Variational Autoencoder

January 5, 2019

This document is written as I read "Tutorial on Variational Autoencoders" by Carl Doersch [1].

1 Introduction

- Let x denote a data point. Let $\mathcal{X}$ denotes the space of all data points.
- We assume that the data points are generated from an unknown distribution P_{gt}, where "gt" stands for "ground truth."
  - Here, X can be an image of, say, a cat. P_{gt} is the distribution of cat images.
- Given many examples $x_1, x_2, \ldots, x_n$ sampled from P_{gt}, we are interested in learning a probability distribution "model" P which can be used to sample a new data point x so that it looks like it is being generated from P_{gt}.
- The probability distribution P that a variational autoencoder learns is a **latent variable model**.
  - To generate a sample with a latent variable model, we first sample a **latent vector** z from a space $\mathcal{Z}$.
    - $\mathcal{Z}$ is much smaller than $\mathcal{X}$.
    - We can think of z as being the low dimensional representation of the real data point x.
    - The probability distribution $P(z)$ where z is sampled from is chosen to be the multidimentional normal distribution $\mathcal{N}(0, I)$ where I is the identity matrix whose size equals to the dimention of $\mathcal{Z}$.
  - To generate a real data point, we pass z to a deterministic function $\mu_E(\cdot, \theta)$ where θ is the model's parameter.
    - In our case, μ_E is a deep neural networks, and θ is the networks' parameters.
  - The probability distribution of any data point x being generated from our model is given by:

    $$P(x) = \int_{\mathcal{Z}} P(x|z;\theta)P(z)\ \mathrm{d}z$$

    where

    $$P(x|z;\theta) = \mathcal{N}\big(x; \mu_D(z,\theta), \sigma_D^2 I\big),$$

    is the Gaussian distribution with mean $\mu_D(z,\theta)$ with element-wise variance of σ_D^2 with zero cross-element correlations. The subscript "D" here stands for "decoder." Note also that σ_D is a hyperparameter of the algorithm.
    - Note that, in other treatments of VAE, the element-wise variance is modeled by a neural network $\sigma_D^2(\cdot,\theta)$ instead of being fixed constant like in the treatment by Doersch. In this case, the covariance matrix is given by $\mathrm{diag}(\sigma_D^2(\cdot,\theta))$.

- The choice of using Gaussian distribution seems to come from expediency. It is easily computable and continuous in θ and x.

- With the above definition, we would like to find θ that maximizes the probability that the samples $x_1, x_2, \ldots, x_n$ are generated by our model:

$$\begin{aligned}\operatorname*{argmax}_\theta \left(\prod_{i=1}^{n} P(x_i) \right) &= \operatorname*{argmax}_\theta \left(\sum_{i=1}^{n} \log P(x_i) \right) \\ &= \operatorname*{argmax}_\theta \left(\frac{1}{n} \sum_{i=1}^{n} \log P(x_i) \right)\end{aligned}$$

- Note that the expression being argmax'ed in the last line is an unbiased estimate of $E_{x\sim P_{gt}}[\log P(x)]$. In fact, it is this expectation that we want to maximize. Again, in its full form, we want to solve the following problem:

$$\operatorname*{argmax}_\theta E_{x\sim P_{gt}} \left[\log \int_{\mathcal{Z}} P(x|z,\theta)P(z) \, \mathrm{d}z \right]$$

- The problematic part of the above problem is the integral inside the expectation. It is hard to approximate it accurately.

  - We may apply the usual trick: Monte Carlo integration. Sample $z_1, z_2, \ldots, z_m$ according of $P(z)$ and compute:

$$\frac{1}{m} \sum_{j=1}^{m} P(x|z_j,\theta)P(z_j).$$

    However, we would need a lot of samples to get an accurate estimate because most $P(x|z_j,\theta)$ would be close to zero. We need to throw a lot of darts before we find a $f(z_j,\theta)$ that is close enough to x.

  - MCMC might help here, but that is too complicated.

2 Variational Autoencoders

- The key idea behind VAE is, for each x, we will sample z's that are likely to produce x and compute $P(x)$ just from those.

- We need a new conditional probability distribution $Q(z|x)$, which, when given a data point x, generates latent vectors that are likely to produce x.

  - For VAE, this is given by:

$$Q(z|x) = \mathcal{N}\big(z; \mu_E(x,\phi), \operatorname{diag}(\sigma_E^2(x,\phi))\big)$$

    where $\mu_E(\cdot,\phi)$ and $\sigma_E(\cdot,\phi)$ are deep neural networks, and ϕ is the network parameters. The subscript "E" stands for "encoder."

- Consider $\log P(x)$, the log of the probability that a given data point x is generated. Because $P(x)$ does not depend on z, we have that:

$$\log P(x) = E_{z\sim Q(\cdot|x)}[\log P(x)]$$

By Bayes rule:

$$P(x) = \frac{P(x|z)P(z)}{P(z|x)}.$$

So,

$$\begin{aligned}
\log P(x) &= E_{z\sim Q(\cdot|x)}\left[\log \frac{P(x|z)P(z)}{P(z|x)}\right] \\
&= E_{z\sim Q(\cdot|x)}\left[\log\left(\frac{P(x|z)P(z)}{P(z|x)}\frac{Q(z|x)}{Q(z|x)}\right)\right] \\
&= E_{z\sim Q(\cdot|x)}[\log P(x|z)] - E_{z\sim Q(\cdot|x)}\left[\log \frac{Q(z|x)}{P(Z)}\right] + E_{z\sim Q(\cdot|x)}\left[\log \frac{Q(z|x)}{P(z|x)}\right] \\
&= E_{z\sim Q(\cdot|x)}[\log P(x|z)] - \mathcal{D}(Q(z|x)\|P(z)) + \mathcal{D}(Q(z|x)\|P(z|x))
\end{aligned}$$

where $\mathcal{D}(\cdot\|\cdot)$ denotes the Kullback–Leibler divergence between two probability distributions.

Rearranging, we have:

$$\log P(x) - \mathcal{D}(Q(z|x)\|P(z|x)) = E_{z\sim Q(\cdot|x)}[\log P(x|z)] - \mathcal{D}(Q(z|x)\|P(z))$$

- What we would like to maximize is $\log P(x)$. VAE, however, maximizes $\log P(x) - \mathcal{D}(Q(z|x)\|P(z|x))$ by maximizing $E_{z\sim Q(\cdot|x)}[\log P(x|z)] - \mathcal{D}(Q(z|x)\|P(z))$ instead.
- While $\log P(x) - \mathcal{D}(Q(z|x)\|P(z|x))$ is not the same as $\log P(x)$, we hope that $\mathcal{D}(Q(z|x)\|P(z|x))$ is effectively zero because we will be using a high-capacity model for $Q(z|x)$, and so it should be able to approximate any probability distribution. The Doersch paper [1] contains a proof that, as the capacity of the model goes to infinity, this term actually goes to 0 in a 1D case.
- The term $\mathcal{D}(Q(z|x)\|P(z))$ can be computed in closed form because it is a KL divergence between two Gaussian distributions. Abbreviating $\mu_E(x,\phi)$ as $\mu_E(x)$ and $\mathrm{diag}(\sigma_E^2(x,\phi))$ as $\Sigma_E(x)$, we have

$$\begin{aligned}
\mathcal{D}(Q(z|x)\|P(z)) &= \mathcal{D}(\mathcal{N}(\mu_E(x),\Sigma_E(x))\|\mathcal{N}(0,I)) \\
&= \frac{1}{2}\big(\mathrm{tr}(\Sigma_D(x)) + \mu_E(x)^T\mu_E(x) - k + \log\det\Sigma_E(x)\big)
\end{aligned}$$

  where k is the dimensionality of the distribution.
- The term $E_{z\sim Q(\cdot|x)}[\log P(x|z)] - \mathcal{D}(Q(z|x)\|P(z))$ together can be maximized through gradient descent. We just need a way to compute it in a forward pass that is differentiable.

  From the previous item, $\mathcal{D}(Q(z|x)\|P(z))$ can be computed just by passing x to μ_E and σ_E. However, $E_{z\sim Q(\cdot|x)}[\log P(x|z)]$ is more problematic because it is an expectation.

  The VAE paper estimates $E_{z\sim Q(\cdot|x)}[\log P(x|z)]$ by taking a sample x' from $\mathcal{N}(\mu_D(x),\Sigma_D(x))$ and compute $\log P(x'|z)$. To generate the sample, there's a special unit that connects the encoder (i.e., μ_E and σ_E) to the decoder (i.e., μ_D) that takes as input a $\xi\sim\mathcal{N}(0,I)$. Then, the unit computes:

$$x' = \mu_E(x,\phi) + \xi\odot|\sigma_E(x,\phi)|$$

  where $\odot$ denotes element-wise multiplication. Note that ξ comes as an input to the whole encoder-decoder network. So, we are not just passsing x in. We pass (x,ξ) in.
- The introduction of the unit to sample from a distribution inside a network by taking extra input is called the **reparameterization trick**. This can be done easily with the Gaussian distribution as we can generate deterministic networks that compute the distribution paramaters. Then, from these parameters, we can easily sample by passing in random numbers.

3 Conditional Variational Autoencoders

- CVAE solves the problem of generating a sample y from a distribution conditioned on another input x.
  - x might be an image that has a hole in it, and we would like y to be a plausible image with the hole filled.
- To generate y from x, we introduce a latent variable $z \sim \mathcal{N}(0, I)$. Then, we say that:

  $$P(y|x, z) = \mathcal{N}(\mu_D(z, x), \sigma_D^2 I)$$

  where, again, μ_D is a neural network, and σ_D is a hyperparameter.
- Using the same derivation as in the previous section, we have that:

  $$\log P(y|x) - \mathcal{D}(Q(z|y, x)\|p(z|y, x)) = E_{z\sim Q(\cdot|y,x)}\big[\log P(y|z, x)\big] - \mathcal{D}(Q(z|y, x)\|P(z|x)).$$

  Because z is sampled independent of x, we have:

  $$\log P(y|x) - \mathcal{D}(Q(z|y, x)\|p(z|y, x)) = E_{z\sim Q(\cdot|y,x)}\big[\log P(y|z, x)\big] - \mathcal{D}(Q(z|y, x)\|P(z)).$$

  From this equation, we can derive the training algorithm.

References

[1] Carl Doersch. Tutorial on variational autoencoders, 2016.