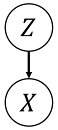
R250 Advanced topics in machine learning Topic 5: autoencoders

Variational autoencoder: the foundations

Damon Wischik, Cambridge University

- Kingma and Welling, *Autoencoding variational Bayes*, ICLR 2014
- Burda, Grosse, Salakhutdinov, *Importance weighted autoencoders*, ICLR 2016

A Bayesian's favourite model, of a biased coin

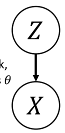


$Z \sim \text{Beta}(\alpha, \beta)$ for fixed parameters α, β

$X \sim \text{Bin}(1, Z)$

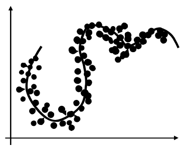
The Bayesian likes to ask:
given observations $x_1, x_2, \dots, x_N$ drawn from distribution X, what is the posterior distribution of Z?

A latent-random-variable generative model



$Z \sim \text{Normal}(0,1)$

$X \sim \text{Normal}(f_\theta(Z), \sigma^2)$



Z measures distance along the line

$f_\theta(Z)$ specifies the shape of the line

σ is noise around the line

The machine-learning modeller might ask:
given observations $x_1, x_2, \dots, x_N$ drawn from distribution X, how can I tune θ so that the model fits the data?

What does "train a generative model" mean?

It means "Pick θ so that the distribution $\Pr_X(x|\theta)$ is a good fit for the dataset $x_1, \dots, x_N$".

maximize $\log \text{lik}(\theta)$ over θ
where $\log \text{lik}(\theta) = \frac{1}{N}\sum_{i=1}^N \log \Pr_X(x_i|\theta)$

This also provides the evaluation metric,
$\frac{1}{M}\sum_{i=1}^M \log P_X(x_i|\hat\theta)$ summed over the holdout dataset $x_1, \dots, x_M$, using the fitted $\hat\theta$

How to train a generative model using Monte Carlo

In toy examples, we can write down a formula for the log likelihood $\log \Pr_X(x|\theta)$.
For interesting neural networks, this expression is intractable, so we approximate.

$$\log \text{lik}(\theta) = \frac{1}{N}\sum_{i=1}^N \log \Pr_X(x_i|\theta)$$

Law of total probability:
$\Pr_X(x) = \int_z \Pr_X(x|Z=z)\Pr_Z(z)\,dz = \mathbb{E}_Z \Pr_X(x|Z)$

$$= \frac{1}{N}\sum_{i=1}^N \log \mathbb{E}_Z \Pr_X(x_i|Z,\theta)$$

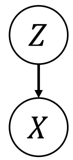
Jensen's inequality, for concave functions:
$f(\mathbb{E}X) \geq \mathbb{E}f(X)$

$$\geq \frac{1}{N}\sum_{i=1}^N \mathbb{E}_Z \log \Pr_X(x_i|Z,\theta)$$

Monte Carlo approximation, where z_j are sampled from Z

$$\approx \frac{1}{N}\sum_{i=1}^N \frac{1}{J}\sum_{j=1}^J \log \Pr_X\big(x_i|Z=z_j,\theta\big)$$

Our model gives us the explicit formula. Gradient descent does the rest.



$Z \sim \text{N}(0,1)$

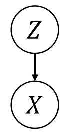
neural network, weights θ

$X \sim \text{N}(f_\theta(Z), \sigma^2)$

observations $x_1, \dots, x_N$

How to train a generative model using Monte Carlo

Our training objective: find θ to maximize the lower bound
$\log \text{lik lb}(\theta) = \frac{1}{N}\sum_{i=1}^N \mathbb{E}_Z\{\log \Pr_X(x|Z,\theta)\}$



$Z \sim \text{N}(0,1)$

neural network, weights θ

$X \sim \text{N}(f_\theta(Z), \sigma^2)$

observations $x_1, \dots, x_N$

```
For each datapoint x_i (or each batch):
    Generate one or more random z samples from Pr_Z
    Compute the loss function, L(θ) = −log Pr_X(x_i|Z = z, θ)
    as well as its gradient dL/dθ
    Update θ to reduce the loss function
```

PROBLEM: the Monte Carlo approximation is pretty terrible, for this problem, since most values of z give
$\Pr_X(x_i|Z=z,\theta) \approx \text{zero}$
This makes the lower bound very weak.

Digression on Importance Sampling

Given a random distribution Z and a function h, how can we approximate $\mathbb{E}_Z h(Z)$?

In our generative model, we picked $Z \sim N(0,1)$ and want to compute
$$\mathbb{E}_Z \underbrace{\log \Pr_X(x|Z,\theta)}_{h(Z)}$$
Let's sample instead from some other distribution $\tilde{Z}$ (our choice). We just have to throw in a correction factor.

MONTE CARLO APPROXIMATION

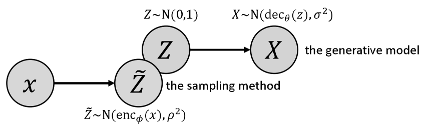
Sample $z_1, \dots, z_J$ from Z. Then $\mathbb{E}_Z h(Z) \approx \frac{1}{J}\sum_1^J h(z_j)$

IMPORTANCE SAMPLING APPROXIMATION

Choose a distribution $\tilde{Z}$. Sample $z_1, \dots, z_J$ from $\tilde{Z}$. Then $\mathbb{E}_Z h(Z) \approx \frac{1}{J}\sum_1^J h(z_j) \frac{\Pr_Z(z_j)}{\Pr_{\tilde{Z}}(z_j)}$

This approximation is valid for any distribution $\tilde{Z}$.
It works best (i.e. is good for small J) if $\tilde{Z}$ is biased in favour of values where $h(z)$ is large.

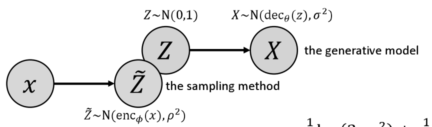
How to train a generative model using importance sampling



For each datapoint x_i (or each batch):

- Generate one or more random z samples from $\Pr_{\tilde{Z}}$
- and compute the loss, $L = -\log\left\{\Pr_X(x_i|Z=z,\theta)\frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z|x,\phi)}\right\}$
- as well as the gradients $\partial/\partial\theta$ and $\partial/\partial\phi$
- Update (θ,ϕ) to reduce the loss

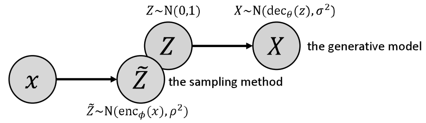
Our loss function can be thought of in terms of reconstruction error



$$\frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\big(x_i - \mathrm{dec}(\mathrm{enc}(x_i) + \mathrm{noise})\big)^2$$
"Denoising reconstruction loss"

Generate one or more random z samples from $\Pr_{\tilde{Z}}$
and compute the loss, $L = -\log\left\{\Pr_X(x_i|Z=z,\theta)\frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z|x,\phi)}\right\}$

Backpropagation needs derivatives, and there's a trick needed ...



Generate one or more random z samples from $\Pr_{\tilde{Z}}$
and compute the loss, $L = \cdots$
as well as the gradients $\partial/\partial\theta$ and $\partial/\partial\phi$

But $\Pr_{\tilde{Z}}$ depends on ϕ. How do we differentiate "sample from $\Pr_{\tilde{Z}}$"?

Some neat maths ("variational inference")

$$\begin{aligned}
\log \Pr_X(x|\theta) &= \log \mathbb{E}_Z \Pr_X(x|Z,\theta) \\
&= \log \mathbb{E}_{Z\sim\Pr_{\tilde{Z}}}\left\{\Pr_X(x|Z,\theta)\frac{\Pr_Z(Z)}{\Pr_{\tilde{Z}}(Z)}\right\} \\
&\geq \mathbb{E}_{Z\sim\Pr_{\tilde{Z}}} \log\left\{\Pr_X(x|Z,\theta)\frac{\Pr_Z(Z)}{\Pr_{\tilde{Z}}(Z)}\right\} \\
&= \underbrace{\mathbb{E}_{Z\sim\Pr_{\tilde{Z}}} \log \Pr_X(x|Z,\theta)}_{\text{"denoising reconstruction loss"}} - \underbrace{\mathbb{E}_{Z\sim\Pr_{\tilde{Z}}} \log\frac{\Pr_{\tilde{Z}}(Z)}{\Pr_Z(Z)}}_{\text{The Kullback-Leibler divergence, } \mathrm{KL}(\Pr_{\tilde{Z}} \parallel \Pr_Z) \geq 0, \text{ which is behaving here as a regularizer}}
\end{aligned}$$

With a little bit of algebra, the error in this approximation is $\mathrm{KL}(\Pr_{\tilde{Z}} \parallel \Pr_{Z|X=x})$

So, if our encoder is a perfect match for the Bayesian posterior of Z given $X=x$, the error is zero. (And also importance sampling is very efficient.)