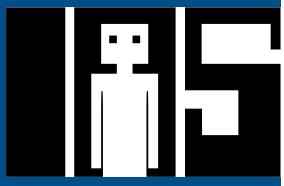
# Amortized Variational Inference with Gaussian Mixture Models

**Amortisierte Variationsinferenz mit Gaußschen Mischmodellen**
Master thesis by Daniel Musekamp
Date of submission: October 10, 2022

1. Review: Dr. Ing. Oleg Arenz
2. Review: Prof. Jan Peters, Ph.D.
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT



## Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Daniel Musekamp, die vorliegende Masterarbeit 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 Fall 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 der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

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

Darmstadt, 10. Oktober 2022

D. Musekamp

# Abstract

Amortized variational inference is essential to many areas of machine learning. However, most approaches still rely on simple distributions, leading to a potentially large approximation error. In this work, we investigate to use Gaussian Mixture Models, which are a promising alternative as they provide a flexible and powerful representation. Building on a recently proposed lower bound to decompose the mixture objective, we adopt trust region optimization in order to encourage exploration and promote mixture diversity. We describe a numerical instability in the objective function and prevent it using a covariance regularization technique based on eigenvalue bounds. Sample efficiency is improved by applying resampling to evaluate a mixture component only on relevant inputs. Furthermore, we investigate strategies to adapt the number of mixture components and discuss the feasibility of sharing parameters between components.
We evaluate the proposed algorithm on a set of inverse kinematic tasks and show results competitive to other generative modeling techniques in terms of task space accuracy. Compared to standard amortized inference using Gaussian Mixture Models, the algorithm can generate more accurate and diverse samples.

# Zusammenfassung

Amortisierte Variationsinferenz spielt eine bedeutende Rolle in vielen Bereichen des maschinellen Lernens. Die meisten Ansätze verwenden jedoch einfache Verteilungsfunktionen, was zu einem potenziell großen Näherungsfehler führen kann. Gaußsche Mischmodelle stellen eine vielversprechende Alternative dar, da sie eine flexible und mächtige Repräsentation ermöglichen. In dieser Arbeit werden, ausgehend von einer kürzlich vorgeschlagenen unteren Grenze zur Zerlegung der Zielfunktion, Vertrauensbereiche verwendet, um die Erkundung zu fördern und Modelle mit mehr aktiven Komponenten zu lernen. Aufgrund einer gefundenen numerischen Instabilität in der Zielfunktion werden Eigenwertsgrenzen verwendet um die Kovarianzmatrix zu regularisieren. Zudem wird die Optimierung effizienter gestaltet indem Modellkomponenten nur auf relavanten Eingaben optimiert werden. Des Weiteren werden Strategien zur Anpassung der Anzahl der Modellkomponenten und der Nutzen des Teilens von Parametern zwischen den Komponenten untersucht.
Experimente mit einer Reihe von Inverskinematik-Problemen zeigen, dass der vorgeschlagene Algorithmus ähnlich geringe Fehler wie andere generative Modellierungstechniken aufweist. Im Vergleich mit herkömmlicher amortisierter Variationsinferenz mit Gaußschen Mischmodellen werden eine höhere Genauigkeit und vielfältigere Verteilungen erreicht.

# Contents

# 1. Introduction

Variational Inference (VI) [1, 2] is a ubiquitous tool in machine learning which allows to sample from intractable, unnormalized distributions by rephrasing inference as an optimization problem: by minimizing the reverse Kullback-Leibler divergence (KL) between a tractable distribution $q$ and the target distribution $p$, we can use $q$ to generate samples from $p$. However, standard VI is only suitable to approximate unconditional distributions. In case of a conditional distribution, VI needs to be performed for every input, which may be too slow for many application cases. Thus in amortized VI, a model learns to predict the solution of VI for each input, replacing the costly optimization procedure with a simple inference step. Amortized VI has provided the basis for well-known algorithms such as Variational Auto-Encoders (VAE) [3] or Soft Actor-Critic [4].

A common problem of VI is that often only simple distributions are used to approximate the target distribution, typically normal distributions with a diagonal covariance matrix, which can lead to potentially large approximation errors. One approach to go beyond such simple approximations is the Variational Inference by Policy Search (VIPS) [5] algorithm. VIPS use Gaussian Mixture Models (GMM), a powerful, yet interpretable representation. GMMs are also employed by VIPS to enhance exploration by dynamically adding new mixture components to promising regions. Furthermore, VIPS improves the optimization procedure by using trust regions [6, 7, 8]. Since VI is like reinforcement learning (RL) an exploration problem, trust region optimization is useful to encourage exploration by limiting the update step size in distribution space [5].

In this work, we present an amortized version of VIPS called aVIPS. We train neural networks to predict GMMs using the lower bound presented in VIPS. To perform trust region constrained updates, we build on the technique used by MPO [9] in the context of reinforcement learning and adapt it to VI. Since dynamic adaptation of the number of mixture components is an important part of the exploration process in VIPS, we also investigate component adding strategies for amortized VI.

Next to the core parts of VIPS, some additional challenges specific to amortized VI are addressed. To speed up training, we propose to train components only on relevant inputs by using importance resampling. While we mainly focus on a mixture-of-experts model, we also present a version of aVIPS using a shared encoder network. Finally, we describe a numerical instability in the GMM-VI objective which is mitigated using covariance regularization.

To evaluate our approach, we consider Inverse Kinematic (IK) tasks. IK describes the problem of finding joint configurations corresponding to a desired robot endeffector pose in task space, and is not analytically solvable for many robots. Furthermore, kinematic chains with a redundant number of degrees of freedom allow to reach a target with many joint configurations, leading to a set of valid solutions. Traditionally, IK was approached using optimization procedures such as Damped Least Squares [10] or sequential quadratic programming [11]. Recently, multiple authors have proposed to view IK as a distribution learning problem and have employed deep generative modeling techniques like Generative Adversarial Nets (GAN) [12] or normalizing flows [13] to predict a set of solutions. Approaching IK using VI [14] is promising, since the mode-seeking behavior of the reverse KL will force the model to concentrate on good solutions instead of trying to capture all modes and accepting to cover low reward regions in between.
Experimental results show that aVIPS generates samples with a low task space error. Compared to a naive training of conditional GMMs, aVIPS generates mixtures with more active components and higher entropy. However, we have also demonstrated that a core component of VIPS, i.e. dynamically adding new components to the mixture, is difficult to realize in the amortized setting and did not show an advantage.

The presented work is organized as follows. First, we give an overview over the problem setting and VIPS in Chap. 2. We discuss aVIPS in Chap. 3 and review related work in Chap. 4. Next, the experimental results are discussed (Chap. 5). Finally, we present a summary and outline possible future work in Chap. 6.

# 2. Preliminaries

In this chapter, we discuss the foundations of our presented approach, including the problem setting of classical and amortized VI as well as an overview of VIPS.

## 2.1. Variational Inference

In its most general form, VI is a technique to conduct statistical inference from an intractable probabilistic model $p(\mathbf{x})$ by approximation with a tractable distribution $q(\mathbf{x})$. A common use case is Bayesian inference in a hidden variable model. Here, likelihood $p(\mathbf{y}|\mathbf{x})$ and prior $p(\mathbf{x})$ are known and the goal is to calculate the posterior distribution of the hidden state $\mathbf{x}$ given an observation $\mathbf{y}$:

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}.$$

However, since the probability of the evidence $p(\mathbf{y})$ is unknown, it has to be calculated first by marginalizing $\mathbf{x}$, $p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})d\,\mathbf{x}$. This integral can be prohibitively expensive to solve, especially if $\mathbf{x}$ is a high-dimensional vector or if the likelihood $p(\mathbf{y} \mid \mathbf{x})$ is defined by a deep neural network. To approximate $p(\mathbf{x}|\mathbf{y})$ for a specific observation $\mathbf{y}$ or any other probability density function $p(\mathbf{x})$, VI minimizes the KL divergence:

$$\min_{q(\mathbf{x})} \mathrm{KL}\left(q(\mathbf{x})||p(\mathbf{x})\right).$$

However, the distribution $p(\mathbf{x})$ is often only known up to a constant $Z$, i.e. $p(\mathbf{x}) = \tilde{p}(\mathbf{x})/Z$. In Bayesian inference, $Z$ corresponds to the unknown and often intractable probability of the evidence $p(\mathbf{y})$, and $\tilde{p}(\mathbf{x})$ to the product of likelihood and prior. Since $p(\mathbf{x})$ and

subsequently the KL divergence can not be calculated, the VI objective is reformulated using

$$\begin{aligned}\mathrm{KL}\left(q(\mathbf{x})||p(\mathbf{x})\right) &= \int_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} d\,\mathbf{x} \\ &= \int_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})Z}{\tilde{p}(\mathbf{x})} d\,\mathbf{x} \\ &= \underbrace{\int_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{\tilde{p}(\mathbf{x})} d\,\mathbf{x}}_{:=-L} + \log Z,\end{aligned}$$

where $L$ is known as the Evidence Lower Bound (ELBO) and does not include the unknown normalization constant $Z$ anymore. By reformulating VI as maximization of the ELBO, the model $q(\mathbf{x})$ can hence be fitted without the normalization constant.

## 2.2. Variational Inference by Policy Search

ELBO maximization can be interpreted as an RL problem [5] with an additional entropy term $\mathcal{H}\left(q\left(\mathbf{x}\right)\right) = -\int_{\mathbf{x}} q(\mathbf{x}) \log q(\mathbf{x}) d\,\mathbf{x}$ :

$$\begin{aligned}L &= -\int_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{\tilde{p}(\mathbf{x})} d\,\mathbf{x} \\ &= \int_{\mathbf{x}} q(\mathbf{x}) \log \tilde{p}(\mathbf{x}) d\,\mathbf{x} - \int_{\mathbf{x}} q(\mathbf{x}) \log q(\mathbf{x}) d\,\mathbf{x} \\ &= \mathop{\mathbb{E}}_{q(\mathbf{x})} R(\mathbf{x}) + \mathcal{H}\left(q\left(\mathbf{x}\right)\right).\end{aligned}$$

In this view, $q(\mathbf{x})$ corresponds to a policy acting in a Markov decision process with a single state and a Q-function $R(\mathbf{x})$. Since the goal is to find the policy with the maximum expected reward, this optimization problem is also known as policy search. The interpretation of VI as policy search highlights that VI is affected by one of RL's main challenges, known as the exploration-exploitation trade-off: if the policy $q(\mathbf{x})$ is updated greedily towards the samples with the highest rewards (exploitation), the improvement of expected reward in the current iteration may be maximal. However, since now more of the policy's probability mass is focused on the seemingly good regions, the samples of the next iteration are less likely to be put into yet uncovered regions (exploration). Thus, there is a conflict

between short-term greedy improvement of the policy and keeping the policy's entropy high in order to discover possibly unseen high-reward regions. The practical benefit of the connection between VI and RL is that it allows to employ policy search algorithms, which can deal with the exploration-exploitation trade-off, in VI.

Building on the interpretation of VI as RL, Arenz et al. [5] present VIPS, which uses a GMM with full covariance matrices to approximate the target function, $q(\mathbf{x}) = \sum_o q(o)q(\mathbf{x}\,|o)$, where $q(o)$ represents the categorical distribution over the mixture weights and $q(\mathbf{x}\,|o)$ Gaussian component $o$ of the mixture. The ELBO of the GMM is given as

$$L = \sum_o q(o) \int_{\mathbf{x}} q(\mathbf{x}\,|o)(R(\mathbf{x}) - \log q(\mathbf{x}))d\,\mathbf{x}\,. \tag{2.1}$$

A key feature of VIPS is that it can divide the overall optimization of the GMM into independent sub-problems for each component $q(\mathbf{x}\,|o)$. Therefore, the term $\log q(\mathbf{x})$, which consists of all components, has to be replaced. Arenz et al. [5] first insert the log-responsibilities $\log q(o|\,\mathbf{x}) = \log q(o) + \log q(\mathbf{x}\,|o) - \log q(\mathbf{x})$ into Eq. 2.1:

$$\begin{aligned} L &= \sum_o q(o) \int_{\mathbf{x}} q(\mathbf{x}\,|o)(R(\mathbf{x}) + \log q(o|\,\mathbf{x}) - \log q(o) - \log q(\mathbf{x}\,|o))d\,\mathbf{x} \\ &= \sum_o q(o) \left[\int_{\mathbf{x}} q(\mathbf{x}\,|o)(R(\mathbf{x}) + \log q(o|\,\mathbf{x}))d\,\mathbf{x} + \mathcal{H}(q(\mathbf{x}\,|o))\right] + \mathcal{H}(q(o)). \end{aligned}$$

Secondly, an auxiliary distribution $\tilde{q}(o|\,\mathbf{x})$ is integrated:

$$\begin{aligned} L =& \sum_o q(o) \left[\int_{\mathbf{x}} q(\mathbf{x}\,|o)(R(\mathbf{x}) + \log q(o|\,\mathbf{x}) + \log \tilde{q}(o|\,\mathbf{x}) - \log \tilde{q}(o|\,\mathbf{x}))d\,\mathbf{x} + \mathcal{H}(q(\mathbf{x}\,|o))\right] \\ &+ \mathcal{H}(q(o)) \\ =& \sum_o q(o) \left[\int_{\mathbf{x}} q(\mathbf{x}\,|o)(R(\mathbf{x}) + \log \tilde{q}(o|\,\mathbf{x}))d\,\mathbf{x} + \mathcal{H}(q(\mathbf{x}\,|o))\right] + \mathcal{H}(q(o)) \\ &+ \int_{\mathbf{x}} q(\mathbf{x})\mathrm{KL}(q(o|\,\mathbf{x})||\tilde{q}(o|\,\mathbf{x}))d\,\mathbf{x} \\ \geq& \sum_o q(o) \left[\int_{\mathbf{x}} q(\mathbf{x}\,|o)(R(\mathbf{x}) + \log \tilde{q}(o|\,\mathbf{x}))d\,\mathbf{x} + \mathcal{H}(q(\mathbf{x}\,|o))\right] + \mathcal{H}\left(q\left(o\right)\right) := \tilde{L}. \end{aligned} \tag{2.2}$$

In the last step, a new lower bound is derived using that the KL-divergence is always positive. In this new lower bound, the components $q(\mathbf{x}\,|o)$ can now be updated independently of each other. The optimization is conducted in an expectation-maximization fashion. In the

maximization step, the parameters of $q(\mathbf{x})$ are optimized regrading the lower bound $\tilde{L}$. In the expectation step, the auxiliary distribution $\tilde{q}(o|\,\mathbf{x})$ is set to $q(o|\,\mathbf{x}) = q(\mathbf{x}\,|o)q(o)/q(\mathbf{x})$, so that the KL-divergence in Eq. 2.2 becomes zero. Thus, the lower bound is tightened and $L$ and $\tilde{L}$ are equal again.
The maximization step of the component update is defined as:

$$\begin{aligned} &\max_{q(\mathbf{x}\,|o)} \int_{\mathbf{x}} q(\mathbf{x}\,|o)(R(\mathbf{x}) + \log \tilde{q}(o|\,\mathbf{x}))d\,\mathbf{x} + \mathcal{H}(q(|o)) \\ &\text{s.t. KL}(q(\mathbf{x}\,|o)||\tilde{q}(\mathbf{x}\,|o)) \leq \epsilon(o). \end{aligned} \tag{2.3}$$

Here, VIPS adds a trust region constraint to the optimization, which is well-known in RL [6, 8, 15]. It ensures that the updated distribution $q(\mathbf{x}\,|o)$ stays close to the previous $\tilde{q}(\mathbf{x}\,|o)$. Therefore a too greedy update of the policy is prevented and the exploration-exploitation trade-off can be controlled using the threshold $\epsilon(o)$. Since the component update only consists of an ordinary Gaussian model, VIPS can employ the MORE algorithm [6] to obtain a closed-form solution for the optimization. Furthermore, VIPS++ [16] uses a replay buffer [17, 18] to store samples, which can later be reused for the Monte Carlo estimate of the objective via importance weighting.
For the weights of the mixture model, the following optimization problem is solved:

$$\max_{q(o)} \sum_{o} q(o)R(o) + \mathcal{H}(q(o)).$$

While the original version of VIPS adds a constraint here as well, the constraint is omitted in VIPS++ [16]. The problem can be solved in closed form using

$$q(o) = \frac{\exp(R(o))}{\sum_o \exp(R(o))},$$

where $R(o)$ denotes the Monte Carlo estimate of the component objective [16].
Another feature of VIPS is that it can dynamically add and remove mixture elements during the optimization. Based on a set of heuristics, it places new components on samples which promise the largest initial reward, and removes unsuccessful components. The new mean is selected from the replay buffer of previous samples $\mathbf{x}_s$ based on the criteria

$$\tilde{R}_{\mathbf{x}_s}(o_n) = R(\mathbf{x}_s) - \log\left((1 - q(o_n))q(\mathbf{x}_s) + q(o_n)\exp\left(\frac{1}{2}D - \mathcal{H}_{\text{init}}\right)\right),$$

where D represents the number of dimensions of $\mathbf{x}$ and $o_n$ the new mixture component. The selection criteria is further refined by considering multiple initial component weights [16]. The initial entropy $\mathcal{H}_{\text{init}}$ is set to a weighted average of the entropy of all previous components.

## 2.3. Amortized Variational Inference

In amortized VI [3, 19], an inference or recognition model is trained to predict the outcome of a VI optimization procedure. Amortized VI is important for latent variable models: while we can use VI as described in Sec. 2.1, the posterior distribution has to be inferred for many observations in some scenarios. Since optimizing the ELBO for each observation from scratch would be very costly if the number of observations is large, amortized VI trains a recognition model using VI across all data points.

In general, amortized VIPS aims at approximating a conditional distribution $p(\mathbf{x} \mid \mathbf{z})$ using a tractable and parameterized function $q(\mathbf{x} \mid \mathbf{z})$. The optimization objective in the conditional case is defined as minimization of the expected KL-divergence, where the expectation is taken with respect to the distribution of context variables $\mathbf{z}$,

$$\min_{q(\mathbf{x} \mid \mathbf{z})} \mathop{\mathbb{E}}_{p(\mathbf{z})} \mathrm{KL}\left(q(\mathbf{x} \mid \mathbf{z}) || p(\mathbf{x} \mid \mathbf{z})\right). \tag{2.4}$$

As in the non-conditional case, $p(\mathbf{x} \mid \mathbf{z})$ is not required explicitly, but could also be given as an unnormalized conditional density function $\tilde{p}(\mathbf{x} \mid \mathbf{z})$ with $p(\mathbf{z}|\mathbf{x}) = \tilde{p}(\mathbf{x} \mid \mathbf{z})/Z(\mathbf{z})$ or even as the unnormalized joint distribution $\tilde{p}(\mathbf{x}, \mathbf{z})$ with $p(\mathbf{x} \mid \mathbf{z}) = \tilde{p}(\mathbf{x}, \mathbf{z})/Z(\mathbf{z})$. Since the expectation over context variables $\mathbf{z}$ has to be taken in Eq. 2.4, either a dataset of samples ($\mathbf{z} \sim D$) or an explicit distribution $p(\mathbf{z})$ as an additional input are required. Using the same derivation as in the non-amortized case, the conditional ELBO can be written as:

$$L = \mathop{\mathbb{E}}_{p(\mathbf{z})} \left[ \int_{\mathbf{x}} q(\mathbf{x} \mid \mathbf{z}) \log \tilde{p}(\mathbf{x}, \mathbf{z}) d\,\mathbf{x} + \mathcal{H}(q(\mathbf{x} \mid \mathbf{z})) \right]. \tag{2.5}$$

The interpretation of VI as an RL problem [5] is even clearer in the conditional case, because the context variable $\mathbf{z}$ could be seen as the state. With $\mathbf{x}$ interpreted as the actions, $q(\mathbf{x} \mid \mathbf{z})$ as the policy and $\log \tilde{p}(\mathbf{x}, \mathbf{z})$ as the Q-function, Eq. 2.5 corresponds to reward maximization with an additional entropy objective in a Markov decision process with an action-independent transition function. This connection to maximum-entropy RL is rather natural, considering that it can be derived from the control-as-inference view [20]. In this setting, maximum-entropy RL is stated as minimizing the KL-divergence between the trajectory distribution of the policy and the optimal distribution and solved using VI. Thus, the connection between VI and maximum-entropy RL is bidirectional.

# 3. Method

In this section, we will present amortized VIPS (aVIPS). First, we describe an amortized VIPS lower bound. Based on the separated component objectives, we present a gradient-based trust region algorithm and a resampling strategy. Next, heuristics to add and remove components are discussed. We perform an analysis of a numerical instability within the mixture objective, which we resolve using an adaptive regularization scheme. Finally, we present a version of aVIPS using parameter sharing.

## 3.1. Main Objective

We define our posterior distribution $q(\mathbf{x} \mid \mathbf{z})$ as a conditional GMM consisting of a categorical distribution $q(o \mid \mathbf{z})$ over the mixture's weights $o$ and Gaussian component distributions $q(\mathbf{x} \mid o, \mathbf{z})$:

$$q(\mathbf{x} \mid \mathbf{z}) = \sum_o q(o \mid \mathbf{z}) q(\mathbf{x} \mid o, \mathbf{z}). \tag{3.1}$$

The parameters of each component distribution and the categorical one are predicted using separate neural networks. The component covariance matrices $\Sigma$ are parameterized by predicting their lower triangular Cholesky factor $L$ with $\Sigma = LL^T$ to ensure positive definiteness.

By inserting the model definition into the standard amortized VI objective function, we obtain the conditional ELBO:

$$L(\mathbf{z}) = \sum_o q(o \mid \mathbf{z}) \mathop{\mathbb{E}}_{q(\mathbf{x} \mid o, \mathbf{z})} (\log \tilde{p}(\mathbf{x} \mid \mathbf{z}) - \log q(\mathbf{x} \mid \mathbf{z})). \tag{3.2}$$

Like in the default VIPS setting, the objective for a single component $o$ is dependent on the other components. In order to separate the loss terms, we can use the derivation from VIPS in Sec. 2.2 and simply replace the distributions with conditional ones. Thus,

by adding the log-responsibilities of the auxiliary distribution $\log \tilde{q}\,(o|\,\mathbf{x}, \mathbf{z})$ as an artificial zero, we receive the conditional VIPS lower bound,

$$L(\mathbf{z}) \geq \sum_o q(o|\,\mathbf{z}) R(o, \mathbf{z}) + \mathcal{H}(q(o|\,\mathbf{z})), \tag{3.3}$$

with the context-wise component reward $R(o, \mathbf{z})$:

$$R(o, \mathbf{z}) = \mathop{\mathbb{E}}_{q(\mathbf{x}|o,\mathbf{z})} \left(\log \tilde{p}\,(\mathbf{x}\,|\,\mathbf{z}) + \log \tilde{q}\,(o|\,\mathbf{x}, \mathbf{z})\right) + \mathcal{H}\left(q\,(\mathbf{x}\,|o, \mathbf{z})\right).$$

In Eq. 3.3, the objective function is now the sum over individual component objectives. As in non-amortized VIPS, Eq. 3.3 is updated in an EM fashion. In the E-step, $\tilde{q}(\mathbf{x}\,|\,\mathbf{z})$ is reset to be equal to $q(\mathbf{x}\,|\,\mathbf{z})$. During the M-step, the model is updated using gradient ascent.

The gradient updates have to be performed based on a Monte-Carlo gradient estimate. Since the samples $\mathbf{x}$ will be drawn according to the Gaussian component $q$, the reparameterization trick can be utilized [3, 21]. The reparameterization trick leads to a gradient estimate with a relatively low variance in most cases [21, 22, 23]. In this method, samples are drawn from a noise distribution first, i.e. $\epsilon \sim \mathcal{N}(\epsilon; 0, I)$, and then transformed to the output space using $\mathbf{x} = f(\epsilon) = \mu(\mathbf{z}) + L(\mathbf{z})\epsilon$.

If we performed only a single gradient step in each M-step, the VIPS bound is actually almost identical to the original ELBO: by rewriting Eq. 3.3 using Bayes' rule as

$$\begin{aligned} L(\mathbf{z}) \geq \sum_o q(o|\,\mathbf{z}) \big( & \mathop{\mathbb{E}}_{q(\mathbf{x}|o,\mathbf{z})} (\log \tilde{p}(\mathbf{x}\,|\,\mathbf{z}) - \log \tilde{q}(\mathbf{x}\,|\,\mathbf{z})) \\ & - \mathrm{KL}\left(q\,(\mathbf{x}\,|o, \mathbf{z})\,||\tilde{q}\,(\mathbf{x}\,|o, \mathbf{z})\right)\big) - \mathrm{KL}\left(q(o|\,\mathbf{z})||\tilde{q}(o|\,\mathbf{z})\right), \end{aligned}$$

it is easy to see that the KL terms vanish if $\tilde{q}(o|\,\mathbf{x})$ and $q(o|\,\mathbf{x})$ are equal. The only difference to the original ELBO in Eq. 3.2 for the gradient update is that $\tilde{q}(o|\,\mathbf{x})$ does not contain the parameters for which gradients are calculated, but only a copy of them. Interestingly, this form corresponds to the method of Roeder et al. [24], who showed that blocking the gradient through the term $-\log q(\mathbf{x}\,|\,\mathbf{z})$ corresponds to an unbiased gradient estimator of the ELBO gradient with lower variance.

Despite being almost identical for vanilla single-step gradient ascent, the amortized VIPS objective allows us to formulate each component objective separately as a trust region constrained problem and to perform importance resampling of the input for each component individually as described in the following sections.

## 3.2. Trust Region Optimization

Information-geometric trust regions are an important tool to prevent a model from converging too fast and exploiting local minima. To implement trust regions, we modify the component's objective, i.e. all terms of Eq. 3.3 which depend on its parameters, by constraining the expected KL-divergence to the auxiliary distribution:

$$\begin{aligned} &\max_{q(\mathbf{x}\,|o,\mathbf{z})} \mathop{\mathbb{E}}_{p(\mathbf{z})} q(o|\,\mathbf{z}) R(o,\mathbf{z}) \\ &\quad \text{s.t.} \mathop{\mathbb{E}}_{p(\mathbf{z})} q(o|\,\mathbf{z}) \mathrm{KL}\left(q\left(\mathbf{x}\,|o,\mathbf{z}\right)||\tilde{q}\left(\mathbf{x}\,|o,\mathbf{z}\right)\right) \leq \epsilon. \end{aligned} \tag{3.4}$$

VIPS uses MORE to solve the constrained component objective, which is not applicable in the conditional case. Extensions of MORE to the conditional case exist, but in closed-form only for linear models [25]. Prior work in the reinforcement learning literature has addressed the challenge to perform trust region constrained updates with neural networks [7, 8, 9]. We follow recent work [9, 26] and solve Eq. 3.4 by optimizing the Lagrangian dual function directly using a fixed number of gradient ascent steps. As proposed by [9], we split the Gaussian KL divergence into a part containing the terms depending on the mean $\mu$ of $q\left(\mathbf{x}\,|o,\mathbf{z}\right)$,

$$\mathrm{KL}_\mu(o,\mathbf{z}) = (\tilde{\mu}(o,\mathbf{z}) - \mu(o,\mathbf{z}))^T \tilde{\Sigma}(o,\mathbf{z})^{-1} (\tilde{\mu}(o,\mathbf{z}) - \mu(o,\mathbf{z}))/2$$

and a part containing the covariance,

$$\mathrm{KL}_\Sigma(o,\mathbf{z}) = (\log \frac{|\tilde{\Sigma}(o,\mathbf{z})|}{|\Sigma(o,\mathbf{z})|} + \mathrm{tr}(\tilde{\Sigma}(o,\mathbf{z})^{-1}\Sigma(o,\mathbf{z})) - D)/2.$$

By constraining both separately, the covariance bound $\epsilon_\Sigma$ can be set to a smaller value, thus causing the component's entropy to decrease slower and encouraging exploration. Including the separated trust region, the new component objective is

$$\max_{q(\mathbf{x}\,|o,\mathbf{z})} \min_{\alpha_{\mu,o},\alpha_{\Sigma,o}} \mathop{\mathbb{E}}_{p(\mathbf{z})} q(o|\,\mathbf{z}) \left(R(o,\mathbf{z}) \; + \alpha_{\mu,o}(\epsilon_\mu - \mathrm{KL}_\mu(o,\mathbf{z})) + \alpha_{\Sigma,o}(\epsilon_\Sigma - \mathrm{KL}_\Sigma(o,\mathbf{z}))\right),$$

with component-wise Lagrangian multipliers $\alpha_{\mu,o} \geq 0$ and $\alpha_{\Sigma,o} \geq 0$.

### 3.2.1. Mixture Weight Optimization

The mixture weights will be updated similarly to the component objective using a trust region constraint on the categorical distribution:

$$\begin{aligned} \max_{q(o|\,\mathbf{z})} \; & \mathop{\mathbb{E}}_{p(\mathbf{z})} \sum_o q(o|\,\mathbf{z}) R(o, \mathbf{z}) \\ \text{s.t.} \; & \mathop{\mathbb{E}}_{p(\mathbf{z})} \mathrm{KL}\left(q(o|\,\mathbf{z}) || \tilde{q}(o|\,\mathbf{z})\right) \leq \epsilon_o. \end{aligned} \tag{3.5}$$

In amortized VIPS, the mixture weights are especially important, since they define the influence of an input space sample on the component objective. This behavior is in contrast to standard VIPS, where the weight of a component can be omitted in its objective, and may lead to instabilities in the training: in the beginning, the components will have very different objective function values due to the random initialization. Since the optimal weights are given by a softmax over the expected component rewards [5], these initialization differences alone would lead to a weight distribution where one component would have a weight of almost one. Therefore a component would only be incentivized to learn for inputs where it was by random initialization the best one and would not explore the rest. Adding a KL constraint to the weight objective limits the possible change of component weights and stabilizes a component's input distribution.

### 3.2.2. Adaptive Learning Rate Selection

In contrast to the RL application, the "environment" in many VI tasks, such as IK, is cheap to sample and running multiple network updates on the same set of previously acquired output samples is not necessary. With IK as target function, the overall computational effort is more important than the number of samples and the dual function needs to be solved in as few steps as possible. This requirement leads, however, to a few practical issues.

First, since the loss and gradients are typically very high in the beginning, the Lagrangian multipliers must in turn reach very high values. To counteract this imbalance, we adjust for each component the context-wise targets $R(o, \mathbf{z})$ by dividing through its standard deviation, measured in the M-step's first batch.
Second, it is necessary to adapt the learning rate: during the course of training, the entropy will become increasingly small and the same change in parameter space will lead to a much higher KL divergence. Even a single gradient step could already lead to a KL-divergence greater than $\epsilon$. Such behavior can lead to undesired consequences:

if the step size is too large, the model can start to oscillate around the allowed trust region. Since the KL bound is not fulfilled in this case, the Lagrangian multiplier will grow. However, since the problem was due to the step size, not the penalty weight, the KL will not sink and hence the multiplier diverges.
In this work, we use a simple heuristic to prevent such instabilities. We assign each component (and the categorical distribution) its own learning rate and adapt it so that the component KL does not change too quickly. Therefore, we define a desired average change in KL $\Delta_{\text{KL,d}}$, and measure its actual value $\Delta_{\text{KL}}$ per component. By assuming a linear relation between learning rate and change in Kl, we set the new learning rate to a moving average of $s^{t+1} = \Delta_{\text{KL,d}}/\Delta_{\text{KL}} s^t$. Furthermore, $\Delta_{\text{KL}}$ is measured by taking the maximum of the change in KL in the beginning and in the end of an M-step, because it is on the one hand important that the KL does not rise too fast, and on the other hand it should be able to converge at the end. The learning rate of the Lagrangian multipliers is not adapted and has to be chosen manually to be high enough to enforce the bound within the limited number of gradient steps. However, if the learning rate is chosen too high, the penalty will react too aggressively to temporary violations or outliers in the batch-wise measurement of the expected KL divergence.
Third, it has also shown to be helpful to reset the used ADAM [27] optimizer's exponential moving average of the gradient's mean and variance at each E-step. Since each M-step involves solving a different constrained optimization problem, the optimizer would have ideally converged in the end. Thus, we would expect the gradients to have a different statistic compared to when the optimization of the next M-step is started, making the current estimates of mean and variance of the last one useless. In the same view of the M-step as an optimization problem to be solved in a small number of steps, it proved to be necessary to reduce the default smoothing factor of the variance moving average. Otherwise, the variance estimate was not able to adapt fast enough.

In conclusion, such repeated solving of the trust region optimization problem requires to carefully configure the optimization parameters, as well as to balance and adapt the learning rate of model and multipliers. In the future, it would be interesting to reformulate the learning rate adaptation as a control problem, where the variance in KL divergence change would represent the inputs and the learning rates the outputs. The learning rate could then be adapted using an algorithm from control theory, e.g. a PI controller. A similar ideas was presented in ControlVAE [28], where the weight of the KL towards the prior in the VAE objective is adapted. Additionally, it would be helpful to consider the actually reached KL bound as an input. Otherwise, it can happen that the desired KL divergence is not reached if the model learning rate is too low or the multipliers are adapted too aggressively.

## 3.3. Context Resampling

Since components typically specialize and cover only parts of the input space, many samples from the context distribution $p(\mathbf{z})$ will only have very little influence on a component's objective. In order to evaluate a component only on relevant inputs $\mathbf{z}$, we can reinterpret the component's objective as an expectation over a component input distribution $q(\mathbf{z}\,|o)$ by applying Bayes' rule:

$$\int_{\mathbf{z}} p(\mathbf{z})q(o|\,\mathbf{z})R(o,\mathbf{z})d\,\mathbf{z} = q(o)\int_{\mathbf{z}} q(\mathbf{z}\,|o)R(o,\mathbf{z})d\,\mathbf{z}\,. \tag{3.6}$$

However, $q(\mathbf{z}\,|o)$ can not be sampled directly. Therefore, we use sampling importance resampling [29, 30]: for the component update, we take a batch of samples of $p(\mathbf{z})$, assign each one importance weights $q(\mathbf{z}\,|o)/p(\mathbf{z}) = q(o|\,\mathbf{z})/q(o)$ and draw a subset of the batch accordingly. Afterwards, each component will only be evaluated on relevant contexts. The sampling is performed with replacement, which can lead to evaluating a model on the same context more than once. However, this procedure is not entirely wasteful, because different output samples for each duplicate input will be drawn subsequently. Furthermore, since the weight update requires to evaluate all components on the same input, a full update step without resampling is performed in a predefined interval.

## 3.4. Adapting the Number of Mixture Components

Adding and removing components is an important part in non-conditional VIPS. In this section, we will present some strategies to include these aspects into aVIPS.

### 3.4.1. Adding Components

In VIPS, adding components to promising regions plays a major role in its exploration process. The mean value of a new component is initialized at high reward samples which weren't sufficiently covered yet. However, such a procedure is more difficult to realize in the conditional setting, since the conditional probability density functions consist of neural networks, which can not be initialized to fulfill a desired behavior instantaneously.

We investigate three different adding strategies. In all cases, a new component is added

in a given interval. First, we try to initialize the model's mean and covariance bias term according to the VIPS heuristic presented in Sec. 2.2.
Second, a natural idea to reach a desired initial component behavior is pre-training. In particular, it seems promising to pre-train components on contexts on which the current model is performing worst. Therefore, we select the k contexts with the worst model performance out of each batch and use the latest few batches as a pre-training data set. Using this data set as the input distribution, the new component alone is trained using Eq. 3.4. Selecting contexts according to the ELBO assumes that the normalization constant of the target distribution is constant w.r.t to the context. Otherwise, even a perfect target approximation on all contexts would have a difference in ELBO and contexts with a smaller normalization constant would be undesirably preferred. The same assumption has to be made for selecting the bias according to the VIPS heuristic. Another approach would be to use the VIPS non-contextual mean initialization criteria as a regression target and to pre-train the component to match the VIPS heuristic on several contexts. However, such procedure is difficult since the most promising sample on a neighboring context might come from a different mode by chance, making the pre-training target non-smooth.
Third, it may be reasonable to simply copy a successful component. While this strategy would not lead to discovering new modes, it may help to cover the already found modes better (especially if they are non-Gaussian). The entropy part in the ELBO will push the components away from each other. We select the component to duplicate randomly according to their average weight $q(o)$.

Another challenge with adding a new component is the initialization of its weight. It is important to initialize a new component with a sufficiently small weight, because its reward will be comparatively bad after initialization and thus the new component will worsen the overall ELBO strongly otherwise. Furthermore, it may be desirable to initialize the weights such that the component covers the input space uniformly, since it can not be automatically foreseen for which contexts a component will be successful. However, the influence of a context on the component objective is proportional to $q(o|\,\mathbf{z})/q(o)$ (Sec. 3.3). For a small average weight $q(o)$, the absolute deviation of $q(o|\,\mathbf{z})$ is hence only allowed to be very small.

Implementing these requirements within a neural network is difficult as well. In this work, we simply choose to set the bias sufficiently small and scale down the linear projection from the last layer output to the new component's weight in order to reduce the variance of the relative weights. However, for most weight activation functions, these two steps are not guaranteed to work. In a classical softmax with $\mathbf{w}_i(\mathbf{a}) = \exp(\mathbf{a}_i)/\sum_j \exp(\mathbf{a}_j)$ for example, the sum over the previous pre-activations $\mathbf{a}_j$ will vary from context to context. Hence, the pre-activation $\mathbf{a}_i$ to reach a desired output weight $\mathbf{w}_i(\mathbf{a})$ will vary from input to

input as well. This problem makes an initialization scheme, where the new component's weight pre-activation $\mathbf{a}_i$ is simply everywhere the same small value, difficult.

Such variation can be partly controlled by limiting the maximum and minimum pre-activation value $\mathbf{a}_j$, for example using a sigmoid. Hence, the maximum possible the normalization constant is limited. By setting the minimum $\mathbf{a}_i$ to a value greater zero, we can also restrict the minimum normalization constant. Furthermore, we decide to use a squared cosine as weight activation function. Besides mapping the components to $[0, 1]$, the cosine does not have saturation areas. Additionally, a low and high value can be reached with a limited input value: using a sigmoid, the necessary bias to reach a low initial weight would need to be big and hence many iterations may be necessary to reach a relevant weight value.

### 3.4.2. Removing Components

VI is an exploration problem and thus some components can converge to bad solutions. Such components will retrieve only a small expected weight $q(o)$ and have almost no influence on the output distribution. Therefore we delete components whose average weight falls below a predefined threshold in order to reduce the model size. If adding components is included at the same time, a second necessary condition would be to check if the component objective is not increasing anymore.

## 3.5. Covariance Regularization

Since the covariance matrix is required to be positive-definite, we use the log-Cholesky parameterization [31, 32], $\Sigma = LL^T$. However, during some of our experiments, we noticed numerical instabilities in the training procedure, which would cause the entropy of the model to diverge to very high values, meaning the log-likelihood $q(\mathbf{x} \,|\, \mathbf{z})$ of the model's own samples becomes increasingly low. In consequence, the ELBO reaches very high values.

The cause of this instability proved to be the triangular matrices $L$: an ill-conditioned $L$ will lead to numerical errors in its inverse, which is necessary to compute the Mahalanobis distance in the Gaussian log-likelihoods,

$$\log \mathcal{N}(\mathbf{x} \,|\, \mu, \Sigma) = -\frac{1}{2} \left[ (\log |2\pi\Sigma|) + (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) \right] .$$

If we insert the true inverse Choleky factor $L^{-1}$ and account for a numerical inversion error $\Delta$, the Mahalanobis distance becomes

$$\begin{aligned} d(\Delta) =& (\mathbf{x}-\mu)^T (L^{-1}+\Delta)^T (L^{-1}+\Delta)(\mathbf{x}-\mu) \\ =& (\mathbf{x}-\mu)^T (L^{-T}L^{-1} + L^{-T}\Delta + \Delta^T L^{-1} + \Delta^T\Delta)(\mathbf{x}-\mu), \end{aligned}$$

where $\Delta$ denotes the difference between the true inverse and the numerically derived one. The expected difference to the true distance $(\mathbf{x}-\mu)^T L^{-T}L^{-1}(\mathbf{x}-\mu)$ can be written as

$$\mathop{\mathbb{E}}_{\Delta}(\mathbf{x}-\mu)^T (L^{-T}\Delta + \Delta^T L^{-1} + \Delta^T\Delta)(\mathbf{x}-\mu) \geq 0. \tag{3.7}$$

For a zero-mean distributed $\Delta$ with uncorrelated entries, the first two terms have expectation zero. However, because $\Delta^T\Delta$ is positive semi-definite, the third term is positive or zero for every $\Delta$, therefore also being in expectation greater or equal zero. Hence, errors in the inversion of $L$ lead to an overestimation of the Mahalanobis distance, and subsequently to an overestimation of the model's entropy.

If one would directly learn a Cholesky factor of the inverse, i.e. $\Sigma^{-1} := SS^T$ the bias in Eq. 3.7 seems to be avoidable. However, in this case the sampling procedure would require inversion of $S$, i.e. $\mathbf{x} = S^{-T}\epsilon + \mu$ and therefore the same term quadratic in the inversion error would occur:

$$\begin{aligned} d(\Delta) =& ((S_b^{-T}+\Delta)\epsilon + \mu_b - \mu_a)^T \Sigma_a^{-1} ((S_b^{-T}+\Delta)\epsilon + \mu_b - \mu_a) \\ =& (\Delta\epsilon + c)^T \Sigma_a^{-1} (\Delta\epsilon + c) \\ =& \epsilon^T\Delta^T\Sigma_a^{-1}\Delta\epsilon + \epsilon^T\Delta^T\Sigma_a^{-1}c + c^T\Sigma_a^{-1}\Delta\epsilon + c^T\Sigma_a^{-1}c. \end{aligned}$$

Similar derivations for the expected squared error can be found in [33]. [34] examines the error in the Mahalanobis distance for covariance matrices derived from sample estimates. This bias in the ELBO is problematic for three reasons. First, it will affect the model's training objective and gradients. Second, since the bias is also present at test time, fair comparisons between the models would be impossible. Third, inducing a significant bias due to numerical errors can happen easily: for a high enough learning rate and a small number of samples per update, the covariance matrix will be stretched far in gradient direction. Since the eigenvalues of a covariance matrix correspond to the extension of the Gaussian along its principal axis, a large gradient step will most likely increase the covariance matrix condition number, defined as the quotient between maximum and minimum singular value, i.e. $\kappa(A) = |\sigma_{\text{max}}/\sigma_{\text{min}}|$. For a positive definite, symmetric matrix like $\Sigma$, singular values and eigenvalues are identical. Since on the other hand the

singular values of a general matrix $B$ equal the square root of the eigenvalues of $BB^T$, the condition number of the covariance matrix is the square of its Cholesky factor's condition number, i.e. $\kappa(\Sigma) = \kappa(L)^2$. Thus, an increase in the covariance matrix's condition number will worsen the condition number of $L$ as well and can cause the bias.

The described problem is most likely only relevant for learning mixture of Gaussians with full covariance matrices in a variational setting. For a single Gaussian, the entropy could be calculated in closed-form, without the Mahalanobis distance. Even for GMM's, the most prevalent use case of (amortized) variational inference are VAEs, which contain a bias term which should automatically regularize the covariance matrices.

### 3.5.1. Bound-based Covariance Regularization

To prevent the covariance matrix and hence its Cholesky factor $L$ from becoming ill-conditioned, we need to regularize it. Often, such regularization is performed using a simple offset to the diagonal. However, the necessary offset may change not only between problems, but also over the course of training and between components. We may want to select the lowest offset possible to restrict the solution space of the covariance matrices as little as possible. Furthermore, a fixed offset is not guaranteed to prevent ill-conditioning. In this work, we extend a technique discussed by Rajamäki et al. [35], who proposed to select the necessary regularization constant based on easy-to-calculate bounds on the eigenvalues of $\Sigma$. Different regularization techniques of the covariance matrix to constrain its condition number explicitly have also been presented [36, 37], which come, however, with the cost of an additional eigenvalue decomposition.

The condition number of $\Sigma$ after adding a constant $s$ to its diagonal is $\kappa = (\lambda_{\text{max}} + s)/(\lambda_{\text{min}} + s)$ with minimum and maximum eigenvalues $\lambda_{\text{min}}$ and $\lambda_{\text{max}}$ of $\Sigma$. Given a lower bound $\lambda^l_{\text{min}}$ of $\lambda_{\text{min}}$, an upper bound $\lambda^u_{\text{max}}$ of $\lambda_{\text{max}}$ and a desired maximum condition number $\kappa^u$, we can get an upper bound of the condition number after adding the constant, solve for the necessary constant and get $s = \max\left\{(\lambda^u_{\text{max}} - \kappa^u\lambda^l_{\text{min}})/(\kappa^u - 1), 0\right\}$. Since $\Sigma$ is positive definite due to its Cholesky parameterization, $\lambda_{\text{min}}$ is greater than zero.
In contrast to Rajamäki et al. [35], we use the full Gershgorin's eigenvalue theorem to

calculate the bounds:

$$\lambda_{\text{max}} \leq \max_{i=1,\dots,D} \left\{ \Sigma_{ii} + \sum_{j=1, j\neq i}^{D} |\Sigma_{ij}| \right\}$$
$$\lambda_{\text{min}} \geq \min_{i=1,\dots,D} \left\{ \Sigma_{ii} - \sum_{j=1, j\neq i}^{D} |\Sigma_{ij}| \right\}.$$

As an additional upper bound we can use that the trace of a matrix equals the sum of its eigenvalues [38]. Since the lower bound on the smallest eigenvalue is zero, the maximum eigenvalue is also bounded by the trace, $\lambda_{\text{max}} \leq \text{Tr}(\Sigma)$. This bound alone is lower or equal to the upper bound used by Rajamäki et al. [35]: $\lambda_{\text{max}} \leq D \cdot \max_{i=1,\dots,D} \Sigma_{ii}$.
Another effect of the covariance regularization is that it acts as an implicit penalty on ill-conditioned matrices: if the model predicts a matrix which has to be regularized significantly, the Gaussian mode will expand in a direction where the input covariance matrix was indicating a lower probability density. Since the reverse KL punishes probability density mass in areas with low target probability density, the model should avoid to have a too large condition number. Overall, the regularization procedure is differentiable and does not require parameter tuning, but has the disadvantage that its output consists of full covariance matrices. Hence, additional Cholesky decompositions need to be applied (for sampling and computing likelihoods), leading to a small increase in training time of about 3%.

## 3.6. Shared Encoder

Especially in problems with high dimensional contexts, it may become too costly to keep one network for each component and it would be desirable to share parameters between them. Therefore, we investigate integrating a shared encoder in the model architecture. The encoder $q(\mathbf{s} \mid \mathbf{z})$ projects the input $\mathbf{z}$ into a shared latent space $\mathbf{s}$. By integrating out $\mathbf{s}$, the output distribution can be described as

$$q(\mathbf{x} \mid \mathbf{z}) = \sum_{o} q(o \mid \mathbf{z}) \int_{\mathbf{s}} q(\mathbf{s} \mid \mathbf{z}) q(\mathbf{x} \mid \mathbf{s}, o) d\,\mathbf{s}\,. \tag{3.8}$$

A first possibility would be to choose a Gaussian encoder. In this case, an additional VIPS decomposition would lead to the objective

$$L(\mathbf{z}) \geq \sum_o q(o|\,\mathbf{z}) \int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \Bigg[ \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s}, o)(R(\mathbf{x}, \mathbf{z}) + \log \tilde{q}(o|\,\mathbf{x}, \mathbf{z}) + \log \tilde{q}(\mathbf{s}\,|\,\mathbf{x}, o, \mathbf{z}))d\,\mathbf{x} \\ + \mathcal{H}(q(\mathbf{x}\,|\,\mathbf{s}, o)) \Bigg] d\,\mathbf{s} + \mathcal{H}(q(o|\,\mathbf{z})) + \mathcal{H}(q(\mathbf{s}\,|\,\mathbf{z})),$$

in which the encoder objective is similar to the component one in objective 3.3 (derivation in Appendix A). Eq. 3.8 can only be solved directly if the integral is tractable, e.g. if the components only consist of a linear Gaussian model, $q(\mathbf{x}\,|\,\mathbf{s}, o) = \mathcal{N}(\mathbf{x}; K_o\,\mathbf{s} + k_o, \Sigma_o)$, and using

$$\int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) q(\mathbf{x}\,|\,\mathbf{s}, o) d\,\mathbf{s} = \int_{\mathbf{s}} \mathcal{N}(\mathbf{x}; K_o\,\mathbf{s} + k_o, \Sigma_o) \mathcal{N}(\mathbf{s}\,|\mu(\mathbf{z}), \Sigma(\mathbf{z})) d\,\mathbf{s} \\ = \mathcal{N}(\mathbf{x}\,|k_o + K_o \mu(\mathbf{z}), \Sigma_o + K_o \Sigma(\mathbf{z}) K_o^T).$$

While this approach has interesting properties such as to be able to optimize the linear components using MOTO [25], a variant of MORE [6] for linear policies, the linear structure of the heads, especially the non-conditional covariance matrix, is limiting the model expressiveness.
Thus we use a Dirac distribution instead, i.e. a deterministic encoder output, resolving Eq. 3.8 to simply

$$q(\mathbf{x}\,|\,\mathbf{z}) = \sum_o q(o|\,\mathbf{z}) q(\mathbf{x}\,|f(\mathbf{z}), o).$$

Here, it would also be possible to make the weight distribution dependent on the shared latent space $\mathbf{s}$. The disadvantage of this approach is that we can not perform a VIPS decomposition over the shared latent space, since the KL-divergence to the old Dirac distribution would become infinite. Therefore, we only perform the standard VIPS decomposition, leading to (derivation in Appendix A):

$$L(\mathbf{z}) \geq \int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \sum_o q(o|\,\mathbf{s}) \Bigg[ \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s}, o)(R(\mathbf{x}, \mathbf{z}) - \log \tilde{q}(\mathbf{x}\,|\,\mathbf{z}))d\,\mathbf{x} \\ - \mathrm{KL}(q(\mathbf{x}\,|o, \mathbf{s})||\tilde{q}(\mathbf{x}\,|o, \mathbf{z})) \Bigg] - \mathrm{KL}(q(o|\,\mathbf{s})||\tilde{q}(o|\,\mathbf{z})).$$

We found it to be more successful to update the encoder and heads only in alternating M-steps. When combined with the resampling strategy, we update encoder and weights

simultaneously within the M-step with full batch size. Another challenge concerns the trust regions presented in Sec. 3.2: without considering the component trust regions in the encoder update, they would become obsolete since the encoder update in between the component updates could still change the components with unconstrained, potentially large steps. Using the Gaussian encoder, an equivalent trust region could be added to the encoder objective, which is not possible using the Dirac distribution. To implement a constrained update for the deterministic encoder, we simply constrain the encoder update to fulfill the trust regions of all components.

# 4. Related Work

In this section, we will give an overview over related work concerning generative modeling in general, amortized VI and IK.

## 4.1. Deep Generative Modeling

Amortized VI is a form of deep generative modeling, i.e. using neural networks to represent a (conditional) probability density function. Other generative modeling techniques include maximum likelihood estimation, variational auto-encoders, noise contrastive estimation or GANs.

In maximum likelihood estimation (MLE), the model is optimized by maximizing the likelihood of the dataset. The objective can be reformulated as minimization of the forward KL (or M-projection), $\mathrm{KL}(p||q)$, which incentives $q$ to capture as much of the support of $p$ as possible [39, 40]. In contrast, the reverse KL (or I-projection), $\mathrm{KL}(q||p)$, forces $q$ to be zero where $p$ is zero and focuses on capturing the modes of $p$ [39, 40]. The reverse KL is typically preferable over the forward KL: for example in a grasp prediction scenario, putting a low probability density mass on bad grasp positions and concentrating on a few good grasps is better than trying to catch as much support of $p$ as possible and assigning low quality grasps a high probability.

An important subcase of MLE are Variational Auto-Encoders [3], which train a latent variable model to represent a (conditional) target distribution. VAEs consist of an encoder to predict the latent variable distribution given a data sample, and a decoder is optimized to reconstruct the input given a sample from the latent posterior distribution. The decoder objective corresponds to MLE, but the encoder is trained using amortized inference with the reconstruction error of the decoder plus a latent prior as the target distribution. After training, output samples can be drawn by applying the decoder

to samples from the prior. The likelihood of the trained latent variable model is intractable.

In GANs [41], a discriminator competes with a generator network. The discriminator is trained to distinguish between samples from the true data distribution $p$ and samples from the generator. On the other hand, the generator is trained to be as indistinguishable from the data distribution as possible. Together, this optimization procedure corresponds to a minimax game. In GANs, sampling from the generator is generally done by sampling from a fixed prior and transforming the samples using a trained decoder network. Thus, the trained generator does not allow likelihood computation. GANs are similar to noise contrastive estimation (NCE) [42], which is an estimation procedure to train unnormalized probability distributions. In NCE, the model is trained to have higher probability than a noise distribution on true data samples and to have lower probability on samples of the noise distribution at the same time.

## 4.2. Amortized Variational Inference

Multiple lines of work have been pursued to improve on vanilla amortized VI. In the following, we will discuss different posteriors, optimization techniques and problem formulations.

### 4.2.1. Posterior distributions

A fundamental limitation of a VI algorithm is the chosen posterior distribution. Most approaches in (amortized) VI rely on a single Gaussian with a diagonal covariance matrix, known as mean field VI [1, 2]. Such simple approximations are not flexible enough to capture complex true posterior distributions. In this work, we rely on GMMs instead. GMMs in amortized VI were already used in areas like clustering [43] or semi-supervised-classification [44, 45]. Since mixture models can not be used solely with the reparameterization trick, Jang et al. [45] have proposed to replace the categorical distribution with a Gumbel-Softmax distribution, which allows to differentiate through the component selection sampling path. Kim et al. [46] have applied GMMs to VAEs in general and proposed a boosting approach to avoid mixture collapse. Similarly, Liu et al. [47] have used GMMs for VAE, but proposed to use an upper bound on the KL divergence to their GMM prior in order to make it solvable in closed-form. Additionally, they used Householder transformations [48] to increase the expressiveness of the components.

A prominent class of posterior distributions in VI are normalizing flows [49], which transform a simple base distribution using a series of invertible transformations with an easy-to-calculate Jacobian determinant. Based on the change of variable formula, normalizing flows provide a tractable log-likelihood and sampling procedure. Examples of flows within amortized VI are inverse autoregressive [50, 51], Sylvester [52] or hyperbolic normalizing flows [53].
Multiple authors [54, 55] have investigated an amortized version of Stein variational gradient descent [56]. A network is used to draw samples for its particle-based variational representation, which are then updated using a single Stein variational gradient descent step. The network is subsequently optimized in a supervised fashion w.r.t. the squared distance between its output and the updated particle set.

### 4.2.2. Optimization

Another important aspect of our approach is the use of trust region optimization, a technique especially prominent in reinforcement learning [7, 8]. TRPO [8] performs a natural gradient update on neural networks by performing conjugate gradient descent using Fisher-Vector products. To reduce the computational effort, KFAC [57, 58] uses a block-diagonal approximation of the Fisher matrix. Wang et al. [59] have proposed a linearized version of the trust region update, which blends between the gradient of the primary objective and the gradient of the KL w.r.t. the distribution parameters. In the next step, the manipulated gradient is applied to the model parameters using simple backpropagation. In PPO [7], trust regions are encouraged by clipping the importance weights occurring in their off-policy RL objective. Otto et al. [60] propose a network layer, which reprojects the policy mean and covariance back into the trust region. The layer is derived by optimizing mean and covariance such that they are as close as possible to the original output, but stay in the trust region. However, the trust region layer of each iteration requires the parameters of the previous policy, which is itself dependent on the parameters of its predecessor. To prevent this chain, Otto et al. [60] add a regression loss to the reinforcement learning objective, which incentives the output of the unprojected Gaussian policy to be close to its projection.

In non-amortized VI, closed form solutions for natural gradient descent, i.e. an approximate solution to the trust region constrained VI problem, exist [61]. Similarly, VIPS uses MORE to solve the update in closed-form [6].
In amortized VI, only a few authors have studied trust region methods so far [62, 63]. Kim et al. [63] apply trust region optimization in the form of TRPO [8] for inference in

state space models. Altosaar et al. [62] added a loss term to penalize changes in the output distribution space. In contrast to their approach, we do not add the penalty using a constant scaling factor, but adapt the scaling dynamically.
Another technique to improve exploration is energy tempering [64, 65], where an additional weight is used to increase the influence of the entropy on the overall objective initially. Hence, the increased entropy will facilitate exploration. Afterwards, the original ELBO is retrieved by slowly decreasing the weight to one over the course of the optimization.
Our work presents an amortized version of the VIPS algorithm [5]. Previously, the VIPS loss has already been used in an amortized variant in Expected Information Maximization [66]. We optimize the same loss function, however extend the optimization among other aspects using trust region constraints. Celik et al. [67] have used an amortized VIPS loss for reinforcement learning, however only considering linear models. Interestingly, they do not model the conditional mixture weights directly, but learn the responsibilities over the context space for each component individually instead in order to facilitate component specialization.

### 4.2.3. Objective Functions

Some authors have proposed variations to the default VI objective of minimizing the reverse KL. The Importance Weighted Auto-encoder [68] reinterprets the difference between target and model log-likelihood as an importance sampling estimate of the target density based on a single sample, and hence proposes a stricter lower bound by using multiple samples. Furthermore, different divergence measures have been proposed, for example the Wasserstein distance [69] or forward KL [70].
In semi-amortized VI [71], the model is trained to predict the optimal initial distribution for classical VI, i.e. the model objective is measured in terms of the ELBO after optimizing the predicted distribution w.r.t to the ELBO using a series of gradient steps. Marino et al. [72] have proposed to learn the optimal update step given the last distribution parameters and the ELBO gradient.

## 4.3. Learning Inverse Kinematics

Recently, multiple works have explored generative modeling techniques in order to learn the distribution of possible joint configurations for a given target pose. GANs [41] have

been used by Lembono et al. [12] to model IK tasks with additional constraints (e.g. static stability). Their approach extends the standard GAN with an ensemble of generators and adds IK-specific terms to the generator loss, e.g. the task space error. Additionally, they augment the discriminator input with the forward kinematic solution of the generator output. However, only a fixed target orientation was considered. Different types of GANs have also been evaluated in the context of IK and inverse dynamics [73].
Ames et al. [13] proposed IKFlow, a conditional normalizing flow approach for IK based on Glow [74]. Trained using a maximum likelihood loss, their approach reaches a low positional and angular error on several kinematic chains with up to 10 DoF. Similarly, Kim et al. [75] investigated the application of Block Neural Autoregressive Flows [76].
The IKNet architecture [77] uses a hypernet to predict the weights of one neural network for each robot joint. These networks describe a GMM distribution over the joint angles, conditioned on a sample from the previous joint in the chain.
In this paper, we investigate amortized VI for IK. Pignat et al. [14] have already formulated IK as VI using GMMs. However, they did only apply non-amortized VI. Also related to our approach, Ho et al. [78] used a VAE to generate IK solutions for a Panda robot.
Reinforcement learning has also been applied to the IK problem [79, 80, 81, 82]. For example, Phaniteja et al. [81] utilize deep deterministic policy gradient [83] to solve IK for humanoid robots considering static stability and self-collision.
Another line of work aims at learning the IK of robots which can not be described by a traditional kinematic chain, for example tendon-driven architectures [84] or tube continuum robots [85]. In a hybrid approach, Fang et al. [86] train a neural network to predict both forward kinematics and Jacobian matrix of a soft robot and use them to feed a traditional IK solver.

# 5. Experiments

In this chapter, we will first compare aVIPS to prior work on IK and constrained IK tasks. IK tasks are an interesting example problem for amortized VI, since they do not contain a changing target function. In contrast, the most prominent use case for amortized VI, VAE, contains the decoder network in the target function. The decoder network is optimized at the same time and will adapt itself to the output of the encoder and may even compensate a bad encoder behavior. Since amortized VI is relevant for more use cases, which may contain a non-adapting target function or at least one with a different adaption behavior, it is insightful to use a task solely depending on the amortized VI component. Finally, we will perform a study investigating the effects of aVIPS' main design choices. Since the shared encoder network and component adding did not proved to be advantageous on the considered tasks, the main experiments in Sec. 5.1 and 5.2 are performed without them.

## 5.1. Inverse Kinematics

First, we compare aVIPS to IKFlow [13] on their IK tasks. However, since both methods consider different problem settings (variational inference based on a reward function in contrast to maximum likelihood based on samples), we also had to define and tune a reward function to obtain similar trade-offs between accuracy in position and orientation, and variability. Therfore, we only used the following three experiments from IKFlow: "Panda" (7 DoF), "ATLAS (2013) - Arm and Waist" (9 DoF) and "Valkyrie - Lower Arm" (4 DoF).

### 5.1.1. Setup

Using the target position and orientation (in form of the rotation matrix) as inputs $\mathbf{z}$, the model is trained to approximate a conditional target distribution $\tilde{p}(\mathbf{x} \mid \mathbf{z})$ over the joint

angles $\mathbf{x}$. As proposed by Pignant et al. [14], the target distribution is given as a product of three expert distributions. The first expert term is a normal distribution centered at the target position. The second one is centered at zero and is given as a Gaussian over the geodesic distance to the target orientation. In order to prevent exceeding the joint limits, the third expert is a cumulative distribution function [14], defined by the probability that a normal distribution centered at $\mathbf{x}$ will generate a sample outside the joint limits. It would be also possible to alter the objective by considering the accuracy after one step of a Jacobian-based IK solver in future work [14].
Instead of predicting the joint angles directly, we scale their values to a fixed range such that the mixture initialization can be the same for all robots. The input data for IK can be generated by sampling uniformly within the joint limits and then using forward kinematics to retrieve a feasible goal endeffector pose [13]. As the input is synthetic, we generate goal poses on-the-fly instead of gathering a fixed data set.

Each component network uses four hidden layers with 450 units each, while the gate network uses 800 units per hidden layer. We train with a batch size of 512 for 800 epochs with 1000 batches each. For aVIPS, an M-step is set to span an entire epoch. In the resampling M-steps, we sample 128 contexts out of the 512 proposals per component. Training is started using 70 components, which are removed from the mixture if they reach a weight of under 0.0005.
As additional comparisons, we train a model with a single Gaussian distribution and a naively-trained (gradient descent on the whole model) GMM. However, because we could not achieve stable learning with a vanilla GMM, we also had to apply the covariance regularization. Both models use full covariance matrices and are trained with exponential learning rate decay. The capacity of the Gaussian model was increased with respect to a single component of the GMMs, to five layers with 1536 units each. Evaluation is performed on 10240 target poses with 25 output samples each. We repeat all experiments with ten random seeds and report the 95% confidence interval of the mean.

### 5.1.2. Results

aVIPS shows lower errors on Panda and Valkyrie, however does not reach the accuracy of IKFlow on Atlas (Tab. 5.1). By removing obsolete components, aVIPS learns much smaller models than IKFlow. On the Valkyrie task, a single Gaussian model is a strong baseline that surpasses all other models in terms of accuracy and ELBO, as the 4-DOF chain lacks redundancy for reaching the target pose. On the other tasks, GMMs clearly outperform a single Gaussian, not only in terms of entropy, but also with respect to the

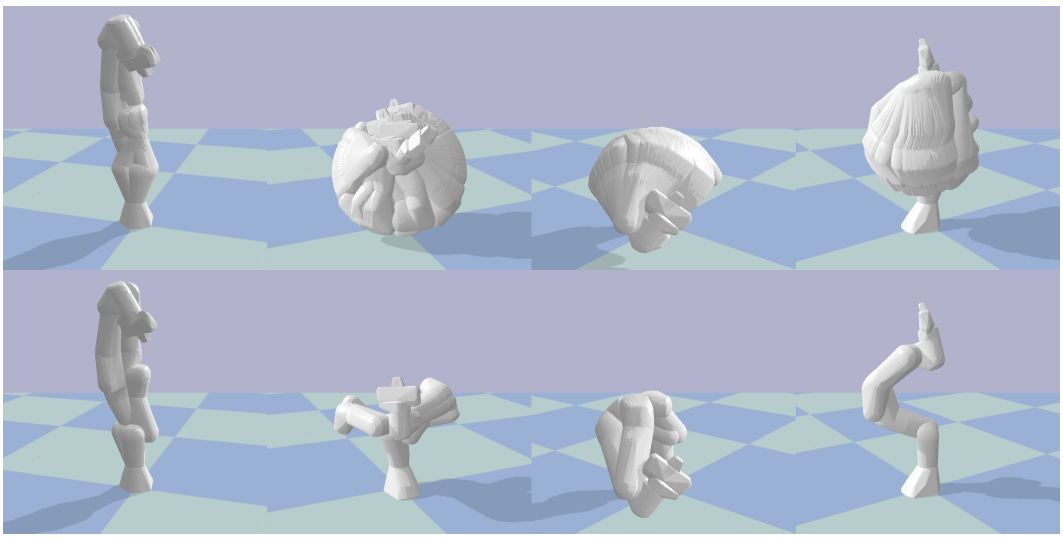


Figure 5.1.: Hundred samples of IKFlow (top) and AVIPS (bottom) on four random target poses.

accuracy. The improved accuracy shows that GMMs can explore the output space better: if one component is stuck in a local optima, another component may still be able to reach a good mode due to different initialization.

For evaluating variability, we can not use the MMD, as proposed in IKFLow [13], because the optimal solution to the VI objective would be different to the data generating distribution. For an exact match, the target density function would have to be a Dirac distribution centered on the exact solution in task space, which would be impossible to learn with VI. Furthermore, IKFlow samples in an artificially extended output space [13]. Thus, it is also not possible to compare the variability in terms of entropy. Instead, Fig. 5.1 shows a visual comparison of the generated samples. It can be seen that aVIPS can find multiple target modes, but covers fewer solutions than IKFlow. However, IKFlow produces some outliers despite the reduced variance of the normalizing flow latent base distribution.

To further compare aVIPS to the VI competitors, we list ELBO and entropy of all three models in Tab. 5.2. Compared to the vanilla GMM and Gaussian, aVIPS reaches higher entropy and generates more diverse mixtures due to the weight KL (further studied in the ablations below). Tab. 5.2 also shows that a naive training of GMMs will almost certainly load to mixture collapse, while aVIPS can clearly increase the weight distribution

| | | Pos. error in mm | Rot. error in ° | # Param. in Mio. |
|---|---|---|---|---|
| Valkyrie | aVIPS | $0.20 \pm 0.01$ | $0.13 \pm 0.01$ | $30.44 \pm 1.63$ |
| | Gaussian | $\mathbf{0.15} \pm 0.01$ | $\mathbf{0.11} \pm 0.01$ | $9.48$ |
| | GMM | $0.22 \pm 0.02$ | $0.14 \pm 0.01$ | $27.45$ |
| | IKFlow | $0.36$ | $0.15$ | $38.36$ |
| Panda | aVIPS | $5.60 \pm 0.04$ | $\mathbf{2.05} \pm 0.02$ | $26.13 \pm 2.12$ |
| | Gaussian | $10.75 \pm 0.11$ | $5.73 \pm 0.09$ | $9.52$ |
| | GMM | $5.80 \pm 0.21$ | $2.19 \pm 0.09$ | $27.83$ |
| | IKFlow | $7.72$ | $2.81$ | $50.93$ |
| Atlas | aVIPS | $3.43 \pm 0.05$ | $1.07 \pm 0.02$ | $16.21 \pm 1.40$ |
| | Gaussian | $4.65 \pm 0.07$ | $1.51 \pm 0.03$ | $9.55$ |
| | GMM | $4.02 \pm 0.11$ | $1.18 \pm 0.04$ | $28.17$ |
| | IKFlow | $\mathbf{2.66}$ | $\mathbf{0.61}$ | $38.36$ |

Table 5.1.: Comparison of aVIPS to IKFlow as well as to a simple Gaussian and GMM trained using VI. The positional and rotational errors are measured in terms of the average Euclidean and geodesic distance respectively. The variance in the number of parameters of aVIPS is due to the component removal. IKFlow diverged for several seeds and, hence, we list the reported results [13] that were based on a single seed.

entropy. Furthermore, the ELBO was significantly increased on Atlas. aVIPS may have been more successful on this task due to a more difficult to explore target space with more dimensions.

| | | Entropy | Weight Entropy | ELBO |
|---|---|---|---|---|
| Valkyrie | aVIPS | $\mathbf{-23.95} \pm 0.10$ | $\mathbf{0.46} \pm 0.09$ | $-8.34 \pm 3.78$ |
| | Gaussian | $-24.05 \pm 0.10$ | N/A | $\mathbf{-1.04} \pm 0.13$ |
| | GMM | $-24.40 \pm 0.03$ | $0.01 \pm 0.01$ | $-10.19 \pm 3.19$ |
| Panda | aVIPS | $\mathbf{-22.70} \pm 0.08$ | $\mathbf{0.78} \pm 0.07$ | $-38.29 \pm 0.82$ |
| | Gaussian | $-24.68 \pm 0.14$ | N/A | $-229.78 \pm 12.65$ |
| | GMM | $-23.60 \pm 0.07$ | $0.10 \pm 0.01$ | $-39.51 \pm 2.34$ |
| Atlas | aVIPS | $\mathbf{-33.93} \pm 0.07$ | $\mathbf{0.36} \pm 0.06$ | $\mathbf{-71.42} \pm 2.65$ |
| | Gaussian | $-36.01 \pm 0.09$ | N/A | $-187.94 \pm 27.52$ |
| | GMM | $-35.22 \pm 0.06$ | $0.04 \pm 0.01$ | $-85.99 \pm 6.08$ |

Table 5.2.: Comparison of aVIPS to a simple Gaussian and GMM model in terms of overall model entropy, entropy of the weight distribution and ELBO. Model entropy was measured in joint space.

## 5.2. Constrained Inverse Kinematics

In our second experiment, we compare to Lembono et al. [12], who trained mixtures of GANs to solve IK tasks with additional constraints for the Panda arm and the Talos humanoid. On their Panda task, the goal is to generate samples with a given target position, while the endeffector should be oriented horizontally. The Talos task is to reach a given right hand position while keeping the feet at a fixed pose and the center of mass over a predefined area to ensure static stability. Combining 28 joint angles (six for each arm and leg) and the floating base position, the output dimension of the Talos task is 31. Both tasks use a fixed set of target positions and also enforce joint limits. For fairness, we equip our component networks with the same architecture as the components in the mixture of GANs and also use a mixture of ten components. Furthermore, we train both models until convergence (longer than the author's original model). Each model is evaluated on 1000 random target points.

| | | Initial Costs | Success Rate | Opt. Steps |
|---|---|---|---|---|
| Panda | aVIPS | $\mathbf{2.17} \pm 2.03$ | $\mathbf{0.96} \pm 0.01$ | $\mathbf{1.08} \pm 0.02$ |
| | GAN | $14.64 \pm 2.08$ | $0.93 \pm 0.02$ | $2.77 \pm 0.18$ |
| Talos | aVIPS | $1.02 \pm 0.15$ | $1.00$ | $1.89 \pm 0.14$ |
| | GAN | $\mathbf{0.68} \pm 0.07$ | $1.00$ | $1.88 \pm 0.07$ |

Table 5.3.: Comparison of aVIPS to the GAN approach by [12] in terms of the initial primary cost sum, the success rate of the subsequently applied IK solver and the number of steps the solver took in case of a success.

To evaluate their IK tasks, Lembono et al. [12] measured performance in terms of success rate and number of optimization steps of a Jacobian-based IK solver after warm-starting it with the predicted solutions. Additionally, they analyze their IK GAN in the context of motion planning algorithm by drawing joint positions for a random target within a relevant space of the task space. However, we observed that the motion planner performance increased neither with a better IK performance nor a higher entropy, thus we hypothesize that this evaluation is to indirect or different from the original IK training objective. Therefore, we exclude this task from our reported experiments.

As shown in Tab. 5.3, aVIPS can generate more accurate samples on Panda, where it can provide better initial samples, leading to a substantial decrease in the number of necessary optimization steps. On the Talos task, the GAN performs better in terms of the initial cost function value. However, both models provide a 100% IK solver success rate and an almost identical number of necessary optimization steps.

Fig. 5.2 depicts a visual comparison of the generated samples. The positional errors are higher than in the previous experiment, because the target function punishes positional errors less strict in the constrained IK tasks. In return, the variability of the generated samples is increased and appears to be on a comparable level to the GAN samples. On Talos, both models predict samples which only show a low variance in the right arm, while the variance in the left arm is high as its endeffector position is not specified in the objective function.

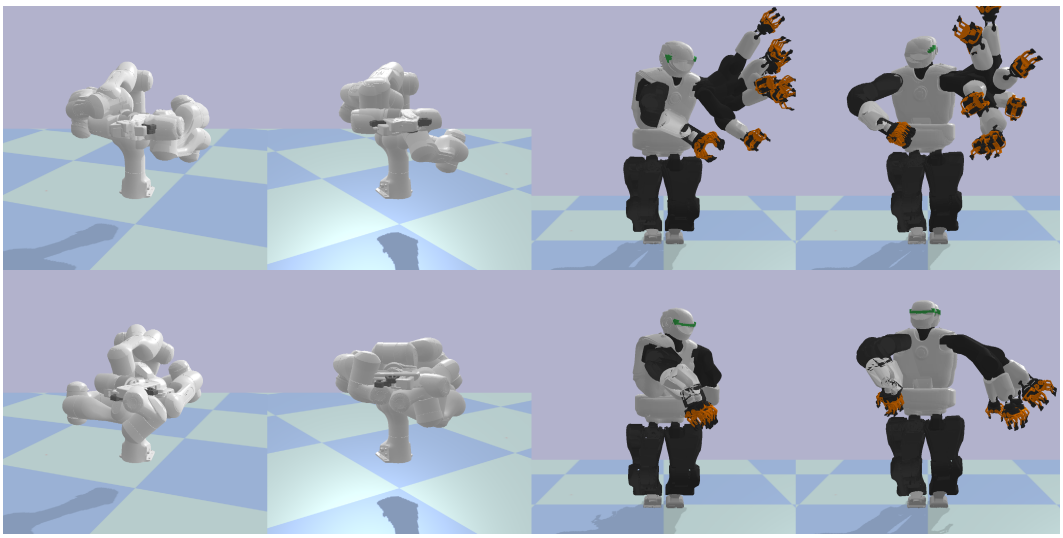


Figure 5.2.: Hundred samples generated by aVIPS (top) and GANs (bottom) for two random target poses for the constrained Panda (left) and Talos (right) IK tasks.

## 5.3. Ablations

aVIPS uses a few extensions to a vanilla GMM amortized VI. In the following, we will examine and discuss their effect. The ablation experiments were performed on the Panda robot task from IKFlow, using five random seeds and mixtures of 20 components with reduced capacity.

### 5.3.1. Trust Region Optimization

Fig. 5.3 shows the influence of the component KL bound. The mean KL bound $\epsilon_\mu$ is set to 20 times $\epsilon_\Sigma$ and the target change is varied to be a third of the KL bound in all three experiments. Lowering the bounds leads to a slower decline and higher final value of the model entropy. Subsequently, the ELBO converges slower in the beginning, but can reach a slightly better level in the end. As seen in the bottom of Fig. 5.3, the algorithm can follow the desired target KL bound accurately.

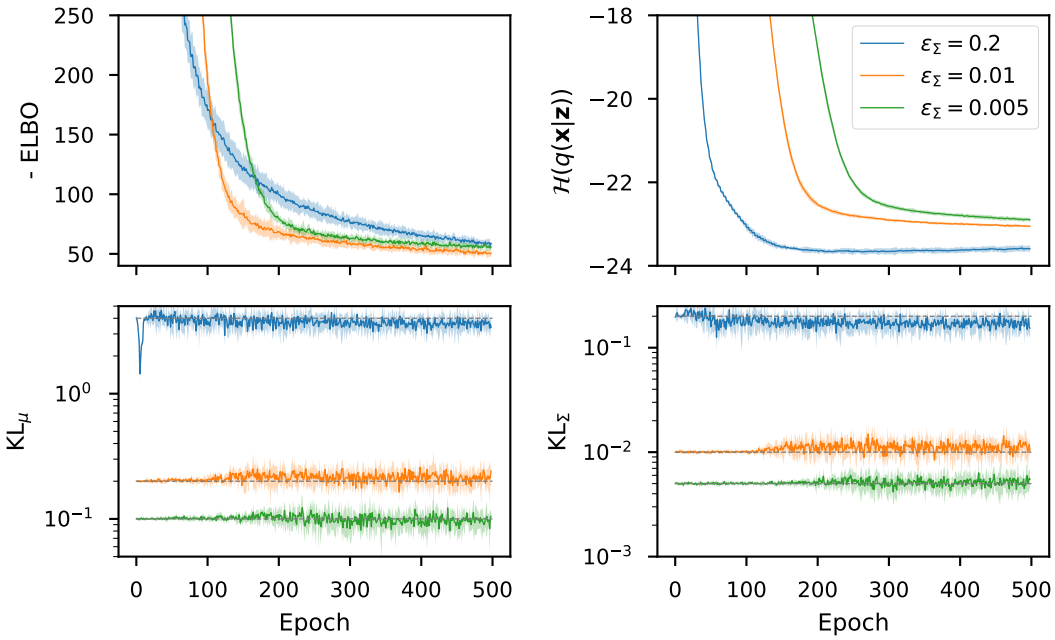



Figure 5.3.: Influence of the component trust region on the ELBO and (expected) model entropy (top). The average mean and covariance part of the component KL divergence are depicted in the bottom. The gray, dashed lines show the corresponding target KL values.

The KL bound on the weights (Fig. 5.4) has a similar effect: the expected entropy of the categorical distribution $H(q(o|\,\mathbf{z}))$ becomes small quickly without imposing a strong enough KL bound. Decreasing the bound leads to a more diverse final mixture. However, it can also be seen that a bound too strong can harm the ELBO by preventing the components from specializing.

### 5.3.2. Resampling

The effect of resampling the component's inputs is shown in Fig. 5.5. Despite using only 128 contexts per component out of the base batch size of 512, resampling does not degrade the accuracy. However, direct comparison to using the full batch is difficult as the resampling strategy updates the mixture weights only every four iterations. Thus, we also

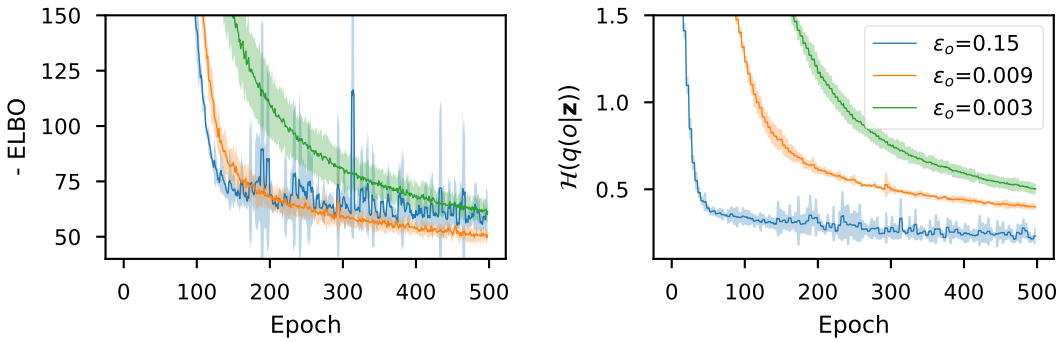



Figure 5.4.: Influence of the categorical trust region on ELBO and entropy of the weight distribution.

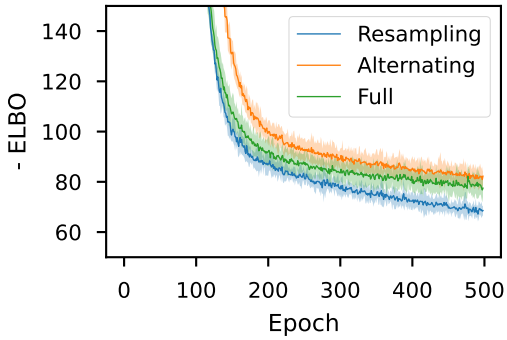



Figure 5.5.: aVIPS with context resampling compared to using the full batch size and alternating between full and reduced batch size without resampling.

compare against a version which switches between using the full batch and only using 128 contexts (and only updating the components) without resampling.
Resampling leads to a better ELBO, demonstrating that its effect is beyond selecting a different batch size. On average, resampling reduced the run time by 33% and the number of output samples by 56% under the chosen batch size hyperparamters.

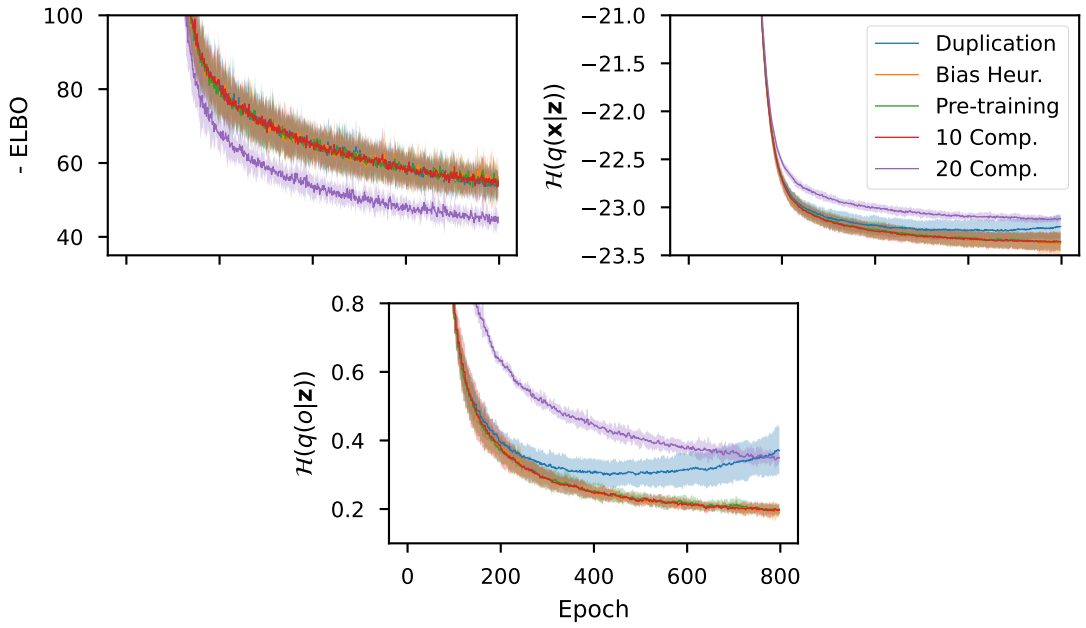



Figure 5.6.: Adding mixture components. Next to the three presented adding strategies, we compare against two mixtures consisting of a fixed number of 10 and 20 components.

### 5.3.3. Adding Mixture Components

As presented in Sec. 3.4.1, we consider three different adding heuristics: selecting the best bias according to the heuristic presented in VIPS, pre-training on the worst approximated contexts and duplicating components. To compare them, we start the mixture using only 10 components, and add a new one every 50 E-steps until a maximum of 20 components is reached.

The results show that adding mixture components did not improve the ELBO (Fig. 5.6). The weight entropy is only improved, i.e. more components are actually used on average, if the components are duplicated. However, one might suspect that this is only due to splitting the previous weight between the original and duplicated component, without covering actually more solutions. As shown in the bottom of Fig. 5.6, the overall model entropy is increased, and thus the component copy actually covers new, different parts of the output space. Within the other two strategies, new components are not successful and the weight

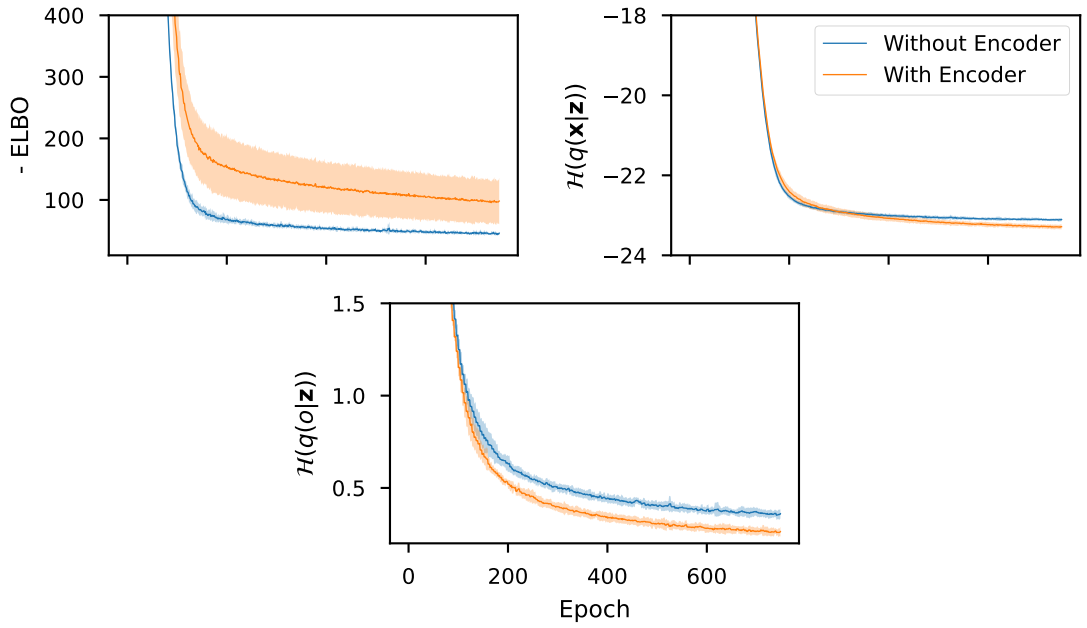



Figure 5.7.: Effect of the encoder on ELBO, model entropy and weight entropy.

entropy is on the level of a mixture with a fixed number of 10 components. While the duplication strategy can reach the entropy of the setting with 20 initial components, it can also not increase the ELBO. A possible explanation could be that the new components do not explore new regions and hence modes may be missed. Additionally, it can be preferable to cover more distant modes instead, and hence we also do not include the duplication strategy into the main version of aVIPS.

### 5.3.4. Encoder

Next, we examine the effect of including a shared encoder into the model architecture. Therefore, we compare a model with no encoder to one with an additional encoder layer (Fig. 5.7). Adding shared layers strongly slows down convergence and also leads to a worse final error, even if more training steps are taken. Also, the runs with an encoder show a lot more variance in the final ELBO. While the model entropy is only slightly affected, the encoder model also learns less diverse mixtures with fewer active components.

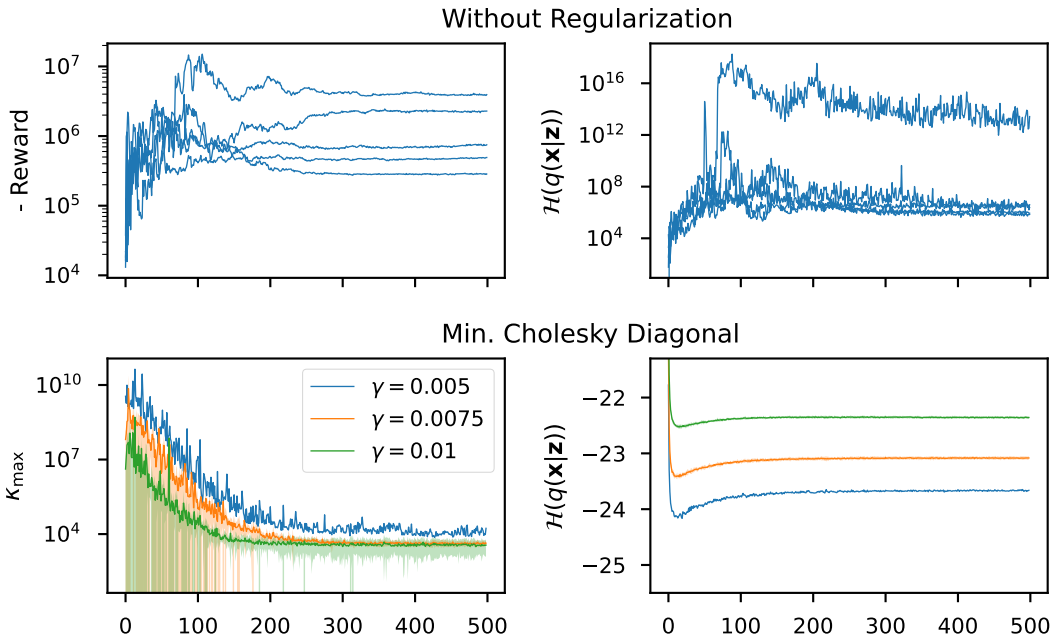



Figure 5.8.: Comparison of different regularization strategies. Top: expected reward and entropy for five different seeds without further regularization. Bottom: maximum measured condition number for different minimum Cholesky diagonal values $\gamma$. For $\gamma = 0.005$, only one seed did not diverge and is hence plotted.

### 5.3.5. Regularization

Finally, the behavior of the regularization procedure is demonstrated. Since component trust regions can at least prevent the divergence of a run due to ill-conditioning, we will only use models without trust regions. Additionally, only a single gradient step per M-step was taken, and the model size was even further decreased due to the number of experiments.

To demonstrate the problem first, Fig. 5.8 shows reward and entropy without further regularization. While the entropy of the model reaches very high values, the reward is not actually being improved. However, the entropy estimate is so high that the overall ELBO is positive and hence appears to be very good.

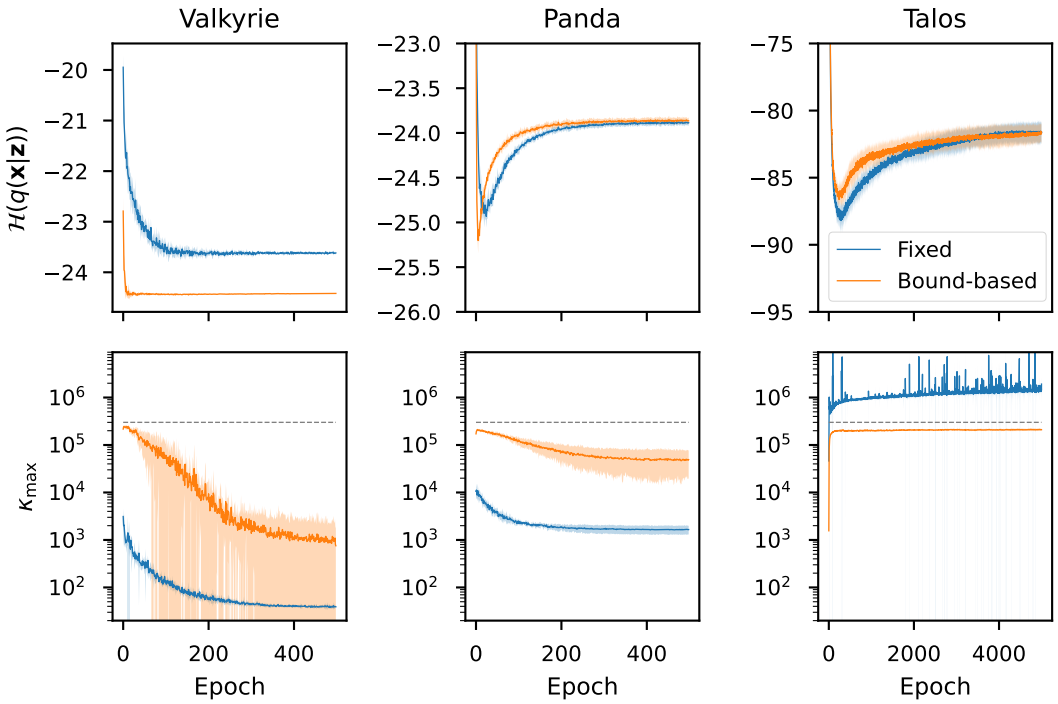



Figure 5.9.: Bound-based covariance regularization compared to a fixed covariance offset on multiple robots with different DoF in terms of entropy and maximum condition number. The desired maximum condition number is shown in gray.

A typical measure to tackle numerical instability is to set the diagonal entries of the Cholesky factor to a minimum value $\gamma$, a method which would not require a further decomposition. While this approach is sufficient for diagonal covariance matrices, the effect on the covariance condition number for lower triangular Cholesky factors is not as straightforward: since $\Sigma = LL^T$, adding an offset to the diagonal of the Cholesky factor $L$ does affect all entries of $\Sigma$ and not only the diagonal entries of $\Sigma$. Hence, the effect on the condition number can not be quantified.
Experiments on the Panda robot demonstrate that this approach nevertheless can lower the maximum encountered condition number $\kappa_{\max}$ (Fig. 5.8). However, $\kappa_{\max}$ was very high in the beginning for all $\gamma$, and dropped about several magnitudes later. The expected entropy seems to be very sensitive to the minimum value, i.e. this regularization approach quickly leads to a notable restriction of possible covariance matrices. Additionally, for

the lowest considered value of $\gamma$ the numerical instability occurred in four out of the five considered seeds. Interestingly, even the entropy of the stable run was slightly higher than the ones encountered if using a different regularization strategy (Fig. 5.9). Hence, even under a setting with a high risk of divergence this method was comparably restrictive.

In the final experiment, we compare the bound-based regularization to simply adding a fixed offset to the covariance diagonal. As shown in Fig. 5.9, both approaches can limit $\kappa_{\max}$ effectively. The advantage of the bound-based offset selection is demonstrated by comparing both approaches on multiple tasks. While the same, fixed offset is too strong on the Valkyrie task and the minimum reached entropy is increased, $\kappa_{\max}$ is comparatively high on the Talos task. In contrast, the bound-based covariance strategy allows the model to generate covariance matrices with similar $\kappa_{\max}$ on all tasks.

# 6. Conclusion

In this work, we investigated how the ideas presented in VIPS can be transferred to amortized VI.

## 6.1. Summary

aVIPS optimizes a lower bound based on the VIPS decomposition to split up the full objective. The separated component objectives and the weight objective are amended with a trust region constraint, which is optimized using gradient descent on the Lagrangian dual function. To circumvent instabilities in the optimization, we proposed a technique to adapt the learning rate according to the sensibility of an update step on the expected KL divergence. Since components are often evaluated on contexts irrelevant to their objective, a context resampling scheme was introduced. Additionally, we presented an analysis of a numerical instability in the GMM loss, and employed an adaptive covariance regularization technique to circumvent it. Furthermore, we presented three approaches of adding mixture components and a framework to include a shared encoder into aVIPS.

To evaluate our approach, we considered IK tasks. aVIPS can compete against recent work using normalizing flows and GANs in terms of accuracy. However, the output coverage is still small compared to normalizing flows. Additionally, experiments regarding the effect of the individual algorithmic components have shown that a shared encoder network does degrade the convergence on the considered tasks. Similarly, we could not measure an advantage of adding components to the mixture. Finally, we have shown that aVIPS, using all advantageous design choices together, learns consistently more diverse mixtures and can reach a better ELBO compared to GMMs trained with vanilla amortized VI.

## 6.2. Future Work

Our work has demonstrated some difficulties regarding the mixture representation. Sharing layers between the components suffered from worse performance and slower convergence. The problem may be related to multi-task learning (MTL): the VIPS loss decomposes the full objective into a set of component objectives, which have to be fulfilled at the same time. Depending on the problem structure, the component density functions to be learned may be too dissimilar and the different component objectives could be conflicting. Similar problems are common in the MTL literature, known under the name of negative transfer [87, 88], i.e. learning multiple tasks together can lead to a worse performance than learning each task on its own. Hence, it may be fruitful to treat amortized VI of GMM as an MTL problem, and apply techniques from the MTL literature. MTL approaches can be divided into optimization and architectural solutions [87].

An interesting architectural solution could be to apply transformer networks [89], which have been shown to be effective in MTL [90]. For translation tasks, transformers can decode the input sentence into an output sentence with variable length. This architecture could be very helpful for GMMs: by decoding the input into a sequence of mixture components, the architecture could on the one hand place the components depending on the previously placed ones and on the other hand it could generate only as many components as really necessary. Since currently all models have to be evaluated on all contexts, the decoding strategy could lead to much smaller models. However, component-wise trust regions could not directly be applied to this architecture due to the variable number of mixture components. To encourage exploration, an entropy bonus could be used instead, which does not rely on an old model.

A simpler change to make the architecture more compact could be to split the mixture into clusters, where each cluster shares its own encoder. Starting with a single component per cluster, we could use the duplication strategy to gradually add new, but similar components to the cluster and hence ensure that the component objectives do not conflict too much within a cluster .

Another problem with aVIPS is that exploration and representation are intertwined. In our experience, increasing the component network capacity can only increase the input space coverage up to a certain degree. Hence, a key component of increasing the model efficiency of VIPS could be to investigate how to train a component to cover contexts. We hypothesize that this behavior is simply due to the component being stuck in a local optima

at the inputs which it does not cover. Hence, it appears to be promising to investigate how to separate exploration of the output space with how to actually represent the components. For example, it might be possible to start classical VI on a set of inputs and to try matching only the most successful components with the component networks using supervised learning and a set matching loss [91].

Finally, VIPS uses a replay buffer to reuse previous output samples in order to increase sample efficiency. A similar approach could be useful for aVIPS, especially for target functions which are more costly to evaluate. Hence, it may be fruitful to build on prior work on replay buffers within deep reinforcement learning [18, 92].

# Bibliography


[1] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[2] C. Zhang, J. Bütepage, H. Kjellström, and S. Mandt, "Advances in variational inference," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 8, pp. 2008–2026, 2018.

[3] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations (ICLR)*, 2014.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 1861–1870, PMLR, 2018.

[5] O. Arenz, G. Neumann, and M. Zhong, "Efficient gradient-free variational inference using policy search," in *International Conference on Machine Learning (ICML)*, pp. 234–243, PMLR, 2018.

[6] A. Abdolmaleki, R. Lioutikov, J. R. Peters, N. Lau, L. Pualo Reis, and G. Neumann, "Model-based relative entropy stochastic search," *Advances in Neural Information Processing Systems*, vol. 28, pp. 3537–3545, 2015.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[8] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 1889–1897, PMLR, 2015.

[9] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. A. Riedmiller, "Maximum a posteriori policy optimisation," in *6th International Conference on Learning Representations (ICLR)*, 2018.

[10] C. W. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 93–101, 1986.

[11] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 928–935, IEEE, 2015.

[12] T. S. Lembono, E. Pignat, J. Jankowski, and S. Calinon, "Learning constrained distributions of robot configurations with generative adversarial network," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 4233–4240, 2021.

[13] B. Ames, J. Morgan, and G. Konidaris, "Ikflow: Generating diverse inverse kinematics solutions," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7177–7184, 2022.

[14] E. Pignat, T. Lembono, and S. Calinon, "Variational inference with mixture model approximation for applications in robotics," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3395–3401, IEEE, 2020.

[15] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[16] O. Arenz, M. Zhong, and G. Neumann, "Trust-region variational inference with gaussian mixture models.," *J. Mach. Learn. Res.*, vol. 21, pp. 163:1–163:60, 2020.

[17] L.-J. Lin, *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[19] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proceedings of the 31th International Conference on Machine Learning (ICML)*, pp. 1278–1286, PMLR, 2014.

[20] S. Levine, "Reinforcement learning and control as probabilistic inference: Tutorial and review," *arXiv preprint arXiv:1805.00909*, 2018.

[21] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning.," *J. Mach. Learn. Res.*, vol. 21, no. 132, pp. 1–62, 2020.

[22] M. Jankowiak and F. Obermeyer, "Pathwise derivatives beyond the reparameterization trick," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 2235–2244, PMLR, 2018.

[23] M. Xu, M. Quiroz, R. Kohn, and S. A. Sisson, "Variance reduction properties of the reparameterization trick," in *The 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 2711–2720, PMLR, 2019.

[24] G. Roeder, Y. Wu, and D. Duvenaud, "Sticking the landing: Simple, lower-variance gradient estimators for variational inference," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 6925–6934, 2017.

[25] R. Akrour, G. Neumann, H. Abdulsamad, and A. Abdolmaleki, "Model-free trajectory optimization for reinforcement learning," in *Proceedings of the 33nd International Conference on Machine Learning (ICML)*, vol. 48, pp. 2961–2970, 2016.

[26] H. F. Song, A. Abdolmaleki, J. T. Springenberg, A. Clark, H. Soyer, J. W. Rae, S. Noury, A. Ahuja, S. Liu, D. Tirumala, *et al.*, "V-mpo: On-policy maximum a posteriori policy optimization for discrete and continuous control," in *International Conference on Learning Representations (ICLR)*, 2019.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

[28] H. Shao, S. Yao, D. Sun, A. Zhang, S. Liu, D. Liu, J. Wang, and T. Abdelzaher, "Controlvae: Controllable variational autoencoder," in *International Conference on Machine Learning (ICML*, pp. 8655–8664, PMLR, 2020.

[29] T. Li, M. Bolic, and P. M. Djuric, "Resampling methods for particle filtering: Classification, implementation, and strategies," *IEEE Signal processing magazine*, vol. 32, no. 3, pp. 70–86, 2015.

[30] S. T. Tokdar and R. E. Kass, "Importance sampling: a review," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 1, pp. 54–60, 2010.

[31] T. Muschinski, G. J. Mayr, T. Simon, N. Umlauf, and A. Zeileis, "Cholesky-based multivariate gaussian regression," *Econometrics and Statistics*, 2022.

[32] J. C. Pinheiro and D. M. Bates, "Unconstrained parametrizations for variance-covariance matrices," *Statistics and Computing*, vol. 6, no. 3, pp. 289–296, 1996.

[33] A. D. Ker, "Stability of the mahalanobis distance: A technical note," *Technical Report CS-RR-10-20*, 2010.

[34] Y. Kobayashi, "Effects of numerical errors on sample mahalanobis distances," *IEICE TRANSACTIONS on Information and Systems*, vol. 99, no. 5, pp. 1337–1344, 2016.

[35] J. Rajamäki and P. Hämäläinen, "Regularizing sampled differential dynamic programming," in *2018 Annual American Control Conference (ACC)*, pp. 2182–2189, IEEE, 2018.

[36] A. Aubry, A. De Maio, L. Pallotta, and A. Farina, "Maximum likelihood estimation of a structured covariance matrix with a condition number constraint," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 3004–3021, 2012.

[37] J. H. Won and S.-J. Kim, "Maximum likelihood covariance estimation with a condition number constraint," in *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*, pp. 1445–1449, IEEE, 2006.

[38] J. K. Merikoski, H. Sarria, and P. Tarazaga, "Bounds for singular values using traces," *Linear Algebra and its Applications*, vol. 210, pp. 227–254, 1994.

[39] C. M. Bishop, "Pattern recognition and machine learning," 2006.

[40] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[42] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 297–304, JMLR Workshop and Conference Proceedings, 2010.

[43] N. Dilokthanakul, P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan, "Deep unsupervised clustering with gaussian mixture variational autoencoders," *arXiv preprint arXiv:1611.02648*, 2016.

[44] D. P. Kingma, S. Mohamed, D. Jimenez Rezende, and M. Welling, "Semi-supervised learning with deep generative models," *Advances in Neural Information Processing Systems (NIPS)*, vol. 27, 2014.

[45] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *5th International Conference on Learning Representations (ICLR)*, 2017.

[46] M. Kim and V. Pavlovic, "Recursive inference for variational autoencoders," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 19632–19641, 2020.

[47] G. Liu, Y. Liu, M. Guo, P. Li, and M. Li, "Variational inference with gaussian mixture model and householder flow," *Neural Networks*, vol. 109, pp. 43–55, 2019.

[48] J. M. Tomczak and M. Welling, "Improving variational auto-encoders using householder flow," *arXiv preprint arXiv:1611.09630*, 2016.

[49] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International Conference on Machine Learning (ICML)*, pp. 1530–1538, PMLR, 2015.

[50] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," *Advances in Neural Information Processing Systems (NIPS)*, vol. 29, 2016.

[51] A. Vahdat and J. Kautz, "NVAE: A deep hierarchical variational autoencoder," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 19667–19679, 2020.

[52] R. Van Den Berg, L. Hasenclever, J. M. Tomczak, and M. Welling, "Sylvester normalizing flows for variational inference," in *34th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 393–402, Association For Uncertainty in Artificial Intelligence (AUAI), 2018.

[53] J. Bose, A. Smofsky, R. Liao, P. Panangaden, and W. Hamilton, "Latent variable modelling with hyperbolic normalizing flows," in *International Conference on Machine Learning (ICML)*, pp. 1045–1055, PMLR, 2020.

[54] Y. Pu, Z. Gan, R. Henao, C. Li, S. Han, and L. Carin, "VAE learning via stein variational gradient descent," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 4239–4248, 2017.

[55] Y. Feng, D. Wang, and Q. Liu, "Learning to draw samples with amortized stein variational gradient descent," in *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence (UAI)*, AUAI Press, 2017.

[56] Q. Liu and D. Wang, "Stein variational gradient descent: A general purpose bayesian inference algorithm," in *Advances in Neural Information Processing Systems (NIPS)* (D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, eds.), pp. 2370–2378, 2016.

[57] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 2408–2417, PMLR, 2015.

[58] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," *Advances in Neural Information Processing Systems (NIPS)*, vol. 30, 2017.

[59] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," in *5th International Conference on Learning Representations (ICLR)*, 2017.

[60] F. Otto, P. Becker, V. A. Ngo, H. C. M. Ziesche, and G. Neumann, "Differentiable trust region layers for deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2021.

[61] W. Lin, M. E. Khan, and M. Schmidt, "Fast and simple natural-gradient variational inference with mixture of exponential-family approximations," in *International Conference on Machine Learning (ICML)*, pp. 3992–4002, PMLR, 2019.

[62] J. Altosaar, R. Ranganath, and D. Blei, "Proximity variational inference," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1961–1969, PMLR, 2018.

[63] G.-H. Kim, Y. Jang, J. Lee, W. Jeon, H. Yang, and K.-E. Kim, "Trust region sequential variational inference," in *Asian Conference on Machine Learning*, pp. 1033–1048, PMLR, 2019.

[64] C.-W. Huang, S. Tan, A. Lacoste, and A. C. Courville, "Improving explorability in variational inference with annealed variational objectives," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[65] K. Katahira, K. Watanabe, and M. Okada, "Deterministic annealing variant of variational bayes method," in *Journal of Physics: Conference Series*, vol. 95, p. 012015, IOP Publishing, 2008.

[66] P. Becker, O. Arenz, and G. Neumann, "Expected information maximization: Using the i-projection for mixture density estimation," in *International Conference on Learning Representations (ICLR)*, 2019.

[67] O. Celik, D. Zhou, G. Li, P. Becker, and G. Neumann, "Specializing versatile skill libraries using local mixture of experts," in *Conference on Robot Learning*, pp. 1423–1433, PMLR, 2022.

[68] Y. Burda, R. B. Grosse, and R. Salakhutdinov, "Importance weighted autoencoders," in *International Conference on Learning Representations (ICLR)*, 2016.

[69] I. O. Tolstikhin, O. Bousquet, S. Gelly, and B. Schölkopf, "Wasserstein auto-encoders," in *6th International Conference on Learning Representations (ICLR)*, 2018.

[70] L. Ambrogioni, U. Güçlü, J. Berezutskaya, E. Borne, Y. Güçlütürk, M. Hinne, E. Maris, and M. Gerven, "Forward amortized inference for likelihood-free variational marginalization," in *The 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 777–786, PMLR, 2019.

[71] Y. Kim, S. Wiseman, A. Miller, D. Sontag, and A. Rush, "Semi-amortized variational autoencoders," in *International Conference on Machine Learning (ICML)*, pp. 2678–2687, PMLR, 2018.

[72] J. Marino, Y. Yue, and S. Mandt, "Iterative amortized inference," in *International Conference on Machine Learning (ICML)*, pp. 3403–3412, PMLR, 2018.

[73] H. Ren and P. Ben-Tzvi, "Learning inverse kinematics and dynamics of a robotic manipulator using generative adversarial networks," *Robotics and Autonomous Systems*, vol. 124, p. 103386, 2020.

[74] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[75] S. Kim and J. Perez, "Learning reachable manifold and inverse mapping for a redundant robot manipulator," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4731–4737, IEEE, 2021.

[76] N. De Cao, W. Aziz, and I. Titov, "Block neural autoregressive flow," in *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 1263–1273, PMLR, 2020.

[77] R. Bensadoun, S. Gur, N. Blau, and L. Wolf, "Neural inverse kinematic," in *Proceedings of the 39th International Conference on Machine Learning (ICML)*, vol. 162 of *Proceedings of Machine Learning Research*, pp. 1787–1797, PMLR, 2022.

[78] C.-K. Ho and C.-T. King, "Automating the learning of inverse kinematics for robotic arms with redundant dofs," *arXiv*, 2022.

[79] D. Blinov, A. Saveliev, and A. Shabanova, "Deep q-learning algorithm for solving inverse kinematics of four-link manipulator," in *Proceedings of 15th International Conference on Electromechanics and Robotics" Zavalishin's Readings"*, pp. 279–291, Springer, 2021.

[80] A. Schmitz and P. Berthet-Rayne, "Using deep-learning proximal policy optimization to solve the inverse kinematics of endoscopic instruments," *IEEE Transactions on Medical Robotics and Bionics*, vol. 3, no. 1, pp. 273–276, 2020.

[81] S. Phaniteja, P. Dewangan, P. Guhan, A. Sarkar, and K. M. Krishna, "A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots," in *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1818–1823, IEEE, 2017.

[82] A. Perrusquía, W. Yu, and X. Li, "Multi-agent reinforcement learning for redundant robot control in task-space," *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 1, pp. 231–241, 2021.

[83] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations (ICLR)*, 2016.

[84] W. Xu, J. Chen, H. Y. Lau, and H. Ren, "Data-driven methods towards learning the highly nonlinear inverse kinematics of tendon-driven surgical manipulators," *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 13, no. 3, p. e1774, 2017.

[85] R. Grassmann, V. Modes, and J. Burgner-Kahrs, "Learning the forward and inverse kinematics of a 6-dof concentric tube continuum robot in SE(3)," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5125–5132, IEEE, 2018.

[86] G. Fang, Y. Tian, Z.-X. Yang, J. M. Geraedts, and C. C. Wang, "Efficient jacobian-based inverse kinematics with sim-to-real transfer of soft robots by learning," *IEEE/ASME Transactions on Mechatronics*, 2022.

[87] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool, "Multi-task learning for dense prediction tasks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[88] S. Wu, H. R. Zhang, and C. Ré, "Understanding and improving information transfer in multi-task learning," in *8th International Conference on Learning Representations (ICLR)*, 2020.

[89] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems (NIPS)*, vol. 30, 2017.

[90] S. Liu, E. Johns, and A. J. Davison, "End-to-end multi-task learning with attention," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR9)*, pp. 1871–1880, 2019.

[91] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[92] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *4th International Conference on Learning Representations (ICLR)*, 2016.

[93] Z. Ye, "Efficient gradient-based variational inference with gmms," Master's thesis, KIT Department of Informatics, 2021.

# A. Encoder Objective Derivation

## A.1. Gaussian Encoder

The Gaussian encoder is similar to the one proposed by [93], however we will use an additional VIPS decomposition over the hidden variable. By inserting the model definition, we get the ELBO

$$L(\mathbf{z}) = \sum_o q(o|\,\mathbf{z}) \int_s q(\mathbf{s}\,|\,\mathbf{z}) \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s}, o)(R(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{x}\,|\,\mathbf{z}))d\,\mathbf{s}\,d\,\mathbf{x}\,.$$

We will use the normal VIPS decomposition over the component responsibilities first. Inserting the log-responsibilities $-\log q(\mathbf{x}\,|\,\mathbf{z}) = -\log q(o|\,\mathbf{z}) - \log q(\mathbf{x}\,|o, \mathbf{z}) + \log q(o|\,\mathbf{x}, \mathbf{z})$ via Bayes' rule:

$$\begin{aligned} L(\mathbf{z}) = &\sum_o q(o|\,\mathbf{z}) \int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s}, o)(R(\mathbf{x}, \mathbf{z}) - \log q(o|\,\mathbf{z}) - \log q(\mathbf{x}\,|o, \mathbf{z}) \\ &+ \log q(o|\,\mathbf{x}, \mathbf{z}))d\,\mathbf{s}\,d\,\mathbf{x} \\ = &\sum_o q(o|\,\mathbf{z}) \int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s}, o)(R(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{x}\,|o, \mathbf{z}) + \log q(o|\,\mathbf{x}, \mathbf{z}))d\,\mathbf{s}\,d\,\mathbf{x} \\ &+ \mathcal{H}(q(o|\,\mathbf{z})). \end{aligned}$$

Inserting the first auxiliary distribution $\tilde{q}(o|\,\mathbf{x},\mathbf{z})$:

$$\begin{aligned}
L(\mathbf{z}) =& \sum_o q(o|\,\mathbf{z}) \int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s},o)(R(\mathbf{x},\mathbf{z}) - \log q(\mathbf{x}\,|o,\mathbf{z}) + \log q(o|\,\mathbf{x},\mathbf{z}) \\
& - \log \tilde{q}(o|\,\mathbf{x},\mathbf{z}) + \log \tilde{q}(o|\,\mathbf{x},\mathbf{z}))d\,\mathbf{s}\,d\,\mathbf{x} + \mathcal{H}(q(o|\,\mathbf{z})) \\
=& \sum_o q(o|\,\mathbf{z}) \int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s},o)(\underbrace{R(\mathbf{x},\mathbf{z}) + \log \tilde{q}(o|\,\mathbf{x},\mathbf{z})}_{=:R(\mathbf{x},o,\mathbf{z})} - \log q(\mathbf{x}\,|o,\mathbf{z}))d\,\mathbf{s}\,d\,\mathbf{x} \\
& + \mathcal{H}(q(o|\,\mathbf{z})) + \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{z})\mathrm{KL}(q(o|\,\mathbf{x},\mathbf{z})||\tilde{q}(o|\,\mathbf{x},\mathbf{z}))d\,\mathbf{x}\,.
\end{aligned}$$

Since $\log q(\mathbf{x}\,|o,\mathbf{z})$ includes encoder and mixture heads, we will now apply a second decomposition in order to separate them.
To replace $q(\mathbf{x}\,|o,\mathbf{z})$ with responsibilities of $\mathbf{s}$, $q(\mathbf{s}\,|\,\mathbf{x},o,\mathbf{z})$, we use Bayes' apply again,

$$\begin{aligned}
q(\mathbf{x}\,|o,\mathbf{z}) =& \frac{q(\mathbf{x},o,\mathbf{z})}{q(o,\mathbf{z})} = \frac{q(\mathbf{x},o,\mathbf{z})q(\mathbf{x},\mathbf{s},o,\mathbf{z})}{q(o,\mathbf{z})q(\mathbf{x},\mathbf{s},o,\mathbf{z})} = \frac{q(\mathbf{x}\,|\,\mathbf{s},o,\mathbf{z})q(\mathbf{s},o,\mathbf{z})}{q(o,\mathbf{z})q(\mathbf{s}\,|\,\mathbf{x},o,\mathbf{z})} \\
=& \frac{q(\mathbf{x}\,|\,\mathbf{s},o,\mathbf{z})q(\mathbf{s}\,|o,\mathbf{z})}{q(\mathbf{s}\,|\,\mathbf{x},o,\mathbf{z})} = \frac{q(\mathbf{x}\,|\,\mathbf{s},o)q(\mathbf{s}\,|\,\mathbf{z})}{q(\mathbf{s}\,|\,\mathbf{x},o,\mathbf{z})}.
\end{aligned}$$

Additionally, we used that $\mathbf{x}$ is independent of $\mathbf{z}$ given $o$ and that $\mathbf{s}$ is independent of o given $\mathbf{z}$. Inserting the responsibilities, we get

$$\begin{aligned}
L(\mathbf{z}) =& \sum_o q(o|\,\mathbf{z}) \int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s},o)(R(\mathbf{x},o,\mathbf{z}) - \log q(\mathbf{x}\,|\,\mathbf{s},o) - \log q(\mathbf{s}\,|\,\mathbf{z}) \\
& + \log q(\mathbf{s}\,|\,\mathbf{x},o,\mathbf{z}))d\,\mathbf{x}\,d\,\mathbf{s} + \mathcal{H}(q(o|\,\mathbf{z})) + \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{z})\mathrm{KL}(q(o|\,\mathbf{x},\mathbf{z})||\tilde{q}(o|\,\mathbf{x},\mathbf{z}))d\,\mathbf{x} \\
=& \sum_o q(o|\,\mathbf{z}) \int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \Bigg[ \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s},o)(R(\mathbf{x},o,\mathbf{z}) + \log q(\mathbf{s}\,|\,\mathbf{x},o,\mathbf{z}))d\,\mathbf{x} \\
& + \mathcal{H}(q(\mathbf{x}\,|\,\mathbf{s},o)) \Bigg] d\,\mathbf{s} + \mathcal{H}(q(o|\,\mathbf{z})) + \mathcal{H}(q(\mathbf{s}\,|\,\mathbf{z})) \\
& + \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{z})\mathrm{KL}(q(o|\,\mathbf{x},\mathbf{z})||\tilde{q}(o|\,\mathbf{x},\mathbf{z}))d\,\mathbf{x}\,.
\end{aligned}$$

Introducing the auxiliary distribution $\tilde{q}(\mathbf{s} \mid \mathbf{x}, o, \mathbf{z})$:

$$
\begin{aligned}
L(\mathbf{z}) = & \sum_o q(o \mid \mathbf{z}) \int_{\mathbf{s}} q(\mathbf{s} \mid \mathbf{z}) \Bigg[ \int_{\mathbf{x}} q(\mathbf{x} \mid \mathbf{s}, o)(R(\mathbf{x}, o, \mathbf{z}) + \log q(\mathbf{s} \mid \mathbf{x}, o, \mathbf{z}) - \log \tilde{q}(\mathbf{s} \mid \mathbf{x}, o, \mathbf{z}) \\
& + \log \tilde{q}(\mathbf{s} \mid \mathbf{x}, o, \mathbf{z})) d\,\mathbf{x} + \mathcal{H}(q(\mathbf{x} \mid \mathbf{s}, o)) \Bigg] d\,\mathbf{s} + \mathcal{H}(q(o \mid \mathbf{z})) + \mathcal{H}(q(\mathbf{s} \mid \mathbf{z})) \\
& + \int_{\mathbf{x}} q(\mathbf{x} \mid \mathbf{z}) \mathrm{KL}(q(o \mid \mathbf{x}, \mathbf{z}) || \tilde{q}(o \mid \mathbf{x}, \mathbf{z})) d\,\mathbf{x} \\
= & \sum_o q(o \mid \mathbf{z}) \int_{\mathbf{s}} q(\mathbf{s} \mid \mathbf{z}) \Bigg[ \int_{\mathbf{x}} q(\mathbf{x} \mid \mathbf{s}, o)(R(\mathbf{x}, o, \mathbf{z}) \\
& + \log \tilde{q}(\mathbf{s} \mid \mathbf{x}, o, \mathbf{z})) d\,\mathbf{x} + \mathcal{H}(q(\mathbf{x} \mid \mathbf{s}, o)) \Bigg] d\,\mathbf{s} + \mathcal{H}(q(o \mid \mathbf{z})) + \mathcal{H}(q(\mathbf{s} \mid \mathbf{z})) \\
& + \int_{\mathbf{x}} q(\mathbf{x} \mid \mathbf{z}) \mathrm{KL}(q(o \mid \mathbf{x}, \mathbf{z}) || \tilde{q}(o \mid \mathbf{x}, \mathbf{z})) d\,\mathbf{x} \\
& + \sum_o q(o \mid \mathbf{z}) \int_{\mathbf{x}} q(\mathbf{x} \mid \mathbf{z}) \mathrm{KL}(q(\mathbf{s} \mid \mathbf{x}, o, \mathbf{z})) || \tilde{q}(\mathbf{s} \mid \mathbf{x}, o, \mathbf{z})) d\,\mathbf{x}\,.
\end{aligned}
$$

In this form, the component objective could be optimized using MOTO [25], and the encoder objective is identical to the previous component objective.

## A.2. Dirac Encoder

Starting from the ELBO,

$$
L(\mathbf{z}) = \int_{\mathbf{s}} q(\mathbf{s} \mid \mathbf{z}) \sum_o q(o \mid \mathbf{s}) \int_{\mathbf{x}} q(\mathbf{x} \mid \mathbf{s}, o)(R(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{x} \mid \mathbf{z})) d\,\mathbf{s}\, d\,\mathbf{x},
$$

we only apply the normal VIPS bound using the log-responsibilities

$$
-\log q(\mathbf{x} \mid \mathbf{z}) = -\log q(o \mid \mathbf{z}) - \log q(\mathbf{x} \mid o, \mathbf{z}) + \log q(o \mid \mathbf{x}, \mathbf{z}),
$$

and insert them via Bayes' rule:

$$
\begin{aligned}
L(\mathbf{z}) = & \int_{\mathbf{s}} q(\mathbf{s} \mid \mathbf{z}) \sum_o q(o \mid \mathbf{s}) \int_{\mathbf{x}} q(\mathbf{x} \mid \mathbf{s}, o)(R(\mathbf{x}, \mathbf{z}) - \log q(o \mid \mathbf{z}) - \log q(\mathbf{x} \mid o, \mathbf{z}) \\
& + \log q(o \mid \mathbf{x}, \mathbf{z})) d\,\mathbf{x}\, d\,\mathbf{s}\,.
\end{aligned}
$$

Next, we insert the auxiliary distribution $\tilde{q}(o|\,\mathbf{x},\mathbf{z})$:

$$\begin{aligned}L(\mathbf{z}) = &\int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \sum_o q(o|\,\mathbf{s}) \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s},o)(R(\mathbf{x},\mathbf{z}) - \log q(o|\,\mathbf{z}) - \log q(\mathbf{x}\,|o,\mathbf{z}) + \log q(o|\,\mathbf{x},\mathbf{z}) \\ &- \log \tilde{q}(o|\,\mathbf{x},\mathbf{z}) + \log \tilde{q}(o|\,\mathbf{x},\mathbf{z}))d\,\mathbf{x}\,d\,\mathbf{s} \\ = &\int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \sum_o q(o|\,\mathbf{s}) \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s},o)(R(\mathbf{x},\mathbf{z}) - \log q(o|\,\mathbf{z}) - \log q(\mathbf{x}\,|o,\mathbf{z}) \\ &+ \log \tilde{q}(o|\,\mathbf{x},\mathbf{z}))d\,\mathbf{x}\,d\,\mathbf{s} + \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{z})\mathrm{KL}(q(o|\,\mathbf{x},\mathbf{z})||\tilde{q}(o|\,\mathbf{x},\mathbf{z}))d\,\mathbf{x},\end{aligned}$$

and add the auxiliary log-responsibilities,

$$\begin{aligned}L(\mathbf{z}) = &\int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \sum_o q(o|\,\mathbf{s}) \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s},o)(R(\mathbf{x},\mathbf{z}) - \log q(o|\,\mathbf{z}) - \log q(\mathbf{x}\,|o,\mathbf{z}) \\ &- \log \tilde{q}(\mathbf{x}\,|\,\mathbf{z}) + \log \tilde{q}(o|\,\mathbf{z}) + \log \tilde{q}(\mathbf{x}\,|o,\mathbf{z}))d\,\mathbf{x}\,d\,\mathbf{s} \\ &+ \int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{z})\mathrm{KL}(q(o|\,\mathbf{x},\mathbf{z})||\tilde{q}(o|\,\mathbf{x},\mathbf{z}))d\,\mathbf{x} \\ \geq &\sum_o q(o|f(\mathbf{z})) \int_{\mathbf{x}} q(\mathbf{x}\,|f(\mathbf{z}),o)(R(\mathbf{x},\mathbf{z}) - \log q(o|\,\mathbf{z}) - \log q(\mathbf{x}\,|o,\mathbf{z}) \\ &- \log \tilde{q}(\mathbf{x}\,|\,\mathbf{z}) + \log \tilde{q}(o|\,\mathbf{z}) + \log \tilde{q}(\mathbf{x}\,|o,\mathbf{z}))d\,\mathbf{x}\,.\end{aligned}$$

Finally, we reformulate the lower bound to

$$\begin{aligned}L'(\mathbf{z}) = &\sum_o q(o|f(\mathbf{z}))\Big[\int_{\mathbf{x}} q(\mathbf{x}\,|f(\mathbf{z}),o)(R(\mathbf{x},\mathbf{z}) - \log \tilde{q}(\mathbf{x}\,|\,\mathbf{z}))d\,\mathbf{x} \\ &- \mathrm{KL}(q(\mathbf{x}\,|o,\mathbf{z})||\tilde{q}(\mathbf{x}\,|o,\mathbf{z}))\Big] - \mathrm{KL}(q(o|\,\mathbf{z})||\tilde{q}(o|\,\mathbf{z})) \\ = &\int_{\mathbf{s}} q(\mathbf{s}\,|\,\mathbf{z}) \sum_o q(o|\,\mathbf{s})\Big[\int_{\mathbf{x}} q(\mathbf{x}\,|\,\mathbf{s},o)(R(\mathbf{x},\mathbf{z}) - \log \tilde{q}(\mathbf{x}\,|\,\mathbf{z}))d\,\mathbf{x} \\ &-\mathrm{KL}(q(\mathbf{x}\,|o,\mathbf{s})||\tilde{q}(\mathbf{x}\,|o,\mathbf{z}))\Big] - \mathrm{KL}(q(o|\,\mathbf{s})||\tilde{q}(o|\,\mathbf{z})),\end{aligned}$$

where the new objective is now written in terms of the KL.