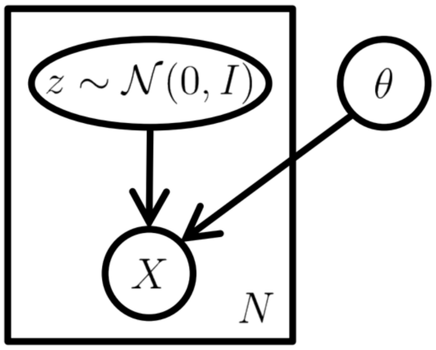
variational autoencoders

- we can think of a probability distribution as a joint distribution with a hidden variable $p(z, x)$. Inference is through the posterior, $p(z|x)$ which is $\frac{p(z,x)}{p(x)}$. The denominator is often intractable, which is why we resort to approximate methods to estimate the denominator. **variational inference** is a way to do this.
- variational inference turns inference into an optimization problem. we will come up with a set of variational parameters (maybe like a deep neural net) and then fit the variational parameters to be as close as possible (in KL) to the exact posterior, using optimization.
- variational inference came from methods in statistical physics.
- here we deal with probabilistic graphical models.
- generative modeling deals with models of distributions P(X), defined over datapoints in some high dimensional space. eg. images and the generative model's job is to capture the dependencies/joint distributions between the datapoints in each dimensions (pixels in an image).
- we can imagine at least two kinds of uses of generative models: one is to use them to compute the probaility/pdf of a given X. Another is a way to sample X based on likelihood -> to generate realistic images, for example.
- This has long been a problem of high interest/value. But past solutions have had one of these downfalls. First, they might require strong assumptions about the structure in the data. Second, they might make severe approximations, leading to suboptimal models. Or third, they might rely on computationally expensive inference procedures like Markov Chain Monte Carlo.
- Variational Autoencoders (VAEs) are high capacity, make little assumptions and have relatively low inference approximation error -> a good tradeoff of the above mentioned downsides.
- VAEs contain a bunch of latent variables (hidden variables). Informally, these latent variables learn to control what gets generated.
- Formally, say we have a vector of latent variables z in some high dimensional space that we can easily sample according to some probability distribution pdf P(z). Also say we have some deterministic funcion parametrized by some vector θ, where $f : z; \theta$ -> X. f is deterministic but z is random and θ is fixed so $f(z; \theta)$ is a random variable in the space of X. We wish to optimize θ such that with high probability, we can generate X's that will be like the X's in our dataset.
- Precisely, our goal is to make the probabilities of the X's in our dataset high according to $P(X) = \int P(X|z; \theta)P(z)dz$. The intuition behind this framework—called "maximum likelihood"— is that if the model is likely to produce training set samples, then it is also likely to produce similar samples, and unlikely to produce dissimilar ones.
- In VAEs the choice the of resulting output distribution is often multi-variate Gaussian with mean $f(z; \theta)$ and covariance equal to the identity matrix times a fixed constant, which is usually a hyper parameter. Note that the mean is what captures the complicated probability distribution the data (like how image pixels nearby are similar color and pixels are organized to look like objects). The covariance part just produces some simple model of variance.
- The probabilistic graphical model representing this VAE model is:



- Note that lack of any "encoder" pathway. We sample from the model without any input.
- The mathematical basis of VAEs actually has relatively little to do with classical autoencoders, e.g. sparse autoencoders or denoising autoencoders, both which have inputs, and explicit encoder decoder steps.
- They are called “autoencoders” only because the final training objective that derives from this setup does have an encoder and a decoder, and resembles a traditional autoencoder.
- Given a random variable z with a given distribution, we can create another random variable $X = g(z)$ with a completely different distribution. Look at how we can use sampled points from a gaussian distribution to create a ring like distribution, using a deterministic function g. This is the strategy that VAEs use. The deterministic function g is learned from the data.
- First, how do we choose the latent variables z such that we capture latent information? Returning to our digits example, the ‘latent’ decisions that the model needs to make before it begins painting the digit are actually rather complicated. It needs to choose not just the digit, but the angle that the digit is drawn, the stroke width, and also abstract stylistic properties. Worse, these properties may be correlated: a more angled digit may result if one writes faster, which also might tend to result in a thinner stroke. Ideally, we want to avoid deciding by hand what information each dimension of z encodes.
- VAEs assume that the samples of z (the latent variables) can be drawn from a simple multivariate normal with 0 mean and identity matrix as covariance matrix.
- The key is to notice that any distribution in d dimensions can be generated by taking a set of d variables that are normally distributed and mapping them through a sufficiently complicated function. proof idea: In one dimension, you can use the inverse cumulative distribution function (CDF) of the desired distribution composed with the CDF of a Gaussian. For multiple dimensions, do the stated process starting with the marginal distribution for a single dimension, and repeat with the conditional distribution of each additional dimension.
- Hence, provided powerful function approximators (we will use a neural net, surprise surprise!), we can simply learn a function which maps our independent, normally-distributed z values to whatever latent variables might be needed for the model, and then map those latent variables to X.
- In general, we don’t need to worry about ensuring that the latent structure exists. If such latent structure helps the model accurately reproduce (i.e. maximize the likelihood of) the training set, then the network will learn that structure at some layer. Here we are using the neural net as the function $f(z; \theta)$.
- The model we will learn is $P(X|z; \theta) = \mathcal{N}(X|f(x; \theta), \sigma^2 * I)$ where $P(z) = \mathcal{N}(0, I)$. As we have seen in neural nets, once we make up a loss function, the rest is just optimizing using SGD or your favorite optimization algorithm.

- Given a VAE and given an X, we can in theory get an approximation of $P(X)$ by sampling many many times and computing probability by sampling. The problem with this is that this method of estimation has extremely high variance in high dimensional spaces (like images) and is practically useless. For most z, $P(X|z)$ will be practically 0 and contribute nothing.
- Instead, given an X, we sample z values that are likely to have produced X (using say some $Q(z|X)$) and compute $P(X|z)$ from these, and from this then compute the marginal $P(X)$. This is the key idea behind variational auto-encoders.
- Now we want to train a model that maximizes $P(X)$ for the X's in the dataset we are training on.
- Note that we model Q as a gaussian with mean $\mu(X)$ (approximated by a neural net) and co-variance $\sigma(X)$ which is $k * I$, where k (a scalar) is also approximated by a neural net.
- From the definition of KL divergence:
  $\mathcal{D}[Q(z|X)||P(z|X)]= E_{z\ from\ Q(z|X)}[log(Q(z|X)) - log(P(z|X))]$
  On rearranging and simplifying we get,

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

- In two sentences, the left hand side has the quantity we want to maximize: log P(X) (plus an error term, which makes Q produce z's that can reproduce a given X; this term will become small if Q is high-capacity). The right hand side is something we can optimize via stochastic gradient descent given the right choice of Q → a neural net with re-parametrization to allow for differentiation over the stochastic normal function.
- The second term has a KL divergence between $Q(z|X)$ and the normal distribution since $P(z)$ is standard normal. This can be computed analytically in terms of $\mu(X)$ and $\sigma(X)$, which we can in turn backprop thru since it is a just a neural net as well.
- We want to optimize all X's in the dataset, so our equation to optimize becomes:

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

- We can estimate this by sampling a single value of X (from the dataset) and a single value of $z$and then compute the gradient of $log\ P(X|z) - \mathcal{D}[Q(z|X)||P(z)]$, and then average over a minibatch like we usually do in stochastic gradient descent.
- To produce new samples, we simply sample z from the standard normal and run the $P(X|z)$ step.
- One can view $\mathcal{D}[Q(z|X)||P(z)]$ (in the RHS above) as a regularization term that leads the parameters towards good latent representations.
- Note that because we maximize $log\ P(X) - \mathcal{D}[Q(z|X)||P(z|X)]$, we maximize a *lower bound* on the log likelihood.
- **The whole motivation for VAEs is that because the marginalization of the z we talked about earlier is intractable, we resort to the trick mentioned here to get a lower bound for $P(X)$ instead.**