
Measure-Valued Derivatives for Machine Learning

Measure-Valued Derivatives für Maschinelles Lernen

Master thesis by Mattis Manfred Kaemmerer

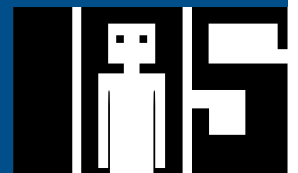
Date of submission: 31.08.2021

1. Review: João Carvalho, M.Sc.

2. Review: Prof. Dr. Jan Peters
Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Mattis Manfred Kaemmerer, die vorliegende Masterarbeit gemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 31.08.2021

M. Kaemmerer

Contents

1	Introduction	1
1.1	Notation	2
1.2	Probabilistic Objectives	2
1.3	Related Work	6
2	Gradient Estimators	7
2.1	Pathwise Gradient Estimator	7
2.2	Score-Function Gradient Estimator	10
2.3	Measure-Valued Derivative	13
2.4	Convex Combination of Estimators	18
2.5	Gaussian Mixture Models	19
3	A Framework for Monte-Carlo Gradient Estimators	22
3.1	Framework Description	22
3.2	Extensibility of the Framework	24
3.3	Bayesian Logistic Regression	25
3.4	Variational Auto-Encoders	26
4	Gradient Analysis	29
4.1	Bayesian Logistic Regression	29
4.2	Variational Auto-Encoder	30
4.3	Randomized Convex Combinations	38
4.4	Discrete Distributions	41
5	Discussion	55
5.1	The Expected Advantage of the Pathwise Estimator	55
5.2	Issues of the Score-Function Estimator	56
5.3	Potential of the Measure-Valued Estimator	57

Figures and Tables

List of Figures

- 1.1 Simple computation graph when trying to backpropagate the gradient through a probabilistic node p . Since we cannot directly compute $\partial z/\partial\theta$, the backpropagation ends at p . In this graph, $z \sim p(z; \theta)$ is the latent variable, and f is the cost. We omit the subgraph of the direct dependency of $f(z; \theta)$ on θ , since it does not impact this computation. 6
 - 2.1 Computational graph of the pathwise gradient estimator. The new deterministic g node replaces p , and allows the gradient to flow through the whole path. In this graph, $z = g(\epsilon; \theta)$, where $\epsilon \sim q(\epsilon)$, is the latent variable, and f is the cost. We omit the subgraph of the direct dependency of $f(z; \theta)$ on θ , since it does not impact this computation. 9
 - 2.2 Computational graph of the score-function gradient estimator. We sample from p , which is not a problem, because we do not require its gradient for the computation of $f(z)\nabla_{\theta} \log p(z)$. In this graph, $z \sim p(z; \theta)$ is the latent variable, and f is the cost. We omit the subgraph of the direct dependency of $f(z; \theta)$ on θ , since it does not impact this computation. 11
 - 2.3 Computational graph of the measure-valued gradient estimator. We sample from the two components p_i^+ and p_i^- of the decomposition, evaluate f , and multiply the difference of the results by the normalization factor θ . In this graph, $z \sim p(z; \theta)$ is the latent variable, and f is the cost. We omit the subgraph of the direct dependency of $f(z; \theta)$ on θ_i , since it does not impact this computation. 14
-



2.4	Probability densities for the measure-valued derivative decomposition for the gradient $\partial\mathcal{N}/\partial\mu$ of the univariate standard normal distribution $\mathcal{N}(0, 1)$, where $p^+(\mu, \sigma^2) = \mu + \sigma\mathcal{W}(2, 0.5)$, and $p^-(\mu, \sigma^2) = \mu - \sigma\mathcal{W}(2, 0.5)$	16
2.5	Probability densities for the measure-valued derivative decomposition for the gradient $\partial\mathcal{N}/\partial\sigma$ of the univariate standard normal distribution $\mathcal{N}(0, 1)$, where $p^+(\mu, \sigma^2) = \mathcal{M}(0, 1)$, and $p^-(\mu, \sigma^2) = \mathcal{N}(0, 1)$	17
3.1	Typical implementation of VAEs in PyTorch, omitting boilerplate code. . .	24
3.2	A high-level representation of the basic variational auto-encoder architecture. The encoder and decoder are neural networks. The decoder is trained directly through backpropagating the losses, but we require a gradient estimator for the encoder gradients. In this figure, we call the data $\mathbf{x} \in \mathbb{R}^n$, the latent representation $\mathbf{z} \sim p(\mathbf{z}; \text{Encoder}(\mathbf{x}))$, $\mathbf{z} \in \mathbb{R}^m$, and the reconstruction $\text{Decoder}(\mathbf{z}) = \tilde{\mathbf{x}}$, $\tilde{\mathbf{x}} \in \mathbb{R}^n$	27
3.3	VAE implementation using the framework of this thesis, omitting boilerplate code. In contrast to figure 3.1, the loss function in this case only calculates the binary cross-entropy loss.	28
4.1	Variances of Bayesian Logistic Regression with $m = 30$ latent dimensions, trained on the breast cancer dataset [10]. The optimizer used is SGD with learning rate $\alpha = 10^{-3}$ and batch size $B = 32$	30
4.2	Train and test losses of Bayesian Logistic Regression with $m = 30$ latent dimensions, trained on the breast cancer dataset [10]. The optimizer used is SGD with learning rate $\alpha = 10^{-3}$ and batch size $B = 32$	31
4.3	Train and test losses of a VAE with two fully connected 400-dimensional layers in both the encoder and the decoder with ReLU activations, $m = 20$ latent dimensions, trained on the MNIST dataset. The optimizer used is ADAM with learning rate $\alpha = 10^{-3}$ and batch size $B = 128$	33
4.4	Reconstructions by VAEs generated after the training shown in figure 4.3. Here, MVD means the measure-valued estimator, PD means the pathwise estimator, and SF means the score-function estimator. All reconstructions are generated using models trained with the highest number of samples. .	34





4.5	Training loss of a VAE with two fully connected 400-dimensional layers in both the encoder and the decoder with ReLU activations, $m = 20$ latent dimensions, trained on the MNIST dataset. The optimizer used is ADAM with learning rate $\alpha = 10^{-3}$ and batch size $B = 128$. The tracked time in the horizontal axis does not correspond to a real-time measure, but reflects the relative processing time required by the gradient estimator computations only. At the end of the graph, the pathwise estimator with 1 sample completed 10 epochs, while the others are cut off at that point. . .	35
4.6	Train and test losses of a VAE with convolutional networks, $m = 64$ latent dimensions, trained on the Omniglot dataset. The optimizer used is ADAM with learning rate $\alpha = 10^{-3}$ and batch size $B = 144$	36
4.7	Variances of the estimators while training VAEs on MNIST and Omniglot datasets. The variances are computed during training, using the same experiment set-up as figures 4.3 and 4.6 respectively. The y-axis of both graphs is shown in logarithmic scale.	37
4.8	Train and test losses of a VAE with recurrent LSTM, GRU, and vanilla RNN, $m = 20$ latent dimensions, trained on the ECG5000 dataset . The optimizer used is ADAM with learning rate $\alpha = 5 * 10^{-4}$	39
4.9	Train and test losses of a VAE with recurrent LSTM, GRU, and vanilla RNN, $m = 4$ latent dimensions, trained on the synthetic sinusoidal dataset . The optimizer used is ADAM with learning rate $\alpha = 5 * 10^{-4}$	40
4.10	Losses of training varying combinations of measure-valued and score-function (MVSF) estimators. The hyperparameters as well as the model are the same as in figure 4.3. We only vary how many dimensions of the score-function estimate are randomly replaced by a measure-valued estimate. The estimators are set up such that they use the same number of loss evaluations, which in this case are always 80 in total. In the legend, a MV / b SF stands for a dimensions of measure-valued gradient estimates, and b dimensions of score-function gradient estimates.	42





4.11 Losses of training varying sample sized of the randomized, convex MVSF combination estimator. In this figure, all MVSF combinations replace 10 of 40 parameter dimensions by measure-valued estimates. **Both plots show the same data**, but with different color schemes. In the second plot, we show the MVSF estimator in red and the pure SF estimator in blue to be able to directly compare the estimator types. The hyperparameters as well as the model are the same as in figure 4.3. The estimators are set up such that they use the same number of loss evaluations, which in this case are always 80 in total. 43

4.12 Probability mass functions while training a categorical distribution with $k = 3$, using the SGD optimizer with learning rate $\alpha = 0.1$, using softmax parameterization. The loss function is a simple, shifted parabola $f(x) = (x - 0.5)^2$, shown as a dashed blue line. We train for 100 iterations, plotting every 20 iterations from top to bottom. 45

4.13 Probability mass functions while training a categorical distribution with $k = 100$, learning rate $\alpha = 0.1$, using softmax parameterization. The loss function is a simple, shifted parabola $f(x) = (x - 0.5)^2$, shown as a dashed blue line. We train for 1000 iterations, plotting every 200 iterations from top to bottom. Since the density of categories is so high here, we plot them as lines, even though they are still discrete. 47

4.14 Evaluating different Gumbel softmax temperatures τ training a categorical distribution with $k = 100$, learning rate $\alpha = 0.1$, using softmax parameterization. The loss function is a simple, shifted parabola $f(x) = (x - 0.5)^2$, shown as a dashed blue line. We train for 1000 iterations, plotting every 200 iterations from top to bottom. Since the density of categories is so high here, we plot them as lines, even though they are still discrete. 48

4.15 Probability mass functions while training a categorical distribution with $k = 100$, learning rate $\alpha = 0.1$, using softmax parameterization. The loss function is a sine $f(x) = \sin(4\pi x)$, shown as a dashed blue line. The loss function is shifted and squished such that we can see the extrema in the plot, but it was unaltered during training. We train for 1000 iterations, plotting every 200 iterations from top to bottom. Since the density of categories is so high here, we plot them as lines, even though they are still discrete. 49





4.16	Approximate expected values while training a categorical distribution with the same set-up as in figure 4.15. We initialize the class probabilities equally as $1/k$	50
4.17	Approximate expected values while training a categorical distribution with the same set-up as in figure 4.15. For this training, we initialize the probabilities randomly.	51
4.18	Approximate expected values while training a Poisson distribution with a shifted parabola loss. The measure-valued estimator using 1 sample, i.e., 2 loss evaluations, while the score-function estimator uses 2 samples for the same number of loss evaluations. Learning rate is $\alpha = 0.01$, training $\ln \lambda$	53
4.19	Probability mass functions while training a Poisson distribution with the <i>score-function estimator</i> using 2 samples. We only vary the initial rates, learning rate is $\alpha = 0.1$, training $\ln \lambda$. The learning rate is too high in this case, hence the estimator overshoots and gets stuck in an extremely skewed distribution.	54
4.20	Probability mass functions while training a Poisson distribution with the <i>measure-valued estimator</i> using 1 sample, i.e., 2 loss evaluations. We only vary the initial rates, learning rate is $\alpha = 0.1$, training $\ln \lambda$. Unlike the score-function estimator in figure 4.19, the measure-valued estimator reliably finds a solution, independent of starting parameters.	54

List of Tables

1.1	Common distributions, their notation, and probability density/mass functions. We list the univariate versions here, because we assume factorizable distributions for our evaluations.	3
2.1	Reparameterizations for some common univariate distributions. We sample from the base distribution $q(\epsilon)$, and transform the samples with $g(\epsilon; \theta)$. $\mathcal{U}(a, b)$ denotes the uniform distribution.	10
2.2	Measure-valued derivative triples (c_θ, p^+, p^-) to estimate $\nabla_\theta p$ for common distributions. See Table 1.1 for information on the distributions.	18

Abstract

In this thesis, we analyze the *measure-valued* derivative as an unbiased gradient estimator for stochastic models.

We compare it to two other approaches, the *pathwise* derivative, and the *score-function* method. We give a formal introduction to the topic by defining the general probabilistic objective, as well as the estimators. Then, we develop a Python framework based on PyTorch, which supports extensible modules for datasets, models, probability distribution families, and gradient estimators.

Using our framework, we conduct several experiments using approaches such as Bayesian logistic regression, and variational auto-encoders. To find new use cases for the measure-valued estimator, we explore some novel approaches, such as a convex combination of estimators, and applying it to discrete mixture models. We design these experiments to reveal some of the characteristics of the three estimators, focusing more on the measure-valued estimator. Based on these results, we discuss the implications on the usefulness of the measure-valued estimator. Also, we show some of the advantages and disadvantages of the other estimators, and find rules on when which estimator should be applied.

We find that the measure-valued estimator shows potential in many use cases, and performs very well compared to the score-function estimator.

Zusammenfassung

In dieser Thesis analysieren wir die *Measure-Valued*-Derivation als erwartungstreuen Gradientenschätzer für stochastische Modelle.

Wir vergleichen das *Measure-Valued*-Derivativ mit zwei anderen Ansätzen, dem *Pathwise*-Derivativ und dem *Score-Function*-Derivativ. Wir definieren den generellen, probabilistischen Richtwert und geben eine formale Einführung, sowohl in die Problemstellung als auch in die Schätzer. Dann entwickeln wir ein Python Framework, basierend auf PyTorch, welches erweiterbare Module für Datensätze, Familien von Wahrscheinlichkeitsverteilungen und Gradientenschätzern unterstützt.

Mit unserem Framework führen wir einige Experimente durch, in denen wir Ansätze wie Bayes'sche logistische Regression und Variational Auto-Encoder verwenden. Um neue Anwendungsfälle für den *Measure-Valued*-Schätzer zu finden, erforschen wir einige Ansätze, wie zum Beispiel eine konvexe Kombination von Schätzern, oder ihn auf diskrete Mischverteilungen anzuwenden. Wir entwerfen diese Experimente so, dass wir einige Charakteristiken der drei Schätzverfahren aufzeigen können und fokussieren uns dabei auf den *Measure-Valued*-Schätzer. Basierend auf diesen Ergebnissen diskutieren wir die Implikationen für die Nützlichkeit des *Measure-Valued*-Schätzers. Außerdem zeigen wir einige Vor- und Nachteile der anderen Schätzer auf und finden Regeln, wann welcher Schätzer verwendet werden sollte.

Wir stellen fest, dass der *Measure-Valued*-Schätzer Potenzial in vielen Anwendungen aufweist und besonders im Vergleich zum *Score-Function*-Schätzer sehr gute Ergebnisse liefert.

1 Introduction

In this thesis, we evaluate the measure-valued derivative (MVD) [46, 52] against the more commonly used pathwise derivative (PD) [52] and the score-function derivative (SF) [15, 58]. All of these algorithms are used for estimating the parameter gradient of models, which contain a probabilistic component. We focus on the MVD, since it is the least explored estimator of the three [46], and is sufficiently distinct to warrant further exploration. The rest of this chapter contains a detailed explanation of the problem formulation, as well as the definitions of the three estimators.

Before going into formal details, this section introduces the task on a higher level, and is supposed to build some context for what is shown in the following sections. For more details refer to these respective sections. Since the models we focus on are by design non-deterministic, i.e., produce different outputs for the same input, we cannot directly estimate their gradients. Instead, we model the probabilistic component as a random variable, and optimize the parameters using the gradient of its probability density. For some specific cases, this leads to a closed-form solution, but, in general, we have to rely on stochastic gradient optimization [5]. However, since our models produce probabilistic outcomes, we can not directly optimize the objective value, but only its expected value given the probabilistic nature of the model. This is the case, because the probabilistic component depends on the model parameters that we are optimizing. In practice, estimating an expectation can be achieved via Monte-Carlo estimation [45]. Alas, once we formulate our problem as finding a stochastic estimate of the gradient of the expectation, we see that the gradient of an expected value is not directly an expected value itself. If it was possible to rearrange the gradient in such a way that we could express it as an expected value, then we could apply Monte-Carlo estimation again. The approaches we compare in this thesis all focus on this task. Given that we applied one of these approaches, we can estimate the parameter gradients of the probabilistic components' parameters.

Here is an outline of the contents of the thesis. In chapter 2, we derive the three approaches and discuss them in detail. In chapter 3, we present how we implement the approaches to

evaluate them on practical tasks. We build a framework based on PyTorch [49], which allows us to represent probability distributions as probabilistic nodes, and easily extend the set of available distributions to be used by all three approaches. Chapter 4 shows the results of our evaluations. Using Bayesian logistic regression, and variational auto-encoders, we compare the approaches in various situations, and analyze their characteristics on practical experiments. In chapter 5, we discuss the results and draw some conclusions based on our findings. We work out where the measure-valued derivative should be considered as a valuable alternative, and how we take the most advantage from its benefits. Also, we outline some options for future research to explore possible improvements to the algorithm. At last, we give a high-level overview of our findings in chapter 6.

1.1 Notation

Throughout the thesis, bold letters like \mathbf{x} and \mathbf{z} represent vectors. We write the probability $p(X = \mathbf{x})$ in the shorter form $p(\mathbf{x})$, and write $p(\mathbf{x}; \theta)$ for the distribution over random vectors \mathbf{x} with distributional parameters $\theta \in \mathbb{R}^D$. When we sampled $\mathbf{x} \sim p(\mathbf{x})$, we assume $\mathbf{x} \in \mathcal{S}$, where \mathcal{S} is the support of the distribution, i.e., the set of possible outcomes. Furthermore, we write ∇_{θ} to indicate a vector of the partial derivatives of the respective elements of θ , i.e.

$$\nabla_{\theta} f = \left[\frac{\partial f}{\partial \theta_1}, \dots, \frac{\partial f}{\partial \theta_D} \right]^T.$$

As a short form for $\mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}; \theta)} [\dots]$, we write $\mathbb{E}_{p(\mathbf{x}; \theta)} [\dots]$.

1.2 Probabilistic Objectives

Many machine learning problems, e.g., variational inference [6], Bayesian logistic regression, or variational auto-encoders, require a probability distribution in their models. In these cases, we write the probability distribution as $\mathbf{x} \sim p(\mathbf{x}; \theta)$, where θ are the parameters of the distribution. We also define a cost function $f(\mathbf{x}; \theta)$, which is a deterministic objective function, and does not have to depend on θ , but will often contain some form of regularization. This function depends on the problem, e.g., variational inference uses the

Table 1.1: Common distributions, their notation, and probability density/mass functions. We list the univariate versions here, because we assume factorizable distributions for our evaluations.

Name	Support	Notation	Probability Density/Mass Function
Uniform	\mathbb{R}	$\mathcal{U}(x; a, b)$	$\frac{1}{b-a}$ for $x \in [a, b]$ otherwise 0
Normal	\mathbb{R}	$\mathcal{N}(x; \mu, \sigma^2)$	$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$
Double-sided Maxwell	\mathbb{R}	$\mathcal{M}(x; \mu, \sigma^2)$	$\frac{1}{\sigma^3\sqrt{2\pi}}(x-\mu)^2 \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$
Weibull	\mathbb{R}^+	$\mathcal{W}(x; \alpha, \beta, \mu)$	$\alpha\beta(x-\mu)^{\alpha-1} \exp(-\beta(x-\mu)^\alpha) \mathbb{1}_{\{x \geq 0\}}$
Erlang	\mathbb{R}^+	$\mathcal{E}r(x; \theta, \lambda)$	$\frac{\lambda^\theta x^{\theta-1} \exp(-\lambda x)}{(\theta-1)!}$
Gamma	\mathbb{R}^+	$\mathcal{G}(x; \alpha, \beta)$	$\frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} \exp(-x\beta) \mathbb{1}_{\{x \geq 0\}}$
Exponential	\mathbb{R}^+	$\mathcal{E}(x; \lambda)$	$\mathcal{G}(1, \lambda)$
Poisson	\mathbb{N}_0	$\mathcal{P}(x; \theta)$	$\exp(-\theta) \sum_{j=1}^{\infty} \frac{\theta^j}{j!} \delta_j$

evidence lower bound (ELBO), which we use for some evaluations in this thesis as well. Combining this setup lets us define a probabilistic objective J as

$$J(\theta) = \mathbb{E}_{p(\mathbf{x};\theta)} [f(\mathbf{x};\theta)].$$

Or, given the definition of expectations, we can also write this as the indefinite integral

$$J(\theta) = \int p(\mathbf{x};\theta) f(\mathbf{x};\theta) d\mathbf{x}.$$

In many cases, $f(\mathbf{x};\theta)$ also depends on other parameters than θ , but we assume that these are independent of θ , and absorb them into the definition of f for the sake of simplicity. Since p is not deterministic, we cannot directly formulate its gradient as a deterministic function. In a gradient propagation framework, this would mean that the computational graph of the gradient is interrupted at p . Figure 1.1 visualizes the issue in a simple computational graph. The computational graphs in this thesis are inspired by Schulman et al. [53], but differ in that they use the nodes for operations, and edges for values. Round nodes produce probabilistic values, while rectangular nodes produce deterministic values. We show the graphs for the gradient calculations in blue, and the normal computation path in black. Also, we separate the two graphs visually by a dashed horizontal line.

Gradient Derivation. Using gradient ascent, we want to find $\arg \max_{\theta} J(\mathbf{x};\theta)$. Hence, we take the gradient of $J(\theta)$ w.r.t. θ ,

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \int p(\mathbf{x};\theta) f(\mathbf{x};\theta) d\mathbf{x}.$$

With some assumptions based on probability theory [13, 25], we assume that we can interchange the derivative and integral,

$$\nabla_{\theta} J(\theta) = \int \nabla_{\theta} [p(\mathbf{x};\theta) f(\mathbf{x};\theta)] d\mathbf{x}.$$

Due to the product rule of derivatives and the same interchanging trick as before applied on both parts, we have

$$\nabla_{\theta} J(\theta) = \underbrace{\int (\nabla_{\theta} p(\mathbf{x};\theta)) f(\mathbf{x};\theta) d\mathbf{x}}_A + \underbrace{\int p(\mathbf{x};\theta) \nabla_{\theta} f(\mathbf{x};\theta) d\mathbf{x}}_B, \quad (1.1)$$

where we can solve the two parts A and B of the sum separately.

Estimating the Second Summand. First, we look on the second integral B , because this one is substantially easier to estimate, assuming we chose f to be differentiable w.r.t. θ . Note that this integral is an expectation

$$\mathbb{E}_{p(\mathbf{x};\theta)} [\nabla_{\theta} f(\mathbf{x}; \theta)] = \int p(\mathbf{x}; \theta) \nabla_{\theta} f(\mathbf{x}; \theta) d\mathbf{x}.$$

Monte-Carlo estimation allows us to estimate this integral by sampling $\hat{\mathbf{x}}^{(n)} \sim p(\hat{\mathbf{x}}; \theta)$, N times, and evaluating

$$\mathbb{E}_{p(\mathbf{x};\theta)} [\nabla_{\theta} f(\mathbf{x}; \theta)] \approx \frac{1}{N} \sum_{n=1}^N \nabla_{\theta} f(\hat{\mathbf{x}}^{(n)}; \theta).$$

With a high enough N , this should give us a good estimate for this component of the gradient, and, by the strong law of large numbers, it approaches the true value for $N \rightarrow \infty$, which is called *consistent* estimation [56]. This estimate is also *unbiased*, since

$$\mathbb{E}_{p(\mathbf{x};\theta)} \left[\frac{1}{N} \sum_{n=1}^N \nabla_{\theta} f(\hat{\mathbf{x}}^{(n)}; \theta) \right] = \frac{1}{N} \sum_{n=1}^N \mathbb{E}_{p(\mathbf{x};\theta)} [\nabla_{\theta} f(\hat{\mathbf{x}}^{(n)}; \theta)] = \mathbb{E}_{p(\mathbf{x};\theta)} [\nabla_{\theta} f(\mathbf{x}; \theta)].$$

However, this only solved the easier part of the gradient calculation.

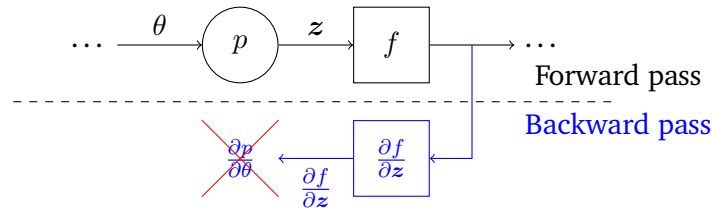
Estimating the First Summand. Recall the first component of the gradient from equation 1.1,

$$A = \int (\nabla_{\theta} p(\mathbf{x}; \theta)) f(\mathbf{x}; \theta) d\mathbf{x}.$$

Ideally, we would like to apply Monte-Carlo estimation here as well, which would also guarantee unbiasedness and consistency. Note, however, that we cannot directly express this as an expectation, since $\nabla_{\theta} p(\mathbf{x}; \theta)$ is generally not a probability distribution.

Solving this problem is the main goal of the approaches we evaluate in this thesis, and we present three different ways to do it, which all bring their own advantages and disadvantages. We refer to these approaches as Monte-Carlo estimators, because they all produce an expectation, which is intended to be estimated via Monte-Carlo estimation. In chapter 2, we introduce the three approaches, and show how they interact with Monte-Carlo sampling to estimate this component of the gradient.

Figure 1.1: Simple computation graph when trying to backpropagate the gradient through a probabilistic node p . Since we cannot directly compute $\partial z / \partial \theta$, the backpropagation ends at p . In this graph, $z \sim p(z; \theta)$ is the latent variable, and f is the cost. We omit the subgraph of the direct dependency of $f(z; \theta)$ on θ , since it does not impact this computation.



1.3 Related Work

In this section, we give an overview of the current state of research on measure-valued derivatives for variational methods. We do not focus on the pathwise and score-function estimators, as they are well explored, and have been in use for quite a while.

The measure-valued derivative was introduced by Pflug [51] in 1989 as the *weak derivative*. Heidergott et al. [28] extend the theory to a general differentiation concept for Markov chains. They also provide decompositions for many common distributions, e.g., for Gaussian distributions [27].

Buesing et al. [7] discuss MVDs in the machine learning setting as a finite-difference estimator, and compare it to other Monte-Carlo gradient estimation methods. Although the measure-valued derivative has mainly been used for other statistics applications yet, Mohamed et al. [46] and Rosca & Figurnov [52] find that it is also interesting for machine learning research in their surveys and empirical analysis. The topic is gaining traction in some specific areas, for example, reinforcement learning [44], and a general exploration of machine-learning applications seems worthwhile.

2 Gradient Estimators

In this chapter, we present three stochastic gradient estimators, the pathwise estimator in section 2.1, the score-function estimator in section 2.2, and the measure-valued estimator in section 2.3. Since this thesis is focused on the measure-valued estimator, we do not compare the pathwise and score-function estimators in detail. Instead, we work out the most important characteristics in comparison to the measure-valued estimator, and discuss them more thoroughly. There are two major metrics on which we base our comparisons. For one, we look at the variance of each estimator. This is an important measurement for the value of a stochastic estimator, because it not only influences the quality of the results, but also the number of Monte-Carlo samples required to get close to the true gradient. While a lower variance might be advantageous in most situations, we should also note that there is always a trade-off here. For example, a low variance estimator might get stuck in small, bad local optima. However, a higher variance estimator will struggle especially in smooth areas. The second major measure is the computation time of the estimators. This includes the number of loss function evaluations required for a single sample, as well as the computations required to compute the estimate. A general comparison of this measure is not possible, since it largely depends on the type of problem and the complexity of the model. However, we discuss some general tendencies of the estimators, e.g., the measure-valued estimator scaling linearly with the size of the latent space.

2.1 Pathwise Gradient Estimator

The pathwise gradient estimator [52] is the most frequently used approach of the three when the loss function f is differentiable w.r.t. the distribution parameters θ , and, as we will see, with good reason. It has some strict requirements, and often requires some work around it to be applicable.

As a general concept, the term pathwise is supposed to mean that we can follow the path of the gradient backwards through the whole computation. In practical terms, the pathwise estimator allows us to apply automatic gradient frameworks like TensorFlow [43] or PyTorch [49]. To get a visual understanding of the pathwise estimator, consider the graph in figure 2.1. In comparison to the issue presented in figure 1.1, we see that we avoid the probabilistic node in the computation graph of the gradient. In fact, avoiding the gradient computation through a probabilistic node is what all three approaches presented in this thesis have in common. The pathwise estimator achieves this by moving the dependency on θ out of the probabilistic node, and instead finding a new deterministic node $g(x; \theta)$, which we call the *reparameterization* of p . This new node $g(x; \theta)$ must fulfill a few restrictions for this method to work, and we cannot guarantee that there always exists a suitable reparameterization. First of all, it must ensure that there is a probability distribution $q(\epsilon)$, which does not depend on θ , such that $x = g(\epsilon; \theta)$ with $\epsilon \sim q(\epsilon)$. Then, we can rewrite A from equation 1.1 as

$$A = \int (\nabla_{\theta} p(x; \theta)) f(x; \theta) dx = \int q(\epsilon) \nabla_{\theta} f(g(\epsilon; \theta); \theta) d\epsilon, \text{ where } \epsilon \sim q(\epsilon),$$

which is an expectation in the form of

$$A = \mathbb{E}_{q(\epsilon)} [\nabla_{\theta} f(g(\epsilon; \theta); \theta)],$$

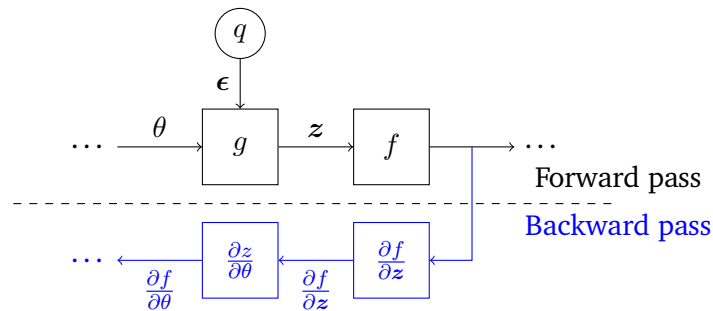
and in this form, we can directly estimate the gradient via Monte-Carlo estimation,

$$A \approx \frac{1}{N} \sum_{n=1}^N \nabla_{\theta} f(g(\epsilon^{(n)}; \theta); \theta), \text{ where } \epsilon^{(n)} \sim q(\epsilon),$$

which is called pathwise gradient estimator.

Variance. The pathwise gradient estimator is often seen as the default stochastic gradient estimator, because it usually has low variance. When we talk about variance in the context of stochastic gradient estimators, we mean the variance of single gradient estimates. This is a very important measure to compare the estimators, because it directly influences the required number of Monte-Carlo samples N to get a good estimate of the true gradient. The more variance an estimator has, the larger N has to be to sufficiently reduce the estimate's error. Intuitively, the variance of pathwise is low, because we keep all information on the intermediate computations, and propagate this information to the gradient estimate. This is different to the other two approaches, where we do not have access to the intermediate

Figure 2.1: Computational graph of the pathwise gradient estimator. The new deterministic g node replaces p , and allows the gradient to flow through the whole path. In this graph, $z = g(\epsilon; \theta)$, where $\epsilon \sim q(\epsilon)$, is the latent variable, and f is the cost. We omit the subgraph of the direct dependency of $f(z; \theta)$ on θ , since it does not impact this computation.



gradients. The price we pay for that is the strict requirements of the pathwise gradient estimator. We are required to ensure that we can propagate the gradient through all computations that use the parameters θ . This includes not only the mapping g , but especially the objective function f . A lot of research has been conducted to apply the pathwise estimator to many problems, where these restrictions are not given beforehand [21, 14, 42].

Computation Time. Since backpropagation is well optimized in modern machine learning frameworks, this gradient estimator is often also the most efficient timewise. However, this may become a problem in models, where we create a large computational graph between the sampling and computation of the cost function. This could, e.g., happen in variational auto-encoders, or generative adversarial networks, where we have a potentially large neural network model between the sampling and cost function.

Distribution $p(x; \theta)$	Base $q(\epsilon; \theta)$	Reparameterization $g(\epsilon; \theta)$
Normal $\mathcal{N}(\mu; \sigma)$	$\epsilon \sim \mathcal{N}(0, 1)$	$\mu - \sigma\epsilon$
Standard Normal $\mathcal{N}(0; 1)$	$\epsilon_1, \epsilon_2 \sim \mathcal{U}(0, 1)$	$\sqrt{\ln(1/\epsilon_1)} \cos(2\pi\epsilon_2)$
Exponential $\mathcal{E}(\lambda)$	$\epsilon \sim \mathcal{E}(1)$	$\frac{1}{\lambda}\epsilon$

Table 2.1: Reparameterizations for some common univariate distributions. We sample from the base distribution $q(\epsilon)$, and transform the samples with $g(\epsilon; \theta)$. $\mathcal{U}(a, b)$ denotes the uniform distribution.

2.2 Score-Function Gradient Estimator

The score-function gradient estimator [15], also called Reinforce [58], or log-ratio trick, uses a form of importance sampling to estimate the gradient using samples of the current distribution. Again, let's look at the problematic term

$$A = \int (\nabla_{\theta} p(\mathbf{x}; \theta)) f(\mathbf{x}; \theta) d\mathbf{x},$$

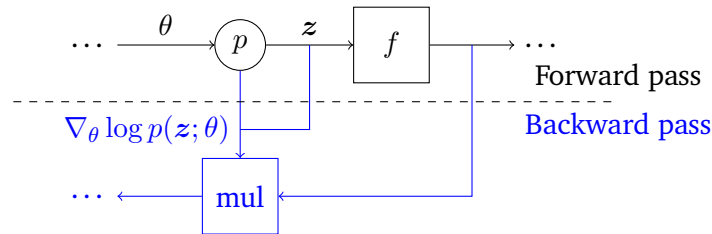
from equation 1.1. We want to express this in terms of expectations. Using the fact that

$$\nabla_{\theta} \log p(\mathbf{x}; \theta) = \frac{\nabla_{\theta} p(\mathbf{x}; \theta)}{p(\mathbf{x}; \theta)},$$

we can derive the score function gradient as

$$\begin{aligned} A &= \int (\nabla_{\theta} p(\mathbf{x}; \theta)) f(\mathbf{x}; \theta) d\mathbf{x} \\ &= \int \frac{p(\mathbf{x}; \theta)}{p(\mathbf{x}; \theta)} (\nabla_{\theta} p(\mathbf{x}; \theta)) f(\mathbf{x}; \theta) d\mathbf{x} \\ &= \int p(\mathbf{x}; \theta) \frac{\nabla_{\theta} p(\mathbf{x}; \theta)}{p(\mathbf{x}; \theta)} f(\mathbf{x}; \theta) d\mathbf{x} \\ &= \int p(\mathbf{x}; \theta) f(\mathbf{x}; \theta) \nabla_{\theta} \log p(\mathbf{x}; \theta) d\mathbf{x} \\ &= \mathbb{E}_{p(\mathbf{x}; \theta)} [f(\mathbf{x}; \theta) \nabla_{\theta} \log p(\mathbf{x}; \theta)]. \end{aligned}$$

Figure 2.2: Computational graph of the score-function gradient estimator. We sample from p , which is not a problem, because we do not require its gradient for the computation of $f(z)\nabla_{\theta}\log p(z)$. In this graph, $z \sim p(z; \theta)$ is the latent variable, and f is the cost. We omit the subgraph of the direct dependency of $f(z; \theta)$ on θ , since it does not impact this computation.



In terms of Monte-Carlo estimation, we have the score-function gradient estimator

$$A \approx \frac{1}{N} \sum_{n=1}^N f(\mathbf{x}; \theta) \nabla_{\theta} \log p(\mathbf{x}; \theta), \text{ where } \mathbf{x} \sim p(\mathbf{x}; \theta),$$

which is more generally applicable than the pathwise gradient estimator, but also comes with some downsides. Figure 2.2 shows the computational graph of the score-function gradient estimator. Important to note is that this estimator neither requires a differentiable objective f , nor a reparameterizable p . Even though, in many cases, we would like to choose f to be differentiable, sometimes we cannot avoid a non-differentiable objective. E.g., in some reinforcement learning settings, where we do not know the concrete form of f , and can only evaluate it as a black-box. Also, whenever we do not have a reparameterization g for p , which is required for the pathwise gradient estimator as shown in section 2.1, we can still apply the score function estimator. The only hard requirement of the score-function estimator is that we can compute $\nabla_{\theta} \log p(\mathbf{x}; \theta)$.

Variance. The score-function estimator has the highest variance of all three estimators. One reason is the term $\nabla_{\theta} \log p(\mathbf{x}; \theta)$, which produces large negative values for low probabilities. Especially for higher latent dimensions, this causes a huge problem, because the probability of any single sample becomes very small as the space grows exponentially. Thus, the variance is not only high for the estimate's direction, but also for its magnitude. For this reason, the score-function estimator depends a lot more on a good learning rate

schedule, and profits strongly from optimization schemes which adjust the gradient's magnitude, e.g., Adam [36]. Intuitively, we can imagine that, since we sample from p directly, we tend to underestimate the importance of outcomes which are unlikely under the current distribution. Due to the strong negative response to unlikely samples, the gradient is sometimes forced into a lesser informed direction. Hence, we have to average over a higher number of gradient estimates. To reduce the variance of the score-function estimate, we can introduce a baseline b , e.g., a running average of the cost, such that the estimator becomes

$$A \approx \frac{1}{N} \sum_{n=1}^N (f(\mathbf{x}; \theta) - b) \nabla_{\theta} \log p(\mathbf{x}; \theta), \text{ where } \mathbf{x} \sim p(\mathbf{x}; \theta).$$

The estimate remains unbiased. To show the unbiasedness, we first use the linearity of expectations to separate the expectation into the two parts

$$\mathbb{E}_{p(\mathbf{x}; \theta)} [(f(\mathbf{x}; \theta) - b) \nabla_{\theta} \log p(\mathbf{x}; \theta)] = \underbrace{\mathbb{E}_{p(\mathbf{x}; \theta)} [f(\mathbf{x}; \theta) \nabla_{\theta} \log p(\mathbf{x}; \theta)]}_{\text{unbiased}} - \underbrace{b \mathbb{E}_{p(\mathbf{x}; \theta)} [\nabla_{\theta} \log p(\mathbf{x}; \theta)]}_0,$$

as long a b is independent of \mathbf{x} . The second term is zero, because

$$\begin{aligned} \mathbb{E}_{p(\mathbf{x}; \theta)} [\nabla_{\theta} \log p(\mathbf{x}; \theta)] &= \int p(\mathbf{x}; \theta) \nabla_{\theta} \log p(\mathbf{x}; \theta) d\mathbf{x} \\ &= \int p(\mathbf{x}; \theta) \frac{\nabla_{\theta} p(\mathbf{x}; \theta)}{p(\mathbf{x}; \theta)} d\mathbf{x} \\ &= \nabla_{\theta} \int p(\mathbf{x}; \theta) d\mathbf{x} = \nabla_{\theta} 1 = 0. \end{aligned}$$

However, in most cases, b depends on evaluations of f , which is not unbiased, but still produces decent results [23].

Computation Time. The time required for computing a single estimate is comparable to the pathwise gradient estimator, though it does not depend as much on the complexity of f , as we do not need to compute its gradient. However, as we show in chapter 4, the algorithm suffers more from a higher number of samples than the measure-valued estimator with coupled sampling, which we introduce in detail in the next section.

2.3 Measure-Valued Derivative

The measure-valued gradient estimator [46], also called weak derivative, or measure-valued derivative (MVD), is the main focus of this thesis. In equation 1.1, we derived the term

$$A = \int (\nabla_{\theta} p(\mathbf{x}; \theta)) f(\mathbf{x}; \theta) d\mathbf{x},$$

as a summand of the gradient of our probabilistic objective $\nabla_{\theta} J(\theta)$. Since this is not an expectation, we cannot directly apply Monte-Carlo estimation. Instead, using the measure-valued derivative, we rearrange this term such that it becomes a sum of expectations. The measure-valued derivative is based on a Hahn-Jordan decomposition of the gradient $\nabla_{\theta} p(\mathbf{x}; \theta)$ into

$$\nabla_{\theta_i} p(\mathbf{x}; \theta) = c_{\theta_i} [p_i^+(\mathbf{x}; \theta) - p_i^-(\mathbf{x}; \theta)].$$

This decomposition exists for all probability distributions, and is not unique. We explicitly use the index i here, to indicate that this decomposition must be applied for all dimensions of θ . Hence, the measure-valued gradient estimate scales linearly in the dimensions of θ . Using the gradient decomposition, A can be written as

$$\begin{aligned} A &= \int (\nabla_{\theta_i} p(\mathbf{x}; \theta)) f(\mathbf{x}; \theta) d\mathbf{x} \\ &= \int c_{\theta_i} (p_i^+(\mathbf{x}; \theta_i) - p_i^-(\mathbf{x}; \theta_i)) f(\mathbf{x}; \theta) d\mathbf{x} \\ &= c_{\theta_i} \left(\int p_i^+(\mathbf{x}; \theta_i) f(\mathbf{x}; \theta) d\mathbf{x} - \int p_i^-(\mathbf{x}; \theta_i) f(\mathbf{x}; \theta) d\mathbf{x} \right), \end{aligned}$$

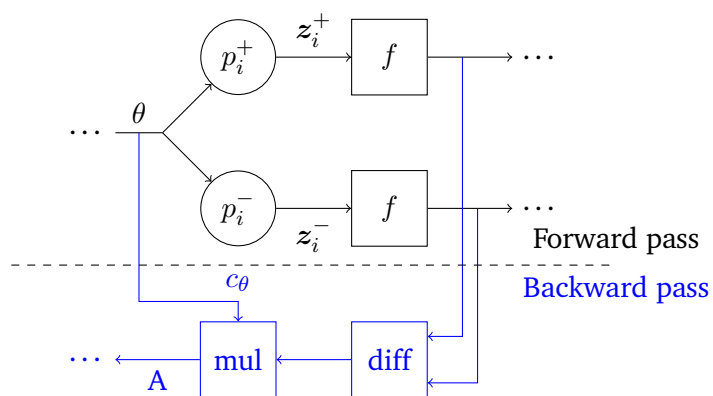
for $i = 1, \dots, D$ and $\theta \in \mathbb{R}^D$. Hence, the measure-valued gradient is

$$c_{\theta_i} \left(\mathbb{E}_{p_i^+(\mathbf{x}; \theta)} [f(\mathbf{x}; \theta)] - \mathbb{E}_{p_i^-(\mathbf{x}; \theta)} [f(\mathbf{x}; \theta)] \right),$$

where we can use Monte-Carlo estimation. The computational graph is shown in figure 2.3. Table 2.2 shows some common decompositions. We list only a few here, but decompositions exist for all probability distributions.

Variance. The measure-valued gradient estimator has a considerably lower variance than the score-function estimator presented in section 2.2, and, as we show in chapter 4, is comparable to the pathwise estimator from section 2.1. This comes from the sampling

Figure 2.3: Computational graph of the measure-valued gradient estimator. We sample from the two components p_i^+ and p_i^- of the decomposition, evaluate f , and multiply the difference of the results by the normalization factor θ . In this graph, $z \sim p(z; \theta)$ is the latent variable, and f is the cost. We omit the subgraph of the direct dependency of $f(z; \theta)$ on θ_i , since it does not impact this computation.



method used by this estimator. Figure 2.4 and figure 2.5 show how we sample from a univariate normal distribution to get a measure-valued gradient estimate for the mean μ and the variance σ^2 . Generally, we focus the sampling very specifically for the positive measure $z^+ \sim p^+(z; \theta)$, and the negative measure $z^- \sim p^-(z; \theta)$. Due to the construction of the gradient estimate, we separate the change induced by $\nabla_{\theta} p(z; \theta)$ into the part p^+ , where we add mass, and the part p^- , where we remove mass. By focussing the sampling on exactly these areas, each sample becomes more meaningful compared to simple importance sampling, as it is used by the score function estimator. The variance is given by

$$\mathbb{V}_{p(\mathbf{x}; \theta)} [\nabla_{\theta} f(\mathbf{x})] = \mathbb{V}_{p^+(\mathbf{x}; \theta)} [f(\mathbf{x})] + \mathbb{V}_{p^-(\mathbf{x}; \theta)} [f(\mathbf{x})] - 2\text{Cov}_{p^+(\mathbf{x}'; \theta)p^-(\mathbf{x}; \theta)} [f(\mathbf{x}'), f(\mathbf{x})],$$

We can also reduce the variance further by positively correlating $f(\mathbf{x}')$ and $f(\mathbf{x})$, e.g., by using the common random numbers during sampling. This trick is called coupling, and we implicitly use it for all our evaluations, similar to the baseline for the score function.

Computation Time. Since we need to sample in each dimension of θ , the computation time of one estimate grows linearly with the number of dimensions of θ . E.g., for a k -dimensional multivariate Gaussian with diagonal, i.e. factorizable, covariance, we require k samples for the positive and negative components, which sums up to $4k$ samples and evaluations of f for one estimate. This is why, whenever we talk about a fair comparison between the score-function estimator, the pathwise estimator, and the measure-valued derivative estimator, we refer to using the same number of evaluations of f with all of them, by increasing the number of samples of the other estimators accordingly. In short, let $\theta \in \mathbb{R}^D$ be the parameters of a distribution, then the measure-valued gradient estimator requires $2D$ evaluations for a single estimate, while the other two estimators only require one.

If $p(\mathbf{x}; \theta)$ is fully factorizable, i.e., the full gradient is equal to the sum of the partial gradients, we can estimate the gradient of any multivariate distribution by sampling from the distribution and replacing the i -th dimension with samples from p_i^+ and p_i^- . This allows us to compute correlated samples for all dimensions using only one sample of the multivariate distribution, which reduces the variance and saves computation. For the rest of this thesis, unless stated otherwise, we assume $p(\mathbf{x}; \theta)$ is fully factorizable, or, in other words, the univariate $p_i(\mathbf{x}; \theta_i)$ are pairwise independent. This is mostly relevant when we talk about the implementation in chapter 3 and evaluation in chapter 4. Note, though, that the measure-valued estimator is still applicable in non-factorizable cases, e.g., for Gaussians with full covariance.

Figure 2.4: Probability densities for the measure-valued derivative decomposition for the gradient $\partial\mathcal{N}/\partial\mu$ of the univariate standard normal distribution $\mathcal{N}(0, 1)$, where $p^+(\mu, \sigma^2) = \mu + \sigma\mathcal{W}(2, 0.5)$, and $p^-(\mu, \sigma^2) = \mu - \sigma\mathcal{W}(2, 0.5)$.

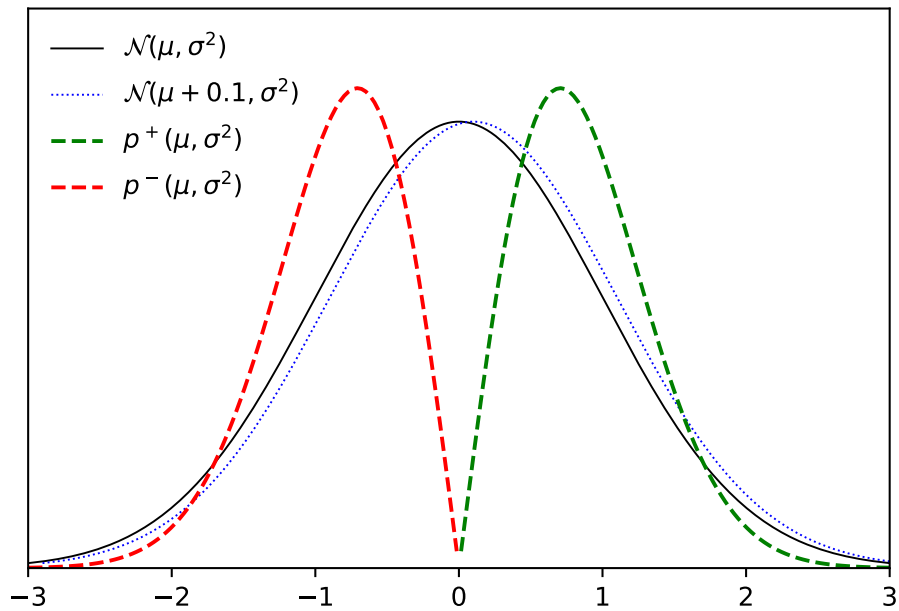
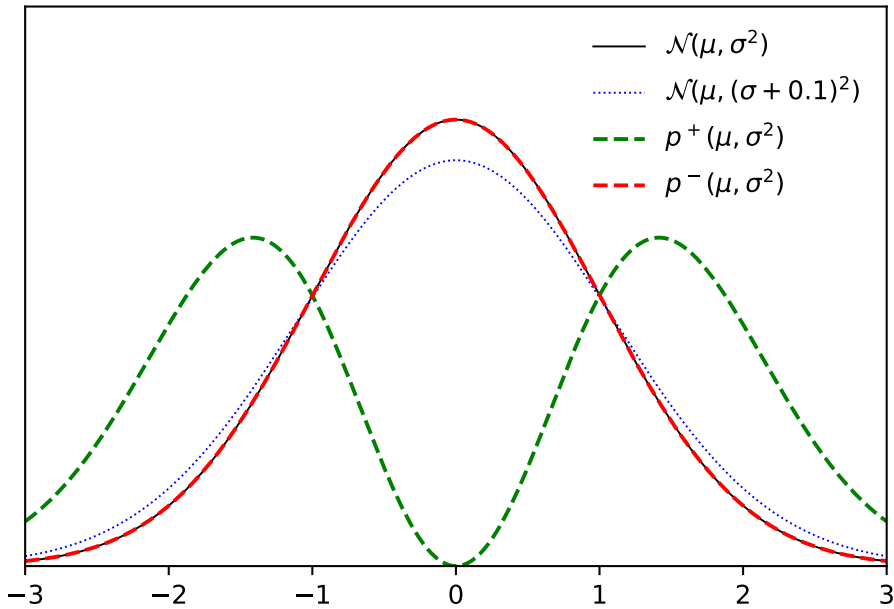


Figure 2.5: Probability densities for the measure-valued derivative decomposition for the gradient $\partial\mathcal{N}/\partial\sigma$ of the univariate standard normal distribution $\mathcal{N}(0, 1)$, where $p^+(\mu, \sigma^2) = \mathcal{M}(0, 1)$, and $p^-(\mu, \sigma^2) = \mathcal{N}(0, 1)$.



Distribution $p(x; \theta)$	Constant c_θ	Positive part p^+	Negative part p^-
Bernoulli(θ)	1	δ_1	δ_0
Poisson(θ)	1	$\mathcal{P}(\theta) + 1$	$\mathcal{P}(\theta)$
Normal(θ, σ^2)	$1/\sigma\sqrt{2\pi}$	$\theta + \sigma\mathcal{W}(2, 0.5)$	$\theta - \sigma\mathcal{W}(2, 0.5)$
Normal(μ, θ^2)	$1/\theta$	$\mathcal{M}(\mu, \theta^2)$	$\mathcal{N}(\mu, \theta^2)$
Exponential(θ)	$1/\theta$	$\mathcal{E}(\theta)$	$\theta^{-1}\mathcal{E}r(2)$
Gamma(a, θ)	a/θ	$\mathcal{G}(a, \theta)$	$\mathcal{G}(a + 1, \theta)$
Weibull(α, θ)	$1/\theta$	$\mathcal{W}(\alpha, \theta)$	$\mathcal{G}(2, \theta)^{1/\alpha}$

Table 2.2: Measure-valued derivative triples (c_θ, p^+, p^-) to estimate $\nabla_{\theta} p$ for common distributions. See Table 1.1 for information on the distributions.

2.4 Convex Combination of Estimators

Convex combinations of multiple unbiased estimators are still unbiased. Let us define two unbiased estimators \widehat{T}_1 and \widehat{T}_2 . Considering the expectation of the convex combination of \widehat{T}_1 and \widehat{T}_2 , we can show that due to the linearity of the expectation

$$\begin{aligned}
\mathbb{E}_{p(\mathbf{x}; \theta)} \left[c\widehat{T}_1(\mathbf{x}; \theta) + (1 - c)\widehat{T}_2(\mathbf{x}; \theta) \right] &= c\mathbb{E}_{p(\mathbf{x}; \theta)} \left[\widehat{T}_1(\mathbf{x}; \theta) \right] + (1 - c)\mathbb{E}_{p(\mathbf{x}; \theta)} \left[\widehat{T}_2(\mathbf{x}; \theta) \right] \\
&= c\mathbb{E}_{p(\mathbf{x}; \theta)} [f(\mathbf{x}; \theta)] + (1 - c)\mathbb{E}_{p(\mathbf{x}; \theta)} [f(\mathbf{x}; \theta)] \\
&= \mathbb{E}_{p(\mathbf{x}; \theta)} [f(\mathbf{x}; \theta)],
\end{aligned}$$

hence, the convex combination of \widehat{T}_1 and \widehat{T}_2 remains unbiased.

Using a convex combination of estimators might be interesting for the measure-valued estimator, because it allows us to use the estimator only for some dimensions. This gives us some flexibility in choosing how much computation we want to use. In section 4.3, we discuss the option of naively replacing random dimensions of a score-function estimate with measure-valued estimates. Though, generally, we find that there are a lot of unexplored options in this approach.

2.5 Gaussian Mixture Models

As the measure-valued estimator is applicable for all distributions [28], we consider discrete mixture models in this thesis as well. These are discrete combinations of multiple probability distributions. Hence, we have two main parts. The first, which we will refer to as the *selector* $s(\phi)$, is a discrete probability distribution. We call the second *components* $c(\theta_z)$, which is a set of probability distributions from a family of distributions. To sample from discrete mixtures, we first sample from the selector

$$z \sim s(z; \phi),$$

then sample from the selected component

$$\mathbf{x} \sim c(\mathbf{x}; \theta_z).$$

As a practical and often used example, we first focus on Gaussian mixture models, where the selector is a categorical distribution $\mathcal{C}(\phi)$, and the $k \in \mathbb{N}^+$ components are Gaussian distributions $\mathcal{N}(\mu, \Sigma)$.

Gaussian Mixtures and the Score-Function Estimator. Since the only hard requirement is that we know the derivatives $\frac{\partial}{\partial \phi} \log \mathcal{C}(z; \phi)$ and $\frac{\partial}{\partial \theta} \log \mathcal{N}(\mathbf{x}; \theta_z)$, we can apply the score-function estimator to these problems. The log-derivative of a multivariate distribution is calculated as

$$\frac{\partial}{\partial \mu} \log \mathcal{N}(\mathbf{x}; \mu, \Sigma) = \Sigma^{-1} (\mathbf{x} - \mu),$$

and

$$\frac{\partial}{\partial \Sigma} \log \mathcal{N}(\mathbf{x}; \mu, \Sigma) = -\frac{1}{2} [2\Sigma^{-1} - (\Sigma^{-1} \odot I) + \Sigma^{-1}(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T \Sigma^{-1}],$$

where $(\Sigma^{-1} \odot I)$ denotes the Hadamard product, i.e., a diagonal matrix containing the diagonal of Σ^{-1} . For the categorical distribution, we find that the derivative of the log-probability is

$$\nabla_{\phi} \log \mathcal{C}(\mathbf{x}; \phi) = \nabla_{\phi} \log \prod_{i=1}^k \phi_i^{x_i^T \mathbf{x}} = \nabla_{\phi} \sum_{i=1}^k x_i^T \mathbf{x} \log \phi_i = \left[\frac{1}{\phi_1}, \dots, \frac{1}{\phi_k} \right]^T,$$

where $\mathbf{x}_i \in \{0, 1\}^k$ is a vector with 1 at the i -th position and 0 everywhere else. Hence, we can directly use the score-function estimator.

Gaussian Mixtures and the Pathwise Estimator. To use the pathwise estimator for discrete mixtures, we have to use a trick to get around the discrete distribution. Because the sampling requires a non-differentiable $\arg \max$ operation, we cannot reparameterize discrete distributions. Hence, we need a substitute distribution, which samples as if it was a categorical distribution, but can be reparameterized. Current research uses the Gumbel softmax [33] as a substitute. To sample from the Gumbel-softmax distribution, we sample $g_i \sim \text{Gumbel}(0, 1)$, and calculate

$$z \sim \arg \max_i [g_i + \log \phi_i].$$

In practical implementations, we sample from the Gumbel distribution using uniform samples $u \sim \mathcal{U}(0, 1)$, and calculate $g = -\log(-\log(u))$. Since the samples g_i do not depend on the class probabilities ϕ_i , and all other operations are differentiable, we can calculate the pathwise gradient estimate of $\mathcal{C}(z; \phi)$ via

$$\nabla_{\phi} \mathcal{C}(z; \phi) = \nabla_{\phi} [-\log(-\log(u)) + \log \phi] = \left[\frac{1}{\phi_i}, \dots, \frac{1}{\phi_k} \right]^T,$$

which is the same as for the score-function gradient. However, since the sampling is not the same, the estimators still produce different results.

Gaussian Mixtures and the Measure-valued Estimator. At last, we can also estimate the gradient of a Gaussian mixture using the measure-valued estimator. For the measure-valued estimate of the gradient of the components $\mathcal{N}(\mathbf{x}; \theta_i)$ for $i = 1, \dots, k$, we refer to the decomposition listed in table 2.2.

As for the measure-valued gradient of the categorical distribution, we use the probability mass function

$$\mathcal{C}(\mathbf{x}; \phi) = \sum_{j=1}^k \mathbf{x}_j^T \mathbf{x} \phi_j = \mathbf{x}_i^T \mathbf{x} \phi_i + (\mathbf{1} - \mathbf{x}_i)^T \mathbf{x} (1 - \phi_i),$$

for every output $i = 1, \dots, k$. Same as with the score-function derivation, we assume the distribution outputs a one-hot encoded class $\mathbf{x} \in \{0, 1\}^k$, $\mathbf{x} \sim \mathcal{C}(\mathbf{x}; \phi)$. Taking the partial derivative for every dimension $j = 1, \dots, k$, we get

$$\frac{\partial}{\partial \phi_i} \mathcal{C}(\mathbf{x}; \phi) = \frac{\partial}{\partial \phi_i} (\mathbf{x}_i^T \mathbf{x}) \phi_i + ((\mathbf{1} - \mathbf{x}_i)^T \mathbf{x}) (1 - \phi_i) = \mathbf{x}_i^T \mathbf{x} - \sum_{\substack{j=1 \\ j \neq i}}^k \mathbf{x}_j^T \mathbf{x}.$$

Due to \mathbf{x} and \mathbf{x}_i being one-hot encodings, the expression $\mathbf{x}_i^T \mathbf{x}$ is 1 if and only if $\mathbf{x} = \mathbf{x}_i$. This means we can replace terms in the form of $\mathbf{x}_i^T \mathbf{x}$ with the Dirac delta distribution $\delta(\mathbf{x}_i)$. Hence, we get the measure-valued triplet

$$c_\phi = \mathbf{1}, \quad p_i^+(\phi) = \delta(\mathbf{x}_i), \quad p_i^-(\phi) = \delta(\mathbf{1} - \mathbf{x}_i).$$

We should note that this means we need to evaluate the loss function only once per category. As discussed in section 2.3, we normally expect $2\mathcal{D}$, where \mathcal{D} is the number of dimensions of ϕ , i.e., k . However, in this case we require only \mathcal{D} evaluations. This is due to the fact that we are evaluating the full domain of the distribution, essentially marginalizing over it, instead of sampling from it.

In conclusion, we can apply all three estimators to Gaussian mixtures models. We conduct some experiments to evaluate their respective performances in section 4.4.

3 A Framework for Monte-Carlo Gradient Estimators

We introduced the probabilistic objective in chapter 1, and the three gradient estimators - pathwise, score-function, and measure-valued - in the last chapter 2. In this chapter, we design a Python framework based on PyTorch and describe how these estimators are implemented in it. The main contribution of this framework is the extensibility with regards to different estimators, probability distributions, models, and datasets. Essentially, we can extend and replace any of these parts.

At the end of the chapter, and in chapter 4, we conduct some experiments and evaluate their results to compare the characteristics of the estimators in practice. Finally, in chapter 5, we discuss the results and develop some practical insights on how the estimators should be used.

3.1 Framework Description

In this section, we describe some of the issues with most implementations of variational methods in PyTorch. First, let us consider how most current implementations work in PyTorch. As an example, we use a small variational auto-encoder [35], which we show on a high level in figure 3.2.

The variational auto-encoder is commonly implemented using `rsample` of PyTorch distributions, which implements the reparameterization trick, i.e., the pathwise gradient estimator. In figure 3.1 we can see how many assumptions are implicitly put into most variational auto-encoder implementations. While we also find in 4 that this is not a bad default, it is problematic that this is treated as the only option. Hence, in our framework, we want to make obvious, that we are using the pathwise estimator here. Also, we want to design this such that the surrounding code does not need to know about the specific

estimator used. Unfortunately, we cannot directly integrate the other estimators into the automatic gradient calculation offered by the framework. The reason is that they do not use the derivative of their successor like the pathwise estimator does, but instead rely on the losses directly. To be able to implement general Monte-Carlo gradient estimators, we define an estimator as $\widehat{\mathcal{MC}}(f, d, x)$, where f is the loss function, d is a decoding or prediction function, mapping from samples to the outputs we expect of the model, and x is the data or the label, depending on what the model expects. This allows each estimator to function on the same inputs. In addition to the different inputs for the gradient estimation, the other estimators also rely on different sampling methods than the pathwise estimator. For the score function estimator, we need to make sure that the samples are not attached to the computation graph, which we do by simply sampling directly from p . For the measure-valued estimator, we even need to sample from the two distributions p^+ and p^- . We do this by stacking the samples of the two distributions, then relying on PyTorch's broadcasting logic to calculate the losses without having to change the loss function. This is also the reason why we cannot directly pass the losses to $\widehat{\mathcal{MC}}$, as we don't know the samples beforehand. Consider figure 3.3 on how this looks in terms of code.

Important to notice is that we commonly see the output of the encoder being the distribution parameters directly. This has two major disadvantages. For one, we obviously have to change the encoder model whenever we change the distribution. While this is not problematic for the implementation itself, it causes many implicit dependencies. Also, it hides the parameterization of the distribution. Most distributions expect parameters of a specific domain \mathbb{D} , e.g., for multivariate normal distributions $\mathcal{N}(\mu, \Sigma)$, Σ must be positive semidefinite. Since we often work with models that produce outputs in \mathbb{R} , we use functions $\mathbb{R} \rightarrow \mathbb{D}$, to ensure that model outputs are in the required domain. As an example, for a multivariate normal distribution with diagonal covariance, we could train the model to produce $\exp \ln \sigma$ instead, which allows us to train the model for $\ln \sigma$, and let the automatic backpropagation handle the gradient. There are always advantages and disadvantages to these transformations, hence one might be inclined to compare, or exchange them in some situations. This is once again a case, where the typical implementations are very resistant to changes. In figure 3.1, we don't see anything of this transformation, because it was absorbed into the encoder. We argue that the encoder should not be the entity which this transformation depends on. Instead, we build the framework such that the encoder can be an arbitrary model which can output any $\mathbb{R}^{\mathcal{D}}$, where \mathcal{D} is the sum of the dimensions of all parameters of the distribution. E.g., for a multivariate normal distribution with 5 latent dimensions, i. e. $\mathcal{D} = 30$, we expect the encoder output to be in \mathbb{R}^{30} , which puts as much flexibility into the encoder as possible. We call this output of the encoder *raw parameters*. To deal with the parameterization of the distribution, we instead create



Figure 3.1: Typical implementation of VAEs in PyTorch, omitting boilerplate code.

```
mu, sigma = vae.encode(original)
normal = torch.distributions.Normal(mu, sigma^2)
sample = normal.rsample()
reconstruction = vae.decode(sample)
# Binary Cross-Entropy + KL Divergence
losses = loss_function(reconstruction, original)
losses.mean().backward()
```

subclasses of the distributions, which transform the raw parameters into a valid parameter set for the distribution. This gets rid of the two disadvantages mentioned above. For one, we can now replace the encoder with any model, without requiring the model to know anything about the distribution. Secondly, we can also replace the distribution, as well as its parameterization, without having to change anything about the surrounding model, as long as we provide the correct number of raw parameters. Producing the correct number of parameters, however, should be trivial in most situations, as we also implement a way to ask the distribution for the dimensions of its parameters.

3.2 Extensibility of the Framework

In this section, we describe which parts of the algorithms can be extended — datasets, models, distributions, and gradient estimators.

Datasets. Datasets are loaded using a dataset registry, which allows us to add new datasets by registering a dataset class. This class requires a function for loading the dataset, the shape of the data, as well as a string identifier. This allows us to derive the models' dimensions directly from the given dataset. The datasets also support configurable batch sizes.

Models. The framework is generally compatible with any PyTorch models, and supports the same functionalities, e.g., persisting to disk, and automatic parameter detection for

optimizers. To add a probabilistic node to a model, we instantiate one of our model classes, which have the same sampling interface as the standard PyTorch distributions.

Distributions. Distributions of the framework adhere to the sampling interface of PyTorch distributions. However, implement the distributions such that they take any real numbers as inputs, and assume they ensure the restrictions. One example for this would be a categorical distribution which applies a softmax on its parameters to ensure normalization. This allows us to implement different parameterizations via subclasses which are separated from models. The distributions also implement a backward function, which estimates the gradient given an estimation strategy, as well as backpropagating that gradient through the PyTorch autograd [1]. This means that, as long as the distribution parameters are attached to the computational graph, the gradient is correctly propagated through the graph regardless of the estimator.

Gradient Estimator Strategies. We support the score-function estimator, as well as the measure-valued estimator for all implemented distributions, and the pathwise estimator where it is applicable. This includes categorical distributions using the Gumbel softmax, which could also be extended to other discrete distributions. To estimate a gradient, we call backward on the distribution instance with the estimator strategy, and a function that calculates the losses from samples. This allows each strategy to implement their own sampling and loss evaluations. The clean separation of distributions and estimators allows us to replace the specific components for evaluations and comparisons, while it guarantees a persistent setup. I.e., whenever we compare sets of configurations, we can be sure that the differences we show come from these changes.

3.3 Bayesian Logistic Regression

In this section, we present how we implement Bayesian logistic regression in our framework. We describe Bayesian linear regression first, because Bayesian logistic regression is based on it.

Bayesian linear regression uses a linear transformation $w^T x$ of the data x and the sampled weights w . The intuitive idea here is that we do not train the model parameters directly, but instead we train a distribution over models which most likely explain the observed

data. To do this, we maximize the likelihood $p(y|\mathbf{x}, w, \theta)$, with respect to the distribution parameters θ . The objective of our optimization becomes

$$\arg \max_{\theta} L(\theta) = ELBO_{p,q}(\theta) = \mathbb{E}_{w \sim p(w;\theta)} [D_{KL}(p||q) + \log p(y|\mathbf{x}, w, \theta)],$$

where *ELBO* is the Evidence Lower Bound, and $D_{KL}(p||q)$ is the Kullback-Leibler divergence between the latent distribution p and a prior q , which we have to choose. When compared to vanilla linear regression, the Bayesian approach has some advantages. The prior serves as a regularizer, as well as enabling us to add some prior knowledge into the training process. Also, the variance of the model gives us an indication of the confidence of the predictions.

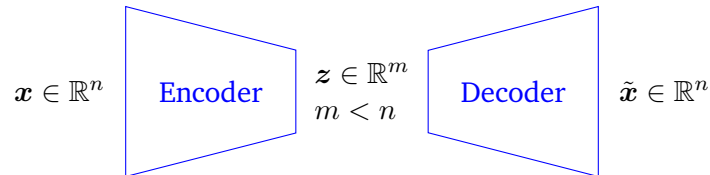
Because the posterior can be found analytically for Gaussian distributions, using variational inference in this context is not very useful. However, Bayesian logistic regression is based on this approach and does not have a closed-form solution. Bayesian logistic regression simply adds a logistic function to confine the outputs to the interval $[0, 1]$. This transforms the model into a binary classifier, which learns to separate the classes in the latent space. To implement this in the framework, all we need to implement is a new model class which applies the linear transformation, instantiates the distribution, and lastly applies the logistic function to get the prediction. We present the experiments and results we get from Bayesian logistic regression in section 4.1.

3.4 Variational Auto-Encoders

In this section, we describe how we implement variational auto-encoders (VAEs) in our framework, and propose experiments to show some relevant differences between the estimators. In the course of our evaluations, we focus a lot on VAEs, because, for one, they allow for varying complexity through the two networks used in the model. Also, the unsupervised training and the probabilistic reconstructions offer an interesting solution for many problems where training a deterministic network in a supervised fashion would not be feasible. For this reason, VAEs are often used for generating stochastic alterations of data, e.g., to generate images in the style of an artist, or to generate faces of non-existent people [54, 57, 55]. The flexibility of the approach allows for a lot of use-cases, making VAEs an interesting model for various applications.

In general, we define a VAE as the combination of an encoder, a decoder, and a latent distribution, which is parameterized through the encoder. The high-level flow of the

Figure 3.2: A high-level representation of the basic variational auto-encoder architecture. The encoder and decoder are neural networks. The decoder is trained directly through backpropagating the losses, but we require a gradient estimator for the encoder gradients. In this figure, we call the data $\mathbf{x} \in \mathbb{R}^n$, the latent representation $\mathbf{z} \sim p(\mathbf{z}; \text{Encoder}(\mathbf{x}))$, $\mathbf{z} \in \mathbb{R}^m$, and the reconstruction $\text{Decoder}(\mathbf{z}) = \tilde{\mathbf{x}}$, $\tilde{\mathbf{x}} \in \mathbb{R}^n$.



model is shown in figure 3.2. Like Bayesian logistic regression, the encoder network uses variational inference to encode the data into a latent space. We use a much smaller number of dimensions m for the latent space, than the dimensions n of the data. Hence, the encoder $E(\mathbf{x}; \mathbf{w}_E)$ outputs the parameters θ of the distribution $p(\mathbf{z}; \theta)$. To make the distribution tractable, we choose a family of distributions and limit our search to that family, e.g., Gaussian distributions. Then, to train the decoder, we simply sample from the latent space using the distribution $\mathbf{z} \sim p(\mathbf{z}; \theta)$, and run the samples through the decoder $D(\mathbf{z}; \mathbf{w}_D)$ to get reconstructions. The complete objective is

$$\arg \max_{\mathbf{w}_E, \mathbf{w}_D} J(\mathbf{w}_E, \mathbf{w}_D) = \int \int p(\mathbf{z}; E(\mathbf{x}; \mathbf{w}_E)) f(D(\mathbf{z}; \mathbf{w}_D)) d\mathbf{z} d\mathbf{x},$$

where \mathbf{w}_E are the parameters of the encoder network, and \mathbf{w}_D are the parameters of the decoder network.

For the training, we fix the weights of one network, while we train the other. Training the decoder like this is done easily by running detached samples $\mathbf{z} \sim p(\mathbf{z}; \theta)$ for fixed θ through the decoder. By detached, we mean that the computational graph is cut off from the sample, which stops the backpropagation at the sample. Hence, we approximate the gradient with regards to the decoder parameters using the Monte-Carlo sampling and batch stochastic gradient ascent with batch size B

$$\nabla_{\mathbf{w}_D} J(\mathbf{w}_D) \approx \frac{1}{B} \sum_{j=1}^B \frac{1}{N} \sum_{i=1}^N \nabla_{\mathbf{w}_D} f(D(\mathbf{z}_i)), \quad \text{where, } \mathbf{z}_i \sim p(\mathbf{z}; E(\mathbf{x}_j; \mathbf{w}_E)).$$

Figure 3.3: VAE implementation using the framework of this thesis, omitting boilerplate code. In contrast to figure 3.1, the loss function in this case only calculates the binary cross-entropy loss.

```
normal = vae.encode(original)
# Calculate gradients of the decoder.
reconstruction = vae.decode(normal.sample())
bce_function(reconstruction, original).mean().backward()
# Calculate gradients of the encoder.
normal.kl(prior).mean().backward()
normal.backward(bce_function, vae.decode, original)
```

Since the decoder is a deterministic model, we calculate the gradients of w_D through standard backpropagation, i.e., autograd in the case of PyTorch.

For the encoder, however, we require variational inference, as its gradients depend on the stochastic output of the distribution $p(z; E(w_E))$. Here, we can use one of our gradient estimator approaches with Monte-Carlo sampling to get gradient estimates for w_E . Once again, we fix the parameters w_D , and view $f(D(z; w_D))$ as a loss function which we call $f_D(z)$ for convenience, such that

$$\arg \max_{w_E} J(w_E) = \int \int p(z; E(x; w_E)) f_D(z) dz dx,$$

is the objective for the encoder, which is already in the form we established in section 1.2. This allows us to use the three stochastic gradient estimators to get gradient estimates for w_E .

To implement VAEs in our framework, we only require a new model class. This class contains the encoder, and decoder networks, as well as the distribution family used. When encoding, it instantiates the distribution using the output of the encoder. Then, we can ask it for a reconstruction of a sample of the distribution, for which it uses the decoder. Since the encoder and decoder are separate submodules of the model, they are already separated for the training. A code snippet of the core training steps is shown in figure 3.3. We use this implementation to conduct various experiments on the Variational Auto-Encoder in section 4.2.

4 Gradient Analysis

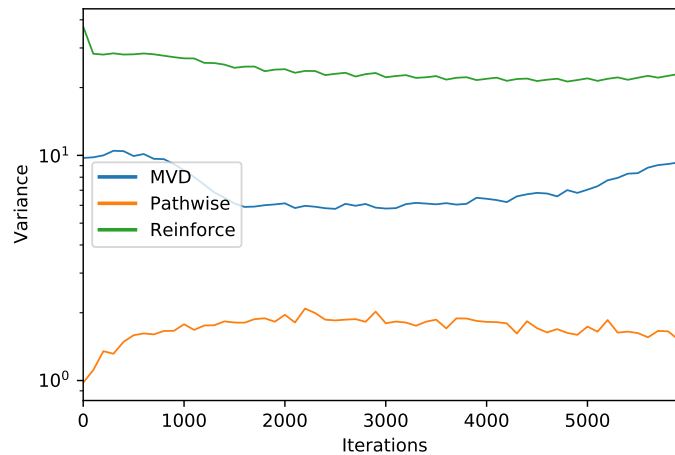
In this chapter, we present the results of our experiments concerning the three gradient estimators. We focus on results showing characteristics of the estimators to make some distinctions between them. For this work, we are only comparing Monte-Carlo estimators. Hence, we look at metrics like the variance of the estimators, and their performance in terms of results and time required for training. To conduct the experiments, we use the framework presented in chapter 3. In experiments using normal distributions, we parameterize the distributions with the logarithm of the standard deviation, and a diagonal covariance matrix for multivariate normal distributions.

4.1 Bayesian Logistic Regression

The model for Bayesian logistic regression does not have as many moving elements around the distribution itself. That means, there are little choices in terms of hyperparameters, or model complexity. The upside of this task is that we can easily compare the estimators without risking a choice of model that lends itself more towards one of them. This task stands in contrast to a variational auto-encoder, where we could imagine a certain network architecture affecting the performance of the estimators differently. While this gives us results that apply in most situations for the task, it also means that we expect only small differences in the performance of the estimators. To make clear that the same observations do not hold in more complex scenarios, we analyze the performance of the estimators on variational auto-encoders in section 4.2. In this section, we rather focus on the variance of the estimates, as this metric characterizes the estimators more generally.

In figure 4.2, we find that the performances of the estimators are indeed very close. While there are marginal differences, they would not be enough to indicate superiority of any one estimator. However, looking at the variances in figure 4.1, we see that the score-function

Figure 4.1: Variances of Bayesian Logistic Regression with $m = 30$ latent dimensions, trained on the breast cancer dataset [10]. The optimizer used is SGD with learning rate $\alpha = 10^{-3}$ and batch size $B = 32$.



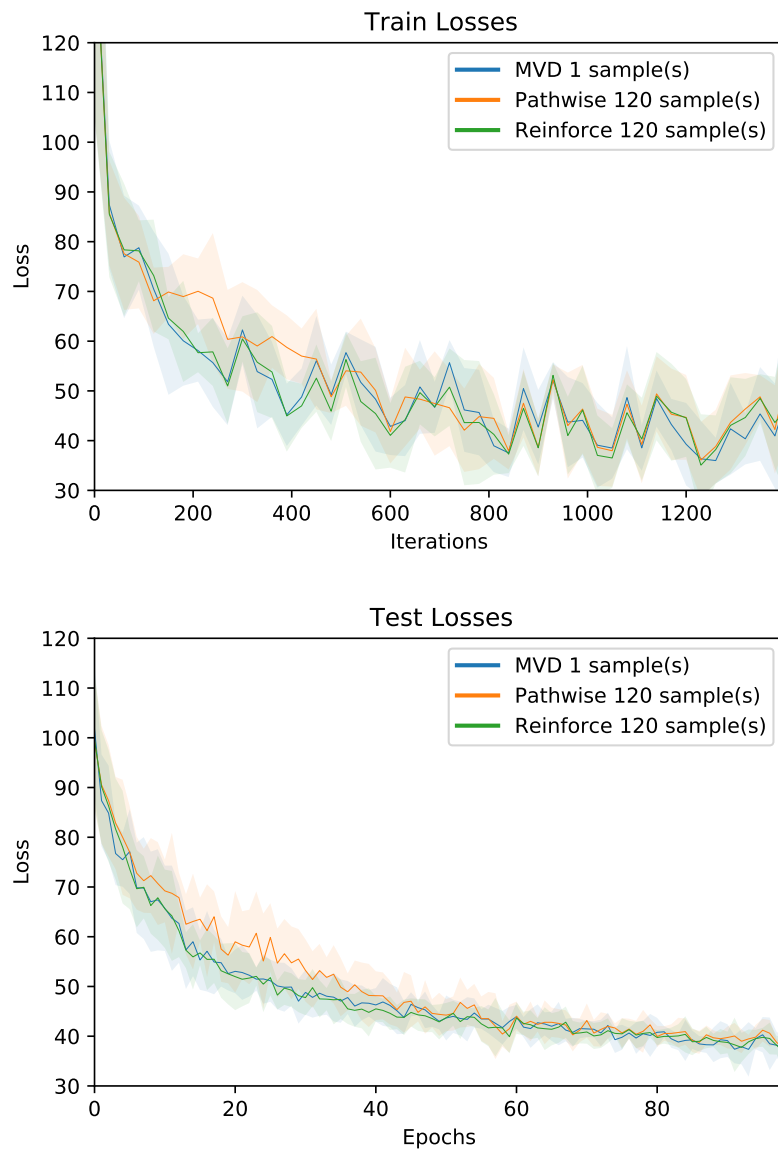
estimator has by far the highest variance. As expected, the pathwise estimator has the lowest variance, while the measure-valued estimator is between the other estimators.

4.2 Variational Auto-Encoder

As discussed in section 3.4, we conduct experiments using variational auto-encoders with fully connected, convolutional, and recurrent encoder/decoder networks. We train our models on the MNIST [41], Omniglot [40], ECG5000 [20] datasets.

MNIST Dataset. In figure 4.3, we show the losses of training a variational auto-encoder on the MNIST dataset. We find that the performances of the score-function and measure-valued estimators are similar given the score-function estimator is allowed the same number of function evaluations. The pathwise estimator achieves the best test loss on this dataset. However, looking at the reconstructions in figure 4.4, it seems unlikely to be able

Figure 4.2: Train and test losses of Bayesian Logistic Regression with $m = 30$ latent dimensions, trained on the breast cancer dataset [10]. The optimizer used is SGD with learning rate $\alpha = 10^{-3}$ and batch size $B = 32$.



to identify an obviously better choice. There are clearly some artifacts, especially in the two fives and the six for all reconstructions.

However, the performances change dramatically when considering the process time instead of iterations. Figure 4.5 shows the losses with respect to the process time. For this plot, we track only the time required for sampling, evaluating the function, and calculating the gradient estimate from the loss evaluations. This means that we get little noise from other calculations, letting us compare only the computations directly influenced by the respective estimators. Also, we compare process times, which do not correspond to a fixed unit of time, but should only be compared to each other. We cut off the plot once the first estimator was done, which in this case was the pathwise estimator with one sample. We can see that in this comparison, the score-function estimator falls behind irrespective of sample sizes. Also, the pathwise estimators are very close, even favoring the one sample estimator in the end. This in itself shows that for shorter runs, it is not worth it to use a high number of samples. The additional computation required does not pay off in these scenarios. Also, a higher variance at the start of the training may allow the estimators to escape poor local minima more easily. Considering the performance of the measure-valued estimator, we see that it now shows a clear advantage over the score-function estimator. A possible reason for this might be slightly better optimization of the measure-valued estimator in our implementation. However, it also profits from reusing the same random sampling for its coupling, which saves on random number generation. As for the computation of the gradient, it requires less backpropagation than the other estimators, as it uses no explicit gradient terms. We can conclude that considering process time, the measure-valued estimator performs worse than the pathwise estimator. For the score-function estimator, we find that a large number of samples slows down the estimator a lot. It would be useful to analyze the root of this problem, and re-evaluate the comparison if better optimization is possible, though this is out of the scope of this thesis.

Omniglot dataset. For a more complex task, we train a VAE using convolutional layers on the Omniglot dataset [40]. Since the dataset contains 105x105 images of a larger number of symbols, the reconstruction requires a much more complex model for reasonable results than MNIST. Our results can be seen in figure 4.6.

In this more complex task, we find that the score-function estimator becomes very unstable. This also becomes clear in the variances displayed in figure 4.7 we recorded during the trainings on MNIST and Omniglot. Looking at the MNIST dataset, we see that the variance of the score-function estimator is initially very high, but it is at least stable once the baseline is established. However, in the case of Omniglot, the score-function estimator becomes

Figure 4.3: Train and test losses of a VAE with two fully connected 400-dimensional layers in both the encoder and the decoder with ReLU activations, $m = 20$ latent dimensions, trained on the MNIST dataset. The optimizer used is ADAM with learning rate $\alpha = 10^{-3}$ and batch size $B = 128$.

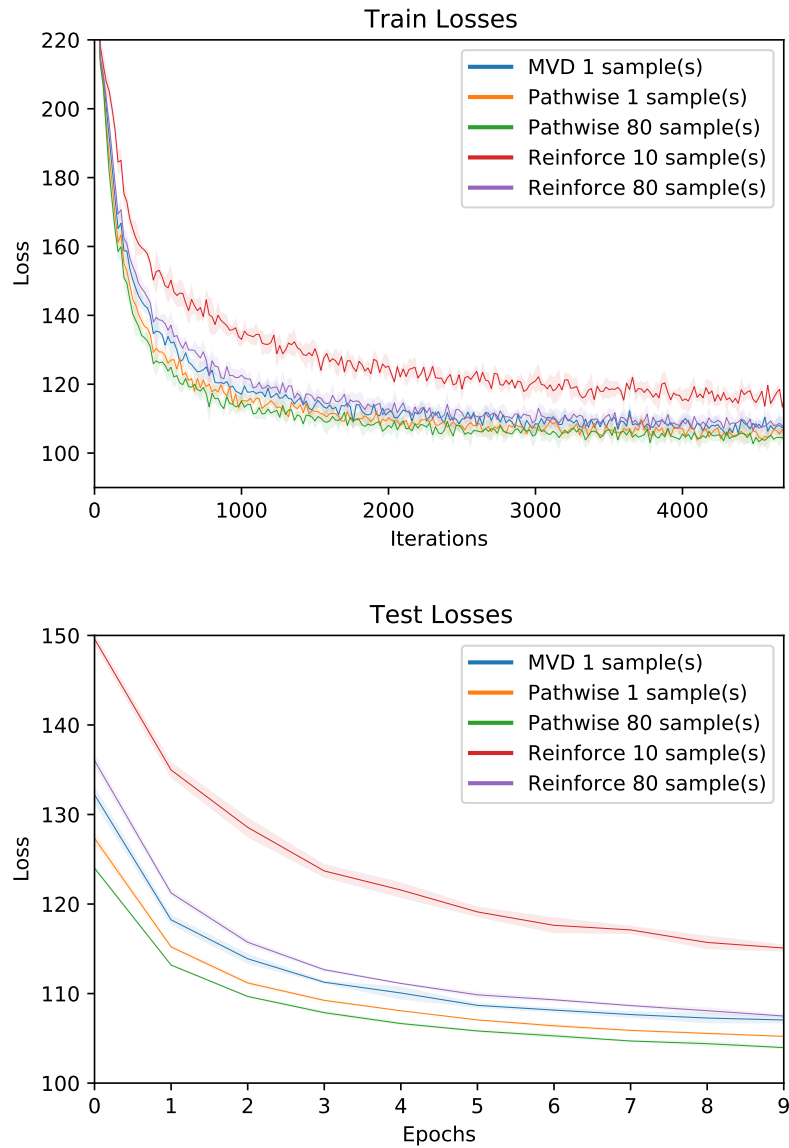
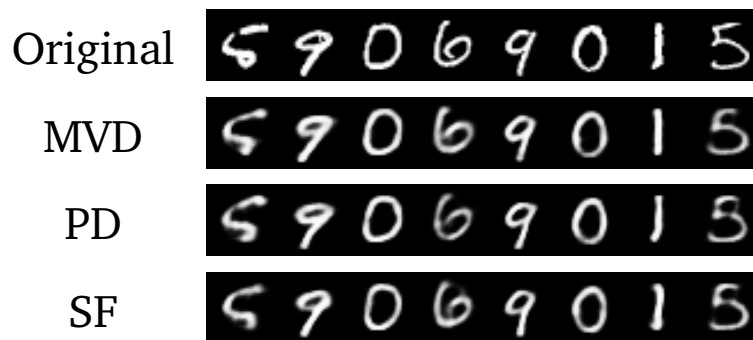


Figure 4.4: Reconstructions by VAEs generated after the training shown in figure 4.3. Here, MVD means the measure-valued estimator, PD means the pathwise estimator, and SF means the score-function estimator. All reconstructions are generated using models trained with the highest number of samples.



unstable after about five epochs. On the other hand, the measure-valued and pathwise estimators behave similarly to the MNIST training, suggesting that they remain stable even for more complex tasks.

ECG5000 dataset. A third common architecture for variational auto-encoders are recurrent neural nets [11]. For this architecture we use the ECG5000 dataset [20], a time-series classification dataset. Since the data is ordered in a temporal manner, the model has to incorporate its previous reconstructions into the next reconstruction. The architecture we use for this allows us to train recurring modules of the same structure for which the previous output is an input to the next module. We use long short-term memory (LSTM) [29], gated recurrent unit (GRU) [9], and vanilla RNN modules.

Figure 4.8 contains the losses on the ECG5000 dataset. In this case, we can see that the estimators perform similarly. The score-function estimator is still less stable than the other two estimators, but it does not diverge like in the case of convolutional architectures. For LSTMs, we find that the pathwise and measure-valued estimators perform very similarly. However, for GRU modules, the pathwise estimator yields significantly better results. Vanilla RNN modules represent an interesting case here, because they suffer more from vanishing and exploding gradients [29]. LSTMs and GRUs aim to mitigate this problem, and we can also see this in our losses. Since the pathwise estimator relies on the gradient

Figure 4.5: Training loss of a VAE with two fully connected 400-dimensional layers in both the encoder and the decoder with ReLU activations, $m = 20$ latent dimensions, trained on the MNIST dataset. The optimizer used is ADAM with learning rate $\alpha = 10^{-3}$ and batch size $B = 128$. The tracked time in the horizontal axis does not correspond to a real-time measure, but reflects the relative processing time required by the gradient estimator computations only. At the end of the graph, the pathwise estimator with 1 sample completed 10 epochs, while the others are cut off at that point.

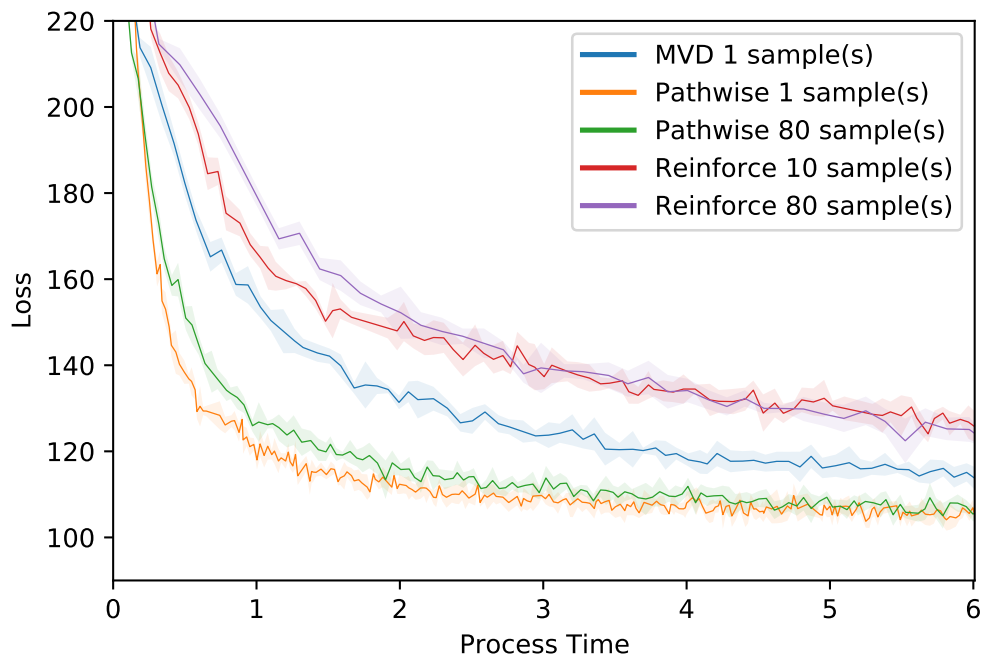


Figure 4.6: Train and test losses of a VAE with convolutional networks, $m = 64$ latent dimensions, trained on the Omniglot dataset. The optimizer used is ADAM with learning rate $\alpha = 10^{-3}$ and batch size $B = 144$.

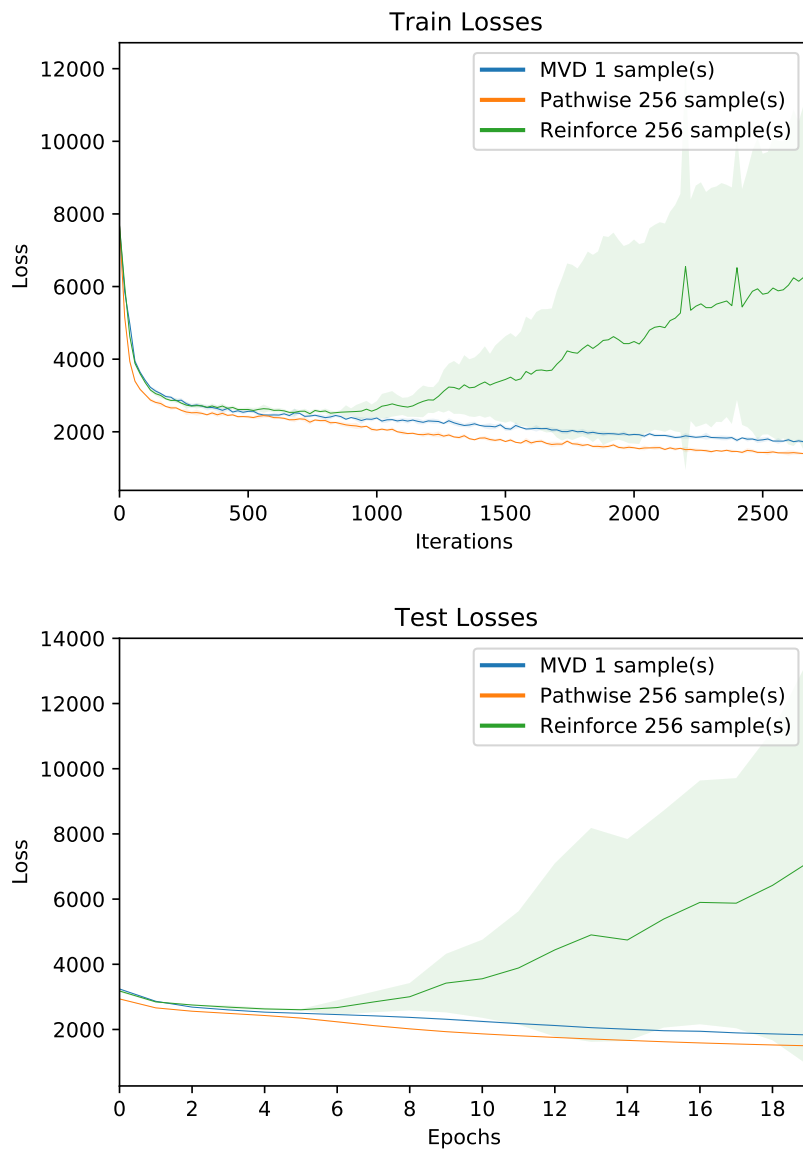
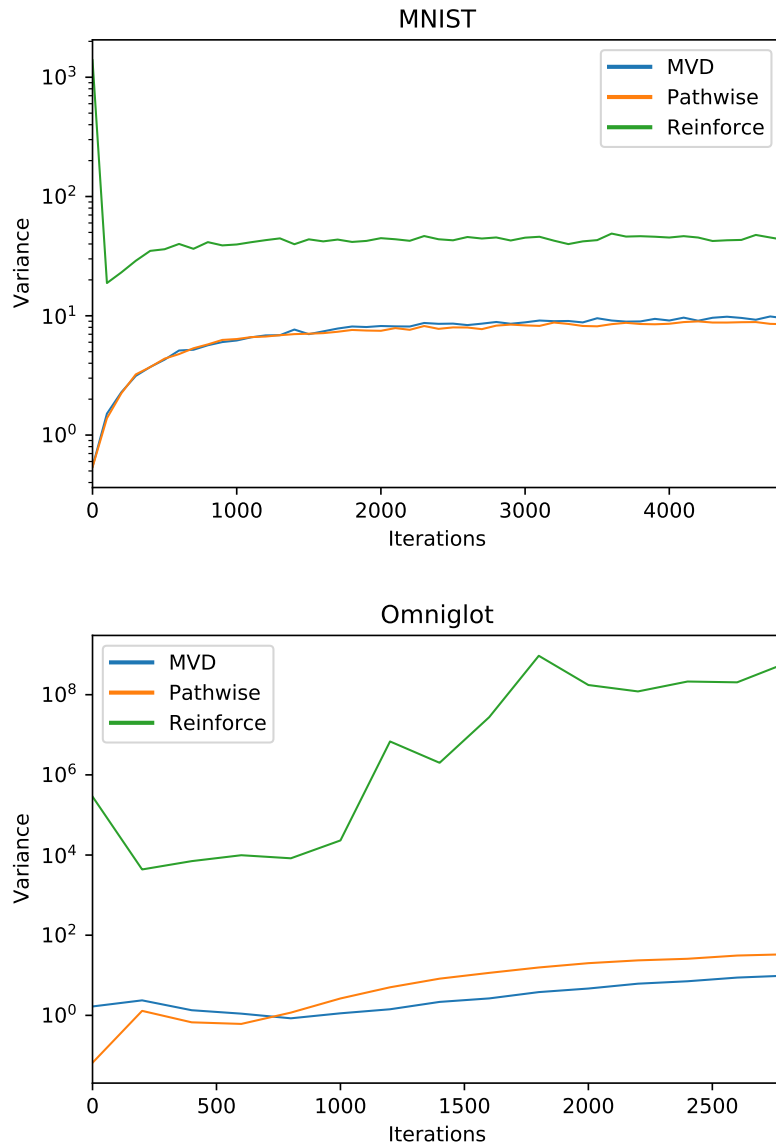


Figure 4.7: Variances of the estimators while training VAEs on MNIST and Omniglot datasets. The variances are computed during training, using the same experiment set-up as figures 4.3 and 4.6 respectively. The y-axis of both graphs is shown in logarithmic scale.



which is propagated through the whole computation, we observe a higher variance in that case.

Sinusoidal data. In the results for the ECG5000 dataset, we have seen an indication of exploding gradients while training the pathwise estimator. As an attempt to isolate the issue, we train the estimators on synthetic data in the form of noisy sine functions. We show the results using the three module types in figure 4.9. From our experiments, it seems like the vanilla RNN modules are unable to model the data, independent of the estimator used. With LSTMs, it seems like the estimator does not produce much of a difference in the results as well. The biggest difference we observe is the large spike in variance using the pathwise estimator with GRU modules. Once again, this happens due to the estimator relying on the gradient propagation through the whole calculation. However, this is not enough to completely destabilize the learning process. Hence, our results do not suggest that the pathwise estimator is a worse choice with RNN models. We should note that the score-function estimator is competitive to the others in this task as compared to our other experiments.

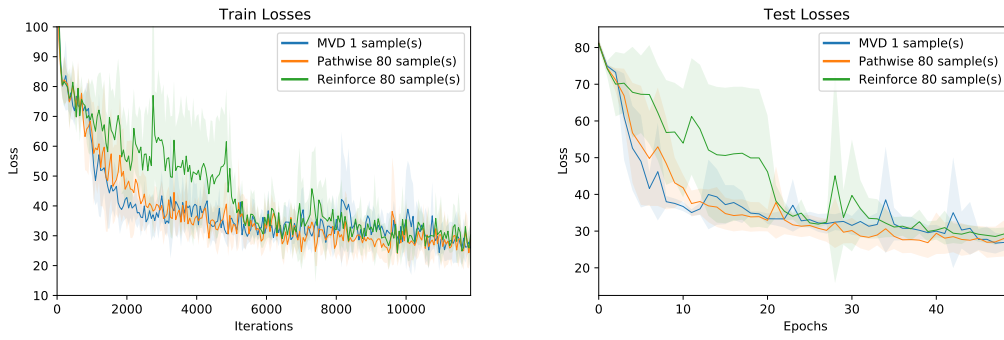
4.3 Randomized Convex Combinations

We have shown in section 2.4 how we can combine two unbiased estimates into one unbiased estimate. In this section, we use this to create a convex combination of the score-function and measure-valued estimator.

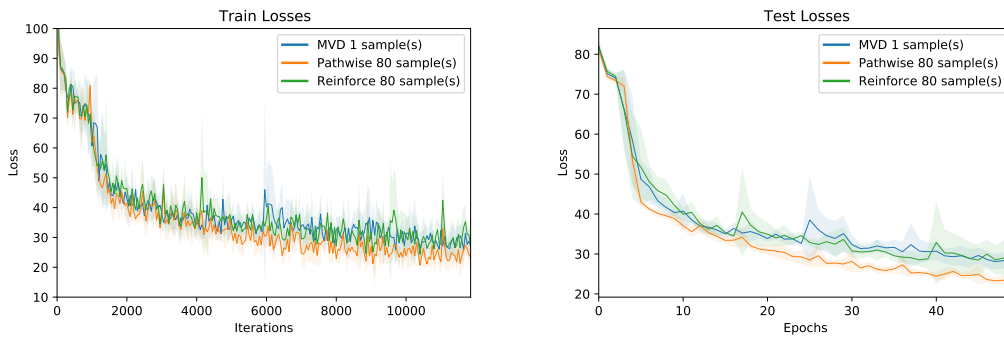
For the following argument, we refer to the notation established in section 2.4. Let \hat{T}_1 be the score-function estimator and \hat{T}_2 be the measure-valued estimator. We choose this combination, because the characteristics of these estimators work well together. As we discussed in chapter 2.2, the score-function estimator gives us a high variance estimate, but is more flexible in terms of sample size. On the other hand, the measure-valued estimator presented in section 2.3 is a low variance estimate, but it requires loss evaluations linear to the number of parameter dimensions of the latent distribution. However, if we consider a convex combination, we might be able to reduce the variance of the score-function estimate by replacing it with a partial measure-valued estimate. To do this, we choose the factor c as a vector $c \in [0, 1]^{\mathcal{D}}$. This means, we completely replace the score-function estimate by a measure-valued estimate in the dimensions i where $c_i = 0$.

Figure 4.8: Train and test losses of a VAE with recurrent LSTM, GRU, and vanilla RNN, $m = 20$ latent dimensions, trained on the **ECG5000 dataset**. The optimizer used is ADAM with learning rate $\alpha = 5 * 10^{-4}$.

LSTM



GRU



Vanilla RNN

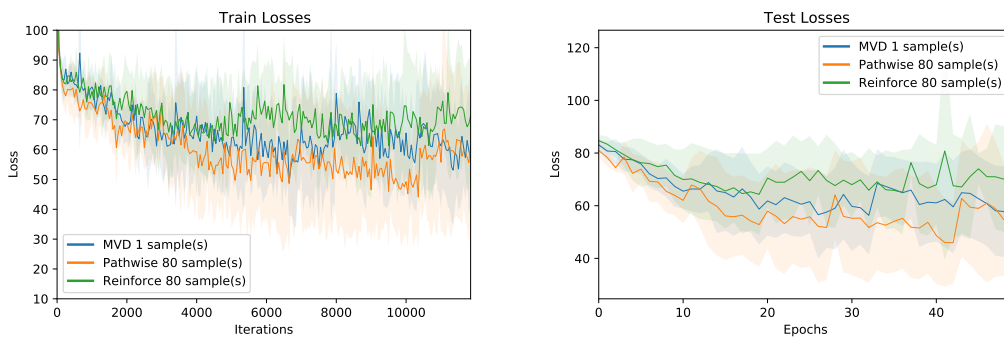
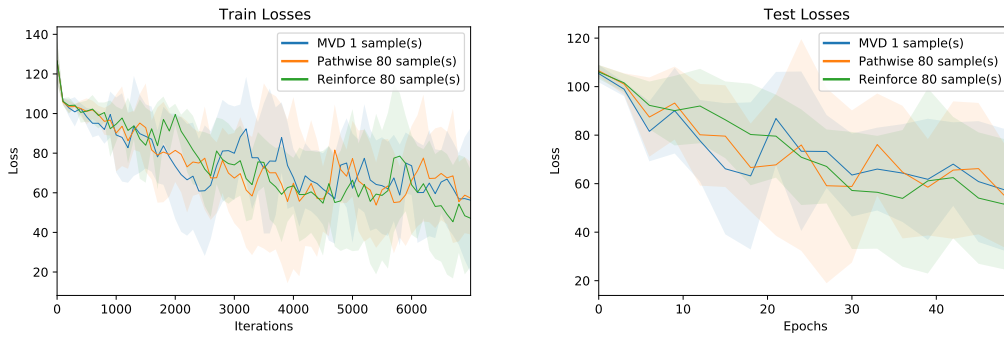
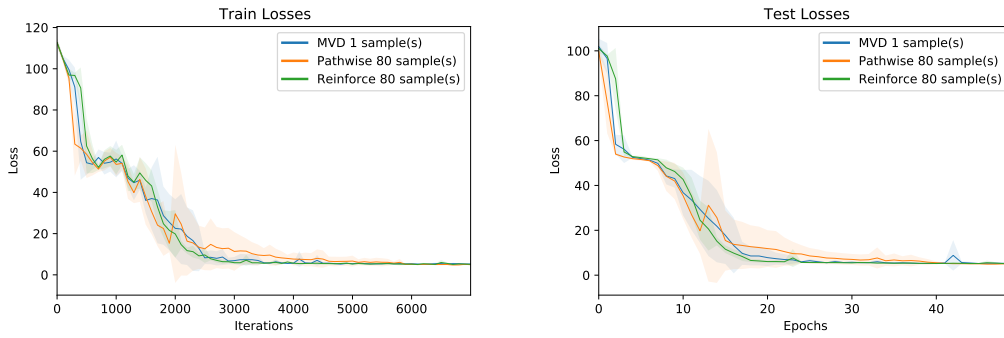


Figure 4.9: Train and test losses of a VAE with recurrent LSTM, GRU, and vanilla RNN, $m = 4$ latent dimensions, trained on the **synthetic sinusoidal dataset**. The optimizer used is ADAM with learning rate $\alpha = 5 * 10^{-4}$.

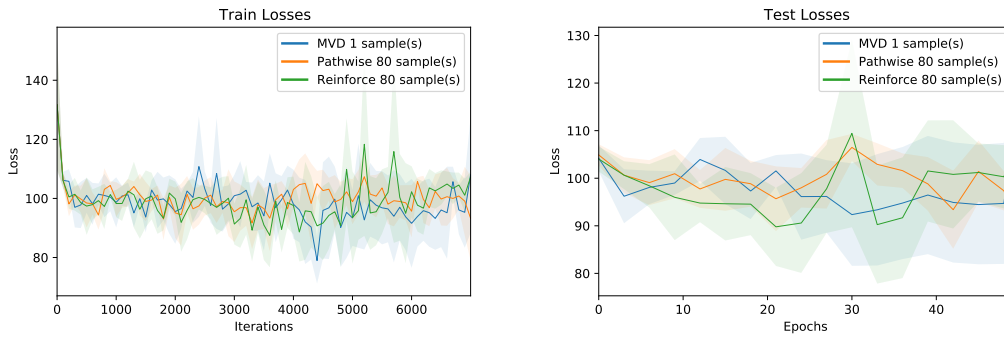
LSTM



GRU



Vanilla RNN



To evaluate this approach, we implement the combination in our framework and train the same model as we used in section 4.2. We also use the MNIST dataset and the same hyperparameters. This allows us to directly compare the performances with varying combinations. To see the general performance of the approach, we evaluate all combinations using the same number of loss evaluations. Hence, we use $4\mathcal{D} - 2|c|$ for a normal distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with diagonal covariance to calculate the number of samples for the score-function estimate. For the measure-valued estimate, we always use the minimal number of loss evaluations, calculating one estimate only. We always replace the same dimensions in the mean and covariance gradients. In total, we end up with $4\mathcal{D}$ evaluations for all combinations.

The losses are shown in figure 4.10. Unfortunately, all combinations perform worse than purely using the score-function estimator. For this reason, it does not seem like we can improve upon the score-function estimate using a randomized convex combination with measure-valued estimates. The upshot, however, is that we can indeed produce stable estimators using such a naive combination. Since we generate the full score-function estimate before we need to decide which dimensions we want to replace by a partial measure-valued estimate, there is a lot of design space in this approach. At this point, we don't take any additional information about the estimate into account. For example, if we were able to incorporate an upper bound on the variance of the score-function estimate in each dimension, a replacement strategy could focus on the dimensions with the highest estimated variance.

Another thing to note here, is that the performance using the measure-valued estimate for five dimensions yields the closest performance to the full score-function estimator. In figure 4.11, we show how these estimator behave when we gradually lower the sample size of the score-function estimate. It becomes clear in this case, that the randomized convex combination does not yield better results than a pure score-function estimator. The convex combination with 60 evaluations is about as good as a pure score-function estimator with 40 evaluations here. In section 5.3, we discuss the implications of this observation in more detail.

4.4 Discrete Distributions

In this section, we evaluate the characteristics of the estimators on discrete mixture models. As we have shown in section 2.5, all three estimators can be applied to finite, discrete Gaussian mixture models.

Figure 4.10: Losses of training varying combinations of measure-valued and score-function (MVSF) estimators. The hyperparameters as well as the model are the same as in figure 4.3. We only vary how many dimensions of the score-function estimate are randomly replaced by a measure-valued estimate. The estimators are set up such that they use the same number of loss evaluations, which in this case are always 80 in total. In the legend, a MV / b SF stands for a dimensions of measure-valued gradient estimates, and b dimensions of score-function gradient estimates.

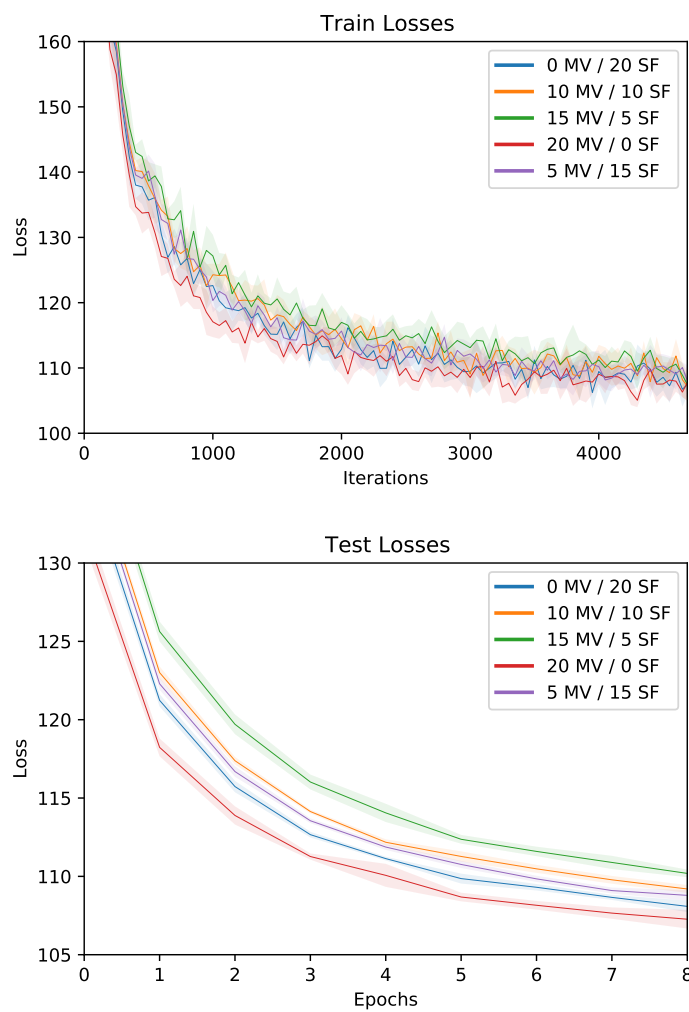
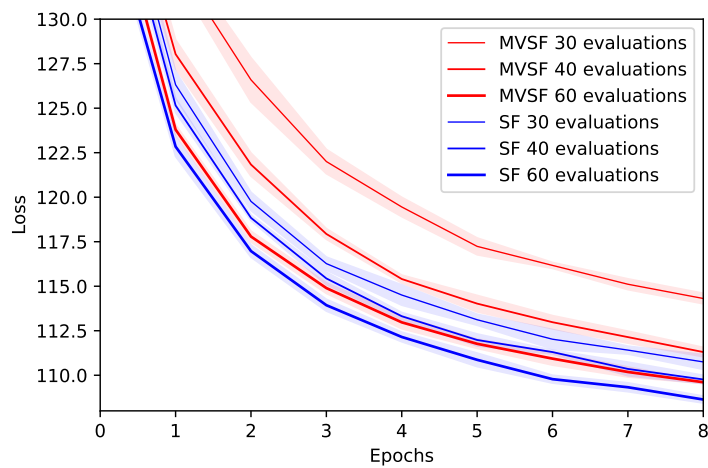
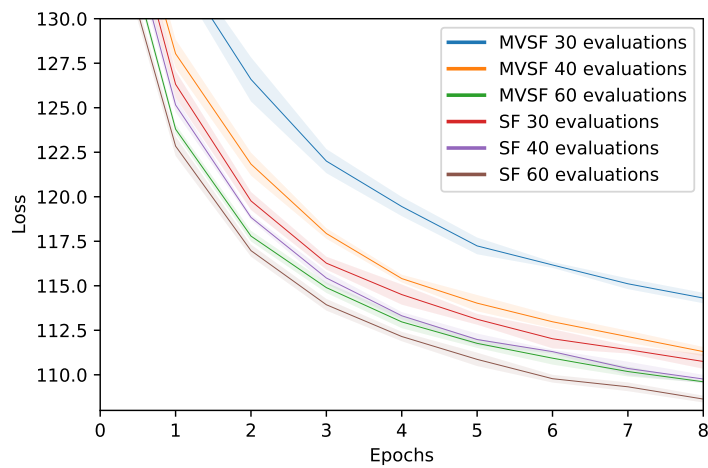


Figure 4.11: Losses of training varying sample sized of the randomized, convex MVSF combination estimator. In this figure, all MVSF combinations replace 10 of 40 parameter dimensions by measure-valued estimates. **Both plots show the same data**, but with different color schemes. In the second plot, we show the MVSF estimator in red and the pure SF estimator in blue to be able to directly compare the estimator types. The hyperparameters as well as the model are the same as in figure 4.3. The estimators are set up such that they use the same number of loss evaluations, which in this case are always 80 in total.



To get a first impression on the estimators in simple scenarios, we start by using only categorical distributions. For all of these experiments, we use categorical distributions with k categories, and start with equal probabilities, i.e., $p(x_i) = 1/k$, $i = 0, \dots, k$, unless stated otherwise. We convert the samples to integers and divide them by k , such that the samples are in $[0, 1]$ to display varying k using the same area of the loss. As we allow the trained parameters w to be in \mathbb{R} , we use a softmax-parameterized categorical distribution, which normalizes the probabilities. Note that due to the construction of the framework, we ensure that the gradient flows through the softmax parameterization for all three estimators. We always use k samples for the score-function and pathwise estimators to have a fair comparison to the k loss function evaluations required by the measure-valued estimator. To make sure we stay in the same area of the loss function, we normalize the categorical outputs i by dividing by $k - 1$, such that $i/k-1 \in [0, 1]$.

Small Numbers of Categories. Figure 4.12 shows how the probability mass functions are adapted when training with the three algorithms. We can see that the measure-valued estimator on the left gives the most aggressive estimate in this case.

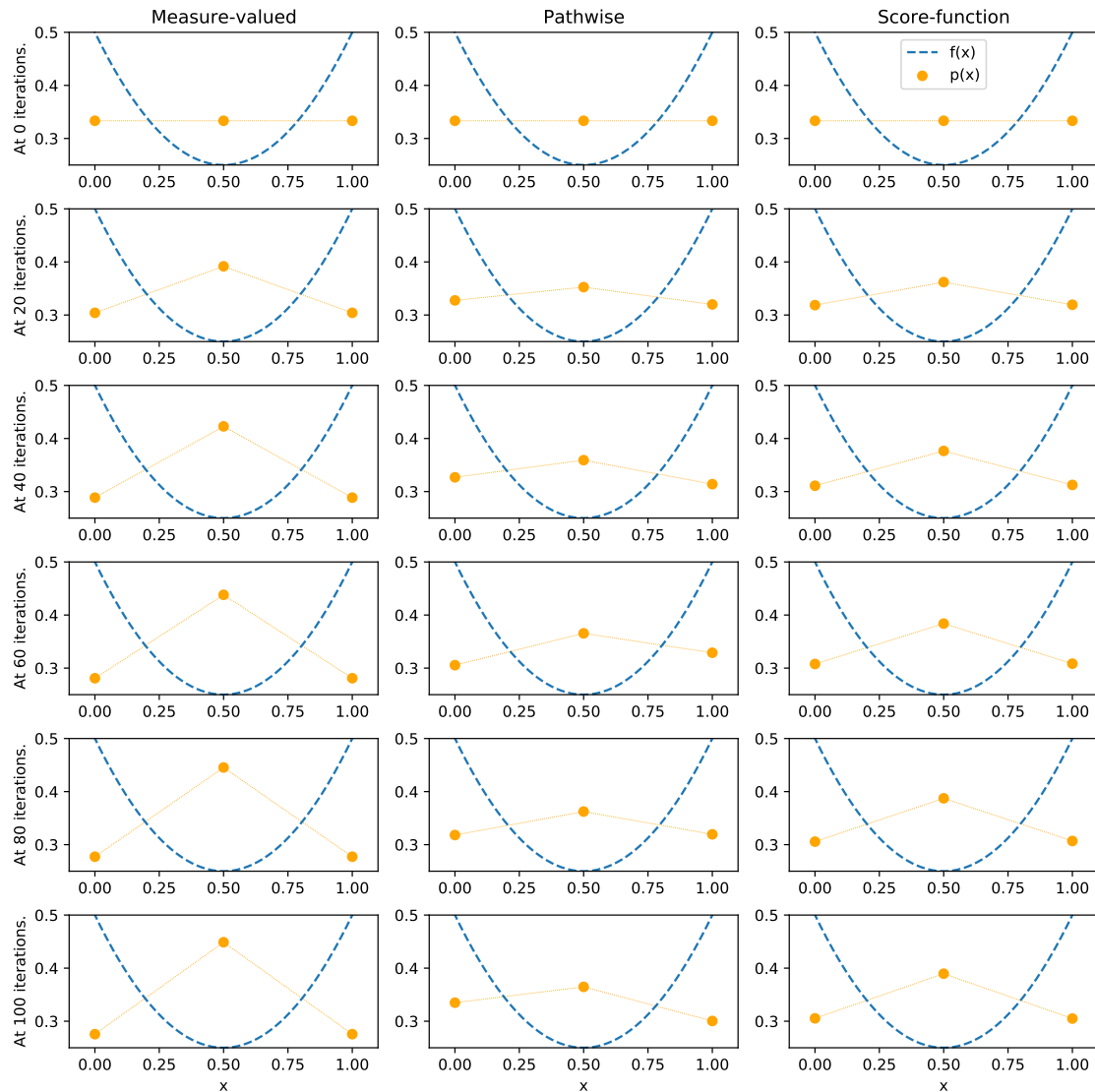
As for the pathwise estimator, the probabilities only change slowly. Also, especially in the lower probabilities, we see a slight bias, even though the losses are equal for both of them. These characteristics stem from the Gumbel softmax. To show this more clearly, we can look at a larger number k .

The score-function estimator performs very similar to the pathwise estimator here. We find that the difference surfaces in larger k as well.

Large Number of Categories. As we increase the number of categories, some characteristics become more clearly visible. In figure 4.13, we see the probability mass functions of categorical distributions with $k = 100$ categories. Looking at the measure-valued estimator, we still see promising results. The estimate is unbiased and shows the clearest peak of all three estimators.

The behavior of the pathwise estimator is a bit more visual than with three categories. We can still see the same two issues, namely, the slow learning and the slight bias. Unfortunately, the bias is inherent to the Gumbel softmax [33]. However, we can influence the learning rate by choosing a lower temperature, as shown in figure 4.14. As we can see, the temperature controls the aggressiveness of the estimate through altering the uniform samples. With lower temperatures, the model leans towards the general area of interest

Figure 4.12: Probability mass functions while training a categorical distribution with $k = 3$, using the SGD optimizer with learning rate $\alpha = 0.1$, using softmax parameterization. The loss function is a simple, shifted parabola $f(x) = (x - 0.5)^2$, shown as a dashed blue line. We train for 100 iterations, plotting every 20 iterations from top to bottom.



more quickly, but the bias becomes more pronounced. Since the pathwise estimator is not the focus of this thesis, we discuss this only with regards to the measure-valued estimator.

As for the score-function estimator, we get a similar shape to the measure-valued estimator. However, the higher variance of the gradients causes many small errors, and the resulting expected loss is not as low as the measure-valued estimator result.

Multiple Modes. We looked at a simple parabola to spot characteristics of the estimators, but in reality that is not what we would like to use mixture models for. In general, we would like to be able to represent multiple modes of the data, e.g., when encoding digits for the MNIST dataset. To reflect this in a toy problem, we use a sinusoidal loss function. We show some intermediate probability mass functions in figure 4.16, since they show the differences of the estimators well.

In these results, we see the bias of the Gumbel softmax very clearly. The peaks do not match the minima of the loss function, and the distribution becomes biased towards zero.

On the other hand, we see that the score-function estimator still estimates the general direction of the gradient correctly. In comparison to the measure-valued estimate, the results are still not as clear.

The approximate expected values during trainings are shown in figure 4.16, initializing the parameters with equal probabilities, and figure 4.17 with random initialization. In general, the measure-valued gradient performs so well here, because it is not really estimating by sampling. Instead, this gradient is based on an enumeration of the whole domain, comparable to marginalizing. Obviously, this is only possible for distributions with finite domains. Still, for finite discrete mixture models based on categorical distributions, the measure-valued gradient gives us the best results for the gradient of the categorical distribution. We should note in this case that we can combine different estimators for the mixture, e.g., using measure-valued estimates for the categorical selector, and pathwise estimates for the Gaussian components.

Discrete Distributions With Infinite Support. We established that the measure-valued derivative is a good choice for stable results in the context of finite discrete mixture models. However, we also find that this result is not surprising, since it actually enumerates the support of the distribution instead of sampling, and is not stochastic for this reason. While this yields a reliable gradient for the distribution, we have to evaluate whole support of the distribution, which can be a downside for very large supports.

Figure 4.13: Probability mass functions while training a categorical distribution with $k = 100$, learning rate $\alpha = 0.1$, using softmax parameterization. The loss function is a simple, shifted parabola $f(x) = (x - 0.5)^2$, shown as a dashed blue line. We train for 1000 iterations, plotting every 200 iterations from top to bottom. Since the density of categories is so high here, we plot them as lines, even though they are still discrete.

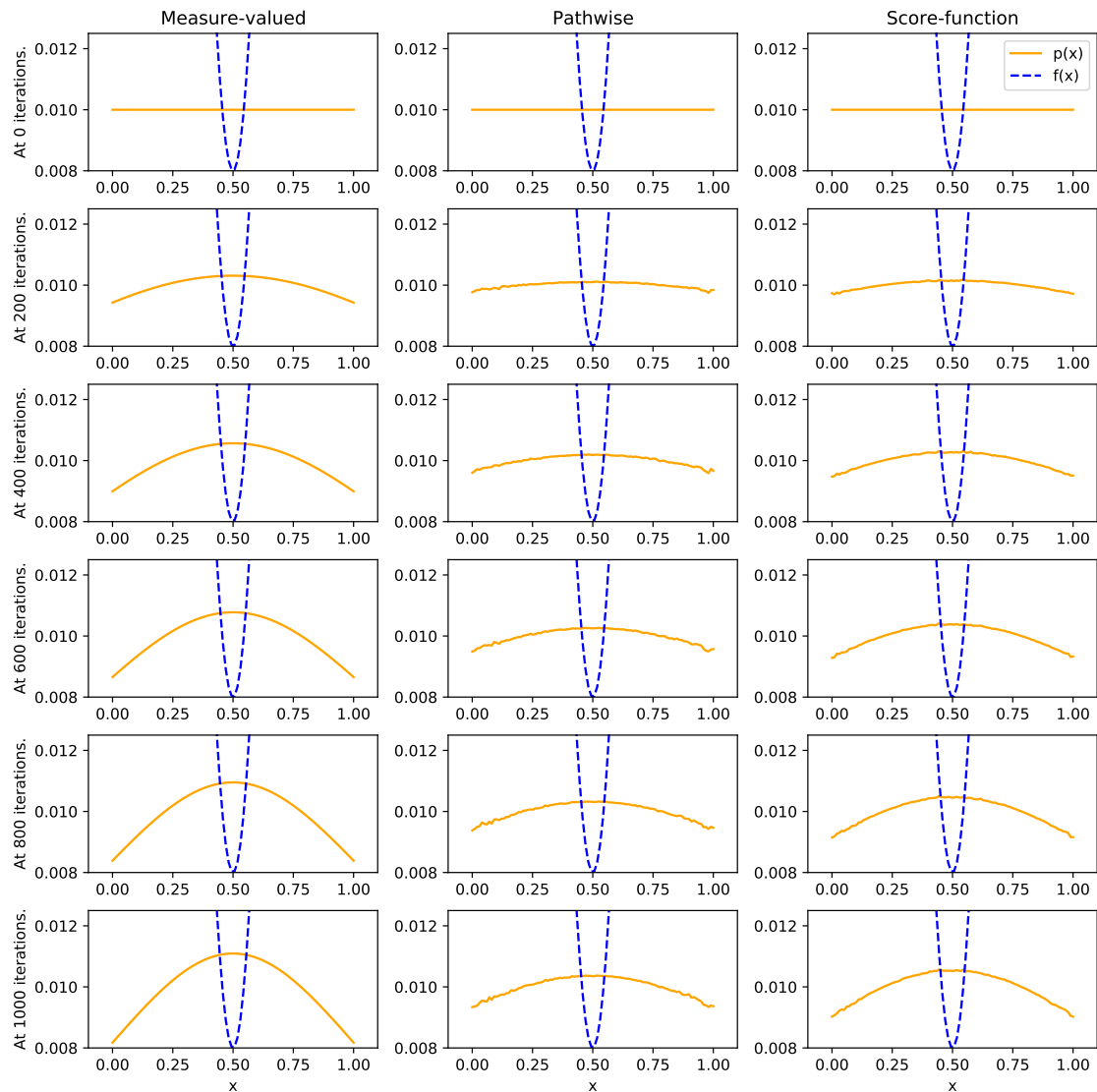


Figure 4.14: Evaluating different Gumbel softmax temperatures τ training a categorical distribution with $k = 100$, learning rate $\alpha = 0.1$, using softmax parameterization. The loss function is a simple, shifted parabola $f(x) = (x - 0.5)^2$, shown as a dashed blue line. We train for 1000 iterations, plotting every 200 iterations from top to bottom. Since the density of categories is so high here, we plot them as lines, even though they are still discrete.

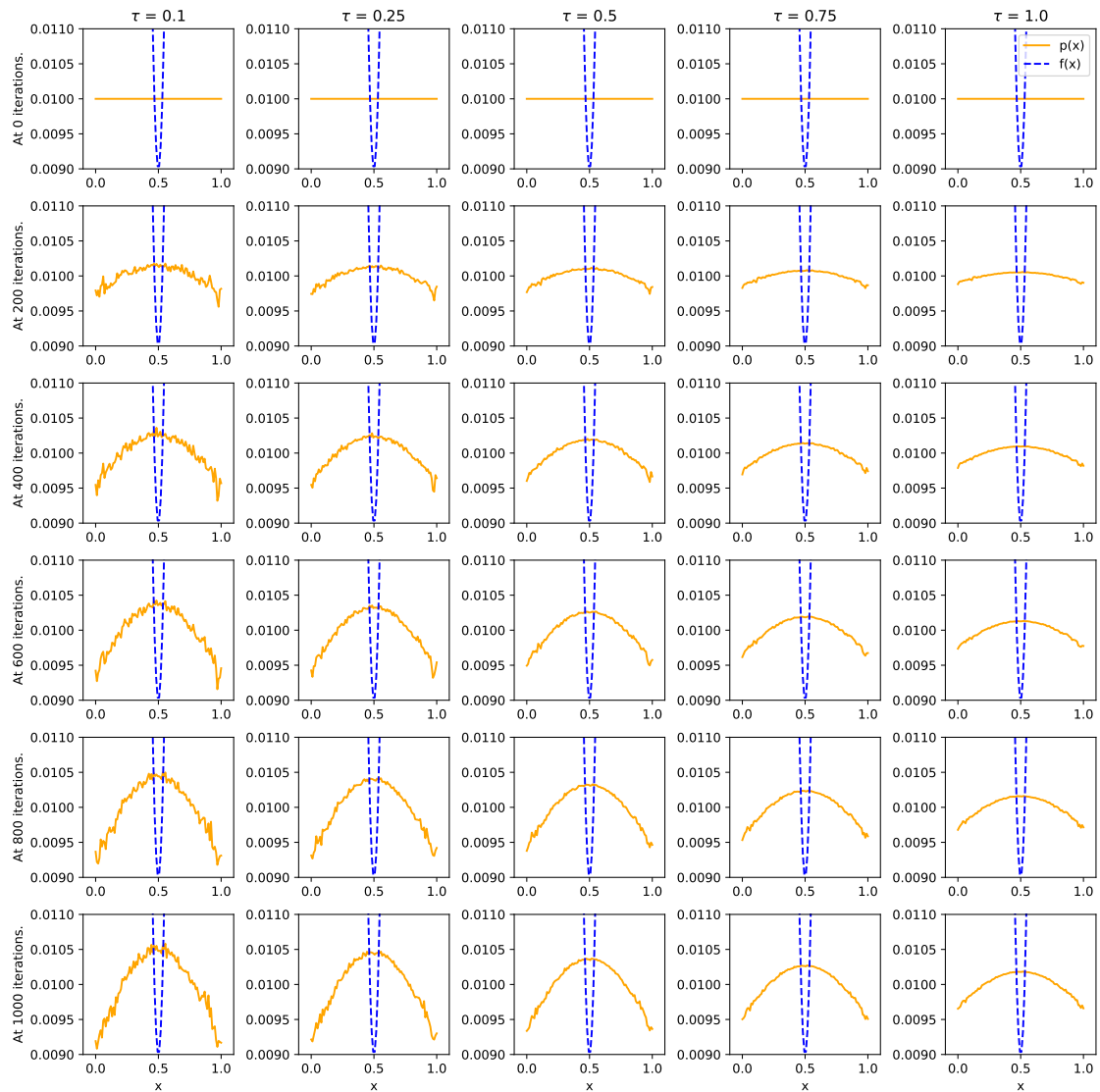


Figure 4.15: Probability mass functions while training a categorical distribution with $k = 100$, learning rate $\alpha = 0.1$, using softmax parameterization. The loss function is a sine $f(x) = \sin(4\pi x)$, shown as a dashed blue line. The loss function is shifted and squished such that we can see the extrema in the plot, but it was unaltered during training. We train for 1000 iterations, plotting every 200 iterations from top to bottom. Since the density of categories is so high here, we plot them as lines, even though they are still discrete.

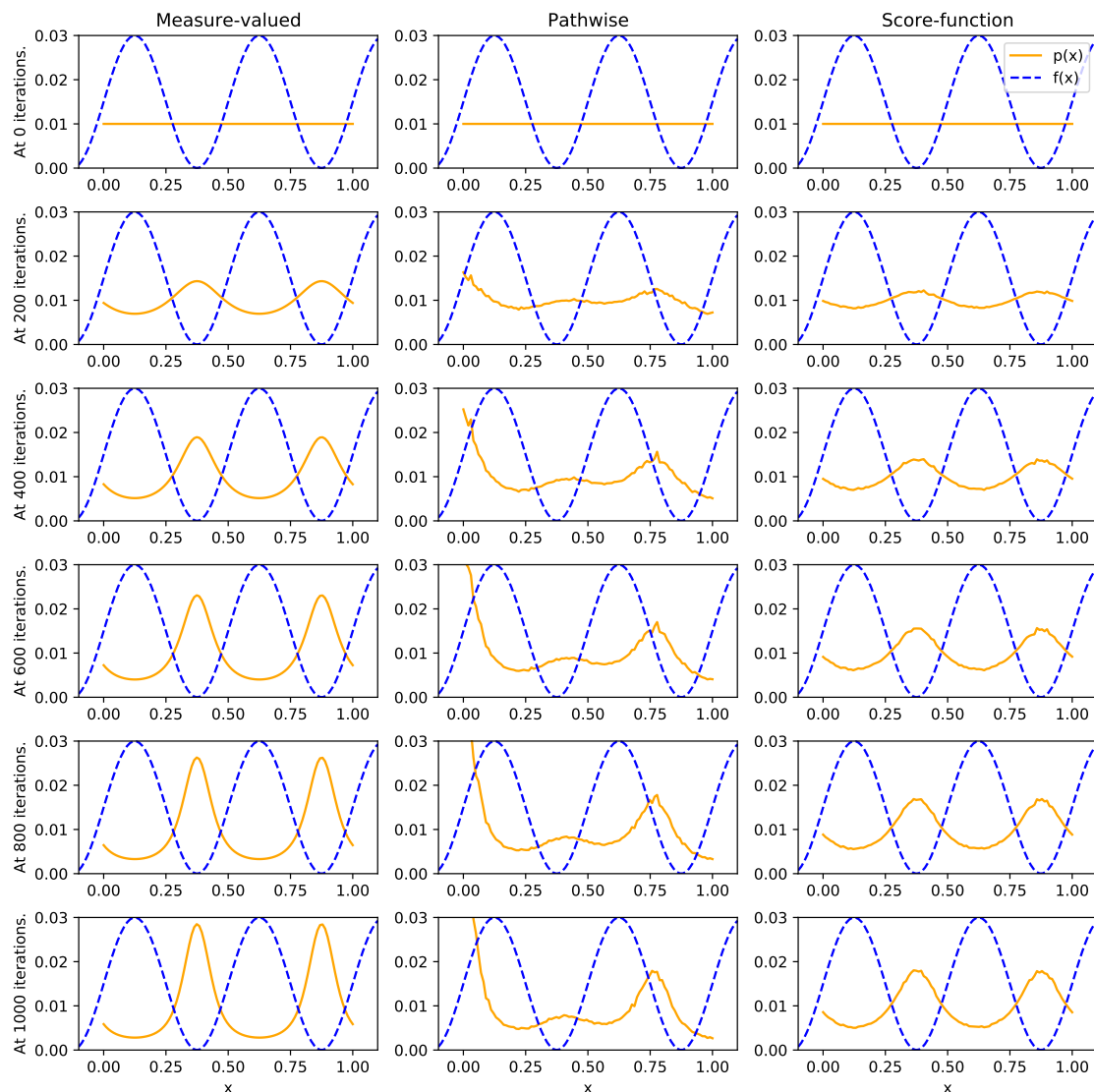


Figure 4.16: Approximate expected values while training a categorical distribution with the same set-up as in figure 4.15. We initialize the class probabilities equally as $1/k$.

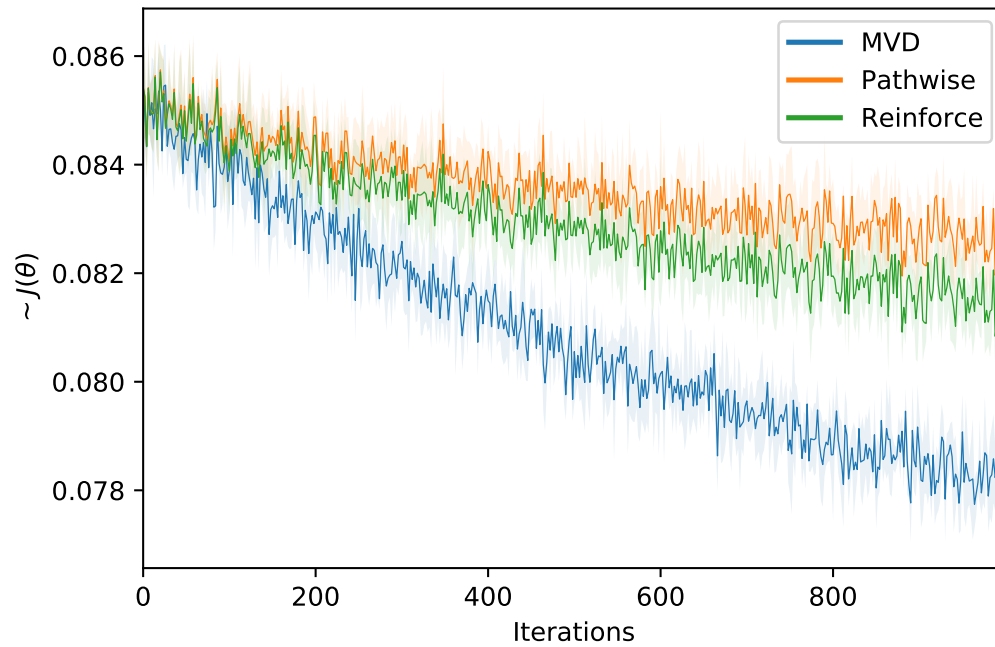
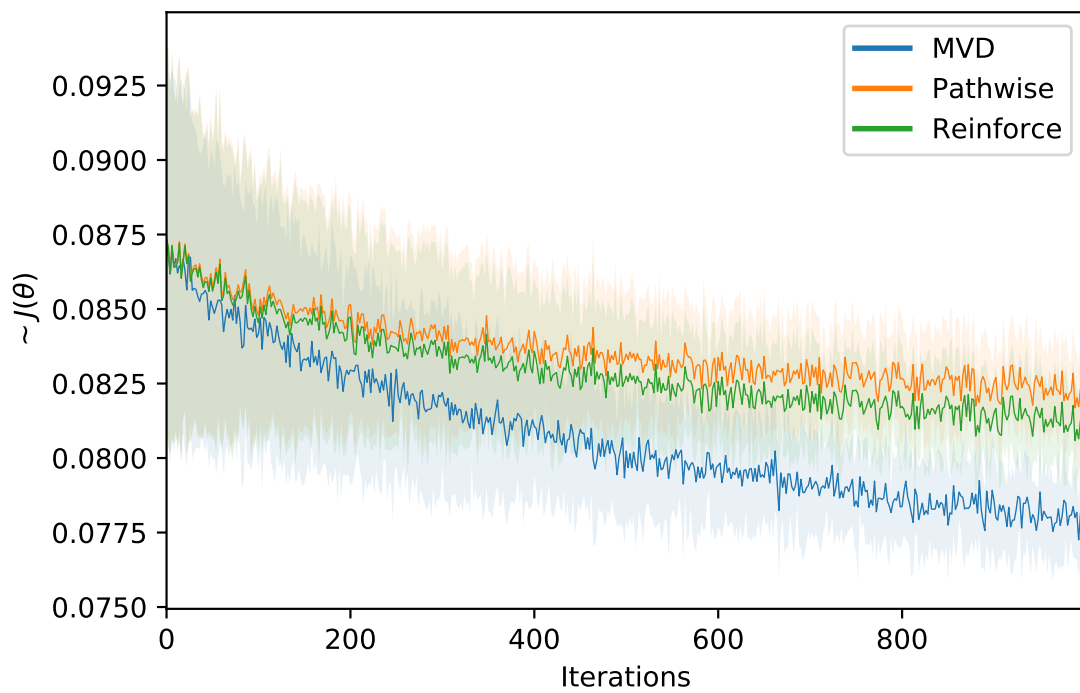


Figure 4.17: Approximate expected values while training a categorical distribution with the same set-up as in figure 4.15. For this training, we initialize the probabilities randomly.



To further explore the behavior of the measure-valued estimator on discrete distributions, we look at the Poisson distribution. Since this distribution has infinite support with only one parameter dimension, the gradient estimation differs with respect to the other distributions we have evaluated. For the measure-valued gradient estimate of a Poisson distribution $\mathcal{P}(\lambda)$, we use the triplet

$$c_\lambda = \frac{1}{\lambda}, \quad p^+ = \mathcal{P}(\lambda) + 1, \quad p^- = \mathcal{P}(\lambda).$$

The sampling strategies for the measure-valued gradient estimates are also listed in table 2.2. Since the Poisson distribution expects the rate λ to be positive, we train the model to output $\ln \lambda$.

In figure 4.18, we show the results when using measure-valued and score-function estimates to train the expected value of a Poisson distribution on a simple, shifted parabola.

With the measure-valued estimator, we reliably get to the optimum independent of the starting rate. This is to be expected, because the difference $f(x+1) - f(x)$ for a sample $x \sim \mathcal{P}(x; \lambda)$ is a finite-difference gradient at x . In this case, the simple convex loss function is easily optimized by that gradient. Figure 4.20 shows, how the stronger skew for small initial rates requires more iterations to get out of. Generally, we should initialize the Poisson distribution with a higher rate to encourage exploration.

When comparing the measure-valued estimator to the score-function estimator in this scenario, we find the score-function estimator to be very unstable. The problem here is that the score-function estimator tends to overshoot for certain initializing, and struggles to get out of small rates. Figure 4.19 demonstrates this problem when trying to use a higher learning rate. This happens, because the score-function estimates rely only on samples $x \sim \mathcal{P}(x; \lambda)$ of the current distribution. Once the distribution becomes very biased, i.e., in this case $\mathcal{P}(0; \lambda) \approx 1$, the estimator is simply stuck. While a different choice of hyperparameters can help with this issue, we consider this a serious downside as compared to the measure-valued estimator. We discuss the implications of these results further in section 5.3.

Figure 4.18: Approximate expected values while training a Poisson distribution with a shifted parabola loss. The measure-valued estimator using 1 sample, i.e., 2 loss evaluations, while the score-function estimator uses 2 samples for the same number of loss evaluations. Learning rate is $\alpha = 0.01$, training $\ln \lambda$.

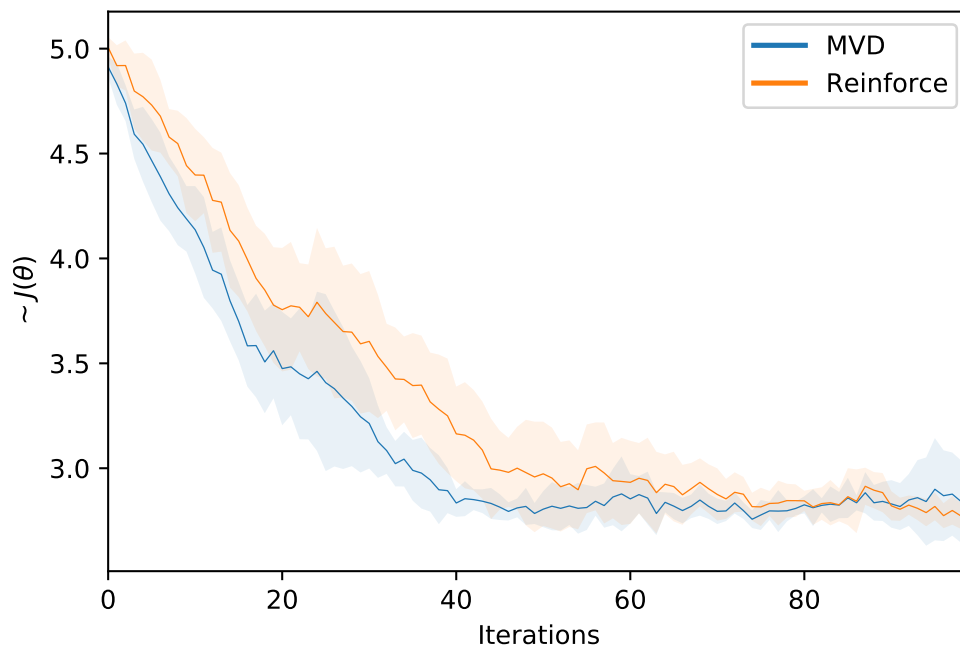


Figure 4.19: Probability mass functions while training a Poisson distribution with the *score-function estimator* using 2 samples. We only vary the initial rates, learning rate is $\alpha = 0.1$, training $\ln \lambda$. The learning rate is too high in this case, hence the estimator overshoots and gets stuck in an extremely skewed distribution.

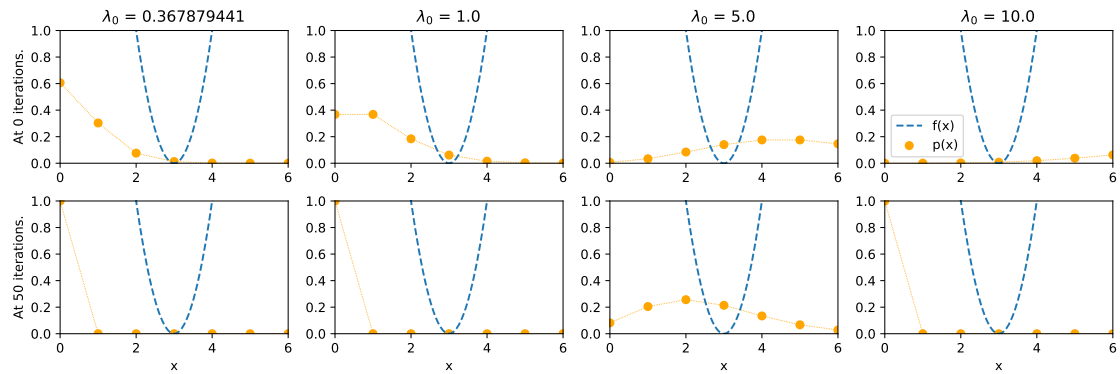
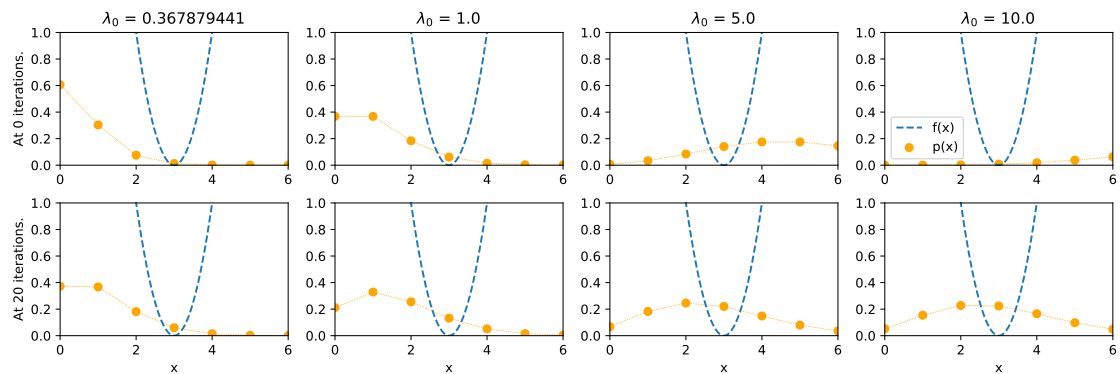


Figure 4.20: Probability mass functions while training a Poisson distribution with the *measure-valued estimator* using 1 sample, i.e., 2 loss evaluations. We only vary the initial rates, learning rate is $\alpha = 0.1$, training $\ln \lambda$. Unlike the score-function estimator in figure 4.19, the measure-valued estimator reliably finds a solution, independent of starting parameters.



5 Discussion

In this chapter, we discuss the results presented in chapter 4. The sections highlight some of the most important characterizing differences we find, and discuss their implications. Also, we work out some aspects which would require more research to get a better understanding.

5.1 The Expected Advantage of the Pathwise Estimator

We conclude that the pathwise estimator is almost always the preferred choice if applicable.

Throughout chapter 4, we have seen the pathwise estimator outperforming the other two estimators. This is generally not surprising, as we discussed in section 2.1. Since the pathwise estimator uses all the gradient information of the whole computation, it produces very low variance estimates. However, we have also shown some downsides to this approach. In some tasks, differentiability of all intermediate computations introduces some other problems.

For discrete distributions, we have seen that the Gumbel-softmax trick allows us to estimate the gradient, but the bias becomes problematic in multimodal situations. We have seen this problem in section 4.4, and figure 4.16, where the other estimators show much better results.

In addition to the downsides of the estimator, we also find some other interesting characteristics. In figure 4.5, we have seen that sometimes the pathwise estimator performs better with less samples. While further investigation of this is outside the scope of the thesis, we propose two lines of reasoning for this. For one, it is likely that the additional samples do not contribute meaningfully to the gradient estimate. Hence, any additional computation on these samples is wasted, as the gradient direction stays the same. Second, and opposite to the first reasoning, it is possible that a smaller number of samples increases

the variance enough to allow the gradient estimate to break out of poor local minima. In our case, we find the first reasoning more likely, since the higher variances estimators still achieve less performance in the same time. These findings are also supported by the considerations of Mohamed et al. [46].

5.2 Issues of the Score-Function Estimator

In this section, we discuss reasons why the score-function estimator may often not be the best choice for the tasks we evaluated.

Especially in more complex tasks, such as the variational auto-encoder, we find that the score-function estimator has very high variance. However, since this was not the focus of this thesis, we did not optimize the score-function estimator as much. Still, we find that its performance in the best cases only just about matches the performance of the pathwise estimator. Though this is not a surprise, as stated in section 5.1, we can only confirm this assumption. Hence, the score-function estimator remains a viable choice only when there are non-differentiable components in either the loss function, or the distribution itself.

When trying to combine the score-function estimator and the measure-valued estimator in a convex combination, we find there is some potential for optimization. We were not able to improve upon the score-function estimator’s performance. Still, we cannot rule out the potential of the approach. Most surprisingly, we find that the combination estimator’s performance was best when we used fewer measure-valued dimensions. This could mean that replacing a small number of dimensions might be enough to benefit from the low variance of the measure-valued estimate. The results indicate that a randomized combination is probably not informed enough, and improving the replacement strategy is required.

We conclude that comparing the score-function estimator to the measure-valued estimator, we found that for complex tasks, the measure-valued estimator tends to have lower variance for the same number of function evaluations. It is hard to say which have the better performance, but our results suggest that the measure-valued estimator has an advantage here. In our process time comparisons, the score-function estimator suffers a lot more from a higher number of samples. We acknowledge, however, that this might be due to the lacking optimization of the score-function estimator. For our score-function estimator implementation, we rely on the sampling and log probability implementations of PyTorch, while the measure-valued derivative uses our own implementation for sampling

and gradient calculations. Of note, however, is that the score-function estimator requires more backpropagation for the gradient of the log probabilities, while the measure-valued derivative computes the gradient of the parameters directly.

5.3 Potential of the Measure-Valued Estimator

In this section, we show the potential of the measure-valued estimator in situations, where the pathwise estimator is not applicable.

Even though we find that the pathwise estimator clearly outperforms the other estimators in most applications, we identify certain tasks in which the pathwise estimator struggles. We suggest that the measure-valued estimator is a strong choice in these scenarios, as it often produces better results than the score-function estimator. Our results of chapter 4 suggest that the measure-valued estimator is a low variance estimator requiring less care in the selection of hyperparameters. It performs well throughout all experiments, with different datasets, models, optimizers, and latent dimension sizes. Its major downside, however, is the scaling of the required function evaluations. The dimension size factor limits the choice of number of samples taken per estimate. In comparison, we are free to choose any number of samples for the other estimators. In a scenario with a large number of latent dimensions, the sample size can become very important, making these situations a big problem for the measure-valued estimator.

The optimal gradient estimator differs often between scenarios, but we cannot recommend the measure-valued estimator as a default choice. When applicable, we find that the pathwise estimator often outperforms it, while providing some more flexibility in the choice of hyperparameters. However, whenever a problem requires additional steps to make the pathwise estimator work, e.g. using discrete distributions, it is worth it to consider the measure-valued estimator as a low variance alternative.

We find that the optimal estimator choice often depends on the specific use case, and the estimators differ enough to be able to differentiate them for most specific requirements. However, in most scenarios we have explored, we can recommend the measure-valued estimator over the score-function estimator. Our experiments have shown that the gradient calculations are quicker with the measure-valued estimator, given the same number of function evaluations. Also, the measure-valued estimator seems less restrictive on the hyperparameters, and reliably finds local solutions in all experiments. Hence, we would recommend the score-function estimator over the measure-valued estimator only

in situations, where a lower number of samples with the score-function estimator is advantageous.

For the rest of this section, we discuss our results with regards to the specific scenarios and ideas we have explored in chapter 4.

Convex Combinations Concerning the approach of convex combinations, our results indicate that simply replacing random dimensions does not yield better results than just using one of the estimators. Still, there is a lot of space for further research here, as we could imagine a lot better performance with a better combination strategy. While the results of convex combinations were worse for us, we have to recognize that we chose the most simple implementation for the approach. We suspect a well informed strategy for choosing the measure-valued estimate dimensions could improve this approach a lot. To find such a strategy we recommend investigating ways to find the dimensions which influence the score-function estimate’s variance the most, and replacing those dimensions.

Discrete Mixture Models. Another approach we evaluate is the use of the measure-valued estimator for discrete mixture models. The most common form of discrete mixture models are Gaussian mixture models. In section 4.2, we conclude that the pathwise estimator is probably the best choice for normal distributions. Hence, we focus our evaluations on the selectors, which are categorical distributions in the case of discrete Gaussian mixtures. We find that the measure-valued estimator enumerates the support of the distribution in this case. This makes the gradient inherently not stochastic. Unsurprisingly, we find that the resulting gradient works very well for various loss shapes. However, it comes with the major downside of being completely rigid in terms of loss function evaluations. It requires exactly k evaluations, where k is the number of categories. We find that the other estimators in this case allow for some more flexibility, but yield much less reliable results. Interestingly, the score-function estimator proved as the second best choice in this scenario, as it still yields an unbiased, but stochastic result. The pathwise estimator using the Gumbel-softmax trick we have to apply for categorical distributions yields less good results. This seems especially problematic for multimodal losses, which we would expect in most real-world applications.

Concluding, for the gradient of the discrete, finite selector of the mixture, we recommend the measure-valued estimator, or the score-function estimator, if less loss evaluations are desirable. For the gradient of the Gaussian components, we recommend the pathwise estimator, as they are easily reparameterizable.

Poisson Distributions. Since the measure-valued derivative relies on enumerations for categorical distributions, we also evaluate Poisson distributions as discrete distributions with infinite support. We find that the measure-valued estimator is much more stable than the score-function estimator.

Our results demonstrate some areas where the *score-function estimator* struggles in figure 4.19. On one side, if the rate λ of the distribution is very low, the distribution becomes skewed towards zero, i.e., $\mathcal{P}(0; \lambda) \approx 1$ for $\lambda \rightarrow 0$. For the score-function estimator, since it samples $x \sim \mathcal{P}(x)$ directly, we end up only sampling one value. To alleviate this problem we have two options. Simply reducing the learning rate is sometimes enough to avoid overshooting into this problem. While this approach is easy, it does not guarantee that this doesn't happen. The other option is trying to choose a baseline with a smaller decay rate. This only works if the initial rate was not in the critical range close to zero, because the baseline has to incorporate the loss difference to other areas. As we can see, both options only reduce one part of the problem. On the other side, if the rate λ is very large, the probabilities become very small. Since the score-function estimator relies on the gradient of the log probabilities, it produces large gradients at this point. If the learning rate is not chosen carefully, or if λ is sufficiently large, it often reduces λ to a very small number. At this point it is again running into the $\lambda \rightarrow 0$ problem where the distribution is skewed towards one value. Hence, the choice of initial λ_0 is essential to the performance of the score-function estimator on Poisson distributions.

The *measure-valued estimator* has shown stable results in our experiments, mostly independent of λ_0 . A very small λ_0 requires a few more iterations, but nonetheless the learning is stable in this case as well. We find that the measure-valued estimator seems the preferred choice of estimator in this case, because it is unaffected by the skew of the distribution. Also, the estimator is not as restrictive for Poisson distributions, as they have only one parameter, so they require at most only two loss evaluations for an estimate. Hence, for Poisson distributions, we recommend the measure-valued estimator as the most stable choice.


6 Conclusion

In this thesis, we implement a framework for gradient estimators for stochastic nodes in computational graphs, and use our framework to explore possible use-cases for the measure-valued gradient estimator. We compare our results to current research in the area, and find that our framework reveals some more details on the distinct characteristics of the measure-valued estimator. For some cases, we show a clear advantage over the score-function estimator. However, for most cases, we show that the pathwise estimator is still preferable. We find that the variances of the measure-valued and pathwise estimators are in fact very close for standard VAE architectures, using fully connected, convolutional, and recurrent layers.

Discrete distributions are one area where we found the measure-valued derivative to perform the most stable, and yield the best results of all three estimators. However, it is not stochastic in this case, as it enumerates the domain of the distribution. The pathwise estimator seems to suffer a lot from the Gumbel-softmax trick. Even though the score-function estimator seems preferable to the pathwise estimator in these cases, we still find that the higher flexibility of sample size compared to the measure-valued derivative might often not be worth it.

Considering the convex combination of the score-function estimator with partial measure-valued estimates, we find some interesting open research questions. While we were not able to improve the performance of the vanilla estimators by this combination, we could show that a convex combination still yields stable results, and remains a valid choice for an estimator. Since we only explored a randomized combination, we suggest that a more informed version may be able to improve upon these results, and could be a valuable alternative to the vanilla estimators.

Generally, the measure-valued estimator has shown stable results, performing well in all cases. Similarly to the pathwise estimator, we find that it does not rely as much on hyperparameter tuning as the score-function estimator. In conclusion, while the measure-valued gradient estimator is probably not the best choice if the pathwise estimator is



applicable, it shows potential in many scenarios, and seems a reliable, stable alternative, for example, when using non-differentiable loss functions.

Bibliography

- [1] Atilim Gunes Baydin, Barak A. Pearlmutter, and Alexey Andreyevich Radul. “Automatic differentiation in machine learning: a survey”. In: *CoRR* abs/1502.05767 (2015). arXiv: 1502.05767. URL: <http://arxiv.org/abs/1502.05767>.
- [2] Yoshua Bengio, Eric Thibodeau-Laufer, and Jason Yosinski. “Deep Generative Stochastic Networks Trainable by Backprop”. In: *CoRR* abs/1306.1091 (2013). arXiv: 1306.1091. URL: <http://arxiv.org/abs/1306.1091>.
- [3] P.J. Bickel and K.A. Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics*. Mathematical Statistics: Basic Ideas and Selected Topics Vol. 1. Prentice Hall, 2001. ISBN: 9780138503635. URL: <https://books.google.de/books?id=8poZAQAIAAJ>.
- [4] P. Billingsley. *Probability and Measure*. Wiley Series in Probability and Statistics. Wiley, 2012. ISBN: 9781118341919. URL: <https://books.google.de/books?id=a3gavZbxyJcC>.
- [5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.
- [6] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. “Variational Inference: A Review for Statisticians”. In: *Journal of the American Statistical Association* 112.518 (Apr. 2017), pp. 859–877. ISSN: 1537-274X. DOI: 10.1080/01621459.2017.1285773. URL: <http://dx.doi.org/10.1080/01621459.2017.1285773>.
- [7] Lars Buesing, T. Weber, and S. Mohamed. “Stochastic Gradient Estimation With Finite Differences”. In: 2016.
- [8] Luca Capriotti. *Reducing the Variance of Likelihood Ratio Greeks with Monte Carlo*. 2008. arXiv: 0808.2332 [physics.data-an].
- [9] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: <http://arxiv.org/abs/1406.1078>.

-
-
- [10] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: <http://archive.ics.uci.edu/ml>.
- [11] Otto Fabius and Joost R. van Amersfoort. *Variational Recurrent Auto-Encoders*. 2015. arXiv: 1412.6581 [stat.ML].
- [12] Michael Figurnov, Shakir Mohamed, and Andriy Mnih. “Implicit Reparameterization Gradients”. In: *CoRR* abs/1805.08498 (2018). arXiv: 1805.08498. URL: <http://arxiv.org/abs/1805.08498>.
- [13] Harley Flanders. “Differentiation Under the Integral Sign”. In: *The American Mathematical Monthly* 80.6 (1973), pp. 615–627. ISSN: 00029890, 19300972. URL: <http://www.jstor.org/stable/2319163>.
- [14] Paul Glasserman. “Gradient Estimation Via Perturbation Analysis”. In: (1991).
- [15] P. Glynn. “Likelihood ratio gradient estimation: an overview”. In: *WSC '87*. 1987.
- [16] Peter W. Glynn. “Likelihood Ratio Gradient Estimation for Stochastic Systems”. In: *Commun. ACM* 33.10 (Oct. 1990), pp. 75–84. ISSN: 0001-0782. DOI: 10.1145/84537.84552. URL: <https://doi.org/10.1145/84537.84552>.
- [17] Peter W. Glynn and Donald L. Iglehart. “Importance Sampling for Stochastic Simulations”. In: *Management Science* 35.11 (1989), pp. 1367–1392. ISSN: 00251909, 15265501. URL: <http://www.jstor.org/stable/2632283>.
- [18] Peter W. Glynn and Pierre L’Ecuyer. “Likelihood Ratio Gradient Estimation for Stochastic Recursions”. In: *Advances in Applied Probability* 27.4 (1995), pp. 1019–1053. ISSN: 00018678. URL: <http://www.jstor.org/stable/1427933>.
- [19] Peter W. Glynn and Roberto Szechtman. “Some New Perspectives on the Method of Control Variates”. In: *Monte Carlo and Quasi-Monte Carlo Methods 2000*. Springer-Verlag, 2000, pp. 27–49.
- [20] Ary Goldberger et al. “PhysioBank, PhysioToolkit, and PhysioNet : Components of a New Research Resource for Complex Physiologic Signals”. In: *Circulation* 101 (July 2000), E215–20. DOI: 10.1161/01.CIR.101.23.e215.
- [21] W.B. Gong and Y.-C. Ho. “Smoothed (conditional) perturbation analysis of discrete event dynamical systems”. In: *IEEE Transactions on Automatic Control* 32 (Oct. 1987), pp. 858–866.
- [22] Ian J. Goodfellow. “NIPS 2016 Tutorial: Generative Adversarial Networks”. In: *CoRR* abs/1701.00160 (2017). arXiv: 1701.00160. URL: <http://arxiv.org/abs/1701.00160>.

-
-
- [23] Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. “Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning”. In: *J. Mach. Learn. Res.* 5 (Dec. 2004), pp. 1471–1530. ISSN: 1532-4435.
- [24] G. Grimmett, G.R. Grimmett, and D. Stirzaker. *Probability and Random Processes*. Probability and Random Processes. OUP Oxford, 2001. ISBN: 9780198572220. URL: <https://books.google.de/books?id=G3ig-0M4wSIC>.
- [25] G. Grimmett and D. Stirzaker. “Probability and random processes”. In: 1982.
- [26] Shixiang Gu et al. *MuProp: Unbiased Backpropagation for Stochastic Neural Networks*. 2016. arXiv: 1511.05176 [cs.LG].
- [27] B. Heidergott, F. Vázquez-Abad, and Warren Volk-Makarewicz. “Sensitivity estimation for Gaussian systems”. In: *Eur. J. Oper. Res.* 187 (2008), pp. 193–207.
- [28] Bernd Heidergott, Felisa Vázquez-Abad, and Member Gerad. “Measure valued differentiation for stochastic processes: The finite horizon case”. In: Jan. 2000.
- [29] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [30] Matthew D. Hoffman and David M. Blei. *Structured Stochastic Variational Inference*. 2014. arXiv: 1404.4114 [cs.LG].
- [31] M. Iri and K. Tanabe. *Mathematical Programming: Recent Developments and Applications*. Mathematics and its Applications. Springer Netherlands, 1989. ISBN: 9780792304906. URL: <https://books.google.de/books?id=bkjavAAAAMAAJ>.
- [32] Tommi Jaakkola and Michael Jordan. “A variational approach to Bayesian logistic regression models and their extensions”. In: (Aug. 2001).
- [33] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax*. 2017. arXiv: 1611.01144 [stat.ML].
- [34] Martin Jankowiak and Theofanis Karaletsos. “Pathwise Derivatives for Multivariate Distributions”. In: *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, Apr. 2019, pp. 333–342. URL: <https://proceedings.mlr.press/v89/jankowiak19a.html>.
- [35] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

-
-
- [36] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).
- [37] Diederik P. Kingma et al. “Semi-supervised Learning with Deep Generative Models”. In: *NIPS*. 2014.
- [38] Alp Kucukelbir et al. *Automatic Differentiation Variational Inference*. 2016. arXiv: 1603.00788 [stat.ML].
- [39] H. Kushner and G.G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Stochastic Modelling and Applied Probability. Springer New York, 2003. ISBN: 9780387008943. URL: <https://books.google.de/books?id=EC2w1SaPb7YC>.
- [40] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. “Human-level concept learning through probabilistic program induction”. In: *Science* 350.6266 (2015), pp. 1332–1338. ISSN: 0036-8075. DOI: 10.1126/science.aab3050. eprint: <https://science.sciencemag.org/content/350/6266/1332.full.pdf>. URL: <https://science.sciencemag.org/content/350/6266/1332>.
- [41] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: <http://yann.lecun.com/exdb/mnist/>.
- [42] Wonyeol Lee, Hangyeol Yu, and Hongseok Yang. “Reparameterization Gradient for Non-differentiable Models”. In: *CoRR* abs/1806.00176 (2018). arXiv: 1806.00176. URL: <http://arxiv.org/abs/1806.00176>.
- [43] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/>.
- [44] J. Carvalho et al. “An Empirical Analysis of Measure-Valued Derivatives for Policy Gradients”. In: *International Joint Conference on Neural Networks (IJCNN)*. 2021. URL: https://www.ias.informatik.tu-darmstadt.de/uploads/Team/JoaoCarvalho/2021_ijcnn-mvd_rl.pdf.
- [45] Nicholas Metropolis and S. Ulam. “The Monte Carlo Method”. In: *Journal of the American Statistical Association* 44.247 (1949). PMID: 18139350, pp. 335–341. DOI: 10.1080/01621459.1949.10483310. eprint: <https://www.tandfonline.com/doi/pdf/10.1080/01621459.1949.10483310>. URL: <https://www.tandfonline.com/doi/abs/10.1080/01621459.1949.10483310>.

-
-
- [46] S. Mohamed et al. “Monte Carlo Gradient Estimation in Machine Learning”. In: *J. Mach. Learn. Res.* 21 (2020), 132:1–132:62.
- [47] Kevin P Murphy. *Machine learning: a probabilistic perspective*. Cambridge, MA, 2012.
- [48] John Paisley, David Blei, and Michael Jordan. *Variational Bayesian Inference with Stochastic Search*. 2012. arXiv: 1206.6430 [cs.LG].
- [49] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].
- [50] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.
- [51] G. Ch. Pflug. “Sampling Derivatives of Probabilities”. In: *Computing* 42.4 (Oct. 1989), pp. 315–328. ISSN: 0010-485X. DOI: 10.1007/BF02243227. URL: <https://doi.org/10.1007/BF02243227>.
- [52] Mihaela Rosca and Michael Figurnov. *Measure-Valued Derivatives for Approximate Bayesian Inference*. 2019.
- [53] John Schulman et al. “Gradient Estimation Using Stochastic Computation Graphs”. In: *NIPS*. 2015.
- [54] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. “Learning Structured Output Representation using Deep Conditional Generative Models”. In: *NIPS*. 2015, pp. 3483–3491. URL: <http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models>.
- [55] A.K Subramanian. *PyTorch-VAE*. <https://github.com/AntixK/PyTorch-VAE>. 2020.
- [56] A. Takeshi, T.A. AMEMIYA, and Harvard University Press. *Advanced Econometrics*. Harvard University Press, 1985. ISBN: 9780674005600. URL: <https://books.google.de/books?id=0bzGQE14CwEC>.
- [57] Ilya Tolstikhin et al. *Wasserstein Auto-Encoders*. 2019. arXiv: 1711.01558 [stat.ML].
- [58] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Machine Learning* 8 (1992), pp. 229–256. DOI: 10.1023/A:1022672621406.
- [59] Mark Zlochins and Yoram Baram. “The Bias-Variance Dilemma of the Monte Carlo Method”. In: July 2000. ISBN: 978-3-540-42486-4. DOI: 10.1007/3-540-44668-0_20.