Generative adversarial networks

- Why GANs?
- GAN intuition
- GAN equilibrium
- GAN implementation
- Practical considerations

Much of these notes are based on the tutorial by Goodfellow, NIPS 2016.

Beyond VAEs for generative models

In the VAE lecture, we discussed how VAEs are generative models and could thus generate data that bore some resemblance to the input data. However, the VAEs had some cons. While there are a whole host of generative models, we'll focus on one particular one: the generative adversarial network (GAN). The GAN is nice for many reasons, some of which are listed here:

- The design of the generator has very few restrictions. We could make it a CNN easily, or an LSTM, etc.
- The GAN doesn't require a variational lower bound.
- GANs are asymptotically consistent.
- GANs are subjectively better at generating samples.

GAN intuition

The GAN can be seen as a game between two players, the **generator** and the **discriminator**.

- **Generator**: the generator is the generative model, and performs a maping $\hat{\mathbf{x}} = G(\mathbf{z})$, where $\mathbf{z}$ is some random noise. Its goal is to produce samples, $\hat{\mathbf{x}}$, from the distribution of the training data $p(\mathbf{x})$.
- **Discriminator**: the discriminator is the generator's opponent, and performs a mapping $D(\mathbf{x}) \in (0, 1)$. Its goal is to look at sample images (that could be real or synthetic from the generator), and determine if they are real samples ($D(\mathbf{x})$ closer to 1) or synthetic samples from the generator ($D(\mathbf{x})$ closer to 0). $D(\mathbf{x})$ can be interpreted as the probability that the image is a real training example.

The generator is trained to fool the discriminator, and thus they can be viewed as adversaries. To succeed in the game, the generator must make samples that are indistinguishable to the discriminator.

Because of its setting as a game, these networks are called "adversarial" but they could also be viewed as "cooperative." Typically, a deep neural network is used to represent both the generator and the discriminator.

GAN intuition (cont.)

The generator, $G(\mathbf{z})$, has parameters $\theta^{(G)}$, and the discriminator, $D(\mathbf{x})$, has parameters $\theta^{(D)}$. The generator can only control $\theta^{(G)}$, while the discriminator can only control $\theta^{(D)}$.

In addition to this, the discriminator and generator have different cost functions they wish to optimize.

- This ought to be intuitive as the generator and discriminator have different goals.
- We denote the discriminator's cost function as $\mathcal{L}^{(D)}(\theta^{(D)}, \theta^{(G)})$. For convenience, we will sometimes denote this as $\mathcal{L}^{(D)}$.
- We denote the generator's cost function as $\mathcal{L}^{(G)}(\theta^{(D)}, \theta^{(G)})$. For convenience, we will sometimes denote this as $\mathcal{L}^{(G)}$.

GAN intuition (cont.)

What is the solution?

- The discriminator wishes to minimize $\mathcal{L}^{(D)}$, but can only do so by changing $\theta^{(D)}$.
- The generator wishes to minimize $\mathcal{L}^{(G)}$, but can only do so by changing $\theta^{(G)}$.

This is slightly different from the optimization problems we've described thus far, where we have one set of parameters to minimize one cost function $\mathcal{L}$.

GAN Nash equilibrium

Instead of treating this as an optimization problem, we treat this as a *game* between two players. The solution to a game is called a Nash equilibrium. For GANs, a Nash equilibrium is a tuple, $(\theta^{(D)}, \theta^{(G)})$ that is:

- a local minimum of $\mathcal{L}^{(D)}$ with respect to $\theta^{(D)}$
- a local minimum of $\mathcal{L}^{(G)}$ with respect to $\theta^{(G)}$.

If $G(\mathbf{z})$ and $D(\mathbf{x})$ have sufficient capacity (e.g., if they are neural networks), then the Nash equilibrium corresponds to:

- The generator draws samples from $p(\mathbf{x})$, i.e., the distribution of the data.
- The discriminator cannot discriminate between them, i.e., $D(\mathbf{x}) = \frac{1}{2}$ for all $\mathbf{x}$.

In this case, the generator is a perfect generative model, sampling from $p(\mathbf{x})$.

Discriminator cost, $\mathcal{L}^{(D)}$

Here we define the discriminator's loss function. We'll let $p_{\text{data}}(\mathbf{x})$ denote the data distribution (which we have prior denoted as $p(\mathbf{x})$) and we let $p_{\text{model}}(\mathbf{x})$ denote the distribution of samples from the generator. Then, the discriminator's cost is:

$$\begin{aligned}\mathcal{L}^{(D)}(\theta^{(D)}, \theta^{(G)}) &= -\frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}} \log D(\mathbf{x}) - \frac{1}{2}\mathbb{E}_{\hat{\mathbf{x}}\sim p_{\text{model}}} \log(1 - D(\hat{\mathbf{x}})) \\ &= -\frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}} \log D(\mathbf{x}) - \frac{1}{2}\mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z})))\end{aligned}$$

The estimator gets to change $\theta^{(D)}$ to optimize this quantity.

This cost is fairly intuitive:

- The loss will be zero if $D(\mathbf{x}) = 1$ for all $\mathbf{x} \sim p_{\text{data}}$ and $D(\hat{\mathbf{x}}) = 0$ for all $\hat{\mathbf{x}} \sim p_{\text{model}}$, i.e., generated via $\hat{\mathbf{x}} = G(\mathbf{z})$.
- This is the same as the cross-entropy loss for a neural network doing binary classification with a sigmoid output.

Discriminator's optimal strategy

The goal of the discriminator is to minimize

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

For the sake of intuition, consider that instead of optimizing w.r.t. $\theta^{(D)}$, we get to optimize $D(\mathbf{x})$ for every value of $\mathbf{x}$. Further, assume that p_{data} and p_{model} are nonzero everywhere. What would be the optimal strategy for D?

To answer this question, we differentiate w.r.t. $D(\mathbf{x})$ and set the derivative equal to zero. In particular, we start with:

$$\mathcal{L}^{(D)} = -\frac{1}{2}\int p_{\text{data}}(\mathbf{x}) \log D(\mathbf{x}) d\mathbf{x} - \frac{1}{2}\int p_{\text{model}}(\mathbf{x}) \log(1 - D(\mathbf{x})) d\mathbf{x}$$

Differentiating w.r.t. $D(\mathbf{x})$, we get:

$$\frac{\mathcal{L}^{(D)}}{dD(\mathbf{x})} = -\frac{1}{2}\int p_{\text{data}}(\mathbf{x}) \frac{1}{D(\mathbf{x})} d\mathbf{x} + \frac{1}{2}\int p_{\text{model}}(\mathbf{x}) \frac{1}{(1 - D(\mathbf{x}))} d\mathbf{x}$$

Discriminator's optimal strategy (cont.)

We can now set the derivative equal to zero to find the optimal $D(\mathbf{x})$. This leads to:

$$\frac{1}{2}\int p_{\text{data}}(\mathbf{x})\frac{1}{D(\mathbf{x})}d\mathbf{x} = \frac{1}{2}\int p_{\text{model}}(\mathbf{x})\frac{1}{(1-D(\mathbf{x}))}d\mathbf{x}$$

A solution occurs when the integrands are equal for all $\mathbf{x}$, i.e.,

$$p_{\text{data}}(\mathbf{x})\frac{1}{D(\mathbf{x})} = p_{\text{model}}(\mathbf{x})\frac{1}{(1-D(\mathbf{x}))}$$

Rearranging terms, this gives:

$$D(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{model}}(\mathbf{x}) + p_{\text{data}}(\mathbf{x})}$$

Discriminator's optimal strategy (cont.)

Intuitively, we see that the optimal discriminator strategy is akin to:

- Accept an input, $\mathbf{x}$.
- Evaluate its probability under the distribution of the data, $p_{\text{data}}(\mathbf{x})$. Note, the discriminator doesn't have access to $p_{\text{data}}(\mathbf{x})$, but learns it through the training process.
- Evaluate its probability under the generator's distribution of the data, $p_{\text{model}}(\mathbf{x})$. The same note applies: it doesn't have access to $p_{\text{model}}(\mathbf{x})$ but learns it through the training process.

If the discriminator has high enough capacity, it can reach its optimum. If the generator has high enough capacity, it will then move to set: $p_{\text{model}}(\mathbf{x}) = p_{\text{data}}(\mathbf{x})$ for all $\mathbf{x}$. This results in the output $D(\mathbf{x}) = 1/2$. This is the Nash equilibrium.

Discriminator's optimal strategy (cont.)

In the VAE, we saw that to generate data, we had to learn some $p_{\text{model}}(\mathbf{x})$. In GANs, the fact that the discriminator learns an approximation of the ratio $p_{\text{data}}(\mathbf{x})/p_{\text{model}}(\mathbf{x})$ is the key insight that allows it to work and makes it distinctly different from other generative models that learn $p_{\text{model}}(\mathbf{x})$ directly (or indirectly via latent variable models).

Because the discriminator learns this, we can now judge how good samples from the generator are.

Zero-sum game

The simplest type of game to analyze is a *zero-sum* game, in which the sum of the generator's loss and the discriminator's loss is always zero. In a zero-sum game, the generator's loss is:

$$\mathcal{L}^{(G)} = -\mathcal{L}^{(D)}$$

The solution for a zero-sum game is called a minimax solution, where the goal is to minimize the maximum loss. Since the game is zero-sum, we can summarize the entire game by stating that the loss function is $\mathcal{L}^{(G)}$ (which is the discriminator's payoff), so that the minimax objective is:

$$\min_{\theta^{(G)}} \max_{\theta^{(D)}} \left[\frac{1}{2} \mathbb{E}_{x \sim p_{\text{data}}} \log D(\mathbf{x}) + \frac{1}{2} \mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z}))) \right]$$

- The discriminator wants to maximize the objective (i.e., its payoff) such that $D(\mathbf{x})$ is close to 1 and $D(G(\mathbf{z}))$ is close to zero.
- The generator wants to minimize the objective (i.e., its loss) so that $D(G(\mathbf{z}))$ is close to 1.

Optimization

For GANs, in practice this game is implemented in an iterative numerical approach. It involves two steps:

- **Gradient ascent for the discriminator**. We modify $\theta^{(D)}$ to maximize the minimax objective.

$$\theta^{(D)} \leftarrow \arg\max_{\theta^{(D)}} \left[\frac{1}{2} \mathbb{E}_{x \sim p_{\text{data}}} \log D(\mathbf{x}) + \frac{1}{2} \mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z}))) \right]$$

- **Gradient descent on the discriminator**. We modify $\theta^{(G)}$ to minimize the minimax objective.

$$\theta^{(G)} \leftarrow \arg\min_{\theta^{(G)}} \frac{1}{2} \mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z})))$$

Details of training

GAN training looks like the following. For some number of training iterations:

- Take k gradient steps for the discriminator (k a hyperparameter), each doing the following:
  - Sample m noise samples, $\{\mathbf{z}^{(1)}, \mathbf{z}^2, \ldots, \mathbf{z}^{(m)}\}$ from a noise prior, $p(\mathbf{z})$.
  - Sample m actual samples, $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}\}$ from $p_{\text{data}}(\mathbf{x})$. (That is, this is a minibatch of your input data.)
  - Perform a SGD step (e.g., with Adam, or your favorite optimizer; note we are also omitting the $1/2$'s since they're just a scale factor):

$$\theta^{(D)} \leftarrow \theta^{(D)} + \varepsilon \nabla_{\theta^{(D)}} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)}))) \right]$$

- Do a gradient descent step for the generator:
  - Sample a minibatch of m noise samples, $\{\mathbf{z}^{(1)}, \mathbf{z}^2, \ldots, \mathbf{z}^{(m)}\}$ from a noise prior, $p(\mathbf{z})$.
  - Perform a SGD step:

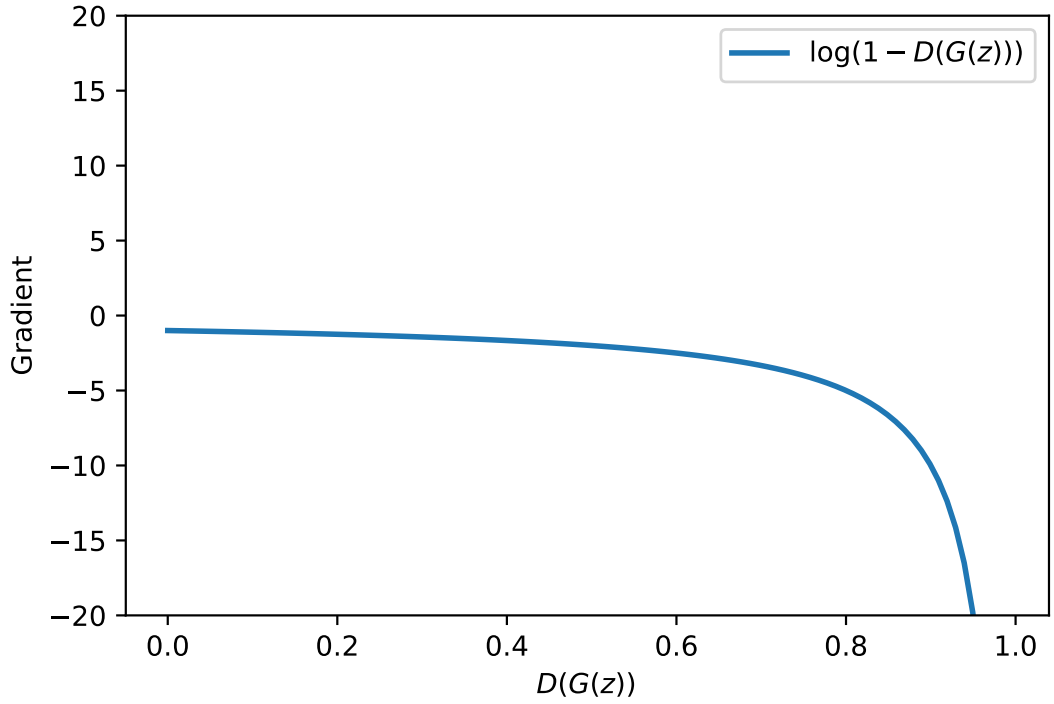
$$\theta^{(G)} \leftarrow \theta^{(G)} - \varepsilon \nabla_{\theta^{(G)}} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(\mathbf{z}^{(i)})))$$

Details of training (cont.)

Goodfellow states, regarding the hyperparameter k, "Many authors recommend running more steps of one player than the other, but as of late 2016, the author's opinion is that the protocol that works best in practice is ... one step for each player."

An important problem with the stated training paradigm

As stated, the current training paradigm has an important limitation. Note that the gradient of $\log(1 - D(G(\mathbf{z})))$ is $-\frac{1}{1-D(G(\mathbf{z}))}$, and thus has the following gradient as a function of $D(G(\mathbf{z}))$:



An important problem with the stated training paradigm (cont.)

Therefore, if $D(G(\mathbf{z})) \approx 0$, as may happen early on in training when the discriminator can tell the difference between real and synthetic examples, the gradient is close to zero. This results in little learning for $\theta^{(G)}$, and thus in practice the generator cost function:

$$\mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z}^{(i)})))$$

is rarely ever used.

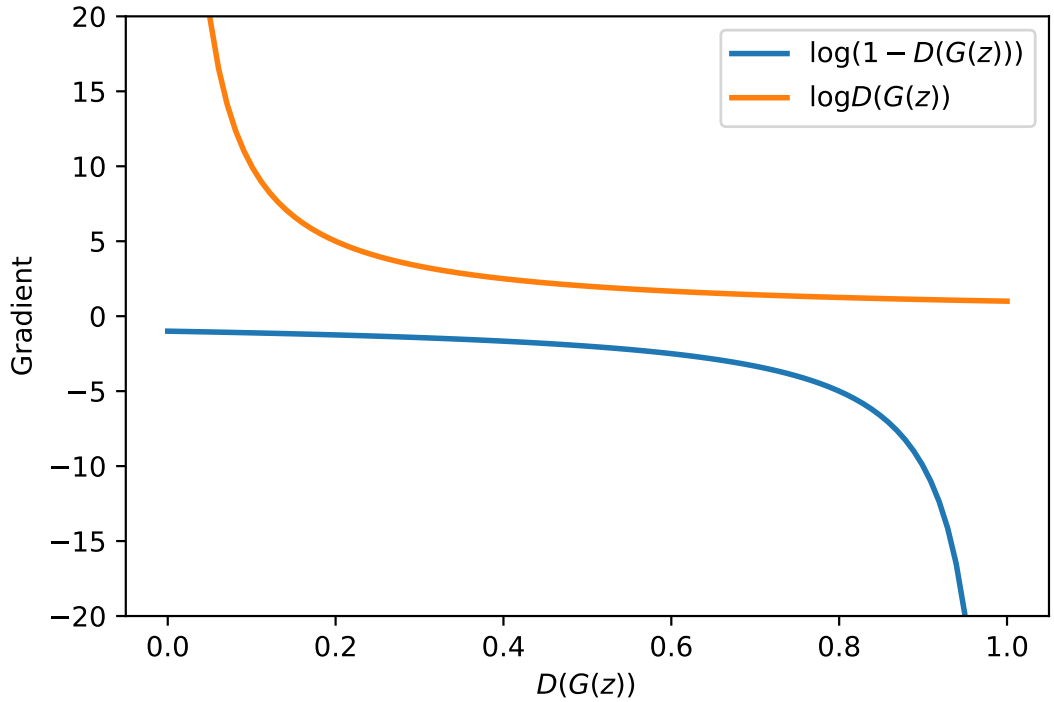
Instead, we opt for a cost function that has a large gradient when $D(G(\mathbf{z})) \approx 0$, so that the generator is encouraged to learn much more early in training. This is the cost function:

$$-\mathbb{E}_{\mathbf{z}} \log(D(G(\mathbf{z})))$$

This still obtains the same overall goal of being minimized when $D(G(\mathbf{z})) = 1$, but now admits far more learning when the generator performs poorly.

Better gradients when the generator performs poorly

The gradient of this new cost for the generator encourages more learning when the generator performs poorly.



New optimization for the vanilla GAN

For the sake of completeness, we repeat the GAN training procedure but with the updated objective for the generator. For some number of training iterations:

- Take k gradient steps for the discriminator (k a hyperparameter), each doing the following:
  - Sample m noise samples, $\{\mathbf{z}^{(1)}, \mathbf{z}^2, \ldots, \mathbf{z}^{(m)}\}$ from a noise prior, $p(\mathbf{z})$.
  - Sample m actual samples, $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}\}$ from $p_{\text{data}}(\mathbf{x})$. (That is, this is a minibatch of your input data.)
  - Perform a SGD step (e.g., with Adam, or your favorite optimizer; note we are also omitting the $1/2$'s since they're just a scale factor):

$$\theta^{(D)} \leftarrow \theta^{(D)} + \varepsilon \nabla_{\theta^{(D)}} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)}))) \right]$$

- Do a gradient descent step for the generator:
  - Sample a minibatch of m noise samples, $\{\mathbf{z}^{(1)}, \mathbf{z}^2, \ldots, \mathbf{z}^{(m)}\}$ from a noise prior, $p(\mathbf{z})$.
  - Perform a SGD step:

$$\theta^{(G)} \leftarrow \theta^{(G)} + \varepsilon \nabla_{\theta^{(G)}} \frac{1}{m} \sum_{i=1}^{m} \log D(G(\mathbf{z}^{(i)}))$$

No longer a zero-sum game

Note that by changing the cost function for the generator network, the game is no longer zero-sum. This is a heuristic change made to the game to solve the practical problem of saturating gradients when the generator isn't doing well.

Tips for training GANs

The following tips come from both Goodfellow's 2016 NIPS tutorial as well as Soumith's github repo (`https://github.com/soumith/ganhacks`).

- **Train with labels**. Instead of telling the discriminator whether the images are real or fake, provide the labels of the image to the discriminator. This involves having the true labels of the images, as well as generating labels for the fake images. The GAN discriminator, instead of outputting $D(\mathbf{x})$ being the probability of $\mathbf{x}$ being real or fake, will now instead output the probability of the class. One example is called an auxiliary classifier GAN (Odenta et al., 2016). As per Goodfellow, 2016, "It is not entirely clear why this trick works."
- **One-sided label smoothing.** When training by providing the discriminator with images being real or fake, instead of providing 1's and 0's, do label smoothing, provide $1-\alpha$'s and 0's. It's important that the probability of the fake sample stay at 0. α is a hyperparameter. This has a regularizing effect, not letting the scores grow very large (w.p. approaching 1).

Tips for training GANs (cont.)

- **Virtual batch normalization**. GAN training improves with batch normalization, but if batches are too small, GAN training worsens. Virtual batch normalization defines a separate reference batch prior to training, and batch normalizes by calculating the mean and variance statistics over the union of the minibatch and the reference batch.
- **Other good practices**, largely empirical.
  - Normalize the images between -1 and 1, and use $\tanh$ as the output of the generator model.
  - Let the prior on $\mathbf{z}$ be Gaussian rather than uniform.
  - Avoid sparse gradients; don't use ReLU or maxpool. Use LeakyReLU and to downsample, increase the stride.
  - Use Adam.
  - Use dropout in $G()$.