Variational Auto-Encoders

16:198:520,536

Frequently it is useful to construct a map from some data space X to some encoding or embedding space Z in a way that the original data can be reconstructed from the encoded values. This is done in a variety of settings, but perhaps most fruitfully in machine translation, translating a sentence x to some embedding vector z in such a way that the original sentence can be reconstructed from z, potentially in a new language than the original sentence.

This kind of mapping is frequently discussed in terms of an encoder function $\Phi : X \mapsto Z$ and a decoder function $\Theta : Z \mapsto X$. We would like to find Φ, Θ that reconstructs the original data as faithfully as possible, or given a data set $x_1, x_2, \ldots, x_m$, solve

$$\min_{\Phi,\Theta} \sum_{i=1}^{m} ||\Theta(\Phi(x_i)) - x_i||^2, \tag{1}$$

to 'minimize the reconstruction cost'. When Φ and Θ are neural networks for instance, this is relatively straightforward to implement with the standard methods. Other loss functions may be equally or more applicable to use. This is the standard formulation of autoencoders, mapping from a data space to a latent space and back to a data space to reconstruct the data as faithfully as possible from the encoding.

One problem that you potentially run into here is a sort of collapse - that each data point x_i is mapped to a very specific z_i value, and the decoder is really really good at decoding those specific z_i values - but no other. As a result, any new x that wasn't in the data point may be mapped to a z that was never before seen, and that z fails to decode back to the original x. This is another instance of overfitting, where the model has captured the training data too perfectly and fails to generalize.

To avoid this, we would like to force the model to 'fill in' the latent space - that is, force it to be able to reasonably decode large swaths of z values in the latent space, rather than focusing on a few key points. To do this, instead of encoding a given x to a specific z, we can encode x *by choosing a z at random, according to a distribution defined by x*. This introduces a natural variation into the encoded z values, and forces the decoder to be able to handle this variation, and decode more of the latent variable space.

How can we factor this into the model? Let $q(\cdot|x)$ be a distribution for z defined by x - frequently this is taken to be a normal distribution, where the mean vector $\underline{\mu}(x)$ and standard deviations $\underline{\sigma}(x)$ are functions of x - potentially defined by neural networks that take x as input. Instead of trying to minimize the reconstruction cost, we try to minimize the *expected* reconstruction cost:

$$\min_{\underline{\mu},\underline{\sigma},\Theta} \sum_{i=1}^{m} \mathbb{E}_{q(\cdot|x_i)} \left[||\Theta(Z) - x_i||^2 \right]. \tag{2}$$

Note this introduces an expectation into the loss function that we may not be able to calculate easily - in practice, we can estimate this loss by sampling some number of Z for each x_i and estimating this expectation from the sample average. We can then compute gradients and update our models $(\underline{\mu}, \underline{\sigma}, \Theta)$ in the usual way.

The problem that we run into here is that something similar to collapse could happen again - if the $\underline{\sigma}$ converge to zero over time, then we lose the 'random sampling' effect that fills out the latent space; in this case, each x_i ends up corresponding to a unique z_i again. Similarly, if the means run far enough away from each other, we can get a similar collapse effect. In general, to keep the distribution q from collapsing in this way, we can add a regularizing term or cost term, that incurs a penalty if this kind of collapse happens. In particular, we can look at the KL-Divergence (a measure of how different two distributions are) and by comparing the q to a 'standard' distribution, we can keep q from collapsing too much.

The KL Divergence of two distributions p, q is generally defined to be

$$\mathrm{KL}(q||p) = \mathbb{E}_q \left[\ln \left(\frac{q(Z)}{p(Z)} \right) \right], \tag{3}$$

which equals zero if the distributions match, and increases the more 'different' they are. This leads to a 'standard' Variational Autoencoder formulation:

$$\min_{\underline{\mu},\underline{\sigma},\Theta} \sum_{i=1}^{m} \mathbb{E}_{q(\cdot|x_i)} \left[\text{ReconstructionCost}(\Theta(Z), x_i)\right] + \lambda \text{KL}(q(\cdot|x_i)||p_0(\cdot)). \tag{4}$$

Standard practice is to compare q to a 'standard normal' distribution p_0, centered at zero with a variance of 1 in each (independent) dimension. You can show that if q is defined in terms of normals as indicated above, this yields an overall KL divergence of (taking the dimension of the latent space to be L),

$$\text{KL}(q(\cdot|x_i)||p_0(\cdot)) = \frac{1}{2}||\underline{\mu}(x_i)||^2 + \frac{1}{2}\left[\sum_{j=1}^{L} \sigma_j^2(x_i) - \ln\left(\sigma_j^2(x_i)\right) - 1\right]. \tag{5}$$

This regularization term (at cost λ) keeps the mean and standard deviation of the 'conditional latent distribution' from collapsing or running away - as a result, the model is forced to 'pack' data into smaller regions of the latent variable space. What this means generally is that similar x will be 'packed' close to one another, producing similar z values, and picking two very distinct x values, if you 'decode' all the z along the path between them, you get smooth interpolation between the two x values. The latent space is therefore 'filled', and the encodings organized in such a way that the latent space 'makes sense' with the data.

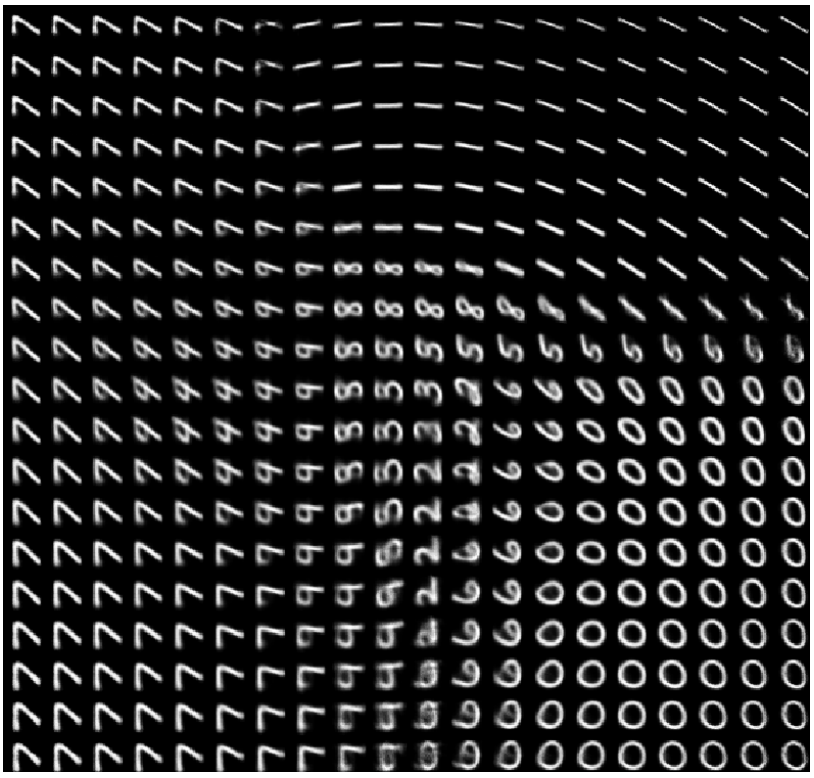

Figure 1: VAE Embedding of MNIST Handwritten Digits in 2 Dimensions

A More Formal Derivation

Suppose you wanted to describe the distribution of a data set, i.e., compute $P(x)$, the probability of x being sampled from this distribution. If P is parameterized, we might like to try to find the distribution that maximizes the

likelihood of the data we have (the usual MLE approach). If, for every x_i, we have a corresponding latent encoding / hidden variable z_i, this amounts to solving for some parameters W etc,

$$\max_W \prod_{i=1}^{m} P_W(x_i, z_i), \tag{6}$$

or in the usual way,

$$\max_W \sum_{i=1}^{m} \ln P_W(x_i, z_i). \tag{7}$$

The problem that we face here is of course that we don't know the latent encodings for each data point, so we have to marginalize, i.e., solve

$$\max_W \sum_{i=1}^{m} \ln P_W(x_i) = \max_W \sum_{i=1}^{m} \ln \left[\int_z P_W(x_i, z) dz \right]. \tag{8}$$

In trying to maximize this, we're faced with a similar situation to what we had for the EM algorithm - maximizing a log of a sum, because we don't know the hidden variable values. This is especially complicated if the marginalization is an integral instead of a sum (taking the latent space to be continuous rather than discrete).

So we do something similar here: instead of trying to maximize the log likelihood itself, we will try to find W to maximize a lower bound on the log likelihood. Observe:

$$\begin{aligned} L_i &= \ln P(x_i) \\ &= \ln \left[\int_z P(x_i, z) dz \right] \\ &= \ln \left[\int_z P(z) P(x_i|z) dz \right] \\ &\geq \int_z P(z) \ln \left[P(x_i|z) \right] dz \\ &= \mathbb{E} \left[\ln \left[P(x_i|Z) \right] \right]. \end{aligned} \tag{9}$$

At this point, if we made some assumptions about the marginal distribution of the latent encodings (the distribution of the encodings sort of over all possible inputs), we could try to maximize

$$\sum_{i=1}^{m} \mathbb{E} \left[\ln \left[P(x_i|Z) \right] \right], \tag{10}$$

approximating for instance this expectation by sampling a number of Z values according the the assumed distribution (usually normal). The problem with this approach, among other things, is that it focuses entirely on the 'decoder' part, the conditional distribution of x given the encoding z, and essentially ignores how that encoding came to be - where do we specify the encoder? It is sort of implicitly described, in the assumption that the conditional distribution of encodings is normal (in whatever dimension) but we can get some computational gains trying to relate this in

more directly - pulling in the contribution of $P(z|x_i)$, the encoding distribution. One possible way to do this is this:

$$\begin{aligned}
L_i &= \ln P(x_i) \\
&= \ln \left[\int_z P(x_i, z) dz \right] \\
&= \ln \left[\int_z P(z) P(x_i|z) dz \right] \\
&= \ln \left[\int_z P(z|x_i) \frac{P(z)}{P(z|x_i)} P(x_i|z) dz \right] \\
&\geq \int_z P(z|x_i) \ln \left[\frac{P(z)}{P(z|x_i)} P(x_i|z) \right] dz \\
&= \int_z P(z|x_i) \ln \left[P(x_i|z) \right] dz + \int_z P(z|x_i) \ln \left[\frac{P(z)}{P(z|x_i)} \right] dz \\
&= \mathbb{E}_{P(\cdot|x_i)} \left[\ln \left[P(x_i|Z) \right] \right] - \mathrm{KL}(P(\cdot|x_i)||P_Z(\cdot)),
\end{aligned} \tag{11}$$

where $\mathrm{KL}(P(\cdot|x_i)||P_Z(\cdot))$ measures the KL divergence between the encoding distribution and the assumed normal marginal distribution of the Z values. Hence, the problem we want to solve is to maximize

$$\sum_{i=1}^{m} \left[\mathbb{E}_{P(\cdot|x_i)} \left[\ln \left[P(x_i|Z) \right] \right] - \mathrm{KL}(P(\cdot|x_i)||P_Z(\cdot)) \right], \tag{12}$$

which we can flip around (multiplying by -1) to be minimizing

$$\text{reconstruction cost} + \text{KL divergence cost}, \tag{13}$$

as described in the previous section. This generalizes slightly, depending on the specifics of how the encoder and decoder distributions are constructed, but this gives a formal indication of where the KL divergence term comes from.