Generative Adversarial Networks

Deep Learning
Bryan Pardo
Spring 2022

Recap: Discriminators & Regressors

- Everything we’ve seen, so far, maps some high-dimensional input (e.g. a still image with 10^6 pixels) to a low dimensional output (e.g. a 10-way classification, or a single real value)
- Something that outputs a finite set of classes is called a classifier.
- Something that outputs a vector of a few real values is a regressor.
- Are there other problems in the world that aren’t classification or regression?

Generative models

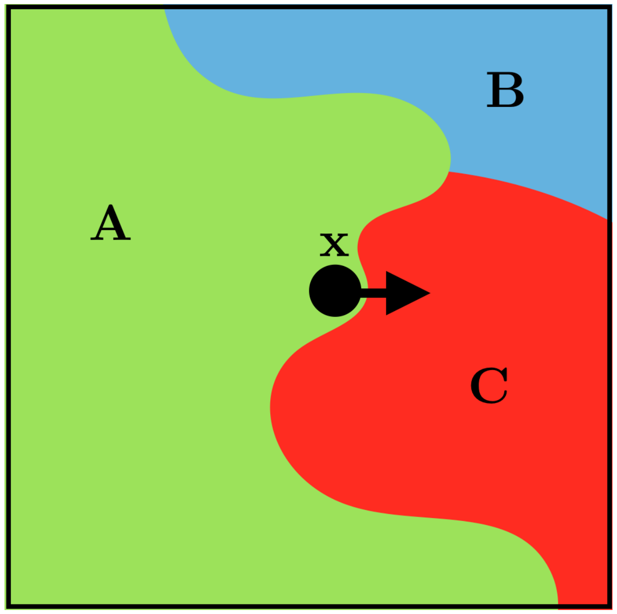
- What about generating new examples of things in the world?

Real images of imaginary people generated by StyleGAN2 https://github.com/NVlabs/stylegan2

How can we create models that do this?

Recall adversarial examples

- We “nudge” an existing example of class A over a discriminator’s decision boundary to be labeled as C
- It typically still looks like class A
- Can we take this idea further to make examples that actually LOOK like the target class?



Fast gradient sign method (undirected attack)

```
Initialize $X^{(0)}$
Repeat until stopping condition met:
    $X^{(t+1)} = X^{(t)} + \eta \cdot sign(\nabla_X L(X^{(t)}, Y; \theta))$
Return $X^{(enough)}$
```

$X^{(t)}$ is an example at time t

$\nabla_X \, L(X^{(t)}, Y; \theta)$ is the gradient of the loss function with respect to example features $X^{(t)}$

η controls the step size

$X^{(enough)}$ is the minimal change needed to flip the category of X

Fast gradient sign method (directed attack)

```
Initialize $X^{(0)}$
Repeat until stopping condition met:
    $X^{(t+1)} = X^{(t)} - \eta \cdot sign(\nabla_X L(X^{(t)}, notY; \theta))$
Return $X^{(enough)}$
```

$X^{(t)}$ is an example at time t

$\nabla_X \, L(X^{(t)}, Y; \theta)$ is the gradient of the loss function with respect to example features $X^{(t)}$

η controls the step size

$X^{(enough)}$ is the minimal change needed to flip the category of X

notY is now the NEW CATEGORY WE WANT TO LABEL THE EXAMPLE AS

Fast gradient sign method (directed attack)

```
Initialize $X^{(0)}$
Repeat until stopping condition met:
    $X^{(t+1)} = X^{(t)} - \eta \cdot sign(\nabla_X L(X^{(t)}, newY; \theta))$
Return $X^{(enough)}$
```

$X^{(t)}$ is an example at time t

$\nabla_X \ L(X^{(t)}, Y; \theta)$ is the gradient of the loss function with respect to example features $X^{(t)}$

η controls the step size

$X^{(enough)}$ is the minimal change needed to flip the category of X

newY is now the NEW CATEGORY WE WANT TO LABEL THE EXAMPLE AS

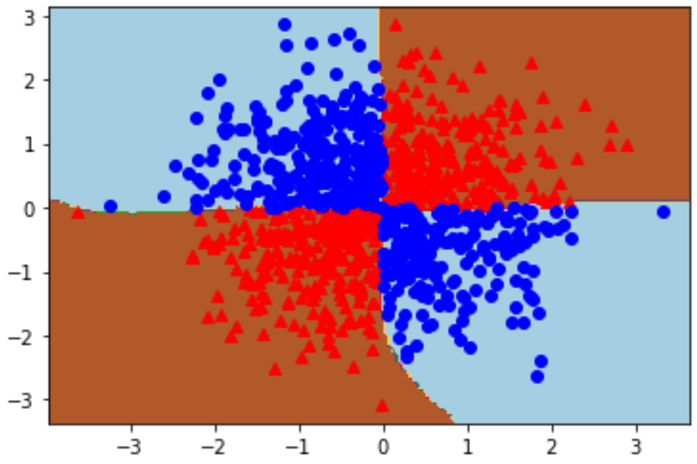
What if $X^{(0)}$ started as random noise?

$$X^{(t+1)} = X^{(t)} - \eta sign(\nabla_X L(X^{(t)}, newY; \theta))$$

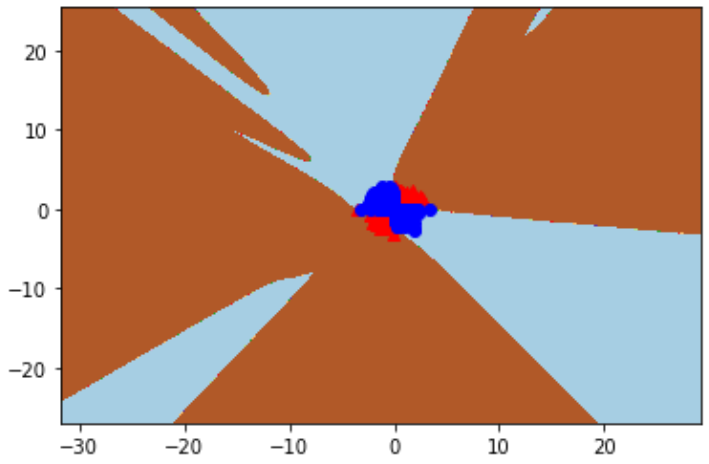
- Would running this repeatedly be enough to generate good-looking examples?
- Why? Or Why not?

Data vs Decision surface

Learned XOR decision surface

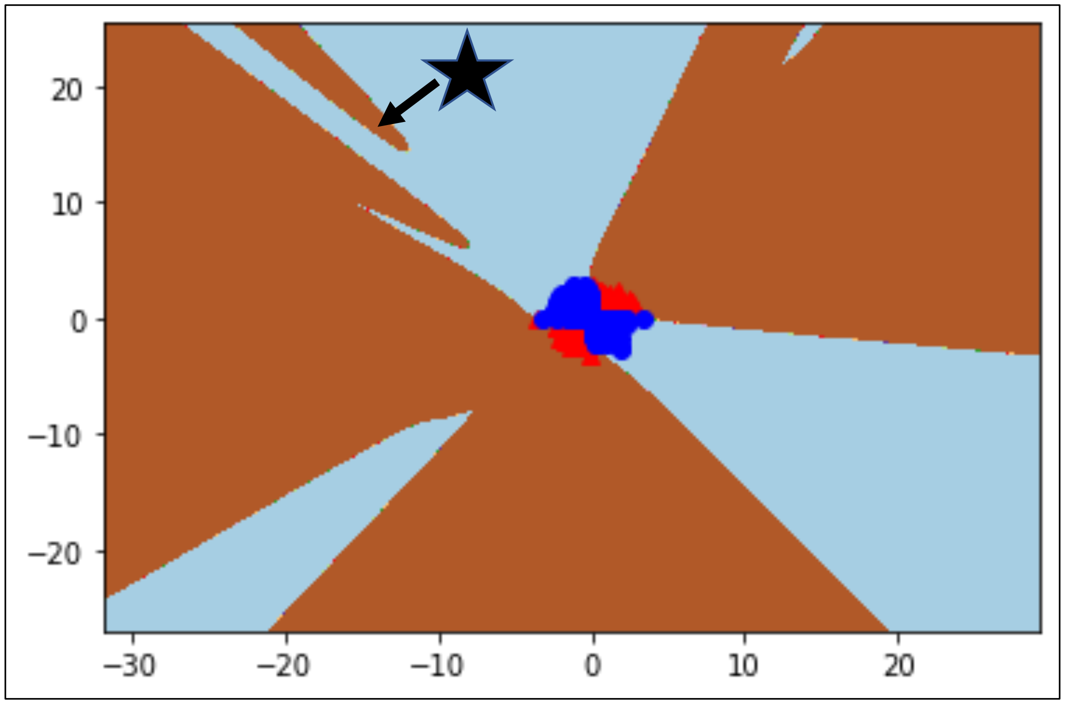


Same decision surface, zoomed out



- A discriminator will assign a label to EVERY region in the space, regardless of whether there was any training data there.

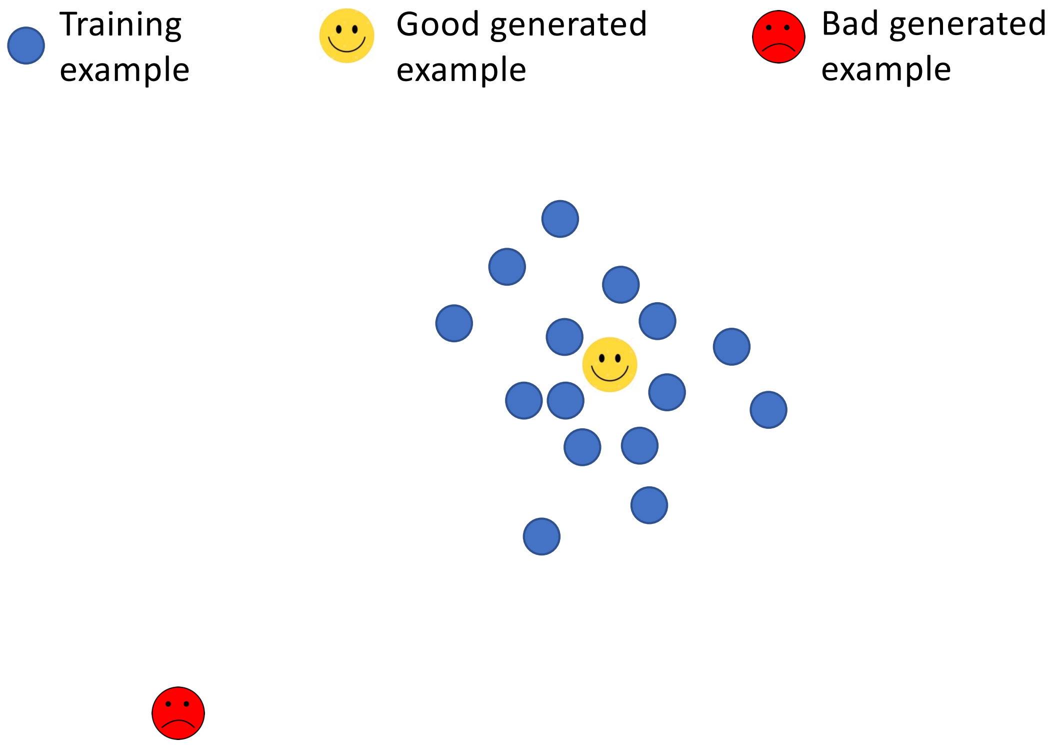
What happens to a nudge here?



- If we start with a random point in the input space, will nudging to the nearest region labeled as the target class result in something similar to examples in the training distribution?

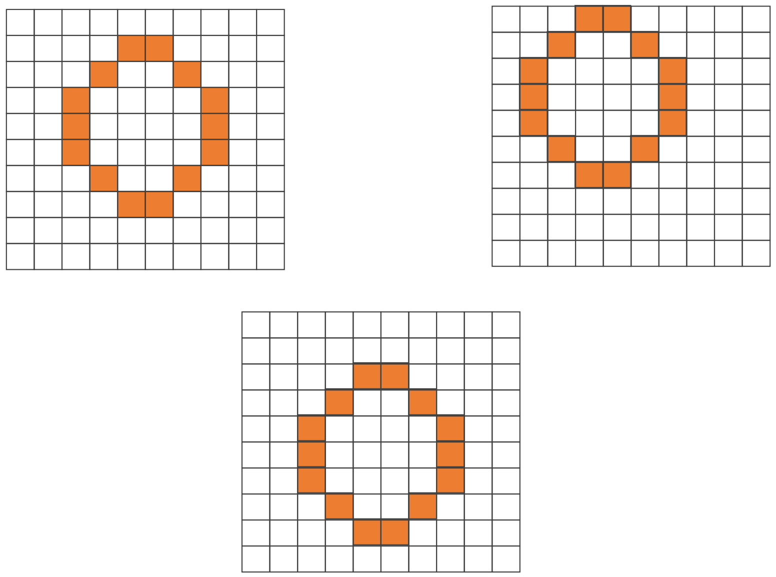
What if we…..

- Aim for making new examples that fall within the training distribution
- How can we tell when we have generated something in the training distribution?

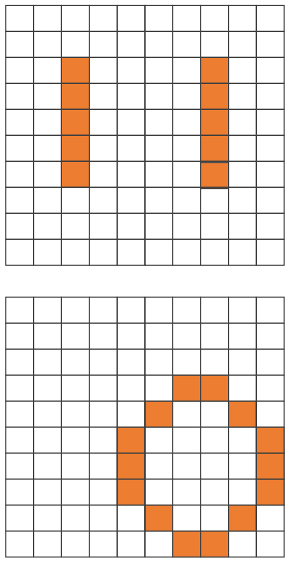


Could we use an LP norm in the image space?

Our 3-example Target Distribution



Two generated examples



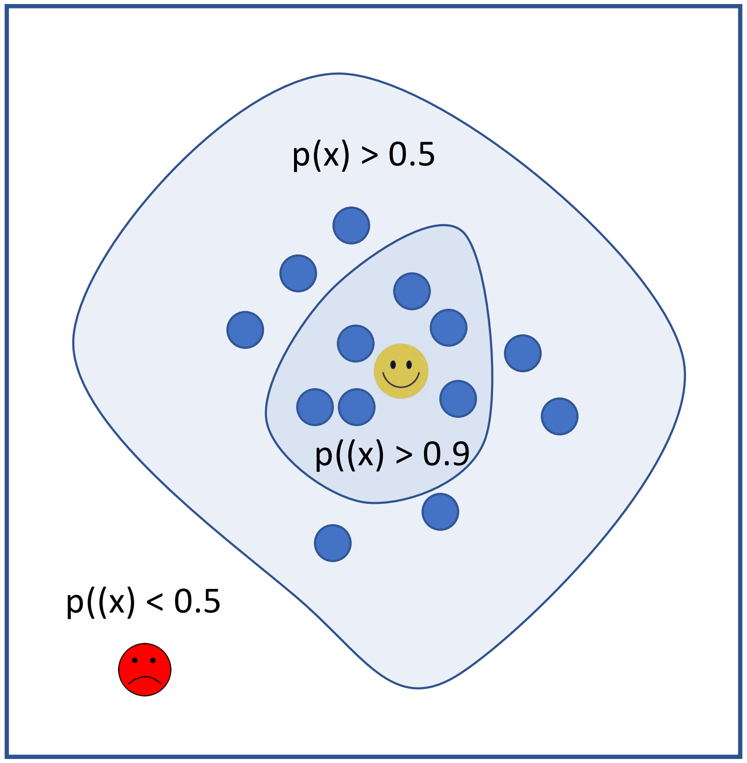
Which generated example is more like the target distribution?
Which generated example is closer to the distribution, as measured by an LP norm in the image space?

Let’s use a deep network as our measure

- Goal: a network that outputs the probability some new example was drawn from the training data.

- …but it should be smarter than just applying a Gaussian fit directly to the input representation.

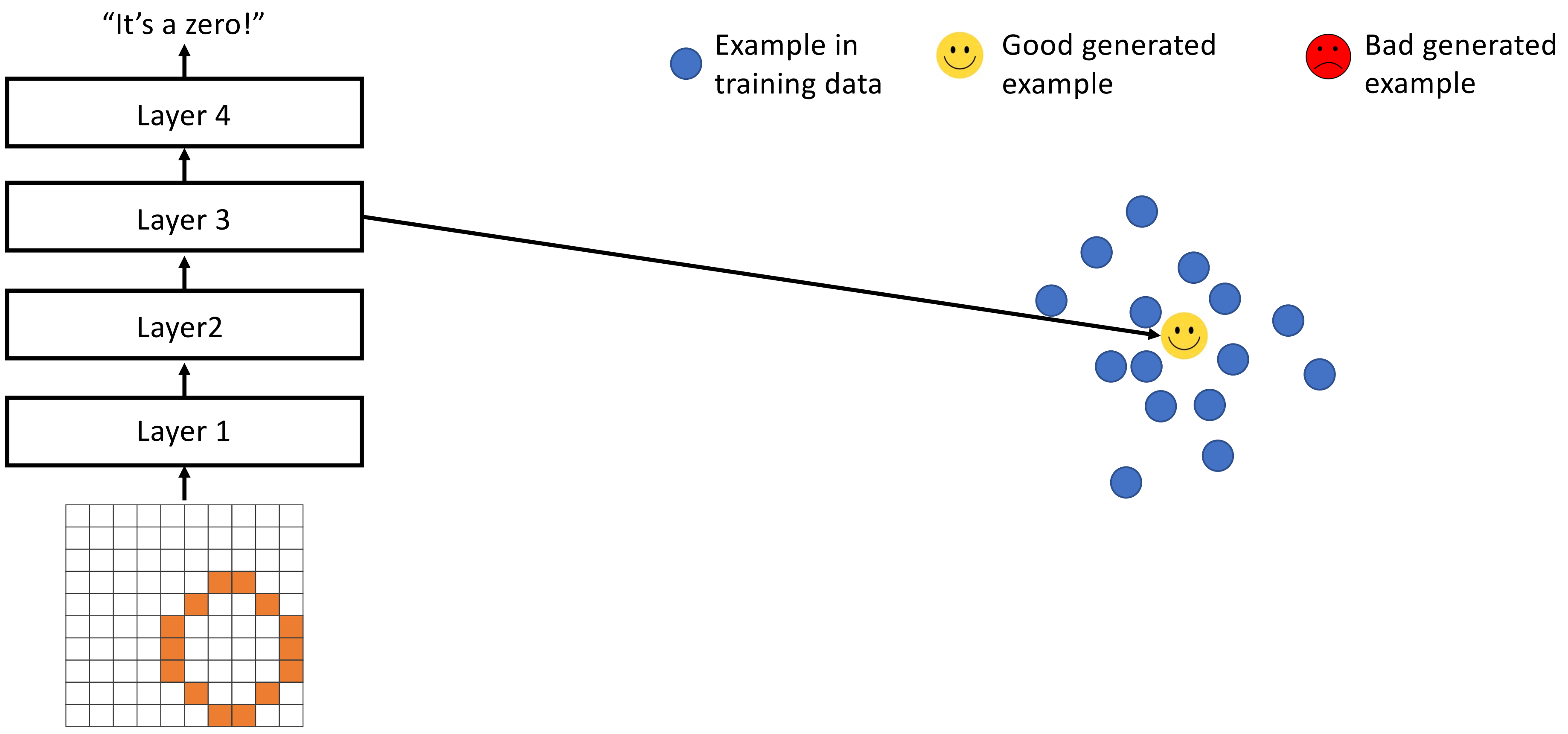
- How will we do this?



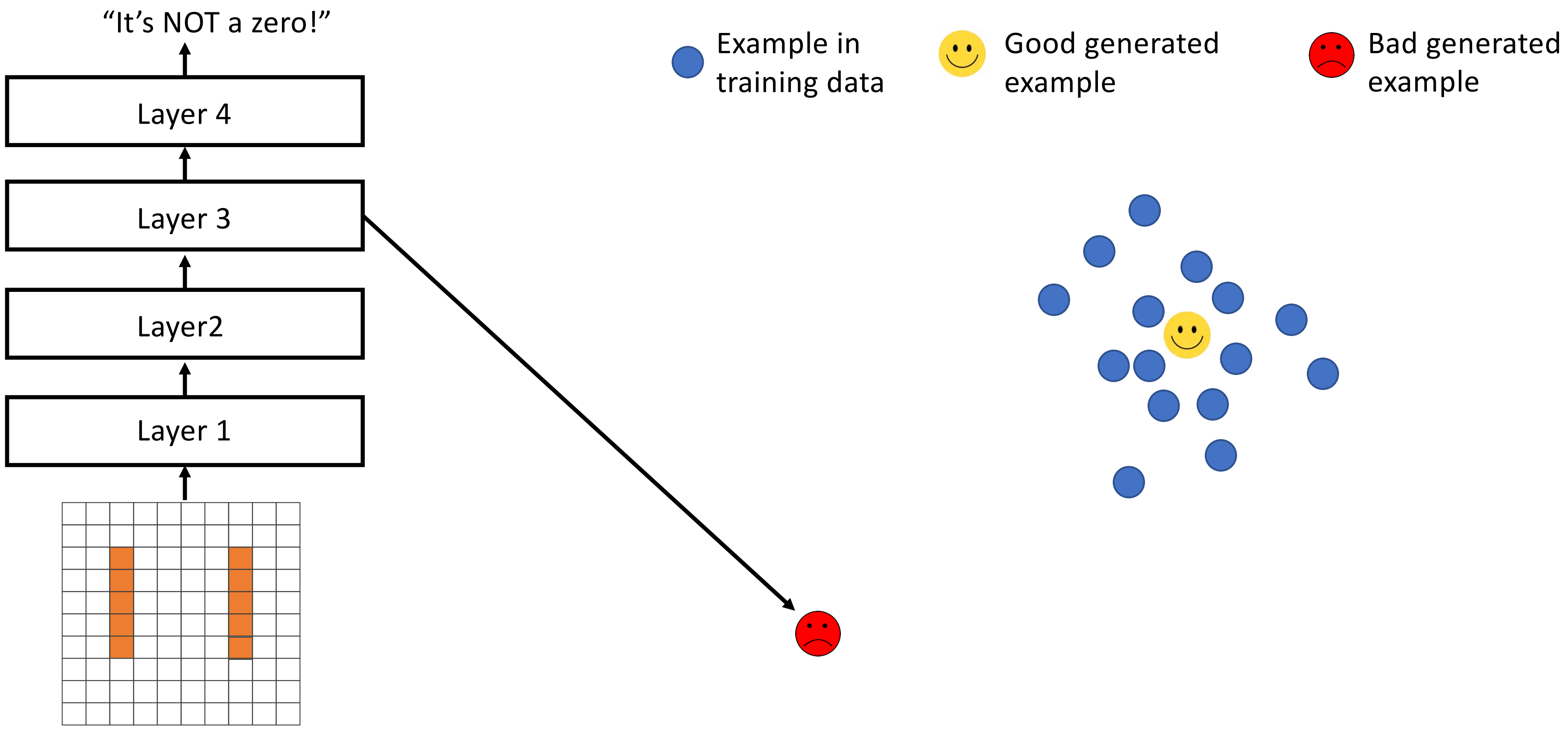
Layers/Embeddings/Data Transformations

- Each layer of a deep net maps its N by M by D input into a N’ by M’ by D’ output
- This output can be higher or lower dimensional than the input
- We can treat the output of any layer as a representation of the data
- This representation is commonly called an “embedding”
- Embeddings from deeper layers of a well-trained discriminator network allow meaningful clustering of the data

Looking at the Discriminator's embedding space



Looking at the Discriminator's embedding space



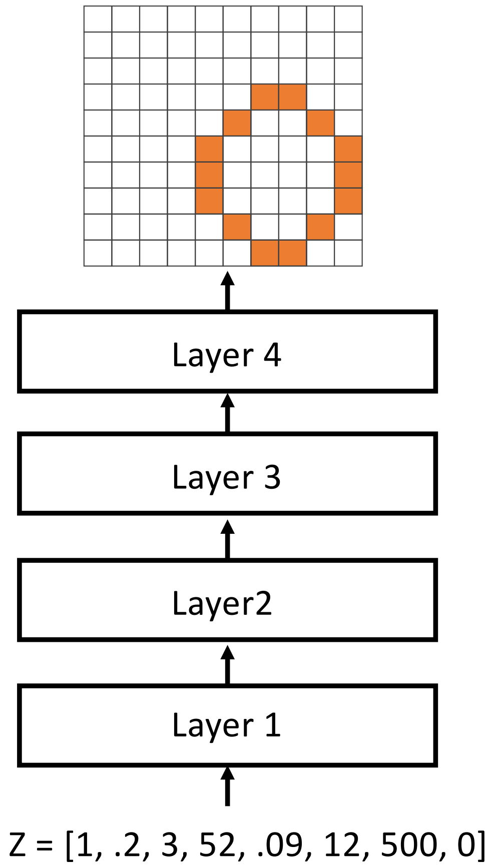
The data distribution is implicitly coded

- The discriminator doesn’t output the distribution. It only tells you if a particular example falls within the distribution.

- So how would you generate a good image with this?

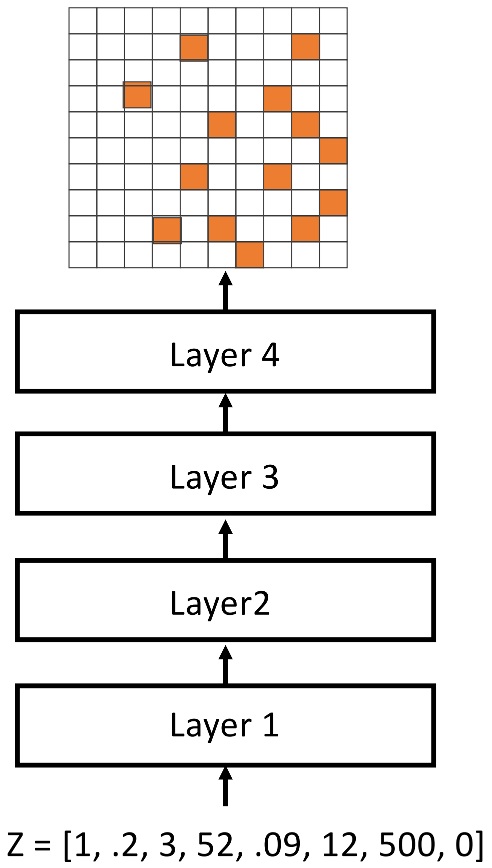
Let’s make an image generator.

- Take some N-dimensional vector as input (N is usually small, like 8 or 128). Call it Z.
- Run it through a bunch of layers that increase the representation size until you…
- Output an H by W by C dimensional matrix that we interpret as an image
- No…I haven’t told you how to train it, yet.



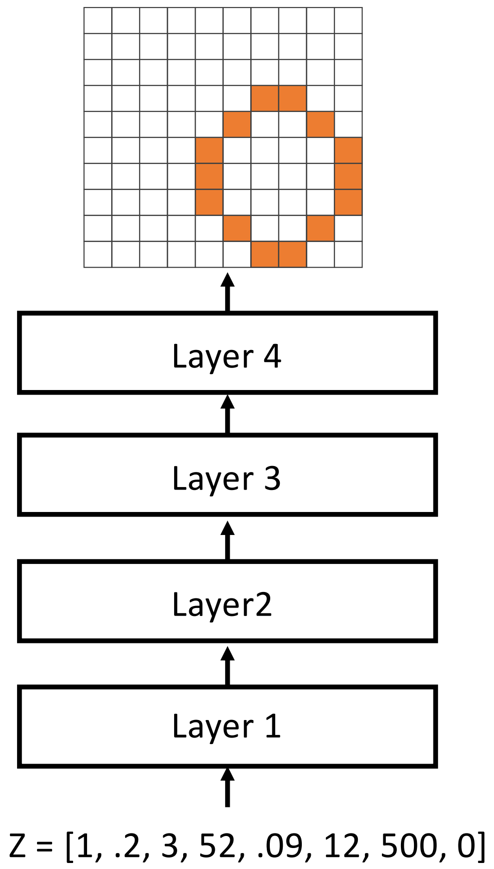
A more accurate representation of the untrained output

- Take some N-dimensional vector as input (N is usually small, like 8 or 128). Call it Z.
- Run it through a bunch of layers that increase the representation size until you…
- Output an H by W by C dimensional matrix that we interpret as an image
- No…I haven't told you how to train it, yet.



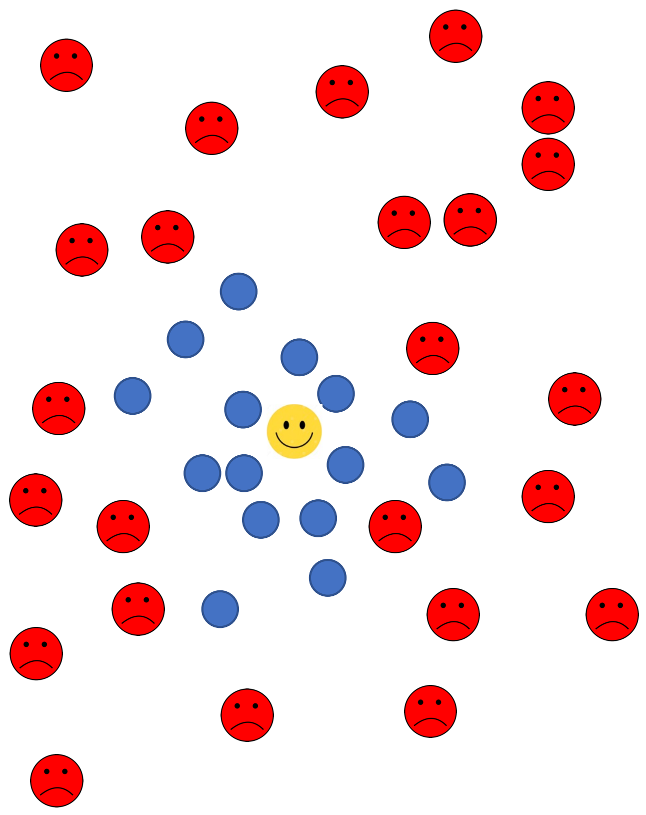
Using our image generator

- (randomly??) pick some input vector Z
- Output an image
- See what our discriminator thinks of it
- Repeat until the discriminator says an image is good



Random guessing

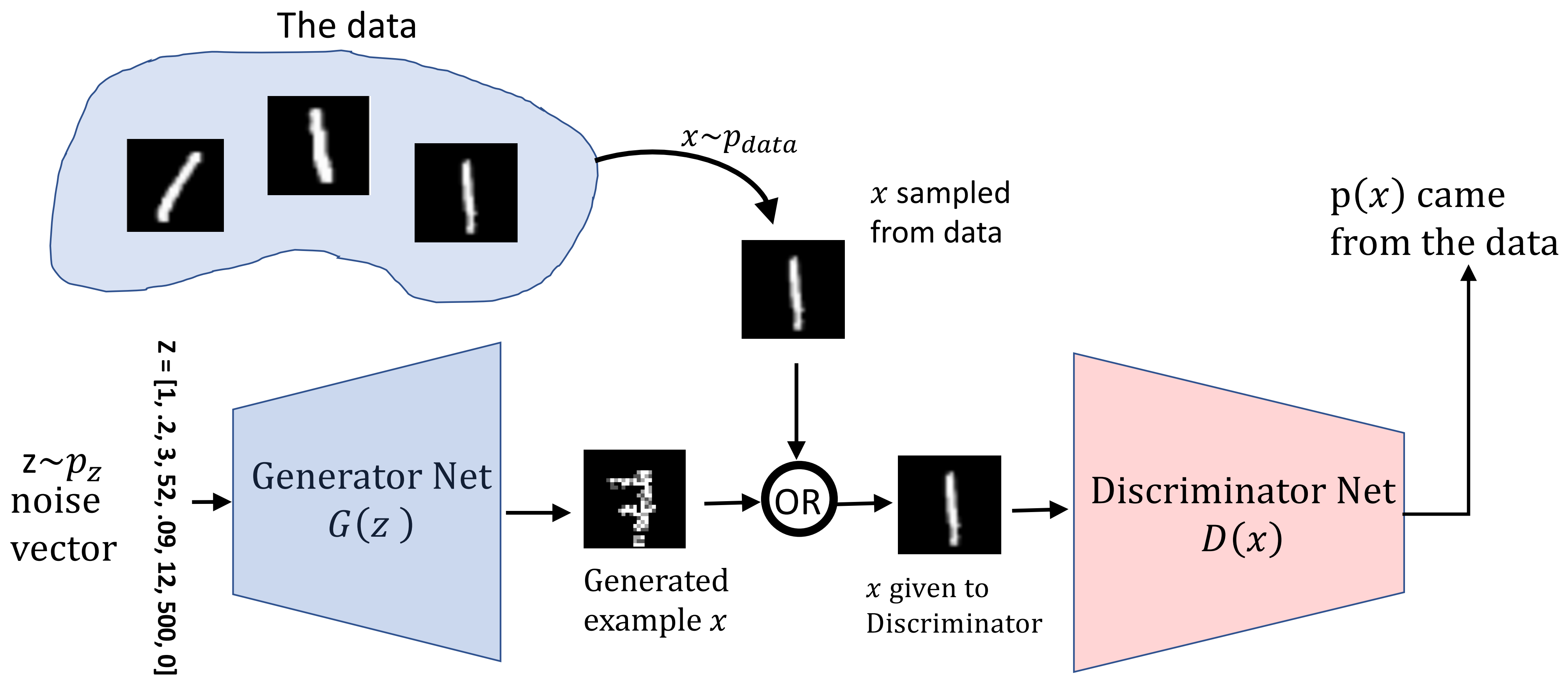
- In the end we want an example generator, not a discriminator.
- If we have a perfect discriminator, how can we ensure that we don’t have to make 10,000 or 10^6 random guesses to generate a one good example?
- Could we train a generator to make only good examples?



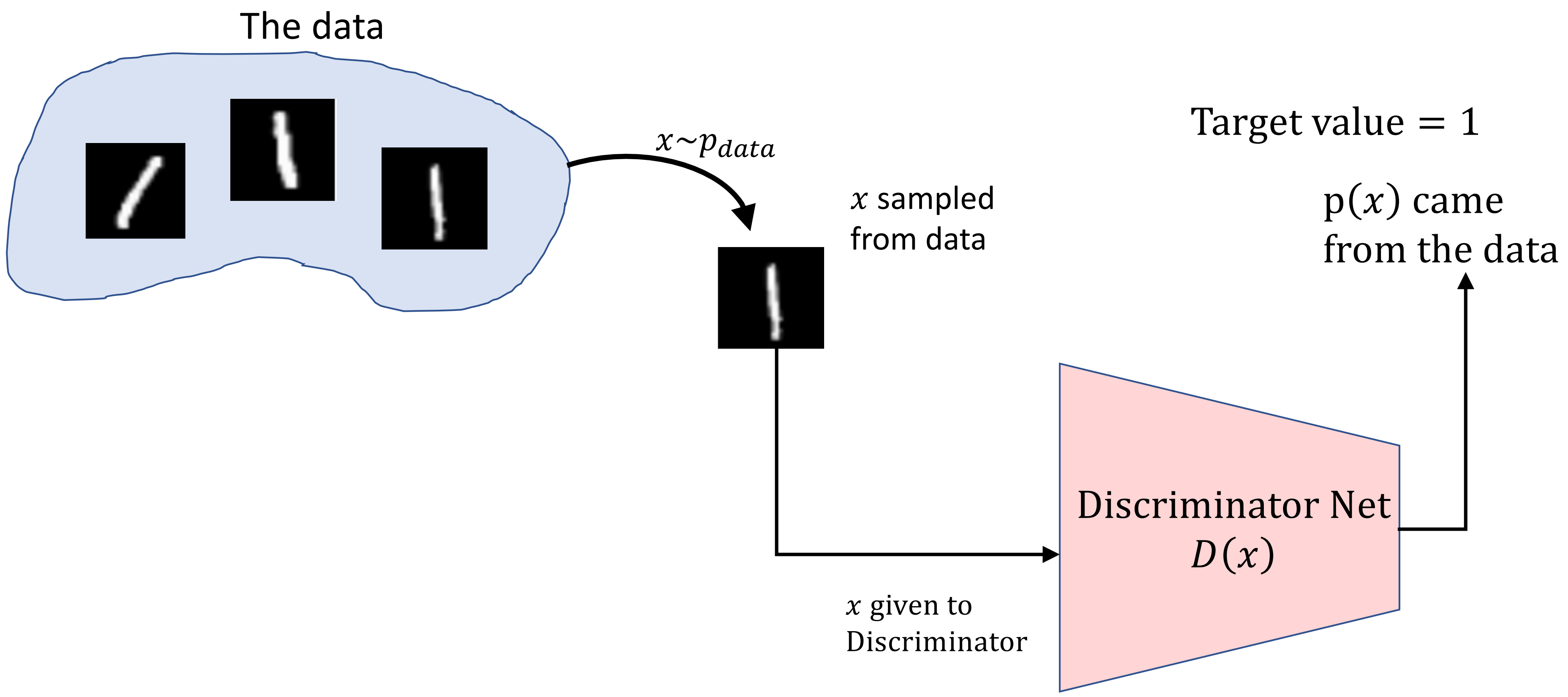
Generative Adversarial Networks (GANs)

- The big idea: We’ll train our Discriminator and our Generator simultaneously, so that they both learn the data distribution.
- That way, once we have a trained Generator, it will generate something good on the 1st try instead of on the 10,000th try.

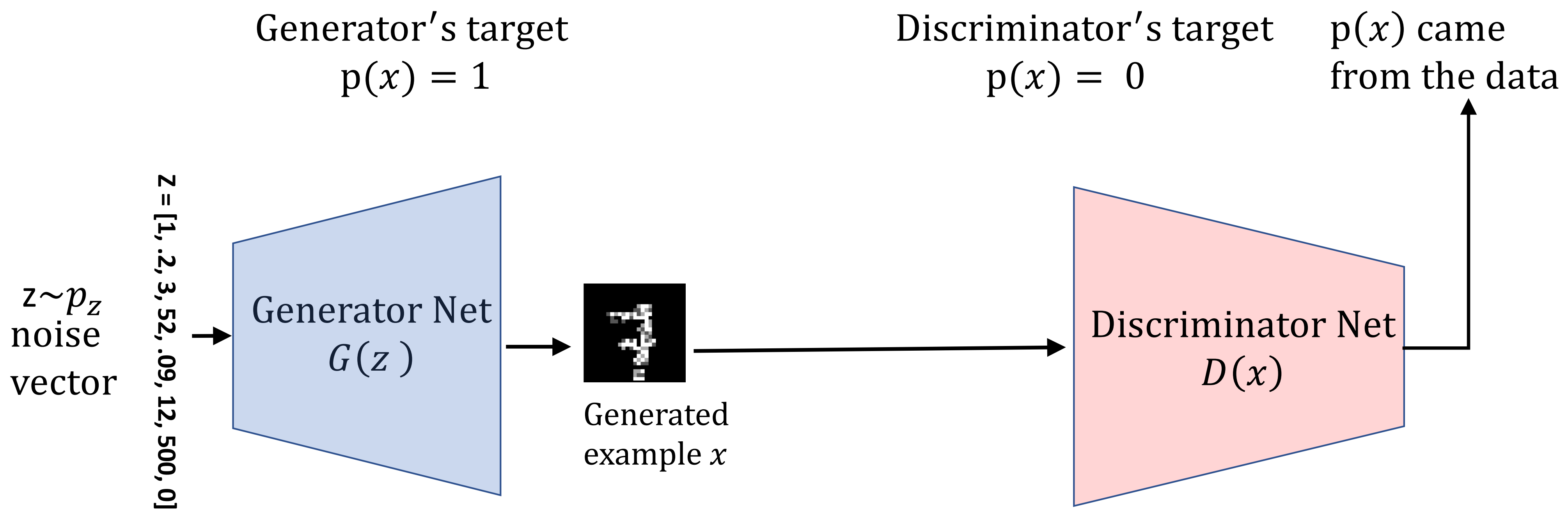
The training setup



For true examples



For generated examples



Framed as mini-max

$$\min_G \max_D V(D, G) =$$

$$\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Training the Discriminator

Let’s put this in Binary Cross Entropy (BCE) terms.

Recall that D(x) is the Discriminator’s …estimate of the probability x is drawn from the real data.

$$\begin{aligned}\mathrm{BCE}_{\mathrm{total}} &= \mathrm{BCE}_{\mathrm{real}} + \mathrm{BCE}_{\mathrm{generated}} \\ &= \mathrm{BCE}(1, D(x)) + \mathrm{BCE}(0, D(G(z))) \\ &= -\log D(x) - \log(1 - D(G(z)))\end{aligned}$$

Training the Generator

$$\begin{aligned}\mathrm{BCE}_{\mathrm{total}} &= \cancel{\mathrm{BCE}_{\mathrm{real}}} + \mathrm{BCE}_{\mathrm{generated}} \\ &= \mathrm{BCE}(1, D(G(z))) \\ &= -\log D(G(z))\end{aligned}$$

Notice that G is NEVER trained directly on the real data!
It learns to generate real-looking images without ever seeing one.

The hope

- As D improves on identifying members of the training distribution, G is forced to make better and better fakes.
- Eventually G only generates items close to the training distribution.
- Put another way, G learns the data distribution…even if it’s never seen any of the data set.

So what’s the catch?

GANs are like the Force. Balance is needed.

- What if the Discriminator is never fooled (or always fooled) by the Generator for some stretch during training?
- What happens to the gradients for the Generator?
- Will either D or G improve?

It can be tricky

- Strategies to make G relatively stronger
  - Give it 2 training steps for every 1 of D
  - **Give it an extra loss function that isn’t based on D**
  - Add capacity to the model (make it bigger)
- Strategies to make D stronger
  - Pretrain on out-of-distribution examples of real data
  - Give it 2 training steps for every 1 of G
  - Add capacity to the model

Also there's this….

Let's look at the generator's loss function again.

$$Loss = -\log(D(G(z))$$

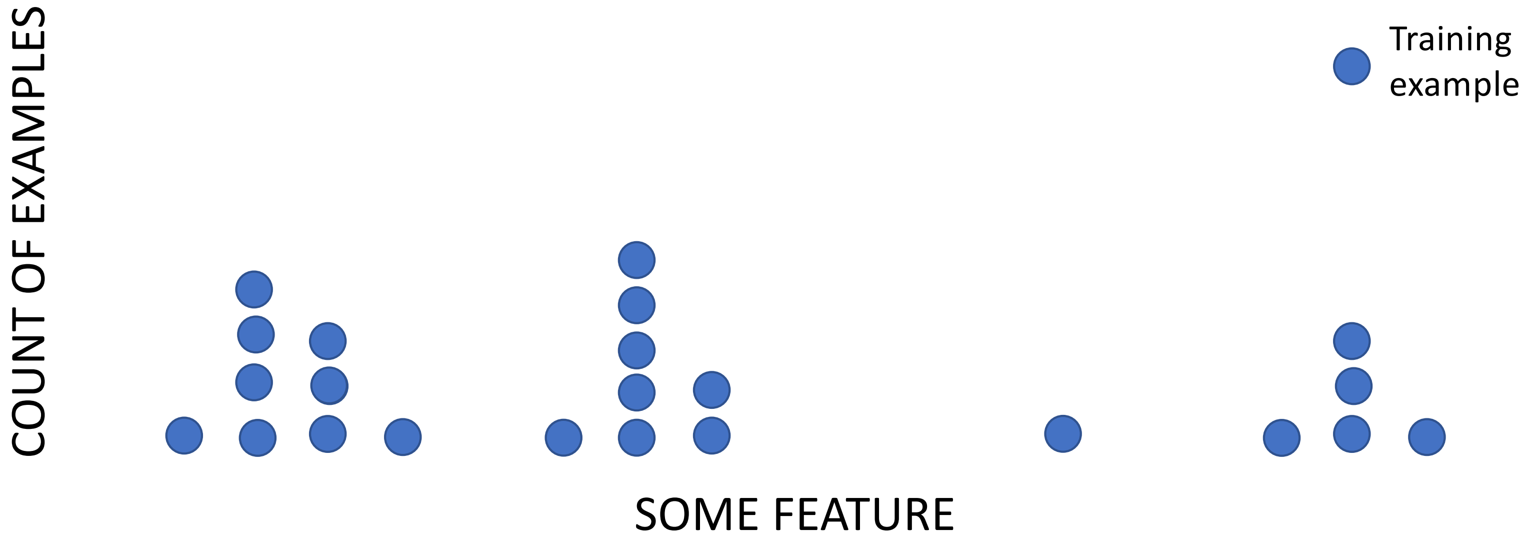
Remember, the generator never sees the true training distribution.

Is there something missing in this loss function?

Modeling the p_{data} distribution with G(z)

$p_{G(z)}$ is the Generator's estimate of p_{data}

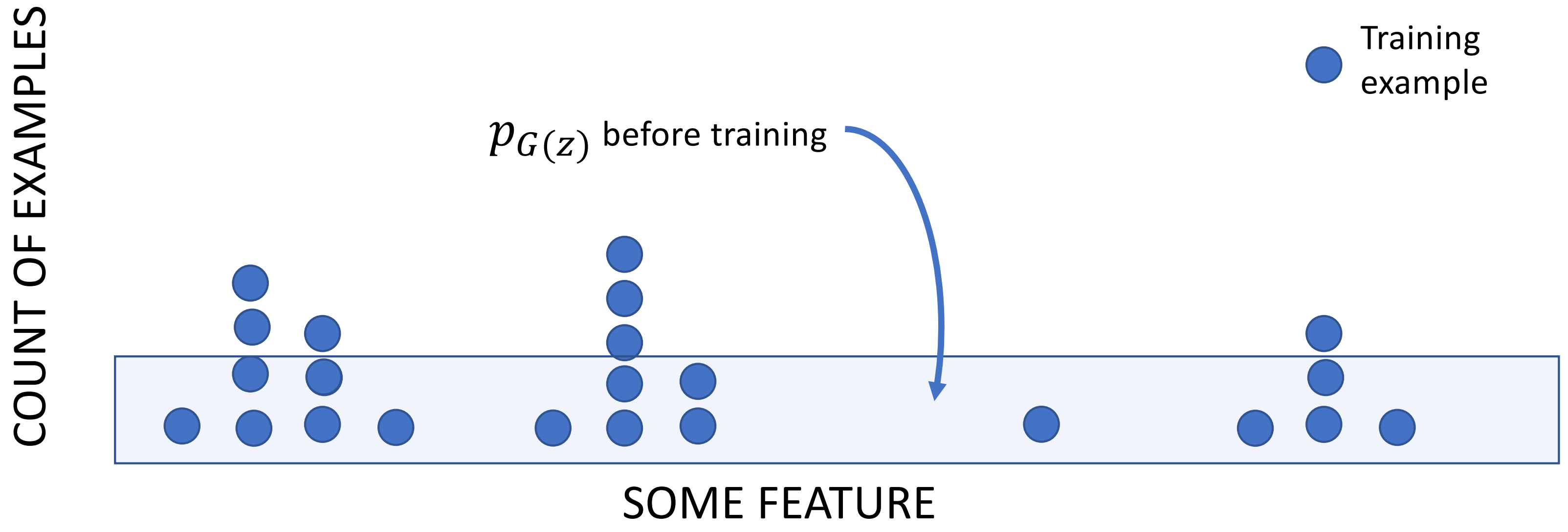
Our loss function is: $Loss = -\log(D(G(z))$



Modeling the p_{data} distribution with G(z)

$p_{G(z)}$ is the Generator's implicit estimate of p_{data}

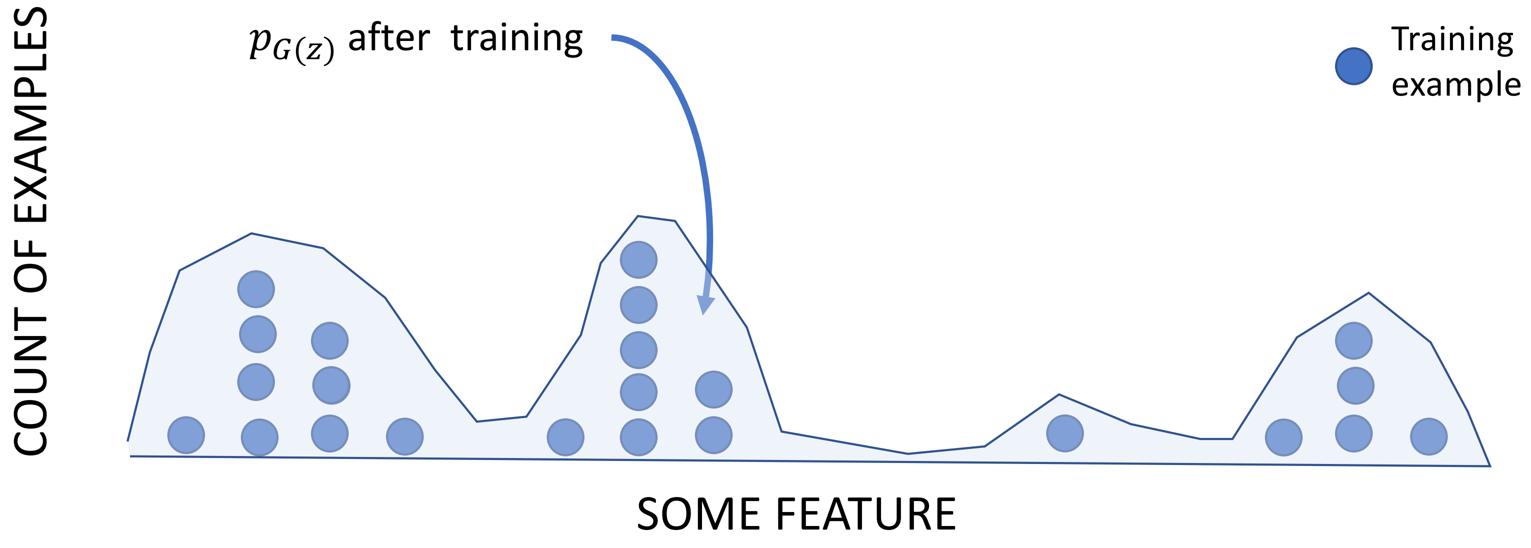
Our loss function is: $Loss = -\log(D(G(z))$



One possible outcome

$p_{G(z)}$ is the Generator's estimate of p_{data}

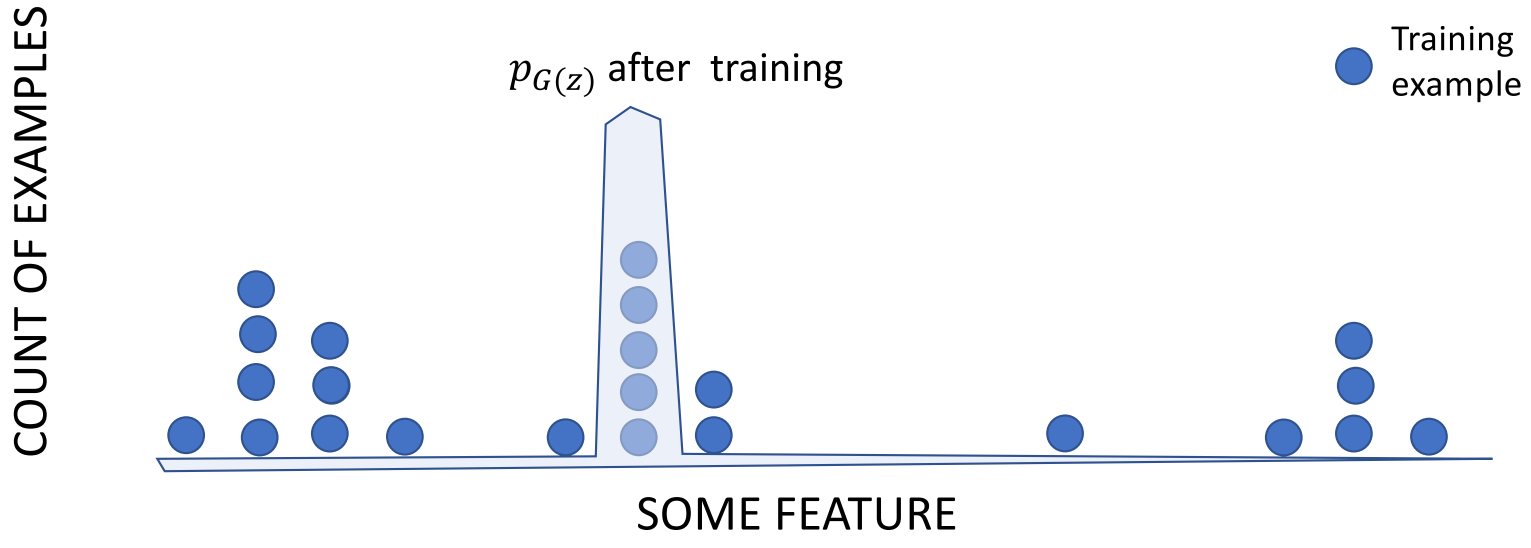
Our loss function is: $Loss = -\log(D(G(z))$



Another possible outcome: Mode Collapse

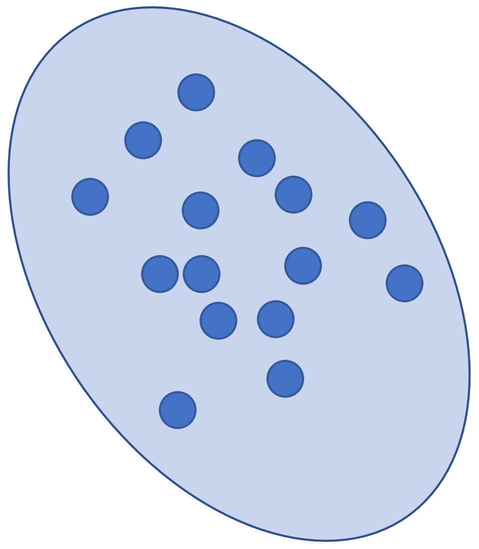
Our loss function is: $Loss = -\log(D(G(z)))$

Is anything stopping this outcome?



Explicitly model the distribution?

- Fit a Gaussian distribution to the training data in the embedding space.
- Variational Auto Encoders (VAEs) do this
- How does that let me generate new example pictures?
- That is for another course…

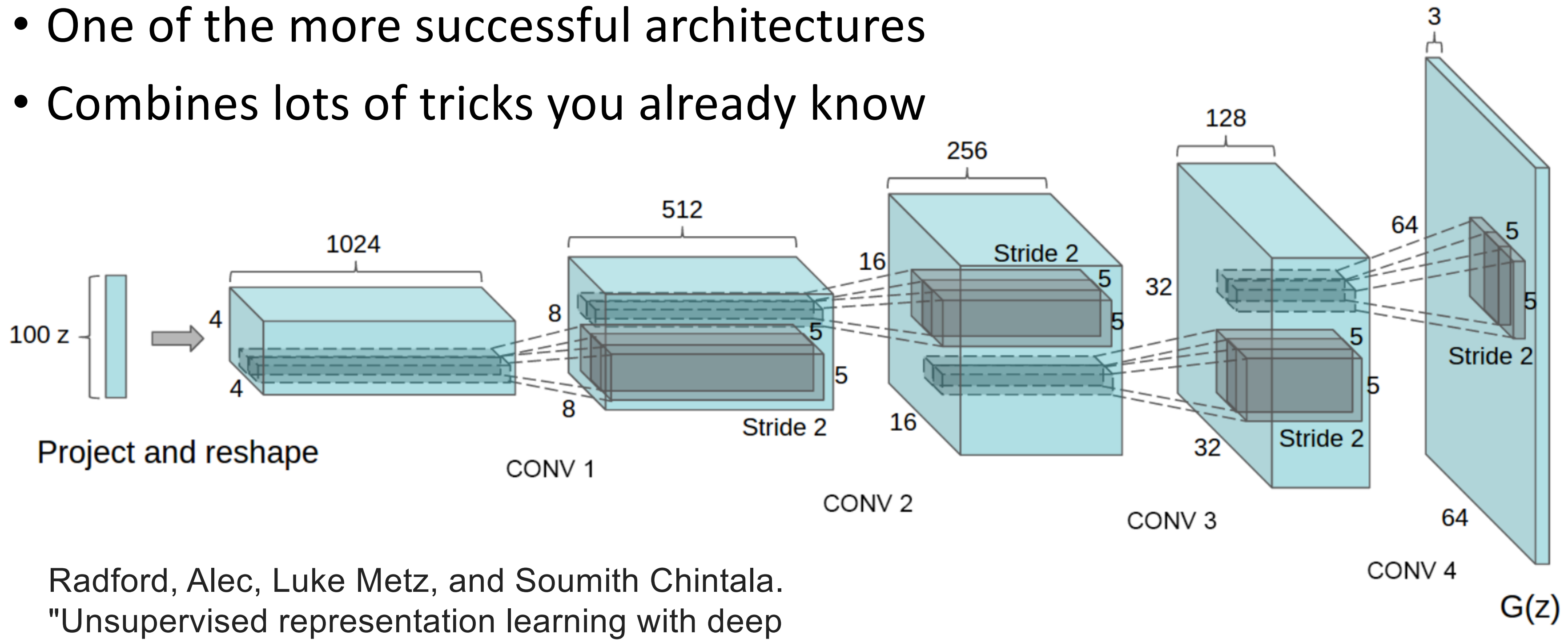


What should D and G look like?

- There is nothing in a GAN formulation that says *what kind* of network is embodied by G or by D.
- The design space is huge. How do we pick?
- Short story:
  - Try a lot of things.
  - Read a lot of papers by other people that tried a lot of things.
  - Borrow the best architecture you can find.
  - Modify to make it work for your problem.

DC GAN: Deep Convolutional GAN

- One of the more successful architectures
- Combines lots of tricks you already know

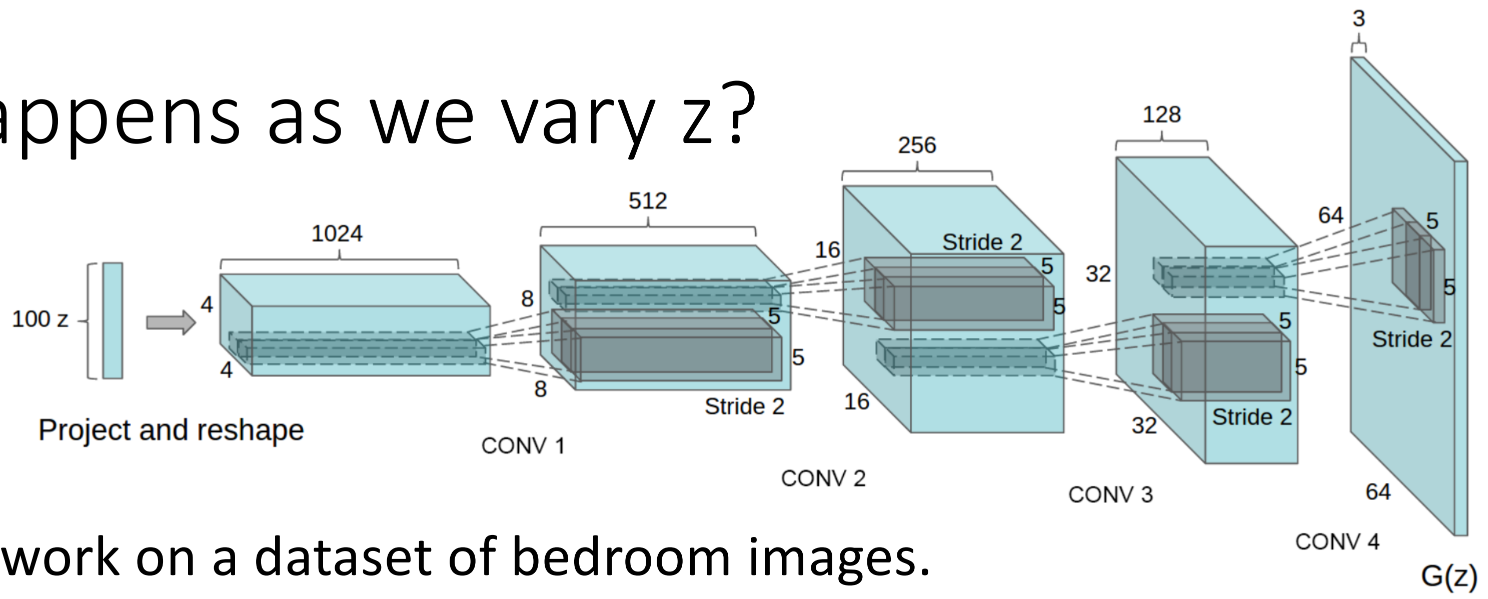


Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." ICLR 2016

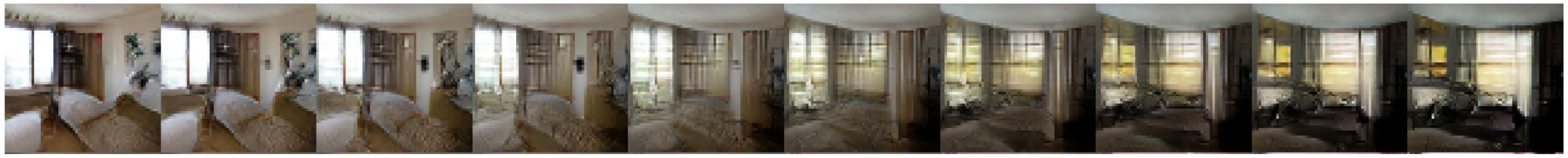
DC GAN: Architectural Guide

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batch norm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

What happens as we vary z?



- Train the network on a dataset of bedroom images.
- Pick 2 values for z and interpolate between them.
- Run these interpolated values through the network. Results are below.



Fun with vector math on z.

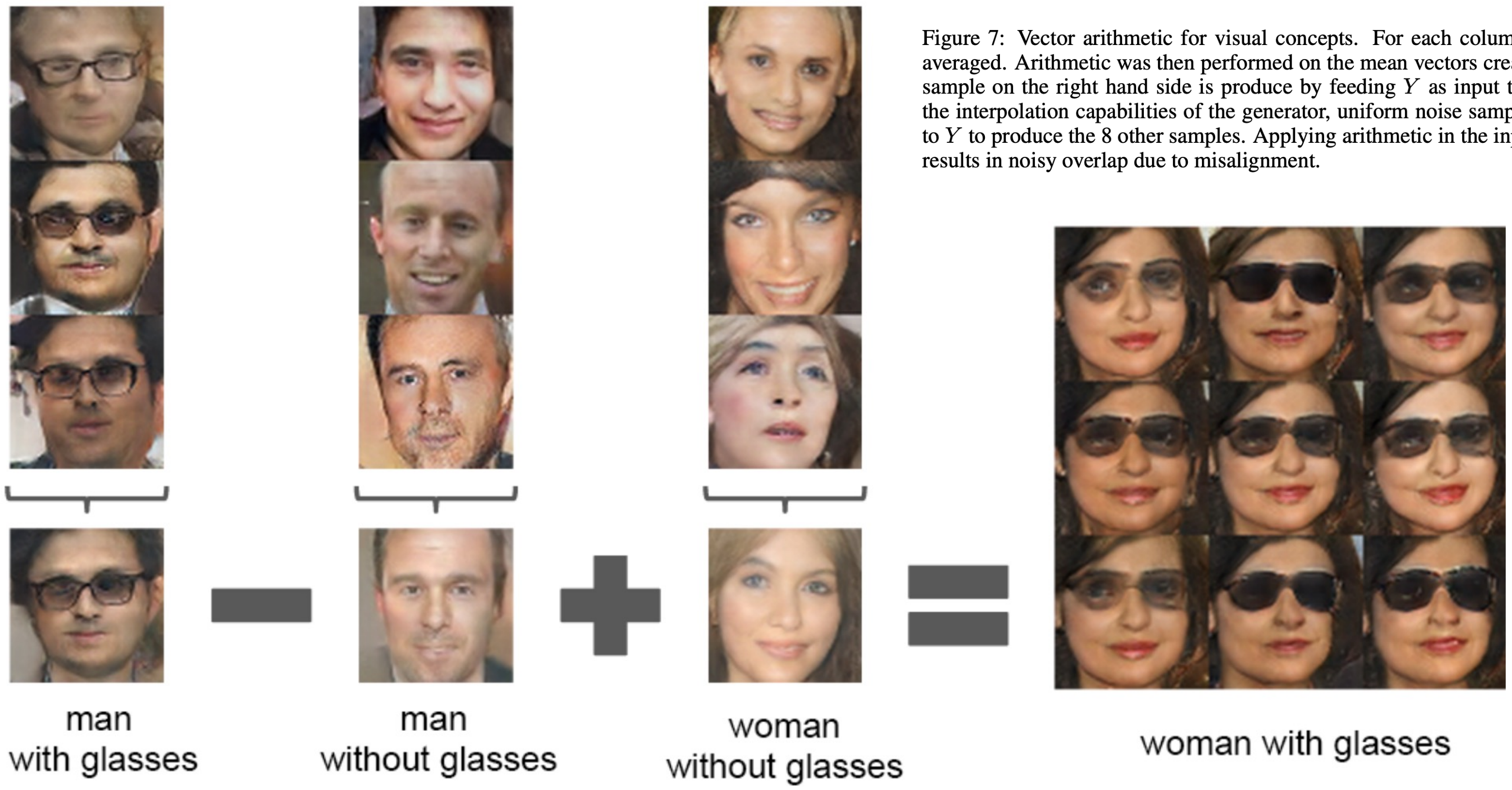
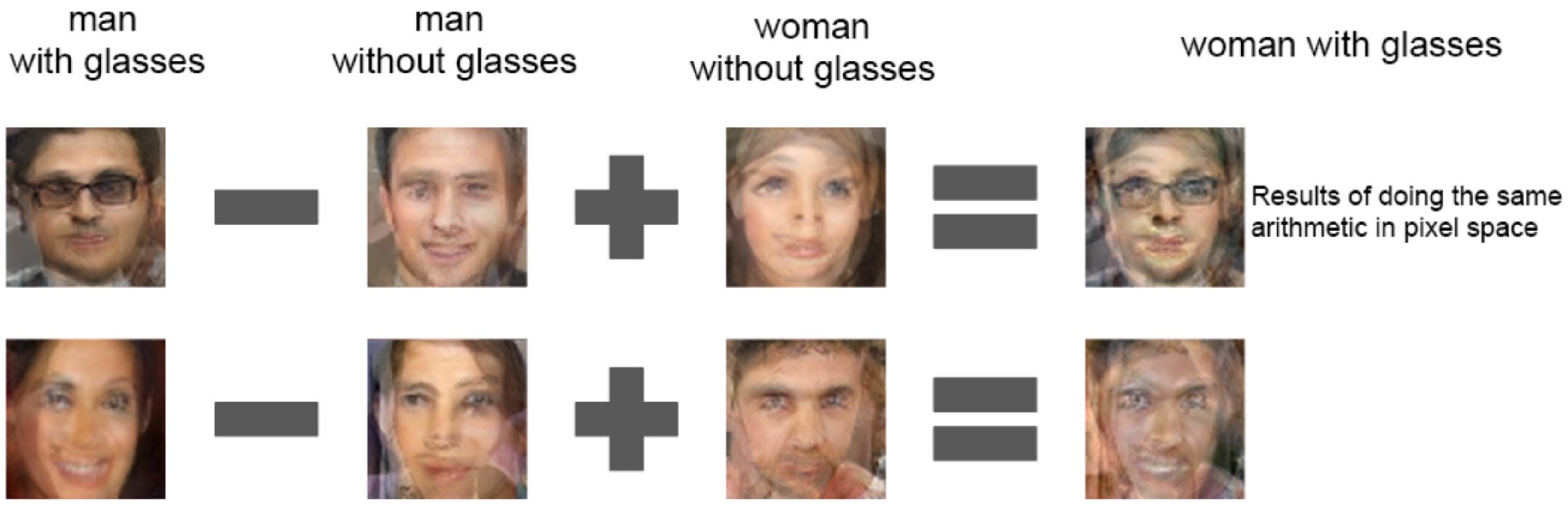
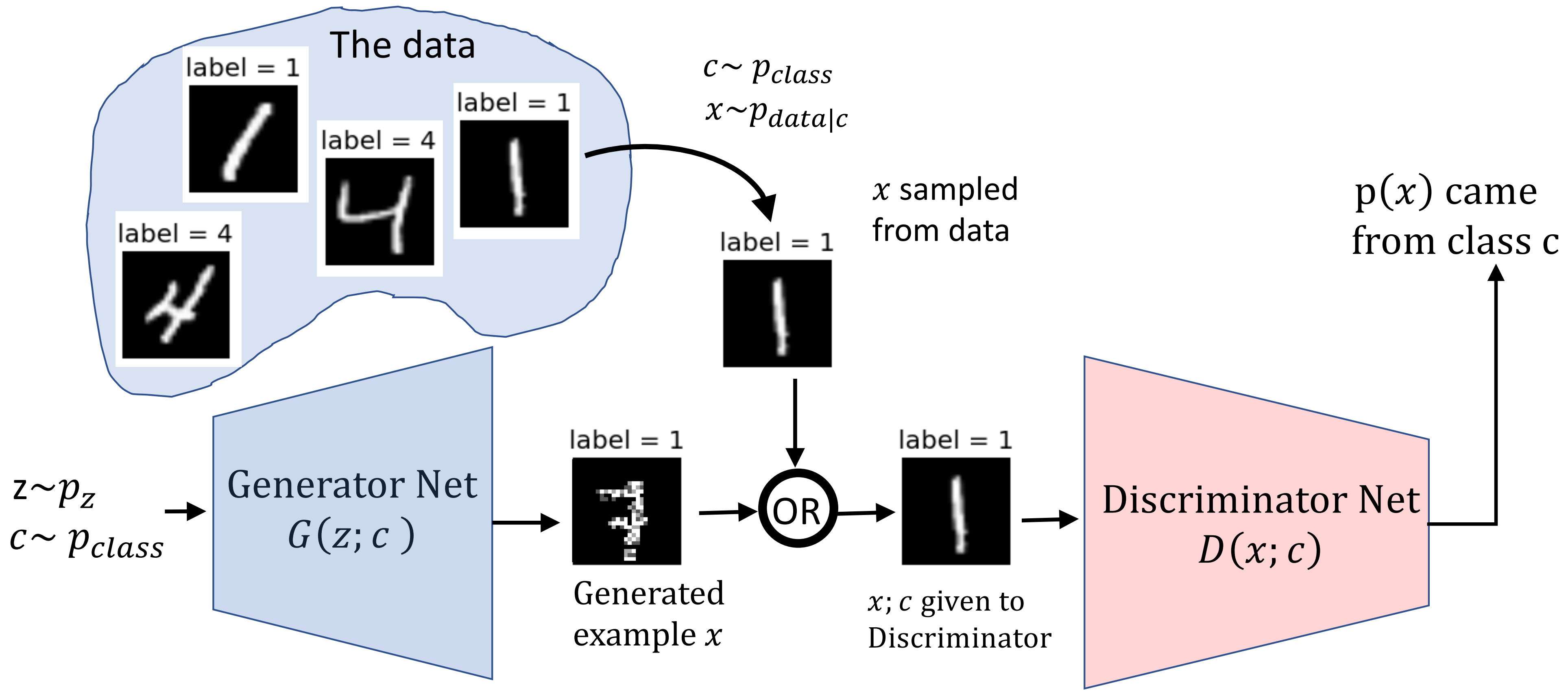


Figure 7: Vector arithmetic for visual concepts. For each column, the Z vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector Y. The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples. Applying arithmetic in the input space (bottom two examples) results in noisy overlap due to misalignment.

That same thing again, but in pixel space



Add class conditioning



Now we can make a variety of images!

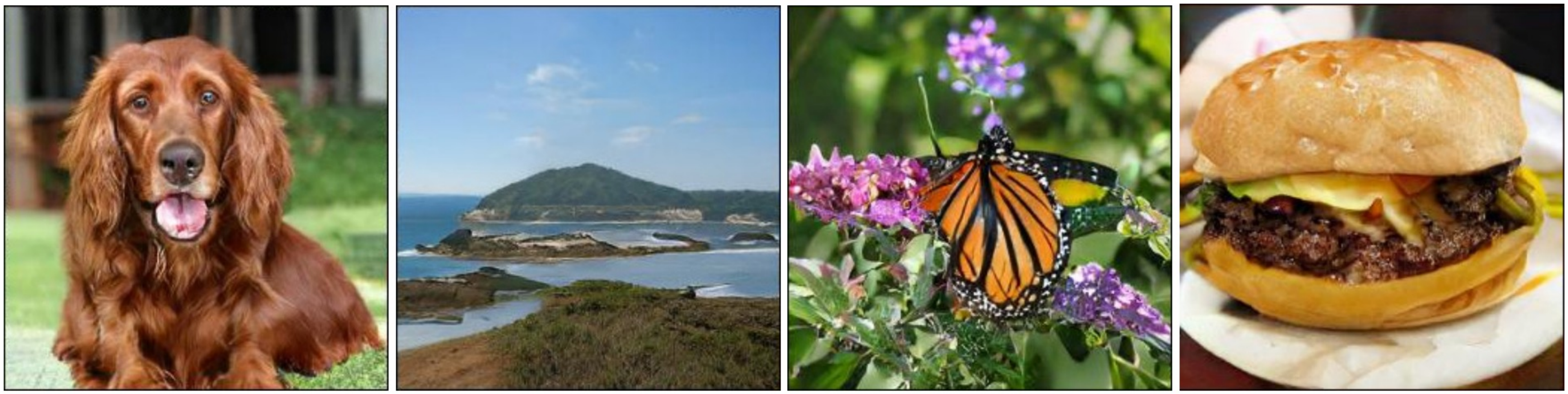
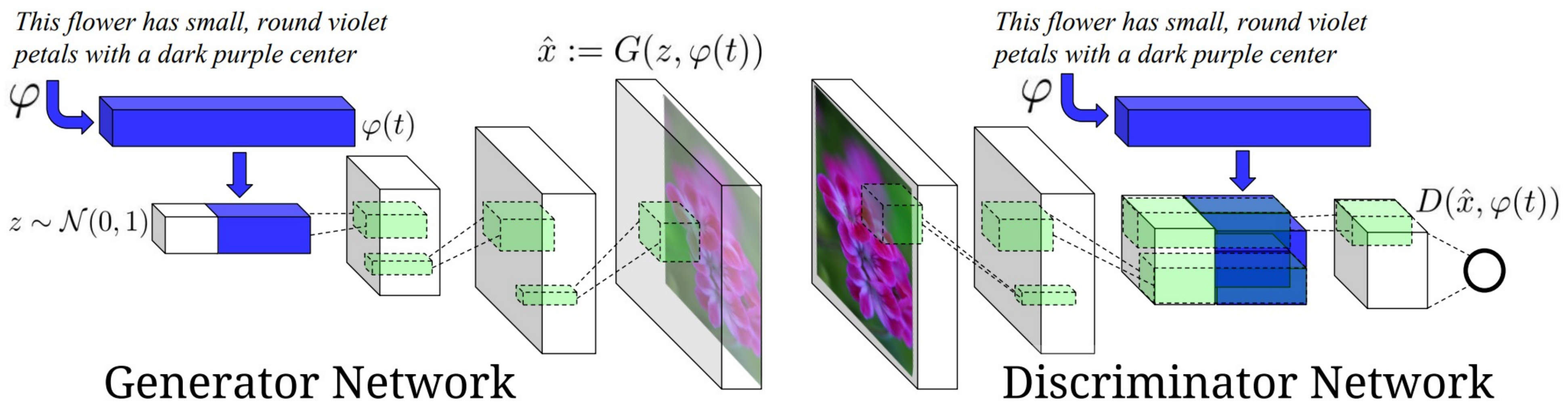


Figure 1: Class-conditional samples generated by our model.

Brock, Andrew, Jeff Donahue, and Karen Simonyan. "Large scale GAN training for high fidelity natural image synthesis." ICLR 2019

https://arxiv.org/pdf/1809.11096.pdf

What if we condition on text?


This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Generator Network
Discriminator Network

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B. &amp; Lee, H.. (2016). Generative Adversarial Text to Image Synthesis. ICML 2016

http://proceedings.mlr.press/v48/reed16.pdf

The algorithm + example output

Algorithm 1 GAN-CLS training algorithm with step size α, using minibatch SGD for simplicity.

```
1: Input: minibatch images $x$, matching text $t$, mis-
   matching $\hat{t}$, number of training batch steps $S$
2: for $n = 1$ to $S$ do
3:     $h \leftarrow \varphi(t)$ {Encode matching text description}
4:     $\hat{h} \leftarrow \varphi(\hat{t})$ {Encode mis-matching text description}
5:     $z \sim \mathcal{N}(0,1)^Z$ {Draw sample of random noise}
6:     $\hat{x} \leftarrow G(z, h)$ {Forward through generator}
7:     $s_r \leftarrow D(x, h)$ {real image, right text}
8:     $s_w \leftarrow D(x, \hat{h})$ {real image, wrong text}
9:     $s_f \leftarrow D(\hat{x}, h)$ {fake image, right text}
10:    $\mathcal{L}_D \leftarrow \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2$
11:    $D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ {Update discriminator}
12:    $\mathcal{L}_G \leftarrow \log(s_f)$
13:    $G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ {Update generator}
14: end for
```

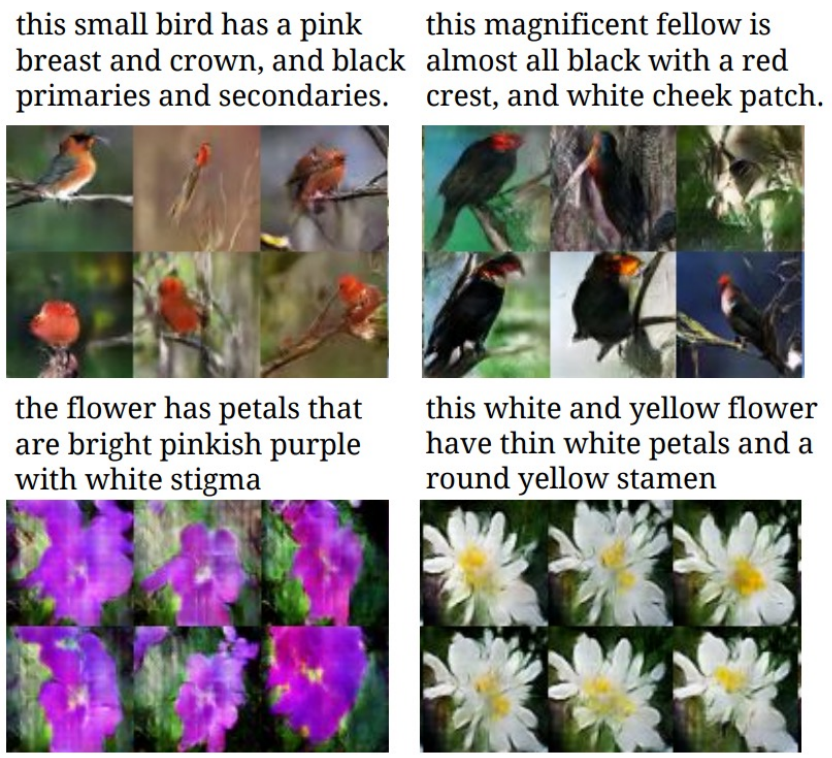

this small bird has a pink breast and crown, and black primaries and secondaries.
this magnificent fellow is almost all black with a red crest, and white cheek patch.
the flower has petals that are bright pinkish purple with white stigma
this white and yellow flower have thin white petals and a round yellow stamen

Figure 1. Examples of generated images from text descriptions. Left: captions are from zero-shot (held out) categories, unseen text. Right: captions are from the training set.

In fact, you could condition on MANY things

- Condition on text to output images
- Condition on text to generate speech
- Condition on photos to output maps
- Condition on drawings to output “photos”
- Try this demo: https://affinelayer.com/pixsrv/
- Check out this overview of GAN projects:

  https://jonathan-hui.medium.com/gan-some-cool-applications-of-gans-4c9ecca35900

Generative Models are a big topic

- Too big for 1 week. There's a class on this in the fall.
- Covers things like…
  - GANs
  - Variational Auto Encoders (VAEs)
  - Transformers
  - Diffusion Models
  - Mixing and matching these approaches