OneNote: Yan Liu -> VADA -> papers

Page 1

> VAEs have already shown promise in generating many kinds of complicated data, including handwritten digits [[1], 2], faces [[1], 3, 4], house numbers [5, 6], CIFAR images [6], physical models of scenes [4], segmentation [7], and predicting the future from static images [8]

> images are a popular kind of data for which we might create generative models

---

> generative model's job is to somehow capture the dependencies between pixels, e.g., that nearby pixels have similar color, and are organized into objects

---

Page 2

> 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

---

> three serious drawbacks

> strong assumptions

> severe approximations

> computationally expensive

Page 3

> Intuitively, it helps if the model first decides which character to generate before it assigns a value to any specific pixel. This kind of decision is formally called a latent variable

---

> z is called 'latent' because given just a character produced by the model, we don't necessarily know which settings of the latent variables generated the character

latent (not directly measured) variables are unobserved variables that are inferred from observed (directly measured variables)

So have some variables that are directly measurable and from them you can infer some other variable.

Wikipedia example is inferring quality of life from variables about a peron's life that are observable (income, car driven, etc.)

For generative models, it seems to be a bit reverse. Take his example of computer vision where you try to generate a 9. You cannot directly measure the 9 - unobservable. You can instead measure the pixels and from that you infer the 9. In a classification approach, you go from observed to hidden variable, pixels to number, but for a generative model, you go from hidden variable to observed variable. He uses the example of the number 9 being a latent variable in the task of generating a hand-written digit

---

> datapoint

e.g. image

> one (or many) settings of the latent variables which causes the model to generate something very similar to X

at least one "path" from latent variable to observed variable

---

> can easily sample according to some probability density function (PDF) P(z) defined over Z

this initially confused me because I was like, how are you going to get z off a function of z. but that's not it. This function defines the distribution of z. And you sample from this distribution. E.g. p(z) could be a gaussian, and so you sample from a gaussian

---

> optimize q such that we can sample z from P(z) and, with high probability, f (z; q) will be like the X's in our dataset

---

> aiming maximize the probability of each X in the training set

---

> law of total probability

law of total probability: if

- ({B_n: 1,2,3, ... }) is a set of pairwise disjoint events whose union is the entire sample space, and
- each even B_n is measurable then for any event A of the same probability space

P(A) = \sum P(A \cap B)

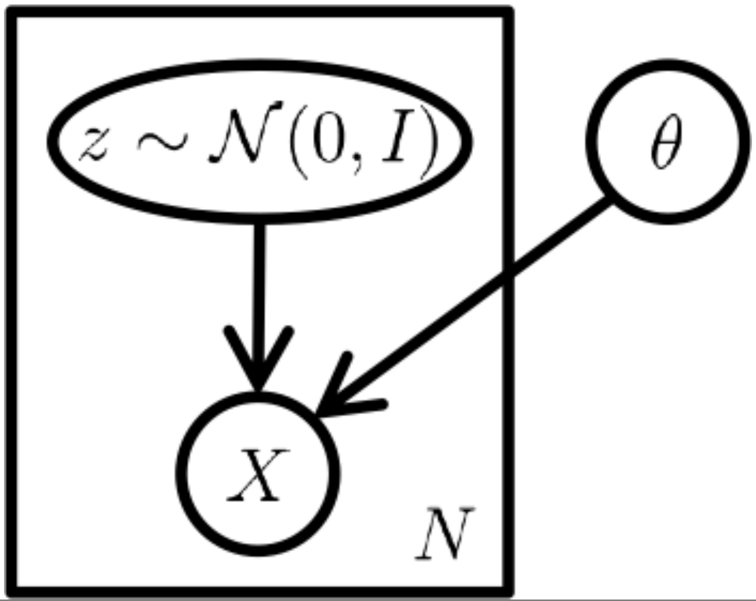
> maximum likelihood

> 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

> output distribution is often Gaussian

Page 4



standard VAE model

> By having a Gaussian distribution, we can use gradient descent (or any other optimization technique) to increase P(X) by making f (z; q) approach X for some z

---

> This wouldn’t be possible if P(XIz) was a Dirac delta function, as it would be if we used X = f (z; q) deterministically!

This tripped me up at first. If X=f(z;q) -> P(f(z;q)Iz) is a dirac delta with probability of one at value f(z;q) and 0 everywhere else. I keep forgeting that f(z;q): Z x Q -> X, i.e some vector from Z cross some vextor from Q mapped to some particular X value

---

> if X is binary, then P(XIz) might be a Bernoulli parameterized by f (z; q)

> From here onward, we will omit q from f (z; q) to avoid clutter.

Haha - this has confused me so much in the past

> VAEs actually has relatively little to do with classical autoencoders

---

> resembles a traditional autoencoder

Page 5

> This is the strategy that VAEs use to create arbitrary distributions: the deterministic function g is learned from data.

> two problems that VAEs must deal with: how to define the latent variables z (i.e., decide what information they represent), and how to deal with the integral over z

---

> how do we choose the latent variables z such that we capture latent information?

remember that the latent variable is not the variable itself but a representation of it (so e.g. some combination of features (line width, angle, et.c) which "represents" the number). There's some space of all possible features and it would really suck if you had to inspect this entire space or even if you had to hand craft each or some subset of the features in the space in order to generate representations for the latent variables z. so what do you do?

> VAEs take an unusual approach to dealing with this problem: they assume that there is no simple interpretation of the dimensions of z, and instead assert that samples of z can be drawn from a simple distribution, i.e., N (0, I)

Page 6

> 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

> 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

this sounds like quite the assumption..

Page 7

> conceptually straightforward to compute P(X) approximately

> in high dimensional spaces, n might need to be extremely large before we have an accurate estimate of P(X)

> isotropic

having a physical property that has the same value when measured in different directions.

> key idea behind the variational autoencoder is to attempt to sample values of z that are likely to have produced X, and compute P(X) just from those

---

> if z is sampled from an arbitrary distribution

> sample a large number of z values

Page 8

> with PDF Q(z), which is not N (0, I), then how does that help us optimize P(X)?

I don't quite understand what motivates this question...

> Kullback-Leibler divergence

The Kullback-Leibler divergence is the penalty you'll have to pay if you try to compress data from one distribution using a scheme optimised for another.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In ICML, 2014.

---

1. Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. ICLR, 2014.