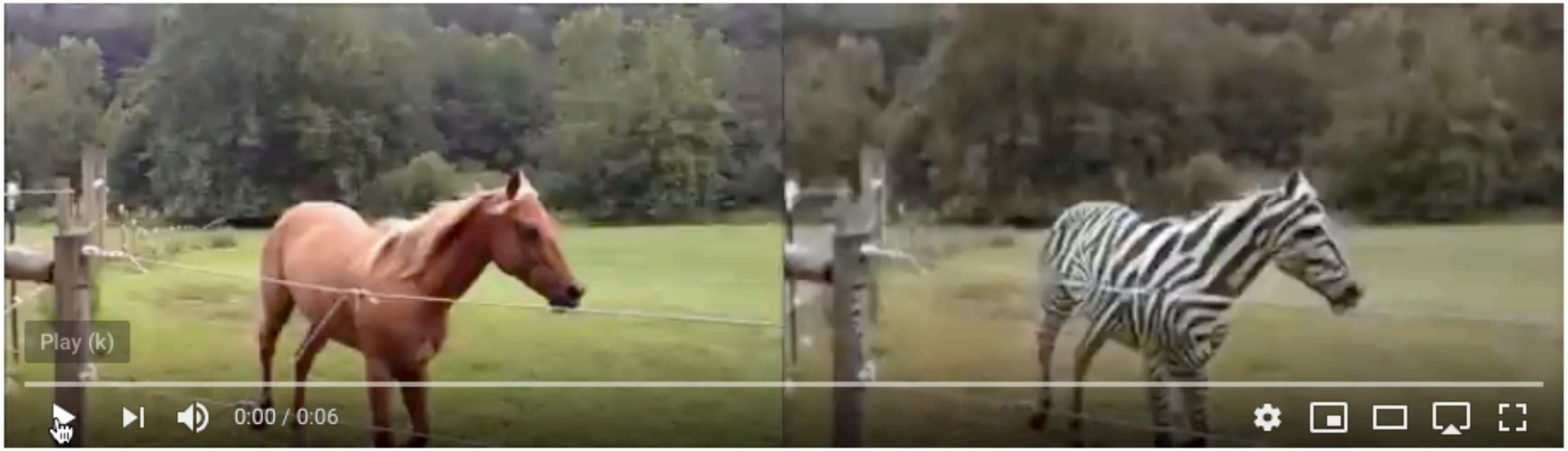
Lecture 17

A Brief Introduction to Generative Adversarial Networks (GAN)

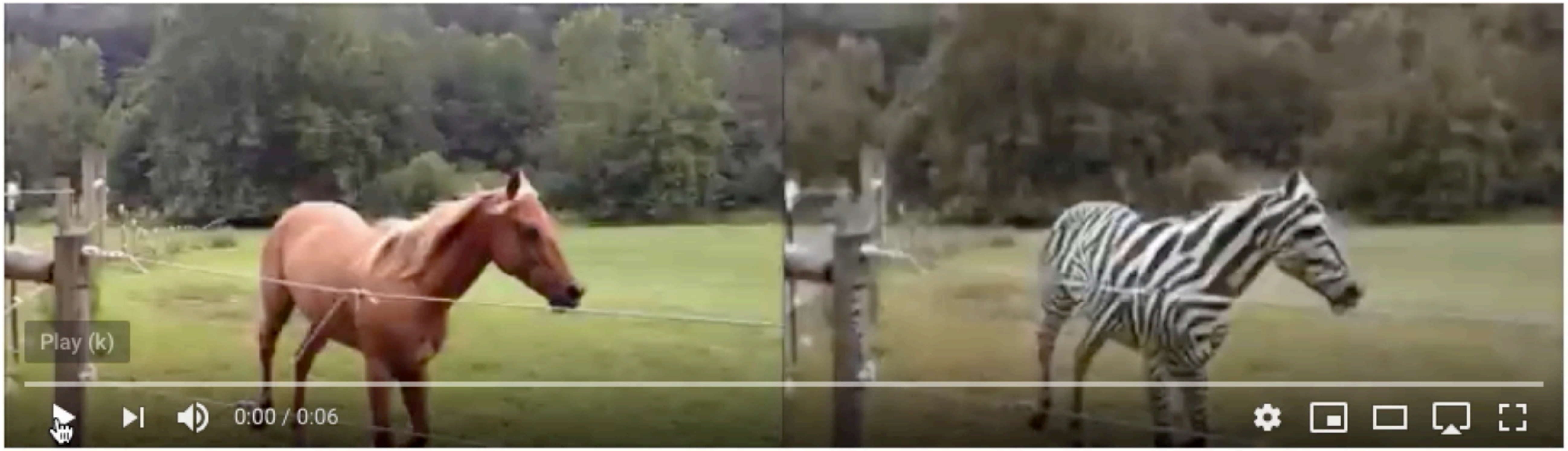
STAT 479: Deep Learning, Spring 2019

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat479-ss2019/



https://github.com/junyanz/CycleGAN
https://www.youtube.com/watch?v=9reHvktowLY

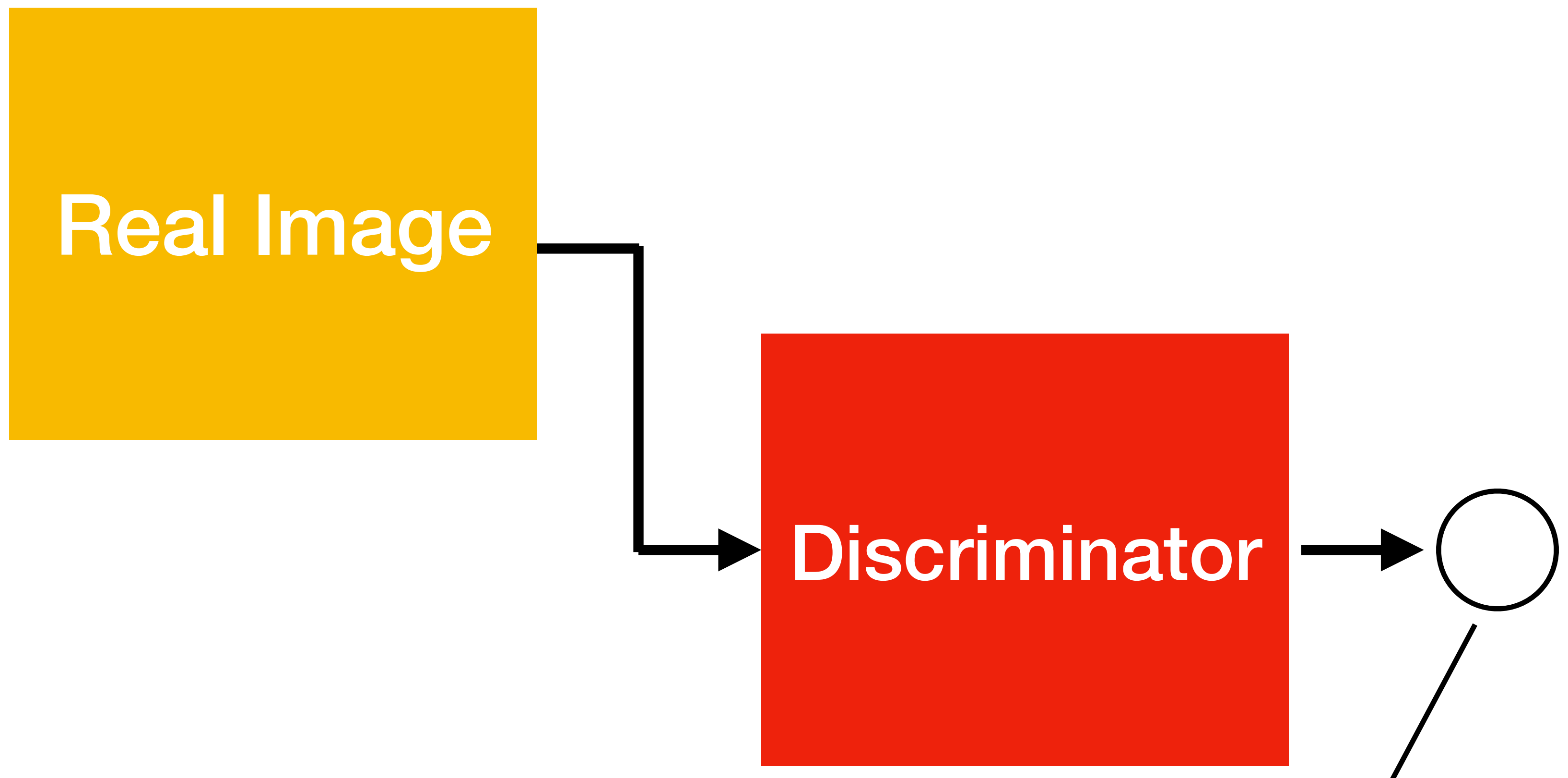


https://github.com/junyanz/CycleGAN
https://www.youtube.com/watch?v=9reHvktowLY

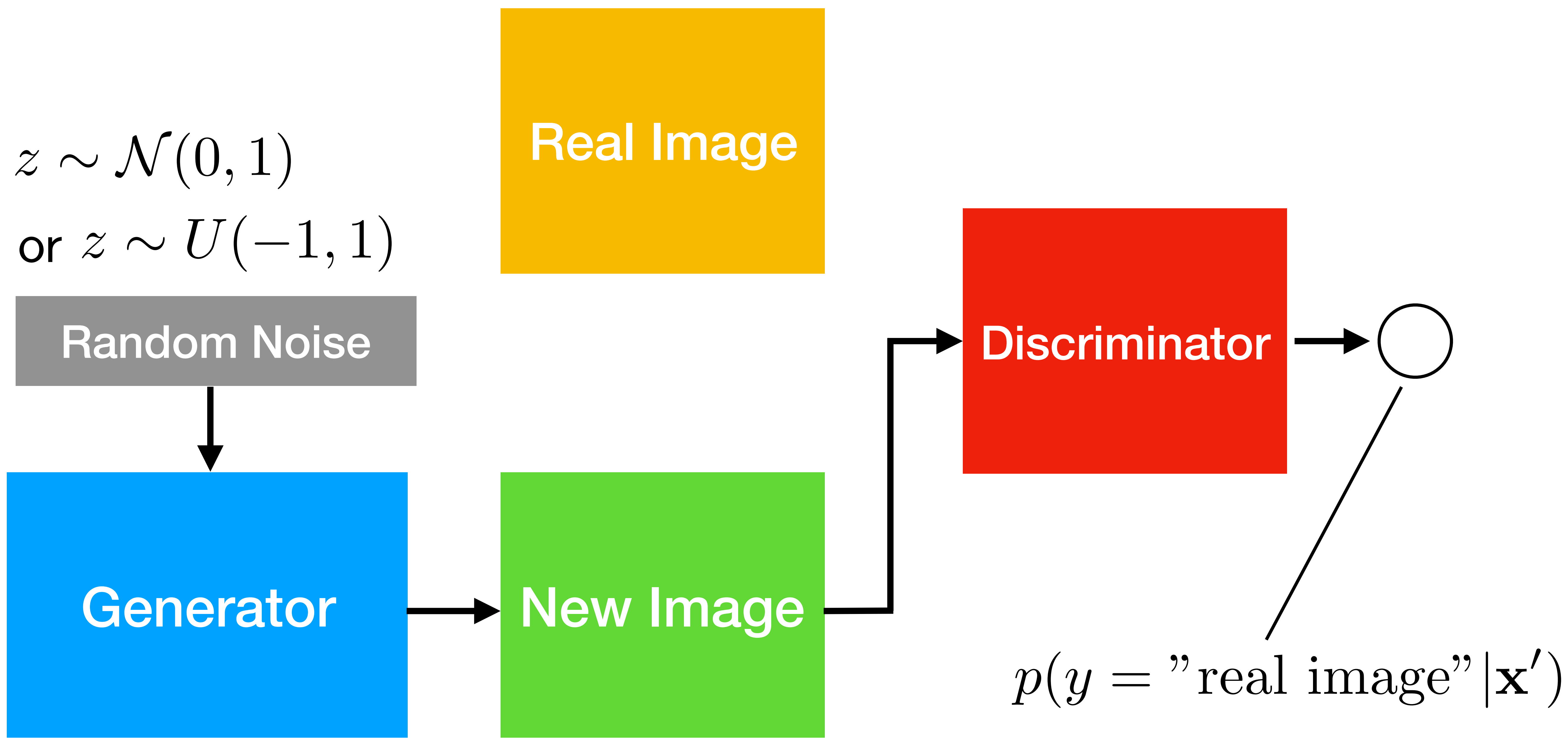
Question: Why does the model change the background as well?

Generative Adversarial Networks (GAN)

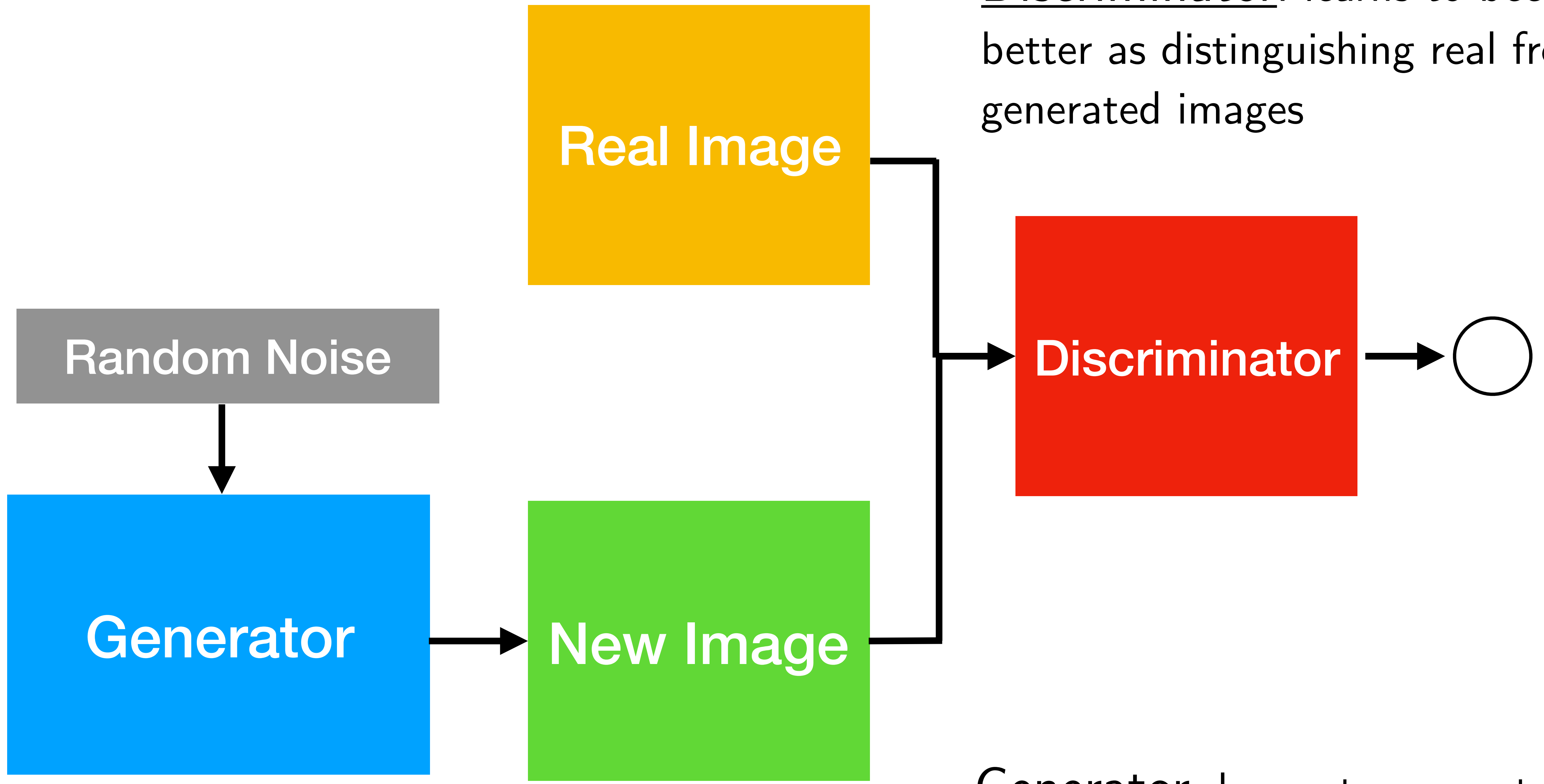
- The original purpose is to generate new data
- Classically for generating new images, but applicable to wide range of domains
- Learns the training set distribution and can generate new images that have never been seen before
- In contrast to e.g., autoregressive models or RNNs (generating one word at a time), GANs generate the whole output all at once



$$p(y = \text{"real image"} | \mathbf{x})$$


$z \sim \mathcal{N}(0,1)$
or $z \sim U(-1,1)$
Random Noise
Real Image
Generator
New Image
Discriminator
$p(y = \text{"real image"} | \mathbf{x}')$

Adversarial Game

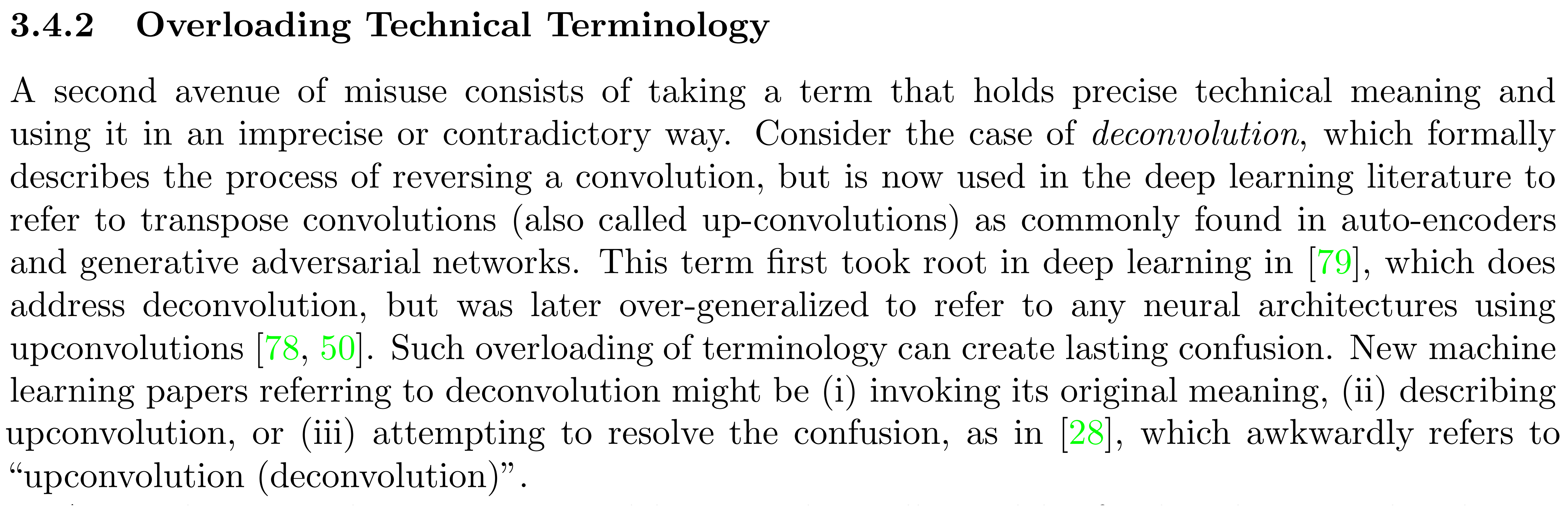


Discriminator: learns to become better as distinguishing real from generated images

Generator: learns to generate better images to fool the discriminator

Sidenote: Transposed Convolutions (Last Lecture)

Lipton, Z. C., & Steinhardt, J. (2018). Troubling Trends in Machine Learning Scholarship. arXiv preprint arXiv:1807.03341.

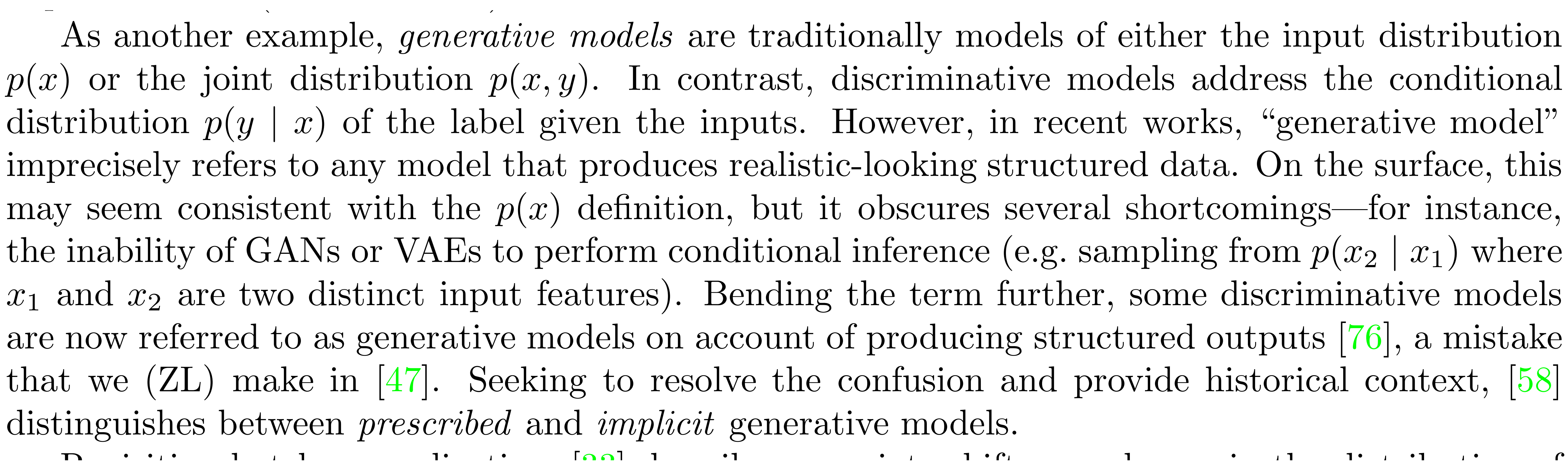


3.4.2 Overloading Technical Terminology

A second avenue of misuse consists of taking a term that holds precise technical meaning and using it in an imprecise or contradictory way. Consider the case of *deconvolution*, which formally describes the process of reversing a convolution, but is now used in the deep learning literature to refer to transpose convolutions (also called up-convolutions) as commonly found in auto-encoders and generative adversarial networks. This term first took root in deep learning in [79], which does address deconvolution, but was later over-generalized to refer to any neural architectures using upconvolutions [78, 50]. Such overloading of terminology can create lasting confusion. New machine learning papers referring to deconvolution might be (i) invoking its original meaning, (ii) describing upconvolution, or (iii) attempting to resolve the confusion, as in [28], which awkwardly refers to "upconvolution (deconvolution)".

If you wonder how the term "generative" fits in the context of your other statistics classes ...

Lipton, Z. C., & Steinhardt, J. (2018). [Troubling Trends in Machine Learning Scholarship](). arXiv preprint arXiv:1807.03341.

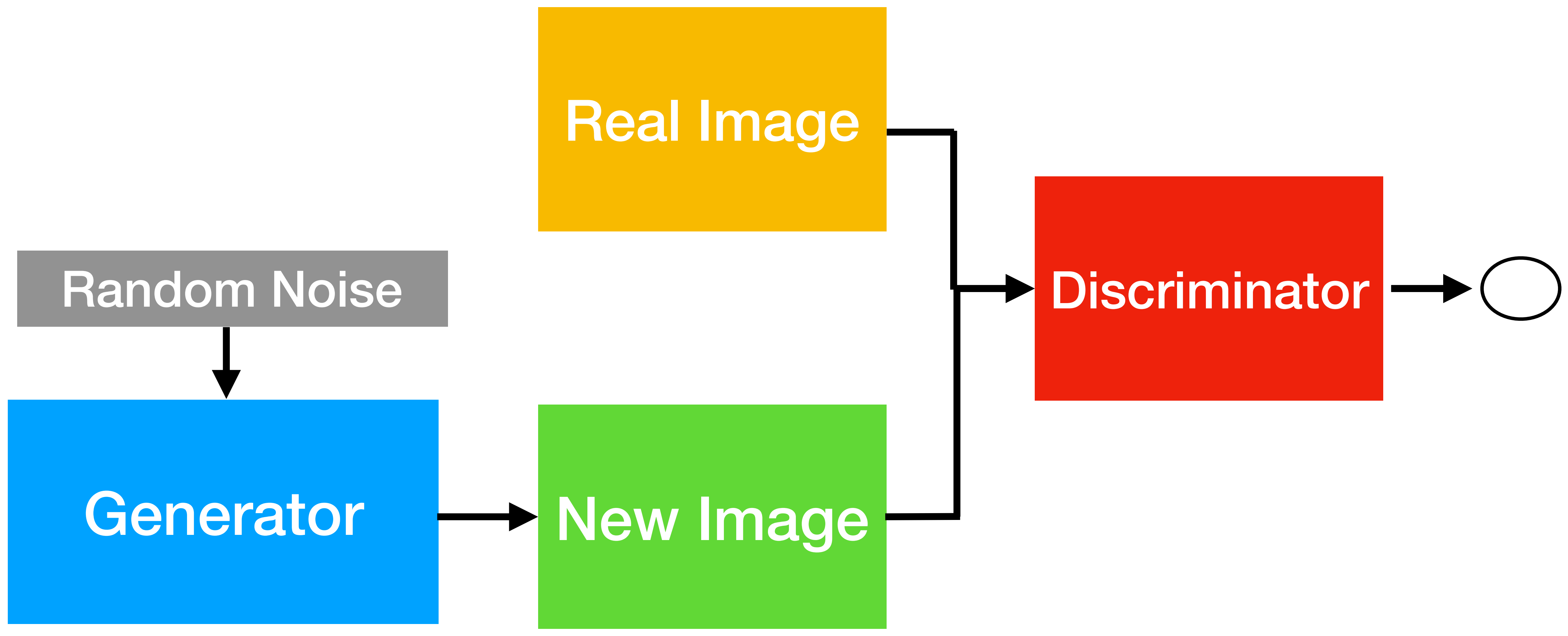


As another example, *generative models* are traditionally models of either the input distribution $p(x)$ or the joint distribution $p(x, y)$. In contrast, discriminative models address the conditional distribution $p(y \mid x)$ of the label given the inputs. However, in recent works, "generative model" imprecisely refers to any model that produces realistic-looking structured data. On the surface, this may seem consistent with the $p(x)$ definition, but it obscures several shortcomings—for instance, the inability of GANs or VAEs to perform conditional inference (e.g. sampling from $p(x_2 \mid x_1)$ where x_1 and x_2 are two distinct input features). Bending the term further, some discriminative models are now referred to as generative models on account of producing structured outputs [76], a mistake that we (ZL) make in [47]. Seeking to resolve the confusion and provide historical context, [58] distinguishes between *prescribed* and *implicit* generative models.

Why Are GANs Are Called Generative Models?

- The generative part comes from the fact that the model "generates" new data
- Usually, generative models use an approximation to compute the usually intractable distribution; here, the discriminator part does that approximation
- So, it does learn p(x)
- Vanilla GANs cannot do conditional inference, though

When Does a GAN Converge?



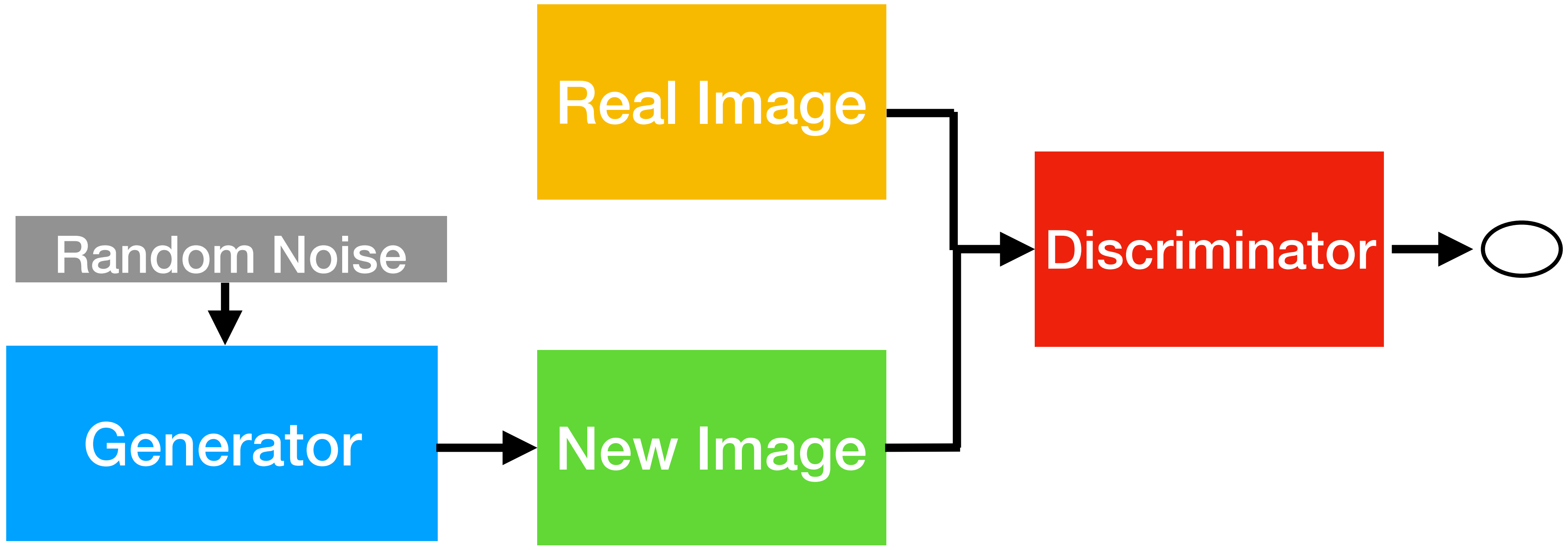
GAN Objective

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

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

Discriminator gradient for update (gradient ascent):

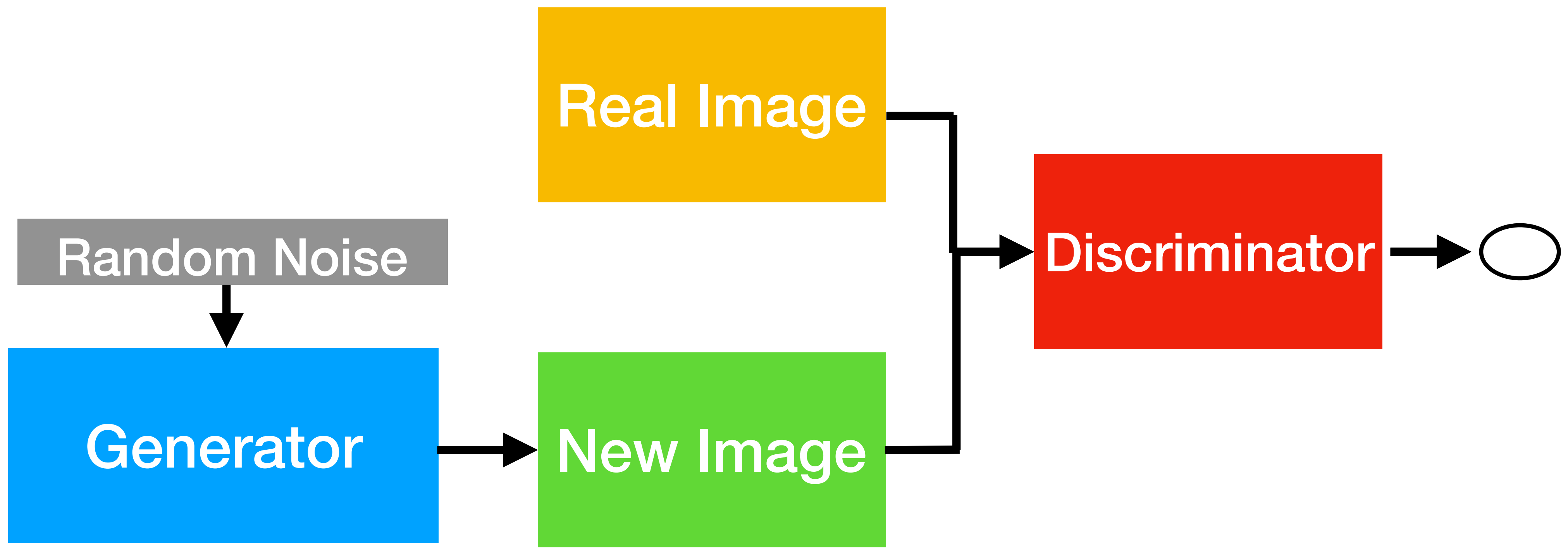
$$\nabla_{\mathbf{W}_D} \frac{1}{n} \sum_{i=1}^{n} \Big[\overbrace{\log D\left(\boldsymbol{x}^{(i)}\right)}^{\substack{\text{predict well on real images} \\ \text{=> probability close to 1}}} + \log \Big(\overbrace{1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)}^{\substack{\text{predict well on fake images} \\ \text{=> probability close to 0}}} \Big) \Big]$$

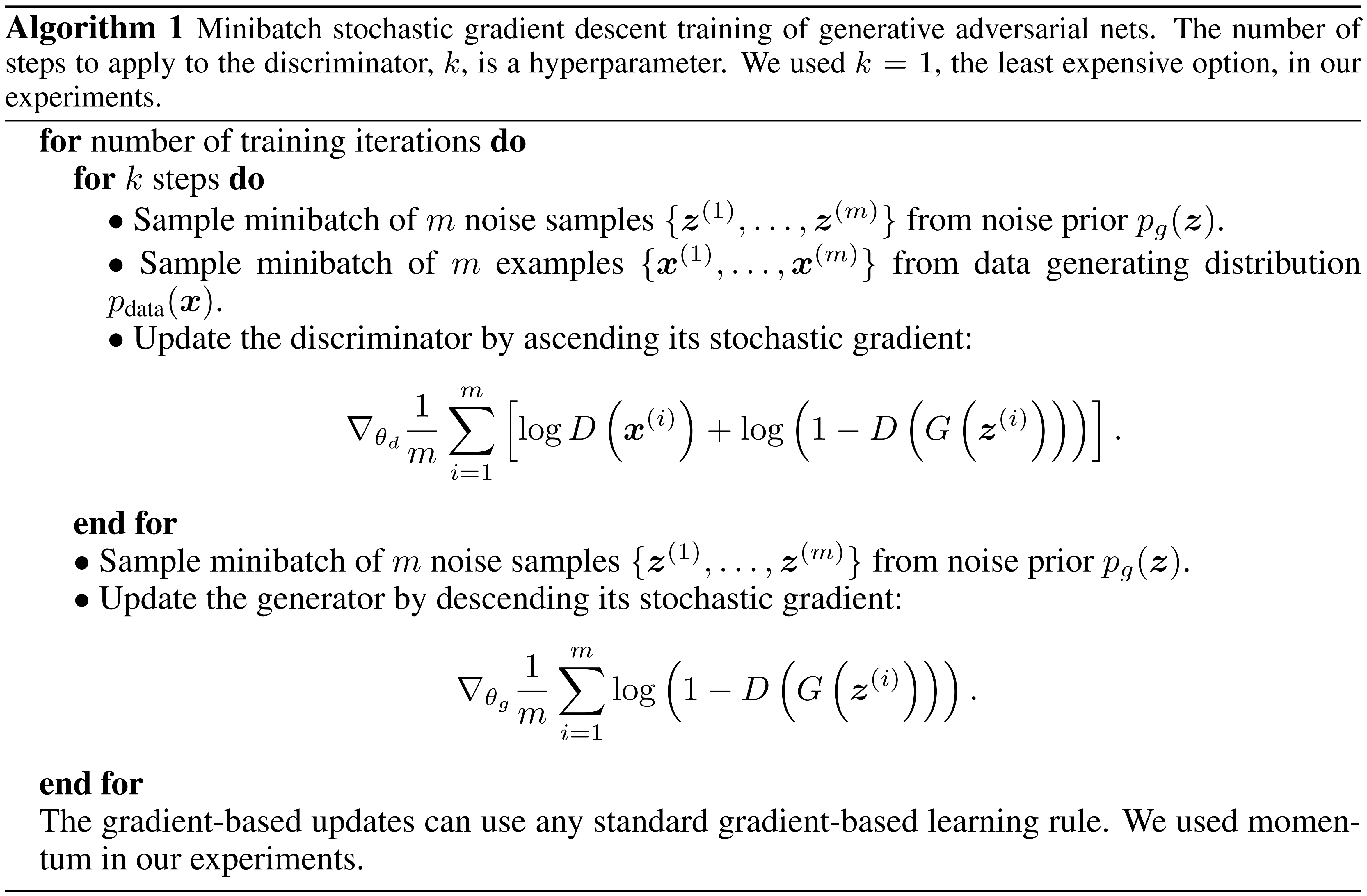


$$\min_G \max_D L(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{tan}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{x}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

<u>Generator gradient for update (gradient descent):</u>

$$\nabla_{\mathbf{W}_G} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 - \overbrace{D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)}^{\substack{\text{predict badly on fake images} \\ \text{=> probability close to 1}}}\right)$$





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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ 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.
```

- Goodfellow, Ian, 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*, pp. 2672-2680. 2014.

GAN Convergence

- Converges when Nash-equilibrium (Game Theory concept) is reached in the minmax (zero-sum) game

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

- Nash-Equilibrium in Game Theory is reached when the actions of one player won't change depending on the opponent's actions
- Here, this means that the GAN produces realistic images and the discriminator outputs random predictions (probabilities close to 0.5)

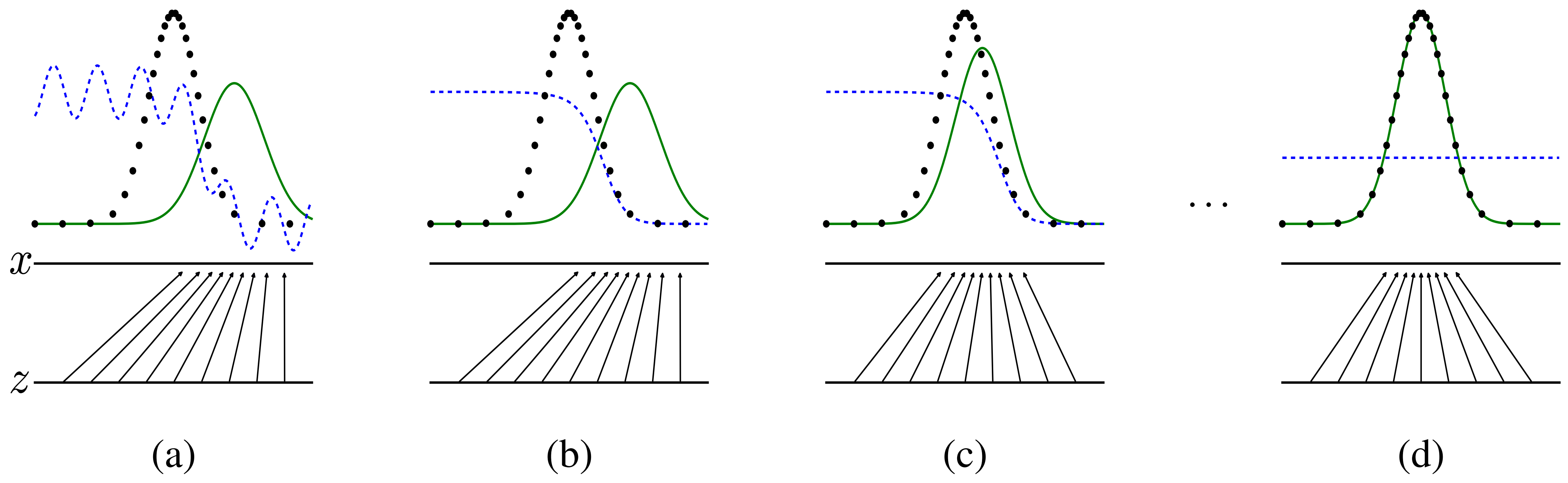


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution 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. $D(\boldsymbol{x}) = \frac{1}{2}$.

- Goodfellow, Ian, 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*, pp. 2672-2680. 2014.

GAN Training Problems

- Oscillation between generator and discriminator loss
- Mode collapse (generator produces examples of a particular kind only)
- Discriminator is too strong, such that the gradient for the generator vanishes and the generator can't keep up
- Discriminator is too weak, and the generator produces non-realistic images that fool it too easily (rare problem, though)

GAN Training Problems

- Discriminator is too strong, such that the gradient for the generator vanishes and the generator can't keep up
- Can be fixed as follows:

Instead of gradient descent with

$$\nabla_{\mathbf{W}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

Do gradient ascent with

$$\nabla_{\mathbf{W}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

GAN Loss Function in Practice (will be more clear in the code examples)

Discriminator

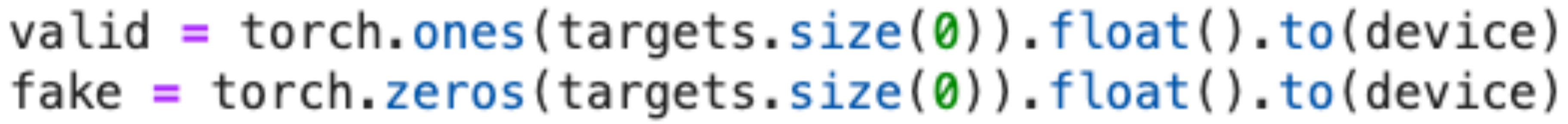
- Maximize prediction probability of real as real and fake as fake
- Remember maximizing log likelihood is the same as minimizing negative log likelihood (i.e., minimizing cross-entropy)

Generator

- Minimize likelihood of the discriminator to make correct predictions (predict fake as fake; real as real), which can be achieved by maximizing the cross-entropy
- This doesn't work well in practice though because of gradient issues (zero gradient if the discriminator makes correct predictions, which is not what we want for the generator)
- Better: flip labels and minimize cross entropy (force the discriminator to output high probability for fake if an image is real, and low probability for real if an image is fake)

GANs In Practice (1)

Setup a vector of 1's for the real images and a vector of 0's for the fake images



```
valid = torch.ones(targets.size(0)).float().to(device)
fake = torch.zeros(targets.size(0)).float().to(device)
```

GANs In Practice (2)

Setup a vector of 1's for the real images and a vector of 0's for the fake images

```
valid = torch.ones(targets.size(0)).float().to(device)
fake = torch.zeros(targets.size(0)).float().to(device)
```

Generate new images from random noise

```
# Make new images
z = torch.zeros((targets.size(0), LATENT_DIM)).uniform_(-1.0, 1.0).to(device)
generated_features = model.generator_forward(z)
```

GANs In Practice (3)

Setup a vector of 1's for the real images and a vector of 0's for the fake images

```
valid = torch.ones(targets.size(0)).float().to(device)
fake = torch.zeros(targets.size(0)).float().to(device)
```

Generate new images from random noise

```
# Make new images
z = torch.zeros((targets.size(0), LATENT_DIM)).uniform_(-1.0, 1.0).to(device)
generated_features = model.generator_forward(z)
```

Generator Loss:

Minimizing likelihood that discriminator makes a correct prediction can be achieved by maximizing likelihood that discriminator makes a wrong prediction (predicting valid images). Maximizing likelihood is the same as minimizing negative log likelihood

```
# Loss for fooling the discriminator
discr_pred = model.discriminator_forward(generated_features.view(targets.size(0), 1, 28, 28))

gener_loss = F.binary_cross_entropy(discr_pred, valid)
```

GANs In Practice (3)

Setup a vector of 1's for the real images and a vector of 0's for the fake images

```
valid
fake
```

Gener

```
# Mak
z = t                                                                      ce)
gener
```

For reference, from Lecture 8:

$$H_{\mathbf{a}}(\mathbf{y}) = -\sum_i \left(y^{[i]} \log(a^{[i]}) + (1 - y^{[i]}) \log(1 - a^{[i]}) \right)$$

Binary Cross Entropy

Generator Loss:

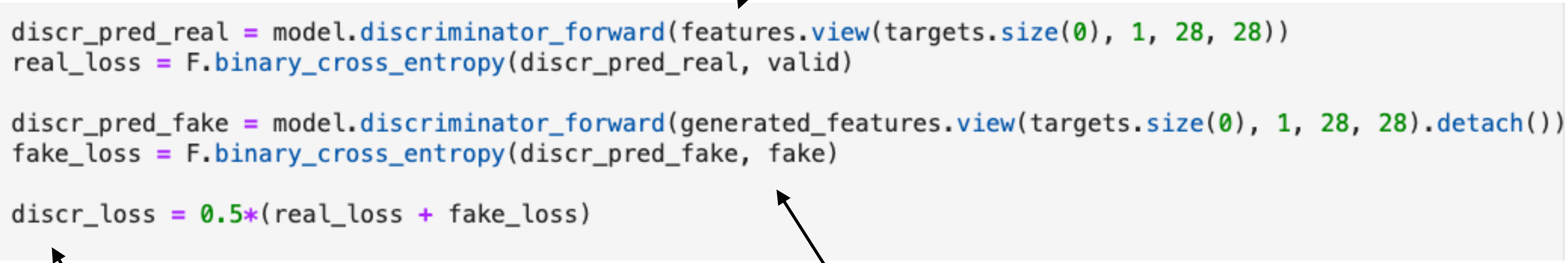
Minimizing likelihood that discriminator makes a correct prediction can be achieved by maximizing likelihood that discriminator makes a wrong prediction (predicting valid images). Maximizing likelihood is the same as minimizing negative log likelihood

```
# Loss for fooling the discriminator
discr_pred = model.discriminator_forward(generated_features.view(targets.size(0), 1, 28, 28))

gener_loss = F.binary_cross_entropy(discr_pred, valid)
```

GANs In Practice (4)

Discriminator Loss:

Train discriminator to recognize real images as real



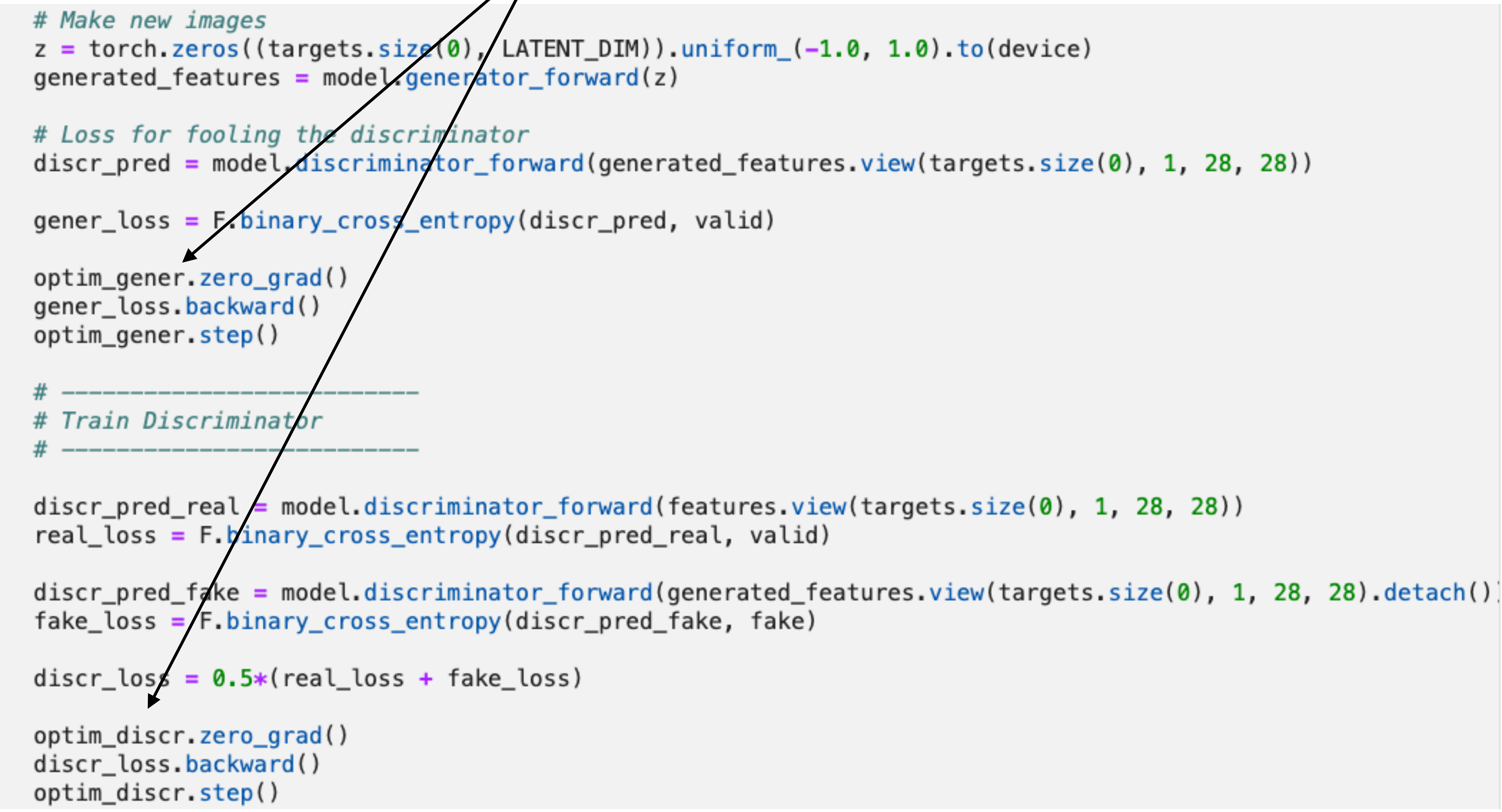
```
discr_pred_real = model.discriminator_forward(features.view(targets.size(0), 1, 28, 28))
real_loss = F.binary_cross_entropy(discr_pred_real, valid)

discr_pred_fake = model.discriminator_forward(generated_features.view(targets.size(0), 1, 28, 28).detach())
fake_loss = F.binary_cross_entropy(discr_pred_fake, fake)

discr_loss = 0.5*(real_loss + fake_loss)
```

Train discriminator to recognize generated images as fake

Combine fake & real parts

GANs In Practice (5)

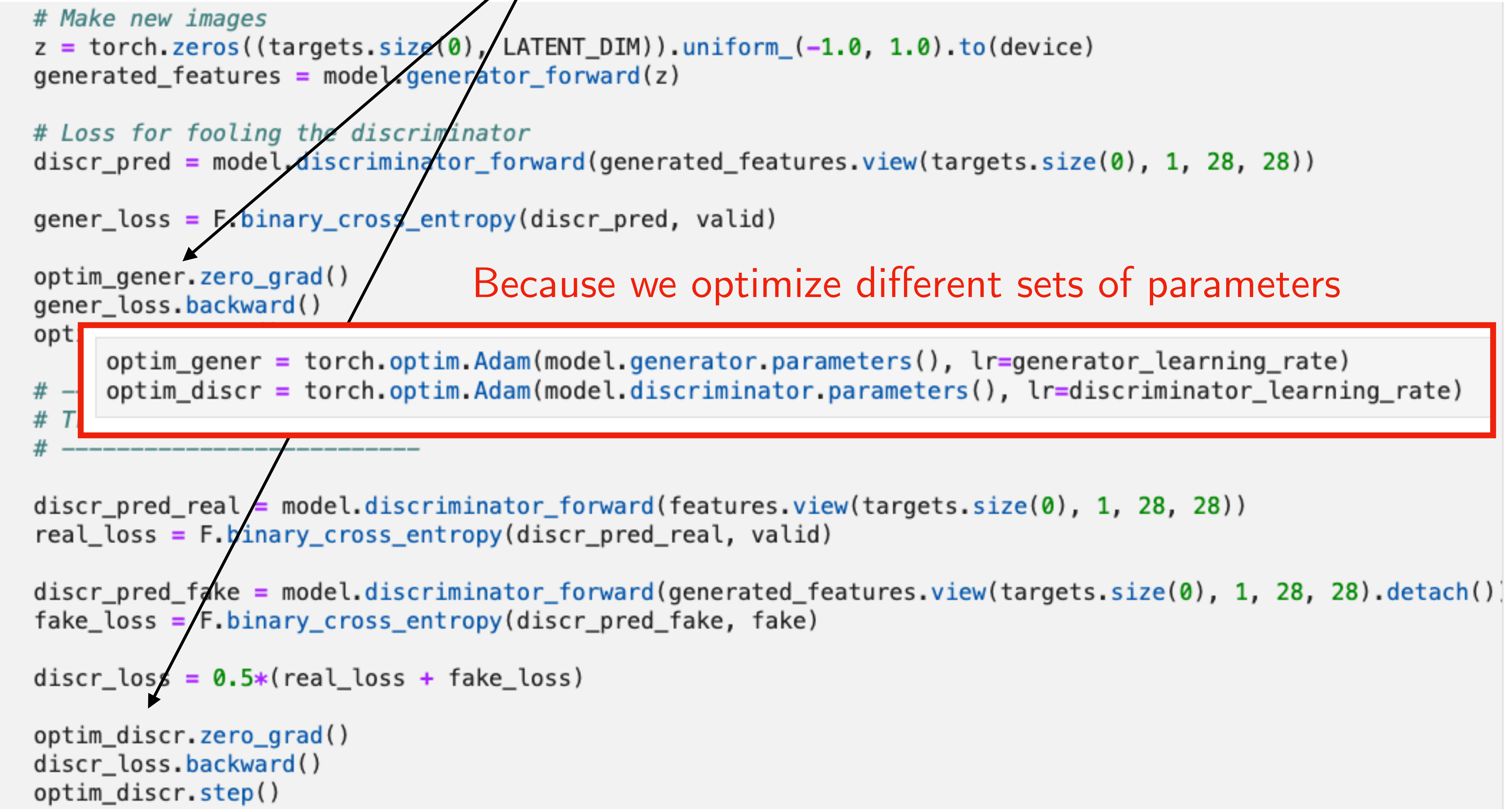
Use separate optimizers for generator and discriminator



```
# Make new images
z = torch.zeros((targets.size(0), LATENT_DIM)).uniform_(-1.0, 1.0).to(device)
generated_features = model.generator_forward(z)

# Loss for fooling the discriminator
discr_pred = model.discriminator_forward(generated_features.view(targets.size(0), 1, 28, 28))

gener_loss = F.binary_cross_entropy(discr_pred, valid)

optim_gener.zero_grad()
gener_loss.backward()
optim_gener.step()

# ---------------------------
# Train Discriminator
# ---------------------------

discr_pred_real = model.discriminator_forward(features.view(targets.size(0), 1, 28, 28))
real_loss = F.binary_cross_entropy(discr_pred_real, valid)

discr_pred_fake = model.discriminator_forward(generated_features.view(targets.size(0), 1, 28, 28).detach()
fake_loss = F.binary_cross_entropy(discr_pred_fake, fake)

discr_loss = 0.5*(real_loss + fake_loss)

optim_discr.zero_grad()
discr_loss.backward()
optim_discr.step()
```

GANs In Practice (5)

Use separate optimizers for generator and discriminator

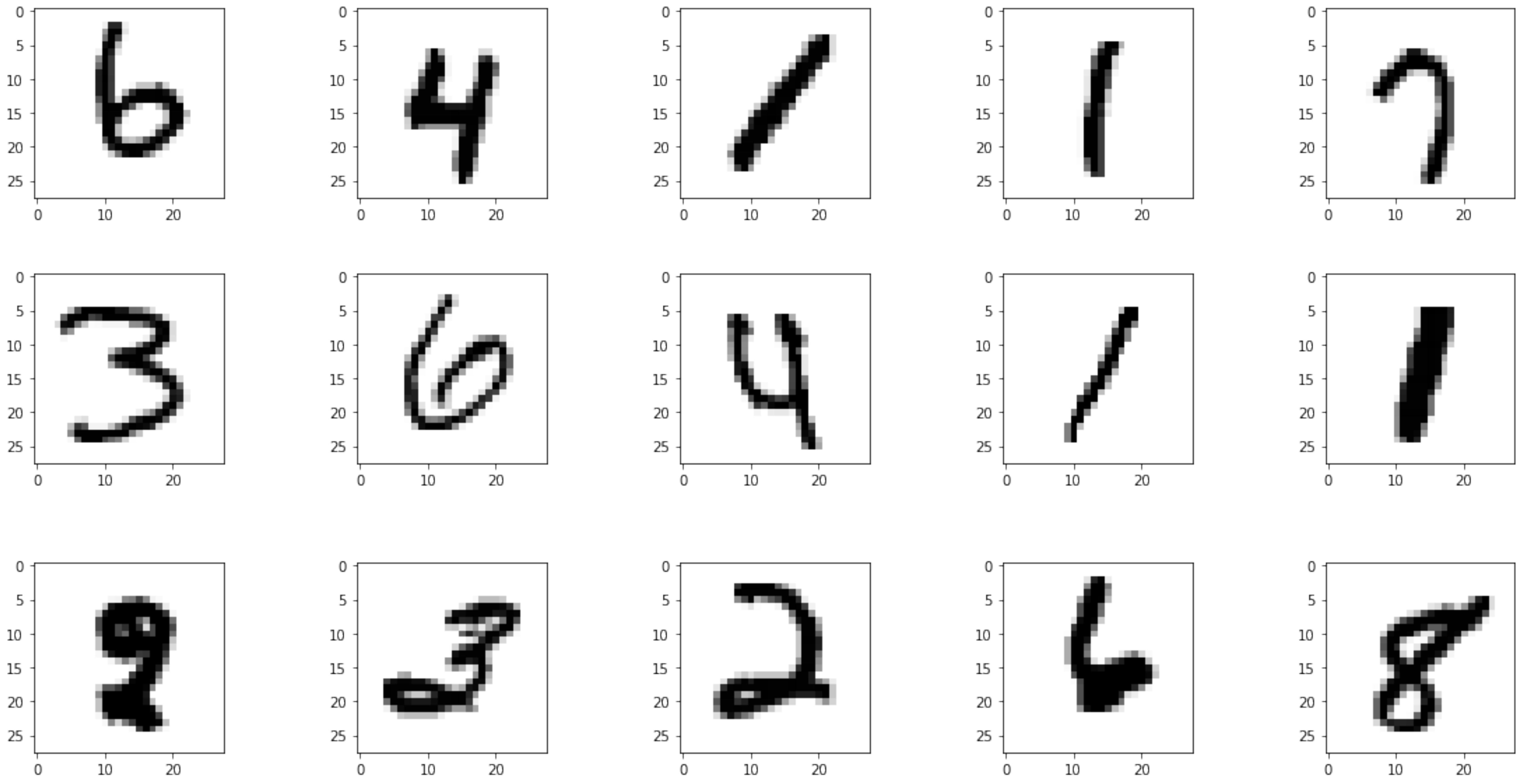


```
# Make new images
z = torch.zeros((targets.size(0), LATENT_DIM)).uniform_(-1.0, 1.0).to(device)
generated_features = model.generator_forward(z)

# Loss for fooling the discriminator
discr_pred = model.discriminator_forward(generated_features.view(targets.size(0), 1, 28, 28))

gener_loss = F.binary_cross_entropy(discr_pred, valid)

optim_gener.zero_grad()
gener_loss.backward()
opt

# -
# T
# -------------------------

discr_pred_real = model.discriminator_forward(features.view(targets.size(0), 1, 28, 28))
real_loss = F.binary_cross_entropy(discr_pred_real, valid)

discr_pred_fake = model.discriminator_forward(generated_features.view(targets.size(0), 1, 28, 28).detach()
fake_loss = F.binary_cross_entropy(discr_pred_fake, fake)

discr_loss = 0.5*(real_loss + fake_loss)

optim_discr.zero_grad()
discr_loss.backward()
optim_discr.step()
```

Because we optimize different sets of parameters

```
optim_gener = torch.optim.Adam(model.generator.parameters(), lr=generator_learning_rate)
optim_discr = torch.optim.Adam(model.discriminator.parameters(), lr=discriminator_learning_rate)
```

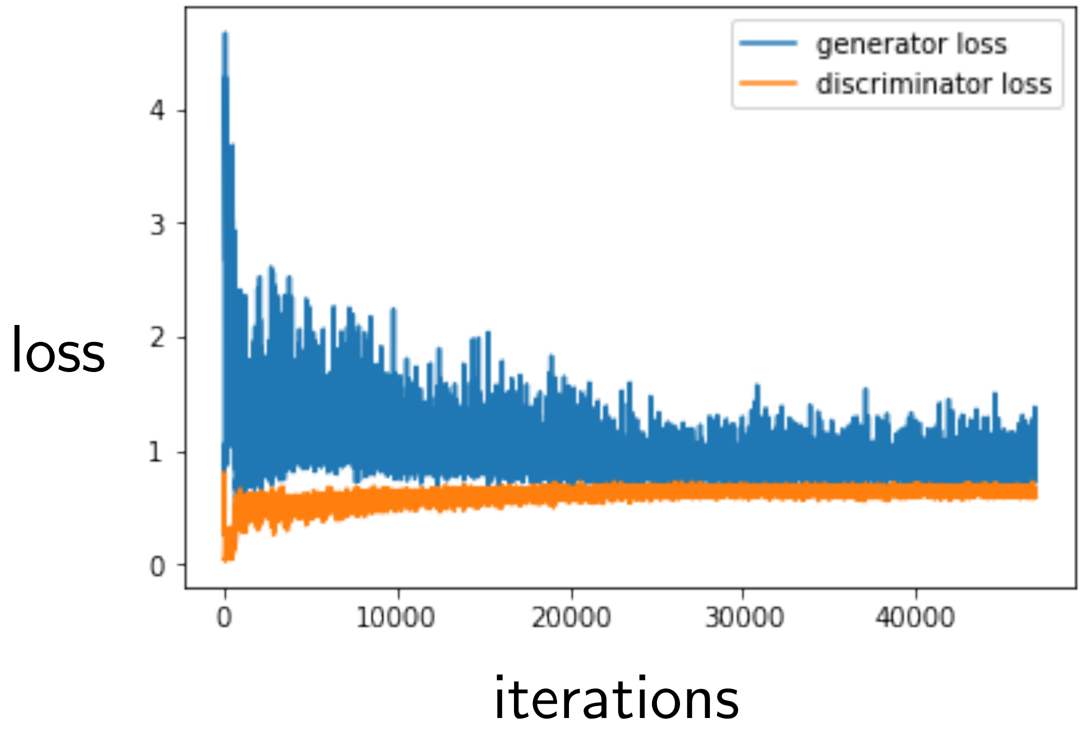
For Reference: Some Real MNIST Images



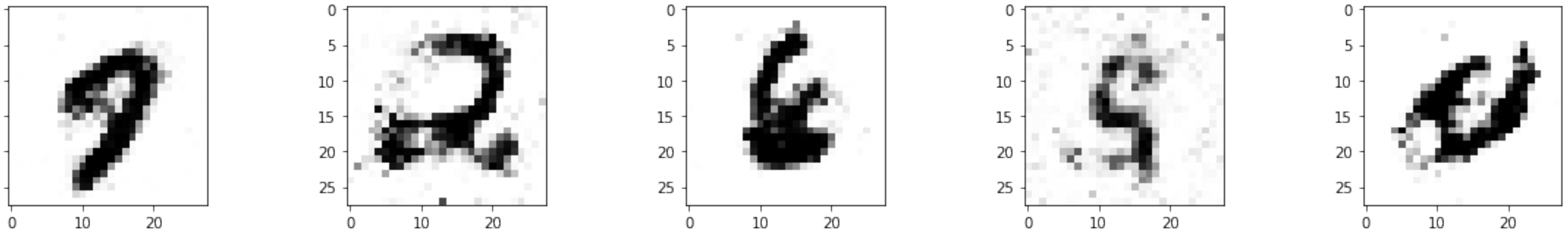
Full Code Examples (1)

MNIST Multilayer Perceptron GAN

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L17_gans/code/gan-mlp-mnist.ipynb



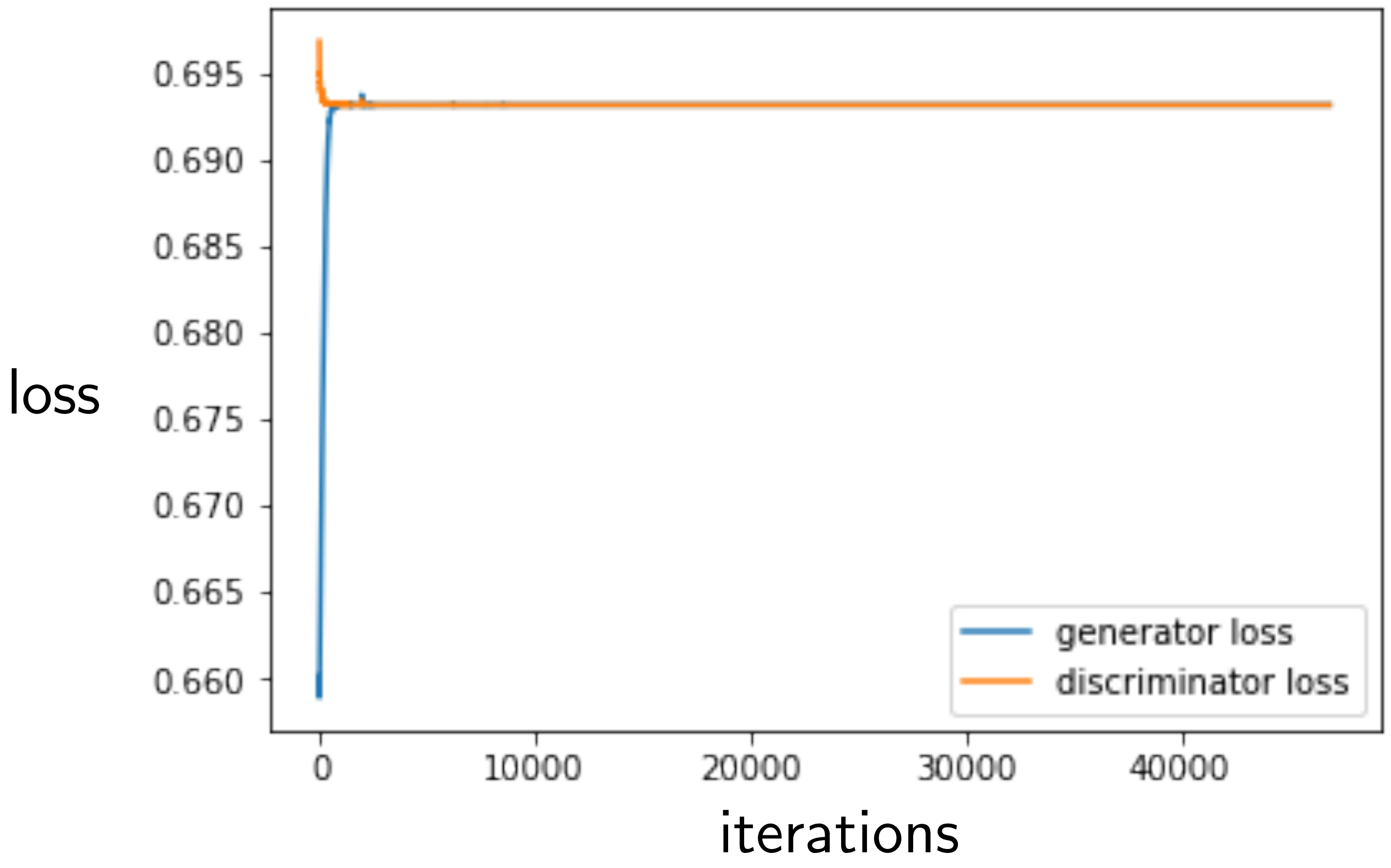
Generated images



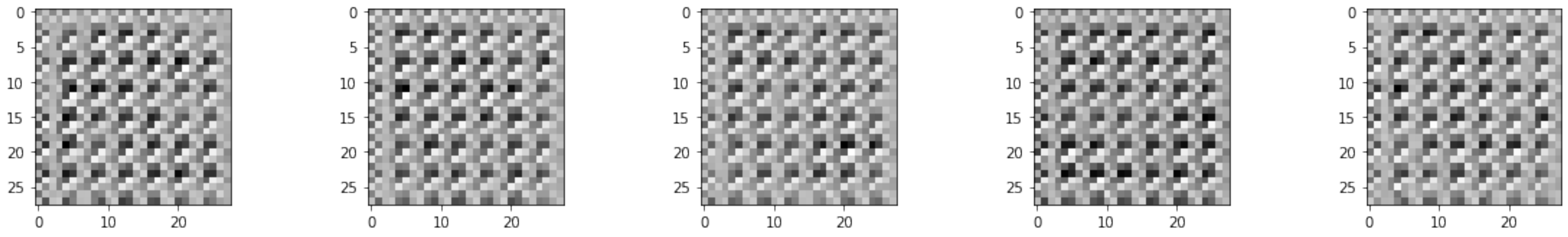
Full Code Examples (2)

MNIST Convolutional GAN that fails

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L17_gans/code/gan-cnn-mnist-converge-but-fail.ipynb



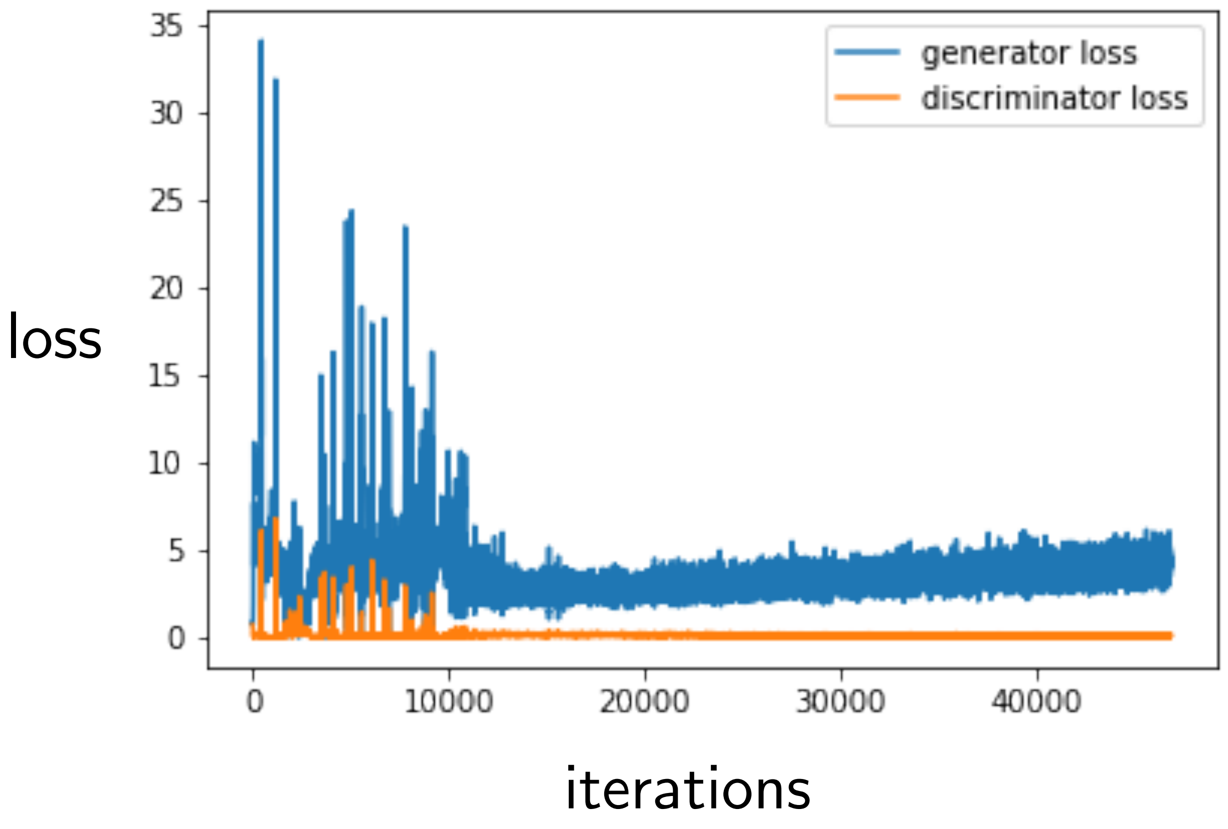
Generated images



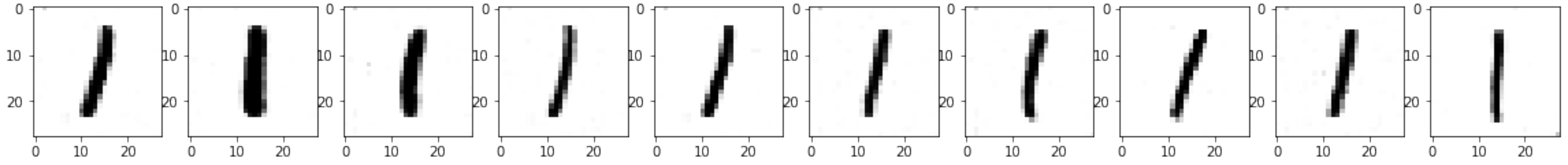
Full Code Examples (3)

MNIST Convolutional GAN with mode collapse

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L17_gans/code/gan-halfcnn-mnist-mode-collapse.ipynb



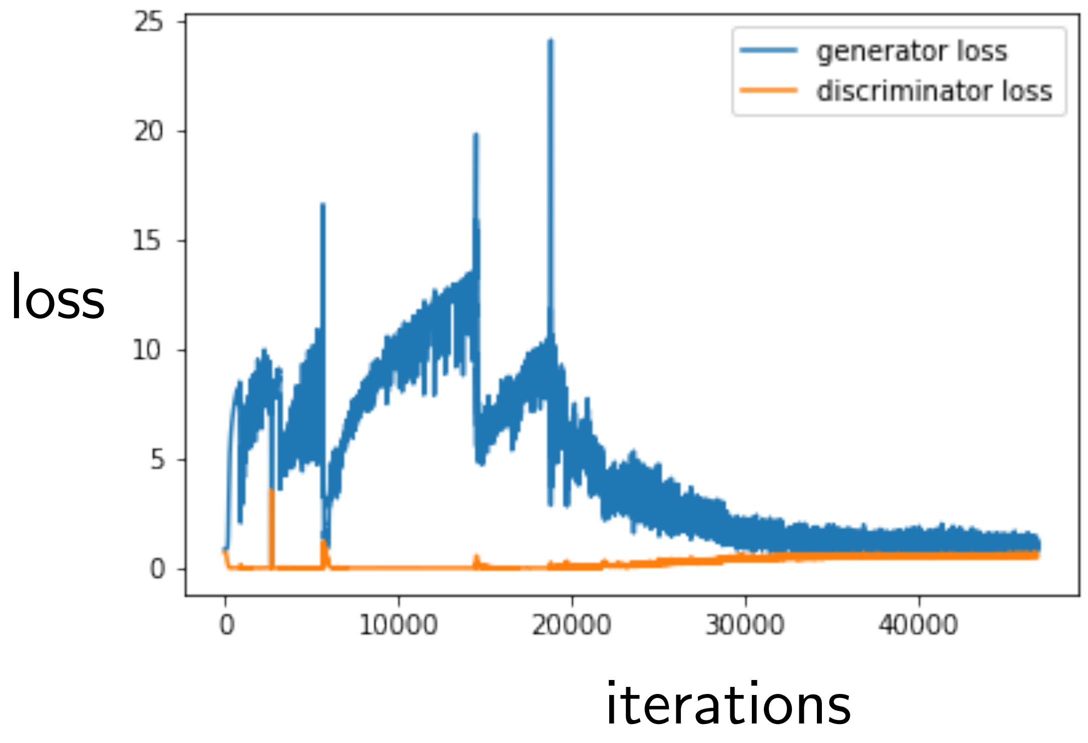
Generated images



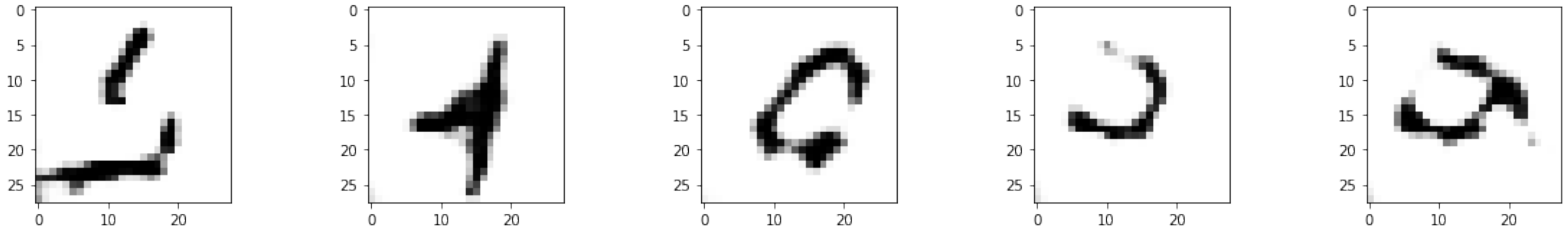
Full Code Examples (4)

MNIST Convolutional GAN -- not great

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L17_gans/code/gan-cnn-mnist-not-great.ipynb



Generated images

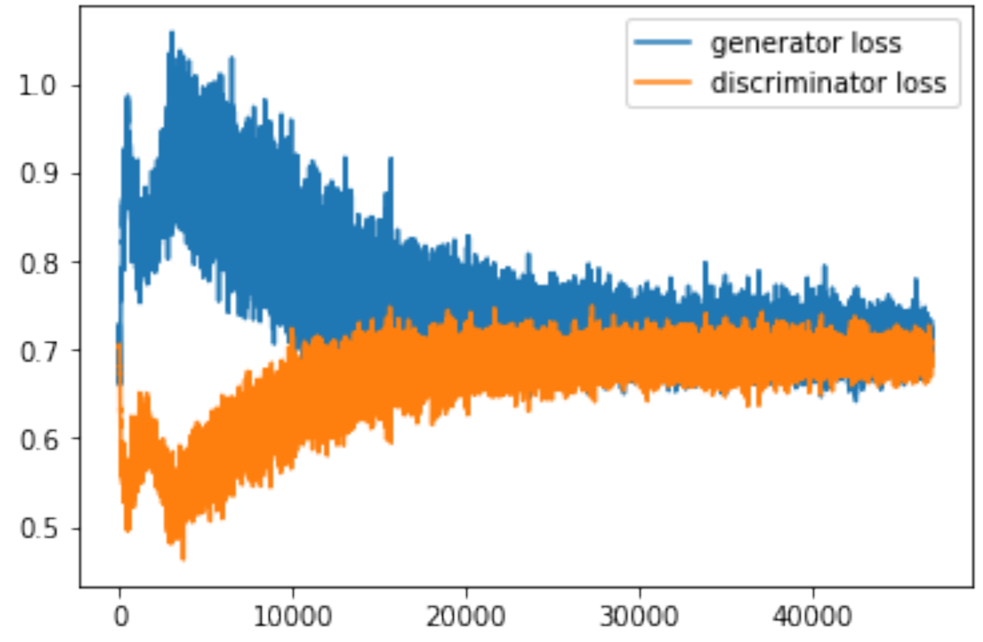


Full Code Examples (5)

MNIST Convolutional GAN -- relatively good

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L17_gans/code/gan-cnn-mnist.ipynb

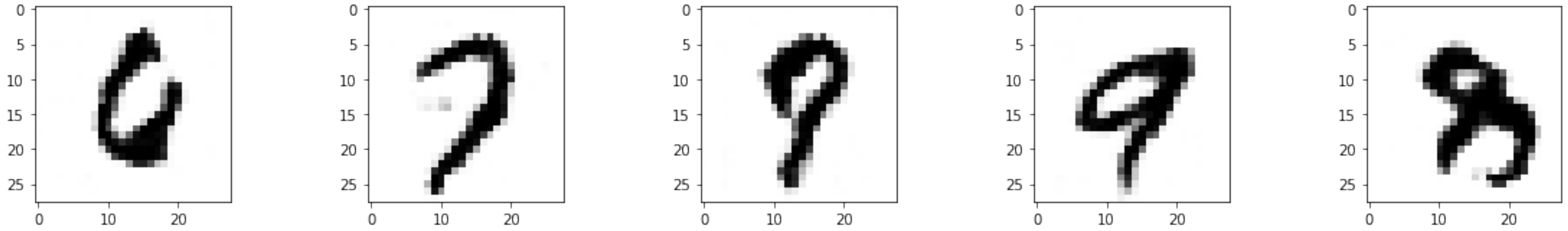
loss



iterations

Compared to the previous slide, this has BatchNorm

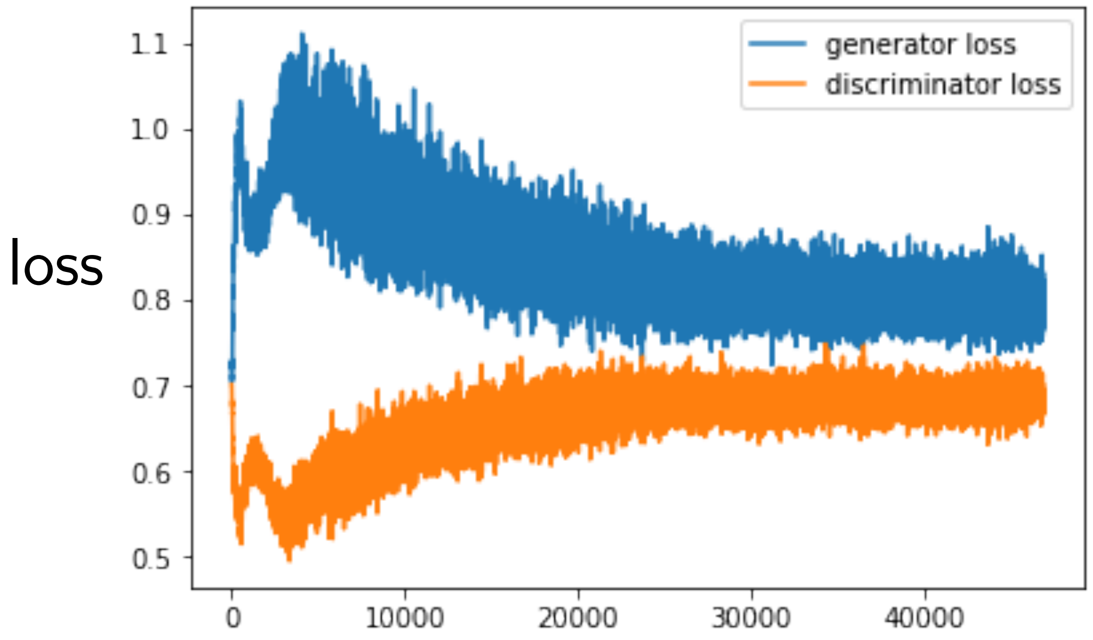
Generated images



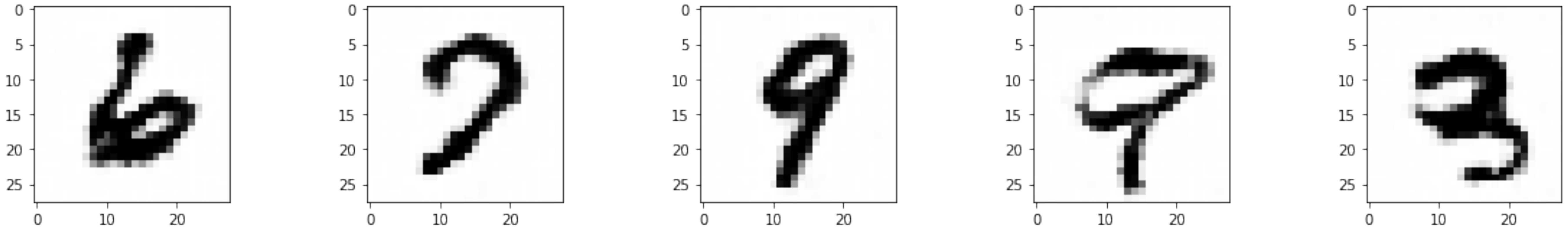
Full Code Examples (6)

Like previous slide but with label smoothing: Replace real images (1's) by 0.9 based on idea in

- Salimans, Tim, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training GANs." In *Advances in Neural Information Processing Systems*, pp. 2234-2242. 2016.

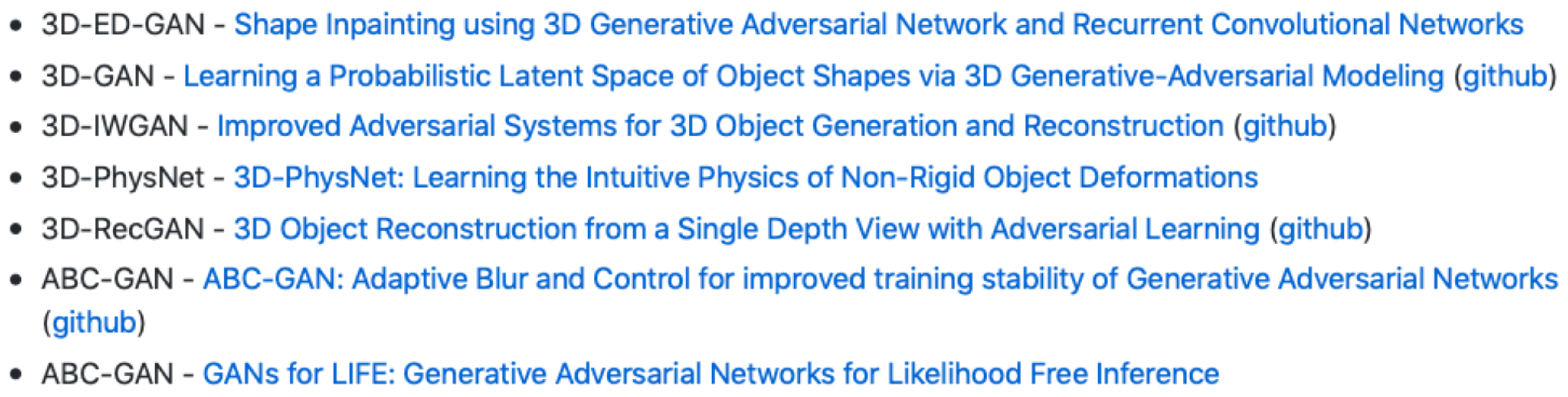


https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L17_gans/code/gan-cnn-mnist-label-smoothing.ipynb



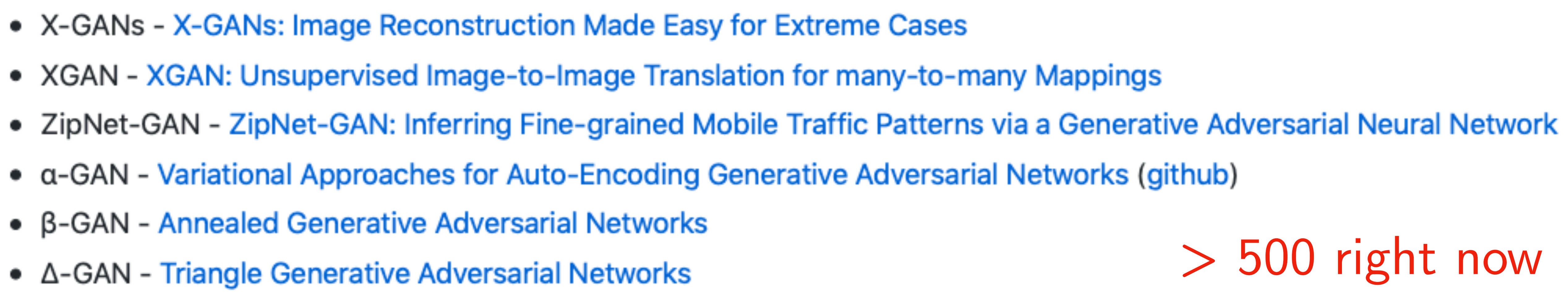
There are many more flavors of GANs that we have time to cover :)

A tentative list of named GANs https://github.com/hindupuravinash/the-gan-zoo



- 3D-ED-GAN - Shape Inpainting using 3D Generative Adversarial Network and Recurrent Convolutional Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling (github)
- 3D-IWGAN - Improved Adversarial Systems for 3D Object Generation and Reconstruction (github)
- 3D-PhysNet - 3D-PhysNet: Learning the Intuitive Physics of Non-Rigid Object Deformations
- 3D-RecGAN - 3D Object Reconstruction from a Single Depth View with Adversarial Learning (github)
- ABC-GAN - ABC-GAN: Adaptive Blur and Control for improved training stability of Generative Adversarial Networks (github)
- ABC-GAN - GANs for LIFE: Generative Adversarial Networks for Likelihood Free Inference

. . .

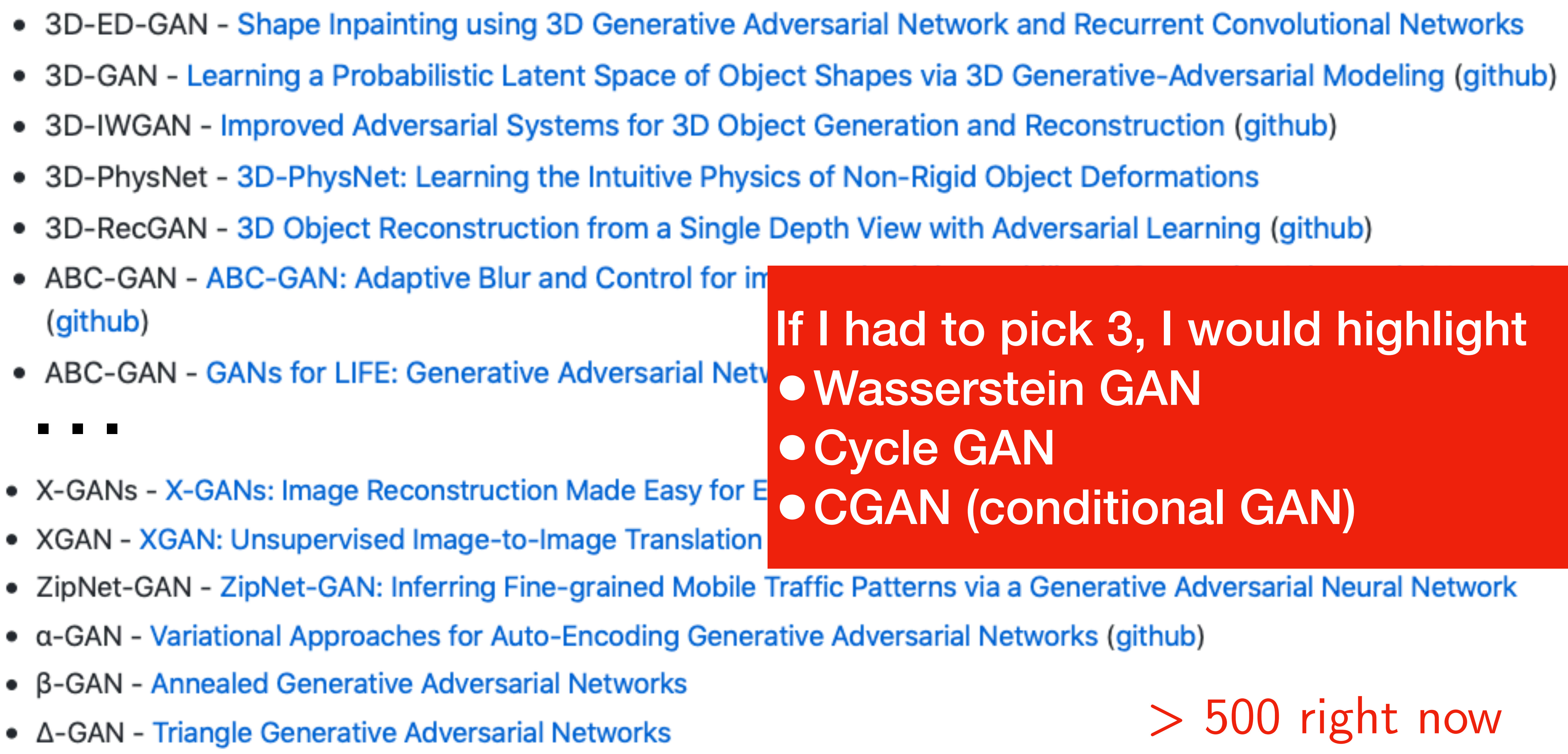


- X-GANs - X-GANs: Image Reconstruction Made Easy for Extreme Cases
- XGAN - XGAN: Unsupervised Image-to-Image Translation for many-to-many Mappings
- ZipNet-GAN - ZipNet-GAN: Inferring Fine-grained Mobile Traffic Patterns via a Generative Adversarial Neural Network
- α-GAN - Variational Approaches for Auto-Encoding Generative Adversarial Networks (github)
- β-GAN - Annealed Generative Adversarial Networks
- Δ-GAN - Triangle Generative Adversarial Networks

> 500 right now

There are many more flavors of GANs that we have time to cover :)

A tentative list of named GANs https://github.com/hindupuravinash/the-gan-zoo



- 3D-ED-GAN - Shape Inpainting using 3D Generative Adversarial Network and Recurrent Convolutional Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling (github)
- 3D-IWGAN - Improved Adversarial Systems for 3D Object Generation and Reconstruction (github)
- 3D-PhysNet - 3D-PhysNet: Learning the Intuitive Physics of Non-Rigid Object Deformations
- 3D-RecGAN - 3D Object Reconstruction from a Single Depth View with Adversarial Learning (github)
- ABC-GAN - ABC-GAN: Adaptive Blur and Control for im
(github)
- ABC-GAN - GANs for LIFE: Generative Adversarial Netw

...

- X-GANs - X-GANs: Image Reconstruction Made Easy for E
- XGAN - XGAN: Unsupervised Image-to-Image Translation
- ZipNet-GAN - ZipNet-GAN: Inferring Fine-grained Mobile Traffic Patterns via a Generative Adversarial Neural Network
- α-GAN - Variational Approaches for Auto-Encoding Generative Adversarial Networks (github)
- β-GAN - Annealed Generative Adversarial Networks
- Δ-GAN - Triangle Generative Adversarial Networks

If I had to pick 3, I would highlight
- Wasserstein GAN
- Cycle GAN
- CGAN (conditional GAN)

> 500 right now

Are GANs supervised or unsupervised?

Further Reading Suggestions

- Original GAN paper
  Goodfellow, Ian, 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*, pp. 2672-2680. 2014.

- Salimans, Tim, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training GANs." In *Advances in Neural Information Processing Systems*, pp. 2234-2242. 2016.

- Open Questions about Generative Adversarial Networks by Augustus Odena, https://distill.pub/2019/gan-open-problems/

- How to Train a GAN? Tips and Tricks to Make GANs Work by Soumith Chintala (PyTorch Dev.) https://github.com/soumith/ganhacks