An Overview of Generative Networks

Paul Gallagher

May 17, 2018

In this short note, we cover 2 kinds of Generative Neural Networks: Variational Autoencoders (VAEs), and Generative Adversarial Networks (GANs). We also look at some ways of combining the two and improving their performance.

Rather than classifying given data, the goal of a Generative Network is to study a base set of data, and learn to construct new data on its own. Such networks could have applications to image reconstruction, time series prediction, and a deeper understanding of neural net architecture.

Both approaches have the same basic idea: Learn a generator G that maps from a latent variable probability space $(\mathcal{Z}, p_z)$ into the data space (X, p_x) such that the induced probability measure on $\mathcal{X}$ from G is the same as the original data distribution p_x. However, VAEs and GANs have very different ways of computing this generator function.

All code can be found at my GitHub: https://github.com/paul-r-gall/generative-models

Variational Autoencoders

We begin with the idea of an **autoencoder** network. Let S be some high dimensional dataset in $\mathbb{R}^n$, and let $m << n$. Consider the following optimization problem:

$$\pi_\theta : \mathbb{R}^n \to \mathbb{R}^m \text{ and } \iota_\theta : \mathbb{R}^m \to \mathbb{R}^n$$
$$C(\theta) = \sum_{s \in S} d(s, \iota(\pi(s)))$$

where d is some metric on $\mathbb{R}^n$ and π and ι are given by neural networks. Then by minimizing this cost function, we will force $\iota \circ \pi$ to be an approximation of the identity map when restricted to this dataset. In particular, given any datapoint d, the projection $\pi(d)$ will encode all of the information needed to approximately reconstruct the datapoint.

Here, $Z = \mathbb{R}^m$ is our latent variable space. For the MNIST handwritten digit dataset, $\mathbb{R}^m$ might control not just the digit, but also slant, heaviness, and other non-categorical factors.

One could hope that by sampling $z \in \mathbb{R}^m$ and applying the learned ι, we could generate new datasets. However, this does not yet work because we do not know what distribution $\pi(x)$ takes on $\mathbb{R}^m$. The goal of a Variational Autoencoder is to force the induced distribution on the latent space Z to be something we can easily understand, such as a unit Gaussian.

Another way to view this is by saying that we are trying to learn a distribution $P(X|z;\theta)$ such that the expression:

$$P(X) = \int P(X|z;\theta)P(z)dz$$

is as large as possible on the points X in our dataset. However, this integral is intractable to compute at each step, and so we must find another function to maximize. We note that $P(X|z)$ (θ will be implied) is practically zero for the vast majority of likely z, and so we only want to compute this integral for z which are likely to induce a datapoint X. That is, we will learn a new distribution $Q(z|X)$ which is concentrated on a small number of z, and compute $P(X)$ via that.

We use the KL-divergence, defined as:

$$D_{KL}(Q(z|X)||P(z|X)) = \mathbb{E}_{z\sim Q}[\log Q(z|X) - \log P(z|X)].$$

We can introduce $P(X|z)$ by applying Bayes' rule:

$$D_{KL}(Q(z|X)||P(z|X)) = \mathbb{E}_{z\sim Q}[\log Q(z|X) - \log P(X|z) - \log P(z)] + \log(P(X))$$

and so:

$$\log P(X) - D_{KL}(Q(z|X)||P(z|X)) = \log \mathbb{E}_{z\sim Q}[\log P(X|z)] - D_{KL}(Q(z|X)||P(z))$$

Recall: our goal was to optimizer $P(X)$ across the data set. We can now attempt this by instead maximizing the right hand side of this equation. If we assume that $Q(z|X) \sim N(\mu(X), \Sigma(X))$, and that $P(z) \sim N(0,1)$, the rightmost term has a closed form expression:

$$D_{KL}(Q(z|X)||P(z)) = \frac{1}{2}(\mathrm{tr}(\Sigma) + \mu^T\mu - k - \log\det(\Sigma))$$

The first term can be approximated by sampling a single z_0 randomly from the distribution $Q(z|X)$ and then using the value of $P(X|z_0)$ as the expected value.

In practice, Q and P are both learned through some sort of neural net architecture; depending on the particular application, different architectures might be used. Note that Q acts as an encoder, giving a distribution of z based on X, and P acts as a decoder, giving a distribution of X based on z. In order to generate new data, we simply sample $z \sim N(0,1)$, and compute $P(X|z)$, forgetting about the encoder portion of the framework. Examples of sample handwritten digits generated by a VAE are below. Note that even though they're reasonably good approximations to hand drawn digits, they are not particularly sharp.

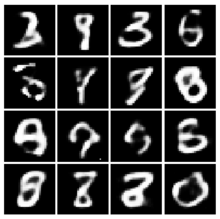

Figure 1: VAE generated data after 2M iterations

Generative Adversarial Networks

The Variational Autoencoder took a Bayesian approach to data generation, with a prior on both the data and latent variable sides. Adversarial networks instead take a game theory approach to data generation. In this framework, there are two players: a Generator G, and a Discriminator D. The Generator takes a random z as input, and outputs an attempt at data, while the Discriminator is given a data-like object as input, and tries to predict the probability that it is an actual datapoint. Mathematically, we have:

$$G : \mathcal{Z} \to \mathcal{X} \text{ and } D : \mathcal{X} \to [0, 1]$$

Our optimization problem then becomes the following:

$$\min_G \max_D V(G, D)$$

where

$$V(G, D) = \mathbb{E}_{\mathcal{X}}[\log(D(x))] + \mathbb{E}_{\mathcal{Z}}[\log(1 - D(G(z)))]$$

Let's look at this cost function for a bit. The first term says that D is trying to maximize the probability that it guesses close to 1 if it is handed a true data point. The second term says that D wants to maximize the probability that it guesses close to 0 if it is handed a fake data point, i.e., one generated by G.

On the other hand, G is trying to minimize that second term, trying as hard as it can to trick D into believing that G is actually generating true data. Typically, due to gradient sizes, G's gradient descent process will actually be structured so as to maximize $\log(D(G(z)))$.

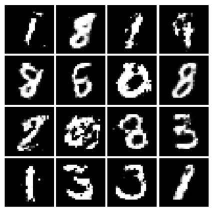

Figure 2: GAN generated data after 700K iterations

Notice how the GAN data are much sharper than the VAE data. I conjecture that this is because of the following: most of the true data has sharper edges, and the discriminator D has developed a way to penalize non-sharp edges. On the other hand, the cost function in a VAE will not penalize any specific aspects of the generated data, only an overall difference.

VAE with GAN costs

We now look at a network which is based off of a VAE, but adds an additional discriminator cost. Our setup is the following:

$$Q : \mathcal{X} \to \mathcal{Z},\ P : \mathcal{Z} \to \mathcal{X}, \text{ and } D : \mathcal{Z} \to [0, 1]$$

Here, D is not a discriminator on data, but instead a discriminator on the latent variable, attempting to discriminate true randomness of the prior on $\mathcal{Z}$ from the distribution induced by Q. Here, the discriminator cost replaces the KL divergence term that appeared in the VAE equation. We still have a reconstruction loss

$$\mathbb{E}_{z \sim Q} \log P(X|z)$$

coming from comparing true datapoints to generated datapoints, but instead of using KL divergence to force Q to approximate a $N(0,1)$ distribution, we instead use the same discriminator/generator costs as in a GAN. This approach produces data which appear similar to data produced by a vanilla VAE.

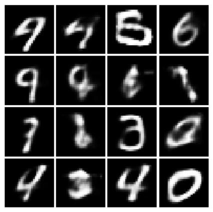

Figure 3: VAE-GAN generated data after 270k iterations

Bidirectional GAN

The last type of Generative Network we look at is a Bidirectional Generative Adversarial Network. This type of network has three different parts: an encoder E, a generator G, and a discriminator D. This type of network differs from a VAE with GAN costs by never actually computing $E \circ G$ or $G \circ E$. They are defined as follows:

$$G : \mathcal{Z} \to \mathcal{X}, \ E : \mathcal{X} \to \mathcal{Z}, \text{ and } D : \mathcal{Z} \times \mathcal{X} \to [0,1]$$

Here, the optimization problem is given by

$$\min_{G,E} \max_{D} V(E,G,D)$$

where

$$V(E,G,D) = \mathbb{E}_{\mathcal{X}}[\log D(x, E(x))] + \mathbb{E}_{\mathcal{Z}}[\log(1 - D(G(z), z))]$$

Again, the discriminator is attempting to discern true datapoints from false datapoints. Data generated by a BiGAN network are below.

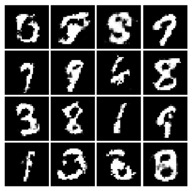

Figure 4: BiGAN generated data after 400k iterations

References

[1] Cark Doersch. Tutorial on Variational Autoencoders. August 16, 2016. https://arxiv.org/pdf/1606.05908.pdf. Accessed May 2, 2018

[2] Diederik P Kingma and Max Wellin. Auto-encoding variational Bayes. *ICLR*, 2014

[3] Goodfellow et al. Generative Adversarial Nets. *NIPS*, 2014

[4] Donahue, Krahenbuhl, and Darrell. Adversarial Feature Learning. *ICLR*, 2017