Autoencoders

Saket Choudhary
saketkc@gmail.com

November 26, 2019

This document summarizes our current understanding of Autoencoders. The first section acts as a short introduction to Variational Inference which forms the backbone of Autoencoders. In section two we summarize the theory behind autoencoders and discuss VAEs in detail. The two sections may appear to be disconnected at first, so we reintroduce the

1 Variational Inference

A common problem in modern statistics is to be able to approximate difficult to compute probability distributions. The most common use of this in bayesian inference where the aim is to infer the unknown quantities using a posterior distribution.

Variational Inference converts the problem of inferring this probability distribution to an approximation problem. It can be thought of as an alternate to MCMC, but is faster and scales easily to larger datasets.

Consider we have a probability distribution $p(\mathbf{x})$. $\mathbf{z} = z_1, z_2, z_3 \dots z_m$ are m latent variables and $\mathbf{x} = x_1, x_2 \dots x_n$ are n observed random variables. Assume we have large number of datapoints.

$$p(x) = \int p(z)p(x|z)dz$$

1.1 Why not MCMC?

Our aim is to infer $p(z|x)$. In an MCMC setting, we would create an ergodic markov chain on z, whose stationary distribution is $p(z|x)$. We would then sample from this chain to collect samples from the stationary distribution and the posterior will be estimated empirically from a subset of these collected samples.

We need a faster way than MCMC to be able to do this, especially given our assumption of large dataset. So we will replace the slow step of sampling with optimization.

MCMC and Variational Inference (VI) essentially solve the same problem. MCMC samples a markov chain while VI solves an optimization problem. MCMC approximates the posterior of the chain while VI approximates through optimization.

1.2 Variational Inference

Let $\mathbb{Q}$ represent a family of approximate densities over the latent variables z. We want to find a member $q^*\mathbf{z}$ such that the KL divergence between this family and the posterior $p(z|x)$ is minimized:

$$q^*(\mathbf{z}) = \text{argmin}_{q(z)\in Q}\ \text{KL}(q(z)||p(z|x))$$

Hence our original inference problem has been converted to an optimization problem. The complexity of this optimization problem depends on how complex is our choice of family of distribution $\mathbb{Q}$. We want $\mathbb{Q}$ to be flexible enough so that $p(z|x)$ can be approximated closely but simple enough so that the problem is still tractable.

Our goal is to be able to approximate a conditional density of the latent variables $p(z|x)$ given observed variables $\mathbf{x}$.

Our strategy would be to use a family of densities over the latent variables, parametrizing them "free" variational parameters. Optimization will find the member of this family with the setting of these parameters that is closest in KL divergence to $p(z|x)$

1.3 Evidence Lower Bound

Consider the KL divergence equation:

$$\begin{aligned}
\text{KL}(q(z)||p(z|x)) &= \int q(z)\log(\frac{q(z)}{p(z|x)}) \\
&= \mathbb{E}_{\shortparallel}[\log(q(z))] - \mathbb{E}_{\shortparallel}[\log(p(z|x))] \\
&= \mathbb{E}_{\shortparallel}[\log(q(z))] - \mathbb{E}_{\shortparallel}[\log(\frac{p(x,z)}{p(x)})] \\
&= \mathbb{E}_{\shortparallel}[\log(q(z))] - \mathbb{E}_{\shortparallel}[\log(p(x,z))] + \mathbb{E}_{\shortparallel}[\log p(x)] \\
&= \mathbb{E}_{\shortparallel}[\log(q(z))] - \mathbb{E}_{\shortparallel}[\log(p(x,z))] + \log p(x)
\end{aligned}$$

However the above equation is intractable because $\log(p(x))$ is intractable. So we optimize an alternate function: $ELBO(q)$

$$\begin{aligned}
ELBO(q) &= \mathbb{E}_q[\log(p(x,z))] - \mathbb{E}_q[\log q(z)] \\
&= \mathbb{E}_q[\log(p(z|x)p(x))] - \mathbb{E}_q[\log q(z)] \\
&= \mathbb{E}_q[\log(p(z|x))] + \mathbb{E}_q[\log(p(z))] - \mathbb{E}_q[\log q(z)] \\
&= \mathbb{E}_q[\log(p(x|z))] + \mathbb{E}_q[\log(p(z))] - \mathbb{E}_q[\log q(z)] \\
&= -KL(q(z)||p(z)) + \mathbb{E}_q[\log p(x|z)]
\end{aligned}$$

ELBO lower bounds the (log) evidence. Assuming we have proven that KL divergence is non-negative $KL(q(z)||p(z)) \geq 0$, we get $\log(p(x)) \geq ELBO(q)$

So instead we maximize this $ELBO(q)$ function. The first term of the last equation implies we are penalizing $q(z)$ to be different from $p(z)$ and at the second term implies we are trying to maximize the expected log likelihood of observing x through $p(x|z)$.

The first term in the original $ELBO(q)$ equation is trying to maximize the expected complete log likelihood, which would generally involve an EM solution. When $q(z) = p(z|x)$ then the ELBO is simply $log(p(x))$. So in a traditional EM setup, the E step would involve computing $p(z|x)$ assuming it is tractable unlike in variational inference.

2 Autoencoders

Autoencoders are a part of the “Generative modeling" paradigm in Machine learning research, where the aim is to model the distribution $P(X)$ given datapoints X in some high dimensional space $\mathbb{R}^m$. They can be thought of neural networks that essentially try to copy the input to output.

More formally, autoencoders consist of an encoding and a decoding layer. Given an input $\mathbf{x}$, the encoding layer represents the input into its coded form $h = f(\mathbf{x})$ while the decoding layer provides a reconstruction of the original input $r = g(h) = g(f(x)) \approx x$ such that a loss $L(g(f(x)), x)$ is minimized, where L is a loss function that penalizes $g(f(x))$ for being dissimilar to x. Autoencoders with dimensions less than the original dimension of input $\mathbf{x}$ are called undercomplete autoencoders.

2.1 PCA

When the decoding function is linear and the loss is mean squared loss, the undercomplete autoencoders is known to span the same subspace as PCA.

2.2 A more intuitive explanation

The overall aim of autoencoders is to be able to model the true probability distribution $P_{true}(X)$ of a given dataset X. Since $P_{true}(X)$ is unknown, our aim is to learn an alternate distribution $Q(X)$ such that $Q(X)$ is "close" to $P_{true}(X)$. We want to avoid putting restrictions on the structure of data and we also want the solution to be tractable (in a computationally expensive context) even for large datasets.

2.3 Sparse Autoencoders

Sparse autoencoders are autoencoders with a sparsity penalty $\Omega(h)$ on the encoding layer $h = f(x)$.

$$L(\mathbf{x}, g(f(\mathbf{x})) + \Omega(h)$$

The sparsity constraint ensures that the autoencoder learns unique statistical properties of the data, rather than acting just as a copy-paste function.

We will focus on a special category of autoencoders called the “Variational Autoencoders" (VAE) for the rest of our sections.

2.4 Sparse Autoencoders and Latent Variables

The sparsity constraint in Sparse Autoencoders can be understood as an approximation of the maximum likelihood training of a generative model that has latent variables.

2.4.1 Latent Variables

Consider a setting where our dataset includes objects like trees, house, and cat and dog pictures. We want to be able to generate images which looks the ones in our dataset, but not exactly the same. If we are able to generate the left half of a tree, then the right half of it will be very different from the right half of a dog. Before we generate any image, it is useful (and essential) to think about the kind of image we are going to generate to ensure that the pixels are coherent with the image we are going to generate.

Before our model generates these images, it will randomly pick up collection of pixels z from $\{\text{tree}, \text{house}, \text{cat}, \text{dog}\}$ and make sure that all pixels are coherent in some sense. For example to generate a dog's image, you do not expect lot of green patches that would resemble a tree. This z is a latent variable. It is latent because given a new generated image, we do not know which settings of the latent variable generated the final image. z for example could be the patten of edges in our images.

2.5 Model

For the model to be representative of the data X, there should be at least on configuration of the latent variable z such that the data generated resembles all data points in X.

More formally, the distribution $P(X)$ is modeled using a generative process conditioned on the latent variable Z and we want to maximize $P(X)$. We have a way of sampling Z using some distribution $P(z)$ and this a deterministic function $f(z;\theta)$ parameterized by θ. We wish to optimize θ, such that $f(z;\theta)$ generates samples that look like X

We want to maximize $P(X)$ for each X in the training set, to be able to hope that this model will be able to generate "similar" samples.

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

where $P(X|z;\theta) = f(z;\theta)$.

In VAEs, the choice of this function $P(X|z;\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 I)$, but this is not a hard requirement. If X is binary, $P(X|z;\theta)$ could be bernoulli. It can be any distribution other than a dirac-delta, as long as it is computable.

VAEs solve this problem of "generative modeling" by telling us what our choice of latent variable z should be and how to overcome the integral involved over all such values of z.

Before our model starts generating any images, it needs to make some decisions. It needs to first decide what image it will be generating, than decide the angles, stroke width etc. We want to avoid deciding such properties by hand and we also want to avoid describing any sort of dependencies this latent variable should capture (the house images for example, always has more edges). VAEs take a very unintuitive approach of dealing with this by imposing a gaussian distribution on $z \sim \mathcal{N}(0, \sigma^2 I)$. How does that work?

Any distribution in d dimensions can be generated by taking a set of d gaussian random variables and passing them through some non-linear transformation. For example we can generate a 2D ring starting with a 2D multivariate gaussian using this transformation: $g(z) = z/\alpha + z/||z||$ where $z \sim N(0, \sigma^2 I)$ and α is a shrinkage parameter.

An example is given below:

2.6 Objective function

For most choices of z $P(X|z)$ will be close to zero. So VAEs tend to sample only those values of z which are more probably of generating X. So we are interested in a function $Q(z|X)$ that takes the data X and gives us a distribution over z that are likely to produce X. Ideally, the space spanned by $Q(z|X)$ is much smaller than the "prior" distribution $P(z)$.

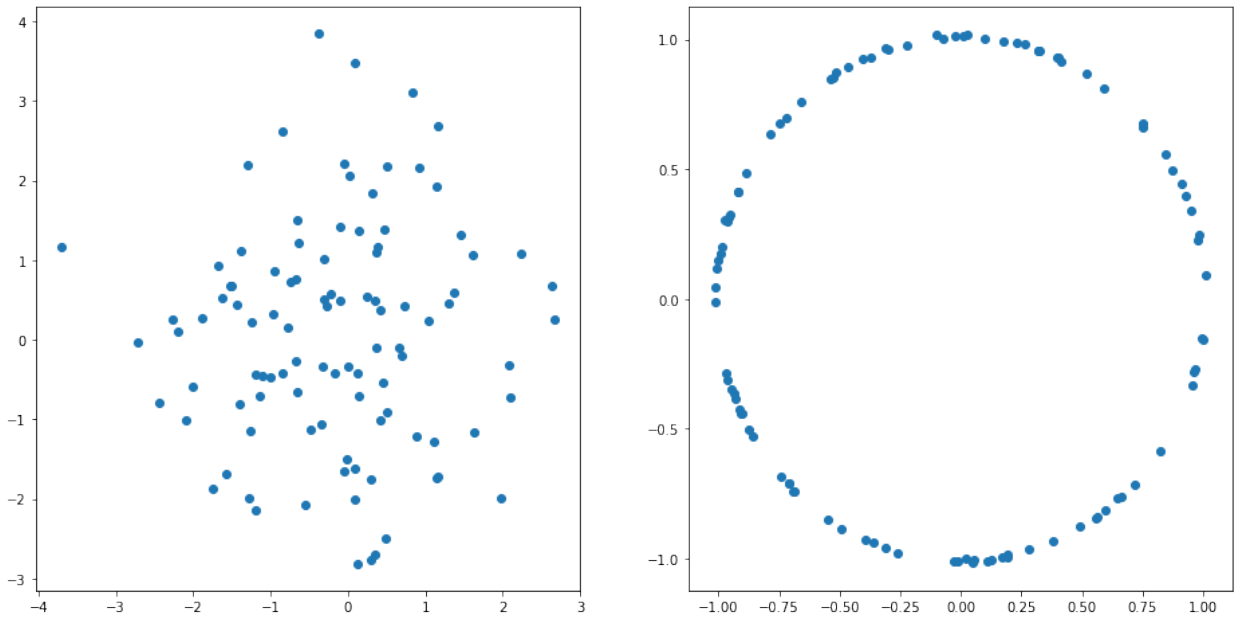

Figure 1: It is possible to transform a gaussian (left) to look like a ring (right)

$$
\begin{aligned}
KL(Q(z)||P(z|X)) &= E_{z\sim Q}[\log Q(z) - \log P(z|X)] \\
P(z|X) &= \frac{P(X|z)P(z)}{P(X)} \\
KL(Q(z)||P(z|X)) &= E_{z\sim Q}[\log Q(z) + \log P(X) - \log P(X|z) - \log P(z)] \\
KL(Q(z)||P(z|X)) &= E_{z\sim Q}[\log Q(z) - \log P(z)] + \log P(X) - E_{z\sim Q}[\log P(X|z)] \\
KL(Q(z)||P(z|X)) &= KL(Q(z)||P(z)) + \log P(X) - E_{z\sim Q}[\log P(X|z)] \\
\log P(X) - KL(Q(z)||P(z|X)) &= E_{z\sim Q}[\log P(X|z)] - KL(Q(z)||P(z))
\end{aligned}
$$

We want to maximize the quantity on left hand side because we want to maximize $\log(P(x))$, however we also want to penalize $Q(z)$ if it does not resemble the "true" unknown distribution $P(z|X)$ Alternatively, our loss function for performing gradient descent is given by:

$$KL(Q(z)||P(z)) - E[\log P(X|z)]$$

where the first term denotes the information lost in representing $P(z)$ as $Q(z)$ and the second term denotes the reconstruction loss

So we want $KL(Q(z)||P(z|X))$ to be minimized. If we find a magic function $Q(z)$ which can approximate $P(z|X)$ perfectly, $KL(Q(z)||P(z|X)) = 0$.

2.7 What should be ideal $Q(z|X)$?

Answer: $Q(z|X) = \mathcal{N}(z|X, \sum)$ just works.

To perform gradient descent on the right hand side, we take one sample of z and treat one sample of that as an approximation of $E[\log P(X|z)]$, this is performed over all datapoints, say D

$$E_{X\sim D}[\log P(X) - KL(Q(z)||P(z|X))] = E_{X\sim D}[E_{z\sim Q}[\log P(X|z)] - KL(Q(z)||P(z))]$$

We can just take the gradient of this equation and everything looks okay. However, during backpropogation, we will encounter a layer that is sampling z from $Q(z|X)$ which is itself a stochastic unit. In order to be able to sample from $Q(z|X) \sim \mathcal{N}(\mu(X), \sum(X))$ we first sample $\epsilon \sim \mathcal{N}(0, I)$ and then compute $z = \mu(X) + \sum^{1/2}(X)\epsilon$. This is called the re-parameterization trick. This was the main contribution of the original VAE paper [1].

2.8 KL divergence between two gaussians (Used above for calculating losses)

Assume $q \sim \mathcal{N}(\mu_0, \sigma_0^2)$ and $p \sim \mathcal{N}(\mu_1, \sigma_1^2)$:

$$\begin{aligned}
KL(q||p) &= \int q(x)(\log(q(x) - \log(p(x))dx \\
q(x) &= (2\pi\sigma_0^2)^{-\frac{1}{2}} \exp \frac{-(x-\mu)^2}{2\sigma_0^2} \\
\log q(x) &= -\frac{1}{2}\log 2\pi - \log(\sigma_0) - \frac{-(x-\mu)^2}{2\sigma_0^2} \\
KL(q||p) &= \int q(\log \frac{\sigma_1}{\sigma_0} + \frac{(x-\mu_1)^2}{2\sigma_1^2} - \frac{(x-\mu_0)^2}{2\sigma_0^2})dx \\
\int q\frac{(x-\mu_0)^2}{2\sigma_0^2}dx &= \frac{1}{2\sigma_0^2} \\
\int q\frac{(x-\mu_1)^2}{2\sigma_1^2}dx &= \frac{1}{2\sigma_1^2}(\int x^2 q dx + \int q\mu_1^2 - \int 2\mu_1 x q dx) \\
&= \frac{EX^2 + \mu_1^2 - 2\mu_1 EX}{2\sigma_1^2} \\
&= \frac{\sigma_0^2 + \mu_0^2 + \mu_1^2 - 2\mu_1\mu_0}{2\sigma_1^2} \\
&= \frac{\sigma_0^2 + (\mu_1 - \mu_0)^2}{2\sigma_1^2} \\
KL(q||p) &= \log \frac{\sigma_1}{\sigma_0} - \frac{1}{2\sigma_0^2} + \frac{\sigma_0^2 + (\mu_1 - \mu_0)^2}{2\sigma_1^2}
\end{aligned}$$

This section borrows a lot of its contents from these papers: [2] and [1].

References

[1] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,”

[2] C. Doersch, “Tutorial on variational autoencoders,”