Skolkovo Institute of Science and Technology

Master's Educational Program:

Data Analysis and Management (Computer Science and Engineering)

Multimodal distributions in variational autoencoders

Author: .....
Ekaterina Yakovleva

Certified by .....
Dmitry Vetrov
Associate Professor HSE, PhD
Thesis Supervisor

Certified by .....
Victor Lempitsky
Associate Professor, PhD
Thesis Supervisor

Accepted by: .....

Dean of Education, Skoltech

Moscow

June 2017

Multimodal distributions in variational autoencoders

by

Ekaterina Yakovleva

Submitted to the Skolkovo Institute of Science and Technology
on May, 2017

Abstract

In this thesis, I studied possibility of usage of multimodal distributions in variational autoencoders. Recent breakthroughs in deep learning have resulted in new neural network architectures that allow to solve a wide range of tasks. Among them are autoencoders which are designed to learn latent representation of data, usually of lower dimension. Further improvements of the model include usage of variational inference when assumptions are made regarding distribution of latent variables, and the resulting model is called variational autoencoder (VAE). However, special tricks have to be applied in order to efficiently learn the model, and not the whole range of distributions can be used in VAEs due to this. At the same time more complex assumptions on distribution in latent space allow to obtain richer representation and are able to better describe the data.

This work presents a new VAE with loguniform prior and spike and slab posterior distribution which is a mixture of Dirac Delta function and a Gaussian probability function. Such choice of distributions promotes learning of sparse latent representation and allows to obtain a more complex latent space structure comparing with VAE with just Gaussian posterior. Since ordinary reparametrization trick is not applicable in this case, a special procedure for backpropagation through mixture weights was derived and implemented for efficient learning of the model.

In a series of experiments the model suggested above demonstrated applicability and allowed to obtain sparse representation of data from different datasets. In addition to that, obtained sparsity was expressed both in terms of elimination of the whole dimensions for all objects at the same time and in terms of individual sparsity where different dimensions were eliminated for different objects.

Thesis Supervisor: Dmitry Vetrov
Title: Associate Professor HSE, PhD

Thesis Supervisor: Victor Lempitsky
Title: Associate Professor, PhD

Acknowledgments

I would like to express the gratitude to Dmitry Vetrov for surpervision and for valuable ideas and expertise in the topic of the work. I would also like to thank Roman Klokov for priceless help in solving practical or theoretical issues. Finally, I express gratitude to Ivan Tsybulin for enormous help with acess to MIPT server.

Contents

1 Introduction **10**

1.1 Motivation 11

1.2 Contribution 12

1.3 Outline 12

2 Related work **13**

2.1 Variational autoencoder 13

2.2 VAEs with more complex priors and posteriors 14

2.3 Sparse autoencoders 15

2.4 Stochastic backpropagation through mixture density weights 16

3 Methods **19**

3.1 Stochastic backpropagation through mixture density distributions 19

3.2 Spike-and-slab posterior 21

3.3 Log-scale uniform prior 23

3.4 Variational autoencoder with sparse latent representation 25

4 Experiments **27**

4.1 Sparsity analysis 28

4.1.1 MNIST 28

4.1.2 Omniglot 29

4.1.3 CIFAR-10 30

5 Summary and Discussion **32**

Appendices **37**

A Stochastic backpropagation through mixture weights **38**

A.1 Derivation of Monte-Carlo estimation 38

A.2 Derivation of joint recursion 39

A.3 Algorithm for stochastic backpropagation 40

B Stochastic backpropagation through mixture density distributions **41**

B.1 Derivation of joint recursion 41

B.2 Algorithm for general stochastic backpropagation 43

B.3 KL-divergence approximation 43

C Experimental results **46**

C.1 Network architectures and parameters 46

C.2 Visualization of model performance 46

List of Figures

1-1 Illustration of the work of autoencoders. Model learns low-dimensional representation of data with encoding network, after that this representation is reconstructed with decoding network. Obtained reconstruction then is compared to the original object. 11

4-1 Histogram of active units on MNIST validation set. For each object the number of units for which the weight of a Gaussian component is larger than 0.5 is calculated, and the result over all objects are depicted as a histogram. 28

4-2 Histogram of weights of the Gaussian component on MNIST validation set for different sizes of latent space. As we can see, the model separates the Gaussian component from the delta function component. 29

4-3 Heatmap of unit activity for all digits. Here for each remained feature of latent variable and for each class the proportion of objects for which it is active is calculated and pictured as a shade of gray, with white corresponding to a higher activity level. Features were sorted so that the mean activity among all objects from the validation set descends from left to right. As it can be seen, the total number of active dimensions for model with 200-dimensional latent space is 64. 30

4-4 Histogram of active units on Omniglot validation set. For each object the number of units for which the weight of a Gaussian component is larger than 0.5 is calculated, and the result over all objects are depicted as a histogram. On the left the fully connected architecture was used, and on the right the convolutional one. 31

4-5 Histogram of weights of the Gaussian component on Omniglot validation set. On the left the fully connected architecture was used, and on the right the convolutional one. The difference in the CNN autoencoder may be explained by the fact of mild tuning of the model (to prevent overfitting optimization was stopped early). 31

5-1 Initialization of component weights in sparse VAE. This distribution of weights is returned by the encoder with Glorot initialization before the start of optimization. 34

C-1 Comparison of decoded images from MNIST dataset. First column corresponds to origianl image, second column - to original VAE, third column - to loguniform VAE, and the last column - to sparse VAE. 48

C-2 Comparison of decoded images from Omniglot dataset on the fully connected architecture. First column corresponds to origianl image, second column - to original VAE, third column - to loguniform VAE, and the last column - to sparse VAE. 49

C-3 Comparison of decoded images from Omniglot dataset on the convolutional architecture. First column corresponds to origianl image, second column - to original VAE, third column - to loguniform VAE, and the last column - to sparse VAE. 50

C-4 Comparison of decoded images from CIFAR-10 dataset on the fully connected architecture. First column corresponds to origianl image, second column - to original VAE, third column - to loguniform VAE, and the last column - to sparse VAE. 51

C-5 Comparison of decoded images from CIFAR-10 dataset on the convolutional architecture. First column corresponds to origianl image, second column - to original VAE, third column - to loguniform VAE, and the last column - to sparse VAE. 52

C-6 Activity of latent features for each of the training classes. Activity level was measured by calculation of the proportion of objects from a particular class for which this feature is active. 53

List of Tables

C.1 Neural network architectures for encoder $q_\phi(\mathbf{z}|\mathbf{x})$ 47
C.2 Neural network architectures for decoder $p_\theta(\mathbf{x}|\mathbf{z})$ 47

Chapter 1

Introduction

Over the last few decades a lot of useful tools and algorithms for data analysis have been developed which allows to solve many theoretical and application tasks. However, data that is used nowadays is quite complicated both in terms of structure and size. Large dimensionality rises new challenges and leads to new methods and algorithms for dimensionality reduction and feature extraction (e.g. PCA, t-SNE, linear discriminant analysis). Most common drawbacks of these methods include inability to handle large datasets and difficult data structures which requires a different approach to dimensionality reduction.

Recent breakthroughs in deep learning have resulted in new neural network architectures that, apart from solving many other tasks, can also handle the challenges mentioned above. In particular, we are talking about autoencoder (AE), a model that tries to learn a low-dimensional latent representation of the data (*encoding*) and after that compares restored and initial objects (*decoding*). Further improvements of this approach include variational methods where some assumptions concerning the distribution over latent variables are made which leads us to variational autoencoders (VAE). Graphical models are used to represent probabilistic encoding and decoding stages in this model, and it is assumed that prior and posterior distributions over latent variables have particular form and parametrization. Here rises a new challenge: while allowing effective inference, simplified assumptions regarding distributions lead to poor detection of structure in real data, therefore requiring methods to handle more complicated distributions which are able to describe real data better.

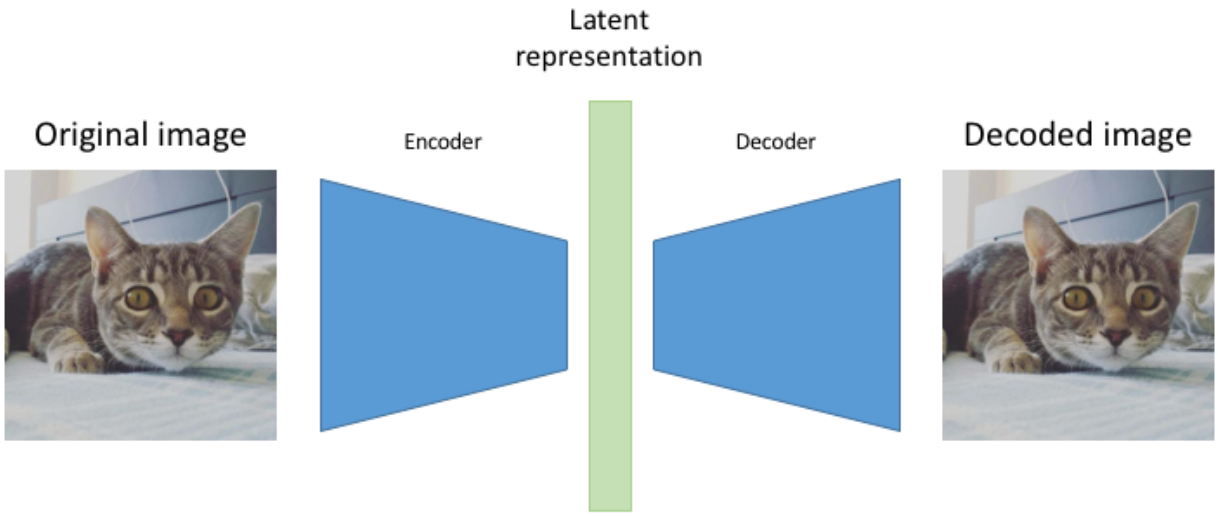

Figure 1-1: Illustration of the work of autoencoders. Model learns low-dimensional representation of data with encoding network, after that this representation is reconstructed with decoding network. Obtained reconstruction then is compared to the original object.

1.1 Motivation

The goal of this work is to obtain and implement a new, sparse VAE with computationally efficient posterior inference assuming that prior and posterior distributions over latent variables have a complicated, probably multimodal structure.

Quality of the learnt latent representation depends directly on its dimensionality d. However, some experiments show that at some point models stop improving their results with the growth of dimensionality of the latent space. A reasonable question arises then: is there a data-specific or architecture-specific optimal dimensionality of the latent representation? Can some dimensions, maybe individual for each object from the dataset, be redundant so that their substitution with zeros will not affect the quality of the model? Learning of sparse representation helps to shed a light on both of these questions and can potentially improve work of the models that use obtained latent representation as input data.

1.2 Contribution

For the purpose of the work an effective technique for backpropagation through parameters in mixture density models was derived. Ability to perform effective inference in such models allows to build more advanced and powerful VAEs which is useful for coding and data representation tasks, as well as for recognition, denoising and visualization purposes.

In particular, a spike-and-slab distribution in the form of the mixture of a Gaussian and a Dirac delta function was introduced into VAE as a posterior. Commonly used as a prior in linear regression models to perform feature selection, here this distribution got a completely new role as a trainable posterior but with the similar purpose - learning of sparse representation.

Finally, an improper log-scale uniform prior, not common in VAEs for certain reasons but having a number of benefits in the context of sparse VAE, was also introduced into the final model.

1.3 Outline

For a better understanding of the topic area and of the state-of-the-art results in development of variational autoencoders related work will be reviewed in **Chapter 2**. Backpropagation technique for mixture models that formed the basis of the final algorithm will also be described in that chapter. After that this technique will be generalized in **Chapter 3** which will allow to introduce there a sparse VAE with mixture density posterior and task-specific prior. **Chapter 4** contains details and experimental results obtained during application of the resulted VAE to different datasets. Finally, in **Chapter 5** summary and future work will be discussed.

Chapter 2

Related work

Autoencoders are artificial neural networks designed for unsupervised learning of representation of the data. Normally these representations have lower number of dimensions than the input objects, thus allowing to perform dimensionality reduction. Architecturally, basic and most simple autoencoder is a feedforward neural net similar to the multilayer perceptron with a bottleneck hidden layer which corresponds to desired representation. It sets the target values to be equal to the inputs and due to the constraint imposed on the hidden layer it learns not the identity function but rather compressed representation of the initial objects.

2.1 Variational autoencoder

Variational autoencoders (VAE) were first introduced by Kingma and Welling in [9] where they consider a generative model $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})$, where $\mathbf{x}$ is observed data, $\mathbf{z}$ is a latent representation and θ is parametrization of the generative model. Thus, posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is regarded as probabilistic *encoder* and likelihood $p_\theta(\mathbf{x}|\mathbf{z})$ is regarded as probabilistic *decoder*. Usually true posterior is intractable requiring some approximation $q_\phi(\mathbf{z}|\mathbf{x})$ where ϕ denotes variational parameters. Posterior inference for object $\mathbf{x}^{(i)}$ is obtained with variational lower bound maximization.

$$\mathcal{L}\left(\theta, \phi; \mathbf{x}^{(i)}\right) = -D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})\right) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left(\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right) \longrightarrow \max_{\theta,\phi} \tag{2.1}$$

Since it is considered that dataset is quite large, stochastic variational inference [6] with stochastic gradient ascend should be used for this task. It can be made especially efficient for continuous latent variables through their reparameterization which results in a highly scalable learning procedure [9, 19].

$$\mathbf{z} \sim g_\phi(\boldsymbol{\epsilon}, \mathbf{x}), \quad \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}), \tag{2.2}$$

here $\boldsymbol{\epsilon}$ is an auxiliary noise variable and $g_\phi(\boldsymbol{\epsilon}, \mathbf{x})$ is its differentiable transformation. SGVB estimator looks as follows:

$$\begin{gathered}\tilde{\mathcal{L}}\left(\theta, \phi; \mathbf{x}^{(i)}\right) = -D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})\right) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}) \\ \mathbf{z}^{(i,l)} \sim g_\phi(\boldsymbol{\epsilon}^{(i,l)}, \mathbf{x}^{(i)}), \quad \boldsymbol{\epsilon}^{(i,l)} \sim p(\boldsymbol{\epsilon})\end{gathered} \tag{2.3}$$

Unfortunately, there are some significant limitations regarding which $q_\phi(\mathbf{z}|\mathbf{x})$ can be considered under such approach, and that leads to impossibility to obtain true posterior for many tasks.

2.2 VAEs with more complex priors and posteriors

In [18] and [11] authors suggest to start with simple-structured $q_\phi(z_0|x)$ and after that apply invertible transformation $f(x, z_0, \phi)$. Resulting distribution for z_1 will then be

$$q_\phi(z_1|x) = q_\phi(z_0|x)\left|\det \frac{\delta f}{\delta z_0}\right|^{-1} \tag{2.4}$$

After application of a series of such transformations we can obtain posterior of a quite general form. Still, this approach has some drawbacks. For example, in [18] quite limiting family of transformations were observed whereas more complicated become computationally difficult (e.g. Langevin flow, Hamiltonian flow). Still in general the flow approach is quite promising and offers a quite powerful instrument to obtain the desired level of complexity for posterior distributions.

Another approach – a more complex prior – was suggested in [2] for the purpose of

unsupervised clusterization. In this graphical model there is a set of three latent variables $\mathbf{w}$, $\mathbf{z}$ and $\mathbf{x}$ that generate observed object $\mathbf{y}$, with one of them being dependent on the other two. Proposed generative model looks as follows:

$$\mathbf{w} \sim \mathcal{N}(0, \mathbf{I}) \tag{2.5}$$

$$\mathbf{z} \sim Mult(\boldsymbol{\pi}) \tag{2.6}$$

$$\mathbf{x}|\mathbf{z}, \mathbf{w} \sim \prod_{k=1}^{K} \mathcal{N}\left(\boldsymbol{\mu}_{z_k}(\mathbf{w}; \beta), diag\left(\boldsymbol{\sigma}^2_{z_k}(\mathbf{w}; \beta)\right)\right)^{z_k} \tag{2.7}$$

$$\mathbf{y}|\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{\mu}(\mathbf{x}; \theta), diag\left(\boldsymbol{\sigma}^2(\mathbf{x}; \theta)\right)\right) \quad \text{or} \quad \mathcal{B}\left(\boldsymbol{\mu}(\mathbf{x}; \theta)\right) \tag{2.8}$$

Training in such model does not require sampling from a discrete distribution and thus uses standard methods for backpropagation. While the prior distribution is a mixture, suggested posterior is from the mean-field variational family and factorises into the product of simple univariate distributions which is still a significant restriction.

2.3 Sparse autoencoders

In [15] a k-sparse autoencoder is suggested where during forward pass activations in the hidden layer $\mathbf{z} = \mathbf{W}^T\mathbf{x} + \mathbf{b}$ corresponding to the learning representation are computed, but only k largest of them are taken into consideration, while others are being set to zero. In this article it is shown that this model is able to learn invariant features. However, drawbacks include necessity to specifically predefine the level of desired sparsity and difficulty in the learning of models with high sparsity (i.e. with low k in the context of this model). In this case the model tends to pick and re-enforce units that were largest during the first epochs, while others are left unadjusted.

In another work [20] a structured VAE with a hierarchy of L sparse latent features is considered. In this model a rectified Gaussian (RG) distribution is proposed for each element $\mathbf{z}^j$ from an hierarchy, both as a prior and approximation to a posterior. This distribution is

defined as follows:

$$\text{If} \quad \epsilon \sim \mathcal{N}(0,1) \quad \text{and} \quad z_i^j = \max(\mu_i^j + \sigma_i^j \epsilon, 0), \quad \text{then} \quad z_i^j \sim \text{RG}(\mu_i^j, \sigma_i^j) \tag{2.9}$$

Suggested VAE was tested on MNIST[14] datatset and showed results comparable but not superior to VAEs with a single layer of latent variables. And since the structure of the latent space is uninterpretable, obtained sparsity is also hard to interpret in terms of latent representation.

2.4 Stochastic backpropagation through mixture density weights

One of the crucial elements of successful learning of variational autoencoders is the ability to effectively backpropagate stochastic gradients through continuous latent distributions. Two major criteria of gradient estimation should be satisfied: unbiasedness and low variance. The "reparameterization trick", a transformation technique that allows to obtain such estimators for many continuous distributions from location-scale family, does not extend to mixture density models and thus not universally applicable. The key difficulty is reparameterization of mixture weights which come from discrete distributions. However, an alternative transform [3] can be applied to any continuous multivariate distribution that is differentiable and can be effectively sampled from. Together with classical reparameterization trick separately applied to each of the mixture components, this technique allows to introduce mixture-distributed latent variables into variational autoencoders.

Consider cumulative density function $F(\mathbf{x})$ and probability density function $f(\mathbf{x})$ over $\mathbf{x} \in \mathbb{R}^D$ which can be rewritten as:

$$f(\mathbf{x}) = \prod_{d=1}^{D} f_d(x_d|\mathbf{x}_{<d}), \tag{2.10}$$

where $\mathbf{x}_{<d} = x_1, \ldots, x_{d-1}$, $f_1(x_1|\mathbf{x}_{<1}) = f_1(x_1)$ and $f_d(x_d|\mathbf{x}_{<d})$ for $d \geq 2$ is the marginal probability density function of x_d conditioned on $\mathbf{x}_{<d}$. Using multivariate quantile transform,

we can recursively obtain samples $\hat{\mathbf{x}}$ from f:

$$\begin{aligned} \mathbf{u} = (u_1, \ldots, u_D) &\sim \mathcal{U}(0,1) \\ \hat{x}_1 &= F_1^{-1}(u_1) \\ \hat{x}_d &= F_d^{-1}(u_d | \hat{\mathbf{x}}_{<d}) \end{aligned} \tag{2.11}$$

Here F_d^{-1} stands for inverse cumulative density function, or quantile function. Thus, the following equation holds place:

$$F_d(\hat{x}_d | \hat{\mathbf{x}}_{<d}) = \int_{t=-\infty}^{\hat{x}_d} f_d(t | \hat{\mathbf{x}}_{<d}) dt = u_d \tag{2.12}$$

Assuming that f depends on some parameter θ, it can be seen that

$$\frac{\partial \hat{x}_d}{\partial \theta} = -\frac{1}{f_d(\hat{x}_d | \hat{\mathbf{x}}_{<d})} \int_{t=-\infty}^{\hat{x}_d} \frac{\partial f_d(t | \hat{\mathbf{x}}_{<d})}{\partial \theta} dt \tag{2.13}$$

In variational autoencoders loss function can be represented as a sum of expectations over posterior distribution of latent variables. Assuming that f is a posterior, Monte-Carlo estimation of the an expectation h over f of an arbitrary differentiable function g of $\mathbf{x}$ looks as follows:

$$h = \int_{\mathbf{x} \in \mathbb{R}^D} g(\mathbf{x}) dF(\mathbf{x}) \tag{2.14}$$

Its partial derivative of parameter θ can be estimated using Monte-Carlo sampling:

$$\frac{\partial h}{\partial \theta} \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{d=1}^{D} \frac{\partial g(\mathbf{x}^n)}{\partial x_d^n} \frac{\partial x_d^n}{\partial \theta}, \quad \mathbf{x}^n \sim f(\mathbf{x}) \tag{2.15}$$

Thus, if we can estimate $\frac{\partial x_d^n}{\partial \theta}$, we can estimate an expectation h as well. Let f be a mixture density distribution with K components:

$$f(\mathbf{x}) = \sum_{k=1}^{K} \pi_k f^k(\mathbf{x}) \tag{2.16}$$

Marginal distributions can be rewritten as

$$f_d(x|\mathbf{x}_{<d}) = \sum_{k=1}^{K} p_d^k f_d^k(x_d), \tag{2.17}$$

where $p_d^k = \Pr(k|\mathbf{x}_{<d})$ is the posterior responsibility of the component k given the prior mixture density weight π_k and observation sequence $\mathbf{x}_{<d}$. It is also assumed that mixture components have diagonal covariance.

Posterior responsibility of the component k is defined by the following recurrent relation:

$$\begin{aligned} p_1^k &= \pi_k \\ p_d^k &= \frac{p_{d-1}^k f_{d-1}^k(x_{d-1})}{f_{d-1}(x_{d-1}|\mathbf{x}_{<d-1})} \end{aligned} \tag{2.18}$$

Using (2.17) and (2.18), we can obtain formulas for joint recurrent estimation of $\frac{\partial x_d^n}{\partial \theta}$:

$$\frac{\partial \hat{x}_d}{\partial \pi_j} = -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \sum_k \frac{\partial \log p_d^k}{\partial \pi_j} p_d^k F_d^k(\hat{x}_d) \tag{2.19}$$

$$\begin{aligned} \frac{\partial \log p_d^k}{\partial \pi_j} = \frac{\partial \log p_{d-1}^k}{\partial \pi_j} - \sum_l p_d^l \frac{\partial \log p_{d-1}^l}{\partial \pi_j} + \\ \left[\frac{\partial \log f_{d-1}^k(\hat{x}_{d-1})}{\partial \hat{x}_{d-1}} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(\hat{x}_{d-1})}{\partial \hat{x}_{d-1}} \right] \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \end{aligned} \tag{2.20}$$

The procedure starts from the initial conditions

$$\begin{aligned} \frac{\partial \log p_1^k}{\partial \pi_j} &= \frac{\delta_{jk}}{\pi_j} \\ \frac{\partial \hat{x}_1}{\partial \pi_j} &= -\frac{F_1^j(\hat{x}_1)}{f_1(\hat{x}_1)} \end{aligned} \tag{2.21}$$

Obtained estimation of $\frac{\partial x_d^n}{\partial \theta}$ can further be used in (2.15) for estimation of the expectation

$$\frac{\partial h}{\partial \pi_j} \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{d=1}^{D} \frac{\partial g(\mathbf{x}^n)}{\partial x_d^n} \frac{\partial x_d^n}{\partial \pi_j}, \quad \mathbf{x}^n \sim f(\mathbf{x}) \tag{2.22}$$

Full algorithm and detailed mathematical derivations can be found in the Appendix A.

Chapter 3

Methods

3.1 Stochastic backpropagation through mixture density distributions

Using approach described in Section 2.4, a more general algorithm for backpropagation through mixture density distributions can be derived. First of all, additional parameters θ that each of the mixture components individually depends on can be introduced. In this case a mixture density distribution f looks as follows:

$$f(\mathbf{x}) = \sum_{k=1}^{K} \pi_k f^k(\mathbf{x}|\theta_k) \tag{3.1}$$

Marginal probability density function of x_d conditioned on $\mathbf{x}_{<d} = x_1, \ldots, x_{d-1}$ now becomes

$$f_d(x_d|\mathbf{x}_{<d}) = \sum_{k=1}^{K} p_d^k f_d^k(x_d|\theta_d^k), \tag{3.2}$$

where $p_d^k = \Pr(k|\mathbf{x}_{<d})$ is the posterior responsibility of the component k given the prior mixture density weight π_k and observation sequence $\mathbf{x}_{<d}$. It is still assumed that mixture components have diagonal covariance, so in this case they depend only on individual parameters θ_d^k.

As in the Section 2.4, p_d^k is defined by the recurrence formula

$$\begin{aligned} p_1^k &= \pi_k \\ p_d^k &= \frac{p_{d-1}^k f_{d-1}^k(x_{d-1}|\theta_{d-1}^k)}{f_{d-1}(x_{d-1}|\mathbf{x}_{<d-1})} \end{aligned} \tag{3.3}$$

However, in this model backpropagation is done not only through mixture weights but also through each of the mixture components parameters θ_d^k. Hence partial derivatives of expectation h (2.14) over posterior need to be calculated with respect to these parameters in addition to (2.22):

$$\frac{\partial h}{\partial \theta_c^j} \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{d=1}^{D} \frac{\partial g(\mathbf{x}^n)}{\partial x_d^n} \frac{\partial x_d^n}{\partial \theta_c^j}, \quad \mathbf{x}^n \sim f(\mathbf{x}) \tag{3.4}$$

Here index c denotes dimensionality, and j denotes the component of the mixture.

Similarly to Section 2.4, for Monte-Carlo estimation of partial derivatives $\frac{\partial h}{\partial \pi_j}$ and $\frac{\partial h}{\partial \theta_c^j}$ we need to obtain estimations $\frac{\partial x_d^n}{\partial \pi_j}$ and $\frac{\partial x_d^n}{\partial \theta_c^j}$ respectively. Derivation of the formulas and algorithm can be found in Appendix B. Final expressions for joint estimation are

$$\frac{\partial \hat{x}_d}{\partial \pi_j} = -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \sum_k \frac{\partial \log p_d^k}{\partial \pi_j} p_d^k F_d^k(\hat{x}_d|\theta_d^k) \tag{3.5}$$

$$\begin{aligned} \frac{\partial \log p_d^k}{\partial \pi_j} &= \frac{\partial \log p_{d-1}^k}{\partial \pi_j} - \sum_l p_d^l \frac{\partial \log p_{d-1}^l}{\partial \pi_j} \\ &+ \left[\frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \hat{x}_{d-1}} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \hat{x}_{d-1}} \right] \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \end{aligned} \tag{3.6}$$

$$\frac{\partial \hat{x}_d}{\partial \theta_c^j} = -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \sum_{k=1}^{K} \left[\frac{\partial \log p_d^k}{\partial \theta_c^j} p_d^k F_d^k(\hat{x}_d|\theta_d^k) + p_d^k \frac{\partial F_d^k(\hat{x}_d|\theta_d^k)}{\partial \theta_d^k} \delta_{cd}^{jk} \right] \tag{3.7}$$

$$\begin{aligned} \frac{\partial \log p_d^k}{\partial \theta_c^j} &= \frac{\partial \log p_{d-1}^k}{\partial \theta_c^j} - \sum_l p_d^l \frac{\partial \log p_{d-1}^l}{\partial \theta_c^j} \\ &+ \left[\frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \hat{x}_{d-1}} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l)}{\partial \hat{x}_{d-1}} \right] \frac{\partial \hat{x}_{d-1}}{\partial \theta_c^j} \\ &+ \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \theta_{d-1}^k} \delta_{c(d-1)}^{jk} - p_d^j \frac{\partial \log f_{d-1}^j(\hat{x}_{d-1}|\theta_{d-1}^j)}{\partial \theta_{d-1}^j} \delta_{c(d-1)} \end{aligned} \tag{3.8}$$

Here δ denotes Kronecker delta function:

$$\begin{cases} \delta^{kl}_{mn} = 1 & \Leftrightarrow m = n \text{ and } k = l \\ \delta^{kl}_{mn} = 0 & \text{otherwise} \end{cases} \tag{3.9}$$

Iterative procedure starts from the initial conditions

$$\frac{\partial \log p_1^k}{\partial \pi_j} = \frac{\delta_{jk}}{\pi_j} \tag{3.10}$$

$$\frac{\partial \hat{x}_1}{\partial \pi_j} = -\frac{F_1^j(\hat{x}_1|\theta_1^j)}{f_1(\hat{x}_1|\theta_1^j)} \tag{3.11}$$

$$\frac{\partial \log p_1^k}{\partial \theta_c^j} = 0 \tag{3.12}$$

$$\frac{\partial \hat{x}_1}{\partial \theta_c^j} = -\frac{p_1^j}{f_1(\hat{x}_1)} \frac{\partial F_1^j(\hat{x}_1|\theta_1^j)}{\partial \theta_1^j} \delta_{1c} \tag{3.13}$$

With this instrument for stochastic backpropagation in mixture density models we can now introduce them into variational autoencoders as posterior distributions. However, new important questions arise: which distributions to use as mixture components and how prior should change in order to reflect desired latent structure proposed by this choice of posterior.

3.2 Spike-and-slab posterior

First introduced as a prior for regression coefficients in [16] for feature selection in linear regression problems, "spike-and-slab" type of distributions essentially represents a mixture of two distributions, probability mass of one of which is concentrated at 0 and the other one is more flatly distributed. Later a number of modified versions of this distribution were suggested ([8, 7]), including generalized spike-and-slab prior [4] which is a mixture of a Gaussian with zero mean and some variance (the slab) and a Dirac delta function centered at the origin (the spike). The latter model was chosen as a basis for the posterior in our autoencoder with the purpose of learning sparse representation of the data. The major difference in use, however, is that in this case parameters of this distribution are learned

from the input data instead of being chosen a priori. Such application of spike-ans-slab distributions becomes possible due to backpropagation technique discussed in Section 3.1 that allows to introduce mixture density distributions into VAE as posterior.

Mathematically this posterior can be defined in the following way. Let $\mathbf{x}$ be an observed sample generated from a latent variable $\mathbf{z}$ by some random process that will be discussed later. Then spike-and-slab approximation $q_\phi(\mathbf{z}|\mathbf{x})$ to the true posterior will be

$$q_\phi(\mathbf{z}|\mathbf{x}) = \prod_d^D \left[\pi_d(\mathbf{x};\phi)\mathcal{N}\left(z_d|\mu_d(\mathbf{x};\phi),\sigma_d^2(\mathbf{x};\phi)\right)\right) + (1-\pi_d(\mathbf{x};\phi))\delta\left(z_d\right)\Big], \tag{3.14}$$

where $\boldsymbol{\pi}(\cdot;\phi)$, $\boldsymbol{\mu}(\cdot;\phi)$ and $\boldsymbol{\sigma}^2(\cdot;\phi)$ are given by neural network with parameter ϕ and $\delta(\cdot)$ is a Dirac delta function

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases} \tag{3.15}$$

Choice of prior over unobserved variable $\mathbf{z}$ will be discussed later. In this model mixture weights form a vector and are independent of each other, meaning that for each object and for each dimensionality individual weights on Gaussian and delta function are assigned. Mixture components are still considered to have diagonal covariance which makes our posterior factorizable in dimension. Thus, recurrent relations (3.3) no longer hold true, and as a result formulas and the algorithm for estimation $\frac{\partial \hat{z}_d}{\partial \pi_c}$, $\frac{\partial \hat{z}_d}{\partial \mu_c^j}$ and $\frac{\partial \hat{z}_d}{\partial \sigma_c^j}$ need to be modified.

$$\frac{\partial \hat{z}_d}{\partial \pi_c} = -\frac{1}{q_{\phi,d}(\hat{z}_d|\mathbf{x})}\left[F_\mathcal{N}\left(\hat{z}_d|\mu_d(\mathbf{x};\phi),\sigma_d^2(\mathbf{x};\phi)\right) - H(\hat{z}_d)\right]\delta_{cd} \tag{3.16}$$

Here $\delta_{(\cdot,\cdot)}$ is a Kronecker delta function

$$\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \tag{3.17}$$

$F_\mathcal{N}(\cdot|\mu,\sigma)$ is a cumulative density function for normal distribution

$$F_\mathcal{N}(x|\mu,\sigma) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sqrt{2\sigma^2}}\right)\right] \tag{3.18}$$

and $H(\cdot)$ is a Heaviside step function

$$H(x) = \begin{cases} 1, & x > 0 \\ \frac{1}{2}, & x = 0 \\ 0, & x < 0 \end{cases} \tag{3.19}$$

Partial derivatives with respect to μ_j and σ_j:

$$\frac{\partial \hat{z}_d}{\partial \mu_c} = -\frac{\pi_d}{q_{\phi,d}(\hat{z}_d|\mathbf{x})} \frac{\partial F_{\mathcal{N},d}(\hat{z}_d|\mu_d, \sigma_d^2)}{\partial \mu_d} \delta_{cd} = \frac{\pi_d}{q_{\phi,d}(\hat{z}_d|\mathbf{x})} \mathcal{N}_d(\hat{z}_d|\mu_d, \sigma_d^2) \delta_{cd} \tag{3.20}$$

$$\frac{\partial \hat{z}_d}{\partial \sigma_c} = -\frac{\pi_d}{q_{\phi,d}(\hat{z}_d|\mathbf{x})} \frac{\partial F_{\mathcal{N},d}(\hat{z}_d|\mu_d, \sigma_d^2)}{\partial \sigma_d} \delta_{cd} = \frac{\pi_d}{q_{\phi,d}(\hat{z}_d|\mathbf{x})} \frac{\hat{z}_d - \mu_d}{\sigma_d} \mathcal{N}_d(\hat{z}_d|\mu_d, \sigma_d^2) \delta_{cd} \tag{3.21}$$

Expressions (3.16), (3.20) and (3.21) are a direct result of (3.5) and (3.7) with $K = 2$ after substitution of p_d^1 with π_d, p_d^2 with $1 - \pi_d$ and θ_c^1 with μ_c and σ_c. There is no θ_c^2 since the second component, delta function, has no parameters. These formulas do not require derivation of any additional expressions and can be vectorized, so with this posterior there is no joint recursion. As a result, derivative (3.4) now has the form

$$\frac{\partial h}{\partial \pi_c} \approx -\frac{1}{N} \sum_{n=1}^{N} \frac{\partial g(\mathbf{z}^n)}{\partial z_c^n} \frac{1}{q_{\phi,d}(\hat{z}_c|\mathbf{x})} \left[F_{\mathcal{N}} \left(\hat{z}_c | \mu_c(\mathbf{x}; \phi), \sigma_c^2(\mathbf{x}; \phi) \right) - H(\hat{z}_c) \right] \tag{3.22}$$

$$\frac{\partial h}{\partial \mu_c} \approx \frac{1}{N} \sum_{n=1}^{N} \frac{\partial g(\mathbf{z}^n)}{\partial z_c^n} \frac{\pi_c}{q_{\phi,c}(z_c^n|\mathbf{x})} \mathcal{N}_c(z_c^n|\mu_c, \sigma_c^2) \tag{3.23}$$

$$\frac{\partial h}{\partial \sigma_c} \approx \frac{1}{N} \sum_{n=1}^{N} \frac{\partial g(\mathbf{z}^n)}{\partial z_c^n} \frac{\pi_c}{q_{\phi,c}(z_c^n|\mathbf{x})} \frac{z_c^n - \mu_c}{\sigma_c} \mathcal{N}_c(z_c|\mu_c, \sigma_c^2) \tag{3.24}$$

$$\mathbf{z}^n \sim q_\phi(\mathbf{z}|\mathbf{x}) \tag{3.25}$$

3.3 Log-scale uniform prior

Choice of prior over latent variables $\mathbf{z}$ directly affects the representation learnt by VAE. Isotropic multivariate Gaussian, commonly used in regular VAEs, allows to obtain structured

and interpretable representation but at the same time does not allow it to be complicated by limiting it to only be unimodal. At the same time more complex priors help to obtain richer representations.

In this work improper log-scale distribution is suggested as a prior over latent variable. It is useful in cases when little is known about the underlying distribution of a random variable.

$$p\left(\log|z_d|\right) = \text{const} \quad \Leftrightarrow \quad p\left(|z_d|\right) \propto \frac{1}{|z_d|} \tag{3.26}$$

In addition to that, in [11] it was explained that this prior has a connection with floating point format for storing numbers. In particular, KL divergence with this prior has an interpretation of a regularizer of the number of significant digits under posterior $q_{\phi,d}(z_d)$ of corresponding floating-point number z_d:

$$-D_{\mathrm{KL}}\left(q_{\phi,d}(z_d)||p(z)\right) = \mathcal{H}\left(q_{\phi,d}(\log|z_d|)\right) + \text{const} \tag{3.27}$$

Log-scale uniform prior is an improper prior, so $D_{\mathrm{KL}}(q_{\phi,d}(z_d|\mathbf{x})||p(z_d))$ can only be computed up to an additive constant C. In addition to that, it cannot be computed analytically. However, in [17] a quite accurate approximation for KL diversion with this prior was suggested in case where posterior is a Gaussian. For this approximation a new parameter α is introduced:

$$\alpha = \frac{\sigma^2}{\mu^2}, \tag{3.28}$$

where μ and σ are parameters of the Gaussian. KL diversion term then can be approximated by

$$D_{\mathrm{KL}}\left(q_\phi(z|\mu,\sigma^2)||p(z_d)\right) \approx -0.64\sigma(1.5(1.3+\log\alpha_d)) + 0.5\log(1+\alpha_d^{-1}) + C \tag{3.29}$$

Here $\sigma(\cdot)$ denotes sigmoid function. As it can be seen, (3.29) is minimized and approaches a constant when $\alpha \longrightarrow +\infty$.

Our posterior $q_\phi(\mathbf{z}|\mathbf{x})$ can be represented as a mixture of two Gaussians, one in regular form with trainable μ_d and σ_d and the other one in fixed form with very small, almost zero

mean $\mu_0 \approx 0$ and very small variance σ_0, with $\mu_0 \ll \sigma_0$ and common for all dimensions. With such parametrization and considering $\alpha_0 = \frac{\sigma_0^2}{\mu_0^2}$ and with an assumption that components do not overlap, we can rewrite (3.29) in the following way:

$$\begin{aligned} -D_{\mathrm{KL}}\left(q_{\phi,d}(z_d|\pi_d,\mu_d,\sigma_d^2)||p(z_d)\right) \approx \pi_d\left[0.64\sigma(1.5(1.3+\log\alpha_d)) - 0.5\log(1+\alpha_d^{-1}) + C_1\right] \\ +(1-\pi_d)\left[0.64\sigma(1.5(1.3+\log\alpha_0)) - 0.5\log(1+\alpha_0^{-1}) + C_0\right] \end{aligned} \tag{3.30}$$

Detailed explanation and justification of this formula is in Section B.3.

In sparse VAE, which is our goal, some dimensions of latent variable $\mathbf{z}$ are turned off by equating to zero, either individually for each object or in general for the whole data set. It is achieved when these dimensions have point probability mass at zero. It corresponds to maximization of α when $\mu \to 0$, $\sigma \to 0$ and $\frac{\sigma}{\mu} \gg 1$. Thus, it seems natural to take constant in (3.29) as a zero-level so that KL-divergence goes to zero when sparsity is gained, so $C_0 = C_1 = -0.64$. By default $\alpha_0 \gg 1$ which promotes learning of sparse representation as well. In fact, since μ_0 can not be considered strictly equal to zero for numerical issues, by changing ratio of σ_0 to μ_0 we can influence the sparsity level that we want to obtain. In particular, the greater is α_0, the closer to zero is the second term in (3.30) which promotes π_d to be zero and puts bigger weight on delta function component.

3.4 Variational autoencoder with sparse latent representation

In this section a proposed model of sparse VAE is described. Consider the generative model $p_\theta(\mathbf{x},\mathbf{z}) = p(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z})$, where observed samples $\mathbf{x}$ are generated from unobserved variables $\mathbf{z}$ by the following process:

$$\log|\mathbf{z}| \sim \text{const} \tag{3.31}$$

$$\mathbf{x}|\mathbf{z} \sim \begin{cases} \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}(\mathbf{z};\theta), diag[\boldsymbol{\sigma}^2(\mathbf{z};\theta)]\right), & \text{if X is continuous} \\ \mathcal{B}\left(\boldsymbol{\mu}(\mathbf{x};\theta)\right), & \text{if X is discrete} \end{cases} \tag{3.32}$$

Proxy to the posterior is

$$q_\phi(\mathbf{z}|\mathbf{x}) = \prod_d^D \left[\pi_d(\mathbf{x};\phi)\mathcal{N}\left(z_d|\mu_d(\mathbf{x};\phi),\sigma_d^2(\mathbf{x};\phi)\right) + (1-\pi_d(\mathbf{x};\phi))\delta\left(z_d\right)\right] \tag{3.14}$$

Variational lower bound estimator for an object $\mathbf{x}^{(i)}$ can be written as

$$\begin{aligned}\mathcal{L}_{ELBO}\left(\theta,\phi,\mathbf{x}^{(i)}\right) = \frac{1}{L}\sum_{l=1}^{L}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})\right] \\ -\sum_{d=1}^{D}\pi_d(\mathbf{x}^{(i)};\phi)\left[0.64\sigma(1.5(1.3+\log\alpha_d(\mathbf{x}^{(i)};\phi))) - 0.5\log(1+\alpha_d^{-1}(\mathbf{x}^{(i)};\phi)) + 0.64\right] \\ -\sum_{d=1}^{D}(1-\pi_d(\mathbf{x}^{(i)};\phi))\left[0.64\sigma(1.5(1.3+\log\alpha_0)) - 0.5\log(1+\alpha_0^{-1}) + 0.64\right],\end{aligned} \tag{3.33}$$

where $\mathbf{z}^{(i,l)} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$. For the full dataset $\mathbf{X}$ lower bound estimator is

$$\mathcal{L}_{\text{ELBO}}(\theta,\phi,\mathbf{X}) = \sum_{i=1}^{N}\mathcal{L}_{ELBO}\left(\theta,\phi,\mathbf{x}^{(i)}\right) \tag{3.34}$$

During training backpropagation is done by straightforward differentiation of the second and the third terms in (3.33) with respect to parameters π_d, μ_d, σ_d and by applying (3.22), (3.23) and (3.24) to the first term in (3.33).

Therefore, in this section a new model of variational autoencoder (we will refer to it as sparse VAE) was obtained. This model incorporates the use of loguniform and spike-and-slab distributions as a prior and a posterior. For this case a special optimization procedure capable of updating the weights of components and the parameters of the Gaussian component in the posterior was derived and implemented in code. The evaluation of the obtained model is presented below.

Chapter 4

Experiments

Model of sparse VAE proposed in Section 3.4 was tested on three datasets: MNIST[14], Omniglot[13] and CIFAR-10[12].

- **MNIST** is a handwritten digits dataset, it consists of $60,000$ training and $10,000$ test objects, each of which is a 28×28 grayscale image.

- **Omniglot** dataset consists of 1623 different handwritten characters from 50 different alphabets. Each character is represented by 20 samples of a 105×105 image. Used primarily for one-shot learning[13], this dataset is originally split into a train set of 30 alphabets and a test set of remaining 20 alphabets, meaning that the latter contains objects from classes that are not seen during training. Such split requires use of models that were developed to work specifically with such type of data, and this was not the purpose of this work. With that being said, a new split was performed so that train set contains 15 examples for each character from each of 50 alphabets, and remaining 5 samples are put into the test set.

- **CIFAR-10** contains 50,000 train samples and 10,000 test samples of 32×32 colour images, with equal number of examples from each of 10 classes, both in train and test sets.

On each of them proposed model (we will refer to it as "sparse VAE"), was compared to VAE with log-scale uniform prior and Gaussian posterior(will be referred to as "loguniform

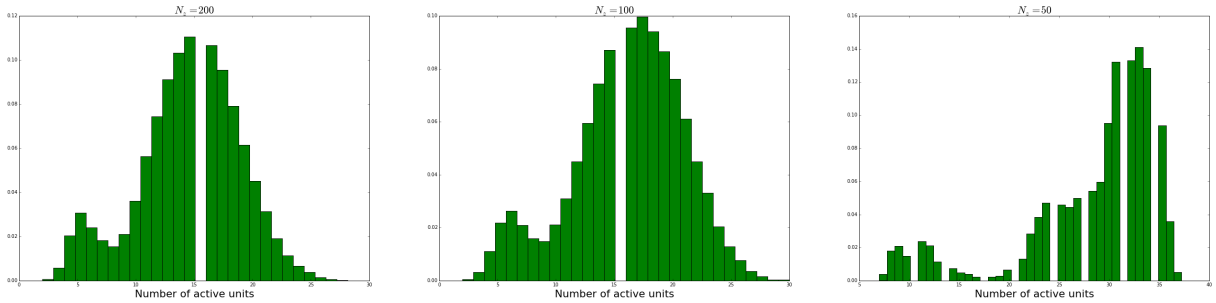

Figure 4-1: Histogram of active units on MNIST validation set. For each object the number of units for which the weight of a Gaussian component is larger than 0.5 is calculated, and the result over all objects are depicted as a histogram.

prior VAE") and to regular VAE proposed in [9] (will be referred as "original VAE") in terms of decoded images, since numerical comparison is inaccurate due to lack of knowledge about the true value of the constant in (3.27). Architectures of neural networks and other training details can be found in Section C.1, results of the experiments are described below.

4.1 Sparsity analysis

4.1.1 MNIST

For experiments on this dataset a fully connected architecture described in Section C.1 was used. Sparse VAE was tested on the same architecture with dimensionality of latent space 200, 100 and 50. In each case the model learned both total sparsity, when a number of dimensions become silent for all objects, and individual sparsity, when a set of active dimensions changes from object to object. Total number of active dimensions depends on the model (64 for $N_z = 200$, 45 for $N_z = 100$ and 41 for $N_z = 50$), however, no singular object has those dimensions active all together. Maximum number of of active units for a singular object is 30 for $N_z = 200$, 30 for $N_z = 100$ and 45 for $N_z = 50$. Overall distribution of active dimensions is concentrated around 15-20 dimensions. One can find these distributions in Figure 4-1.

Distributions on weights for all objects is depicted in Figure 4-2. Interestingly, all learned models are confident in their choice of Gaussian or delta-function, meaning that all weights are equal either 1 or 0. This outcome is favorable, since it frees us from the necessity to

choose any threshold.

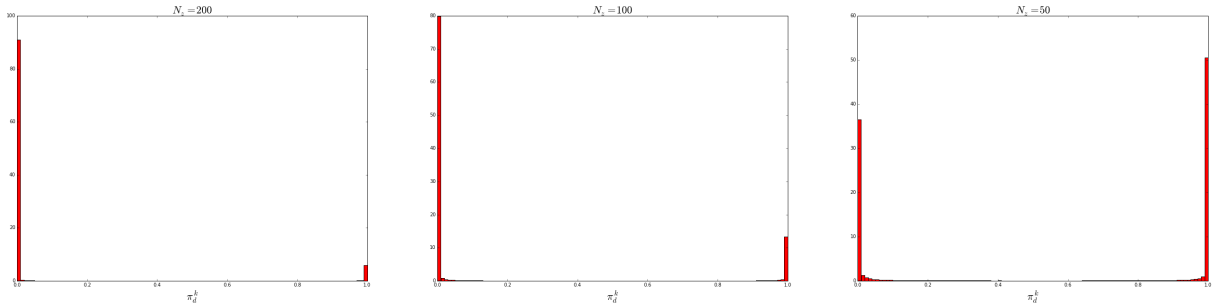

Figure 4-2: Histogram of weights of the Gaussian component on MNIST validation set for different sizes of latent space. As we can see, the model separates the Gaussian component from the delta function component.

As for the learned latent units, their activity also has a structure that depends on the class of the object. In Figure C-6 one can find activity of each of the remained units with respect to a digit for a sparse VAE with $N_z = 200$. Another illustration of this activity is depicted with the use of heatmap Figure 4-3. It can be seen that while there are a couple dimensions that are active for majority of classes, most of them are only active for certain digits, and all digits have individual dimensions for which their activity is at the highest.

4.1.2 Omniglot

Fully connected architecture with 200-dimensional latent space learned a representation with total sparsity level of 111, meaning that only 89 dimensions were active in the encoded objects. Unlike the results on MNIST, remained dimensions were active for all objects. As in the previous case, all obtained weights are either 1 or 0, meaning that the model separates the Gaussian component from the second one.On convolutional architecture the result turned out to be similar to those on MNIST. Instead of learning total sparsity, the model learned individual sparsity when different dimensions are active for different objects. In addition to that, dimensions have a different rate of activity, meaning that there are dimensions that are active for all objects, and there are those that are active for only 30% of objects. As we can see in Figure 4-4, majority of objects have around 75-80 active dimensions, and for no object all latent units remain active. Histogram of weights of the Gaussian component

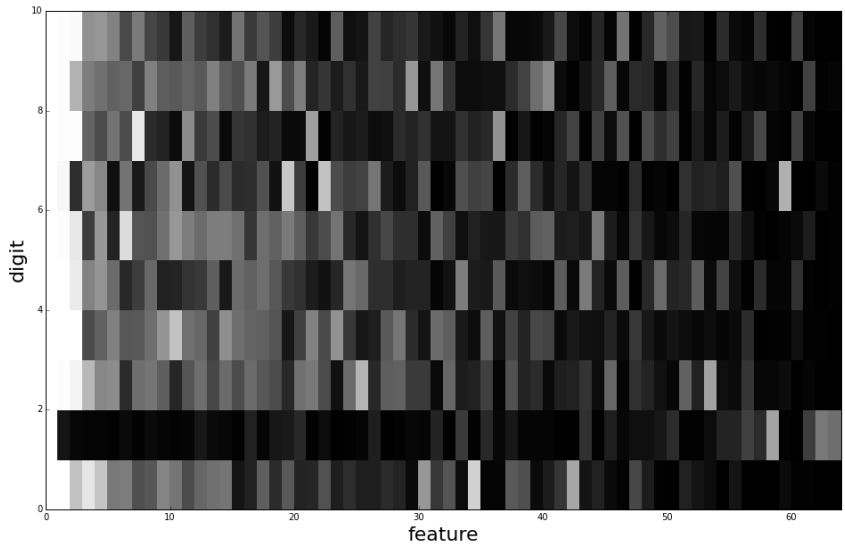

Figure 4-3: Heatmap of unit activity for all digits. Here for each remained feature of latent variable and for each class the proportion of objects for which it is active is calculated and pictured as a shade of gray, with white corresponding to a higher activity level. Features were sorted so that the mean activity among all objects from the validation set descends from left to right. As it can be seen, the total number of active dimensions for model with 200-dimensional latent space is 64.

for all objects form the validation is pictured in Figure 4-5. As we can see, here the model is not absolutely sure about its choice of the component. It can be explained by the fact the optimization of the model was stopped early to prevent overfitting (the dataset contains small number of training objects).

4.1.3 CIFAR-10

Model with fully connected architecture and with 200-dimensional latent variable was able to learn a representation where only 118 units were active, and those active units were the same for all objects. On convolutional architecture active dimensionality was 111. Similarly to the first architecture, all of the left dimensions remained active for all objects. In both cases sparse VAE remains confident about the choice of component and separates a Gaussian from the delta function quite well, with weights being equal to either 1 or 0. Comparison of decoded images in Figure C-4, Figure C-5 shows that images from sparse VAE are smoother than from the other two VAEs, which might be a result of these two overfitting on the data.

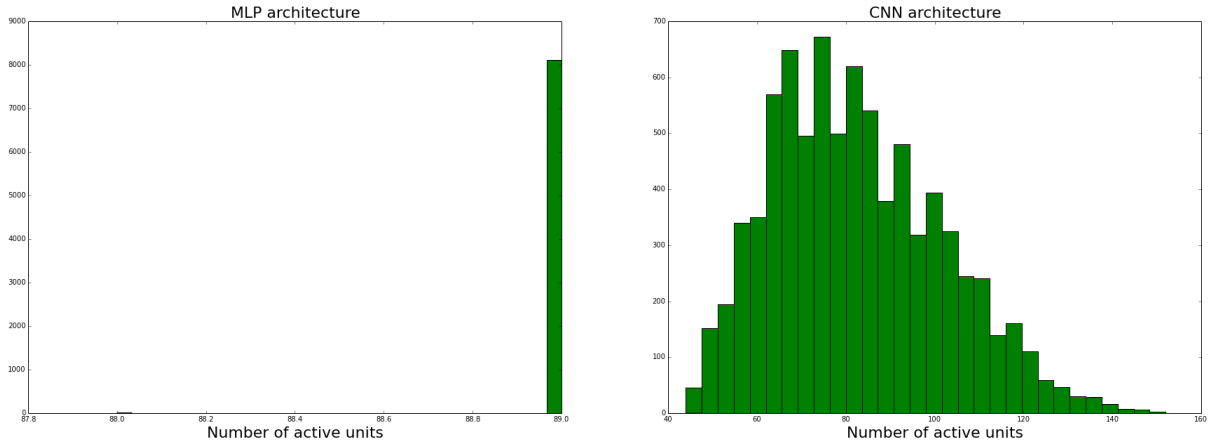

Figure 4-4: Histogram of active units on Omniglot validation set. For each object the number of units for which the weight of a Gaussian component is larger than 0.5 is calculated, and the result over all objects are depicted as a histogram. On the left the fully connected architecture was used, and on the right the convolutional one.

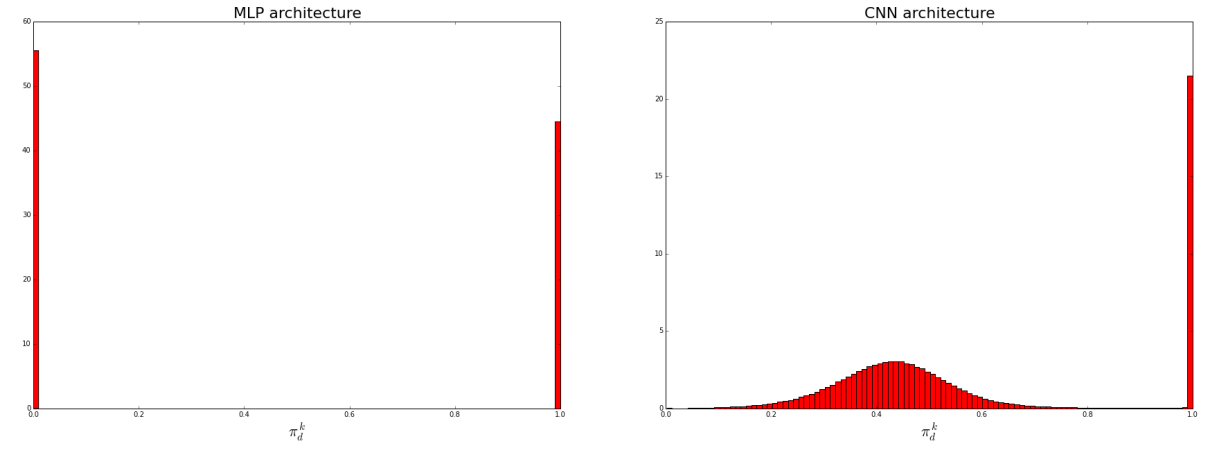

Figure 4-5: Histogram of weights of the Gaussian component on Omniglot validation set. On the left the fully connected architecture was used, and on the right the convolutional one. The difference in the CNN autoencoder may be explained by the fact of mild tuning of the model (to prevent overfitting optimization was stopped early).

Chapter 5

Summary and Discussion

In this work a model of sparse variational autoencoder was proposed. For this purpose a spike-and-slab mixture posterior and a log-scale uniform prior over latent variable were used. Such assumptions on latent space, while allowing to obtain desired structure of the learned latent representation, led to a serious adjustment of the model and required new methods for backpropagation. With some additional assumptions on the form of the mixture posterior, a viable approximation of variational lower bound was used, and a technique for backpropagation through mixture models was adjusted and implemented. Conducted experiments demonstrated efficiency of the model and its ability to learn sparse representation without the loss of visual quality compared to other VAEs.

Obtained results revealed that the desired feature of the model – separation of the Gaussian component from the delta function component – was obtained, making our assumptions about the mixture posterior discussed in Section B.3 consistent with the results. In particular, all obtained component weights are either zeros or ones, which makes the term $H(\pi_d^k)$ equal to zero. Nevertheless, introduction of the entropy term discussed in the Section B.3 is a topic for discussion and will be studied in the future, as it can be used to define the desired sparsity of the model.

Another interesting aspect of the obtained results is that the sparsity level on MNIST is consistent with the results demonstrated in [9]. In this article the authors showed that lower bound on this dataset stopped to noticeably improve with the dimensionality of latent space greater than 20. Our model also learned the representation where majority of objects

have 15-20 active units. What is important, the model was able to define the necessary dimensionality itself, without direct prior definition which is done in majority of works on the similar topic.

One more topic for discussion is initialization of weights of the mixture components. Glorot initialization of network weights led to weight initialization shown in the Figure 5-1 with a peak at 0.5. While on MNIST it was clear that dimensionality 200 of the latent space was more than 2 times enough for obtaining a good representation on this architecture, on Omniglot or CIFAR-10 datasets there was no prior knowledge about required dimensionality. Models learned during experiments on CIFAR-10 with $N_z = 400$ had active units in the range 150-166. The same result was seen after experiments with component weights initializaed with 1 and with latent dimensionality 200. This in combination with results discussed in Section 4.1.3 implies that our initialization might not be able to effectively learn far more than half of active latent units, meaning that it requires using dimensionality of latent space at least 2 times larger than required. Results on MNIST with $N_z = 50$ also indirectly imply this, since they were were slightly worse than in two other cases.

Final topic left for future work is the analysis of whether the learned dimensionality is data-related or architecture-related. Early tests on CIFAR-10 on deliberately weakened architecture showed that the model learns too low dimensionality, approximately 40. This implies that architecture quality directly affects the sparsity of latent representation. Research of the possibility to use the model for architecture estimation is left for future discussion.

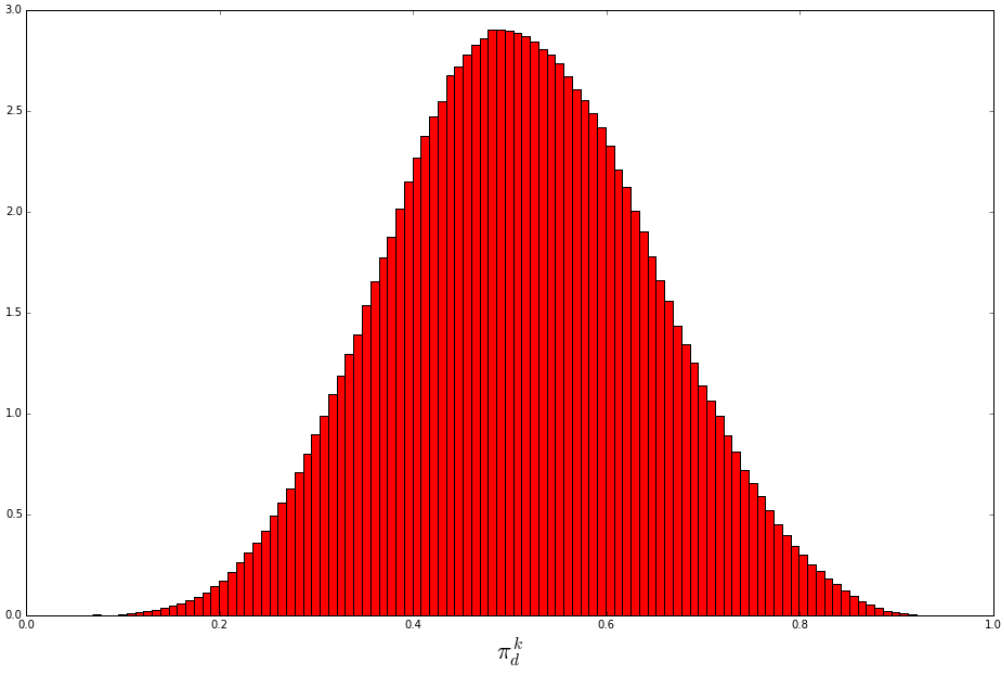

Figure 5-1: Initialization of component weights in sparse VAE. This distribution of weights is returned by the encoder with Glorot initialization before the start of optimization.

Bibliography

[1] Sander Dieleman, Jan Schlter, Colin Raffel, Eben Olson, et al. Lasagne: First release., August 2015.

[2] Nat Dilokthanakul, Pedro A. M. Mediano, Marta Garnelo, Matthew C. H. Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *CoRR*, abs/1611.02648, 2016.

[3] Alex Graves. Stochastic backpropagation through mixture density distributions. *CoRR*, abs/1607.05690, 2016.

[4] Daniel Hernández-Lobato, José Miguel Hernández-Lobato, and Pierre Dupont. Generalized spike-and-slab priors for bayesian group feature selection using expectation propagation. *Journal of Machine Learning Research*, 14(1):1891–1945, 2013.

[5] Irina Higgins, Loic Matthey, Xavier Glorot, Arka Pal, Benigno Uria, Charles Blundell, Shakir Mohamed, and Alexander Lerchner. Early visual concept learning with unsupervised deep learning. *arXiv preprint arXiv:1606.05579*, 2016.

[6] Matthew D Hoffman, David M Blei, Chong Wang, and John William Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[7] H Ishwaran and JS Rao. Bayesian nonparametric mcmc for large variable selection problems. *Unpublished manuscript*, 2000.

[8] Hemant Ishwaran and J Sunil Rao. Spike and slab variable selection: frequentist and bayesian strategies. *Annals of Statistics*, pages 730–773, 2005.

[9] D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013.

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[12] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[13] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[15] Alireza Makhzani and Brendan Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

[16] Toby J Mitchell and John J Beauchamp. Bayesian variable selection in linear regression. *Journal of the American Statistical Association*, 83(404):1023–1032, 1988.

[17] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

[18] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[19] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[20] Tim Salimans. A structured variational auto-encoder for learning deep hierarchies of sparse features. *arXiv preprint arXiv:1602.08734*, 2016.

[21] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

Appendices

Appendix A

Stochastic backpropagation through mixture weights

In this section detailed derivation of formulas from [3] is reproduced.

A.1 Derivation of Monte-Carlo estimation

Let $Q(\mathbf{u})$ be the sample from f obtained by quantile transform of $\mathbf{u}$. Then expectation h (2.14) can be rewritten as

$$h = \int_{\mathbf{u}\in[0,1]^D} g(Q(\mathbf{u}))d\mathbf{u} \tag{A.1}$$

Thus, partial derivative of parameter θ now looks as follows:

$$\frac{\partial h}{\partial \theta} = \int_{\mathbf{u}\in[0,1]^D} \frac{\partial g(Q(\mathbf{u}))}{\partial \theta} d\mathbf{u} = \int_{\mathbf{u}\in[0,1]^D} \sum_{d=1}^{D} \frac{\partial g(Q(\mathbf{u}))}{\partial Q_d(\mathbf{u})} \frac{\partial Q_d(\mathbf{u})}{\partial \theta} d\mathbf{u} \tag{A.2}$$

Monte-Carlo estimation of this derivative is (2.15). Important aspect of this estimation is that it does not require the use of inverse cumulative density function, even though $\hat{\mathbf{x}}$ is considered to be obtained via quantile transform.

A.2 Derivation of joint recursion

In order to estimate derivatives of h with respect to mixture weights π_j, we need to estimate $\frac{\partial x_d^n}{\partial \pi_j}$. Using (2.17) in integral part of (2.13) and differentiating it yields

$$\frac{\partial f_d(t|\mathbf{x}_{<d})}{\partial \pi_j} = \sum_k \left[\frac{\partial p_d^k}{\partial \pi_j} f_d^k(t) + \frac{\partial f_d^k(t)}{\partial \pi_j} \frac{\partial t}{\partial \pi_j} p_d^k \right] = \sum_k \frac{\partial p_d^k}{\partial \pi_j} f_d^k(t), \tag{A.3}$$

since t does not depend on f and $\frac{\partial t}{\partial \pi_j} = 0$. After substitution of the relation (A.3) and samples $\hat{x}$ into (2.13) we get

$$\frac{\hat{x}_d}{\partial \pi_j} = -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \sum_k \frac{\partial p_d^k}{\partial \pi_j} \int_{t=-\infty}^{\hat{x}_d} f_d^k(t) dt \tag{A.4}$$

$$= -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \sum_k \frac{\partial \log p_d^k}{\partial \pi_j} p_d^k F_d^k(\hat{x}_d) \tag{A.5}$$

The only unknown part here is $\frac{\partial \log p_d^k}{\partial \pi_j}$. It can be obtained by differentiation of (2.18):

$$\frac{\partial \log p_d^k}{\partial \pi_j} = \frac{\partial}{\partial \pi_j} \left[\log p_{d-1}^k + \log f_{d-1}^k(\hat{x}_{d-1}) - \log \left(\sum_k p_{d-1}^k f_{d-1}^k(\hat{x}_{d-1}) \right) \right] \tag{A.6}$$

$$= \frac{\partial p_{d-1}^k}{\partial \pi_j} + \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1})}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} - \frac{1}{\sum_{l'} p_{d-1}^{l'} f_{d-1}^{l'}(\hat{x}_{d-1})} \sum_l f_{d-1}^l(\hat{x}_{d-1}) \frac{\partial p_{d-1}^l}{\partial \pi_j} \tag{A.7}$$

$$- \frac{1}{\sum_{l'} p_{d-1}^{l'} f_{d-1}^{l'}(\hat{x}_{d-1})} \sum_l p_{d-1}^l \frac{\partial f_{d-1}^l(\hat{x}_{d-1})}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \tag{A.8}$$

$$= \frac{\partial p_{d-1}^k}{\partial \pi_j} + \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1})}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} - \sum_l \frac{f_{d-1}^l(\hat{x}_{d-1})}{\sum_{l'} p_{d-1}^{l'} f_{d-1}^{l'}(\hat{x}_{d-1})} \frac{\partial p_{d-1}^l}{\partial \pi_j} \tag{A.9}$$

$$- \sum_l \frac{p_{d-1}^m}{\sum_{l'} p_{d-1}^{l'} f_{d-1}^{l'}(\hat{x}_{d-1})} \frac{\partial f_{d-1}^l(\hat{x}_{d-1})}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \tag{A.10}$$

$$= \frac{\partial p_{d-1}^k}{\partial \pi_j} + \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1})}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} - \sum_l \frac{p_d^l}{p_{d-1}^l} \frac{\partial p_{d-1}^l}{\partial \pi_j} \tag{A.11}$$

$$- \sum_l \frac{p_d^l}{f_{d-1}^l(\hat{x}_{d-1})} \frac{\partial f_{d-1}^l(\hat{x}_{d-1})}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \tag{A.12}$$

$$= \frac{\partial p_{d-1}^k}{\partial \pi_j} + \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1})}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} - \sum_l p_d^l \frac{\partial \log p_{d-1}^l}{\partial \pi_j} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(\hat{x}_{d-1})}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \tag{A.13}$$

After rearrangement we get the relation (2.20).

A.3 Algorithm for stochastic backpropagation

Pseudocode for complete computation is described in Algorithm 1:

Algorithm 1 Stochastic backpropagation through mixture density weights

initialize $\frac{\partial h}{\partial \pi_j} \leftarrow 0$,
for $n = 1$ to N **do**
  draw $\mathbf{x} \sim f(\mathbf{x})$
  $p_1^k \leftarrow \pi_k$
  $f_1(x_1) \leftarrow \sum_k \pi_k f_1^k(x_1)$
  $\frac{\partial p_1^k}{\partial \pi_j} \leftarrow -\frac{\delta_{jk}}{\pi_j}$
  $\frac{\partial \log p_1^k}{\partial \theta_c^j} \leftarrow 0$
  for $d = 1$ to D **do**
    $f_d(x_d|\mathbf{x}_{<d}) \leftarrow \sum_k p_d^k f_d^k(x_d)$
    $p_d^k \leftarrow \frac{p_{d-1}^k f_{d-1}^k(x_{d-1})}{f_{d-1}(x_{d-1}|\mathbf{x}_{<d-1})}$
    $\frac{\partial \log p_d^k}{\partial \pi_j} \leftarrow \frac{\partial \log p_{d-1}^k}{\partial \pi_j} - \sum_l p_d^l \frac{\partial \log p_{d-1}^l}{\partial \pi_j} +$
      $\left[\frac{\partial \log f_{d-1}^k(x_{d-1})}{\partial x_{d-1}} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(x_{d-1})}{\partial x_{d-1}}\right] \frac{\partial x_{d-1}}{\partial \pi_j}$
    $\frac{\partial \hat{x}_d}{\partial \pi_j} \leftarrow -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \sum_{k=1}^K \frac{\partial \log p_d^k}{\partial \pi_j} p_d^k F_d^k(x_d)$
  end for
  $\frac{\partial h}{\partial \pi_j} \leftarrow \frac{\partial h}{\partial \pi_j} + \sum_d \frac{\partial g(\mathbf{x})}{\partial x_d} \frac{\partial x_d}{\partial \pi_j}$
end for
$\frac{\partial h}{\partial \pi_j} \leftarrow \frac{1}{N} \frac{\partial h}{\partial \pi_j}$

Appendix B

Stochastic backpropagation through mixture density distributions

B.1 Derivation of joint recursion

The logic behind derivation of formulas remains the same as in Section 2.4 and Section A.2: we need to differentiate (3.2) with respect to π_j and θ_c^j and use it as integral part of expression (2.13).

$$\frac{\partial f_d(t|\mathbf{x}_{<d})}{\partial \pi_j} = \sum_k \left[\frac{\partial p_d^k}{\partial \pi_j} f_d^k(t|\theta_d^k) + \frac{\partial f_d^k(t|\theta_d^k)}{\partial \pi_j} \frac{\partial t}{\partial \pi_j} p_d^k \right] = \sum_k \frac{\partial p_d^k}{\partial \pi_j} f_d^k(t|\theta_d^k) \tag{B.1}$$

$$\begin{aligned} \frac{\partial f_d(t|\mathbf{x}_{<d})}{\partial \theta_c^j} &= \sum_{k=1}^{K} \left[\frac{\partial p_d^k}{\partial \theta_c^j} f_d^k(t|\theta_d^k) + p_d^k \frac{\partial f_d^k(t|\theta_d^k)}{\partial t} \frac{\partial t}{\partial \theta_c^j} + p_d^k \frac{\partial f_d^k(t|\theta_d^k)}{\partial \theta_d^k} \frac{\partial \theta_d^k}{\partial \theta_c^j} \right] \\ &= \sum_{k=1}^{K} \left[\frac{\partial p_d^k}{\partial \theta_c^j} f_d^k(t|\theta_d^k) + p_d^k \frac{\partial f_d^k(t|\theta_d^k)}{\partial \theta_d^k} \delta_{cd}^{jk} \right] \end{aligned} \tag{B.2}$$

Here, again, t does not depend on f, so $\frac{\partial t}{\partial \pi_j} = 0$ and $\frac{\partial t}{\partial \theta_c^j} = 0$. Replacing integral expression in (2.13) with (B.1) and (B.2) yields

$$\frac{\hat{x}_d}{\partial \pi_j} = -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \sum_k \frac{\partial p_d^k}{\partial \pi_j} \int_{t=-\infty}^{\hat{x}_d} f_d^k(t|\theta_d^k) dt \tag{B.3}$$

$$= -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \sum_k \frac{\partial \log p_d^k}{\partial \pi_j} p_d^k F_d^k(\hat{x}_d|\theta_d^k) \tag{B.4}$$

$$\frac{\partial \hat{x}_d}{\partial \theta_c^j} = -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \int_{t=-\infty}^{\hat{x}_d} \sum_{k=1}^{K} \left[\frac{\partial p_d^k}{\partial \theta_c^j} f_d^k(t|\theta_d^k) + p_d^k \frac{\partial f_d^k(t|\theta_d^k)}{\partial \theta_d^k} \delta_{cd}^{jk} \right] dt \tag{B.5}$$

$$= -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \sum_{k=1}^{K} \left[\frac{\partial p_d^k}{\partial \theta_c^j} \int_{t=-\infty}^{\hat{x}_d} f_d^k(t|\theta_d^k) dt + p_d^k \int_{t=-\infty}^{\hat{x}_d} \frac{\partial f_d^k(t|\theta_d^k)}{\partial \theta_d^k} \delta_{cd}^{jk} dt \right] \tag{B.6}$$

$$= -\frac{1}{f_d(\hat{x}_d|\hat{\mathbf{x}}_{<d})} \sum_{k=1}^{K} \left[\frac{\partial \log p_d^k}{\partial \theta_c^j} p_d^k F_d^k(\hat{x}_d|\theta_d^k) + p_d^k \frac{\partial F_d^k(\hat{x}_d|\theta_d^k)}{\partial \theta_d^k} \delta_{cd}^{jk} \right] \tag{B.7}$$

In order to obtain $\frac{\partial \log p_d^k}{\partial \pi_j}$ and $\frac{\partial \log p_d^k}{\partial \theta_c^j}$ needed for computation of expressions above we need to differentiate (3.3):

$$\frac{\partial \log p_d^k}{\partial \pi_j} = \frac{\partial}{\partial \pi_j} \left[\log p_{d-1}^k + \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k) - \log \left(\sum_k p_{d-1}^k f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k) \right) \right] \tag{B.8}$$

$$= \frac{\partial p_{d-1}^k}{\partial \pi_j} + \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \tag{B.9}$$

$$- \frac{1}{\sum_{l'} p_{d-1}^{l'} f_{d-1}^{l'}(\hat{x}_{d-1}|\theta_{d-1}^{l'})} \sum_l f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l) \frac{\partial p_{d-1}^l}{\partial \pi_j} \tag{B.10}$$

$$- \frac{1}{\sum_{l'} p_{d-1}^{l'} f_{d-1}^{l'}(\hat{x}_{d-1}|\theta_{d-1}^{l'})} \sum_l p_{d-1}^l \frac{\partial f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l)}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \tag{B.11}$$

$$= \frac{\partial p_{d-1}^k}{\partial \pi_j} + \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} - \sum_l \frac{f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l)}{\sum_{l'} p_{d-1}^{l'} f_{d-1}^{l'}(\hat{x}_{d-1}|\theta_{d-1}^{l'})} \frac{\partial p_{d-1}^l}{\partial \pi_j} \tag{B.12}$$

$$- \sum_l \frac{p_{d-1}^m}{\sum_{l'} p_{d-1}^{l'} f_{d-1}^{l'}(\hat{x}_{d-1}|\theta_{d-1}^{l'})} \frac{\partial f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l)}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \tag{B.13}$$

$$= \frac{\partial p_{d-1}^k}{\partial \pi_j} + \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} - \sum_l \frac{p_d^l}{p_{d-1}^l} \frac{\partial p_{d-1}^l}{\partial \pi_j} \tag{B.14}$$

$$- \sum_l \frac{p_d^l}{f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l)} \frac{\partial f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l)}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \tag{B.15}$$

$$= \frac{\partial p_{d-1}^k}{\partial \pi_j} + \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \tag{B.16}$$

$$- \sum_l p_d^l \frac{\partial \log p_{d-1}^l}{\partial \pi_j} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l)}{\partial \hat{x}_{d-1}} \frac{\partial \hat{x}_{d-1}}{\partial \pi_j} \tag{B.17}$$

$$\frac{\partial \log p_d^k}{\partial \theta_c^j} = \frac{\partial \log p_{d-1}^k}{\partial \theta_c^j} - \sum_l p_d^l \frac{\partial \log p_{d-1}^l}{\partial \theta_c^j} \tag{B.18}$$

$$+ \left[\frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \hat{x}_{d-1}} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l)}{\partial \hat{x}_{d-1}} \right] \frac{\partial \hat{x}_{d-1}}{\partial \theta_c^j} \tag{B.19}$$

$$+ \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \theta_{d-1}^k} \frac{\partial \theta_{d-1}^k}{\partial \theta_c^j} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l)}{\partial \theta_{d-1}^l} \frac{\partial \theta_{d-1}^l}{\partial \theta_c^j} \tag{B.20}$$

$$= \frac{\partial \log p_{d-1}^k}{\partial \theta_c^j} - \sum_l p_d^l \frac{\partial \log p_{d-1}^l}{\partial \theta_c^j} \tag{B.21}$$

$$+ \left[\frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \hat{x}_{d-1}} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(\hat{x}_{d-1}|\theta_{d-1}^l)}{\partial \hat{x}_{d-1}} \right] \frac{\partial \hat{x}_{d-1}}{\partial \theta_c^j} \tag{B.22}$$

$$+ \frac{\partial \log f_{d-1}^k(\hat{x}_{d-1}|\theta_{d-1}^k)}{\partial \theta_{d-1}^k} \delta_{c(d-1)}^{jk} - p_d^j \frac{\partial \log f_{d-1}^j(\hat{x}_{d-1}|\theta_{d-1}^j)}{\partial \theta_{d-1}^j} \delta_{c(d-1)} \tag{B.23}$$

After some rearrangement we can obtain relations (2.20) and (3.8).

B.2 Algorithm for general stochastic backpropagation

Pseudocode for complete computation is described in Algorithm 2:

B.3 KL-divergence approximation

$$D_{\mathrm{KL}}\left(q_{\phi,d}(z_d|\pi_d, \mu_d, \sigma_d^2)||p(z_d)\right) = \tag{B.24}$$

$$= D_{\mathrm{KL}}\left(\pi_d \mathcal{N}\left(z_d|\mu_d, \sigma_d^2\right) + (1-\pi_d)\delta\left(z_d\right)||p(z_d)\right) \tag{B.25}$$

$$= \pi_d \mathbb{E}_{\mathcal{N}} \log \frac{\pi_d \mathcal{N}\left(z_d|\mu_d, \sigma_d^2\right) + (1-\pi_d)\delta(z_d)}{p(z_d)} \tag{B.26}$$

$$+ (1-\pi_d)\mathbb{E}_{\delta} \log \frac{\pi_d \mathcal{N}\left(z_d|\mu_d, \sigma_d^2\right) + (1-\pi_d)\delta(z_d)}{p(z_d)} \tag{B.27}$$

Since the second term is 0 everywhere except the origin, and since we want to approximate delta-function as a Gaussian with small mean μ_0 and variance σ_0, the numerator in the

second term will become

$$\pi_d \mathcal{N}\left(0|\mu_d, \sigma_d^2\right) + (1-\pi_d)\mathcal{N}\left(0|\mu_0, \sigma_0^2\right) \sim const + (1-\pi_d)\mathcal{N}\left(0|\mu_0, \sigma_0^2\right) \tag{B.28}$$

We will assume that this constant is insignificantly small, meaning that our components mostly do not overlap. With the same approximation of delta-function and with the same assumption of no overlap, the first term becomes $\pi_d \mathbb{E}_{\mathcal{N}} \log \frac{\pi_d \mathcal{N}\left(z_d|\mu_d, \sigma_d^2\right)}{p(z_d)}$. Thus, KL divergence will be

$$D_{\mathrm{KL}}\left(q_{\phi,d}(z_d|\pi_d, \mu_d, \sigma_d^2)||p(z_d)\right) = \pi_d D_{KL}\left(\mathcal{N}\left(z_d|\mu_d, \sigma_d^2\right)||p(z_d)\right) \tag{B.29}$$

$$+(1-\pi_d)D_{KL}\left(\mathcal{N}\left(z_d|\mu_0, \sigma_0^2\right)||p(z_d)\right) + \pi_d \log \pi_d + (1-\pi_d)\log(1-\pi_d) \tag{B.30}$$

After some rearrangement,

$$D_{\mathrm{KL}}\left(q_{\phi,d}(z_d|\pi_d, \mu_d, \sigma_d^2)||p(z_d)\right) = \pi_d D_{KL}\left(\mathcal{N}\left(z_d|\mu_d, \sigma_d^2\right)||p(z_d)\right) \tag{B.31}$$

$$+(1-\pi_d)D_{KL}\left(\mathcal{N}\left(z_d|\mu_0, \sigma_0^2\right)||p(z_d)\right) + \pi_d \log \pi_d + (1-\pi_d)\log(1-\pi_d) \tag{B.32}$$

The term $\pi_d \log \pi_d + (1-\pi_d)\log(1-\pi_d)$ corresponds to the negative entropy $-H(\pi_d)$, so finally

$$D_{\mathrm{KL}}\left(q_{\phi,d}(z_d|\pi_d, \mu_d, \sigma_d^2)||p(z_d)\right) = \pi_d D_{KL}\left(\mathcal{N}\left(z_d|\mu_d, \sigma_d^2\right)||p(z_d)\right) \tag{B.33}$$

$$+(1-\pi_d)D_{KL}\left(\mathcal{N}\left(z_d|\mu_0, \sigma_0^2\right)||p(z_d)\right) - H(\pi_d) \tag{B.34}$$

$-H(\pi_d)$ is minimized at $\pi_d = 0.5$. It contradicts the goal of our task to separate zero units from informative ones, which corresponds to the case when all weights are either zeros or ones, or equally when $H(\pi_d) = 0$. That is why we introduce a hyperparameter η that adjusts the strength of impact by the entropy term, which is a common trick ([5, 2]). In this work, since the goal was to prevent components from merging and to promote zeros and ones in weights, it was considered that $\eta \approx 0$, but the research of the way to introduce this term is left for future work.

Algorithm 2 Stochastic backpropagation through mixture density distributions

initialize $\frac{\partial h}{\partial \pi_j} \leftarrow 0$, $\frac{\partial h}{\partial \theta_c^j} \leftarrow 0$

for $n = 1$ to N **do**

draw $\mathbf{x} \sim f(\mathbf{x})$

$p_1^k \leftarrow \pi_k$

$f_1(x_1) \leftarrow \sum_k \pi_k f_1^k(x_1|\theta_1^k)$

$\frac{\partial p_1^k}{\partial \pi_j} \leftarrow -\frac{\delta_{jk}}{\pi_j}$

$\frac{\partial \log p_1^k}{\partial \theta_c^j} \leftarrow 0$

$\frac{\partial x_1}{\partial \pi_j} \leftarrow -\frac{F_1(x_1)}{f_1(x_1)}$

$\frac{\partial x_1}{\partial \theta_c^j} \leftarrow -\frac{p_1^j}{f_1(x_1)} \frac{\partial F_1^j(x_1|\theta_1^j)}{\partial \theta_1^j} \delta_{1c}$

for $d = 1$ to D **do**

$f_d(x_d|\mathbf{x}_{<d}) \leftarrow \sum_k p_d^k f_d^k(x_d|\theta_d^k)$

$p_d^k \leftarrow \frac{p_{d-1}^k f_{d-1}^k(x_{d-1}|\theta_{d-1}^k)}{f_{d-1}(x_{d-1}|\mathbf{x}_{<d-1})}$

$\frac{\partial \log p_d^k}{\partial \pi_j} \leftarrow \frac{\partial \log p_{d-1}^k}{\partial \pi_j} - \sum_l p_d^l \frac{\partial \log p_{d-1}^l}{\partial \pi_j} +$
$\left[\frac{\partial \log f_{d-1}^k(x_{d-1}|\theta_{d-1}^k)}{\partial x_{d-1}} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(x_{d-1}|\theta_{d-1}^l)}{\partial x_{d-1}}\right] \frac{\partial x_{d-1}}{\partial \pi_j}$

$\frac{\partial \log p_d^k}{\partial \theta_c^j} \leftarrow \frac{\partial \log p_{d-1}^k}{\partial \theta_c^j} - \sum_l p_d^l \frac{\partial \log p_{d-1}^l}{\partial \theta_c^j} +$
$+ \left[\frac{\partial \log f_{d-1}^k(x_{d-1}|\theta_{d-1}^k)}{\partial x_{d-1}} - \sum_l p_d^l \frac{\partial \log f_{d-1}^l(x_{d-1}|\theta_{d-1}^l)}{\partial x_{d-1}}\right] \frac{\partial x_{d-1}}{\partial \theta_c^j} +$
$+ \frac{\partial \log f_{d-1}^k(x_{d-1}|\theta_{d-1}^k)}{\partial \theta_{d-1}^k} \delta_{c(d-1)}^{jk} - p_d^j \frac{\partial \log f_{d-1}^j(x_{d-1}|\theta_{d-1}^j)}{\partial \theta_{d-1}^j} \delta_{c(d-1)}$

$\frac{\partial x_d}{\partial \pi_j} \leftarrow -\frac{1}{f_d(x_d|\mathbf{x}_{<d})} \sum_{k=1}^K \frac{\partial \log p_d^k}{\partial \theta_c^j} p_d^k F_d^k(x_d|\theta_d^k)$

$\frac{\partial x_d}{\partial \theta_c^j} \leftarrow -\frac{1}{f_d(x_d|\mathbf{x}_{<d})} \sum_{k=1}^K \left[\frac{\partial \log p_d^k}{\partial \theta_c^j} p_d^k F_d^k(x_d|\theta_d^k) + p_d^k \frac{\partial F_d^k(x_d|\theta_d^k)}{\partial \theta_d^k} \delta_{cd}^{jk}\right]$

end for

$\frac{\partial h}{\partial \pi_j} \leftarrow \frac{\partial h}{\partial \pi_j} + \sum_d \frac{\partial g(\mathbf{x})}{\partial x_d} \frac{\partial x_d}{\partial \pi_j}$

$\frac{\partial h}{\partial \theta_c^j} \leftarrow \frac{\partial h}{\partial \pi_j} + \sum_d \frac{\partial g(\mathbf{x})}{\partial x_d} \frac{\partial x_d}{\partial \theta_c^j}$

end for

$\frac{\partial h}{\partial \pi_j} \leftarrow \frac{1}{N} \frac{\partial h}{\partial \pi_j}$

$\frac{\partial h}{\partial \theta_c^j} \leftarrow \frac{1}{N} \frac{\partial h}{\partial \theta_c^j}$

Appendix C

Experimental results

C.1 Network architectures and parameters

In this section neural network architectures that were used during experiments are described. Here N_z denotes the dimensionality of the latent space. Models were optimized with Adam[10] with learning rate 10^{-4} and standard values of hyperparameters: $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-8}$. Minibatches of the size 100 with 10 Monte Carlo samples were used during training. Description of the networks is split into two tables, in Table C.1 the encoding network is defined, and in Table C.2 one can find description of the decoding network. These architectures were implemented in Python 2.7 with the use of deep learning libraries Theano[21] and Lasagne[1].

C.2 Visualization of model performance

In this section one can find examples of outputs of encoding-decoding procedure performed by each of the three VAEs described in this work. Structure of these comparison images is the following: the first column is an original random image from the validation set of one of the datasets, while the second, third and fourth columns correspond to the encoding-decoding of this random image returned by the original VAE, loguniform prior VAE and Sparse VAE respectively.

Dataset	Input	Hidden	Output
MNIST	28×28	FC 500 Tanh FC $N_z \times 2$ + FC N_z Sigmoid	$N_z = 200 \times 2$ + N_z for SparseVae
Omniglot	105×105	FC 500 Tanh FC $N_z \times 2$ + FC N_z Sigmoid	$N_z = 200 \times 2$ + N_z for SparseVae
Omniglot	105×105	Conv $32 \times 10 \times 10$ ReLu (1-0) MaxPool (2-2) Conv $64 \times 7 \times 7$ ReLu (1-0) MaxPool (2-2) Conv $64 \times 4 \times 4$ ReLu (1-0) MaxPool (2-2) Conv $128 \times 4 \times 4$ ReLu (1-0) FC $N_z \times 2$ + FC N_z Sigmoid	$N_z = 200 \times 2$ + N_z for SparseVae
CIFAR-10	$3 \times 32 \times 32$	FC 500 Tanh FC $N_z \times 2$ + FC N_z Sigmoid	$N_z = 400 \times 2$ + N_z for SparseVae
CIFAR-10	$3 \times 32 \times 32$	Conv $512 \times 6 \times 6$ Tanh (2-0) Conv $256 \times 4 \times 4$ Tanh (2-0) Conv $128 \times 3 \times 3$ Tanh (1-0) Conv $500 \times 4 \times 4$ Tanh (1-0)} FC $N_z \times 2$ + FC N_z Sigmoid	$N_z = 400 \times 2$ + N_z for SparseVae

Table C.1: Neural network architectures for encoder $q_\phi(\mathbf{z}|\mathbf{x})$

Dataset	Input	Hidden	Output
MNIST	200	FC 500 Tanh FC $28 * 28$ Sigmoid	28×28
Omniglot	200	FC 500 Tanh FC $105 * 105$ Sigmoid	105×105
Omniglot	200	FC 500 ReLu FC 128*6*6 ReLu TranspConv $64 \times 4 \times 4$ ReLu (1-0) TranspConv $64 \times 4 \times 4$ ReLu (1-0) TranspConv $32 \times 7 \times 7$ ReLu (1-0) TranspConv $1 \times 10 \times 10$ ReLu (1-0) Sigmoid	105×105
CIFAR-10	400	FC 500 Tanh FC $3 * 32 * 32 \times 2$	$\{3 \times 32 \times 32\} \times 2$
CIFAR-10	400	FC 500 Tanh TranspConv $128 \times 4 \times 4$ Tanh (1-0) TranspConv $256 \times 3 \times 3$ Tanh (1-0) TranspConv $512 \times 4 \times 4$ Tanh (2-0) TranspConv $3 \times 6 \times 6$ Tanh (2-0) $\times 2$	$\{3 \times 32 \times 32\} \times 2$

Table C.2: Neural network architectures for decoder $p_\theta(\mathbf{x}|\mathbf{z})$

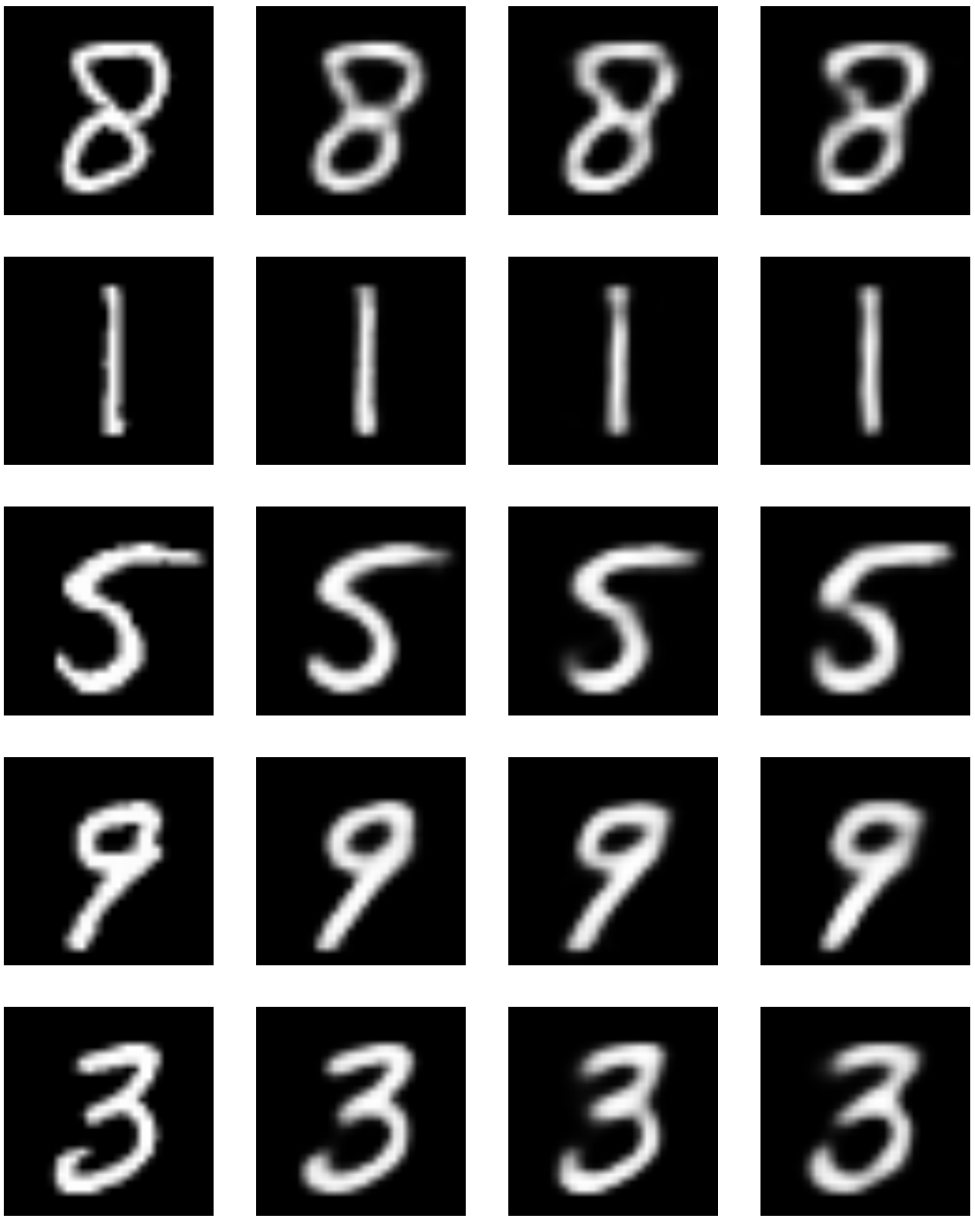

Figure C-1: Comparison of decoded images from MNIST dataset. First column corresponds to origianl image, second column - to original VAE, third column - to loguniform VAE, and the last column - to sparse VAE.

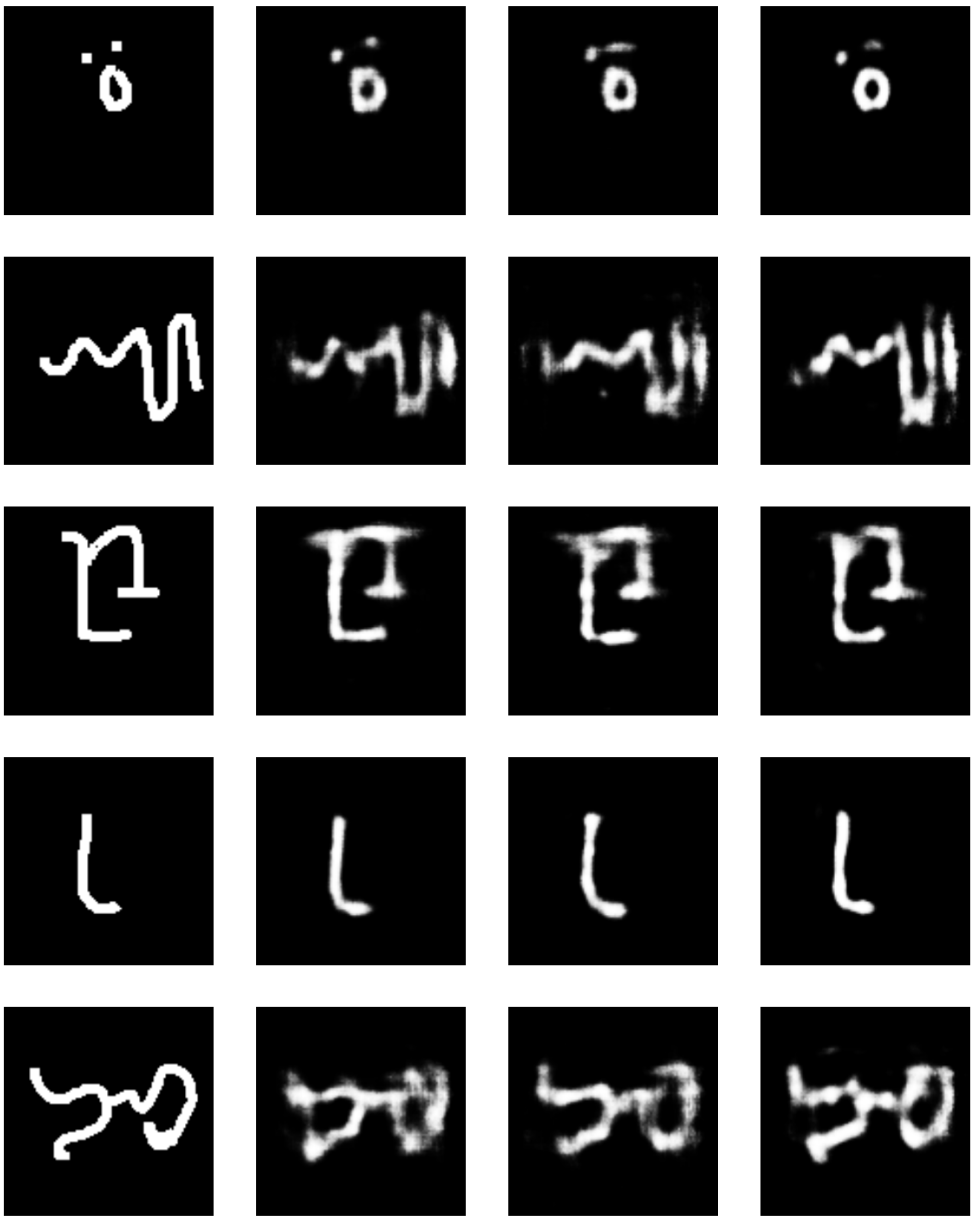

Figure C-2: Comparison of decoded images from Omniglot dataset on the fully connected architecture. First column corresponds to origianl image, second column - to original VAE, third column - to loguniform VAE, and the last column - to sparse VAE.

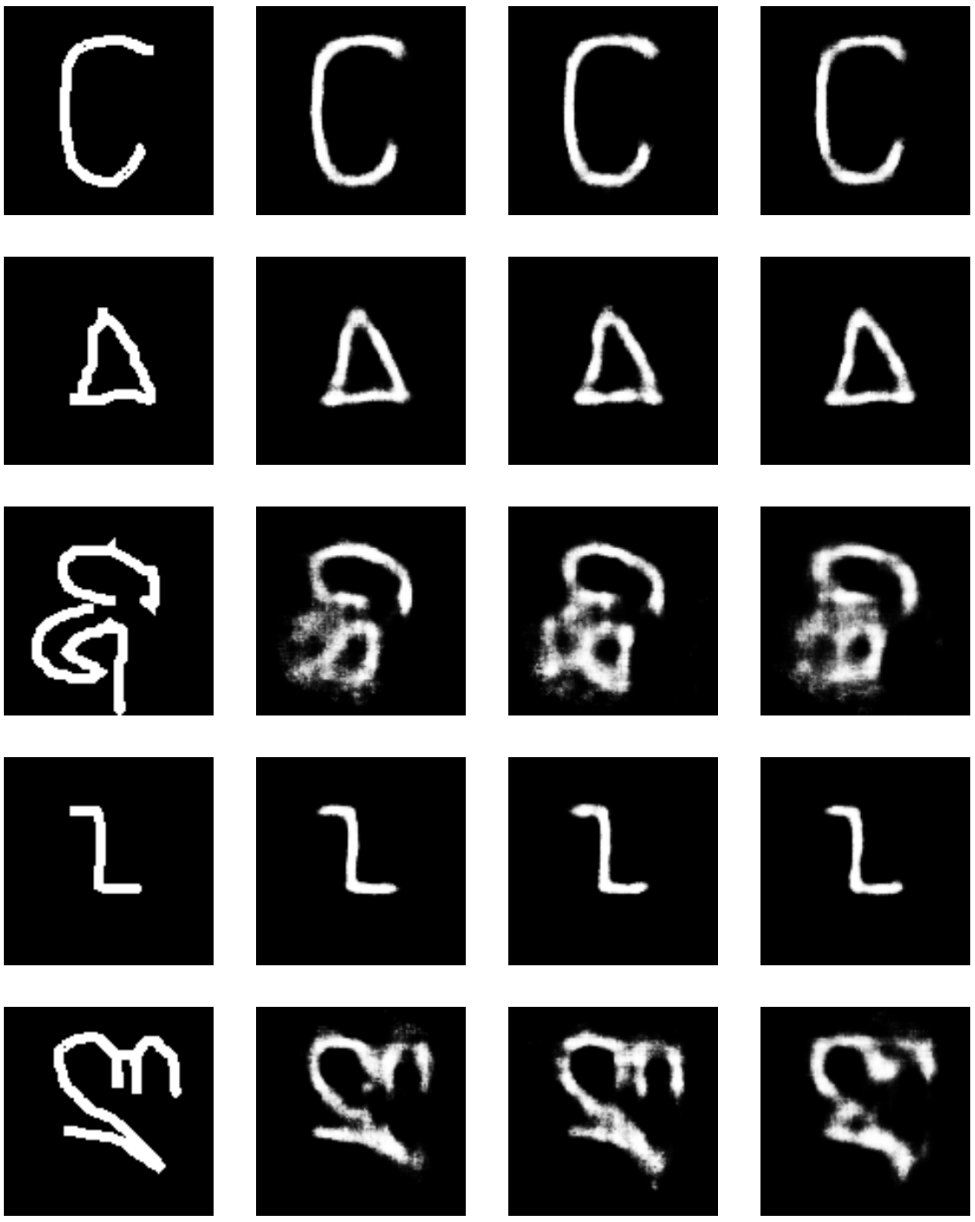

Figure C-3: Comparison of decoded images from Omniglot dataset on the convolutional architecture. First column corresponds to origianl image, second column - to original VAE, third column - to loguniform VAE, and the last column - to sparse VAE.

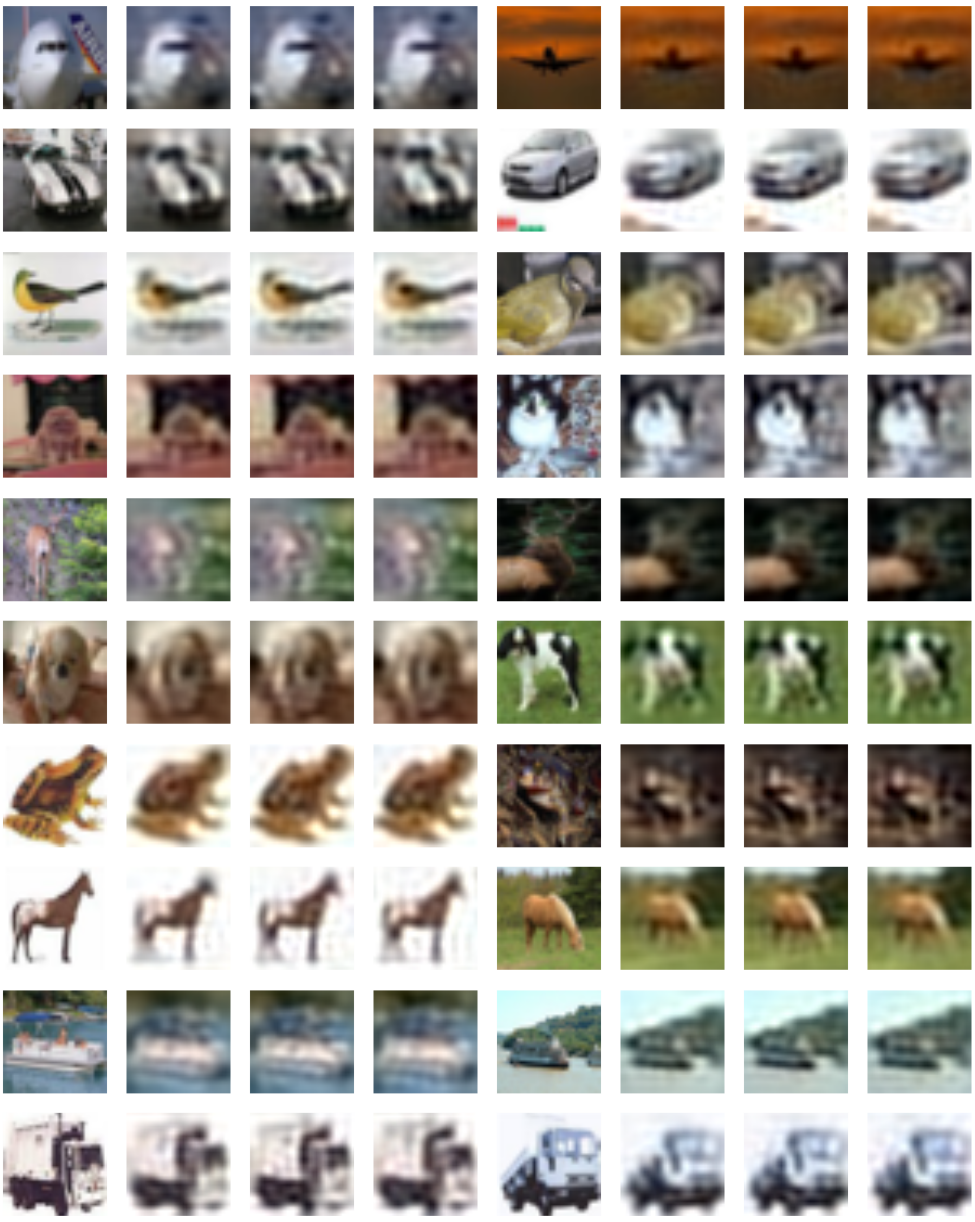

Figure C-4: Comparison of decoded images from CIFAR-10 dataset on the fully connected architecture. First column corresponds to origianl image, second column - to original VAE, third column - to loguniform VAE, and the last column - to sparse VAE.

Figure C-5: Comparison of decoded images from CIFAR-10 dataset on the convolutional architecture. First column corresponds to origianl image, second column - to original VAE, third column - to loguniform VAE, and the last column - to sparse VAE.

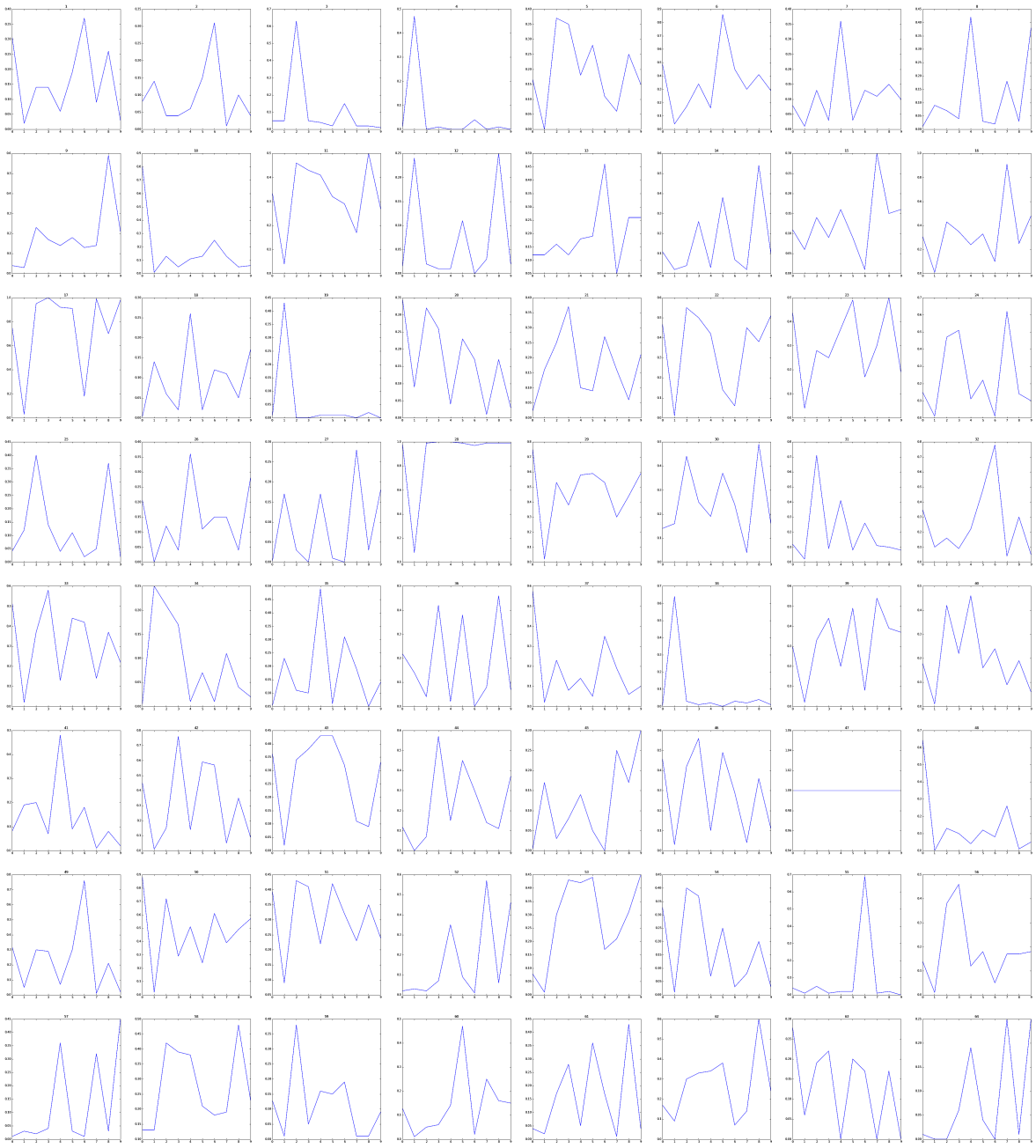

Figure C-6: Activity of latent features for each of the training classes. Activity level was measured by calculation of the proportion of objects from a particular class for which this feature is active.