Deep Generative Models (Part 2)

Woojeong Jin
woojeong.jin@usc.edu

September 17, 2019

1 Back-Propagation through Random Operations

Traditional neural networks implement a deterministic transformation of some input variables x. When developing generative models, we often wish to extend neural networks to implement stochastic transformations of x. One straightforward way to do this is to augment the neural network with extra inputs z that are sampled from some simple probability distribution, such as a uniform or Gaussian distribution. The neural network can then continue to perform deterministic computation internally, but the function $f(x, z)$ will appear stochastic to an observer who does not have access to z.

As an example, let us consider the operation consisting of drawing samples y from a Gaussian distribution with mean μ and variance σ^2:

$$y \sim N(\mu, \sigma^2) \tag{1}$$

Because an individual sample of y is produced not by a function, but rather by a sampling process whose output changes every time we query it, it may seem counterintuitive to take the derivatives of y with respect to the parameters of its distribution, μ and σ^2. However, we can rewrite the sampling process as transforming an underlying random value $z \sim N(z; 0, 1)$ to obtain a sample from the desired distribution:

$$y = \mu + \sigma z \tag{2}$$

We are now able to back-propagate through the sampling operation, by regarding it as a deterministic operation with an extra input z.

1.1 Back-Propagating through Discrete Stochastic Operations

Suppose that the model takes inputs x and parameters θ, both encapsulated in the vector ω, and combines them with random noise z to produce y:

$$y = f(z; \omega) \tag{3}$$

Because y is discrete, f must be a step function. The derivatives of a step function are not useful at any point. Right at each step boundary, the derivatives are undefined. The large problem is that the derivatives are zero almost everywhere on the regions between step boundaries. The derivatives of any cost function $J(y)$ therefore do not give any information for how to update the model parameters θ.

The REINFORCE algorithm provides a framework defining a family of simple but powerful solutions [1]. The core idea is that even though $J(f(z; \omega))$ is a step function with useless derivatives, the expected cost $E_{z \sim p(z)} J(f(z; \omega))$ is often a smooth function amenable to gradient descent.

The simplest version of REINFORCE can be derived by simply differentiating the expected cost:

$$\mathbb{E}_z[J(y)] = \sum_y J(y)p(y) \tag{4}$$

$$\frac{\partial \mathbb{E}[J(y)]}{\partial \omega} = \sum_y J(y)\frac{\partial p(y)}{\partial \omega} \tag{5}$$

$$= \sum_y J(y)p(y)\frac{\partial \log p(y)}{\partial \omega} \tag{6}$$

$$\simeq \frac{1}{m}\sum_{y^{(i)}\sim p(y), i=1}^{m} J(y^{(i)})\frac{\partial \log p(y^{(i)})}{\partial \omega}. \tag{7}$$

Equation (5) relies on the assumption that J does not reference ω directly. It is trivial to extend the approach to relax this assumption. Equation (6) exploits the derivative rule for the logarithm. $\frac{\partial \log p(y)}{\partial \omega} = \frac{1}{p(y)}\frac{\partial p(y)}{\partial \omega}$.

One issue with the simple REINFORCE estimator is that it has a very high variance, so that many samples of y need to be drawn to obtain a good estimator of the gradient, or equivalently, if only one sample is drawn, SGD will converge very slowly and will require a smaller learning rate. It is possible to considerably reduce the variance of that estimator by using variance reduction methods. The idea is to modify the estimator so that its expected value remains unchanged but its variance gets reduced. In the context of REINFORCE, the proposed variance reduction methods involve the computation of a baseline that is used to offset $J(y)$.

$$E_{p(y)}\left[(J(y) - b(\omega))\frac{\partial \log p(y)}{\partial \omega}\right] = E_{p(y)}\left[J(y)\frac{\partial \log p(y)}{\partial \omega}\right] - b(\omega)E_{p(y)}\left[\frac{\partial \log p(y)}{\partial \omega}\right] \tag{8}$$

$$= E_{p(y)}\left[J(y)\frac{\partial \log p(y)}{\partial \omega}\right] \tag{9}$$

$$E_{p(y)}\left[\frac{\partial \log p(y)}{\partial \omega}\right] = \sum_y p(y)\frac{\partial \log p(y)}{\partial \omega} \tag{10}$$

$$= \sum_y \frac{\partial p(y)}{\partial \omega} \tag{11}$$

$$= \frac{\partial}{\partial \omega}\sum_y p(y) = \frac{\partial}{\partial \omega}1 = 0 \tag{12}$$

2 Directed Generative Nets

Directed graphical models make up a prominent class of graphical models. While directed graphical models have been very popular within the greater machine learning community, within the smaller deep learning community they have until roughly 2013 been overshadowed by undirected models such as the RBM. In this section we review some of the standard directed graphical models that have traditionally been associated with the deep learning community.

2.1 Sigmoid Belief Networks

Sigmoid belief networks [2] are a simple form of directed graphical model with a specific kind of conditional probability distribution. In general, we can think of a sigmoid belief network as having a vector of binary states s, with each element of the state influenced by its ancestors:

$$p(s_i) = \sigma\left(\sum_{j<i} W_{j,i}s_j + b_i\right). \tag{13}$$

The most common structure of sigmoid belief network is one that is divided into many layers, with ancestral sampling proceeding through a series of many hidden layers and then ultimately generating the visible layer. This structure is very similar to the deep belief network, except that the units at the beginning of the sampling process are independent from each other, rather than sampled from a restricted Boltzmann machine.

2.2 Differentiable Generator Networks

Many generative models are based on the idea of using a differentiable generator network. The model transforms samples of latent variables z to samples x or to distributions over samples x using a differentiable function $g(z; \theta^{(g)})$, which is typically represented by a neural network. This model class includes variational autoencoders, which pair the generator net with an inference net; generative adversarial networks, which pair the generator network with a discriminator network; and techniques that train generator networks in isolation.

2.3 Variational Autoencoders

The variational autoencoder, or VAE [3, 4], is a directed model that uses learned approximate inference and can be trained purely with gradient-based methods.

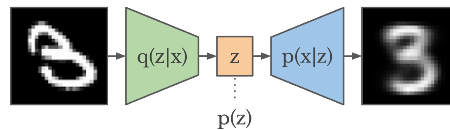

Figure 1: Variational Autoencoders.

To generate a sample from the model, the VAE first draws a sample z from the code distribution $p_{model}(z)$. The sample is then run through a differentiable generator network $g(z)$. Finally, x is sampled from a distribution $p_{model}(x; g(z)) = p_{model}(x|z)$. During training, however, the approximate inference network (or encoder) $q(z|x)$ is used to obtain z, and $p_{model}(x|z)$ is then viewed as a decoder network.

The key insight behind variational autoencoders is that they can be trained by maximizing the variational lower bound $L(q)$ associated with data point x:

$$L(q) = \mathbb{E}_{z \sim q(z|x)} \log p_{model}(x|z) - D_{KL}(q(z|x)||p_{model}(z)) \tag{14}$$

We recognize the first term as the reconstruction log-likelihood found in other autoencoders. The second term tries to make the approximate posterior distribution $q(z|x)$ and the model prior $p_{model}(z)$ approach each other.

The variational autoencoder approach is elegant, theoretically pleasing, and simple to implement. It also obtains excellent results and is among the state-of-the-art approaches to generative modeling. Its main drawback is that samples from variational autoencoders trained on images tend to be somewhat blurry. The causes of this phenomenon are not yet known.

2.4 Generative Adversarial Networks

Generative adversarial networks, or GANs [5], are another generative modeling approach based on differentiable generator networks. The main motivation for the design of GANs is that the learning process requires neither approximate inference nor approximation of a partition function gradient.

Generative adversarial networks are based on a game theoretic scenario in which the generator network must compete against an adversary. The generator network directly produces samples $x = g(z; \theta^{(g)})$. Its adversary, the discriminator network, attempts to distinguish between samples drawn from the training data and samples drawn from the generator. The discriminator emits a probability value given by $d(x; \theta^{(d)})$, indicating the probability that x is a real training example rather than a fake sample drawn from the model.

The simplest way to formulate learning in generative adversarial networks is as a zero-sum game, in which a function $v(\theta(g), \theta(d))$ determines the payoff of the discriminator. The generator receives $-v(\theta(g), \theta(d))$ as its own payoff. During learning, each player attempts to maximize its own payoff, so that at convergence

$$g^* = \arg\min_g \max_d v(g, d). \tag{15}$$

The default choice for v is

$$v(\theta^{(g)}, \theta^{(d)}) = \mathbb{E}_{x \sim p_{data}} \log d(x) + \mathbb{E}_{x \sim p_{model}} \log(1 - d(x)). \tag{16}$$

This drives the discriminator attempt to learn to correctly classify samples as real or fake. Simultaneously, the generator attempts to fool the classifier into believing its samples are real. At convergence, the generator's samples are indistinguishable from real data, and the discriminator outputs $\frac{1}{2}$ everywhere. The discriminator may then be discarded.

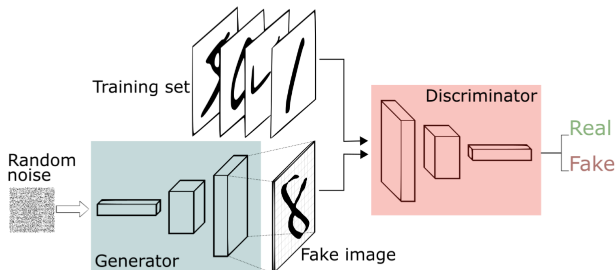

Figure 2: Generative Adversarial Networks.

2.5 Generative Moment Matching Networks

Generative moment matching networks (GMMNs) [6] are another form of generative model based on differentiable generator networks. Unlike VAEs and GANs, they do not need to pair the generator network with any other network—neither an inference network, as used with VAEs, nor a discriminator network, as used with GANs. Training a GAN, however, requires careful optimization of a difficult minmax problem. Instead, generative moment matching networks can be trained by minimizing a cost function called maximum mean discrepancy, or MMD. GMMNs are similar to GANs, but they replace the Discriminator with the MMD measure, making their optimization more stable. Maximum mean discrepancy is a simple objective that can be interpreted as matching all orders of statistics between a dataset and samples (generated dataset) from the model.

Suppose we are given two sets of samples $X = \{x_i\}_{i=1}^N$ and $Y = \{y_j\}_{j=1}^M$ and are asked whether the generating distributions $P_X = P_Y$. Maximum mean discrepancy is a frequentist estimator for answering this question, also known as the two sample test. The idea is simple: compare statistics between the two datasets and if they are similar then the samples are likely to come from the same distribution.

$$L_{MMD}^2 = \left\| \frac{1}{N} \sum_{i=1}^{N} \phi(x_i) - \frac{1}{M} \sum_{i=1}^{M} \phi(y_i) \right\|^2 \tag{17}$$

where ϕ is a kernel function.

2.6 Convolutional Generative Networks

When generating images, it is often useful to use a generator network that includes a convolutional structure. To do so, we use the “transpose” of the convolution operator, described in Figure 3. This approach often yields more realistic images and does so using fewer parameters than using fully connected layers without parameter sharing.

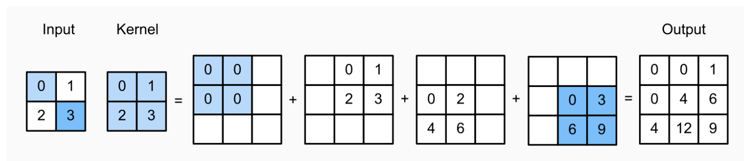

Figure 3: Transposed Convolution.

Convolutional networks for recognition tasks have information flow from the image to some summarization layer at the top of the network, often a class label. As this image flows upward through the network, information is discarded as the representation of the image becomes more invariant to nuisance transformations. In a generator network, the opposite is true. Rich details must be added as the representation of the image to be generated propagates through the network, culminating in the final representation of the image, which is of course the image itself, in all its detailed glory, with object positions and poses and textures and lighting.

2.7 Auto-Regressive Networks

Auto-regressive networks are directed probabilistic models with no latent random variables. Auto-regressive models are well known for sequence data. The conditional probability distributions in these models are represented by neural

networks (sometimes extremely simple neural networks, such as logistic regression). The graph structure of these models is the complete graph. They decompose a joint probability over the observed variables using the chain rule of probability to obtain a product of conditionals of the form $P(x_d|x_{d-1},...,x_1)$. Such models have been called fully-visible belief networks (FVBNs) and used successfully in many forms, first with logistic regression for each conditional distribution, and then with neural networks with hidden units.

This graphical model represents the joint distribution of random variables $x_1,...,x_n$ with

$$P(x_1,...,x_n) = \prod_{i=1}^{n} p(x_i|x_1,...,x_{i-1}). \tag{18}$$

2.8 Linear Auto-Regressive Networks

The simplest form of auto-regressive network has no hidden units and no sharing of parameters or features. Each $P(x_d|x_{d-1},...,x_1)$ is parametrized as a linear model (linear regression for real-valued data, logistic regression for binary data, softmax regression for discrete data).

$$P(x_i = 1|x_1,...,x_{i-1}) = \text{sigmoid}(w_0 + \sum_{j<i} w_j x_j) \tag{19}$$

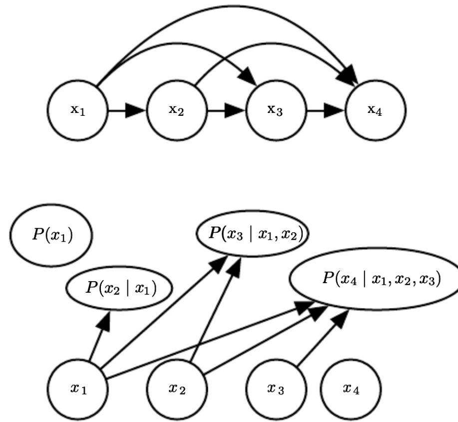

Figure 4: Fully visible belief network.

Linear auto-regressive networks are essentially the generalization of linear classification methods to generative modeling. They therefore have the same advantages and disadvantages as linear classifiers. Like linear classifiers, they maybe trained with convex loss functions and sometimes admit closed form solutions (as in the Gaussian case). Like linear classifiers, the model itself does not offer a way of increasing its capacity, so capacity must be raised using techniques like basis expansions of the input or the kernel trick.

2.9 Neural Auto-Regressive Networks

Neural auto-regressive networks [7] have the same left-to-right graphical model as logistic auto-regressive networks (figure 4) but employ a different parametrization of the conditional distributions within that graphical model structure. The new parametrization can also improve generalization by introducing a parameter sharing and feature sharing principle common to deep learning in general. The models were motivated by the objective of avoiding the curse of dimensionality arising out of traditional tabular graphical models.

$$P(x_i = 1|x_1,...,x_{i-1}) = P(x_i = 1|g_i(x_1,...,x_{i-1})) \tag{20}$$

where $g_i(x_1,...,x_{i-1})$ is the vector-value ouput of the i-th group of output units, and it gives the value of the parameters of the distribution of x_i.

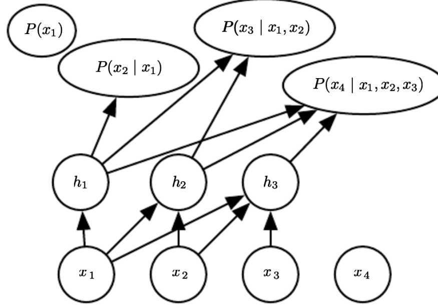

Figure 5: A neural auto-regressive network predicts the i-th variable x_i from the $i-1$ previous ones, but is parametrized so that features (groups of hidden units denoted h_i) that are functions of $x_1, ..., x_i$ can be reused in predicting all the subsequent variables $x_{i+1}, x_{i+2}, ..., x_d$.

2.10 Neural Auto-Regressive Density Estimator (NADE)

The neural auto-regressive density estimator (NADE) [8] is a very successful recent form of neural auto-regressive network. The connectivity is the same as for the original neural auto-regressive network of [7], but NADE introduces an additional parameter sharing scheme. The parameters of the hidden units of different groups j are shared.

$$P(x_i = 1 | x_1, ..., x_{i-1}) = \text{sigmoid}(c_i + V_{i,:} h_i) \tag{21}$$

$$h_i = \text{sigmoid}(b + W_{:,1} x_1 + ... + W_{:,i-1} x_{i-1}) \tag{22}$$

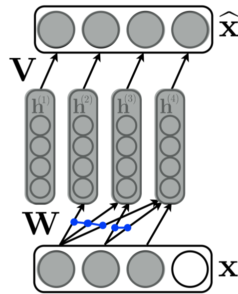

Figure 6: Neural Auto-regressive Density Estimator.

Training under NADE is done by maximizing the average log-likelihood of the parameters given the training set:

$$\frac{1}{N} \sum_{i=1}^{N} \log p(x^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{D} \log p(x_k^{(i)} | x_{<k}^{(i)}). \tag{23}$$

References

[1] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[2] Radford M Neal. Learning stochastic feedforward networks. *Department of Computer Science, University of Toronto*, 64:1283, 1990.

[3] Diederik P Kingma. Fast gradient-based inference with continuous latent variable models in auxiliary form. *arXiv preprint arXiv:1306.0733*, 2013.

[4] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *International Conference on Machine Learning*, pages 1718–1727, 2015.

[7] Yoshua Bengio and Samy Bengio. Modeling high-dimensional discrete data with multi-layer neural networks. In *Advances in Neural Information Processing Systems*, pages 400–406, 2000.

[8] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011.