A Layman's Understanding of Discrete Variational Autoencoders

Erik Paul

September 08, 2023

1 Variational Autoencoders

1.1 Motivation

Given data points $x_1, \ldots, x_n$, our aim is to sample new data points which are in some way similar to $x_1, \ldots, x_n$. For instance, the given data points may be images of cats and our goal is to sample new images of cats.

For variational autoencoders [3, 4], our assumption is that $x_1, \ldots, x_n$ are samples of a random variable $\mathbf{x}$ with an unknown underlying probability distribution $p^*(\mathbf{x})$ over a known space of possible data points, e.g., the space of all images of a certain size. We attempt to approximate $\mathbf{x}$ by defining a family of random variables $\mathbf{x}_\theta$, where θ are the model parameters, and determine θ such that the probability distribution p_θ of $\mathbf{x}_\theta$ best approximates p^*. Our known data points are used to assess how well p_θ approximates p^* for a given set of paramters θ.

1.2 Latent Variables

In practice, effectively sampling from a random variable is difficult, so we sample from simple probability distributions, like the uniform distribution or the normal distribution, and transform their outputs. For this, we model p_θ using latent (unobserved) random variables $\mathbf{z}$ which are easy to sample. The marginal distribution, or *likelihood*, over the observed variables is thus

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}.$$

For variational autoencoders, we write $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})$, choose $p_\theta(\mathbf{z}) = p(\mathbf{z})$ as a very simple distribution without parameters, and model $p_\theta(\mathbf{x}|\mathbf{z})$ using the parameters θ. Modeling $p_\theta(\mathbf{x}|\mathbf{z})$ is usually achieved by using θ to compute the parameters of a distribution which is easy to sample from and choose $\mathbf{x}$ as a sample from this distribution. The model used to produce $\mathbf{x}$ from $\mathbf{z}$ is called the *decoder*.

Example 1 ([3]). Assume $\mathbf{x} \in \{0,1\}^D$ is a vector of D-dimensional binary data. We may model $\mathbf{x} = \mathbf{x}_\theta$ and p_θ as follows.

$$\begin{aligned}
\mathbf{z} &\sim \mathcal{N}(0, \mathbf{I}) \\
p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}; 0, \mathbf{I}) \\
\mathbf{p} &= \mathrm{DecoderNeuralNet}_\theta(\mathbf{z}) \\
x_i &\sim \mathrm{Bernoulli}(p_i) \\
\log p_\theta(\mathbf{x}_\theta|\mathbf{z}) &= \sum_{i=1}^{D} \log \mathrm{Bernoulli}(x_i; p_i) \\
&= \sum_{i=1}^{D} x_i \log p_i + (1 - x_i) \log(1 - p_i),
\end{aligned}$$

where $\mathrm{DecoderNeuralNet}_\theta$ outputs a vector $\mathbf{p} \in [0,1]^D$, for instance by using a sigmoid non-linearity after the last layer, and p_i is used as the probability that $x_i = 1$, i.e., p_i is the parameter of a Bernoulli distribution.

1.3 Conditioning Latent Variables

Although the setup above looks promising, we do not really have a way of training it: the variables $\mathbf{z}$ are not observed, so we do not have training examples telling us how $\mathbf{x}_\theta$ (or p_θ) should look like given $\mathbf{z}$. If the marginal likelihood $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}$ was tractable, i.e., efficiently computable, we could solve this problem by averaging out $\mathbf{z}$ and search θ which maximizes the likelihood of our observed data. However, for models sufficiently flexible to solve real world problems, this integral often does not possess an analytic solution or an efficient estimator. Alternatively, if we knew $p_\theta(\mathbf{z}|\mathbf{x})$, we could compute

$$p_\theta(\mathbf{x}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})} = \frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}.$$

However, $p_\theta(\mathbf{z}|\mathbf{x})$ is usually just as intractable as $\int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}$.

For variational autoencoders, we therefore employ an additional encoding distribution $q_\phi(\mathbf{z}|\mathbf{x})$ in order to approximate $p_\theta(\mathbf{z}|\mathbf{x})$. Modeling $q_\phi(\mathbf{z}|\mathbf{x})$ is usually achieved by using $\mathbf{x}$ and ϕ to compute the parameters of a distribution which is easy to sample from, sample $\mathbf{z}$ from this distribution (q_ϕ), and let $q_\phi(\mathbf{z}|\mathbf{x})$ be the likelihood of $\mathbf{z}$ under this distribution. For example,

$$\begin{gathered}
(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \mathrm{EncoderNeuralNet}_\phi(\mathbf{x}) \\
\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma})) \\
q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}))
\end{gathered}$$

1.4 Optimizing the Evidence Lower Bound

Our goal is still to maximize the likelihood $p_\theta(\mathbf{x})$ over our dataset. For this, we derive the *evidence lower bound (ELBO)* as follows [4].

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\log p_\theta(\mathbf{x})] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] \\
&= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]\right]}_{\substack{=\mathcal{L}_{\theta,\phi}(\mathbf{x})\\ \text{(ELBO)}}} + \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}\left[\log\left[\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right]}_{=D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))} \qquad (*)
\end{aligned}$$

The second term in $(*)$ is the Kullback-Leibler (KL) divergence between $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{z}|\mathbf{x})$. It is always non-negative

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) \geq 0$$

and zero if and only if $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{z}|\mathbf{x})$ coincide. The first term in $(*)$ is called the evidence lower bound

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})].$$

As the KL divergence is non-negative, the ELBO is a lower bound on the log-likelihood of our data:

$$\begin{aligned}
\mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) \\
&\leq \log p_\theta(\mathbf{x}).
\end{aligned}$$

Thus, by maximizing the ELBO, we push $q_\phi(\mathbf{z}|\mathbf{x})$ closer to $p_\theta(\mathbf{z}|\mathbf{x})$ and at the same time maximize $p_\theta(\mathbf{x})$. Note that the ELBO can be rewritten to

$$\begin{aligned}
\mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})).
\end{aligned}$$

This formulation has a simple interpretation as a loss function. The first term is the reconstruction error (or fidelity) and should be maximized: When drawing $\mathbf{z} \sim q_\phi(\cdot|\mathbf{x})$ we want $p_\theta(\mathbf{x}|\mathbf{z})$ to be high. The reconstruction error is also sometimes implemented as taking the L_2-difference between the input $\mathbf{x}$ and a generated sample $\mathbf{x}_\theta$.

The second term is a regularization term which forces the encoder to not encode the training examples as single data points in $\mathcal{Z}$. If our encoder produces parameters for a standard distribution and $p_\theta(\mathbf{z}) = p(\mathbf{z})$ is a known simple distribution, we may compute this divergence analytically. For instance,

$$D_{\mathrm{KL}}(\mathcal{N}(\mu,\sigma^2)||\mathcal{N}(0,1)) = -\log\sigma + \frac{\sigma^2+\mu^2}{2} - \frac{1}{2},$$

which is always non-negative and 0 for $\mu = 0$ and $\sigma = 1$. Alternatively, we could minimize $\mu^2 + (\sigma-1)^2$ instead of the KL divergence, i.e., the L_2 difference between the parameters of $p(\mathbf{z})$ and of $q_\phi(\mathbf{z}|\mathbf{x})$.

1.5 Gradient Estimation

In order to apply gradient descent to optimize the parameters θ and ϕ, we need to compute the gradient of $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ w.r.t. θ and ϕ. For θ, this is easy:

$$\begin{aligned}\nabla_\theta \mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \nabla_\theta \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\nabla_\theta \log p_\theta(\mathbf{x},\mathbf{z})] \\ &\approx \nabla_\theta \log p_\theta(\mathbf{x},\mathbf{z}),\end{aligned}$$

where in the last line, $\mathbf{z}$ is a sample from $q_\phi(\cdot|\mathbf{x})$ to obtain a simple Monte Carlo estimator of the integral.

For ϕ, we cannot proceed in the same way as the expectation is taken w.r.t. the distribution $q_\phi(\cdot|\mathbf{x})$ which itself depends on ϕ. That is, in general we have

$$\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \neq \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\nabla_\phi \log p_\theta(\mathbf{x},\mathbf{z}) - \nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x})].$$

One way to obtain a gradient estimator is via REINFORCE [10], but this estimator has a high variance leading to slow convergence and bad performance. For the derivation of this estimator, consider that

$$\begin{aligned}\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[f(\mathbf{z},\phi)] &= \nabla_\phi \int f(\mathbf{z},\phi) q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} \\ &= \int (\nabla_\phi f(\mathbf{z},\phi)) q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} + \int f(\mathbf{z},\phi) \nabla_\phi q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\nabla_\phi f(\mathbf{z},\phi)] + \int \frac{f(\mathbf{z},\phi)}{q_\phi(\mathbf{z}|\mathbf{x})} q_\phi(\mathbf{z}|\mathbf{x}) \nabla_\phi q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\nabla_\phi f(\mathbf{z},\phi)] + \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}\left[\frac{f(\mathbf{z},\phi)}{q_\phi(\mathbf{z}|\mathbf{x})} \nabla_\phi q_\phi(\mathbf{z}|\mathbf{x})\right].\end{aligned}$$

Thus, we could compute

$$\begin{aligned}&\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}\left[\frac{\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) - 1}{q_\phi(\mathbf{z}|\mathbf{x})} \nabla_\phi q_\phi(\mathbf{z}|\mathbf{x})\right]\end{aligned}$$

However, the high variance of this estimator makes Monte Carlo estimation unreliable. A better way is to apply the *reparameterization trick* [3].

1.6 The Reparameterization Trick

The reparameterization trick is applicable whenever sampling $\mathbf{z} \sim q_\phi(\cdot|\mathbf{x})$ can be expressed by sampling a random variable $\boldsymbol{\epsilon}$ from a distribution p without parameters and computing $\mathbf{z} = g(\phi, \mathbf{x}, \boldsymbol{\epsilon})$. In other words, we need to find a fixed random variable $\boldsymbol{\epsilon}$ which is easy to sample and a function g such that $\mathbf{z} = g(\phi, \mathbf{x}, \boldsymbol{\epsilon})$ has distribution $q_\phi(\cdot|\mathbf{x})$. Examples of distributions for which this is possible are

- "Location-scale" families of distributions in which every distribution of the family is described by a location and a scale parameter and can be obtained from the distribution with location $= 0$ and scale $= 1$. Then $g(\mathbf{x}, \phi, \boldsymbol{\epsilon}) = \text{location}(\mathbf{x}, \phi) + \text{scale}(\mathbf{x}, \phi) \cdot \boldsymbol{\epsilon}$. Examples are Laplace, Elliptical, Student's t, Logistic, Uniform, Triangular, and Gaussian distributions.

- Distributions arising from simple transformations of random variables to which the reparameterization trick applies. For instance, the Log-Normal, Gamma, Dirichlet, Beta, Chi-Squared, and F distributions.
- Distributions whose inverse cumulative distribution functions F^{-1} are tractable. In this case, sampling is achieved by $F^{-1}(\boldsymbol{\epsilon})$ for a uniformly distributed random variable $\boldsymbol{\epsilon} \sim \mathcal{U}(0,1)$.

In the above scenario, we can estimate the gradient by

$$\begin{aligned}\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[f(\mathbf{z},\phi)] &= \nabla_\phi \mathbb{E}_{\boldsymbol{\epsilon}\sim p(\cdot)}[f(g(\phi,\mathbf{x},\boldsymbol{\epsilon}),\phi)] \\ &= \mathbb{E}_{\boldsymbol{\epsilon}\sim p(\cdot)}[\nabla_\phi f(g(\phi,\mathbf{x},\boldsymbol{\epsilon}),\phi)] \\ &\approx \nabla_\phi f(g(\phi,\mathbf{x},\boldsymbol{\epsilon}),\phi).\end{aligned}$$

In particular, we can estimate

$$\nabla_\phi \mathcal{L}_{\theta,\phi}(\mathbf{x}) \approx \nabla_\phi(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})),$$

where $\mathbf{z} = g(\phi,\mathbf{x},\boldsymbol{\epsilon})$ and $\boldsymbol{\epsilon} \sim p(\cdot)$ is a sample.

1.7 Computing $q_\phi(\mathbf{z}|\mathbf{x})$

By the substitution formula and the inverse function theorem, we have for invertible $g(\phi,\mathbf{x},\cdot)$ that

$$q_\phi(\mathbf{z}|\mathbf{x}) = p(\epsilon)\left|\det\left(\frac{\partial\boldsymbol{\epsilon}}{\partial\mathbf{z}}\right)\right| = p(\epsilon)\left|\det\left(\frac{\partial\mathbf{z}}{\partial\boldsymbol{\epsilon}}\right)\right|^{-1}$$

such that

$$\log q_\phi(\mathbf{z}|\mathbf{x}) = \log p(\epsilon) - \log\left|\det\left(\frac{\partial\mathbf{z}}{\partial\boldsymbol{\epsilon}}\right)\right|^{-1}.$$

Here,

$$\frac{\partial\mathbf{z}}{\partial\boldsymbol{\epsilon}} = \frac{\partial(z_1,\dots,z_k)}{\partial(\epsilon_1,\dots,\epsilon_k)} = \begin{pmatrix}\frac{\partial z_1}{\partial\epsilon_1} & \cdots & \frac{\partial z_1}{\partial\epsilon_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_k}{\partial\epsilon_1} & \cdots & \frac{\partial z_k}{\partial\epsilon_k}\end{pmatrix}$$

is the Jacobian as usual.

2 Discrete Variational Autoencoders

So far, we considered $\mathbf{z}$ to be continuous variables, e.g., vectors from $\mathbb{R}^D$. However, from an information theory point of view, a single scalar from $\mathbb{R}$ can encode just as much information as a vector from $\mathbb{R}^D$ since $|\mathbb{R}^D| = |\mathbb{R}|$. Thus, in order to view $\mathbf{z}$ as a form of compressed information about $\mathbf{x}$, it makes sense to consider a discrete or finite latent space $\mathcal{Z}$. For example, we might consider $\mathbf{z}$ to be a binary vector with values in $\{0,1\}^D$.

More generally, we could assume a fixed latent space $\mathcal{Z} = \{z_1,\dots,z_M\}$ and have our encoder produce a vector $\mathbf{p} \in [0,1]^M$ from $\mathbf{x}$ and ϕ such that $\sum p_i = 1$, for instance by using a softmax layer. Then we sample $\mathbf{z}$ according to $\mathbb{P}(\mathbf{z} = z_i) = p_i$. We can implement this using a function $g(\phi, \mathbf{x}, \epsilon) = z_i$ iff $\epsilon \in [\sum_{j=0}^{i-1} p_j, \sum_{j=0}^{i} p_j]$ with a random variable $\epsilon \sim \mathcal{U}(0,1)$. If p_i is differentiable w.r.t. ϕ, it is in particular continuous and as g has only finitely many discontinuities, we have $\nabla_\phi g(\phi, \mathbf{x}, \epsilon) = 0$ almost surely. We also see that $q_\phi(\mathbf{z}|\mathbf{x}) = \sum p_i \mathbb{1}_{z_i}(\mathbf{z})$. If we write the ELBO as $\mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$ and use the estimation via the reparameterization trick, we obtain

$$\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] \approx \frac{1}{p_\theta(\mathbf{x}|\mathbf{z})} \frac{\partial p_\theta(\mathbf{x}|\mathbf{z})}{\partial \mathbf{z}} \nabla_\phi \mathbf{z} = 0$$

Thus, we have

$$\nabla_\phi \mathcal{L}_{\theta,\phi}(\mathbf{x}) \approx -\nabla_\phi D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$$

and the variables ϕ are only encouraged to minimize the KL divergence between $q_\phi(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$. This is minimized for $q_\phi(\mathbf{z}|\mathbf{x}) = p(\mathbf{z})$ at which point no information about $\mathbf{x}$ is conveyed in $\mathbf{z}$. Thus, any reparameterization with $\nabla_\phi \mathbf{z} = 0$ is unsuitable.

The general problem with differentiating a function f defined on a finite set $\mathcal{Z}$ is that derivatives do not make any sense when there is no order on the elements of $\mathcal{Z}$. If the latent space consists of $\{1, \ldots, M\}$, derivatives of f are not defined but make sense as we may want to increase or decrease the input $\mathbf{z}$ and need to know what works best for f. However, if the the latent space consists of $\{0, (0,0), (0,0,0), \ldots\}$, there is no use in differentiating f and gradient descent is not a suitable optimization approach. Therefore, discrete variational autoencoders trained using gradient descent require some structure on the latent space which allow an embedding into a continuous space. Typical choices include a set $\mathcal{Z}$ of one-hot vectors or a set of binary vectors.

2.1 The Straight-Through Estimator

The straight-through estimator [7] ignores the expectation and defines the gradient of $\mathbf{z}$ as $\nabla_\phi q_\phi(\mathbf{z}|\mathbf{x})$. That is, we define

$$\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[f(\mathbf{z})] = \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot|\mathbf{x})}[\nabla_{\mathbf{z}} f(\mathbf{z}) \nabla_\phi q_\phi(\mathbf{z}|\mathbf{x})]$$

and estimate this using Monte Carlo. There is no good mathematical justification for this, as depending on the choice of $\mathcal{Z}$, differentiating f w.r.t. $\mathbf{z}$ may be completely nonsensical. However, it may work in practice.

2.2 Gumbel-Softmax

For the *Gumbel-softmax* trick [2], we assume the latent space $\mathcal{Z}$ to consist of M one-hot vectors from $\{0,1\}^M$ where we sample the unit vector e_j with probability p_j. If $g_1, \ldots, g_M$ are Gumbel$(0,1)$-distributed, then $\mathbb{P}[j = \arg\max_i(g_i + \log p_i)] = p_j$, i.e., $\mathbf{z} = e_{\arg\max_i(g_i + \log p_i)}$ has the required distribution. As the arg max is not differentiable, we apply the softmax with an annealed (during training) temperature τ, i.e., we define

$$z_i = \frac{e^{\frac{\log(p_i)+g_i}{\tau}}}{\sum_{j=1}^{M} e^{\frac{\log(p_j)+g_j}{\tau}}}.$$

For $\tau \to 0$, the vector $(z_1, \dots, z_M)$ approaches the one-hot $\arg\max$ vector with our desired distribution. Note that this increases the variance of the vector.

2.3 Smoothing Transformations

An approach investigated in a series of papers [6, 9, 8] is to draw continuous random variables ζ depending on $\mathbf{z}$. Ignoring the mathematical motivation provided by these papers, the gist is the following. We assume that $\mathcal{Z} = \{0,1\}^M$ is a set of binary vectors and p_i is the probability for the ith entry to be 1. This means for $\mathbf{z} = (z_1, \dots, z_M)$, we have $q_\phi(\mathbf{z}|\mathbf{x}) = \prod_{i=1}^M \Big(p_i z_i + (1-p_i)(1-z_i)\Big)$. Each entry z_i is sampled using the uniform distribution $\epsilon_i \sim \mathcal{U}(0,1)$ but instead of using a step function $z_i = \mathbb{1}_{[1-p_i,1]}(\epsilon_i)$, we generate $z_i = f(\beta, p_i, \epsilon_i)$ using a function f such that

- $f(\beta, p_i, .)$ is increasing, $f(\beta, p_i, 0) = 0$, and $f(\beta, p_i, 1) = 1$,
- $f(\beta, p_i, .)$ converges pointwise to $\mathbb{1}_{[1-p_i,1]}$ for $\beta \to B \in \mathbb{R} \cup \{\infty\}$, and
- $f(\beta, p_i, .)$ is differentiable in all but finitely many points and its derivative is not 0.

The parameter β is annealed towards B during training. One example is obtained via the spike-and-exponential smoothing transformation [6], see also Figure 1

$$f_1(\beta, p, \epsilon) = \mathbb{1}_{[1-p,1]}(\epsilon) \cdot \frac{1}{\beta} \log_a \left(\frac{\epsilon + p - 1}{p} \cdot (a^\beta - 1) + 1 \right)$$

where a is an arbitrary base, for instance e or 2. This function is 0 on the interval $[0, 1-p]$ and then grows from 0 to 1 in a logarithmic curve on $[1-p, 1]$. Similar functions are

$$\begin{aligned}
f_2(\beta, p, \epsilon) &= \mathbb{1}_{[1-p,1]}(\epsilon) \cdot \left(1 - \left(1 - \frac{\epsilon + p - 1}{p} \right)^\beta \right) \\
f_2'(\beta, p, \epsilon) &= \mathbb{1}_{[1-p,1]}(\epsilon) \cdot \sqrt[\beta]{\frac{\epsilon + p - 1}{p}} \\
f_2''(\beta, p, \epsilon) &= \mathbb{1}_{[1-p,1]}(\epsilon) \cdot \left(1 - \frac{1-p}{\beta(\epsilon - p)} \right)
\end{aligned}$$

for $\beta \to \infty$. A more symmetrical choice, inspired by the exponential transformation and the power-function transformation [8], are the functions

$$\begin{aligned}
f_3(\beta, p, \epsilon) &= \mathbb{1}_{[0,1-p]}(\epsilon) \cdot (1-p) \left(\frac{\epsilon}{1-p} \right)^\beta + \mathbb{1}_{[1-p,1]}(\epsilon) \cdot \left(1 - p \left(\frac{1-\epsilon}{p} \right)^\beta \right) \\
f_3'(\beta, p, \epsilon) &= \mathbb{1}_{[0,1-p]}(\epsilon) \cdot (p-1) \left(\sqrt[\beta]{-\frac{\epsilon + p - 1}{1-p}} - 1 \right) + \mathbb{1}_{[1-p,1]}(\epsilon) \cdot \left(1 + p \left(\sqrt[\beta]{-\frac{\epsilon + p - 1}{p}} - 1 \right) \right)
\end{aligned}$$

for $\beta \to \infty$. The derivatives of f_3 are 0 in 0 and 1 and n in $1-p$.

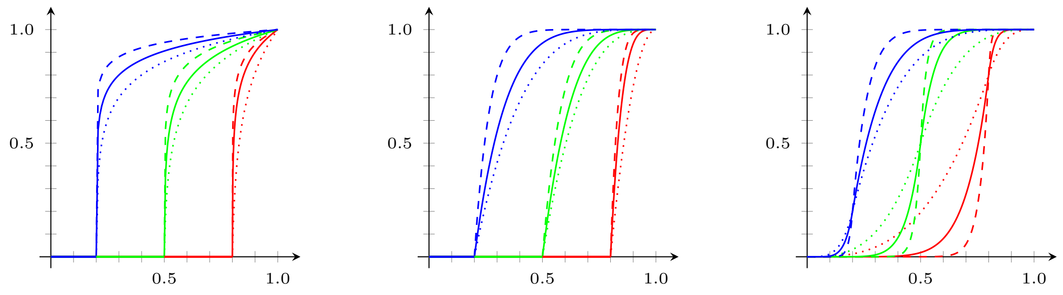

Figure 1: The functions f_1, f_2, f_3 for varying values of β (dotted $<$ solid $<$ dashed) and p (0.2, 0.5, 0.8).

2.4 Vector Quantization

Vector Quantized Variational Autoencoders (VQ-VAEs) [5] differ from other discrete variational autoencoders in that the latent space is learned. The latent space is of the form $\mathcal{Z} = \{z_1, \dots, z_M\} \subseteq \mathbb{R}^D$, the encoder produces outputs $E_\phi(\mathbf{x}) \in \mathbb{R}^D$ and the quantization is achieved by $\mathbf{z} = \arg\min_{z \in \mathcal{Z}} ||E_\phi(\mathbf{x}) - z||_2$. Note that no actual sampling takes place in the generation of $\mathbf{z}$. This is equivalent to defining

$$q_\phi(\mathbf{z}|\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{z} = \arg\min_{z \in \mathcal{Z}} ||E_\phi(\mathbf{x}) - z||_2 \\ 0 & \text{otherwise.} \end{cases}$$

The prior $p(\mathbf{z})$ on $\mathcal{Z}$ is assumed to be uniform. The KL divergence $D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$ thus becomes

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) = \sum_{z \in \mathcal{Z}} q_\phi(z|\mathbf{x}) \log\left(\frac{q_\phi(z|\mathbf{x})}{p(z)}\right) = \log M$$

and can be ignored in the ELBO. The decoder produces an output using the quantized latent $\mathbf{z}$ and otherwise functions as usual.

For training, a three-part loss is minimized.

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = -\log p_\theta(\mathbf{x}|\mathbf{z}) + ||\mathrm{sg}[E_\phi(\mathbf{x})] - e||_2^2 + \beta||E_\phi(\mathbf{x}) - \mathrm{sg}[\mathbf{z}]||_2^2 \qquad (**)$$

Here, sg denotes the stop gradient operator which stops gradients from flowing into its argument. The first term in $(**)$ is (a Monte Carlo estimator of) the reconstruction loss, i.e., the ELBO without the now constant KL divergence term. During the forward pass, the decoder obtains the quantized latent $\mathbf{z}$, but during backpropagation $\nabla_\phi \mathbf{z}$ is replaced with $\nabla_\phi E_\phi(\mathbf{x})$ since arg min is not differentiable. The elements of the latent space therefore do not obtain gradients from the reconstruction loss. This constitutes an application of straight-through estimation discussed earlier. The second term in $(**)$ is the *codebook loss* and pushes the latents closer to the encoder outputs. The third term in $(**)$ is the *commitment loss* and encourages the encoder to commit to a codebook. The coefficient β is a hyperparameter and weighs the commitment loss.

As usual, the reconstruction loss is sometimes implemented as $||D_\theta(\mathbf{z}) - \mathbf{x}||_2^2$, where $D_\theta(\mathbf{z})$ is a sample of the decoder. The codebook loss can be replaced by exponential moving average updates, i.e.,

$$
\begin{aligned}
N_i^{(t)} &= N_i^{(t-1)} \cdot \gamma + n_i^{(t)}(1-\gamma) \\
m_i^{(t)} &= m_i^{(t-1)} \cdot \gamma + \sum_j^{n_i^{(t)}} E_\phi(\mathbf{x})_{i,j}^{(t)}(1-\gamma) \\
z_i^{(t)} &= \frac{m_i^{(t)}}{N_i^{(t)}},
\end{aligned}
$$

where $n_i^{(t)}$ is the number of vectors $E_\phi(\mathbf{x})$ in the minibatch which are quantized to $z_i^{(t-1)}$, $E_\phi(\mathbf{x})_{i,1}^{(t)}, E_\phi(\mathbf{x})_{i,2}^{(t)}, E_\phi(\mathbf{x})_{i,3}^{(t)}, \ldots$ are quantized to $z_i^{(t-1)}$, and γ is a decay parameter in $(0, 1)$, e.g., $\gamma = 0.99$.

References

[1] Evidence lower bound. `https://en.wikipedia.org/wiki/Evidence_lower_bound`.

[2] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[4] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[5] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

[6] Jason Tyler Rolfe. Discrete variational autoencoders. *arXiv preprint arXiv:1609.02200*, 2016.

[7] John Thickstun. Discrete VAE's. `https://courses.cs.washington.edu/courses/cse599i/20au/resources/L09_discretevae.pdf`.

[8] Arash Vahdat, Evgeny Andriyash, and William Macready. DVAE#: Discrete variational autoencoders with relaxed Boltzmann priors. *Advances in Neural Information Processing Systems*, 31, 2018.

[9] Arash Vahdat, William Macready, Zhengbing Bian, Amir Khoshaman, and Evgeny Andriyash. DVAE++: Discrete variational autoencoders with overlapping transformations. In *International conference on machine learning*, pages 5035–5044. PMLR, 2018.

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