CS 194-129 Deep Neural Networks
Spring 2018 Grant Lecture 16

1 Reading

Generative Adversarial Nets

We pit a *generative model*, which generates counterfeit data samples, against a *discriminative model*, which determines whether a sample is from the model distribution or the data distribution. At equilibrium, the generative model distribution should be identical to the training data distribution, and the discriminator should output $\frac{1}{2}$ for all inputs.

More specifically, the generator $G(\mathbf{z}; \theta_g)$ maps noise variables $\mathbf{z}$ [drawn from the prior distribution $p_{\mathbf{z}}(\mathbf{z})$] to data space, while the discriminator $D(\mathbf{x}; \theta_d)$ outputs a single scalar representing the probability that $\mathbf{x}$ came from the data rather than from the generator's distribution p_g. D is trained to maximize the probability of assigning the correct label to both training data and synthetic data. G is trained to minimize $\log(1-D(G(\mathbf{z})))$ (i.e. to make the discriminator say that the synthetic data is real).

2 Lecture

For the purposes of this lecture, a **generative model** defines a probability distribution over data whose individual examples have characteristics that are difficult to describe. In the **implicit** case, which includes GANs, the model outputs *samples* from the underlying distribution rather than the probability distribution itself.

Note: an explicit *model maps an input vector to a scalar probability. It learns to do this by following the gradient of the log density with respect to the parameters. However, it can be very difficult to define densities for complex high-dimensional data, and/or intractable to optimize for this using gradients.*

Such imagined data might be useful for, e.g., RL (where we might want to simulate the future conditioned on a given action), semi-supervised learning (perhaps we have a training set which is missing class labels for some of the data; we can leverage generative descriptions of data even when it is unlabeled), and applications which simply *require* data generation.

We can train generative models according to a maximum likelihood perspective. For example, we can assign a probability density to the training data and find a parameter vector that maximizes the log probability of the entire dataset.

Other Generative Models

Fully Visible Belief Nets

A fully visible belief net is an explicit approach (it represents the probability distribution directly) which uses the chain rule to write a probability for a high-dimensional vector as a product of simpler probabilities, which can be evaluated simultaneously if the entire vector is visible up to that point.

$$p_{\text{model}}(\mathbf{x}) = p_{\text{model}}(x_1) \prod_{i=2}^{n} p_{\text{model}}(x_i \mid x_1, ..., x_{i-1})$$

For example, an n-gram model might generate a token as a function of the $(n-1)$ preceding tokens, and base probabilities off of the frequencies observed in the training set.

One disadvantage of a fully visible belief net is its $O(n)$ sample generation time. The WaveNet TTS model has very good sample quality, but cannot be used in real-time contexts because it is a fully visible belief net.

Change of Variables

In this approach, we have a function g mapping a simple probability distribution (e.g. a Gaussian) to another probability distribution.

$$y = g(\mathbf{x}) \Rightarrow p_{\mathbf{x}}(\mathbf{x}) = p_y(g(\mathbf{x})) \left| \det \left(\frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$

The determinant of the Jacobian of the transformation describes how the density changes as we move from the first space to the second space. Basically, we're mapping a simple distribution (like a Gaussian) to a high-dimensional space (like that of natural images).

A disadvantage: the formulation requires the transformation to be invertible, which forces the latent space to have the same dimension as the observed space and restricts the design of the architecture.

Variational Autoencoder

VAEs are very good at estimating densities, but generate lower quality samples. Accordingly, it's often good to introduce adversarial costs (because adversarial methods can generate crisp, high quality samples).

Boltzmann Machines

Boltzmann machines are explicit density models. They are not scalable, since they use Markov chains (which have not been able to get accurate results in high dimensions) to estimate the partition function.

GANs

GANs are designed to avoid some of the problems with the other generative models.

- They involve a latent code which can be useful for learning something about the structure of the data.
- They are asymptotically consistent (unlike VAEs). Given perfect optimization and sufficient training data, they will perfectly recover the data distribution we're trying to learn.
- They require no Markov chains.
- They produce some of the most plausible samples for a variety of applications.

In the GAN framework, our primary goal is to learn a generator $G(\mathbf{z})$ (an implicit model that transforms random noise $\mathbf{z}$ into samples). The latent code $\mathbf{z}$ can be drawn from any distribution, e.g. a Gaussian or uniform distribution. The discriminator $D(\mathbf{x})$ is used as part of optimization for the generator, and tries to output 1 for $\mathbf{x}$ sampled from actual data and 0 for $\mathbf{x}$ sampled from the model.

D's aim is to be a good binary classifier for real and fake data; as a result, it can simply utilize cross-entropy loss. G wants to worsen the situation for the discriminator, so its objective can be the negative of D's objective. *If the generator and discriminator have enough capacity, they will work their way to a Nash equilibrium in which G produces samples that come from the same distribution as the data, and D will assign every sample a* $\frac{1}{2}$ *probability of being real.*

Generator Network

The generator network $\mathbf{x} = G(\mathbf{z}; \theta^{(G)})$ must be differentiable (because we're using gradients to optimize) but not necessarily invertible. It is trainable for any size of $\mathbf{z}$, although some guarantees require $\mathbf{z}$ to have a higher dimension than $\mathbf{x}$.

Analysis

The simplest way to theoretically analyze GANs is to set up a minimax game (i.e. a zero sum two-player game).

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] - \frac{1}{2}\mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$
$$J^{(G)} = -J^{(D)}$$

The discriminator's cost is $J^{(D)}$; the generator's cost is $J^{(G)}$. The equilibrium of the game is at a saddle point of the discriminator loss (a local maximum w.r.t. one player, and a local minimum w.r.t. the other player).

Potential issue: the discriminator might begin to label the data correctly early on, causing its cost to saturate close to 0 (s.t. the gradients are very small). This is good for D; it doesn't need to learn any more. But this is not good overall, because our main goal was to learn an implicit *generative* model rather than just a discriminator.

(In the above case: if G's cost is just the negative of D's cost, the gradient is just going to be too small for it to learn anything.) To solve the saturation problem, we can introduce a game that has a separate cost for each player:

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] - \frac{1}{2}\mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$
$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\mathbf{z}}[\log D(g(\mathbf{z}))]$$

Now the generator tries to maximize the log probability of the discriminator making a mistake. The advantage of this setup is that gradients will always be large if there is more optimization to be done.

Analysis: Maximum Likelihood

We can also change the formulation to maximize the likelihood (log probability) of the data. This is equivalent to minimizing the KL divergence between the data and the model.

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\mathbf{z}}[\exp(\sigma^{-1}(D(g(\mathbf{z}))))]$$

$J^{(D)}$ remains the same. When D is optimal, G's gradient matches that of maximum likelihood.

Note: this should not be used in practice, since the formulation does not produce samples of the same quality as before (where we were minimizing the Jensen-Shannon divergence). The maximum likelihood approach tends to blur different modes of the data distribution so that it can assign high probability to all of the data. By contrast, JSD chooses one mode of the data distribution and then moves the probability mass of the model to that mode.

Training Procedure

To train a GAN, we can use an SGD-like algorithm on two minibatches simultaneously: (1) a minibatch of training examples, and (2) a minibatch of generated samples. Optionally (e.g. if one player is of a higher capacity than the other), we can run k steps of one player for every step of the other player.

Examples

The **DCGAN** (deep convolutional GAN) architecture allows a generator to produce an image in a single shot, without relying on any sequential procedure. Batch normalization is critical to its success.

Progressive GANs (2017) use a sequential generation procedure, and have been shown to produce very realistic people images after training on the CelebA dataset.

Note: it's easier to generate a very specific thing from a dataset than to generate "anything at all" from the dataset. GANs work best when the output entropy is low.

Conclusion

- GANs are generative models that use a supervised learning problem (binary classification) to approximate an intractable cost function.
- The GAN framework can be used to simulate cost functions such as that of maximum likelihood (as opposed to a more standard Jensen-Shannon divergence, which comes from the minimax objective).

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Nets. In NIPS, 2014.