Predictive Uncertainty Quantification with Compound Density Networks

Agustinus Kristiadi

Matriculation number: 3047084

February 2019

Master Thesis

Computer Science

Supervisors:

Jun.-Prof. Dr. Asja Fischer
Prof. Dr. Jens Lehmann

Institut für Informatik

Rheinische Friedrich-Wilhelms-Universität Bonn

Abstract

Despite the huge success of deep neural networks (NNs), finding good mechanisms for quantifying their prediction uncertainty is still an open problem. Bayesian neural networks are one of the most popular approaches to uncertainty quantification. On the other hand, it was recently shown that ensembles of NNs, which belong to the class of mixture models, can be used to quantify prediction uncertainty. In this thesis, we build upon these two approaches. First, we increase the mixture model's flexibility by replacing the fixed mixing weights by an adaptive, input-dependent distribution, parametrized by NNs, and by considering uncountably many mixture components. The resulting class of models can be seen as the continuous counterpart to mixture density networks and is therefore referred to as *compound density networks* (CDNs). We employ both maximum likelihood and variational Bayesian inference to train CDNs, and empirically show that they can (i) quantify both aleatoric and epistemic uncertainties, (ii) yield better uncertainty estimates on out-of-distribution data, and (iii) are more robust to adversarial examples than the previous recent approaches.

Acknowledgement

The author would like to thank, first and foremost, Jun.-Prof. Dr. Asja Fischer for the supervision. Comments, criticisms, and suggestions, which are integral to the improvement of the original idea of this thesis, came from the fruitful discussions between the author and Jun.-Prof. Dr. Asja Fischer. Moreover, without her as the co-author, the paper version of this thesis would not be realized.

The author would also like to thank Prof. Dr. Jens Lehmann and Smart Data Analytics (SDA) lab in Bonn for hosting the author while this thesis was being written. Furthermore, the author is grateful for the generous support in computing power endowed by SDA, which was crucial for running the experiments presented in this thesis.

Not to forget, the author would like to thank all the anonymous reviewers of the paper version of this thesis. The critical and objective feedbacks given by them are important to further improve the quality of this thesis.

Lastly, the author would like to thank the family of the University of Bonn, the Bonn University and State Library, and all the maintainers of the civilization in Bonn, who altogether provided a suitable environment where the author could work on this thesis with peace of mind.

Table of contents

List of Figures

List of Algorithms

List of Symbols

$\mathbf{0}$	Zero vector
$\mathbf{b}_l$	The l-th bias vector of a neural network
$\mathbf{H}$	Hessian matrix
$\mathbf{I}$	Identity matrix
$\mathbf{I}_m$	Identity matrix of size $m \times m$
$\mathbf{W}_l$	The l-th weight matrix of a neural network
$\mathcal{D}$	Dataset
λ, η	Scalar hyperparameter
μ, $\boldsymbol{\mu}$	Mean of a Gaussian
σ^2, $\boldsymbol{\Sigma}$	Variance/covariance a Gaussian
$\mathbf{h}^{(t)}$	The hidden state at time t of a recurrent network
$\mathbf{h}_l$	The l-th hidden units of a neural network
$\boldsymbol{\omega}$	Variational parameter
$\boldsymbol{\theta}$, $\boldsymbol{\phi}$, $\boldsymbol{\psi}$	Parameter of a model
$\boldsymbol{\theta}^*$	The optimal value of $\boldsymbol{\theta}$
$\mathbf{x}$	Input variable
$\mathbf{y}$	Output variable
z, $\mathbf{z}$	Latent variable

Probability distributions

$\mathcal{MN}$	Matrix-variate Gaussian distribution

$\mathcal{N}$	Gaussian distribution
p	Abstract probability distribution
q	Variational posterior
Bern	Bernoulli distribution

Functions

$\mathcal{L}$	Objective function
σ	Sigmoid function
f, g	Neural network
f_{CNN}	Convolutional neural network
f_{MLP}	Multi-layer perceptron
f_{RNN}	Recurrent neural network
h	Component-wise nonlinear function, acting on a vector/matrix

Operators

$*$	Convolution
dim	Dimension
$\mathbb{E}$	Expectation
D_{KL}	KL-divergence
$\lvert . \rvert$	Determinant
$\lVert . \rVert_F$	Frobenius norm
$\nabla_{\boldsymbol{\theta}}$	Gradient w.r.t. $\boldsymbol{\theta}$
$\otimes$	Kronecker product
diag	Mapping a vector to a diagonal matrices, where the diagonal entries are given by the vector
tr	Matrix trace
vec	Vectorization operator, i.e. stacking the columns of a matrix

Chapter 1

Introduction

Deep neural networks (NNs) have achieved state-of-the-art performance in many application areas, such as computer vision [Krizhevsky et al., 2012] and natural language processing [Collobert et al., 2011]. However, despite achieving impressive prediction accuracy on these supervised machine learning tasks, NNs do not provide good ways of quantifying predictive uncertainty. This is undesirable for many mission-critical applications, where taking wrong predictions with high confidence could have fatal consequences. Consider an NN that is used as an object detector and is applied in an autonomous vehicle, where it is trained to recognize various objects present in a given moment during the vehicle's self-driving. Imagine that the NN misclassify an object with class "human" as "road": the vehicle will then perceive that the road is empty thus it will continue cruising or speed up. Without any doubt, this behavior will be fatal, thus the need of the NN to quantify its prediction uncertainty and reports back to the human behind the steering wheel or fall-backs to the safest possible action, whenever it is unsure.

With the increasing real-world applications of machine learning systems, such as in autonomous vehicles, medical diagnostics, robot control in factories, or power grid systems, concerns about the safeness of these systems for human and the environment have been raised by the community. Applications mentioned above could potentially be life-threatening for humans or damaging the environment. Thus, increasingly there has been a lot of attention in the direction of making sure that machine learning systems to be safe, under the umbrella of a field called *AI safety* [Amodei et al., 2016].

One of the concrete problems in AI safety is how to handle *distributional change* or *out-of-distribution* (OOD) data [Amodei et al., 2016]. Consider a cleaning robot trained to clean a factory floor. The robot might learn that using a very strong industrial-grade cleaning solution to be the best way to clean the floor. However, if the robot is then deployed in an office or residential house, the behavior that it learned might be very dangerous for the people reside in the office or the house. This example illustrates the need for handling out-of-distribution data. At the very least, the robot should know when it is unsure whether it should still use a strong cleaning solution in a new environment, so that a human supervisor can take over,

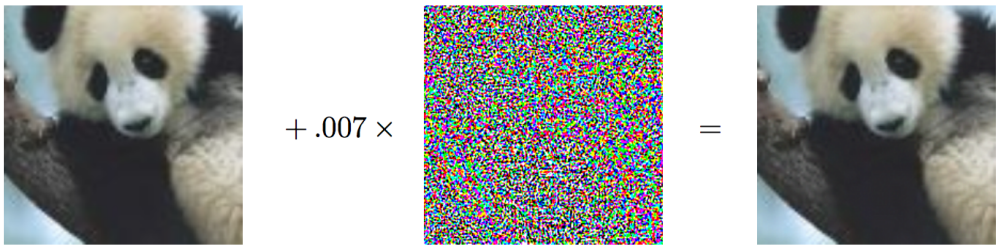


Figure 1.1: An illustration of an adversarial example. The clean image (left image), which has a label of "panda", is perturbed by an imperceptible noise (middle image), resulting in a new image (right image) which has virtually no difference to the clean image. However, when this resulting image, called the adversarial example, is feed into an NN, the NN would predict it as "gibbon" with 99% confidence. Taken from Goodfellow et al. [2015].

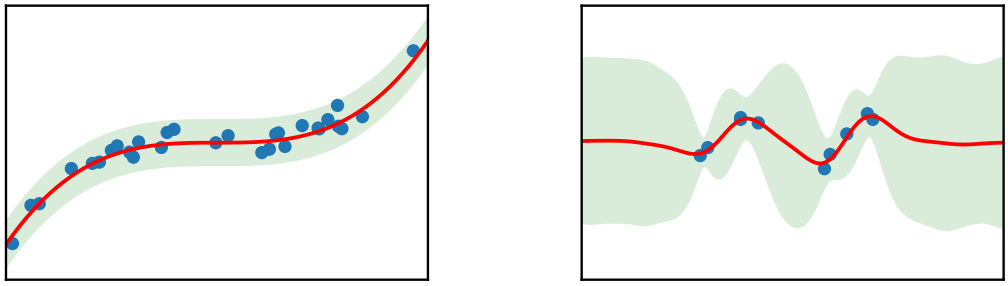
for instance.

Another concrete problem that has gained research traction recently is *adversarial examples* [Szegedy et al., 2014]. Szegedy et al. [2014] found that adding small corrupting noise to the input of an NN could make the NN misclassifies the input with high confidence, even if the NN has been validated to be very accurate on clean input (Figure 4.7). Adversarial examples are major problems in machine learning systems, as hackers can easily construct fake inputs that make mission-critical system at best unusable and at worst life-threatening for humans. At the very least it is desirable to have an NN system that can report high uncertainty under adversarial examples, so that, as in OOD data problem, it can notify the human supervisor. The researches on constructing new ways to "fool" neural networks, called *adversarial attacks* and on constructing new ways to defend against them, called *adversarial defenses*, have been a major focus of the community in the recent years [Goodfellow et al., 2015; Carlini and Wagner, 2017, etc]. However, even if a new defense mechanism is able to defend against previous attacks, a new kind of attack quickly render this defense obsolete [Shafahi et al., 2019]. Therefore, the field of adversarial attack and defense is still wide open and has a big impact on AI safety.

There are two classes of uncertainty [Der Kiureghian and Ditlevsen, 2009], which also present in neural network-based probabilistic models:

- *Aleatoric uncertainty*, which is caused by imprecision of measurements of data. This includes the noise in our observation in dataset $\mathcal{D}$, e.g. imprecision when assigning class label $\mathbf{y}$ given input $\mathbf{x}$ during data gathering. Thus, this type of uncertainty is also referred to as *data uncertainty*. An illustration of aleatoric uncertainty is shown in Figure 1.2a.

- *Epistemic uncertainty*, which includes the uncertainty around the architectural choice of an NN, the choice of distribution $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$, and the parameters $\boldsymbol{\theta}$ that we use. This type of uncertainty therefore includes the *model uncertainty* and the *parameter uncertainty*. Epistemic uncertainty has the properties that



(a) Aleatoric uncertainty (b) Epistemic uncertainty

Figure 1.2: Illustrations of aleatoric and epistemic uncertainty. Blue dots are the data points, red lines are the predictions, and the green shade is the ±3 standard deviation around the prediction. Aleatoric uncertainty captures the noise in the dataset and is thus constant in the case of a dataset with homoscedastic noise, pictured above. Meanwhile, epistemic uncertainty captures the uncertainty of the model and thus decreases when more data points are observed.

it can be reduced by adding more observations to the dataset, as shown in Figure 1.2b.

Both aleatoric and epistemic uncertainty induce the *predictive uncertainty* of a model [Gal, 2016], i.e. given an input $\mathbf{x}$, how confident is our model predicting the output to be $\mathbf{y}$.

A principled and the most explored way to quantify the uncertainty in NNs is through Bayesian inference. In the so-called Bayesian neural networks (BNNs) [Neal, 1995], the NN parameters are treated as random variables and the goal of learning is to infer the posterior probability distribution of the parameters given the training data. Since exact Bayesian inference in NNs is computationally intractable, different approximation techniques have been proposed [Neal, 1995; Blundell et al., 2015; Hernández-Lobato and Adams, 2015; Ritter et al., 2018, etc.]. Given the approximate posterior, the final predictive distribution is obtained as the expected predictive distribution under the posterior. This expectation can be seen as an ensemble of an uncountably infinite number of predictors, where the prediction of each model is weighted by the posterior probability of the corresponding parameters.

Based on a Bayesian interpretation of dropout [Srivastava et al., 2014], Gal and Ghahramani [2016] proposed to apply it not only during training but also when making predictions to estimate predictive uncertainty. Interestingly, dropout has been also interpreted as ensemble model [Srivastava et al., 2014] where the predictions are averaged over the different NNs resulting from different dropout-masks. Inspired by this, Osband et al. [2016] and Lakshminarayanan et al. [2017] proposed to use a simple NN ensemble to quantify the prediction uncertainty, i.e. to train a set of independent NNs and defining the final prediction as the arithmetic mean of the outputs of the individual models, which corresponds to defining a uniformly-

weighted mixture model. It is argued, that the model is able to encode two sources of uncertainty by calibrating the target uncertainty, i.e. uncertainty in target $\mathbf{y}$ given input $\mathbf{x}$ in each component and capturing the model uncertainty by averaging over the components.

In this thesis, we aim at further investigating the potential that lies in employing mixture distributions for uncertainty quantification. The flexibility of the mixture model can be increased by learning input-conditioned mixture weights as it is done by mixture density networks (MDNs) [Bishop, 1994]. Furthermore, one can consider uncountably many component distributions instead of finitely many of them, which turns the mixture distribution into a compound distribution. We combine both by deriving the continuous counterpart of MDNs, which we call *compound density networks* (CDNs). As with MDNs, these networks can be trained by regularized maximum likelihood estimation. Moreover, variational Bayes can be employed to infer the posterior distribution over the CDN parameters, leading to a combination of the mixture model and the Bayesian approach for predictive uncertainty quantification. We experimentally show that CDNs allow for better uncertainty quantification and are more robust to adversarial examples than previous approaches.

This thesis is structured as follows. Chapter 2 will be used as a review of the background knowledge that is useful for this thesis. Specifically, we will review neural networks as probabilistic models, the principled approaches of uncertainty quantification, as well as a brief overview of mixture models and their variants. We will introduce our proposed approach, the compound density networks in Chapter 3. Extensive experimental results will be presented in Chapter 4. Finally, we conclude this thesis in Chapter 5, along with discussion regarding future research in this direction.

Chapter 2

Background

In this chapter, we will review the background knowledge necessary for the derivation of our proposed models. We will begin our discussion with a brief review of neural networks as probabilistic models and their common variants in Section 2.1. We will then discuss the formal methods of uncertainty quantification in probabilistic models, in Section 2.2. Specifically, we will review Bayesian statistics in Section 2.2.1 and frequentist statistics in Section 2.2.2. Finally, we will review the mixture models approaches, along with their variants, such as the mixture of experts and mixture density networks, in Section 2.3

2.1 Neural networks as probabilistic models

Let $\mathcal{D} := \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$ with $\mathbf{x}_n \in \mathbb{R}^p, \mathbf{y}_n \in \mathbb{R}^q \; \forall n = 1, \ldots, N$ be a dataset consisting of N independent and identically distributed (i.i.d.) data points. The goal of supervised learning, which we focus on in this thesis, is to infer the unknown conditional probability distribution $p(\mathbf{y}|\mathbf{x})$ that describes the relationship between each output $\mathbf{y}_n$ and input $\mathbf{x}_n$ in D. Typically, to approach this problem we assume a probabilistic model, given by a parametric distribution in the form of $p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})$ and make statistical estimation on the parameters to find the one that maximize the probability of dataset $\mathcal{D}$.

Examples of the family of supervised learning problems are regression and classification. In regression, we assume that the target vector $\mathbf{y}$ is a continuous random variable, for instance in the stock price prediction problem, where $\mathbf{y}$ denotes the stock price and $\mathbf{x}$ contains variables that the stock price depends upon, such as the revenue of a particular company. Meanwhile in classification, we assume that $\mathbf{y}$ is a discrete random variable, i.e. aside of the condition that $\mathbf{y} \in \mathbb{R}^q$, we have that $y_i \in \{0, 1\}$ and $\sum_{i=1}^q y_i = 1$. An instance of classification problem is predicting the class or label $\mathbf{y}$ of objects (e.g. cat, dog, etc) in an image $\mathbf{x}$.

In regression problems, we assume that our model $p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})$ is a continuous probability distribution, while we assume that our model is a discrete probability distribution in classification problems. A popular choice of a family of distribution for $p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})$ is Gaussian distribution for regression and Categorical distribution for

classification. Given the choice and parametrization of the model, we can estimate the parameters $\boldsymbol{\theta}$ that fit $\mathcal{D}$ the best via *maximum likelihood estimation* (MLE). That is, our goal is to find a particular parameter vector $\boldsymbol{\theta}^*$ that maximize the likelihood function

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{x}_n;\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log p(\mathbf{y}_n|\mathbf{x}_n;\boldsymbol{\theta}) \ . \end{aligned} \tag{2.1}$$

Commonly we put an additional term in eq. 2.1 to state our prior knowledge about the parameters $\boldsymbol{\theta}$ in the form of prior distribution $p(\boldsymbol{\theta})$, which is useful to avoid overfitting

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D}) \\ &= \arg\max_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log p(\mathbf{y}_n|\mathbf{x}_n;\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \ . \end{aligned} \tag{2.2}$$

We call eq. 2.2 the *maximum a posteriori estimation* (MAP). The optimization problems above are most commonly be done using stochastic gradient-based optimization methods [Duchi et al., 2011; Hinton et al., 2012; Kingma and Ba, 2015].

Our focus in this thesis is to parametrize the probabilistic model $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ using NNs. That is, let $f(\mathbf{x};\boldsymbol{\theta})$ be a neural network, the probabilistic model is now defined as $p(\mathbf{y};f(\mathbf{x};\boldsymbol{\theta}))$. In the following sections, three commonly used NN types: multi-layer perceptron, convolutional neural networks, and recurrent neural networks, will be discussed.

2.1.1 Multi-layer perceptrons

Multi-layer perceptrons (MLPs) are defined as composition of $L \in \mathbb{N}$ functions

$$f_{\text{MLP}}(\mathbf{x};\boldsymbol{\theta}) := (f_L \circ f_{L-1} \circ \ldots \circ f_1)(\mathbf{x}) \ ,$$

where each f_l is a nonlinear function parametrized by $\boldsymbol{\theta}_l := \{\mathbf{W}_l, \mathbf{b}_l\}$, the l-th weight matrix and bias vector. The overall parameter of MLPs is thus given by $\boldsymbol{\theta} := \{\boldsymbol{\theta}_l\}_{l=1}^{L}$. Formally, for all l, we define

$$\begin{aligned} f_l : \mathbb{R}^{k_{l-1}} &\longrightarrow \mathbb{R}^{k_l} \\ \mathbf{h}_{l-1} &\longmapsto h(\mathbf{W}_l^{\mathrm{T}}\mathbf{h}_{l-1} + \mathbf{b}_l) =: \mathbf{h}_l \end{aligned} \tag{2.3}$$

where $\mathbf{h}_0 := \mathbf{x}$, $k_l \in \mathbb{N}$ to be the dimension of $\mathbf{h}_l$, and h to be an arbitrary component-wise nonlinear function such as sigmoid, hyperbolic tangent, or ReLU [Nair and Hinton, 2010]. Observe that the above definition implies that $\mathbf{W}_l \in \mathbb{R}^{k_{l-1} \times k_l}$ and $\mathbf{b}_l \in \mathbb{R}^{k_l}$. Moreover, $k_0 = \dim \mathbf{x} = p$ and $k_L = \dim \mathbf{y} = q$.

There are several variables that define the exact form of an MLP. First, the value of $L \in \mathbb{N}$, which corresponds to the number of nonlinear functions composing f. We call this number the *depth* of the NN. Second, the dimensionality of $\mathbf{h}_1, \ldots, \mathbf{h}_{L-1}$, which we call the *width* of each layer of the NN. Together with the depth, the width of an NN represents the *size* of the NN. The final variable needs to be defined is the choice of h, which commonly chosen as (component-wise) sigmoidal or rectifier functions. Altogether, they constitute the *architecture* of the MLP.

2.1.2 Convolutional neural networks

Convolutional neural networks (CNNs) [LeCun et al., 1989], as the name suggests, are NNs which use convolution operator in their computation. Specifically, CNNs differ from MLPs in the way each layer's weight matrix $\mathbf{W}_l$ interacts with the input $\mathbf{h}_{l-1}$, where instead of matrix-vector multiplication, convolution is employed. Formally, in analogue to eq. 2.3, we define

$$\begin{aligned} f_l : \mathbb{R}^{k_{l-1}} &\longrightarrow \mathbb{R}^{k_l} \\ \mathbf{h}_{l-1} &\longmapsto h(\mathbf{W}_l * \mathbf{h}_{l-1} + \mathbf{b}_l) =: \mathbf{h}_l \end{aligned} \tag{2.4}$$

where each $\mathbf{W}_l$ is usually a collection of small matrices (e.g. 128 3x3-matrices). Furthermore, we define CNNs to be

$$f_{\text{CNN}}(\mathbf{x}, \boldsymbol{\theta}) := (f_L \circ f_{L-1} \circ \ldots \circ f_1)(\mathbf{x}) \ .$$

By using the convolution operator, CNNs are able to capture local properties occurring in many places in the input. Moreover, as the weights are usually a collection of small matrices and shared over the spatial location of the input, CNNs have much fewer parameters than MLPs. Finally, typically it is the case that $\dim \mathbf{h}_{l-1} > \dim \mathbf{h}_l$, either via an explicit sub-sampling operator (e.g. pooling) or implicitly as a result of the convolution. Those three properties, called (i) local receptive fields, (ii) weight sharing, and (iii) sub-sampling, are the building blocks of CNNs [Bishop, 2006]. In practice, usually, CNNs are used in conjunction with MLPs [Krizhevsky et al., 2012; Simonyan and Zisserman, 2014]. That is, we usually use a CNN for the first L_0 layers, where $L_0 < L$, and use an MLP for the rest $L - L_0$ layers, with the output of the first L_0 layers is the input of the MLP.

2.1.3 Recurrent neural networks

Recurrent neural networks (RNNs) [Rumelhart et al., 1986] are neural networks constructed based on the idea of parameter sharing over each "time-step" of data.

This allows RNNs to handle variable-length sequential data, such as textual data. Let a data point $\mathbf{x} := \{\mathbf{x}^{(t)}\}_{t=1}^{T}$ be sequence of length T, where we call $\mathbf{x}^{(t)} \in \mathbb{R}^p$ to be the input at time-step t. At the simplest formulation, an RNN is defined as a function

$$\begin{aligned} f_{\text{RNN}} : \mathbb{R}^p \times \mathbb{R}^k &\longrightarrow \mathbb{R}^k \\ \mathbf{x}^{(t)}, \mathbf{h}^{(t-1)} &\longmapsto h(\mathbf{W}^{\mathrm{T}}\mathbf{x}^{(t)} + \mathbf{V}^{\mathrm{T}}\mathbf{h}^{(t-1)} + \mathbf{b}) =: \mathbf{h}^{(t)} \end{aligned} \tag{2.5}$$

and is applied recursively at each time-step t. Note that the RNN defined above is parametrized by two matrices $\mathbf{W} \in \mathbb{R}^{p\times k}$ and $\mathbf{V} \in \mathbb{R}^{k\times k}$ and a vector $\mathbf{b} \in \mathbb{R}^k$ which are shared over all time-step t. Meanwhile $\mathbf{h}^{(t)} \in \mathbb{R}^k$ is called the *hidden state at time-step* t. One can then use the hidden state of each time-step or only the last state $\mathbf{h}^{(T)}$ to compute other quantities, e.g. as inputs to MLPs to do supervised learning.

RNNs have been extensively studied and there exist more sophisticated variations of them, such as Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] and Gated Recurrent Unit (GRU) [Cho et al., 2014]. They differ from eq. 2.5 in the way that f_{RNN} incorporates the more complicated operations, such as gating mechanisms. We refer the reader to Lipton et al. [2015] for more details.

2.2 Method of uncertainty quantification

Notice that MLE (eq. 2.1) and MAP (eq. 2.2) objective for training probabilistic NNs lead to a single solution of the parameter $\boldsymbol{\theta}^*$, referred to as a *point estimate*. This corresponds to a delta distribution over the parameter $\boldsymbol{\theta}$, where

$$p(\boldsymbol{\theta}) = \begin{cases} 1, & \text{if } \boldsymbol{\theta} = \boldsymbol{\theta}^* \\ 0, & \text{otherwise}\,. \end{cases}$$

It can be shown that this distribution is the limit of a Gaussian distribution centered around $\boldsymbol{\theta}^*$ when the variance goes to zero. Thus, the uncertainty over the parameter is not quantified in point estimation which is typically used to train NNs, explaining why NNs tend to be overconfident in their prediction. To remedy this situation, we can employ a range of principled inference techniques over $\boldsymbol{\theta}$ that come from Bayesian and frequentist statistics, which we will discuss in the following sections.

2.2.1 Bayesian statistics

The core of Bayesian statistics is the assumption that the parameter of a probabilistic model is a random variable and we would like to use the posterior distribution to summarize everything we know about it [Murphy, 2012]. The goal of Bayesian statistics is therefore to do Bayesian inference: given observations $\mathcal{D}$, we would like to infer the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ of model's parameter, according to Bayes' rule

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}\,. \tag{2.6}$$

We can then integrate our prediction w.r.t. the posterior distribution to get the posterior predictive distribution

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) = \int p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) \, \mathrm{d}\boldsymbol{\theta} \ , \tag{2.7}$$

in which the uncertainty over the parameters encoded in the posterior is incorporated.

Notice the connection between Bayesian inference and MAP estimation (eq. 2.2). Although in MAP estimation we also work with the posterior distribution, it is a point estimate method, thus unlike Bayesian inference, we do not have information about the full posterior distribution and the parameter uncertainty is not quantified.

Although Bayesian inference has a huge advantage over point estimate methods, it is notoriously hard to be done. This is because of the denominator presents in the Bayes' rule (eq. 2.6).

$$p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta} \tag{2.8}$$

which contains intractable integral even in moderately-sized models. As an illustration, assume that $\boldsymbol{\theta} = \{0, 1\}^d$. This implies that the integration can be done as a summation over 2^d possible values of $\boldsymbol{\theta}$, which is already intractable for, let us say $d = 100$. This fact is made even worse knowing that the parameters of modern NNs are continuous variables, where d can be up to tens of millions. Fortunately, there exist approximation methods that make Bayesian inference tractable. We will discuss these methods, in the context of Bayesian neural networks, in the followings.

Let $\boldsymbol{\theta}$ be the parameter of a neural network. In Bayesian neural networks (BNNs), following the framework of Bayesian inference, our goal is to infer the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ of $\boldsymbol{\theta}$ given dataset $\mathcal{D}$. In neural networks, due to the highly nonlinear dependence of network function (e.g. f_{MLP}) on the parameters values and the high dimensionality of the parameters, exact Bayesian inference cannot be done in a tractable manner. In fact, even the log-posterior function, used in MAP estimate methods, is highly non-convex and complicated, thus point estimate of the NN parameters is already a hard problem, let alone Bayesian treatment [Bishop, 2006]. Fortunately, there exist several families of approximation techniques to approximate the posterior distribution of a BNN in a tractable fashion, such as Laplace approximation, sampling methods, and variational Bayesian inference.

Laplace approximation

Laplace approximation [MacKay, 1992] is a simple local (around a certain point) approximation of the exact posterior. Specifically, given the posterior mode (the MAP estimate), Laplace approximation fits a Gaussian around it with a covariance equals to the inverse of the Hessian matrix evaluated at that point, that is

$$p(\boldsymbol{\theta}|\mathcal{D}) := \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\theta}^*, \mathbf{H}^{-1}|_{\boldsymbol{\theta}^*}) \ , \tag{2.9}$$

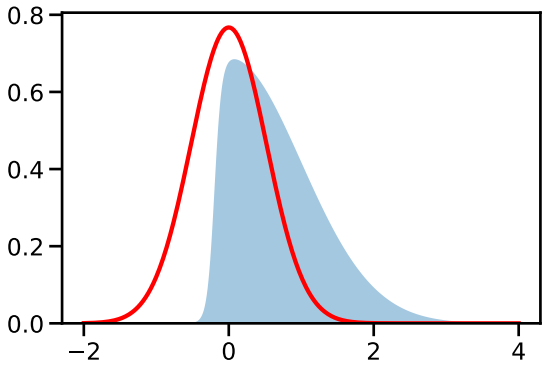


Figure 2.1: Illustration of the Laplace approximation applied to the distribution $p(z) \propto \exp(-z^2/2)\sigma(20z + 4)$ where $\sigma(z)$ is the sigmoid function. The normalized distribution $p(z)$ is shown in blue, together with the Laplace approximation centered on the mode $z^* = 0$ of $p(z)$ in red. Adapted from Bishop [2006].

where $\boldsymbol{\theta}^*$ is the MAP estimate and $\mathbf{H}$ is the Hessian matrix of the posterior with respect to $\boldsymbol{\theta}$. An illustration of Laplace approximation is shown in Figure 2.1.

Note that Laplace approximation is a local approximation (i.e. around a given point) and thus unable to capture the global properties of the exact posterior. Moreover, computing NNs' inverse Hessian is non-trivial as it has quadratic space and cubic time complexity. Fortunately, we can use a cheap approximation of the Hessian, for instance, Ritter et al. [2018] proposed to use Kronecker-factored Fisher information matrix or Gauss-Newton matrix as an approximation of the Hessian alongside matrix-variate normal approximate posterior [Gupta and Nagar, 1999], which makes the approximation highly efficient. Furthermore, they show competitive results compared to other approximate Bayesian inference methods.

Sampling methods

Approximate inference methods based on sampling methods, also known as *Monte Carlo* techniques, are methods of approximating some unknown quantities through numerical sampling [Bishop, 2006]. In BNNs specifically, sampling methods' goal is to draw independent samples from the exact posterior of the parameters, which are then used to compute some interesting quantities, such as the expectation $\mathbb{E}[g] = \int g(\boldsymbol{\theta})p(\boldsymbol{\theta}|D)\,\mathrm{d}\boldsymbol{\theta}$ of some function g (e.g. eq. 2.7), using Monte Carlo integration. Notice that as long as the methods sample from the exact posterior, the estimate will also be unbiased, unlike Laplace approximation. This property makes this class of methods powerful and attractive to practitioners.

One of the most popular subfamilies of sampling methods for BNNs is Markov chain Monte Carlo (MCMC) [Metropolis and Ulam, 1949], where Markov chain is constructed such that its equilibrium distribution coincides with the BNNs' posterior distribution. Hamiltonian Monte Carlo (HMC) [Neal, 1993], which uses Hamiltonian dynamics to reduce the correlation between successive Markov chain transition, is one of the most popular MCMC methods. However, HMC is not scalable to large

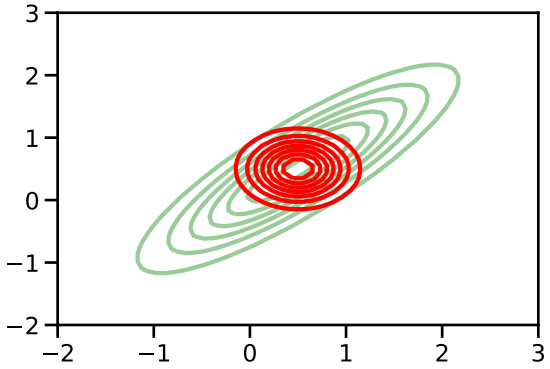


Figure 2.2: Illustration of the variational Bayesian inference. The exact posterior is shown in green, while the approximate/variational posterior is shown in red. Adapted from Bishop [2006].

scale datasets. Stochastic gradient Langevin dynamics (SGLD) [Welling and Teh, 2011] is therefore proposed as a combination of highly scalable stochastic optimization [Robbins and Monro, 1951] and Langevin dynamics [Neal et al., 2011]. Patterson and Teh [2013] further extends SGLD to incorporate preconditioner matrix to SGLD. Chen et al. [2014] further proposed stochastic gradient HMC, marrying the idea of SGLD and HMC, which enables HMC to be used in large scale datasets.

Variational Bayesian inference

Variational Bayesian inference [Peterson, 1987; Hinton and Van Camp, 1993], also called variational Bayes or VB, is a family of approximate Bayesian inference methods where a tractable model $q(\boldsymbol{\theta};\boldsymbol{\omega})$ parametrized by $\boldsymbol{\omega}$, called *variational posterior*, is used to approximate the unknown exact posterior. An illustration of VB is shown in Figure 2.2. Following the calculus of variations framework, $q(\boldsymbol{\theta};\boldsymbol{\omega})$ is found by maximizing the objective functional with respect to the *variational parameter* $\boldsymbol{\omega}$. This objective functional is derived from the KL-divergence between the variational posterior and the exact posterior, and can be shown to be the lower bound of the marginal likelihood $p(\mathcal{D})$. The objective functional is therefore called the *evidence lower bound* (ELBO) and is defined as

$$\mathcal{L}(\boldsymbol{\omega}) := \mathbb{E}_{q(\boldsymbol{\theta};\boldsymbol{\omega})}[\, p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})] - D_{\mathrm{KL}}[\, q(\boldsymbol{\theta};\boldsymbol{\omega}) \| \, p(\boldsymbol{\theta})] \ . \tag{2.10}$$

Clearly, unless $q(\boldsymbol{\theta};\boldsymbol{\omega}) = p(\boldsymbol{\theta}|\mathcal{D})$, the approximation found by VB is biased. However, the advantage of using VB is that unlike sampling methods, it is essentially an optimization method in which standard gradient based optimization methods can be employed, thus VB can be efficiently executed.

In the recent years, VB has become popular in Bayesian neural networks community. Graves [2011] proposed to use fully-factorized Gaussian variational posterior for BNNs and offered practical advices that are useful in implementing and training BNNs. Kingma and Welling [2014] proposed a *stochastic gradient variational Bayes*

estimator (SGVB) along with *reparametrization trick* to enable end-to-end training on generative models with VB. Blundell et al. [2015] further uses the reparametrization trick for BNNs along with fully-factorized Gaussian variational posterior and mixture of Gaussian prior. Different forms of variational posterior, such as matrix-variate Gaussian [Louizos and Welling, 2016; Sun et al., 2017] and compound distribution [Louizos and Welling, 2017], have also been used. Inspired by Goodfellow et al. [2014], the usage of implicit distributions, i.e. distributions which easy to sample from but intractable to evaluate, have also been used, such as by Krueger et al. [2017], Louizos and Welling [2017], and Pawlowski et al. [2017]. Dropout [Srivastava et al., 2014] which is originally introduced as regularization method, has recently been shown by Gal and Ghahramani [2016] to be approximating (in VB sense) the posterior of a deep Gaussian process [Damianou and Lawrence, 2013], giving rise to a method called *MC-dropout*. Furthermore, very recently, the connection between VB and natural gradient [Amari, 1998], which resulting in simple stochastic algorithms based on Adam [Kingma and Ba, 2015] and K-FAC [Martens and Grosse, 2015], has been proposed by Zhang et al. [2018] and Khan et al. [2018].

2.2.2 Frequentist statistics

Exactly the opposite to Bayesian statistics, the core of frequentist statistics is the assumption that parameters are fixed but unknown quantities, while data are random [Murphy, 2012]. Given the data samples, the goal of frequentist methods is, therefore, to estimate the exact values of the parameters through *estimators* such as MLE. Thus, unlike Bayesian methods, we cannot directly quantify the epistemic uncertainty of a model. Methods such as *confidence calibration* (or calibration for short), a frequentist notion of uncertainty which measure the discrepancy between subjective forecasts and long-run frequencies, can be used to measure the confidence of an estimator [DeGroot and Fienberg, 1983; Dawid, 1982]. Another simple way to estimate the properties (such as variance and confidence interval) of an estimator is through *bootstrap* [Efron, 1992]. Bootstrap works by resampling a given dataset without replacement to get several "fake" datasets, which are then used to estimate the parameters. The uncertainty induced by bootstrap's resampling process is then the uncertainty of the parameters estimates.

Recently, those frequentist methods have been applied for uncertainty quantification in NNs. Osband et al. [2016] uses bootstrap to train several NNs to provide uncertainty estimate of an agent's policy in reinforcement learning problems. Lakshminarayanan et al. [2017] proposed to use *proper scoring rules* [Gneiting and Raftery, 2007], i.e. measures of calibration quality e.g. negative log-likelihood (NLL), as the objective function of an NN. Additionally, similar to bootstrap, they train several NNs in parallel to quantify the uncertainty of the parameters estimate and hence the prediction uncertainty. However, unlike traditional bootstrap, they use the full dataset and rely on random initialization as the source of uncertainty. They show that the overall method, called *Deep Ensemble*, is competitive to or even better than BNNs in the estimation of predictive uncertainty. Guo et al. [2017] showed

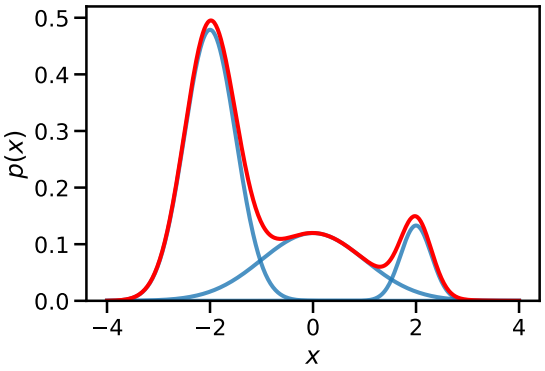


Figure 2.3: Illustration of a mixture distribution (red) over one-dimensional space. The mixture components (blue) are assumed to be Gaussian. Adapted from Bishop [2006].

that modern deep NNs are not well-calibrated due to several factors such as model complexity and the usage of batch normalization [Ioffe and Szegedy, 2015], and proposed to fix the problem using Platt scaling [Platt et al., 1999].

Notice that both of the ensemble approaches proposed by Lakshminarayanan et al. [2017]; Osband et al. [2016], which use the arithmetic mean of the predictive distribution of K neural networks, can be seen as mixture models: The K independent neural networks in the ensembles are parametrizing the K mixture components, while the corresponding mixing probabilities are uniform. We, therefore, shall discuss mixture models further in the following section.

2.3 Mixture models

Mixture models are the simplest form of latent variable models (LVMs) where the latent variable $z \in \{1, \ldots, K\}$ takes a value from a discrete latent space [Murphy, 2012]. Mixture models consist of K *mixture components* $p(\mathbf{x}|z)$ and a *mixing distribution* $p(z)$, and defined as

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(\mathbf{x}|z=k)p(z=k)\,, \tag{2.11}$$

where $\mathbf{x}$ is an arbitrary random variable we would like to model. Note that we can see z as a variable that indexes the mixture components.

The choice of the mixture components and the mixing distribution are arbitrary and induce different family of models. For instance, Gaussian mixture components paired with Categorical mixing distribution yields the *mixture of Gaussian* model [Dasgupta, 1999] (Figure 2.3), while Multinomial distributions and a Dirichlet distribution as the mixture components and the mixing distribution respectively is called the *latent Dirichlet allocation* [Blei et al., 2003]. Mixture models offer a flexible way of modeling data distribution as it can model multimodal distributions. For

instance, given a mixture of Gaussian model, it can capture K distinct modes that possibly exist in the true data distribution.

In the following sections, we will present some variants of the mixture model family that are specifically designed for modeling the conditional distribution of data, e.g. for classification and regression.

2.3.1 Conditional mixture models

As the name suggests, *conditional mixture models* are mixture models where the mixture components are conditional distributions $p(\mathbf{y}|\mathbf{x}, z)$. This induces mixture models in the form of

$$p(\mathbf{y}|\mathbf{x}) = \sum_{k=1}^{K} p(\mathbf{y}|\mathbf{x}, z = k)p(z = k)\,. \tag{2.12}$$

Figure 2.3 is still a valid illustration for conditional mixture models, provided that one change the x-axis to represent variable y and y-axis to represent $p(y|x)$.

The conditional mixture models are often used in regression and classification problems where the true conditional density we are modeling after is multimodal. For instance, in regression problems, it is common to use linear-Gaussian models as the mixture components, which results in *mixture of linear regression* [Bishop, 2006] model, described as

$$p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) = \sum_{k=1}^{K} \mathcal{N}(\mathbf{y}|\mathbf{W}_k^{\mathrm{T}}\mathbf{x}, \boldsymbol{\Sigma}_k)p(z = k)\,, \tag{2.13}$$

where $\mathbf{W}_k$ and $\boldsymbol{\Sigma}_k$ are the weight matrix and covariance matrix of the k-th mixture component, and $\boldsymbol{\theta} := \{\mathbf{W}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$ are the collection of all such matrices. For binary classification problems, one can use *mixture of logistic models* [Bishop, 2006] instead, as the counterpart of the mixture of linear regression. This model is defined as the mixture of K independent logistic regression models

$$p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) = \sum_{k=1}^{K} \mathrm{Bern}(\mathbf{y}|\sigma(\mathbf{W}_k^{\mathrm{T}}\mathbf{x}))p(z = k)\,, \tag{2.14}$$

where $\sigma(x) := 1/(1 + e^{-x})$ is the sigmoid function and $\boldsymbol{\theta} := \{\mathbf{W}_k\}_{k=1}^{K}$. The generalization of this model to a *mixture of softmax* model, to handle more than two possible values of label $\mathbf{y}$, is straightforward.

2.3.2 Mixture of experts

Observe that in the conditional mixture models, a single mixing distribution is being used to model all input data $\mathbf{x}$. This makes the resulting predictive distribution $p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})$ limited. *Mixture of experts* (MoEs) model [Jacobs et al., 1991] further

increase the capability of conditional mixture models by allowing the mixing distribution to be conditioned to the inputs, i.e. the mixing probability is a function of the input variables. Formally, MoEs is defined as

$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = \sum_{k=1}^{K} p(\mathbf{y}|\mathbf{x}, z=k;\boldsymbol{\theta}) p(z=k|\mathbf{x};\boldsymbol{\theta})\,. \tag{2.15}$$

In MoEs, we also call the mixture components as *experts* while the mixing distribution is also known as *gating* function. To train MoEs, one can employ well-established methods for learning LVMs, such as expectation-maximization (EM) with iterative re-weighted least squares being employed in the M step [Jordan and Jacobs, 1994].

2.3.3 Mixture density networks

Mixture density networks (MDNs) have been proposed to further improve the flexibility of MoEs by using a single neural network to model both mixture components and the mixing distribution. This way, MDNs can benefit from the highly powerful and nonlinear properties of the neural network. Furthermore, MDNs are efficient as both the components and the mixing distribution are sharing the same hidden units and parameters of the NN.

The functional form of MDNs is, in general, identical to that of MoEs. However, as discussed above, we are now assuming that $\boldsymbol{\theta}$ is the parameter vector of a neural network f, mapping the input $\mathbf{x}$ to both the parameters of the mixing and the component distributions:

$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = \sum_{k=1}^{K} p(\mathbf{y}; f(\mathbf{x}, z=k;\boldsymbol{\theta})) p(z=k; f(\mathbf{x};\boldsymbol{\theta}))\,, \tag{2.16}$$

where we have assumed that the index of the mixture component z is part of the NN f.

As MDNs are based on highly nonlinear function, unlike MoEs, EM algorithm cannot be used to train them. Fortunately, standard gradient-based optimization methods can be employed to maximize the maximum likelihood objective of MDNs

$$\log p(\mathcal{D};\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \sum_{k=1}^{K} p(\mathbf{y}; f(\mathbf{x}, z=k;\boldsymbol{\theta})) p(z=k; f(\mathbf{x};\boldsymbol{\theta}))\,, \tag{2.17}$$

to find the optimal MDNs' NN parameter.

Chapter 3

Compound Density Networks

In Chapter 2 we have discussed a range of uncertainty quantification methods from different perspectives and reviewed mixture models along with their variants. We have observed that mixture models or ensembles interpretations found in the work of Osband et al. [2016], Deep Ensemble [Lakshminarayanan et al., 2017], and Dropout [Srivastava et al., 2014] are useful for quantifying predictive uncertainty of an NN. We have also observed that more sophisticated mixture models exist, e.g. MoEs (Section 2.3.2) and MDNs (Section 2.3.3). The natural question that we raise is then, can we build upon these more sophisticated mixture models, a novel class of models which works better than the previous approaches in estimating predictive uncertainty quantification?

To fulfill that goal, in this chapter, we will present the *compound density networks* (CDNs): a new class of models which can be seen as the generalization of MDNs (Section 3.1). Training CDNs can be done by employing maximum-likelihood estimation over CDNs' parameters (Section 3.1.1), or by employing Bayesian inference (Section 3.1.2). An instance of CDNs, the *probabilistic hypernetworks* will then be presented extensively in Section 3.2. Finally, in Section 3.3, we will conclude this chapter with the comparison of CDNs to some recent works that have a similar functional form to CDNs.

3.1 Compound density networks

Motivated by MDNs, we generalize it from a finite mixture model into a mixture of uncountable components. This can be done by letting the discrete univariate random variable z of MDNs (eq. 2.16) be a continuous multivariate random variable $\mathbf{z}$. The summation in eq. 2.16 then becomes an integral, i.e.

$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = \int p(\mathbf{y}; f(\mathbf{x},\mathbf{z};\boldsymbol{\theta}))p(\mathbf{z}; f(\mathbf{x};\boldsymbol{\theta}))\,\mathrm{d}\mathbf{z}\,. \tag{3.1}$$

Furthermore, we relax the MDNs assumption that the overall mixture distribution (i.e. both components and mixing distribution) is parametrized by a single NN f. We instead let f to only parametrize the component distribution, retaining the original

parameter $\boldsymbol{\theta}$, and introduce another NN g parametrized by $\boldsymbol{\phi}$, which parametrizes the mixing distribution. Therefore, we have

$$\begin{aligned} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta},\boldsymbol{\phi}) &= \int p(\mathbf{y};f(\mathbf{x},\mathbf{z};\boldsymbol{\theta}))p(\mathbf{z};g(\mathbf{x};\boldsymbol{\phi}))\,\mathrm{d}\mathbf{z} \\ &= \mathbb{E}_{p(\mathbf{z};g(\mathbf{x};\boldsymbol{\phi}))}[\,p(\mathbf{y};f(\mathbf{x},\mathbf{z};\boldsymbol{\theta}))]\,. \end{aligned} \tag{3.2}$$

We call models in this form *compound density networks* (CDNs), as they form compound distributions and we can see them as the continuous counterpart of MDNs.

The log-likelihood function of CDNs is given by

$$\log p(\mathcal{D};\boldsymbol{\theta},\boldsymbol{\phi}) = \sum_{n=1}^{N} \log \mathbb{E}_{p(\mathbf{z};g(\mathbf{x}_n;\boldsymbol{\phi}))}[\,p(\mathbf{y}_n;f(\mathbf{x}_n,\mathbf{z};\boldsymbol{\theta}))]\,, \tag{3.3}$$

and the training can be done by employing point estimate methods such as MLE and MAP, or by employing Bayesian inference (Section 2.2.1) over the parameters $\boldsymbol{\theta},\boldsymbol{\phi}$. We will present both technique in the following sections.

3.1.1 Maximum-likelihood CDNs

Algorithm 3.1 The training procedure of CDNs with $\mathcal{L}_{\mathrm{ML}}$.

Require:

Mini-batch size M, number of samples S of $\mathbf{z}$, regularization strength λ, and learning rate α.

while the stopping criterion is not satisfied **do**
  $\{\mathbf{x}_m,\mathbf{y}_m\}_{m=1}^{M} \sim \mathcal{D}$ ▷ Sample mini-batch from dataset
  for $m=1,\ldots,M;\ s=1,\ldots,S$ **do**
    $\mathbf{z}_{ms} \sim p(\mathbf{z};g(\mathbf{x}_m;\boldsymbol{\phi}))$ ▷ Use reparametrization trick
  end for
  $\mathbb{E}_{\mathcal{D}}\mathbb{E}_{\mathbf{z}}[p(\mathbf{y}|\mathbf{x},\mathbf{z};\boldsymbol{\theta})] = \frac{1}{M}\sum_{m=1}^{M}\log\frac{1}{S}\sum_{s=1}^{S}p(\mathbf{y}_m;f(\mathbf{x}_m,\mathbf{z}_{ms};\boldsymbol{\theta}))$
  $\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi}) = \mathbb{E}_{\mathcal{D}}\mathbb{E}_{\mathbf{z}}[p(\mathbf{y}|\mathbf{x},\mathbf{z};\boldsymbol{\theta})] - \lambda\sum_{m=1}^{M}D_{\mathrm{KL}}[\,p(\mathbf{z};g(\mathbf{x}_m;\boldsymbol{\phi}))\|p(\mathbf{z}))]$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi})$
  $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha\nabla_{\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi})$
end while

To train CDNs, as with MDNs, we can use MLE to estimate the parameters $\boldsymbol{\theta},\boldsymbol{\phi}$ by maximizing the log-likelihood function given by eq. 3.3. Notice that the log-likelihood function contains an intractable integral (see eq. 3.2) which is intractable in general. Therefore we have to resort to approximation. Fortunately we can take cues from a widely-used and efficient estimator called the *SGVB* [Kingma and Welling, 2014]: We use the so-called *reparametrization trick* to sample $\mathbf{z}$ from the mixing distribution $p(\mathbf{z};g(\mathbf{x}_n;\boldsymbol{\phi}))$ and use Monte Carlo integration w.r.t. these

samples to approximate the integral. The usage of the reparametrization trick is crucial as this technique allows gradients to flow through the sampling operations, thus allowing us to use standard gradient-based optimization algorithms. As the choice of $p(\mathbf{z}; g(\mathbf{x}_n; \boldsymbol{\phi}))$ is still abstract at this point, we will give further detail how the reparametrization trick can be done when we discuss a particular CDN model in Section 3.2.

Using only the log-likelihood function (eq. 3.3) as an optimization objective is known to lead to overfitting [Bishop, 2006], thus some forms of regularization are desired. The most straightforward way to do this is to use the standard weight decay over the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. Another form of regularization that can be done is to constrain the mixing distribution $p(\mathbf{z}; g(\mathbf{x}_n; \boldsymbol{\phi}))$ to be close to some $p(\mathbf{z})$, by means of KL-divergence. The regularized objective is thus given by

$$\mathcal{L}_{\mathrm{ML}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \log p(\mathcal{D}; \boldsymbol{\theta}, \boldsymbol{\phi}) - \lambda \sum_{n=1}^{N} D_{\mathrm{KL}}[\, p(\mathbf{z}; g(\mathbf{x}_n; \boldsymbol{\phi})) \| \, p(\mathbf{z})] \ , \tag{3.4}$$

where λ is the weighting coefficient to control the strength of the KL-divergence regularization over $\mathbf{z}$. The choice of $p(\mathbf{z})$ is dependent to the choice of the mixing distribution, as we would like to keep the KL-divergence term analytically computable. Thus, we will discuss this in detail when we discuss the concrete CDN model in Section 3.2. All in all, the overall training procedure of maximum-likelihood CDNs is summarized in Algorithm 3.1.

3.1.2 Bayesian CDNs

Algorithm 3.2 The training procedure of CDNs with $\mathcal{L}_{\mathrm{VB}}$.

Require:

Mini-batch size M, number of samples S used for Monte Carlo integration of eq. 3.3, and learning rate α.

while the stopping criterion is not satisfied **do**
  $\{\mathbf{x}_m, \mathbf{y}_m\}_{m=1}^{M} \sim \mathcal{D}$
  $\boldsymbol{\theta}, \boldsymbol{\phi} \sim q(\boldsymbol{\psi}; \boldsymbol{\omega})$
  for $m = 1, \ldots, M;\ s = 1, \ldots, S$ **do**
    $\mathbf{z}_{ms} \sim p(\mathbf{z}; g(\mathbf{x}_m; \boldsymbol{\phi}))$
  end for
  $\mathbb{E}_{\mathcal{D}}\mathbb{E}_{\mathbf{z}}[\log p(\mathbf{y}|\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})] = \frac{1}{M} \sum_{m=1}^{M} \log \frac{1}{S} \sum_{s=1}^{S} p(\mathbf{y}_m; f(\mathbf{x}_m, \mathbf{z}_{ms}; \boldsymbol{\theta}))$
  $\mathcal{L}(\boldsymbol{\omega}) = \mathbb{E}_{\mathcal{D}}\mathbb{E}_{\mathbf{z}}[\log p(\mathbf{y}|\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})] - D_{\mathrm{KL}}[\, q(\boldsymbol{\psi}; \boldsymbol{\omega}) \| \, p(\boldsymbol{\psi})]$
  $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha \nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega})$ ▷ Update variational parameters $\boldsymbol{\omega}$
end while

Instead of using the point estimate approach to train CDNs, we can instead employ Bayesian inference over the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to further add additional

sources of uncertainty on CDNs' predictive distribution, i.e. by also quantifying the parameters uncertainty. Formally, let $\boldsymbol{\psi} := \{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ for brevity. We treat it as a random variable distributed by some prior $p(\boldsymbol{\psi})$. By marginalizing it, we get the Bayesian CDNs model

$$\begin{aligned} p(\mathbf{y}|\mathbf{x}) &= \int p(\mathbf{y}|\mathbf{x};\boldsymbol{\psi})p(\boldsymbol{\psi})\,\mathrm{d}\boldsymbol{\psi} \\ &= \iint p(\mathbf{y};f(\mathbf{x},\mathbf{z};\boldsymbol{\theta}))p(\mathbf{z};g(\mathbf{x};\boldsymbol{\phi}))p(\boldsymbol{\psi})\,\mathrm{d}\mathbf{z}\,\mathrm{d}\boldsymbol{\psi}\,. \end{aligned} \tag{3.5}$$

Our goal, following the standard Bayesian statistics described in Section 2.2.1, is to infer the posterior distribution $p(\boldsymbol{\psi}|\mathcal{D})$ of CDNs' parameters $\boldsymbol{\psi}$. This in turn induces the posterior predictive distribution of Bayesian CDNs

$$p(\mathbf{y}|\mathbf{x},\mathcal{D}) = \iint p(\mathbf{y};f(\mathbf{x},\mathbf{z};\boldsymbol{\theta}))p(\mathbf{z};g(\mathbf{x};\boldsymbol{\phi}))p(\boldsymbol{\psi}|\mathcal{D})\,\mathrm{d}\mathbf{z}\,\mathrm{d}\boldsymbol{\psi}\,. \tag{3.6}$$

Using variational Bayesian inference (VB) (Section 2.2.1), by proposing an approximate posterior $q(\boldsymbol{\psi};\boldsymbol{\omega}) \approx p(\boldsymbol{\psi}|\mathcal{D})$, gives us the ELBO objective (eq. 2.10) of CDNs:

$$\begin{aligned} \mathcal{L}_{\mathrm{VB}}(\boldsymbol{\psi}) &= \mathbb{E}_{q(\boldsymbol{\psi};\boldsymbol{\omega})}[\log p(\mathbf{y}|\mathbf{x};\boldsymbol{\psi})] - D_{\mathrm{KL}}[\,q(\boldsymbol{\psi};\boldsymbol{\omega})\|\,p(\boldsymbol{\psi})] \\ &= \mathbb{E}_{q(\boldsymbol{\psi};\boldsymbol{\omega})}\left[\log \int p(\mathbf{y};f(\mathbf{x},\mathbf{z};\boldsymbol{\theta}))p(\mathbf{z};g(\mathbf{x};\boldsymbol{\phi}))\,\mathrm{d}\mathbf{z}\right] - D_{\mathrm{KL}}[\,q(\boldsymbol{\psi};\boldsymbol{\omega})\|\,p(\boldsymbol{\psi})]\,. \end{aligned} \tag{3.7}$$

The posterior predictive distribution of Bayesian CDNs can then be approximated w.r.t. $q(\boldsymbol{\psi};\boldsymbol{\omega})$:

$$p(\mathbf{y}|\mathbf{x},\mathcal{D};\boldsymbol{\omega}) = \iint p(\mathbf{y};f(\mathbf{x},\mathbf{z};\boldsymbol{\theta}))p(\mathbf{z};g(\mathbf{x};\boldsymbol{\phi}))q(\boldsymbol{\psi};\boldsymbol{\omega})\,\mathrm{d}\mathbf{z}\,\mathrm{d}\boldsymbol{\psi}\,. \tag{3.8}$$

We present the overall training procedure of Bayesian CDNs in Algorithm 3.2.

Note that, other Bayesian inferences on $\boldsymbol{\psi}$ obviously applicable in Bayesian CDNs. For example, we can also infer the posterior distribution by employing MC-dropout [Gal and Ghahramani, 2016]. That is, we simply apply dropout at each hidden layer of g and train the model using standard point estimate methods. To get the prediction, we simply average S samples of prediction given by f.

Employing VB is an attractive option for training CDNs, due to a couple of reasons. First, the uncertainty in parameters $\boldsymbol{\psi}$ is quantified, resulting in a hypothetically better uncertainty estimate in the predictive distribution. That is, whereas maximum-likelihood CDNs potentially quantify the heteroscedastic aleatoric uncertainty (captured by $\mathbf{z}$), Bayesian CDNs could potentially quantify both the aleatoric and epistemic uncertainty. Second, as shown in the objective of variational CDNs in eq. 3.7, no hyperparameter is needed. Therefore training variational CDNs should be relatively more straight-forward than training CDNs with the regularized maximum-likelihood objective (eq. 3.4), where we need to adjust the regularization hyperparameter λ.

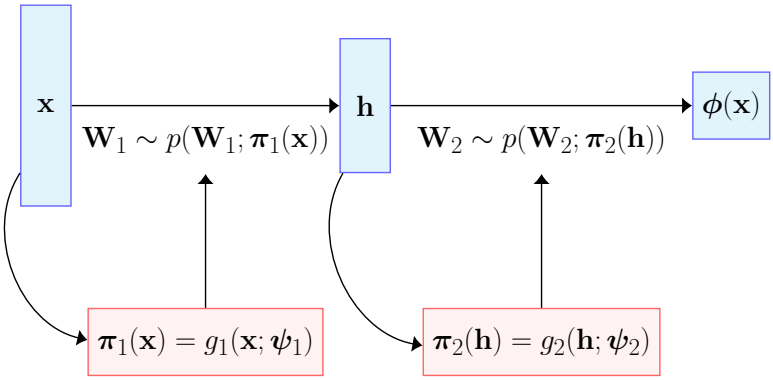


Figure 3.1: An example of a probabilistic hypernetwork applied to a two-layer MLP.

3.2 Probabilistic hypernetworks

CDN is an abstract framework for modeling compound distributions with NNs. In this section, we present a concrete example of how CDNs can be implemented.

Recall that we can see CDNs as defined in eq. 3.2 as two neural networks f and g, parametrized by $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, respectively. We assume that g parametrizes the generation of the stochastic components $\mathbf{z}$ of f (e.g. additional inputs, hidden units, or weights of f), depending on the input $\mathbf{x}$. In this model, we now define $\mathbf{z}$ to be the weights of f. In deterministic setting, this idea is known as *fast weights* or *hypernetworks* [Schmidhuber, 1992; Jia et al., 2016; Ha et al., 2017]. We can, therefore, see CDNs as the probabilistic generalization of the hypernetworks by assuming that the outputs of weight-generating network g to be the parameters of the CDNs' mixing distribution $p(\mathbf{z}; g(\mathbf{x}; \boldsymbol{\phi}))$, which results in a concrete instance of CDNs model called *probabilistic hypernetworks*. In the following, we will concretely define the choice of this distribution among other details.

For simplicity let us assume that f is a multi-layer perceptron (MLP) consisting of L layers, parameterized by a collection of weight matrices[1] $\mathbf{z} := \{\mathbf{W}_l\}_{l=1}^L$. We further assume that we do not use the deterministic parameters $\boldsymbol{\theta}$, that is we write the network as $f(\mathbf{x}; \mathbf{z})$. Note that, as before, we assume that $\boldsymbol{\psi} = \{\boldsymbol{\theta}, \boldsymbol{\phi}\}$, thus as $\boldsymbol{\theta} = \emptyset$, this implies that $\boldsymbol{\psi} = \boldsymbol{\phi}$. Hence, we will use $\boldsymbol{\psi}$ and $\boldsymbol{\phi}$ interchangeably from now on. Furthermore, let $\mathbf{h}_1, \ldots, \mathbf{h}_{L-1}$ be the hidden states of f, with $\mathbf{h}_0 := \mathbf{x}$ and $\mathbf{h}_L := f(\mathbf{x}; \mathbf{z})$. We now define g to be a sequence of L MLPs

$$\begin{aligned} g(\mathbf{x}; \boldsymbol{\psi}) &:= \{g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l)\}_{l=1}^L \\ \boldsymbol{\psi} &:= \{\boldsymbol{\psi}_l\}_{l=1}^L . \end{aligned} \tag{3.9}$$

Note that each g_l maps $\mathbf{h}_{l-1}$ (instead of $\mathbf{x}$) to the parameters of a distribution $p(\mathbf{W}_l; g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l))$ over weight matrix $\mathbf{W}_l$. The justification of this modeling decision is presented in Proposition 3.1.

Proposition 3.1. *Given all definitions above, we can sufficiently define* $g_l(\mathbf{x}; \boldsymbol{\psi}_l) := g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l)$ *, for all* $l = 1, \ldots, L$.

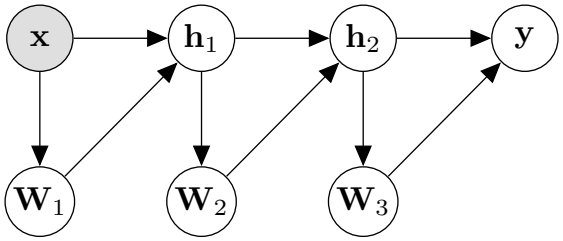


Figure 3.2: Graphical model of three-layer probabilistic hypernetworks.

Proof. For $l = 1$, the claim trivially holds as $\mathbf{h}_0 = \mathbf{x}$ implies $g_1(\mathbf{h}_0; \boldsymbol{\psi}_1) = g_1(\mathbf{x}; \boldsymbol{\psi}_1)$. Now, it is sufficient to show that for all $l = 2, \ldots, L$, $\mathbf{W}_l$ is conditionally independent to $\mathbf{x}$ given $\mathbf{h}_{l-1}$. We write the probabilistic hypernetworks as a graphical model, illustrated w.l.o.g. in Figure 3.2. Let $l > 1$ be arbitrary. Notice that by assumption, $\mathbf{h}_{l-1}$ is in the conditioning set. We observe that, although $\mathbf{h}_{l-1}$ is a collider in the graph, it is easy to see that $\mathbf{h}_{l-1}$ is not a collider in any path from $\mathbf{x}$ to $\mathbf{W}_l$. Thus, any of these paths is blocked by $\mathbf{h}_{l-1}$, implying $\mathbf{x}$ and $\mathbf{W}_l$ to be d-separated given $\mathbf{h}_{l-1}$. Therefore, $\mathbf{W}_l \perp\!\!\!\perp \mathbf{x} \,|\, \mathbf{h}_{l-1}$. □

We further assume independence between each $\mathbf{W}_l$ and let the joint distribution be given by

$$p(\mathbf{z}; g(\mathbf{x}; \boldsymbol{\psi})) := \prod_{l=1}^{L} p(\mathbf{W}_l; g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l)) \,. \tag{3.10}$$

Furthermore, to make our probabilistic hypernetworks fit the VB framework, we need further assumptions on $\boldsymbol{\psi}$, namely the choice of the approximate posterior $q(\boldsymbol{\psi}; \boldsymbol{\omega})$ and the prior $p(\boldsymbol{\psi})$. Similar to our assumption about $\mathbf{z}$, for each hypernetworks g_l, its parameters $\boldsymbol{\psi}_l$ are factored layer-wise. Thus, defining K_l to be the number of layers in g_l, we define

$$\begin{aligned} q(\boldsymbol{\psi}; \boldsymbol{\omega})) &:= \prod_{l=1}^{L} \prod_{k=1}^{K_l} q(\boldsymbol{\psi}_{lk}; \boldsymbol{\omega}_{lk}) \\ p(\boldsymbol{\psi}) &:= \prod_{l=1}^{L} \prod_{k=1}^{K_l} p(\boldsymbol{\psi}_{lk}) \,. \end{aligned} \tag{3.11}$$

An illustration of the probabilistic hypernetworks' computational graph is presented in Figure 3.1. Given the construction of the probabilistic hypernetworks based on variational CDNs above, we are now ready to concretely define the choice of statistical model for each of the $p(\mathbf{W}_l; g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l))$, $q(\boldsymbol{\psi}_{lk}; \boldsymbol{\omega})$, and $p(\boldsymbol{\psi}_{lk})$, and hence the overall mixing distribution $p(\mathbf{z}; g(\mathbf{x}; \boldsymbol{\psi}))$, the variational posterior $q(\boldsymbol{\psi}; \boldsymbol{\omega}))$, and the prior $p(\boldsymbol{\psi})$.

[1] We assume the bias vectors are absorbed to the corresponding weight matrices.

3.2.1 Probabilistic hypernetworks with matrix-variate normal distributions

A distribution that was recently applied as the statistical model of choice in BNNs [Louizos and Welling, 2016; Sun et al., 2017; Zhang et al., 2018; Ritter et al., 2018] is the matrix-variate normal (MVN) distribution [Gupta and Nagar, 1999], denoted by $\mathcal{MN}$. An MVN is the generalization of multi-variate normal for matrix random variable and is parametrized by three parameter matrices: a mean matrix $\mathbf{M}$ and two covariance factor matrices $\mathbf{A}$ and $\mathbf{B}$, and is defined as follows.

Definition 3.2 (Gupta and Nagar, 1999)**.** *A matrix random variable $\mathbf{X} \in \mathbb{R}^{m\times n}$ is distributed by a matrix-variate normal distribution with mean matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ and two (positive semi-definite) covariance factors $\mathbf{A} \in \mathbb{R}^{m\times m}$ and $\mathbf{B} \in \mathbb{R}^{n\times n}$, if*

$$\mathrm{vec}(\mathbf{X}) \sim \mathcal{N}(\mathrm{vec}(\mathbf{X}); \mathrm{vec}(\mathbf{M}), \mathbf{B} \otimes \mathbf{A}) \,,$$

We denote this distribution as $\mathcal{MN}(\mathbf{X}; \mathbf{M}, \mathbf{A}, \mathbf{B})$.

The following proposition shows that an MVN requires fewer parameters compared to a multi-variate Gaussian. This motivates us to use it as the statistical models over the weight matrices.

Proposition 3.3. *Let $\mathbf{X} \in \mathbb{R}^{m\times n}$ be a random matrix. Then using an MVN to model the distribution of $\mathbf{X}$ is more efficient than using a multi-variate Gaussian.*

Proof. By definition, we can write the MVN distribution over $\mathbf{X}$ as

$$\mathcal{N}(\mathrm{vec}(\mathbf{X}); \mathrm{vec}(\mathbf{M}), \mathbf{B} \otimes \mathbf{A}) \,,$$

for some matrices $\mathbf{M}, \mathbf{A}, \mathbf{B}$. It is implied that $\mathrm{vec}(\mathbf{M}) \in \mathbb{R}^{mn}$, $\mathbf{A} \in \mathbb{R}^{m\times m}$, and $\mathbf{B} \in \mathbb{R}^{n\times n}$. Thus the number of parameters of this distribution is $mn + m^2 + n^2$. In contrast, let $\mathcal{N}(\mathrm{vec}(\mathbf{X}); \boldsymbol{\mu}; \boldsymbol{\Sigma})$ be the multi-variate Gaussian over $\mathbf{X}$, for some vector $\boldsymbol{\mu}$ and matrix $\boldsymbol{\Sigma}$. This implies that $\boldsymbol{\mu} \in \mathbb{R}^{mn}$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{mn\times mn}$. Therefore the number of parameters of this distribution is $mn+(mn)^2$. Observe that $m, n \in \mathbb{N}$. For $m, n > 2$, we have that $m+n \leq 2\max(m, n) < mn$, thus $m^2+n^2 < (m+n)^2 < (mn)^2$. Therefore, asymptotically in the size of $\mathbf{X}$, the MVN is more efficient than the multi-variate Gaussian. □

Given the above mixing distribution, the reparametrization trick can be done layer-wise, described in the following proposition.

Proposition 3.4. *Let $\mathbf{M} \in \mathbb{R}^{m\times n}$, $\mathbf{A} \in \mathbb{R}^{m\times m}$, $\mathbf{B} \in \mathbb{R}^{n\times n}$ be some matrices and $\mathbf{I}_m \in \mathbb{R}^{m\times m}$, $\mathbf{I}_n \in \mathbb{R}^{n\times n}$ be identity matrices. Let $\mathbf{X} \in \mathbb{R}^{m\times n}$ be a matrix random variable distributed by $\mathcal{MN}(\mathbf{X}; \mathbf{M}, \mathbf{A}, \mathbf{B})$. Then it can be sampled by*

$$\mathbf{E} \sim \mathcal{MN}(\mathbf{E}; \mathbf{0}, \mathbf{I}_m, \mathbf{I}_n)$$
$$\mathbf{X} = \mathbf{M} + \mathbf{A}^{\frac{1}{2}}\mathbf{E}(\mathbf{B}^{\frac{1}{2}})^{\mathrm{T}}$$

Proof. By definition, we have that

$$\text{vec}(\mathbf{X}) \sim \mathcal{N}(\text{vec}(\mathbf{X}); \text{vec}(\mathbf{M}), \mathbf{B} \otimes \mathbf{A}) \,.$$

Recall the property of multi-variate Gaussian

$$\mathbf{x} = \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{\epsilon} \iff \mathbf{x} \sim \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \,,$$

for some $\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\epsilon} \in \mathbb{R}^n$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$, with $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \mathbf{I})$. Furthermore note that $(\mathbf{B} \otimes \mathbf{A})^{\frac{1}{2}} = \mathbf{B}^{\frac{1}{2}} \otimes \mathbf{A}^{\frac{1}{2}}$ and $\text{vec}(\mathbf{ACB}) = (\mathbf{B}^{\mathrm{T}} \otimes \mathbf{A})\text{vec}(\mathbf{C})$. Therefore, we can write the sampling procedure as follows.

$$\begin{aligned}
\text{vec}(\mathbf{E}) &\sim \mathcal{N}(\text{vec}(\mathbf{E}); \mathbf{0}, \mathbf{I}_{mn}) = \mathcal{N}(\text{vec}(\mathbf{E}); \mathbf{0}, \mathbf{I}_n \otimes \mathbf{I}_m) \\
\text{vec}(\mathbf{X}) &= \text{vec}(\mathbf{M}) + (\mathbf{B} \otimes \mathbf{A})^{\frac{1}{2}} \text{vec}(\mathbf{E}) \\
&= \text{vec}(\mathbf{M}) + (\mathbf{B}^{\frac{1}{2}} \otimes \mathbf{A}^{\frac{1}{2}}) \text{vec}(\mathbf{E}) \\
&= \text{vec}(\mathbf{M}) + \text{vec}(\mathbf{A}^{\frac{1}{2}} \mathbf{E} (\mathbf{B}^{\frac{1}{2}})^{\mathrm{T}}) \,.
\end{aligned}$$

Undoing the vectorization operator, we get

$$\begin{aligned}
\mathbf{E} &\sim \mathcal{MN}(\mathbf{E}; \mathbf{0}, \mathbf{I}_p, \mathbf{I}_q) \\
\mathbf{X} &= \mathbf{M} + \mathbf{A}^{\frac{1}{2}} \mathbf{E} (\mathbf{B}^{\frac{1}{2}})^{\mathrm{T}} \,.
\end{aligned}$$

□

Observe that, from the proposition above, it is easy to see that

$$\mathbf{E} \sim \mathcal{MN}(\mathbf{E}; \mathbf{0}, \mathbf{I}_m, \mathbf{I}_n) \iff e_{ij} \sim \mathcal{N}(e_{ij}; 0, 1)$$

for all elements e_{ij} in $\mathbf{E}$.

The final necessary detail of the probabilistic hypernetworks is how to compute the KL-divergence between two MVN distributions. We present this as a proposition as follows.

Proposition 3.5 (Louizos and Welling, 2016)**.** *Let* $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{A}_1, \mathbf{A}_2 \in \mathbb{R}^{m \times m}$, $\mathbf{B}_1, \mathbf{B}_2 \in \mathbb{R}^{n \times n}$. *Define*

$$\begin{aligned}
p(\mathbf{X}) &:= \mathcal{MN}(\mathbf{X}; \mathbf{M}_1, \mathbf{A}_1, \mathbf{B}_1) \\
q(\mathbf{X}) &:= \mathcal{MN}(\mathbf{X}; \mathbf{M}_2, \mathbf{A}_2, \mathbf{B}_2) \,.
\end{aligned}$$

Then

$$\begin{aligned}
D_{\mathrm{KL}}[\, p(\mathbf{X}) \| q(\mathbf{X})] = \frac{1}{2} \Big(& \text{tr}(\mathbf{A}_2^{-1}\mathbf{A}_1)\text{tr}(\mathbf{B}_2^{-1}\mathbf{B}_1) \\
& + \text{tr}((\mathbf{M}_2 - \mathbf{M_1})^{\mathrm{T}} \mathbf{A}_2^{-1} (\mathbf{M}_2 - \mathbf{M_1}) \mathbf{B}_2^{-1}) \\
& - mn + n \log |\mathbf{A}_2| + m \log |\mathbf{B}_2| - n \log |\mathbf{A}_1| - m \log |\mathbf{B}_1| \Big) \,.
\end{aligned}$$

Proof. By definition of MVN, we can write

$$p(\mathbf{X}) = \mathcal{N}(\mathrm{vec}(\mathbf{X}); \mathrm{vec}(\mathbf{M}_1), \mathbf{B}_1 \otimes \mathbf{A}_1)$$
$$q(\mathbf{X}) = \mathcal{N}(\mathrm{vec}(\mathbf{X}); \mathrm{vec}(\mathbf{M}_2), \mathbf{B}_2 \otimes \mathbf{A}_2)\,.$$

By the standard result in multivariate Gaussian (e.g. see Duchi), the KL-divergence between them is

$$\begin{aligned} D_{\mathrm{KL}}[\,p(\mathbf{X}) \|\, q(\mathbf{X})] &= \frac{1}{2}\Big(\mathrm{tr}(\mathbf{B}_2 \otimes \mathbf{A}_2)^{-1}\mathrm{tr}(\mathbf{B}_1 \otimes \mathbf{A}_1) \\ &\quad + (\mathrm{vec}(\mathbf{M_2} - \mathrm{vec}(\mathbf{M}_1))^{\mathrm{T}}(\mathbf{B}_2 \otimes \mathbf{A}_2)^{-1}(\mathrm{vec}(\mathbf{M_2} - \mathrm{vec}(\mathbf{M}_1)) \\ &\quad - mn + \log\frac{|\mathbf{B}_2 \otimes \mathbf{A}_2|}{|\mathbf{B}_1 \otimes \mathbf{A}_1|}\Big) \\ &=: \frac{1}{2}(x_1 + x_2 - mn + x_3)\,. \end{aligned}$$

We now manipulate each part x_1, x_2, x_3 of the equation, by employing the properties of Kronecker product $\otimes$ and vectorization operator vec. Namely, for some $\mathbf{P}, \mathbf{Q}, \mathbf{R}, \mathbf{S} \in \mathbb{R}^{m\times m}$ and some $\mathbf{T} \in \mathbb{R}^{n\times n}$:

- $(\mathbf{P} \otimes \mathbf{Q})^{-1} = \mathbf{P}^{-1} \otimes \mathbf{Q}^{-1}$,
- $(\mathbf{P} \otimes \mathbf{Q})(\mathbf{R} \otimes \mathbf{S}) = (\mathbf{PR}) \otimes (\mathbf{QR})$,
- $\mathrm{tr}(\mathbf{P} \otimes \mathbf{Q}) = \mathrm{tr}(\mathbf{P})\mathrm{tr}(\mathbf{Q})$,
- $\mathrm{vec}(\mathbf{P} - \mathbf{Q}) = \mathrm{vec}(\mathbf{P}) - \mathrm{vec}(\mathbf{Q})$,
- $(\mathbf{Q} \otimes \mathbf{P})\mathrm{vec}(\mathbf{R}) = \mathrm{vec}(\mathbf{PRQ})$,
- $\mathrm{vec}(\mathbf{P})^{\mathrm{T}}\mathrm{vec}(\mathbf{Q}) = \mathrm{tr}(\mathbf{P}^{\mathrm{T}}\mathbf{Q})$, and
- $|\mathbf{P} \otimes \mathbf{T}| = |\mathbf{P}|^n|\mathbf{T}|^m$.

Thus,

$$\begin{aligned} x_1 &= \mathrm{tr}((\mathbf{B}_2 \otimes \mathbf{A}_2)^{-1}(\mathbf{B}_1 \otimes \mathbf{A}_1)) \\ &= \mathrm{tr}((\mathbf{B}_2^{-1} \otimes \mathbf{A}_2^{-1})(\mathbf{B}_1 \otimes \mathbf{A}_1)) \\ &= \mathrm{tr}((\mathbf{B}_2^{-1}\mathbf{B}_1) \otimes (\mathbf{A}_2^{-1}\mathbf{A}_1)) \\ &= \mathrm{tr}(\mathbf{B}_2^{-1}\mathbf{B}_1)\mathrm{tr}(\mathbf{A}_2^{-1}\mathbf{A}_1)\,, \end{aligned}$$

$$\begin{aligned} x_2 &= (\mathrm{vec}(\mathbf{M_2}) - \mathrm{vec}(\mathbf{M}_1))^{\mathrm{T}}(\mathbf{B}_2 \otimes \mathbf{A}_2)^{-1}(\mathrm{vec}(\mathbf{M_2}) - \mathrm{vec}(\mathbf{M}_1)) \\ &= \mathrm{vec}(\mathbf{M_2} - \mathbf{M}_1)^{\mathrm{T}}(\mathbf{B}_2^{-1} \otimes \mathbf{A}_2^{-1})\mathrm{vec}(\mathbf{M_2} - \mathbf{M}_1) \\ &= \mathrm{vec}(\mathbf{M_2} - \mathbf{M}_1)^{\mathrm{T}}\mathrm{vec}(\mathbf{A}_2^{-1}(\mathbf{M_2} - \mathbf{M}_1)\mathbf{B}_2^{-1}) \\ &= \mathrm{tr}((\mathbf{M_2} - \mathbf{M}_1)^{\mathrm{T}}\mathbf{A}_2^{-1}(\mathbf{M_2} - \mathbf{M}_1)\mathbf{B}_2^{-1})\,, \end{aligned}$$

$$\begin{aligned} x_3 &= \log\frac{|\mathbf{B}_2 \otimes \mathbf{A}_2|}{|\mathbf{B}_1 \otimes \mathbf{A}_1|} \\ &= \log\frac{|\mathbf{B}_2|^m|\mathbf{A}_2|^n}{|\mathbf{B}_1|^m|\mathbf{A}_1|^n} \\ &= m\log|\mathbf{B}_2| + n\log|\mathbf{A}_2| - m\log|\mathbf{B}_1| - n\log|\mathbf{A}_1|\,. \end{aligned}$$

The claim follows by substituting $x1, x2, x3$ back to the KL-divergence formula for multi-variate Gaussian. □

In probabilistic hypernetworks, we specifically assume that all of the covariance factor matrices are diagonal matrices, following Louizos and Welling [2016]. Thus all in all, following eq. 3.10, the mixing distribution of the probabilistic hypernetworks is given by

$$\begin{aligned} p(\mathbf{z}; g(\mathbf{x}; \boldsymbol{\psi})) &:= \prod_{l=1}^{L} \mathcal{MN}(\mathbf{W}_l; g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l)) \\ &:= \prod_{l=1}^{L} \mathcal{MN}(\mathbf{W}_l; \mathbf{M}_l, \operatorname{diag}(\mathbf{a}_l), \operatorname{diag}(\mathbf{b}_l)) \,, \end{aligned} \tag{3.12}$$

where each g_l is now specifically maps $\mathbf{h}_{l-1} \longmapsto \{\mathbf{M}_l, \mathbf{a}_l, \mathbf{b}_l\}$. Meanwhile, the regularizer distribution is defined to be

$$p(\mathbf{z}) := \prod_{l=1}^{L} p(\mathbf{W}_l) := \prod_{l=1}^{L} \mathcal{MN}(\mathbf{W}_l; \mathbf{0}, \mathbf{I}, \mathbf{I}) \,, \tag{3.13}$$

Following eq. 3.11, the variational posterior $q(\boldsymbol{\psi}; \boldsymbol{\omega})$ and the prior $p(\boldsymbol{\psi})$ are defined as

$$\begin{aligned} q(\boldsymbol{\psi}; \boldsymbol{\omega}) &:= \prod_{l=1}^{L} \prod_{k=1}^{K_l} \mathcal{MN}(\boldsymbol{\psi}_{lk}; \boldsymbol{\omega}_{lk}) \\ &:= \prod_{l=1}^{L} \prod_{k=1}^{K_l} \mathcal{MN}(\boldsymbol{\psi}_{lk}; \mathbf{N}_{lk}, \operatorname{diag}(\mathbf{c}_{lk}), \operatorname{diag}(\mathbf{d}_{lk})) \end{aligned}$$

$$p(\boldsymbol{\psi}) := \prod_{l=1}^{L} \prod_{k=1}^{K_l} \mathcal{MN}(\boldsymbol{\psi}_{lk}; \mathbf{0}, \mathbf{I}, \mathbf{I}) \,, \tag{3.14}$$

where we have assumed that $\boldsymbol{\omega}_{lk} = \{\mathbf{N}_{lk}, \operatorname{diag}(\mathbf{c}_{lk}), \operatorname{diag}(\mathbf{d}_{lk})\}$.

As results of Proposition 3.4 and 3.5, in probabilistic hypernetworks, the reparametrization trick for both $\mathbf{z}$ and $\boldsymbol{\psi}$, along with the KL-divergence term in eq. 3.7 are given by the following corollaries.

Corollary 3.6. *Given the mixing distribution (eq. 3.10) and the variational posterior (eq. 3.14) of the probabilistic hypernetworks, Proposition 3.4 implies that reparametrization trick for each $\mathbf{W}_l \in \mathbb{R}^{m \times n}$ of $\mathbf{z}$ for some $m, n \in \mathbb{N}$, can be done by*

$$\begin{aligned} \mathbf{E} &\sim \mathcal{MN}(\mathbf{E}; \mathbf{0}, \mathbf{I}_m, \mathbf{I}_n) \\ \mathbf{W}_l &= \mathbf{M}_l + \operatorname{diag}(\mathbf{a}_l)^{\frac{1}{2}} \mathbf{E} \operatorname{diag}(\mathbf{b}_l)^{\frac{1}{2}} \,. \end{aligned}$$

Similarly, for each parameter matrix $\boldsymbol{\psi}_{lk} \in \mathbb{R}^{m\times n}$ *for some* $m, n \in \mathbb{N}$*:*

$$\begin{aligned}\mathbf{E} &\sim \mathcal{MN}(\mathbf{E}; \mathbf{0}, \mathbf{I}_m, \mathbf{I}_n)\\ \boldsymbol{\psi}_{lk} &= \mathbf{N}_{lk} + \operatorname{diag}(\mathbf{c}_{lk})^{\frac{1}{2}}\mathbf{E}\operatorname{diag}(\mathbf{d}_{lk})^{\frac{1}{2}}\,.\end{aligned}$$

Proof. Trivially implied by Proposition 3.4 and using the fact that diagonal matrices are symmetric. □

Corollary 3.7. *Given the mixing distribution (eq. 3.10) and the variational posterior (eq. 3.14) of probabilistic hypernetworks, Proposition 3.5 implies that for each* $\mathbf{W}_l \in \mathbb{R}^{m\times n}$ *and for some* $m, n \in \mathbb{N}$*,*

$$\begin{aligned}&D_{\mathrm{KL}}[\,\mathcal{MN}(\mathbf{W}_l; \mathbf{M}_l, \operatorname{diag}(\mathbf{a}_l), \operatorname{diag}(\mathbf{b}_l))\|\,\mathcal{MN}(\mathbf{W}_l; \mathbf{0}, \mathbf{I}, \mathbf{I})] =\\ &\quad\frac{1}{2}\left(\sum_{i=1}^{m} a_{li}\sum_{j=1}^{n} b_{lj} + \|\mathbf{M}_l\|_F^2 - mn - n\sum_{i=1}^{m}\log a_{li} - m\sum_{j=1}^{n}\log b_{lj}\right)\,.\end{aligned}$$

Furthermore, for each $\boldsymbol{\psi}_{lk} \in \mathbb{R}^{m\times n}$ *and for some* $m, n \in \mathbb{N}$*,*

$$\begin{aligned}&D_{\mathrm{KL}}[\,\mathcal{MN}(\boldsymbol{\psi}_{lk}; \mathbf{N}_{lk}, \operatorname{diag}(\mathbf{c}_{lk}), \operatorname{diag}(\mathbf{d}_{lk}))\|\,\mathcal{MN}(\boldsymbol{\psi}_{lk}; \mathbf{0}, \mathbf{I}, \mathbf{I})] =\\ &\quad\frac{1}{2}\left(\sum_{i=1}^{m} c_{lki}\sum_{j=1}^{n} d_{lkj} + \|\mathbf{N}_{lk}\|_F^2 - mn - n\sum_{i=1}^{m}\log c_{lki} - m\sum_{j=1}^{n}\log d_{lkj}\right)\,.\end{aligned}$$

Proof. We only prove the first claim, as the proof of the second claim is identical. To avoid clutter, let $p_1 := \mathcal{MN}(\mathbf{W}_l; \mathbf{M}_l, \operatorname{diag}(\mathbf{a}_l), \operatorname{diag}(\mathbf{b}_l))$ and $p_2 := \mathcal{MN}(\mathbf{W}_l; \mathbf{0}, \mathbf{I}, \mathbf{I})$. By Proposition 3.5, employing the identity that the determinant of some diagonal matrix $\operatorname{diag}(\mathbf{t})$ is $|\operatorname{diag}(\mathbf{t})| = \prod_i t_i$ and that $\|\mathbf{X}\|_F = \sqrt{\operatorname{tr}(\mathbf{X}^{\mathrm{T}}\mathbf{X})}$, we have

$$\begin{aligned}D_{\mathrm{KL}}[p_1\|p_2] &= \frac{1}{2}\Big(\operatorname{tr}(\mathbf{I}^{-1}\operatorname{diag}(\mathbf{a}_l))\operatorname{tr}(\mathbf{I}^{-1}\operatorname{diag}(\mathbf{b}_l))\\ &\qquad + \operatorname{tr}((\mathbf{0}-\mathbf{M}_l)^{\mathrm{T}}\mathbf{I}^{-1}(\mathbf{0}-\mathbf{M}_l)\mathbf{I}^{-1}) - mn + n\log|\mathbf{I}|\\ &\qquad + m\log|\mathbf{I}| - n\log|\operatorname{diag}(\mathbf{a}_l)| - m\log|\operatorname{diag}(\mathbf{b}_l)|\Big)\\ &= \frac{1}{2}\Big(\sum_{i=1}^{m} a_{li}\sum_{j=1}^{n} b_{lj} + \operatorname{tr}((-\mathbf{M}_l)^{\mathrm{T}}(-\mathbf{M}_l))\\ &\qquad - mn + n\log\prod_{i=1}^{m} a_{li} - m\log\prod_{j=1}^{n} b_{lj}\Big)\\ &= \frac{1}{2}\left(\sum_{i=1}^{m} a_{li}\sum_{j=1}^{n} d_{lj} + \|\mathbf{M}_l\|_F^2 - mn - n\sum_{i=1}^{m}\log a_{li} - m\sum_{j=1}^{n}\log b_{lj}\right)\,.\end{aligned}$$

□

The above corollary tells us how to compute the KL-divergence between each distribution of $\mathbf{W}_l$ to the standard MVN, in closed-form. The following corollary will show how to compute the KL-divergence, if we consider the overall mixing distribution, i.e. the product of the distribution of $\mathbf{W}_l$, for all $l = 1, \ldots, L$.

Corollary 3.8. *The KL-divergence between the mixing distribution* $p(\mathbf{z}; g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l))$ *and* $p(\mathbf{z})$ *(eq. 3.10) is*

$$D_{\mathrm{KL}}[\, p(\mathbf{z}; g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l)) \| \, p(\mathbf{z})] = \sum_{l=1}^{L} D_{\mathrm{KL}}[\, \mathcal{MN}(\mathbf{W}_l; \mathbf{M}_l, \mathrm{diag}(\mathbf{a}_l), \mathrm{diag}(\mathbf{b}_l)) \| \, \mathcal{MN}(\mathbf{W}_l; \mathbf{0}, \mathbf{I}, \mathbf{I})] \,.$$

Moreover, the KL-divergence between variational posterior $q(\boldsymbol{\psi}; \boldsymbol{\omega})$ *and the prior* $p(\boldsymbol{\psi})$ *defined in eq. 3.14 is given by*

$$D_{\mathrm{KL}}[\, q(\boldsymbol{\psi}; \boldsymbol{\omega}) \| \, p(\boldsymbol{\psi})] = \sum_{l=1}^{L} \sum_{k=1}^{K_l} D_{\mathrm{KL}}[\, \mathcal{MN}(\boldsymbol{\psi}_{lk}; \mathbf{N}_{lk}, \mathrm{diag}(\mathbf{c}_{lk}), \mathrm{diag}(\mathbf{d}_{lk})) \| \, \mathcal{MN}(\boldsymbol{\psi}_{lk}; \mathbf{0}, \mathbf{I}, \mathbf{I})] \,.$$

Proof. Note that KL-divergence admits chain rule

$$D_{\mathrm{KL}}[\, p(x, y) \| \, q(x, y)] = D_{\mathrm{KL}}[\, p(x) \| \, q(x)] + D_{\mathrm{KL}}[\, p(y|x) \| \, q(y|x)] \,,$$

for arbitrary random variables x, y and distributions $p(x, y)$, $q(x, y)$. Assuming x and y are independent, this implies that

$$D_{\mathrm{KL}}[\, p(x)p(y) \| \, q(x)q(y)] = D_{\mathrm{KL}}[\, p(x) \| \, q(x)] + D_{\mathrm{KL}}[\, p(y) \| \, q(y)] \,.$$

In addition, we use the fact that both $p(\mathbf{z}; g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l))$ and $p(\mathbf{z})$, and $q(\boldsymbol{\psi}; \boldsymbol{\omega})$ and $p(\boldsymbol{\psi})$ are independent layer-wise in eqs. (3.12) and (3.13), and in eq. 3.14, respectively. Therefore, the overall KL-divergence between them is the sum over each layer's KL-divergence, which prove the claim. □

These corollaries tell us that both the sampling process and the KL-divergence of the probabilistic hypernetworks can be done layer-wise, leading to efficient computation.

3.2.2 Vector scaling parametrization

The naive formulation of g_l can be very expensive in term of number of parameters. Suppose that for all $l = 1, \ldots, L$, $\mathbf{W}_l \in \mathbb{R}^{m \times n}$ and g_l is a two layer MLP with k hidden units. Then g_l would have $mk + kmn + km + kn$ many parameters, which quickly becomes very large for a moderately sized NNs. The majority of the parameters are needed to define the mean matrix $\mathbf{M}_l$. Following the approach of Ha et al. [2017] and Krueger et al. [2017], we make a trade-off between expressiveness of g_l on the mean matrix with the number of parameter by instead replacing $\mathbf{M}_l$ with a matrix $\mathbf{V}_l$ of the same size and a vector $\mathbf{u}_l \in \mathbb{R}^m$, which is the output of g_l. Thus, now g_l maps $\mathbf{h}_{l-1} \longmapsto \{\mathbf{u}_l, \mathbf{a}_l, \mathbf{b}_l\}$ and we can get $\mathbf{M}_l$ by

$$\mathbf{M}_l = \begin{bmatrix} u_{l1}\mathbf{v}_{l1} \\ u_{l2}\mathbf{v}_{l2} \\ \cdots \\ u_{lr}\mathbf{v}_{lr} \end{bmatrix} \,. \tag{3.15}$$

That is, each element of $\mathbf{u}_l$ is being used to scale the corresponding row of $\mathbf{V}_l$. Note that although $\mathbf{V}_l$ is a parameter matrix with the same size of $\mathbf{M}_l$, it crucially is not an output of g_l as in the naive parametrization. Thus the number of parameter of g_l is now $mk + mn + 2km + kn$, which is more manageable and implementable for larger weight matrices, as the following proposition shows.

Proposition 3.9. *Without loss of generality, suppose that for all* $l = 1, \ldots, L$, $\mathbf{W}_l \in \mathbb{R}^{m \times n}$ *and* g_l *is a two layer MLP with* k *hidden units. Then, the vector scaling parametrization is more efficient than naive parametrization of CDNs mixture component's each layer MVN distribution.*

Proof. We have shown that naive parametrization requires $mk + kmn + km + kn$ parameters, while vector scaling parametrization requires $mk + mn + 2km + kn$, for some $k, m, n \in \mathbb{N}$. Notice that we only need to compare $kmn + km$ with $mn + 2km$. With some trivial algebra, this means we need to compare n with $1 + \frac{n}{k}$, respectively. It is then clear that, for $n > 2$ and $k > 1$, we have $n > 1 + \frac{n}{k}$. □

3.3 Related work

Models similar to CDNs, i.e. in the form of

$$p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\phi}) = \int p(\mathbf{y}|\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}) p(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi}) \, \mathrm{d}\mathbf{z} \,, \tag{3.16}$$

have previously been studied in various settings.

In latent variable models, the approximate posterior predictive distribution of the variational auto-encoder (VAE) [Kingma and Welling, 2014] has similar form to eq. 3.16, although it is focusing on unsupervised learning problem, i.e. we assume $\mathcal{D} := \{\mathbf{x}_n\}_{n=1}^N$:

$$p(\mathbf{x}|\mathcal{D}) = \int p(\mathbf{x}|\mathbf{z}; \boldsymbol{\theta}) p(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi}) \, \mathrm{d}\mathbf{z} \,, \tag{3.17}$$

where $\mathbf{z}$ is the latent variable, $p(\mathbf{x}|\mathbf{z}; \boldsymbol{\theta})$ is the *encoder* parametrized by an NN with parameter $\boldsymbol{\theta}$, and $p(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi})$ is the *decoder* parametrized by another NN with parameter $\boldsymbol{\phi}$. In this sense, CDNs can be seen as a supervised counterpart of VAE. Although similar to CDNs trained with maximum-likelihood objective, VAE learns the point estimates of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, it does so by applying variational Bayes (VB) over $\mathbf{z}$. That is, $p(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi})$ is assumed to approximate the posterior $p(\mathbf{z}|\mathbf{x}, \mathcal{D})$ of $\mathbf{z}$. Therefore, VAE optimizes an ELBO objective in contrast to the maximum-likelihood objective of CDNs (eq. 3.4).

Variational information bottleneck (VIB) [Alemi et al., 2017] can be seen as another counterpart of VAE in supervised setting, i.e. when the dataset is $\mathcal{D} := \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$, as in CDNs. However, it has two crucial differences to CDNs. First, $\mathbf{z}$ in VIB are bottleneck units, i.e. stochastic hidden units in a particular layer of an NN, in contrast to CDNs where $\mathbf{z}$ could be any stochastic component of an NN,

e.g. inputs, hidden units, or weights. Second, VIB is derived from the information bottleneck method [Tishby et al., 2001]: Although the objective of VIB is very similar to the maximum-likelihood CDNs' objective (eq. 3.4), they crucially differ in the order of the log and the expectation. Naturally, both of these objectives get equivalent whenever only a single sample of $\mathbf{z}$ is used to do the Monte Carlo integration in eq. 3.3.

Depeweg et al. [2017, 2018] proposed BNN+LV which equips BNNs with latent variables, resulting in a model given by

$$p(\mathbf{y}|\mathbf{x}) = \iint p(\mathbf{y}|\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}) p(\mathbf{z}|\mathbf{x}) p(\boldsymbol{\theta}) \, \mathrm{d}\mathbf{z} \, \mathrm{d}\boldsymbol{\theta} \,. \tag{3.18}$$

Notice that this model is different to Bayesian CDNs (section 3.1.2) as it employs only a single NN and assumes that the variables $\mathbf{z}$ and parameters $\boldsymbol{\theta}$ of the NN are independent. Moreover, Depeweg et al. [2017] specifically assume that the latent variable $\mathbf{z}$ is incorporated as an additional input to the neural network. Finally, instead of deriving the objective from KL-divergence between the approximate posterior and the exact posterior of $\mathbf{z}$ and $\boldsymbol{\theta}$, they use the more general α-divergence.

Malinin and Gales [2018] proposed Prior Networks to also capture what they call "distributional uncertainty", i.e. uncertainty due to the mismatch between the distributions of test and training data. The model can be described similarly as in eq. 3.16:

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}; \mathbf{z}) p(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi}) \, \mathrm{d}\mathbf{z} \,. \tag{3.19}$$

However, notice that they assume that $\mathbf{z} \sim p(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi})$ is directly parametrizing a distribution $p(\mathbf{y}; \mathbf{z})$. For example, if $\mathbf{z}$ is a Dirichlet random variable, then $p(\mathbf{y}; \mathbf{z})$ can be chosen as Categorical distribution. Thus, in contrast to CDNs, where neural networks are employed in both mixture components and mixing distribution, Prior Networks only assume a single neural network. Furthermore, they do not follow the procedures described in Section 2.2. Rather, they use a multi-task objective that explicitly use out-of-distribution samples to train the model. This is in contrast to CDNs and other models we have discussed so far.

Chapter 4

Experiments

As we saw in the previous chapter, CDNs provide a way to quantify predictive uncertainty of neural networks, by assuming that their weights are random variables. One can then train CDNs using well-established methods described in Chapter 2, such as maximum-likelihood and variational Bayesian inference.

Notice that the predictive distribution of CDNs (eq. 3.3) forms compound distribution as seen in standard BNNs, with the difference in the form of the distribution of the parameters. That is, CDNs assume that the parameters are conditioned to the input, while BNNs do not. CDNs are also connected to the mixture models in the way that they are derived. CDNs extend the formulation of finite (conditional) mixture models, in the form of MDNs, into an uncountably infinite number of mixture components. It is thus fitting to experimentally compare CDNs to recent BNNs and mixture model approaches in uncertainty quantification tasks.

In this chapter, we will present the empirical results of CDNs, in particular, the probabilistic hypernetworks, compared to the state-of-the-art models in uncertainty quantification. This chapter is organized as follows: In Section 4.1 we will discuss our methodology and the experimental setups. We will then validate the predictive uncertainty estimate of CDNs on toy datasets, in Section 4.2. Afterward, we will present our experimental results in standard AI safety tasks: out-of-distribution data classification in Section 4.3 and adversarial examples in Section 4.4. To conclude this chapter, we empirically compare CDNs with the variational information bottleneck, which is a model that is very similar to CDNs (Section 3.3), in Section 4.5.

4.1 Experiment setup

We consider several standard tasks in our experimental analysis: 1D toy regression problems inspired by Hernández-Lobato and Adams [2015], classification under out-of-distribution (OOD) data, and detection of and defense against adversarial examples [Szegedy et al., 2014]. We refer to the CDNs that are trained via $\mathcal{L}_{\text{ML}}$ (eq. 3.4) and $\mathcal{L}_{\text{VB}}$ (eq. 3.7) as ML-CDNs and VB-CDNs, respectively. The following recent models, both Bayesian and non-Bayesian, are considered as the baselines:

- **Variational matrix Gaussian (VMG)** [Louizos and Welling, 2016] is a

BNN where variational Bayesian inference with an MVN distribution is being used to approximate its posterior.

- **Multiplicative normalizing flow (MNF)** [Louizos and Welling, 2017] models the approximate posterior of a BNN as a compound distribution where the mixing density is given by a normalizing flow [Rezende and Mohamed, 2015]. We use the implementation provided by the authors.[1] We anneal the weighting of the KL-term from 0 to 1 over the course of training as we found that it is necessary to achieve the results reported by Louizos and Welling [2017].

- **Noisy K-FAC** [Zhang et al., 2018] uses an MVN approximate posterior and applies approximate natural-gradient-based [Amari, 1998] maximization on the VI objective. The resulting algorithm can be seen as the noisy version of the K-FAC method [Martens and Grosse, 2015]. We use the authors' implementation[2], and use the hyperparameters as suggested by [Zhang et al., 2018], except for the KL-term weighting, where we set it to be 1 to reflect the correct lower bound.

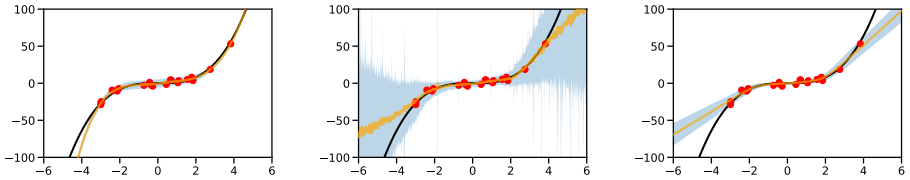
- **Monte Carlo dropout (MC-dropout or MCD)** [Gal and Ghahramani, 2016] uses dropout method [Srivastava et al., 2014] as approximate Bayesian inference. Furthermore, they show that MC-dropout is equivalent to an approximation to the deep Gaussian process [Damianou and Lawrence, 2013]. In our experiments, the dropout probability is set to 0.5 and the weight decay parameter to 0.0001.

- **Deep Ensemble** (DE) [Lakshminarayanan et al., 2017] represents the frequentist approaches of uncertainty quantification by using three ingredients: calibration, mixture model (ensemble), and adversarial training. As suggested by the authors, we set the number of mixture components to 5, while the adversarial perturbation strength is set to 1% of the input range and the weight decay is set to 0.0001.

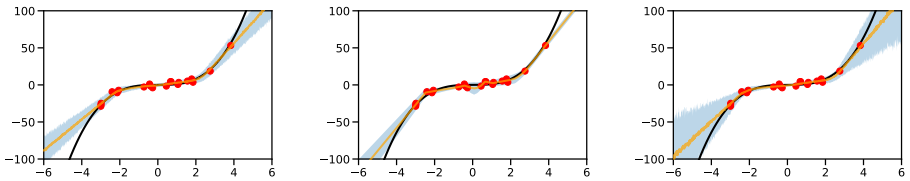
- **Dirichlet Prior Networks** (DPNs) [Malinin and Gales, 2018] also represents the non-Bayesian approaches in uncertainty quantification. Specifically, it proposes to use the so-called Prior Networks to quantify distributional uncertainty. We use the default hyperparameters as suggested by the authors: We set the target precision to 10^{-3} and the smoothing hyperparameter to 10^{-6}. Meanwhile, as DPNs require us to use OOD data during training, we use Fashion-MNIST [Xiao et al., 2017] as the OOD counterpart of MNIST and vice-versa, while SVHN [Netzer et al., 2011] is used as the OOD counterpart of CIFAR-10.

We estimate the predictive distribution $p(\mathbf{y}|\mathbf{x})$ of the CDNs, based on 100 joint samples of $\boldsymbol{\psi} \sim q(\boldsymbol{\psi}; \boldsymbol{\omega}), \mathbf{z} \sim p(\mathbf{z}; g(\mathbf{x}; \boldsymbol{\psi}))$ for VB-CDNs and 100 samples of $\mathbf{z} \sim$

[1] https://github.com/AMLab-Amsterdam/MNF_VBNN
[2] https://github.com/gd-zhang/Noisy-K-FAC



(a) ML-CDN (b) VB-CDN (c) noisy K-FAC



(d) VMG (e) Deep Ensemble (f) MC-dropout

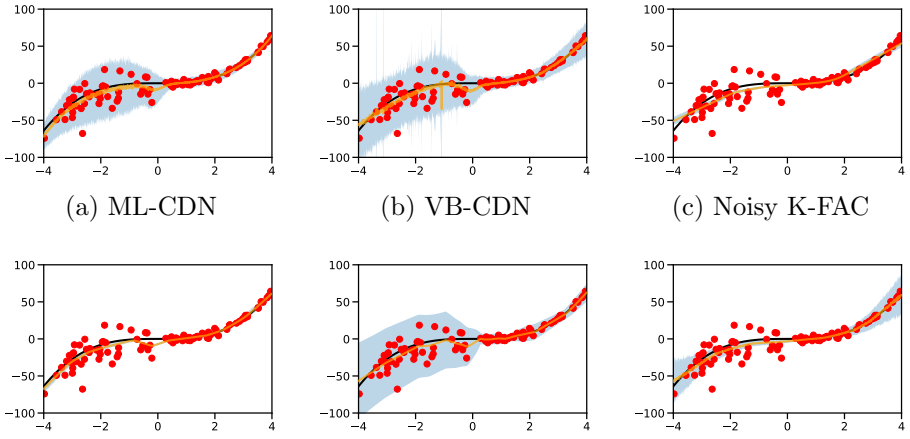
Figure 4.1: Comparison of the predictive distributions given by the CDNs and the baselines on toy datasets with homoscedastic noise and few samples. Black lines correspond to the true noiseless function, red dots correspond to samples, orange lines and shaded regions correspond to the empirical mean and the ± 3 standard deviation of the predictive distribution, respectively.

$p(\mathbf{z}; g(\mathbf{x}; \boldsymbol{\psi}))$ for ML-CDNs. We also draw 100 samples from the posterior to approximate the predictive distribution of BNN baselines. If not stated otherwise, we use a single sample to perform Monte Carlo integration during training.[3] We pick the regularization hyperparameter λ for ML-CDNs (eq. 3.4) out of the set $\{10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$ which maximizes the validation accuracy. We use Adam [Kingma and Ba, 2015] with the default hyperparameters for optimization in all experiments. Where mini-batching is necessary, e.g. on MNIST and CIFAR-10, we use mini-batches of size 200. All models are optimized over 10000 iterations in the toy regression experiments, 20000 iterations (≈67 epochs) in experiments on MNIST and Fashion-MNIST, and 100 epochs in experiments on CIFAR-10. We chose ReLU and hyperbolic tangent as the nonlinearity of the ML-CDNs' and VB-CDNs' hypernetworks, respectively. The source code for all our experiments is available at https://github.com/wiseodd/compound-density-networks.

4.2 Toy regression

Following Hernández-Lobato and Adams [2015], we generate our first toy regression dataset as follows: We sample 20 input points $x \sim \mathcal{U}[-4, 4]$ and their target values $y = x^3 + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 3^2)$, i.e. the data noise is homoscedastic. We aim at analyzing how well the target function is modeled over the larger interval $[-6, 6]$. Having only a few data points, it is a desirable property of a model to express high

[3] Kingma and Welling [2014] argued that using a single sample is sufficient as long as the batch size is sufficiently large, e.g. 100.



(a) ML-CDN (b) VB-CDN (c) Noisy K-FAC

(d) VMG (e) DeepEnsemble (f) MC-dropout

Figure 4.2: Comparison of the predictive distributions given by the CDNs and the baselines on toy datasets with heteroscedastic noise and many samples.

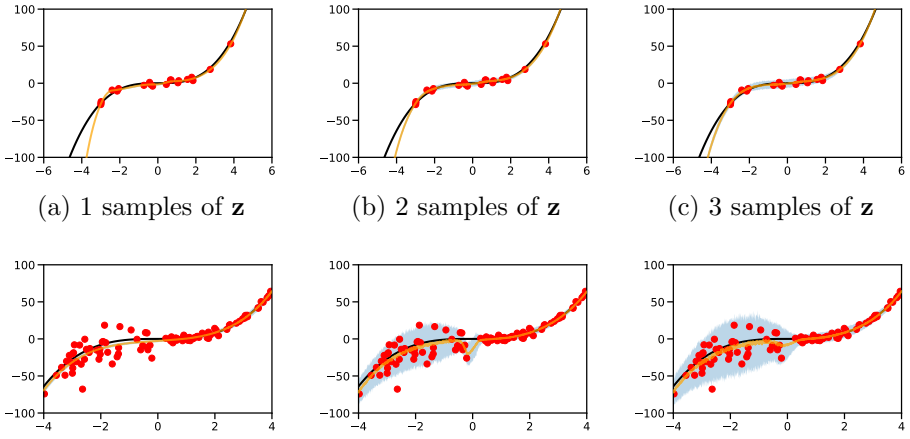
(epistemic) uncertainty in regions with no or only a few samples, e.g. between -6 and -4 or 4 and 6. The second toy regression dataset is constructed by sampling 100 data points as above, this time with different scale of noise in different intervals: $\epsilon \sim \mathcal{N}(0, 3^2)$, if $x \geq 0$ and $\epsilon \sim \mathcal{N}(0, 15^2)$, otherwise. This dataset is designated for testing whether a model can capture heteroscedastic aleatoric uncertainty.

In these experiments, we use a two-layer MLP with 100 hidden units as the predictive network, while the hypernetworks of the CDNs (g_1 and g_2) are modeled with two-layer MLPs with 10 hidden units each. Three samples of $\mathbf{z}$ (along with a single sample of $\boldsymbol{\psi}$ in the case of the VB-CDN) are used to approximate the objectives during training of both CDNs and BNNs. Regularization hyperparameter of $\lambda = 10^{-3}$ is used for training the ML-CDNs.

The results for the first data set (shown in Figure 4.1) demonstrate that the VB-CDN is capable of capturing the epistemic uncertainty like other Bayesian models. This is not the case for the ML-CDN (which displays high confidence everywhere) and the DE (which captures only the uncertainty on the left side). This demonstrates the benefits of using a Bayesian approach for capturing parameter uncertainty. On the other hand, the mixture models, i.e. the CDNs and the DE, are the only ones able to capture the aleatoric uncertainty on the second dataset, as shown in Figure 4.2. This can be explained by the ability of CDNs and DEs to model input-dependent variance.

To further investigate the different roles in uncertainty modeling of the mixing distribution and the approximate posterior of VB-CDNs, we compare their average variance, over the parameters and the input samples.[4] On the first data set, the average variance of the mixing distribution is 0.356 and that of the posterior distribution is 0.916. On the second data set, the average variance of the posterior distribution

[4]We picked 1000 evenly spaced points from $[-6, 6]$ and $[-4, 4]$ for the first and the second dataset, respectively, and approximated the means over the posterior with 100 samples.



(a) 1 samples of $\mathbf{z}$ (b) 2 samples of $\mathbf{z}$ (c) 3 samples of $\mathbf{z}$

(d) 1 samples of $\mathbf{z}$ (e) 2 samples of $\mathbf{z}$ (f) 3 samples of $\mathbf{z}$

Figure 4.3: Effect of number of training samples of $\mathbf{z}$ on the results of ML-CDNs on the toy datasets.

is 0.429 and that of the mixing distribution is 0.618 for $x < 0$ and 0.031 for $x \geq 0$. Therefore, the variance of the posterior is reduced on the second data set (as desired for more training data) while the mixing distribution successfully captures the higher data uncertainty for $x < 0$, indicating that the approximate posterior successfully models epistemic and the mixing distribution aleatoric uncertainty.

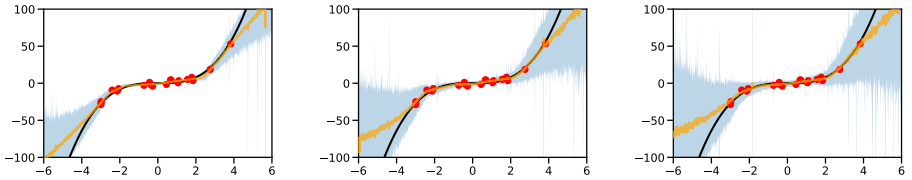
Finally, to give better understanding of the effect of using different number of samples of $\mathbf{z}$ in ML-CDNs' and VB-CDNs' training, we present the results for using $S \in \{1, 2, 3\}$ samples during training on the toy regression datasets in Figure 4.3 and Figure 4.4, respectively. We note that using larger values of S leads to an increase in the ability of CDNs to capture the aleatoric uncertainty of the data.

4.3 Out-of-distribution data

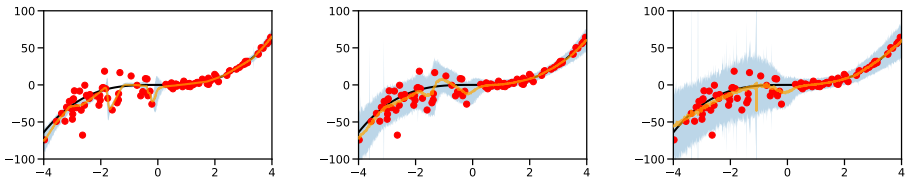
Following Lakshminarayanan et al. [2017], we train all models on the MNIST training set and investigate their performance on the MNIST test set and the notMNIST dataset[5], which contains images (of the same size and format as MNIST) of letters from the alphabet instead of handwritten digits. On such an OOD test set, the predictive distribution of an ideal model should have maximum entropy, i.e. it should have a value of $\ln 10 \approx 2.303$ which would be achieved if all ten classes are equally probable. The predictive NN used for this experiment is an MLP with a 784-100-10 architecture.

We present the results in Figure 4.5, where we plotted the empirical cumulative distribution function (CDF) of the empirical entropy of the predictive distribution, following Louizos and Welling [2017]. While one wishes to observe high confidence in data points similar to those seen during training, the model should express uncer-

[5] http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html.



(a) 1 samples of $\mathbf{z}$ (b) 2 samples of $\mathbf{z}$ (c) 3 samples of $\mathbf{z}$



(d) 1 samples of $\mathbf{z}$ (e) 2 samples of $\mathbf{z}$ (f) 3 samples of $\mathbf{z}$

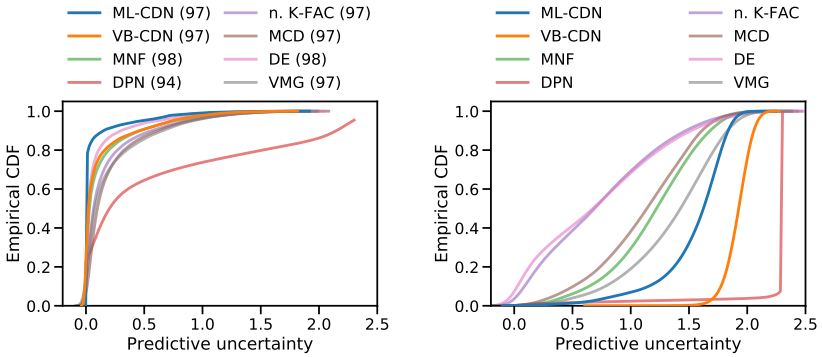
Figure 4.4: Effect of number of training samples of $\mathbf{z}$ on the results of VB-CDNs on the toy datasets.

tainty when exposed to OOD data. That is, we prefer a model to have a CDF curve closer to the bottom-right corner on notMNIST, as this implies it makes mostly uncertain (high entropy) predictions and a curve closer to the upper-left corner for MNIST, which indicates that it makes mostly confident (low entropy) predictions. As the results show, the VB-CDN yields high confidence on the test set of MNIST while having significantly lower confidence on notMNIST compared to all baseline models, except the DPN. Note, however, that training DPNs requires additional data (which makes the comparison unfair) and that the DPN's prediction accuracy and confidence on the MNIST test set are low compared to all other models. For the ML-CDN, we observe that it is more confident than all other models on within-distribution data, at the expense of showing lower uncertainty on OOD data than the VB-CDN. On the more challenging OOD task introduced by Alemi et al. [2018] where Fashion-MNIST [Xiao et al., 2017] is used as training set, while the vanilla and the up-down flipped test set of Fashion-MNIST are used for evaluation (Figure 4.6), the results are less pronounced, but the CDNs still show a performance competitive to that of the baseline models.

4.4 Adversarial attack

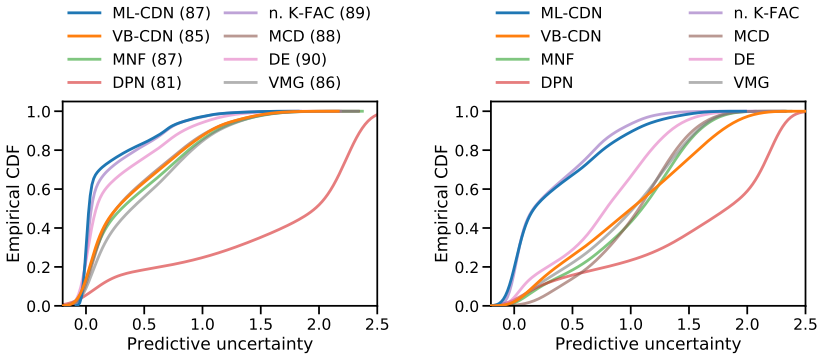
To investigate the robustness and detection performance of CDNs w.r.t. adversarial examples [Szegedy et al., 2014], we apply the Fast Gradient Sign Method (FGSM) [Goodfellow et al., 2015] to a 10% fraction (i.e. 1000 samples) of the MNIST, Fashion-MNIST [Xiao et al., 2017], and CIFAR-10 test set.[6] We do so, by making use of the implementation provided by Cleverhans [Papernot et al., 2018]. We employ a transfer learning scheme by using DenseNet-121 [Huang et al., 2017] trained

[6]We generate the adversarial examples based on a single forward-backward pass.



(a) MNIST (b) notMNIST

Figure 4.5: CDFs of the empirical entropy of the predictive distribution of the models trained on MNIST. Here the y-axis denotes the fraction of predictions having entropy less than the corresponding value on the x-axis. Confident models should have its CDF close to the top-left corner of the figure, while uncertain models to the bottom-right. The number next to each model name indicates its test accuracy.

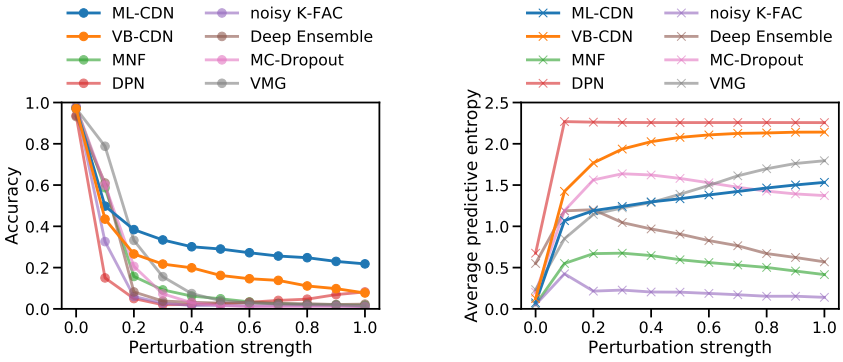


(a) Fashion-MNIST (b) Flipped Fashion-MNIST

Figure 4.6: CDFs of the empirical entropy of the predictive distribution of the models trained on Fashion-MNIST.

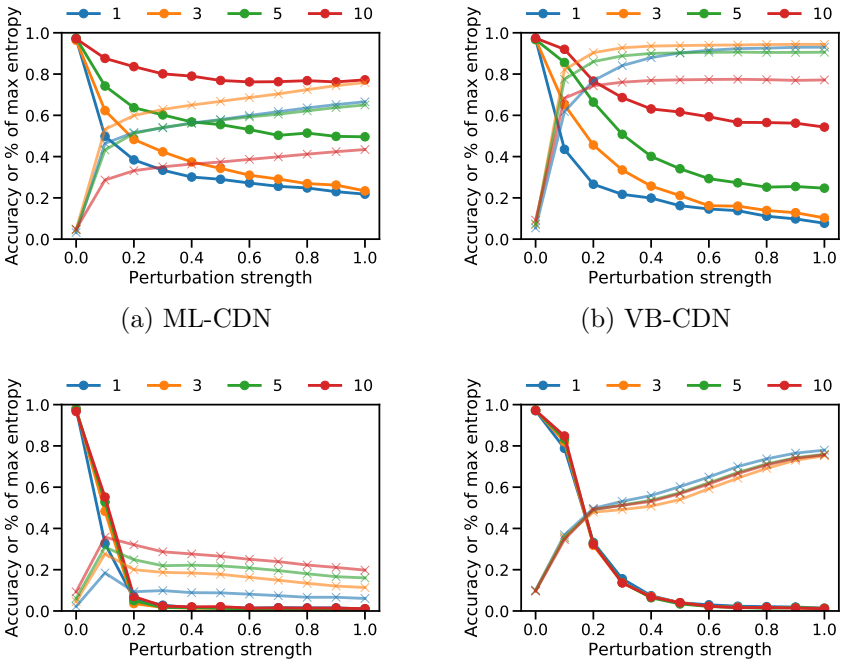
on ImageNet, as a fixed feature extractor for CIFAR-10. The predictive network for both all the datasets is a two-layer MLP with 100 hidden units. The probabilistic hypernetworks are two-layer MLPs with 50 hidden units. Note, that we do not use adversarial training when training the Deep Ensemble in this experiment to allow for a fair comparison.

MNIST Figure 4.7 presents the accuracy and the average empirical entropy of the predictive distribution w.r.t. adversarial examples for MNIST with varying levels of perturbation strength (between 0 and 1). We observe that the CDNs are more robust to adversarial examples than all baseline models. More specifically, the ML-CDN is significantly more robust in terms of accuracy to adversarial examples than



(a) Accuracy (b) Entropy

Figure 4.7: Prediction accuracy and average entropy of models trained on MNIST when attacked by FGSM-based adversarial examples [Goodfellow et al., 2015] with varying perturbation strength.

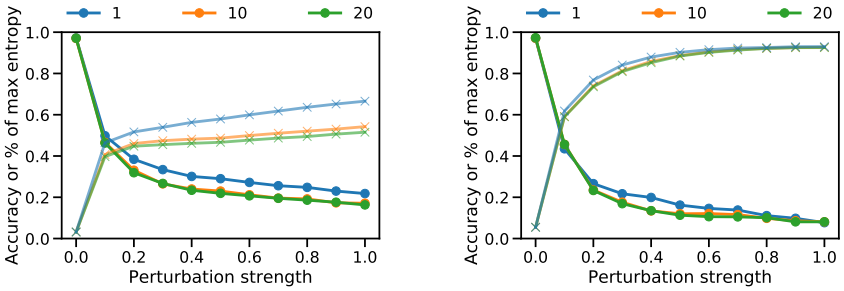


(a) ML-CDN (b) VB-CDN

(c) BNN (Noisy K-FAC) (d) BNN (VMG)

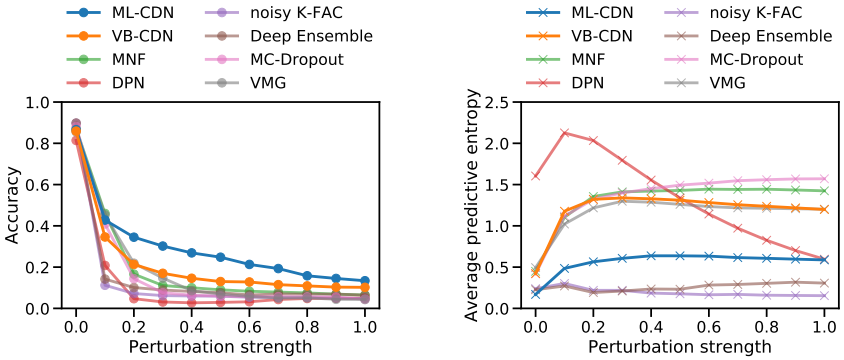
Figure 4.8: Accuracy and average entropy of CDNs and a BNN (Noisy K-FAC) under FGSM attack, with a varying number of samples of $\mathbf{z}$ used during training. Circles indicate accuracy, while crosses indicate entropy. The y-axis represents both the accuracy and the entropy relative to the maximum entropy (i.e. $\ln 10$).

all other models, while showing a competitive and nicely increasing entropy. The VB-CDN has only slightly better prediction accuracy but attains higher uncertainty than all the baselines except the DPN. Moreover, it shows uncertainties close to



(a) ML-CDN (b) VB-CDN

Figure 4.9: Prediction accuracy and average entropy of CDNs for stronger adversarial examples, constructed by averaging over multiple forward-backward passes.

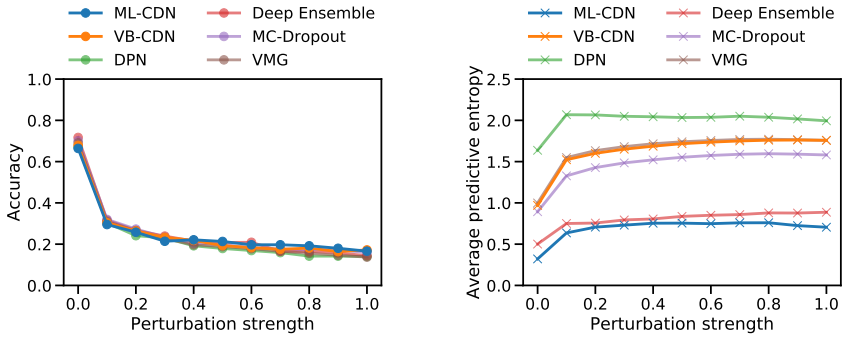


(a) Accuracy (b) Entropy

Figure 4.10: Prediction accuracy and average entropy of models trained on Fashion-MNIST when attacked by FGSM-based adversarial examples with varying perturbation strength.

that of the DPN, while having higher accuracy and without needing additional data during training. Furthermore, we found that using more samples of $\mathbf{z}$ during training is beneficial for the robustness of both ML-CDNs and VB-CDNs, as shown in Figure 4.8. This behavior is significantly more pronounced for CDNs than for BNNs (as exemplary shown for Noisy K-FAC and VMG). When using 10 samples per iteration during training the accuracy stays over 0.7 and 0.5 for ML-CDNs and VB-CDNs respectively, even for strong perturbations. As shown in Figure 4.9, even when the adversarial examples are stronger, i.e. estimated by averaging over multiple forward-backward passes, the performance of both CDNs is only marginally decreased (for VB-CDNs, it stays almost unchanged).

Fashion-MNIST The results on Fashion-MNIST are shown in Figure 4.10: Overall the same observations and conclusions can be made as for MNIST. We note that strangely, the DPN's uncertainty estimate is decreasing with increasing perturbation strength.



(a) Accuracy (b) Entropy

Figure 4.11: Prediction accuracy and average entropy of models trained on CIFAR-10 when attacked by FGSM-based adversarial examples with varying perturbation strength.

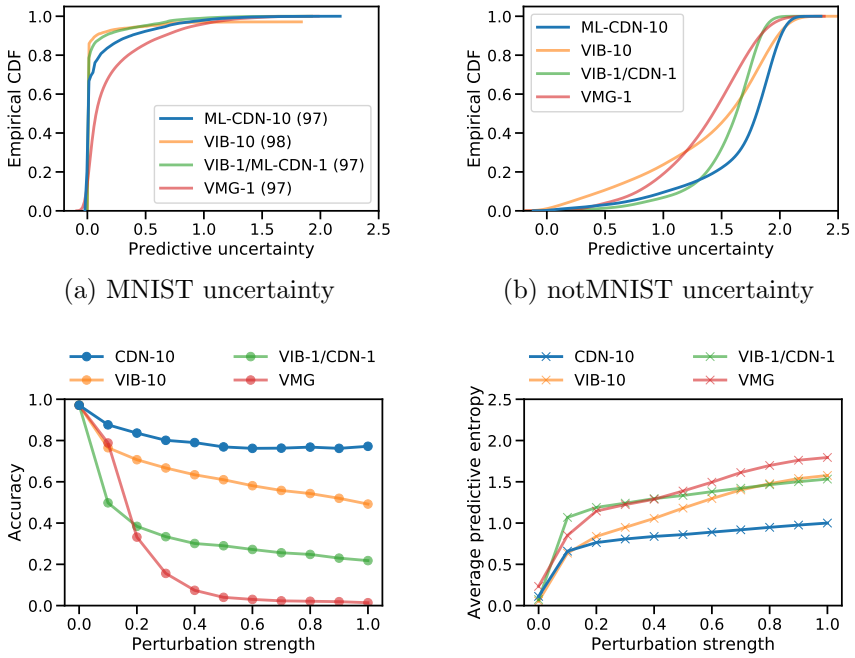
CIFAR-10 The results shown in Figure 4.11 demonstrate that the VB-CDN is competitive to other state-of-the-art models on CIFAR-10. The ML-CDN does not reflect uncertainty very well but has slightly higher accuracy than other models.

4.5 Comparison to training based on VIB objective

In this section, we experimentally compare the effects of optimizing our proposed maximum likelihood objective $\mathcal{L}_{\text{ML}}$ and following the VIB approach, which corresponds to optimizing the objective which results from exchanging the log with the expectation in $\mathcal{L}_{\text{ML}}$. Observe that they become equivalent when approximating the objective by a single sample of $\mathbf{z}$. To better analyze the effects of following the different objectives we, therefore, approximate them based on 10 samples. As a point of comparison, we also investigated the performance of the VMG [Louizos and Welling, 2016], which is a closely related BNN using an MVN distribution as the approximate (input independent) posterior.

Results for the OOD classification and the robustness to adversarial examples are shown in Figure 4.12. We observe that training ML-CDNs with 10 samples of $\mathbf{z}$ gives the best results in both experiments, the VMG gives the worst. The latter suggests that the input dependency of the distribution over $\mathbf{z}$ of CDNs and VIB plays a crucial role in the increased observed performance.

While for the robustness against adversarial attacks, the increased sample size improved the performance of models trained with the VIB as well as with the CDN objective, the model trained with the CDN objective clearly outperforms the others, reaching a surprisingly high accuracy about 0.8 even under huge perturbations.



(a) MNIST uncertainty

(b) notMNIST uncertainty

(c) MNIST adversarial accuracy

(d) MNIST adversarial entropy

Figure 4.12: Comparison of the effects of training the proposed model using the VIB and the maximum-likelihood objective of CDNs. "Objective-S" denotes that the objective was approximated based on S samples of $\mathbf{z}$ during training.

Chapter 5

Conclusion and Future Research

We have introduced compound density networks (CDNs), a novel class of models that allows for better uncertainty quantification in neural networks (NNs) and corresponds to a compound distribution (i.e. a mixture with uncountable components) in which both the component distribution and the input-dependent mixing distribution are parametrized by NNs. CDNs are inspired by the success of recently proposed ensemble methods [Osband et al., 2016; Lakshminarayanan et al., 2017] in predictive uncertainty quantification and represent a continuous generalization of mixture density networks [Bishop, 1994]. They can be implemented by using hypernetworks to map the input to a distribution over the parameters of the target NN, that models a predictive distribution. For training CDNs, regularized maximum likelihood or variational Bayes can be employed.

Extensive experimental analyses showed that CDNs are able to produce promising results in terms of uncertainty quantification. Specifically, Bayesian CDNs are able to capture epistemic as well as aleatoric uncertainty, and yield very high uncertainty on out-of-distribution samples while still making high confidence predictions on within-distribution samples. Furthermore, when facing FGSM-based adversarial attacks, the predictions of CDNs are significantly more robust in terms of accuracy than those of previous models. This robustness under adversarial attacks is especially pronounced for CDNs trained with a maximum-likelihood objective, but also clearly visible for Bayesian CDNs, which also provide a better chance of detecting the attack by showing increased uncertainty compared to the baselines. These promising experimental results indicate the benefits of applying a mixture model approach in conjunction with Bayesian inference for uncertainty quantification in NNs.

Given the exciting results that CDNs are able to achieve, several interesting future research is opened:

CDN model derived from residual networks While our focus in this thesis is on probabilistic hypernetworks, which is a natural CDN model that can be applied in multi-layer perceptrons (MLPs), one can also derive CDN model in residual networks (ResNets) [He et al., 2016] by assuming CDNs' indexing variable $\mathbf{z}_l$ to be the l-th

hidden units, the NN g_l to be the l-th residual block, and the interaction between each $\mathbf{z}_l$ and $\mathbf{h}_{l-1}$ is via point-wise addition. Given this assumption then, we can potentially scale-up CDNs into very deep networks.

Application of CDNs in other kinds of networks CDNs could hypothetically also be implemented in other kinds of networks, such as convolutional (CNNs) and recurrent networks (RNNs). This opens up a possibly better way of quantifying uncertainty in other tasks, such as sequence prediction, compared to previous methods.

More efficient parametrization of probabilistic hypernetworks It is always of our best interest to reduce the number of parameters necessary in our models, including CDNs. One can potentially gain insights from the previous works in hypernetworks, such as Pawlowski et al. [2017]; Sheikh et al. [2017, etc] for such parametrization.

Theoretical understanding of CDNs One could further analyze the theoretical properties of CDNs, for example in the robustness guarantee under adversarial attacks. Having a solid theoretical understanding, in conjunction with the strong empirical results we have shown in this thesis, will make practitioners more confident in applying CDNs in mission-critical applications.

Finally, it is the author's hope for the CDNs framework proposed in this thesis to be useful for machine learning scientists and practitioners alike.

Bibliography

Alemi, A., I. Fischer, J. Dillon, and K. Murphy
2017. Deep variational information bottleneck. In *ICLR*.

Alemi, A. A., I. Fischer, and J. V. Dillon
2018. Uncertainty in the variational information bottleneck. *arXiv preprint arXiv:1807.00906*.

Amari, S.-I.
1998. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276.

Amodei, D., C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané
2016. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*.

Ba, J. and B. Frey
2013. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, Pp. 3084–3092.

Bishop, C. M.
1994. Mixture density networks.

Bishop, C. M.
2006. *Pattern Recognition and Machine Learning*. Springer.

Blei, D. M., A. Y. Ng, and M. I. Jordan
2003. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Blundell, C., J. Cornebise, K. Kavukcuoglu, and D. Wierstra
2015. Weight uncertainty in neural networks. Pp. 1613–1622.

Carlini, N. and D. Wagner
2017. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, Pp. 39–57. IEEE.

Chen, T., E. Fox, and C. Guestrin
2014. Stochastic gradient hamiltonian monte carlo. In *International Conference on Machine Learning*, Pp. 1683–1691.

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio
2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa
2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Damianou, A. and N. Lawrence
2013. Deep gaussian processes. In *Artificial Intelligence and Statistics*, Pp. 207–215.

Dasgupta, S.
1999. Learning mixtures of gaussians. In *Foundations of computer science, 1999. 40th annual symposium on*, Pp. 634–644. IEEE.

Dawid, A. P.
1982. The well-calibrated bayesian. *Journal of the American Statistical Association*, 77(379):605–610.

DeGroot, M. H. and S. E. Fienberg
1983. The comparison and evaluation of forecasters. *The statistician*, Pp. 12–22.

Depeweg, S., J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft
2017. Learning and policy search in stochastic dynamical systems with bayesian neural networks. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2017)*.

Depeweg, S., J.-M. Hernandez-Lobato, F. Doshi-Velez, and S. Udluft
2018. Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. In *International Conference on Machine Learning*, Pp. 1192–1201.

Der Kiureghian, A. and O. Ditlevsen
2009. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112.

Duchi, J.
. Derivations for linear algebra and optimization.

Duchi, J., E. Hazan, and Y. Singer
2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Efron, B.
1992. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*, Pp. 569–593. Springer.

Gal, Y.
2016. Uncertainty in deep learning. *University of Cambridge*.

Gal, Y. and Z. Ghahramani
2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, Pp. 1050–1059.

Gneiting, T. and A. E. Raftery
2007. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio
2014. Generative adversarial nets. In *Advances in neural information processing systems*, Pp. 2672–2680.

Goodfellow, I., J. Shlens, and C. Szegedy
2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Graves, A.
2011. Practical Variational Inference for Neural Networks. In *Advances in Neural Information Processing Systems 24*, Pp. 2348–2356.

Guo, C., G. Pleiss, Y. Sun, and K. Q. Weinberger
2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, Pp. 1321–1330.

Gupta, A. K. and D. K. Nagar
1999. *Matrix variate distributions*. Chapman and Hall/CRC.

Ha, D., A. Dai, and Q. V. Le
2017. HyperNetworks. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2017)*.

He, K., X. Zhang, S. Ren, and J. Sun
2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, Pp. 770–778.

Hernández-Lobato, J. M. and R. Adams
2015. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, Pp. 1861–1869.

Hinton, G., N. Srivastava, and K. Swersky
2012. Rmsprop: Divide the gradient by a running average of its recent magnitude. *Neural networks for machine learning, Coursera lecture 6e*.

Hinton, G. E. and D. Van Camp
1993. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, Pp. 5–13. ACM.

Hochreiter, S. and J. Schmidhuber
1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Huang, G., Z. Liu, L. Van Der Maaten, and K. Q. Weinberger
2017. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Pp. 2261–2269. IEEE.

Ioffe, S. and C. Szegedy
2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, Pp. 448–456, Lille, France. PMLR.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton
1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Jia, X., B. De Brabandere, T. Tuytelaars, and L. V. Gool
2016. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, Pp. 667–675.

Jordan, M. I. and R. A. Jacobs
1994. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214.

Khan, M., D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava
2018. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, Pp. 2611–2620, Stockholmsmässan, Stockholm Sweden. PMLR.

Kingma, D. P. and J. Ba
2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference for Learning Representations*.

Kingma, D. P. and M. Welling
2014. Auto-encoding variational bayes. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton
2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, Pp. 1097–1105.

Krueger, D., C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville
2017. Bayesian Hypernetworks. *arXiv:1710.04759 [cs, stat]*. arXiv: 1710.04759.

Lakshminarayanan, B., A. Pritzel, and C. Blundell
2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, Pp. 6402–6413.

LeCun, Y. et al.
1989. Generalization and network design strategies. *Connectionism in perspective*, Pp. 143–155.

Lipton, Z. C., J. Berkowitz, and C. Elkan
2015. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*.

Louizos, C. and M. Welling
2016. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning*, Pp. 1708–1716.

Louizos, C. and M. Welling
2017. Multiplicative normalizing flows for variational Bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, Pp. 2218–2227.

MacKay, D. J.
1992. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472.

Malinin, A. and M. Gales
2018. Predictive uncertainty estimation via prior networks. *arXiv preprint arXiv:1802.10501*.

Martens, J. and R. Grosse
2015. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, Pp. 2408–2417.

Metropolis, N. and S. Ulam
1949. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341.

Murphy, K. P.
2012. *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Nair, V. and G. E. Hinton
2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, Pp. 807–814.

Neal, R. M.
1993. Bayesian learning via stochastic dynamics. In *Advances in neural information processing systems*, Pp. 475–482.

Neal, R. M.
1995. *BAYESIAN LEARNING FOR NEURAL NETWORKS*. PhD thesis, University of Toronto.

Neal, R. M. et al.
2011. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2.

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng
2011. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, P. 5.

Osband, I., C. Blundell, A. Pritzel, and B. Van Roy
2016. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, Pp. 4026–4034.

Papernot, N., F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, and R. Long
2018. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*.

Patterson, S. and Y. W. Teh
2013. Stochastic gradient riemannian langevin dynamics on the probability simplex. In *Advances in Neural Information Processing Systems*, Pp. 3102–3110.

Pawlowski, N., A. Brock, M. C. H. Lee, M. Rajchl, and B. Glocker
2017. Implicit Weight Uncertainty in Neural Networks. *arXiv:1711.01297 [cs, stat]*. arXiv: 1711.01297.

Peterson, C.
1987. A mean field theory learning algorithm for neural networks. *Complex systems*, 1:995–1019.

Platt, J. et al.
1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74.

Rezende, D. and S. Mohamed
2015. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, Pp. 1530–1538, Lille, France. PMLR.

Ritter, H., A. Botev, and D. Barber
2018. A scalable laplace approximation for neural networks. In *International Conference on Learning Representations*.

Robbins, H. and S. Monro
1951. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams
1986. Learning representations by back-propagating errors. *nature*, 323(6088):533.

Schmidhuber, J.
1992. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139.

Shafahi, A., W. R. Huang, C. Studer, S. Feizi, and T. Goldstein
2019. Are adversarial examples inevitable? In *International Conference on Learning Representations*.

Sheikh, A.-S., K. Rasul, A. Merentitis, and U. Bergmann
2017. Stochastic maximum likelihood optimization via hypernetworks. In *Advances in Neural Information Processing Systems*.

Simonyan, K. and A. Zisserman
2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov
2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Sun, S., C. Chen, and L. Carin
2017. Learning structured weight uncertainty in bayesian neural networks. In *Artificial Intelligence and Statistics*, Pp. 1283–1292.

Szegedy, C., W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus
2014. Intriguing properties of neural networks.

Tishby, N., F. C. Pereira, and W. Bialek
2001. The information bottleneck method. *Proceedings of the 37th Allerton Conference on Communication, Control and Computation*, 49.

Welling, M. and Y. W. Teh
2011. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, Pp. 681–688.

Xiao, H., K. Rasul, and R. Vollgraf
2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Zhang, G., S. Sun, D. Duvenaud, and R. Grosse
2018. Noisy natural gradient as variational inference. In *Proceedings of the 35th International Conference on Machine Learning*, Pp. 5852–5861.

Appendices

Appendix A

Other CDN models

In Chapter 3 of the main text of this thesis, we have constructed the probabilistic hypernetworks: a CDN model where the indexing variable $\mathbf{z}$ of the mixture density is assumed to be the weight matrices of the neural network f that parametrizes the mixture component distribution. However, we have observed that $\mathbf{z}$ could be any components of f: its weights, its inputs, or its hidden units.

In this supplementary chapter, as thought experiments, we will construct two additional CDN models where $\mathbf{z}$ is assumed to be the components of the hidden layers of f. In the first model, we assume $\mathbf{z}$ to be a learned multiplicative noise, akin to Gaussian dropout [Srivastava et al., 2014]. We will show that indeed, this model is the *adaptive dropout* model that is proposed by Ba and Frey [2013]. Thus, we can see that CDNs are the generalization of the adaptive dropout. The second model is constructed to make CDNs more scalable by employing *residual networks* (ResNets) architecture [He et al., 2016]. More specifically, we reinterpret the probabilistic hypernetwork g_l as the l-th residual block, and we assume $\mathbf{z}$ to be the hidden units of a ResNet. This corresponds to the probabilistic hypernetworks models where instead of using multiplication to do forward pass on f, we use addition.

A.1 Adaptive Gaussian dropout

Dropout [Srivastava et al., 2014] is a technique where random noise is used to corrupt the hidden activations of a neural network f and acts as stochastic regularization during training. Ba and Frey [2013] further proposed *adaptive dropout* to make the Bernoulli dropout noise context-dependent, i.e. a function of the input. In this section, we will reinterpret adaptive dropout as an instance of CDN. Specifically, we treat the adaptive dropout noise to be the CDNs' mixing variable $\mathbf{z}$ and plug it into the CDNs formula in eq. 3.2.

We construct the adaptive dropout as follows. Let $\mathbf{z}_l \sim p(\mathbf{z}_l; g_l(\mathbf{h}_{l-1}; \boldsymbol{\phi}_l))$ be the noise vector of layer l, with $\mathbf{z}_l \in \mathbb{R}^{k_l}$ where k_l is the dimension of the l-th hidden layer of f. We would like to multiplicatively apply this noise to the activation vector $\mathbf{h}_l$, i.e. $\mathbf{h}_l \odot \mathbf{z}_l$, where $\odot$ denotes the Hadamard product. Let $\mathbf{z} := \{\mathbf{z}_l\}_{l=1}^{L}$ and let

$g(\mathbf{x};\boldsymbol{\phi}) := \{g_l(\mathbf{h}_{l-1};\boldsymbol{\phi}_l)\}_{l=1}^L$. We define the mixing distribution to be

$$p(\mathbf{z}; g(\mathbf{x};\boldsymbol{\phi})) := \prod_{l=1}^{L} p(\mathbf{z}_l; g_l(\mathbf{h}_{l-1};\boldsymbol{\phi}_l))\,. \tag{A.1}$$

Note that each of the $p(\mathbf{z}_l; g(\mathbf{h}_{l-1};\boldsymbol{\phi}_l))$ can be any distribution as long as we can apply the reparametrization trick, for example

$$p(\mathbf{z}_l; g_l(\mathbf{h}_{l-1};\boldsymbol{\phi}_l)) := \mathcal{N}(\mathbf{z}_l; \mathbf{1}, \mathrm{diag}(g_l(\mathbf{h}_{l-1};\boldsymbol{\phi}_l)))\,, \tag{A.2}$$

which corresponds to the adaptive variant of Gaussian dropout [Srivastava et al., 2014], or the Gaussian counterpart of the method proposed by Ba and Frey [2013].

The above construction is sufficient to concretely define a CDN model. To train it, however, we need to specify the regularization distribution $p(\mathbf{z})$ for maximum-likelihood (ML) training (eq. 3.4) or the variational posterior and prior over the parameters for variational Bayes training (eq. 3.7). We will only show the procedure for ML training. Specifically, for adaptive Gaussian dropout with standard normal regularization distribution, the reparametrization trick is given by the following proposition.

Proposition A.1. *For all $l = 1, \ldots, L$, let $\boldsymbol{\sigma}_l^2 := g_l(\mathbf{h}_{l-1};\boldsymbol{\phi}_l)$ be a variance vector. For every layer l, it is sufficient for eq. A.2 that we sample $\mathbf{z}_l \in \mathbb{R}^k$, for some $k \in \mathbb{N}$, by*

$$\begin{aligned}\mathbf{e} &\sim \mathcal{N}(\mathbf{e}; \mathbf{0}, \mathbf{I}_k)\\ \mathbf{z}_l &= \mathbf{1} + \boldsymbol{\sigma}_l \odot \mathbf{e}\,.\end{aligned}$$

Proof. We use the fact that Gaussian is closed under affine transformation. For example, suppose $\mathbf{x} \sim \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ then $\mathbf{A}\mathbf{x} + \boldsymbol{b} =: \mathbf{y} \sim \mathcal{N}(\mathbf{y}; \mathbf{A}\boldsymbol{\mu} + \boldsymbol{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^{\mathrm{T}})$. Thus, it follows from the claim that

$$\mathbf{z}_l = \mathbf{1} + \mathrm{diag}(\boldsymbol{\sigma}_l)\,\mathbf{e} \sim \mathcal{N}(\mathbf{z}_l; \mathbf{1}, \mathrm{diag}(\boldsymbol{\sigma}_l)\mathbf{I}_k\mathrm{diag}(\boldsymbol{\sigma}_l)) = \mathcal{N}(\mathbf{z}_l; \mathbf{1}, \mathrm{diag}(\boldsymbol{\sigma}_l^2))\,.$$

This implies that $\mathbf{z}_l$ is distributed according to eq. A.2. □

Meanwhile, the KL-divergence regularization term is summarized as follows.

Proposition A.2. *For all $l = 1, \ldots, L$, let $\mathbf{z}_l \in \mathbb{R}^k$ for some $k \in \mathbb{N}$ and $\boldsymbol{\sigma}_l^2 := g_l(\mathbf{h}_{l-1};\boldsymbol{\phi}_l)$. Given eq. A.2 and $p(\mathbf{z}) := \prod_{l=1}^L p(\mathbf{z}_l) := \prod_{l=1}^L \mathcal{N}(\mathbf{0}, \mathbf{I}_k)$, the KL-divergence between them is given by*

$$D_{\mathrm{KL}}[\,p(\mathbf{z}_l|g_l(\mathbf{h}_{l-1};\boldsymbol{\psi}_l))\|p(\mathbf{z}_l)] = \frac{1}{2}\sum_{i=1}^{k} \sigma_{li}^2 - \log\sigma_{li}^2\,.$$

Proof. Following Kingma and Welling [2014, Appendix B], the KL-divergence of $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I})$ with $\mathcal{N}(\mathbf{0}, \mathbf{I})$ is given by

$$D_{\mathrm{KL}}[\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I})\|\,\mathcal{N}(\mathbf{0}, \mathbf{I})] = \frac{1}{2}\sum_{i=1}^{k} \mu_i^2 + \sigma_i^2 - \log\sigma_i^2 - 1\,.$$

By assumption, $\boldsymbol{\mu} = \mathbf{1}$, thus

$$D_{\mathrm{KL}}[\mathcal{N}(\mathbf{1}, \boldsymbol{\sigma}^2\mathbf{I}) \| \mathcal{N}(\mathbf{0}, \mathbf{I})] = \frac{1}{2}\sum_{i=1}^{k} \sigma_i^2 - \log \sigma_i^2 \,,$$

and the claim follows. □

Finally, as we have seen, based on Corollary 3.8, the overall KL-divergence term is just the sum of the L individual KL-divergence terms.

A.2 Probabilistic ResNets

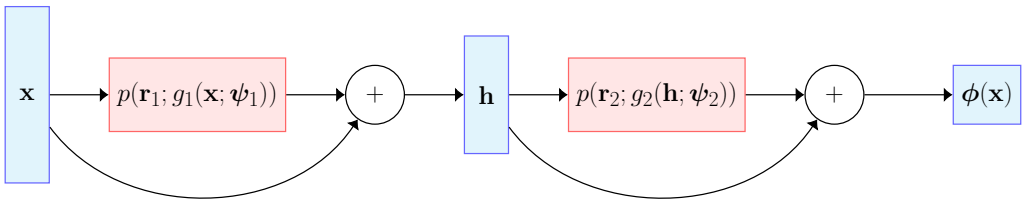


Figure A.1: An example of probabilistic ResNets with two residual blocks g_1, g_2.

Recall that in probabilistic hypernetworks introduced in the main text of this paper, to compute the l-th hidden units $\mathbf{h}_l$, we compute

$$\mathbf{W}_l \sim p(\mathbf{W}_l; g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l)); \; \mathbf{h}_l = \mathbf{W}_l^{\mathrm{T}} \mathbf{h}_{l-1} \,. \tag{A.3}$$

That is, $\mathbf{h}_l$ is attained by multiplying the l-th mixing variables $\mathbf{W}_l \in \mathbf{z}$ with the input of the l-th layer, $\mathbf{h}_{l-1}$.

Let us now write $\mathbf{W}_l$ as $\mathbf{r}_l$ and assume it to have the same dimensionality as $\mathbf{h}_{l-1}$. Furthermore, we replace the multiplication in eq. A.3 with addition, yielding

$$\mathbf{r}_l \sim p(\mathbf{r}_l; g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l)); \; \mathbf{h}_l = \mathbf{r}_l + \mathbf{h}_{l-1} \,. \tag{A.4}$$

Observe that g_l can be seen as an arbitrary NN, which we might choose as in the residual block of ResNets [He et al., 2016]. However, instead of outputting some deterministic variables, we define g_l to output the mean and variance parameters of a fully-factorized Gaussian, from which $\mathbf{r}_l$ can be sampled. Thus, we define $p(\mathbf{r}_l; g_l(\mathbf{h}_{l-1}; \boldsymbol{\psi}_l)) := \mathcal{N}(\mathbf{r}_l; \boldsymbol{\mu}_l, \mathrm{diag}(\boldsymbol{\sigma}_l^2))$. We call g_l the l-th *probabilistic residual block*. As in ResNets, we can thus use a collection of probabilistic residual blocks to build an NN. We call the resulting NNs *probabilistic ResNets*. We illustrate the probabilistic ResNets in Figure A.1.

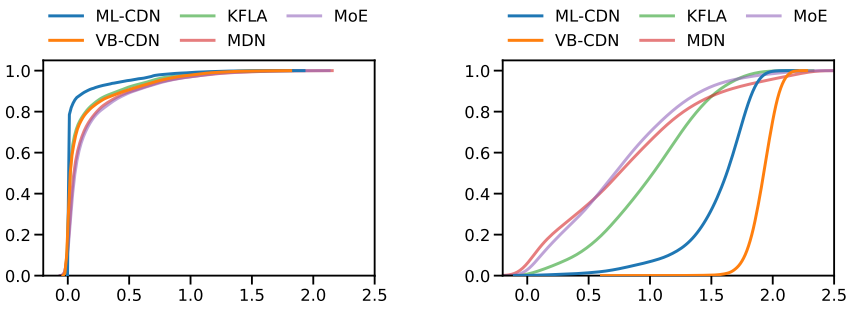
Notice that the random variables $\mathbf{z}$ that are modeled by the CDN's mixture component are now have the same dimension as the hidden units. Furthermore, we only introduce an additional layer to output also the variance of $\mathbf{r}_l$. Therefore, the probabilistic ResNets only introduce small overhead over the vanilla ResNets thus share the scalability property of the vanilla ResNets. This is crucial, as this allows CDNs to be applied in large-scale models with large-scale datasets.

Appendix B

Supplementary Experimental Results

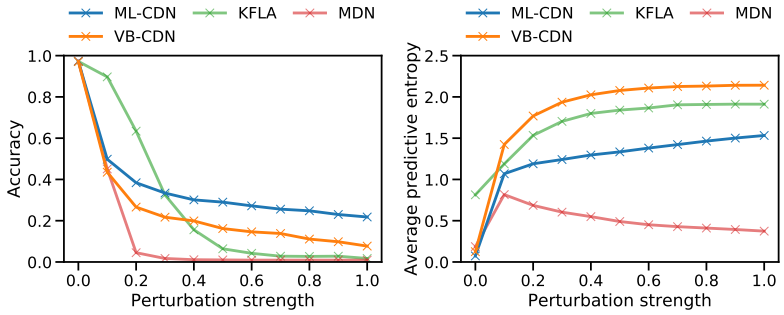
B.1 Experimental results for additional baseline models

In this section, we compare the CDNs described in the main text with the Kronecker-factored Laplace approximation (KFLA) proposed by Ritter et al. [2018], the mixture of experts (MoE) and the MDN models on the MNIST dataset. For both MDN and MoE, we use two-layer MLP with 100 hidden units, with 5 mixture components. Specifically for MoE, the mixing distribution is also given by another NN of the same architecture. The results for the out-of-distribution prediction and adversarial examples are presented in Figure B.1 and Figure B.2, respectively. Note that the results are in line with the conclusions drawn in comparison with respect to other baseline models in the main text.



(a) MNIST (b) notMNIST

Figure B.1: CDF of the prediction entropy on MNIST and notMNIST test set of the additional models.

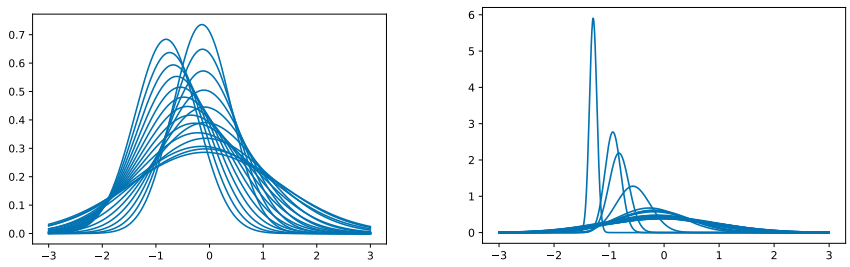


(a) Accuracy (b) Entropy

Figure B.2: Prediction accuracy and average entropy of models trained on MNIST when attacked by FGSM-based adversarial examples [Goodfellow et al., 2015] with varying perturbation strength.

B.2 Visualization of the learned mixing distribution

To further understand the effect of conditioning the distribution over $\mathbf{z}$ (i.e. the mixing distribution) we compute the mixing distribution $p(\mathbf{z}; g(\mathbf{x}_i; \boldsymbol{\psi}))$ for a set of samples $\mathbf{x}_1, \dots \mathbf{x}_n$ and ML-CDN trained on the heteroscedastic cubic regression dataset. The results for two randomly selected weights $w_{li} \in \mathbf{W}_l$ are shown in Figure B.3. We found that the mean and the variance of the marginal distribution (which is Gaussian due to our model) varies depending on the value of the input $\mathbf{x}$ indicating that different mixture components get high probabilities for different inputs.

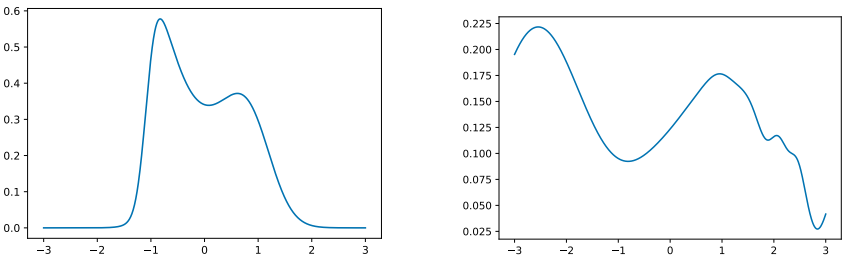


(a) (b)

Figure B.3: Visualization of the distribution $p(w_{li}; g(\mathbf{x}; \boldsymbol{\psi}))$ of a randomly selected weight $w_{li} \in \mathbf{W}_l$ for different samples of input $\mathbf{x}$ from the heteroscedastic toy dataset. w_{li} denotes the i-th weight of the l-th layer of f.

Furthermore, to show that CDNs are able to capture multimodality in weight space, we train a probabilistic hypernetworks model with a 5 hidden units mixture component on a toy classification dataset that is constructed as follows: we sample an input $\mathbf{x}$ from a mixture of Gaussian $p(\mathbf{x}) = \frac{1}{2}\mathcal{N}(-3, 1) + \frac{1}{2}\mathcal{N}(3, 1)$, and assign

a label depending whether it comes from the first ($\mathbf{y} = 0$) or the second Gaussian ($\mathbf{y} = 1$). To evaluate the resulting distribution, we marginalize the mixing distribution $p(\mathbf{z}|g(\mathbf{x};\boldsymbol{\psi}))$ w.r.t. $\mathbf{x}$, i.e. we evaluate $p(\mathbf{z}) = \int p(\mathbf{z}|g(\mathbf{x};\boldsymbol{\psi}))p(\mathbf{x})\,\mathrm{d}\mathbf{x}$. The resulting distribution for two randomly selected weights $w_{li} \in \mathbf{W}_l$ are shown in Figure B.4 below. We observe that indeed our model can learn a multimodal weight distribution.



(a) (b)

Figure B.4: Visualization of the marginal distribution $p(w_{li}) = \int p(w_{li}; g(\mathbf{x};\boldsymbol{\psi}))p(\mathbf{x})\,\mathrm{d}\mathbf{x}$ for two randomly selected weights w_{li} of a CDN trained on a toy classification dataset.