E0 306: Deep Learning: Theory and Practice 21st March 2019

Scribe Notes for Lecture 19 Generative Adversarial Networks

Lecturer: Amit Deshpande *Scribe*: Julian D'Costa

1 Summary

In this class we explored generative adversarial networks, a method to learn probability distributions over a sample space (very generally, eg a set of images can be considered a distribution over the space of pixel values). GANs were introduced by Ian Goodfellow et al in 2014. [1]

They propose a framework for estimating generative models via an adversarial process, in which they simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. We prove that this framework, seen as a two-player game, has the unique solution where G learns the distribution and D guesses 1/2 everywhere. If implemented via multilayer perceptrons, the entire system can be trained with backpropagation. We look at some of their experimental results and the advantages and disadvantages of GANs.

2 Data

Consider the example of MNIST data, which can be thought of as points in a d = 28 * 28 dimensional bounded box. However, the inherent structure of these images (as opposed to random noise), means that they do not fill the entire box.
Manifold Hypothesis: The set of data points that we have lies on a manifold in the data space that has significantly lower dimension that the data space itself. A manifold can be thought of as a surface that looks locally like $\mathbb{R}^k$. So a low dimenional manifold would look like a twisted and distorted subspace of the data space.

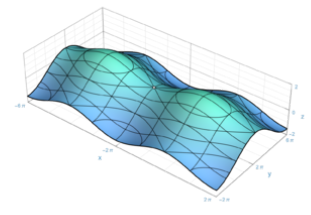


Figure 1: A low dimensional manifold

We make the assumption that there is a simple model that describes the data.

For example, a deep neural network. Take a random seed $z \in \mathbb{R}^k$ drawn from (say) a zero-mean Gaussian. Use the components of z as input to a multilayer neural network $f : \mathbb{R}^k \to \mathbb{R}^d$. We hypothesize that most naturally occuring data can be described as the output of such a neural network applied to a Gaussian. Note that while $x = f(z)$ lies in a d dimensional space, it has only k degrees of freedom. We hope that our data, despite being d-dimensional, has only around k degrees of freedom.

3 GANs

While there are several ways we could use the lower dimensionality of the data to build a generative model, such as Gaussian mixture models, the idea of Generative Adversarial Networks, is to use two different networks, called the Discriminator and the Generator. The discriminator tries to learn to tell apart "fake" examples generated by the Generator and real examples from the input distribution. The Generator tries to fool the Discriminator into getting it wrong. With enough capacity in each network, and appropriate training, this game converges to an equilibrium where $p_g = p_{data}$, ie the Generator perfectly represents the data, and the discriminator assigns P(x came from data) = 1/2 for every input.

3.1 Formal Definition

To learn the generators distribution p_g over data x, we define a prior on input noise variables $p_z(z)$, then represent a mapping to data space as $G(z; \theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters $_g$. We also define a second multilayer perceptron $D(x; d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g . We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $\log(1 - D(G(z)))$:
In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data(x)}}[\log D(x)] + \mathbb{E}_{z \sim p_z(x)}[\log(1 - D(G(z)))]$$

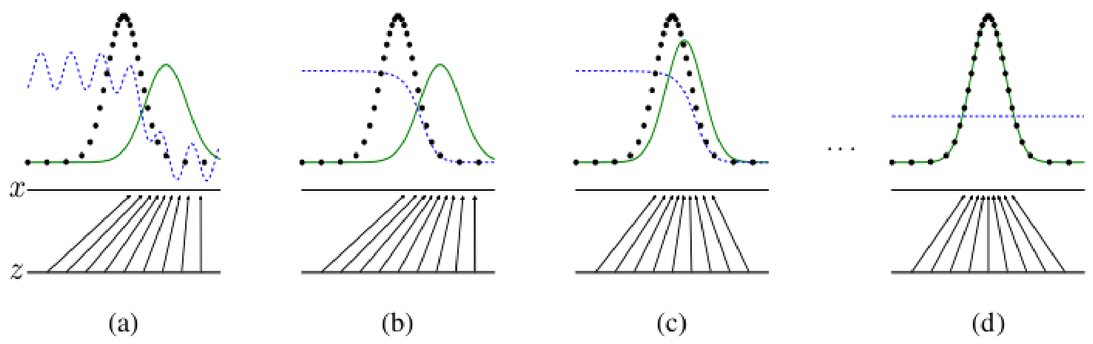


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

Figure 2: The structure of GAN training [1]

Now we prove the claim that the given loss function (from the Discriminator's point of view) is optimized by the Discriminator learning the Bayes-optimal prediction, namely to predict

$$P(x \text{ came from the data}) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}.$$

Proof: The discriminator tries to maximise

$$\begin{aligned} V(G, D) &= \int_x p_{\text{ data }}(x) \log(D(x)) dx + \int_z p_z(z) \log(1 - D(g(z))) dz \\ &= \int_x p_{\text{ data }}(x) \log(D(x)) + p_g(x) \log(1 - D(x)) dx \end{aligned}$$

For any $(a, b) \in \mathbb{R}^2 \backslash \{0, 0\}$, the function $y \to a \log(y) + b \log(1 - y)$ achieves its maximum in $[0, 1]$ at $\frac{a}{a+b}$, as can be checked by differentation. Hence the optimal prediction is

$$D^*(\boldsymbol{x}) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}.$$

Substituting this optimal prediction in, we can reformulate the game in terms of the generator only as

$$C(G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right] + \mathbb{E}_{\boldsymbol{x} \sim p_g} \left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{ data }}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right]$$

We now examine the minimum of $C(G)$. Observe that by adding and subtracting $\log 2$ from each term, we can get the form

$$C(G) = -\log 4 + \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{(p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x}))/2}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{(p_{\text{ data }}(\boldsymbol{x}) + p_g(\boldsymbol{x}))/2}\right],$$

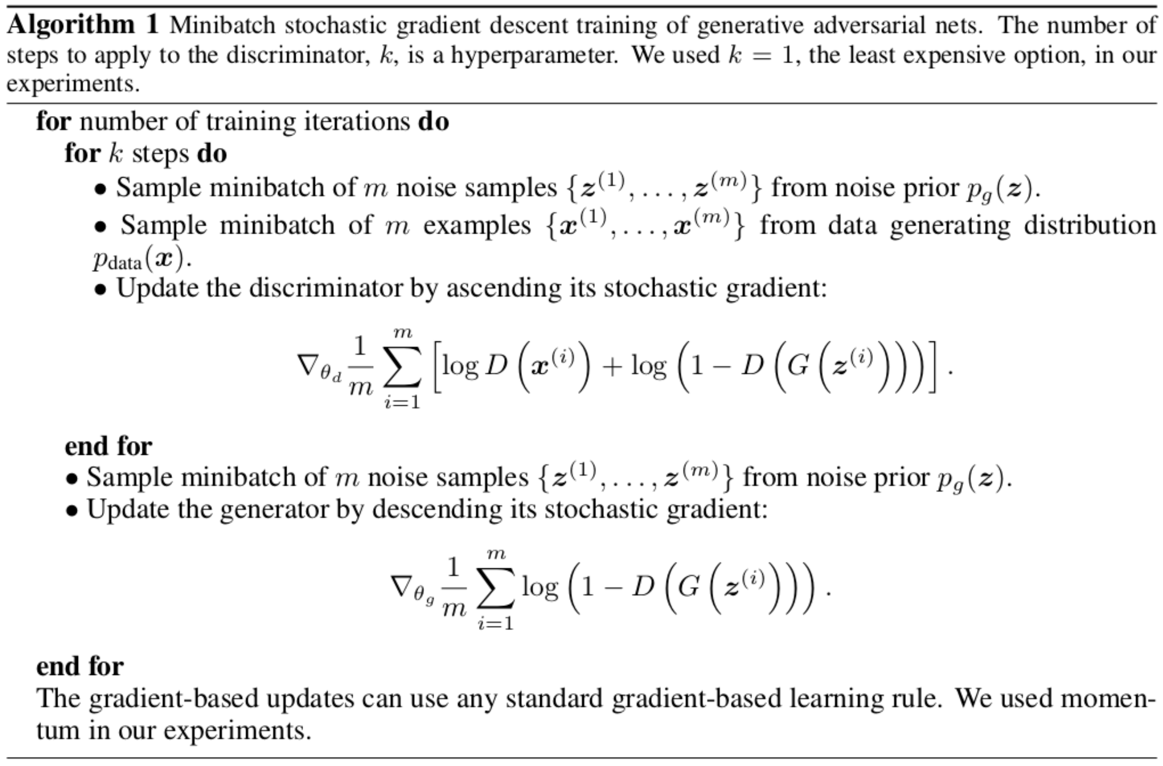
which is

$$C(G) = -\log(4) + KL\left(p_{\text{ data }} \| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \| \frac{p_{\text{ data }} + p_g}{2}\right)$$

where KL is the Kullback-Liebler divergence. This is the Jensen-Shannon divergence,

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{ data }} \| p_g\right).$$

Since the divergence of two ditributions is non-negative, and zero only if the distributions are the same, the only way the generator can minimize $C(G)$ is by perfectly replicating the data distribution, ie $p_g = p_{data}$.
We have thus proved the claim that the only solution of this game is $p_g = p_{data}$ and $D^*(x) = \frac{1}{2}$ for all x.
We can try to use SGD to reach this optimum.



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:
        $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$
  end for
  • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:
      $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$
end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 3: The training algorithm [1]

But does Algorithm 1 converge?

The GAN paper presents a proof of convergence as follows:

Proposition: If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p g is updated so as to improve the criterion

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log D_G^*(\boldsymbol{x})\right]+\mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log\left(1-D_G^*(\boldsymbol{x})\right)\right]$$

then p_g converges to p_{data}.

Proof: Consider $V(G, D) = U(p_g, D)$ as a function of p_g as done in the above criterion. Note that $U(p_g, D)$ is convex in p_g.
According to Danskin's theorem, the subderivatives of a supremum of convex functions include the derivative of the function at the point where the maximum is attained. In other words, if we wish to calculate the derivative of $\max_D U(p_g, D)$ with respect to p_g, we can use the gradient of $U(p_g, D)$ at an optimal D, which is to say, $U(p_g, D^*)$.
Now $\max_D U(p_g, D) = C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right)$ is convex in p_g with a unique global optimum as we proved earlier, therefore with sufficiently small updates of p_g , p_g converges to p_{data} , concluding the proof.

Unfortunately this theorem is valid only if we do manage to get the optimal generator, and while training in practice this may not happen.

4 Experiments

Goodfellow et al trained GANs on MNIST, CIFAR10 and the Toronto Face Database.

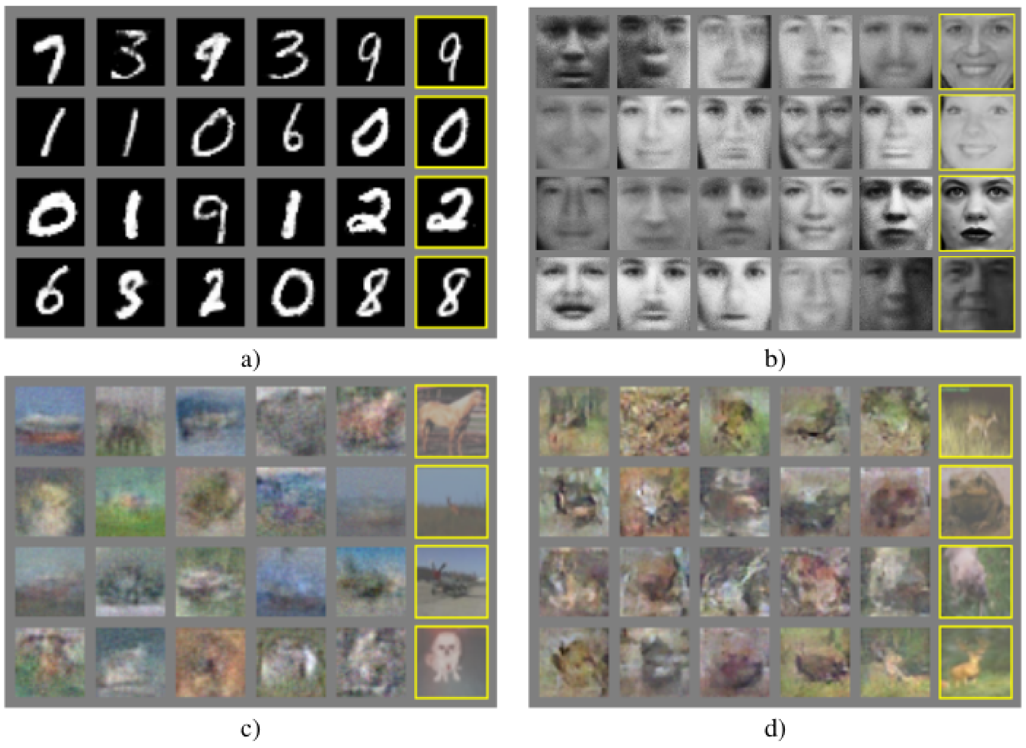


Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

Figure 4: Results from dataset experiments [1]

Once the model is trained, one can vary z between two extremes to smoothly interpolate pictures between those generated at the extreme values of z. In effect, we can use our low-to-high-dim mapping G to translate distances in z-space to distances in image-space.

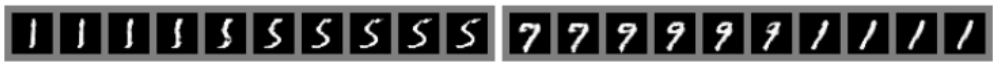


Figure 3: Digits obtained by linearly interpolating between coordinates in z space of the full model.

Figure 5: Interpolation [1]

5 Advantages and disadvantages

While GANs are able to generate sharp images of degenerate distributions, unlike Markov Chain based models which require blurriness to let the chains mix, GAN training is very unstable, and plagued with the issue of mode collapse. Mode collapse occurs when the G collapses too many values of z to the same value of x to have enough diversity to model p_{data}. This occurs when the generator tries to cheat the discriminator by collapsing lots of z values to one x value that the discriminator assigns high probability to having come from the data. Some of these issues can be fixed by changing the loss function, as in Wasserstein GAN, seen in the next class.[2]

Another issue is that it is still an open problem how to evaluate the outputs of GANs. Given an image, which may look completely unlike any real-world image, how do we tell whether the GAN is learning something useful or not?

References

[1] 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.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.