Auto encoder

An autoencoder is a neural network that is trained to attempt to copy its input to its output. Internally, it has a hidden layer h that describes a code used to represent the input. The network may be viewed as consisting of two parts: an encoder function h = f (x) and a decoder that produces a reconstruction r = g(h).
If an autoencoder succeeds in simply learning to set g(f (x)) = x everywhere, then it is not especially useful. Instead, autoencoders are designed to be unable to learn to copy perfectly. Usually they are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data. Modern autoencoders have generalized the idea of an encoder and a decoder beyond deterministic functions to stochastic mappings pencoder(h | x) and pdecoder(x | h).

Introduction

"Generative modeling" is a broad area of machine learning which deals with models of distributions P(X), defined over datapoints X in some potentially high-dimensional space X. For instance, images are a popular kind of data for which we might create generative models. Each "datapoint" (image) has thousands or millions of dimensions (pixels), and the 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. Exactly what it means to "capture" these dependencies depends on exactly what we want to do with the model. One straightforward kind of generative model simply allows us to compute P(X) numerically. In the case of images, X values which look like real images should get high probability, whereas images
that look like random noise should get low probability. However, models like this are not necessarily useful: knowing that one image is unlikely does not help us synthesize one that is likely.
Instead, one often cares about producing more examples that are like those already in a database, but not exactly the same. We could start with a database of raw images and synthesize new, unseen images. We might take in a database of 3D models of something like plants and produce more of them to fill a forest in a video game. We could take handwritten text and try to produce more handwritten text. Tools like this might actually be useful

for graphic designers. 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.

Training this type of model has been a long-standing problem in the machine learning community, and classically, most approaches have had one of three serious drawbacks. First, they might require strong assumptions about the structure in the data. Second, they might make severe approximations, leading to suboptimal models. Or third, they might rely on computationally expensive inference procedures like Markov Chain Monte Carlo. More recently, some works have made tremendous progress in training neural networks as powerful function approximators through backpropagation [9]. These advances have given rise to promising frameworks which can use backpropagation-based function approximators to build generative models. One of the most popular such frameworks is the Variational Autoencoder [1, 3], the subject of this tutorial. The assumptions of this model are weak, and training is fast via backpropagation. VAEs do make an approximation, but the error introduced by this approximation is arguably small given high-capacity models. These characteristics have contributed to a quick rise in their popularity.

This tutorial is intended to be an informal introduction to VAEs, and not a formal scientific paper about them. It is aimed at people who might have uses for generative models, but might not have a strong background in the variatonal Bayesian methods and “minimum description length” coding models on which VAEs are based. This tutorial began its life as a presentation for computer vision reading groups at UC Berkeley and Carnegie Mellon, and hence has a bias toward a vision audience. Suggestions for improvement are appreciated.

Latent variable

A central problem in machine learning is to learn a complicated probability distribution p(x) with only a limited set of high-dimensional data points x drawn from this distribution. For example, to learn the probability distribution over images of cats we need to define a distribution which can

model complex correlations between all pixels which form each image. Modelling this distribution directly is a challenging task or even unfeasible in finite time.

Instead of modelling p(x) directly, we can introduce an unobserved latent variable z and define a conditional distribution p(x|z) for the data, which is called a likelihood. In probabilistic terms z can be interpreted as a continuous random variable. For the example of cat images, z could contain a hidden representation of the type of cat, its color or shape.

Having z, we can further introduce a prior distribution p(z) over the latent variables to compute the joint distribution over observed and latent variables:

$$p(x, z) = p(x|z)p(z)$$

This joint distribution allows us to express the complex distribution p(x) in a more tractable way. Its components, p(x|z) and p(z) are usually much simpler to define e.g. by using distributions from the exponential family.

To obtain the data distribution p(x) we need to marginalize over the latent variables

$$p(x) = \int p(x, z)dz = \int p(x|z)p(z)dz$$

Variational autoencoder

have a look at the parametric joint distribution of equation:

$$p_\theta(x, z) = p_\theta(x|z)p_\theta(z)$$

Theta denotes the unknown parameters of the model that can be learned using deep neural networks

VAE uses such deep neural networks to parameterize the probability distributions that define the latent variable model. Moreover, it provides an efficient approximation inference procedure which scales to large data sets. It is defined by a generative model (the latent variable model), an inference network (the variational approximation) and a way of how to learn the parameters of the VAE.

The generative model is given by equation (6) and here z is a continuous latent variable with K-dimensions. The prior of it is typically a Gaussian with zero mean and an identity covariance matrix,

$$p_\theta(z) = \mathcal{N}(z; 0, \mathcal{I})$$

Gaussian prior with zero mean and identity covariance; Equation (7)

The likelihood is known as the decoder, which is typically a Gaussian distribution for continuous data whose parameter theta are computed by passing the latent state z through a deep neural network. The likelihood then looks like the following,

$$p_\theta(x|z) = \mathcal{N}(x; \mu, \nu)$$

Likelihood as continuous Gaussian distribution; Equation (8)

The mean and the variance are parameterized by two deep neural networks which output vector is of dimensionality D, i.e. the dimensionality of the observation x. The parameter theta are the weights and biases of the decoder neural networks.

The inference network is known as the encoder and allows us to compute the parameters of the posterior approximation. Instead of having a set of parameter for each data point, the variational parameters phi are shared across all data points. Again, in the VAE setting we use deep neural networks that take an input data point and outputs the mean and diagional covariance matrix of the corresponding Gaussian variational approximation,

$$q_\phi(z|x) = \mathcal{N}(z; \mu_q, \nu_q)$$

The shared variational parameters phi are the weights and biases of the encoder neural networks.

the marginal distribution p(x) (parameterized by theta) is intractable in many cases and needs to be approximated. An approximation can be achieved by using the ELBO. To make it explicit that ELBO depends on some parameter theta we can rewrite it as

$$\mathcal{F}(\theta, q) = \mathbb{E}_{q(z)}\left[\log \frac{p_\theta(x, z)}{q(z)}\right]$$

To learn the parameter we could maximize the ELBO with respect to its parameter using Expectation Maximization (EM). For the VAE setting the maximization is instead over q over the parameter phi. Thus, we can decompose the ELBO in two terms:

$$\mathcal{F}(\theta, \phi) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - KL[q_\phi(z|x)||\log p_\theta(z)]$$

The first term of F is the reconstruction loss which encourages the likelihood and inference network to reconstruct the data accurately. The second term is the regularization loss and penalizes posterior approximations that are too far from the prior. Both neural networks with
parameter phi and theta can be effectively computed via gradient descent with back-propagation. Moreover, the parameters are updated jointly and not iteratively like in EM.

Conditional variational autoencoder

Recall, on VAE, the objective is:

logP(X)−DKL[Q(z|X)‖P(z|X)]=E[logP(X|z)]−DKL[Q(z|X)‖P(z)]log⁡P(X)−DKL[Q(z|X)‖P(z|X)]=E[log⁡P(X|z)]−DKL[Q(z|X)‖P(z)]
that is, we want to optimize the log likelihood of our data P(X)P(X) under some “encoding” error. The original VAE model has two parts: the encoder Q(z|X)Q(z|X) and the decoder P(X|z)P(X|z).
Looking closely at the model, we could see why can’t VAE generate specific data, as per our example above. It’s because the encoder models the latent variable zz directly based on XX, it doesn’t care about the different type of XX. For example, it doesn’t take any account on the label of XX.
Similarly, in the decoder part, it only models XX directly based on the latent variable zz.

We could improve VAE by conditioning the encoder and decoder to another thing(s). Let's say that other thing is cc, so the encoder is now conditioned to two variables XX and cc: $Q(z|X,c)Q(z|X,c)$. The same with the decoder, it's now conditioned to two variables zz and cc: $P(X|z,c)P(X|z,c)$.

Hence, our variational lower bound objective is now in this following form:

$$\log P(X|c) - D_{KL}[Q(z|X,c) \| P(z|X,c)] = E[\log P(X|z,c)] - D_{KL}[Q(z|X,c) \| P(z|c)] \log P(X|c) - D_{KL}[Q(z|X,c) \| P(z|X,c)] = E[\log P(X|z,c)] - D_{KL}[Q(z|X,c) \| P(z|c)]$$

i.e. we just conditioned all of the distributions with a variable cc.

Now, the real latent variable is distributed under $P(z|c)P(z|c)$. That is, it's now a conditional probability distribution (CPD). Think about it like this: for each possible value of cc, we would have a $P(z)P(z)$. We could also use this form of thinking for the decoder.