Parameter Continuation with Secant Approximation for Deep Neural Networks

by

Harsh Nilesh Pathak

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Master of Science

in

Data Science

in the

Graduate Division

of the

Worcester Polytechnic Institute, Worcester

Fall 2018

Committee in charge:

Professor Randy Paffenroth, Advisor

Professor Kyumin Lee, Reader

Abstract

Parameter Continuation with Secant Approximation for Deep Neural Networks

by

Harsh Nilesh Pathak

Master of Science in Data Science

Worcester Polytechnic Institute, Worcester

Professor Randy Paffenroth, Advisor

Non-convex optimization of deep neural networks is a well-researched problem. We present a novel application of continuation methods for deep learning optimization that can potentially arrive at a better solution. In our method, we first decompose the original optimization problem into a sequence of problems using a homotopy method. To achieve this in neural networks, we derive the Continuation(C)-Activation function. First, C-Activation is a homotopic formulation of existing activation functions such as Sigmoid, ReLU or Tanh. Second, we apply a method which is standard in the parameter continuation domain, but to the best of our knowledge, novel to the deep learning domain. In particular, we use Natural Parameter Continuation with Secant approximation(NPCS), an effective training strategy that may find a superior local minimum for a non-convex optimization problem. Additionally, we extend our work on Step-up GANs, a data continuation approach, by deriving a method called Continuous(C)-SMOTE which is an extension of standard oversampling algorithms. We demonstrate the improvements made by our methods and establish a categorization of recent work done on continuation methods in the context of deep learning.

Contents

List of Figures

List of Symbols

The next list describes several symbols that will be later used within the body of the document

λ	homotopy parameter
θ	set of parameters of a neural network
C	data embedding code
$f(X)$	encoder function
$g(C)$	decoder function
$J(\theta)$	cost or loss function
$X \in R^{m \times n}$	input data
ϕ	Activation function
D_{JS}	Jensen–Shannon(JS) divergence
D_{KL}	Kullback–Leibler(KL) divergence
p_θ	Generated data distribution
p_z	Noise distribution

p_{data}	True data distribution

Mathematical Models

$AE\text{-}ADAM_{activation}$	Autoencoder with specified activation function and ADAM optimizer
$AE\text{-}NPCS_{activation}$	Autoencoder with specified activation function and optimized using NPCS method
$WGAN_{BN}$	Wasserstein GAN with Batchnormalization
$WGAN_{vanilla}$	Wasserstein GAN with no normalization
AE	Autoencoder
GAN	Generative Adversarial Network
NPC	Natural Parameter Continuation
NPCS	Natural Parameter Continuation with Secant approximation
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
SVM	Support Vector Machine

Acknowledgments

I would like to express my sincere gratitude to my advisor Prof. Randy Paffenroth for his guidelines throughout my research thesis. Also, I would like to thank Xiaozhou Zou, whose collaboration was helpful for me to take correct initial steps and thus boost the research work for this thesis.

Chapter 1

Introduction

Deep learning is an example of representation learning methods that are widely used to transform raw input measurements into one or more abstract levels. Such neural networks provide the flexibility of adding multiple levels of abstraction, through which we can represent many complex transformations. These methods have significantly advanced the state-of-the-art in speech recognition [24], image recognition [33] and text mining [37] for application in domains such as health-care, travel and security [34]. Deep learning has achieved many state-of-the-art results in supervised tasks such as classification and regression, and such methods have also been applied to unsupervised learning such as dimensionality reduction. However, training the neural network to find an optimal solution is a challenging optimization task [8], [19]. Using current techniques, machines configured with large number of CPUs, GPUs and high memory, spend days or even weeks to solve deep learning optimization problems.

The objective of a deep learning model is no different from any other machine learning model. They try to approximate a function f_{true} that is the true function that maps the raw input x to the output labels or targets, y such that $y = f_{true}(x)$ [20]. For example, perhaps f_{true} is a camera taking a picture, or some other physical measurement process. The key difference between deep learning models and other types of machine learning models is that deep learning models use deep composition of functions as represented by a neural network.

Precisely, a deep neural network forms a chain of functions (layers), that defines a mapping $y = f(x;\theta)$, where θ is the set of parameters. These parameters θ are estimated by minimizing an objective (cost) function, such that the mapping f best approximates f_{true}.

Stochastic Gradient Descent(SGD) [20] is a popular choice for minimizing such functions. A few random samples, called a mini-batch, from the input matrix are provided to the deep neural network, after which the average gradients are being computed over those samples and then using backpropagation to adjust the model parameters accordingly. This allows convergence even when the number of training dataset samples is large. We have seen many improvements in SGD that are widely applied by the deep learning community such as RMSprop [25], AdaGrad [15] and Adam [31]. All of these methods work fairly well for many tasks. However, their successful implementation is highly dependent on the *quality of the initial guess*. Moreover, there are many theoretical guarantees that show that the optimizer will always converge to a local-minimum, but convergence to a global minimum is difficult to guarantee.

Loss surfaces are poorly understood [11]. Even a simple feed-forward network can have exponentially many local minima [6],[45]. Previously, researchers have shown that the different optimization and initialization methods can lead to a dramatically different geometry of the solution curve on a loss surface [19],[26],[11]. In other words, a slight variation in initialization may lead the model to converge to a very distinct local minimum. This indicates, that some random initialization may lead to a solution that is very far away from the true solution. Thus, aforementioned non-convex optimization [19],[20] problem requires some rethinking toward its solution. In this thesis, we are interested in reporting and solving similar optimization problems.

Considering non-convex optimization is a challenging task which arises with almost every deep learning model, a well-known potential alternative to SGD for such complex models is a parameter continuation method. Continuation methods can be utilized to organize the training and assist in improving the *quality of an initialization* that may accelerate the convergence of the non-convex optimization problem. Parameter continuation is the prime

focus of our thesis. The fundamental idea is to start from an easier version of the problem and gradually transform it into the original version. During this process one transforms the global minimum (or a superior local minimum [1]) of the easier problem into a minimum of the harder problem. This transformation reveals some unusual mechanics of the deep learning optimization task. *In particular, almost all of the state-of-the-art techniques, of which we are aware, work on a fixed loss surface. However, we transform the loss surface continuously to design an effective training strategy.*

Further in this thesis, we briefly explain our goals and contributions. In Chapter 2, we provide the necessary background for our readers. Then, in chapter 3, we study and discuss other works that are close to our work for deep learning, such as Curriculum learning [8], Mollifying networks [21] and training deep learning by diffusion methods [38]. In later chapters, we discuss our methods 4, experiments and results 5.

In this thesis, we provide background on continuation methods, especially on Natural Parameter Continuation [1]. We attempt to enlighten a few topics, which we believe are not very well discussed in the literature. First, the association of continuation method with deep learning is not well known. Second, we discuss the importance of initialization strategy for stable and faster convergence. These two concerns are frequently encountered in the literature when one attempts to solve non-convex optimization problems. In this thesis, we propose a novel continuation training strategy that leverages the advantage of continuation method along with the standard training procedure of deep neural networks. The novel contributions of this thesis are outlined in the following section.

[1] superior local minima means a local minimum, which is near to the true solution of the objective function

1.1 Contributions

In this thesis, we did a literature survey and extracted some insightful relationships between various work performed on continuation methods for the deep learning optimization problem. We characterize the continuation methods into model continuation and data continuation. To facilitate model continuation in our network, we establish a novel activation function **Continuation(C)-Activation**, that yields a continuous deformation from the linear to the non-linear network.

We present a novel continuation training strategy (or optimization method) for neural networks. Following the principles of Natural Parameter Continuation(NPC), our method develops the idea of using a **Secant** approximation leading to our novel Natural Parameter Continuation using Secant (NPCS) method. NPCS is able to work jointly with Stochastic Gradient Descent(SGD) and its variants, but at each stage the NPCS method provides a SGD solver with a good initial guess. Next, we compare the convergence of NPCS with a well-used optimization technique - Adam. Further, instead of randomly initializing the neural networks, we use Principal Component Analysis(PCA), which usually provides a stable initialization and a robust solution for the NPCS method. Next, we show results for PCA initialization that significantly reduces the number of training steps and we designed an Autoencoder(AE) to demonstrate the results of our optimization method.

Finally, we present **Step-Up GANs**, a data continuation technique derived in previous work [2]. Furthermore, we analyze its shortcomings and overcome them with our pre-processing algorithm **Continuous(C)-SMOTE** – a novel oversampling technique. C-SMOTE can be applied to a variety of datasets such as a synthetic Multi-Gaussian Grid, Sine-Wave and a real-world dataset MNIST. Association of C-SMOTE can condition the training of neural networks with some specific loss functions such as KL-divergence and JS-divergence.

[2] Xiaozhou Zou's Master's thesis was submitted in April 2018.

Chapter 2

Background

2.1 Deep Feedforward Networks

We understand linear models from classic machine learning such as linear regression and logistic regression. Linear models are shown to have convergence guarantees, because they satisfy all the constraints of convex optimization. However, these models are limited to extract only the linear properties of the data. To overcome this limitation, various methods such as kernel trick, RBF, manual engineering and neural networks are developed [20].

Deep Feedforward Networks are modeled with an aim to approximate some function f_{true}. For example, for a classifier, $\mathbf{y} = f_{true}(\mathbf{x})$, f_{true} maps input $\mathbf{x}$ to category $\mathbf{y}$. Here, $\mathbf{x}$ is an input row or feature set, from the input data matrix X. A feedforward network introduces a mapping $\mathbf{y} = f(\mathbf{x};\theta)$, and then learns the value of the parameters θ that result in the approximation of the function f_{true} [20]. Feedforward network is a directed acyclic graph representing the composition of functions. $f(\mathbf{x}) = (f^2(f^1(\mathbf{x})))$ defines a simple feedforward network that has two functions, f^1, is the hidden layer and f^2, is the output layer. In order to make the network deep, we may add several functions in a chain, where the length of the chain determines the depth of the network. In addition to that, we may vary the width (dimensionality) of the network at various hidden layers.

The hidden layers enable to learn nonlinear properties of the input X. However, just stacking many functions won't help us to achieve non-linear behavior from the data. Thus, apart from varying depth and width, we also need to choose a suitable activation function. Some of the popular choices are:

1. Logistic sigmoid function (Sigmoid)

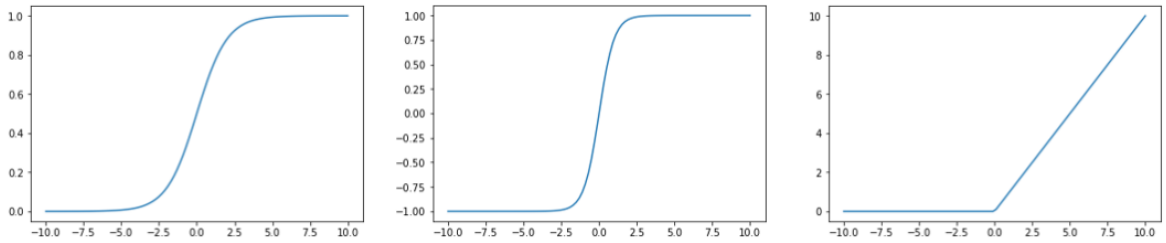
$$\phi_{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

2. Hyperbolic tangent funciton (Tanh)

$$\phi_{tanh}(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.2}$$

3. Rectified linear fucntion (ReLU)

$$\phi_{relu}(x) = max(0, x) \tag{2.3}$$



Sigmoid Tanh ReLU

Figure 2.1: Activation functions

Learning feedforward network consists of specifying an optimization scheme, a cost function, and network parameters. One of the biggest drawbacks of training a neural network is the cost function, which is usually convex with most machine learning algorithms, now becomes non-convex[20]. Usually, a parametric model defines a distribution $p(\mathbf{y}|\mathbf{x}; \theta)$, and

similar to machine learning, we apply the principle of maximum likelihood to estimate the parameters. Neural networks usually learn from the iterative gradient descent. Specifically, SGD is applied to the non-convex cost function along with backpropagation to the parameters of network based on the errors we make [22]. After several backpropagation steps, model distribution is usually a good resemblance of the data distribution.

2.2 Autoencoders

Autoencoder (AE) is a neural network that aims to learn an identity function, given an input to the output in unsupervised fashion [20]. AE has two main components, namely an **encoder** $C = f(X)$, and a **decoder** $X' = g(C)$, where C represents **code** and X' is the reconstruction of X. Code C is the feature representation which is usually of lower dimension.

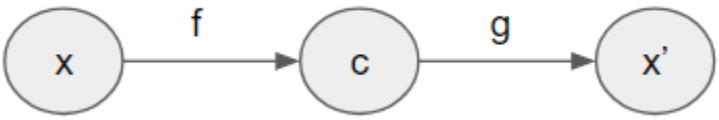


Figure 2.2: Autoencoder

In the above figure, we show input data matrix X, an encoder function f that transforms the data to a code layer C. Furthermore, representation of C is mapped back to X' using a decoder function g.

$$J(\mathbf{x}, g(f(\mathbf{x})); \theta)^1 \tag{2.4}$$

In equation 2.4, we show the objective function of an AE, where J is the objective function. One of the most popular choice is mean squared error $\|g_\theta(f_\theta(\mathbf{x})) - \mathbf{x}\|_F^2$, which is non-convex as with deep autoencoders [8],[45].

[1] $J(\theta)$ short hand notation

Theoretically, there is no constraint on the dimension of the code layer C. Instead of learning key properties from the data, AE can potentially memorize it, when C has a much higher dimension than data [45]. When the dimension of code layer is less than the dimension of the data, then AE is popularly known as Undercomplete AE [20]. It captures the most essential properties of the data, and in this thesis, we are more interested in optimizing Undercomplete AEs.

AE determines the feature representations of data on a linear or a non-linear manifold. These are thus widely used for dimensionality reduction, unsupervised pretraining [20] or feature learning. Recently, AE gained significant attention in generative modeling [30]. However, designing a robust deep autoencoder is a challenging task when dealing with high dimensional datasets [54].

2.2.1 Principal Component Analysis (PCA) and Autoencoders

The curse of dimensionality is a common issue and frequently encountered with high dimensional data; thus analyzing the data in a low dimensional manifold is usually preferred [14],[48]. PCA, a widely used unsupervised algorithm can provide a lower-dimensional linear manifold (subspace) of the data. However, the classic linear manifold is usually not preferred when the data resides in some nonlinear manifold [54] as shown in figure 2.4. On the other hand, AEs are equipped with non-linear activation functions such as Sigmoid, ReLU etc. Thus, AE can be viewed as a non-linear generalization of PCA [54]. In the absence of activation functions, autoencoders learn almost the same linear manifold as PCA, given the loss function, is mean squared error [20]. In this thesis, we study both the methods, understand their similarities and show how autoencoders can benefit from PCA in section 4.

2.2.2 PCA and SVD

PCA calculates the correlation matrix of the data ($X \in R^{m \times n}$) as $X^T X$, to compute the eigen-decomposition of the matrix. To overcome this heavy computation, a numerical

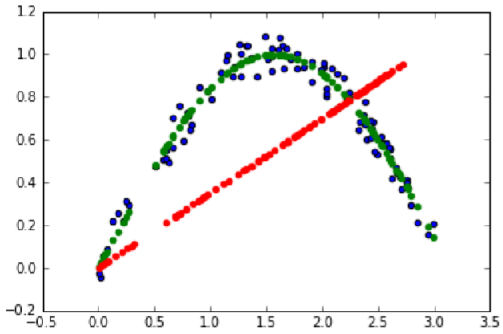


Figure 2.3: Manifold Learning [54]

Points in blue are true data points that clearly lies on a non-linear manifold. The green points show an optimized non-linear projection of the data. The red points show the linear manifold which is projected by PCA. Clearly, PCA fails to learn the non-linear manifold of the data.

method, Singular value decomposition (SVD) has a more efficient approach to calculate the same PCA projections [41].

Given the data matrix X, its singular value decomposition is given by

$$X = USV^T \tag{2.5}$$

where U and V are the unitary matrices, and singular values of X are present in the diagonal matrix S. Rank of the data matrix X is the number of non-zero diagonal entries in S. Next, the rows of V^T are the eigen-vectors of X^TX and columns of U contain eigen-vectors of XX^T and principal components can be calculated by US. We show how SVD can be utilized as an initialization strategy for Autoencoders in chapter 4.

2.3 Non-convex Optimization problem

An optimization problem can be formulated as

$$\begin{aligned} &\min_{\theta \in \mathbb{R}} \; J(\theta) \\ &s.t. \;\; \theta \in \mathcal{C} \end{aligned} \tag{2.6}$$

where θ is a parameter or set of variables to be estimated in the problem, $J : \mathbb{R}^p \rightarrow \mathbb{R}$ is the objective function of the problem, and $\mathcal{C} \subseteq \mathbb{R}^p$ is constraint set of parameters.

There are two conditions for a problem to be convex, i.e. objective function and a constraint set [2] of this problem, both of which are convex [27]. Any optimization problem that fails to meet either one of these conditions is called a **non-convex** optimization problem [27].

A non-convex optimization problem is shown to be NP-hard to solve [36]. Such problems are frequently observed in many applications of machine and deep learning such as recommendation systems, signal processing, and bioinformatics. Popular techniques that are often used for a non-convex problem are gradient descent, alternating minimization, expectation maximization, etc.

2.4 Stochastic Gradient Descent (SGD)

In the previous section, we saw the non-convex optimization problem and methods that are commonly practiced to find their solutions. Especially with deep learning, Stochastic Gradient Descent (SGD) and its variants are generally used to find a superior local minimum. SGD algorithm for an AE is shown here 1. SGD is highly sensitive to its learning rate ϵ and many improved variants of SGD show plausible ways to converge to a superior local minimum with almost no theoretical guarantees.

Usually in convex optimization problems, the local minimum must be a global minimum, but for non-convex problems, generally, tracing the global minimum is a difficult task because the cost surfaces are composed of multiple local minima and saddle points [39], [40]. For any non-convex cost function $J(\theta)$, if $J'(\theta) = 0$ [3],it is called a critical point [20] which can be characterized as local minima, local maxima and saddle points. A local minimum is a point where $J(\theta)$ is lower than all neighboring points, but still may be farther from the global

[2] Constraint Convex Set: A set $\mathcal{C} \in \mathbb{R}^p$ is considered convex if, for every $\theta_1, \theta_2 \in C$ and $\lambda \in [0, 1]$, we have $(1 - \lambda) \cdot \theta_1 + \lambda \cdot \theta_2 \in \mathcal{C}$ as well [27]

[3] $J'(\theta)$ denotes first derivative of $J(\theta)$

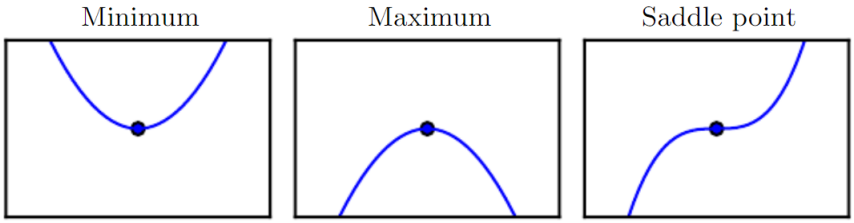


Figure 2.4: Critical Points [20]

In the above figure, we show various critical points in one dimension. From left to right, local minimum, which is lower than the neighboring points; a local maximum, which is higher than the neighboring points; and lastly, a saddle point, which has neighbors that are both lower and higher than the point itself.

minimum or any other superior local minimum. If $J(\theta)$ is a local highest point, then it is referred to as local maximum. Saddle points are special points that are both local minimum and maximum at the same point w.r.t different cross-sections. The simplest neural network is a perceptron or more traditionally a simple logistic system which is convex but as we go deeper i.e. increase the number of layers, the optimization problem becomes more and more non-convex [53], [5] i.e increasing number of local minima and saddle points.

The problem of avoiding or escaping saddle points and bad local minima is a difficult task in itself. Many configurations of saddle points can appear in high dimensional problems [27]. It should be noted that there exist saddle configurations, bypassing which is intractable in itself. For such cases, even finding locally optimal solutions is a NP-hard problem [2]. In this thesis, we mostly use Adam, an advanced and widely accepted variant of SGD optimizer. Adam is a combination of RMSProp [25] and momentum [20]. Since Adam is well evaluated and used in many research work, we thought it would be a good selection to compare our technique.

Algorithm 1 Stochastic Gradient Descent (SGD) for AE

Require: Learning rate schedule ϵ_1, ϵ_2....
Require: Initial parameter θ
1: $k \leftarrow 1$
2: **while** stopping criteria not met **do**
3: Sample a minibatch from data x^i
4: Compute gradient estimate $\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_i J(x, g(f(x)); \theta)$
5: Apply gradient $\theta \leftarrow \theta \ - \epsilon \cdot \hat{g}$
6: $k \leftarrow k + 1$
end while

2.5 Generative Adversarial Networks

Generative Adversarial Network (GAN) is a unique class of neural networks that, when given a set of target images, can learn to generate new images that have a similar distribution, instead of just classifying or reconstructing it. GAN is composed of two components as shown in Fig 2.5, D: the discriminator which learns from the real data and guides G: the generator whose objective is to resemble the true data distribution.

The objective function of GANs [18],:

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{x \sim p_{data})}[logD(x; \theta_D)] + \mathbb{E}_{z \sim p_z}[1 - logD(G(z; \theta_G); \theta_D)] \tag{2.7}$$

here, θ_G is parameter vector of generator network and θ_D is the parameter vector of discriminator network.

$\mathbb{E}_{x \sim p_{data}}[logD(x; \theta_D)]$ - Expected log likelihood of the output of discriminator network when the input x is drawn from true data distribution p_{data}.

$\mathbb{E}_{z \sim p_z}[1 - logD(G(z; \theta_G))]$ - Expected log likelihood of the output of discriminator network classifying the generated data as fake.

Researchers have previously shown that when the discriminator is close to the convergence, the objective of generator network is equivalent to minimizing the Jensen–Shannon(JS) divergence between p_{data} and p_θ[4] [18], [17].

[4]here p_θ, denotes the probability distribution of the generated data

$$D_{JS}(p_{data}\|p_\theta) = \frac{1}{2}D_{KL}(p_{data}\|\frac{p_{data}+p_\theta}{2}) + \frac{1}{2}D_{KL}(p_\theta\|\frac{p_{data}+p_\theta}{2}) \tag{2.8}$$

where D_{KL} is given by:

$$D_{KL}(p_{data}\|p_\theta) = \sum^{i} p_{data}(i)\log\frac{p_{data}(i)}{p_\theta(i)} \tag{2.9}$$

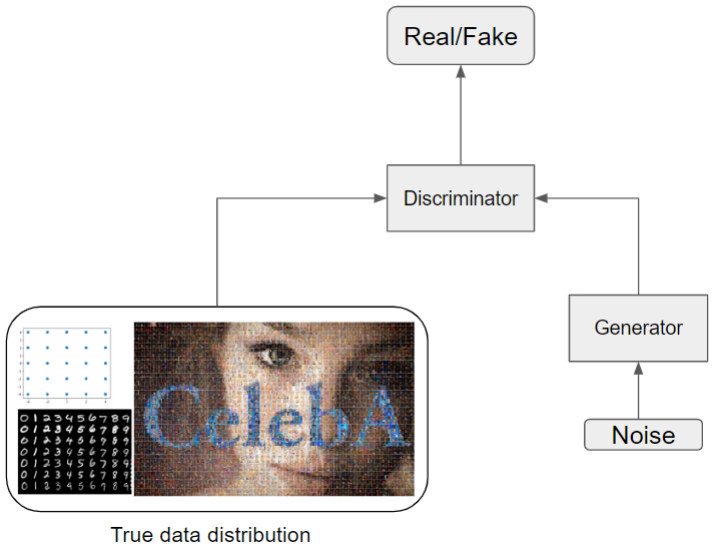


Figure 2.5: GAN Work-flow

In the above figure, we show the primary flow of a Generative Adversarial Network(GANs). Discriminator takes input from the real data such as CelebA [35] or MNIST [42] and also from generator transformed noise. Then it learns to differentiate real and fake data. On the other hand, the generator learns to fool the discriminator or transform the noise distribution to become as similar as possible to the real data distribution.

JS divergence is a well-known metric for measuring the distances between two probability distributions. Both the p_{data} and p_{theta} lie in the lower manifold with high probability as shown in paper [3] with proof that they are almost disjoint. JS divergence is not continuous in such situation and thus, optimizing through gradient descent won't work. Wasserstein paper [4] provides a nice example to illustrate the same. P and Q are the two probability distributions defined as 2.10 and explained with figure 2.6.

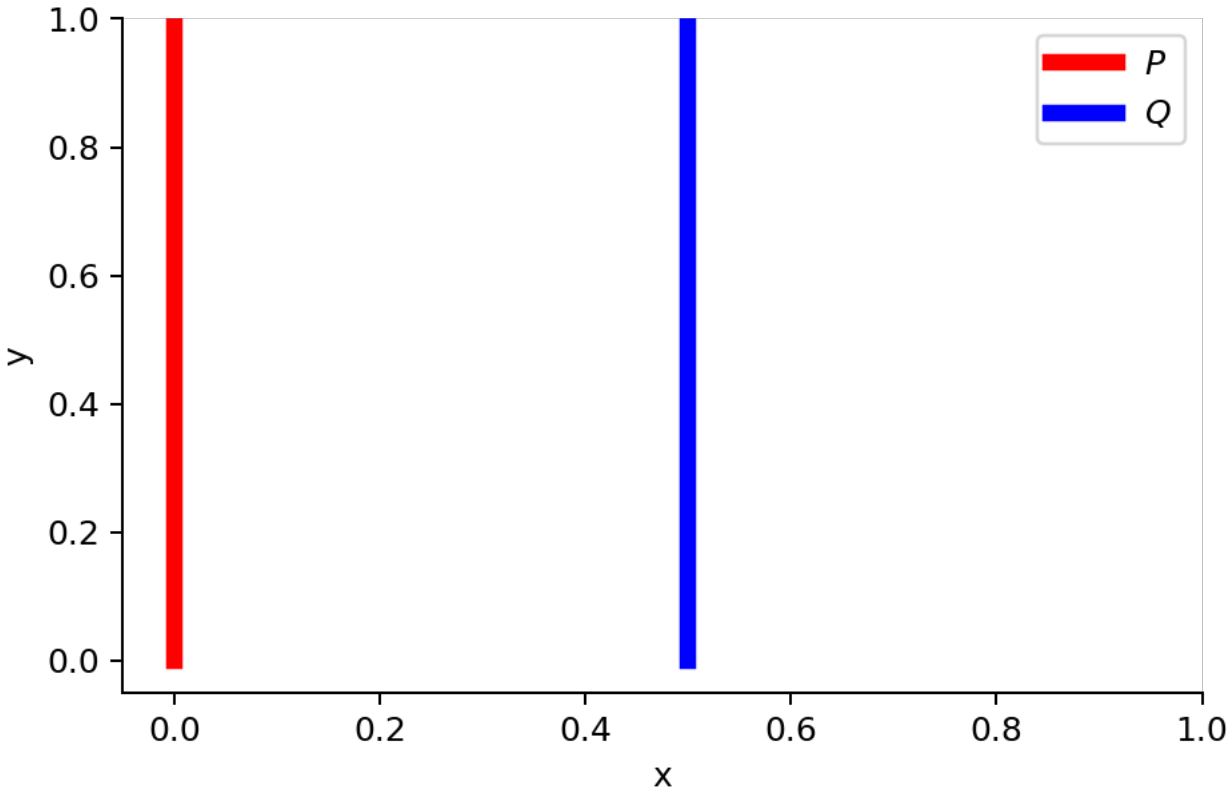


Figure 2.6: Distribution disjoint example [4], [50]

$$\begin{aligned} \forall(x,y) \in P, x = 0 \text{ and } y \sim U(0,1) \\ \forall(x,y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0,1) \end{aligned} \tag{2.10}$$

Following the above definitions for the two distributions, below is the result for different distribution metrics.

$$D_{KL}(P||Q) = \begin{cases} \infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{2.11}$$

$$D_{JS}(P||Q) = \begin{cases} \log 2 & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{2.12}$$

D_{KL} results in ∞ and D_{JS} is not differentiable at $\theta = 0$. Thus, when GAN encounters a disjoint support, we are always capable of finding a perfect discriminator that separates the real and fake samples 100% correctly, which in turn means an unstable GAN. There are other evaluation metrics such as Wasserstein distance (D_W) that is continuous even if there is no overlap between the support of the two distributions 2.13. Wasserstein distance is also referred to as Earthmover distance because it regards a probability distribution as a unit amount of earth piled on a given metric space. In other words, Wasserstein distance is the minimum cost of transforming a pile of earth (probability distribution P) to another pile of earth (probability distribution Q) [4]. It is defined in equation 2.14

$$D_W(P||Q) = |\theta| \tag{2.13}$$

$$D_W(p_{data}, p_\theta) = \inf_{\gamma \sim \Pi(p_{data}, p_\theta)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{2.14}$$

where $\Pi(p_{data}, p_\theta)$ is set of all joint distributions between p_{data} and p_θ. Specifically, $\gamma(x, y)$ tells the percentage of dirt that should be transported from x to y so as to make x follow the same distribution as y [50].

Even though Wasserstein GAN (with few assumptions) is continuous everywhere and differentiable almost everywhere [4], GANs are extremely difficult to train. Most of the real world datasets such as images, music, speech, text, or even finance are multimodal distributions and GANs are commonly seen to have the mode collapse problem [17],[4] with such datasets. In this thesis, we show adding noise to the data helps in faster convergence of WGAN with very few modal collapses. We describe our experiments with WGAN in section 3 and extending our own work to share a novel data continuation strategy that can be effectively used with many deep learning models depending on the loss function such as KL and JS divergence.

2.6 Continuation Methods

Continuation or homotopy methods [44] have long served as a useful technique in numerical methods, the fundamental idea of which has been a part of the literature since 1880's [1]. Continuation method may also be seen as an alternate approach to the gradient descent [40] for non-convex problems. To the best of our knowledge, continuation method for training a neural network was first presented here [12] and thereafter, the idea has been slowly adopted by a small group of researches that we discuss in detail in the next section 3. The main idea is to start from a simple function whose solution is comparatively easy to find, and gradually deform it towards the actual (complex) task. This deformation process occurs by slowly tracking the homotopy path. A homotopy function [1] is described as follows:

$$h(x, \lambda) = (1 - \lambda) \cdot h_1(x) + \lambda \cdot h_2(x) \tag{2.15}$$

where,

$\lambda \in [0, 1]$ is a homotopy parameter

$h_1(x)$ is a system simple function

$h_2(x)$ is a system complex function

After doing an extensive literature survey, we found few ways by which continuation methods can be introduced in a neural network. We present our method using the formulation of the above function in Chapter 4.

2.6.1 Parameter Continuation

Autoencoder is a mapping of data to its code and then back to its reconstruction. This clearly suggests that it is a system of non-linear equations and we can apply numerical continuation methods to compute their approximate solutions. Numerical continuation is a technique of computing approximate solutions of a system of parameterized non-linear equations [1]. The

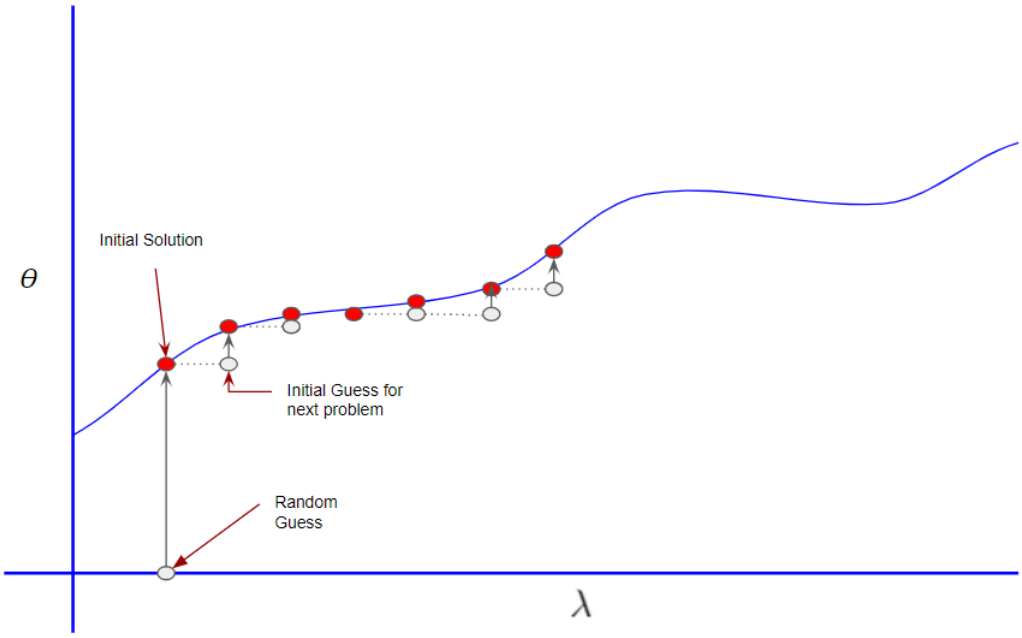


Figure 2.7: Natural Parameter Continuation

In this figure, we show the phenomenon of the natural continuation. We start with a random guess of parameters θ and use some optimizer to find the initial solution. Further, we increase the value of λ to solve the next problem, but this time we use solution to the previous problem as initial guess instead of random guess.

objective function in context of AE, in equation 2.4 becomes 2.16.

$$J(x, g(f(x); \theta, \lambda))^{5} \tag{2.16}$$

Here, the additional homotopy parameter λ is usually a real scalar $\lambda \in [0, 1]$, and the solution θ a n-vector. Natural parameter continuation is an adaptation of the iterative solver to a parameterized problem. The solution at one value of λ is used as the initial estimate for the solution at $\lambda + \Delta\lambda$. With $\Delta\lambda$ being sufficiently small, the iteration applied to the initial estimate should converge [1].

[5] $J(\theta; \lambda)$ short hand notation

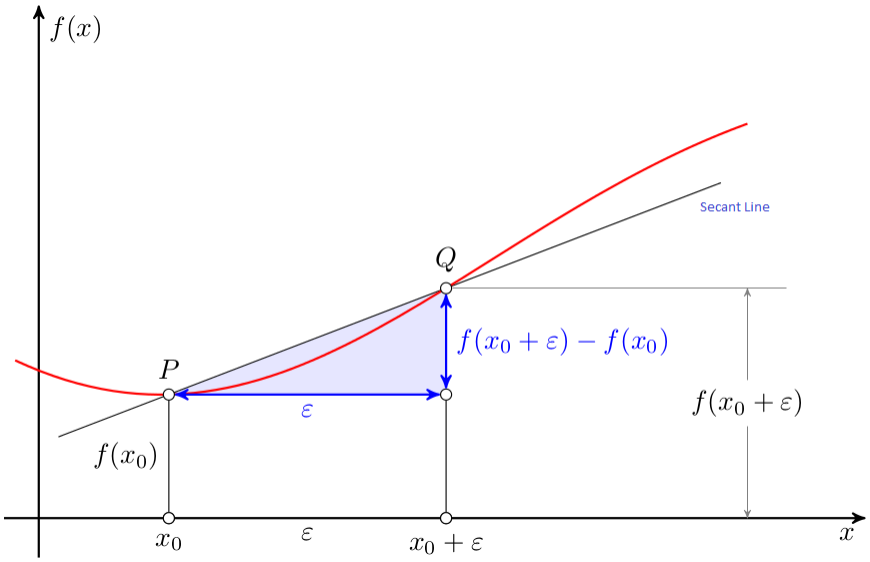


Figure 2.8: Secant Line

In the above figure, we show an example of secant line for a function $f(x)$. P and Q are the two distinct points through which the line intersects the curve.

2.7 Secant Line

A secant line[6] of a curve is a line that passes through at least two distinct points [28], as shown in figure 2.8. The slope of the secant line is given by equation 2.17.

$$slope_{secant} = \frac{f(x_0+\epsilon)-f(x_0)}{\epsilon}$$
$$slope_{tangent} = \lim_{\epsilon \longrightarrow 0} \frac{f(x_0+\epsilon)-f(x_0)}{\epsilon} \tag{2.17}$$

A secant line may be used to approximate the tangent line. A tangent line is a straight line that touches a curve at a single point. So, from the figure 2.8, as $\epsilon \rightarrow 0$, the secant line becomes more closer to tangent line. In this thesis, we use this secant line approximation to enhance Natural Parameter Continuation (NPC). We explain this technique in detail in section 4.

[6]By introducing secant line, we do not intend to use other secant methods which are popular in deep learning such as one step secant [13] or modification to Newton's method.

Chapter 3

Motivation and Related work

3.1 Motivation

One of the strong motivations for this thesis is an observation of convergence 3.1 of GANs and empirical claims on a Multi-Gaussian Grid 5.4. We did this research work in collaboration with Xiaozhou Zou [1]. For this experiment, we used Wasserstein GAN [4] and had a two-dimensional Gaussian grid 5.4 with 25 modes as our target distribution. The aim of this experiment was to generate data points similar to this target distribution.

Observation: For learning the target distribution, if at some intermediate training step, the generated data uniformly covers the support of target distribution, then it is very likely that GAN would converge to a good solution in later training steps. 3.1.

Considering the true distribution is very complex for a generator to learn. During the initial steps of our experiments, we observed that the generator transformed data points were far away from the target distributions, because of very less overlap between the true distribution p_{data} and initial model distribution p_θ. Even in literature, various experiments has been performed to efficiently find this overlap in initial stages of training. One approach

[1] Xiaozhou Zou's Master's thesis was submitted in April 2018. Please refer to this thesis for a complete explanation of the experiment settings. Here, we attempt to provide you with the chief idea used from previous research.

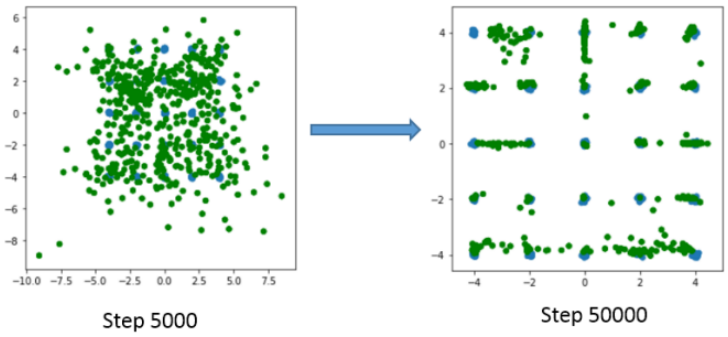


Figure 3.1: Observation

In the above figure, blue points are the target distribution and green points are generated from GANs. Here we want to show at global step 5000 the generated points were uniformly distributed over the support of the target distribution, and after that the convergence was stable and finally reached a good solution at global step 50000.

we discovered is to provide information about the underlying support of the distribution to the generator such that the task becomes simpler.

Consequently, we decomposed the standard training process into multiple phases. We designed a synthetic setting [2] that enabled us to decompose a difficult task (learning a complex distribution) to a series of simpler tasks. As shown in the figure 3.2, we first train the GAN_1 with data uniformly distributed over the target distribution. Next, GAN_2 is initialized with the parameters of previously trained GAN_1, however, in this phase, we add noise of standard deviation 0.3 to every mode of the target distribution. Therefore, this noisy data distribution is more complex than uniform distribution and simpler than target distribution 5.4. Finally, we train GAN_3, where generator is initialized from GAN_2 on the target distribution 5.4. We refer to our architecture as Step-Up GANs 3.2 because after every step we are one step closer to the true distribution to be learned by the generator. In our method, a generator inherits the knowledge learned by all its previous generators. Also note, the discriminator of each GAN is initialized from scratch and we applied Wasserstein distance as the cost function for all our experiments.

[2]Since we knew the modes of the data, we added some noise to the true distribution around that modes

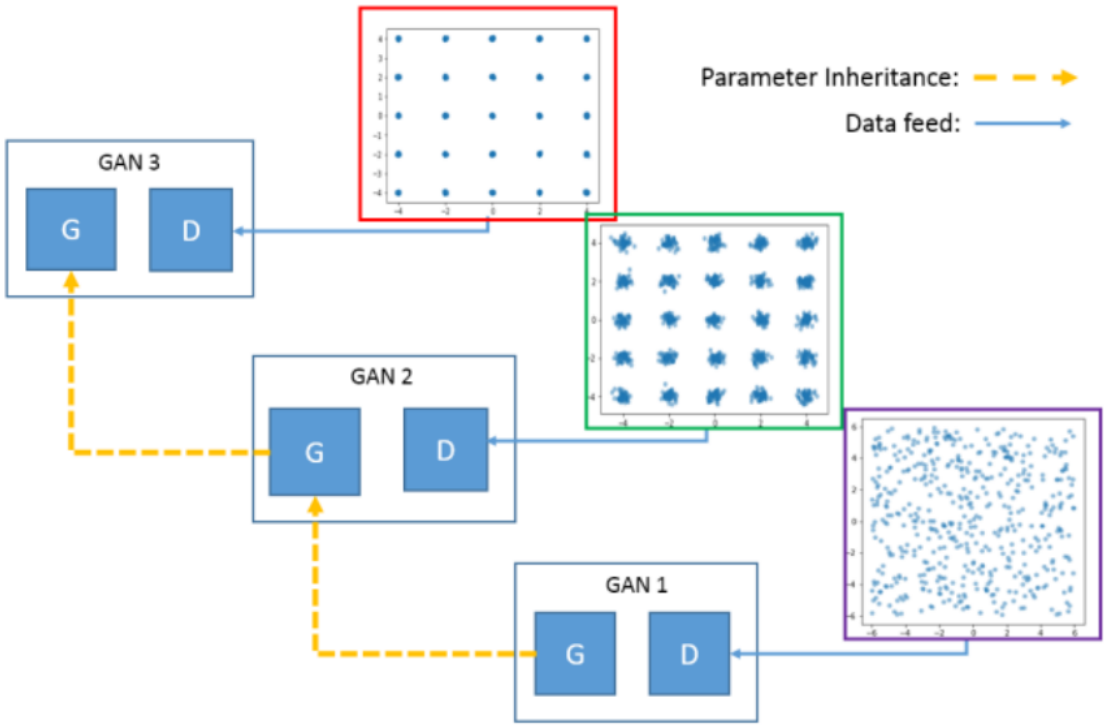


Figure 3.2: Step-Up GANs

In the above figure, we share the architecture of Step-Up GANs. Here the GAN training is discretized into three phases. At GAN 1, we train with the data uniformly distributed over the target distribution. Next, at GAN 2, we add some noise to the target distribution and initialize the weights of the generator of GAN 2 from the generator of trained GAN 1. Finally, GAN 3, initialized from the learn the generator of GAN 2 and learns the target distribution.

Result: Step-Up GANs converged 10x faster 5.9 than the standard Wasserstein-GAN under synthetic setting. We also compared the convergence of Step-Up GAN with WGAN on our own evaluation metric called overlap loss.

Step-Up GANs can also be viewed as a continuation method (through data) where each task is more difficult than its previous one and the original task is the final one to be performed. These findings motivated us to push the boundaries further. We noted a few untouched questions of Step-Up GANs that are listed below.

- How to determine the support of the distribution for the true data distribution?
- Can Step-Up GANs (a data continuation approach) be generalized to work with some well-known datasets for deep learning, the most common example would be MNIST.
- Can this be combined with a model continuation approach, for example, Curriculum Learning [8] presented in literature?

In this thesis, we try to answer these questions and extend our work so that it is applicable to most deep neural networks such as autoencoders.

3.2 Related Work

Neurons (units) in the deep neural networks learn the representation of the data. The last layer is a typical classifier and the previous layers learn different representations of data. Representation learning is particularly interesting because it unfolds a way to do unsupervised and semi-supervised learning [20]. However, training deep neural network is a potentially intractable non-convex problem [7]. One widely applied practical approach is finding a better initialization strategy to find a find superior local minimum. We draw observations from the key techniques used in some of the popular architectures which have been successful in finding the superior local minimum and also achieve better generalization [16]. *Unsupervised pre-training* [23] proved to be a good strategy to train deep neural networks for the supervised and unsupervised tasks. The idea is to greedily train one layer at a time in unsupervised fashion via Restricted Boltzmann Machine (RBM) or an autoencoder, and then use this layer in the task of your interest. This technique has two advantages: first, it provides stable initialization, and second, it increases the regularization effect [20], and hence provide good generalization accuracy [16]. *Fine-tuning* can be seen as an extension of this approach where the learned layers are allowed to retrain or **fine-tune** on the final task [20]. *Transfer learning*, on the other hand, requires two different task, where learning

from one distribution can be transferred to another. This technique is largely used in many visual and text tasks such as image super-resolution, recommendation systems etc.

The basic idea for continuation methods may be described as follows: given a cost function $J(\theta)$, define a series of cost functions $\{J^{(0)}(\theta), J^{(1)}(\theta), J^{(2)}(\theta), ..., J^{(n)}(\theta)\}$ such that $J^{(i)}(\theta)$ is more easier to optimize than $J^{(i+1)}(\theta)$ [20]. Here $J^{(i)}(\theta)$ and $J^{(i+1)}(\theta)$ are preferred to be close so that the solution of $J^{(i)}(\theta)$ can be used to initialize the parameters for $J^{(i+1)}(\theta)$. In other words, one can imagine $J^{(i)}(\theta)$ as a convex envelope [39] for the $J^{(i+1)}(\theta)$ that we gradually reduce as $i \rightarrow n$ and finally we get a solution to the original cost function $J^{(n)}(\theta)$.

We see the above process can be fundamentally done in two ways: model continuation and data continuation. Continuation methods for neural networks were initially introduced [12], and were explained with more insights in *curriculum learning* [8]. Smoothing of the objective function reveals the global curvature of the cost function. Smoothing can be gradually decreased to get a sequence of objective functions with increasing complexity. The mollified objective function is minimized first, and then gradually reducing its smoothness and simultaneously minimizing it as training proceeds to obtain the final solution [8].

When continuation is applied directly to the parameters of a neural network during the training, it can be referred to as **Model Continuation** method. This method forms a convex envelope around the non-convex optimization problem [39]. *Diffusion methods* [38] convoluted a Gaussian noise to the cost function and showed faster convergence. Unlike the *Annealed Gradient descent* , [43] whose cost surface approximation is motivated by some heuristic, diffusion methods have more theoretical backing for Gaussian kernel[39]. *Mollifying Networks* can be seen as an extended version of curriculum learning. Mollified (noisy) objective function can be achieved by injecting normally distributed noise to the weights of each layer of the neural network [21] and following the idea of continuation methods, this noise is gradually reduced as the training proceeds. They also showed that the mollification effect can be achieved by linearization of the network via modified activation functions. Deforming gradually from linear to nonlinear representation has been implemented with varying noise injection methods in [21] and [38]. We, on the other hand, do not use the noise

injection method for linearization of the network, rather discuss how we achieve linearization in section 4.

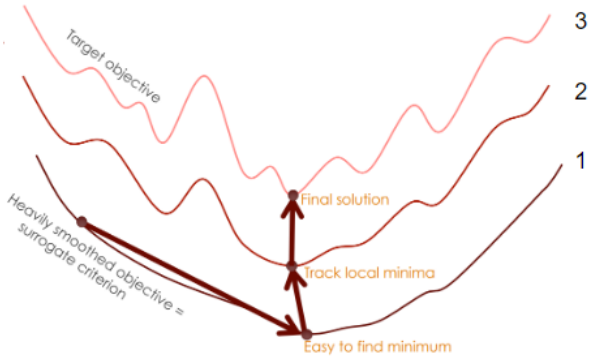


Figure 3.3: Mollified objective function [21]

Three loss surfaces are shown in the above figure, 3 is the original loss surface and 1 & 2 are the surface convoluted with Gaussian noise. Loss surface 1 has more noise than loss surface 2.

Data continuation is intuitively easy to understand but extremely hard to design in the context of continuation methods. The sampling of data points according to its individual complexity is rarely available [49]. Information about the support of the true data distribution can be utilized for reweighting the true distribution, such that the entropy of the distribution increases or the data is more diversified and hence simpler[8]. Initially, weights favor sampling the simple examples from the distribution and gradually as weight increases, it sample more difficult examples from the distribution, hence it is called *curriculum learning*. Notably, this approach was applied discretely (in two steps), however, we found that a continuous sequence of sampling was not well explored. Secondly, in *transfer learning* method [49], the authors recently illustrated a novel method to rank all the data points in the sample, through a data difficulty score computed using SVM. Here, the representations from easy examples were learned first and gradually difficult examples were provided to the model. They also empirically suggested that the direction of gradient step based on easy examples is more effective in traversing the cost surface towards the superior local minimum,

as the variance in the direction of gradients increases with the difficulty of data points [49]. The methods we use in this thesis inherit and extend this idea.

However, some of these methods were tested with a limited number of hidden layers that may not be considered deep when compared to many state-of-art models [21]. Secondly, the conjecture that the combination of data and model continuation may improve the results is not well-tested [21]. There is a lack of systematic experiments to show how the continuation method may help to skip an early local minima and saddle points, with the increasing number of layers, hence the non-convexity [19]. Unlike other papers where experiments were focused on classification task or Recurrent neural networks, we focus on autoencoders.

Chapter 4

Methods

In this section, we introduce a novel method to train and initialize neural networks. To the best of our knowledge, we are first to categorize the ways in which continuation methods may be applied to the deep neural networks. Data continuation and model continuation are two approaches with similar objective of decomposing the original task into a sequence of tasks with increasing level of difficulty.

First, in the model continuation technique, we construct a sequence of cost functions with decreasing mollification, as briefly explained in previous section 3. Second, data continuation can be performed in multiple ways and one of them is gradually modifying the distribution of the data. In this section, we show in detail how we apply these two approaches to enhance learning in deep neural networks.

4.1 Model Continuation

Modifying the objective function while training of deep neural networks is not an easy task and requires a meticulous approach. Researchers have previously shown that linearizing the neural network through activation function can be seen as smoothing of the loss surface [21]. Deep Linear network is the composition of many matrix multiplications without any

activation functions [19]. In literature, researchers have added noise [21] and updated the activation function to control the non-linear behavior of the neural network [38], while keeping the scale of the function bounded [1].

We propose a novel homotopy from a linear to a non-linear network through our activation function which we call, **C-Activation** (Continuation Activation). Our activation function is unbounded at all values of the homotopy parameter. This novel activation function can directly inculcate the effect of continuation methods for optimization of deep neural networks.

4.1.1 C-Activation function

C-Activation can be defined as a homotopic function 2.15 of commonly used activation functions. C-Activation continuously deforms the ability of a network to learn from a linear to a non-linear characteristic of the data. Activation functions such as ReLU, Sigmoid and Tanh can be easily reformulated as shown in this equation 4.1.

$$\begin{gathered}\phi_{C-Activation}(v,\lambda) = (1-\lambda)\cdot v + \lambda \cdot \phi(v) \\ \lambda \in [0,1]\end{gathered} \tag{4.1}$$

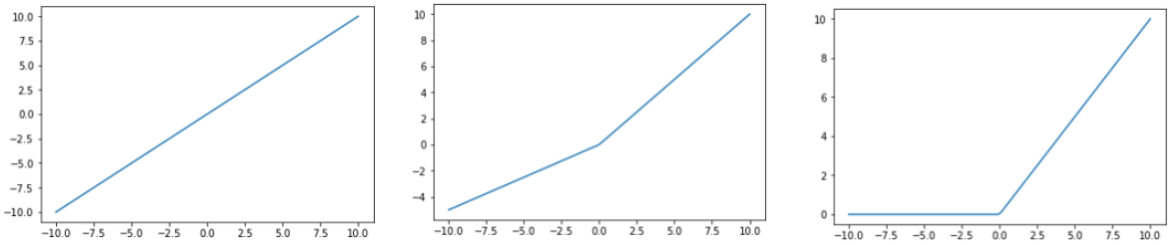
where ϕ can be any activation function such as sigmoid, ReLU, tanh etc. λ is the homotopy parameter and v is the value of the output from current layer. Example of C-sigmoid 4.2, C-ReLU 4.1 and C-tanh 4.3 is shown here [2].

4.1.2 Rethinking of Cost function

C-Activation does not provide a complete picture of the effects of the continuation methods in neural networks. In this section, we want to clarify that changes occur in the behavior of the neural network with the insertion of C-Activation function. Traditionally, we design a neural network to learn from a particular dataset. The deep neural network has many

[1] for example:- if sigmoid is bounded by [0,1], its linear counterpart is also bounded by [0,1].

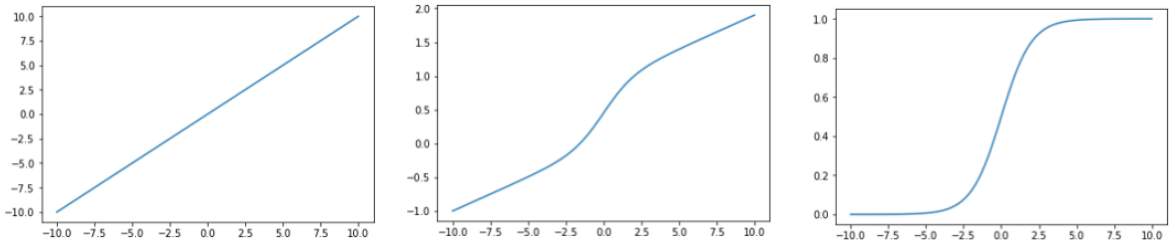
[2] Click the link to see end-to-end deformation from linear to non-linear https://drive.google.com/drive/folders/1hSPzxjweUyX9NL6ZkSdxdj9_cz56boTq?usp=sharing



$\lambda = 0.0$ $\lambda = 0.5$ $\lambda = 1.0$

Figure 4.1: C-ReLU

In this figure, from left to right we show how the C-ReLU function deforms with λ on uniformly distributed points between [-10,10]. At $\lambda = 0.0$, we start from from linear, then at $\lambda = 0.51$ we observe the sigmoid curve is building up slowly. Finally at $\lambda = 1.0$ C-ReLU exactly acts as ReLU Activation function.



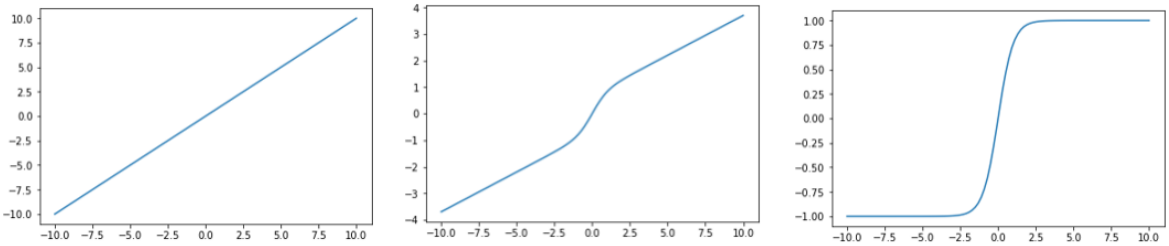
$\lambda = 0.0$ $\lambda = 0.9$ $\lambda = 1.0$

Figure 4.2: C-Sigmoid

In this figure, from left to right we show how the C-sigmoid function deforms with λ on uniformly distributed points between [-10,10]. At $\lambda = 0.0$, we start from from linear, then at $\lambda = 0.918$ we observe the sigmoid curve is building up slowly. Finally at $\lambda = 1.0$ C-Sigmoid exactly acts as Sigmoid Activation function.

hyperparameters to tune from, such as activation functions, depth, width, and cost function etc. We modify this traditional procedure and hence, require some rethinking to understand it better.

What's different? Our activation functions are not fixed for a given neural network with fixed depth and width, because C-Activation is continuously deformed during train-



$\lambda = 0.0$ $\lambda = 0.7$ $\lambda = 1.0$

Figure 4.3: C-Tanh

In this figure, from left to right, we show how the C-Tanh function deforms with λ on uniformly distributed points between [-10,10]. At $\lambda = 0.0$, we start from linear, then at $\lambda = 0.714$, we observe the sigmoid curve is building up slowly. Finally at $\lambda = 1.0$, C-Tanh exactly acts as Tanh Activation function.

ing. Consequently, the underlying cost surface is also changing throughout the training procedure until $\lambda = 1$, because the configuration of network parameters is being altered as we take a step in λ space. Traditionally, a cost function for a deep neural network is a function of its parameters θ as $J(\theta)$, but if we use the C-Activation, we introduce an additional parameter λ. For every λ_i, we solve a different problem $J_i(\theta_i, \lambda_i)$. More precisely at $\lambda = 0$, we solve the linearized network and as λ gradually increases, we solve optimization problems that are slightly difficult and finally at $\lambda = 1$, we solve the original optimization problem with the complete non-linear network. Hence, we decompose the original problem into many intermediate problems as shown in figure 4.4. These intermediate problems can be solved by SGD or its variants, but cannot be solved in parallel. In addition to the usual SGD steps, we take some additional secant steps in lambda (homotopy parameter) space after every batch of SGD steps. We explain this in detail in the next subsection.

Algorithm 2 Natural Parameter Continuation with Secant approximation(NPCS) for AE with model continuation

Require: Learning rate ϵ,Secant frequency $secant\_frequency$, Secant vector scale ω

Require: Initial neural network parameter θ, homotopy parameter λ, Loss history lookup $window\_size$

$k \leftarrow 1$
$loss\_history \leftarrow [\,]$
while stopping criteria not met **do**
    Sample a minibatch from data x^i
    Compute Loss $loss \leftarrow \sum_i J(x, g(f(x)); \theta; \lambda)$
    $loss\_history[k] \leftarrow loss$
    Compute gradient estimate $\hat{g} \leftarrow ADAM\ Optimizer()$
    Apply gradient $\theta \leftarrow \theta\ - \epsilon \cdot \hat{g}$
    $k \leftarrow k + 1$
    if $k\%secant\_frequency == 0$ **then** NPCS
        if $k == secant\_frequency$ **then** NPC
            $\theta \leftarrow \theta_{k-1}$
            $\lambda \leftarrow \lambda_{k-1} + 8e - 3$
        else
            $\theta \leftarrow \theta_{k-1} + \omega \cdot \frac{(\theta_{k-1} - \theta_{k-secant\_frequency})}{||(\theta_{k-1} - \theta_{k-secant\_frequency})||_2 + ||(\lambda_{k-1} - \lambda_{k-secant\_frequency})||_2}$
            $\lambda \leftarrow \lambda_{k-1} + \omega \cdot \frac{(\lambda_{k-1} - \lambda_{k-secant\_frequency})}{||(\theta_{k-1} - \theta_{k-secant\_frequency})||_2 + ||(\lambda_{k-1} - \lambda_{k-secant\_frequency})||_2}$
            if avg($\Delta loss\_history$) of $window\_size$ is more than 0.02 **then** Rescale
                $\omega \leftarrow \omega - 5$
                $secant\_frequency \leftarrow secant\_frequency + 10$
            else
                $\omega \leftarrow \omega + 5$
                $secant\_frequency \leftarrow secant\_frequency - 10$
        $k \leftarrow k + 1$
end while

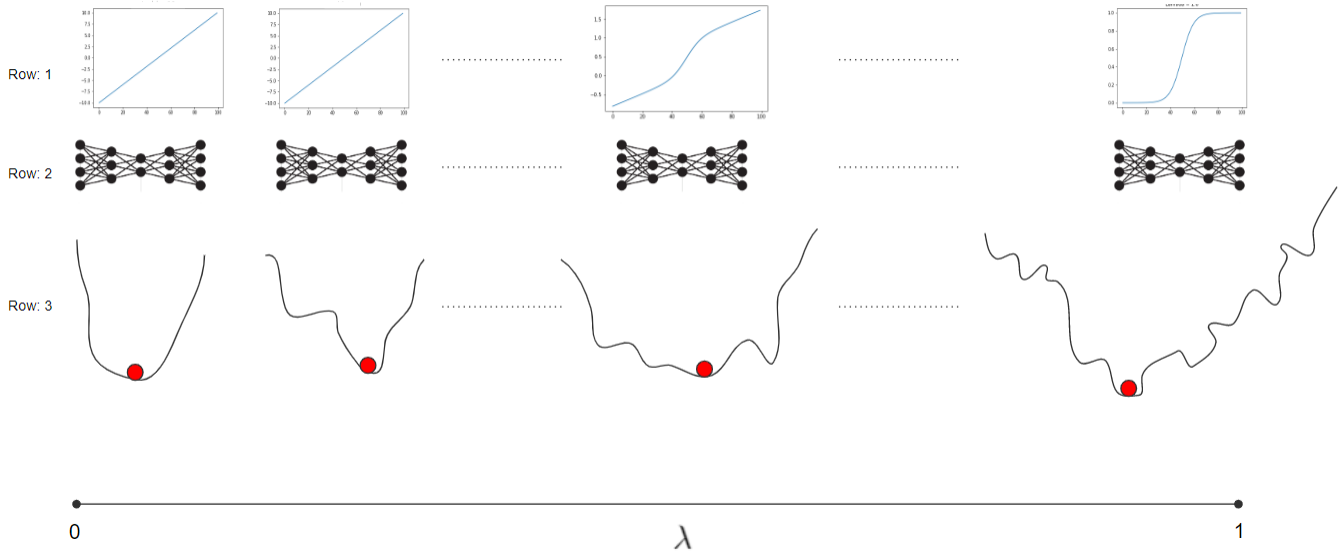


Figure 4.4: What's changing in Lambda space?

In the above figure, we show how the optimization problem changes as λ changes from $0 \rightarrow 1$. From top to bottom, in Row 1: we show how the activation function changes, in Row 2: we show the other settings of autoencoder such as depth and width, which is consistent throughout the network. In Row 3: we show the hypothetical cost surfaces where number of local minimas and saddle points increases as $\lambda \rightarrow 1$.

4.1.3 Natural Parameter Continuation of Neural Networks with Secant approximation

We now have background on how SGD methods 2.4. Also, we know what all internal changes are inculcated to the neural network training as we introduce C-Activation 4.1.2. We start from a relatively simpler problem and gradually solve a more difficult problem, thus forming a sequence of problems. In this section, we will first provide the hypothetical visualization of a coordinate system, which we call $\lambda - \theta$ *curve* and later discuss network initialization technique for every subsequent problem. There are multiple ways to perform this operation, here we will discuss two of them, namely Natural Parameter Continuation (NPC) and Natural parameter Continuation with Secant approximation (NPCS).

Natural Parameter Continuation(NPC): Let us denote the set of parameters of neural

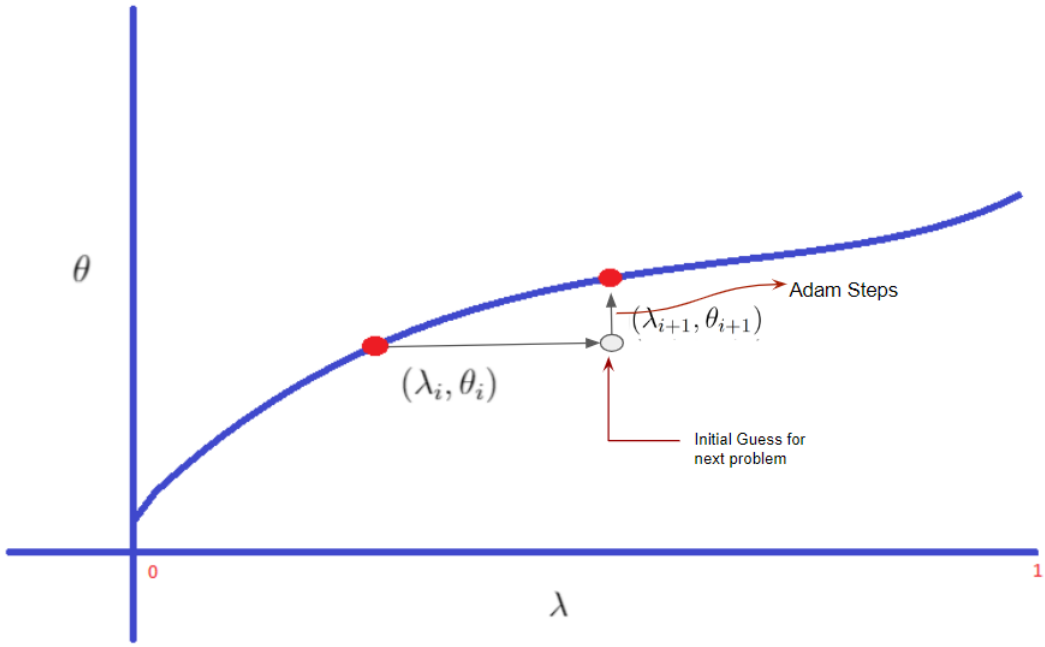


Figure 4.5: Parameters in Lambda space, $\lambda - \theta$ curve

Here we show a hypothetical coordinate space where X-axis is homotopy parameter and Y-axis is the solution to each optimization problem defined at λ. Here θ_i is the solution at λ_i and also used to initialize the problem at λ_{i+1}. θ_{i+1} is the solution at λ_{i+1} after some ADAM steps.

networks as θ. Similar to the approach we explained in section 2.6.1, we can start from the PCA solution (θ_{PCA})[3], because at $\lambda = 0$, we aim to solve a problem that finds a solution to the linear manifold. In other words, at λ_0, most of the activation functions are linear. Hence, we initialize the AE with the solution of PCA, so that we have the advantage of starting from a known solution. Next, we minimize for the optimization problem at λ_i, such that λ_i is very close to λ_0 and obtain θ_i using some SGD or ADAM steps. Next, to find a solution at $(\lambda_{i+1}, \theta_{i+1})$, we use these set of parameters (θ_i), initialize and again apply some ADAM steps to find the solution θ_{i+1}. We may continue this process until $\lambda = 1$, because at this point we solve the original problem. Interestingly, all such solutions or parameters can be

[3]Here, solution at $\lambda = 0$ is θ_{PCA}, which is eqvivalent to θ_0

traced to a one-dimensional curve in lambda space. We have shown this hypothetical $\lambda - \theta$ *curve* 4.5, where homotopy parameter can track all the solutions of gradually deforming problems. However, we do not know the optimal value of $\Delta\lambda$ to take the subsequent steps of the homotopy function. Hence taking many continuation steps on this $\lambda - \theta$ *curve*, could be very slow. So instead of using only the previous solution, we can use the previous two solutions to gain from this $\lambda - \theta$ *curve*, and speed up the process via secant guess to overcome the above mentioned limitation.

Natural Parameter Continuation with Secant approximation(NPCS): This method is very similar to the method explained above, the key difference being the way we initialize the parameters (θ) for the subsequent problem. Here, instead of using the previous solution as the initial guess, we make a secant guess for the subsequent problem. More precisely, at λ_i, we take a certain fixed number of ADAM steps to minimize the cost function, then at λ_{i+1}, we can make a secant approximation by using the parameter of λ_{i-1} and λ_i to initialize the weights of the neural network at λ_{i+1}. In other words, we are doing a piece-wise linear approximation of the curve from λ_i to λ_{i+1} using a secant vector.

We illustrate the same in the figure 4.6. For the optimization problem at λ_3, we can initialize the parameter through a secant update, using the parameter at λ_2 and λ_1. This may prove to be useful in avoiding some local minima. After secant initialization at λ_3, we again apply a certain number ($secant\_frequency$) of ADAM steps, with λ_3 being fixed, as shown in the figure 4.6. We gradually continue this process until $\lambda = 1$; this assists us to find a superior local minimum. The algorithm for parameter continuation via secant approximation is here 2. We use additional parameters, adaptive scaling of secant vector, ω and a frequency at which we update the value of λ, is called $secant\_frequency$. Initial choices of these are dependent on the training data and neural network architecture. We scale these two parameters according to the percentage change in the loss history of a specific window size (loss at previous 400 steps). If we observe decrease in the average loss for the previous window, we increase the value of ω, i.e. we can take longer steps in λ, and at the same time we decrease the $secant\_frequency$, i.e. we take lesser number of ADAM steps to

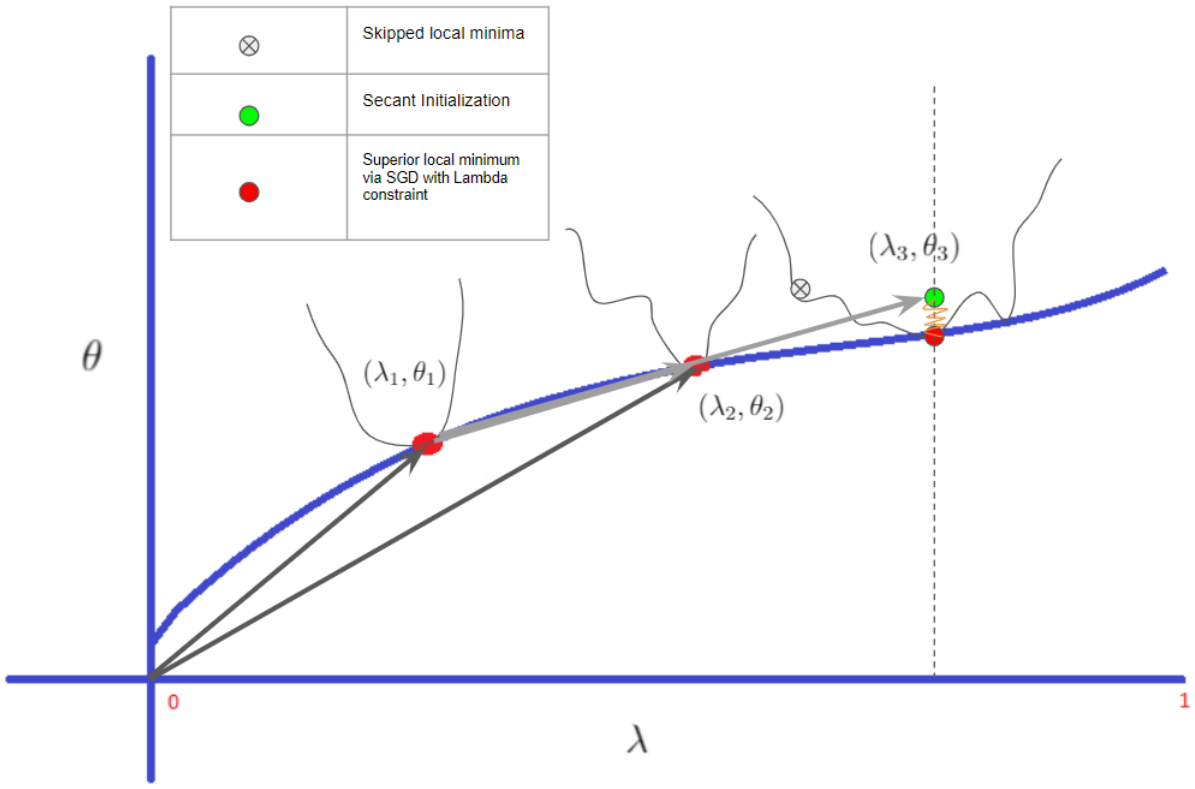


Figure 4.6: Natural parameter continuation via secant in λ space

In this figure, we provide a clear visualization of our algorithm NPCS. Let's say neural network was initialized with θ_{init}; at λ_1, the solution θ_1 is obtained, then we perform our first NPC step, which is defined by user. After some ADAM steps, solution at λ_2 is θ_2. Further, for λ_3, and all subsequent values of λ, we perform with secant approximation using previous two points in this $\lambda - \theta$ *curve*.

minimize the current problem. An illustration of effective scaling is shown in the figure 4.7, which shows if we perform incorrect scaling of ω then we may end doing poor initialization for the subsequent problem.

4.2 Stable initialization of Autoencoder through PCA

PCA is well-known as a dimension reduction technique that provides lower dimensional (linear) manifold of the data. As explained in the Section 2, PCA projections can be computed

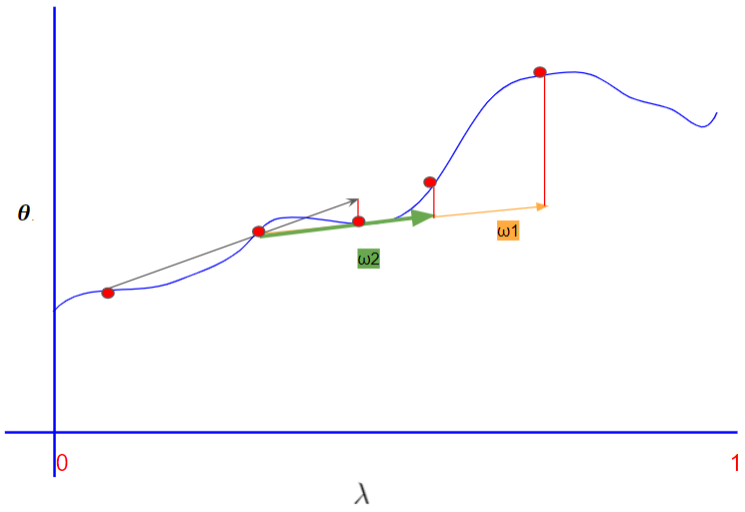


Figure 4.7: Adaptive ω

In this figure, we shown the importance of adaptive *omega*, *omega*2 is the incorrect scaling and we can see if wee take big steps in λ through large ω values we may end up doing a poor initialization for the subsequent problem.

by SVD method. We use SVD to compute U, S, V^T from the centered data as shown in equation 2.5. Further, from the equation 4.3, we understand that V can be seen as a mapping of data from high dimensional space to the lower dimension (linear) manifold and V^T is a mapping from a lower manifold to the data. This behavior of SVD empowers us to initialize the weights of the hidden layers of the autoencoder. More precisely, we use the first n columns of V for the encoder layer with width n and for the decoder, we use the transpose of the weights used for encoder as shown in the equation 4.4.

$$X^{r,c}_{centered} = X^{r,c} - \sum_i \frac{X^{i,c}}{r} \tag{4.2}$$

where $X^{r,c}$ represents input data matrix with rows r and and columns c.

$$\begin{aligned} X_{centered} &= USV^T \\ X_{centered}V &= USV^TV \\ X_{centered}V &= US \end{aligned} \tag{4.3}$$

$$W^n_{encoder} = V_{n\text{-}columns}$$
$$W^n_{decoder} = (W^n_{encoder})^T \tag{4.4}$$

where W^n represents the Weight matrix of the encoder layer with n as width.

There are multiple advantages of initializing an AE using PCA. First, we start the deep learning training from a solution to the linear manifold of the data rather than something random, which in itself is a big win. This is because the set of parameters obtained presumably would have caught some key characteristics of the data. Second, in the NPCS method, C-Activation defines the homotopy as a continuous deformation from a linear to a non-linear network. Note that PCA also gives a solution to the linear manifold of the data. Therefore, the PCA solution is expected to be very close to the solution of the first step of the NPCS method (at $\lambda = 0$). Third, since C-Activation is unbounded, we empirically observed that random initialization of the network with wide and deep layers leads to unstable training, but the PCA initialization proves to be a robust solution here as well.

In order to condition the training of autoencoders, we propose to initialize the encoders and decoder by PCA. The idea of initializing the deep feedforward networks with PCA is not novel and has been independently explored in literature [46], [32], [10].

4.3 Data Continuation

Researchers have shown that the cost functions like KL-divergence and JS-divergence fail when there is no overlap between the support of the true and the generated distributions [4]. Also, adding noise to the data distribution usually helps to increase the stability of GANs training [47]. In this section, we share a data preprocessing technique that could be considered as a smarter way of adding noise to the data. The basic idea for data continuation was shared in curriculum learning [8], to define a sequence of data distributions by re-weighting the data samples in the order of difficulty of the samples. We take this idea further and introduce a data continuation technique by using an oversampling method.

Instead of sampling the data in the order of their difficulty, we oversample it to increase the distribution such that the generated data is uniformly distributed over the support of the target distribution and then we gradually decrease this support to attain the target distribution back or transform back to the intrinsic dimensions of the data. Please note, the final deep learning model we train is always on the true distribution, using data continuation techniques we just modify the sequence of providing this data to the model.

We present a novel data continuation approach that leverages the support of the data distribution. We extend the SMOTE [9] algorithm and oversample the data to expand the coverage over the support of its distribution. This benefits the neural networks for learning from the data. One key advantage of extending SMOTE is that it operates in feature space rather than data space and hence, can be easily applied to various datasets. Originally, SMOTE was designed to oversample the minority class considering the balance between the two classes. We redefine our objective and use C-SMOTE 3 is used to oversample a single class over the support of the distribution of that particular class. We oversample by introducing many samples along the line segments joining any or all of the k-nearest neighbors. Let us denote this line segment by d and we define a parameter α that controls the length of this line segment as shown in the equation 4.5. As α changes from $0 \rightarrow 1$, data distribution is modified from a uniform distribution over the support to the target distribution. For more motivation, an explanation is provided in the figure 4.8 and some example illustrations can be seen in these figures 5.7, 5.8.

$$\begin{gathered} d_{new} = d \cdot m \\ m \in [0, \tfrac{1}{2} - \alpha] or [\tfrac{1}{2} + \alpha, 1] \\ \alpha \in [0, \tfrac{1}{2}] \end{gathered} \tag{4.5}$$

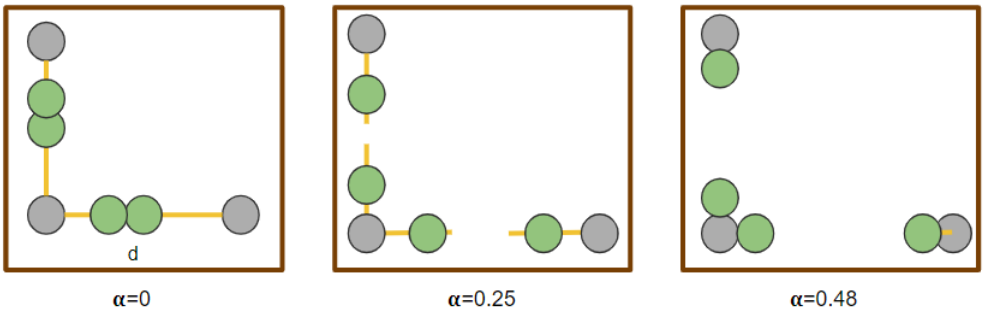


Figure 4.8: Continuous-SMOTE explanation

In the above figure, from left to right, we show oversampling of new data points (green), given some true distribution of data (grey points). Here α controls the length of d (yellow line), where the generation of points are allowed. We can see at $\alpha = 0$ that the generated points can be anywhere on the yellow line d randomly. Further as $\alpha \to 0.5$ in right most figure, the generated points are very close to the nearest neighbour or the point itself.

Algorithm 3 C-SMOTE(x, t, k, α)

Input: Raw data matrix x; t is number of samples to be inserted between a pair of neighbours; Number of nearest neighbours k; α (Noise parameter) controls the distribution of the data

Output: (t* Number of samples in x) Synthetic samples depending on α

numsamples = Number of samples in x
numattrs = Number of attributes
Sample[][] : array for original data matrix
Synthetic[][]: array for synthetic samples
newindex : keeps a count of number of synthetic samples generated, initialized to 0
for $i \leftarrow 1$ **to** numsamples **do**
    Compute k nearest neighbors for i, and save the indices in the nnarray
    for $j \leftarrow 1$ **to** t **do**
        Populate(i, j, t , nnarray)
return Synthetic

Populate(i, j, t, nnarray) - Function to generate the synthetic samples.
Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
for $attr \leftarrow 1$ **to** numattrs **do**
    Compute: dif = Sample[nnarray[nn]][attr] -$Sample[i][attr]$
    Compute: gap = random number between 0 and $\frac{1}{2}$-α
    if $j \geq \frac{t}{2}$ **then**
        Overwrite: gap = random number between $\frac{1}{2}+\alpha$ and 1
    Synthetic[newindex][attr] = Sample[i][attr] + gap $\cdot$ dif
$newindex++$
return - End of Populate

End of Pseudo Code

Chapter 5

Experiments

5.1 Datasets

In our experiments, we used Fashion MNIST and MNIST dataset. MNIST [42] is one of the most widely used datasets in Deep learning research. It is a data set for handwritten digits and can be used to test the supervised or unsupervised learning models. The training set contains 55,000 images, and each image is a 28×28 resolution. Similarly, we have Fashion MNIST [52], where the pictures are of fashion instead of digits. The number of training images and dimensions are exactly the same as MNIST. Thus, Fashion MNIST is considered to be relatively complex dataset than MNIST. Few images from fashion MNIST and MNIST are shown in figure 5.1 and 5.2 respectively.

To examine the robustness of C-SMOTE, in addition to MNIST, we designed some Synthetic datasets such as Sine-wave and 2D Multi-Gaussian Grid. Sine-Wave has four signals and a total of 100 data points as shown in the figure 5.3. Grid data has 25 Gaussian distributions and a total of 500 data points that are evenly placed in $[-2, 2] \times [-2, 2]$ with zero standard deviation as shown in the figure 5.4.

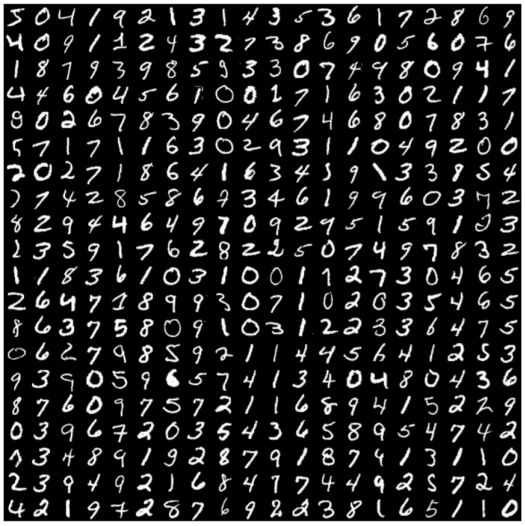


Figure 5.1: MNIST Dataset [42]

In this figure we show the first 400 handwritten digits in the MNIST dataset. Each image has a resolution of 28×28.

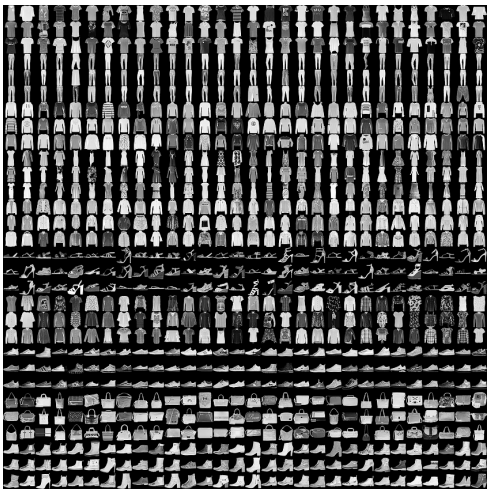


Figure 5.2: Fashion-MNIST Dataset [52]

In this figure, we can see the variety of images in Fashion MNIST dataset. From this dataset, we show three rows from all the ten classes present. Each image has a resolution of 28×28.

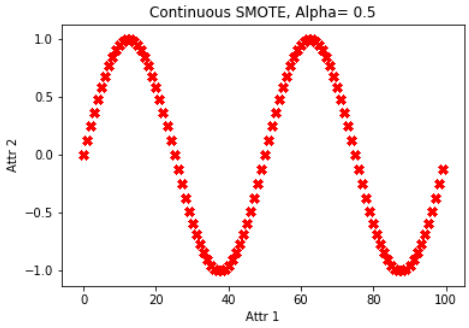


Figure 5.3: Sine-Wave data

In this figure, we have 100 data points distributed along a simple sine wave with a total of four signals or phase shifts. So the shape of this data is (100,2).

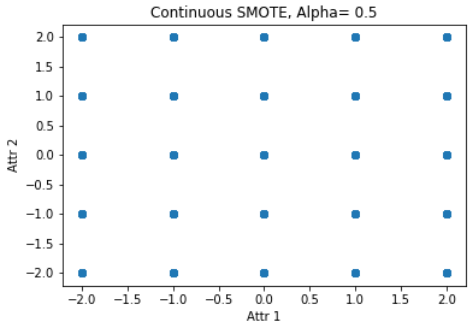


Figure 5.4: 2D Multi-Gaussian Grid data

In this figure, we have total of 25 data points that are evenly placed from in the range of $[-2, 2] \times [-2, 2]$. So the shape of this grid data is (25,2).

5.2 Neural Network Architecture

Autoencoder(AE) we employed in our experiments is depicted in the figure 5.5. The input to the AE is a 784-dimensional image from the Fashion MNIST dataset 5.2 and output is the reconstruction of the same (784 dimensions). We use this AE to attempt the reconstruction of the image from two dimensional manifold, which is encoded in the code layer. We apply one particular *activation* function to all the hidden layers of the network, except the code and the output layer. For instance, activation function can be any C-Activation such as C-ReLU,

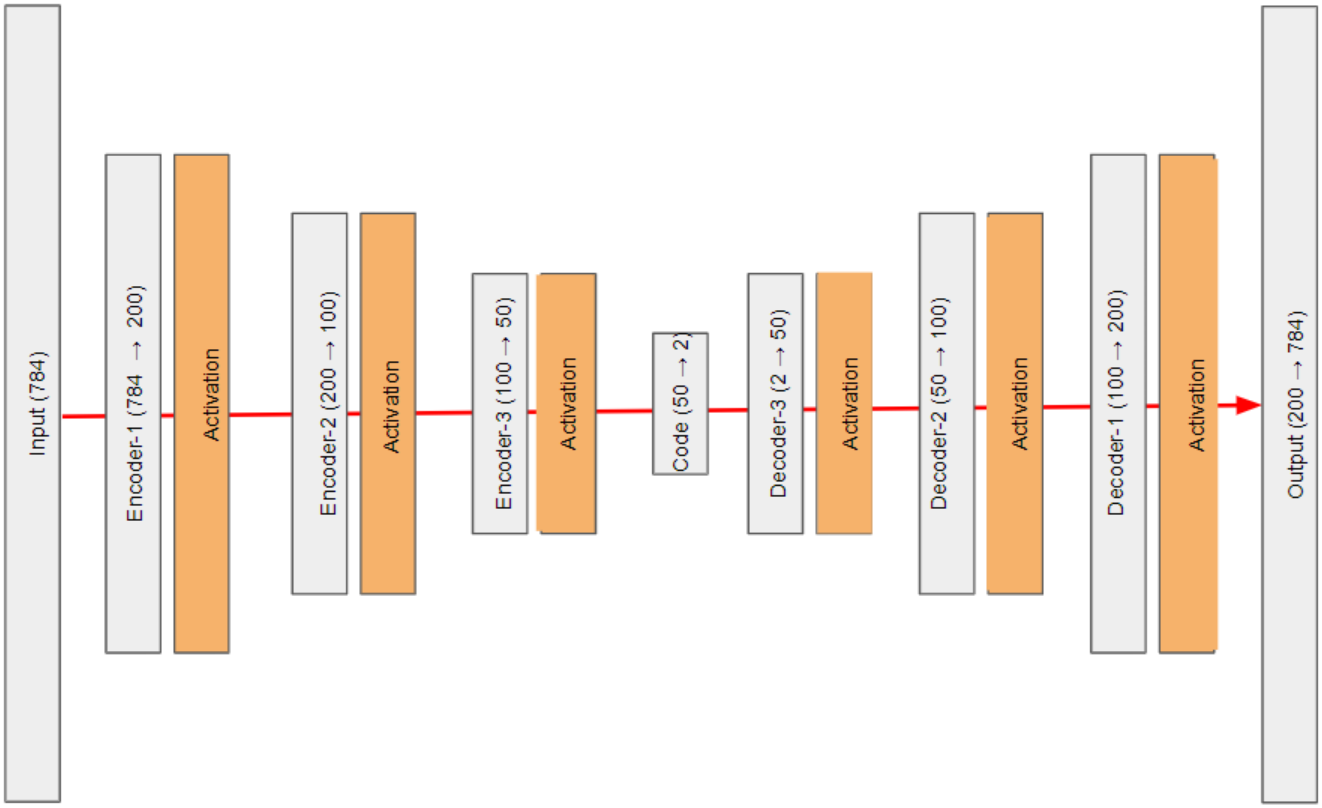


Figure 5.5: Autoencoder with various activation functions

Here, we show an eight layer Autoencoder(AE), with code dimension as two. The activation function used here actually varies from experiment to experiment, but we use the same activation function across all hidden layers as depicted above. C-Relu, ReLU, Sigmoid or C-Sigmoid are few potential examples. At every layer, we also show the dimension or the width of the neural network such as for layer Encoder-1 input dimension is 784 and output dimension is 200.

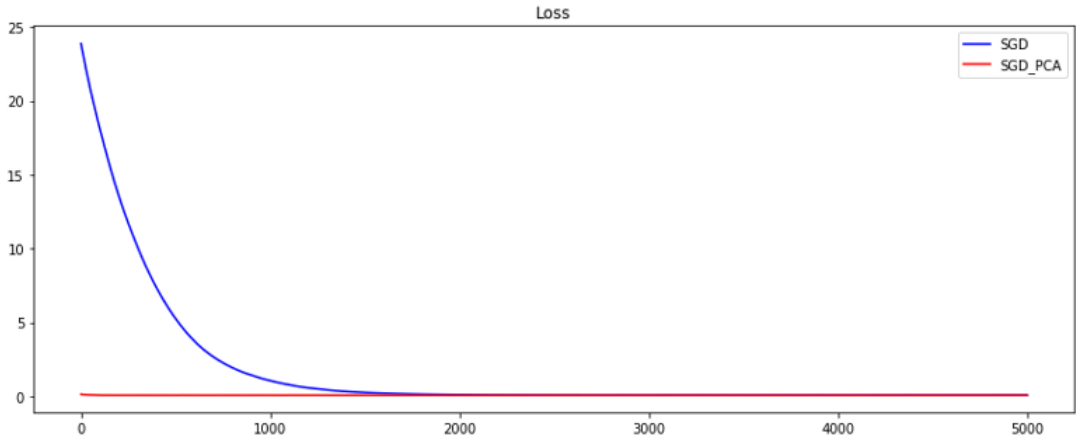


Figure 5.6: Stable Initialization through PCA

In this plot we show the efficiency of the PCA initialization. Blue curve is the loss when neural network is initialized randomly. Red curve shows loss with PCA initialization.

C-Sigmoid or C-Tanh for NPCS method. Further, to compare our method (NPCS) with the traditional ADAM (SGD variant), we use the usual ReLU, Sigmoid, or Tanh as *Activation* in AE 5.5. We will clarify the activation function we used in the respective experiments while reporting our results.

We did not apply any normalization to the data or to the model such as batch normalization, weight normalization [51] etc. We do center the data according to the equation 4.2, when we use PCA initialization for AE in most of our experiments. In addition to that, we used Python 3.6.5 and Tensorflow 1.10.0 on CPUs for all our experiment as the computations are not very expensive.

5.3 Results

5.3.1 PCA initialization results

Random initialization of neural networks is one of the widely accepted techniques in deep learning. Here, we show the results for another effective initialization technique using PCA.

For this experiment we use the AE 5.5, with all the hidden layers sigmoid as the activation function, lets denote this network as $AE\text{-}ADAM_{sigmoid}$. Result for $AE\text{-}ADAM_{sigmoid}$ is shown in the figure 5.6. We clearly see that the PCA initialization is much more stable with traditional ADAM. Precisely, for a particular random seed (85), with random initialization, the loss at step one is 24.194 but with PCA initialization, loss at step one is 0.0675. The dataset used in this experiment is Fashion MNIST.

5.3.2 C-SMOTE results

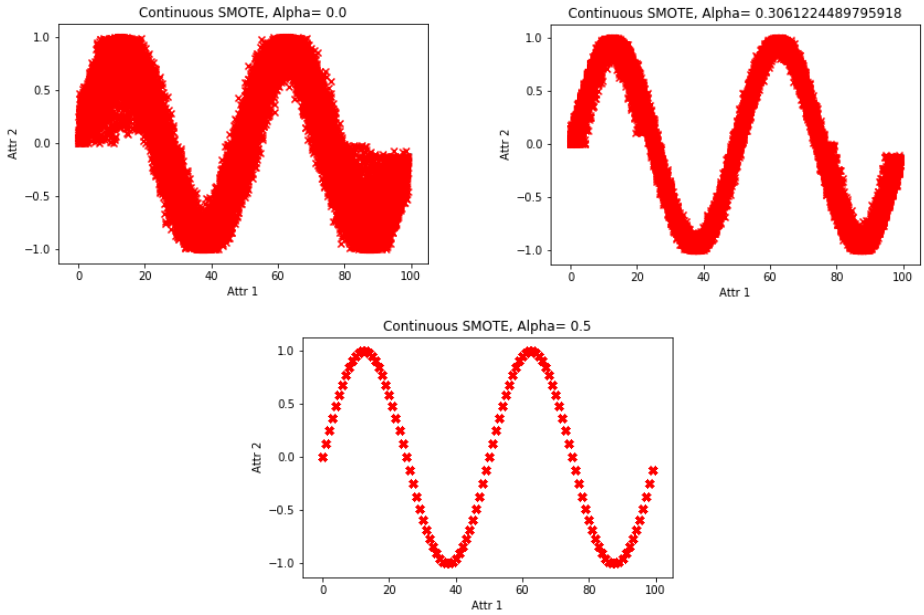


Figure 5.7: C-SMOTE applied to a Sine wave

In this figure, we show the result of C-SMOTE to oversample the Sine-Wave. From left to right we observe the distribution of data is wider at $\alpha = 0$ than $\alpha = 0.30$ and all the oversampled data points are pushed back to the intrinsic dimension of the sine wave at $\alpha = 0.5$

We applied the C-SMOTE algorithm on a variety of datasets. This pre-processing algorithm does not require any modification of datasets before their application. Results for Sine-Wave and 2D Multi-Gaussian Grid data is shown in figure 5.7 and 5.8 respectively.

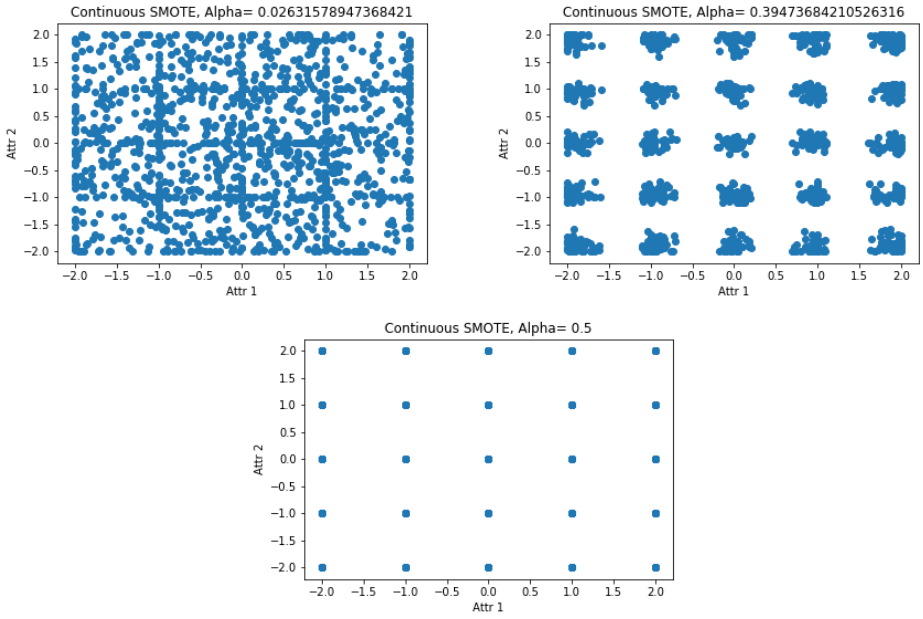


Figure 5.8: C-SMOTE applied to a Grid with 25 Gaussian modes

In this figure, we show the result of C-SMOTE to oversample the grid data. From left to right, we observe the distribution of data is almost uniform at $\alpha = 0$ then at $\alpha = 0.30$ we observe few clusters forming near the means of the 25 distribution and again all the oversampled data points are pushed back to the intrinsic dimension of the grid at $\alpha = 0.5$.

Here, for both the datasets, we used, 25 as the number of neighbors (k) and we populate 200 points between any two neighbors (t). From the images 5.7, we observe that at $\alpha = 0.5$ data strictly follow the true data distribution and for smaller values of the α data distribution is noisier. Under the motivation 3, we witnessed how the noisier data distributions can help to achieve stable WGAN training and even other deep learning architectures with KL and JS divergence as loss functions. We share the results of the Step-up GAN[1] 5.9 here, to empirically illustrate the effectiveness of C-SMOTE. Please note we use Wasserstein distance in all GANs in the presented result 5.9. Clearly, Step-Up GANs is 10 times faster compared with the convergence obtained from the same architecture as $WGAN_{vanilla}$ and less biased compared with the convergence from $WGAN_{BN}$.

[1]From our previous work in collaboration with Xiaozhou Zou

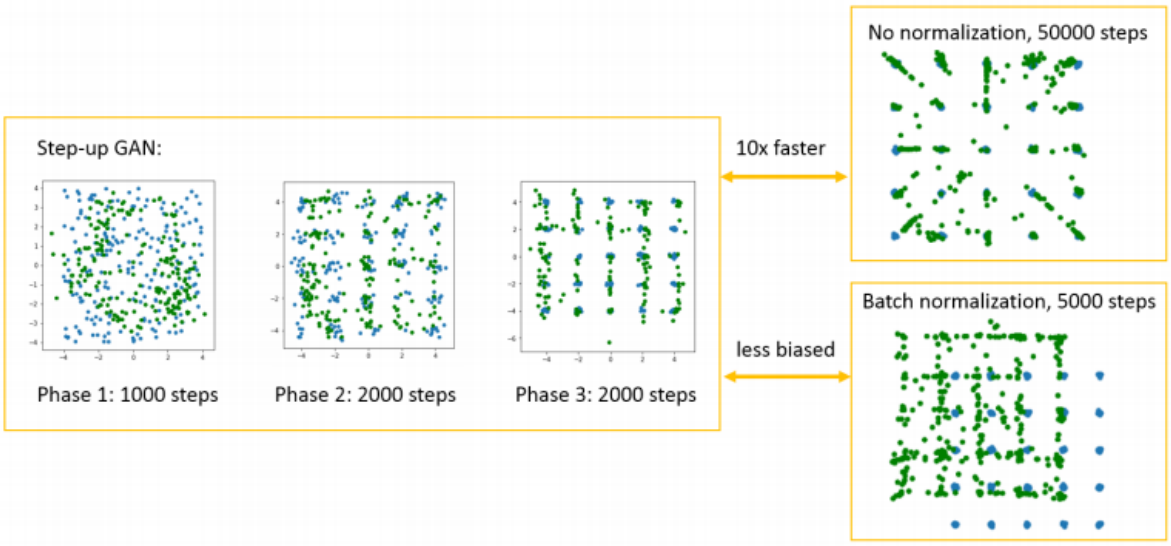
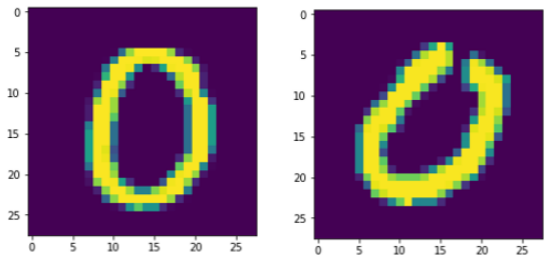


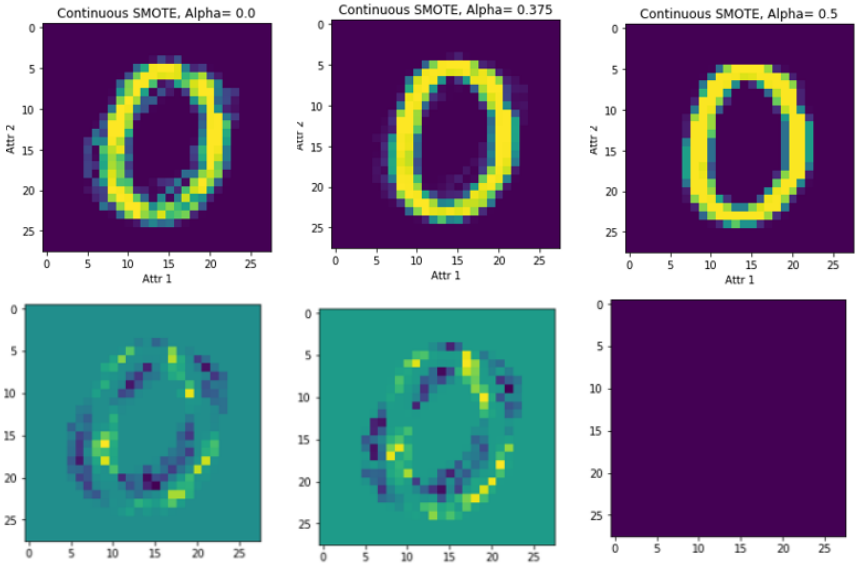
Figure 5.9: Step-Up GAN result

In this figure, we compare the convergence of Step-Up GAN introduced in figure 3.2 with other two. The first convergence is obtained from a simple feedforward $WGAN_{vanilla}$ with no batch-normalization . In this convergence, the generated(green) points are well distributed amongst all the modes, but takes 50000 training steps to obtain. The second convergence is achieved by adding batch normalization to the $WGAN_{vanilla}$, say $WGAN_{BN}$. $WGAN_{BN}$, convergence is reached in 5000 steps, but generated grid is biased. The convergence from the **Step-up GANs** takes in total 5000 steps to reach (1000 steps in the first phase. 2000 steps in second and third phase), and is equally good to the convergence of $WGAN_{vanilla}$ which takes 50000 training steps.

We wanted to investigate how C-SMOTE would effectively perform with the images, hence we used MNIST dataset to oversample more digits. A specific example is shown in figure 5.10. Here, we take first two zeros of the MNIST dataset and use C-SMOTE to oversample a total of 10 digits (zeroes). In this experiment, we kept 2 nearest neighbors (k) and 5 data points (t) to be filled in between these two zeroes.



Zero 1 Zero 2



6.512 1.869 0.0

Figure 5.10: C-SMOTE applied to a MNIST digit zero

In first row, we show the first two zero's in the MNIST dataset. In second row from left to right, we can see that at $\alpha = 0$, the image of zero is a random combination of the two zeros. At $\alpha = 0.375$, we see more clear image of first zero and finally at $\alpha = 0.5$ we have the first zero back again. Finally, in third row, we show the difference in the images of **Zero 1** and zeroes at *alpha* equals 0, 0.375 and 0.5 respectively. We also report the sum of all pixel values of images in the third row, we clearly see the sum is decreasing with the α increasing.

5.3.3 NPCS results

In this section, we share the results of our method - NPCS, compared to ADAM. Specifically, we used Adam optimizer in the implementation, and refer that as a variant of SGD. Here,

we used Fashion MNIST dataset and the AE 5.5 to obtain the results in this section. We compared our methods with three different activation functions. We use the same activation function across the AE for simplicity, such as C-ReLU, ReLU, C-sigmoid, sigmoid etc. First, $AE\text{-}NPCS_{C-ReLU}$ is compared with $AE\text{-}ADAM_{ReLU}$. For both these methods, all other hyperparameters are kept same such that batch size is 200, the learning rate is $5e-5$, input data is initially centered and initialization is through PCA. Additionally, the NPCS method requires two more hyperparameters to be tuned i.e $secant\_frequency = 100$ and $\omega = 5$. We illustrate the results of this experiment in figure 5.11. As observed from the first plot of the figure which shows the reconstruction loss on the y-axis, $AE\text{-}NPCS_{C-ReLU}$ is able to obtain a superior local minimum and on the other hand, $AE\text{-}ADAM_{ReLU}$ is unable to converge better. Next, in the second plot 5.11, we note the sequence of the λ values we took to complete the homotopy. At step 100, our method takes the natural continuation (NPC) update which is fixed, $\Delta\lambda_{100} = 8e-3$. The subsequent lambda updates, $\Delta\lambda_{200}, \Delta\lambda_{300}$ and $\Delta\lambda_{400}$ are computed with secant according to the NPCS algorithm 2. In addition to that, we share the lambda norm and theta norm for all three secant updates, in the third and the fourth plot of the figure 5.11. Please note the ω and $secant\_frequency$ are adaptive to the loss.

Similar to the above experiment we designed another experiment with Tanh. We compared $AE\text{-}NPCS_{C-tanh}$ compared with $AE\text{-}ADAM_{tanh}$, and the results are shown in figure 5.13. Here, both the methods show good convergence, in the first plot of the figure 5.13, but during later steps $AE\text{-}NPCS_{C-tanh}$ shows better convergence and hence reached to superior local minimum. In the subsequent plots of the figure we illustrate the λ values , λ norm and θ norm. Further, we designed the experiments for sigmoid function. Here $AE\text{-}ADAM_{sigmoid}$ is compared with $AE\text{-}NPCS_{C-sigmoid}$ but, we need to change value of ω $5.0 \rightarrow 20.0$, to achieve the stable convergence in this case. Results of this experiments are shown in the figure 5.12. Here, we report that similar results are achieved with ADAM and NPCS optimization methods. Finally, we show a table of experiments 5.3.3, that shows the behaviour of NPCS method with various ω values. We present the cases when we are not able to

$secant\_frequency$	ω	**Final λ**	**No. of continuation steps**	**Loss at final step**
1000	0.5	1	23	1430
1000	1.5	0.1398	50	0.043
1000	5	1	9	0.049
100	1	1	4	0.049
100	5	1	4	0.041
100	10	1	2	0.047
10	0.5	1	6	0.067
10	1	1	4	0.0486
10	5	1	2	0.0482

Table 5.1: $AE-NCPS_{C-ReLU}$ is trained with various initial $secant\_frequency$ and ω values, shown in this table. We also show total number of continuation steps taken and observe respective final value of λ and loss.

complete the homotopy (i.e final $\lambda \neq 1$), and also the cases when the final loss is extremely high because of the bad choice of λ. This explains the importance of these parameters and require tuning depending upon the various hyperparameters of the deep learning model.

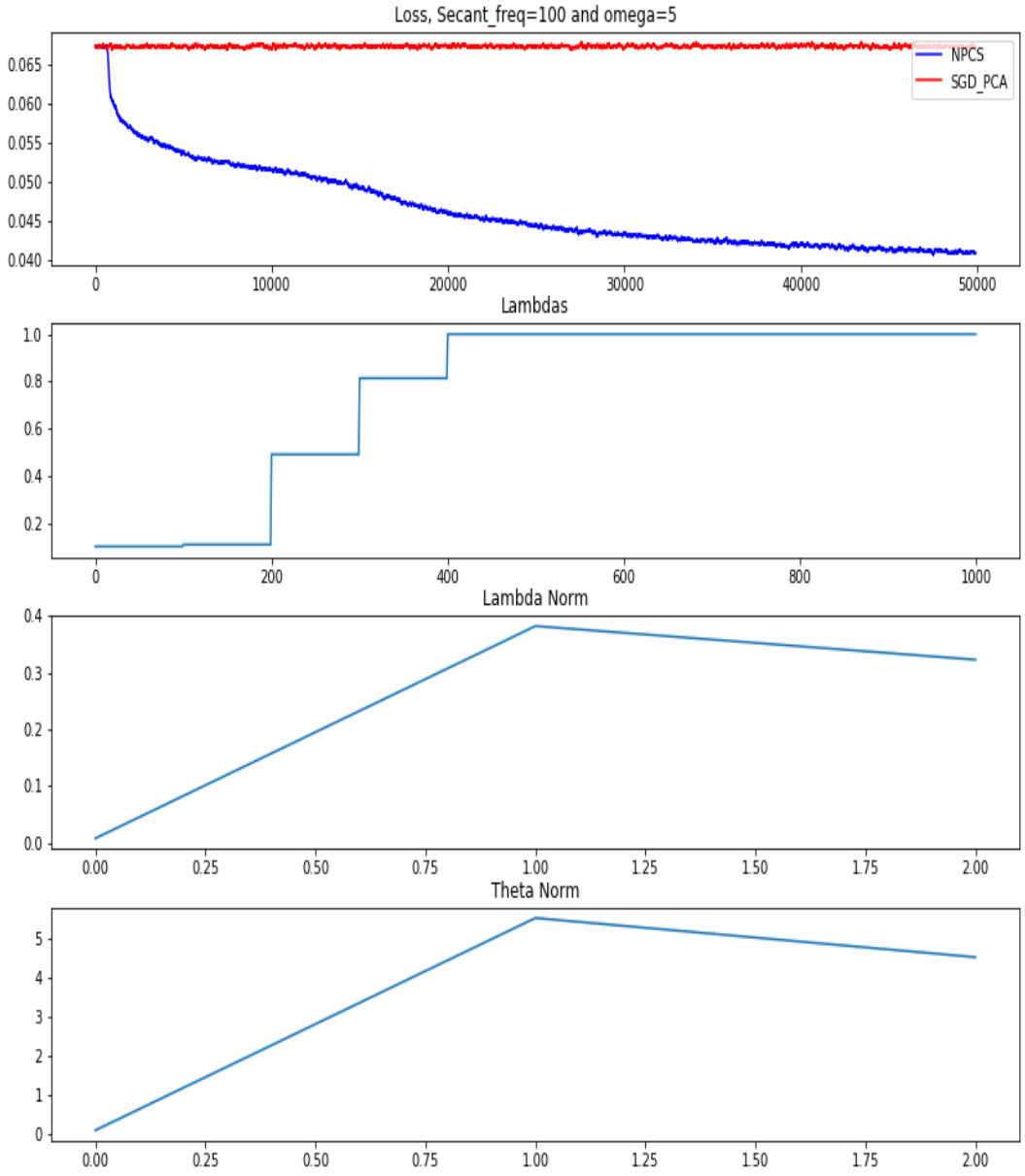


Figure 5.11: NPCS with C-ReLU

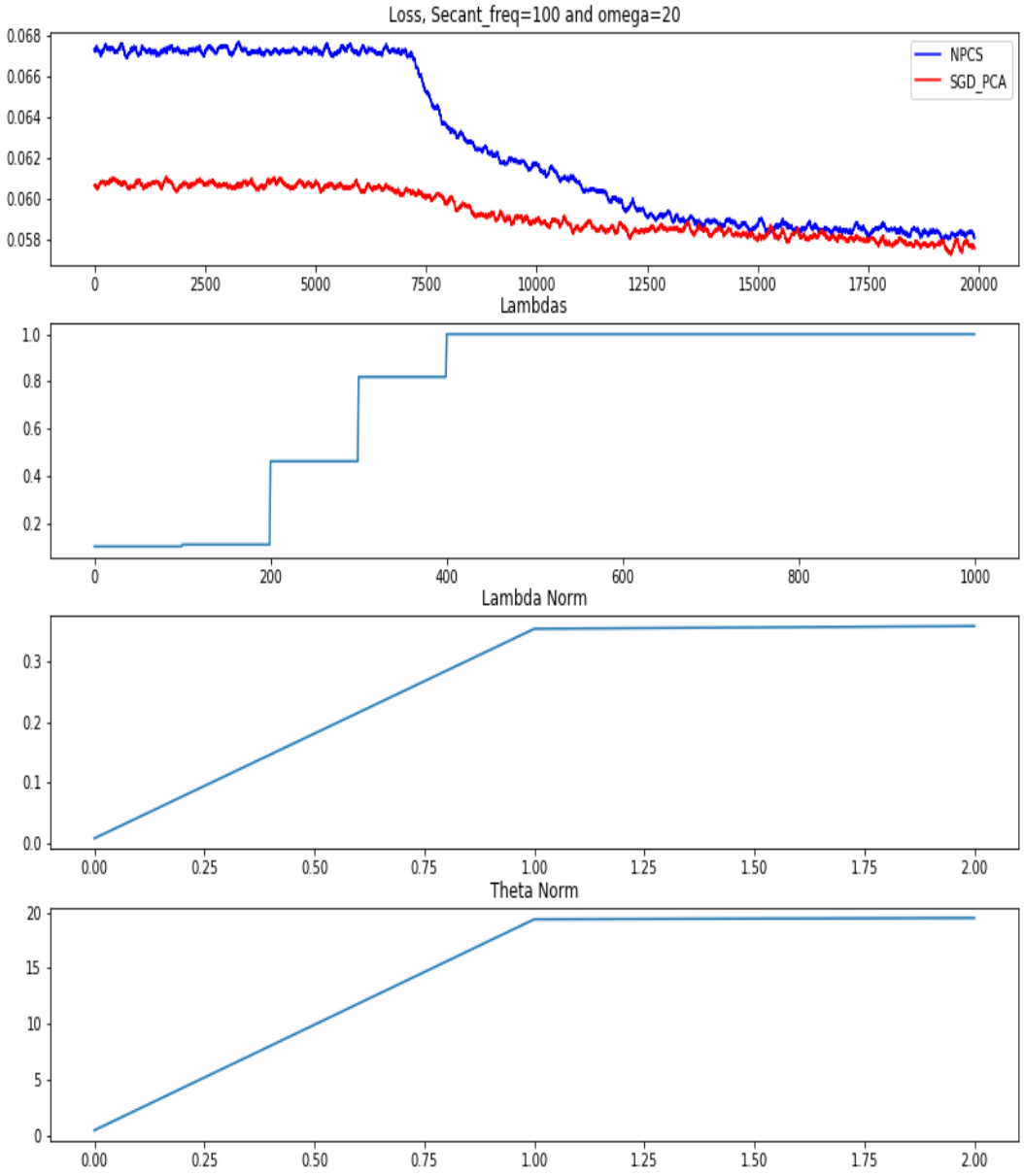


Figure 5.12: NPCS with C-Sigmoid

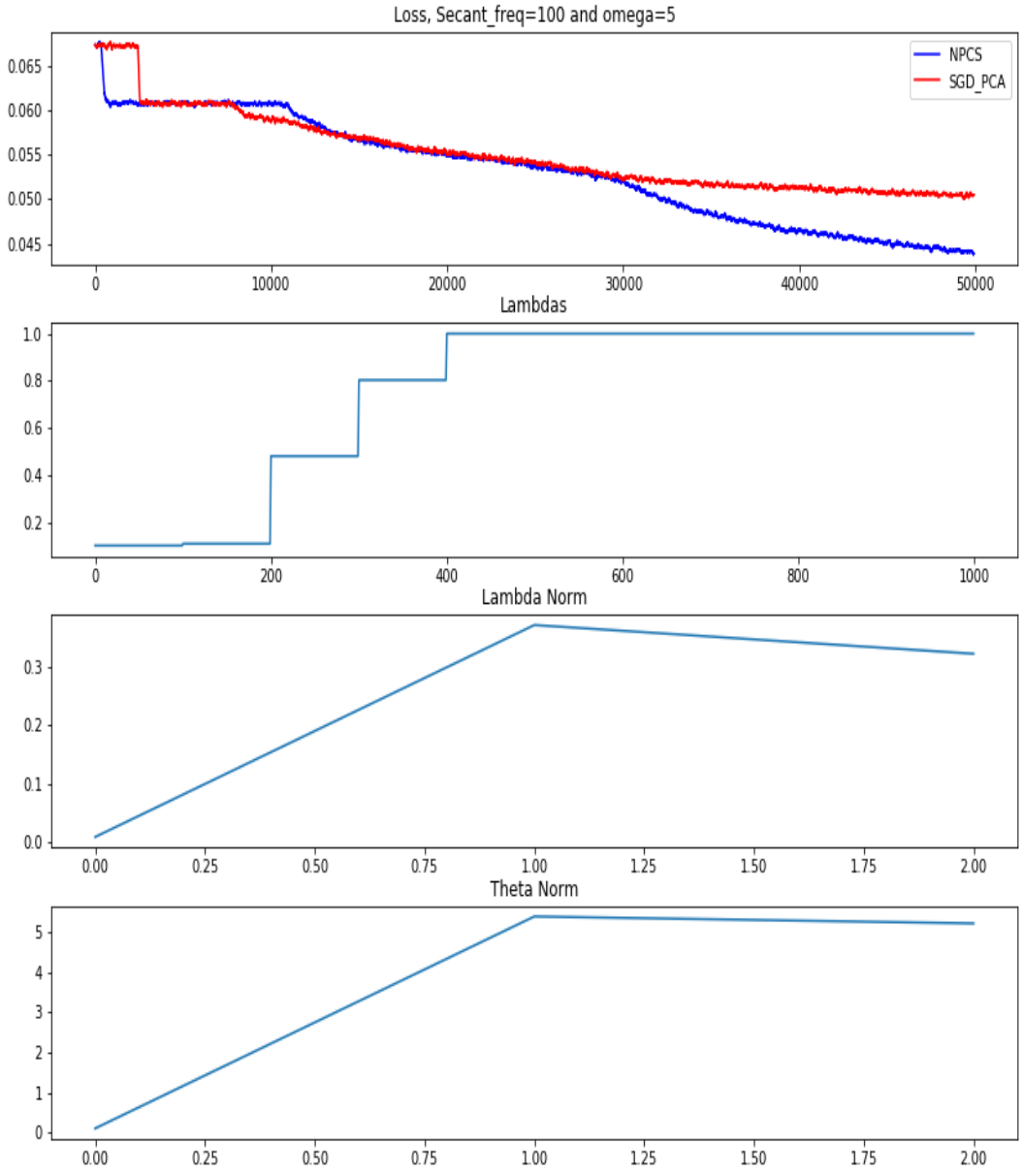


Figure 5.13: NPCS with C-Tanh

Chapter 6

Conclusion and Future Work

In this thesis, we addressed the most common and yet open problem, the deep learning optimization. We studied the background in detail, and design the layout of relationships amongst some quality works on continuation methods. We inferred that the continuation methods can be classified into model and data continuation.

In our model continuation approach, we first define the C-Activation function which allowed us to decompose the original problem into a sequence of problems with increasing complexity. This can also be seen as homotopy from a linear to a non-linear network. Additionally, we developed a method NPCS that potentially accelerated up the training because of the secant steps and also helps to avoid many bad local minima. We then show the results of NPCS method and see that our method outperforms the traditional ADAM optimizer in our experiments with ReLU and Tanh, while for Sigmoid, the results were similar.

Secondly, we presented C-SMOTE as a data continuation approach. We applied this preprocessing technique to various datasets such as MNIST, Sine-wave and Grid data. We also presented our work of Step-Up GANs that shows how C-SMOTE can be effectively applied.

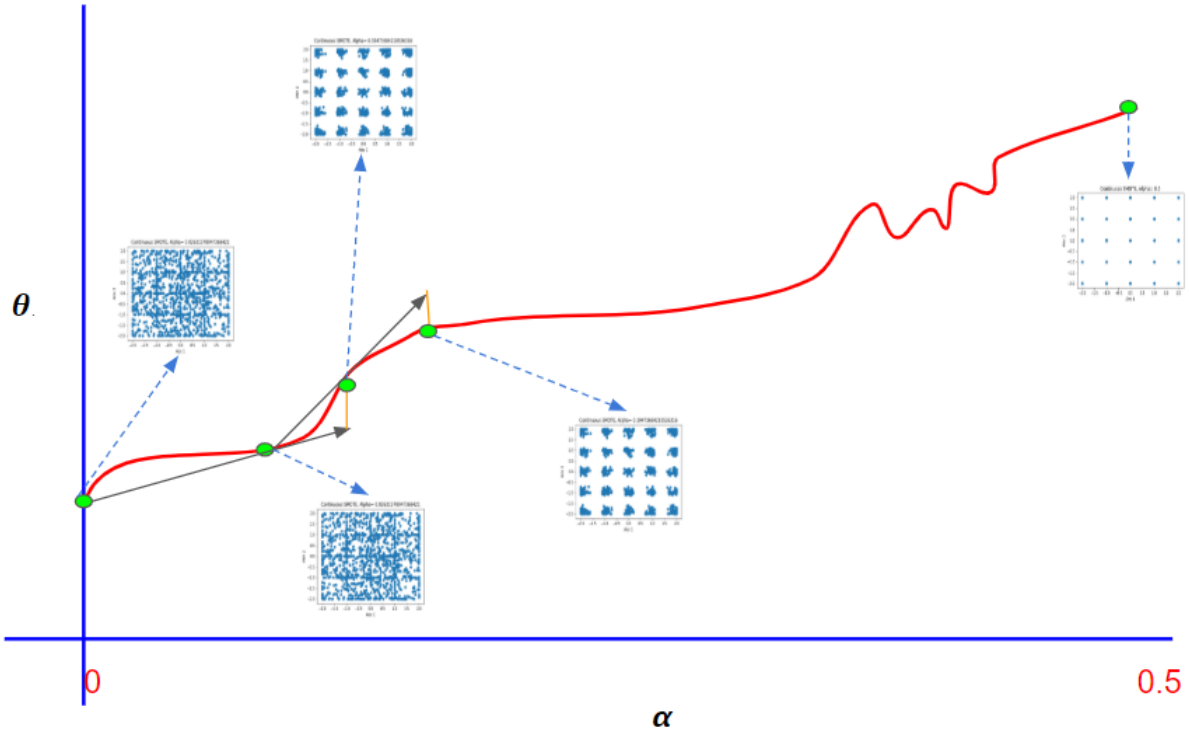


Figure 6.1: Secant approximation on C-SMOTE algorithm's Noise parameter α

6.1 Secant on Noise parameter α

Extending the idea of Secant approximation on λ - θ curve, we can similarly apply the secant approximation on the Noise parameter (α) of C-SMOTE. The idea can be visualized on the $\alpha - \theta$ curve shown in the figure 6.1. Here, instead of the homotopy parameter λ we have noise parameter α. We start from a uniform distribution ($\alpha = 0$) over the support of the true data and then gradually get back to the intrinsic dimensions of the data. Similar to the model continuation (**NPCS**), here we can trace the solutions of all the problems on $\alpha - \theta$ curve 6.1. The idea of using secant approximation in model continuation was to avoid hand picked values of $\Delta\lambda$. Similarly using previous two values on $\alpha - \theta$ curve we can modify the alpha parameter adaptively. In addition to that, we can also reintroduce the re-scaling factor parameter ω_α (similar to ω in 2) which is adaptive to loss. Similar to the NPCS step in the

algorithm 2, we may modify the equation 6.1 to perform secant for data continuation.

$$\begin{aligned}\theta &\leftarrow \theta_{k-1} + \omega_\alpha \cdot \frac{(\theta_{k-1} - \theta_{k-secant\_frequency})}{||(\theta_{k-1} - \theta_{k-secant\_frequency})||_2 + ||(\alpha_{k-1} - \alpha_{k-secant\_frequency})||_2} \\ \alpha &\leftarrow \alpha_{k-1} + \omega_\alpha \cdot \frac{(\alpha_{k-1} - \alpha_{k-secant\_frequency})}{||(\theta_{k-1} - \theta k - secant\_frequency)||_2 + ||(\alpha_{k-1} - \alpha_{k-secant\_frequency})||_2}\end{aligned} \tag{6.1}$$

Hence, we can apply secant on both $\lambda - \theta$ curve for model continuation, and on $\alpha - \theta$ curve for data continuation. Moreover, these methods can be applied simultaneously and may improve the quality and speed of achieving the minimum of an objective function.

There are be more future directions to this research work. First, C-Activation, can be any activation function and in many architectures various activation functions are used, so we would like to work on how to tune multiple activation functions on a single network. Second, the hypothetical $\lambda - \theta$ *curve* we shared, can be of different shapes for which the secant may lead to very bad initialization as shown in figure 6.2, a more advanced method to secant is Pseudo arc-length method [29]. Third, in our experiments, we observed that if we use C-Activation function with random initialization, training wide neural networks results in unstable convergence. For example:- an AE with hidden layers of width 100 or 500 would give extremely high losses, one of the possible reasons of which is our C-activation function is unbounded. PCA provides a more stable initialization with C-activation for any width, but we would like to explore more solutions to address this instability. Fourth, we want to organize our experiments with increasing depth (8,16,32,64,128) so that we can test the ability of the network on very deep networks. Finally, we would like to test our NPCS method on some state-of-art image recognition tasks and check on generalization errors.

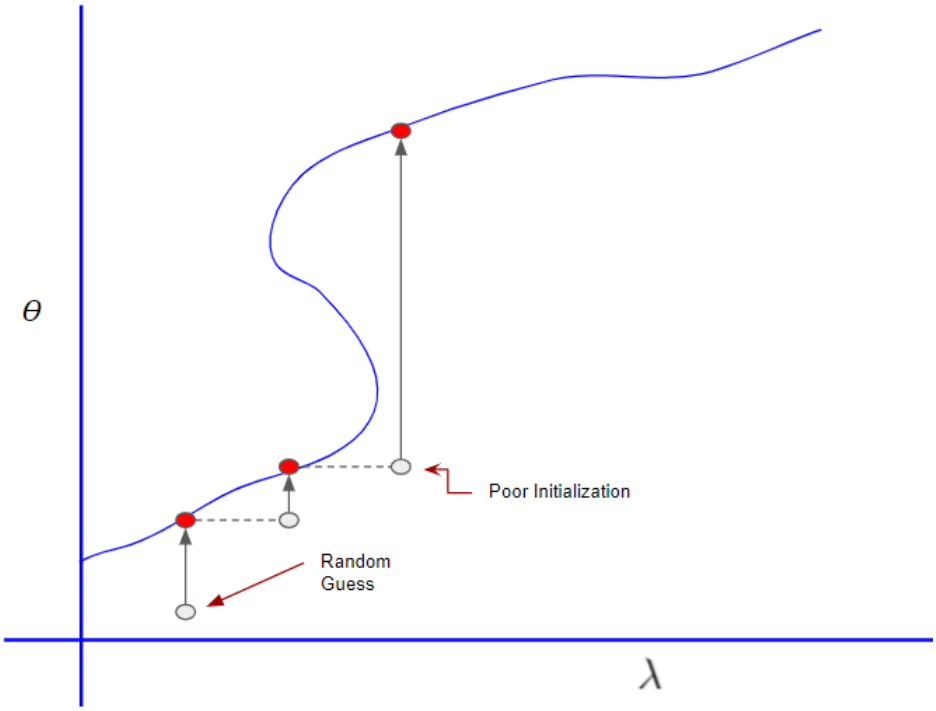


Figure 6.2: Motivation for Pseudo arc-length

Bibliography

[1] E. Allgower and K. Georg. *Introduction to Numerical Continuation Methods*. Society for Industrial and Applied Mathematics, 2003. DOI: 10.1137/1.9780898719154. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898719154. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898719154.

[2] Anima Anandkumar and Rong Ge. "Efficient approaches for escaping higher order saddle points in non-convex optimization". In: *CoRR* abs/1602.05908 (2016). arXiv: 1602.05908. URL: http://arxiv.org/abs/1602.05908.

[3] Martin Arjovsky and Leon Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: *ArXiV* abs/1701.04862 (2017). arXiv: 1701.04862. URL: https://arxiv.org/abs/1701.04862.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN". In: *ArXiV* abs/1701.07875 (2017). arXiv: 1701.07875. URL: https://arxiv.org/abs/1701.07875.

[5] Sanjeev Arora, Nadav Cohen, and Elad Hazan. "On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization". In: *CoRR* abs/1802.06509 (2018). arXiv: 1802.06509. URL: http://arxiv.org/abs/1802.06509.

[6] Peter Auer, Mark Herbster, and Manfred K Warmuth. "Exponentially many local minima for single neurons". In: *Advances in Neural Information Processing Systems 8*. Ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo. MIT Press, 1996, pp. 316–

322. URL: http://papers.nips.cc/paper/1028-exponentially-many-local-minima-for-single-neurons.pdf.

[7] Yoshua Bengio. "Learning Deep Architectures for AI". In: *Foundations and Trends® in Machine Learning* 2.1 (2009), pp. 1–127. ISSN: 1935-8237. DOI: 10.1561/2200000006. URL: http://dx.doi.org/10.1561/2200000006.

[8] Yoshua Bengio, Jerome Louradour, Ronan Collobert, and Jason Weston. *Curriculum Learning*.

[9] Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: Synthetic Minority Over-sampling Technique". In: *CoRR* abs/1106.1813 (2011). arXiv: 1106.1813. URL: http://arxiv.org/abs/1106.1813.

[10] T. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma. "PCANet: A Simple Deep Learning Baseline for Image Classification?" In: *IEEE Transactions on Image Processing* 24.12 (Dec. 2015), pp. 5017–5032. ISSN: 1057-7149. DOI: 10.1109/TIP.2015.2475625.

[11] Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. "The Loss Surface of Multilayer Networks". In: *CoRR* abs/1412.0233 (2014). arXiv: 1412.0233. URL: http://arxiv.org/abs/1412.0233.

[12] J. Chow, L. Udpa, and S. S. Udpa. "Homotopy continuation method for neural networks". In: *1991 Second International Conference on Artificial Neural Networks*. Nov. 1991, pp. 19–23. DOI: 10.1109/ISCAS.1991.176030.

[13] Rodica Constantinescu, Vasile Lazarescu, and Radwan Tahboub. "Geometrical form recognition using "one-step-secant" algorithm in case of neural network". In: ().

[14] Li Deng and Dong Yu. "Deep Learning: Methods and Applications". In: *Foundations and Trends® in Signal Processing* 7.3–4 (2014), pp. 197–387. ISSN: 1932-8346. DOI: 10.1561/2000000039. URL: http://dx.doi.org/10.1561/2000000039.

[15] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12 (July 2011), pp. 2121–2159. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.2021068.

[16] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. "The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training - erhan09a.pdf". In: (). URL: http://jmlr.csail.mit.edu/proceedings/papers/v5/erhan09a/erhan09a.pdf.

[17] Ian J. Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *NIPS* abs/1701.00160 (2017). arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160.

[18] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Networks". In: *NIPS* abs/1406.2661v1 (2014). arXiv: 1406.2661v1. URL: https://arxiv.org/abs/1406.2661v1.

[19] Ian J. Goodfellow and Oriol Vinyals. "Qualitatively characterizing neural network optimization problems". In: *CoRR* abs/1412.6544 (2014). arXiv: 1412.6544. URL: http://arxiv.org/abs/1412.6544.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[21] Çaglar Gülçehre, Marcin Moczulski, Francesco Visin, and Yoshua Bengio. "Mollifying Networks". In: *CoRR* abs/1608.04980 (2016). arXiv: 1608.04980. URL: http://arxiv.org/abs/1608.04980.

[22] Robert Hecht-Nielsen. "Theory of the backpropagation neural network". In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[23] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507. ISSN: 0036-8075. DOI: 10.1126/science.1127647. eprint: http://science.sciencemag.org/content/313/5786/504.full.pdf. URL: http://science.sciencemag.org/content/313/5786/504.

[24] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". In: *IEEE Signal Processing Magazine* 29.6 (Nov. 2012), pp. 82–97. ISSN: 1053-5888. DOI: 10.1109/MSP.2012.2205597.

[25] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Rmsprop: Divide the gradient by a running average of its recent magnitude". In: *Neural networks for machine learning, Coursera lecture 6e* (2012).

[26] Daniel Jiwoong Im, Michael Tao, and Kristin Branson. "An Empirical Analysis of Deep Network Loss Surfaces". In: *CoRR* abs/1612.04010 (2016). arXiv: 1612.04010. URL: http://arxiv.org/abs/1612.04010.

[27] Prateek Jain and Purushottam Kar. "Non-convex Optimization for Machine Learning". In: *Foundations and Trends® in Machine Learning* 10.3-4 (2017), pp. 142–336. ISSN: 1935-8237. DOI: 10.1561/2200000058. URL: http://dx.doi.org/10.1561/2200000058.

[28] R.E. Johnson and F.L. Kiokemeister. *Calculus, with analytic geometry*. Allyn and Bacon, 1964. URL: https://books.google.com/books?id=X4%5C_UAQAACAAJ.

[29] H. B. Keller. "Numerical solution of bifurcation and nonlinear eigenvalue problems". In: *Applications of Bifurcation Theory*. Ed. by P. H. Rabinowitz. New York: Academic Press, 1977, pp. 359–384.

[30] D. P Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *ArXiv e-prints* (Dec. 2013). arXiv: 1312.6114 [stat.ML].

[31] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[32] Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. "Data-dependent Initializations of Convolutional Neural Networks". In: *CoRR* abs/1511.06856 (2015). arXiv: 1511.06856. URL: http://arxiv.org/abs/1511.06856.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems*. P. 2012.

[34] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* (2015). URL: https://www.nature.com/articles/nature14539.

[35] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*. 2015.

[36] Raghu Meka, Prateek Jain, Constantine Caramanis, and Inderjit S. Dhillon. "Rank Minimization via Online Learning". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: ACM, 2008, pp. 656–663. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390239. URL: http://doi.acm.org/10.1145/1390156.1390239.

[37] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Curran Associates, Inc.,

2013, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[38] Hossein Mobahi. "Training Recurrent Neural Networks by Diffusion". In: *CoRR* abs/1601.04114 (2016). arXiv: 1601.04114. URL: http://arxiv.org/abs/1601.04114.

[39] Hossein Mobahi and John W. Fisher III. "On the Link Between Gaussian Homotopy Continuation and Convex Envelopes". In: *Lecture Notes in Computer Science (EMM-CVPR 2015)*. Springer, 2015. URL: http://people.csail.mit.edu/hmobahi/pubs/gaussian_convenv_2015.pdf.

[40] Hossein Mobahi and John W. Fisher III. "A Theoretical Analysis of Optimization by Gaussian Continuation". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 1205–1211. ISBN: 0-262-51129-0. URL: http://dl.acm.org/citation.cfm?id=2887007.2887174.

[41] Klaus Nordhausen. "The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition by Trevor Hastie, Robert Tibshirani, Jerome Friedman". In: *International Statistical Review* 77.3 (), pp. 482–482. DOI: 10.1111/j.1751-5823.2009.00095_18.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1751-5823.2009.00095_18.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1751-5823.2009.00095_18.x.

[42] *Online Resource*. URL: http://yann.lecun.com/exdb/mnist/.

[43] Hengyue Pan and Hui Jiang. "Annealed Gradient Descent for Deep Learning". In: *UAI*. 2015.

[44] Werner C. Rheinboldt. "Numerical analysis of continuation methods for nonlinear structural problems". In: *Computers Structures* 13.1 (1981), pp. 103–113. ISSN: 0045-7949. DOI: https://doi.org/10.1016/0045-7949(81)90114-0. URL: http://www.sciencedirect.com/science/article/pii/0045794981901140.

[45] Itay Safran and Ohad Shamir. "On the Quality of the Initial Basin in Overspecified Neural Networks". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 774–782. URL: http://dl.acm.org/citation.cfm?id=3045390.3045473.

[46] Mathias Seuret, Michele Alberti, Rolf Ingold, and Marcus Liwicki. "PCA-Initialized Deep Neural Networks Applied To Document Image Analysis". In: *CoRR* abs/1702.00177 (2017). arXiv: 1702.00177. URL: http://arxiv.org/abs/1702.00177.

[47] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. "Amortised MAP Inference for Image Super-resolution". In: *CoRR* abs/1610.04490 (2016). arXiv: 1610.04490. URL: http://arxiv.org/abs/1610.04490.

[48] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". In: *Science* 290.5500 (2000), pp. 2319–2323. ISSN: 0036-8075. DOI: 10.1126/science.290.5500.2319. eprint: http://science.sciencemag.org/content/290/5500/2319.full.pdf. URL: http://science.sciencemag.org/content/290/5500/2319.

[49] Daphna Weinshall and Gad Cohen. "Curriculum Learning by Transfer Learning: Theory and Experiments with Deep Networks". In: *CoRR* abs/1802.03796 (2018). arXiv: 1802.03796. URL: http://arxiv.org/abs/1802.03796.

[50] Lilian Weng. *From GAN to WGAN*. 2017. URL: https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html.

[51] Sitao Xiang and Hao Li. "On the Effects of Batch and Weight Normalization in Generative Adversarial Networks". In: *ArXiV* abs/1704.03971 (2017). arXiv: 1704.03971. URL: https://arxiv.org/abs/1704.03971.

[52] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[53] Reza Zadeh. *The hard thing about deep learning*. 2016. URL: https://www.oreilly.com/ideas/the-hard-thing-about-deep-learning.

[54] Chong Zhou and Randy C Paffenroth. "Anomaly detection with robust deep autoencoders". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2017, pp. 665–674.