Optimization, Robustness and Risk-Sensitivity in Machine Learning: A Dynamical Systems Perspective

Kamil Nar

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-154
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-154.html

August 13, 2020

Optimization, Robustness and Risk-Sensitivity in Machine Learning: A Dynamical Systems Perspective

by

Kamil Nar

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor S. Shankar Sastry, Chair
Professor Murat Arcak
Professor Anil Aswani

Summer 2020

Optimization, Robustness and Risk-Sensitivity in Machine Learning:
A Dynamical Systems Perspective

Abstract

Optimization, Robustness and Risk-Sensitivity in Machine Learning: A Dynamical Systems Perspective

by

Kamil Nar

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor S. Shankar Sastry, Chair

Training models that are multi-layer or recursive, such as neural networks or dynamical system models, entails solving a nonconvex optimization problem in machine learning. These nonconvex problems are usually solved with iterative optimization algorithms, such as the gradient descent algorithm or any of its variants. Once an iterative algorithm is involved, the dynamics of this algorithm will become critical in determining the specific solution obtained for the optimization problem. In this dissertation, we use tools from nonlinear and adaptive control theory to analyze and understand how the dynamics of the training procedures affects the solutions obtained, and we synthesize new methods to facilitate optimization, to provide robustness for the trained models, and to help explain observed outcomes in a more accurate way.

By studying Lyapunov stability of the fixed points of the gradient descent algorithm, we show that this algorithm can only yield a solution from a bounded class of functions when training multi-layer models. We establish a relationship between the learning rate of the algorithm and the Lipschitz constant of the function estimated by the multi-layer model. We also show that keeping every layer of the model close to the identity operation boosts the stability of the optimization algorithm and allows the use of larger learning rates.

We use a classical concept in system identification and adaptive control, namely, the persistence of excitation, to study the robustness of multi-layer models. We show that when trained with the gradient descent algorithm, robust estimation of the unknown parameters in a multi-layer model requires not only the richness of the training data, but also the richness of the hidden-layer activations throughout the training procedure. We derive necessary and sufficient richness conditions for the signals in each layer of the model, and we show that these conditions are usually not satisfied by models that have been naively trained with the gradient descent algorithm, since the signals in their hidden layers become low-dimensional during training. By revisiting the common regularization methods for single-layer models, reinterpreting them in terms of enhancing the richness of the training data, and drawing an analogy for multi-layer models, we design a training mechanism that provides the required richness for the signals in the hidden-layers of multi-layer models. This training procedure leads to similar margin distributions for the training and test data for a neural network trained for a classification task, indicating its effectiveness as a regularization method.

We study the dynamics of the gradient descent algorithm on dynamical systems as well. We show that when learning the unknown parameters of an unstable dynamical system, the observations taken from the system at different times influence the dynamics of the gradient descent algorithm in substantially different degrees. In particular, the observations taken from the system near the end of the time horizon imposes an exponentially strict constraint on the learning rate that could be used for the gradient descent algorithm, whereas such small learning rates cannot recover the stable modes of the system. We show that warping the observations of the system in a particular way and creating risk-sensitivity in the observations remedies this imbalance and allows learning both the stable and the unstable modes of a linear dynamical system.

The results in this dissertation lay out the strong connection between the machine learning problems involving nonconvex optimization and the classical tools in nonlinear and adaptive control theory. While analyses with Lyapunov stability and persistence of excitation are able to help understand and enhance the machine learning models trained with iterative optimization algorithms, the major effect of altering the training dynamics on multi-layer machine learning models indicates the potential for improving system identification for dynamical systems by designing alternative loss functions.

To my parents, my sister Iraz, and my brother Ilgın.

Contents

List of Figures

Acknowledgments

First and foremost, I thank my advisor Shankar Sastry. His vision for research has molded my research interests over the past six years, and his enthusiasm has been a constant source of motivation during this time. It was his guidance that has helped me build the connection between the classical tools in adaptive control and contemporary problems in machine learning, and this connection has formed the backbone of this dissertation.

I thank my thesis and qualification examination committee, Murat Arcak, Anil Aswani and Peter Bartlett, for their inputs and their support while working on this dissertation.

I am fortunate to have worked with Tamer Başar during my years at the University of Illinois. This dissertation would not have been possible without the depth of understanding of control theory and dynamical systems that I was exposed to while working with him at Illinois.

I am thankful for having had the support and encouragement of Lillian Ratliff and Roy Dong in my first years at Berkeley. Our joint work with Lillian on prospect theory has become one of the chapters in this dissertation.

Having the chance to teach a class on convex optimization and random processes undoubtedly gave a deeper understanding on these subjects. It was pleasure to work with great professors while teaching these classes: Kannan Ramchandran, Gireeja Ranade, Alex Bayen, and Laurent El Ghaoui.

I am also thankful for having had the chance to work with Yuan Xue and Andrew Dai during my internship at Google Health. They introduced me to various interesting problems that were being studied at Google, and they encouraged me to bring my own perspective to address these problems. Our joint work produced as a result of this internship has also become one of the chapters in my dissertation.

I also want to acknowledge the grants that financially supported my doctoral research. I acknowledge the support received by the U.S. Office of Naval Research MURI grant N00014-16-1-2710. I also acknowledge the support received by the Defense Advanced Research Projects Agency award number FA8750-18-C-0101 for the design of high-confidence learning-enabled systems. I am grateful for having worked along with Radha Poovendran and Claire Tomlin on these grants.

I also thank the members of my research group at Berkeley. The feedback I received from Eric Mazumdar and Tyler Westenbroek has been very valuable in most projects I have worked on. I also thank Oladapo Afolabi, Joseph Menke, Dexter Scobee, Joshua Achiam, David McPherson, Dorsa Sadigh, Jaime Fernandez Fisac, Sam Burden, and all other members of the Semiautonomous group.

I also want to thank Shirley Salanio, Jessica Gamble and Yovana Gomez as they have helped me navigate the graduate school throughout the past six years.

I am lucky to have had the support of great friends during my years at Berkeley. I cannot thank enough to Orhan Ocal and Tugce Gurek for their constant support and encouragement for the past six years. I also thank Ebru Toprak, Burak Eminoglu, Soner Sonmezoglu and Alper Ozgurluk.

Lastly, I am grateful for my parents, my sister Iraz, and my brother Ilgın. I would not have made it this far if it was not for their love, support and encouragement.

Chapter 1

Introduction

A large class of supervised and unsupervised learning tasks in machine learning involves solving an optimization problem (Hastie et al., 2009). For example, training a model for a supervised learning task usually entails minimizing an objective defined as the empirical risk:

$$\mathcal{R}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f_\theta(x_i)),$$

where f_θ is the model being used, θ is the parameters of the model, ℓ is the loss function that quantifies the mismatch between the true labels and the estimated labels, and $\{(x_i, y_i)\}_{i=1}^N$ is the training data. Such optimization problems are in general solved with iterative algorithms, such as the gradient descent algorithm, Newtons methods, or any of their variants.

When an iterative optimization algorithm is involved, there are a set of questions that naturally arise:

1. Is there a fixed point of the algorithm?

2. If there is a fixed point of the algorithm, is it unique and under what conditions does the algorithm converge to it?

3. If there are multiple fixed points, which point does the algorithm converge to?

If the optimization problem is strictly convex over a bounded region, the answers to these questions are clear: there is a unique fixed point for the algorithm, and the algorithm converges to this unique point from every initialization. This is the case, for example, when the parameters of f_θ appear linearly inside the model and the loss function ℓ is strictly convex.

On the other hand, when the objective function is nonconvex, as in the case of training neural networks or dynamical system models, there will typically be multiple local optima, and each of these local optima will be a fixed point for the optimization algorithm. In this case, the initialization and the dynamics of the optimization algorithm will determine the solution obtained. Consequently, properties of the model learned will depend on the dynamics of the optimization algorithm, and conversely, the dynamics of the algorithm will be influenced by the model to be estimated.

In this dissertation, we look closely into the relationship between dynamical systems, machine learning models, optimization, robustness, and risk-sensitivity.

1.1 Optimization for Learning

As a model whose training requires solving a nonconvex optimization problem, consider a neural network. The depth of the neural network determines the size of the class of functions that it can represent. As the depth is increased, this class of functions expands provided that the new layers are able to express the identity mapping. Therefore, the minimum training error that can be achieved by a network diminishes as its depth is increased. However, the training error of most neural networks degrades in practice once the number of layers exceeds a certain value; and deeper networks start to perform worse than their shallower counterparts, as shown in Figure 1.1 (He et al., 2015). This deterioration in the training error with increased depth indicates a problem with the method used for training the neural network; namely, a problem with the convergence of the gradient descent algorithm.

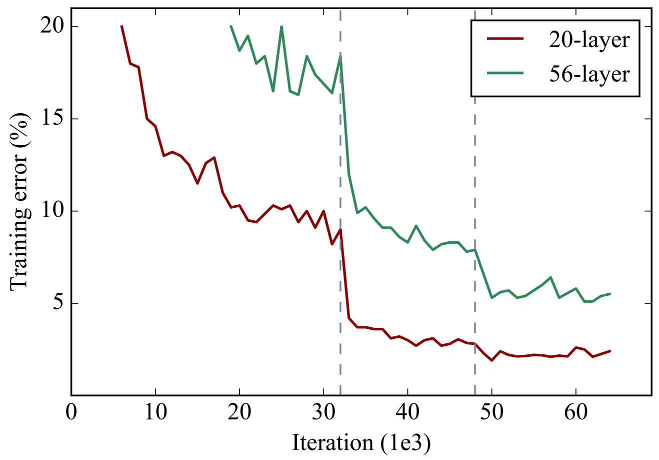


Figure 1.1: Training error of a 20-layer and a 56-layer neural network on CIFAR-10 dataset. The deeper network leads to a significantly larger training error. Step size of the gradient descent is divided by 10 at iterations 32k and 48k; these instants are shown with the dashed lines. Right after the step size is decreased, the training error plummets. It cannot be the case that the parameters are slowly converging to a local optimum before the step size is changed, nor can they be stuck near a local optimum or a saddle point, because decreasing the step size would have further slowed down the convergence in those cases. This behavior can only be explained by the fact that the initial step size is too large for certain regions in the parameter space and the parameters keep oscillating around a local optimum until the step size is changed. Once the step size is decreased, the magnitude of the oscillations around the local optimum diminishes, and so does the training error. The figure is adapted from (He et al., 2015).

When gradient descent is used to minimize a function, say $f : \mathbb{R}^n \to \mathbb{R}$, it leads to a discrete-time dynamical system:

$$x[k+1] = x[k] - \delta \nabla f(x[k]), \tag{1.1}$$

where $x[k]$ is *the state* of the system, which consists of the parameters updated by the algorithm, and δ is the step size of the algorithm. Every fixed point of the system (1.1) is called *an equilibrium* of the system, and they correspond to the critical points of the function f.

Unless f is a quadratic function of the parameters, the system described by (1.1) is either a nonlinear system or a hybrid system that switches from one dynamics to another over time. Consequently, the system (1.1) can exhibit behaviors that are typically observed in nonlinear and hybrid systems, such as convergence to an orbit but not to a fixed point, or dependence of the convergence on the initialization. The step size of the gradient descent algorithm has a critical effect on these behaviors, as shown in the following two examples.

Example 1. Convergence to a periodic orbit: Consider the continuously differentiable and convex function $f(x) = \frac{2}{3}|x|^{3/2}$, which has a unique local minimum at the origin. The gradient descent algorithm on this function yields

$$x[k+1] = \begin{cases} x[k] - \delta\sqrt{x[k]}, & x[k] \geq 0, \\ x[k] + \delta\sqrt{-x[k]}, & x[k] < 0. \end{cases}$$

As expected, the origin is the only equilibrium of this system. Interestingly, however, $x[k]$ converges to the origin only when the initial state $x[0]$ belongs to a countable set $\mathcal{S}$:

$$\mathcal{S} = \left\{ 0, \delta^2, -\delta^2, \frac{3+\sqrt{5}}{2}\delta^2, -\frac{3+\sqrt{5}}{2}\delta^2, \dots \right\}.$$

For all other initializations, $x[k]$ converges to an oscillation between $\delta^2/4$ and $-\delta^2/4$. This implies that, if the initial state $x[0]$ is randomly drawn from a continuous distribution, then almost surely, $x[k]$ does not converge to the origin, yet $|x[k]|$ converges to $\delta^2/4$. In other words, with probability 1, the state $x[k]$ does not converge to a fixed point, such as a local optimum or a saddle point, even though the estimation error converges to a finite non-optimal value.

Example 2. Dependence of convergence on the initialization: Consider the function $f(x) = x^L$ where $L \in \mathbb{N}$ is an even number larger than 2. The gradient descent results in the system

$$x[k+1] = x[k] - \delta L x[k]^{L-1}.$$

The state $x[k]$ converges to the origin if the initial state satisfies $x[0]^{L-2} < (2/L\delta)$ and $x[k]$ diverges if $x[0]^{L-2} > (2/L\delta)$.

These two examples demonstrate:

1. the convergence of training error does not imply the convergence of the gradient descent algorithm to a local optimum or a saddle point,

2. the step size determines the magnitude of the oscillations if the algorithm converges to an orbit but not to a fixed point,

3. the step size influences the convergence of the algorithm differently for each initialization.

Note that these are consequences of the nonlinear dynamics of the algorithm and not of the convexity or nonconvexity of the function to be minimized. While both of the functions used in the examples are convex, the identical behaviors are observed during the minimization of nonconvex cost functions of neural networks as well. In fact, only these behaviors can provide a satisfactory explanation for the phenomenon observed in Figure 1.1: right after the step size of the algorithm is decreased, the training error plummets. It cannot be the case that the parameters are slowly converging to an equilibrium right before the step size is changed, nor can they be stuck near a local optimum or a saddle point, because if either were the case, decreasing the step size would have further slowed down the convergence. These sharp falls can only be explained by the fact that the initial step size is too large for some regions in the parameter space, and the parameters are oscillating around a local optimum right before the step size is changed. Once the step size is decreased, the radius of the oscillations around the equilibrium point diminishes, the distance to the equilibrium point in the parameter space falls sharply, and consequently, so does the training error.

While training a deep neural network, the dynamical system created by the gradient descent will usually have multiple equilibria, which coincide with the critical points of the training cost function. Convergence to these equilibria is in general affected unequally by the step size. For example, for a given step size, the algorithm might be able to converge to a subset of the local optima but not to the others independent of the initializations. Therefore, the step size also plays a critical role in understanding why some solutions are more likely to be obtained instead of the others when the gradient descent algorithm is used.

1.2 Robustness for Learning

State-of-the-art neural networks provide high accuracy for plenty of machine learning tasks, but their performance has proven vulnerable to small perturbations in their inputs (Szegedy et al., 2013). This lack of robustness prohibits the use of neural networks in safety-critical applications such as computer security, control of cyber-physical systems, self-driving cars, and medical prediction and planning (Kurakin et al., 2016).

Given the combination of large number of training parameters, nonlinearity of the model, nonconvexity of the optimization problem used for training, involvement of an iterative algorithm for optimization, and high-dimensionality of the commonly used data sets, it is challenging to pinpoint what particularly causes the lack of robustness in neural networks. It has been speculated that the high nonlinearity of neural networks gives rise to this vulnerability (Szegedy et al., 2013), but the fact that support vector machines with nonlinear feature mappings remain much more robust against similar perturbations readily invalidates this claim (Goodfellow et al., 2015).

It has been observed that the lack of robustness is not a consequence of the depth of the network; even networks with few layers are shown to produce drastically different outputs for almost identical inputs. Comparison of this observation with the robustness of support vector machines has led to the belief that neural networks might in fact not be introducing sufficient level of nonlinearity to execute the learning tasks involved — since shallow networks function as an affine mapping in most of their domains, whereas most feature mappings used in support vector machines are unarguably nonlinear (Goodfellow et al., 2015). Although this belief still prevails, the level of nonlinearity is not the only aspect that neural networks and support vector machines differ in. There are many other factors these two structures do not share in common, and these factors might also account for their dissimilarity in robustness.

One of the important yet overlooked differences between neural networks and support vector machines is the first-line loss functions used in their training. For classification tasks, neural networks are almost always trained with the cross-entropy loss, whereas training support vector machines involves the hinge loss function (Hastie et al., 2009). This raises the question whether the choice of training loss function, and in particular the cross-entropy loss function, is one of the factors leading to the lack of robustness in neural networks. To provide a preliminary answer to this question, we train a two-layer neural network for a binary classification task with two different loss functions: the cross-entropy loss and the squared-error loss. Figure 1.2 demonstrates the decision boundaries of the network trained with the two loss functions for different initializations. Even though the network architecture, the training data, and the optimization algorithm are kept identical, Figure 1.2 shows that the network trained with the cross-entropy loss consistently has a much smaller margin than the network trained with the squared-error loss, and hence, it is less robust.

Figure 1.2 confirms that at least some ingredients of neural network training procedure have an influence on the robustness of networks. Naturally, the next question to ask is whether the identical training procedure could lead to a similar vulnerability in models that are different than neural networks. Figure 1.3 illustrates two examples of a linear classifier trained with the cross-entropy loss function and the gradient descent algorithm. It appears that the decision boundary of a linear classifier could also have an extremely poor margin when it is trained in exactly the same way as neural networks are trained.

To understand the observations in Figure 1.2 and Figure 1.3, we will first look into a concept called persistency of excitation, and then explain how it relates to training neural networks.

1.2.1 Persistency of Excitation

Consider the training of a linear model with the squared-error loss:

$$\min_{\theta} \sum\nolimits_{i\in\mathcal{I}} \|x_i^\top \theta - y_i\|_2^2,$$

where $\{x_i\}_{i\in\mathcal{I}}$ is the set of training data in $\mathbb{R}^n$ and $\{y_i\}_{i\in\mathcal{I}}$ is the set of target values in $\mathbb{R}$. This problem is convex, and if the set $\{x_i\}_{i\in\mathcal{I}}$ spans whole $\mathbb{R}^n$, the optimal value for θ will be unique. We can use the gradient descent algorithm to find this optimal value:

$$\theta \leftarrow \theta - \delta \sum\nolimits_{i\in\mathcal{I}} x_i(x_i^\top \theta - y_i), \tag{1.2}$$

where δ is some fixed step size. As long as δ is sufficiently small, the parameter estimate is guaranteed to converge to the optimal value of θ.

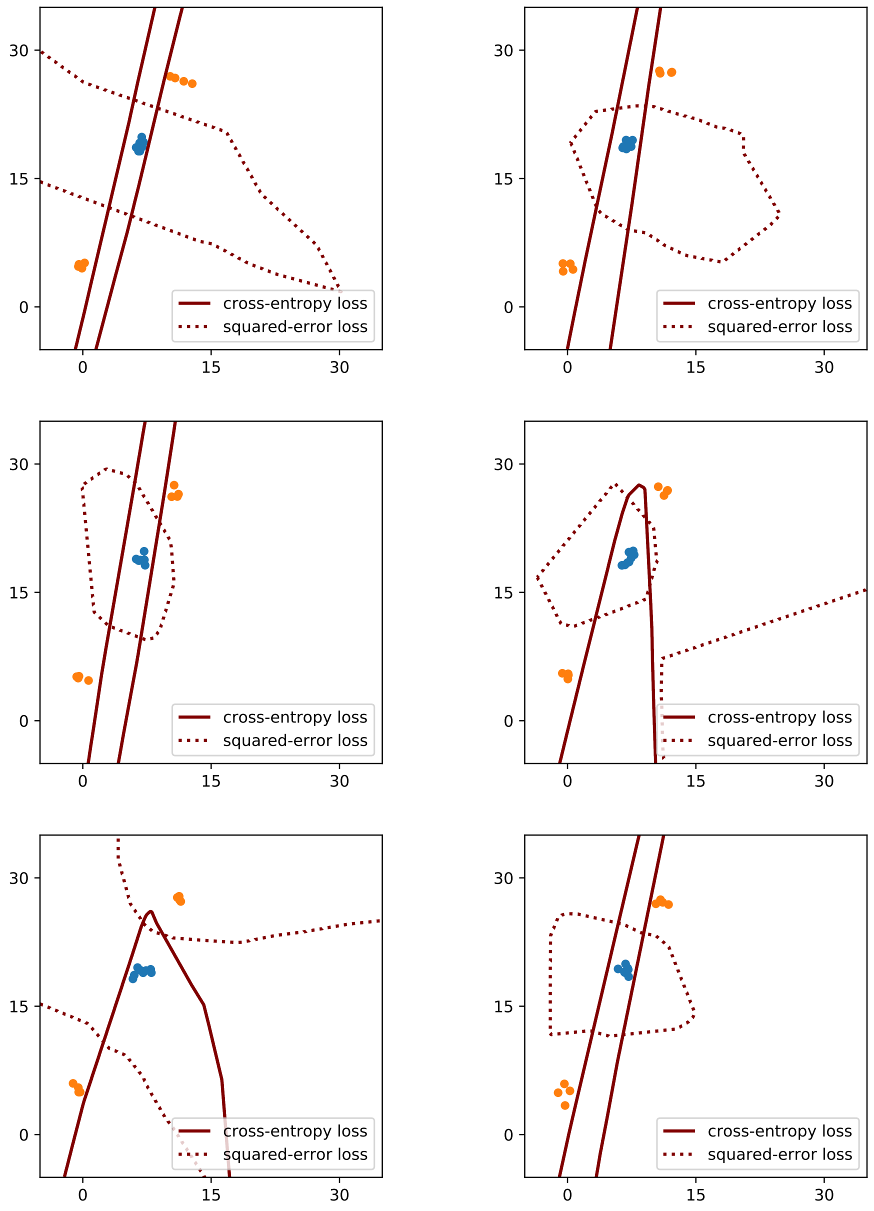


Figure 1.2: The decision boundaries of a two-layer network trained with the cross-entropy loss and the squared-error loss. The task is binary classification in $\mathbb{R}^2$; the orange clusters form one class and the blue cluster represents the other class. Each plot shows the decision boundaries for a different initialization, corresponding to a different random seed. The network architecture, the optimization algorithm, and the training data are identical within each plot; only the training loss functions are different. The networks trained with the cross-entropy loss consistently have a substantially poorer margin.

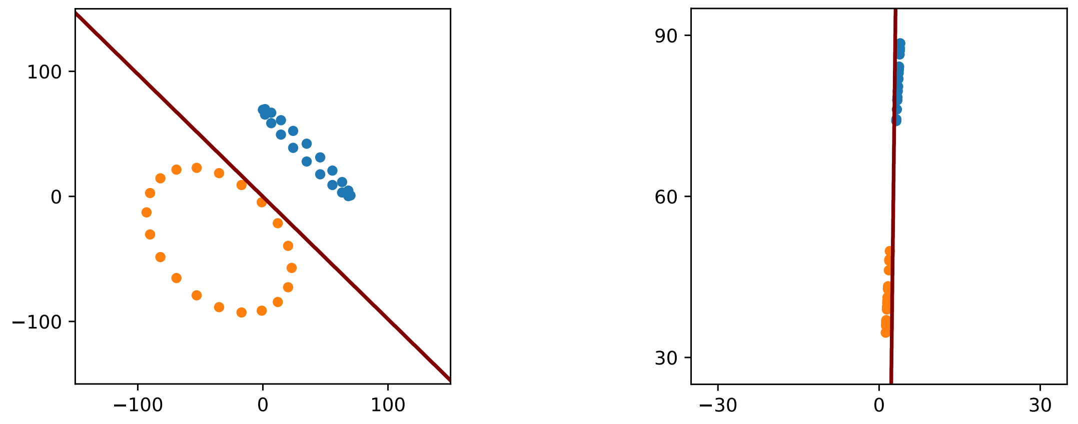


Figure 1.3: Two different data sets on which a linear classifier is trained by minimizing the cross-entropy loss using the gradient descent algorithm. The solid lines represent the decision boundary of the classifier, which lie very close to the training data. Standard ingredients of neural network training seem to cause extremely poor margin and lack of robustness for linear classifiers as well.

Now, assume for some reason the measurements are attenuated at each iteration of the gradient descent algorithm. That is, the update rule (1.2) is followed at each iteration by

$$(x_i, y_i) \leftarrow (\alpha x_i, \alpha y_i) \quad \forall i \in \mathcal{I},$$

for some $\alpha \in (0, 1)$. In this case, the gradient descent algorithm will lose its ability to find the true value of θ. For every initialization, the algorithm will still converge, the error term will still become zero, but the parameters will not necessarily converge to the optimal θ.

This might seem like a contrived example, but it is a fundamental problem that arises in various settings, such as identification and adaptive control of dynamical systems, optimization with stochastic gradient methods, and exploration in multi-armed bandits.

- **System identification and adaptive control:** Consider a discrete-time linear dynamical system:

$$x_{t+1} \leftarrow \theta x_t + u_t \quad \forall t \in \mathbb{N}, \tag{1.3}$$

where $x_t \in \mathbb{R}$ and $u_t \in \mathbb{R}$ are the state and the input of the system at time t, and $\theta \in (0, 1)$ is the unknown parameter of the system. Assume an estimator system of the form

$$\hat{x}_{t+1} \leftarrow \hat{\theta} \hat{x}_t + u_t \quad \forall t \in \mathbb{N}$$

is used to identify the value of θ by decreasing the distance between x_t and $\hat{x}_t$ over time with the gradient descent algorithm. Then both x_t and $\hat{x}_t$ can decay to zero while $\hat{\theta}$ converges to some value different than θ (Kumar and Varaiya, 1986).

- **Stochastic gradient methods:** Consider a convex function $f(\theta)$, which can be decomposed as $\sum_{i\in\mathcal{I}} f_i(\theta)$, where each f_i is also a convex function. Assume we use the following stochastic gradient method with a fixed step size δ:

  1. Randomly choose $i \in \mathcal{I}$,
  2. $\theta_{t+1} \leftarrow \theta_t - \delta \frac{\partial}{\partial\theta} f_i(\theta)$,
  3. Scale down $f(\theta)$; in other words, set $f_i \leftarrow \alpha_t f_i$ for all $i \in \mathcal{I}$ for some $\alpha_t \in (0,1)$,
  4. Return to 1.

  This algorithm does not necessarily converge to the optimal value for θ if α_t attenuates $f(\theta)$ too quickly (Bottou et al., 2018).

- **Multi-armed bandits:** Consider a two-armed bandit problem where the arms produce the independent and identically distributed random processes $\{X_t\}_{t\in\mathbb{N}}$ and $\{\tilde{X}_t\}_{t\in\mathbb{N}}$ with distinct means; that is, $\mathbb{E}X_t \neq \mathbb{E}\tilde{X}_t$. Assume we try to solve

$$\max_{\{d_t\}_{t\in\mathbb{N}}\in\{0,1\}^{\mathbb{N}}} \mathbb{E}\left(\lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\left(d_t X_t + (1-d_t)\tilde{X}_t\right)\right)$$

  by using a causal feedback mechanism $f : \mathbb{N}^2 \times \mathbb{R}^2 \mapsto \{0,1\}$ that prescribes a decision at time $t \in \mathbb{N}$ as:

$$d_t = f\left(t, \sum\nolimits_{t'=1}^{t-1} d_{t'}, \sum\nolimits_{t'=1}^{t-1} d_{t'}X_{t'}, \sum\nolimits_{t'=1}^{t-1}(1-d_{t'})\tilde{X}_{t'}\right).$$

  If the feedback policy f decreases the frequency of pulling the arm with the lower average of observed rewards too quickly, then the algorithm can fail to discover the arm with the higher mean (Bubeck and Cesa-Bianchi, 2012).

In these three examples, there is an input, a signal, or an action that excites the unknown parameters; that is, something that forces the parameters to release some information about themselves. We observe that when this excitation decays too quickly, the online learning algorithm cannot receive necessary amount of information about the parameters and fail to estimate them correctly. This leads to a concept called *persistency of excitation*. For online algorithms to learn the unknown parameters of a model correctly, the signals interacting with the parameter estimates need to remain persistently exciting during the estimation procedure (Kumar and Varaiya, 1986; Sastry and Bodson, 1989). If this persistency of excitation condition is not satisfied, the error terms inside the learning algorithm can become zero even if the parameters converge to a wrong value or do not converge at all.

1.2.2 Connection Between Persistency of Excitation and Neural Networks

During training of a feedforward neural network, the iterative optimization algorithm involved creates the dynamics of an online learning problem as shown in Figure 1.4. The function f_θ

represents the true mapping with the ideal parameters, whereas $f_{\hat{\theta}}$ denotes the estimate obtained with the gradient descent algorithm. Assume that the neural network has L layers, and it is described as

$$f_\theta(x) = f_{\theta_L} \circ f_{\theta_{L-1}} \circ \cdots \circ f_{\theta_1}(x),$$

where f_{θ_k} represents the operation the k-th layer of the network performs and θ_k stands for the parameters of that particular layer for each $k \in [L] = \{1, 2, \ldots, L\}$. Given a set of training data $\{x_i\}_{i\in\mathcal{I}}$, the parameters of the k-th layer, θ_k, are excited by the signals of the preceding layer throughout the training, which are given as $\{f_{\theta_{k-1}} \circ \cdots \circ f_{\theta_1}(x_i)\}_{i\in\mathcal{I}}$. Note that no matter how large the training data set is, there is no strong reason for these signals in the intermediate layers to remain persistently exciting during training, and we will show that they indeed do not remain persistently exciting with standard training procedures. The consequence is that even if the training error converges to zero, the estimated parameters may not be the same as the ideal parameters of the true mapping, and they may not predict the output of the network accurately for unseen data.

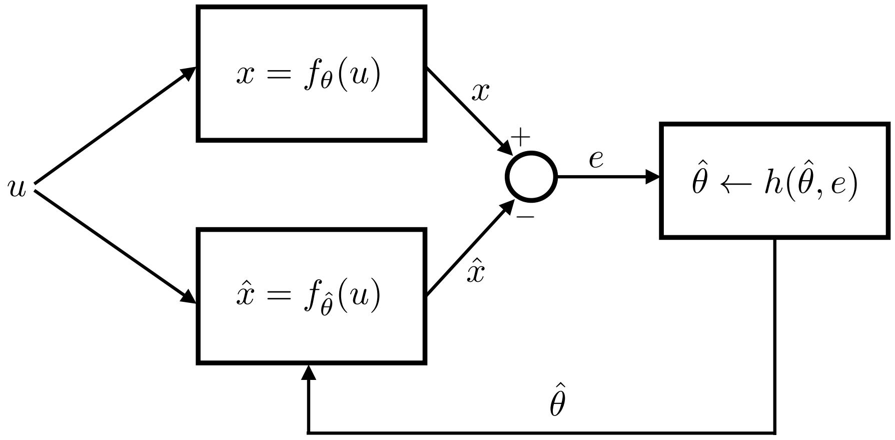

u
$x = f_\theta(u)$
x
$+$
e
$-$
$\hat{x}$
$\hat{x} = f_{\hat{\theta}}(u)$
$\hat{\theta} \leftarrow h(\hat{\theta}, e)$
$\hat{\theta}$

Figure 1.4: Closed-loop representation of the dynamics of neural network training. Function f_θ represents the true mapping that we want to estimate, while $f_{\hat{\theta}}$ represents the mapping estimated via the gradient descent algorithm. Signal e denotes the training error of the model estimated, and function h gives the update rule for the parameter estimates.

We will see in the following chapters that there exist some conditions on the training data and the hidden-layer activations which ensure the persistent excitation of the network parameters. Given these conditions, convergence of training error to zero will imply the convergence of the estimate $\hat{\theta}$ to an optimal parameter θ that will induce the network to produce similar outputs for similar inputs.

The concept of persistency of excitation already provides an interpretation of Figure 1.2 and Figure 1.3. Different training loss functions give rise to different dynamics for the gradient descent algorithm; hence, they involve different conditions on the training data for the persistent excitation of the parameters — some of which are easier to satisfy than the others. Moreover, for some loss functions, even the linear models are not exempted from the restrictiveness of these conditions.

1.3 Risk-Sensitivity for Learning

Risk-sensitivity is an important concept in microeconomics and finance (Rubinstein, 2012). When faced with uncertainty, the choices or actions of an agent could be risk-averse, risk-seeking or risk-neutral. Given two choices with equal expected rewards but with different uncertainties, a risk-averse agent chooses the choice with the less uncertainty, whereas a risk-seeking agent chooses the one with the more uncertainty.

If the value of different outcomes for an agent is described by a utility function, the convexity of this function reflects the risk-sensitivity of the agent. For risk-averse agents, this function is concave; for risk-seeking agents, it is convex; and for risk-neutral agents, it is linear.

The classical decision models in economics, such as the expected utility theory (Rubinstein, 2012), assume that the values an agent assigns to different outcomes is not affected by the way the outcomes are presented to the agent, and therefore, the risk-sensitivity of the agent is definite. On the other hand, there exist models in behavioral economics, such as the prospect theory (Kahneman and Tversky, 1979), which posit that the risk-sensitivity of the agent is affected by how the choices are presented to the decision-makers and how the choice problem is framed by the decision-makers with respect to their references. For example, the prospect theory argues that the risk attitudes of people are also influenced by their reference point, and people tend to act more risk-averse when making a choice between positive outcomes and more risk-seeking when making a choice between negative outcomes. Consequently, the utility or the value function of the agent becomes concave in the region for positive outcomes and convex in the region for negative outcomes, as demonstrated in Figure 1.5.

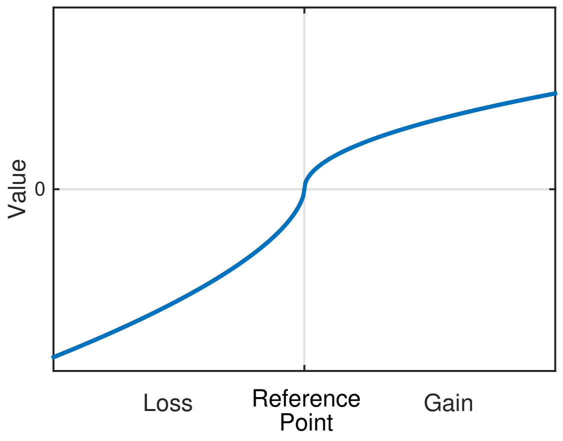


Figure 1.5: Value function of an agent in prospect theory

When an agent tries to maximize its reward over a time horizon, the utility, and consequently, the risk-sensitivity of the agent affects the dynamics of the optimization procedure. For example, consider the following problem:

$$\begin{aligned} \max_{\theta \in \Theta} \quad & \sum_{t=1}^{T} U(x_t) \\ \text{subject to} \quad & x_{t+1} = f_\theta(x_t), \end{aligned}$$

where U is the utility function of the agent, x_t is the outcome observed by the agent at time t, T is the length of the time horizon, Θ is the set of parameters the agent could choose from, and f_θ is the mapping describing the evolution of the outcomes based on the actions of the agent. Depending on the mapping f_θ, the outcomes observed by the agent could potentially grow exponentially large towards the end of the horizon; and any slight change in the parameter θ could cause drastic changes in the outcomes near the end of the horizon. In such a case, the concave or convex structure of the utility U could balance the growth of x_t, thereby improving the stability of the optimization algorithm, or it could further magnify the changes in the outcome and render the convergence of the algorithm to a fixed point challenging.

1.4 Overview of Chapters

In Chapter 2, we analyze the dynamics of the gradient descent algorithm when it is used for training multi-layer models. By studying the Lyapunov stability of the distinct equilibria of the algorithm, we establish a relationship between the learning rate of the algorithm and the set of solutions the algorithm can converge to. We then show that keeping layers of the model close to the identity mapping enables convergence of the algorithm with learning rates close to the maximum step size allowed.

In Chapter 3, we study the robustness of neural networks trained with the squared-error loss from the perspective of system identification. We analyze the relationship between the richness in training data and hidden-layer activations during training and the robustness of the model obtained at the end of training. We establish necessary and sufficient richness conditions on the training data and the hidden-layer activations in order for the convergence of the gradient descent algorithm to imply the boundedness of the model trained. We then demonstrate that these richness conditions are not satisfied by naively trained networks as the signals in the hidden-layers of the network easily become low-dimensional. To understand regularization of neural networks better, we revisit the classical regularization methods for single-layer linear models and reinterpret them in terms of boosting the richness of training data. Based on this analogy, and in light of the richness conditions derived, we propose a new training algorithm for neural networks that improves the richness of the signals passing through the parameters of the network during training. Lastly, we demonstrate that the proposed algorithm yields comparable margin characteristics on the training and test data for a network trained for classification with CIFAR-10 dataset.

In Chapter 4, we study the robustness of single-layer and multi-layer models trained with the cross-entropy loss. We demonstrate that the removal of correctly classified training data from the dynamics of the gradient descent algorithm exponentially quickly leads to extremely poor margins for the classifiers trained with the cross-entropy loss. Similar to Chapter 3, we derive richness conditions on the training data to ensure robustness of the models trained with the cross-entropy loss.

In Chapter 5, we study the dynamics of the gradient descent algorithm when it is used for training a dynamical system model with time-series data. By analyzing the Lyapunov stability of the fixed points of the gradient descent algorithm, we reveal how differently the stable and unstable modes of the dynamical system influence the stability of the gradient descent algorithm. For unstable dynamical systems, we show that the observations collected from the system near the end of the time horizon impose an exponentially strict bound on the learning rate that could be used for the gradient descent algorithm. We then demonstrate that employing risk-sensitive observations can address this imbalance and enable training unstable dynamical systems as well.

In Chapter 6, we study learning risk-sensitive utilities of an agent in a dynamic context. We build a hidden Markov model to represent the evolution of the reference values of the agent, and estimate their risk-sensitivity and the transition probabilities for their reference values by the expectation-maximization algorithm. We then test the suggested algorithm on the data set of New York City taxi drivers, and we demonstrate its ability to explain the trade-off the taxi drivers make when deciding how long they drive each day based on their references.

Finally in Chapter 7, we summarize the key results in this dissertation and discuss possible directions for future research.

Chapter 2

Lyapunov Analysis for Training Multi-Layer Models

2.1 Introduction

Consider the nonconvex function $f(x) = (x^2 + 1)(x - 1)^2(x - 2)^2$, which has two local minima at $x = 1$ and $x = 2$ as shown in Figure 2.1. Note that these local minima are also the two of the isolated equilibria of the dynamical system created by the gradient descent algorithm. The *stability* of these equilibria *in the sense of Lyapunov* is determined by the step size of the algorithm. In particular, since the *smoothness* parameter of f around these equilibria is 4 and 10, they are stable only if the step size is smaller than 0.5 and 0.2, respectively, and the gradient descent algorithm can converge to them only when these conditions are satisfied. Due to the difference in the largest step size allowed for different equilibria, step size conveys information about the solution obtained by the gradient descent algorithm. For example, if the algorithm converges to an equilibrium with step size 0.3 from a randomly chosen initial point, then this equilibrium is almost surely $x = 1$.

Based on this observation, in this chapter, we study the gradient descent algorithm as a discrete-time dynamical system during training deep neural networks, and we show the relationship between the step size of the algorithm and the solutions that can be obtained with this algorithm. In particular, we achieve the following:

1. We analyze the Lyapunov stability of the gradient descent algorithm on deep linear networks and find different upper bounds on the step size that enable convergence to each solution. We show that for every step size, the algorithm can converge to only a subset of the local optima, and there are always some local optima that the algorithm cannot converge to independent of the initialization.

2. We establish that for deep linear networks, there is a direct connection between the smoothness parameter of the training loss function and the largest singular value of the estimated linear function. In particular, we show that if the gradient descent algorithm can converge to a

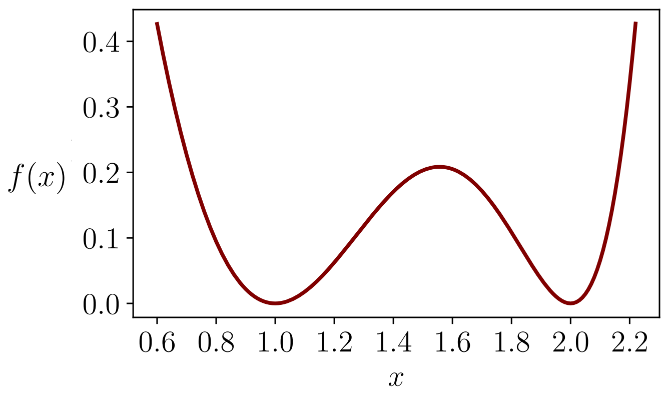


Figure 2.1: The function $f(x) = (x^2+1)(x-1)^2(x-2)^2$ of Example 2. Since the smoothness parameter of f at $x = 1$ is smaller than that at $x = 2$, the gradient descent algorithm cannot converge to $x = 2$ but can converge to $x = 1$ for some values of the step size. If, for example, the algorithm converges to an equilibrium from a randomly chosen initial point with step size 0.3, then this equilibrium is almost surely $x = 1$.

solution with a large step size, the function estimated by the network must have small singular values, and hence, the estimated function must have a small Lipschitz constant.

3. We show that symmetric positive definite matrices can be estimated with a deep linear network by initializing the weight matrices as the identity, and this initialization allows the use of the largest step size. Conversely, the algorithm is most likely to converge for an arbitrarily chosen step size if the weight matrices are initialized as the identity.

4. We show that symmetric matrices with negative eigenvalues, on the other hand, cannot be estimated with the identity initialization, and the gradient descent algorithm converges to the closest positive semidefinite matrix in the Frobenius norm.

5. For 2-layer neural networks with ReLU activations, we obtain an explicit relationship between the step size of the gradient descent algorithm and the output of the solution that the algorithm can converge to.

The results in this chapter have appeared in (Nar and Sastry, 2018a).

2.1.1 Related work

It is a well-known problem that the gradient of the training cost function can become disproportionate for different parameters when training a neural network. Several works in the literature tried to address this problem. For example, changing the geometry of optimization was proposed in (Neyshabur et al., 2017) and a regularized descent algorithm was proposed to prevent the gradients from exploding and vanishing during training.

Deep residual networks, which is a specific class of neural networks, yielded exceptional results in practice with their peculiar structure (He et al., 2016). By keeping each layer of the network close to the identity function, these networks were able to attain lower training and test errors as the depth of the network was increased. To explain their distinct behavior, the training cost function of their linear versions was shown to possess some crucial properties (Hardt and Ma, 2016). Later, equivalent results were also derived for nonlinear residual networks under certain conditions (Bartlett et al., 2018a).

The effect of the step size on training neural networks was empirically investigated in (Daniel et al., 2016). A step size adaptation scheme was proposed in (Rolinek and Martius, 2018) for the stochastic gradient method and shown to outperform the training with a constant step size. Similarly, some heuristic methods with variable step size were introduced and tested empirically in (Magoulas et al., 1997) and (Jacobs, 1988).

Two-layer linear networks were first studied in (Baldi and Hornik, 1989). The analysis was extended to deep linear networks in (Kawaguchi, 2016), and it was shown that all local optima of these networks were also the global optima. It was discovered in (Hardt and Ma, 2016) that the only critical points of these networks were actually the global optima as long as all layers remained close to the identity function during training. The dynamics of training these networks were also analyzed in (Saxe et al., 2013) and (Gunasekar et al., 2017) by assuming an infinitesimal step size and using a continuous-time approximation to the dynamics.

Lyapunov analysis from the dynamical system theory (Khalil, 1996; Sastry, 2013), which is the main tool for our results in this work, was used in the past to understand and improve the training of neural networks – especially that of the recurrent neural networks (Michel et al., 1988; Matsuoka, 1992; Barabanov and Prokhorov, 2002). State-of-the-art feedforward networks, however, have not been analyzed from this perspective.

We summarize the major differences between our contributions and the previous works as follows:

1. We relate the vanishing and exploding gradients that arise during training feedforward networks to the Lyapunov stability of the gradient descent algorithm.

2. Unlike the continuous-time analyses given in (Saxe et al., 2013) and (Gunasekar et al., 2017), we study the discrete-time dynamics of the gradient descent with an emphasis on the step size. By doing so, we obtain upper bounds on the step size to be used, and we show that the step size restricts the set of local optima that the algorithm can converge to. Note that these results cannot be obtained with a continuous-time approximation.

3. For deep linear networks with residual structure, (Hardt and Ma, 2016) shows that the gradient of the cost function cannot vanish away from a global optimum. This is not enough, however, to suggest the fast convergence of the algorithm. Given a fixed step size, the algorithm may also converge to an oscillation around a local optimum. We rule out this possibility and provide a step size so that the algorithm converges to a global optimum with a linear rate.

4. The convergence of the gradient descent algorithm was also studied in (Bartlett et al., 2018b) for symmetric positive definite matrices independently of and concurrently with our preliminary work (Nar and Sastry, 2018b). However, unlike (Bartlett et al., 2018b), we explicitly give a step size value for the algorithm to converge with a linear rate, and we emphasize the fact that the identity initialization allows convergence with the largest step size.

2.2 Upper Bounds on the Step Size for Training Deep Linear Networks

Deep linear networks are a special class of neural networks that do not contain nonlinear activations. They represent a linear mapping and can be described by a multiplication of a set of matrices, namely, $W_L \cdots W_1$, where $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$ for each $i \in [L] := \{1, 2, \dots, L\}$. Due to the multiplication of different parameters, their training cost is never a quadratic function of the parameters, and therefore, the dynamics of the gradient descent algorithm is always nonlinear during training of these networks. For this reason, they provide a simple model to study some of the nonlinear behaviors observed during neural network training.

Given a cost function $\ell(W_L \cdots W_1)$, if point $\{\hat{W}_i\}_{i\in[L]}$ is a local minimum, then $\{\alpha_i \hat{W}_i\}_{i\in[L]}$ is also a local minimum for every set of scalars $\{\alpha_i\}_{i\in[L]}$ that satisfy $\alpha_1\alpha_2\cdots\alpha_L = 1$. Consequently, independent of the specific choice of ℓ, the training cost function have infinitely many local optima, none of these local optima is isolated in the parameter space, and the cost function is not strongly convex at any point in the parameter space.

Although multiple local optima attain the same training cost for deep linear networks, the dynamics of the gradient descent algorithm exhibits distinct behaviors around these points. In particular, the step size required to render each of these local optima *stable in the sense of Lyapunov* is very different. Since the Lyapunov stability of a point is a necessary condition for the convergence of the algorithm to that point, the step size that allows convergence to each solution is also different, which is formalized in Theorem 2.1.

Theorem 2.1. *Given a nonzero matrix $R \in \mathbb{R}^{n_L \times n_0}$ and a set of points $\{x_i\}_{i\in[N]}$ in $\mathbb{R}^{n_0}$ that satisfy $\frac{1}{N}\sum_{i=1}^N x_i x_i^\top = I$, assume that R is estimated as a multiplication of the matrices $\{W_j\}_{j\in[L]}$ by minimizing the squared error loss*

$$\frac{1}{2N}\sum_{i=1}^N \|Rx_i - W_L W_{L-1}\dots W_2 W_1 x_i\|_2^2 \tag{2.1}$$

where $W_j \in \mathbb{R}^{n_j \times n_{j-1}}$ for all $j \in [L]$. Then the gradient descent algorithm with random initialization can converge to a solution $\{\hat{W}_j\}_{j\in[L]}$ only if the step size δ satisfies

$$\delta \le \frac{2}{\sum_{j=1}^L p_{j-1}^2 q_{j+1}^2} \tag{2.2}$$

where

$$p_j = \left\|\hat{W}_j \cdots \hat{W}_2 \hat{W}_1 v\right\|, \quad q_j = \left\|u^\top \hat{W}_L \hat{W}_{L-1} \cdots \hat{W}_j\right\| \quad \forall j \in [L],$$

and u and v are the left and right singular vectors of $\hat{R} = \hat{W}_L \cdots \hat{W}_1$ corresponding to its largest singular value.

Considering all the solutions $\{\alpha_i \hat{W}_i\}_{i \in [L]}$ that satisfy $\alpha_1 \alpha_2 \cdots \alpha_L = 1$, the bound in (2.2) can be arbitrarily small for some of the local optima. Therefore, given a fixed step size δ, the gradient descent can converge to only a subset of the local optima, and there are always some solutions that the gradient descent cannot converge to independent of the initialization.

Remark 2.1. Theorem 1 provides a necessary condition for convergence to a specific solution. It rules out the possibility of converging to a large subset of the local optima; however, it does *not* state that given a step size δ, the algorithm converges to a solution which satisfies (2.2). It might be the case, for example, that the algorithm converges to an oscillation around a local optimum which violates (2.2) even though there are some other local optima which satisfy (2.2).

As a necessary condition for the convergence to a global optimum, we can also find an upper bound on the step size independent of the weight matrices of the solution, which is given next.

Corollary 2.1. *For the minimization problem in Theorem 1, the gradient descent algorithm with random initialization can converge to a global optimum only if the step size δ satisfies*

$$\delta \leq \frac{2}{L \rho(R)^{2(L-1)/L}}, \tag{2.3}$$

where $\rho(R)$ is the largest singular value of R.

Remark 2.2. Corollary 2.1 shows that, unlike the optimization of the ordinary least squares problem, the step size required for the convergence of the algorithm depends on the parameter to be estimated, R. Consequently, estimating linear mappings with larger singular values requires the use of a smaller step size. Conversely, the step size used during training gives information about the solution obtained if the algorithm converges. That is, if the algorithm has converged with a large step size, then the Lipschitz constant of the function estimated must be small.

Corollary 2.2. *Assume that the gradient descent algorithm with random initialization has converged to a local optimum $\hat{R} = \hat{W}_L \ldots \hat{W}_1$ for the minimization problem in Theorem 1. Then the largest singular value of $\hat{R}$ satisfies*

$$\rho(\hat{R}) \leq \left(\frac{2}{L\delta} \right)^{L/(2L-2)}$$

almost surely.

The smoothness parameter of the training cost function is directly related to the largest step size that can be used, and consequently, to the Lyapunov stability of the gradient descent algorithm. The denominators of the upper bounds (2.2) and (2.3) in Theorem 2.1 and Corollary 2.1 necessarily provide a lower bound for the smoothness parameter of the training cost function around corresponding local optima. As a result, Theorem 2.1 implies that there is no finite Lipschitz constant for the gradient of the training cost function over the whole parameter space.

2.3 Identity Initialization for Estimating Symmetric Positive Definite Matrices

Corollary 2.1 provides only a necessary condition for the convergence of the gradient descent algorithm, and the bound (2.3) is not tight for every estimation problem. However, if the matrix to be estimated is symmetric and positive definite, the algorithm can converge to a solution with step sizes close to (2.3), which requires a specific initialization of the weight parameters.

Theorem 2.2. *Assume that $R \in \mathbb{R}^{n\times n}$ is a symmetric positive semidefinite matrix, and given a set of points $\{x_i\}_{i\in[N]}$ which satisfy $\frac{1}{N}\sum_{i=1}^{N} x_i x_i^\top = I$, the matrix R is estimated as a multiplication of the square matrices $\{W_j\}_{j\in[L]}$ by minimizing*

$$\frac{1}{2N}\sum_{i=1}^{N} \|Rx_i - W_L \dots W_1 x_i\|_2^2.$$

If the weight parameters are initialized as $W_i[0] = I$ for all $i \in [L]$ and the step size satisfies

$$\delta \le \min\left\{\frac{1}{L}, \frac{1}{L\rho(R)^{2(L-1)/L}}\right\},$$

then each W_i converges to $R^{1/L}$ with a linear rate.

Remark 2.3. Theorem 2.2 shows that the algorithm converges to a global optimum despite the nonconvexity of the optimization, and it provides a case where the bound (2.3) is almost tight. The tightness of the bound implies that for the same step size, most of the other global optima are *unstable in the sense of Lyapunov*, and therefore, the algorithm cannot converge to them independent of the initialization. Consequently, using identity initialization allows convergence to a solution which is most likely to be *stable* for an arbitrarily chosen step size.

Remark 2.4. Given that the identity initialization on deep linear networks is equivalent to the zero initialization of linear residual networks (Hardt and Ma, 2016), Theorem 2.2 provides an explanation for the exceptional performance of deep residual networks as well (He et al., 2016).

When the matrix to be estimated is symmetric but not positive semidefinite, the bound (2.3) is still tight for some of the global optima. In this case, however, the eigenvalues of the estimate cannot attain negative values if the weight matrices are initialized with the identity.

Theorem 2.3. *Let $R \in \mathbb{R}^{n\times n}$ in Theorem 2.2 be a symmetric matrix such that the minimum eigenvalue of R, $\lambda_{\min}(R)$, is negative. If the weight parameters are initialized as $W_i[0] = I$ for all $i \in [L]$ and the step size satisfies*

$$\delta \le \min\left\{\frac{1}{1-\lambda_{\min}(R)}, \frac{1}{L}, \frac{1}{L\rho(R)^{2(L-1)/L}}\right\},$$

then the estimate $\hat{R} = \hat{W}_L \cdots \hat{W}_1$ converges to the closest positive semidefinite matrix to R in Frobenius norm.

From the analysis of symmetric matrices, we observe that the step size required for convergence to a global optimum is largest when the largest singular vector of R is amplified or attenuated equally at each layer of the network. If the initialization of the weight matrices happens to affect this vector in the opposite ways, i.e., if some of the layers attenuate this vector and the others amplify this vector, then the required step size for convergence could be very small.

2.4 Effect of Step Size on Training Two-Layer Networks with ReLU Activations

In Section 2.2, we analyzed the relationship between the step size of the gradient descent algorithm and the solutions that can be obtained by training deep linear networks. A similar relationship exists for nonlinear networks as well. The following theorem, for example, provides an upper bound on the step size for the convergence of the algorithm when the network has two layers and ReLU activations.

Theorem 2.4. *Given a set of points $\{x_i\}_{i\in[N]}$ in $\mathbb{R}^n$, let a function $f:\mathbb{R}^n\to\mathbb{R}^m$ be estimated by a two-layer neural network with ReLU activations by minimizing the squared error loss:*

$$\min_{W,V}\ \frac{1}{2}\sum\nolimits_{i=1}^{N}\|Wg(Vx_i-b)-f(x_i)\|_2^2,$$

where $g(\cdot)$ is the ReLU function, $b\in\mathbb{R}^r$ is the fixed bias vector, and the optimization is only over the weight parameters $W\in\mathbb{R}^{m\times r}$ and $V\in\mathbb{R}^{r\times n}$. If the gradient descent algorithm with random initialization converges to a solution $(\hat{W},\hat{V})$, then the estimate $\hat{f}(x)=\hat{W}g(\hat{V}x-b)$ satisfies

$$\max_{i\in[N]}\ \|x_i\|_2\|\hat{f}(x_i)\|_2\le\frac{1}{\delta}$$

almost surely.

Theorem 2.4 shows that if the algorithm is able to converge with a large step size, then the estimate $\hat{f}(x)$ must have a small magnitude for large values of $\|x\|$.

Similar to Corollary 2.1, the bound given by Theorem 2.4 is not necessarily tight. Nevertheless, it highlights the effect of the step size on the convergence of the algorithm. To demonstrate that small changes in the step size could lead to significantly different solutions, we generated a piecewise continuous function $f:[0,1]\to\mathbb{R}$ and estimated it with a two-layer network by minimizing

$$\sum\nolimits_{i=1}^{N}|Wg\left(Vx_i-b\right)-f\left(x_i\right)|^2$$

with two different step sizes $\delta\in\{2\cdot10^{-4},3\cdot10^{-4}\}$, where $W\in\mathbb{R}^{1\times20}$, $V\in\mathbb{R}^{20},b\in\mathbb{R}^{20}$, $N=1000$ and $x_i=i/N$ for all $i\in[N]$. The initial values of W,V and the constant vector b were all drawn from independent standard normal distributions; and the vector b was kept the same for both of the step sizes used. As shown in Figure 2.2, training with $\delta=2\cdot10^{-4}$ converged to a fixed solution, which provided an estimate $\hat{f}$ close the original function f. In contrast, training with $\delta = 3 \cdot 10^{-4}$ converged to an oscillation and not to a fixed point. That is, after sufficient training, the estimate kept switching between $\hat{f}_{\text{odd}}$ and $\hat{f}_{\text{even}}$ at each iteration of the gradient descent. The code for the experiment is available at `https://github.com/nar-k/NIPS-2018`.

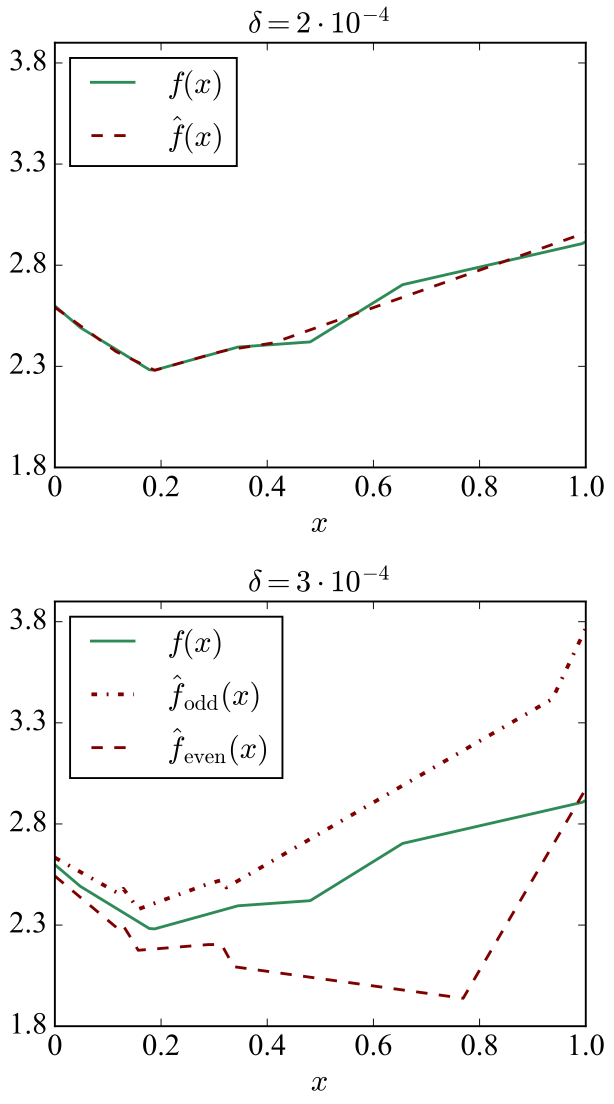


Figure 2.2: Estimates of the function f obtained by training a two-layer neural network with two different step sizes. [Top] When the step size of the gradient descent is $\delta = 2 \cdot 10^{-4}$, the algorithm converges to a fixed point, which provides an estimate $\hat{f}$ close to f. [Bottom] When the step size is $\delta = 3 \cdot 10^{-4}$, the algorithm converges to an oscillation and not to a fixed solution. That is, after sufficient training, the estimate keeps switching between $\hat{f}_{\text{odd}}$ and $\hat{f}_{\text{even}}$ at each iteration.

2.5 Discussion

When gradient descent algorithm is used to minimize a function, typically only three possibilities are considered: convergence to a local optimum, to a global optimum, or to a saddle point. In this chapter, we considered the fourth possibility: the algorithm may not converge at all – even in the deterministic setting. The training error may not reflect the oscillations in the dynamics, or when a stochastic optimization method is used, the oscillations in the training error might be wrongly attributed to the stochasticity of the algorithm. We underlined that, if the training error of an algorithm converges to a non-optimal value, that does not imply the algorithm is stuck near a bad local optimum or a saddle point; it might simply be the case that the algorithm has not converged at all.

We showed that the step size of the gradient descent algorithm influences the dynamics of the algorithm substantially. It renders some of the local optima unstable in the sense of Lyapunov, and the algorithm cannot converge to these points independent of the initialization. It also determines the magnitude of the oscillations if the algorithm converges to an orbit around an equilibrium point in the parameter space.

In Corollary 2.2 and Theorem 2.4, we showed that the step size required for convergence to a specific solution depends on the solution itself. In particular, we showed that there is a direct connection between the smoothness parameter of the training loss function and the Lipschitz constant of the function estimated by the network. This reveals that some solutions, such as linear functions with large singular values, are harder to converge to. Given that there exists a relationship between the Lipschitz constants of the estimated functions and their generalization error (Bartlett et al., 2017), this result could provide a better understanding of the generalization of deep neural networks.

The analysis in this chapter was limited to the gradient descent algorithm. It remains as an important open problem to investigate if the results in this work have analogs for the stochastic gradient methods and the algorithms with adaptive step sizes.

2.6 Proofs

This section provides the proofs for the theorems and the corollaries of this chapter.

2.6.1 Proof of Theorem 2.1 and Corollary 2.1

Lemma 2.1. *Let $A, B \in \mathbb{R}^{n\times n}$ be symmetric and positive semidefinite. Then, $\langle A, B\rangle \geq 0$.*

Proof. We can write B as $B = \sum_{i=1}^n \lambda_i u_i u_i^\top$, where $\lambda_i \geq 0$ for all $i \in [n]$ and $u_i^\top u_j = 0$ if $i \neq j$. Then,

$$\langle A, B\rangle = \text{trace}\,\{AB\} = \text{trace}\left\{A\sum_{i=1}^n \lambda_i u_i u_i^\top\right\} = \sum_{i=1}^n \lambda_i u_i^\top A u_i \geq 0. \qquad \blacksquare$$

Lemma 2.2. *Let $f : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ be a linear map defined as $f(X) = \sum_{i=1}^{L} A_i X B_i$, where $A_i \in \mathbb{R}^{m\times m}$ and $B_i \in \mathbb{R}^{n\times n}$ are symmetric positive semidefinite matrices for all $i \in [L]$. Then, for every nonzero $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$, the largest eigenvalue of f satisfies*

$$\lambda_{\max}(f) \geq \frac{1}{\|u\|_2^2\|v\|_2^2}\sum\nolimits_{i=1}^{L}(u^\top A_i u)(v^\top B_i v).$$

Proof. First, we show that f is symmetric and positive semidefinite. Given two matrices $X, Y \in \mathbb{R}^{m\times n}$, we can write

$$\langle X, f(Y)\rangle = \text{trace}\left\{\sum\nolimits_i X^\top A_i Y B_i\right\} = \text{trace}\left\{\sum\nolimits_i B_i Y^\top A_i X\right\} = \langle Y, f(X)\rangle,$$

$$\langle X, f(X)\rangle = \text{trace}\left\{\sum\nolimits_i X^\top A_i X B_i\right\} = \sum\nolimits_i \langle X^\top A_i X, B_i\rangle \geq 0,$$

where the last inequality follows from Lemma 2.1. This shows that f is symmetric and positive semidefinite. Then, for every nonzero $X \in \mathbb{R}^{m\times n}$, we have

$$\lambda_{\max}(f) \geq \frac{1}{\langle X, X\rangle}\langle X, f(X)\rangle.$$

In particular, given two nonzero vectors $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$,

$$\lambda_{\max}(f) \geq \frac{1}{\langle uv^\top, uv^\top\rangle}\langle uv^\top, f(uv^\top)\rangle = \frac{1}{\|u\|_2^2\|v\|_2^2}\sum\nolimits_{i=1}^{L}(u^\top A_i u)(v^\top B_i v). \qquad \blacksquare$$

Proof of Theorem 2.1. The cost function (2.1) in Theorem 2.1 can be written as

$$\frac{1}{2}\text{trace}\left\{(W_L\cdots W_1 - R)^\top (W_L\cdots W_1 - R)\right\}.$$

Let E denote the error in the estimate, i.e. $E = W_L \cdots W_1 - R$. The gradient descent yields

$$W_i[k+1] = W_i[k] - \delta W_{i+1}^\top[k]\cdots W_L^\top[k]E[k]W_1^\top[k]\cdots W_{i-1}^\top[k] \quad \forall i \in [L]. \tag{2.4}$$

By multiplying the update equations of $W_i[k]$ and subtracting R, we can obtain the dynamics of E as

$$E[k+1] = E[k] - \delta\sum\nolimits_{i=1}^{L} A_i[k]E[k]B_i[k] + o(E[k]), \tag{2.5}$$

where $o(\cdot)$ denotes the higher order terms, and

$$A_i = W_L W_{L-1}\cdots W_{i+1}W_{i+1}^\top\cdots W_{L-1}^\top W_L^\top \quad \forall i \in [L],$$

$$B_i = W_1^\top W_2^\top\cdots W_{i-1}^\top W_{i-1}\cdots W_2 W_1 \quad \forall i \in [L].$$

Lyapunov's indirect method of stability (Khalil, 1996; Sastry, 2013) states that given a dynamical system $x[k+1] = F(x[k])$, its equilibrium x^* is stable in the sense of Lyapunov only if the linearization of the system around x^*

$$(x[k+1] - x^*) = (x[k] - x^*) + \left.\frac{\partial F}{\partial x}\right|_{x=x^*}(x[k] - x^*)$$

does not have any eigenvalue larger than 1 in magnitude. By using this fact for the system defined by (2.4)-(2.5), we can observe that an equilibrium $\{\hat{W}_j\}_{j\in[L]}$ with $\hat{W}_L\cdots\hat{W}_1=\hat{R}$ is stable in the sense of Lyapunov only if the system

$$\Big(E[k+1]-\hat{R}+R\Big)=\Big(E[k]-\hat{R}+R\Big)-\delta\sum\nolimits_{i=1}^{L}A_i\Big|_{\{\hat{W}_j\}}\Big(E[k]-\hat{R}+R\Big)B_i\Big|_{\{\hat{W}_j\}}$$

does not have any eigenvalue larger than 1 in magnitude, which requires that the mapping

$$f(\tilde{E})=\sum\nolimits_{i=1}^{L}A_i\Big|_{\{\hat{W}_j\}}\tilde{E}B_i\Big|_{\{\hat{W}_j\}} \tag{2.6}$$

does not have any real eigenvalue larger than $(2/\delta)$. Let u and v be the left and right singular vectors of $\hat{R}$ corresponding to its largest singular value, and let p_j and q_j be defined as in the statement of Theorem 2.1. Then, by Lemma 2.2, the mapping f in (2.6) does not have an eigenvalue larger than $(2/\delta)$ only if

$$\sum\nolimits_{i=1}^{L}p_{i-1}^2q_{i+1}^2\le\frac{2}{\delta},$$

which completes the proof. ■

Proof of Corollary 2.1. Note that

$$q_{i+1}p_i=\|u^\top W_LW_{L-1}\cdots W_{i+1}\|_2\|W_i\cdots W_2W_1v\|_2\ge\|u^\top W_L\cdots W_1v\|_2=\rho(R).$$

As long as $\rho(R)\neq 0$, we have $p_i\neq 0$ for all $i\in[L]$, and therefore,

$$p_{i-1}^2q_{i+1}^2\ge\frac{p_{i-1}^2}{p_i^2}\rho(R)^2. \tag{2.7}$$

Using inequality (2.7), the bound in Theorem 2.1 can be relaxed as

$$\delta\le 2\Big(\sum\nolimits_{i=1}^{L}\frac{p_{i-1}^2}{p_i^2}\rho(R)^2\Big)^{-1}. \tag{2.8}$$

Since $\prod_{i=1}^{L}(p_i/p_{i-1})=\rho(R)\neq 0$, we also have the inequality

$$\sum\nolimits_{i=1}^{L}\frac{p_{i-1}^2}{p_i^2}\rho(R)^2\ge\sum\nolimits_{i=1}^{L}\frac{\rho(R)^2}{\left(\rho(R)^{1/L}\right)^2}=L\rho(R)^{2(L-1)/L},$$

and the bound in (2.8) can be simplified as

$$\delta\le\frac{2}{L\rho(R)^{2(L-1)/L}}.$$ ■

2.6.2 Proof of Theorem 2.2

Lemma 2.3. *Let $\lambda > 0$ be estimated as a multiplication of the scalar parameters $\{w_i\}_{i\in[L]}$ by minimizing $\frac{1}{2}(w_L \cdots w_2 w_1 - \lambda)^2$ via gradient descent. Assume that $w_i[0] = 1$ for all $i \in [L]$. If the step size δ is chosen to be less than or equal to*

$$\delta_c = \begin{cases} L^{-1}\lambda^{-2(L-1)/L} & \text{if } \lambda \in [1,\infty), \\ (1-\lambda)^{-1}(1-\lambda^{1/L}) & \text{if } \lambda \in (0,1), \end{cases}$$

then $|w_i[k] - \lambda^{\frac{1}{L}}| \le \beta(\delta)^k |1 - \lambda^{\frac{1}{L}}|$ for all $i \in [L]$, where

$$\beta(\delta) = \begin{cases} 1 - \delta(\lambda-1)(\lambda^{1/L}-1)^{-1} & \text{if } \lambda \in (1,\infty), \\ 1 - \delta L \lambda^{2(L-1)/L} & \text{if } \lambda \in (0,1]. \end{cases}$$

Proof. Due to symmetry, $w_i[k] = w_j[k]$ for all $k \in \mathbb{N}$ for all $i, j \in [L]$. Denoting any of them by $w[k]$, we have

$$w[k+1] = w[k] - \delta w^{L-1}[k](w^L[k] - \lambda).$$

To show that $w[k]$ converges to $\lambda^{1/L}$, we can write

$$w[k+1] - \lambda^{1/L} = \mu(w[k])(w[k] - \lambda^{1/L}),$$

where

$$\mu(w) = 1 - \delta w^{L-1} \sum_{j=0}^{L-1} w^j \lambda^{(L-1-j)/L}.$$

If there exists some $\beta \in [0,1)$ such that

$$0 \le \mu(w[k]) \le \beta \text{ for all } k \in \mathbb{N}, \tag{2.9}$$

then $w[k]$ is always larger or always smaller than $\lambda^{1/L}$, and its distance to $\lambda^{1/L}$ decreases by a factor of β at each step. Since $\mu(w)$ is a monotonic function in w, the condition (2.9) holds for all k if it holds only for $w[0] = 1$ and $\lambda^{1/L}$, which gives us δ_c and $\beta(\delta)$. ∎

Proof of Theorem 2.2. There exists a common invertible matrix $U \in \mathbb{R}^{n\times n}$ that can diagonalize all the matrices in the system created by the gradient descent: $R = U\Lambda_R U^\top$, $W_i = U\Lambda_{W_i}U^\top$ for all $i \in [L]$. Then the dynamical system turns into n independent update rules for the diagonal elements of Λ_R and $\{\Lambda_{W_i}\}_{i\in[L]}$. Lemma 2.3 can be applied to each of the n systems involving the diagonal elements. Since δ_c in Lemma 2.3 is monotonically decreasing in λ, the bound for the maximum eigenvalue of R guarantees linear convergence. ∎

2.6.3 Proof of Theorem 2.3

Lemma 2.4. *Assume that $\lambda < 0$ and $w_i[0] = 1$ is used for all $i \in [L]$ to initialize the gradient descent algorithm to solve*

$$\min_{(w_1,\ldots,w_L)\in\mathbb{R}^L} \frac{1}{2}\left(w_L \ldots w_2 w_1 - \lambda\right)^2.$$

Then, each w_i converges to 0 unless $\delta > (1-\lambda)^{-1}$.

Proof. We can write the update rule for any weight w_i as

$$w[k+1] = w[k]\left(1 - \delta\sigma w^{L-2}[k]\left(w^L[k] - \lambda\right)\right)$$

which has one equilibrium at $w^* = \lambda^{1/L}$ and another at $w^* = 0$. If $0 < \delta \leq 1/\sigma(1-\lambda)$ and $w[0] = 1$, it can be shown by induction that

$$0 \leq 1 - \delta\sigma w^{L-2}[k]\left(w^L[k] - \lambda\right) < 1$$

for all $k \geq 0$. As a result, $w[k]$ converges to 0. ■

Proof of Theorem 2.3. Similar to the proof of Theorem 2.2, the system created by the gradient descent can be decomposed into n independent systems of the diagonal elements of the matrices Λ_R and $\{\Lambda_{W_i}\}_{i\in[L]}$. Then, Lemma 2.3 and Lemma 2.4 can be applied to the systems with positive and negative eigenvalues of R, respectively. ■

2.6.4 Proof of Theorem 2.4

To find a necessary condition for the convergence of the gradient descent algorithm to $(\hat{W}, \hat{V})$, we analyze the local stability of that solution in the sense of Lyapunov. Since the analysis is local and the function g is fixed, for each point x_i we can use a matrix G_i that satisfies $G_i(\hat{V}x_i - b) = g(\hat{V}x_i - b)$. Note that G_i is a diagonal matrix and all of its diagonal elements are either 0 or 1. Then, we can write the cost function around an equilibrium as

$$\frac{1}{2}\sum\nolimits_{i=1}^{N} \text{trace}\left\{[WG_i(Vx_i - b) - f(x_i)]^\top [WG_i(Vx_i - b) - f(x_i)]\right\}.$$

Denoting the error $WG_i(Vx_i - b) - f(x_i)$ by e_i, the gradient descent gives

$$W[k+1] = W[k] - \delta\sum\nolimits_{i=1}^{N} e_i[k](V[k]x_i - b)^\top G_i^\top,$$

$$V[k+1] = V[k] - \delta\sum\nolimits_{i=1}^{N} G_i^\top W[k]^\top e_i[k]x_i^\top.$$

Let e denote the vector $(e_1^\top \ \ldots \ e_N^\top)^\top$. Then we can write the update equation of e_j as

$$\begin{aligned} e_j[k+1] \quad &= \quad e_j[k] - \delta W[k]G_j\sum\nolimits_i G_i^\top W[k]^\top e_i[k]x_i^\top x_j \\ &\quad -\delta\sum\nolimits_i e_i[k](V[k]x_i - b)^\top G_i^\top G_j(V[k]x_j - b) + o(e[k]). \end{aligned}$$

Similar to the proof of Theorem 1, the equilibrium $(\hat{W}, \hat{V})$ can be stable in the sense on Lyapunov only if the system

$$e_j[k+1] = e_j[k] - \delta\sum\nolimits_i \hat{W}G_jG_i^\top\hat{W}^\top e_i[k]x_i^\top x_j - \delta\sum\nolimits_i e_i[k](\hat{V}x_i - b)^\top G_i^\top G_j(\hat{V}x_j - b) \quad (2.10)$$

does not have any eigenvalue larger than 1 in magnitude. Note that the linear system in (2.10) can be described by a symmetric matrix, whose eigenvalues cannot be larger in magnitude than the eigenvalues of its sub-blocks on the diagonal, in particular those of the system

$$e_j[k+1] = e_j[k] - \delta \hat{W} G_j G_j^\top \hat{W}^\top e_j[k] x_j^\top x_j - \delta e_j[k](\hat{V}x_j - b)^\top G_j^\top G_j(\hat{V}x_j - b). \tag{2.11}$$

The eigenvalues of the system (2.11) are less than 1 in magnitude only if the eigenvalues of the system

$$h(u) = \hat{W} G_j G_j^\top \hat{W}^\top u x_j^\top x_j + u(\hat{V}x_j - b)^\top G_j^\top G_j(\hat{V}x_j - b)$$

are less than $(2/\delta)$. This requires that for all $j \in [N]$ for which $\hat{f}(x_j) \neq 0$,

$$\begin{aligned}
\frac{2}{\delta} &\geq \frac{\langle \hat{f}(x_j), h(\hat{f}(x_j))\rangle}{\langle \hat{f}(x_j), \hat{f}(x_j)\rangle} \\
&= \frac{1}{\|\hat{f}(x_j)\|^2}\left(\|G_j^\top \hat{W}^\top \hat{f}(x_j)\|^2 \|x_j\|^2 + \|\hat{f}(x_j)\|^2 \|G_j(\hat{V}x_j - b)\|^2\right) \\
&\geq \frac{1}{\|\hat{f}(x_j)\|^2} \frac{\|(\hat{V}x_j - b)^\top G_j^\top G_j^\top \hat{W}^\top \hat{f}(x_j)\|^2}{\|(\hat{V}x_j - b)^\top G_j^\top\|^2} \|x_j\|^2 + \|G_j(\hat{V}x_j - b)\|^2 \\
&= \frac{1}{\|G_j(\hat{V}x_j - b)\|^2} \|\hat{f}(x_j)\|^2 \|x_j\|^2 + \|G_j(\hat{V}x_j - b)\|^2 \\
&\geq 2\|\hat{f}(x_j)\|\|x_j\|.
\end{aligned}$$

As a result, Lyapunov stability of the solution $(\hat{W}, \hat{V})$ requires

$$\frac{1}{\delta} \geq \max_i \|\hat{f}(x_i)\|\|x_i\|. \qquad \blacksquare$$

Chapter 3

Persistency of Excitation for Robustness of Multi-Layer Models

3.1 Introduction

When a linear model is trained for a supervised learning task, the training data set needs to span the whole input space for the model parameters to be learned accurately. If this condition is not satisfied, the model will not be trained on a subspace in its domain, and the response of the model will be unpredictable for inputs containing any component in this subspace. If the model is trained by an iterative optimization algorithm, this richness condition on the training data set must be satisfied *throughout* the training procedure. Fulfilling such a richness requirement is in general not difficult for single-layer models with fixed input data, but it is a nontrivial problem when identifying unknown parameters of dynamical systems (Kumar and Varaiya, 1986; Sastry and Bodson, 1989), and as we will show in this chapter, when training multi-layer models.

Consider, for example, a linear dynamical system in $\mathbb{R}^n$:

$$h_{t+1} = Ah_t + Bx_t \quad \forall t \in \mathbb{N} \tag{3.1a}$$

$$y_t = Ch_t \qquad \forall t \in \mathbb{N} \tag{3.1b}$$

with internal state $h_t \in \mathbb{R}^n$, input $x_t \in \mathbb{R}$, output $y_t \in \mathbb{R}$, and unknown parameters $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^n$ and $C \in \mathbb{R}^{1\times n}$. Assume we feed a constant input $\{x_t\}_{t\in\mathbb{N}}$ into this system, that is, we set $x_t = x_0$ for all $t \in \mathbb{N}$, and we try to learn the input-output relationship of the system by observing $\{y_t\}_{t\in\mathbb{N}}$ while the internal state of the system evolves. Note that as time increments, the eigenvalues of A with magnitude less than 1 cause the internal state h_t to lose its components in the eigenspaces corresponding to these eigenvalues. In other words, the stable eigenspaces of A vanish from $\{h_t\}_{t\in\mathbb{N}}$ exponentially fast, and consequently, not enough information is received about how the system behaves in these eigenspaces. As a result, the accurate input-output relationship of the system cannot be recovered.

This is a classical problem in system identification: when parameters of a dynamical system is estimated while the internal state of the system evolves, the input fed into the system needs to satisfy a certain richness condition. In particular, for successful identification of the system in (3.1), the input signal $\{x_t\}_{t\in\mathbb{N}}$ needs to contain a certain number of frequencies (Boyd and Sastry, 1983, 1986). As this example shows, training a dynamical model in an online fashion necessitates the injection of sustained perturbations into the system throughout the training procedure.

In this chapter, we show that similar requirements arise while training multi-layer models, even though there does not appear a dynamical model. As a preamble, consider a fixed data set $\{x_i\}_{i\in\mathcal{I}}$ and an L-layer feedforward neural network:

$$F(x) = f_L(f_{L-1}(\cdots(f_2(f_1(x))))),$$

where f_k represents the operation of the k-th layer of the network. The parameters of the k-th layer are excited by the activations of the previous layer, $\{f_{k-1}(\cdots(f_1(x_i)))\}_{i\in\mathcal{I}}$. Therefore, robust estimation of these parameters rely on whether this set is rich enough and whether this richness is maintained throughout the training procedure.

3.1.1 Outline and Contributions

By analyzing the dynamics of the of the gradient descent algorithm on feedforward neural networks trained with the squared-error loss, we achieve the following.

1. For deep linear networks, we show that having a full-rank data set is enough to ensure implicit regularization; that is, the convergence of the gradient descent algorithm provides a Lipschitz bound for the mapping represented by the network.

2. For a two-layer network with ReLU activations, we establish a richness condition on the activation signals such that the convergence of the algorithm provides a Lipschitz bound for the function represented by the network. In particular, we show that each node in the hidden-layer being activated by a set of points with full rank is a *necessary* condition.

3. For multi-layer networks with ReLU activations, we provide a richness condition on the hidden-layer activations that is sufficient for building a connection between the convergence of the gradient descent algorithm and the boundedness of the trained parameters. Then we show that this condition may not be satisfied by networks trained with the gradient descent algorithm naively. This is expected given the existence of adversarial examples for naively trained models.

4. We reinterpret the classical regularization terms for single-layer linear models in terms of boosting the richness of the training data. By building an analogy, and in light of the sufficient richness conditions derived in this chapter, we introduce a training algorithm that improves the richness of the hidden-layer activations of multi-layer networks. Lastly, we demonstrate that this algorithm leads to similar margin characteristics on the training and test data when a network is trained for a classification task with CIFAR-10 data set.

The results in this chapter have appeared in (Nar and Sastry, 2019) and (Nar and Sastry, 2020).

3.1.2 Related Work

Implicit regularization effect of the gradient descent algorithm on matrix factorization, deep linear networks and multi-layer structures has recently been studied in (Arora et al., 2019; Gidel et al., 2019; Gunasekar et al., 2017, 2018; Du et al., 2018; Ji and Telgarsky, 2019). Our work also addresses the same subject, but the emphasis is on the necessary and sufficient richness requirements on the training data and hidden-layer activations for implicit regularization to take place. Furthermore, our analysis uses a non-vanishing learning rate, thereby elucidating the effect of learning rate on regularization.

Robustness of a model against small perturbations in its input is closely tied to presence of an effective regularization during its training. State-of-the-art neural networks, however, are known to lack this robustness; imperceptibly small perturbations can change their outputs drastically (Szegedy et al., 2013; Kurakin et al., 2016; Goodfellow et al., 2015). We demonstrate in this work that naively trained networks will fail to fulfill the richness requirements on the hidden-layer activations for implicit regularization, which provides an alternative understanding for the lack of robustness in these models.

We discuss other related subjects in Section 3.6, after presenting our results.

3.2 Deep Linear Networks

Deep linear networks are feedforward networks with no nonlinear activations. Although they can represent only linear mappings, analysis of the gradient descent algorithm on these models reveals the importance of having multiple layers for implicit regularization. The following theorem shows that when a deep linear network is trained with the gradient descent algorithm, the mapping learned will necessarily have a bounded Lipschitz constant, provided that the training data set is rich enough.

Theorem 3.1. *Given a set of points $\{x_i\}_{i\in\mathcal{I}}$ in $\mathbb{R}^{n_0}$ and corresponding target values $\{y_i\}_{i\in\mathcal{I}}$ in $\mathbb{R}^{n_L}$, assume an L-layer deep linear network is trained by minimizing the squared-error loss via the gradient descent algorithm:*

$$\min_{W_1,\dots,W_L} \frac{1}{2}\sum\nolimits_{i\in\mathcal{I}} \|W_L\cdots W_2W_1x_i - y_i\|_2^2$$

where $W_j \in \mathbb{R}^{n_j\times n_{j-1}}$ is the weight matrix of the j-th layer of the network for $j\in[L]=\{1,\dots,L\}$. For almost every initialization, convergence of the gradient descent algorithm to the solution $(\hat{W}_1,\dots,\hat{W}_L)$ with learning rate δ implies that

$$\rho(\hat{W}_L\cdots\hat{W}_1) \le \left(\frac{2}{\delta\sigma}\right)^{L/(2L-2)}$$

where σ is the minimum eigenvalue of $\sum_{i\in\mathcal{I}} x_i x_i^\top$ and $\rho(\cdot)$ denotes the largest singular value of its argument. $\square$

Note that what is bounded in Theorem 3.1 is the Lipschitz constant of the mapping learned; it is not the Lipschitz constant of the gradient of the loss function, nor is it the smoothness of the loss function. In this respect, the result of Theorem 1 does not have an analogue for single-layer linear models trained with the squared-error loss.

Theorem 3.1 provides a necessary and sufficient condition for the implicit regularization of deep linear networks: full-rankness of the training data. There is no additional requirement for the hidden-layers, and this is peculiar to deep linear networks. This is caused by the fact that as inputs pass through the layers of the linear network, they can only lose rank; and

$$\text{span}\left(\{\hat{W}_j \cdots \hat{W}_1 x_i\}_{i\in\mathcal{I}}\right) = \text{span}\left(\{\hat{W}_j \cdots \hat{W}_1 x\}_{x\in\mathbb{R}^{n_0}}\right) \quad \forall j \in [L]$$

provided that $\{x_i\}_{i\in\mathcal{I}}$ is a full-rank data set. Therefore, ensuring the richness of the signals at the input layer suffices to ensure the richness of the signals in all layers of a deep linear network. We will see in the next section that when nonlinear activations are introduced into the network, the requirement on the richness of the hidden-layer signals will become more explicit.

3.3 Nonlinear Networks

In this section, we consider feedforward networks with ReLU activations, which are denoted with the element-wise operation

$$(z)_+ = \begin{cases} z & z > 0, \\ 0 & z \leq 0. \end{cases}$$

The following theorem provides a Lipschitz bound for a two-layer ReLU network trained with the gradient descent algorithm and the squared-error loss.

Theorem 3.2. *Given a set of points $\{x_i\}_{i\in\mathcal{I}}$ in $\mathbb{R}^n$ and corresponding target values $\{y_i\}_{i\in\mathcal{I}}$ in $\mathbb{R}^m$, assume that a two-layer neural network with parameters $W \in \mathbb{R}^{m\times r}$, $V \in \mathbb{R}^{r\times n}$, $b \in \mathbb{R}^r$ and ReLU activations is trained by minimizing the squared error loss*

$$\min_{W,V,b} \frac{1}{2} \sum_{i\in\mathcal{I}} \|W(Vx_i + b)_+ - y_i\|_2^2 \tag{3.2}$$

via the gradient descent algorithm. Let V_k and b_k denote the k-th rows of V and b for each $k \in [r]$. For almost every initialization, convergence of the algorithm with learning rate δ to a solution $(\hat{W}, \hat{V}, \hat{b})$ implies that the Lipschitz constant of f(x) = $\hat{W}(\hat{V}x + \hat{b})_+$ is upper bounded by

$$n_{\text{active}}^{\max} \left(\sqrt{\frac{2}{\delta\underline{\lambda}}}\right) \left(\frac{2\overline{\mu}\|\hat{b}\|_\infty}{\underline{\lambda}} + \sqrt{\frac{1}{\underline{\lambda}}\left|\frac{2}{\delta} - \|\hat{b}\|_\infty^2\right|}\right) \tag{3.3}$$

where $n_{\text{active}}^{\max}$ is the maximum number of nodes that can be simultaneously activated in the hidden layer:

$$n_{\text{active}}^{\max} = \max_{x\in\mathbb{R}^n} \sum_{k=1}^{r} \mathbb{I}\{\hat{V}_k x + \hat{b}_k > 0\},$$

$\underline{\lambda}$ is a lower bound for the minimum eigenvalue of the covariance matrices of the training points that activate the same hidden-layer node:

$$\underline{\lambda} = \min_{k\in[r]}\ \lambda_{\text{min}}\Big(\sum\nolimits_{i\in\mathcal{I}} x_i x_i^\top \mathbb{I}\{\hat{V}_k x + \hat{b}_k > 0\}\Big),$$

and $\overline{\mu}$ is an upper bound for the ℓ_2 norm of the summation of the training points that activate the same hidden-layer node:

$$\overline{\mu} = \max_{k\in[r]}\ \Big\|\sum\nolimits_{i\in\mathcal{I}} x_i \mathbb{I}\{\hat{V}_k x + \hat{b}_k > 0\}\Big\|_2. \qquad \square$$

Before we discuss Theorem 3.2 in detail, note that the bound provided in (3.3) is valid only when each node in the hidden layer is activated by a set of training points with full-rank. This, in fact, is a necessary condition for the convergence of the gradient descent algorithm to imply a bound on the parameters of the learned model via implicit regularization — provided that there is no assumption on the initialization of the parameters.

Corollary 3.1. *Assume a two-layer neural network with ReLU activations is trained by minimizing the squared error loss with the gradient descent algorithm. For the convergence of the algorithm to imply a bound on the parameters of the estimated function, it is **necessary** that each hidden-layer node be activated by a set of points with full rank at equilibrium.* □

Theorem 3.2 reveals the following:

1. Similar to the result for deep linear networks in Theorem 3.1, convergence with a larger step size implies a smaller Lipschitz constant for the mapping represented by the ReLU network, from its input to its output.

2. If the set of points activating any one of the hidden layer nodes has a large bias, then $\overline{\mu}$ will be large, and so will the upper bound for the Lipschitz constant.

The term $n_{\text{active}}^{\text{max}}$ in Theorem 3.2 is needed for the worst-case analysis. For an example of a data set and a network for which this multiplier is required, see Appendix 3.8.3. Note that we do not make any assumption about the initialization about the weight parameters, nor do we impose an explicit regularization.

Similar to Theorem 3.2, we can state a sufficient richness condition for multi-layer networks with ReLU activations. This is given next.

Theorem 3.3. *Consider an L-layer network with ReLU activations:*

$$\begin{aligned} h_0(x) &= x, \\ h_j(x) &= (W_j h_{j-1}(x))_+ \quad j = 1, 2, \cdots, L-1, \\ h_L(x) &= W_L h_{L-1}(x) \end{aligned}$$

with n_j nodes in its j-th hidden layer, and assume it has been trained by minimizing the squared-error loss with the gradient descent algorithm on the data set $\{x_i\}_{i\in\mathcal{I}}$. Let $\hat{W}_j$ and $\hat{h}_j$ denote the weight *matrix and the output of the j-th layer after the training, and define $\mathcal{I}' = \{i \in \mathcal{I} : \hat{h}_L(x_i) \neq 0\}$. If all hidden-layer activations are bounded over the training data set:*

$$\max_{i\in\mathcal{I}'} \max_{j\in[L]} \|\hat{h}_j(x_i)\|_2 < \infty,$$

and if every hidden-layer node is activated by a set of signals with full-rank in the preceding layer:

$$\sum_{i\in\mathcal{I}'} \mathbb{I}\{e_r^\top \hat{W}_j \hat{h}_{j-1}(x_i) > 0\} \cdot \hat{h}_{j-1}(x_i)\hat{h}_{j-1}^\top(x_i) \succ 0 \quad \forall r \in [n_j], \forall j \in [L],$$

then the convergence of the gradient descent algorithm implies the boundedness of $\hat{W}_j$ for all $j \in [L]$ for almost every initilization. □

Please refer to Appendix 3.8.4 for a detailed description of the bound on the trained parameters. Theorem 3.3 states that the convergence of the gradient descent algorithm guarantees the boundedness of the weight parameters if every hidden-layer node is activated by a set of activation patterns with a full rank in the preceding layer. This condition, however, is not easily satisfied, particularly by the networks trained naively by the gradient descent algorithm.

To demonstrate this, we trained a 5-layer convolutional neural network for a binary classification task. The training data was chosen as the two classes corresponding to the planes and the horses in CIFAR-10 data set. The network was trained with the squared-error loss. Figure 3.1 shows the principal component analysis of the signals in the hidden layers of the network after training. Note that the signals in the upper layers of the network can be explained by the first few principal components, which indicates that these signals are very low dimensional; and more importantly, the rank of these signals are lower than the width of corresponding layers. This shows that the richness condition described in Theorem 3.3 is not satisfied by this network. Note, however, this is no surprise given that this naively-trained network is susceptible to adversarial examples; that is, minute changes in its input can change the output of the network drastically. In the next section, we will develop an alternative training method to produce richer sets of activations in the hidden-layers of the network.

3.4 Reinterpreting Regularization for Single-layer and Multi-layer Models

When a linear model is trained on a rank-deficient data set, or when a linear model is desired to be robust against small perturbations in its input, the classical procedure is introducing a regularization term to the training loss function. This regularization typically involves penalizing some norm of the model parameters, and it is considered to assign a prior distribution on the values the parameters can take or to limit the class of functions that can be learned.

We can provide a dual interpretation for these regularization terms in terms of boosting the richness of the training data, as stated in the following theorem.

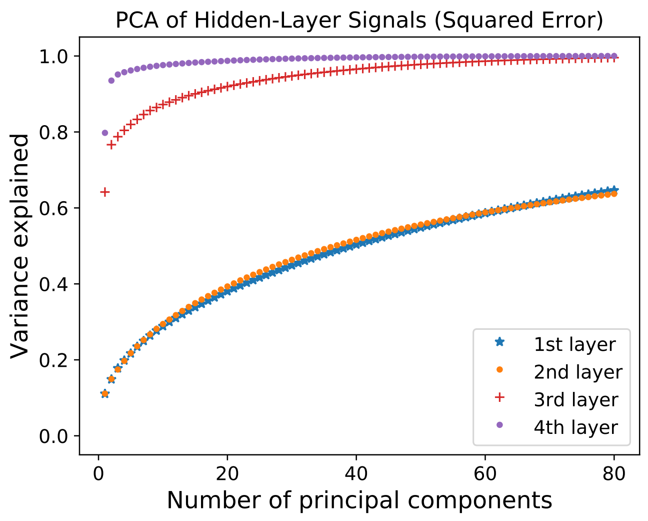


Figure 3.1: Principal component analysis for the activations in the hidden layers of a 5-layer convolutional neural network trained with the squared error loss. The plot displays the variance explained by the first 80 principal components for each layer. The layers have 4704, 1800, 120 and 84 nodes, respectively. The rank of the activation patterns in the upper layers are much lower than the number of nodes in those layers.

Theorem 3.4. *Given a set of points $\{x_i\}_{i\in\mathcal{I}}$ in $\mathbb{R}^n$ and corresponding target values $\{y_i\}_{i\in\mathcal{I}}$ in $\mathbb{R}$, consider the following two problems:*

$$\min_w \sum\nolimits_{i\in\mathcal{I}} (y_i - w^\top x_i)^2 + \lambda\|w\|_p^m, \tag{3.4a}$$

$$\min_w \sum\nolimits_{i\in\mathcal{I}} \frac{1}{2}\left(y_i - \min_{d:\|d\|_q\le\epsilon} w^\top(x_i+d)\right)^2 + \frac{1}{2}\left(y_i - \max_{d:\|d\|_q\le\epsilon} w^\top(x_i+d)\right)^2, \tag{3.4b}$$

where $\|\cdot\|_p$ and $\|\cdot\|_q$ are dual norms, $m\in[1,\infty)$ is some fixed number, and $\lambda,\epsilon\in(0,\infty)$ are hyperparameters. For each λ, there exists some ϵ such that the solutions of the two problems are identical. Conversely, for each ϵ, there exists some λ such that the solutions of the two problems are identical. □

Problem (3.4b) shows that for each training point, perturbing the input of a linear mapping in two directions such that the output of the mapping is maximized and minimized creates a richness in the training data that is equivalent to penalizing the norm of the parameters. An analogous procedure for multi-layer neural networks is inserting perturbations to every hidden-layer of the network preceding the affine operations. To illustrate, consider an L-layer feedforward network:

$$h_0(x) = x, \tag{3.5a}$$

$$h_j(x) = (W_j h_{j-1}(x) + b_j)_+ \quad j = 1,2,\ldots,L, \tag{3.5b}$$

where $\{W_j\}_{j\in[L]}$ and $\{b_j\}_{j\in[L]}$ are the weight and bias parameters of the network, and $(\cdot)_+$ is the ReLU operation. The parameter W_j corresponds to a linear operation in the j-th layer of the network, such as a matrix multiplication or a convolution. We can insert perturbations to this network as

$$\tilde{h}_0(x;d) = x, \tag{3.6a}$$

$$\tilde{h}_j(x;d) = (W_j[\tilde{h}_{j-1}(x;d) + d_j] + b_j)_+ \quad j = 1, 2, \dots, L, \tag{3.6b}$$

where $d = (d_1, d_2, \dots, d_L)$ is the concatenation of the perturbations applied to each layer of the network. Then, solving the regression problem with the cost function

$$\sum_{i\in\mathcal{I}} \frac{1}{2}\left(y_i - \min_{d\in\mathcal{D}} \tilde{h}_L(x_i;d)\right)^2 + \frac{1}{2}\left(y_i - \max_{d\in\mathcal{D}} \tilde{h}_L(x_i;d)\right)^2,$$

where $\mathcal{D}$ is the allowed set of perturbations, should force the output of the network to remain close to the target values despite changes in the input and in the activations in the hidden layers.

Before proceeding to the next section, we summarize in Algorithm 1 the procedure for training a neural network while ensuring persistent excitation of the parameters. For simplicity, the algorithm is described for the stochastic gradient method with momentum.

Algorithm 1 Training with Persistent Excitation

```
1: input: training data $\{(x_i, y_i)\}_{i\in\mathcal{I}}$,
          neural network $f_\theta(x;d) \equiv \tilde{h}_L(x;d)$ in (3.6),
          set of allowed perturbations $\mathcal{D}$,
          learning rate $\eta$, momentum parameter $\gamma$
2: initialize: $\Delta\theta \leftarrow 0$
3: repeat
4:     randomly choose $i \in \mathcal{I}$
5:     $d_1 \leftarrow \text{argmax}_{d\in\mathcal{D}} f_\theta(x_i;d)$
6:     $d_2 \leftarrow \text{argmin}_{d\in\mathcal{D}} f_\theta(x_i;d)$
7:     $\Delta\theta \leftarrow \gamma\Delta\theta + (1-\gamma)\nabla_\theta\left[(f_\theta(x_i;d_1) - y_i)^2 + (f_\theta(x_i;d_2) - y_i)^2\right]$
8:     $\theta \leftarrow \theta - \eta\Delta\theta$
9: until training is complete
```

3.5 Experimental Results

In this section, we test Algorithm 1 on a binary classification task. Only two classes of images, the horses and the planes, have been chosen from the CIFAR-10 data set for the classification task. The same network architecture is used in all of the experiments: two convolutional layers followed by three fully-connected layers with leaky-ReLu activations. Neither batch-normalization nor dropout is implemented in the experiments.

The first experiment is to see the effectiveness of Algorithm 1. We train the convolution network described with Algorithm 1 by using three different sets of perturbation, which are ℓ_∞ balls with radii 0.005, 0.010 and 0.020 on all layers. The plots in Figure 3.2 show the percentage of points the network misclassify versus the amount of disturbance needed in the input of the network to cause misclassification at evaluation phase, which is computed by using the projected gradient attack algorithm (Madry et al., 2018; Rauber et al., 2017). In this sense, the plots represent the cumulative distribution of the margin of the data; and the lower the plot, the more robust the network.

We observe that applying a small perturbation during training phase increases the margin of the training data conspicuously. We also observe that the training error is not able to reach zero when the perturbation is relatively large, as shown by the curve for 0.020 perturbation[1]. Nevertheless, the same magnitude of perturbation attains the largest margin on the test data.

The second experiment is to demonstrate the necessity of perturbing all layers, and not only the first layer, in order to ensure robustness of the network against small perturbations in its input. Figure 3.3 shows the margin distribution of the same network trained in two different ways: perturbing all layers of the network as described in Algorithm 1 and perturbing only the input of the network during training, similar to adversarial training (Madry et al., 2018). For both cases, the perturbations are restricted to be in ℓ_∞ ball with radius 0.020. We observe that perturbing only the first layer of the network during training substantially increases the margin of the training data; however, this does not correspond to an improvement for the margin of the test data. In contrast, when all layers are perturbed as described in Algorithm 1 to improve the richness of the activations in the hidden layers, the margin of the training data becomes a good indicator of the margin of the test data.

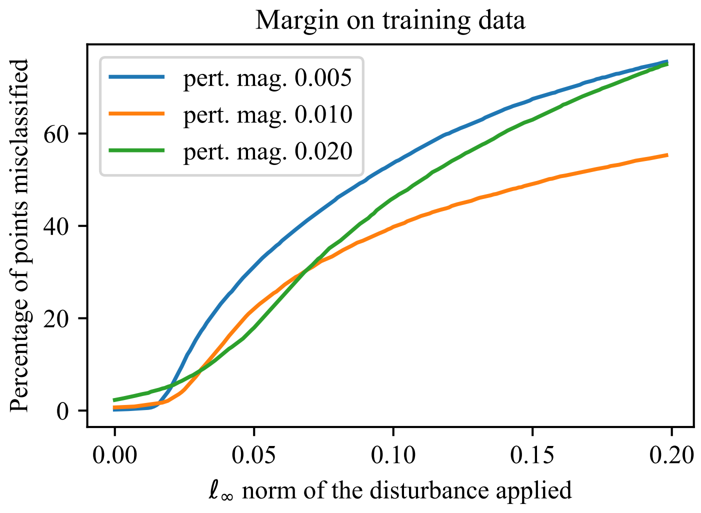
3.6 Discussion

In this section, we compare our results to closely related works.

Dropout. Using dropout in the hidden-layer neurons, that is, setting the output of random subsets of neurons to zero during training, is known to prevent overfitting (Srivastava et al., 2014). This can be reinterpreted based on the richness of hidden-layer activations. Randomly setting some of the neurons in the hidden layers during training increases the variety in the hidden-layer activations, thereby improving the richness of the signals that excite the parameters during training. Nevertheless, the space where the random perturbations is needed is very high-dimensional (it is equal to the dimension of the input space plus the total number of hidden-layer nodes in all layers of the network), and it is difficult to fill up the neighborhood of the training data in this space with random perturbations.

Importance of low-dimensional activations. The fact that the activations in the hidden layers of a naively-trained neural network will become low-dimensional is critical in realizing that augmenting the training data set may not help achieving robustness in neural networks. This is because adding more training data will not necessarily be effective for attaining the required richness in the

[1]The size of the network was fixed for all magnitudes of perturbations.



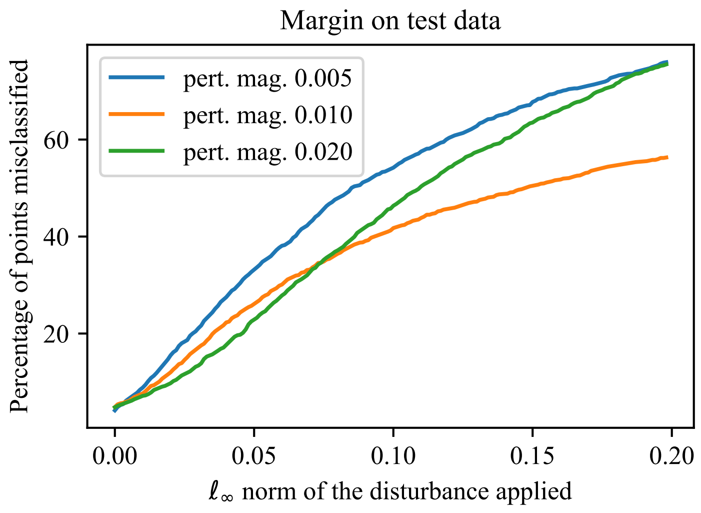


Figure 3.2: The effect of training with persistent excitation as described in Algorithm 1. Small perturbations during training improves the margin for the training data. The larger perturbations prevent the training error from reaching zero; however, they yield larger margins on the test data.

hidden layers, and consequently, the parameters may not be trained robustly. However, if the low-dimensionality of the hidden-layer activations is overlooked, the conclusion will be different. For example, Schmidt et al. (2018) analyzes the robustness of classifiers with full-rank, non-degenerate data sets and single-layer, linear models, and arrives at the conclusion that the robustness could be achieved with more training data.

Adversarial training. Madry et al. (2018) proposed adversarial training to ensure robustness of neural networks by adding adversarial perturbations to the training data. It was observed that this

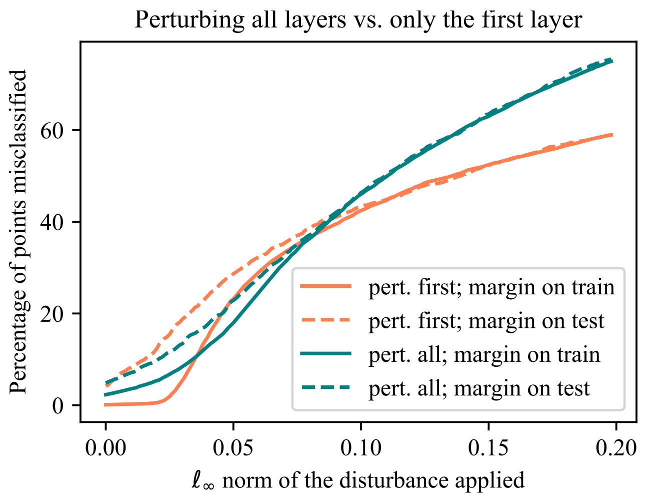


Figure 3.3: The same network is trained in two different ways: by perturbing every layer of the network as described in Algorithm 1 in order to boost the richness of the hidden-layer activations, and by perturbing only the training data (the first layer) similar to adversarial training. Perturbing only the first layer substantially increases the margin for the training data; however, this is not reflected in the test data. In contrast, when all layers are perturbed for improving the richness of the hidden-layer activations as outlined in Algorithm 1, the margin of the training data becomes a good indicator of the margin of the test data.

method substantially increased the margin of the neural networks between their decision boundaries and the *training* data points, but not the margin of the test data. Note that adversarial training emulates augmenting the training data set — and only the training data set. This strategy does not necessarily prevent the activations in the hidden layers of the network from becoming low dimensional during training, and consequently, richness requirements on the hidden-layer activations of the network cannot be guaranteed.

All-layer margin and generalization. Wei and Ma (2020) recently showed that a new concept of margin, which is computed for multi-layer models by allowing perturbations in the hidden-layers as well as in the inputs, can be used for obtaining generalization bounds for these models. This is closely aligned with our results involving the richness requirements on the hidden-layer activations of multi-layer models and the effect of boosting this richness by injecting perturbations into the hidden layers during training.

3.7 Conclusion

For models with parameters that act on the fixed features of training data linearly, augmenting the training data set, adding random or adversarial perturbations to the training data, or using robust optimization techniques are effective methods to achieve robustness against small perturbations in the inputs of these models (El Ghaoui and Lebret, 1997; Bertsimas et al., 2011). Note that all of these methods could be considered as an approach to meet the richness condition on the training data set. In this chapter, we showed that for multi-layer structures, the richness condition required for implicit regularization is not only on the training data set, but also on the activations in the intermediate layers of the model. Understanding these distinct requirements on the richness of activations on multi-layer structures could open up a direction to find effective regularization methods for neural networks and improve their robustness.

3.8 Proofs and Further Remarks

In this section, we provide the proofs for the theorems and corollaries of this chapter and elaborate on some of the remarks.

3.8.1 Proof of Theorem 3.1

Given $\{x_i\}_{i\in\mathcal{I}}$, define $\Sigma = \sum_{i\in\mathcal{I}} x_i x_i^\top$. Define the training loss function

$$\ell(W_1,\dots,W_L) = \frac{1}{2}\sum_{i\in\mathcal{I}} \|W_L \cdots W_2 W_1 x_i - y_i\|_2^2.$$

The gradient descent algorithm can be written as

$$W_j \leftarrow W_j - \delta \frac{\partial \ell(W_1,\dots,W_L)}{\partial W_j} \quad \forall j \in [L], \tag{3.7}$$

where δ is the learning rate of the algorithm.

For the iterations of the gradient descent algorithm to converge to the solution $(\hat{W}_1,\dots,\hat{W}_L)$ from almost every point in its neighborhood, it is necessary that the solution $(\hat{W}_1,\dots,\hat{W}_L)$ be stable in the sense of Lyapunov (Sastry, 2013). For nonlinear dynamical systems like (3.7), a necessary condition for the Lyapunov stability of $(\hat{W}_1,\dots,\hat{W}_L)$ is given by the dynamics of the linear approximation of (3.7) around that equilibrium. Consider the linearization of (3.7) around $(\hat{W}_1,\dots,\hat{W}_L)$:

$$\tilde{W}_j \leftarrow \tilde{W}_j - \delta \sum_{i\in[L]} f_{i,j}(\tilde{W}_i) \quad \forall j \in [L], \tag{3.8}$$

where $f_{i,j}$ denotes the Jacobian with respect to W_i of the gradient with respect to W_j of the loss function ℓ at $(\hat{W}_1,\dots,\hat{W}_L)$. Then, for $(\hat{W}_1,\dots,\hat{W}_L)$ to be a stable equilibrium of (3.7), all eigenvalues of the mapping in (3.8) must be less than or equal to 1 in magnitude (Sastry, 2013). Note that $f_{i,j}$ and $f_{j,i}$ are the derivatives of the same function with respect to the same variables in different orders; consequently, the linear system (3.8) can be represented by a symmetric matrix.

Since system (3.8) can be represented by a symmetric matrix, its eigenvalues being less than 1 in magnitude implies that all eigenvalues of its diagonal blocks are also less than 1 in magnitude. In other words, Lyapunov stability of $(\hat{W}_1,\dots,\hat{W}_L)$ implies that the eigenvalues of

$$\tilde{W}_j \leftarrow \tilde{W}_j - \delta f_{j,j}(\tilde{W}_j)$$

are less 1 in magnitude for all $j \in [L]$. Equivalently, it implies that the eigenvalues of

$$\tilde{W}_j \leftarrow f_{j,j}(\tilde{W}_j)$$

are less than $\frac{2}{\delta}$ for all $j \in [L]$.

To obtain a more explicit expression for $f_{j,j}$, note that the update for the gradient descent algorithm is:

$$\begin{aligned} W_j \leftarrow W_j - \delta \sum_{i \in \mathcal{I}} & W_{j+1}^\top \cdots W_L^\top W_L \cdots W_1 x_i x_i^\top W_1^\top \cdots W_{j-1}^\top \\ + \delta \sum_{i \in \mathcal{I}} & W_{j+1}^\top \cdots W_L^\top y_i x_i^\top W_1^\top \cdots W_{j-1}^\top, \end{aligned}$$

and $f_{j,j}$ is given by

$$\begin{aligned} \tilde{W}_j &\leftarrow \sum_{i \in \mathcal{I}} \hat{W}_{j+1}^\top \cdots \hat{W}_L^\top \hat{W}_L \cdots \hat{W}_{j+1} \tilde{W}_j \hat{W}_{j-1} \cdots \hat{W}_1 x_i x_i^\top \hat{W}_1^\top \cdots \hat{W}_{j-1}^\top \\ &\leftarrow \left(\hat{W}_{j+1}^\top \cdots \hat{W}_L^\top \hat{W}_L \cdots \hat{W}_{j+1}\right) \tilde{W}_j \left(\hat{W}_{j-1} \cdots \hat{W}_1 \Sigma \hat{W}_1^\top \cdots \hat{W}_{j-1}^\top\right) \end{aligned}$$

for all $j \in [L]$, where the last line follows from the fact that $\sum_{i \in \mathcal{I}} x_i x_i^\top = \Sigma$. The largest eigenvalue of this linear mapping is given by

$$\lambda_{\max}(f_{j,j}) = \lambda_{\max}\left(\hat{W}_{j+1}^\top \cdots \hat{W}_L^\top \hat{W}_L \cdots \hat{W}_{j+1}\right) \lambda_{\max}\left(\hat{W}_{j-1} \cdots \hat{W}_1 \Sigma \hat{W}_1^\top \cdots \hat{W}_{j-1}^\top\right).$$

Note that the last term can be lower bounded:

$$\lambda_{\max}\left(\hat{W}_{j-1} \cdots \hat{W}_1 \Sigma \hat{W}_1^\top \cdots \hat{W}_{j-1}^\top\right) \geq \lambda_{\max}\left(\hat{W}_{j-1} \cdots \hat{W}_1 \hat{W}_1^\top \cdots \hat{W}_{j-1}^\top\right) \lambda_{\min}(\Sigma).$$

Therefore, if $\lambda_{\max}(f_{j,j}) \leq \frac{2}{\delta}$ for all $j \in [L]$, we have

$$\begin{aligned} \frac{2}{\delta} &\geq \lambda_{\max}(f_{j,j}) \\ &= \lambda_{\max}\left(\hat{W}_{j+1}^\top \cdots \hat{W}_L^\top \hat{W}_L \cdots \hat{W}_{j+1}\right) \lambda_{\max}\left(\hat{W}_{j-1} \cdots \hat{W}_1 \Sigma \hat{W}_1^\top \cdots \hat{W}_{j-1}^\top\right) \\ &\geq \lambda_{\max}\left(\hat{W}_{j+1}^\top \cdots \hat{W}_L^\top \hat{W}_L \cdots \hat{W}_{j+1}\right) \lambda_{\max}\left(\hat{W}_{j-1} \cdots \hat{W}_1 \hat{W}_1^\top \cdots \hat{W}_{j-1}^\top\right) \lambda_{\min}(\Sigma) \\ &= \rho^2\left(\hat{W}_L \cdots \hat{W}_{j+1}\right) \rho^2\left(\hat{W}_{j-1} \cdots \hat{W}_1\right) \lambda_{\min}(\Sigma) \qquad \forall j \in [L], \end{aligned}$$

where $\rho(\cdot)$ represents the largest singular value of its argument. This shows that the Lyapunov stability of $(\hat{W}_1, \ldots, \hat{W}_L)$ implies that

$$\sqrt{\frac{2}{\delta \lambda_{\min}(\Sigma)}} \geq \rho(\hat{W}_L \cdots \hat{W}_{j+1}) \rho(\hat{W}_{j-1} \cdots \hat{W}_1) \quad \forall j \in [L].$$

Since

$$\rho(\hat{W}_L \cdots \hat{W}_1) \leq \rho(\hat{W}_L \cdots \hat{W}_{j+1})\rho(\hat{W}_j)\rho(\hat{W}_{j-1} \cdots \hat{W}_1),$$

we also have

$$\sqrt{\frac{2}{\delta \lambda_{\min}(\Sigma)}} \geq \frac{\rho(\hat{W}_L \cdots \hat{W}_1)}{\rho(\hat{W}_j)} \quad \forall j \in [L].$$

This implies

$$\sqrt{\frac{2}{\delta \lambda_{\min}(\Sigma)}} \geq \frac{\rho(\hat{W}_L \cdots \hat{W}_1)}{\min_{j\in[L]} \rho(\hat{W}_j)} \geq \frac{\rho(\hat{W}_L \cdots \hat{W}_1)}{\rho^{1/L}(\hat{W}_L \cdots \hat{W}_1)} = \rho^{(L-1)/L}(\hat{W}_L \cdots \hat{W}_1).$$

As a result, we obtain

$$\rho(\hat{W}_L \cdots \hat{W}_1) \leq \left(\frac{2}{\delta \lambda_{\min}(\Sigma)}\right)^{L/(2L-2)}.$$

■

3.8.2 Proof of Theorem 3.2 and Corollary 3.1

Proof of Theorem 3.2

To begin with, assume b is fixed and not updated by the gradient descent algorithm. Let $(\hat{W}, \hat{V})$ denote the local optimum that the algorithm has converged to. For point x_i, let $G_i \in \{0,1\}^{r\times r}$ be the diagonal matrix that satisfies

$$W(Vx_i + b)_+ = WG_i(Vx_i + b).$$

The update rule for the gradient descent algorithm *calculated with automatic differentiation* is given as

$$W \leftarrow W - \delta \left\{\sum\nolimits_{i\in\mathcal{I}} \left[WG_i(Vx_i + b) - f(x_i)\right](Vx_i + b)^\top G_i^\top\right\}, \tag{3.9a}$$

$$V \leftarrow V - \delta \left\{\sum\nolimits_{i\in\mathcal{I}} G_i^\top W^\top \left[WG_i(Vx_i + b) - f(x_i)\right] x_i^\top\right\}, \tag{3.9b}$$

followed by

$$(G_i)_{kk} \leftarrow \mathbb{I}\{V_k x_i + b_k > 0\} \quad \forall k \in [r],\ \forall i \in \mathcal{I}, \tag{3.9c}$$

where $(G_i)_{kk}$ denotes the k-th diagonal element of the matrix G_i, and V_k and b_k represent the k-th rows of V and b, respectively. If the algorithm has converged, switching between 1 and 0 must have ended in a finite time. Therefore, at an equilibrium point $(\hat{W}, \hat{V})$ with activations $\{\hat{G}_i\}_{i\in\mathcal{I}}$, we will assume $\hat{G}_i$ is fixed for all $i \in \mathcal{I}$.

Like all dynamical systems, the gradient descent algorithm (3.9) can converge to an equilibrium from randomly chosen close neighbors of this point only if the equilibrium is stable in the sense of Lyapunov (Sastry, 2013). A necessary condition for an equilibrium of a nonlinear dynamical system to be stable is that the linear approximation of this dynamical system not be unstable around the same equilibrium. We will use this fact to obtain a necessary condition for the algorithm to converge to an equilibrium from any randomly chosen initial point.

Linearization of the system (3.9) around the equilibrium $(\hat{W}, \hat{V})$ gives

$$\tilde{W} \leftarrow \tilde{W} - f_1(\tilde{W}) - f_2(\tilde{V}), \tag{3.10a}$$

$$\tilde{V} \leftarrow \tilde{V} - f_3(\tilde{W}) - f_4(\tilde{V}), \tag{3.10b}$$

where

$$\begin{aligned} f_1(\tilde{W}) &= \delta \sum_{i\in\mathcal{I}} \tilde{W}\hat{G}_i(\hat{V}x_i + b)(\hat{V}x_i + b)^\top \hat{G}_i^\top, \\ f_2(\tilde{V}) &= \delta \sum_{i\in\mathcal{I}} \hat{W}\hat{G}_i\tilde{V}x_i x_i^\top \hat{V}\hat{G}_i^\top + \delta \sum_{i\in\mathcal{I}} [\hat{W}\hat{G}_i(\hat{V}x_i + b) - f(x_i)]x_i^\top \tilde{V}^\top \hat{G}_i^\top, \\ f_3(\tilde{W}) &= \delta \sum_{i\in\mathcal{I}} \hat{G}_i^\top \hat{W}^\top \tilde{W}\hat{G}_i(\hat{V}x_i + b)x_i^\top + \delta \sum_{i\in\mathcal{I}} \hat{G}_i^\top \tilde{W}^\top [\hat{W}\hat{G}_i(\hat{V}x_i + b) - f(x_i)]x_i^\top, \\ f_4(\tilde{V}) &= \delta \sum_{i\in\mathcal{I}} \hat{G}_i^\top \hat{W}^\top \hat{W}\hat{G}_i\tilde{V}x_i x_i^\top. \end{aligned}$$

We have the equality

$$\langle \tilde{W}, f_2(\tilde{V})\rangle = \langle f_3(\tilde{W}), \tilde{V}\rangle \quad \forall \tilde{W} \in \mathbb{R}^{m\times r},\ \forall \tilde{V} \in \mathbb{R}^{r\times n};$$

therefore, the linearized system (3.10) can be represented by a symmetric matrix[2], and the system (3.10) is stable in the sense of Lyapunov only if the matrix corresponding to the system

$$\tilde{W} \leftarrow \tilde{W} - f_1(\tilde{W}) \tag{3.11a}$$

$$\tilde{V} \leftarrow \tilde{V} - f_4(\tilde{V}) \tag{3.11b}$$

has all of its eigenvalues inside the unit circle. This implies that the convergence of the system (3.10) requires the largest eigenvalues of the mappings $f_1(\cdot)$ and $f_4(\cdot)$ to be smaller than 2.

The largest eigenvalue of the mapping $f_1(\cdot)$ being smaller than 2 implies that

$$e_k^\top \sum\nolimits_{i\in\mathcal{I}} \hat{G}_i(\hat{V}x_i + b)(\hat{V}x_i + b)^\top \hat{G}_i^\top e_k \le \frac{2}{\delta} \quad \forall k \in [r],$$

where e_k denotes the standard basis vector with 1 in its k-th element. Then,

$$e_k^\top \sum\nolimits_{i\in\mathcal{I}_k} (\hat{V}x_i + b)(\hat{V}x_i + b)^\top e_k \le \frac{2}{\delta} \quad \forall k \in [r]$$

where $\mathcal{I}_k$ denotes the set of indices of the points that activate node k at equilibrium, i.e.,

$$\mathcal{I}_k = \{i \in \mathcal{I} : e_k^\top(\hat{V}x_i + b) > 0\}.$$

[2]This is also a direct consequence of the fact that f_2 and f_3 are the Jacobians of the gradient of the training loss function with respect to the same parameters in different orders.

Let $\hat{V}_k$ and b_k denote the k-th rows of $\hat{V}$ and b. Then we need

$$\sum\nolimits_{i\in\mathcal{I}_k} (\hat{V}_k x_i + b_k)(x_i^\top \hat{V}_k^\top + b_k) \leq \frac{2}{\delta},$$

or equivalently,

$$\hat{V}_k^\top \left(\sum\nolimits_{i\in\mathcal{I}_k} x_i x_i^\top\right) \hat{V}_k + 2b_k \hat{V}_k^\top \left(\sum\nolimits_{i\in\mathcal{I}_k} x_i\right) + b_k^2 - \frac{2}{\delta} \leq 0.$$

This is a quadratic inequality in $\hat{V}_k$, and the largest value $\left\|\hat{V}_k\right\|_2$ can take is upper bounded by the larger root of

$$\lambda_{\text{min}}\left(\sum\nolimits_{i\in\mathcal{I}_k} x_i x_i^\top\right) \left\|\hat{V}_k\right\|_2^2 - 2\left|b_k\right| \left\|\sum\nolimits_{i\in\mathcal{I}_k} x_i\right\|_2 \left\|\hat{V}_k\right\|_2 + b_k^2 - \frac{2}{\delta} = 0.$$

Therefore, we have

$$\left\|\hat{V}_k\right\|_2 \leq \frac{|b_k|\,\mu_k}{\lambda_k^{\text{min}}} + \frac{1}{\lambda_k^{\text{min}}}\sqrt{b_k^2(\mu_k)^2 + \lambda_k^{\text{min}}\left(\frac{2}{\delta} - b_k^2\right)}$$

where

$$\mu_k = \left\|\sum\nolimits_{i\in\mathcal{I}_k} x_i\right\|_2,$$
$$\lambda_k^{\text{min}} = \lambda_{\text{min}}\left(\sum\nolimits_{i\in\mathcal{I}_k} x_i x_i^\top\right).$$

As an upper bound independent of k, we can write

$$\left\|\hat{V}_k\right\|_2 \leq \frac{\overline{\mu}\|b\|_\infty}{\underline{\lambda}} + \sqrt{\frac{\overline{\mu}^2\|b\|_\infty^2}{\underline{\lambda}^2} + \frac{1}{\underline{\lambda}}\left(\frac{2}{\delta} - \|b\|_\infty^2\right)} \qquad \forall k \in [r]$$

where $\overline{\mu} = \max_{k\in[r]} \mu_k$ and $\underline{\lambda} = \min_{k\in[r]} \lambda_k^{\text{min}}$.

So far we have only used the fact that the largest eigenvalue of $f_1(\cdot)$ is less than 2. Similarly, the largest eigenvalue of $f_4(\cdot)$ being less than 2 implies that

$$\sum\nolimits_{i\in I_k} e_k^\top \hat{W}^\top \hat{W} e_k \cdot \lambda_{\text{min}}\left(\sum\nolimits_{i\in\mathcal{I}_k} x_i x_i^\top\right) \leq \frac{2}{\delta}.$$

If $\hat{W}_k$ denotes the k-th column of $\hat{W}$, we have

$$\left\|\hat{W}_k\right\|_2 \leq \sqrt{\frac{2}{\delta\underline{\lambda}}} \qquad \forall k \in [r].$$

Given the estimates $\hat{W}$ and $\hat{V}$, for every $x \in \mathbb{R}^n$, the function estimated by the network is $\hat{W}(\hat{V}x + b)_+$, and the Lipschitz constant of this estimate is bounded by

$$\max_{x\in\mathbb{R}^n} \sum\nolimits_{k=1}^{r} \mathbb{I}\left\{\hat{V}_k x + b_k > 0\right\} \left\|\hat{W}_k \hat{V}_k\right\|_F,$$

which is further bounded by

$$n_{\text{active}}^{\max}\left(\sqrt{\frac{2}{\delta\underline{\lambda}}}\right)\left(\frac{\overline{\mu}\|b\|_\infty}{\underline{\lambda}}+\sqrt{\frac{\overline{\mu}^2\|b\|_\infty^2}{\underline{\lambda}^2}+\frac{1}{\underline{\lambda}}\left(\frac{2}{\delta}-\|b\|_\infty^2\right)}\right)$$

and

$$n_{\text{active}}^{\max}\left(\sqrt{\frac{2}{\delta\underline{\lambda}}}\right)\left(\frac{2\overline{\mu}\|b\|_\infty}{\underline{\lambda}}+\sqrt{\frac{1}{\underline{\lambda}}\left|\frac{2}{\delta}-\|b\|_\infty^2\right|}\right)$$

where $n_{\text{active}}^{\max}$ is the maximum number of nodes that a point in $\mathbb{R}^n$ can activate, i.e.,

$$n_{\text{active}}^{\max}=\max_{x\in\mathbb{R}^n}\sum\nolimits_{k=1}^{r}\mathbb{I}\left\{\hat{V}_k x+b_k>0\right\}.$$

To complete the proof, now assume that b is not fixed and it is also updated by the gradient descent algorithm. We can write the linearization of the update rule for (W,V,b) as

$$\begin{bmatrix}W\\V^\top\end{bmatrix}\leftarrow\begin{bmatrix}W\\V^\top\end{bmatrix}-g_1\left(\begin{bmatrix}W\\V^\top\end{bmatrix}\right)-g_2\left(b\right), \tag{3.12a}$$

$$b\leftarrow b-g_3\left(\begin{bmatrix}W\\V^\top\end{bmatrix}\right)-g_4\left(b\right), \tag{3.12b}$$

where g_1,g_2,g_3 and g_4 are the linear operators obtained by taking Jacobians of the gradients of the training loss function with respect to W, V and b. Similar to f_2 and f_3 in system (3.10), the operators g_2 and g_3 are the Hermitian of each other, and therefore, the matrix corresponding to the system (3.12) is still symmetric. As a result, its eigenvalues are less than 1 in magnitude only if its diagonal sub-blocks have eigenvalues less than 1 in magnitude, which leads to the identical condition for the case with fixed b. ■

Proof of Corollary 3.1

Given the training data set $\{x_i\}_{i\in\mathcal{I}}$ in $\mathbb{R}^n$, assume that the parameters of the two-layer neural network in Theorem 3.2 have converged to $(\hat{W},\hat{V},\hat{b})$ with activations $\{\hat{G}_i\}_{i\in\mathcal{I}}$. Without loss of generality, assume the first node in the hidden layer is not activated by a full-rank data set at equilibrium:

$$\operatorname{rank}\left\{x_i : i\in\mathcal{I},\ e_1^\top(\hat{V}x_i+\hat{b})>0\right\}<n.$$

Let $\eta\in\mathbb{R}^n$ be a nonzero vector orthogonal to this set of points activating the first node in the hidden-layer:

$$\langle\eta,x\rangle=0\quad\forall x\in\left\{x_i : i\in\mathcal{I},\ e_1^\top(\hat{V}x_i+\hat{b})>0\right\}.$$

Then, at the equilibrium, the first row of $\hat{V}$ could have arbitrarily large component in the direction of η. In other words, convergence of the gradient descent algorithm does not provide a bound for $|e_1^\top\hat{V}\eta|$. ■

3.8.3 Dependence of Implicit Regularization on Width of Nonlinear Networks

Consider, for example, a two-layer network with r hidden-layer nodes and the parameters $W \in \mathbb{R}^{m\times r}$, $V \in \mathbb{R}^{r\times n}$, $b \in \mathbb{R}^r$, and the following training data set in $\mathbb{R}^n$:

$$\begin{aligned} x_1 &= e_1, \\ x_j &= \cos(\theta)e_1 + \sin(\theta)e_j \quad && \forall j \in \{2,\dots,r\}, \\ \tilde{x}_i &= e_{i+1} && \forall i \in \{1,\dots,n-1\}, \end{aligned}$$

where e_i denotes the i-th standard basis vector of $\mathbb{R}^n$ for each $i \in [n]$, and θ is some small angle satisfying $0 < \theta \ll \pi/2$. Assume that at the end of training, the parameters of the network become $(\hat{W}, \hat{V}, \hat{b})$, the column space of $\hat{W}$ attains rank 1, and the j-th node in the hidden-layer is activated only by $\{x_j\} \cup \{\tilde{x}_i\}_{i\in[n-1]}$ for all $j \in [r]$:

$$\begin{aligned} \hat{V}_j x_j + \hat{b}_j &> 0 \quad \forall j \in [r], \\ \hat{V}_j x_i + \hat{b}_j &\le 0 \quad \forall i \in [r]\backslash\{j\},\ \forall j \in [r], \\ \hat{V}_j \tilde{x}_i + \hat{b}_j &> 0 \quad \forall i \in [n-1],\ \forall j \in [r]. \end{aligned}$$

As θ is close to zero, there will be a point in the direction of e_1 that simultaneously activates all of the hidden-layer nodes — although there was no point in the training data set that activated all of the nodes and that was aligned with e_1. Consequently, the change in the network output with respect to change in the input around this point will require a multiplier of r, which stands for $n_{\text{active}}^{\max}$ in Theorem 3.2.

3.8.4 Proof of Theorem 3.3

Restatement of Theorem 3.3. *Consider an L-layer network with ReLU activations:*

$$\begin{aligned} h_0(x) &= x, \\ h_j(x) &= (W_j h_{j-1}(x))_+ \quad j = 1,2,\cdots,L-1, \\ h_L(x) &= W_L h_{L-1}(x) \end{aligned}$$

with n_j nodes in its j-th hidden layer, and assume it has been trained by minimizing the squared-error loss with the gradient descent algorithm on the data set $\{x_i\}_{i\in\mathcal{I}}$. Let $\hat{W}_j$ and $\hat{h}_j$ denote the weight matrix and the output of the j-th layer after the training, and define $\mathcal{I}' = \{i \in \mathcal{I} : \hat{h}_L(x_i) \neq 0\}$. Define $\mathcal{I}_j^k$ as the set of points that activate the k-th node in the j-th layer of the network after training:

$$\mathcal{I}_j^k = \{i \in \mathcal{I}' : e_k^\top \hat{h}_j(x_i) > 0\} \quad \forall k \in [n_j],\ \forall j \in [L-1].$$

Assume all hidden-layer activations are bounded over the training data set:

$$\max_{i\in\mathcal{I}'} \max_{j\in[L]} \|\hat{h}_j(x_i)\|_2 < \infty,$$

and every hidden-layer node is activated by a set of signals with full-rank in the preceding layer:

$$\textstyle\sum_{i\in\mathcal{I}_j^k}\hat{h}_{j-1}(x_i)\hat{h}_{j-1}^\top(x_i)\succ 0\quad\forall k\in[n_j],\forall j\in[L].$$

Then the convergence of the gradient descent algorithm from almost every initialization implies that

$$\|\hat{W}_j\|_2^2\leq\frac{2}{\delta}\sum_{k\in[n_j]}\frac{1}{\lambda_{\min}(\sum_{i\in\mathcal{I}_j^k}\hat{h}_{j-1}(x_i)\hat{h}_{j-1}^\top(x_i))}\sum_{i\in\mathcal{I}_j^k}\frac{\|\hat{h}_j(x_i)\|_2^2}{\|\hat{h}_L(x_i)\|_2^2}\quad\forall j\in[L-1],$$

and

$$\|\hat{W}_L\|_2^2\leq\frac{2}{\delta}\sum_{k\in[n_{L-1}]}\frac{1}{\max_{i\in\mathcal{I}_{L-1}^k}\|\hat{h}_{L-2}(x_i)\|_2^2}.\qquad\square$$

Similar to the proof of Theorem 3.2, given the neural network

$$\begin{aligned}&h_0(x)=x,\\&h_j(x)=(W_jh_{j-1}(x))_+\quad j=1,2,\ldots,L-1,\\&h_L(x)=W_Lh_{L-1}(x),\end{aligned}$$

for each point in $\{x_i\}_{i\in\mathcal{I}}$ and for each layer $j\in[L-1]$, define the diagonal activation matrix G_j^i as

$$(G_j^i)_{kk}=\begin{cases}1 & \text{if } e_k^\top W_jh_{j-1}(x_i)>0,\\ 0 & \text{otherwise,}\end{cases}$$

where e_k is the k-th standard basis vector in $\mathbb{R}^{n_j}$ for each $k\in[n_j]$. Then we can write

$$h_L(x_i)=W_LG_{L-1}^iW_{L-1}G_{L-2}^iW_{L-2}\cdots G_1^iW_1x_i\quad\forall i\in\mathcal{I}.$$

The training loss function is

$$\frac{1}{2}\sum_{i\in\mathcal{I}}\|h_L(x_i)-y_i\|_2^2,$$

where $\{y_i\}_{i\in\mathcal{I}}$ is the set of target values corresponding to the input points. For each weight matrix W_j, can write the gradient descent algorithm calculated with automatic differentiation as

$$\begin{aligned}W_j\leftarrow W_j-\delta\textstyle\sum_{i\in\mathcal{I}}\big(&G_j^iW_{j+1}^\top G_{j+1}^i\cdots W_L^\top W_L\cdots G_{j+1}^iW_{j+1}G_j^iW_jG_{j-1}^iW_{j-1}\cdots\\&\cdots G_1^iW_1x_ix_i^\top W_1^\top G_1^i\cdots W_{j-1}^\top G_{j-1}^i\big)\\+\delta\textstyle\sum_{i\in\mathcal{I}}\big(&G_j^iW_{j+1}^\top G_{j+1}^i\cdots W_L^\top y_ix_i^\top W_1^\top G_1^i\cdots W_{j-1}^\top G_{j-1}^i\big)\end{aligned}$$

followed by the updates of the activation matrices:

$$(G_j^i)_{kk}\leftarrow\mathbb{I}(W_jh_{j-1}(x_i)>0)\quad\forall k\in[n_j],\ \forall j\in[L-1],\ \forall i\in\mathcal{I}.$$

Let $\{\hat{W}_j\}_{j\in[L]}$ and $\{\hat{G}^i_j\}_{i\in\mathcal{I},j\in[L-1]}$ denote the parameters and the activation matrices at equilibrium. Similar to the proof of Theorem 3.2, Lyapunov stability of this equilibrium implies that

$$\lambda_{\max}\left(\hat{G}^i_j\hat{W}^\top_{j+1}\hat{G}^i_{j+1}\cdots\hat{W}^\top_L\hat{W}_L\cdots\hat{G}^i_{j+1}\hat{W}_{j+1}\hat{G}^i_j\right)\lambda_{\max}\left(\hat{h}_{j-1}(x_i)\hat{h}^\top_{j-1}(x_i)\right)<\frac{2}{\delta}. \tag{3.13}$$

Note that

$$\hat{W}_L\cdots\hat{G}^i_{j+1}\hat{W}_{j+1}\hat{G}^i_j\hat{h}_j(x_i)=\hat{h}_L(x_i),$$

and therefore,

$$\frac{\|\hat{h}_L(x_i)\|_2^2}{\|\hat{h}_j(x_i)\|_2^2}\leq\lambda_{\max}\left(\hat{G}^i_j\hat{W}^\top_{j+1}\hat{G}^i_{j+1}\cdots\hat{W}^\top_L\hat{W}_L\cdots\hat{G}^i_{j+1}\hat{W}_{j+1}\hat{G}^i_j\right)\quad\forall i\in\mathcal{I}'. \tag{3.14}$$

Combining (3.13) and (3.14), we obtain

$$\lambda_{\max}\left(\hat{h}_{j-1}(x_i)\hat{h}^\top_{j-1}(x_i)\right)<\frac{2}{\delta}\frac{\|\hat{h}_j(x_i)\|_2^2}{\|\hat{h}_L(x_i)\|_2^2}\quad\forall i\in\mathcal{I}'.$$

On the other hand,

$$\lambda_{\max}\left(\hat{h}_{j-1}(x_i)\hat{h}^\top_{j-1}(x_i)\right)\geq e_k^\top\hat{G}^i_{j-1}\hat{W}_{j-1}\hat{h}_{j-2}(x_i)\hat{h}^\top_{j-2}(x_i)\hat{W}^\top_{j-1}\hat{G}^i_{j-1}e_k\quad\forall k\in[n_{j-1}].$$

Let $\mathcal{I}^k_j$ denote the set of points that activates the k-th node of the j-th layer at equilibrium:

$$\mathcal{I}^k_j=\{i\in\mathcal{I}':e_k^\top\hat{h}_j(x_i)>0\}\quad\forall k\in[n_j],\ \forall j\in[L-1].$$

Then we can write

$$\lambda_{\max}\left(\hat{h}_{j-1}(x_i)\hat{h}^\top_{j-1}(x_i)\right)\geq e_k^\top\hat{W}_{j-1}\hat{h}_{j-2}(x_i)\hat{h}^\top_{j-2}(x_i)\hat{W}^\top_{j-1}e_k\quad\forall i\in\mathcal{I}^k_{j-1},$$

which implies

$$e_k^\top\hat{W}_{j-1}\hat{h}_{j-2}(x_i)\hat{h}^\top_{j-2}(x_i)\hat{W}^\top_{j-1}e_k\leq\frac{2}{\delta}\frac{\|\hat{h}_j(x_i)\|_2^2}{\|\hat{h}_L(x_i)\|_2^2}\quad\forall i\in\mathcal{I}^k_{j-1}.$$

By summing over all points activating the k-th node of the $(j-1)$-th layer:

$$e_k^\top\hat{W}_{j-1}\left(\sum\nolimits_{i\in\mathcal{I}^k_{j-1}}\hat{h}_{j-2}(x_i)\hat{h}^\top_{j-2}(x_i)\right)\hat{W}^\top_{j-1}e_k\leq\sum\nolimits_{i\in\mathcal{I}^k_{j-1}}\frac{2}{\delta}\frac{\|\hat{h}_j(x_i)\|_2^2}{\|\hat{h}_L(x_i)\|_2^2}.$$

This gives a bound on the k-th row of $\hat{W}_{j-1}$:

$$\|e_k^\top\hat{W}_{j-1}\|_2^2\leq\frac{1}{\lambda_{\min}\left(\sum_{i\in\mathcal{I}^k_{j-1}}\hat{h}_{j-2}(x_i)\hat{h}^\top_{j-2}(x_i)\right)}\sum\nolimits_{i\in\mathcal{I}^k_{j-1}}\frac{2}{\delta}\frac{\|\hat{h}_j(x_i)\|_2^2}{\|\hat{h}_L(x_i)\|_2^2}.$$

As a result, if

$$\sum\nolimits_{i\in\mathcal{I}'} \mathbb{I}\{e_k^\top \hat{W}_{j-1}\hat{h}_{j-2}(x_i) > 0\} \cdot \hat{h}_{j-2}(x_i)\hat{h}_{j-2}^\top(x_i) \succ 0 \quad \forall k \in [n_{j-1}],$$

all rows of $\hat{W}_{j-1}$ are bounded, for $j = 2, 3, \ldots, L$.

To bound the norm of $\hat{W}_L$, consider the gradient update for W_{L-1}. Similar to (3.13), we have

$$\lambda_{\max}\left(\hat{G}_{L-1}^i \hat{W}_L^\top \hat{W}_L \hat{G}_{L-1}^i\right) \lambda_{\max}\left(\hat{h}_{L-2}(x_i)\hat{h}_{L-2}^\top(x_i)\right) < \frac{2}{\delta} \quad \forall i \in \mathcal{I}'.$$

For every point activating the k-th node in the $(L-1)$-th layer, we have

$$\hat{G}_{L-1}^i e_k = e_k,$$

which yields

$$e_k^\top \hat{W}_L^\top \hat{W}_L e_k \le \lambda_{\max}\left(\hat{G}_{L-1}^i \hat{W}_L^\top \hat{W}_L \hat{G}_{L-1}^i\right) \quad \forall i \in \mathcal{I}_{L-1}^k.$$

Therefore, we can write

$$\|\hat{W}_L e_k\|_2^2 \le \frac{2}{\delta \|\hat{h}_{L-2}(x_i)\|_2^2} \quad \forall i \in \mathcal{I}_{L-1}^k,$$

which implies

$$\|\hat{W}_L e_k\|_2^2 \le \frac{2}{\delta \max_{i\in\mathcal{I}_{L-1}^k} \|\hat{h}_{L-2}(x_i)\|_2^2}.$$

This proves that k-th column of $\hat{W}_L$ is bounded in norm. ■

3.8.5 Proof of Theorem 3.4

By duality of the norms $\|\cdot\|_p$ and $\|\cdot\|_q$, we have

$$\min_{d:\|d\|_q\le\epsilon} w^\top(x_i + d) = w^\top x_i - \epsilon\|w\|_p,$$

$$\max_{d:\|d\|_q\le\epsilon} w^\top(x_i + d) = w^\top x_i + \epsilon\|w\|_p.$$

Then problem (3.4b) can be written as

$$\min_w \sum_{i\in\mathcal{I}} \frac{1}{2}\left(y_i - w^\top x_i + \epsilon\|w\|_p\right)^2 + \frac{1}{2}\left(y_i - w^\top x_i - \epsilon\|w\|_p\right)^2,$$

which can be simplified to

$$\min_w \sum_{i\in\mathcal{I}} \left(y_i - w^\top x_i\right)^2 + \epsilon^2\|w\|_p^2.$$

This is a convex problem in w, and we can introduce a slack variable to bring the second term into a constraint form:

$$\begin{aligned} \underset{w,t}{\text{minimize}} \quad & \sum\nolimits_{i\in\mathcal{I}} \left(y_i - w^\top x_i\right)^2 + \epsilon^2 t \\ \text{subject to} \quad & \|w\|_p^2 \le t. \end{aligned} \tag{3.15}$$

Fix $\epsilon > 0$, and assume that (w_0, t_0) is the solution of (3.15). Then w_0 is also a solution to the problem

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \sum\nolimits_{i\in\mathcal{I}} (y_i - w^\top x_i)^2 \\ \text{subject to} \quad & \|w\|_p^2 \le t_0, \end{aligned}$$

as well as

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \sum\nolimits_{i\in\mathcal{I}} (y_i - w^\top x_i)^2 \\ \text{subject to} \quad & \|w\|_p^m \le t_0^{m/2}. \end{aligned} \tag{3.16}$$

If $t_0 = 0$, then w_0 is also zero, and this solution can be obtained by (3.4a) by choosing λ large enough. Therefore, without loss of generality assume $t_0 \neq 0$. Then problem (3.16) satisfies the Slater's condition, and strong duality holds (Boyd and Vandenberghe, 2004). Then we can find its solution by solving

$$\min_w \sum\nolimits_{i\in\mathcal{I}} (y_i - w^\top x_i)^2 + \lambda^*(\|w\|_p^m - t_0^{m/2})$$

where λ^* is the dual solution. Note that this problem is strictly convex in w, and therefore, its solution is unique, for which the only candidate is w_0. We conclude that

$$w_0 = \underset{w}{\operatorname{argmin}} \sum\nolimits_{i\in\mathcal{I}} (y_i - w^\top x_i)^2 + \lambda^* \|w\|_p^m.$$

This completes the one direction of the proof, and the other direction is identical. ■

Chapter 4

Robustness of Models Trained with the Cross-Entropy Loss

4.1 Introduction

Training neural networks is challenging and involves making several design choices. Among these are the architecture of the network, the training loss function, the optimization algorithm used for training, and their hyperparameters, such as the learning rate and the batch size. Most of these design choices influence the solution obtained by the training procedure and have been studied in detail (Kingma and Ba, 2014; Hardt et al., 2015; He et al., 2016; Wilson et al., 2017; Nar and Sastry, 2018a; Smith et al., 2018). Nevertheless, one choice has been mostly taken for granted when the network is trained for a classification task: the training loss function.

Cross-entropy loss function is almost the sole choice for classification tasks in practice. Its prevalent use is backed theoretically by its association with the minimization of the Kullback-Leibler divergence between the empirical distribution of a dataset and the *confidence* of the classifier for that dataset. Given the particular success of neural networks for classification tasks (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; He et al., 2016), there seems to be little motivation to search for alternatives for this loss function, and most of the software developed for neural networks incorporates an efficient implementation for it, thereby facilitating its further use.

Recently there has been a line of work analyzing the dynamics of training a linear classifier with the cross-entropy loss function (Soudry et al., 2018; Nacson et al., 2018a,b; Ji and Telgarsky, 2018). They specified the decision boundary that the gradient descent algorithm yields on linearly separable datasets and claimed that this solution achieves the maximum ℓ_2 margin. However, these claims were observed not to hold in the simple experiments we ran. For example, Figure 4.1 displays a case where the cross-entropy minimization for a linear classifier leads to a decision boundary which attains an extremely poor margin and is nearly orthogonal to the solution given by the hard-margin support vector machine (SVM).

We set out to understand this discrepancy between the claims of the previous works and our observations on the simple experiments. We can provide an outline for this chapter as follows.

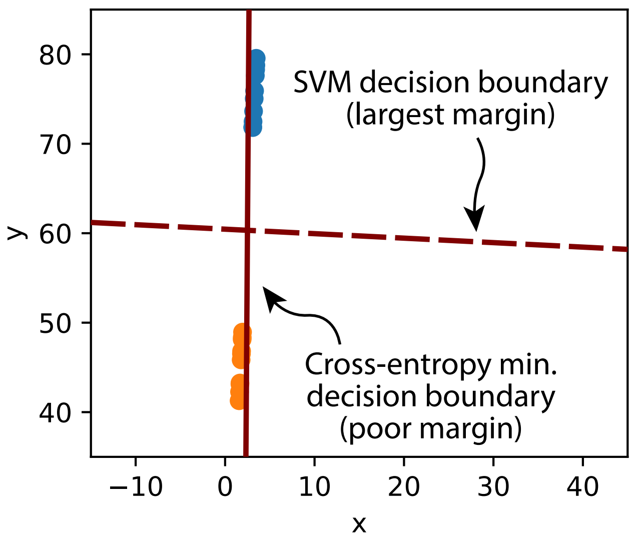


Figure 4.1: Orange and blue points represent the data from two different classes in $\mathbb{R}^2$. Cross-entropy minimization for a linear classifier on the given training points leads to the decision boundary shown with the solid line, which attains a very poor margin and is almost orthogonal to the solution given by the SVM.

1. We analyze the minimization of the cross-entropy loss for a linear classifier by using only two training points, i.e., only one point from each of the two classes, and we show that the dynamics of the gradient descent algorithm could yield a poor decision boundary, which could be almost orthogonal to the boundary with the maximum margin.

2. We identify the source of discrepancy between our observations and the claims of the recent works as the misleading abbreviation of notation in the previous works. We clarify why the solution obtained with cross-entropy minimization is different from the SVM solution.

3. We show that for linearly separable datasets, if the features of the training points lie in an affine subspace, and if the cross-entropy loss is minimized by a gradient method with no regularization to train a linear classifier, the margin between the decision boundary of the classifier and the training points could be much smaller than the optimal value. We verify that when a neural network is trained with the cross-entropy loss to classify two classes from the CIFAR-10 dataset, the output of the penultimate layer of the network indeed produces points that lie on an affine subspace.

4. We show that if there is no explicit and effective regularization, the weights of the last layer of a neural network could grow to infinity during training with a gradient method. Even though this has been observed in recent works as well, we are the first to point out that this divergence drives the confidence of the neural network to 100% at almost every point in the input space if the network is trained for long. In other words, the confidence depends heavily on the training duration, and its exact value might be of little significance as long as it is above 50%.

5. We introduce differential training, which is a training paradigm that uses a loss function defined on pairs of points from each class – instead of only one point from any class. We show that the decision boundary of a linear classifier trained with differential training indeed produces the SVM solution with the maximum hard margin.

The results in this chapter have appeared in (Nar et al., 2019a,b; Nar and Sastry, 2019).

4.2 Classification of Two Points with the Cross-Entropy Loss

We start with a simple binary classification problem. Given two points $x \in \mathbb{R}^d$ and $-y \in \mathbb{R}^d$ from two different classes, we can find a linear classifier by minimizing the cross-entropy loss function

$$\min_{w\in\mathbb{R}^d, b\in\mathbb{R}} \left\{ -\log\left(\frac{1}{e^{-w^\top x - b}+1}\right) - \log\left(\frac{e^{w^\top y - b}}{e^{w^\top y-b}+1}\right)\right\},$$

or equivalently, by solving

$$\min_{\tilde{w}\in\mathbb{R}^{d+1}} \left\{\log(e^{-\tilde{w}^\top\tilde{x}}+1) + \log(e^{-\tilde{w}^\top\tilde{y}}+1)\right\}, \tag{4.1}$$

where $\tilde{x} = [x^\top\ 1]^\top$, $-\tilde{y} = [-y^\top\ 1]^\top$ and $\tilde{w} = [w^\top\ b]^\top$. Unless the two points x and $-y$ are equal, the function (4.1) does not attain its minimum at a finite value of $\tilde{w}$. Consequently, if the gradient descent algorithm is used to minimize (4.1), the iterate at time k, $\tilde{w}[k]$, diverges as k increases. The following theorem characterizes the growth rate of $\tilde{w}[k]$ and its direction in the limit by using a continuous-time approximation to the gradient descent algorithm.

Theorem 4.1. *Given two points $x \in \mathbb{R}^d$ and $-y \in \mathbb{R}^d$, let $\tilde{x}$ and $-\tilde{y}$ denote $[x^\top\ 1]^\top$ and $[-y^\top\ 1]$, respectively. Without loss of generality, assume $\|x\| \leq \|y\|$. If the two points are in different classes and we minimize the cross-entropy loss*

$$\min_{\tilde{w}\in\mathbb{R}^{d+1}} \log(1+e^{-\tilde{w}^\top\tilde{x}}) + \log(1+e^{-\tilde{w}^\top\tilde{y}})$$

by using the continuous-time approximation to the gradient descent algorithm

$$\frac{d\tilde{w}}{dt} = \tilde{x}\frac{\delta e^{-\tilde{w}^\top\tilde{x}}}{1+e^{-\tilde{w}^\top\tilde{x}}} + \tilde{y}\frac{\delta e^{-\tilde{w}^\top\tilde{y}}}{1+e^{-\tilde{w}^\top\tilde{y}}}$$

with the initialization $\tilde{w}(0) = 0$ and the learning rate δ, then

$$\lim_{t\to\infty}\frac{\tilde{w}(t)}{\log(t)} = \begin{cases} \frac{\sigma_y-\sigma_{xy}}{\sigma_x\sigma_y-\sigma_{xy}^2}\tilde{x} + \frac{\sigma_x-\sigma_{xy}}{\sigma_x\sigma_y-\sigma_{xy}^2}\tilde{y} & \text{if } \sigma_{xy} < \sigma_x, \\ \frac{1}{\sigma_x}\tilde{x} & \text{if } \sigma_{xy} \geq \sigma_x, \end{cases} \tag{4.2}$$

where $\sigma_x = \|\tilde{x}\|^2$, $\sigma_{xy} = \tilde{x}^\top\tilde{y}$ and $\sigma_y = \|\tilde{y}\|^2$.

Note that first d coordinates of (4.2) represent the normal vector of the decision boundary obtained by minimizing the cross-entropy loss (4.1). This vector is different from $x+y$, which is the direction of the maximum-margin solution given by the SVM. In fact, the direction in (4.2) could be almost orthogonal to the SVM solution in certain cases, which implies that the margin between the points and the decision boundary could be much smaller than the optimal value. Corollary 4.1 describes a subset of these cases.

Corollary 4.1. *Given two points x and $-y$ in $\mathbb{R}^d$, let ψ denote the angle between the solution given by (4.2) and the solution given by the SVM, i.e., $(x+y)$. If $x^\top y = 1$, then*

$$\cos^2 \psi \leq \frac{4}{2 + \frac{\sigma_y}{\sigma_x}\left(1 - \frac{1}{\sigma_x}\right)},$$

where $\sigma_x = \|x\|^2+1$ and $\sigma_y = \|y\|^2+1$. Consequently, as $\|x\|/\|y\|$ approaches 0 while maintaining the condition $x^\top y = 1$, the angle ψ converges to $\pi/2$.

Remark 4.1. Corollary 4.1 shows that if x and $-y$ have disparate norms, the minimization of the cross-entropy loss with gradient descent algorithm could lead to a direction which is almost orthogonal to the maximum-margin solution. It may seem like this problem could be avoided with preprocessing the data so as to normalize the data points. However, this approach will not be effective for neural networks: if we consider an L-layer neural network, $w^\top \phi_{L-1}(x)$, and regard the first $L-1$ layers, $\phi_{L-1}(\cdot)$, as a feature mapping, preprocessing a dataset $\{x_i\}_{i\in I}$ will not produce a normalized set of features $\{\phi_{L-1}(x_i)\}_{i\in I}$. Note that we could not normalize $\{\phi_{L-1}(x_i)\}_{i\in I}$ directly either, since the mapping $\phi_{L-1}(\cdot)$ evolves during training.

Remark 4.2. Theorem 4.1 shows that the norm of w keeps growing unboundedly as the training continues. The same behavior will be observed for larger datasets in the next sections as well. Since the "confidence" of the classifier for its prediction at a point x is given by

$$\max\left(\frac{1}{e^{-w^\top x-b}+1}, \frac{e^{-w^\top x-b}}{e^{-w^\top x-b}+1}\right),$$

this unbounded growth of $\|w\|$ drives the confidence of the classifier to 100% at every point in the input space, except at the points on the decision boundary, if the algorithm is run for long. Given the lack of effective regularization for neural networks, a similar unbounded growth is expected to be observed in neural network training as well, which is mentioned in (Bartlett et al., 2017). As a result, the confidence of a neural network might be highly correlated with the training duration, and whether a neural network gives 99% or 51% confidence for a prediction might be of little importance as long as it is above 50%. In other words, regarding this confidence value as a measure of similarity between an input and the training dataset from the most-likely class should be reconsidered.

4.3 Margins of Linear Classifiers Trained with the Cross-Entropy Loss

In this section, we examine the binary classification of a linearly separable dataset by minimizing the cross-entropy loss function. Recently, this problem has also been studied in (Soudry et al., 2018; Nacson et al., 2018b,a; Ji and Telgarsky, 2018). We restate an edited version of the main theorem of (Soudry et al., 2018), followed by the reason of the edition.

Theorem 4.2. *[Adapted from Theorem 3 of (Soudry et al., 2018)] Given two sets of points* $\{x_i : i \in \mathcal{I}\}$ *and* $\{x_j : j \in \mathcal{J}\}$ *that are linearly separable in* $\mathbb{R}^n$*, let* $\tilde{x}_i$ *and* $\tilde{x}_j$ *denote* $[x_i^\top\ 1]^\top$ *and* $[x_j^\top\ 1]^\top$*, respectively, for all* $i \in \mathcal{I}$*,* $j \in \mathcal{J}$*. Then the iterate of the gradient descent algorithm,* $\tilde{w}(t)$*, on the cross-entropy loss function*

$$\min_{\tilde{w}\in\mathbb{R}^{n+1}} \sum\nolimits_{i\in\mathcal{I}} \log(1+e^{-\tilde{w}^\top \tilde{x}_i}) + \sum\nolimits_{j\in\mathcal{J}} \log(1+e^{\tilde{w}^\top \tilde{x}_j})$$

with a sufficiently small step size will converge in direction:

$$\lim_{t\to\infty} \frac{\tilde{w}(t)}{\|\tilde{w}(t)\|} = \frac{\overline{w}}{\|\overline{w}\|},$$

where $\overline{w}$ *is the solution to*

$$\begin{aligned} \underset{z\in\mathbb{R}^{n+1}}{\text{minimize}} \quad & \|z\|^2 \\ \text{subject to} \quad & \langle z, \tilde{x}_i\rangle \geq 1 \quad \forall i \in \mathcal{I}, \\ & \langle z, \tilde{x}_j\rangle \leq -1 \quad \forall j \in \mathcal{J}. \end{aligned} \tag{4.3}$$

The solution (4.3) given in Theorem 4.2 was referred in (Soudry et al., 2018), and consequently in the other works, as the maximum-margin solution. However, due to the absence of the bias term in the notation, this claim is not completely accurate. Given the linearly separable sets of points $\{x_i\}_{i\in I}$ and $\{-y_j\}_{j\in J}$, the maximum-margin solution given by the SVM solves

$$\begin{aligned} \underset{w,b}{\text{minimize}} \quad & \|w\|_2^2 \\ \text{subject to} \quad & \langle w, x_i\rangle + b \geq 1 \qquad \forall i \in I, \\ & \langle w, -y_j\rangle + b \leq -1 \quad \forall j \in J. \end{aligned} \tag{P1}$$

On the other hand, the solution given by Theorem 4.2 corresponds to

$$\begin{aligned} \underset{w,b}{\text{minimize}} \quad & \|w\|_2^2 + b^2 \\ \text{subject to} \quad & \langle w, x_i\rangle + b = \langle \tilde{w}, \tilde{x}_i\rangle \geq 1 \qquad \forall i \in I, \\ & \langle w, -y_j\rangle + b = \langle \tilde{w}, -\tilde{y}_j\rangle \leq -1 \quad \forall j \in J, \end{aligned} \tag{P2}$$

where we define $\tilde{w} = [w^\top \; b]^\top$, $\tilde{x}_i = [x_i^\top \; 1]^\top$ and $\tilde{y}_j = [y_j^\top \; -1]^\top$ for all $i \in I, j \in J$. Even though the sets of constraints for both problems are identical, their objective functions are different, and consequently, the solutions are different. As a result, the decision boundary obtained by cross-entropy minimization does not necessarily attain the maximum hard margin. In fact, as the following theorem shows, its margin could be arbitrarily worse than the maximum margin.

Theorem 4.3. *Assume that the points $\{x_i\}_{i\in\mathcal{I}}$ and $\{x_j\}_{j\in\mathcal{J}}$ are linearly separable, and a linear classifier is trained by minimizing the cross entropy loss:*

$$\min_{w,b} \sum\nolimits_{i\in\mathcal{I}} \log\left(1 + e^{-w^\top x_i - b}\right) + \sum\nolimits_{j\in\mathcal{J}} \log\left(1 + e^{w^\top x_j + b}\right),$$

via the gradient descent algorithm. Let $\langle \overline{w}, \cdot\rangle + B = 0$ denote the decision boundary obtained, and assume that $\overline{w}$ and B are scaled such that

$$\min_{i\in\mathcal{I}} \langle \overline{w}, x_i\rangle - \max_{j\in\mathcal{J}} \langle \overline{w}, x_j\rangle = 2.$$

Define the set of indices for the support vectors as

$$\begin{aligned} \mathcal{I}_{sup} &= \left\{ i \in \mathcal{I} : \langle \overline{w}, x_i\rangle \le \langle \overline{w}, x_{i'}\rangle \; \forall i' \in \mathcal{I} \right\}, \\ \mathcal{J}_{sup} &= \left\{ j \in \mathcal{J} : \langle \overline{w}, x_j\rangle \ge \langle \overline{w}, x_{j'}\rangle \; \forall j' \in \mathcal{J} \right\}. \end{aligned}$$

If the support vectors lie in an affine subspace, that is, if there exist a set of orthonormal vectors $\{r_k\}_{k\in K}$ and a set of scalars $\{\Delta_k\}_{k\in K}$ such that

$$\langle r_k, x_i\rangle = \langle r_k, x_j\rangle = \Delta_k \quad \forall i \in \mathcal{I}_{sup}, \; \forall j \in \mathcal{J}_{sup}, \; \forall k \in K,$$

then the minimization of the cross-entropy loss yields a margin smaller than or equal to

$$\frac{1}{\sqrt{\frac{1}{\gamma_{OPT}^2} + B^2 \sum_{k\in K} \Delta_k^2}}$$

where γ_{OPT} denotes the optimal hard margin in the input space given by the SVM solution.

Theorem 4.3 points out that the margin of the classifier is determined by only the support vectors, which are the training points that are closest to the decision boundary of the classifier. The points that are further to the decision boundary have no effect on the margin, and there is a reason for this: their *excitation* of the parameters do not *persist* as the gradient descent algorithm continues to update the parameters. Once these points are correctly classified, their contribution to the gradient of the loss function is removed exponentially fast, and consequently, the dynamics of the algorithm is dominated by the other points.

Remark 4.3. Theorem 4.3 shows that if the training points lie in an affine subspace, the margin obtained by the cross-entropy minimization will be smaller than the optimal margin value. As the dimension of this affine subspace decreases, the cardinality of the set K increases and the term $\sum_{k\in K}\Delta_k^2$ could become much larger than $1/\gamma^2$. Therefore, as the dimension of the subspace containing the training points gets smaller compared to the dimension of the input space, cross-entropy minimization with a gradient method becomes more likely to yield a poor margin. Note that this argument also holds for classifiers of the form $w^\top\phi(x)$ with the fixed feature mapping $\phi(\cdot)$.

It is essential to note that the bias term B in Theorem 4.3 is *not* the bias of the training data set; it is the bias of the support vectors. It is likely that the training data points have zero mean, whereas the support vectors have non-negligible bias.

Note also that making the bias term B zero requires the a priori knowledge of the set of support vectors — which is not available until the algorithm has completed and the optimization problem is solved. Therefore, the term B can *not* be made zero simply by preprocessing the data and removing its mean. This is the primary reason why having a poor margin is unavoidable when the cross-entropy loss is used for low-dimensional data sets.

The next theorem relaxes the condition of Theorem 4.3 and allows the training points to be near an affine subspace instead of being exactly on it. Note that the ability to compare the margin obtained by cross-entropy minimization with the optimal value is lost. Nevertheless, it highlights the fact that same set of points could be assigned a different margin by cross-entropy minimization if all of them are shifted away from the origin by the same amount in the same direction.

Theorem 4.4. *Assume that the points $\{x_i\}_{i\in\mathcal{I}}$ and $\{x_j\}_{j\in\mathcal{J}}$ in $\mathbb{R}^n$ are linearly separable and there exist a set of orthonormal vectors $\{r_k\}_{k\in K}$ and a set of scalars $\{\Delta_k\}_{k\in K}$ such that*

$$\langle r_k, x_i\rangle \geq \Delta_k,\ \langle r_k, x_j\rangle \leq \Delta_k \quad \forall i\in\mathcal{I}_{sup},\ \forall j\in\mathcal{J}_{sup},\ \forall k\in K.$$

Let $\langle\overline{w},\cdot\rangle + B = 0$ denote the decision boundary obtained by minimizing the cross-entropy loss, as in Theorem 4.3. Then the minimization of the cross-entropy loss yields a margin smaller than or equal to

$$\frac{1}{\sqrt{B^2\sum_{k\in K}\Delta_k^2}}.$$

Remark 4.4. Both Theorem 4.3 and Theorem 4.4 consider linearly separable datasets. If the dataset is not linearly separable, (Ji and Telgarsky, 2018) predicts that the normal vector of the decision boundary, w, will have two components, one of which converges to a finite vector and the other diverges. The diverging component still has the potential to drive the decision boundary to a direction with a poor margin. In fact, the margin is expected to be small especially if the points intruding into the opposite class lie in the same subspace as the optimal normal vector for the decision boundary. Nevertheless, we have focused on the case of separable datasets as this case provides critical insight into the issues of state-of-the-art neural networks, given they can easily attain zero training error even on randomly generated datasets, which indicates the linear separability of the features obtained at their penultimate layers (Zhang et al., 2017).

4.4 Two-Layer Nonlinear Network Trained with the Cross-Entropy Loss

When a two-layer neural network is trained via the gradient descent algorithm, the dynamics of the algorithm will be nonlinear. If, in addition, the cost function used for training is the cross-entropy loss, the parameters of the network will grow unboundedly — provided that there are enough parameters to classify every training point correctly with a proper initialization (Zhang et al., 2017). However, most analysis tools for nonlinear dynamical systems primarily focus on the behaviors of these systems around their equilibria, which are the fixed points of their dynamics, or their limiting behaviors inside bounded regions (Khalil, 1996; Sastry, 2013). Therefore, the common tools in nonlinear analysis will become inapplicable for the analysis of the cross-entropy loss. In this case, instead of studying the convergence of the actual values of the parameters, we can analyze the convergence of the ratio of the parameters. This leads us to consider an alternative concept of convergence: convergence in direction.

Definition 4.1. (Convergence in direction) Given a set of functions, $W_k : [0, \infty) \mapsto \mathbb{R}^{m_k \times n_k}$ for $k \in [L]$, assume $\lim_{t\to\infty} \|W_k(t)\|_F = \infty$ for some $k \in [L]$. The set $(W_1, \dots, W_L)$ is said to converge in direction to $(\overline{W}_1, \dots, \overline{W}_L)$ if

$$\lim_{t\to\infty} \frac{\|W_k(t) - h(t)\overline{W}_k\|_F}{h(t)} = 0 \quad \forall k \in [L]$$

for some function $h : [0, \infty) \mapsto (0, \infty)$ diverging to $+\infty$, where $\overline{W}_k \in \mathbb{R}^{m_k \times n_k}$ is a matrix with bounded elements for each $k \in [L]$, and $\overline{W}_k \neq \mathbf{0}$ at least for some $k \in [L]$.

With this concept of convergence, we are ready to state the next theorem.

Theorem 4.5. *Assume that a two-layer neural network is trained to classify the sets $\{x_i\}_{i\in\mathcal{I}}$ and $\{x_j\}_{j\in\mathcal{J}}$ by minimizing the cross-entropy loss*

$$\sum\nolimits_{i\in\mathcal{I}} \log\left(1 + e^{-w^\top (Vx_i+b)_+}\right) + \sum\nolimits_{j\in\mathcal{J}} \log\left(1 + e^{w^\top (Vx_j+b)_+}\right)$$

via the continuous-time gradient descent algorithm, where $w \in \mathbb{R}^r$, $V \in \mathbb{R}^{r\times n}$ and $b \in \mathbb{R}^r$. Assume all the training points are correctly classified[1]*, and the parameters (w, V, b) converge in direction to $(\overline{w}, \overline{V}, \overline{b})$ as defined above. Let $(\overline{w}, \overline{V}, \overline{b})$ be scaled such that*

$$\overline{w}^\top (\overline{V}x_i + \overline{b})_+ \geq 1 \quad \forall i \in \mathcal{I}, \tag{4.4a}$$

$$\overline{w}^\top (\overline{V}x_j + \overline{b})_+ \leq -1 \quad \forall j \in \mathcal{J} \tag{4.4b}$$

[1]State-of-the-art neural networks are able to achieve zero training error even on randomly generated and randomly labeled data sets (Zhang et al., 2017).

with either equality holding for at least one point. Let $\mathcal{I}_{sup}$ and $\mathcal{J}_{sup}$ denote the points that achieve equality in (4.4a) and (4.4b), respectively. Then the Lipschitz constant of the mapping $x \mapsto \overline{w}^\top(\overline{V}x + \overline{b})_+$ is upper bounded by

$$\frac{n_{\text{node}}^{\min}\sqrt{n_{\text{sup}}^{\max}}}{\sqrt{\underline{\lambda}}},$$

where $n_{\text{sup}}^{\max}$ is the maximum number of support vectors that activate the same hidden layer node:

$$n_{\text{sup}}^{\max} = \max_{k\in[r]} \max_{S\subseteq\mathcal{I}_{sup}\cup\mathcal{J}_{sup}} \left\{|S| : \overline{V}_k x_s + \overline{b}_k > 0 \; \forall s \in S\right\},$$

$n_{\text{node}}^{\min}$ is the minimum number of nodes that are activated by all of the support vectors:

$$n_{\text{node}}^{\min} = \min_{K\subseteq[r]} \left\{|K| : \max\nolimits_{k\in K} \overline{V}_k x_s + \overline{b}_k > 0 \; \forall s \in \mathcal{I}_{sup} \cup \mathcal{J}_{sup}\right\},$$

and $\underline{\lambda}$ is a lower bound for the minimum eigenvalue of $X_k^\top X_k$, where the <u>columns</u> of X_k are the support vectors activating node k.

Theorem 4.5 reveals a new notion of richness for the data set that is markedly different than the full-rankness of the data or of the support vectors; it is the linear independence of the support vectors. If the support vectors activating any one of the hidden-layer nodes are linearly dependent, then $\underline{\lambda}$ in Theorem 4.5 will be zero, and the bound will become void. Note that this notion of richness also reinforces the result of Theorem 4.3.

4.5 Low-Dimensionality of Hidden-Layer Activations

As seen in Section 3.3 and Section 4.4, robustness guarantees for neural networks seem to require certain amount of richness in the training data set and the hidden-layer activations. While the full rankness of the training data activating each hidden-layer node provides persistency of excitation for a network trained with the squared-error loss, the use of cross-entropy loss necessitates the linear independence of all support vectors activating the same hidden-layer nodes. These two conditions are different, but both of them are likely to be violated if the training data is large in number and the features of the data are low-dimensional in some layer of the network. Even though this appears to be a degenerate case, there are two main reasons why it is also the prevalent case:

1. The raw image, audio, and video data are low-dimensional by nature; this is the fundamental fact that has enabled data compression for decades by discovering and utilizing low-dimensional representations specific to different applications.

2. Even if the input data is full rank, the use of the gradient descent algorithm for training multiple-layer networks induces low-rank signals in the intermediate layers of the network.

The latter item has been observed in several empirical works, for example, in (Martin and Mahoney, 2019). The following theorem confirms its validity for deep linear networks.

Theorem 4.6. *Assume that a deep linear network with L layers is trained to classify the points in the sets $\{x_i\}_{i\in\mathcal{I}}$ and $\{x_j\}_{j\in\mathcal{J}}$ by minimizing*

$$\ell(W_1,\ldots,W_L) = \sum\nolimits_{s\in\mathcal{I}\cup\mathcal{J}} d(W_L\cdots W_1 x_s, y_s) + \sum\nolimits_{k\in[L]} \mu_k \|W_k\|_F^2 \tag{4.5}$$

where $\{y_s : s\in\mathcal{I}\cup\mathcal{J}\}$ is the set of labels, $d(\cdot,\cdot)$ is any loss function differentiable in its first argument, $W_k\in\mathbb{R}^{n_k\times n_{k-1}}$ for all $k\in[L]$ and $n_L=1$, i.e., the output of the network is scalar, and $\mu_k>0$ for all $k\in[L]$. If the gradient descent algorithm converges from a random initialization to a solution $(\overline{W}_1,\ldots,\overline{W}_L)$, then each weight matrix $\overline{W}_k$ has rank 1, almost surely.

Theorem 4.6 shows that adding the Frobenius norms of the weight parameters, which is commonly referred to as adding weight-decay, causes all signals in the hidden layers to have rank 1 — independently of the loss function used for training. The following theorem shows that when the cross-entropy loss is used, the result is the same even without the addition of weight-decay.

Theorem 4.7. *Assume that the sets $\{x_i\}_{i\in\mathcal{I}}$ and $\{x_j\}_{j\in\mathcal{J}}$ are separable by a hyperplane passing through the origin, and a deep linear network with L layers is trained to classify them by minimizing the cross-entropy loss:*

$$\ell(W_1,\ldots,W_L) = \sum\nolimits_{i\in\mathcal{I}} \log\left(1+e^{W_L\cdots W_1 x_i}\right) + \sum\nolimits_{j\in\mathcal{J}} \log\left(1+e^{-W_L\cdots W_1 x_j}\right) \tag{4.6}$$

via the continuous-time gradient descent algorithm:

$$\frac{dW_k(t)}{dt} = -\frac{\partial\ell(W_1(t),\ldots,W_L(t))}{\partial W_k(t)} \quad \forall k\in[L].$$

If the weight matrices converge in direction to $(\overline{W}_1,\ldots,\overline{W}_L)$ from a random initialization, then each $\overline{W}_k$ has rank 1 almost surely.

Theorem 4.6 and Theorem 4.7 show that the conditions given in the previous sections for the persistency of excitation are easily violated when the model has multiple layers and the gradient descent algorithm is used for training.

4.6 Differential Training for Linear Classifiers

In previous sections, we saw that the cross-entropy minimization could lead to poor margins, and the main reason for this was the appearance of the bias term in the objective function of (P2). In order to remove the effect of the bias term, consider the SVM problem (P1) and note that this problem could be equivalently written as

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \|w\|_2^2 \\ \text{subject to} \quad & \langle w, x_i + y_j\rangle \geq 2 \quad \forall i\in I,\ \forall j\in J \end{aligned} \tag{P3}$$

if we only care about the weight parameter w. This gives the hint that if we use the set of differences $\{x_i + y_j : i \in I, j \in J\}$ instead of the individual sets $\{x_i\}_{i\in I}$ and $\{-y_j\}_{j\in J}$, the bias term could be excluded from the problem. This was also noted in (Keerthi et al., 2000; Ishibashi et al., 2008) previously. Indeed, this approach allows obtaining the SVM solution with a loss function similar to the cross-entropy loss, as the following theorem shows.

Theorem 4.8. *Given two sets of points $\{x_i\}_{i\in I}$ and $\{-y_j\}_{j\in J}$ that are linearly separable in $\mathbb{R}^d$, if we solve*

$$\min_{w\in\mathbb{R}^d} \sum\nolimits_{i\in I} \sum\nolimits_{j\in J} \log(1 + e^{-w^\top (x_i+y_j)}) \tag{4.7}$$

by using the gradient descent algorithm with a sufficiently small learning rate, the direction of w converges to the direction of maximum-margin solution, i.e.

$$\lim_{t\to\infty} \frac{w(t)}{\|w(t)\|} = \frac{w_{SVM}}{\|w_{SVM}\|}, \tag{4.8}$$

where w_{SVM} is the solution of (P3).

Proof. Apply Theorem 4.2 by replacing the sets $\{x_i\}_{i\in I}$ and $\{-y_j\}_{j\in J}$ with $\{x_i + y_j\}_{i\in I, j\in J}$ and the empty set, respectively. Then the minimization of the loss function (4.7) with the gradient descent algorithm leads to

$$\lim_{t\to\infty} \frac{w}{\|w\|} = \frac{\overline{w}}{\|\overline{w}\|}$$

where $\overline{w}$ satisfies

$$\overline{w} = \arg\min_w \|w\|^2 \text{ such that } \langle w, x_i + y_j\rangle \geq 1 \quad \forall i \in I,\ \forall j \in J.$$

Since w_{SVM} is the solution of (P3), we obtain $\overline{w} = \frac{1}{2} w_{\text{SVM}}$, and the claim of the theorem holds. ■

Remark 4.5. Theorem 4.8 is stated for the gradient descent algorithm, but the identical statement could be made for the stochastic gradient method as well by invoking the main theorem of (Nacson et al., 2018).

Minimization of the cost function (4.7) yields the weight parameter $\hat{w}$ of the decision boundary. The bias parameter, b, could be chosen by plotting the histogram of the inner products $\{\langle \hat{w}, x_i\rangle\}_{i\in I}$ and $\{\langle \hat{w}, -y_j\rangle\}_{j\in J}$ and fixing a value for $\hat{b}$ such that

$$\langle \hat{w}, x_i\rangle + \hat{b} \geq 0 \quad \forall i \in I, \tag{4.9a}$$

$$\langle \hat{w}, -y_j\rangle + \hat{b} \leq 0 \quad \forall j \in J. \tag{4.9b}$$

The largest hard margin is achieved by

$$\hat{b} = -\frac{1}{2}\min_{i\in I}\langle \hat{w}, x_i\rangle - \frac{1}{2}\max_{j\in J}\langle \hat{w}, -y_j\rangle. \tag{4.10}$$

However, by choosing a larger or smaller value for $\hat{b}$, it is possible to make a tradeoff between the Type-I and Type-II errors.

The cost function (4.7) includes a loss defined on every pair of data points from the two classes. This cost function can be considered as the cross-entropy loss on a new dataset which contains $|I| \times |J|$ points. There are two aspects of this fact:

1. When standard loss functions are used for classification tasks, we need to oversample or undersample either of the classes if the training dataset contains different number of points from different classes. This problem does not arise when we use the cost function (4.7).

2. Number of pairs in the new dataset, $|I| \times |J|$, will usually be much larger than the original dataset, which contains $|I| + |J|$ points. Therefore, the minimization of (4.7) might appear more expensive than the minimization of the standard cross-entropy loss computationally. However, if the points in different classes are well separated and the stochastic gradient method is used to minimize (4.7), the algorithm achieves zero training error after using only a few pairs, which is formalized in Theorem 4.9. Further computation is needed only to improve the margin of the classifier. In addition, in our experiments to train a neural network to classify two classes from the CIFAR-10 dataset, only a few percent of $|I| \times |J|$ points were observed to be sufficient to reach a high accuracy on the training dataset.

Theorem 4.9. *Given two sets of points $\{x_i\}_{i \in I}$ and $\{-y_j\}_{j \in J}$ that are linearly separable in $\mathbb{R}^d$, assume the cost function (4.7) is minimized with the stochastic gradient method. Define*

$$R_x = \max\{\|x_i - x_{i'}\| : i, i' \in I\}, \quad R_y = \max\{\|y_j - y_{j'}\| : j, j' \in J\}$$

and let γ denote the hard margin that would be obtained with the SVM:

$$2\gamma = \max_{u \in \mathbb{R}^d} \min_{i \in I, j \in J} \langle x_i + y_j, u/\|u\| \rangle.$$

If $2\gamma \geq 5\max(R_x, R_y)$, then the stochastic gradient algorithm produces a weight parameter, $\hat{w}$, only in one iteration which satisfies the inequalities (4.9a)-(4.9b) along with the bias, $\hat{b}$, given by (4.10).

4.7 Experiments

In this section, we present numerical experiments supporting our claims.

Differential training. In Figure 4.2, we show the decision boundaries of two linear classifiers, where one of them is trained by minimizing the *cross-entropy loss*, and the other through *differential training*. Unlike the example shown in Figure 4.1, here the data do not exactly lie in an affine subspace. In particular, one of the classes is composed of 10 samples from a normal distribution with mean $(2, 12)$ and variance 25, and the other class is composed of 10 samples from a normal distribution with mean $(40, 50)$ and variance 25. As can be seen from the figure, the cross-entropy minimization yields a margin that is smaller than differential training, even though when the training dataset is not low-dimensional, which is predicted by Theorem 4.4.

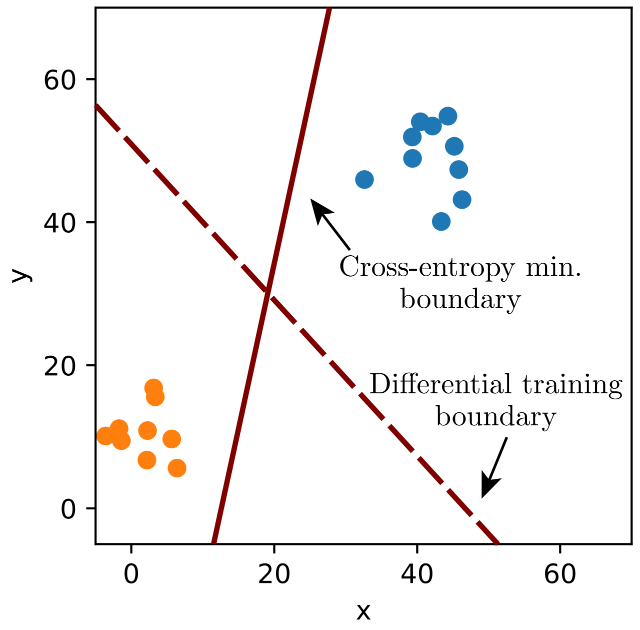


Figure 4.2: Classification boundaries obtained via differential training and cross-entropy minimization. The margin recovered by cross-entropy minimization is worse than that is obtained by differential training even when the training dataset is not low-dimensional.

Low-dimensionality. We empirically evaluated if the features obtained at the penultimate layer of a neural network indeed lie in a low-dimensional affine subspace. For this purpose, we trained a convolutional neural network architecture to classify horses and planes from the CIFAR-10 dataset (Krizhevsky and Hinton, 2009). Figure 4.3 shows the cumulative variance explained for the features that feed into the soft-max layer as a function of the number of principal components used. Similarly, Figure 4.4 displays the principal component analysis for all layers of the network trained with the cross-entropy loss. We observe that the features, which are the outputs of the penultimate layer of the network, lie in a low-dimensional affine subspace, and this holds for a variety of training modalities for the network. This observation is relevant to Remark 4.3. The dimension of the subspace containing the training points is at most 20, which is much smaller than the dimension of the feature space, 84. Consequently, cross-entropy minimization with a gradient method is expected to yield a poor margin on these features.

4.8 Discussion

We compare our results with related works and discuss their implications for the following subjects.

Adversarial examples. State-of-the-art neural networks have been observed to misclassify inputs that are slightly different from their training data, which indicates a small margin between their decision boundaries and the training dataset (Szegedy et al., 2013; Goodfellow et al., 2015; Moosavi-Dezfooli et al., 2017; Fawzi et al., 2017). Our results reveal that the combination of

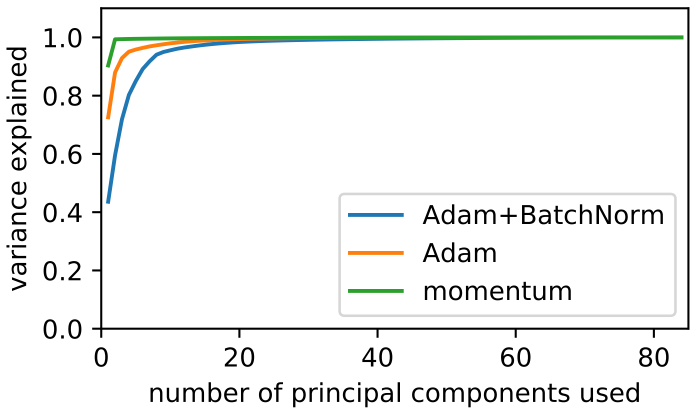


Figure 4.3: The activations feeding into the soft-max layer could be considered as the features for a linear classifier. Plot shows the cumulative variance explained for these features as a function of the number of principal components used. Almost all the variance in the features is captured by the first 20 principal components out of 84, which shows that the input to the soft-max layer resides predominantly in a low-dimensional subspace.

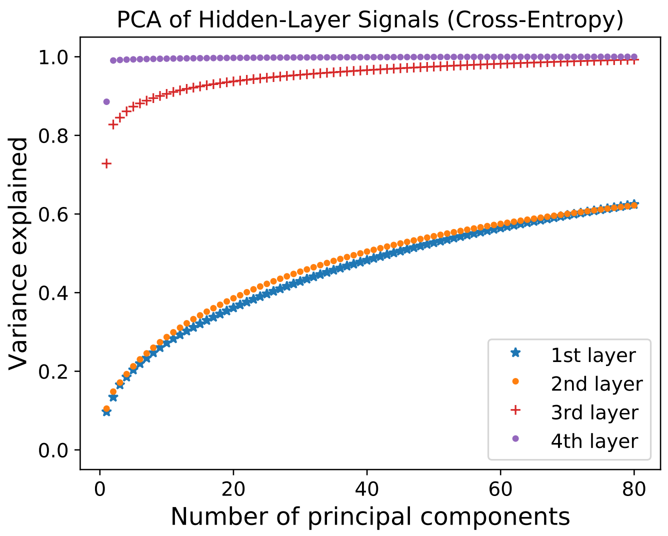


Figure 4.4: Principal component analysis for all layers of the network trained with the cross-entropy loss. The plot shows the variance explained versus number of principal components used for each layer.

gradient methods, cross-entropy loss function and the low-dimensionality of the training dataset (at least in some domain) has a responsibility for this problem. Note that SVM with the radial basis function was shown to be robust against adversarial examples, and this was attributed to the high nonlinearity of the radial basis function in (Goodfellow et al., 2015). Given that the SVM uses neither the cross entropy loss function nor the gradient descent algorithm for training, we argue that the robustness of SVM is no surprise – independent of its nonlinearity.

Low-dimensionality of the training dataset. As stated in Remark 4.3, as the dimension of the affine subspace containing the training dataset gets very small compared to the dimension of the input space, the training algorithm will become more likely to yield a small margin for the classifier. This observation confirms the results of (Marzi et al., 2018), which showed that if the set of training data is projected onto a low-dimensional subspace before feeding into a neural network, the performance of the network against adversarial examples is improved – since projecting the inputs onto a low-dimensional domain corresponds to decreasing the dimension of the input space. Even though this method is effective, it requires the knowledge of the domain in which the training points are low-dimensional. Because this knowledge will not always be available, finding alternative training algorithms and loss functions that are suited for low-dimensional data is still an important direction for future research.

Robust optimization. Using robust optimization techniques to train neural networks has been shown to be effective against adversarial examples (Madry et al., 2018; Athalye et al., 2018). Note that these techniques could be considered as inflating the training points by a presumed amount and training the classifier with these inflated points. Consequently, as long as the cross-entropy loss is involved, the decision boundaries of the neural network will still be in the vicinity of the inflated points. Therefore, even though the classifier is robust against the disturbances of the presumed magnitude, the margin of the classifier could still be much smaller than what it could potentially be.

Differential training. We introduced differential training, which allows the feature mapping to remain trainable while ensuring a large margin between different classes of points. Therefore, this method combines the benefits of neural networks with those of support vector machines. Even though moving from $2N$ training points to N^2 seems prohibitive, it points out that a true classification should in fact be able to differentiate between the pairs that are hardest to differentiate, and this search will necessarily require an N^2 term. Some heuristic methods are likely to be effective, such as considering only a smaller subset of points closer to the boundary and updating this set of points as needed during training. If a neural network is trained with this procedure, the network will be forced to find features that are able to tell apart between the hardest pairs.

Nonseparable data. What happens when the training data is not linearly separable is an open direction for future work. However, as stated in Remark 4.4, this case is not expected to arise for the state-of-the-art networks, since they have been shown to achieve zero training error even on randomly generated datasets (Zhang et al., 2017), which implies that the features represented by the output of their penultimate layer eventually become linearly separable.

4.9 Proofs

This sections provides the proofs for the theorems and the corollaries in this chapter.

4.9.1 Proof of Theorem 4.1

Theorem 4.1 could be proved by using Theorem 4.2, but we provide an independent proof here. Gradient descent algorithm with learning rate δ on the cross-entropy loss (4.1) yields

$$\frac{d\tilde{w}}{dt} = \delta\tilde{x}\frac{e^{-\tilde{w}^\top\tilde{x}}}{1+e^{-\tilde{w}^\top\tilde{x}}} + \delta\tilde{y}\frac{e^{-\tilde{w}^\top\tilde{y}}}{1+e^{-\tilde{w}^\top\tilde{y}}}.$$

If $\tilde{w}(0)=0$, then $\tilde{w}(t) = p(t)\tilde{x} + q(t)\tilde{y}$ for all $t \geq 0$, where

$$\dot{p} = \delta\frac{e^{-p\|\tilde{x}\|^2 - q\langle\tilde{x},\tilde{y}\rangle}}{1+e^{-p\|\tilde{x}\|^2 - q\langle\tilde{x},\tilde{y}\rangle}}, \quad \dot{q} = \delta\frac{e^{-q\|\tilde{y}\|^2 - p\langle\tilde{x},\tilde{y}\rangle}}{1+e^{-q\|\tilde{y}\|^2 - p\langle\tilde{x},\tilde{y}\rangle}}.$$

Define

$$\alpha = p\|\tilde{x}\|^2 + q\langle\tilde{x},\tilde{y}\rangle, \quad \beta = q\|\tilde{y}\|^2 + p\langle\tilde{x},\tilde{y}\rangle,$$
$$a = \delta\|\tilde{x}\|^2 = \delta\sigma_x, \quad c = \delta\|\tilde{y}\|^2 = \delta\sigma_y, \quad b = \delta\langle\tilde{x},\tilde{y}\rangle = \delta\sigma_{xy}.$$

Then we can write

$$\dot{\alpha} = a\frac{e^{-\alpha}}{1+e^{-\alpha}} + b\frac{e^{-\beta}}{1+e^{-\beta}},$$
$$\dot{\beta} = c\frac{e^{-\beta}}{1+e^{-\beta}} + b\frac{e^{-\alpha}}{1+e^{-\alpha}}.$$

Finally, define $z = e^\alpha$ and $v = e^\beta$ so that

$$\dot{z} = \frac{z}{z+1}\left(a + b\frac{z+1}{v+1}\right),$$
$$\dot{v} = \frac{v}{v+1}\left(c + b\frac{v+1}{z+1}\right).$$

Without loss of generality, assume $c \geq a$. Before we proceed, note that

$$\frac{d}{dt}\left(\frac{z}{v}\right) = \frac{c-b}{z+1}\frac{z}{v}\left(\frac{a-b}{c-b} - \frac{z+1}{v+1}\right),$$
$$\frac{d}{dt}\left(\frac{z+1}{v+1}\right) = \frac{z}{(v+1)^2}\left[\frac{v+1}{z+1}a - \frac{v}{z}b - \left(\frac{v}{z}\frac{z+1}{v+1}c - b\right)\right].$$

Let u and w denote $\frac{z}{v}$ and $\frac{z+1}{v+1}$, respectively. Then,

$$\dot{u} < 0 \quad \text{if } w > \frac{a-b}{c-b}$$
$$\dot{u} > 0 \quad \text{if } w < \frac{a-b}{c-b}$$
$$\dot{w} < 0 \quad \text{if } \left(\frac{a}{w} + b\right)u < cw + b$$
$$\dot{w} > 0 \quad \text{if } \left(\frac{a}{w} + b\right)u > cw + b$$

Lemma 4.1. *If $b = 0$, then*

$$\lim_{t\to\infty} \frac{\tilde{w}(t)}{\log(t)} = \frac{1}{\|\tilde{x}\|}\tilde{x} + \frac{1}{\|\tilde{y}\|}\tilde{y}.$$

Proof. Note that

$$\frac{d(z+\log(z))}{dt} = a \implies z(t) - z_0 + \log(z(t)/z_0) = at,$$

$$\frac{d(v+\log(v))}{dt} = c \implies v(t) - v_0 + \log(v(t)/v_0) = ct.$$

Then,

$$\lim_{t\to\infty} \frac{\alpha(t)}{\log(t)} = \lim_{t\to\infty} \frac{\log(z(t))}{\log(t)} = 1 = \lim_{t\to\infty} \frac{\log(v(t))}{\log(t)} = \lim_{t\to\infty} \frac{\beta(t)}{\log(t)},$$

and

$$\lim_{t\to\infty} \frac{\tilde{w}(t)}{\log(t)} = \frac{\tilde{x}}{\|\tilde{x}\|^2} + \frac{\tilde{y}}{\|\tilde{y}\|^2}.$$ ∎

Lemma 4.2. *If $b < 0$, then there exists $t_0 \in (0, \infty)$ such that*

$$\frac{-b}{c} \le \frac{z+1}{v+1} \le \frac{a}{-b} \quad \forall t \ge t_0.$$

Proof. Note that $\frac{-b}{c} \le \frac{a-b}{c-b} \le \frac{a}{-b}$ because $b < 0$. First assume $\frac{z_0+1}{v_0+1} \ge \frac{a}{-b}$. Then, $\dot{z} \le 0$ and

$$\dot{v} = \frac{v}{v+1}\left(c + b\frac{v+1}{z+1}\right) \ge \frac{v}{v+1}\left(c - \frac{b^2}{a}\right) \ge \frac{v_0}{v_0+1}\frac{ac-b^2}{a},$$

which implies that

$$\frac{z+1}{v+1} \le (z_0+1)\left(\frac{v_0}{v_0+1}\frac{ac-b^2}{a}t + 1\right)^{-1}$$

as long as $\frac{z+1}{v+1} \ge \frac{a}{-b}$, and this can be satisfied only for a finite time. Now assume $\frac{z_0+1}{v_0+1} \le \frac{-b}{c}$. Then, $\dot{v} \le 0$ and

$$\dot{z} = \frac{z}{z+1}\left(a + b\frac{z+1}{v+1}\right) \ge \frac{z_0}{z_0+1}\frac{ac-b^2}{c},$$

which implies

$$\frac{z+1}{v+1} \ge \left(\frac{z_0}{z_0+1}\frac{ac-b^2}{c}t + 1\right)(v_0+1)^{-1}$$

as long as $\frac{z+1}{v+1} \le \frac{-b}{c}$, and this can be satisfied only for a finite time as well. ∎

Lemma 4.3. *If $b < 0$, then*

$$0 \le \dot{z} \le \frac{ac-b^2}{c}, \quad 0 \le \dot{v} \le \frac{ac-b^2}{a} \quad \forall t \ge t_0,$$

where t_0 is given by Lemma 4.2.

Proof.

$$\dot{z} = \frac{z}{z+1}\left(a + b\frac{z+1}{v+1}\right) \leq \frac{z}{z+1}\frac{ac-b^2}{c} \leq \frac{ac-b^2}{c}$$

$$\dot{v} = \frac{v}{v+1}\left(c + b\frac{v+1}{z+1}\right) \leq \frac{v}{v+1}\frac{ac-b^2}{a} \leq \frac{ac-b^2}{a}$$

■

Lemma 4.4. *If* $b < 0$, *then*

$$\lim_{t\to\infty}\frac{\log(z)}{\log t} = \lim_{t\to\infty}\frac{\log(v)}{\log t} = 1$$

Proof. From Lemma 4.3,

$$\dot{v} \leq 0 \iff c + b\frac{v+1}{z+1} \geq 0,$$

and

$$v \leq \frac{ac-b^2}{a}t + v_0'.$$

Combining these two inequalities, we have

$$c + \frac{b}{a}\left[(ac-b^2)t + v_0' + 1\right]\frac{1}{z+1} \geq 0 \iff z+1 \geq \frac{-b}{ac}[(ac-b^2)t + v_0' + 1]$$

As a result,

$$\frac{(-b)(ac-b^2)}{ac}t + z_1 \leq z(t) \leq \frac{ac-b^2}{c}t + z_2 \quad \forall t \geq t_0 \implies \lim_{t\to\infty}\frac{\log(z)}{\log(t)} = 1.$$

By using Lemma 4.2,

$$\log(-b/c) \leq \log(z+1) - \log(v+1) \leq \log(-a/b) \implies \lim_{t\to\infty}\frac{\log(v)}{\log(t)} = 1.$$

■

Lemma 4.5. *If* $b < 0$, *then*

$$\lim_{t\to\infty}\frac{\tilde{w}(t)}{\log(t)} = \delta\frac{c-b}{ac-b^2}\tilde{x} + \delta\frac{a-b}{ac-b^2}\tilde{y} = \frac{\sigma_y - \sigma_{xy}}{\sigma_x\sigma_y - \sigma_{xy}^2}\tilde{x} + \frac{\sigma_x - \sigma_{xy}}{\sigma_x\sigma_y - \sigma_{xy}^2}\tilde{y}$$

Proof. Solving the set of equations

$$1 = \lim\frac{\alpha}{\log(t)} = \frac{a}{\delta}\lim\frac{p}{\log(t)} + \frac{b}{\delta}\lim\frac{q}{\log(t)},$$

$$1 = \lim\frac{\beta}{\log(t)} = \frac{c}{\delta}\lim\frac{q}{\log(t)} + \frac{b}{\delta}\lim\frac{p}{\log(t)},$$

we obtain

$$\lim_{t\to\infty}\frac{p}{\log(t)} = \delta\frac{c-b}{ac-b^2}, \quad \lim_{t\to\infty}\frac{q}{\log(t)} = \delta\frac{a-b}{ac-b^2}$$

■

Lemma 4.6. *If $b > 0$, then*

$$\lim_{t\to\infty} \frac{z}{v} = \lim_{t\to\infty} \frac{z+1}{v+1} = \begin{cases} 0 & \text{if } a \le b \\ \frac{a-b}{c-b} & \text{if } a > b \end{cases}$$

Proof. Note that $\dot{z} \geq a/2$ and $\dot{v} \geq c/2$; therefore,

$$\lim_{t\to\infty} \frac{z+1}{v+1} = \lim_{t\to\infty} \frac{z}{v} \implies \lim_{t\to\infty} u = \lim_{t\to\infty} w$$

if either side exists. Remember that

$$\dot{w} < 0 \iff u < \frac{cw^2 + bw}{a + bw} =: f(w).$$

We can compute

$$f'(w) = \frac{2acw + bcw^2 + ab}{b^2w^2 + 2abw + a^2}.$$

The function f is strictly increasing and convex for $w > 0$. We have

$$f(0) = 0,$$

$$f\left(\frac{a-b}{c-b}\right) = \frac{a-b}{c-b}.$$

Therefore, when $b \geq a$, the only fixed point of f over $[0, \infty)$ is the origin, and when $a > b$, 0 and $(a-b)/(c-b)$ are the only fixed points of f over $[0, \infty)$.

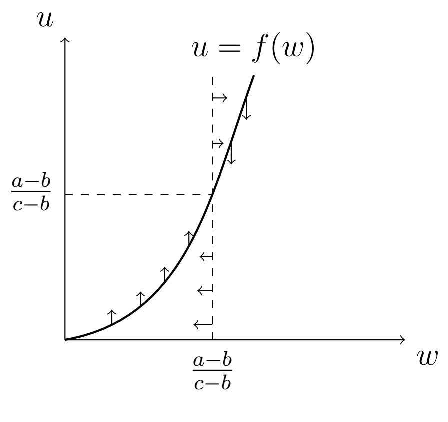


Figure 4.5: Stationary points of function f.

Figure 4.5 shows the curves over which $\dot{u} = 0$ and $\dot{w} = 0$. Since $\lim_{t\to\infty} u = \lim_{t\to\infty} w$, the only points (u, w) can converge to are the fixed points of f. Remember that

$$\dot{u} = \frac{c-b}{z+1} u \left(\frac{a-b}{c-b} - w\right),$$

so when $a > b$, the origin $(0,0)$ is unstable in the sense of Lyapunov, and (u,w) cannot converge to it. Otherwise, $(0,0)$ is the only fixed point, and it is stable. As a result,

$$\lim_{t\to\infty} \frac{z}{v} = \lim_{t\to\infty} \frac{z+1}{v+1} = \begin{cases} 0 & \text{if } a \le b \\ \frac{a-b}{c-b} & \text{if } a > b \end{cases} \qquad \blacksquare$$

Lemma 4.7. *If $a > b > 0$, then*

$$\lim_{t\to\infty} \frac{\tilde{w}(t)}{\log(t)} = \delta \frac{c-b}{ac-b^2}\tilde{x} + \delta \frac{a-b}{ac-b^2}\tilde{y} = \frac{\sigma_y - \sigma_{xy}}{\sigma_x\sigma_y - \sigma_{xy}^2}\tilde{x} + \frac{\sigma_x - \sigma_{xy}}{\sigma_x\sigma_y - \sigma_{xy}^2}\tilde{y}.$$

Proof. From Lemma 4.6,

$$\lim_{t\to\infty} \frac{z}{t} = \lim_{t\to\infty} \dot{z} = \lim_{t\to\infty} \frac{z}{z+1}\left(a + b\frac{z+1}{v+1}\right) = a + b\frac{a-b}{c-b} = \frac{ac-b^2}{c-b},$$

$$\lim_{t\to\infty} \frac{v}{t} = \frac{ac-b^2}{a-b}.$$

Consequently,

$$\lim_{t\to\infty} \frac{\log(z)}{\log(t)} = \lim_{t\to\infty} \frac{\log(v)}{\log(t)} = 1$$

which gives the same solution as Lemma 4.5:

$$\lim_{t\to\infty} \frac{p}{\log(t)} = \delta\frac{c-b}{ac-b^2}, \quad \lim_{t\to\infty} \frac{q}{\log(t)} = \delta\frac{a-b}{ac-b^2}. \qquad \blacksquare$$

Lemma 4.8. *If $b \ge a$, then*

$$\lim_{t\to\infty} \frac{\tilde{w}(t)}{\log(t)} = \frac{1}{\|\tilde{x}\|}\tilde{x}$$

Proof.

$$\lim_{t\to\infty} \frac{z}{t} = a, \quad \lim_{t\to\infty} \frac{\log(z)}{\log(t)} = 1,$$

$$\lim_{t\to\infty} \frac{v}{t} = \lim_{t\to\infty} \dot{v} = \infty$$

$$\lim_{t\to\infty} \frac{\log(v)}{\log(t)} = \lim_{t\to\infty} \frac{\dot{v}}{v}t = \lim_{t\to\infty} \frac{1}{v+1}\left(c + b\frac{v+1}{z+1}\right)t = \lim_{t\to\infty} \frac{ct}{v+1} + \lim_{t\to\infty} \frac{bt}{z+1} = \frac{b}{a}$$

$$\lim_{t\to\infty} \frac{p}{\log(t)} = \frac{1}{\|\tilde{x}\|^2}, \quad \lim_{t\to\infty} \frac{q}{\log(t)} = 0 \implies \lim_{t\to\infty} \frac{px+qy}{\log(t)} = \frac{1}{\|\tilde{x}\|^2}\tilde{x} \qquad \blacksquare$$

Proof of Theorem 4.1. Lemma 4.1, Lemma 4.5, Lemma 4.7 and Lemma 4.8 prove all cases of Theorem 4.1. ■

4.9.2 Proof of Corollary 4.1

Since $x^\top y = 1$, we have $\sigma_{xy} = \tilde{x}^\top \tilde{y} = x^\top y - 1 = 0 < a$. Then the normal vector of the decision boundary is proportional to $\frac{1}{\sigma_x}x + \frac{1}{\sigma_y}y$. For the angle between this vector and the solution of the SVM, we can write

$$\cos\psi = \frac{\langle \frac{1}{\sigma_x}x + \frac{1}{\sigma_y}y, x+y\rangle}{\left\|\frac{1}{\sigma_x}x + \frac{1}{\sigma_y}y\right\| \|x+y\|} = \frac{2}{\left\|\frac{1}{\sigma_x}x + \frac{1}{\sigma_y}y\right\| \sqrt{\sigma_x+\sigma_y}}.$$

We can obtain a lower bound for square of the denominator as

$$\left\|\frac{1}{\sigma_x}x + \frac{1}{\sigma_y}y\right\|^2 (\sigma_x+\sigma_y) \geq \left(2 + \frac{\sigma_y}{\sigma_x} - \frac{\sigma_y}{\sigma_x^2}\right) + \left(\frac{1}{\sigma_x} + \frac{1}{\sigma_y} + \frac{\sigma_x}{\sigma_y} - \frac{\sigma_x}{\sigma_y^2}\right) \geq 2 + \frac{\sigma_y}{\sigma_x}\left(1 - \frac{1}{\sigma_x}\right).$$

As a result,

$$\cos^2\psi \leq \frac{4}{2 + \frac{\sigma_y}{\sigma_x}\left(1 - \frac{1}{\sigma_x}\right)}.$$ ■

4.9.3 Proof of Theorem 4.3

Assume that $\overline{w} = u + \sum_{k\in K}\alpha_k r_k$, where $u \in \mathbb{R}^n$ and $\langle u, r_k\rangle = 0$ for all $k \in K$. By denoting $z = [w^\top\ b]^\top$, the Lagrangian of the problem (4.3) can be written as

$$\frac{1}{2}\|w\|^2 + \frac{1}{2}b^2 + \sum_{i\in\mathcal{I}}\mu_i(1 - \langle w, x_i\rangle - b) + \sum_{j\in\mathcal{J}}\nu_j(-1 + \langle w, x_j\rangle + b),$$

where $\mu_i \geq 0$ for all $i \in \mathcal{I}$ and $\nu_j \geq 0$ for all $j \in \mathcal{J}$. KKT conditions for the optimality of $\overline{w}$ and B requires that

$$\overline{w} = \sum_{i\in\mathcal{I}}\mu_i x_i - \sum_{j\in\mathcal{J}}\nu_j x_j, \quad B = \sum_{i\in\mathcal{I}}\mu_i - \sum_{j\in\mathcal{J}}\nu_j,$$

and consequently, for each $k \in K$,

$$\begin{aligned}\langle \overline{w}, r_k\rangle &= \sum_{i\in\mathcal{I}}\mu_i\langle x_i, r_k\rangle - \sum_{j\in\mathcal{J}}\nu_j\langle x_j, r_k\rangle \\ &= \sum_{i\in\mathcal{I}}\Delta_k\mu_i - \sum_{j\in\mathcal{J}}\Delta_k\nu_j = B\Delta_k.\end{aligned}$$

Then, we can write $\overline{w}$ as

$$\overline{w} = u + \sum_{k\in K}B\Delta_k r_k.$$

Let $\langle w_{\text{SVM}}, \cdot\rangle + b_{\text{SVM}} = 0$ denote the hyperplane obtained as the solution of maximum hard-margin SVM in the original space — not in the augmented space. Then w_{SVM} solves

$$\begin{aligned}&\underset{w}{\text{minimize}} && \|w\|^2 \\ &\text{subject to} && \langle w, x_i - x_j\rangle \geq 2 \quad \forall i\in\mathcal{I}, \forall j\in\mathcal{J}.\end{aligned} \tag{4.11}$$

Since the vector u also satisfies $\langle u, x_i - x_j\rangle = \langle w, x_i - x_j\rangle \geq 2$ for all $i \in \mathcal{I}, j \in \mathcal{J}$, we have $\|u\| \geq \|w_{\text{SVM}}\| = \frac{1}{\gamma_{\text{OPT}}}$. As a result, the margin obtained by minimizing the cross-entropy loss is

$$\frac{1}{\|\overline{w}\|} = \frac{1}{\sqrt{\|u\|^2 + \sum \|B\Delta_k r_k\|^2}} \leq \frac{1}{\sqrt{\frac{1}{\gamma_{\text{OPT}}^2} + B^2 \sum \Delta_k^2}}.$$

■

4.9.4 Proof of Theorem 4.4

If $B < 0$, we could consider the hyperplane $\langle \overline{w}, \cdot\rangle - B = 0$ for the points $\{-x_i\}_{i\in\mathcal{I}}$ and $\{-x_j\}_{j\in\mathcal{J}}$, which would have the identical margin due to symmetry. Therefore, without loss of generality, assume $B \geq 0$. As in the proof of Theorem 4.3, KKT conditions for the optimality of $\overline{w}$ and B requires

$$\overline{w} = \sum_{i\in\mathcal{I}} \mu_i x_i - \sum_{j\in\mathcal{J}} \nu_j x_j, \quad B = \sum_{i\in\mathcal{I}} \mu_i - \sum_{j\in\mathcal{J}} \nu_j$$

where $\mu_i \geq 0$ and $\nu_j \geq 0$ for all $i \in \mathcal{I}, j \in \mathcal{J}$. Note that for each $k \in K$,

$$\begin{aligned}\langle \overline{w}, r_k\rangle &= \textstyle\sum_{i\in\mathcal{I}} \mu_i \langle x_i, r_k\rangle - \sum_{j\in\mathcal{J}} \nu_j \langle x_j, r_k\rangle \\ &= B\Delta_k + \textstyle\sum_{i\in\mathcal{I}} \mu_i(\langle x_i, r_k\rangle - \Delta_k) \\ &\quad - \textstyle\sum_{j\in\mathcal{J}} \nu_j(\langle -x_j, r_k\rangle - \Delta_k) \;\geq\; B\Delta_k.\end{aligned}$$

Since $\{r_k\}_{k\in K}$ is an orthonormal set of vectors,

$$\|\overline{w}\|^2 \geq \textstyle\sum_{k\in K} \langle \overline{w}, r_k\rangle^2 \geq \sum_{k\in K} B^2\Delta_k^2.$$

The result follows from the fact that $\|\overline{w}\|^{-1}$ is an upper bound on the margin. ■

4.9.5 Proof of Theorem 4.5

Lemma 4.9. *The direction parameters $\overline{w}, \overline{V}$ and $\overline{b}$ satisfy*

$$\|\overline{w}\|_2^2 = \|\overline{V}\|_F^2 + \|\overline{b}\|_2^2.$$

Proof The continuous-time gradient descent algorithm gives

$$\frac{dw}{dt} = \sum_{i\in\mathcal{I}} G_i(Vx_i + b)\frac{e^{-w^\top G_i(Vx_i+b)}}{1 + e^{-w^\top G_i(Vx_i+b)}} - \sum_{j\in\mathcal{J}} G_j(Vx_j + b)\frac{e^{w^\top G_j(Vx_j+b)}}{1 + e^{w^\top G_j(Vx_j+b)}} \tag{4.12a}$$

$$\frac{dV}{dt} = \sum_{i\in\mathcal{I}} G_i w x_i^\top \frac{e^{-w^\top G_i(Vx_i+b)}}{1 + e^{-w^\top G_i(Vx_i+b)}} - \sum_{j\in\mathcal{J}} G_j w x_j^\top \frac{e^{w^\top G_j(Vx_j+b)}}{1 + e^{w^\top G_j(Vx_j+b)}} \tag{4.12b}$$

$$\frac{db}{dt} = \sum_{i\in\mathcal{I}} G_i w \frac{e^{-w^\top G_i(Vx_i+b)}}{1 + e^{-w^\top G_i(Vx_i+b)}} - \sum_{j\in\mathcal{J}} G_j w \frac{e^{w^\top G_j(Vx_j+b)}}{1 + e^{w^\top G_j(Vx_j+b)}} \tag{4.12c}$$

Note that, for all $t \geq 0$ we have

$$\frac{d}{dt}\langle w(t), w(t)\rangle = \frac{d}{dt}\langle V(t), V(t)\rangle + \frac{d}{dt}\langle b(t), b(t)\rangle.$$

Consequently, for all $t \geq 0$:

$$\frac{\|w(t)\|_2^2 - \|w(0)\|_2^2}{h^2(t)} = \frac{\|V(t)\|_F^2 - \|V(0)\|_F^2}{h^2(t)} + \frac{\|b(t)\|_2^2 - \|b(0)\|_2^2}{h^2(t)},$$

which proves that

$$\|\overline{w}\|_2^2 = \|\overline{V}\|_F^2 + \|\overline{b}\|_2^2.$$

■

Lemma 4.10. *The solution obtained by the continuous-time gradient descent algorithm satisfies*

$$\begin{aligned}\overline{w} &= \sum_{i\in\mathcal{I}} \mu_i G_i(\overline{V}x_i + \overline{b}) - \sum_{j\in\mathcal{J}} \mu_j G_j(\overline{V}x_j + \overline{b}) \\ \overline{V} &= \sum_{i\in\mathcal{I}} \mu_i G_i \overline{w} x_i^\top - \sum_{j\in\mathcal{J}} \mu_j G_j \overline{w} x_j^\top \\ \overline{b} &= \sum_{i\in\mathcal{I}} \mu_i G_i \overline{w} - \sum_{j\in\mathcal{J}} \mu_j G_j \overline{w}\end{aligned}$$

for some set of nonnegative scalars $\{\mu_s : s \in \mathcal{I} \cup \mathcal{J}\}$.

Proof Given the dynamics (4.12) of the gradient descent algorithm, define the set of support vectors as

$$\mathcal{I}^{\text{sup}} = \left\{ i \in \mathcal{I} : \lim_{t\to\infty} \frac{e^{-w^\top (Vx_{i'}+b)_+}}{e^{-w^\top (Vx_i+b)_+}} < \infty \; \forall i' \in \mathcal{I}, \; \lim_{t\to\infty} \frac{e^{w^\top (Vx_{j'}+b)_+}}{e^{-w^\top (Vx_i+b)_+}} < \infty \; \forall j' \in \mathcal{J} \right\},$$

$$\mathcal{J}^{\text{sup}} = \left\{ j \in \mathcal{J} : \lim_{t\to\infty} \frac{e^{-w^\top (Vx_{i'}+b)_+}}{e^{w^\top (Vx_j+b)_+}} < \infty \; \forall i' \in \mathcal{I}, \; \lim_{t\to\infty} \frac{e^{w^\top (Vx_{j'}+b)_+}}{e^{w^\top (Vx_j+b)_+}} < \infty \; \forall j' \in \mathcal{J} \right\}.$$

Note that the points corresponding to these indices dominate the dynamics of the algorithm as the training continues. Let $\overline{b}_{s_1}, \overline{b}_{s_2}$ be two nonzero coordinates of $\overline{b}$, let x_S be any of the support vectors. Then,

$$
\begin{aligned}
\frac{\overline{b}_{s_1}}{\overline{b}_{s_2}} &= \lim_{t\to\infty} \frac{b_{s_1}(t)}{b_{s_2}(t)} \\
&= \lim_{t\to\infty} \frac{\frac{db_{s_1}}{dt}}{\frac{db_{s_2}}{dt}} \\
&= \lim_{t\to\infty} \frac{\sum_{i\in\mathcal{I}} e_{s_1}^\top G_i w \frac{e^{-w^\top G_i(Vx_i+b)}}{1+e^{-w^\top G_i(Vx_i+b)}} - \sum_{j\in\mathcal{J}} e_{s_1}^\top G_j w \frac{e^{w^\top G_j(Vx_j+b)}}{1+e^{w^\top G_j(Vx_j+b)}}}{\sum_{i\in\mathcal{I}} e_{s_2}^\top G_i w \frac{e^{-w^\top G_i(Vx_i+b)}}{1+e^{-w^\top G_i(Vx_i+b)}} - \sum_{j\in\mathcal{J}} e_{s_2}^\top G_j w \frac{e^{w^\top G_j(Vx_j+b)}}{1+e^{w^\top G_j(Vx_j+b)}}} \\
&= \lim_{t\to\infty} \frac{\sum_{i\in\mathcal{I}} e_{s_1}^\top G_i w \frac{e^{-w^\top G_i(Vx_i+b)}}{e^{w^\top G_S(Vx_S+b)}} - \sum_{j\in\mathcal{J}} e_{s_1}^\top G_j w \frac{e^{w^\top G_j(Vx_j+b)}}{e^{w^\top G_S(Vx_S+b)}}}{\sum_{i\in\mathcal{I}} e_{s_2}^\top G_i w \frac{e^{-w^\top G_i(Vx_i+b)}}{e^{w^\top G_S(Vx_S+b)}} - \sum_{j\in\mathcal{J}} e_{s_2}^\top G_j w \frac{e^{w^\top G_j(Vx_j+b)}}{e^{w^\top G_S(Vx_S+b)}}} \\
&= \frac{\sum_{i\in\mathcal{I}} e_{s_1}^\top G_i \overline{w} \mu_i - \sum_{j\in\mathcal{J}} e_{s_1}^\top G_j \overline{w} \mu_j}{\sum_{i\in\mathcal{I}} e_{s_2}^\top G_i \overline{w} \mu_i - \sum_{j\in\mathcal{J}} e_{s_2}^\top G_j \overline{w} \mu_j}
\end{aligned}
$$

where

$$
\begin{aligned}
\mu_i &\propto \lim_{t\to\infty} \frac{e^{-w^\top G_i(Vx_i+b)}}{e^{w^\top G_S(Vx_S+b)}} \quad \forall i \in \mathcal{I}, \\
\mu_j &\propto \lim_{t\to\infty} \frac{e^{-w^\top G_j(Vx_j+b)}}{e^{w^\top G_S(Vx_S+b)}} \quad \forall j \in \mathcal{J}.
\end{aligned}
$$

Similarly, if $e_{s_1}^\top \overline{b}$ and $e_{s_2}^\top \overline{V} e_{s_3}$ are two nonzero elements of $\overline{b}$ and $\overline{V}$, then

$$
\begin{aligned}
\frac{e_{s_1}^\top \overline{b}}{e_{s_2}^\top \overline{V} e_{s_3}} &= \lim_{t\to\infty} \frac{e_{s_1}^\top b}{e_{s_2}^\top V e_{s_3}} \\
&= \lim_{t\to\infty} \frac{e_{s_1}^\top \frac{db}{dt}}{e_{s_2}^\top \frac{dV}{dt} e_{s_3}} \\
&= \lim_{t\to\infty} \frac{\sum_{i\in\mathcal{I}} e_{s_1}^\top G_i w \frac{e^{-w^\top G_i(Vx_i+b)}}{e^{w^\top G_S(Vx_S+b)}} - \sum_{j\in\mathcal{J}} e_{s_1}^\top G_j w \frac{e^{w^\top G_j(Vx_j+b)}}{e^{w^\top G_S(Vx_S+b)}}}{\sum_{i\in\mathcal{I}} e_{s_2}^\top G_i w x_i e_{s_3} \frac{e^{-w^\top G_i(Vx_i+b)}}{e^{w^\top G_S(Vx_S+b)}} - \sum_{j\in\mathcal{J}} e_{s_2}^\top G_j w x_j e_{s_3} \frac{e^{w^\top G_j(Vx_j+b)}}{e^{w^\top G_S(Vx_S+b)}}} \\
&= \frac{\sum_{i\in\mathcal{I}} e_{s_1}^\top G_i \overline{w} \mu_i - \sum_{j\in\mathcal{J}} e_{s_1}^\top G_j \overline{w} \mu_j}{\sum_{i\in\mathcal{I}} e_{s_2}^\top G_i \overline{w} x_i e_{s_3} \mu_i - \sum_{j\in\mathcal{J}} e_{s_2}^\top G_j \overline{w} x_j e_{s_3} \mu_j}.
\end{aligned}
$$

As a result, we have

$$
\begin{aligned}
\overline{V} &= \sum_{i\in\mathcal{I}} \mu_i G_i \overline{w} x_i^\top - \sum_{j\in\mathcal{J}} \mu_j G_j \overline{w} x_j^\top \\
\overline{b} &= \sum_{i\in\mathcal{I}} \mu_i G_i \overline{w} - \sum_{j\in\mathcal{J}} \mu_j G_j \overline{w}
\end{aligned}
$$

for some set of nonnegative scalars $\{\mu_s : s \in \mathcal{I} \cup \mathcal{J}\}$.

An identical analysis of $\overline{w}_{s_1}/\overline{w}_{s_2}$ for two nonzero coordinates of $\overline{w}$ shows that

$$\overline{w} = \sum_{i\in\mathcal{I}} \alpha\mu_i G_i(\overline{V}x_i + \overline{b}) - \sum_{j\in\mathcal{J}} \alpha\mu_j G_j(\overline{V}x_j + \overline{b})$$

for some $\alpha \in (0, \infty)$. In order to find the value of α, note that

$$\begin{aligned}
\langle \overline{w}, \overline{w}\rangle &= \left\langle \overline{w}, \sum_{i\in\mathcal{I}} \alpha\mu_i G_i(\overline{V}x_i + \overline{b}) - \sum_{j\in\mathcal{J}} \alpha\mu_j G_j(\overline{V}x_j + \overline{b})\right\rangle \\
&= \alpha \left\langle \sum_{i\in\mathcal{I}} \mu_i G_i \overline{w} x_i^\top - \sum_{j\in\mathcal{J}} \mu_j G_j \overline{w} x_j^\top, \overline{V}\right\rangle + \alpha \left\langle \sum_{i\in\mathcal{I}} \mu_i G_i \overline{w} - \sum_{j\in\mathcal{J}} \mu_j G_j \overline{w}, \overline{b}\right\rangle \\
&= \alpha\langle \overline{V}, \overline{V}\rangle + \alpha\langle \overline{b}, \overline{b}\rangle
\end{aligned}$$

which shows that α must be 1 due to Lemma 4.9. This completes the proof. ■

Proof of Theorem 4.5. For any $k \in [r]$, let $\overline{w}_k$, $\overline{V}_k$ and $\overline{b}_k$ denote the k-th row of $\overline{w}$, $\overline{V}$ and $\overline{b}$, respectively. From Lemma 4.9, we have that

$$\overline{w}_k^2 = \|\overline{V}_k\|_2^2 + \overline{b}_k^2.$$

Let $\mathcal{I}^k \subseteq \mathcal{I}$ and $\mathcal{J}^k \subseteq \mathcal{J}$ denote the support vectors that activate the j-th node in the hidden layer, i.e.,

$$\begin{aligned}
\mathcal{I}^k &= \left\{ i \in \mathcal{I} : \overline{V}_k x_i + \overline{b}_k > 0,\ \overline{w}^\top(\overline{V}x_i + \overline{b})_+ = 1\right\}, \\
\mathcal{J}^k &= \left\{ j \in \mathcal{J} : \overline{V}_k x_j + \overline{b}_k > 0,\ \overline{w}^\top(\overline{V}x_j + \overline{b})_+ = -1\right\}.
\end{aligned}$$

Then we have

$$\begin{aligned}
\overline{w}_k &= \sum\nolimits_{i\in\mathcal{I}^k} \mu_i(\overline{V}_k x_i + \overline{b}_k) - \sum\nolimits_{j\in\mathcal{J}^k} \mu_j(\overline{V}_k x_j + \overline{b}_k) \\
\overline{V}_k &= \sum\nolimits_{i\in\mathcal{I}^k} \mu_i \overline{w}_k x_i^\top - \sum\nolimits_{j\in\mathcal{J}^k} \mu_j \overline{w}_k x_j^\top \\
\overline{b}_k &= \sum\nolimits_{i\in\mathcal{I}^k} \mu_i \overline{w}_k - \sum\nolimits_{j\in\mathcal{J}^k} \mu_j \overline{w}_k.
\end{aligned}$$

Plugging the expressions for $\overline{V}_k$ and $\overline{b}_k$ into that of $\overline{w}_k$:

$$\overline{w}_k = \overline{w}_k \mu^\top X_k^\top X_k \mu + \frac{1}{\overline{w}_k}\overline{b}_k^2,$$

which gives

$$\begin{aligned}
\mu^\top X_k^\top X_k \mu = 1 - \frac{\overline{b}_k^2}{\overline{w}_k^2} &\implies \lambda_{\min}(X_k^\top X_k)\|\mu\|_2^2 \leq 1 - \frac{\overline{b}_k^2}{\overline{w}_k^2}, \\
&\implies \lambda_{\min}(X_k^\top X_k)\frac{\|\mu\|_1^2}{n_k} \leq 1 - \frac{\overline{b}_k^2}{\overline{w}_k^2},
\end{aligned}$$

where X_k is a matrix with columns $\{x_i : i \in \mathcal{I}^k\}$ and $\{-x_j : j \in \mathcal{J}^k\}$, μ is a column vector with elements $\{\mu_s : s \in \mathcal{I}^k \cup \mathcal{J}^k\}$, and n_k is the number of support vectors that activate node k. Note that if the columns of X_k are not linearly independent, then μ_s can be arbitrarily large for any support vector x_s. From the last inequality, we have

$$\sum\nolimits_{s\in\mathcal{I}^k\cup\mathcal{J}^k} \mu_s \le \frac{\sqrt{|\mathcal{I}^k| + |\mathcal{J}^k|}}{\sqrt{\lambda_{\min}(X_k^\top X_k)}}.$$

From the definition of the support vectors, we also have that

$$\begin{aligned}\sum\nolimits_{k\in[r]} \overline{w}_k^2 &= \sum\nolimits_{k\in[r]} \Big(\sum\nolimits_{i\in\mathcal{I}^k} \mu_i \overline{w}_k(\overline{V}_k x_i + \overline{b}_k) - \sum\nolimits_{j\in\mathcal{J}^k} \mu_j \overline{w}_k(\overline{V}_k x_j + \overline{b}_k)\Big)\\ &= \sum\nolimits_{s\in\mathcal{I}\cup\mathcal{J}} \mu_s\end{aligned}$$

due to complementary slackness condition.

Our goal is to bound the Lipschitz constant of the estimate $\hat{f}(x) = \overline{w}^\top(\overline{V}x + \overline{b})_+$:

$$\begin{aligned}\sum\nolimits_{k\in[r]} |\overline{w}_k| \|\overline{V}_k\|_2 &\le \sum\nolimits_{k\in[r]} \overline{w}_k^2\\ &= \sum\nolimits_{s\in\mathcal{I}\cup\mathcal{J}} \mu_s\\ &\le \frac{n_{\text{node}}^{\text{min}} \sqrt{n_{\text{sup}}^{\text{max}}}}{\min_{k\in[r]} \sqrt{\lambda_{\min}(X_k^\top X_k)}}\end{aligned}$$

where $n_{\text{node}}^{\text{min}}$ denotes the minimum number of nodes that are activated by all of the support vectors:

$$n_{\text{node}}^{\text{min}} = \min_{K\subseteq[r]} \left\{|K| : \forall s \in \mathcal{I}^{\text{sup}} \cup \mathcal{J}^{\text{sup}}\ \exists k \in K \text{ such that } \overline{V}_k x_s + \overline{b}_k > 0\right\}.$$

■

4.9.6 Proof of Theorem 4.6

The first order stationarity condition implies that

$$\frac{\partial \ell}{\partial W_k} = 0 \quad \forall k \in [L],$$

which yields

$$\begin{aligned}0 &= \sum\nolimits_{s\in\mathcal{I}\cup\mathcal{J}} \overline{W}_{k+1}^\top \cdots \overline{W}_L^\top \left.\frac{\partial d(z, y_s)}{\partial z}\right|_{z=\overline{W}_L\cdots\overline{W}_1 x_s} x_s^\top \overline{W}_1^\top \cdots \overline{W}_{k-1}^\top + 2\mu_k \overline{W}_k\\ &= \overline{W}_{k+1}^\top \cdots \overline{W}_L^\top \left(\sum\nolimits_{s\in\mathcal{I}\cup\mathcal{J}} \left.\frac{\partial d(z, y_s)}{\partial z}\right|_{z=\overline{W}_L\cdots\overline{W}_1 x_s} x_s^\top\right) \overline{W}_1^\top \cdots \overline{W}_{k-1}^\top + 2\mu_k \overline{W}_k.\end{aligned}$$

Since the first term has rank 1, so does $\overline{W}_k$, for each $k \in [L]$. ■

4.9.7 Proof of Theorem 4.7

For brevity in notation, define

$$\begin{aligned}\tilde{x}_i &= x_i \quad \forall i \in \mathcal{I},\\ \tilde{x}_j &= -x_j \quad \forall j \in \mathcal{J},\end{aligned}$$

and $\mathcal{S} = \mathcal{I} \cup \mathcal{J}$. Then the cost function (4.6) could be written as

$$\ell(W_1, \dots, W_L) = \sum\nolimits_{s \in \mathcal{S}} \log\left(1 + e^{W_L \cdots W_1 \tilde{x}_s}\right).$$

Note that at least one of the weight matrices must diverge to ∞ in norm since the loss function does not attain its minimum at a finite point. In addition, we have

$$\begin{aligned}\frac{d\|W_k(t)\|_F^2}{dt} &= 2\left\langle W_k(t), \frac{dW_k(t)}{dt}\right\rangle\\ &= 2\left\langle W_k(t), -\frac{\partial \ell}{\partial W_k}\right\rangle\\ &= \frac{d\|W_{k'}(t)\|_F^2}{dt} \quad \forall k, k' \in [L],\end{aligned}$$

and consequently,

$$\|W_k(t)\|_F^2 - \|W_{k'}(t)\|_F^2 = \|W_k(0)\|_F^2 - \|W_{k'}(0)\|_F^2 \quad \forall t \geq 0,\ \forall k, k' \in [L].$$

Therefore, all of the weight matrices must diverge to ∞ in norm.

We can define the set of indices for the support vectors:

$$\mathcal{S}^{\text{sup}} = \left\{ s \in \mathcal{S} : \lim_{t \to \infty} \frac{\exp(W_L \cdots W_1 \tilde{x}_{s'})}{\exp(W_L \cdots W_1 \tilde{x}_s)} < \infty \ \forall s' \in \mathcal{S} \right\},$$

which is a nonempty set. For any $k \in [L]$, remember that $\overline{W}_k \in \mathbb{R}^{n_k \times n_{k-1}}$. Let $i_1, i_2 \in [n_k]$ and $j_1, j_2 \in [n_{k-1}]$ be such that $e_{i_1}^\top \overline{W}_k e_{j_1}$, $e_{i_2}^\top \overline{W}_k e_{j_1}$ and $e_{i_2} \overline{W}_k e_{j_2}$ are nonzero. Note that if such a tuple of (i_1, i_2, j_1) does not exist, then the matrix $\overline{W}_k$ has rank 1 already, and the following analysis is not needed. Given such a tuple of (i_1, i_2, j_1), we have

$$
\begin{aligned}
\frac{e_{i_1}^\top \overline{W}_k e_{j_1}}{e_{i_2}^\top \overline{W}_k e_{j_1}} &= \lim_{t\to\infty} \frac{e_{i_1}^\top W_k(t) e_{j_1}}{e_{i_2}^\top W_k(t) e_{j_1}} \\
&= \lim_{t\to\infty} \frac{e_{i_1}^\top \frac{dW_k}{dt} e_{j_1}}{e_{i_2}^\top \frac{dW_k}{dt} e_{j_1}} \\
&= \lim_{t\to\infty} \frac{\sum_{s\in\mathcal{S}} e_{i_1}^\top \left(W_{k+1}^\top \cdots W_L^\top \tilde{x}_s^\top W_1^\top \cdots W_{k-1}^\top\right) e_{j_1} e^{W_L\cdots W_1 \tilde{x}_s}}{\sum_{s\in\mathcal{S}} e_{i_2}^\top \left(W_{k+1}^\top \cdots W_L^\top \tilde{x}_s^\top W_1^\top \cdots W_{k-1}^\top\right) e_{j_1} e^{W_L\cdots W_1 \tilde{x}_s}} \\
&= \frac{e_{i_1}^\top \overline{W}_{k+1}^\top \cdots \overline{W}_L^\top}{e_{i_2}^\top \overline{W}_{k+1}^\top \cdots \overline{W}_L^\top} \lim_{t\to\infty} \frac{\sum_{s\in\mathcal{S}^{\text{sup}}} \left(\tilde{x}_s^\top \overline{W}_1^\top \cdots \overline{W}_{k-1}^\top\right) e_{j_1} e^{W_L\cdots W_1 \tilde{x}_s - W_L\cdots W_1 \tilde{x}_{s_{\text{sup}}}}}{\sum_{s\in\mathcal{S}^{\text{sup}}} \left(\tilde{x}_s^\top \overline{W}_1^\top \cdots \overline{W}_{k-1}^\top\right) e_{j_1} e^{W_L\cdots W_1 \tilde{x}_s - W_L\cdots W_1 \tilde{x}_{s_{\text{sup}}}}} \\
&= \frac{e_{i_1}^\top \overline{W}_{k+1}^\top \cdots \overline{W}_L^\top}{e_{i_2}^\top \overline{W}_{k+1}^\top \cdots \overline{W}_L^\top} \lim_{t\to\infty} \frac{\sum_{s\in\mathcal{S}^{\text{sup}}} \left(\tilde{x}_s^\top \overline{W}_1^\top \cdots \overline{W}_{k-1}^\top\right) e_{j_2} e^{W_L\cdots W_1 \tilde{x}_s - W_L\cdots W_1 \tilde{x}_{s_{\text{sup}}}}}{\sum_{s\in\mathcal{S}^{\text{sup}}} \left(\tilde{x}_s^\top \overline{W}_1^\top \cdots \overline{W}_{k-1}^\top\right) e_{j_2} e^{W_L\cdots W_1 \tilde{x}_s - W_L\cdots W_1 \tilde{x}_{s_{\text{sup}}}}} \\
&= \lim_{t\to\infty} \frac{e_{i_1}^\top \frac{dW_k}{dt} e_{j_2}}{e_{i_2}^\top \frac{dW_k}{dt} e_{j_2}} \\
&= \frac{e_{i_1}^\top \overline{W}_k e_{j_2}}{e_{i_2}^\top \overline{W}_k e_{j_2}}
\end{aligned}
$$

where $\tilde{x}_{s_{\text{sup}}}$ denotes any point in the set $\{x_s\}_{s\in\mathcal{S}^{\text{sup}}}$. Note that this equality shows that all columns of $\overline{W}_k$ are in the span of the same single vector in $\mathbb{R}^{n_k}$, which proves that $\overline{W}_k$ has rank 1. ■

4.9.8 Proof of Theorem 4.9

In order to achieve zero training error in one iteration of the stochastic gradient algorithm, it is sufficient to have

$$\min_{i'\in I}\langle x_{i'}, x_i + y_j\rangle > \max_{j'\in J}\langle -y_{j'}, x_i + y_j\rangle \quad \forall i \in I,\ \forall j \in J,$$

or equivalently,

$$\langle x_{i'} + y_{j'}, x_i + y_j\rangle > 0 \quad \forall i, i' \in I,\ \forall j, j' \in J. \tag{4.13}$$

By definition of the margin, there exists a vector $w_{\text{SVM}} \in \mathbb{R}^d$ with unit norm which satisfies

$$2\gamma = \min_{i\in I, j\in J}\langle x_i + y_j, w_{\text{SVM}}\rangle.$$

Note that w_{SVM} is orthogonal to the decision boundary given by the SVM. Then we can write every $x_i + y_j$ as

$$x_i + y_j = 2\gamma w_{\text{SVM}} + \delta_i^x + \delta_j^y,$$

where $\delta_i^x, \delta_j^y \in \mathbb{R}^d$ and $\|\delta_i^x\| \le R_x$ and $\|\delta_j^y\| \le R_y$. Then, condition (4.13) is satisfied if

$$\langle 2\gamma w_{\text{SVM}} + \delta_i^x + \delta_j^y, 2\gamma w_{\text{SVM}} + \delta_{i'}^x + \delta_{j'}^y \rangle > 0 \quad \forall i, i' \in I,\ \forall j, j' \in J,$$

or equivalently if

$$4\gamma^2 + 2\gamma\langle w_{\text{SVM}}, \delta_i^x + \delta_j^y + \delta_{i'}^x + \delta_{j'}^y \rangle + \langle \delta_i^x + \delta_j^y, \delta_{i'}^x + \delta_{j'}^y \rangle > 0 \quad \forall i, i' \in I,\ \forall j, j' \in J. \tag{4.14}$$

If we choose $\gamma > \frac{5}{2}\max(R_x, R_y)$, we have

$$4\gamma^2 - 2\gamma(2R_x + 2R_y) - (R_x + R_y)^2 > 0,$$

which guarantees (4.14) and completes the proof. ■

Chapter 5

Learning Linear Dynamical Systems

5.1 Introduction

Systems with memories that evolve over time require the use of a dynamical model for their representation. This model describes how the memory, or the state, of this system changes over time, how its state is affected by inputs to the system, and how it generates observable outputs. System identification corresponds to the task of learning the unknown parameters of this dynamical model from the known inputs and the observed outputs of the system.

Identification of dynamical systems from time-series data is an important problem for various applications, such as model prediction in reinforcement learning (Lambert et al., 2019; Zhang et al., 2016), analysis of medical health records (Rubanova et al., 2019) and prediction with financial time-series data (Tsay, 2014; Ganeshapillai et al., 2013). However, the identification problems that arise in these applications pose some theoretical challenges:

1. Unless the state of the system is observed with a known noiseless mapping, the identification of the system model is coupled with the state estimation. Consequently, the system identification task is in general a nonconvex problem (Hardt et al., 2018). To circumvent this nonconvexity, the initial state can be assumed to be zero in control settings, and a known input can be used to drive the state of the system (Sastry, 1984; Sastry and Bodson, 1989). However, in medical and financial settings, the initial state of the system is typically not known a priori, and the deviations of the initial state from a nominal value cannot be neglected. Therefore, a joint and nonconvex optimization procedure is unavoidable in these settings to estimate the initial state of the system along with the unknown model parameters (Frigola et al., 2014; Duncker et al., 2019).

2. For control of a dynamical system in a reinforcement learning task, it is most critical that the unstable[1] modes of the system be discovered and stabilized properly. Similarly, financial data

[1]The term stability refers to bounded-input bounded-output stability. For continuous-time linear time-invariant systems, this corresponds to the condition where the eigenvalues of the state transition matrix have strictly negative real parts.

and medical health records usually exhibit sudden changes in their pattern, which call for potentially unstable dynamics in their representation and estimation. However, the primary tools for nonconvex optimization, namely, the gradient methods, fail to converge and find an accurate model representation for unstable systems (Hardt et al., 2018).

3. Especially in medical and financial data sets, the data are sampled irregularly; that is, the observations are not periodically sampled. The common heuristic approach to handle this situation is imputing the absent observations by interpolating the observed values of the output (Che et al., 2018). This approach, however, might fail to capture the correct dynamics of the underlying system. An alternative is to use a model that can take account for the evolution of the state of the system during unobserved intervals without requiring periodic observations (Chen et al., 2018).

In this chapter, we use the gradient descent algorithm to identify the unknown parameters of a linear dynamical system from its observed outputs. We look into the dynamics of this algorithm and try to pinpoint what causes the inability of the gradient methods to converge when they are used to identify an unstable dynamical system. Similar to the work of Chen et al. (2018), our analysis uses a continuous-time model so that it directly applies to irregularly sampled data sets with no need for imputation.

By analyzing the dynamics of the gradient descent algorithm during identification of a linear dynamical system, we achieve the following.

1. We obtain an upper bound on the learning rate of the gradient descent algorithm so that it can converge while learning a dynamical system with the squared-error loss. This upper bound explicitly depends on the eigenvalue of the system with the largest real part, and it shows that identifying a system becomes harder as the system becomes unstable. Furthermore, the upper bound on the learning rate shows that the samples taken at different times affect the convergence of the gradient descent algorithm in substantially different degrees.

2. To enable the convergence of the gradient descent algorithm even when learning unstable systems, we introduce a new loss function which balances the influence of the samples taken at different times on the convergence of the algorithm. Then we demonstrate the effectiveness of this loss function while estimating linear dynamical systems.

The results in this chapter have appeared in (Nar et al., 2020). Note that the primary question our work addresses is about the use of the gradient descent algorithm while learning a dynamical system: can this algorithm converge at all while learning the parameters of a dynamical system model? This is a different problem than whether a specific algorithm, or a specific model can learn the dynamical system of interest more accurately than the state-of-the-art.

5.1.1 Related works

Hardt et al. (2018) studied the convergence of the gradient descent algorithm while learning linear dynamical systems. They demonstrated the failure of this algorithm to learn even stable systems, and proposed a projected gradient method that fixed the issue for linear stable systems. Learning an unstable system, however, was considered to be infeasible. In contrast, we retain the standard gradient descent algorithm in this work, and we introduce a new loss function that allows learning even unstable systems with no necessity for projection.

If the state of a linear system is directly accessed, that is, if the output of the system is equal to the state of the system possibly with some additive noise, learning the system parameters can be formulated as an ordinary least squares problem. Alaeddini et al. (2018) and Sarkar and Rakhlin (2019) make this assumption and arrive at a convex optimization problem. By doing so, they avoid the use of gradient descent algorithm, and therefore, they do not suffer from the issues pointed out by Hardt et al. (2018). However, as mentioned earlier, the assumption of having an access to the true internal state is unrealistic in many application domains, such as, health and finance.

Using variational inference is a common approach to estimate the initial state jointly with the dynamical model parameters in a Bayesian setting (Frigola et al., 2014; Archer et al., 2015; Krishnan et al., 2017; Eleftheriadis et al., 2017; Duncker et al., 2019; Gregor et al., 2019). In this approach, a separate model is employed to estimate the initial state from the whole observed trajectory. One of the models that we will consider in this work is a simpler, deterministic counterpart of this approach. We show that convergence issues of the gradient descent algorithm are also valid for this deterministic counterpart of variational inference.

Neural ordinary differential equations (Chen et al., 2018; Rubanova et al., 2019) use a neural network to represent a continuous-time dynamical system. Since these models are also trained with the gradient descent algorithm, they also suffer from the stability issues of the gradient descent algorithm while learning the parameters of a dynamical model. Indeed, the training data of all the examples outlined in these works involve trajectories that converge to either a stable equilibrium or a stable limit cycle of the system.

5.2 Problem Formulation

For each $k \in \mathcal{K} = \{1, \ldots, K\}$, let $z_k : [0, \infty) \mapsto \mathbb{R}^n$ denote a continuous-time process representing the state of a linear time-invariant dynamical system:

$$\frac{dz_k(t)}{dt} = Az_k(t) \quad \forall t \geq 0, \; \forall k \in \mathcal{K},$$

where $A \in \mathbb{R}^{n \times n}$ denotes the state transition dynamics of the system. Then the evolution of the process is described by $z_k(t) = e^{At} z_k(0)$ for all $t \geq 0$ for each $k \in \mathcal{K}$ (Callier and Desoer, 1991). Let $\{x_k(t)\}_{t \in \mathcal{T}_k}$ be the set of samples obtained from z_k at time instants $t \in \mathcal{T}_k$ via an observation matrix $C \in \mathbb{R}^{m \times n}$:

$$x_k(t) = Cz_k(t) = Ce^{At} z_k(0) \quad \forall t \in \mathcal{T}_k, \; \forall k \in \mathcal{K}.$$

Define the initial state of the trajectory of z_k as $s_k \in \mathbb{R}^n$; that is, let $s_k = z_k(0)$ for all $k \in \mathcal{K}$. We will look for a linear dynamical system model that fits all the trajectories, and we will use the gradient descent algorithm to reveal the difficulty of its convergence. In particular, our goal is to study whether the gradient descent algorithm is able to converge to a solution while solving the problem

$$\underset{A,C}{\text{minimize}} \quad \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} \ell\left(x_k(t), Ce^{At}s_k\right) \tag{5.1a}$$

where ℓ is a differentiable loss function. We consider two choices for ℓ in the following sections: the squared-error loss as it is used both in classical works (Åström and Eykhoff, 1971) and in recent works (Hardt et al., 2018), and the time-weighted logarithmic loss introduced in Section 5.4.

The set of initial states $\{s_k\}_{k\in\mathcal{K}}$ is left arbitrary in the statement of (5.1); we consider three possible cases for these initial states, and our analysis in the following sections applies to all of these three cases.

1. Each s_k is known or has a fixed value. In other words, the set $\{s_k\}_{k\in\mathcal{K}}$ is not updated by the gradient descent algorithm.

2. Each s_k is also a variable, and the gradient descent algorithm optimizes over $\{s_k\}_{k\in\mathcal{K}}$ as well:

$$\underset{A,C,\{s_k\}_{k\in\mathcal{K}}}{\text{minimize}} \quad \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} \ell\left(x_k(t), Ce^{At}s_k\right) \tag{5.2}$$

3. Each s_k is output of a state estimator:

$$s_k = g_\phi\left(\{t, x_k(t)\}_{t\in\mathcal{T}_k}\right) \quad \forall k\in\mathcal{K},$$

where ϕ is the parameters of this estimator, and the gradient descent algorithm solves the problem

$$\underset{A,C,\phi}{\text{minimize}} \quad \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} \ell\left(x_k(t), Ce^{At}s_k\right) + \mathcal{L}(\phi) \tag{5.3a}$$

$$\text{subject to} \quad s_k = g_\phi\left(\{t, x_k(t)\}_{t\in\mathcal{T}_k}\right) \quad \forall k\in\mathcal{K}, \tag{5.3b}$$

where $\mathcal{L}$ is an additional loss term associated with the estimation of the initial state, and it satisfies

$$\frac{\partial\mathcal{L}}{\partial A} = 0,\ \frac{\partial\mathcal{L}}{\partial C} = 0.$$

This case can be considered as the deterministic counterpart of the framework used in variational inference of state space models (Jordan et al., 1999; Archer et al., 2015). This comparison is discussed further in Section 5.6.

In the following sections, we will demonstrate the analysis and state the theorems for problem (5.2) in the second case. The statements are identically valid for the other two cases, as explained in Appendix 5.8.4.

5.3 Learning with Squared-Error Loss

In this section, we consider problem (5.2) with the squared-error loss:

$$\underset{A,C,\{s_k\}_{k\in\mathcal{K}}}{\text{minimize}} \quad \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} \|x_k(t) - Ce^{At}s_k\|_2^2$$

If we use the gradient descent algorithm to solve problem (5.2), the learning rate of the algorithm needs to be sufficiently small for the algorithm to converge (Bertsekas, 1999). The next theorem gives an upper bound on the learning rate as a necessary condition for the convergence of the algorithm.

Theorem 5.1. *Let $\{z_k\}_{k\in\mathcal{K}}$ be a set of trajectories, and let s_k denote the initial state for trajectory z_k for each $k \in \mathcal{K}$. Define the set of sampling instants of z_k as $\mathcal{T}_k$, and denote the samples taken from this trajectory by $\{x_k(t)\}_{t\in\mathcal{T}_k}$. Assume that the gradient descent algorithm is used to solve the problem*

$$\min_{A,C,\{s_k\}_{k\in\mathcal{K}}} \quad \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} \left\|x_k(t) - Ce^{At}s_k\right\|_2^2. \tag{5.4}$$

Then for almost every initialization, the learning rate of the gradient descent algorithm, δ, must satisfy

$$\delta \le \frac{2}{\lambda_{\min}\left(\rho^2 \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} t^2 e^{2\mathrm{Re}(\Lambda)t}\hat{s}_k\hat{s}_k^\top\right)}$$

so that the algorithm can converge to the solution $\left(\hat{A}, \hat{C}, \{\hat{s}_k\}_{k\in\mathcal{K}}\right)$ achieving zero training error, where $\lambda_{\min}(\cdot)$ denotes the minimum eigenvalue of its argument, Λ is the eigenvalue of $\hat{A}$ with the largest real part, $\rho^2 = \max_{u\in\mathcal{U}} \|\hat{C}u\|_2^2$, and $\mathcal{U}$ is the set of eigenvectors of $\hat{A}$ corresponding to Λ.

Proof. See Appendix 5.8.1. ∎

Note that the eigenvalues of a linear dynamical system have a particular meaning in control theory: they describe the stability of the system (Callier and Desoer, 1991). If any eigenvalue of $\hat{A}$ has a real part that is strictly positive, then the state of the system will grow unboundedly large from almost all initial points; and the system is called unstable in this case. If, on the other hand, all eigenvalues of $\hat{A}$ has a negative real part, then the state of the system will converge to a fixed point from all initial points, and the system will be stable.

The condition about reaching zero training error might be somewhat restrictive, but the main purpose of Theorem 5.1 is not to prescribe a learning rate for all possible cases; it is to reveal that the samples taken at different times affect the convergence of the algorithm very differently. Indeed, Theorem 5.1 shows that if the gradient descent algorithm is used to learn an unstable system, samples taken at later times impose a bound on the required learning rate exponentially more strict, which renders learning an unstable dynamical system infeasible.

Note that if the set of initial states $\{\hat{s}_k\}_{k\in\mathcal{K}}$ does not span the whole state space, then the bound given in Theorem 5.1 will be void. This suggests that it will be easier to train a dynamical model if the initial states of the trajectories given in the training data do not have a large variance. However, this does not mean the learned model will be accurate. Since there is no information available about how the system evolves for the initial states in the nullspace of $\sum_{k\in\mathcal{K}} \hat{s}_k\hat{s}_k^\top$, the model learned will fail to predict the behavior of the system for the initial states with a nonzero component in this unlearned subspace as well.

The appearance of ρ in Theorem 5.1 reflects the notion of observability (Callier and Desoer, 1991). Based on the relationship between the matrices $\hat{A}$ and $\hat{C}$, it may not be possible to observe certain eigenvalues, or modes, of the learned system in its output; these modes are called unobservable modes. As these modes do not appear in the output of the learned system, they cannot affect the gradient descent algorithm.

Remark 5.1. The analysis for Theorem 5.1 shows that, for the Hessian H of the loss function (5.4) at $(\hat{A}, \hat{C})$, the ratio of the largest eigenvalue to the smallest eigenvalue of H satisfies

$$\frac{\lambda_{\max}(H)}{\lambda_{\min}(H)} \geq \frac{\lambda_{\min}\left(\rho_1^2 \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} t^2 e^{2Re(\lambda_1)t}\hat{s}_k\hat{s}_k^\top\right)}{\lambda_{\max}\left(\rho_2^2 \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} t^2 e^{2Re(\lambda_2)t}\hat{s}_k\hat{s}_k^\top\right)}$$

for any pair of eigenvalues (λ_1, λ_2) of $\hat{A}$, where $\rho_1 = \|Cu_1\|_2$, $\rho_2 = \|Cu_2\|_2$, and u_1, u_2 are the right eigenvectors of $\hat{A}$ corresponding to λ_1, λ_2, respectively. This implies that, if the loss function can be represented well by its second order approximation around $(\hat{A}, \hat{C})$, local convergence rate for estimating the eigenvalue λ_2 will require

$$O\left(\left[\log\left(\left(1-\beta\frac{\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} t^2 e^{2Re(\lambda_2)t}}{\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} t^2 e^{2Re(\lambda_1)t}}\right)^{-1}\right)\right]^{-1}\right)$$

iterations of the gradient descent algorithm, where β is some constant depending on ρ_1, ρ_2 and $\sum_{k\in\mathcal{K}} \hat{s}_k\hat{s}_k^\top$. This shows that learning the stable modes of a system can become infeasible when the system is unstable. See Appendix 5.8.3 for more details.

The necessary condition given in Theorem 5.1 implies that the convergence of the algorithm gives us information about the rightmost eigenvalue of the dynamical system that is being estimated. This is stated in Corollary 5.1.

Corollary 5.1. *Assume that the observation matrix $C = I$, the gradient descent algorithm is used to solve the problem (5.4) and the algorithm has converged from a random*[2] *initialization to the solution $\left(\hat{A}, \{\hat{s}_k\}_{k\in\mathcal{K}}\right)$ achieving zero training error. Then the eigenvalue of $\hat{A}$ with the largest real part, Λ, almost surely satisfies*

$$\mathrm{Re}(\Lambda) \leq \inf_{\tau>0} \frac{1}{2\tau}\log\left[\frac{1}{\delta\tau^2}\frac{2}{\lambda_{\min}\left(\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k}\hat{s}_k\hat{s}_k^\top \mathbf{1}_{\{t\geq\tau\}}\right)}\right] \qquad \textit{if } \mathrm{Re}(\Lambda) > 0,$$

$$\mathrm{Re}(\Lambda) \leq \inf_{\tau_2>\tau_1>0} \frac{1}{2\tau_2}\log\left[\frac{1}{\delta\tau_1^2}\frac{2}{\lambda_{\min}\left(\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k}\hat{s}_k\hat{s}_k^\top \mathbf{1}_{\{\tau_1\leq t\leq\tau_2\}}\right)}\right] \qquad \textit{if } \mathrm{Re}(\Lambda) < 0.$$

[2]The random distribution is assumed to assign zero probability to every set with Lebesgue measure zero.

5.4 Learning with Time-Weighted Logarithmic Loss

Theorem 5.1 shows that when the gradient descent algorithm is used to learn the parameters of an unstable dynamical system, the effect of the samples taken at later times are exponentially more weighted around a global minimum. It is important to note that this is the case for the choice of squared-error loss as the training loss function. In this section, we introduce a new loss function in order to balance the effects of all samples on the dynamics of the algorithm. This new loss function greatly relaxes the necessary condition given in Theorem 1, and it enables training even unstable linear systems with the gradient descent algorithm.

For any $\epsilon > 0$, define $F_\epsilon : \mathbb{R} \to \mathbb{R}$ as

$$F_\epsilon(\xi) = \begin{cases} \log(\epsilon + \xi) - \log(\epsilon) & \xi \geq 0, \\ -\log(\epsilon - \xi) + \log(\epsilon) & \xi < 0. \end{cases} \tag{5.5}$$

Given two trajectories $\{x(t)\}_{t\in\mathcal{T}}$ and $\{y(t)\}_{t\in\mathcal{T}}$ in $\mathbb{R}^n$, consider the loss function defined as

$$\ell(x, y) = \sum_{t\in\mathcal{T}} \sum_{j=1}^{n} \frac{1}{t^2} \left(F_\epsilon(e_j^\top x(t)) - F_\epsilon(e_j^\top y(t))\right)^2,$$

where e_j denotes the j-th standard basis vector with a 1 in its j-th coordinate and 0 in all other coordinates. Note that $\ell(x, y)$ is zero if and only if $x(t) = y(t)$ for all $t \in \mathcal{T}$; and it is strictly positive otherwise. Similar to Section 5.3, we will analyze this loss functions for learning linear dynamical systems.

Theorem 5.2. *Let $\{z_k\}_{k\in\mathcal{K}}$ be a set of trajectories, and let s_k denote the initial state for trajectory z_k for each $k \in \mathcal{K}$. Define the set of sampling instants of z_k as $\mathcal{T}_k$, and denote the samples taken from this trajectory by $\{x_k(t)\}_{t\in\mathcal{T}_k}$. Assume that the gradient descent algorithm is used to solve*

$$\min_{A,C,\{s_k\}_{k\in\mathcal{K}}} \sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}_k} \sum_{j=1}^{n} \frac{1}{t^2} \left(F_\epsilon(e_j^\top x_k(t)) - F_\epsilon(e_j^\top C e^{At} s_k)\right)^2, \tag{5.6}$$

where F_ϵ is as defined in (5.5). Then for almost every initialization, the learning rate δ of the gradient descent algorithm must satisfy

$$\delta \leq \frac{2}{\lambda_{\min}\left(\sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}_k} \frac{\rho^2 e^{2Re(\Lambda)t}}{(\|\hat{C}e^{\hat{A}t}\hat{s}_k\|_\infty + \epsilon)^2} \hat{s}_k \hat{s}_k^\top\right)}$$

so that the algorithm can converge to the solution $(\hat{A}, \hat{C}, \{\hat{s}_k\}_{k\in\mathcal{K}})$ achieving zero training error, where Λ is the eigenvalue of $\hat{A}$ with the largest real part, $\rho^2 = \max_{u\in\mathcal{U}} \|\hat{C}u\|_2^2$, and $\mathcal{U}$ is the set of right-eigenvectors of $\hat{A}$ corresponding to its eigenvalue Λ.

Proof. See Appendix 5.8.2. ■

The necessary conditions on the step size given in Theorem 5.2 and in Theorem 5.1 are obtained by following the identical analysis procedure. Theorem 5.2 shows that the loss function (5.6) substantially relaxes the necessary condition given in Theorem 5.1, and it balances the weights of all the sampling instants on the dynamics of the gradient descent algorithm. In other words, it makes it easier for the gradient descent algorithm to converge to the global minima. This is demonstrated in the next section.

5.5 Experiments

To check if the time-weighted logarithmic loss function introduced in Theorem 5.2 allows learning linear dynamical systems with the gradient descent algorithm, we generated a set of output trajectories from randomly generated linear systems and trained a linear model with this data set by using the logarithmic loss function. We also trained the model with the same data set by using the mean-squared-error loss to compare the two estimates.

For the experiments, we considered the discretized version of the dynamical systems. In other words, we used

$$z_k(t) = A^t z_k(0) \quad \forall t \in \mathbb{N},\ \forall k \in \mathcal{K}.$$

Note that with this discrete-time representation, the stability of the system is described based on the position of the eigenvalues relative to the unit circle. The system is stable if all of its eigenvalues are inside the unit circle.

We randomly generated $A \in \mathbb{R}^{n\times n}$ and $C \in \mathbb{R}^n$ to produce a set of observation sequences. In particular, we generated A as $A = I + \Delta A$, where ΔA is a matrix whose entries are independent and uniformly distributed between $[-0.5, 0.5]$. The elements of C were drawn from independent standard normal distributions. We obtained 50 trajectories from the generated system by providing different initial states, and each trajectory consisted of 50 observations.

For training a linear model on this data set, we used the stochastic gradient method with momentum. Both for the mean-squared-error loss and for the time-weighted logarithmic loss, the gradients were normalized to unit norm if their ℓ_2 norm exceeded 1. Figure 5.1 shows a typical plot for the training error of an unstable system for each of these loss functions. We observe that the gradient descent algorithm is not able to decrease the mean-squared error loss, whereas the time-weighted logarithmic loss function is diminished easily.

To check if this decrease in the loss function corresponds to an effective learning of the actual model, we computed the eigenvalues of the estimated system throughout training and compared them with the eigenvalues of the actual system. Figure 5.2 demonstrates an example of how the estimates for the eigenvalues evolve during training. The state space of the system in Figure 5.2 is three dimensional, and the system is unstable as one of its eigenvalues is outside of the unit circle. When the mean-squared-error loss is used, only the unstable mode of the system is estimated correctly. In contrast, the time-weighted logarithmic loss function is able to discover all three modes of the system.

Figure 5.3 and Figure 5.4 demonstrate the comparison of the estimated eigenvalues for a different initialization and for a system with a four-dimensional state space, respectively. Note that we were

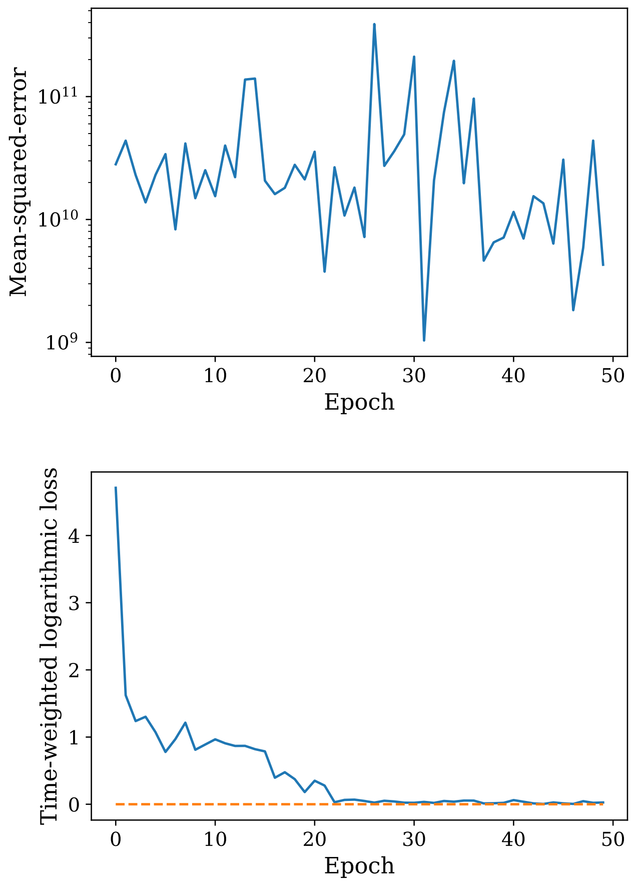
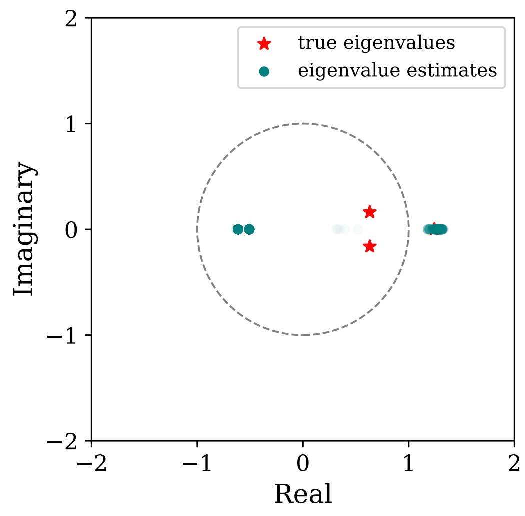


Figure 5.1: Typical plots of training error when mean-squared-error is used [top] and when time-weighted logarithmic loss function is used [bottom].

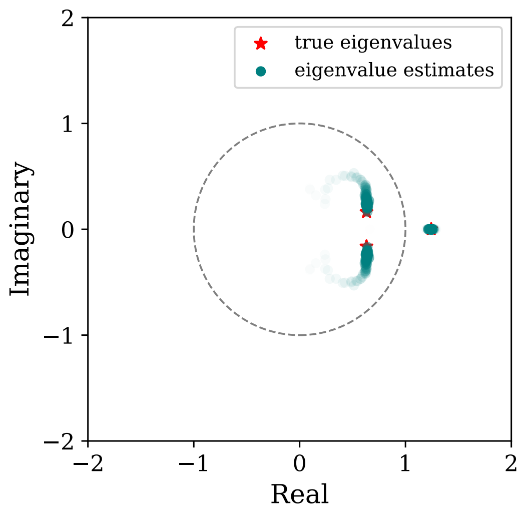
not able to enable the gradient descent algorithm to learn any of the eigenvalues correctly when the training loss is mean-squared error despite the fact that we used various learning rates for these experiments.

5.6 Discussion

Variational inference. Variational inference is a Bayesian approach to handle the unknown parameters and the unobserved states of a dynamical system simultaneously (Jordan et al., 1999; Archer et al., 2015). For variational inference, the system is described by a generative model: $p_\theta(x, z)$, where $x = \{x(t)\}_{t\in\mathcal{T}}$ and $z = \{z(t)\}_{t\in\mathcal{T}}$ are the sequence of observations and hidden states of the system, and θ is the parameters of the model. Given the observations, the posterior is approximated by another model: $g_\phi(\cdot|x)$. Then, the objective function to be minimized is described as (Archer



(a) Eigenvalues with mean-squared-error



(b) Eigenvalues with logarithmic loss

Figure 5.2: A linear system with three-dimensional state space is trained with mean-squared-error loss [left] and time-weighted logarithmic loss [right]. The red stars show the eigenvalues of the real system, whereas the green dots show the eigenvalues of the estimated system. Earlier estimates of the eigenvalues are depicted with faded colors. Mean-squared-error loss is able to find only the unstable mode, whereas the logarithmic loss function discovers all three modes correctly.

et al., 2015)

$$-\mathcal{H}(g_\phi(\mathbf{z}|x)) - \mathbb{E}_{g_\phi(\mathbf{z}|x)}[\log(p_\theta(x,\mathbf{z})], \tag{5.7}$$

where $\mathcal{H}$ is the entropy of its argument. Assume the stochasticity of the initial state and the state transitions is removed, and each observation $x(t)$ is obtained through an observation mapping with an additive Gaussian noise:

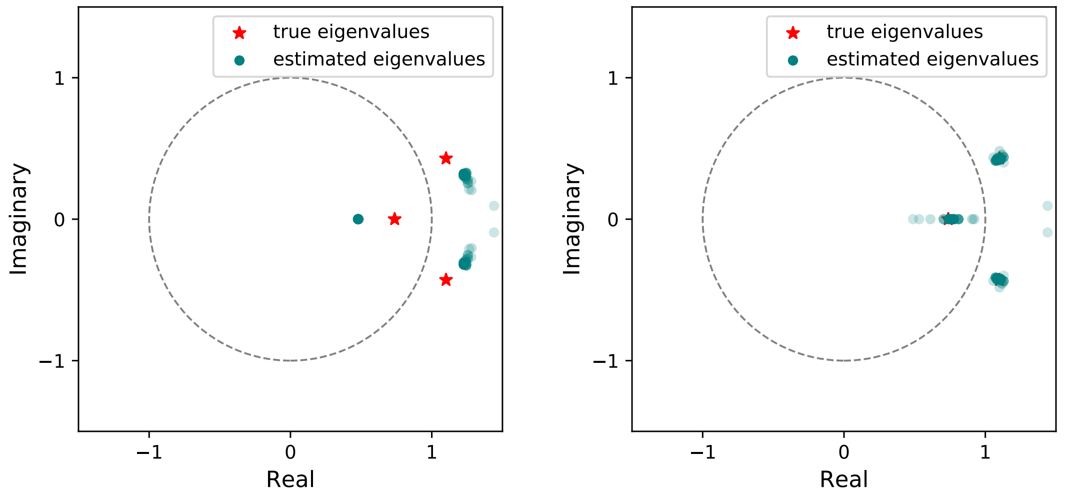
$$x(t) = c(z(t)) + \xi_t,$$

where $\{\xi_t\}_{t\in\mathcal{T}}$ is an independent and identically distributed sequence. Then the minimization of the loss function (5.7) reduces to the problem

$$\begin{array}{ll} \underset{\theta,\phi}{\text{minimize}} & \sum_{t\in\mathcal{T}} \|x(t) - c(z(t))\|_2^2 \\ \text{subject to} & z(0) = \underset{\tilde{z}}{\operatorname{argmax}}\, g_\phi(\tilde{z}|x), \end{array}$$

and the system identification problem becomes equivalent to problem (5.3). This is the reason why we referred to (5.3) as the deterministic counterpart of the variational inference formulation.

Random initial states. In our analyses, we treated the initial states as unknown but deterministic values that could be learned during training. With this deterministic viewpoint, Theorem 5.1 and Theorem 5.2 assumed that the observations could be matched to the latent state of the system perfectly and the training loss function could be made identically zero. It is not possible in general



(a) Eigenvalues with mean-squared-error (b) Eigenvalues with logarithmic loss

Figure 5.3: A linear system with three-dimensional state space is trained with mean-squared-error loss [left] and time-weighted logarithmic loss [right]. The red stars show the eigenvalues of the real system, whereas the green dots show the eigenvalues of the estimated system. Earlier estimates of the eigenvalues are depicted with faded colors.

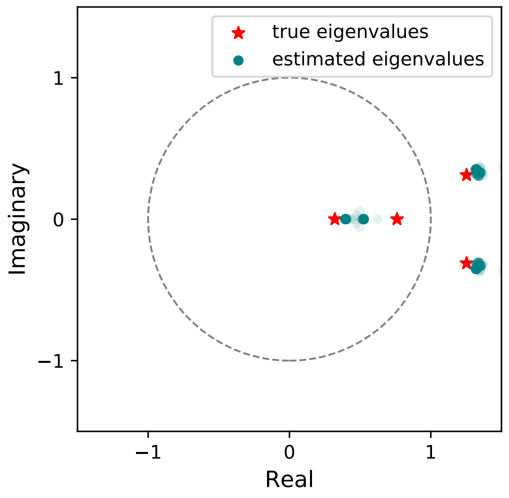
to satisfy this requirement with an expected loss over a set of random initial states. Therefore, Theorem 5.1 and Theorem 5.2 do not apply to the formulations with random initial states verbatim.

Convergence of policy gradient. Even though the focus of this work has been on system identification, the gradient descent algorithm will exhibit similar convergence problems when maximizing an objective over a time horizon while altering the dynamics of a dynamical system. Note that policy gradient methods in reinforcement learning (Sutton and Barto, 2018) fall into this category. This is why our analysis in this work can potentially be used for studying and improving the stability of policy gradient methods.

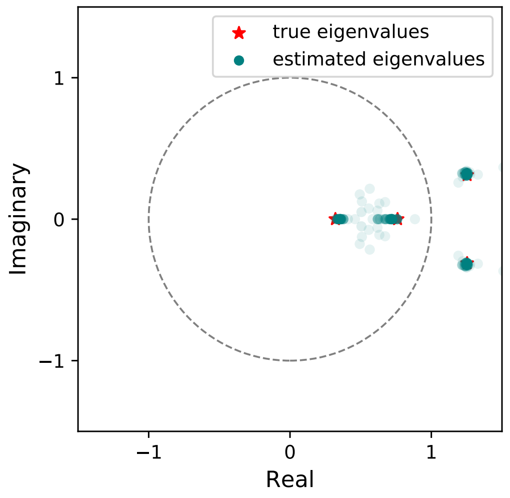
5.7 Conclusion

To understand the hardness of learning dynamical systems from observed trajectories, we analyzed the dynamics of the gradient descent algorithm while training the parameters of a dynamical model, and we observed that samples taken at different times affect the dynamics of the algorithm in substantially different degrees. To balance the effect of samples taken at different times, we introduced the time-weighted logarithmic loss function and demonstrated its effectiveness.

In this chapter, we focused on learning linear dynamical systems. Whether a similar loss function improves training of nonlinear models is an important direction for future research. In addition, we considered a deterministic framework for our problem formulation with a dynamical



(a) Eigenvalues with mean-squared-error



(b) Eigenvalues with logarithmic loss

Figure 5.4: A linear system with four-dimensional state space is trained with mean-squared-error loss [left] and time-weighted logarithmic loss [right]. The red stars show the eigenvalues of the real system, whereas the green dots show the eigenvalues of the estimated system. Earlier estimates of the eigenvalues are depicted with faded colors.

system. An interesting question is whether allowing randomness in the state of the system or the state transitions could trade off the accuracy of the estimated model for the efficiency of the training procedure.

5.8 Proofs and Further Remarks

In this section, we provide the proofs for the theorems and corollaries of this section and elaborate on some of the remarks.

5.8.1 Proof of Theorem 5.1

To begin with, assume that C is a fixed matrix, and consider only one trajectory z with only one sample taken at time t. Then the loss function to be minimized is

$$\ell(A, s) = \frac{1}{2}\|x - Ce^{At}s\|_2^2,$$

where s denotes the initial state of the trajectory. The update rule for the gradient descent algorithm gives

$$A \leftarrow A - \frac{\delta}{2}\frac{\partial}{\partial A}\langle Ce^{At}s - x, Ce^{At}s - x\rangle \tag{5.8a}$$

$$s \leftarrow s - \frac{\delta}{2}\frac{\partial}{\partial s}\langle Ce^{At}s - x, Ce^{At}s - x\rangle \tag{5.8b}$$

This update rule creates a nonlinear dynamical system where the state of the system is the parameters (A, s).

A dynamical system can converge to its equilibrium only if that equilibrium is stable in the sense of Lyapunov. A standard tool to analyze the stability for nonlinear systems is given by Lyapunov's direct method: an equilibrium of a nonlinear system can be stable only if the linearization of the system around that equilibrium has no unstable mode (Khalil, 1996). If, on the other hand, the linearized model has an eigenvalue larger than 1 in magnitude, then the nonlinear system is definitely unstable — which rules out the possibility of convergence to this equilibrium from its neighbors, except for a set on a low-dimensional manifold, which has Lebesgue measure zero. This shows that the system (5.8) can converge to an equilibrium only if all eigenvalues of the linearized model around that equilibrium are less than 1 in magnitude.

We can write the linearization of (5.8) around an equilibrium $(\hat{A}, \hat{s})$ as

$$\tilde{A} \leftarrow \tilde{A} - \delta f_1(\tilde{A}) - \delta f_2(\tilde{s}),$$
$$\tilde{s} \leftarrow \tilde{s} - \delta f_3(\tilde{A}) - \delta f_4(\tilde{s}),$$

where

- f_1 is the Jacobian with respect to A of the gradient with respect to A of the loss function ℓ at $(\hat{A}, \hat{s})$,
- f_2 is the Jacobian with respect to s of the gradient with respect to A of the loss function ℓ at $(\hat{A}, \hat{s})$,

and f_3 and f_4 are defined similarly. Note that f_2 and f_3 are the Jacobians of the gradients of the same function with respect to the same parameters in different orders; therefore, they are hermitian of each other:

$$\langle \tilde{A}, f_2(\tilde{s})\rangle = \langle f_3(\tilde{A}), \tilde{s}\rangle \quad \forall \tilde{A}, \forall \tilde{s}.$$

This shows that the linearized model can be associated with a symmetric matrix; and consequently, all of its eigenvalues are real-valued, and its eigenvalues can be less than 1 only if all of its diagonal blocks have eigenvalues less than 1. In other words, a necessary condition for the solution $(\hat{A}, \hat{s})$ to be stable is that the mappings

$$\tilde{A} \leftarrow \tilde{A} - \delta f_1(\tilde{A}) \tag{5.9}$$

$$\tilde{s} \leftarrow \tilde{s} - \delta f_4(\tilde{s}) \tag{5.10}$$

have eigenvalues less than 1 in magnitude, or equivalently, the functions f_1 and f_4 have eigenvalues less than $2/\delta$. Note that this conclusion would be identical if C was also updated via the gradient descent algorithm. In particular, we would need the eigenvalue of the mapping f_1 to be less than 1 in magnitude around the equilibrium $(\hat{A}, \hat{C}, \{\hat{s}_k\}_{k\in\mathcal{K}})$.

Finding a lower bound for the largest eigenvalue of the mapping f_1 will be easier with the following lemma.

Lemma 5.1. *Let $f_i : \mathbb{R}^n \to \mathbb{R}$ be a twice-differentiable function for all $i \in \mathcal{I}$, and define*

$$F(x) = \frac{1}{2} \sum_{i\in\mathcal{I}} f_i^2(x).$$

If $F(x_0) = 0$, then the Hessian of F at x_0 satisfies

$$\nabla^2 F(x_0) = \sum_{i\in\mathcal{I}} \nabla f_i(x_0) \nabla f_i(x_0)^\top.$$

Proof. We can write the gradient and the Hessian of F, respectively, as

$$\begin{aligned}\nabla F(x_0) &= \sum_{i\in\mathcal{I}} (\nabla f_i(x_0)) f_i(x_0),\\ \nabla^2 F(x_0) &= \sum_{i\in\mathcal{I}} \nabla f_i(x_0)\nabla f_i(x_0)^\top + f_i(x_0)\cdot \nabla^2 f_i(x_0).\end{aligned}$$

Note that $F(x_0) = 0$ implies that $f_i(x_0) = 0$ for all $i \in \mathcal{I}$. Then we have

$$\nabla^2 F(x_0) = \sum_{i\in\mathcal{I}} \nabla f_i(x_0) \nabla f_i(x_0)^\top.$$

■

Remember that $f_1(A)$ is the Jacobian with respect A of the gradient with respect to A of the loss function

$$\ell(A, C, s) = \frac{1}{2} \left\langle Ce^{At}s - x, Ce^{At}s - x \right\rangle.$$

Given $A \in \mathbb{R}^{n\times n}$, we can write

$$\ell(A, C, s) = \frac{1}{2} \sum_{j=1}^{n} \left(e_j^\top Ce^{At}s - e_j^\top x\right)^2,$$

where e_j is the j-th standard basis vector with a 1 in its j-th coordinate and 0 in all other coordinates. Then, by using Lemma 5.1, the largest eigenvalue of the mapping f_1 can be lower bounded by

$$\max_{Y:\|Y\|_F=1} \sum_{j=1}^{n} \left|\left\langle Y, \nabla_A (e_j^\top Ce^{At}s - e_j^\top x)\right\rangle\right|^2. \tag{5.11}$$

To find the gradient, we can expand the matrix exponential:

$$\nabla_A \left(e_j^\top C \sum_{k=0}^{\infty} \frac{t^k}{k!} A^k s \right) = \sum_{k=1}^{\infty} \sum_{r=0}^{k-1} \frac{t^k}{k!} (A^\top)^r C^\top e_j s^\top (A^\top)^{k-1-r}.$$

If we choose $\tilde{Y} = uv^\top$, where u and v are the unit-norm right and left eigenvectors of A corresponding to its eigenvalue Λ with the largest real part, we obtain

$$\begin{aligned}\left\langle \tilde{Y}, \nabla_A \left(e_j^\top C \sum_{k=0}^{\infty} \frac{t^k}{k!} A^k s \right) \right\rangle &= \sum_{k=1}^{\infty} \sum_{r=0}^{k-1} \frac{t^k}{k!} \Lambda^{k-1} \langle u, C^\top e_j \rangle \langle v, s \rangle \\ &= \sum_{k=1}^{\infty} \frac{t^k}{(k-1)!} \Lambda^{k-1} \langle u, C^\top e_j \rangle \langle v, s \rangle \\ &= t e^{\Lambda t} \langle u, C^\top e_j \rangle \langle v, s \rangle \\ &= t e^{\Lambda t} \langle Cu, e_j \rangle \langle v, s \rangle.\end{aligned}$$

Remember that (5.11) is a lower bound for the largest eigenvalue of f_1, and so is

$$\begin{aligned}\sum_{j=1}^{n} \left| \left\langle \tilde{Y}, \nabla_A (e_j^\top C e^{At} s - e_j^\top x) \right\rangle \right|^2 &= \sum_{j=1}^{n} t^2 e^{2\text{Re}(\Lambda) t} \left| \langle Cu, e_j \rangle \right|^2 \left| \langle v, s \rangle \right|^2 \\ &= \rho^2 t^2 e^{2\text{Re}(\Lambda) t} \left| \langle v, s \rangle \right|^2,\end{aligned}$$

where $\text{Re}(\Lambda)$ is the largest real part of the eigenvalues of A and $\rho^2 = \|Cu\|_2^2$. If we have multiple trajectories, this lower bound will become

$$\sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}_k} \rho^2 t^2 e^{2\text{Re}(\Lambda) t} \left| \langle v, s_k \rangle \right|^2,$$

where $\{s_k\}_{k \in \mathcal{K}}$ is the set of initial states of the trajectories.

As a result, for convergence of the gradient descent algorithm to a solution $(\hat{A}, \hat{C}, \hat{s})$, it is necessary that

$$\sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}_k} \rho^2 t^2 e^{2\text{Re}(\Lambda) t} \left| \langle v, \hat{s}_k \rangle \right|^2 \leq \frac{2}{\delta}.$$

Without making any assumptions about the eigenvectors of $\hat{A}$, we can obtain the final necessary condition as

$$\lambda_{\text{min}} \left(\sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}_k} \rho^2 t^2 e^{2\text{Re}(\Lambda) t} \hat{s}_k \hat{s}_k^\top \right) \leq \frac{2}{\delta},$$

or equivalently as

$$\delta \leq \frac{2}{\lambda_{\text{min}} \left(\rho^2 \sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}_k} t^2 e^{2\text{Re}(\Lambda) t} \hat{s}_k \hat{s}_k^\top \right)}.$$

This completes the proof. $\square$

5.8.2 Proof of Theorem 5.2

Similar to the proof of Theorem 5.1, we will use Lemma 5.1 to find a lower bound for the largest eigenvalue of the linearized system around $(\hat{A}, \hat{C}, \{\hat{s}_k\}_{k\in\mathcal{K}})$. Note that

$$\begin{aligned}\nabla_A \log\left(e_j^\top C e^{At} s + \epsilon\right) &= \nabla_A \log\left(e_j^\top C \sum_{k=0}^{\infty} \frac{t^k}{k!} A^k s + \epsilon\right)\\ &= \frac{1}{e_j^\top C e^{At} s + \epsilon} \sum_{k=1}^{\infty}\sum_{r=0}^{k-1} \frac{t^k}{k!} (A^\top)^r C^\top e_j s^\top (A^\top)^{k-1-r}.\end{aligned}$$

For the matrix $\tilde{Y} = uv^\top$, where u and v are the right and left eigenvectors of A corresponding to its eigenvalue Λ with the largest real part, we have

$$\begin{aligned}\left\langle \tilde{Y}, \nabla_A \log\left(e_j^\top C e^{At} s + \epsilon\right)\right\rangle &= \frac{1}{e_j^\top C e^{At} s + \epsilon} \sum_{k=1}^{\infty} \frac{t^k}{(k-1)!} \Lambda^{k-1} \langle u, C^\top e_j\rangle \langle v, s\rangle\\ &= \frac{t e^{\Lambda t}}{e_j^\top C e^{At} s + \epsilon} \langle u, C^\top e_j\rangle\langle v, s\rangle.\end{aligned}$$

By using Lemma 5.1, we obtain a lower bound for the largest eigenvalue of the linearization of the gradient descent algorithm around $(\hat{A}, \hat{C}, \{\hat{s}_k\}_{k\in\mathcal{K}})$ as

$$\sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}_k} \sum_{j=1}^{n} \frac{1}{t^2} \left| \frac{t e^{\Lambda t}}{e_j^\top C e^{At} s_k + \epsilon} \langle Cu, e_j\rangle \langle v, s_k\rangle \right|^2 .$$

We can write a further lower bound for this expression as

$$\begin{aligned}&\sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}_k} \sum_{j=1}^{n} \frac{e^{2\mathrm{Re}(\Lambda)t}}{\left(\|Ce^{At}s_k\|_\infty + \epsilon\right)^2} \left|\langle Cu, e_j\rangle\right|^2 \left|\langle v, s_k\rangle\right|^2\\ &\qquad = \sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}_k} \frac{\rho^2 e^{2\mathrm{Re}(\Lambda)t}}{\left(\|Ce^{At}s_k\|_\infty + \epsilon\right)^2} \left|\langle v, s_k\rangle\right|^2,\end{aligned}$$

and finally,

$$\lambda_{\min}\left(\sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}_k} \frac{\rho^2 e^{2\mathrm{Re}(\Lambda)t}}{\left(\|Ce^{At}s_k\|_\infty + \epsilon\right)^2} s_k s_k^\top \right),$$

where $\rho^2 = \|\hat{C}u\|_2^2$ and u is the right-eigenvector of $\hat{A}$ corresponding to its eigenvalue Λ. For stability of the algorithm around the equilibrium point $(\hat{A}, \{\hat{s}_k\}_{k\in\mathcal{K}})$, we need

$$\lambda_{\min}\left(\sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}_k} \frac{\rho^2 e^{2\mathrm{Re}(\Lambda)t}}{\left(\|\hat{C}e^{\hat{A}t}\hat{s}_k\|_\infty + \epsilon\right)^2} \hat{s}_k \hat{s}_k^\top \right) \leq \frac{2}{\delta},$$

where δ is the step size of the algorithm.

5.8.3 Remarks on Convergence Rate

In the proof of Theorem 5.1, we considered the mapping

$$\tilde{A} \leftarrow \tilde{A} - \delta f_1(\tilde{A}),$$

where f_1 is the Jacobian of the gradient of the loss function

$$\ell(A, s) = \frac{1}{2}\|x - Ce^{At}s\|_2^2$$

with respect to A at the point $(\hat{A}, \hat{C}, \hat{s})$. For Theorem 5.1, we computed the largest learning rate at which the algorithm can still converge to the specified equilibrium. Note that this was equivalent to computing a lower bound for the largest eigenvalue of the mapping f_1. Similar to the proof of Theorem 5.1, we can compute an upper bound for the smallest eigenvalue of f_1 around the solution $(\hat{A}, \hat{C}, \hat{s})$.

By using Lemma 5.1, the smallest eigenvalue of the mapping f_1 can be upper bounded by

$$\min_{Y:\|Y\|_F=1} \sum_{j=1}^{n} \left|\langle Y, \nabla_A(e_j^\top Ce^{At}s - e_j^\top x)\rangle\right|^2. \tag{5.12}$$

Similar to the proof of Theorem 5.1, we can expand the matrix exponential:

$$\nabla_A\left(e_j^\top C\sum_{k=0}^{\infty}\frac{t^k}{k!}A^k s\right) = \sum_{k=1}^{\infty}\sum_{r=0}^{k-1}\frac{t^k}{k!}(A^\top)^r C^\top e_j s^\top (A^\top)^{k-1-r}.$$

If we choose $\tilde{Y} = uv^\top$, where u and v are the unit-norm right and left eigenvectors of A corresponding to its eigenvalue λ_2, we obtain

$$\begin{aligned}\left\langle \tilde{Y}, \nabla_A\left(e_j^\top C\sum_{k=0}^{\infty}\frac{t^k}{k!}A^k s\right)\right\rangle &= \sum_{k=1}^{\infty}\sum_{r=0}^{k-1}\frac{t^k}{k!}\lambda_2^{k-1}\langle u, C^\top e_j\rangle\langle v, s\rangle \\ &= \sum_{k=1}^{\infty}\frac{t^k}{(k-1)!}\lambda_2^{k-1}\langle u, C^\top e_j\rangle\langle v, s\rangle \\ &= te^{\lambda_2 t}\langle u, C^\top e_j\rangle\langle v, s\rangle \\ &= te^{\lambda_2 t}\langle Cu, e_j\rangle\langle v, s\rangle.\end{aligned}$$

Remember that (5.12) is an upper bound for the smallest eigenvalue of f_1, and so is

$$\begin{aligned}\sum_{j=1}^{n}\left|\left\langle \tilde{Y}, \nabla_A(e_j^\top Ce^{At}s - e_j^\top x)\right\rangle\right|^2 &= \sum_{j=1}^{n} t^2 e^{2\mathrm{Re}(\lambda_2)t}\left|\langle Cu, e_j\rangle\right|^2\left|\langle v, s\rangle\right|^2 \\ &= \rho^2 t^2 e^{2\mathrm{Re}(\lambda_2)t}\left|\langle v, s\rangle\right|^2,\end{aligned}$$

where $\rho^2 = \|Cu\|_2^2$. If we have multiple trajectories, this upper bound will become

$$\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} \rho^2 t^2 e^{2\mathrm{Re}(\lambda_2)t} \left|\langle v, s_k\rangle\right|^2,$$

where $\{s_k\}_{k\in\mathcal{K}}$ is the set of initial states of the trajectories. We can bring this upper bound into a form independent of v:

$$\lambda_{\max}\left(\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} \rho^2 t^2 e^{2\mathrm{Re}(\lambda_2)t} s_k s_k^\top\right).$$

This shows that the ratio of the largest eigenvalue to the smallest eigenvalue of f_1 satisfies

$$\frac{\lambda_{\max}(f_1)}{\lambda_{\min}(f_1)} \geq \frac{\lambda_{\min}\left(\rho_1^2 \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} t^2 e^{2Re(\lambda_1)t}\hat{s}_k\hat{s}_k^\top\right)}{\lambda_{\max}\left(\rho_2^2 \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} t^2 e^{2Re(\lambda_2)t}\hat{s}_k\hat{s}_k^\top\right)}$$

for any pair of eigenvalues (λ_1, λ_2) of $\hat{A}$, where $\rho_1 = \|Cu_1\|_2$, $\rho_2 = \|Cu_2\|_2$, and u_1, u_2 are the right eigenvectors of $\hat{A}$ corresponding to λ_1, λ_2. If H denotes the Hessian of the loss function ℓ at the point $(\hat{A}, \hat{C}, \{\hat{s}_k\}_{k\in\mathcal{K}})$, we have $\lambda_{\max}(H) \geq \lambda_{\max}(f_1)$ and $\lambda_{\min}(H) \leq \lambda_{\min}(f_1)$. Therefore, we also have

$$\frac{\lambda_{\max}(H)}{\lambda_{\min}(H)} \geq \frac{\lambda_{\min}\left(\rho_1^2 \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} t^2 e^{2Re(\lambda_1)t}\hat{s}_k\hat{s}_k^\top\right)}{\lambda_{\max}\left(\rho_2^2 \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}_k} t^2 e^{2Re(\lambda_2)t}\hat{s}_k\hat{s}_k^\top\right)}. \tag{5.13}$$

To understand the relationship of (5.13) to the convergence rate, consider a quadratic function $h : \mathbb{R}^n \mapsto \mathbb{R}$ defined as

$$h(w) = \frac{1}{2}(w - w^*)^\top H(w - w^*),$$

where H is the Hessian of h and w^* is the point where h attains its minimum. For the gradient descent algorithm

$$w \leftarrow w - \delta H(w - w^*)$$

to converge to the minimum of h from arbitrary initializations, we need the learning rate δ to be smaller than $\frac{2}{\lambda_{\max}(H)}$. Assume $(w_0 - w^*)$, where w_0 is the initial point where the algorithm starts, is in the direction of the eigenvector of H corresponding to its minimum eigenvalue. In other words,

$$H(w_0 - w^*) = \lambda_{\min}(H)(w_0 - w^*).$$

Then the iterations of the gradient descent algorithm becomes

$$\begin{aligned}(w_k - w^*) &\leftarrow (w_{k-1} - w^*) - \delta H(w_{k-1} - w^*)\\ &\leftarrow (w_{k-1} - w^*) - \delta\lambda_{\min}(H)(w_{k-1} - w^*)\\ &\leftarrow (1 - \delta\lambda_{\min}(H))(w_{k-1} - w^*)\\ &\leftarrow (1 - \delta\lambda_{\min}(H))^k(w_0 - w^*).\end{aligned}$$

Attaining $\|w_k - w^*\|_2 \leq \epsilon$ for any $\epsilon > 0$ will require

$$(1-\delta\lambda_{\min}(H))^k \|w_0 - w^*\|_2 \leq \epsilon \implies k\log(1-\delta\lambda_{\min}(H)) + \log(\|w_0 - w^*\|_2) \leq \log(\epsilon),$$

which gives a lower bound for the number of iterations needed:

$$k \geq \frac{1}{\log\left(\frac{1}{1-\delta\lambda_{\min}(H)}\right)}\left(\log\left(\frac{1}{\epsilon}\right) + \log(\|w_0 - w^*\|_2)\right).$$

As a result, convergence of the gradient descent algorithm to the minimum of h in the direction of the bottom eigenvector of H requires

$$O\left(\left[\log\left((1-\delta\lambda_{\min}(H))^{-1}\right)\right]^{-1}\right) \tag{5.14}$$

iterations. Remember that for convergence of the algorithm, we require $\delta < \frac{2}{\lambda_{\max}(H)}$; therefore, $\delta\lambda_{\min}(H) < 2\frac{\lambda_{\min}(H)}{\lambda_{\max}(H)}$. Combining (5.13) and (5.14) gives the local convergence rate for the loss function ℓ, if we assume the second approximation of ℓ represents it well around $(\hat{A}, \hat{C}, \{\hat{s}_k\}_{k\in\mathcal{K}})$.

5.8.4 Alternatives for Initial States

For the proof of Theorem 5.1 and Theorem 5.2, we considered the loss function

$$\ell(A, C, s) = \frac{1}{2}\sum_{t\in\mathcal{T}} \|x(t) - Ce^{At}s\|_2^2,$$

and analyzed the linearization of the dynamics of the gradient descent algorithm around the solution $(\hat{A}, \hat{C}, \hat{s})$:

$$\tilde{A} \leftarrow f_{1,1}(\tilde{A}) + f_{1,2}(\tilde{C}) + f_{1,3}(\tilde{s}) \tag{5.15a}$$

$$\tilde{C} \leftarrow f_{2,1}(\tilde{A}) + f_{2,2}(\tilde{C}) + f_{2,3}(\tilde{s}) \tag{5.15b}$$

$$\tilde{s} \leftarrow f_{3,1}(\tilde{A}) + f_{3,2}(\tilde{C}) + f_{3,3}(\tilde{s}), \tag{5.15c}$$

where $\{f_{i,j}\}_{i\in[3],j\in[3]}$ are the Jacobians of the partial derivatives of ℓ with respect to A, C and s, evaluated at the point $(\hat{A}, \hat{C}, \hat{s})$. We used the fact that system (5.15) can be represented by a symmetric matrix to use only the eigenvalues of $f_{1,1}$ in order to obtain a lower bound for the largest eigenvalue of the system (5.15).

Note that fixing the initial state s and not updating it with the gradient descent algorithm will not affect the eigenvalues of $f_{1,1}$. Therefore, the results for Theorem 5.1 and Theorem 5.2, which only depend on the largest eigenvalues of $f_{1,1}$, will still hold when the initial state is fixed.

Now assume the initial state is obtained via a state estimator:

$$s = g_\phi(\{t, x(t)\}_{t\in\mathcal{T}}),$$

where $\mathcal{T}$ is the set of sampling instants for the trajectory and $\{x_t\}_{t\in\mathcal{T}}$ is the set of samples obtained. While solving the problem

$$\underset{A,C,\phi}{\text{minimize}} \quad \sum_{t\in\mathcal{T}} \ell\left(x(t), Ce^{At} g_\phi\left(\{t, x(t)\}_{t\in\mathcal{T}}\right)\right) + \mathcal{L}\left(\phi\right),$$

the linear approximation to the gradient descent algorithm can be written as

$$\tilde{A} \leftarrow \hat{f}_{1,1}(\tilde{A}) + \hat{f}_{1,2}(\tilde{C}) + \hat{f}_{1,3}(\tilde{\phi}), \tag{5.16a}$$

$$\tilde{C} \leftarrow \hat{f}_{2,1}(\tilde{A}) + \hat{f}_{2,2}(\tilde{C}) + \hat{f}_{2,3}(\tilde{\phi}), \tag{5.16b}$$

$$\tilde{\phi} \leftarrow \hat{f}_{3,1}(\tilde{A}) + \hat{f}_{3,2}(\tilde{C}) + \hat{f}_{3,3}(\tilde{\phi}), \tag{5.16c}$$

where $\{\hat{f}_{i,j}\}_{i\in[3],j\in[3]}$ are the Jacobians of the partial derivatives of ℓ with respect to A, C and ϕ, evaluated at the point $(\hat{A}, \hat{C}, \hat{\phi})$. Note that system (5.16) can still be represented by a symmetric matrix; therefore, the largest eigenvalues of $\hat{f}_{1,1}$ can be used to obtain an upper bound on the learning rate of the algorithm. Furthermore, given that $\frac{\partial\mathcal{L}}{\partial A} = 0$, $f_{1,1}$ in (5.15) and $\hat{f}_{1,1}$ in (5.16) are identical, with the substitution $s = g_\phi\left(\{t, x(t)\}_{t\in\mathcal{T}}\right)$. For this reason, the results of Theorem 5.1 and Theorem 5.2 still hold for systems with a state estimator g_ϕ, provided that the estimation error at equilibrium is zero; that is,

$$\sum_{t\in\mathcal{T}} \ell\left(x(t), \hat{C}e^{\hat{A}t} g_{\hat{\phi}}\left(\{t, x(t)\}_{t\in\mathcal{T}}\right)\right) = 0,$$

which is needed only to allow the use of Lemma 5.1.

Chapter 6

Learning Risk-Sensitive Value Functions with Sequential Decisions

6.1 Introduction

Utility functions are used to represent the preferences of a person for a set of outcomes. They assign larger values to the outcomes which are more preferable to the person than the others. Having these functions enables understanding and predicting the decisions of people.

In the traditional economics literature, based on some rationality and consistency axioms, people are assumed to make their decisions by maximizing their utility function or its expectation if the consequences of their decision are stochastic (Rubinstein, 2012). However, people are observed to deviate from these axioms of rationality in real life (Kahneman and Tversky, 1979). Prospect theory provides one of the first and most acknowledged decision models capable of explaining the observed behavior of people (Kahneman and Tversky, 1979).

According to prospect theory, given a decision problem, people first create a reference point in their mind. This reference point could depend on several factors, such as the status quo (Tversky and Kahneman, 1991) or the recent expectations of the person about the future (Kőszegi and Rabin, 2006). After determining a reference point, the outcomes that are more preferable compared to the reference point are considered as gain, and the others are considered as loss. It has been observed that the effect of a loss is greater on decisions than that of an equal amount of gain, which is called loss aversion (Tversky and Kahneman, 1991).

Risk attitudes of people are also influenced by their reference point. People become more risk averse when making a choice between gains and more risk seeking when making a choice between losses. Consequently, to express the effect of the reference point on the decisions, prospect theory replaces the utility function of a person with a value function, which is a function of both the outcome and the reference point.

If $U(x; r)$ denotes the value function of a person for the outcome x given the reference point r, and if larger x values correspond to better outcomes, then the risk averse and risk seeking behavior of the person can be reflected in $U(x; r)$ as concavity for $x > r$ and convexity for $x < r$. In addition, loss aversion causes $U(x;r)$ to change more sharply for losses than it does for gains. As a result, a value function has the S-shape as shown in Figure 6.1.

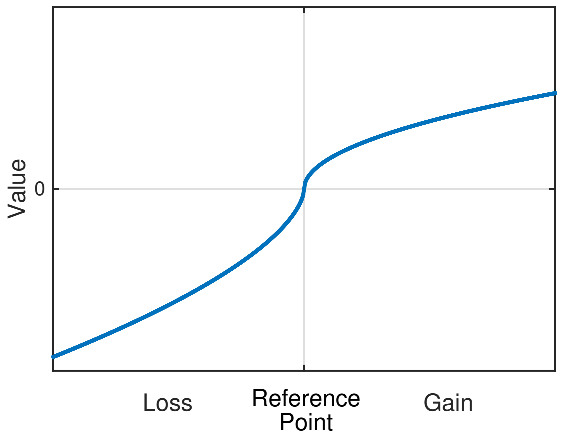


Figure 6.1: Value function of prospect theory

Learning the utility function of a decision maker is well studied in the literature with rationality axioms and expected utility theory; e.g., (Ng and Russell, 2000), (Chajewska et al., 2001). Despite their stronger ability to describe the behavior of people in real life, the literature on estimation of value functions and reference points, on the other hand, is rather limited. In particular, reference points are usually chosen heuristically as the median, average, best or worst values of possible outcomes (Avineri and Bovy, 2008; Gao et al., 2010; Zhou et al., 2014). An iterative algorithm is suggested in (Hu et al., 2012) that shifts the estimate of the reference point until the person exhibits loss aversion around that estimate. This algorithm produces a fixed reference point for the person.

In this chapter, we assume that a person is given a decision problem repeatedly and the person chooses an action to maximize her value function. We allow the reference point of the person to change over time, and we learn the value function and the dynamics of the reference point from the observed actions. The organization of the paper is as follows. The next section introduces the type of decision problems we consider and presents the problem formulation. The relation between the optimal actions and the value functions is obtained in Section 5.3. A hidden Markov model is constructed for the sequential decision problem and expectation-maximization (EM) is used to learn the value function and the reference point of the decision maker in Section 5.4. Section 5.5 extends the results of Section 5.3 to nonnegative actions. In Section 5.6, the algorithm suggested is tested on the data of New York City taxi drivers. Section 5.7 discusses some future directions and concludes this chapter. The results in this chapter have appeared in (Nar et al., 2017).

6.2 Formulation of the Decision Problem

We consider a certain group of decision problems which involve determining a balance between two contrasting factors. Many decision problems belong to this group; for example, when buying a product, the buyer needs to find a middle point between the price and the quality of the product. Similarly, while using a service, increase in the speed of the service might require or lead to decrease in the quality or the safety, and one needs to decide how much to compromise on one or the other.

We will assume that the decision maker has a separate value function for each of the contrasting factors, and their values are added for the decision:

$$U(x; r_1, r_2) = U_1(x; r_1) + U_2(x; r_2).$$

The variables r_1 and r_2 denote the reference points of the decision maker for each factor. We assume that larger x values correspond to better outcomes for the factor represented by U_1 and worse outcomes for the factor represented by U_2. Therefore, $U_1(x; r_1)$ and $U_2(x; r_2)$ are increasing and decreasing functions of x, respectively, for every r_1 and r_2.

When a person is going to buy a computer, for example, she has in mind some desired features for the computer, r_1, and she sets some price amount that she is willing to pay, r_2. Increasing the price, x, improves the features of the computer, which can be reflected with U_1. On the other hand, buying a computer for a price higher than r_2 feels like a loss, which can be described by U_2. As another example, when a person is driving a car, she may want to increase the speed of the car to arrive at her destination earlier, but increasing the speed will decrease her safety. In this case, U_1 and U_2 correspond to the values of the duration and the safety of the trip, respectively, and the chosen speed will depend on how soon she needs to be at her destination, r_1, and on how risky a driver she is, r_2.

A value function as shown in Figure 6.1 can be approximated as

$$U(x; r) = \begin{cases} |x - r|^p & \text{if } x \geq r \\ \Lambda |x - r|^q & \text{if } x < r \end{cases}$$

where $p, q \in (0, 1)$ and $\Lambda > 1$ is the loss aversion coefficient (Tversky and Kahneman, 1992). In this paper, we use

$$U_1(x; r_1) = \begin{cases} (x - r_1)^p & \text{if } x \geq r_1 \\ -a(r_1 - x)^p & \text{if } x < r_1 \end{cases}$$

$$U_2(x; r_2) = \begin{cases} -b(x - r_2)^p & \text{if } x \geq r_2 \\ c(r_2 - x)^p & \text{if } x < r_2 \end{cases}$$

for some fixed $p \in (0, 1)$, $b > c > 0$ and $a > 1$. The choice of $p = q$ will help elicit an explicit relation between the reference points of the decision maker and her actions in the following sections.

Our goal is to learn the parameters a, b, c for a person and the reference points r_1 and r_2, along with their dynamics, when the person is given the same decision problem repeatedly and the actions of the person are observed.

6.3 Deriving Optimal Actions from the Value Function

Let r denote the pair of reference points $(r_1, r_2) \in R$, where $R \subset \mathbb{R}^2$ is a finite set, and let $x \in \mathbb{R}$ denote the action of the decision maker. An optimal action x^* is assumed to maximize the function

$U(x; r)$:

$$x^* = \arg\max_{x\in\mathbb{R}} U(x; r).$$

In order to guarantee the existence of an optimal action in this section, we assume $a > c$ and $b > 1$ so that

$$\lim_{x\to\pm\infty} U(x; r) \neq \infty.$$

Note that due to loss aversion, we also have $a > 1$ and $b > c$.

Depending on the reference points (r_1, r_2), the optimal action of the person will change. Figure 6.2 illustrates $U(x; r_1, r_2)$ with $p = 0.5$, $a = 1.5, b = 2$ and $c = 1$ for two different pairs of reference points: $(r_1, r_2) = (1, 4)$ and $(r_1, r_2) = (3, 2)$. The markers on the plots denote the optimal actions.

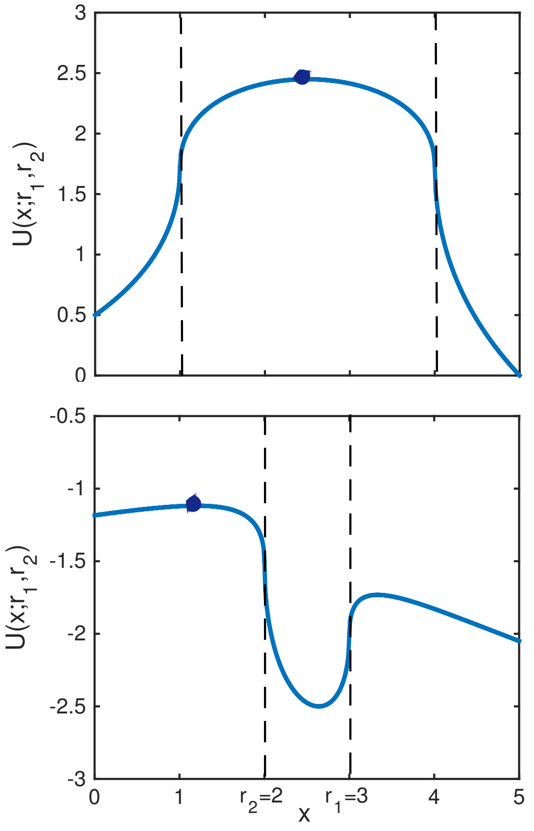


Figure 6.2: $U(x; r_1, r_2)$ with $p = 0.5$, $a = 1.5, b = 2, c = 1$ when $(r_1, r_2) = (1, 4)$ [top] and $(r_1, r_2) = (3, 2)$ [bottom].

We want to find the point, x^*, at which the function U is maximized. Since U is not differentiable at some points, we are going to consider each possible case separately:

1a) $x_1^* < r_1 \leq r_2$

$$U(x; r_1, r_2) = -a(r_1 - x)^p + c(r_2 - x)^p$$

$$\frac{\partial U}{\partial x} = \frac{pa}{(r_1 - x)^{(1-p)}} - \frac{pc}{(r_2 - x)^{(1-p)}}$$

Since $a > c$ and $(r_1 - x) \leq (r_2 - x)$, U is monotonically increasing in x, and there is no local maximum in this region.

1b) $r_1 \leq x_2^* \leq r_2$

$$U(x; r_1, r_2) = (x - r_1)^p + c(r_2 - x)^p$$

U is strictly concave in this region:

$$\frac{\partial U}{\partial x} = \frac{p}{(x - r_1)^{(1-p)}} - \frac{cp}{(r_2 - x)^{(1-p)}} = 0$$

$$x_2^* = \frac{c^{1/(1-p)}}{c^{1/(1-p)} + 1} r_1 + \frac{1}{c^{1/(1-p)} + 1} r_2 \in (r_1, r_2)$$

1c) $r_1 \leq r_2 < x_3^*$

$$U(x; r_1, r_2) = (x - r_1)^p - b(x - r_2)^p$$

$$\frac{\partial U}{\partial x} = \frac{p}{(x - r_1)^{(1-p)}} - \frac{b}{(x - r_2)^{(1-p)}}$$

Since $b > 1$ and $(x - r_2) \leq (x - r_1)$, U is monotonically decreasing in x, and there is no local maximum in this region.

2a) $x_4^* \leq r_2 < r_1$

$$U(x; r_1, r_2) = -a(r_1 - x)^p + c(r_2 - x)^p$$

$$\frac{\partial U}{\partial x} = 0 \implies x_4^* = r_2 - \frac{c^{1/(1-p)}}{a^{1/(1-p)} - c^{1/(1-p)}}(r_1 - r_2)$$

$$U(x_4^*; r_1, r_2) = -(a^{1/(1-p)} - c^{1/(1-p)})^{(1-p)}(r_1 - r_2)^p$$

2b) $r_2 < x_5^* < r_1$

$$U(x; r_1, r_2) = -a(r_1 - x)^p - b(x - r_2)^p$$

U is strictly convex in this region, so there is no local maximum.

2c) $r_2 < r_1 \leq x_6^*$

$$U(x; r_1, r_2) = (x - r_1)^p - b(x - r_2)^p$$

$$\frac{\partial U}{\partial x} = 0 \implies x_6^* = r_1 + \frac{1}{b^{1/(1-p)} - 1}(r_1 - r_2)$$

$$U(x_6^*; r_1, r_2) = -(b^{1/(1-p)} - 1)^{(1-p)}(r_1 - r_2)^p$$

We observe that if $r_1 \leq r_2$, there is a unique local maximum at x_2^*, and hence, it is the global maximum. On the other hand, if $r_1 > r_2$, there are two local maxima at x_4^* and x_6^*. However, for fixed values of a, b and c, the relation between $U(x_4^*; r_1, r_2)$ and $U(x_6^*; r_1, r_2)$ is also fixed and independent of (r_1, r_2). Therefore, a person with fixed parameters always chooses either x_4^* or x_6^* whenever her reference point satisfies $r_1 > r_2$.

If we define $\lambda \in (0,1), \beta_1 \in (0,\infty)$ and $\beta_2 \in (0,\infty)$ as

$$\begin{aligned}\lambda &= \frac{c^{1/(1-p)}}{c^{1/(1-p)}+1},\\ \beta_1 &= \frac{c^{1/(1-p)}}{a^{1/(1-p)}-c^{1/(1-p)}},\\ \beta_2 &= \frac{1}{b^{1/(1-p)}-1},\end{aligned}$$

we can summarize the computation of x^* as in Table I.

Table 6.1: Optimal actions

C1: $\frac{\beta_1(1-\lambda)}{\lambda} > \beta_2$	$r_1 \leq r_2$	$x^* = \lambda r_1 + (1-\lambda) r_2$
	$r_1 > r_2$	$x^* = r_2 - \beta_1(r_1 - r_2)$
C2: $\frac{\beta_1(1-\lambda)}{\lambda} \leq \beta_2$	$r_1 \leq r_2$	$x^* = \lambda r_1 + (1-\lambda) r_2$
	$r_1 > r_2$	$x^* = r_1 + \beta_2(r_1 - r_2)$

Case C1 and C2 correspond to people who choose x_4^* and x_6^*, respectively, when $r_1 > r_2$. Since either C1 or C2 is going to hold for a specific person, it is clear that either β_1 or β_2 will not appear in the optimal actions, and the segment of the utility function that is related to this unobserved parameter will never be used, nor will it be needed. We will obtain a bound on this parameter, however, by the condition of C1 or C2.

6.4 Learning Parameters from a Hidden Markov Model

Let the person be given the same decision problem repeatedly. We assume that the reference point r of this person evolves as a time-homogeneous Markov chain, which has a probability transition matrix A. Our goal is to estimate the parameters $\lambda, \beta_1, \beta_2$ and the matrix A from the observed actions $\{y^k\}_{k=0}^N$ of this person from time 0 to N.

First, consider the case C1. Given the reference point r, we are going to model the action y of the person as a Gaussian with mean $\mu(r) = x^*$ and variance σ^2:

$$p(y|r) \sim \mathcal{N}(\mu(r), \sigma^2),$$

where

$$\mu(r) = \begin{cases} \lambda r_1 + (1-\lambda) r_2 & \text{if } r_1 \le r_2, \\ -\beta_1 r_1 + (1+\beta_1) r_2 & \text{if } r_1 > r_2. \end{cases}$$

The graphical representation of this hidden Markov model is given in Figure 6.3.

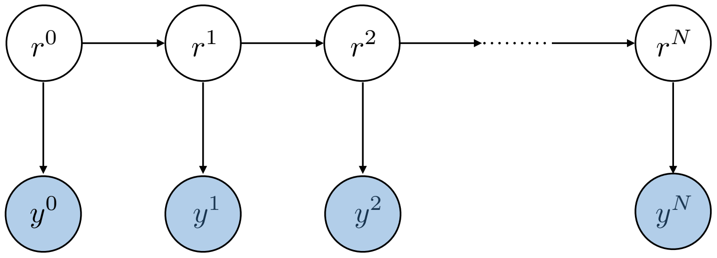


Figure 6.3: Graphical representation of sequential decision making

We can write the complete loglikelihood for this model as

$$L(r, y) = \log p(r^0) + \sum_{k=0}^{N-1} \log p(r^{k+1} | r^k) + \sum_{k=0}^{N} \log p(y^k | r^k).$$

Let $a_{rr'}$ denote $p(r^{k+1} = r' | r^k = r)$ and π_r denote $p(r^0 = r)$. In addition, let

$$z_r^k = \mathbb{I}(r^k = r) = \begin{cases} 1 & \text{if } r^k = r, \\ 0 & \text{if } r^k \neq r. \end{cases}$$

Then, the complete loglikelihood can be expressed as

$$\begin{aligned} L(r, y) = & \sum_{r \in R} z_r^0 \log(\pi_r) + \sum_{k=0}^{N-1} \sum_{r, r' \in R} z_r^k z_{r'}^{k+1} \log(a_{rr'}) \\ & - \sum_{k=0}^{N} \sum_{r \in R} z_r^k \frac{1}{2\sigma^2} (y^k - \lambda r_1 - (1-\lambda) r_2)^2 \mathbb{I}(r_1 \le r_2) \\ & - \sum_{k=0}^{N} \sum_{r \in R} z_r^k \frac{1}{2\sigma^2} (y^k + \beta_1 r_1 - (1+\beta_1) r_2)^2 \mathbb{I}(r_1 > r_2) \\ & - \frac{1}{2} \sum_{k=0}^{N} \left(\log(2\pi) + \log(\sigma^2) \right). \end{aligned}$$

To estimate the unknown parameters, we implement the EM algorithm (Hastie et al., 2009). At the E step, we compute

$$\mathbb{E}[z_r^k | y] = p(r^k = r | y) = \gamma_k^r,$$

$$\mathbb{E}[z_r^k z_{r'}^{k+1}|y] = p(r^k = r, r^{k+1} = r'|y) = \xi_{k,k+1}^{r,r'},$$

where γ_k^r and $\xi_{k,k+1}^{r,r'}$ could be obtained by $\alpha - \gamma$ algorithm (Hastie et al., 2009). At M step, maximization over each parameter leads to

$$\hat{\pi}_r = \gamma_0^r, \quad \hat{a}_{rr'} = \frac{\sum_{k=0}^{N-1} \xi_{k,k+1}^{r,r'}}{\sum_{k=0}^{N-1} \gamma_k^r},$$

$$\hat{\lambda} = \frac{\sum_{r\in R}(r_2 - r_1)\mathbb{I}(r_1 \le r_2)\sum_{k=0}^N \gamma_k^r(r_2 - y^k)}{\sum_{r\in R}(r_2 - r_1)^2\mathbb{I}(r_1 \le r_2)\sum_{k=0}^N \gamma_k^r},$$

$$\hat{\beta}_1 = \frac{\sum_{r\in R}(r_1 - r_2)\mathbb{I}(r_1 > r_2)\sum_{k=0}^N \gamma_k^r(r_2 - y^k)}{\sum_{r\in R}(r_1 - r_2)^2\mathbb{I}(r_1 > r_2)\sum_{k=0}^N \gamma_k^r},$$

$$\hat{\sigma}^2 = \sum_{r\in R} \frac{\mathbb{I}(r_1 \le r_2)}{N+1} \sum_{k=0}^N \gamma_k^r (y^k - \hat{\lambda} r_1 - (1-\hat{\lambda}) r_2)^2 + \sum_{r\in R} \frac{\mathbb{I}(r_1 > r_2)}{N+1} \sum_{k=0}^N \gamma_k^r (y^k + \hat{\beta}_1 r_1 - (1+\hat{\beta}_1) r_2)^2.$$

After estimating the parameters and computing the loglikelihood, we repeat the same procedure for the case C2 and choose the case with the higher loglikelihood. For case C2, the actions are modeled as

$$p(y|r) \sim \mathcal{N}(\mu(r), \sigma^2),$$

where

$$\mu(r) = \begin{cases} \lambda r_1 + (1-\lambda) r_2 & \text{if } r_1 \le r_2, \\ (1+\beta_2) r_1 - \beta_2 r_2 & \text{if } r_1 > r_2. \end{cases}$$

The E step and the M step are similar to the previous case with the modification

$$\hat{\beta}_2 = \frac{\sum_{r\in R}(r_1 - r_2)\mathbb{I}(r_1 > r_2)\sum_{k=0}^N \gamma_k^r(y^k - r_1)}{\sum_{r\in R}(r_1 - r_2)^2\mathbb{I}(r_1 > r_2)\sum_{k=0}^N \gamma_k^r},$$

$$\hat{\sigma}^2 = \sum_{r\in R} \frac{\mathbb{I}(r_1 \le r_2)}{N+1} \sum_{k=0}^N \gamma_k^r (y^k - \hat{\lambda} r_1 - (1-\hat{\lambda}) r_2)^2 + \sum_{r\in R} \frac{\mathbb{I}(r_1 > r_2)}{N+1} \sum_{k=0}^N \gamma_k^r (y^k - (1+\hat{\beta}_2) r_1 - \hat{\beta}_2 r_2)^2.$$

6.5 Learning Parameters for Nonnegative Actions

It is usually the case that the action space is bounded from below and/or above. For instance, the price of a computer mentioned in Section 6.2 cannot take a negative value, and choosing zero as the action corresponds to opting-out and not buying a computer. In this section, we assume $r_1, r_2 \in [0, \infty)$ and restrict the action space to be $[0, \infty)$, so the optimal action becomes

$$x^* = \arg \max_{x \in [0,\infty)} U(x; r_1, r_2).$$

Again, to guarantee the existence of an optimal action, we require $b > 1$. Depending on the relation between a and c, the possible cases for optimal actions will change. Since the computation of the optimal action x^* is similar to that in Section 6.3, we only provide the results in Table 6.2.

In some cases, value of the function U at zero needs to be compared with its local maxima elsewhere, and this leads to the dependence of some conditions both on the unknown parameters and on the reference points. As a result, the number of cases for which we need to run an EM algorithm becomes much larger than that in Section 6.4.

6.6 Analysis of New York City Taxi Drivers

New York City (NYC) Taxi and Limousine Commission has been collecting the data of all taxi trips in NYC since 2010 (Donovan and Work, 2014). The collected data set contains the date, time and location of pick-up and drop-off of passengers, the fare amount of trips, and the identification number of the drivers and the vehicles (which are reassigned each year for anonymity of the drivers). The data set has attracted much attention for the information it has provided for labor economics (Kőszegi and Rabin, 2006; Camerer et al., 1997; Farber, 2008; Crawford and Meng, 2011) and transportation problems (Donovan and Work, 2015; Terelius and Johansson, 2015).

In (Camerer et al., 1997), for example, some drivers were shown to drive for a shorter time in the afternoon than usual if they earned a larger amount in the morning than they had anticipated. This was attributed to taxi drivers having a reference amount to earn each day. If the drivers earned less than they had expected, they continued driving; and they quit earlier if they earned more than what they had expected.

The decision of the taxi drivers about when to stop driving belongs to the group of decision problems involving two contrasting factors. The drivers want to have a large earning each day, which requires working for longer time; but on the other hand, they value their free time as well. Therefore, we can express the value of these factors with two terms: daily earning amount with U_1, and daily work time with U_2.

We selected 7 drivers from the data set and analyzed their daily earning and work time over the time interval from April 1st 2010 until June 30th 2010. The drivers were chosen such that 3 of them had negative correlation, 3 of them had positive correlation and 1 of them had little correlation between their daily earning rate and work time. In addition, the chosen drivers worked at least 60 consecutive days in the three-month interval, and the amount of time they worked each day had no conspicuous periodicity, such as working extra hours on Mondays or on weekends.

Table 6.2: Optimal nonnegative actions

$a < c$ and $r_1 \leq r_2$:	$x^* = \begin{cases} \lambda r_1 + (1-\lambda) r_2 & \text{if } (c^{\frac{1}{(1-p)}} + 1)^{1-p} > \frac{c r_2^p - a r_1^p}{(r_2 - r_1)^p} \\ 0 & \text{otherwise} \end{cases}$
$a < c$ and $r_1 > r_2$:	$x^* = \begin{cases} (1+\beta_2) r_1 - \beta_2 r_2 & \text{if } (b^{\frac{1}{(1-p)}} - 1)^{1-p} < \frac{a r_1^p - c r_2^p}{(r_1 - r_2)^p} \\ 0 & \text{otherwise} \end{cases}$
$a > c$ and $r_1 \leq r_2$:	$x^* = \lambda r_1 + (1-\lambda) r_2$
$a > c$ and $r_1 > r_2$:	$x^* = \begin{cases} (1+\beta_2) r_1 - \beta_2 r_2 & \text{if } b^{\frac{1}{(1-p)}} - 1 < a^{\frac{1}{(1-p)}} - c^{\frac{1}{(1-p)}} \\ & \text{or } b^{\frac{1}{(1-p)}} - 1 > a^{\frac{1}{(1-p)}} - c^{\frac{1}{(1-p)}} \\ & \text{and } r_1/r_2 \leq a^{\frac{1}{(1-p)}} / c^{\frac{1}{(1-p)}} \\ & \text{and } (b^{\frac{1}{(1-p)}} - 1)^{1-p} < \frac{a r_1^p - c r_2^p}{(r_1 - r_2)^p} \\ r_2 - \beta_1 (r_1 - r_2) & \text{if } b^{\frac{1}{(1-p)}} - 1 > a^{\frac{1}{(1-p)}} - c^{\frac{1}{(1-p)}} \\ & \text{and } r_1/r_2 > a^{\frac{1}{(1-p)}} / c^{\frac{1}{(1-p)}} \\ 0 & \text{otherwise} \end{cases}$

The daily earning of each driver was calculated by subtracting the toll fees from the total payment to the driver in that day.[1] The work time was computed by summing up the trip durations

[1]The beginning of the day was decided based on what time the driver started to drive every day.

in a day and adding the time intervals between dropping off a passenger and picking up the next passenger as long as they did not exceed 20 minutes. That is, any interval exceeding 20 minutes without a fare was regarded as a break and not included in the work time. Finally, the daily earning rate was calculated as the ratio of the earning amount on a day to the work time on that day.

After computing the earning, the work time and the earning rate of all drivers for each day, we determined their set of reference points for earning amount and work time based on the histogram of their data. Specifically, for each driver, we chose 3 reference points $\{r_{1L}, r_{1M}, r_{1H}\}$ for the daily earning amount as the 10th, 50th and 90th percentile of their earning amounts in the 60-90 day interval over which their data were analyzed. Then we obtained 3 other reference points $\{r_{2L}, r_{2M}, r_{2H}\}$ for their work time in the same way. As a result, the set of reference points for a driver was

$$R = \big\{(r_1, r_2)\big| r_1 \in \{r_{1L}, r_{1M}, r_{1H}\}, r_2 \in \{r_{2L}, r_{2M}, r_{2L}\}\big\}.$$

In the derivation of the optimal actions from the value functions in Section III, the functions $U_1(\cdot\,; r_1)$ and $U_2(\cdot\,; r_2)$ had the same variable as their argument. To analyze the data of the drivers, we needed to express the earning amount for each day in terms of the work time on that day, or vice versa. We assumed the earning rate of the driver to be a constant for each day, and built a linear relation between the earning amount and the work time. As a result, the value function of the driver on the k^{th} day was described as

$$U^{(k)}(x; r_1, r_2) = U_1^{(k)}\left(x; \frac{r_1}{E^{(k)}}\right) + U_2^{(k)}\Big(x; r_2\Big),$$

where $E^{(k)}$ is the earning rate of the driver on the k^{th} day.

We used the EM algorithm suggested in Section 6.4 to learn the parameters of the chosen drivers, along with the transition probabilities of their reference points. The estimated parameters are given in Table 6.3.

Table 6.3: Estimated parameters for the chosen drivers

Driver ID	Correlation	$c^{1/(1-p)}$	$a^{1/(1-p)}$	days
2010001271	-0.38	3.16	12.21	90
2010002704	-0.29	0.82	2.23	90
2010007579	-0.18	8.09	25.30	60
2010007519	0.04	0.41	1.65	90
2010007770	0.09	0.43	1.31	90
2010003240	0.20	0.61	1.10	77
2010002920	0.23	0.32	1.16	60

The results given in Table 6.3 suggest that the drivers who had negative correlation between their daily earning rate and daily work time had larger a and c values, provided that all the drivers had comparable values for p. Larger a and c values give us two interpretations:

1. These drivers were highly loss averse about their daily earning, which would require them to drive longer if they could not earn their reference earning amount.

2. These drivers assigned higher value for their free time than the other drivers; therefore, they chose to stop driving earlier even if their earning rate was high.

Note that both of these interpretations explain why these drivers had negative correlation between their daily earning rate and daily work time.

The EM algorithm yields the likelihood of each reference point for each day as well. As an example, the daily work time and the daily earning rate of the driver with ID number 2010001271 over the week 20-26 April 2010 are plotted in Figure 6.4. Estimated reference points with the highest likelihood for each day of that week are also given in Table 6.4 for comparison. We observe that the driver worked longer on the first five days, and the estimated reference point for daily earning was medium or high on those days.

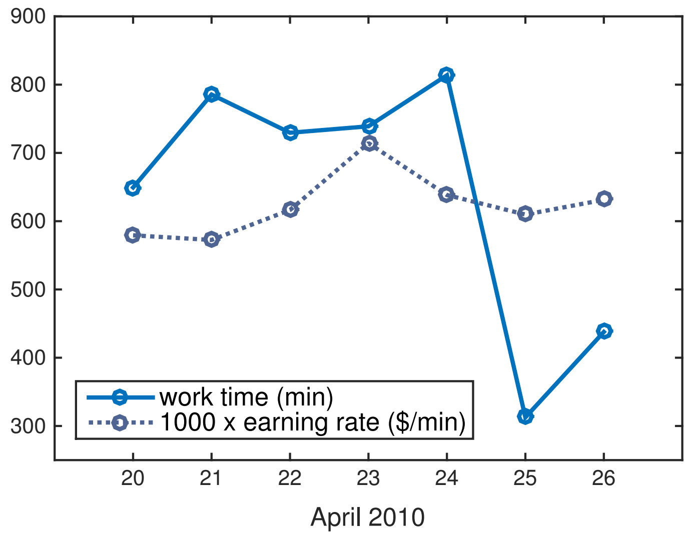


Figure 6.4: Work time and earning rate of the driver with ID No. 2010001271 for the week 20-26 April 2010

The transition probabilities of the reference points of the driver are also calculated in the EM algorithm. The estimated probability transition matrix, A, where A_{ij} denotes the probability of going from reference point i to reference point j, is given in Figure 6.5.

The transition probability matrix provides a means to predict the behavior of the driver. For example, we observe from the 7th and 8th rows of the estimated matrix that if the driver had a large reference point for daily earning on a day and planned not to work for long on that day, then she expected to work for long on the next day.

Table 6.4: Estimated reference points of the driver

Date	Reference point
20 April	(r_{1M}, r_{2M})
21 April	(r_{1H}, r_{2H})
22 April	(r_{1M}, r_{2H})
23 April	(r_{1H}, r_{2H})
24 April	(r_{1H}, r_{2H})
25 April	(r_{1L}, r_{2L})
26 April	(r_{1L}, r_{2M})

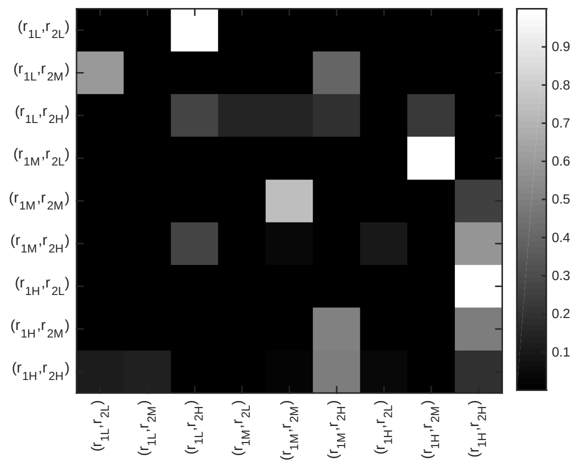


Figure 6.5: The estimate of the probability transition matrix

6.7 Conclusion

We introduced a specific class of decision problems and analyzed the relation between the optimal actions of a person for these problems and her value function. Using this relation, we built a hidden Markov model with the reference point of the person as the hidden state and the observed actions as the output of the model. Then we estimated the value function and the reference points of the person along with their transition probabilities using expectation-maximization. We tested the suggested method on the data set of NYC taxi drivers. We observed that the estimated parameters were able to explain and give insight about the behavior of the drivers.

Given a sequential decision problem, reference point of a person could also depend on the outcome of her previous decisions in addition to her last reference point. Using an input/output hidden Markov model to include these dependencies and the effect of external factors is a future direction of research.

Chapter 7

Conclusion and Future Directions

Training machine learning models with iterative optimization methods, such as the gradient descent algorithm and its variants, creates a closed-loop dynamical system, as demonstrated in Figure 7.1. In this dissertation, we relied on this fact to establish a relationship between the optimization and robustness of machine learning models trained with the gradient descent algorithm, and the classical tools in control theory and dynamical systems.

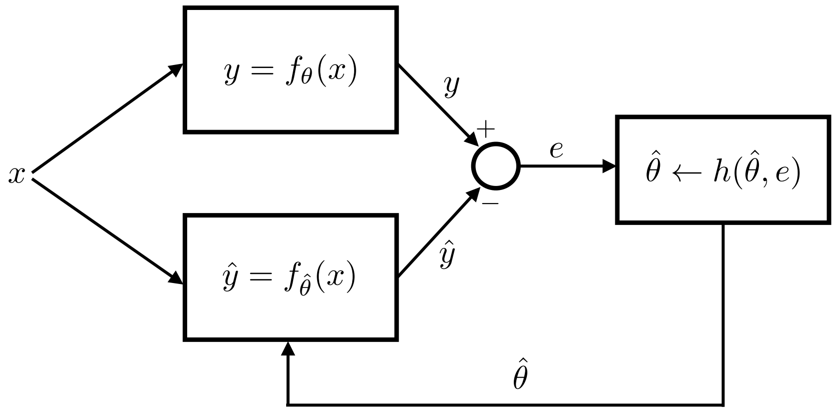

$y = f_\theta(x)$
y
$+$
e
$\hat{\theta} \leftarrow h(\hat{\theta}, e)$
x
$-$
$\hat{y} = f_{\hat{\theta}}(x)$
$\hat{y}$
$\hat{\theta}$

Figure 7.1: Closed-loop dynamical system created by the iterative optimization algorithms employed during training of a machine learning model.

We used Lyapunov analysis to study the dynamics of the gradient descent algorithm when it is being used for nonconvex optimization with multiple local optima. We showed the effect of learning rate of the gradient descent algorithm on the solutions when training multi-layer models. We demonstrated that the optimization algorithm itself limited the class of functions that could be estimated by the multi-layer structure, thereby introducing an implicit regularization. We also showed that keeping every layer close to the identity operation facilitated the convergence of the gradient descent algorithm.

We studied robustness of neural networks with the concepts in system identification and adaptive control literature, namely, sufficient richness of inputs injected into the models and persistent excitation of the estimated parameters during training. We showed that robust estimation of parameters in multi-layer models required not only the richness of the training data but also the richness of the hidden-layer activations, and we derived necessary and sufficient richness conditions for these signals.

We showed that neural networks naively-trained for image classification tasks failed to satisfy the richness requirements in their hidden-layer activations. To find a remedy, we studied the classical methods of regularization for single-layer linear models in terms of the richness of the training data. We demonstrated that penalizing the norms of the parameters in a linear model was equivalent to training the model under random or adversarial perturbations. In models with high-dimensional input space and large number of parameters, however, mere random perturbations would not be able to provide an effective regularization in practice. For this reason, we introduced an algorithm that would provide the persistent excitation for the parameters of a neural network by injecting adversarial perturbation into its each layer.

The training algorithm with persistent excitation further displayed the connection between the training of neural networks and the identification of dynamical systems. In system identification, it was well-known that identifying the unknown parameters in the system required injecting sinusoidal signals into the system; in other words, maintaining the perturbations of the parameters as long as the system operated, as demonstrated in Figure 7.2. Similarly, the training algorithm with persistent excitation provided a mechanism to inject sustained perturbations into all layers of the neural network as long as the training procedure continued, as demonstrated in Figure 7.3.

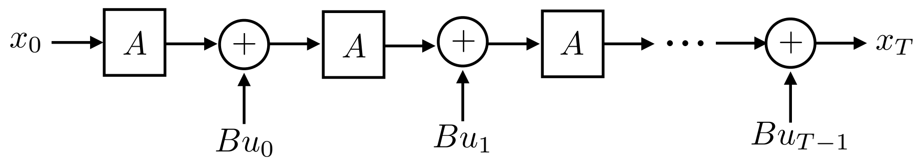


Figure 7.2: Exogenous perturbations injected into the dynamical system during identification of its unknown parameters.

Logistic regression is one of the most common methods for classification in machine learning. By analyzing the dynamics of the gradient descent algorithm on the cross-entropy loss function, we demonstrated that the removal of correctly classified points from the training dynamics exponentially quickly had in fact deleterious effects on the robustness of the classifier obtained. This effect was particularly explicit when the training data were low-dimensional.

We also studied learning dynamical systems with the gradient descent algorithm. We showed that a common loss function such as the squared-error loss would not be able to discover stable modes of a stable system. We revealed that for unstable systems the influence of observations

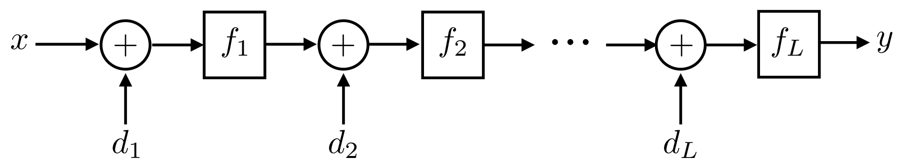


Figure 7.3: Exogenous perturbations injected into the layers of a neural network during training of its parameters.

collected at different times would be substantially different on the dynamics of the gradient descent algorithm. We showed that bringing the observations into a risk-sensitive form could stabilize the gradient descent algorithm and render learning unstable dynamical systems also possible.

As one of the risk-sensitive decision models, we studied the prospect theory in a dynamical context. We introduced a hidden Markov model to express the transitions in the reference frames of a decision maker when they are given a choice problem sequentially. We demonstrated the ability of the model introduced in expressing the decisions of New York City taxi drivers about when they would stop driving in each day based on their daily earnings.

Our Lyapunov analyses in most chapters relied on the deterministic dynamics of the full-batch gradient descent algorithm. The deterministic dynamics allowed us to use a non-vanishing learning rate while studying the convergence of the gradient descent algorithm, and this allowed the relationship between the learning rate and the class of functions learned by the algorithm to manifest itself. Nevertheless, the deterministic algorithms are neither practical for most applications, nor do they yield better solutions. Instead, stochastic algorithms such as the mini-batch gradient descent algorithm are preferred in most applications. Analysis of these stochastic algorithms with Lyapunov stability for stochastic dynamical processes (Kushner, 1965, 1972) remains an open direction for future research.

We provided a reinterpretation for regularization by showing the equivalence of penalizing the norms of the parameters and introducing random or adversarial perturbations into the training data. As adding random perturbations would not be practical in high-dimensional spaces with large number of parameters, we introduced a training algorithm that computed adversarial perturbations during the training procedure, which led to a bilevel optimization problem. Designing methods or alternative loss functions to improve the stability of this bilevel optimization problem is another important direction for research.

We showed that the cross-entropy loss function combined with the soft-max function can lead to extremely suboptimal solutions for classification tasks. This was mainly caused by the soft-max function. Similar to the fact that decaying the learning rate in stochastic gradient methods and the exploration rate in reinforcement learning algorithms is not suitable, removing the correctly classified points from the dynamics of the gradient descent algorithm was not appropriate. Designing alternative functions to convert the Euclidian space into a probability simplex and analyzing their effect on robustness is an open problem.

We analyzed the dynamics of the gradient descent algorithm while learning deterministic linear dynamical systems, and we showed that the stable and unstable modes of the system affect the dynamics of the algorithm substantially differently. Whether introducing randomness into the state transitions helps or exacerbates the problem of instability of the gradient descent algorithm remains an open problem. When the state transitions are nonlinear, the required change in the time-weighting and the specific shape of the risk-sensitive warping function for the observations of the system is also an open direction for future research.

Bibliography

A. Alaeddini, S. Alemzadeh, A. Mesbahi, and M. Mesbahi. Linear model regression on time-series data: non-asymptotic error bounds and applications. In *IEEE Conference on Decision and Control*, pages 2259–2264, 2018.

Evan Archer, Il Memming Park, Lars Buesing, John Cunningham, and Liam Paninski. Black box variational inference for state space models. *arXiv preprint arXiv:1511.07367*, 2015.

Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. In *Advances in Neural Information Processing Systems*, pages 7411–7422, 2019.

Karl Johan Åström and Peter Eykhoff. System identification — a survey. *Automatica*, 7(2):123–162, 1971.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

Erel Avineri and Piet HL Bovy. Identification of parameters for a prospect theory model for travel choice analysis. *Transportation Research Record*, 2082(1):141–147, 2008.

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

Nikita E Barabanov and Danil V Prokhorov. Stability analysis of discrete-time recurrent neural networks. *IEEE Transactions on Neural Networks*, 13(2):292–303, 2002.

Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249, 2017.

Peter L Bartlett, Steven N Evans, and Philip M Long. Representing smooth functions as compositions of near-identity functions with implications for deep network optimization. *arXiv preprint arXiv:1804.05012*, 2018a.

Peter L Bartlett, Dave Helmbold, and Philip Long. Gradient descent with identity initialization efficiently learns positive definite linear transformations by deep residual networks. In *International Conference on Machine Learning*, pages 521–530, 2018b.

Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999.

Dimitris Bertsimas, David B Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

Stephen Boyd and Shankar Sastry. On parameter convergence in adaptive control. *Systems & Control Letters*, 3(6):311–319, 1983.

Stephen Boyd and Shankar Sastry. Necessary and sufficient conditions for parameter convergence in adaptive control. *Automatica*, 22(6):629–639, 1986.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

Frank M. Callier and Charles A. Desoer. *Linear System Theory*. Springer-Verlag, 1991.

Colin Camerer, Linda Babcock, George Loewenstein, and Richard Thaler. Labor supply of new york city cabdrivers: One day at a time. *The Quarterly Journal of Economics*, 112(2):407–441, 1997.

Urszula Chajewska, Daphne Koller, and Dirk Ormoneit. Learning an agent's utility function by observing behavior. In *International Conference on Machine Learning*, pages 35–42, 2001.

Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8(1):6085, 2018.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

Vincent P Crawford and Juanjuan Meng. New york city cab drivers' labor supply revisited: Reference-dependent preferences with rational-expectations targets for hours and income. *American Economic Review*, 101(5):1912–32, 2011.

Christian Daniel, Jonathan Taylor, and Sebastian Nowozin. Learning step size controllers for robust neural network training. In *AAAI Conference on Artificial Intelligence*, 2016.

Brian Donovan and Daniel B Work. Using coarse gps data to quantify city-scale transportation system resilience to extreme events. *arXiv preprint arXiv:1507.06011*, 2015.

Brian Donovan and DB Work. New york city taxi data (2010-2013), 2014. URL `http://dx.doi.org/10.13012/J8PN93H8`.

Simon S Du, Wei Hu, and Jason D Lee. Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced. In *Advances in Neural Information Processing Systems*, pages 384–395, 2018.

Lea Duncker, Gergo Bohner, Julien Boussard, and Maneesh Sahani. Learning interpretable continuous-time models of latent stochastic dynamical systems. In *International Conference on Machine Learning*, volume 97, pages 1726–1734, 2019.

Laurent El Ghaoui and Hervé Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1035–1064, 1997.

Stefanos Eleftheriadis, Tom Nicholson, Marc Deisenroth, and James Hensman. Identification of gaussian process state space models. In *Advances in Neural Information Processing Systems*, pages 5309–5319, 2017.

Henry S Farber. Reference-dependent preferences and labor supply: The case of new york city taxi drivers. *American Economic Review*, 98(3):1069–82, 2008.

A. Fawzi, S. Moosavi-Dezfooli, and P. Frossard. The robustness of deep networks: A geometrical perspective. *IEEE Signal Processing Magazine*, 34(6):50–62, Nov 2017.

Roger Frigola, Yutian Chen, and Carl Edward Rasmussen. Variational gaussian process state-space models. In *Advances in Neural Information Processing Systems*, pages 3680–3688, 2014.

Gartheeban Ganeshapillai, John Guttag, and Andrew Lo. Learning connections in financial time series. In *International Conference on Machine Learning*, pages 109–117, 2013.

Song Gao, Emma Frejinger, and Moshe Ben-Akiva. Adaptive route choices in risky traffic networks: A prospect theory approach. *Transportation Research Part C: Emerging Technologies*, 18(5): 727–740, 2010.

Gauthier Gidel, Francis Bach, and Simon Lacoste-Julien. Implicit regularization of discrete gradient dynamics in linear neural networks. In *Advances in Neural Information Processing Systems*, pages 3196–3206, 2019.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

Karol Gregor, George Papamakarios, Frederic Besse, Lars Buesing, and Theophane Weber. Temporal difference variational auto-encoder. In *International Conference on Learning Representations*, 2019.

Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, pages 6151–6159, 2017.

Suriya Gunasekar, Jason D Lee, Daniel Soudry, and Nati Srebro. Implicit bias of gradient descent on linear convolutional networks. In *Advances in Neural Information Processing Systems*, pages 9461–9471, 2018.

Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *arXiv preprint arXiv:1611.04231*, 2016.

Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240*, 2015.

Moritz Hardt, Tengyu Ma, and Benjamin Recht. Gradient descent learns linear dynamical systems. *Journal of Machine Learning Research*, 19(29):1–44, 2018.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2nd edition, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Guotao Hu, Aruna Sivakumar, and John W Polak. Modelling travellers' risky choice in a revealed preference context: A comparison of EUT and non-EUT approaches. *Transportation*, 39(4): 825–841, 2012.

Kosuke Ishibashi, Kohei Hatano, and Masayuki Takeda. Online learning of approximate maximum p-norm margin classifiers with bias. In *Conference on Learning Theory*, 2008.

Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

Ziwei Ji and Matus Telgarsky. Risk and parameter convergence of logistic regression. *CoRR*, abs/1803.07300, 2018.

Ziwei Ji and Matus Telgarsky. Gradient descent aligns the layers of deep linear networks. In *International Conference on Learning Representations*, 2019.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–291, 1979.

Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in neural information processing systems*, pages 586–594, 2016.

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks*, 11(1):124–136, 2000.

Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 2nd edition, 1996.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Botond Kőszegi and Matthew Rabin. A model of reference-dependent preferences. *The Quarterly Journal of Economics*, 121(4):1133–1165, 2006.

Rahul G. Krishnan, Uri Shalit, and David Sontag. Structured inference networks for nonlinear state space models. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 2101–2109, 2017.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

P. R. Kumar and Pravin Varaiya. *Stochastic Systems: Estimation, Identification and Adaptive Control*. Prentice-Hall, Upper Saddle River, NJ, USA, 1986.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

Harold J Kushner. On the stability of stochastic dynamical systems. *Proceedings of the National Academy of Sciences of the United States of America*, 53(1):8, 1965.

Harold J Kushner. Stochastic stability. In *Stability of stochastic dynamical systems*, pages 97–124. Springer, 1972.

Nathan O Lambert, Daniel S Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer SJ Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

George D Magoulas, Michael N Vrahatis, and George S Androulakis. Effective backpropagation training with variable stepsize. *Neural Networks*, 10(1):69–82, 1997.

Charles H. Martin and Michael W. Mahoney. Traditional and heavy tailed self regularization in neural network models. In *International Conference on Machine Learning*, 2019.

Z. Marzi, S. Gopalakrishnan, U. Madhow, and R. Pedarsani. Sparsity-based Defense against Adversarial Attacks on Linear Classifiers. *ArXiv e-prints*, 2018.

Kiyotoshi Matsuoka. Stability conditions for nonlinear continuous neural networks with asymmetric connection weights. *Neural networks*, 5(3):495–500, 1992.

Anthony N Michel, Jay A Farrell, and Wolfgang Porod. Stability results for neural networks. In *Neural Information Processing Systems*, pages 554–563, 1988.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 86–94, 2017.

M. Shpigel Nacson, J. Lee, S. Gunasekar, P. H. P. Savarese, N. Srebro, and D. Soudry. Convergence of Gradient Descent on Separable Data. *ArXiv e-prints*, 2018a.

M. Shpigel Nacson, N. Srebro, and D. Soudry. Stochastic Gradient Descent on Separable Data: Exact Convergence with a Fixed Learning Rate. *ArXiv e-prints*, 2018b.

Kamil Nar and S Shankar Sastry. Persistency of excitation for robustness of neural networks. *arXiv preprint arXiv:1911.01043*, 2019.

Kamil Nar and S Shankar Sastry. Richness of training data does not suffice: Robustness of neural networks requires richness of hidden-layer activations. *Workshop on Uncertainty and Robustness in Deep Learning, International Conference on Machine Learning*, 2020.

Kamil Nar and Shankar Sastry. Step size matters in deep learning. In *Advances in Neural Information Processing Systems*, pages 3436–3444, 2018a.

Kamil Nar and Shankar Sastry. Residual networks: Lyapunov stability and convex decomposition. *arXiv preprint arXiv:1803.08203*, 2018b.

Kamil Nar, Lillian J Ratliff, and Shankar Sastry. Learning prospect theory value function and reference point of a sequential decision maker. In *IEEE Conference on Decision and Control*, pages 5770–5775, 2017.

Kamil Nar, Orhan Ocal, S Shankar Sastry, and Kannan Ramchandran. Cross-entropy loss and low-rank features have responsibility for adversarial examples. *arXiv preprint arXiv:1901.08360*, 2019a.

Kamil Nar, Orhan Ocal, S. Shankar Sastry, and Kannan Ramchandran. Cross-entropy loss leads to poor margins. *Openreview*, 2019b.

Kamil Nar, Yuan Xue, and Andrew M Dai. Learning unstable dynamical systems with time-weighted logarithmic loss. *arXiv preprint arXiv:2007.05189*, 2020.

Behnam Neyshabur, Ryota Tomioka, Ruslan Salakhutdinov, and Nathan Srebro. Geometry of optimization and implicit regularization in deep learning. *arXiv preprint arXiv:1705.03071*, 2017.

Andrew Ng and Stuart J Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017.

Michal Rolinek and Georg Martius. L4: Practical loss-based stepsize adaptation for deep learning. In *Advances in Neural Information Processing Systems*, pages 6433–6443, 2018.

Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent ODEs for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, 2019.

Ariel Rubinstein. *Lecture Notes in Microeconomic Theory: The Economic Agent*. Princeton University Press, 2012.

Tuhin Sarkar and Alexander Rakhlin. Near optimal finite time identification of arbitrary linear dynamical systems. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 5610–5618, 2019.

S Shankar Sastry. Model-reference adaptive control – stability, parameter convergence, and robustness. *IMA Journal of Mathematical Control and Information*, 1(1):27–66, 1984.

Shankar Sastry. *Nonlinear Systems: Analysis, Stability, and Control*. Springer-Verlag, 2013.

Shankar Sastry and Marc Bodson. *Adaptive Control: Stability, Convergence and Robustness*. Prentice Hall, 1989.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In *Advances in Neural Information Processing Systems*, pages 5014–5026, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018.

D. Soudry, E. Hoffer, M. Shpigel Nacson, S. Gunasekar, and N. Srebro. The Implicit Bias of Gradient Descent on Separable Data. *ArXiv e-prints*, 2018.

Daniel Soudry, Elad Hoffer, and Nathan Srebro. The implicit bias of gradient descent on separable data. In *International Conference on Learning Representations*, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

Håkan Terelius and Karl Henrik Johansson. An efficiency measure for road transportation networks with application to two case studies. In *54th IEEE Conference on Decision and Control*, pages 5149–5155, 2015.

Ruey S Tsay. Financial time series. *Wiley StatsRef: Statistics Reference Online*, 2014.

Amos Tversky and Daniel Kahneman. Loss aversion in riskless choice: A reference-dependent model. *The Quarterly Journal of Economics*, 106(4):1039–1061, 1991.

Amos Tversky and Daniel Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and Uncertainty*, 5(4):297–323, 1992.

Colin Wei and Tengyu Ma. Improved sample complexities for deep neural networks and robust classification via an all-layer margin. In *International Conference on Learning Representations*, 2020.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. In *IEEE International Conference on Robotics and Automation*, pages 528–535, 2016.

Lizhen Zhou, Shiquan Zhong, Shoufeng Ma, and Ning Jia. Prospect theory based estimation of drivers' risk attitudes in route choice behaviors. *Accident Analysis and Prevention*, 73:1–11, 2014.