Technische Universität Berlin
Institute of Software Engineering and Theoretical Computer Science
Machine Learning

Structured Image Decompositions using Unsupervised Object-Centric Learning

Master's Thesis

Markus Norden

Examiners:	Prof. Dr. Klaus-Robert Müller, TU Berlin
	Prof. Dr. Manfred Opper, TU Berlin
Supervisor:	Msc. Thanh Binh Bui, TU Berlin

English:
Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.

This paper was not previously presented to another examination board and has not been published.

German:
Eidesstattliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen direkt und indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Ich weiß, dass die Arbeit in digitalisierter Form daraufhin überprüft werden kann, ob unerlaubte Hilfsmittel verwendet wurden und ob es sich - insgesamt oder in Teilen - um ein Plagiat handelt. Zum Vergleich meiner Arbeit mit existierenden Quellen darf sie in eine Datenbank eingestellt werden und nach der Überprüfung zum Vergleich mit künftig eingehenden Arbeiten dort verbleiben. Weitere Vervielfältigungs- und Verwertungsrechte werden dadurch nicht eingeräumt.

Die Arbeit wurde weder einer anderen Prüfungsbehörde vorgelegt noch veröffentlicht.

City, Date

Signature

English:
Abstract

Object-centered perception lays the foundation of human intelligence and may be a stepping stone towards general artificial intelligence. Despite recent advances, artificial vision systems that decompose images into objects and their relations without any supervision lack real-world applicability and only work on simplistic datasets. While there are multiple reasons associated to this obstacle, we hypothesize that integrating a multi-class paradigm into object-structured representations is an essential prerequisite to model the variability of real-world images. In this thesis, we present and test novel generative models that aim to retrieve a multi-class representation from raw images without any supervision by interpreting objects as prototype modifications. For single-object datasets such as MNIST, these approaches can equally be used for image clustering. Despite its simplicity, our best model discovers clusters that match the original MNIST classes with an accuracy of 93.52%. For multi-object datasets, we present a model that directly learns to locate and classify objects in a completely unsupervised manner. The full implementation (based on Pytorch) and the trained networks are available at http://www.github.com/borea17/StrImaDec.

German:
Zusammenfassung

Objektbasierte Wahrnehmung ist die Grundlage menschlicher Intelligenz and könnte ein wichtiger Schritt auf dem Weg zur starken künstlichen Intelligenz sein. Selbstüberwachte künstliche Bildverarbeitungssysteme, die versuchen, Bilder in Objekte und deren Relation zu zerlegen, können trotz jüngster Fortschritte nicht für Bilder aus der realen Welt verwendet werden, sondern nur für grob vereinfachte Datensätze. Obwohl es mehrere Gründe für diese Limitierung gibt, nehmen wir an, dass ein Multi-Klassen Paradigma notwendig ist um die Variabilität innerhalb realer Bilder modellieren zu können. In der vorliegenden Arbeit präsentieren und testen wir neuartige generative Algorithmen, die das Ziel haben eine Multi-Klassen Repräsentation aus unverarbeiteten Bildern zu extrahieren ohne explizit vorgegebene Labels. Diese Algorithmen interpretieren Objekte in Bildern als Prototype-Modifizierungen. Für Datensätze, in denen nur ein Objekt pro Bild vorkommt wie bspw. MNIST, können unsere Algorithmen auch als Bild-Clustering verstanden werden. Trotz der Einfachheit unserer Algorithmen, entdeckt unser bestes Modell Cluster, die den originallen MNIST Klassen entsprechen, so dass eine Genauigkeit von 93.52% erreicht wird. Für Datensätze mit mehreren Objekten pro Bild stellen wir einen Algorithmus vor, der die Position und Klasse der jeweiligen Objekte ohne jegliche Supervision lernt. Die gesamte Implementierung, sowie die trainierten Netze sind unter folgendem Link verfügbar: http://www.github.com/borea17/StrImaDec.

Contents

1. Introduction

Cognitive sciences indicate that humans maintain internal representations about the outer world which allows them to efficiently reason about the consequences of their interactions. Objects seem to be the building blocks of this representation (Spelke & Kinzler, 2007). In particular, visual complex scenes are perceived as a composition of objects (Scholl, 2001). Thus, artificial systems that interpret images in this way are promising with a widespread range of applications across multiple domains such as reinforcement learning (Watters, Matthey, Bosnjak et al., 2019), visual reasoning (Yi et al., 2019) and simulation of physical systems (Battaglia et al., 2016).

However, most approaches to obtain object-centric representation from raw pixels require either supervision (Sun et al., 2019; Watters et al., 2017) or use hand-crafted object detection systems (Battaglia et al., 2016; Kansky et al., 2017). These approaches are impractical due to their limited generalization capabilities (i.e., models cannot adapt), high costs (i.e., labeling the data is expensive) and portability problems (i.e., trained models cannot be easily transferred to new datasets). As a results, new research focuses on obtaining object-centric representation in an unsupervised learning setting by leveraging inductive biases about objects (Eslami et al., 2016; Burgess et al., 2019; Greff et al., 2019; Engelcke et al., 2019; Crawford & Pineau, 2019; Lin et al., 2020). These inductive biases include

- compositional structure of the data, i.e., scenes are composed of multiple objects,
- structurally similar scene elements, i.e., all objects originate from a class.

In this thesis, we build upon this new line of research and extend the inductive biases by including a multi-class paradigm. To the best of our knowledge, all previous work assumes in some way that all objects originate from the same class. Furthermore, a continuous latent space is used to determine the attributes of each object. As a result, the following inherent shortcomings can be identified

- **Inefficient/Incomprehensible use of attributes:** How should the attributes to generate different images of a *hand* relate to the attributes to generate images of *cars*? Clearly, these representations are hardly capable of being integrated into each other.
- **Boundary issues:** Due to the continuous latent space, object attributes are assumed to transition smoothly. Thus, discrete boundaries (e.g., a dataset containing circles and squares) typically lead to gibberish images when the attributes are smoothly transitioned.

To overcome these problems, we propose a novel generative approach in which objects are associated with prototypes and the resulting images are understood as prototype modifications, see figure 1.1. We use a variational autoencoder (Kingma & Welling, 2014) approach to infer the associated class and prototype modification from raw pixel images in an unsupervised manner. The introduced discreteness demands special care which is why part of the thesis focuses on gradient estimators for discrete random variables.

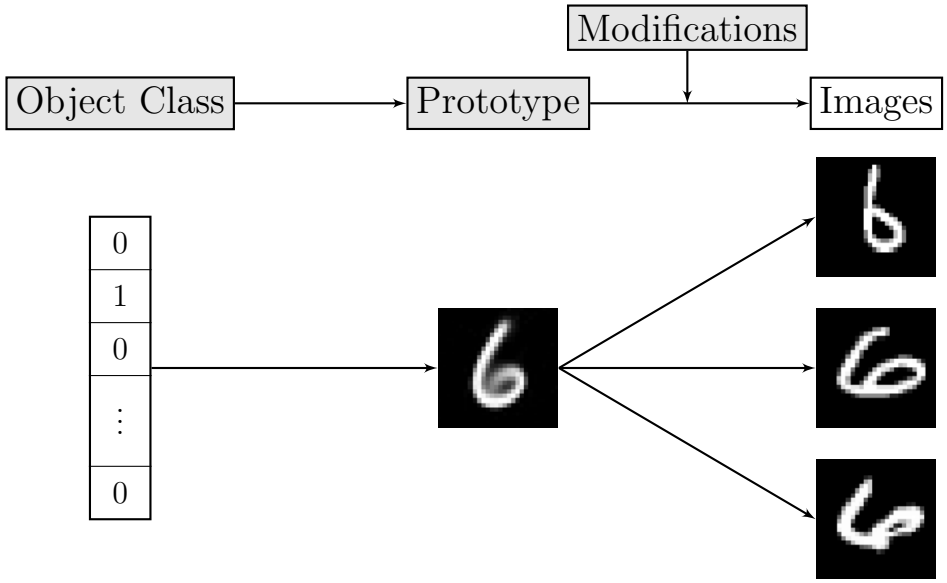


Figure 1.1.: **Objects as Prototype Modifications:** In our generative approach each image is generated by sampling an *object class* and *a modification*. Each object class has an associated *prototype* and the observed *images* are understood as object modifications. Notably, we only observe the *images* and our model learns to infer the *object class*, *prototype* and *modifications* using a variational autoencoder approach in which the inference and generative model parameters are trained in tandem.

Our main contributions are as follows:

- comparison of state-of-the-art gradient estimators for discrete random variables,
- novel generative models to learn multi-class representations (clusters) in single-object datasets with notable clustering results,
- extending existing generative models to include the multi-class paradigm for multi-object datasets.

2. Related Work

In order to put the thesis and its contributions into perspective, the following chapter briefly summarizes related work. Therefore, the first two parts serve as building blocks for the last section on unsupervised object-centric learning on images: Firstly, a short review on gradient estimators for random variables is given. These are needed for optimizing deep generative models. Secondly, two neural network architectures, the variational autoencoder (VAE) and the spatial transformer (ST), are explained in more detail. Lastly, structured image decompositions using unsupervised object-centric learning on images is motivated and two state-of-the-art approaches are presented.

2.1. Gradient Estimators for Random Variables

The ability to perform gradient-based optimization through the well-known back-propagation algorithm (Rumelhart et al., 1986) lays the foundation of deep learning. In essence, this algorithm can be abstracted into two steps:

1. *Forward Path*: Compute the network loss $\mathcal{L}_{\boldsymbol{\theta}}$ by feeding an input $\mathbf{x}$ through the neural network where the parameter vector $\boldsymbol{\theta}$ includes all trainable weights and biases of the network. E.g., the loss could be a measure of the difference between a desired output $\mathbf{y}$ and the actual output of the neural network $\widetilde{\mathbf{y}}_{\boldsymbol{\theta}} = f_{\boldsymbol{\theta}}(\mathbf{x})$ corresponding to the input $\mathbf{x}$.

2. *Backward Path*: Compute the gradients $g_i = \frac{\partial \mathcal{L}_{\boldsymbol{\theta}}}{\partial \theta_i}$ for all parameters $\theta_i \in \boldsymbol{\theta}$ by repeatedly applying the chain rule along the nodes of the computation graph (starting from the loss $\mathcal{L}_{\boldsymbol{\theta}}$) and storing the intermediate results. Thereby, the gradient computation avoids redundancy which makes it significantly more efficient than computing all gradients separately. This step is commonly referred to as *backpropagation*.

Due to its simplicity, efficiency and ease of handling, the back-propagation algorithm has been adopted in nearly every training procedure of modern deep learning architectures[1]. However, backpropagation lacks applicability to architectures with random variables as nodes (Schulman et al., 2015).

This section aims to provide an understanding of how and why gradient estimators for random nodes need to be formulated in order to make backpropagation applicable. Firstly, a simple stochastic optimization problem is presented which is tackled by a neural network approach including one stochastic node. Furthermore, a more detailed explanation of why gradients cannot be computed directly in this setup is given. The two subsequent sections build upon this problem to review gradient estimators for continuous and discrete random variables.

[1] There are some exceptions, e.g., Bayesian networks (Stephenson, 2000) typically use belief propagation to perform inference.

2.1.1. Problem Statement

For the sake of clarity, we consider a simple toy problem: Let $\mathcal{D} = \left\{\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)\right\}_{i=1}^{N}$ be a labeled dataset of N samples $\mathbf{x}^{(i)}$ with corresponding labels $\mathbf{y}^{(i)}$. Furthermore, let there be some stochasticity in the labels such that for each sample $\mathbf{x}^{(i)}$ the label is drawn from a conditional distribution $\mathbf{y}^{(i)} \sim p(\mathbf{y}|\mathbf{x}^{(i)})$. As a result, there may be identical samples with distinct labels in the dataset. We assume that the family of the conditional distribution is known, while its parameters are not. The goal is to estimate these parameters using a neural network.

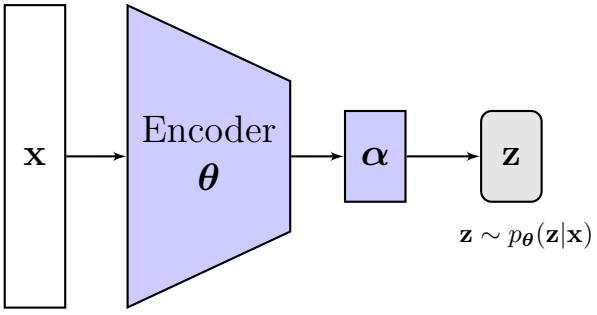
Figure 2.1 depicts how we choose to approach this task by summarizing the network architecture and the corresponding objective function: The architecture consists of a deterministic encoder to obtain the conditional distribution parameters $\boldsymbol{\alpha}$ and a random node $\mathbf{z}$ as the predicted label which is generated by sampling from the parametrised conditional distribution $\mathbf{z} \sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. The distribution parameters $\boldsymbol{\alpha}$ depend on the distribution family at hand, e.g.,

- for an isotropic Gaussian $\boldsymbol{\alpha} = \left(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\right)$ such that $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I}\right)$,
- for a categorical distribution $\boldsymbol{\alpha} = \mathbf{p}$ such that $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \mathrm{Cat}(\mathbf{p})$.

Learning the optimal network parameters $\boldsymbol{\theta}$ can then be posed as a stochastic optimization problem, i.e.,

$$\min_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) = \min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z} \sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[f_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{y})\Big], \tag{2.1}$$

where $\mathcal{L}_{\boldsymbol{\theta}}$ denotes the objective function and $f_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{y})$ is some function to measure the loss between the predicted label $\mathbf{z}$ and the actual label $\mathbf{y}$. While f may be independent of any network parameters $\boldsymbol{\theta}$, e.g., the l_2-norm, we explicitly assume the more general case in which f includes some trainable network parameters $\boldsymbol{\theta}$.



(a) Network Architecture

$$\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{\mathbf{z} \sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[f_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{y})\Big]$$

(b) Objective Function

Figure 2.1.: **Toy Problem:** The network architecture (a) and the corresponding objective function (b) are shown. In this setup, standard backpropagation does not work to obtain the optimal network parameters $\boldsymbol{\theta}$.

Why does standard backpropagation not work?

The intuitive answer is that gradients cannot flow through random nodes due to the associated non-differentiable sampling operation (Jang et al., 2017). More accurately, some form of empirical gradient estimator is needed due to the expectation in the objective function. However, Monte Carlo integration as the standard approach to approximate expectations will not work to obtain a valid gradient estimator. This can be shown by differentiating the objective

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[f_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{y})\Big] = \nabla_{\boldsymbol{\theta}} \int f_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{y}) p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) d\mathbf{z} \tag{2.2}$$

$$= \int \nabla_{\boldsymbol{\theta}} \Big(f_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{y}) p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \Big) d\mathbf{z} \tag{2.3}$$

$$= \underbrace{\int \Big(\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{y})\Big) p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) d\mathbf{z}}_{\mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{y})\Big]} + \int f_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{y}) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) d\mathbf{z}, \tag{2.4}$$

in which the second term is problematic since it does not have the form of an expectation w.r.t. $\mathbf{z}$ and thus does not lead to a Monte Carlo gradient estimator. Note that the first term can be estimated with Monte Carlo integration.

What makes the toy problem different to a typical classification problem?

In fact, the toy problem looks very similar to a classical supervised classification problem which is typically solved without random nodes in the network architecture. So what do we do in classification approaches from a probabilistic point of view? Essentially, we compute the distribution parameters $\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{x})$ and optimize them directly by maximizing the probability a label $\mathbf{y}$ in the predicted distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, i.e.,

$$\max_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}\Big(\mathbf{y}|\boldsymbol{\alpha}\Big) = \max_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}\Big(\mathbf{y}|\boldsymbol{\alpha}\Big). \tag{2.5}$$

E.g., for an isotropic Gaussian with unit variance $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\alpha}, \mathbf{I})$, maximizing the log-likelihood of $\mathbf{y}$ boils down to minimizing the mean squared error between $\boldsymbol{\alpha}$ and $\mathbf{y}$. At the same time, we can interpret $\boldsymbol{\alpha}$ as the predicted label.

The classification approach is not applicable for the toy problem, since we not only fit the predicted distribution, but also need to minimize the function f which depends on samples from the predicted distribution.

2.1.2. Continuous Random Variables

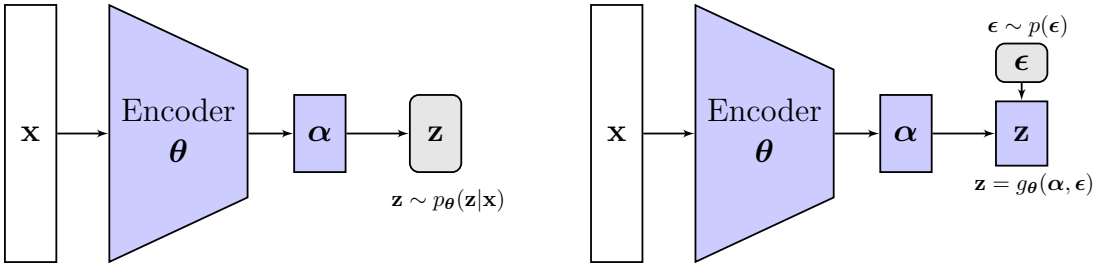
For most continuous random variables, an unbiased, low-variance gradient estimator can be obtained through the **reparameterization trick** which was (independently) introduced by Kingma & Welling (2014) and Rezende et al. (2014). The idea behind this trick is to use samples from an auxiliary noise distribution $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ such that sampling $\mathbf{z} \sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ can be written as a deterministic mapping $\mathbf{z} = g_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon})$. As a result, the gradient operator can

be moved inside the expectation of the objective and a Monte Carlo gradient estimator is applicable, i.e.,

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[f_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{y})\Big] = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\epsilon}\sim p(\boldsymbol{\epsilon})}\Big[f_{\boldsymbol{\theta}}\big(g_{\boldsymbol{\theta}}(\mathbf{x},\boldsymbol{\epsilon}),\mathbf{y}\big)\Big] = \mathbb{E}_{\boldsymbol{\epsilon}\sim p(\boldsymbol{\epsilon})}\Big[\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}\big(g_{\boldsymbol{\theta}}(\mathbf{x},\boldsymbol{\epsilon}),\mathbf{y}\big)\Big] \tag{2.6}$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}\left(\underbrace{g_{\boldsymbol{\theta}}\left(\mathbf{x},\boldsymbol{\epsilon}^{(i)}\right)}_{\mathbf{z}^{(i)}},\mathbf{y}\right) \quad \text{with} \quad \boldsymbol{\epsilon}^{(i)} \sim p(\boldsymbol{\epsilon}). \tag{2.7}$$

In fact, the reparameterization trick is a small modification of the network architecture which transforms the random node $\mathbf{z}$ to a deterministic node by introducing an auxiliary random variable $\boldsymbol{\epsilon}$ and some vector-valued function $g_{\boldsymbol{\theta}}(\cdot)$, see figure 2.2.



(a) Original Network Architecture (b) Reparametrization Trick Architecture

Figure 2.2.: **Illustration of the Reparameterization Trick for the Toy Problem**: The original network architecture (a) (see section 2.1.1) is slightly modified through the reparameterization trick into (b). Random nodes are gray with round edges, while deterministic layers are hard-edged blue.

Reparameterization Trick for an Isotropic Gaussian

A multivariate Gaussian with diagonal covariance structure is one of the most common use cases of continuous random variables. Referring to the toy problem, we have $\boldsymbol{\alpha} = (\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ and $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I})$. In this case, samples from a standard normal distribution $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ can be transformed into the desired distribution using $g_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ ($\odot$ denotes element-wise multiplication).

2.1.3. Discrete Random Variables

Due to its widespread range of applications such as reinforcement learning with discrete actions (Yue et al., 2020), finite mixture models (Bouguila & ElGuebaly, 2009) and hard attention mechanisms (V. Mnih et al., 2014), gradient estimators of discrete random variables have been studied immensely in the last decade. In contrast to continuous random variables, any reparameterization of discrete random variables includes discontinuous operations (e.g., arg max) resulting in undefined gradients. Thus, the reparameterization trick is not applicable to discrete random nodes.

In contrast to continuous random variables, there is only a finite number of probabilities such that an exact solution for the gradient can be computed without any sampling operations, i.e.,

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \left[f_{\boldsymbol{\theta}} \left(\mathbf{z}, \mathbf{y} \right) \right] = \nabla_{\boldsymbol{\theta}} \sum_i p_{\boldsymbol{\theta}}(\mathbf{z}^{(i)}|\mathbf{x}) \cdot f_{\boldsymbol{\theta}} \left(\mathbf{z}^{(i)}, \mathbf{y} \right). \tag{2.8}$$

However, the exact gradients may become intractable when the number of discrete states gets too large such as in ancestral sampling schemes.

Therefore, this section reviews alternative methods to obtain a gradient estimator in their chronological order of appearance. In equation 2.4, we derived that

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \Big[f_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{y}) \Big] = \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \Big[\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{y}) \Big] + \int f_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{y}) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) d\mathbf{z}, \tag{2.9}$$

from which only the second term is problematic. Therefore, we remove the functional dependence of $\boldsymbol{\theta}$ from f in this section. Then, the gradient computation simplifies into

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \Big[f(\mathbf{z}, \mathbf{y}) \Big] = \int f(\mathbf{z}, \mathbf{y}) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) d\mathbf{z}. \tag{2.10}$$

Note that for discrete random variables the integral becomes a sum. For convenience and since the following gradient estimators could in principle also be used for continuous random variables, we keep the integral notation.

REINFORCE

REINFORCE (Williams, 1992), also known as *score-function estimator*, is an unbiased gradient estimator that uses the log-derivative trick to move the gradient operator inside the expectation such that Monte-Carlo integration becomes possible, i.e.,

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \Big[f(\mathbf{z}, \mathbf{y}) \Big] &= \int f(\mathbf{z}, \mathbf{y}) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) d\mathbf{z} && (2.11) \\
&= \int f(\mathbf{z}, \mathbf{y}) \Big(\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \Big) p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) d\mathbf{z} \quad \text{(log derivative trick)} && (2.12) \\
&= \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \Big[f(\mathbf{z}, \mathbf{y}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \Big] && (2.13) \\
&\approx \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{z}^{(i)}, \mathbf{y}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(i)}|\mathbf{x}) \quad \text{with} \quad \mathbf{z}^{(i)} \sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}). && (2.14)
\end{aligned}$$

The log-derivative trick comes from the application of the chain rule on the log-likelihood:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) &= \frac{\partial h}{\partial g} \frac{\partial g}{\partial \boldsymbol{\theta}} \quad \text{with} \quad h(g) = \log g \quad \text{and} \quad g(\boldsymbol{\theta}) = p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) && (2.15) \\
&= \frac{1}{g} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \frac{1}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}), && (2.16)
\end{aligned}$$

multiplying by $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ leads to the identity $\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$.

This estimator is generally applicable as long as $\log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is differentiable w.r.t. $\boldsymbol{\theta}$. Notably, it does not require any knowledge about f (except for its evaluation at the given points) nor does it set any limitations on the properties of f, e.g., f does not need to be continuous or differentiable. Although this might be beneficial, it attributes high variance to the empirical gradient estimator as the gradient estimator does not use any information about how f depends on $\mathbf{z}$. In summary, it only uses the final outcome $f(\mathbf{z}, \mathbf{y})$ as a scalar to determine the step size along the log-likelihood gradient $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}^{(i)}|\mathbf{x})$.

Due to its high variance, this gradient estimator is impractical in practice, see e.g., Paisley et al. (2012), and requires some form of variance reduction to be applicable.

REINFORCE with Control Variates

A widely used variation reduction technique for the REINFORCE gradient estimator is the introduction of control variates, also known as *baselines*. Although there are numerous proposals on how to design these control variates in different contexts, e.g., Paisley et al. (2012), A. Mnih & Gregor (2014), Titsias & Lázaro-Gredilla (2015), the underlying concept is always the same: Modify the function in the expectation with some control variate such that its expectation remains the same, but its variance decreases. Mathematically, this translates into

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[f(\mathbf{z}, \mathbf{y})\right] = \nabla_{\boldsymbol{\theta}} \left(\mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[f(\mathbf{z}, \mathbf{y}) - c(\mathbf{z}) + c(\mathbf{z})\Big]\right) \tag{2.17}$$

$$= \underbrace{\nabla_{\boldsymbol{\theta}} \left(\mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[f(\mathbf{z}, \mathbf{y}) - c(\mathbf{z})\Big]\right)}_{\mathcal{CVE}_{\boldsymbol{\theta}}} + \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[c(\mathbf{z})\Big], \tag{2.18}$$

where $c(\mathbf{z})$ denotes the control variate (some function dependent on $\mathbf{z}$) and in which the first term will be approximated with REINFORCE, i.e.,

$$\mathcal{CVE}_{\boldsymbol{\theta}} = \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[\underbrace{f(\mathbf{z}, \mathbf{y})\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}_{\mathbf{A}}\Big] - \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[\underbrace{c(\mathbf{z})\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}_{\mathbf{B}}\Big], \tag{2.19}$$

and the second term accounts for the introduced bias. Clearly, the expectation remains the same, but what about the variance? The control variate $c(\mathbf{z})$ is typically designed such that its expectation can be computed analytically. Therefore, we only need to look at the variance of $\mathcal{CVE}_{\boldsymbol{\theta}}$ which is given by

$$\mathrm{Var}\Big[\mathcal{CVE}_{\boldsymbol{\theta}}\Big] = \mathrm{Var}\Big[\mathbf{A}\Big] + \mathrm{Var}\Big[\mathbf{B}\Big] - 2 \cdot \mathrm{Cov}\Big[\mathbf{A}, \mathbf{B}\Big]. \tag{2.20}$$

Thus, if $\mathbf{A}$ and $\mathbf{B}$ are positively correlated and the variance of $\mathbf{B}$ is smaller than twice the aforementioned covariance, the introduced control variate decreases the variance.

NVIL Gradient Estimator: The Neural Variational Inference and Learning (NVIL) estimator by A. Mnih & Gregor (2014) is a simple, yet powerful approach to introduce a control variate for discrete latent models such as in the toy problem described in section 2.1.1 with the discrete distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. In essence, it uses a neural network $\boldsymbol{\phi}$ as an observation-dependent neural baseline $C_{\boldsymbol{\phi}}(\mathbf{x})$ by regressing it on the non-differentiable function f using the l_2-norm, i.e.,

$$\min_{\boldsymbol{\phi}} \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[\big(f(\mathbf{z}, \mathbf{y}) - C_{\boldsymbol{\phi}}(\mathbf{x})\big)^2\Big]. \tag{2.21}$$

The baseline is independent of $\mathbf{z}$, thus the second term in equation 2.18 vanishes. While this approach ensures that $C_{\boldsymbol{\phi}}$ and f are positively correlated, it does not result in maximal variance reduction. Furthermore, there are two networks trained in tandem which can cause a lack of stability. To address this issue, A. Mnih & Gregor (2014) propose to use some kind of variance normalization and a five times smaller learning rate for the neural baseline. However, in follow-up work by Eslami et al. (2016), the learning rate for the baseline network $\boldsymbol{\phi}$ is an order of magnitude higher than for other network parameters $\boldsymbol{\theta}$. Accordingly, the success of NVIL may be highly dependent on the learning schedule for both networks.

Continuous Relaxation of Discrete Random Variables

Motivated by the reparameterization trick and its ability to retrieve low-variance gradient estimators (see section 2.1.2), Maddison et al. (2017) and Jang et al. (2017) (independently) proposed to approximate sampling from a discrete distribution by sampling from a **con**tinuous relaxation of dis**crete** random variables, termed **concrete distribution** by Maddison et al. (2017).

The concrete distribution builds upon the Gumbel-Max trick (Luce, 1959) which essentially describes the reparametrization trick (c.f. section 2.1.2) for a categorical distribution using the standard Gumbel distribution as auxiliary noise $\boldsymbol{\epsilon} \sim \text{Gumbel}(\mathbf{0}, \mathbf{1})$, i.e., sampling $\mathbf{z} \sim \text{Cat}(\mathbf{p})$ is reparametrized by

$$\mathbf{z} = g(\mathbf{p}, \boldsymbol{\epsilon}) = \text{one\_hot}\Big(\arg\max \big(\log \mathbf{p} + \boldsymbol{\epsilon} \big) \Big). \tag{2.22}$$

The *argmax*-operation is discontinuous and not defined at the boundary of state changes, thus it is not suitable for standard backpropagation. To circumvent this problem, Maddison et al. (2017) and Jang et al. (2017) propose to use the tempered *softmax*-operation as a continuous relaxation of *argmax* such that

$$\widetilde{\mathrm{z}}_i = \widetilde{g}_\lambda(\mathbf{p}, \boldsymbol{\epsilon}) = \frac{\exp\left(\frac{\log \mathrm{p}_i + \epsilon_i}{\lambda}\right)}{\sum_j \exp\left(\frac{\log \mathrm{p}_j + \epsilon_j}{\lambda}\right)}, \tag{2.23}$$

where $\lambda \in [0, \infty[$ denotes the temperature that controls the sharpness of the softmax, i.e., how much of the winner-takes-all dynamics is taken:

- $\lambda \to 0$: *Tempered softmax* smoothly approaches *argmax*,
- $\lambda \to \infty$: *Tempered softmax* leads to a uniform distribution.

Jang et al. (2017) term this reparametrization the **Gumbel-Softmax trick** since the main idea consists of replacing *argmax* by *softmax* in the Gumbel-Max trick. Note that $\widetilde{\mathbf{z}}$ follows a concrete distribution, while $\mathbf{z}$ comes from a discrete distribution.

The gradient estimator obtained from the concrete distribution,

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z} \sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \big[f(\mathbf{z}, \mathbf{y}) \big] \approx \nabla_{\boldsymbol{\theta}}\, \mathbb{E}_{\widetilde{\mathbf{z}} \sim p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}|\mathbf{x})} \left[f(\widetilde{\mathbf{z}}, \mathbf{y}) \right] = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})} \left[f\Big(\underbrace{\widetilde{g}_{\boldsymbol{\theta},\lambda}(\mathbf{x}, \boldsymbol{\epsilon})}_{\widetilde{\mathbf{z}}}, \mathbf{y} \Big) \right], \tag{2.24}$$

is clearly biased. Furthermore, decreasing the bias by setting $\lambda \to 0$ results in higher variances. Thus, the bias-variance tradeoff can be controled via the temperature. The success of this estimator is highly dependent of λ.

REBAR

Tucker et al. (2017) combine the unbiased, high-variance REINFORCE estimator with the low-variance, but biased concrete distribution estimator as a control variate in order to obtain an unbiased, low-variance gradient estimator for discrete random variables. To this end, they introduce the REINFORCE gradient estimator for the relaxed model,

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\widetilde{\mathbf{z}} \sim p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}|\mathbf{x})} \left[f(\widetilde{\mathbf{z}}, \mathbf{y}) \right] = \mathbb{E}_{\widetilde{\mathbf{z}} \sim p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}|\mathbf{x})} \left[f(\widetilde{\mathbf{z}}, \mathbf{y}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}|\mathbf{x}) \right] = (a), \tag{2.25}$$

where $\widetilde{\mathbf{z}} \sim p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}|\mathbf{x}) = \text{Concrete}(\mathbf{p}, \lambda)$. Then, they derive and perform a conditional marginalization over $\widetilde{\mathbf{z}}$ given the non-relaxed samples $\mathbf{z} \sim \text{Cat}(\mathbf{p})$ resulting in

$$(a) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \left[\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\widetilde{\mathbf{s}}|\mathbf{z})} \left[f(\widetilde{\mathbf{s}}, \mathbf{y}) \right] \right] + \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \left[\mathbb{E}_{p_{\boldsymbol{\theta}}(\widetilde{\mathbf{s}}|\mathbf{z})} \left[f(\widetilde{\mathbf{s}}, \mathbf{y}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \right] \right]. \tag{2.26}$$

An low-variance estimate for the first term in equation 2.26 can be obtained by using an adapted reparameterization trick for concrete variables conditioned on a discrete distribution, i.e., $\widetilde{\mathbf{s}} = g_{\boldsymbol{\theta},\lambda}(\mathbf{v}, \mathbf{z}) \sim p_{\boldsymbol{\theta}}(\widetilde{\mathbf{s}}|\mathbf{z})$ with the auxiliary noise $\mathbf{v} \sim \text{Uniform}(\mathbf{0}, \mathbf{1})$, see Appendix C of Tucker et al. (2017). The second term in equation 2.26 is advantageous since it includes the log-likelihood gradient $\nabla_\theta \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ and thus can be directly included as a REINFORCE control variate. As a result, they combine REINFORCE and the concrete gradient estimator as follows to obtain a new gradient estimator for discrete random variables

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \left[f(\mathbf{z}, \mathbf{y}) \right] = \underbrace{\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \left[f(\mathbf{z}, \mathbf{y}) \right]}_{\text{REINFORCE}} + (\eta - \eta) \underbrace{\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}|\mathbf{x})} \left[f(\widetilde{\mathbf{z}}, \mathbf{y}) \right]}_{\text{Concrete Estimator}} \tag{2.27}$$

$$\begin{aligned} &= \eta \nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}|\mathbf{x})} \left[f(\widetilde{\mathbf{z}}, \mathbf{y}) \right] - \eta \, \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \left[\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\widetilde{\mathbf{s}}|\mathbf{z})} \left[f(\widetilde{\mathbf{s}}, \mathbf{y}) \right] \right] \\ &\quad + \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})} \left[\left(f(\mathbf{z}, \mathbf{y}) - \eta \, \mathbb{E}_{p_{\boldsymbol{\theta}}(\widetilde{\mathbf{s}}|\mathbf{z})} \left[f(\widetilde{\mathbf{s}}, \mathbf{y}) \right] \right) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \right], \end{aligned} \tag{2.28}$$

where η is a scaling on the control variate. They term the single-sample Monte Carlo estimator of this expectation the **REBAR**[2] estimator.

Clearly, REBAR is unbiased irrespective of the temperature λ for the relaxed model and the scaling η. Thus, instead of fixing these parameters a priori, Tucker et al. (2017) propose to estimate them online by minimizing the variance of the REBAR estimator. While they provide a good explanation on how they optimize the temperature λ, they do not elaborate on how exactly η is estimated. For the temperature λ, they denote the REBAR estimator by $r(\lambda)$ such that

$$\min_{\lambda} \text{Var} \left[r(\lambda) \right] = \min_{\lambda} \left(\mathbb{E} \left[r(\lambda)^2 \right] - \left(\underbrace{\mathbb{E} \left[r(\lambda) \right]}_{\text{independent of } \lambda} \right)^2 \right) = \min_{\lambda} \mathbb{E} \left[r(\lambda)^2 \right]. \tag{2.29}$$

[2]The term **REBAR** is borrowed from steel terminology in which REBAR is an abbreviation for **re***inforcing* **bar** that is used, among other things, for *reinforced concrete*.

This relieves the burden of treating the temperature λ as a hyperparameter that needs to be tuned carefully. However, REBAR comes at a prize of a more involved implementation and higher computation times, i.e., as noted by Tucker et al. (2017) REBAR requires twice as much computations per step as NVIL and Concrete.

RELAX

Essentially, Grathwohl et al. (2018) combine the ideas of REBAR and NVIL by using a surrogate neural network $\boldsymbol{\phi}$ to construct a control variate $C_{\boldsymbol{\phi}}$ that aims at minimizing the variance. Similar to REBAR, the control variate is evaluated both at $\widetilde{\mathbf{z}}$, the relaxed input, and at $\widetilde{\mathbf{s}}$, the relaxed input conditioned on the discrete variable, such that the objective gradient looks as follows

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[f(\mathbf{z},\mathbf{y})\Big] = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\Big[\Big(f(\mathbf{z},\mathbf{y}) - \mathbb{E}_{p_{\boldsymbol{\theta}}(\widetilde{\mathbf{s}}|\mathbf{z})}\Big[C_{\boldsymbol{\phi}}(\widetilde{\mathbf{s}},\mathbf{y})\Big]\Big)\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\Big] \\ + \nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}|\mathbf{x})}\Big[C_{\boldsymbol{\phi}}(\widetilde{\mathbf{z}})\Big] - \nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\widetilde{\mathbf{s}}|\mathbf{z})}\Big[C_{\boldsymbol{\phi}}(\widetilde{\mathbf{s}})\Big].\end{aligned} \tag{2.30}$$

Grathwohl et al. (2018) term the single-sample Monte-Carlo estimator of this expectation the **RELAX** estimator and prove that this estimator is unbiased for any $C_{\boldsymbol{\phi}}$. The network parameters $\boldsymbol{\phi}$ are jointly trained with the original parameters $\boldsymbol{\theta}$ by minimizing the variance:

$$\min_{\boldsymbol{\phi}} \operatorname{Var}\Big[r_{\boldsymbol{\phi},\boldsymbol{\theta}}\Big] = \min_{\boldsymbol{\phi}} \mathbb{E}\Big[r^2_{\boldsymbol{\phi},\boldsymbol{\theta}}\Big], \tag{2.31}$$

where $r_{\boldsymbol{\phi},\boldsymbol{\theta}}$ denotes the RELAX estimator.

In case that $C_{\boldsymbol{\phi}}(\widetilde{\mathbf{z}},\mathbf{y}) = f(\widetilde{\mathbf{z}}(\lambda),\mathbf{y})$ and $C_{\boldsymbol{\phi}}(\widetilde{\mathbf{s}},\mathbf{y}) = f(\widetilde{\mathbf{s}}(\lambda),\mathbf{y})$, the RELAX estimator becomes the REBAR estimator with fixed scaling of $\eta = 1$. Thus, RELAX can be seen as a generalization of REBAR.

2.2. Neural Network Architectures

The variational autoencoder (VAE) and the spatial transformer (ST) module are two powerful neural network architectures on which the forthcoming sections and chapters will build on. A VAE is not only an architecture, but also a deep generative model that aims to decompose and reconstruct input data in an unsupervised setting. Accordingly, a VAE forms an essential building block of this thesis and in fact, all methods described in section 2.3.2 can be abstracted into VAEs.

2.2.1. Variational Autoencoder (VAE)

The variational autoencoder (VAE) was originally introduced by Kingma & Welling (2014) to showcase how their Auto-Encoding Variational Bayes (AEVB) algorithm can be used in practice. In summary, this algorithm assumes a generative process with parameters $\boldsymbol{\theta}$, introduces a variational approximation with parameters $\boldsymbol{\phi}$ and optimizes the model parameters jointly by maximizing the evidence lower bound. A VAE uses a neural network as an approximation for the recognition and generative model, respectively. As a result, a VAE can be understood as a probabilistic autoencoder which uses variational inference to regularize the coding space.

Latent Variable Model and Variational Approximation

Let $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ denote a dataset of N i.i.d. samples $\mathbf{x}^{(i)}$ and let $\mathbf{z}^{(i)}$ denote the associated unobserved latent variable. Kingma & Welling (2014) assume that each sample $\mathbf{x}^{(i)}$ is obtained by first sampling a latent vector $\mathbf{z}^{(i)}$ from a prior distribution $p_{\boldsymbol{\theta}}(\mathbf{z})$ (where $\boldsymbol{\theta}$ denote the distribution parameters) and then sampling $\mathbf{x}^{(i)}$ itself from the scene model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. Then, they introduce an auxiliary distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ (with its own distribution parameters $\boldsymbol{\phi}$) as an approximation to the true, but unknown posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. The two associated graphical models are depicted in figure 2.3.

a) Generative Process $p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ **b) Variational Approximation** $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$

Figure 2.3.: The directed graphical models represent the assumed generative process (a) and the variational approximation of the intractable posterior (b) in the AEVB algorithm.

Variational Inference

Given the assumed generative process, the data likelihood of a sample $\mathbf{x}^{(i)}$ is given by

$$p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) = \mathbb{E}_{\mathbf{z}\sim p_{\boldsymbol{\theta}}(\mathbf{z})}\left[p_{\boldsymbol{\theta}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)\right] = \int p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)d\mathbf{z} = \int p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)},\mathbf{z}\right)d\mathbf{z}.$$

We can change the sampling distribution $p_{\boldsymbol{\theta}}(\mathbf{z})$ into the approximated posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})$ via the method of importance sampling, i.e.,

$$p_{\boldsymbol{\theta},\boldsymbol{\phi}}\left(\mathbf{x}^{(i)}\right) = \int \frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)},\mathbf{z}\right)}{q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)} q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)d\mathbf{z} = \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)},\mathbf{z}\right)}{q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)}\right]. \tag{2.32}$$

Lastly, we apply Jensen's Inequality on the marginal log-likelihood to arrive at the **evidence lower bound (ELBO)**:

$$\log p_{\boldsymbol{\theta},\boldsymbol{\phi}}\left(\mathbf{x}^{(i)}\right) = \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)},\mathbf{z}\right)}{q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)}\right] \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log \frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)},\mathbf{z}\right)}{q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)}\right] = \mathcal{L}^{\text{ELBO}}_{\boldsymbol{\theta},\boldsymbol{\phi}}. \tag{2.33}$$

Variational inference maximizes this quantity, thereby optimizing a lower-bound on the data log-likelihood[3] $\log p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X})$. At the same time, the marginal log-likelihood of a sample $\mathbf{x}^{(i)}$ can

[3] Note that the marginal likelihood of the dataset $\mathbf{X}$ is composed of a sum over the marginal log-likelihoods of individual datapoints, i.e., $\log p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{X}) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^{(i)})$.

also be rewritten as

$$\log p_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^{(i)}) = \underbrace{D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)||p_{\boldsymbol{\theta}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)\right)}_{\geq 0} + \mathcal{L}^{\mathrm{ELBO}}_{\boldsymbol{\theta},\boldsymbol{\phi}}\left(\mathbf{x}^{(i)}\right), \tag{2.34}$$

where the first term measures the KL divergence between the approximate and the true posterior. Thus, maximizing the ELBO can also be understood as minimizing this KL-divergence, i.e., as finding a distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})$ that is close to the true, but unknown posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})$.

Since the ELBO in this form is less intuitive, it is commonly rewritten as follows

$$\mathcal{L}^{\mathrm{ELBO}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left(-\log q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right) + \log p_{\boldsymbol{\theta}}(\mathbf{z}) + \log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}|\mathbf{z}\right)\right) \tag{2.35}$$

$$= \underbrace{-D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)||p_{\boldsymbol{\theta}}(\mathbf{z})\right)}_{\text{Regularization Term}} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}|\mathbf{z}\right)\right]}_{\text{Reconstruction Accuracy}}, \tag{2.36}$$

where the two terms can be interpreted as the regularization term, i.e., how much differs the encoder distribution $q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)$ from the assumed prior distribution $p_{\boldsymbol{\theta}}(\mathbf{z})$, and the reconstruction accuracy, i.e., how well fits the observed sample $\mathbf{x}^{(i)}$ into the predicted distribution $p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}|\mathbf{z}\right)$.

VAE: Network Architecture and Training Procedure

A VAE simply uses neural networks as function approximators to parametrize the encoder $q_{\boldsymbol{\phi}}$ and decoder $p_{\boldsymbol{\theta}}$ distributions. The VAE architecture is depicted in figure 2.4, it consists of

- a deterministic encoder $\boldsymbol{\phi}$ that encodes the conditional distribution parameters $\boldsymbol{\alpha}$, e.g., $\boldsymbol{\alpha} = \left(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\right)$ such that $q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I})$,
- a random node $\mathbf{z} \sim q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)$, and
- a deterministic decoder that takes the sampled latent code $\mathbf{z}^{(i)}$ to obtain the distribution parameters of the scene model $p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}|\mathbf{z}\right)$.

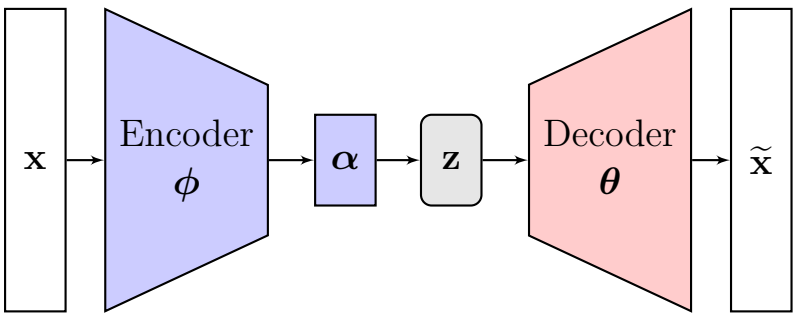


Figure 2.4.: **VAE Network Architecture:** A VAE consists of deterministic encoder $\boldsymbol{\phi}$ and decoder $\boldsymbol{\theta}$ with a sampled latent code $\mathbf{z}$ in between. $\boldsymbol{\alpha}$ denote the distribution parameters of the encoder distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

Unfortunately, we cannot apply standard gradient ascent methods to the objective function,

$$\mathcal{L}^{\text{ELBO}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}^{(i)}) = \underbrace{-D_{\text{KL}}\left(q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)||p_{\boldsymbol{\theta}}(\mathbf{z})\right)}_{\text{analytically solved}} + \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}|\mathbf{z}\right)\right]}_{\text{requires sampling}}, \tag{2.37}$$

due to the fact that the second term requires sampling from the encoder distribution through which we cannot backpropagate, see section 2.1. Note that the first term is typically analytically solved, e.g., the KL-Divergence between two Gaussians can be computed in terms of the associated means and variances. To circumvent this problem, Kingma & Welling (2014) use the reparametrization trick (c.f. section 2.1.2) and then approximate the expectation using Monte Carlo integration, leading to the following empirical VAE objective

$$\begin{aligned}\mathcal{L}^{\text{VAE}}_{k}\left(\mathbf{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi}\right) = \frac{1}{k}\sum_{l=1}^{k}\log p_{\boldsymbol{\theta}}\Big(\mathbf{x}|\underbrace{g_{\boldsymbol{\phi}}\left(\boldsymbol{\epsilon}^{(l)},\mathbf{x}^{(i)}\right)}_{\mathbf{z}^{(l)}}\Big) - D_{\text{KL}}\left(q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)||p_{\boldsymbol{\theta}}(\mathbf{z})\right), \\ \text{with } \boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon})\end{aligned} \tag{2.38}$$

where $p(\boldsymbol{\epsilon})$ denotes the auxiliary noise distribution, $g_{\boldsymbol{\phi}}$ describes the deterministic function from the reparametrization and k denotes the number of samples used for the Monte Carlo estimator. As a result, a VAE can be trained end-to-end.

2.2.2. Spatial Transformer (ST)

The **spatial transformer** (ST) module empowers standard neural networks to actively spatially transform input data or feature maps. It was introduced by Jaderberg et al. (2015) with the aim to perform these transformations in a reasonable and efficient way. Notably, the ST can be integrated as a plug-in module into existing network architectures without any modification to the training procedure. In essence, the ST can be understood as a trainable module that applies some spatial transformation (e.g., crop, scale, rotate) to a given input conditioned on the particular input during a single forward path. To this end, the ST is divided into three consecutive parts

- **Localisation network**: Its purpose is to retrieve the parameters $\boldsymbol{\theta}$ of the spatial transformation $\mathcal{T}_{\boldsymbol{\theta}}$ taking the current feature map $\mathbf{U}$ as input, i.e., $\boldsymbol{\theta} = f_{\text{loc}}(\mathbf{U})$. E.g., for an affine transformation the parameter vector $\boldsymbol{\theta}$ reduces to a 6-dimensional vector describing a simple transformation matrix

$$\mathcal{T}_{\boldsymbol{\theta}} = \begin{bmatrix}\theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6\end{bmatrix}. \tag{2.39}$$

- **Grid generator**: Its purpose is to create the new sampling grid $\widetilde{\mathbf{G}} = \left\{\begin{bmatrix}x_i^s & y_i^s\end{bmatrix}^{\text{T}}\right\}$ defined on the input feature map $\mathbf{U}$ by applying the parametrised transformation $\mathcal{T}_{\boldsymbol{\theta}}$ on the regular grid $\mathbf{G} = \left\{\begin{bmatrix}x_i^t & y_i^t\end{bmatrix}^{\text{T}}\right\}$ defined on the output feature map $\mathbf{V}$. E.g., for

an affine transformation we simply multiply each position of the regular grid $\mathbf{G}$ by the transformation matrix $\mathcal{T}_{\boldsymbol{\theta}}$ to obtain the new sampling grid $\widetilde{\mathbf{G}}$

$$\forall_{i=1}^{H^t \cdot W^t} : \begin{bmatrix} x_i^s \\ y_i^s \end{bmatrix} = \mathcal{T}_{\boldsymbol{\theta}} \begin{bmatrix} x_i^t \\ y_i^t \\ 1 \end{bmatrix} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \end{bmatrix} \begin{bmatrix} x_i^t \\ y_i^t \\ 1 \end{bmatrix}, \tag{2.40}$$

where H^t and W^t denote the height and width of the regular grid (which can be chosen freely by the user).

- **Sampler**: Its purpose is to compute the warped version of the input feature map $\mathbf{U}$ by estimating the pixel values in the new sampling grid $\widetilde{\mathbf{G}}$ obtained from the grid generator. The sampler can be understood as some kind of interpolation mechanism to account for the fact that the new sampling grid does not necessarily align with the input feature map grid. Jaderberg et al. (2015) formulate this interpolation through the application of a sampling kernel centered at a particular location in the input feature map, i.e.,

$$V_i^c = \sum_{n=1}^{H^s} \sum_{m=1}^{W^s} U_{n,m}^c \cdot k_{\boldsymbol{\Phi}_x}(x_i^s - x_m^t) \cdot k_{\boldsymbol{\Phi}_y}(y_i^s - y_n^t), \tag{2.41}$$

where V_i^c denotes the new pixel value of the c-th channel at the i-th position of the new sampling grid coordinates $\begin{bmatrix} x_i^s & y_i^s \end{bmatrix}^{\mathrm{T}}$ and $\boldsymbol{\Phi}_x, \boldsymbol{\Phi}_y$ are the parameters of the sampling kernel $k(\cdot)$. As a result, we obtain a very generic interpolation scheme in which all pixel values from the input feature map $\mathbf{U}$ are included in the computation of one pixel at the output feature map. The sampling kernel k determines the type of interpolation as it acts as a scalar for each input location. Note that in practice the sum reduces to the close neighborhood of i as common kernels (e.g., bilinear or integer) are zero outside of this neighborhood.

Figure 2.5 summarizes the ST architecture as a plug-in module and shows how the individual parts interact with each other.

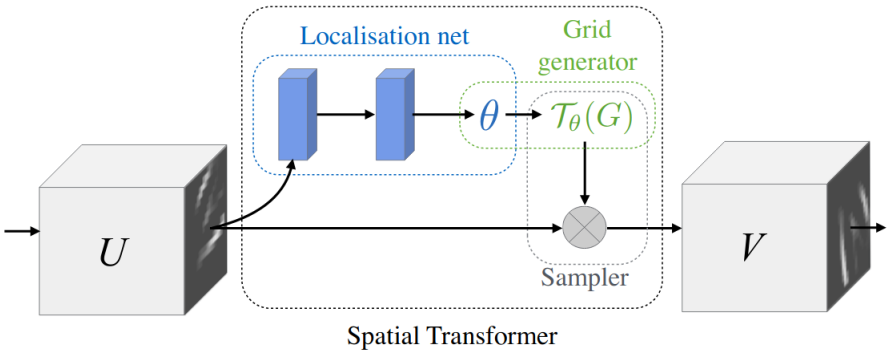


Figure 2.5.: **Spatial Transformer Architecture:** Firstly, the *localisation network* computes the transformation parameters $\boldsymbol{\theta}$. Secondly, the *grid generator* transforms the regular grid $\mathbf{G}$ of $\mathbf{V}$ into the new sampling $\widetilde{\mathbf{G}}$ on $\mathbf{U}$ using the parametrised transformation $\mathcal{T}_{\boldsymbol{\theta}}$ obtained from the localisation network. Lastly, the *sampler* computes the warped output feature map $\mathbf{V}$ by estimating the pixel values in the new sampling grid $\widetilde{\mathbf{G}}$ through interpolation of the pixel values of $\mathbf{U}$. Taken from Jaderberg et al. (2015).

2.3. Unsupervised Object-Centric Learning on Images

Due to the remarkable and rapid advances of modern deep learning in many challenging domains such as image recognition, natural language processing and self-driving cars, machine learning literature has exploded in the last decade with extensive ongoing research in many promising directions. Obtaining object-oriented representations from raw pixels in an unsupervised setting has become one of those directions which can be acknowledged by the fact that both ICML[4] and NeurIPS[5] were recently hosting workshops on this matter. Clearly, this line of research is still in its infancy and much remains to be determined as the organizers of the NeurIPS workshop state themselves: "How to learn object representations is still an open question."

This section aims to provide an understanding of why unsupervised object-centric learning has become so popular in the past few years and how state-of-the-art approaches try to learn structured image decompositions using unsupervised object-oriented learning on simple multi-object scenes.

2.3.1. Motivation

Deep reinforcement learning (Deep RL) merges traditional reinforcement learning and state-of-the-art deep neural networks which led to several breakthroughs such as end-to-end trained agents capable of outperforming humans in many Atari2600 games (V. Mnih et al., 2016), (Schrittwieser et al., 2020). Despite the recent growth and developments, three major challenges associated with Deep RL remain unsolved:

[4]The "Object-Oriented Learning: Perception, Representation & Reasoning" workshop took place in July 2020.
[5]The "Object Representations for Learning and Reasoning" workshop took place in December 2020.

- **sample efficiency**: typically hundreds of millions of training steps are needed,
- **generalization**: knowledge transfer is often very limited (Taylor & Stone, 2009),
- **interpretability**: a lack of transparency due to black-box models.

Developmental psychology and cognitive science indicates that humans excel in these challenges (e.g., humans typically learn to play Atari games within a few episodes, (Lake et al., 2017)) due to their ability to decompose the world into objects and their relations (Spelke & Kinzler, 2007). As summarized in a review by Baillargeon et al. (2009), even infants that are only a few months old have some kind of object-representation system that allows them to recognize, distinguish and reason about the physical behavior of objects. E.g., experiments have shown that infants are surprised when objects magically disappear behind a temporary introduced cover (the experimenter uses some hidden trap door) indicating that they are able to detect persistence violations.

Known as *core knowldege*, *common sense* or *intuitive physics*, humans are able to use this representation to reason about the consequences of their interactions. Although the mechanisms behind human cognition remain to be understood, Battaglia et al. (2018) argue that integrating relational inductive biases (such as an object-oriented representation) into deep learning architecture may be a stepping stone towards solving the aforementioned challenges and towards human-like intelligence.

The supervised solution is expensive (due to the dataset generation) and limited by the *symbol grounding problem* (Harnad, 1990), i.e., handcrafted representations are always limited to the designers understanding and cannot support ongoing adaptation. It is therefore desirable to avoid handcrafted parsing of scenes into objects and their relations. Furthermore, there is an increasing amount of unlabeled data available such that unsupervised object-centric learning seems much more promising.

Despite its huge potentials, there is still much work to be done and lots of open questions need to be addressed, e.g.,

- What biases should be integrated, i.e., what constitutes an object?
- How to store the representation, i.e., what format should be used?
- How to encode partial observability, i.e., how to deal with occlusions?
- How to measure success during learning, i.e., what is a good training signal?

The thesis takes only a glimpse on the field and is limited to image data. It should be noted that important inductive biases such as temporal continuity or learning about interactions and relations are not tackled.

2.3.2. Generative Models

In this section, two state-of-the-art generative models **Attend-Infer-Repeat (AIR)** and the **Multi-Object Network (MONet)** are reviewed. These models aim to obtain object-centric representations through structured reconstruction-based methods in multi-object scenes. Furthermore, both approaches build on the inductive bias that the world (or rather *simple images*

of the world) can be approximated as a composition of individual objects with the same underlying structure (i.e., different instantiations of the same class). To this end, they employ VAEs (see section 2.2.1) as learnable object instantiators. The difference between the two models lies in the structure-imposing mechanisms: AIR enforces much more structure than MONet.

It should be noted that there are many other works (Greff et al., 2019; Engelcke et al., 2019; Crawford & Pineau, 2019; Lin et al., 2020) in this direction which are not covered here since they are very similar in spirit to the presented approaches.

Attend-Infer-Repeat (AIR)

Eslami et al. (2016) introduce the Attend-Infer-Repeat (AIR) framework as an end-to-end trainable generative model capable of decomposing multi-object scenes into its constituent objects in an unsupervised learning setting. As the name suggests, the image decomposition process can be abstracted into three steps:

- **Attend**: Firstly, the model uses a ST (see section 2.2.2) to focus on a specific region of the image, i.e., crop the image.
- **Infer**: Secondly, the cropped image is encoded by a VAE. The same VAE is used for every cropped image.
- **Repeat**: Lastly, these stepped are repeated until the full image is described or the maximum number of steps is reached.

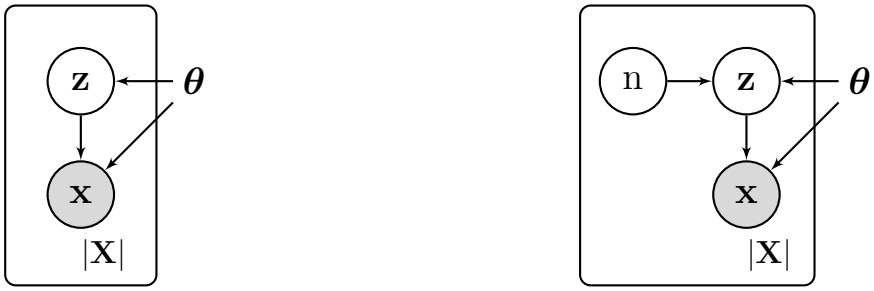
Similar to VAEs, AIR is based on a structured probabilistic model whose parameters are obtained via variational inference. In fact, AIR can be understood as a special VAE architecture with a group-structured latent space.

Latent Variable Model and Variational Approximation: In figure 2.6, the modeling assumption of AIR is compared against standard VAEs using directed graphical models as representations. Clearly, AIR introduces a novel layer of abstraction with an additional random variable n that describes the number of objects in a scene. The generative process can be summarized in three steps:

1. The number of objects n is sampled from some discrete prior distribution p_K (e.g., geometric distribution) with maximum value K.
2. The latent scene descriptor $\mathbf{z} = \left(\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(n)}\right)$ is generated by sampling from the scene description model $\mathbf{z} \sim p_{\boldsymbol{\theta}}(\mathbf{z}|n)$, where each vector $\mathbf{z}^{(i)}$ describes the attributes of one object in the scene and $\boldsymbol{\theta}$ denote the distribution parameters.
3. The image $\mathbf{x}$ is generated by sampling from the scene model $\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

Accordingly, the marginal likelihood of an image given the generative model parameters can be stated as follows

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{n=1}^{K} p_K(n) \int p_{\boldsymbol{\theta}}(\mathbf{z}|n) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) d\mathbf{z}. \tag{2.42}$$



a) Standard VAE
$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) d\mathbf{z}$

b) AIR
$p_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{n=1}^{K} p_K(n) \int p_{\boldsymbol{\theta}}(\mathbf{z}|n)\, p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$

Figure 2.6.: **Generative Process VAE vs AIR**: The directed graphical models represent the generative process of a standard VAE (a) and AIR (b), respectively. The dataset is denoted by $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{L}$ where each $\mathbf{x}_i$ describes an observable image. AIR introduces objects through structuring the generative process with an additional variable n as the number of objects.

Eslami et al. (2016) introduce an auxiliary distribution $q_{\boldsymbol{\phi}}(\mathbf{z}, n|\mathbf{x})$ as an approximation to the true, but unknown posterior $p_{\boldsymbol{\theta}}(\mathbf{z}, n|\mathbf{x})$. The objective of minimizing the KL divergence between the parametrized variational approximation and the true (but unknown) posterior is approximated by maximizing the ELBO, i.e.,

$$\mathcal{L}^{\text{ELBO}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = -D_{\text{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}, n|\mathbf{x}) \,||p_{\boldsymbol{\theta}}(\mathbf{z}, n)\right) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z},n|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, n)\right], \tag{2.43}$$

where $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, n)$ denotes the parametrized generative model (i.e., decoder in VAE parlance) and $p_{\boldsymbol{\theta}}(\mathbf{z}, n) = p_{\boldsymbol{\theta}}(\mathbf{z}|n)p_N(n)$ describes the prior on the joint probability of $\mathbf{z}$ and n that we need to define a priori. As a result, the whole model can be trained end-to-end by jointly optimizing the parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}$ for each image $\mathbf{x}_i$.

Structure-Imposing Mechanism and Network Architecture: In contrast to VAEs, it is not straightforward how to use neural networks in order to parametrize the probabilistic encoder $q_{\boldsymbol{\phi}}(\mathbf{z}, n|\mathbf{x})$ and decoder $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, n)$. Especially, the following obstacles are in our way:

1. How can we infer a variable number of objects n ? Actually, we would need to evaluate $q_{\boldsymbol{\phi}}(n|\mathbf{x}) = \int q_{\boldsymbol{\phi}}(\mathbf{z}, n|\mathbf{x}) d\mathbf{z}$ for all $n = 1, \ldots, K$ and then sample from the resulting distribution. However, the integral is intractable.

2. What the *first* and *second* object in a scene constitutes is somewhat arbitrary. As a result, object assignments $\left[\mathbf{z}^{(1)} \quad \ldots \quad \mathbf{z}^{(n)}\right] = \mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, n)$ should be exchangeable and the decoder $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, n)$ should be permutation invariant in terms of $\mathbf{z}^{(i)}$. Thus, the latent representation needs to preserve some strong symmetries.

Eslami et al. (2016) tackle these challenges by using the same decoder for each latent object instantiation $\mathbf{z}^{(i)}$ and defining inference as an iterative process using a recurrent neural network (RNN) that is run for K (maximum number of objects) steps. The corresponding high-level network architecture is shown and described in figure 2.7. Sampling $n \sim q_{\boldsymbol{\phi}}(n|\mathbf{x})$ is then

reparametrized through a sequential sampling scheme in which n is described as a variable-length vector $\mathbf{z}_{\text{pres}}$ of n ones followed by one zero. Therefore, $\mathrm{z}_{\text{pres}}^{(i)}$ is introduced as a binary variable sampled from a Bernoulli distribution whose probability is obtained from the RNN at each iteration step i. Whenever $\mathrm{z}_{\text{pres}}^{(i)} = 0$, the inference process stops and no more objects can be described, i.e., $\mathrm{z}_{\text{pres}}^{(i+j)} = 0$ is enforced for all j subsequent steps.

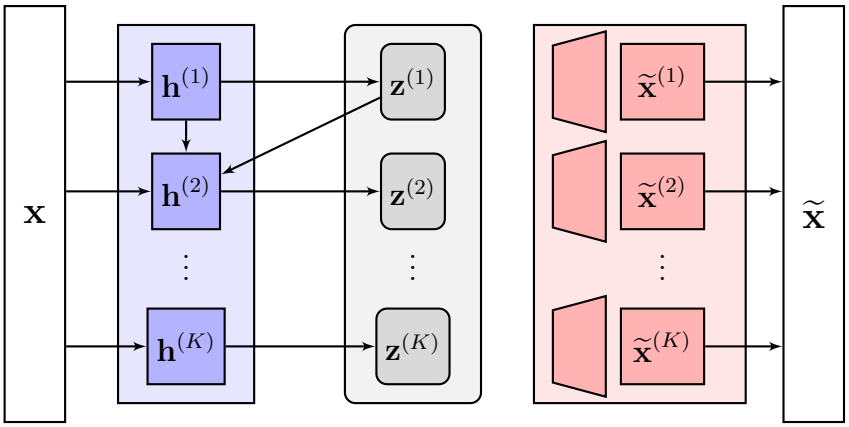


Figure 2.7.: **High-Level Network Architecture of AIR**: In essence, AIR can be understood as special VAE architecture with a group-structured latent space. To this end, the encoder is replaced with an recurrent inference network (left side, light blue) that is run for K steps. Each group $\mathbf{z}^{(i)}$ should ideally correspond to one object where the entries can be understood as the compressed attributes. The decoder is applied group-wise, i.e., each vector $\mathbf{z}^{(i)}$ is fed through the same decoder network.

The resulting encoder distribution takes the following form

$$q_{\boldsymbol{\phi}}(\mathbf{z}, \mathbf{z}_{\text{pres}}|\mathbf{x}) = q_{\boldsymbol{\phi}}\left(z^{(n+1)}, \mathbf{z}^{(1:n)}|\mathbf{x}\right) \prod_{i=1}^{n} q_{\boldsymbol{\phi}}\left(\mathbf{z}^{(i)}, z_{\text{pres}}^{(i)} = 1|\mathbf{x}, \mathbf{z}^{(1:i-1)}\right), \tag{2.44}$$

which is an equivalent representation of $q_{\boldsymbol{\phi}}(\mathbf{z}, n|\mathbf{x})$, see Appendix B of Eslami et al. (2016) for a proof. It is important to note that conditioning on the past latent variables $\mathbf{z}^{(1:i-1)}$ is necessary to avoid explaining the same object twice.

Additional structure is imposed by disentangling the objects appearance from its position through dividing the latent space of each object into "*what*" and "*where*". Thus, the process for generating one object in the original image dimension is structured by:

1. Sampling the latent appearance vector $\mathbf{z}_{\text{what}}^{(i)}$ and then generating an image of the object $\mathbf{x}_{\text{att}}^{(i)}$ by sampling it from the object model $p_{\boldsymbol{\theta}}(\mathbf{x}_{\text{att}}|\mathbf{z}_{\text{what}})$.
2. Sampling the latent transformation vector $\mathbf{z}_{\text{where}}^{(i)}$ and the decision variable $\mathrm{z}_{\text{pres}}^{(i)}$ in which $\mathbf{z}_{\text{where}}^{(i)}$ is used to scale and shift $\mathbf{x}_{\text{att}}^{(i)}$ into $\mathbf{x}^{(i)}$ using a ST and $\mathrm{z}_{\text{pres}}^{(i)}$ describes whether the contribution of $\mathbf{x}_{\text{att}}^{(i)}$ is added to the otherwise empty canvas $\mathbf{x}^{(i)}$.

Figure 2.8 summarizes the whole AIR architecture and its mode of operation.
Finally, in order to train the model, the reconstruction accuracy and regularization term of

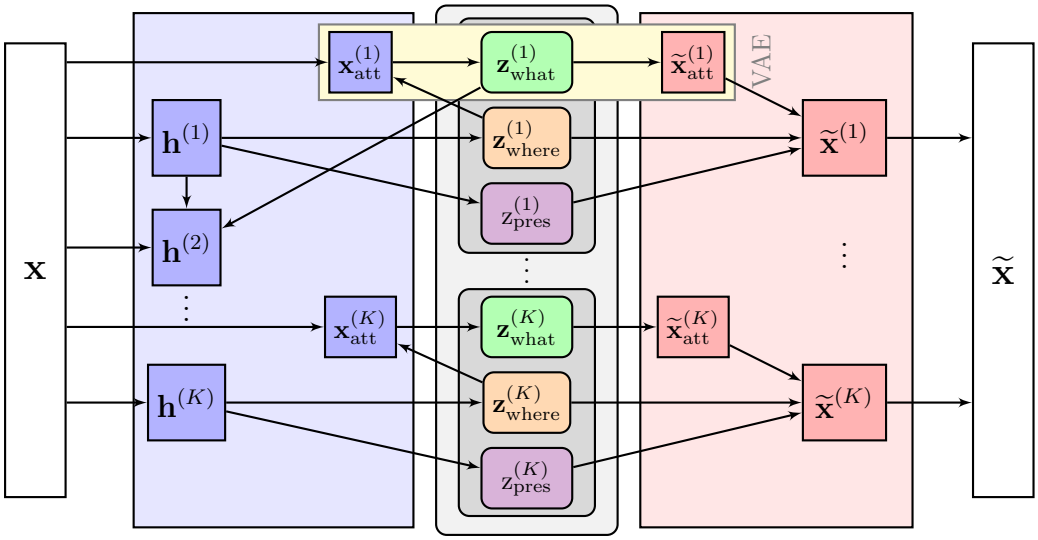


Figure 2.8.: **Full Network Architecture of AIR**: At each iteration step i, the inference network computes distribution parameters from which the binary variable $\mathrm{z}_{\mathrm{pres}}^{(i)}$ and the continuous variable $\mathbf{z}_{\mathrm{where}}^{(i)}$ are sampled. An attention crop $\mathbf{x}_{\mathrm{att}}^{(i)}$ is obtained through an affine transformation of $\mathbf{x}$ using $\mathbf{z}_{\mathrm{where}}^{(i)}$ (a ST is used for this operation). Ideally, the attention crop should contain one object. Then, the latent appearance vector $\mathbf{z}_{\mathrm{what}}^{(i)}$ of the object is determined by feeding each attention crop $\mathbf{x}_{\mathrm{att}}^{(i)}$ through the same VAE. Lastly, the inverse transformation is used to obtain a reconstructed image $\widetilde{\mathbf{x}}^{(i)}$ that only contains the reconstructed attention crop $\widetilde{\mathbf{x}}_{\mathrm{att}}^{(i)}$ if $\mathrm{z}_{pres}^{(i)} = 1$. The whole reconstructed image is simply a sum of the individual object images, i.e., $\widetilde{\mathbf{x}} = \sum_{i=1}^{K} \widetilde{\mathbf{x}}^{(i)}$.

the ELBO objective need to be computed, see equation 2.43. The reconstruction accuracy is approximated by using a single-sample Monte Carlo estimator of the expectation with the reparametrization trick for continuous random variables $\mathbf{z}_{\mathrm{what}}^{(i)}$, $\mathbf{z}_{\mathrm{where}}^{(i)}$ and the NVIL estimator for the discrete variables $\mathrm{z}_{\mathrm{pres}}^{(i)}$. The regularization term can be computed analytically, however requires appropriate prior distributions to be defined a priori.

Eslami et al. (2016) assume the following priors in their experiments

- $p_{\boldsymbol{\theta}}\left(\mathbf{z}_{\mathrm{what}}^{(i)}\right) \sim \mathcal{N}\left(\mathbf{0}, \mathbf{1}\right)$: A centered isotropic Gaussian for the latent appearance vector.
- $p_{\boldsymbol{\theta}}\left(\mathbf{z}_{\mathrm{where}}^{(i)}\right) \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\sigma}^2\mathbf{I}\right)$: A centered Gaussian for the transformation vector. Note that this distribution can encode any prior knowledge about the objects locality, i.e., average size and location of objects and their standard deviations.
- $p_{\boldsymbol{\theta}}\left(\mathbf{z}_{\mathrm{pres}}^{(i)}\right) \sim \mathrm{Bern}\left(p_{\mathrm{pres}}\right)$: An annealing geometric distribution as prior on the number of objects is used[6], i.e., the probability p_{pres} decreases from a value close to 1 to a small value close to 0 during the course of the training. Intuitively, this encourages the model

[6]This information is not mentioned in their paper, however Adam Kosiorek received it upon request from the authors, see http://akosiorek.github.io/ml/2017/09/03/implementing-air.html

to explore the use of objects in the initial phase and then constrains the model to use as few objects as possible.

Due to the patch-wise object instantiations, the model enforces the inductive bias of objects being locally self-contained. Furthermore, the trade-off between reconstruction accuracy and the cost that arises from instantiating objects due to the regularization term pushes the model towards recognizing similar structures as objects.

Performance and Limitations: As a proof of concept, Eslami et al. (2016) show that AIR could successfully learn to decompose multi-object scenes in distinct simplistic datasets. The learned representations could be used to accurately count and locate objects. Furthermore, the structured latent space is much easier to interpret compared to the unstructured counterpart learned by pure VAEs.

However, AIR does not have a mechanism to deal with partial observability and thus will assign a different "*what*" latent vector to partially occluded objects in comparison to their fully visible counterpart. Effectively, it treats partially occluded objects and their fully visible counterpart as different objects. Another limitation is that all objects are assumed to originate from the same class. While a class definition in general could be arbitrary complex such that all object originate from one class, the regularization term restricts the latent space and it is therefore unlikely that the model would perform well on datasets with multiple objects from different classes.

Multi-Object Network (MONet)

Similar to AIR, Burgess et al. (2019) developed the Multi-Object Network (MONet) as an end-to-end trainable model to decompose multi-object scenes into its constituent objects in an unsupervised learning setting. In essence, their model combines a VAE with a recurrent attention network to spatially decompose scenes into attention masks (over which the VAE needs to reconstruct masked regions) and latent representation of each masked region. In contrast to AIR, MONet does not contain a fully generative model and its latent space is less structured.

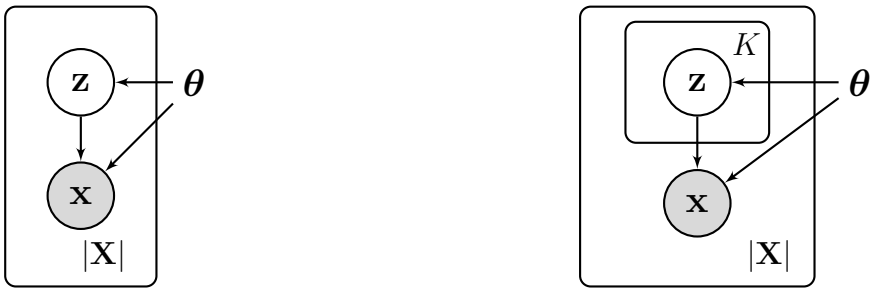
Latent Variable Model and Variational Approximation: MONet is based on a compositional generative model in which scenes are spatially decomposed into parts that have to be individually modeled through a common representation code. The spatial decomposition is achieved through a complete set of K attentions masks for the image $\mathbf{x} \in \mathbb{R}^{W \times H}$, i.e., $\sum_{i=1}^{K} \mathbf{m}^{(i)} = \mathbf{1}$ with $\mathbf{m}^{(i)} \in [0,1]^{W \times H}$ denoting the attention masks of the i-th component and K describing the maximum number of objects. Ideally, each masks should segment an individual object or the background.

The generative process consists of composing an image of K components $\mathbf{x}^{(i)}$ where each component is multiplied by its respective attention mask $\mathbf{m}^{(i)}$. The image components $\mathbf{x}^{(i)}$ are generated by firstly sampling from a latent representation $\mathbf{z}^{(i)} \sim p_{\boldsymbol{\theta}}(\mathbf{z})$ and then sampling the

component from the scene model $\mathbf{x}^{(i)} \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. Accordingly, MONet is based on a conditional generative model whose data likelihood is given by a pixel-wise mixture distribution

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{m}) = \sum_{i=1}^{K} \mathbf{m}^{(i)} \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, p_{\boldsymbol{\theta}}(\mathbf{z})\, d\mathbf{z}. \tag{2.45}$$

Figure 2.9 compares the modeling assumption of MONet against standard VAEs using directed graphical models as representations.



a) Standard VAE
$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) d\mathbf{z}$

b) MONet
$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{m}) = \sum_{i=1}^{K} \mathbf{m}^{(i)} \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, p_{\boldsymbol{\theta}}(\mathbf{z})\, d\mathbf{z}$

Figure 2.9.: **Generative Process VAE vs MONet**: The directed graphical models represent the generative process of a standard VAE (a) and MONet (b), respectively. The dataset is denoted by $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{L}$ where each $\mathbf{x}_i$ describes an observable image. MONet does not contain a fully generative process, but rather a conditional generative model in which the the probability of $\mathbf{x}$ is conditioned on the set of attention masks $\{\mathbf{m}_k\}_{k=1}^{K}$ with K as the number of objects.

Clearly, the attention mask distribution is decoupled from the scene model in the aforementioned conditional generation mechanism. This is problematic since there is no way to ensure that the image component $\mathbf{x}^{(i)}$ contains a complete object at the segmented area given in $\mathbf{m}^{(i)}$. As a workaround to this problem, Burgess et al. (2019) introduce a variational approximation $q_{\boldsymbol{\phi}}\left(\mathbf{z}^{(i)}|\mathbf{x}, \mathbf{m}^{(i)}\right)$ that is conditioned both on the image $\mathbf{x}$ and the respective attention mask $\mathbf{m}^{(i)}$. Additionally, the scene model is extended to also reconstruct the attention mask distribution $p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{z})$, where $\mathbf{c}$ denotes the discrete probabilities that pixels belong to a particular component i, i.e., $\mathbf{m}^{(i)} = p(\mathbf{c} = i)$.

Until now, we assumed that the complete set of attention masks $\{\mathbf{m}^{(k)}\}_{k=1}^{K}$ were simply given. In fact, a recurrent attention network $q_{\boldsymbol{\psi}}(\mathbf{c}|\mathbf{x})$ (conditioned on the image) is used to obtain them such that the whole model can be trained end-to-end with the following loss

$$\begin{aligned}\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta},\boldsymbol{\psi}}(\mathbf{x}) = & \underbrace{-\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{m})}\left[\log\left(\sum_{i=1}^{K} \mathbf{m}^{(i)} p_{\boldsymbol{\theta}}\left(\mathbf{x}|\mathbf{z}^{(k)}\right)\right)\right]}_{\text{Reconstruction Error between } \mathbf{x} \text{ and } \widetilde{\mathbf{x}}} + \beta \underbrace{D_{\mathrm{KL}}\left(\prod_{i=1}^{K} q_{\boldsymbol{\phi}}\left(\mathbf{z}^{(i)}|\mathbf{x}, \mathbf{m}^{(i)}\right) || p(\mathbf{z})\right)}_{\text{Regularization Term for Distribution of } \mathbf{z}} \\ & + \gamma \underbrace{D_{\mathrm{KL}}\left(q_{\boldsymbol{\psi}}(\mathbf{c}|\mathbf{x}) || p_{\boldsymbol{\theta}}(\mathbf{c}|\{\mathbf{z}^{(k)}\})\right)}_{\text{Reconstruction Error between } \mathbf{m} \text{ and } \widetilde{\mathbf{m}}},\end{aligned} \tag{2.46}$$

where the first two terms are derived from the standard VAE loss and the third term is used to couple the inferred attention masks $\mathbf{m}^{(i)} \sim q_{\boldsymbol{\psi}}(\mathbf{c}|\mathbf{x})$ with the reconstructed attention masks $\widetilde{\mathbf{x}}^{(i)} \sim p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})$. β is a hyperparamter that may help to learn disentangled representations (Higgins et al., 2017) and γ is a hyperparameter that can be used to encourage the model to get closer mask distributions.

Structure-Imposing Mechanism and Network Architecture: The network architecture of MONet can be abstracted into a VAE architecture with a group-structured latent space and a recurrent attention network that is used for the upstream task of inferring the attention masks, see figure 2.10. The recurrent attention network can be understood as a *segmentation network* and is based on the U-Net architecture (Ronneberger et al., 2015).

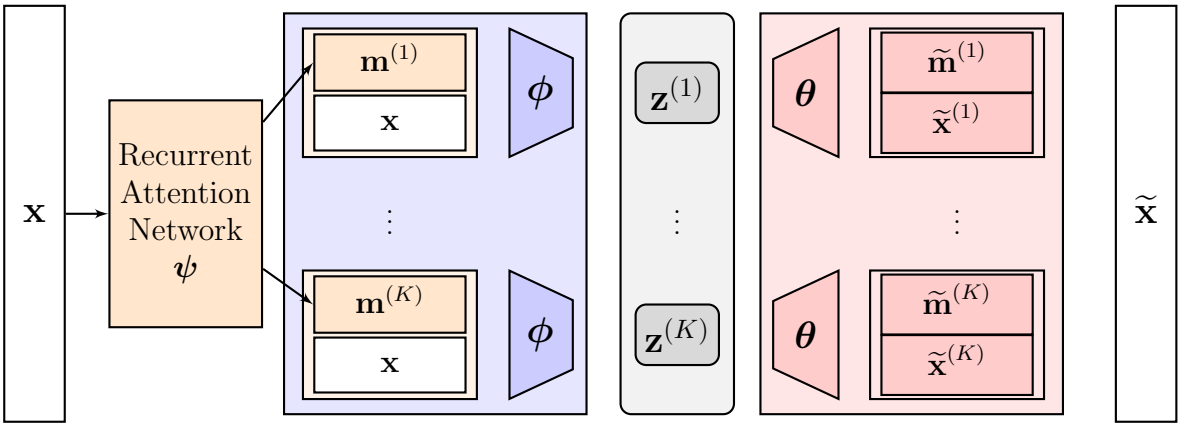


Figure 2.10.: **Network Architecture of MONet**: A recurrent attention network is used to obtain the attention masks $\mathbf{m}^{(i)}$. Afterwards, a group structured representation is obtained by feeding each concatenation of $\mathbf{m}^{(i)}, \mathbf{x}$ through the same VAE with encoder parameters $\boldsymbol{\phi}$ and decoder parameters $\boldsymbol{\theta}$. The outputs of the VAE are unmasked image reconstructions $\widetilde{\mathbf{x}}^{(i)}$ and mask reconstructions $\widetilde{\mathbf{m}}^{(i)}$. Lastly, the reconstructed image is computed via $\widetilde{\mathbf{x}} = \sum_i^K \mathbf{m}^{(k)} \cdot \widetilde{\mathbf{x}}^{(k)}$.

In contrast to AIR, MONet does not explicitly disentangle "*what*" from "*where*", however MONet uses the spatial broadcast decoder (Watters, Matthey, Burgess & Lerchner, 2019) to put an architectural prior on disentangling positions from other latent variables. The segmentation of the scene is entirely up to the model (i.e., shape and intensity of the segmentation can be arbitrary complex), whereas AIR is restricted to attention crops. Another difference to AIR is that in MONet each latent code $\mathbf{z}^{(i)}$ is used for the reconstruction of the scene. In practice, MONet learns to handle a variable number of objects by letting some codes encode empty scenes.

Similar to AIR, MONet uses the same VAE for every concatenation of $\mathbf{m}^{(i)}, \mathbf{x}$ and thus assumes that all objects originate from the same class. The trade-off between reconstruction accuracy and regularization pushes the model towards recognizing similar structures.

Performance and Limitations: Burgess et al. (2019) could show that MONet is capable of learning to decompose and represent even challenging 3D scenes. Notably, MONet has a principled way to deal with partial observability by adapting the attention masks, i.e., the

unmasked component reconstruction $\widetilde{\mathbf{x}}^{(i)}$ contains the full object and only its non-occluded part is taken via adapting $\mathbf{m}^{(i)}$.

The major limitation of MONet lies in its formulation, the loss in equation 2.46 is rather motivated empirically than probabilistically. The mask distribution is treated ambiguously as an inferred and a generated variable which is fairly unusual. While we would intuitively expect that the network pushes the VAE to obtain masks reconstruction $\widetilde{\mathbf{m}}$ that match the inferred masks $\mathbf{m}$, the network will at the same time also push the attention network to produce masks that match the mask reconstructions. As a result, training tends to become unstable and hyperparameter tuning becomes an essential prerequisite for good performance.

3. Methods

In this thesis, we adopt a novel view on structured image decompositions using unsupervised object-centric learning by building on a **multi-class paradigm**, i.e., we integrate the inductive bias that different objects may be instantiated from different classes. To the best of our knowledge, all previous work assumes in some way that all objects within an image originate from the same class. While a class can be arbitrary complex (i.e., the number of attributes defined in the latent space could be arbitrary large), it seems rather unnatural and inefficient to force an "*one class to rule them all*" paradigm. E.g., when humans think of *cars*, they have some representation that is hardly capable of being integrated into their representation of *hands*.

Melanie Klein introduced the concept of **internal objects** in the context of psychoanalysis to provide a way to theorize about the self, its development and mental disorders. In summary, an internal object is the mental and emotional image of an external object that is obtained through repeated cycles of projection and introjection, see Spillius et al. (2011) for a detailed review. We take inspiration from the Kleinian approach by using the notion of **prototypes** as the defining property of a class, i.e., each class has an assigned prototype. Object instantiations may then be understood as modifications to these prototypes. Due to the large variability in real-world images, we hypothesize that integrating a multi-class paradigm is an essential prerequisite for the real-world applicability of unsupervised object-centric learning.

In this chapter, novel generative models are presented that aim to retrieve a multi-class representation in a completely unsupervised learning setting. Therefore, the first section presents two simple approaches for the task of decomposing single-object-multi-class scenes. Interestingly, these models could also be understood as generative clustering approaches in which each recognized class corresponds to a cluster. In the second section, we show that these models can be easily integrated in state-of-the-art methods such that a multi-object-multi-class representation is obtained.

3.1. Single-Object-Multi-Class Models

In this section, we assume a dataset $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ of N images $\mathbf{x}^{(i)}$ each containing exactly one object. Furthermore, there are L classes and each object shall be derived from one of those classes. Each of the following approaches contains some form of recognition model that assigns a class variable $c^{(i)} \in \{1, \dots, L\}$ to a given image $\mathbf{x}^{(i)}$ which can equivalently be understood as cluster assignments.

We start by explaining why the standard VAE with a L-dimensional discrete latent space may be interpreted as a single-object-multi-class instantiator in this setting. Then, we build upon this model to show how we can further structure the generative process in order to obtain a better structured latent space.

3.1.1. Discrete VAE (D-VAE)

The discrete VAE (D-VAE) is simply a standard VAE with a discrete latent space. While this model is very limited in its application, it will be helpful to understand how the concept of prototypes naturally arises in it. Therefore, we investigate a D-VAE with a L-dimensional categorical distribution in the latent space whose samples are represented through one-hot vectors. Note that L represents an upper-bound on the number of classes.

Latent Variable Model and Variational Approximation: The generative process can be summarized in two steps:

1. The latent class vector $\mathbf{c}^{(i)}$ is sampled from a categorical prior distribution $p_{\boldsymbol{\theta}}(\mathbf{c})$.
2. The image $\mathbf{x}^{(i)}$ is generated by sampling from the scene model $\mathbf{x}^{(i)} \sim p_{\boldsymbol{\theta}}\left(\mathbf{x}|\mathbf{c}^{(i)}\right)$

Accordingly, the marginal likelihood of an image given the generative model parameters can be stated as follows

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{l=1}^{L} p_{\boldsymbol{\theta}}\left(\mathbf{x}|\mathbf{c}^{(l)}\right) p_{\boldsymbol{\theta}}\left(\mathbf{c}^{(l)}\right) = \frac{1}{L}\sum_{k=1}^{L} p_{\boldsymbol{\theta}}\left(\mathbf{x}|\mathbf{c}^{(l)}\right), \tag{3.1}$$

where we assumed that each class has an equal prior probability such that $p_{\boldsymbol{\theta}}\left(\mathbf{c}^{(l)}\right) = \frac{1}{L}$.

Following the standard VAE approach, we introduce an auxiliary distribution $q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{x})$ as an approximation to the true, but unknown posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. Then, the ELBO can be used as an objective to minimize the KL divergence between the variational approximation and true posterior, i.e.,

$$\mathcal{L}^{\text{ELBO}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = -D_{\text{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{x})\,||p_{\boldsymbol{\theta}}(\mathbf{c})\right) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{c}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{c})\right]. \tag{3.2}$$

Network Architecture and Structure-Imposing Mechanism: The associated neural network architecture is equal to a standard VAE with different notations for the latent space variable, see figure 3.1. It is important to note that there are only L distinct latent states possible, each represented through a one-hot vector. Accordingly, there are only L possible deterministic decoders and only L possible parametrizations for the generative model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{c})$.

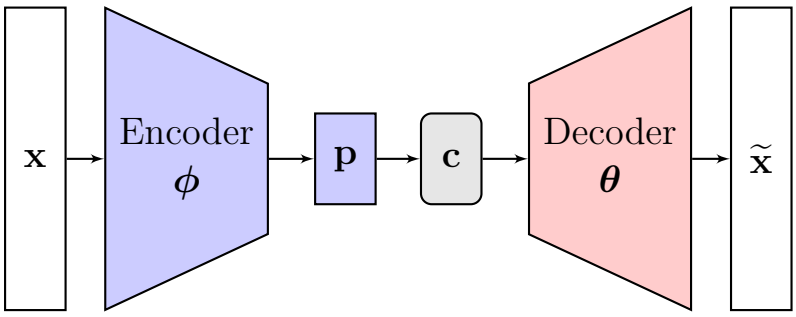


Figure 3.1.: **D-VAE Network Architecture**: A deterministic encoder $\boldsymbol{\phi}$ estimates the categorical distribution parameters $\mathbf{p}$ from which the class latent vector $\mathbf{c}$ is sampled. Lastly, a deterministic decoder $\boldsymbol{\phi}$ is used to reconstruct the input image.

Why do we learn prototypes in a D-VAE?

Intuitively, the D-VAE aims to reconstruct all images within the dataset $\mathbf{X}$ using only L realizations of $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{c})$. Thus, it has to find class-associated prototypes as the means[1] of the decoder distributions $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{c})$ in order to minimize the reconstruction error. More accurately, we can deduce that prototypes (centroids) are indeed the optimal solution by reviewing the generative process in which we assumed that each image is generated from one of those L distributions $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{c})$.

Limitations: Due to their modeling assumptions, D-VAEs are highly limited in practice: Each image must be highly resembled by some prototype image. I.e., positional and/or shape variations are only included through the variance of the scene model.

3.1.2. Discrete VAE with Spatial Transformer (D-VAE-ST)

The discrete VAE with a spatial transformer (D-VAE-ST) architecture aims to overcome the limitations of the D-VAE by adapting the generative process such that positional and shape variations are included. In this model, object instantiations can be understood as modifications to a class-associated prototype. These modifications are affine transformations performed by a spatial transformer.

Latent Variable Model and Variational Approximation: The generative process can be summarized in three steps:

1. The latent class vector $\mathbf{c}^{(i)}$ is sampled from a categorical distribution $p_{\boldsymbol{\theta}}(\mathbf{c})$ and the latent transformation vector $\mathbf{t}^{(i)}$ is sampled from centered Gaussian $p_{\boldsymbol{\theta}}(\mathbf{t}) = \mathcal{N}\left(\mathbf{0}, \boldsymbol{\sigma}^2\mathbf{I}\right)$.

2. The prototype image $\mathbf{x}_{\mathrm{p}}^{(i)}$ is generated by sampling from the prototype model $p_{\boldsymbol{\theta}}\left(\mathbf{x}_{\mathrm{p}}|\mathbf{c}^{(i)}\right)$.

3. The image $\mathbf{x}$ is generated by disturbing prototype $\mathbf{x}_{\mathrm{p}}$ using parameters $\mathbf{t}^{(i)}$.

Accordingly, the marginal likelihood of an image given the generative model parameters can be stated as follows

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{\mathbf{c}\sim p_{\boldsymbol{\theta}}(\mathbf{c}),\mathbf{t}\sim p_{\boldsymbol{\theta}}(\mathbf{t})}\Big[g\left(p_{\boldsymbol{\theta}}\left(\mathbf{x}_{\mathrm{p}}|\mathbf{c}\right), \mathbf{t}\right)\Big], \tag{3.3}$$

where $g(\cdot)$ denotes the affine transformation that is performed on the prototype $\mathbf{x}_{\mathrm{p}} \sim p_{\boldsymbol{\theta}}(\mathbf{x}_{\mathrm{p}}|\mathbf{c})$ using the parameters $\mathbf{t} \sim p_{\boldsymbol{\theta}}(\mathbf{t})$.

We treat the affine transformation and the class variable independently, therefore we introduce two independent variational approximations:

- $q_{\boldsymbol{\phi}}\left(\mathbf{c}|\mathbf{x}\right)$ to infer the class variable of an input image,
- $q_{\boldsymbol{\beta}}\left(\mathbf{t}|\mathbf{x}\right)$ to infer the transformation vector of an image.

[1] In this thesis, we only consider unimodal distributions.

The ELBO as the standard objective can be stated as follows

$$\mathcal{L}^{\text{ELBO}}_{\boldsymbol{\theta},\boldsymbol{\phi},\boldsymbol{\beta}} = D_{\text{KL}}\left(q_{\boldsymbol{\phi},\boldsymbol{\beta}}\left(\mathbf{c},\mathbf{t}|\mathbf{x}\right)||p_{\boldsymbol{\theta}}\left(\mathbf{c},\mathbf{t}\right)\right) + \mathbb{E}_{q_{\boldsymbol{\phi},\boldsymbol{\beta}}(\mathbf{c},\mathbf{t}|\mathbf{x})}\Big[\log \underbrace{p_{\boldsymbol{\theta}}\left(\mathbf{x}|\mathbf{c},\mathbf{t}\right)}_{g(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{c}),\mathbf{t})}\Big]. \tag{3.4}$$

Note that any prior knowledge about the positional and shape variance (e.g., scale or shear variance) can be encoded into the transformation prior $p_{\boldsymbol{\theta}}\left(\mathbf{t}\right)$.

Network Architecture and Structure-Imposing Mechanism: The resulting network architecture is described and depicted in figure 3.2. The D-VAE-ST adds an additional level of abstraction to the D-VAE architecture through the introduced affine transformation vector $\mathbf{t}$. Notably, the plain class-associated prototypes $\mathbf{x}_{\text{p}}$ can still be recovered from this architecture which results in a highly interpretable model.

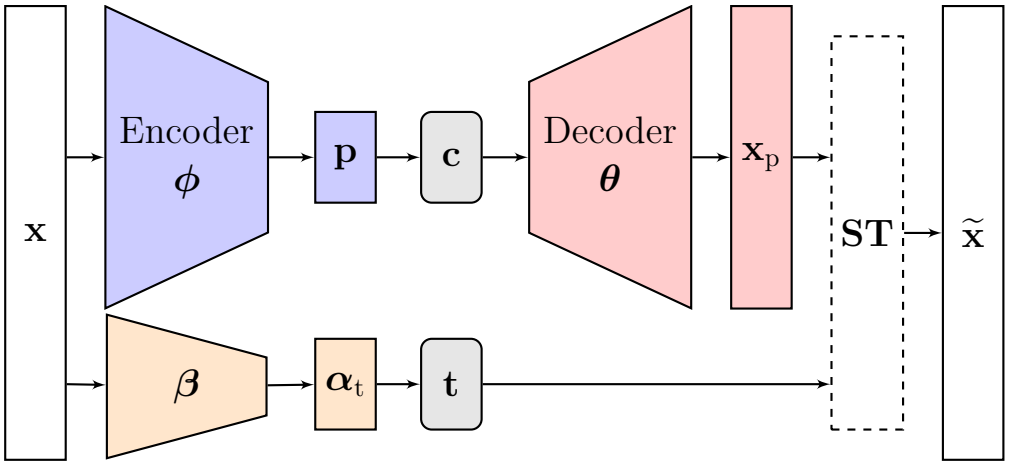


Figure 3.2.: **D-VAE-ST Network Architecture**: Essentially, the D-VAE-ST is an extension to the D-VAE architecture by simply adding a second inference network $\boldsymbol{\beta}$ and a ST that combines the prototype $\mathbf{x}_{\text{p}}$ and the sampled transformation vector $\mathbf{t}$. Note that $\boldsymbol{\alpha}_t$ denotes the distribution parameters of $q_{\boldsymbol{\theta}}\left(\mathbf{t}|\mathbf{x}\right)$. Random nodes are gray with round edges, while deterministic layers are hard-edged.

Limitations: Although the D-VAE-ST is based on a more realistic generative model, it is still only applicable to simple 2D data in which the same texture is assumed for all objects within a class.

3.2. Multi-Object-Multi-Class Model

In this section, we assume a dataset $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ of N images $\mathbf{x}^{(i)}$ each containing a variable number of objects that is upper-bounded by K. Furthermore, there are L classes and each object shall be derived from one of those classes.

3.2.1. Discrete Attend-Infer-Repeat (DAIR)

We propose to adapt AIR by replacing the *what*-VAE with a D-ST-VAE and term the resulting model DAIR. It is important to note that DAIR includes multiple discrete latent states: the

number of objects n and the associated class vector $\mathbf{c}^{(i)}$ for each of the objects i. Ideally, after training DAIR should be able to locate and classify objects within an image without any supervision. To the best of our knowledge, this is the first model with such capabilities.

Latent Variable Model and Variational Approximation: The generative process can be summarized in three steps:

1. The number of objects n is sampled from a discrete prior distribution p_K with maximum value K.
2. The latent scene descriptor $\mathbf{z} = \left(\mathbf{c}^{(1)}, \mathbf{c}^{(2)}, \ldots, \mathbf{c}^{(n)}\right) \in \{0,1\}^{L \times n}$ is generated by sampling from the scene description model $\mathbf{z} \sim p_{\boldsymbol{\theta}}(\mathbf{z}|n)$, where each vector $\mathbf{c}^{(i)}$ is one-hot encoded and associated with a prototype object $\widetilde{\mathbf{x}}_p$ (when decoded).
3. The image is generated by sampling from the scene model $\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

The scene description $p_{\boldsymbol{\theta}}(\mathbf{z}|n)$ and scene model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ are exactly as in the AIR model, see section 2.3.2. The only difference lies in the way how the object images $\mathbf{x}_{\text{att}}^{(i)}$ are created which is done through a D-ST-VAE instead of a standard continuous VAE.

Network Architecture and Structure-Imposing Mechanism: The resulting network architecture is depicted in figure 3.3. Since AIR creates attention crops and these attention crops should ideally focus on objects, we could use the trained D-VAE-ST in DAIR and test it on a single-object dataset.

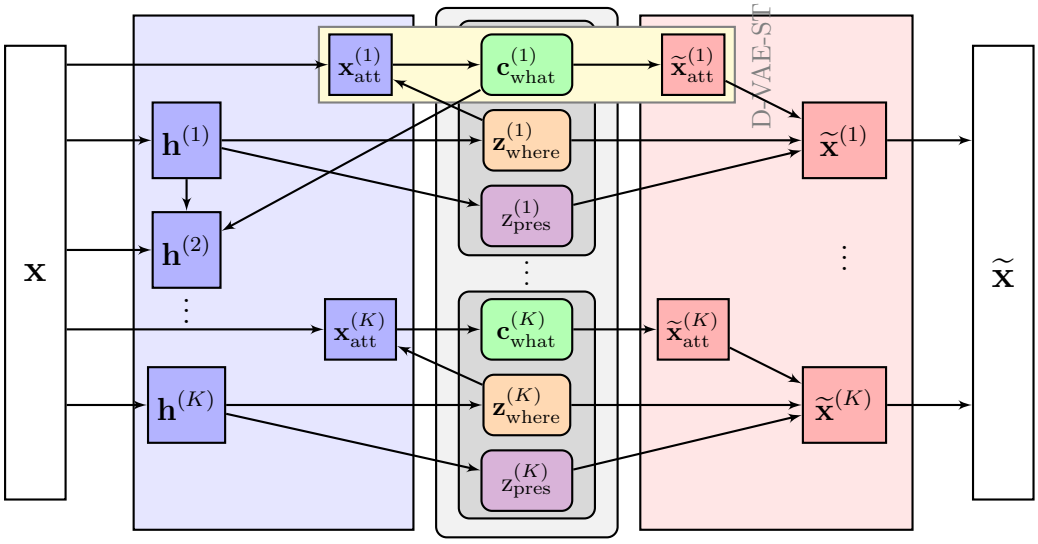


Figure 3.3.: **DAIR Network Architecture**: The DAIR architecture can be understood as replacing the *what*-VAE with a D-VAE-ST in AIR. For the sake of clarity, the representation hides the latent transformation vector $\mathbf{t}$ and the latent prototype images $\widetilde{\mathbf{x}}_{att,p}^{(i)}$. Random nodes are gray with round edges, while deterministic layers are hard-edged.

Limitations: DAIR suffers from the same limitations as the D-VAE-ST and AIR, i.e., it is a rather complex and sophisticated model even though it is only applicable to relatively simple datasets.

4. Experiments and Results

As a proof of concept, we perform three sets of experiments from which the first set serves to verify our implementation of discrete gradient estimators and to get a better understanding about the different hyperparameters in this context. The second set of experiments aims to evaluate our single-object-multi-class models using labeled datasets. In this setup, we can seek the best mapping from learned clusters to true classification labels such that the unsupervised classification accuracy can be estimated as a surrogate performance measure. Lastly, we compare AIR and DAIR on a multi-object-multi-class dataset. The full implementation (based on Pytorch) and the trained networks are available at `http://www.github.com/borea17/StrImaDec`.

4.1. Toy Experiment on Discrete Gradient Estimators

We build upon the toy problem described in section 2.1.1 which can equivalently be understood as an adaptation to the toy experiment presented by Tucker et al. (2017). The problem is summarized in figure 4.1: We take a fixed input vector $\mathbf{x}$, a fixed target $\mathbf{t} \in [0,1]^L$ and a categorical distribution $p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})$ whose discrete probabilities $\mathbf{p} \in [0,1]^L$ are obtained by feeding $\mathbf{x}$ through an encoder network $\boldsymbol{\theta}$. Furthermore, we include a sampling operation $\mathbf{c} \sim p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})$ such that discrete gradient estimators are necessary to make backpropagation applicable. Note that the sampled class variable c_i is represented through a one-hot vector $\mathbf{c}$. We wish to learn the optimal network parameters $\boldsymbol{\theta}$ based on the following stochastic optimization problem

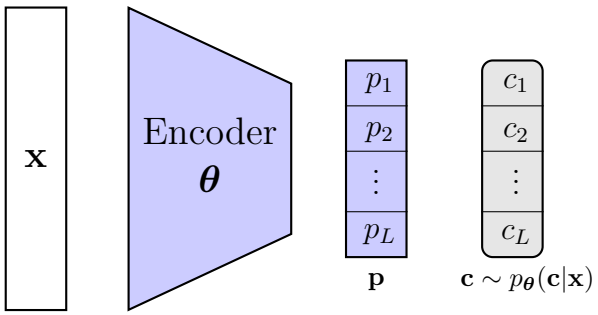
$$\min_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{t}) = \min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{c} \sim p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})} \left[\frac{1}{L} \sum_{i=1}^{L} (t_i - c_i)^2 \right], \tag{4.1}$$

where $\mathcal{L}_{\boldsymbol{\theta}}$ denotes the objective function and L represents the number of classes.

Notably, we can choose $\mathbf{t}$ such that the optimal solution for the objective $\mathcal{L}_{\boldsymbol{\theta}}$ and the discrete probabilities $\mathbf{p}$ are deterministic and unique. E.g., for $\mathbf{t} = \begin{bmatrix} 0.34 & 0.33 & 0.33 \end{bmatrix}^{\mathrm{T}}$, the optimal solution is to always sample $\mathbf{c} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$ resulting in the optimal discrete probabilities $\mathbf{p} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$ and the optimal objective

$$\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{t}) = \frac{1}{3} \left((1 - 0.34)^2 + (0 - 0.33)^2 + (0 - 0.33)^2 \right) \approx 0.2178. \tag{4.2}$$

Due to the finite number of probabilities p, we can compute *exact gradients* by feeding each possible sample $\mathbf{c}_i$ into the loss function weighting it with the corresponding probability p_i. As a result, we get the expected loss without any sampling operation (see section 2.1.3).



(a) Network Architecture

$$\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{t}) = \mathbb{E}_{\mathbf{c} \sim p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})} \left[\frac{1}{L} \sum_{i=1}^{L} (t_i - c_i)^2 \right]$$

(b) Objective Function

Figure 4.1.: **Toy Experiment:** The network architecture (a) and the corresponding objective function (b) are shown. In this setup, standard backpropagation does not work to obtain the optimal network parameters $\boldsymbol{\theta}$.

4.1.1. Results

In fact, the presented toy experiment can be seen as a generalization to the toy experiment of Tucker et al. (2017). Therefore, we verify that our implementation works by performing a replication experiment with $\mathbf{t} = \begin{bmatrix} 0.499 & 0.501 \end{bmatrix}^{\mathrm{T}}$ and comparing the results qualitatively with Grathwohl et al. (2018). Secondly, we extend this experiment to a 3-class categorical distribution with $\mathbf{t} = \begin{bmatrix} 0.34 & 0.33 & 0.33 \end{bmatrix}^{\mathrm{T}}$ on which we compare all discrete gradient estimators of section 2.1.3. Lastly, the extended toy experiment is used to analyse different hyperparameters and their effects on the final outcome. Details about the implementation can be found in appendix A.1.1.

Replication Experiment - Bernoulli Distribution

The toy experiment by Tucker et al. (2017) aims to minimize $\mathcal{L}_{\boldsymbol{\theta}}^{\text{Tucker}} = \mathbb{E}_{b \sim p_{\boldsymbol{\theta}}(b)} \left[(b - t)^2 \right]$, where t is a fixed target, $p_{\boldsymbol{\theta}}(b)$ is Bernoulli distribution parameterized by $\boldsymbol{\theta}$ and thus $b \in \{0, 1\}$ can be seen as a decision variable. While Tucker et al. (2017) used $t = 0.45$ to show that their introduced REBAR estimator is superior to a CONCRETE gradient estimator, Grathwohl et al. (2018) set $t = 0.499$ to highlight that their introduced RELAX estimator experiences much less variance than REINFORCE and REBAR.

Instead of a Bernoulli distribution, we can equivalently express this setup with a two-class categorical distribution in which $\mathbf{t} = \begin{bmatrix} 0.499 & 0.501 \end{bmatrix}^{\mathrm{T}}$ (see equation 4.1). In figure 4.2, we show our implementation results (Categorical distribution) with those from Grathwohl et al. (2018) (Bernoulli distribution). Due to the randomness within the experiment (sampling operation), we executed the experiment 50 times with different seeds and visualize the mean and standard deviation (shaded area).

Clearly, the overall results could be replicated: REBAR and RELAX perform similarly in terms of the loss and log variance, while RELAX experiences consistently less variance and approaches the optimal loss much faster. This indicates that our implementation is correct.

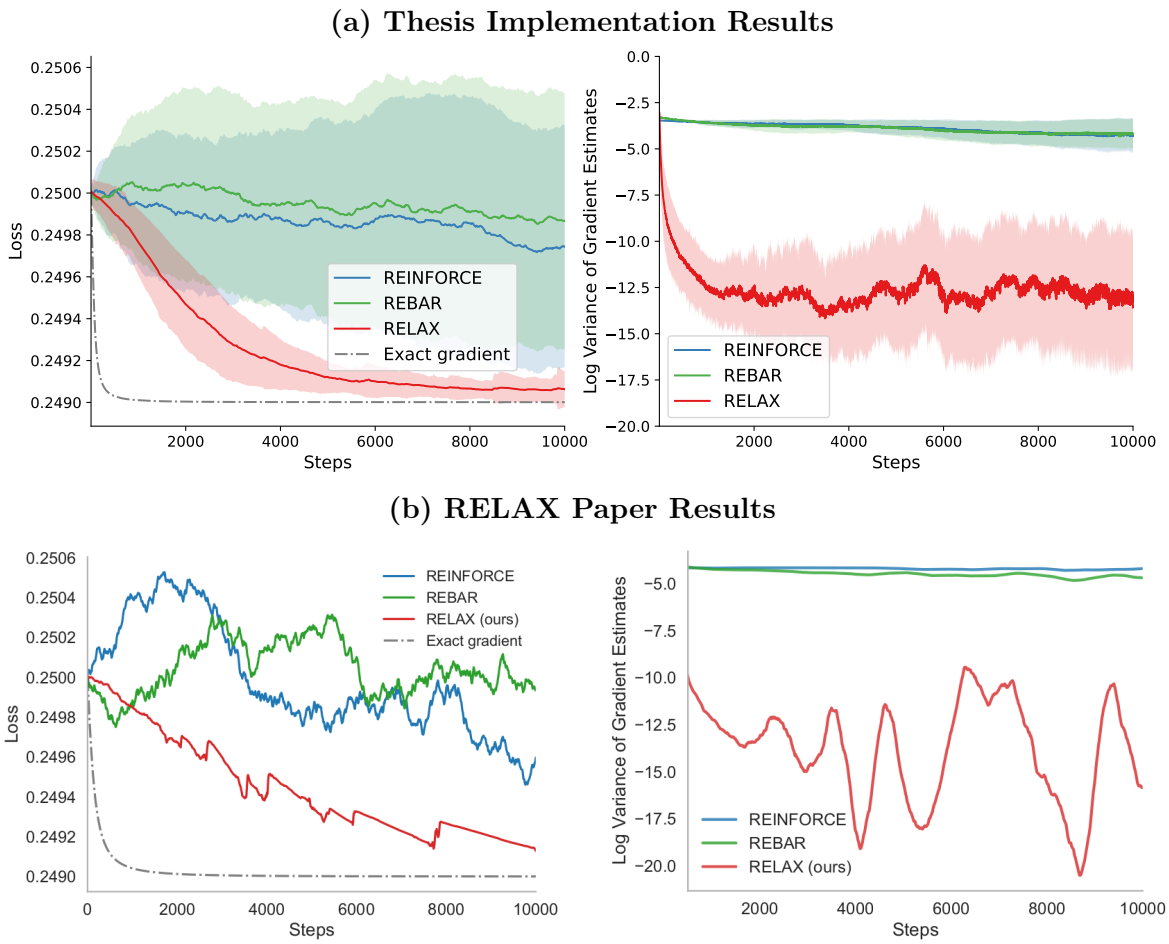


Figure 4.2.: **Replication Experiment Results**: We mimic the toy experiment of the RELAX paper (Grathwohl et al., 2018) and obtain similar results in our implementation for the loss and variance distributions of the respective estimators.

Toy Experiment - Categorical Distribution

In this experiment, we keep the structure of the replication experiment and only adapt the target to $\mathbf{t} = \begin{bmatrix} 0.34 & 0.33 & 0.33 \end{bmatrix}^{\mathrm{T}}$. Figure 4.3 shows the relative performance and gradient log-variance of REINFORCE, NVIL, CONCRETE, REBAR and RELAX. Again, we executed the experiment 50 times with different seeds and visualize the mean and standard deviation.

Clearly, RELAX outperforms the other gradient estimators. Notably, NVIL also consistently approaches the optimal loss although it seems to experience much higher variances than REINFORCE and REBAR in the initial phase. CONCRETE has consistently less variance than REINFORCE and REBAR, but does not converge to the optimal loss. This can be explained by the fact that CONCRETE is a biased estimator.

In table 4.1, we show a quantitative comparison between the different gradient estimates. This comparison highlights that RELAX and REBAR suffer from much higher step times compared to the other estimates.

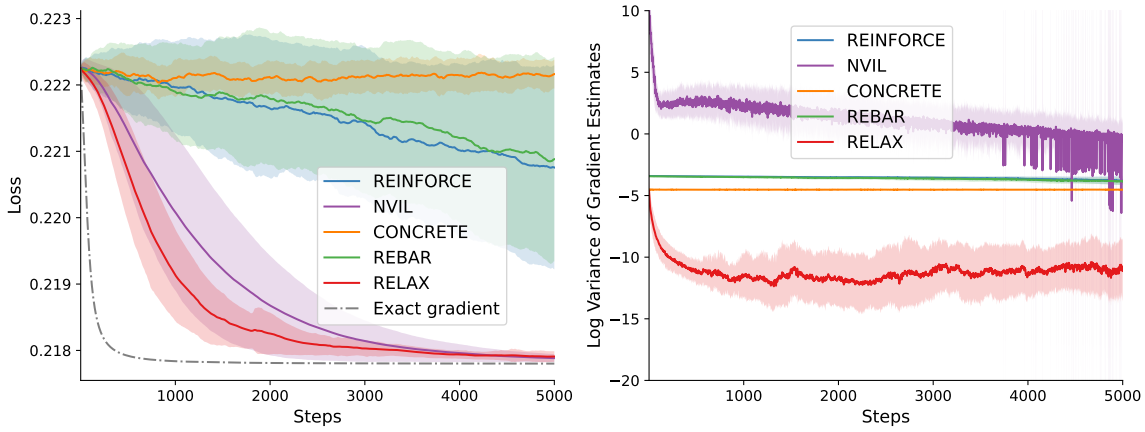


Figure 4.3.: **Toy Experiment Results**: We compare the discrete gradient estimators in this toy experiment with $\mathbf{t} = \begin{bmatrix}0.34 & 0.33 & 0.33\end{bmatrix}^{\mathrm{T}}$. For the CONCRETE gradient estimator, the temperature was set to $\lambda = 0.66$ as suggested by Maddison et al. (2017).

Table 4.1.: **Quantitative Toy Experiment Results:** We report the average absolute error (error between current loss and optimal loss), the average variance of the gradient estimate, the average step time and the average total time per gradient estimate.

	Error [1000]	Var [100]	Step Time [ms]	Total Time [1]
REINFORCE	3.71	2.9	**0.82**	**4.1**
NVIL	1.17	27,552.99	1.2	5.99
CONCRETE	4.33	1.09	1.17	5.87
REBAR	3.8	2.75	5.05	25.23
RELAX	**0.85**	**0.01**	5.46	27.3
Exact gradient	0.1	0	0.41	2.04

Hyperparameter Analysis

In the following experiments, we use the same setup as in the last section to analyze different hyperparameters. This analysis serves to

- empirically validate analytical findings,
- gain understanding and interpretation about the hyperparameters,
- verify our implementation.

For the analysis, each experiment was repeated 25 times with different seeds.

Batch Size Experiment: In this experiment, we highlight the significance of higher batch sizes. All gradient estimates except CONCRETE are Monte Carlo estimators. It has been analytically proven that the error of Monte Carlo integration decreases with $\sqrt{N}$ (in our setup N denotes the batch size). Figure 4.4 summarizes our experimental results and verifies that a larger batch size significantly improves the results of all gradient estimators except for

CONCRETE. Since a batch size of 50 fits easily into memory, the average step times did not increase significantly (in the order of nanoseconds).

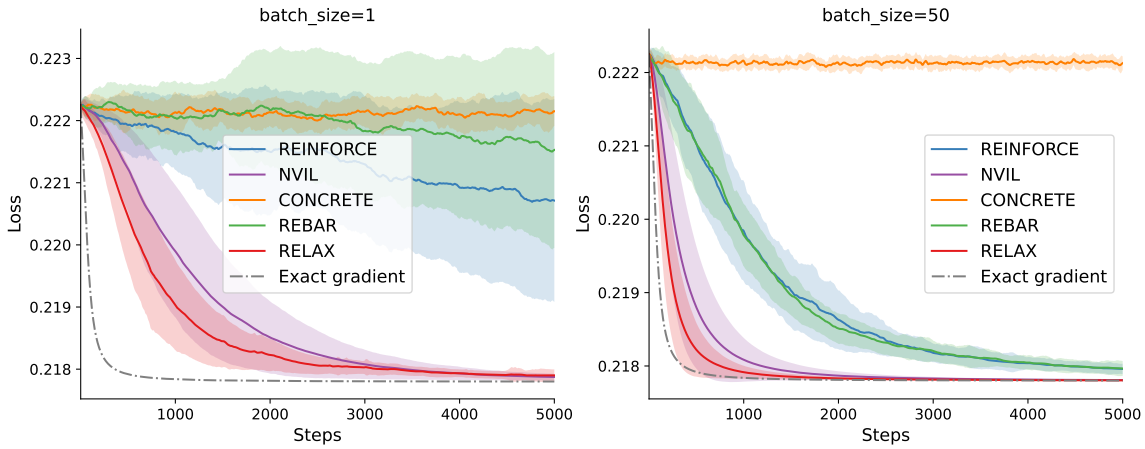


Figure 4.4.: **Batch Size Experiment Results**: We compare the discrete gradient estimators for different batch sizes. Clearly, a larger batch size improves the results significantly in all gradient estimates except for CONCRETE.

CONCRETE Experiment: In this experiment, we show the impact of different temperatures λ in the concrete distribution. Theoretically, a smaller temperature $\lambda \to 0$ should lead to higher variances and reduce the bias. Conversely, higher temperatures should decrease the variance. Figure 4.5 summarizes our experimental results and verifies that a larger temperature decreases the variance. However, the loss did not change much which is slightly unexpected. A possible explanation is that there is some kind of local minimum in which all three estimators get trapped.

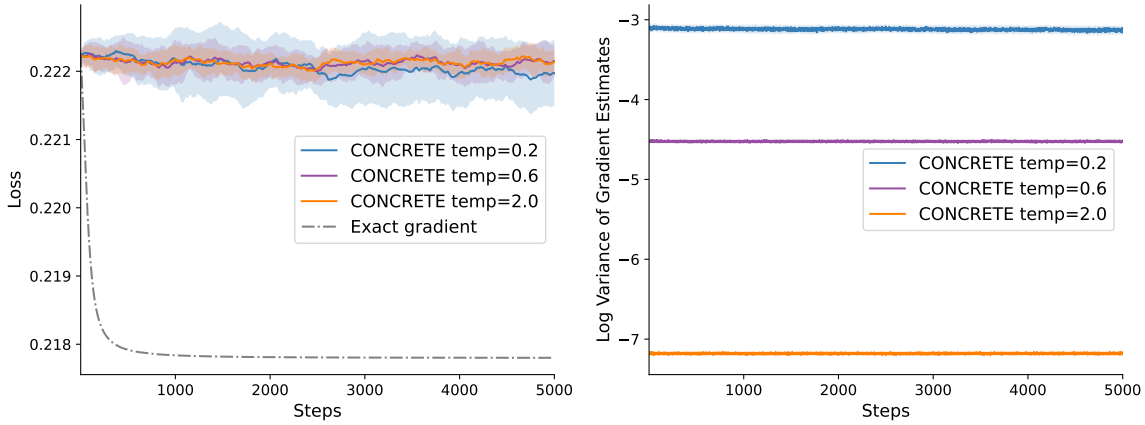


Figure 4.5.: **CONCRETE Experiment Results**: We compare the CONCRETE estimator for three different temperatures.

Tuning Learning Rate Experiment: In this experiment, we highlight the effects of different tuning learning rates for the gradient estimators in which the variance is decreased via an additional tuneable parameter, i.e., for NVIL, REBAR and RELAX. Table 4.2 summarizes the quantitative results. Clearly, for NVIL a learning rate that is higher than the base learning rate of 0.01 increases performance. For REBAR and RELAX setting the tuning learning rate equal to the base learning rate seems to lead to the best results.

Table 4.2.: **Quantitative Tuning Rate Experiment Results:** We report the average absolute error (error between current loss and optimal loss) and the average variance of the gradient estimate for different tuning learning rates. The learning rate of the other parameters was set to 0.01.

	tune lr=0.001		tune lr=0.01		tune lr=0.1	
	Error [1000]	Var [100]	Error [1000]	Var [100]	Error [1000]	Var [100]
NVIL	2.0	$1.6 \cdot 10^6$	1.3	$1.4 \cdot 10^5$	**1.08**	$\mathbf{2.1 \cdot 10^4}$
REBAR	4.19	2.78	**4.17**	**2.77**	4.09	$2.8 \cdot 10^{16}$
RELAX	1.25	0.06	**0.81**	**0.01**	1.36	3.1
Exact gradient	0.1	0	0.1	0	0.1	0

4.2. Single-Object-Multi-Class Experiments

We evaluate our single-object-multi-class models on four datasets:

- **SimplifiedMNIST**: A simpler version of the 28×28 images of handwritten digits (MNIST) dataset which only contains images of the digit classes 2, 6 and 9. As a results, this dataset contains $17,825$ images from 3 classes.
- **FullMNIST**: The well-known handwritten digits dataset containing 60,000 samples of 28×28 images from the 10 different digit classes.
- **FashionMNIST**: A clothing image dataset containing 60,000 gray-scale images (28×28) of 10 different classes (e.g., pullover, shirt, sneaker). This dataset was introduced by Zalando[1] as a direct drop-in replacement for the original MNIST dataset. Figure 4.6 shows exemplary images from FashionMNIST. There is much more variability within each class (e.g., different textures, shape variations), thus this dataset is particularly challenging for our models.
- **Letters**: A dataset containing 124,000 samples of 26 different handwritten letter images (28×28). This dataset has been introduced by Cohen et al. (2017) as an extension to the MNIST dataset.

[1] https://github.com/zalandoresearch/fashion-mnist

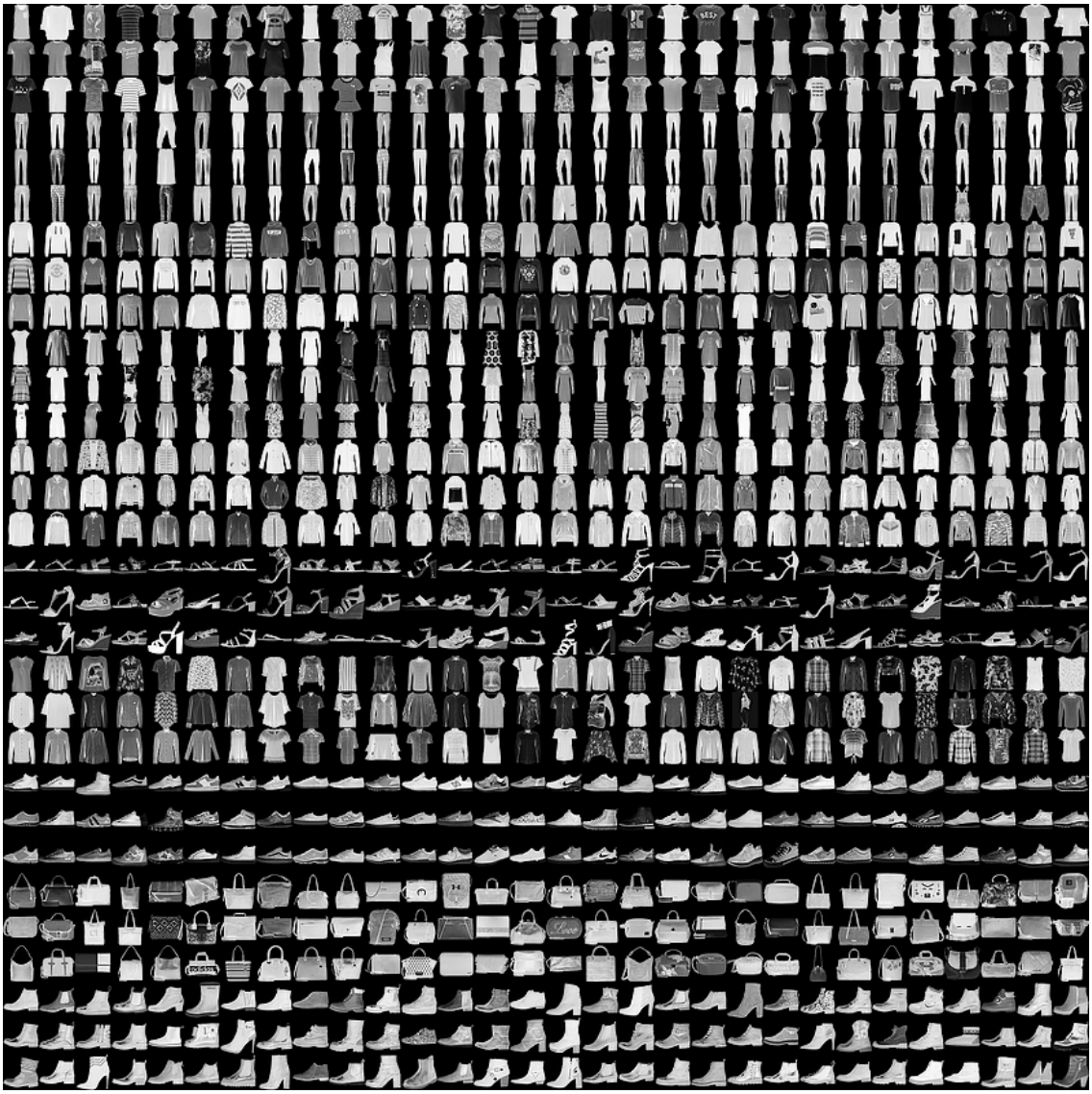


Figure 4.6.: **FashionMNIST Dataset**: Exemplary images for each class where each class takes three rows.

In this section, we will first describe how the performance of our models can be measured. Afterwards, the results for the D-VAE and D-VAE-ST are presented in which the number of classes of the model corresponds to the number of classes in the datasets. Lastly, we perform an overclustering experiment on the D-VAE-ST in which we aim to evaluate how the prior on the number of classes in the dataset effects the model performance.

Implementation details can be found in appendix A.1.2. It is important to note that we opted for a comparison between the gradient estimators and the models which is why we used a very simplistic hyperparameter setup. Furthermore, we can compute exact gradients in these models since there is only one discrete latent layer with a maximum of 26 latent states. However, due to the random network initialization and non-linear optimization different seeds lead to different results for the exact gradient approach.

4.2.1. Measuring Performance

VAEs are typically compared through the negative log-likelihood (reconstruction error) of the output and the KL-divergence of the latent space. While these measures may help to find VAEs that can use the latent space more efficiently (i.e., are not trapped in some local minima), it does not necessarily mean that the representation quality of such VAEs is better. The ultimate goal of learning object-centric representations is to obtain useful representations and not to produce good reconstructions. However, it is still under debate how to measure representation quality since a good representation is ultimately task-dependent.

Fortunately, we can view the learned representation in the discrete setup as a classification output and then compute the *"unsupervised classifcation accuracy"*, i.e., the best possible accuracy if each cluster is mapped to exactly one class. To compute this accuracy, we define an assignment matrix $\mathbf{M} \in \mathbb{N}^{C \times L}$ in which C denotes the number of clusters (latent space dimension) and L the number classes in the dataset. Each entry is defined as follows

$$\mathrm{M}_{i,j} = \text{number of samples classified as } j \text{ belonging to cluster } i, \tag{4.3}$$

i.e., to compute this matrix we need the actual labels from the dataset. We use this matrix to find the optimal assignment (which cluster belongs to which class) and compute the accuracy with this assignment.

If the number of classes is equal to the number of clusters, than we want to have perfect matching, i.e., each class has only one cluster assigned to it. In this case, we solve the assignment with the Hungarian Method (Kuhn, 1955).

If there are more clusters than classes, we simply take argmax along the rows to find the optimal assignment. In this case, we do not enforce that each class has one cluster assigned to it, i.e., we allow that classes have none or multiple clusters assigned to them.

4.2.2. D-VAE Experiment Results

Due to the randomness associated with the experiments, we perform each experiment (i.e., each gradient estimator for each dataset) ten times with different seeds. The quantitative results are presented in table 4.3 in which we show the negative log-likelihood (NLL), KL-divergence and the unsupervised classification accuracy. Despite the simplicity of the D-VAE approach, it achieves considerable results of 94.45% accuracy for SimplifiedMNIST, 58.86% accuracy for FullMNIST and 38.62% for Letters in the best cases. More detailed experimental results can be found at `https://tensorboard.dev/experiment/kl1NNgLaSEyBM0nKLJEIPg/`.

General Comparison between Gradient Estimators

We observe that huge differences between the best and worst accuracy occur several times in table 4.3, e.g., for the *exact gradient* in the SimplifiedMNIST experiment the best accuracy is 93.7% and the worst is 33.42%. A likely explanation is that the model gets trapped in local minima. This explanation is supported by the fact that there are also considerable differences in the worst and best NLL. Typically, the best and worst scores of the accuracy are associated

with the best and worst scores the NLL. While these differences occur for all estimators, it seems that they are most apparent for REBAR, RELAX and the exact gradient approach in the SimplifiedMNIST dataset. In future works, it would be interesting to study how we could make the training procedure more robust, e.g., using drop-out (Srivastava et al., 2014), batch normalization (Ioffe & Szegedy, 2015) and/or convolutional layers could potentially help.

There is much less deviation in the KL-divergence scores except for the CONCRETE gradient estimator. However, CONCRETE optimizes for a different latent distribution which is why it is less comparable to the other gradient estimators. Furthermore, CONCRETE seems to perform significantly worse in the Letters dataset (compared to all other estimators). This is probably due to the fact that we used the same temperature parameter for all three datasets. In practice, one should adapt and tune this parameter to the number of classes at hand.

Table 4.3.: **Quantitative D-VAE Experiment Results:** We report the average, best and worst scores for the negative log-likelihood (NLL), KL-divergence and unsupervised classification accuracy after training, i.e., taking all 10 training runs into account for each experiment.

	NLL			KL-Divergence			Accuracy [%]		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
SimplifiedMNIST									
REINFORCE	146.49	161.9	149.89	**0.59**	1.09	1.04	**94.45**	51.41	72.95
NVIL	146.41	**150.7**	**148.54**	1.09	1.09	1.09	94.08	51.2	72.95
CONCRETE*	157.09	166.72	160.22	0.72	**0.84**	**0.8**	93.02	**58.14**	**83.46**
REBAR	146.26	170.52	164.43	1.02	1.1	1.09	93.9	33.42	47.95
RELAX	**146.2**	170.49	167.98	1.09	1.1	1.1	93.63	33.42	39.45
Exact gradient	146.23	170.52	165.05	1.09	1.1	1.1	93.7	33.42	45.87
FullMNIST									
REINFORCE	132.56	135.73	133.69	2.24	2.26	2.26	57.38	44.46	51.29
NVIL	131.29	**133.77**	**131.85**	2.27	2.27	2.27	58.55	**51.36**	54.29
CONCRETE*	154.23	158.84	156.51	**1.55**	**1.62**	**1.59**	58.28	50.06	53.38
REBAR	131.07	134.57	132.24	2.27	2.28	2.27	**58.86**	50.86	**55.29**
RELAX	**130.89**	135.34	132.31	2.28	2.28	2.28	58.25	47.56	54.38
Exact gradient	131.23	144.87	136.83	2.28	2.29	2.29	58.41	45.22	52.69
FashionMNIST									
REINFORCE	108.32	112.75	109.84	2.25	2.27	2.26	57.68	48.33	52.2
NVIL	107.07	110.18	108.37	2.25	2.27	2.27	58.63	46.96	51.25
CONCRETE*	139.79	152.44	145.55	**1.75**	**1.8**	**1.78**	53.32	45.75	48.74
REBAR	107.04	109.89	108.55	2.26	2.27	2.27	57.13	47.67	52.62
RELAX	106.9	**109.24**	**108.14**	2.27	2.28	2.27	**59.12**	**49.43**	**54.98**
Exact gradient	**106.78**	111.3	108.65	2.28	2.28	2.28	58.76	48.16	54.88
Letters									
REINFORCE	158.44	160.75	159.58	3.18	3.2	3.19	35.34	30.8	33.37
NVIL	155.8	158.61	156.82	3.2	3.22	3.21	36.99	33.57	35.29
CONCRETE*	203.99	213.57	208.26	**2.13**	**2.23**	**2.19**	21.64	19.45	20.44
REBAR	155.26	156.31	155.72	3.22	3.22	3.22	**38.79**	**36.55**	**37.62**
RELAX	154.69	157.07	155.55	3.22	3.23	3.23	38.59	35.5	37.1
Exact gradient	**154.4**	**154.53**	**154.48**	3.23	3.23	3.23	38.22	35.21	36.58

* It should be noted that during training time, CONCRETE optimizes for a different NLL and KL-divergence since it is based on the concrete distribution instead of a categorical distribution. During test time, we exchange the concrete distribution by the categorical distribution.

Overall all gradient estimators perform similarly well (except for CONCRETE in the Letters dataset) in terms of the best accuracy. Furthermore, it can be noted that the D-VAE model seems to work fine for simplistic datasets, but struggles with more realistic datasets.

Visualizations

In figure 4.8, we compare learned prototypes in the SimplifiedMNIST setup for the best and worst case in terms of the measured accuracy. Clearly, in the best case the learned prototypes can easily be mapped to the true classes. In the worst case, some local optimum is reached and the learned prototypes are either noise (REBAR, RELAX, exact gradient) or multiple prototypes map to the same class. Furthermore, it can be noted that even in the best case the generated prototype are rather fuzzy than clean digit representations.

The results for the FullMNIST dataset are very similar to the SimplifiedMNIST results, which is why we only show the overall best and worst learned prototypes in figure 4.7. Notably, in both cases the majority of the learned prototypes can easily be mapped to a digit class. However, even in the best case multiple prototypes map to the same class. Furthermore, the generated prototypes are rather fuzzy again. This behavior results from the limitations of the generative model, i.e., the D-VAE can only use fuzziness or different classes to account for positional and/or shape variations.

REBAR (best trained model with accuracy: 58.86%)

REINFORCE (worst trained model with accuracy: 44.46%)

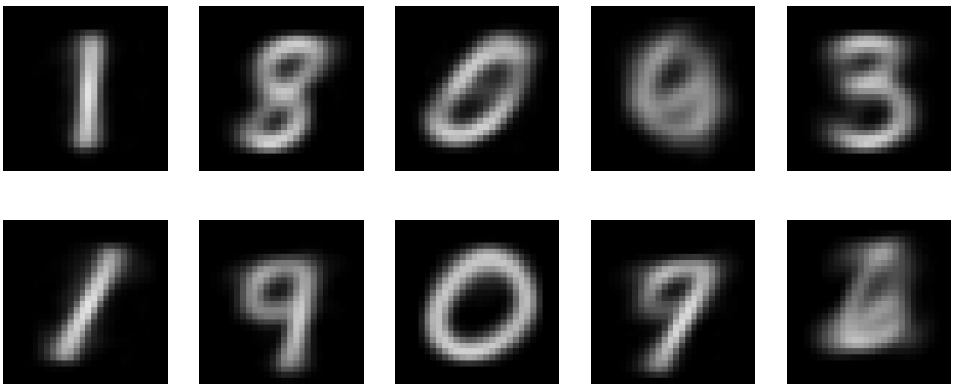


Figure 4.7.: **D-VAE FullMNIST Prototypes:** We compare the overall best learned prototypes with the worst ones based on the obtained accuracy scores.

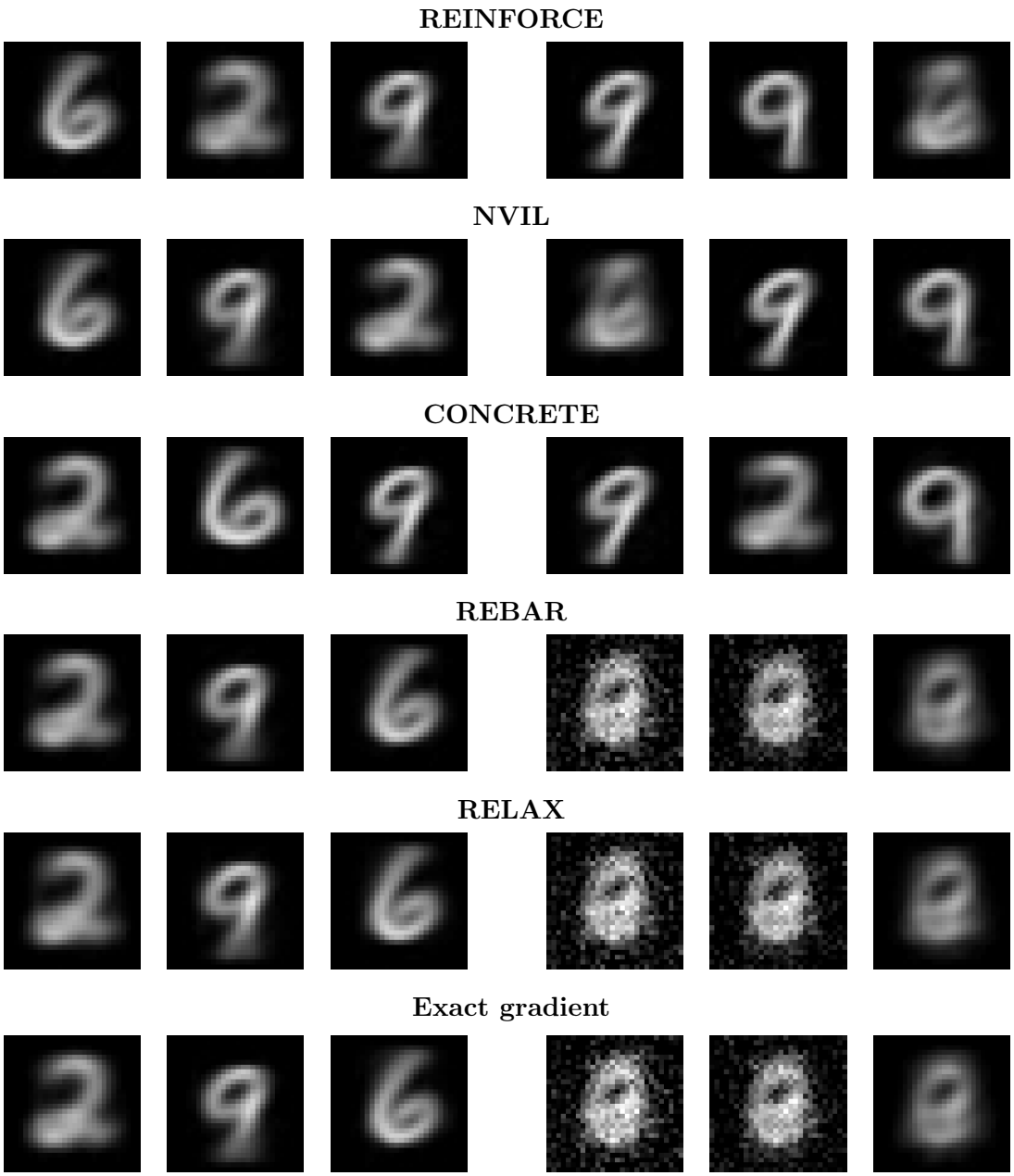


Figure 4.8.: **D-VAE SimplifiedMNIST Prototypes:** We compare the learned prototypes with the best accuracy score (left) with these that obtained the worst accuracy score (right).

Figure 4.9 visualizes the learned prototypes of the best and worst models for the FashionMNIST dataset. Although the D-VAE has no mechanism to create or identify different textures, most of the learned prototypes can easily be associated with the original classes. However, we observe again that multiple prototypes map to the same class (*Pullover* and *Shirt* with different color schemes) which we attribute to the lack of texture generation.

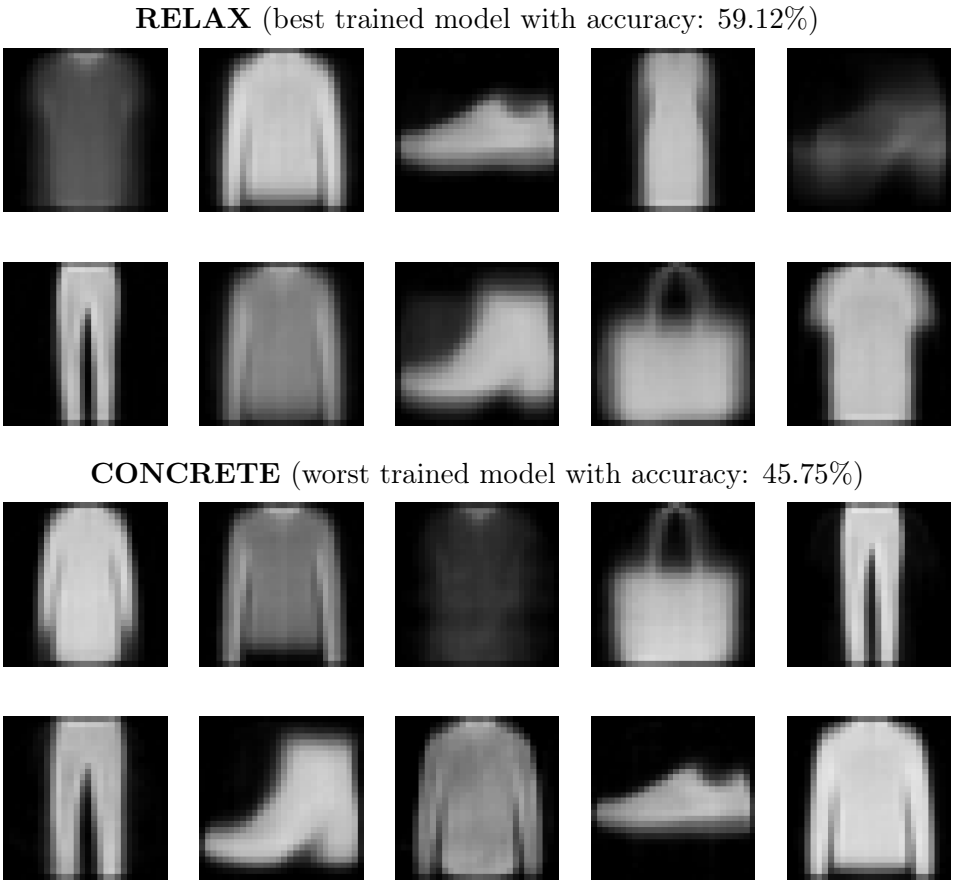


Figure 4.9.: **D-VAE FashionMNIST Prototypes:** We compare the overall best learned prototypes with the worst ones based on the obtained accuracy scores.

4.2.3. D-VAE-ST Experiment Results

Due to the randomness associated with the experiments, we perform each experiment (i.e., each gradient estimator for each dataset) ten times with different seeds. The quantitative results are presented in table 4.4 in which we show the NLL, KL-divergence and the unsupervised classification accuracy. The results of the D-VAE-ST are consistently superior to those of the D-VAE approach with 99.43% accuracy for SimplifiedMNIST, 93.52% for FullMNIST and 66.34% for Letters in the best cases. More detailed experimental results can be found at `https://tensorboard.dev/experiment/OReBEWwyRYGhbUqK1Vfk3Q/`.

General Comparison between Gradient Estimators

Similar to the D-VAE experiments, we observe again that there are huge differences between the best and worst accuracy scores, especially in the SimplifiedMNIST experiment. Again, it seems very likely that models sometimes get trapped into some local minima since typically a bad accuracy score is also connected with a higher NLL.

In contrast to the D-VAE results, the NLL is consistently lower (nearly halved) and the accuracy larger (especially for the FullMNIST and Letters dataset) which we attribute to

D-VAE-ST's capacity of modeling positional and shape variations. Furthermore, the KL-divergence is significantly higher which comes from the fact that the KL-divergence of the D-VAE-ST includes the KL-divergence for the latent transformation distribution. In the D-VAE, this distribution does not exist.

For the FashionMNIST dataset the accuracies are only slightly better than for the D-VAE approach, while the NLL is always significantly lower. We hypothesize that the lack of texture identification/generation is the main obstacle which cannot be addresed via affine transformations. Similar to the D-VAE experiment, CONCRETE performs significantly worse in the Letters dataset (compared to all other estimators) which we attribute to a bad choice for the temperature hyperparameter.

Table 4.4.: **Quantitative D-VAE-ST Experiment Results:** We report the average, best and worst scores for the negative log-likelihood (NLL), KL-divergence and unsupervised classification accuracy after training, i.e., taking all 10 training runs into account for each experiment. The KL-divergence includes both latent priors (equal class distribution and transformation).

	NLL			KL-Divergence			Accuracy [%]		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
SimplifiedMNIST									
REINFORCE	73.17	94.97	**76.98**	15.76	18.04	16.93	99.2	53.02	90.06
NVIL	**72.97**	**94.55**	79.6	15.76	17.21	**16.57**	99.32	51.09	85.08
CONCRETE*	84.59	87.25	85.98	16.59	**16.88**	16.76	99.23	**97.55**	**98.45**
REBAR	73.37	113.35	89.39	15.67	18.49	16.93	99.42	33.42	70.22
RELAX	73.62	153.54	101.18	**9.42**	18.08	15.66	99.38	33.42	64.74
Exact gradient	73.01	113.76	94.27	15.71	18.49	16.86	**99.43**	33.42	64.69
FullMNIST									
REINFORCE	68.43	75.7	72.15	16.85	17.39	17.05	84.91	58.78	73.4
NVIL	69.81	73.47	71.14	16.87	17.32	17.09	81.48	67.48	75.42
CONCRETE*	78.83	83.66	81.71	16.84	**17.17**	**16.99**	78.98	**75.64**	77.2
REBAR	69.8	72.3	71.5	16.88	17.24	17.06	81.74	68.02	71.75
RELAX	66.82	71.73	70.44	**16.76**	17.47	17.03	**93.52**	68.5	**77.3**
Exact gradient	**66.64**	**70.76**	**69.05**	16.91	17.4	17.18	82.78	67.98	77.27
FashionMNIST									
REINFORCE	66.07	69.84	67.61	14.06	16.04	15.07	57.01	46.49	51.71
NVIL	65.12	68.88	66.76	13.53	16.0	15.08	**61.91**	46.66	53.88
CONCRETE*	81.86	88.78	83.39	**13.46**	**14.71**	**13.8**	58.5	**50.48**	**56.31**
REBAR	65.06	68.18	66.61	14.07	15.82	15.27	55.48	48.19	51.68
RELAX	65.57	70.74	68.06	14.72	16.4	15.4	58.65	44.82	52.46
Exact gradient	**63.29**	**66.29**	**64.37**	14.0	15.76	15.06	59.03	48.83	53.54
Letters									
REINFORCE	87.45	92.77	89.51	18.03	**18.47**	**18.28**	60.6	50.75	55.3
NVIL	84.0	86.93	85.38	18.18	18.78	18.54	63.33	55.47	58.7
CONCRETE*	140.76	146.22	142.98	18.32	18.81	18.59	36.07	29.23	32.98
REBAR	83.29	85.25	84.32	18.11	18.48	18.31	64.5	56.73	61.53
RELAX	84.08	87.93	85.42	18.17	18.79	18.4	64.84	**57.23**	**61.87**
Exact gradient	**81.73**	**83.83**	**82.78**	**17.99**	18.5	18.3	**66.34**	56.04	61.41

* It should be noted that during training time, CONCRETE optimizes for different a NLL and KL-divergence as it uses the concrete distribution instead of a categorical distribution. During test time, we exchange the concrete distribution by the categorical distribution.

Visualizations

In figure 4.10, we compare the overall best and worst learned prototypes for all datasets. In contrast to the D-VAE, the prototypes with the best accuracy scores are much cleaner and less fuzzy. Notably, the best model for FullMNIST dataset contains all digit classes as very clean prototype images. For the worst case scenarios, we see again that either noisy prototypes are generated or multiple prototypes that map to the same class. In future works, it would be interesting whether adding some penalty term for similar prototypes would yield improved results in terms of consistency.

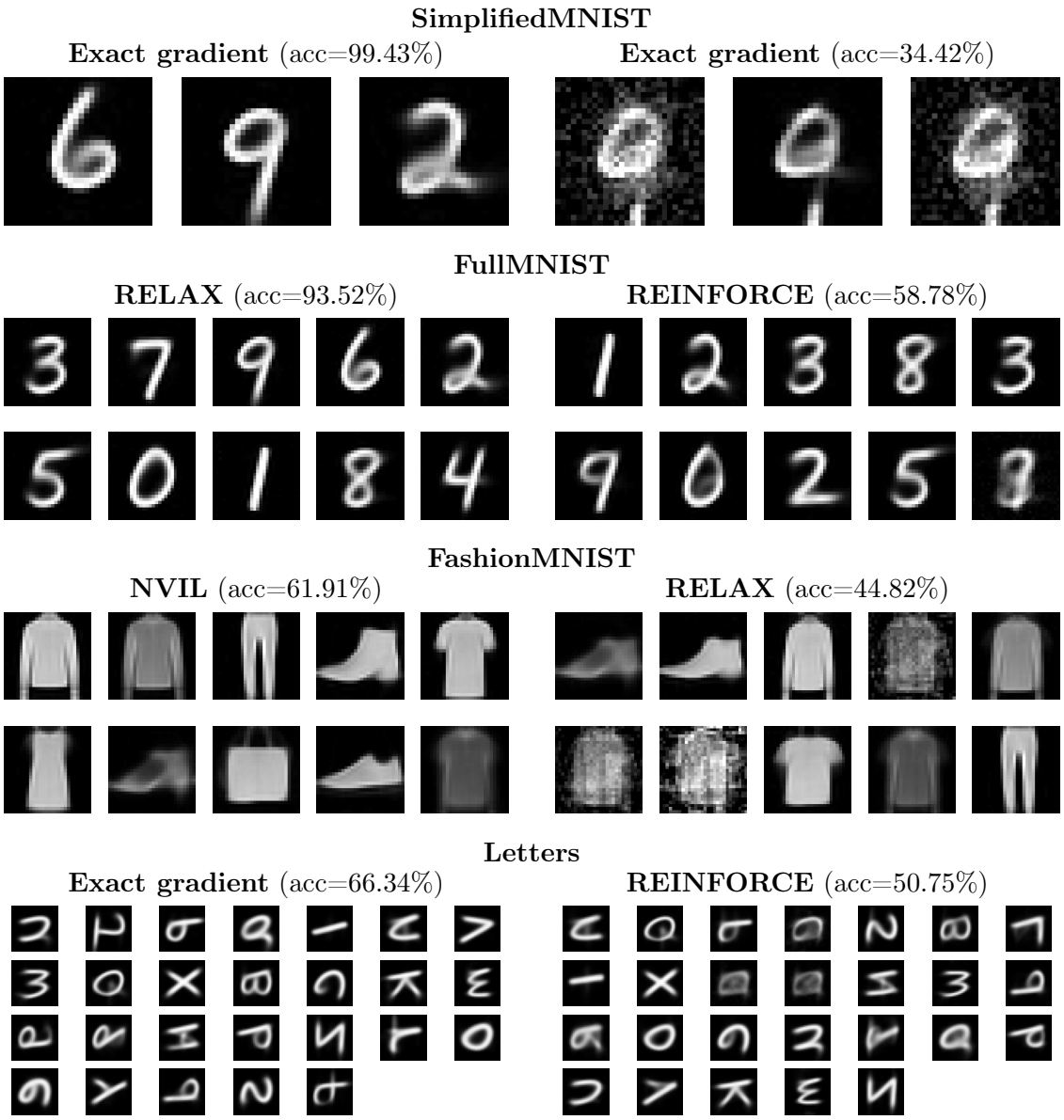


Figure 4.10.: **D-VAE-ST Prototypes**: We compare the overall best learned prototypes with the worst ones based on the obtained accuracy scores for each dataset. We excluded CONCRETE in the selection of the Letters dataset, since its hyper-parameter setup is clearly not suited for 26 classes.

Clearly, the prototypes are easily interpretable which is a main advantage of the D-VAE-ST model over other clustering approaches. I.e., we could simply label the prototype images and then obtain classification scores of 99.43% for the SimplifiedMNIST and 93.52% for the FullMNIST dataset. In this process, we massively reduced the number of labels to the number of classes in the dataset. Conversely, if the prototypes are very noisy or similar prototypes occur multiple times, one can deduce that the trained model fails to detect the correct clusters.

In figure 4.11, we show exemplary FullMNIST image reconstructions obtained through the D-VAE-ST together with the prototypes. This visualization illustrates impressively how positional and shape variations can easily be modeled with a fixed prototype.

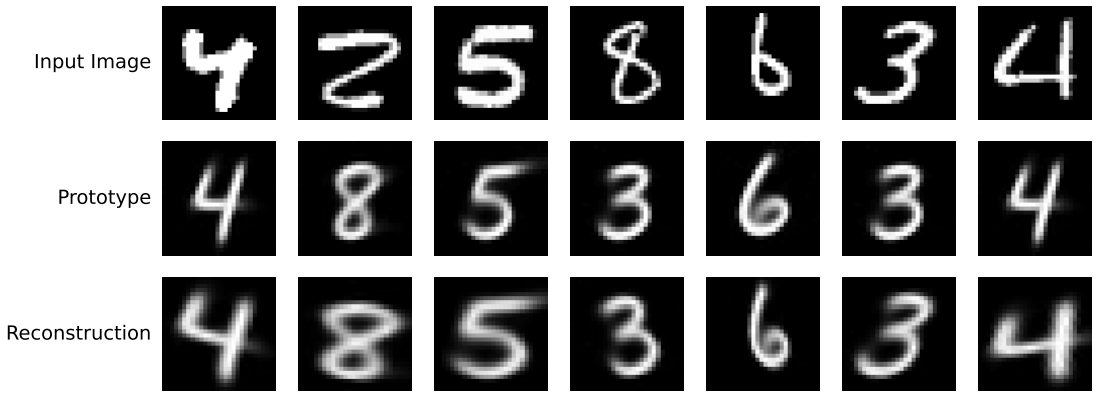


Figure 4.11.: **D-VAE-ST Exemplary Reconstructions**: We take the best RELAX model (acc=93.52%) on the FullMNIST dataset and show for 7 input images $\mathbf{x}$ their associated prototypes $\mathbf{x}_p$ and reconstructions $\widetilde{\mathbf{x}}$.

4.2.4. D-VAE-ST Overclustering Experiment

In the former experiments, we have always set the number of clusters to the number of classes. We noted that in the worst case scenarios often multiple prototypes mapped to the same class. Therefore, in this experiment we ask the question what happens if there are more clusters in the model than actual classes in the dataset. For the sake of clarity, we only perform this overclustering for the D-VAE-ST on the SimplifiedMNIST, FullMNIST and FashionMNIST datasets. Due to the randomness, we repeat each experiment 5 times for each gradient estimator and each dataset. The obtained best, worst and average accuracies are summarized in table 4.5. More detailed experimental results can be found at `https://tensorboard.dev/experiment/ioZI5WXMQ4Kocah7l6YblA/`.

We observe that increasing the number of clusters by one leads to consistently higher accuracy scores. However, for SimplifiedMNIST and FullMNIST further incrementing the latent space does hardly improve nor degrade the accuracies scores. To investigate this behavior, we show the learned prototypes of the best models for each cluster set in figure 4.12. We note that there are consistently two clusters that map to the digit class *2*: One with a curly curve and one with a angular curve on the bottom left. Clearly, affine transformations preserve the topology of the object such that it cannot create *holes*. Thus, D-VAE-ST struggles when the number of clusters is equal to the number of classes in the dataset.

This experiment highlights the benefits associated with the D-VAE-ST, namely

- **interpretability**: We can easily analyze the learned representations (prototypes).
- **simplicity**: The number of clusters can be increased without adapting any other hyperparameters (except for CONCRETE).

Table 4.5.: **Quantitative D-VAE-ST Overclustering Results**: We report the average, best and worst accuracy scores (in percent) for each cluster setup [2].

	|clusters| = |classes|			|clusters| = |classes| + 1			|clusters| = |classes| + 2		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
SimplifiedMNIST									
REINFORCE	99.1	52.25	80.65	99.22	98.84	99.1	99.21	66.25	85.89
NVIL	99.33	51.47	72.02	99.34	66.23	92.68	99.4	99.15	99.31
CONCRETE*	99.0	**97.92**	**98.38**	**99.66**	**99.65**	**99.66**	**99.6**	**99.44**	**99.53**
REBAR	99.23	33.42	56.56	99.47	66.32	92.78	99.53	99.34	99.43
RELAX	66.15	33.42	49.84	99.49	66.29	92.77	99.56	66.33	92.88
Exact gradient	**99.41**	33.42	56.53	99.51	33.42	79.56	99.57	99.38	99.48
FullMNIST									
REINFORCE	80.41	65.98	73.39	93.28	74.28	85.48	93.07	81.3	87.53
NVIL	81.41	67.42	73.38	**94.4**	76.52	85.03	85.5	83.64	84.78
CONCRETE*	78.72	**76.74**	78.02	89.12	79.07	81.92	89.71	78.78	85.34
REBAR	81.84	70.66	76.82	87.13	**84.01**	84.85	85.5	**84.95**	85.26
RELAX	82.72	68.07	78.11	84.82	75.93	82.6	85.65	83.99	84.97
Exact gradient	**82.79**	68.06	**78.27**	93.38	83.03	**86.48**	**94.82**	83.23	**88.33**
FashionMNIST									
REINFORCE	51.84	48.98	50.79	59.7	55.52	57.6	**63.99**	51.34	56.9
NVIL	57.45	50.62	52.88	58.56	56.09	57.07	60.33	52.48	57.8
CONCRETE*	**58.38**	**50.9**	**55.42**	59.26	55.81	58.22	59.63	51.39	56.32
REBAR	55.47	47.61	51.28	59.29	51.84	56.02	63.01	**57.23**	**59.37**
RELAX	52.23	49.92	51.13	**64.73**	51.73	58.38	63.88	53.01	58.24
Exact gradient	54.7	50.48	51.76	64.51	**57.02**	**60.05**	59.7	56.19	58.18

* It should be noted that during training time, CONCRETE optimizes for different a NLL and KL-divergence as it uses the concrete distribution instead of a categorical distribution. During test time, we exchange the concrete distribution by the categorical distribution.

[2] The numbers for the equal case do not align with table 4.4, since we only repeated the experiments 5 times here.

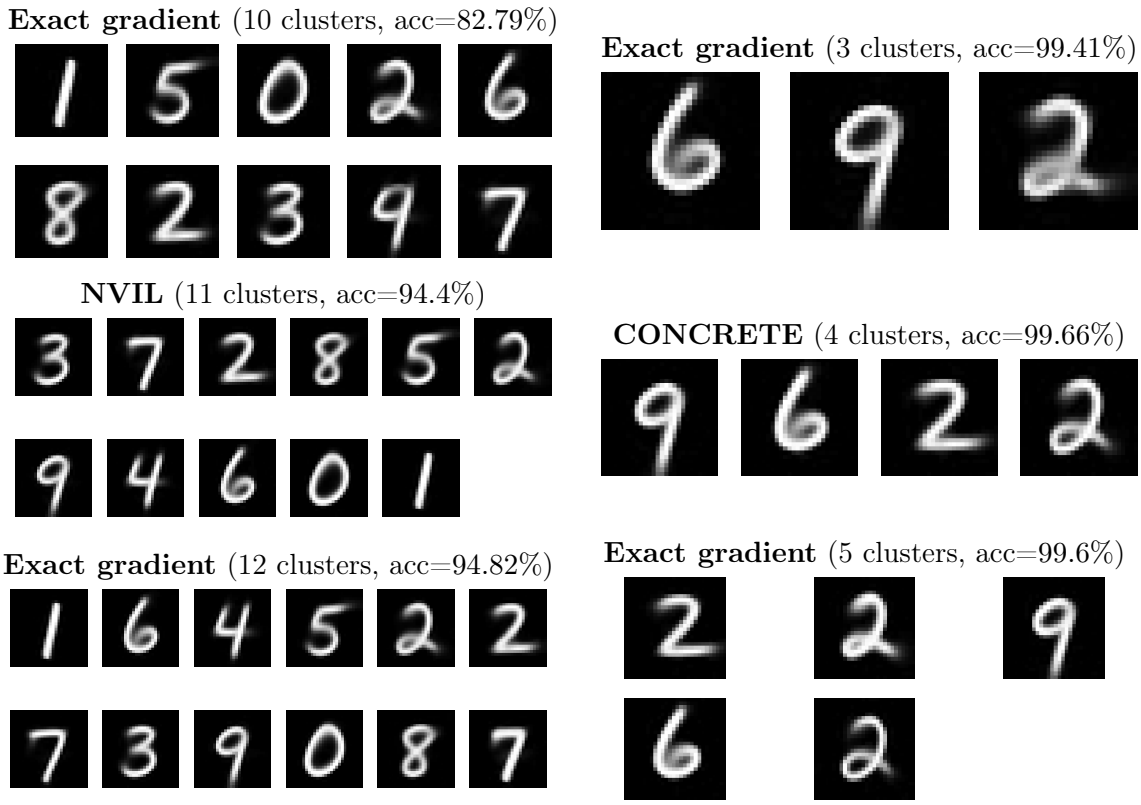


Figure 4.12.: **D-VAE-ST Overclustering Prototypes**: The learned prototypes for each cluster setup's best model are shown (left: FullMNIST, right: SimplifiedMNIST). Higher accuracies seem to be associated with digit class *2*.

4.3. Multi-Object-Multi-Class Experiment

We compare DAIR against AIR on the self-generated MultiMNIST dataset which consists of $20,000$ samples of 64×64 gray-scale images containing zero, one or two non-overlapping random MNIST digits with equal probability. For the sake of simplicity, we ensure that there is minimal overlap between the digits. Eslami et al. (2016) did not publish their original implementation, thus we can only compare against our reimplementation. Implementation details about AIR and DAIR can be found in appendix A.1.3.

It is important to note that AIR is a rather complex model and due to a limited amount of computational resources, we did not perform a dedicated hyperparameter search and aimed to be as close as possible to the original implementation. Eslami et al. (2016) used the NVIL estimator for the discrete number of objects, we extended their model and therefore only used the NVIL estimator as well.

Due to the randomness associated with the experiment, we repeated each experiment ten times with different seeds. The quantitative results are presented in table 4.6 in which we compare the best, worst and average count accuracy (i.e., number of detected objects vs. objects of objects at hand), the REINFORCE term with neural control variate on the number of objects and the negative log-likelihood (using the same fixed variance). More detailed experimental results can be found at `https://tensorboard.dev/experiment/7GSRoBArQAOQ4vtn55h08w/`.

Clearly, AIR outperforms DAIR in every aspect. In figure 4.13, we visualize the learned prototypes and reconstructions of DAIR in the best cases in terms of high counting accuracy and low negative log-likelihood. Somehow the D-VAE-ST does not learn useful prototypes, although its capacity inside DAIR is equal to the former experiments. Possible explanations are a faulty implementation or some local optimum in which DAIR gets trapped. For future works a hyperparameter study is advised.

Our implementation of AIR replicates the results from Eslami et al. (2016). Figure 4.14 shows impressively that in the best case AIR correctly locates and reconstructs single digits.

Table 4.6.: **Quantitative MultiMNIST Experiment Results:** We report average, best and worst count accuracy, negative log-likelihood and REINFORCE scores for each model based on 10 training runs.

	count accuracy [%]			NLL			NVIL Term		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
AIR	**97.26**	**54.86**	**82.36**	**20.33**	**21.85**	**21.05**	-0.02	**0.39**	**0.1**
DAIR	78.76	32.03	58.01	33.67	40.61	37.86	**-0.71**	7.2	1.38

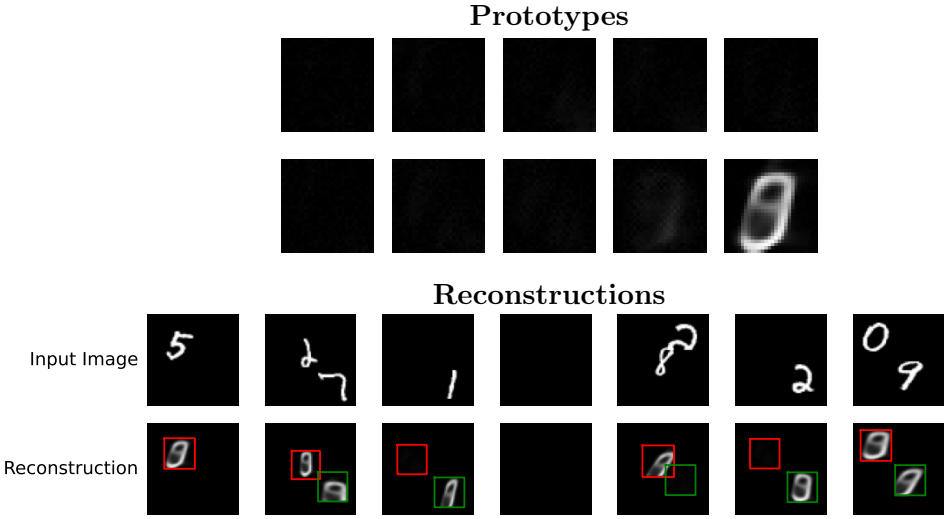


Figure 4.13.: **DAIR Prototypes and Reconstructions**: The learned prototypes and full reconstructions for the best DAIR model in terms of count accuracy and negative log-likelihood are shown. Clearly, the model does not use its latent space efficiently and seems to be trapped in some local optimum. Note that the rectangular boxes are not part of the model output, but added to highlight where and how many objects are located by the model.

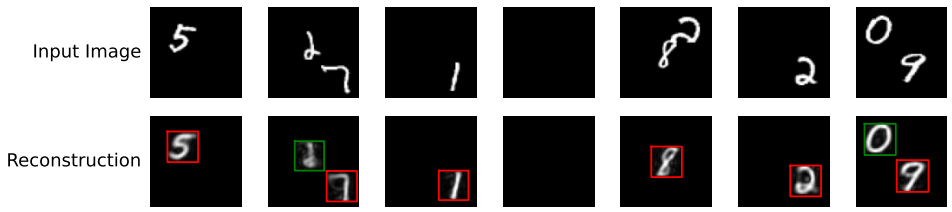


Figure 4.14.: **AIR Reconstructions**: Full reconstructions for the best AIR model in terms of count accuracy and negative log-likelihood are shown. In contrast to DAIR, AIR reconstructs the individual objects much betters.

5. Conclusion

In this thesis, we have highlighted the benefits associated with unsupervised object-centric learning and developed a new multi-class paradigm in which objects can be understood as prototype modifications. This paradigm comes at the cost of discreteness which is why we provided an in-depth understanding and comparison between different state-of-the-art gradient estimators for discrete random variables. Furthermore, we implemented these estimators with a clean interface as plug-in modules such that future works could easily include or adapt them.

We demonstrated that a discrete VAE (D-VAE) can be understood as a simple multi-class model and showed how this model can be extended into our novel D-VAE-ST approach. Essentially, the D-VAE-ST combines a D-VAE with a spatial transformer to model positional and shape variations of the data. As a result, it features **interpretability** (i.e., prototypes can easily be visualized), **variability** (i.e., there infinite possible images as the prototypes are modified by sampling the transformation from a continuous latent space) and **simplicity** (i.e., the generative model can be summarized in three steps). Furthermore, it enables **one-shot learning** through labeling the learned prototypes.

Our experiments could empirically validate that the D-VAE-ST is consistently better suited for detecting and clustering different classes in the data (compared to a D-VAE). Notably, the D-VAE-ST model could achieve an unsupervised classification accuracy (in which the learned clusters can be visualized as prototype images) of 93.52% for the MNIST dataset in the best case despite its simplicity. Although state-of-the-art approaches like invariant information clustering (Ji et al., 2019) have an even higher unsupervised classification accuracy of about 99.2% on MNIST, our D-VAE-ST approach beats the accuracy of other autoencoder based approaches by a large margin, see table 5.1.

Table 5.1.: **Unsupervised Image Clustering Comparison:** The unsupervised classification accuracy on MNIST is shown for several autoencoder based approaches and a random network. Entries are taken from Ji et al. (2019). Not that in contrast to our method, all the autoencoder based methods do not directly learn a clustering function, therefore require further application of k-means (Zelnik-Manor & Perona, 2005) to be used for image clustering.

Algorithm	MNIST Classification Accuracy [%]
Random network	26.1
Autoencoder (Bengio et al., 2007)	81.2
Sparse autoencoder (Ng et al., 2011)	82.7
Denoising autoencoder (Vincent et al., 2010)	83.2
Variational autoencoder (Kingma & Welling, 2014)	83.2
D-VAE-ST (best)	**93.52**

Lastly, we showed how the D-VAE-ST can be included into AIR (Eslami et al., 2016), an existing structured model that explicitly reasons about the number of objects. We term this adaptation DAIR and compared it against AIR on the MultiMNIST dataset. While the preliminary results for DAIR were rather poor, we expect that some modifications and a different hyperparameter setup would lead to significant improvements.

During our experiments, we observed that our approaches get often trapped into local minima. In future works, modifications to the training process should be studied to increase the robustness of our models. Furthermore, the definition of a prototype could be extended to be more applicable in real-world images, e.g., we could define prototypes as three-dimensional objects.

List of Figures

A. Appendix

A.1. Implementation Details

A.1.1. Toy Experiment

In alignment with the toy experiment performed by Tucker et al. (2017) which was then adopted by Grathwohl et al. (2018), we use the simplest possible network architecture with $\mathbf{x} = 1$ (a one 1×1-vector) and the encoder $\boldsymbol{\theta}$ consisting of one linear layer without any bias that directly predicts the probabilities $\mathbf{p}$ in logits. Similarly, we use a linear layer for the neural baseline in NVIL and a linear layer combined with the concrete relaxation for RELAX. We initialize all weights to 0. Single-sample Monte Carlo estimates are used for all estimators unless otherwise stated.The variance of each gradient estimators is empirically estimated using a Monte-Carlo estimator with a fixed sample size of 1000. We use the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 0.01 for all parameters except for the NVIL network whose learning rate is set to 0.1.

The temperature for the CONCRETE is set to $\lambda = 0.66$ as suggested by Maddison et al. (2017). For REBAR, we set $\eta = 1$ and the trainable temperature hyperparameter in log-units to 0. The neural control variate of RELAX is defined as follows $C_{\boldsymbol{\phi}}(z) = f_{\boldsymbol{\alpha}}\Big(\sigma_\lambda(z)\Big)$, where $\boldsymbol{\alpha}$ denotes the network parameters of the linear layer, λ is the trainable temperature hyperparameter and σ_z denotes the concrete relaxation (tempered softmax) operation.

A.1.2. Single-Object-Multi-Class Experiments

We use a very simplistic setup for the single-object-multi-class experiments: The VAE encoder takes the flattened input image and feeds it through a 3 layer MLP with output sizes $(400, 400, L)$ (L denotes the number of clusters) and ReLU activations in between. Similarly, the VAE decoder consists of a 2 layer MLP with output sizes $(800, I)$ (I denotes the input image dimension, i.e., image channels times image dimension times image dimension) and ReLU activations in between. The output of the decoder is reshaped into the original image shape. We interpret the output as the mean of a Gaussian distribution with fixed variance σ^2 and set this variance to $\sigma^2 = 0.15$. Note that σ^2 is a hyperparameter that inversely scales reconstruction error. We use the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $5 \cdot 10^{-4}$ and weight decay of $1 \cdot 10^{-6}$.

Transformation Estimation Network

For the D-VAE-ST, an additional 3-layer MLP with output sizes $(400, 400, 12)$ and ReLU activations in between is used to estimate the mean and log variance of the affine transformation parameter $\mathbf{t} \in \mathbb{R}^6$. We initialize the mean to the identity transform and the log variance to

$\log 0.1$ by setting the weights to zero and the biases to the formerly mentioned values. This is done to encourage the model to learn useful transformations starting from the identity transform. Furthermore, we do not use an isotropic Gaussian for the latent prior distribution, a Gaussian distribution in which the mean and variance corresponds to the initialized values. The same optimizer that is used for the VAE is used for the transformation estimation network.

NVIL

For the neural baseline of the NVIL gradient estimator, we use a 3-layer MLP with output size $[400, 400, 1]$ and ReLU activations in between. This network is optimized via the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $1 \cdot 10^{-3}$ and weight decay of $1 \cdot 10^{-6}$.

CONCRETE

For the CONCRETE gradient estimator, we set the temperature of the latent distribution to $\lambda_1 = \frac{2}{3}$ and the temperature of the prior distribution to $\lambda_2 = 1$. Using different temperatures is suggested by Maddison et al. (2017).

REBAR

For the REBAR gradient estimator, we set $\eta = 1$ and the trainable temperature in log-units to $\log \lambda = 0$. This parameter is optimized via the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $1 \cdot 10^{-4}$ and weight decay of $1 \cdot 10^{-6}$.

RELAX

For the RELAX gradient estimator, we define the neural control variate as follows $C_{\boldsymbol{\phi}}(z) = f_{\boldsymbol{\alpha}}\Big(\sigma_\lambda(z)\Big)$, where σ_λ denotes the tempered softmax using the trainable temperature λ (initialized to 0 in log units) and $\boldsymbol{\alpha}$ denotes a 3 layer MLP with output-sizes $(400, 400, L)$ (L denotes the number of clusters) and ReLU activations in between.

A.1.3. Multi-Object-Multi-Class Experiment

AIR

We use very simple neural network architectures for the AIR model consisting of multi-layer MLPs: Similar to Eslami et al. (2016), we use a standard recurrent network (no LSTM) that is run for a fixed number of steps and in each step compute the distribution parameters of $z^{(i)}_{\text{pres}}$ and $z^{(i)}_{\text{where}}$. This network has a hidden state dimension of 256 and uses a 2 layer MLP with output sizes $(400, 256 + 7)$ with ReLU activations in between. Furthermore, we use a similar recurrent neural network for the neural baseline (NVIL) with a hidden state dimension of 256 and a 3 layer MLP with output sizes $(256, 256, 1)$ and ReLU activations in between.

The *what-VAE* encoder consists of a 2-layer MLP with output sizes $(400, 2 \cdot |\text{clusters}|)$ which parameterizes the mean and log variance of the Gaussian latent distribution.

The *what-VAE* decoder consists of a 2-layer MLP with output sizes $(400, I)$ (I denotes the input image dimension, i.e., image channels times image dimension times image dimension) and ReLU activations in between. The output of the decoder is reshaped into the input image dimension. We assume a window size of 28 in the spatial transformer such that each input to the *what*-VAE has input shap $1 \times 28 \times 28$. For the VAE we add an additional constraint that the output has to be always positive, i.e., by applying the absolute value operation. This should encourage to learn useful reconstruction, otherwise we could overlay negative and positive images to recreate the final image.

As priors, we use the following priors:

- For the object location, we set $\boldsymbol{\mu}_{\text{where}} = \begin{bmatrix} -3 & 0 & 0 \end{bmatrix}^{\text{T}}$ which leads to a center crop with (approximate) size of inserted digits. Furthermore, we set $\boldsymbol{\sigma}^2_{\text{where}} = \begin{bmatrix} 0.1 * *2 & 1. & 1. \end{bmatrix}$ which further encourages to keep the crop size, but allows for positional variation. We initialize the recurrent neural network to predict these priors.
- For the number of objects, we use a fixed Bernoulli distribution with $p_{\text{pres}} = 0.01$. This encourages the model to use as few objects as possible. To encourage the model to use objects, we initialize the output of the recurrent neural network that predict p_{pres} to 0.8.
- For the latent dimension of the what VAE, we use the standard approach of an isotropic Gaussian.

We assume that the fully reconstructed image $\widetilde{x}$ comes from a Gaussian distribution with fixed variance $\sigma^2 = 0.2$.

For training, we use the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $1 \cdot 10^{-4}$ and a weight decay of $1 \cdot 10^{-6}$ for the network parameters of the *what*-VAE and the recurrent neural network that predicts the distribution parameters. For the neural baseline, we use Adam with a learning rate of $1 \cdot 10^{-2}$ and a weight decay of $1 \cdot 10^{6}$.

DAIR

The DAIR implementation is very similar to the AIR implementation except for some subtle differences: We use discrete latent space in the *what*-VAE (categorical distribution) and a uniform prior on the latent class distribution. Furthermore, an additional localization network is used on the cropped image which is realized through 3-layer MLP with output sizes $(400, 400, 12)$. Similar to the D-VAE-ST experiment, we initialize this network to produce an identity transformation with small variance.

References

Baillargeon, R., Li, J., Ng, W. & Yuan, S. (2009). An account of infants' physical reasoning. *Learning and the infant mind*, 66–116.
17

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., ... others (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.
17

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D. & Kavukcuoglu, K. (2016). Interaction networks for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*.
1

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H. et al. (2007). Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, *19*, 153.
53

Bouguila, N. & ElGuebaly, W. (2009). Discrete data clustering using finite mixture models. *Pattern Recognition*, *42*(1), 33-42.
6

Burgess, C. P., Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M. & Lerchner, A. (2019). Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*.
1, 22, 23, 24

Cohen, G., Afshar, S., Tapson, J. & Van Schaik, A. (2017). EMNIST: Extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 2921–2926).
38

Crawford, E. & Pineau, J. (2019). Spatially Invariant Unsupervised Object Detection with Convolutional Neural Networks. In *Association for the Advancement of Artificial Intelligence* (p. 3412-3420). AAAI Press.
1, 18

Engelcke, M., Kosiorek, A. R., Jones, O. P. & Posner, I. (2019). Genesis: Generative scene inference and sampling with object-centric latent representations. *arXiv preprint arXiv:1907.13052*.
1, 18

Eslami, S. M. A., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Kavukcuoglu, K. & Hinton, G. E. (2016). Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (p. 3225-3233).
1, 9, 18, 19, 20, 21, 22, 49, 50, 54, VI

Grathwohl, W., Choi, D., Wu, Y., Roeder, G. & Duvenaud, D. (2018). Backpropagation through the Void: Optimizing control variates for black-box gradient estimation. In *International Conference on Learning Representations (Poster).* OpenReview.net.
11, 34, 35, III, V

Greff, K., Kaufmann, R. L., Kabra, R., Watters, N., Burgess, C., Zoran, D., ... Lerchner, A. (2019). Multi-Object Representation Learning with Iterative Variational Inference. In K. Chaudhuri & R. Salakhutdinov (Eds.), *International Conference on Machine Learning* (Vol. 97, p. 2424-2433). PMLR.
1, 18

Harnad, S. (1990). The symbol grounding problem. *Physica D*, *40*, 335–346.
17

Higgins, I., Matthey, L., Pal, A., Burgess, C. P., Glorot, X., Botvinick, M., ... Lerchner, A. (2017). beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *International Conference on Learning Representations.*
24

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning* (pp. 448–456).
41

Jaderberg, M., Simonyan, K., Zisserman, A. & Kavukcuoglu, K. (2015). Spatial Transformer Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (p. 2017–2025). Cambridge, MA, USA: MIT Press.
14, 15, 16, I

Jang, E., Gu, S. & Poole, B. (2017, April). Categorical Reparametrization with Gumbel-Softmax. In *Proceedings International Conference on Learning Representations 2017.* OpenReviews.net. Retrieved from https://openreview.net/pdf?id=rkE3y85ee
5, 9

Ji, X., Henriques, J. F. & Vedaldi, A. (2019). Invariant information clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 9865–9874).
53

Kansky, K., Silver, T., Mély, D. A., Eldawy, M., Lázaro-Gredilla, M., Lou, X., ... George, D. (2017). Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *International Conference on Machine Learning* (pp. 1809–1818).
1

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
V, VI, VII

Kingma, D. P. & Welling, M. (2014). Auto-Encoding Variational Bayes. In Y. Bengio & Y. LeCun (Eds.), *International Conference on Learning Representations*.
1, 5, 11, 12, 14, 53

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval research logistics quarterly*, *2*(1-2), 83–97.
40

Lake, B. M., Ullman, T. D., Tenenbaum, J. B. & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, *40*.
17

Lin, Z., Wu, Y.-F., Peri, S. V., Sun, W., Singh, G., Deng, F., ... Ahn, S. (2020). Space: Unsupervised object-oriented scene representation via spatial attention and decomposition. *arXiv preprint arXiv:2001.02407*.
1, 18

Luce, R. D. (1959). *Individual Choice Behavior: A Theoretical analysis*. New York, NY, USA: Wiley.
9

Maddison, C. J., Mnih, A. & Teh, Y. W. (2017). The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations (Poster)*. OpenReview.net.
9, 36, III, V, VI

Mnih, A. & Gregor, K. (2014). Neural Variational Inference and Learning in Belief Networks. *CoRR*, *abs/1402.0030*.
8, 9

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning* (pp. 1928–1937). Retrieved from https://arxiv.org/pdf/1602.01783.pdf
16

Mnih, V., Heess, N., Graves, A. & Kavukcuoglu, K. (2014). Recurrent Models of Visual Attention. In *Advances in Neural Information Processing Systems* (p. 2204-2212).
6

Ng, A. et al. (2011). Sparse autoencoder. *CS294A Lecture notes*, *72*(2011), 1–19.
53

Paisley, J. W., Blei, D. M. & Jordan, M. I. (2012). Variational Bayesian Inference with Stochastic Search. In *International Conference on Machine Learning*. icml.cc / Omnipress.
8

Rezende, D. J., Mohamed, S. & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning* (pp. 1278–1286).
5

Ronneberger, O., Fischer, P. & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (pp. 234–241).
24

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.
3

Scholl, B. J. (2001). Objects and attention: The state of the art. *Cognition*, *80*(1-2), 1–46.
1

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., ... others (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, *588*(7839), 604–609.
16

Schulman, J., Heess, N., Weber, T. & Abbeel, P. (2015). Gradient estimation using stochastic computation graphs. *arXiv preprint arXiv:1506.05254*.
3

Spelke, E. S. & Kinzler, K. D. (2007). Core knowledge. *Developmental science*, *10*(1), 89–96.
1, 17

Spillius, E. B., Milton, J., Garvey, P., Couve, C. & Steiner, D. (2011). *The new dictionary of Kleinian thought*. Taylor & Francis.
27

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.
41

Stephenson, T. A. (2000). *An introduction to Bayesian network theory and usage* (Tech. Rep.). IDIAP.
3

Sun, C., Karlsson, P., Wu, J., Tenenbaum, J. B. & Murphy, K. (2019). Stochastic prediction of multi-agent interactions from partial observations. *arXiv preprint arXiv:1902.09641*.
1

Taylor, M. E. & Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, *10*, 1633-1685.
17

Titsias, M. K. & Lázaro-Gredilla, M. (2015). Local Expectation Gradients for Black Box Variational Inference. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (p. 2638-2646).
8

Tucker, G., Mnih, A., Maddison, C. J., Lawson, D. & Sohl-Dickstein, J. (2017). REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models. In I. Guyon et al. (Eds.), *Advances in Neural Information Processing Systems* (p. 2627-2636).
10, 11, 33, 34, V

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A. & Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, *11*(12).
53

Watters, N., Matthey, L., Bosnjak, M., Burgess, C. P. & Lerchner, A. (2019). Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*.
1

Watters, N., Matthey, L., Burgess, C. P. & Lerchner, A. (2019). Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes. *arXiv preprint arXiv:1901.07017*.
24

Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R. & Tacchetti, A. (2017). Visual interaction networks: Learning a physics simulator from video. *Advances in Neural Information Processing Systems*, *30*, 4539–4547.
1

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3-4), 229–256.
7

Yi, K., Gan, C., Li, Y., Kohli, P., Wu, J., Torralba, A. & Tenenbaum, J. B. (2019). Clevrer: Collision events for video representation and reasoning. *arXiv preprint arXiv:1910.01442*.
1

Yue, Y., Tang, Y., Yin, M. & Zhou, M. (2020). Discrete Action On-Policy Learning with Action-Value Critic. In S. Chiappa & R. Calandra (Eds.), *AISTATS* (Vol. 108, p. 1977-1987). PMLR.
6

Zelnik-Manor, L. & Perona, P. (2005). Self-tuning spectral clustering. *Advances in Neural Information Processing Systems*, *17*.
53