CS182 Project

Michael Huang, Chancharik Mitra, Joshua You, Jason Zhang

March 2023

1 AEs

1.1 Autoencoder Warmup

Let $X \in \mathbb{R}^{nxd}$, $f : \mathbb{R}^{nxd} \to \mathbb{R}^{nxk}$, $g : \mathbb{R}^{nxk} \to \mathbb{R}^{nxd}$. Recall that the optimization problem we solve for autoencoders were:

$$\arg\min_{f,g} ||X - g(f(X))||_F^2 \tag{1}$$

What do the functions f and g represent here? How can we learn f and g?

2 VAEs

VAEs, or (V)ariational (A)uto(e)ncoders are a special type of autoencoder that generates new outputs rather than just output lower dimensionality data we've already seen before. More generally, VAEs generate *new* information given patterns we've seen about the data. To do this, we'll need some probability theory under our belt.

2.1 On the Loss Function of VAEs

1. Recall in autoencoders, we have an encoder to decoder structure with a bottleneck in between. We can denote the learning parameters for the encoder as ϕ and the decoder as θ, and z as our latent representation. The variational autoencoder has the same principles, but we now fit a probability distribution over the encoder decoder structure to generate new samples.

   More formally, we create a *probabilistic encoder* and fit a probability distribution $q_\phi(z|x)$ to generate latent representations z, then fit a *probabilistic decoder* and fit a probability distribution $p_\theta(x|z)$. The true goal of using these distributions is to sample $z \sim q_\phi(z|x)$ and use z to generate new

$x' \sim p_\theta(x|z)$. (Intuition Goes Here)

Let's try to further our understanding of why we do this intuitively. Suppose we have a hidden latent random variable Z that affects a seen random variable X. **Given observations of X, (i.e we have $p(x|z)$), explain why it's difficult to find $p(z|x)$.**
hint: Recall that the definition of conditional probability is that:

$$p(z|x) = \frac{p(z,x)}{p(x)} = \frac{p(x|z)p(z)}{p(x)}$$

2. Instead of trying to find $p(z|x)$ to generate new samples x', we can instead try to find a distribution $q(z|x)$ that is as close to $p(z|x)$ as possible that has a *tractable* solution. One way can measure the distance between probability distributions is to use something know as the *Kullback-Leibler Divergence*, which is defined to be $D_{KL}(p||q) = \mathbb{E}_p[\log(\frac{p(X)}{q(X)})]$. From an information theory perspective, it measures the relative entropy from using q as a model when the actual distribution is p. **Show that by using the KL divergence, we can rewrite the minimization problem** $D_{KL}(q_\phi(z|x)||p_\theta(z|x))$ as:

$$\arg\max_{\theta,\phi} \mathbb{E}_{q_\phi(z|x)}[\log(p(x|z))] - D_{KL}(q_\phi(z|x)||p_\theta(z))$$

rewriting this expression gives us

$$\arg\min_{\theta,\phi} D_{KL}(q_\phi(z|x)||p_\theta(z)) - \mathbb{E}_{q_\phi(z|x)}[\log(p_\theta(x|z))]$$

Intuitively speaking, what does $-\mathbb{E}_{q_\phi(z|x)}[\log(p_\theta(x|z))]$ represent?
hint 1: Start with the definition of KL Divergence, and remove terms that are irrelavant to the minimization problem.
hint 2: Recall that we draw a sample z, and use that to reconstruct x' What does it mean then to maximize the log probability of our training set when reconstructing x'?

2.2 Modeling VAEs using Gausssians and The Reparamerization Trick

We definitely don't know the distribution of our data at first glance, but one surefire method that works well in machine learning is **to model everything as a Gaussian**. In other words, we model $p(z) \sim N(0, I)$ and $p(x|z) \sim N(f(z), cI)$, where f isn't known to us *yet*. We say the mean is $f(z)$ because we're not sure how the distribution will shift given the latent variable. Recall that $q(z|x)$ is an approximation of $p(z|x)$, which was intractable to find. Thus, *the next best thing is to model it as a flexible normal*, i.e $q(z|x) \sim N(g(x), h(x))$, where g and h are not known yet.

1. Recall that the loss function for VAEs were:

$$D_{KL}(q(z|x)||p(z)) - \mathbb{E}_{q(z|x)}[\log(p(x|z))] \tag{2}$$

   Show that we can minimize (1) as:

$$\arg\min_{f,g,h} \mathbb{E}_{q(z|x)}[||x - f(z)||_2^2] + \lambda D_{KL}(q(z|x)||p(z))] \tag{3}$$

   Where λ is a term you derive. Does this look similar to the autoencoder loss function? What would f, g, h represent in the vanilla autoencoder structure?

2. f, g, and h are all learnable functions that can all be trained using the neural network architectures that we've learned in class. However, with the way VAEs work, our backpropagation pipeline is seperated because we are sampling z from $q(z|x)$, so there are no derivatives that can flow back to the encoder. Luckily, researchers came up with the *Reaper Reparameterization Trick*. Instead of sampling directly from $z \sim N(f(x), g(x))$, we instead say $z = \_\_\_ + \_\_\_ * Z$, where $Z \sim N(0, I)$. **Fill in the blanks, and explain how this allows derivatives to flow back to the encoder through backpropagation**.

3. Now you'll try it for yourself! We will create a Variational Autoencoder using Pytorch to generate digits from the MNIST dataset, then use the VAE to train a convolutional neural network to classify digits to see how well it can retain information.

2.3 An alternative formulation

3 VQ-VAEs

Recall that there are three steps when it comes to Variational Auto Encoders:

1. Make our encoder *random* by giving it an input x, sample from seeing x, and try to reconstruct x

2. Add a regularizer during training that tries to keep our latent distribution close to where we want to sample from (i.e keep our latent distribution close to $N(0, I)$)

3. Train using the Reaper Reparameterization Trick.

However, VAEs suffer from a phenomenon known as *posterior collapse*, meaning that the decoder starts ignoring the latent distribution, forcing the decoder the create generic outputs. In image generation, this is possibly due to the fact that images are naturally drawn from a discrete distribution rather than a continuous distribution, so the our sampling codes (i.e a standard normal) may not be able to learn the signals from our training data x resulting in posterior collapse. To mitigate this, we use the VQ-VAE (Vector Quantitized Variational AutoEncoder). Instead of using contionious latent codes, the VQ-VAE using discrete latent codes to generate images. In this problem, we build the VQ-VAE together.

3.1 On the VQ-VAE Architecture

Everything about the VQ-VAE architecture and the VAE architecture is the same except for the latent distribution. Instead of sampling from a continious distribution to generate randomness, we now sample from a discrete distribution. We can do this by introducing a set of vector embeddings (the number of vectors is a number we choose) that will represent our discrete distribution. Figure 1 represents this.
Suppose we can decompose an image x into n different latent vectors (i.e we use a convolutional encoder). **Give one way we can sample the encoding space by using the n latent vectors from our encoding**.

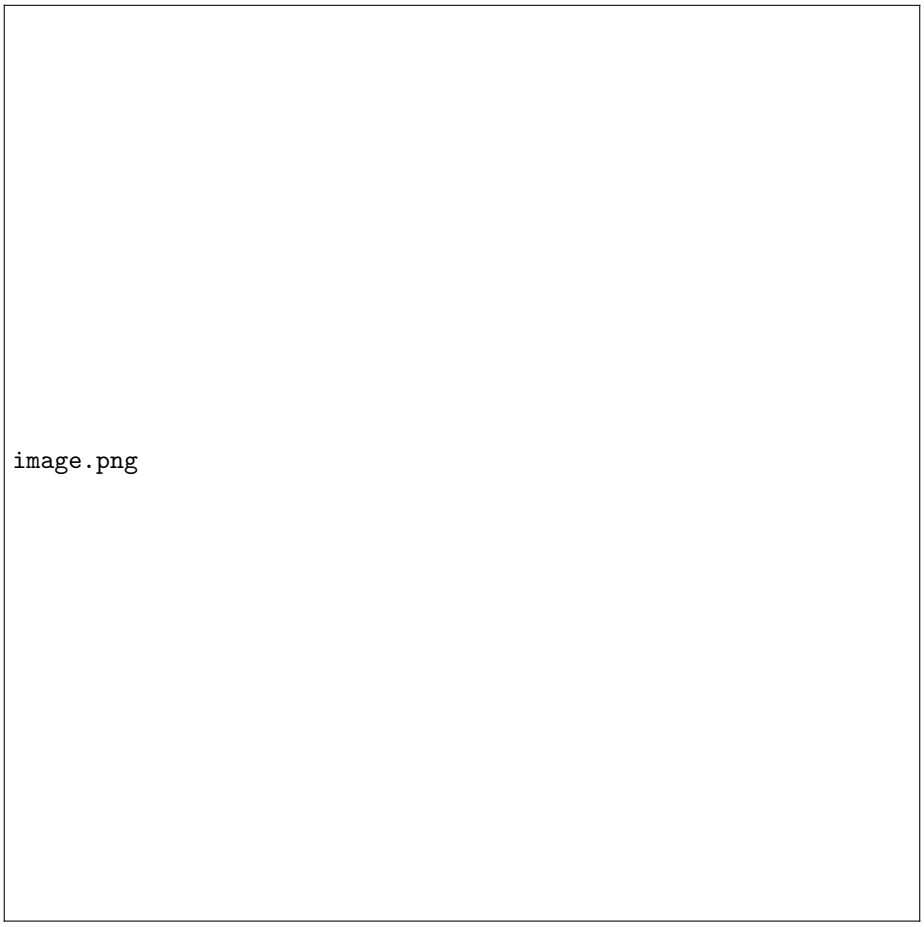

Figure 1: The VQ-VAE Architecture

3.2 Training VQ-VAEs

1. Recall that the loss function for VAEs was defined as:

$$\arg\min_{\theta,\phi} D_{KL}(q_\phi(z|x)||p_\theta(z)) - \mathbb{E}[log(q_\phi(x|z))]$$

Where $\mathbb{E}[log(q_\phi(x|z))]$ is our reconstruction loss, and $D_{KL}(q_\phi(z|x)||p_\theta(z))$ is our regularizer to keep it our latent distribution close to what we want to sample from. However, there is a problem with simply reflecting this loss function with the VQ-VAE.

As with before, since we are sampling from the discrete embedding space, none of our gradients will backpropogate to the decoder. We can't use the reparameterization trick, because we don't actually know the discrete

distribution, we are just fitting it. One method researchers have used to combat this is to simply copy the gradients from the decoder to the encoder. **Let $\tilde{x}$ be our latent output from the encoder, and let $\hat{x}$ be the sampled input to the decoder. Furthermore, let the stop gradient function $sg(x)$ represent a function that blocks gradients to backpropagate through. Define $\hat{x}$ in a way that allows the decoder gradients to be copied over to the encoder using $\hat{x}, \tilde{x}$, and the stopgradient function $sg(x)$.**

2. The way we are sampling from the discrete distribution does not allow gradients to update the distribution itself. Ideally, the encoding of x represents its latent sampling, so we would like our distribution to represent the latent samples we get from the encoder. **Using the stop gradient function sg, our sampled encodings from our distribution e, and our encoded data $z_e(x)$, provide a term that will allow our sampled encodings to update to be closer to our encoder outputs.**

3. Another problem is that our encoder output can grow arbitrarily and not fit to the discrete encoding distribution we create. Thus, we should add a committment loss term that keeps the encoder committed to the latent encoding distribution. **Using the stop gradient function sg, our sampled encodings from our distribution e, and our encoded data $z_e(x)$, provide a term that will allow our encoder to update to be closer to our sampled encodings.**

4. After we've trained a VQ-VAE to encode and decode images, we are left with encodings from our discrete latent distribution that is able to create images from the CIFAR-10 distribution. However, we don't have an immediate way to generate an image from CIFAR-10 because we don't know the distributions of the encodings, just the patterns themselves. **Given the latent distribution samples, give one way we can generate latent distribution samples so that we can generate images from the CIFAR-10 dataset.**