THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Variational Mixture of Normalizing Flows

Guilherme P. Grijó Pires

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. Mário Alexandre Teles de Figueiredo

Examination Committee

Chairperson: Prof. Full Name
Supervisor: Prof. Mário Alexandre Teles de Figueiredo
Member of the Committee: Prof. Full Name 3

Month Year

What I cannot create, I do not understand.

- Richard Feynman

Acknowledgments

A few words about the university, financial support, research advisor, dissertation readers, faculty or other professors, lab mates, other friends and family...

Resumo

Inserir o resumo em Português aqui com o máximo de 250 palavras e acompanhado de 4 a 6 palavras-chave...

Palavras-chave: palavra-chave1, palavra-chave2,...

Abstract

Keywords:

Contents

List of Tables

List of Figures

List of Acronyms

ELBO	Evidence Lower Bound
EM	Expectation-Maximization
GMM	Gaussian Mixture Model
MAP	Maximum a-posteriori
MDL	Minimum Description Length
MLE	Maximum Likelihood Estimation
NF	Normalizing Flow
PReLU	Parameterized Rectified Linear Unit
ReLU	Rectified Linear Unit
VAE	Variational Auto-Encoder
VMoNF	Variational Mixture of Normalizing Flows

Chapter 1

Introduction

1.1 Motivation

Neural network based generative models - Variational Autoencoders, Generative Adversarial Networks, Normalizing Flows and their variations - have experienced increased interest and progress in their capabilities, in the last few years.

Less (although some) attention has been given to the extension of such models with discrete structure, such as the one found in Mixture Models. Exploiting such structure, while still being able to benefit from the expressiveness of neural generative models - specifically, Normalizing Flows - is the goal of this work. Doing so will naturally produce an approach which lends itself to clustering, semi-supervised learning, and (multimodal) density estimation.

1.2 Related Work

The work presented here intersects with several active directions of research. In the sense of combining deep neural networks with probabilistic modelling, particularly with the goal of endowing simple probabilistic graphical models with more expressiveness, [8] and [11] propose a framework to use neural network parameterized likelihoods, composed with latent probabilistic graphical models. Still in line with this topic, but with an approach more focused towards clustering and semi-supervised learning, [4] proposes a VAE-inspired model, where the prior is a Gaussian Mixture. [14] describes an unsupervised method for clustering using deep neural networks, which is a task that can also be fulfilled by the model presented in this work.

The following are brief descriptions of the two works that are most related to the present work.

[6], similarly to this work, tries to reconcile Normalizing Flows with multimodal/discrete structure. It does so by partitioning the latent space in disjoint subsets and using a Mixture Model where each component has non-zero weight exclusively within its respective subset. Then, using a set identification function and a piecewise invertible function, a variation of the change-of-variable formula is devised.

[7] also tries to exploit multimodal structure while using Normalizing Flows for expressiveness. How-

ever, while the present work relies on a variational posterior parameterized by a neural network and learns K flows (one for each mixture component), [7] resorts to the use of a latent Mixture of Gaussians as the base distribution for its flow model, and it learns a single flow.

1.3 Objectives

The objectives of the present work are:

- The design of a Normalizing Flow Mixture Model, with a tractable learning procedure
- The empirical analysis of the capabilities of such model, namely in the tasks of:
    - Density Estimation
    - Clustering
    - Semi-supervised learning

1.4 Thesis Outline

In Chapter 2, the concepts on Probabilistic Modelling needed for the remainder of the work are introduced. In Chapter 3, the framework and theoretical background of Normalizing Flows is presented. Chapter 4 describes in detail the model proposed by the present work, and Chapter 5 contains empirical results and their interpretation.

1.5 Notation

The notation used throughout this work is as follows:

- Scalars and vectors are lower-case letters. To differentiate between them, vectors will be present in bold. E.g.: x is a scalar $\mathbf{z}$ is a vector.
- Upper-case letters are matrices.
- For distributions, subscript notation will only be used when the distribution isn't clear from context.
- The operator $\odot$ denotes the element-wise product
- The letter x is preferred for observations
- The letter z is preferred for latent variables
- The letter θ is preferred for parameter vectors

Chapter 2

Probabilistic Modelling

2.1 Introduction

Probabilistic modelling is a set of techniques that leverage probability distributions and random variables to posit, test and refine hypothesis about the behaviour of systems. Given observations of a system, the task of probabilistic modelling normally boils down to finding a probability distribution which:

- Is consistent with the observed data;
- Is consistent with **new**, previously unobserved data, originated in the same system.

This probability distribution is commonly called the *model*. A good model will be a good *emulator* of the true generative process that originated the observed data. In loose terms, this can be summarized as:

$$\text{data} \sim p(\text{data}|\text{hypothesis}^*), \tag{2.1}$$

where hypothesis* is the optimal hypothesis.

Via Bayes' Law, we can write:

$$p(\text{hypothesis}|\text{data}) = \frac{p(\text{data}|\text{hypothesis})p(\text{hypothesis})}{p(\text{data})} \tag{2.2}$$

In practice, a modeller will search for an hypothesis that maximizes (some form, or approximation of) this expression. For simpler problems, this search happens in closed-form, i.e., there is an expression to compute the optimal hypothesis, given data. However, for most real-world problems there is no closed-form solution, and the modeller has to resort to algorithms and approximations, and will only be able to find **local** optima for the above expression, in most cases.

It's also worth noting that there are effectively infinite candidate distributions - each one an hypothesis for how the system at hand generates data. It is common to make use of domain knowledge and assume the true system has a certain intrinsic structure and form, and to use these assumptions to constrain the space of candidate hypothesis. Assumptions about structure usually translate to conditional independence claims between some or all of the observed variables; assumptions about form translate into the use of specific parametric families to govern some or all of the observed variables. These assumptions are commonly connected between themselves (for example, when conjugate likelihood-prior pairs are used).

When parametric forms are used, an hypothesis is uniquely defined by the set of parameters it requires - commonly called θ[1].

2.2 Model Complexity

Intuitively, the dimension of θ is deeply connected with the expressiveness of the distribution. In practice, this translates to the observation that if we make the model[2] expressive enough, it can fit the observed data arbitrarily well. Naïvely, this would be a desirable characteristic to exploit - it's always possible to increase the likelihood by adding parameters to the model. However, increasing model complexity normally comes at the expense of generalization. This phenomenon is commonly referred to as *overfitting*, and there are several angles from which to explain it and interpret it. Namely:

- The classical perspective is that of the bias-variance tradeoff. To understand this, consider the concept of an Hypothesis Class - a set of hypotheses in which, via some procedure, the modeller will search for an hypothesis that is consistent with the observed data, and is expected to generalize to unseen data. Said procedure is what is normally referred to as *fitting* the model to the data. In the case of parametric models, the set of models of a given parametric form, with a parameter-vector of a certain fixed dimension, is an example of an Hypothesis Class. Intuitively, a more complex Hypothesis Class is more likely to contain the true hypothesis (or a good approximation to it). However, the more complex the Hypothesis Class, the larger the search-space - the higher the number of candidate hypothesis. In this sense, an increase in the size of the search-space often translates into an increase of the sensitivity to the problem variables (in the case of learning and inference, this means sensitivity to initialization and to the data used to fit). Conversely, a simpler model will constitute a smaller search-space, hence the search procedure will be less sensitive to initalization and problem variables. However, the true hypothesis (or a good approximation to it) is less likely to be contained in it - precisely because it is a smaller Hypothesis Class. The bias-variance tradeoff is a summary of these observations: a highly complex model is potentially able to achieve a low expected error on observed data (low bias), but will tend to be extremely sensible to small variations on its input (high variance). Conversely, a simpler model will be more robust to variations on its input (low variance), but won't have the same modelling capacity and will produce a larger expected error (high bias).[3]

[1] For the type of models and problems dealt with in this work, I will assume θ is finite, but it's worth noting that there are models for which the dimension of θ *grows* with the dataset size. These are called non-parametric models. They come with their own advantages and disadvantages, which are out of the scope of this work.

[2] Throughout this work I will be using the words *model* and *distribution* almost interchangeably, making it clear when context isn't enough.

[3] The number of parameters is far from being the best measure of complexity of a model. Nevertheless, it is a good proxy to compare model complexity between models of the same parametric family. However, recent work by Belkin et al. [1] shows that modern machine learning contexts, in which the number of parameters is far larger than in classical settings, have to be understood under a measure of model complexity different than the traditional ones. This is because it is now common practice to fit highly overparameterized models to a point of interpolation (close to zero training error), still being able to achieve good

- Andrey Kolmogorov's and Gregory Chaitin's ideas on Algorithmic Information Theory [2], and Kolmogorov complexity are another useful lens through which to regard this question. Consider that data are measurements of phenomena. Modelling is concerned with finding the laws that explain/govern these phenomena. Intuitively, if the laws are as complex as the data they intend to explain, they aren't explaining anything. AIT formalizes this notion by borrowing the concept of *program* to define the generative process by which observed data comes to existence. The complexity of a dataset is then easy to define: it is the size of the **smallest**[4] program that generates the observed data. And the appropriate unit with which to measure the size of a program - and, as we've now seen, the complexity of a dataset - is bits[5]. The parallel between these ideas and the question of overfitting is thus easy to make: a program (or a model and its parameter vector) is useful if it *compresses* the data, intuitively because to do so it leverages the patterns therein, which are the object of interest in the modelling task.

Both of these lines of reasoning make clear that there is a certain balance in complexity that a good model has to achieve: it should be parsimonious enough that it won't overfit, but flexible enough that it is able to properly explain the observed data. There are strategies to make this mathematically objective. Some of those methods are the Bayesian Information Criterion, the Akaike Information Criterion and the Minimum Description Length.

2.3 MAP and ML Estimation

Once the parametric form of the model is defined, the task at hand becomes the discovery of the parameter vector $\boldsymbol{\theta}$ that best explains the observed data, within the defined parametric family.

The naïve (but often the only possible) approach is to maximize what is called the likelihood function, given by:

$$\mathcal{L}(\boldsymbol{\theta}) = p(x|\boldsymbol{\theta}) \tag{2.3}$$

This approach is called *Maximum Likelihood Estimation*. Note that $\mathcal{L}$ is a function of $\boldsymbol{\theta}$. Depending on the model, finding $\boldsymbol{\theta}^*$ - the optimum - can be as simple as computing an analytical expression, or as difficult as using gradient-based methods to optimize a non-convex objective. In the latter case, local optima are usually the best one can expect to obtain.

Another approach, called *Maximum a posteriori*, works by retrieving the mode of the posterior distribution of the parameter-vector, given by:

$$p(\boldsymbol{\theta}|x) = \frac{p(x|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(x)} \tag{2.4}$$

It's easy to see that these two approaches are intimately related. MAP differs from ML by the fact

generalization.

[4] Note the emphasis on ”smallest” - this is because any program can be made arbitrarily redundant, and thus arbitrarily large.

[5] Or the basic unit of memory of the computer where the data generating program would run

that it employs the prior $p(\boldsymbol{\theta})$ to give different weights to different hypothesis (i.e., different instances of $\boldsymbol{\theta}$). This is useful if there is domain knowledge available that can be encoded in the prior.

It is worth noting that ML is a special case of MAP, when the prior is uniform.

2.4 Structure and Latent Variables

In some cases, one might want to leverage some available domain knowledge. This often translates into assuming that there is some latent structure in the data. This structure is commonly encoded into latent variables and their influence over the observable variables.

In this scenario, we become interested in the distribution given by $p(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\theta_x}, \boldsymbol{\theta_z})$, where z is the latent variable, $\boldsymbol{\theta_x}$ is the parameter vector for the distribution over $\boldsymbol{\mathcal{X}}$, and $\boldsymbol{\theta_z}$ is the parameter vector for the distribution over $\boldsymbol{\mathcal{Z}}$.

For structure and latent variables to be useful, we normally make the additional assumption that we have the ability of factorizing their joint distribution in ways that make it tractable. If we have a dataset $\boldsymbol{X}$, with N i.i.d. samples $\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, ..., \boldsymbol{x_N}$ and N latent variables $\boldsymbol{z_1}, \boldsymbol{z_2}, \boldsymbol{z_3}, ..., \boldsymbol{z_N}$, one common factorization is:

$$p(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{\theta}) = \Big(\prod_{i=1}^{N} p(\boldsymbol{x_i}|\boldsymbol{z_i}, \boldsymbol{\theta_x})p(\boldsymbol{z_i}|\boldsymbol{\theta_z})\Big)p(\boldsymbol{\theta_x})p(\boldsymbol{\theta_z}). \tag{2.5}$$

It's also possible that the samples in $\boldsymbol{X}$ have some sort of causal relation, for instance if they occur ordered in time. In this case, they are not i.i.d. One way to encode this assumption is to posit an *autoregressive* model, i.e., a model in which a random variable depends on the realizations of the variables that come before it. If each random variable depends solely on the random variable that precedes it, this is called a Markov Model. A common variation of the Markov Model is the Hidden Markov Model, where the autoregressive part of the model is present only in the latent variables:

$$p(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{\theta}) = p(\boldsymbol{z_1})\Big(\prod_{i=2}^{N} p(\boldsymbol{x_i}|\boldsymbol{z_i}, \boldsymbol{\theta_x})p(\boldsymbol{z_i}|\boldsymbol{z_{i-1}}, \boldsymbol{\theta_z})\Big)p(\boldsymbol{\theta_x})p(\boldsymbol{\theta_z}). \tag{2.6}$$

These are merely examples of models with different structure assumptions encoded into them. Normally, if the structure has a certain regularity, it's possible to exploit it to obtain tractable (approximate) inference and estimation methods. This notion is explored in the following section, for a subset of the family of models described by 2.5.

2.5 Mixture Models and the EM Algorithm

Mixture Models are a subset of the structure ”family” described in 2.5, and they have a central role in this work.

In a Mixture Model there is a discrete latent variable z_i which selects one of K components from which an observation $\boldsymbol{x_i}$ will be sampled. This can be summarized as:

$$z_i \sim p(z_i|\boldsymbol{\pi}) \tag{2.7}$$

$$x_i \sim p(\boldsymbol{x_i}|z_i) \tag{2.8}$$

The probability of $\boldsymbol{x_i}$ being sampled from component k (that is, the proability of $z_i = k$) is commonly referred to as the *weight* of component k.

It's common to assume that all of the K components are part of the same parametric family. In that case, we can rewrite the above as:

$$z_i \sim p(z_i|\boldsymbol{\pi}) \tag{2.9}$$

$$x_i \sim p(\boldsymbol{x_i}|\boldsymbol{\theta}_{z_i}), \tag{2.10}$$

where it is made evident that the discrete variable z_i is selecting the **parameter vector** to be used for sample x_i.

The most discussed mixture model is the Gaussian Mixture Model, in which the K components of the model are Gaussian distributions.

The EM Algorithm

The Expectation-Maximization algorithm is the most commonly used algorithm to fit Mixture Models[6].

The starting point of EM is the realisation that if all variables were observed, it would be easy to apply ML or MAP estimation for the parameters. Given that, the algorithm can be generally described as an alternation between two steps:

- **E-step**: infer the most probable values of the unobserved variables. (In the case of Mixture Models, this corresponds to inferring the discrete variables that select the component from which each observed data point was sampled. This can be roughly thought of as a cluster assignment).
- **M-step**: given the observed variables and the values inferred on the previous step, optimize the model parameters. (In the case of Mixture Models, this correponds to inferring the parameters of each component, and its weight).

A more rigorous description of the procedure follows. Consider the following expression:

$$\ell_c(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{x_i}, z_i|\boldsymbol{\theta}) \tag{2.11}$$

This is the complete data log-likelihood. Like the likelihood function, it is a function of $\boldsymbol{\theta}$, which is easy to compute, given all the $\boldsymbol{x_i}$ and z_i. However, the z_i aren't observed.

To overcome this, let us define the expected complete data log likelihood:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t-1)}) = \mathbb{E}_{Z|X,\boldsymbol{\theta}^{(t-1)}}[\ell_c(\boldsymbol{\theta})|X, Z, \boldsymbol{\theta}^{(t-1)}], \tag{2.12}$$

[6] Although it can be used to fit other types of models.

where $\boldsymbol{\theta}^{(t)}$ represents the value of $\boldsymbol{\theta}$ at time step t of the fitting procedure.

Note that the expectation is w.r.t. Z, given X and $\boldsymbol{\theta}^{(t-1)}$. It is the expected value of $\ell_c(\boldsymbol{\theta})$, given the parameter values obtained at the previous step of the the algorithm. Depending on the nature of Z, this expectation can be obtained either in closed form or approximated, for instance via samples of z_i (if a sampling procedure is available).

In the case of Mixture Models, where the z_i are instances of a categorical random variable, the expression for $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t-1)})$ can be made simpler as such:

$$
\begin{aligned}
Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t-1)}) &= \mathbb{E}_{Z|X,\boldsymbol{\theta}^{(t-1)}}[\ell_c(\boldsymbol{\theta})|X, Z, \boldsymbol{\theta}^{(t-1)}] && (2.13)\\
&= \mathbb{E}_{Z|X,\boldsymbol{\theta}^{(t-1)}}[\sum_{i=1}^{N} \log p(x_i, z_i|\boldsymbol{\theta})] && (2.14)\\
&= \mathbb{E}_{Z|X,\boldsymbol{\theta}^{(t-1)}}[\sum_{i=1}^{N} \log \prod_{k=1}^{K} \pi_k p(x_i|\boldsymbol{\theta}_k)^{\mathbb{I}(z_i=k)}] && (2.15)\\
&= \mathbb{E}_{Z|X,\boldsymbol{\theta}^{(t-1)}}[\sum_{i=1}^{N}\sum_{k=1}^{K} \log \left(\pi_k p(x_i|\boldsymbol{\theta}_k)^{\mathbb{I}(z_i=k)}\right)] && (2.16)\\
&= \mathbb{E}_{Z|X,\boldsymbol{\theta}^{(t-1)}}[\sum_{i=1}^{N}\sum_{k=1}^{K} \mathbb{I}(z_i=k) \log \left(\pi_k p(x_i|\boldsymbol{\theta}_k)\right)] && (2.17)\\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} \underbrace{\mathbb{E}_{Z|X,\boldsymbol{\theta}^{(t-1)}}[\mathbb{I}(z_i=k)]}_{\text{Let this quantity be represented by } r_{ik}} \log \left(\pi_k p(x_i|\boldsymbol{\theta}_k)\right) && (2.18)\\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik} \log \left(\pi_k p(x_i|\boldsymbol{\theta}_k)\right) && (2.19)\\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik} \log \pi_k + \sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik} \log p(x_i|\boldsymbol{\theta}_k) && (2.20)\\
& && (2.21)
\end{aligned}
$$

The quantity defined above as r_{ik} is referred to as the *responsability*. It is trivial to arrive at a closed-form expression for it, given the value of $\boldsymbol{\theta}$ arrived at on the previous iteration, i.e. $\boldsymbol{\theta}^{(t-1)}$:

$$
\begin{aligned}
r_{ik} &= p(z_i = k|x_i\ ; \boldsymbol{\theta}^{(t-1)}) && (2.22)\\
&= \frac{p(z_i = k, x_i\ ; \boldsymbol{\theta}^{(t-1)})}{p(x_i\ ; \boldsymbol{\theta}^{(t-1)})} && (2.23)\\
&= \frac{p(z_i = k, x_i\ ; \boldsymbol{\theta}^{(t-1)})}{p(x_i\ ; \boldsymbol{\theta}^{(t-1)})} && (2.24)\\
&= \frac{p(z_i = k, x_i\ ; \boldsymbol{\theta}^{(t-1)})}{\sum_{k'} p(z_i = k', x_i\ ; \boldsymbol{\theta}^{(t-1)})} && (2.25)\\
&= \frac{\pi_k p(x_i|\boldsymbol{\theta}_k^{(t-1)})}{\sum_{k'} \pi_{k'} p(x_i|\boldsymbol{\theta}_{k'}^{(t-1)})} && (2.26)
\end{aligned}
$$

The EM algorithm for Mixture Models then becomes an alternation between the following two steps:

- **E-step**: Compute the responsabilies r_{ik} for each x_i.

- **M-step**: Given r_{ik}, solve the following optimization problem:

$$\boldsymbol{\theta}^{(t)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t-1)}) \qquad \underbrace{+\log p(\boldsymbol{\theta})}_{\text{Optional, if we want to do MAP estimation}} \qquad , \tag{2.27}$$

  which can have a closed-form solution in some of the simplest Mixture Models, like Gaussian Mixture Models, but can require a gradient-based optimization procedure for more flexible models.

2.6 Approximate Inference

Take the expression $p(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\theta})$. For simplicity, let us consider θ as part of the latent variables $\boldsymbol{z}$. This means that the model is simply written as the joint distribution: $p(\boldsymbol{x}, \boldsymbol{z})$. *Inference*[7] is the task of finding the most probable $\boldsymbol{z}$ after having observed $\boldsymbol{x}$. Specifically, the goal is to find the posterior distribution of $\boldsymbol{z}$, given $\boldsymbol{x}$, i.e.: $p(\boldsymbol{z}|\boldsymbol{x})$.

Recall Bayes' Law:

$$\begin{aligned} p(\boldsymbol{z}|\boldsymbol{x}) &= \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})} && (2.28) \\ &= \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{\int p(\boldsymbol{x}|\boldsymbol{z'})p(\boldsymbol{z'})d\boldsymbol{z'}} && (2.29) \end{aligned}$$

For the vast majority of cases, the integral on the denominator will be intractable. To overcome this difficulty we normal resort to two families of methods: Monte-Carlo methods - which are out of the scope of this work - and Variational methods.

2.6.1 Variational Methods

Variational methods work by turning the problem of integration into one of optimization. They propose a family of parametric distributions, and then optimize the parameters so as to minimize the "distance" between the approximate (normally called "variational") distribution and the distribution of interest.

There are two ways to derive the most commonly used objective function for this problem, which will be detailed in the two following subsections.

Kullback-Leibler Divergence

The Kullback-Leibler divergence is a measure[8] of the distance between two probability distributions p, and q. It is given by:

[7] If we hadn't collapsed $\boldsymbol{\theta}$ into $\boldsymbol{z}$ and were instead handling it separately, we would call **inference** to the task of finding $\boldsymbol{z}$ and **learning** to the task of finding $\boldsymbol{\theta}$

[8] Note that the KL divergence isn't symmetric and as such I haven't called it a *metric*

$$KL(q||p) = \int q \log \frac{q}{p} \tag{2.30}$$

In the setting of inference, p is the posterior $p(\boldsymbol{z}|\boldsymbol{x})$ and q is a distribution in some parametric family, with parameters ϕ, i.e., $q(\boldsymbol{z};\phi)$. However, it is clear that we can't compute the Kullback-Leibler directly, because it requires the knowledge of both distributions, and finding $p(z|\boldsymbol{x})$ is precisely the task at hand. Let us expand the KL divergence expression:

$$\begin{align}
KL(q||p) &= \int q(\boldsymbol{z})(\log q(\boldsymbol{x}) - \log p(\boldsymbol{z}|\boldsymbol{x}))dz \tag{2.31}\\
&= \int q(\boldsymbol{z})(\log q(\boldsymbol{z}) - (\log p(\boldsymbol{x},\boldsymbol{z}) - \log p(\boldsymbol{x})))d\boldsymbol{z} \tag{2.32}\\
&= \mathbb{E}_q[\log q(\boldsymbol{z})] - \mathbb{E}_q[\log p(\boldsymbol{x},\boldsymbol{z})] + \mathbb{E}_q[\log p(\boldsymbol{x})] \tag{2.33}\\
&= \mathbb{E}_q[\log q(\boldsymbol{z})] - \mathbb{E}_q[\log p(\boldsymbol{x},\boldsymbol{z})] + \log p(\boldsymbol{x}) \tag{2.34}\\
& \tag{2.35}
\end{align}$$

The last term is constant w.r.t $q(\boldsymbol{z})$. In that sense, for a fixed $p(\boldsymbol{x})$, minimizing the KL divergence is equivalent to minimizing

$$\mathbb{E}_q[\log q(\boldsymbol{z})] - \mathbb{E}_q[\log p(\boldsymbol{x},\boldsymbol{z})], \tag{2.36}$$

which is equivalent to maximizing

$$\mathbb{E}_q[\log p(\boldsymbol{x},\boldsymbol{z})] - \mathbb{E}_q[\log q(\boldsymbol{z})]. \tag{2.37}$$

This quantity is commonly refered to as ELBO - Evidence Lower BOund. It can be rewritten as:

$$\begin{align}
ELBO(q) &= \mathbb{E}_q[\log p(\boldsymbol{x},\boldsymbol{z})] - \mathbb{E}_q[\log q(\boldsymbol{z})] \tag{2.38}\\
&= \mathbb{E}_q[\log p(\boldsymbol{x}|\boldsymbol{z})] + \mathbb{E}_q[\log p(\boldsymbol{z})] - \mathbb{E}_q[\log q(\boldsymbol{z})] \tag{2.39}
\end{align}$$

In this form, each term of the ELBO has an easily interpretable role:

- $\mathbb{E}_q[\log p(\boldsymbol{x}|\boldsymbol{z})]$ tries to maximize the conditional likelihood of $\boldsymbol{x}$. That can be seen as assigning high probability mass to values of $\boldsymbol{z}$ that *explain* $\boldsymbol{x}$ well.
- $\mathbb{E}_q[\log p(\boldsymbol{z})]$ is the symmetric of the crossentropy between $q(\boldsymbol{z})$ and $p(\boldsymbol{z})$. Maximizing this quantity is equivalent of minimizing that crossentropy. This can be regarded as a regularizer that discourages $q(\boldsymbol{z})$ of being too different from the prior $p(\boldsymbol{z})$.
- $-\mathbb{E}_q[\log q(\boldsymbol{z})]$ is the entropy of $q(\boldsymbol{z})$. Maximizing this term incentivizes the probability mass of $q(\boldsymbol{z})$ to be spread out: another form of regularization.

A lower bound on $\log p(\boldsymbol{x})$

Another way of approaching the intractable posterior is to start by stating that our inherent goal is to maximize $p(\boldsymbol{x})$, or equivalently $\log p(\boldsymbol{x})$. Given that, consider the following:

$$\begin{aligned}
\log p(\boldsymbol{x}) &= \log \int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} && (2.40)\\
&= \log \int q(\boldsymbol{z}) \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z})} d\boldsymbol{z} && (2.41)\\
&= \log \mathbb{E}_q[\frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z})}] && (2.42)\\
&\geq \mathbb{E}_q[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z})}] && (2.43)\\
&\geq \mathbb{E}_q[\log p(\boldsymbol{x}, \boldsymbol{z})] - \mathbb{E}_q[q(\boldsymbol{z})] && (2.44)
\end{aligned}$$

To understand this derivation, consider Jensen's inequality, given (in one of its forms) by:

$$\phi(\mathbb{E}[X]) \leq \mathbb{E}[\phi(X)], \qquad (2.45)$$

where $\phi(.)$ is a convex function.

If $\xi(.)$ is a concave function, then $-\xi(.)$ is a convex function, and we obtain the reverse inequality (substituting $\phi(.)$ with $-\xi(.)$ in the inequality in 2.45):

$$\begin{aligned}
-\xi(\mathbb{E}[X]) &\leq \mathbb{E}[-\xi(X)] && (2.46)\\
\xi(\mathbb{E}[X]) &\geq \mathbb{E}[\xi(X)] && (2.47)
\end{aligned}$$

This form is the most useful for us, since $\log$ is a concave function. Using this, the step between 2.42 and 2.43 is made obvious.

Note that the right-hand side of 2.44 is the same quantity we arrived at in 2.37, and that it is a lower-bound on the quantity we want to maximize, and so we want to maximize it. It's worth noting that when $q(\boldsymbol{z}) = p(\boldsymbol{z}|\boldsymbol{x})$, the bound is tight.

Chapter 3

Normalizing Flows

3.1 Introduction

The best known and studied probability distributions are rarely expressive enough for real-world datasets. However, they have properties that make them amenable to work with, for instance: tractable parameter estimation, closed-form likelihood functions, and simple sampling procedures.

As has been described, one way to obtain more expressive models is to assume the existence of latent variables, leverage certain factorization structures, and to use well-known distributions for the individual factors of the product that constitutes the model's joint distribution. By using these structures and choosing specific combinations of distributions (namely, conjugate prior-likelihood pairs), these models are able to stay tractable - normally via bespoke estimation/inference/learning algorithms.

Another approach to obtaining expressive probabilistic models is to apply transformations to a simple distribution, and use the Change of Variables formula to compute probabilities in the transformed space. This is the basis of Normalizing Flows, an approach proposed by Rezende and Mohamed in [13], and which has since evolved and developed into the basis of multiple SoA techniques for density estimation ([10], [5], [3], [12]).

3.2 Change of Variables

Given a probability distribution $p(\boldsymbol{z})$, with probability density function $f_Z(.)$, and a bijective and continuous function g, it's possible to write an expression for the probability density function $f_X(.)$ of the random variable x that is obtained by applying g to samples of $p(\boldsymbol{z})$:

$$\text{if } \boldsymbol{z} \sim p(\boldsymbol{z}) \tag{3.1}$$

$$\text{and } \boldsymbol{x} = g(\boldsymbol{z}) \tag{3.2}$$

$$\text{then } f_X(\boldsymbol{x}) = f_Z(g^{-1}(\boldsymbol{x}))\Big|\det\left(\frac{d}{d\boldsymbol{x}}g^{-1}(\boldsymbol{x})\right)\Big| \tag{3.3}$$

For this to be useful, some parts of the above expression have to be easily computable:

- $f_Z(\boldsymbol{z})$ - the starting[1] distribution's probability density function. It is assumed that there is a closed-form expression to compute this. In practice, this is normally one of the basic distributions (Gaussian, Uniform, etc.)
- $\det\left(\frac{d}{d\boldsymbol{x}} g^{-1}(\boldsymbol{x})\right)$ - the determinant of the Jacobian matrix of $g^{-1}(.)$. For most transformations this is not "cheap" to compute. As will be shown, the main challenge o thef Normalizing Flows framework is to find transformations that are expressive and for which the determinants of their Jacobian matrices are "cheap" to compute.

3.3 Normalizing Flows

Let us have L transformations h_ℓ that fulfill the two requirements listed above, and let $\boldsymbol{z_\ell}$ be the result of applying transformation $h_{\ell-1}$, with the exception of $\boldsymbol{z_0}$, which is obtained by sampling from $p(\boldsymbol{z_0})$, the base distribution. Furthermore, let g be the composition of the L transformations.

Applying the Change of Variables formula:

$$\text{if } \boldsymbol{z_0} \sim p(\boldsymbol{z_0}) \tag{3.4}$$

$$\text{and } \boldsymbol{x} = h_{L-1} \circ h_{L-2} \circ ... \circ h_0(\boldsymbol{z_0}) \tag{3.5}$$

$$\text{then } f_X(\boldsymbol{x}) = f_Z(g^{-1}(\boldsymbol{x}))\Big|\det\left(\frac{d}{d\boldsymbol{x}} g^{-1}(\boldsymbol{x})\right)\Big| \tag{3.6}$$

$$= f_Z(g^{-1}(\boldsymbol{x}))\prod_{\ell=0}^{L-1}\Big|\det\left(\frac{d}{d\boldsymbol{z_{\ell+1}}} h_\ell^{-1}(\boldsymbol{z_{\ell+1}})\right)\Big| \tag{3.7}$$

$$= f_Z(g^{-1}(\boldsymbol{x}))\prod_{\ell=0}^{L-1}\Big|\det\left(\frac{d}{d\boldsymbol{z_\ell}} h_\ell\Big(h_\ell^{-1}(\boldsymbol{z_{\ell+1}})\Big)\right)\Big|^{-1} \tag{3.8}$$

Replacing $h_\ell^{-1}(\boldsymbol{z_{\ell+1}}) = \boldsymbol{z_\ell}$ in 3.8:

$$f_X(\boldsymbol{x}) = f_Z(g^{-1}(\boldsymbol{x}))\prod_{\ell=0}^{L-1}\Big|\det\left(\frac{d}{d\boldsymbol{z_\ell}} h_\ell\Big(\boldsymbol{z_\ell}\Big)\right)\Big|^{-1} \tag{3.9}$$

$$\log f_X(\boldsymbol{x}) = \log f_Z(g^{-1}(\boldsymbol{x})) - \sum_{\ell=0}^{L-1}\log\Big|\det\left(\frac{d}{d\boldsymbol{z_\ell}} h_\ell(\boldsymbol{z_\ell})\right)\Big| \tag{3.10}$$

Depending on the task, one might prefer to replace the second term in 3.10 with a sum of log-abs-determinants of the Jacobians of the inverse transformations. This switch would imply replacing the minus sign before the sum with a plus sign:

$$\log f_X(\boldsymbol{x}) = \log f_Z(g^{-1}(\boldsymbol{x})) + \sum_{\ell=0}^{L-1}\log\Big|\det\left(\frac{d}{d\boldsymbol{z_{\ell+1}}} h_\ell^{-1}(\boldsymbol{z_{\ell+1}})\right)\Big| \tag{3.11}$$

With this expression, gradient-based MLE becomes feasible. Moreover, sampling from the resulting distribution is simply achieved by sampling from the base distribution and applying the chain of

[1] From here on the starting distribution will be referred to as *base distribution*

transformations. Because of this, Normalizing Flows lend themselves to be used as flexible variational posteriors, in Variational Inference settings.

3.4 Examples of transformations

3.4.1 Affine Transformation

The Affine transformation is the simplest transform that can be applied. This transformation can stretch, shrink, rotate and translate space. It is simply achieved by the multiplication by a matrix A and summation of a bias vector $\boldsymbol{b}$:

$$\boldsymbol{z} \sim p(\boldsymbol{z}) \tag{3.12}$$

$$\boldsymbol{x} = A\boldsymbol{z} + \boldsymbol{b} \tag{3.13}$$

The Jacobian of this transformation is simply the determinant of the matrix A.

However, in general, computing the determinant of a $N \times N$ matrix has a complexity of $\mathcal{O}(N^3)$. For that reason, it is common to use matrices with a certain structure which makes their determinants easier to compute. For instance, if A is triangular, its determinant is simply the product of its diagonal's elements. The downside of using matrices that are constrained to a certain structure is that they correspond to less flexible transformations.

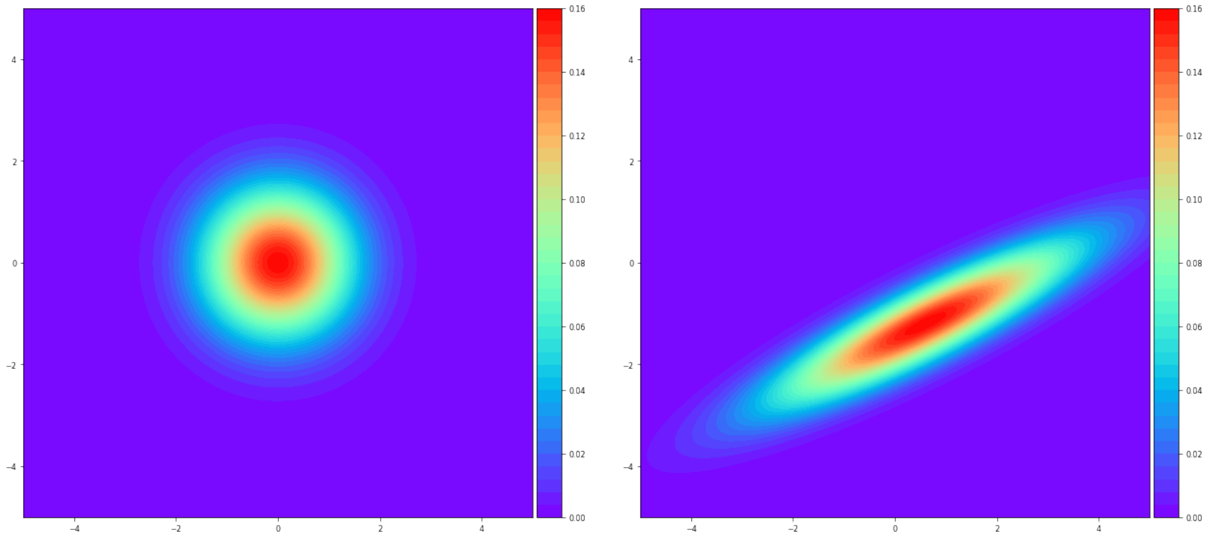
It is possible, however, to design Affine transformations whose Jacobian determinants are of $\mathcal{O}(N)$ complexity and that are more expressive than simple triangular matrices. In [10], one such transformation is proposed. It constrains the matrix A to be decomposable as $A = PL\big(U + diag(\boldsymbol{s})\big)$, where $diag(\boldsymbol{s})$ is a diagonal matrix whose diagonal's elements are the values of vector $\boldsymbol{s}$. The following additional constrains are in place:

- P is a permutation matrix
- L is a lower triangular matrix, with ones on its diagonal
- U is an upper triangular matrix, with zeros on its diagonal

Given these constraints, the determinant of the matrix A is simply the product of the elements of $\boldsymbol{s}$.

3.4.2 PReLU Transformation

Intuitively, introducing non-linearity endows Normalizing Flows with more flexibility to represent complex distributions. This can be done in similar fashion to the activation functions of neural networks. One example of that, is the Parameterized Rectified Linear Unit transformation. It is defined in the following



(a) (b)

Figure 3.1: (a) Density of a Gaussian distribution with $\mu = [0, 0]$ and $\Sigma = I$ (b) Density of the distribution that results from applying some affine transformation to the Gaussian distribution in (a)

manner, for a d-dimensional input:

$$f_i(z_i) = \begin{cases} z_i, & \text{if } z_i \geq 0 \\ \alpha z_i, & \text{otherwise} \end{cases} \tag{3.14}$$

$$f(\boldsymbol{z}) = [f_0(z_0), f_1(\boldsymbol{z_1}), ..., f_d(z_d)] \tag{3.15}$$

Note that in order for the transformation to be invertible, it is necessary that $\alpha > 0$.

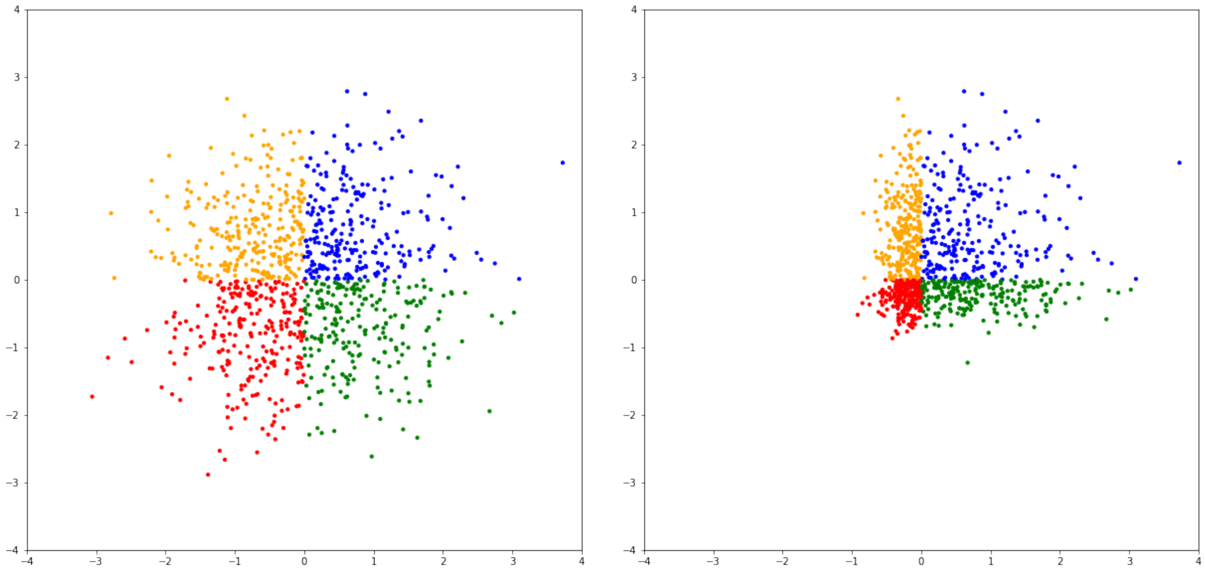
Let us define an auxiliary function $j(.)$ s.t.:

$$j(z_i) = \begin{cases} 1, & \text{if } z_i \geq 0 \\ \alpha, & \text{otherwise} \end{cases} \tag{3.16}$$

It's trivial to see that the jacobian of the transformation is a diagonal matrix, whose diagonal elements are $j(z_i)$:

$$A = \begin{bmatrix} j(z_0) & & & \\ & j(\boldsymbol{z_1}) & & \\ & & \ddots & \\ & & & j(z_d) \end{bmatrix} \tag{3.17}$$

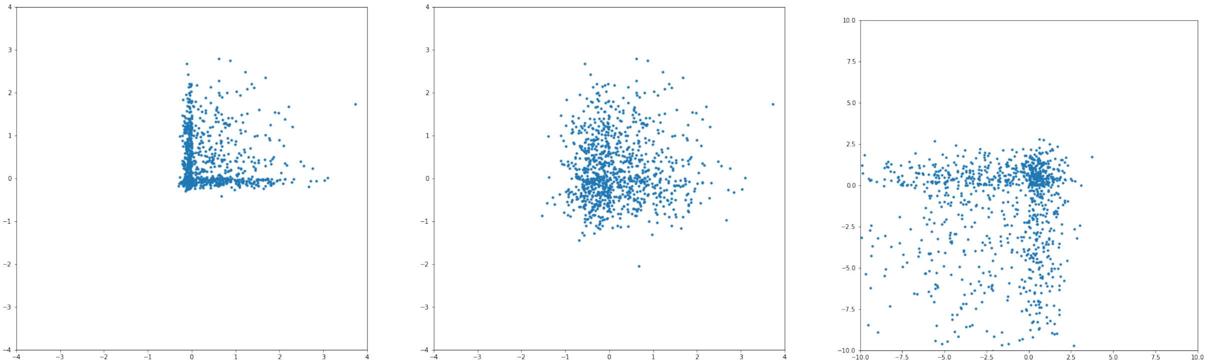
With that, it is easy to arrive at the log-abs-determinant of this transformantion's jacobian, which is given by $\sum_{i=0}^{d} \log \left| j(z_i) \right|$



(a)

(b)

Figure 3.2: (a) Samples from of a Gaussian distribution with $\mu = [0, 0]$ and $\Sigma = I$. The samples are colored according to the quadrant they belong to. (b) Samples from the distribuion in a) transformed by a PReLU transformation.



(a)

(b)

(c)

Figure 3.3: Samples from a Gaussian with $\mu = [0, 0]$ and $\Sigma = I$, transformed by PReLU transformations with different α parameters. (a) $\alpha = 0.1$ (b) $\alpha = 0.5$ (c) $\alpha = 5$

3.4.3 Batch-Norm Transformation

In [5], the authors propose a Batch-Norm transformation, similar to the Batch-Norm layer normally used in neural networks. This transform simply applies a rescaling, given as a function the batch mean $\tilde{\mu}$ and variance $\tilde{\sigma}^2$:

$$f(z) = \frac{z - \tilde{\mu}}{\sqrt{\tilde{\sigma}^2 + \epsilon}}, \tag{3.18}$$

where $\epsilon \ll 1$ is a term used to ensure that there never is a division by zero.

This transformation's Jacobian is trivial:

$$\prod \frac{1}{\sqrt{\tilde{\sigma}_i^2 + \epsilon}} \tag{3.19}$$

3.4.4 Affine Coupling Transformation

As mentioned previously, one of the active research challenges within the Normalizing Flows framework is the search and design of transformations that are expressive and whose Jacobians are not computationally heavy. One brilliant example of such transformations was proposed by Dinh, Sohl-Dickstein, and Bengio in [5], and is called Affine Coupling Layer.

This transformation is characterized by two **arbitrary** functions $s(.)$ and $t(.)$, as well as a mask that splits an input z of dimension D into two parts, $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$. In practice, $s(.)$ and $t(.)$ are neural networks, whose parameters will be optimized so as to make the transformation approximate the desired output distribution. The outputs of $s(.)$ and $t(.)$ need to have the same dimension as $\boldsymbol{z_1}$. This should be taken into account when designing the mask and the functions $s(.)$ and $t(.)$.

It is defined as:

$$\boldsymbol{x_1} = \boldsymbol{z_1} \odot \exp\big(s(\boldsymbol{z_2})\big) + t(\boldsymbol{z_2}) \tag{3.20}$$

$$\boldsymbol{x_2} = \boldsymbol{z_2} \tag{3.21}$$

To see why this transformation is suitable to being used within the framework of Normalizing Flows, let us derive its Jacobian.

- $\frac{\partial \boldsymbol{x_2}}{\partial \boldsymbol{z_2}} = \mathbb{I}$ is trivial, because $\boldsymbol{x_2} = \boldsymbol{z_2}$.
- $\frac{\partial \boldsymbol{x_2}}{\partial \boldsymbol{z_1}}$ is a matrix of zeros, and it is also trivial, because $\boldsymbol{x_2}$ does not depend on $\boldsymbol{z_1}$.
- $\frac{\partial \boldsymbol{x_1}}{\partial \boldsymbol{z_1}}$ is a diagonal matrix, whose diagonal is simply given by $\exp\big(s(\boldsymbol{z_2})\big)$, since those values are constant w.r.t $\boldsymbol{z_1}$ and they are multiplying each element of $\boldsymbol{z_1}$.
- $\frac{\partial \boldsymbol{x_1}}{\partial \boldsymbol{z_2}} = 1$ is not needed for our purposes, as will become clear ahead.

Writing the above in matrix form:

$$J_{f(z)} = \begin{bmatrix} \dfrac{\partial \boldsymbol{x_1}}{\partial \boldsymbol{z_1}} & \dfrac{\partial \boldsymbol{x_1}}{\partial \boldsymbol{z_2}} \\ \dfrac{\partial \boldsymbol{x_2}}{\partial \boldsymbol{z_1}} & \dfrac{\partial \boldsymbol{x_2}}{\partial \boldsymbol{z_2}} \end{bmatrix} \tag{3.22}$$

$$= \begin{bmatrix} \exp\big(s(\boldsymbol{z_2})\big) & \dfrac{\partial \boldsymbol{x_1}}{\partial \boldsymbol{z_2}} \\ \mathbf{0} & \mathbb{I} \end{bmatrix} \tag{3.23}$$

The Jacobian matrix is triangular. Its determinant - the only thing we need, in fact - is therefore easy to compute: it is simply the product of the diagonal. Moreover, part of the diagonal is simply composed of ones. The determinant, and the log-abs-determinant become:

$$det\big(J_{f(z)}\big) = \prod_i \exp\big(s(\boldsymbol{z_2}^{(i)})\big) \tag{3.24}$$

$$\log\Big|det\big(J_{f(z)}\big)\Big| = \sum_i s(\boldsymbol{z_2}^{(i)}), \tag{3.25}$$

where $\boldsymbol{z_2^{(i)}}$ is the i-th element of $\boldsymbol{z_2}$.

Since a single Affine Coupling Layer doesn't transform all of the elements in $\boldsymbol{z}$, in practice several layers are composed, and each layer's mask is changed so as to make all dimensions affect each other. This can be done for instance with a checkerboard pattern, which alternates for each layer. In the case of image inputs, the masks can operate at the channel level.

3.5 Fitting Normalizing Flows

Generally speaking, Normalizing Flows can be used in one of two scenarios: (direct) density estimation, where the goal is to optimize the NF's parameters so as to make the model approximate the distribution of some observed data; in a variational inference scenario, as way to have a flexible variational posterior (i.e. $q(\boldsymbol{z})$ in the terminology used in the previous chapter). The second scenario is out of the scope of this work.

The task of density estimation with Normalizing Flows reduces to finding the optimal parameters of a parametric model. As mentioned in 2.3, there are two ways to go about estimating the parameters of a parametric model, given data: MLE and MAP. In the case of Normalizing Flows, MLE is the usual approach[2]. To fit a Normalizing Flow via MLE, a gradient based optimizer is used to minimize $\hat{\mathcal{L}}(\boldsymbol{\theta}) = -\mathbb{E}[\log p(\boldsymbol{x}|\boldsymbol{\theta})]$

[2] In theory it is possible to place a prior on the NF's parameters and do MAP estimation. To accomplish this, similar strategies to those used in Bayesian Neural Networks would have to be used.

Chapter 4

Variational Mixture of Normalizing Flows

4.1 Introduction

As mentioned in sections 2.4 and 2.5, the ability of leveraging domain knowledge to endow a probabilistic model with structure is often useful. The goal of this work is to devise a model that combines the flexibility of Normalizing Flows with the ability to exploit class-membership structure. Specifically, such model would be able to learn K Normalizing Flows, each responsible for one of K clusters in a dataset.

4.2 Model Definition

Let us define a Mixture Model as in 2.5, where each of the K components is a Normalizing Flow. For simplicity, consider that all of the K Normalizing Flows have the same architecture [1], i.e., they are all composed of the same stack of transformations, but they each have their own parameters.

Additionally, let $q(z|\boldsymbol{x};\gamma)$ be a neural network with a softmax output, with parameters γ. This network will receive as input an instance from the data, and produce the probability of that instance belonging to each of the K classes.

Recall the Evidence Lower Bound given in 2.37[2]:

$$\text{ELBO} = \mathbb{E}_q[\log p(\boldsymbol{x}, z)] - \mathbb{E}_q[\log q(z|\boldsymbol{x})]$$

[1] This is not a requirement, and in cases where we have classes with different levels of complexity, we can have components with different architectures. However, the training procedure does not guarantee that that the most flexible Normalizing Flow is ”allocated” to the most complex cluster. This is definitely an interesting direction for future research.

[2] Here the dependence of q on x is made explicit

Let us rearrange it:

$$\text{ELBO} = \mathbb{E}_q[\log p(\boldsymbol{x}|z)] + \mathbb{E}_q[\log p(z)] - \mathbb{E}_q[\log q(z|\boldsymbol{x})] \tag{4.1}$$

$$\text{ELBO} = \mathbb{E}_q[\log p(\boldsymbol{x}|z) + \log p(z) - \log q(z|\boldsymbol{x})] \tag{4.2}$$

Since $q(z|\boldsymbol{x})$ is given by the forward-pass of a neural network, and is therefore straightforward to obtain, the expectation in 4.2 is given by computing the expression inside the expectation for each possible value of z, and summing the obtained values, weighed by the probabilities given by the variational posterior. Thus, the whole ELBO is easy to compute, provided that each of the terms inside the expectation is itself easy to compute. Let us consider each of those terms:

- $\log p(\boldsymbol{x}|z)$ is the log-likelihood of $\boldsymbol{x}$ under the Normalizing Flow indexed by z. It was shown in the previous chapter how to compute this.
- $\log p(z)$ is the log-prior of the component weights. For simplicity, let us assume this is set by the modeller. When nothing is known about the component weights, the best assumption is that they are uniform. Nevertheless, as will be shown empirically, this too can be optimized.
- $-\log q(z|\boldsymbol{x})$ is the negative logarithm of the output of the encoder.

For a better intuition about each of these terms, it is useful to review the last paragraph of the first subsection of section 2.6.1.

I call this model Variational Mixture of Normalizing Flows (VMoNF). For an overview of the model, consider figures 4.1 and 4.2

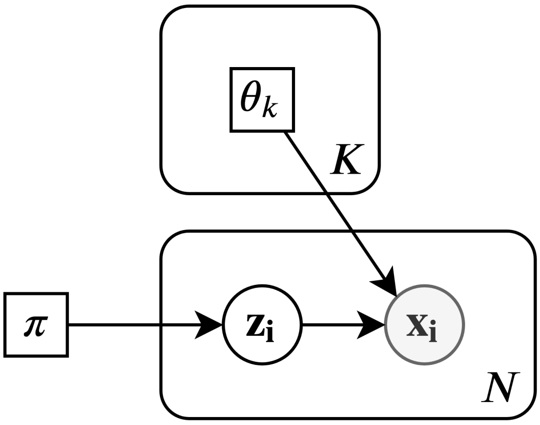


Figure 4.1: Plate diagram of a Mixture of K Normalizing Flows. $\boldsymbol{\theta}_k$ is the parameter vector of component k.

In a similar fashion to how the Variational Auto-Encoder, proposed in [9] works, a VMoNF is fitted by jointly optimizing the parameters of the variational posterior $q(z|\boldsymbol{x};\gamma)$ and the parameters of the generative process $p(\boldsymbol{x}|z;\boldsymbol{\theta})$.

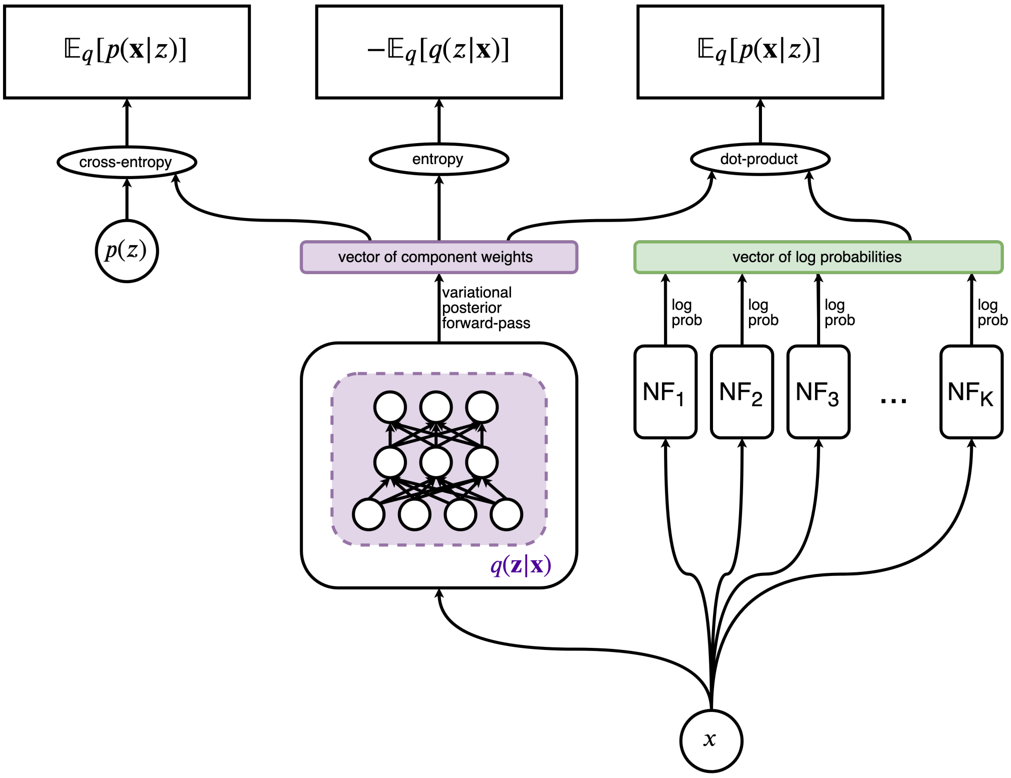

$\mathbb{E}_q[p(\mathbf{x}|z)]$
$-\mathbb{E}_q[q(z|\mathbf{x})]$
$\mathbb{E}_q[p(\mathbf{x}|z)]$
cross-entropy
entropy
dot-product
$p(z)$
vector of component weights
vector of log probabilities
variational posterior forward-pass
log prob
log prob
log prob
log prob
$q(\mathbf{z}|\mathbf{x})$
NF_1
NF_2
NF_3
...
NF_K
x

Figure 4.2: Overview of the training procedure.

Chapter 5

Conclusions

Insert your chapter material here...

5.1 Achievements

The major achievements of the present work...

5.2 Future Work

A few ideas for future work...

Bibliography

[1] Mikhail Belkin et al. “Reconciling modern machine learning and the bias-variance trade-off”. In: *arXiv* (Dec. 2018).

[2] Gregory Chaitin. “Doing Mathematics Differently”. In: *Inference - International Review of Science* Volume Two.Issue One (Feb. 2016).

[3] Nicola De Cao, Ivan Titov, and Wilker Aziz. “Block Neural Autoregressive Flow”. In: *35th Conference on Uncertainty in Artificial Intelligence (UAI19)* (2019).

[4] Nat Dilokthanakul et al. *Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders*. 2016.

[5] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using Real NVP”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017.

[6] Laurent Dinh et al. *A RAD approach to deep mixture models*. 2019.

[7] Pavel Izmailov et al. “Semi-Supervised Learning with Normalizing Flows”. In: *International Conference on Machine Learning*. *Workshop on Invertible Neural Networks and Normalizing Flows*. 2019.

[8] M. Johnson et al. “Composing graphical models with neural networks for structured representations and fast inference”. In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 2946–2954.

[9] Diederik P Kingma and Max Welling. “Auto-encoding variational Bayes”. In: *International Conference on Learning Representations (ICLR)*. 2014.

[10] Durk P Kingma and Prafulla Dhariwal. “Glow: Generative Flow with Invertible 1x1 Convolutions”. In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 10215–10224.

[11] Wu Lin, Mohammad Emtiyaz Khan, and Nicolas Hubacher. “Variational Message Passing with Structured Inference Networks”. In: *International Conference on Learning Representations*. 2018.

[12] George Papamakarios, Theo Pavlakou, and Iain Murray. “Masked Autoregressive Flow for Density Estimation”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 2338–2347.

[13] Danilo Rezende and Shakir Mohamed. “Variational Inference with Normalizing Flows”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1530–1538.

[14] Junyuan Xie, Ross Girshick, and Ali Farhadi. “Unsupervised Deep Embedding for Clustering Analysis”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 478–487.