores (IS) and Fréchet Inception Distance (FID) metrics, see Section 2.1.4. Our code is available at `https://github.com/vasily789/adaptive-weighted-gans`

Notation: We use $\langle \cdot, \cdot \rangle_2$ to denote the Euclidean inner product, $\|x\|_2$ the Euclidean 2-norm, and $\angle_2(x, y) := \arccos\left(\frac{\langle x, y\rangle_2}{\|x\|_2 \|y\|_2}\right)$ the angle between vectors x and y.

2.1.2 Related Work

GAN was first proposed in [72] for creating generative models via simultaneous optimization of a discriminative and a generative model. The original GAN may suffer from vanishing gradients during training, non-convergence of the model(s), and mode collapse; see [30, 138, 140, 158, 165] for discussions. Several papers [6, 74, 121, 130] have addressed the issues of vanishing gradients by introducing new loss functions. The LSGAN proposed in [130] adopted the least squares loss function for the discriminator that relies on minimizing the Pearson χ^2 divergence, in contrast to the Jensen–Shannon divergence used in GAN. The WGAN model [6, 74] introduced another way to solve the problem of convergence and mode collapse by incorporating Wasserstein-1 distance into the loss function. As a result, WGAN has a loss function associated with image quality, improving learning stability and convergence. The hinge loss function introduced in [121, 184] achieved smaller error rates than cross-entropy, being stable against different regularization techniques and having a low computational cost [49]. The models in [12, 117, 189] adopted a loss function called maximum mean discrepancy (MMD). A repulsive function to stabilize the MMD-GAN training was employed in [198], and the MMD loss function was weighted in [48] according to the contribution of data to the loss function. [151] presented a dual discriminator GAN that combines two discriminators in a weighted sum.

New loss functions are not the only way of improving GAN's framework. DCGAN [158], one of the first and more significant improvements in the GAN architecture, was the incorporation of deep convolutional networks. The Progressive Growing GAN [104] was created based on [6, 74] with the main idea of progressively adding new layers of higher resolution during training, which helps to create highly realistic images. [51, 71, 180] developed neural architecture search methods to find an optimal neural network architecture to train GAN for a particular task.

Many works are dedicated to conditional GAN, for example, BigGAN [18], which utilized a model with many parameters and larger batch sizes showing a significant benefit of scaling.

There are many works devoted to improving or analyzing GAN training. [140] trained the generator by optimizing a loss function unrolled from several training iterations of the discriminator training. SN-GAN [142], normalized the spectral norm of each weight to stabilize the training. Recent work [168] introduced stable rank normalization that simultaneously controls the Lipschitz constant and the stable rank of a layer. [119] developed an analysis to suggest that first-order approximations of the discriminator lead to instability and mode collapse. [146] proved local stability under the model that both the generator and the discriminator are updated simultaneously via gradient descent. [38] analyzed the stability of GANs through the stationarity of the generator. [139] points out that absolute continuity is necessary for GAN training to converge. Relativistic GAN [101] addressed the observation that with generator training increasing the probability that fake data is real, the probability of real data being real would decrease. [11] proposed a method of re-weighting training samples to correct for the mass shift between the transferred distributions in the domain transfer setup. [31] viewed GAN as an energy-based model and proposed an MCMC sampling-based method.

2.1.3 Adaptive Weighted Discriminator

In GAN training, if we maximize $\mathcal{L}_D$ to convergence in training discriminator D, we should meet the goal to increase both $\mathcal{L}_r$ and $\mathcal{L}_f$. However, in practice, we train with a gradient ascent step along $\nabla\mathcal{L}_D = \nabla\mathcal{L}_r + \nabla\mathcal{L}_f$, which may be dominated by either $\nabla\mathcal{L}_r$ or $\nabla\mathcal{L}_f$. Then, the training may be done primarily on one of the losses, either $\mathcal{L}_r$ or $\mathcal{L}_f$. Consider a gradient ascent training iteration for $\mathcal{L}_D$,

$$\theta_1 \longleftarrow \theta_0 + \lambda\nabla\mathcal{L}_D, \tag{2.3}$$

where λ is a learning rate. Then using the Taylor Theorem, we can expand both $\mathcal{L}_r$ and $\mathcal{L}_f$ about θ_0,

$$\begin{aligned}\mathcal{L}_r(\theta_1) &= \mathcal{L}_r(\theta_0) + \lambda\nabla\mathcal{L}_r^T\nabla\mathcal{L}_D + \mathcal{O}(\lambda^2) \\ &= \mathcal{L}_r(\theta_0) + \lambda\left\|\nabla\mathcal{L}_r\right\|_2\left\|\nabla\mathcal{L}_D\right\|_2\cos\left(\angle_2\left(\nabla\mathcal{L}_r,\nabla\mathcal{L}_D\right)\right) + \mathcal{O}(\lambda^2)\end{aligned} \tag{2.4}$$

and

$$\mathcal{L}_f(\theta_1) = \mathcal{L}_f(\theta_0) + \lambda\left\|\nabla\mathcal{L}_f\right\|_2\left\|\nabla\mathcal{L}_D\right\|_2\cos\left(\angle_2\left(\nabla\mathcal{L}_f,\nabla\mathcal{L}_D\right)\right) + \mathcal{O}(\lambda^2), \tag{2.5}$$

where we have omitted the evaluation point θ_0 in all gradients (i.e. $\nabla\mathcal{L}_* = \nabla\mathcal{L}_*(\theta_0)$) to avoid cumbersome expressions. If one of $\angle_2\left(\nabla\mathcal{L}_r,\nabla\mathcal{L}_D\right)$ and $\angle_2\left(\nabla\mathcal{L}_f,\nabla\mathcal{L}_D\right)$ is obtuse, then to the first order approximation, the corresponding loss is decreased. This causes a decrease in the discriminator assigning a correct score $D(\cdot)$ to the real (or fake) data. Thus, a gradient ascent step with loss (2.1) may turn out to decrease one of the losses if the angle $\angle_2\left(\nabla\mathcal{L}_r,\nabla\mathcal{L}_f\right) > 90°$. This situation occurs often in GAN training; see Section 2.1.5 for some experimental results illustrating this.

This undesirable situation is expected to happen in GAN training when the generator has produced samples close to the data distribution or specific modes. If a training step in the direction $\nabla\mathcal{L}_D$ results in an increase in the fake loss or, equivalently, a decrease in the discriminator scores $D(G(z))$ on the fake data, it will decrease the scores $D(x)$ on the real data as well by the continuity of $D(\cdot)$. Equivalently, this reduces the real loss. With the updated discriminator assigning lower scores to the regions of the data where the generator previously approximated well, the generator update using the new discriminator will likely move in the direction where the discriminator scores are higher and hence leave the region it was converging to. We suggest that this is one of the causes of instability in GAN training. If the regions with high discriminator scores contain only a few modes of data distribution, this leads to mode collapse; see the study in Section 2.1.5.3.

To remedy this situation, we propose to modify the training gradient $\nabla \mathcal{L}_D$ to encourage high discriminator scores for real data. We propose a new family of discriminator loss functions, which we call adaptive weighted loss function or aw-loss function; see equation (2.2).

We first show that the proposed weighted discriminator (2.2) with fixed weights carries the same theoretical properties of the original GAN as stated in [72, 139] for binary-cross-entropy loss function, i.e., when the min-max problem is solved exactly, we recover the data distribution.

Theorem 1. *Let $p_{\mathrm{d}}(x)$ and $p_{\mathrm{g}}(x)$ be the density functions for the data and model distributions, $\mathbb{P}_{\mathrm{d}}$ and $\mathbb{P}_{\mathrm{g}}$, respectively. Consider $\mathcal{L}^{aw}(D, p_{\mathrm{g}}) = w_r \mathbb{E}_{x\sim p_d}\big[\log D(x)\big] + w_f \mathbb{E}_{z\sim p_g}\big[\log(1-D(G(z)))\big]$ with fixed $w_r, w_f > 0$.*

1. *Given a fixed $p_{\mathrm{g}}(x)$, $\mathcal{L}^{aw}(D, p_{\mathrm{g}})$ is maximized by*

$$D^*(x) = \frac{w_r p_{\mathrm{d}}(x)}{w_r p_{\mathrm{d}}(x) + w_f p_{\mathrm{g}}(x)} \tag{2.6}$$

   for $x \in \mathrm{supp}(p_{\mathrm{d}}) \cup \mathrm{supp}(p_{\mathrm{g}})$.

2. *Moreover,*

$$\min_{p_{\mathrm{g}}} \max_{D} \mathcal{L}^{aw}(D, p_{\mathrm{g}}) = w_r \log \frac{w_r}{w_r + w_f} + w_f \log \frac{w_f}{w_r + w_f} \tag{2.7}$$

   with the minimum attained by $p_{\mathrm{g}}(x) = p_{\mathrm{d}}(x)$.

Proof.

1. First, the function $f(t) = a \log t + b \log(1-t)$ has its maximum in $[0,1]$ at $t = \dfrac{a}{a+b}$. Given a fixed $p_{\mathrm{g}}(x)$, $w_r > 0$ and $w_f > 0$.

$$\begin{aligned}
\mathcal{L}^{aw}(D, p_{\mathrm{g}}) &= w_r \mathbb{E}_{x\sim p_{\mathrm{d}}}\left[\log\left(D(x)\right)\right] + w_f \mathbb{E}_{x\sim p_{\mathrm{g}}}\left[\log\left(1-D(x)\right)\right] && (2.8)\\
&= \int_x w_r p_{\mathrm{d}}(x) \log\left(D(x)\right) + w_f p_{\mathrm{g}}(x) \log\left(1-D(x)\right) dx && (2.9)\\
&\le \int_x w_r p_{\mathrm{d}}(x) \log\left(D^*(x)\right) + w_f p_{\mathrm{g}}(x) \log\left(1-D^*(x)\right) dx && (2.10)\\
&= w_r \mathbb{E}_{x\sim p_{\mathrm{d}}}\left[\log\left(\frac{w_r p_{\mathrm{d}}(x)}{w_r p_{\mathrm{d}}(x) + w_f p_{\mathrm{g}}(x)}\right)\right] \\
&\qquad + w_f \mathbb{E}_{x\sim p_{\mathrm{g}}}\left[\log\left(\frac{w_f p_{\mathrm{g}}(x)}{w_r p_{\mathrm{d}}(x) + w_f p_{\mathrm{g}}(x)}\right)\right]. && (2.11)
\end{aligned}$$

   where the equality holds if $D(x) = D^*(x)$. Therefore, $\mathcal{L}^{aw}(D, p_{\mathrm{g}})$ is maximum when $D = D^*$.

2. If $p_{\mathrm{g}}(x) = p_{\mathrm{d}}(x)$, then $D^*(x) = \dfrac{w_r}{w_r + w_f}$ and

$$\max_D \mathcal{L}^{aw}(D, p_{\mathrm{g}}) = w_r \mathbb{E}_{x\sim p_{\mathrm{d}}}\left[\log\left(\frac{w_r}{w_r + w_f}\right)\right] + w_f \mathbb{E}_{x\sim p_{\mathrm{g}}}\left[\log\left(\frac{w_f}{w_r + w_f}\right)\right] \tag{2.12}$$

$$= w_r \log\left(\frac{w_r}{w_r + w_f}\right) + w_f \log\left(\frac{w_f}{w_r + w_f}\right). \tag{2.13}$$

On the other hand,

$$\max_D \mathcal{L}^{aw}(D, p_{\mathrm{g}}) = w_r \mathbb{E}_{x\sim p_{\mathrm{d}}}\left[\log\left(\frac{w_r p_{\mathrm{d}}(x)}{w_r p_{\mathrm{d}}(x) + w_f p_{\mathrm{g}}(x)}\right)\right] + w_f \mathbb{E}_{x\sim p_{\mathrm{g}}} \log\left[\left(\frac{w_f p_{\mathrm{g}}(x)}{w_r p_{\mathrm{d}}(x) + w_f p_{\mathrm{g}}(x)}\right)\right] \tag{2.14}$$

$$= w_r \log\left(\frac{w_r}{w_r + w_f}\right) + w_f \log\left(\frac{w_f}{w_r + w_f}\right) + w_r KL\left(p_{\mathrm{d}}\middle|\frac{w_r p_{\mathrm{d}} + w_f p_{\mathrm{g}}}{w_r + w_f}\right) + w_f KL\left(p_{\mathrm{g}}\middle|\frac{w_r p_{\mathrm{d}} + w_f p_{\mathrm{g}}}{w_r + w_f}\right) \tag{2.15}$$

$$\geq w_r \log\left(\frac{w_r}{w_r + w_f}\right) + w_f \log\left(\frac{w_f}{w_r + w_f}\right), \tag{2.16}$$

where KL is the Kullback-Leibler divergence and equality holds when $p_{\mathrm{d}} = \dfrac{w_r p_{\mathrm{d}} + w_f p_{\mathrm{g}}}{w_r + w_f}$ and $p_{\mathrm{g}} = \dfrac{w_r p_{\mathrm{d}} + w_f p_{\mathrm{g}}}{w_r + w_f}$. Thus we have shown that

$$\min_{p_{\mathrm{g}}} \max_D \mathcal{L}^{aw}(D, p_{\mathrm{g}}) = w_r \log\left(\frac{w_r}{w_r + w_f}\right) + w_f \log\left(\frac{w_f}{w_r + w_f}\right). \tag{2.17}$$

and the minimum is attained when $p_{\mathrm{g}} = p_{\mathrm{d}}$.

□

To choose the weights w_r and w_f, we propose an adaptive scheme where the weights w_r and w_f are determined using gradient information of both $\mathcal{L}_r$ and $\mathcal{L}_f$. This structure allows us to adjust the direction of the discriminator loss function's gradient to achieve the training goal to increase both $\mathcal{L}_r$ and $\mathcal{L}_f$, or at least not to decrease either loss. We propose Algorithm 1 based on the following gradient relations with various weight choices.

Theorem 2. *Consider $\mathcal{L}_D^{aw}$ in (2.2) and the gradient $\nabla\mathcal{L}_D^{aw}$.*

1. If $w_r = \dfrac{1}{\|\nabla\mathcal{L}_r\|_2}$ *and* $w_f = \dfrac{1}{\|\nabla\mathcal{L}_f\|_2}$, *then* $\nabla\mathcal{L}_D^{aw}$ *is the angle bisector of* $\nabla\mathcal{L}_r$ *and* $\nabla\mathcal{L}_f$*, i.e.*

$$\angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_r\right) = \angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_f\right) = \angle_2\left(\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\right)/2. \tag{2.18}$$

2. If $w_r = \dfrac{1}{\|\nabla\mathcal{L}_r\|_2}$ *and* $w_f = -\dfrac{\langle\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\rangle_2}{\|\nabla\mathcal{L}_f\|_2^2\cdot\|\nabla\mathcal{L}_r\|_2}$, *then*

$$\angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_f\right) = 90^\circ,\ \angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_r\right) \le 90^\circ. \tag{2.19}$$

3. If $w_r = -\dfrac{\langle\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\rangle_2}{\|\nabla\mathcal{L}_r\|_2^2\cdot\|\nabla\mathcal{L}_f\|_2}$ *and* $w_f = \dfrac{1}{\|\nabla\mathcal{L}_f\|_2}$, *then*

$$\angle_2\left(\mathcal{L}_D^{aw}, \nabla\mathcal{L}_r\right) = 90^\circ,\ \angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_f\right) \le 90^\circ. \tag{2.20}$$

Proof.

1. If $w_r = \dfrac{1}{\|\nabla\mathcal{L}_r\|_2}$ and $w_f = \dfrac{1}{\|\nabla\mathcal{L}_f\|_2}$, then

$$\mathcal{L}_D^{aw} = \frac{1}{\|\nabla\mathcal{L}_r\|_2}\mathcal{L}_r + \frac{1}{\|\nabla\mathcal{L}_f\|_2}\mathcal{L}_f. \tag{2.21}$$

Using the definition of Euclidean inner product,

$$\begin{aligned}
\cos\left(\angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_r\right)\right) &= \frac{\langle\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_r\rangle_2}{\|\nabla\mathcal{L}_D^{aw}\|_2\,\|\nabla\mathcal{L}_r\|_2} && (2.22)\\
&= \frac{\dfrac{1}{\|\nabla\mathcal{L}_r\|_2}\langle\nabla\mathcal{L}_r, \nabla\mathcal{L}_r\rangle_2 + \dfrac{1}{\|\nabla\mathcal{L}_f\|_2}\langle\nabla\mathcal{L}_f, \nabla\mathcal{L}_r\rangle_2}{\|\nabla\mathcal{L}_D^{aw}\|_2\,\|\nabla\mathcal{L}_r\|_2} && (2.23)\\
&= \frac{1}{\|\nabla\mathcal{L}_D^{aw}\|_2} + \frac{\langle\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\rangle_2}{\|\nabla\mathcal{L}_D^{aw}\|_2\,\|\nabla\mathcal{L}_r\|_2\,\|\nabla\mathcal{L}_f\|_2} && (2.24)
\end{aligned}$$

$$\begin{aligned}
\cos\left(\angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_f\right)\right) &= \frac{\langle\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_f\rangle_2}{\|\nabla\mathcal{L}_D^{aw}\|_2\,\|\nabla\mathcal{L}_f\|_2} && (2.25)\\
&= \frac{\dfrac{1}{\|\nabla\mathcal{L}_r\|_2}\langle\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\rangle_2 + \dfrac{1}{\|\nabla\mathcal{L}_f\|_2}\langle\nabla\mathcal{L}_f, \nabla\mathcal{L}_f\rangle_2}{\|\nabla\mathcal{L}_D^{aw}\|_2\,\|\nabla\mathcal{L}_f\|_2} && (2.26)\\
&= \frac{1}{\|\nabla\mathcal{L}_D^{aw}\|_2} + \frac{\langle\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\rangle_2}{\|\nabla\mathcal{L}_D^{aw}\|_2\,\|\nabla\mathcal{L}_r\|_2\,\|\nabla\mathcal{L}_f\|_2} && (2.27)
\end{aligned}$$

We can rewrite $\|\nabla\mathcal{L}_D^{aw}\|_2$ in term of $\angle_2\left(\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\right)$, that is

$$\|\nabla\mathcal{L}_D^{aw}\|_2^2 = \langle\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_D^{aw}\rangle_2$$

$$= \left\langle \frac{1}{\|\nabla \mathcal{L}_r\|_2} \nabla \mathcal{L}_r + \frac{1}{\|\nabla \mathcal{L}_f\|_2} \nabla \mathcal{L}_f, \frac{1}{\|\nabla \mathcal{L}_r\|_2} \nabla \mathcal{L}_r + \frac{1}{\|\nabla \mathcal{L}_f\|_2} \nabla \mathcal{L}_f \right\rangle_2$$
$$= \frac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_r \rangle_2}{\|\nabla \mathcal{L}_r\|_2^2} + \frac{\langle \nabla \mathcal{L}_f, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_f\|_2^2} + \frac{2 \langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_r\|_2 \|\nabla \mathcal{L}_f\|_2}$$
$$= 2 \left(1 + \cos\left(\angle_2 \left(\nabla \mathcal{L}_r, \nabla \mathcal{L}_f\right)\right)\right). \tag{2.28}$$

Notice that (2.24) can be rewritten using $\angle_2 (\nabla \mathcal{L}_r, \nabla \mathcal{L}_f)$ as

$$\cos\left(\angle_2 \left(\nabla \mathcal{L}_D^{aw}, \nabla \mathcal{L}_r\right)\right) = \frac{1}{\|\nabla \mathcal{L}_D^{aw}\|_2} + \frac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_D^{aw}\|_2 \|\nabla \mathcal{L}_r\|_2 \|\nabla \mathcal{L}_f\|_2} \tag{2.29}$$
$$= \frac{1}{\|\nabla \mathcal{L}_D^{aw}\|_2} \left(1 + \cos\left(\angle_2 \left(\nabla \mathcal{L}_r, \nabla \mathcal{L}_f\right)\right)\right) \tag{2.30}$$
$$= \sqrt{\frac{1 + \cos\left(\angle_2 \left(\nabla \mathcal{L}_r, \nabla \mathcal{L}_f\right)\right)}{2}} \tag{2.31}$$
$$= \cos\left(\angle_2 \left(\nabla \mathcal{L}_r, \nabla \mathcal{L}_f\right)/2\right). \tag{2.32}$$

Thus, $\angle_2 (\nabla \mathcal{L}_D^{aw}, \nabla \mathcal{L}_r) = \angle_2 (\nabla \mathcal{L}_D^{aw}, \nabla \mathcal{L}_f) = \angle_2 (\nabla \mathcal{L}_r, \nabla \mathcal{L}_f)/2$.

2. If $w_r = \dfrac{1}{\|\nabla \mathcal{L}_r\|_2}$ and $w_f = -\dfrac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_f\|_2^2 \|\nabla \mathcal{L}_r\|_2}$ then

$$\mathcal{L}_D^{aw} = \frac{1}{\|\nabla \mathcal{L}_r\|_2} \mathcal{L}_r - \frac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_f\|_2^2 \|\nabla \mathcal{L}_r\|_2} \mathcal{L}_f. \tag{2.33}$$

Using this aw-loss function, we have

$$\langle \nabla \mathcal{L}_D^{aw}, \nabla \mathcal{L}_f \rangle_2 = \left\langle \frac{1}{\|\nabla \mathcal{L}_r\|_2} \nabla \mathcal{L}_r - \frac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_f\|_2^2 \|\nabla \mathcal{L}_r\|_2} \nabla \mathcal{L}_f, \nabla \mathcal{L}_f \right\rangle_2 \tag{2.34}$$
$$= \frac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_r\|_2} - \frac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2 \langle \nabla \mathcal{L}_f, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_f\|_2^2 \|\nabla \mathcal{L}_r\|_2} \tag{2.35}$$
$$= \frac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_r\|_2} - \frac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_r\|_2} = 0, \tag{2.36}$$

and

$$\langle \nabla \mathcal{L}_D^{aw}, \nabla \mathcal{L}_r \rangle_2 = \left\langle \frac{1}{\|\nabla \mathcal{L}_r\|_2} \nabla \mathcal{L}_r - \frac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_f\|_2^2 \|\nabla \mathcal{L}_r\|_2} \nabla \mathcal{L}_f, \nabla \mathcal{L}_r \right\rangle_2 \tag{2.37}$$
$$= \frac{\|\nabla \mathcal{L}_r\|_2^2}{\|\nabla \mathcal{L}_r\|_2} - \frac{\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2^2}{\|\nabla \mathcal{L}_f\|_2^2 \|\nabla \mathcal{L}_r\|_2} \tag{2.38}$$
$$\geq \|\nabla \mathcal{L}_r\|_2 - \frac{\|\nabla \mathcal{L}_f\|_2^2 \|\nabla \mathcal{L}_r\|_2^2}{\|\nabla \mathcal{L}_f\|_2^2 \|\nabla \mathcal{L}_r\|_2} \tag{2.39}$$
$$= \|\nabla \mathcal{L}_r\|_2 - \|\nabla \mathcal{L}_r\|_2 = 0. \tag{2.40}$$

Thus, $\angle_2 (\nabla \mathcal{L}_D^{aw}, \nabla \mathcal{L}_f) = 90^\circ$ and $\angle_2 (\nabla \mathcal{L}_D^{aw}, \nabla \mathcal{L}_r) \leq 90^\circ$.

3. If $w_r = -\dfrac{\langle \nabla\mathcal{L}_r, \nabla\mathcal{L}_f \rangle_2}{\|\nabla\mathcal{L}_r\|_2^2 \cdot \|\nabla\mathcal{L}_f\|_2}$ and $w_f = \dfrac{1}{\|\nabla\mathcal{L}_f\|_2}$ then

$$\mathcal{L}_D^{aw} = -\frac{\langle \nabla\mathcal{L}_r, \nabla\mathcal{L}_f \rangle_2}{\|\nabla\mathcal{L}_r\|_2^2 \cdot \|\nabla\mathcal{L}_f\|_2}\mathcal{L}_r + \frac{1}{\|\nabla\mathcal{L}_f\|_2}\mathcal{L}_f, \tag{2.41}$$

and similar argument as above proves that $\angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_r\right) = 90°$ and $\angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_f\right) \leq 90°$.

□

The first case in the above theorem allows us to choose weights for (2.2) such that we can train $\mathcal{L}_r$ and $\mathcal{L}_f$ by going in the direction of the angle bisector. However, sometimes the direction of the angle bisector might not be optimal. For example, if the angle between $\nabla\mathcal{L}_r$ and $\nabla\mathcal{L}_f$ is close to 180°, then the bisector direction will effectively not train either loss. During training, $\mathcal{L}_f$ is often easier to train than $\mathcal{L}_r$ meaning that the fake gradient has a larger magnitude. In this situation, we might want to train just on the real gradient direction by simply choosing $w_f = 0$, but if the angle between $\nabla\mathcal{L}_r$ and $\nabla\mathcal{L}_f$ is obtuse, we will increase $\mathcal{L}_r$ but significantly decrease $\mathcal{L}_f$ which is undesirable. The second case in Theorem 2 suggests a direction that forms an acute angle with $\nabla\mathcal{L}_r$ and orthogonal to $\nabla\mathcal{L}_f$ (see Figure 2.1); such a direction will increase $\mathcal{L}_r$ and to the first order approximation will leave $\mathcal{L}_f$ unchanged. When $\mathcal{L}_r$ is high, the third case in Theorem 2 would allow us to increase $\mathcal{L}_f$ while minimizing changes to $\mathcal{L}_r$.

Inspired by the Theorem 2 and observations that we have made, we can calibrate discriminator training in a way that produces and maintains high real loss to reduce fluctuations in the real loss (or real discriminator scores) to improve stability. Algorithm 1 describes the procedure for updating weights of the aw-loss function in (2.2) during training using the information of $\nabla\mathcal{L}_r$ and $\nabla\mathcal{L}_f$.

Algorithm 1 is designed to first avoid, up to the first order approximation, decreasing $\mathcal{L}_r$ or $\mathcal{L}_f$ during a gradient ascent iteration. Furthermore, it chooses to favor training real loss unless the mean real score is greater than the mean fake score (i.e., $s_f \leq s_r$) and the real mean score is at least $\alpha_1 = 0.5$ (i.e., $\alpha_1 \leq s_r$). Here the mean discriminator scores s_r and s_f represent the mean probability that the discriminator assigns to x_i's and y_j's, respectively, as real data. When s_r is highly satisfactory with $s_r \geq \alpha_2 = 0.75$ (the midpoint between the minimum probability 0.5 and the maximum probability 1 for correct classifications of real data), we favor training the fake loss; otherwise, we train both equally. Maintaining these training criteria will reduce the fluctuations in real and fake discriminator scores and avoid instability. See the study in Section 2.1.5.3. Note that we impose a small gap $\delta = 0.05$ in $s_f - \delta > s_r$ to account for situations when s_r is nearly identical to s_f.

The way we favor training the real or fake loss depends on whether the angle between $\nabla\mathcal{L}_r$ and $\nabla\mathcal{L}_f$ is obtuse or not. In Algorithm 1, the first and the third cases are concerned with the more frequent situation (see Section 2.1.5.1 and Figure 2.4) where the angle between $\nabla\mathcal{L}_r$ and $\nabla\mathcal{L}_f$ is obtuse. These cases are the ones that are developed in Theorem 2. In the first case, we favor training real loss by going in the

Algorithm 1: Adaptive weighted discriminator method with normalization for one step of discriminator training.

```
1:  Given: $\mathbb{P}_{\rm d}$ and $\mathbb{P}_{\rm g}$ - data and model distributions;
2:  Given: $\alpha_1 = 0.5$, $\alpha_2 = 0.75$, $\varepsilon = 0.05$, $\delta = 0.05$;
3:  Sample: $x_1, \ldots, x_n \sim \mathbb{P}_{\rm d}$ and $y_1, \ldots, y_n \sim \mathbb{P}_{\rm g}$;
4:  Compute: $\nabla \mathcal{L}_r$, $\nabla \mathcal{L}_f$, $s_r = \frac{1}{n}\sum_{i=1}^{n} \sigma(D(x_i))$, $s_f = \frac{1}{n}\sum_{j=1}^{n} \sigma(D(y_j))$;
5:  if $s_r < s_f - \delta$ or $s_r < \alpha_1$ then
6:      if $\angle_2 (\nabla \mathcal{L}_r, \nabla \mathcal{L}_f) > 90^\circ$ then
7:          $w_r = \frac{1}{\|\nabla \mathcal{L}_r\|_2} + \varepsilon$ and $w_f = \frac{-\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_f\|_2^2 \cdot \|\nabla \mathcal{L}_r\|_2} + \varepsilon$;
8:      else
9:          $w_r = \frac{1}{\|\nabla \mathcal{L}_r\|_2} + \varepsilon$ and $w_f = \varepsilon$;
10:     end
11: else if $s_r > s_f - \delta$ and $s_r > \alpha_2$ then
12:     if $\angle_2 (\nabla \mathcal{L}_r, \nabla \mathcal{L}_f) > 90^\circ$ then
13:         $w_r = \frac{-\langle \nabla \mathcal{L}_r, \nabla \mathcal{L}_f \rangle_2}{\|\nabla \mathcal{L}_r\|_2^2 \cdot \|\nabla \mathcal{L}_f\|_2} + \varepsilon$ and $w_f = \frac{1}{\|\nabla \mathcal{L}_f\|_2} + \varepsilon$;
14:     else
15:         $w_r = \varepsilon$ and $w_f = \frac{1}{\|\nabla \mathcal{L}_f\|_2} + \varepsilon$;
16:     end
17: else
18:     $w_r = \frac{1}{\|\nabla \mathcal{L}_r\|_2} + \varepsilon$ and $w_f = \frac{1}{\|\nabla \mathcal{L}_f\|_2} + \varepsilon$;
19: end
```

direction orthogonal to the $\nabla \mathcal{L}_f$, illustrated in Figure 2.1. In the third case, we favor the fake loss by going in the direction orthogonal to $\nabla \mathcal{L}_r$. Similarly, the second and the fourth cases are concerned with the situation when the angle between $\nabla \mathcal{L}_r$ and $\nabla \mathcal{L}_f$ is acute. We use the same criteria to decide if training should favor the real or fake directions, but in this case we favor training the real or fake loss by using the direction of the corresponding gradient. Lastly, in the fifth case, it is desirable to increase both s_r and s_f without either taking priority, so we choose to train in the direction of the angle bisector between $\nabla \mathcal{L}_r$ and $\nabla \mathcal{L}_f$.

The two thresholds α_1 and α_2 in Algorithm 1 can be treated as hyperparameters. Our ablation studies show that the default $\alpha_1 = 0.5$ and $\alpha_2 = 0.75$ as discussed earlier, are indeed good choices; see Ablation Study in Section 2.1.5.4.

All weights stated in Algorithm 1 normalize both the real and fake gradients for the purpose of avoiding differently sized gradients, which has the effect of preventing exploding gradients and speeds up training, i.e., achieves better IS and FID with fewer

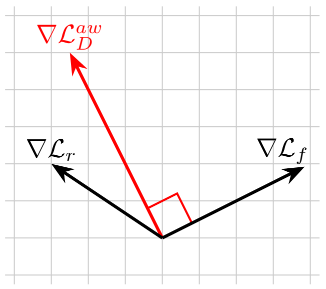
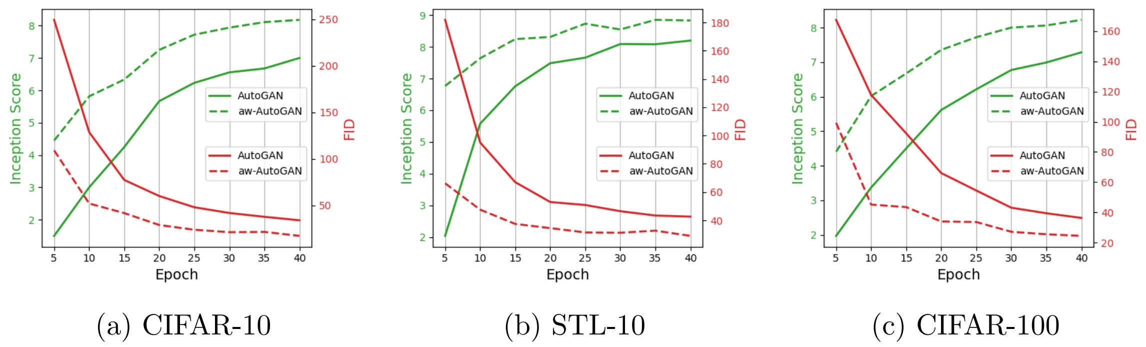


Figure 2.1: Depiction of the second case of Theorem 2.



(a) CIFAR-10 (b) STL-10 (c) CIFAR-100

Figure 2.2: AutoGAN vs. aw-AutoGAN IS and FID plots for the first 40 epochs.

epochs, see Figure 2.2. With this implementation, we implement a linear learning rate decay to ensure convergence. However, the aw-method performs well without normalization and achieves comparable results. The corresponding results are stated as the following theorem.

Theorem 3. *Consider $\mathcal{L}_D^{aw}$ in (2.2) and the gradient $\nabla\mathcal{L}_D^{aw}$.*

1. *If $w_r = 1$ and $w_f = -\dfrac{\langle\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\rangle_2}{\|\nabla\mathcal{L}_f\|_2^2}$, then*

$$\angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_f\right) = 90^\circ,\ \angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_r\right) \leq 90^\circ. \tag{2.42}$$

2. *If $w_r = -\dfrac{\langle\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\rangle_2}{\|\nabla\mathcal{L}_r\|_2^2}$ and $w_f = 1$, then*

$$\angle_2\left(\mathcal{L}_D^{aw}, \nabla\mathcal{L}_r\right) = 90^\circ,\ \angle_2\left(\nabla\mathcal{L}_D^{aw}, \nabla\mathcal{L}_f\right) \leq 90^\circ. \tag{2.43}$$

Proof. Identical to the proof of Theorem 2. □

Similar to Algorithm 1, we have developed Algorithm 2 using Theorem 3. The key difference between Algorithms 1 and 2 is the normalization of the gradients; the rest of the algorithm is unchanged including the values for $\alpha_1 = 0.5$, $\alpha_2 = 0.75$, $\varepsilon = 0.05$ and $\delta = 0.05$.

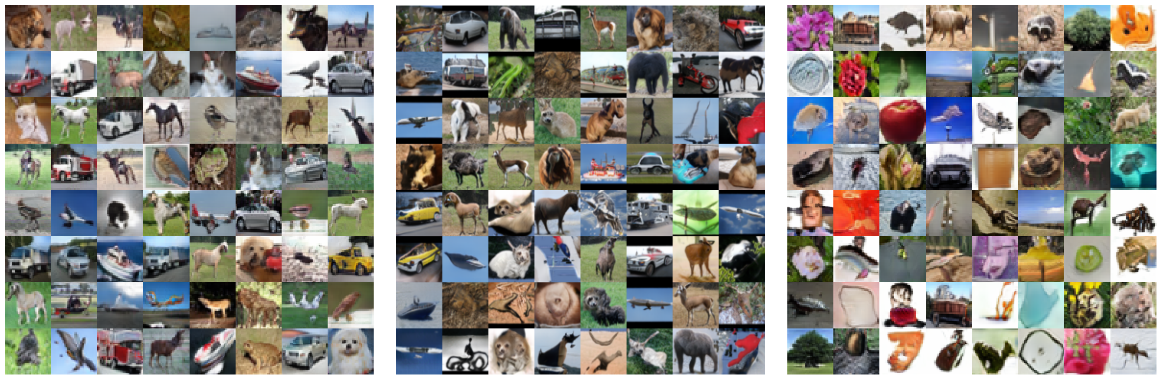
Algorithm 2: Adaptive weighted discriminator method without normalization for one step of discriminator training.

```
1: Given: $\mathbb{P}_\mathrm{d}$ and $\mathbb{P}_\mathrm{g}$ - data and model distributions;
2: Given: $\alpha_1 = 0.5$, $\alpha_2 = 0.75$, $\varepsilon = 0.05$, $\delta = 0.05$;
3: Sample: $x_1, \ldots, x_n \sim \mathbb{P}_\mathrm{d}$ and $y_1, \ldots, y_n \sim \mathbb{P}_\mathrm{g}$;
4: Compute: $\nabla\mathcal{L}_r$, $\nabla\mathcal{L}_f$, $s_r = \frac{1}{n}\sum_{i=1}^n \sigma(D(x_i))$, $s_f = \frac{1}{n}\sum_{j=1}^n \sigma(D(y_j))$;
5: if $s_r < s_f - \delta$ or $s_r < \alpha_1$ then
6:     if $\angle_2(\nabla\mathcal{L}_r, \nabla\mathcal{L}_f) > 90^\circ$ then
7:         $w_r = 1 + \varepsilon$ and $w_f = -\frac{\langle\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\rangle_2}{\|\nabla\mathcal{L}_f\|_2^2} + \varepsilon$;
8:     else
9:         $w_r = 1 + \varepsilon$ and $w_f = \varepsilon$;
10:    end
11: else if $s_r > s_f - \delta$ and $s_r > \alpha_2$ then
12:    if $\angle_2(\nabla\mathcal{L}_r, \nabla\mathcal{L}_f) > 90^\circ$ then
13:        $w_r = -\frac{\langle\nabla\mathcal{L}_r, \nabla\mathcal{L}_f\rangle_2}{\|\nabla\mathcal{L}_r\|_2^2} + \varepsilon$ and $w_f = 1 + \varepsilon$;
14:    else
15:        $w_r = \varepsilon$ and $w_f = 1 + \varepsilon$;
16:    end
17: else
18:    $w_r = 1 + \varepsilon$ and $w_f = 1 + \varepsilon$;
19: end
```

On average, normalized weights achieve better results than non-normalized ones. We advocate the normalized version (Algorithm 1), but both produce quite competitive results (See Subsection 2.1.4) and should be considered in implementations.

A small constant ε is added to all the weights in Algorithms 1 and 2 to avoid numerical discrepancies in cases that would prevent the discriminator model from training and updating. For example, there are cases when our algorithm would set $w_r = 0$ but at the same time, $\nabla\mathcal{L}_f$ would be almost zero, which will result in $\nabla\mathcal{L}_D^{aw}$ being practically zero. We have set $\varepsilon = 0.05$ in all of our experiments.

Algorithms 1 and 2 have a small computational overhead. At each iteration, we compute inner products and norms that are used for computing w_r and w_f, and then use these weights to update $\nabla\mathcal{L}_D^{aw}$. If we have k trainable parameters, then it takes an order of $6k$ operations to compute inner products between real–fake, real–real, and fake–fake gradients, and an order of $3k$ operations to form $\nabla\mathcal{L}_D^{aw}$, totaling to an order of $9k$ operations for Algorithms 1 and 2. This is a fraction of the total computational complexity for one training iteration.



(a) CIFAR-10 (b) STL-10 (c) CIFAR-100

Figure 2.3: Samples randomly generated by aw-AutoGAN model.

2.1.4 Experiments and Results

We implement our Adaptive Weighted Discriminator for SN-GAN [142] and AutoGAN [71] models, and for SN-GAN [142] and BigGAN [18] models, on unconditional and conditional image generating tasks, respectively (commonly referred to as unconditional and conditional GANs). AutoGAN is an architecture based on neural search. In our experiments, we do not invoke a neural search with our aw-loss; we have implemented the aw-method on the model and architecture exactly provided by [71].

We test our method on three datasets: *CIFAR-10* [111], *STL-10* [39], and *CIFAR-100* [111]. The datasets details are provided below.

- The *CIFAR-10* dataset [111] consists of 60,000 color images with 50,000 for training and 10,000 for testing. All images have resolution 32×32 pixels and are divided equally into 10 classes, with 6,000 images per class. No data augmentation;

- The *STL-10* is a dataset proposed in [39] and designed for image recognition and unsupervised learning. STL-10 consists of 100,000 unlabeled images with 96×96 pixels and is split into 10 classes. All images are resized to 48×48 pixels, without any other data augmentation;

- The *CIFAR-100* from [111] is a dataset similar to CIFAR-10 that consists of 60,000 color 32×32 pixel images that are divided into 100 classes. No data augmentation.

We follow the original implementations of SN-GAN, AutoGAN and BigGAN-PyTorch [17] that use the following hyperparameters:

- *Generator:* learning rate: 0.0002; batch size: 128 (SN-GAN, AutoGAN) and 50 (BigGAN); optimizer: Adam optimizer with $\beta_1 = 0$ and $\beta_2 = 0.999$ [109]; loss: hinge [121, 184]; spectral normalization: False; learning rate decay: linear; # of training epochs: 320 (SN-GAN), 300 (AutoGAN), and 1000 (BigGAN);

- *Discriminator:* learning rate: 0.0002; batch size: 64 (SN-GAN, AutoGAN) and 50 (BigGAN); optimizer: Adam optimizer with $\beta_1 = 0$ and $\beta_2 = 0.999$; loss: hinge; spectral normalization: True; learning rate decay: linear; training iterations ratio: 3 (SN-GAN) and 2 (AutoGAN, BigGAN).

Experiments based on SN-GAN and AutoGAN models are performed on a single NVIDIA® QUADRO® P5000 GPU running Python 3.6.9 with PyTorch v1.1.0 for AutoGAN-based models and Chainner v4.5.0 for SN-GAN based models. Experiments based on BigGAN-Pytorch [17] model are performed on two NVIDIA® Tesla® V100 GPU running Python 3.6.12 with PyTorch v1.4.0.

The above-mentioned models train the discriminator by minimizing the negative hinge loss [121, 184]. Our aw-loss also uses the negative hinge loss as follows:

$$\mathcal{L}_D^{aw} = -\, w_r \cdot \mathbb{E}_{x\sim p_d}\big[\min(0, D(x)-1)\big] - w_f \cdot \mathbb{E}_{z\sim p_z}\big[\min(0, -1-D(G(z))\big], \quad (2.44)$$

with w_r and w_f updated every iteration using either Algorithm 1 with weights normalization or Algorithm 2 without it.

To evaluate the performance of the models, we employ the widely used Inception Score [165] (IS) and Fréchet Inception Distance [85] (FID) metrics; see [127] for more details. Proposed in [165], IS is one of the most popular methods for evaluating GAN's performance. It uses a pre-trained ImageNet dataset V3 network model for image classification to classify the generated images. FID [85] is another popular method for evaluating the performance of GAN models.

We compute these metrics every 5 epochs and we report the best IS and FID achieved by each model within the 320 (SN-GAN), 300 (AutoGAN), and BigGAN (1,000) training epochs as in the corresponding original works.

We first present the results for the unconditional GAN for the datasets CIFAR-10, STL-10, and CIFAR-100 in Tables 2.1, 2.2, and 2.3, respectively. In addition to baseline results, we have included the top published results for each dataset for comparison purposes.

Table 2.1: **Results for Unconditional GAN on CIFAR-10 dataset**

Method	IS ↑	FID ↓
ProbGAN [79]	7.75	24.6
Imp. MMD GAN [198]	8.29	16.21
MGAN [87]	8.33±.10	26.7
Dist-GAN [186]	-	17.61
Progressive GAN [104]	8.80±.05	-
SS-GAN [33]	-	19.73
MSGAN [187]	-	11.40
SRN-GAN [168]	8.53±.04	19.57
StyleGAN2 [105]	**9.21±.09**	**8.32**
SN-GAN [142]	8.22±.05	21.7
aw-SN-GAN (ours; Algorithm 1)	8.53±.11	12.32
aw-SN-GAN (ours; Algorithm 2)	8.43±.07	12.65
AutoGAN [71]	8.55±.10	12.42
aw-AutoGAN (ours; Algorithm 1)	9.01±.03	11.82
aw-AutoGAN (ours; Algorithm 2)	8.98±.06	13.17

Our methods significantly improve the baseline results for CIFAR-10 in Table 2.1. Indeed, our aw-AutoGAN achieves the IS substantially above all comparisons other than StyleGAN2. StyleGAN2 outperforms aw-AutoGAN but uses 26.2M parameters vs. 5.4M for aw-AutoGAN.

Table 2.2: **Results for Unconditional GAN on STL-10 dataset**

Method	IS ↑	FID ↓
ProbGAN [79]	8.87±.09	46.74
Imp. MMD GAN [198]	9.34	37.63
MGAN [87]	9.22±.11	-
Dist-GAN [186]	-	36.19
MSGAN [187]	-	27.10
SN-GAN [142]	9.10±.04	40.10
aw-SN-GAN (ours; Algorithm 1)	**9.53±.10**	36.41
aw-SN-GAN (ours; Algorithm 2)	9.61±.12	34.72
AutoGAN [71]	9.16±.12	31.01
aw-AutoGAN (ours; Algorithm 1)	9.41±.09	**26.32**
aw-AutoGAN (ours; Algorithm 2)	9.59±.14	27.97

For STL-10 in Table 2.2, our methods also significantly improve SN-GAN and AutoGAN baseline results. Our aw-SN-GAN achieved the highest IS and aw-AutoGAN achieved the lowest FID score among comparisons.

Table 2.3: **Results for Unconditional GAN on CIFAR-100 dataset:** * - results from our test; † - quoted from [187].

Method	IS $\uparrow$	FID $\downarrow$
SS-GAN [33]	-	21.02†
MSGAN [187]	-	19.74
SRN-GAN [168]	8.85	19.55
SN-GAN [142]	8.18$\pm$.12*	22.40*
aw-SN-GAN (ours; Algorithm 1)	8.31$\pm$.02	19.08
aw-SN-GAN (ours; Algorithm 2)	8.30$\pm$.11	19.48
AutoGAN [71]	8.54$\pm$.10*	19.98*
aw-AutoGAN (ours; Algorithm 1)	**8.90$\pm$.06**	**19.00**
aw-AutoGAN (ours; Algorithm 2)	8.72$\pm$.05	19.89

For CIFAR-100 in Table 2.3, our methods improve the IS significantly for Auto-GAN but modestly for SN-GAN. Our aw-Auto-GAN achieved the highest IS and the lowest FID score among comparisons.

We have also included some visual examples that were randomly generated by our aw-Auto-GAN model in Figure 2.3. We also consider the convergence of our method against training epochs by plotting in Figure 2.2 the IS and FID scores of 50,000 generated samples at every 5 epochs for AutoGAN vs. aw-AutoGAN. Our model consistently achieves faster convergence than the baseline for all the datasets.

We next consider our aw-method for a class conditional image generating task using two base models, SN-GAN [142] and BigGAN [18], on CIFAR-10 and CIFAR-100 datasets. Results are listed in Table 2.4.

Table 2.4: **Results for Conditional GAN on CIFAR-10 and CIFAR-100 datasets:** † - quoted from [173]; BigGAN [18] CIFAR-100 results of our tests using code from [17].

	CIFAR-10		CIFAR-100	
Method	IS ↑	FID ↓	IS ↑	FID ↓
FQGAN [217]	8.48±.03	5.59	9.59±.04	7.42
cGAN [143]	8.62	17.5	9.04	23.2
CRGAN [216]	-	11.48	-	-
MHinge [106]	**9.58±.09**	7.50	**14.36±.17**	17.3
SNGAN [142]	8.60±.08	17.5	$9.30^\dagger$	$15.6^\dagger$
aw-SNGAN (ours; Algorithm 1)	9.03±.11	8.11	9.48±.13	14.42
aw-SNGAN (ours; Algorithm 2)	9.00±.12	8.03	9.44±.16	14.00
BigGAN [18]	9.22	14.73	10.99±.14	11.73
aw-BigGAN (ours; Algorithm 1)	9.52±.10	**7.03**	11.22±.17	**10.23**
aw-BigGAN (ours; Algorithm 2)	9.50±.07	6.89	11.26±.20	10.25

Table 2.4 shows that our method works well for conditional GAN. The aw-method significantly improves the SN-GAN and BigGAN baselines. Indeed, our aw-BigGAN achieved the best FID for both CIFAR-10 and CIFAR-100 among comparisons.

2.1.5 Exploratory & Ablation Studies

In this section, we present four studies to illustrate potential problems of equally weighted GAN loss, the advantages of our adaptive weighted loss, and analysis on newly introduced hyperparameters. The hinge loss is implemented in the first, second, and fourth studies, and a binary cross-entropy loss function is used for the third.

2.1.5.1 Angles between Gradients

In the first study, we examine the angles between $\nabla\mathcal{L}_r$, $\nabla\mathcal{L}_f$, $\nabla\mathcal{L}_D$ (or $\nabla\mathcal{L}_D^{aw}$). We use the CIFAR-10 dataset with the DCGAN architecture [158] and look at 50 iterations in the first epoch of training. In Figure 2.4, we plot the following 3 angles: $\angle_2(\nabla\mathcal{L}_r, \nabla\mathcal{L}_f)$, $\angle_2(\nabla\mathcal{L}_r, \nabla\mathcal{L}_D)$ and $\angle_2(\nabla\mathcal{L}_f, \nabla\mathcal{L}_D)$ against iterations for the original loss $\mathcal{L}_D$ (2.1) on the top and for the aw-loss $\mathcal{L}_D^{aw}$ on the bottom. For the original loss (Left), $\angle_2(\nabla\mathcal{L}_r, \nabla\mathcal{L}_f)$ (blue) stays greater then 90°, closer to 180°. $\angle_2(\nabla\mathcal{L}_r, \nabla\mathcal{L}_D)$ (green) often goes above 90°, and so the training is often done to decrease the real loss. $\angle_2(\nabla\mathcal{L}_f, \nabla\mathcal{L}_D)$ also goes above 90°, though to a lesser extent. With the aw-loss (Right), $\angle_2(\nabla\mathcal{L}_r, \nabla\mathcal{L}_D^{aw})$ and $\angle_2(\nabla\mathcal{L}_f, \nabla\mathcal{L}_D^{aw})$ stay below the 90° line and indicate that we train in the direction of $\nabla\mathcal{L}_r$ and orthogonal to $\nabla\mathcal{L}_f$ in most steps.

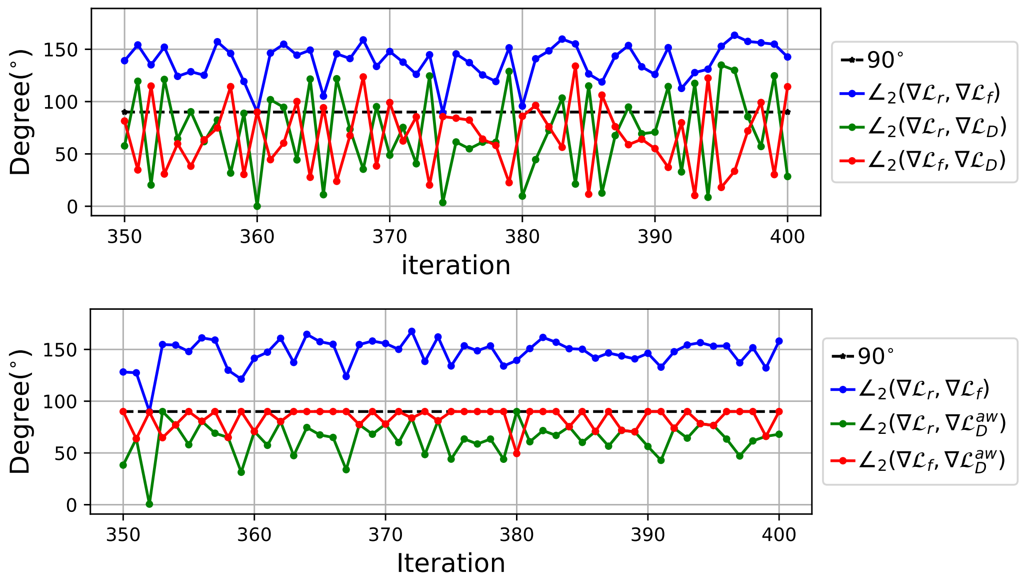


Figure 2.4: Angles between gradients at each iteration. Top: original loss; Bottom: aw-loss.

2.1.5.2 Real Discriminator Scores and Real-Fake Gap after Training

Our second experiment is an ablation study to show that aw-loss increases the discriminator scores for real data and increases the gap between real and fake discriminator scores. We again apply the DCGAN model with the original loss $\mathcal{L}_D$ to CIFAR-10. At every iteration, we examine the mean discriminator score for the mini-batch of the real set and the mean discriminator scores for the mini-batch of the fake dataset generated by the generator. We use the logit output of the discriminator network as the score. We plot these two mean scores against each iteration before training in the first row of Figure 2.5 and after training (with the original loss $\mathcal{L}_D$) in the second row. At each of the above training iterations, we replace $\mathcal{L}_D$ by the aw-loss $\mathcal{L}_D^{aw}$ (2.2) and train for one iteration with the same training mini-batch. We plot the mean discriminator scores for the mini-batches of the real and fake datasets after this training in the third row of Figure 2.5. We further present the gaps between the two scores before training and after training using the original loss and using aw-loss in the fourth row of Figure 2.5.

Figure 2.5 shows that training with aw-loss leads to higher real discriminator scores (0.921 epoch average) than training with the original loss (0.248 epoch average). The average gap between real and fake scores is also larger, with the aw-loss at 1.413 vs. 1.262 of the original loss. Therefore, with the same model and training mini-batch, the aw-loss produces higher discriminator scores for the real dataset and larger gaps between real and fake scores. These are two essential properties of a discriminator for generator training.

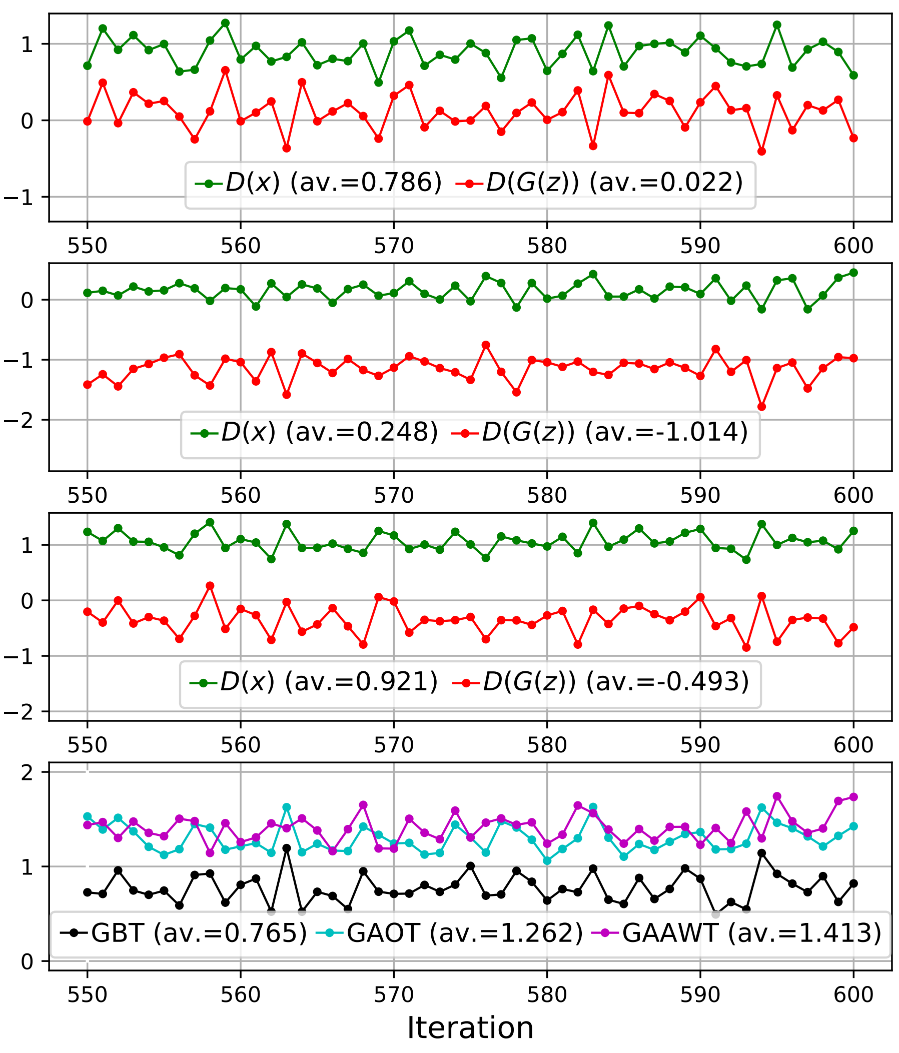


Figure 2.5: Mean discriminator scores for real data $D(x)$ and fake data $D(G(z))$ (Row 1: before training, Row 2: after original training with $\mathcal{L}_D$, Row 3: after training with aw-loss $\mathcal{L}_D^{aw}$) and their gap (Row 4: GBT - gap before training; GAOT - gap after original training; GAAWT - gap after aw-loss training)

2.1.5.3 Instability and Real Discriminator Scores

Our third study examines the benefits of high discriminator scores for a real dataset regarding instability and mode collapse of GAN training. We use a synthetic dataset with a mixture of Gaussian distributions to test unrolled GAN in [140]. The dataset consists of eight 2D Gaussian distributions centered at eight equally distanced points on the unit circle. We train with a plain GAN as in [140] and we plot samples of (fake) data generated by the generator every 5,000 iterations on the first row of Figure 2.6. We see the generated data converges to two or three points but then moves off, demonstrating instability and mode collapse. To understand this phenomenon, at each of the iterations that we study in Figure 2.6, we generate 100 (real) data points

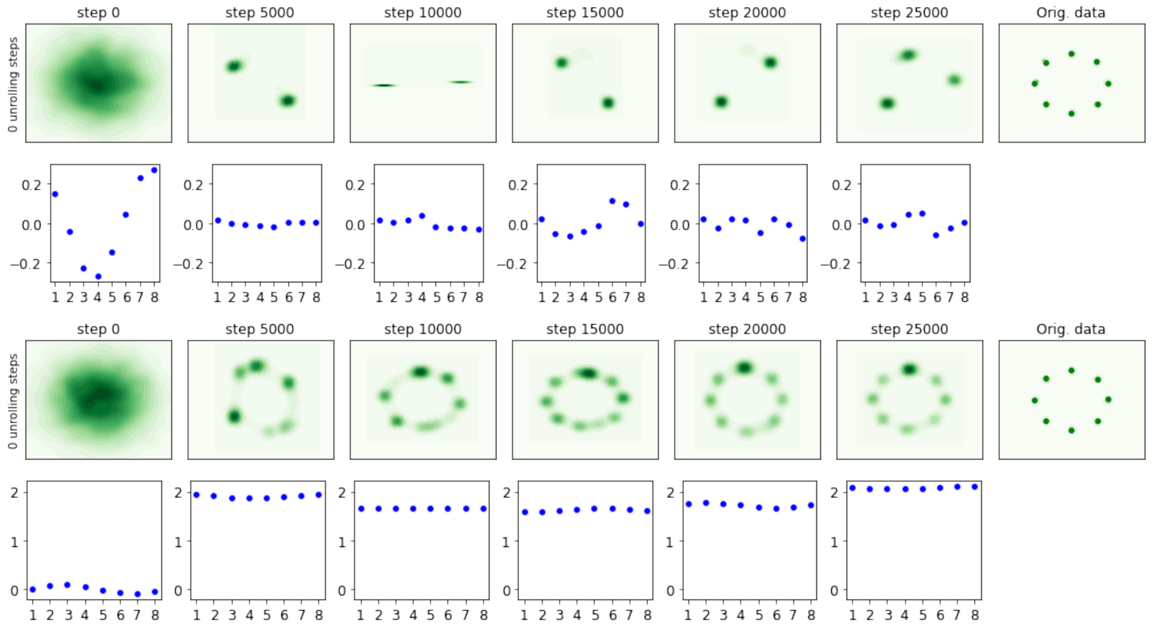


Figure 2.6: Mixture of eight 2D Gaussian distributions centered at 8 points (rightmost column). Row 1: GAN sample points produced by generators; Row 2: GAN mean discriminator scores for each of 8 classes; Row 3: aw-GAN sample points produced by generators; Row 4: aw-GAN mean discriminator scores for each of eight classes;

from each of the eight classes and compute their mean discriminator scores (as the logit output of the discriminator). We plot the mean scores against the classes in the second row of Figure 2.6. We observe that the discriminator scores for the real data do not increase much during training, staying around 0, which corresponds to 0.5 probability after the logistic sigmoid function. The scores are also uneven among different classes. We believe these cause the instability in the generator training.

We compare the GAN results with aw-GAN that applies our adaptive weighted discriminator to the plain GAN. We present the corresponding plots of generated data points (fake) in the third row of Figure 2.6 and the corresponding discriminator scores on the eight classes in the bottom row. In this case, the generator gradually converges to all eight classes and the discriminator scores stay high for all eight classes. Even though the generator started converging to a few classes (step 5,000), the discriminator scores remained high for all classes. Then the generator continues to converge while convergence to other classes occurs. We believe the high real discriminator scores maintain stability and prevent mode collapse in this case.

2.1.5.4 Study of α_1 and α_2 parameters

In our last study, we have considered the choice of α_1 and α_2 by experimenting with aw-AutoGAN models on the CIFAR-10 dataset. We recorded IS and FID scores after the first and the fifth epochs, where the models were trained with the parameters in the grids shown in Figure 2.7 with all other settings fixed. We present the IS and FID

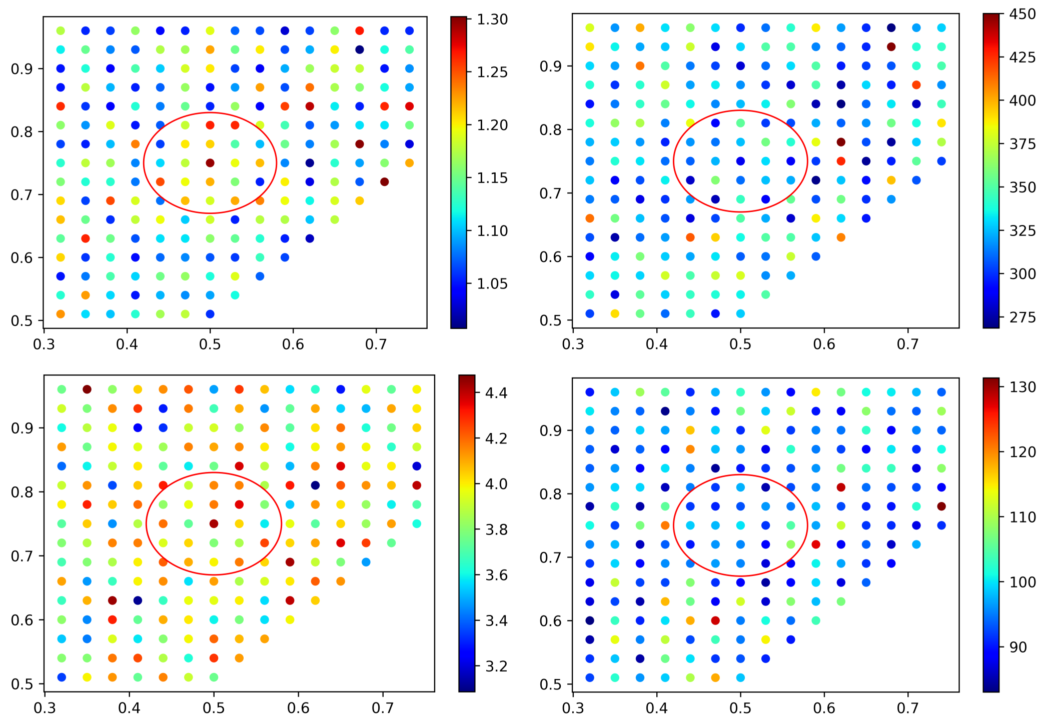


Figure 2.7: Top-left: IS grid after the first epoch; Bottom-left: IS grid after the fifth epoch; Top-right: FID grid after the first epoch; Bottom-right: FID grid after the fifth epoch.

scores in heat map plots in Figure 2.7. The results show that neighbors around the point $(0.5, 0.75)$ lead to good scores; the point $(0.5, 0.75)$ produces one of the highest IS and one of the lowest FID. In particular, the performance is not too sensitive to the selections.

2.1.6 Conclusion

Section 2.1 pinpoints the potential negative effects of traditional GAN training on real loss (and fake loss). It points out that this is a potential cause of instability and mode collapse. We have proposed the Adaptive Weighted Discriminator method to remedy these issues to increase and maintain high real loss. Our experiments demonstrate the benefits and the competitiveness of this method. This work has been published in [215].

2.2 SCP-GAN: Self-Correcting Discriminator Optimization for Training Consistency Preserving Metric GAN on Speech Enhancement Tasks

In recent years, Generative Adversarial Networks (GANs) have produced significantly improved results in speech enhancement (SE) tasks. They are difficult to train, however. In this work, we introduce several improvements to the GAN training schemes, which can be applied to most GAN-based SE models. We propose using consistency loss functions, which target the inconsistency in time and time-frequency domains caused by Fourier and Inverse Fourier Transforms. We also present self-correcting optimization for training a GAN discriminator on SE tasks, which helps avoid "harmful" training directions for parts of the discriminator loss function. We have tested our proposed methods on several state-of-the-art GAN-based SE models and obtained consistent improvements, including new state-of-the-art results for the Voice Bank+DEMAND dataset.

2.2.1 Introduction

Speech Enhancement (SE) is a process of making deteriorated speech signals more understandable and perceptually pleasing. The SE has been widely used for various applications, including mobile communication, speech recognition systems, hearing aids, etc. SE as an area of research interest has been around for several decades. Traditional SE techniques [13, 120] often use a heuristic or straightforward signal processing algorithm to estimate a gain function, which is then applied to the noisy input to produce improved speech. Recent developments in deep learning have inspired many Deep Neural Network (DNN)-based SE techniques [19, 58, 155, 185, 203] that outperform conventional signal processing-based methods. One particular DNN-based architecture, Generative Adversarial Net (GAN), has garnered much interest in the SE community for the past few years [23, 58, 59, 155]. In the applications of SE, GAN architecture is primarily employed to generate enhanced speech. One of the earliest works where GAN models were implemented on the SE domain is the SEGAN [155] model. It utilizes an adversarial framework to map the noisy waveform to a corresponding enhanced speech. Later, MetricGAN [58] introduced a metric score optimization scheme, where an evaluated metric was introduced into adversarial loss functions, replacing a traditional binary-classifier [155] and creating a new branch for SE GAN-based research. There have been several improvements to the MetricGAN model, e.g. MetricGAN+ [59], iMetricGAN [118], CMGAN [23], etc. More recently, with a rise of Transformers [192] and Conformers [73], models such as DB-AIAT [214], DPT-FSNet [44], SE-Conformer [107], CMGAN [23], etc. show significant improvements on SE tasks.

Despite much work, training of GAN-based models is prone to problems such as non-convergence, overfitting, and gradient instabilities. One common issue in GAN's discriminator training is potentially "harmful" gradient direction [215] where parts of the model might train opposite to the desired direction. To overcome this problem, we propose a new method called Self-Correcting (SC) Discriminator Optimization. At the same time, the SE DNN-based models are subject to problems caused by the signal-processing tools, e.g., an inconsistency in the Short-Time Fourier Transform (STFT) and its inverse (iSTFT) [112, 203]. Inspired by [15], we adapt and introduce the consistency loss function as a part of Consistency Preserving (CP) Net into the GAN framework, where loss and architecture take into account the iSTFT effects. From our experiments, the combination of SC and CP methods improves the SE GAN-based models even further than either single method; we call such a combination SCP-GAN.

The remainder of this paper is laid out as follows. In section 2.2.2, we list earlier works pertinent to our work. In section 2.2.3, we introduce improvements to current GAN-based SE models. We present and compare the SCP-GAN results on Voice Bank+DEMAND dataset [190] to the current state-of-the-art (SOTA) models in section 2.2.4. Then, in section 2.2.6, we provide an extensive ablation study to show the advantages of the proposed methods. Finally, in section 2.2.3, we highlight the methods' contributions to the field.

2.2.2 Related Work

2.2.2.1 Adaptively Weighted GAN (awGAN)

The discriminator plays a very important role in training GAN-based models. However, optimizing the discriminator loss function(s) has been a challenge [215]. In the image generation domain, most discriminator loss functions have the following form:

$$\mathcal{L}_D = \mathcal{L}_r + \mathcal{L}_f, \tag{2.45}$$

where $\mathcal{L}_r$ is the part that only relies on the original dataset; [215] calls it the 'real part'. $\mathcal{L}_f$ depends on the generator network, its output, and not the original data; [215] calls this one the 'fake part'. However, the training with $\mathcal{L}_D$ is not performed equally on the real and fake parts, but it depends on the angle between $\nabla\mathcal{L}_r$ and $\nabla\mathcal{L}_f$ and their magnitudes. Under such conditions, the actual training direction $\nabla\mathcal{L}_D$ might end up being in the opposite direction to either $\nabla\mathcal{L}_r$ or $\nabla\mathcal{L}_f$, which is undesirable. To solve such issue [215] proposed the method of adaptive weights for the discriminator loss function:

$$\mathcal{L}_D^{aw} = w_r\mathcal{L}_r + w_f\mathcal{L}_f, \tag{2.46}$$

and the algorithm for choosing these weights.

2.2.2.2 STFT Consistencies in SE DNN models

In audio signal processing, the short-time Fourier transform (STFT) is one of the most fundamental and widely used methods. Most DNN-based SE models [23, 112, 203] use a complex-valued STFTs generator to suppress noise and preserve speech. However, using STFT methods has its issues. One of those issues is the STFT consistency. This is an issue when a loss function does not consider iSTFT signal reconstruction.

Several works have been done to resolve this issue. [112] presented an algorithm for a phase reconstruction based on a local approximation of the consistency constraints.

Adding simple differentiable projection layers to the enhancement DNN to solve the issue was proposed by [203]. More recently, [15] introduced the iSTFT into back-propagation methods for SE DNN-based models.

2.2.3 SCP-GAN

We propose the following two innovative learning strategies to enhance the performance of the SE GAN-based models.

2.2.3.1 Self-Correcting Discriminator Optimization

Notation: The angle between two gradients $\nabla\mathcal{L}_\alpha$ and $\nabla\mathcal{L}_\beta$ is defined as

$$\angle_2\left(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\right) = \cos^{-1}\left(\frac{\langle\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\alpha\|_2\,\|\nabla\mathcal{L}_\beta\|_2}\right), \tag{2.47}$$

where $\langle\cdot,\cdot\rangle_2$ and $\|\cdot\|_2$ denote the Euclidean inner product and the Euclidean 2-norm, respectively.

We introduce the Self-Correcting (SC) Discriminator Optimization method, a generalization of the method from [215] to the SE domain. A large number of existing SE GAN-based models have the discriminator loss function consisting of either two [23, 58] or three [59] equally weighted parts:

$$\mathcal{L}_D = \mathcal{L}_C + \mathcal{L}_E \tag{2.48}$$

$$\mathcal{L}_D = \mathcal{L}_C + \mathcal{L}_E + \mathcal{L}_N, \tag{2.49}$$

where $\mathcal{L}_C$, $\mathcal{L}_E$, and $\mathcal{L}_N$ exclusively rely on clean, enhanced, and noisy datasets, respectively; for example, MetricGAN [58] has a two-part discriminator loss with

$$\mathcal{L}_C = \mathbb{E}_y\left(D(y,y) - Q(y,y)\right)^2 \tag{2.50}$$

$$\mathcal{L}_E = \mathbb{E}_{x,y}\left(D(G(x),y) - Q(G(x),y)\right)^2, \tag{2.51}$$

where x is a noisy signal, y is its corresponding clean version, and $D(\cdot,\cdot)$, $G(\cdot)$, and $Q(\cdot,\cdot)$ are the discriminative model, generative model, and evaluation metric function, respectively. Moreover, notations $\mathbb{E}_y$ and $\mathbb{E}_{x,y}$ denote the expectation over $\{y\}$ and $\{(x,y)\}$, respectively. In such a setup [58], G only takes the noisy signal while both D and Q take two inputs, either (y,y) (as in (2.50)) or $(G(x),y)$ (as in (2.51)) for training D to approximate Q on clean and enhanced signals, respectively.

However, gradient descent training with $\nabla\mathcal{L}_D$ is not performed equally on clean and enhanced parts; its effect depends on the angle between $\nabla\mathcal{L}_C$ and $\nabla\mathcal{L}_E$ and their magnitudes. For example, if the angle between $\nabla\mathcal{L}_C$ and $\nabla\mathcal{L}_E$ is a large obtuse angle and $\|\nabla\mathcal{L}_C\|_2 >> \|\nabla\mathcal{L}_E\|_2$, then $\nabla\mathcal{L}_D$ would make an obtuse angle with $\nabla\mathcal{L}_E$ and thus training along $\nabla\mathcal{L}_D$ would increase the loss $\mathcal{L}_E$, which would be harmful to the enhanced part of the model. In addition, MetricGAN+ [59] uses the $\mathcal{L}_N$ of the following form

$$\mathcal{L}_N = \mathbb{E}_{x,y}\left(D(x,y) - Q(x,y)\right)^2. \tag{2.52}$$

To address this issue, as in [215] for GAN, we introduce weights into the two-part discriminator loss in Equation (2.48),

$$\mathcal{L}_D^{SC} = w_C\mathcal{L}_C + w_E\mathcal{L}_E \tag{2.53}$$

and call it SC$_2$ - Self-Correcting two terms discrimination loss. Moreover, we introduce weights into the three-part discriminator in Equation (2.49)

$$\mathcal{L}_D^{SC} = w_C\mathcal{L}_C + w_E\mathcal{L}_E + w_N\mathcal{L}_N \tag{2.54}$$

and call it SC$_3$ - Self-Correcting three terms discrimination loss. We choose the weights so that $\nabla\mathcal{L}_D^{SC}$ does not make an obtuse angle with any of $\nabla\mathcal{L}_C$, $\nabla\mathcal{L}_E$, or $\nabla\mathcal{L}_N$. While for the SC$_2$ method, weights can be easily generalized from the aw-GAN algorithm in [215]; for the SC$_3$ method, determination of the weights is much more complicated involving many cases. We have analyzed all possible cases and derived corresponding formulas for each scenario. The following Theorem summarizes them.

Theorem 4. *Consider Self-Correcting discriminator loss function*

$$\mathcal{L}_D^{SC} = w_C\mathcal{L}_C + w_E\mathcal{L}_E + w_N\mathcal{L}_N = \sum_{i\in\mathcal{I}} w_i\mathcal{L}_i \tag{2.55}$$

*where $\mathcal{I} = \{C, E, N\}$ index represents **C**lean, **E**nhanced, and **N**oisy datasets, respectively.*

1. *(no obtuse angles) If $\angle_2\left(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\right) \leq 90^\circ$ for any $\alpha \in \mathcal{I}$ and $\beta \in \mathcal{I}\setminus\{\alpha\}$, then*

$$\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\eta\right) \leq 90^\circ \tag{2.56}$$

for any $\eta \in \mathcal{I}$ and any $w_C, w_E, w_N \geq 0$.

2. *(One obtuse angle) Let $\alpha \in \mathcal{I}$, $\beta \in \mathcal{I}\setminus\{\alpha\}$, and $\gamma \in \mathcal{I}\setminus\{\alpha,\beta\}$.*

   a) *(Opposite to $\nabla\mathcal{L}_\alpha$) If $\angle_2\left(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\right) \leq 90^\circ$, $\angle_2\left(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\right) \leq 90^\circ$, and $\angle_2\left(\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\right) \geq 90^\circ$, then*

$$\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\eta\right) \leq 90^\circ \tag{2.57}$$

   for any $\eta \in \mathcal{I}$ and any $w_\alpha, w_\beta \geq 0$ and $w_\gamma = -w_\gamma^* \dfrac{\langle\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\rangle_2}{\|\nabla\mathcal{L}_\gamma\|_2^2}$ where $w_\beta \geq w_\gamma^* \geq w_\beta + w_\alpha \dfrac{\langle\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\rangle_2}{\langle\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\rangle_2}$.

   b) *(Adjacent to $\nabla\mathcal{L}_\alpha$) If $\angle_2\left(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\right) \geq 90^\circ$, $\angle_2\left(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\right) \leq 90^\circ$, and $\angle_2\left(\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\right) \leq 90^\circ$, then*

$$\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\eta\right) \leq 90^\circ \tag{2.58}$$

for any $\eta \in \mathcal{I}$ and any $w_\alpha, w_\gamma \geq 0$ and $w_\beta = -w^*_\beta \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}$ where $w_\alpha \geq w^*_\beta \geq w_\alpha + w_\gamma \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}$.

3. *(Two obtuse angles) Let $\alpha \in \mathcal{I}$, $\beta \in \mathcal{I} \setminus \{\alpha\}$, and $\gamma \in \mathcal{I} \setminus \{\alpha, \beta\}$.*

   a) *(One adjacent and one opposite to $\nabla\mathcal{L}_\alpha$) If $\angle_2(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta) \leq 90°$, $\angle_2(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma) \geq 90°$, and $\angle_2(\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma) \geq 90°$, then*

      i. *If $\cos(\angle_2(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta)) \geq \cos(\angle_2(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma))\cos(\angle_2(\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma))$, then*

$$\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\alpha\right) \leq 90°, \quad \angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\beta\right) \leq 90°, \tag{2.59}$$

$$\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\gamma\right) = 90° \tag{2.60}$$

      for any $w_\alpha, w_\beta \geq 0$ and $w_\gamma = -\frac{w_\alpha \langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\rangle_2 + w_\beta \langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\rangle_2}{\|\nabla\mathcal{L}_\gamma\|_2^2}$.

      ii. *If $\cos(\angle_2(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta)) < \cos(\angle_2(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma))\cos(\angle_2(\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma))$, then $\nabla\mathcal{L}_{\beta'}$ satisfies conditions of 3.-(a)-i. with $\nabla\mathcal{L}_\beta = \nabla\mathcal{L}_{\beta'}$ for any $\nabla\mathcal{L}_{\beta'} := \nabla\mathcal{L}_\alpha + \lambda\nabla\mathcal{L}_\beta$*
      and $0 \leq \lambda \leq \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\rangle_2^2 - \|\nabla\mathcal{L}_\alpha\|_2^2 \|\nabla\mathcal{L}_\gamma\|_2^2}{\|\nabla\mathcal{L}_\gamma\|_2^2 \langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2 - \langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\rangle_2 \langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\rangle_2}$.

   b) *(Both adjacent to $\nabla\mathcal{L}_\alpha$) If $\angle_2(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta) \geq 90°$, $\angle_2(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma) \geq 90°$, and $\angle_2(\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma) \leq 90°$, then*

$$\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\alpha\right) \leq 90°, \quad \angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\beta\right) = 90°, \tag{2.61}$$

$$\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\gamma\right) = 90° \tag{2.62}$$

   for any $w_\alpha > 0$,

$$w_\beta = w_\alpha \left(\frac{\left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma - \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta \right\rangle_2 \langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\left\|\nabla\mathcal{L}_\gamma - \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\right\|_2^2 \|\nabla\mathcal{L}_\beta\|_2^2} - \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2} \right),$$

   and $w_\gamma = -w_\alpha \frac{\left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma - \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta \right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma - \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\right\|_2^2}$.

4. *(No acute angles) If $\angle_2(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta) \geq 90°$ for any $\alpha \in \mathcal{I}$ and $\beta \in \mathcal{I} \setminus \{\alpha\}$, then*

$$\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\alpha\right) \leq 90°, \quad \angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\beta\right) = 90°, \tag{2.63}$$

$$\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\gamma\right) = 90° \tag{2.64}$$

for any $w_\alpha > 0$,

$$w_\beta = w_\alpha \left(\frac{\left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma - \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta \right\rangle_2 \langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\left\| \nabla\mathcal{L}_\gamma - \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta \right\|_2^2 \|\nabla\mathcal{L}_\beta\|_2^2} - \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2} \right),$$

and $w_\gamma = -w_\alpha \dfrac{\left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma - \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta \right\rangle_2}{\left\| \nabla\mathcal{L}_\gamma - \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta \right\|_2^2}.$

Proof.

1. Let $w_C, w_E, w_N \geq 0$ and fix $\eta \in \mathcal{I}$, then

$$\mathcal{L}_D^{SC} = \sum_{i\in\mathcal{I}} w_i \mathcal{L}_i \quad \text{and} \quad \nabla\mathcal{L}_D^{SC} = \sum_{i\in\mathcal{I}} w_i \nabla\mathcal{L}_i \tag{2.65}$$

and

$$\left\langle \sum_{i\in\mathcal{I}} w_i \nabla\mathcal{L}_i, \nabla\mathcal{L}_\eta \right\rangle_2 = \left\langle \sum_{i\in\mathcal{I}\setminus\{\eta\}} w_i \nabla\mathcal{L}_i, \nabla\mathcal{L}_\eta \right\rangle_2 + w_\eta \langle \nabla\mathcal{L}_\eta, \nabla\mathcal{L}_\eta\rangle_2 \tag{2.66}$$

$$= \sum_{i\in\mathcal{I}\setminus\{\eta\}} w_i \langle \nabla\mathcal{L}_i, \nabla\mathcal{L}_\eta\rangle_2 + w_\eta \|\nabla\mathcal{L}_\eta\|_2^2 \geq 0. \tag{2.67}$$

Thus, $\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\eta\right) \leq 90°$ for any $\eta \in \mathcal{I}$ and any $w_C, w_E, w_N \geq 0$.

2. Let $\alpha \in \mathcal{I}$, $\beta \in \mathcal{I}\setminus\{\alpha\}$, and $\gamma \in \mathcal{I}\setminus\{\alpha,\beta\}$.

   a) Let $w_\alpha, w_\beta \geq 0$ and $w_\gamma = -w_\gamma^* \dfrac{\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\rangle_2}{\|\nabla\mathcal{L}_\gamma\|_2^2}$
   for $w_\beta \geq w_\gamma^* \geq w_\beta + w_\alpha \dfrac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\rangle_2}{\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\rangle_2}$. Note that under such conditions on w_γ^*, we have $w_\beta - w_\gamma^* \geq 0$ and

$$w_\gamma^* \geq w_\beta + w_\alpha \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\rangle_2}{\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\rangle_2} \tag{2.68}$$

$$\left(w_\beta - w_\gamma^*\right) \langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\rangle_2 + w_\alpha \langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\rangle_2 \geq 0 \tag{2.69}$$

   and as a consequence

$$w_\alpha \langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\rangle_2 - w_\gamma^* \langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\rangle_2 \geq 0. \tag{2.70}$$

With such weights, we have

$$\mathcal{L}_D^{SC} = w_\alpha \mathcal{L}_\alpha + w_\beta \mathcal{L}_\beta - w_\gamma^* \frac{\langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2}{\|\nabla \mathcal{L}_\gamma\|_2^2} \mathcal{L}_\gamma \tag{2.71}$$

and

$$\nabla \mathcal{L}_D^{SC} = w_\alpha \nabla \mathcal{L}_\alpha + w_\beta \nabla \mathcal{L}_\beta - w_\gamma^* \frac{\langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2}{\|\nabla \mathcal{L}_\gamma\|_2^2} \nabla \mathcal{L}_\gamma. \tag{2.72}$$

Then using Cauchy-Schwarz inequality, (2.69), and (2.70), we have

$$\begin{aligned}
\left\langle \nabla \mathcal{L}_D^{SC}, \nabla \mathcal{L}_\alpha \right\rangle_2 &= w_\alpha \|\nabla \mathcal{L}_\alpha\|_2^2 + w_\beta \langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\alpha \rangle_2 \\
&\quad - w_\gamma^* \frac{\langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2 \langle \nabla \mathcal{L}_\gamma, \nabla \mathcal{L}_\alpha \rangle_2}{\|\nabla \mathcal{L}_\gamma\|_2^2} &(2.73)\\
&\geq \frac{\langle \mathcal{L}_\alpha, \mathcal{L}_\gamma \rangle_2}{\|\nabla \mathcal{L}_\gamma\|_2^2} \left(w_\alpha \langle \nabla \mathcal{L}_\alpha, \nabla \mathcal{L}_\gamma \rangle_2 - w_\gamma^* \langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2 \right) \\
&\quad + w_\beta \langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\alpha \rangle_2 \geq 0, &(2.74)
\end{aligned}$$

$$\begin{aligned}
\left\langle \nabla \mathcal{L}_D^{SC}, \nabla \mathcal{L}_\beta \right\rangle_2 &= w_\alpha \langle \nabla \mathcal{L}_\alpha \nabla \mathcal{L}_\beta \rangle_2 + w_\beta \|\nabla \mathcal{L}_\beta\|_2^2 - w_\gamma^* \frac{\langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2^2}{\|\nabla \mathcal{L}_\gamma\|_2^2} &(2.75)\\
&= w_\alpha \langle \nabla \mathcal{L}_\alpha \nabla \mathcal{L}_\beta \rangle_2 + w_\beta \frac{\|\nabla \mathcal{L}_\beta\|_2^2 \|\nabla \mathcal{L}_\gamma\|_2^2}{\|\nabla \mathcal{L}_\gamma\|_2^2} \\
&\quad - w_\gamma^* \frac{\langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2^2}{\|\nabla \mathcal{L}_\gamma\|_2^2} &(2.76)\\
&\geq w_\alpha \langle \nabla \mathcal{L}_\alpha \nabla \mathcal{L}_\beta \rangle_2 + \frac{\langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2^2}{\|\nabla \mathcal{L}_\gamma\|_2^2} \left(w_\beta - w_\gamma^* \right) \geq 0, &(2.77)
\end{aligned}$$

and

$$\begin{aligned}
\left\langle \nabla \mathcal{L}_D^{SC}, \nabla \mathcal{L}_\gamma \right\rangle_2 &= w_\alpha \langle \nabla \mathcal{L}_\alpha, \nabla \mathcal{L}_\gamma \rangle_2 + w_\beta \langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2 \\
&\quad - w_\gamma^* \frac{\langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2}{\|\nabla \mathcal{L}_\gamma\|_2^2} \|\nabla \mathcal{L}_\gamma\|_2^2 &(2.78)\\
&= w_\alpha \langle \nabla \mathcal{L}_\alpha, \nabla \mathcal{L}_\gamma \rangle_2 + \left(w_\beta - w_\gamma^* \right) \langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2 \geq 0. &(2.79)
\end{aligned}$$

Thus, we can conclude that $\angle_2 \left(\nabla \mathcal{L}_D^{SC}, \nabla \mathcal{L}_\eta \right) \leq 90^\circ$ for any $\eta \in \mathcal{I}$ and any $w_\alpha, w_\beta \geq 0$ and $w_\gamma = -w_\gamma^* \frac{\langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2}{\|\nabla \mathcal{L}_\gamma\|_2^2}$ with $w_\beta \geq w_\gamma^* \geq w_\beta + w_\alpha \frac{\langle \nabla \mathcal{L}_\alpha, \nabla \mathcal{L}_\gamma \rangle_2}{\langle \nabla \mathcal{L}_\beta, \nabla \mathcal{L}_\gamma \rangle_2}$.

b) Let $w_\alpha, w_\gamma \geq 0$ and $w_\beta = -w_\beta^* \frac{\langle \nabla \mathcal{L}_\alpha, \nabla \mathcal{L}_\beta \rangle_2}{\|\nabla \mathcal{L}_\beta\|_2^2}$

for $w_\alpha \geq w_\beta^* \geq w_\alpha + w_\gamma \frac{\langle \nabla \mathcal{L}_\gamma, \nabla \mathcal{L}_\beta \rangle_2}{\langle \nabla \mathcal{L}_\alpha, \nabla \mathcal{L}_\beta \rangle_2}$. Note that under such conditions on w_β^*, we have $w_\alpha - w_\beta^* \geq 0$ and

$$w_\beta^* \geq w_\alpha + w_\gamma \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2} \tag{2.80}$$

$$\left(w_\alpha - w_\beta^*\right)\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2 + w_\gamma \langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2 \geq 0. \tag{2.81}$$

and as a consequence

$$w_\gamma \langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2 - w_\beta^* \langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2 \geq 0. \tag{2.82}$$

With such weights, we have

$$\mathcal{L}_D^{SC} = w_\alpha \mathcal{L}_\alpha - w_\beta^* \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\mathcal{L}_\beta + w_\gamma \mathcal{L}_\gamma \tag{2.83}$$

and

$$\nabla\mathcal{L}_D^{SC} = w_\alpha \nabla\mathcal{L}_\alpha - w_\beta^* \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta + w_\gamma \nabla\mathcal{L}_\gamma. \tag{2.84}$$

Then by Cauchy-Schwarz inequality, (2.81), and (2.82), we have

$$\left\langle \nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\alpha\right\rangle_2 = w_\alpha \|\nabla\mathcal{L}_\alpha\|_2^2 - w_\beta^* \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2^2}{\|\nabla\mathcal{L}_\beta\|_2^2} + w_\gamma \langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\alpha\rangle_2 \tag{2.85}$$

$$\geq \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2^2}{\|\nabla\mathcal{L}_\beta\|_2^2}\left(w_\alpha - w_\beta^*\right) + w_\gamma \langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\alpha\rangle_2 \geq 0, \tag{2.86}$$

$$\begin{aligned}\left\langle \nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\beta\right\rangle_2 &= w_\alpha \langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2 - w_\beta^* \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2 \|\nabla\mathcal{L}_\beta\|_2^2}{\|\nabla\mathcal{L}_\beta\|_2^2} \\ &\quad + w_\gamma \langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2 & (2.87) \\ &= \left(w_\alpha - w_\beta^*\right)\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2 + w_\gamma \langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2 \geq 0, & (2.88)\end{aligned}$$

and

$$\begin{aligned}\left\langle \nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\gamma\right\rangle_2 &= w_\alpha \langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\rangle_2 - w_\beta^* \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2 \langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2} \\ &\quad + w_\gamma \|\nabla\mathcal{L}_\gamma\|_2^2 & (2.89) \\ &\geq w_\alpha \langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\rangle_2 + \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2} \\ &\quad \times \left(w_\gamma \langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2 - w_\beta^* \langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2\right) \geq 0. & (2.90)\end{aligned}$$

Thus, we can conclude that $\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\eta\right) \leq 90°$ for any $\eta \in \mathcal{I}$ and any $w_\alpha, w_\gamma \geq 0$ and $w_\beta = -w_\beta^* \frac{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}$ with $w_\alpha \geq w_\beta^* \geq w_\alpha + w_\gamma \frac{\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}$.

3. Let $\alpha \in \mathcal{I}$, $\beta \in \mathcal{I} \setminus \{\alpha\}$, and $\gamma \in \mathcal{I} \setminus \{\alpha, \beta\}$.

   a) i. Assume that

$$\cos\left(\angle_2\left(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\right)\right) \geq \cos\left(\angle_2\left(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma\right)\right)\cos\left(\angle_2\left(\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\right)\right).$$

Let $w_\alpha, w_\beta \geq 0$, and $w_\gamma = -\frac{w_\alpha \left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma \right\rangle_2 + w_\beta \left\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma \right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}$. With such weights, we have

$$\mathcal{L}_D^{SC} = w_\alpha \mathcal{L}_\alpha + w_\beta \mathcal{L}_\beta - \frac{w_\alpha \left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma \right\rangle_2 + w_\beta \left\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma \right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}\mathcal{L}_\gamma \tag{2.91}$$

and

$$\begin{aligned}\nabla\mathcal{L}_D^{SC} &= w_\alpha \nabla\mathcal{L}_\alpha + w_\beta \nabla\mathcal{L}_\beta \\ &\quad - \frac{w_\alpha \left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma \right\rangle_2 + w_\beta \left\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma \right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}\nabla\mathcal{L}_\gamma.\end{aligned} \tag{2.92}$$

Then

$$\begin{aligned}\left\langle \nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\alpha \right\rangle_2 &= w_\alpha \left\|\nabla\mathcal{L}_\alpha\right\|_2^2 + w_\beta \left\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\alpha \right\rangle_2 \\ &\quad - \frac{w_\alpha \left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma \right\rangle_2 + w_\beta \left\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma \right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}\left\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\alpha \right\rangle_2 && (2.93) \\ &\geq w_\beta \left\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\alpha \right\rangle_2 - \frac{w_\beta \left\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma \right\rangle_2 \left\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\alpha \right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2} && (2.94) \\ &= w_\beta \left\|\nabla\mathcal{L}_\beta\right\|_2 \left\|\nabla\mathcal{L}_\alpha\right\|_2 \Big(\cos\left(\angle_2\left(\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\alpha\right)\right) \\ &\quad - \cos\left(\angle_2\left(\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma\right)\right)\cos\left(\angle_2\left(\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\alpha\right)\right) \Big) \geq 0, && (2.95)\end{aligned}$$

similarly

$$\begin{aligned}\left\langle \nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\beta \right\rangle_2 &= w_\alpha \left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta \right\rangle_2 + w_\beta \left\|\nabla\mathcal{L}_\beta\right\|_2^2 \\ &\quad - \frac{w_\alpha \left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma \right\rangle_2 + w_\beta \left\langle \nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma \right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}\left\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta \right\rangle_2 && (2.96) \\ &\geq w_\alpha \left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta \right\rangle_2 - \frac{w_\alpha \left\langle \nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma \right\rangle_2 \left\langle \nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta \right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2} && (2.97) \\ &= w_\alpha \left\|\nabla\mathcal{L}_\alpha\right\|_2 \left\|\nabla\mathcal{L}_\beta\right\|_2 \Big(\cos\left(\angle_2\left(\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\right)\right)\end{aligned}$$

$$-\cos\left(\angle_2\left(\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right)\right)\cos\left(\angle_2\left(\nabla\mathcal{L}_\gamma,\nabla\mathcal{L}_\beta\right)\right)\Big) \geq 0. \tag{2.98}$$

Lastly,

$$\begin{aligned}\left\langle\nabla\mathcal{L}_D^{SC},\nabla\mathcal{L}_\gamma\right\rangle_2 &= w_\alpha\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2 + w_\beta\left\langle\nabla\mathcal{L}_\beta,\nabla\mathcal{L}_\gamma\right\rangle_2\\ &\quad-\frac{w_\alpha\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2 + w_\beta\left\langle\nabla\mathcal{L}_\beta,\nabla\mathcal{L}_\gamma\right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}\left\|\nabla\mathcal{L}_\gamma\right\|_2^2 = 0.\end{aligned} \tag{2.99}$$

Thus, we can conclude that $\angle_2\left(\nabla\mathcal{L}_D^{SC},\nabla\mathcal{L}_\alpha\right) \leq 90°$, $\angle_2\left(\nabla\mathcal{L}_D^{SC},\nabla\mathcal{L}_\beta\right) \leq 90°$, and $\angle_2\left(\nabla\mathcal{L}_D^{SC},\nabla\mathcal{L}_\gamma\right) = 90°$ for any $w_\alpha, w_\beta \geq 0$
and $w_\gamma = -\dfrac{w_\alpha\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2 + w_\beta\left\langle\nabla\mathcal{L}_\beta,\nabla\mathcal{L}_\gamma\right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}$.

ii. Assume that

$$\cos\left(\angle_2\left(\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\beta\right)\right) < \cos\left(\angle_2\left(\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right)\right)\cos\left(\angle_2\left(\nabla\mathcal{L}_\beta,\nabla\mathcal{L}_\gamma\right)\right).$$

Define $\nabla\mathcal{L}_{\beta'} := \nabla\mathcal{L}_\alpha + \lambda\nabla\mathcal{L}_\beta$ with

$$0 \leq \lambda \leq \frac{\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2^2 - \left\|\nabla\mathcal{L}_\alpha\right\|_2^2\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\beta\right\rangle_2 - \left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2\left\langle\nabla\mathcal{L}_\beta,\nabla\mathcal{L}_\gamma\right\rangle_2}.$$

Note that

$$\lambda \leq \frac{\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2^2 - \left\|\nabla\mathcal{L}_\alpha\right\|_2^2\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\beta\right\rangle_2 - \left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2\left\langle\nabla\mathcal{L}_\beta,\nabla\mathcal{L}_\gamma\right\rangle_2} \tag{2.100}$$

$$\begin{aligned}&\left\|\nabla\mathcal{L}_\alpha\right\|_2^2 - \frac{\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2^2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}\\ &\quad+\lambda\left(\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\beta\right\rangle_2 - \frac{\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2\left\langle\nabla\mathcal{L}_\beta,\nabla\mathcal{L}_\gamma\right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}\right) \geq 0\end{aligned} \tag{2.101}$$

$$\left\langle\nabla\mathcal{L}_\alpha - \frac{\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\alpha + \lambda\nabla\mathcal{L}_\beta\right\rangle_2 \geq 0 \tag{2.102}$$

$$\left\langle\nabla\mathcal{L}_\alpha - \frac{\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2}\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_{\beta'}\right\rangle_2 \geq 0 \tag{2.103}$$

$$\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_{\beta'}\right\rangle_2 \geq \frac{\left\langle\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right\rangle_2\left\langle\nabla\mathcal{L}_\gamma,\nabla\mathcal{L}_{\beta'}\right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma\right\|_2^2} \tag{2.104}$$

$$\cos\left(\angle_2\left(\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_{\beta'}\right)\right) \geq \cos\left(\angle_2\left(\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_\gamma\right)\right)\cos\left(\angle_2\left(\nabla\mathcal{L}_{\beta'},\nabla\mathcal{L}_\gamma\right)\right) \tag{2.105}$$

Moreover $\angle_2\left(\nabla\mathcal{L}_\alpha,\nabla\mathcal{L}_{\beta'}\right) \leq 90°$, $\angle_2\left(\nabla\mathcal{L}_\gamma,\nabla\mathcal{L}_{\beta'}\right) \geq 90°$. Thus, we can apply the 3.-(a)-i. with $\nabla\mathcal{L}_\beta = \nabla\mathcal{L}_{\beta'}$.

b) Let $w_\alpha > 0$, then the weights for w_β and w_γ are obtained using the Gram–Schmidt process on $\{\nabla\mathcal{L}_\beta, \nabla\mathcal{L}_\gamma, w_\alpha\nabla\mathcal{L}_\alpha\}$ (exactly in this order) and where the third Gram–Schmidt vector is set to be $\nabla\mathcal{L}_D^{SC}$:

$$\begin{aligned}\nabla\mathcal{L}_D^{SC} := {}& w_\alpha\nabla\mathcal{L}_\alpha - w_\alpha\frac{\langle\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta \\ & - w_\alpha\frac{\left\langle\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma - \frac{\langle\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma - \frac{\langle\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\right\|_2^2}\nabla\mathcal{L}_\gamma \\ & + w_\alpha\frac{\left\langle\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma - \frac{\langle\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\right\rangle_2 \langle\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\left\|\nabla\mathcal{L}_\gamma - \frac{\langle\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\right\|_2^2 \|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\end{aligned} \tag{2.106}$$

or equivalently

$$\begin{aligned}\nabla\mathcal{L}_D^{SC} := {}& w_\alpha\nabla\mathcal{L}_\alpha \\ & + w_\alpha\frac{\left\langle\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma - \frac{\langle\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\right\rangle_2 \langle\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\left\|\nabla\mathcal{L}_\gamma - \frac{\langle\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\right\|_2^2 \|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta \\ & - w_\alpha\frac{\langle\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta \\ & - w_\alpha\frac{\left\langle\nabla\mathcal{L}_\alpha, \nabla\mathcal{L}_\gamma - \frac{\langle\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\right\rangle_2}{\left\|\nabla\mathcal{L}_\gamma - \frac{\langle\nabla\mathcal{L}_\gamma, \nabla\mathcal{L}_\beta\rangle_2}{\|\nabla\mathcal{L}_\beta\|_2^2}\nabla\mathcal{L}_\beta\right\|_2^2}\nabla\mathcal{L}_\gamma.\end{aligned} \tag{2.107}$$

From the construction, using the Gram–Schmidt process, $\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\alpha\right) \leq 90°$, $\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\beta\right) = 90°$, and $\angle_2\left(\nabla\mathcal{L}_D^{SC}, \nabla\mathcal{L}_\gamma\right) = 90°$.

4. Identical to the proof of 3.-(b).

□

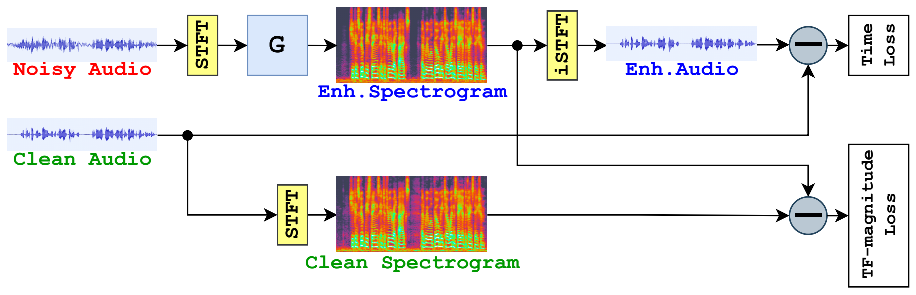
It is important to note that the MetricGAN discriminator loss function(s) has little in common with traditional loss functions [215] (e.g., approximation of the metric vs.

binary classification). Moreover, SE GAN's behavior differs from image-generating GAN since one takes a noisy counterpart of clean speech and the other takes a random sample from a simple distribution. These differences affect their behavior and must be considered when designing algorithms for adaptive decision-making regarding which parts of the loss to prioritize, how to rotate gradients, etc. With that in mind, we propose Algorithm 3, which determines weights for (2.53) and (2.54) using $\nabla\mathcal{L}_C$, $\nabla\mathcal{L}_E$, or $\nabla\mathcal{L}_N$, resulting in a self-correcting discriminator gradient. Similar to [215], the goal of Algorithm 3 is to minimize the potential harm to parts of the model by ensuring the training gradient $\nabla\mathcal{L}_D^{SC}$ does not go obtuse to $\nabla\mathcal{L}_C$, $\nabla\mathcal{L}_E$, or $\nabla\mathcal{L}_N$.

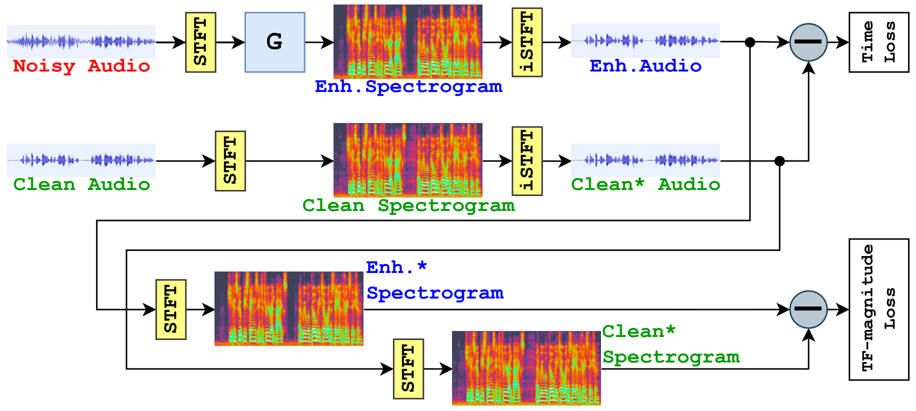
Algorithm 3: Self-Correcting Discriminator Method

```
1:  Compute: $\nabla\mathcal{L}_C$, $\nabla\mathcal{L}_E$, $\nabla\mathcal{L}_N$ if $\mathcal{L}_N$ is used
2:  if $\angle_2(\nabla\mathcal{L}_C, \nabla\mathcal{L}_E) < 90^\circ$ then
3:      $w_C = 1$ and $w_E = 1$
4:      if $\mathcal{L}_N$ is used and $\angle_2(w_C\nabla\mathcal{L}_C + w_E\nabla\mathcal{L}_E, \nabla\mathcal{L}_N) < 90^\circ$ then
5:          $w_N = 1$
6:      else
7:          $w_N = -\frac{\langle\nabla\mathcal{L}_C, \nabla\mathcal{L}_N\rangle_2}{\|\nabla\mathcal{L}_N\|_2^2} - \frac{\langle\nabla\mathcal{L}_E, \nabla\mathcal{L}_N\rangle_2}{\|\nabla\mathcal{L}_N\|_2^2}$
8:      end
9:  else
10:     $w_C = 1$ and $w_E = -\frac{\langle\nabla\mathcal{L}_C, \nabla\mathcal{L}_E\rangle_2}{\|\nabla\mathcal{L}_E\|_2^2}$
11:     if $\mathcal{L}_N$ is used and $\angle_2(w_C\nabla\mathcal{L}_C + w_E\nabla\mathcal{L}_E, \nabla\mathcal{L}_N) < 90^\circ$ then
12:         $w_N = 1$
13:     else
14:         $w_N = -\frac{\langle\nabla\mathcal{L}_C, \nabla\mathcal{L}_N\rangle_2}{\|\nabla\mathcal{L}_N\|_2^2} + \frac{\langle\nabla\mathcal{L}_C, \nabla\mathcal{L}_E\rangle_2 \langle\nabla\mathcal{L}_E, \nabla\mathcal{L}_N\rangle_2}{\|\nabla\mathcal{L}_E\|_2^2 \|\nabla\mathcal{L}_N\|_2^2}$
15:     end
16: end
```

Algorithm 3 is designed using Theorem 4 to ensure that the angle between $\nabla\mathcal{L}_C$ and $\nabla\mathcal{L}_E$ never becomes obtuse while prioritizing the $\nabla\mathcal{L}_C$ direction, which is particularly important in earlier training since the model has not seen much clean data. Towards the end of the training, $\nabla\mathcal{L}_E$ leans towards $\nabla\mathcal{L}_C$ (in both direction and magnitude) as enhanced examples become less noisy; therefore, prioritization of direction would not be necessary. With the add-on of the noisy part, we want to ensure that $\nabla\mathcal{L}_N$ never goes in the opposite direction to $w_C\nabla\mathcal{L}_C + w_E\nabla\mathcal{L}_E$ (after the rotation), which is already a strong direction since the 2-term loss model performs at a good level. Figure 2.8 depictures the aforementioned behavior of $\angle_2(\nabla\mathcal{L}_N, \nabla\mathcal{L}_E)$ and $\angle_2(\nabla\mathcal{L}_C, \nabla\mathcal{L}_E)$ during training.



(a) Process of computing Time and TF-magnitude losses inside the GAN-based SE model Generator (G)



(b) **Consistency Preserving (CP) Net:** Depiction of the process for computing Time and TF-magnitude losses with CP method inside the GAN-based SE model Generator (G)

Figure 2.9: Traditional vs. Consistency Preserving SE GAN-based models

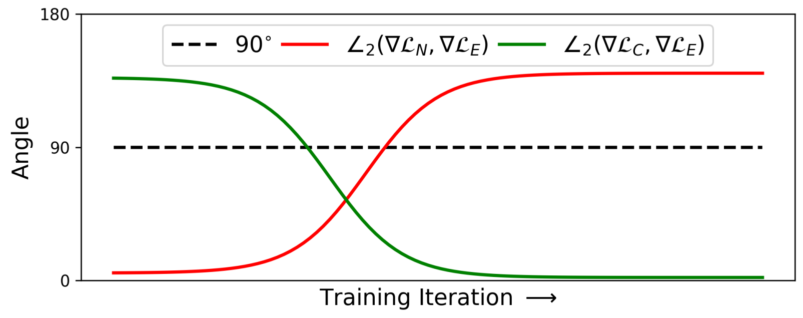


Figure 2.8: Depiction of the behavior of $\angle_2(\nabla\mathcal{L}_N, \nabla\mathcal{L}_E)$ and $\angle_2(\nabla\mathcal{L}_C, \nabla\mathcal{L}_E)$ during the SE MetricGAN's training

Note: When implementing Algorithm 3, it is enough to check the inner product's sign without any angle computation.

2.2.3.2 Consistency Preserving Network

Most GAN-based SE models [23, 44, 58, 214] have a generator (G) that accepts the STFT spectrogram of a noisy waveform as input. The G's output is an enhanced spectrogram that later uses iSTFT to produce the enhanced waveform; Figure 2.9a illustrates the process. The G is then updated using a combination of various loss functions, e.g., Time Loss [3], TF-magnitude Loss [16], Adversarial Metric Loss [58], etc. For example, the TF-magnitude Loss [16] is computed between the enhanced and clean spectrograms; see Figure 2.9a. However, such loss and architectural setup do not consider the effect of the iSTFT reconstruction, which causes inconsistencies between signals.

We incorporate the idea from [15] to SE GAN-based model by modifying architecture and loss function(s) such that any input into a loss function (including the Adversarial Loss) undergoes the same process, taking into consideration the effects of signal reconstruction from the spectrogram; we call such process and loss function a Consistency Preserving (CP) Network and a consistency loss, respectively. Particularly, the CP Net ensures that the same number of STFT and iSTFT transforms are applied on clean, enhanced, and noisy (if used) signals and avoids distortion(s) that could happen on the ends of the audio segments, where the edge regions have insufficient data to reconstruct a signal from the spectrogram with the overlap-add operation. Our approach addresses this issue using the same STFT-iSTFT process and avoids such unexpected behavior. Figure 2.9b depicts the process of computing Time and TF-magnitude losses using the proposed CP method by ensuring that the same transforms are applied on enhanced, clean, and noisy (if used) signals.

Note: The Clean Audio to the Clean* Audio process inside the CP Net (2$^{\text{nd}}$ row of Figure 2.9b) can be performed at the data preprocessing stage.

2.2.4 Experiments and Results

2.2.4.1 Dataset

We use the publicly accessible Voice Bank+DEMAND [190] dataset to evaluate and compare our proposed SCP-GAN method. The training set of the Voice Bank+DEMAND dataset consists of 11,572 individual recordings of 28 speakers from the Voice Bank corpus [193], which are mixed with DEMAND [179] database and some artificial background noises at the signal-to-noise ratios (SNRs) of 0, 5, 10, and 15 dB. The test set has 824 utterances of two speakers from the Voice Bank corpus, mixed with unseen DEMAND noises at the SNRs of 2.5, 7.5, 12.5, and 17.5 dB.

2.2.4.2 Evaluation Metrics

To assess the speech quality, we select a set of widely used metrics, including the Perceptual Evaluation of Speech Quality (PESQ) [161] (ranging between -0.5 and 4.5), the Mean Opinion Score (MOS) [91], prediction of the signal distortion (CSIG), the MOS prediction of the intrusiveness of background noise (CBAK), MOS prediction of the overall effect (COVL) (all MOS metrics range between 1 and 5), the Seg-

Table 2.5: **Performance comparison on Voice Bank+DEMAND dataset [190]:** "-" denotes the results not provided in the original paper; † - quoted from [29]; repro. - our reproduction of experiments

Model	# of Param.	PESQ	CSIG	CBAK	COVL	SSNR	STOI
Noisy Data	n/a	1.97	3.35	2.44	2.63	1.68	0.91
SE-Conformer [107]	-	3.13	4.45	3.55	3.82	-	0.95
MANNER [154]	-	3.21	4.53	3.65	3.91	-	-
DB-AIAT [214]	2.81M	3.31	4.61	3.75	3.96	10.79	0.96
DPT-FSNet [44]	0.91M	3.33	4.58	3.72	4.00	-	0.96
PCS [29]	-	3.35	4.43	-	3.92	-	0.95
MetricGAN+ [59]	2.6M	3.15	4.14	3.16	3.64	-	$0.93^{\dagger}$
MetricGAN+ (repro.)	2.6M	3.08	4.05	3.01	3.60	-	0.92
SCP-MetricGAN+ (ours)	2.6M	3.19	4.20	3.20	3.65	-	0.93
CMGAN [23]	1.83M	3.41	4.63	3.94	4.12	**11.10**	0.96
CMGAN (repro.)	1.83M	3.39	4.62	3.93	4.13	10.61	0.96
SCP-CMGAN (ours)	1.83M	**3.52**	**4.75**	**3.97**	**4.25**	10.82	**0.96**

mental Signal-to-Noise Ratio (SSNR), and the Short-Time Objective Intelligibility (STOI) [176] (with a range 0 to 1). For all metrics, higher numbers denote better performance.

2.2.5 Experimental Results

We have applied our proposed methods to two baseline models: a widely-used MetricGAN+ [59] model and a current SOTA model - CMGAN [23]. Our SCP method shows a consistent improvement over the compared baseline. On the MetricGAN+ model, our SCP-MetricGAN+ improved by 0.04, 0.06, 0.04, and 0.01 on the PESQ, CSIG, CBAK, and COVL metrics, respectively. Our improvements with the SCP-CMGAN model are 0.11, 0.12, 0.03, and 0.13 on the same scores. Moreover, we have compared our method to other recent SOTA models, which can be seen in Table 2.5.

Note: Results provided in Table 2.5 for MetricGAN+ and CMGAN models are quoted from the original papers; however, to verify them, we have obtained our results: MetricGAN+ (repro.) and CMGAN (repro.). The results for the CMGAN (repro.) model are very similar to the results from [23] with one exception - the SSNR metric, where our result is lower: 10.61 (ours) vs. 11.10 [23]. Furthermore, the results for MetricGAN+ (repro.) model are slightly lower than the ones provided in the original paper [59].

Finally, to demonstrate the significance of our best model, we ran a paired T-test between SCP-CMGAN and CMGAN(repro.) (as our baseline) models (`scipy.ttest_rel(scp-cmgan,cmgan(repro.))`) using a VoiceBank-DEMAND test dataset, see Table 2.6 for results. The test confirmed that our results are better than the baseline, with a p-value less than 0.05 for every metric.

Table 2.6: **T-test Statistics and p-value:** Results of a paired T-test between SCP-CMGAN and CMGAN(repro.) models on Voice Bank+DEMAND test dataset [190], a p-value less than 0.05 indicates statistically significant results.

Metric	Statistic	p-value
PESQ	14.889	$1.419 \cdot 10^{-44}$
CSIG	18.402	$1.326 \cdot 10^{-63}$
CBAK	14.565	$6.322 \cdot 10^{-43}$
COVL	19.515	$5.362 \cdot 10^{-70}$
SSNR	4.830	$1.630 \cdot 10^{-6}$
STOI	2.511	$1.222 \cdot 10^{-2}$

2.2.6 Exploratory & Ablation Studies

We have conducted an ablation study to demonstrate the importance of our methods. We have chosen the CMGAN [23] model as the base model due to its SOTA performance at the time of this study. Table 2.7 shows the average results of each model's best performance over three randomly chosen seeds.

First, we have retrained the CMGAN [23] model to verify the results from [23]. The results obtained from our experiments are relatively close to the results stated in [23], except for the SSNR metric where we have obtained slightly lower results, i.e., SSNR of 10.61 (ours) vs. SSNR of 11.10 [23].

Next, we have added Noisy Data (ND) to the CMGAN model discriminator training ('+ ND' in Table 2.7); however, such an addition had slight improvement over the baseline. Following it, we have considered adding our SC method to the baseline model ('+ SC_2' in Table 2.7) and nothing else. With SC_2, we saw some improvements in PESQ, COVL, and SSNR metrics. Furthermore, we have analyzed the advantages of the CP method ('+ CP' in Table 2.7) without any add-ons. The CP shows significant improvements in PESQ, CSIG, and COVL metrics and is comparable in the others.

Then, we combined ND and SC_3 methods ('+ ND, SC_3' in Table 2.7). Such a setup further improves baseline as well as single methods, particularly in COVL and SSNR metrics. A combination of ND and CP methods ('+ ND, CP' in Table 2.7) has the same nature of improvements, producing better results in PESQ, CSIG, and COVL metrics. The last combination of SC_2 and CP methods ('+ SC_2, CP' in Table 2.7) demonstrates that together both proposed methods achieve significant improvements on the SE task. Moreover, this particular model achieved the highest SSNR result of 10.91.

Finally, we have combined ND, SC_3, and CP methods in the model we call SCP-CMGAN in Table 2.5. Adding ND and switching from SC_2 to SC_3 further improves the 'CMGAN + SC_2, CP' model, achieving new SOTA results.

Note that all the above models were trained under the same conditions without changing the hyperparameters and with identical software and hardware settings: Python 3.8.13, PyTorch 1.10, and CUDA 11 on NVIDIA Tesla V100 GPUs.

Table 2.7: **Ablation Study on Voice Bank+DEMAND:** STOI results are equal to 0.96 for all the tests; † - results from our tests; ND - Noisy Data, CP - Consistency Preserving Generator, SC$_2$ - SC with $\mathcal{L}_C$ and $\mathcal{L}_E$, SC$_3$ - SC with $\mathcal{L}_C$, $\mathcal{L}_E$, $\mathcal{L}_N$.

Model	PESQ	CSIG	CBAK	COVL	SSNR
CMGAN (repro.)†	3.39	4.62	3.93	4.13	10.61
+ ND	3.41	4.65	3.92	4.13	10.68
+ SC$_2$	3.44	4.65	3.92	4.17	10.70
+ CP	3.47	4.71	3.93	4.20	10.54
+ ND, SC$_3$	3.43	4.64	3.93	4.18	10.76
+ ND, CP	3.47	4.73	3.93	4.22	10.53
+ SC$_2$, CP	3.49	4.72	3.96	4.24	**10.91**
+ ND, SC$_3$, CP	**3.52**	**4.75**	**3.97**	**4.25**	10.82

2.2.7 Conclusion

This paper presents several improvements to SE GAN-based models. The proposed method of Consistency Preservation reconciles the issue with Fourier and Inverse-Fourier transforms inside the generative models. At the same time, the Self-Correcting Discriminator Optimization method helps with training the discriminative model by avoiding gradient directions that are potentially harmful to the training. Our experiments and ablation study demonstrate the advantages of using one or both of the proposed methods, including new SOTA results for the Voice Bank+DEMAND dataset.

Chapter 3 Recurrent Neural Networks

3.1 NC-GRU: Orthogonal Gated Recurrent Unit with Neumann-Cayley Transformation

In recent years, using orthogonal matrices has been shown to be a promising approach to improving Recurrent Neural Networks (RNNs) with training, stability, and convergence, particularly to control gradients. While Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM) architectures address the vanishing gradient problem by using a variety of gates and memory cells, they are still prone to the exploding gradient problem. In this work, we analyze the gradients in GRU and propose the usage of orthogonal matrices to prevent exploding gradient problems and enhance long-term memory. We study where to use orthogonal matrices and propose a Neumann series-based Scaled Cayley transformation for training orthogonal matrices in GRU, which we call Neumann-Cayley Orthogonal GRU, or simply NC-GRU. We present detailed experiments of our model on several synthetic and real-world tasks, which show that NC-GRU significantly outperforms GRU as well as several other RNNs.

3.1.1 Introduction

One of the preferred neural network models for working with sequential data is the Recurrent Neural Network (RNN) [90, 164]. RNNs can efficiently model sequential data through a hidden sequence of states. However, training vanilla RNNs have obstacles [90, 164], one of which is their susceptibility to vanishing and exploding gradients [10]. In the case of vanishing gradients, the optimization algorithm faces difficulty continuing to learn due to very small changes in the model parameters. In the case of exploding gradients, the training could overstep local minima, potentially causing instabilities such as divergence or oscillatory behavior. It may also lead to computational overflows.

There have been several works studying how to solve these problems. For example, gates have been introduced into the RNN architecture: Long Short-Term Memory (LSTM) [88] and Gated Recurrent Units (GRU) [37]. They can pass long-term information and help to overcome vanishing gradients. In practice, GRU and LSTM models are still prone to the problem of exploding gradients.

More recently, several RNN models have been proposed using unitary or orthogonal matrices for the recurrent weights [7, 50, 81, 99, 100, 128, 141, 195, 204] along with methods to preserve those properties. Introducing such weights into RNN models brought new development into the RNN community. One of the main reasons behind this is a theoretical explanation of why the performance is better when using unitary or orthogonal weights [7]. The key step in these methods is preserving orthogonal or unitary properties at every training iteration. There have been several different techniques for updating the recurrent weights to preserve either orthogonal or unitary properties, including, for example, multiplicative updates [204], Givens rotations [100], Householder reflections [141], Cayley transforms [81, 82, 116, 128], and other similar ideas that have shown effective usage of orthogonal or unitary matrices [7, 84, 92, 169, 177, 196].

In this work, we study the benefits of applying orthogonal matrices to one of the most widely used RNN models, Gated Recurrent Unit (GRU) [37], both theoretically and experimentally. We analyze the gradients of the GRU loss and based on this analysis, we propose the usage of orthogonal matrices in several hidden state weights of the model. We introduce a Neumann series-based Scaled Cayley transform for training the orthogonal weights. Our method utilizes a reliable and stable method of Scaled Cayley transforms, which was studied and used in [81, 128] for training orthogonal weights for RNNs. In addition, we propose a Neumann series approximation of the matrix inverse inside the Cayley transform. Such substitution not only performs on the same or even better level as the traditional inverse (see Section 3.1.5.1 for experiments) but also decreases computation time which might come of particular assistance when working with larger models. We call our method *Neumann-Cayley Orthogonal Gated Recurrent Unit* or *NC-GRU* for simplicity. Experiments show that the proposed method is more stable with faster convergence and produces better results on several synthetic and real-world tasks.

3.1.2 Related Work

Many models have been designed to improve classical RNN [90, 164] for sequential data. They include, for example, the establishment of gates [37, 88], normalization methods [8, 42, 93, 166, 188, 206, 208], and the introduction of unitary and orthogonal matrices into RNN structure [7, 50, 99, 100, 141, 196], etc. This section will discuss some of the works most relevant to our proposed method.

Unitary RNNs (uRNNs) [7] presented an architecture that learns a unitary weight matrix. The construction of the recurrent weight matrix consists of a composition of diagonal matrices, reflection matrices in the complex domain, and Fourier transform. The uRNN model presented in [204] is based on constrained optimization over the space of all unitary matrices rather than a product of parameterized matrices. EUNN [100] is another work that utilizes the product of unitary matrices. The recurrent matrix in this architecture is parameterized with products of Givens Rotations. Also, the representation capacity of the unitary space is fully tunable and ranges from a subspace of unitary matrices to the entire unitary space.

Work by [141] proposed orthogonal RNNs, or simply oRNNs, which involve the application of Householder reflections. Such parameterization of the transition matrix allows efficient training and maintains the orthogonality of the recurrent weights while training. The method introduced in [196] proposes a weight matrix factorization by bounding the matrix norms. Moreover, it controls the degree of gradient expansion during backpropagation. Besides that, this technique enforces orthogonality as well. GORU [99] presented a RNN, an extension of unitary RNN with a gating mechanism. Unitarity is preserved by using the ideas from EURNN [100]. The results compared to GRU [37] are mixed, depending on the task. More recently, [50] presented an embedding of a linear time-invariant system that contains Laguerre polynomials in the model.

3.1.2.1 Gated Recurrent Unit (GRU)

We have studied in depth the Gated Recurrent Unit (GRU) architecture which was proposed in [37] as an alternative to the well-known LSTM [88] cell. The structure of a GRU cell is

$$\begin{aligned} r_t &= \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right) \\ u_t &= \sigma\left(W_u x_t + U_u h_{t-1} + b_u\right) \\ c_t &= \Phi\left(W_c x_t + U_c\left(r_t \odot h_{t-1}\right) + b_c\right) \\ h_t &= \left(1 - u_t\right) \odot h_{t-1} + u_t \odot c_t \end{aligned} \tag{3.1}$$

where W_r, W_u, and W_c are input weights in $\mathbb{R}^{n\times m}$, U_r, U_u, and U_c are recurrent weights in $\mathbb{R}^{n\times n}$, and b_r, b_u and b_c are the bias parameters in $\mathbb{R}^n$. Here, m represents the dimension of the input data, and n represents the dimension of the hidden state. In (3.1), the activation functions σ and Φ are sigmoid and hyperbolic tangent function (tanh) respectively, and $\odot$ is the Hadamard product. In addition, the initial hidden state h_0 is set to zero.

The main difference between GRU and LSTM is the long-term memory implementation not as a separate channel but inside the hidden state h_t itself. GRU has a single gate u_t that controls both forget and input gates. For example, if the output of u_t is 1, the forget gate is open, implying that the input gate is closed. Similarly, if u_t is 0, the forget gate is closed, and the input gate is open. This structure allows GRUs to discard random or insignificant information while grasping the important details.

3.1.2.2 Cayley Transform Orthogonal RNN

We have implemented and studied the effects of orthogonal matrices inside the GRU cell based on the Cayley transforms [177]. Some initial work to use Cayley transform for orthogonal weights in RNNs was presented in [81] together with the scoRNN model. This model introduced a skew-symmetric matrix A, which is used to define an orthogonal matrix W via the Scaled Cayley transform

$$W = \left(I + A\right)^{-1}\left(I - A\right)D, \tag{3.2}$$

where matrix D is a diagonal matrix of ones and negative ones, which scales the traditional Cayley transform [177]. It is proved in [102] that the matrix D, with a suitable choice on the number of negative ones, can avoid a potential problem of eigenvalue(s) of A being negative one(s), making matrix $I + A$ non-invertible. The number of negative ones in matrix D can be considered a tunable hyperparameter. Further, it guarantees that the skew-symmetric matrix A that generates the orthogonal matrix will be bounded.

[81] presents the following process to train the scoRNN model using Scaled Cayley transforms:

$$A^{(k+1)} = A^{(k)} - \lambda \nabla_A L\left(U_{sco}\left(A^{(k)}\right)\right) \tag{3.3}$$

$$U_{sco}^{(k+1)} = \left(I + A^{(k+1)}\right)^{-1}\left(I - A^{(k+1)}\right) D \tag{3.4}$$

where $\nabla_A L\left(U_{sco}\left(A\right)\right)$ is computed using

$$\nabla_A L\left(U_{sco}\left(A\right)\right) = V^T - V, \tag{3.5}$$

with

$$V = \left(I + A\right)^{-T} \nabla_{U_{sco}} L\left(U_{sco}\left(A\right)\right)\left(D + U_{sco}^T\right) \tag{3.6}$$

in which $\nabla_{U_{sco}} L\left(U_{sco}\left(A\right)\right)$ is computed via standard backpropogation methods.

Even though rounding errors may accumulate over several repeated matrix multiplications, orthogonality in scoRNN [81] is maintained to the machine's precision. This property helps to achieve significant improvements over other orthogonal or unitary RNNs for long sequences on several benchmark tasks; see the Experiments section in [81] for more details.

3.1.3 Efficient Orthogonal Gated Recurrent Unit

We now present an efficient orthogonal GRU.

3.1.3.1 Gradient Analysis of Hidden States in GRU

Gradient behavior is important in model training, convergence, stability, and performance. However, when it comes to backpropagation through time for the GRU model from (3.1), the gradients of the loss function $\mathcal{L}$ with respect to intermediate hidden states, weights, and biases can be found from the respective gradients of the final hidden state h_T, which is simplified to finding the gradient of h_t with respect to h_{t-1} for t between 1 and T. Namely,

$$\frac{\partial \mathcal{L}}{\partial h_i} = \frac{\partial \mathcal{L}}{\partial h_T} \prod_{t=i+1}^{T} \frac{\partial h_t}{\partial h_{t-1}}. \tag{3.7}$$

Thus, to analyze the gradients, we consider the gradient of the hidden state h_t with respect to the hidden state h_{t-1} as well as its upper bound in the following theorem.

Theorem 5. *Let h_{t-1} and h_t be two consecutive hidden states from the GRU model stated in (3.1). Then*

$$\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|_2 \leq \alpha + \beta \left\| U_c \right\|_2 \tag{3.8}$$

where

$$\alpha = \delta_u \left(\max_i \left\{ [h_{t-1}]_i \right\} + \max_i \left\{ [c_t]_i \right\} \right) \left\| U_u \right\|_2 + \max_i \left\{ (1 - [u_t]_i) \right\} \tag{3.9}$$

and

$$\beta = \max_i \{[u_t]_i\} \left(\delta_r \|U_r\|_2 \max_i \{[h_{t-1}]_i\} + \max_i \{[r_t]_i\}\right), \tag{3.10}$$

with constants δ_u and δ_r defined as follows:

$$\delta_u = \max_i \{[u_t]_i (1 - [u_t]_i)\} \tag{3.11}$$

and

$$\delta_r = \max_i \{[r_t]_i (1 - [r_t]_i)\}. \tag{3.12}$$

Proof.

Since h_t depends on u_t and c_t (which is also depends on r_t), we will start by finding $\frac{\partial r_t}{\partial h_{t-1}}$, and $\frac{\partial u_t}{\partial h_{t-1}}$, and $\frac{\partial c_t}{\partial h_{t-1}}$ as well as corresponding bounds of these gradients. Recall that

$$r_t = \sigma (W_r x_t + U_r h_{t-1} + b_r) \tag{3.13}$$

then

$$\frac{\partial r_t}{\partial h_{t-1}} = diag\left(\sigma' (W_r x_t + U_r h_{t-1} + b_r)\right) U_r = diag\left(r_t \odot (1 - r_t)\right) U_r \tag{3.14}$$

since $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. Moreover, we can bound $\frac{\partial r_t}{\partial h_{t-1}}$ in the following way

$$\left\|\frac{\partial r_t}{\partial h_{t-1}}\right\|_2 \leq \delta_r \|U_r\|_2 \tag{3.15}$$

with

$$\delta_r := \max_i \{[r_t]_i (1 - [r_t]_i)\}. \tag{3.16}$$

The constant δ_r is defined to be the largest entry of the vector $r_t \odot (1 - r_t)$ and it is bounded by $\frac{1}{4}$. Similarly,

$$\frac{\partial u_t}{\partial h_{t-1}} = diag\left(\sigma' (W_u x_t + U_u h_{t-1} + b_u)\right) U_u = diag\left(u_t \odot (1 - u_t)\right) U_u, \tag{3.17}$$

and

$$\left\|\frac{\partial u_t}{\partial h_{t-1}}\right\|_2 \leq \delta_u \|U_u\|_2 \tag{3.18}$$

with

$$\delta_u := \max_i \{[u_t]_i (1 - [u_t]_i)\}. \tag{3.19}$$

The definition of the vector c_t is

$$c_t = \Phi (W_c x_t + U_c (r_t \odot h_{t-1}) + b_c) \tag{3.20}$$

and

$$\frac{\partial c_t}{\partial h_{t-1}} = diag\left(\Phi' (W_c x_t + U_c(r_t \odot h_{t-1}) + b_c)\right) U_c \left(diag(h_{t-1}) \frac{\partial r_t}{\partial h_{t-1}} + diag(r_t)\right), \tag{3.21}$$

with Φ' applied entrywise, and

$$\left\|\frac{\partial c_t}{\partial h_{t-1}}\right\|_2 \leq \max_i\{[\Phi'(W_c x_t + U_c(r_t \odot h_{t-1}) + b_c)]_i\} \cdot \left(\left\|\frac{\partial r_t}{\partial h_{t-1}}\right\|_2 \max_i\{[h_{t-1}]_i\} + \max_i\{[r_t]_i\}\right)\|U_c\|_2 \tag{3.22}$$

$$\leq \left(\delta_r \|U_r\|_2 \max_i\{[h_{t-1}]_i\} + \max_i\{[r_t]_i\}\right)\|U_c\|_2 \tag{3.23}$$

since Φ' is bounded by 1. Finally,

$$h_t = (1-u_t)\odot h_{t-1} + u_t \odot c_t \tag{3.24}$$

with

$$\frac{\partial h_t}{\partial h_{t-1}} = -diag(h_{t-1})\frac{\partial u_t}{\partial h_{t-1}} + diag(1-u_t) + diag(c_t)\frac{\partial u_t}{\partial h_{t-1}} + diag(u_t)\frac{\partial c_t}{\partial h_{t-1}} \tag{3.25}$$

and

$$\begin{aligned}\left\|\frac{\partial h_t}{\partial h_{t-1}}\right\|_2 \leq{} & \max_i\{[h_{t-1}]_i\}\left\|\frac{\partial u_t}{\partial h_{t-1}}\right\|_2 + \max_i\{(1-[u_t]_i)\} \\ & + \max_i\{[c_t]_i\}\left\|\frac{\partial u_t}{\partial h_{t-1}}\right\|_2 + \max_i\{[u_t]_i\}\left\|\frac{\partial c_t}{\partial h_{t-1}}\right\|_2.\end{aligned} \tag{3.26}$$

Furthermore, using (3.15), (3.18), and (3.23), we get

$$\left\|\frac{\partial h_t}{\partial h_{t-1}}\right\|_2 \leq \delta_u\left(\max_i\{[h_{t-1}]_i\} + \max_i\{[c_t]_i\}\right)\|U_u\|_2 + \max_i\{(1-[u_t]_i)\}$$

$$+ \max_i\{[u_t]_i\}\left(\delta_r\|U_r\|_2 \max_i\{[h_{t-1}]_i\} + \max_i\{[r_t]_i\}\right)\|U_c\|_2 \tag{3.27}$$

$$=: \alpha + \beta\|U_c\|_2 \tag{3.28}$$

where

$$\alpha = \delta_u\left(\max_i\{[h_{t-1}]_i\} + \max_i\{[c_t]_i\}\right)\|U_u\|_2 + \max_i\{(1-[u_t]_i)\} \tag{3.29}$$

and

$$\beta = \max_i\{[u_t]_i\}\left(\delta_r\|U_r\|_2 \max_i\{[h_{t-1}]_i\} + \max_i\{[r_t]_i\}\right). \tag{3.30}$$

□

The following corollary provides some simple upper bounds for α, β obtained in the above theorem.

Corollary 1. *For the hyperbolic tangent activation function in (3.1) (i.e. $\Phi = \mathtt{tanh}$), we have $\delta_u, \delta_r \leq \frac{1}{4}$, $[h_t]_i \leq 1$ for any i and t as well as*

$$\alpha \leq \frac{1}{2}\|U_u\|_2 + 1 \quad \textit{and} \quad \beta \leq \frac{1}{4}\|U_r\|_2 + 1. \tag{3.31}$$

Proof.

The function $\sigma'(x) = \sigma(x)\,(1-\sigma(x))$ is bounded above by $\frac{1}{4}$, thus both $\delta_u, \delta_r \leq \frac{1}{4}$.

Now, to show that $[h_t]_i \leq 1$ for any i and t, we first need to note that h_0 is initialized to zero (i.e. $[h_0]_i = 0$ for all i) and $0 \leq [u_t]_i, [c_t]_i \leq 1$ for all i and t from the definition of GRU cell in (3.1). Then for any fixed i

$$\begin{aligned} [h_0]_i &= 0 && (3.32)\\ [h_1]_i &= [1-u_1]_i \cdot [h_0]_i + [u_1]_i \cdot [c_1]_i && (3.33)\\ &= [u_1]_i \cdot [c_1]_i \leq 1 && (3.34) \end{aligned}$$

Furthermore, if we assume that $[h_\tau]_i \leq 1$ for some $\tau \geq 1$, then

$$\begin{aligned} [h_{\tau+1}]_i &= [1-u_{\tau+1}]_i \cdot [h_\tau]_i + [u_{\tau+1}]_i \cdot [c_{\tau+1}]_i && (3.35)\\ &\leq [1-u_{\tau+1}]_i \cdot 1 + [u_{\tau+1}]_i \cdot 1 && (3.36)\\ &= [1-u_{\tau+1}]_i + [u_{\tau+1}]_i = 1. && (3.37) \end{aligned}$$

Thus, by induction, we can conclude that $[h_t]_i \leq 1$ for any i and t. Finally, using these obtained bounds, we can bound constants α and β as follows

$$\begin{aligned} \alpha &= \delta_u \left(\max_i \{[h_{t-1}]_i\} + \max_i \{[c_t]_i\} \right) \|U_u\|_2 + \max_i \{(1-[u_t]_i)\} && (3.38)\\ &\leq \frac{1}{4} \cdot (1+1) \cdot \|U_u\|_2 + 1 && (3.39)\\ &= \frac{1}{2} \|U_u\|_2 + 1 && (3.40) \end{aligned}$$

and

$$\begin{aligned} \beta &= \max_i \{[u_t]_i\} \left(\delta_r \|U_r\|_2 \max_i \{[h_{t-1}]_i\} + \max_i \{[r_t]_i\} \right) && (3.41)\\ &\leq 1 \cdot \left(\frac{1}{4} \cdot \|U_r\|_2 \cdot 1 + 1 \right) && (3.42)\\ &= \frac{1}{4} \|U_r\|_2 + 1. && (3.43) \end{aligned}$$

Note, that all of these bounds are independent of i and t.

□

These bounds may be pessimistic because the gate elements may be expected to be close to 0 or 1. Consequently, the following corollary presents the relationship between the constants α and β presented in Theorem 5 when GRU's gates are nearly closed or opened. Below we use a notation $x \lesssim y$ to denote that x is bounded by a quantity approximately equal to y.

Corollary 2. *When elements of GRU gates u_t and r_t are nearly either 0 or 1, then constants α and β from Theorem 5 satisfy the following inequality:*

$$\alpha + \beta \lesssim 2. \tag{3.44}$$

Moreover, if u_t and r_t are nearly either the zero vector or the vector of all ones, then

$$\alpha + \beta \lesssim 1. \tag{3.45}$$

Proof.

Recall the definition of α and β

$$\alpha = \delta_u \left(\max_i \{[h_{t-1}]_i\} + \max_i \{[c_t]_i\}\right) \|U_u\|_2 + \max_i \{(1 - [u_t]_i)\} \tag{3.46}$$

and

$$\beta = \max_i \{[u_t]_i\} \left(\delta_r \|U_r\|_2 \max_i \{[h_{t-1}]_i\} + \max_i \{[r_t]_i\}\right). \tag{3.47}$$

If we assume that $[u_t]_i \approx 0$ and $[u_t]_j \approx 1$ for some $i \neq j$, then $\delta_u \approx 0$ and $\alpha \lesssim 1$. Additionally, if we assume that the elements of r_t are nearly either 0 or 1, then $\delta_r \approx 0$ and $\beta \lesssim 1$. Putting these two inequalities together yields

$$\alpha + \beta \lesssim 2. \tag{3.48}$$

Now, if we assume that u_t is nearly a zero vector (i.e. $\max_i \{[u_t]_i\} \approx 0$), then constant $\delta_u \approx 0$, $\alpha \approx 1$, and $\beta \approx 0$. On the other hand, if we assume that u_t is nearly the vector of all ones (i.e. $\min_i \{[u_t]_i\} \approx 1$) then $\delta_u \approx 0$ and $\alpha \approx 0$. Moreover, if we also assume that r_t is nearly a zero vector (i.e. $\max_i \{[r_t]_i\} \approx 0$) then $\delta_r \approx 0$ and $\beta \approx 0$. However, if we assume that r_t approaches a vector of all ones (i.e. $\min_i \{[r_t]_i\} \approx 1$), then $\delta_r \approx 0$ but $\beta \approx 1$ for this case. Putting all of these cases together, we obtain

$$\alpha + \beta \lesssim 1. \tag{3.49}$$

□

3.1.3.2 Neumann-Cayley Orthogonal Transformation

Based on Theorem 5 and Corollary 2, we propose the usage of orthogonal weights in the hidden parameters of GRU to obtain better-conditioned gradients. As discussed in section 3.1.2, there have been different techniques proposed and used to initialize and preserve orthogonal weights while training, for example, Givens rotations [100], Householder reflections [141] etc. In this work, we implement a version of the Scaled-Cayley transformation method discussed in 3.1.2.2 with one key difference. The Scaled Cayley transform method requires a calculation of the inverse of $I + A^{(k)}$ to update the orthogonal matrix $U^{(k)}$ in (3.4). When it comes to the computation of this inverse, classical numerical methods such as using LU-decomposition or solving the Least Squares problem can be implemented. These methods work well when the dimension of the matrix is small. However, if the matrix's dimension is large, these methods are very expensive from both memory and computational time perspectives. Moreover, classical methods might overflow and not converge at all. We propose solving this possible complication using the Neumann Series method to approximate the inverse of $I + A^{(k)}$.

To derive the Neumann series approximation for the inverse of $I + A^{(k)}$ in (3.4), we consider the following:

$$\left(I + A^{(k)}\right)^{-1} = \left(I + A^{(k-1)} - \delta A^{(k)}\right)^{-1} \tag{3.50}$$

$$= \left(I - \left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)}\right)^{-1} \left(I + A^{(k-1)}\right)^{-1} \tag{3.51}$$

$$= \left(\sum_{i=0}^{\infty} \left(\left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)}\right)^{i}\right) \left(I + A^{(k-1)}\right)^{-1} \tag{3.52}$$

where $\delta A^{(k)} := opt_A \left(\nabla_A \mathcal{L} = {V^{(k)}}^T - V^{(k)}\right)$, here opt_A includes a learning rate λ inside of it. Note that the equality in Equations (3.51) and (3.52) relies on the assumption that $\left\| \left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)} \right\| < 1$ for some operator norm $\|\cdot\|$; see [45] for more details. We have conducted an ablation study that shows empirical evidence that this condition is indeed satisfied; see [145] for more details.

In our experiments, we have considered the first and the second-order Neumann series approximations, estimating the series in (3.52) with two ($i = 0, 1$) and three ($i = 0, 1, 2$) terms respectively. As expected, the model performs slightly better when using second-order approximation. However, it comes with a marginal increase in computational time; see 3.1.5.1 for the ablation study regarding the accuracy and computational time of such approximations. Mathematically speaking, if we are using the second-order approximation, the error is of order $\mathcal{O}\left(\left(\left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)}\right)^{3}\right)$. Even though this error is quite small, there is a chance that the errors from this approximation can accumulate and cause a loss of orthogonality. To avoid this issue, we recommend resetting orthogonality by computing the matrix inverse explicitly using a factorization method at the beginning of each epoch. However, it might be necessary to do it more often, particularly in the earlier training (e.g., every 100 iterations), due to more fluctuations in the gradients.

Algorithm 4: Update Rule for Orthogonal Weight U

```
1: Given: $D$, $A^{(0)}$, $U^{(0)}$, $\nabla_U \mathcal{L}\left(U^{(0)}\left(A^{(0)}\right)\right)$, $opt_A$
2: Define: $\tilde{A}^{(0)} := \left(I + A^{(0)}\right)^{-1}$
3: for $k = 1, 2, \ldots$ do
4:     $V^{(k)} := {\tilde{A}^{(k-1)}}^T \nabla_U \mathcal{L}\left(U^{(k-1)}\left(A^{(k-1)}\right)\right)\left(D + {U^{(k-1)}}^T\right)$
5:     $\delta A^{(k)} := opt_A\left(\nabla_A \mathcal{L} = {V^{(k)}}^T - V^{(k)}\right)$
6:     $A^{(k)} := A^{(k-1)} - \delta A^{(k)}$
7:     $\tilde{A}^{(k)} := \left(I + \tilde{A}^{(k-1)}\delta A^{(k)} + \left(\tilde{A}^{(k-1)}\delta A^{(k)}\right)^2\right)\tilde{A}^{(k-1)}$
8:     $U^{(k)} := \tilde{A}^{(k)}\left(I - A^{(k)}\right)D$
9: end
```

We present Algorithm 4 that outlines the Neumann-Cayley Orthogonal Transformation method for training weight A and updates the corresponding orthogonal weight U. It is important to note that during the initialization step, the weight $A^{(0)}$ is defined to be skew-symmetric using the same initialization technique as in [81], which is based on the idea from [84]. Then, we apply the Cayley transform in $A^{(0)}$ to obtain

$U^{(0)}$. Another peculiar detail that we want to point out is that Algorithm 4 includes opt_A, which is the standard optimizer such as SGD, RMSProp [181], or Adam [110], etc., that takes $\nabla_A \mathcal{L} = V^{(k)^T} - V^{(k)}$ as an input. Moreover, the skew-symmetric property of the weight A and its gradient are preserved under such an optimizer.

3.1.3.3 Neumann-Cayley Orthogonal GRU (NC-GRU)

Finally, we introduce a Neumann-Cayley Orthogonal GRU (NC-GRU) model that utilizes the proposed Neumann-Cayley Orthogonal Transform.

The structure of the NC-GRU cell is shown below:

$$\begin{aligned} r_t &= \sigma\left(W_r x_t + U_r(A_r) h_{t-1} + b_r\right) \\ u_t &= \sigma\left(W_u x_t + U_u h_{t-1} + b_u\right) \\ c_t &= \Phi\left(W_c x_t + U_c(A_c)\left(r_t \odot h_{t-1}\right)\right) \\ h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t \end{aligned} \tag{3.53}$$

here σ - sigmoid function, $\odot$ - Hadamard product, and Φ - modReLU function defined in [7] as

$$\Phi(x) := \text{modReLU}(x) := \text{sgn}(x) \cdot \text{ReLU}\left(|x| + b\right) \tag{3.54}$$

with b as a trainable bias.

Most experiments show that the best performance is achieved using orthogonality in U_c and U_r hidden weights. For an additional ablation study about the usage and performance of orthogonal weights throughout the GRU model, see [145]. Similar to the GRU Cell, W_r, W_u, W_c, U_u, b_r, b_u, b, are trainable parameters as well as U_r and U_c together with their associated weights A_r and A_c respectively. Moreover, all of them except U_r with A_r and U_c with A_c are trained using standard backpropagation algorithms such as Stochastic Gradient Descent (SGD), RMSProp [181], or Adam [110] similarly as in GRU [37], but U_r, A_r, U_c, and A_c are trained using Algorithm 4. Figures 3.1a and 3.1b depict the NC-GRU cell 3.53 forward pass and backward propagation of orthogonal weight $U_c(A_c)$, respectively.

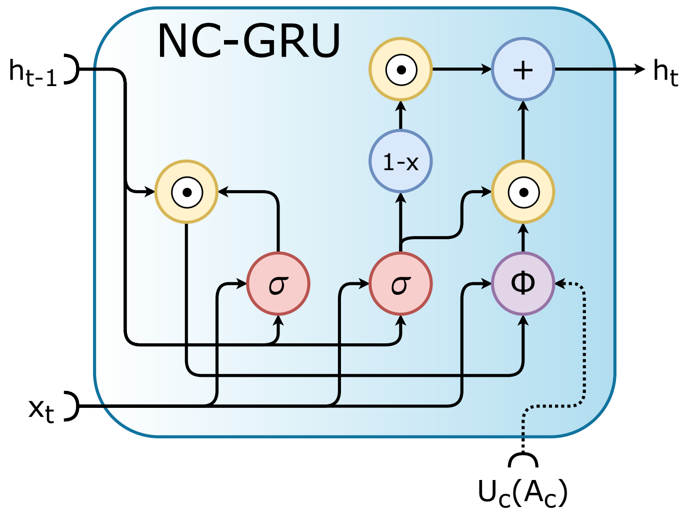
As we mentioned, orthogonal weights lead to a better-conditioned gradient, and the following Corollary summarizes this result.

Corollary 3. *Let h_{t-1} and h_t be two consecutive hidden states from the NC-GRU model defined in (3.53). Then $\|U_r\|_2 = \|U_c\|_2 = 1$ and if elements of the gates u_t and r_t are nearly 0 or 1, then the following inequality is satisfied:*

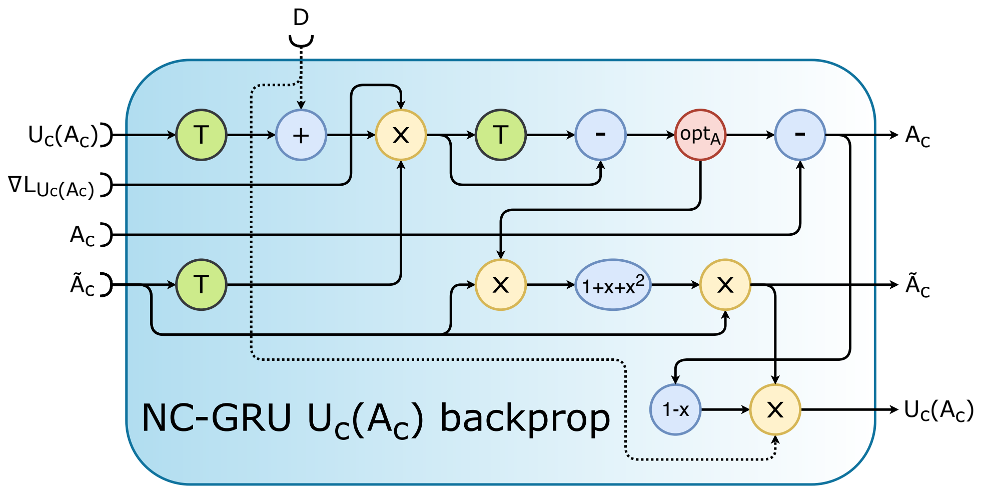
$$\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|_2 \lesssim 2. \tag{3.55}$$

Furthermore, if u_t and r_t are nearly either zero vector or vector of all ones,

$$\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|_2 \lesssim 1. \tag{3.56}$$



(a) NC-GRU Cell



(b) Backpropagation of $U_c(A_c)$

Figure 3.1: NC-GRU Cell and diagram for updating the orthogonal weight $U_c(A_c)$ Notation: σ - sigmoid function, Φ - modReLU [7], $\odot$ - Hadamard product (entrywise multiplication), T - transpose, $\times$ - matrix multiplication, opt_A - weight A optimizer, any algebraic expression (e.g., $1-x$) is evaluated with previous step output as input (1 represents an identity matrix); refer to Algorithm 4 for the order of non-commutative operations

Proof.

By the definition of NC-GRU model, weights U_r and U_c are orthogonal which implies $\|U_r\|_2 = \|U_c\|_2 = 1$, and by Theorem 5 and Corollary 2, we conclude

$$\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|_2 \lesssim 2 \tag{3.57}$$

when element of the gates u_t and r_t approach either 0 or 1; and

$$\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|_2 \lesssim 1 \tag{3.58}$$

if u_t and r_t approach either zero vector or vector of all ones.

□

3.1.4 Experiments and Results

We have performed various experiments to demonstrate the robustness and efficiency of our NC-GRU method. To this end, we apply NC-GRU to commonly used synthetic tasks: Parenthesis [57, 99], Denoise, Adding, and Copying Tasks, see [145] for the last two tasks. In addition, we have considered non-synthetic experiments, Language Modeling with the character and word level tasks for the Penn TreeBank (PTB) [133] dataset as well as WikiText-2 [137], see [145] for more information and results for these tasks.

All the experiments were trained with equal numbers of trainable parameters, also known as parameter-matching architecture. Moreover, all models were trained using a single NVIDIA® Tesla® V100 GPU with TensorFlow 1.13.2 (Parenthesis, Denoise, Adding, and Copying Tasks), PyTorch 1.1.0 (PTB and WikiText-2), and Python 3.6.9.

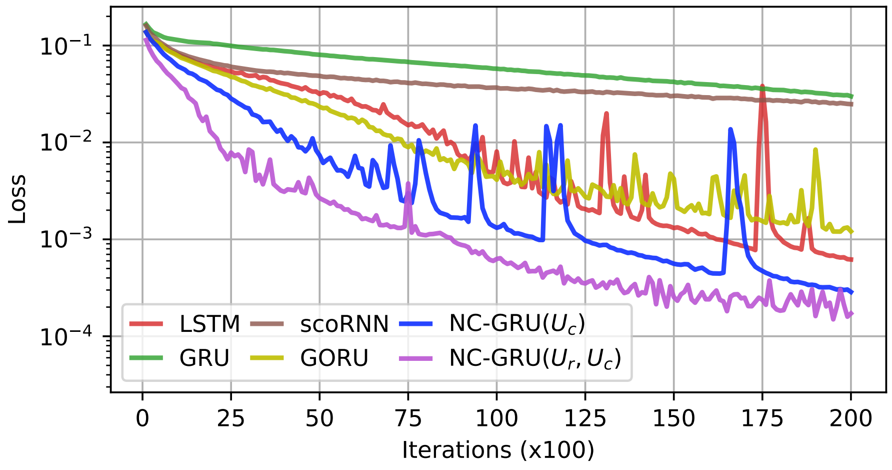
3.1.4.1 Parenthesis Task

This experiment derives from the descriptions in [57] and [99]. This task tests the ability of the network to remember the number of unmatched parentheses contained in our input data. The input data consists of 10 pairs of different types of parentheses combined with some noise data in between, and it is given as a one-hot encoding vector of length T. As stated in [99], there are not more than 10 types of parentheses. The output data is given as one hot-encoding vector, counting the number of unpaired parentheses in the corresponding input data. The goal of our model is to forget the noise data and absorb information from the long-term dependencies related to the parentheses. This synthetic data requires the model to develop a memory and to be able to select the most relevant information.

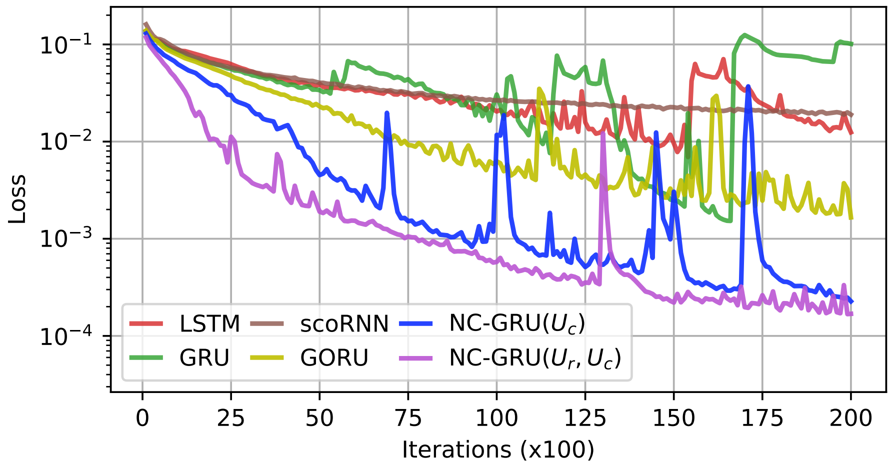
We present two versions of the NC-GRU model; the first one only has orthogonality in U_c weight (NC-GRU(U_c)); however, the second model utilizes orthogonality in both weights U_r and U_c (NC-GRU(U_r, U_c)). Both models were trained using the Neumann series method with a reset every 50 iterations.

Implementation Details: All of the models consisted of a single layer net with the following hidden dimensions for each model: LSTM [88] - 42, GRU [37] - 50, scoRNN [81] - 110, GORU [99] - 64, and NC-GRU - 56. Furthermore, we trained all models for 200 epochs with a batch size of 16, and the number of negative ones for the D matrix in scoRNN [81] and NC-GRU models was 20 and 40, respectively. All models were trained using Adam optimizer [110] with a learning rate of 10^{-3} including A associated weights in scoRNN [81] and NC-GRU models. We conducted experiments using input length of $T = 100$, see Figure 3.2a, and $T = 200$, see Figure 3.2b.

Results: We observed and recorded the behavior of the five models on the Parenthesis task when the input length is set to 100 and 200. Our results showed that both versions of NC-GRU models outperform GRU [37], LSTM [88], scoRNN [81], and GORU [99] models with a significant gap; see Figure 3.2. On this task, NC-GRU(U_r, U_c) model shows a better performance than NC-GRU(U_c); however, both of them perform better than the rest of the compared models. The minimum value of the loss attained during training is presented in Table 3.1.



(a) $T = 100$



(b) $T = 200$

Figure 3.2: **Parenthesis Task Results:** NC-GRU(U_c) denotes the NC-GRU model (3.53) where the Neumann-Cayley method was only applied to the weight U_c, similarly NC-GRU(U_r, U_c) represents the NC-GRU model (3.53) where both U_r and U_c weights were updated using the Neumann-Cayley method.

Table 3.1: **Parenthesis Task Results:** Minimum attained loss values ($\downarrow$ - the smaller, the better). All results are based on our tests.

	Loss $\times 10^{-3} \downarrow$	
Sequence Length	$T = 100$	$T = 200$
LSTM	0.62	6.93
GRU	29.94	1.526
scoRNN	24.85	18.957
GORU	1.199	1.66
NC-GRU(U_c) (ours)	0.286	0.227
NC-GRU(U_r, U_c) (ours)	0.149	0.167

3.1.4.2 Denoise Task

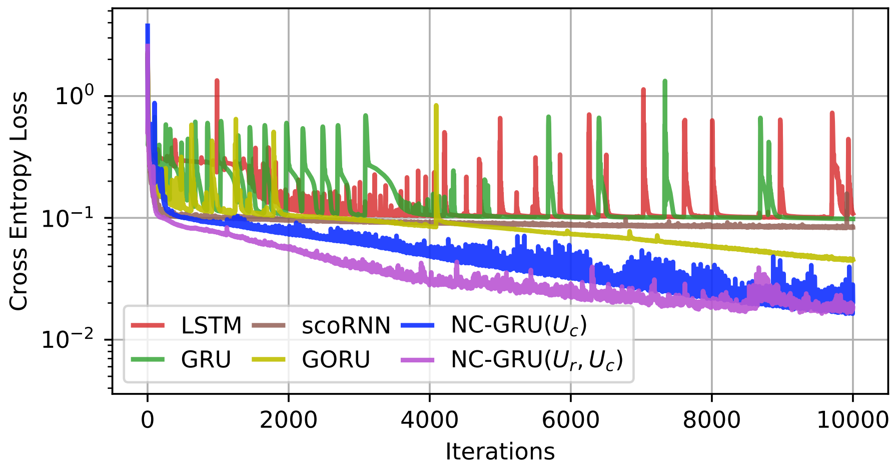
The Denoise Task [99] is another synthetic problem requiring filtering out the noise from a noisy sequence. This problem requires the forgetting ability of the network as well as learning long-term dependencies coming from the data [99]. The input sequence of length T contains 10 randomly located data points and the other $T - 10$ points are considered noise data. These 10 points are selected from a dictionary $\{a_i\}_{i=0}^{n+1}$, where the first n elements are data points, and the other two are the "*noise*" and the "*marker*" respectively. The output data consists of the list of the data points from the input, and it should be outputted as soon as it receives the "*marker*". The model task is to filter out the noise part and output the random 10 data points chosen from the input.

Implementation Details: We implemented one NC-GRU cell with a hidden size of 118 and the number of negative ones in the D matrix to 50. The hidden size for the LSTM [88] was 90, GRU [37] – 100, scoRNN [81] – 200, and GORU [99] – 128. We implemented Adam optimizer [110] with a learning rate of 10^{-3} to train all the aforementioned models, including scoRNN [81] and NC-GRU A weights. We trained all the models for 10,000 iterations with a batch size of 128. Similar to the Parenthesis Task 3.1.4.1, we implemented the Neumann series method of approximation the $\left(I + A^{(k)}\right)^{-1}$ when training the NC-GRU model with the reset option to be applied every 50 iterations.

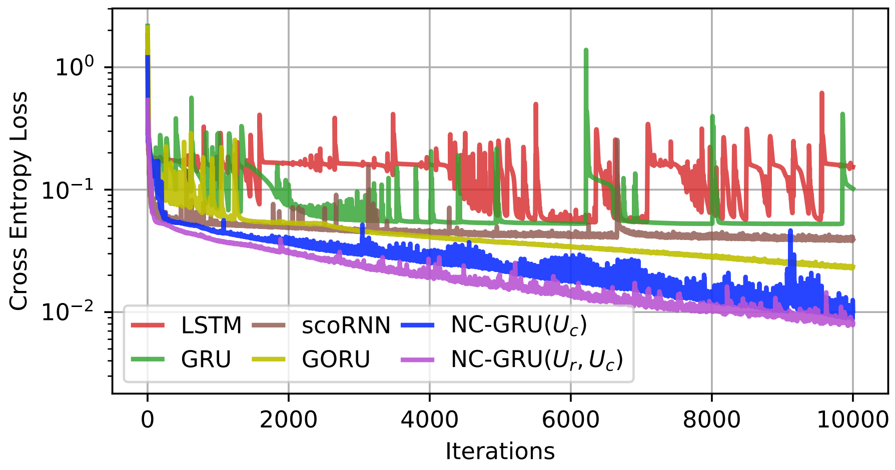
Results: Based on our experiments, NC-GRU(U_c) and NC-GRU(U_r, U_c) models significantly outperformed LSTM [88], GRU [37], scoRNN [81], and GORU [99] models on the denoise data; see Figure 3.3. Similar to what we observed from the parenthesis task, using two orthogonal weights leads to better results. In addition, Table 3.2 compares attained minimum loss for each model.

3.1.5 Exploratory & Ablation Studies

This section considers several ablation studies that help us justify using the Neumann series, orthogonal matrices, and Scaled-Cayley transforms.



(a) $T = 200$



(b) $T = 400$

Figure 3.3: **Denoise Task Results**

3.1.5.1 Neumann Series Method vs Inverse

In this experiment, we study the sharpness of approximating $\left(I + A^{(k-1)}\right)^{-1}$ with the Neumann series in the NC-GRU(U_c) model on the Parenthesis task; see 3.1.4.1 for implementation details and description of NC-GRU(U_c) model. We consider using the Neumann series approximation of orders 1, 2, and 3 and compare them to the Least Squares method for taking a matrix inverse, one of the widely used methods from Deep Learning libraries such as TensorFlow, PyTorch, and NumPy.

Our experiments showed that the Neumann series approximation method achieves better results than the classical Least Squares method for finding the matrix inverse. Figure 3.4 shows that the Neumann series method of order 2 performs marginally better than order 1 and order 3 Neumann series methods and significantly better than the Least Squares method.

We have also compared the time it takes to train our model using the methods

Table 3.2: **Denoise Task Results:** Minimum attained loss values. All results are based on our tests.

	Loss $\times 10^{-2} \downarrow$	
Sequence Length	$T = 200$	$T = 400$
LSTM	10.08	5.427
GRU	9.799	5.2338
scoRNN	8.258	3.812
GORU	4.453	2.29
NC-GRU(U_c) (ours)	1.639	0.878
NC-GRU(U_c, U_r) (ours)	1.633	0.774

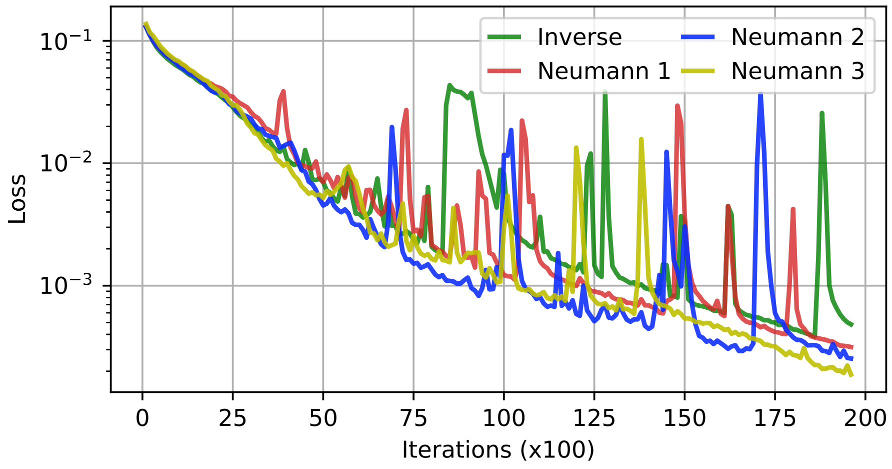


Figure 3.4: **Inverse vs Neumann series:** In this graphs, `Inverse` represents NC-GRU(U_c) model with $\left(I + A^{(k-1)}\right)^{-1}$ computed using the Least Squares method and `Neumann i` represents NC-GRU(U_c) model with $\left(I + A^{(k-1)}\right)^{-1}$ computed using the Neumann series method of order i.

mentioned. Figure 3.5 shows the time it takes to train one epoch of the NC-GRU(U_c) model on a Character Level PTB dataset on a single NVIDIA® Tesla® V100 GPU using Inverse (Least Square), Neumann 1, Neumann 2, and Neumann 3 methods.

The observed behavior appears to be quite general, and we have conducted all of the experiments in section 3.1.4 using the second-order Neumann series method.

Several other Ablation Studies regarding the Orthogonality in the hidden weights of GRU and necessary conditions for the Neumann series method can be found in [145].

3.1.6 Conclusion

We have presented a thorough analysis of the Gated Recurrent Unit (GRU) model's gradients. Based on this analysis, we introduced the Neumann-Cayley Gated Re-

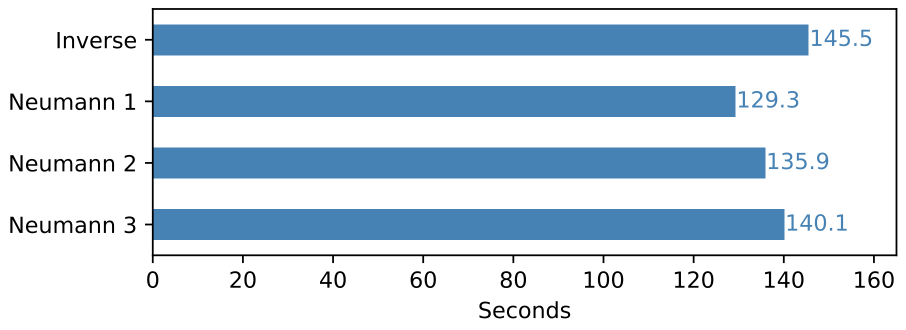
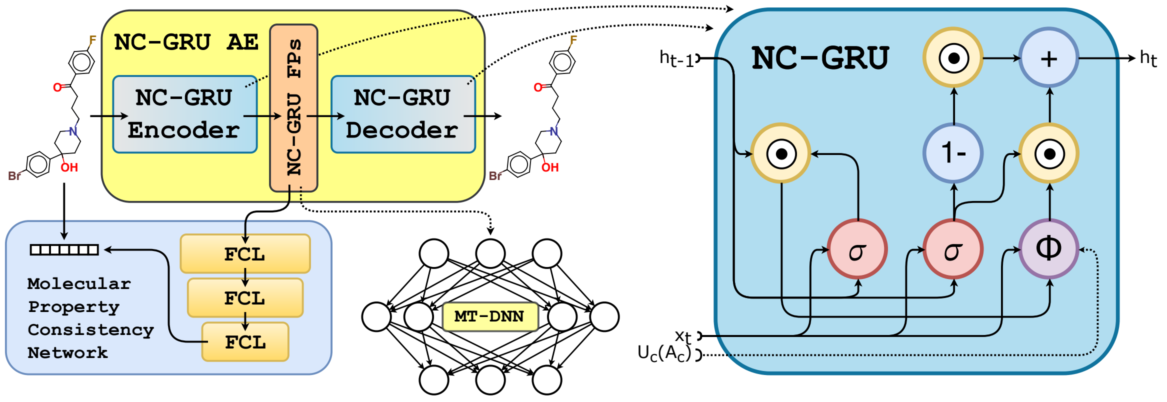


Figure 3.5: **Computational time comparisons:** The amount of time (in seconds) it takes to train one epoch of NC-GRU(U_c) model on Character Level PTB dataset using a single NVIDIA® Tesla® V100 GPU.

current Unit model, NC-GRU. Our model incorporates orthogonal weights in the hidden states of the GRU model, which are trained using a newly proposed method of Neumann-Cayley transformation for maintaining the desired orthogonality in those weights. We have conducted experiments demonstrating the superiority of our proposed method outperforming GRU, LSTM, scoRNN, and GORU models on different tasks. Moreover, we conducted several ablation studies that empirically confirmed our theoretical results.

3.2 Novel Molecular Representations using Neumann-Cayley Orthogonal Gated Recurrent Unit



Researchers in various fields of study, including the bio-medical and cheminformatics communities, now have access to a potent machine learning method thanks to advancements in Deep Neural Networks and Deep Learning. They aid in tasks like protein performance, molecular design, drug discovery, etc. Many of those activities use molecular descriptors to express molecular properties in cheminformatics. Unfortunately, quantitative prediction of molecular attributes is still tricky despite major efforts and the advent of several approaches that produce molecular descriptors. Molecular fingerprints are a popular technique for encoding molecule characteristics into bit strings. This section suggests creating neural molecular fingerprints called NC-GRU fingerprints utilizing novel Neumann-Cayley Gated Recurrent Units (NC-GRU) inside the Neural Nets encoder (AutoEncoder). Orthogonal weights are added by NC-GRU AutoEncoder to the famous GRU architecture, producing quicker, more stable training and more trustworthy molecular fingerprints. The performance of different molecular-related tasks, including toxicity, partition coefficient, lipophilicity, and solvation-free energy, is improved by integrating unique NC-GRU fingerprints with Multi-Task Deep Neural Network schematics, leading to state-of-the-art results on multiple benchmarks. The above Figure summaries the methods used and developed in this section.

3.2.1 Introduction

Drug development has made significant strides in recent years, yet it is still difficult to create affordable, effective molecules with appropriate pharmacological and biochemical qualities [63]. A drug candidate's binding affinity, toxicity, and octanol-water partition coefficient (logP) are all essential factors to consider [62]. Before coming to the market, a drug goes through several stages, including target identification, lead optimization, preclinical development, and three phases of clinical testing [63]. According to [191], over half of the medication candidates fail to make it to market because of unfavorable results on the toxicity and pharmacokinetic features. To test the qualities of drugs, experiments are conducted *in vivo* or *in vitro*. These methods are time- and money-consuming, as well as animal research raises important moral questions and concerns.

Recently, machine and deep learning algorithms have been successfully used in drug development. A significant amount of work has been devoted to deriving molecular descriptors from the representation of a molecule [32, 63, 202], particularly molecular fingerprints that profile a molecule, usually in the form of a bit string or a vector, with each vector element indicating the existence, the degree, or the frequency of one particular structure feature [27, 126, 162]. The majority of molecular fingerprints are created using either two-dimensional [54, 76, 153, 162, 174, 183] or three-dimensional [147, 194] molecular structural formulas, where two-dimensional structure can be viewed as if molecules were flat. The most well-known fingerprints include Molecular Access System (MACCS) [54], Extended Reduced Graph (ERG) [174], Electro Topological State (Estate) [76], Extended-Connectivity Fingerprint(ECFP) [162], FP2 [153], Daylight [183], etc. In machine and deep learning, methods such as [32, 55, 70, 202, 209, 212] have been very successful in getting helpful and useful information on molecular fingerprints.

For instance, two-dimensional deep learning algorithms try to learn an appropriate data representation from a straightforward embedding layer, where the input is a one-hot vector of each atom in a molecule [78]. Such embedding is often a component of the encoding mechanism, and one of the popular deep learning methods is AutoEncoder [163], which learns the descriptors in an unsupervised and data-driven manner [32, 70, 202, 209]. In theory, AutoEncoder can accept any molecular representation or nomenclature as input; however, in practice, researchers tend to concentrate on sequence-based molecular representations like Simplified Molecular-Input Line-Entry System (SMILES) [199], International Chemical Identifier (InChI) [83], International Union of Pure and Applied Chemistry (IUPAC) [56], etc. A typical AutoEncoder consists of Encoder and Decoder networks, where embedded input passes through the Encoder and outputs a latent representation which will be fed into the Decoder network. Then, the Decoder network takes that latent vector and attempts to turn it back into the input sequence of a different or identical nomenclature. The latent representation vector is connected to an information bottleneck between the Encoder and Decoder networks. Since the information is compressed, the latent representation vector learns more general molecular information [209]. Although the structure of the AutoEncoder can vary, the Gated Recurrent Unit (GRU) is one of the most optimized architectures to implement as the AutoEncoder cells [202] in comparison to Long-Short Term Memory (LSTM) [89] or Convolutional Neural Network (CNN) [115].

Finally, as deep learning approaches benefit from a high number of training samples, they may generate better and more effective molecular descriptors from big training datasets like ChEMBL [64], ZINC15 [96], PubChem [108], etc. The resulting fingerprints can then be applied to various prediction tasks, including toxicity prognosis, partition coefficient analysis, solubility predictions, etc., where the latent representation vector is used as the input for a prediction model [63, 167, 202].

This work proposes a novel AutoEncoder equipped with Neumann-Cayley Orthogonal Gated Recurrent Units (NC-GRU) [145] to generate high-quality fingerprints for complex and diverse molecules. To do this, we used the ChEMBL 28 [64] dataset to train NC-GRU AutoEncoder with the Canonical SMILES representation of a molecule inputted into and outputted from the NC-GRU Autoencoder. The ability of NC-GRU to capture long-term relationships using orthogonal matrices and the capacity of GRU gates to forget unneeded information make NC-GRU architectures preferable to ordinary GRU cells. The combination of training AutoEncoder on the ChEMBL 28 dataset with NC-GRU cells results in NC-GRU FingerPrints (FPs), improving many benchmarks during the inference phase. Using NC-GRU FPs produced state-of-the-art results on several prediction tasks, including toxicity, partition coefficient, solubility, and solvation-free energy.

3.2.2 Neumann-Cayley Orthogonal Gated Recurrent Unit based AutoEncoder

3.2.2.1 Architecture

In this study, our primary goal is to apply NC-GRU [145] to the AutoEncoder's hidden layers [163]. Refer to Figures 3.1a and 3.1b for the NC-GRU cell architecture and the orthogonal weight $U_c(A_c)$ update process diagrams, respectively. Before feeding it to the AutoEncoder, the input sequence-based molecular representation, provided as canonical SMILES, is tokenized and encoded in a one-hot vector representation. Our NC-GRU AutoEncoder has 2 or 3 stacked NC-GRU layer cells, depending on the effectiveness of the fine-tuning procedure. The two-layer NC-GRU model has hidden layer dimensions of 160 and 320. An additional NC-GRU layer cell of dimension 640 is used for the three-layer version. Afterward, the state of each cell from the Encoder is concatenated and used as an input vector in a Fully-Connected Layer of dimensions 512. The Fully-Connected Layer is activated using a hyperbolic tangent (`tanh`) function. The 512-unit extracted features vector is used as an input vector in another Fully-Connected Layer. The output of this Fully-Connected Layer is divided into three parts, corresponding to each dimension from the Encoder, and used as the initialization in every Decoder cell. Simultaneously, the Molecular Properties Consistency Network (MPCN) uses the output of the Fully Connected mentioned above Layer. In its architecture, the Molecular Properties Consistency Network is a regression network with three Fully-Connected Layers of dimensions 512, 128, and 7 (as an output dimension) and ReLU activation function after the first and second layers and no activation after the output layer, respectively. The Molecular Properties Consistency Network predicts the following properties logP, the Molar refractivity, Balaban's J-value, the number of acceptors, the number of hydrogen bond donors, the number of valence electrons, and the Topological polar surface area. These properties are extracted from the encoded input sequence's molecular structure using the Python RDKit library. The network functions as a regularizer for the AutoEncoder and helps AutoEncoder to produce better molecular descriptors while maintaining the RDKit attributes. The AutoEncoder with Molecular Properties Consistency Network is trained to reduce the Mean Squared Error between RDKit features and the output of the Molecular Properties Consistency Network, $\mathcal{L}_{MPCN}$, while simultaneously it is trained to minimize the softmax cross-entropy between each input sequence and the Decoder output, $\mathcal{L}_{AutoEncoder}$, the following Equation provides the total loss function.

$$\mathcal{L}_{total} = \mathcal{L}_{AutoEncoder} + \mathcal{L}_{MPCN} \tag{3.59}$$

In addition to our suggested NC-GRU AutoEncoder, we experiment with traditional GRU [37] in AutoEncoder for a fair comparison. Figure 3.6 provides a visual aid to help comprehend the NC-GRU AutoEncoder design discussed above.

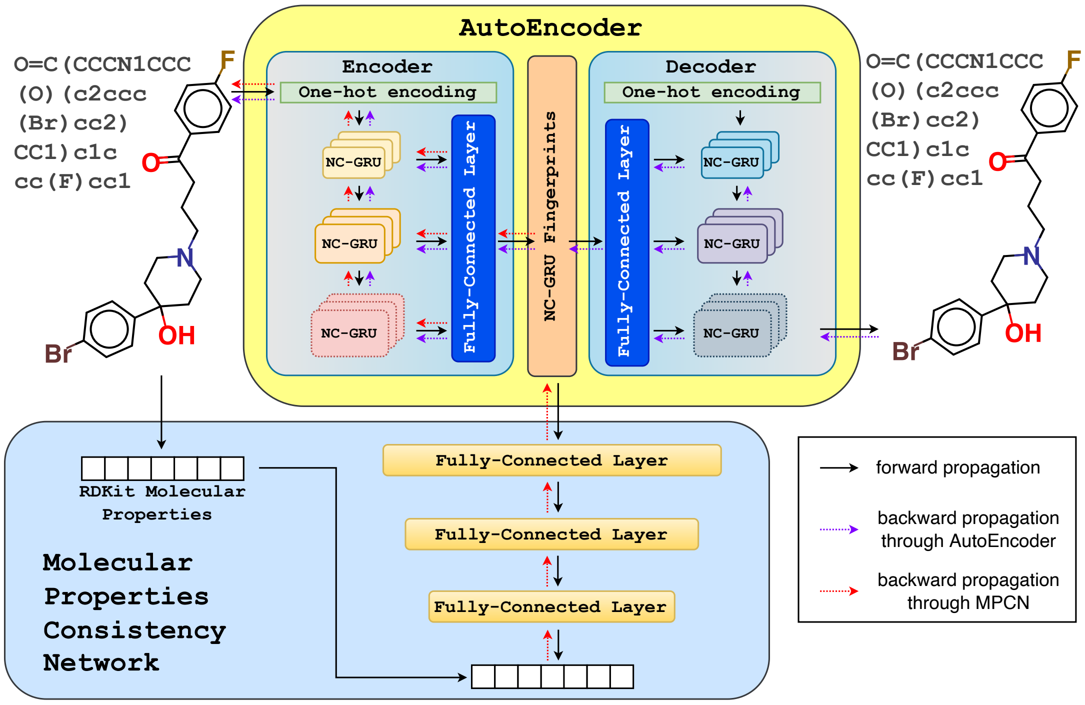


Figure 3.6: **NC-GRU AutoEncoder w/ MPCN:** Architecture for training NC-GRU fingerprints using NC-GRU AutoEncoder together with Molecular Properties Consistency Network (MPCN)

3.2.2.2 Data Processing

We have trained AutoEncoder architecture with Molecular Properties Consistency Network on the ChEMBL 28 dataset [64]. The ChEMBL 28 dataset was processed with the help of the RDKit Python module. First, we removed all the duplicates. Then, we filtered the remaining molecules with the following criteria: only organic molecules, molecules with molecular weight between 12 and 600, molecules with at least three heavy atoms, molecules with a partition coefficient $\log P$ between -7 and 5, only non-stereochemistry molecules, no salts, and molecules that RDKit could not process were removed. The remaining post-filtered dataset has 1,852,637 chemical compounds, which we randomly split 90/10 into training and testing sets of sizes 1,667,373 and 185,264, respectively. After the processing, we extracted seven RDKit molecular properties for each molecule that are used in the Molecular

Properties Consistency Network: $\log P$ (`MolLogP` in the RDKit), number of valence electrons (`NumValenceElectrons` in the RDKit), number of hydrogen bond donors (`NumHDonors` in the RDKit), number of acceptors (`NumHAcceptors` in the RDKit), Balaban's J-value (`BalabanJ` in the RDKit), molar refractivity (`MolMR` in the RDKit), and topological polar surface area (`TPSA` in the RDKit). Further, the above properties are normalized using the following equation:

$$\hat{x} = \frac{x - \mu}{\sigma}, \tag{3.60}$$

where x, $\hat{x}$, μ, and σ represent the element of the dataset under the property, its normalized version, and the mean and standard deviation of the property for the whole dataset, respectively. The above molecular properties were chosen to follow setups from [202] and [70]. These published works have testified that the constraints of RDKit features on the latent space are necessary to avoid dead areas in molecular representation learning; otherwise, it could cause the decoder network to produce invalid SMILES strings.

Table 3.3 provides data processing criteria and selected statistics about normalized RDKit molecular properties.

Table 3.3: **ChEMBL 28 Dataset:** ChEMBL 28 processing criteria and statistics of RDKit processed and normalized molecular properties

Processing Criteria	**Normalized Molecular Properties**		
Removal of duplicates	Property	min	max
Only organic molecules	$\log P$	-5.08	2.26
Molecular weight between 12 and 600	# of valence electrons	-3.46	3.16
More than three heavy atoms	# of hydrogen bond donors	-1.18	10.89
A partition coefficient $\log P$ between 7 and 5	# of acceptors	-2.47	7.86
Only non-stereochemistry compounds	Balaban's J-value	-2.53	12.86
No salts	Molar refractivity	-4.08	3.40
Molecules that RDKit could not process were removed	Topological polar surface area	-2.26	8.68

3.2.2.3 NC-GRU Fingerprints

Molecular descriptors are critical to chemoinformatics because they encode crucial chemical information about molecules in a manner that computers can understand [182]. The benefit of the AutoEncoder models over the traditional fingerprints is their capacity to learn a broad range of molecules and provide encoded data in the appropriate latent space - fingerprints [209]. This study uses the ChEMBL 28 dataset to train the suggested NC-GRU AutoEncoder. The NC-GRU AutoEncoder's Decoder must output molecular sequence in the exact representation - Canonical SMILES. As previously indicated [37], GRU is one of the best-optimized models for generating neural fingerprints; nevertheless, the recently developed NC-GRU cell [145] exhibits superior theoretical characteristics than GRU cell. The NC-GRU is equipped with orthogonality methods to identify and preserve long-term dependencies. However, its gates also aid in erasing unnecessary information from memory. Our proposed

NC-GRU FingerPrints (NC-GRU FP), which inherit these cutting-edge properties, would deliver durable molecular descriptors suitable for various applications. As a result, the NC-GRU fingerprint vector representation between the encoder and decoder is expected to comprehend the input molecular sequence more thoroughly than GRU-based fingerprints. We evaluated NC-GRU fingerprint performance on several prediction tasks, including toxicity, solubility, partition coefficient, and solvation-free energy predictions, using seven benchmark datasets to show the value of our fingerprints compared to existing methods.

3.2.2.4 AutoEncoder Translation Accuracy

We examined training accuracies on ChEMBL 28 to show the benefits of the NC-GRU-based AutoEncoder over the GRU-based one. We trained models for 100,000 iterations, and for every 1,000 iterations, we recorded test accuracy results.

The corresponding findings are displayed in Figure 3.7, where NC-GRU- and GRU-based AutoEncoders have two hidden layers with sizes 160 and 320. (we observed a similar performance with three-layer AutoEncoders). When using NC-GRU AutoEncoder, we noticed a considerable improvement in training accuracy vs. GRU AutoEncoder, particularly in the early training. However, in the later epochs, the accuracy of both models reach 99%.

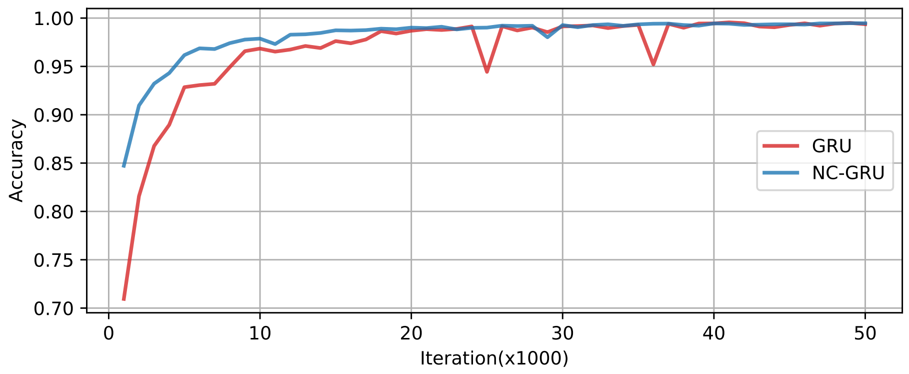


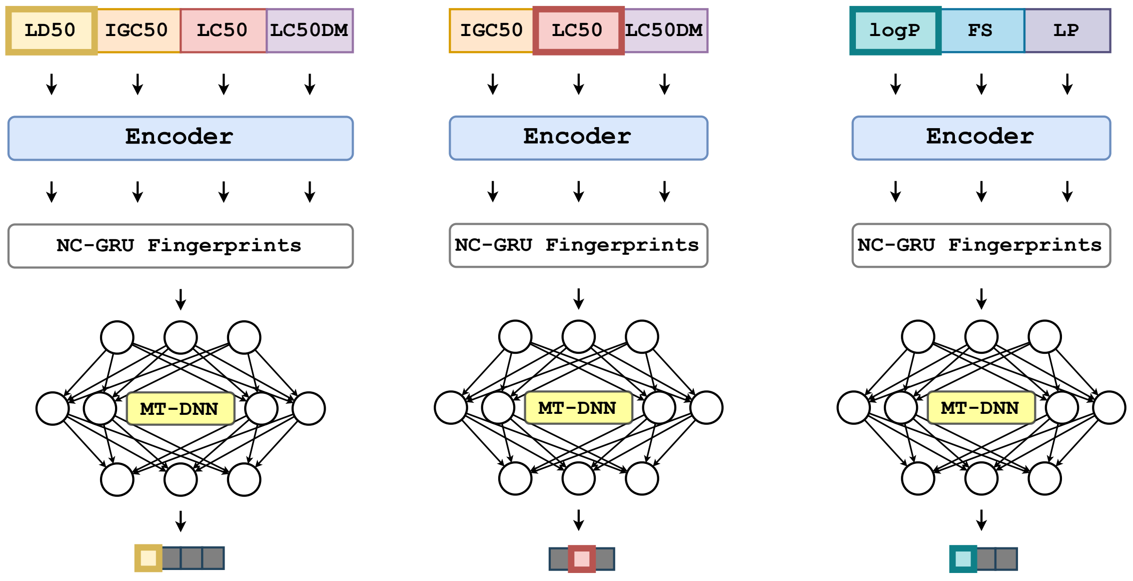
Figure 3.7: **NC-GRU vs. GRU Accuracies:** Comparison of Testing Accuracies between NC-GRU and GRU-based AutoEncoders on CheMBL 28 Dataset

We believe that the earlier-faster convergence of NC-GRU-based AutoEncoder creates more reliable and robust AutoEncoder fingerprints based on our tests with molecular property prediction tasks.

3.2.3 Prediction Models with Molecular Fingerprints

By identifying patterns in input data, a prediction model helps to forecast future results. Numerous machine and deep learning algorithms have shown to be quite successful in applications requiring prediction, including predicting molecular characteristics. The classical predicting models include linear and logistic regressions, logistic classification, k-nearest neighbors, support vector machine [43], etc. More recently, deep learning methods based on ideas from algebraic topology [21, 22], differential geometry [149], geometric graph theory [14, 150], and algebraic graph theory [148] show promising results on predictive modeling. However, several sophisticated algorithms combine the aforementioned techniques with molecular fingerprints to increase convergence speed and provide a more accurate model: Gradient Boosting Decision Tree (GBDT) [170], Random Forest (RF) [175], Single-Task Deep Neural Networks [9], Multi-Task Deep Neural Networks [26], etc. These methods employ fingerprints as input into prediction models because they include more structural information about compounds, particularly stereochemical descriptions, than chemical formulae or other non-neural fingerprint representations. Such algorithms frequently demonstrate high efficiency when using two- or three-dimensional molecular fingerprints.

We use multitask neural networks in our prediction studies to enhance the accuracy of predicting molecular properties.



(a) LD50 and IGC50 (b) LC50 and LC50DM (c) logP, FreeSolv (FS), and LipoPhilicity(LP)

Figure 3.8: MT-DNN models for prediction tasks;

3.2.3.1 Multitask Deep Neural Network

Multi-Task Deep Neural Network (MT-DNN) [25] or simply a multitask model is a technique for learning numerous tasks or activities simultaneously. Multitask networks have been used effectively in various applications, including natural language processing [41], computer vision [66], speech recognition [47], and drug discovery [24, 62, 159, 200, 213]. The multitask model training method uses a combined representation of trainable parameters to learn from various jobs and improve performance. It gains strength by simultaneously learning several datasets. However, the multitask model relies significantly on the notion that the input datasets are correlated. The number of tasks used in the input data determines how many neurons are present in the output layer of the multitask network architecture. Despite the fact that the output layer contains several neurons, the loss function only considers the specific output that corresponds to the input task when updating the parameters. For instance, the four-task multitask model begins training by using a batch of data from the first task and updating the shared and first task output weights without affecting the output weights of the other tasks. Then the multitask model goes to the second dataset and trains the shared and only the second task output weights after it has finished with the full dataset for the first task; this is also known as task-epoch training. The third and fourth tasks are then completed as part of the procedure. This process comprises a single complete epoch of multitask training with four tasks. Note that multitask models are trained using standard backpropagation algorithms such as Stochastic Gradient Descent (SGD), RMSProp [181], and Adam [110] while using a single optimizer throughout all the training and tasks.

Figure 3.8 illustrates the multitask models implemented in our experiments.

3.2.3.2 Prediction Tasks and Datasets

We have used a total of seven datasets in our experiments over four prediction tasks. This section describes each task and some information about each of the used datasets.

Table 3.4: **Prediction Datasets:** Selected Statistics for Prediction Tasks Datasets; " - " - no Valid. (validation) data; Part. Coeff. - Partition Coefficient.

Dataset	**Train**	**Valid.**	**Test**	**Min. Value**	**Max. Value**	**Units**	**Task**
LD50	7,413	-	1,482	0.291	7.201	$-\log_{10}$ mol/L	Toxicity
IGC50	1,434	-	358	0.334	6.36	$-\log_{10}$ mol/L	Toxicity
LC50	659	-	164	0.037	9.261	$-\log_{10}$ mol/L	Toxicity
LC50DM	283	-	70	0.117	10.064	$-\log_{10}$ mol/L	Toxicity
logP	8,199	-	406	-4.64	8.42	n/a	Part. Coeff.
FreeSolv	513	65	65	-25.47	3.43	kcal/mol	Free energy
Lipophilicity	3,360	420	420	-1.5	4.5	n/a	Solubility

Toxicity Prediction Datasets

For public health, toxicology forecasting is essential. One of the most significant applications of toxicity prediction is reducing the cost and effort of a drug's preclinical and clinical studies. Because of the anticipated toxicity, many pharmacological investigations can be avoided. We employed four datasets for our toxicity prediction experiments: the oral rate LD50 (LD50), the 40-hour Tetrahymenapyriformis IGC50 (IGC50), the 96-hour fathead minnow LC50 (LC50), and the 48-hour Daphnia Magna LC50DM (LC50DM).

The LD50 [2, 134] task measures the number of chemicals that can kill half of the rat population when orally ingested. The IGC50 [5, 218] records the 50% growth inhibitory concentration of Tetrahymena pyriformis organism after 40 hours. The LC50 [1, 135] reports the concentration of test chemicals in the water in milligrams per liter that cause 50% of fathead minnows to die after 96 hours. Lastly, the LC50DM [1, 135] represents the concentration of test chemicals in the water in milligrams per liter that cause 50% Daphna Magna to die after 48 hours. The unit of toxicity reported in these four datasets is $-\log_{10}$ moles per liter (mol/L). Note that the small size of the LC50 and LC50DM datasets and LD50 data being very uncertain [62] are only some of the reasons why training these datasets can be very difficult and challenging. Table 3.4 provides more information and some selected statistics about these datasets.

Partition Coefficient Prediction Dataset

We have been using the logP dataset to complete the task of partition coefficient prediction. The expression "logP" stands for the logarithm of the octanol-water partition coefficient of a chemical compound. The ratio of a compound's concentrations in a two-phase equilibrium system is known as the partition coefficient. It is a quantitative approach to describe lipophilicity, the ability to dissolve, which affects a pharmaceutical drug's elimination, toxicity, distribution, metabolism, and absorption. Note that the Food and Drug Administration (FDA) has approved all the components in the test data as organic drugs. The logP values for the partition coefficient data are compiled by [34]. See Table 3.4 for selected statistics about the logP dataset.

Lipophilicity Prediction Dataset

A drug's potency, distribution, and elimination in the body are determined by its lipophilicity. The dataset for the current study is selected from the CheMBL database, yielding 4,200 compounds [207], with the distribution coefficient of octanol/water at pH 7.4 serving as the basis for the lipophilicity index. See Table 3.4 for selected statistics about this dataset.

Solvation Free Energy Prediction Dataset

A transfer of a solute molecule from an ideal gas to water calls Solvation-free energy. To understand the uncertainty in predicting the binding free energy between tiny molecules and proteins, it is necessary to characterize solvation-free energy precisely. A lot of people are interested in this for computer-aided drug discovery. The solvation-free energy dataset utilized in this study is FreeSolv, which contains 643 molecules and is created in [144]. Then, MoleculeNet [207] suggested dividing it into training, validation, and testing subsets in an 80/10/10 split. See Table 3.4 for selected statistics about this dataset.

3.2.3.3 Optimized FingerPrints for Prediction Tasks

Multitask models may greatly enhance prediction tasks, as described in section 3.2.3.1. However, a near-optimal AutoEncoder structure will provide desired molecular representations, further enhancing downstream prediction networks. This section describes how we choose the near-optimal GRU and NC-GRU AutoEncoders to provide more desired molecular fingerprints for the datasets used for prediction tasks and the following multitask training.

According to the research presented in the NC-GRU paper [145], a model's performance may be enhanced by altering the number of layers and the gate initializations.

We investigated and evaluated six AutoEncoder fingerprint extraction models based on this claim: two GRU-based models with two and three layers, and four NC-GRU-based AutoEncoders with two and three layers and two different initializations (He Normal [80] and Glorot Uniform [68]).

We have considered a Single-Task Deep Neural Network (ST-DNN) model with a fully-connected architecture to select preferable fingerprints for a prediction job and future multitask training. For prediction datasets with more than 1,000 data points, the single-task model has two fully-connected layers with dimensions of 256 and 128. The dimensions for datasets with fewer than 1,000 molecules are 128 and 64. One of the main reasons why we consider two different models is the issue of overfitting. Our studies confirmed that smaller datasets are more likely to overfit on structures with higher complexity [152]. All single-task models were trained for 1,000 epochs using Adam [110] optimizer with the learning rate $5 \cdot 10^{-3}$ and the batch size of 32. Table 3.5 lists the above hyperparameters.

Table 3.5: **ST-DNN Prediction Model hyperparameters:** d.p. - data points; ReLU - Rectified Linear Unit [60]

Hyperparameter	**Values**	
Input size	512	
Hidden sizes	$>$ 1K d.p.	$<$ 1K d. p.
	256, 128	128, 64
Activation function	ReLU	
Batch size	32	
Optimizer	Adam [110]	
Learning rate	$5 \cdot 10^{-3}$	

By comparing the average r^2 and RMSE values over ten random runs of the single-task models where we use 10-Fold Cross-Validation (CV) for the datasets without validation sets (LD50, IGC50, LC50, LC50DM, and logP) and validation sets for the ones with one (Lipophilicity and FreeSolv) to select an appropriate AutoEncoder fingerprint extraction model. Table 3.6 summarizes the selected AutoeEncoder architecture for each benchmark.

3.2.3.4 Multitask Deep Neural Network: Hyperparameters and Setup

Machine and deep learning algorithms can learn various properties from a molecular fingerprint. We apply multitask learning to create Multitask Deep Neural Network (see Section 3.2.3.1) models to predict toxicity, partition coefficient, solubility, and solvation-free energy. Multitask Deep Neural Network hyperparameters are provided in Table 3.7. Note that the input size of 512 corresponds to the latent representation vector from the fingerprint extraction AutoEncoder models. The multitask models have four hidden layers of dimensions, and all models were trained using an initial learning rate of 10^{-2} and a step-learning rate decay, where the initial learning rate was decayed to 10^{-3} after 2,000 epochs. For the FreeSolv dataset, we used a validation

Table 3.6: **NC-GRU/GRU Autoencoder hyperparameters**
With the exception of the datasets for FreeSolv and Lipophilicity, which utilize their own validation data instead, the autoencoder for each dataset is chosen using ten-fold cross-validation.

	NC-GRU		**GRU**
Dataset	**Hidden sizes**	**Gate Init.**	**Hidden sizes**
IGC50	160, 320	He Normal	160,320, 640
LC50	160, 320	Glorot Uniform	160, 320, 640
LC50DM	160, 320, 640	He Normal	160, 320
LD50	160, 320, 640	He Normal	160, 320, 640
logP	160, 320	He Normal	160, 320, 640
FreeSolv	160, 320	He Normal	160, 320
Lipophilicity	160, 320, 640	He Normal	160, 320

set to determine the termination of training criteria. Moreover, we apply the Batch Normalization [94] on every task to enhance the models' predictive power.

Table 3.7: **MT-DNN Prediction Model hyperparameters:** * - this batch size was chosen to minimize the cutoff of data in the last batch training; † - initial learning rate was used for the first 2,000 epochs and then reduced to 10^{-3}; ‡ - validation set was used to determine the termination of training criteria for FreeSolv dataset

Hyperparameter	**Values**
Input size	512
Hidden sizes	1024, 512, 256, 64
Activation function	`ReLU` [60]
Batch size	18^*
Initial Learning rate	$10^{-2\dagger}$
Learning rate decay	True†
Number of epochs	3,000‡
Optimizer	SGD
Momentum	0.5
Batch Normalization [94]	True

There is a physicochemical link between the toxicity datasets; see [62] for more details. During training toxicity datasets, we use this assumption to create two Multitask Deep Neural Network models. One model uses all of the toxicity data to train the LD50 and IGC50 predictors, and another for training the LC50 and LC50DM without using the LD50 dataset; Figures 3.8a and 3.8b depict those models. Due to the high levels of uncertainty in the LD50 dataset [62], which may be detrimental for multitasks learning when test datasets are small, the LD50 dataset was excluded from the second model. Similarly, logP, FreeSolv, and Lipophilicity datasets have a chemical correlation. We use three of them altogether to implement the Multitask Deep Neural Network model; see Figure 3.8c.

Note: We do not use transfer learning of any kind; instead, we use ChEMBL 28 dataset pretrained AutoEncoder to obtain the fingerprints for prediction datasets. Following that, the obtained fingerprints were fed into the prediction models.

3.2.4 Experiments and Results

Table 3.8: **Results on Toxicity Prediction Tasks**; "*" - result from our experiments

	Toxicity Data (r^2 ↑)			
Dataset	**IGC50**	**LC50**	**LC50DM**	**LD50**
NC-GRU (ours)	**0.816**	**0.759**	0.785	0.634
AGBT$_s$-FP [32]	0.805	0.75	**0.83**	0.612
GRU*	0.813	0.73	0.706	0.6
BTAMDL1 [98]	0.721	0.750	0.7	0.605
HybridModel [103]	0.81	0.678	0.616	0.629
Daylight [62]	0.717	0.724	0.694	0.617
Estate1 [62]	0.735	0.694	0.684	0.605
Hierarchical [134]	0.719	0.710	0.695	0.578
Estate2 [62]	0.742	0.662	0.623	0.589
Nearest Neighbour [134]	0.600	0.667	0.733	0.557
FDA [134]	0.747	0.626	0.565	0.557
MACCS [62]	0.643	0.608	0.434	**0.643**
FP2 [62]	0.681	0.609	0.357	0.631
ECFP [62]	0.647	0.573	0.452	0.586
Pharm2D [62]	0.384	0.528	0.275	0.443
ERG [62]	0.274	0.348	0.336	0.392

We present the results of various experiments to demonstrate the reliability and effectiveness of the proposed NC-GRU fingerprints using four types of molecular properties: toxicity, partition coefficient, solubility, and solvation-free energy predictions where we have used seven benchmark datasets: IGC50, LC50DM, LC50, LD50, logP, Lipophilicity, and FreeSolv; see section 3.2.3.2 for details about these tasks and datasets. To validate our suggested models, we compare them to existing models that incorporate two- and three-dimensional molecular fingerprints, including results for GRU-based fingerprints as a baseline. The performance of the models is measured in the squared Pearson correlation coefficient (r^2) for IGC50, LC50DM, LC50, LD50, and logP datasets, and the Root Mean Square Error (RMSE) for FreeSolv and Lipophilicity datasets.

The NC-GRU and GRU results reported in Tables 3.8, 3.9, and 3.10 are the consensuses among five random seeds to limit the variation in deep learning model

Table 3.9: **Partition Coefficient Data Results**
MT-DNN is the model applied in NC-GRU and GRU to obtain the prediction. r^2 is the measure used in all of the toxic data. " - " denotes the results not provided in the original paper. " * " results from our experiments

Dataset	**logP**($r^2 \uparrow$)
NC-GRU (ours)	**0.913**
GRU*	**0.913**
AGBT$_s$-FP [32]	0.905
ESTD1 [205]	0.893
Estate2 [62]	0.893
XLOGP3 [34]	0.872
Estate1 [62]	0.870
MACCS [62]	0.867
ECFP [62]	0.857
ESTD2 [205]	0.848
XLOGP3-AA [34]	0.847
AG-FP [32]	0.838
CLOGP [34]	0.838
Daylight [62]	0.819
TOPKAT [34]	0.815
xlogp2 [34]	0.800
alogp98 [34]	0.777
KOWWIN [34]	0.771
HINT [62]	0.491

performances. In addition, we include the performance of earlier models in these tables. Our NC-GRU fingerprint-based models show encouraging results, coming in first place in three out of seven tests. The best r^2 values are specifically obtained by NC-GRU predictors on the IGC50 (0.816) and LC50 (0.759) datasets. The NC-GRU model has the lowest RMSE of 0.757 kcal/mol for the solvation-free energy prediction challenge. Our NC-GRU is ranked second on the LC50DM and LD50 benchmarks, with r^2 coefficients of 0.785 and 0.634, respectively. The top model on LC50DM is AGBT$_s$-FP [32] (0.830) and the best performance on LD50 is MACCS [62] (0.643). Our model is the fourth-ranked predictor in the Lipophilicity dataset; however, it is still superior to the GRU model, with RMSE=0.688, whereas Chemprop [211] is the top predictor, with RMSE=0.555. In all the interested experiments, we include GRU FP-based models for a direct comparison with its successor, NC-GRU. We include GRU fingerprint-based models for a direct comparison with its successor, NC-GRU, in all aforementioned tests. As seen in Tables 3.8, 3.9, and 3.10, our NC-GRU outperforms GRU in all the benchmarks except for the logP task, where both models produce the same r^2 of 0.913.

In addition, Figure 3.9 provides the comparison plots of the NC-GRU vs. GRU vs. target data points for the seven prediction datasets mentioned above. We also

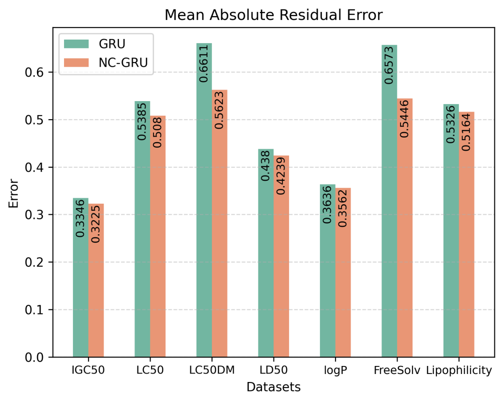
Table 3.10: **Solubility and Solvation Free Energy Data Results**
MT-DNN is the model applied in NC-GRU and GRU to obtain the prediction. RMSE is the measure used in all of the toxic data. “ - ” denotes the results not provided in the original paper. “ * ” results from our experiments

Dataset	**FreeSolv** (RMSE $\downarrow$)	**Lipophilicity** (RMSE $\downarrow$)
NC-GRU (ours)	**0.757**	0.688
GRU*	0.882	0.704
Chemprop [211]	1.075	**0.555**
AGBT$_s$-FP [32]	1.039	0.579
MMNB [171]	1.155	0.625
GraphConv [207]	1.150	0.715

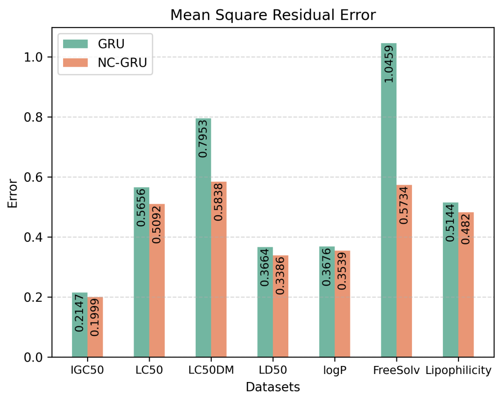
computed the mean absolute and the mean square residual errors, provided in Figures 3.9a and 3.9b, respectively.

3.2.5 Conclusion

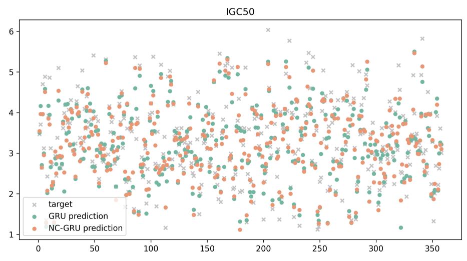
A fingerprint-based AutoEncoder with a Gated Recurrent Unit (GRU) can reliably depict molecules to predict molecular properties. But unfortunately, the GRU-AutoEncoder systems cannot attain cutting-edge precision when dealing with various biological datasets because of the exploding gradient problem and long-term dependency restriction. This issue inspired us to create an improved GRU variant called NC-GRU, where orthogonal weights help to capture small molecular structures more effectively.



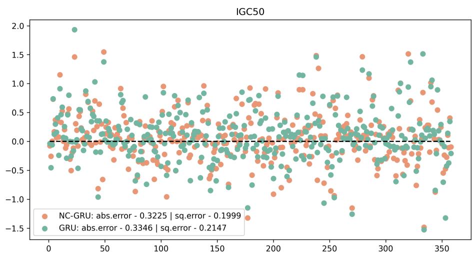
(a) Mean Absolute Residual Error



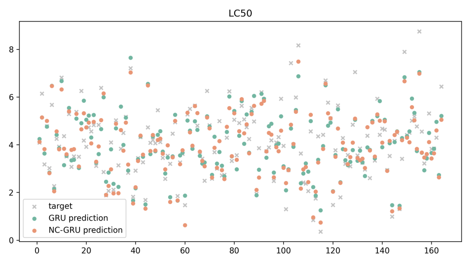
(b) Mean Square Residual Error



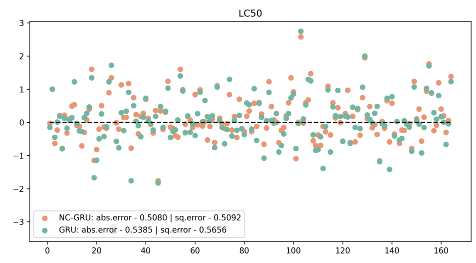
(c) NC-GRU and GRU predicted points for IGC50 with ground truth (target)



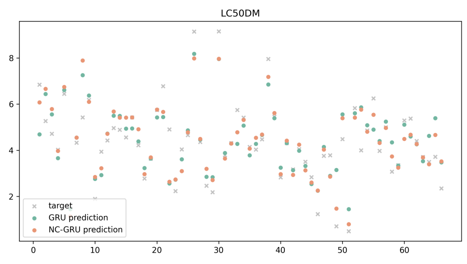
(d) NC-GRU and GRU residual plots for IGC50



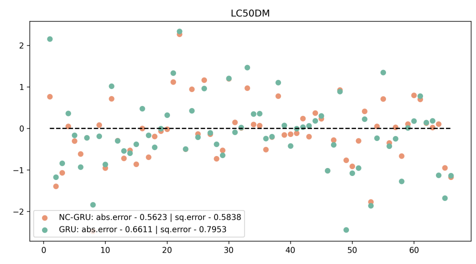
(e) NC-GRU and GRU predicted points for LC50 with ground truth (target)



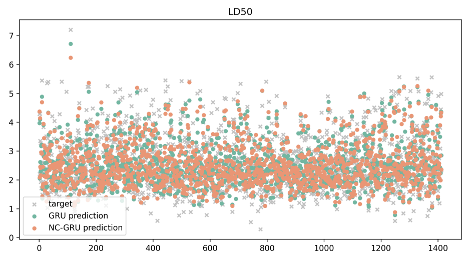
(f) NC-GRU and GRU residual plots for LC50



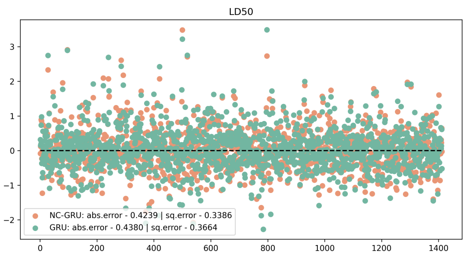
(g) NC-GRU and GRU predicted points for LC50DM with ground truth (target)



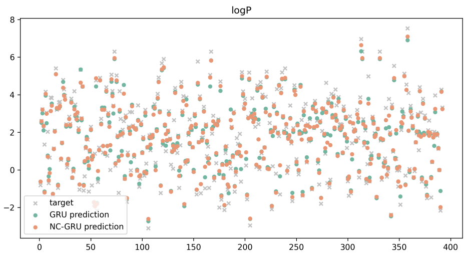
(h) NC-GRU and GRU residual plots for LC50DM



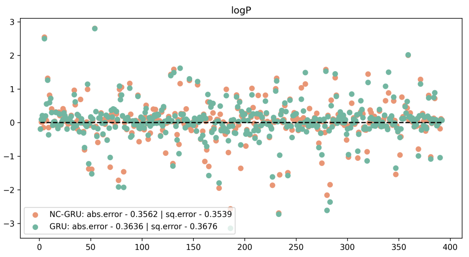
(i) NC-GRU and GRU predicted points for LD50 with ground truth (target)



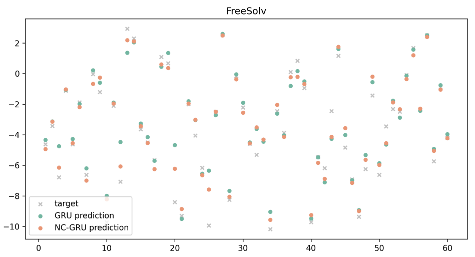
(j) NC-GRU and GRU residual plots for LD50



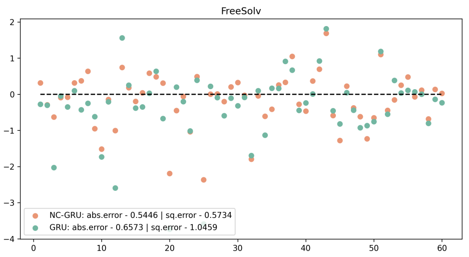
(k) NC-GRU and GRU predicted points for logP with ground truth (target)



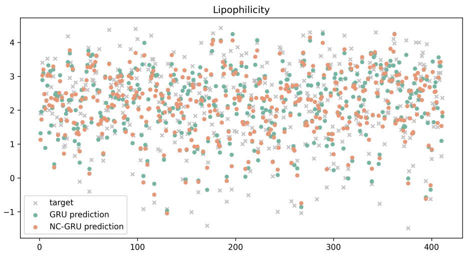
(l) NC-GRU and GRU residual plots for logP



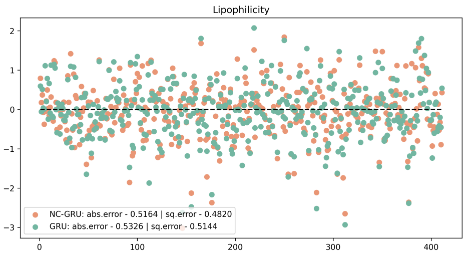
(m) NC-GRU and GRU predicted points for FreeSolv with ground truth (target)



(n) NC-GRU and GRU residual plots for FreeSolv



(o) NC-GRU and GRU predicted points for Lipophilicity with ground truth (target)



(p) NC-GRU and GRU residual plots for Lipophilicity

Figure 3.9: **Predictions and Residual Plots:** Predictions (GRU and NC-GRU) vs. Target and Residual plots with mean absolute and mean square residual errors for IGC50, LC50, LC50DM, LD50, logP, FreeSolv, and Lipophilicity datasets

3.3 Breaking Time Invariance: Assorted-Time Normalization for Recurrent Neural Networks

It has been demonstrated that normalization techniques like Layer Normalization and Batch Normalization efficiently enhance the training of Recurrent Neural Networks (RNNs). However, current approaches only normalize using the data available at a single point in time. The final result of such normalizations is a preactivation state with a time-independent distribution. The RNN with such architecture does not consider the temporal connections of the input. However, RNN weights are shared across temporal dimensions, and it could also be helpful to introduce such links across time steps into normalization. We propose a normalization technique called Assorted-Time Normalization (ATN), which preserves information from multiple consecutive temporal steps and then normalizes using those steps. This configuration can extend the conventional normalizing techniques' temporal dependencies without adding additional trainable parameters. To demonstrate the weight scaling invariance property, we provide theoretical derivations for the gradient backpropagation. In addition, our tests showed steady improvement when ATN is added to Layer Normalization on several synthetic and real-world tasks.

3.3.1 Introduction

Some of the primary architectures used for modeling time-series data in Deep Learning today include the Recurrent Neural Network (RNN) [90, 164], and variants such as Gated Recurrent Unit (GRU) [37, 99, 145] or Long Short Term Memory (LSTM) [88]. Unfortunately, these recurrent models are susceptible to issues with exploding gradients and over-fitting, even though LSTMs and GRUs are good at avoiding issues with vanishing gradients. One of the most effective concepts that have been proposed is the normalization of RNN preactivation states using techniques like Layer Normalization (Layer Norm) [8] and Batch Normalization (Batch Norm) [42, 95]. All the mentioned methods use statistics from a particular time step to recenter and rescale the preactivation data. Such techniques help expedite training and eliminate exploding gradients by controlling the model's states and gradients' norms.

These normalizing techniques have proved themselves effective, but when applied to RNNs, they do not consider variation across the temporal dimension. For instance, the Layer Norm or Batch Norm models are independent of the norm of the input vector at each time step because they are invariant to the scaling in the input at every time step. This might have disastrous effects depending on the situation. Moreover, these methods produce a preactivation state with an invariant distribution across the temporal dimension. Such time invariance architectural design of RNN limits it to use temporal dependencies fully. Including this reliance in the normalization procedure would be logical since RNNs also share weights across the temporal direction. In the literature, [42] mentions an unsuccessful attempt to involve averaging statistics across time; however, they presented it with few details. Simply averaging over the entire temporal dimension is an overcorrection that makes the statistics susceptible to diluted averages and loses effectiveness further into the sequence. We are considering collecting the mean and variance across a smaller subsequence where one can benefit from these time dependencies without overly weakening the impact of a single time step.

We present a normalization technique called Assorted-Time Normalization, or simply ATN, which preserves information from multiple consecutive temporal steps and then normalizes using those steps. Other normalization techniques, such as Layer Norm and Batch Norm that normalize input data along the none-temporal dimension can be coupled with our ATN method. It can account for the temporal dependencies in a way that prior approaches could not by keeping a short-term memory of the preceding k time steps. We use that memory to calculate the statistics, which we normalize, resulting in an output with a regulated mean and variance capable of varying across time steps. We may avoid the issues that arise from utilizing all or none of the sequences by using a small subsequence at each time point. In addition, we can choose the most appropriate number of subsequences (k value) for the dataset of interest. This procedure adapts without introducing additional trainable parameters since it gives the normalization method a temporal component.

We provide theoretical derivations for the gradient backpropagation to demonstrate the weight scaling invariance property. Our tests show consistent improvement when ATN is added to Layer Normalization on several synthetic and real-world tasks.

3.3.2 Related Work

Batch Normalization [95] (BN) was one of the early efforts to employ a normalizing approach throughout model layers. Fully Connected (FC) and Convolutional (CNN) Neural Networks were the original architectures to employ Batch Normalization to normalize network activations across the batch dimension. Batch Normalization is frequently credited as offering a more dependable and speedier training schedule while enhancing generalizations. In contrast to Batch Normalization, the Instance Normalization (IN) [188] approach behaves like contrast normalization and has mostly been used for image datasets.

The research points out that normalizing the instances is helpful since the output-styled pictures should not rely on the contrast of the input image content. The Group Normalization (GN) [206] approach, which is generally used for Convolutional networks, normalizes a three-dimensional feature in a convolutional layer by grouping the features in the group in all three dimensions after separating the channels into groups.

Consider a standard Recurrent Neural Network (RNN) cell:

$$h^{(t)} = f\left(W_h h^{(t-1)} + W_x x^{(t)} + \beta_h\right) \tag{3.61}$$

$$y^{(t)} = W_y h^{(t)} + \beta_y \tag{3.62}$$

where f is some nonlinear activation function defined by the application.

Recurrent Batch Normalization [42] reduces the internal covariate shift across successive time steps by applying Batch Normalization to the hidden-to-hidden and memory cell components of the Long Short-Term Memory (LSTM) model [88]. Weight

Normalization (WN) is a technique that was suggested by [166]. They propose to alter the network settings to accelerate training by decoupling the magnitude from the direction of the weight vector. Unfortunately, given that Weight Normalization is less stable than Batch Normalization [67], it does not seem to be employed as much in practice.

Layer Normalization [8] (LN) was introduced to normalize activations along the hidden dimension for Fully-Connected and Recurrent Neural Networks. Since then, Layer Normalization has gained much popularity in Recurrent-like networks. The preactivation state of the Recurrent Network is normalized by Layer Normalization as follows:

$$h^{(t)} = f\left(LN\left(W_h h^{(t-1)}\right) + LN\left(W_x x^{(t)}\right) + \beta_h\right) \tag{3.63}$$

$$y^{(t)} = LN\left(W_y h^{(t)}\right) + \beta_y \tag{3.64}$$

where the LN (Layer Normalization) operator is defined by

$$\mu_t = \frac{1}{n}\sum_{i=1}^{n} a_i^{(t)} \quad \sigma_t^2 = \frac{1}{n}\sum_{i=1}^{n}\left(a_i^{(t)} - \mu_t\right)^2 \tag{3.65}$$

$$y^{(t)} = LN(a^{(t)};\gamma,\beta) = \gamma \odot \frac{a^{(t)} - \mu_t}{\sqrt{\sigma_t^2 + \varepsilon}} + \beta \tag{3.66}$$

with $\odot$ - Hadamard product (entrywise multiplication). A setup like this makes it easier to apply to RNNs and helps remove the Batch Normalization batch reliance.

More recently, Adaptive Normalization (AdaNorm) [208] conducted a thorough analysis of Layer Normalization and came to the conclusion that the backward gradients of the mean and variance within the Layer Normalization method are more important than the rescaling and recentering factors, γ and β, in (3.66). They also suggested a brand-new approach called AdaNorm, which substitutes a new transformation function for weight and bias.

3.4 Assorted Time Normalization

Calculating the normalization statistics at each time step leads to a post-normalization state with mean and variance that are invariant across temporal dimension, which is an unfavorable aspect of the adaption of Layer Normalization to Recurrent Nets. Due to this, the model cannot accurately capture the changing distributions over time, which may be essential for modeling sequential data. For instance, the normalization $LN\left(W_x x^{(t)}\right)$ in (3.64) is invariant to scaling in $x^{(t)}$, which prevents the model from learning the changing norm of $x^{(t)}$. A bias in the linear term, frequently utilized in implementations, could help lessen this effect. Most of what was previously discussed also can be applied to Batch Normalization.

To overcome this temporal invariance, we suggest a novel normalizing approach. Consider a sequence $\mathbf{a} = \left\{a^{(t)}\right\} \subset \mathbb{R}^n$ produced in a Recurrent Net, such as the

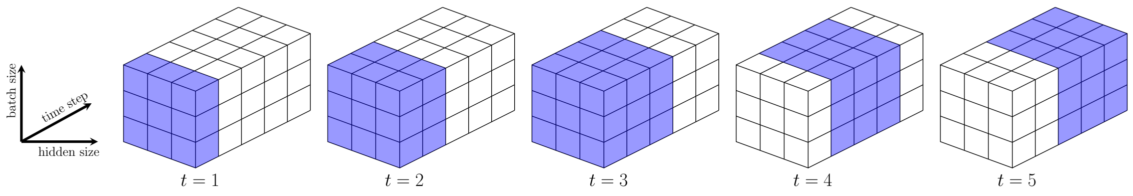


Figure 3.10: **ATN Method:** Illustration of the Assorted Time Normalization method combined with Layer Normalization using three-time steps (i.e., $k = 3$). Consider the preactivation state tensor in $\mathbb{R}^{5\times3\times3}$. At $t = 1$, we use a standard Layer Normalization; at $t = 2$, we normalize using information from time steps $1, 2$; at $t = 3$ Assorted Time Normalization method uses information from time step $t = 1, 2, 3$; at $t = 4$, we normalize with respect to time steps $t = 2, 3, 4$; and so on after that.

preactivation state that we wish to normalize. At time step t of the model, we maintain a memory of the previous k entries, $\mathbf{a}_k^{(t)} = \left\{a^{(t-k+1)}, \ldots, a^{(t-1)}, a^{(t)}\right\} \subset \mathbf{a}$, in the normalization layer, using this extended set we compute the mean and variance to use in the normalization layer. Such idea and approach can be combined with other normalization methods, e.g., Layer or Batch Normalizations. Combining with Layer Normalization, we calculate statistics at time-step t as follows:

$$\mu_{t,k} = \frac{1}{nk} \sum_{j=0}^{k-1} \sum_{s=1}^{n} a_s^{(t-j)} \tag{3.67}$$

$$\sigma_{t,k}^2 = \frac{1}{nk} \sum_{j=0}^{k-1} \sum_{s=1}^{n} \left(a_s^{(t-j)} - \mu_{t,k}\right)^2 \tag{3.68}$$

Figure 3.10 visually depicts our Assorted Time Normalization method.

After these statistics are calculated, we normalize only the current term $a^{(t)}$ and optionally recenter and rescale using γ and β, two trainable parameters shared across time while adding a small epsilon to the variance to prevent division by zero, similar to the Layer Normalization method in (3.66).

$$y^{(t)} = ATN(\mathbf{a}_k^{(t)}; \gamma, \beta) := \gamma \odot \frac{a^{(t)} - \mu_{t,k}}{\sqrt{\sigma_{t,k}^2 + \varepsilon}} + \beta \tag{3.69}$$

We use several time steps in our statistic computations, which vary from the method in (3.66) and result in a double sum rather than a single one for Layer Normalization. With only one word in the set being replaced at each time step, this definition of the statistics is more stable over time, at least for large k. This leads to a normalized output that does not have a mean and variance that are uniform over time steps. We contend that this is advantageous for sequential tasks. This capacity for variation enables the model to account for shifting input norms throughout the sequence, adding details about the distribution that were lost with earlier approaches.

The statistics may be computed using all prior terms in the series, although they will vary more in the earlier time steps than in the later ones. The statistics will change gradually over time if just k time steps of the series are kept, and we can reduce the memory and computing expenses, which might be substantial for lengthy sequences.

Effectively, using data from numerous time steps gives a larger set on which to calculate statistics. This makes it possible to approximate the underlying data distribution better. In other words, Assorted Time Normalization uses statistics over a larger set that is more stable across time so that the normalized state can retain more variations in time. In contrast, conventional normalization techniques create a normalized state that is time-invariant by using high-frequency statistics at each time step. The Assorted Time Normalization network particularly depends on scaling the input vector at a time step, while Layer and Batch Normalizations do not. However, Assorted Time Normalization preserves the desirable weight scaling invariant property, which we show as follows:

Let H and $\tilde{H}$ be weight matrices for two sets of model parameters, θ and $\tilde{\theta}$ respectively, which differ by a scaling factor of δ, i.e. $\tilde{H} = \delta H$. Then the outputs of Assorted Time Normalization are the same:

$$\tilde{y}^{(t)} = \frac{\gamma}{\tilde{\sigma}_{t,k}} \odot \left(\tilde{H}a^{(t)} - \tilde{\mu}_{t,k}\right) + \beta \ = \frac{\gamma}{\sigma_{t,k}} \odot \left(Ha^{(t)} - \mu_{t,k}\right) + \beta = y^{(t)} \tag{3.70}$$

where $\tilde{\sigma}_{t,k} = \delta\sigma_{t,k}$ and $\tilde{\mu}_{t,k} = \delta\mu_{t,k}$. This invariance property makes the Assorted Time Normalization network independent of the weight matrix H norm, reducing the exploding/vanishing gradient problems. Assorted Time Normalization is likewise invariant to rescaling the entire input sequence, but it is not invariant to rescaling only one element in the sequence. See [157] for a complete list of invariant properties.

We backpropagate the gradients with respect to the model parameters during training. Propagating the gradient via the normalization layer with Assorted Time Normalization, $\frac{\partial y_i^{(t)}}{\partial a_i^{(t-m)}}$, is an important step. The formulae for calculating these derivatives are provided in the following statement.

Proposition 1. *Consider ATN for a sequence* $\mathbf{a} = \{a^{(t)}\} \subset \mathbb{R}^n$ *produced in a RNN and let* $y^{(t)} = ATN(\mathbf{a}_k^{(t)}; \gamma, \beta)$. *Then, for* $0 \leq m \leq k-1$, *we have:*

$$\frac{\partial y_i^{(t)}}{\partial a_i^{(t-m)}} = \gamma \odot \frac{\frac{\partial a_i^{(t)}}{\partial y_i^{(t-m)}} \frac{\partial y_i^{(t-m)}}{\partial a_i^{(t-m)}} - \frac{\partial \mu_{t,k}}{\partial a_i^{(t-m)}}}{\sqrt{\sigma_{t,k}^2 + \varepsilon}} - \gamma \odot \frac{a_i^{(t)} - \mu_{t,k}}{2\left(\sigma_{t,k}^2 + \varepsilon\right)^{3/2}} \frac{\partial \sigma_{t,k}^2}{\partial a_i^{(t-m)}} \tag{3.71}$$

where

$$\frac{\partial \mu_{t,k}}{\partial a_i^{(t-m)}} = \frac{1}{nk} \sum_{j=0}^{m} \frac{\partial a_i^{(t-j)}}{\partial a_i^{(t-m)}} \tag{3.72}$$

$$\frac{\partial \sigma_{t,k}^2}{\partial a_i^{(t-m)}} = \frac{2}{nk}\sum_{j=0}^{m}\left(a_i^{(t-j)} - \mu_{t,k}\right)\frac{\partial a_i^{(t-j)}}{\partial a_i^{(t-m)}} - \sum_{j=0}^{k-1}\sum_{s=1}^{n}\left(a_s^{(t-j)} - \mu_{t,k}\right)\frac{\partial \mu_{t,k}}{\partial a_i^{(t-m)}}. \tag{3.73}$$

See [157] for the proof of the above proposition. Also, the computations of $\frac{\partial y_i^{(t)}}{\partial \beta}$ and $\frac{\partial y_i^{(t)}}{\partial \gamma}$ are straightforward and are omitted.

In our experiments, we equip Long Short-Term Memory (LSTM) networks with Assorted Time Normalization. Following [8] and [42], our Assorted Time Normalization method for LSTM is as follows :

$$\begin{pmatrix} f^{(t)} \\ i^{(t)} \\ o^{(t)} \\ g^{(t)} \end{pmatrix} = ATN(W_h h^{(t-1)}) + ATN(W_x x^{(t)}) + b \tag{3.74}$$

$$c^{(t)} = \sigma(f^{(t)}) \odot c^{(t-1)} + \sigma(i^{(t)}) \odot \tanh(g^{(t)}) \tag{3.75}$$

$$h^{(t)} = \sigma(o^{(t)}) \odot \tanh(ATN(c^{(t)})) \tag{3.76}$$

where $\odot$ is the Hadamard product and $\sigma(\cdot)$ is the sigmoid function.

3.4.1 Experiments and Results

We have performed a series of experiments which include the Copying [89], Adding [89], and Denoise problems [57, 99] as well as Language Modeling on character level Penn Treebank dataset [132] and word level WikiText-2 dataset [137]. This section presents results on the language modeling task with word level Wikitext-2 dataset. Detailed experiments regarding others can be found in [157]. All experiments were run using Python 3.7.0, PyTorch 1.1.0, and CUDA 9.0 on a single NVIDIA Tesla V100 GPU.

3.4.1.1 WikiText-2

The WikiText-2 dataset was introduced in [137]. It is approximately two times the size of the Penn Treebank dataset and contains preprocessed Wikipedia articles while maintaining the original structure, punctuation, and symbols. The WikiText-2 dataset consists of approximately 2.2 million words: 2 million for the training set and 200 thousand for the validation and test sets, with a vocabulary size of 33,278. This task is a word-level Language Modeling problem with the goal of predicting the next word given the preceding sequence of words.

Implementation Details: We used a batch size of 32; three LSTM layers with embedding and hidden sizes of 400 and 1,150, respectively; BPTT values of 70; gradient clipping on the norm of 0.25; and learning rate of 30 with Stochastic Gradient Descent (SGD) optimizer without any momentum or learning rate decay, and switch to ASGD [156] optimizer using nonmono criteria from [136] with value 5 (our experiments showed that switching happens approximately between epochs 20 and 30 for all models: LSTM, LN, and ATN). The ATN model is implemented with a k value of 25.

Results: In this experiment, the ATN method shows improvement over LSTM and LN method in both training and validation perplexity (PPL); see Table 3.11.

Table 3.11: **WikiText-2 Results:** Attained minimum values. $\downarrow$ - denotes the smaller, the better result.

	PPL $\downarrow$	
	train	validation
LSTM	80.68	65.65
LN	80.24	58.0
ATN	78.55	56.06

3.4.2 Exploratory & Ablation Studies

3.4.2.1 Optimal k Value for ATN method

To highlight the importance of normalizing with respect to k time steps instead of just one or all of them, we present a study on various k values. In Figure 3.11, we present results on the Copying Problem [89] with $T = 100$, a full description of the task and dataset can be found in [157]. For this experiment, we have trained the Long Short-Term Memory model (LSTM), LSTM with Layer Normalization (LN), and three LSTM with Assorted Time Normalization models, which we abbreviate as ATN(k), with values of k being 25, 45, and 65 under the same conditions.

All Assorted Time Normalization (ATN) models perform better than LSTM and LSTM with LN. The ATN($k = 45$) model performs better than ATN($k = 25$), which should not be a surprise since the larger k value would mean we are normalizing with respect to a larger set and getting better statistics for the mean and variance; however, ATN($k = 65$) performs poorer than ATN($k = 45$) and even poorer than ATN($k = 25$) which suggests that too large k may actually degrade the result. This may be due to numerical difficulties in propagating derivative through k steps in ATN for a large k.

3.4.2.2 Post Normalization Statistics

In Figures 3.12a, 3.12b, and 3.12c, we present the statistics of the post-normalization components from a single iteration of training for the Adding Problem [89] with $T = 75$, a full description of the task and dataset can be found in [157]. We present the statistics from four different models, an LN-LSTM, and three ATN(k) models with k values of 5, 25, and 55. All of the models did not include the use of trainable bias and gain parameters inside the normalization methods.

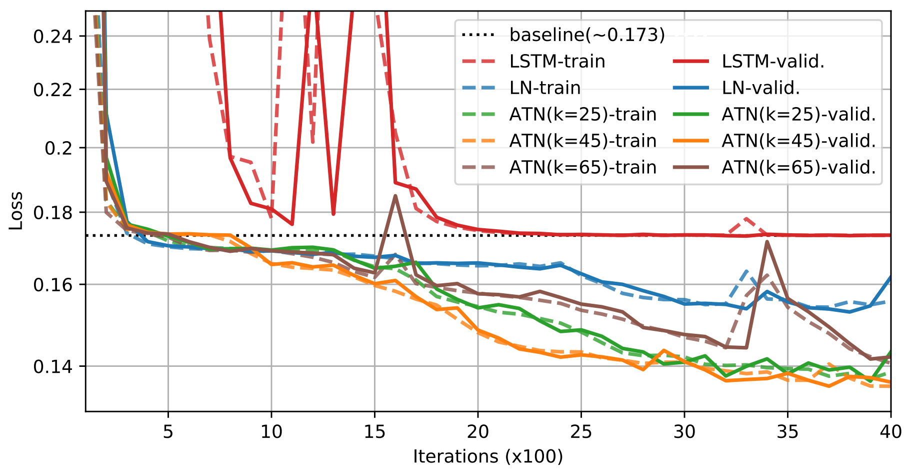


Figure 3.11: **Ablation Study on Optimal k Value:** k value study in ATN method

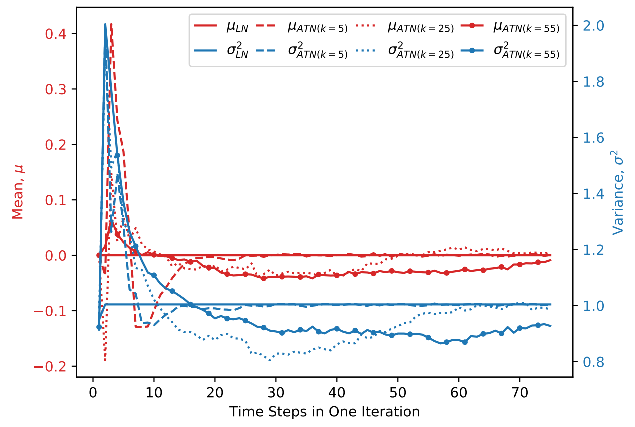
In Figure 3.12a, we show the mean and variance after normalization of the product of the hidden-to-hidden weight and the hidden state, $W_h h^{(t-1)}$. While Layer Normalization produces constant mean and variance, the ATN method allows the statistics to vary at each time step, resulting in curves that do not differ too much from those for LN in terms of scale but demonstrate the natural fluctuations in the hidden states. From this, we can see that we are achieving the combination of a controlled output that can still reflect the temporal changes of the network.

In Figure 3.12b, we show the statistics from the product of the input-to-hidden weight and the input, $W_x x^{(t)}$. The ATN model provides highly variable means and variances, showcasing the amount of information about the dataset which is lost when LN resets the statistics to these constant values.

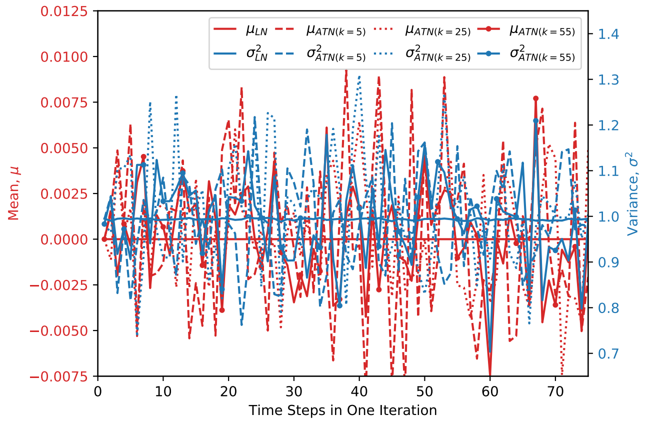
In Figure 3.12c, we show the post-normalization statistics of the memory cell, $c^{(t)}$. These statistics demonstrate the effect of a shorter k value instead of a longer one in the mean. In the early iterations for ATN($k = 5$), the mean has a larger spike that flattens to a bit above zero by the end. For the larger k values, this initially increased mean was maintained throughout a larger portion of the iteration, causing the lower values further along to have less influence on the statistics.

3.4.3 Conclusion

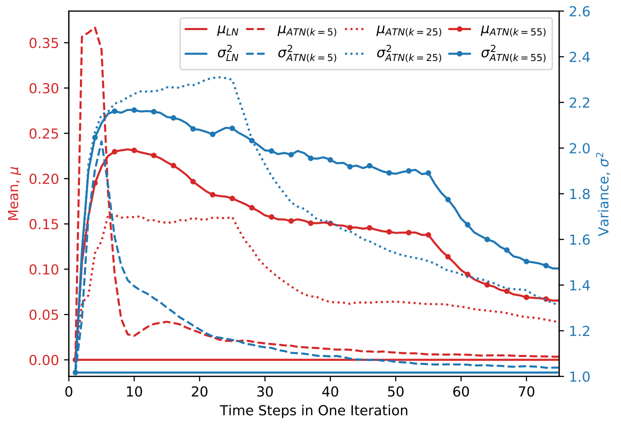
To overcome the temporal invariance of the conventional normalization methods, we have developed a technique for applying statistics-based normalization methods to Recurrent Neural Networks (RNNs). In addition to demonstrating the retention of invariance to rescaling the weight matrix, we have given analytical findings on this technique's effect on the model's gradients. Our tests show that the ATN-LSTM outperforms the Layer Normalization (LN) for Long Short-Term Memory (LSTM) model on training and testing outcomes. Our approach presents a significant alternative for enhancing RNN performance, given the prevalence of LN in real-world settings.



(a) Hidden-to-Hidden



(b) Input-to-Hidden



(c) Memory Cell

Figure 3.12: **Ablation Study on Post Normalization Statistics:** Post Normalization Statistics for Adding Problem with $T = 75$

Chapter 4 Convolutional Neural Networks

4.1 SCNN: Symmetry Structured Convolutional Neural Network

This section studies convolutional neural networks (CNNs) with symmetric spatially organized 2D features. Such networks find applications in areas with pairwise relationships, for example, modeling for sequential recommendation problems and RNA and protein sequences secondary structure inference problems, etc. We developed a CNN architecture that produces and maintains the CNN's symmetry structure. In addition, we present parameterizations for the convolutional kernels that create update rules to maintain symmetry during the training are presented. We use the sequential recommendation task to test our architecture and update rules, and the findings demonstrate that symmetric structured networks perform better while using fewer machine-trainable parameters.

4.1.1 Introduction

Convolutional Neural Networks (CNNs) were first created to model visual data; however, they have been adapted to numerous other issues. Contrary to image domains, some applications could have non-obvious structural features, for example, symmetry in the spatial dimensions. We are interested in such architectural variants of CNNs that can provide symmetry-structured features to increase performance for such tasks while lowering the computational time and memory costs.

Our interest lies in characteristics that explain how elements of a 1-dimensional sequence interact with each other. A symmetric matrix can be used to explain this relationship. We want to create a 2-dimensional matrix with the entry (i, j) that describes the interaction of elements i and j from a 1-dimensional sequential input. The entries (i, j) and (j, i) of such a matrix will be identical since they represent the same mutual interaction of the elements i and j. Forming a self-Cartesian of the 1-dimensional series, which a CNN can subsequently process, is one of the popular methods to extract a 2-dimensional interaction feature from a 1-dimensional sequence of n individuals. A general CNN does not generate symmetric features from the self-Cartesian input because the self-Cartesian product is not symmetric. However, even if the CNN's input is symmetric, a standard convolution layer does not maintain that symmetry. In addition, if we can generate and preserve the symmetry, storing the entire symmetric matrix is unreasonable; thus, only the lower (or upper) triangular portion of feature maps is needed to represent all the interactions. Therefore, it is desirable to alter the conventional CNN designs to enable the generation and preservation of symmetric features for improved interpretability and effectiveness.

We propose the Symmetry-Generating and Symmetry-Preserving Convolutional Layers, which provide symmetric feature maps over the spatial dimensions. In addition, we present a backpropagation reparametrization technique that could be utilized for updating symmetry-generating and symmetry-preserving networks. We demonstrate how the network's overall performance and training may be enhanced by symmetry using three application problems: the sequential recommendation problem [210], the RNA secondary structure prediction problem [201], and the Protein Contact map prediction problem [197]. The Recommender Systems [86, 178, 210] are aimed at suggesting relevant items to users given the sequence of previously interacted items. The hidden interaction features are matrices processed by a CNN. Then the feature is naturally symmetric and can be better captured using our symmetry-structured CNNs. On the other hand, the RNA Secondary Structure Inference problem [75] is to find the native structure of a given RNA sequence of nucleotide, and the Protein Prediction Problems [4, 197] is to predict the contact map from the sequence profile and predicted structure, and co-evolution and pairwise potential features. CNNs are widely used in these problems to produce a 2D matrix that describes the secondary structure or the contact map. Such a matrix is symmetric and can benefit from using our symmetry-structured CNNs. Information and results regarding the last two experiments can be found in [129]. Our experimental results show that imposing the symmetric structure increases the network's capacity for prediction while lowering network trainable parameters and, therefore, the computational and memory costs.

4.1.2 Related Work

To recognize patterns in medical picture data, CNNs with rotation and shift-invariant kernels were proposed in [124]. The same team also created the CNN with wavelet kernels, also known as CNN/WK [123], and the CNN with circular kernels - CNN/CK [125]. Because each updated convolution kernel in the CNN/WK network was made to be orthonormal, features chosen for the transform domain are linearly independent. As a result, the fully connected layers of CNN's classification level may work more efficiently. There have been other suggested improvements to the CNN structure, for example, the kernel structure and different from our method symmetric structured CNNs [65, 172]. Recently, [35, 131] suggested rotation invariant CNNs based on Fourier Transform. In [35], a novel scheme is proposed using the magnitude response of the 2D-discrete-Fourier transform to encode rotational invariance in neural networks, along with a new, efficient convolutional scheme for encoding rotational equivariance throughout convolutional layers.

Deep symmetry networks [65] (symnets) form feature maps over any symmetry group. The symnets propose a generalization of ConvNet and use kernel-based interpolation to tie parameters tractably and pool over symmetry spaces of any dimension based on symmetry group theory, which is a relatively different and complicated architecture compared to our model.

When an input transformation does not include interpolation, the transformationally identical CNN [122] (TI-CNN) aims to use families of transformationally identical vectors that can cause the CNN to give a quantitatively same output over a sequence of operations in the CNN. The kernel of the TI-CNN employs a dihedral symmetric group structure made up of reflection and $90°$ rotation. The group-equivariant CNNs [40] (G-CNNs) broaden the CNN convolution operation from summing over the spatial translations to an operation termed G-correlation that sums over all transformations of a symmetry group G. These convolutions can capture characteristics that are invariant under a symmetry group. These approaches differ from ours in calculating features that identify potential symmetry invariance in the input rather than attempting to construct symmetric output features. Note that the transpose operation may be included in such a symmetry group.

Structured receptive field networks [97] express the kernel as a weighted sum of a smooth and compact filter basis which learns this structured kernel from the basis. It improves efficiency and generalization. Structure pruning [20] also imposes certain structures on the CNN kernels for the pruning of parameters. A conditional generative neural decoding approach [52] explores using CNNs as a decoder to generate a structured multi-output for fMRI data through a constrained optimization. In our work, we use an explicit kernel structure, i.e., a special basis of the kernel, to achieve a symmetric output exactly.

Four distinct degrees of symmetry are added to convolutional kernels for image applications as a regularizer to enhance generalizations [53]. These degrees of symmetry force the convolution kernels to be invariant under horizontal flips, horizontal and vertical flips, and 90° rotation. However, their networks' feature maps may have different degrees of symmetry. Our approach varies from theirs in that it generates and preserves symmetric features in the spatial dimensions using symmetry-structured networks, and we need the kernel to have symmetry in both the channel and spatial dimensions.

To overcome CNNs' limitations in handling non-Euclidean structured data, for example, traffic flow data on traffic networks, relational data on social networks, and active data on molecule structure networks, structure-aware convolution [28] (SACNNs) has been developed. Structure-aware convolution combines local inputs with various topological structures using a single shared filter. Using the function approximation theory, SACNN generalizes the classical filters to univariate functions that can be parameterized efficiently and effectively. It also introduces local structure representations to express topological features quantitatively.

4.1.3 Symmetry Structured CNN

We are drawn to structures with convolution layers that are symmetric in the spatial dimension. A convolution layer feature $\mathbf{Z} = [\mathbf{Z}_{i,j,k}] \in \mathbb{R}^{n\times n\times c}$ is symmetric if $\mathbf{Z}_{i,j,:} = \mathbf{Z}_{j,i,:}$ for all i, j. This kind of structure appears in the CosRec architecture [210] pairwise embedding structure in the Sequential Recommendation Problem, RNA secondary structure inference problem [201], and protein contact map prediction problem [197]. We present two new kernels for CNNs: symmetry-generating and symmetry-preserving kernels that can be used in the CNNs and propose update rules together with the reparametrization technique to be used during the backpropagation stage.

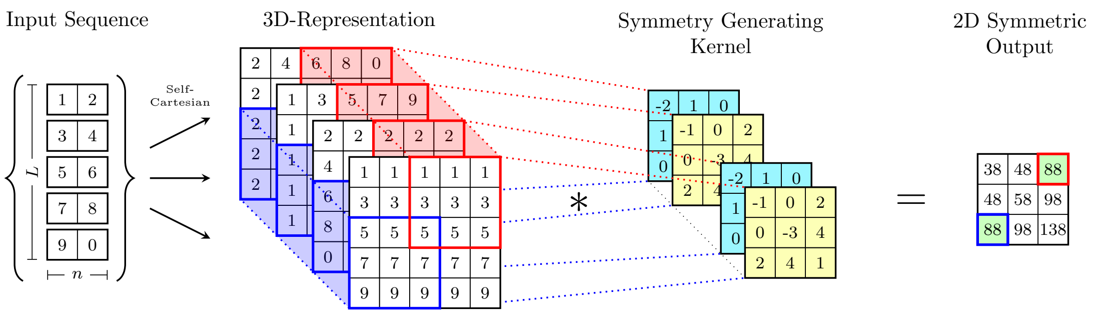


Figure 4.1: **Symmetry-Generating Layer:** Illustration of a symmetry-generating convolutional layer for one output channel: from an input of a vector sequence of dimension L, a self-Cartesian product produces a 3D $L \times L \times 2n$ representation. For one output channel, a $C \times C \times 2n$ ($C = 3$ here) kernel has a symmetry in the channel direction and, for each channel, is symmetric in the spatial dimensions. This results in output with symmetric features.

4.1.3.1 Symmetry-Generating Kernel

We look at creating a 2-dimensional feature matrix using CNNs to describe certain subject interactions in a 1-dimensional sequence. We must construct two-dimensional structures before applying CNNs to a series of inputs. Utilizing the self-Cartesian product of the 1-dimensional input feature is one of the most popular approaches to achieve this. Consider a 1-dimensional sequence $x = (x^{(1)}, \ldots, x^{(L)}) \in \mathbb{R}^{n \times L}$ where $x^{(\ell)} = \left[x^{\ell}_{k}\right]_{k=1}^{n} \in \mathbb{R}^n$ then the self-Cartesian product of this sequence with itself is the tensor $y = [y_{i,j,k}]_{i,j,k=1}^{L,L,2n} \in \mathbb{R}^{L \times L \times 2n}$ defined as

$$y_{i,j,k} = \begin{cases} x_k^{(i)} & \text{if} \quad 1 \leq k \leq n, \\ x_{k-n}^{(j)} & \text{if} \quad n+1 \leq k \leq 2n. \end{cases} \tag{4.1}$$

Under such conditions, $y(i,j,:)$ is the i^{th} and j^{th} terms of $x^{(i)}$ and $x^{(j)}$ stacked together; and tensor y is not symmetric. In order to produce a symmetric output from the self-Cartesian product y, we need to convolve it with a kernel. We call such a convolutional kernel a symmetry-generating kernel.

Consider a convolutional layer that inputs the self-Cartesian product y. Let $C, 2n$, and F be the kernel size, the number of input channels, and the number of output channels, respectively. We say $\mathbf{W} = [\mathbf{W}_{i,j,k,f}] \in \mathbb{R}^{C \times C \times 2n \times F}$ is a symmetry-generating kernel if

$$\mathbf{W}_{i,j,:,:} = \mathbf{W}_{j,i,:,:} \quad \text{and} \quad \mathbf{W}_{i,j,k,:} = \mathbf{W}_{i,j,n+k,:} \tag{4.2}$$

for any i, j and any $k \leq n$.

Theorem 6. *Let y be the self-Cartesian product of the 1D sequential input x of the shape $[L, n]$. Consider a convolutional layer with $\boldsymbol{W} \in \mathbb{R}^{C \times C \times 2n \times F}$ as the kernel and y as the input. If $\boldsymbol{W}$ is a symmetry-generating kernel, then the convolution layer's output is symmetric.*

See Figure 4.1 for an illustration of such a convolution layer and see Appendix A of [129] for proof.

Notation: To avoid the cumbersome notation, let $\dfrac{\partial L}{\partial \mathbf{W}^{(k)}}$ denotes $\dfrac{\partial L}{\partial \mathbf{W}}\left(\mathbf{W}^{(k)}\right)$

We first initialize the kernel as a symmetry-generating kernel. But if we train the network as normal, let's say using a gradient descent step

$$\mathbf{W}^{(k+1)} = \mathbf{W}^{(k)} - \alpha \frac{\partial L}{\partial \mathbf{W}^{(k)}}, \tag{4.3}$$

we are not guaranteed the symmetry of $\mathbf{W}^{(k)}$ since $\dfrac{\partial L}{\partial \mathbf{W}^{(k)}}$ has no underlying structure of $\mathbf{W}^{(k)}$. To keep the symmetric structure throughout the training, we introduce reparametrization of $\mathbf{W} \in \mathbb{R}^{C\times C\times 2n\times F}$ with $\mathbf{S} = [\mathbf{S}_{l,k,f}] \in \mathbb{R}^{\frac{C(C+1)}{2}\times n\times F}$ as follows

$$\mathbf{W}_{i,j,k,f} = \begin{cases} \mathbf{S}_{l,k,f} & \text{if } i \le j,\ l = \frac{1}{2}(j-1)j + i,\ k \le n, \\ \mathbf{S}_{l,k,f} & \text{if } i > j,\ l = \frac{1}{2}(i-1)i + j,\ k \le n, \\ \mathbf{S}_{l,k-n,f} & \text{if } i \le j,\ l = \frac{1}{2}(j-1)j + i,\ k > n, \\ \mathbf{S}_{l,k-n,f} & \text{if } i > j,\ l = \frac{1}{2}(i-1)i + j,\ k > n. \end{cases} \tag{4.4}$$

Under such reparametrization, $\mathbf{W} = \mathbf{W}(\mathbf{S}) \in \mathbb{R}^{C\times C\times 2n\times F}$ is defined by $\mathbf{S}_{l,k,f}$ with $1 \le k \le n$ and $1 \le l \le \frac{1}{2}C(C+1)$. Proposition 2 guarantees that l can be uniquely written as $l = \frac{1}{2}(j-1)j + i$ under the condition that $i \le j$.

Proposition 2. *For every $l \in \left\{1, \ldots, \frac{1}{2}C(C+1)\right\}$, there exist unique $i, j \in \{1, \ldots, C\}$ and $i \le j$, such that $l = \frac{1}{2}(j-1)j + i$.*

Proof.

Since $1 \le l \le \frac{1}{2}C(C+1)$, there exists a unique $j \in \{1, \ldots, C\}$ such that $\frac{1}{2}(j-1)j < l \le \frac{1}{2}j(j+1)$. After such j is chosen, i is uniquely determined by $i = l - \frac{1}{2}(j-1)j$. Under this condition,

$$1 \le i = l - \frac{1}{2}(j-1)j \le \frac{1}{2}j(j+1) - \frac{1}{2}(j-1)j = j. \tag{4.5}$$

□

Trainable parameters **S** are updated using gradient descent during CNN training. Then we use these updated **S** to update **W** using Equation (4.4). In this case, for a single channel, our symmetry-generating kernel needs to be symmetric in spatial dimensions and has dimension $C \times C$, which is parameterized by $\mathbf{S}$, contributing only $C \times (C+1)/2$ trainable parameters. Additionally, there is a symmetrical behavior in the dimension of input channels, which is reduced to n with our parameterization $\mathbf{S}$ vs. $2n$ in the original. This reduces the number of kernel trainable parameters from $C^2 2nF$ to $C(C+1)nF/2$.

We may calculate the gradient of a loss function from its gradient with respect to $\mathbf{W}$ using $\mathbf{S}$ as trainable parameters. The update rule for a backpropagation algorithm may be found in the following theorem.

Theorem 7. *Let L be a differentiable loss function for a CNN with symmetry generating kernel $\boldsymbol{W}$, i.e., $L = L(\boldsymbol{W})$. Let $\boldsymbol{W} = [\boldsymbol{W}_{i,j,k,f}]$ be parameterized by $\boldsymbol{S} = [\boldsymbol{S}_{l,k,f}]$ as in (4.4). Then the gradient $\frac{\partial L}{\partial \boldsymbol{S}} = \left[\frac{\partial L}{\partial \boldsymbol{S}_{l,k,f}}\right]$ satisfies*

$$\frac{\partial L}{\partial \boldsymbol{S}_{l,k,f}} = \begin{cases} \frac{\partial L}{\partial \boldsymbol{W}_{i,j,k,f}} + \frac{\partial L}{\partial \boldsymbol{W}_{j,i,k,f}} + \frac{\partial L}{\partial \boldsymbol{W}_{i,j,k+n,f}} + \frac{\partial L}{\partial \boldsymbol{W}_{j,i,k+n,f}} & \text{if } l = \frac{1}{2}j(j-1)+i,\ i<j, \\ \frac{\partial L}{\partial \boldsymbol{W}_{j,j,k,f}} + \frac{\partial L}{\partial \boldsymbol{W}_{j,j,k+n,f}} & \text{if } l = \frac{1}{2}j(j+1). \end{cases} \tag{4.6}$$

Proof.

For any $i < j$, $k \leq n$, and $f \leq F$, all the entries $\mathbf{W}_{i,j,k,f}$, $\mathbf{W}_{j,i,k,f}$, $\mathbf{W}_{i,j,k+n,f}$, and $\mathbf{W}_{j,i,k+n,f}$ are equal and are parameterized by $\mathbf{S}_{l,k,f}$, where $l = \frac{1}{2}(j-1)j + i$. Thus the gradient term for $i < j, l = \frac{1}{2}(j-1)j + i$ is

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{S}_{l,k,f}} &= \sum_{s,t,u,v} \frac{\partial L}{\partial \mathbf{W}_{s,t,u,v}} \frac{\partial \mathbf{W}_{s,t,u,v}}{\partial \mathbf{S}_{l,k,f}} \\ &= \frac{\partial L}{\partial \mathbf{W}_{i,j,k,f}} + \frac{\partial L}{\partial \mathbf{W}_{j,i,k,f}} + \frac{\partial L}{\partial \mathbf{W}_{i,j,k+n,f}} + \frac{\partial L}{\partial \mathbf{W}_{j,i,k+n,f}}, \end{aligned} \tag{4.7}$$

where $\frac{\partial \mathbf{W}_{s,t,u,v}}{\mathbf{S}_{l,k,f}} = 0$ except for indices (i,j,k,f), (j,i,k,f), $(i,j,k+n,f)$, and $(j,i,k+n,f)$.

When $i = j$, $\mathbf{W}_{j,j,k,f}$ and $\mathbf{W}_{j,j,k+n,f}$ are parameterized by $\mathbf{S}_{\frac{1}{2}j(j+1),k,f}$. This yields the gradient:

$$\frac{\partial L}{\partial \mathbf{S}_{l,k,f}} = \sum_{s,t,u,v} \frac{\partial L}{\partial \mathbf{W}_{s,t,u,v}} \frac{\partial \mathbf{W}_{s,t,u,v}}{\partial \mathbf{S}_{l,k,f}} = \frac{\partial L}{\partial \mathbf{W}_{j,j,k,f}} + \frac{\partial L}{\partial \mathbf{W}_{j,j,k+n,f}}, \tag{4.8}$$

where $\frac{\partial \mathbf{W}_{s,t,u,v}}{\mathbf{S}_{l,k,f}} = 0$ for $(s,t,u,v) \notin \{(j,j,k,f),(j,j,k+n,f)\}$. From Equation (4.7) and Equation (4.8) we have the desired gradient as in Equation (4.6).

□

When we train a symmetry-generating kernel, we first calculate $\dfrac{\partial L}{\partial \mathbf{W}}$ using the standard backpropagation methods, and then we update $\dfrac{\partial L}{\partial \mathbf{S}}$ using Theorem 7 and $\dfrac{\partial L}{\partial \mathbf{W}}$. After that, we update $\mathbf{S}$ by $\mathbf{S}^{(k+1)} = \mathbf{S}^{(k)} - \alpha \frac{\partial L}{\partial \mathbf{S}^{(k)}}$, where α is the learning rate, and then finally, we obtain $\mathbf{W}^{(k+1)}$ using Equation (4.4).

4.1.3.2 Symmetry-Preserving Kernel

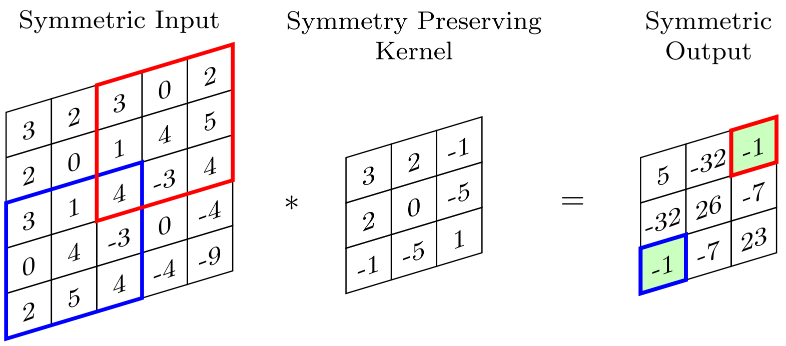


Figure 4.2: **Symmetry-Preserving Layer:** Symmetry-Preserving Convolution for 1 input and 1 output channel: given a symmetric input, applying a symmetric kernel to a windowed section and its transpose produces the same outputs.

A typical deep network requires more than one convolution layer. The input for further convolution layers would be already symmetric since it would be the output of the symmetry-generating convolutional layer. We introduce and employ a symmetry-preserving kernel $\mathbf{Q} = \left[\mathbf{Q}_{i,j,k,f}\right] \in \mathbb{R}^{C\times C\times m\times F}$ with

$$\mathbf{Q}_{i,j,:,:} = \mathbf{Q}_{j,i,:,:} \tag{4.9}$$

for any i and j to preserve the output's symmetry throughout training with the multiple-layer network. Figure 4.2 illustrates such process. Further, we use an aforementioned reparametrization technique to parameterize $\mathbf{Q}$ by $\mathbf{R} = [\mathbf{R}_{l,k,f}] \in \mathbb{R}^{\frac{C(C+1)}{2}\times m\times F}$ as:

$$\mathbf{Q}_{i,j,k,f} = \begin{cases} \mathbf{R}_{l,k,f} & \text{if } i \le j,\ l = \dfrac{1}{2}(j-1)j + i, \\ \mathbf{R}_{l,k,f} & \text{if } i > j,\ l = \dfrac{1}{2}(i-1)i + j. \end{cases} \tag{4.10}$$

Similar to symmetry-generating kernel case, $\mathbf{Q} = \mathbf{Q}(\mathbf{R})$ is defined by $\mathbf{R}$. Then, during the training, we update $\mathbf{Q}$ through updating $\mathbf{R}$:

$$\frac{\partial L}{\partial \mathbf{R}_{l,k,f}} = \begin{cases} \dfrac{\partial L}{\partial \mathbf{Q}_{i,j,k,f}} + \dfrac{\partial L}{\partial \mathbf{Q}_{j,i,k,f}} & \text{if } l = \dfrac{1}{2}(j-1)j + i,\ i < j, \\ \dfrac{\partial L}{\partial \mathbf{Q}_{j,j,k,f}} & \text{if } l = \dfrac{1}{2}j(j+1). \end{cases} \tag{4.11}$$

Consequently, this method reduces the number of trainable parameters in the convolutional kernel from C^2mF to $C(C+1)mF/2$ in $\mathbf{Q}$ and $\mathbf{R}$, respectively.

The 2-dimensional symmetry-preserving convolution kernel and its backpropagation updates from Equation (4.11) help to maintain the symmetry in all feature maps, including the final output feature maps for the symmetry-structured CNNs (SCNNs). See the section 4.1.4 and the Experiment section from [129] for experimental results and further details.

4.1.3.3 Complexity and Initialization

While improving the CNN model issues with underlying symmetric structures is our main goal, our designs save nearly half of the computational and memory expenses associated with the CNN layer(s) during training and inference. Our symmetry-generating kernel has $C(C+1)nF/2$ trainable parameters, compared with $C^2 2nF$ of a full kernel. Furthermore, for the forward propagation during training or inference, each feature map is symmetric in the spacial dimension requiring the memory of the lower (or upper, depending on the implementation) triangular part with $L(L+1)m/2$ entries for input and the lower triangular part with $L(L+1)F/2$ for output. This reduces the cost of memory by nearly half. Additionally, rather than computing the entire matrix, the convolution technique may be used to compute only the bottom triangular portion of the result, which roughly reduces the computational cost in half. To benefit from this, the conventional convolution algorithm must be modified.

Last but not least, we advise utilizing half of the usual initialization, such as Glorot [69], for the initialization of the trainable parameters $\mathbf{S}$ and $\mathbf{R}$ since each element of $\mathbf{S}$ and $\mathbf{R}$ contributes twice to the $\mathbf{W}$ and $\mathbf{Q}$, respectively. Therefore, in our studies, half-Glorot initialization is applied.

4.1.4 Experiments and Results

In this section, we compare the performance of a symmetry-structured CNN architecture with a corresponding traditional CNN. We mainly compare the two architectures by matching the architecture hyperparameters with respect to feature map dimensions and kernel dimensions. We study the application of CNNs where the feature maps are naturally symmetric, which the traditional CNN architecture ignores.

4.1.4.1 Sequential Recommendation

Recommender systems have become a core technology in many applications. In sequential recommendation models, each user is represented as a sequence of items

Table 4.1: **Sequential Recommendation Task Results:** Sequential Recommendation Problem - CosRec [210] vs. SCosRec. On both datasets, SCosRec outperforms the CosRec model in every metric. It also outperforms other neural networks models such as GRU4Rec [86] and Caser [178]; * - quoted from [210] and reproduced with our experiments.

Dataset	**Metric**	**GRU4Rec***	**Caser***	**CosRec***	**SCosRec**(our)
ML-1M	MAP	0.1440	0.1507	0.1883	**0.1970**
	Prec@1	0.2515	0.2502	0.3308	**0.3458**
	Prec@5	0.2146	0.2175	0.2831	**0.2920**
	Prec@10	0.1916	0.1991	0.2493	**0.2586**
	Recall@1	0.0153	0.0148	0.0202	**0.0223**
	Recall@5	0.0629	0.0632	0.0843	**0.0895**
	Recall@10	0.1093	0.1121	0.1438	**0.1519**
			# of trainable parameters:	1.983M	1.730M
Gowalla	MAP	0.0580	0.0928	0.0980	**0.1006**
	Prec@1	0.1050	0.1961	0.2135	**0.2171**
	Prec@5	0.0721	0.1129	0.1190	**0.1211**
	Prec@10	0.0782	0.0571	0.0884	**0.0898**
	Recall@1	0.0155	0.0310	0.0337	**0.0350**
	Recall@5	0.0529	0.0845	0.0890	**0.0920**
	Recall@10	0.0826	0.1223	0.1305	**0.1330**
			# of trainable parameters:	5.641M	5.383M

that interacted with in the past, and we aim to predict the next item or top N items that a user will likely interact with in the near future. The order of interaction implies that sequential patterns play an essential role where more recent items in a sequence significantly impact the next item. Some early works on this problem include [86, 178, 210].

Our experiment was motivated by the most competitive model (at the time of experimenting) called CosRec [210], which is based on CNNs. Having feature maps modeling interactions of items, it is well suited for applying our symmetric kernels to the CNN layers. We compare the performance of our new model Symmetric CosRec (SCosRec) with the CosRec model.

The sequential recommendation problem can be formulated as follows. Suppose we have a set of users $\mathcal{U} = u_1, u_2, \ldots, u_{|\mathcal{U}|}$ and a set of items $\mathcal{I} = i_1, i_2, \ldots, i_{|\mathcal{I}|}$. For each user $u \in \mathcal{U}$, given the sequence of previously interacted items $S^u = (S^u_1, \ldots S^u_{|S^u|})$, $S^u_i \in \mathcal{I}$, we seek to predict the next item to match the user's preferences. We follow the same setup given in the CosRec model that embeds the item matrix $E_{\mathcal{I}} \in \mathbb{R}^{|\mathcal{I}|\times d}$ and user matrix $E_{\mathcal{U}} \in \mathbb{R}^{|\mathcal{U}|\times d}$, where d is the latent dimensionality, e_i and e_u denote the ith and the uth rows in $E_{\mathcal{I}}$ and $E_{\mathcal{U}}$ respectively. Then for user u at time step t, we retrieve the input embedding matrix $E^L_{u,t} \in \mathbb{R}^{L\times d}$ by looking up the previous L items

$(S^u_{t-L}, \dots S^u_{t-1})$ in the item embedding matrix $E_{\mathcal{I}}$. Using pairwise encoding [210], CosRec is a CNN model that creates a three-way tensor $T^L_{(u,t)} \in \mathbb{R}^{L \times L \times 2d}$ on top of the input embeddings $E^L_{u,t}$. The input structure of the three-way tensor $T^L_{(u,t)}$ can be incorporated with our symmetry generating and symmetry preserving CNN layers to obtain a symmetric $T^L_{(u,t)}$ that better models the pairwise encoding. We call our model Symmetric CosRec or simply SCosRec.

We test our SCosRec model with hyperparameter matching architecture with the CosRec model on two standard benchmark datasets: *MovieLens-1M (ML-1M)* [77] and *Gowalla* [36]. We use the *MovieLens-1M (ML-1M)* version of a popular benchmark dataset for evaluating the performance of collaborative filtering algorithms, and the *Gowalla* dataset, which is a location-based social networking website uses time and location information from check-ins made by users.

Evaluation metrics: Results were evaluated in three top-N metrics: Mean Average Precision (MAP), Precision@N, and Recall@N with $N = 1$, 5, and 10.

Implementation Details: All experiments were implemented using Python 3.6.9 and PyTorch 1.1.0 on an NVIDIA Quadro P5000 GPU. Similarly to CosRec models, we used 2 *symmetric* convolution blocks with 2 layers in each. The latent dimension d is 50 and 100 for *ML-1M* and *Gowalla* datasets, respectively. Markov order L is 5, prediction of the next T items is 3, the learning rate is 10^{-3}, learning rate decay on plateau with reducing factor 0.15 and patience 3 with respect to the MAP metric, the batch size is 512, negative sampling rate is 3, and the dropout rate is 0.5.

Results: Indeed, SCosRec models improvements are between $3 - 10\%$ on *ML-1M* and $1 - 4\%$ on *Gowalla* datasets compared to the original CosRec models while using less trainable parameters, see the number of trainable parameters in Table 4.1. Results of our experiments are provided in Table 4.1.

4.1.5 Conclusion

We presented a new symmetry-structured convolutional neural network architecture, SCNN, which generates and preserves symmetric structures in CNN using symmetry-generating and symmetry-preserving kernels. Moreover, we introduced an update scheme and reparametrization technique to optimize trainable parameters, their training, and memory (computational and storage) used in the symmetry-generating and symmetry-preserving kernels. We have demonstrated that our SCNN architecture improves the performance of traditional CNN while having fewer trainable parameters, which saves computational costs at both training and inference.

Bibliography

[1] Ecotox. `https://cfpub.epa.gov/ecotox/`.

[2] National library of medicine. `https://chem.nlm.nih.gov/chemidplus/`.

[3] S. Abdulatif, K. Armanious, J. T. Sajeev, K. Guirguis, and B. Yang. Investigating cross-domain losses for speech enhancement. *2021 29th European Signal Processing Conference (EUSIPCO)*, pages 411–415, 2020.

[4] B. Adhikari, J. Hou, and J. Cheng. DNCON2: improved protein contact prediction using two-level deep convolutional neural networks. *Bioinformatics*, 34(9):1466–1472, 12 2017.

[5] K. S. Akers, G. D. Sinks, and T. W. Schultz. Structure–toxicity relationships for selected halogenated aliphatic chemicals. *Environmental toxicology and pharmacology*, 7(1):33–39, 1999.

[6] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[7] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1120–1128. JMLR.org, 2016.

[8] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.

[9] I. A. Basheer and M. Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3–31, 2000.

[10] Y. Bengio, P. Frasconi, and P. Simard. The problem of learning long-term dependencies in recurrent networks. In *Proceedings of 1993 IEEE International Conference on Neural Networks (ICNN '93)*, pages 1183–1195, San Francisco, CA, 1993. IEEE Press.

[11] M. Binkowski, R. D. Hjelm, and A. C. Courville. Batch weight for domain adaptation with mass shift. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1844–1853, 2019.

[12] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton. Demystifying mmd gans. *arXiv preprint arXiv:1801.01401*, 2018.

[13] S. Boll. Suppression of acoustic noise in speech using spectral subtraction. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(2):113–120, 1979.

[14] D. Bramer and G.-W. Wei. Multiscale weighted colored graphs for protein flexibility and rigidity analysis. *The Journal of chemical physics*, 148(5):054103, 2018.

[15] S. Braun, H. Gamper, C. K. A. Reddy, and I. Tashev. Towards efficient models for real-time deep noise suppression. *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 656–660, 2021.

[16] S. Braun and I. Tashev. A consolidated view of loss functions for supervised deep learning-based speech enhancement. *2021 44th International Conference on Telecommunications and Signal Processing (TSP)*, pages 72–76, 2021.

[17] A. Brock. BigGAN-PyTorch. `https://github.com/ajbrock/BigGAN-PyTorch`.

[18] A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

[19] A. E. Bulut and K. Koishida. Low-latency single channel speech enhancement using u-net convolutional neural networks. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6214–6218, 2020.

[20] Y. Cai, W. Hua, H. Chen, G. E. Suh, C. De Sa, and Z. Zhang. Structured pruning is all you need for pruning cnns at initialization, 2022.

[21] Z. Cang, L. Mu, and G.-W. Wei. Representability of algebraic topology for biomolecules in machine learning based scoring and virtual screening. *PLoS computational biology*, 14(1):e1005929, 2018.

[22] Z. Cang and G.-W. Wei. Integration of element specific persistent homology and machine learning for protein-ligand binding affinity prediction. *International journal for numerical methods in biomedical engineering*, 34(2):e2914, 2018.

[23] R. Cao, S. Abdulatif, and B. Yang. Cmgan: Conformer-based metric gan for speech enhancement. *arXiv preprint arXiv:2203.15149*, 2022.

[24] S. J. Capuzzi, R. Politi, O. Isayev, S. Farag, and A. Tropsha. Qsar modeling of tox21 challenge stress response and nuclear receptor signaling toxicity assays. *Frontiers in Environmental Science*, 4, 2016.

[25] R. Caruana. Multitask learning: A knowledge-based source of inductive bias. In *ICML*, 1993.

[26] R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[27] A. Cereto-Massagué, M. J. Ojeda, C. Valls, M. Mulero, S. Garcia-Vallvé, and G. Pujadas. Molecular fingerprint similarity search in virtual screening. *Methods*, 71:58–63, 2015.

[28] J. Chang, J. Gu, L. Wang, G. Meng, S. Xiang, and C. Pan. Structure-aware convolutional neural networks. In *Advances in Neural Information Processing Systems 31*, pages 11–20. Curran Associates, Inc., 2018.

[29] R. Chao, C. Yu, S.-W. Fu, X. Lu, and Y. Tsao. Perceptual contrast stretching on target feature for speech enhancement. *Proc. of INTERSPEECH*, 2022.

[30] T. Che, Y. Li, A. P. Jacob, Y. Bengio, and W. Li. Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*, 2016.

[31] T. Che, R. Zhang, J. Sohl-Dickstein, H. Larochelle, L. Paull, Y. Cao, and Y. Bengio. Your gan is secretly an energy-based model and you should use discriminator driven latent sampling, 2020.

[32] D. Chen, K. Gao, D. D. Nguyen, X. Chen, Y. Jiang, G.-W. Wei, and F. Pan. Algebraic graph-assisted bidirectional transformers for molecular property prediction. *Nature Communications*, 12(1):1–9, 2021.

[33] T. Chen, X. Zhai, M. Ritter, M. Lucic, and N. Houlsby. Self-supervised gans via auxiliary rotation loss. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12146–12155, 2019.

[34] T. Cheng, Y. Zhao, X. Li, F. Lin, Y. Xu, X. Zhang, Y. Li, R. Wang, and L. Lai. Computation of octanol- water partition coefficients by guiding an additive model with knowledge. *Journal of chemical information and modeling*, 47(6):2140–2148, 2007.

[35] B. Chidester, M. N. Do, and J. Ma. Rotation equivariance and invariance in convolutional neural networks, 2018.

[36] E. Cho, S. Myers, and J. Leskovec. Friendship and mobility: User movement in location-based social networks. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1082–1090, 08 2011.

[37] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[38] C. Chu, K. Minami, and K. Fukumizu. Smoothness and stability in gans. In *International Conference on Learning Representations*, 2020.

[39] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[40] T. S. Cohen and M. Welling. Group equivariant convolutional networks. *CoRR*, abs/1602.07576, 2016.

[41] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[42] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. C. Courville. Recurrent batch normalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[43] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[44] F. Dang, H. Chen, and P. Zhang. Dpt-fsnet: Dual-path transformer based full-band and sub-band fusion network for speech enhancement. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6857–6861, 2022.

[45] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Jan. 1997.

[46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[47] L. Deng, G. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8599–8603. IEEE, 2013.

[48] M. Diesendruck, E. R. Elenberg, R. Sen, G. W. Cole, S. Shakkottai, and S. A. Williamson. Importance weighted generative networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 249–265. Springer, 2019.

[49] H.-W. Dong and Y.-H. Yang. Towards a deeper understanding of adversarial losses under a discriminative adversarial network setting, 2019.

[50] S. A. Dorado-Rojas, B. Vinzamuri, and L. Vanfretti. Orthogonal laguerre recurrent neural networks. *34th Conference on Neural Information Processing Systems*, 2020.

[51] S. Doveh and R. Giryes. Degas: Differentiable efficient generator search, 2019.

[52] C. Du, C. Du, L. Huang, and H. He. Conditional generative neural decoding with structured cnn feature prediction, Apr. 2020.

[53] V. Dudar and V. Semenov. Use of symmetric kernels for convolutional neural networks. *CoRR*, abs/1805.09421, 2018.

[54] J. L. Durant, B. A. Leland, D. R. Henry, and J. G. Nourse. Reoptimization of mdl keys for use in drug discovery. *Journal of chemical information and computer sciences*, 42(6):1273–1280, 2002.

[55] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

[56] H. Favre and W. Powell. *Nomenclature of Organic Chemistry: IUPAC Recommendations and Preferred Names 2013.* International Union of Pure and Applied Chemistry. Royal Society of Chemistry, 2014.

[57] J. N. Foerster, J. Gilmer, J. Chorowski, J. Sohl-Dickstein, and D. Sussillo. Input switched affine networks: An rnn architecture designed for interpretability, 2016.

[58] S.-W. Fu, C.-F. Liao, Y. Tsao, and S.-D. Lin. MetricGAN: Generative adversarial networks based black-box metric scores optimization for speech enhancement. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2031–2041. PMLR, 09–15 Jun 2019.

[59] S.-W. Fu, C. Yu, T.-A. Hsieh, P. Plantinga, M. Ravanelli, X. Lu, and Y. Tsao. Metricgan+: An improved version of metricgan for speech enhancement. *arXiv preprint arXiv:2104.03538*, 2021.

[60] K. Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3):121–136, 1975.

[61] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[62] K. Gao, D. D. Nguyen, V. Sresht, A. M. Mathiowetz, M. Tu, and G.-W. Wei. Are 2d fingerprints still valuable for drug discovery? *Physical chemistry chemical physics*, 22(16):8373–8390, 2020.

[63] K. Gao, D. D. Nguyen, M. Tu, and G.-W. Wei. Generative network complex for the automated generation of drug-like molecules. *Journal of chemical information and modeling*, 60(12):5682–5698, 2020.

[64] A. Gaulton, A. Hersey, M. Nowotka, A. P. Bento, J. Chambers, D. Mendez, P. Mutowo, F. Atkinson, L. J. Bellis, E. Cibrián-Uhalte, et al. The chembl database in 2017. *Nucleic acids research*, 45(D1):D945–D954, 2017.

[65] R. Gens and P. M. Domingos. Deep symmetry networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2537–2545. Curran Associates, Inc., 2014.

[66] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[67] I. Gitman and B. Ginsburg. Comparison of batch normalization and weight normalization algorithms for the large-scale image classification, 2017.

[68] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[69] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, Sardinia, Italy, 2010. JMLR: W&CP.

[70] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[71] X. Gong, S. Chang, Y. Jiang, and Z. Wang. Autogan: Neural architecture search for generative adversarial networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.

[72] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[73] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang. Conformer: Convolution-augmented transformer for speech recognition. *ArXiv*, abs/2005.08100, 2020.

[74] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 5769–5779, Red Hook, NY, USA, 2017. Curran Associates Inc.

[75] R. R. Gutell, J. C. Lee, and J. J. Cannone. The accuracy of ribosomal rna comparative structure models. *Curr Opin Struct Biol*, 12(3):301–10, Jun 2002.

[76] L. H. Hall and L. B. Kier. Electrotopological state indices for atom types: a novel combination of electronic, topological, and valence state information. *Journal of Chemical Information and Computer Sciences*, 35(6):1039–1045, 1995.

[77] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), Dec. 2015.

[78] D. Harris and S. Harris. *Digital Design and Computer Architecture, Second Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2012.

[79] H. He, H. Wang, G.-H. Lee, and Y. Tian. Probgan: Towards probabilistic gan with theoretical guarantees. In *ICLR*, 2019.

[80] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[81] K. Helfrich, D. Willmott, and Q. Ye. Orthogonal Recurrent Neural Networks with Scaled Cayley Transform. *In Proceedings of ICML 2018, Stockholm, Sweden, PMLR 80*, 2018.

[82] K. Helfrich and Q. Ye. Eigenvalue normalized recurrent neural networks for short term memory, 2019.

[83] S. Heller, A. McNaught, S. Stein, D. Tchekhovskoi, and I. Pletnev. Inchi - the worldwide chemical structure identifier standard. *Journal of Cheminformatics*, 5(1):7, Jan 2013.

[84] M. Henaff, A. Szlam, and Y. LeCun. Recurrent orthogonal networks and long-memory tasks. In *Proceedings of the 33rd International Conference on Machine Learning (ICML 2017)*, volume 48, New York, NY, 2017. JMLR: W&CP.

[85] 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 I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6626–6637. Curran Associates, Inc., 2017.

[86] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939, 2016.

[87] Q. Hoang, T. D. Nguyen, T. Le, and D. Phung. MGAN: Training generative adversarial nets with multiple generators. In *International Conference on Learning Representations*, 2018.

[88] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[89] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[90] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[91] Y. Hu and P. C. Loizou. Evaluation of objective quality measures for speech enhancement. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(1):229–238, 2008.

[92] S. L. Hyland and R. Gunnar. Learning unitary operators with help from u(n). In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAI 2017)*, pages 2050–2058, San Francisco, CA, 2017.

[93] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[94] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[95] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[96] J. J. Irwin and B. K. Shoichet. Zinc - a free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling*, 45(1):177–182, 2005.

[97] J. Jacobsen, J. V. Gemert, Z. Lou, and A. M. Smeulders. Structured receptive fields in cnns. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2610–2619, Los Alamitos, CA, USA, jun 2016. IEEE Computer Society.

[98] J. Jiang, R. Wang, M. Wang, K. Gao, D. D. Nguyen, and G.-W. Wei. Boosting tree-assisted multitask deep learning for small scientific datasets. *Journal of chemical information and modeling*, 60(3):1235–1244, 2020.

[99] L. Jing, C. Gulcehre, J. Peurifoy, Y. Shen, M. Tegmark, M. Soljacic, and Y. Bengio. Gated orthogonal recurrent units: On learning to forget. *Neural Computation*, 31(4):765–783, 2019.

[100] L. Jing, Y. Shen, T. Dubcek, J. Peurifoy, S. Skirlo, Y. LeCun, M. Tegmark, and M. Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1733–1741. JMLR.org, 2017.

[101] A. Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard GAN. In *International Conference on Learning Representations*, 2019.

[102] W. Kahan. Is there a small skew cayley transform with zero diagonal? *Linear Algebra and its Applications*, 417(2):335–341, 2006. Special Issue in honor of Friedrich Ludwig Bauer.

[103] A. Karim, A. Mishra, M. H. Newton, and A. Sattar. Efficient toxicity prediction via simple features using shallow neural networks and decision trees. *Acs Omega*, 4(1):1874–1888, 2019.

[104] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *ICLR*, 2018.

[105] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of StyleGAN. In *Proc. CVPR*, 2020.

[106] I. Kavalerov, W. Czaja, and R. Chellappa. A multi-class hinge loss for conditional gans. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1290–1299, January 2021.

[107] E. Kim and H. Seo. SE-Conformer: Time-Domain Speech Enhancement Using Conformer. In *Proc. Interspeech 2021*, pages 2736–2740, 2021.

[108] S. Kim, P. A. Thiessen, E. E. Bolton, J. Chen, G. Fu, A. Gindulyte, L. Han, J. He, S. He, B. A. Shoemaker, J. Wang, B. Yu, J. Zhang, and S. H. Bryant. PubChem Substance and Compound databases. *Nucleic Acids Research*, 44(D1):D1202–D1213, 09 2015.

[109] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[110] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[111] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[112] J. Le Roux, H. Kameoka, N. Ono, and S. Sagayama. Fast signal reconstruction from magnitude stft spectrogram based on spectrogram consistency. In *Proc. DAFx*, volume 10, pages 397–403, 2010.

[113] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *NIPS*, 1989.

[114] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[115] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[116] M. Lezcano-Casado and D. Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3794–3803. PMLR, 09–15 Jun 2019.

[117] C.-L. Li, W.-C. Chang, Y. Cheng, Y. Yang, and B. Poczos. Mmd gan: Towards deeper understanding of moment matching network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2203–2213. Curran Associates, Inc., 2017.

[118] H. Li, S.-W. Fu, Y. Tsao, and J. Yamagishi. imetricgan: Intelligibility enhancement for speech-in-noise using generative adversarial network-based metric learning. *ArXiv*, abs/2004.00932, 2020.

[119] J. Li, A. Madry, J. Peebles, and L. Schmidt. On the limitations of first-order approximation in GAN dynamics. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3005–3013, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[120] J. Lim and A. Oppenheim. All-pole modeling of degraded speech. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(3):197–210, 1978.

[121] J. H. Lim and J. C. Ye. Geometric gan, 2017.

[122] S.-C. Lo, M. Freedman, S. Mun, and S. Gu. Transformationally identical and invariant convolutional neural networks through symmetric element operators [https://arxiv.org/abs/1806.03636], 06 2018.

[123] S.-C. Lo, H. Li, J.-S. Lin, A. Hasegawa, C. Wu, M. Freedman, and S. Mun. Artificial convolution neural network with wavelet kernels for disease pattern recognition. *Proceedings of SPIE - The International Society for Optical Engineering*, 02 1995.

[124] S.-C. B. Lo, H.-P. Chan, J.-S. Lin, H. Li, M. T. Freedman, and S. K. Mun. Artificial convolution neural network for medical image pattern recognition. *Neural Networks*, 8(7):1201 – 1214, 1995. Automatic Target Recognition.

[125] S.-C. B. Lo, H. Li, A. Hasegawa, Y. J. Wang, M. T. F. M.D., and S. K. Mun. Detection of mammographic masses using sector features with a multiple-circular-path neural network. In K. M. Hanson, editor, *Medical Imaging 1998: Image Processing*, volume 3338, pages 1205 – 1214. International Society for Optics and Photonics, SPIE, 1998.

[126] Y.-C. Lo, S. E. Rensi, W. Torng, and R. B. Altman. Machine learning in chemoinformatics and drug discovery. *Drug discovery today*, 23(8):1538–1546, 2018.

[127] M. Lucic, K. Kurach, M. Michalski, O. Bousquet, and S. Gelly. Are gans created equal? a large-scale study. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 698–707, Red Hook, NY, USA, 2018. Curran Associates Inc.

[128] K. D. G. Maduranga, K. E. Helfrich, and Q. Ye. Complex unitary recurrent neural networks using scaled cayley transform, 2018.

[129] K. D. G. Maduranga, V. Zadorozhnyy, and Q. Ye. Symmetry-structured convolutional neural networks. *Neural Computing and Applications*, 35:4421 – 4434, 2022.

[130] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2813–2821, 2017.

[131] D. Marcos, M. Volpi, and D. Tuia. Learning rotation invariant convolutional filters for texture classification. *CoRR*, abs/1604.06720, 2016.

[132] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993.

[133] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[134] T. Martin et al. User's guide for test (version 4.2)(toxicity estimation software tool) a program to estimate toxicity from molecular structure. *Washington (USA): US-EPA*, 505, 2016.

[135] T. M. Martin and D. M. Young. Prediction of the acute toxicity (96-h lc50) of organic compounds to the fathead minnow (pimephales promelas) using a group contribution method. *Chemical Research in Toxicology*, 14(10):1378–1385, 2001.

[136] S. Merity, N. S. Keskar, and R. Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*, 2018.

[137] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models, 2016.

[138] L. Mescheder, S. Nowozin, and A. Geiger. The numerics of gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1825–1835. Curran Associates, Inc., 2017.

[139] L. Mescheder, S. Nowozin, and A. Geiger. Which training methods for gans do actually converge? In *International Conference on Machine Learning (ICML)*, 2018.

[140] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017*, 2017.

[141] Z. Mhammedi, A. D. Hellicar, A. Rahman, and J. Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *Proceedings of ICML 2017*, volume 70, pages 2401–2409. PMLR, 2017.

[142] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

[143] T. Miyato and M. Koyama. cGANs with projection discriminator. In *International Conference on Learning Representations*, 2018.

[144] D. L. Mobley and J. P. Guthrie. Freesolv: a database of experimental and calculated hydration free energies, with input files. *Journal of computer-aided molecular design*, 28(7):711–720, 2014.

[145] E. Mucllari, V. Zadorozhnyy, C. Pospisil, D. Nguyen, and Q. Ye. Orthogonal gated recurrent unit with neumann-cayley transformation, 2022.

[146] V. Nagarajan and J. Z. Kolter. Gradient descent gan optimization is locally stable. In *Advances in neural information processing systems*, pages 5585–5595, 2017.

[147] D. D. Nguyen, Z. Cang, and G.-W. Wei. A review of mathematical representations of biomolecular data. *Physical Chemistry Chemical Physics*, 22(8):4343–4367, 2020.

[148] D. D. Nguyen, Z. Cang, K. Wu, M. Wang, Y. Cao, and G.-W. Wei. Mathematical deep learning for pose and binding affinity prediction and ranking in d3r grand challenges. *Journal of computer-aided molecular design*, 33(1):71–82, 2019.

[149] D. D. Nguyen and G.-W. Wei. Dg-gl: Differential geometry-based geometric learning of molecular datasets. *International journal for numerical methods in biomedical engineering*, 35(3):e3179, 2019.

[150] D. D. Nguyen, T. Xiao, M. Wang, and G.-W. Wei. Rigidity strengthening: A mechanism for protein–ligand binding. *Journal of chemical information and modeling*, 57(7):1715–1721, 2017.

[151] T. D. Nguyen, T. Le, H. Vu, and D. Q. Phung. Dual discriminator generative adversarial nets. *CoRR*, abs/1709.03831, 2017.

[152] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

[153] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, and G. R. Hutchison. Open babel: An open chemical toolbox. *Journal of cheminformatics*, 3(1):1–14, 2011.

[154] H. J. Park, B. H. Kang, W. Shin, J. S. Kim, and S. W. Han. Manner: Multi-view attention network for noise erasure. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7842–7846. IEEE, 2022.

[155] S. Pascual, A. Bonafonte, and J. Serrà. Segan: Speech enhancement generative adversarial network, 2017.

[156] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855, jul 1992.

[157] C. Pospisil, V. Zadorozhnyy, and Q. Ye. Breaking time invariance: Assorted-time normalization for rnns, 2022.

[158] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[159] B. Ramsundar, B. Liu, Z. Wu, A. Verras, M. Tudor, R. P. Sheridan, and V. Pande. Is multitask deep learning practical for pharma? *Journal of Chemical Information and Modeling*, 57(8):2068–2076, 2017. PMID: 28692267.

[160] M. Ranzato, Y.-L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In *NIPS*, 2007.

[161] A. W. Rix, J. G. Beerends, M. Hollier, and A. P. Hekstra. Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, 2:749–752 vol.2, 2001.

[162] D. Rogers and M. Hahn. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010.

[163] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[164] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, 1986.

[165] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.

[166] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[167] M. V. Santana and F. P. Silva-Jr. De novo design and bioactivity prediction of sars-cov-2 main protease inhibitors using recurrent neural network-based transfer learning. *BMC chemistry*, 15(1):1–20, 2021.

[168] A. Sanyal, P. H. Torr, and P. K. Dokania. Stable rank normalization for improved generalization in neural networks and gans. *arXiv preprint arXiv:1906.04659*, 2019.

[169] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to nonlinear dynamics of learning in deep linear neural networks, 2014.

[170] R. E. Schapire. The boosting approach to machine learning: An overview. *Nonlinear estimation and classification*, pages 149–171, 2003.

[171] W. X. Shen, X. Zeng, F. Zhu, C. Qin, Y. Tan, Y. Y. Jiang, Y. Z. Chen, et al. Out-of-the-box deep learning prediction of pharmaceutical properties by broadly learned knowledge-based molecular representations. *Nature Machine Intelligence*, 3(4):334–343, 2021.

[172] Shih-Chung B Lo, Huai Li, Yue Wang, L. Kinnard, and M. T. Freedman. A multiple circular path convolution neural network system for detection of mammographic masses. *IEEE Transactions on Medical Imaging*, 21(2):150–158, 2002.

[173] K. Shmelkov, C. Schmid, and K. Alahari. How good is my gan?, 2018.

[174] N. Stiefl, I. A. Watson, K. Baumann, and A. Zaliani. Erg: 2d pharmacophore descriptions for scaffold hopping. *Journal of chemical information and modeling*, 46(1):208–220, 2006.

[175] V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston. Random forest: a classification and regression tool for compound classification and qsar modeling. *Journal of chemical information and computer sciences*, 43(6):1947–1958, 2003.

[176] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen. A short-time objective intelligibility measure for time-frequency weighted noisy speech. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4214–4217, 2010.

[177] H. D. Tagare. Notes on optimization on stiefel manifolds. *Yale*, 2011. "Technical report, Yale University".

[178] J. Tang and K. Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 565–573, New York, NY, USA, 2018. Association for Computing Machinery.

[179] J. Thiemann, N. Ito, and E. Vincent. The diverse environments multi-channel acoustic noise database (demand): A database of multichannel environmental noise recordings. *Proceedings of Meetings on Acoustics*, 19(1):035081, 2013.

[180] Y. Tian, Q. Wang, Z. Huang, W. Li, D. Dai, M. Yang, J. Wang, and O. Fink. Off-policy reinforcement learning for efficient and effective gan architecture search, 2020.

[181] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

[182] R. Todeschini and V. Consonni. *Handbook of molecular descriptors*. John Wiley & Sons, 2008.

[183] D. Toolkit. Daylight chemical information systems. *Inc.: Aliso Viejo, CA*, 2007.

[184] D. Tran, R. Ranganath, and D. M. Blei. Hierarchical implicit models and likelihood-free variational inference, 2017.

[185] D. N. Tran and K. Koishida. Single-channel speech enhancement by subspace affinity minimization. In *INTERSPEECH*, pages 2447–2451, 2020.

[186] N.-T. Tran, T.-A. Bui, and N.-M. Cheung. Dist-gan: An improved gan using distance constraints. In *ECCV*, 2018.

[187] N.-T. Tran, V.-H. Tran, B.-N. Nguyen, L. Yang, and N.-M. M. Cheung. Self-supervised gan: Analysis and improvement with multi-class minimax game. In *Advances in Neural Information Processing Systems 32*, pages 13253–13264. Curran Associates, Inc., 2019.

[188] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

[189] T. Unterthiner, B. Nessler, G. Klambauer, M. Heusel, H. Ramsauer, and S. Hochreiter. Coulomb gans: Provably optimal nash equilibria via potential fields. *CoRR*, abs/1708.08819, 2017.

[190] C. Valentini-Botinhao, X. Wang, S. Takaki, and J. Yamagishi. Investigating rnn-based speech enhancement methods for noise-robust text-to-speech. In *SSW*, 2016.

[191] H. Van De Waterbeemd and E. Gifford. Admet in silico modelling: towards prediction paradise? *Nature reviews Drug discovery*, 2(3):192–204, 2003.

[192] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[193] C. Veaux, J. Yamagishi, and S. King. The voice bank corpus: Design, collection and data analysis of a large regional accent speech database. In *2013 International Conference Oriental COCOSDA held jointly with 2013 Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE)*, pages 1–4, 2013.

[194] J. Verma, V. M. Khedkar, and E. C. Coutinho. 3d-qsar in drug design-a review. *Current topics in medicinal chemistry*, 10(1):95–115, 2010.

[195] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal. On orthogonality and learning recurrent networks with long term dependencies, 2017.

[196] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal. On orthogonality and learning recurrent networks with long term dependencies. *arXiv preprint arXiv:1702.00071*, 2017.

[197] S. Wang, S. Sun, Z. Li, R. Zhang, and J. Xu. Accurate de novo prediction of protein contact map by ultra-deep learning model. *PLOS Computational Biology*, 13(1):1–34, 01 2017.

[198] W. Wang, Y. Sun, and S. Halgamuge. Improving MMD-GAN training with repulsive loss function. In *ICLR*, 2019.

[199] D. Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988.

[200] J. Wenzel, H. Matter, and F. Schmidt. Predictive multitask deep neural network models for adme-tox properties: Learning from large data sets. *Journal of Chemical Information and Modeling*, 59(3):1253–1268, 2019.

[201] D. Willmott. *Recurrent neural networks and their applications to RNA secondary structure inference.* PhD thesis, University of Kentucky, 2018.

[202] R. Winter, F. Montanari, F. Noé, and D.-A. Clevert. Learning continuous and data-driven molecular descriptors by translating equivalent chemical representations. *Chemical science*, 10(6):1692–1701, 2019.

[203] S. Wisdom, J. R. Hershey, K. W. Wilson, J. Thorpe, M. Chinen, B. Patton, and R. A. Saurous. Differentiable consistency constraints for improved deep speech enhancement. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 900–904, 2019.

[204] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems 29*, pages 4880–4888. Curran Associates, Inc., 2016.

[205] K. Wu, Z. Zhao, R. Wang, and G.-W. Wei. Topp–s: Persistent homology-based multi-task deep neural networks for simultaneous predictions of partition coefficient and aqueous solubility. *Journal of computational chemistry*, 39(20):1444–1454, 2018.

[206] Y. Wu and K. He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[207] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[208] J. Xu, X. Sun, Z. Zhang, G. Zhao, and J. Lin. Understanding and improving layer normalization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[209] Z. Xu, S. Wang, F. Zhu, and J. Huang. Seq2seq fingerprint: An unsupervised deep molecular embedding for drug discovery. In *Proceedings of the 8th ACM international conference on bioinformatics, computational biology, and health informatics*, pages 285–294, 2017.

[210] A. Yan, S. Cheng, W.-C. Kang, M. Wan, and J. McAuley. Cosrec: 2d convolutional neural networks for sequential recommendation, 2019.

[211] K. Yang, K. Swanson, W. Jin, C. Coley, P. Eiden, H. Gao, A. Guzman-Perez, T. Hopper, B. Kelley, M. Mathea, et al. Analyzing learned molecular representations for property prediction. *Journal of chemical information and modeling*, 59(8):3370–3388, 2019.

[212] K. Yang, K. Swanson, W. Jin, C. W. Coley, P. Eiden, H. Gao, A. Guzman-Perez, T. Hopper, B. Kelley, M. Mathea, A. Palmer, V. Settels, T. Jaakkola,

K. F. Jensen, and R. Barzilay. Are learned molecular representations ready for prime time? *ArXiv*, abs/1904.01561, 2019.

[213] Z. Ye, Y. Yang, X. Li, D. Cao, and D. Ouyang. An integrated transfer learning and multitask learning approach for pharmacokinetic parameter prediction. *Molecular Pharmaceutics*, 16(2):533–541, 2019.

[214] G. Yu, A. Li, C. Zheng, Y. Guo, Y. Wang, and H. Wang. Dual-branch attention-in-attention transformer for single-channel speech enhancement. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7847–7851, 2022.

[215] V. Zadorozhnyy, Q. Cheng, and Q. Ye. Adaptive weighted discriminator for training generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4781–4790, June 2021.

[216] H. Zhang, Z. Zhang, A. Odena, and H. Lee. Consistency regularization for generative adversarial networks. In *International Conference on Learning Representations*, 2020.

[217] Y. Zhao, C. Li, P. Yu, J. Gao, and C. Chen. Feature quantization improves GAN training. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11376–11386. PMLR, 13–18 Jul 2020.

[218] H. Zhu, A. Tropsha, D. Fourches, A. Varnek, E. Papa, P. Gramatica, T. Oberg, P. Dao, A. Cherkasov, and I. V. Tetko. Combinatorial qsar modeling of chemical toxicants tested against tetrahymena pyriformis. *Journal of chemical information and modeling*, 48(4):766–784, 2008.

Vita

Vasily I Zadorozhnyy

Education:

- University of Kentucky, Lexington, KY
  Ph.D. in Mathematics, May 2023 (expected)
  Advisor: Dr. Qiang Ye
  Dissertation Title: Novel Architectures and Optimization Algorithms for Training Neural Networks and Applications
- University of Kentucky, Lexington, KY
  M.S. in Mathematics, December 2019
- Grand Valley State University, Allendale, MI
  B.S. in Mathematics, May 2017
  cum laude
- Lake Michigan College, Benton Harbor, MI
  A.S. in Mathematics, May 2015
  high honors

Professional Positions:

- Research Contractor, Microsoft, Summer 2022–Spring 2023
- Graduate Research Assistant, University of Kentucky, Summer 2019–Spring 2023
- Graduate Teaching Assistant, University of Kentucky, Fall 2017–Fall 2022
- Ph.D. Research Intern, Microsoft, Summer 2022
- Peer Coach for Numerical Analysis Preliminary Exam, University of Kentucky, Summer 2020–Fall 2021

Honors

- Spring 2023 Research Assistantship
- SIAM Certification for outstanding efforts and accomplishments of the University of Kentucky SIAM Chapter
- Max Steckler Fellowship Recipient

Publications:

- Edison Mucllari*, **Vasily Zadorozhnyy***, Duc Nguyen, Qiang Ye. Novel Molecular Representations using Neumann-Cayley Orthogonal Gated Recurrent Unit. Journal of Chemical Information and Modeling Article; DOI: 10.1021/acs.jcim.2c01526
- Kehelwala Dewage Gayan Maduranga*, **Vasily Zadorozhnyy***, Qiang Ye. Symmetry Structured Convolutional Neural Networks. Neural Computing and Applications (NCAA), 35:4421 – 4434, 2022 (* - equal contribution)
- Darren B. Parker and **Vasily Zadorozhnyy**. A group labeling version of the lights-out game. Involve, a Journal of Mathematics 14-4 (2021), 541–554. DOI 10.2140/involve.2021.14.541
- **Vasily Zadorozhnyy**, Qiang Cheng, Qiang Ye. Adaptive Weighted Discriminator for Training Generative Adversarial Networks. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 4781-4790
- **Zadorozhnyy, V. I.**, Zadorozhnyy, I. V., Kuzurman, V. A. Calculation of the optimal parameters for technological process of penocarbides manufacturing. XXVIII International Scientific Conference: Mathematical Methods in Engineering and Technology MMTT-28 (translated from Russian). 8(1), 204-206, 2015, ISBN: 9785743323869
- **Zadorozhnyy, V. I.**, Kuzurman, V. A. Mathematical planning and optimization of technological process parameters for manufacturing porous carbide materials (translated from Russian). Youth Forum: Technical and Mathematical Science, 3(7): 382-386, 2015, doi: 10.12737/14884
- **Zadorozhnyy, V. I.**, Kuzurman, V. A., Mamanova, R. G. Assessment of nitrate content in food products of plant origin and methods for their reduction (translated from Russian). Materials of the Scientific and Practical Conference in the framework of the Days of Science of Students and Postgraduates of Vladimir State University named after Alexander Grigoryevich and Nikolai Grigoryevich Stoletov (translated from Russian). 1.(1): 397—399, 2012, ISBN: 9785998402548

Preprints:

- **Vasily Zadorozhnyy**, Qiang Ye, and Kazuhito Koishida. SCP-GAN: Self-Correcting Discriminator Optimization for Training Consistency Preserving Metric GAN on Speech Enhancement Tasks. arxiv.org/abs/2210.14474, 2022. Submitted to the 24th INTERSPEECH Conference (INTERSPEECH 2023), 2023

- **Vasily Zadorozhnyy***, Edison Mucllari*, Cole Pospisil, Duc Nguyen, Qiang Ye. Orthogonal Gated Recurrent Unit with Neumann-Cayley Transformation. arxiv.org/abs/2208.06496, 2022. Submitted to the Neural Computation Journal, 2023 (* - equal contribution)
- Cole Pospisil*, **Vasily Zadorozhnyy***, Qiang Ye. Breaking Time Invariance: Assorted-Time Normalization for RNNs. arxiv.org/abs/2209.14439, 2022. Under the review in the Springer Journal of Applied Intelligence, 2023. (* - equal contribution)