6.867 Machine Learning Nov 12, 2019

Lecture 18: Deep Generative Models

Instructors: David Sontag (Lecturer), Devavrat Shah, Suvrit Sra *Scribe: Zhongxia Yan*

Note: the lecture notes have not been thoroughly checked for errors and are not at the level of publication.

1 Background

So far we've seen probabilistic models for a generative process with relatively simple latent factors, such as Latent Dirichlet Allocation (LDA). We have also already seen the use of deep autoregressive generative models for language modeling. In this lecture, we introduce the key ideas behind generative modeling, revisit autoregressive models and briefly introduce generative adversarial networks, and focus most attention on variational autoencoders.

2 Generative Modeling

Generative modeling is an unsupervised learning task. We are given finite samples of data vectors

$$\mathcal{D} = \{x \mid x \sim p_{\text{data}}(x)\}$$

$\mathcal{D}$ could be RGB images, spectrograms of music, etc. We would like to find a *generative* model with likelihood

$$p_{\text{model}}(x\,;\,\theta) \approx p_{\text{data}}(x)$$

To generate high dimensional data, generative models must model the underlying structure of the data. As Richard Feynman famously said, "What I cannot create, I do not understand."

Generative processes often describe the underlying generative process in a Bayesian network (directed graphical model). For example, consider the toy generative process below

$$\begin{aligned}
\pi(z) &= \mathcal{N}(0, \mathbb{I}_{d_z}) \\
\rho(\theta) &= \mathcal{N}(0, \kappa^2 \mathbb{I}_{d_\theta}) \\
h_1 &= \theta_0 + \theta_1 z \\
h_2 &= \exp(\theta_2) \\
p(x \mid z, \theta) &= \mathcal{N}(h_1, h_2)
\end{aligned}$$

Figure 1: A toy generative process. Figure from **?**

In this network, θ is the parameters of the model and z is the latent factor. In many generative processes, the latent factor(s) z is distributed according to some prior distribution, which represents the underlying variations in the world. For example, if we would like to generate a picture of a person, one latent factor that affects generation could be the height of the person, and height in the real world is distributed according to an approximately Gaussian distribution.

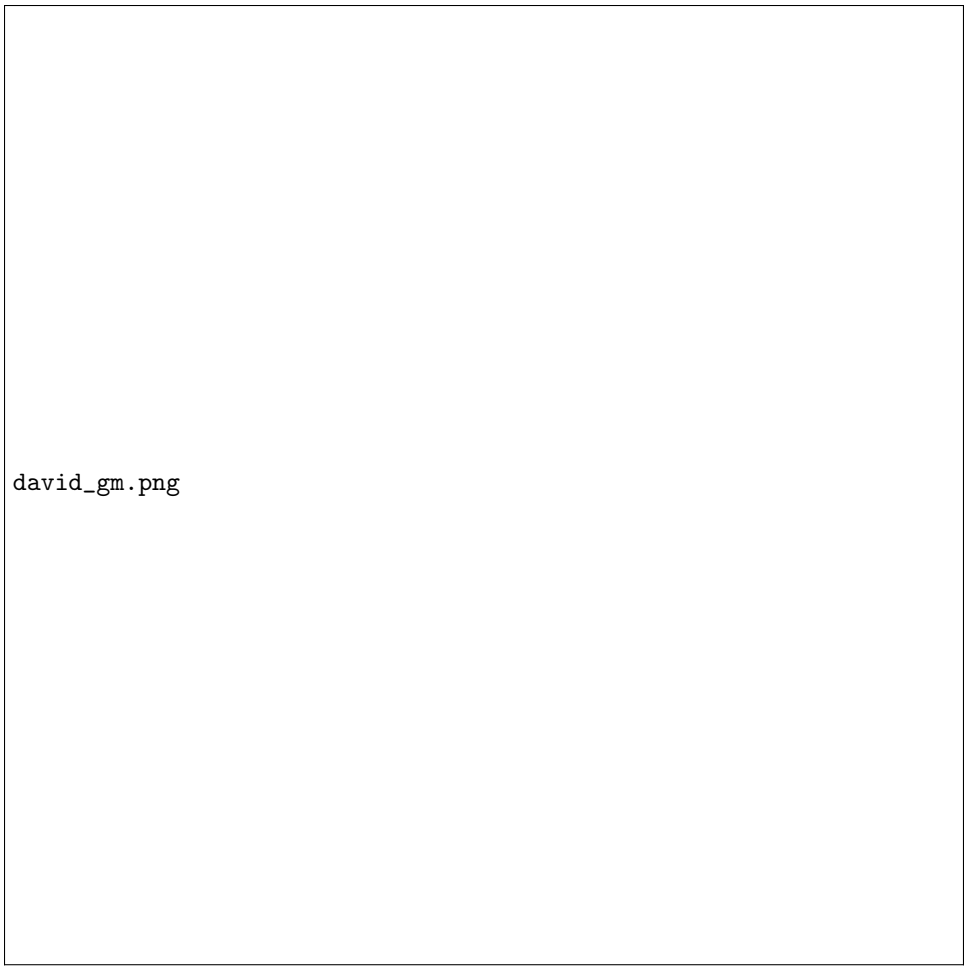

Figure 2: An example generative process

As seen in Figure **??**, the relationship between the generation x and the latent factors z may be complicated in general, and is often encoded through a neural network.

Generative modeling is useful in several ways

- A generative model may be able to generate new instances of $\mathcal{D}$. This could be artistic (i.e. if pictures or paintings are generated). This could also be used to generate additional data for downstream machine learning procedures, if the given data is insufficient.
- The effects of latent vectors z can be analyzed to see which latent factors are learned. This kind of analysis may allow us to assign interpretations to each factor in the vector z, and understand which factors contribute to variations in $\mathcal{D}$.

3 Taxonomy of Generative Models

To approximate $p_{\text{data}}(x)$, generative models maximize likelihood $p_{\text{model}}(x\,;\,\theta)$ for $x \in \mathcal{D}$. This is either done *explicitly*, i.e. define an explicit equation for $p_{\text{model}}(x\,;\,\theta)$ or *implicitly*, where the model directly learns to sample from $p_{\text{model}}(x\,;\,\theta)$ without defining it explicitly.

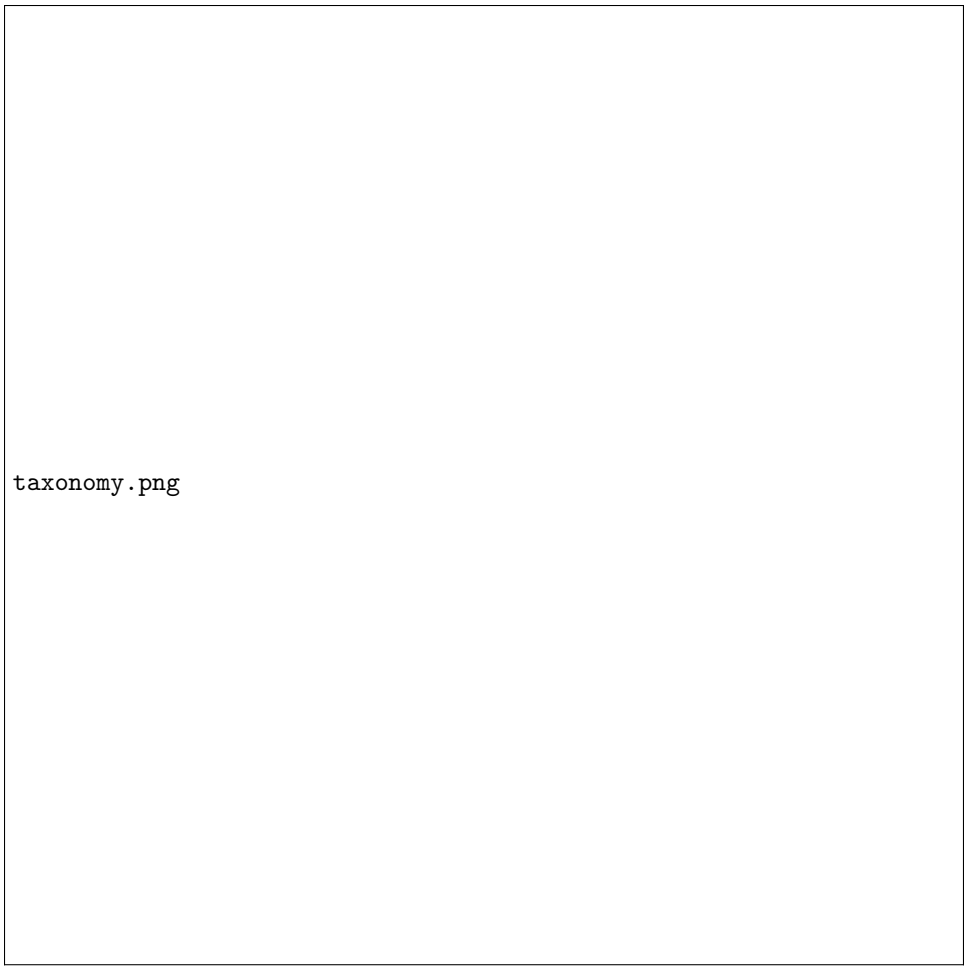

Figure 3: Taxonomy of generative models, copied from **?**

Here we will focus on the variational autoencoder, and briefly mention autoregressive model and generative adversarial network. In this course, we do not discuss Markov-chain based methods as well as other explicit, tractable density models, but you may refer to **?** for GSN, **?** for Boltzmann machine, **?** for NADE, and **?** for MADE.

4 Autoregressive Generative Model

Autoregressive models are generative models for sequential data. Since the data is sequential, the goal is to learn to model

$$p(x) = p(x_1, \cdots, x_N)$$

where $x_1, \cdots, x_N$ are tokens (e.g. words) in a sequence x (e.g. a sentence) of length N.

Autoregressive models factorize $p(x)$ with the chain rule, allowing for *explicit*, tractable calculation of $p(x)$

$$p(x_1, \cdots, x_N) = p(x_1) \prod_{i=2}^{N} p(x_i \mid x_1, \cdots, x_{i-1})$$

The model fits $p(x_i \mid x_1, \cdots, x_{i-1})$ for any i, which is sufficient to estimate $p(x)$.

Figure 4: Bayesian network of a sequence, figure from **?**

As seen in previous lectures, one common choice of neural network architecture for autoregressive modeling is the LSTM. Here the LSTM learns to model

$$p(x_i \mid x_1, \cdots, x_{i-1}) = p(x_i \mid s_i(s_{i-1}, x_i))$$

where s_i is the LSTM state after seeing x_i and s_{i-1}.

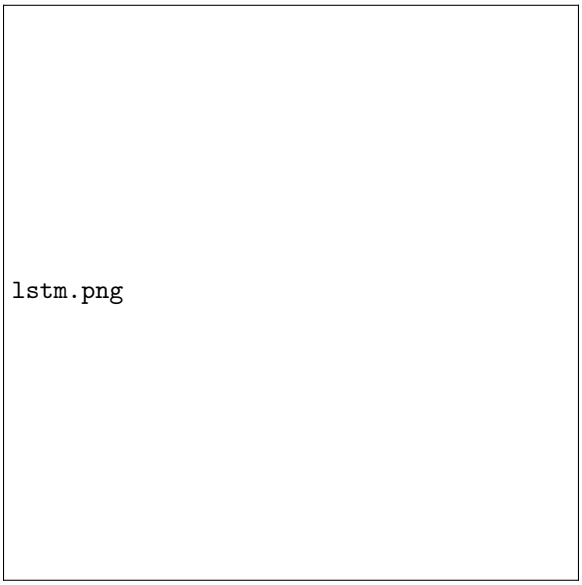

Figure 5: LSTM autoregressive model, figure from **?**

Since our model fits $p(x_i \mid x_1, \cdots, x_{i-1})$, we can easily sample $\hat{x}_i$ given $x_1, \cdots, x_{i-1}$. Thus to generate a sequence, we first sample $\hat{x}_1$ from some prior distribution, then sample $\hat{x}_2$ by feeding $\hat{x}_1$ into our model, then sample $\hat{x}_3$ by feeding $\hat{x}_1$ and $\hat{x}_2$ into our model, and so on. This is why this type of modeling is "autoregressive": the model receives its own previous prediction as an input for the next prediction.

5 Generative Adversarial Network (GANs)

Generative Adversarial Networks (GANs) are generative models involving two deep neural networks — the generator G and the discriminator D — with opposing optimization goals. GANs model the data distribution

implicitly and in general cannot compute $p_{\text{model}}(x)$. Here we discuss using GANs to model images, but GANs could be used to model arbitrary continuous (or roughly continuous, like pixel values) data.

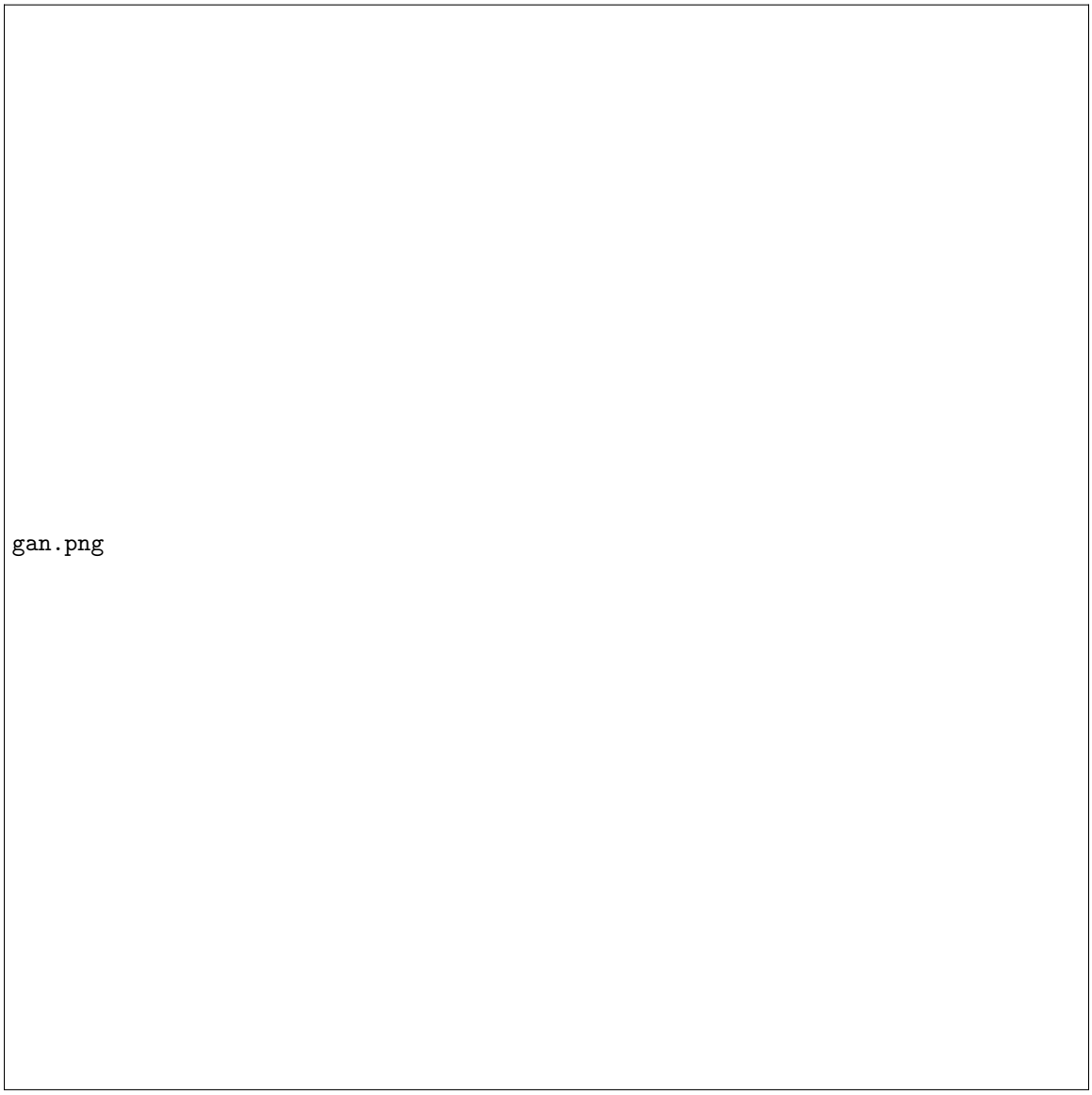

Figure 6: Components of a generative adversarial network

The generator takes in random noise z from some multivariate distribution (e.g. multivariate Gaussian) and outputs an image $G(z)$ with the same shape as the images in the training set. The discriminator takes in either an image x from the training set or an image $G(z)$ generated by the discriminator, and then classifies the image as 1 (real, from training set) or 0 (fake, generated by discriminator). The generator and discriminator are optimized alternatively.

The generator's objective is to generate as realistic of an image as possible, i.e. to maximize $D(G(z))$, which is the discriminator's evaluation on how likely the generated image is to belong in the training distribution.

$$\min_G \mathbb{E}_{z \sim p_{\text{noise}}(z)} \left[\log(1 - D(G(z)))\right]$$

The discriminator's objective is to correctly evaluate whether an image is from the training distribution or the generator's output, i.e. to maximize $D(x)$ for x from the training distribution and to minimize $D(G(z))$ for $G(z)$ from the generator output. According to **?**, a good discriminator should be able to identify differences between the low-/high-order moments of the training distribution and the generated output.

$$\max_D \left[\mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim p_{\text{noise}}(z)}\left[\log(1 - D(G(z)))\right]\right]$$

GANs are notoriously difficult to optimize, but in theory the optimal generative model is one that perfectly generates the training distribution given $z \sim p(z)$. Notice that in this case, the discriminator would satisfy $D(x) = D(G(z)) = \frac{1}{2}$, since the discriminator wouldn't be able to distinguish x from the training distribution and $G(z)$ from the generator output.

GANs have been used to generate amazingly realistic images. We will not go more in detail about GANs here, and you can refer to **?** for further reading if curious.

6 Variational Autoencoders (VAEs)

Variational autoencoders *explicitly* model the lower bound for the likelihood $p_{\text{model}}(x)$, rather than modeling the exact likelihood. VAEs innovate on concepts from autoencoder and variational learning.

6.1 Autoencoders

Let $x_1, \ldots, x_N \in \mathbb{R}^d$ be some data type (e.g. images) and $z \in \mathbb{R}^m$ be a much smaller latent vector space ($m << d$). In general, autoencoders consists of a pair of *encoder* and *decoder* networks. The encoder network $\hat{z} = E(x)$ projects the data into latent space $\mathbb{R}^k$ while the decoder network $\hat{x} = D(\hat{z})$ projects latent vector $\hat{z} = E(x)$ back into data space $\mathbb{R}^k$. The goal for the encoder and decoder pair is to minimize reconstruction error

$$\sum_{i=1}^{N} \|\hat{x_i} - x_i\|^2 = \sum_{i=1}^{N} \|D(E(x_i)) - x_i\|^2$$

Figure 7: An autoencoder, figure from **?**

For example, principle component analysis (PCA) as seen previously can be considered as a linear autoencoder. PCA finds matrix U with shape $d \times m$ which minimizes projection error between x and $\hat{x} = UU^T x$. Here $\hat{z} = E(x) = U^T x$ and $\hat{x} = D(E(x)) = UU^T x$.

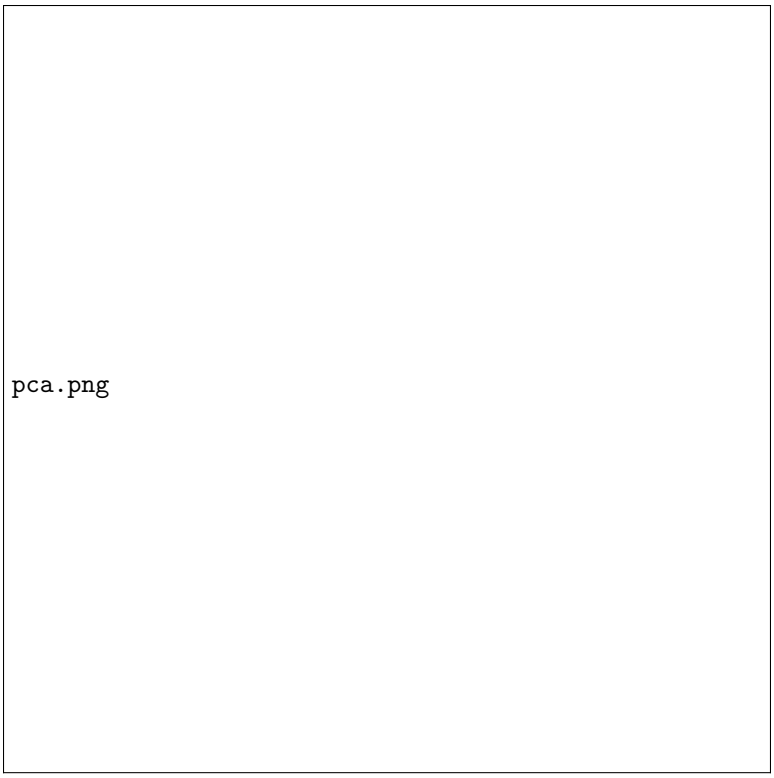

Figure 8: PCA as an autoencoder

$m << d$ means that autoencoder perform dimensionality reduction on the data. $\hat{z} = E(x)$ captures meaningful factors of variation in x.

6.2 Variational Learning

Recall that for a generative model we are provided with data $\mathcal{D} = \{x_1, \ldots, x_N\}$, and we would like to maximize the log-likelihood of the data

$$\mathcal{L}(\mathcal{D}\,;\,\theta) = \sum_{x_i \in \mathcal{D}} \log p_\theta(x_i)$$

Variational learning assumes that the generative process behind the underlying data is factorized into first generating z_i then x_i

1. $z_i \sim p(z)$. This states that factors in the latent vector are distributed according to some prior distribution (e.g. Gaussian). For example, height may be a latent factor with Gaussian distribution.
2. $x_i \sim p(x \mid z)$. This states that, given z_i, each x_i is sampled from a conditional distribution. For example, if we are trying to generate an image x_i of a person and we are given the height, there is a distribution of possible candidates to generate, given the height.

Figure 9: Bayesian network for a simple generative process

We model the above factors with $p_\theta(z)$ and $p_\theta(x \mid z)$, where θ is the parameters for the *generation network*. We lower bound the likelihood for each data point

$$\log p_\theta(x_i) \geq \mathcal{L}(x_i\,;\,\theta, q_i) = \mathbb{E}_{q_i(z)}\left[\log p_\theta(x_i, z)\right] + H(q_i(z))$$

where the right-hand side is the evidence lower bound (ELBO) and $q_i(z)$ is an approximation for $p(z \mid x_i)$. We then maximize the lower bound

$$\max_{\theta, q_1, \ldots, q_N} \sum_{i=1}^{N} \mathcal{L}(x_i\,;\,\theta, q_i)$$

which brings $q_i(z) \to p(z \mid x_i)$.

6.3 Recognition Network

Variational autoencoder combines applies probabilistic assumptions from variational learning to the autoencoder structure. The *generation network* from variational learning assumes the role of the *decoder network*. As explained below, VAE introduces a *recognition network* (also called *inference network*) to assume the role of the *encoder network*.

Recall that in variational learning, we optimize a q_i for each x_i in order to maximize the ELBO. On the other hand, VAE replaces $q_i(z)$ with the *recognition network* output $q_\phi(z \mid x_i)$, parameterized by ϕ. The learning objective becomes

$$\max_{\theta, \phi} \sum_{i=1}^{N} \mathcal{L}(x_i\,;\,\theta, \phi)$$

$$\mathcal{L}(x_i\,;\,\theta, \phi) = \mathbb{E}_{q_\phi(z \mid x_i)}\left[\log p_\theta(x_i, z)\right] + H(q_\phi(z \mid x_i))$$

The corresponding plate model is Figure **??**. The solid arrows denote the *generation* process $p_\theta(z)$ and $p_\theta(x \mid z)$, while the dotted arrows denote the *recognition* process $q_\phi(z \mid x)$.

Figure 10: Bayesian network for a VAE

While variational learning has $O(N)$ parameters for $q_1, \ldots, q_N$, the variational autoencoder has a *recognition network* whose parameters ϕ are shared across all the data points and thus $O(1)$.

We show here that using a logistic regression with linear features as $q_\phi(z \mid x)$ is sufficient to exactly represent a Naive Bayes generative process with Bernoulli $p(x_i \mid z)$. We stipulate that $x \in \{0,1\}^d$ and $z \in \{0,1\}$. Suppose that the generative process is

1. Sample a binary latent $z \sim p(z)$
2. We choose $x \sim p(x \mid z)$. In particular we assume conditional independence (hence Naive Bayes) and assume $p(x \mid z) = \prod_{i=1}^{d} p(x_i \mid z)$, where $p(x_i \mid z)$ is a Bernoulli distribution

$$p(x_i \mid z = 0) = \alpha^{x_i}(1-\alpha)^{1-x_i}$$

$$p(x_i \mid z = 1) = \beta^{x_i}(1-\beta)^{1-x_i}$$

Given this formulation of a generative process, we can show that $p(z \mid x)$ is in the form of a linear feature logistic regression.

$$\begin{aligned}
p(z=0 \mid x) &= \frac{p(x, z=0)}{p(x)} \\
&= \frac{p(x, z=0)}{p(x, z=0) + p(x, z=1)} \\
&= \frac{1}{1 + \frac{p(x,z=1)}{p(x,z=0)}} \\
&= \frac{1}{1 + \exp\left(-\ln \frac{p(x,z=0)}{p(x,z=1)}\right)} \\
&= \sigma\left(\ln \frac{p(x, z=0)}{p(x, z=1)}\right) \\
&= \sigma\left(\ln \frac{p(z=0)\prod_{i=1}^{d} p(x_i \mid z=0)}{p(z=1)\prod_{i=1}^{d} p(x_i \mid z=1)}\right) \\
&= \sigma\left(\ln \frac{p(z=0)}{p(z=1)} + \sum_{i=1}^{d} \ln \frac{p(x_i \mid z=0)}{p(x_i \mid z=1)}\right) \\
&= \sigma\left(\ln \frac{p(z=0)}{p(z=1)} + \sum_{i=1}^{d} \ln \frac{\alpha^{x_i}(1-\alpha)^{1-x_i}}{\beta^{x_i}(1-\beta)^{1-x_i}}\right) \\
&= \sigma\left(\ln \frac{p(z=0)}{p(z=1)} + \sum_{i=1}^{d} \left[x_i \ln \frac{\alpha}{\beta} + (1-x_i) \ln \frac{1-\alpha}{1-\beta}\right]\right) \\
&= \sigma\left(d + \ln \frac{p(z=0)}{p(z=1)} + \sum_{i=1}^{d} x_i \left[\ln \frac{\alpha}{\beta} - \ln \frac{1-\alpha}{1-\beta}\right]\right) \\
&= \sigma\left(\left(d + \ln \frac{p(z=0)}{p(z=1)}\right) + \sum_{i=1}^{d} x_i \ln \frac{\alpha(1-\beta)}{\beta(1-\alpha)}\right) \\
&= \sigma(b + \sum_{i=1}^{d} x_i w_i) \\
&= \sigma(b + w^T x)
\end{aligned}$$

Therefore, we can exactly represent $p(z \mid x)$ with a linear-feature logistic regression $q_\phi(z \mid x)$. We next discuss how to compute the ELBO and its gradient in order to maximize the ELBO.

6.4 Algorithm

We are ready to state the algorithm for training a variational autoencoder for a dataset $\mathcal{D}$ with doubly stochastic gradient ascent.

On input dataset $\mathcal{D}$:

1. Sample $x \sim \mathcal{D}$
2. Sample $z_1, \ldots, z_K \sim q_\phi(z \mid x)$
3. Estimate ELBO

$\mathbb{E}_{q_\phi(z|x_i)}[\log p_\theta(x_i, z)] \approx \frac{1}{K}\sum_{k=1}^{K} \log p_\theta(x_i, z_k)$
$H(q_\phi(z \mid x_i))$ can often be calculated analytically

4. Estimate gradients. We discuss how to estimate $\nabla_\theta \mathcal{L}(x\,;\theta,\phi)$ and $\nabla_\phi \mathcal{L}(x\,;\theta,\phi)$ below
5. Update θ and ϕ with stochastic gradient ascent
   $\theta \leftarrow \theta + \eta_\theta \nabla_\theta \mathcal{L}(x\,;\theta,\phi)$
   $\phi \leftarrow \phi + \eta_\phi \nabla_\phi \mathcal{L}(x\,;\theta,\phi)$
6. Go to step 1

6.4.1 Estimating Gradients

We would like to compute $\nabla_\theta \mathcal{L}(x\,;\theta,\phi)$ and $\nabla_\phi \mathcal{L}(x\,;\theta,\phi)$ so we can perform gradient ascent.

$$\begin{aligned}
\nabla_\theta \mathcal{L}(x\,;\theta,\phi) &= \nabla_\theta \left[\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x,z)] + H(q_\phi(z \mid x))\right] \\
&= \nabla_\theta \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x,z)] \text{ since second term has no } \theta \\
&= \mathbb{E}_{q_\phi(z|x)}[\nabla_\theta \log p_\theta(x,z)] \text{ since expectation does not involve } \theta \\
&\approx \frac{1}{K}\sum_{k=1}^{K} \nabla_\theta \log p_\theta(x, z_k)
\end{aligned}$$

Thus we can calculate an estimation for $\nabla_\theta \mathcal{L}(x\,;\theta,\phi)$ with a sample $x_i \sim \mathcal{D}$ and samples $z_1, \ldots, z_K \sim q_\phi(z \mid x_i)$.

We derive a trick that we will use shortly for calculating $\nabla_\phi \mathcal{L}(x_i\,;\theta,\phi)$

$$\begin{aligned}
\nabla_\phi \log(q_\phi(z \mid x)) &= \frac{1}{q_\phi(z \mid x)} \nabla_\phi q_\phi(z \mid x) \\
\nabla_\phi q_\phi(z \mid x) &= q_\phi(z \mid x) \nabla_\phi \log q_\phi(z \mid x) \text{ after rearranging}
\end{aligned}$$

We define $g(z,\phi) = \log p_\theta(x,z) - \log q_\phi(z \mid x)$

We start from an equivalent equation for $\mathcal{L}$

$$\begin{aligned}
\mathcal{L}(x\,;\theta,\phi) &= \mathbb{E}_{q_\phi(z|x)}\left[\frac{\log p_\theta(x,z)}{q_\phi(z \mid x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}[g(z,\phi)] \\
\nabla_\phi \mathcal{L}(x\,;\theta,\phi) &= \nabla_\phi \mathbb{E}_{q_\phi(z|x)}[g(z,\phi)] \\
&= \nabla_\phi \int q_\phi(z \mid x) g(z,\phi) dz \\
&= \int \nabla_\phi \left[q_\phi(z \mid x) g(z,\phi)\right] dz \\
&= \int g(z,\phi) \nabla_\phi q_\phi(z \mid x) + q_\phi(z \mid x) \nabla_\phi g(z,\phi) dz \\
&= \int g(z,\phi) q_\phi(z \mid x) \nabla_\phi \log q_\phi(z \mid x) + q_\phi(z \mid x) \nabla_\phi g(z,\phi) dz \text{ using above trick} \\
&= \mathbb{E}_{q_\phi(z|x)}[g(z,\phi) \nabla_\phi \log q_\phi(z \mid x) + \nabla_\phi g(z,\phi)] \\
&= \mathbb{E}_{q_\phi(z|x)}[g(z,\phi) \nabla_\phi \log q_\phi(z \mid x)] + \mathbb{E}_{q_\phi(z|x)}[\nabla_\phi g(z,\phi)]
\end{aligned}$$

We simplify the second term

$$\begin{aligned}
\mathbb{E}_{q_\phi(z|x)}\left[\nabla_\phi g(z,\phi)\right] &= \mathbb{E}_{q_\phi(z|x)}\left[\nabla_\phi\left(\log p_\theta(x,z) - \log q_\phi(z \mid x)\right)\right] \\
&= -\mathbb{E}_{q_\phi(z|x)}\left[\nabla_\phi \log q_\phi(z \mid x)\right] \text{ since first term doesn't depend on } \phi \\
&= -\mathbb{E}_{q_\phi(z|x)}\left[\frac{1}{q_\phi(z \mid x)}\nabla_\phi q_\phi(z \mid x)\right] \text{ using above trick} \\
&= -\int \nabla_\phi q_\phi(z \mid x) dz \\
&= -\nabla_\phi \int q_\phi(z \mid x) dz \\
&= -\nabla_\phi 1 \text{ since } q \text{ is a normalized distribution} \\
&= 0
\end{aligned}$$

Thus we have

$$\begin{aligned}
\nabla_\phi \mathcal{L}(x\,;\theta,\phi) &= \mathbb{E}_{q_\phi(z|x)}\left[g(z,\phi)\nabla_\phi \log q_\phi(z \mid x)\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\left(\log p_\theta(x,z) - \log q_\phi(z \mid x)\right)\nabla_\phi \log q_\phi(z \mid x)\right] \\
&\approx \frac{1}{K}\sum_{k=1}^{K}\left(\log p_\theta(x,z_k) - \log q_\phi(z_k \mid x)\right)\nabla_\phi \log q_\phi(z_k \mid x)
\end{aligned}$$

Thus we can calculate an estimation for $\nabla_\phi \mathcal{L}(x\,;\theta,\phi)$ with a sample $x_i \sim \mathcal{D}$ and samples $z_1,\ldots,z_K \sim q_\phi(z \mid x_i)$.

The gradient $\nabla_\phi \mathcal{L}$ is known as the *likelihood ratio* or *REINFORCE gradient*, while this method also has the name of *score-function estimator*.

- Advantage: fully general and applies to any diffentiable density involving continuous or discrete random variables
- Advantage: unbiased estimator for $\nabla_\phi \mathcal{L}$
- Disadvantage: can have high variance

6.4.2 Pathwise Gradient Estimators (Reparameterization Trick)

The *pathwise gradient estimator*, also known as *reparameterization trick*, offers a low-variance alternative to the score-function estimator for $\nabla_\phi \mathcal{L}$ for certain distributions (e.g. Gaussian).

Notice how earlier we could not move the ∇_ϕ into the expectation

$$\nabla_\phi \mathcal{L}(x\,;\theta,\phi) = \nabla_\phi \mathbb{E}_{q_\phi(z|x)}\left[g(z,\phi)\right]$$

because the expectation operator $\mathbb{E}_{q_\phi(z|x)}$ is over $q_\phi(z \mid x)$.

Instead of taking the expectation over $z \sim q_\phi(z \mid x)$, we may be able to replace this with an expectation over $\epsilon \sim p(\epsilon)$ if we can find differentiable h_ϕ such that $z = h_\phi(x,\epsilon)$. Essentially, this isolates the randomness

in the $q_\phi(z \mid x)$ distribution into a $p(\epsilon)$ distribution which does not depend on ϕ. Then we can substitute

$$\begin{aligned}
\mathbb{E}_{z\sim q_\phi(z|x)}\left[g(z,\phi)\right] &= \mathbb{E}_{\epsilon\sim p(\epsilon)}\left[g(h(x,\epsilon),\phi)\right] \\
\nabla_\phi \mathbb{E}_{z\sim q_\phi(z|x)}\left[g(z,\phi)\right] &= \nabla_\phi \mathbb{E}_{\epsilon\sim p(\epsilon)}\left[g(h(x,\epsilon),\phi)\right] \\
&= \mathbb{E}_{\epsilon\sim p(\epsilon)}\left[\nabla_\phi g(h(x,\epsilon),\phi)\right] \\
&= \mathbb{E}_{\epsilon\sim p(\epsilon)}\left[\nabla_\phi \left(\log p_\theta(x, h(x,\epsilon)) - \log q_\phi(h(x,\epsilon) \mid x)\right)\right] \\
&= -\mathbb{E}_{\epsilon\sim p(\epsilon)}\left[\nabla_\phi \log q_\phi(h(x,\epsilon) \mid x)\right] \text{ the first term does not depend on } \phi \\
&\approx \frac{1}{K}\sum_{k=1}^{K} \nabla_\phi \log q_\phi(h(x,\epsilon_k) \mid x)
\end{aligned}$$

Thus the $\nabla_\phi \mathcal{L}$ can easily be approximated over a sample x_i and samples $\epsilon_1, \ldots, \epsilon_K$.

- Advantage: lower variance than score-function estimator
- Disadvantage: does not work for all distributions

6.5 Application: Latent Gaussian Models

In this setup, we stipulate a generative process for $\mathcal{D} = \{x_1, \ldots, x_N\}$

- $z \sim \mathcal{N}(0, I)$ is a sample from the prior latent vector distribution
- $x \sim \mathcal{N}(f_\theta(z), I)$ is a sample from the conditional distribution of the generative network

We define a Gaussian recognition network $q(z \mid x)$

- $z \sim \mathcal{N}(f_\phi(x), I)$ is a sample from the posterior latent vector distribution

We can run the VAE algorithm stated above on example.

1. Sample $x \sim \mathcal{D}$
2. Sample $z_1, \ldots, z_k \sim \mathcal{N}(f_\phi(x), I)$
3. Estimate ELBO
   $\mathbb{E}_{q_\phi(z|x_i)}\left[\log p_\theta(x_i, z)\right] \approx \frac{1}{K}\sum_{k=1}^{K} \log p_\theta(z_k) p_\theta(x \mid z_k)$. Both terms can be evaluated easily
   $H(q_\phi(z \mid x_i))$ sole depends on the covariance, which is constant and can be derived
4. Estimate gradients.
   $\nabla_\theta \mathcal{L}(x\,;\theta,\phi) \approx \frac{1}{K}\sum_{k=1}^{K} \nabla_\theta \log p_\theta(x, z_k)$. This can be calculated through backpropagation
   $\nabla_\phi \mathcal{L}(x\,;\theta,\phi) \approx \frac{1}{K}\sum_{k=1}^{K} \nabla_\phi \log p(\epsilon_k)$. This can easily be calculated, see below for derivation
5. Update θ and ϕ with stochastic gradient ascent
   $\theta \leftarrow \theta + \eta_\theta \nabla_\theta \mathcal{L}(x\,;\theta,\phi)$
   $\phi \leftarrow \phi + \eta_\phi \nabla_\phi \mathcal{L}(x\,;\theta,\phi)$
6. Go to step 1

We derive $\nabla_\phi \mathcal{L}$ using the pathwise gradient estimator. We can reparameterize

$$z = h(x,\epsilon) = f_\phi(x) + \epsilon$$

where $\epsilon \sim \mathcal{N}(0, I)$. Thus

$$q_\phi(h(x,\epsilon_k) \mid x) = q_\phi(f_\phi(x) + \epsilon \mid x) = p(\epsilon)$$