CS182 - Scribe Notes 11/29

Zipeng Lin, Numi Sveinsson

November 29th 2022

1 Introduction

Today the topic of the lecture are VAEs (variational autoencoders).

2 Recall from last time

Two ideas that don't quite work:

1. Plain autoencoder: we have an input X_i sent into the encoder and decoder to get $\hat{X}_i$ at training time. At testing time, we just use the encoder to generate the model. (bottleneck) The reason this might not work is because we need to decide from which distribution to sample the space $\vec{z}$ and we don't know that distribution.

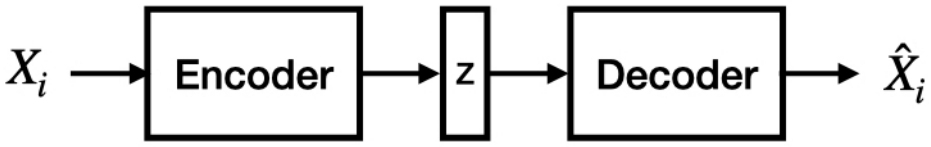

Figure 1: Autoencoder

   The distribution we input would be different from the output.

2. Plain repeated denoising generation. Multiply the data with $\sqrt{B_i}$, and then add a noise that is $N(0, 1 - B_i)$. To make the sequence long enough, we will get it would look Gaussian if T is large. Then we train a denoiser network that takes samples from this space X_T.
   Why does this fail: the samples from the space are very blurry after running through the denoisers. Each of the trained denoisers go a little outside the correct distribution and together it ends up being an accumulative error.

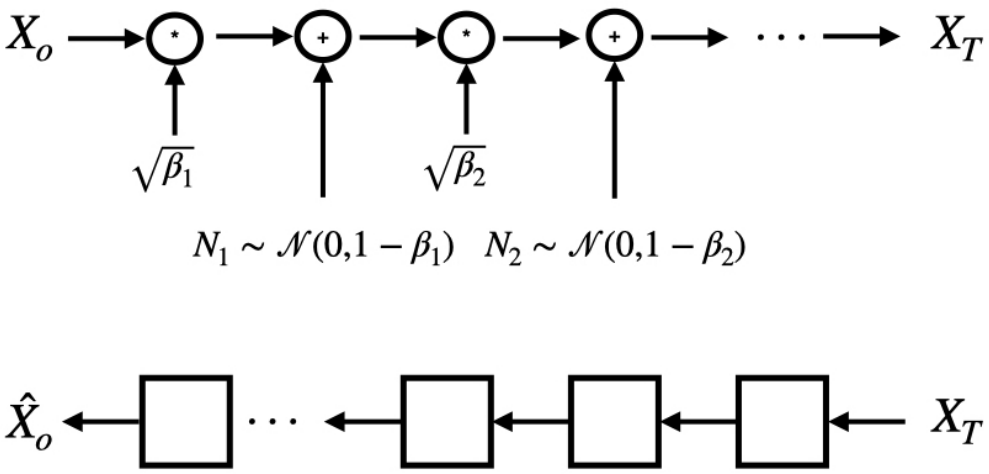

Figure 2: Denoise generation. The bottom row represents learned denoisers.

The distribution would be blurry too.

Now let's talk about how we fix the first approach above.

3 VAE: Variational Autoencoders

Intuition: decoder is just for training, encoder is for dimension reduction.
Question: from a hacking point of view, how to know the unknown distribution of $\hat{z}$?
Solution: force $\hat{z}$ to have a particular known distribution.

3 Ingredients:

1. Make the input to the "decoder" actually random during training.
2. Add a loss term on the distribution of $\hat{z}$. Make it look like what we want.
3. Parameterize so that we can do SGD.

For example, X will be sent to an encoder to a distribution of some kind and sent to loss and output a loss. Then, the sample from the distribution will be sent to a decoder and get $\hat{X}_i$, input the x_i to calculate loss and output $\mathbb{R}$.

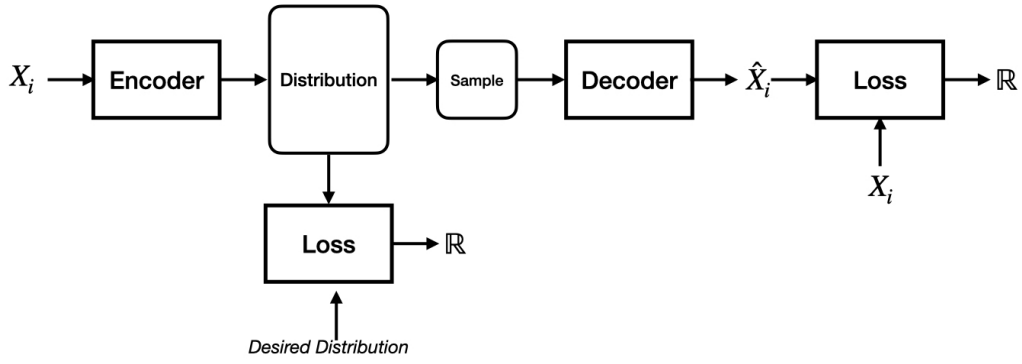

Figure 3: VAE

Questions: if we just take the input data, there are some amount of training data, we will get a collection of data point. If we treat the empirical data points as a distribution, we only get the reconstruction. We want to understand the distribution being generated. We can perturb those points at testing time. We want to have more diversity to emerge.

3.1 Desired Distribution

Desiderata for distribution:

1. Continuous
2. Easy to sample
3. Easy to compute the loss on distributions.

There are two candidates that we usually use: uniform distribution and Gaussian distribution. We choose normal distribution $N(\vec{0}, I)$.

Now we can just have the output from the encoder being a mean and covariance of that normal distribution. Now we have parameterized the distribution and can use SGD to change these variables.

Aside: we have the output being the squared root of the covariance to ensure positive definiteness later.

Sampling: is done by sampling a normal distribution $\hat{\epsilon}$, multiplying that with the covariance and adding to the mean.

3.2 Loss on Distribution

What loss should we use? How do we measure the distance between two probability distributions? There are lots of choices.

Choose KL divergence $KL(Q \mid P)$ "Relative entropy from P to Q", we have

$$KL(Q \mid P) = \int Q(z) \ln \frac{Q(z)}{P(z)} dz = E_{Z \sim Q}[\ln \frac{Q(z)}{P(z)}] \geq 0$$

this is nonnegative by Jensen's Inequality (proved in discussion) since ln is concave.

More importantly than being nonnegative, if two distributions are similar then it would be zero. It is also asymmetric. It means "it does not like it when Q puts lots of probabilities where P does not": this means if $P(z) \ll Q(z)$

the value would be really big. Follow the aside example to experience more about KL being asymmetric.

Aside: If we have iid draws from P, probability that it looks like they are drawn from Q is around $e^{-nD(Q|P)}$. If Q is a coin with only heads and P is half heads half tails, it is not going to happen. On the other hand, it might happen but not likely.

In particular, the KL divergence, $KL(Q \mid N(0, I_k)) = 1/2(\mathrm{Tr}(\Sigma_Q) + \vec{\mu_Q}^\intercal \vec{\mu_Q} - k - \log\det\Sigma_Q)$. It is important because PyTorch can take derivative for us. The gradient could flow back to the mean and covariance to update them. We try to regularize by imposing a loss term saying that please make the KL loss be normal.

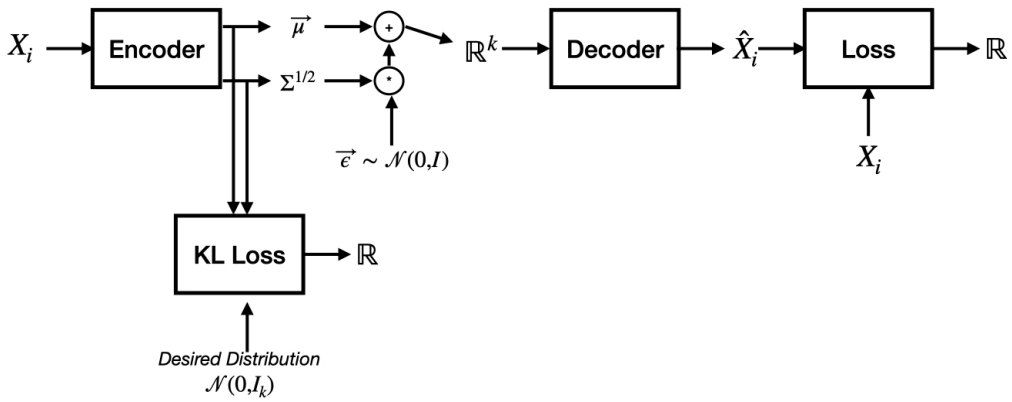

Figure 4: VAE - Enforcing Normal Distribution with KL-Divergence Loss

Question: can we do SGD? Does the loss affect the weights in the encoder? The spirit of SGD says we look at the entire distribution, while SGD says we take a sample and do it. We have x_i is random from training set. Gradient would hit the decoder, and would go towards $\hat{\mu}$ and $\Sigma^{1/2}$ because it would go through the multiplication. We can inject noise to the term D and add regularization term to make the system more robust.

3.3 Training

Challenges:

1. KL loss ends up dominating, we disconnect input. In this case the decoder carries no information, so it collapses. It would set "blurry average" outputs to D.

2. Alternatively, if the reconstruction loss dominates too much then we will get bad samples when we use it.

We need to have a balance between them.

3.4 Tricks

- Often reconstruction loss is allowed to dominate early in training and then distribution loss is used afterwards to tweak the sampling space without impacting reconstruction performance.

- During inference: the sample from $\vec{z}$ is run through decoder but then **rerun** through the encoder to refine the sample. That output is then used for generation. We only do this once or twice to prevent it from becoming 'mush'.

4 Diffusion Models with Denoising

Suppose we want x_t based on x_0

$$X_t \sim N(\sqrt{\alpha_t}X_0, (1-\alpha_t)I)$$

those are easy to understand and also for any s

$$X_t|X_s, s < t$$

What we want is to go backward for $X_{t-1}|X_t$, which is hard (this is intractable misbehavior).

Key idea 1: We approximate $X_{t-1}|X_t$ with $N(\mu_\theta(x_t,t), \Sigma_\theta(x_t,t))$. If we make those increments small enough, maybe the reverse thing might be the same. We want to do the same thing that we did VAE, we want to have a loss at the level of distribution that has what we want.

VAE-style idea: add a loss $KL(\dots \mid \dots)$ on this distribution.
We want to target something on the reverse path.

Key idea 2: $X_{t-1}|X_t, X_0$ is intractable by joint normality, then we will get a normal distribution with its means and variance. We can train the denoising autoencoder with using the same kind of regularization. Every time we do this we inject some noise. The added noise is what gives robustness.

Denoising is a form of feedback control.