1 Introduction

Assume that you are given some dataset X and assume that this dataset follows some unknown distribution $P_{\text{actual}}(x)$. The goal of the autoencoder is to learn a distribution $P_{\text{predicted}}(x)$ such that $P_{\text{predicted}}(x)$ is as similar as possible to $P_{\text{actual}}(x)$

1.1 Latent Variable Models

a latent variable is when the model generates anything, it first randomly samples the "object" it want to create.

For example, if we have a model that generates a image of a handwritten digit, the latent variable would be a random digit value $z \in \{0, 1, 2, \ldots, 9\}$

Assume we have a vector of latent variables z which exists in a high dimensional space Z. In this space we can easily sample from this space using a probability density function P(z).

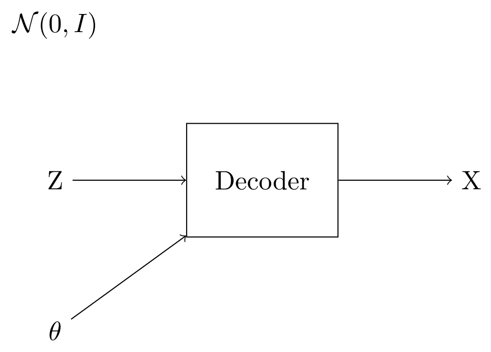
Essentially we have a family of functions parameterized by θ. This means that θ defines the behavior of the function. The objective is to find the best θ such that when we sample inputs z from a distribution $P(z)$ (the latent space originally mentioned), the outputs of the function $f(z; \theta)$ closely match the data in our dataset X. This means that every combination of z and $f(z; \theta)$ will map to some point X.

Mathematically we aim to maximize the probability of each x in the training set. This is modeled by the following equation where z is a d dimensional vector in the latent space and θ is the learned parameters of the model.

$$P(X) = Z \int P(X|z; \theta) P(z) \, dz$$

$f(z; \theta)$ is replaced by a probability distribution $P(X|z; \theta)$. This focuses the probability of X dependent on z. The goal of this is maximum likelihood, where we need to find the best parameters for a probability distribution such that the observed data is maximally probable according to that distribution.

For example, with variational autoencoders, the $P(X|z; \theta)$ distribution we choose is a normal distribution, where the parameters we want to optimize are the mean μ and variance σ^2 for the latent space. Note that z here is a random point in a normal distribution with a mean of 0 and a covariance matrix I.



Using a Gaussian distribution and gradient descent we push *theta* to maximize $P(X)$for a given z. This distribution $P(X|z)$ is not required to be a Gaussian distribution, using it allows us to use stochastic gradient descent.

2 Variational Autoencoders

variational autoencoders learn by taking z from a normal distribution and then putting it through a function g, to get an arbitrary distribution. How do we choose z though? Choosing z can be difficult because not only are there complex connected properties of the image but also the model must learn these connections. Auto encoders make the simple assumption that that samples of z can be drawn from a normal distribution with mean 0 and identity matrix.

Given a distribution that exists in d dimensions, we can take a set of d variables such that $\mathbf{d} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and put these variables through a complicated function to generate the distribution in d dimensions.

To compute $P(X)$ approximately, we first sample a large number of z values $\{z_1, \ldots, z_n\}$, and compute $P(X) \approx \frac{1}{n} \sum_{i=1}^{n} P(X|z_i)$. The problem here is that in high-dimensional spaces, n might need to be extremely large before we have an accurate estimate of $P(X)$.

2.1 Objective Function for sampling z

Let us define a new function $Q(z|X)$ which takes in X and will give a distribution of z values that are likely to produce X. Instead of now guessing all possible z values we can use a distribution.

This means we can easily compute and equation like $\mathbb{E}_{z\sim Q}[P(X|z)]$ which is the average likelihood of observing X over all possible values of z drawn from the distribution $Q(z)$.

Theorem Using KL divergence loss we look at how a distribution of $Q(z)$, which may not be from a normal distribution is similar compared to the distribution $P(z|X)$.

$$D[Q(z)||P(z|X)] = \mathbb{E}_{z\sim Q}[\log Q(z) - \log P(z|X)]$$

Using Bayes Equation we can transform

$$\log P(z|X) = \log \frac{P(X|z) \cdot P(z)}{P(X)}$$

and the using log rules we get

$$\log P(z|X) = \log P(X|z) + \log P(z) - \log P(X)$$

which combined into the previous equation gives us

$$D[Q(z)||P(z|X)] = \mathbb{E}_{z\sim Q}[\log Q(z) - \log P(X|z) + \log P(z) - \log P(X)]$$

because X is not dependent on z and by that extension Q(z) we can change the equation to

$$D[Q(z)||P(z|X)] = \mathbb{E}_{z\sim Q}[\log Q(z) - \log P(X|z) - \log P(z)] + \log P(X)$$

by rearranging equation we can create a KL function on the right hand side

$$\log P(X) - D[Q(z)||P(z|X)] = \mathbb{E}_{z\sim Q}[\log P(X|z)] - D[Q(z)||P(z)]$$

because we want to maximize P(X) we want a Q that does depend on X such that it makes the $D[Q(z)||P(z|X)]$ small. We can change the statement such that the right hand equation can be changed to

$$\log P(X) - \underline{D[Q(z|\overset{0}{X})||P(z|X)]} = \mathbb{E}_{z\sim Q}[\log P(X|z)] - D[Q(z|X)||P(z)]$$

Note that in $D[Q(z|X)||P(z|X)] = \mathbb{E}_{z\sim Q}[\log Q(z|X) - \log P(z|X)]$ Q is "encoding" X into z, and P is "decoding" it to reconstruct X. If both get the same log probability the loss is zero, meaning that the model has learned the correct mapping for X and hence has maximized P(X). Therefore while P(z—X) isn't analytically computable we can just solve for $Q(z|X)$ to get $P(z|X)$ value.

In order to optimize this function we use mathematical calculations to see that if we sample epsilon from a normal distribution with mean of 0 and the covariance matrix being the identity matrix we can get sample z to the following equation

$$z = \mu(X) + \Sigma^{1/2}(X) \cdot e$$

we use this equation in the book by only using d dimensional vectors, with d being a vector and d being the dimension of both the mean and the log variance. Note that because the covariance matrix is just a diagonal we actually have a d dimension vector, same as mean, otherwise we would have to do matrix multiplication. Because we are taking log of variance to scale it between -infinity and +infinity, we get an equation like the following.

$$z = \mu(X) + \exp\left(\frac{1}{2} \cdot \log \Sigma(X)\right) \cdot e$$

3 B-Variational Autoencoders

this is when the KL loss is multiplied by a hyper parameter Beta and added to the reconstruction loss. Depending on the scenario, you can multiply by the reconstruction loss. Favoring one will lead to less favor in the other.

4 Textbook Notes

- Autoencoders use items unseen in the latent space to create generated items in the higher dimensional space
- Make sure to always have a activation function after a convulational layer as it adds non linearity in the model.
- Convulational Transpose layers work by taking a smaller input and adding appropriate padding so that the output size is the input size * stride. Convulational Transpose Layers work by adding 0s for the padding.

There are two main losses that are used in combination with autoencoders: Root Mean Squared Error Loss and the Binary Cross Entropy Loss.

Root Mean Square Error

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{1}$$

RMSE loss penalizes both overestimation and underestimation equally

Binary Cross Entropy Loss

$$\textit{Binary Cross Entropy Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\right] \tag{2}$$

BCE loss penalizes errors that lean toward the extremes much more heavily than errors that lean toward the center.

Problems with the normal autoencoder

- Larger bounded classes are more likely to be chosen even though its supposed to be a normal distribution

- Even in one boundary, there are large gaps in between points, making it hard for the autoencoder to correctly generate.
- Even if you have a close point like (-1.1,-1.1) to (-1,-1) there is no guarantee that (-1.1,-1.1) will be decoded correctly even if (-1,-1) is an actual training sample. This is because the distributions aren't continuous.

A solution to these problems is using a variational autoencoder. Essentially now images fall under a distribution, specifically a Gaussian distribution, with a mean of 0 and variance of 1.

A multivariate normal distribution is a probability distribution with a distinct bell shape curve defined by mean and variance. Using the math above we can use the following equation to sample a z from the latent space.

$$z = z_{\text{mean}} + z_{\sigma} \times \epsilon \tag{3}$$

where ϵ is a sampled point from a random, normal distribution with the same dimension as the latent space. Note for higher dimensions we use a mean vector and a diagonal covariance matrix, which can be represented as a vector with the diagonal elements of the matrix. Note that this assumes that the distribution is independent in each dimension, meaning variables are not correlated with each other. A variational autoencoder takes in an input and spits out a mean and log variance vectors. Then using equation 3 give a z that exists in the latent dimension.

The z_{mean} output is the mean point of the entire distribution and the z_{σ} is the log of the variance, σ^2, of each dimension. However equation 3 takes in a z_{σ}. Therefore we need a way to convert σ into a form of $\log(\sigma^2)$.

$$\begin{aligned} \sigma &= \exp(\log(\sigma)) \\ &= \exp\left(\frac{2\log(\sigma)}{2}\right) \\ &= \exp\left(\frac{\log(\sigma^2)}{2}\right) \end{aligned} \tag{4}$$

Therefore, we can now utilize equations 3 and 4 to create the following

$$z = z_{\text{mean}} + \exp\left(\frac{\sigma^2}{2}\right) \cdot \epsilon \tag{5}$$

Due to epsilon being completely random it can be considered a constant in derivation, and as such allows us to backpropogate

Adding on to earlier, the KL loss essentially is checking how different our distribution is compared to a normal distribution. Get too far and reconstruction may be great but not good enough to generate images. Goal is to get z_{mean} and $\log(z_{\sigma^2})$ to be as close to 0 as possible.

5 Manipulating Generated Images

To add a certain feature we first find the feature vector. We do this by taking the average of all points with the feature and subtract it from the average of all points without the feature. Computationally this can be done in batches where at the end of each batch we check the current average difference vector to the culminated average difference vector and given enough closeness distinguish an accurate feature vector. Then with an alpha of 0-1 we can affect a random point in the latent space.

$$z_{\text{new}} = z + \alpha \times \text{feature\_vector}$$

To morph images together we just need an alpha from 0-1 and the following equation

$$z_{\text{new}} = z_{\text{imageA}} \times (1 - \alpha) + z_{\text{imageB}} \times \alpha$$

These are just two instances of how we can do vector algebra in the latent space to affect the generated images, specifically the features.