entropy

MDPI

Review

Probabilistic Models with Deep Neural Networks

Andrés R. Masegosa [1], Rafael Cabañas [2,*], Helge Langseth [3], Thomas D. Nielsen [4], Antonio Salmerón [1]

[1] Department of Mathematics, Center for the Development and Transfer of Mathematical Research to Industry (CDTIME), University of Almería, 04120 Almería, Spain; andresmasegosa@ual.es (A.R.M.); antonio.salmeron@ual.es (A.S.)
[2] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), CH-6962 Lugano, Switzerland
[3] Department of Computer Science, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway; helge.langseth@ntnu.no
[4] Department of Computer Science, Aalborg University, DK-9220 Aalborg, Denmark; tdn@cs.aau.dk
* Correspondence: rcabanas@idsia.ch

Abstract: Recent advances in statistical inference have significantly expanded the toolbox of probabilistic modeling. Historically, probabilistic modeling has been constrained to very restricted model classes, where exact or approximate probabilistic inference is feasible. However, developments in variational inference, a general form of approximate probabilistic inference that originated in statistical physics, have enabled probabilistic modeling to overcome these limitations: (i) Approximate probabilistic inference is now possible over a broad class of probabilistic models containing a large number of parameters, and (ii) scalable inference methods based on stochastic gradient descent and distributed computing engines allow probabilistic modeling to be applied to massive data sets. One important practical consequence of these advances is the possibility to include deep neural networks within probabilistic models, thereby capturing complex non-linear stochastic relationships between the random variables. These advances, in conjunction with the release of novel probabilistic modeling toolboxes, have greatly expanded the scope of applications of probabilistic models, and allowed the models to take advantage of the recent strides made by the deep learning community. In this paper, we provide an overview of the main concepts, methods, and tools needed to use deep neural networks within a probabilistic modeling framework.

Keywords: deep probabilistic modeling; variational inference; neural networks; latent variable models; Bayesian learning

Citation: Masegosa, A.R.; Cabañas, R.; Lanseth, H.; Nielsen, T.D.; Salmerón, A. Probabilistic Models with Deep Neural Networks. *Entropy* **2021**, *23*, 117. https://doi.org/10.3390/e23010117

Received: 12 December 2020
Accepted: 12 January 2021
Published: 18 January 2021

1. Introduction

The seminal works about probabilistic graphical models (PGMs) [1,2] made probabilistic modeling an indispensable tool for dealing with uncertainty within many different fields, such as artificial intelligence [3], statistics [4], and machine learning [5,6]. PGMs have been present in the literature for over 30 years and have become a well established and highly influential body of research. At the same time, the problem of computing the posterior probability over hidden quantities given the known evidence, also known as the inference problem [1,2], has been the corner-stone, as well as the bottleneck that defines of the feasibility and applicability of probabilistic modeling (Ref. [7,8] provide in-depth introductions to inference in PGMs).

In the beginning, the first proposed inference algorithms [1,2] were able to compute this posterior in an exact way by exploiting the conditional independence relationships encoded by the graphical structure of the model. However, the set of supported probability distributions was strongly restricted, and mainly multinomial and conditional linear Gaussian distributions were used [2,9]. Researchers quickly realized that the high computational costs of these exact inference schemes made them inappropriate for dealing with the complex stochastic dependency structures that arise in many relevant problems and, consequently, approximate inference methods became a main research focus [8].

Markov Chain Monte Carlo methods were one of the first approximate methods employed for doing inference over complex PGMs [10–12]. These techniques are extremely versatile and powerful, and they are able to approximate complex posterior distributions. However, they have serious issues wrt., e.g., the convergence of the underlying Markov chain and poor mixing when approximating high dimensional distributions [10]. Computing such high dimensional posteriors started to become relevant in many domains, specifically when researchers applied a Bayesian approach for learning the parameters of their PGMs from data [5,6,13]. In this setup, the model parameters are treated as unobserved random variables, and the learning problem therefore reduces to computing the posterior probability over the parameters. For models with a large number of parameters, the approach leads to high dimensional posteriors, where the application of Monte Carlo methods becomes infeasible. These issues gave rise to the development of alternative approximate inference schemes.

Belief propagation (BP) [1,14], and the closely related expectation propagation (EP) algorithm [15], have been successfully used to overcome many of the limitations of Monte Carlo methods. These deterministic approximate inference techniques can be implemented using a message-passing scheme that takes advantage of the graph structure of the PGM and, hence, the underlying conditional independence relationships among variables. In terms of distributional assumptions, BP has mainly been used with multinomial and Gaussian distributions. Although EP allows for a more general family of distributions, it is restricted by the need to define a non-trivial quotient operation between the involved densities. While these techniques overcame some of the difficulties of Monte Carlo methods, they presented two new issues: (i) they do not guarantee convergence to an approximate and meaningful solution; and (ii) do not scale to the kind of models that appear in the context of Bayesian learning [6,13]. Again, these challenges motivated researchers to look into alternative approximate inference schemes.

Variational methods [16] were firstly explored in the context of PGMs during the late 90s [17], inspired by their successful application to inference problems encountered in statistical physics. Like BP and EP, they are deterministic approximate inference techniques. The main innovation is to cast the inference problem as a minimization-problem with a well defined loss function, namely the negative evidence lower bound (ELBO) function, which acts as an inference proxy. In general, variational methods guarantee convergence to a local maximum of this ELBO function and therefore to a meaningful solution. By transforming the inference problem into a continuous optimization problem, variational methods can take advantage of recent advances in continuous optimization theory. This was the case for the widely adopted stochastic gradient descent algorithm, which has successfully been used by the machine learning community to scale learning algorithms to big data sets [18]. This same learning algorithm was adapted to variational inference in [19], giving the opportunity to apply probabilistic modeling to problems involving massive data sets. In terms of distributional assumptions, these variational inference methods were restricted to the conjugate exponential family [20], where the gradient of the ELBO wrt. the model parameters can be computed in closed-form [21]. Ad-hoc approaches were developed for models outside this distributional family.

From the start of the field at end of the 1980's and up to around 2010, probabilistic models had mainly been focused on using distributions from the conjugate exponential family, even though this family of distributions is only able to model linear relationships between the random variables [21]. On the other hand, one of the reasons for the success of deep learning methods (Ref. [22] provides a good introduction to the field) is the ability of deep neural networks to model non-linear relationships among high-dimensional objects, as is, e.g., observed between the pixels in an image or the words in a document. Subsequent advances in variational inference [23,24] enabled the integration of deep neural networks in probabilistic models, thus also making it possible to capture such non-linear relationships among the random variables. This gave rise to a whole new family of probabilistic models, which are often denoted deep generative models [25–28]. This new

family of probabilistic models can encode objects like images, text, audio, and video probabilistically, thus bringing many of the recent advances produced by the deep learning community to the field of probabilistic modeling. The release of modern probabilistic programming languages [29–33] relying on well established deep learning engines like Tensorflow [34] and PyTorch [35] have also significantly contributed to the adoption of these powerful probabilistic modeling techniques.

In this paper, we give a coherent overview of the key concepts and methods needed for integrating deep neural networks in probabilistic models. We do not aim to provide a detailed review of all the methods published in the last years about this topic like in other recent reviews of, e.g., deep generative models [28] and variational inference methods [36]. In contrast, this paper introduces the basic concepts of variational inference methods (Section 2) and neural networks (Section 3). By building on these concepts, in Section 4, we start describing how probabilistic models with deep neural networks are represented in terms of stochastic computational graphs, which is the key data structure for implementing tractable inference methods. In Section 5, we give an overview of the main variational methods used to make inference over this power class of probabilistic models and how they are encoded in terms of a stochastic computational graph. Section 6 provides a brief discussion of current software tools for dealing with probabilistic models containing deep neural networks, all of them take the form a probabilistic programming language [37,38] and are based on the variational inference framework presented here. This paper is also accompanied by online material, where the running examples of the paper together with other basic probabilistic models containing artificial neural networks are implemented to illustrate the presented theoretical concepts and methods (https://github.com/PGM-Lab/ProbModelsDNNs).

2. Probabilistic Models within the Conjugate Exponential Family

2.1. Latent Variable Models

The conjugate exponential family of distributions [20] covers a broad and widely used range of probability distributions and density functions such as Multinomial, Normal, Gamma, Dirichlet and Beta. They have been used by the machine learning community [5,6,8] due to their convenient properties related to parameter learning and inference tasks.

In the following, we focus on probabilistic graphical models with structure as shown in Figure 1, and where the full model belongs to the conjugate exponential family. These models are also known as latent variable models (LVMs) [13,39]. LVMs are widely used as a tool for discovering patterns in data sets. The model in Figure 1 captures "local" patterns, which are specific to sample i of the data, using unobservable (or latent) random variables denoted by $\boldsymbol{Z}_i$. "Global" patterns, those that are shared among all the samples of the data set, are modeled by means of a set of latent random variables denoted by $\boldsymbol{\beta}$. The observed data sample i, $\boldsymbol{X}_i$, is modeled as random variables whose distribution is conditioned on both the local ($\boldsymbol{Z}_i$) and global ($\boldsymbol{\beta}$) latent variables. $\boldsymbol{\alpha}$, a vector of fixed hyper-parameters, is also included in the model.

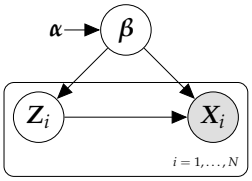


Figure 1. Structure of the probabilistic model examined in this paper, defined for a sample of size N.

While the model structure in Figure 1 at first sight can appear restrictive, it is in fact quite versatile, and many books contain entire sections devoted to LVMs [5,6,8]. For instance, LVMs include popular models like latent Dirichlet allocation (LDA) models used to uncover the hidden topics in a text corpora [40], mixture of Gaussian models to discover hidden clusters in data [5], probabilistic principal component analysis for dimensionality

reduction [41], and models to capture drift in a data stream [42,43]. They have been used for knowledge extraction from GPS data [44], genetic data [45], graph data [46], and so on.

The joint distribution of this probabilistic model factorizes into a product of local terms and a global term as

$$p(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\beta}) = p(\boldsymbol{\beta}) \prod_{i=1}^{N} p(\boldsymbol{x}_i, \boldsymbol{z}_i | \boldsymbol{\beta}),$$

where N is the number of samples. The local latent variables $\boldsymbol{Z}_i$ are assumed to be conditionally independent given the global latent variables $\boldsymbol{\beta}$.

Another standard assumption in these models is known as the assumption of complete conditional form [19]. Now, the distribution of one latent variable given the other variables in the model can be expressed in exponential family form,

$$\begin{aligned} \ln p(\boldsymbol{\beta}|\boldsymbol{x}, \boldsymbol{z}) &= h_g(\boldsymbol{\beta}) + \boldsymbol{\eta}_g(\boldsymbol{x}, \boldsymbol{z})^{\mathrm{T}} \boldsymbol{t}(\boldsymbol{\beta}) - a_g(\boldsymbol{\eta}_g(\boldsymbol{x}, \boldsymbol{z})), \\ \ln p(\boldsymbol{z}_i|\boldsymbol{x}_i, \boldsymbol{\beta}) &= h_l(\boldsymbol{z}_i) + \boldsymbol{\eta}_l(\boldsymbol{x}_i, \boldsymbol{\beta})^{\mathrm{T}} \boldsymbol{t}(\boldsymbol{z}_i) - a_l(\boldsymbol{\eta}_l(\boldsymbol{x}_i, \boldsymbol{\beta})). \end{aligned} \tag{1}$$

where the scalar functions $h_\cdot(\cdot)$ and $a_\cdot(\cdot)$ are the base measures and the log-normalizers functions, respectively; the vector functions $\boldsymbol{\eta}_\cdot(\cdot)$ and $\boldsymbol{t}_\cdot(\cdot)$ are the natural parameter and the sufficient statistics vectors, respectively. The subscripts of these functions, here g for "global" and l for "local", are used to signify that the different functions differ between variables. The subscripts will be removed when clear from context.

By conjugacy properties, the above assumptions also ensure that the conditional distribution $p(\boldsymbol{x}_i, \boldsymbol{z}_i|\boldsymbol{\beta})$ is in the exponential family,

$$\ln p(\boldsymbol{x}_i, \boldsymbol{z}_i|\boldsymbol{\beta}) = \ln h(\boldsymbol{x}_i, \boldsymbol{z}_i) + \boldsymbol{\beta}^T \boldsymbol{t}(\boldsymbol{x}_i, \boldsymbol{z}_i) - a(\boldsymbol{\beta}), \tag{2}$$

and, similarly, for the prior distribution $p(\boldsymbol{\beta})$,

$$\ln p(\boldsymbol{\beta}) \quad = \quad \ln h_{\boldsymbol{\beta}}(\boldsymbol{\beta}) + \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{t}_{\boldsymbol{\beta}}(\boldsymbol{\beta}) - a_{\boldsymbol{\beta}}(\boldsymbol{\alpha}). \tag{3}$$

Combining Equations (2) and (3), we see that the posterior $p(\boldsymbol{\beta}|\boldsymbol{x}, \boldsymbol{z})$ remains in the same distribution family as the prior $p(\boldsymbol{\beta})$ (that is, we have conjugacy) and, in consequence, the natural parameter of the global posterior $\boldsymbol{\eta}_g(\boldsymbol{x}, \boldsymbol{z})$ can be expressed as

$$\boldsymbol{\eta}_g(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\alpha} + \sum_{i=1}^{N} \boldsymbol{t}(\boldsymbol{x}_i, \boldsymbol{z}_i).$$

This representation of the complete conditional will be used later to derive the variational inference scheme over this model.

Example 1. *Principal component analysis (PCA) is a classic statistical technique for dimensionality reduction. It defines a mapping between the d-dimensional data-representation of a point $\boldsymbol{x}$ and its k-dimensional latent representation, $\boldsymbol{z}$. The latent representation is known as the scores, and the affine transformation is performed using the loading matrix $\boldsymbol{\beta}$, which has dimensions $k \times d$.*

A simplified probabilistic view of PCA [41] is given in Algorithm 1, which provides pseudo-code for the generative process of a probabilistic PCA model. This model is obviously an LVM, as the loadings represented by $\boldsymbol{\beta}$ are global latent variables and $\boldsymbol{Z}_i$ is the vector of local latent variables associated with the i-th element in the sample.

This model belongs to this conjugate exponential family with complete conditionals, because the joint of $p(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\beta})$ is multivariate normal and, by standard properties of the multivariate normal distribution, the conditional $p(\boldsymbol{\beta}|\boldsymbol{z}, \boldsymbol{x})$ and $p(\boldsymbol{z}_i|\boldsymbol{x}_i, \boldsymbol{\beta})$ are both conditional multivariate Gaussians. A multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ is a member of the exponential family with natural parameters $\boldsymbol{\eta} = \left[\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}, -1/2\boldsymbol{\Sigma}^{-1}\right]^T$ and sufficient statistics $\boldsymbol{t}(\boldsymbol{x}) = [\boldsymbol{x}, \boldsymbol{x}\boldsymbol{x}^T]^T$.

Note that while this linear relationship between the latent and the observed variables is a strong limitation of this model [47], it guarantees that the model belongs to the conjugate exponential family. Using a non-linear relationship would put PCA outside this model family and would prevent, as we will see in the next section, the use of efficient inference algorithms to calculate $p(\boldsymbol{\beta}|\boldsymbol{z},\boldsymbol{x})$ *and* $p(\boldsymbol{z}_i|\boldsymbol{x}_i,\boldsymbol{\beta})$*. Similarly, if the variance parameter* σ_x *(see Algorithm 1) depends on the latent variables* $\boldsymbol{z}_i$*, the model falls outside the conjugate exponential family.*

Figure 2 illustrates the behavior of Probabilistic PCA as a feature reduction method on two different data sets, Iris and (a reduced version of) MNIST. The data is projected from data-dimension $d = 4$ *(Iris) or* $d = 784$ *(MNIST) down into* $k = 2$ *latent dimensions. It can be seen that the method captures some of the underlying structure in the Iris-data, and even generates a representation where the three types of flower can be separated. On the other hand, the MNIST representation appears less informative. Images of the three digits "1", "2" and "3" are given to the PCA, but even though these three groups of images are quite distinct, the learned representation is not able to clearly separate the classes from one another. As we will see later in this paper, when we consider a more expressive mapping between the local latent* $\mathbf{Z}_i$ *and* $\mathbf{X}_i$ *(using artificial neural networks), the latent representations will become more informative.*

Algorithm 1 Pseudo-code of the generative model of a probabilistic PCA model.

Sample from global random variables
$\beta_{u,v} \sim \mathcal{N}(0,1)$ # Sample for $u = 1,\ldots,k$, $v = 1,\ldots,d$.
for $i = 1,\ldots,N$ **do**
  # Sample from the local latent variables
  $\boldsymbol{z}_i \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$
  # Sample from the observed variables
  $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\beta}^{\intercal}\boldsymbol{z}_i,\sigma_x^2\boldsymbol{I})$
end for

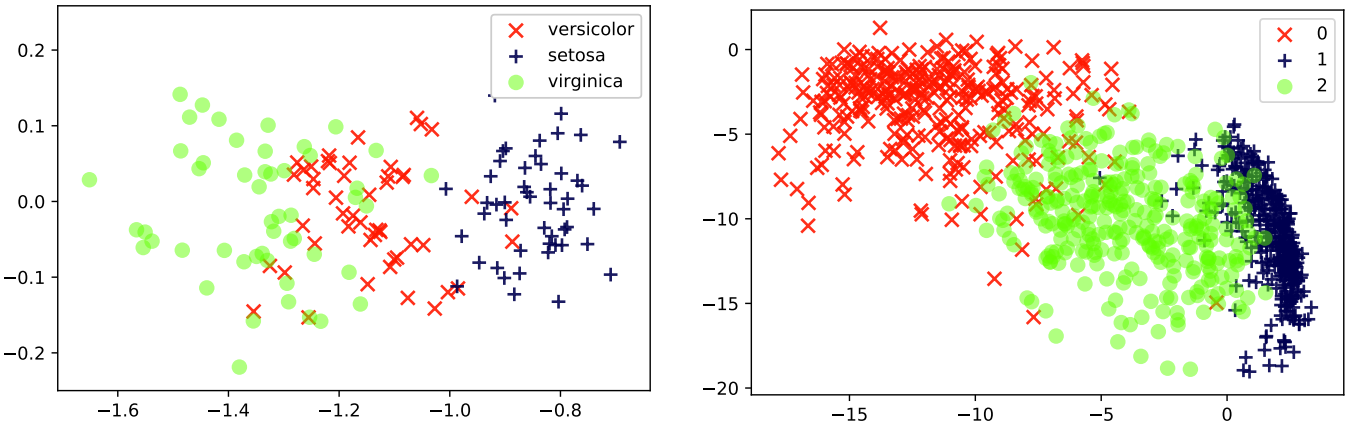


Figure 2. Two-dimensional latent representations resulting of applying a probabilistic PCA of: (**Left**) the iris dataset [48] and (**Right**) a subset of 1000 instances from the MNIST dataset [49] corresponding to the handwritten digits 1, 2 and 3.

2.2. Mean-Field Variational Inference

The problem of Bayesian inference reduces to computing the posterior over the unknown quantities (i.e., the global and local latent variables $\boldsymbol{\beta}$ and $\boldsymbol{z}$, respectively) given the observations,

$$p(\boldsymbol{\beta},\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{\beta})p(\boldsymbol{z}|\boldsymbol{\beta})p(\boldsymbol{\beta})}{\int\int p(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{\beta})p(\boldsymbol{z}|\boldsymbol{\beta})p(\boldsymbol{\beta})d\boldsymbol{z}d\boldsymbol{\beta}}.$$

Computing the above posterior is intractable for many interesting models, because it requires to solve the complicated multidimensional integral in the denominator. As commented in the introduction, variational inference (VI) methods are one of the best per-

forming options to address this problem. In this section, we revise the main ideas behind this approach.

Example 2. *Computing $p(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{x})$ for the probabilistic PCA model described in Example 1 is not feasible since the integral*

$$p(\boldsymbol{x}) = \int\int p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\beta}) p(\boldsymbol{z}|\boldsymbol{\beta}) p(\boldsymbol{\beta}) d\boldsymbol{z} d\boldsymbol{\beta}$$

is intractable. The source of the problem is that $p(\boldsymbol{x}|\boldsymbol{\beta}) = \int p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\beta}) p(\boldsymbol{z}|\boldsymbol{\beta}) d\boldsymbol{z}$ is not in the conjugate exponential family.

Variational inference is a deterministic technique that finds a tractable approximation to an intractable (posterior) distribution. We will use q to denote the approximation, and use p to signify the true distribution (like $p(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{x})$ in the example above). More specifically, let $\mathcal{Q}$ denote a set of possible approximations q. Now, VI solves the following optimization problem:

$$\min_{q \in \mathcal{Q}} KL(q||p), \tag{4}$$

where KL denotes the Kullback–Leibler divergence between two probability distributions. For the specific problem at hand, this general formulation is more precisely written as

$$\min_{q(\boldsymbol{\beta}, \boldsymbol{z}) \in \mathcal{Q}} KL(q(\boldsymbol{\beta}, \boldsymbol{z})||p(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{x}))$$

Notice that while q depends on the observations $\boldsymbol{x}$, it is customary to make this implicit in the notation, and write, e.g., $q(\boldsymbol{\beta}, \boldsymbol{z})$ instead of $q(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{x})$. In practice, one will typically posit that $\mathcal{Q}$ is a convenient distributional family indexed by some parameters, say $\boldsymbol{\theta}$, and the minimization of Equation (4) amounts to finding the parameters $\boldsymbol{\theta}^\star$ that minimize the KL divergence.

Under the mean field variational approach, the approximation family $\mathcal{Q}$ is assumed to fully factorize. Following the notation in [19], we have that

$$q(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{\lambda}, \boldsymbol{\phi}) = q(\boldsymbol{\beta}|\boldsymbol{\lambda}) \prod_{i=1}^{N} q(\boldsymbol{z}_i|\boldsymbol{\phi}_i), \tag{5}$$

where $\boldsymbol{\lambda}$ parameterizes the variational distribution of $\boldsymbol{\beta}$, while $\boldsymbol{\phi}_i$ has the same role for the variational distribution of $\boldsymbol{Z}_i$.

Furthermore, if the model is within the conjugate exponential family, each factor in the variational distribution is assumed to belong to the same family of the model's complete conditionals (see Equation (1)),

$$\begin{aligned} \ln q(\boldsymbol{\beta}|\boldsymbol{\lambda}) &= h(\boldsymbol{\beta}) + \boldsymbol{\lambda}^\intercal \boldsymbol{t}(\boldsymbol{\beta}) - a(\boldsymbol{\lambda}), \\ \ln q(\boldsymbol{z}_i|\boldsymbol{\phi}_i) &= h(\boldsymbol{z}_i) + \boldsymbol{\phi}_i^\intercal \boldsymbol{t}(\boldsymbol{z}_i) - a(\boldsymbol{\phi}_i). \end{aligned} \tag{6}$$

To solve the minimization problem in Equation (4), the variational approach exploits the transformation

$$\ln p(\boldsymbol{x}) = \mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}) + KL(q(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{\lambda}, \boldsymbol{\phi})||p(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{x})), \tag{7}$$

where $\mathcal{L}$ can be expressed as

$$\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}) = \mathbb{E}_q[\ln p(\boldsymbol{x}, \boldsymbol{Z}, \boldsymbol{\beta})] - \mathbb{E}_q[\ln q(\boldsymbol{\beta}, \boldsymbol{Z}|\boldsymbol{\lambda}, \boldsymbol{\phi})]. \tag{8}$$

$\mathcal{L}$ is of interest in its own right. Notice in particular that $\mathcal{L}$ in Equation (7) is a *lower bound* of $\ln p(\boldsymbol{x})$ since the KL-divergence is non-negative. For this reason, $\mathcal{L}$ is usually referred to as

the ELBO (evidence lower bound). Furthermore, as $\ln p(\boldsymbol{x})$ is constant in the optimization wrt. q, minimizing the KL divergence in Equation (4) is equivalent to maximizing the lower bound $\mathcal{L}$. Variational methods maximize $\mathcal{L}$ using gradient based techniques.

The key advantage of having a conjugate exponential model is that the gradients of $\mathcal{L}$ wrt. its parameters can always be computed in closed form [21]. This is important, as it leads to a natural scheme in which the parameters are updated iteratively: For a parameter θ_j, simply choose the value $\theta_j^\star$ so that $\nabla_{\theta_j}\mathcal{L}(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}:\theta_j=\theta_j^\star} = 0$. In practice, it is beneficial to use the natural gradient, which is the standard gradient pre-multiplied by the inverse of the Fisher information matrix, to account for the Riemannian geometry of the parameter space [50].

The gradients with respect to the variational parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\phi}$ can be computed as follows,

$$\begin{aligned}\nabla^{nat}_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{\lambda},\boldsymbol{\phi}) &= \boldsymbol{\alpha} + \sum_{i=1}^{N}\mathbb{E}_{\boldsymbol{Z}_i}[\boldsymbol{t}(\boldsymbol{x}_i,\boldsymbol{Z}_i)] - \boldsymbol{\lambda},\\ \nabla^{nat}_{\boldsymbol{\phi}_i}\mathcal{L}(\boldsymbol{\lambda},\boldsymbol{\phi}) &= \mathbb{E}_{\boldsymbol{\beta}}[\boldsymbol{\eta}_l(\boldsymbol{x}_i,\boldsymbol{\beta})] - \boldsymbol{\phi}_i,\end{aligned} \tag{9}$$

where ∇^{nat} denotes natural gradients and $\mathbb{E}_{\boldsymbol{Z}_i}[\cdot]$ and $\mathbb{E}_{\boldsymbol{\beta}}[\cdot]$ denote expectations with respect to $q(\boldsymbol{z}_i|\boldsymbol{\phi}_i)$ and $q(\boldsymbol{\beta}|\boldsymbol{\lambda})$, respectively.

From the above gradients, we can derive a coordinate ascent algorithm to optimize the ELBO function with the following coordinate ascent rules,

$$\begin{aligned}\boldsymbol{\lambda}^\star &= \arg\max_{\lambda}\mathcal{L}(\boldsymbol{\lambda},\boldsymbol{\phi}) = \boldsymbol{\alpha} + \sum_{i=1}^{N}\mathbb{E}_{\boldsymbol{Z}_i}[\boldsymbol{t}(\boldsymbol{x}_i,\boldsymbol{Z}_i)],\\ \boldsymbol{\phi}_i^\star &= \arg\max_{\boldsymbol{\phi}_i}\mathcal{L}(\boldsymbol{\lambda},\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{\beta}}[\boldsymbol{\eta}_l(\boldsymbol{x}_i,\boldsymbol{\beta})].\end{aligned} \tag{10}$$

By iteratively running the above updating equations, we are guaranteed to (i) monotonically increase the ELBO function at every time step and (ii) to converge to a stationary point of the ELBO function or, equivalently, the function minimizing Equation (4).

Example 3. *For the PCA model in Example 1, the variational distributions are*

$$\begin{aligned}q(\boldsymbol{\beta}|\boldsymbol{\mu}_\beta,\boldsymbol{\Sigma}_\beta) &= \mathcal{N}(\boldsymbol{\beta}|\boldsymbol{\mu}_\beta,\boldsymbol{\Sigma}_\beta),\\ q(\boldsymbol{z}_i|\boldsymbol{\mu}_{z_i},\boldsymbol{\Sigma}_{z_i}) &= \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{\mu}_{z_i},\boldsymbol{\Sigma}_{z_i}).\end{aligned}$$

Given the above variational family, the coordinate updating equations derived from Equation (10) can be written, after some algebraic manipulations, as [5]

$$\begin{aligned}\boldsymbol{\Sigma}_\beta &= \left(\sum_{i=1}^{N}\mathbb{E}[\boldsymbol{Z}_i\boldsymbol{Z}_i^T] + \sigma_x^2\boldsymbol{A}\right)^{-1},\\ \boldsymbol{\mu}_\beta &= \left[\sum_{i=1}^{N}\boldsymbol{x}_i\mathbb{E}[\boldsymbol{Z}_i]\right]^T\boldsymbol{\Sigma}_\beta,\\ \boldsymbol{\Sigma}_{z_i} &= \left(\boldsymbol{I} + \boldsymbol{\mu}_\beta^T\boldsymbol{\mu}_\beta/\sigma_x^2\right)^{-1},\\ \boldsymbol{\mu}_{z_i} &= \boldsymbol{\Sigma}_{z_i}\boldsymbol{\mu}_\beta^T\boldsymbol{x}_i/\sigma_x^2,\end{aligned}$$

where $\boldsymbol{A}$ is a diagonal matrix with element at index (i,i) given by $d/\boldsymbol{\mu}_{\beta,i}^T\boldsymbol{\mu}_{\beta,i}$. Again, we have a set of closed-form equations that guarantees convergence to the solution of the inference problem. We should note that this is possible due to the strong assumptions, imposed both on the probabilistic model p and on the family of variational approximations $\mathcal{Q}$.

2.3. Scalable Variational Inference

Performing VI on large data sets (measured by the number of samples, N) raises many challenges. Firstly, the model itself may not fit in memory, and, secondly, the cost of computing the gradient of the ELBO with respect to $\boldsymbol{\lambda}$ linearly depends on the size of the data set (see Equation (9)), which can be prohibitively expensive when N is very large. Stochastic variational inference (SVI) [19] is a popular method for scaling VI to massive data sets, and relies on stochastic optimization techniques [18,51].

We start by re-parameterizing the ELBO so that $\mathcal{L}$ is expressed only in terms of the global parameters $\boldsymbol{\lambda}$. This is done by defining

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}^{\star}(\boldsymbol{\lambda})), \tag{11}$$

where $\boldsymbol{\phi}^{\star}(\boldsymbol{\lambda})$ is defined as in Equation (10), i.e., it returns a local optimum of the local variational parameters $\boldsymbol{\phi}$ for a given $\boldsymbol{\lambda}$. Now $\mathcal{L}(\boldsymbol{\lambda})$ has the following form:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_q[\ln p(\boldsymbol{\beta})] - \mathbb{E}_q[\ln q(\boldsymbol{\beta}|\boldsymbol{\lambda})] \\ + \sum_{i=1}^{N} \max_{\boldsymbol{\phi}_i} \{\mathbb{E}_q[\ln p(\boldsymbol{x}_i, \boldsymbol{Z}_i|\boldsymbol{\beta})] - \mathbb{E}_q[\ln q(\boldsymbol{Z}_i|\boldsymbol{\phi}_i)]\}\end{aligned} \tag{12}$$

As shown in [19], we can compute the gradient of $\mathcal{L}(\boldsymbol{\lambda})$ by first finding $\boldsymbol{\phi}^{\star}(\boldsymbol{\lambda})$, and then compute the gradient w.r.t. $\boldsymbol{\lambda}$ while keeping $\boldsymbol{\phi}^{\star}(\boldsymbol{\lambda})$ fixed (because $\nabla_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}^{\star}(\boldsymbol{\lambda}))$). By exploiting properties of the conjugate exponential family, Ref. [19] computes the the natural gradient with respect to $\boldsymbol{\lambda}$ in closed-form as

$$\nabla_{\boldsymbol{\lambda}}^{nat}\mathcal{L}(\boldsymbol{\lambda}) = \boldsymbol{\alpha} + \sum_{i=1}^{N} \mathbb{E}_{q(\boldsymbol{z}_i|\boldsymbol{\phi}_i^{\star})}[\boldsymbol{t}(\boldsymbol{x}_i, \boldsymbol{Z}_i)] - \boldsymbol{\lambda}.$$

The key idea behind SVI is to compute unbiased albeit noisy estimates of $\nabla_{\boldsymbol{\lambda}}^{nat}\mathcal{L}$, denoted $\hat{\nabla}_{\boldsymbol{\lambda}}^{nat}\mathcal{L}$, by randomly selecting a mini-batch of M data samples, and then define

$$\hat{\nabla}_{\boldsymbol{\lambda}}^{nat}\mathcal{L}(\boldsymbol{\lambda}) = \boldsymbol{\alpha} + \frac{N}{M}\sum_{m=1}^{M} \mathbb{E}_{q(\boldsymbol{z}_i|\boldsymbol{\phi}_i^{\star})}[t(\boldsymbol{x}_{i_m}, \boldsymbol{Z}_{i_m})] - \boldsymbol{\lambda},$$

where i_m is the variable index form the subsampled mini-batch. It is immediate that $\mathbb{E}[\hat{\nabla}_{\boldsymbol{\lambda}}^{nat}\mathcal{L}] = \nabla_{\boldsymbol{\lambda}}^{nat}\mathcal{L}$, hence the estimator is unbiased. Utilizing stochastic optimization theory [51], the ELBO can be optimized by following noisy estimates of the gradient,

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \rho_t \hat{\nabla}_{\boldsymbol{\lambda}}^{nat}\mathcal{L}(\boldsymbol{\lambda}_t), \tag{13}$$

where the learning rate ρ_t should satisfy the Robbins–Monro conditions (A sequence $\{\rho_t\}_{t=1}^{\infty}$ satisfies the Robbins–Monro conditions if $\sum_{t=1}^{\infty}\rho_t = \infty$ and $\sum_{t=1}^{\infty}\rho_t^2 < \infty$).

To choose the size of the mini-batch M, two conflicting issues should be considered: smaller values of M (i.e., $M \ll N$) lead to a reduction in the computational complexity of computing the gradient, while larger values of M (i.e., $M \gg 1$) reduce the variance of the estimator. The optimal value for M is problem dependent [52].

Alternative ways to scale up variational inference in conjugate exponential models involve the use of distributed computing clusters. For example, it can be assumed that the data set is stored in a distributed way among different machines [53]. Then, the problem of computing the ELBO's gradient given in Equation (9) is scaled up by distributing the computation of the gradient $\nabla_{\boldsymbol{\phi}_i}^{nat}\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi})$, so that each machine computes this term for those samples that are locally stored. Finally, all the terms are sent to a master node, which aggregates them and computes the gradient $\nabla_{\boldsymbol{\lambda}}^{nat}\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi})$ (see Equation (9)).

Example 4. *For Example 3, we detailed the variational updating equations for the probabilistic PCA model introduced in Example 1. In order to update $\boldsymbol{\mu}_{\boldsymbol{\beta}}^{\star}$, we need to iterate over the whole data set. Furthermore, the number of local variational parameters $\boldsymbol{\mu}_{z_i}^{\star}$ and $\Sigma_{z_i}^{\star}$ is equal to the number of data points. Therefore, if N is very large, the computation of these variational updating equations becomes infeasible.*

Following the methodology presented in this section, we can obtain a new set of variational updating equations,

$$\begin{aligned}\boldsymbol{\Sigma}_{\beta,t+1} &= \left[(1-\rho_t)\boldsymbol{\Sigma}_{\beta,t}^{-1} + \rho_t\left(\frac{N}{M}\sum_{m=1}^{M}\mathbb{E}_{t,i_m}\left[\boldsymbol{Z}_{i_m}\boldsymbol{Z}_{i_m}^{T}\right] + \sigma_x^2\boldsymbol{A}\right)\right]^{-1},\\ \boldsymbol{\mu}_{\beta,t+1} &= (1-\rho_t)\boldsymbol{\mu}_{\beta,t+1} + \rho_t\left(\frac{N}{M}\sum_{m=1}^{M}\boldsymbol{x}_{i_m}\mathbb{E}_{t,i_m}[\boldsymbol{Z}_{i_m}]\right)^{T}\boldsymbol{\Sigma}_{\beta,t+1},\end{aligned}$$

where $\{i_1,\ldots,i_M\}$ are the indexes of the mini-batch, and $\mathbb{E}_{t,i_m}[\cdot]$ denotes expectations when $\boldsymbol{Z}_{i_m}$ follows a multivariate normal distribution with parameters

$$\begin{aligned}\boldsymbol{\Sigma}_{t,z_{i_m}} &= \left(\boldsymbol{I} + \sigma_x^{-2}\boldsymbol{\mu}_{\beta,t}^{T}\boldsymbol{\mu}_{\beta,t}\right)^{-1},\\ \boldsymbol{\mu}_{t,z_{i_m}} &= \boldsymbol{\Sigma}_{t,z_{i_m}}\boldsymbol{\mu}_{\beta,t}^{T}\boldsymbol{x}_{i_m}/\sigma_x^2;\end{aligned}$$

confer also Example 3. Using this set-up, we do not need to go thorough the full data set to get an update on the global variational parameters.

2.4. Variational Message Passing

So far, we have treated the set of variables $\boldsymbol{x}$, $\boldsymbol{z}$ and $\boldsymbol{\beta}$ as undividable blocks of variables without internal structure. However, as we are dealing with flexible probabilistic graphical models, these sets of variables can often encode conditional independencies that can be further exploited when using VI. variational message passing (VMP) [21] is a VI scheme which readily exploits such conditional independencies when performing approximate inference. Now, $\boldsymbol{Z}_i$ and $\boldsymbol{X}_i$, the set of latent and observable variables associated to the i-th data sample, are separated into individual variables $\boldsymbol{Z}_i = \{Z_{i,1},\ldots,Z_{i,K}\}$, and similarly for $\boldsymbol{X}_i$. Additionally, $\boldsymbol{\beta}$ is regarded as a set of J separate random variables $\boldsymbol{\beta} = \{\beta_1,\ldots,\beta_J\}$. Now, under the mean field assumption, the variational distribution is expressed as

$$q(\boldsymbol{\beta},\boldsymbol{z}|\boldsymbol{\lambda}_i,\boldsymbol{\phi}) = \prod_{j=1}^{J} q(\beta_j|\boldsymbol{\lambda}_j)\prod_{i=1}^{N}\prod_{k=1}^{K} q(z_{i,k}|\boldsymbol{\phi}_{i,k}).$$

Using the VMP scheme, the gradients wrt. the variational parameters can be computed using a message-passing algorithm which exploits the conditional independencies between the variables in $\boldsymbol{X}_i$, $\boldsymbol{Z}_i$ and $\boldsymbol{\beta}$. The flow of messages is similar to the one employed by loopy belief propagation [1]. The messages are expected sufficient statistics of the variables involved, and since the model is in the conjugate exponential family, both the messages and the update rules can be expressed analytically, leading to parameter updates akin to Equation (10); cf. [21] for details.

3. Deep Neural Networks and Computational Graphs

3.1. Deep Neural Networks

An artificial neural network (ANN) [54] can be seen as a deterministic non-linear function $f(\cdot:\boldsymbol{W})$ parametrized by a matrix $\boldsymbol{W}$. An ANN with L hidden layers defines a

mapping from a given input $\boldsymbol{x}$ to a given output $\boldsymbol{y}$. This mapping is built by the recursive application of a sequence of (non-)linear transformations,

$$\begin{aligned} \boldsymbol{h}_0 &= \boldsymbol{r}_0(\boldsymbol{W}_0^T \boldsymbol{x}), \\ &\cdots \\ \boldsymbol{h}_l &= \boldsymbol{r}_l(\boldsymbol{W}_l^T \boldsymbol{h}_{l-1}), \\ &\cdots \\ \boldsymbol{y} &= \boldsymbol{r}_L(\boldsymbol{W}_L^T \boldsymbol{h}_{L-1}), \end{aligned} \tag{14}$$

where $\boldsymbol{r}_l(\cdot)$ defines the (non-linear) activation function at the l-th layer; standard activation functions include the *soft-max* function and the *relu* function [55,56]. $\boldsymbol{W}_l$ are the parameters defining the linear transformation at the l-th layer, where the dimensionality of the target layer is defined by the size of $\boldsymbol{W}_l$. Deep neural networks (DNNs) is a renaming of classic ANNs, with the key difference that DNNs usually have a higher number of hidden layers compared to what classical ANNs used to have.

Learning a DNN from a given data set of input-output pairs $(\boldsymbol{x}, \boldsymbol{y})$ reduces to solving the optimization problem

$$\boldsymbol{W}^\star = \arg\min_{\boldsymbol{W}} \sum_{i=1}^{N} \ell(y_i, f(\boldsymbol{x}_i; \boldsymbol{W})), \tag{15}$$

where $\ell(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i)$ is a loss function which defines the quality of the model, i.e, how well the output $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \boldsymbol{W})$ returned by the DNN model matches the real output $\boldsymbol{y}_i$. This continuous optimization problem is usually solved by applying a variant of the stochastic gradient descent method, which involves the computation of the gradient of the loss function with respect to the parameters of the ANN, $\nabla_{\boldsymbol{W}} \ell(y_i, f(\boldsymbol{x}_i; \boldsymbol{W}))$. The algorithm for computing this gradient in an ANN is known as the *back-propagation* algorithm, which is based on the recursive application of the chain-rule of derivatives and typically implemented based in the computational graph of the ANN. A detailed and modern introduction to this field is provided in [22].

3.2. Computational Graphs

Computational graphs [34,35,57] have been extremely useful when developing algorithms and software packages for neural networks and other models in machine learning. The main idea of a computational graph is to express a (deterministic) function, as is the case of a neural network, as an acyclic directed graph defining a sequence of computational operations. A computational graph is composed of input and output nodes as well as operation nodes. The data and the parameters of the model serve as input nodes, whereas the operation nodes (represented as squares in the subsequent diagrams) correspond to the primitive operations of the network and also define the output of the network. The directed edges in the graph specify the inputs of each node. Input nodes are usually defined over tensors (n-dimensional arrays) and operations are thus similarly defined over tensors, thereby also enabling the computational graph to, e.g., process batches of data. Figure 3 shows a simple example of a computational graph.

With computational graphs, simple/primitive functions can be combined to form complex operations, and the vast majority of current neural networks can be defined using computational graphs. However, the key strength of computational graphs is that they allow for automatic differentiation [58]. As shown in the previous section (see Equation (15)), most neural network learning algorithms translate to a continuous optimization problem of a differentiable loss function often solved by a gradient descent algorithm. Automatic differentiation is a technique for automatically computing all the partial derivatives of the function encoded by a computational graph: once the graph has been defined using underlying primitive operations, derivatives are automatically calculated based on the "local" derivatives of these operations and the recursive application of the chain rule of

derivatives, incurring only a small computational overhead. Before the use of computational graphs in deep learning, these derivatives had to be computed manually, giving rise to a slow and error-prone development process.

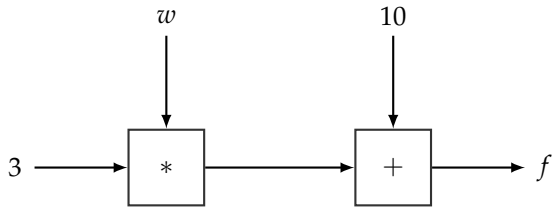


Figure 3. Example of a simple Computational Graph. Squared nodes denote operations, and the rest are input nodes. This computational graph encodes the operation $f = 3 \cdot w + 10$, where w is a variable wrt. which we can differentiate.

Example 5. *Figure 4 provides an example of a computational graph encoding a neural network with $\boldsymbol{x}$ as input, $\hat{\boldsymbol{y}}$ as output, and two hidden layers. This computational graph also encodes the loss function $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$. As computational graphs can be defined over tensors, the above computational graph can encode the forward (and backward) pass of the neural network for a whole data batch $\boldsymbol{x}$, and thereby also provide the loss (and the gradient) for this set of data samples. Algorithm 2 shows the pseudo-code description for defining and learning this neural network using standard gradient descent.*

Algorithm 2 Pseudo-code of the definition and learning of a simple neural network.

input $\boldsymbol{x}, \boldsymbol{y}$ the labels.
  # Define the computational graph encoding the ANN and the loss function
  $\boldsymbol{W}_0, \boldsymbol{W}_1, \boldsymbol{W}_2 = Parameters()$
  $\boldsymbol{h}_0 = relu(\boldsymbol{W}_0^T \boldsymbol{x})$
  $\boldsymbol{h}_1 = relu(\boldsymbol{W}_1^T \boldsymbol{h}_0)$
  $\hat{\boldsymbol{y}} = relu(\boldsymbol{W}_2^T \boldsymbol{h}_1)$
  $\ell = ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$
  # Follow the gradients until convergence.
  $\boldsymbol{W} = (\boldsymbol{W}_0, \boldsymbol{W}_1, \boldsymbol{W}_2)$
  repeat
    $\boldsymbol{W} = \boldsymbol{W} - \rho \nabla_{\boldsymbol{W}} \ell$
  until convergence

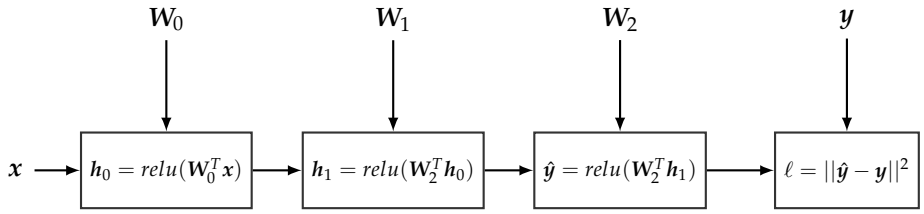


Figure 4. Example of a simple computational graph encoding a neural network with two hidden layers and the squared loss function. Note that each operation node encapsulates a part of the CG encoding the associated operations, we do not expand the whole CG for the sake of simplicity.

4. Probabilistic Models with Deep Neural Networks

4.1. Deep Latent Variable Models

LVMs have usually been restricted to the conjugate exponential family because, in this case, inference is feasible (and scalable) as we showed in Section 2. But recent advances in VI (which will be outline in Section 5) have enabled LVMs to be extended with DNNs. Variational auto-encoders (VAE) [23,59] are probably the most influential models combining LVMs and DNNs. VAEs extend the classical technique of PCA for data representation in lower-dimensional spaces. More precisely, the probabilistic version of the PCA model [41]

is extended in [23], where the relationship between the low-dimensional representation and the observed data is governed by a DNN, i.e., a highly non-linear function, as opposed to the standard linear transformation in the basic version of the PCA model. These models are able to capture more compact low-dimensional representations, especially in cases where data is high-dimensional but "lives" in a low-dimensional manifold [60]. This is, e.g., the case for image data [23,61–64], text data [65], audio data [66], chemical molecules [67], to name some representative applications of this technique. We note that, in this section and in the following ones, we will use VAEs as a running example illustrating how DNNs can be used in probabilistic modeling.

VAEs have also given rise to a plethora of extensions of classic LVMs to their *deep* counterpart. For instance, different examples of this approach are given in [68], along with extensions of Gaussian mixture models, latent linear dynamical systems and latent switching linear dynamical systems with the non-linear relationships modeled by DNNs. Hidden semi-Markov models are extended with recurrent neural networks in [69]. Extensions of popular LDA models [40] for uncovering topics in text data can be found in [70,71]. Many other works are following the same trend [72–75].

Example 6. *VAEs are widely adopted LVMs containing DNNs [23]. Algorithm 3 provides a simplified pseudo-code description of the generative part of a VAE model. It can also be seen as a non-linear probabilistic PCA model, where the non-linearity is included in the form of an artificial neural network.*

This model is quite similar to the PCA model presented in Example 1. The main difference comes from the conditional distribution of $\boldsymbol{X}$. In the PCA model, the mean of the normal distribution of $\boldsymbol{X}$ linearly depends on $\boldsymbol{Z}$ through $\boldsymbol{\beta}$. In the VAE model, the mean depends on $\boldsymbol{Z}$ through a DNN parametrized by $\boldsymbol{\beta}$. This DNN is also known as the decoder network of the VAE [23].

Note that some formulations of this model also include another DNN component, which connects $\boldsymbol{Z}$ with the variance σ^2 of the normal distribution of $\boldsymbol{X}$; for the sake of simplicity, we have not included this extension in the example.

Figure 5 experimentally illustrates the advantage of using a non-linear PCA model over the classic PCA model. As can be seen, the non-linear version separates more clearly the three digits than the linear model did. We shall return to this example in Section 5.2, where we will introduce the so-called encoder network used for inference.

Algorithm 3 Pseudo-code of the generative model of a variational auto-encoder (or non-linear probabilistic PCA).

```
# Define the global parameters
α_0, β_0, α_1, β_1 ~ N(0, I)
for i = 1, ..., N do
  # Define the local latent variables
  Z_i ~ N(0, I)
  # Define the ANN with a single hidden layer h_i
  h_i = relu(β_0^T z_i + α_0)
  μ_i = β_1^T h_i + α_1
  # Define the observed variables
  X_i ~ N(μ_i, σ^2 I)
end for
```

LVMs with DNNs can also be found in the literature under the name of deep generative models [25–28]. They generate data samples using probabilistic constructs that include DNNs. This new capacity has also resulted in substantial impact within the deep learning community because it has opened up for the possibility of dealing with unsupervised learning problems, e.g., in the form of generative adversarial nets [27]. This should be seen in contrast to the classic deep learning methods, which are mainly focused on supervised learning settings. In any case, this active area of research is out of the scope of this paper

and contains many alternative models, which do not fall within the category of the models explored in this paper (Ref. [76] provides a recent survey of this field).

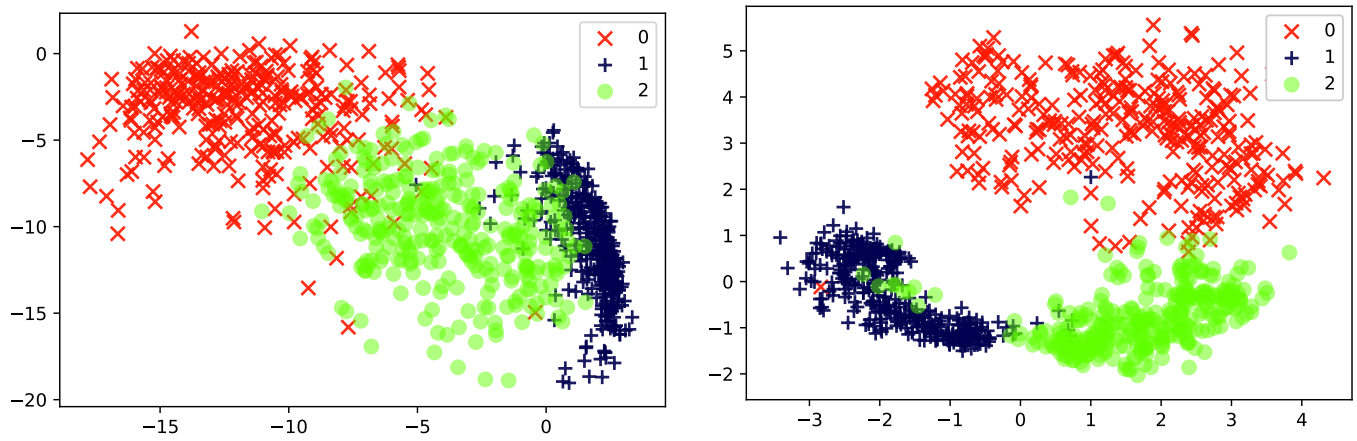
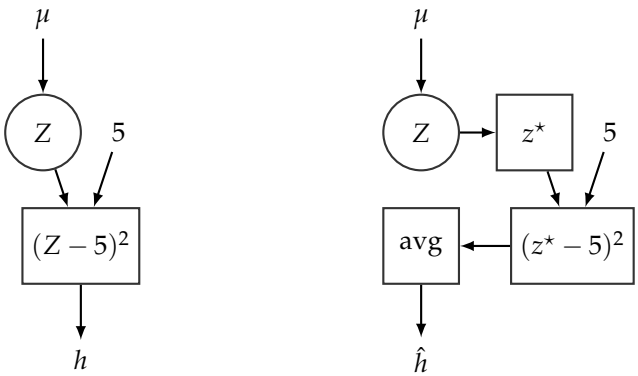


Figure 5. Two-dimensional latent representations of the the MNIST dataset resulting of applying: (**Left**) a standard probabilistic PCA (reproduced from Figure 2 to ease comparison), and (**Right**) a non-linear probabilistic PCA with a ANN containing a hidden layer of size 100 with a *relu* activation function.

4.2. Stochastic Computational Graphs

The key data structure for representing probabilistic models with deep neural networks is the so-called stochastic computational graph (SCG) [77]. SCGs extend standard computational graphs (defined Section 3.2 with stochastic nodes (represented as circles in the subsequent diagrams). The probability distributions associated with stochastic nodes are defined conditionally on their parents and enable the specification of complex functions involving expectations over random variables. Figure 6 (**Left**) shows an example of a simple SCG involving an expectation over a random variable $\boldsymbol{Z}$. Modern PPLs support a wide and diverse range of probability distributions for defining SCGs [78]. These probability distributions are defined over tensor objects to seamlessly accommodate the underlying CGs, which define operations over tensor objects too.



$$h = \mathbb{E}_{Z\sim\mathcal{N}(\mu,1)}[(Z-5)^2] \qquad \hat{h} = \frac{1}{k}\sum_{i=1}^{k}(z_i^\star - 5)^2, \quad Z_i^\star \sim \mathcal{N}(\mu,1)$$

Figure 6. (**Left**) A stochastic computational graph encoding the function $h = \mathbb{E}_Z[(Z-5)^2]$, where $Z \sim N(\mu,1)$. (**Right**) Computational graph processing k samples from Z and producing $\hat{h}$, an estimate of $\mathbb{E}_Z[(Z-5)^2]$.

We note that SCGs are not directly implemented within PPLs, because computing the exact expected value of a complex function is typically infeasible. However, they are indirectly included through the use of a standard computational graph engine: Each stochastic node, $\boldsymbol{Z}$, is associated with a tensor, $z^\star$, which represents a (set of) sample(s) from the

distribution associated with z, and the generated samples can thus be fed to the underlying computational graph through the tensor $z^\star$. Hence, SCGs can be *simulated* by sampling from the stochastic nodes and processing these samples by a standard CG. Figure 6 illustrates how SCG can be simulated using standard CGs. Note that CGs are designed to operate efficiently with tensors (current toolboxes like TensorFlow exploit high-performance computing hardware such as GPUs or TPUs [34]), and it is therefore much more efficient to run the CG once over a collection of samples, rather than running the CG multiple times over a single sample.

In this way, SCGs can be used to define and support inference and learning of general probabilistic models, including the ones referenced in Section 4.1. More generally, all the concepts outlined in this paper apply to any probabilistic model that can be defined by means of an SCG or which can be compiled into an equivalent SCG representation. For instance, the following model specification (illustrated by the top part in Figure 7) relating $\boldsymbol{Z}$ with the natural parameters $\boldsymbol{\eta}_x$ of $\boldsymbol{x}$ can be equivalently represented by the SCG illustrated in the lower part of Figure 7.

$$\begin{aligned}
\ln p(\boldsymbol{\beta}) &= \ln h(\boldsymbol{\beta}) + \boldsymbol{\alpha}^T t(\boldsymbol{\beta}) - a_g(\boldsymbol{\alpha}),\\
\ln p(\boldsymbol{z}_i|\boldsymbol{\beta}) &= \ln h(\boldsymbol{z}_i) + \eta_z(\boldsymbol{\beta})^T t(\boldsymbol{z}_i) - a_z(\eta_z(\boldsymbol{\beta})),\\
\boldsymbol{h}_0 &= \boldsymbol{r}_0(\boldsymbol{z}_i^T \boldsymbol{\beta}_0),\\
&\cdots\\
\boldsymbol{h}_l &= \boldsymbol{r}_l(\boldsymbol{h}_{l-1}^T \boldsymbol{\beta}_{l-1}),\\
&\cdots\\
\boldsymbol{h}_L &= \boldsymbol{r}_L(\boldsymbol{h}_L^T \boldsymbol{\beta}_L),\\
\ln p(\boldsymbol{x}_i|\boldsymbol{z}_i, \boldsymbol{\beta}) &= \ln h(\boldsymbol{x}_i) + \eta_x(\boldsymbol{h}_L)^T t(\boldsymbol{x}_i) - a_x(\eta_x(\boldsymbol{h}_L)).
\end{aligned} \tag{16}$$

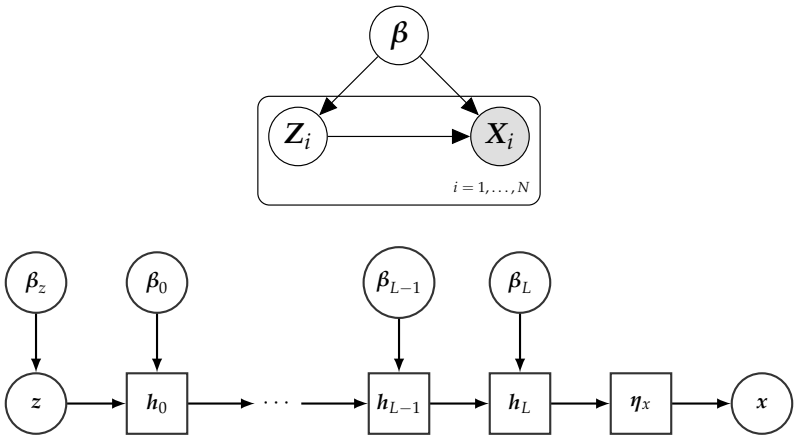


Figure 7. The top part depicts a probabilistic graphical model using plate notation [8]. The lower part depicts an abstract representation of a stochastic computational graph encoding the model, where the relation between $\boldsymbol{z}$ and $\boldsymbol{x}$ is defined by a DNN with $L+1$ layers. See Section 4 for details.

From this example, we again see the main difference with respect to standard LVMs (see Section 2.1) is the conditional distribution of the observations $\boldsymbol{x}_i$ given the local hidden variables $\boldsymbol{Z}_i$ and the global parameters $\boldsymbol{\beta}$, which is here governed by a DNN parameterized by $\boldsymbol{\beta}$.

5. Variational Inference with Deep Neural Networks

Similarly to standard probabilistic models, performing variational inference in deep latent variable models (as described in the previous section) also reduces to maximizing the ELBO function $\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi})$ given in Section 2.2 (Equation (8)); recall that this is equivalent

to minimizing the KL divergence between the variational posterior $q(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{\lambda}, \boldsymbol{\phi})$ and the target distribution $p(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{x})$. However, as was also noted in the previous section, when the probabilistic model contains complex constructs like DNNs, it falls outside the conjugate exponential family and the traditional VI methods, tailored to this specific family form, can therefore not be applied.

In terms of the variational distribution, we will still assume the same factorization scheme defined in Equation (5) for the deep latent variable models considered in this section. However, as we will see below, we need not adopt the conjugate models' strong restrictions on the variational approximation family (see Equation (6)). Instead, the only (and much weaker) restriction that we will impose is that (*i*) the log probability of the variational distribution, $\ln q(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{\lambda}, \boldsymbol{\phi})$, can be represented by a computational graph (and, as a consequence, that it is differentiable wrt. $\boldsymbol{\lambda}$ and $\boldsymbol{\phi}$) and (*ii*) that we can sample from the variational distribution $q(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{\lambda}, \boldsymbol{\phi})$. Depending on the specific method being applied, additional requirements may be introduced.

In the rest of the section, we will give an overview of the two main techniques employed in modern variational methods to perform inference in probabilistic models with deep neural networks. In Section 5.1, we provide an overview to black-box variational inference [24,79,80] whose main purpose is the computation of the gradient of the ELBO. Within this section, we described the two main approaches for computing this gradient using Monte Carlo methods. Section 5.2 introduces amortized variational inference [81,82], a widely used technique for dealing with probabilistic models containing local latent variables [23]. We end this section by discussing the pros and cons of each of the presented methods.

5.1. Black Box Variational Inference

For the sake of presentation, we reparameterize the ELBO function with $\boldsymbol{r} = (\boldsymbol{\beta}, \boldsymbol{z})$ and $\boldsymbol{\nu} = (\boldsymbol{\lambda}, \boldsymbol{\phi})$ and define $g(\boldsymbol{r}, \boldsymbol{\nu}) = \ln p(\boldsymbol{x}, \boldsymbol{r}) - \ln q(\boldsymbol{r}|\boldsymbol{\nu})$. With this notation the ELBO function $\mathcal{L}$ of Equation (8) can then be expressed as

$$\mathcal{L}(\boldsymbol{\nu}) = \mathbb{E}_{\boldsymbol{R}}[g(\boldsymbol{r}, \boldsymbol{\nu})] = \int q(\boldsymbol{r}|\boldsymbol{\nu}) g(\boldsymbol{r}, \boldsymbol{\nu}) d\boldsymbol{r}, \tag{17}$$

from which we see that the ELBO function can easily be represented by an SCG as shown in Figure 8. If the SCG in Figure 8 did not include stochastic nodes (thus corresponding to a standard CG), we could, in principle, perform variational inference (maximizing $\mathcal{L}(\boldsymbol{\nu})$ wrt. $\boldsymbol{\nu}$) by simply relying on automatic differentiation and a variation of gradient ascent. However, optimizing over SCGs is much more challenging because automatic differentiation does not readily apply. The problem is that the variational parameters $\boldsymbol{\nu}$ (wrt. which we should differentiate) also affects the expectation inherent in the ELBO function, see Equation (17):

$$\nabla_{\boldsymbol{\nu}} \mathcal{L} = \nabla_{\boldsymbol{\nu}} \mathbb{E}_{\boldsymbol{R}}[g(\boldsymbol{r}, \boldsymbol{\nu})]. \tag{18}$$

In the case of conjugate models, we can take advantage of their properties and derive closed-form solutions for this problem, as detailed in Section 2.2. In general, though, there are no closed-form solutions for computing gradients in non-conjugate models; a simple concrete example is the Bayesian logistic regression model [6] (p. 756).

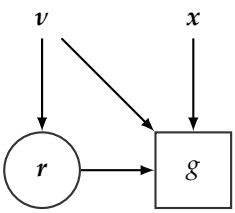


Figure 8. SCG representing the ELBO function $\mathcal{L}(\nu)$. $\boldsymbol{r}$ is distributed according to the variational distribution, $\boldsymbol{r} \sim q(\boldsymbol{r}|\boldsymbol{\nu})$.

In this section, we provide two generic solutions for computing the gradient of the ELBO function for probabilistic models including DNNs. Both methods directly rely on the automatic differentiation engines available for standard computational graphs. In this way, the methods can be seen as extending the automatic differentiation methods of standard computational graphs to SCGs, giving rise to a powerful approach to VI for generic probabilistic models. The main idea underlying both approaches is to compute the gradient of the expectation given in Equation (18) using Monte Carlo techniques. More precisely, we will show how we can build unbiased estimates of this gradient by sampling from the variational (or an auxiliary) distribution without having to compute the gradient of the ELBO analytically [24,79,80].

5.1.1. Pathwise Gradients

The idea of this approach is to exploit reparameterizations of the variational distribution in terms of deterministic transformations of a noise distribution [83,84]. A distribution $q(\boldsymbol{r}|\boldsymbol{\nu})$ is reparameterizable if it can be expressed as

$$\begin{aligned} \boldsymbol{\epsilon} &\sim q(\boldsymbol{\epsilon}), \\ \boldsymbol{r} &= t(\boldsymbol{\epsilon}; \boldsymbol{\nu}), \end{aligned} \tag{19}$$

where $\boldsymbol{\epsilon}$ does not depend on parameter $\boldsymbol{\nu}$ and $t(\cdot; \boldsymbol{\nu})$ is a deterministic function which encapsulates the dependence of $\boldsymbol{r}$ with respect to $\boldsymbol{\nu}$. This transforms the expectation over $\boldsymbol{r}$ to an expectation over $\boldsymbol{\epsilon}$. By exploiting this reparameterization property, we can express the gradient of $\mathcal{L}$ in Equation (18) as [23,85,86],

$$\begin{aligned} \nabla_{\boldsymbol{\nu}} \mathcal{L}(\boldsymbol{\nu}) &= \nabla_{\boldsymbol{\nu}} \mathbb{E}_{\boldsymbol{R}}[g(\boldsymbol{r}, \boldsymbol{\nu})] \\ &= \nabla_{\boldsymbol{\nu}} \mathbb{E}_{\boldsymbol{\epsilon}}[g(t(\boldsymbol{\epsilon}; \boldsymbol{\nu}), \boldsymbol{\nu})] \\ &= \mathbb{E}_{\boldsymbol{\epsilon}}[\nabla_{\boldsymbol{\nu}} g(t(\boldsymbol{\epsilon}; \boldsymbol{\nu}), \boldsymbol{\nu})] \\ &= \mathbb{E}_{\boldsymbol{\epsilon}}\left[\nabla_t g(t(\boldsymbol{\epsilon}; \boldsymbol{\nu}), \boldsymbol{\nu})^T \nabla_{\boldsymbol{\nu}} t(\boldsymbol{\epsilon}; \boldsymbol{\nu}) + \nabla_{\boldsymbol{\nu}} g(t(\boldsymbol{\epsilon}; \boldsymbol{\nu}), \boldsymbol{\nu})\right] \\ &= \mathbb{E}_{\boldsymbol{\epsilon}}\left[\nabla_{\boldsymbol{r}} g(\boldsymbol{r}, \boldsymbol{\nu})^T \nabla_{\boldsymbol{\nu}} t(\epsilon; \boldsymbol{\nu}) + \nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu})\right] \\ &= \mathbb{E}_{\boldsymbol{\epsilon}}\left[\nabla_{\boldsymbol{r}} g(\boldsymbol{r}, \boldsymbol{\nu})^T \nabla_{\boldsymbol{\nu}} t(\epsilon; \boldsymbol{\nu})\right]. \end{aligned} \tag{20}$$

In the last step we have exploited that $\mathbb{E}_{\boldsymbol{\epsilon}}[\nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu})] = 0$. To see this, we first utilize that

$$\mathbb{E}_{\boldsymbol{\epsilon}}[\nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu})] = \int q(\boldsymbol{\epsilon}) \nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu}) d\boldsymbol{\epsilon} = \int q(\boldsymbol{r}|\boldsymbol{\nu}) \nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu}) d\boldsymbol{r} = \mathbb{E}_{\boldsymbol{R}}[\nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu})].$$

Next, as $g(\boldsymbol{r}, \boldsymbol{\nu}) = \ln p(\boldsymbol{x}, \boldsymbol{r}) - \ln q(\boldsymbol{r}|\boldsymbol{\nu})$, it follows that $\nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu}) = -\nabla_{\boldsymbol{\nu}} \ln q(\boldsymbol{r}|\boldsymbol{\nu})$. Finally, since $\mathbb{E}_{\boldsymbol{R}}[\nabla_{\boldsymbol{\nu}} \ln q(\boldsymbol{r}|\boldsymbol{\nu})] = 0$, we have that $\mathbb{E}_{\boldsymbol{\epsilon}}[\nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu})] = 0$.

Note that once we employ this reparameterization trick, the gradient enters the expectation, and afterwards we simply apply the chain rule of derivatives. Here, it is also worth noticing that the gradient estimator is informed by the gradient with respect to $\boldsymbol{g}$, which gives the direction of the maximum posterior mode (we shall return to this issue in Section 5.1.2).

Example 7. *The normal distribution is the best known example where this technique can be applied: a variable* $W \sim \mathcal{N}(\mu, \sigma^2)$ *can be reparameterized as* $\epsilon \sim \mathcal{N}(0, 1)$ *and* $W = \mu + \sigma\epsilon$*. So, by exploiting this reparameterization, we can compute the gradient of stochastic functions as the one defined in Figure 6, i.e., compute* $\nabla_\mu \mathbb{E}_Z[(Z-5)^2]$*, where* $Z \sim \mathcal{N}(\mu, 1)$*,*

$$\nabla_\mu \mathbb{E}_Z\left[(Z-5)^2\right] = \mathbb{E}_\epsilon\left[\nabla_\mu (\mu + \epsilon - 5)^2\right] = \mathbb{E}_\epsilon[2(\mu + \epsilon - 5)] = 2(\mu - 5).$$

In practice, this expectation is approximated using Monte Carlo sampling,

$$\nabla_\mu \mathbb{E}_Z[(Z-5)^2] \approx \frac{1}{K}\sum_{i=1}^{K} 2(\mu+\epsilon_i-5) \quad \epsilon_i \sim \mathcal{N}(0,1).$$

In terms of SCGs, this reparameterization trick can be captured by the transformation of the (original) SCG shown in Figure 8 to the SCG shown in Figure 9. For the transformed SCG, the underlying CG (exemplified in Figure 6) can be readily applied and from automatic differentiation we obtain unbiased estimates of the gradients of the ELBO.

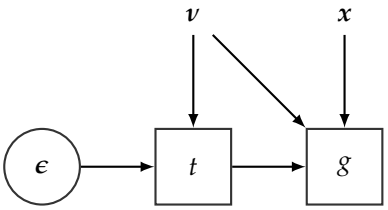


Figure 9. Reparameterized SCG representing the ELBO function $\mathcal{L}(\boldsymbol{\nu})$.

More generally, and pertinently, through the reparameterization trick we can define a CG representation of the ELBO function $\mathcal{L}$, which in turn can be used for computing a Monte Carlo estimation of $\mathcal{L}$,

$$\hat{\mathcal{L}} = \frac{1}{K}\sum_{i=1}^{K} \ln p(\boldsymbol{x}, t(\boldsymbol{\epsilon}_i;\boldsymbol{\nu})) - \ln q(t(\boldsymbol{\epsilon}_i;\boldsymbol{\nu})|\boldsymbol{\nu}) \quad \boldsymbol{\epsilon}_i \sim q(\boldsymbol{\epsilon}), \tag{21}$$

and the associated automatic differentiation engine of the CG can be used for finding the derivatives of $\mathcal{L}$ (cf. Equation (20)). The CG thus also serves as a generic tool for abstracting away and hiding the details of the gradient calculations from the user.

The applicability of the reparameterization trick only extends to distributions that can be expressed in the form shown in Equation (19). Fortunately, [87] recently introduced an implicit reparameterization approach, which apply to a wider range of distributions including Gamma, Beta, Dirichlet and von Mises (i.e., distributions not covered by Equation (19)). This method computes the gradient of $\mathcal{L}$ as

$$\nabla_{\boldsymbol{\nu}} \mathcal{L}(\boldsymbol{\nu}) = -\mathbb{E}_{\boldsymbol{R}}\left[\frac{\nabla_{\boldsymbol{r}} g(\boldsymbol{r},\boldsymbol{\nu})^T \nabla_{\boldsymbol{\nu}} F(\boldsymbol{r};\boldsymbol{\nu})}{q(\boldsymbol{r}|\boldsymbol{\nu})}\right], \tag{22}$$

where $F(\boldsymbol{r};\boldsymbol{\nu})$ is the cumulative distribution function of $q(\boldsymbol{r}|\boldsymbol{\nu})$. Other similar approaches have been proposed for models with discrete latent random variables [88,89].

The above family of gradient estimators usually have lower variance than other methods [44] and, in many cases, they can even provide good estimates with a single Monte Carlo sample. However, the estimators only apply to distributions that support explicit or implicit reparameterizations. Although many distributions provide this support, there are also other relevant distributions, such as the multinomial distribution, which cannot be handled using either of the reparameterization techniques.

Example 8. *We end this sub-section with our running example about VAEs. In this case, we consider a VAE without an encoder network; the encoder network will be discussed in the Section 5.2. This model can thus be seen as a non-linear PCA model (the non-linearity is defined in terms of an ANN) as described in Example 6. For this model, the ELBO function can be expressed as*

$$\begin{aligned}\mathcal{L}(\boldsymbol{\lambda},\boldsymbol{\phi}) \quad = \quad & \mathbb{E}_q[\ln p(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{\beta})] + \mathbb{E}_q[\ln p(\boldsymbol{z})] + \mathbb{E}_q[\ln p(\boldsymbol{\beta})] \\ & -\mathbb{E}_q[\ln q(\boldsymbol{z}|\boldsymbol{\phi}) - \mathbb{E}_q[\ln q(\boldsymbol{\beta}|\boldsymbol{\lambda})].\end{aligned}$$

Algorithm 4 gives a pseudo-code specification of the SCG defining the ELBO function using the reparameterization trick; here we only use a single sample from the variational distribution $q(\boldsymbol{\beta},\boldsymbol{Z}|\boldsymbol{\lambda},\boldsymbol{\phi})$ in reparameterized form. The definition of the ELBO function $\mathcal{L}$ is introduced together

with the specification of the decoder network, hence gradients wrt. the variational parameters can be readily computed and optimized using standard algorithms.

Algorithm 4 Pseudo-code for defining the ELBO function $\hat{\mathcal{L}}$, and by translation the SCG, of a VAE with no encoder network (see Algorithm 3). We use a single sample to compute the Monte Carlo estimate of $\hat{\mathcal{L}}$ (see Equation (21)). $\ln p_{\mathcal{N}}(\cdot|\cdot,\cdot)$ denotes the log-probability function of a normal distribution.

input Data: $\boldsymbol{x}_{train}$, Variational Parameters: $\boldsymbol{\lambda}, \boldsymbol{\phi}$
Sample (using reparameterization) from $q(\boldsymbol{\beta}|\boldsymbol{\lambda})$ and $q(\boldsymbol{z}|\boldsymbol{\phi})$.
$\epsilon_{\alpha_0}, \epsilon_{\beta_0}, \epsilon_{\alpha_1}, \epsilon_{\beta_1}, \epsilon_z \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
$\boldsymbol{\alpha}_0 = \boldsymbol{\lambda}_{\alpha_0,\mu} + \epsilon_{\alpha_0}\boldsymbol{\lambda}_{\alpha_0,\sigma}, \quad \boldsymbol{\beta}_0 = \boldsymbol{\lambda}_{\beta_0,\mu} + \epsilon_{\beta_0}\boldsymbol{\lambda}_{\beta_0,\sigma}$
$\boldsymbol{\alpha}_1 = \boldsymbol{\lambda}_{\alpha_1,\mu} + \epsilon_{\alpha_1}\boldsymbol{\lambda}_{\alpha_1,\sigma}, \quad \boldsymbol{\beta}_1 = \boldsymbol{\lambda}_{\beta_1,\mu} + \epsilon_{\beta_1}\boldsymbol{\lambda}_{\beta_1,\sigma}$
$\boldsymbol{z} = \boldsymbol{\phi}_{z,\mu} + \epsilon_z \boldsymbol{\phi}_{z,\sigma}$
Pass the variational sample $\boldsymbol{z}$ through the decoder ANN
$\boldsymbol{h}_0 = relu(\boldsymbol{z}\boldsymbol{\beta}_0^T + \boldsymbol{\alpha}_0)$
$\boldsymbol{\mu}_x = \boldsymbol{h}_0\boldsymbol{\beta}_1^T + \boldsymbol{\alpha}_1$
Define the "energy part" of the ELBO function $E_q[\ln p(x_{train}, \boldsymbol{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})]$.
$\mathcal{L} = \ln p_{\mathcal{N}}(\boldsymbol{x}_{train}|\boldsymbol{\mu}_x, \sigma_x^2\boldsymbol{I})$
$\mathcal{L} = \mathcal{L} + \ln p_{\mathcal{N}}(\boldsymbol{z}|\mathbf{0}, \boldsymbol{I}) + \sum_i \ln p_{\mathcal{N}}(\boldsymbol{\alpha}_i|\mathbf{0}, \boldsymbol{I}) + \ln p_{\mathcal{N}}(\boldsymbol{\beta}_i|\mathbf{0}, \boldsymbol{I})$
Define the "entropy part" of the ELBO function $\mathbb{E}_q[\ln q(\boldsymbol{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})]$.
$\mathcal{L} = \mathcal{L} - \ln p_{\mathcal{N}}(\boldsymbol{z}|\boldsymbol{\phi}_{z,\mu}, \boldsymbol{\phi}_{z,\sigma}^2)$
$\mathcal{L} = \mathcal{L} - \sum_i \ln p_{\mathcal{N}}(\boldsymbol{\alpha}_i|\boldsymbol{\lambda}_{\alpha_i,\mu}, \boldsymbol{\lambda}_{\alpha_i,\sigma}^2) - \sum_i \ln p_{\mathcal{N}}(\boldsymbol{\beta}_i|\boldsymbol{\lambda}_{\beta_i,\mu}, \boldsymbol{\lambda}_{\beta_i,\sigma}^2)$
return $\mathcal{L}$

5.1.2. Score Function Gradients

This is a classical method for gradient estimation, also known as the REINFORCE method [24,90,91]. It builds on the following generic transformations to compute the gradient of an expected value,

$$\begin{aligned}\nabla_{\boldsymbol{\nu}}\mathcal{L}(\boldsymbol{\nu}) &= \nabla_{\boldsymbol{\nu}} \int q(\boldsymbol{r}|\boldsymbol{\nu}) g(\boldsymbol{r}, \boldsymbol{\nu}) d\boldsymbol{r} \\ &= \int g(\boldsymbol{r}, \boldsymbol{\nu}) \nabla_{\boldsymbol{\nu}} q(\boldsymbol{r}|\boldsymbol{\nu}) + q(\boldsymbol{r}|\boldsymbol{\nu}) \nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu}) d\boldsymbol{r} \\ &= \int g(\boldsymbol{r}, \boldsymbol{\nu})\, q(\boldsymbol{r}|\boldsymbol{\nu}) \nabla_{\boldsymbol{\nu}} \ln q(\boldsymbol{r}|\boldsymbol{\nu}) + q(\boldsymbol{r}|\boldsymbol{\nu}) \nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu}) d\boldsymbol{r} \\ &= \mathbb{E}_{\boldsymbol{R}}[g(\boldsymbol{r}, \boldsymbol{\nu})\, \nabla_{\boldsymbol{\nu}} \ln q(\boldsymbol{r}|\boldsymbol{\nu}) + \nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu})].\end{aligned} \tag{23}$$

Following the discussion surrounding the derivation of Equation (20), we have that $\mathbb{E}_{\boldsymbol{R}}[\nabla_{\boldsymbol{\nu}} g(\boldsymbol{r}, \boldsymbol{\nu})] = \mathbb{E}_{\boldsymbol{R}}[-\nabla_{\boldsymbol{\nu}} \ln q(\boldsymbol{r}|\boldsymbol{\nu})] = 0$ and the gradient of the ELBO therefore simplifies to

$$\nabla_{\boldsymbol{\nu}}\mathcal{L}(\boldsymbol{\nu}) = \mathbb{E}_{\boldsymbol{R}}[g(\boldsymbol{r}, \boldsymbol{\nu})\, \nabla_{\boldsymbol{\nu}} \ln q(\boldsymbol{r}|\boldsymbol{\nu})]. \tag{24}$$

The term $\nabla_{\boldsymbol{\nu}} \ln q(\boldsymbol{r}|\boldsymbol{\nu})$ (the gradient of the log of a probability distribution) is referred to as the score function, hence the name of the method.

From the above equation, we obtain unbiased estimates of the gradient by sampling from $q(\boldsymbol{r}|\boldsymbol{\nu})$. This method is general in the sense that it only requires being able to evaluate the function $g(\boldsymbol{r}, \boldsymbol{\nu})$ and computing the score function, $\nabla_{\boldsymbol{\nu}} \ln q(\boldsymbol{r}|\boldsymbol{\nu})$. In consequence, the method applies to a wide range of models, including those covered by the pathwise gradient estimator. However, in practice, the score function gradient often yields high variance estimates when the dimensionality of $\boldsymbol{\nu}$ is relatively high. This is accentuated by the gradient estimator only being guided by the gradient of the (log of the) variational distribution and not the likelihood term of the model (which was the case for the pathwise gradient estimator). To reduce the variance, one often relies on variance reduction techniques for improved performance [24,86,92,93], but, still, in a practical setting the

score function estimator mostly serve as the fall-back method when the pathwise gradient estimator is not applicable.

Example 9. *We revisit Example 7. We have to compute the gradient of an expectation* $\nabla_\mu \mathbb{E}_Z[(Z-5)^2]$, *where* $Z \sim \mathcal{N}(\mu, 1)$. *By applying the score function gradient estimator, we get*

$$\begin{aligned} \nabla_\mu \mathbb{E}_Z[(Z-5)^2] &= \mathbb{E}_Z[(Z-5)^2 \nabla_\mu \ln N(Z|\mu,1)] \\ &= \mathbb{E}_Z\left[(Z-5)^2 \nabla_\mu \left(-\frac{1}{2}(Z-\mu)^2\right)\right] \\ &= \mathbb{E}_Z[(Z-5)^2(Z-\mu)], \end{aligned}$$

which can be approximated by Monte Carlo sampling, $\nabla_\mu \mathbb{E}_Z[(Z-5)^2] \approx \frac{1}{K}\sum_{i=1}^K (z_i-5)^2(z_i-\mu)$, *where* z_i *are samples from* $\mathcal{N}(\mu, 1)$.

In [94], it is detailed an elegant implementation of this technique using SCGs.

5.2. ELBO Optimization with Amortized Variational Inference

In principle, we can address the optimization of the ELBO function using an off-the-shelf gradient ascent algorithm combined with the techniques presented in the previous section. The ELBO function $\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi})$, in this case, is again expressed in terms of global variational parameters $\boldsymbol{\lambda}$ (defining the variational distribution over the global latent variables $q(\boldsymbol{\beta}|\boldsymbol{\lambda})$) and in terms of local variational parameters $\boldsymbol{\phi}$ (defining the variational distribution over the local latent variables $q(\boldsymbol{z}_i|\boldsymbol{\phi}_i)$); we implicitly assume that the variational posterior fully factorizes, as shown in Equation (5), although this assumption is not crucial for the discussion below. Unfortunately, as the number of local variational parameters $\boldsymbol{\phi} = (\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_N)$ grows with the size N of the data set, straight-forward optimization using gradient ascent quickly becomes computationally infeasible as the size of the data grows.

To address this issue we can rely on some of the tricks detailed in Section 2.3. First, we can express $\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi})$ only in terms of $\boldsymbol{\lambda}$, as previously shown in Equations (11) and (12),

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_q[\ln p(\boldsymbol{\beta})] - \mathbb{E}_q[\ln q(\boldsymbol{\beta}|\boldsymbol{\lambda})] + \sum_{i=1}^N \max_{\boldsymbol{\phi}_i} (\mathbb{E}_q[\ln p(\boldsymbol{x}_i, \boldsymbol{Z}_i|\boldsymbol{\beta})] - \mathbb{E}_q[\ln q(\boldsymbol{Z}_i|\boldsymbol{\phi}_i)]).$$

As done in Section 2.3, we can get unbiased noisy estimates of this ELBO by data subsampling. If I is a randomly chosen data index, $I \in \{1, \ldots, N\}$, and

$$\mathcal{L}_I(\boldsymbol{\lambda}) = \mathbb{E}_q[\ln p(\boldsymbol{\beta})] - \mathbb{E}_q[\ln q(\boldsymbol{\beta}|\boldsymbol{\lambda})] + N \max_{\boldsymbol{\phi}_I} (\mathbb{E}_q[\ln p(\boldsymbol{x}_I, \boldsymbol{Z}_I|\boldsymbol{\beta})] - \mathbb{E}_q[\ln q(\boldsymbol{Z}_I|\boldsymbol{\phi}_I)]),$$

then the expectation of $\mathcal{L}_I(\boldsymbol{\lambda})$ is equal to $\mathcal{L}(\boldsymbol{\lambda})$ [19] and computing the gradient of $\mathcal{L}_I(\boldsymbol{\lambda})$ wrt. $\boldsymbol{\lambda}$ will give us a noisy unbiased estimate. However, in this case, we require solving an maximization problem for each subsampled data point (i.e., $\max_{\boldsymbol{\phi}_I}$). In the case of conjugate exponential models, this inner maximization step can be computed in closed form as shown in Equation (10). However, for models outside the conjugate exponential family, we would have to resort to iterative algorithms, based on the methods described in Section 5.1, making the approach infeasible.

Amortized inference [81,82] aims to address this problem by learning a mapping function, denoted by $\boldsymbol{s}$, between $\boldsymbol{x}_i$ and $\boldsymbol{\phi}_i$ parameterized by $\boldsymbol{\theta}$, i.e., $\boldsymbol{\phi}_i = \boldsymbol{s}(\boldsymbol{x}_i|\boldsymbol{\theta})$. Hence, $\mathcal{L}_I(\boldsymbol{\lambda})$ is expressed as $\mathcal{L}_I(\boldsymbol{\lambda}, \boldsymbol{\theta})$,

$$\begin{aligned} \mathcal{L}_I(\boldsymbol{\lambda}, \boldsymbol{\theta}) &= \mathbb{E}_q[\ln p(\boldsymbol{\beta})] - \mathbb{E}_q[\ln q(\boldsymbol{\beta}|\boldsymbol{\lambda})] \\ &+ N \cdot \mathbb{E}_q[\ln p(\boldsymbol{x}_I, \boldsymbol{Z}_I|\boldsymbol{\beta})] - N \cdot \mathbb{E}_q[\ln q(\boldsymbol{Z}_I|\boldsymbol{x}_I, \boldsymbol{\phi}_I = \boldsymbol{s}(\boldsymbol{x}_i|\boldsymbol{\theta}))]. \end{aligned}$$

The parameter vector $\boldsymbol{\theta}$ is shared among all the data points and does not grow with the data set as was previously the case when each data point was assigned its own local variational parameters, $\boldsymbol{\phi} = \{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_N\}$. On the other hand, amortized inference assumes that the parameterized function $\boldsymbol{s}$ is flexible enough to allow for the estimation of the local variational parameters $\boldsymbol{\phi}_i$ from the data points $\boldsymbol{x}_i$. Thus, the family of variational distributions defined by this technique,

$$q(\boldsymbol{\beta}, \boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\theta}) = q(\boldsymbol{\beta}|\boldsymbol{\lambda}) \prod_{i=1}^{N} q(\boldsymbol{z}_i|\boldsymbol{x}_i, \boldsymbol{\phi}_i = \boldsymbol{s}(\boldsymbol{x}_i|\boldsymbol{\theta})),$$

is more restricted than the one defined in Equation (5), which directly depends of $\boldsymbol{\lambda}$ and $\boldsymbol{\phi}$. So, there is a trade-off between flexibility in the variational approximation and computational efficiency when applying amortized inference techniques.

Note that the amortized function greatly simplifies the use of the model when making predictions over unseen data $\boldsymbol{x}'$. If we need the posterior $p(\boldsymbol{z}'|\boldsymbol{x}')$ over a new data sample $\boldsymbol{x}'$ (e.g., for dimensionality reduction when using a VAE model), we just need to invoke the learnt amortized inference function to recover this posterior, $q(\boldsymbol{z}'|\boldsymbol{\phi} = \boldsymbol{s}(\boldsymbol{x}'|\boldsymbol{\theta}^\star))$.

An unbiased estimate of the gradient of $\mathcal{L}_I(\boldsymbol{\lambda}, \boldsymbol{\theta})$ wrt to $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$ can be computed using the techniques described in the previous section, as both affect an expectation term. Note that the unbiased estimate of the gradient of $\mathcal{L}_I(\boldsymbol{\lambda}, \boldsymbol{\theta})$ is also an unbiased estimate of the gradient of $\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\theta})$. Similar to Equation (13), the ELBO can be maximized by following noisy estimates of the gradient,

$$\begin{aligned} \boldsymbol{\lambda}_{t+1} &= \boldsymbol{\lambda}_t + \rho_t \hat{\nabla}_{\boldsymbol{\lambda}} \mathcal{L}_{I_t}(\boldsymbol{\lambda}_t, \boldsymbol{\theta}_t), \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \rho_t \hat{\nabla}_{\boldsymbol{\theta}} \mathcal{L}_{I_t}(\boldsymbol{\lambda}_t, \boldsymbol{\theta}_t) \end{aligned} \tag{25}$$

where I_t are the indexes of randomly subsampled data points at time step t.

Example 10. *We finally arrive at the original formulation of VAEs, which includes an amortized inference function linking the data samples with the latent variables $\boldsymbol{Z}$. This amortized function takes the form of a neural network and is referred to as the encoder network because it translates an observation $\boldsymbol{X}$ to (a distribution over) its hidden representation $\boldsymbol{Z}$; recall that the decoder network (part of the model specification) links the latent variables $\boldsymbol{Z}$ to (a distribution over) the observable variables $\boldsymbol{X}$. The existence of these two networks, the encoder and the decoder, establishes a direct link with the previously known auto-encoder networks [95]. In this example, both the encoder and the decoder network have a single hidden layer with a relu activation function.*

Algorithm 5 shows pseudo-code defining the ELBO function associated with this model. The model falls outside the conjugate exponential family, but due to distributional assumptions of the VAE's we can estimate the gradients by applying the reparameterization trick (see Section 5.1.1). Specifically, from the encoder network, we sample from the variational distribution over $\boldsymbol{Z}_I$ given $\boldsymbol{X}_I$ (in reparameterized form), and at the end of the algorithm, we define the ELBO function $\mathcal{L}_I$, which includes the definition of the decoder network. As for the previous example, the pseudo-code specification directly translates into a computational graph. From this representation, the gradients wrt. the variational parameters can be readily computed and the ELBO function optimized using, in this case, stochastic gradient ascent or some of a variation hereof.

Figure 10 shows the two-dimensional latent embedding found by the non-linear probabilistic PCA (Left; reproduced from Figure 5) and VAE (Right) for the same reduced MNIST data set used previously. The three classes are clearly separated in latent space for both models.

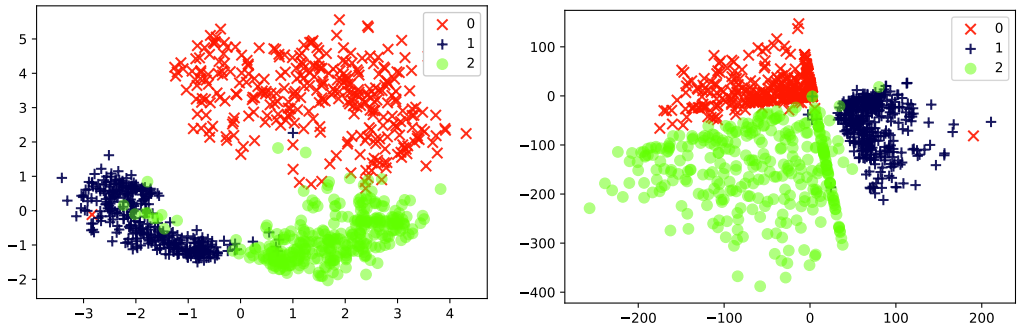


Figure 10. Two-dimensional latent representation of the the MNIST dataset resulting of applying: (**Left**) a non-linear probabilistic PCA, and (**Right**) a VAE. The ANNs of the non-linear PCA and the ones defining the VAE's decoder and encoder contain a single hidden layer of size 100.

Algorithm 5 Pseudo-code for the estimation of the ELBO function $\mathcal{L}_I$ of a Variational Auto-encoder. We use a single sample to compute the Monte Carlo estimation of $\hat{\mathcal{L}}$ (see Equation (21)). $\ln p_{\mathcal{N}}(\cdot|\cdot,\cdot)$ denotes the log-probability function of a Normal distribution.

input Data: $\boldsymbol{x}_I$ a single data-sample, N size of the data, Variational Parameters: $\boldsymbol{\lambda}, \boldsymbol{\theta}$

Sample (using reparameterization) from $q(\boldsymbol{\beta}|\boldsymbol{\lambda})$.

$\epsilon_{\theta_0}, \epsilon_{\theta'_0}, \epsilon_{\theta_1}, \epsilon_{\theta'_1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

$\boldsymbol{\theta}_0 = \boldsymbol{\lambda}_{\theta_0,\mu} + \epsilon_{\theta_0}\boldsymbol{\lambda}_{\theta_0,\sigma}, \quad \boldsymbol{\theta}'_0 = \boldsymbol{\lambda}_{\theta'_0,\mu} + \epsilon_{\theta'_0}\boldsymbol{\lambda}_{\theta'_0,\sigma}$

$\boldsymbol{\theta}_1 = \boldsymbol{\lambda}_{\theta_1,\mu} + \epsilon_{\theta_1}\boldsymbol{\lambda}_{\theta_1,\sigma}, \quad \boldsymbol{\theta}'_1 = \boldsymbol{\lambda}_{\theta'_1,\mu} + \epsilon_{\theta'_1}\boldsymbol{\lambda}_{\theta'_1,\sigma}$

Pass $\boldsymbol{x}$ through the encoder network and sample $\boldsymbol{z}_I \sim q(\boldsymbol{z}|\boldsymbol{\phi} = \boldsymbol{s}(\boldsymbol{x}_I|\boldsymbol{\theta}))$

$\boldsymbol{h}_{z,0} = relu(\boldsymbol{x}_I\boldsymbol{\theta}_0^T + \boldsymbol{\theta}'_0)$

$\boldsymbol{h}_{z,1} = \boldsymbol{h}_{z,0}\boldsymbol{\theta}_1^T + \boldsymbol{\theta}'_1$

$\boldsymbol{h}_{z,1}$ contains both the mean, $\boldsymbol{h}_{z,1,\mu}$, and the scale, $\boldsymbol{h}_{z,1,\sigma}$.

$\epsilon_z \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

$\boldsymbol{z}_I = \boldsymbol{h}_{z,1,\mu} + \epsilon_z \boldsymbol{h}_{z,1,\sigma}$

Pass the variational sample $\boldsymbol{z}$ through the decoder network

$\epsilon_{\alpha_0}, \epsilon_{\beta_0}, \epsilon_{\alpha_1}, \epsilon_{\beta_1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

$\boldsymbol{\alpha}_0 = \boldsymbol{\lambda}_{\alpha_0,\mu} + \epsilon_{\alpha_0}\boldsymbol{\lambda}_{\alpha_0,\sigma}, \quad \boldsymbol{\beta}_0 = \boldsymbol{\lambda}_{\beta_0,\mu} + \epsilon_{\beta_0}\boldsymbol{\lambda}_{\beta_0,\sigma}$

$\boldsymbol{\alpha}_1 = \boldsymbol{\lambda}_{\alpha_1,\mu} + \epsilon_{\alpha_1}\boldsymbol{\lambda}_{\alpha_1,\sigma}, \quad \boldsymbol{\beta}_1 = \boldsymbol{\lambda}_{\beta_1,\mu} + \epsilon_{\beta_1}\boldsymbol{\lambda}_{\beta_1,\sigma}$

$\boldsymbol{h}_0 = relu(\boldsymbol{z}_I\boldsymbol{\beta}_0^T + \boldsymbol{\alpha}_0)$

$\boldsymbol{\mu}_x = \boldsymbol{h}_0\boldsymbol{\beta}_1^T + \boldsymbol{\alpha}_1$

Define the "energy part" of the ELBO function

$\mathcal{L}_I = N \cdot \ln p_{\mathcal{N}}(\boldsymbol{x}_I|\boldsymbol{\mu}_x, \sigma_x^2\boldsymbol{I}) + N \cdot \ln p_{\mathcal{N}}(\boldsymbol{z}_I|\mathbf{0}, \boldsymbol{I})$

$\mathcal{L}_I = \mathcal{L}_I + \sum_i \ln p_{\mathcal{N}}(\boldsymbol{\alpha}_i|\mathbf{0}, \boldsymbol{I}) + \ln p_{\mathcal{N}}(\boldsymbol{\beta}_i|\mathbf{0}, \boldsymbol{I})$

Define the "entropy part" of the ELBO function

$\mathcal{L}_I = \mathcal{L}_I - N \cdot \ln p_{\mathcal{N}}(\boldsymbol{z}_I|\boldsymbol{h}'_{1,\mu}, \boldsymbol{h}'_{1,\sigma})$

$\mathcal{L}_I = \mathcal{L}_I - \sum_i \ln p_{\mathcal{N}}(\boldsymbol{\alpha}_i|\boldsymbol{\lambda}_{\alpha_i,\mu}, \boldsymbol{\lambda}^2_{\alpha_i,\sigma}) + \ln p_{\mathcal{N}}(\boldsymbol{\beta}_i|\boldsymbol{\lambda}_{\beta_i,\mu}, \boldsymbol{\lambda}^2_{\beta_i,\sigma})$

return $\mathcal{L}_I$

5.3. Discussion

Although the different variational techniques we have presented in this section can be used to make inference over probabilistic models with deep neural networks, each of them presents different trade-offs which should be taken into account. The black-box methods presented in Section 5.1 are extremely powerful in theory, as they make only few assumptions about the probabilistic model. However, this comes at a price: a high variance is introduced when the gradient is estimated using Monte Carlo sampling, and this can strongly hinder the convergence of the underlying optimization method. In this sense, pathwise gradients (Section 5.1.1) should be preferred to score function gradients (Section 5.1.2), because the former produces less noisy gradient estimates than the latter. Score function gradients should only be used when the pathwise gradient estimator is not

applicable. Furthermore, as commented before, they have to be implemented using the variance reduction technique [24,86,92,93].

Amortized variational inference (Section 5.2) is usually the default choice. Although amortized variational inference reduces the flexibility of the variational approximation, it greatly improves the computational efficiency, to the point that most standard LVMs cannot be used for analyzing any meaningful data sample without this technique. The lack of flexibility of amortized variational inference can be alleviated by the use of more expressive mapping functions (i.e., more powerful neural networks).

6. Probabilistic Programming Languages

One of the main reasons for the wide adoption of deep learning has been the availability of (open-source) software tools containing robust and well-tested implementations of the main building blocks for defining and learning DNNs [34,35]. Recently, a new wave of software tools have appeared, building on top of these deep learning frameworks in order to accommodate modern probabilistic models containing DNNs [29–33,96–98]. These software tools usually fall under the umbrella term *probabilistic programming languages* (PPLs) [37,38], and support methods for learning and reasoning about complex probabilistic models. Although PPLs have been present in the field of machine learning for many years, traditional PPLs have mainly focused on defining languages for expressing (more restricted types of) probabilistic models [8] with only little focus on issues such as scalability. The advent of deep learning and the introduction of probabilistic models containing DNNs has motivated the development of a new family of PPLs offering support for flexible and complex models as well as scalable inference. Some of the main PPLs supporting the definition of models with DNNs are listed below.

- Edward2 [29,30], developed by Google, is a fast Python PPL built over TensorFlow-probability [34,78]. This framework is compatible with neural networks defined with Keras [99].
- InferPy [32,33] is a Python package built on top of Edward which focuses on the ease of use. It provides a compact and simple way to code probabilistic models with DNNs, at the expense of slightly reducing expressibility and flexibility.
- Pyro [31], developed by Uber, is based on Pytorch and allows for the definition of probabilistic models with DNNs in Python [35].
- PyMC3 [96] is a PPL written in Python that uses Theano [100] as calculus framework.
- Stan [97] is a PPL in C++ for statistical modeling and high-performance statistical computation. Even though the integration with DNNs is not natively supported, an extension for this was proposed [101].
- Turing.jl [98] is a Julia library for probabilistic programming inference. Originally, Mote Carlo methods were only considered, but recent releases of this library also provide support for variational inference.

There are other examples of PPLs, but these alternatively PPLs do typically not support DNNs in the model specifications. Examples of such languages include: Birch [102] is a C++ library with inference algorithms based on Sequential Monte Carlo (SMC); Bean Machine [103] is a declarative PPL in Python with a special focus on compositional and block inference; Infer.net [104] is a framework for running Bayesian inference in graphical models which can also be used for probabilistic programming.

7. Conclusions and Open Issues

In this paper, we have discussed the recent breakthroughs in approximate inference for PGMs. In particular, we have considered variational inference (VI), a scalable and versatile approach for doing approximate inference in probabilistic models. The versatility of VI enables the data analyst to build flexible models, without the constraints of limiting modeling assumptions (e.g., linear relationship between random variables). VI is supported by a sound and well-understood mathematical foundation and exhibits good theoretical properties. For instance, VI is (theoretically) guaranteed to converge to an

approximate posterior q, contained in a set of viable approximations $\mathcal{Q}$, that corresponds to a (local) maximum of the ELBO function, as defined in Equation (8). Nevertheless, variational inference often encounters difficulties when used in practice. Different random initializations of the parameter space can have significant effect on the end-result and, unless extra care is taken, issues wrt. numerical stability may also endanger the robustness of the obtained results. More research is needed to develop practical guidelines for using variational inference.

As the power of deep neural networks has entered in PGMs, the PGM community has largely responded enthusiastically, embracing the new extensions to the PGM toolbox and used them eagerly. This has lead to new and interesting tools and models, some of which are discussed in this paper. However, we also see a potential pitfall here: The trend is to move away from the modeling paradigm that the PGM community has traditionally held in so high regard and instead move towards catch-all LVMs (like the one depicted in Figure 1). These models "*let the data speak for itself*", but at the cost of interpretability. PGMs are typically seen as fully transparent models, but risk becoming more opaque with the increased emphasis on LVMs parameterized through deep neural networks and driven by general purpose inference techniques. Initial steps have, however, already been made to leverage the PGM's modeling power also in this context (e.g., Ref. [68] combines structured latent variable representations with non-linear likelihood functions), but a seamless and transparent integration of neural networks and PGMs still requires further developments: Firstly, in a PGM model where some variables are defined using traditional probability distributions and others use deep neural networks, parts of the model may lend itself to efficient approximative inference (e.g., using VMP as described in Section 2.4), while others do not. An inference engine that utilizes an efficient (mixed) strategy approach for approximate inference in such models would be a valuable contribution. Secondly, VI reduces the inference problem to a continuous optimization problem. However, this is insufficient if the model contains latent categorical variables. While some PPLs, like the current release (Pyro version 1.5.1.) of Pyro [31], implements automatic enumeration over discrete latent variables, alternative approaches like the Concrete distribution [105] are also gaining some popularity. Thirdly, with a combined focus on inference and modeling, we may balance the results of performing approximate inference in "exact models" and performing exact inference in "approximate models" (with the understanding that all models are approximations). Here, the modeling approach may lead to better understood approximations, and therefore give results that are more robust and better suited for decision support.

Author Contributions: Conceptualization, A.R.M., H.L., T.D.N. and A.S.; methodology, A.R.M., H.L., T.D.N. and A.S.; software, A.R.M. and R.C.; validation, A.R.M. and R.C.; formal analysis, H.L., T.D.N. and A.S.; investigation, A.R.M., R.C., H.L., T.D.N. and A.S.; writing–original draft preparation, A.R.M., R.C., H.L., T.D.N. and A.S.; visualization, R.C.; supervision, H.L., T.D.N. and A.S.; funding acquisition, A.R.M. and A.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been partly funded by the Spanish Ministry of Science and Innovation, through projects TIN2015-74368-JIN, TIN2016-77902-C3-3-P, PID2019-106758GB-C31, PID2019-106758GB-C32 and by ERDF funds.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The running examples of the paper together with other basic models are available at https://github.com/PGM-Lab/ProbModelsDNNs.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*; Morgan Kaufmann Publishers: San Mateo, CA, USA, 1988.
2. Lauritzen, S.L. Propagation of probabilities, means, and variances in mixed graphical association models. *J. Am. Stat. Assoc.* **1992**, *87*, 1098–1108.
3. Russell, S.J.; Norvig, P. *Artificial Intelligence: A Modern Approach*; Pearson: Upper Saddle River, NJ, USA, 2016.
4. Hastie, T.; Tibshirani, R.; Friedman, J. *The Elements of Statistical Learning*; Springer: Berlin/Heidelberg, Germany, 2001.
5. Bishop, C.M. *Pattern Recognition and Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2006.
6. Murphy, K.P. *Machine Learning: A Probabilistic Perspective*; MIT Press: Cambridge, MA, USA, 2012.
7. Jensen, F.V.; Nielsen, T.D. *Bayesian Networks and Decision Graphs*; Springer: Berlin, Germany, 2007.
8. Koller, D.; Friedman, N. *Probabilistic Graphical Models: Principles and Techniques*; MIT Press: Cambridge, MA, USA, 2009.
9. Salmerón, A.; Rumí, R.; Langseth, H.; Nielsen, T.; Madsen, A. A review of inference algorithms for hybrid Bayesian networks. *J. Artif. Intell. Res.* **2018**, *62*, 799–828.
10. Gilks, W.R.; Richardson, S.; Spiegelhalter, D. *Markov Chain Monte Carlo in Practice*; Chapman and Hall/CRC: Boca Raton, FL, USA, 1995.
11. Salmerón, A.; Cano, A.; Moral, S. Importance sampling in Bayesian networks using probability trees. *Comput. Stat. Data Anal.* **2000**, *34*, 387–413.
12. Plummer, M. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna, Austria, 20–22 March 2003; Volume 124.
13. Blei, D.M. Build, compute, critique, repeat: Data analysis with latent variable models. *Annu. Rev. Stat. Its Appl.* **2014**, *1*, 203–232.
14. Murphy, K.P.; Weiss, Y.; Jordan, M.I. Loopy belief propagation for approximate inference: An empirical study. In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm Sweden, 30 July–1 August 1999; Morgan Kaufmann Publishers: San Fransisco, CA, USA, 1999; pp. 467–475.
15. Minka, T.P. Expectation propagation for approximate Bayesian inference. In Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence, Seattle, WA, USA, 2–5 August 2001; Morgan Kaufmann Publishers: San Fransisco, CA, USA, 2001; pp. 362–369.
16. Wainwright, M.J.; Jordan, M.I. *Graphical Models, Exponential Families, and Variational Inference*; Foundations and Trends® in Machine Learning; Now Publishers Inc.: Norwell, MA, USA; 2008; Volume 1, pp. 1–305.
17. Jordan, M.I.; Ghahramani, Z.; Jaakkola, T.S.; Saul, L.K. An introduction to variational methods for graphical models. *Mach. Learn.* **1999**, *37*, 183–233.
18. Bottou, L. Large-scale machine learning with stochastic gradient descent. In Proceedings of the COMPSTAT'2010, Paris, France, 22–27 August 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 177–186.
19. Hoffman, M.D.; Blei, D.M.; Wang, C.; Paisley, J. Stochastic Variational Inference. *J. Mach. Learn. Res.* **2013**, *14*, 1303–1347.
20. Barndorff-Nielsen, O. *Information and Exponential Families in Statistical Theory*; John Wiley & Sons: Hoboken, NJ, USA, 2014.
21. Winn, J.M.; Bishop, C.M. Variational Message Passing. *J. Mach. Learn. Res.* **2005**, *6*, 661–694.
22. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
23. Kingma, D.P.; Welling, M. Auto-encoding variational Bayes. *arXiv* **2013**, arXiv:1312.6114.
24. Ranganath, R.; Gerrish, S.; Blei, D. Black box variational inference. In Proceedings of the Artificial Intelligence and Statistics, Reykjavic, Iceland, 22–25 April 2014; pp. 814–822.
25. Hinton, G.E. Deep belief networks. *Scholarpedia* **2009**, *4*, 5947.
26. Hinton, G.E. A practical guide to training restricted Boltzmann machines. In *Neural Networks: Tricks of the Trade*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 599–619.
27. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, USA, 8–13 December 2014; pp. 2672–2680.
28. Salakhutdinov, R. Learning deep generative models. *Annu. Rev. Stat. Its Appl.* **2015**, *2*, 361–385.
29. Tran, D.; Kucukelbir, A.; Dieng, A.B.; Rudolph, M.; Liang, D.; Blei, D.M. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv* **2016**, arXiv:1610.09787.
30. Tran, D.; Hoffman, M.W.; Moore, D.; Suter, C.; Vasudevan, S.; Radul, A. Simple, distributed, and accelerated probabilistic programming. In Proceedings of the Advances in Neural Information Processing Systems, Montréal, QC, Canada, 3–8 December 2018; pp. 7608–7619.
31. Bingham, E.; Chen, J.P.; Jankowiak, M.; Obermeyer, F.; Pradhan, N.; Karaletsos, T.; Singh, R.; Szerlip, P.; Horsfall, P.; Goodman, N.D. Pyro: Deep Universal Probabilistic Programming. *arXiv* **2018**, arXiv:1810.09538.
32. Cabañas, R.; Salmerón, A.; Masegosa, A.R. InferPy: Probabilistic Modeling with TensorFlow Made Easy. *Knowl.-Based Syst.* **2019**, *168*, 25–27.
33. Cózar, J.; Cabañas, R.; Salmerón, A.; Masegosa, A.R. InferPy: Probabilistic Modeling with Deep Neural Networks Made Easy. *Neurocomputing* **2020**, *415*, 408–410.
34. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Software. Available online: https://www.tensorflow.org (accessed on 15 January 2021).

35. Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic differentiation in PyTorch. In Proceedings of the NIPS AutoDiff Workshop, Long Beach, CA, USA, 9 December 2017.
36. Zhang, C.; Bütepage, J.; Kjellström, H.; Mandt, S. Advances in variational inference. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *41*, 2008–2026.
37. Gordon, A.D.; Henzinger, T.A.; Nori, A.V.; Rajamani, S.K. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*; ACM: New York, NY, USA, 2014; pp. 167–181.
38. Ghahramani, Z. Probabilistic machine learning and artificial intelligence. *Nature* **2015**, *521*, 452.
39. Bishop, C.M. Latent variable models. In *Learning in Graphical Models*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 371–403.
40. Blei, D.M.; Ng, A.Y.; Jordan, M.I. Latent Dirichlet allocation. *J. Mach. Learn. Res.* **2003**, *3*, 993–1022.
41. Tipping, M.E.; Bishop, C.M. Probabilistic principal component analysis. *J. R. Stat. Soc. Ser. B (Stat. Methodol.)* **1999**, *61*, 611–622.
42. Masegosa, A.; Nielsen, T.D.; Langseth, H.; Ramos-Lopez, D.; Salmerón, A.; Madsen, A.L. Bayesian Models of Data Streams with Hierarchical Power Priors. *arXiv* **2017**, arXiv:1707.02293.
43. Masegosa, A.; Ramos-López, D.; Salmerón, A.; Langseth, H.; Nielsen, T. Variational inference over nonstationary data streams for exponential family models. *Mathematics* **2020**, *8*, 1942.
44. Kucukelbir, A.; Tran, D.; Ranganath, R.; Gelman, A.; Blei, D.M. Automatic differentiation variational inference. *J. Mach. Learn. Res.* **2017**, *18*, 430–474.
45. Pritchard, J.K.; Stephens, M.; Donnelly, P. Inference of population structure using multilocus genotype data. *Genetics* **2000**, *155*, 945–959.
46. Kipf, T.N.; Welling, M. Variational graph auto-encoders. *arXiv* **2016**, arXiv:1611.07308.
47. Schölkopf, B.; Smola, A.; Müller, K.R. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput.* **1998**, *10*, 1299–1319.
48. Fisher, R.A. The use of multiple measurements in taxonomic problems. *Ann. Eugen.* **1936**, *7*, 179–188.
49. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324.
50. Amari, S.I. Natural gradient works efficiently in learning. *Neural Comput.* **1998**, *10*, 251–276.
51. Robbins, H.; Monro, S. A stochastic approximation method. *Ann. Math. Stat.* **1951**, *22*, 400–407.
52. Li, M.; Zhang, T.; Chen, Y.; Smola, A.J. Efficient mini-batch training for stochastic optimization. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; ACM: New York, NY, USA, 2014; pp. 661–670.
53. Masegosa, A.R.; Martinez, A.M.; Langseth, H.; Nielsen, T.D.; Salmerón, A.; Ramos-López, D.; Madsen, A.L. Scaling up Bayesian variational inference using distributed computing clusters. *Int. J. Approx. Reason.* **2017**, *88*, 435–451.
54. Hopfield, J.J. Artificial neural networks. *IEEE Circuits Devices Mag.* **1988**, *4*, 3–10.
55. Hahnloser, R.H.; Sarpeshkar, R.; Mahowald, M.A.; Douglas, R.J.; Seung, H.S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* **2000**, *405*, 947–951.
56. Glorot, X.; Bordes, A.; Bengio, Y. Deep sparse rectifier neural networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence And Statistics, Fort Lauderdale, FL, USA, 11–13 April 2011; pp. 315–323.
57. Chen, T.; Li, M.; Li, Y.; Lin, M.; Wang, N.; Wang, M.; Xiao, T.; Xu, B.; Zhang, C.; Zhang, Z. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv* **2015**, arXiv:1512.01274.
58. Griewank, A. On automatic differentiation. *Math. Program. Recent Dev. Appl.* **1989**, *6*, 83–107.
59. Doersch, C. Tutorial on variational autoencoders. *arXiv* **2016**, arXiv:1606.05908.
60. Pless, R.; Souvenir, R. A survey of manifold learning for images. *IPSJ Trans. Comput. Vis. Appl.* **2009**, *1*, 83–94.
61. Kulkarni, T.D.; Whitney, W.F.; Kohli, P.; Tenenbaum, J. Deep convolutional inverse graphics network. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, USA, 7–12 December 2015; pp. 2539–2547.
62. Gregor, K.; Danihelka, I.; Graves, A.; Rezende, D.J.; Wierstra, D. Draw: A recurrent neural network for image generation. *arXiv* **2015**, arXiv:1502.04623.
63. Sohn, K.; Lee, H.; Yan, X. Learning structured output representation using deep conditional generative models. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, USA, 7–12 December 2015; pp. 3483–3491.
64. Pu, Y.; Gan, Z.; Henao, R.; Yuan, X.; Li, C.; Stevens, A.; Carin, L. Variational autoencoder for deep learning of images, labels and captions. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 2352–2360.
65. Semeniuta, S.; Severyn, A.; Barth, E. A hybrid convolutional variational autoencoder for text generation. *arXiv* **2017**, arXiv:1702.02390.
66. Hsu, W.N.; Zhang, Y.; Glass, J. Learning latent representations for speech generation and transformation. *arXiv* **2017**, arXiv:1704.04222.
67. Gómez-Bombarelli, R.; Wei, J.N.; Duvenaud, D.; Hernández-Lobato, J.M.; Sánchez-Lengeling, B.; Sheberla, D.; Aguilera-Iparraguirre, J.; Hirzel, T.D.; Adams, R.P.; Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Cent. Sci.* **2018**, *4*, 268–276.
68. Johnson, M.; Duvenaud, D.K.; Wiltschko, A.; Adams, R.P.; Datta, S.R. Composing graphical models with neural networks for structured representations and fast inference. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 2946–2954.

69. Linderman, S.W.; Miller, A.C.; Adams, R.P.; Blei, D.M.; Paninski, L.; Johnson, M.J. Recurrent switching linear dynamical systems. *arXiv* **2016**, arXiv:1610.08466.
70. Zhou, M.; Cong, Y.; Chen, B. The Poisson Gamma belief network. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, USA, 7–12 December 2015; pp. 3043–3051.
71. Card, D.; Tan, C.; Smith, N.A. A Neural Framework for Generalized Topic Models. *arXiv* **2017**, arXiv:1705.09296.
72. Chung, J.; Kastner, K.; Dinh, L.; Goel, K.; Courville, A.C.; Bengio, Y. A recurrent latent variable model for sequential data. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, USA, 7–12 December 2015; pp. 2980–2988.
73. Jiang, Z.; Zheng, Y.; Tan, H.; Tang, B.; Zhou, H. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv* **2016**, arXiv:1611.05148.
74. Xie, J.; Girshick, R.; Farhadi, A. Unsupervised deep embedding for clustering analysis. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 478–487.
75. Louizos, C.; Shalit, U.; Mooij, J.M.; Sontag, D.; Zemel, R.; Welling, M. Causal effect inference with deep latent-variable models. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 6446–6456.
76. Ou, Z. A Review of Learning with Deep Generative Models from Perspective of Graphical Modeling. *arXiv* **2018**, arXiv:1808.01630.
77. Schulman, J.; Heess, N.; Weber, T.; Abbeel, P. Gradient estimation using stochastic computation graphs. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, USA, 7–12 December 2015; pp. 3528–3536.
78. Dillon, J.V.; Langmore, I.; Tran, D.; Brevdo, E.; Vasudevan, S.; Moore, D.; Patton, B.; Alemi, A.; Hoffman, M.; Saurous, R.A. TensorFlow Distributions. *arXiv* **2017**, arXiv:1711.10604.
79. Wingate, D.; Weber, T. Automated variational inference in probabilistic programming. *arXiv* **2013**, arXiv:1301.1299.
80. Mnih, A.; Gregor, K. Neural variational inference and learning in belief networks. *arXiv* **2014**, arXiv:1402.0030.
81. Dayan, P.; Hinton, G.E.; Neal, R.M.; Zemel, R.S. The Helmholtz machine. *Neural Comput.* **1995**, *7*, 889–904.
82. Gershman, S.; Goodman, N. Amortized inference in probabilistic reasoning. In Proceedings of the Annual Meeting of the Cognitive Science Society, Quebec City, QC, Canada, 23–26 July 2014; Volume 36, pp. 517–522.
83. Glasserman, P. *Monte Carlo Methods in Financial Engineering*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 53.
84. Fu, M.C. Gradient estimation. *Handbooks Oper. Res. Manag. Sci.* **2006**, *13*, 575–616.
85. Rezende, D.J.; Mohamed, S.; Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv* **2014**, arXiv:1401.4082.
86. Titsias, M.; Lázaro-Gredilla, M. Doubly stochastic variational Bayes for non-conjugate inference. In Proceedings of the International Conference on Machine Learning, Beijing, China, 21–26 June 2014; pp. 1971–1979.
87. Figurnov, M.; Mohamed, S.; Mnih, A. Implicit Reparameterization Gradients. *arXiv* **2018**, arXiv:1805.08498.
88. Tucker, G.; Mnih, A.; Maddison, C.J.; Lawson, J.; Sohl-Dickstein, J. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 2627–2636.
89. Grathwohl, W.; Choi, D.; Wu, Y.; Roeder, G.; Duvenaud, D. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *arXiv* **2017**, arXiv:1711.00123.
90. Glynn, P.W. Likelihood ratio gradient estimation for stochastic systems. *Commun. ACM* **1990**, *33*, 75–84.
91. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256.
92. Ruiz, F.; Titsias, M.; Blei, D. The generalized reparameterization gradient. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 460–468.
93. Mnih, A.; Rezende, D.J. Variational inference for Monte Carlo objectives. *arXiv* **2016**, arXiv:1602.06725.
94. Foerster, J.; Farquhar, G.; Al-Shedivat, M.; Rocktäschel, T.; Xing, E.P.; Whiteson, S. DiCE: The Infinitely Differentiable Monte-Carlo Estimator. *arXiv* **2018**, arXiv:1802.05098.
95. Hinton, G.E.; Salakhutdinov, R.R. Reducing the dimensionality of data with neural networks. *Science* **2006**, *313*, 504–507.
96. Salvatier, J.; Wiecki, T.V.; Fonnesbeck, C. Probabilistic programming in Python using PyMC3. *PeerJ Comput. Sci.* **2016**, *2*, e55.
97. Carpenter, B.; Gelman, A.; Hoffman, M.; Lee, D.; Goodrich, B.; Betancourt, M.; Brubaker, M.A.; Guo, J.; Li, P.; Riddell, A. Stan: A probabilistic programming language. *J. Stat. Softw.* **2016**, *20*, 1–37.
98. Ge, H.; Xu, K.; Ghahramani, Z. Turing: A Language for Flexible Probabilistic Inference. In Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, Playa Blanca, Lanzarote, Canary Islands, 9–11 April 2018; Storkey, A., Perez-Cruz, F., Eds.; Proceedings of Machine Learning Research; PMLR: Playa Blanca, Lanzarote, Spain, 2018; Volume 84, pp. 1682–1690.
99. Ketkar, N. Introduction to keras. In *Deep Learning with Python*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 97–111.
100. Bergstra, J.; Breuleux, O.; Bastien, F.; Lamblin, P.; Pascanu, R.; Desjardins, G.; Turian, J.; Warde-Farley, D.; Bengio, Y. Theano: A CPU and GPU math expression compiler. In Proceedings of the Python for Scientific Computing Conference (SciPy), Austin, TX, USA, 28 June–3 July 2010; Volume 4, pp. 3–10.
101. Baudart, G.; Burroni, J.; Hirzel, M.; Kate, K.; Mandel, L.; Shinnar, A. Extending Stan for deep probabilistic programming. *arXiv* **2020**, arXiv:1810.00873.

102. Murray, L.M.; Schön, T.B. Automated learning with a probabilistic programming language: Birch. *Annu. Rev. Control* **2018**, *46*, 29–43.
103. Tehrani, N.; Arora, N.S.; Li, Y.L.; Shah, K.D.; Noursi, D.; Tingley, M.; Torabi, N.; Masouleh, S.; Lippert, E.; Meijer, E.; et al. Bean machine: A declarative probabilistic programming language for efficient programmable inference. In Proceedings of the 10th International Conference on Probabilistic Graphical Models, Aalborg, Denmark, 23–25 September 2020.
104. Minka, T.; Winn, J.; Guiver, J.; Webster, S.; Zaykov, Y.; Yangel, B.; Spengler, A.; Bronskill, J. Infer.NET. 2014. Available online: https://research.microsoft.com/infernet (accessed on 15th January 2021).
105. Maddison, C.J.; Mnih, A.; Teh, Y.W. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *arXiv* **2016**, arXiv:1611.00712.