WHAT ARE AUTOENCODERS?

Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible.
An autoencoder network is actually a pair of two connected networks, an encoder and a decoder. An encoder network takes in an input, and converts it into a smaller, dense representation, which the decoder network can use to convert it back to the original input.
Autoencoders map the data they are fed to a lower dimensional space by combining the data's most important features. They encode the original data into a more compact representation and decide how the data is combined, hence the auto in Autoencoder. These encoded features are often referred to as latent variables.
There are a few reasons doing this can be useful
1. Dimensionality reduction can decrease training time
2. Using latent feature representations can enhance model performance

INTITUTION

"Generative modeling" is a broad area of machine learning which deals with models of distributions P(X), defined over data points X in some potentially high-dimensional space X.
One straightforward kind of generative model simply allows us to compute P(X) numerically. In the case of images, X values which look like real images should get high probability, whereas images that look like random noise should get low probability. However, models like this are not necessarily useful: knowing that one image is unlikely does not help us synthesize one that is likely.
We can formalize this setup by saying that we get examples X distributed according to some unknown distribution Pgt(X), and our goal is to learn a model P which we can sample from, such that P is as similar as possible to Pgt.

Training this type of model has been a long-standing problem in the machine learning community

- They might require strong assumptions about the structure in the data.
- They might make severe approximations, leading to suboptimal models.
- They might rely on computationally expensive inference procedures like Markov Chain Monte Carlo.

The fundamental problem with autoencoders, for generation, is that the latent space they convert their inputs to and where their encoded vectors lie, may not be continuous, or allow easy interpolation.

Because if the space has discontinuities and you sample/generate a variation from there, the decoder will simply generate an unrealistic output, because the decoder has no idea how to deal with that region of the latent space. During training, it never saw encoded vectors coming from that region of latent space.

Variational Autoencoders (VAEs) have one fundamentally unique property that separates them from vanilla autoencoders, and it is this property that makes them so useful for generative modeling: their latent spaces are, by design, continuous, allowing easy random sampling and interpolation.

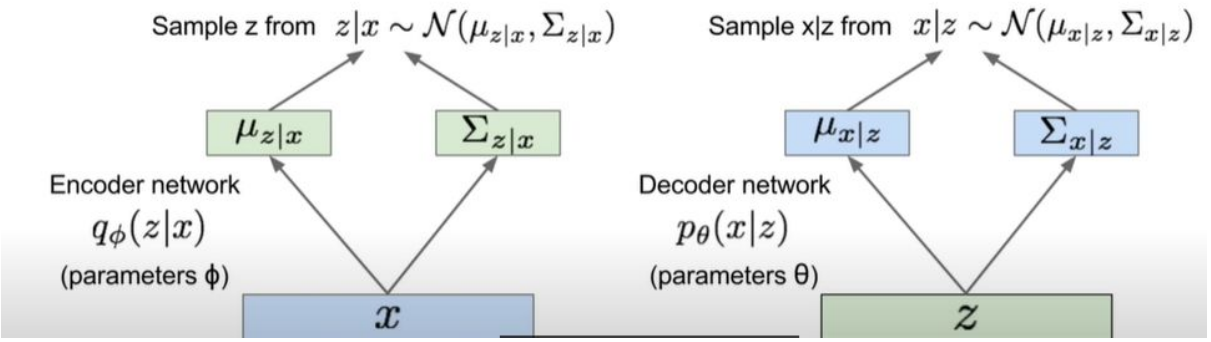
LEARNING IN LATENT VARIABLE MODEL

Our major goal is to find P(X) , but the major problem is to learn these **complicated** probability distribution P(X).So instead of modelling P(X) directly we introduced an unseen latent variable z and define a continuous distribution P(X|Z) known as likelihood. Z can be interpreted as a continuous random variable.
The latent variables z are drawn from a prior P(z)

The model defines a joint probability distribution over data and latent variables: p(x, z)

P(x,z)=p(x|z)p(z)

Unlike the normal autoencoders, the encoder of the VAE (called the recognition model) outputs a probability distribution for each latent attribute. For example, assume the distribution is a normal distribution. The output of the recognition model will be two vectors: one for the mean and the other for the standard deviation. The mean will control where the encoding of the input should be centered around while the standard deviation will control how much can the encoding vary from the mean. To decode, the values of each latent state is randomly sampled from the corresponding distribution and given as input to the decoder (called the generation model). The decoder then attempts to reconstruct the initial input to the network.



MATHS BEHIND VAE

Above section was able to provide you with some basic intuition of what VAEs do. This section will look at the mathematics running in the background.

VAEs map the input to a distribution p_θ, parametrized by θ. In order to generate a sample that looks like a real data point $x^{(i)}$, we follow these steps:

1. A value $z^{(i)}$ is generated from a prior distribution $p_\theta(x)$.
2. A value $x^{(i)}$ is generated from a conditional distribution $p_\theta(x|z)$.

Our major goal is to maximize the likelihood p(x) over θ. but let's look at the problem in optimizing this.

We can only see x, but we would like to infer the characteristics of z. In other words, we'd like to compute:

$$\mathrm{p}_\theta(\mathrm{z}|\mathrm{x}) = \frac{\mathrm{p}_\theta(\mathrm{x}|\mathrm{z})\mathrm{p}_\theta(\mathrm{z})}{\mathrm{p}_\theta(\mathrm{x})}$$

But, $\mathrm{p}_\theta(\mathrm{x}) = \int \mathrm{p}_\theta(\mathrm{x}|\mathrm{z})\mathrm{p}_\theta(\mathrm{z})\mathrm{dz}$ is intractable. As a result, it is not possible to infer $p_\theta(z|x)$ as well as we can't directly maximize this likelihood .

Solution to this problem:

In addition to decoder network modelling $p_\theta(x|z)$,define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(x|z)$. We will see that this allows us to derive a lower bound on the data likelihood that is trackable , which we can optimize.

The KL divergence is a measure of the similarity between two probability distributions. Thus, if we want to ensure that $q_\phi(z|x)$ is similar to $p_\theta(z|x)$, we could minimize the KL divergence between the two distributions.

$$\min(\mathrm{D}_{\mathrm{KL}}(\mathrm{q}_\phi(\mathrm{z}|\mathrm{x})||\mathrm{p}_\theta(\mathrm{z}|\mathrm{x})))$$

Now, let's expand the KL Divergence:

$$\mathcal{D}\left[Q(z)\|P(z|X)\right] = E_{z\sim Q}\left[\log Q(z) - \log P(z|X)\right].$$

We can get both P(X) and P(X|z) into this equation by applying Bayes rule to P(z|X):

$$\mathcal{D}\left[Q(z)\|P(z|X)\right] = E_{z\sim Q}\left[\log Q(z) - \log P(X|z) - \log P(z)\right] + \log P(X).$$

Here, log P(X) comes out of the expectation because it does not depend on z.

$$\log P(X) - \mathcal{D}\left[Q(z)\|P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z)\|P(z)\right].$$

The L.H.S. consists of the terms we want to maximize:

- The log-likelihood of generating real data: $\log P(X)$
- Negative of the difference between the real and estimated posterior distribution: the D[ ] term.

The R.H.S. is called the evidence lower bound (ELBO) as it is always <= $\log P(X)$. This is because the KL Divergence is always non-negative.
The loss function will be the negative of the ELBO (as we minimize the loss and ELBO is maximized).

$$\mathbf{Loss}(\theta,\phi) = -\mathbf{E}_{\mathbf{z}\sim\mathbf{q}_\phi(\mathbf{z}|\mathbf{x})}\log \mathbf{p}_\theta(\mathbf{x}|\mathbf{z}) + \mathbf{D}_{\mathbf{KL}}(\mathbf{q}_\phi(\mathbf{z}|\mathbf{x})||\mathbf{p}_\theta(\mathbf{z}))$$

The first term of this loss is reconstructed loss which encourages the likelihood and inference network to reconstruct data accurately
The second term is regularisation loss and penalizes posterior approximations that are too far from the prior. Both neural networks can be computed by gradient descent algorithms.
To use SGD, sample a value X and a value z, then compute the gradient of RHS by backpropagation. Do this for m times and take the average to get the result converging to the gradient of RHS.

we need to be a bit more specific about the form that Q(z|X) will take. The usual choice is to say that Q(z|X) = N (z|μ(X; ϑ), Σ(X; ϑ)), where μ and Σ are arbitrary deterministic functions with parameters ϑ that can be learned from data.

REPARAMETERIZED TRICK

During forward pass, you have an encoder that maps the input xx to a posterior distribution, typically assumed to be a normal distribution with mean $\mu(x)\mu(x)$ and standard deviation $\sigma(x)\sigma(x)$. Then, you sample a

point zz from the distribution $N(\mu(x),\sigma(x))N(\mu(x),\sigma(x))$, and pass zz through a decoder.

Now, when you have to backpropagate, the issue is that you cannot get a gradient of the sampled point w.r.t. $\mu(x)\mu(x)$ and $\sigma(x)\sigma(x)$, because the sampling operation cannot be differentiated. So, what you do is, you sample a point tt from $N(0,1)N(0,1)$, and construct zz as $\mu(x)+t\cdot\sigma(x)\mu(x)+t\cdot\sigma(x)$. Now, you can pretend that the sampled value tt is a constant, and differentiate zz to get gradients w.r.t. $\mu(x)\mu(x)$ and $\sigma(x)\sigma(x)$.

CONDITIONAL VARIATIONAL AUTOENCODERS

On VAE our objective is :

$$\log P(X) - D_{KL}[Q(z|X)\|P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X)\|P(z)]$$

that is, we want to optimize the log likelihood of our data P(X) under some “encoding” error. The original VAE model has two parts: the encoder Q(z|x)
And the decoder P(X|z)

Looking closely at the model, we could see why can’t VAE generate specific data, as per our example above. It’s because the encoder models the latent variable z directly based on X, it doesn’t care about the different type of X
. For example, it doesn’t take any account on the label of X.
Similarly, in the decoder part, it only models X directly based on the latent variable z
We could improve VAE by conditioning the encoder and decoder to another thing(s). Let’s say that other thing is c, so the encoder is now conditioned to two variables : X and c Q(z|X, c) .the same with decoder P(z|X, c)

Hence, our variational lower bound objective is now in this following form:

$$\log P(X|c) - D_{KL}[Q(z|X,c)\|P(z|X,c)] = E[\log P(X|z,c)] - D_{KL}[Q(z|X,c)\|P(z|c)]$$

i.e. we just conditioned all of the distributions with a variable c.