Information-Theoretic Perspectives on Generalization and Robustness of Neural Networks

by

Adrian Tovar Lopez

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

(Mathematics)

at the

UNIVERSITY OF WISCONSIN-MADISON

2022

Date of Final Oral Exam: 1/13/2022

The dissertation is approved by the following members of the Final Oral Committee:

Varun Jog, Assistant Professor, Electrical and Computer Engineering
Qin Li, Associate professor, Mathematics
Kangwook Lee, Assistant Professor, Electrical and Computer Engineering
David F. Anderson, Professor, Mathematics

Information-Theoretic Perspectives on Generalization and Robustness of Neural Networks

Adrian Tovar Lopez

Abstract

Neural networks as efficient as they are in practice, remain in several aspects still a mystery. Some of the most studied questions are: where does their generalization capabilities come from? What are the reason behind the existence of adversarial examples? In this thesis I use a formal mathematical representation of neural networks to investigate this questions. I also develop new algorithms based on the theory developed.

The first par of the thesis is concerned with the generalization error which characterizes the gap between an algorithm's performance on test data versus performance on training data. I derive upper bounds on the generalization error in terms of a certain Wasserstein distance involving the distributions of input and the output under the assumption of a Lipschitz continuous loss function. Unlike mutual information-based bounds, these new bounds are useful for algorithms such as stochastic gradient descent. Moreover, I show that in some natural cases these bounds are tighter than mutual information-based bounds.

In the second part of the thesis I study manifold learning. The goal is to learn a manifold that captures the inherent low-dimensionality of high-dimensional data. I present a novel training procedure to learn manifolds using neural networks. Parametrizing the manifold via a neural network with a low-dimensional input and a high-dimensional output. During training, I calculate the distance between the training data points and the manifold via a geometric projection and update the network weights so that this distance diminishes. The learned manifold is seen to interpolate the training data, analogous to autoencoders. Experiments show that the procedure leads to lower reconstruction errors for noisy inputs, and higher adversarial accuracy when used in manifold defense methods than those of autoencoders.

In the final part of the thesis I propose an information bottleneck principle for causal time-series prediction. I develop variational bounds on the information bottleneck objective function that can be efficiently optimized using recurrent neural networks. Then implement an algorithm on simulated data as well as real-world weather-prediction and stock market-prediction datasets and show that these problems can be successfully solved using the new information bottleneck principle.

Dedication

To my friends Annette, Arturo and Ashna for all their unconditional support. And because their joy of life made Madison a great place to live.

Acknowledgements

I want to thank my advisor Varun for all the invaluable advice and guidance. And Professor Kangwook for all his help.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Neural networks have revolutionized the way we analyze problems presenting a high degree of non linearity. In practice these algorithms perform tasks that go from image recognition, self driving, chess playing, stock market prediction among many others (see Krizhevsky, Ilya Sutskever, and Hinton 2012; Rao and Frtunikj 2018; Silver et al. 2017; Vargas, De Lima, and Evsukoff 2017). Mathematically speaking this involves finding optimal probability distributions that *best* describes a dataset, interpolating a set of high dimensional points ordered in a non linear submanifold of the parent space, estimation of non linear functions, just to name a few.

In this introduction we will describe in general how these algorithms work and then present some of the most common questions surrounding them. Then we will outline which questions will be addressed in this thesis giving a small summary of each of the chapters.

Although there is a wide variety of neural networks algorithms all of them function under the same principle. Given a task our algorithm will produce a function f defined by a set of parameters w that will perform the desired task. For example if our task is the classification of some images, our algorithm will produce a function f whose input will be the image to classify and its output the image's classification. However in order to produce this function the algorithm has to undergo a training/learning process.

The training process consist of the following:

- We start with a initial function f_0 with parameters w_0, that will learn how to perform the desired task; at this point f_0 doesn't amount to anything more than a function that assigns a random output to its input.

- We define a loss ℓ that measures how well our algorithm is performing the task. For some input x we will measure $\ell(f_0(x))$. Low values of ℓ mean a good performance. In our example low values of $\ell(f_0(x))$ will mean that the image x is classified correctly by f_0.

- We train the algorithm by minimizing $\ell(f_0(x))$ with respect to the parameters that define f_0. This is done by evaluating $\ell(f_0(\cdot))$ on a training dataset containing different values of x and then optimizing via gradient descent. After updating the parameters w with gradient descent the process is repeated until we achieve a function f for which $\ell(f(x))$ is small for *all* the desired values of x.

The reason why these algorithms are so popular is because they are easy to train and give high accuracy. This is despite the fact that the amount of parameters used during the training is huge—sometimes an order of magnitude greater than the amount of training data used—but the network doesn't overfit the training data. In fact these networks generalize really well and perform the task at hand correctly on objects they haven't seen before (see Hardt, Recht, and Y. Singer 2016; Belkin, Ma, and Mandal 2018; Neyshabur, Tomioka, and Srebro 2014), showing that they actually learn the mathematical law governing the task: they learn the manifold where all the data are, or they learn distribution from which the data are generated. Going against the intuition that a over-specified model with too many parameters will just memorize how to perform the task on the examples used to train the algorithm.

Another surprising quality of neural networks is that the function f, that performs the desired task is *almost* linear. It is a composition of linear functions with some mild non linearities mixed in (like the ReLu function). However these functions have a huge capacity to approximate or model high dimensional non linear complex functions (see

Daniely, Frostig, and Yoram Singer 2016; Daniely 2017; Eldan and Shamir 2016; Telgarsky 2015).

Other kind of question about neural networks arise when comparing them with actual human neural networks. One example is the existence of adversarial examples. As a concrete example consider adversarial examples in an image recognition task. The adversarial example $\hat{x}$ is generated from a original image x. For a human x and $\hat{x}$ are without a doubt in the same class, and in some situations they are even indistinguishable from one another. However for the network $\hat{x}$ is going to be misclassified. Why do such examples exist? What is fundamentally different between a human neural network and an artificial neural network that produces such phenomena? What is more intriguing is that adversarial examples can be found everywhere, nearly any neural network will have its set of adversarial examples (see Papernot and McDaniel 2018; Ilyas et al. 2017; Gilmer et al. 2018).

Although some of these questions had been studied in recent years a lot of questions on how neural networks work still remain. It is also worth noticing that a big part of the existing literature focuses more on developing new and more efficient algorithms with either mathematical justification guaranteeing that in practice during training, the loss function will be minimized, or experimental evidence that show the effectiveness of the new algorithm. Yet the focus on understanding how and why these algorithms work is sometimes overlooked.

In this thesis we focus on using mathematical techniques not standard in the study of neural networks to understand the underlying reason of why these algorithms work.

In chapter 2 we look at the problem of generalization. The generalization error measures how good is the performance of an algorithm in data that was not used for its training. We follow the work of M. Raginsky et al. (2016) in which they use the mutual information between the data used for training and the parameters of the trained algorithm to upper bound the generalization error. This sheds light on why these algorithms generalize well: regardless of how the algorithm is constructed, under some assumption it

can be shown that the generalization error can't go above a certain point. However the choice of mutual information by M. Raginsky et al. (2016) imposes artificial conditions on the data and produces a non sharp bound. We propose a different *measure* of information: the Wasserstein distance. We are able to produce an upper bound of the generalization error that in some cases is sharper than the bound by M. Raginsky et al. (2016) and that requires a less restrictive and more natural condition on the data.

One thing that can be observed from this chapter is that neither the mutual information nor the Wasserstein distance are "the" measure to understand how neural networks process information. Both are options with advantages and disadvantages coming from other fields that seem to capture at some level the way these algorithms are processing information. There is no a priori reason that tells us that these are the correct measures to use when studying neural networks. However, since some results can be obtained from them it is a good idea to follow this line of research but keeping in mind the the goal of finding precisely how information is processed and may be coming with a measure for information that is intrinsic of these algorithms.

In chapter 3 we tackle the adversarial problem. There had been several defenses (see Pouya Samangouei, Maya Kabkab, and Rama Chellappa 2018; Athalye, Carlini, and Wagner 2018; Ilyas et al. 2017) that try to produce or modify algorithms in which there are no adversarial examples. Rather than coming with a new defense our goal is to try to find the simplest situation in which adversarial examples stop happening so may be we can understand where they come from. We start with a neural network N_1 that performs a classification task for which there are adversarial examples. Then we construct a data manifold by interpolating the training data using a neural network N_2. Once this manifold is constructed, adversarial examples are geometrically projected on it. We show that the projected examples stop being adversarial as the network N_1 classifies them correctly. The surprising thing is that the construction of the data manifold is completely independent from the classification network. There seem to be something natural about how the data are arranged that precludes adversarial examples, and it is the classification network that

introduces those examples. This is perhaps because its generalization capabilities makes it consider inputs in a bigger space than it should.

Finally chapter 4 is concerned with the deep information bottleneck Alemi et al. (2016). A neural network algorithm in which the loss function uses mutual information which ensures that relevant mutual information from the input will be used to perform the task at hand while irrelevant information will be discarded. We apply this information bottleneck idea to time series data that is causally related and we use recurrent neural networks. We observe that even though we are limiting ourselves to the use of mutual information there is still freedom in how to design our loss function. We explore and compare our results with the ones in D. Xu and Fekri (2018) and produce an algorithm that has good predictive power. Once again running into the idea that even though there is no one correct measure of information to study how neural networks work, mutual information seems to be a good guide.

The proofs of theorems and lemmas can be found in the appendix.

Chapter 2

Generalization error bounds using Wasserstein distance

2.1 Introduction

Learning algorithms need to be engineered to learn as much as possible from the available data while avoiding over-fitting. The degree to which an algorithm overfits is measured using the *generalization error*, which measures the gap in performance between the training set and the test set. Understanding and upper-bounding generalization error is an important topic in learning theory and several different approaches have been proposed in the literature. A popular approach seeks to bound the generalization error in terms of the diversity of the function class used for optimization, which is measured via quantities such as the VC-dimension, or Rademacher complexity (see Vapnik 1998; Shalev-Shwartz and Ben-David 2014). More relevant to us, however, are notions of algorithmic stability (see Bousquet and Elisseeff 2002; Elisseeff, Evgeniou, and Pontil 2005; Mukherjee et al. 2006; M. Raginsky et al. 2016) wherein the ability of an algorithm to generalize well is related to its stability; here stability is understood as the robustness of the algorithm to certain kinds of perturbations. Recently, this idea has been used to analyze specific algorithms such as the stochastic gradient descent (SGD) (see Hardt, Recht, and Y. Singer 2016;

London 2016) and stochastic gradient Langevin dynamics (SGLD) (see Mou et al. 2017).

In recent years, there have been efforts to obtain generalization error bounds for data-dependent notions of stability. We mention in particular the works of Russo & Zou (2016) and Xu & Raginsky 2017. In these, a learning problem is cast into an information-theoretic framework: The algorithm acts as a channel that (noisily) maps input data S into the algorithm output W. In A. Xu and M. Raginsky (2017), the authors defined a notion of information-theoretic stability in terms of the mutual information $I(W;S)$ between the algorithm input S and output W. When the loss function is sub-Gaussian with respect to the data distribution, the authors derived bounds on the generalization error in terms of the information-theoretic stability of the algorithm. This new notion of stability differs from traditional notions of algorithmic stability, namely, stability is no longer defined in terms of data perturbations. Furthermore, information-theoretic stability is data-dependent (and not just algorithm dependent) as the data-distribution influences the sub-Gaussian parameter of the loss function, which in turn affects stability.

Mutual information satisfies a number of important properties, in particular the data-processing inequality and chain rule, and this makes information-theoretic stability a useful notion for several applications (see Ankit Pensia, Varun Jog, and P.-L. Loh 2018; Bassily et al. 2018). However, using mutual information to define stability has certain drawbacks. A crucial one is that mutual information may be infinite even in benign settings that arise commonly in learning or optimization algorithms. In such cases, we are unable to get any sensible bounds on the generalization error using mutual information. (We note that recent work of Asadi, Abbe, and Verdú (2018) has proposed techniques that lead to useful bounds in such cases as well.) We give two examples to illustrate this phenomenon.

Example 2.1. *In linear regression, the goal is to fit a linear model for a given data matrix X and observation vector y; i.e., $y = X\hat{\beta}$. Using the least squares loss function yields $\hat{\beta} = (X^T X)^{-1} X^T y$. In this example, the algorithm output $W = \hat{\beta}$ is a deterministic function of the algorithm input $S = \{(X_i, y_i) \mid 1 \leq i \leq n\}$. Assuming S has a well-defined differential entropy, it is easy to verify that $I(S;W) = +\infty$. This shows that a mutual information-based upper bound will not be useful in bounding the generalization error of linear regression.*

Example 2.2. *Consider the empirical risk minimization problem with a loss function $\ell : (\mathcal{X}, \mathcal{W}) \to \mathbb{R}$. Given a dataset $S = (X_1, \ldots, X_n)$, the goal is to minimize the empirical loss $\sum_{i=1}^{n} \ell(X_i, w)$. A common algorithm is to use gradient descent starting from some fixed point. However, it is easily seen that in this case, the estimate W is a deterministic function of S. One may use stochastic gradient descent (SGD) to introduce randomness, but even then W may only take countably many values; i.e., the distribution of W conditioned on S has atoms. In such a situation, the mutual information $I(W;S)$ is again equal to $+\infty$.*

Yet another difficulty with using mutual information-based bounds is checking the assumption of sub-Gaussianity of the loss function with respect to the data distribution, which is unknown.

For these reasons, in this chapter we explore a different approach towards bounding the generalization error. We try to capture the intuition that an algorithm generalizes well if the output W does not contain too much information about S by using a different metric of "information". This new metric is defined using Wasserstein distances from optimal transport theory (see Villani 2003) which does not display the shortcomings in Examples 1 and 2. Furthermore, the only assumption we make on the loss function is that it is Lipschitz—which is easy to verify compared to sub-Gaussianity, as illustrated in the following example:

Example 2.3. *A loss function $\ell(x, w)$ depends on the data x and the hypothesis w. In several learning algorithms we predict an outcome as a function $f_w(x)$ of the data, where f_w depends on the parameters w. In such a situation, the loss function is simply $\ell(x, w) = \bar{\ell}(f_w(x))$ for some function $\bar{\ell}$. If f_w is Lipschitz on $\mathcal{X}$ for all $w \in \mathcal{W}$, and if $\bar{\ell}$ is also Lipschitz, then $\ell(x, w)$ is also Lipschitz. Assuming f_w is Lipschitz, $\bar{\ell}$ may be the mean absolute error (MAE) or Huber loss (see Vapnik 1998). If $\bar{\ell}$ is the mean square error (MSE), then ℓ is Lipschitz as long as $\mathcal{Z}$ is compact and f_w is continuous, and if $\bar{\ell}$ is the cross-entropy function, then ℓ is Lipschitz as long as the image of f_w lies in a compact subset of $(0, 1)$.*

The outline of the chapter is as follows: In Section 2, we define the generalization error and the Wasserstein metric. In Section 3 we state and prove the main results of this chapter. In Section 4, we compare our bounds to the mutual information-based bounds and show that in some natural cases, our bound is tighter. Finally, in Section 5 we conclude with some open problems and some speculations.

2.2 Problem setting

2.2.1 Learning Algorithms

Let $\mathcal{X}$ be the sample space. A data set S consists of n data points sampled i.i.d. from an unknown distribution $\mathbb{P}_X$ over $\mathcal{X}$. Thus, S will be a random element of the space:

$$S \in \mathcal{S} = \underbrace{\mathcal{X} \times \mathcal{X} \times \ldots \times \mathcal{X}}_{n \text{ times}} \tag{2.1}$$

We may denote the distribution on S by

$$\mathbb{P}_S := \underbrace{\mathbb{P}_X \times \mathbb{P}_X \times \ldots \times \mathbb{P}_X}_{n \text{ times}} := \mathbb{P}_X^{\otimes n}. \tag{2.2}$$

Let $(\mathcal{W}, \sigma_{\mathcal{W}})$ be the parameter space endowed with an appropriate sigma algebra. As

per the framework in Russo and Zou (2016) and A. Xu and M. Raginsky (2017), an algorithm is defined as a random map from $\mathcal{S}$ to $\mathcal{W}$. Equivalently, an algorithm can be thought of as a Markov kernel $\mathbb{P}_{W|S} : \sigma_{\mathcal{W}} \times \mathcal{S} \to \mathbb{R}$ satisfying that for each $S \in \mathcal{S}$ then $\mathbb{P}_{W|S=s}(\cdot)$ is a probability measure on $\mathcal{W}$. Observe that the distributions $\mathbb{P}_{W|S=s}$ alongside $\mathbb{P}_S$ will induce a distribution on $\mathcal{W} \times \mathcal{S}$, given by $\mathbb{P}_{W,S} = \mathbb{P}_S \mathbb{P}_{W|S}$.

Denote the loss function by $\ell : \mathcal{X} \times \mathcal{W} \to \mathbb{R}$.

Example 2.4. *Consider a image recognition problem where we want to train a feedforward neural network to identify, from a pool of images of cats and dogs, which ones are cats and which ones are dogs.*

A point in our training data set will consist of an image $\hat{x}_i$ with it's corresponding label y_i: $x_i = (\hat{x}_i, y_i)$. $\mathbb{P}_X$ is the unknown distribution on the x_i's.

$\mathcal{W}$ will be the space of weights and biases of the neural network.

The loss function will be the binary cross-entropy.

The algorithm $\mathbb{P}_{W|S=s}$ is a function that given a fixed training data set s will produce a distribution on $\mathcal{W}$. More specifically lets assume our training process uses stochastic gradient descent (SGD). In this case different training sessions will result in different values of w, even if the training data set is the same. The distribution $\mathbb{P}_{W|S=s}$ gives the probability of getting parameter w when we train with the data set s. This is presented more formally in section 3.2.

We want to design an algorithm $\mathbb{P}_{W|S}$ giving high probabilities to values of W that will make the loss function $\ell(x, w)$ small. Regardless of what data set S was used for training.

More formally we want to find w that minimizes:

$$L_{\mathbb{P}_X}(w) := \int_{\mathcal{X}} \ell(x, w) d\mathbb{P}_X(x) = \mathbb{E}_{\mathbb{P}_X} \ell(X, w).$$

$L_{\mathbb{P}_X}(w)$ is called the population risk.

The problem is that in practice we can't directly compute the population risk since we don't have access to the $\mathbb{P}_X$ distribution.

What we do instead is: first approximate the population risk and try to minimize such approximation. And second try to make sure that the average loss of our algorithm won't be far away from the true minimum of the population risk.

The population risk is approximated by the empirical risk. The empirical risk is defined as follows: for a data set $S = (x_1, \ldots, x_n) \in \mathcal{S}$ and for each $w \in \mathcal{W}$, the empirical risk is:

$$L_S(w) := \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, w).$$

The difference between the average loss of our algorithm and the true minimum of the population risk is called the excess risk and is:

$$\mathbb{E}_{(S,W)\sim\mathbb{P}_S\mathbb{P}_{W|S}}[L_{\mathbb{P}_X}(W)] - L_{\mathbb{P}_X}(w^*),$$

Where $w^* := \arg\min_{w\in\mathcal{W}} \mathbb{E}_{X\sim\mathbb{P}_X}[\ell(X, W)]$ is the true minimizer of the population risk.

Empirical risk can be minimized by different methods. SGD is one of the most efficient ones (see Hardt, Recht, and Y. Singer 2016; London 2016). Here we focus in studying the excess risk. To that end we introduce the generalization error.

The generalization error with respect to the real distribution on the data $\mathbb{P}_S$ and the algorithm $\mathbb{P}_{W|S}$ is then defined to be the expected difference between the population risk and the empirical risk:

$$\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S}) := \mathbb{E}_{(S,W)\sim\mathbb{P}_S\mathbb{P}_{W|S}}[L_{\mathbb{P}_X}(W) - L_S(W)].$$

The excess risk can be expressed as:

$$\mathbb{E}[L_{\mathbb{P}_X}(W)] - L_{\mathbb{P}_X}(w^*) = \mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S}) + \Big(\mathbb{E}[L_S(W)] - L_{\mathbb{P}_X}(w^*)\Big).$$

Where all expectation are with respect to $(S, W) \sim \mathbb{P}_S\mathbb{P}_{W|S}$.

Furthermore, it may be shown (cf. Lemma 5.1 of Hardt, Recht, and Y. Singer (2016)) that $\mathbb{E}_{S\sim\mathbb{P}_S}[L_S(w_S^*)] \leq L_{\mathbb{P}_X}(w^*)$, where $w_S^* := \arg\min_{w\in\mathcal{W}} L_S(w)$ is the true empirical risk

minimizer. Hence, we have the bound

$$\mathbb{E}[L_{\mathbb{P}_X}(W)] - L_{\mathbb{P}_X}(w^*) \leq \left|\mathcal{E}(\mathbb{P}_X, \mathbb{P}_{W|S})\right| + \left|\mathbb{E}[L_S(W) - L_S(w_S^*)]\right|.$$

Where all expectation are with respect to $(S, W) \sim \mathbb{P}_S \mathbb{P}_{W|S}$.

Observe there is two components of the excess risk. The second one, $\left|\mathbb{E}[L_S(W) - L_S(w_S^*)]\right|$, is the error between the true minimum of the empirical risk and the value obtained in practice.

The first term, the generalization error is what we focus in this chapter.

Remark 2.5. *Note that $\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S}) = 0$ means than, in average, the resulting loss of our algorithm on the training data $E_{(S,W)\sim\mathbb{P}_S\mathbb{P}_{W|S}} L_S(W)$, is the same as the average loss in general $E_{(S,W)\sim\mathbb{P}_S\mathbb{P}_{W|S}}[L_{\mathbb{P}_X}(W)]$. Which means that our algorithm preforms similarly in the training data that in general data, in other words it doesn't overfit.*

Before moving on, we present this lemma that let us express the generalization error in a more suitable form for our study:

Lemma 2.6. *Let $f(s, w) := \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, w)$ then:*

$$\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S}) = \int_{\mathcal{S}\times\mathcal{W}} f(s, w) d\mathbb{P}_S(s) d\mathbb{P}_W(w) - \int_{\mathcal{S}\times\mathcal{W}} f(s, w) d\mathbb{P}_{S,W}(s, w) \tag{2.3}$$

2.2.2 Information-theoretic bounds on generalization error

In this section, we give a brief overview of the results in A. Xu and M. Raginsky (2017), where the authors obtained a bound on the generalization error of an algorithm in terms of the mutual information $I(\mathbb{P}_S; \mathbb{P}_W)$. First a definition.

Definition 2.7. *A random variable X is σ-**sub-Gaussian** if there exist $\sigma > 0$ such that for any $\lambda > 0$, the following inequality holds:*

$$\mathbb{E}[\exp(\lambda(X - \mathbb{E}X)] \leq \exp\left(\frac{\sigma^2\lambda^2}{2}\right). \tag{2.4}$$

And now the theorem.

Theorem 2.8. *If $\ell(X, w)$ is a σ-sub-Gaussian random variable for every $w \in \mathcal{W}$, then the following bound holds (Theorem 1 in A. Xu and M. Raginsky (2017)):*

$$\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S}) \leq \sqrt{\frac{2\sigma^2}{n} I(\mathbb{P}_S; \mathbb{P}_W)}. \tag{2.5}$$

This expression indicates if an algorithm has an output W that does not depend too much on the input, then this algorithm will not overfit to the data.

2.2.3 Wasserstein metric

Definition 2.9. *Let $(\mathcal{X}, \sigma_{\mathcal{X}})$ be a measurable space. We denote the space of all probability distributions on $(\mathcal{X}, \sigma_{\mathcal{X}})$ as $\mathbb{P}(\mathcal{X})$. Define $(\bar{\mathcal{X}}, \sigma_{\bar{\mathcal{X}}})$ to be an identical copy of $(\mathcal{X}, \sigma_{\mathcal{X}})$. Consider distributions $\mathbb{P}_1$ and $\mathbb{P}_2$ on $\mathcal{X}$ and $\bar{\mathcal{X}}$ respectively. We denote the subset of distributions in $\mathbb{P}(\mathcal{X} \times \bar{\mathcal{X}})$ that have $\mathbb{P}_1$ as a marginal on $\mathcal{X}$ and $\mathbb{P}_2$ as a marginal on $\bar{\mathcal{X}}$ by $\Pi(\mathbb{P}_1, \mathbb{P}_2)$. We are using $\sigma_{\mathcal{X}} \otimes \sigma_{\bar{\mathcal{X}}}$ as the sigma algebra for $\mathcal{X} \times \bar{\mathcal{X}}$.*

*For $p \geq 1$, the p-**Wasserstein distance** between distributions $\mathbb{P}_1$ and $\mathbb{P}_2$ is defined as:*

$$W_p(\mathbb{P}_1, \mathbb{P}_2) := \inf_{\pi \in \Pi(\mathbb{P}_1, \mathbb{P}_2)} \left(\int_{\mathcal{X} \times \bar{\mathcal{X}}} ||(x - \bar{x})||_p^p \, d\pi(x, \bar{x}) \right)^{\frac{1}{p}}.$$

It can be shown that W_p is a metric on $\mathbb{P}(\mathcal{X})$, and that there is a measure $\pi^* \in \Pi(\mathbb{P}_1, \mathbb{P}_2)$ that attains the infimum on the definition (see Villani 2003).

2.3 Wasserstein bounds on the generalization error

We proceed now to prove a bound on the generalization error in terms of the Wasserstein distance and the Lipchitz constant of the loss function.

Theorem 2.10. *Let $\mathcal{S} = \mathcal{X}^n$ be the sample space and $\mathcal{W}$ be the parameter space. Let $\mathbb{P}_{W|S}$ be an algorithm. and let $\mathbb{P}_{S,W}$ be the distribution on $\mathcal{S} \times \mathcal{W}$ induced by the algorithm and the distribution on the data ($\mathbb{P}_{S,W} = \mathbb{P}_S \mathbb{P}_{W|S}$). If the loss function ℓ is Lipschitz continuous on $\mathcal{X}$ for every $w \in \mathcal{W}$; i.e,*

$$|\ell(\bar{x}, w) - \ell(x, w)| \leq L||\bar{x} - x||_p \quad \forall w \in W$$

Then:

$$|\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S})| \leq \frac{L}{n^{\frac{1}{p}}} \left(\int_{\mathcal{W}} W_p^p(\mathbb{P}_S, \mathbb{P}_{S|w}) d\mathbb{P}_W(w) \right)^{\frac{1}{p}}.$$

It is instructive to compare the form of upper bound in the above theorem to the information-theoretic bound in (2.5). Mutual information $I(\mathbb{P}_S; \mathbb{P}_W)$ is the KL-divergence $D(\mathbb{P}_W \mathbb{P}_S || \mathbb{P}_{S,W})$. This may be equivalently expressed as

$$I(\mathbb{P}_S; \mathbb{P}_W) = \int_{\mathcal{W}} D(\mathbb{P}_{S|W=w} || \mathbb{P}_S) d\mathbb{P}_W(w).$$

Viewing $\mathbb{P}_S$ as the a weighted average of $\mathbb{P}_{S|W=w}$ with weights $\mathbb{P}_W(w)$, we see that mutual information may be thought of as a kind of "variance" in the space of distributions, where instead of the squared distance one uses KL-divergence. In contrast, the result in Theorem 2.10 uses an actual metric; i.e., the Wasserstein distance between $\mathbb{P}_S$ and $\mathbb{P}_{S|W=w}$. Note also that when W is independent of S, the bounds in both cases evaluate to 0. Thus, the right hand term in our upper bound may be thought of some measure of "information" that is capturing how different (on average) the posterior distribution of $\mathbb{P}_{S|W=w}$ is from the prior distribution $\mathbb{P}_S$.

Another observation is the following: Consider a data distribution $\mathbb{P}_S$ and an algorithm

$\mathbb{P}_{W|S}$. For the joint distribution $\mathbb{P}_{W,S}$, we have a forward channel $\mathbb{P}_{W|S}$ and a backward channel $\mathbb{P}_{S|W}$. Mutual information, being symmetric, captures both the forward and backward channels in one expression. However, the Wasserstein distance in Theorem 2.10 is not symmetric and is only in terms of the backward channel $\mathbb{P}_{S|W}$.

2.4 Comparison to information-theoretic bounds

Recall the bound from A. Xu and M. Raginsky (2017) under the assumption of a σ-sub-Gaussian loss function:

$$|\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S})| \leq \sqrt{\frac{2}{n}\sigma^2 I(W;S)}, \tag{2.6}$$

where n is the number of samples used to train the algorithm. It is not possible in general to compare this bound with the bound from Theorem 2.10. However, when the data distribution $\mathbb{P}_X$ satisfies a *transportation-cost inequality* (see Maxim Raginsky, Sason, et al. 2013), then such a comparison is possible as we will show in Theorem 2.12.

Definition 2.11. *A probability measure μ satisfies an L^p transportation-cost inequality with a constant $c > 0$, denoted by $T_p(c)$, if for every distribution ν that is absolutely continuous with respect to μ the following inequality holds:*

$$W_p(\mu, \nu) \leq \sqrt{2cD(\nu||\mu)}, \tag{2.7}$$

where W_p is the p-Wasserstein distance, and $D(\nu||\mu)$ is the KL-divergence of ν with respect to μ.

In particular, a standard normal distribution satisfies $T_2(1)$ transportation cost inequality.

Theorem 2.12. *Assume that $\mathbb{P}_X$ satisfies $T_p(c)$ for some $p \in [1,2]$ and some $c > 0$. Then*

$$\frac{L}{n^{\frac{1}{p}}} \left(\int_W W_p^p(\mathbb{P}_S, \mathbb{P}_{S|w}) d\mathbb{P}_W(w) \right)^{\frac{1}{p}} \leq L\sqrt{\frac{2c}{n} I(\mathbb{P}_S; \mathbb{P}_W)} \tag{2.8}$$

Note that the left hand side is the bound from Theorem 2.10, whereas the right hand side is a scalar multiple of the bound in A. Xu and M. Raginsky (2017).

A special case worth mentioning is when $\mathbb{P}_X$ is isotropic Gaussian, where the following holds:

1. Loss function $\ell(x, w)$ is L-Lipschitz implies $\ell(x, w)$ is L-sub-Gaussian
2. $\mathbb{P}_X$ satisfies a $T_2(1)$ transportation-cost inequality, and thus $\mathbb{P}_S$ satisfies the same transportation-cost inequality

In this setting, our result in Theorem 2.12 states that the Wasserstein distance-based bounds from Theorem 2.10 are better than the information-theoretic bounds in A. Xu and M. Raginsky (2017).

2.5 Conclusions

An important question worth investigating is how can one modify a learning algorithm to ensure that the Wasserstein distance-based bound in Theorem 2.10 is small. For mutual information, it is always possible to add noise to an algorithm to deliberately reduce mutual information and improve generalization behavior. Indeed, noisy optimization algorithms that have been proposed in recent years (see Welling and Teh 2011; Ge et al. 2015; Jin et al. 2017) for escaping from shallow local minima have good generalization properties (see Ankit Pensia, Varun Jog, and P.-L. Loh 2018). Analyzing the bound in Theorem 2.10 for such noisy algorithms and for stochastic gradient descent would be a direction worth exploring. The main challenge is the lack of appropriate notions of data-processing inequalities and chain rules for Wasserstein distance.

Another open problem concerns the coupling π_0 used to prove Theorem 2.10. It is not clear if this is the best possible coupling we can use to obtain sharp bounds on the generalization error. It is possible that additional assumptions on the loss functions, such as joint Lipschitz continuity in the space $\mathcal{W} \times \mathcal{X}$, will require using different optimal couplings.

Lastly, the backward channel interpretation of generalization error suggests a connection to rate distortion theory (see Thomas M Cover and Joy A Thomas 2012). The main hurdle appears to be the assumptions about the loss function—in traditional rate distortion theory, the loss function is assumed to be separable. The loss function $f(s, w)$ for a fixed w is indeed separable in s, and we would like to explore these connections in future work.

Chapter 3

A Geometrical procedure for learning manifolds

3.1 Introduction

In Chapter 2 we introduced the general concept of a learning algorithm. We gave a theoretical bound on the generalization capabilities of such algorithms. Our results are true in general, and in certain conditions we can guarantee that ideally algorithms should have good performances. However in practice there are some obstacles. In this chapter we turn our attention to two well known problems. The first one is that algorithms that seek to classify, estimate or cluster high dimensional data may suffer from the curse of dimensionality, wherein learning becomes exponentially harder. The other problem we address, common in the field of image recognition, is the existence of adversarial images. This are images that can easily fool a classification algorithm but that to the human eye are no different than images that are classified correctly. One of the most common way of attacking both problems is by the use of *Dimensionality reduction.*

Dimensionality reduction is an area of machine learning where high-dimensional data are represented in a lower dimension to enable efficient learning. Of particular interest is manifold learning: a geometric approach toward dimensionality reduction. In manifold

learning, the goal is to learn a manifold—a low-dimensional surface lying within the high-dimensional data space—that contains almost all of the data points. This approach is justified by the belief that data are not truly high-dimensional as they contain numerous commonalities and redundancies which may be stripped away to reduce dimensionality.

Linear manifolds or subspaces may be learned efficiently via principle component analysis (PCA), but the complexities of most modern datasets are not captured by simple linear manifolds. Many methods exist in the literature for non-linear dimensionality reduction (see Smola and Schölkopf 1998; Roweis and Saul 2000; Ghodsi 2006). In recent years, deep learning-based techniques have been remarkable successful in learning complex, non-linear manifolds, especially for image data. Some popular algorithms include autoencoders (AEs), denoising autonecoders (DAEs), variational autoencoders (VAEs) (see Kingma and Welling 2013), and generative adversarial networks (GANs) (see Goodfellow et al. 2014; Vincent et al. 2008). These algorithms have found several applications in computer vision such as data generation, image denoising, and image inpainting. More recently, these algorithms have supplanted traditional LASSO-based methods for compressed sensing (see Bora et al. 2017; Jalali and Yuan 2019).

Although similar in many aspects, there are pros and cons to using each of these methods. For instance, GANs and VAEs are *generative models*; i.e., these may be used to generate new data samples. This is much harder to do for AEs and DAEs. On the other hand, dimensionality reduction is simpler for AEs nad DAEs since the "encoder" provides an explicit map from the high-dimensional data space to the low-dimensional latent space. Such explicit maps are harder to obtain in GANs; recent work in this direction may be found in the references (see Creswell and Bharath 2018; Lipton and Tripathi 2017). Like AEs, VAEs also have an encoder component. But this component learns a kernel (probabilistic map) instead of a single deterministic map.

Manifold learning has also found applications in robust machine learning. Szegedy et al. (2013) observed that deep learning-based image classification algorithms are susceptible to adversarial attacks: the algorithm is likely to make a mistake on an image with a small

but carefully-crafted perturbation added to it. Many ways to craft such perturbations have appeared in the literature as "attacks" on classifiers. Similarly, several methods exist to "defend" classifiers against attacks.

One of the more successful defenses proposed in recent years is the manifold defense algorithm (see Ilyas et al. 2017; P. Samangouei, M. Kabkab, and R. Chellappa 2018). The basic idea of the manifold defense algorithm is simple: first learn the data manifold and then use it as a way to "denoise" adversarially perturbed images example. Denoising is achieved by projecting the adversarial example back to the learned manifold. This defense is based on the observation that adversarially perturbing an image pushes it outside the data manifold. The classifier, which has been trained on clean images on the data manifold, is prone to errors on examples outside the data manifold. The projection step brings the perturbed image back to the manifold where it can be correctly classified.

The manifold defense and generative model-based compressed sensing or image denoising algorithms depend critically on a pre-learned data manifold. This raises the natural question: how does the performance of these algorithms depend on the quality of the learned manifold? For example, in compressed sensing, the learned generator (GAN) contributes an error that does not diminish even with additional linear observations. Similarly, manifold defense algorithms will likely fail if the learned manifold does not match the true data manifold closely. Learning a *better* manifold is therefore crucial to the success of any algorithm that relies on manifold projections.

The current toolset for deep learning-based manifold learning is limited: GANs, autoencoders, and closely related variants. It is desirable to enhance this toolset by proposing entirely new training procedures for manifold learning. These new algorithms may succeed in leaning "better" manifolds, which could substitute those learned by existing methods.

Motivated by the above observation, we propose a novel procedure for manifold learning which we call the *surface algorithm*. A key aspect of the surface algorithm that sets it apart from GANs and autoencoders is a *geometric projection step* during the training procedure. Our contributions may be summarized as follows:

(a) Propose a new training procedure to efficiently learn the data manifold.

(b) Perform experiments implementing the manifold defense. Showing that the learned manifold leads to higher accuracy for the classification network on adversarially manipulated images. As well as a lower reconstruction error for noisy images.

The structure of this chapter is as follows: In section 3.2 we give a formal definition of neural networks that will help us develop our ideas more clearly. In section 3.3 we describe the surface algorithm as weel as AEs and DAEs and its training procedure to find the data manifold. We shall also discuss and contrast our algorithm with GANs and autoencoders. In Section 3.4 we evaluate the performance of the surface algorithm on real and synthetic data. Finally, in Section 3.5 we conclude the paper with some open problems and future research directions.

3.2 Preliminaries: Neural Networks and Manifolds

In this section we give a formal definition of a neural network. We also give a simple definition of manifolds and it's relation with neural networks.

3.2.1 Neural Networks

A feed forward neural network is a family $\mathcal{N}$ of functions that have the following characteristics:

For each function $f \in \mathcal{N}$:

- f goes from $\mathbb{R}^d$ to $\mathbb{R}^n$ for some fixed d and n integers.
- Let $(l_1, l_2, \ldots, l_k)$ be a sequence of positive integers . Let $(W_1, \ldots, W_{k+1}) \in \mathbb{M}_{l_1 \times d} \times \ldots \times \mathbb{M}_{n \times l_k}$ and $(b_1, \ldots, b_{k+1}) \in \mathbb{R}^{l_1} \times \ldots \times \mathbb{R}^{l_k}$ We call this matrices and vectors the weights and biases of f, respectively.
- Let $(\sigma_1, \ldots, \sigma_k)$ scalar functions. We call this functions the *activation functions* of f.

- Let $x \in \mathbb{R}^d$ be the input of f. Consider the function

$$x_1(x) := \sigma_1(W_1 x + b_1)$$

  where σ_1 is applied elementwise. We call x_1 the *layer 1* of f. Each entree of the vector x_1 is called a *unit* of layer 1.

- We define recursively *layer* j as:

$$x_j(x) := \sigma_j(W_j x_{j-1}(x) + b_j)$$

  for $1 < j \leq k$. Each entree of the vector x_j is a *unit* of layer j. Middle layers are called *hidden layers*.

- The output of f is called *output layer* and is defined as:

$$y = W_{k+1} x_k(x) + b_{k+1}$$

  Note the reason of the dimension for the matrices $W_1, \ldots, W_{k+1}$ is so that layer j can be an input for layer $j + 1$.

- We say f has depth k.

All functions belonging to the $\mathcal{N}$ will have the same activation functions, the same depth and each of it's layer will be the same amount of units.

$\mathcal{N}$ is a family of function parameterized by the wights and biases: the difference between two function belonging to $\mathcal{N}$ is the value of it's weight and biases. We denote the space of parameters as:

$$\mathcal{W} = (\mathbb{M}_{l_1 \times d} \times \ldots \times \mathbb{M}_{n \times l_k}) \times (\mathbb{R}^{l_1} \times \ldots \times \mathbb{R}^{l_k})$$

A neural network can be used to generate an algorithm as defined in chapter 2.

Given a sample space $\mathcal{X}$ with a unknown distribution $\mathbb{P}_X$. We can define a data set $S = (x_1, \ldots, x_m)$ as in equations (2.1) and (2.2).

If we also have a loss function $\ell(x, w) : \mathcal{X} \times \mathcal{W} \to \mathbb{R}$ of the form $\ell(x, w) = \ell(f_w(x))$ for $f_w \in \mathcal{N}$. Then we can define a algorithm $\mathbb{P}_{W|S=s}$ where we generate a distribution in $\mathcal{W}$ by minimizing ℓ with respect to w. Note this minimization process is not necessarily deterministic and that is why, in general, we get a distribution on $\mathcal{W}$ rather than a single point.

3.2.2 Manifolds

We refer the reader to Cayton (2005) for precise definitions of manifolds, diffeomorphisms, smooth manifolds, and embeddings. For the purpose of this chapter, we have a simpler interpretation of a manifold as given below:

Definition 3.1. *Let $d \leq n$ and let $f : \mathbb{R}^d \to \mathbb{R}^n$ be a smooth function. Let M be the range of f; i.e.,*

$$M = \{x \in \mathbb{R}^n \mid f(t) = x \text{ for some } t \in \mathbb{R}^d\}.$$

*Then M is defined as the d-dimensional **manifold** characterized by f.*

Note that the function f itself may be parametrized by some parameter w in a parameter space $\mathcal{W}$, namely f_w. Each $w \in \mathcal{W}$ generates a manifold M_w via the range of f_w. In this chapter, we consider a neural network-based parametrization of f. A neural network with d inputs, n outputs, and smooth activation functions (such as the sigmoid function) defines a smooth map from $\mathbb{R}^d$ to $\mathbb{R}^n$. This map is parametrized by the weights and biases of the neural network.

3.3 Manifold learning

Let S be a data set of points in $\mathbb{R}^n$. Assume that the points in S belong to a unknown manifold M of dimension $d < n$.

Manifold learning consist of approximating this M using only the information given by the data set S.

In this section we describe how manifold learning is implemented with the surface algorithm as well as with AEs and DAEs.

3.3.1 AEs and DAEs

Let $S = \{x_i \in \mathbb{R}^n \mid 1 \leq i \leq N\}$ be the data set contained in a unknown manifold M.

Each AE and DEA can be viewed as the union of two parts, a decoder and a encoder network. Let $\mathcal{E}$ be the decoder network and let $\mathcal{W}_e$ be it's space of parameters. For $f_w \in \mathcal{E}$ we have that $f_w : \mathbb{R}^n \to \mathbb{R}^d$ with $d < n$.

Let $\mathcal{D}$ be the decoder network and let $\mathcal{W}_{\text{dec}}$ be it's space of parameters. For $g_{\hat{w}} \in \mathcal{D}$ we have that $g_{\hat{w}} : \mathbb{R}^d \to \mathbb{R}^n$.

The AEs are then formed by stacking the decoder network on top of the encoder network i.e. we consider an autoencoder network $\mathcal{A}$ with space of parameters $\mathcal{W}_e \cup \mathcal{W}_{\text{dec}}$ and with functions of the form $h_{(w,\hat{w})} = g_{\hat{w}}(f_w)$, with $f_w \in \mathcal{E}$ and $g_{\hat{w}} \in \mathcal{D}$.

Consider the loss function $\ell(x, w) = ||x - h_(w, \hat{w})(x)||$.

The hope is that minimizing this ℓ the manifold $M_w = \text{Image}(f_w)$ will approximate M.

The only difference for DAEs is that are DAEs are trained on a noisy version of the S data set.

3.3.2 Surface Algorithm

Let $S = \{x_i \in \mathbb{R}^n \mid 1 \leq i \leq N\}$ be the data set contained in a unknown manifold M.

Let $\mathcal{N}$ be a neural network from $\mathbb{R}^d$ to $\mathbb{R}^n$ with smooth activation functions. Let $\mathcal{W}$ the set of all possible weights and biases of $\mathcal{N}$. Let $f_w \in \mathcal{N}$ and M_w the manifold generated by

f_w. We aim to find a $w^* \in \mathcal{W}$ such that M_{w^*} approximates M. Consider the loss function

$$\ell(x, w) = \|x - P(x, M_w)\|, \tag{3.1}$$

where $P(x, M_w)$ is the projection of x on M_w. In other words,

$$P(x, M_w) = \underset{\hat{x} \in M_w}{\operatorname{argmin}} \|x - \hat{x}\|.$$

If $\ell(x_i, w) = 0$, for all $x_i \in S$, then the corresponding manifold M_w perfectly interpolates the training data. So to approximate M we want to minimize ℓ in the data set S.

Before minimizing ℓ note that in general there is no closed-form expression for the projection $P(x_i, M_w)$. So to compute it we use the following projection algorithm. First we preform gradient descent on the manifold: Initialize a random $t_0 \in \mathbb{R}^d$ and perform gradient descent on $\|x_i - f_w(t)\|$ (the gradient $\nabla_t \|x_i - f_w(t)\|$ is obtained via backpropagation through the surface network f_w). Next, denote by $T(x_i, M_w)$ the point where gradient descent converges and set $P(x_i, M_w) := f_w(T(x_i, M_w))$. The projection algorithm is described in Algorithm 1.

Now to minimize $\ell(x_i, w)$, it is desirable to update w by taking steps as per $-\nabla_w \ell(x_i, w)$. Notice that according to equation (3.1), the loss $\ell(x_i, w)$ depends on the projections $P(x_i, M_w)$, which themselves presents a minimization problem. However, the gradient of $\ell(x_i, w)$ may still be calculated by an application of Danskin's theorem (see Danskin 2012):

Theorem 3.2. *(Danskin's) Let D be nonempty compact topological space and $g : D \times \mathbb{R}^n \to \mathbb{R}$ be such that $g(t, \cdot)$ is differentiable for every $t \in D$ and $\nabla_w g(t, w)$ is continuous on $D \times \mathbb{R}^n$. Also assume the set $\operatorname{argmin}_{t \in S} g(t, w)$ is a single point, denoted by $t^*(w)$. Then the corresponding max-function*

$$\phi(w) = \min_{t \in D} g(t, w)$$

is locally Lipschitz continuous, differentiable at w and

$$\nabla \phi(w) = \nabla_w g(t^*(w), w)$$

To see that the hypothesis of the theorem are satisfied observe that ℓ can be rewritten as:

$$\begin{aligned} \ell(x_i, w) &= ||x_i - \operatorname*{argmin}_{\hat{x} \in M_w} \|x_i - \hat{x}\|| \| \\ &= \min_{t \in \mathbb{R}^d} ||x_i - f_w(t)|| \end{aligned}$$

The function $g(t, w) = ||x_i - f_w(t)||$ satisfies the hypothesis of differentiability and continuity. Although the minimum is taken over $t \in \mathbb{R}^d$, in practice we feed f_w values form a bounded region $D \subset \mathbb{R}^d$, so the compactness hypothesis is also met. And we make the assumption that the set $\operatorname{argmin}_{t \in D} ||x_i - f_w(t)||$ is a single point; reasonable assumption in practice.

Then, as a result of applying Danskin's theorem we get that the gradient of ℓ with respect to w is:

$$\nabla_w \ell(x_i, w) = \nabla_w \|x_i - f_w(t^*(w))\|$$

where $t^*(w) = \operatorname{argmin}_{t \in S} ||x_i - f_w(t)||$.

However in practice we cannot find $t^*(w)$ precisely, so we approximate with the pro-

jection algorithm described above. Let $T_i = T(x_i, M_w)$ be the output of the projection algorithm with inputs x_i and M_w. So we calculate

$$\nabla_w \ell(x_i, w) = \nabla_w \|x_i - f_w(T_i)\|.$$

The latter expression is easily obtained using backpropagation for the surface network. We note that Danskin's theorem is also used in adversarial training—an algorithm for defending against adversarial attacks (see Madry et al. 2017). Algorithm 2 shows how to update w via minibatch stochastic gradient descent (SGD) such that the loss function $\ell(x_i, w)$ diminishes during training. Note that we may use any first-order optimization algorithm in place of SGD.

As noted before, the surface algorithm learns an interpolating manifold for the training data. GANs and AEs also learn such a manifold, but the surface algorithm differs from these in one key regard: both GANs and AEs have two deep networks being trained simultaneously—generator and discriminator for GANs; and encoder and decoder for AEs. Note that learning the manifold only requires us to learn the decoder or generator that transforms a low dimensional input into a high dimensional output. The added complexity of a deep network for the discriminator or encoder renders the algorithm susceptible to overfitting, since the expressive power of the network may be so large that it memorizes the dataset without actually achieving a meaningful data representation. Moreover, it also makes the training procedure unstable. For instance, training GANs involve solving a min-max optimization problem with multiple saddle points, and it is known hard to stabilize the GAN training procedure.

In contrast, the surface algorithm learns only what is necessary: the decoder. The encoder is replaced by a simple projection operator. This projection step is intuitive from a geometric point of view—a data point is mapped to the nearest point on the learned manifold. During the training procedure, the manifold stretches and twists to accommodate all training data points as best as possible. The main advantage of this procedure is that we are able to replace the black-box of the encoder neural network by

the projection step, which is understood much better. We posit that the accuracy of the projection step combined with the reduced complexity of the entire network leads to a manifold that avoids overfitting.

Algorithm 1 Projection Algorithm

Input: Data point x, manifold M_w, step sizes $[\eta_i]_0^{\text{iter}-1}$
Output: $T(x, M_w)$ and $P(x, M_w)$

Initialize a random $t_0 \in \mathbb{R}^d$
for $i = 1, 2, \ldots,$ iter **do**
    $t_i := t_{i-1} - \eta_{i-1} \nabla_t \|x - f_w(t_{i-1})\|$
end for
Set $T(x, M_w) := t_{\text{iter}}$
$P(x, M_w) = f_\theta(T(x, M_w))$
return $T(x, M_w)$, $P(x, M_w)$

Algorithm 2 Surface Algorithm

Input: dataset S, neural network with parameters in $\mathcal{W}$, step sizes $[\eta_i]_0^{\text{iter}-1}$
Output: Parameters $w^* \in \mathcal{W}$ such that M_w^* interpolates S

Randomly initialize weights to $w_0 \in \mathcal{W}$
for $i = 1, 2, \ldots,$ iter **do**
    Sample minibatch i given by $S_i = \{x'_1, \ldots, x'_B\}$
    for $j = 1, 2, \ldots, B$ **do**
        $T_j =$ Algorithm 1(x'_j, M_w)
    end for
    Update weights:

$$w_i = w_{i-1} - \eta_{i-1} \nabla_w \left[\frac{1}{B} \sum_{j=1}^{B} \|x'_j - f_w(T_j)\| \right]$$

end for
return w_{iter}

3.4 Experimental Results

We perform two kind of experiments.

For the first kind, we generate synthetic data in two dimensions and then use the surface algorithm to learn the data manifold.

For the second kind of experiments, we implement the manifold defense as follows: Let

$\mathcal{D}$ be a classification learning algorithm trained with some data set S. Algorithm $\mathcal{D}$ input is some value x and it's output is some label y. From a set $\hat{S}$ of test points, adversarial and noisy data is generated to try to fool $\mathcal{D}$.

The manifold defense consists on learning a data manifold using the same set S that was used to train $\mathcal{D}$. Then the adversarial or noisy data are projected into the learned manifold, and then the projection is fed into $\mathcal{D}$.

We use a feedforwrd neural network trained in the MNIST data set as the classifier. Then learn the data manifolds using the surface algorithm, AEs and DAEs. Adversarial and noisy data that fooled the classifier are projected to each of the three learned manifolds and the projections are fed into the classifier, comparing which of the projections will result in a correct classification.

All codes were written in PyTorch and experiments were deployed on a workstation equipped with two NVIDIA TITAN Xp GPUs.

3.4.1 Experiments in Two Dimensions

We generate synthetic data in two dimensions and seek a one-dimensional manifold that best describes the dataset.

Architecture and Training

For each of the experiments, the surface algorithm's neural network consisted of an input layer of unit size, three hidden layers of 75 units each, and an output layer of size two. The activation is the ReLu function. Training was done using Adam optimization with the standard parameters: learning rate= 0.001, $\beta_0 = 0.9$, $\beta_1 = 0.999$ and $\epsilon = \text{e}{-8}$. Training was done over 50 epochs. The projection onto the manifold was done using the dual annealing routine on NumPy (see Oliphant 2006).

Data

We generate three sets of synthetic dataset. The first set is drawn from two independent 2-dimensional Gaussian distributions with parameters $\mu_1 = (-5, -5)$, $\sigma_1 = I$ and $\mu_2 = (-10, -10)$, $\sigma_2 = 2I$; shown in Figure 3.1a. The second dataset has two noisy moon shapes with the noise being Gaussian along the normal direction to the moons; shown in Figure 3.1b. The variance of the noise is 0.5. This data was generated using the scikit-learn library (see Pedregosa et al. 2011). The third dataset consists of two concentric noisy circles with the noise being Gaussian along the normal direction to the circles; shown in Figure 3.1c. The radii of the circles are 2 and 5 and the common center is $(-5, -5)$. The variance of the noise for both circles is 0.5.

Results

We trained the surface algorithm for each of the synthetic datasets. It can be seen in Figure 3.1 the network learns a suitable lower dimensional manifold as in (b) and (c). In (a) it can be seen that the learned manifold captures the direction that differentiates the two cluster of points, similar to PCA.

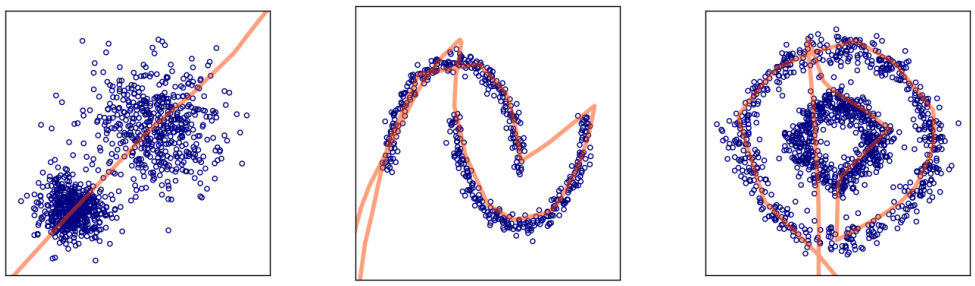
3.4.2 Experiments on MNIST

In this section we detail experiments performed on the MNIST data set.

Architecture and Training

For the experiments on the MNIST dataset, four different kind of networks were trained: a fully-connected neural network classifier, a set of AEs, a set of DAEs, and the surface algorithm.

The classifier has an input layer of 784 units, two hidden layers of 200 units each, and an output layer of 10 units. The activation is the ReLu function and the loss is the cross-entropy. The training was done on the MNIST train data set using SGD with a learning rate of .08 and mini batches of size 500. The network was trained for 100 epochs.



(a) (b) (c)

Figure 3.1: **Examples of the learned manifold constructed by the Surface Network on two dimensions** The orange line is the manifold constructed by the Surface Network (with 3 hidden layers) and the dots are the synthetic generated data. In (a) data are generated by two independent 2-dimensional Gaussian random variable with different means and variances I and $2I$. (b) Moon shapes with Gaussian noise along the normal direction. The variance of the noise is .5. (c) Two concentric circles with Gaussian noise along the normal direction. The variance of the noise is .5. The above figures demonstrate that the manifold is able to capture the shape of the training data reasonably well via a one-dimensional curve.

After training, the accuracy on the MNIST test dataset was 98%.

A total of seven AEs were trained. As described in section 3.3), each AE can be viewed as the union of two parts, a decoder and a encoder network. The decoder network was the same for all of our AEs while the encoder network was of of variable depth (varying from a depth of 2 to a depth of 8). The decoder network has input layer of 10 units, two hidden layers of 300 and 700 units and an output layer of 784 units. The encoder network consists of an input layer of 784 units and hidden layers that decrease in size to an output layer of 10 units (the size of the hidden layers decrease with depth). The AEs are then formed by stacking the decoder network on top of the encoder network. Activation function used is the sigmoid function. Training was done on the MNIST test set with the reconstruction error as the loss function; i.e., if x is a training data point and f denotes the AE, the loss function used is $||x - f(x)||$. The parameters for the training can be seen in Table 3.1.

Architectures for the DAEs were indentical to that of the AEs, and training parameters were similar (Table 3.1). The only difference is that DAEs were trained on a noisy version of the MNIST training set. Specifically, each point x in the training dataset was modified

	Surface	DAE	AE
Epochs	15	1000	300
Batch Size	1000	1000	500
Learning Rate	1e-3	1e-3	1e-3
Projection			
Epochs	3×100, 400	-	-
Batch Size	100, 784	-	-
Learning Rate	5, 1e-3	-	-

Table 3.1: **Training parameters:** All training was done using Adam optimization with the same parameters $\beta_0 = .9$, $\beta_1 = .999$ and $\epsilon = 1e{-}8$. Algorithm 1 was implemented by resetting Adam optimization three times with the same parameters and one more time with different parameters (separated by a "," in the table).

by adding independent Gaussian noise of mean 0 and variance 0.5 to each of its pixels. If x_n is the noisy version of x and f_D denotes the DAE, then the loss function minimized was $||x - f_D(x_n)||$.

The surface algorithm as described above consists of a network that generates a low dimensional manifold M and a projection step that given a point x, finds the the closest point to x on the manifold M. The architecture of the network that generates a lower dimensional manifold M is the same than the architecture of the decoder part of the AEs and DAEs. The projection step is performed using Adam optimization to minimize the distance between a point x in the training dataset to the manifold M. The loss function to minimize is the distance of the training data point to the manifold as in equation (3.1). Training parameters can be found in Table 3.1.

Test Data

Three kinds of test data were generated from the MNIST test dataset. Noisy data was generated by adding independent Guassian noise to each of the pixels of a test data point. The mean of the noise was always 0 and the variance varied from .1 to .9.

Adversarial data was generated for the regular classifier using projected gradient descent (PGD) on the MNIST test dataset as follows:

Let $\ell_c(x, w)$ be the loss function of the trained classifier network. Let x_0 a test data

point. We construct a adversarial example from x_0 by setting:

$$\hat{x}_j = x_{j-1} + \epsilon \cdot \text{sign}(\nabla_{x_{j-1}} \ell_c(x_{j-1}, w))$$
$$x_j = P(\hat{x}_j, B_\alpha^\infty(x_0))$$

Where ϵ and α are parameters known as the step size and adversarial strength respectively and $P(\hat{x}_j, B_\alpha^\infty(x_0))$ is the projection of $\hat{x}_j$ on the ℓ-infinity ball of radius α around x_0. The idea behind this adversarial data is to move x_0 in the direction that maximizes the classifier loss but without making the adversarial example too different to the human eye by keeping it inside a ℓ-infinity ball around the original point. This way producing a miss classification without changing the original point too much from the human perspective.

The step size used was $\epsilon = .3$ and the adversarial strength α varied from .1 to .9. We used 20 iterations of PGD.

Finally adversarial data that stayed on the learned manifolds was also generated. Three sets of adversarial data in the learned manifolds were constructed, one for each learned manifolds (generated by the AEs, DAEs and surface algorithm). Adversarial data was generated for the regular classifier by doing 20 iterations of PGD on the MNIST test dataset but after each iteration the adversarial example being constructed was projected into the learned manifold. The step size used was $\epsilon = .1$ and the adversarial strength α varied from .1 to .9.

Experimental design

Once trained, the AEs, DAEs and surface algorithm produce data manifolds of a lower dimension (in this case 10) embedded within the higher dimensional space (in this case 784). Given a point x in the 784 dimensional spaces, its projection can be found in each of this lower dimensional learned manifolds. If the point x is fed into the AEs or DAEs the encoder part of this networks will map x to a lower dimensional vector z. Then this z will be mapped back to a x' 784 dimensional vector by the decoder part of this same networks. We say then that $\hat{x}$ is the projection of x in the data manifold generated by

	Surface	DAE	AE
Test			
Accuracy	0.94	0.81 (2)	0.95 (3)
Reconstruction	3.4	5.4 (5)	2.8 (3)
Noise			
Accuracy	0.89	0.92 (3)	0.30 (7)
Reconstruction	4.0	3.6 (3)	8.1 (8)
Adversarial			
Accuracy	0.82	0.53 (6)	0.35 (6)
Reconstruction	6.5	5.8 (4)	6.0 (6)

Table 3.2: **Performance of surface algorithm, denoising autoencoder (DAE) and autoencoder (AE)** on raw test data, noise data (with strength $\sigma = .5$) and adversarial data (with strength $\alpha = .2$). Accuracy and reconstruction error ($||\cdot||$); the number in parenthesis is the depth of the DAE and AE for which the values were calculated. In each case the encoder depth was chosen so the accuracy was maximized and the reconstruction error minimized.

the AEs or DAEs. If the point x is geometrically projected (as described in section 3.3) to the manifold generated by the surface algorithm to a point $\hat{x}$ we say that this point is the projection of x in the manifold generated by the surface algorithm.

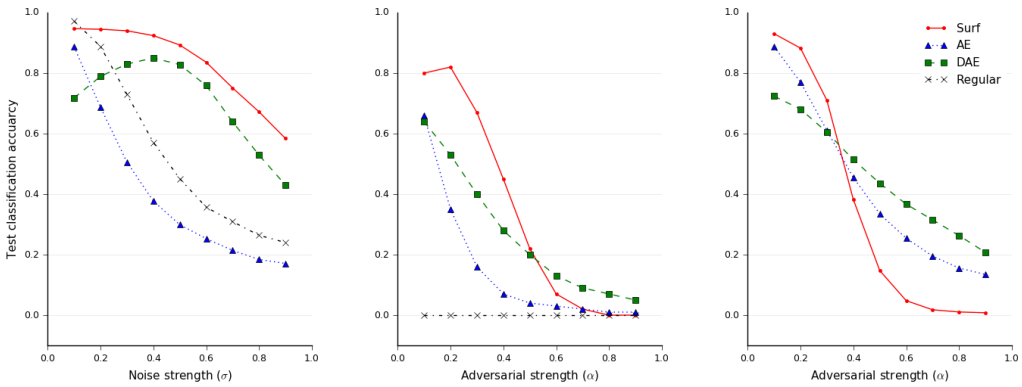
The experiments consists of the following. Noisy data x is constructed from the original test data (as mentioned in the previous sections). Then this data is projected into the manifolds generated by the AEs, DAEs and surface algorithm, producing three sets of data $\hat{x}_i$, $i = 1, 2, 3$. Then the projected data $\hat{x}_i$ is fed into the regular classifier. The accuracy of the classifier is compared on the projections in the three manifolds. Alternatively the projected point is compared to the original data x. This is the reconstruction error $||x - \hat{x}||$ as it can be seen in Table 3.2. We repeat the same experiment with the adversarial data as well as with the adversarial that was generated to stay in the learned manifold.

Results

The result of the manifold defense implemented with the AE, DAE and surface algorithm is shown in Figure 3.2. The adversarial and noise strength of data is plotted against the accuracy of the classification of such data when projected onto the learned manifolds.

The result for noisy and adversarial data are shown in figures 3.2(a) and 3.2(b). Data

shown is for the AE and DAE are for the depths that had the best performance among all the once we trained. It can be seen that for noisy data, surface algorithm outperforms AE and DAEs for all the levels of noise added. As for adversarial data, surface algorithm is better than AE and DAE for adversarial strength less than .5. Note that adversarial strengths larger than 0.5 are rarely considered in practice, since the adversary has sufficient power to turn an image entirely gray (each pixel value 0.5), or indeed completely change an image to resemble an image of a competing class.

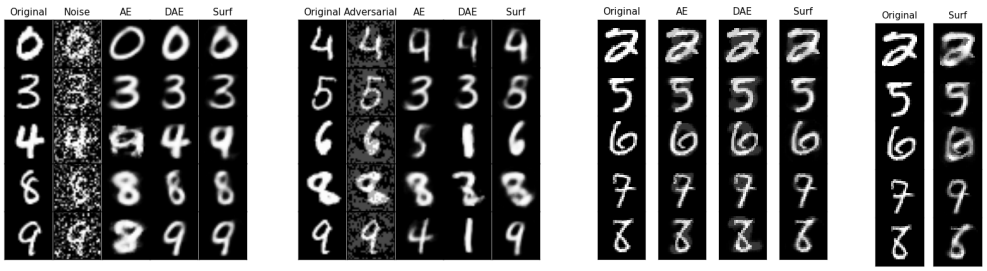


(a) (b) (c)

Figure 3.2: **Manifold defense performance comparison:** Adversarial and noise strength of data plotted against the accuracy of the classification of such data when projected onto the learned manifolds. For (a): noisy data, (b): adversarial data and (c): adversarial data on the learned manifold. (a) The Noisy data was generated by adding Gaussian noise with mean 0 and variance σ, independently to each pixel of the MNIST test set. (b) The adversarial data was generated with 20 iterations of PGD applied to the Standard Classifier Network also on the MNIST test set with step size $\epsilon = .3$ and adversarial strength α. (c) Adversarial data on the learned manifold were generated for each one of the three schemes. This was done with 20 iterations of PGD applied to the Standard Classifier Network on the MNIST test. Additionally, in each iteration the resulting point was projected back to the learned manifolds. The step size for the three schemes is $\epsilon = .1$ and the adversarial strength is α. The surface algorithm has 10 input units. For (a) and (b) The AE and DAE used on the noisy data both have 6-layers and the AE and DAE networks used on the adversarial data have 7 and 3 layers respectively. For (c) the AE and DAE networks have 7 and 6 layers respectively. The depths are the ones that showed better results.

In Figure 3.3 we see examples of how points on the data manifolds constructed by AE, DAE and surface algorithm look like. For the noisy data all three manifold seem to clean

the data reasonably well, but for the adversarial data, AE and DAE produce completely different pictures while surface algorithm retains a more faithful representation of the original data.



(a) (b) (c) (d)

Figure 3.3: **Examples of points on the learned manifold** constructed by autoencoder (AE), denoising autoencoder (DAE), and surface algorithm (Surf). (a): Reconstruction of noise data (second column) by AE, DAE and Surface Network. The noisy data was generated by adding Gaussian noise with mean 0 and variance .5 to the original data.(b): Reconstruction of adversarial data (second column) by AE, DAE and Surface Network. The adversarial data was generated with 20 iterations of PGD applied to the standard classifier network on the MNIST test set with step size $\epsilon = .3$ and adversarial strength .2. (c): Adversarial data on the learned manifolds. Examples were constructed with 20 iterations of PGD applied to the standard classifier network on the MNIST test and projecting each iteration to the manifolds generated by the surface algorithm, the AE and the DAE. The step size is $\epsilon = .1$. All the examples are being incorrectly classified by AE and DAE but correctly classified by surface algorithm. Strength of the adversarial data is $\alpha = .2$. (d): Adversarial data incorrectly classified by the surface algorithm. Strength of the adversarial data is $\alpha = .4$.

For the adversarial data guaranteed to stay on the learned manifolds, the experiment results can be seen in Figure 3.2(c). Surface algorithm performs well for low values of the adversarial strength and the performance degrades with increasing strength of the adversary.

However, in Figure 3.3(c) we can see that when the adversarial strength is low, adversarial data on the manifolds generated by AE and DAE produce pictures that are easily classified by a human, yet lead to incorrect classification by the regular classifier. But if the adversarial example is on the manifold generated by the surface algorithm there in no incorrect classification. Furthermore if the adversarial strength is bigger, when an

incorrect classification happens on an adversarial points on the manifold generated by the surface algorithm, as in 3.3(d), the picture generated starts to blend the shapes of two different numbers. This indicates that the "adversarial examples" constructed when the surface algorithm is used in the manifold defense are closer to being truly borderline images between two different classes.

3.5 Conclusion

We presented a new algorithm for manifold learning called the surface algorithm, which is easily described and efficiently trained. Our algorithm leveraged the expressive power of neural networks and simple geometric ideas to learn a data manifold. Our main innovation was incorporating a geometric projection step during the training process and applying Danskin's theorem to efficiently calculate derivatives. We compared the manifold learned by the surface algorithm to manifolds learned by autoencoders and denoising autoencoders. Our experimental results concerning reconstruction error and adversarial accuracy indicate that the surface algorithm learns a better manifold compared to AEs or DAEs. In particular, when noisy images have lower reconstruction error for the surface algorithm compared to AEs or DAEs. Manifold defense methods are more accurate when using the surface manifold compared to using AE or DAE in the same defense method. Moreover, we observed that adversarial data on the surface manifold often look like true borderline images, unlike AE or DAE.

There are several interesting research questions that emerge from our work. Just like AEs or DAEs, the surface algorithm cannot be used as a generator. A natural question is to explore more geometric training methods for learning GANs and VAEs. We also investigated an intriguing phenomena that there is a sweet spot for the encoder complexity in an AE. Encoder networks that are too small are unable to learn complex encodings, whereas complicated encoder networks lead to overfitting. Our experiments are preliminary and a more in-depth study seems warranted. Finally, there are numerous questions regarding the actual implementation of the surface algorithm—How to initialize network weights?

Is the sigmoid activation function a good choice? How to ensure learned representations are confined to some region of $\mathbb{R}^d$?

Chapter 4

A Variational Information Bottleneck Principle for Recurrent Neural Networks

4.1 Introduction

In the previous chapter we used manifold learning as a method of dimensionality reduction. The idea is that when we have a high dimensional input, by reducing it's dimension, we can extract relevant information and discard useless information from such input. The we will use this information for whatever task we have: prediction, classification, etc.

This idea of extracting relevant information from our input and discarding unnecessary one is a general principle that is not limited to dimensionality reductions. We can extract useful information in different ways. In this chapter we explore the Information Bottleneck (IB) principle in which we use the measure of mutual information to decide what is useful information about our input. Moreover we do it in the context of causal time series, where we want to predict a sequence of events base on a input sequence and we want to use the extra information given by the causality of the input sequence.

The Information Bottleneck principle, proposed in Tishby, Pereira, and Bialek (2000),

is one of the most influential information theoretic approaches to the problem of feature extraction in machine learning. The IB principle functions in the standard statistical learning setting: Predict a random variable Y, called the label, after observing a random variable X, called the sample. For instance, X could be an image and Y could be a binary label that indicates the presence or absence of a face in the image. Machine learning algorithms seek to extract *features* Z – which are (possibly random) functions of the sample X – that are then used to predict Y using simpler algorithms. Naturally, it is desirable to extract features that are "simple" (e.g. low-dimensional) and "relevant"; i.e., they are highly predictive of Y. The IB principle elegantly solves this problem by measuring simplicity and relevance using a most natural information theoretic metric: mutual information.

The IB principle stems from the observation that, often, the sample X contains information that is redundant or irrelevant to the task of predicting Y. An extracted feature Z should get rid of this excess baggage while maintaining its relevance to Y. Stated in the language of information theory, the mutual information $I(X;Z)$ should be made small, but not at the expense of $I(Y;Z)$. Since Z is extracted from X, the random variables satisfy the Markov chain $Y \to X \to Z$, which leads to a tension between the two mutual informations: If $I(X;Z)$ is small, the data-processing inequality dictates that $I(Y;Z)$ cannot remain large. This tradeoff is presented in the form of an optimization problem:

$$\max_{p_{Z|X}(\cdot|\cdot)} I(Y;Z) - \beta I(X;Z), \tag{4.1}$$

where $\beta > 0$ is a parameter that trades off the two mutual information quantities. Solving this optimization problem gives an optimal stochastic map to extract features Z from X.

The IB principle is remarkable for its generality. The extracted features depend only on the data distribution p_{XY} of (X,Y) and the parameter β, with no reference to loss functions and hyperparameters. As such, it has been applied to a wide variety to machine learning problems (see Slonim and Tishby 2000; Chechik and Tishby 2003; Gondek and Hofmann 2003; Chalk, Marre, and Tkacik 2016; Alemi et al. 2016).

A drawback of using the IB principle in practice is solving the optimization problem in (4.1) in large datasets. In real-world datasets, the joint distribution p_{XY} is unknown, so solving (4.1) analytically is not possible. Also, due to the high-dimensionality of p_{XY}, the number of samples is insufficient to approximate p_{XY} and solve the IB objective.

The work of Alemi et al. (2016) developed a simple and clever idea to get around this obstacle. Alemi et al. suggested approximating the IB objective using variational bounds on the two mutual informant terms. Crucially, they showed that is possible to efficiently optimize the approximate objective using stochastic gradient descent on deep neural networks. Their algorithm, called the "deep variational information bottleneck" was inspired by the the seminal work of Kingma and Welling (2013) concerning variational autoencoders. The idea of using variational bounds that are suitable for deep learning and has appeared in numerous papers since (see Wang et al. 2016; Poole et al. 2019; A. Pensia, V. Jog, and P. Loh 2020).

In this chapter, we consider the problem of *causal time-series prediction*. In the the statistical learning framework, we observe a time-series $X^T := (X_1, \ldots, X_T)$ and predict time-series $Y^T := (Y_1, \ldots Y_T)$. The causality constraint dictates the the predicated value of Y_i, denoted by $\hat{Y}_i$, depends causally on X^T; i.e., it depends on X^T only trough $(X_1, \ldots, X_i)$. Mathematically, the Markov chain $\hat{Y}_i \to (X_1, \ldots, X_i) \to X^T$ must hold. Causal time series prediction shows up naturally in many real-world settings, such as forecasting weather, forecasting financial markets, predicting demand of products in supply chains, predicting energy demand in power systems, and many other domains.

Our goal in this chapter is to develop an information theoretic approach to solving the causal time-series prediction problem. Observe that the standard IB principle is not suitable for such a setting for several reasons. The standard IB principle would suggest extracting a feature Z using the entire time series X^T, and predicting the entire time series Y^T from Z. But this would violate causality. Another difficulty is that learning to predict Y^T is essentially T separate problems: learning to predict Y_i for each i. The standard IB problem is designed for extracting features for a single prediction problem. Thus, the IB

principle may not be applied naïvely to this problem and needs to be modified suitably.

In the theory of neural networks, there exist powerful algorithms for predicting time-series called *recurrent neural networks*. Recurrent neural networks have been successfully applied to a wide variety of problems, such as machine translation, time-series prediction for financial data (see Sezer, Gudelek, and Ozbayoglu 2020), and generating text (see I. Sutskever 2013). Recurrent neural networks, explained in more detail in Section 4.4.1, are neural networks with a constantly updating internal state. This evolving internal state gives these networks the ability to remember the past while predicting the future.

A natural question is whether a modified IB principle designed for time-series prediction can be used for real-world data using a recurrent neural network. This would be exactly analogous to the contribution of Alemi et al. (2016) for the standard IB principle and vanilla deep networks.

Our work in this chapter addresses this question as follows: In sections 4.2 and 4.3 we propose a novel IB principle that generalizes the standard IB principle and is designed for time-series prediction. In section 4.4 we show that our approximate objective can be efficiently optimized using recurrent neural networks. And in section 4.5 we show that our algorithm is successful on simulated and real data, including time-series data for weather and stock prices.

Notation: For a sequence $\{x_i\}_{i=1}^n$, we use the following notation: $x^t := (x_1, x_2, \dots x_t)$. If we have N sequences of length T we index them as $\{x_n^T\}_{n=1}^N$ and we denote the t-th element of the n-th sequence as $x_{t,n}$. The density function of a random vector X at a point x is denoted by $p_X(x)$ in standard information theoretic notation. To simplify clutter, we drop the subscript and simply use $p(x)$ when there is no ambiguity. In case of ambiguity, we explicitly write the subscript. This is the notation used in T. M. Cover and J. A. Thomas (2012) as well. The entropy of X (discrete or differential) is denoted by $H(X)$. The KL-divergence between the distributions of random vectors X and Y is denoted by $D(p(x)\|p(y))$.

4.2 Problem Setting

For $T \geq 1$ let X^T and Y^T be sequences generated from an unknown probability density $p(x^T, y^T)$. Our goal is to construct a learning algorithm that predict the sequence Y^T using X^T.

The relation between X^T and Y^T could be complicated, and extracting relevant information from X^T about Y^T is not a straightforward task.

The information bottleneck principle theorizes that the way information is extracted from X^T is trough a third random sequence Z^T that discards irrelevant information from Y^T. We want an algorithm that constructs such Z^T and then use it to predict Y^T.

Remark 4.1. *Two equivalent interpretations can be given to the information bottleneck principle. One is that the variable Z^T is really happening in the phenomena studied but we cannot measure it. For example let X^T be some visual stimulus and Y^T the response of a subject to such stimulus. In that subject's brain the the neural activity would be the variable Z^T, which cannot be measured directly (as is an invasive procedure). With the IB principle we are trying to learn a distribution that lets us model this Z^T*

Another interpretation is that the variable Z^T is artificially constructed to extract information from X^T, it doesn't happen in reality. In either case the mathematical treatment is the same.

Mathematically speaking, the information bottle neck principle says there is a joint distribution $p(x^T, z^T, y^T)$ relating (X^T, Z^T, Y^T). The algorithm we construct will try to learn $p_{Z^T|X^T}(\cdot|\cdot)$ and $p_{Y^T|Z^T}(\cdot|\cdot)$.

The part of the algorithm in charge of learning $p_{Z^T|X^T}(\cdot|\cdot)$ will be called the encoder. The part of the algorithm in charge of learning $p_{Y^T|Z^T}(\cdot|\cdot)$ will be called the decoder.

To make Z^T extract only relevant information about Y^T from X^T, we want to train the encoder and the decoder so that the following optimization problem is solved:

$$\max_{p_{Z^T|X^T}(\cdot|\cdot),p_{Y^T|Z^T}(\cdot|\cdot)} I(Y^T;Z^T) - \beta I(X^T;Z^T), \tag{4.2}$$

In other words: from all the possible densities p that relate (X^T, Z^T, Y^T) we want the one such that $p_{Z^T|X^T}(\cdot|\cdot)$ and $p_{Y^T|Z^T}(\cdot|\cdot)$ will maximize $I(Y^T;Z^T) - \beta I(X^T;Z^T)$. (For some positive value of β to be determined).

Remark 4.2. *The parameter β will serve as a control to train the algorithm. A trade off between the information Z^T learns form X^T and the information Y^T learns from Z^T. Note that a big value of β will make Z^T just a copy of X^T without any prediction capabilities to estimate Y^T. And a small value of β won't allow Z^T to ecxtrac information from X^T.*

However there is two things to note before trying to design our algorithm:

- In the same way than in a physics problem, mathematically acceptable solutions have to be discarded because they don't make physical sense; The causal characteristics of our problem impose conditions on the possible densities p to consider. We investigate this conditions in section 4.2.1.

- Once we impose the corresponding conditions on p we will see in section 4.2.2 that equation (4.2) will be hard or impossible to compute explicitly. Therefore having to approximate it by something that can be computed as we will do in section 4.3.

4.2.1 Causal conditions

We want to make sure that p will generate causal time series. In other word series in which the present only depends on the past and not the future. We want also for Z^T to only depend on X^T and Y^T only to depend on Z^T. The following definitions state this conditions formally.

Definition 4.3. *The probability density $p(x^T, z^T, y^T)$ satisfy the* ***causal encoder condition*** *if the following Markov chain holds for each $1 \leq t \leq T$:*

$$Z_i \to X^t \to (X^T, Y^T, Z^{t-1}), \tag{4.3}$$

where $Z^0 \equiv 0$.

Recall that the Markov property (4.3) implies that:

$$p(x^T, y^T, z^{t-1}, z_t | x^t) = p(x^T, y^T, z^{t-1} | x^t) p(z_t | x^t)$$

From this we can see that intuitively, this means that Z_t can be generated using X^i alone, independently of the future $X^{t+1:T}$, the entire Y^T, and the previous extracted features Z^{t-1}, is unnecessary.

Definition 4.4 (Causal decoder)**.** *The probability density $p(z^T, y^T)$ satisfy the* ***causal decoder condition*** *if the following Markov chain holds for each $1 \leq t \leq T$:*

$$Y_t \to Z^t \to (Z^T, Y^{t-1}), \tag{4.4}$$

where $Y^0 \equiv 0$.

The Markov property (4.4) implies that:

$$p(z^T, y^{t-1} y_t | z^t) = p(z^T, y^{t-1} | z^t) p(y_t | z^t)$$

This means that given Z^t, the value of Y_t is independent of the it's past values; i.e. Y^{t-1}, as well as independent of future values of Z.

Remark 4.5. *Note that one may also define causal features more generally by only requiring the Markov chain* $Z_t \to (X^t, Z^{t-1}) \to (X^T, Y^T)$. *Intuitively, this means that the feature* Z_t *extracted at time* t *is generated by looking not only at* X^t, *but also at the history of generated features* Z^{t-1}. *This will be discussed in Section 4.6.*

Remark 4.6. *A consequence of the causal encoder condition is that the conditional distribution* $p(z^t|x^t)$ *factorizes as follows for* $t \leq T$*:*

$$p(z^t|x^t) = \prod_{i=1}^{t} p(z_i|x^t, z^{i-1}) = \prod_{i=1}^{t} p(z_i|x^i).$$

We can also verify that if p *satisfies the causal encoder condition then* $Z^t \to X^t \to Y^t$ *is a Markov chain for* $t \leq T$*:*

$$\begin{aligned} p(z^t|x^t, y^t) &= \prod_{i=1}^{t} p(z_i|x^t, y^t, z^{i-1}) \\ &= \prod_{i=1}^{t} p(z_i|x^i) \\ &= p(z^t|x^t). \end{aligned}$$

A consequence of the causal decoder condition is that the joint distribution $p(y^t|z^t)$ *factorizes as follows:*

$$p(y^t|z^t) = \prod_{i=1}^{t} p(y_i|z^t, y^{i-1}) = \prod_{i=1}^{t} p(y_i|z^i).$$

4.2.2 Intractability of the information bottleneck

The probability density p factorizes as:

$$p(x^T, z^T, y^T) = p(z^T|x^T, y^T)p(y^T|x^T)p(x^T)$$

However from remark 4.6 we see that $Z^t \to X^t \to Y^t$ is a Markov chain for $t \leq T$, and therefore $p(z^T|x^T, y^T) = p(z^T|x^T)$, factoring p as:

$$p(x^T, z^T, y^T) = p(z^T|x^T)p(y^T|x^T)p(x^T) \tag{4.5}$$

Observe that this implies that once we know $p(z^T|x^T)$ then p is totally determined since $p(y^T|x^T)$ and $p(x^T)$ are unknown but given by the problem.

This tells us that once we construct $p(z^T|x^T)$, rather than optimizing (4.2) over both distributions $p(z^T|x^T)$ and $p(y^T|z^T)$, we have to optimize:

$$\max_{p_{Z^T|X^T}(\cdot|\cdot)} I(Y^T; Z^T) - \beta I(X^T; Z^T), \tag{4.6}$$

where p satisfies the causal encoder property.

However, to optimize (4.6) we need to compute $p(z^T|x^T)$, since:

$$I(Y^T; Z^T) = \int p(y^T, z^T) \ln \left(\frac{p(y^T|z^T)}{p(y^T)} \right) dz^T dy^T$$

The problem is that we do not have access to $p(y^T|z^T)$.

Similarly in

$$I(X^T; Z^T) = \int p(x^T, z^T) \ln \left(\frac{p(z^T|x^T)}{p(z^T)} \right) dx^T dx^T$$

we don't have access to $p(z^T)$.

To overcome this difficulty, rather than directly computing and optimizing $I(Y^T; Z^T) - \beta I(X^T; Z^T)$ we are going to compute and optimize a lower bound.

In this lower bound we are going substitute $p(y^T|z^T)$ and $p(z^T)$ by variational approximation $q(y^T|z^T)$ and $r(z^T)$ that will allow a explicit computation.

The encoder of our algorithm will learn $p(z^T|x^T)$ as mentioned before, but the decoder will learn the variational approximation $q(y^T|z^T)$ rather than the intractable $p(y^T|z^T)$. However we will still force q to satisfy the causal decoder property.

4.3 Approximation of the information bottleneck

The result of this section is the following variational bound on the objective function in equation (4.6).

Theorem 4.7. *Let $\beta > 0$. Let (X^T, Z^T, Y^T), be random variables with joint probability density $p(x^T, y^T, z^T)$, where p satisfies the causal encoder property. Let $q(y^T, z^T)$ be a probability density satisfying the causal decoder property, and let $r(z^T)$ be any probability density satisfying $r(z^T) = \prod_{t=1}^{T} r(z_t)$. Then we have that:*

$$\begin{aligned} I(Y^T; Z^T) - \beta I(X^T; Z^T) \geq & \sum_{t=1}^{T} \int p(x^t, y_t) p(z^t|x^t) \log q(y_t|z^t)\, dx^t dy_t\, dz^t \\ & -\beta \sum_{t=1}^{T} \int p(x^t) p(z_t|x^t) p(x^t) \log \left(\frac{p(z_t|x^t)}{r(z_t)} \right) dx^t dz_t \\ & + H(Y^T). \end{aligned}$$

4.4 Optimizing using RNNs

In this section we give some background in recurrent neural networks. Then we show how we use this RNNs to build our learning algorithm.

4.4.1 Background in recurrent neural networks

In the following we'll do a formal definition of a RNN in the same fashion we did in chapter 2

A one layer recurrent neural network is a family $\mathcal{R}$ of functions that have the following characteristics:

For each function $f \in \mathcal{R}$:

- f goes from $\mathbb{R}^d \times \mathbb{R}^T$ to $\mathbb{R}^n \times \mathbb{R}^T$ for some fixed d and n integers but where T can be variable. The input of the network is a sequence of length T, with each element of the sequence having dimension d. The output will be a sequence with the same length of the input sequence. Each element of the output sequence will have dimension n.
- The input will be the sequence $x^T = (x_1, \ldots, x_T)$.
- Let l_h be positive integer. We will call this *the dimension of the hidden state.* Let $W_h \in \mathbb{M}_{l_h \times d}$, $U_h \in \mathbb{M}_{l_h \times d}$ and $b_h \in \mathbb{R}^{l_h}$. Finally let the scalar function σ_h be the *hidden activation function.*
- Let $W_y \in \mathbb{M}_{n \times l_h}$, $b_y \in \mathbb{R}^n$ and the scalar function σ_y be the activation function.
- The hidden state and output are defined recursively by:

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned}$$

  Where h_0 is a fixed vector and both σ_h and σ_y are applied elementwise. Observe that to compute y_t we are using the information of the present (x_t) as well as the information from the past (h_t).

Intuitively a RNN can be viewed as an unfolded neural network as shown in Figure 4.1, and the unfolding it a consequence of feeding the output of the network h_i back into the input. Observe that h_i is a function of the entire past x^i, and thus it serves the purpose of retaining memory in the system.

A RNN is preferred for time-series prediction problems because it is not constrained by the size of the input sequence T—the same network can be used for any length of the input sequence—and in theory, the RNN has memory that can stretch arbitrarily far back in the past. In practice, however, the vanilla RNN described above tends to have short memories. Ingenious modifications such as Long Short Term Memory (LSTM) networks (see

Hochreiter and Schmidhuber 1997) or Gated Recurrent Units (GRU) networks (see Cho et al. 2014) need to be implemented to ensure the networks are competitive in practice. The precise modifications are not important to understand for the purpose of this chapter; the basic idea presented in Figure 4.1 is the same in these modifications. For the sake of completion we'll just state the difference between the vanilla RNN and the LSTM that is the one we use in the experiments.

The only difference in a LTSM is the way the hidden state is constructed:

$$\begin{aligned} f_t =& \sigma_g(W_f x_t + U_f c_{t-1} + b_f) \\ i_t =& \sigma_g(W_i x_t + U_i c_{t-1} + b_i) \\ o_t =& \sigma_g(W_o x_t + U_o c_{t-1} + b_o) \\ c_t =& f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + b_c) \\ h_t =& o_t \circ \sigma_h(c_t) \end{aligned}$$

Where σ_g and σ_c are activation functions applied elementwise. The operation $\circ$ is element wise multiplication. The W's and the U's are matrices and the b's are vectors. c_o is a fixed vector.

However figure 4.1 is the simplest example of an RNN and it will be the architecture considered for the theory throughout this chapter. One may consider architectures where along with h_i, the output y_i is also fed back into the input. We shall discuss such RNNs briefly in Section 4.6.

The network used to construct the encoder and decoder in the following sections consists of a feedforward network on top of a recurrent neural network. More formally: Let $\mathcal{R}$ be our RNN and $\mathcal{N}$ our feedforward network. Let $r \in \mathcal{R}$ and $g \in \mathcal{N}$. And let our input be a sequence x^T of length T. Recall $r(x^T)$ is itself a sequence of length T so it can be written as $r(x^T) = (r_1(x^T), \ldots, r_T(x^T))$.

We consider functions f of the following form:

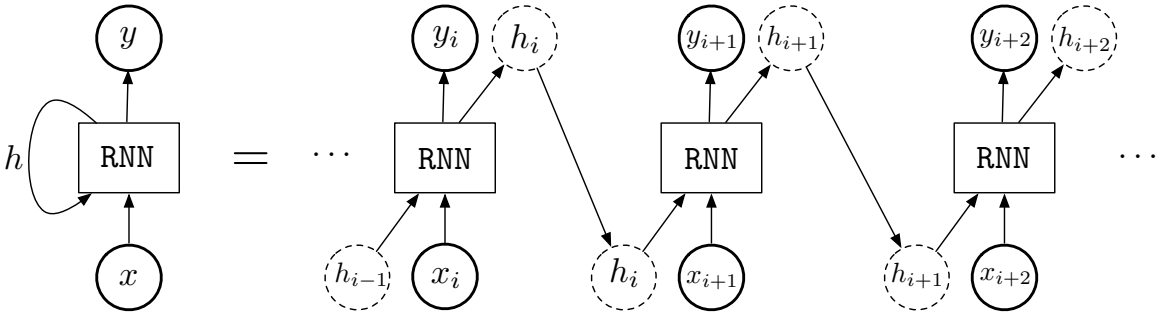


Figure 4.1: A simple recurrent neural network and its equivalent "unfolded" representation. The network may be thought of as transforming an input (x_i, h_{i-1}) to the output (y_i, h_i) at time i. The state h_i generated at time i is an input to the network at time $i+1$.

$$f = (g(r_1(x^T)), \ldots, g(r_T(x^T))) \tag{4.7}$$

Observe the same g is applied to all the elements of the output sequence of the RNN. During training the weights of g are updated simultaneously for all the elements of the sequence evaluated on the loss function.

4.4.2 Computation of the loss function for training

We assume that there is a training dataset that is a collection of sequences $\{x_n^T\}_{n=1}^N$, $\{y_n^T\}_{n=1}^N$ generated by the random variables X^T and Y^T with the unknown distribution $p(x^T, y^T)$. The goal is to use this data to train an algorithm that can predict the y_n^T from the x_n^T causally using an RNN.

We will focus on the optimizing the lower bound on the IB objective obtained in Theorem 4.7. Note that the quantity $H(Y^T)$ does not depend on the optimizing parameters, and so we may simply ignore it. We will train two RNNs to learn $p(z^t|x^t)$ and $q(y^t|z^t)$ to optimize the bound in Theorem 4.7: the encoder and the decoder, respectively. In what follows, we will describe the three parts of the algorithm: the encoder, the decoder, and the loss function.

Encoder: The encoder generates the distribution $p(z^T|x^T)$. The encoder consists of a RNN, as in Figure 4.1, stacked with a feedforward network. The input of the RNN is the sequence X_T. The output of the RNN is fed into a feedforward network (each element of the output sequence is fed to the same feedforward network as in (4.7)). The output of the feedforward networks is the sequence containing the parameters of the distribution $p(z_t|x^t)$. We parametrize $p(z_t|x^t)$ using the mean and variance of a normal distribution:

$$p(z_t|x^t) \sim N(\mu_e(h_e(x^t)), \sigma_e(h_e(x^t))I) := N(\mu_{e,t}, \sigma_{e,t}).$$

where $h_e(x^t) = h_e(x_t, h_e(x^{t-1}))$ and $h_e(x^0)$ is defined as a constant. The quantity $h_e(\cdot)$ is the hidden state of the RNN. And $(\mu_e, \sigma_e I)$ is the output of the feedforward network.

Since we are using a RNN we have that once x^t is given, the variable z_t only depends on x^t. This is the causal encoder property, and so $p(z^t|x^t) = \prod_i^t p(z_i|x^i)$ for all $t \leq T$.

The Gaussian assumption on $p(z_t|x^t)$ is not limiting due to the expressive power of neural networks and the fact that any distribution can be approximated using Gaussian kernels.

Decoder: The decoder has to learn $q(y^T|z^T)$. It has the same structure as the encoder: a RNN as in Figure 4.1, stacked with a feedforward network. The input of the RNN is the sequence Z_t. The output of the RNN is fed into a feedforward network. The output of the feedforward networks is the sequence containing the parameters of the distribution $q(y^T|z^T)$. We parametrize $q(y_t|z^t)$ using the mean and variance of a normal distribution:

$$q(y_t|z^t) \sim N(\mu_d(h_d(z^t)), \sigma_d(h_d(z^t))I) := N(\mu_{d,t}, \sigma_{d,t}),$$

where $h_d(z^t) = h_d(z_t, h_d(z^{t-1}))$ and $h_d(z^0)$ is defined as a constant. The quantity $h_d(\cdot)$ is the hidden state of the RNN. And $(\mu_d, \sigma_d I)$ is the output of the feedforward network.

Observe that since we are using a RNN we have that once z^t is given, the variable y_t only depends on z^t. This is the causal decoder property, and so $q(y^T|z^T) = \prod_{t=1}^T q(y_t|z^t)$

for all $t \leq T$.

Loss function: The loss function (ignoring the $H(Y^T)$ term) is given by

$$\begin{aligned} &\sum_{t=1}^{T} \int p(x^t, y_t) p(z^t|x^t) \log q(y_t|z^t)\, dx^t dy_t\, dz^t \\ -\beta &\sum_{t=1}^{T} \int p(x^t) p(z_t|x^t) p(x^t) \log \left(\frac{p(z_t|x^t)}{r(z_t)} \right) dx^t dz_t. \end{aligned} \tag{4.8}$$

With our dataset $\{x_n^T\}_{n=1}^N$, $\{y_n^T\}_{n=1}^N$, we use the empirical approximation of the distribution $p(x^T, y^T)$ on (4.8) to get the empirical loss

$$\begin{aligned} \frac{1}{N} \sum_t \sum_n &\int p(z^t|x_n^t) \log q(y_{t,n}|z^t)\, dz^t \\ -\beta &\int p(z_t|x_n^t) \log \left(\frac{p(z_t|x_n^t)}{r(z_t)} \right) dz_t \end{aligned} \tag{4.9}$$

We shall assume $r \sim N(0, I)$—this is a common assumption to encourage the learned features to be well-distributed (see Kingma and Welling 2013). In the second term, we get the KL-divergence between the Gaussian distribution $p(z_t|x^t)$ with parameters $(\mu_{e,t}, \sigma_{e,t})$ and the standard normal, which has a closed-form expression.

To compute the first term of (4.9) we would like to approximate the integrals using Monte Carlo method by sampling from $p(z^t|x_n^t)$. The problem is that this distribution depends on the parameters of the encoder and when we are training we can't backpropagate the derivative with respect to this parameters through the operation of sampling. To solve this problem we use the reparametrization trick (see Kingma and Welling 2013). Note that:

$$\int p(z^t|x_n^t) \log q(y_{t,n}|z^t)\, dz^t = \mathbb{E}_{Z^t \sim p(z^t|x_n^t)}[\log q(y_{t,n}|Z^t)]. \tag{4.10}$$

Recall $p(z^t|x_n^t) = \prod_i^t p(z_i|x_n^i)$ for any $t \leq T$ where each $p(z_i|x_n^i) \sim N(\mu_e(h_e(x_n^i)), \sigma_e(h_e(x_n^i))I)$.

Let $\epsilon^i = \{\epsilon_i\}_{i=1}^t$ independent standard random normals. Then if $Z^t \sim p(z^t|x_n^t)$:

$$Z^T = (Z_1, \ldots, Z_t) \tag{4.11}$$

$$= (\sigma_e(h_e(x_n^1))I\epsilon_1 + \mu_e(h_e(x_n^1)), \ldots, \sigma_e(h_e(x_n^t))I\epsilon_t + \mu_e(h_e(x_n^t))). \tag{4.12}$$

If we define

$$f(\epsilon^t, x_n^t) = (\sigma_e(h_e(x_n^1))I\epsilon_1 + \mu_e(h_e(x_n^1)), \ldots, \sigma_e(h_e(x_n^t))I\epsilon_t + \mu_e(h_e(x_n^t))), \tag{4.13}$$

then from (4.10), (4.12) and (4.13) we get that the first term in (4.9) satisfies:

$$\frac{1}{N}\sum_t \sum_n \int p(z^t|x_n^t) \log q(y_{t,n}|z^t)\, dz^t = \mathbb{E}_{\epsilon^i}[\log q(y_{t,n}|f(\epsilon^t, x_n^t))]. \tag{4.14}$$

This way the sampling is done for the independent standard normal ϵ^i and is independent of the encoder parameters allowing us to run backpropagation through the loss function.

4.5 Experimental results

In this section we present the setup and result of our experiments done with simulated and real data.

Encoder and decoder each consist of a LSTM recurrent layer and a fully connected layer. Trough all the experiments the structure of the networks was kept constant. Units of the recurrent layer: 50, Activation of the recurrent layer tanh, units of the fully connected layer: 100, Activation of the fully connected layer: Relu. No activation for the output layer.

The dimension of the variable Z was always 10, the optimization method was Adam, and the learning rate 0.001.

For our data $\{y_n^T\}_{n=1}^N$ and our prediction $\{\hat{y}_n^T\}_{n=1}^N$ we compute the normalized absolute

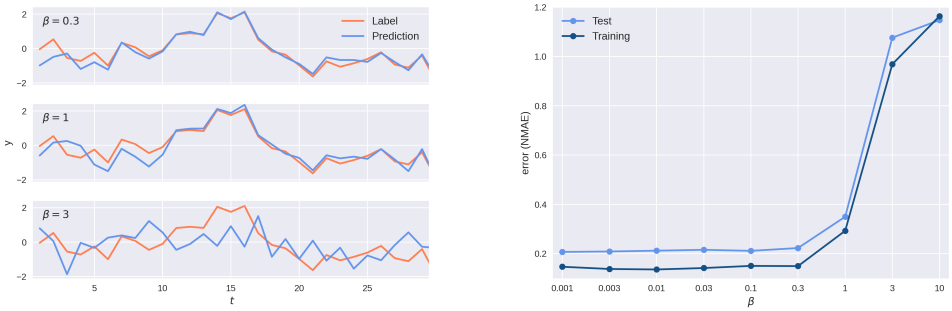


Figure 4.2: (a) Prediction $\hat{y}$ compared with actual values y for a test sequence at different values of β. (b) Normalized absolute error (NMAE) as a function of β. At $\beta = 1$ error is small but test and train error are close, showing the algorithm is not overfitting. β is plotted in a $\log_{10}$ scale.

error (NMAE) defined as:

$$\text{NMAE} = \frac{1}{\bar{y}} \sum_{n=1}^{N} \sum_{i=1}^{T} |y_{t,n} - \hat{y}_{t,n}|,$$

where $\hat{y} = \sum_{n=1}^{N} \sum_{i=1}^{T} |y_{t,n}|$.

Simulated data: We generate a sequence x^T with $T = 10,000$ of realizations of independent standard normal random variables. From it we construct a sequence y^T by means of:

$$y_t = \alpha x_t + \alpha^1 x_{t-1} \ldots + \alpha^t x_1 + \epsilon_t,$$

where $\alpha = 0.8$ and ϵ_i is noise generated from $N(0, .1)$. From x^T we obtain sequences of length 30 of the form $(x_t, \ldots, x_{t+30})$. We do the same for y^T. The collection of this sequences would be our data: $\{(x_t, \ldots, x_{t+30})\}_{t=1}^{T-30}$, $\{(y_t, \ldots, y_{t+30})\}_{t=1}^{T-30}$. We will use the first 80% on the data for training and the rest for testing. we trained using mini batches of size 200 during 2000 epochs.

We found that approximation was good for the correct values of β as it can be seen in Figure **??**. We have several observations:

- The approximation is better for the last elements of each sequence. This is because while maximizing the loss function, we have less information on the first elements since the variables Y^t and Z^t only depend on the past. This is not an issue in actual implementation since we will use the past information at all times.

- As the values of β get high the algorithm stops learning information about Y^T and only focuses minimizing $I(X^T; Z^T)$. This leads to a worse prediction.

- Even though the algorithm was trained with a sequence of a single length, it performs similarly with sequences of different lengths (data not shown).

The dependence of the error NMAE on parameter β can be seen in figure **??**. Here we observed that for lower values of β the algorithm is overfitting since the test error is bigger than the train error. But as β increases the gap between the two errors decreases. The optimal value of β is when the gap between the error is smallest but the error itself is still reasonable. In this case the value is $\beta = 1$ and the test NMAE error at this point is 0.349 while the train error is 0.292.

Weather data: We use the weather time series data recorded by Max Planck Institute for Biogeochemistry (see *Weather dataset from the Max Planck Institute for BioGeochemistry* n.d.). We use data from 2009 to 2016. The data is recorded every 10 minutes but we take only the hourly measures. We use 14 of the features including atmospheric pressure and humidity, to predict temperature. Similar to our simulated data experiments, we set sequences of one week length and take %80 percent of the data for training and the rest for testing. We normalized all the data using the empirical mean and empirical standard deviation from the training set. We use mini batches of length 500 over 200 epochs. Figure 4.3 shows the good performance of the algorithm on this data for $\beta \approx 0.1$, which degrades with larger values of β as expected.

Stock market data: We use the historical prices of the Dow Jones index (DJ30) and the Shanghai Stock Exchange (SSE), downloaded from Yahoo finance (see *Yahoo finance*

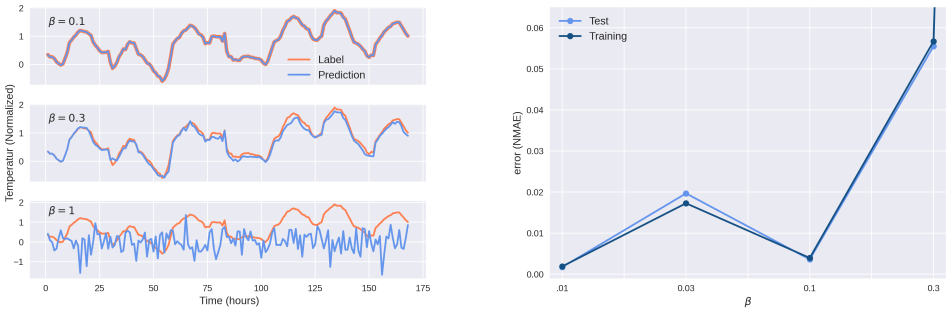


Figure 4.3: (a) Prediction of the temperature for a test sequence at different values of β. (b) Normalized absolute error (NMAE) as a function of β. Parameter β is plotted in a $\log_{10}$ scale.

n.d.). We use data from 2006 to 2020. The data is recorded every day. We use the DJ30 index to predict the SSE index. We partition our data into sequences of 60 days length and take 80% percent of the data for training (roughly from 2006 to 2017) and the rest for testing. We normalized all the data using the empirical mean and empirical standard deviation from the training set. We use mini batches of size 200.

When training for 2000 epochs and finding the NMAE error as a function β we found that error were still large for training and test data but that a value of $\beta = .2$ could achieve good performance without overfitting. Then wee trained the algorithm with $\beta = .2$ during 10,000 epochs. The NMAE training error was reduced to 0.099. We found that approximation was good for test data points that are close in time to the training data points (data from the first 150 days of the test set), as Figure 4.4 shows. The test NMAE error in data from the first 150 days of the test set is 0.15. We suspect this is because because stock market patterns change and the training set cannot remain valid for test sets far in the future.

4.6 Discussion

In this chapter, we presented an information bottleneck principle for causally predicting a time-series Y^T using a time-series X^T. There are several interesting directions that may

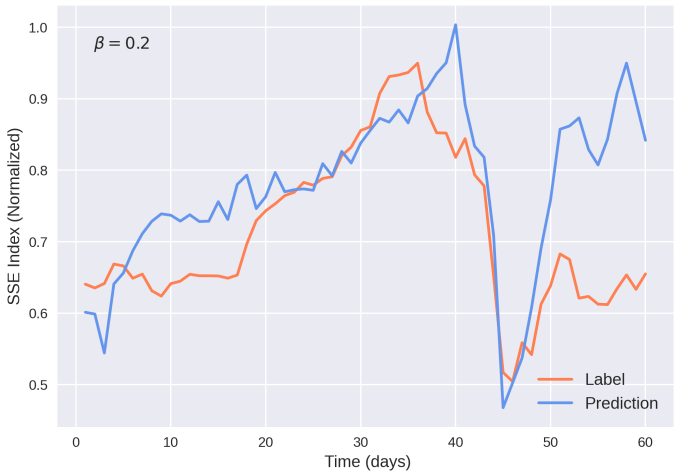


Figure 4.4: Prediction of the SSE index by observing the DJ30 index for a test sequence that happens not long after the last sequence used for training.

be pursued from this starting point.

In our IB principle, we assumed causal features are those that satisfy the Markov chain $Z_t \to X^t \to (X^T, Y^T, Z^{t-1})$, denoted by (M2). This choice lead to a loss function that is easily optimized using RNNs. But one may have picked a smaller class of causal features that satisfy $Z_t \to X_t \to (X^T, Y^T, Z^{t-1})$ (M1), or a larger class where $Z_t \to (X_t, Z^{t-1}) \to X^T$ (M3). D. Xu and Fekri (2018) suggested studying the smaller class by using a standard neutral network to encode X_t into Z_t, and then using an RNN to generate Y_t from Z_t. However, the authors incorrectly assumed the decoder RNN satisfies $q(y^t|z^t) = \prod_{i=1}^{t} q(y_i|z_i)$ (instead of $\prod_{i=1}^{n} q(y_i|z^i)$) leading to a wrong variational bound on the information bottleneck objective. We did not try to correct their algorithm since we found there is no drawback to considering the larger class (M2) from a computational perspective. Class (M3) may actually confer benefits with regards to compressing X^T. This can be implemented using an RNN where the output Z_i is fed into the input at time $i + 1$. However, the loss function becomes significantly more complicated, with an additional Monte Carlo sampling step. We did not investigate this algorithm here—since our algorithm already yielded strong results—but this could be a promising direction to

pursue. Another aspect we did not consider was the presence of side information when predicting Y^T. The simplest example of side information is knowledge of the past; i.e., one may use Y^{t-1} to predict Y, in addition to Z^t. We plan to pursue these lines of work in the future.

Chapter 5

Conclusion

In each chapter in this thesis we addressed a different aspect of neural networks. Here we will present observations we made while working in each of these chapters, and how these observations gave us an idea of how neural networks work.

Recall form chapter 2: If we have data generated by a distribution $\mathbb{P}_X$, our goal is to train an algorithm with loss function ℓ, such that its population risk $L_{\mathbb{P}_X}(w) = \mathbb{E}_{\mathbb{P}_X}\ell(X, w)$ is minimized with respect to the parameters of the model w. However, since we don't have access to $\mathbb{P}_X$ we minimize the empirical risk instead $L_S(w) = \frac{1}{n}\sum_{i=1}^{n}\ell(x_i, w)$ were we approximate $\mathbb{P}_X$ by the empirical uniform distribution. To ensure our result is meaningful we want to make sure that the excess risk: the difference between the empirical risk (the one we compute in practice) and the population risk (our goal) is small. It turns out that the excess risk consists of two terms: the difference between the actual minimum of the empirical risk and the one obtained in practice, and the generalization error. Therefore to have a meaningful algorithm we have to minimize the empirical risk and simultaneously have a small generalization error.

However in practice it happens often that the algorithms are trained by minimizing the empirical risk and the generalization error becomes automatically small.

To understand why this happens recall that training an algorithm with data generated by $\mathbb{P}_S = \mathbb{P}_X^{\otimes n}$ amounts to finding a conditional distribution $\mathbb{P}_{W|S=s}$ defined on the param-

eters of such algorithm. In this case we find $\mathbb{P}_{W|S=s}$ by minimizing $L_S(w)$. Recall also that this will determine a joint distribution $\mathbb{P}_{S,W}$.

What happens is that as A. Xu and M. Raginsky (2017) found and we confirmed, under some assumptions on the loss function and the distribution $\mathbb{P}_X$, the generalized error is bounded in terms of some measure of the "information" between the data and the parameters of the algorithm (either $I(S;W)$ or $W_p^p(\mathbb{P}_S, |Pb_{W|s})$. Then when minimizing $L_S(w)$ it happens that this quantities ($I(S;W)$ or $W_p^p(\mathbb{P}_S, \mathbb{P}_{W|s})$) are also being minimized by the training process. Therefore making the generalization error small.

Another thing to note from chapter 2 is the form of the bounds on the generalization error. As mentioned before both $I(S;W)$ and $W_p^p(\mathbb{P}_S, \mathbb{P}_{W|s})$, are measures of the "information" between the data distribution $\mathbb{P}_S$ and the distribution on the model parameters $\mathbb{P}_W$. And while $I(S;W)$ is easier to compute in practice, $W_p^p(\mathbb{P}_S, \mathbb{P}_{S|w})$ may seem more natural because the restrictions that are imposed to obtain it are only on the loss function and not on the data distribution. Either way, these two quantities are both ways to measure the information between $\mathbb{P}_S$ and $\mathbb{P}_W$ by looking how different are the two distributions $\mathbb{P}_S\mathbb{P}_W$ and $\mathbb{P}_{S,W}$.

This observations leads us to note the following. We use different definitions of information that seem to capture something of how this "information" is being processed through a learning algorithm. But we still don't fully understand how these algorithms work. That may be because we really don't have a formal general concept of information. Future research can look into finding different ways of measuring information where may be looking at how the two distributions $\mathbb{P}_S\mathbb{P}_W$ and $\mathbb{P}_{S,W}$ differ might be a good starting point.

In chapter 3 we turn to the adversarial problem. We found that for a classification task, adversarial examples seem to disappear when the classifier operates only on the data manifold. Our main contribution is that this data manifold was constructed in the simplest way possible. It's simply an interpolation of the training data, no adversarial training or generative models are involved.

This lets us think about adversarial examples from a different perspective: Is not that the network is "misclassifying" adversarial examples. It is that the network's generalization capabilities are so great that the algorithm generalizes even in examples that wouldn't be produced naturally. And under the logic of the network this is how adversarial examples should be classified.

Supporting this perspective is the fact that the data itself has a lot of information that is often overlooked during training. For example consider a image classification task where the training data consist of pictures of animals. There is a lot of information already there: you probably won't find phosphorescent colors on the images, figure with more than two eyes are likely, images near the edges of the picture will not be important. However, in a supervised learning task this side-information is not used for the training, the only thing the network cares about is about the labels. It doesn't matter if we are using images of animals or handwritten numbers the training process, and even the loss functions are the same.

Another thing that makes adversarial examples mysterious is that the way we deem an example adversarial or not is subjective. If an image is misclassified by the network but *for a human it obviously belongs to a specific class* then we call it adversarial. But how can we measure if a human thinks an image belongs to a class? This also poses the matter of how information is processed in different contexts. Is information processed differently in an artificial neural network than in the brain or does it always follow some general principles? How is that a humans can easily classify this image? Is the adversarial example something that only happens in artificial neural networks or is it a general information phenomena?

Remark 5.1. *A possible line of work mixing artificial neural networks with neuroscience research and involving adversarial example could be as follows. We could train a monkey to classify images while doing an online recording of the medial temporal visual cortex (MT). Then train an artificial network to classify the images in the same way the neural activity of the monkey does. So an image instead of being classified as a cat or a dog is classified as: this image produced $f(t)$ neural response, or $g(t)$ neural response. Then generate what would be adversarial examples by maximizing the loss function of our artificial network without moving to far a way form an original image. We then would like to see if these are adversarial images, not for the final image classification, but for the neural activity classification; i.e., if these two similar images (the original and the adversarial) will produce a different neural response. It might be that this adversarial examples also exist in the brain but they are rectified upstream by another mechanism or that they are not at all present.*

Finally in 4 we are forcing the loss function to guarantee that only relevant information from the input is extracted. In here we are using some information that is intrinsic to the data: the fact that we have causal series. We incorporate this by using recurrent neural networks and forcing our distributions to satisfy certain conditions expected from the causality assumption.

What is interesting in here and also in Alemi et al. (2016) is that we don't actually optimize our information bottleneck objective function. We optimize a bound for it, yet in practice this seem to work well. One possibility is that this bound is close to the actual objective function. But another possibility is that the information bottleneck objective function is not what is important. There may be another quantity that is important for the algorithm to work, and it happens that the bound that we are optimizing captures this quantity.

Trying to understand neural networks is trying to understand how information is processed. Tools like mutual information and Wasserstein distance seem to point us in the

right direction not only in the artificial neural network setting but in other situations too.

Remark 5.2. *In W. Huang, X. Huang, and Zhang (2017) an artificial network is constructed to model a biological network. The input is an image and the output is supposed to model the neural response of the MT cortex in a monkey. The artificial network was trained through a unsupervised paradigm by maximizing the mutual information between the input images ant the output neural response. After training the artificial network produced similar responses as those recorded in the monkey showing that the brain is also using mutual information in some capacity, to process information.*

It seem there is some general formal concept of information that governs how neural networks work. Mutual information and Wasserstein distance may be just particular cases of this general concept.

Appendix A

Proofs

Chapter 3 proofs

Lemma A.1. *Let* $f(s,w) := \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, w)$ *then:*

$$\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S}) = \int_{\mathcal{S}\times\mathcal{W}} f(s,w) d\mathbb{P}_S(s) d\mathbb{P}_W(w) - \int_{\mathcal{S}\times\mathcal{W}} f(s,w) d\mathbb{P}_{S,W}(s,w)$$

Proof:

Observe that since the samples X_i are i.i.d., we may equivalently express the population risk as

$$L_{\mathbb{P}_X}(w) := \int_{\mathcal{X}} \ell(x,w) d\mathbb{P}_X(x) = \int_{\mathcal{S}} \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, w) d\mathbb{P}_S(s)$$

Then the generalization error satisfies:

$$\begin{aligned}
\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S}) &= \int_S \int_W [L_{\mathbb{P}_X}(w) - L_s(w)] d\mathbb{P}_{W|s}(w) d\mathbb{P}_S(s) \\
&= \int_S \int_W \left[\int_S \frac{1}{n} \sum_{i=1}^{n} l(x_i, w) d\mathbb{P}_S(s) - \frac{1}{n} \sum_{i=1}^{n} l(x_i, w) \right] d\mathbb{P}_{W|s}(w) d\mathbb{P}_S(s) \\
&= \int_{S \times W} \int_S f(s, w) d\mathbb{P}_S(s) d\mathbb{P}_{S,W}(s, w) - \int_{S \times W} f(s, w) d\mathbb{P}_{S,W}(s, w) \\
&= \int_W \int_S f(s, w) d\mathbb{P}_S(s) d\mathbb{P}_W(w) - \int_{S \times W} f(s, w) d\mathbb{P}_{S,W}(s, w)
\end{aligned}$$

We have concluded that:

$$\int_{\mathcal{S} \times \mathcal{W}} f(s, w) d\mathbb{P}_S(s) d\mathbb{P}_W(w) - \int_{\mathcal{S} \times \mathcal{W}} f(s, w) d\mathbb{P}_{S,W}(s, w) \tag{A.1}$$

Lemma A.2. *Let $\mathcal{S} = \mathcal{X}^n$ be the sample space and $\mathcal{W}$ the parameter space. Let the loss function $\ell : \mathcal{X} \times \mathcal{W} \to \mathbb{R}$ be Lipschitz continuous on $\mathcal{X}$ for every $w \in \mathcal{W}$; i.e.*

$$|\ell(\bar{x}, w) - \ell(x, w)| \leq L||\bar{x} - x||_p, \quad \forall w \in \mathcal{W},$$

for some $p \geq 1$. Let $s = (x_1, \dots x_n) \in \mathcal{S}$ with f defined as:

$$f(s, w) = \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, w).$$

Then for any $s, \bar{s} \in S$ and $w \in \mathcal{W}$, we have:

$$|f(\bar{s}, w) - f(s, w)| \leq \frac{L}{n^{\frac{1}{p}}} ||\bar{s} - s||_p.$$

Proof. Let $x = (x_1, \ldots, x_n)$. From Holder's inequality and for $p \geq 1$ we have that:

$$\begin{aligned} ||x||_1 &= \sum_{i=1}^{n} 1 \cdot |x_i| \\ &\leq \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \left(\sum_{i=1}^{n} (1)^{\frac{p}{p-1}} \right)^{1-\frac{1}{p}} \\ &= ||x||_p n^{1-\frac{1}{p}} \end{aligned}$$

Using this we have that:

$$\begin{aligned} |f(\bar{s}, w) - f(s, w)| &\leq \frac{1}{n} \sum_{i=1}^{n} |\ell(\bar{x}_i, w) - \ell(x_i, w)| \\ &\leq \frac{L}{n} \sum_{i=1}^{n} ||\bar{x}_i - x_i||_p \\ &\leq \frac{L}{n^{\frac{1}{p}}} \left(\sum_{i=1}^{n} ||\bar{x}_i - x_i||_p^p \right)^{\frac{1}{p}} \\ &= \frac{L}{n^{\frac{1}{p}}} ||\bar{s} - s||_p. \end{aligned}$$

□

Theorem A.3. *Let $\mathcal{S} = \mathcal{X}^n$ be the sample space and $\mathcal{W}$ be the parameter space. Let $\mathbb{P}_{W|S}$ be an algorithm. and let $\mathbb{P}_{S,W}$ be the distribution on $\mathcal{S} \times \mathcal{W}$ induced by the algorithm and the distribution on the data ($\mathbb{P}_{S,W} = \mathbb{P}_S \mathbb{P}_{W|S}$). If the loss function ℓ is Lipschitz continuous on $\mathcal{X}$ for every $w \in \mathcal{W}$; Then:*

$$|\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S})| \leq \frac{L}{n^{\frac{1}{p}}} \left(\int_W W_p^p(\mathbb{P}_S, \mathbb{P}_{S|w}) d\mathbb{P}_W(w) \right)^{\frac{1}{p}}.$$

Proof. From lemma A.1 recall that the generalization error equals

$$\begin{aligned} \mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S}) = &\int_{\mathcal{S} \times \mathcal{W}} f(s, w) d\mathbb{P}_S(s) d\mathbb{P}_W(w) \\ &- \int_{\mathcal{S} \times \mathcal{W}} f(s, w) d\mathbb{P}_{S,W}(s, w) \end{aligned} \tag{A.2}$$

where $s = (x_1, \dots x_n)$ and $f(s, w) = \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, w)$. From now on, we are going to work in the space $(\mathcal{S} \times \mathcal{W}) \times (\mathcal{S} \times \mathcal{W})$. To avoid confusion, we will denote as $\bar{\mathcal{S}} \times \bar{\mathcal{W}}$ the space where the distribution is the product measure $\mathbb{P}_S\mathbb{P}_W$, and we will denote elements on this space as $(\bar{s}, \bar{w})$. Let π be any distribution in $\Pi(\mathbb{P}_S\mathbb{P}_W, \mathbb{P}_{S,W})$; i.e., the marginals of π on $(\bar{\mathcal{S}} \times \bar{\mathcal{W}})$ and $(\mathcal{S} \times \mathcal{W})$ are $\mathbb{P}_S\mathbb{P}_W$ and $\mathbb{P}_{S,W}$ respectively. Therefore we have that:

$$\begin{aligned} &\int_{\bar{\mathcal{S}}\times\bar{\mathcal{W}}} f(\bar{s}, \bar{w}) d\mathbb{P}_S(\bar{s}) d\mathbb{P}_W(\bar{w}) - \int_{\mathcal{S}\times\mathcal{W}} f(s, w) d\mathbb{P}_{S,W}(s, w) \\ &\qquad = \int_{(\bar{\mathcal{S}}\times\bar{\mathcal{W}})\times(\mathcal{S}\times\mathcal{W})} f(\bar{s}, \bar{w}) - f(s, w)\ d\pi(\bar{s}, \bar{w}, s, w). \end{aligned} \tag{A.3}$$

From (A.2) and (A.3) we get:

$$\begin{aligned} &|\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S})| \\ &\leq \int_{(\bar{\mathcal{S}}\times\bar{\mathcal{W}})\times(\mathcal{S}\times\mathcal{W})} |f(\bar{s}, \bar{w}) - f(s, w)|\ d\pi(\bar{s}, \bar{w}, s, w). \end{aligned} \tag{A.4}$$

Observe that (A.4) is valid for any $\pi \in \Pi(\mathbb{P}_S\mathbb{P}_W, \mathbb{P}_{S,W})$. Thus, we may choose an appropriate coupling π_0 that makes the right hand side small thus strengthening the upper bound which ultimately will lead to the conclusion of the theorem.

Construction of π_0:

For each $w \in \mathcal{W}$ consider the family of distributions $\Pi_w(\mathbb{P}_S, \mathbb{P}_{S|w})$ on $\bar{\mathcal{S}} \times \mathcal{S}$. Let π_w be the distribution that achieves the Wasserstein metric (as mentioned before, such measure exists from Villani 2003); i.e.:

$$W_p^p(\mathbb{P}_S, \mathbb{P}_{S|w}) = \int_{\bar{\mathcal{S}}\times\mathcal{S}} ||\bar{s} - s||_p^p d\pi_w(\bar{s}, s). \tag{A.5}$$

The idea to generate a coupling π_0 is as follows: Generate W according to $\mathbb{P}_W$, and set $\bar{W} = W$. Using the realized value w of W, generate the pair $(\bar{S}, S)$ according the π_w defined above.

More precisely, let $\bar{\pi}_0$ be the distribution on $\bar{\mathcal{S}} \times \mathcal{S} \times \mathcal{W}$ induced by $\mathbb{P}_W$ and the family of distribution $\{\pi_w\}_{w \in \mathcal{W}}$ on $\bar{\mathcal{S}} \times \mathcal{S}$, i.e. $\bar{\pi}_0(\bar{s}, s, w) = \pi_w(\bar{s}, s)\mathbb{P}_W(w)$. We will define $\pi_0 \in \bar{\mathcal{S}} \times \bar{\mathcal{W}} \times \mathcal{S} \times \mathcal{W}$ on terms of $\bar{\pi}_0$: Let D be the diagonal of $\bar{\mathcal{W}} \times \mathcal{W}$ and let $p(\bar{w}, w) = w$ be the canonical projection. (It does not really matter if the projection is on $\bar{\mathcal{W}}$ or $\mathcal{W}$, as it will only be evaluated on D). For an event A of $\bar{\mathcal{S}} \times \mathcal{S}$ and B an event of $\bar{\mathcal{W}} \times \mathcal{W}$, define

$$\pi_0(A \times B) = \bar{\pi}_0(A \times p(B \cap D)). \tag{A.6}$$

In other words, the measure π_0 is assigning a measure of zero to all events of $\bar{\mathcal{S}} \times \bar{\mathcal{W}} \times \mathcal{S} \times \mathcal{W}$ that are off the diagonal of $\bar{\mathcal{W}} \times \mathcal{W}$. And is assigning measure $\bar{\pi_0}$ to events intersecting the diagonal. (Note that by Carathéodory's extension theorem it is enough to define π_0 in the rectangles of $(\bar{\mathcal{S}} \times \bar{\mathcal{W}} \times \mathcal{S} \times \mathcal{W})$).

Finally, we argue that π_0 constructed as above lies in $\Pi(\mathbb{P}_S\mathbb{P}_W, \mathbb{P}_{S,W})$. Let $A \subseteq \bar{\mathcal{S}}$ and $B \subseteq \bar{\mathcal{W}}$. Then:

$$\begin{aligned}
\pi_0(A \times B \times \mathcal{S} \times \mathcal{W}) &= \bar{\pi}_0(A \times \mathcal{S} \times p([B \times \mathcal{W}] \cap \mathcal{D})) \\
&= \bar{\pi}_0(A \times \mathcal{S} \times B) \\
&= \int_B \int_{A \times \mathcal{S}} d\pi_w(\bar{s}, s) d\mathbb{P}_W(w) \\
&\overset{(a)}{=} \int_B \int_A d\mathbb{P}_S(\bar{s}) d\mathbb{P}_W(w) \\
&= \mathbb{P}_S\mathbb{P}_W(A, B),
\end{aligned}$$

where (a) follows since $\pi_w \in \Pi(\mathbb{P}_S, \mathbb{P}_{S|w})$. Similarly if $A \subseteq \mathcal{S}$ and $B \subseteq \mathcal{W}$. Then:

$$\begin{aligned}
\pi_0(\bar{\mathcal{S}} \times \bar{\mathcal{W}} \times A \times B) &= \bar{\pi}_0(\bar{\mathcal{S}} \times A \times p([\bar{\mathcal{W}} \times B] \cap D)) \\
&= \bar{\pi}_0(\bar{\mathcal{S}} \times A \times B) \\
&= \int_B \int_{\bar{\mathcal{S}} \times A} d\pi_w(\bar{s}, s) d\mathbb{P}_W(w) \\
&\overset{(a)}{=} \int_B \int_A d\mathbb{P}_{S|w}(s) d\mathbb{P}_W(w) \\
&= \mathbb{P}_{S,W}(A, B),
\end{aligned}$$

where (a) follows since $\pi_w \in \Pi(\mathbb{P}_S, \mathbb{P}_{S|w})$.

Now that we constructed π_0, going back to (A.4) we get that:

$$\begin{aligned}
&|\mathcal{E}(\mathbb{P}_S, \mathbb{P}_{W|S})| \\
&\leq \int_{(\bar{\mathcal{S}} \times \bar{\mathcal{W}}) \times (\mathcal{S} \times \mathcal{W})} |f(\bar{s}, \bar{w}) - f(s, w)| \, d\pi_0(\bar{s}, \bar{w}, s, w) \\
&\overset{(a)}{=} \int_{(\bar{\mathcal{S}} \times \bar{\mathcal{W}}) \times (\mathcal{S} \times \mathcal{W})} \mathbb{1}_{\{\bar{w}=w\}} |f(\bar{s}, \bar{w}) - f(s, w)| \, d\pi_0(\bar{s}, \bar{w}, s, w) \\
&\overset{(b)}{=} \int_{\bar{\mathcal{S}} \times \mathcal{S} \times \mathcal{W}} |f(\bar{s}, w) - f(s, w)| d\bar{\pi}_0(\bar{s}, s, w) \\
&\overset{(c)}{=} \int_{\bar{\mathcal{S}} \times \mathcal{S} \times \mathcal{W}} |f(\bar{s}, w) - f(s, w)| d\pi_w(\bar{s}, s) d\mathbb{P}_W(w) \\
&\overset{(d)}{\leq} \int_{\bar{\mathcal{S}} \times \mathcal{S} \times \mathcal{W}} \frac{L}{n^{\frac{1}{p}}} ||\bar{s} - s||_p d\pi_w(\bar{s}, s) d\mathbb{P}_W(w) \\
&\overset{(e)}{\leq} \frac{L}{n^{\frac{1}{p}}} \left(\int_{\bar{\mathcal{S}} \times \mathcal{S} \times \mathcal{W}} ||\bar{s} - s||_p^p d\pi_w(\bar{s}, s) d\mathbb{P}_W(w) \right)^{\frac{1}{p}} \\
&\overset{(f)}{=} \frac{L}{n^{\frac{1}{p}}} \left(\int_W W_p^p(\mathbb{P}_S, \mathbb{P}_{S|w}) d\mathbb{P}_W(w) \right)^{\frac{1}{p}}
\end{aligned}$$

where (a) and (b) follows from the definition of π_0 in (A.6), (c) follows from the definition of $\bar{\pi}_0$, (d) follows by Lemma A.2, (e) follows from Jensen's inequality, and (f) follows from the choice of π_w given by equation (A.5). $\square$

Lemma A.4. *If μ satisfies $T_p(c)$, then $\mu^{\otimes n}$ satisfies $T_p(cn^{2/p-1})$.*

Proof. Proof in Maxim Raginsky, Sason, et al. 2013. □

Lemma A.5. *Let $\mathbb{P}_{S,W} = \mathbb{P}_{S|W}\mathbb{P}_W$ a distribution on $\mathcal{S} \times \mathcal{W}$ then:*

$$\frac{d\mathbb{P}_{S|W}}{d\mathbb{P}_S} = \frac{d\mathbb{P}_{S,W}}{d\mathbb{P}_S\mathbb{P}_W}$$

Proof. By the definition of Radon-Nikodin derivative we have that for any measurable set $B \subset W$:

$$\int_B \frac{d\mathbb{P}_{S|W}}{d\mathbb{P}_S} d\mathbb{P}_S = \int_B d\mathbb{P}_{S|W}$$

Integrating bot sides over an arbitrary measurable set $A \subset W$ with respect to $\mathbb{P}_W$ we get:

$$\begin{aligned}\int_A \int_B \frac{d\mathbb{P}_{S|W}}{d\mathbb{P}_S} d\mathbb{P}_S d\mathbb{P}_W &= \int_A \int_B d\mathbb{P}_{S|W} d\mathbb{P}_W \\ &= \int_A \int_B d\mathbb{P}_{S,W}\end{aligned}$$

Which tells us that for any measurable rectangle $A \times B$ the function $\frac{d\mathbb{P}_{S|W}}{d\mathbb{P}_S}$ is the Radon-Nikodin derivative $\frac{d\mathbb{P}_{S,W}}{d\mathbb{P}_S\mathbb{P}_W}$. But because of Carathéodory's extension theorem this equality then also holds for any measurable sets of $\mathcal{S} \times \mathcal{W}$ concluding the proof.

□

Theorem A.6. *Assume that $\mathbb{P}_X$ satisfies $T_p(c)$ for some $p \in [1, 2]$ and some $c > 0$. Then*

$$\frac{L}{n^{\frac{1}{p}}} \left(\int_W W_p^p(\mathbb{P}_S, \mathbb{P}_{S|w}) d\mathbb{P}_W(w) \right)^{\frac{1}{p}} \leq L\sqrt{\frac{2c}{n} I(\mathbb{P}_S; \mathbb{P}_W)} \tag{A.7}$$

Proof. First we show that

$$I(\mathbb{P}_S;\mathbb{P}_W) = \int_{\mathcal{W}} D(\mathbb{P}_{S|W=w}||\mathbb{P}_S)d\mathbb{P}_W(w). \tag{A.8}$$

Using the definition off mutual information in terms of the KL- divergence:

$$\begin{aligned}
'I(\mathbb{P}_S;\mathbb{P}_W) &= D(\mathbb{P}_{S,W}||\mathbb{P}_S\mathbb{P}_W) \\
&= \int_{\mathcal{S}\times\mathcal{W}} \ln\left(\frac{d\mathbb{P}_{S,W}}{d\mathbb{P}_S\mathbb{P}_W}\right) d\mathbb{P}_{S,W} \\
&\overset{(a)}{=} \int_{\mathcal{S}\times\mathcal{W}} \ln\left(\frac{d\mathbb{P}_{S|W}}{d\mathbb{P}_S}\right) d\mathbb{P}_{S,W} \\
&= \int_{\mathcal{W}} D(\mathbb{P}_{S|W}||\mathbb{P}_S)d\mathbb{P}_W.
\end{aligned}$$

where (a) is from lemma A.5.

Now by Lemma A.4, we have that $\mathbb{P}_S$ satisfies a $T_p(cn^{2/p-1})$ inequality. We now have the following sequence of inequalities:

$$\begin{aligned}
I(W;S) &= \int_{\mathcal{W}} D(\mathbb{P}_{S|W=w}||\mathbb{P}_S)d\mathbb{P}_W(w) \\
&\overset{(a)}{\geq} \int_{\mathcal{W}} \frac{W_p(\mathbb{P}_{S|W=w},\mathbb{P}_S)^2}{2cn^{2/p-1}} d\mathbb{P}_W(w) \\
&= \frac{1}{2cn^{2/p-1}} \int_{\mathcal{W}} \left(W_p(\mathbb{P}_{S|W=w},\mathbb{P}_S)^p\right)^{2/p} d\mathbb{P}_W(w) \\
&\overset{(b)}{\geq} \frac{1}{2cn^{2/p-1}} \left(\int_{\mathcal{W}} W_p(\mathbb{P}_{S|W=w},\mathbb{P}_S)^p d\mathbb{P}_W(w)\right)^{2/p}.
\end{aligned}$$

Here, (a) follows from the transportation-cost inequality for $\mathbb{P}_S$ and (b) follows from the Jensen's inequality and the fact that $2/p \geq 1$. Taking square-roots and simplifying, this yields

$$\sqrt{\frac{2cI(W;S)}{n}} \geq \frac{1}{n^{1/p}} \left(\int_{\mathcal{W}} W_p(\mathbb{P}_{S|W=w},\mathbb{P}_S)^p d\mathbb{P}_W(w)\right)^{1/p}.$$

Multiplying both sides by L completes the proof. □

Chapter 4 proofs

Theorem A.7. *Let $\beta > 0$. Let (X^T, Z^T, Y^T), be random variables with joint probability density $p(x^T, y^T, z^T)$, where p satisfies the causal encoder property. Let $q(y^T, z^T)$ be a probability density satisfying the causal decoder property, and let $r(z^T)$ be any probability density satisfying $r(z^T) = \prod_{t=1}^{T} r(z_t)$. Then we have that:*

$$\begin{aligned} I(Y^T; Z^T) - \beta I(X^T; Z^T) \geq & \sum_{t=1}^{T} \int p(x^t, y_t) p(z^t|x^t) \ln q(y_t|z^t)\, dx^t dy_t\, dz^t \\ & -\beta \sum_{t=1}^{T} \int p(x^t) p(z_t|x^t) p(x^t) \ln \left(\frac{p(z_t|x^t)}{r(z_t)} \right) dx^t dz_t \\ & + H(Y^T). \end{aligned}$$

Proof. The proof of Theorem A.7 proceeds in two steps. We first lower bound the $I(Y^T; Z^T)$ term, and then we upper bound the $I(X^T; Z^T)$ term.

Lower bound on $I(Z^T; Y^T)$: We can write $I(Z^T; Y^T)$ as:

$$\begin{aligned} & \int p(y^T, z^T) \ln \left(\frac{p(y^T|z^T)}{p(y^T)} \right) dy^T dz^T \\ & = \int p(y^T, z^T) \ln \left(\frac{p(y^T|z^T)}{p(y^T)} \right) dy^T dz^T + \int p(y^T, z^T) \ln \left(\frac{q(y^T|z^T)}{p(y^T)} \right) dy^T dz^T \\ & \qquad - \int p(y^T, z^T) \ln \left(\frac{q(y^T|z^T)}{p(y^T)} \right) dy^T dz^T \end{aligned} \tag{A.9}$$

Here we are only adding and subtracting the same term to $I(Z^T, Y^T)$. Looking at the the first and third terms on the right hand side in (A.9) we find they are positive:

$$\begin{aligned}
&\int p(y^T, z^T) \ln\left(\frac{p(y^T, z^T)}{p(y^T)p(z^T)}\right) dy^T dz^T - \int p(y^T, z^T) \ln\left(\frac{q(y^T, z^T)}{p(y^T)q(z^T)}\right) dy^T dz^T \\
&= \int p(y^T, z^T) \ln\left(\frac{p(y^T, z^T)p(y^T)q(z^T)}{q(y^T, z^T)p(y^T)p(z^T)}\right) dy^T dz^T \\
&= \int p(y^T|z^T)p(z^T) \ln\left(\frac{p(y^T|z^T)}{q(y^T|z^T)}\right) dy^T dz^T \\
&= \int D(p(y^T|z^T)||q(y^T|z^T))p(z^T) dz^T \qquad \text{(A.10)}
\end{aligned}$$

Since the KL divergence is positive for every value of z^t, the integral (A.10) is positive. Putting (A.9) and (A.10) together we conclude that:

$$\begin{aligned}
I(Y^T; Z^T) &\geq \int p(y^T, z^T) \ln\left(\frac{q(y^T|z^T)}{p(y^T)}\right) dy^T dz^T \qquad \text{(A.11)} \\
&= \int p(y^T, z^T) \ln q(y^T|z^T) dy^T dz^T + H(Y^T). \qquad \text{(A.12)}
\end{aligned}$$

Since q satisfies the causal decoder property, from remark 4.6, we get that $q(y^T|z^T) = \prod_{t=1}^{T} q(y_t|z^t)$. Using this on the first term in (A.12) we get:

$$\begin{aligned}
\int p(y^T, z^T) \ln q(y^T|z^T)\, dyT dz^T &= \int p(y^T, z^T) \sum_{t=1}^{T} \ln q(y_t|z^t) dy^T dz^T \\
&= \sum_{t=1}^{T} \int p(y_t, z^t) \ln q(y_t|z^t) dy_t\, dz^t. \qquad \text{(A.13)}
\end{aligned}$$

Simultaneously observe that:

$$\begin{aligned}
p(y_t, z^t) &= \int p(x^t, y_t, z^t)\, dx^t, \\
&= \int p(z^t|x^t, y_t)p(x^t, y_t)\, dx^t, \\
&= \int p(z^t|x^t)p(x^t, y_t)\, dx^t, \qquad \text{(A.14)}
\end{aligned}$$

where the last equality follows from the Markov property $Z^t \to X^t \to Y^t$ of the causal

encoder. Substituting equalities (A.13) and (A.14) in the bound (A.12), we get the lower bound:

$$I(Y^T;Z^T) \geq \sum_{t=1}^{T} \int p(z^t|x^t)p(x^t,y_t)\ln q(y_t|z^t)\,dx^t dy_t\,dz^t + H(Y^T). \tag{A.15}$$

Upper bound on $I(X^T;Z^T)$: Using a similar strategy as above, we have

$$\begin{aligned} I(X^T;Z^T) &= \int p(x^T,z^T)\ln\left(\frac{p(z^T|x^T)}{p(z^T)}\right)dx^T dz^T \\ &= \int p(x^T,z^T)\ln p(z^T|x^T)\,dx^T dz^T - \int p(z^T)\ln p(z^T)\,dz^T \end{aligned} \tag{A.16}$$

Since $D(p(z^T)||r(z^T)) \geq 0$ we have that $\int p(z^T)\ln p(z^T)\,dz^T \geq \int p(z^T)\ln r(z^T)\,dz^T$. Substituting this in (A.16) we get:

$$\begin{aligned} I(X^T;Z^T) &\leq \int p(x^T,z^T)\ln p(z^T|x^T)\,dx^T dz^T - \int p(z^T)\ln r(z^T)\,dz^T \\ &= \int p(x^T,z^T)\ln\left(\frac{p(z^T|x^T)}{r(z^T)}\right)dx^T dz^T \\ &\leq \int p(x^T,z^T)\ln\left(\frac{p(z^T|x^T)}{\prod_{t=1}^{T} r(z_t)}\right)dx^T dz^T, \end{aligned} \tag{A.17}$$

where we used the product form $r(z^T) = \prod_{t=1}^{T} r(z_t)$ in the las equality. Since p satisfies the causal encoder property, from remark 4.6, we have $p(z^T|x^T) = \prod_{t=1}^{T} p(z_t|x^t)$. Substituting in (A.17) we get the upper bound:

$$I(X^T;Z^T) \leq \sum_{t=1}^{T} \int p(z_t|x^t)p(x^t)\ln\left(\frac{p(z_t|x^t)}{r(z_t)}\right)dx^t dz_t.$$

Putting both bounds together, we obtain the statement of the theorem. □

Bibliography

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25, pp. 1097–1105.

Rao, Qing and Jelena Frtunikj (2018). "Deep learning for self-driving cars: Chances and challenges". In: *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, pp. 35–38.

Silver, David et al. (2017). "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815*.

Vargas, Manuel R, Beatriz SLP De Lima, and Alexandre G Evsukoff (2017). "Deep learning for stock market prediction from financial news articles". In: *2017 IEEE international conference on computational intelligence and virtual environments for measurement systems and applications (CIVEMSA)*. IEEE, pp. 60–65.

Hardt, M., B. Recht, and Y. Singer (20–22 Jun 2016). "Train faster, generalize better: Stability of stochastic gradient descent". In: *Proceedings of The 33rd International Conference on Machine Learning*. Vol. 48. New York, New York, USA: PMLR, pp. 1225–1234.

Belkin, Mikhail, Siyuan Ma, and Soumik Mandal (2018). "To understand deep learning we need to understand kernel learning". In: *International Conference on Machine Learning*. PMLR, pp. 541–549.

Neyshabur, Behnam, Ryota Tomioka, and Nathan Srebro (2014). "In search of the real inductive bias: On the role of implicit regularization in deep learning". In: *arXiv preprint arXiv:1412.6614*.

Daniely, Amit, Roy Frostig, and Yoram Singer (2016). "Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity". In: *Advances In Neural Information Processing Systems* 29, pp. 2253–2261.

Daniely, Amit (2017). "Depth separation for neural networks". In: *Conference on Learning Theory*. PMLR, pp. 690–696.

Eldan, Ronen and Ohad Shamir (2016). "The power of depth for feedforward neural networks". In: *Conference on learning theory*. PMLR, pp. 907–940.

Telgarsky, Matus (2015). "Representation benefits of deep feedforward networks". In: *arXiv preprint arXiv:1509.08101*.

Papernot, Nicolas and Patrick McDaniel (2018). "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning". In: *arXiv preprint arXiv:1803.04765*.

Ilyas, A. et al. (2017). "The robust manifold defense: Adversarial training using generative models". In: *arXiv preprint arXiv:1712.09196*.

Gilmer, Justin et al. (2018). "Adversarial spheres". In: *arXiv preprint arXiv:1801.02774*.

Raginsky, M. et al. (Sept. 2016). "Information-theoretic analysis of stability and bias of learning algorithms". In: *2016 IEEE Information Theory Workshop (ITW)*, pp. 26–30. DOI: 10.1109/ITW.2016.7606789.

Samangouei, Pouya, Maya Kabkab, and Rama Chellappa (2018). "Defense-gan: Protecting classifiers against adversarial attacks using generative models". In: *arXiv preprint arXiv:1805.06605*.

Athalye, A., N. Carlini, and D. Wagner (2018). "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples". In: *arXiv preprint arXiv:1802.00420*.

Alemi, A. A. et al. (2016). "Deep variational information bottleneck". In: *arXiv:1612.00410*.

Xu, D. and F. Fekri (2018). "Time Series Prediction Via Recurrent Neural Networks with the Information Bottleneck Principle". In: *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, pp. 1–5.

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley-Interscience.

Shalev-Shwartz, S. and S. Ben-David (2014). *Understanding Machine Learning: From Theory to Algorithms*. New York, NY, USA: Cambridge University Press.

Bousquet, O. and A. Elisseeff (Mar. 2002). "Stability and Generalization". In: *Journal of Machine Learning Research* 2, pp. 499–526.

Elisseeff, A., T. Evgeniou, and M. Pontil (Dec. 2005). "Stability of Randomized Learning Algorithms". In: *Journal of Machine Learning Research* 6, pp. 55–79.

Mukherjee, S. et al. (July 2006). "Learning theory: Stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization". In: *Advances in Computational Mathematics* 25, pp. 161–193.

London, B. (2016). "Generalization bounds for randomized learning with application to stochastic gradient descent". In: *NIPS Workshop on Optimizing the Optimizers*.

Mou, W. et al. (2017). "Generalization Bounds of SGLD for Non-convex Learning: Two Theoretical Viewpoints". In: *arXiv preprint:1707.05947*. arXiv: 1707.05947.

Russo, D. and J. Zou (2016). "Controlling Bias in Adaptive Data Analysis Using Information Theory". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 1232–1240.

Xu, A. and M. Raginsky (2017). "Information-theoretic analysis of generalization capability of learning algorithms". In: *Advances in Neural Information Processing Systems*, pp. 2521–2530.

Pensia, Ankit, Varun Jog, and Po-Ling Loh (2018). "Generalization Error Bounds for Noisy, Iterative Algorithms". In: *arXiv preprint arXiv:1801.04295*.

Bassily, Raef et al. (2018). "Learners that Use Little Information". In: *Algorithmic Learning Theory*, pp. 25–55.

Asadi, Amir R, Emmanuel Abbe, and Sergio Verdú (2018). "Chaining Mutual Information and Tightening Generalization Bounds". In: *arXiv preprint arXiv:1806.03803*.

Villani, Cédric (2003). *Topics in optimal transportation*. 58. American Mathematical Soc.

Raginsky, Maxim, Igal Sason, et al. (2013). "Concentration of measure inequalities in information theory, communications, and coding". In: *Foundations and Trends® in Communications and Information Theory* 10.1-2, pp. 1–246.

Welling, M. and Y. W. Teh (2011). "Bayesian learning via stochastic gradient Langevin dynamics". In: *Proceedings of the 28th International Conference on Machine Learning*, pp. 681–688.

Ge, R. et al. (2015). "Escaping from saddle points—Online stochastic gradient for tensor decomposition". In: *Conference on Learning Theory*, pp. 797–842.

Jin, C. et al. (2017). "How to Escape Saddle Points Efficiently". In: *arXiv preprint:1703.00887*.

Cover, Thomas M and Joy A Thomas (2012). *Elements of information theory*. John Wiley & Sons.

Smola, A. J. and B. Schölkopf (1998). *Learning with Kernels*. GMD-Forschungszentrum Informationstechnik.

Roweis, S. T. and L. K. Saul (2000). "Nonlinear dimensionality reduction by locally linear embedding". In: *Science* 290.5500, pp. 2323–2326.

Ghodsi, A. (2006). "Dimensionality reduction a short tutorial". In: *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada* 37, p. 38.

Kingma, D. P. and M. Welling (2013). "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114*.

Goodfellow, I. et al. (2014). "Generative adversarial nets". In: *Advances in neural information processing systems*, pp. 2672–2680.

Vincent, P. et al. (2008). "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 1096–1103.

Bora, A. et al. (2017). "Compressed sensing using generative models". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 537–546.

Jalali, S. and X. Yuan (2019). "Using auto-encoders for solving ill-posed linear inverse problems". In: *arXiv preprint arXiv:1901.05045*.

Creswell, A. and A. A. Bharath (2018). "Inverting the generator of a generative adversarial network". In: *IEEE transactions on neural networks and learning systems*.

Lipton, Z. C. and S. Tripathi (2017). "Precise recovery of latent vectors from generative adversarial networks". In: *arXiv preprint arXiv:1702.04782*.

Szegedy, C. et al. (2013). "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199*.

Samangouei, P., M. Kabkab, and R. Chellappa (2018). "Defense-gan: Protecting classifiers against adversarial attacks using generative models". In: *arXiv preprint arXiv:1805.06605*.

Cayton, L. (2005). "Algorithms for manifold learning". In: *Univ. of California at San Diego Tech. Rep* 12.1-17, p. 1.

Danskin, J. M. (2012). *The theory of max-min and its application to weapons allocation problems*. Vol. 5. Springer Science & Business Media.

Madry, A. et al. (2017). "Towards deep learning models resistant to adversarial attacks". In: *arXiv preprint arXiv:1706.06083*.

Oliphant, Travis E (2006). *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Tishby, N., F. C. Pereira, and W. Bialek (2000). "The information bottleneck method". In: *arXiv preprint physics/0004057*.

Slonim, N. and N. Tishby (2000). "Document clustering using word clusters via the information bottleneck method". In: *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, pp. 208–215.

Chechik, G. and N. Tishby (2003). "Extracting relevant structures with side information". In: *Advances in NIPS*, pp. 881–888.

Gondek, D. and T. Hofmann (2003). "Conditional information bottleneck clustering". In: *3rd IEEE ICDM, Workshop on Clustering Large Data Sets*. Citeseer, pp. 36–42.

Chalk, M., O. Marre, and G. Tkacik (2016). "Relevant sparse codes with variational information bottleneck". In: *Advances in NIPS*, pp. 1957–1965.

Wang, W. et al. (2016). "Deep variational canonical correlation analysis". In: *arXiv preprint arXiv:1610.03454*.

Poole, B. et al. (2019). "On variational bounds of mutual information". In: *International Conference on Machine Learning*. PMLR, pp. 5171–5180.

Pensia, A., V. Jog, and P. Loh (2020). "Extracting robust and accurate features via a robust information bottleneck". In: *IEEE Journal on Selected Areas in Information Theory* 1.1, pp. 131–144.

Sezer, O. B., M. U. Gudelek, and A. M. Ozbayoglu (2020). "Financial time series forecasting with deep learning: A systematic literature review: 2005–2019". In: *Applied Soft Computing* 90, p. 106181.

Sutskever, I. (2013). *Training recurrent neural networks*. University of Toronto Toronto, Canada.

Cover, T. M. and J. A. Thomas (2012). *Elements of Information Theory*. John Wiley & Sons.

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Cho, K. et al. (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*.

Weather dataset from the Max Planck Institute for BioGeochemistry (n.d.). `https://www.bgc-jena.mpg.de/wetter/`. Accessed: February 3, 2020.

Yahoo finance (n.d.). `https://finance.yahoo.com`. Accessed: February 3, 2020.

Huang, Wentao, Xin Huang, and Kechen Zhang (2017). "Information-theoretic interpretation of tuning curves for multiple motion directions". In: *2017 51st Annual Conference on Information Sciences and Systems (CISS)*. IEEE, pp. 1–4.