Tutorial on Variational Autoencoders

1 Citation

Doersch, Carl. "Tutorial on variational autoencoders." arXiv preprint arXiv:1606.05908 (2016).

`https://arxiv.org/pdf/1606.05908.pdf`

2 Abstract

VAEs do unsupervised learning and can generate examples that resemble those from the training dataset.

3 Introduction

Given examples X distributed according to $P_{gt}(X)$, we seek to infer a $P(X)$ (which should be very similar to $P_{gt}(X)$) that we can sample from. Prior work has either required strong assumptions of the data, severe approximations, or compute-intensive techniques (e.g. Markov Chain Monte Carlo). VAEs do not have these limitations.

We assume that there are some continuous latent variables z. We estimate $P(X|z,\theta) \sim \mathcal{N}(X|f(z;\theta),\sigma^2 I)$, where θ are parameters of neural network f and σ is a hyperparameter. Note that we could use a Bernoulli (instead of Gaussian) distribution for binary output variables. Then, we aim to pick the θ to maximize:

$$P(X) = \int P(X|z;\theta)P(z)dz \tag{1}$$

4 Variational Autoencoders

We model $P(z) \sim \mathcal{N}(0, I)$, and let the neural network figure out how to map this Gaussian noise into an example.

Now, for most z, we expect $P(X|z)$ to be zero because in real life, examples usually lie on a low-dimensional manifold in a high dimensional space. Suppose we have a probability distribution $Q(z)$ that puts most probability mass on z values on this manifold. Now let's try and relate the quantities we have been discussing ($\mathcal{D}$ is KL divergence):

$$\begin{aligned}
\mathcal{D}[Q(z)||P(z|X)] &= \mathbb{E}_{z\sim Q}[\log Q(z) - \log P(z|X)] && (2)\\
\mathcal{D}[Q(z)||P(z|X)] &= \mathbb{E}_{z\sim Q}[\log Q(z) - \log P(X|z) - \log P(z)] + \log P(X) \text{ (Bayes rule)} && (3)\\
\log P(X) - \mathcal{D}[Q(z)||P(z|X)] &= \mathbb{E}_{z\sim Q}[\log P(X|z)] - \mathcal{D}[Q(z)||P(z)] && (4)\\
\log P(X) - \mathcal{D}[Q(z|X)||P(z|X)] &= \mathbb{E}_{z\sim Q}[\log P(X|z)] - \mathcal{D}[Q(z|X)||P(z)] \text{ (Turn } Q(z) \text{ into } Q(z|X)) && (5)
\end{aligned}$$

Notice that $Q(z|X)$ and $P(X|z)$ resemble an encoder and decoder, respectively. If we can find a way to maximize the RHS of the equation, we will maximize the LHS of the equation, which is the probability density that we care about ($P(X)$) plus an error term (call this ERROR TERM). When $Q(z|X)$ is high capacity, the error term will be near zero, so let's ignore it. Let's look at the RHS.

First, we assume $Q(z|X) \sim \mathcal{N}(z|\mu(X), \Sigma(X))$, where the μ and Σ functions are computed by the neural network. Now, since we assumed $P(z)$ is Gaussian, we can easily compute an analytical expression for $\mathcal{D}[Q(z|X)||P(z)]$ in terms of their means and covariances (see paper). For $\mathbb{E}_{z\sim Q}[\log P(X|z)]$, we could assume that we can approximate this with $\log P(X|z)$ where X is drawn randomly from the dataset. The problem with this approach is that we lose the dependency on Q. To fix this and not introduce sampling inside the network (we cannot backpropagate through stochastic units), we define $\epsilon \sim \mathcal{N}(0, I)$ and define $z = \mu(X) + \Sigma^{1/2}(X) * \epsilon$ - this reparameterization trick now lets us sample from the zero-mean unit-variance Gaussian rather than Q. With this trick, we just need to compute the gradient of the following (where D represents a uniform distribution over the dataset):

$$\mathbb{E}_{X\sim D}[\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[\log P(X|z = \mu(X) + \Sigma^{1/2}(X) * \epsilon)] - \mathcal{D}[Q(z|X)||P(z)]] \tag{6}$$

So, we just need to sample an example from our dataset and some Gaussian noise. Then we feed both into our neural network to compute $Q(z|X)$ followed by $P(X|z)$. These values then feed into the loss function above, and we backpropagate to optimize our weights. Also, because of how we have defined f, the math works out to $P(X|z) = ||X - f(z;\theta)||^2$. Because of the way we defined Q and $P(z)$, the math works out to $\mathcal{D}[Q(z|X)||P(z)] = \mathcal{D}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)]$

At test time, we don't need the encoder $Q(z|X)$. We can simply sample $z \sim \mathcal{N}(0, I)$ and feed it through the decoder $f(z;\theta)$. Evaluating $P(X)$ is not possible because of ERROR TERM, so the best we can do is get a lower bound.

There's no guarantee that ERROR TERM will be near zero for high capacity models, but we can prove this claim under specific assumptions and VAEs work well in practice.

You can look at our equation through an information theoretic lens. $-\log P(X)$ is the number of bits needed to represent X. $\mathcal{D}[Q(z|X)||P(z)]$ is information about X that Q gives us (recall that $P(z)$ contains no information about X). $-\log P(X|z)$ is the information needed to reconstruct X from z. ERROR TERM is a penalty that we impose because $Q(z|X)$ is a suboptimal encoding of X to z.

Sparse autoencoders minimize the following loss function (a weighted sum of reconstruction cost and L0 norm of the encoding):

$$||\phi(\psi(X)) - X||^2 + \lambda||\psi(X)||_0 \tag{7}$$

VAEs don't have any regularization terms, like the L0 norm above (which encourages sparsity). The closest thing that VAEs have is the σ hyperparameter, which trades off how closely we expect the reconstruction to match X.

5 Conditional Variational Autoencoders

Sometimes we may want to condition our VAE with something. For example, we may want to fill in missing pixels in an image conditioned on available pixels. A supervised learning problem is not suitable for this pixel-filling problem because there may be many possible fillings that are sensible (i.e. we have a one-to-many mapping from available pixels to missing pixels). We can easily tweak the equations of our VAE (see paper) to get a Conditional VAE.

6 Examples

We use the default MNIST Autoencoder that comes with Caffe (but we use ReLU activation). MNIST pixels are constrained to [0, 1], so we use sigmoid cross entropy. We experimented with 5 learning rates and trained the model without many errors. The generated digits look reasonable, but there are some digits that look like a hybrid of two other digits. The dimensionality of z does not matter much. Just don't make it too small or we won't be able to capture enough information in the encoding and don't make it too large or gradient descent will struggle to optimize the network.

7 MNIST Conditional Variational Autoencoder

You could make a VAE that generates the left half of a digit given the right half. Injecting noise into the dataset to generate more examples helps.