All About Autoencoders: Conceptual Guide

Michael Huang, Chancharik Mitra, Joshua You, Jason Zhang

CS 182 - Spring 2023

1 AEs

1.1 Autoencoders Warmup

1. 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 is given by:

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

What do the functions f and g represent here? How can we learn f and g?
f represents the encoder, and g represents the decoder. If we restrict f and g to learning linear mappings to / from $\mathbb{R}^k$, then this optimization problem can be solved analytically using principal component analysis. For arbitrary, nonlinear mappings, we can use the standard deep learning approach of parameterizing f and g as neural networks then using a descent algorithm (SGD or some variant thereof) to optimize them.

2. Suppose we choose the layer types of the encoder and decoder to be of the following types. **For each layer type, suggest 1 or 2 tasks or data types those architectures would be most suitable for. You may alternatively list drawbacks of those architecture types as well.** *Note: There may be overlap between categories. And although not required, try to think about deeper applications beyond just say, image reconstruction.*

   (a) Linear

   (b) Convolutional

   (c) Transformer

   Solutions may vary

(a) Linear: For the most part, linear layers can be used for tabular data and small-sized images. But some drawbacks include loss of spatial context as well as very high number of parameters.

(b) Convolutional: convolutional layers are most naturally suited to image processing or computer vision tasks, in particular image reconstruction. They can also be applied on visual representations of non-visual data (e.g. spectrograms). Graph convolutional layers would also obviously be suited for graph processing tasks. Beyond this, a larger array of visual tasks such as image inpainting/imputation or image denoising are also possible answers.

(c) Transformer: transformer layers with a self-supervised style loss of reconstructing linguistic / textual data have been used to great effect in creating large language models. More generally, any type of sequential data would be handled well by a transformer architecture. Additionally, with the advent of vision transformers, these types of architectures can also subsume tasks usually well-suited for convolutional networks (Dosovitskiy et. al.).

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.

A quick general note on Autoencoder development. You will notice throughout this assignment and in your future reading that much of the main development in AE architectures happens in the ***bottleneck****.*

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:

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

   Finding the posterior (which is the representation we are trying to learn) is generally quite hard. Unless z can only take on a discrete set of well defined values, calculating the normalization factor in the denominator $p(x)$ is probably infeasible. In particular, it requires evaluating $p(x) = \sum_z p(z)p(x|z)$, which in the continuous setting would be an integral over infinitely many possible values that the prior z can take on.

   Aside: there are actually ways to approximate this value, namely one can attempt to use the Monte-Carlo estimator with K samples $M = \frac{1}{K}\sum_i p(x|z_i)$ where $z_i \sim p(z)$ are independently and identically distributed (i.e. sampled from the prior distribution). However, due to the fact that

$p(x|z_i)$ is likely to be extremely small for most prior samples, the Monte Carlo efficiency is extremely low (intuitively, if you pick a prior completely at random, it probably will not be a good representation for any particular data point, meaning that this sample would be wasted; ideally, you would only pick values z_i which contribute a lot to $p(x)$, i.e. latent representations that represent your data well). This problem only gets worse as you increase the latent dimension and dimension of your data), and too many samples will required for this to be practical.

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'?
Intuitively speaking, the term $-\mathbb{E}_{q_\phi(z|x)}[\log(p_\theta(x|z))]$ represents *reconstruction error*. In particular, our network will sample values from the result of the encoder $q_\phi(z|x)$, then sample values from the result of the decoder $x' \sim p_\theta(x|z)$. Thus, evaluating an expectation over potential samples from the posterior over z of $p_\theta(x|z)$ tells us how likely we are to reconstruct the original input x given this sampling procedure.

$$
\begin{aligned}
D_{KL}(q_\phi(z|x)||p_\theta(z|x)) &= \mathbb{E}_{q_\phi(z|x)}[\log(\frac{q_\phi(z|x)}{p_\theta(z|x)})] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log(q_\phi(z|x)) - \log(p_\theta(z|x))] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log(q_\phi(z|x)) - \log(\frac{p_\theta(x,z)}{p_\theta(x)})] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log(q_\phi(z|x)) - \log(\frac{p_\theta(x|z)p(z)}{p_\theta(x)})] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log(q_\phi(z|x)) - \log(p_\theta(x|z)) - \log(p(z)) + \log(p_\theta(x)))] \\
&= \mathbb{E}_{q_\phi(z|x)}[\log(q_\phi(z|x)) - \log(p(z)))] - \mathbb{E}_{q_\phi(z|x)}[\log(p_\theta(x|z))] + \log(p_\theta(x)) \\
&= D_{KL}(q_\phi(z|x)||p_\theta(z)) - \mathbb{E}_{q_\phi(z|x)}[\log(p_\theta(x|z))]
\end{aligned}
$$

Note: in theory, there is no reason you couldn't create a different proposal distributions q for each data point x_i, since you could optimize $D_{KL}(q(z|x)||p_\theta(z)) - \mathbb{E}_{q(z|x)}[\log(p_\theta(x|z))]$ for each individual data point. However, this is computationally intractable. We rely on the idea that a single q, parameterized as a deep neural network with parameters ϕ, should be able to learn a good mapping from our input domain to the latent space. This allows us to amortize the cost of finding this "proposal distribution" over all of the training examples in our data set, which is very beneficial and also has a regularizing effect. It also means that when we receive a new input at test time, we don't need to optimize for a new choice of q, as we can use the existing q_ϕ. This idea is known as "amortized inference".

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 $q(z|x) \sim N(g(x), \text{diag}(h(x)))$, 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?
f represents the decoder network. g and h together would be made up by the encoder network.

Substituting the Gaussian density function into the expression, we get

$$D_{KL}(q(z|x)||p(z)) - \mathbb{E}_{q(z|x)}[\log(\frac{1}{(2\pi)^{\frac{d}{2}}\det|\Sigma|}\exp(-\frac{1}{2}(x-f(z))^T\Sigma^{-1}(x-f(z)))]$$

We have $\Sigma = cI$, so the normalizing factor of the gaussian density is entirely independent of x, z and can be ignored for the purposes of minimization. $-\frac{1}{2}(x-f(z))^T\Sigma^{-1}(x-f(z))$ becomes just $-\frac{1}{2c}\|x-f(z)\|_2^2$, and thus the final expression is

$$\arg\min_{f,g,h} \frac{1}{2c}\mathbb{E}_{q(z|x)}[||x-f(z)||_2^2] + D_{KL}(q(z|x)||p(z))]$$

or equivalently,

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

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 separated 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 $z \sim N(g(x), \text{diag}(h(x)))$, we instead say $z = \_\_\_ + \_\_\_ * \epsilon$, where $\epsilon \sim N(0, I)$. **Fill in the blanks, and explain how this allows derivatives to flow back to the encoder through backpropagation**.

   We can use the observation that every Gaussian is an affine transformation of a standard normal gaussian, and $z = g(x) + h(x) \odot \epsilon$.

   In order to perform gradient descent steps on the loss, we need $\nabla_{\phi,\theta}(\mathbb{E}_{q(z|x)}[||x-f(z)||_2^2] + 2cD_{KL}(q(z|x)||p(z))])$. Finding the gradient with respect to the KL loss is not difficult due to everything being gaussian, and finding the gradient of the reconstruction loss with respect to θ is not too bad either. However, the fact that the expectation in the reconstruction loss is being taken over $q_\phi(z|x)$, a distribution dependent on the value of ϕ, means that we cannot do the normal procedure of using $\nabla_\phi\|x_i - f(z_i)\|_2^2$ (in the case of minibatches of size 1) as an unbiased estimator for $\nabla_\phi \mathbb{E}_{q(z|x)}[||x-f(z)||_2^2]$,

since the z_i have already been sampled from $q(z|x)$, which is not a differentiable operation.

This reparameterization allows us to compute $\nabla_\phi \mathbb{E}_{q(z|x)}[||x - f(z)||_2^2]$ by reparameterizing z in terms of two deterministic values (the output of the encoder network g and h) and a random value which is completely independent of the parameters ϕ. In particular, we have that $\mathbb{E}_{q(z|x)}[||x - f(z)||_2^2] = \mathbb{E}_\epsilon[||x - f(g(x) + h(x) \odot \epsilon)||_2^2]$. Now, $\nabla_\phi \mathbb{E}_\epsilon[||x - f(g(x) + h(x) \odot \epsilon)||_2^2]$ has an unbiased estimator $\nabla_\phi ||x - f(g(x) + h(x) \odot \epsilon_i)||_2^2$ where $\epsilon_i \sim \mathcal{N}(0, I)$ which can easily be computed using standard backprop.

This trick is also called the "pathwise derivative," and a good illustration / cartoon of it can be found on the Wikipedia page for VAEs.

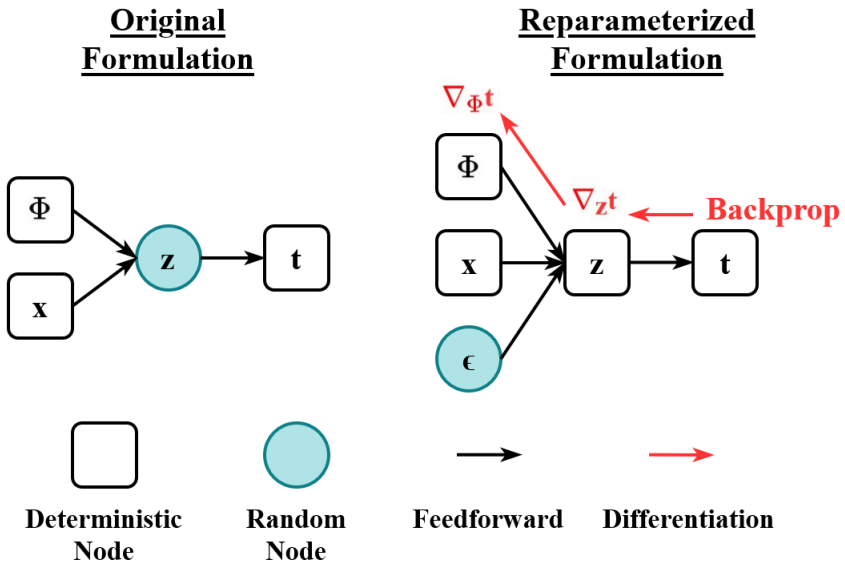

Figure 1: the "reparameterization trick", credit: By EugenioTL - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=107231103

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

Gaussian priors over your latent variables are by far the most widely used option, mostly due to the power of this reparameterization trick. However, there are perhaps reasons you might want to select something non-Gaussian for your prior. For example, if you know that your input data is discrete, perhaps it would be reasonable to have 1 coordinate of your latent space be discrete, and the others be continuous. An example might be that you hope one variable of your latent space represents the digit of an MNIST training point and the continuous variables capture the angle, stroke width, etc. This question will explore an alternate formulation of the VAE which allows you to learn without the aid of the reparameterization trick.

In particular, recall that the problem that the reparameterization trick solved was allowing us to find a good estimator for $\nabla_\phi \mathbb{E}_{q(z|x)}[\log(p_\theta(x|z))]$ (the more general form of reconstruction loss when we assume nothing about the probabilistic decoder p_θ) which would allow us to perform gradient descent to optimize ϕ. **Show that** $\nabla_\phi \mathbb{E}_{z\sim q(z|x)}[\log(p_\theta(x|z))] = \mathbb{E}_{z\sim q(z|x)}[\log(p_\theta(x|z))\nabla_\phi(\log(q_\phi(z|x)))]$**. What would an unbiased estimator of this value be? Intuitively, why would this estimator be higher variance than the one derived using the reparameterization trick?**

Hint: expand out the expectation as a summation over all possible values of z*. Recall that* $\nabla_x \log(f(x)) = \frac{\nabla_x f(x)}{f(x)}$ *for scalar-valued* f

$$\begin{aligned}
\nabla_\phi \mathbb{E}_{z\sim q(z|x)}[\log(p_\theta(x|z))] &= \nabla_\phi \sum_z [q_\phi(z|x)\log(p_\theta(x|z))] \\
&= \sum_z \nabla_\phi [q_\phi(z|x)\log(p_\theta(x|z))] \\
&= \sum_z [\log(p_\theta(x|z))\nabla_\phi(q_\phi(z|x))] \\
&= \sum_z [q_\phi(z|x)\log(p_\theta(x|z))\frac{\nabla_\phi(q_\phi(z|x))}{q_\phi(z|x)}] \\
&= \sum_z [q_\phi(z|x)\log(p_\theta(x|z))\nabla_\phi(\log(q_\phi(z|x)))] \\
&= \mathbb{E}_{z\sim q(z|x)}[\log(p_\theta(x|z))\nabla_\phi(\log(q_\phi(z|x)))]
\end{aligned}$$

Where we use the linearity of the gradient operator, the fact that $\log(p_\theta(x|z))$ is independent of ϕ, and the definition of expectation. An unbiased estimator for this value is just a single sample of the expression inside the expectation, $\log(p_\theta(x|z_0))\nabla_\phi(\log(q_\phi(z_0|x)))$ where $z_0 \sim q(z|x)$. Intuitively, this is a high variance estimator because it will always pull ϕ in a direction which makes the likelihood of the sampled z_0 larger, even if that particular z_0 is not a good

latent representation (i.e. the likelihood of the probabilistic decoder outputting x given z_0 as an input is fairly low). We rely on the fact that $p_\theta(x|z_0)$ is larger for good latent representations to push ϕ in the right direction in expectation.

2.4 Ablation & Visualization Questions

1. After visualizing the losses between AE and VAEs as well as their reconstruction, does anything stand out to you? How do you explain the difference in performance? If you'd like, play around with the latent dimension size (e.g. 512, 256, 128) and the regularization weight for the VAE (e.g. .01, .001, .0001). What do you notice?

   Answers vary somewhat for answers in this subsection. However, students might be surprised (a) by how well both models did in reconstructing the images with so little training resources (can be chalked up to the simplicity of the dataset and (b) by how the VAE didn't explicitly do better than the AE in terms of reconstruction.

   The explanation for this comes from the fact that we added a regularizer that prevents the model from overfitting. Thus, we note a slightly worse loss and image quality. *However, the point of regularization is to provide better generalization to more complex tasks.* So it can be said that the innate simplicity of MNIST is a factor (overfitting not much of an issue).

   Finally, when it comes to latent dimension, notice that lower the size does increase the number of trained parameters, normally associated with a clear-cut improvement in performance. However, since we are limited in our expressive capacity by the bottleneck, the relationship here isn't as clear-cut as in other deep-learning architectures. For the regularization weight, students should find the smallest option given performs the best.

2. What do you notice about the differences in the layout of the latent spaces between AE and VAE? How can you explain this difference?

3. Compare the loss performance between the denoising and no noise case. Did either of the models relatively improve in the denoising case? Explain why this model may be better at denoising. Try different noise_weight values and comment on relative changes in performance (e.g. .25, .50, .75).

   Joshua, essentially want them to say that VAE gets better with more noise, and so, it's better at denoising (I'll leave wording and explanation to you).

4. Comment on the difference in quality of sample generation between AE and VAE. 5 epochs might not be enough to generate high-quality samples (we recommend retraining on 25), but the relative difference should still be obvious. Explain what causes the difference.

   The vanilla autoencoder produces vastly inferior samples in comparison to the VAE. In particular, the latent space that it learns is most likely sparse, so sampling it randomly is almost certain to yield something which is meaningless and out-of-distribution for the decoder network, resulting in low-quality samples being generated. The VAE "molds" its latent space into a Gaussian, which is not sparse, and it is much easier to generate high quality samples from. This also affords a way to interpolate between different latent variables as well while maintaining a reasonable-looking output.

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 continuous 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 hyperparameter of the model) that will represent our discrete distribution. Figure 1 represents this.

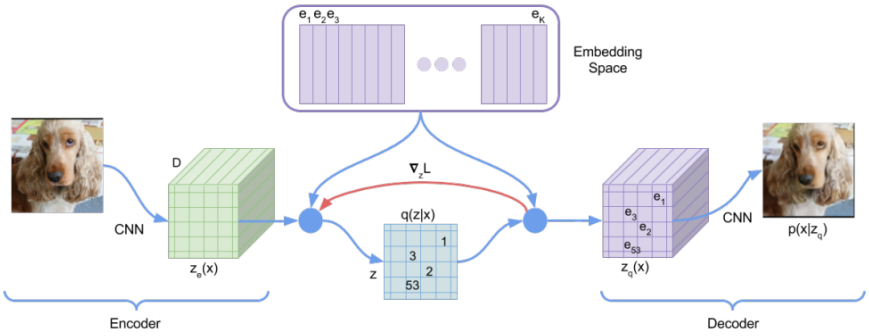

Figure 2: The VQ-VAE Architecture

Suppose that our vector embeddings all live in $\mathbb{R}^d$, and our encoder (possibly a convolutional neural network) produces n vectors in $\mathbb{R}^d$ (e.g. in the case of a convolutional network, n would be the number of pixels in the final output and d would the the number of channels in the final output). **Give one way we can sample the discrete latent space using the output of the encoder**. (note: the original paper uses the term "latent embedding space" to refer to $\mathbb{R}^{k \times d}$ the set of embedding vectors, and "latent feature space" to refer to the 1D, 2D, or 3D set of features where each feature is a vector in the embedding space).
The most natural way to sample the discrete latent space is to use a nearest neighbors approach and simply map each vector of the encoder output to the closest vector in the set of embeddings. Alternatively, you could also imagine randomly choosing one of the embedding vectors weighted by the distance to the encoder output.

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 backpropagate to the encoder. 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 $\text{sg}(x)$ represent a function that blocks gradients to backpropagate through. Define an expression for the input of the decoder that allows the decoder gradients to be copied over to the encoder using $\hat{x}, \tilde{x}$, and the stopgradient function $\text{sg}(x)$.**

$$\tilde{x} + \text{sg}(\hat{x} - \tilde{x})$$

This somewhat unintuitive looking expression passes the gradients directly through to the input of the quantization step. In particular, letting y be the input to the decoder, $\frac{\partial \mathcal{L}}{\partial \tilde{x}} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial \tilde{x}} = \frac{\partial \mathcal{L}}{\partial y}$ due to the fact that y is the identity function for $\tilde{x}$ plus a stopgradient term that prevents any gradients from passing through to the $\tilde{x}$ within.

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 encoder outputs $z_e(x)$, provide a term that will allow our sampled encodings to update to be closer to our encoder outputs.**

$$\|e - \text{sg}(z_e(x))\|_2^2$$

This term pushes the embedding vector that some feature of the encoder output was quantized to towards the value of that feature.

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.**

$$\|\text{sg}(e) - z_e(x)\|_2^2$$

This term pushes the encoder output to be closer to the vector in the embedding space that it was quantized to, pushing the encoder to "commit" to a particular encoding. This is important since the effective learning rates of the encoder and of the embeddings is not necessarily equal, and as mentioned in the problem this could cause the encoder output to grow unboundedly unless a term is added to counteract that effect.

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.**
   One potential idea is to simply sample uniformly among the K different embedding vectors for each feature. As you will see in one of the notebook demos, this does not work particularly well. A common approach is to build a powerful autoregressive prior fitted on the encoder outputs of your training data using an architecture like PixelCNN that outputs a probability distribution for the next latent feature given all of the ones generated before, not unlike the autoregressive sequence models we've seen in class (in 2D, sampling the outputs in a raster pattern, one row at a time) .

3.3 Ablation & Visualization Questions

1. You should notice a difference between your AE/VAE losses and your VQ-VAE losses. **Which of these is higher? And explain why.**
   The VQ-VAE losses are higher. This makes sense as we have now added a vector quantization loss element. It is not just about minimizing reconstruction in the loss objective anymore; the loss objective is more complex. We will see why in the next question.

2. Take a look at the reconstructed samples closely. **Compare and contrast the relative visual quality of the reconstruction between AE, VAE, and VQ-VAE. Now focusing on the VAE and VQ-VAE**

results, what particular qualities of the image do you notice are different? Explain what causes this.

We find the relative visual reconstruction quality of the three models to be AE, VQ-VAE, and finally VAE in descending order. Between the VAE and VQ-VAE, one might notice that the VQ-VAE has sharper edges and more defined object outlines. This is a result of the model having discrete priors for its latent representation rather than a continuous one!

3.4 References

Any references utilized in this project can be found in the README of our repo.