5-2019

Efficient Distribution Regression

Nicholas Wilkins
npw3202@rit.edu

ROCHESTER INSTITUTE OF TECHNOLOGY

Efficient Distribution Regression

by

Nicholas Wilkins

A thesis submitted in partial fulfillment for the degree of
Masters of Science in Computer Science

in the
Golisano College of Computing and Information Sciences
Computer Science Department

May 2019

Committee Approval

Name	**Signature**
Ifeoma Nwogu (Advisor)	_____
Rui Li (Reader)	_____
Qi Yu (Observer)	_____

ROCHESTER INSTITUTE OF TECHNOLOGY

Abstract

Golisano College of Computing and Information Sciences

Computer Science Department

Masters of Science

by Nicholas Wilkins

In this thesis we devise a method for performing distribution regression utilizing an extension of mixture density networks. This method will be benchmarked against other state-of-the-art methods (utilizing negative log loss) and distribution-agnostic models of similar complexity. We demonstrate that our proposed method performs at least as well as the state-of-the-art methods in distribution regression. Additionally, we show that adding distribution predictive capabilities does not significantly decrease performance in models. We illustrate how one can utilize this methodology to regress to confidence interval predictions. Finally, we demonstrate a variety of novel and interesting applications of our framework.

Acknowledgements

I would like to thank Professor Ifeoma Nwogu for always being by my side and helping me through this challenging project. I would also like to thank Professors Rui Li and Qi Yu for helping me through this project and working on my committee.

I would like to express my gratitude to my colleagues for their wonderful collaboration. You helped me greatly.

I would like to thank my friends, family, and loved ones; without your emotional support, this project would not have been possible.

I would also like to extend my thanks to the faculty of the School of Mathematical Sciences including Professors Wiandt, Radin, Coppenbarger, and Agarwal. Without your support, I would have never made it to here.

Finally, I would like to thank all those that helped me along this journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning is often categorized as either supervised or unsupervised[1] (sometimes other categories such as semi-supervised are included). Classically, supervised machine learning problems (in the terms of PAC learning) are framed as learning a mapping from a feature space ($\mathbb{X}$) to a target space ($\mathbb{Y}$): $f : \mathbb{X} \to \mathbb{Y}$ such that some loss function $\mathcal{L}$ is minimized. Supervised machine learning problems are often categorized as either classification or regression. Traditional regression algorithms (e.g. linear, polynomial based, neural networks) often only provide point estimates.

This can be problematic when the feature space does not capture sufficient information to provide an adequate map onto the target space. Unfortunately, including additional features is often expensive and time consuming. Even when those features are available, under the presence of limited data, over-extending the feature space can lead to the *curse of dimensionality*[1].

Additionally, processes we are trying to model frequently have inherent stochasticity. Thus, modeling the outcome perfectly (or even adequately) with a point estimate is frequently impossible. Take for example the problem of predicting the location of an

electron when observed given it's orbital. Due to the inherent stochasticity of quantum mechanics, providing a point prediction provides little information.

Existing methods to remedy this issue are not satisfactory. Frequently, under the presence of uncertainty, people commonly utilize uncertainty measures (such as MC Dropout[2], Gaussian / Deep Gaussian Processes[3], Kernel Methods[1], Bayesian Neural Networks[4][5][6], etc...). However, these methods often are either significantly more computationally intensive than or lack many of the benefits (e,g, complexity, variety of input domain) of their point estimate counterparts. Frequently, these methods focus exclusively on regressing to a single variable.

Additionally, often uncertainty bounds do not provide sufficient information. Take again for example the aforementioned problem of predicting the location of an electron. As the orbital structure cannot be well approximated with a multidimensional Gaussian, uncertainty bounds provide little information.

Existing methods do exist to regress onto distributions (namely Mixture Density Networks[7]), but those methods often do not extend well into multi or high dimensional data. In addition, these models require the specification of the number of components in a mixture when regressing to a mixture model.

In this thesis, we present a novel extension to Mixture Density Networks (we will refer to this extension as an infinite Mixture Density Network abbreviated as iMDN) which extends nicely to multiple dimensions. In addition, this method does not utilize the number of components as a hyper-parameter and thus is capable of regressing to an arbitrary and variable number of components.

We also present and examine a degenerate case of Mixture Density Networks(we refer to this degenerate case as a Gaussian Network or GN) which can simply derive uncertainty

measures.

1.0.1 Scope

This thesis will examine two separate extensions of MDNs: GNs and iMDNs. In addition, we will demonstrate through a multitude of experiments that our GNs outperform other uncertainty quantification methods and that iMDNs outperform MDNs. We will also demonstrate through various experiments the utilities of both of these methods.

1.0.2 Outline

This thesis will begin with a discussion of prior works and prior attempts to solving these problems. We will then cover the background information required to become acquainted with MDNs. Next, we will describe the theoretical underpinnings and the practical implementation of both GNs and iMDNs. In addition, we will discuss one powerful application of GNs (specifically unsupervised anomaly detection). Finally, we will demonstrate that our methods achieve state-of-the-art accuracies.

1.0.3 Prior Works

Deep Gaussian Processes Deep Gaussian processes are a class of models utilized for regression that combine Gaussian processes (GPs) with deep architectures. These were initially introduced by Damianou [3]. Deep GP is a composition of GP's where each layer l consists of $D^{(l)}$ GP units that connect it to the next layer. Imagine a neural network with one or more hidden nodes and each edge connecting any two nodes in the network is a GP. Exact inference on deep GPs is intractable, and although several variational approximation methods have been proposed, they are difficult to implement

and do not extend readily to arbitrary kernels. They can be used to perform a regression with uncertainty bounds through a technique known as Kriging, but struggle with high dimensional or large-scale data.

Because our proposed approach does not bear the significant overhead burden when training, it is capable of regressing on high-dimensional, large scale image data and provides uncertainty measures on the predictions made.

MCDropout MC Dropout[2], a method aimed at replicating the behavior of a Deep Gaussian process, can also be used for uncertainty quantification. However, utilizing this method requires ensembling on the network. This leads to multiple evaluations of the network which results in additional computational expense.

As we regress directly on the distribution parameters, we do not need to sample from our network thus making the network substantially faster (as we only need to make one forward pass through our network).

Ensemble Methods Recently there has been research into utilizing Deep Ensembles[8] (an ensemble of deep learners) to create multiple hypotheses. From these hypotheses, an uncertainty can be inferred. While extremely promising, utilizing this method requires one to train multiple deep learners and evaluate multiple deep networks to generate uncertainty resulting in a fairly computationally expensive process. We avoid these issues by utilizing a single regressor. Since we are only utilizing a single regressor, we only need to train, evaluate, and store one regressor.

Bayesian Neural Networks Drawing inspiration from statistics and probability theory, Bayesian regression assigns each parameter a prior probability distribution.

Bayesian learning, via the Bayesian update rule, is utilized to update the probability distributions to best fit the data. One can extend Bayesian Regression to neural networks[4][5][6] utilizing a similar methodology to produce uncertainty. As before, this requires storing and optimizing a distribution for each parameter, and thus is computationally expensive. Additionally, to determine a hypotheses and uncertainty, one must sample the network utilizing techniques such as Variational Inference[9] or Markov chain Monte Carlo (MCMC) methods [10].

We are performing ordinary regression on the distribution parameters; each of our weights and biases take on a single value. Thus, we do not need to sample from our network. This enables us to use our method on large scale imaging datasets.

Mixture Density Networks MDNs (the basis from which our work was inspired) regress to a finite and fixed number of distribution parameters[7]. While technically encompassing the Gaussian Networks proposed here, MDNs are seldom utilized for uncertainty predictions. In addition MDNs do not work well on high dimensional output due to the number of output variables. Our iMDN methodology solves this latter problem and enables MDNs to be applied to arbitrarily high output dimensions.

1.0.4 Background

1.0.4.1 Mixture Models

A mixture between two distributions D_1, D_2 with PDFs f_1, f_2 (respectively) with mixing coefficients α_1, α_2 ($0 \leq \alpha_0, \alpha_1 \leq 1$ and $\alpha_0 + \alpha_1 = 1$) is defined as a distribution with the following PDF

$$f_{mix}(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x).$$

Therefore, one can sample from this distribution by drawing a random value $x \sim U_{[0,1]}$ and returning a sample from D_1 if $x < \alpha$ and otherwise returning a sample from D_2. Mixture models can be extended to arbitrarily many distributions. Suppose we have distributions $D_1, D_2, ..., D_n$ with PDFs $f_1, f_2, ..., f_n$. Suppose further we have mixing coefficients $\alpha_1, \alpha_2, ..., \alpha_n$ such that $0 \leq \alpha_i \leq 1$ and $\sum_i \alpha_i = 1$. Then, we define a mixture of these distributions as a distribution with PDF

$$f_{mix}(x) = \sum_{i=1}^{n} \alpha_i f_i(x)$$

1.0.4.2 Infinite Mixture Models

Mixture Models can be naturally extended to infinite mixture models. Suppose we have an infinite series of distributions $\{D_i\}_{i=1}^{\infty} = D_1, D_2, ...$ with PDFs $\{f_i\}_{i=1}^{\infty} = f_1, f_2, ...$ Suppose furthermore we have a sequence of mixing coefficients $\{\alpha_i\}_{i=1}^{\infty}$ $0 \leq \alpha_i \leq 1$ and $\sum_i \alpha_i = 1$. Without loss of generality, we assume that sequence is monotonically decreasing [1]. Then, our infinite mixture will be defined as a distribution with PDF

$$f_{mix}(x) = \lim_{N \to \infty} \sum_{i=1}^{N} \alpha_i f_i(x)$$

1.0.4.3 Summary of MDNs

Mixture Density Networks provide a framework for regressing onto distributions. MDNs can be applied to convert a wide variety of point regressor networks to distribution regressors. Rather than regressing to a single value, one imposes a distribution (typically

[1] As our sum will converge absolutely, we rearrange terms to make our α sequence monotonically decreasing.

a Gaussian Mixture Model) on the target space. One then regresses directly onto the distribution's parameters.

For notational simplicity, let N denote the network, $N(x)$ denote the output of the network, $D_{N(x)}$ denote the distribution induced by the parameters from $N(x)$ and $f_{N(x)}(y)$ indicate the PDF of $D_{N(x)}$.

To train this network, one trains on the Negative Log Likelihood (NLL) loss function:

$$\mathcal{L} = -\sum_{i=1}^{N} log(f_{N(x_i)}(y_i)).$$

Under the presence of infinite i.i.d. data, an optimal learning algorithm will learn the distribution parameters of the process which generated the data. A proof for this is provided in the appendix.

Chapter 2

Gaussian Networks

We begin with a discussion on Gaussian Networks. These can be viewed as a simplification of MDNs. For this simplification, we regress to a Gaussian Distribution. This confers multiple benefits: this is a fairly simple distribution and therefore easy to interpret; in addition, this can be viewed as an uncertainty prediction; finally, this provides minimal overhead to both training and evaluation. Therefore, this chapter provides the following contributions:

1. The development of an uncertainty aware regressor which can be applied to almost any problem.

2. The application of this uncertainty aware regressor to various interesting problems ranging from vision to finance.

3. A rigorous analysis of this regressor compared to other state-of-the-art regressors.

4. A data cleaning framework which can be utilized to significantly improve the performance of any regressor on a noisy dataset.

To learn this mapping, we train a regressor to output the parameters of our target distribution with the following log-likelihood loss:

$$\mathcal{L} = -\iint_S \log(\rho_x(y))p(x,y)dS \tag{2.1}$$

where $p(x,y)$ is the PDF of the true joint distribution on X, Y and ρ_x is a Gaussian induced with parameters from the regressor. In Appendix A we demonstrate that an optimal learning scheme (with appropriate assumptions) will converge to the true distribution parameters assuming the target distribution is a Gaussian. Thus, a scheme which is optimal under this loss will also have a minimal mean squared error (or any other loss) to the target.

This loss under finite data degenerates into NLL loss:

$$\mathcal{L} = -\sum_{x\in X}\sum_{y\in Y}\log(\rho_x(y))f(x,y) \tag{2.2}$$

$$= -\sum_{i=1}^{N}\log(\rho_{x_i}(y_i)) \tag{2.3}$$

where $f(x,y)$ is the frequency (x,y) occurs in the dataset (i.e. the joint probability mass function). As our target distribution is a Gaussian[1],

$$\mathcal{L} = -\sum_{i=1}^{N}\log\left(\frac{1}{\sqrt{2\pi\sigma_{x_i}^2}}e^{-\frac{(y_i-\mu_{x_i})^2}{2\sigma_{x_i}^2}}\right) \tag{2.4}$$

$$= -\sum_{i=1}^{N}\left(-\log\left(\sqrt{2\pi\sigma^2}\right) - \frac{(y_i-\mu_{x_i})^2}{2\sigma_{x_i}^2}\right) \tag{2.5}$$

$$= \frac{1}{2}\sum_{i=1}^{N}\left(\left(\frac{y_i-\mu_{x_i}}{\sigma_{x_i}}\right)^2 + \log(2\pi) + 2\log(\sigma_{x_i})\right) \tag{2.6}$$

[1] for this simplification, please note that log is base e

which will reach its minimum where

$$\mathcal{L}_2 = \sum_{i=1}^{N} \left(\frac{y_i - \mu_{x_i}}{\sigma_{x_i}} \right)^2 + 2 \sum_{i=1}^{N} \log(\sigma_{x_i}) \tag{2.7}$$

reaches its minimum. This loss is preferable over a multitude of other losses (such as KL Divergence) as it does not require defining an auxiliary ground truth probability distribution.

This Gaussian network can be utilized for both uncertainty quantification and anomaly detection.

2.0.1 Uncertainty Prediction

To test our architectures, we test it on both toy, numerical, and various interesting real problems.

2.0.1.1 Toy Caloric Dataset

As an initial test of our framework, we perform a one dimensional regression utilizing one parameter on a toy dataset created from Kaggle ("Exercise and Calories"). This is used purely for illustrative purposes. In addition, this will provide empirical evidence that this network is capable of capturing aleatory uncertainty (at least for basic scenarios). We attempt to determine how many calories an individual burned based on body heat. Please note that we add artificial noise to discourage the network from memorizing the mean and standard deviations of each input.

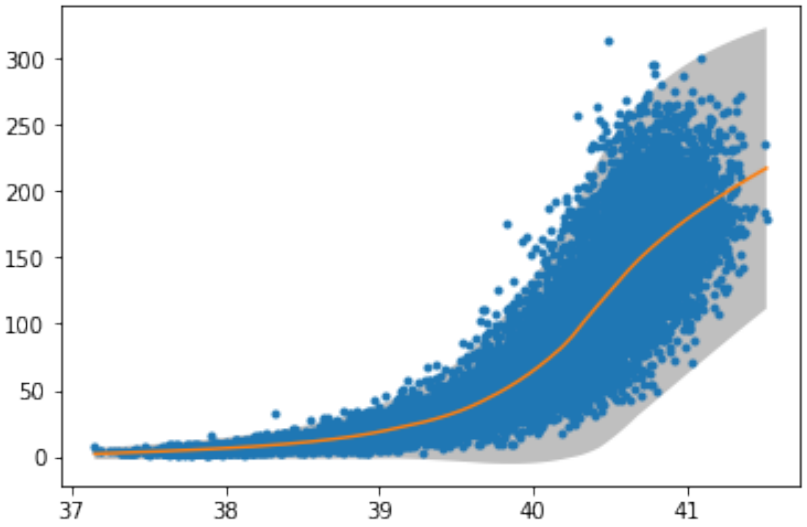

FIGURE 2.1: Regression on calories burned from body heat data. The yellow line is the regression. The gray shaded region is the 3σ confidence interval

In Figure 2.1, one can observe that this network successfully converges to perform the distribution regression. As this dataset is purely for demonstrative purposes, we do not include quality metrics.

2.0.1.2 Numerical Datasets

Dataset	[6] PBP	[2] MC-Dropout	[8] Deep Ensembles	GN	iMDN
Boston	2.57 ± 0.09	2.46 ± 0.25	2.41 ± 0.25	**2.23 ± 0.05**	2.38 ± 0.06
Concrete	3.16 ± 0.02	3.04 ± 0.09	3.06 ± 0.18	3.05 ± 0.04	**3.028 ± 0.02**
Energy	2.04 ± 0.02	1.99 ± 0.09	**1.38 ± 0.22**	1.91 ± 0.02	1.69 ± 0.01
Kin8nm	-0.90 ± 0.01	-0.95 ± 0.03	-1.20 ± 0.02	-1.18 ± 0.02	**-1.27 ± 0.05**
Naval-propulsion	-3.73 ± 0.01	-3.80 ± 0.05	**-5.63 ± 0.05**	-3.82 ± 0.09	-3.48 ± 0.04
Power plant	2.84 ± 0.01	2.80 ± 0.05	2.79 ± 0.04	2.85 ± 0.01	**2.80 ± 0.01**
Protein	2.97 ± 0.00	2.89 ± 0.01	2.83 ± 0.02	**2.14 ± 0.01**	2.77 ± 0.02
Wine	0.97 ± 0.01	0.93 ± 0.06	0.94 ± 0.12	0.87 ± 0.02	**-0.11 ± 0.01**
Yacht	1.63 ± 0.02	1.55 ± 0.12	**1.18 ± 0.21**	4.06 ± 0.00	3.89 ± 0.07
MSD	3.60 ± NA	v3.59 ± NA	3.35 ± NA	3.40 ± NA	**3.32 ± 0.01**

TABLE 2.1: Comparison of different architectures performance for NLL on popular benchmark datasets. Measurements courtesy of Deep Ensembles paper by Lakshminarayanan et al. [7].

We demonstrate that the model is on-par or superior to other popular uncertainty quantification models (specifically PBP [6], MC Dropout[2] and Deep Ensembles[8]) for regression under several benchmark datasets commonly used to measure the quality of a regression algorithm. As can be observed in Table 2.1, with the exception of the Yacht

dataset where our technique is subpar, we performed on-par with or out-performed all other approaches in terms of NLL (negative log-loss).

2.0.1.3 Stock Data

To test our ability to quantify uncertainty in predictions of large, complex, stochastic, and highly volatile *time series data*, we applied our methodology on financial market data. We specifically modeled similar stocks from the entertainment industry[2]. The family is comprised of stocks from 21^{st} *Century Fox, Inc. (FOX), Netflix, Inc. (NFLX), Time Warner, Inc. (TWX), Amazon.com, Inc. (AMZN), Walt Disney Co. (DIS), Comcast Corporation (CMCSA)*. The stocks are classified as a family based on their sector, industry, asset class, and the prices of the stocks over an extended period of time being highly correlated with each other.

Suppose we are given the stock close prices for n days prior to day T : $\{x_t\}_{T-n-1}^{T-1}$. We wish to predict the closing price on day T: x_T. To do this, we predict to prescribe a distribution onto $x_T \sim \mathcal{N}(\mu_T, \sigma_T)$.

To analyze the uncertainties resulting from the implementation, we set a threshold of 0.5 so that days on which the uncertainty measure was above this threshold were flagged as anomalous trading days. We provide a list of *FOX*-related news [3] in that period and compare with the anomalous days predicted by our network. The results are shown in Table 2.2.

[2]One of the authors spent his summer internship at a financial organization and specifically analyzed this family of stocks.

[3]News data was obtained from `https://www.reuters.com/finance/stocks/FOX/key-developments`. We threw away many other events leaving those related to where our uncertainties were high.

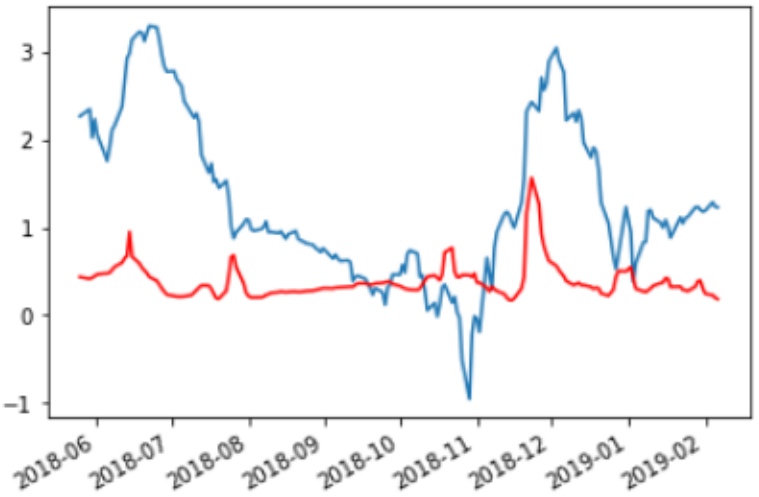

FIGURE 2.2: The blue graph is the stock price chart for *FOX* while the red graph is the measure of uncertainty estimated by the network. The Y-Axis is normalized via Z-Score with trends removedImage is best viewed in color

Real Date	**Network predictions**	**News related to 21st Century FOX**
05-17-18	05-31-18	-Suzanne Scott named CEO Of *FOX* News
06-13-18	06-15-18	-Comcast offers to buy 21st Century Fox media assets for $65B in cash
10-19 till 10-20-18	10-19 till 10-22-18	-Walt Disney receives unconditional approval from China For 21st Century Fox deal; -Amazon/Blackstone bid for Disneys 22 regional sports networks;
11-26-18	11-26-18	-Disney, Fox sued in U.S. for $1B over Malaysia theme park
01-07-19	—	-21st Century Fox announces filing of registration statement on Form 10 for Fox

TABLE 2.2: The left column shows true dates on which major events occurred at 21st Century Fox; the second column shows the closest date estimated by our network, and the last column describes the event.

2.0.1.4 Age Estimation from Face Image

To test how well this architecture will work on large complex datasets, we applied it on the nontrivial problem of age estimation. Given an image of a face, the network was tasked with predicting the age of the individual in the picture. Posed as a general problem, this task is a very challenging regression problem.

We utilized the IMDb-Wiki Dataset[11]: a dataset of half a million faces scraped from both IMDb and Wikipedia (primarily IMDb), and tagged with the corresponding ages of individuals in the images. This dataset was generated by first identifying faces in

FIGURE 2.3: Examples of invalid data in the IMDb-Wiki dataset. Images were identified as having extremely high uncertainty and loss values.

images utilizing the Mathias et. al. face detector[12]. The faces were then given a 40% margin around the border and cropped out. Finally, the age was automatically extracted from the document by extracting both the time of the photograph and the year of the individual's birth. Due to the highly automated nature of the collection of the data, this dataset is very noisy with multiple entries in the dataset containing either no faces or multiple faces. Additionally, several of the entries contain just a copyright sign. Also, in some cases, the collection year was incorrectly extracted from the webpage. See Table 2.3 for examples of invalid face images.

Although this dataset contained a similar distribution of males to females (see Figure 2.4 (right)), it contained primarily individuals between 20 and 40 years old. Additionally, because the IMDb dataset contained a random sampling of Hollywood actors, the dataset was primarily composed of young Caucasian individuals resulting in a high implicit bias. We empirically demonstrate that our method is still capable of correctly identifying underrepresented samples in spite of the imbalances in the data.

We did not wish to excessively clean the data, but rather remove the clearly wrong data.

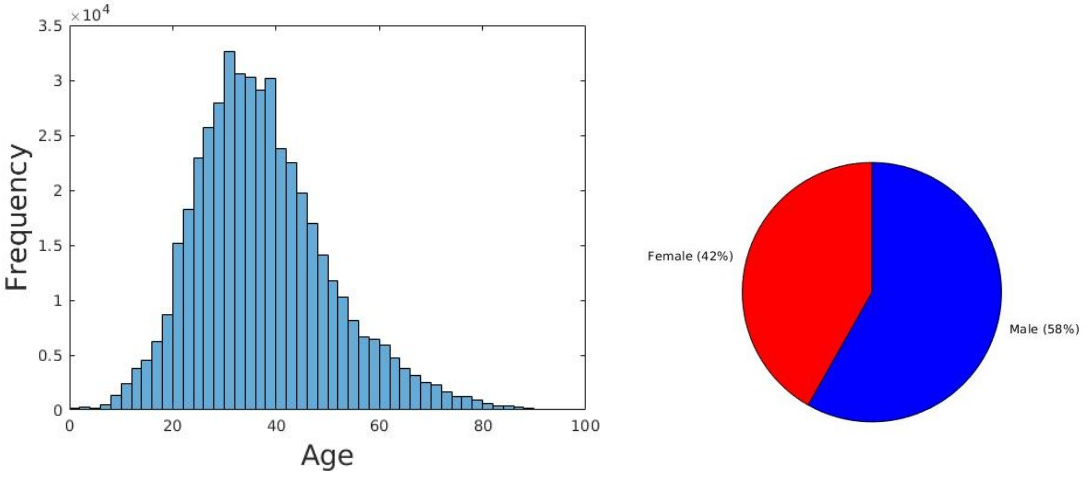

FIGURE 2.4: Statistics of the IMDb dataset. From the image labels provided on the left is age distribution and the right shows gender distribution.

We did this by removing samples which had individuals younger than three years old or older than 100 years old. Additionally, we removed images which were too small (namely smaller than 16 pixels by 16 pixels). We then re-sized all the images to 224 pixels by 224 pixels.

We did not remove invalid images which contained multiple or no faces. However, we did convert all the images to RGB format (if they were black and white, we replicated that channel over the R, G, and B channels). Additionally, we did not remove the mislabeled entries (those which had valid ages and valid images, but were clearly mislabeled). We did not remove these so we could test the ability of the uncertainty quantification. One would expect an appropriate uncertainty quantification algorithm would give high uncertainty for invalid data or abnormal data. We exploit this later to automatically clean the dataset.

This image-based network was trained utilizing a 16-layer convolutional neural network (CNN) with an Adam optimizer, a learning rate of 0.00005 and a batch size of 8 until convergence (6 epochs).

FIGURE 2.5: The three numbers below each image correspond to (i) the actual age (as provided in the dataset)/(ii) the estimated age (as predicted by the regression network)/(iii) the uncertainty value reported by the network (the higher the value, the more uncertain the prediction). The top row shows some of the faces on which the network reported the lowest error values. The middle row shows the faces on which were reported the highest errors; and the last row shows the faces on which the network reported the highest uncertainty.

Method	**MAE**	**NLL**
CNN + Regressor	7.54	-
CNN + Regressor + Uncertainty	7.57	3.63
CNN + Regressor + Uncertainty + Cleaning	5.22	3.53

TABLE 2.3: The accuracy (both mean absolute error and negative log likelihood) of various approaches on age estimation.

The results of these experiments were very promising (first row of Figure 2.5). Even on this noisy dataset, the architecture only performed poorly when the ground truth annotations provided to us was wrong (see the middle row of Figure 2.5). These results demonstrate that our model is capable of capturing uncertainty. Additionally, this architecture's uncertainty not only expressed how confident the model was, but also how clean the data sample was. Thus, this model often reported high confidence if a sample was well represented within the dataset. Empirically we can see that difficult samples (those in a class with low representation, poor lighting, side facing faces, ambiguous individuals, multiple faces) obtained high uncertainties. Images in the dataset which

were incorrectly scraped along with excessively noisy or incorrect data had the highest uncertainty (see the last row of Figure 2.5). This architecture can therefore be used to evaluate the quality of samples (assuming a large portion of the training data is of good quality.)

2.0.2 Determining the overhead of uncertainty quantification

An error quantification network is often only appealing if it does not have a significant impact on performance. Thus, the quantification of the discrepancy of some error metric (say RMSE) between a classical regressor with ω parameters and that of an uncertainty-aware regressor with ω parameters should be minimized. To this end, we train two networks utilizing the same initial configuration of parameters and same number of parameters (except for the last layer) until convergence (one vanilla regressor and one error quantification regressor).

Examining Table 2.3, we can observe that the discrepancy between the MAE of the uncertainty-agnostic regressor and the uncertainty-aware regressor is negligible. Thus, computing the uncertainty does not provide any significant additional overhead to this model. This final layer can therefore be added to almost any regressor to provide uncertainty metrics.

2.0.3 Automated data cleaning

As described earlier, this architecture can be utilized to determine the quality of a sample by examining the uncertainty produced. After training, we identified the samples with the top 5% uncertainty and removed them (please note that we left the validation samples unchanged). After removing these samples from our training set and retraining on the

now cleaned dataset, we obtained significantly better results on the validation dataset (see Table 2.3). Thus, this architecture is uniquely well-suited for unclean datasets to generate relatively high performing regressors.

Chapter 3

Anomaly Detection

In the era of "Big Data", anomaly detection has become increasingly important. Anomaly detection enables a wide variety of tasks ranging from detecting suspicious network traffic to determining anomalous health information. Anomaly detection algorithms are crucial to parse through the increasingly massive amounts of data we have at our fingertips.

Unfortunately, most popular methods of anomaly detection utilize classical approaches and domain specific methodologies (such as ARIMA [13]). While useful in a specific domains, these methodologies cannot be applied (without significant manipulation) to diverse datasets (such as video information).

In this chapter, we introduce a novel method of performing unsupervised anomaly detection on time series information. This methodology is founded on our Gaussian Network Framework devised in the previous chapter.

We demonstrate the merits of our anomaly detection framework through rigorous experimentation and challenging tasks. In addition, we rigorously and theoretically justify the principles behind its operations.

Therefore, the contribution of this chapter is as follows

1. Utilizing the GN framework, we develop and demonstrate a novel unsupervised anomaly detection framework.

3.1 Prior Work in Anomaly Detection

As anomaly detection is a highly useful task there are a myriad of techniques which have been devised to detect anomalous data. These approaches can widely be classified into two separate families of approaches: supervised and unsupervised. In the supervised approach, one examines anomalies and attempts to model anomalous behavior. Alternatively, one can utilize an unsupervised approach. In this approach, one only trains on "normal data". When queried, one determines if the provided data deviates significantly from the developed model.

Statistical Methods One popular approach to anomaly detection is to apply statistical measures[14]. If data deviates significantly from measured statistical quantities (e.g. deviates significantly from the mean), one concludes that that datapoint is anomalous. These methods struggle on non-standard input data (such as video data).

Clustering Based Approaches One can also utilize clustering based approaches[14]. If one assumes that anomalous data are primarily outliers, one can cluster the data and examine when datapoints are far away from any clusters. While this often works well, this tends to breakdown in non-numerical datasets which are difficult to cluster. Additionally, this is difficult to apply to data which lack obvious metrics (such as video data).

SUPERVISED APPROACHES One can also apply supervised approaches[14] where one attempts to predict directly if a datapoint is anomalous or normal. The primary drawback of this approach is that it requires collecting a representative sample of the anomalies which could occur (which often is infeasible).

3.2 Methodology Overview

Suppose we are given multiple time series $X_j = \{x_{i,j}\}_{i=0}^{T_j}$ with $x_{i,j} \in \mathcal{X}$ (where $\mathcal{X}$ is any measurable space). Suppose each X_j is drawn from some stochastic process P (i.e. $X_j \sim P$). When presented with a new time sequence $Y = \{y_i\}_{i=0}^{T_y}$ with $y_i \in \mathcal{X}$, drawn from some unknown process, we wish to determine if that unknown process deviates significantly from our original process P.

3.2.1 Problem Definition

Although our problem is largely unsupervised, we prescribe the following supervised task: Given $X_{t-k-1:t-1}$, determine $(\mu, \Sigma) \in \Omega$ such that $P(X_{t+P}|\mathcal{N}(\mu, \Sigma))$ is maximized. Note that this problem is the identical problem posed in the previous chapter. Thus, we utilize Gaussian Networks to model this problem.

3.2.2 Application to Anomaly Detection

Recall that we have a collection of normal time series X. In addition, we have a time series sampled from an unknown process. We wish to determine if this sample is anomalous. Thus, we want to determine if this sample could have feasibly come from the process P.

To do this, we utilize our Gaussian Network Framework to develop an auto regressive model which models the signals in X. We then apply said model to Y and calculate $\frac{1}{N-P-k-1}\sum_t \mathcal{L}^*(Y_{t-k-1:t-1}, Y_{t+P})$. This loss on the entire time series will be large if our regressive model expected a fairly deterministic behavior but was *surprised*. We define surprise as when the network expected a specific event with relatively high probability (and low uncertainty) and something completely different occurred. For the context of this chapter, we will exclusively be focusing on incidents of surprise. Please note that this will completely ignore situations of turbulent signals which suddenly become tranquil.

3.3 Experiments

As stated previously, we are interested in the problem of anomalous behavior detection. We are specifically interested in the problem of detecting anomalous behavior from surveillance videos. Thus, we will utilize this methodology on the problem of determining anomalous videos from security footage. We wish to be able to detect anomalous portions of videos after **only** training on normal videos. To do this, we employ the methodology illustrated above.

We will be utilizing the *UCF-Crime*[15] dataset. This dataset is comprised of 1900 videos totaling 128 hours of video. These videos are down-sampled to a resolution of 128×128 pixels.

We train a CNN-LSTM-CNN network on the problem illustrated above (predicting frame $t+P$ given frames $t-k-1$ to $t-1$). We assume for simplicity that each pixel is independent of every other pixel (i.e. we constrain our covariance matrix to be a diagonal matrix).

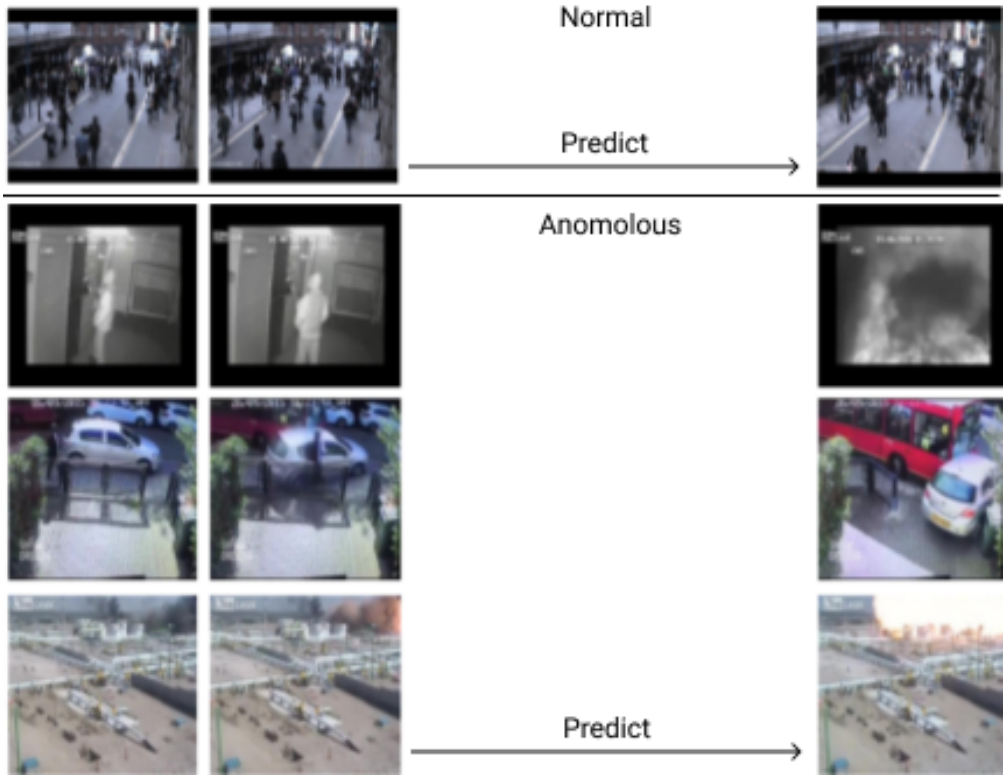

FIGURE 3.1: Anomalous prediction task vs Normal prediction task. The left columns have the provided initial frames. The right frame is the frame our system must predict.

Thus, our network takes as input a video stream. Due to the aforementioned assumption, our network needs only output two images: μ, σ. We then utilize the NLL loss to train this network.

3.3.1 Results

The following is the results of running the algorithm on normal and anomalous videos. In the normal error not only is the error low, but also consistent. In contrast, in the anomalous videos the error is variable. In addition, one can observe areas of highest anomaly.

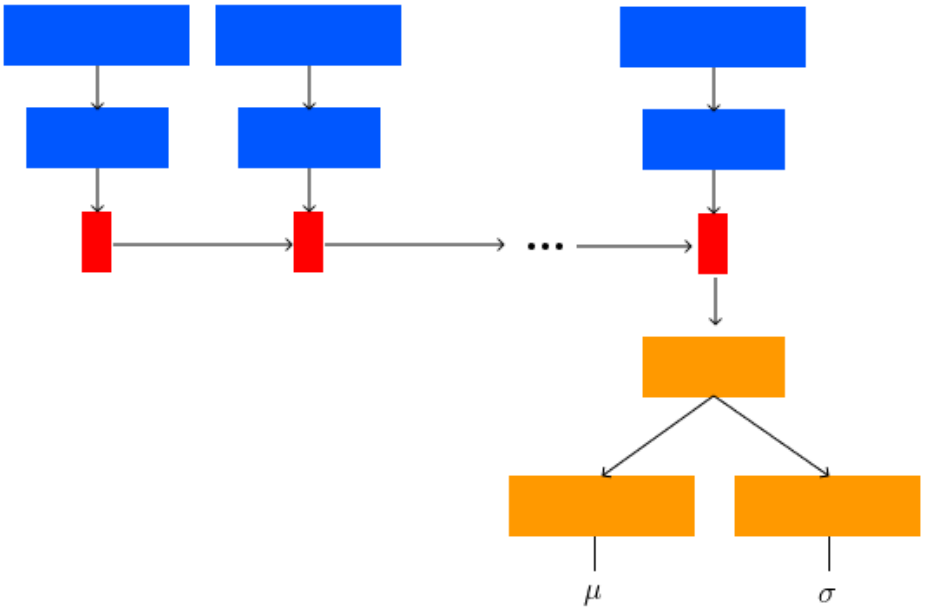

FIGURE 3.2: Network Architecture

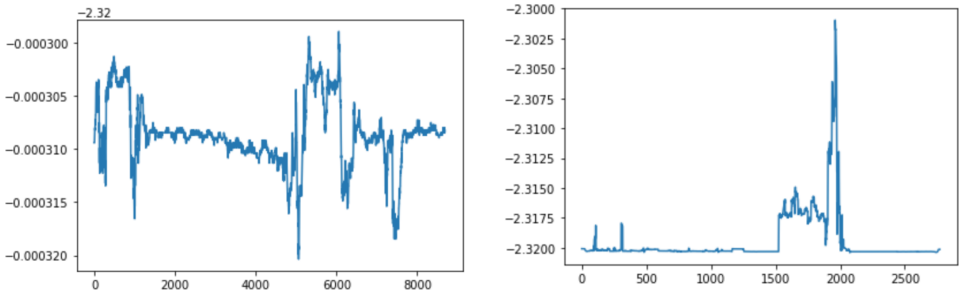

FIGURE 3.3: Normal (on the left) and anomalous (on the right). X axis is the frame number, Y axis is the loss for each frame.

Chapter 4

Infinite Mixture Density Network

In this chapter we explore the problem of regressing to a distribution. Regressing to a distribution has a multitude of benefits: one can extract MLE and mean from the regressed distribution to obtain point estimates; in addition, one can gain a holistic view of likely outcomes (including holes in the distribution); finally, one can sample from the expected distribution to create natural generative models (rather than passing in a random sample as in GANs or sampling in the middle of the model as VAEs).

Mixture Density Networks[7] are a popular method of regressing directly onto a distribution. This framework was introduced by Bishop in his 1994 doctoral thesis [7]. Given $\{(x, y)\}_{i=0}^{\infty} \sim \mathcal{D}$ where $\mathcal{D}$ is a joint distribution on $(\mathbb{X}, \mathbb{Y})$, we wish to determine some function ϕ such that $\phi(x) = \mathcal{D}_{|X=x}$. To do this, one regresses directly to the parameters of some parametric distribution (namely one with a finite, fixed number of parameters). For example, if one imposes a one dimensional Gaussian on our output variable, one would regress to the mean and variance of said Gaussian. One

popular distribution to regress to is a Gaussian Mixture Model (GMM). One then evaluates the quality of the regression by utilizing a negative-log-likelihood cost function: $\mathcal{L} = -\sum_i \log(P(y_i|\mathcal{D}(\phi(x_i))))$.

Unfortunately, one must choose a distribution with a fixed number of parameters. Thus, for the GMM typically used, one must fix the number of components utilized throughout the entire problem. This becomes problematic in higher dimensions where data often clusters in highly unintuitive ways. Furthermore, various cross-sections of complex data typically does not produce the same number of clusters (take for example bifurcated data - see Figure 4.3).

Additionally, MDNs frequently fail to operate (fail to converge to a meaningful minimum when training) when regressing to many parameters. Thus, this framework becomes increasingly unwieldy for more then one component in high distributions. This is because the number of parameters required to specify a GMM with k components in n dimensional space is $\mathcal{O}(kn^2)$.

In this chapter we present a variant of a Mixture Density Networks capable of regressing to an Infinite Gaussian Mixture Model, thus only requiring the specification of a finite number of parameters. The resultant iMDN demonstrates exciting new results: our iMDN do not require the specification of the number of parameters; in addition, our iMDN work in arbitrarily high dimension and can regress to distributions on non-traditional spaces (such as images or videos). Therefore, this chapter makes the following contributions:

1. The introduction of the infinite mixture density network (iMDN). This iMDN is beneficial for the following reasons:

(a) The iMDN is capable of automatically discovering the number of components to best model a distribution.

(b) The iMDN can provably regress to almost any distribution.

(c) The iMDN can be applied to problems with high dimensional output data.

(d) The iMDN takes the best of Deep Learning and infinite mixture techniques.

2. a Myriad of novel applications such as sequence modeling.

4.1 Infinite Mixture Density Network

Our formulation of iMDN was largely inspired by Dirichlet Processes Mixture Models[16] (specifically the stick-breaking formulation). However, unlike Dirichlet Processes, rather then sample our mixing coefficients and distribution parameters from some base distribution, we query our model. Note that rather then utilizing a concentration parameter, we utilize a regularization parameter which serves the same purpose.

Suppose rather then regressing to a finite number of parameters, we regressed to a process which was capable of outputting an infinite number of components (or parameters). Specifically, suppose we regressed to the input of a recursive model (such as an LSTM network) which outputted the parameters for each component. Such a model would enable us to regress directly to an infinite Gaussian Mixture Model. Additionally, said model would have a finite number of parameters which could be learned from the data through any standard learning program (e.g. gradient descent).

We propose a system with two sub-models a *encoder* and a *generator*. The encoder will process the input and produce an encoding of an infinite Gaussian Mixture Model. The generator will then unravel the encoding into Gaussian components. In practice, we will

truncate this process when we have decided that an adequate amount of the distribution has been explained.

4.1.1 Encoder

Our encoder will be a function of the form $\eta : \mathbb{X} \to \mathbb{R}^j$ where j is the dimension of the embedding. For the purposes of this paper, we will use a neural net (however, any kind of regression which is differentiable can be utilized). This encoder will analyze the input and transform (or encode) it into a format conducive for producing the components of a mixture model)

4.1.2 Generator

Our generator will recursively generate each component. Thus, for (for example) the left diagram in Figure 4.2 we would have three recursive steps: producing the green component, producing the red component, and producing the blue component. Our generator will be a recursive function $\rho : \mathbb{R}^{j+r} \to \mathbb{R}^{p+1+r}$ where r is dimension of the hidden state and p is the number of parameters for each component in our mixture. An additional output is utilized to specify the mixing coefficient. We apply a stick breaking process to this final parameter to determine the final mixing coefficients. Additional requirements can be put on specific parameters depending on the distribution chosen. We can truncate this process when enough of our mixture has been outputted.

4.1.3 Full Model

We utilize the following algorithm to combine the encoder and generator to produce a full distribution. The production of components will stop when $1 - \epsilon$ of the distribution

is explained.

Algorithm 1: Full Model Algorithm

$enc \leftarrow \eta(\mathbf{x})$ ▷ Encode our x into a format best suited for the decoder

$h \leftarrow \mathbf{0}$ ▷ Initialize our hidden recursive state to the 0 vector

$i \leftarrow 0$ ▷ This is our "component counter"

while $\sum_{j<i} \alpha_j < 1 - \epsilon$ **do**

  $\alpha'_i, \theta_i, h \leftarrow \rho(enc, h)$ ▷ Predict mixing coeffs., component params., and hidden state

  $\alpha_i \leftarrow \alpha'_i \prod_{j<i}(1 - \alpha'_j)$ ▷ Stick break the mixing coefficient

  $D_i \leftarrow \mathcal{D}(\theta_i)$ ▷ Instantiate the distribution

  $i \leftarrow i + 1$

end

return $\frac{\sum_{j<i} \alpha_j D_j}{\sum_{j<i} \alpha_j}$ ▷ Normalize the distribution

One can train this model utilizing any traditional training scheme. Additionally, due to the flexibility of this scheme, this model can be applied to model almost any data type. In addition, if one utilizes a symmetric Gaussian as the component distribution, one can use this on almost any data modality (e.g. images). The symmetric Gaussian is advantageous over other possible components (such as a full Gaussian) as one needs only to predict a linear number of outputs (as each dimension is independent of each other) rather than predicting a quadratic number of outputs (specifically the full covariance matrix).

4.1.4 Additional Regularizers

One can also apply additional constraints to control the quality of the produced distribution. To train faster and obtain convergence we introduce an additional regularizer to the objective function: semantic validity.

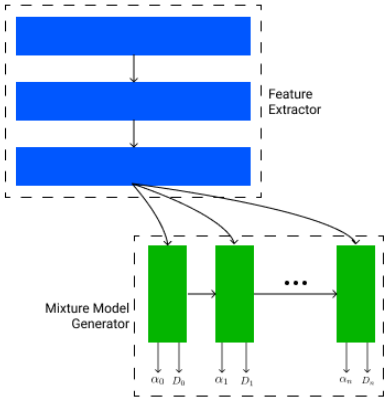

FIGURE 4.1: Example implementation of an iMDN utilizing a recurrent network

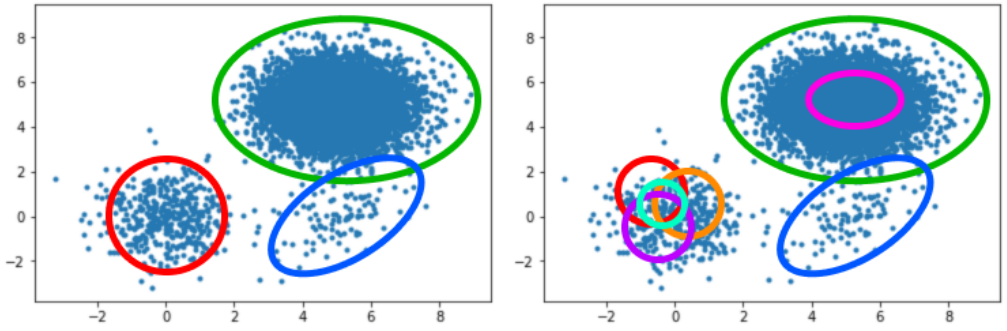

FIGURE 4.2: A comparison of a semantically valid (left) and semantically invalid (right) Gaussian mixture model

One powerful constraint we will impose on our network is that it creates *semantically valid components*. By this, we mean that all produced components are necessary. Stated differently, if this mixture was removed the predicted distribution would differ significantly from the true distribution. Although this condition is **not** necessary to obtain a valid distribution, this can be helpful for producing useful hypotheses.

To comply with this requirement (stated above) that all components of the distribution be necessary, we employ the follow regularization factor: $-\gamma \sum_{i=0}^{\infty} \alpha_i^2$. This regularization factor is designed so that when two mixtures can be feasibly merged the loss function will decrease.

4.2 Experiments

To adequately test iMDNs against MDNs we utilized several benchmarks. We began by applying our methodology several toy datasets. We then measured our performance on the standard regression benchmarks from UCI. Finally, we applied our methods to several interesting non-trivial problems.

4.2.1 UCI Benchmarks

To ensure our algorithm works on real life datasets, we compared the NLL and RMSE of our method with the NLL and RMSE of GNs and other uncertainty predictors. The results can be seen in 2.1.

4.2.2 Bifurcated Data

To ensure our method could adequately handle changing densities in practice, we applied an iMDN to a synthetic bifurcated dataset. To generate this, we produced a grid of possible x values. We then imposed either a normal of a bimodal (two normal) distribution onto the Y variable which we sampled.

One can observe that our model was able to differentiate from the first section and the second, correctly prescribing a unimodal distribution to the first section and a bimodal distribution to the second.

4.2.3 Video Prediction

To further illustrate the utility of our methodology, we apply this methodology to the problem of predicting frame $t + T$ given frames $t - n$ to t. This same framework can

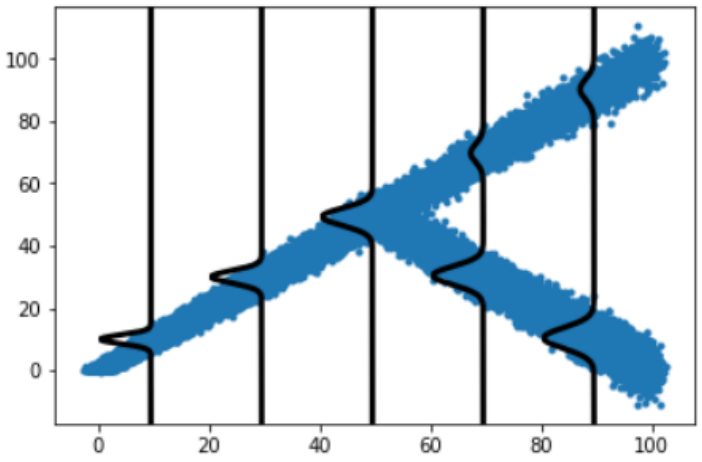

FIGURE 4.3: Our performance on the toy dataset. We wish to predict a distribution of y values given the x value. One can observe the various distributions we predicted for each x value (displayed in black). One can observe that our model adequately anticipates the bifurcation at $x = 50$ and bifurcates it's predictions.

be applied to *any* video prediction task (e.g. bouncing MNIST, etc...). In addition, this can be applied to various interesting categories of stochastic videos (such as rolling a dice). We will develop this framework in a much more constrained environment so we can measure and quantify the performance.

Stated more concretely, suppose we have a collection of videos each containing N frames: $\{\{x_i\}_{i=0}^{N} : x \in X\}$. Learn a mapping ϕ such that $\phi(\{x_k\}_{t-n}^{t}) = x_{t+T}, \forall x \in X$.

4.2.3.1 Car Problem

As stated earlier, for this problem we limit our videos to a fairly constrained problem. For this example, we will apply our iMDN framework to a simulated car going down a highway. Given 10 consecutive frames predict the $t + 5$th frame. As can be seen Figure 4.4, this road branches into two separate roads. When the car hits the branching point it will either traverse on its left or right path (both with a probability of 0.5).

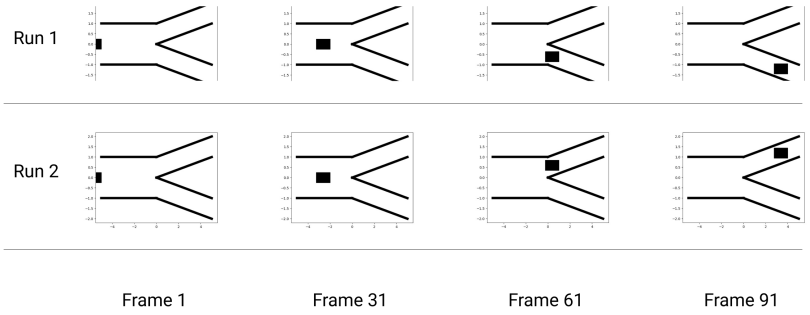

FIGURE 4.4: Example frames from two runs of simulation. For simplicity both the runs have the car initialized to the same spot with the same speed.

The car starts at a variable x location and travels forward with a constant speed (this speed varies between runs). As stated earlier, the system is provided with 10 consecutive frames from a run and expected to predict the $t + 5$th frame.

As can be observed in Figure 4.4, there are sections of the video where multiple outcomes could occur. However, there are also sections of the video where there is exactly one possible next frame.

To approach this problem, we make the simplifying assumption that each of our component's marginalization are independent of each other (i.e. that the covariance matrix is strictly a diagonal matrix).

4.2.3.2 Results

As this is an illustrative problem, exact losses have been omitted from this manuscript. As one can observe in Figure 4.5, our iMDN is able to intelligently discover the sections of the video which require more components and those where only one outcome is possible.

Contrast this with a normal (vanilla) regressor. A traditional regressor would produce the MLE prediction and thus be incapable of modeling the inherent stochasticity. An uncertainty aware regressor (such as a GN) would explain when and where stochasticity

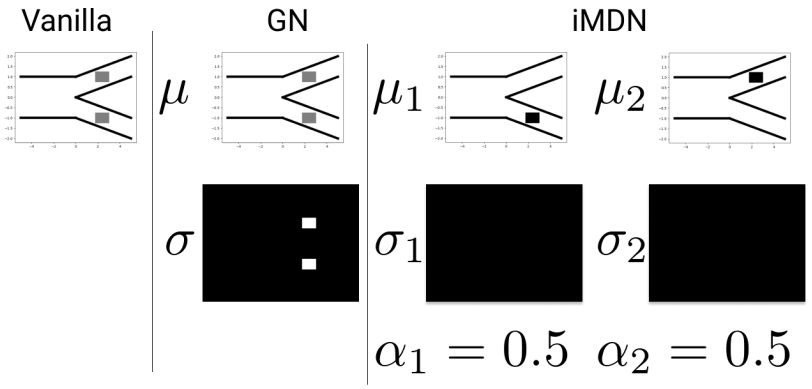

FIGURE 4.5: Example of how a vanilla network, Gaussian Network, and iMDN performs on the car problem. This shows the output of the three networks on one sample. The vanilla network describes what an optimal (under MSE) traditional image to image regressor would produce when trained on MSE till convergence. The GN and iMDN demonstrate what an optimal Gaussian Network and infinite Mixture Density Network would produce when trained until convergence under NLL loss.

occurred but would be unable to produce actual concrete predictions of where the car would be expected to be. Contrast this with our iMDN. Our iMDN can produce each individual possibility, quantify the noise associated with each possibility, and give the probability of each possibility occurring.

Chapter 5

Conclusion

5.1 Conclusion

In this thesis we explored two extensions of mixture density network: GNs and iMDNs. Both of these performed on-par or exceed state-of-the-art results. We also investigated how to utilize these methods to develop unsupervised anomaly detection algorithms. In addition, we demonstrated their applicability to a variety of problems. There are still many open problems regarding this research.

5.1.1 Future Work

Small Datasets In this thesis we observed the utility of GNs and iMDNs. However, it seemed that these methods struggled on datasets of smaller sizes (e.g. smaller than 100 samples). It would be useful to determine how these methods could be extended to work on datasets of smaller size.

Hybrid Bayesian / Graphical Model In this thesis we focused purely on non-graphical deep learning model. It would be interesting to employ graphical models with this methodology to develop hybrid models.

Auto-discovery of regularization coefficients In addition, we observed how important the various regularization constants were in iMDNs. Unfortunately, discovering the proper value for these constants required trial and error. It would be beneficial to determine how to automatically discover these coefficients

Generative Abilities It would also be beneficial to investigate the generative abilities of iMDNs. Suppose we have a distribution D on two **sets** of variables: $\mathcal{X} = \{X_0, X_1, ..., X_{n-1}, X_n\}, \mathcal{Y} = \{Y_0, Y_1, ..., Y_{m-1}, Y_m\}$, (e.g. $\mathcal{X} = \{age, gender, race\}, \mathcal{Y} = \mathbb{R}^{(64*64)}$. $\mathcal{Y}$ contains the images of a variety of individuals with details specified in $\mathcal{X}$). This distribution D is sampled and we are presented with $\{(x_i, y_i)\}_{i=0}^{N} \sim D$

Utilizing iMDNs, one should be able to learn a mapping $\phi : \mathcal{X} \to \mathcal{D}(\mathbb{R}^{(64*64)})$. One should then be able to **naturally** sample $y \sim \phi(x)$ to generate samples y with the criteria specified in x.

It would both be interesting and beneficial to verify this and compare this scheme with other popular generative frameworks (such as GANs and VAEs).

Data Cleaning It would be interesting and beneficial to further investigate the cleaning scheme presented in Chapter 2 to determine how it could be applied to other datasets. As this scheme seemed very promising, we would be curious to determine how it could apply to other noisy datasets.

Categorical Data Finally, as we focused primarily on regression in this work, we did not investigate the potential application to categorical datasets. It would be both beneficial and interesting to determine the potential utility of these methods on classification problems.

Appendix A

Proofs

As stated in the thesis, we are learning $\rho_x(y)$ from data drawn from $p(x, y)$

Theorem A.1.

$$\mathcal{L} = -\iint_S \log(\rho_x(y))p(x, y)dS \tag{A.1}$$

is minimized (under infinite i.i.d. data) when $\rho_x(y) = p(x, y)$.

Proof. Suppose we are given $p(x, y)$. Then,

$$\mathcal{L} = -\iint_S \log(\rho_x(y))p(x, y)dS \tag{A.2}$$

$$\tag{A.3}$$

Furthermore, $\rho_x(y) = \frac{\partial}{\partial y} F^x(y)$ for some cumulative probability distribution $F^x(y)$. Please note we notate $\frac{\partial}{\partial y} F^x(y)$ as $F^x_y(y)$ for brevity. Thus,

$$\mathcal{L} = -\iint_S \log(F^x_y(y)) p(x,y) dy\, dx. \tag{A.4}$$

By the Euler Lagrange theorem, this function reaches its minimum when

$$\frac{\partial \mathcal{M}}{\partial \rho} - \frac{\partial}{\partial x}\left(\frac{\partial \mathcal{M}}{\partial F^x_x}\right) - \frac{\partial}{\partial y}\left(\frac{\partial \mathcal{M}}{\partial F^x_y}\right) = 0 \tag{A.5}$$

$$\frac{\partial}{\partial y}\left(\frac{\partial \mathcal{M}}{\partial F^x_y}\right) = 0 \tag{A.6}$$

$$\frac{\partial^2}{\partial y\, \partial F^x_y}\left(\log(F^x_y(y)) p(x,y)\right) = 0 \tag{A.7}$$

where $\mathcal{M}$ is the integrand of $\mathcal{L}$. Differentiating, one obtains

$$\frac{\partial}{\partial y}\left(\frac{p(x,y)}{F^x_y(y)}\right) = \frac{F^x_y(y) p_y(x,y) - p(x,y) F^x_{yy}(y)}{(F^x_y(y))^2} = 0 \tag{A.8}$$

Assuming the original distribution has support everywhere,

$$F^x_y(y) p_y(x,y) = p(x,y) F^x_{yy}(y) \tag{A.9}$$

Thus, solving this differential equation yields

$$\frac{p_y(x,y) dy}{p(x,y)} = \frac{dF^x_y(y)}{F^x_y(y)} \tag{A.10}$$

$$p(x,y) = C F^x_y(y) \tag{A.11}$$

Thus,

$$p(x, y) = C\rho_x(y) \tag{A.12}$$

As $p(x, y)$ and $\rho_x(y)$ are both probability distributions (and thus $\int_{-\infty}^{\infty} \rho_x(y)dy = \int_{-\infty}^{\infty} p(x, y)dy = 1$, $C = 1$. Hence, $\rho_x(y) = p(x, y)$ as desired. □

As an optimal learning scheme reaches the global optimum, under an optimal learning scheme, $\rho_x(y)$ would be learned to be $p(x, y)$

Bibliography

[1] Trevor Hastie, Jerome Friedman, and Robert Tisbshirani. *The Elements of statistical learning: data mining, inference, and prediction.* Springer, 2017.

[2] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2015.

[3] Andreas Damianou and Neil Lawrence. Deep gaussian processes. In Carlos M. Carvalho and Pradeep Ravikumar, editors, *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 207–215, Scottsdale, Arizona, USA, 29 Apr–01 May 2013. PMLR. URL http://proceedings.mlr.press/v31/damianou13a.html.

[4] Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3438–3446. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5965-bayesian-dark-knowledge.pdf.

[5] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks, 2015.

[6] Jos Miguel Hernndez-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks, 2015.

[7] Christopher Bishop. Mixture density networks. Technical report, January 1994. URL https://www.microsoft.com/en-us/research/publication/mixture-density-networks/.

[8] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles, 2016.

[9] Alex Graves. Practical variational inference for neural networks. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2348–2356. Curran Associates, Inc., 2011. URL http://papers.nips.cc/paper/4329-practical-variational-inference-for-neural-networks.pdf.

[10] A. Vehtari, S. Sarkka, and J. Lampinen. On mcmc sampling in bayesian mlp neural networks. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, 2000. doi: 10.1109/ijcnn.2000.857855.

[11] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision (IJCV)*, July 2016.

[12] Markus Mathias, Rodrigo Benenson, Marco Pedersoli, and Luc Van Gool. Face detection without bells and whistles. In *European Conference on Computer Vision (ECCV)*, pages 720–735. Springer International Publishing, 2014.

[13] Dimitrios Asteriou and Stephen G. Hall. Arima models and the box-jenkins methodology. *Applied Econometrics*, page 275296, 2016. doi: 10.1057/978-1-137-41547-9_13.

[14] B Ng. Survey of anomaly detection methods. 2006. doi: 10.2172/900157.

[15] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. doi: 10.1109/cvpr.2018.00678.

[16] Dilan Grr and Carl Edward Rasmussen. Dirichlet process gaussian mixture models: Choice of the base distribution. *Journal of Computer Science and Technology*, 25 (4):653664, 2010. doi: 10.1007/s11390-010-9355-8.