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.

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:

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

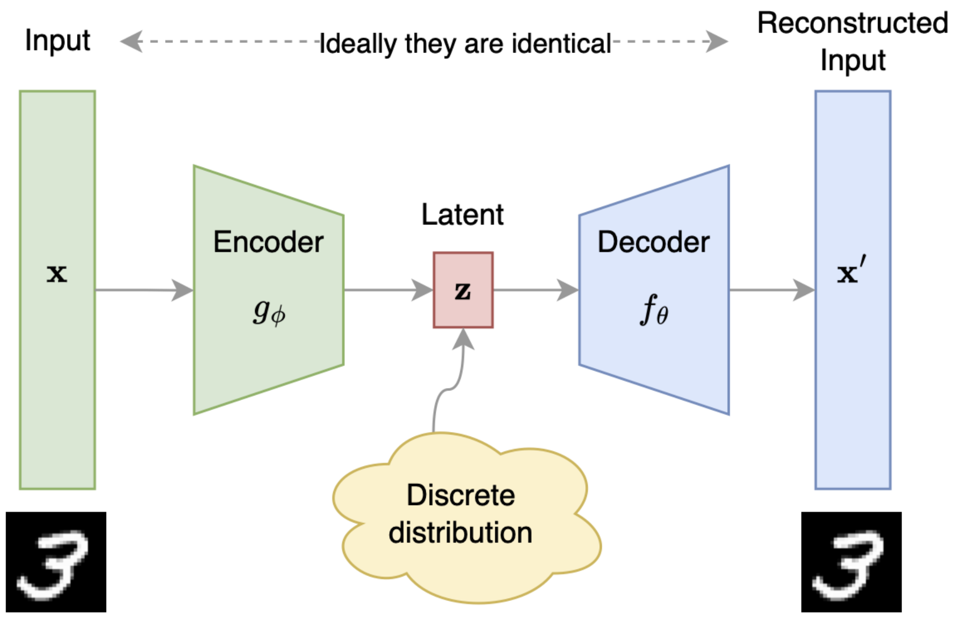


Figure 1: Variational Autoencoder (VAE) architecture

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:

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

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

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

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 $k = 1, \dots, K$. 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 methods. We can generalize them as follows. Suppose that x is a random variable, f if a function (say, the loss function), and we are interested in computing $\frac{\partial}{\partial\theta}\mathbb{E}_x[f(x)]$. It is quite natural decision because typical ML problem looks like this. So, two different ideas come to us:

- *Score function* (SF) estimator. In this case, we are given a parameterized probability distribution $x \sim p(\cdot;\theta)$ and use
$$\frac{\partial}{\partial\theta}\mathbb{E}_x[f(x)] = \mathbb{E}_x\left[f(x)\frac{\partial}{\partial\theta}\log p(x;\theta)\right].$$
- *Pathwise derivative* (PD) estimator. In this case x is a determinisitc, differentiable function of θ and another random variable z, i.e. we can write $x(z,\theta)$:
$$\frac{\partial}{\partial\theta}\mathbb{E}_x[f(x(z,\theta))] = \mathbb{E}_z\left[\frac{\partial}{\partial\theta}f(x(z,\theta))\right].$$

The latter one we have seen previously in the VAE example! A sample x from $\mathcal{N}(\mu, \sigma^2)$ can be obtained by sampling z from the standard normal distribution $\mathcal{N}(0,1)$ and then transforming it using $x(z,\theta) = \sigma z + \mu$. And this is called reparameterization trick.

However, when x is a discrete variable, it is quite tricky to make a pathwise derivative estimator, i.e. to reparameterize the discrete distribution. And this is the moment of relaxation! We replace x with a continuous relaxation $x(z,\theta) \approx x_\tau(z,\theta)$, where $\tau > 0$ is a temperature that controls the tightness of the relaxaton (at low temperatues, the relaxation is nearly high).

2.1 Relaxed Bernoulli [4]

The reparameterization trick is inspired by the idea of stochastic gates and aims to approximate a Bernoulli random variable in a more relaxed manner. This technique involves drawing a random variable, denoted as ϵ, from a normal distribution with a mean of 0 and a variance of σ^2, where σ is a fixed parameter. The random variable ϵ is then used to compute z as follows:

$$\epsilon \sim \mathcal{N}(0, \sigma^2),$$

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

where μ is a learnable parameter that can be tuned during the training process. This transformation ensures that the resulting z value is bounded between 0 and 1, thereby relaxing the Bernoulli distribution.

2.2 Correlated relaxed Bernoulli [5]

This method generates correlated gate vectors from a multivariate Bernoulli distribution using a Gaussian copula:

$$C_R(U_1, \dots, U_p) = \Phi_R(\Phi^{-1}(U_1), \dots, \Phi^{-1}(U_p)),$$

where Φ_R is the joint CDF of a multivariate Gaussian distribution with correlation matrix R, and Φ^{-1} is the inverse CDF of the standard univariate Gaussian distribution. The gate vector m is generated as:

$$m_k = \begin{cases} 1 & \text{if } U_k \le \pi_k \\ 0 & \text{if } U_k > \pi_k \end{cases} \quad k = 1, \dots, p,$$

where U_k are correlated random variables preserving the input feature correlations. For differentiability, a continuous relaxation is applied:

$$m_k = \sigma\left(\frac{1}{\tau}\left(\log\frac{U_k}{1-U_k} + \log\frac{\pi_k}{1-\pi_k}\right)\right),$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function, and τ is a temperature hyperparameter.

2.3 Gumbel-Softmax TOP-K [6]

- Suppose we want to get K samples without replacement (i.e., not repeating) according to the distribution π. Similar to the Gumbel-Max method, let $g_k \sim \text{Gumbel}(0,1)$ for $k = 1, \dots, K$, then the Gumbel-Max-Top$K$ Theorem says, that the values of the form

$$c_1, \dots, c_K = \underset{k}{\text{Argtop}K}[\log \pi_k + g_k]$$

  have the Categorical($\boldsymbol{\pi}$) distribution without replacement.

- This approach has all the same pros and cons as the classical Gumbel-Max trick, however, they can be fixed with the Gumbel-Softmax relaxation using a simple loop:

```
for $k = 1, \dots, K$ do
    $c_k$ = Gumbel-Softmax($\boldsymbol{\pi}$)
    $\pi_k = -\inf$
end for
```

2.4 Straight-Through Bernoulli [7]

The Straight-Through Bernoulli distribution can be written as follows

$$p_i = \sigma(a_i)$$
$$b_i \sim \text{Binomial}(\sqrt{p_i})$$
$$h_i = b_i\sqrt{p_i},$$

where a_i – parameter of this distribution.

2.5 Invertible Gaussian [8]

The idea is to remove interpretability of parameters in Gumbel-Softmax relaxation, and achieve then higher quality.

Objective: relax one-hot $\mathbf{z} \sim \text{Cat}(\boldsymbol{\pi})$

GS: $\tau \to 0$ concentrates mass on vertices: $\tilde{\mathbf{z}} = \text{softmax}(\frac{\log \boldsymbol{\pi} + \mathbf{G}}{\tau}), G_i \sim \text{Gumbel}(0,1)$

IGR: map $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to simplex, using invertible $g(\cdot,\tau)$ with temperature τ:

$$\mathbf{y} = \boldsymbol{\mu} + \text{diag}(\boldsymbol{\sigma})\boldsymbol{\epsilon},$$
$$\tilde{\mathbf{z}} = g(\mathbf{y},\tau) = \text{softmax}_{++}(\mathbf{y}/\tau)$$

2.6 Hard Concrete [9]

Hard Concrete is a special case of Concrete distribution. It is written as follows:

$$u \sim \mathcal{U}[0,1]$$
$$s = \sigma(\frac{logu + log(1-u) + log\alpha}{\beta})$$
$$\bar{s} = s(\zeta - \gamma) + \gamma$$
$$z = \min(1, \max(0, \bar{s})),$$

where $\alpha > 0, \beta > 0, \zeta > 1, \gamma < 0$ – parameters of this distribution.

Let consider pdf $q_s(s|\phi)$ and cdf $Q_s(s|\phi)$ of hard concrete variable :

$$q_s(s \mid \phi) = \frac{\beta\alpha s^{-\beta-1}(1-s)^{-\beta-1}}{(\alpha s^{-\beta} + (1-s)^{-\beta})^2},$$
$$Q_s(s \mid \phi) = \text{Sigmoid}((\log s - \log(1-s))\beta - \log\alpha).$$
$$q_{\bar{s}}(\bar{s} \mid \phi) = \frac{1}{|\zeta-\gamma|} q_s\left(\frac{\bar{s}-\gamma}{\zeta-\gamma}\middle|\phi\right), \quad Q_{\bar{s}}(\bar{s} \mid \phi) = Q_s\left(\frac{\bar{s}-\gamma}{\zeta-\gamma}\middle|\phi\right)$$

The final density and distribution function:

$$q(z|\phi) = Q_{\bar{s}}(0|\phi)\delta(z) + (1 - Q_{\bar{s}}(1|\phi))\delta(z-1) +$$
$$+ (Q_{\bar{s}}(1|\phi) - Q_{\bar{s}}(0|\phi))q_{\bar{s}}(z|\bar{s} \in (0,1), \phi)$$

2.7 Closed-form Laplace Bridge [10]

Here we do the following:

- We implement a `LogisticNormalSoftmax` distribution, which is a transformed distribution from the `Normal` one. In contrast to original `LogisticNormal` from `pyro` or `torch`, this one uses `SoftmaxTransform`, instead of `StickBreakingTransform` that allows us to remain in the same dimensionality.
- We implement two distinct functions, each of them has a distribution on input (Dirichlet or Logistic-Normal), and returns an approximation distribution.

2.8 REINFORCE [11]

- The REINFORCE method is fundamentally based on the *score function* estimator. The idea behind it is as follows:
$$\frac{\partial}{\partial\theta}\mathbb{E}_{x\sim p(\cdot,\theta)}[f(x)] = \mathbb{E}_{x\sim p(\cdot,\theta)}\left[f(x)\frac{\partial}{\partial\theta}\log p(x,\theta)\right].$$
This equation is valid if and only if $p(x,\theta)$ is a continuous function of θ; however, it does not need to be a continuous function of x.

- The REINFORCE approach has been greatly developed for reinforcement learning problems (as the name says) in which $p(\cdot,\theta)$ is the policy, x is the trajectory obtained by using the policy $p(\cdot,\theta)$, and $f(x)$ is the discounted reward function.

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`.

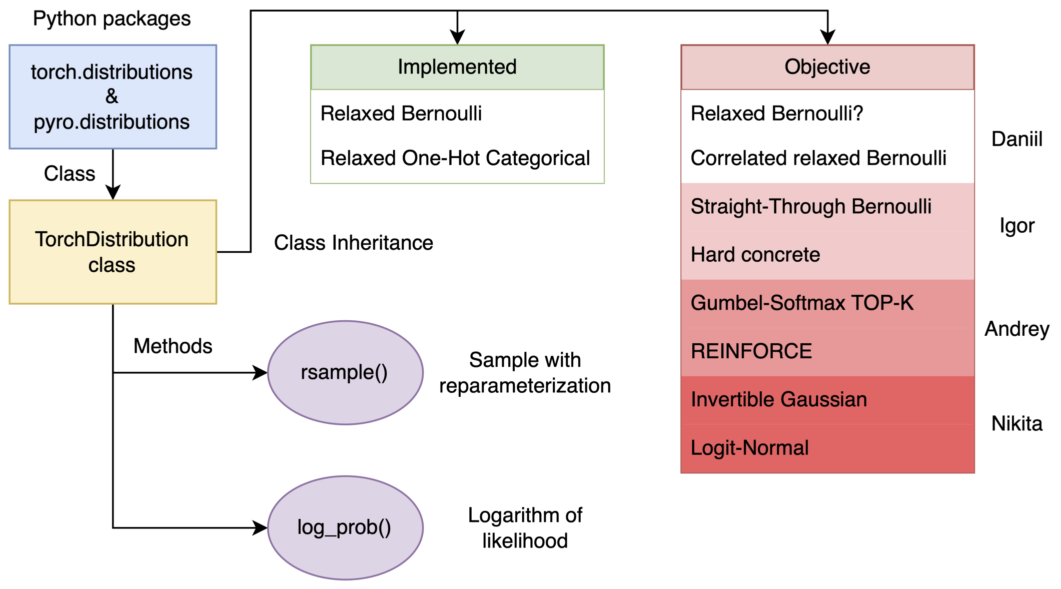
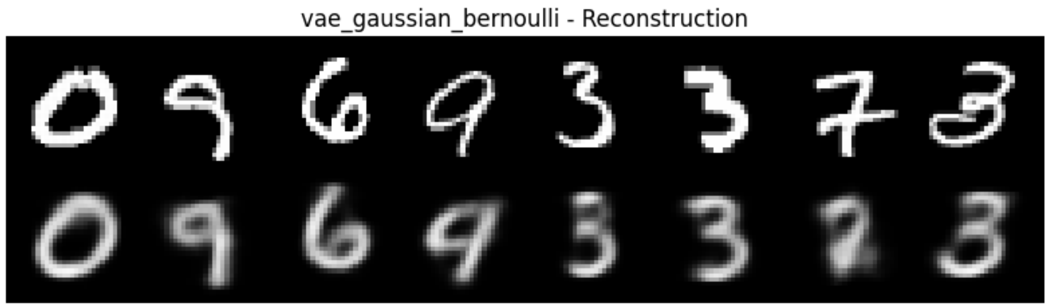


Figure 2: Implementation Scheme

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 techniques 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`.

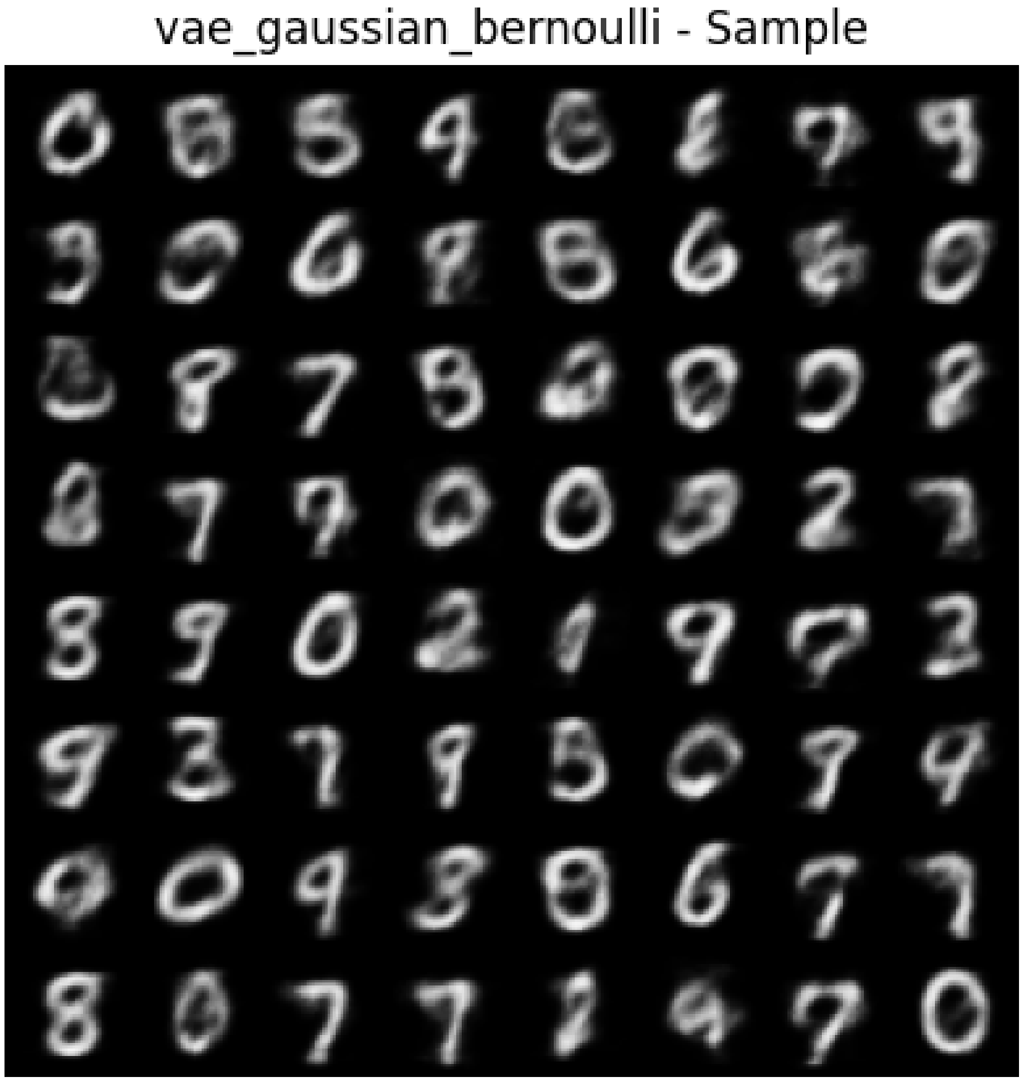
[5] Changhee Lee, Fergus Imrie, and Mihaela van der Schaar. Self-supervision enhanced feature selection with correlated gates. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=oDFvtxzPOx`.

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

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