

# Distribution and Histogram (DisH) Learning

Junier Bárbaro Oliva

JULY 2018

CMU-ML-18-105

Machine Learning Department  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213

**Thesis Committee:**

Barnabás Póczos, Co-Chair  
Jeff Schneider, Co-Chair  
Ruslan Salakhutdinov  
Le Song, Georgia Tech

*Submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy.*

Copyright © 2018 Junier Bárbaro Oliva

This research was supported by: the Office of Naval Research award number N000140610104; the Army Research Laboratory award number W911NF1020022; the National Science Foundation award numbers IIS1247658, IIS1250350, IIS1563887; and Foxconn Technology Group award number A019975.

**Keywords:** Distributions, Sets, Sequences, Nonparametric, Statistics, Machine Learning

*Dedicated to my beautiful, loving wife.*





# Acknowledgements

If you are the type that hates sappy acknowledgement sections, then read no further.

I would be nowhere without my family. My parents instilled a love for learning in me, and made such great sacrifices to get me to a country where one may succeed regardless of political affiliations. *Gracias por todo Mami y Papi*. Of course my grandparents were also pivotal, Abuela Cucha did nothing but shower me with love, and Abuelo Nelo scraped together the little he had to buy me my first computer in 1998, getting me started on this whole journey. My sister, Janet, her husband, Jorge, and my niece, Joanna, have always been so supportive and loving.

Almost a year ago, I extended my family to include my lovely wife Gabriela. She has had to bare the brunt of my PhD related stress and long hours, but she has been nothing but supportive (well, at least *after* we got off that bus). Sincerely, she has been a source of love, support, and inspiration for so many years; she is my best friend, and life partner in every sense of the word. I would be remiss if I didn't also thank my in-laws—Dennis, Susan, and Nicole—who are now also family and have lent their ears to my whining about grad school through the years.

It's almost unbelievable how great my advisors, Barnabás Póczos and Jeff Schneider, have been. They have been terrific mentors, collaborators, and friends. They have encouraged me to publish and seek interesting directions, while never micro-managing and allowing me to grow as my own researcher. I can honestly say that Barnabás is the kindest, most easy-going advisor that I could of asked for. Jeff is a master of how to explain and sell ideas, and a resource for advice at all layers of research. Both have made my PhD journey as painless as possible by cutting through any nonsense and allowing me to focus on the fun part, the research! I must also thank the rest of my committee Ruslan Salakhutdinov, and Le Song. Their comments and insights have made my thesis not just a finish line to cross, but a process that has allowed me to grow as a researcher.

Before even reaching CMU, I was fortunate to have two great advisors in high-school, Joseph Walpole, and Juan Catala. Mr. Walpole was (and is) the greatest English teacher to live; his classroom was a proving ground for ideas that shaped me as a thinker. Mr. Catala was a great physics teacher, but more than that he was a cheerleader that made a boy from Hialeah believe that he could make it in a top university like CMU. Carnegie Mellon itself deserves a great deal of recognition. Many years ago, as a recent high-school graduate, CMU provided the first place where I felt at home intellectually. Today, the Machine Learning Department and the university as a whole has been the perfect environment for me. In large part, this is due to the great faculty and collaborators I've been able to work with,

including: Artur Dubrawski, Aarti Singh, Geoff Gordon, Tom Mitchell, and Eric Xing. The whole administrative staff, and Diane Stidle have also done a marvelous job helping grad students feel at home in the department.

I was so fortunate to have been a part of the best cohort in the Machine Learning Department (a biased, but very low variance opinion). Wei Dai (*Wei-dog*) was always the optimist, achieving an MSE of 4 in predicting our graduation date. Kirstin Early (*K-town*) has been the best supporter of my horrible jokes. Willy Neiswanger (*Willow*) always provided comic-relief and fun. Micol Marchetti-Bowick (*Ultra*) has mastered a brand of dry-humor that always cracks me up. Nicole Rafidi (*Slaw*) has been a great co-lifecoach. Ben Cowley (*Benjo*) and I have had countless great conversations about nonsense. There are also some folks that might as well have been a part of the cohort: Avinava Dubey (*Llama*) has taken dad-jokes to new heights previously thought not possible, and Calvin Murdock (*Cal-town*) has shared my same sensibilities on shenanigans (to a fault, I would say). Of course, so many more great friends have made my time as a PhD student incredible, including (but not limited to!): Aaditya Ramdas, Pete Lund, Dougal Sutherland, Liang Xiong, Maria De Arteaga, Alex Beutel, Mariya Toneva, Dan Howarth, Benedikt Boecking, Manzil Zaheer, Jing Xiang, Kirthevasan Kandasamy, Anthony Platanios, and Ankur Parikh.

Lastly, I absolutely must thank the amazing foundations that allowed me to study at CMU as an undergraduate without loans. The Gates foundation, the Hispanic Scholarship fund, and the Jorge Mas Canosa Freedom Foundation, gave me the scholarships that were pivotal to me pursuing graduate school without financial pressure. A lot is said about helping underprivileged and underrepresented groups in higher education, but few are *doing* more for such groups than these organizations; they are heroes in my book, and I hope I can pay their generosity forward soon.

# Abstract

Machine learning has made incredible advances in the last couple of decades. Notwithstanding, a lot of this progress has been limited to basic point-estimation tasks. That is, a large bulk of attention has been geared at solving problems that take in a *static finite vector* and map it to another *static finite vector*. However, we do not navigate through life in a series of point-estimation problems, mapping  $x$  to  $y$ . Instead, we find broad patterns and gather a far-sighted understanding of data by considering collections of points like sets, sequences, and distributions. Thus, contrary to what various billionaires, celebrity theoretical physicists, and sci-fi classics would lead you to believe, true machine intelligence is fairly out of reach currently. In order to bridge this gap, this thesis develops algorithms that understand data at an aggregate, holistic level.

This thesis pushes machine learning past the realm of operating over static finite vectors, to start reasoning ubiquitously with complex, dynamic collections like sets and sequences. We develop algorithms that consider distributions as functional covariates/responses, and methods that use distributions as internal representations. We consider distributions since they are a straightforward characterization of many natural phenomena and provide a richer description than simple point data by detailing information at an aggregate level. Our approach may be seen as addressing two sides of the same coin: on one side, we use traditional machine learning algorithms adjusted to directly operate on inputs and outputs that are probability functions (and sample sets); on the other side, we develop better estimators for traditional tasks by making use of and adjusting internal distributions.

We begin by developing algorithms for traditional machine learning tasks for the cases when one's input (and/or possibly output) is not a finite point, but is instead a distribution, or sample set drawn from a distribution. We develop a scalable nonparametric estimator for regressing a real valued response given an input that is a distribution, a case which we coin distribution to real regression (DRR). Furthermore, we extend this work to the case when both the output response and the input covariate are distributions; a task we call distribution to distribution regression (DDR).

After, we look to expand the versatility and efficacy of traditional machine learning tasks through novel methods that operate with distributions of features. For example, we show that one may improve the performance of kernel learning tasks by learning a kernel's spectral distribution in a data-driven fashion using Bayesian nonparametric techniques. Moreover, we study how to perform sequential modeling by looking at summary statistics from past points. Lastly, we also develop methods for high-dimensional density estimation that make use of flexible transformations of variables and autoregressive conditionals.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	Orthogonal Basis Functions and Projections . . . . .	5
2.2	Kernel Density Estimation . . . . .	10
2.3	Maximum Mean Discrepancy and Mean Map Embeddings . . . . .	12
2.4	Random Fourier Features . . . . .	13
2.5	Connections . . . . .	14
2.6	Related Work . . . . .	15
<b>I</b>	<b>Learning on Distributional Data</b>	<b>19</b>
<b>3</b>	<b>Scaling Up Distributional Learning: The Double and Triple Basis Estimators</b>	<b>21</b>
3.1	Distribution to Real Regression . . . . .	21
3.1.1	Kernel Smoother Approach . . . . .	22
3.2	Double Basis Estimator . . . . .	25
3.2.1	Orthonormal Basis . . . . .	25
3.2.2	Random Basis . . . . .	26
3.2.3	Estimator . . . . .	27
3.2.4	Evaluation Computational Complexity . . . . .	28
3.2.5	Ridge Double Basis Estimator . . . . .	29
3.3	Distribution to Distribution Regression . . . . .	29
3.3.1	Kernel Smoother Approach . . . . .	30
3.4	Triple Basis Estimator . . . . .	30
3.5	Experiments . . . . .	33
3.5.1	Synthetic Mapping . . . . .	34
3.5.2	Choosing $k$ : model selection for Gaussian mixtures . . . . .	34
3.5.3	Low Sample Dirichlet Parameter Estimation . . . . .	36
3.5.4	Rectifying 2LPT Simulations . . . . .	37

<b>4</b>	<b>Beyond the Euclidean Metric: Random Features for Homogenous Density Distances</b>	<b>39</b>
4.1	Embedding Information Theoretic Kernels . . . . .	39
4.1.1	Homogeneous Density Distances (HDDs) . . . . .	40
4.1.2	Embedding HDDs into $L_2$ . . . . .	40
4.1.3	Finite Embeddings of $L_2$ . . . . .	41
4.1.4	Embedding RBF Kernels into $\mathbb{R}^D$ . . . . .	42
4.1.5	Finite Sample Estimates . . . . .	42
4.1.6	Summary and Complexity . . . . .	43
4.2	Experiments . . . . .	43
<b>5</b>	<b>Conclusion</b>	<b>47</b>
5.1	Discussion . . . . .	47
<b>II</b>	<b>Learning with Distributions</b>	<b>49</b>
<b>6</b>	<b>Bayesian Spectral Distributions</b>	<b>51</b>
6.1	Introduction . . . . .	51
6.2	Model . . . . .	53
6.2.1	Random Features for Kernel Estimation . . . . .	53
6.2.2	Graphical Model . . . . .	55
6.3	Inference . . . . .	56
6.3.1	Sampling $Z_j$ . . . . .	56
6.3.2	Sampling $\mu_k$ and $\Sigma_k$ . . . . .	57
6.3.3	Sampling $W$ . . . . .	57
6.3.4	Runtime Complexity . . . . .	59
6.4	Experiments . . . . .	59
6.4.1	Synthetic Data . . . . .	59
6.4.2	Regression . . . . .	61
6.4.3	Classification . . . . .	62
6.4.4	Timing Experiments . . . . .	62
6.5	Conclusion . . . . .	64
<b>7</b>	<b>Distributions of Past Data for Sequence Modeling</b>	<b>65</b>
7.1	Introduction . . . . .	65
7.2	Model . . . . .	66
7.2.1	Recurrent Statistics . . . . .	66
7.2.2	Update Equations . . . . .	67
7.2.3	Intuitions from Mean Map Embeddings . . . . .	69
7.2.4	Viewpoints of the Past . . . . .	70
7.2.5	Vanishing Gradients . . . . .	71
7.3	Experiments . . . . .	72
7.3.1	Synthetic Recurrent Unit Generated Data . . . . .	73

7.3.2	MNIST Image Classification . . . . .	74
7.3.3	Polyphonic Music Modeling . . . . .	77
7.3.4	Electronica-Genre Music MFCC . . . . .	77
7.3.5	Climate Data . . . . .	77
7.3.6	SportVu NBA Tracking data . . . . .	78
7.4	Discussion . . . . .	79
<b>8</b>	<b>Transformation Autoregressive Networks</b>	<b>81</b>
8.1	Introduction . . . . .	81
8.2	Model . . . . .	83
8.2.1	Autoregressive Models . . . . .	83
8.2.2	Transformations . . . . .	85
8.2.3	Combined Approach, TANs . . . . .	87
8.3	Related Work . . . . .	88
8.4	Experiments . . . . .	90
8.4.1	Synthetic . . . . .	91
8.4.2	Efficacy on Real World Data . . . . .	92
8.4.3	Ablation Study . . . . .	93
8.4.4	Anomaly Detection . . . . .	96
8.4.5	Learning Parametric Family of Distributions . . . . .	96
8.5	Discussion . . . . .	97
8.6	Conclusion . . . . .	98
<b>9</b>	<b>Discussion</b>	<b>105</b>
9.1	Conclusion . . . . .	105
9.2	Future Directions . . . . .	107





# Chapter 1

## Introduction

The fast-paced growth of machine learning (ML) has been well documented; still, much of the attention so far in ML has been devoted to performing *point estimation tasks*. That is, much of the focus in ML has been geared towards methods that take in a single static finite vector, and map it to another single static finite vector. For example, point estimation includes mapping features of a person such as their age, income, credit-score, etc. to the probability that they are a default risk for a loan (Figure 1.1).

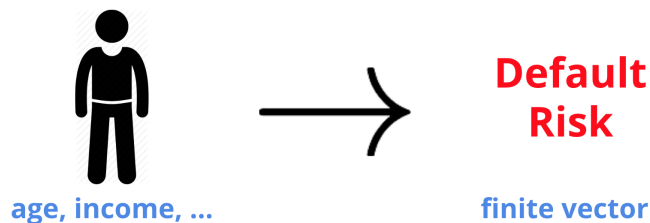


Figure 1.1: An example point estimation task where we are mapping the finite vector of features for a person to the chance that he/she is a default risk on a loan.

While point estimation tasks like these encompass many interesting problems, we do not live our lives simply going through a series of point estimation tasks mapping  $x$  to  $y$ . Instead, we often gather broad patterns from data by considering collections like *sets* and *distributions*. For instance, if I am contemplating whether to open a new restaurant in my neighborhood, I am not going to consider a single person and their features, but rather I will consider the *distribution* of people in my neighborhood. That is, the assessment of how fit a neighborhood is for a new business will come as a result of an analysis of an entire population rather than any single point (Figure 1.2).

Similarly, one may be interested in analyzing basketball players by the shots they take (Figure 1.3). For example, perhaps regressing the efficiency or plus-minus<sup>1</sup> of a player,

<sup>1</sup>The average difference in score between a player's team and the opponent whilst the player is on the court.

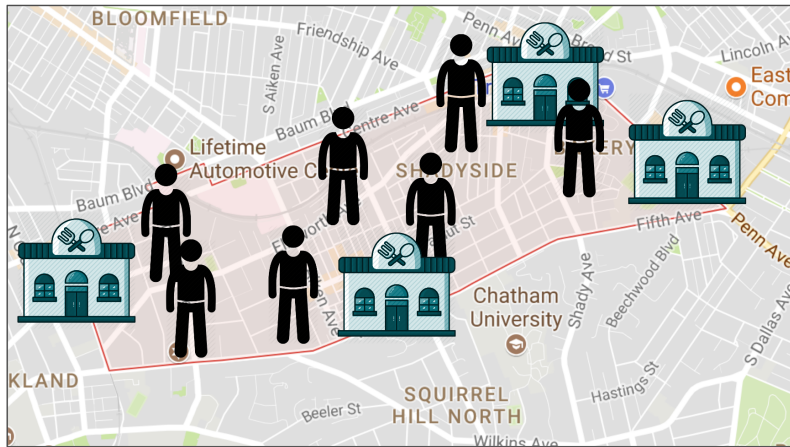


Figure 1.2: If one were trying to assess how profitable a new restaurant will be in a certain neighborhood, a holistic aggregate view of the distribution of people in the neighborhood will be much more effective than looking at any one individual. Similarly, one would also want to inspect the distribution of other restaurants in the neighborhood.

given the set of  $x, y$  coordinates for the shots taken by player. Once again, any single shot coordinate is insufficient to make an assessment on the player; rather, an aggregate view of the *distribution* of shots will be much more telling. Furthermore, one may also view the

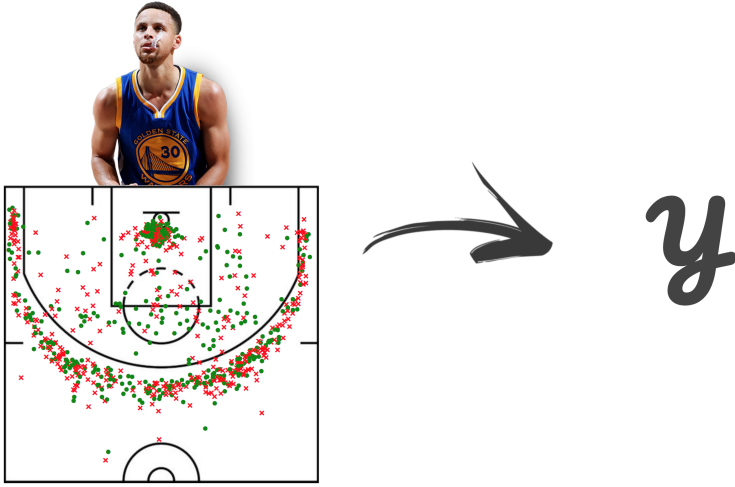


Figure 1.3: The distribution of the shots taken by a basketball player will be very indicative of the offensive style of the player. Thus, one may look to regress a value of efficiency ( $Y$ , such as plus-minus) based on the sample set of coordinates of the shots taken. As before, the analysis of any one particular shot coordinate will be ineffective.

task of classifying a  $3d$  point cloud of points describing a shape (Figure 1.4) as an instance of a problem where labeling any single point in a population is insufficient and, in turn, an aggregate, holistic view of the set of points is needed.

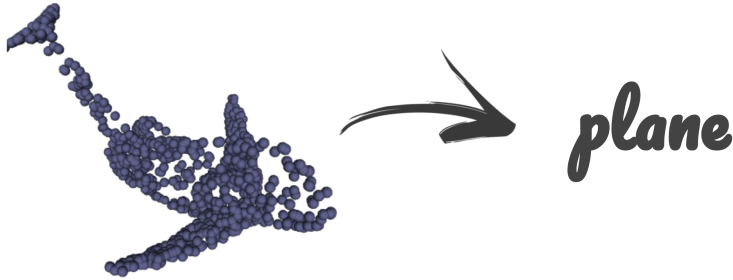


Figure 1.4: Classifying a set of  $3d$  points of an object’s point cloud. Here we do not wish to put a label on each individual point, but rather on the entire sample set itself.

In each of these problems, one has an input that is an entire population, a sample set  $\mathcal{X} = \{x_i \in \mathbb{R}^d\}_{i=1}^n$ , which we are trying to label. For instance, in the business example we are taking in the set of features  $x_i$  (such as, age, education level, income, etc.) of each person in a neighborhood, and mapping this set to a real-value of how much profit a new restaurant will make in such a neighborhood (Figure 1.5a). One’s first instinct may be to simply concatenate the points in the set  $\mathcal{X}$  into a long feature vector  $(x_1, \dots, x_n) \in \mathbb{R}^{nd}$  (Figure 1.5b). However, this approach begs two very important questions. First, since the input is a set without any intrinsic ordering one needs to decide the ordering of points in the concatenation. Indeed, any ordering of the points  $(x_{\pi_1}, \dots, x_{\pi_n})$  is as valid as any other ordering. Furthermore, one may look to regress a new neighborhood  $\mathcal{Y} = \{y_i \in \mathbb{R}^d\}_{i=1}^m$ , which may have a different cardinality  $m$  (Figure 1.5c). This would result in a different length feature vector  $(y_1, \dots, y_m) \in \mathbb{R}^{md}$ , and thus we would have to somehow adjust our learned model to operate over a different length.

In order to mitigate these complications previous approaches have considered the use of summary statistics or parametric distributions to represent sets [58, 53, 51]. For instance, one may assume that sets are draws from Gaussian distributions, and fit the means  $\mu$ , and covariances  $\Sigma$  to each set. One would then use  $(\mu, \Sigma)$  as the featurization of the input set  $\mathcal{X}$  (Figure 1.5d). However, such an approach would prove rather inflexible since one would be making strong assumptions on the distributions generating input sets. Furthermore, even if one such assumption were valid, it would require a high-level of domain knowledge to a-priori determine what assumption to make.

Instead, this thesis looks to develop machine learning methods that can reason over collective objects such as distributions using methods that are both *flexible* and *scalable*. We look to develop methods that are data-driven and flexible not just in the types of distributions that they can consider (e.g. beyond Gaussian), but are also flexible in terms of the relations they can capture between inputs and outputs (e.g. beyond linear). Moreover, we look to ensure that these methods are also *scalable*; allowing us to make fast predictions on new inputs and train on large datasets.

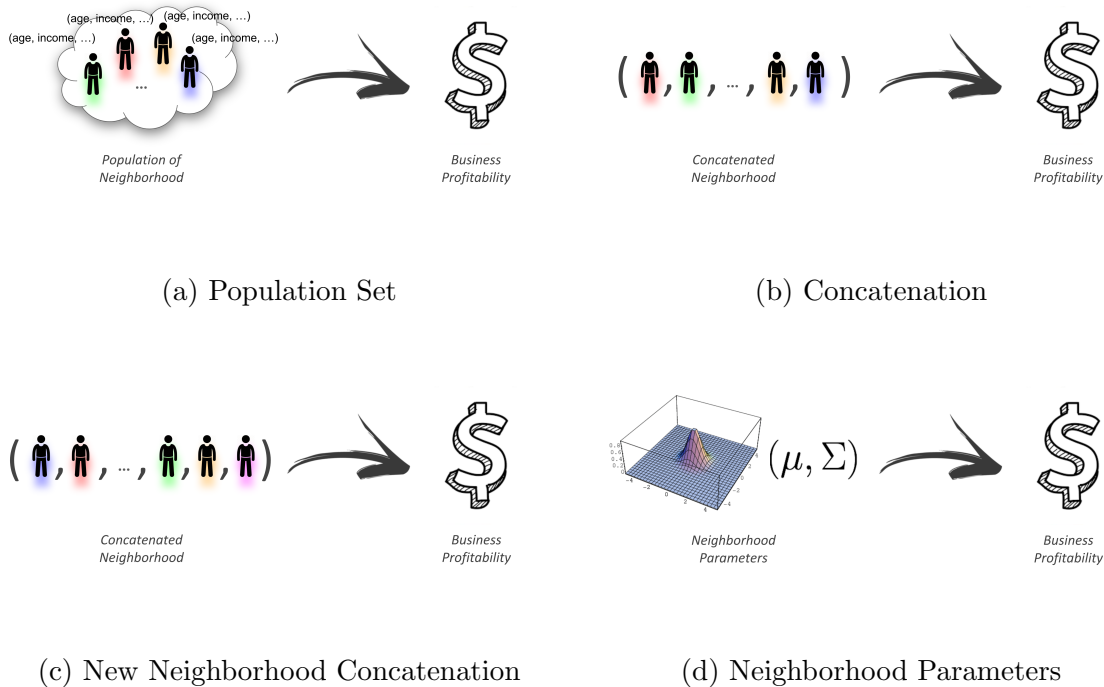


Figure 1.5: Simple approaches to dealing with an input sample set. (a) Task of mapping a set of features for people living in a neighborhood to how profitable a new restaurant would be in that neighborhood. (b) One simple approach would be to concatenate the features of people in the set into a long feature vector. Of course, any permutation of people is valid, leading to a factorial number of valid representations. (c) Furthermore, when attempting to label a new set of people the cardinality may be different, leading to a different length feature vector. (d) A way to mitigate these problems would be to make some parametric assumption about the underlying distribution of points, such as a Gaussian assumption, and fitting the parameters on each set. However, this proves inflexible since it is possible for this strong assumption to not hold.

We expand the use of distributions in machine learning in several related directions. First, we consider performing classical machine learning tasks such as regression when inputs (and possibly outputs) are not static finite vectors, but are instead distributions or sample sets drawn from distributions. For instance, we explore performing regression when one’s input covariate is a distribution and the output response is a real-value–Distribution to Real Regression (DRR). Furthermore, we also consider the case where the output response is a distribution–Distribution to Distribution Regression (DDR). Secondly, we enhance the performance of classical machine learning tasks by studying distributions of features. For example, we perform kernel learning for traditional ML tasks by learning an implicit distribution of kernel spectral frequencies. Moreover, we consider performing sequential modeling by looking at summary statistics from past points. Lastly, we also develop methods for high-dimensional density estimation.

# Chapter 2

## Background

Below we provide some background on common methods and literature that will be of use when operating with collections and distributions.

### 2.1 Orthogonal Basis Functions and Projections

First we expound on orthogonal basis functions for the estimation and representation of functions in a nonparametric fashion. We shall see that one can effectively estimate smooth functions using only a finite number of *projection coefficients* in our basis. This affords us a way of approximately representing general functions, which are inherently infinitely dimensional, with a finite vector of projection coefficients.

We will mainly focus on the space of square integrable functions  $L_2(\Omega)$ :

$$L_2(\Omega) = \left\{ f : \Omega \mapsto \mathbb{R} \mid \int_{\Omega} f^2 < \infty \right\}, \quad (2.1)$$

equipped with the standard inner-product on functions:

$$\forall f, g \in L_2(\Omega) \quad \langle f, g \rangle = \int_{\Omega} f(x)g(x)dx. \quad (2.2)$$

Let  $\{\varphi_i\}_{i \in \mathbb{Z}}$  be an orthonormal basis for  $L_2([0, 1])$ . That is,

$$\forall i, j \in \mathbb{Z} \quad \langle \varphi_i, \varphi_j \rangle = 0, \quad \text{span}(\{\varphi_i\}_{i \in \mathbb{Z}}) = L_2([0, 1]). \quad (2.3)$$

The tensor product of  $\{\varphi_i\}_{i \in \mathbb{Z}}$  serves as an orthonormal basis for  $L_2([0, 1]^d)$ ; that is, the following is an orthonormal basis for  $L_2([0, 1]^d)$ :

$$\{\varphi_{\alpha}\}_{\alpha \in \mathbb{Z}^d} \quad \text{where} \quad \varphi_{\alpha}(x) = \prod_{i=1}^d \varphi_{\alpha_i}(x_i), \quad x \in [0, 1]^d. \quad (2.4)$$

Note the abuse of notation on univariate and vector-valued indices  $\varphi_i$  and  $\varphi_{\alpha}$ . We have that  $\forall \alpha, \rho \in \mathbb{Z}^d$ ,  $\langle \varphi_{\alpha}, \varphi_{\rho} \rangle = \mathbb{I}\{\alpha = \rho\}$ .

Much like a vector basis, we may represent functions by projecting onto each basis element. Let  $p \in L_2([0, 1]^d)$ , then

$$p(x) = \sum_{\alpha \in \mathbb{Z}^d} a_\alpha(p) \varphi_\alpha(x), \quad \text{where} \quad a_\alpha(p) = \langle \varphi_\alpha, p \rangle = \int_{[0,1]^d} \varphi_\alpha(z) p(z) dz \in \mathbb{R}. \quad (2.5)$$

Note that unlike with a vector basis, here we are employing an infinite number of basis elements, and our projections,  $\langle \varphi_\alpha, p \rangle$ , are a functional inner-product (Figure 2.1).

Figure 2.1: We may represent a function (blue) as the functional projections onto each basis functions (red).

If the function we are representing,  $p$ , is a pdf, then we may estimate it through a sample  $\mathcal{X} = \{x_1, \dots, x_n\}$  where  $x_j \stackrel{iid}{\sim} p$ . This follows since:

$$a_\alpha(p) = \langle \varphi_\alpha, p \rangle = \int_{[0,1]^d} \varphi_\alpha(z) p(z) dz = \mathbb{E}_{X \sim p}[\varphi_\alpha(X)]. \quad (2.6)$$

That is, the projection of the pdf  $p$  onto the basis function  $\varphi_\alpha$  will correspond to an expectation (2.6) (Figure 2.2a), which may then be estimated through the empirical mean (Figure 2.2b):

$$a_\alpha(\mathcal{X}) = \frac{1}{n} \sum_{j=1}^n \varphi_\alpha(x_j) \approx \mathbb{E}_{X \sim p}[\varphi_\alpha(X)] = a_\alpha(p). \quad (2.7)$$

Thus, our estimator for  $p$  will be:

$$\tilde{p}(x) = \sum_{\alpha \in M} a_\alpha(\mathcal{X}) \varphi_\alpha(x), \quad (2.8)$$

where  $M$  is a finite set of indices for basis functions.

For  $L_2([0, 1])$  we may use the cosine basis:

$$\varphi_0(x) = 1, \quad \forall k \geq 1, \quad \varphi_k(x) = \sqrt{2} \cos(k\pi x). \quad (2.9)$$

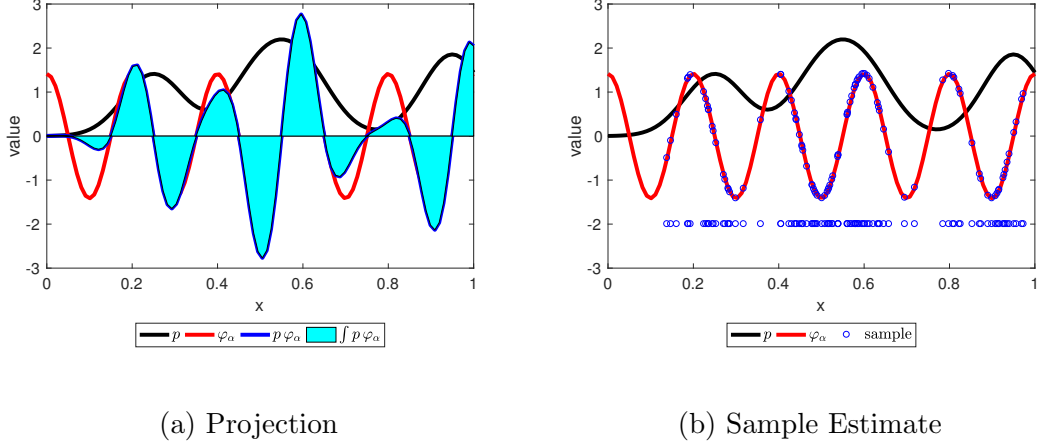


Figure 2.2: Since the projection of a pdf  $p$  onto a basis function  $\varphi_\alpha$ , (a), may be written as an expectation, we may estimate it using a sample drawn from  $p$  (b).

Thus, in  $L_2([0, 1]^2)$  our basis functions will be indexed by a  $2d$  vector  $\alpha = (\alpha_1, \alpha_2)$  and will be the outer product of cosines (Figure 2.3). We may restrict the set of basis functions to a set  $M_t = \{\alpha : \|\alpha\| \leq t\}$ , which will restrict the frequencies that we consider. Looking at the effect of different selections of  $t$  in Figure 2.4, we see that here  $t$  controls the smoothness of our estimates. We see that for a fixed choice of  $t$ , our estimate will improve as we observe more samples (Figure 2.5). For finite samples, the choice of  $t$  may lead to spurious modes and other artifacts, thus one would cross validate  $t$  as described below.

It is also worth noting that for functions

$$\tilde{p}(x) = \sum_{\alpha \in M} a_\alpha(\mathcal{X}) \varphi_\alpha(x) \quad \text{and} \quad \tilde{q}(x) = \sum_{\alpha \in M} a_\alpha(\mathcal{Y}) \varphi_\alpha(x),$$

we have that:

$$\langle \tilde{p}, \tilde{q} \rangle = \left\langle \sum_{\alpha \in M} a_\alpha(\mathcal{X}) \varphi_\alpha, \sum_{\alpha \in M} a_\alpha(\mathcal{Y}) \varphi_\alpha \right\rangle \quad (2.10)$$

$$= \sum_{\alpha \in M} \sum_{\beta \in M} a_\alpha(\mathcal{X}) a_\beta(\mathcal{Y}) \langle \varphi_\alpha, \varphi_\beta \rangle \quad (2.11)$$

$$= \sum_{\alpha \in M} a_\alpha(\mathcal{X}) a_\alpha(\mathcal{Y}) \quad (2.12)$$

$$= \langle \vec{a}_M(\mathcal{X}), \vec{a}_M(\mathcal{Y}) \rangle, \quad (2.13)$$

where  $\vec{a}_M(\cdot) = (a_{\alpha_1}(\cdot), \dots, a_{\alpha_s}(\cdot))$ ,  $M = \{\alpha_1, \dots, \alpha_s\}$ , and the last inner product is the vector dot product. Thus, we can use the Euclidean distance on vectors of projection coefficients to compute the functional  $L_2$  distance on functional estimates (Figure 2.6). That is, we can compare vectors of projection coefficients to compare respective functions.

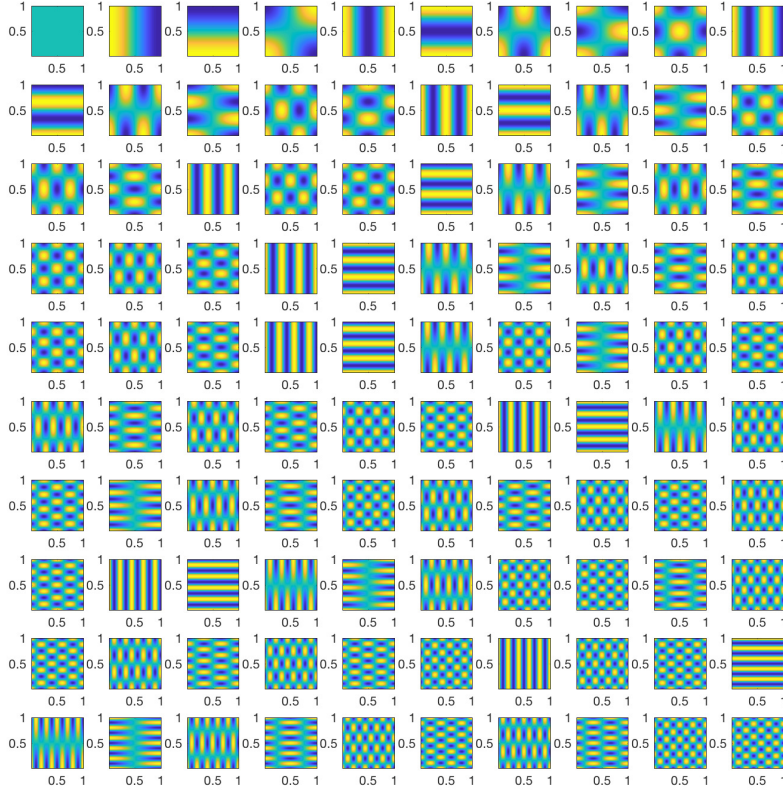


Figure 2.3: Cosine basis functions for  $L_2([0, 1]^2)$ , arranged in ascending index norm  $\|\alpha\|$  top-left to bottom-right.

### Cross-validation

In practice, one would choose indices  $M$  in (2.8) through cross-validation. The number of projection coefficients one chooses will depend on the smoothness of the function  $p$  as well as the number of points in  $\mathcal{X}$ . Typically, a larger  $|i|$  will correspond to a higher frequency 1-dimensional basis function  $\varphi_i$ ; thus, a natural way of selecting  $M$  is to consider sets

$$M_t = \{\alpha \in \mathbb{Z}^d : \|\alpha\|_2 \leq t\} \quad (2.14)$$

with  $t \in (0, \infty)$ . One would then choose the value of  $t$  (setting  $M = M_t$ ) that minimizes a loss between  $p$  and

$$\tilde{p}_t(x) = \sum_{\alpha \in M_t} a_\alpha(\mathcal{X}) \varphi_\alpha(x). \quad (2.15)$$

For instance, we may use a proxy to the squared loss:

$$\|\tilde{p}_t - p\|^2 \propto \|\tilde{p}_t\|^2 - 2\mathbb{E}_{X \sim p}[\tilde{p}_t(X)] = \sum_{\alpha \in M_t} a_\alpha^2(\mathcal{X}) - 2\mathbb{E}_{X \sim p}[\tilde{p}_t(X)] \equiv S_t. \quad (2.16)$$



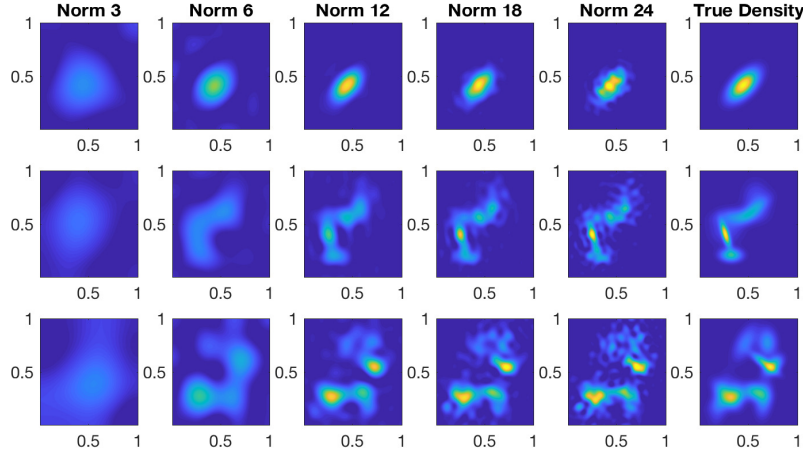


Figure 2.4: Different estimates given various norm maximums for indices of basis functions,  $\|\alpha\|_2 \leq t$  as captioned above, and a fixed sample size of 1000. True densities are show in right-most column.

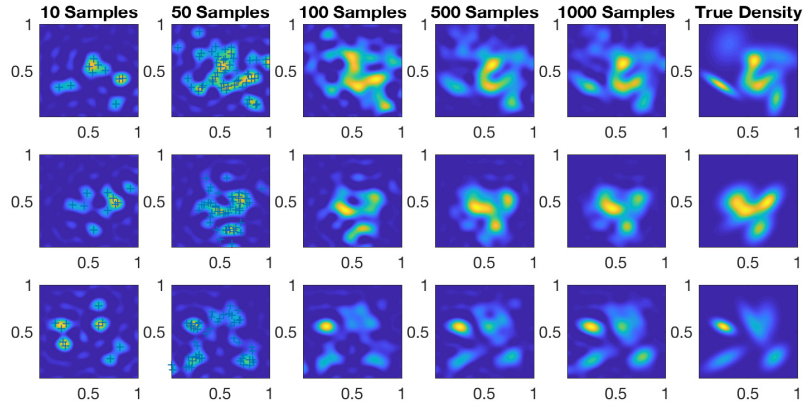


Figure 2.5: Density estimate with different sample sizes. Samples for small sizes (10, 50) are shown in crosses, true densities are show in right-most column.

$$\begin{array}{c}
 \text{[Heatmap of two overlapping curves]} \\
 = \|\tilde{p} - \tilde{q}\|_2 = \|\vec{a}_M(\mathcal{X}) - \vec{a}_M(\mathcal{Y})\|_2 = \text{[Vector diagram with two arrows]}
 \end{array}$$

Figure 2.6: The norm on the LHS is the functional  $L_2$  norm and corresponds to the vector  $\ell_2$  norm on the RHS.

Here, we may estimate the expectation  $\mathbb{E}_{X \sim p}[\tilde{p}_t(X)]$  by using a leave-one-out (LOO) pdf estimate,  $\tilde{p}_t^{-i}$ :

$$\tilde{p}_t^{-i}(x_i) = \sum_{\alpha \in M_t} \left( \frac{1}{n-1} \sum_{j \neq i} \varphi_\alpha(x_j) \right) \varphi_\alpha(x_i) \tag{2.17}$$

$$= \sum_{\alpha \in M_t} \frac{1}{n-1} (na_\alpha(\mathcal{Y}) - \varphi_\alpha(x_i)) \varphi_\alpha(x_i) \tag{2.18}$$

$$= \frac{n}{n-1} \sum_{\alpha \in M_t} a_\alpha(\mathcal{X}) \varphi_\alpha(x_i) - \frac{1}{n-1} \sum_{\alpha \in M_t} \varphi_\alpha^2(x_i). \tag{2.19}$$

We may plug the LOO pdf estimate (2.19) into our score (2.16):

$$\hat{S}_t = \sum_{\alpha \in M_t} a_\alpha^2(\mathcal{X}) - \frac{2}{n} \sum_{i=1}^n \tilde{p}_t^{-i}(x_i) \quad (2.20)$$

$$= \sum_{\alpha \in M_t} a_\alpha^2(\mathcal{X}) - \frac{2}{n-1} \sum_{i=1}^n \sum_{\alpha \in M_t} a_\alpha(\mathcal{X}) \varphi_\alpha(x_i) + \frac{2}{n(n-1)} \sum_{\alpha \in M_t} \varphi_\alpha^2(x_i) \quad (2.21)$$

$$= \sum_{\alpha \in M_t} a_\alpha^2(\mathcal{X}) - \frac{2n}{n-1} \sum_{\alpha \in M_t} a_\alpha(\mathcal{X}) \left( \frac{1}{n} \sum_{i=1}^n \varphi_\alpha(x_i) \right) + \frac{2}{n(n-1)} \sum_{i=1}^n \sum_{\alpha \in M_t} \varphi_\alpha^2(x_i) \quad (2.22)$$

$$= \sum_{\alpha \in M_t} a_\alpha^2(\mathcal{X}) - \frac{2n}{n-1} \sum_{\alpha \in M_t} a_\alpha^2(\mathcal{X}) + \frac{2}{n-1} \sum_{\alpha \in M_t} \left( \frac{1}{n} \sum_{i=1}^n \varphi_\alpha^2(x_i) \right) \quad (2.23)$$

$$= -\frac{n+1}{n-1} \sum_{\alpha \in M_t} a_\alpha^2(\mathcal{X}) + \frac{2}{n-1} \sum_{\alpha \in M_t} \left( \frac{1}{n} \sum_{i=1}^n \varphi_\alpha^2(x_i) \right), \quad (2.24)$$

which may easily be computed by taking the empirical means of  $\varphi_\alpha(\cdot)$  and  $\varphi_\alpha^2(\cdot)$  and selecting the threshold  $\|\alpha\| \leq t$  to minimize  $\hat{S}_t$  (2.24).

For more information on orthogonal bases and their application to functional estimation see [122].

## 2.2 Kernel Density Estimation

Next, we also consider the use kernel density estimation (KDE), which estimates the density  $p$  as a mixture of “bumps” centered at each data point.

We may begin motivating kernel density estimation as [112], with the observation that for the 1d density  $p : \mathbb{R} \mapsto \mathbb{R}^+$ :

$$p(x) = \lim_{h \rightarrow 0} \frac{1}{2h} \Pr[x - h < X < x + h] = \lim_{h \rightarrow 0} \frac{1}{2h} \mathbb{E}[\mathbb{I}\{X \in (x - h, x + h)\}]. \quad (2.25)$$

Given a sample  $\mathcal{X} = \{x_i \in \mathbb{R}\}_{i=1}^n \stackrel{iid}{\sim} p$  (2.25) may be estimated using small  $h$  as:

$$\hat{p}(x) = \frac{1}{2nh} \sum_{i=1}^n \mathbb{I}\{x_i \in (x - h, x + h)\} = \frac{1}{n} \sum_{i=1}^n w_h(x_i, x) \quad (2.26)$$

where  $w_h(x_i, x) = \frac{1}{2h} \mathbb{I}\{x_i \in (x - h, x + h)\}$ . In essence, (2.26) leads to a boxed estimate of the pdf with indicators centered at each sample point. This in turn will lead to a non-smooth estimate with multiple discontinuities, which will be at a disadvantage when estimating smooth distributions.

In order to produce a smooth estimate, we may replace the boxed indicator function  $w_h$  with a *smoothing kernel*  $k_h$ :

$$\tilde{p}_h(x) = \frac{1}{n} \sum_{i=1}^n k_h(x_i, x). \quad (2.27)$$

For instance, we may use a Gaussian centered at the data with a bandwidth  $h$ :

$$k_h(x_i, x) = \frac{1}{\sqrt{2\pi}h} \exp\left(-\frac{|x_i - x|^2}{2h^2}\right). \quad (2.28)$$

If the kernel  $k_h$  is a normalized pdf then  $\tilde{p}$  is clearly a distribution, composed of an equi-weighted mixture of the densities  $k_h(x_i, x)$ . A common choice of kernel in multiple dimensions is the product kernel:

$$k_h(x, y) = \prod_{j=1}^d k_h(x_j, y_j), \quad \forall x, y \in \mathbb{R}^d \quad (2.29)$$

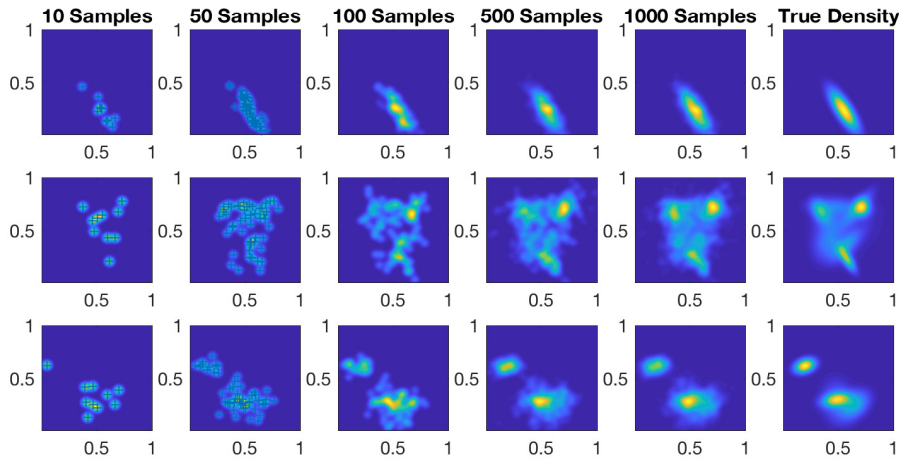


Figure 2.7: Kernel density estimates for various sample sizes using a fixed bandwidth. For small sample sizes (10, 50) points are displayed with crosses. Our estimates are composed of the average of ‘bumps’ placed on each sample point; a fact that is especially apparent with smaller samples. One may see that as sample sizes increase, our estimates approach the true distribution, shown in the right-most column.

## Cross-validation

The bandwidth parameter,  $h$  (2.27), is controlling the amount of smoothness in our estimate, and poor choices may lead to over or under-smoothing (Figure 2.8). In order to cross-validate the bandwidth, we may proceed similarly to the orthonormal series estimator (2.16):

$$\|\tilde{p}_h - p\|_2^2 \propto \|\tilde{p}_h\|_2^2 - 2\mathbb{E}_p[\tilde{p}_h(X)] \equiv S_h. \quad (2.30)$$

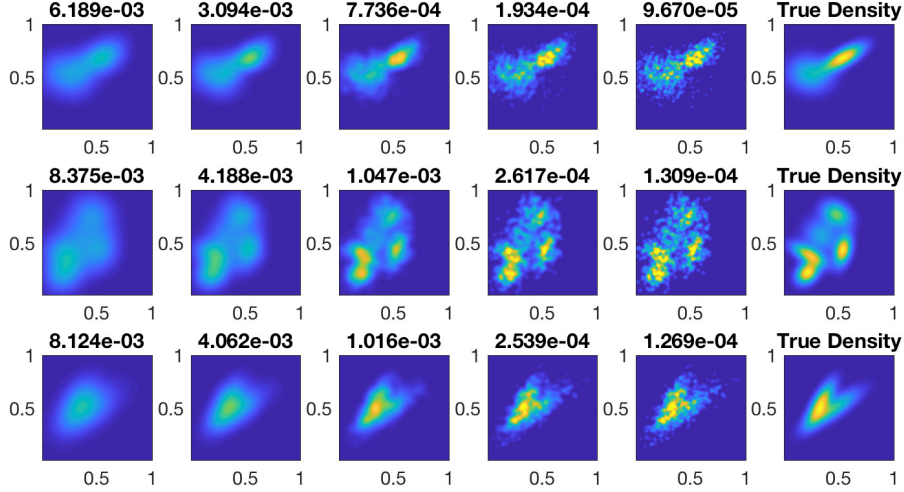


Figure 2.8: Kernel density estimates for a fixed (1000) sample size using various bandwidths (displayed above figure). One may see that for bigger bandwidths (left) one will have overly smoothed estimates; in contrast, smaller bandwidths (right) may led to under-smoothed estimates. True densities are shown in the right-most column.

As before, we may make use of a leave-on-out estimate  $\tilde{p}_h^{-1}(x) = \frac{1}{n-1} \sum_{j \neq i} k_h(x_j, x)$ :

$$\hat{S}_h \propto \|\tilde{p}_h\|_2^2 - \frac{2}{n} \sum_{i=1}^n \tilde{p}_h^{-i}(x_i) \quad (2.31)$$

$$= \frac{1}{n^2} \sum_{i=1}^n \|k_h(x_i, \cdot)\|_2^2 + \frac{2}{n^2} \sum_{i=1}^n \sum_{j=i+1}^n \langle k_h(x_i, \cdot), k_h(x_j, \cdot) \rangle - \frac{2}{n(n-1)} \sum_{i=1}^n \sum_{j \neq i} k_h(x_j, x_i) \quad (2.32)$$

where  $\|k_h(x_i, \cdot)\|_2^2$ ,  $\langle k_h(x_i, \cdot), k_h(x_j, \cdot) \rangle$  are the functional  $L_2$  norm and inner products:  $\|k_h(x_i, \cdot)\|_2^2 = \int_{\mathbb{R}^d} k_h^2(x_i, x) dx$ , and  $\langle k_h(x_i, \cdot), k_h(x_j, \cdot) \rangle = \int_{\mathbb{R}^d} k_h(x_i, x) k_h(x_j, x) dx$ , which depend on our choice of kernel  $k_h$ .

For a general discussion of KDEs please see [122].

## 2.3 Maximum Mean Discrepancy and Mean Map Embeddings

We also make use of the maximum mean discrepancy (MMD) and mean map embedding (MME) to characterize distributions [44, 79, 120, 115, 116, 117]. The MMD is a metric on distributions that may be defined via the MME, an embedding of distributions in a RKHS. Suppose that  $k$  is a Mercer kernel inducing RKHS  $\mathcal{H}$ . The MME embeds a distribution  $p$

in  $\mathcal{H}$  as  $\mu[p]$ :

$$\mu[p](x) \equiv \mathbb{E}_{X \sim p}[k(X, x)]. \quad (2.33)$$

This embedding  $\mu[p]$  is unique if one uses a characteristic kernel such as the RBF. The MME then yields a kernel (inner product) and metric on distributions. This metric is exactly the MMD:

$$\text{MMD}(p, q) \equiv \|\mu[p] - \mu[q]\|_{\mathcal{H}} = \sup_{\|g\|_{\mathcal{H}} \leq 1} \mathbb{E}_{X \sim p}[g(X)] - \mathbb{E}_{Y \sim q}[g(Y)], \quad (2.34)$$

where the last equation is an alternative formulation of the MMD.

We may make an estimate of the MME of  $p$  using a sample  $\mathcal{X} = \{x_i\}_{i=1}^n \stackrel{iid}{\sim} p$ :

$$\mu[p](\cdot) = \mathbb{E}_{X \sim p}[k(X, \cdot)] \approx \frac{1}{n} \sum_{i=1}^n k(x_i, \cdot) \equiv \mu[\mathcal{X}](\cdot). \quad (2.35)$$

This may then be interpreted as inducing a kernel on sets  $\mathcal{X} = \{x_i\}_{i=1}^n$ , and  $\mathcal{Y} = \{y_i\}_{i=1}^m$ :

$$\langle \mu[\mathcal{X}], \mu[\mathcal{Y}] \rangle_{\mathcal{H}} = \left\langle \frac{1}{n} \sum_{i=1}^n k(x_i, \cdot), \frac{1}{m} \sum_{j=1}^m k(y_j, \cdot) \right\rangle_{\mathcal{H}} \quad (2.36)$$

$$= \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m \langle k(x_i, \cdot), k(y_j, \cdot) \rangle_{\mathcal{H}} \quad (2.37)$$

$$= \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m k(x_i, y_j), \quad (2.38)$$

which amounts to an average pair-wise similarity as induced by  $k$ .

## 2.4 Random Fourier Features

Next, we discuss random features to better scale the use of kernels to large datasets. Kernel methods often scale as  $O(N)$  for evaluations where  $N$  is the number of instances. This is prohibitive in big datasets where  $N$  is large. With the use of random features, one will be able to approximate a linear smoother efficiently over large datasets.

Rahimi and Recht [102] show that if one has a shift-invariant kernel  $K$  (in particular we consider the RBF kernel  $K(x) = \exp(-x^2/2)$ ) then for  $x, y \in \mathbb{R}^d$ :

$$K(\|x - y\|_2 / \sigma) \approx z(x)^T z(y), \quad \text{where } z(x) \equiv \sqrt{\frac{2}{D}} [\cos(\omega_1^T x + b_1) \cdots \cos(\omega_D^T x + b_D)]^T \quad (2.39)$$

with  $\omega_i \stackrel{iid}{\sim} \mathcal{N}(0, \sigma^{-2} I_d)$ ,  $b_i \stackrel{iid}{\sim} \text{Unif}[0, 2\pi]$ . The quality of the approximation in (2.39) depends on a variety of factors including the number of random features  $D$ ; see [102] for details.

One may approximate a linear smoother on  $N$  instances as a linear mapping on the  $D$  random features; let  $K_\sigma(t) = K(t/\sigma)$ , then for  $\{x_i \in \mathbb{R}^d\}_{i=0}^N$ :

$$\sum_{i=1}^N \theta_i K_\sigma(\|x_i - x_0\|) \approx \sum_{i=1}^N \theta_i z(x_i)^T z(x_0) = \left( \sum_{i=1}^N \theta_i z(x_i) \right)^T z(x_0) = \psi^T z(x_0) \quad (2.40)$$

where  $\psi = \sum_{i=1}^N \theta_i z(x_i) \in \mathbb{R}^D$ . This approximation will be key to scaling up estimators to large datasets.

Note that we may make an estimate of the MME of  $p$  using  $\mathcal{X} = \{x_i\}_{i=1}^n \stackrel{iid}{\sim} p$  and random features (2.39):

$$\mu[p](x) = \mathbb{E}_{X \sim p}[K(X, x)] \approx \frac{1}{n} \sum_{i=1}^n K(x_i, x) \approx \frac{1}{n} \sum_{i=1}^n \langle z(x_i), z(x) \rangle. \quad (2.41)$$

Thus, we may represent  $p$  as the finite vector of mean random features,  $\mu_z[\mathcal{X}] = \frac{1}{n} \sum_{i=1}^n z(x_i) \in \mathbb{R}^D$  in the approximate primal space induced by the random features  $x \mapsto \langle z(x), z(\cdot) \rangle$ .

## 2.5 Connections

There exist numerous connections among orthonormal series, kernel densities, and mean map embeddings. Indeed, each may be interpreted as a version or approximation of the other.

First we begin by discussing connections to orthonormal series estimates. Recall that for a sample  $\mathcal{X} = \{x_i\}_{i=1}^n \stackrel{iid}{\sim} p$  we can write our OSE for  $p$  as:

$$\tilde{p}(x) = \langle \vec{a}_M(\mathcal{X}), \vec{a}_M(x) \rangle \quad (2.42)$$

$$= \frac{1}{n} \sum_{i=1}^n \langle \vec{a}_M(x_i), \vec{a}_M(x) \rangle \quad (2.43)$$

$$= \frac{1}{n} \sum_{i=1}^n k_M(x_i, x), \quad (2.44)$$

where  $k_M(x_i, x) = \langle \vec{a}_M(x_i), \vec{a}_M(x) \rangle$  may be interpreted as a kernel. Clearly then OSE may be interpreted as both KDE (2.27) and MME (2.33) with a very specific choice of kernel. It is also interesting to note that the MME of a set may be written as follows with random features

$$\mu_z[\mathcal{X}](x) = \frac{1}{n} \sum_{i=1}^n \langle z(x_i), z(x) \rangle = \left\langle \frac{1}{n} \sum_{i=1}^n z(x_i), z(x) \right\rangle. \quad (2.45)$$

Thus, one may view the MME  $\mu_z[\mathcal{X}](x)$  as the projection coefficients onto the (random) basis  $\{z_{\omega_k}(x)\}_{k=1}^D$ .

There are clear connections between KDEs and MMEs. Note that the expected value of the KDE is under a shift invariant kernel  $k(x, y) = K(x - y)$ :

$$\mathbb{E}_p [\tilde{p}(y)] = \int p(x)k(x, y)dx = \mu_k[p](x). \quad (2.46)$$

That is, the expected value of the KDE is the MME under the corresponding kernel. Of course, the MME need not be tied to a density estimate and may use, as is often the case, unnormalized, arbitrary inner-products as kernels.

## 2.6 Related Work

**Learning on distributions** There are some traditional approaches in machine learning that have considered some form of distributional covariates. For instance, in computer vision, the popular “bag of words” model [67] represents a distribution by quantizing it onto codewords (usually by  $k$ -means on points from all sets), then compares those histograms with some kernel (often exponentiated  $\chi^2$ ).

Early work on general learning over distributions include fitting parametric models on input distributions and estimating a distance between distributions, often the  $L_2$  distance or Kullback-Leibler (KL) divergence [50, 78, 54]. Some efforts to estimate these distances nonparametrically include [118, 60]. The distances can then be used in kernel smoothing [100, 93] or Mercer kernels [78, 58, 54, 101, 81].

Another approach is to represent a distribution by its mean RKHS embedding under some kernel  $k$ . The RKHS inner product is known as the *mean map kernel* (MMK), and the distance the *maximum mean discrepancy* (MMD) [44, 79, 120, 115]. When  $k$  is the RBF kernel, the MMK estimate is proportional to an  $L_2$  inner product between Gaussian kernel density estimates [112].

These approaches can be powerful, but usually require computing an  $N \times N$  matrix of kernel evaluations, which can be infeasible for large datasets. One way to alleviate the scaling problem is the Nyström extension [136], in which some columns of the Gram matrix are used to estimate the remainder. In practice, one frequently must compute many columns, and methods to make the result PSD are known only for mildly-indefinite kernels [7]. Instead, we shall make extensive use of approximate kernel embeddings [102] to scale kernel methods. That is, we shall approximate shift-invariant kernels by sampling their Fourier transform, allowing us to work linearly in an approximate primal space of random features. In Chapters 3 and 4 we make use of bases, including random bases with Fourier features, to learn over distributions. Furthermore, in Chapter 6 we explore learning the spectral distribution of ones kernel with Bayesian priors.

**Multi-Instance Learning** Multiple instance learning (MIL) [147, 16] also attempts to classify a set or “bag” of points. However, typically in MIL a bag is considered positive if at least one of its point is positive; otherwise it is negative. Hence, the MIL task is inherently determined by individual points, rather than any holistic assessment on the entirety of the bag or the distribution of its points. Nevertheless, there has been prior

work approaching MIL with set kernels [13, 33]. It is worth noting that the MIL task of scanning an iid bag for positive points can be considered a special case of a distribution learning task where a distribution is labeled as positive depending on whether it contains modes that concentrate on positive data points. That is, suppose that  $f : \mathbb{R}^d \mapsto \{0, 1\}$  is the classification decision function on points; then, the inner-product of  $f$  and the density generating a bag,  $p : \mathbb{R}^d \mapsto \mathbb{R}^+$ ,  $\langle f, p \rangle > 0$  will indicate whether the bag is positive or not.

**Learning over Graphs** Another related field is learning on graphs [103], where inputs are graphs over a collection of nodes. That is, learning over graphs attempts to apply either a categorical label (for classification) or real label (for regression) over input instances that are graphs (represented either through an adjacency matrix or edge list). Unlike with learning over distributions, learning over graphs considers a structured, non-iid, collection of nodes (as opposed to an iid sample, for instance). One common approach is to consider a kernel over graphs. Often, kernels on graphs will count common substructures in a pair of graphs. For example, the random walk kernel [132, 34] measures the path similarity of random walks in different graphs. Recent work [59, 21] has also considered embedding graphs into a feature space that has invariances to node permutations.

**Function (to Function) Regression** Function regression when inputs and possibly outputs are general functions has also been explored. A previous nonparametric function to function regression (FFR) estimator was proposed in [55]. [55] attempts to perform FFR on a functional RKHS. That is, if we consider  $\mathcal{F}$  as a functional Hilbert space, where  $f \in \mathcal{F}$  is such that  $f : \mathcal{G}_x \mapsto \mathcal{G}_y$ , then  $f$  is estimated by  $f^* = \arg \min_{\hat{f}} \sum_{i=1}^N \|q_i - \hat{f}(p_i)\|_{\mathcal{G}_y}^2 + \lambda \|\hat{f}\|_{\mathcal{F}}^2$ . However, when each function is observed through  $n$  noisy function evaluations this estimator will require the inversion of a  $Nn \times Nn$  matrix, which will be computationally infeasible for data-sets of even a modest size. We note further that work has been done in linear models for function to real regression (FRR) (e.g. [104, 95]). However, such models work over a strong assumption on the linearity of the mapping  $f$ , and will not be able to capture non-linear mappings. Moreover, FRR and FFR are specific cases of general functional analysis [104, 30, 105].

**Sequence Modeling** The analysis of sequential data has long been a staple in machine learning. Domain areas like natural language [142, 131], speech [41, 40], music [20], and video [25] processing have recently garnered much attention. Indeed, sequences themselves are complex objects that one may learn over. Furthermore, while the study of sequences itself is broad and may be extended to general functional analysis [105], most recent success has been from neural network based models, especially from recurrent architectures.

Recurrent networks are dynamical systems that represent time recursively. For example, the simple recurrent unit [27] contains a hidden state that itself depends on the previous hidden state. However, training such networks has been observed to be difficult in practice due to exploding and vanishing gradients when propagating error gradients through time [46]. While exploding gradients can be mitigated with techniques like gradient clipping and normalization [99], vanishing gradients may be harder to deal with. As a result,



sophisticated gated architectures like Long-Short Term Memory (LSTM) networks [47] and Gated Recurrent Units (GRU) networks [19] have been developed. These gated architectures contain “memory cells” along with gates to control how much they decay through time thereby aiding the networks’ ability to learn long term dependencies in sequences.

In addition to the more complex gated architectures, there has been recent work focusing on alternative simpler, un-gated architectures. For example, employing convolutions [144, 35], or attention based methods [127] for sequence modeling. In a similar vein, we explore an architecture, the statistical recurrent unit (SRU), that only employs moving averages of statistics to model sequential data in Chapter 7.

**High-dimensional Modeling** Nonparametric approaches like kernel density estimation suffer greatly from the curse of dimensionality and do not perform well when data does not have a small number of dimensions ( $d \lesssim 3$ ). To alleviate this, several semiparametric approaches have been explored. Such approaches include forest density estimation [70], which assumes that the data has a forest (i.e. a collection of trees) structured graph. This assumption leads to a density which factorizes in a first order Markovian fashion through a tree traversal of the graph. Another common semiparametric approach is to use a nonparanormal type model [69]. This approach uses a Gaussian copula with a rank-based transformation and a sparse precision matrix. While both approaches are well-understood theoretically, their strong assumptions lead to inflexible models.

In order to provide greater flexibility with semiparametric models, recent work has employed deep learning for density estimation. The use of neural networks for density estimation dates back to early work by Bishop [12] and has seen success in areas like speech [143, 123], music [14], etc. Typically such approaches use a network to learn the parameters of a parametric model for data. Recent work has also explored the application of deep learning to build density estimates in image data [97, 23]. However, such approaches are heavily reliant on exploiting structure in neighboring pixels, often subsampling, reshaping or re-ordering data, and using convolutions to take advantage of neighboring correlations.

Modern approaches for general density estimation in real-valued data include a variety of diverse solutions exploiting different aspects of the problems. A large number of methods have considered auto-regressive models to estimate the conditional factors  $p(x_i|x_{i-1}, \dots, x_1)$ , for  $i \in \{1, \dots, d\}$  in the chain rule [63, 126, 124, 37, 43] (Figure 8.2b). While some methods directly model the conditionals  $p(x_i|x_{i-1}, \dots)$  using sophisticated semiparametric density estimates, other methods apply sophisticated transformations of variables  $x \mapsto z$  and take the conditionals over  $z$  to be a restricted, often independent base distribution  $p(z_i|z_{i-1}, \dots) \approx f(z_i)$  [22, 23](Figure 8.2a). Furthermore, [98] identified that single component Gaussian conditional autoregressive models for density estimation can be seen as deterministic shift and rescale transformations. In Chapter 8 we explore combining both flexible transformations and autoregressive conditionals to better model data.

Other methods have looked to model the data via partially or fully bypassing density estimation. For instance, variational auto-encoders (VAEs) [24], optimize the log-likelihood through a lower bound on the likelihood  $p(x)$  via encoding and estimated decoding distributions  $p(x|z)$  and  $q(z|x)$ , respectively. Although it is simple to generate a new sample

as  $Z \sim p(z)$ ,  $X \sim p(x|z)$ , it is difficult to marginalize the codes to obtain the likelihood for a data-point:  $p(x) = \int p(x|z)p(z)dz$ . Recent work on generative adversarial networks (GANs) [39] has looked to model data in a likelihood-free manner by considering an decoder (non-invertible transformation) of random codes coming from a simple set base distribution. The outputs of this decoder is then fed to a discriminator that judges if the output belongs to the set of actual samples.

Although GANs have produced exciting results (especially in the image domain), they also have been noted to be difficult to train, and suffer from mode collapse [38]. In addition, recent empirical studies have found that GANs are generating distributions whose support is only a small constant,  $c$ , larger than the original dataset size,  $c|\mathcal{D}|$  [4]. This directly negates the claim the GANs are learning to generate the underlying true distribution  $p$  over a domain  $\mathcal{I}$ . For example, a simple approach like applying  $c$  distinct transformations  $\{f_k : \mathcal{I} \mapsto \mathcal{I}\}_{k=1}^c$  to each instance in a dataset  $\mathcal{D}$  would generate the same support of novel instances while clearly not learning to generate the distribution. It is also worth noting that quantitatively assessing the quality of likelihood-free methods like GANs is very difficult. Although domain-specific heuristics exist, like the inception score for images [6], there is a lack of a general score to assess the fit of these models for generating the true distribution. In fact, achieving a score for generated data-points  $\{x'_i\}_{i=1}^m$ , based on a training dataset  $\{x_i\}_{i=1}^N$ ,  $s(\{x'_i\}_{i=1}^m | \{x_i\}_{i=1}^N)$  may be as difficult as the original modeling task, since one could use  $s$  and an MCMC sampling method to generate data.

# Part I

## Learning on Distributional Data

We begin by considering traditional machine learning tasks when one's input, and possibly output, is not a single finite vector, but is rather a distribution observed through a sample set. First, we develop scalable methods for distribution to real and distribution to distribution regression. In Chapter 3 we shall combine the use of random and orthonormal bases to perform DDR and DRR tasks on big datasets with many instances. Second, we shall extend these approaches to be able to consider non- $L_2$  metrics in kernels on distributions. In Chapter 4 we introduce random features to approximate kernels that use a broad range of metrics over distributions.



# Chapter 3

## Scaling Up Distributional Learning: The Double and Triple Basis Estimators

In the next chapters we consider how to scalably perform ML tasks when inputs, and possibly outputs, are distributions. Indeed, we do not live in a world of simple objects; from the contact lists we keep, to the stock prices we follow, and the distribution of cells we have, distributional data is all around us. Furthermore, with ever-increasing data collection capacities at our disposal, not only are we collecting more data, but richer and more bountiful complex data are becoming the norm.

This chapter aims to make learning on massive data-sets of distributions tractable. We study distribution to real regression (DRR) where input covariates are arbitrary distributions and output responses are real values. Here, we provide an estimator, the Double Basis Estimator (2BE) [91], that scales well with data-set size and is efficient at evaluation-time. Furthermore, we consider distribution to distribution regression (DDR) where both input covariate and output response are distributions. We shall see that we may scale up DDR by considering an extension to the 2BE, the Triple Basis Estimator (3BE) [92], which makes use of an additional orthonormal basis for representing the output pdf.

### 3.1 Distribution to Real Regression

First we focus on regression when input covariates are nonparametric distributions and output responses are real values, a task we coin distribution to real regression (DRR). We consider a mapping  $f : \mathcal{I} \mapsto \mathbb{R}$  that takes  $p \in \mathcal{I}$ , an input pdf from a broad family of densities  $\mathcal{I}$ , and produces  $Y \in \mathbb{R}$  a real-valued response as:

$$Y = f(p) + \epsilon, \quad \text{where } \mathbb{E}[\epsilon] = 0, \mathbb{E}[\epsilon^2] \leq \sigma_\epsilon^2. \quad (3.1)$$

Of course, it is infeasible to directly observe a pdf in practice. Thus, we will work instead on a data-set of  $N$  input sample-sets/responses:

$$\mathcal{D} = \{(\mathcal{X}_i, Y_i)\}_{i=1}^N, \quad \text{where } \mathcal{X}_i = \{x_{i1}, \dots, x_{in_i}\}, \quad x_{ij} \stackrel{iid}{\sim} p_i, \quad (3.2)$$

and  $Y_i = f(p_i) + \epsilon_i$ . Note that we may think of  $\mathcal{X}_i$  as a noisy observation of  $p_i$  (see Figure 3.1).

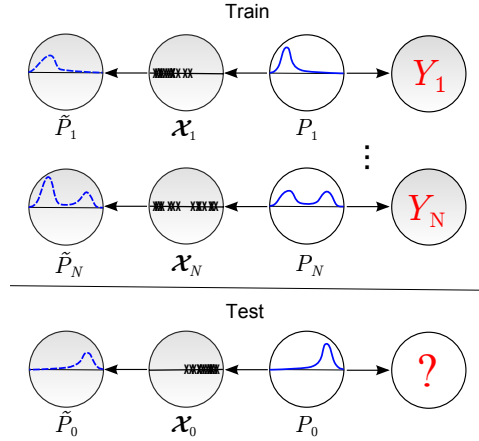


Figure 3.1: A graphical representation of the DRR model. We observe a data-set of input sample-set/output response pairs  $\{(\mathcal{X}_i, Y_i)\}_{i=1}^N$ , where  $\mathcal{X}_i = \{x_{i1}, \dots, x_{in_i}\}$ ,  $x_{ij} \sim P_i$  and  $Y_i = f(p_i) + \epsilon_i$ , for some noise  $\epsilon_i$ .

Many interesting problems across various domains fit the DRR framework. For example, one may be interested in studying the mapping that takes in the distribution of star locations in a galaxy and outputs the galaxy’s age. Also, one may consider a mapping that takes in the distribution of prices for stocks of a particular sector and outputs the future average change in stock price for that sector.

In fact, many estimation tasks in statistics can be framed as a distribution to real regression problem. For instance, in statistical parameter estimation one studies a mapping that takes in a distribution (usually restricted to be in a parametric class of distributions) and outputs a corresponding parameter. We will see that our estimators can be used to leverage previously seen sample sets to outperform standard estimation procedures, to perform model selection when cross validation is expensive, and to perform parameter estimation when no analytical sample estimate is available. In effect, our estimators, and the concept of distribution to real regression, is powerful enough to itself learn how to perform general statistical procedures.

At its core, the problem of distribution to real value regression is a learning task over infinite dimensional objects (distributions) and would benefit greatly from learning on data-sets with a large number of input/output pairs. Hence, this chapter focuses on the case where one has a massive data-set in terms of instances, i.e.  $n_i = o(N)$ .

### 3.1.1 Kernel Smoother Approach

DRR for the case of general input distributions in a Hölder class and a smooth class of mappings has been previously studied in [100]. There, a linear smoothing estimator—the Kernel-Kernel estimator—analogue to the Nadaraya-Watson kernel smoothing estimator for functional distribution inputs was considered [122]. For the more typical case with a dataset

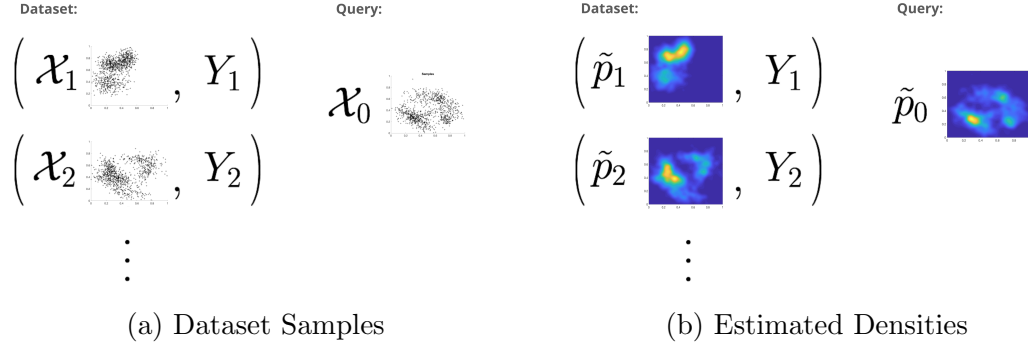


Figure 3.2: Graphical representation of the kernel linear smoother method. (a) Dataset of pairs of input sample sets and real valued responses  $(\mathcal{X}_i, Y_i)$ , and query input sample set  $\mathcal{X}_0$ . (b) On each input sample set a nonparametric density estimate is built,  $\tilde{p}_i$ . (c) The distance of the query density to each training density is used in a Nadaraya-Watson kernel smoother.

of vector valued input covariates and real responses,  $\mathcal{D} = \{(X_i, Y_i) \mid X_i \in \mathbb{R}^d, Y_i \in \mathbb{R}\}_{i=1}^N$ , a kernel smoother estimate of the response for an unseen input query point  $X$  is as follows:

$$\hat{f}(X_0) = \sum_{i=1}^N W(X_i, X_0) Y_i \quad \text{where} \quad W(X_i, X_0) = \frac{K(\Delta(X_i, X_0))}{\sum_{j=1}^N K(\Delta(X_j, X_0))}, \quad (3.3)$$

$\Delta(\cdot, \cdot)$  is some distance on vectors (e.g. the Euclidean distance) and  $K(\cdot)$  is a kernel that decays with larger values (see Figure 3.3).

This smoothing estimator (3.3) is quite intuitive; it is essentially weighing the outputs of similar training instances higher in a weighted average across the training dataset (see Figure 3.4).

Inspecting the estimator (3.3), one will note that the main interaction between the query and the estimator is through the distance  $\Delta(\cdot, \cdot)$ . When considering vector data,  $\Delta$  is a distance over vectors, but one may naturally extend this to distributional covariates by

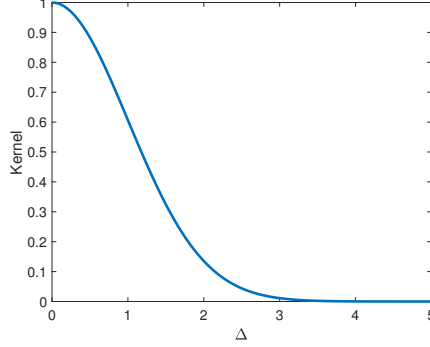


Figure 3.3: Example kernel  $K(\Delta) = \exp(-\frac{1}{2}\Delta^2)$ .

$$\hat{f}(\nearrow) = \overbrace{W(\swarrow, \nearrow)}^{\text{green bar}} Y_1 + \dots + \overbrace{W(\nwarrow, \nearrow)}^{\text{green bar}} Y_N$$

Figure 3.4: Graphical representation of the weighted average given by a kernel smoother (3.3). The outputs of more similar training instances (right) are weighted more than the outputs of less similar instances (left).

considering a distance over pdfs. For instance, we may consider the  $L_2$  metric over pdfs,  $\Delta(g, h) = \|g - h\|_2 = \sqrt{\int (g - h)^2}$ .

Recall that the data-set one works with for DRR is composed of pairs of input sample sets, and real valued responses (3.2) (Figure 3.2a). Thus, first one could use nonparametric estimators on input sample sets  $\{\mathcal{X}_1, \dots, \mathcal{X}_N\}$  to make density estimates  $\{\tilde{p}_1, \dots, \tilde{p}_N\}$  (Figure 3.2b). Similarly for an unseen query input sample set  $\mathcal{X}_0 \sim p_0$ , one makes a density estimate  $\tilde{p}_0$ . Then, one would compute the distances from the query  $\tilde{p}_0$ , to each training density:  $\Delta(\tilde{p}_1, \tilde{p}_0)$ ,  $\Delta(\tilde{p}_2, \tilde{p}_0)$ ,  $\dots$  (Figure 3.2c). That is, the Kernel-Kernel estimator is as follows:

$$\hat{f}(\tilde{p}_0) = \sum_{i=1}^N W(\tilde{p}_i, \tilde{p}_0) Y_i, \quad \text{where} \quad W(\tilde{p}_i, \tilde{p}_0) = \frac{K(\Delta(\tilde{p}_i, \tilde{p}_0))}{\sum_{j=1}^N K(\Delta(\tilde{p}_j, \tilde{p}_0))}. \quad (3.4)$$

That is, the Kernel-Kernel estimator (3.4), will output a linear combination of the training data outputs (Figure 3.5).

Clearly, the computation for (3.4) scales as  $\Omega(N)$  in terms of the number of input distributions in ones data-set. Furthermore, if one uses a Gaussian KDE, and takes  $\Delta(\tilde{p}_i, \tilde{p}_0) = \|\tilde{p}_i - \tilde{p}_0\|_2 = \sqrt{\int (\tilde{p}_i - \tilde{p}_0)^2}$  and  $n_i \asymp n$ , then the computation required for evaluating (3.4) is  $\Omega(Nn^2)$ .

Since evaluating the kernel smoothing estimator (3.4) for new predictions scales as  $\Omega(N)$  in the number of input/output instances in a data-set, the Kernel-Kernel estimator is not feasible for data-sets where the number of distributions,  $N$ , is in the high-thousands,



$$\hat{f}(\text{img}) = \overbrace{W(\text{img}_1, \text{img}_2)}^{\text{green bar}} Y_1 + \dots + \overbrace{W(\text{img}_N, \text{img}_N)}^{\text{green bar}} Y_N$$

Figure 3.5: Graphical representation of the weighted average given by a kernel smoother with distributional covariates (3.4). As with vector data, outputs of more similar training instances (right) are weighted more than the outputs of less similar instances (left).

millions, or even more. In this chapter we shall introduce an estimator for DRR, the Double-Basis estimator, which does not depend on  $N$  for evaluating an estimate for a new input distribution.

## 3.2 Double Basis Estimator

We now detail the Double Basis Estimator (2BE) for DRR. At a high level, the 2BE will embed densities into a non-linear space, where we can perform linear operations to regress our real valued response (Figure 3.6). To achieve this we make use of two bases: a set of orthonormal basis functions, to represent input densities as a vector of projections; a set of random basis functions to capture nonlinear relations in an approximate primal space to a kernel.

$$p \xrightarrow{\text{orthonormal basis}} \vec{a}(p) \xrightarrow{\text{random basis}} z(\vec{a}(p)) \rightarrow Y$$

Figure 3.6: High level illustration of the 2BE. First we make use of an orthonormal basis to represent the input density  $p$  as a vector of projections  $\vec{a}(p)$ , then we use a random basis to construct nonlinear random features  $z(\vec{a}(p))$ , finally these random features are linearly mapped to the real value response  $Y$ .

### 3.2.1 Orthonormal Basis

Recall from Section 2.1, that we may use orthonormal basis projection estimators for estimating the densities. That is, for each input sample set  $\mathcal{X}_i \stackrel{iid}{\sim} p_i$ , we shall build a density estimate using a set of orthonormal basis functions,  $\tilde{p}_i$ . These estimates are built through a vector of projection coefficients  $\vec{a}(\tilde{p}_i)$ , whose vector Euclidean distance will correspond to a  $L_2$  metric on respective density estimates: This (3.5) allows one to compare densities by comparing their respective vectors of projection coefficients, which will be useful as a featurization for constructing random features. Let  $\{\varphi_i\}_{i \in \mathbb{Z}}$  be an orthonormal basis for

$$\| \tilde{p} - \tilde{q} \|_2 = \| \vec{a}_M(\mathcal{X}) - \vec{a}_M(\mathcal{Y}) \|_2 = \quad (3.5)$$

$L_2([0, 1])$ . Recall that the outer product constitutes a basis for  $[0, 1]^l$ :

$$\{\varphi_\alpha\}_{\alpha \in \mathbb{Z}^l} \quad \text{where} \quad \varphi_\alpha(x) = \prod_{i=1}^l \varphi_{\alpha_i}(x_i), \quad x \in \Lambda^l.$$

Given a sample  $\mathcal{X}_i = \{x_{i1}, \dots, x_{in_i}\}$  where  $x_{ij} \stackrel{iid}{\sim} p_i$ , our estimator for  $p_i$  will be:

$$\tilde{p}_i(x) = \sum_{\|\alpha\| \leq t} a_\alpha(\mathcal{X}_i) \varphi_\alpha(x) \quad \text{where} \quad a_\alpha(\mathcal{X}_i) = \frac{1}{n_i} \sum_{j=1}^{n_i} \varphi_\alpha(x_{ij}), \quad (3.6)$$

for a chosen  $t \geq 0$  (see Section 2.1 for cross-validation). Here the vector of projection coefficients

$$\vec{a}(\tilde{p}_i) = (a_{\alpha_1}(\mathcal{X}_i), \dots, a_{\alpha_s}(\mathcal{X}_i)) \quad \text{where} \quad \|\alpha_j\| \leq t \quad (3.7)$$

represents the density (estimate).

### 3.2.2 Random Basis

Next, we shall use random basis functions from Random Kitchen Sinks (RKS) [102] to compute our estimate of the response. Rahimi and Recht [102] show that if one has a shift-invariant kernel  $K$  (in particular we consider the RBF kernel  $K(t) = \exp(-t^2/2)$ ) then for  $x, y \in \mathbb{R}^d$ :

$$K(\|x - y\|_2 / \sigma) \approx z(x)^T z(y), \quad \text{where} \quad (3.8)$$

$$z(x) \equiv \sqrt{\frac{2}{D}} [\cos(\omega_1^T x + b_1) \cdots \cos(\omega_D^T x + b_D)]^T \quad (3.9)$$

with  $\omega_i \stackrel{iid}{\sim} \mathcal{N}(0, \sigma^{-2} I_d)$ ,  $b_i \stackrel{iid}{\sim} \text{Unif}[0, 2\pi]$ . That is, the random basis functions

$$\left\{ z_i(x) = \sqrt{\frac{2}{D}} \cos(\omega_i^T x + b_i) \right\}_{i=1}^D$$

make an approximate basis for functions in the space induced by the kernel  $K$ :

$$f(x) = \sum_{i=1}^N \theta_i K(\|y_i - x\|_2 / \sigma) \approx \sum_{i=1}^N \theta_i z(y_i)^T z(x) = \left( \sum_{i=1}^N \theta_i z(y_i) \right)^T z(x) = \psi^T z(x), \quad (3.10)$$

where  $\psi \in \mathbb{R}^d$ . Thus, we may operate linearly on the random features in order to regress a response.

Coupled with the first orthonormal basis, we shall use this random basis to construct random features  $z(\vec{a}(\tilde{p}))$  that we can operate linearly on.



Let  $\omega_i \stackrel{iid}{\sim} \mathcal{N}(0, \sigma^{-2}I_s)$ ,  $b_i \stackrel{iid}{\sim} \text{Unif}[0, 2\pi]$ , be fixed. Let  $K_\sigma(x) = K(x/\sigma)$ . Then,

$$\hat{f}(\tilde{p}_0) = \sum_{i=1}^N \theta_i K_\sigma(\|\tilde{p}_i - \tilde{p}_0\|_2) \quad (3.11)$$

$$= \sum_{i=1}^N \theta_i K_\sigma(\|\vec{a}(\tilde{p}_i) - \vec{a}(\tilde{p}_0)\|_2) \quad (3.12)$$

$$\approx \sum_{i=1}^N \theta_i z(\vec{a}(\tilde{p}_i))^T z(\vec{a}(\tilde{p}_0)) \quad (3.13)$$

$$= \left( \sum_{i=1}^N \theta_i z(\vec{a}(\tilde{p}_i)) \right)^T z(\vec{a}(\tilde{p}_0)) \quad (3.14)$$

$$= \psi^T z(\vec{a}(\tilde{p}_0)) \quad (3.15)$$

where  $\psi = \sum_{i=1}^N \theta_i z(\vec{a}(\tilde{p}_i)) \in \mathbb{R}^D$ . Hence, we consider estimators of the form (3.15); that is, we consider linear estimators in the non-linear space induced by  $z(\vec{a}(\cdot))$ . Note that by doing so we have reduced a linear smoother, that scales  $\Omega(N)$  for new estimates, to a linear map in the  $D$  random features. Since  $D$  will be independent of  $N$  [91], we find that this is a great improvement computationally for large datasets ( $N \rightarrow \infty$ ). In particular, we consider the OLS estimator using the data-set  $\{(z(\vec{a}(\tilde{p}_i)), Y_i)\}_{i=1}^N$  :

$$\hat{f}(\tilde{p}_0) \equiv \hat{\psi}^T z(\vec{a}(\tilde{p}_0)) \text{ where} \quad (3.16)$$

$$\hat{\psi} \equiv \arg \min_{\beta} \|\vec{Y} - \mathbf{Z}\beta\|_2^2 \quad (3.17)$$

$$= (\mathbf{Z}^T \mathbf{Z})^{-1} \mathbf{Z}^T \vec{Y} \quad (3.18)$$

for  $\vec{Y} = (Y_1, \dots, Y_N)^T$ , and with  $\mathbf{Z}$  being the  $N \times D$  matrix:  $\mathbf{Z} = [z(\vec{a}(\tilde{p}_1)) \cdots z(\vec{a}(\tilde{p}_N))]$ . For further details, including an upperbound on risk, please see [91].

### 3.2.4 Evaluation Computational Complexity

We see that after computing  $\hat{\psi}$ , evaluating our estimator on a new distribution  $p_0$  amounts to taking an inner product with a  $D \times 1$  vector. Including the time required for computing  $z(\vec{a}(\tilde{p}_0))$ , the computation required for the evaluation,  $\hat{f}(\tilde{p}_0) = \hat{\psi}^T z(\vec{a}(\tilde{p}_0))$ , is: one, the time for evaluating the projection coefficients  $\vec{a}(\tilde{p}_0)$ ,  $O(sn)$ ; two, the time to compute the RKS features  $z(\cdot)$ ,  $O(Ds)$ ; three, the time to compute the inner product,  $\langle \hat{\psi}, \cdot \rangle$ ,  $O(D)$ . Hence, the total time is  $O(D + Ds + sn)$ . We can take  $D = O(n \log(n))$  and  $s = O(n)$  [91] hence the total run-time for evaluating  $\hat{f}(\tilde{p}_0)$  is  $O(n^2 \log(n))$ . Since we are considering data-sets where the number of instances  $N$  far outnumbers the number of points per sample set  $n$ ,  $O(n^2 \log(n))$  is a substantial improvement over  $O(Nn^2)$ .

### 3.2.5 Ridge Double Basis Estimator

We note that a straightforward extension to the Double Basis estimator is to use a ridge regression estimate on features  $z(\vec{a}_t(\cdot))$  rather than a OLS estimate. That is, for  $\lambda \geq 0$  let

$$\hat{\psi}_\lambda^T \equiv \arg \min_{\beta} \|\vec{Y} - \mathbf{Z}\beta\|_2^2 + \lambda \|\beta\|_2 \quad (3.19)$$

$$= (\mathbf{Z}^T \mathbf{Z} + \lambda I)^{-1} \mathbf{Z}^T \vec{Y}. \quad (3.20)$$

Clearly the Ridge Double Basis estimator is still evaluated via a dot product with  $\hat{\psi}_\lambda^T$ , and our above complexity analysis holds. Furthermore, we note that the Double Basis estimator is a special case of the Ridge Double Basis estimator with  $\lambda = 0$ .

## 3.3 Distribution to Distribution Regression

Next, we study distribution to distribution regression (DDR) where one aims to learn a mapping  $f$  that takes in a general input pdf covariate  $p : \mathbb{R}^l \mapsto \mathbb{R}$  and outputs a pdf response

$$q = f(p) : \mathbb{R}^k \mapsto \mathbb{R}. \quad (3.21)$$

As before, since we cannot directly observe pdfs, we cannot work over a data-set  $\bar{\mathcal{D}} = \{(p_i, q_i)\}_{i=1}^N$  where  $q_i = f(p_i)$ . Instead we shall work with a data-set of instances that are (inexact) observation pairs from input/output pdfs  $\mathcal{D} = \{(\mathcal{X}_i, \mathcal{Y}_i)\}_{i=1}^N$  where  $\mathcal{X}_i$ , and  $\mathcal{Y}_i$  are sample-sets from  $p_i$  and  $q_i$  respectively (Figure 3.9).

Dataset:

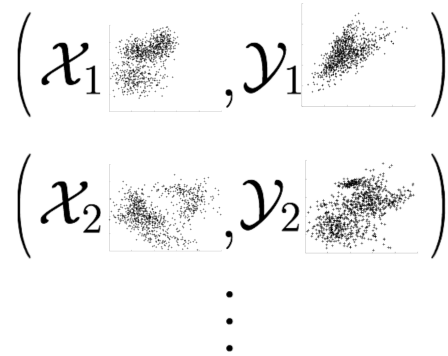
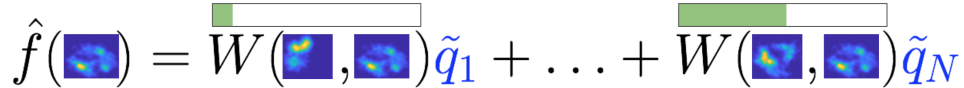


Figure 3.9: The dataset used for DDR tasks is composed of pairs of input and output sample-sets  $\mathcal{X}_i \stackrel{iid}{\sim} p_i$ ,  $\mathcal{Y}_i \stackrel{iid}{\sim} q_i$ .

The DDR framework is also quite general and includes many interesting problems. An example of a financial domain related DDR problem is learning the mapping that takes in the pdf of stock prices in a specific industry and outputs the pdf of stock prices in another industry. In biology one may be interested in regressing a mapping that takes in

the distribution of a certain protein in the body and outputs the distribution of a different protein in the body. Additionally, in cosmology one may be interested in regressing a mapping that takes in the pdf of simulated particles from a computationally inexpensive but inaccurate simulation and outputs the corresponding pdf of particles from a computationally expensive but accurate simulation. In essence, one would be enhancing the inaccurate simulation using previously seen data from accurate simulations.

### 3.3.1 Kernel Smoother Approach



$$\hat{f}(\text{input}) = W(\text{input}_1, \text{input}_1) \tilde{q}_1 + \dots + W(\text{input}_N, \text{input}_N) \tilde{q}_N$$

Figure 3.10: Weighted average given by a kernel smoother with distributional covariates and distributional responses.

In much the same way that we may extend a Nadaraya-Watson type estimator for DRR, we may also extend the Kernel-Kernel estimator (3.4) for distribution to distribution regression. As before the smoothing operator will come via a weighted average over training instances that is weighted according to a kernel similarity over input distributions (see Figure 3.10). However, in the case of DDR our output responses are now pdfs rather than reals. Still, our smoothing proceeds as before, but is now a functional average (Figure 3.11):

$$\hat{f}(\tilde{p}_0) = \sum_{i=1}^N W(\tilde{p}_i, \tilde{p}_0) \tilde{q}_i, \quad \text{where} \quad W(\tilde{p}_i, \tilde{p}_0) = \frac{K(\Delta(\tilde{p}_i, \tilde{p}_0))}{\sum_{j=1}^N K(\Delta(\tilde{p}_j, \tilde{p}_0))}. \quad (3.22)$$

While the averaging of output functions may seem abstract, the evaluation of the resulting estimate is merely an average of the real-valued training output function evaluations:

$$[\hat{f}(\tilde{p}_0)](x) = \sum_{i=1}^N W(\tilde{p}_i, \tilde{p}_0) \tilde{q}_i(x). \quad (3.23)$$

As before the resulting estimator will be quite flexible, but scales  $\Omega(N)$ , making its use in large datasets intractable. In order to mitigate this we extend the Double Basis Estimator for scalable distribution to distribution regression.

## 3.4 Triple Basis Estimator

The Double Basis Estimator is able to perform flexible, scalable distribution to real regression. However, for distribution to distribution regression where the output responses are entire functions rather than single reals, it is not immediately obvious how to extend the 2BE. The key to our extension, the Triple Basis Estimator (3BE) [92], is to consider an additional

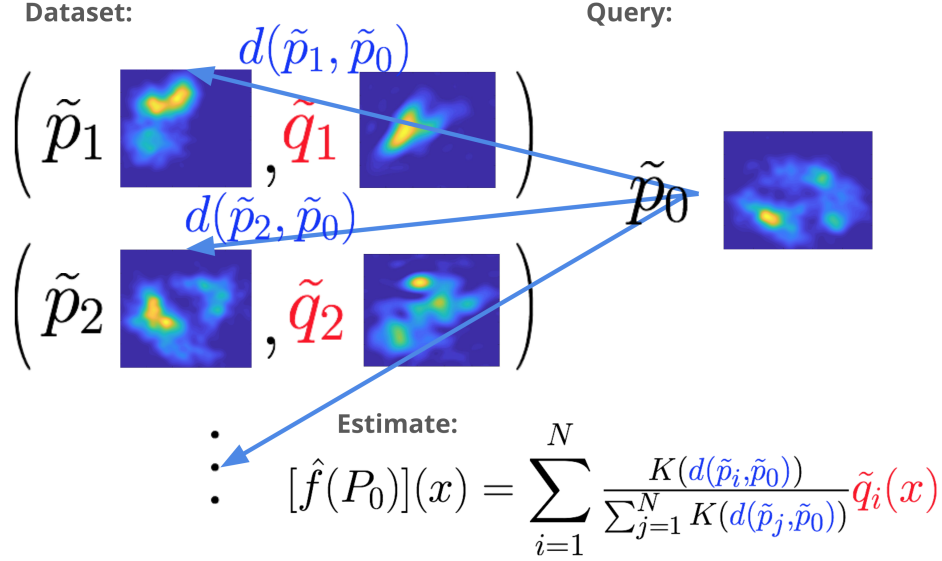


Figure 3.11: A linear smoother estimator on the DDR dataset will output a weighted average of output pdfs  $\tilde{q}_i$ , using distances  $d(\tilde{p}_i, \tilde{p}_j)$  on input pdfs. Estimated input/output pdfs,  $\tilde{p}_i$  and  $\tilde{q}_i$ , are estimated from input/output sample sets,  $\mathcal{X}_i$  and  $\mathcal{Y}_i$ , respectively.

$$p \xrightarrow{\text{orthonormal basis}} \vec{a}(p) \xrightarrow{\text{random basis}} z(\vec{a}(p)) \xrightarrow{\text{orthonormal basis}} \vec{a}(q) \rightarrow q$$

Figure 3.12: High level illustration of the 3BE. As with the 2BE first we make use of an orthonormal basis to represent the input density  $p$  as a vector of projections  $\vec{a}(p)$ , then we use a random basis to construct nonlinear random features  $z(\vec{a}(p))$ . However, we now also make use of an additional orthonormal basis to represent output functions  $q$ . The 3BE maps the random features  $z(\vec{a}(p))$  linearly to the vector of output function projection coefficients  $\vec{a}(q)$ .

basis to represent the functional responses,  $q$ . This, in effect, reduces the DDR task to multiple DRR tasks that we may regress using random features as before.

Recall that we represent an output pdf  $q = f(p) \in \mathcal{O}$  using its real valued projection coefficients, and approximate it using a finite set of basis functions  $M_{\mathcal{O}}$  and projections estimated using  $\mathcal{Y} \stackrel{iid}{\sim} q$ :

$$q(x) = [f(p)](x) = \sum_{\beta \in \mathbb{Z}^k} a_{\beta}(q) \varphi_{\beta}(x) \approx \sum_{\beta \in M_{\mathcal{O}}} a_{\beta}(\mathcal{Y}) \varphi_{\beta}(x) = \tilde{q}(x). \quad (3.24)$$

That is, if we know the  $r = |M_{\mathcal{O}}|$  projections of (the estimate of)  $q$ , then we can evaluate an estimate of  $q$  since the evaluation  $\tilde{q}(x)$  is just a linear combination of the basis functions weighted by the projections (see Figure 3.13).

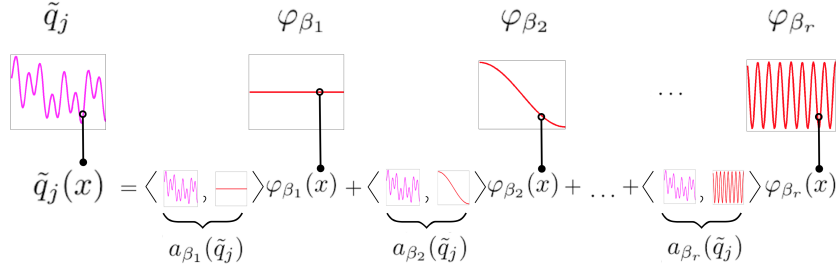


Figure 3.13: Since we are using orthonormal basis functions, our estimate of the output pdf is a linear combination of  $r$  basis functions  $\varphi_{\beta_i}$  weighted by respective projections. Thus, if we know the  $r$  projections, then we may estimate our output function.

Hence, we look to regress each of the output function projection coefficients given the input function. In other words, we look to perform  $r$  different DRR tasks, one for each of the  $r$  output projection coefficients (see Figure 3.14). These mappings will take in the input pdf (observed through a sample set  $\mathcal{X}$ ) and map it to an output function projection coefficient. We may write this estimate as:

$$\hat{q}(x) = \sum_{\beta \in M_{\mathcal{O}}} \hat{f}_{\beta}(\tilde{p}) \varphi_{\beta}(x). \quad (3.25)$$

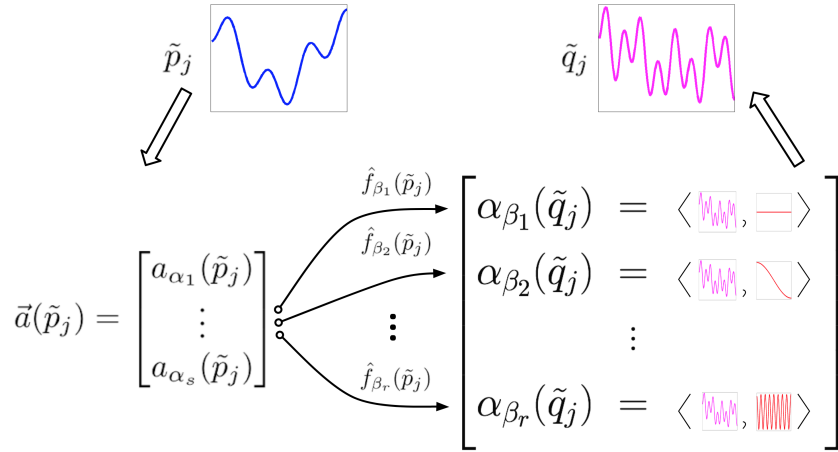


Figure 3.14: Framing the DDR task as multiple DRR tasks, one for each output function projection coefficient.

We wish to learn  $f_{\beta}$  in a scalable fashion, so we may proceed as the DRR case, and estimate  $f_{\alpha}$  using a linear map on the random features  $z(\vec{a}(\tilde{p}))$  (Figure 3.15). That is, the 3BE will be mapping  $z(\vec{a}(\tilde{p}))$  to  $\vec{a}(q)$  with a linear map on each coordinate,  $\vec{a}(q) \approx \Psi z(\vec{a}(\tilde{p}))$ ,  $a_{\beta}(q) \approx \psi_{\beta}^T z(\vec{a}(\tilde{p}))$ .

Suppose that we are given a data-set of input, output sample-set pairs  $\{(\mathcal{X}_i, \mathcal{Y}_i) \mid \mathcal{X}_i \stackrel{iid}{\sim} p_i, \mathcal{Y}_i \stackrel{iid}{\sim} q_i\}_{i=1}^N$ ; we build the 3BE estimator as a multiple linear regressor with the



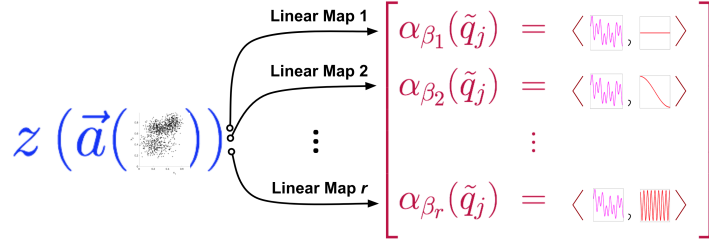


Figure 3.15: In order to perform the  $r$  DRR tasks (Figure 3.14) in a scalable fashion, we proceed as in the 2BE and use random features  $z$ . However, the 3BE will incorporate  $r$  distinct linear maps, one per output function projection coefficient.

data-set of input random feature, real valued projection coefficient vector responses  $\{(z(\vec{a}_{M_{\mathcal{I}}}(\mathcal{X}_i)), \vec{a}_{M_{\mathcal{O}}}(\mathcal{Y}_i))\}_{i=1}^N$ . Our estimator (3.25) is  $\hat{f}_{\alpha}(\mathcal{X}) = \hat{\psi}_{\alpha\lambda}^T z(\vec{a}_{M_{\mathcal{I}}}(\mathcal{X}))$  for  $\alpha \in M_{\mathcal{O}}$  where:

$$\hat{\psi}_{\beta\lambda} \equiv \arg \min_{\psi \in \mathbb{R}^D} \|\vec{Y}_{\alpha} - \mathbf{Z}\psi\|_2^2 + \lambda \|\beta\|_2^2 = (\mathbf{Z}^T \mathbf{Z} + \lambda I)^{-1} \mathbf{Z}^T \vec{Y}_{\alpha}, \quad (3.26)$$

$\lambda \geq 0$ , and  $\vec{Y}_{\alpha} = (a_{\alpha}(\mathcal{Y}_1), \dots, a_{\alpha}(\mathcal{Y}_N))^T$ . That is, we look to do multiple linear regression on a dataset of covariates that are the random features of input pdf projection coefficients  $z(\vec{a}(\tilde{p}_i))$ , and responses of output pdf projection coefficients,  $\vec{a}(\tilde{q}_i)$  (see Figure 3.16).

$$\{(z(\vec{a}(\tilde{p}_i)), \vec{a}(\tilde{q}_i))\}_{i=1}^N \rightarrow \vec{a}(\tilde{q}) \approx \Psi z(\vec{a}(\tilde{p}))$$

Figure 3.16: The 3BE takes in a dataset of random features input covariates and performs multiple linear regression to regress output responses of output pdf projection coefficients.

For further details and analysis of the 3BE please see [92].

### 3.5 Experiments

We perform experiments that demonstrate the ability of the Double Basis and Triple-Basis estimator to learn distribution-to-real mappings from large training datasets, which can be applied to yield fast, accurate, and useful predictions.

In all of the following experiments, we train on data of pairs of input sample-sets and real responses,  $\mathcal{D} = \{(\mathcal{X}_i, Y_i)\}_{i=1}^N$ , for DRR tasks and pairs of input sample-sets, output sample-sets,  $\mathcal{D} = \{(\mathcal{X}_i, \mathcal{Y}_i)\}_{i=1}^N$ , for DDR tasks.

### 3.5.1 Synthetic Mapping

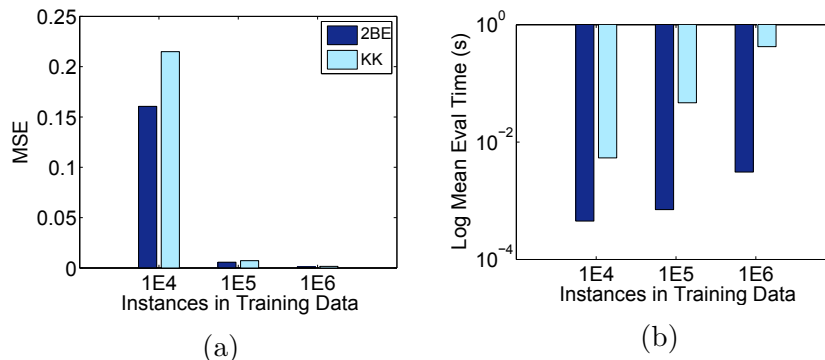


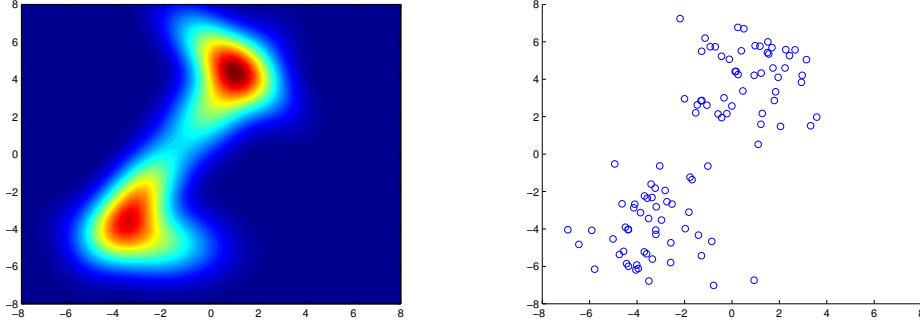
Figure 3.17: Results on predicting synthetic mapping  $f$ .

First, we look to emphasize the computational improvement in evaluation time of the Double Basis estimator over the linear smoothing Kernel-Kernel estimator using experiments with synthetic data. Our experiments are as follows. We first set  $N \in \{1E4, 1E5, 1E6\}$ . Then, we generate a random mapping  $f$  such that  $f(p) = \sum_{i=1}^{10} \theta_i K_\sigma(g_i, P)$ , where  $K_\sigma$  is the RBF kernel with bandwidth  $\sigma$ . We took  $\sigma = 1$ ,  $\theta_i \sim \text{Unif}[-5, 5]$ , and  $g_i$  to be the pdf of a mixture of two truncated Gaussians (each with weight .5) on the interval  $[0, 1]$ , whose mean locations are chosen uniformly at random in  $[0, 1]$ , and whose variance parameters are taken uniformly at random in  $[.05, .1]$ . For  $j = \{1, \dots, N\}$  we also set  $p_j$  to be a randomly generated mixture of two truncated Gaussians as previously described. We then generate  $Y_i$  under the noiseless case, i.e  $Y_i = f(p_i)$  (kernel values were computed numerically). Then, we generated  $\mathcal{X}_i = \{x_{i1}, \dots, x_{in}\}$  where  $n \propto N^{3/5}$  and  $x_{i1} \stackrel{iid}{\sim} p_i$ .  $\tilde{p}_i$  was then estimated using the samples  $\mathcal{X}_i$ .

We compared the performance of both the Double Basis (2BE), and the Kernel-Kernel (KK) estimator on a separate test set of  $\mathcal{D}_t = \{(\mathcal{X}_j, Y_j)\}_{j=1}^{N_t}$  where  $N_t = 1E5$ , that was generated as  $\mathcal{D}$  was. We measured performance in terms of mean squared error (MSE) and mean evaluation time per new query  $\mathcal{X}_0$  (Figures 3.17a and 3.17b respectively). One can see that in this case both estimators have similar MSEs, with the 2BE estimator doing somewhat better in each configuration of the data-set size. However, one can observe a striking difference in the average time to evaluate a new estimate  $\hat{f}(\tilde{p})$ . Figure 3.17b is presented in a log scale, and illustrates the Kernel-Kernel estimator’s lack of scaling on data-set size,  $N$ . On the other hand, the Double Basis estimator is considerably efficient even at large  $N$  and has a speed-up of about  $\times 12$ ,  $\times 67$ , and  $\times 139$  over the Kernel-Kernel estimate for  $N = 1E4, 1E5, 1E6$  respectively.

### 3.5.2 Choosing $k$ : model selection for Gaussian mixtures

Many common statistical tasks involve producing a mapping from a distribution to a real value, and may be tackled using DRR. One such task is that of model selection, where one is given a set  $\mathcal{X}_0 = \{x_{01}, \dots, x_{0n_0}\}$  drawn from an unknown distribution  $p$  and wants to



(a) Input pdf with 4 components.

(b) Corresponding sample set.

Figure 3.18: (a) A sample input distribution,  $p_i$ , and (b) sample set,  $\mathcal{X}_i$ . Based on the sample  $\mathcal{X}_i$  we would like to predict  $Y_i = 4$ , the number of components in the input pdf.

find some parameter that is indicative of the complexity of the true distribution. In other words, the mapping of interest takes in a distribution and outputs a hyperparameter of the distribution that is often illustrative of the distribution’s complexity.

In particular, we shall consider the model selection problem of selecting  $k$ , the number of components in a Gaussian mixture model (GMM). GMMs are often used in modeling data, however the selection of how many components to use is often a difficult choice. Attempting an MLE fit to training data will lead to choosing  $k = n_0$  with one mixture component corresponding to each data-point. Hence, in order to effectively select  $k$ , one must fit a GMM for each potential choice of  $k$  using an algorithm such as the expectation maximization algorithm (EM) [77], then select the choice of  $k$  that optimizes some score. In practice this often becomes computationally expensive. Typically scores used include Akaike information criterion (AIC), Bayesian information criterion (BIC), or a cross-validated data-fitting score on a holdout set (CV). We note that often GMMs are used to cluster data, where each data-point  $x_{0i}$  is assigned to a cluster based on which mixture component most likely generated it. Hence, the problem of selecting the number of mixture components in a GMM is closely related to the problem of selecting the number of clusters to use, which is itself a difficult problem.

Since selecting  $k$  in GMMs is a DRR problem, and it is a relatively smooth mapping (that is, similar distributions should have a similar number of components), we hypothesize that one may learn to perform model selection in GMMs using the Double Basis estimator. Particularly, by using a supervised dataset of {sample-set,  $k$ } pairs, the Double Basis estimator will be able to leverage previously seen data to perform model selection for a new unseen input sample set.

Our experiment proceeds as follows. We can generate our own training data for this task by randomly drawing a value for  $k$  (over some bounded range), then drawing 2-dimensional Gaussian mixture parameters for each of the  $k$  components, and finally drawing samples from each Gaussian. That is, we generate  $N = 28,000$  input sample set/ $k$  response pairs:  $\mathcal{D} = \{(\mathcal{X}_i, k_i)\}_{i=1}^N$ , where  $\mathcal{X}_i = \{x_{i1}, \dots, x_{in}\}$ ,  $x_{ij} \in \mathbb{R}^2$ ,  $x_{ij} \stackrel{iid}{\sim} \text{GMM}(k_i)$ ,  $k_i \sim \text{Unif}\{1, \dots, 10\}$ ,

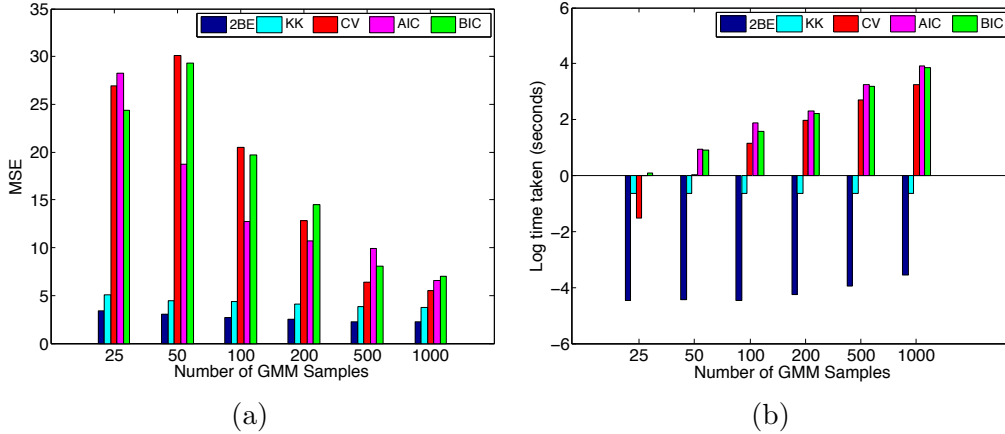


Figure 3.19: Results on predicting the number of GMM components.

and  $\text{GMM}(k_i)$  is a random GMM generated as follows, for  $j = 1, \dots, k_i$ : the prior weights for each component is taken to be  $\pi_j = 1/k_i$ ; the means are  $\mu_j \sim \text{Unif}[-5, 5]^2$ ; and covariances are  $\Sigma_j = a^2 AA^T + B$ , where  $a \sim \text{Unif}[1, 2]$ ,  $A_{uv} \sim \text{Unif}[-1, 1]$ , and  $B$  is a diagonal  $2 \times 2$  matrix with  $B_{uu} \sim \text{Unif}[0, 1]$ . We train and get results using  $n$  in the following range:  $n \in 10, 25, 50, 200, 500, 1000$ . We perform model selection using the mapping learned by the Ridge Double Basis estimator (3.19) (denoted 2BE in experiments), and compare it with model selection via AIC, BIC, and CV. We also compare against the Kernel-Kernel (KK) smoother. For all methods we computed the mean squared error between the true and predicted value for  $k$  over 2000 test sample sets (Figure 3.19). We see that the Double Basis estimator has both the lowest MSE and the lowest average evaluation time for computing a new prediction. In fact, the Double Basis estimator can carry out the model selection prediction orders of magnitude faster than the CV, AIC, or BIC procedures.

### 3.5.3 Low Sample Dirichlet Parameter Estimation

Similar to model selection, general parameter point estimation is a statistical task that may be posed as a DRR problem. That is, in parameter estimation one considers a set  $\mathcal{X}_0 = \{x_{01}, \dots, x_{0n_0}\}$  where points are drawn from some distribution  $p(\eta_0)$  that is parameterized by  $\eta_0$ , and attempts to estimate  $\eta_0$ . In particular, we use DRR and the Double Basis estimator to perform parameter estimation for Dirichlet distributions. The Dirichlet distribution is a family of continuous, multivariate distributions parameterized by a vector  $\alpha \in \mathbb{R}_+^d$ , with support over the  $d$ -simplex. Since every element of the support sums to one, the Dirichlet is often used to model distributions over proportion data. As before, we hypothesize that the Double Basis estimator will serve as a way to leverage previously seen sample sets to help perform parameter estimation for new unseen sets. Effectively, our estimator will be able to “boost” the sample-size of a new input sample set by making use of what was learned on previously seen labeled sample sets.

Maximum likelihood parameter estimation for  $\alpha$ , given a set of Dirichlet samples, is often performed via iterative optimization algorithms, such as gradient ascent or Newton’s

method [74], as a closed form solution for the MLE does not appear to exist in the literature. In this experiment, we aim to use DDR as a new method for Dirichlet parameter estimation. In particular, we generate samples from Dirichlet distributions with parameter values in a pre-specified range, and use these as training data to learn a mapping from data samples to Dirichlet  $\alpha$  parameter values.

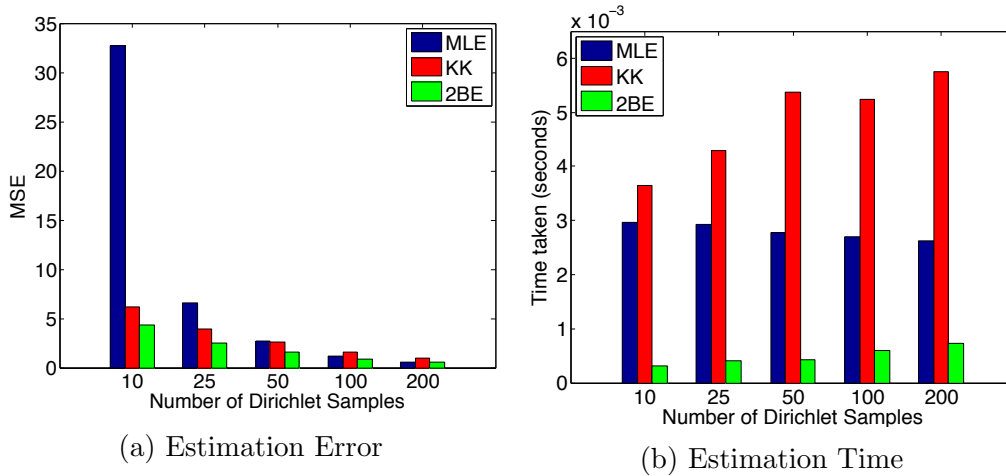


Figure 3.20: Results predicting Dirichlet parameters.

In our experiments, we first fix the range of  $\alpha$  values to be constrained such that the  $i^{\text{th}}$  component  $\alpha_i \in [0.1, 10]$ . For each 28,000 training instances, we uniformly sample a new  $\alpha$  parameter vector within this range, and then generate  $n$  points from the associated Dirichlet( $\alpha$ ) distribution, where  $n \in \{10, 25, 50, 200, 500, 1000\}$ . We compare the Ridge Double Basis estimator (3.19) against a Newtons-method procedure for maximum likelihood estimation (MLE) from the fastfit toolbox [75], and again against the Kernel-Kernel smoother. For all methods, for each  $n$ , we compute the mean squared error between the true and the estimated  $\alpha$  parameter. We also record the time taken to perform the parameter estimation in each case. Results are shown in Figure 3.20. We see that the Double Basis estimator achieves the lowest MSE in all cases, and has the lowest average compute time. It is worth noting that the Double Basis estimator performs particularly well relative to the MLE in cases where the sample size is low. We envision that Double Basis estimator is particularly well suited for cases where one hopes to quickly, and in an automatic fashion, construct an estimator that can achieve highly accurate results for a statistical estimation problem for which an optimal estimator might be hard to derive analytically.

### 3.5.4 Rectifying 2LPT Simulations

Numerical simulations have become an essential tool to study cosmological structure formation. Cosmologists use N-body simulations [121] to study the gravitational evolution of collisionless particles like dark matter particles. Unfortunately, N-body simulations require forces among particles to be recomputed over multiple time intervals, leading to a large magnitude of time steps to complete a single simulation. In order to mitigate the large

computational costs of running N-body simulations, often simulations based on Second Order Lagrange Perturbation Theory (2LPT) [110] are used.

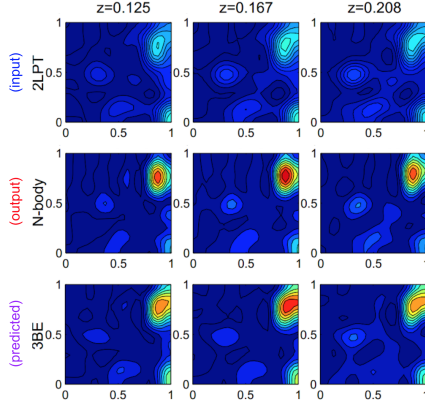


Figure 3.21: Slices of particle pdfs coming from the input pdfs (resulting particles from 2LPT simulation), the output pdf (resulting particles from N-body simulation), and the predicted pdf (from the 3BE).

Method	MSE	MPT(s)
<b>3BE</b>	<b>4.958</b>	<b>0.009</b>
LSE	6.816	4.977
2LPT	6.424	NA
AD	9.289	NA

Table 3.1: MSE and mean prediction time (MPT) results.

Although 2LPT simulations are several orders of magnitude faster, they prove to be inaccurate, especially at smaller scales. In this experiment we bridge the gap between the speed of 2LPT simulations and the accuracy of N-body simulations using DDR and the 3BE. Namely, we regress the mapping between a distribution of particles in an area coming from a 2LPT simulation and the distribution of the particles in the same area under an equivalent N-body simulation (see Figure 3.21).

We regress the distribution of 3d (spatial) N-body simulation particles in  $16 \text{ Mpc}^3$  cubes when given the distribution of particles of the 2LPT simulation in the same cube (note that each distribution is estimated through the set of particles in each cube). A training-set of over 900K pairs of 2LPT cube sample-set/N-body cube sample-set instance was used, along with a test-set of 5K pairs. The number of projection coefficients used to represent input and output distributions was 365/401 respectively, chosen by cross-validating the density estimates. We chose the number of RKS features to be 15K based on rules-of-thumb. We cross-validated the  $\sigma$  and  $\lambda$  parameters of the ridge variant 3BE (3.19) and the smoothing parameter of the LSE and reported back the MSE and mean prediction time (MPT, in seconds) of our DDR estimates to the distributions truly coming directly through N-body simulation (Table 3.1); we also report the MSE of predicting the average output distribution (AD).

We see that the 3BE is about  $500\times$  faster than the LSE in terms of prediction time and achieved an improvement in  $R^2$  of over 50% over using the distribution coming directly from the 2LPT simulation (2LPT). Note also that the LSE does not achieve an improvement in MSE over 2LPT.

# Chapter 4

## Beyond the Euclidean Metric: Random Features for Homogenous Density Distances

In the previous chapter we showed how to perform scalable DRR and DDR tasks by using random features on vectors of projection coefficients. These methods operate over input distributions in a Euclidean space through the  $L_2$  metric. However, there are a myriad of other useful metrics available, such as total variation, Hellinger distance, and the Jensen-Shannon divergence.

This chapter develops the first random features for pdfs whose dot product approximates kernels using these non-Euclidean metrics, which we coin *homogeneous density distances* (HDDs) [119]. These random features allow estimators to scale to large datasets by working in a primal space, without computing large Gram matrices. We show empirically the quality of our approximation in solving learning tasks in real-world and synthetic data.

### 4.1 Embedding Information Theoretic Kernels

For a broad class of distributional distances  $\Delta$ , including many common and useful information theoretic divergences, we consider generalized RBF kernels of the form

$$K(p, q) = \exp\left(-\frac{1}{2\sigma^2}\Delta^2(p, q)\right), \quad (4.1)$$

for pdfs  $p, q: [0, 1]^\ell \rightarrow \mathbb{R}^+$ . At a high-level we construct features  $z(A(\cdot))$  such that  $K(p, q) \approx z(A(p))^T z(A(q))$  as follows:

**Embedding HDDs into  $L_2$**  We define a random function  $\psi$  such that

$$\Delta(p, q) \approx \|\psi(p) - \psi(q)\|_2, \quad (4.2)$$

where  $\psi(p)$  is a function from  $[0, 1]^\ell$  to  $\mathbb{R}^{2M}$ . Thus the metric space of densities with distance  $\Delta$  is approximately embedded into the metric space of  $2M$ -dimensional  $L_2$  functions.

**Finite Embeddings of  $L_2$**  We use orthonormal basis functions to approximately embed smooth  $L_2$  functions into finite vectors in  $\mathbb{R}^{|V|}$ . Combined with the previous step, we obtain features  $A(p) \in \mathbb{R}^{2M|V|}$  such that  $\Delta$  is approximated by Euclidean distances between the  $A(\cdot)$  vectors.

**Embedding RBF Kernels into  $\mathbb{R}^D$**  We use the random feature embedding  $z(\cdot)$  so that inner products between  $z(A(\cdot))$  features, in  $\mathbb{R}^D$ , approximate  $K(p, q)$ .

We can thus use the powerful kernel  $K$  without needing to compute an expensive  $N \times N$  Gram matrix.

### 4.1.1 Homogeneous Density Distances (HDDs)

We consider kernels based on metrics which we term homogeneous density distances (HDDs):

$$\Delta^2(p, q) = \int_{[0,1]^\ell} \kappa(p(x), q(x)) dx, \quad (4.3)$$

where  $\kappa(x, y) : \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$  is a negative-type kernel, i.e. a squared Hilbertian metric, and  $\kappa(tx, ty) = t\kappa(x, y)$  for all  $t > 0$ . Table 4.1 shows a few important instances. Note we assume the distributions are supported within  $[0, 1]^\ell$ .

Name	$\kappa(p(x), q(x))$	$d\mu(\lambda)$
JS	$\frac{p(x)}{2} \log \left( \frac{2p(x)}{p(x)+q(x)} \right) + \frac{q(x)}{2} \log \left( \frac{2q(x)}{p(x)+q(x)} \right)$	$\frac{d\lambda}{\cosh(\pi\lambda)(1+\lambda^2)}$
H <sup>2</sup>	$\frac{1}{2} \left( \sqrt{p(x)} - \sqrt{q(x)} \right)^2$	$\frac{1}{2} \delta(\lambda = 0) d\lambda$
TV	$ p(x) - q(x) $	$\frac{2}{\pi} \frac{1}{1+4\lambda^2} d\lambda$

Table 4.1: Squared HDDs. JS is Jensen-Shannon divergence; H is Hellinger distance; TV is total variation distance.

We then use these distances in a generalized RBF kernel Equation (4.1).  $\Delta$  is a Hilbertian metric [32], so  $K$  is positive definite [45]. Note we use the  $\sqrt{\text{TV}}$  metric, even though TV is itself a metric.

Below we expound on the embeddings used to construct features  $z(A(\cdot))$  such that  $K(p, q) \approx z(A(p))^T z(A(q))$ .

### 4.1.2 Embedding HDDs into $L_2$

Fuglede [32] shows that  $\kappa$  corresponds to a bounded measure  $\mu(\lambda)$ , as in Table 4.1, with

$$\kappa(x, y) = \int_{\mathbb{R}^+} |x^{\frac{1}{2}+i\lambda} - y^{\frac{1}{2}+i\lambda}|^2 d\mu(\lambda). \quad (4.4)$$

Let  $Z = \mu(\mathbb{R}^+)$  and  $c_\lambda = (-\frac{1}{2} + i\lambda)/(\frac{1}{2} + i\lambda)$ ; then

$$\kappa(x, y) = \mathbb{E}_{\lambda \sim \frac{\mu}{Z}} [ |g_\lambda(x) - g_\lambda(y)|^2 ]$$



where  $g_\lambda(x) = \sqrt{Z}c_\lambda(x^{\frac{1}{2}+i\lambda} - 1)$ .

We can approximate the expectation with an empirical mean. Let  $\lambda_j \stackrel{iid}{\sim} \frac{\mu}{Z}$  for  $j \in \{1, \dots, M\}$ ; then,

$$\kappa(x, y) \approx \frac{1}{M} \sum_{j=1}^M |g_{\lambda_j}(x) - g_{\lambda_j}(y)|^2.$$

Hence, using  $\Re, \Im$  to denote the real and imaginary parts,  $\Delta^2(p, q)$  is equal to:

$$\int_{[0,1]^\ell} \kappa(p(x), q(x)) dx \tag{4.5}$$

$$= \int_{[0,1]^\ell} \mathbb{E}_{\lambda \sim \frac{\mu}{Z}} |g_\lambda(p(x)) - g_\lambda(q(x))|^2 dx \tag{4.6}$$

$$\approx \frac{1}{M} \sum_{j=1}^M \int_{[0,1]^\ell} \left( (\Re(g_{\lambda_j}(p(x))) - \Re(g_{\lambda_j}(q(x))))^2 + (\Im(g_{\lambda_j}(p(x))) - \Im(g_{\lambda_j}(q(x))))^2 \right) dx \tag{4.7}$$

$$= \|\psi(p) - \psi(q)\|^2, \tag{4.8}$$

where

$$[\psi(p)](x) = \frac{1}{\sqrt{M}} (p_{\lambda_1}^R(x), \dots, p_{\lambda_M}^R(x), p_{\lambda_1}^I(x), \dots, p_{\lambda_M}^I(x)),$$

defining  $p_{\lambda_j}^R(x) = \Re(g_{\lambda_j}(p(x)))$ ,  $p_{\lambda_j}^I(x) = \Im(g_{\lambda_j}(p(x)))$ . Hence, the HDD between densities  $p$  and  $q$  is approximately the  $L_2$  distance from  $\psi(p)$  to  $\psi(q)$ , where  $\psi$  maps a function  $f : [0, 1]^\ell \mapsto \mathbb{R}$  to a vector-valued function  $\psi(f) : [0, 1]^\ell \mapsto \mathbb{R}^{2M}$  of  $\lambda$  functions.  $M$  can typically be quite small, since the kernel it approximates is one-dimensional.

### 4.1.3 Finite Embeddings of $L_2$

If densities  $p$  and  $q$  are smooth, then the  $L_2$  metric between the  $p_\lambda$  and  $q_\lambda$  functions may be well approximated using projections to basis functions. Recall that if  $\{\varphi_i\}_{i \in \mathbb{Z}}$  is an orthonormal basis for  $L_2([0, 1])$ ; then we can construct an orthonormal basis for  $L_2([0, 1]^\ell)$  by the tensor product:

$$\begin{aligned} \{\varphi_\alpha\}_{\alpha \in \mathbb{Z}^\ell} \quad \text{where} \quad \varphi_\alpha(x) &= \prod_{i=1}^{\ell} \varphi_{\alpha_i}(x_i), \quad x \in [0, 1]^\ell, \\ \forall f \in L_2([0, 1]^\ell), \quad f(x) &= \sum_{\alpha \in \mathbb{Z}^\ell} a_\alpha(f) \varphi_\alpha(x) \end{aligned}$$

and  $a_\alpha(f) = \langle \varphi_\alpha, f \rangle = \int_{[0,1]^\ell} \varphi_\alpha(t) f(t) dt \in \mathbb{R}$ . Let  $V \subset \mathbb{Z}^\ell$  be an appropriately chosen finite set of indices. If  $f, f' \in L_2([0, 1]^\ell)$  are smooth and  $\vec{a}(f) = (a_{\alpha_1}(f), \dots, a_{\alpha_{|V|}}(f))$ , then

$\|f - f'\|^2 \approx \|\vec{a}(f) - \vec{a}(f')\|^2$ . Thus we can approximate  $\Delta^2$  as the squared distance between finite vectors:

$$\begin{aligned} \Delta^2(p, q) &\approx \|\psi(p) - \psi(q)\|^2 \\ &\approx \frac{1}{M} \sum_{j=1}^M \|\vec{a}(p_{\lambda_j}^R) - \vec{a}(q_{\lambda_j}^R)\|^2 + \|\vec{a}(p_{\lambda_j}^I) - \vec{a}(q_{\lambda_j}^I)\|^2 \\ &= \|A(p) - A(q)\|^2 \end{aligned} \quad (4.9)$$

where  $A : L_2([0, 1]^\ell) \rightarrow \mathbb{R}^{2M|V|}$  has  $A(p)$  given by

$$\frac{1}{\sqrt{M}} (\vec{a}(p_{\lambda_1}^R), \dots, \vec{a}(p_{\lambda_M}^R), \vec{a}(p_{\lambda_1}^I), \dots, \vec{a}(p_{\lambda_M}^I)). \quad (4.10)$$

We will discuss how to estimate  $\vec{a}(p_{\lambda}^R)$ ,  $\vec{a}(p_{\lambda}^I)$  shortly.

#### 4.1.4 Embedding RBF Kernels into $\mathbb{R}^D$

The  $A$  features approximate the HDD (4.3) in  $\mathbb{R}^{2M|V|}$ ; thus applying the RKS embedding [102] to the  $A$  features will approximate our generalized RBF kernel (4.1). The RKS embedding is  $z : \mathbb{R}^m \rightarrow \mathbb{R}^D$  such that for fixed  $\omega_i \stackrel{iid}{\sim} \mathcal{N}(0, \sigma^{-2}I_m)$ ,  $b_i \stackrel{iid}{\sim} \text{Unif}[0, 2\pi]$  and for each  $x, y \in \mathbb{R}^m$ :

$$\begin{aligned} z(x)^T z(y) &\approx \exp\left(-\frac{1}{2\sigma^2} \|x - y\|^2\right), \text{ where} \\ z(x) &\equiv \sqrt{\frac{2}{D}} [\cos(\omega_1^T x + b_1) \cdots \cos(\omega_D^T x + b_D)]^T. \end{aligned} \quad (4.11)$$

Thus we can approximate the HDD kernel (4.1) as:

$$\begin{aligned} K(p, q) &= \exp\left(-\frac{1}{2\sigma^2} \Delta^2(p, q)\right) \\ &\approx \exp\left(-\frac{1}{2\sigma^2} \|A(p) - A(q)\|^2\right) \\ &\approx z(A(p))^T z(A(q)). \end{aligned} \quad (4.12)$$

#### 4.1.5 Finite Sample Estimates

Our final approximation for HDD kernels (4.12) depends on integrals of densities  $p$  and  $q$ . In practice, we are unlikely to directly observe an input density, but even given a pdf  $p$ , the integrals that make up the elements of  $A(p)$  are not readily computable. We thus first estimate the density as  $\hat{p}$ , e.g. with kernel density estimation (KDE), and estimate  $A(p)$  as  $A(\hat{p})$ . Recall that the elements of  $A(\hat{p})$  are:

$$a_\alpha(\hat{p}_{\lambda_j}^S) = \int_{[0,1]^\ell} \varphi_\alpha(t) \hat{p}_{\lambda_j}^S(t) dt \quad (4.13)$$

where  $j \in \{1, \dots, M\}$ ,  $S \in \{R, I\}$ ,  $\alpha \in V$ . In lower dimensions, we can approximate (4.13) with simple Monte Carlo numerical integration. Choosing  $\{u_i\}_{i=1}^{n_e} \stackrel{iid}{\sim} \text{Unif}([0, 1]^\ell)$ :

$$\hat{a}_\alpha(\hat{p}_{\lambda_j}^S) = \frac{1}{n_e} \sum_{i=1}^{n_e} \varphi_\alpha(u_i) \hat{p}_{\lambda_j}^S(u_i), \quad (4.14)$$

obtaining  $\hat{A}(\hat{p})$ . We note that in high dimensions, one may use any high-dimensional density estimation scheme (e.g. [61]) and estimate (4.13) with MCMC techniques (e.g. [48]).

### 4.1.6 Summary and Complexity

The algorithm for computing features  $\{z(A(p_i))\}_{i=1}^N$  for a set of distributions  $\{p_i\}_{i=1}^N$ , given sample sets  $\{\mathcal{X}_i\}_{i=1}^N$  where  $\mathcal{X}_i = \{x_j^{(i)} \in [0, 1]^\ell\}_{j=1}^{n_i} \stackrel{iid}{\sim} p_i$ , is thus:

1. Draw  $M$  scalars  $\lambda_j \stackrel{iid}{\sim} \frac{\mu}{Z}$ ,  $D$  vectors  $\omega_r \stackrel{iid}{\sim} \mathcal{N}(0, \sigma^{-2} I_{2M|V|})$ , and  $b_i \stackrel{iid}{\sim} \text{Unif}[0, 2\pi]$ , in  $O(M|V|D)$  time.
2. For each of the  $N$  input distributions  $i$ :
  - (a) Compute a kernel density estimate from  $\mathcal{X}_i$ ,  $\hat{p}_i(u_j)$  for each  $u_j$  in (4.14), in  $O(n_i n_e)$  time.
  - (b) Compute  $\hat{A}(\hat{p}_i)$  using a numerical integration estimate as in (4.14), in  $O(M|V|n_e)$  time.
  - (c) Get the RKS features,  $z(\hat{A}(\hat{p}_i))$ , in  $O(M|V|D)$  time.

Supposing each  $n_i \asymp n$ , this process takes a total of  $O(Nnn_e + NM|V|n_e + NM|V|D)$  time. Taking  $|V|$  to be asymptotically  $O(n)$ ,  $n_e = O(D)$ , and  $M = O(1)$  for simplicity, this is  $O(NnD)$  time, compared to about  $O(N^2n \log n + N^3)$  for the methods of Póczos et al. [101] and  $O(N^2n^2)$  for Muandet et al. [79].

## 4.2 Experiments

Below we present several select experiments ran on both synthetic and real world data-sets. Throughout these experiments we use  $M = 5$ ,  $|V| = 10^\ell$  (selected as rules of thumb; larger values did not improve performance), and use a validation set (10% of the training set) to choose bandwidths for KDE and the RBF kernel as well as model regularization parameters. Except in the scene classification experiments, the histogram methods used 10 bins per dimension; performance with other values was not better. The KL estimator used the fourth nearest neighbor.

We evaluate RBF kernels based on various distances. First, we try our JS, Hellinger, and TV embeddings. We compare to  $L_2$  kernels as in Oliva et al. [91]:  $\exp\left(-\frac{1}{2\sigma^2}\|p - q\|_2^2\right) \approx z(\vec{a}(\hat{p}))\tilde{p}z(\vec{a}(\hat{q}))$  (L2). We also try the MMD distance [79] with approximate kernel embeddings:  $\exp\left(-\frac{1}{2\sigma^2}\widehat{\text{MMD}}(p, q)\right) \approx z(\vec{z}(\hat{p}))\tilde{p}z(\vec{z}(\hat{q}))$ , where  $\vec{z}$  is the mean embedding  $\vec{z}(\hat{p}) = \frac{1}{n} \sum_{i=1}^n z(X_i)$  (MMD). We further compare to RKS with histogram JS embeddings [130] (Hist JS); we also tried  $\chi^2$  embeddings, but their performance was quite similar. We

finally try the full Gram matrix approach of Póczos et al. [101] with the KL estimator of Wang, Kulkarni, and Verdú [133] in an RBF kernel (KL), as did Ntampaka et al. [88].

### Estimating the Number of Mixture Components

We revisit the use of the random features on densities for estimating the number of components from a mixture of truncated Gaussians (Section 3.5.2). We generate the distributions as follows: Draw the number of components  $Y_i$  for the  $i$ th distribution,  $p_i$ , as  $Y_i \sim \text{Unif}\{1, \dots, 10\}$ . For each component select a mean  $\mu_k^{(i)} \sim \text{Unif}[-5, 5]^2$  and covariance  $\Sigma_k^{(i)} = a_k^{(i)} A_k^{(i)} A_k^{(i)\top} + B_k^{(i)}$ , where  $a \sim \text{Unif}[1, 4]$ ,  $A_k^{(i)}(u, v) \sim \text{Unif}[-1, 1]$ , and  $B_k^{(i)}$  is a diagonal  $2 \times 2$  matrix with  $B_k^{(i)}(u, u) \sim \text{Unif}[0, 1]$ . Then weight each component equally in the mixture. Given a sample  $\mathcal{X}_i \stackrel{iid}{\sim} p_i$ , we predict the number of components  $Y_i$ . An example distribution and sample are shown in Figure 4.1; predicting the number of components is difficult even for humans.

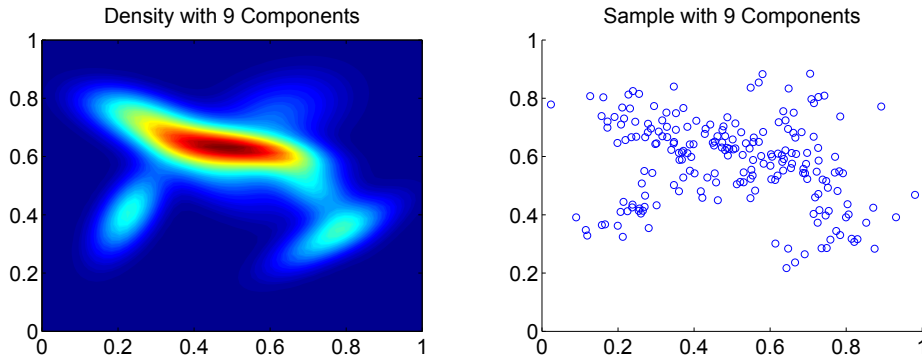


Figure 4.1: A GMM and 200 points drawn from it.

Figure 4.2 presents results for predicting with ridge regression the number of mixture components  $Y_i$ , given a varying number of sample sets  $\mathcal{X}_i$ , with  $|\mathcal{X}_i| \in \{200, 800\}$ ; we use  $D = 5000$ . The HDD-based kernels achieve lower error than the  $L_2$  and MMD kernels. They also outperform a histogram kernel approach [129], especially with  $|\mathcal{X}_i| = 200$ , and the KL kernel. Note that fitting mixtures with EM and selecting a number of components using AIC [3] or BIC [109] performed much worse than regression; only AIC with  $|\mathcal{X}_i| = 800$  outperformed the best constant predictor of 5.5. Linear versions of the  $L_2$  and MMD kernels were also no better than the constant predictor.

The HDD embeddings were more computationally expensive than the other embeddings, but much less expensive than the KL kernel, which grows at least quadratically in the number of distributions. Note that the histogram embeddings used an optimized C implementation [128], as did the KL kernel<sup>1</sup>, while the HDD embeddings used a simple Matlab implementation.

<sup>1</sup>[github.com/dougalsutherland/skl-groups/](https://github.com/dougalsutherland/skl-groups/)

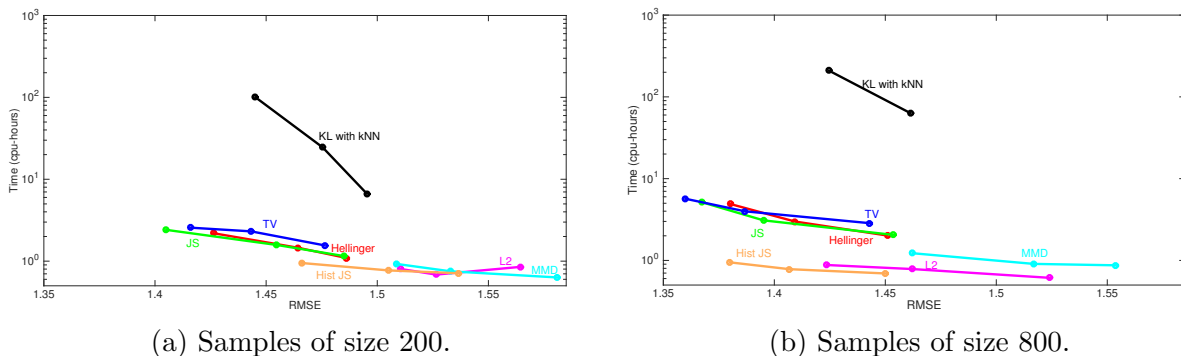


Figure 4.2: Error and computation time for estimating the number of mixture components. The three points on each line correspond to training set sizes of 4K, 8K, and 16K; error is on the fixed test set of size 2K. Note the logarithmic scale on the time axis. The KL kernel for  $|\mathcal{X}_i| = 800$  with 16K training sets was too slow to run. AIC-based predictions achieved RMSEs of 2.7 (for 200 samples) and 2.3 (for 800); BIC errors were 3.8 and 2.7; a constant predictor of 5.5 had RMSE of 2.8.

## Scene Classification

Modern computer vision classification systems typically consist of a deep network with several convolutional and pooling layers to extract complex features of input images, followed by one or two fully-connected classification layers. The activations are of shape  $n \times h \times w$ , where  $n$  is the number of filters; each unit corresponds to an overlapping patch of the original image. We can thus treat the final pooled activations as a sample of size  $hw$  from an  $n$ -dimensional distribution, similarly to how [101] and [79] used SIFT features from image patches. [139] set accuracy records on several scene classification datasets with a particular ad-hoc method of extracting features from distributions (D3); we compare to our more principled alternatives.

We consider the Scene-15 dataset [64], which contains 4485 natural images in 15 location categories, and follow [139] in extracting features from the last convolutional layer of the `imagenet-vgg-verydeep-16` model [113]. We replace that layer’s rectified linear activations with sigmoid squashing to  $[0, 1]$ .<sup>2</sup>  $hw$  ranges from 400 to 1000. There are 512 filter dimensions; we concatenate features extracted from each independently.

We train on the standard for this dataset of 100 images from each class (1500 total) and test on the remainder; Figure 4.3 shows results. We did not include spatial information; still, we match the best prior published performance of  $91.59 \pm 0.48$ , trained on a large scene classification dataset [146]. Adding spatial information brought the D3 method to about 92% accuracy; their best hybrid method obtained 92.9%. With these features, however, our methods match or beat MMD and substantially outperform D3,  $L_2$ , and the histogram embeddings.

<sup>2</sup>We used piecewise-linear weights before the sigmoid function such that 0 maps to 0.5, the 90th percentile of the positive observations maps to 0.9, and the 10th percentile of the negative observations to 0.1, for each filter.

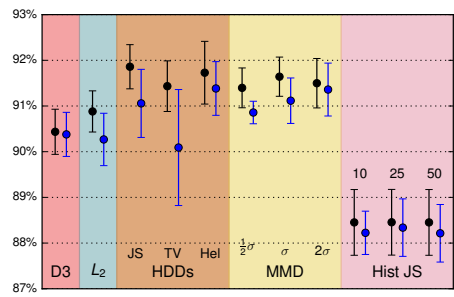


Figure 4.3: Mean and standard deviation of accuracies on the Scene-15 dataset in 10 random splits. The left, black lines use linear projection coefficient type features; the right, blue lines show random features.

# Chapter 5

## Conclusion

In conclusion, Part I presents new estimators, the Double Basis (2BE) and Triple Basis (3BE) estimators, for performing distribution to real regression and distribution to distribution regression, respectively. In particular, these estimators scale independently of  $N$  (the number input sample-set/response pairs) in a large dataset for performing evaluations for response predictions. This is a great improvement over the linear scaling with  $N$  that linear smoother estimators have and allows one to explore DRR and DDR in new domains with large collections of distributions, such as astronomy and finance. Also, we empirically showed the improved scaling of the 2BE and 3BE, as well improvements in risk over linear smoothing approaches. It is worth noting that while the 2BE and 3BE regress a mapping in a nonlinear space (induced by the random features), the linear smoothing approaches output only a weighted average of training set responses. Moreover, in Chapter 4 we introduce random features to approximate kernels that use a broad range of metrics over distributions. Although these features come at an added computational expense, they were shown to increase performance over Euclidean based random features in various tasks. As a rule of thumb, however, the  $L_2$  based random features originally used in the 2BE and 3BE estimators often perform well enough to be used as a baseline benchmark across tasks.

### 5.1 Discussion

We discuss several nuanced points below.

First, it is worth noting that we had chosen nonparametric orthonormal bases a priori in the above application of the 2BE and 3BE. Particularly, we made extensive use of the cosine basis (Section 2.1). Since the number of basis functions needed to estimate pdfs with finite samples depends on the smoothness w.r.t. chosen basis, the cosine basis might require many projection coefficients to represent input/output pdfs. Thus, while such bases may be nonparametric and able to represent arbitrary functions in  $L_2$  asymptotically, they may be far from efficient for the sorts of input/output pdfs present in a dataset. The use of many projection coefficients will be a computational hindrance, as we need to compute and store the projections. Worst still, the use of many projection coefficients will make for a harder learning problem, as we will need to estimate more responses and higher dimensional

kernels. Perhaps one approach to address this issue is to use functional PCA (FPCA) [104], which essentially performs PCA over the  $A : N \times s$  matrix of  $s$  projection coefficients for the  $N$  input (or output) functions. FPCA would then yield  $\{\psi_i = \sum_{\alpha} v_{\alpha}^{(i)} \varphi_{\alpha}\}_{i=1}^k$  ( $k \ll s$ ) new basis functions, comprised of linear combinations of the original basis functions,  $\varphi_{\alpha}$  (e.g. the cosine basis functions). However, FPCA is only a partial solution. While FPCA may help the estimation burden by lessening the final number of basis functions used to represent input/output pdfs, it does little to alleviate the computational burden. This is because to compute the new basis functions  $\psi_i$  one must still compute the  $s$  original basis functions. Moreover, one must perform an eigendecomposition of the large product  $A^T A$ , which will be expensive since both  $N$ , the number of instances, and  $s$  the number of basis functions will be large for big datasets with even modest domains  $d \gtrsim 3$ . In addition, a straightforward approach would also necessitate computing all the elements in  $A$ ; i.e. computing the projection coefficients for all the basis functions and input (or output) pdfs. It remains to be seen how much FPCA type approaches can aid estimation, or how to adjust these approaches to be computationally efficient.

One may also consider supervised, end-to-end methods for selecting the bases to use for input/output functions in addition to unsupervised approaches such as FPCA. That is, instead of finding a basis based only on the functions, we also consider the final error of DRR or DDR tasks. Multitask learning (MTL) [145, 134], is one direction that may be fruitful for this. Currently, the 3BE solves for the projection coefficients of the output function independently; however, an MTL approach may help to exploit correlations for better estimation. It is also worth noting that there is a lack of a true  $L_2$  error in real-world data. Although subtle, it bears reminding that when one computes  $\|\tilde{q}_i - \hat{f}(\tilde{p}_i)\|$  one is computing the error w.r.t. an estimate of the output function  $q_i$  and not  $q_i$  itself since it is unknown in real-world data. While this is perhaps a bit unsettling at first glance, it is akin to the typical case where one only has access to noisy responses  $Y_i = f(X_i) + \epsilon_i$ . Of course, here one’s “noise” stems from the number of samples observed for  $q_i$ , the smoothness, etc.

Finally, it is interesting to note that several neural network based approaches to deal with distributional covariates have been recently developed. For instance, [141, 96] considers permutation invariant architectures, and [106] makes use of  $3d$  convolutions of histograms. Some of these approaches may provide better scaling to input domain dimensionality due to the data-driven nature of neural network features. However, a thorough comparison of nonparametric methods such as the 2BE and neural network approaches is lacking in the literature and would be of interest. Furthermore, it remains to be seen if a superior amalgamation of both approaches exists, or how to best use neural network methods for distributional responses in DDR tasks.



## Part II

# Learning with Distributions

Next, we continue to expand the use of distributions in machine learning. In this part we now consider aiding traditional machine learning tasks with both implicit and explicit distributions. First, we learn an implicit spectral distribution to perform scalable kernel learning over inputs. In Chapter 6 a Bayesian nonparametric prior is used to learn the spectral distribution of a kernel. Second, we consider the distribution of previously seen points to perform sequential modeling. In Chapter 7 we modify the traditional use of summary statistics to be more amenable to sequential modeling with the statistical recurrent unit. Lastly, we explore density estimation itself, and develop flexible models for high dimensional data. In Chapter 8, we make use of transformations of variables and autoregressive conditionals to model data.



# Chapter 6

## Bayesian Spectral Distributions

We now transition from the study of treating distributions and sample sets as inputs and/or outputs to the study of an implicit distribution for the improved performance of regression and classification estimators on typical vector valued data. Below we shall learn the implicit spectral distribution, which defines a kernel and function class, to perform kernel learning for classification and regression. As we shall see, through the use of a Bayesian nonparametric prior, Bayesian Nonparametric Kernel-Learning (BaNK) [90] will be able to adjust the kernel used in general learning tasks and achieve better performance.

### 6.1 Introduction

Kernel methods such as support vector machines (SVMs), kernel-ridge regression, kernel-PCA, and Gaussian processes (GPs) have become the cornerstone of many machine learning approaches. However, the choice of kernel, which profoundly affects performance, has until recently received little attention.

In fact, finite sample estimates will be affected by the kernel choice notwithstanding the use of an universal approximating kernel. Indeed, the a priori choice of a fixed kernel in kernel methods is typically ad-hoc and not data-driven. Even when learning kernel hyperparameters, one is typically limited to an arbitrarily chosen and restrictive family of kernel functions, exploring only a very small subset of reasonable possibilities. Given that the choice of kernel is an important free parameter in kernel methods, and generally there are few a priori reasons for kernel selections, a principled and data-driven method for learning kernels is extremely useful.

Furthermore, as previously discussed, kernel methods often do not scale to datasets with a large number of instances due to the need to compute and store an  $N \times N$  Gram matrix,  $\mathbf{K}$ , for  $N$  training points. Moreover, kernel methods, such as GPs, will often require manipulations of  $\mathbf{K}$  like solving linear systems and computing log determinants, leading to a  $O(N^3)$  time complexity. Considering that modern datasets are only increasing in size, and complicated machine learning tasks require large datasets, it is vital to mitigate the high computational cost of kernel methods.

This is because kernel methods typically require the computation of a large Gram matrix

of kernel evaluations for all pairs of instances in a dataset. That is, if one is optimizing over a dataset of  $N$  instances using a kernel  $K$ , then one will need to compute a Gram matrix  $\mathbf{K} \in \mathbb{R}^{N \times N}$ , where  $\mathbf{K}_{ij} = K(x_i, x_j)$  and  $x_i$ 's are input covariates. When  $N$  is in the many thousands or more the computation of  $\mathbf{K}$  will be prohibitive. Furthermore, kernel methods will often require manipulations of  $\mathbf{K}$  such as taking inverses, which will result in a worse time complexity than the  $O(N^2)$  time required for computing  $\mathbf{K}$ . Considering that modern datasets are only increasing in size, and complicated machine learning tasks require large datasets for achieving a low risk, it is vital to mitigate the high computational cost of kernel methods.

In order to provide a method that scales to large datasets and adaptively learns the kernel to use in a data-driven fashion, we present the *Bayesian nonparametric kernel-learning* (BaNK) framework [90]. BaNK learns a latent spectral distribution of frequencies that are used in random features to both provide a scalable solution and learn kernels. This approach scales through random features and places a Bayesian nonparametric distribution over kernels, with support for any stationary kernels.

Random features have been recently shown to be an effective way to scale kernel methods to large datasets. Roughly speaking, random feature techniques like random kitchen sinks (RKS) [102] work as follows. Given a shift invariant kernel  $K(x, x') = k(x - x')$ , one constructs an approximate primal space to estimate kernel evaluations  $K(x, x')$  as the dot product of finite vectors  $z(x)^T z(x')$ . The vectors  $z$  are constructed with random frequencies drawn from a distribution  $\mathcal{D}$  that is defined by  $K$ . Similarly, a distribution  $\mathcal{D}$  from which random frequencies are drawn from defines a kernel  $K$  that the random frequencies approximate. It is this last observation that is key for the BaNK framework.

BaNK will allow the distribution  $\mathcal{D}$  to vary with the given data, effectively learning the kernel. In particular, BaNK shall vary  $\mathcal{D}$  with a graphical model approach where we treat  $\mathcal{D}$  as a latent parameter and place a prior on it (Figure 6.1). The prior on  $\mathcal{D}$ , along with the data generation model, will allow one to sample from a posterior over  $\mathcal{D}$  in order to learn the corresponding kernel.

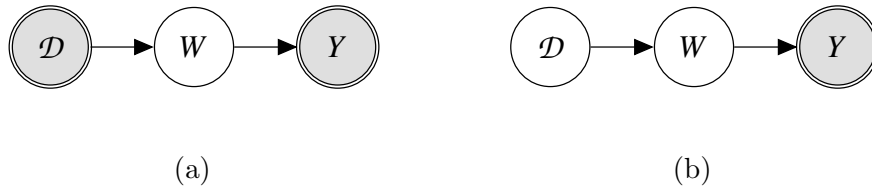


Figure 6.1: (a) Traditional random feature approach where the distribution  $\mathcal{D}$  of random features  $W$  is held fixed. (b) BaNK framework where  $\mathcal{D}$  is random.

We model  $\mathcal{D}$  as a mixture of Gaussians with a Dirichlet process prior, which allows BaNK to learn a kernel from a rich, broad class. Furthermore, with the use of random features, we are able to efficiently sample the model parameters and work over larger datasets. Moreover, by using Metropolis-Hastings we sample from a proper posterior, thus the kernels we learn are interpretable since the random features are asymptotically guaranteed to come from

the underlying posterior distribution unlike greedy non-convex optimization methods.

The rest of this chapter is structured as follows. First we review the use of random features for kernel approximation and show how such an approach can be used for flexible and efficient kernel learning. Second, we detail our graphical model framework both for supervised regression and classification tasks. Third, we expound on our inference method for sampling from the model posterior. Forth, we illustrate the use and performance of BaNK for both regression and classification on several datasets. Lastly, we cover related works and give concluding remarks.

## 6.2 Model

### 6.2.1 Random Features for Kernel Estimation

Below we briefly review the method of random Fourier features for the approximation of kernels [102]. The details of the method will help motivate and explain our BaNK model. Henceforth, we will only consider continuous shift-invariant kernels defined over  $\mathbb{R}^d$ :  $K(x, y) = k(x - y)$  where  $x, y \in \mathbb{R}^d$  and  $k$  is a positive definite function. The use of random Fourier features for kernel approximation is a result of Monte Carlo integration using Bochner’s theorem [107]. Bochner’s theorem states that a continuous shift-invariant kernel  $K(x, y) = k(x - y)$  is a positive definite function if and only if  $k(t)$  is the Fourier transform of a non-negative measure  $\rho(\omega)$ . Note further, that if  $k(0) = 1$ , then  $\rho(\omega)$  will be a normalized density. That is, if we define  $\zeta_\omega(x) \equiv \exp(i\omega^T x)$ , then

$$k(x - y) = \int_{\mathbb{R}^d} \rho(\omega) \exp(i\omega^T(x - y)) d\omega = \mathbb{E}_{\omega \sim \rho}[\zeta_\omega(x)\zeta_\omega(y)^*]. \quad (6.1)$$

Hence, using Monte Carlo integration, we can approximate  $K(x, y) = k(x - y)$  using  $\omega_j \stackrel{iid}{\sim} \rho$ :

$$k(x - y) \approx \frac{1}{M} \sum_{j=1}^M \zeta_{\omega_j}(x)\zeta_{\omega_j}(y)^*. \quad (6.2)$$

In particular, if our kernel  $k$  is real-valued, then we can discard the imaginary part of (6.2):

$$k(x - y) \approx z(x)^T z(y), \quad (6.3)$$

where

$$z(x) \equiv \frac{1}{\sqrt{M}}[\cos(\omega_1^T x), \dots, \cos(\omega_M^T x), \sin(\omega_1^T x), \dots, \sin(\omega_M^T x)]^T. \quad (6.4)$$

The great advantage of such an approximation is that we may now estimate a function in the RKHS as a linear operator in the random features:

$$f(x) = \sum_{i=1}^m \alpha_i K(x_i, x) \approx \sum_{i=1}^m \alpha_i z(x_i)^T z(x) = \psi^T z(x), \quad (6.5)$$

where  $\psi \equiv \sum_{i=1}^m \alpha_i z(x_i)$ . Thus we may work directly in a primal space of  $z(x)$  and avoid computing large Gram matrices. To recap, using the approximation of kernels with random features works as follows: choose a kernel defined by  $k$  (with  $k(0) = 1$ ), take its Fourier transform,  $p(\omega)$ , which will be a pdf over  $\mathbb{R}^d$ ; draw  $M$  i.i.d. samples from  $\rho(\omega)$ ,  $\{\omega_j\}_{j=1}^M$ ; estimate the kernel with  $K(x, y) \approx z(x)^T z(y)$  as in (6.4).

However, Bochner’s theorem also allows one to work in the other direction. That is, we may start with a distribution  $\mathcal{D}$  with pdf  $\rho(\omega)$  and take the characteristic function (the inverse Fourier transformation) to define a shift-invariant kernel  $k$ . For example, suppose that  $\rho(\omega) = \mathcal{N}(\omega|\mu, \Sigma)$ , where  $\mathcal{N}(\omega|\mu, \Sigma)$  is the pdf of  $\mathcal{N}(\mu, \Sigma)$ . Taking its characteristic function we see that  $k(t) = \exp(i\mu^T t - \frac{1}{2}t^T \Sigma t)$  would be the corresponding shift-invariant kernel. From the kernel learning perspective, Bochner’s theorem yields an object to manipulate for the learning of one’s kernel:  $\rho(\omega)$  the distribution of random features.

We consider distributions that are mixtures of Gaussians:

$$\rho(\omega) = \sum_{\ell=1}^L \pi_\ell \mathcal{N}(\omega|\mu_\ell, \Sigma_\ell) \rightarrow k(t) = \sum_{\ell=1}^L \pi_\ell \exp(i\mu_\ell^T t - \frac{1}{2}t^T \Sigma_\ell t). \quad (6.6)$$

This makes for very general kernels; for a discussion on general properties of these kernels for finite  $L$  please see [137]. In fact, i) noting that Gaussian mixture models are universal approximators of densities and may hence approximate any spectral distribution, and ii) using Plancherel’s Theorem to relate spectral accuracies to the original domain [112, 140] it follows that:

**Proposition 6.2.1.** *The expression of  $\rho(\omega)$  in (6.6) can approximate any shift invariant kernel.*

For our applications we only need real-valued kernels, hence we use the real part of (6.6):

$$K(x, y) = \sum_{\ell=1}^L \pi_\ell \exp(-\frac{1}{2}(x - y)^T \Sigma_\ell (x - y)) \cos(\mu_\ell^T (x - y)) \quad (6.7)$$

$$\approx z(x)^T z(y), \quad (6.8)$$

where  $z(x)$  is as in (6.4). An application of the random feature approximation bounds found in [sutherland2015error, 102, 66] yields that:

**Proposition 6.2.2.** *For compact  $\mathcal{X} \subset \mathbb{R}^d$  with finite diameter, we have that*

$$\Pr \left[ \sup_{x, y \in \mathcal{X}} |K(x, y) - z(x)^T z(y)| \geq \epsilon \right] = O \left( \frac{1}{\epsilon^2} \exp \left( \frac{-M\epsilon^2}{4(d+2)} \right) \right).$$

Using the above, it may be seen that one can effectively approximate shift-invariant kernels using random features drawn from Gaussian mixtures. However, in order to learn the kernel, one still needs a mechanism to determine the Gaussian mixture to use. We take a graphical model approach to determine the mixture for  $\rho(\omega)$  in a principled, data-driven fashion. The concept of using nonparametric mixture prior on  $\rho$  with random frequencies was considered in [138] without empirical details.

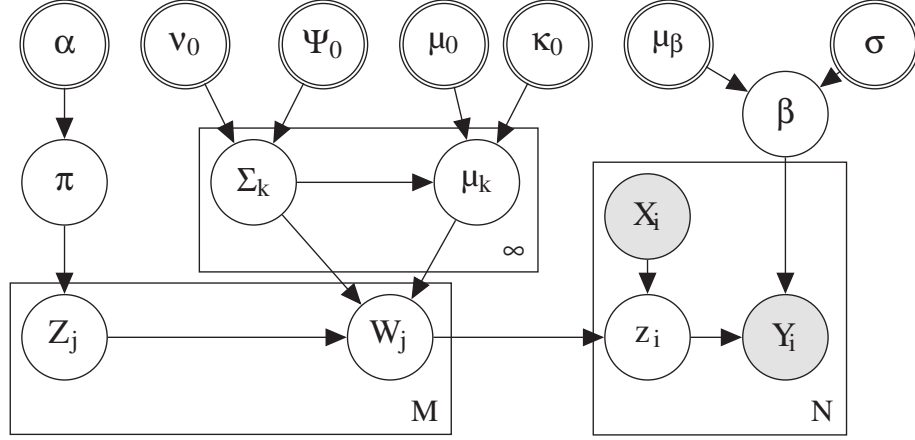


Figure 6.2: Plate diagram for the graphical model for BaNK learning framework.

## 6.2.2 Graphical Model

As described above, one may vary and tune kernels with the choice of density over random features,  $\rho(\omega)$ . Thus, in our model, we take this distribution itself to be a random latent parameter, in effect placing a prior over all stationary kernels, resulting in a strictly more general method than the traditional approach of using a fixed RBF kernel.

Roughly speaking, the BaNK model will consist of three major parts: one, a prior for stochastically generating the random feature distribution  $\rho(\omega)$ ; two, a prior for the generation of the parameters of a linear model in the primal space of random features; three, a generative data model with noise to generate labels given input covariates and the rest of the parameters.

First, the spectral distribution  $\rho(\omega)$  is generated. As previously mentioned, a robust and flexible choice of  $\rho(\omega)$  is a Gaussian mixture model; Since the number of modes of  $\rho(\omega)$  is not a priori known, we will assume it to be infinite ( $\rho(\omega) = \sum_{k=1}^{\infty} \pi_k \mathcal{N}(\omega | \mu_k, \Sigma_k)$  where  $\sum_k \pi_k = 1$  and  $\pi_k > 0$ ), but given a finite dataset the model will realize only a finite number of Gaussians in the mixture. We use a Dirichlet process (DP) prior on the components of the Gaussian mixture ( $\pi$ ).

The Dirichlet process is a distribution over discrete probability measures (i.e., atoms),  $G = \sum_{k=1}^{\infty} \pi_k \delta_{\pi_k}$ , with countably infinite support, where the finite-dimensional marginals are distributed according to a finite Dirichlet distribution [29]. We sample the mixture weights from a stick breaking prior, i.e.  $\pi \sim GEM(\alpha)$  where  $GEM$  is the stick breaking prior [111]. We also put a Normal-Inverse-Wishart prior on the mean  $\mu_k$  and variance  $\Sigma_k$  of each of the Gaussian components.

Secondly, model parameters are generated. In Section 6.2.1 we discussed how functions in a kernel's RKHS can be approximated using a linear mapping in the random features. Thus, we consider models that operate linearly in the random features using a vector  $\beta \in \mathbb{R}^{2M}$ . As is standard in Bayesian regression and classification models [11], we generate  $\beta$  from a Normal prior,  $\beta \sim \mathcal{N}(\mu_\beta, \sigma I)$ .

Lastly, our observations are generated given a dataset  $X := (x_1, \dots, x_N)^T$  where each

$x_i \in \mathbb{R}^d$ . For example in regression tasks we have:

$$y = g(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon I), \quad (6.9)$$

where  $g$  is  $\beta^T z(x)$  and  $z(x)$  is calculated using (6.4). Thus  $y \sim \mathcal{N}(\beta^T z(x), \sigma_\epsilon)$ .

The complete generative model is given below and the corresponding plate diagram is shown in Figure 6.2.

1. Draw the mixture weights  $\pi$  over components of the kernel:  $\pi \sim GEM(\alpha)$ .
2. Draw the mixture components from a Normal-Inverse-Wishart distribution. I.e. draw  $\Sigma_k \sim \mathcal{W}^{-1}(\Psi_0, \nu_0)$ , and  $\mu_k \sim \mathcal{N}(\mu_0, \frac{1}{\kappa_0} \Sigma_k)$  for  $k = 1, \dots, \infty$ .
3. For each random frequency index  $j = 1, \dots, M$ 
  - (a) Draw the component from which the frequency vector is drawn.  $Z_j \sim Mult(\pi)$ .
  - (b) Draw the corresponding random frequency vector  $W_j \sim \mathcal{N}(\omega | \mu_{Z_j}, \Sigma_{Z_j})$ .
4. Draw the weight vector,  $\beta \sim \mathcal{N}(\mu_\beta, \sigma I)$ .
5. For each data point index  $i = 1, \dots, N$ 
  - (a) Define  $z(X_i)$  as in (6.4).
  - (b) Draw the observation, e.g. for regression:  $Y_i \sim \mathcal{N}(z(X_i)^T \beta, \sigma_\epsilon I)$ .

We note that the only change when going from regression to classification is in the step 5(b) of the generative procedure. This time we draw  $Y_i$  from a sigmoid.

- 5 For each data point index  $i = 1, \dots, N$ 
  - (b) Draw the output binary label  $Y_i \sim \sigma(z(X_i)^T \beta)$ , where  $\sigma(x) = \frac{1}{1 + \exp(-x)}$ .

## 6.3 Inference

We propose a MCMC based solution for inferring the parameters of the mixture of Gaussian distribution that defines  $\rho(\omega)$ . This includes finding the component assignment vector  $Z$  and the mean and covariance  $\mu_k$  and  $\Sigma_k$  for each component. We will also sample the random frequencies  $W$  while marginalizing other parameters including  $\pi$  and  $\beta$  whenever possible. We will first describe the sampling equations for  $Z$ ,  $\mu_k$ ,  $\Sigma_k$ , which remain the same for both regression and classification. Afterwards we describe inference for  $W$  which depends on the specific application.

We want to sample from  $p(Z, \mu, \Sigma, W | X, Y, \text{rest})$ , where rest are all the hyper-parameter of our model while other parameters including  $\beta$  and  $\pi$  have been integrated out. We use Gibbs sampling and sample each variable at a time given all other variables.

### 6.3.1 Sampling $Z_j$

Recall that  $Z_j$  indicates which component the random frequency  $W_j$  is drawn from. We use the Chinese restaurant process analogy to integrate out  $\pi$ , the component priors. Let



$m_k \equiv \sum_l \delta(Z_l = k)$ . The sampling equation for  $Z_j$  can be derived from [85] and is shown below

$$P(Z_j = k | \mu, \Sigma, W, X, Y, \text{rest}) = \begin{cases} \frac{m_k^{-j}}{M-1+\alpha} \mathcal{N}(\omega_j | \mu_k, \Sigma_k) & m_k^{-j} > 0 \\ \frac{\alpha}{M-1+\alpha} \int_{\mu, \Sigma} \mathcal{N}(\omega_j | \mu, \Sigma) NIW(\mu, \Sigma) d\mu d\Sigma & m_k^{-j} = 0 \end{cases} \quad (6.10)$$

where  $m_k^{-j} = \sum_{l:l \neq j} \delta(Z_l = k)$ ,  $W_j = \omega_j$ ,  $m_k^{-j} = 0$  corresponds to unseen mixture component and  $NIW$  is the Normal-Inverse-Wishart prior on mean and variance.

### 6.3.2 Sampling $\mu_k$ and $\Sigma_k$

Given the component assignment  $Z$  and the random frequencies  $W$ , the posterior distribution of the covariance of each Gaussian component in the mixture is Inverse-Wishart, ie  $\Sigma_k \sim \mathcal{W}^{-1}(\Psi_k, \nu_k)$  where  $\Psi_k = \Psi_0 + \sum_{j:Z_j=k}^M (W_j - \bar{W}^k)(W_j - \bar{W}^k)^T + \frac{\kappa_0 m_k}{\kappa_0 + m_k} (\bar{W}^k - \mu_0)(\bar{W}^k - \mu_0)^T$ , where  $\bar{W}^k = \frac{1}{m_k} \sum_{j:Z_j=k} W_j$  and  $\nu_k = \nu_0 + m_k$ . Similarly, the posterior distribution of  $\mu_k$  given,  $\Sigma_k$ ,  $Z$  and  $W$  is a normal; i.e.  $\mu_k \sim \mathcal{N}(\mu_k, \frac{1}{\kappa_k} \Sigma_k)$ , where  $\mu_k = \frac{\kappa_0 \mu_0 + m_k \bar{W}^k}{\kappa_0 + m_k}$  and  $\kappa_k = \kappa_0 + m_k$ . See [36] for details.

### 6.3.3 Sampling $W$

We derive a Metropolis-Hasting (MH) sampler for sampling  $W$ . The posterior distribution of the random frequencies  $W$  given the assignment  $Z$ , the parameters of the component  $\mu$  and  $\Sigma$ , and the data,  $X$  and  $Y$  is proportional to

$$P(W|Z, \mu, \Sigma, Y, X, \text{rest}) \propto P(W|Z, \mu, \Sigma) P(Y|X, W, \text{rest}). \quad (6.11)$$

The first term in the LHS is a normal distribution  $P(W|Z, \mu, \Sigma) = \prod_j \mathcal{N}(W_j | \mu_{Z_j}, \Sigma_{Z_j})$ . Since it is difficult to sample directly from the posterior, we use MH, where the first factor of the RHS of (6.11) is used as a proposal distribution; i.e.  $Q(W) = P(W|Z, \mu, \Sigma)$ . Now, the acceptance ratio for a newly proposed  $W^*$  is given by

$$r = \min \left\{ 1, \frac{P(Y|X, W^*, \text{rest})}{P(Y|X, W, \text{rest})} \right\}. \quad (6.12)$$

Here the second term on RHS of (6.12) is a ratio of model evidences and is calculated differently for regression and classification.

### Regression

For regression we make use for conjugacy between prior of  $\beta$ ,  $\sigma_\epsilon$  and the likelihood to get a closed form solution for  $P(Y|X, W, \text{rest})$ . In this case we sample  $\sigma_\epsilon$  from Inverse – Gamma( $a_0, b_0$ ). The model evidence is then:

$$P(Y|X, W, \text{rest}) = \int P(Y|W, X, \beta, \sigma_\epsilon) P(\beta) P(\sigma_\epsilon) d\beta d\sigma_\epsilon \propto \frac{\Gamma(a_n) b_0^{a_0}}{\Gamma(a_0) b_n^{a_n}} \sqrt{\frac{|\Lambda_0|}{|\Lambda_n|}}, \quad (6.13)$$

where  $\Lambda_0 = \frac{1}{\sigma^2}I$ ,  $\Phi(X) = (z(X_1)^T \dots z(X_N)^T)^T$ ,  $\Lambda_n = \Phi(X)^T \Phi(X) + \Lambda_0$ ,  $\mu_n = \Lambda_n^{-1}(\Lambda_0 \mu_\beta + \Phi(X)^T Y)$ ,  $a_n = a_0 + \frac{n}{2}$  and  $b_n = b_0 + \frac{1}{2}(Y^T Y + \mu_0^T \Lambda_0 \mu_0 - \mu_n^T \Lambda_n \mu_n)$ . For more details refer to [76].

It is worth noting that one may efficiently compute ratios of model evidences if proposing a single  $W_j$  at a time. That is, for each  $j \in \{1, \dots, M\}$  we propose  $W_j^* \sim \mathcal{N}(W_j | \mu_{Z_j}, \Sigma_{Z_j})$  and calculate an acceptance ratio of

$$r_j = \min \left\{ 1, \frac{P(Y|X, W_j^*, W_{-j}, \text{rest})}{P(Y|X, W_j, W_{-j}, \text{rest})} \right\} \quad (6.14)$$

where  $W_{-j} = \{W_\ell\}_{\ell \neq j}$ . This can be done efficiently because computing  $P(Y|X, W_j^*, W_{-j}, \text{rest})$  only requires low-rank updates on  $\Phi(X)^T \Phi(X)$ , allowing for fast Cholesky updates.

## Classification

The aforementioned inference algorithm requires one to analytically obtain the model evidence of the data in terms of the model's random frequencies (eq: 6.12, 6.14). However, a lack of conjugacy may make it intractable to marginalize other parameters to obtain the model evidence. For instance, the Gaussian prior on  $\beta$  is not conjugate to a sigmoid. As a result it is difficult to directly compute the model evidence for a logistic regression model ie  $P(Y|X, W, \text{rest})$  where  $\beta$  has been integrated out. Thus, for such situations where marginalization is intractable we must take a different approach to computing acceptance ratios for accepting random frequencies.

An approach one may take is to use a Laplace approximation to estimate the evidence as

$$\begin{aligned} & \log(p(Y|X, W, \text{rest})) \\ & \approx \log(p(Y|X, W, \beta_{\text{MAP}}, \text{rest})) + \log(p(\beta_{\text{MAP}})) + \frac{N}{2} \log(2\pi) - \frac{1}{2} \log(|A|) \end{aligned} \quad (6.15)$$

$$\text{where } \beta_{\text{MAP}} = \arg \min_{\beta} \log(P(Y|X, W, \beta, \text{rest})P(\beta)) \quad (6.16)$$

$$\text{and } A = -\nabla^2 \log(P(Y|X, W, \beta, \text{rest})P(\beta))|_{\beta=\beta_{\text{MAP}}}. \quad (6.17)$$

However, there are a few drawbacks to using a Laplace approximation in this manner. First, due to approximation, one is no longer sampling from the true posterior. Second, calculating  $\beta_{\text{MAP}}$  when a closed form solution is not available (as with logistic regression) requires solving a costly  $2M$  dimensional optimization problem when computing acceptance ratios.

In order to address these drawbacks whilst still mixing well we jointly sample the  $j$ th random frequency  $W_j$  and the weight vector  $\beta_j^{\cos}$  and  $\beta_j^{\sin}$  corresponding to features  $\cos(W_j^T x)$  and  $\sin(W_j^T x)$  respectively. Specifically we sample from the joint distribution  $W_j$  and  $\beta_j^\bullet = \{\beta_j^{\cos}, \beta_j^{\sin}\}$  given by:

$$P(W_j, \beta_j^\bullet | Z, \mu, \Sigma, Y, X, \text{rest}) \propto P(W_j | Z, \mu, \Sigma) P(\beta_j^\bullet | \text{rest}) P(Y | X, W_j, \beta_j^\bullet). \quad (6.18)$$

However, samples from (6.18) are not readily available, so we use Metropolis Hastings with the proposal distribution being

$$Q = P(W_j|Z_j, \mu, \Sigma)\text{Lap}(\beta_j^\bullet|X, Y, W_j, \text{rest}) \quad (6.19)$$

where  $\text{Lap}(\beta_j^\bullet|X, Y, W_j, \text{rest})$  is the Laplace approximation of the posterior of  $\beta_j^\bullet$ , which requires only a 2-dimensional optimization:

$$\text{Lap}(\beta_j^\bullet|X, Y, W_j, \text{rest}) \approx P(\beta_j^\bullet|\text{rest})P(Y|X, W_j, \beta_j^\bullet). \quad (6.20)$$

Hence, acceptance ratio for jointly sampling a new  $\{W_j^*, \beta_j^{\bullet*}\}$  can be calculated as

$$\min \left\{ 1, \frac{P(Y|X, W_j^*, \beta_j^{\bullet*}, \text{rest})P(\beta_j^{\bullet*})\text{Lap}(\beta_j^\bullet|X, Y, W_j)}{P(Y|X, W_j, \beta_j^\bullet, \text{rest})P(\beta_j^\bullet)\text{Lap}(\beta_j^{\bullet*}|X, Y, W_j^*)} \right\}. \quad (6.21)$$

### 6.3.4 Runtime Complexity

We expound on the runtime complexity per iteration for the inference algorithms detailed above. Suppose that the  $L$  is the number of components considered,  $d$  is the data dimension,  $M$  is the number of frequencies and  $N$  is the number of data points. The runtime per iteration for sampling component parameters for both regression and classification is as follows: 1) sampling component parameters  $\mu_\ell$ 's and  $\Sigma_\ell$ 's (and maintaining stats):  $O(Ld^3)$ ; 2) sampling component assignments  $Z_j$ :  $O(MLd^2)$ .

For regression, sampling the random frequencies  $W$  using low rank update takes  $O(M(d^2 + dN + MN + M^2))$ . Thus, the total runtime per iteration is  $O(M^2d^2 + M^2N) = O(M^2N)$  for large datasets where  $N \gg M > d$ , and  $M \geq L$ .

For classification, sampling  $W_j$ 's is  $O(MN(d + \epsilon^{-2}))$ ; where the  $\epsilon^{-2}$  term arises from performing the 2-dimensional optimization required in (6.20) to  $\epsilon$  precision [17]. Treating  $\epsilon$  as a constant, we have a total runtime of  $O(MNd)$  for inference.

Hence, we see that inference is linear in  $N$  in either case, and so our method allows one to perform kernel learning in large datasets.

## 6.4 Experiments

We illustrate the use and performance of BaNK for both regression and classification on synthetic and real-world datasets below.

### 6.4.1 Synthetic Data

We give a simple 1-d kernel learning illustration with BaNK using synthetic data. We consider the shift-invariant kernel  $k(t) = \exp\left(-\frac{1}{2(2^2)}t^2\right)\left(\frac{1}{2} + \frac{1}{2}\cos\left(\frac{3}{4}\pi t\right)\right)$ ; that is, the kernel whose random frequency distribution is  $\rho(\omega) = \frac{1}{2}\mathcal{N}(\omega|0, \frac{1}{2^2}) + \frac{1}{2}\mathcal{N}(\omega|\frac{3}{4}\pi, \frac{1}{2^2})$  (see Figure 6.3). We look to learn the underlying kernel using 250 frequencies. We generated  $N = 1000$  instances  $D = \{X_i, Y_i\}_{i=1}^N$  where  $X_i \stackrel{iid}{\sim} \mathcal{N}(0, 4^2)$ ,  $Y_i \sim \mathcal{N}(z_\rho(X_i)^T\beta, 1)$ , with  $z_\rho$  being the

random features from the kernel’s true spectral distribution  $\omega_j \stackrel{iid}{\sim} \rho$  and  $\beta \sim \mathcal{N}(0, I)$ . As explained above, using BaNK one may estimate  $\rho$  by drawing from the posterior. We plot one such draw in Figure 6.3(b). One can see that BaNK approximates the kernel rather well even though the underlying spectral distribution is multi-modal, and the kernel is not easily decernable to the human eye based on the data plot (Figure 6.3(a)).

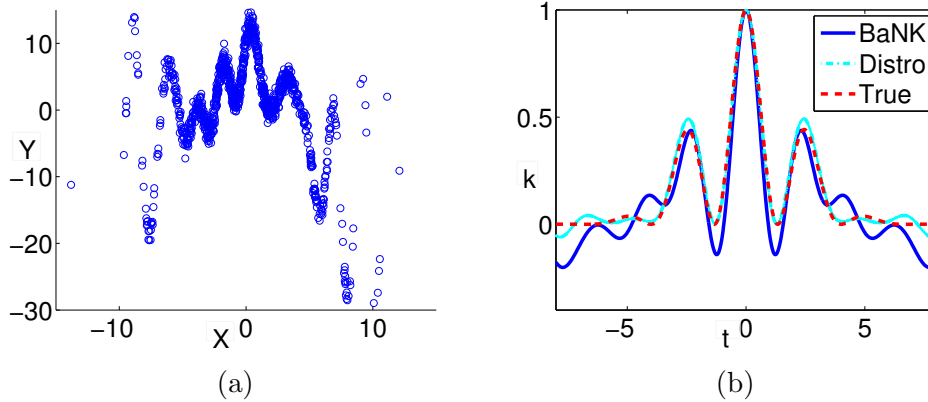


Figure 6.3: (a) Synthetic dataset used for regression. (b) True kernel,  $k$ , in dashed red,  $k$  estimated with true spectral distribution  $\rho$  in cyan, and BaNK estimate in blue.

Dataset	$N$	$d$	RKS	MKL	AlaC	BaNK
concrete	1030	8	0.1313 $\pm$ 0.0189	0.0942 $\pm$ 0.0100	<b>0.0682 <math>\pm</math> 0.0092*</b>	0.1195 $\pm$ 0.0108
noise	1503	5	0.6974 $\pm$ 0.0244	<b>0.3217 <math>\pm</math> 0.0256*</b>	<b>0.3395 <math>\pm</math> 0.0489</b>	<b>0.3359 <math>\pm</math> 0.0354</b>
prop	11934	16	0.0006 $\pm$ 3.6 $\times 10^{-6}$	4.2 $\times 10^{-5}$ $\pm$ 5.7 $\times 10^{-6}$	0.0003 $\pm$ 0.0002	<b>8.9 <math>\times 10^{-6}</math> <math>\pm</math> 1.2 <math>\times 10^{-6}</math>*</b>
bike	17379	12	0.1832 $\pm$ 0.0049	0.1509 $\pm$ 0.0057	<b>0.0467 <math>\pm</math> 0.0016*</b>	<b>0.0496 <math>\pm</math> 0.0022</b>
tom’s	28179	96	0.0479 $\pm$ 0.0109	0.0891 $\pm$ 0.0100	0.6583 $\pm$ 0.0413	<b>0.0083 <math>\pm</math> 0.0018*</b>
cte	53500	386	0.0886 $\pm$ 0.0014	0.0442 $\pm$ 0.0003	0.6223 $\pm$ 0.0053	<b>0.0101 <math>\pm</math> 0.0003*</b>
music	515345	90	0.8333 $\pm$ 0.0028	0.8524 $\pm$ 0.0029	0.7318 $\pm$ 0.0705	<b>0.7042 <math>\pm</math> 0.0038*</b>
twitter	583250	77	0.3837 $\pm$ 0.0397	0.4572 $\pm$ 0.0175	0.2223 $\pm$ 0.0358	<b>0.0981 <math>\pm</math> 0.0211*</b>

Table 6.1: Regression MSE on UCI MLR. Asterisks denote the lowest MSE per dataset, methods in bold text were not found to be statistically different from the lowest MSE using a paired t-test with  $p$ -value  $<$  0.05.

Dataset	$N$	$d$	RKS	MKL	RFO	BaNK
pima	768	8	0.332 $\pm$ 0.0201	0.4455 $\pm$ 0.0533	<b>0.2592 <math>\pm</math> 0.0214*</b>	<b>0.263 <math>\pm</math> 0.0111</b>
diabetic	1151	20	0.3312 $\pm$ 0.0083	0.3051 $\pm$ 0.0004	0.4263 $\pm$ 0.0007	<b>0.279 <math>\pm</math> 0.0022*</b>
eeg	14980	15	0.0706 $\pm$ 0.0019	<b>0.0544 <math>\pm</math> 0.0021*</b>	0.2411 $\pm$ 0.1123	<b>0.0686 <math>\pm</math> 0.0012</b>
space	58000	9	0.0022 $\pm$ 0.0004	<b>0.0015 <math>\pm</math> 0.0001</b>	0.0018 $\pm$ 0.00007	<b>0.0009 <math>\pm</math> 0.0001*</b>
susy	100000	18	0.2009 $\pm$ 0.0002	0.201 $\pm$ 0.0001	0.2089 $\pm$ 0.00026	<b>0.2005 <math>\pm</math> 0.0001*</b>
skin	245053	3	0.1135 $\pm$ 0.113	0.2737 $\pm$ 0.105	0.0043 $\pm$ 0.001	<b>0.0004 <math>\pm</math> 0.0001*</b>

Table 6.2: Classification Prediction Error on UCI MLR. Asterisks denote the lowest error per dataset, while the methods in bold were not found to be statistically different from the lowest error using a McNemar’s test [28] with  $p$ -value  $<$  0.05.

## 6.4.2 Regression

Below we run experiments with various real-world datasets found in the UCI machine learning repository (UCI MLR)<sup>1</sup>. We compare BaNK to a straight-forward random feature approach with a fixed kernel as well as other competitive random feature based kernel learning methods. In particular we compare to the following methods:

**RKS** For this method we take input covariates to be random features  $z(x_i)$  as in (6.4). Here we take the random frequencies  $\omega_j \stackrel{iid}{\sim} \mathcal{N}(0, \sigma^{-2}I)$ . This corresponds to approximating the RBF kernel:  $K(x_i, x_l) = \exp(-\frac{1}{2\sigma^2}\|x_i - x_l\|^2)$ . Using these random features, we regress responses with ridge regression.

**MKL** One of the most widely used approaches to kernel-learning is multiple kernel learning (MKL) [5, 62]. Here, one attempts to learn a kernel using a non-negative linear combination of a fixed bank of kernels. That is, MKL attempts to learn a kernel  $K$ :

$$K(x_i, x_l) = \sum_{m=1}^M \alpha_m K_m(x_i, x_l), \text{ where } \alpha_m \geq 0, \quad (6.22)$$

and  $K_1, \dots, K_M$  are predefined kernels. The kernel weights  $\alpha_m$  would then be optimized according to one’s loss. Note that (6.22) still requires the computation of a  $N \times N$  Gram matrix, in fact, it requires  $M$  such Gram matrices. However, we extend MKL to use random features and scale to larger datasets. If  $K_m(x_i, x_l) \approx z_m(x_i)^T z_m(x_l)$ , then

$$K(x_i, x_l) \approx \sum_{m=1}^M \alpha_m z_m(x_i)^T z_m(x_l) = \bar{z}(x_i)^T \bar{z}(x_l), \quad (6.23)$$

where  $\bar{z}(x_i) = [\sqrt{\alpha_1}z_1(x_i)^T, \dots, \sqrt{\alpha_M}z_M(x_i)^T]^T$ . Hence, it is possible to work directly over input covariates of  $\bar{z}(x_i) = [z_1(x_i)^T, \dots, z_M(x_i)^T]^T$ , the concatenation of the random features for each kernel  $K_1, \dots, K_M$ . We take our bank of kernels to be Laplace, RBF, and Cauchy kernels at various scalings. As with RKS, we regress responses through ridge regression.

**AlaC** Recently, independent work by [140] has considered an optimization approach, called A la Carte, to learning a mixture of kernels. Here, an unconstrained, unpenalized, and non-convex GP likelihood problem is posed for regression and optimized over the parameters of a mixture model for random frequencies<sup>2</sup>.

We perform 5-fold cross-validation (picking parameters on validation sets and reporting back the error on test sets). For AlaC we cross-validate the total number of mixture components and frequencies per components for datasets with fewer than 100K instances; for larger datasets we use the suggested hyper-parameters in [140]. The total number of random features was chosen to be 768 for RKS, MKL, and BaNK methods. For better interpretability, we standardized the output responses. In Table 6.1 we report the mean squared error (MSE)  $\pm$  standard errors. One may see that BaNK performs better or as well as other methods on nearly all the datasets. Furthermore, it seems like BaNK is better able to leverage larger datasets. Lastly, we note that BaNK’s sampling based approach

<sup>1</sup><https://archive.ics.uci.edu/ml/index.html>

<sup>2</sup>Optimization was done using code provided by authors of [140].

with priors on mixture components seems more robust to local minima and over-fitting and has the ability to draw more frequencies from dominant components, which explains better performance w.r.t. Alac (e.g. for tom’s dataset, Table 6.1).

### 6.4.3 Classification

As previously mentioned, we may use the BaNK framework to perform kernel learning in classification tasks. Below we illustrate the use of BaNK for classification and kernel learning on real-world datasets from the UCI MLR. We compare the accuracies BaNK models achieve to traditional scalable kernel methods for classification; namely, we consider using the aforementioned RKS and MKL random features in a logistic model.

Furthermore, we also compare to an optimization approach based on AlaC [140], which we term random frequency optimization (RFO). Although it is possible to write and differentiate a data likelihood solely in terms of spectral density parameters (through the kernel they induce) for GP regression, a lack of conjugacy with a logistic likelihood and Gaussian priors requires approximate inference for classification. Thus, directly applying the approach of [140] will be troublesome. To mitigate this difficulty, we jointly optimize a logistic loss both in terms of linear weights  $\beta$  and spectral parameters  $\{\nu, M, \mu\}$ .

Specifically, we minimize the following problem:

$$\begin{aligned}
& - \sum_{i=1}^N Y_i \left\{ \sum_{k=1}^K \nu_k^2 \sum_{j=1}^D (\beta_{kj}^{\cos} \cos(\zeta_{ijk}) + \beta_{kj}^{\sin} \sin(\zeta_{ijk})) + \beta_0 \right\} \\
& + \log \left[ 1 + \exp \left\{ \sum_{k=1}^K \nu_k^2 \sum_{j=1}^D (\beta_{kj}^{\cos} \cos(\zeta_{ijk}) + \beta_{kj}^{\sin} \sin(\zeta_{ijk})) + \beta_0 \right\} \right] \\
& + \frac{\lambda}{2} (\|\beta^{\cos}\|^2 + \|\beta^{\sin}\|^2 + \beta_0^2),
\end{aligned}$$

where  $\zeta_{ijk} = X_i^T M_k w_{kj} + X_i^T \mu_k$  and  $\beta^{\cos} \in \mathbb{R}^{KD}$ ,  $\beta^{\sin} \in \mathbb{R}^{KD}$ ,  $\beta_0 \in \mathbb{R}$ ,  $\nu \in \mathbb{R}^K$ ,  $M_k \in \mathbb{R}^{d \times d}$ ,  $\mu_k \in \mathbb{R}^d$  are optimized;  $w_{kj} \in \mathbb{R}^d$  are standard Gaussian vectors that are drawn before optimizing and held fixed.

We again performed 5 fold cross validation and report the mean prediction error on test sets in Table 6.2. The results in Table 6.2 show that BaNK consistently performed better or as well as the baselines.

### 6.4.4 Timing Experiments

We empirically investigate the linear scaling of the BaNK method in terms of the number of instances  $N$ . It is this linear dependence that allows BaNK to perform kernel learning in large datasets, where naive kernel methods are generally  $\Omega(N^2)$ . We hold the number of random frequencies fixed at  $M = 384$  and vary  $N$  to study the empirical dependence of runtime on dataset size for BaNK. We see the linear scaling is empirically verified in Figures 6.4b and 6.4c. Also, we observe lower test errors as dataset sizes increase indicating

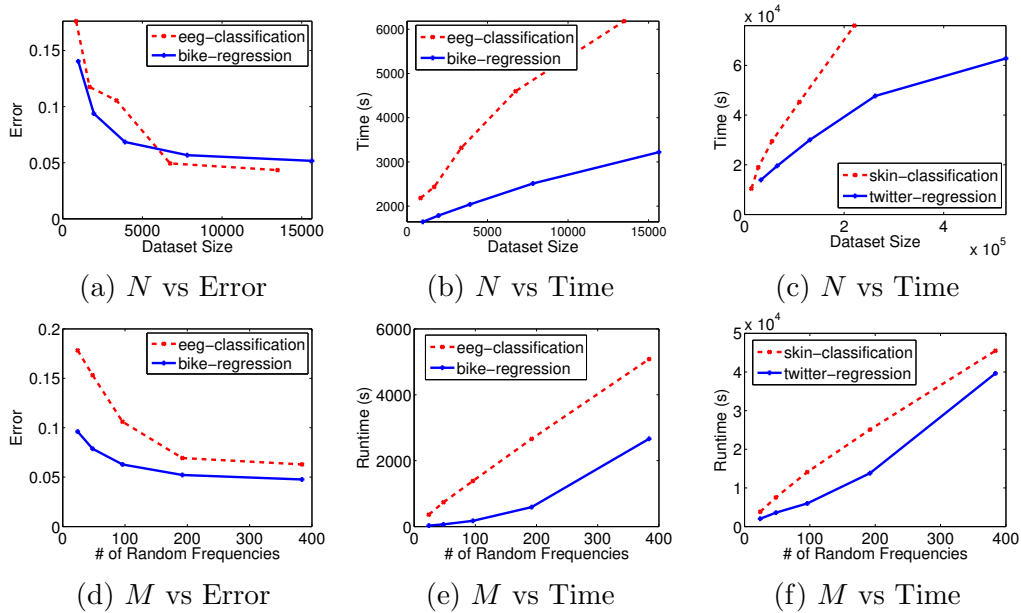


Figure 6.4: Runtime experiments. In all figures we denote classification curves with a dashed red line and regression curves with a solid blue line. Dataset names are shown in legends. Errors are prediction errors for classification tasks and MSE for regression tasks. Figure (a) shows how error changes with number of instances  $N$ ; (b,c) shows the effect of increase in number of instances on runtime; (d) shows how error changes with number of random frequencies  $M$ ; (e,f) show the effect of number of frequencies on computational time.

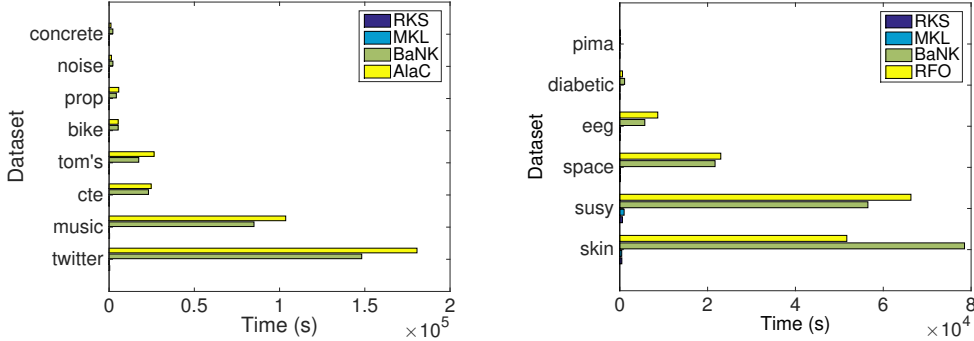


Figure 6.5: Runtimes for datasets.

that BaNK is able to leverage more data to learn effective kernels and increase accuracy (Figure 6.4a).

Furthermore, we illustrate the scaling of our BaNK method for regression and classification in terms of the number of random frequencies,  $M$ . As discussed in Section 6.3, the run-time complexity in large datasets will be  $O(NM^2)$  for regression and  $O(MNd)$  for classification. We empirically study the dependence of  $M$  on runtimes by varying  $M \in \{24, 48, 96, 192, 384\}$  and recording the runtime of BaNK using fixed datasets. Figures 6.4f and 6.4e show a quadratic growth in runtime for regression and a linear growth for classification. We see similar trends on other datasets. Moreover, we observe in Figure 6.4d an increase in performance with diminishing returns as  $M$  increases. Thus, we see that more frequencies aid kernel approximation and accuracy, but performance stabilizes after enough frequencies are chosen.

Lastly, we record the runtimes on each dataset (Figure 6.5) with the total number of random frequencies fixed at  $M = 384$  for all methods. While restricted methods that consider only a fixed set of random frequencies (RKS and MKL) perform fast, we see that BaNK's runtime is comparable to other methods that learn the random frequencies (AlaC and RFO) and can scale to large datasets.

## 6.5 Conclusion

In this chapter we propose an efficient and general data driven framework, BaNK, for learning of kernels that scales to large datasets. By representing the spectral density using a nonparametric mixture of Gaussians, we capture a large class of kernels that can be learned. We provide a generative model for learning kernels while performing regression and classification tasks, and propose novel MCMC based sampling schemes to infer parameters of the mixtures. We show that our proposed framework outperforms other scalable kernel learning methods on a variety of real world datasets in both classification and regression task.



# Chapter 7

## Distributions of Past Data for Sequence Modeling

Next we consider the use of distributions for the modeling of sequential data. In [94], we modify mean map embeddings for sequential modeling and propose the Statistical Recurrent Unit (SRU). The SRU makes use of moving averages of recurrent statistics kept at multiple scales to capture temporal dependencies with a simple architecture that does not contain any gates. Notwithstanding this simple architecture, we show that the SRU often performs favorably when compared to more complicated alternatives like GRUs and LSTMs.

### 7.1 Introduction

The analysis of sequential data has long been a staple in machine learning. Domain areas like natural language [142, 131], speech [41, 40], music [20], and video [25] processing have recently garnered much attention. While the study of sequences itself is broad and may be extended to general functional analysis [105], most recent success has been from neural network based models, especially from recurrent architectures.

Recurrent networks are dynamical systems that represent time recursively. For example, the simple recurrent unit [27] contains a hidden state that itself depends on the previous hidden state. However, training such networks has been observed to be difficult in practice due to exploding and vanishing gradients when propagating error gradients through time [46]. While exploding gradients can be mitigated with techniques like gradient clipping and normalization [99], vanishing gradients may be harder to deal with. As a result, sophisticated gated architectures like Long-Short Term Memory (LSTM) networks [47] and Gated Recurrent Units (GRU) networks [19] have been developed. These gated architectures contain “memory cells” along with gates to control how much they decay through time thereby aiding the networks’ ability to learn long term dependencies in sequences.

Notwithstanding, there are still challenges in capturing long term dependencies in gated architectures [65]. In this chapter we present a simple un-gated architecture, the Statistical Recurrent Unit, that often outperforms these more complicated alternatives. Although the SRU keeps only simple moving averages of summary statistics, its novel architecture

makes it more adept than previous gated units for capturing long term information in sequences and comparing them across different windows of time. For instance, the SRU, unlike traditional recurrent units, can obtain a multitude of viewpoints of the past by simple linear combinations of only a few averages. We shall illustrate the efficacy of the SRU below using both real-world and synthetic sequential data tasks.

The structure of the chapter is as follows: first we detail the architecture of the SRU as well as provide several key intuitions and insights for its design; after, we describe our experiments comparing the SRU to popular gated alternatives, and we perform a “dissective” study of the SRU, gaining further understanding of the unit by exploring how various hyper-parameters affect performance; finally, we discuss conclusions from our study.

## 7.2 Model

The SRU maintains long term sequential dependencies in a rather intuitive fashion—through summary statistics. As the name implies, statisticians often employ summary statistics when trying to represent a dataset. Quite naturally then, we look to an algorithm that itself learns to represent data seen previously in much the same vein as a neural statistician [26].

Of course, unlike with unordered i.i.d. samples, simply averaging statistics of sequential points will lose valuable temporal information. The SRU maintains sequential information in two ways: first, we generate recurrent statistics that depend on a context of previously seen data; second, we generate moving averages at several scales, allowing the model to distinguish the type of data seen at different points in the past. We expound on these methods for creating temporally-aware statistics below.

We shall see that the statistical design of the SRU yields a powerful yet simple model that is able to analyze sequential data and, on the fly, create summary statistics for learning over sequences. Furthermore, through the use of ReLUs and exponential moving averages, the SRU is able to overcome vanishing gradient issues that are common to many recurrent units.

### 7.2.1 Recurrent Statistics

We consider an input sequence of real valued points  $x_1, x_2, \dots, x_T \in \mathbb{R}^d$ . As seen in the second row of Table 7.1, we can compute a vector of statistics  $\phi(x_i) \in \mathbb{R}^D$  for each point. Here, each vector  $\phi(x_i)$  is independent of other points  $x_j$  for  $j \neq i$ . One may then average these vectors as  $\mu = \frac{1}{T} \sum_{i=1}^T \phi(x_i)$  to produce summary statistics of the sequence. This approach amounts to treating the sequence as a set of i.i.d. points drawn from some distribution and marginalizing out time. Clearly, here one will lose temporal information that will be useful for many sequence related ML tasks. It is interesting to note that global average pooling operations have gained a lot of recent traction in convolutional networks [68, 49]. Analogously to the i.i.d. statistic approach, global averaging will lose spatial information, yet the high-level summary statistics provide an effective representation. Still,

not marginalizing out time should provide a more robust approach for sequence tasks, thus we consider the following methods for producing statistics.

First, we provide temporal information whilst still utilizing averages through recurrent statistics that also depend on the values of previous points (see third row of Table 7.1). That is, we compute our statistics on the  $i^{\text{th}}$  point  $x_i$  not only as a function of  $x_i$ , but also as a function of the previous statistics of  $x_{i-1}$ ,  $\vec{\gamma}_{i-1}$  (which itself depends on  $\vec{\gamma}_{i-2}$ , etc.):

$$\vec{\gamma}_1 = \gamma(x_1, \vec{\gamma}_0), \vec{\gamma}_2 = \gamma(x_2, \vec{\gamma}_1), \dots \quad (7.1)$$

where  $\gamma(\cdot, \cdot)$  is a function for producing statistics given the current point and previous statistics, and  $\vec{\gamma}_0$  is a constant initial vector for convention. We note that from a general standpoint if given a flexible model and enough dimensions, then recurrent summary statistics like (7.1) can perfectly encode ones sequence. Take for instance the following illustrative example where  $x_i \in \mathbb{R}^+$  and statistics

$$\vec{\gamma}_i = (0, \dots, 0, Tx_i, 0, \dots) \quad (7.2)$$

$$\vec{\gamma}_{i+1} = (0, \dots, 0, 0, Tx_{i+1}, 0, \dots). \quad (7.3)$$

That is, one records the  $i^{\text{th}}$  input in the  $i^{\text{th}}$  index. When averaged the statistics will be  $\frac{1}{T} \sum_{i=1}^T \vec{\gamma}_i = (x_1, x_2, \dots)$ , i.e. the complete sequence. Such recurrent statistics will undoubtedly suffer from the curse of dimensionality. Hence, we consider a more restrictive model of recurrent statistics which we expound on below (7.6).

Second, we provide even more temporal information by considering summary statistics at multiple scales. We shed light on the dynamics of statistics through time by using several weights of the same summary statistics. As a simple hypothetical example consider taking multiple means across separate time windows (for instance taking means over indices 1-10, then over indices 11-20, etc.). Such an approach (7.4) will illustrate how summary statistics evolve through time.

$$\underbrace{\phi_1, \dots, \phi_{10}}_{\mu_{1:10}}, \underbrace{\phi_{11}, \dots, \phi_{20}}_{\mu_{11:20}}, \dots \quad (7.4)$$

The SRU will use exponential moving averages  $\mu_i = \alpha \vec{\gamma}_i + (1 - \alpha) \mu_{i-1}$  to compute means; hence, we consider multiple weights by taking the exponential means at various scales  $\alpha_1, \dots, \alpha_m$  as shown in the last row of Table 7.1. Later we show that this multi-scaled approach is capable of a combinatorial number of viewpoints of past statistics through simple linear combinations.

## 7.2.2 Update Equations

We have discussed in broad terms how one may create temporally-aware summary statistics through multi-scaled recurrent statistics. Below, we cover specifically how the SRU creates and uses summary statistics for sequences.

Recall that our input is a sequence of ordered points:  $\{x_1, x_2, \dots\}$ ,  $x_t \in \mathbb{R}^d$ . Throughout, we apply an element-wise non-linearity  $f(\cdot)$ , which we take to be the ReLU [52, 83]:

Table 7.1: Methods for keeping statistics of sequences.

inputs	$x_1, x_2, \dots, x_T$
i.i.d. statistics	$\phi(x_1), \phi(x_2), \dots, \phi(x_T)$
recurrent statistics	$\gamma(x_1, \vec{\gamma}_0), \gamma(x_2, \vec{\gamma}_1), \dots, \gamma(x_T, \vec{\gamma}_{T-1})$
recurrent multi-scaled statistics	$\alpha_1^{T-1}\gamma(x_1, \vec{\gamma}_0), \alpha_1^{T-2}\gamma(x_2, \vec{\gamma}_1), \dots$ $\dots$ $\alpha_m^{T-1}\gamma(x_1, \vec{\gamma}_0), \alpha_m^{T-2}\gamma(x_2, \vec{\gamma}_1), \dots$

$f(\cdot) = \max(\cdot, 0)$ . The SRU operates via exponential moving averages,  $\mu^{(\alpha)} \in \mathbb{R}^s$  (7.7), kept at various scales  $\alpha \in A = \{\alpha_1, \dots, \alpha_m\}$ , where  $\alpha_i \in [0, 1)$ . These moving averages,  $\mu^{(\alpha)}$ , are of recurrent statistics  $\varphi$  (7.6) that are dependent not only on the current input but also on features of averages,  $r$  (7.5). The moving averages are then concatenated as  $\mu = (\mu^{(\alpha_1)}, \dots, \mu^{(\alpha_m)})$  and used to create an output  $o$  (7.8) that is fed upwards in the network.

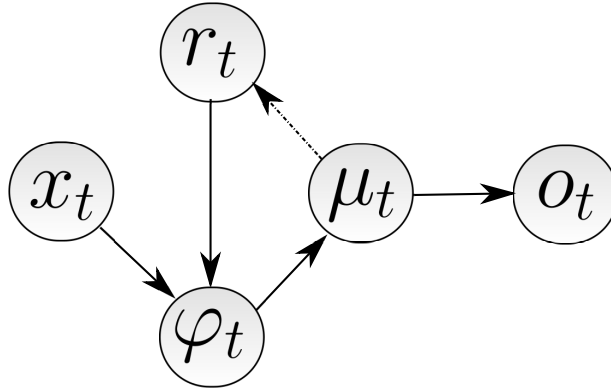


Figure 7.1: Graphical representation of the SRU. Solid lines indicate a dependence on the current value of a node. Dashed lines indicate a dependence on the previous value of a node. We see that both the current point  $x_t$  as well as a summary of the previous data  $r_t$  are used to make statistics  $\varphi_t$ , which in turn are used in moving averages  $\mu_t$ , finally an output  $o_t$  is feed-forward through the rest of the network.

We detail the update equations for the SRU below (and in Figure 7.1):

$$r_t = f(W^{(r)}\mu_{t-1} + b^{(r)}) \quad (7.5)$$

$$\varphi_t = f(W^{(\varphi)}r_t + W^{(x)}x_t + b^{(\varphi)}) \quad (7.6)$$

$$\forall \alpha \in A, \mu_t^{(\alpha)} = \alpha\mu_{t-1}^{(\alpha)} + (1 - \alpha)\varphi_t \quad (7.7)$$

$$o_t = f(W^{(o)}\mu_t + b^{(o)}) \quad (7.8)$$

In practiced we noted that it suffices to use only a few  $\alpha$ 's such as  $A = \{0, 0.25, 0.5, 0.9, 0.99\}$ .

It is worth noting that exponential averages of inputs has been considered previously [73]. However, that approach performs a moving average of a linear features (specifically the identity mapping) that depends only on the current observation, which is fairly inflexible. Furthermore, such work considers only one scale per feature, limiting the views available per statistic to just one. The use of ReLUs in recurrent units has also been recently explored by Le, Jaitly, and Hinton [65], however there no statistics are kept and their use is limited to the simple RNN when initialized in a special manner.

### 7.2.3 Intuitions from Mean Map Embeddings

The design of the SRU is deliberately chosen to allow for long term dependencies to be learned. To better elucidate the design and its intuition, let us take a brief excursion to another use of (summary) statistics in machine learning for the representation of data: mean map embeddings (MMEs) of distributions [114]. Recall that at its core, the concept of MMEs is that one may embed, and thereby represent, a distribution through statistics (such as moments). The MME for a distribution  $\mathcal{D}$  given a positive semidefinite kernel  $k$  is:

$$\mu[\mathcal{D}] = \mathbb{E}_{X \sim \mathcal{D}} [\phi_k(X)], \quad (7.9)$$

where  $\phi_k$  are the reproducing kernel Hilbert space (RKHS) features of  $k$ , which may be infinite dimensional. To represent a set  $Y = \{y_1, \dots, y_n\} \stackrel{iid}{\sim} \mathcal{D}$  one would use an empirical mean version of the MME:

$$\mu[Y] = \frac{1}{n} \sum_{i=1}^n \phi_k(y_i). \quad (7.10)$$

Numerous works have shown success in representing distributions and sets through MMEs [80]. One interpretation for the design of SRUs is that we are modifying MME's for use on sequences. Of course, one way of applying MMEs directly on sequences is to simply ignore the non-i.i.d. nature of sequences and treat points as comprising a set. This however loses important sequential information, as previously mentioned. Below we discuss the specific modifications we make from traditional MMEs and the benefits they yield.

#### Data-driven Statistics

First, we note the clear analogue between the mean embedding of a set  $Y$ ,  $\mu[Y]$  (7.10), and the moving average  $\mu^{(\alpha)}$  (7.7). The moving averages  $\mu^{(\alpha)}$  are clearly serving as summary statistics of previously seen data. However, the statistics we are averaging for  $\mu^{(\alpha)}$ ,  $\varphi$  (7.6), are not comprised of a-priori RKHS features as is typical with MMEs, but rather are learned non-linear features. This has the benefit of using data-driven statistics, and may be interpreted as using a linear kernel in the learned features.

#### Recursive Statistics from the Past

Second, recall that typical MMEs use statistics that depend only on a single point  $x$ ,  $\phi_k(x)$ . As aforementioned this is fine for i.i.d. data, however it loses sequential information when

averaged. Instead, we wish to assign statistics that depend on the data we have seen so far, since it provides context for one’s current point in the sequence. For instance, one may want to have a statistic that keeps track of the difference between the current point and the mean of previous data. We provide a context based on previous data by making the statistics considered at time  $t$ ,  $\varphi_t$  (7.6), a function not only of  $x_t$  but also of  $\{x_1, \dots, x_{t-1}\}$  through  $r_t$  (7.5).  $r_t$  may be interpreted as a condensation of the sequence seen so far, and allows us to keep sequential information even through an averaging operation.

### Multi-scaled Statistics

Third, the use of multi-scaled moving averages of statistics gives the SRU a simple and powerful rich view of past data that is unique to this recurrent unit. In short, by keeping moving averages at different scales  $\{\alpha_1, \dots, \alpha_m\}$ , we are able to uncover differences in statistics at various times in the past. Note that we may unroll moving averages as:

$$\mu_t^{(\alpha)} = (1 - \alpha) (\varphi_t + \alpha\varphi_{t-1} + \alpha^2\varphi_{t-2} + \dots) \tag{7.11}$$

Thus, a smaller  $\alpha$  weighs current statistics more than older statistics; hence, a concatenated vector  $\mu = (\mu^{(\alpha_1)}, \dots, \mu^{(\alpha_m)})$  itself provides a multi-scale view of statistics through time (see Figure 7.2). For instance, keeping statistics for short and long terms pasts already yields information on the evolution of the sequence through time.

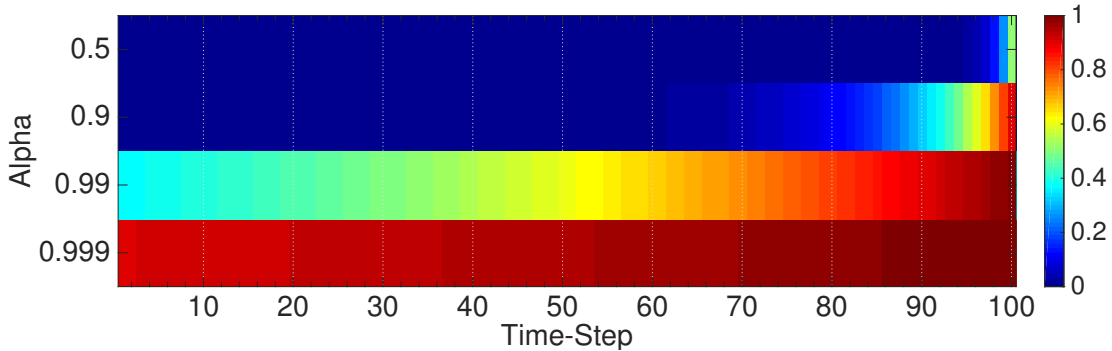


Figure 7.2: We may unroll the moving average updates as (7.11). To visualize the different emphasis in the past that varying  $\alpha$  has on statistics we plot the values of weights in moving averages (i.e.  $\alpha^i$ ) for 100 points in the past across rows. We see that alpha values closer to 0 focus only on the recent past, where values close to 1 maintain an emphasis on the distant past as well.

### 7.2.4 Viewpoints of the Past

An interesting and useful property of keeping multiple scales for each statistic is that one can obtain a combinatorial number of viewpoints of the past through simple linear combinations of ones statistics. For instance, for properly chosen  $w_j, w_k \in \mathbb{R}$ ,  $w_j\mu^{(\alpha_j)} - w_k\mu^{(\alpha_k)}$  provides an aggregate of statistics from the past for  $\alpha_j > \alpha_k$  (Figure 7.3). Of course, more complicated

linear combinations may be performed to obtain richer viewpoints that are comprised of multiple windows. Furthermore, by using a linear projection of our statistics  $\mu_t$ , as we do with  $o_t$  (7.8), we are able to compute output features of combined viewpoints of several statistics.

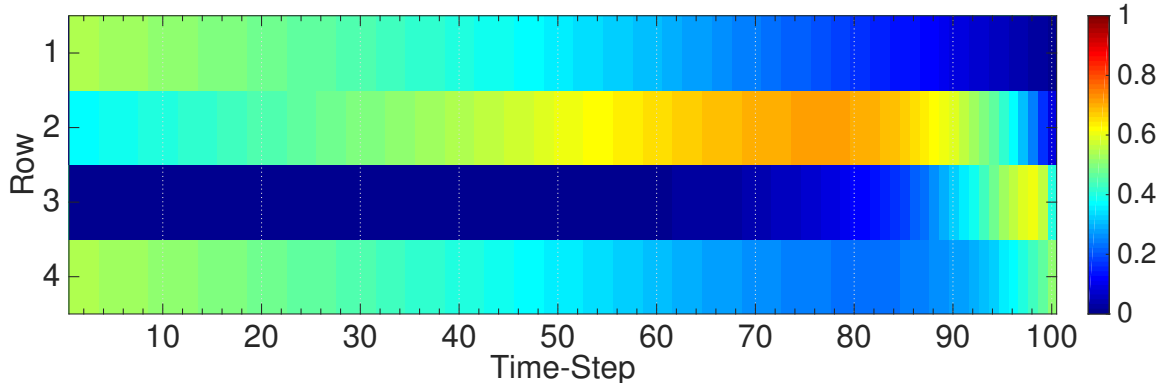


Figure 7.3: We visualize the power of taking linear combinations of  $\mu^{(\alpha)}$ 's for providing different viewpoints into past data. In row 1 we show the effective weights that would be used for weighing statistics  $\varphi_t$  if one considers  $.001^{-1}\mu^{(.999)} - .01^{-1}\mu^{(.99)}$ ; we see that this is equivalent to considering only statistics from the distant past. Similarly, we show the effective weights when taking  $.01^{-1}\mu^{(.99)} - .1^{-1}\mu^{(.9)}$  and  $.1^{-1}\mu^{(.9)} - .5^{-1}\mu^{(.5)}$  on rows 2 and 3 respectively. We see that these linear combinations amount to considering viewpoints concentrated at various points in the past. Lastly its worth noting that more complicated linear combinations may lead to even richer views on previous statistics; for instance, we show  $.001^{-1}\mu^{(.999)} - .01^{-1}\mu^{(.99)} + \frac{.5}{.09}\mu^{(.9)}$  on row 4, which concentrates on the statistics of the distant and very recent past, but de-emphasizes statistics of data from less recent past.

This kind of multi-viewpoint perspective of previously seen data is difficult to produce in traditional gated recurrent units since they must encode where in the sequence they currently are and then store an activation on separate nodes per each viewpoint for future use. SRUs, on the other hand, only need to take simple linear combinations to capture various viewpoints in the past. For example, as shown above, statistics from just the distant past are available via a simple subtraction of two moving averages (Figure 7.3, row 1). Such a windowed view would require a gated unit to learn to stop averaging after a certain point in the sequence, and the corresponding statistic would not yield an information outside of this window. In contrast, each statistic kept by the SRU provides a combinatorial number of varying perspectives in the past through linear combinations and their multi-scaled nature.

## 7.2.5 Vanishing Gradients

As previously mentioned, it has been shown that vanishing gradients make learning recurrent units difficult due to an inability to propagate error gradients through time. Notwithstanding its simple un-gated structure, the SRU features several safeguards to alleviate vanishing gradients. First, units and statistics are comprised of ReLUs. ReLUs have been observed to

be easier to train for general deep networks [83] and have had success in recurrent units [65]. Intuitively, ReLUs allow for the propagation on error on positive inputs without saturation and vanishing gradients as with traditional sigmoid units. The ability of the SRU to use ReLUs (without any special initialization) makes it especially adept at learning long term dependencies through time.

Furthermore, the explicit moving average of statistics allows for longer term learning. Consider the following derivative of the error signal  $E$  w.r.t. an element  $\left[\mu_{t-1}^{(\alpha)}\right]_k$  of the unit’s moving averages when  $[\varphi_t]_k = 0$ :

$$\frac{\partial E}{\partial \left[\mu_{t-1}^{(\alpha)}\right]_k} = \frac{\partial \left[\mu_t^{(\alpha)}\right]_k}{\partial \left[\mu_{t-1}^{(\alpha)}\right]_k} \frac{\partial E}{\partial \left[\mu_t^{(\alpha)}\right]_k} = \alpha \frac{\partial E}{\partial \left[\mu_t^{(\alpha)}\right]_k}.$$

That is, the factor  $\alpha$  directly controls the decay of the error signal through time. Thus, by including an  $\alpha$  explicitly near 1 (i.e. 0.999), the decay for that moving average can be made minuscule for the lengths of sequences in ones data. Also, it is interesting to note that, with a large  $\alpha$  near 1, SRUs with ReLUs can implement part of the functionality of a gate (“remembering”) by carrying through the previous moving average  $[\mu_{t-1}^{(\alpha)}]_k$  when the corresponding statistic  $[\varphi_t]_k$  has be zeroed out (7.7). The other functionality of a gate (forgetting) can be had by including an  $\alpha$  near 0; if the ReLU statistic is not zeroed out, then the moving average for a small  $\alpha$  will “forget” the previous value.

## 7.3 Experiments

We compared the performance of the SRU<sup>1</sup> to two popular gated recurrent units, the GRU and LSTM unit. All experiments were performed in `Tensorflow` [1] and used the standard implementations of `GRUCell` and `BasicLSTMCell` for GRUs and LSTMs respectively. In order to perform a fair, unbiased comparison of the recurrent units and their hyper-parameters, which greatly affect performance [8], we used the `Hyperopt` [9] hyper-parameter optimization package. We believe that such an approach gives each algorithm a fair shot to succeed without injecting biases from experimenters or imposing gross restrictions on architectures considered.

In all experiments we used SGD for optimization using gradient clipping [99] with a norm of 1 on all algorithms. Unless otherwise specified 100 trials were performed to search over the following hyper-parameters on a validation set: one, `initial_learning_rate` the initial learning rate used for SGD, in range of  $[\exp(-10), 1]$ ; two, `lr_decay` the multiplier to multiply the learning rate by every 1k iterations, in range of  $[0.8, 0.999]$ ; three, `dropout_keep_rate`, percent of output units that are kept during dropout, in range  $(0, 1]$ ; four, `num_units` number of units for recurrent unit, in  $\{1, \dots, 256\}$ . In addition, the following two parameters were searched over for the SRU: `num_stats`, the dimensionality of  $\varphi$  (7.6), in  $\{1, \dots, 256\}$ ; `summary_dims`, the dimensionality of  $r$  (7.5), in  $\{1, \dots, 64\}$ .

<sup>1</sup>See <http://www.cs.cmu.edu/~joliva/sru.py> for code.



### 7.3.1 Synthetic Recurrent Unit Generated Data

First we provide evidence that traditional gated units have difficulties capturing the same type of multi-scale recurrent statistic based dependencies that the SRU offers. We show the relative inefficiency of traditional gated units at learning long term dependencies of statistics by considering 1d synthetic data from a ground truth SRU.

We begin the sequences with  $x_1 \stackrel{iid}{\sim} \mathcal{N}(0, 100^2)$ , and  $x_t$  is the results of a projection of  $o_t$ . We generate a total of 176 points per sequence for 3200 training sequences, 400 validation sequences, and 400 testing sequences.

The ground truth statistical recurrent unit has three statistics  $\phi_t$  (7.6): the positive part of inputs  $(x)_+$ , the negative part of inputs  $(x)_-$ , and an internal statistic,  $z$ . We use  $\alpha \in \{\alpha_i\}_{i=1}^5 = \{0.0, 0.5, 0.9, 0.99, 0.999\}$ . Denote  $\mu_+^{(\alpha)}$ ,  $\mu_-^{(\alpha)}$ ,  $\mu_z^{(\alpha)}$  as the moving averages using  $\alpha$  for each respective statistic. The internal statistic  $z$  does not get used (through  $r_t$  (7.5)) in updating the statistics for  $(x)_+$  or  $(x)_-$ .  $z$  is itself updated as:

$$z_t = (z_{t-1})_+ + \left( \mu_+^{(\alpha_4)} - \mu_+^{(\alpha_5)} - 0.01 \right)_+ - \left( -\mu_-^{(\alpha_4)} + \mu_-^{(\alpha_5)} - 0.01 \right)_+ \\ - \left( -\mu_+^{(\alpha_4)} + \mu_+^{(\alpha_5)} - 0.05 \right)_+ + \left( \mu_-^{(\alpha_4)} - \mu_-^{(\alpha_5)} - 0.05 \right)_+,$$

where each of the summands are  $r_t$  features. Furthermore we have  $o_t \in \mathbb{R}^{15}$  (7.8):

$$o_t = \left( (x_t)_+, -(x_t)_-, v_1^T \mu_t, \dots, v_{13}^T \mu_t \right),$$

where  $v_j$ 's were initialized and fixed as  $(v_j)_k \stackrel{iid}{\sim} \mathcal{N}(0, (\frac{1}{100})^2)$ . Finally the next point is generated as:

$$x_{t+1} = (x_t)_+ - (x_t)_- + w^T o_{t,3:},$$

where  $w$  was initialized and fixed as  $(w)_k \stackrel{iid}{\sim} \mathcal{N}(0, 1)$ , and  $o_{t,3:}$  are the last 13 dimensions of  $o_t$ .

After the ground truth SRU was constructed we generated the training, validation, and testing sequences. As can be seen in Figure 7.4, the sequences follow a simple pattern: at the start negative values are quickly pushed to zero and positive values follow a parabolic line until hitting zero, at which point they slope downward depending on initial values. While simple, it is clear that trained recurrent units must be able to hold long-term information since all sequences converge at one point and future behaviour depends on initial values.

We look to minimize the mean of squared errors (MSE); that is, the loss we consider per sequence is  $\frac{1}{175} \sum_{t=1}^{175} |x_{t+1} - p_t|^2$ , where  $p_t$  is the output of the network after being fed  $x_t$ . We conducted 100 trials of hyper-parameter optimization as described above and obtained the following results in Table 7.2.

Table 7.2: MSEs for synthetically generated dataset.

	SRU	GRU	LSTM
Error	<b>0.62</b>	21.72	161.62

Not surprisingly, the SRU performs far better than traditional gated recurrent units. This suggests that the types of long-term statistical relationships captured by the SRU

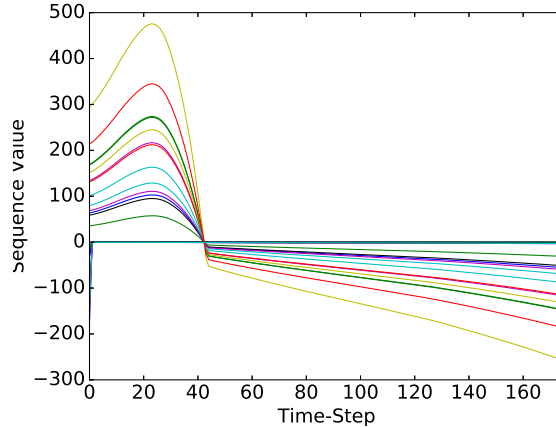


Figure 7.4: 25 sequences generated from the ground truth SRU model.

are indeed different than those of traditional recurrent units. As previously mentioned, the SRU is able to obtain a multitude of different views from its statistics, a task that traditional units achieve less efficiently since they must devote one whole memory cell per viewpoint and statistic. As we show below, the SRU is able to outperform traditional gated units in long term problems even for real data that is not generated from its model class.

### 7.3.2 MNIST Image Classification

Next we explore the ability of recurrent units to use long-term dependencies in ones data with a synthetic task using a real dataset. It has been observed that LSTMs perform poorly in classifying a long pixel-by-pixel sequence of MNIST digits [65]. In this synthetic task, each  $28 \times 28$  gray-scale MNIST digit image is flattened and observed as a sequence  $\{x_1, \dots, x_{784}\}$ , where  $x_i \in [0, 1]$  (see Figure 7.5). The task is, based on the output observed after feeding  $x_{784}$  through the network, to classify the digit of the corresponding image in  $\{0, \dots, 9\}$ . Hence, we project the output after  $x_{784}$  of each recurrent unit to 10 dimensions and use a softmax activation.

We report the hyper-parameter optimized results below in Table 7.3; due to resource constraints each trial consisted only of 10K training iterations. We see that the SRU is able to out-perform both GRUs and LSTMs. Given the long length and dependencies of pixel sequences in this experiment, it is not surprising that SRUs’ abilities to capture long-term dependencies are aiding it to achieve a much lower error.

Table 7.3: Test error rate for MNIST pixel sequence classification.

	SRU	GRU	LSTM
Error Rate	<b>0.11</b>	0.28	0.48

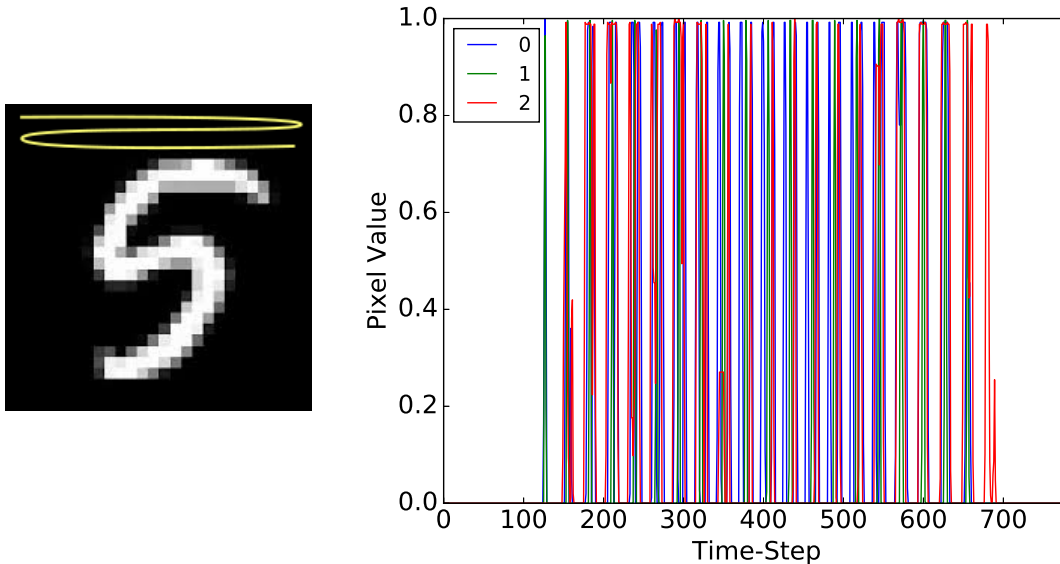


Figure 7.5: Right: example MNIST  $28 \times 28$  image, which is taken as a pixel-by-pixel sequence of length 784 unrolled as shown in yellow. Left: example pixel sequences for 0, 1, and 2 digit images.

## Dissective Study

Next, we study the behavior of the statistical recurrent unit with a dissective study where we vary several parameters of the architecture. We consider variants to a base model with: `num_stats=200`; `r_dims=60`; `num_units=200`. We keep the parameters `initial_learning_rate`, `lr_decay` fixed at the the optimal values found (0.1, 0.99 respectively) unless we find no learning, in which case we also try learning rates of 0.01 and 0.001.

**The need for multi-scaled recurrent statistics.** Recall that we designed the statistics used by the SRU expressly to capture long term time dependencies in sequences. We did so both with recurrent statistics, i.e. statistics that themselves depend on previous points' statistics, and with multi-scaled averages. We show below that both of these time-dependent design choices are vital to capturing long term dependencies in data. Furthermore, we show that the use of ReLU statistics lends itself to better learning.

We explored the impact that time-dependent statistics had on learning by first considering naive i.i.d. summary statistics for sequences. This was achieved by using `r_dims=0` and  $\alpha \in A = \{0.99999\}$ . Here no past-dependent context is used for statistics, i.e. we used i.i.d.-type statistics as is typical for unordered sets. Furthermore, the use of a single scale  $\alpha$  near 1 means that all of the points' statistics will be weighted nearly identically (7.11) regardless of index. We optimized the SRU when using no recurrent statistics and a single scale (`iid`), when using recurrent statistics with a single scale (`recur`), and when using no recurrent statistics with multiple scales (`multi`). We report errors below in Table 7.4.

Predictably, we cannot learn by simply keeping i.i.d. type statistics of pixel values at a

Table 7.4: Test error rate for MNIST pixel sequence classification.

	iid	recur	multi
Error Rate	0.88	0.88	0.63

single scale. Furthermore, we find that only using recurrent statistics (**recur**) in the SRU is not enough. It is interesting to note, however, that keeping i.i.d. statistics at multiple scales is able to predict digits with limited success. This lends evidence for the need of *both* recurrent statistics and multiple scales.

Next, we explored the effects of the scales at which we keep our statistics by varying from  $\alpha \in A = \{0.0, 0.5, 0.9, 0.99, 0.999\}$  considering  $\alpha \in A = \{0.0, 0.5, 0.9\}$ ,  $\alpha \in A = \{0.0, 0.5, 0.9, 0.99\}$ . We see in Table 7.5 that additional, longer scales aid our learning for this dataset. This is not very surprising given the long term nature of the pixel sequences.

Table 7.5: Test error rate for MNIST pixel sequence classification.

A	{0.0, 0.5, 0.9}	{0.0, 0.5, 0.9, 0.99}
Error Rate	0.79	0.21

Lastly, we considered the use of non-ReLU statistics by changing the element-wise non-linearity  $f(\cdot)$  (7.5)-(7.8) to be the hyperbolic tangent  $f(\cdot) = \tanh(\cdot)$ . We postulated that the use of ReLUs would help our learning since they have been observed to better handle the problem of vanishing gradients. We find evidence of this when swapping ReLUs for hyperbolic tangent units in SRUs: we get an error rate of 0.18 when using hyperbolic tangent units. Although the previous uses of ReLUs in RNN required careful initialization [65], SRUs are able to use ReLUs for better learning without any special considerations.

**Dimension of recurrent summary.** Next we explore the effect of varying the number of dimensions used for the recurrent summary of statistics  $r_t$  (7.5). We consider **r\_dims** in  $\{5, 20, 240\}$ . As previously discussed  $r_t$  provides a context based on past data so that the SRU may produce non-i.i.d. statistics as it moves along a sequences. As one would expect the dimensionality of  $r_t$  will limit the information flow from the past and values that are too small will hinder performance. It is also interesting to see that after enough dimensions, there are diminishing returns to adding more.

Table 7.6: Test error rate varying recurrent summary  $r_t$ .

r_dims	5	20	240
Error Rate	0.25	0.20	0.10

**Number of statistics and outputs.** Finally, we vary the number of statistics **num\_stats**, and outputs **units**. Interestingly the SRU seems robust to the number of outputs propagated in the network. However, performance is considerably affected by the number of statistics considered.

Table 7.7: Test error rate varying number of units.

	num_stats		units	
	10	50	10	50
Error Rate	0.88	0.32	0.15	0.15

### 7.3.3 Polyphonic Music Modeling

Henceforth we consider real data and sequence learning tasks. First, we used the polyphonic music datasets from [15]. Each time-step is a binary vector representing the notes played at the respective time-step. Since we were required to predict binary vectors we used the element-wise sigmoid  $\sigma$ . I.e., the binary vector of notes  $x_{t+1}$  was modeled as  $\sigma(p_t)$ , where  $p_t$  is the output after feeding  $x_t$  (and previous values  $x_1, \dots, x_{t-1}$ ) through the recurrent network.

It is interesting to note in Table 7.8 that the SRU is able to outperform one of the traditional gated units in every dataset and it outperforms both in two datasets.

Table 7.8: Test negative log-likelihood for polyphonic music data.

Data set	SRU	GRU	LSTM
JSB	<b>8.260</b>	8.548	8.393
Muse	6.336	6.429	<b>6.293</b>
Nottingham	3.362	3.386	<b>3.359</b>
Piano	<b>7.737</b>	7.929	7.931

### 7.3.4 Electronica-Genre Music MFCC

In the following experiment we modeled the Mel frequency cepstrum coefficients (MFCCs) in a dataset of nearly 18 000 scraped 30s sound clips of electronica-genre songs. MFCCs are perceptually based spectral features positioned logarithmically on the mel scale, which approximates the human auditory system’s response [82]. We looked to model the 13 real-valued coefficients using the recurrent units, by modeling  $x_{t+1}$  as a projection of the output of a recurrent unit after being fed  $x_1, \dots, x_t$ .

Table 7.9: Test-set MSEs of MFCC Music data.

	SRU	GRU	LSTM
Error	<b>1.176</b>	2.080	1.183

As can be seen in Table 7.9, SRUs again are outperforming gated architectures and are especially beating GRUs by a wider margin.

### 7.3.5 Climate Data

Next we consider weather data prediction using the North America Regional Reanalysis (NARR) Project. The dataset provides a long-term set of consistent climate data on a

regional scale for the North American domain. The period of the reanalyses is from October 1978 to the present and analyses were made 8 times daily (3 hour intervals).

We take our input sequences to be year-long sequences of weather variables in a location for the year 2006. I.e. an input sequence will be a 2920 length sequence of weather variables at a given lat/lon coordinate. We considered the following 7 variables: `pres10m`, 10 m pressure (pa); `tcdc`, total cloud cover (%); `rh2m`, relative humidity 2m (%); `tmpsfc`, surface temperature (k); `snod`, snow depth surface (m); `ugrd10m`, u component of wind 10m above ground; `vgrd10m`, v component of wind 10m above ground. The variables were standardized, see Figure 7.6 for example sequences.

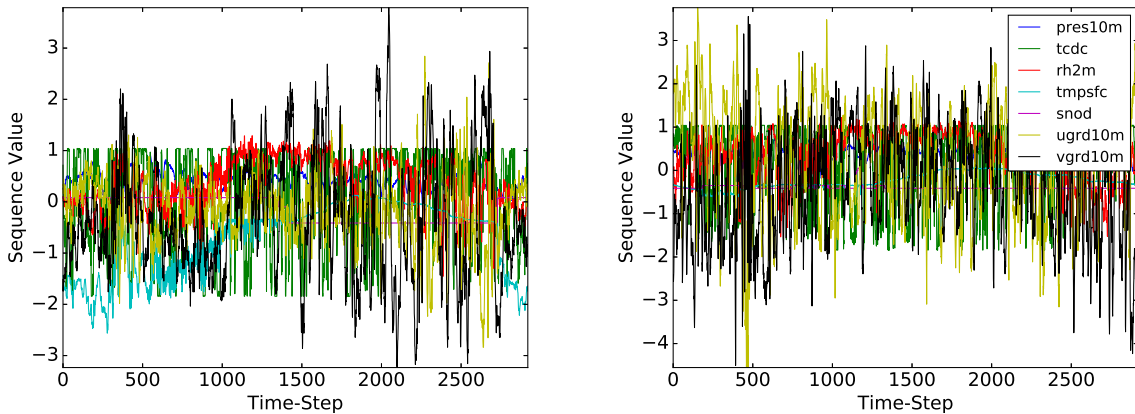


Figure 7.6: Two example sequences for weather variables at distinct locations for the year 2006.

Below we see results using 51 200 training location sequences and 6 400 validation and testing instances. Again, we look to model the next point in a sequence as a projection of the output of the recurrent unit after feeding the previous points. One may see in Table 7.10 that SRUs and LSTMs perform nearly identically; perhaps the cyclical nature of climate data was beneficial to the gated units.

Table 7.10: Test MSEs for weather data.

	SRU	GRU	LSTM
Error	<b>0.465</b>	0.487	0.466

### 7.3.6 SportVu NBA Tracking data

Finally, we look to predict the positions of National Basketball Association (NBA) players based on previous court positions during a play. Optical tracking data for this project were provided by STATS LLC from their SportVU product and obtained from [84]. The data are composed of  $x$  and  $y$  coordinates for each of the ten players and the ball. We again minimize the squared norm of errors for predictions.

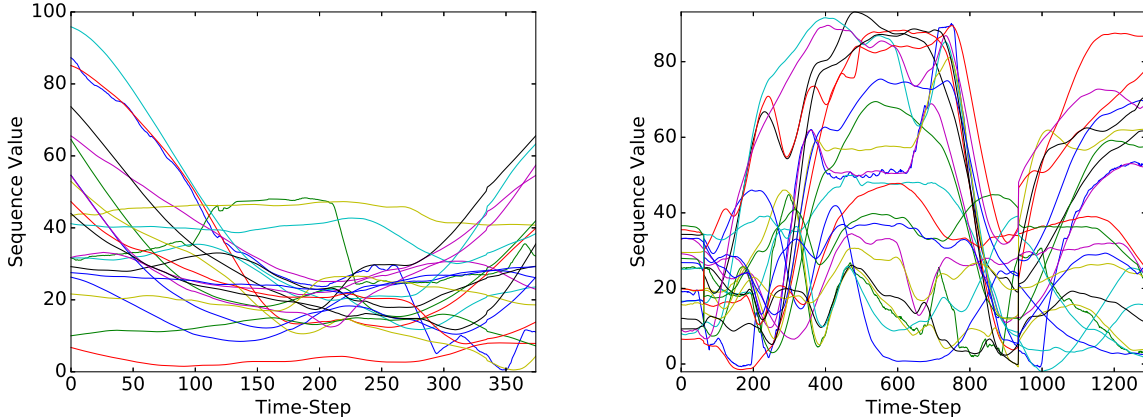


Figure 7.7: Example player/ball  $x, y$  positions for two plays.

Table 7.11: Test-set MSEs of NBA data.

	SRU	GRU	LSTM
Error	<b>34.505</b>	329.921	296.908

We observed a large margin of improvement for SRUs over gated architectures in Table 7.11 that is reminiscent of the synthetic data experiment in §7.3.1. This suggests that this dataset contains long term dependencies that the SRU is able to exploit.

## 7.4 Discussion

We believe that the use of summary statistics has been under-explored in modern recurrent units. Although recent studies in convolutional networks have considered global average pooling, which is essentially using high-level summary statistics to represent images, there has been little exploration of summary statistics for modern recurrent networks. To this end we introduce the Statistical Recurrent Unit, a novel architecture that seeks to capture long term dependencies in data using only simple moving averages and rectified-linear units.

The SRU was motivated by the success of mean-map embeddings for representing unordered datasets, and may be interpreted as an alteration of MMEs for sequential data. The main modifications are as follows: first, the SRU uses data-driven statistics unlike typical MMEs, which will use RKHS features from an a-priori selected class of kernels; second, SRUs will use recurrent statistics that are dependent not only on a current point, but on previous points’ statistics through a condensation of kept moving averages; third, the SRU will keep moving averages at various scales. We provide evidence that the combination of these modifications yield much better results than any one of them in isolation.

The resulting recurrent unit is especially adept for capturing long term dependencies in data and readily has access to a combinatorial number of viewpoints of past windows through simple linear combinations. Moreover, it is interesting to note that even though the SRU is gate-less, it may implement part of both “remembering” and “forgetting” functionalities

through ReLUs and moving averages.

We showed empirically that the SRU is comparable or better than traditional gated units for long term dependencies via synthetic and real-world data experiments.



# Chapter 8

## Transformation Autoregressive Networks

Thus far we have leveraged standard nonparametric representations of distributions like orthonormal series estimators to learn over distributions as inputs and/or outputs. However, these nonparametric representations often fail to accurately represent distributions past even fairly modest dimensions ( $\gtrsim 4$ ). Primarily, these representations suffer greatly from the curse of dimensionality where the amount of space grows exponentially, and observing a dense sample is no longer tenable. In order to overcome these shortcomings, we developed novel methods for high dimensional estimation. At a high-level, our goal is to develop a model that is expressive enough to capture a variety of real-world dependencies in covariates, but not so flexible that it is unable to generalize. To this end we propose Transformation Autoregressive Networks (TANs) [89], which jointly leverage transformations of variables and autoregressive conditional models, and proposes novel methods for both. Below we provide a deeper understanding of our methods, showing a considerable improvement through a comprehensive study over both real world and synthetic data. Moreover, we illustrate the use of our models in outlier detection and image modeling tasks.

### 8.1 Introduction

Density estimation is at the core of a multitude of machine learning applications. However, this fundamental task, which encapsulates the understanding of data, is difficult in the general setting due to issues like the curse of dimensionality. Furthermore, for general data, unlike spatial/temporal data, we do not have known correlations a priori among covariates that may be exploited. For example, image data has known correlations among neighboring pixels that may be hard-coded into a model, whereas one must find such correlations in a data-driven fashion with general data.

The main challenge to model high dimensional data lies in constructing models that are flexible enough while having tractable learning algorithms. A variety of diverse solutions exploiting different aspects of the problems have been proposed in the literature. A large number of methods have considered auto-regressive models to estimate the condi-

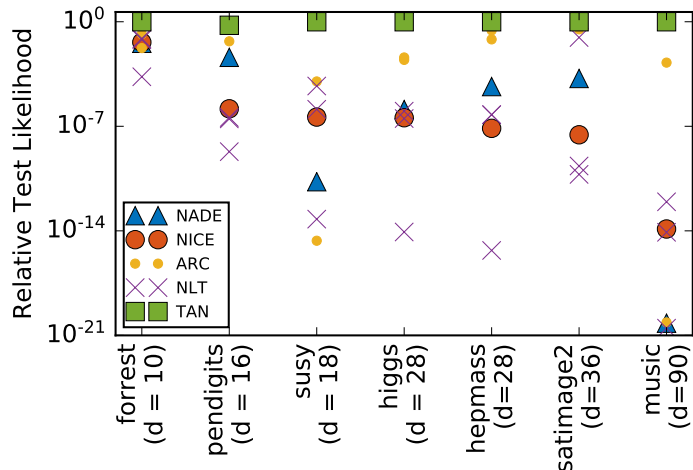


Figure 8.1: The proposed *TAN* models for density estimation, which jointly leverages non-linear transformation and autoregressive conditionals, shows considerable improvement over other methods across datasets of varying dimensions. The scatter plots shows that only utilizing autoregressive conditionals (ARC) without transformations (e.g. existing works like NADE [125] and other variants) or only using non-linear transformation (NLT) with simple restricted conditionals (e.g. existing works like NICE [22] and other variants) is not sufficient for all datasets.

tional factors  $p(x_i|x_{i-1}, \dots, x_1)$ , for  $i \in \{1, \dots, d\}$  in the chain rule [63, 126, 124, 37, 43] (Figure 8.2b). While some methods directly model the conditionals  $p(x_i|x_{i-1}, \dots)$  using sophisticated semiparametric density estimates, other methods apply sophisticated transformations of variables  $x \mapsto z$  and take the conditionals over  $z$  to be a restricted, often independent base distribution  $p(z_i|z_{i-1}, \dots) \approx f(z_i)$  [22, 23](Figure 8.2a). Further related works are discussed in Section 8.3. However, looking across a diverse collection of dataset, as in Fig. 8.1, neither of these approaches have the flexibility required to accurately model real world data.

In this chapter we take a step back and start from the basics. If we only model the conditionals, the factors  $p(x_i|x_{i-1}, \dots)$ , may become increasingly complicated as  $i$  increases to  $d$ . On the other hand if we use a complex transformation with restricted conditionals then the transformation has to ensure that the transformed variables are independent. This requirement of independence on the transformed variables can be very restrictive. Now note that the transformed space is homeomorphic to the original space and a simple relationship between the density of the two space exists through the Jacobian. Thus, we can employ conditional modeling on the transformed variables to alleviate the independence requirement, while being able to recover density in the original space in a straightforward fashion. In other words, we propose *transformation autoregressive networks* (TANs) which composes the complex transformations and autoregressive modeling of the conditionals (Figure 8.4). The composition not only increases the flexibility of the model but also reduces the expressibility needed from each of the individual components. This leads to an improved performance as can be seen from Fig. 8.1.

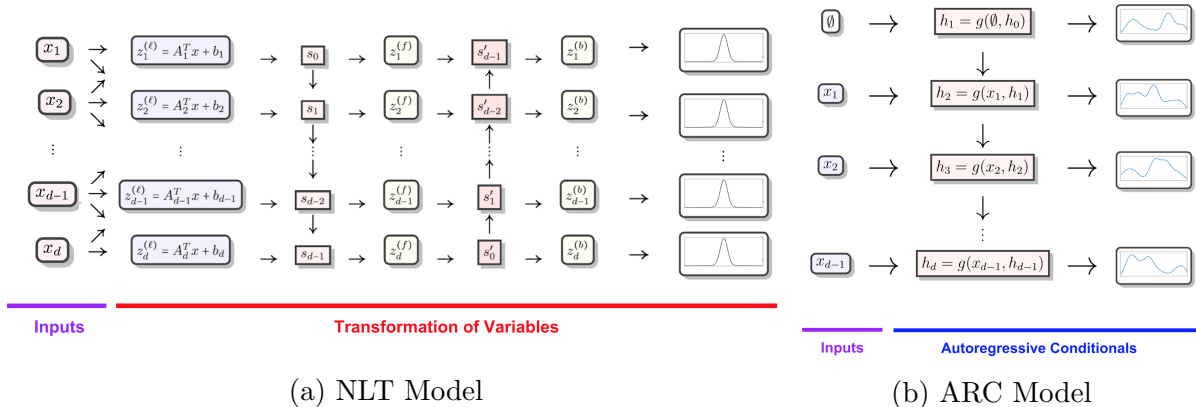


Figure 8.2: (a) Example NLT model, where covariates are transformed and modeled independently with a simple distribution (such as a standard Gaussian). (b) Example ARC model, where input covariates are directly (with no transformation) modeled using an autoregressive conditional model.

First we propose two flexible autoregressive models for modeling conditional distributions: the linear autoregressive model (LAM), and the recurrent autoregressive model (RAM) (Section 8.2.1). Secondly, we introduce several novel transformations of variables: 1) an efficient method for learning a linear transformations on covariates; 2) an invertible RNN-based transformation that directly acts on covariates; 3) an additive RNN-base transformation (Section 8.2.2). Extensive experiments on both synthetic (Section 8.4.1) and real-world (Section 8.4.2) datasets show the power of TANs for capturing complex dependencies between the covariates. We run an ablation study to demonstrate contributions of various components in TAN Section 8.4.3, Moreover, we show that the learned model can be used for anomaly detection (Section 8.4.4) and learning a family of distribution (Section 8.4.5).

## 8.2 Model

As mentioned above, TANs are composed of two modules: *a*) autoregressive module for modeling conditional factor and *b*) transformations of variables. We first introduce our two proposed novel autoregressive models to estimate the conditional distribution of input covariates  $x \in \mathbb{R}^d$ . Later, we show how to use such models over a transformation  $z = q(x)$ , while renormalizing to obtain density values for  $x$ .

### 8.2.1 Autoregressive Models

Autoregressive models decompose density estimation of a multivariate variable  $x \in \mathbb{R}^d$  into multiple conditional tasks on a growing set of inputs through the chain rule:

$$p(x_1, \dots, x_d) = \prod_{i=1}^d p(x_i | x_{i-1}, \dots, x_1). \quad (8.1)$$

That is, autoregressive models will look to estimate the  $d$  conditional distributions  $p(x_i|x_{i-1}, \dots)$ . A particular class of autoregressive models can be defined by approximating conditional distributions through a mixture model,  $\text{MM}(\theta(x_{i-1}, \dots, x_1))$ , with parameters depending on  $x_{i-1}, \dots, x_1$ :

$$p(x_i|x_{i-1}, \dots, x_1) = p(x_i | \text{MM}(\theta(x_{i-1}, \dots, x_1))), \quad (8.2)$$

$$\theta(x_{i-1}, \dots, x_1) = f(h_i) \quad (8.3)$$

$$h_i = g_i(x_{i-1}, \dots, x_1), \quad (8.4)$$

where  $f(\cdot)$  is a fully connected network that may use an element-wise non-linearity on inputs, and  $g_i(\cdot)$  is some general mapping that computes a hidden state of features,  $h_i \in \mathbb{R}^p$ , which help in modeling the conditional distribution of  $x_i | x_{i-1}, \dots, x_1$ . One can control the flexibility of the model through  $g_i$ . It is important to be powerful enough to model our covariates while still generalizing. In order to achieve this we propose two methods for modeling  $g_i$ .

**Linear Autoregressive Model (LAM):** This uses a straightforward linear map as  $g_i$  in (8.4):

$$g_i(x_{i-1}, \dots, x_1) = W^{(i)}x_{<i} + b, \quad (8.5)$$

where  $W^{(i)} \in \mathbb{R}^{p \times (i-1)}$ ,  $b \in \mathbb{R}^p$ , and  $x_{<i} = (x_{i-1}, \dots, x_1)^T$ . Notwithstanding the simple form of (8.5), the resulting model is quite flexible as it may model consecutive conditional problems  $p(x_i|x_{i-1}, \dots, x_1)$  and  $p(x_{i+1}|x_i, \dots, x_1)$  very differently owing to different  $W^{(i)}$ s.

**Recurrent Autoregressive Model (RAM):** This features a recurrent relation between  $g_i$ 's. As the set of covariates is progressively fed into  $g_i$ 's, it is natural to consider a hidden state evolving according to RNN recurrence relationship:

$$h_i = g(x_{i-1}, g(x_{i-2}, \dots, x_1)) = g(x_{i-1}, h_{i-1}). \quad (8.6)$$

In this case  $g(x, h)$  is a RNN function for updating one's state based on an input  $x$  and prior state  $h$ . In the case of gated-RNNs, the model will be able to scan through previously seen dimensions remembering and forgetting information as needed for conditional densities without making any strong Markovian assumptions.

Both LAM and RAM are flexible and able to adjust the hidden states,  $h_i$  in (8.4), to model the distinct conditional tasks  $p(x_i|x_{i-1}, \dots)$ . However, there is a trade-off of added flexibility and transferred information between the two models (see Figure 8.3). LAM treats the conditional tasks for  $p(x_i|x_{i-1}, \dots)$  and  $p(x_{i+1}|x_i, \dots)$  in a largely independent fashion. This makes for a very flexible model, however the parameter size is also large and there is no sharing of information among the conditional tasks. On the other hand, RAM provides a framework for transfer learning among the conditional tasks by allowing the hidden state  $h_i$  to evolve through the distinct conditional tasks. This leads to fewer parameters and more sharing of information in respective tasks, but also yields less flexibility since conditional estimates are tied, and may only change in a smooth fashion.

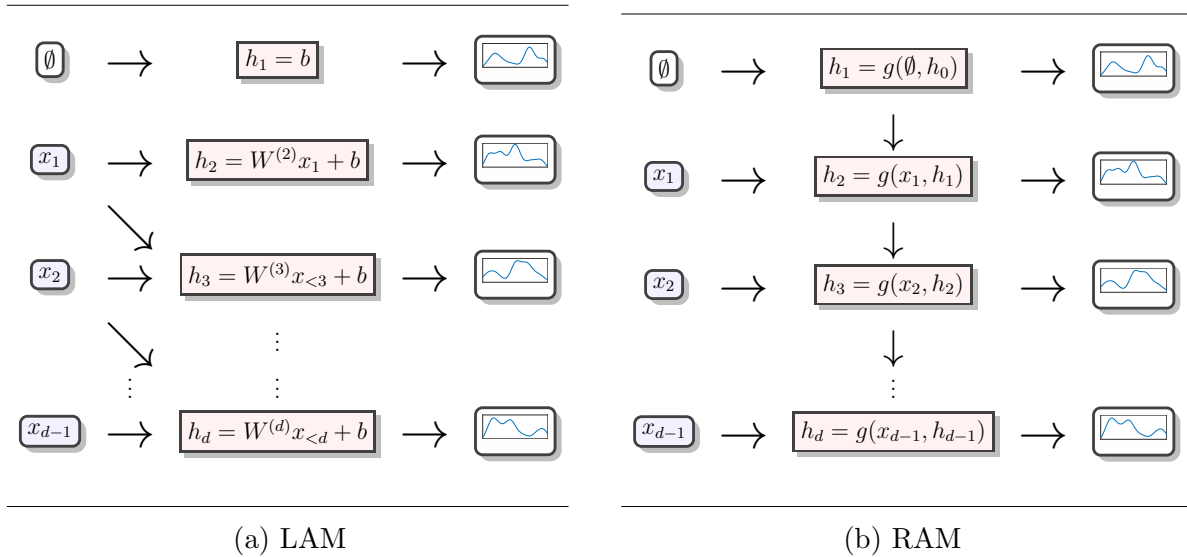


Figure 8.3: Illustration of both LAM (left) and RAM (right) models. Hidden states  $h_k$ 's are updated and then used to compute the parameters of the next conditional density for  $x_k$ . Note that in LAM the hidden states  $h_j$ 's are not tied together, where in RAM the hidden state  $h_j$  along with  $x_j$  are used to compute the hidden state  $h_{j+1}$  which determines the parameters of  $p(x_{j+1} | h_{j+1})$ .

## 8.2.2 Transformations

Next we introduce the second module of TANs, the transformations. When using an invertible transformation of variables  $z = (z_1(x), \dots, z_d(x)) \in \mathbb{R}^d$ , one can establish relationship between the pdf of  $x$  and  $z$  as:

$$p(x_1, \dots, x_d) = \left| \det \frac{dq}{dx} \right| \prod_{i=1}^d p(z_i | z_{i-1}, \dots, z_1), \quad (8.7)$$

where  $\left| \det \frac{dq}{dx} \right|$  is the Jacobian of the transformation. For analytical and computational considerations, we require transformations to be invertible, efficient to compute and invert, and have a structured Jacobian matrix. In order to meet these criteria we consider the following transformations.

**Linear Transformation:** It is an affine map of the form:

$$z = Ax + b, \quad (8.8)$$

where we take  $A$  to be invertible. Note that even though this linear transformation is simple, it includes permutations, and may also perform a PCA-like transformation, capturing coarse and highly varied features of the data before moving to more fine grained details. In order to not incur a high cost for updates, we wish to compute the determinant of the Jacobian

efficiently. Thus, we propose to directly work over an LU decomposition  $A = LU$  where  $L$  is a lower triangular matrix with unit diagonals and  $U$  is a upper triangular matrix with arbitrary diagonals. As a function of  $L, U$  we have that  $\det \frac{dz}{dx} = \prod_{i=1}^d U_{ii}$ ; hence we may efficiently optimize the parameters of the linear map. Furthermore, inverting our mapping is also efficient through solving two triangular matrix equations.

**Recurrent Transformation:** Recurrent neural networks are also a natural choice for variable transformations. Due to their dependence on only previously seen dimensions, RNN transformations have triangular Jacobians, leading to simple determinants. Furthermore, with an invertible output unit, their inversion is also straight-forward. We consider the following form to an RNN transformation:

$$z_i = r_\alpha (yx_i + w^T s_{i-1} + b), \quad s_i = r (ux_i + v^T s_{i-1} + a), \quad (8.9)$$

where  $r_\alpha$  is a leaky ReLU unit  $r_\alpha(t) = \mathbb{I}\{t < 0\}\alpha t + \mathbb{I}\{t \geq 0\}t$ ,  $r$  is a standard ReLU unit,  $s \in \mathbb{R}^p$  is the hidden state  $y, u, b, a$  are scalars, and  $w, v \in \mathbb{R}^p$  are vectors. As compared to the linear transformation, the recurrent transformation is able to transform the input with different dynamics depending on its values. Inverting (8.9) is a matter of inverting outputs and updating the hidden state (where the initial state  $s_0$  is known and constant):

$$x_i = \frac{1}{y} \left( r_\alpha^{-1} \left( z_i^{(r)} \right) - w^T s_{i-1} - b \right), \quad s_i = r (ux_i + v^T s_{i-1} + a). \quad (8.10)$$

Furthermore, the determinant of the Jacobian for (8.9) is the product of diagonal terms:

$$\det \frac{dz}{dx} = y^d \prod_{i=1}^d r'_\alpha (yx_i + w^T s_{i-1} + b), \quad (8.11)$$

where  $r'_\alpha(t) = \mathbb{I}\{t > 0\} + \alpha \mathbb{I}\{t < 0\}$ . For added dependence among all dimensions, we consider both forward passes of RNN transformations (8.9) as well as RNN transformations in a backwards pass.

**Recurrent Shift Transformation:** It is worth noting that the rescaling brought on by the recurrent transformation effectively incurs a penalty through the log of the determinant (8.11). However, one can still perform a transformation that depends on the values of covariates through a shift operation. In particular, we propose an additive shift based on a recurrent function on prior dimensions:

$$z_i = x_i + m(s_{i-1}), \quad s_i = g(x_i, s_{i-1}), \quad (8.12)$$

where  $g$  is recurrent function for updating states, and  $m$  is a fully connected network. Inversion proceeds as before:

$$x_i = z_i - m(s_{i-1}), \quad s_i = g(x_i, s_{i-1}). \quad (8.13)$$

The Jacobian is again lower triangular, however due to the additive nature of (8.12), we have a unit diagonal. Thus,  $\det \frac{dz}{dx} = 1$ . One interpretation of this transformation is that one can shift the value of  $x_k$  based on  $x_{k-1}, x_{k-2}, \dots$  for better conditional density estimation without any penalty coming from the determinant term in (8.7).

**Composing Transformations:** Lastly, we consider stacking (i.e. composing) several transformations  $q = q^{(1)} \circ \dots \circ q^{(T)}$  and renormalizing:

$$p(x_1, \dots, x_d) = \prod_{t=1}^T \left| \det \frac{dq^{(t)}}{dq^{(t-1)}} \right| \prod_{i=1}^d p(q_i(x) | q_{i-1}(x), \dots, q_1(x)), \quad (8.14)$$

where we take  $q^{(0)}$  to be  $x$ . We note that composing several transformations together allows one to leverage the respective strengths of each transformation. Moreover, inserting a reversal mapping  $(x_1, \dots, x_d \mapsto x_d, \dots, x_1)$  as one of the  $q_i$ s yields bidirectional relationships.

### 8.2.3 Combined Approach, TANs

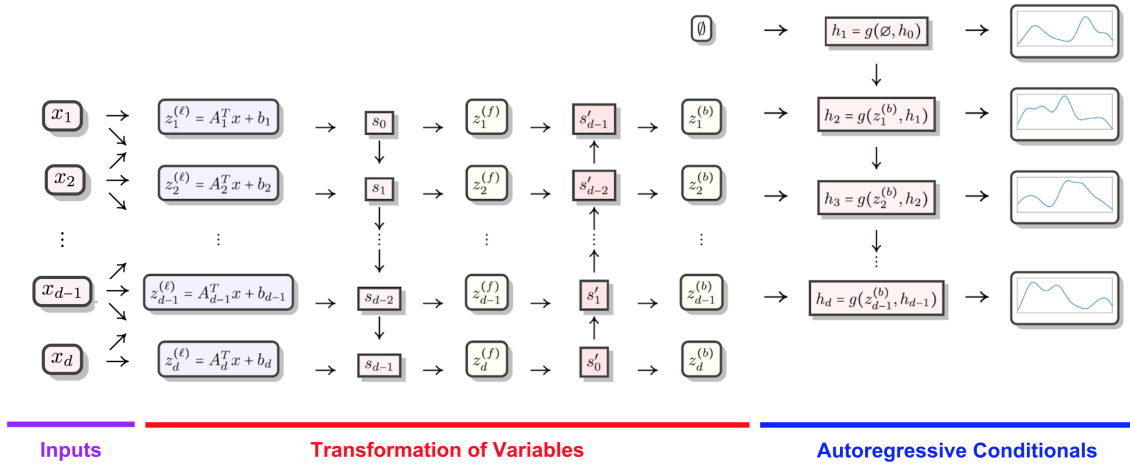


Figure 8.4: An example TAN model that employs both: a) a transformation of variables to transform covariates into a more adept space for modeling, and b) an autoregressive model that models the conditionals of the transformed covariates. For instance, we may combine an initial linear transformation of covariates  $z$  with a forward and backwards pass of RNN transformations. Then, we can model the resulting transformed covariates with an RNN autoregressive model (RAM). The entire network (transformation and autoregressive conditionals) may be optimized end-to-end via the log likelihood (8.15).

We combine the use of both transformations of variables and rich autoregressive models (Figure 8.4) by: 1) writing the density of inputs,  $p(x)$ , as a normalized density of a transformation:  $p(q(x))$  (8.14). Then we estimate the conditionals of  $p(q(x))$  using an autoregressive model, i.e. , to learn our model we minimize the negative log likelihood:

$$-\log p(x_1, \dots, x_d) = - \sum_{t=1}^T \log \left| \det \frac{dq^{(t)}}{dq^{(t-1)}} \right| - \sum_{i=1}^d \log p(q_i(x) | h_i), \quad (8.15)$$

which is obtained by substituting (8.2) into (8.14) with  $h_i$  as defined in (8.4).

## 8.3 Related Work

Nonparametric density estimation has been a well studied problem in statistics and machine learning [135]. Unfortunately, nonparametric approaches like kernel density estimation suffer greatly from the curse of dimensionality and do not perform well when data does not have a small number of dimensions ( $d \lesssim 3$ ). To alleviate this, several semiparametric approaches have been explored. Such approaches include forest density estimation [70], which assumes that the data has a forest (i.e. a collection of trees) structured graph. This assumption leads to a density which factorizes in a first order Markovian fashion through a tree traversal of the graph. Another common semiparametric approach is to use a nonparanormal type model [69]. This approach uses a Gaussian copula with a rank-based transformation and a sparse precision matrix. While both approaches are well-understood theoretically, their strong assumptions lead to inflexible models.

In order to provide greater flexibility with semiparametric models, recent work has employed deep learning for density estimation. The use of neural networks for density estimation dates back to early work by Bishop [12] and has seen success in areas like speech [143, 123], music [14], etc. Typically such approaches use a network to learn the parameters of a parametric model for data. Recent work has also explored the application of deep learning to build density estimates in image data [97, 23]. However, such approaches are heavily reliant on exploiting structure in neighboring pixels, often subsampling, reshaping or re-ordering data, and using convolutions to take advantage of neighboring correlations. Modern approaches for general density estimation in real-valued data include [126, 124, 37, 43, 22, 56, 98].

NADE [126] is an RBM-inspired density estimator with a weight-sharing scheme across conditional densities on covariates. It may be written as a special case of LAM (8.5) with tied weights:

$$q_i(x_{i-1}, \dots, x_1) = W_{<i} x_{<i} + b, \quad (8.16)$$

where  $W_{<i} \in \mathbb{R}^{p \times i-1}$  is the weight matrix composed of the first  $i-1$  columns of a shared matrix  $W = (w_1, \dots, w_d)$ . We note also that LAM and NADE are both related to fully visible sigmoid belief networks [31, 86].

Even though the weight-sharing scheme in (8.16) reduces the number of parameters, it also greatly limits the types of distributions one can model. Roughly speaking, the NADE weight-sharing scheme makes it difficult to adjust conditional distributions when expanding the conditioning set with a covariate that has a small information gain. We illustrate these kinds of limitations with a simple example. Consider the following 3-dimensional distribution:

$$x_1 \sim \mathcal{N}(0, 1), \quad x_2 \sim \mathcal{N}(\text{sign}(x_1), \epsilon), \quad x_3 \sim \mathcal{N}(\mathbb{I}\{|x_1| < C_{0.5}\}, \epsilon), \quad (8.17)$$

where  $C_{0.5}$  is the 50% confidence interval of a standard Gaussian distribution, and  $\epsilon > 0$  is some small constant. That is,  $x_2$ , and  $x_3$  are marginally distributed as an equi-weighted bimodal mixture of Gaussian with means  $-1, 1$  and  $0, 1$ , respectively. Using (8.16) we see that the states determining the distributions of  $x_2$  and  $x_3$  are  $h_2 = w_1 x_1 + b$  and  $h_3 = w_1 x_1 + w_2 x_2 + b$ , respectively. That is, the difference in the distribution of  $x_3$



from that of  $x_2$  must be captured entirely by  $w_2x_2$ . However, the sign of  $x_1$  will not be informative of its magnitude, hence weight-sharing will prohibit one from correctly modeling  $x_3$ , notwithstanding the fact that it is conditioned on  $x_1$ . I.e. the conditional model for  $x_2$  will suffer if we try to model  $x_3$  better and vice-versa. Due to NADE’s weight-sharing linear model, it will be difficult to adjust  $h_2$  and  $h_3$  jointly to correctly model  $x_2$  and  $x_3$  respectively. However, given their additional flexibility, both LAM and RAM are able to adjust hidden states to remember and transform features as needed.

In a broader sense, the NADE weight-sharing scheme makes it difficult to adjust conditional distributions when expanding the conditioning set with a covariate that has a small information gain. However, both of our proposed models do not suffer from this issue. Revisiting the distribution in (8.17), we see that our simple linear model (8.5) will model  $x_2$  using  $h_2 = w_1^{(2)}x_1 + b$  and  $x_3$  with  $h_3 = w_1^{(3)}x_1 + w_2^{(3)}x_2 + b$ . Since  $w_1^{(3)}$  and  $w_1^{(2)}$  are not tied, they may be fitted to each respective task (along with  $w_2^{(3)}$ ). Similarly, in the RNN model (8.6), we have  $h_2 = g(x_1, h_1)$  and  $h_3 = g(x_2, h_2)$ . Given an expressive enough recurrent unit and a large enough state size,  $h_2$ , which was created through  $x_1$ , may have relevant statistics for modeling  $x_3$ . Under an appropriate recurrent relation, these statistic can be remembered through (and transformed as needed) when computing  $h_3$ .

NICE [22] and its successor Real NVP [23] models assume that data is drawn from a latent independent Gaussian space and transformed. The transformation uses several “additive coupling” shifting on the second half of dimensions, using the first half of dimensions. For example NICE’s additive coupling proceeds by splitting inputs into halves  $x = (x_{<d/2}, x_{\geq d/2})$ , and transforming the second half as an additive function of the first half:

$$z = (x_{<d/2}, x_{\geq d/2} + m(x_{<d/2})), \tag{8.18}$$

where  $m(\cdot)$  is the output of a fully connected network. Inversion is simply a matter of subtraction  $x = (z_{<d/2}, z_{\geq d/2} - m(z_{<d/2}))$ . The full transformation is the result of stacking several of these additive coupling layers together followed by a final rescaling operation. Furthermore, as with the RNN shift transformation, the additive nature of (8.18) yields a simple determinant,  $\det \frac{dz}{dx} = 1$ .

MAF [98] identified that Gaussian conditional autoregressive models for density estimation can be seen as transformations. This enabled them to stack multiple autoregressive models that increases flexibility. However, stacking Gaussian conditional autoregressive models amounts to just stacking deterministic shift and scale transformations. Unlike MAF, in the TAN framework we not only propose novel and more complex equivalence like the recurrent transformation (Section 8.2.2), but also systematically composing stacks of such transformations with flexible autoregressive models.

We also note that are several methods for obtaining samples from an unknown distribution that by-pass density estimation. For instance, generative adversarial networks (GANs) apply a (typically noninvertible) transformation of variables to a base distribution by optimizing a minimax loss [38, 56]. Furthermore, one can also obtain samples with only limited information about the density of interest. For example, if one has an unnormalized pdf, one may use Markov chain Monte Carlo (MCMC) methods to obtain samples [87].

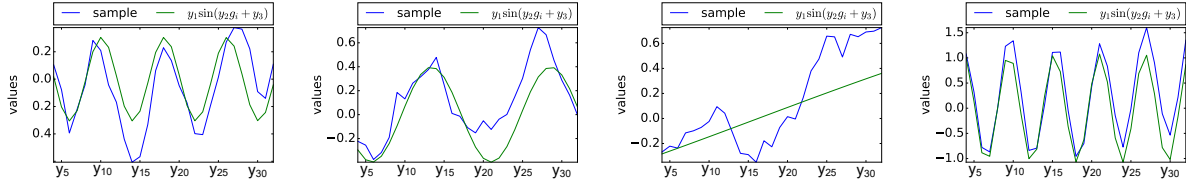


Figure 8.5: RNN+4xSRNN+Re & RAM model samples. Each plot shows a single sample. We plot the sample values of unpermuted dimensions  $y_4, \dots, y_{32} | y_1, y_2, y_3$  in blue and the expected value of these dimensions (i.e. without the Markovian noise) in green. One may see that the model is able to correctly capture both the sinusoidal and random walk behavior of our data.

## 8.4 Experiments

We now present empirical studies for our TAN framework in order to establish (i) the superiority of TANs over one-prong approaches (Section 8.4.1), (ii) that TANs are accurate on real world datasets (Section 8.4.2), (iii) the importance of various components of TANs, (iv) that TANs are easily amenable to various tasks (Section 8.4.4), such as learning a parametric family of distributions and being able to generalize over unseen parameter values (Section 8.4.5).

**Methods** We study the performance of various instantiation of TANs using different combinations of conditional models  $p(q_i(x) | h_i)$  and various transformations  $q(\cdot)$ . In particular the following conditional models were considered: **LAM**, **RAM**, **Tied**, **MultiInd**, and **SingleInd**. Here, **LAM**, **RAM**, and **Tied** are as described in equations (8.5), (8.6), and (8.16), respectively. **MultiInd** takes  $p(q_i(x) | h_i)$  to be  $p(q_i(x) | \text{MM}(\theta_i))$ , that is we shall use  $d$  distinct independent mixtures to model the transformed covariates. Similarly, **SingleInd** takes  $p(q_i(x) | h_i)$  to be  $p(q_i(x))$ , the density of a standard single component. For transformations we considered: **None**, **RNN**, **2xRNN**, **4xAdd+Re**, **4xSRNN+Re**, **RNN+4xAdd+Re**, and **RNN+4xSRNN+Re**. **None** indicates that no transformation of variables was performed. **RNN** and **2xRNN** perform a single recurrent transformation (8.9), and two recurrent transformations with a reversal permutation in between, respectively. Following [22], **4xAdd+Re** performs four additive coupling transformations (8.18) with reversal permutations in between followed by a final element-wise rescaling:  $x \mapsto x * \exp(s)$ , where  $s$  is a learned variable. Similarly, **4xSRNN+Re**, instead performs four recurrent shift transformations (8.12). **RNN+4xAdd+Re**, and **RNN+4xSRNN+Re** are as before, but performing an initial recurrent transformation. Furthermore, we also considered performing an initial linear transformation (8.8). We flag this by prepending an **L** to the transformation; e.g. **L RNN** denotes a linear transformation followed by a recurrent transformation.

**Implementation** Models were implemented in Tensorflow [2]<sup>1</sup>. Both RAM conditional models as well as the RNN shift transformation make use of the standard GRUCe11 GRU implementation. We take the mixture models of conditionals (8.2) to be mixtures of 40 Gaussians. We optimize all models using the AdamOptimizer [57] with an initial learning rate of 0.005. Training consisted of 30 000 iterations, with mini-batches of size 256. The learning rate was decreased by a factor of 0.1, or 0.5 (chosen via a validation set) every 5 000 iterations. Gradient clipping with a norm of 1 was used. After training, the best iteration according to the validation set loss was used to produce the test set results.

### 8.4.1 Synthetic

To showcase strengths of TANs and short-comings of only conditional models & only transformations, we carefully construct two synthetic datasets

**Data Generation** Our first dataset consists of a Markovian structure that features several exploitable correlations among covariates, is constructed as:  $y_1, y_2, y_3 \sim \mathcal{N}(0, 1)$  and  $y_i | y_{i-1}, \dots, y_1 \sim f(i, y_1, y_2, y_3) + \epsilon_i$  for  $i > 3$  where  $\epsilon_i \sim \mathcal{N}(\epsilon_{i-1}, \sigma)$ ,  $f(i, y_1, y_2, x_3) = y_1 \sin(y_2 g_i + y_3)$ , and  $g_i$ 's are equi-spaced points on the unit interval. That is, instances are sampled using random draws of amplitude, frequency, and shift covariates  $y_1, y_2, y_3$ , which determine the mean of the other covariates,  $y_1 \sin(y_2 g_i + y_3)$ , stemming from function evaluations on a grid, and random noise  $\epsilon_i$  with a Gaussian random walk. The resulting instances contain many correlations as visualized in Fig. 8.5. To further exemplify the importance of employing conditional and transformations in tandem, we construct a second dataset with much fewer correlations. In particular, we use a star-structured graphical model where fringe nodes are very uninformative of each-other and estimating the distribution of the fringe vertices are difficult without conditioning on all the center nodes. To construct the dataset: divide the covariates into disjoint center and edge vertex sets  $C = \{1, \dots, 4\}$ ,  $V = \{5, \dots, d\}$  respectively. For center nodes  $j \in C$ ,  $y_j \sim \mathcal{N}(0, 1)$ . Then, for  $j \in V$ ,  $y_j \sim \mathcal{N}(f_j(w_j^T y_C), \sigma)$  where  $f_j$  is a fixed step function with 32 intervals,  $w_j \in \mathbb{R}^4$  is a fixed vector, and  $y_C = (y_1, y_2, y_3, y_4)$ . In both datasets, to test robustness to correlations from distant (by index) covariates, we observe covariates that are shuffled using a fixed permutation  $\pi$  chosen ahead of time:  $x = (y_{\pi_1}, \dots, y_{\pi_d})$ . We take  $d = 32$ , and the number of training instances to be 100 000.

**Observations** We detail the mean log-likelihoods on a test set for TANs using various combinations of conditional models and transformations in Tab. 8.2 and Tab. 8.3 respectively. We see that both LAM and RAM conditionals are providing most of the top models. We observe good samples from the best performing model as shown in Fig. 8.5. Particularly in second dataset, simpler conditional methods are unable to model the data well, suggesting that the complicated dependencies need a two-prong TAN approach. We observe a similar pattern when learning over the star data with  $d = 128$  (see Appendix, Tab. 8.4).

<sup>1</sup>See <https://github.com/lupalab/tan>.

Table 8.1: Average test log-likelihood comparison of TANs with baselines MADE, Real NVP, MAF as reported by [98]. For TANs the best model is picked using validation dataset and are reported here. Parenthesized numbers indicate number of transformations used. Standard errors with  $2\sigma$  are shown. Largest values per dataset are shown in **bold**.

	POWER <small>d=6; N=2,049,280</small>	GAS <small>d=8; N=1,052,065</small>	HEPMASS <small>d=21; N=525,123</small>	MINIBOONE <small>d=43; N=36,488</small>	BSDS300 <small>d=63; N=1,300,000</small>
MADE	$-3.08 \pm 0.03$	$3.56 \pm 0.04$	$-20.98 \pm 0.02$	$-15.59 \pm 0.50$	$148.85 \pm 0.28$
MADE MoG	$0.40 \pm 0.01$	$8.47 \pm 0.02$	$-15.15 \pm 0.02$	$-12.27 \pm 0.47$	$153.71 \pm 0.28$
Real NVP (5)	$-0.02 \pm 0.01$	$4.78 \pm 1.80$	$-19.62 \pm 0.02$	$-13.55 \pm 0.49$	$152.97 \pm 0.28$
Real NVP (10)	$0.17 \pm 0.01$	$8.33 \pm 0.14$	$-18.71 \pm 0.02$	$-13.84 \pm 0.52$	$153.28 \pm 1.78$
MAF (5)	$0.14 \pm 0.01$	$9.07 \pm 0.02$	$-17.70 \pm 0.02$	$-11.75 \pm 0.44$	$155.69 \pm 0.28$
MAF (10)	$0.24 \pm 0.01$	$10.08 \pm 0.02$	$-17.73 \pm 0.02$	$-12.24 \pm 0.45$	$154.93 \pm 0.28$
MAF MoG (5)	$0.30 \pm 0.01$	$9.59 \pm 0.02$	$-17.39 \pm 0.02$	$-11.68 \pm 0.44$	$156.36 \pm 0.28$
TAN	<b><math>0.48 \pm 0.01</math></b> <small>L RNN+4xAdd+Re &amp; RAM</small>	<b><math>11.19 \pm 0.02</math></b> <small>L RNN+4xSRNN+Re &amp; RAM</small>	<b><math>-15.12 \pm 0.02</math></b> <small>L RNN &amp; RAM</small>	<b><math>-11.01 \pm 0.48</math></b> <small>4xSRNN+Re &amp; RAM</small>	<b><math>157.03 \pm 0.07</math></b> <small>L RNN+4xSRNN+Re &amp; RAM</small>

## 8.4.2 Efficacy on Real World Data

We performed several real-world data experiments and compared to several state-of-the-art density estimation methods to substantiate improved performance of TAN.

**Datasets** We carefully followed [98] and code [71] to ensure that we operated over the same instances and covariates for each of the datasets considered in [98]. Specifically we performed unconditional density estimation on four datasets from UCI machine learning repository<sup>2</sup>: **power**: Containing electric power consumption in a household over 47 months. **gas**: Readings of 16 chemical sensors exposed to gas mixtures. **hepmass**: Describing Monte Carlo simulations for high energy physics experiments. **minibone**: Containing examples of electron neutrino and muon neutrino. We also used BSDS300 which were obtained from extracting random  $8 \times 8$  monochrome patches from the BSDS300 datasets of natural images [72]. These are multivariate datasets from varied set of sources meant to provide a broad picture of performance across different domains. Further, to demonstrate that our proposed models can even be used to model high dimensional data and produce coherent samples, we consider image modeling task, treating each image as a flattened vector. We consider  $28 \times 28$  grayscale images of MNIST digits and  $32 \times 32$  natural colored images of CIFAR-10. Following Dinh, Krueger, and Bengio [22], we dequantize pixel values by adding noise and rescaling to the unit interval.

**Metric** We use the average test log-likelihoods of the best TAN model selected using a validation set and compare to values reported by [98] for MADE [37], Real NVP [23], and MAF [98] methods for each dataset. For images, we use transformed version of test

<sup>2</sup><http://archive.ics.uci.edu/ml/>

log-likelihood, called bits per pixel, which is more popular. In order to calculate bits per pixel, we need to convert the densities returned by a model back to image space in the range  $[0, 255]$ , for which we use the same logit mapping provided in [98, Appendix E.2].



Figure 8.6: Samples from best TAN model.

**Observations** Tab. 8.1 and Fig. 8.7 shows our results on various multivariate datasets and images respectively, with error bars computed over 5 runs. As can be seen, our TAN models are considerably outperforming other state-of-the-art methods across all multivariate as well as image datasets, justifying our claim of utilizing both complex transformations and conditionals. Furthermore, we plot samples for MNIST case in Fig. 8.6. We see that TAN is able to capture the structure of digits with very few artifacts in samples, which is also reflected in the likelihoods.

### 8.4.3 Ablation Study

To study how different components of the models affect the log-likelihood, we perform a comprehensive ablation study across different datasets.

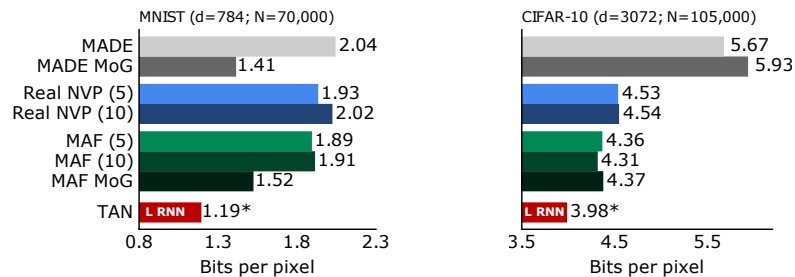


Figure 8.7: Bits per pixel for models (lower is better) using logit transforms on MNIST & CIFAR-10. MADE, Real NVP, and MAF values are as reported by [98]. The best achieved value is denoted by \*.

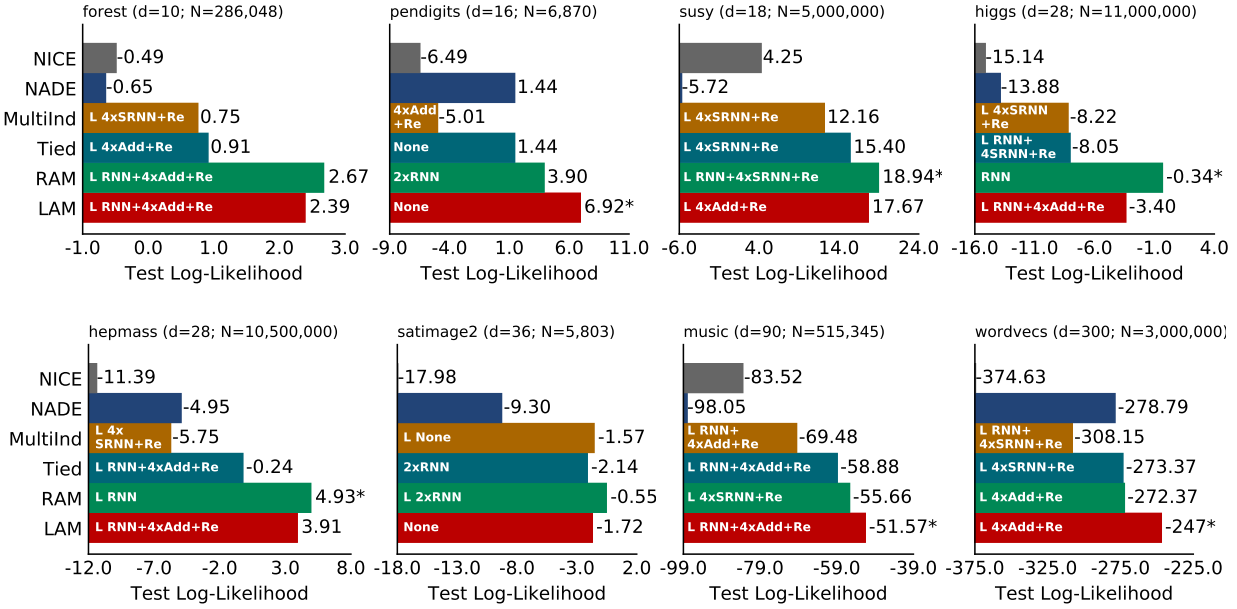


Figure 8.8: Ablation Study of various components TAN. For each dataset and each conditional model, top transformations is selected using log-likelihoods on a validation set. The picked transformation is reported within the bars for each conditional. \* denotes the best model for each dataset picked by validation. Simple conditional `MultiInd`, always lags behind sophisticated conditionals such as `LAM` & `RAM`.

**Datasets** We used multiple datasets from the UCI machine learning repository<sup>3</sup> and Stony Brook outlier detection datasets collection (ODDS)<sup>4</sup> to evaluate log-likelihoods on test data. Broadly, the datasets can be divided into: **Particle acceleration**: `higgs`, `hepmass`, and `susy` datasets where generated for high-energy physics experiments using Monte Carlo simulations; **Music**: The `music` dataset contains timbre features from the million song dataset of mostly commercial western song tracks from the year 1922 to 2011; [10]. **Word2Vec**: `wordvecs` consists of 3 million words from a Google News corpus. Each word represented as a 300 dimensional vector trained using a word2vec model<sup>5</sup>. **ODDS datasets**: We used several ODDS datasets—`forest`, `pendigits`, `satimage2`. These are multivariate datasets from varied set of sources meant to provide a broad picture of performance across anomaly detection tasks. To not penalize models for low likelihoods on outliers in ODDS, we removed anomalies from test sets when reporting log-likelihoods.

As noted in [22], data degeneracies and other corner-cases may lead to arbitrarily low negative log-likelihoods. In order to avoid such complications, we remove discrete features, standardized all datasets, and add independent Gaussian noise with a standard deviation of 0.01 to training sets.

<sup>3</sup><http://archive.ics.uci.edu/ml/>

<sup>4</sup><http://odds.cs.stonybrook.edu>

<sup>5</sup><https://code.google.com/archive/p/word2vec/>

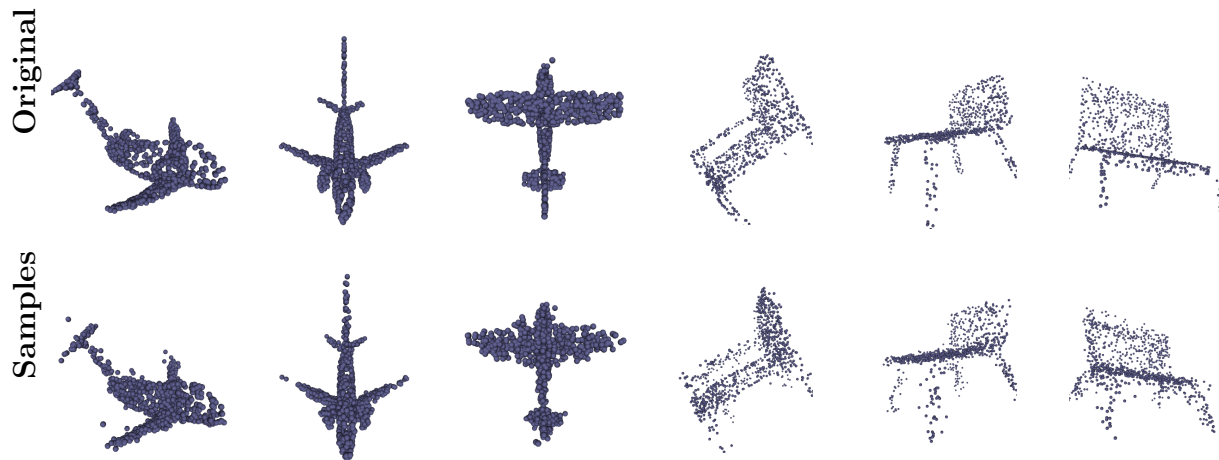


Figure 8.9: Qualitative samples obtained from TAN for task of learning parametric family of distributions where we treat each category of objects as a family and each point cloud for an object as the sample set. Top row shows unseen test point clouds and bottom row represents samples produced from TAN for these inputs. Presence of few artifacts in samples of unseen objects indicates a good fit.

**Observations** We report average test log-likelihoods in Fig. 8.8 for each dataset and conditional model for the top transformations picked on a validation dataset. The tables with test log-likelihoods for all combinations of conditional models and transformations for each dataset is in Appendix Tab. 8.6-8.12. We observe that the best performing models in real-world datasets are those that incorporate a flexible transformation *and* conditional model. In fact, the best model in each of the datasets considered always has LAM or RAM autoregressive components. Each row of these tables show that using a complex conditional is always better than using a restricted, independent conditionals. Similarly, each column of the table shows that for a given conditional, it is better to pick a complex transformation rather than having no transformation. It is interesting to note that many of these top models also contain a linear transformation. Of course, linear transformations of variables are common to most parametric models, however they have been under-explored in the context of autoregressive density estimation. Our methodology for efficiently learning linear transformations coupled with their strong empirical performance encourages their inclusion in autoregressive models for most datasets.

Finally, we want to pick the “overall” winning combination of transformation and conditional. For this we compute the fraction of the top likelihood achieved by each transformation  $t$  and conditional model  $m$  for dataset  $D$ :  $s(t, m, D) = \exp(l_{t,m,D}) / \max_{a,b} \exp(l_{a,b,D})$ , where  $l_{t,m,D}$  is the test log-likelihood for  $t, m$  on  $D$ . We then average  $S$  over the datasets:  $S(t, m) = \frac{1}{T} \sum_D S(t, m, D)$ , where  $T$  is the total number of datasets and reported all these score in Appendix Tab. 8.5. This provides a summary of which models performed better over multiple datasets. In other words, the closer this score is to 1 for a model means the more datasets for which the model is the best performer. We see that RAM conditional with L RNN transformation, and LAM conditional with L RNN+4xAdd+Re were the two best

performers.

### 8.4.4 Anomaly Detection

Next, we apply density estimates to anomaly detection. Typically anomalies or outliers are data-points that are unlikely given a dataset. In terms of density estimations, such a task is framed by identifying which instances in a dataset have a low corresponding density. That is, we shall label an instance  $x$ , as an anomaly if  $\hat{p}(x) \leq t$ , where  $t \geq 0$  is some threshold and  $\hat{p}$  is the density estimate based on training data. Note that this approach is trained in an unsupervised fashion. Density estimates were evaluated on test data with anomaly/non-anomaly labels on instances. We used thresholded log-likelihoods on the test set to compute precision and recall. We use the average-precision metric and show our results in Fig. 8.10. TAN performs the best on all three datasets. Beyond providing another interesting use for our density estimates, seeing good performance in these outlier detection tasks further demonstrates that our models are learning semantically meaningful patterns.

### 8.4.5 Learning Parametric Family of Distributions

To further demonstrate flexibility of TANs, we consider a new task of learning parametric family of distributions together. Suppose we have a family of density  $\mathcal{P}_\theta$ . We assume in training data there are  $N$  sets  $X_1, \dots, X_N$ , where the  $n$ -th set  $X_n = \{x_{n,1}, \dots, x_{n,m_n}\}$  consists of  $m_n$  i.i.d. samples from density  $\mathcal{P}_{\theta_n}$ , i.e.  $X_n$  is a set of sample points, and  $x_{n,j} \sim \mathcal{P}_{\theta_n}, j = 1, \dots, m_n$ . We assume that we do not have access to underlying true parameters  $\theta_n$ . We want to jointly learn the density estimate and parameterization of the sets to predict even for sets coming from unseen values of  $\theta$ .

We achieve this with a novel approach that models each set  $X_i$  with  $p(\cdot|\phi(X_i))$  where  $p$  is a shared TAN model for the family of distributions and  $\phi(X_i)$  are a learned embedding (parameters) for the  $i$ th set with DeepSets [141]. In particular, we use a permutation invariant network of DeepSets parameterized by  $W_1$  to extract the embedding  $\phi(X)$  for the given sample set  $X$ . The embedding is then fed along with sample set to TAN model

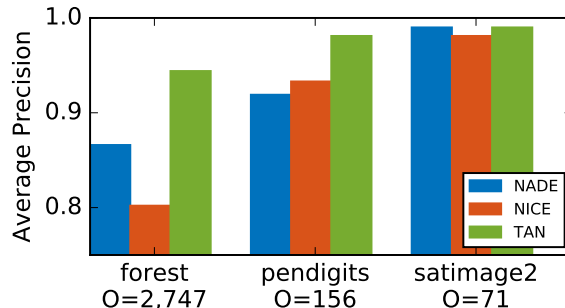


Figure 8.10: Average precision score on outlier detection datasets. For each dataset, the best performing TAN model picked using likelihood on a validation set, is shown.



parameterized by  $W_2$ . We then optimize the following modified objective:

$$\min_{W_1, W_2} -\frac{1}{N} \sum_i \frac{1}{m_i} \sum_j \log p_{W_2}(x_{ij} | \phi_{W_1}(X_{i \setminus j})). \quad (8.19)$$

We attempt to model point-cloud representation of objects from ShapeNet [18]. We produce point-clouds with 1000 particles each ( $x, y, z$ -coordinates) from the mesh representation of objects using the point-cloud-library’s sampling routine [108]. We consider each category of objects (e.g. aeroplane, chair, car) as a family and each point cloud for each object in the category as a sample set. We train a TAN and only show samples in Fig. 8.9 produced for unseen test sets, as there are neither any baselines for this task nor ground truth for likelihood. From the samples, we see that our model is able to capture the structure of different kinds of unseen aeroplanes and chairs, with very few artifacts in samples, which reflects a good fit.

Note that this task is subtly different from conditional density estimation as we do not have access to class/parameter values during training. Also we want to caution users against using this method when the test sample set is very different from training or comes from a different family distribution.

## 8.5 Discussion

We begin by noting the breadth of our proposed methods. As mentioned above, previous approaches considered a complex conditional model with a simple or no transformation and vice-versa. As such, some previous works have proposed a single new type of transformation, or a single new conditional model. Here, we propose multiple methods for transformations (linear, recurrent, and shift recurrent) and multiple autoregressive conditional models (LAM, and RAM). Furthermore, we consider the various combinations of transformations and autoregressive models, most of which constitute a novel TAN.

We draw several conclusions through our comprehensive empirical study. First, we consider our experiments on synthetic data. Methods that only consider complex transformations or condition models are illustrated in the entire row corresponding to the `None` transformation and the `MultiInd` and `SingleInd` columns, respectively. The performance of some of these models, which include `None & Tied` (NADE), `4xAdd+Re & SingleInd` (NICE), was moderate on the Markovian data, however these one-prong approaches fail in the star dataset. Overall LAM and RAM methods provided considerable improvements, especially in the star dataset, where the flexibility of LAM made it possible to learn the widely different conditional probabilities present in the data.

Similarly, we observe that the best performing models in real-world datasets are those that incorporate a flexible transformation *and* conditional model. In fact, the best model (according to validation dataset) always has LAM or RAM autoregressive components. Hence, validation across models would always select one of these methods. In fact, 95% of top-10 models (aggregated across all datasets) have a LAM and RAM conditional model (see Tables 8.6-8.12). It is interesting to see that many of these top models also contain a linear transformation. Of course, linear transformations of variables are common to most

parametric models, however they have been under-explored in the context of autoregressive density estimation. Our methodology for efficiently learning linear transformations coupled with their strong empirical performance encourages their inclusion in autoregressive models.

Finally, we digest results over the real-world datasets by computing the percentage of the top likelihood achieved by each transformation  $t$ , and conditional model  $m$ , in dataset  $D$ :  $s(t, m, D) = \exp(l_{t,m,D}) / \max_{a,b} \exp(l_{a,b,D})$ , where  $l_{t,m,D}$  is the test log-likelihood for  $t, m$  on  $D$ . We then average  $S$  over the datasets:  $S(t, m) = \frac{1}{T} \sum_D S(t, m, D)$ , where  $T$  is the total number of datasets. We show this score in the Appendix, Table 8.5. This table gives a summary of which models performed better (closer to the best performing model per dataset) over multiple datasets. We see that **RAM** conditional with **L RNN** transformation, and **LAM** conditional with **L RNN+4xAdd+Re** were the two best performers.

## 8.6 Conclusion

In conclusion, this work jointly leverages transformations of variables and autoregressive models, and proposes novel methods for both. We show a considerable improvement with our methods through a comprehensive study over both real world and synthetic data. Also, we illustrate the utility of our models in outlier detection and digit modeling tasks.

# Appendix

Table 8.2: Held out test log-likelihoods for the Markovian dataset. The superscripts denote rankings of log-likelihoods on the validation dataset.

Transformation	LAM	RAM	TIED	MultiInd	SingleInd
None	14.319	-29.950	-0.612	-41.472	- - -
L None	<b>15.486<sup>(9)</sup></b>	14.538	10.906	5.252	-9.426
RNN	14.777	-37.716	11.075	-30.491	-37.038
L RNN	<b>15.658<sup>(5)</sup></b>	10.354	10.910	5.370	3.310
2xRNN	14.683	13.698	11.493	-18.448	-34.268
L 2xRNN	<b>15.474<sup>(8)</sup></b>	<b>15.752<sup>(3)</sup></b>	12.316	5.385	3.739
4xAdd+Re	15.269	12.257	12.912	12.446	11.625
L 4xAdd+Re	<b>15.683<sup>(6)</sup></b>	12.594	13.845	12.768	12.069
4xSRNN+Re	14.829	14.381	11.798	11.738	12.932
L 4xSRNN+Re	15.289	<b>16.202<sup>(1)</sup></b>	12.748	<b>15.415<sup>(10)</sup></b>	13.908
RNN+4xAdd+Re	15.171	12.991	14.455	11.467	10.382
L RNN+4xAdd+Re	15.078	12.655	14.415	12.886	12.315
RNN+4xSRNN+Re	14.968	<b>16.216<sup>(2)</sup></b>	12.590	<b>15.589<sup>(4)</sup></b>	14.231
L RNN+4xSRNN+Re	15.429	<b>15.566<sup>(7)</sup></b>	14.179	14.528	13.961

Table 8.3: Held out test log-likelihoods for star 32d dataset. The superscript denotes ranking of log-likelihood on cross validation dataset

Transformation	LAM FC	RAM FC	TIED FC	MultiInd	SingleInd
None	-2.041	2.554	-10.454	-29.485	- - -
L None	5.454	8.247	-7.858	-26.988	-38.952
RNN	-1.276	2.762	-6.292	-25.946	-41.275
L RNN	7.775	6.335	-1.157	-25.986	-34.408
2xRNN	3.705	8.032	-0.565	-25.100	-38.490
L 2xRNN	<b>14.878<sup>(3)</sup></b>	9.946	0.901	-23.772	-33.075
4xAdd+Re	<b>13.278<sup>(6)</sup></b>	<b>11.561<sup>(9)</sup></b>	7.146	-16.740	-21.332
L 4xAdd+Re	<b>15.728<sup>(2)</sup></b>	<b>12.444<sup>(7)</sup></b>	9.031	-6.091	-11.225
4xSRNN+Re	3.496	8.429	-1.380	-15.590	-23.712
L 4xSRNN+Re	<b>16.042<sup>(1)</sup></b>	<b>9.939<sup>(10)</sup></b>	5.598	-12.530	-16.889
RNN+4xAdd+Re	<b>14.071<sup>(5)</sup></b>	<b>14.123<sup>(4)</sup></b>	6.868	-14.773	-20.483
L RNN+4xAdd+Re	<b>11.819<sup>(8)</sup></b>	9.253	2.638	-7.662	-14.530
RNN+4xSRNN+Re	-0.679	3.320	-6.172	-12.879	-19.204
L RNN+4xSRNN+Re	7.433	7.324	3.554	-10.427	-15.243

Table 8.4: Held out test log-likelihood for Star 128d dataset. The superscript denotes ranking of log-likelihood on crossvalidation dataset

Transformation	LAM FC	RAM FC	TIED FC	MultiInd	SingleInd
None	15.671	15.895	-83.115	-128.238	- - -
L None	57.881	-82.100	-28.206	-123.939	-159.391
RNN	18.766	48.295	-22.485	-113.181	-178.641
L RNN	<b>66.070<sup>(9)</sup></b>	-49.084	31.136	-107.083	-155.324
2xRNN	27.295	45.834	-11.930	-113.210	-178.331
L 2xRNN	<b>85.681<sup>(3)</sup></b>	-84.524	30.974	-105.368	-162.635
4xAdd+Re	<b>77.195<sup>(6)</sup></b>	<b>61.947<sup>(10)</sup></b>	16.062	-75.206	-111.542
L 4xAdd+Re	<b>88.837<sup>(1)</sup></b>	-21.882	20.234	-65.694	-96.071
4xSRNN+Re	33.577	-98.796	3.256	-88.912	-98.936
L 4xSRNN+Re	<b>86.375<sup>(2)</sup></b>	<b>76.968<sup>(5)</sup></b>	33.481	-85.590	-93.086
RNN+4xAdd+Re	<b>66.540<sup>(8)</sup></b>	-57.861	-16.277	-75.491	-114.729
L RNN+4xAdd+Re	<b>80.063<sup>(4)</sup></b>	32.104	21.944	-71.933	-100.384
RNN+4xSRNN+Re	21.719	-87.335	-6.517	-76.459	-85.422
L RNN+4xSRNN+Re	<b>72.463<sup>(7)</sup></b>	56.201	26.269	-71.843	-91.695

Table 8.5: Average performance percentage score for each model across all datasets. Note that this measure is not over a logarithmic space.

Transformation	LAM	RAM	TIED	MultiInd	SingleInd	MAX
None	0.218	0.118	0.006	0.000	0.000	0.218
L None	0.154	0.179	0.026	0.051	0.001	0.179
RNN	0.086	0.158	0.014	0.001	0.000	0.158
L RNN	0.173	<b>0.540</b>	0.014	0.040	0.013	0.540
2xRNN	0.151	0.101	0.045	0.001	0.000	0.151
L 2xRNN	0.118	0.330	0.015	0.045	0.025	0.330
4xAdd+Re	0.036	0.047	0.015	0.010	0.006	0.047
L 4xAdd+Re	0.153	0.096	0.025	0.014	0.009	0.153
4xSRNN+Re	0.086	0.051	0.031	0.010	0.008	0.086
L 4xSRNN+Re	0.109	0.143	0.023	0.021	0.018	0.143
RNN+4xAdd+Re	0.121	0.096	0.023	0.011	0.011	0.121
L RNN+4xAdd+Re	0.336	0.165	0.024	0.016	0.013	0.336
RNN+4xSRNN+Re	0.102	0.151	0.017	0.012	0.014	0.151
L RNN+4xSRNN+Re	0.211	0.288	0.024	0.018	0.016	0.288
MAX	0.336	0.540	0.045	0.051	0.025	

Table 8.6: Held out test log-likelihood for forest dataset. The superscript denotes ranking of log-likelihood on crossvalidation dataset

Transformation	LAM FC	RAM FC	TIED FC	MultiInd	SingleInd
None	0.751	-1.383	-0.653	-12.824	- - -
L None	1.910	1.834	-0.243	-7.665	-11.062
RNN	1.395	0.053	0.221	-5.130	-15.983
L RNN	<b>2.189<sup>(8)</sup></b>	1.747	-0.087	-4.001	-5.807
2xRNN	1.832	1.830	0.448	-6.162	-9.095
L 2xRNN	<b>2.240<sup>(6)</sup></b>	<b>2.432<sup>(3)</sup></b>	0.264	-3.956	-5.125
4xAdd+Re	1.106	1.430	0.420	-0.021	-0.492
L 4xAdd+Re	2.043	1.979	0.909	0.365	-0.088
4xSRNN+Re	1.178	1.428	0.187	-0.029	-0.212
L 4xSRNN+Re	<b>2.089<sup>(9)</sup></b>	<b>2.061<sup>(10)</sup></b>	0.611	0.754	0.593
RNN+4xAdd+Re	1.962	<b>2.226<sup>(7)</sup></b>	0.857	0.081	0.086
L RNN+4xAdd+Re	<b>2.389<sup>(4)</sup></b>	<b>2.672<sup>(1)</sup></b>	0.852	0.450	0.251
RNN+4xSRNN+Re	1.599	1.545	0.510	0.182	0.369
L RNN+4xSRNN+Re	<b>2.297<sup>(5)</sup></b>	<b>2.443<sup>(2)</sup></b>	0.804	0.600	0.480

Table 8.7: Held out test log-likelihood for pendigits dataset. The superscript denotes ranking of log-likelihood on crossvalidation dataset

Transformation	LAM FC	RAM FC	TIED FC	MultiInd	SingleInd
None	<b>6.923<sup>(1)</sup></b>	<b>3.911<sup>(8)</sup></b>	1.437	-14.138	- - -
L None	<b>4.104<sup>(9)</sup></b>	2.911	-2.872	-9.997	-15.617
RNN	<b>5.464<sup>(3)</sup></b>	3.273	-1.676	-10.144	-19.719
L RNN	<b>4.072<sup>(6)</sup></b>	1.398	-2.299	-10.840	-13.103
2xRNN	<b>6.376<sup>(5)</sup></b>	<b>3.896<sup>(7)</sup></b>	-4.002	-12.132	-16.576
L 2xRNN	2.987	0.871	-3.977	-10.890	-12.711
4xAdd+Re	-1.924	-3.087	-3.172	-5.010	-6.498
L 4xAdd+Re	-1.796	-1.438	-2.288	-4.951	-7.834
4xSRNN+Re	<b>5.854<sup>(2)</sup></b>	2.146	-2.827	-5.970	-7.084
L 4xSRNN+Re	3.758	-1.020	-3.370	-5.885	-12.978
RNN+4xAdd+Re	-2.357	-2.869	-2.187	-5.454	-8.053
L RNN+4xAdd+Re	-2.687	-2.103	-2.185	-4.742	-6.941
RNN+4xSRNN+Re	<b>5.207<sup>(4)</sup></b>	2.425	-2.126	-5.147	-8.859
L RNN+4xSRNN+Re	<b>3.466<sup>(10)</sup></b>	0.496	-2.761	-7.205	-13.897

Table 8.8: Held out test log-likelihood for `susy` dataset. The superscript denotes ranking of log-likelihood on crossvalidation dataset

Transformation	LAM FC	RAM FC	TIED FC	MultiInd	SingleInd
None	9.736	-14.821	-5.721	-21.369	- - -
L None	15.731	<b>16.930<sup>(8)</sup></b>	6.410	-8.846	-17.130
RNN	12.784	3.347	6.114	-18.575	-44.273
L RNN	16.381	<b>18.389<sup>(2)</sup></b>	6.772	-5.744	-11.489
2xRNN	11.052	14.362	3.595	-16.478	-33.126
L 2xRNN	14.523	<b>17.373<sup>(7)</sup></b>	10.687	-6.884	-10.420
4xAdd+Re	9.835	8.033	7.238	6.031	4.245
L 4xAdd+Re	<b>17.673<sup>(3)</sup></b>	<b>16.500<sup>(10)</sup></b>	11.613	10.941	9.034
4xSRNN+Re	8.798	13.235	1.234	6.936	3.378
L 4xSRNN+Re	14.242	<b>17.870<sup>(5)</sup></b>	15.397	12.161	13.413
RNN+4xAdd+Re	15.408	12.480	9.409	7.619	5.446
L RNN+4xAdd+Re	<b>17.474<sup>(6)</sup></b>	16.376	13.765	10.951	8.269
RNN+4xSRNN+Re	14.066	<b>17.691<sup>(4)</sup></b>	9.136	10.088	7.656
L RNN+4xSRNN+Re	<b>16.627<sup>(9)</sup></b>	<b>18.941<sup>(1)</sup></b>	13.469	12.105	12.349

Table 8.9: Held out test log-likelihood for `higgs` dataset. The superscript denotes ranking of log-likelihood on crossvalidation dataset

Transformation	LAM FC	RAM FC	TIED FC	MultiInd	SingleInd
None	-6.220	-5.848	-13.883	-25.793	- - -
L None	<b>-3.798<sup>(8)</sup></b>	-10.651	-9.084	-16.025	-36.051
RNN	-5.800	<b>-2.600<sup>(3)</sup></b>	-10.797	-25.760	-66.223
L RNN	<b>-3.975<sup>(9)</sup></b>	<b>-0.340<sup>(1)</sup></b>	-8.574	-18.607	-32.753
2xRNN	-6.456	-4.833	-9.192	-25.398	-60.040
L 2xRNN	-5.866	<b>-3.222<sup>(5)</sup></b>	-8.216	-16.083	-30.730
4xAdd+Re	-6.502	-10.491	-9.356	-13.678	-15.138
L 4xAdd+Re	-5.377	-5.611	-8.006	-12.106	-14.129
4xSRNN+Re	-7.422	-6.863	-11.033	-11.878	-12.182
L 4xSRNN+Re	-5.999	-9.329	-8.474	-8.223	-8.926
RNN+4xAdd+Re	<b>-4.242<sup>(10)</sup></b>	-4.804	-9.187	-12.321	-15.261
L RNN+4xAdd+Re	<b>-3.396<sup>(6)</sup></b>	<b>-3.049<sup>(4)</sup></b>	-8.052	-12.246	-13.765
RNN+4xSRNN+Re	-5.262	<b>-2.116<sup>(2)</sup></b>	-10.105	-12.307	-9.388
L RNN+4xSRNN+Re	<b>-3.756<sup>(7)</sup></b>	-4.773	-8.097	-9.378	-7.721

Table 8.10: Held out test log-likelihood for `hepmass` dataset. The superscript denotes ranking of log-likelihood on crossvalidation dataset

Transformation	LAM FC	RAM FC	TIED FC	MultiInd	SingleInd
None	2.328	<b>3.710<sup>(6)</sup></b>	-4.948	-19.771	- - -
L None	<b>3.570<sup>(7)</sup></b>	2.517	-4.052	-9.266	-35.042
RNN	2.088	<b>4.935<sup>(1)</sup></b>	-1.639	-19.851	-47.686
L RNN	<b>2.869<sup>(10)</sup></b>	<b>5.047<sup>(2)</sup></b>	-2.920	-16.032	-30.210
2xRNN	1.774	0.902	-1.909	-15.440	-36.754
L 2xRNN	2.053	<b>3.680<sup>(5)</sup></b>	-2.150	-15.457	-24.079
4xAdd+Re	1.678	1.873	-4.046	-9.117	-11.387
L 4xAdd+Re	1.961	2.543	-2.259	-6.907	-9.275
4xSRNN+Re	1.443	2.156	-2.904	-6.091	-7.186
L 4xSRNN+Re	2.072	2.730	-3.014	-5.747	-6.245
RNN+4xAdd+Re	2.817	0.912	-2.514	-6.003	-9.284
L RNN+4xAdd+Re	<b>3.906<sup>(3)</sup></b>	-1.869	-3.847	-6.339	-9.103
RNN+4xSRNN+Re	2.663	<b>3.586<sup>(8)</sup></b>	-0.863	-7.146	-3.939
L RNN+4xSRNN+Re	<b>3.759<sup>(4)</sup></b>	<b>3.487<sup>(9)</sup></b>	-0.239	-7.522	-6.102

Table 8.11: Held out test log-likelihood for `satimage2` dataset. The superscript denotes ranking of log-likelihood on crossvalidation dataset

Transformation	LAM FC	RAM FC	TIED FC	MultiInd	SingleInd
None	<b>-1.716<sup>(9)</sup></b>	<b>-1.257<sup>(3)</sup></b>	-9.296	-50.507	- - -
L None	-20.164	<b>-1.079<sup>(4)</sup></b>	-2.635	<b>-1.570<sup>(5)</sup></b>	-5.972
RNN	-7.728	-4.949	-5.466	-6.047	-16.521
L RNN	-31.296	<b>-0.773<sup>(2)</sup></b>	-3.944	<b>-1.824<sup>(8)</sup></b>	-2.977
2xRNN	-12.283	<b>-2.193<sup>(7)</sup></b>	-2.137	-5.447	-8.075
L 2xRNN	-20.968	<b>-0.550<sup>(1)</sup></b>	-5.140	<b>-1.699<sup>(6)</sup></b>	<b>-2.276<sup>(10)</sup></b>
4xAdd+Re	-19.931	-7.539	-11.826	-18.901	-17.977
L 4xAdd+Re	-21.128	-9.944	-12.336	-21.677	-24.070
4xSRNN+Re	-7.519	-11.368	-2.549	-7.730	-7.232
L 4xSRNN+Re	-18.170	-7.709	-5.533	-17.085	-15.347
RNN+4xAdd+Re	-19.278	-11.789	-12.837	-21.249	-22.786
L RNN+4xAdd+Re	-20.899	-12.949	-12.867	-26.164	-28.302
RNN+4xSRNN+Re	-13.476	-3.951	-6.284	-15.025	-16.443
L RNN+4xSRNN+Re	-20.179	-12.128	-7.258	-18.065	-18.125

Table 8.12: Held out test log-likelihood for music dataset. The superscript denotes ranking of log-likelihood on crossvalidation dataset

<b>Transformation</b>	<b>LAM FC</b>	<b>RAM FC</b>	<b>TIED FC</b>	<b>MultiInd</b>	<b>SingleInd</b>
None	-57.873	-97.925	-98.047	-113.099	- - -
L None	<b>-52.954<sup>(4)</sup></b>	-74.220	-72.441	-82.866	-104.287
RNN	<b>-54.933<sup>(10)</sup></b>	-80.436	-74.361	-106.219	-144.735
L RNN	<b>-52.710<sup>(3)</sup></b>	-59.815	-66.536	-82.731	-98.813
2xRNN	-56.958	-85.359	-77.456	-104.440	-133.898
L 2xRNN	<b>-53.956<sup>(8)</sup></b>	-57.611	-65.016	-82.678	-96.542
4xAdd+Re	-56.349	-69.302	-67.064	-73.886	-83.524
L 4xAdd+Re	<b>-53.169<sup>(5)</sup></b>	-59.282	-59.093	-69.887	-79.330
4xSRNN+Re	-57.670	-68.116	-74.006	-78.032	-121.197
L 4xSRNN+Re	<b>-53.879<sup>(7)</sup></b>	-55.665	-63.894	-77.564	-81.188
RNN+4xAdd+Re	<b>-53.177<sup>(6)</sup></b>	-67.377	-63.372	-73.882	-84.032
L RNN+4xAdd+Re	<b>-51.572<sup>(1)</sup></b>	-56.190	-58.885	-69.484	-79.555
RNN+4xSRNN+Re	<b>-54.065<sup>(9)</sup></b>	-61.204	-76.437	-71.814	-81.087
L RNN+4xSRNN+Re	<b>-52.617<sup>(2)</sup></b>	-68.756	-65.061	-83.292	-78.997



# Chapter 9

## Discussion

### 9.1 Conclusion

In conclusion, this thesis has made progress in the following directions:

1. *Machine learning on complex objects*, to perform classic machine learning tasks when inputs and outputs are not static, finite vectors.
2. *Leveraging random features*, to use latent distributions when learning kernels.
3. *Modeling sequences through summaries*, to use simple, data-driven statistics of past data to learn long-term temporal dependencies.
4. *High dimensional density estimation*, to produce likelihoods and samples of general, high-dimensional data.

I expound on these areas and future directions below.

**Machine Learning on Complex Objects** A lot of effort in machine learning has been devoted to studying how to perform tasks over high-dimensional inputs and outputs. But what about infinite dimensional data? General functions and distributions are infinite-dimensional since they require an infinite number of parameters in their exact representation. However, distributional data are all around us: from the sound waves we hear, to the stock prices we follow, to the distribution of our cells. Given the complexity of distributional, it would be beneficial to work over a large training set in order to achieve a low risk; however, many nonparametric methods do not scale to large datasets. In order to solve this paradox, we created the first estimators to use random features over functional inputs and/or outputs. I developed two efficient estimators, the Double Basis and Triple Basis Estimator, Sections 3.2 and 3.4, respectively. These algorithms were shown analytically and empirically to scale to datasets of millions or more instances. Furthermore, in Chapter 4 we extended the Double and Triple Basis Estimators to work over distributions using non- $L_2$ , information-theoretic distances, which improved accuracy in several applications.

**Leveraging Randomness in Features** In a novel combination of distributions and random features, we constructed a method to learn the latent distribution of random

frequencies in Chapter 6. Here we show that one may improve the performance of kernel learning tasks by learning the spectral distribution in a data-driven fashion using Bayesian nonparametric techniques. By representing the spectral density using a nonparametric mixture of Gaussians, we capture a large class of kernels that can be learned. We provide a generative model for learning kernels while performing regression and classification tasks, and propose novel MCMC based sampling schemes to infer parameters of the mixtures. We show that our proposed framework outperforms other scalable kernel learning methods on a variety of real world datasets in both classification and regression task.

**Modeling Sequences through Summaries** As the name implies, statisticians often employ summary statistics when trying to represent a dataset. It is natural then to consider an algorithm that itself learns to represent previously seen data in much the same vein as a neural statistician. To this end, we constructed the Statistical Recurrent Unit (SRU) in Chapter 7, a novel architecture that captures long-term dependencies in data using only moving averages and rectified-linear units. Of course, simply averaging statistics of sequential points will lose valuable temporal information. The SRU avoids losing sequential information in two ways: first, it generates recurrent statistics that depend on a context of previously seen data; second, it generates moving averages at several scales, allowing the model to distinguish the type of data seen at different points in the past. Empirical studies show that even though the SRU has a relatively simple architecture when compared to popular gated architectures like LSTMs and GRUs, it often retains long-term dependencies better and achieves higher accuracies.

**High Dimensional Density Estimation** Density estimation is at the core of a multitude of machine learning applications. However, this fundamental task, which encapsulates the understanding of data, is difficult in the general setting due to issues like the curse of dimensionality. Furthermore, general data, unlike spatial/temporal data, does not contain a priori known correlations among covariates that may be exploited and engineered with. For example, image data has known correlations among neighboring pixels that may be hard-coded into a model, whereas one must find such correlations in a data-driven fashion with general data. Modern advances in density estimation have either: a) proposed a flexible model to estimate the conditional factors of the chain rule, or b) used flexible, non-linear transformations of variables of a simple base distribution. Instead, Transformation Autoregressive Networks (TANs) jointly leverage transformations of variables and autoregressive conditional models. In Chapter 8 we proposed novel linear and recurrent neural network methods for both transformation and autoregressive conditional components. Comprehensive empirical studies show that the combined use of these novel components in TANs produce better models that achieve a higher held-out test likelihood than other state of the art autoregressive models.

## 9.2 Future Directions

Reflecting on the work brought forth in this thesis, there remain many open problems and avenues to further machines ability to reason holistically on complex data. Two prominent directions are: better representations of sets/distributions and graphs (Figure 9.1), and better generative models (Figure 9.2).

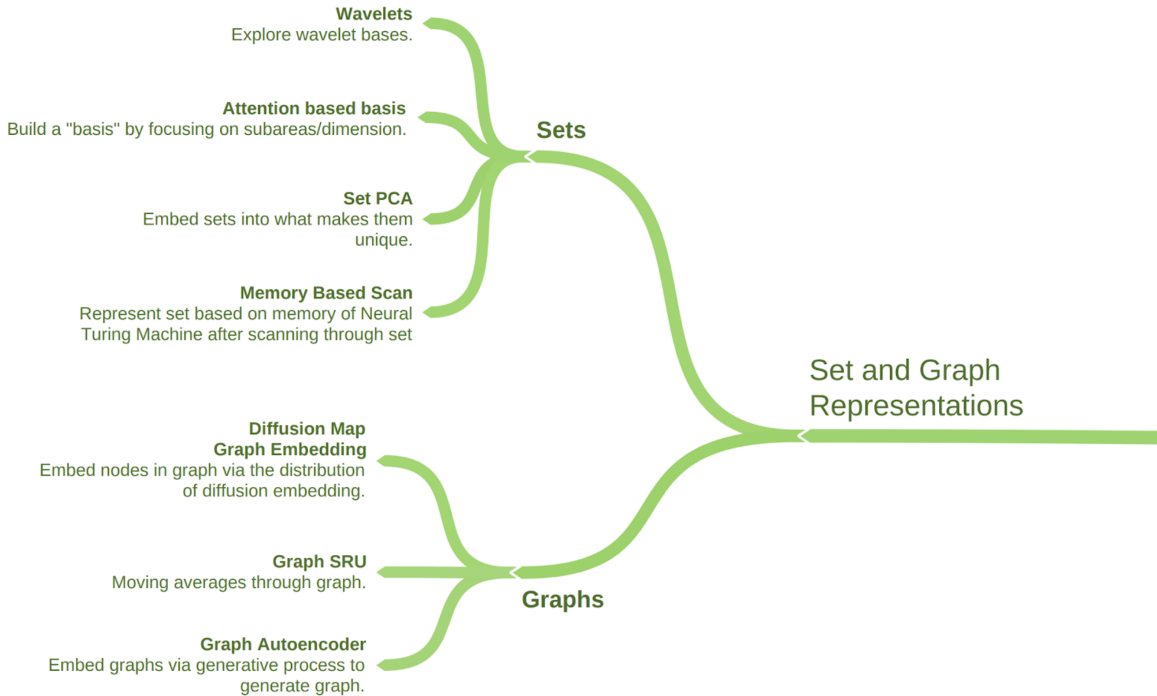


Figure 9.1: Future directions for representing sets/distributions and graphs.

**Set and Graph Representations** There are many questions to investigate with regards to how to represent sets/distributions as well as graphs for performing machine learning tasks over them (Figure 9.1). In addition to the directions discussed in Section 5.1, there are several other ways that one may look to build a “basis” for input distributions. For example, one may employ wavelet bases for representations that use fewer basis functions. However, with a wavelet approach one would have to address the resulting sparse, but large, projection coefficient vectors due to many locations/scales resulting in a negligible inner-product. One may also look to model a set of points via an attention or projection type model, which projects points to (possibly many) lower dimensional spaces where the resulting distributions are analyzed. Perhaps a good way to characterize sets would be based on a collection of classifiers that can best discern between pairs of sets in training data. Moreover, it may be possible to represent sets as the memory of a Neural Turing Machine [42], thereby learning to scan elements in sets to represent them. Furthermore, there are

also many interesting avenues for representing graphs as inputs. These include embedding graphs based on a generative, autoencoding method, or inspecting the distributions of nodes' embeddings (e.g. using diffusion maps) for each graph, or even employing a SRU like model where we propagate statistics across nodes.

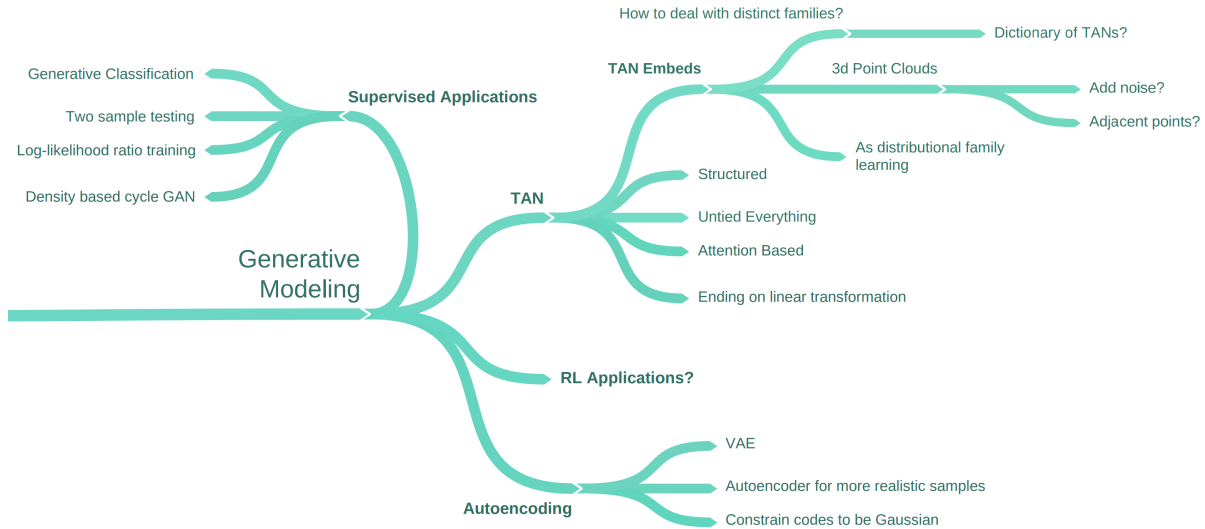


Figure 9.2: Future directions for generative modeling.

**Generative Modeling and Density Estimation** TANs and other recent generative modeling methods have shown tremendous promise for modeling high-dimensional data. Still, a lot of interesting avenues remain to be explored (Figure 9.2). One straightforward extension of TANs to explore is a conditional version to model  $x$  given extraneous data  $y$ ,  $p(x|y)$ . Besides conditional modeling, which is interesting in its own right, conditional TANs may be combined with likelihood ratio training to perform tasks like classification and two-sample testing. Moreover, we may exploit a priori known correlations by learning a conditioning scheme across structured data. For instance, we may do sequential modeling of time indexed covariates  $x_t$  by writing  $p(x_t|x_{t-1}, \dots, x_1) \approx p(x_t|h_{t-1})$ , where  $h_{t-1}$  is an RNN state updated after observing  $x_1, \dots, x_{t-1}$ . Similarly, a subsampling approach and convolutional networks may be used with conditional TANs to model natural images. In the same vein, there a lot of uncovered ground remains for TAN based embedding methods (as discussed in Section 8.4.5). For instance, even simple questions such as how to deal with data degeneracies present in point-cloud data of objects, which lie in a lower dimensional manifold, are yet to be answered. One may also explore the use of autoencoders for TANs, and vice-versa. Further applications include modeling the dynamics in reinforcement learning, and the featurization of high-dimensional distributions.

**Long-term Directions** There are also many long-term, far-reaching directions for building intelligent systems that reason holistically on data. A large component of human intelligence that remains under explored in machine learning is memory. Recent work like the Neural Turing Machine [42] has shown that one can develop a differentiable memory system. Still, it is very unclear how to best build and leverage algorithms that can carry long-term memories while navigating through collections of data points. Furthermore, truly flexible reasoning comes not only from memorizing or representing collections, but from exploiting the relations amongst multiple data points, often from very different domains. A fruitful direction is to develop algorithms that can, on the fly, examine sets or sequences of multiple distinct domains and develop connections among their contained instances for learning. Another extreme challenge is developing systems that can perform “one shot” or “few shot” learning, and are able to effectively generalize given a small number of examples. Humans are especially adept at extrapolation by using memories of collections of objects, and their corresponding relationships, hence it is imperative that we explore how to leverage these approaches in machines for better generalization.



# References

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. “TensorFlow: A system for large-scale machine learning”. In: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Savannah, Georgia, USA. 2016.
- [2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. “Tensorflow: Large-scale machine learning on heterogeneous distributed systems”. In: *arXiv preprint arXiv:1603.04467* (2016).
- [3] Hirotugu Akiake. “Information theory and an extension of the maximum likelihood principle”. In: *2nd Int. Symp. on Inf. Theory*. 1973.
- [4] Sanjeev Arora and Yi Zhang. “Do GANs actually learn the distribution? an empirical study”. In: *arXiv preprint arXiv:1706.08224* (2017).
- [5] Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. “Multiple kernel learning, conic duality, and the SMO algorithm”. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 6.
- [6] Shane Barratt and Rishi Sharma. “A Note on the Inception Score”. In: *arXiv preprint arXiv:1801.01973* (2018).
- [7] Serge Belongie, Charless Fowlkes, Fan Chung, and Jitendra Malik. “Spectral partitioning with indefinite kernels using the Nyström extension”. In: *ECCV*. 2002.
- [8] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization”. In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.
- [9] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. “Hyperopt: a python library for model selection and hyperparameter optimization”. In: *Computational Science & Discovery* 8.1 (2015), p. 014008.
- [10] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. “The Million Song Dataset.” In: *ISMIR*. Vol. 2. 2011, p. 10.
- [11] C. M. Bishop and M. E. Tipping. “Bayesian Regression and Classification”. In: *IOS Press* (2003).
- [12] Christopher M Bishop. “Mixture density networks”. In: *Technical Report* (1994).
- [13] Matthew B Blaschko and Thomas Hofmann. “Conformal multi-instance kernels”. In: *NIPS 2006 Workshop on Learning to Compare Examples*. IEEE Computer Society. 2006, pp. 1–6.

- [14] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. “Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription”. In: *International Conference on Machine Learning* (2012).
- [15] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. “Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription”. In: *arXiv preprint arXiv:1206.6392* (2012).
- [16] Marc-André Carbonneau, Veronika Cheplygina, Eric Granger, and Ghyslain Gagnon. “Multiple instance learning: A survey of problem characteristics and applications”. In: *Pattern Recognition* (2017).
- [17] Coralia Cartis, Nicholas IM Gould, and Ph L Toint. “On the complexity of steepest descent, Newton’s and regularized Newton’s methods for nonconvex unconstrained optimization problems”. In: *SIAM Journal on Optimization* 20.6 (2010), pp. 2833–2852.
- [18] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. “Shapenet: An information-rich 3d model repository”. In: *arXiv preprint arXiv:1512.03012* (2015).
- [19] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. “On the properties of neural machine translation: Encoder-decoder approaches”. In: *arXiv preprint arXiv:1409.1259* (2014).
- [20] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).
- [21] Hanjun Dai, Bo Dai, and Le Song. “Discriminative embeddings of latent variable models for structured data”. In: *International Conference on Machine Learning*. 2016, pp. 2702–2711.
- [22] Laurent Dinh, David Krueger, and Yoshua Bengio. “NICE: Non-linear independent components estimation”. In: *arXiv preprint arXiv:1410.8516* (2014).
- [23] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using Real NVP”. In: *arXiv preprint arXiv:1605.08803* (2016).
- [24] Carl Doersch. “Tutorial on variational autoencoders”. In: *arXiv preprint arXiv:1606.05908* (2016).
- [25] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. “Long-term recurrent convolutional networks for visual recognition and description”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 2625–2634.
- [26] Harrison Edwards and Amos Storkey. “Towards a Neural Statistician”. In: *arXiv preprint arXiv:1606.02185* (2016).
- [27] Jeffrey L Elman. “Finding structure in time”. In: *Cognitive science* 14.2 (1990), pp. 179–211.
- [28] Morten W Fagerland, Stian Lydersen, and Petter Laake. “The McNemar test for binary matched-pairs data: mid-p and asymptotic are better than exact conditional”. In: *BMC Medical Research Methodology* (2013).



- [29] Thomas S Ferguson. “A Bayesian Analysis of Some Nonparametric Problems”. In: *The Annals of Statistics* 1.2 (Mar. 1973), pp. 209–230.
- [30] F. Ferraty and P. Vieu. *Nonparametric functional data analysis: theory and practice*. Springer, 2006.
- [31] Brendan J Frey. *Graphical models for machine learning and digital communication*. MIT press, 1998.
- [32] Bent Fuglede. “Spirals in Hilbert space: With an application in information theory”. In: *Exposition. Math.* 23.1 (Apr. 2005), pp. 23–45.
- [33] Thomas Gärtner, Peter A Flach, Adam Kowalczyk, and Alexander J Smola. “Multi-instance kernels”. In: *ICML*. Vol. 2. 2002, pp. 179–186.
- [34] Thomas Gärtner, Peter Flach, and Stefan Wrobel. “On graph kernels: Hardness results and efficient alternatives”. In: *Learning Theory and Kernel Machines*. Springer, 2003, pp. 129–143.
- [35] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. “Convolutional sequence to sequence learning”. In: *arXiv preprint arXiv:1705.03122* (2017).
- [36] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2003.
- [37] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. “MADE: masked autoencoder for distribution estimation”. In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015, pp. 881–889.
- [38] Ian Goodfellow. “NIPS 2016 tutorial: Generative adversarial networks”. In: *arXiv preprint arXiv:1701.00160* (2016).
- [39] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative adversarial nets”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 2672–2680.
- [40] Alex Graves and Navdeep Jaitly. “Towards End-To-End Speech Recognition with Recurrent Neural Networks.” In: *ICML*. Vol. 14. 2014, pp. 1764–1772.
- [41] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech recognition with deep recurrent neural networks”. In: *Acoustics, speech and signal processing (icassp), 2013 IEEE international conference on*. IEEE. 2013, pp. 6645–6649.
- [42] Alex Graves, Greg Wayne, and Ivo Danihelka. “Neural Turing machines”. In: *arXiv preprint arXiv:1410.5401* (2014).
- [43] Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. “Deep autoregressive networks”. In: *ICML* (2014).
- [44] A Gretton, K Fukumizu, Z Harchaoui, and B K Sriperumbudur. “A fast, consistent kernel two-sample test”. In: *NIPS*. 2009.
- [45] Bernard Haasdonk and Claus Bahlmann. “Learning with Distance Substitution Kernels”. In: *Pattern Recognition: 26th DAGM Symposium*. 2004, pp. 220–227.
- [46] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. 2001.

- [47] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [48] Matthew D Hoffman and Andrew Gelman. “The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo”. In: *JMLR* 15.1 (2014), pp. 1593–1623. arXiv: 1111.4246.
- [49] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5 MB model size”. In: *arXiv preprint arXiv:1602.07360* (2016).
- [50] T. Jaakkola and D. Haussler. “Exploiting Generative Models in Discriminative Classifiers”. In: *NIPS*. 1998.
- [51] T.S. Jaakkola, D. Haussler, et al. “Exploiting generative models in discriminative classifiers”. In: *Advances in neural information processing systems* (1999), pp. 487–493.
- [52] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. “What is the best multi-stage architecture for object recognition?” In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 2146–2153.
- [53] T. Jebara, R. Kondor, and A. Howard. “Probability product kernels”. In: *The Journal of Machine Learning Research* 5 (2004), pp. 819–844.
- [54] Tony Jebara, Risi Kondor, and Andrew Howard. “Probability product kernels”. In: *JMLR* 5 (2004), pp. 819–844.
- [55] Hachem Kadri, Emmanuel Duflos, Philippe Preux, Stéphane Canu, Manuel Davy, et al. “Nonlinear functional regression: a functional RKHS approach”. In: *JMLR Workshop and Conference Proceedings*. Vol. 9. 2010, pp. 374–380.
- [56] Diederik P Kingma, Tim Salimans, and Max Welling. “Improving variational inference with inverse autoregressive flow”. In: *arXiv preprint arXiv:1606.04934* (2016).
- [57] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [58] Risi Kondor and Tony Jebara. “A Kernel Between Sets of Vectors”. In: *ICML*. 2003.
- [59] Risi Kondor, Hy Truong Son, Horace Pan, Brandon Anderson, and Shubhendu Trivedi. “Covariant Compositional Networks For Learning Graphs”. In: *arXiv preprint arXiv:1801.02144* (2018).
- [60] Akshay Krishnamurthy, Kirthevasan Kandasamy, Barnabas Poczos, and Larry Wasserman. “Nonparametric estimation of renyi divergence and friends”. In: *International Conference on Machine Learning*. 2014, pp. 919–927.
- [61] John Lafferty, Han Liu, Larry Wasserman, et al. “Sparse nonparametric graphical models”. In: *Statistical Science* 27.4 (2012), pp. 519–537.
- [62] Gert RG Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I Jordan. “Learning the kernel matrix with semidefinite programming”. In: *The Journal of Machine Learning Research* 5 (2004), pp. 27–72.
- [63] Hugo Larochelle and Iain Murray. “The neural autoregressive distribution estimator”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 29–37.

- [64] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. “Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories”. In: *CVPR*. 2006.
- [65] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. “A simple way to initialize recurrent networks of rectified linear units”. In: *arXiv preprint arXiv:1504.00941* (2015).
- [66] Quoc Le, Tamas Sarlos, and Alex Smola. “Fastfood—Approximating kernel expansions in loglinear time”. In: *Proceedings of the international conference on machine learning*. 2013.
- [67] Thomas Leung and Jitendra Malik. “Representing and Recognizing the Visual Appearance of Materials using Three-dimensional Textons”. In: *IJCV* 43 (2001).
- [68] Min Lin, Qiang Chen, and Shuicheng Yan. “Network in network”. In: *arXiv preprint arXiv:1312.4400* (2013).
- [69] Han Liu, John Lafferty, and Larry Wasserman. “The nonparanormal: Semiparametric estimation of high dimensional undirected graphs”. In: *Journal of Machine Learning Research* 10.Oct (2009), pp. 2295–2328.
- [70] Han Liu, Min Xu, Haijie Gu, Anupam Gupta, John Lafferty, and Larry Wasserman. “Forest density estimation”. In: *Journal of Machine Learning Research* 12.Mar (2011), pp. 907–951.
- [71] MAF Git Repository. *The MAF Git Repository*. <https://github.com/gpapamak/maf>. Accessed: 2017-12-03.
- [72] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. “A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics”. In: *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*. Vol. 2. IEEE. 2001, pp. 416–423.
- [73] Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc’Aurelio Ranzato. “Learning longer memory in recurrent neural networks”. In: *arXiv preprint arXiv:1412.7753* (2015).
- [74] Thomas Minka. *Estimating a Dirichlet distribution*. 2000.
- [75] Thomas Minka. *The Fastfit Matlab toolbox*.
- [76] Thomas P. Minka. *Bayesian Linear Regression*. Tech. rep. 3594 Security Ticket Control, 1999.
- [77] Todd K Moon. “The expectation-maximization algorithm”. In: *Signal processing magazine, IEEE* 13.6 (1996), pp. 47–60.
- [78] Pedro J Moreno, Purdy P Ho, and Nuno Vasconcelos. “A Kullback-Leibler divergence based kernel for SVM classification in multimedia applications”. In: *NIPS*. 2003.
- [79] Krikamol Muandet, Kenji Fukumizu, Francesco Dinuzzo, and Bernhard Schölkopf. “Learning from distributions via support measure machines”. In: *NIPS*. 2012.
- [80] Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, and Bernhard Schölkopf. “Kernel Mean Embedding of Distributions: A Review and Beyonds”. In: *arXiv preprint arXiv:1605.09522* (2016).

- [81] Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, Bernhard Schölkopf, et al. “Kernel mean embedding of distributions: A review and beyond”. In: *Foundations and Trends® in Machine Learning* 10.1-2 (2017), pp. 1–141.
- [82] Meinard Müller. *Information retrieval for music and motion*. Vol. 2. Springer, 2007.
- [83] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.
- [84] *NBA Movement Data*. <https://github.com/sealneaward/nba-movement-data>. Accessed: 2016-10-17.
- [85] R. M. Neal. *Markov chain sampling methods for Dirichlet process mixture models*. Tech. rep. 9815. Dept. of Statistics, University of Toronto, 1998.
- [86] Radford M Neal. “Connectionist learning of belief networks”. In: *Artificial intelligence* 56.1 (1992), pp. 71–113.
- [87] Radford M Neal. “Probabilistic inference using Markov chain Monte Carlo methods”. In: (1993).
- [88] Michelle Ntampaka, Hy Trac, Dougal J Sutherland, Nicholas Battaglia, Barnabás Póczos, and Jeff Schneider. “A Machine Learning Approach for Dynamical Mass Measurements of Galaxy Clusters”. In: *The Astrophysical Journal* 803.2 (Oct. 2015), p. 50. arXiv: 1410.0686.
- [89] Junier B Oliva, Avinava Dubey, Barnabás Póczos, Jeff Schneider, and Eric P Xing. “Transformation Autoregressive Networks”. In: *arXiv preprint arXiv:1801.09819* (2018).
- [90] Junier B Oliva, Avinava Dubey, Andrew G Wilson, Barnabás Póczos, Jeff Schneider, and Eric P Xing. “Bayesian nonparametric kernel-learning”. In: *Artificial Intelligence and Statistics*. 2016, pp. 1078–1086.
- [91] Junier B Oliva, Willie Neiswanger, Barnabas Poczsoz, Jeff Schneider, and Eric Xing. “Fast Distribution To Real Regression”. In: *AISTATS* (2014).
- [92] Junier B Oliva, Willie Neiswanger, Barnabás Póczos, Eric P Xing, Hy Trac, Shirley Ho, and Jeff G Schneider. “Fast Function to Function Regression.” In: *AISTATS*. 2015.
- [93] Junier B Oliva, Barnabás Póczos, and Jeff Schneider. “Distribution to distribution regression”. In: *ICML*. 2013.
- [94] Junier B Oliva, Barnabás Póczos, and Jeff Schneider. “The statistical recurrent unit”. In: *ICML* (2017).
- [95] Junier B Oliva, Barnabas Poczsoz, Timothy Verstynen, Aarti Singh, Jeff Schneider, Fang-Cheng Yeh, and Wen-Yih Tseng. “FuSSO: Functional Shrinkage and Selection Operator”. In: *AISTATS* (2014).
- [96] Junier B Oliva, Dougal J Sutherland, Barnabás Póczos, and Jeff Schneider. “Deep Mean Maps”. In: *arXiv preprint arXiv:1511.04150* (2015).
- [97] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. “Pixel recurrent neural networks”. In: *arXiv preprint arXiv:1601.06759* (2016).

- [98] George Papamakarios, Theo Pavlakou, and Iain Murray. “Masked Autoregressive Flow for Density Estimation”. In: *arXiv preprint arXiv:1705.07057* (2017).
- [99] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks.” In: *ICML (3)* 28 (2013), pp. 1310–1318.
- [100] Barnabás Póczos, Alessandro Rinaldo, Aarti Singh, and Larry Wasserman. “Distribution-Free Distribution Regression”. In: *AISTATS* (2013).
- [101] Barnabás Póczos, Liang Xiong, Dougal J Sutherland, and Jeff Schneider. “Nonparametric kernel estimators for image classification”. In: *CVPR*. 2012.
- [102] Ali Rahimi and Benjamin Recht. “Random features for large-scale kernel machines”. In: *Advances in neural information processing systems*. 2007, pp. 1177–1184.
- [103] Karthikeyan Rajendran, Assimakis A Kattis, Alexander Holiday, Risi Kondor, and Ioannis G Kevrekidis. “Data mining when each data point is a network”. In: *arXiv preprint arXiv:1612.02908* (2016).
- [104] James O Ramsay and B.W. Silverman. *Functional data analysis*. Wiley Online Library, 2006.
- [105] J.O. Ramsay and B.W. Silverman. *Applied functional data analysis: methods and case studies*. Vol. 77. Springer New York: 2002.
- [106] Siamak Ravanbakhsh, Junier Oliva, Sebastian Fromenteau, Layne Price, Shirley Ho, Jeff Schneider, and Barnabás Póczos. “Estimating cosmological parameters from the dark matter distribution”. In: *International Conference on Machine Learning*. 2016, pp. 2407–2416.
- [107] Walter Rudin. *Fourier analysis on groups*. 12. John Wiley and Sons, 1990.
- [108] Radu Bogdan Rusu and Steve Cousins. “3D is here: Point Cloud Library (PCL)”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011.
- [109] Gideon Schwarz. “Estimating the Dimension of a Model”. In: *Ann. Statist.* 6.2 (Mar. 1978), pp. 461–464.
- [110] Roman Scoccimarro. “Transients from initial conditions: a perturbative analysis”. In: *Monthly Notices of the Royal Astronomical Society* 299.4 (1998), pp. 1097–1118.
- [111] Jayaram Sethuraman. “A constructive definition of Dirichlet priors”. In: *Statistica Sinica* 4 (1994), pp. 639–650.
- [112] Bernard W Silverman. *Density estimation for statistics and data analysis*. Vol. 26. CRC press, 1986.
- [113] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *ICLR*. 2015. arXiv: 1409.1556.
- [114] A. Smola, A. Gretton, L. Song, and B. Schölkopf. “A Hilbert space embedding for distributions”. In: *Algorithmic Learning Theory*. Springer. 2007, pp. 13–31.
- [115] Le Song. “Learning via Hilbert space embedding of distributions”. In: (2008).
- [116] Le Song, Byron Boots, Sajid M Siddiqi, Geoffrey J Gordon, and Alex J Smola. “Hilbert space embeddings of hidden Markov models”. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 991–998.

- [117] Le Song, Kenji Fukumizu, and Arthur Gretton. “Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models”. In: *IEEE Signal Processing Magazine* 30.4 (2013), pp. 98–111.
- [118] Kumar Sricharan, Dennis Wei, and Alfred O. Hero III. “Ensemble estimators for multivariate entropy estimation”. In: *IEEE Trans. Inf. Theory* 59 (7 2013), pp. 4374–4388.
- [119] Dougal J Sutherland, Junier B Oliva, Barnabás Póczos, and Jeff Schneider. “Linear-time Learning on Distributions with Approximate Kernel Embeddings”. In: *arXiv preprint arXiv:1509.07553* (2015).
- [120] Zoltán Szabó, Arthur Gretton, Barnabás Póczos, and Bharath Sriperumbudur. “Two-stage sampled learning theory on distributions”. In: *AISTATS* (2015).
- [121] Hy Trac and Ue-Li Pen. “Out-of-core hydrodynamic simulations for cosmological applications”. In: *New Astronomy* 11.4 (2006), pp. 273–286.
- [122] Alexandre B Tsybakov. *Introduction to nonparametric estimation*. 2009.
- [123] B. Uria. *Connectionist multivariate density-estimation and its application to speech synthesis*. 2015.
- [124] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. “Neural Autoregressive Distribution Estimation”. In: *Journal of Machine Learning Research* 17.205 (2016), pp. 1–37.
- [125] Benigno Uria, Iain Murray, and Hugo Larochelle. “A Deep and Tractable Density Estimator.” In: *ICML*. 2014, pp. 467–475.
- [126] Benigno Uria, Iain Murray, and Hugo Larochelle. “RNADE: The real-valued neural autoregressive density-estimator”. In: *Advances in Neural Information Processing Systems*. 2013, pp. 2175–2183.
- [127] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6000–6010.
- [128] A. Vedaldi and B. Fulkerson. *VLFeat: An Open and Portable Library of Computer Vision Algorithms*. <http://www.vlfeat.org/>. 2008.
- [129] Andrea Vedaldi and Andrew Zisserman. “Efficient additive kernels via explicit Feature Maps”. In: *CVPR*. 2010.
- [130] Sreekanth Vempati, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. “Generalized RBF feature maps for Efficient Detection”. In: *British Machine Vision Conference*. 2010.
- [131] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. “Grammar as a foreign language”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2773–2781.
- [132] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. “Graph kernels”. In: *Journal of Machine Learning Research* 11.Apr (2010), pp. 1201–1242.

- [133] Qing Wang, Sanjeev R Kulkarni, and Sergio Verdú. “Divergence estimation for multidimensional densities via  $k$ -nearest-neighbor distances”. In: *IEEE Transactions on Information Theory* 55.5 (2009), pp. 2392–2405.
- [134] Xuezhi Wang, Junier B Oliva, Jeff G Schneider, and Barnabás Póczos. “Nonparametric Risk and Stability Analysis for Multi-Task Learning Problems.” In: *IJCAI*. 2016, pp. 2146–2152.
- [135] L Wasserman. *All of Nonparametric Statistics*. 2007.
- [136] Christopher K I Williams and Matthias Seeger. “Using the Nyström method to speed up kernel machines”. In: *NIPS*. 2001.
- [137] A. G. Wilson and R. P. Adams. “Gaussian Process Kernels for Pattern Discovery and Extrapolation”. In: *ICML* (2013).
- [138] Andrew Gordon Wilson. *A Process over all Stationary Covariance Kernels*. Tech. rep. 2012.
- [139] Jianxin Wu, Bin-Bin Gao, and Guoqing Liu. *Visual Recognition Using Directional Distribution Distance*. 2015. arXiv: 1504.04792.
- [140] Zichao Yang, Alexander J Smola, Le Song, and Andrew Gordon Wilson. “A la carte-learning fast kernels”. In: *AISTATS*. 2015.
- [141] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. “Deep sets”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 3394–3404.
- [142] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. “Recurrent neural network regularization”. In: *arXiv preprint arXiv:1409.2329* (2014).
- [143] Heiga Zen and Andrew Senior. “Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE. 2014, pp. 3844–3848.
- [144] Xiang Zhang, Junbo Zhao, and Yann LeCun. “Character-level convolutional networks for text classification”. In: *Advances in neural information processing systems*. 2015, pp. 649–657.
- [145] Yu Zhang and Qiang Yang. “A survey on multi-task learning”. In: *arXiv preprint arXiv:1707.08114* (2017).
- [146] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. “Learning Deep Features for Scene Recognition using Places Database”. In: *NIPS*. 2014.
- [147] Zhi-Hua Zhou. “Multi-instance learning: A survey”. In: *Department of Computer Science & Technology, Nanjing University, Tech. Rep* (2004).