Just Relax It! Or how we made a cutting-edge library for Discrete Variables Relaxation

Daniil Dorin, Igor Ignashin, Nikita Kiselev, Andrey Veprikov

November 2024

1 Introduction

In this blog post, we are going to talk about a really interesting topic that is super useful in modern neural networks, especially for generative models like VAEs. The topic is discrete probability distributions and how to optimize their parameters. Why is this important? Well, no generative model can be complete without some randomness, because that is what makes new objects (images, audio, videos, etc.) appear. But sometimes we need to adjust the distribution parameters to minimize errors. With continuous distributions, it is usually pretty easy, but things get tricky when we get into discrete ones, like Bernoulli and categorical.

Thus, we present our new Python library called "Just Relax It" that combines the best techniques for relaxing discrete distributions (we will explain what that means later) into an easy-to-use package. And it is compatible with PyTorch!

We start with a basic example that shows how parameter optimization typically happens for continuous distributions, then we move on smoothly to the case of discrete distributions. After that, we talk about the main relaxation techniques used in our library and make a demo of training a VAE with discrete latent variables, using each of these algorithms.

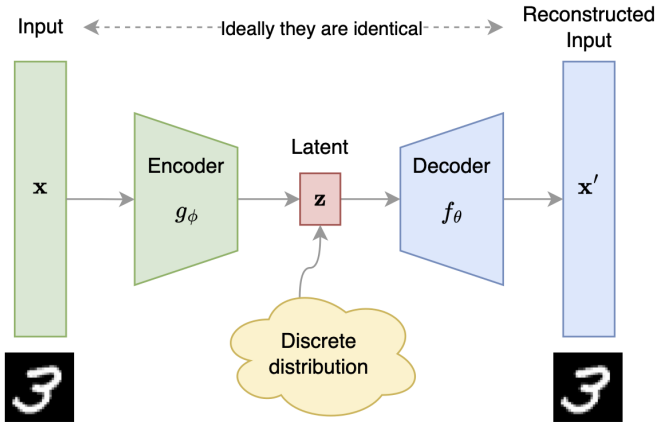


Figure 1: Variational Autoencoder (VAE) architecture

VAE example

The original VAE [1] consists of two parts (see Fig. 1):

1. Encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, which is represented by a neural network $g_{\boldsymbol{\phi}}(\mathbf{x})$ that outputs parameters of the latent Gaussian distribution;

2. Decoder $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, which is represented by a neural network $f_{\boldsymbol{\theta}}(\mathbf{z})$ that outputs parameters of the sample distribution (typically Gaussian or Bernoulli).

The math behind training a VAE is not really something the average person can understand, so we will just focus on the ELBO (evidence lower bound), which needs to be maximized w.r.t. the parameters of the encoder and decoder:

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) \to \max_{\boldsymbol{\phi},\boldsymbol{\theta}}.$$

During the M-step, we gonna derive the unbiased estimator for the $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x})$:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) &= \nabla_{\boldsymbol{\theta}} \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) d\mathbf{z} \\ &= \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) d\mathbf{z} \\ &\approx \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}^*), \quad \mathbf{z}^* \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}),\end{aligned}$$

where the last approximation is a Monte-Carlo sampling estimator.

However, on the E-step it is quite tricky to get unbiased estimator for the $\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x})$: as density function $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ depends on the parameters $\boldsymbol{\phi}$, it is

impossible to use the Monte-Carlo estimation:

$$\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \nabla_{\boldsymbol{\phi}}\int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})d\mathbf{z} - \nabla_{\boldsymbol{\phi}}KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$$
$$\neq \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})d\mathbf{z} - \nabla_{\boldsymbol{\phi}}KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})),$$

and this is the moment where the **reparameterization trick** arises, we reparameterize the outputs of the **encoder**:

$$\nabla_{\boldsymbol{\phi}}\int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})d\mathbf{z} = \int p(\boldsymbol{\epsilon})\nabla_{\boldsymbol{\phi}}\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{g}_{\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon}))d\boldsymbol{\epsilon}$$
$$\approx \nabla_{\boldsymbol{\phi}}\log p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\mathbf{x})\odot\boldsymbol{\epsilon}^* + \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x})), \quad \boldsymbol{\epsilon}^*\sim\mathcal{N}(0,\mathbf{I}),$$

so we move the randomness to the $\boldsymbol{\epsilon}\sim p(\boldsymbol{\epsilon})$, and use the deterministic transform $\mathbf{z} = \mathbf{g}_{\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon})$ in order to get unbiased gradient. It also needs to be mentioned that normal assumptions for $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$ allows us to compute KL analytically and thus calculate the gradient $\nabla_{\boldsymbol{\phi}}KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$.

The above example gives us an understanding of a crucial reparameterization trick, which allows us to get unbiased gradient estimations for the continuous latent space in VAE model. But actually discrete representations $\mathbf{z}$ are potentially a more natural fit for many of the modalities (like texts or images), which moves us to the **discrete VAE latentes**. Therefore

- Our encoder should output discrete distribution;
- We need the analogue of the reparameterization trick for the dicrete distribution;
- Our decoder should input discrete random variable.

The classical solution for the discrete variables reparameterization trick is **Gumbel-Softmax** [2] or **Concrete relaxation** [3].

Gumbel distribution

$$g\sim\text{Gumbel}(0,1) \quad\Leftrightarrow\quad g = -\log(-\log u), \quad u\sim\text{Uniform}[0,1]$$

Theorem (Gumbel-Max trick)

Let $g_k\sim\text{Gumbel}(0,1)$ for $k = 1,\ldots,K$. Then a discrete random variable

$$c = \arg\max_k[\log\pi_k + g_k]$$

has a categorical distribution $c\sim\text{Categorical}(\boldsymbol{\pi})$.

- We could sample from the discrete distribution using Gumbel-Max reparameterization;

- Here parameters and random variable sampling are separated (reparameterization trick);
- **Problem:** we still have non-differentiable arg max operation.

Gumbel-Softmax relaxation

$$\hat{\mathbf{c}} = \text{softmax}\left(\frac{\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) + \mathbf{g}}{\tau}\right)$$

Here τ is a temperature parameters. Now we have differentiable operation, but the gradient estimator is biased now. However, if $\tau \to 0$, then the estimation becomes more and more accurate.

Other relaxation methods

So far, we have talked about one possible example of a discrete variable relaxation (VAE with discrete latent variables) and the classical approach to solving this problem. However, the Gumbel-Softmax relaxation was actually proposed a long time ago. There are actually many other relaxation techniques that can provide more flexible and accurate (and even unbiased) gradient estimates. The rest of our blog-post will focus on cutting-edge relaxation techniques and how we built a Python library that uses them, which works with the PyTorch framework to train neural networks efficiently.

2 Algorithms

In this section, we provide a short description for each of the implemented relaxation methods.

2.1 Relaxed Bernoulli [4]

- Given $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$, and independent Bernoulli random variables $s_d \sim \text{Be}(\pi_d)$, $d = 1, \ldots, D$, composed in the vector $\mathbf{s} = [s_1, \ldots, s_D] \in \mathbb{R}^D$, the authors originally propose a relaxation method for an embedded feature selection problem

  $$\min_{\boldsymbol{\theta}, \boldsymbol{\pi}} \mathbb{E}_{\mathbf{x},\mathbf{y}} \mathbb{E}_{\mathbf{s}} \left[L(f_{\boldsymbol{\theta}}(\mathbf{x} \odot \mathbf{s}), \mathbf{y}) + \lambda \|\mathbf{s}\|_0 \right],$$

  where L is a loss function, $f_{\boldsymbol{\theta}}$ is a parameterized model with parameters $\boldsymbol{\theta}$, and $\|\mathbf{s}\|_0$ is the number of non-zero entries $s_1, \ldots, s_D$
- However, to simplify a discussion, one can consider a similar task in which the proposed method is needed:

  $$\min_{\pi} \mathbb{E}_s \left[f(s) \right]$$

- The expectation $\mathbb{E}_s f(s)$ is hard to optimize with respect to π, as s is a discrete random variable, that leads to the infeasibility of the gradient estimation $\frac{\partial}{\partial \pi}\mathbb{E}_s\left[f(s)\right]$
- The solution is proposed: a Gaussian-based continuous relaxation for the Bernoulli variable. The authors refer to the relaxed Bernoulli variable as a stochastic gate (STG) defined by

  $$z = \max(0, \min(1, \mu + \epsilon)),$$

  where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and σ is fixed throughout training
- The objective becomes

  $$\min_{\mu} \mathbb{E}_z\left[f(z)\right],$$

  thus, we can update the parameter μ via gradient descent, as a Monte Carlo sampling gradient estimator gives us

  $$\frac{\partial}{\partial \mu}\mathbb{E}_z\left[f(z)\right] \approx f'(z)\frac{\partial z}{\partial \mu},$$

  which is commonly known as the reparameterization trick
- During the testing phase, they use a deterministic gate $z = \max(0, \min(1, \mu))$

2.2 Correlated relaxed Bernoulli [5]

- Instead of assuming independence in the gating mechanism, the authors model the gate vector distribution as a multivariate Bernoulli distribution where dependencies among the gates are determined by the correlation structure of the input features
- To this goal, they use *Gaussian copula* to construct a valid multivariate Benoulli distribution

2.3 Gumbel-Softmax TOP-K [6]

TODO

2.4 Straight-Through Bernoulli [7]

TODO

2.5 Invertible Gaussian [8]

TODO

2.6 Hard Concrete [9]

TODO

2.7 REINFORCE [10]

TODO

2.8 Closed-form Laplace Bridge [11]

TODO

3 Implementation

Our implementation scheme is on the Fig. 2.

1. The most famous Python probabilistic libraries with a built-in differentiation engine are PyTorch and Pyro. Specifically, we are mostly interested in the `distributions` package in both of them.

2. Base class for PyTorch-compatible distributions with Pyro support is `TorchDistribution`, for which we refer to this page on documentation. This should be the base class for almost all new Pyro distributions. Therefore in our project we are planning to inherit classes from this specific one.

3. To make our library compatible with modern deep learning packages, we implement our classes with the following methods and properties, as it is mentioned in the Pyro documentation: Derived classes must implement the methods `sample()` (or `rsample()` if `.has_rsample == True`) and `log_prob()`, and must implement the properties `batch_shape`, and `event_shape`. Discrete classes may also implement the `enumerate_support()` method to improve gradient estimates and set `.has_enumerate_support = True`.

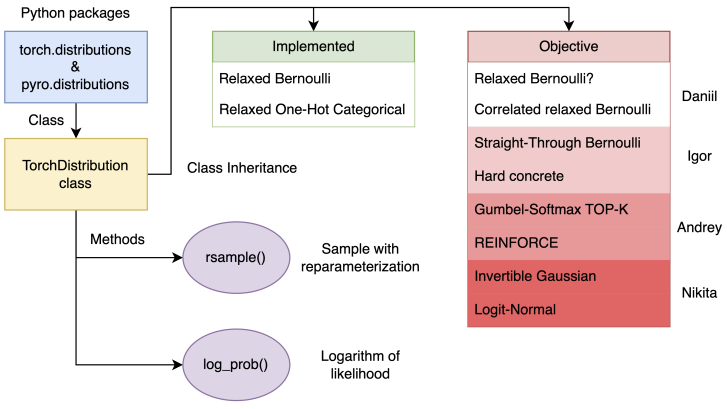
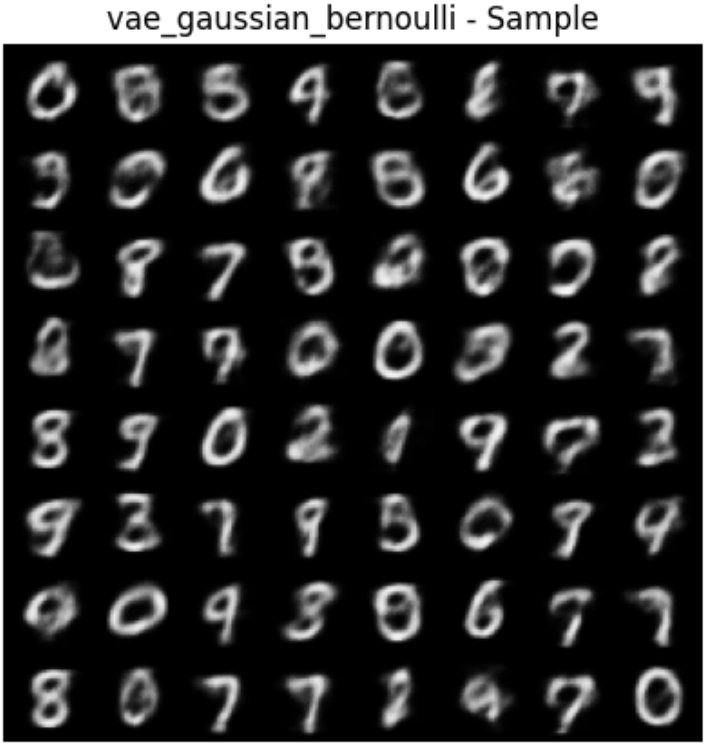


Figure 2: Implementation Scheme

4 Demo

For demonstration purposes, we have implemented a simple VAE with discrete latents. Our code is available at this link. Each of the discussed relaxation techniqures allowed us to learn the latent space with the corresponding distribution. Here we provide some examples on the figures below (all the results are available in the above repository). We trained the VAE on the MNIST dataset for several epochs, and printed reconstruction and sample digits examples.



TODO make more examples and (probably) discuss differencies

5 Conclusion

In summary, "Just Relax It" is a powerful tool for researchers and practitioners working with discrete variables in neural networks. By offering a comprehensive set of relaxation techniques, our library aims to make the optimization process more efficient and accessible. We encourage you to explore our library, try out the demo, and contribute to its development. Together, we can push the boundaries of what is possible with discrete variable relaxation in machine learning.

Thank you for reading, and happy coding!

Daniil Dorin, Igor Ignashin, Nikita Kiselev, Andrey Veprikov

References

[1] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[2] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017. URL `https://arxiv.org/abs/1611.01144`.

[3] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables, 2017. URL `https://arxiv.org/abs/1611.00712`.

[4] Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. Feature selection using stochastic gates, 2020. URL `https://arxiv.org/abs/1810.04247`.

[5] Changhee Lee, Fergus Imrie, and Mihaela van der Schaar. Self-supervision enhanced feature selection with correlated gates. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=oDFvtxzPOx`.

[6] Wouter Kool, Herke van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement, 2019. URL `https://arxiv.org/abs/1903.06059`.

[7] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL `https://arxiv.org/abs/1308.3432`.

[8] Andres Potapczynski, Gabriel Loaiza-Ganem, and John P. Cunningham. Invertible gaussian reparameterization: Revisiting the gumbel-softmax, 2022. URL `https://arxiv.org/abs/1912.09588`.

[9] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization, 2018. URL https://arxiv.org/abs/1712.01312.

[10] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[11] Marius Hobbhahn, Agustinus Kristiadi, and Philipp Hennig. Fast predictive uncertainty for classification with bayesian deep networks, 2022. URL https://arxiv.org/abs/2003.01227.