Variational Autoencoders For (Classical) Statisticians

Vafa Behnam Roudsari

May 20, 2019

1 From Maximum Likelihood to Kullback-Leibler

Because the variational autoencoder has so many constituent parts, I defer to the end of this paper the motivation, as I feel like one can truly understand why such a model is useful only after the necessary foundations have been built up.

For now, suppose we want to learn a continuous distribution p(x).

We are all familiar with the maximum likelihood approach. Given data, and given a parametric form, e.g. Gaussian, for the marginals $p_\theta(x_i)$, we can learn the parameters of the model through maximization.

$$\max_\theta L(x;\theta) = \max_\theta \prod_{i=1}^{N} \hat{p}_\theta(x_i) = \max_\theta \sum_{i=1}^{N} \log \hat{p}_\theta(x_i)$$

Now, something interesting happens when we introduce the true distribution $p(x)$ into this equation. Since the true distribution is a "constant" with respect to θ, throwing it into the mix does not affect the maximization problem.

$$\max_\theta \sum_{i=1}^{N} \log \hat{p}_\theta(x_i) = \max_\theta \sum_{i=1}^{N} \log \hat{p}_\theta(x_i) - \sum_{i=1}^{N} \log p(x) = \max_\theta \sum_{i=1}^{N} \log \frac{\hat{p}_\theta(x_i)}{p(x_i)}$$

Again, introducing a constant $1/N$ does not change the maximization problem.

$$= \max_\theta \frac{1}{N} \sum_{i=1}^{N} \log \frac{\hat{p}_\theta(x_i)}{p(x_i)} \xrightarrow{N} \max_\theta \mathbb{E}_{x\sim p}\left[\log \frac{\hat{p}_\theta(x)}{p(x)}\right]$$

Where the last result holds due to Law of Large Numbers.

$$\max_\theta \mathbb{E}_{x\sim p}\left[\log \frac{\hat{p}_\theta(x)}{p(x)}\right] = \min_\theta \mathbb{E}_{x\sim p}\left[\log \frac{p(x)}{\hat{p}_\theta(x_i)}\right] = \min_\theta D_{KL}(p||\hat{p}_\theta)$$

The last equation is actually an information-theoretic concept called the Kullback-Leibler divergence. We therefore see that minimizing the KL divergence is (asymptotically) equivalent to maximizing the negative log likelihood. This shift in perspective will be motivated by the theory of variational inference and latent variables. But intuitively it captures how

much two distributions "diverge" from each other, and the KL-divergence of two distributions equals 0 if and only if the two are identical.

2 Latent Modeling

We now have the optimization problem

$$\max_{\theta} \mathbb{E}_{x \sim p} \left[\log \frac{\hat{p}_\theta(x)}{p(x)} \right]$$

What form should $\hat{p}_\theta(x) \in \mathcal{P}$ be? Ideally, in the vein of classical machine learning tradeoffs, we want $\mathcal{P}$ to be flexible enough so it can accurately approximate the original distribution (but not too flexible so that it overfits), yet we also want it to be reasonably computationally tractable. This is a combination of the famous bias-variance tradeoff, mixed with concerns about algorithmic runtime.

Usually, only a handful of distributions, the "canonical" ones (e.g. normal, poisson, exponential) etc. have densities which are computationally tractable. Yet restricting ourselves to only these distributions may not accurately approximate $p(x)$. A happy medium is to leverage latent variables. While latent variables might be from the same "simple" distributions as $p_\theta(x)$, mixing these together will allow us to represent more complicated distributions, without making the problem computationally intractable. Note that we actually don't observe the latent variables in our current dataset.

For simplicity, let us assume that there is only one latent variable z, and that z is discrete. We are all familiar with the law of total probability,

$$p(x) = \sum_{k=1}^{M} p(x|z_k) * p(z_k)$$

The choice of the form of $p(x|z_k)$ and $p(z_k)$ is a paper in and of itself, but usually two important considerations involve prior knowledge, and ease of computation. To help motivate why we might want to use latent variables, we will use the famous mixture of Gaussians model (set $p(x|z_k)$ to be Gaussian and $p(x)$ to be a simple vector of probabilities π_k).

$$\sum_{k=1}^{M} p(z_k)p(x|z_k) = \sum_{k=1}^{M} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

While a contrived example, this is hardly computationally intensive as computing densities of Gaussians is a cheap operation in most software. Yet we can design far more complicated models than if we just restricted ourselves to simple marginal distributions.

Note that now we are maximizing for a tuple $(p_\theta(x|z), p_\theta(z))$ from the joint distribution of $\mathcal{P}_{x|z} \times \mathcal{P}_z$. Now that we have made some progress with respect to $\mathcal{P}$, now actually let us

return to our original problem. Letting z now be continuous, and applying the law of total probability,

$$\max_\theta \mathbb{E}_{x\sim p}\left[\log \frac{\hat{p}_\theta(x)}{p(x)}\right] = \max_\theta \mathbb{E}_{x\sim p}\left[\log \frac{\int p_\theta(x|z)p_\theta(z)dz}{p(x)}\right] (1)$$

The integral looks ugly. And what we have gained in expressiveness we have lost in concavity (which makes our optimization problem much harder). But we have tricks from the field of variational inference that can help with computing this.

3 Variational Inference

The whole point of variational techniques is to approximate a distribution $p(x)$ via minimizing some notion of "dissimilarity" (one of which is the Kullback-Leibler divergence) between said $p(x)$ and a $\hat{p}(x)$ belonging to a class of (usually tractable) distributions $\mathcal{P}$. As might be apparent, this is a hard problem, but variational inference uses multiple techniques to make this tractable.

To reduce clutter, let us just focus on the numerator of (1),

$$\log \int p_\theta(x|z)p_\theta(z)dz = \log \int p_\theta(x,z)dz = \log \int p_\theta(x,z) * \frac{q_\lambda(z|x)}{q_\lambda(z|x)}dz$$

Now this is where the variational magic comes in. It will shortly be clear why we introduce another parametrized distribution $q_\lambda(z|x) \in \mathcal{Q}$.

Using a result in convex analysis called Jensen's inequality, which states $\forall$ concave functions ϕ, $\phi(\mathbb{E}[X]) \geq \mathbb{E}[\phi(x)]$. Since the logarithm is concave, we can apply Jensen's inequality.

$$\log \mathbb{E}_{z|x\sim q}\left[\frac{p_\theta(x,z)}{q_\lambda(z|x)}\right] \geq \mathbb{E}_{z|x\sim q}\left[\log \frac{p_\theta(x,z)}{q_\lambda(z|x)}\right]$$

The right hand side is therefore a lower bound for our problem. While maximizing this quantity does not necessarily imply that we have maximized our original problem, it turns out this quantity has an interesting interpretation.

$$(2)\max_\lambda \mathbb{E}_{z|x\sim q}\left[\log \frac{p_\theta(x,z)}{q_\lambda(z|x)}\right] = \max_\lambda \mathbb{E}_{z|x\sim q}\left[\log \frac{p_\theta(x)p_\theta(z|x)}{q_\lambda(z|x)}\right] = \max_\lambda \mathbb{E}_{z|x\sim q}\left[\log p_\theta(x) + \log \frac{p_\theta(z|x)}{q_\lambda(z|x)}\right]$$

$$= \max_\lambda \left(\int q_\lambda(z|x)\log p_\theta(x)dz + \int q_\lambda(z|x)\log \frac{p_\theta(z|x)}{q_\lambda(z|x)}dz\right)$$

$$= \max_\lambda \left(\log(p_\theta(x))\int q_\lambda(z|x)dz + \int q_\lambda(z|x)\log \frac{p_\theta(z|x)}{q_\lambda(z|x)}dz\right)$$

Since $q_\lambda(z|x)$ is a density, it integrates to one, and since $p_\theta(x)$ is a "constant" with respect to q, the maximization only depends on the second term.

$$= \log p_\theta(x) + \max_\lambda \left(\int q_\lambda(z|x)\log \frac{p_\theta(z|x)}{q_\lambda(z|x)}dz\right) = \log p_\theta(x) - \min_\lambda \left(\int q_\lambda(z|x)\log \frac{q_\lambda(z|x)}{p_\theta(z|x)}dz\right)$$

Where we have simply converted the maximization problem to a minimization one by multiply -1 to the objective.

$$= \log p_\theta(x) - \min_\lambda D_{KL}(q_\lambda(z|x) \;||\; p_\theta(z|x)) \quad (3)$$

We now see our old friend, the Kullback-Leibler divergence. We can therefore see that maximizing (2) is equivalent to seeking out a conditional distribution $q \in \mathcal{Q}(z|x)$ that is as close as possible to $p_\theta(z|x)$.

Plugging (2) in place of the troublesome integral in our original optimization problem (1): (after omitting constants and cleaning up the notation a bit, which does not affect the original optimization problem)

$$= \max_{\theta,\lambda} \mathbb{E}_{x\sim p}\left[\mathbb{E}_{z|x\sim q}\left[\log \frac{p_\theta(x,z)}{q_\lambda(z|x)}\right]\right] (4)$$

This is called the evidence lower bound (ELBO). If we plug in (3) to (4),

$$= \max_{\theta} \mathbb{E}_{x\sim p}\left[\log p_\theta(x) - \min_\lambda D_{KL}(q_\lambda(z|x) \;||\; p_\theta(z|x))\right] (5)$$

Now it is a bit clearer what this problem is actually maximizing for. Remember, we maximize (5) instead of just the $\log p_\theta(x)$ term directly because $(p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz)$ is assumed to be intractable. In a sense, the price we pay for tractability is the "error" given by the $D_{KL}(q|p)$ term. This "error" approaches 0 as the two distributions converge, which appears to be a realistic goal if we have a reasonably rich set of distributions $q \in \mathcal{Q}$.

There are two other indirect benefits of using this loss function as well. The authors of the original paper suggest that the D_{KL} term acts as a regularizer (as in it prevents the learning of trivial functions or overfitting). As a final added bonus, we may learn a useful approximation of the posterior distribution $p_\theta(z|x)$ along the way.

4 Autoencoders

Now that we have our optimization objective (or equivalently our loss function), we can turn to the autoencoder framework. This is where the fancy deep learning stuff comes in.

Autoencoders are a form of unsupervised learning, i.e. learning from unlabelled datasets, and are feedforward neural networks. Autoencoders aim to deconstruct data through the encoder, and then reconstruct it through a decoder. The goal is therefore to maximize the similarity between the input and output. If the encoder portion sounds like a Principal Component Analysis, it is, if we use the mean square error as our loss function and one hidden layer. Indeed, shallow (i.e. one hidden, linear layer) autoencoder architectures are basically inefficient implementations of PCA. So why bother? If we allow more flexible functional forms, and more hidden layers, then autoencoders can reconstruct more than just linear manifolds in higher-dimensional space.

Let the output of the neural network be $y = \sigma_m(...(W_2 * \sigma_1(W_1 * x + b_1) + b_2) + ...)$ where m represents the number of hidden layers, the number of rows of W_i represents the number of

neurons in layer i, and the number of columns represent the number of neurons in layer $i-1$ (if $i-1=0$, then its just the length of the input vector $x \in \mathcal{X}$). Let $\phi : \mathcal{X} \mapsto \mathcal{F}$ be the encoder and $\psi : \mathcal{F} \mapsto \mathcal{X}$ be the decoder, so $y = \psi(\phi(x))$. If we denote the output of the encoder as z, and the bottleneck layer by r, then $z = \phi(x) = \sigma_{m-r}(...(\sigma_1(W_1 * x + b_1) + ...) + b_{m-r})$ and $\psi(z) = \sigma_m(...(\sigma_{(m-r)+1}(W_{(m-r)+1} * z + b_{(m-r)+1}) + ...)$.

We can therefore see the parameters of our model are all the weights $(W_i)_{i=1}^m$ and biases $(b_i)_{i=1}^m$. Let us decongest this notation and represent the parameters in the encoder ϕ by $\lambda = ((W_i)_{i=1}^{m-r}, (b_i)_{i=1}^{m-r})$ and the parameters of the decoder ψ by $\theta = ((W_i)_{i=(m-r)+1}^{m-1}, (b_i)_{i=(m-r)+1}^{m-1})$.

The dimensions of the feature space $\mathcal{F}$ can be lower, greater or equal to $\mathcal{X}$. In simple applications of autoencoders, usually $\mathcal{F}$ is set to be less than that of $\mathcal{X}$. This is because when $dim(\mathcal{F}) \geq dim(\mathcal{X})$, and without proper regularization (explicit or otherwise) the risk of learning the identity function, which is useless, becomes non-negligible, i.e. $\psi_\theta(\phi_\lambda(x)) = x$.

Using the loss function, we can impose a probabilistic or non-probabilistic interpretation onto the network. If we use the mean-squared error, which is sensible for continuous variables, the autoencoder will be a non-probabilistic discriminative model.

$$\min_{(W_i)_{i=1}^m, (b_i)_{i=1}^m} ||x - \sigma_m(...(W_2 * \sigma_1(W_1 * x + b_1) + b_2) + ...)||_2^2 = \min_{\theta,\lambda} ||x - \psi_\theta(\phi_\lambda(x))||_2^2$$

We can throw in a regularization term into this loss function to counteract overfitting. The regularization can penalize multiple things, such as the impact of an activation function, or the quantitative size of derivatives. The former is called a sparse autoencoder, and the latter is a contractive autoencoder. Another regularization trick that is not as explicit is the insertion of noise on the input that is fed into the encoding and decoding functions, which results in the appropriately named denoising autoencoder.

However, to keep consistency with the prior loss functions introduced before, we will utilize the probabilistic likelihood approach. Let us now merge our notation from the variational inference section. Our deterministic encoder function $z = \phi_\lambda(x)$ is now the probabilistic $q_\lambda(z|x)$, and our decoder $\psi_\theta(z)$ corresponds to $p_\theta(x|z)$. We can interpret the output of the encoder z as a form of latent variable.

How do neural networks, or autoencoders in this particular case, learn a distribution? One way is to set q_λ and p_θ to some distribution, and let the parameters λ, θ be determined by the data, which is passed through the neural network. This will allow us to model complicated relationships, and is why we bring in the neural network in the first place. We can express the negative log likelihood as,

$$\min_{\theta,\lambda} \sum -\log L(x|\psi_\theta(\phi_\lambda(x))) = \min_{\theta,\lambda} \sum -\log p_\theta(x|\phi_\lambda(x))$$

$$= \min_{\theta,\lambda} \sum -\log p_\theta(x|z) \text{ where } z \sim q_\lambda(z|x)$$

If the activation functions σ_i's are linear, we can derive analytic solutions for this particular optimization problem. Furthermore, the loss function is quadratic (which implies

convexity) and hence we are assured any minima are global minima (and that there are no other complicated structures such as saddle points).

But the whole point of autoencoders is to have non-linear activation functions, and usually in such cases we won't be able to analytically solve for θ, λ. Since the autoencoder is a feedforward neural network (with the little caveat that the input x is also the target of optimization), we turn to popular iterative gradient-based methods to learn the parameters of the model (a detailed discussion of gradient optimization is beyond the scope of this paper).

$$\theta_{i+1} = \theta_i + \alpha_i \nabla_{\theta_i} \log p_{\theta_i}(x|z) \text{ until convergence is achieved}$$

α_i is the step size. ∇_{θ_i} can be computed via the famous backpropagation algorithm, or through Monte Carlo sampling techniques (we will see both these approaches in solving the variational autoencoder problem)

We can therefore see that the vanilla autoencoder framework is discriminative, both probabilistic or non-probabilistic (we learn a function $f(y|x)$ or density $p(y|x)$ but not the more general joint density $p(y, x)$). Introducing the variational element is therefore a particular example of how adopting a probabilistic generative perspective over a discriminative one can stimulate new insights to problems.

5 ELBO Loss Autoencoders

This autoencoder language of "encoding" and "decoding" sounds makes it seem like information theoretic loss functions are very applicable to this problem. Let us return to the ELBO loss function we painstakingly developed in the variational inference section. If we take (5) and apply Bayes Rule and do some algebra, we get:

$$\max_{\theta} \mathbb{E}_{x\sim p}\left[\mathbb{E}_{z|x\sim q}\left[\log p_\theta(x|z)\right] - \min_{\lambda} D_{KL}(q_\lambda(z|x) \;||\; p_\theta(z))\right] (6)$$

The dependence on the encoder and decoder is more apparent in this formulation. Given the output of the encoder z, the $\log p_\theta(x|z)$ term computes the reconstruction loss of having our decoder trying to reconstruct x. The KL term here can be interpreted as a regularizer term enforcing the candidate distribution to look like the prior (which may prevent the learning of the trivial identity function).

Let us try to optimize for θ, λ, using equation (4)'s formulation of ELBO. As intimated before, due to the non-linearity of our model, we use gradient descent to solve for the parameters of our model. Let us begin with θ,

$$\nabla_\theta \mathbb{E}_{x\sim p}\left[\mathbb{E}_{z|x\sim q}\left[\log \frac{p_\theta(x,z)}{q_\lambda(z|x)}\right]\right] = \mathbb{E}_{x\sim p}\left[\nabla_\theta \mathbb{E}_{z|x\sim q}\left[\log \frac{p_\theta(x,z)}{q_\lambda(z|x)}\right]\right]$$

(where this equality holds due to Leibniz rule, which will be explained more in a bit). For conciseness, we omit the outer expectation $\mathbb{E}_{x\sim p}$:

$$\nabla_\theta \, \mathbb{E}_{z|x\sim q}\left[\log \frac{p_\theta(x,z)}{q_\lambda(z|x)}\right] = \nabla_\theta \int q_\lambda(z|x) \log \frac{p_\theta(x,z)}{q_\lambda(z|x)} dz$$

$$\doteq \int \nabla_\theta \left(q_\lambda(z|x) \log \frac{p_\theta(x,z)}{q_\lambda(z|x)} \right) dz \overset{*}{=} \int q_\lambda(z|x) \nabla_\theta \left(\log \frac{p_\theta(x,z)}{q_\lambda(z|x)} \right) dz$$

If we assume $q_\lambda(z|x)$ and $p_\theta(x,z)$ are C^1 (i.e. a function whose first derivative is continuous), then $\doteq$ holds due to Leibniz Rule.[1] (we are allowed to interchange derivatives and integrals if the differentials of the integral (dz) and of the derivative ($d\theta$) are different). $\overset{*}{=}$ holds because $q_\lambda(z|x)$ does not depend on ∇_θ (i.e. $\nabla_\theta\big(q_\lambda(z|x) * f_\theta(x)\big) = q_\lambda(z|x)\nabla_\theta f_\theta(x)$).

$$= \mathbb{E}_{z|x\sim q}\left[\nabla_\theta \log \frac{p_\theta(x,z)}{q_\lambda(z|x)}\right] = \mathbb{E}_{z|x\sim q}\left[\nabla_\theta \log p_\theta(x,z)\right]$$

The last equality holds because $\nabla_\theta \log q_\lambda(z|x) = 0$

We can approximate this gradient by using Monte Carlo sampling methods. We will not discuss these techniques in depth, but what is important is that Monte Carlo methods can approximate expectations, by repeated sampling. (This is one reason why we require the distribution q to be easily sampled from.)

$$\mathbb{E}_{z|x\sim q}\left[\nabla_\theta \log p_\theta(x,z)\right] \approx \frac{1}{N}\sum_{j=1}^{N} \nabla_\theta \log p_\theta(x,z_j) \text{ where } z_j \sim q_\lambda(z|x)$$

We therefore have a method to optimize for θ. And this turns out to be an unbiased estimator. Let us try now for λ.

$$\nabla_\lambda \, \mathbb{E}_{z|x\sim q}\left[\log \frac{p_\theta(x,z)}{q_\lambda(z|x)}\right] = \nabla_\lambda \int q_\lambda(z|x) \log \frac{p_\theta(x,z)}{q_\lambda(z|x)} dz$$

$$= (7) \int \nabla_\lambda \left(q_\lambda(z|x) \log \frac{p_\theta(x,z)}{q_\lambda(z|x)} \right) dz \overset{!!!}{\neq} \int q_\lambda(z|x) \nabla_\lambda \left(\log \frac{p_\theta(x,z)}{q_\lambda(z|x)} \right) dz$$

Unfortunately, in this case, we cannot just simply push the gradient through, because this time $q_\lambda(z|x)$ DOES depend on ∇_λ. Is (7) salvageable? Can we use Monte Carlo estimation on this term anyway? Unfortunately not, because in its current form the integral is not an expectation with respect to $q_\lambda(z|x)$ anymore. ($\nabla_\lambda q_\lambda(z|x)$ is not necessarily a distribution)

5.1 Score Matching (Optional)

There are two ways to fix this issue, one using an estimator called the score matcher, and the other the "reparametrization" trick, the latter of which was proposed in the original variational autoencoder paper. Let us begin with the score matcher.

Going back to (7),

$$(7) = \int \nabla_\lambda \left(q_\lambda(z|x) \log p_\theta(x,z) \right) dz - \int \nabla_\lambda \left(q_\lambda(z|x) \log q_\lambda(z|x) \right) dz$$

[1] Leibniz Rule even holds for certain discontinuous functions, if it satisfies something called the Dominated Convergence Theorem, which is beyond the scope of the paper.

Noting the gradient of the first integral doesn't depend on $\log p_\theta(x,z)$, and applying the product rule in the second integral:

$$= \int \Big(\nabla_\lambda q_\lambda(z|x)\Big) \log p_\theta(x,z) dz - \int \Big(\nabla_\lambda q_\lambda(z|x)\Big) \log q_\lambda(z|x) + \frac{\nabla_\lambda q_\lambda(z|x)}{q_\lambda(z|x)} q_\lambda(z|x) dz$$

$$= \int \Big(\nabla_\lambda q_\lambda(z|x)\Big) \log p_\theta(x,z) dz - \int \nabla_\lambda q_\lambda(z|x) \Big(\log q_\lambda(z|x) + 1\Big) dz$$

Since $\int \nabla_\lambda q_\lambda(z|x) dz = \nabla_\lambda \int q_\lambda(z|x) dz = \nabla_\lambda(1) = 0$, the very last term disappears.

$$= \int \Big(\nabla_\lambda q_\lambda(z|x)\Big) \log p_\theta(x,z) dz - \int \nabla_\lambda q_\lambda(z|x) \Big(\log q_\lambda(z|x)\Big) dz$$

$$= \int \nabla_\lambda q_\lambda(z|x) \Big(\log p_\theta(x,z) - \log q_\lambda(z|x)\Big) dz \quad (8)$$

Noting from calculus the gradient of the log,

$$\nabla_x \log(f(x)) = \frac{\nabla_x f(x)}{f(x)} \Leftrightarrow f(x) \nabla_x \log(f(x)) = \nabla_x f(x)$$

We can therefore substitute this in (8),

$$\int q_\lambda(z|x) \nabla_\lambda \log(q_\lambda(z|x)) \Big(\log p_\theta(x,z) - \log q_\lambda(z|x)\Big) dz$$

$$= \mathbb{E}_{z|x\sim q} \Big[\nabla_\lambda \log(q_\lambda(z|x)) \Big(\log p_\theta(x,z) - \log q_\lambda(z|x)\Big)\Big]$$

Yay! We have an expectation here and can use Monte Carlo techniques. However, empirically the variance of this estimator is very high. That's why in the orignal variational autoencoder paper, the authors suggested an ingenious trick to use in place of this. The catch? We will have to impose stronger (but pretty reasonable) conditions on the functions q and p.

5.2 Reparametrization trick

$$\nabla_\lambda \mathbb{E}_{z|x\sim q} \left[\log \frac{p_\theta(x,z)}{q_\lambda(z|x)}\right]$$

Recall why this was problematic. The gradient w.r.t. λ is unable to be passed through the expectation because said expectation depends on λ through $q_\lambda(z|x)$. The idea behind the reparametrization trick is to convert $q_\lambda(z|x)$ into a deterministic portion g, which depends on λ, and a stochastic portion ϵ, which doesn't. This way ∇_λ will be able to pass through the expectation like ∇_θ by simply ignoring the stochastic element.

Instead of $z \sim q_\lambda(z|x)$, let us define

$$z = g_\lambda(x, \epsilon) \text{ where } \epsilon \sim \zeta(\epsilon)$$

Note that $g_\lambda(x, \epsilon)$ is a deterministic function, which takes in a given ϵ only after it has been sampled. What form should $g_\lambda(x, \epsilon)$ and $\zeta(\epsilon)$ take? We want to be able to simulate $q_\lambda(z|x)$'s behavior. A strategy is to have $g_\lambda(x, \epsilon)$ determine the parameters of the distribution, which when mixed with an appropriate stochastic element ϵ, fully describes the behavior of the distribution. The most natural choice is the location-scale family of distributions, which can be fully represented by their "mean" (location), "variance" (scale) and a unit member of the distribution. The most famous member of this family is unsurprisingly the Gaussian. All Gaussian random variables $Y \sim \mathcal{N}(\mu, \Sigma)$ have the property $Y = \mu + \sigma * X$ where $X \sim \mathcal{N}(0, I)$.

In our formalism, if we set $q_\lambda(z|x) = \mathcal{N}(\mu_\lambda(x), \Sigma_\lambda(x))$, then we let

$$g_\lambda(x, \epsilon) = \mu_\lambda(x) + \Sigma_\lambda(x) * \epsilon \text{ where } \epsilon \sim \mathcal{N}(0, I)$$

In this scenario, the parameter λ affects the conditional $q_\lambda(z|x)$ through its effect on the two Gaussian parameters μ and Σ. Also, as a corollary we can see one (parametric) way in which neural networks can learn conditional densities (by having each parameter of a conditional distribution be an output, and having the distribution's negative log likelihood as the loss function).

We are now done. Going back to that pesky expectation (7) and subbing in our new function in place of z,

$$\nabla_\lambda \mathbb{E}_{z|x\sim q}\left[\log \frac{p_\theta(x,z)}{q_\lambda(z|x)}\right] = \nabla_\lambda \mathbb{E}_{\epsilon\sim\zeta(\epsilon)}\left[\log \frac{p_\theta(x, g_\lambda(x,\epsilon))}{q_\lambda(g_\lambda(x,\epsilon)|x)}\right]$$

$$= \nabla_\lambda \int \zeta(\epsilon) \log \frac{p_\theta(x, g_\lambda(x,\epsilon))}{q_\lambda(g_\lambda(x,\epsilon)|x)} d\epsilon = \int \nabla_\lambda \zeta(\epsilon) \log \frac{p_\theta(x, g_\lambda(x,\epsilon))}{q_\lambda(g_\lambda(x,\epsilon)|x)} d\epsilon$$

Now that $\zeta(\epsilon)$ does not depend on λ, we can push ∇_λ through,

$$= \int \zeta(\epsilon) \nabla_\lambda \log \frac{p_\theta(x, g_\lambda(x,\epsilon))}{q_\lambda(g_\lambda(x,\epsilon)|x)} d\epsilon$$

We can now, just like the score matching estimator, estimate this quantity using Monte Carlo techniques.

However, another path has opened up for us. Previously we couldn't use backpropagation to solve for the optimal value of λ, as backpropagation can handle stochastic inputs, but not stochastic units. However, if we enforce continuity on $q_\lambda(z|x)$, we can use backpropagation for computing the gradients, as we have converted our stochastic unit into a stochastic input with the reparametrization trick. Therefore, unlike the score matcher, we have the option of using Monte Carlo or backpropagation to compute the gradient. However, discrete distributions like the binomial cannot then be learned by using backpropagation!

Let us now slightly alter our notation so we can make it clear what the reparametrization does. The output of our encoder is now $z = g(\mu_\lambda(x), \Sigma_\lambda(x), \epsilon)$ where $\epsilon \sim \zeta(\epsilon)$, and μ_λ and Σ_λ are functions determined by our artificial neural network that take in the input x. This is just another way of expressing $z \sim q_\lambda(z|x)$. Then the rest of the architecture remains as before (Given the encoder z, $y \sim p_\theta(x|z)$).

6 Parametric Families

While we have selected a latent modeling approach, up until now we have remained silent on the functional forms of our distributions p and q (e.g. Gaussian, exponential, gamma, beta, or even going for a non-parametric approach). In standard implementations of the variational autoencoder, we have:

- The prior $p_\theta(z)$ is set to a $\mathcal{N}(z|0, I)$ (it does not depend on θ).
- The likelihood $p_\theta(x|z)$ is set to a $\mathcal{N}(x|\mu_\theta(z), \sigma_\theta(z))$
- The posterior $q_\lambda(z|x)$ is set to a $\mathcal{N}(x|\mu_\lambda(x), \sigma_\lambda(x))$

One is tempted to think the authors were lazy, and just went with Gaussians because everything is Gaussian. But there were actually important theoretical considerations in their selection. Beyond general considerations like ease of computability and faithfulness to the natural process that is trying to be modeled, there were some specific design choices made with such a selection, and armed with our background we can try to understand them.

Let us start with the prior. The prior is set to a simple isotropic Gaussian. Recall from equation (6) that our loss function punishes deviations from this prior, so in a sense this is similar to a "Bayesian" prior, which is what we think a priori the latent variable should be distributed by. Discussion on the prior can often get very heated, so I'll avoid going into depth. But in this case it seems "reasonable" that our process is distributed by a Gaussian.

There is a second consideration for the prior, which is dependent on the form of the loss function we adopt for the ELBO. While the estimators derived in this paper are general, if we use equation (6), and pick our prior $p(z)$ and candidate posterior $q_\lambda(z|x)$ carefully, we may be able to do things quicker or more precisely. If we force both to be Gaussian, it turns out we can compute $D_{KL}(q_\lambda(z|x)||p_\theta(z))$ analytically, hence we will only need to estimate $\mathbb{E}_{z|x\sim q}\left[\log p_\theta(x|z)\right]$ with gradient techniques.

The main consideration for the posterior $q_\lambda(z|x)$, in addition to what we just talked about in the previous paragraph, depends on whether we go for score matching or for the reparametrization trick. If we go for score matching, the key is that $q_\lambda(z|x)$ is amenable to variance reduction techniques like importance sampling. If we go for the reparametrization trick, the importance is that the distribution can be decomposed into its parameters and its stochastic elements. Beyond members of the location-scale family, in theory we can select other distributions like the chi-square with k degrees of freedom (which can be decomposed into the sum of the square of k Gaussian variables) or any distribution that can be expressed using the inverse CDF method (which decomposes into a uniform distribution on [0,1]).

So clearly we have some restrictions on what combinations of distributions can be selected. But in theory many combinations of distributions satisfy these above conditions, and one of the most promising avenues for further work will be deviating from this standard implementation (pun intended).

7 Applications

Let us take stock of what we did. We introduced latent variables for a richer represent of our dataset. We use neural networks to model complicated relationships between the data and the parameters of the probabilistic model we are training. We use an approximation of the log likelihood as to avoid computation with a difficult posterior, and we use a clever reparametrization to allow us to use standard gradient based methods like backpropagation somewhat efficiently.

In theory, all kinds of complicated natural processes or structures can be successfully modeled by the VAEs. Therefore, VAEs seem suited to the task of learning media, such as images or music. On the flip side, it is overkill to use VAEs on simpler datasets, as the model may choose to not even utilize the latent variable z (for an example of this see [1]).

On top of this, due to its generative properties, a VAE can successfully generate new media that is similar in form to its input data. Therefore we can generate images or music. It is quite simple to do this, as once we have learned the model, all we need to do is sample from, in the standard implementation, the isotropic Gaussian and pass it into the decoder $p_\theta(x|z)$).

More than just generation, however, we are also able to recreate mixes of objects that may not be represented in our dataset, and hopefully these new features will be adequately represented due to the complexity of the model. For example, if we have a sample of celebrity faces without glasses, and a sample of individuals wearing glasses, and by averaging out their vector representations of their parameters, we can create a new class of celebrities with glasses!

Using the encoder $q_\lambda(z|x)$ and setting the dimensions of the feature space to be less than that of the input space, we can also learn useful compressions of data that occupy a non-linear manifold.

References

[1] R. Shu. (2017, jan) Autoencoding a single bit. [Online]. Available: http://ruishu.io/2017/01/14/one-bit/

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[3] Y. Bengio, "Learning deep architectures for ai," *Foundations and Trends® in Machine Learning: Vol. 2*, 2009.

[4] D. P. Kingma and M. 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. [Online]. Available: http://arxiv.org/abs/1312.6114

[5] R. Shu. (2018, mar) Density estimation: Variational autoencoders. [Online]. Available: http://ruishu.io/2018/03/14/vae/

[6] C. Doersch, "Tutorial on variational autoencoders," 2016, cite arxiv:1606.05908. [Online]. Available: http://arxiv.org/abs/1606.05908

[7] S. E. Aditya Grover. (2018, Mar) Variational autoencoders, stanford university cs236. [Online]. Available: https://deepgenerativemodels.github.io/notes/vae/

[8] A. Mnih and K. Gregor, "Neural variational inference and learning in belief networks," in *ICML*, 2014.

[9] L. Weng. (2018, Aug) From autoencoder to beta-vae. [Online]. Available: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.htmlautoencoder

[10] S. E. Volodymyr Kuleshov. Stanford university cs228 notes. [Online]. Available: https://ermongroup.github.io/cs228-notes/

[11] J. Jordan. (2019, Mar) Introduction to autoencoders/variational autoencoders. [Online]. Available: https://www.jeremyjordan.me/autoencoders/

[12] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[13] K. Frans. (2016, aug) Variational autoencoders explained. [Online]. Available: http://kvfrans.com/variational-autoencoders-explained/