Talk: Variational Autoencoders 101

September 10th, 2024 Rohan Subramanian, Yiming Song, Sahil Rane, Saheli Patel

1 Introduction

SR: Greetings and introductions.

Today, we'd like to talk about variational autoencoders, a type of generative model that combines mathematical framework of variational inference with the power of deep learning, with important applications in computer vision and computational biology. I'm personally interested in VAEs because they were my first introduction to probabilistic and generative models, and I think they showcase some amazing fundamental mathematical ideas applicable to many more kinds of models.

RS: Our goals are to **(1)** dive into the mathematical foundations of variational inference and VAEs and **(2)** discuss the variations of VAEs that lend themselves to semi-supervised learning.

If you know VAEs inside out, we hope this will be a good recap for you, and if you're like us and just getting into these kinds of models we hope that we can explain the full picture of VAEs from the ground up.

2 Autoencoders

SP: We'd like to start off by recapping autoencoders.

1. *Manifold hypothesis*: Consider randomly generating an n by n pixel image in RGB color space. That image is hardly going to look like anything we see in the real world. In real life, the images we see likely lie on some lower dimensional latent manifold, and we don't probably really need all 5 dimensions to reconstruct an image.

   This is the idea informing compression, PCA, etc.

2. *Autoencoders*: Autoencoders are essentially a non-linear generalization of PCA. But instead of using linear combinations of variables, autoencoders rely on neural networks to form non-linear encodings of the data. They learn an encoding function which transforms the input, and then an decoding function which recreates the input, as best as it can.

3. *Diagram*: The basic architecture of an autoencoder is as follows.

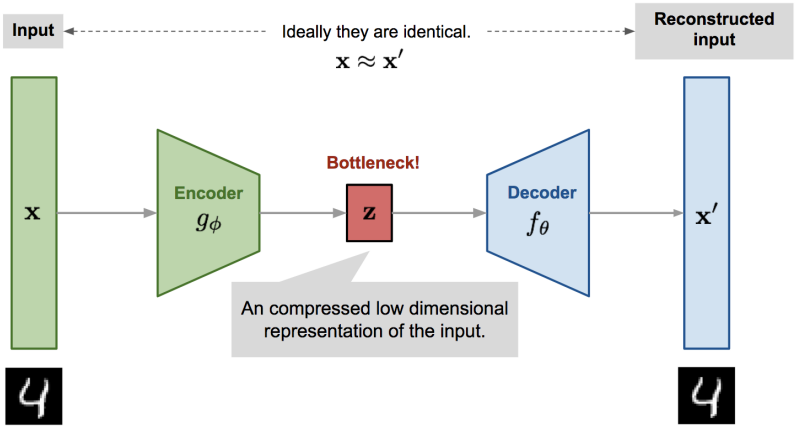

Figure 1: Architecture of a basic autoencoder

- We have our input $\mathbf{x} \in \mathcal{X}$, which we pass through some learned encoder function $h_\phi : \mathcal{X} \to \mathcal{Z}$ to get a lower-dimensional representation $\mathbf{z}$. Usually $\mathcal{X}$ and $\mathcal{Z}$ are Euclidean spaces, with the latter being of a lower dimension to ensure that we don't just learn an identity function.
- We also have the decoder function $f_\theta(\mathbf{z})$ which takes our lower-dimensional representation and reconstructs it into an approximation $\mathbf{x}' \in \mathcal{X}$. We hope to mininize the difference between $\mathbf{x}$ and $\mathbf{x}'$.
- The two functions are respectively parameterized by ϕ and θ, and are usually neural networks.

4. *Autoencoders are limited*: For one input $\mathbf{x}$, eg. a picture of a horse, it will encode a single lower dimensional representation containing essence of this horse belonging somewhere in $\mathcal{Z}$, and the decoder will take this essence and spit back out the original horse picture. But the latent space that all the encoded representations inhabit is limited. It probably won't be continuous, and thus it will not allow for interpolations. Thus if we're trying to generate a large variety of synthetic outputs, or interpolate between two existing inputs, it is difficult.

   Let's say our model was trained on MNIST and it has learned latent mappings for each of the digits, something like this: *Draw diagram*

   There could be rough learned clusters for each digit, but let's say we want to see how 5 morphs into 3, or what something would look like. The model has no idea of what this would look like, since the latent space in between these two entries is empty!

   We are unable to randomly sample from the latent space to generate new horses. Now this is very heartbreaking because what if I wanted to interpolate between two different horses?

3 Variational Autoencoders

SR: Well I can fix that for you! VAEs were devised as a way to conserve the structure of the autoencoder, but better serve as generative models. The changes are as follows:

1. Since we want to sample from a latent distribution, we make our encoding function output parameters μ and σ of a distribution (usually Gaussian) for each input.
2. Then, we sample from the latent conditional distribution $p(\mathbf{z}|\mathbf{x})$ for a value of $\mathbf{z}$ which we feed into the decoder.
3. Then, we take $\mathbf{z}$ and feed it into our decoder, which maps it to another set of parameters ψ of some distribution D.
4. Finally, from $D(\psi)$ we sample our final output $\mathbf{x}'$.

Compared to the deterministic autoencoder, we have created something stochastic. Also, compared to a regualar autoencoder, there exists a constraint on the distribution of the latent space, most often $\mathbf{z}_i \sim \mathcal{N}(0, I)$. The output of the decoder is also the parameters of a distribution (often Gaussian), but can be something else.

We can view VAEs from two perspectives:

1. Stochastic autoencoders: The encoder and decoder output parameters for a distribution from which we can sample. Draw picture:

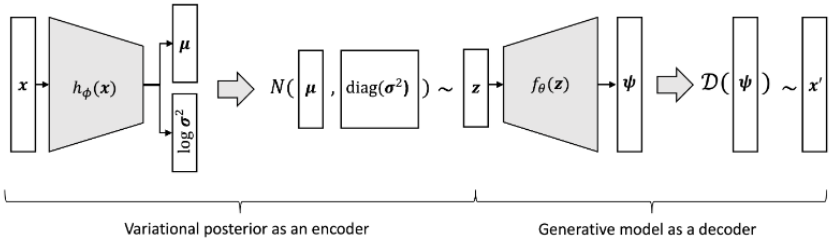

Figure 2: Architecture of a basic VAE

2. Probabilistic graphical model: Draw plate diagram. Explain how encoder corresponds to $p(\mathbf{z}|\mathbf{x})$ and decoder corresponds to $p(\mathbf{x}|\mathbf{z})$.

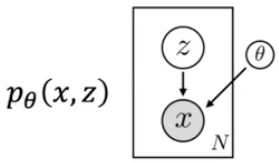

Figure 3: Plate diagram of a basic VAE

But if both $p(\mathbf{z})$ and $p(\mathbf{x}|\mathbf{z})$ are Gaussians, how can you learn a possibly very complex distribution of $\mathbf{x}$? I don't think the essential features of horses are normally distributed.

SR: I'm glad you asked. Even if each conditional distribution $p(\mathbf{x}|\mathbf{z})$ is a Gaussian, we end up being able to reconstruct possibly very complex distributions for $\mathbf{x}$. Consider the fact that f_θ is a neural network that can learn non-linear mappings from each sampled latent vector $\mathbf{z}$ to the parameters ψ. We get an image something like this:

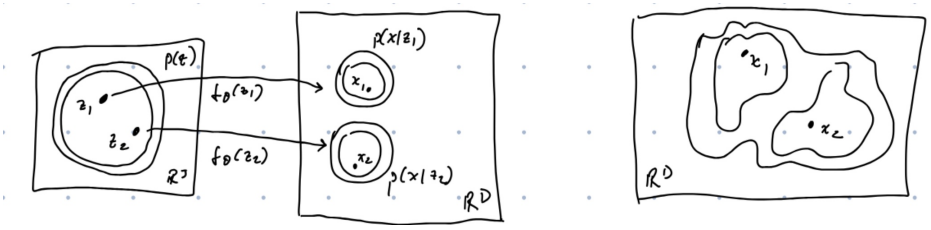

Figure 4: Learning complex marginal distribution

4 Inference in VAEs

RS: Now I get it! Suppose I then want to use a VAE to generate images, for example. There are two inference tasks I need:

1. Given x, calculate the posterior distribution over the latent space.

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{\int p_\theta(\mathbf{x},\mathbf{z})d\mathbf{z}} \tag{1}$$

   Why: Learn how to encode images, get the latent representation of an image and then generate similar images.

   Note on Bayesian terminology: The posterior is $p(z|x)$, the likelihood is $p(x|z)$, the prior is $p(z)$, and the marginalization is $p(x)$.

2. Find parameters θ to maximize the log-likelihood of the data under our model.

$$\hat{\theta} := \arg\max_\theta p_\theta(x) = \arg\max_\theta \int p_\theta(x,z)dz \tag{2}$$

   Easy, right? Just integrate.

SR: But Rohan, don't we have to integrate over all the dimensions of the latent space $\mathcal{Z}$? This seems really complex because $p_\theta(x|z)$ is defined by a neural network and integrating over a NN would be intractable. We can't use conventional EM algorithms which involve knowing $p_\theta(\mathbf{z}|\mathbf{x})$.

5 Bad Idea: Direct Monte Carlo Sampling

RS: I have an idea. Let's use Monte Carlo sampling, where we sample z_k from the prior $p(z)$ in the last step. If we sample enough points, it will basically be perfect.

$$\begin{aligned} p(x) &= \int p(x|z)\, p(z)\, dz \\ &= \mathbb{E}_{z\sim p(z)}[p(x|z)] \\ &\approx \frac{1}{K}\sum_k p(x|z_k) \end{aligned} \tag{3}$$

SP: Hmm, let's see. If we have a 2-dimensional latent space, I could see how sampling can get us somewhat accurate results, but what if we increase the number of dimensions? Doesn't this lead us to a curse of dimensionality scenario? With each added latent dimension the number of points has to grow exponentially! This also seems unfeasible.

Also, the prior $p(z)$ is a lot simpler than the true posterior $p(z|x)$. Sampling from the prior might not be truly representative of the data distribution. Why don't we try to actually approximate the posterior?

6 Variational Inference and ELBO

RS: Great idea! Let's try to approximate $p(z|x)$ with another distribution called $q_\phi(z|x)$. This is called the variational distribution. This distribution will be parametrized by $\phi = (\mu, \sigma)$. This means that q comes from the family of Gaussians, so our task is basically to have q be the Gaussian most similar to the true posterior.

Now with this new tool, we can revisit our inference task and try to find the log likelihood. Begin ELBO derivation...

6.1 ELBO Derivation

$$\begin{aligned}
\log p(x) &= \log \int_z p(x,z)\, dz \\
&= \log \int_z p(x,z) \frac{q(z|x)}{q(z|x)}\, dz \\
&= \log \mathbb{E}_{z\sim q(z|x)} \left[\frac{p(x,z)}{q(z|x)}\right] \\
&\geq \mathbb{E}_{z\sim q(z|x)} \left[\log \frac{p(x,z)}{q(z|x)}\right] \text{ (By Jensen's inequality)} \\
&= \mathbb{E}_{z\sim q(z|x)} \left[\log \frac{p(x|z)p(z)}{q(z|x)}\right] \\
&= \mathbb{E}_{z\sim q(z|x)} \left[\log p(x|z)\right] + \mathbb{E}_{z\sim q(z|x)} \left[\log \frac{p(z)}{q(z|x)}\right] \\
&= \mathbb{E}_{z\sim q(z|x)} \left[\log p(x|z)\right] - \mathbb{E}_{z\sim q(z|x)} \left[\log \frac{q(z|x)}{p(z)}\right]
\end{aligned}$$

SR: Wait Rohan, isn't that second term just the KL divergence?
RS: Indeed!
SR:

Definition 6.1. *The KL divergence is a measure of how similar two probability distributions are. It is simply the expected value of the log-ratio of the entire dataset.*

$$KL(P||Q) = \mathbb{E}_{x\sim p(x)} \left[\log \frac{q(x)}{p(x)}\right] \geq 0 \tag{4}$$

Hence we can conclude our derivation of the ELBO as follows:

$$\log p_\theta(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)} \left[\log p_\theta(x|z)\right] - \mathrm{KL}\left(q_\phi(z|x) \| p(z)\right) := \mathrm{ELBO}(\theta, \phi) \tag{5}$$

So by
We notice several amazing things about this equation.

1. The first term corresponds to the "reconstruction loss". This is because if you substitute in the Gaussian PDF, you will obtain the MSE between x and the reconstructed x.

$$\log p_\theta(x_i \mid z_i) = -\frac{d}{2} \log(2\pi\sigma^2_{\text{decoder}}) - \frac{1}{2\sigma^2_{\text{decoder}}} \|x_i - f_\theta(z_i)\|_2^2 \tag{6}$$

2. The second term can be viewed as a "regularizer" that constrains the latent space to be close the the prior i.e. normally distributed.

3. So the first recon term is analogous to the autoencoder where the output is an accurate representation, and the second KL term is a regularizer that constrains the latent space to follow some distribution, preventing rote memorization.

4. The KL term can be computed analytically because there's a simple formula for the KL between two Gaussians, and we know both the distributions (variational posterior $q_\phi(z|x)$ and prior $p(z)$. So it is also differentiable.

$$KL(N_1||N_2) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma^2} - \frac{1}{2} \tag{7}$$

So just to revisit our three goals: **(1)** A lower bound on the log likelihood **(2)** Reduce sampling and the variance of gradient **(3)** Make the loss differentiable.

We showed that **(1)** is true for the ELBO. **(2)** and **(3)** are satisfied for the KL term. But what about the second term? If we try to evaluate the expectation, we run into the same intractable integral:

$$\mathbb{E}_{z\sim q_\phi(z|x)} \left[\log p_\theta(x|z)\right] = \int_z \log(p(x|z)) q_\phi(z|x) dz \tag{8}$$

This integral is intractable just like before, so it turns out we can't really escape Monte Carlo sampling. We will have to do Monte Carlo sampling from the variational posterior $q_\phi(x|z)$.

Why exactly is this better than direct Monte Carlo sampling from the prior that we concluded was a bad idea? For one, we are sampling from variational posterior. This gives us a better approximation based on our data and also has less variance. So we settle on a compromise for **(2)**. But we still need to resolve the last requirement. Our ELBO needs to differentiable so we can take the gradient and optimize the parameters. The KL is differentiable, but what about the first term?

6.2 Reparameterization Trick

So when we sample from the posterior, it looks like this:

$$\begin{aligned} z \sim q_\phi(x|z) &= \mathcal{N}(\mu_\phi(x), \sigma_\phi(x)) \\ &= \mu_\phi(x) + \sigma_\phi(x)\epsilon \\ &\text{where } \epsilon \sim \mathcal{N}(0,1) \\ &g(x,\epsilon) = \mu_\phi(x) + \sigma_\phi(x)\epsilon \end{aligned} \tag{9}$$

The gradient of the ELBO with respect to ϕ now depends on ϵ and x, but not directly on the random variable z. This separation makes the gradient computation deterministic relative to ϕ, which reduces variance.

Thus, instead of sampling straight from our variational posterior, which could lead to a lot of variance in our gradient, by reparameterizing, we get to first sample a surrogate random variable ϵ while ensuring that our final $\mathbf{z}$ still follows the distribution q_ϕ.

$$
\begin{aligned}
\log p(\mathbf{x}) &\geq \text{ELBO}(\phi, \theta) \\
&= \sum_{i=1}^{n} E_{\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_\theta(\boldsymbol{x}_i|\boldsymbol{z}_i)\right] - \text{KL}\left[q_\phi(\boldsymbol{z_i}|\boldsymbol{x_i})||p(\boldsymbol{z_i})\right] \\
&= \sum_{i=1}^{n} \int_{\boldsymbol{z}_i} q_\phi(\boldsymbol{z}_i \mid \boldsymbol{x}_i) \log p_\theta(\boldsymbol{x}_i|\boldsymbol{z}_i) d\boldsymbol{z}_i - \text{KL}\left[q_\phi(\boldsymbol{z_i}|\boldsymbol{x_i})||p(\boldsymbol{z_i})\right] \\
&= \sum_{i=1}^{n} E_{\epsilon_i \sim \mathcal{N}} \log p_\theta(\boldsymbol{x}_i|g_\phi(\boldsymbol{\epsilon}_i, \boldsymbol{x}_i)) - \text{KL}\left[q_\phi(\boldsymbol{z_i}|\boldsymbol{x_i})||p(\boldsymbol{z_i})\right] \\
&\approx \sum_{i=1}^{n} \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(\boldsymbol{x}_i|g_\phi(\boldsymbol{\epsilon}_{i,l}, \boldsymbol{x}_i)) - \text{KL}\left[q_\phi(\boldsymbol{z_i}|\boldsymbol{x_i})||p(\boldsymbol{z_i})\right]
\end{aligned}
\tag{10}
$$

Kingma and Welling emphasize in the original paper that g needs to be differentiable. This way, we can now backpropagate through the entire network. So ultimately, we solved our 3 goals as well as possible:

1. We have a lower bound on the log likelihood through the ELBO.
2. Reduced sampling and the variance of gradient as much as possible by making the KL term analytic and using the reparametrization trick while sampling the reconstruction term.
3. We made the loss differentiable w.r.t the parameters θ and ϕ using the reparametrization trick. Optimizing the ELBO w.r.t these two parameters trains our model.

7 Implementation

SP: Just to close our discussion of the theory, I want to mention a few practical considerations. We have defined our loss function such that PyTorch's automatic differentiation correctly computes the gradients. We can use `nn.Module` to define an Encoder and Decoder with `sample` and `forward` methods for each one.

In practice, we actually have to change a few things from the theory. For example, we said that our encoder neural network outputs a mean and standard deviation. Firstly, we have to actually output the log variance, to ensure that it is positive. Also, it turns out that for images especially, it doesn't really learn a standard deviation for a pixel very well. What works better in practice is to define a shared standard deviation for every pixel and train it

as a parameter instead.

Secondly, the two terms in the loss function often have very different scales. Usually the recon loss ends up being a lot larger than the KL term and we see something called KL-vanishing. Hence, we usually introduce a parameter β as a weight for the KL loss and choose it to make the values roughly equal. We can also apply beta-annealing i.e. Start off with a small beta, so the model focuses on accurate reconstruction at the start of training. Gradually increase beta so the model the doesn't forget to constrain the distribution of the latent space.

Most of all, I'm always pretty amazed that this complex math reduces to 5-10 lines of PyTorch code!

8 Application: MNIST

SR: Now let's look at a few applications. Consider the MNIST dataset. We can train our VAE on the dataset, but then in addition to encoding-decoding the digits, we can sample from the learned latent space to generate pseudo-digits.

If we recall the ELBO, which consists of a reconstruction loss term and a regularization loss term, the first term ensures that when I decode something in the latent space, it actually looks like a digit. The second term will make sure that our learned posterior matches closely with the prior, so for example, if I input two 3s into the model, the conditional distributions will be close on the latent space, so all the 3s are going to positioned close to each other in the space.

If, for any reason, I want to generate more 3s, I need to first train the VAE on the entire MNIST dataset. Then, for a given 3 that corresponds to some z_3 in the latent space, I can sample from the $D(\psi)$ that it corresponds to, and end up with new, randomly generated 3s.

If I wanted to interpolate between a 3 and a 5, we would take two points z_3 and z_5, and traverse the latent space between these two points, generated samples on this line. In a normal autoencoder, there's no guarantee that these in between digits would look like meaningful interpolations between a 3 and a 5. But Rohan, what if I wanted to cure cancer instead of generate digits?

9 Semi-supervised learning

SR: In semi-supervised learning we are given a few labels but most of our data is unlabelled. The approach to semi-supervised learning using deep generative models is that we want to learn the distribution of our data in some latent space so that we are able to perform better on our unlabelled data. Essentially, we want to avoid overfitting to our labelled data but we are able to construct a latent space of our data that separates the labels well within the latent space so that we are able to leverage the clustering of observations with similar labels in our latent space to perform better on the unsupervised component of the learning.

9.1 Deep Generative Models for semi-supervised learning

9.1.1 M1 Model: Latent Feature Discriminative

SP: The M1 model is a simple approach that basically uses a VAE to generate latent embeddings, then trains a classifier on the embeddings instead of the raw data/images.

We train the VAE on the entire unlabeled dataset, then use the encoder to generate latent embeddings for all the images. Now, using the latent representation a separate classifier is trained to predict the labels. The embeddings are meant to reduce the dimensionality of the problem and make it easier to train the classifier by improving clustering of observations in the latent space. Essentially, we construct a deep generative model of the data that is able to provide a more robust set of latent features. We sample from the posterior distribution over the latent variables and train the classifier (SVM or logistic regression). These low dimensional embeddings should now also be more easily separable since we make use of independent latent Gaussian posteriors whose parameters are formed by a sequence of non-linear transformations of the data.

9.1.2 M2 Model: Generative Semi-Supervised Learning

RS: The M2 model attempts to jointly learn the latent variable z and the partially observed labels y. In a regular VAE, we describe that the data x is generated by a latent variable z. However in the M2 model, we describe that the data x is generated by a latent variable z **and** a class variable y that has frequencies π. This leads to several added dependencies.

Now, the encoder and decoder will both not only use x but also y to encode and decode. This leads to two cases - one where we have labeled data and another where we have unlabeled data. In the unlabeled case, the encoder/decoder training is the same as a regular VAE. In the labeled case, we add the class as an input to the encoder/decoder. There is a different loss function/objective in each case. The discriminative classifier is represented by $q_\phi(y|x)$.

$$p(y) = \text{Cat}(y|\pi) \tag{11}$$

$$p(z) = \mathcal{N}(z|0, \mathbf{I}) \tag{12}$$

$$p_\theta(y, z) = f(x; y, z, \theta) \tag{13}$$

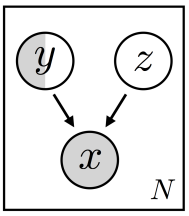

Figure 5: Graphical model representation of M2 model.

The M2 model has several problems in practice. Mainly, learning the latent space and the classification can become decoupled. Prof. Hope can give more insight into this.

9.1.3 Prediction-Constrained VAE

SR: The M2 model has several problems in practice. The prediction-constrained VAE attempts to mitigate the decoupling of learning the latent space and learning the classifier. Here, the labels y are classified from the latent representations z, which is basically the principle of M1 model. Unlike the M2, the class label y is not an input to the encoder/decoder.

The basic addition of this model is to add a third term to the loss, which is the classification loss (BCE). In every step of training, we have a batch of labeled and unlabeled data. The labeled data is passed through a classifier (e.g. logistic regression) and a loss term is identified. The unlabeled data is used to train the VAE similar to the regular method.

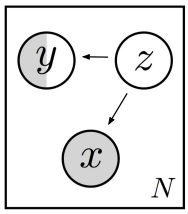

Figure 6: Graphical model representation of PC-VAE model.

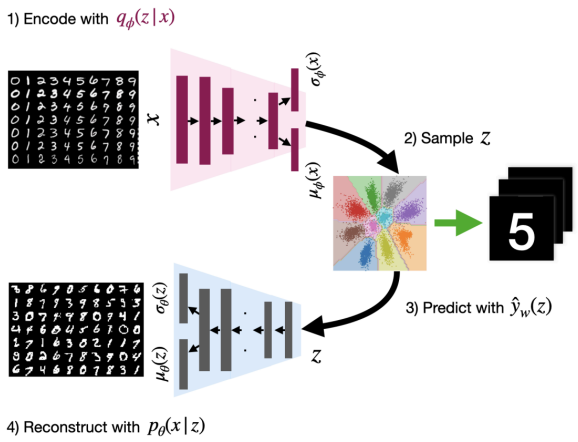

Figure 7: Computational flow diagram of a prediction-constrained variational autoencoder.

9.1.4 Results from last semester

1. **Implementation:** We implemented linear and convolutional VAE, β-VAE, M1 model and PC-VAE models that run on MNIST and CIFAR-10 dataset.

2. **Modular Codebase:** We have organized the PyTorch codebase to be modular (swap out architectures, loss functions, modules, datasets) so it's easier to build on this.

3. **Visualization Toolkit:** We have written tools to visualize the latent space using PCA, interpolate between points, plot stacked histograms and other useful figures.

4. **Experiments:** Compare the classification accuracy on MNIST and CIFAR-10 with an 80-20 train/test split, versus 100 labels.

   (a) Logistic regression on raw images

   (b) Neural network with the same architecture as encoder on raw images

   (c) Logistic regression on VAE-generated embeddings

   (d) Logistic regression on M1 VAE-generated embeddings generated embeddings

   (e) Logistic regression on PC-VAE-generated embeddings

10 Appendix

More on the reparametrization trick:

Assuming we have n samples:

$$\begin{aligned}
\log p(\mathbf{x}) &\geq \text{ELBO}(\phi, \theta) \\
&= \sum_{i=1}^{n} E_{\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_\theta(\boldsymbol{x}_i, \boldsymbol{z}_i) - \log q_\phi(\boldsymbol{z}_i \mid \boldsymbol{x}_i)\right] \\
&= \sum_{i=1}^{n} \int_{\boldsymbol{z}_i} q_\phi(\boldsymbol{z}_i \mid \boldsymbol{x}_i) \left[\log p_\theta(\boldsymbol{x}_i, \boldsymbol{z}_i) - \log q_\phi(\boldsymbol{z}_i \mid \boldsymbol{x}_i)\right] \, d\boldsymbol{z}_i \\
&= \sum_{i=1}^{n} E_{\epsilon_i \sim \mathcal{N}} \left[\log p_\theta(\boldsymbol{x}_i, g_\phi(\boldsymbol{\epsilon}_i, \boldsymbol{x}_i)) - \log q_\phi(g_\phi(\boldsymbol{\epsilon}_i, \boldsymbol{x}_i) \mid \boldsymbol{x}_i)\right]
\end{aligned} \tag{14}$$

Upon first glance it's not clear why this fixes our problems. However:

1. **Deterministic Gradient Path**: The gradient of the ELBO with respect to ϕ now depends on ϵ and x, but not directly on the random variable z. This separation makes the gradient computation deterministic relative to ϕ, which reduces variance.

2. **Efficient Gradient Estimation**: By fixing ϵ during backpropagation for each x, we compute gradients that are consistent and have lower variance compared to direct sampling from $q_\phi(z \mid x)$.

3. **Integration Simplification**: The reparameterization transforms the integral over z into an expectation over ϵ, simplifying the computation of the ELBO and making it more amenable to gradient-based optimization.

Thus, we can then approximate the ELBO using Monte Carlo sampling, only this time it poses less of a problem than before since we don't face issues with dimensionality and etc.

Finally, we arrive at a differentiable estimate for the ELBO which allows us to perform stochastic gradient ascent:

$$\begin{aligned}
\tilde{\text{ELBO}}(\phi, \theta) &:= \frac{1}{n} \sum_{i=1}^{n} \frac{1}{L} \sum_{l=1}^{L} \left[\log p_\theta(\boldsymbol{x}_i, g_\phi(\boldsymbol{\epsilon}'_{i,l}, \boldsymbol{x}_i)) - \log q_\phi(g_\phi(\boldsymbol{\epsilon}'_{i,l}, \boldsymbol{x}_i) \mid \boldsymbol{x}_i)\right] \nabla_{\phi,\theta} \\
\tilde{\text{ELBO}}(\phi, \theta) &= \frac{1}{n} \sum_{i=1}^{n} \frac{1}{L} \sum_{l=1}^{L} \nabla_{\phi,\theta} \left[\log p_\theta(\boldsymbol{x}_i, g_\phi(\epsilon'_{i,l}, \boldsymbol{x}_i)) - \log q_\phi(g_\phi(\epsilon'_{i,l}, \boldsymbol{x}_i) \mid \boldsymbol{x}_i)\right]
\end{aligned} \tag{15}$$

11 References

1. https://mbernste.github.io/posts/vae/
2. https://jmtomczak.github.io/blog/4/4_VAE.html
3. https://arxiv.org/abs/1312.6114
4. https://openreview.net/forum?id=Sy2fzU9gl
5. https://www.nature.com/articles/s41592-018-0229-2
6. https://gregorygundersen.com/blog/2018/04/29/reparameterization/