Improved Techniques for Training GANs

1 Citation

Salimans, Tim, et al. "Improved techniques for training gans." Advances in Neural Information Processing Systems. 2016.

`http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf`

2 Abstract

We present techniques to make GANs easier to train. We get state of the art results on semi-supervised classification on MNIST, SVHN, and CIFAR-10. We also found that humans struggle to tell generated examples apart from real ones (21.4% accuracy on CIFAR-10 images).

3 Introduction

In Generative Adversarial Network (GAN), a generator network takes a noise vector and tries to create a realistic looking example (e.g. an image). A discriminator tries to identify whether a given input example belongs to the training set or was generated by the generator. By making these two networks compete against each other, we reach a Nash equilibrium where the generator is able to generate realistic examples. In practice, however, training the GAN to a Nash equilibrium is hard, so we describe improvements to GANs to make convergence easier.

4 Related Work

Our work is inspired by DCGANs, batch normalization, and work that uses GANs for semi-supervised learning.

5 Toward Convergent GAN Training

The discriminator aims to maximize $J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ w.r.t. $\boldsymbol{\theta}^{(D)}$ while the generator aims to minimize $J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ w.r.t. $\boldsymbol{\theta}^{(G)}$. A Nash equilibrium occurs when each player has minimized their cost function assuming the other player's parameters are fixed.

Our first improvement, called **feature matching**, pushes the generator to match the statistics of real data rather than the discriminator output. To do this, we let $\boldsymbol{f}(\boldsymbol{x})$ be an intermediate layer of the discriminator. Our generator's objective function then becomes:

$$||\mathbb{E}_{\boldsymbol{x}\sim p_{data}} \boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})} \boldsymbol{f}(G(\boldsymbol{z}))||_2^2 \tag{1}$$

Generators have a bad habit of collapsing $\boldsymbol{z}$ vectors into a single example. To fix this, we use **minibatch discrimination**, which is when we allow the discriminator to look at a minibatch of examples (this way, it can tell if the generator keeps producing the same point). Let $\boldsymbol{f}(\boldsymbol{x}_i) \in \mathbb{R}^A$ be a vector of activations from the discriminator. We multiply this by tensor $T \in \mathbb{R}^{A \times B \times C}$, which results in $M_i \in \mathbb{R}^{B \times C}$. Then, for an n-example minibatch ($i \in \{1...n\}$), we compute $c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-||M_{i,b} - M_{j,b}||_{L_1}) \in \mathbb{R}$. We then compute similarity to other examples as follows:

$$o(\boldsymbol{x}_i)_b = \sum_{j=1}^{n} c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathbb{R} \tag{2}$$

$$o(\boldsymbol{x}_i) = [o(\boldsymbol{x}_i)_1, o(\boldsymbol{x}_i)_2, ..., o(\boldsymbol{x}_i)_B] \in \mathbb{R}^B \tag{3}$$

We then concatenate $o(\boldsymbol{x}_i)$ with $\boldsymbol{f}(\boldsymbol{x}_i)$ and feed it into the next layer of the discriminator. The discriminator still produces a label for each example, but it considers a whole minibatch when making its descision. Minibatch discrimination generates realistic examples more quickly than feature matching, but the latter seems to work better when you want to make a semi-supervised classifier.

We also apply **historical averaging**, where we update each player's cost to include $||\boldsymbol{\theta} - \frac{1}{t}\sum_{i=1}^{t} \boldsymbol{\theta}[i]||^2$, where $\boldsymbol{\theta}[i]$ is the parameter vector at step i.

We also apply **one-sided label smoothing**, where we replace the 1 label with α and 0 label with β. We set $\beta = 0$ so that the model does not do ridiculous things when it is in an area of space where p_{data} is low.

We also apply **virtual batch normalization**, where each example is normalized based on the activations of a reference minibatch. This requires processing the reference minibatch for each training minibatch, which is compute-intensive, so we only apply this to the generator.

6 Assessment of Image Quality

We use Amazon Mechanical Turk workers get humans to discriminate between real and generated data. One downside is that human annotators change their approach (they become more discerning) when you give them feedback. So, as an alternative, we apply the Inception model to get the label distribution for each image. We want to make sure our generator generates diverse labels and that each image's label distribution (as computed by Inception) is low-entropy. Our metric is $\exp(\mathbb{E}_{\boldsymbol{x}}[KL(p(y|\boldsymbol{x})||p(y))])$

7 Semi-supervised Learning

To improve a classifier, we add a new class (so we have $K+1$ classes now) called the *generated* class. The probability of this class is $1 - D(\boldsymbol{x})$ for all input images in our dataset. We also use the generator to generate many training examples (about half our dataset is generated examples). Our loss is then

$$\begin{aligned}
L &= -\mathbb{E}_{\boldsymbol{x},y \sim p_{data}(\boldsymbol{x},y)}[\log p_{model}(y|\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim G}[\log p_{model}(y = K+1|\boldsymbol{x})] && (4)\\
&= L_{supervised} + L_{unsupervised} && (5)\\
L_{supervised} &= -\mathbb{E}_{\boldsymbol{x},y \sim p_{data}(\boldsymbol{x},y)}[\log p_{model}(y|x, y < K+1)] && (6)\\
L_{unsupervised} &= -\{\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log(1 - p_{model}(y = K+1|\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x}\sim G}[\log(p_{model}(y = K+1|\boldsymbol{x}))]\} && (7)\\
&= -\{\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim noise}[\log(1 - D(G(\boldsymbol{z})))]\} && (8)
\end{aligned}$$

For the above loss function, we train G with feature matching (minibatch discrimination does not work at all).

8 Experiments

For MNIST, feature matching does not produce visually appealing results, but it does work better for semi-supervised learning.

For CIFAR-10, we find that we can do better than Mechanical Turk workers. We find that using the Inception model to pick the best generated images makes it harder for Mechanical Turk workers to distinguish between real and generated images.

For ImageNet, our GAN generates furry blobs.

9 Conclusion

We propose techniques to stabilize GAN training. More theoretical justification for what we did here is needed.