University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 1: November 6

Lecturer: Alexander Schwing Scribe: Guangyuan Wang

1 Overview

This lecture introduced Generative Adversarial Network(GAN), a generative model which aim to learn underlaying data distribution by using a generative network and discriminator network. The goal of the lecture includes:

- Recap on generative model and some related work to GAN.
- Understand the concept of GAN.

The following sections are organized as below: Section 2 provides an introduction to generative model and related work to GAN; Section 3 describe GAN in detail with formulas, training algorithm, theoretical analysis and toy example; Section 4 introduces some of the application using GAN as well as extension for GAN;

2 Recap on Generative Model and Variational Auto-Encoder

Generally, generative models refer to any model that takes a training set, which has underlying distribution p_{data},and learns an estimated distribution p_{model} from the training set.

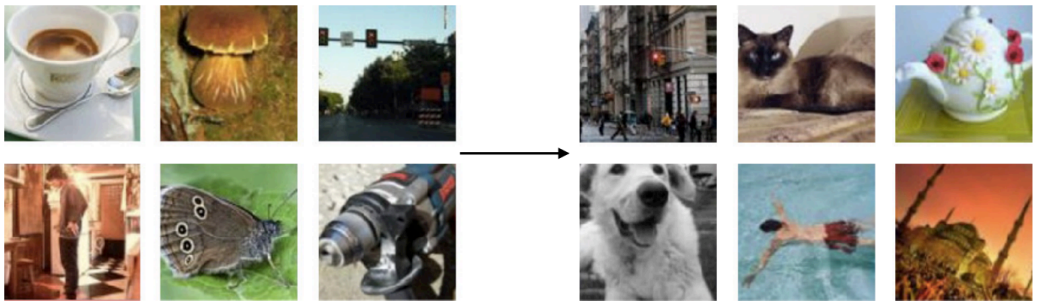

Figure 1: Examples generated by some generative model [2]

Figure 1 shows some examples where we can first train a generative model from training set and be able to generate more examples that are similar from the ones in training set. An ideal generative model would be able to train on examples as shown on the left and then create more examples from the same distribution shown on the right [2].

In the past lecture, we have covered one of such model Variational Auto-Encoder(VAE). The purpose of VAE is to learn a generative model which can generate similar data from the training dataset.

From Figure 2, we can see the VAE architecture. VAE consist of an encoder which calculate $q_\phi(z|x)$. Encoder $q_\phi(z|x)$ computes mean and variance of the Gaussian Distribution $\mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x))$.

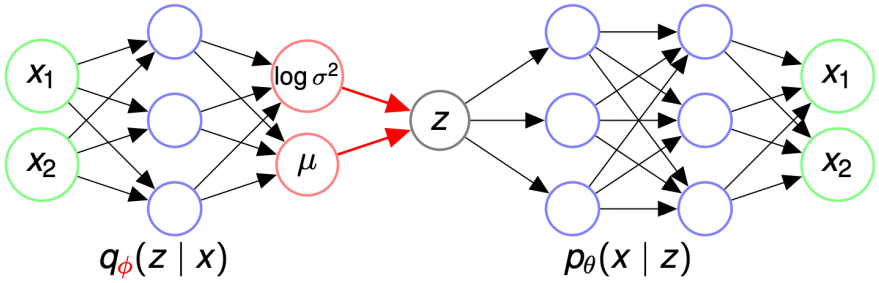

Figure 2: VAE Model

Then we can recover input data by sampling z from Gaussian Distribution and feed the z vector in to decoder network $p_\theta(x|z)$ to recover the sample back. We train VAE by maximizing the log likelihood of entire training dataset which is shown in the following equations.

$$\begin{aligned} log\ p(x) &= \int_z q_\phi(z|x) log p_\theta(x) \\ &= \int_z q_\phi(z|x) log \frac{p_\theta(x,z)}{p_\theta(z|x)} \\ &= \int_z q_\phi(z|x) log (\frac{p_\theta(x,z)}{p_\theta(z|x)} \frac{q_\phi(z|x)}{q_\phi(z|x)}) \\ &= \int_z q_\phi(z|x) log \frac{p_\theta(x,z)}{q_\phi(z|x)} + \int_z q_\phi(z|x) log \frac{q_\phi(x,z)}{p_\theta(z|x)} \\ &= \mathcal{L}(p_\theta, q_\phi) + D_{KL}(q_\phi, p_\theta) \end{aligned}$$

VAE can be trained by maximizing the lower bound $\mathcal{L}(p_\theta, q_\phi)$ since $D_{KL}(q_\phi, p_\theta)$ is always greater or equal to 0. However, since the integration over z is intractable, $\mathcal{L}$ has to be trained by sampling z which is one of the drawback. GAN in contrast, doesn't require $p(x)$ explicitly which doesn't need to sum over large probability spaces. GAN will learn to sample directly.

3 GAN Algorithm

3.1 Formulation of GAN

GAN consist of models, a discriminator and a generator. The generator captures the data distribution and the discriminator estimated the probability that a sample came from the training data rather than generator.

To learn the generator's distribution p_g over data x, a prior on input noise variable $p_z(z)$ is defined. The generator samples z from prior distribution and maps z to data space using $G_\theta(z)$, where G is a differential function with parameter θ. Typically, multilayer perceptron is chosen to be the differential function. The discriminator is another differential function represented as $D_w(x)$ which take in some input x and output a single scalar. x is some vector in data space either from dataset or from generator $G_\theta(z)$. $D_w(x)$ represents the probability the x came from the data rather than generator.

Figure 3 represents the GAN model pictorially. In Figure 3, both generator and discriminator are multilayer perceptron, where generator is the network shown in the bottom right corner and discriminator is shown in the mid top section of the image. From Figure 3, we can see, some z is

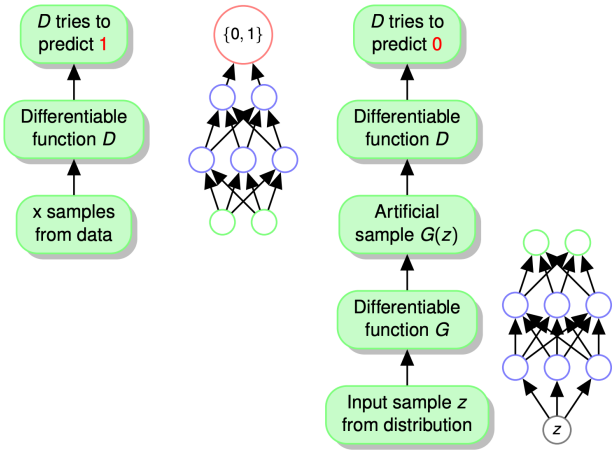

Figure 3: GAN model shown pictorially.

first sampled from the noise distribution and feed into the generator network to generate the fake sample. Then the fake sample is feed into the discriminator which we expect the output of the prediction to be 0. Besides we can get sample from generator, we can also sample data directly from dataset. In this case, the sampled data is directly feed into the discriminator which we expect the discriminator to output 1.

In order the learn the parameters for GAN, we want to train $D_w(x)$ to minimize the negative log probability of assigning the correct label to both true samples and samples from $G_\theta(z)$. Simultaneously, we want to train $G_\theta(z)$ to maximize the $-log(1 - D_w(G_\theta(z))$, meaning we want to produce high quality fake samples so discriminator think they are real samples came from the dataset. In other words, $G_\theta(z)$ and $D_w(x)$ plays the minimax game.

Mathematically, we have generator $G_\theta(z)$, and discriminator $D_w(x) = p(y = 1|x)$, we can find w using the following equation:

$$\min_w -\sum_x logD_w(x) - \sum_z log(1 - D_w(G_\theta(z))$$

We can then find θ using the following equation:

$$\max_\theta \min_w -\sum_x logD_w(x) - \sum_z log(1 - D_w(G_\theta(z)) \tag{1}$$

The detailed training algorithm is shown below in Algorithm 1. From Algorithm 1 we can see discriminator and generator is trained alternatively, when discriminator is trained, generator is fixed, and vise versa. However, the training procedure shown in Algorithm 1 is not optimal due to low gradient when updating generator at early stage of training. In practice, many heuristics are used to make the optimization more stable, more in detail will be covered in section 3.3.

3.2 Theoretical Results of GAN

In this section, we will show that the training criterion allows one to recover the data generation distribution as G and D are given enough capacity. Since generator G implicitly defines a probability distribution p_g as the distribution of the samples $G_\theta(z)$ obtained when $z \sim p_z$, we would want out network to coverage a good estimator of p_{data}.

Algorithm 1: Minibatch stochastic gradient descent training of GAN [3]

for *number of training iterations* **do**

  for *k steps* **do**

    • Sample minibatch of m noise samples $\{z^{(1)}, \cdots, z^{(m)}\}$ from noise prior $p_z(z)$.

    • Sample minibatch of m examples $\{x^{(1)}, \cdots, x^{(m)}\}$ from dataset.

    • Update the discriminator by descending its stochastic gradient:

$$\nabla_w \frac{1}{m} \sum_{i=1}^{m} [-logD_w(x^{(i)}) - log(1 - D_w(G_\theta(z^{(i)}))]$$

  • Sample minibatch of m noise samples $\{z^{(1)}, \cdots, z^{(m)}\}$ from noise prior $p_z(z)$.

  • Update the generator by ascending its stochastic gradient:

$$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -log(1 - D_w(G_\theta(z^{(i)}))$$

First we want to show the optimal discriminator assuming arbitrary capacity for any given generator G. We can find the optimal discriminator using following equation:

$$\min_D - \int_x p_{data}(x) log D(x) dx - \int_z p_z(z) log(1 - D(G(z)) dz$$

since generator G draws z from $p_z(z)$, therefore we can rewrite $\int_z p_z(z) log(1 - D(G(z)) dz$ using $p_G(x)$ and integrating over x space as following:

$$\min_D - \int_x p_{data}(x) log D(x) dx - \int_x p_G(x) log(1 - D(x) dx$$

with little simplification, we get:

$$\min_D - \int_x p_{data}(x) log D(x) dx + p_G(x) log(1 - D(x) dx \tag{2}$$

Notice the above form of equation is form of Euler-Lagrange equation

$$S(D) = \int_x L(x, D(x), \dot{D}(x))$$

Function D is stationary at local maxima and minima, and when D is in stationary form, we have:

$$\frac{\partial L(x, D(x), \dot{D}(x))}{\partial D} - \frac{d}{dx} \frac{\partial L(x, D(x), \dot{D}(x))}{\partial \dot{D}} \tag{3}$$

since $\dot{D}$ is not in equation 2, we can ignore the last term in equation 2, thus we have

$$\frac{\partial L(x, D(x), \dot{D}(x))}{\partial D} = 0 \tag{4}$$

by solving equation 3, we get:

$$\frac{\partial L(x, D(x), \dot{D}(x))}{\partial D} = -\frac{p_{data}}{D} + \frac{p_G}{1 - D} = 0$$

consequently we get the optimal discriminator as following:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

Now we have found the optimal discriminator, now we want to find the optimal generator given this discriminator. We can take the equation 1 with discriminator fixed and optimize for maximum. So we can get the following equation:

$$\max - \int_x p_{data}(x) log D(x) dx + p_G(x) log(1 - D(x) dx \tag{5}$$

Since we know the optimal discriminator D is $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$, we can replace $D(x)$ in equation 4 with $D^*(x)$, thus we get:

$$\max - \int_x p_{data}(x) log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} dx + p_G(x) log \frac{p_G(x)}{p_{data}(x) + p_G(x)} dx \tag{6}$$

$$= max(-2JSD(p_{data}, p_G) + log(4)) \tag{7}$$

where

$$JSD(p_{data}, p_G) = \frac{1}{2} KL(p_{data}, M) + \frac{1}{2} KL(p_G, M) \text{ with } M = \frac{1}{2}(p_{data} + p_G) \tag{8}$$

In order to maximize equation 6, we want to minimize JSD. Since KL between two distribution is always greater or equal to 0, which means the minimum JSD is 0 with both KL divergence equal to 0. When JSD is equal to 0, we get the maximum value for equation 5 which is log(4).

When JSD is equal to 0, we have

$$JSD(p_{data}, p_G) = \frac{1}{2} KL(p_{data}, p_{data}) + \frac{1}{2} KL(p_G, p_G) = 0$$

from which we can infer M in equation 7 is $M = p_{data} = p_G$, consequently, we have $p_G(x) = p_{data}(x)$. This shows that the optimal generator is the one that perfectly replication the data distribution.

3.3 Heuristics for Optimization

In practice the training procedure described in Algorithm 1 may not provide sufficient gradient for G to learn well. Early in learning, when G is poor, D can reject samples from G with high confidence because they are clearly different from the training data. In this case $-log(1 - D(G(x))$ from equation 1 saturates with little gradient flowing back to update on G. This case can be seen on left figure in Figure 4.

In Figure 4, the red cross represents the loss for some samples. In the left figure in Figure 4, we want to move the red cross to the left direction in x-axis in order to maximized the loss. However, we can see when G is not very good, D can separate samples from G from the training data really well, resulting in small gradient to update G. Since G has very small gradient, resulting minimal update on G, therefore D can easily separate samples from G and from training data. The saturation on G will result G not be able to update very well, and when G is updated, D can quickly change its parameter to compensate the change G has made. In order to solve this issue, we change the original objective in G from:

$$\max_\theta - log(1 - D(G_\theta(x))$$

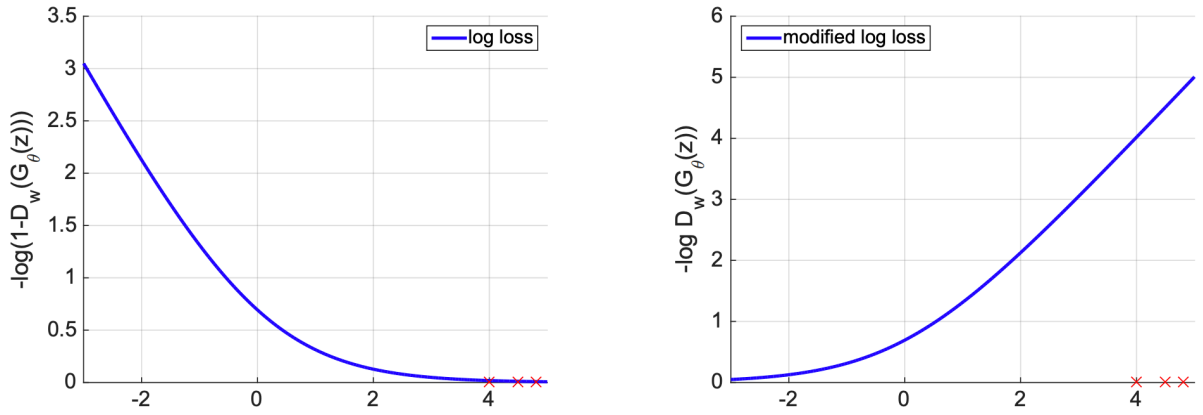

Figure 4: Original Generator Loss(left) vs Modified Generator Loss (right)

to

$$\min_{\theta} -log(D(G_\theta(x))$$

By changing this objective, we now flipped the loss function. As shown in Figure 4, the new modified loss has the curve that's flipped. From the modified objective function, we can see when G generate easily classified samples and D detects it, $-log(D(G_\theta(x))$ is large. Since we want G to generate the samples as close to real sample as possible, we want to minimize $-log(D(G_\theta(x))$. From the right part of Figure 4, we can see when G is poor the gradient on easily classified sample is large resulting a quick update on G. This change will make both G and D converge. One drawback for this change, is that since now we change the objective for G, there is no joint cost function for both D and G.

3.4 Advantages and Disadvantages of GAN

- Advantage
  - Can avoid sampling difficulty appeared in Variation Auto-Encoder.
  - The images generated from GAN appear to be much sharper than those generated from VAE.
  - Adversarial models may also gain some statistical advantage from the generator network not being updated directly with data examples, but only with gradients flowing through the discriminator [3]. This means the generator's parameter doesn't copy input components.
  - Another advantage of adversarial networks is that they can represent very sharp, even degenerate distributions, while methods based on Markov chains require that the distribution be somewhat blurry in order for the chains to be able to mix between modes [3].
- Disadvantage
  - Difficulty to train the network due to its unstable nature. During training, D must be synchronized well with G. In particular, G must not be trained too much without updating D, in order to avoid the Helvetica scenario in which G collapses too many values of z to the same value of x to have enough diversity to model p_{data} [3].
  - $p_G(x)$ is not explicitly represented.

3.5 Toy Exmples

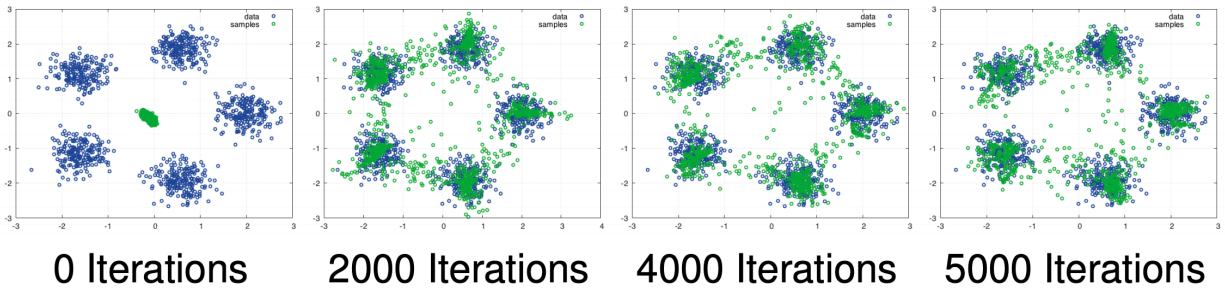

Figure 5: Toy Example on Training of GAN

Figure 5 shows the training process of GAN. In Figure 5, green dots represent the samples from Generator and blue dots represent the samples from dataset. At 0th iteration, we can see that all samples generated by generator are centered together and don't capture the underlying data distribution. At 2000th iteration, the samples generated by generator mostly matches the underlying data distribution. At 4000, and 5000 iterations, the model is converged with generator being to capture and replicate the underlying data distribution.

4 Application and Extension of GAN

GAN received increasing attention over the past few year. There are a lot of works that built upon the original GAN. Original GAN has the ability to model images from different dataset which is shown in Figure 6.

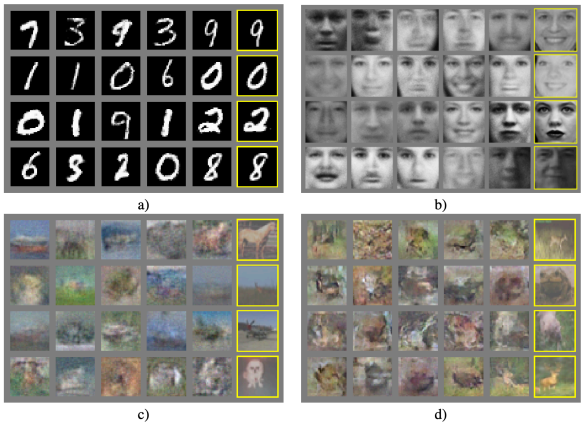

Figure 6: Visualization of Samples from Model

In Figure 6, right most columns visualization shows the nearest training examples of the neighboring sample. We can see GAN is able to generate various images that looks similar to the ones in dataset not strictly following the dataset distribution.

4.1 DCGAN

In DCGAN [4], author tried to bridge the gap between the success of CNN for supervised learning and unsupervised learning. In DCGAN, the generator in normal GAN is replaced with CNN layers that takes the noise distribution sample z and deconv z into an image.

Figure 7: DCGAN Image Generation Example

The images that are generated by this network has much more improvement in visual quality. As we can see from Figure 7, which shows the generated bedroom image, the image looks really shape and clear comparing to the images generated from the original GAN paper.

4.2 StackGAN

In StackGAN [5] the author tired to synthesize photo-realistic images from text descriptions. The problem is decomposed into two sub-problems through a sketch-refinement process. The first sub-problem is to sketches the primitive shape and colors of the object based on the given text description, yielding Stage-I low-resolution images. In second stage, the model takes Stage 1 results and text descriptions as inputs, and generates high-resolution images with photo-realistic details. The result of this this work in shown in Figure 9, where we can see the process from text to low-res image to high-res image. In comparsion, Vanilla GAN is not be able to generate plausible image at such high resulion.

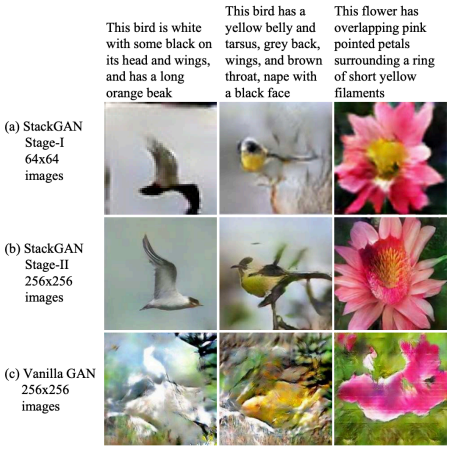

Figure 8: StackGAN Example

4.3 Wasserstein GAN

Wasserstein GAN(WGAN) [1] proposes a new cost function using Wasserstein distance to measure the difference between two distribution. Using Wasserstein distance will solve the problem where generator may not get enough gradient to update since Wasserstein distance has a smoother gradient everywhere. The plot shown below plots the D(x) value for both GAN and WGAN. For GAN(shown in red), it has vanishing gradient when fake data is easily recognized by discriminator. In contrast, WGAN(shown in blue), has smooth gradient everywhere which makes WGAN learns better even when generator is poor.

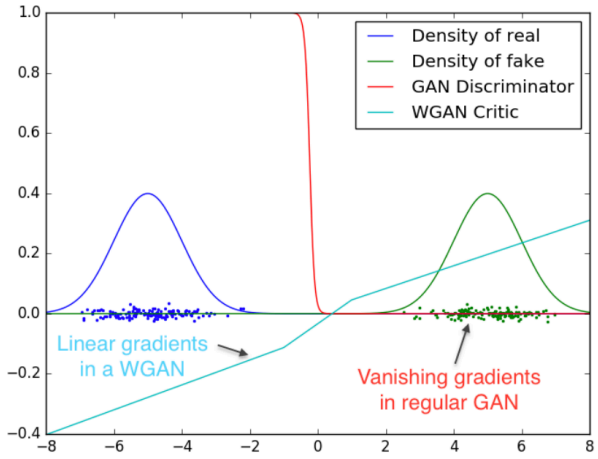

Figure 9:

Mathematically, Wasserstein distance is represented as following:

$$W(p_{data}, p_G) = \min_{p_j(x,x') \in \Pi(p_{data}, p_G)} \mathbb{E}_{p_j}[||x - x'||] \tag{9}$$

From equation 9, we can reformulate GAN as following:

$$\min_{p_G} W(p_{data}, p_G) = \min_{p_G} \min_{p_j(x,x') \in \Pi(p_{data}, p_G)} \mathbb{E}_{p_j}[||x - x'||] \tag{10}$$

From equation 10, we can see two problems. First, p_{data} is not avaliable, we only have samples from p_{data}. Also, we can't compute the joint distribution p_j. The solution to that is to use Kantorovich-Rubinstein duality to obtain the following equations:

$$W(p_{data}, p_G) = \max_{||f||_L \leq 1} \mathbb{E}_{p_{data}}[f(x)] - \mathbb{E}_{p_G}[f(x')] \tag{11}$$

$$\min_{p_G} \max_{w} \mathbb{E}_{p_{data}}[f_w(x)] - \mathbb{E}_{p_G}[f_w(x')] \tag{12}$$

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[2] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[4] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[5] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint*, 2017.