10-315 Notes
Variational Autoencoders

Carnegie Mellon University
Machine Learning Department

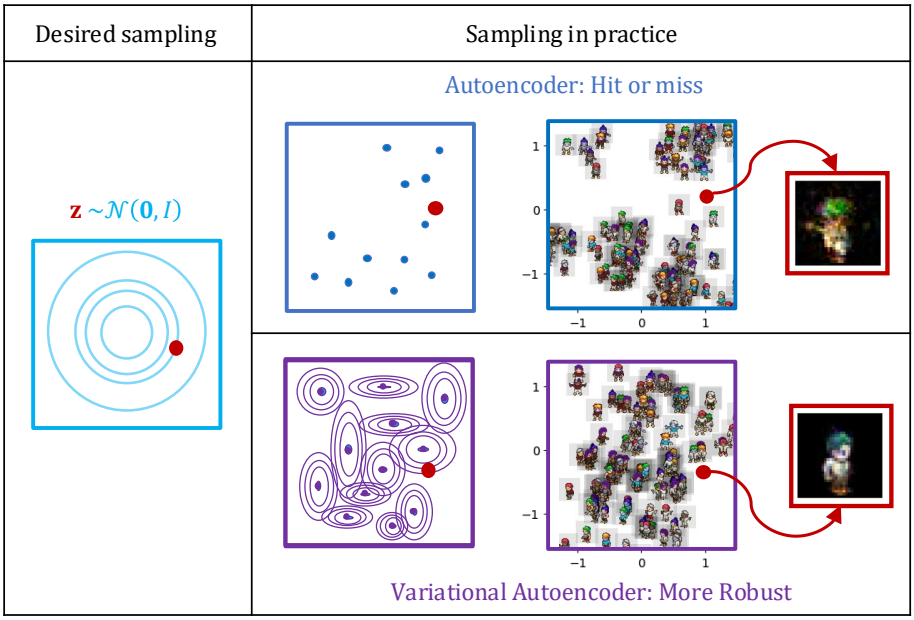
Contents

1 VAE Overview

Recall that for an autoencoder, our goal was to encode data into a latent space. We then sample specific points from the latent space and feed them through a decoder. This allows us to generate new images (or other types of high-dimensional data). However, this process has 2 main weaknesses:

1. There can be gaps in the latent space. When we sample specific points in the latent space, they are highly likely to produce gibberish if we have not encoded similar data.
2. We don't really know which areas in the latent space are likely to have higher quality embedded regions.

Hence, rather than having the latent space simply be a bunch of points, what if we made it a probability distribution instead? In other words, we encode the input data to specific parameters of a distribution, and then the decoder could sample from that distribution and then go through the decoding process. This would fix both problems above, and would make sampling a bit more "continuous", since we could directly sample from the distributions!



2 From Autoencoders to Variational Autoencoders

Here are a few ideas that will get us from an autoencoder to a variational autoencoder.

2.1 Idea: Add noise during training

Let's start with the same autoencoder training process, where we map each input to a point in the K-dimensional embedded space and then convert that point back to the original size, hoping to perfectly reconstruct the input data (or at least minimize the L2 reconstruction error). However, to make the autoencoder more robust, every time we map a point to the K-dimensional space, we'll add a bit of random noise to that embedded vector and still try to perfectly reconstruct the input data. This added challenge is analogous to humans training with weights or a blindfold to make them perform better later.

$$input \longrightarrow embedded\ point + noise \longrightarrow reconstruction$$

We're going to add randomly sampled noise from a multivariate Gaussian distribution. Let $\mathbf{z}_{original}$ be the K-dimensional output of the encoder network. To add noise, we can just sample a K-dimensional noise vector, $\boldsymbol{\epsilon}$, from a Gaussian distribution with zero mean and Σ covariance and then add it to the embedded point:

$$\mathbf{z}_{original} = f_E(\mathbf{x}, \phi) \tag{1}$$
$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \Sigma) \tag{2}$$
$$\mathbf{z}_{noisy} = \mathbf{z}_{original} + \boldsymbol{\epsilon} \tag{3}$$

Another way to look at this is that the encoder output, $\mathbf{z}_{original}$, is the K-dimensional mean of a Gaussian distribution with covariance Σ:

$$\boldsymbol{\mu} = f_E(\mathbf{x}, \phi) \tag{4}$$
$$\mathbf{z}_{noisy} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma) \tag{5}$$

As we'll see when we dig deeper into variational autoencoders, both of these equivalent formulations of a noisy embedded vector will be helpful.

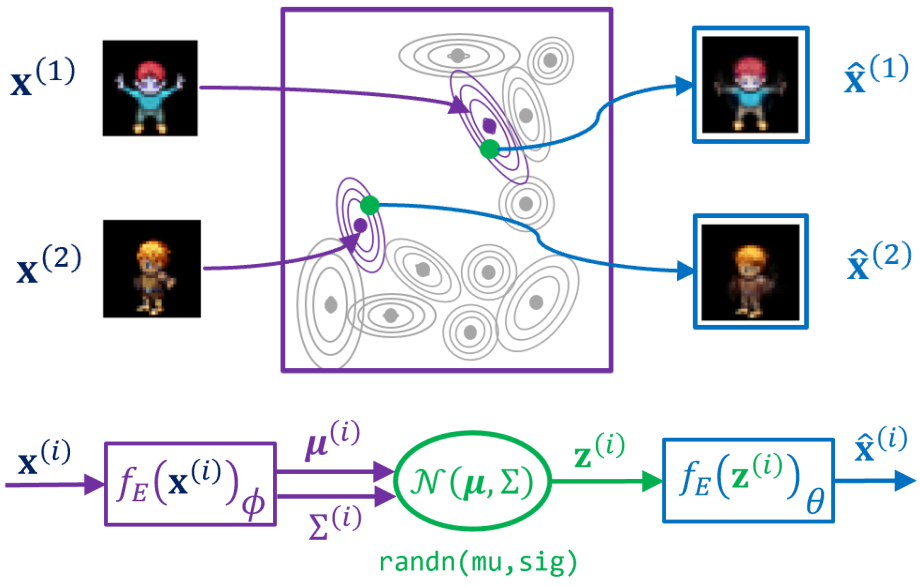
2.2 Idea: Learn how much noise to add to the embedded point

But how much noise do we add? In other words, where do we get the covariance Σ? Well, we could just use a standard identity covariance matrix, I. Alternatively, we could design the encoder network to output a few more values, so that it gives us the covariance values in addition to the mean:

$$\boldsymbol{\mu}, \Sigma = f_E(\mathbf{x}, \phi) \tag{6}$$
$$\mathbf{z}_{noisy} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma) \tag{7}$$

Cool, now every time a training input is passed through the encoder, we get the original embedded point (the mean) and a covariance for that embedded point. We then sample the noisy version of the embedded point and then pass it through the decoder to try to reconstruct the original input.

Note, as we force our upgraded autoencoder to handle noise added to the embedded vectors, the decoder learns to generate quality output from a broader range of regions in the embedded space.



2.3 Idea: Regularize toward standard normal

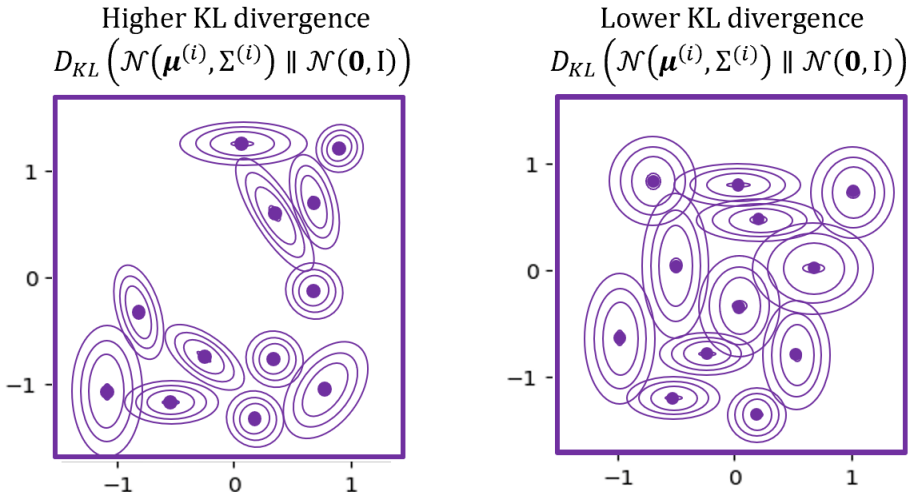
So, we've been taking steps to make our autoencoder more robust and move the latent space closer to a distribution that is easier to sample from and generate new images. However, we're missing a big piece... What will happen to the covariance if we train this autoencoder with our standard reconstruction error loss as an objective?

While the covariance, and the associated Gaussian noise, is meant to help make the autoencoder more robust, it makes it much harder for the autoencoder to achieve its goal of zero reconstruction error. If we train with only the reconstruction error as an objective, the encoder network will quickly learn to shrink any variance and make the decoder network's job easier. Thus, leading us back to where we started where our embedded points are essentially point mass probabilities, $\mathcal{N}\left(\boldsymbol{\mu}^{(i)}, 0\right)$, i.e., a spike of probability density the embedded training points.

To keep our new normal distributions from collapsing, we're going to refocus on our goal of being able to sample the latent space, specifically using a standard normal distribution. For each training point, we'll keep the reconstruction loss in the objective, but we'll also add a second component to the objective to push the embedded space distributions toward the standard normal distributions:

$$\mathcal{N}\left(\boldsymbol{\mu}^{(i)}, \Sigma^{(i)}\right) \longrightarrow \mathcal{N}\left(\mathbf{0}, I\right)$$

To this end, our objective will also try to minimize the Kullback–Leibler (KL) divergence of $\mathcal{N}\left(\boldsymbol{\mu}^{(i)}, \Sigma^{(i)}\right)$ from $\mathcal{N}\left(\mathbf{0}, I\right)$ for each embedded training point.



2.4 Variational Autoencoder Objective

An now we have arrived at a variational autoencoder objective function that combines the reconstruction error with the KL divergence term:

$$J(\theta, \phi) = \sum_{i=1}^{N} \left(\underbrace{\left\| \mathbf{x}^{(i)} - f_D\left(\mathbf{z}^{(i)}; \theta\right) \right\|_2^2}_{\text{Reconstruction error}} + \underbrace{D_{KL}\left(\mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)}) \parallel \mathcal{N}(0, 1)\right)}_{\text{KL Divergence}} \right) \tag{8}$$

3 KL Divergence

Consider two probability distributions P and Q with pdfs (or pmfs) $p(x)$ and $q(x)$. We define the KL divergence of P and Q to be:

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) \, dx \quad \text{or} \quad \sum_{x \in X} p(x) \log\left(\frac{p(x)}{q(x)}\right)$$

Intuitively, this measures the "difference" between the probability distributions P and Q. As a sanity check, notice that if $p(x) = q(x)$ for all x, the divergence would be 0, indicating no difference.

Also, an important note is that KL Divergence is not symmetric. In other words, $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$. This is because we fix the first term to be the "true" distribution, and we see how much the second term differs from it.

You might notice that this also looks like an expected value, since it seems like we are taking the expected value of the differences over Q. Hence, some textbooks might also write it as:

$$D_{KL}(P \parallel Q) = \mathbb{E}_{x \sim p(x)}\left[\log\left(\frac{p(x)}{q(x)}\right)\right] = \mathbb{E}_{x \sim p(x)}\left[\log(p(x) - \log(q(x)\right]$$

which mathematically are all equivalent; it's just different notation.

Some of you might be thinking back to cross-entropy, which is also a measure of how "different" probability distributions are. In fact, they are quite similar! We have that:

$$\begin{aligned} D_{KL}(P \parallel Q) &= \sum_{x \in X} p(x) \log\left(\frac{p(x)}{q(x)}\right) \\ &= -\sum_{x \in X} p(x) \log(q(x)) - \left(-\sum_{x \in X} p(x) \log(p(x))\right) \\ &= H(P, Q) - H(P) \end{aligned}$$

where $H(P)$ is the entropy of P and $H(P, Q)$ is the cross-entropy of P and Q. If we assume that $H(P)$ is constant, then we are basically working with the same thing! Intuitively, the reason we use KL divergence over cross-entropy is so that we measure how much more uncertainty Q has over P.

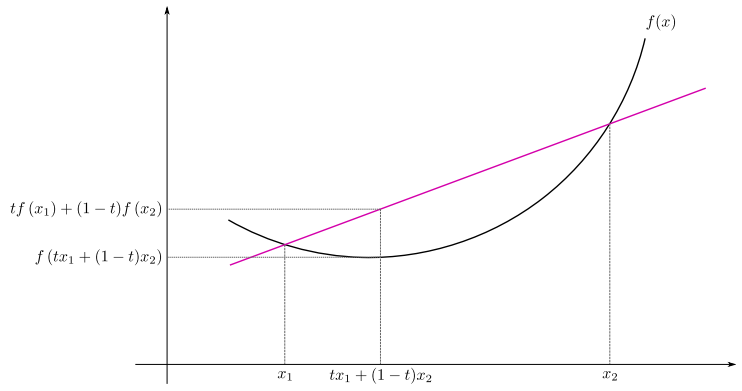
4 Helpful Math Resources

4.1 Expectation notation

We will be using a notation like $\mathbb{E}_{x\sim p(x)}[f(x)]$ quite often. All this means is to take the expected value of $\mathbb{E}[f(X)]$ given that X follows the distribution $p(x)$.

4.2 Jensen's inequality

A really useful inequality that we will use is Jensen's inequality, which states that for any convex function $f(x)$, the secant line should always lie above the graph:



Or mathematically,

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

for $t \in [0, 1]$. This can be extended to probability theory, where

$$\varphi(\mathbb{E}[X]) \leq \mathbb{E}[\varphi(X)]$$

when φ is convex. As an example, $-log$ is convex, so

$$-\log(\mathbb{E}[X]) \leq -\mathbb{E}[\log(X)] \implies \log(\mathbb{E}[X]) \geq \mathbb{E}[\log(X)]$$

5 Regular Autoencoder Objective:

5.1 Reconstruction error review

Recall that the objective function for a regular (non-variational) autoencoder is just the reconstruction error between the original input $\mathbf{x}^{(i)} \in \mathbb{R}^M$ and the reconstructed version $\hat{\mathbf{x}}^{(i)} \in \mathbb{R}^M$, which has been encoded and then decoded, $\mathbf{z}^{(i)} = f_E\left(\mathbf{x}^{(i)}; \phi\right)$ and then $\hat{\mathbf{x}}^{(i)} = f_D\left(\mathbf{z}^{(i)}; \theta\right)$:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)} \right\|_2^2$$

5.2 MLE formulation of decoder

In the same manner that linear regression mean squared error can be formulated as an MLE problem, we can arrive at the same reconstruction error objective from a probabilistic formulation.

Given an encoded sample from an input datapoint, $\mathbf{z}^{(i)} = f_E\left(\mathbf{x}^{(i)}; \phi\right)$, we model the noise around the j-th recostructed pixel as a normal distribution with a variance, τ^2, and a mean represented by the pixel value of the decoded latent representation, $\hat{\mathbf{x}}^{(i)} = f_D\left(\mathbf{z}^{(i)}; \theta\right)$:

$$p\left(x_j^{(i)} \mid \mathbf{z}^{(i)}\right) \sim \mathcal{N}\left(\hat{x}_j^{(i)}, \tau^2\right)$$

If we treat all these noisy reconstructed pixels as *i.i.d.*, we have the following M(C)LE formulation:

$$\begin{aligned}
\underset{\theta,\phi}{\operatorname{argmin}} & -\log p(\mathcal{D} \mid \theta, \phi) \\
& -\sum_{i=1}^{N}\sum_{j=1}^{M} \log p\big(x_j^{(i)} \mid \mathbf{z}^{(i)}\big) \\
& -\sum_{i=1}^{N}\sum_{j=1}^{M} \log \frac{1}{\sqrt{2\pi\tau^2}} e^{\frac{-\left(x_j^{(i)} - \hat{x}_j^{(i)}\right)^2}{2\tau^2}} \\
& -\sum_{i=1}^{N}\sum_{j=1}^{M} \log \frac{1}{\sqrt{2\pi\tau^2}} - \frac{1}{2\tau^2}\left(x_j^{(i)} - \hat{x}_j^{(i)}\right)^2 \\
& \sum_{i=1}^{N}\sum_{j=1}^{M} (x_j^{(i)} - \hat{x}_j^{(i)})^2 \\
& \sum_{i=1}^{N} \left\| \mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)} \right\|_2^2 \\
& \sum_{i=1}^{N} \left\| \mathbf{x}^{(i)} - f_D\left(f_E\big(\mathbf{x}^{(i)}\big)\right) \right\|_2^2
\end{aligned}$$

Whoa, that seems much more complicated than it needs to be. Why couldn't we just stick to the reconstruction loss without all the probability?

Well, as we dig into variational autoencoders, it will be helpful to have the probabilistic model of the decoder, $p\big(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}\big)$. With variational autoencoders, we'll model the encoder as a probability distribution also, $q\big(\mathbf{z}^{(i)} \mid \mathbf{x}^{(i)}\big)$, which will help us move closer to the desired sampling distribution $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, I)$.