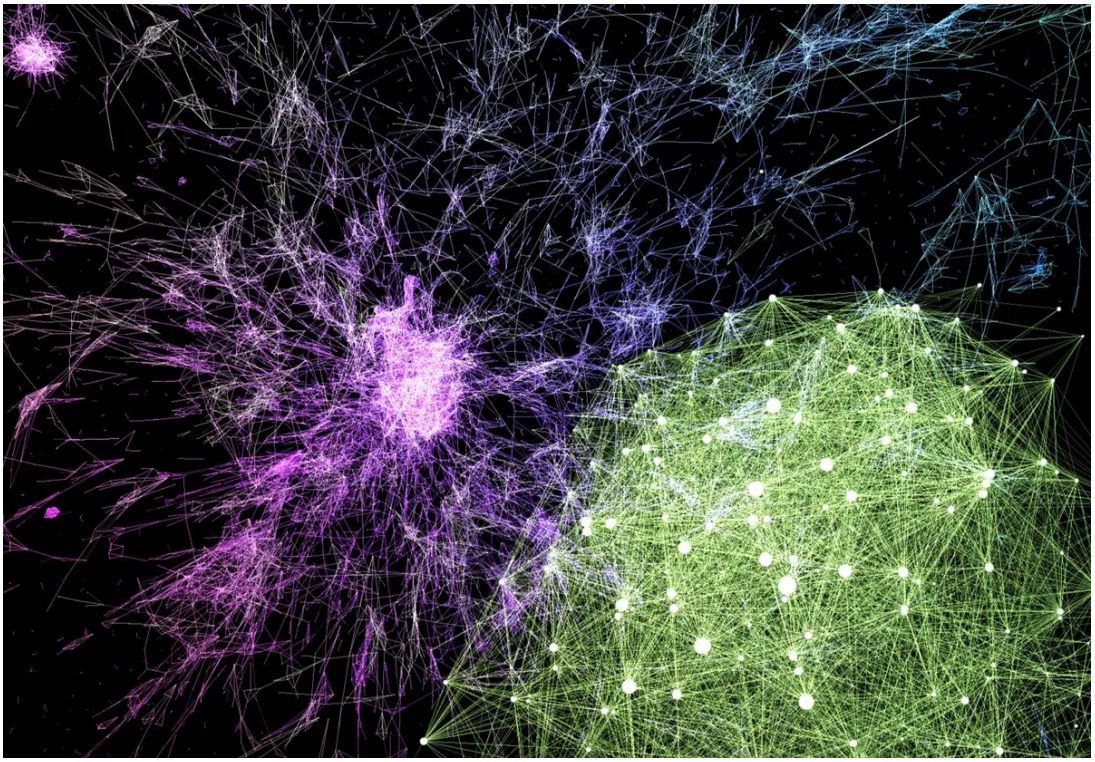
With Great Power Comes Poor Latent Codes: Representation Learning in VAEs (Pt. 2)

Cody Marie Wild · Follow
Published in Towards Data Science · 14 min read · May 7, 2018

(If you haven't done so yet, I recommend going back and reading Part 1 of this series on VAE failure modes; I spent more time there explaining the basics of generative models in general and VAEs in particular, and, since this post will pretty much jump right in where it left off, that one will provide useful background)

A machine learning idea I find particularly compelling is that of embeddings, representations, encodings: all of these vector spaces that can seem nigh-on magical when you zoom in and see the ways that a web of concepts can be beautifully mapped into mathematical space. I've spent enough time in the weeds of parameter optimization and vector algebra to know that calling any aspect of machine learning "magical" is starry-eyed even as I say it, but: approaches like this are unavoidably tantalizing because they offer the possibility of finding some optimal representation of concepts, a "optimal mathematical language" with which we can feed all of the world's information to our machines.



TIL: it's (un)surprisingly hard to find good visualizations of high-dimensional embedding spaces, especially if you don't want the old "king — queen" trick

That fuzzy dream, and an acute awareness of just how far we currently are from it, is what motivated me to write this pair of posts exploring the ways that one of our current best efforts at unsupervised representation learning can fail. The earlier post discussed how VAE representation can fail by embedding information in a hidden code in ways that are too dense, complex, and entangled for many of our needs. This post explores a solution motivated by a different dilemma: when we use a decoder with so much capacity that it chooses to not store information in the latent code at all, a result that leaves important information about our distribution locked up in decoder parameters, rather than neatly extracted as an internal representation.

If all you care about is building a generative model for the purposes of sampling new (artificial) observations, then this isn't an issue. However, if you wanted to be able to generate systematically different kinds of samples by modifying your z code, or if you wanted to use your encoder as a way of compressing input observations into useful information that another model could consume, then you have a problem.

Much Ado About No Information

Sometimes, when reading technical papers, you see a statement being made — in paper after paper— offhandedly, without explanation, as if it's too obvious to be worth explaining. You start to wonder if you've just developed a mental block, and there's some bit of glaringly obvious insight thateveryone else has that you've missed.

That's how I felt when I kept reading papers that talk about Variational Auto Encoders (VAEs) that manage to reconstruct their inputs, while storing no information in their latent codes about each individual input.

But, before we dive into **why** and **how** this happens, let's take a few steps back and walk through what the above statement actually means. As you (hopefully) read in my earlier post, VAEs are structured around an information bottleneck. The encoder takes in each observation X and calculates a compressed, lower-dimensional representation z, that is notionally supposed to capture high-level structure about this particular X. Then, a decoder takes in z as input and uses it to produce its best guess at the original input X. The decoder's reconstructed guess and the original X are compared to one another, and the pixelwise distance between the two — along with a regularization term that pushes each p(z|x) to be closer to a typically-Gaussian prior distribution — is used to update the parameters of the model. Conceptually, in such a model, information about the data distribution is stored in two places: the code z, and the weights of the network to transform z into the reconstructed X.

Top highlight

What we want, when we train a VAE for representation learning, is for z to represent high level concepts that describe what's present in this specific image, and the decoder parameters to learn generalized information on how to instantiate those concepts into actual pixel values. (As an aside: the same logic applies for any given observation you're reconstructing; I will tend to refer to image and pixels because almost all recent papers focus on images, and use that language),

When all of those papers I read alleged that the z distribution was uninformative, what they meant was: the network converges to a point where the z distribution that the encoder network produced was the same regardless of which X the encoder was given. If the z distribution doesn't differ as a function of specific X differing, then, by definition, it can't be

carrying any information about those specific inputs. Under this paradigm, when the decoder creates its reconstruction, it was essentially just sampling from the global data distribution, rather than a particular corner of the distribution informed by knowledge of X.

I can't speak for everyone, but it was really difficult for me to intuitively understand how this could happen. The whole notional structure of a VAE is as an autoencoder: it learns by calculating the pixel distance between a reconstructed and actual output. Even if we imagined that we had a way to perfectly sample from the real data distribution, it seems like there would be obvious value in using z to communicate some amount of information about the X we're trying to reconstruct (for example, that a scene was of a cat, rather than a tree). In order to build a better understanding, I had to take take two intellectual journeys; first through the **mechanics of autoregressive decoders**, and secondly through the often-thorny **math of the VAE loss itself**. Once we get out on the other end, we'll be better placed to understand the conceptual foundation underlying the — mechanically simple — solution proposed in Zhao et al's InfoVAE paper.

When Autoencoding Meets Autoregression

A simple, foundational equation in the theory of probability is the Chain Rule of Probability, which governs the decomposition of joint distributions into prior and conditional probability distributions. If you want to know p(x, y), which is to say, the probability of both x and y happening, which is to say, the value of the joint distribution P(X, Y) at the point (x, y), you can write it as:

$$p(x, y) = p(x)p(y|x)$$

Autoregressive generative models, of which PixelRNN and PixelCNN are the most well known, take this idea, and apply it to generation: instead of trying to generate each pixel independently (the typical VAE approach), or trying to generate every pixel as a conditional function of every other pixel (a computationally infeasible approach), what if you pretended that the pixels in images could be treated like a sequence, and generated pixels as the equation above would suggest: first select the pixel x1 based on the unconditional distribution over x1 pixel values, then x2 based on the

distribution conditioned on the x1 you chose, then x3 conditioned conditioned on both x1 and x2, etc. The idea behind PixelRNN is: in a RNN you inherently aggregate information about past generated pixels into the hidden state, and can use that to generate your next pixel, starting at the top left and moving down and right.

Though PixelRNN does a better job of aligning with the purist intuition of autoregressive models, the much more common autoregressive image model is a PixelCNN. This is due to two meaningful shortcomings of RNNs:

1. It can be difficult for them to remember global context (i.e. store information over long time windows)
2. You can't parallelize training of a RNN, because each pixel in the image needs to use the hidden state generated from creating the full image "before" where you currently are

Ahh, the smell of pragmatic-yet-effective hacks in the morning

In the great machine learning tradition of valuing practical trainability over airtight theory, the PixelCNN was born. In the PixelCNN, each pixel is generated by using a convolutions calculated using the pixels around it, but, importantly, **only the pixels that come "before" it in our arbitrary top-left-to-bottom-right ordering**. That's what that dark-greyed-out area is on the above image: a mask applied to a convolution to be sure that the pixel at location "e" isn't using any information "from the future" to condition itself. This is particularly important when we're generating images from scratch,

since by definition, if we generate from left to right, it will be impossible for a given pixel to condition its value on pixels further right and down that have not yet been generated.

PixelCNNs solve the two PixelRNN problems listed above, because:
1) As you add higher convolutional layers, each layer has a bigger "receptive field", i.e. it's higher on the pyramid of convolution, and thus has a base that is conditioned on a wider pixel range. This helps give the PixelCNN access to global structure information

2) Because each pixel is only conditioned on the pixels directly around it, and the training setup for this model is calculating loss by loss pixel values, this training is easily parallelizable, by sending different patches of a single image to different workers

Back in the land of VAEs, these autoregressive approaches started to look pretty appealing; historically, VAEs generate each pixel of the image independently (conditioned on a shared code), and this inherently leads to fuzzy reconstructions. This happens because, when all of the pixels are generated simultaneously, each pixel's best strategy is to generated its expected value conditioned on the shared code, without knowing what pixel values each other pixel chose. So, there started to be an interest in using autoregressive decoders in VAEs. This was done by appending z as an additional feature vector, alongside the convolution of nearby pixel values currently used by a PixelCNN. As expected, this led to sharper, better-detailed, reconstructions, because the pixels were better able to "coordinate" with each other.

Since I think best when I think in metaphors, the process of independent pixel generation is a bit like commissioning parts of a machine to be built by different manufacturing plants; since each plant doesn't know what the others are building, it's totally dependent on central direction for the parts to work together coherently. By contrast, I think of autoregressive generation as being a bit like that story game, where each person writes a sentence of a story, and then passes it to the next person, who writes a new sentence based on the last, and so on. There, instead of having central direction facilitate coordination between the parts of the whole, each part uses the context of the part before to make sure it is coordinated.

This starts to give us an inkling of why autoregressive decoders might lead to less information in the latent code: multi-pixel coordination that had previously been facilitated by the shared latent code could now be done be handled by giving your network visibility over a range of previously-generated pixels, and having it modulate its pixel output as a result.

Dealing With Loss (Functions)

The above is all well and good: I can understand fairly well how the more flexible parametrization of an autoregressive model gives it the ability to model complex data without using a latent code. But, what it took me longer to understand was: ostensibly, these models are autoencoders, and need to be able to successfully pixel-reconstruct their input to satisfy their loss function. **If the model isn't using z to communicate information about what kind of an image was given as input, how does is the decoder able to produce something that has low reconstruction loss with the input?**

To understand how the model's incentives line up this way, we should start with the equation that describe's the VAE's objective function.

The equation on the bottom is how the VAE function is typically characterized:

1. A term incentivizing p(x|z) to be high, which is to say, incentivizing the probability of the model generating the image you got as input, which is to say — if your output distributions are all Gaussian — the squared distance between your input and reconstructed pixels
2. A term incentivizing the distribution of encoded z|x to be close to the global prior, which is just a zero-mean Gaussian. (KL divergence measures how far these distributions are apart, and since this is a negative term in the an objective that is being maximized, the model is trying to make the magnitude of the KL Divergence smaller)

Under this framework, it's easy to think of the primary function of the model as being autoencoding, with the penalty term simply being a relatively unimportant regularizing term. But, the equation above can also be arranged as shown below:

It's not the most important that you exactly follow the math above; I'm mostly showing the derivation so that the second equation doesn't come out of thin air for you. In this formulation, the VAE's objective can also be seen as:

1. Increase p(x), which is to say, increase the model's probability on and ability to generate each observation in the data x
2. Decrease the KL divergence between the encoding distribution q(z|x) and the true underlying posterior distribution p(z|x)

What does this second bit of the equation imply? It can be a little hard to wrap your head around what exactly the posterior means in this context. It's a bit of an amorphous idea. But, one thing we do know about it is that:

Put into words, that means that the prior p(z) is a mixture of all of the conditional distributions, p(z|x), each weighted by how likely it's attendant x value is. The canonical form of a VAE is to use Gaussian distributions for all of it's conditional distributions q(z|x). In clearer, non-probability speak, that means that the encoder network maps from input values X, into the mean and variance of a Gaussian. We also typically set the prior to be a Gaussian: zero mean, and variance of 1. These two facts have an interesting implication, that is well visualized by this graph:

What this shows is: the only way that you can add multiple Gaussians together, and have their sum also be a Gaussian is if all of the Gaussians you are adding together have the same mean and variance parameters. What this means is: if you want to have a q(z|x) that perfectly perfectly matches p(z|x), either

1. Your q(z|x) needs to not be Gaussian, or
2. q(z|x) must be equivalent to p(z): the same uninformative unit Gaussian, regardless of the value of the input x.

If q(z|x) does anything other than the two options outlined above, it will fail to be identical to p(z|x), and will thus incur some cost through the KL divergence between q(z|x) and p(z|x). **Because we typically make the structural choice to only allow Gaussian p(z|x), that means that the only option available to the network, that allows it to incur zero loss from this second term, is to make the conditional distribution uninformative**.

Traversing back up the content stack, this means that the network will only choose to make its z value informative if doing so is necessary to model the full data distribution, p(x). Otherwise, the penalty it suffers for using an informative z will typically outweigh the individual-image accuracy benefit it gets from using it.

For Your Information

With all of this context in hand, we're now better placed to understand the solution that the InfoVAE paper proposed to this problem. The two main critiques leveled by the paper at the vanilla VAE objective were

1. The information preference property, which is what we outlined above:

the tendency of VAEs to prefer to not encode information in their latent code, if they can avoid it. This tends to happen when the regularization term is too strong.

2. The "exploding latent space" problem. This is essentially what I discussed in my earlier blog post, where, without sufficient regularization, the network has the incentive to make the conditional z distributions for each x non-overlapping, which generally leads to poor sampling ability and more entangled representations.

For all that the problem is a complex one to understand, the solution they suggest is actually remarkably simple. Remember how, in the original VAE equation, we penalize the KL divergence between the posterior over z, and the prior over z?

An important thing to remember here is that this is calculated **for each individual x**. That's because the coding layer is stochastic, and so for each input x, we don't simply produce one code, but instead produce the mean and standard deviation of a *distribution over codes*. It's that distribution, for each individual input, that is compared with the prior of a standard normal distribution.

Instead of this, the InfoVAE paper proposes a different regularization term: incentivizing the aggregated z distribution to be close to p(x), rather than pushing each individual z to be close. The aggregated z distribution is:

In words, that is basically just saying that, instead of taking the distribution defined by each individual input x, we should aggregate together the conditional distributions produced by all of the x values. The rationale for this is fairly coherent, if you've been following along so far. The information

preference property comes from the fact that, when you incentivize each individual conditional to be close to the prior, you are essentially incentivizing it to be uninformative. But, when you incentivize the aggregated z to be close to the prior, you allow more room for each individual z code to diverge from N(0, 1), and in doing so carry information about the specific X that produced it. However, because we still are pushing the aggregate distribution to be close to the prior, we disincentivize the network from falling into the "exploding latent space" problem, where it pushes its mean values to high magnitudes to create the non-overlapping distributions that would be most informative.

Note that, instead of being able to analytically calculate the difference between the parametrized conditional distribution and the prior, this new formulation requires some kind of sample-based way to measure a divergence between a set of samples and the prior. One example of this is an adversarial loss, where you take the aggregated set of z values that the encoder samples, and give those to a discriminator along with a set of z values drawn from the prior, and incentivize the model to make those two sets indistinguishable. The paper authors go into more such methods, but since they're fairly orthogonal to the thrust of this post, I'll leave you to explore those yourself if you read the paper.

Empirically, the authors found that this modification led to autoregressive VAEs making more use of the latent code, without a meaningful drop in reconstruction accuracy

Outstanding Questions

Source: http://kiriakakis.net/comics/mused/a-day-at-the-park

I know a lot more about representation learning then I did when I first conceived of this post series, and, if I've done by job right, so do you. But, finishing with a conclusion implies some pat set of answers, and being of the opinion that questions are often more interesting than answers, I'll leave you with some of those.

- If our goal is actually representation learning, does it actually matter if you have fuzzy reconstructions? This is basically asking: are sharp reconstructions, of the kind PixelCNN can give us, actually what we most care about?
- Could we get better representation learning if we combined some of the disentanglement techniques from the earlier post with an objective of reconstructing concept-level features (i.e. layers of the encoding network other than the raw pixels)

- How valuable is it to have stochastic codes in order to facilitate feature learning? If you use an “aggregate z” prior enforcing approach, like the ones outlined in InfoVAE, could that free us from using Gaussians for our latent codes in way that adds representational power?
- Could we also usefully employ adversarial loss on the reconstruction part of the network (that is: have a discriminator try to tell apart input and reconstruction), to get away from the over-focus on exact detail reconstruction that comes with pixel-wise loss

Sources:

- Variational Lossy Autoencoder
- Information Maximizing Variational Autoencoders
- Autoencoding a Single Bit

Machine Learning | Artificial Intelligence | Generative Model | Data Science | Towards Data Science

Written by Cody Marie Wild

Follow

2.8K Followers · Writer for Towards Data Science

machine learning engineer; lover of cats, languages, and elegant systems; professional curious person.