Virtual Summer School on Deep Learning

July 5^{th} - 9^{th}, 2021

4^{th} International Summer School on Deep Learning - Virtual Edition

#VSSonDL
#ISSonDL

Introduction to Generative Adversarial Networks

Sebastian Raschka

University of Wisconsin-Madison

Organizers

GDAŃSK UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND INFORMATICS
BIOMEDICAL ENGINEERING DEPARTMENT
aiBAY
Digital Innovation Hub dih4.ai
intel
SiMa ai

Sponsors

Diamond Sponsor
amazon

Platinum Sponsor
JIT
Jit Team™ the human factor of IT

GDAŃSK UNIVERSITY OF TECHNOLOGY
NVIDIA
FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND INFORMATICS
servodata
BIOMEDICAL ENGINEERING DEPARTMENT
Digital Innovation Hub dih4.ai
aiBAY
intel
SiMa ai

It all began in 2014

Generative Adversarial Networks = GAN/GANs

arXiv.org > stat > arXiv:1406.2661

Search...

Help | Advanced

Statistics > Machine Learning

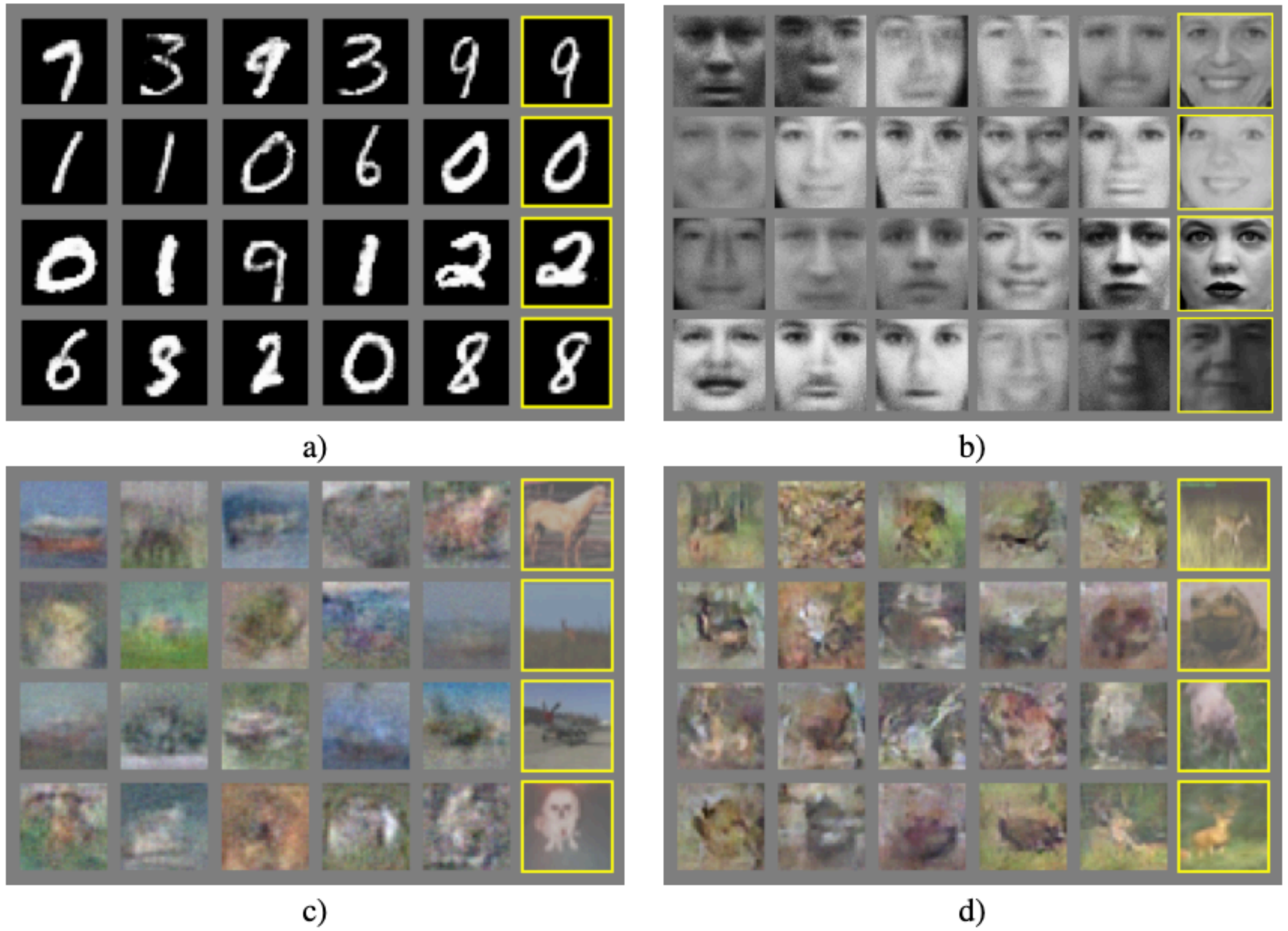
[Submitted on 10 Jun 2014]

Generative Adversarial Networks

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio

We propose a new framework for estimating generative models via an adversarial process, in which we 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. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to 1/2 everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

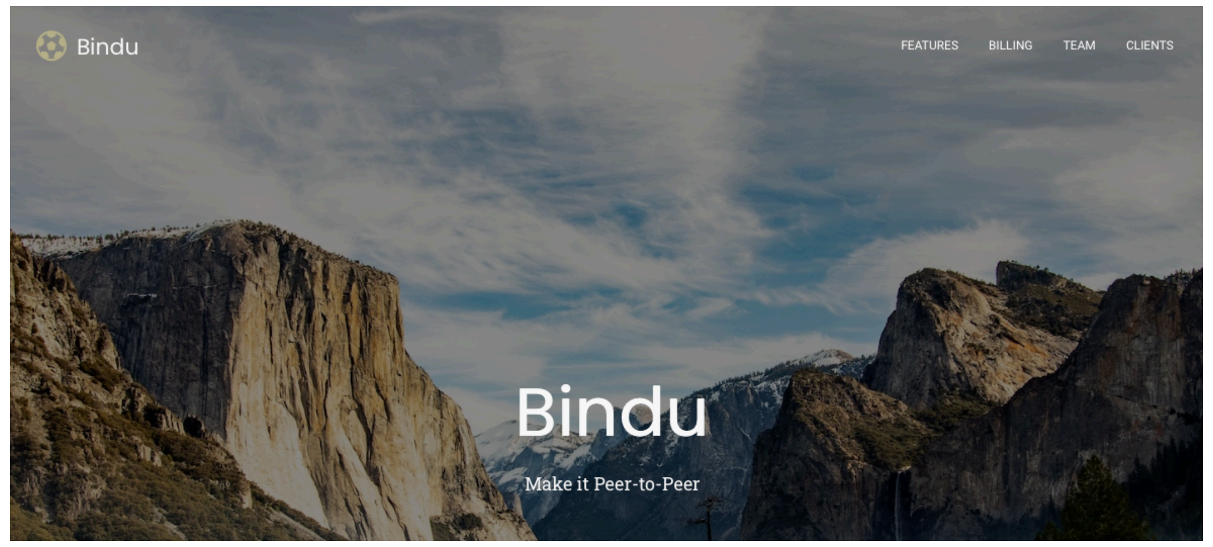
https://arxiv.org/abs/1406.2661



Face generation has come a long way

Image source: https://blog.eduonix.com/artificial-intelligence/grand-finale-applications-gans/

https://thisstartupdoesnotexist.com



https://thiscatdoesnotexist.com

https://thisponydoesnotexist.net

https://thispersondoesnotexist.com

Today's Topics

1. The Main Idea Behind GANs

2. The GAN Objective

3. Modifying the GAN Loss Function for Practical Use

4. A Simple GAN Generating Handwritten Digits in PyTorch

5. Tips and Tricks to Make GANs Work

6. A DCGAN for Generating Face Images in PyTorch

7. GAN Resources

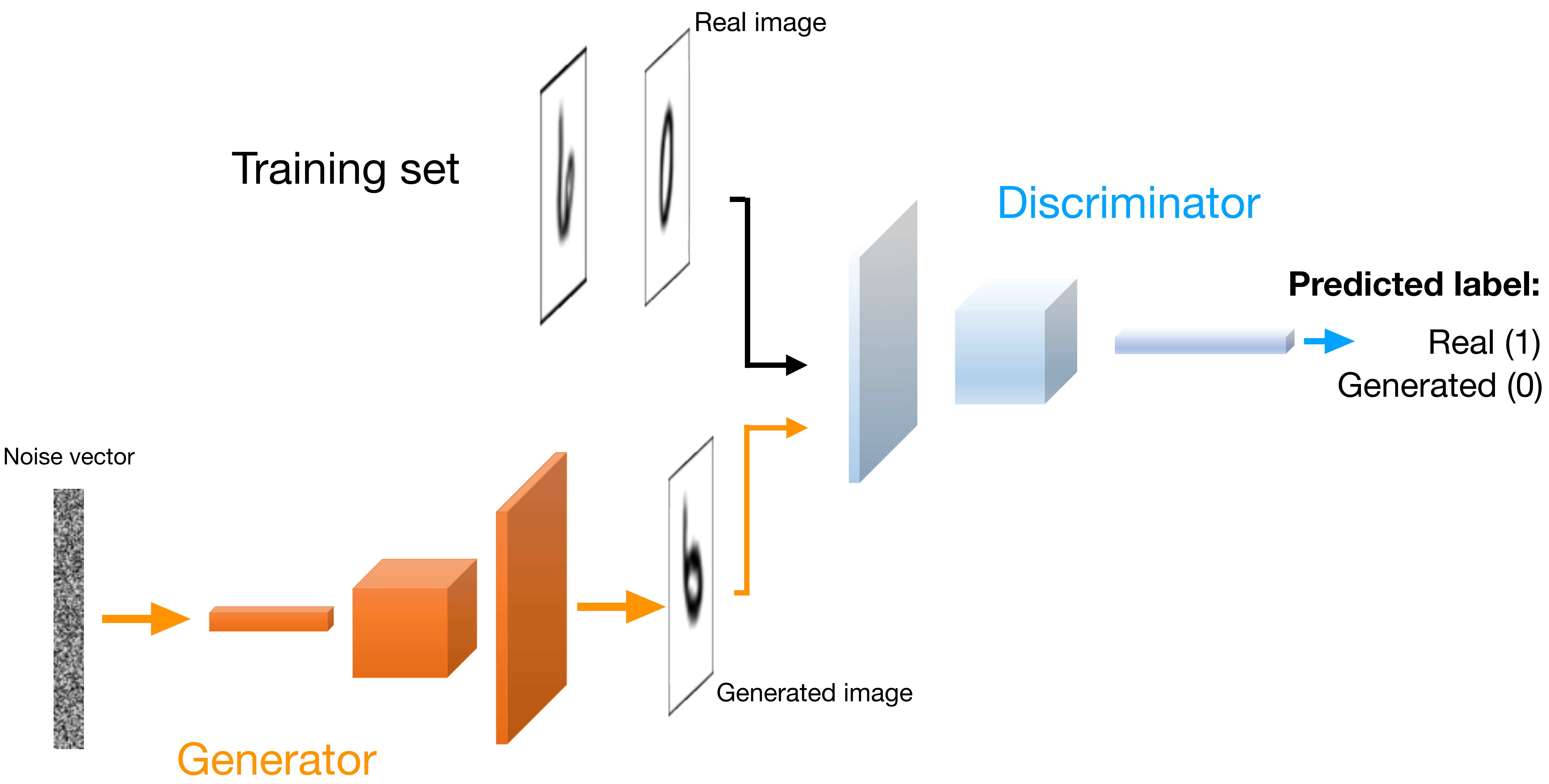
Letting two neural networks compete with each other

1. **The Main Idea Behind GANs**
2. The GAN Objective
3. Modifying the GAN Loss Function for Practical Use
4. A Simple GAN Generating Handwritten Digits in PyTorch
5. Tips and Tricks to Make GANs Work
6. A DCGAN for Generating Face Images in PyTorch
7. GAN Resources

Generative Adversarial Networks (GAN)

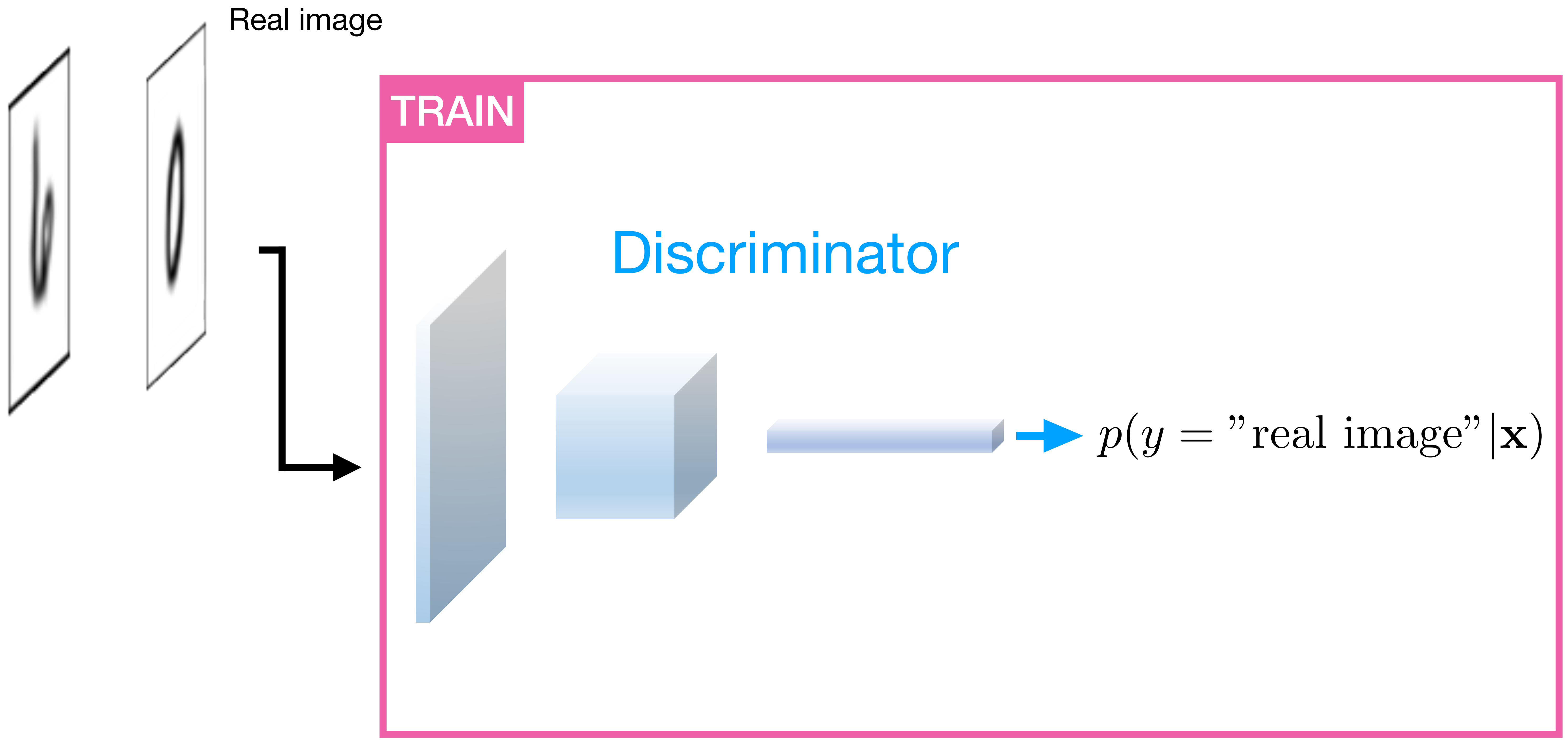
- The original purpose is to generate new data
- Classically for generating new images, but applicable to wide range of domains
- Learns the training set distribution and can generate new images that have never been seen before

Deep Convolutional GAN (DCGAN or just GAN)



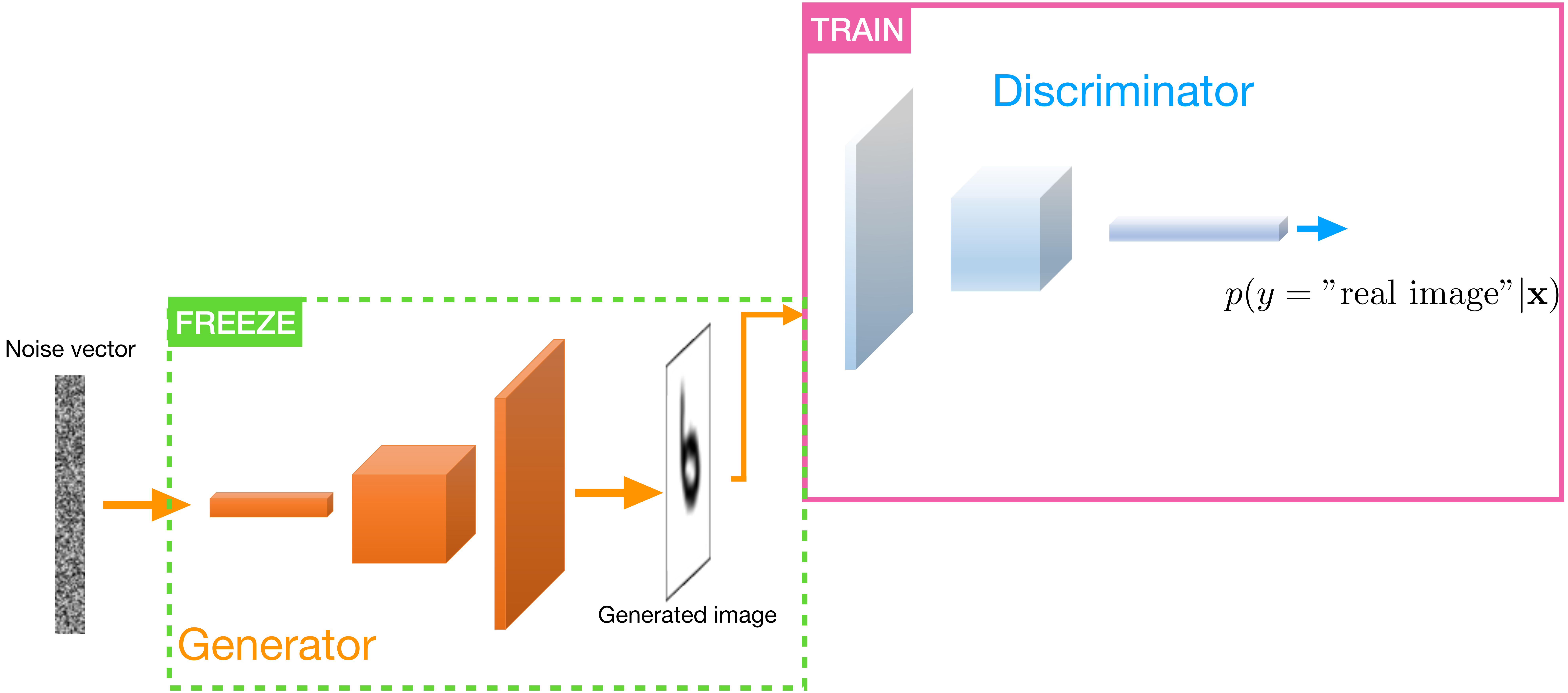
Step 1.1: Train Discriminator

Training set



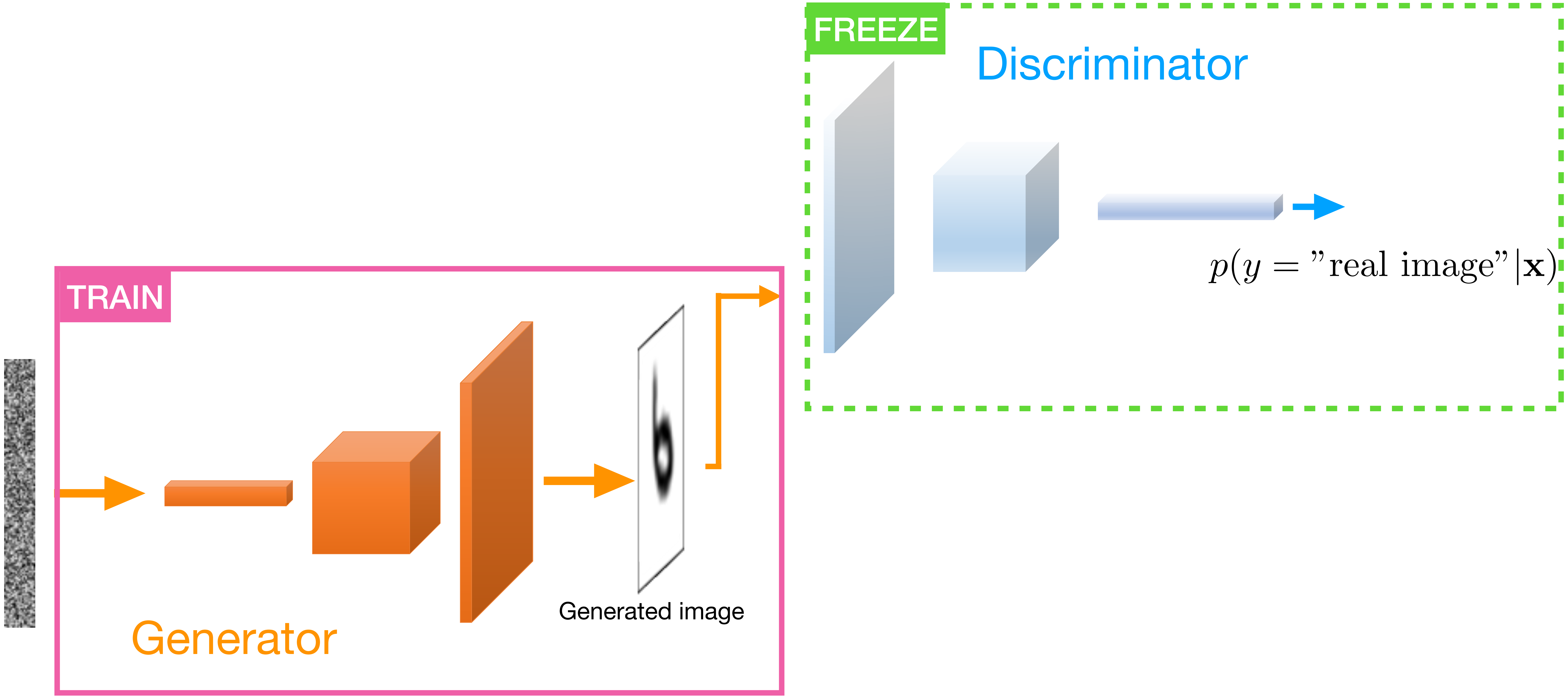
Train to predict that real image is real

Step 1.2: Train Discriminator



Train to predict that fake image is fake

Step 2: Train Generator



Train to predict that fake image is real

Adversarial Game

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

Generator: learns to generate better images to fool the discriminator

How do the loss functions look like?

1. The Main Idea Behind GANs

2. The GAN Objective

3. Modifying the GAN Loss Function for Practical Use

4. A Simple GAN Generating Handwritten Digits in PyTorch

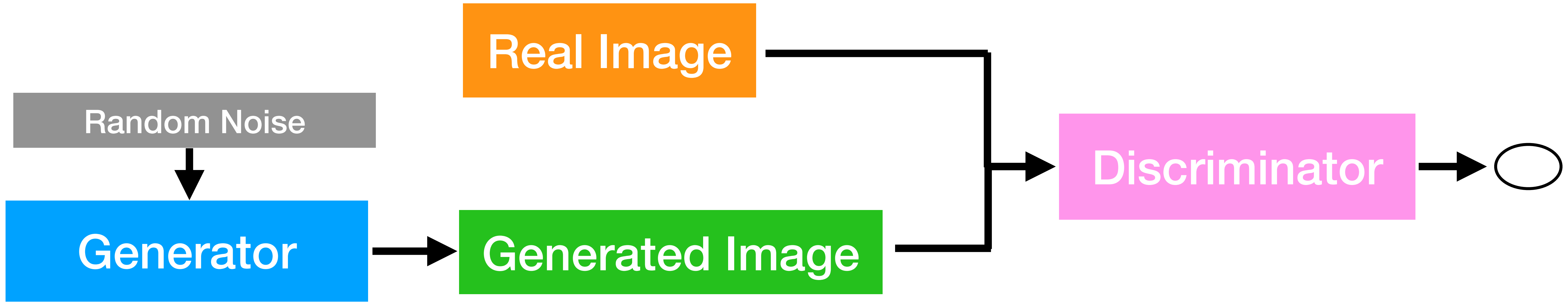
5. Tips and Tricks to Make GANs Work

6. A DCGAN for Generating Face Images in PyTorch

7. GAN Resources

GAN Objective

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



Turning the minmax optimization setting into a minimization problem for SGD

Simplified GAN Loss

(1) Minimize discriminator loss

Real image

Generated image

$$\min_D \; -\left(E_{x \sim p_x}[\log D(x)] + E_{z \sim p_z}[\log(1 - D(G(z)))]\right)$$

using "original" labels

Real images: class label 1
Generated images: class label 0

Simplified GAN Loss

(1) Minimize discriminator loss

Real image

Generated image

$$\min_D \; -\left(E_{x\sim p_x}[\log \underbrace{D(x)}_{\text{value range: } [0,1]}] + E_{z\sim p_z}[\log(1 - D(G(z)))]\right)$$

want it to predict *real* -> 1
(correct prediction)

$$\underbrace{\log D(x)}_{\text{value range: } [-\infty, 0]}$$

$$\underbrace{-\left(E_{x\sim p_x}[\log D(x)]\right)}_{\text{loss value range: } [\infty, 0]}$$

Simplified GAN Loss

(1) Minimize discriminator loss

Real image

Generated image

$$\min_D \; -\left(E_{x\sim p_x}[\log D(x)] + E_{z\sim p_z}[\log(1 - \underbrace{D(G(z))}_{\text{value range: } [0,1]})]\right)$$

predict *generated* -> 0
(correct prediction)

$$\underbrace{1 - D(G(z))}_{\text{value range: } [1,0]}$$

$$\underbrace{-\left(E_{z\sim p_z}[\log(1 - D(G(z)))]\right)}_{\text{loss value range: } [0,\infty]}$$

Simplified GAN Loss

(2) Minimize generator loss (to fool discriminator)

Generated image

$$\min_{G} \; -\left(E_{z \sim p_z}[\log D(\boxed{G(z)})]\right)$$

Want to fool discriminator to make a wrong prediction

Generated images: predict class label 1

Simplified GAN Loss

(2) Minimize generator loss (to fool discriminator)

Generated image

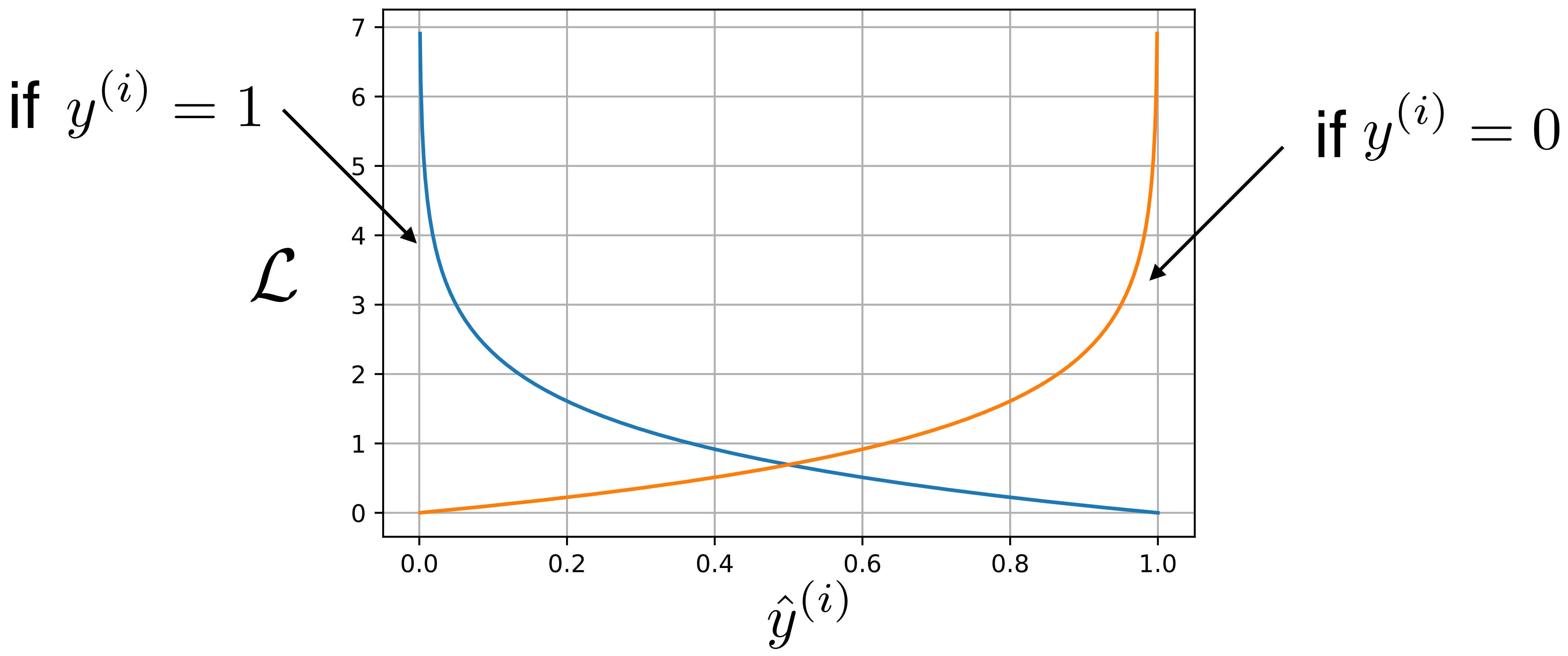
$$\min_G \; -\left(E_{z \sim p_z}[\log \underbrace{D(\boxed{G(z)})}_{\text{value range: } [0,1]}]\right)$$

want it to ~~predict *generated* -> 0~~
predict real -> 1

$$\underbrace{\log D(G(z))}_{\text{value range: } [-\infty, 0]}$$

$$\underbrace{-\left(E_{x \sim p_x}[\log D(G(z))]\right)}_{\text{loss value range: } [\infty, 0]}$$

Negative Log-Likelihood / Binary Cross Entropy Loss



$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \left[-\left(y^{(i)} \log \left(\hat{y}^{(i)} \right) + \left(1 - y^{(i)} \right) \log \left(1 - \hat{y}^{(i)} \right) \right) \right]$$

Negative Log-Likelihood / Binary Cross Entropy Loss

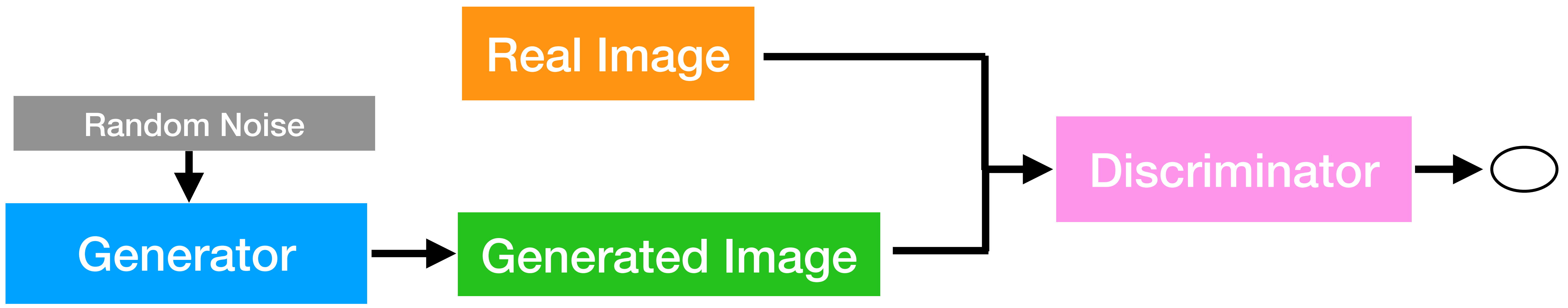
$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\left[-\left(y^{(i)}\log\left(\hat{y}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-\hat{y}^{(i)}\right)\right)\right]$$

Discriminator: real images $y = [1 \;\; 1 \;\; \ldots \;\; 1]$

want $\hat{y} = [1 \;\; 1 \;\; \ldots \;\; 1]$

generated images $y = [0 \;\; 0 \;\; \ldots \;\; 0]$

want $\hat{y} = [0 \;\; 0 \;\; \ldots \;\; 0]$



Negative Log-Likelihood / Binary Cross Entropy Loss

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\left[-\left(y^{(i)}\log\left(\hat{y}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-\hat{y}^{(i)}\right)\right)\right]$$

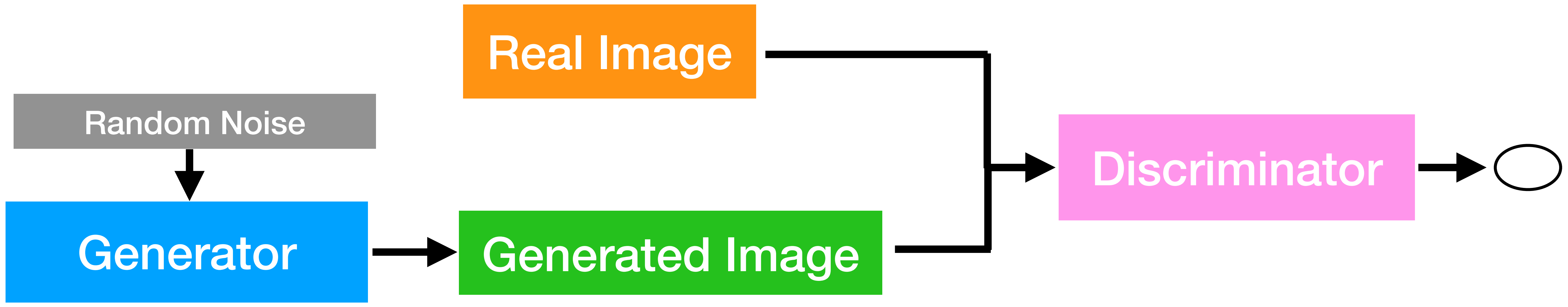
Generator:

generated images $y = [0 \;\; 0 \;\; \dots \;\; 0]$

label flip

$y = [1 \;\; 1 \;\; \dots \;\; 1]$

want to fool the discriminator $\hat{y} = [1 \;\; 1 \;\; \dots \;\; 1]$



Implementing our first GAN

1. The Main Idea Behind GANs

2. The GAN Objective

3. Modifying the GAN Loss Function for Practical Use

4. **A Simple GAN Generating Handwritten Digits in PyTorch**

5. Tips and Tricks to Make GANs Work

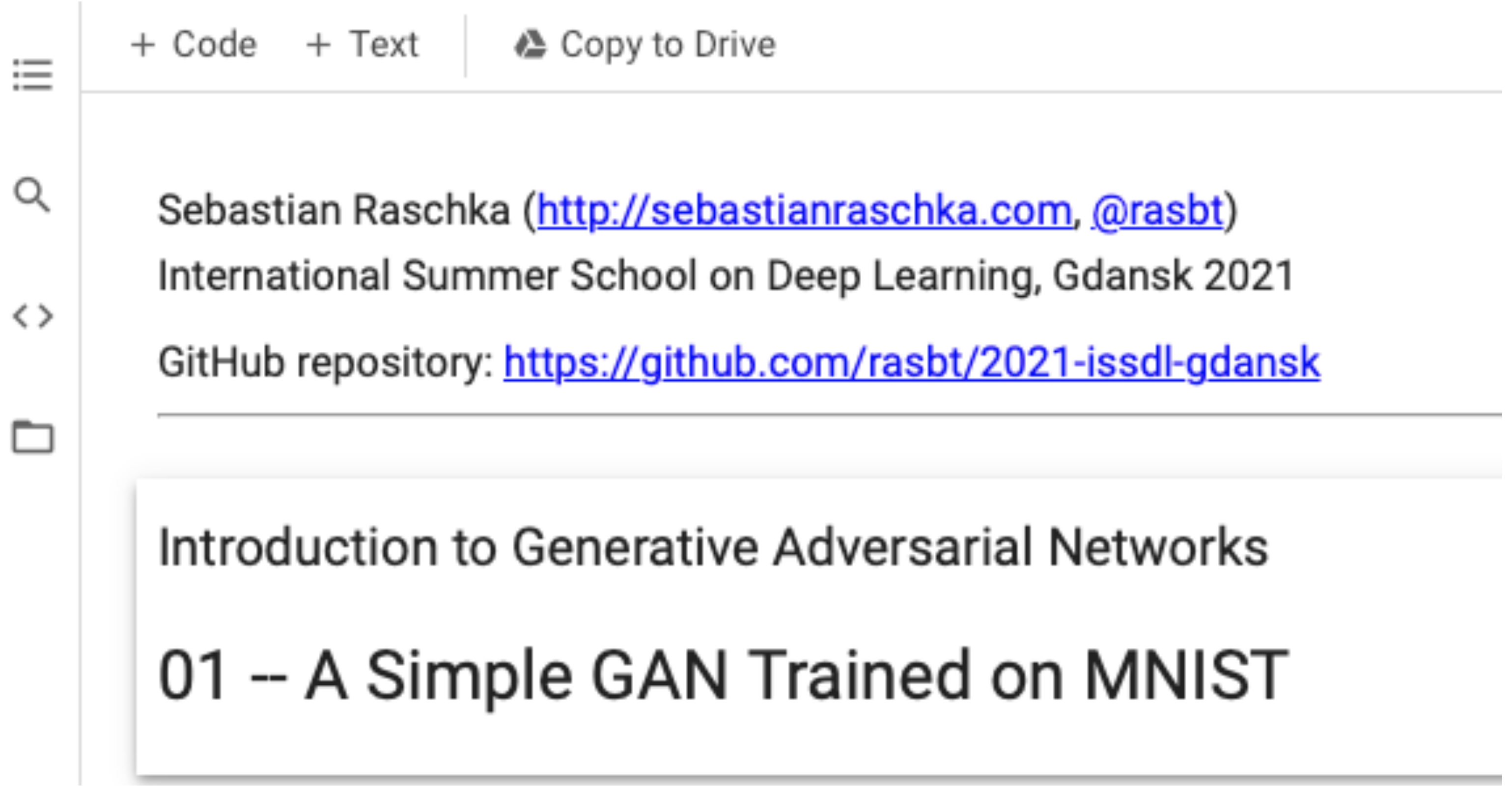
6. A DCGAN for Generating Face Images in PyTorch

7. GAN Resources

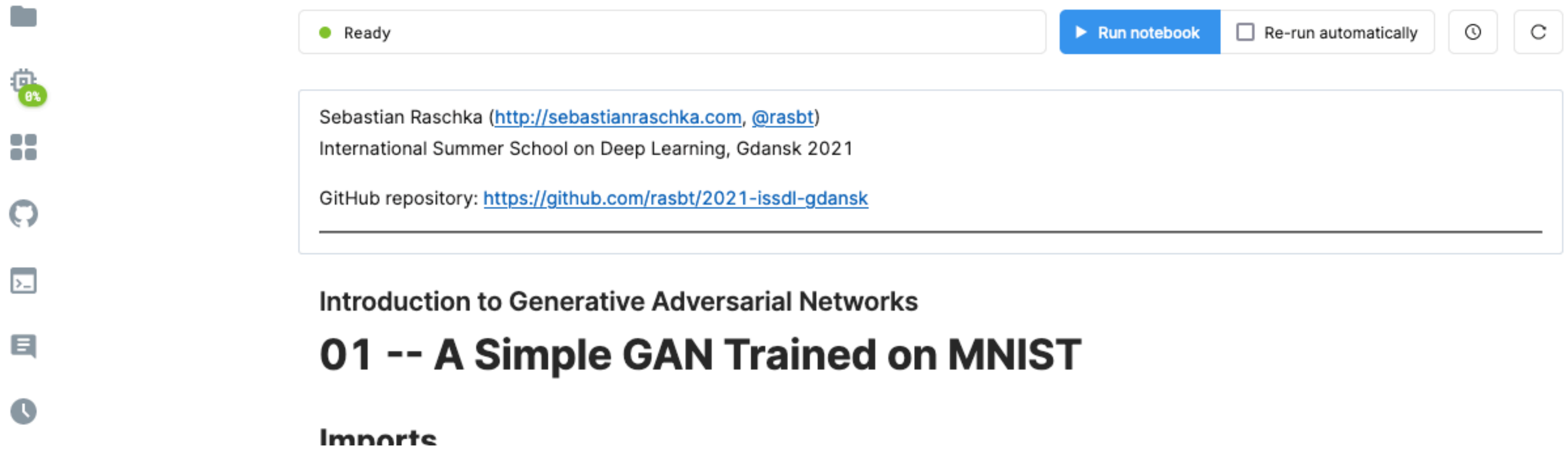
```python
self.generator = nn.Sequential(
    nn.Linear(latent_dim, 128),
    nn.LeakyReLU(inplace=True),
    nn.Dropout(p=0.5),
    nn.Linear(128, image_height*image_width*color_channels),
    #################################################
    ## Exercise 1: ##
    #################
    ## Which activation function,
    ##   tanh or logistic sigmoid?
    ##   Uncomment the correct one below.

    # a)
    # nn.Sigmoid
    #
    # b)
    # nn.Tanh

    #################################################
)
```

```python
model.train()
for batch_idx, (features, _) in enumerate(train_loader):

    batch_size = features.size(0)

    # real images
    real_images = features.to(device)

    ##################################################
    ## Exercise 2: ##
    #################
    ## Which labels for the real images?
    ## Uncomment the correct code below

    # a)
    # real_labels = torch.zeros(batch_size, device=device)
    #
    # b)
    # real_labels = torch.ones(batch_size, device=device)
    ##################################################
```

```python
# generated (fake) images
noise = torch.randn(batch_size, latent_dim, 1, 1, device=device)
fake_images = model.generator_forward(noise)

##################################################
## Exercise 3: ##
#################
## Which labels for the generated images?
## Uncomment the correct code below

# a)
# fake_labels = torch.zeros(batch_size, device=device)
#
# b)
# fake_labels = torch.ones(batch_size, device=device)
##################################################

flipped_fake_labels = real_labels
```

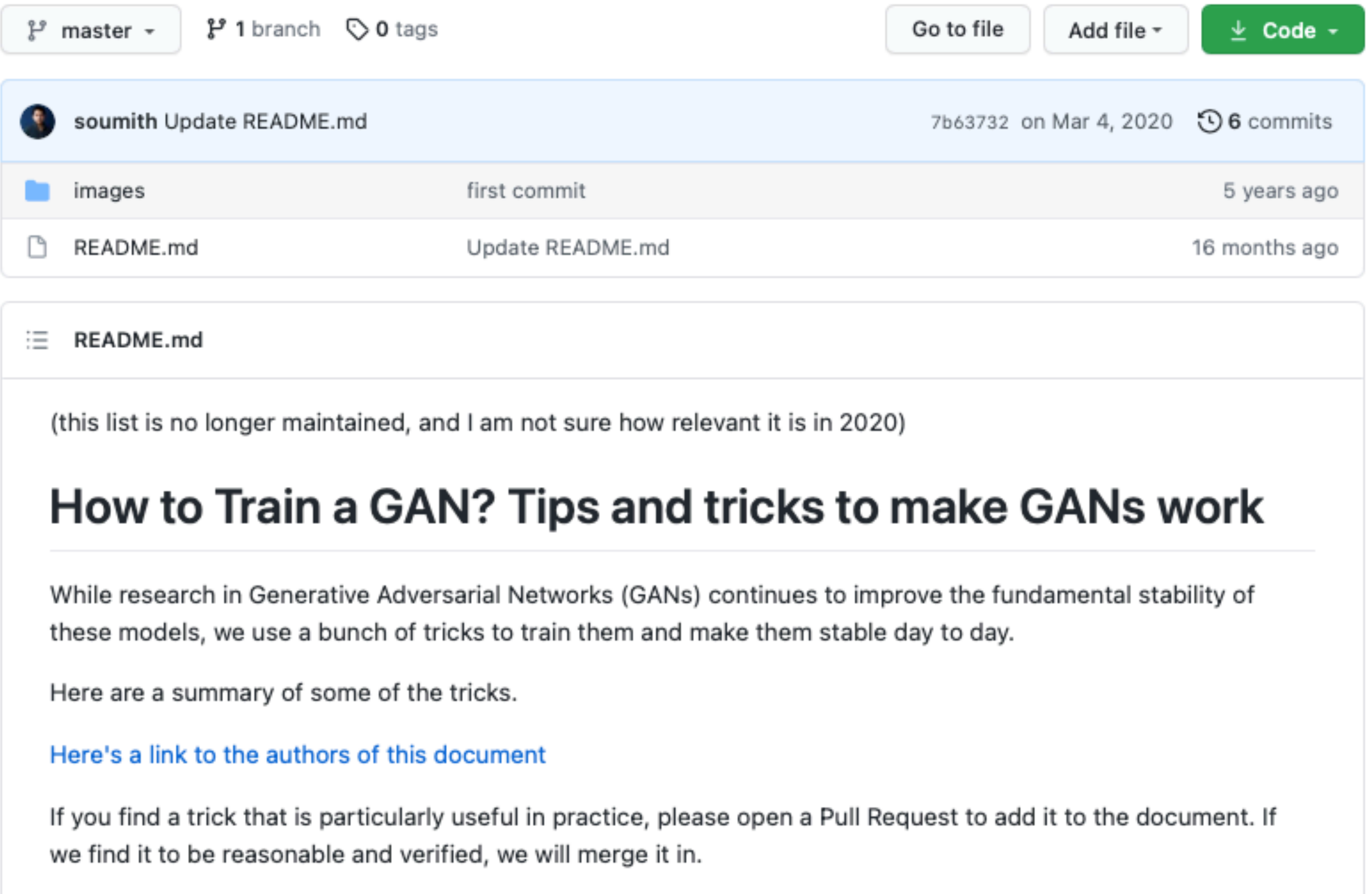


https://colab.research.google.com/github/rasbt/2021-issdl-gdansk/blob/main/01_gan-mnist-exercise.ipynb



https://deepnote.com/project/2021-issdl-gdansk-qUQLtJxgQeKj0NrLtxDeow/%2F01_gan-mnist-exercise.ipynb

Looking at some of the best practices for GAN training


master · 1 branch · 0 tags · Go to file · Add file · Code

soumith Update README.md · 7b63732 on Mar 4, 2020 · 6 commits

images	first commit	5 years ago
README.md	Update README.md	16 months ago

README.md

(this list is no longer maintained, and I am not sure how relevant it is in 2020)

How to Train a GAN? Tips and tricks to make GANs work

While research in Generative Adversarial Networks (GANs) continues to improve the fundamental stability of these models, we use a bunch of tricks to train them and make them stable day to day.

Here are a summary of some of the tricks.

Here's a link to the authors of this document

If you find a trick that is particularly useful in practice, please open a Pull Request to add it to the document. If we find it to be reasonable and verified, we will merge it in.

https://github.com/soumith/ganhacks

1. Normalize the inputs

- normalize the images between -1 and 1
- Tanh as the last layer of the generator output

1. Normalize the inputs

- normalize the images between -1 and 1
- Tanh as the last layer of the generator output

```
self.generator = nn.Sequential(
    nn.Linear(latent_dim, 128),
    nn.LeakyReLU(inplace=True),
    nn.Dropout(p=0.5),
    nn.Linear(128, image_height*image_width*color_channels),
    nn.Tanh()
)
```

```
custom_transforms = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5,), (0.5,))
])
```

2: A modified loss function

In GAN papers, the loss function to optimize G is `min (log 1-D)`, but in practice folks practically use `max log D`

- because the first formulation has vanishing gradients early on
- Goodfellow et. al (2014)

In practice, works well:

- Flip labels when training generator: real = fake, fake = real

2: A modified loss function

In GAN papers, the loss function to optimize G is `min (log 1-D)`, but in practice folks practically use `max log D`

we used `min -log D`, which is the same

- because the first formulation has vanishing gradients early on
- Goodfellow et. al (2014)

In practice, works well:

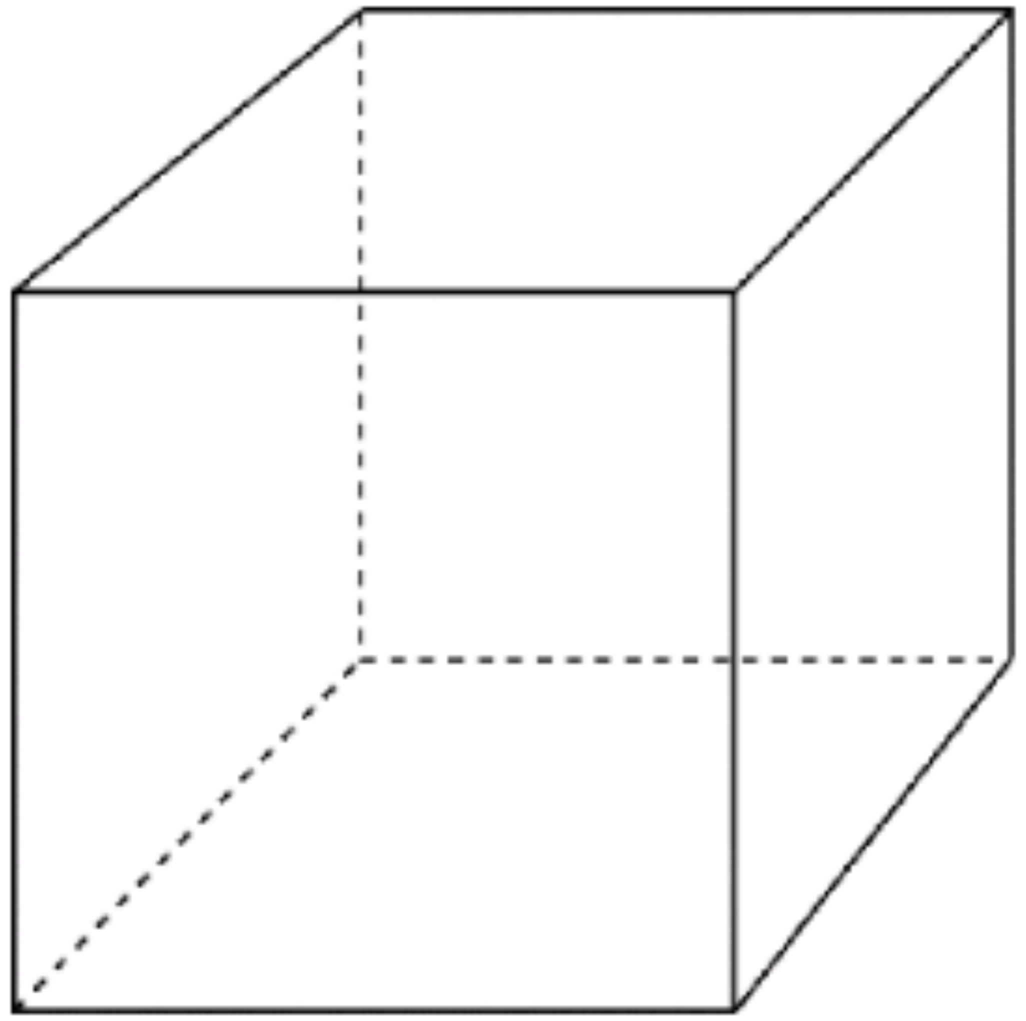
- Flip labels when training generator: real = fake, fake = real

```
# real images
real_images = features.to(device)
real_labels = torch.ones(batch_size, device=device) # real label = 1

# generated (fake) images
noise = torch.randn(batch_size, latent_dim, 1, 1, device=device)  # format NCHW
fake_images = model.generator_forward(noise)
fake_labels = torch.zeros(batch_size, device=device) # fake label = 0
flipped_fake_labels = real_labels # here, fake label = 1
```

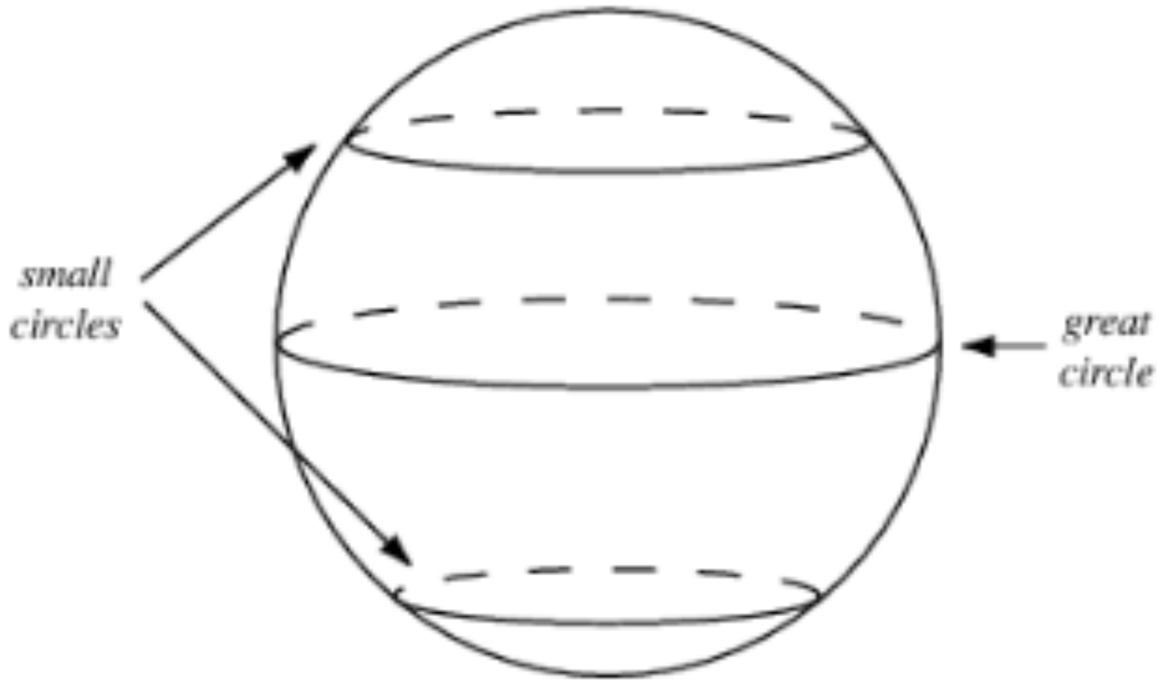
```
# --------------------------
# Train Generator
# --------------------------

optimizer_gen.zero_grad()

# get discriminator loss on fake images with flipped labels
discr_pred_fake = model.discriminator_forward(fake_images).view(-1)
gener_loss = loss_fn(discr_pred_fake, flipped_fake_labels)
gener_loss.backward()

optimizer_gen.step()
```

3: Use a spherical Z

- Dont sample from a Uniform distribution



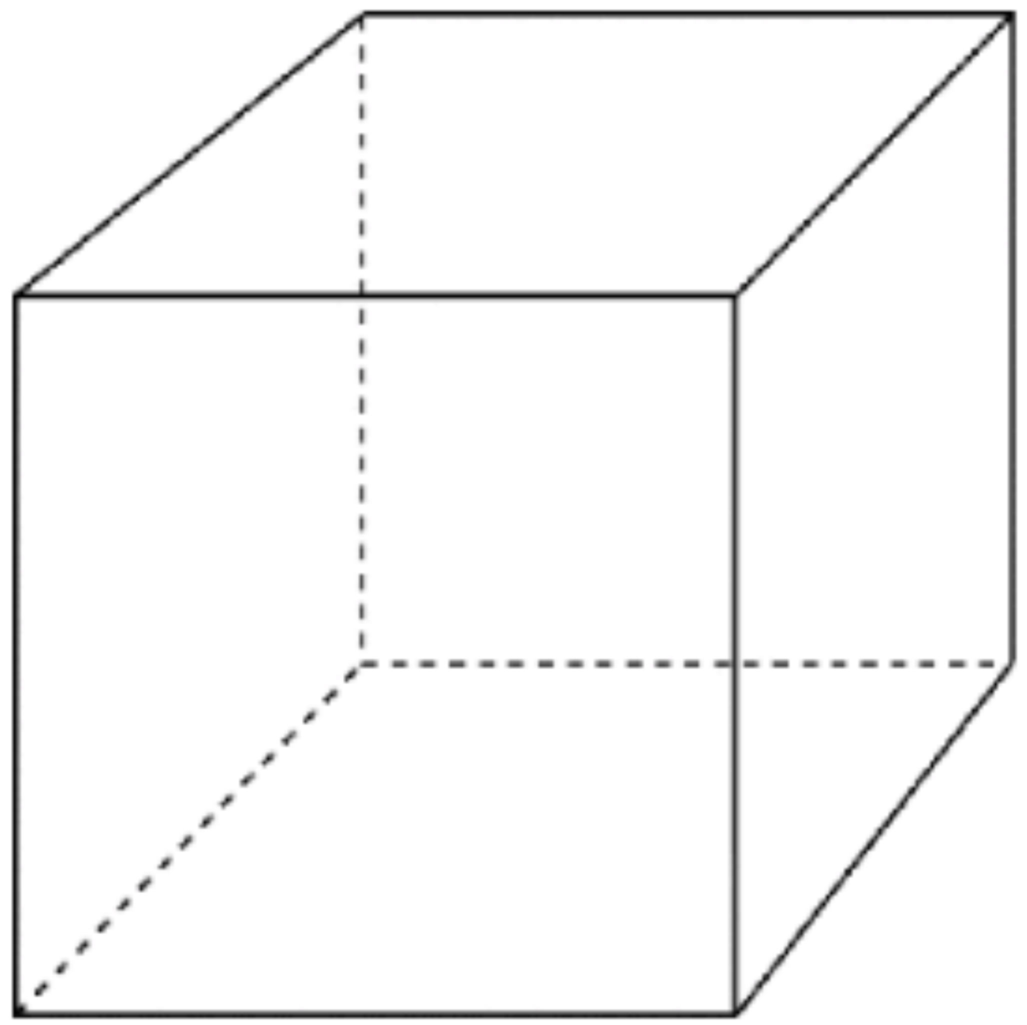
- Sample from a gaussian distribution



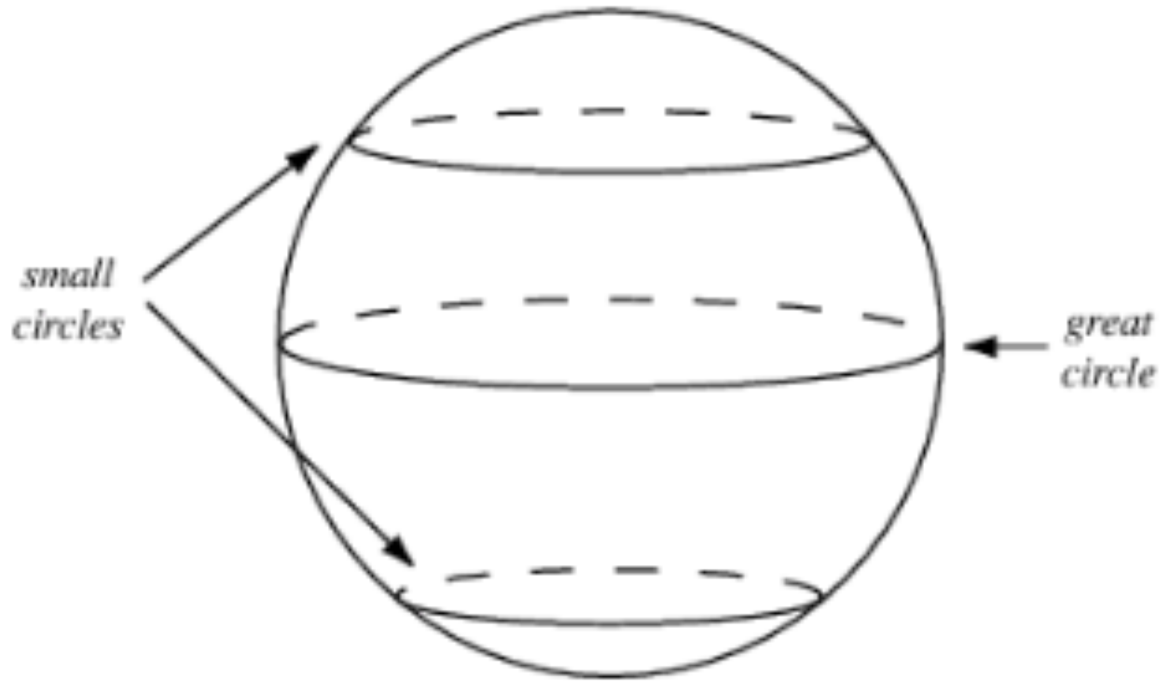
- When doing interpolations, do the interpolation via a great circle, rather than a straight line from point A to point B
- Tom White's Sampling Generative Networks ref code https://github.com/dribnet/plat has more details

3: Use a spherical Z

- Dont sample from a Uniform distribution



- Sample from a gaussian distribution

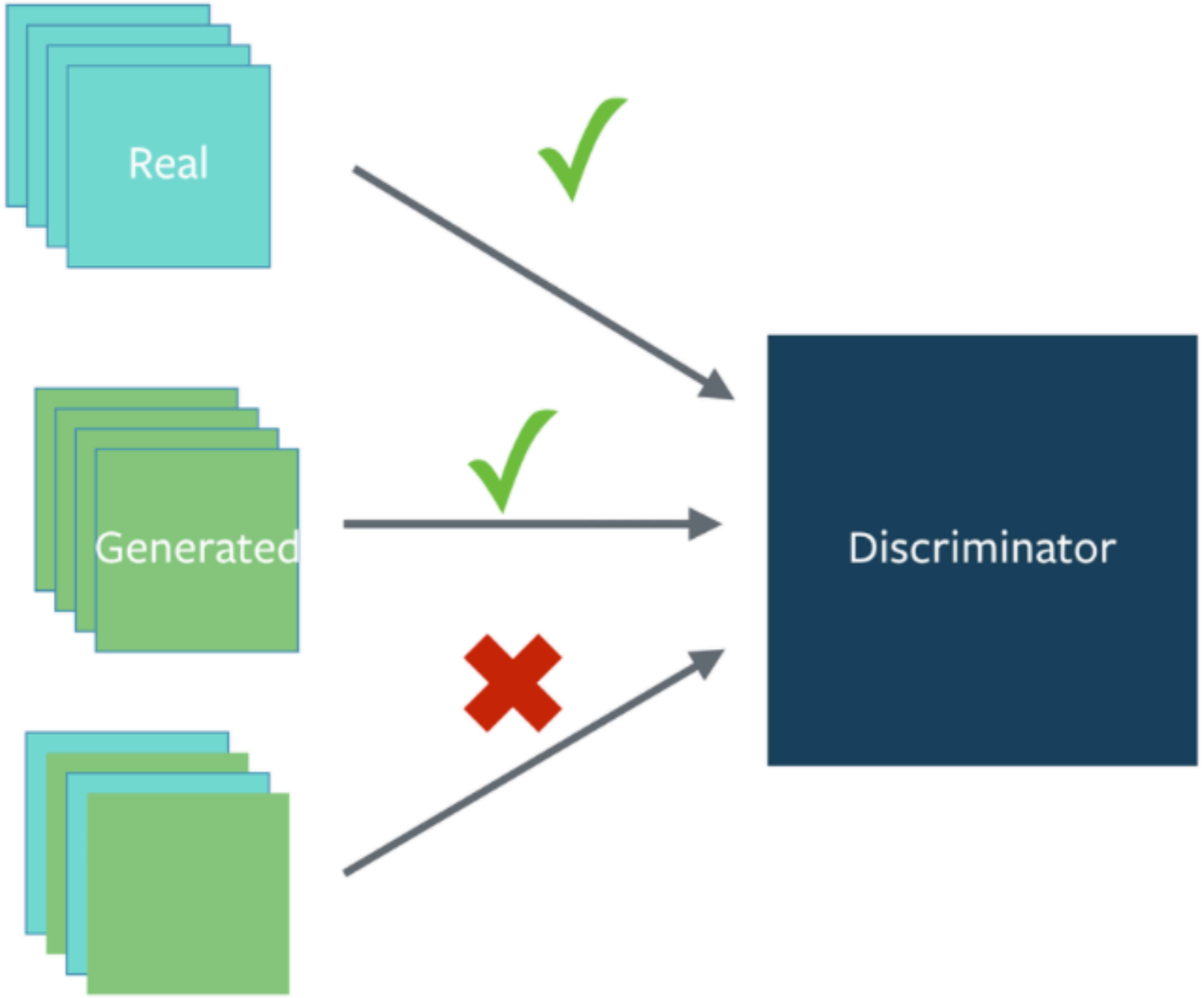



```
model.train()
for batch_idx, (features, _) in enumerate(train_loader):

    batch_size = features.size(0)

    # real images
    real_images = features.to(device)
    real_labels = torch.ones(batch_size, device=device) # real label = 1

    # generated (fake) images
    noise = torch.randn(batch_size, latent_dim, 1, 1, device=device)  # format NCHW
    fake_images = model.generator_forward(noise)
    fake_labels = torch.zeros(batch_size, device=device) # fake label = 0
    flipped_fake_labels = real_labels # here, fake label = 1
```

- When doing interpolations, do the interpolation via a great circle, rather than a straight line from point A to point B
- Tom White's Sampling Generative Networks ref code https://github.com/dribnet/plat has more details

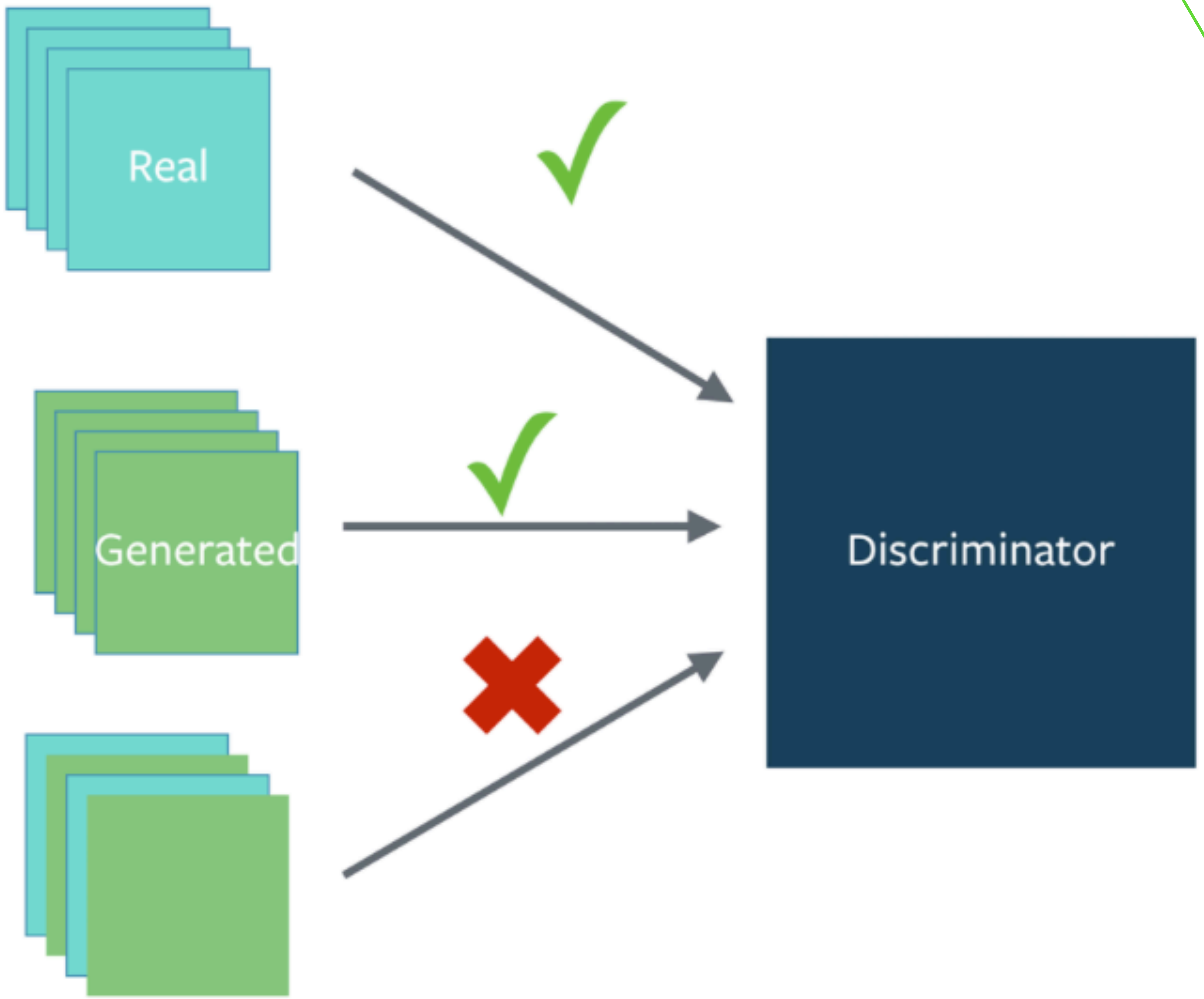
4: BatchNorm

- Construct different mini-batches for real and fake, i.e. each mini-batch needs to contain only all real images or all generated images.
- when batchnorm is not an option use instance normalization (for each sample, subtract mean and divide by standard deviation).



4: BatchNorm

- Construct different mini-batches for real and fake, i.e. each mini-batch needs to contain only all real images or all generated images.
- when batchnorm is not an option use instance normalization (for each sample, subtract mean and divide by standard deviation).




```python
# get discriminator loss on real images
discr_pred_real = model.discriminator_forward(real_images).view(-1) # Nx1 -> N
real_loss = loss_fn(discr_pred_real, real_labels)
# real_loss.backward()

# get discriminator loss on fake images
discr_pred_fake = model.discriminator_forward(fake_images.detach()).view(-1)
fake_loss = loss_fn(discr_pred_fake, fake_labels)
# fake_loss.backward()

# combined loss
discr_loss = 0.5*(real_loss + fake_loss)
discr_loss.backward()
```

BatchNorm in upcoming

02_dcgan-celeba.ipynb

```python
class DCGAN(torch.nn.Module):

    def __init__(self, latent_dim=100,
                 num_feat_maps_gen=64, num_feat_maps_dis=64,
                 color_channels=3):
        super().__init__()


        self.generator = nn.Sequential(
            nn.ConvTranspose2d(latent_dim, num_feat_maps_gen*8,
                               kernel_size=4, stride=1, padding=0,
                               bias=False),
            nn.BatchNorm2d(num_feat_maps_gen*8),
            nn.LeakyReLU(inplace=True),
```

...

Implementing a GAN with convolutional layers

Deep Convolutional GAN

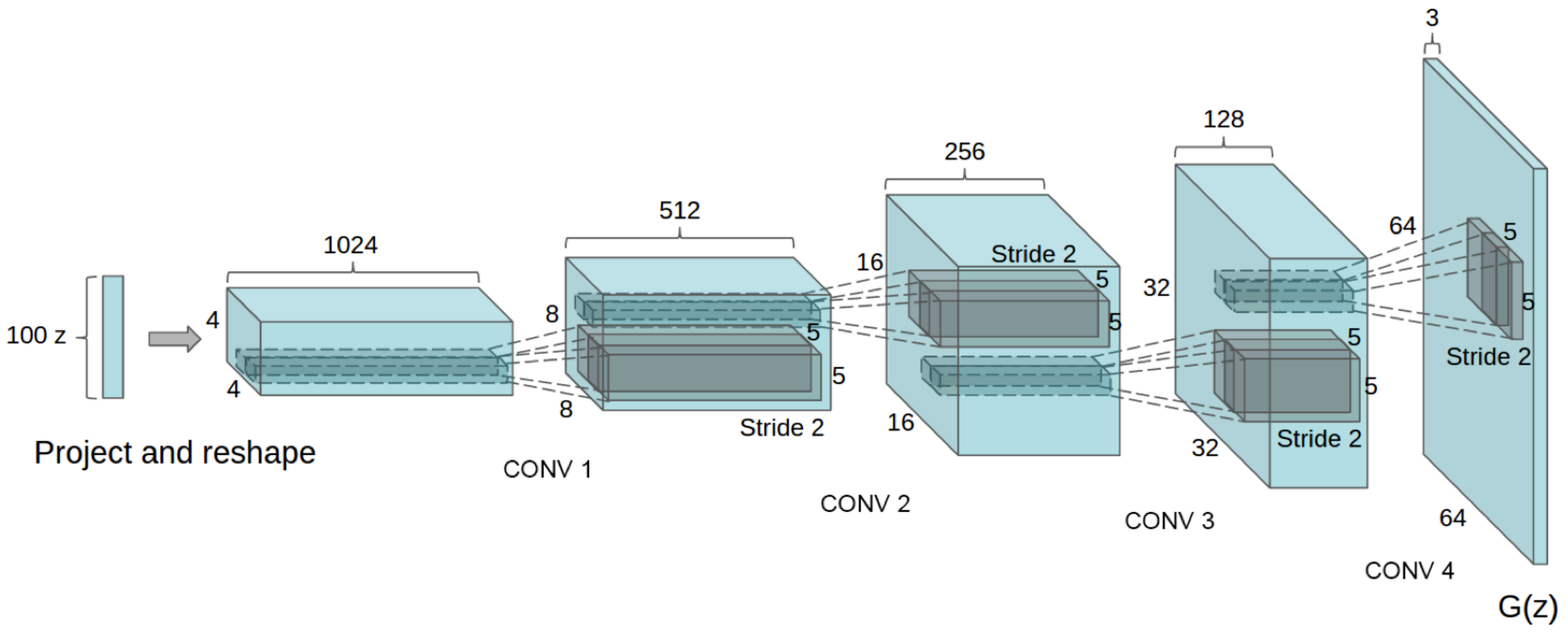
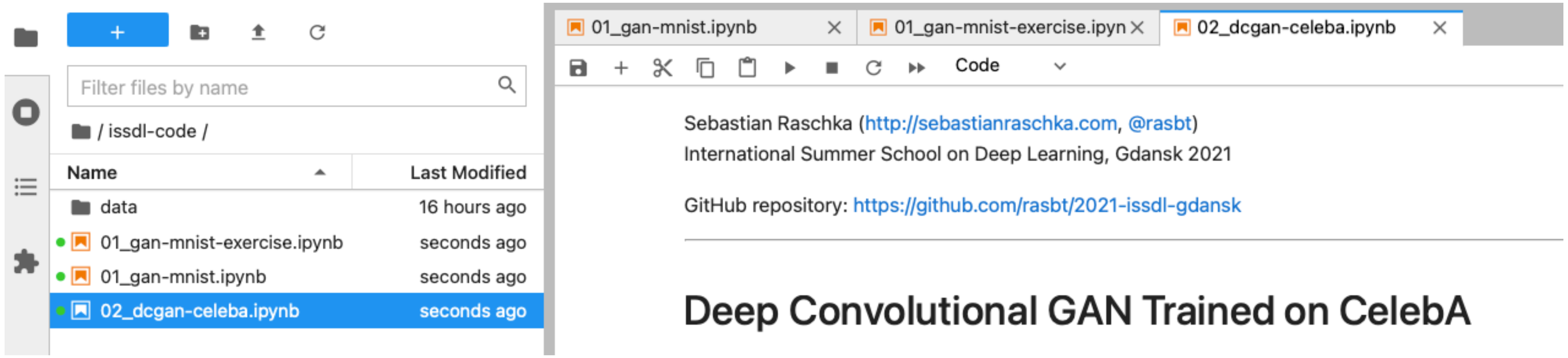
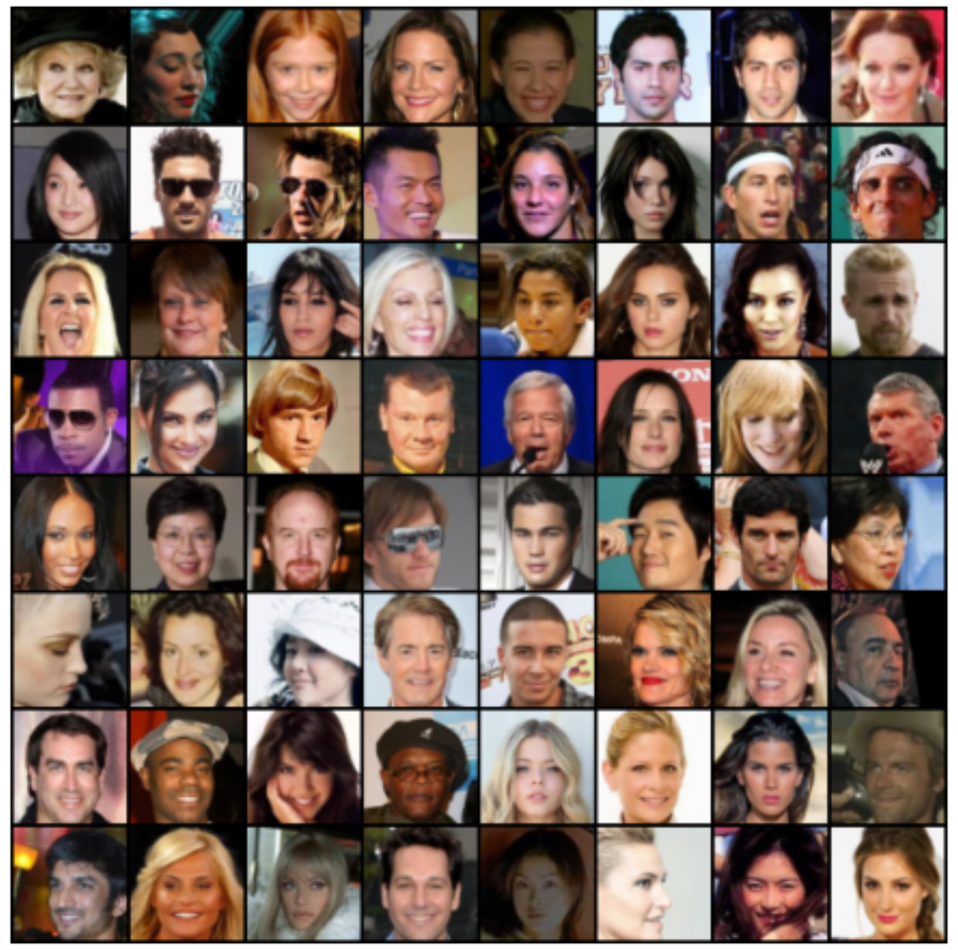


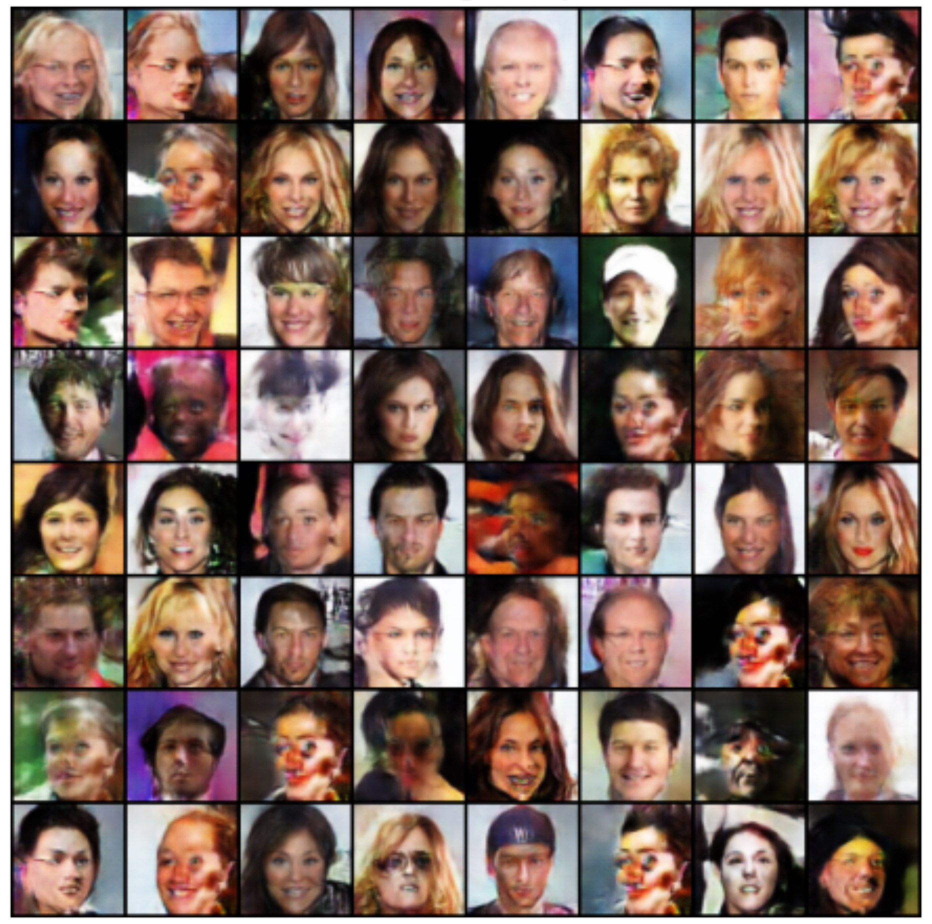
Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.



Code:
https://github.com/rasbt/2021-issdl-gdansk





If the CelebA download causes problems (e.g., because the daily download quota was exceeded), you can download the dataset from the original CelebA page: https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

Or download celeba.zip (1.7 Gb) and from https://drive.google.com/file/d/1m8-EBPgi5MRubrm6iQjafK2QMHDBMSfJ/view?usp=sharing
and unzip it in the notebook directory

A subselection of popular GANs and further reading resources

1. The Main Idea Behind GANs
2. The GAN Objective
3. Modifying the GAN Loss Function for Practical Use
4. A Simple GAN Generating Handwritten Digits in PyTorch
5. Tips and Tricks to Make GANs Work
6. A DCGAN for Generating Face Images in PyTorch
7. **GAN Resources**

GAN Resources

Wang, Z., She, Q. and Ward, T.E., 2021.
Generative adversarial networks in computer vision: A survey and taxonomy.
ACM Computing Surveys (CSUR), *54*(2), pp.1-38.
https://arxiv.org/abs/2001.06937

Gui, J., Sun, Z., Wen, Y., Tao, D. and Ye, J., 2020.
A review on generative adversarial networks: Algorithms, theory, and applications.
https://dl.acm.org/doi/pdf/10.1145/3439723

GAN Papers to Read in 2020
https://towardsdatascience.com/gan-papers-to-read-in-2020-2c708af5c0a4

A very much abbreviated timeline

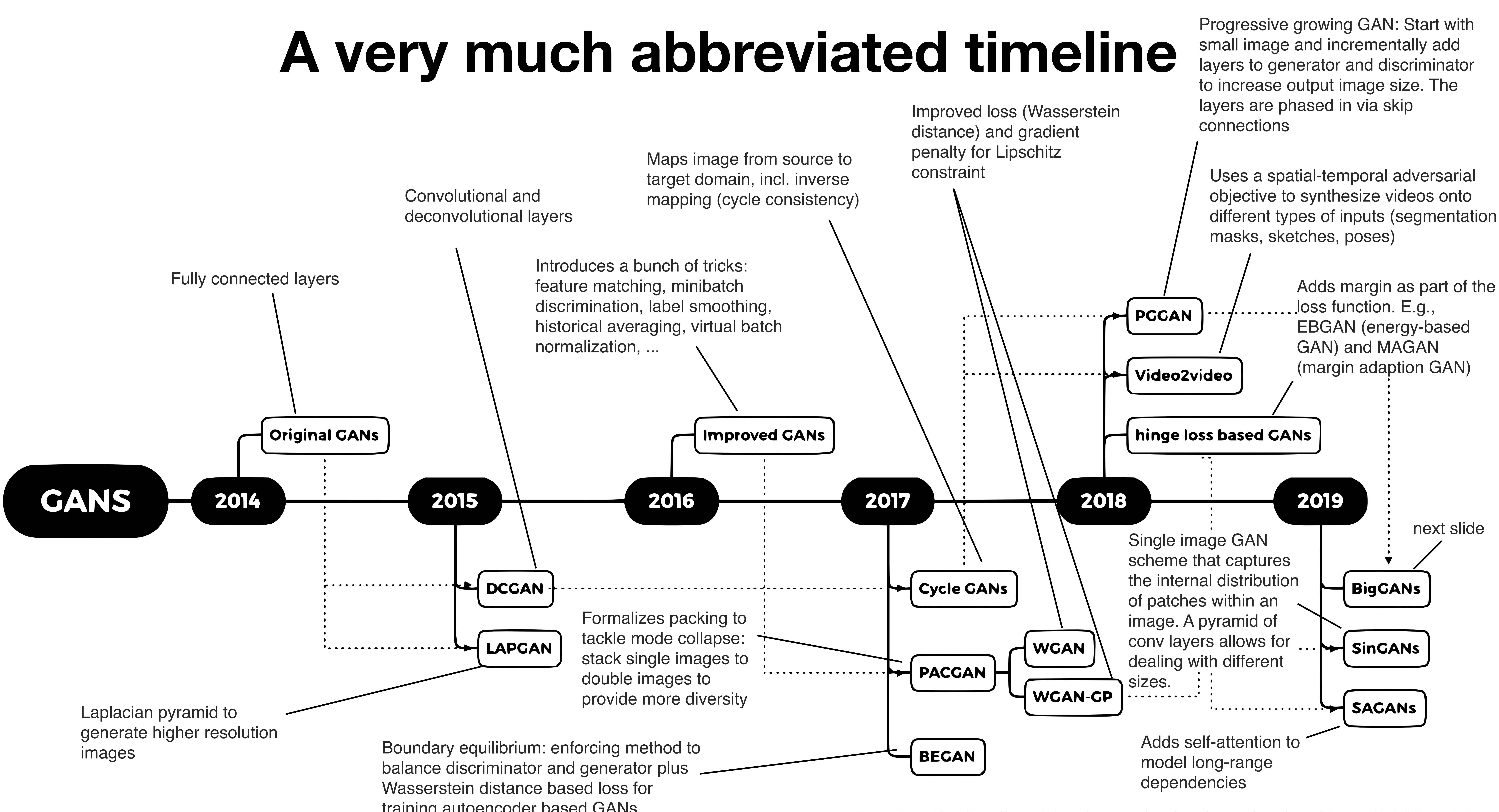


Figure adapted from https://towardsdatascience.com/a-review-of-generative-adversarial-networks-9af21e94bda4

BigGANs

Published as a conference paper at ICLR 2019

LARGE SCALE GAN TRAINING FOR HIGH FIDELITY NATURAL IMAGE SYNTHESIS

Andrew Brock[*,†]
Heriot-Watt University
ajb5@hw.ac.uk

Jeff Donahue[†]
DeepMind
jeffdonahue@google.com

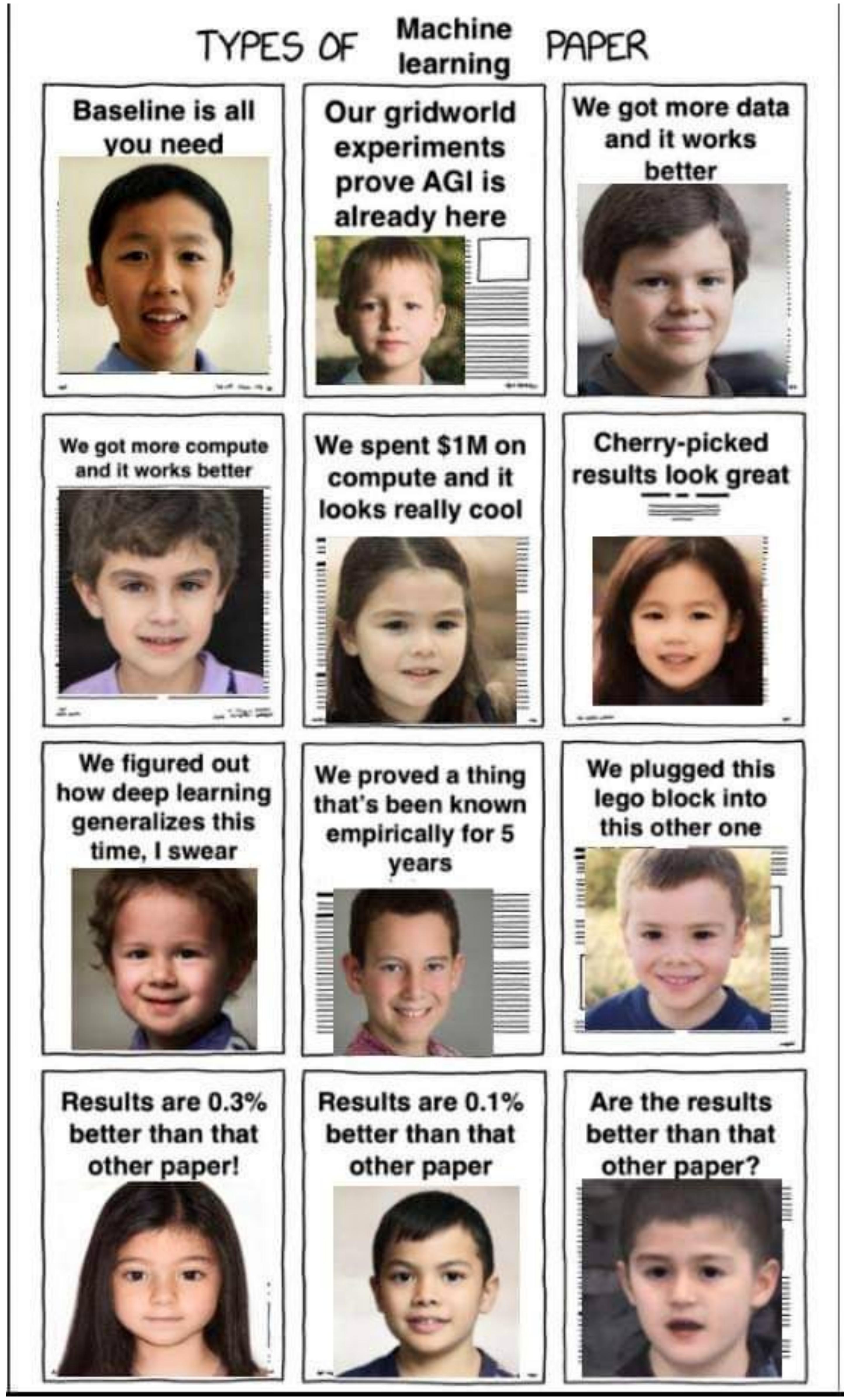
Karen Simonyan[†]
DeepMind
simonyan@google.com

https://arxiv.org/pdf/1905.01164.pdf

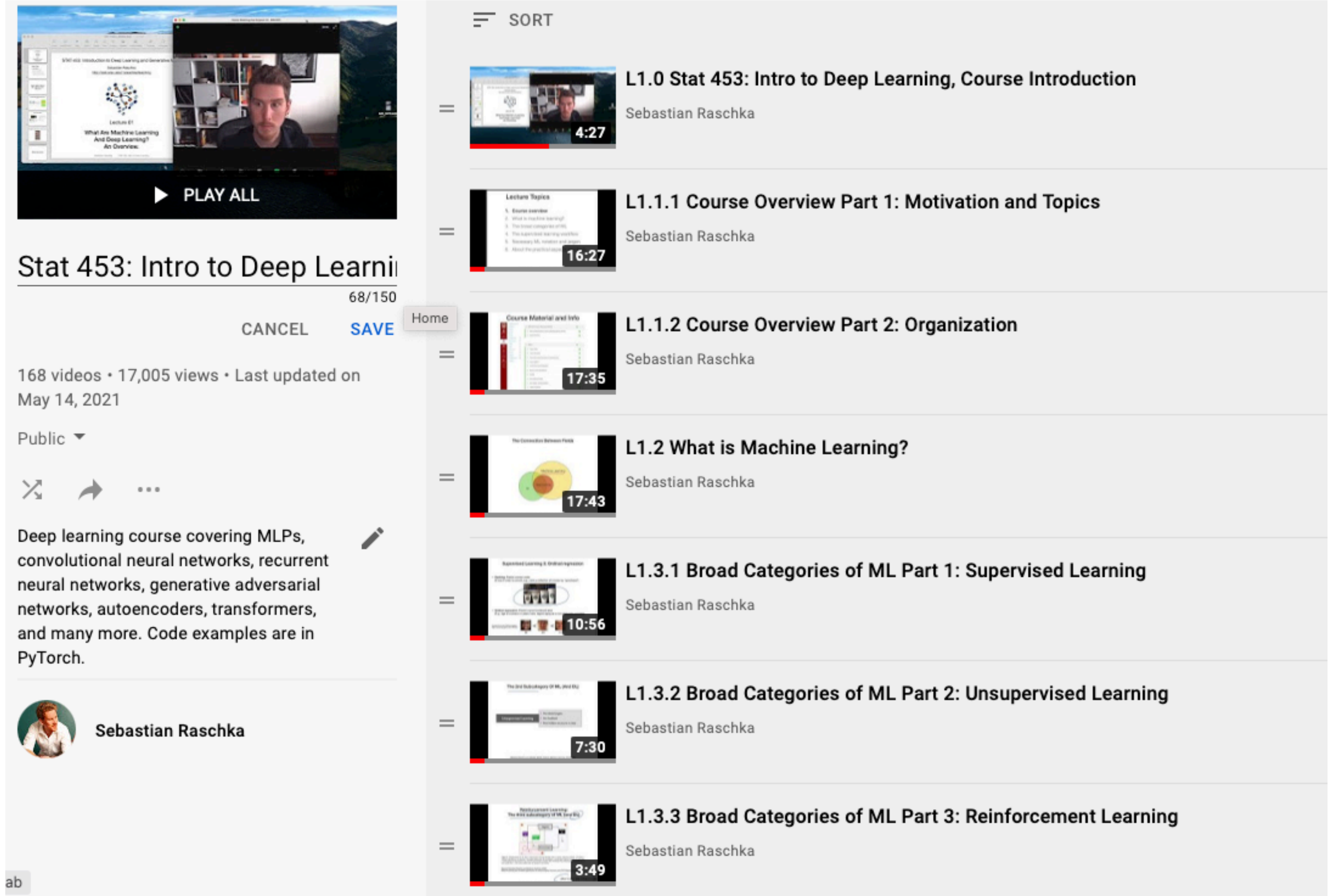
Figure 1: Class-conditional samples generated by our model.

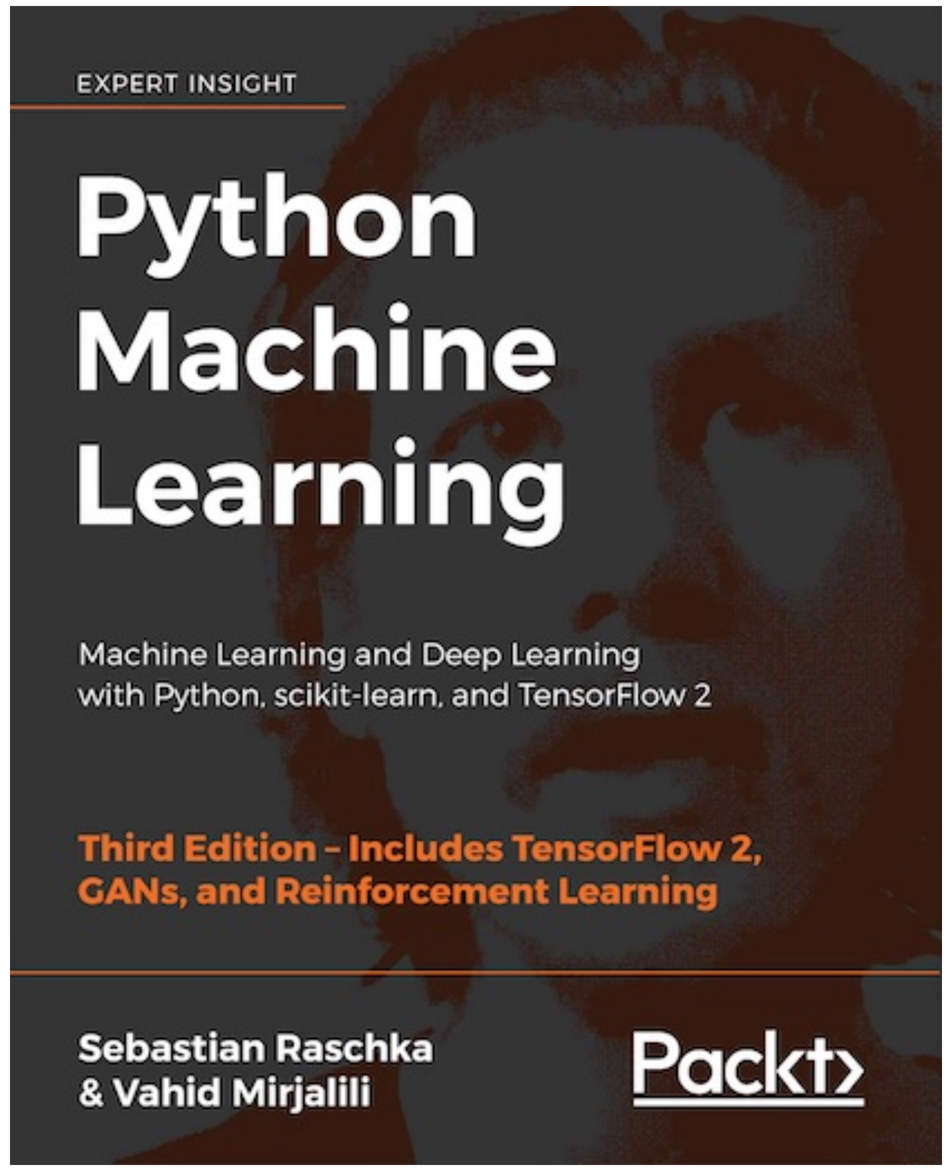
- A class-conditional GAN with scaled up model and batch size
- Uses self-attention (based on SAGAN) and hinge loss
- Uses conditional BatchNorm, spectral normalization for weights, the truncation trick (truncated Gaussian during inference), and many other tricks

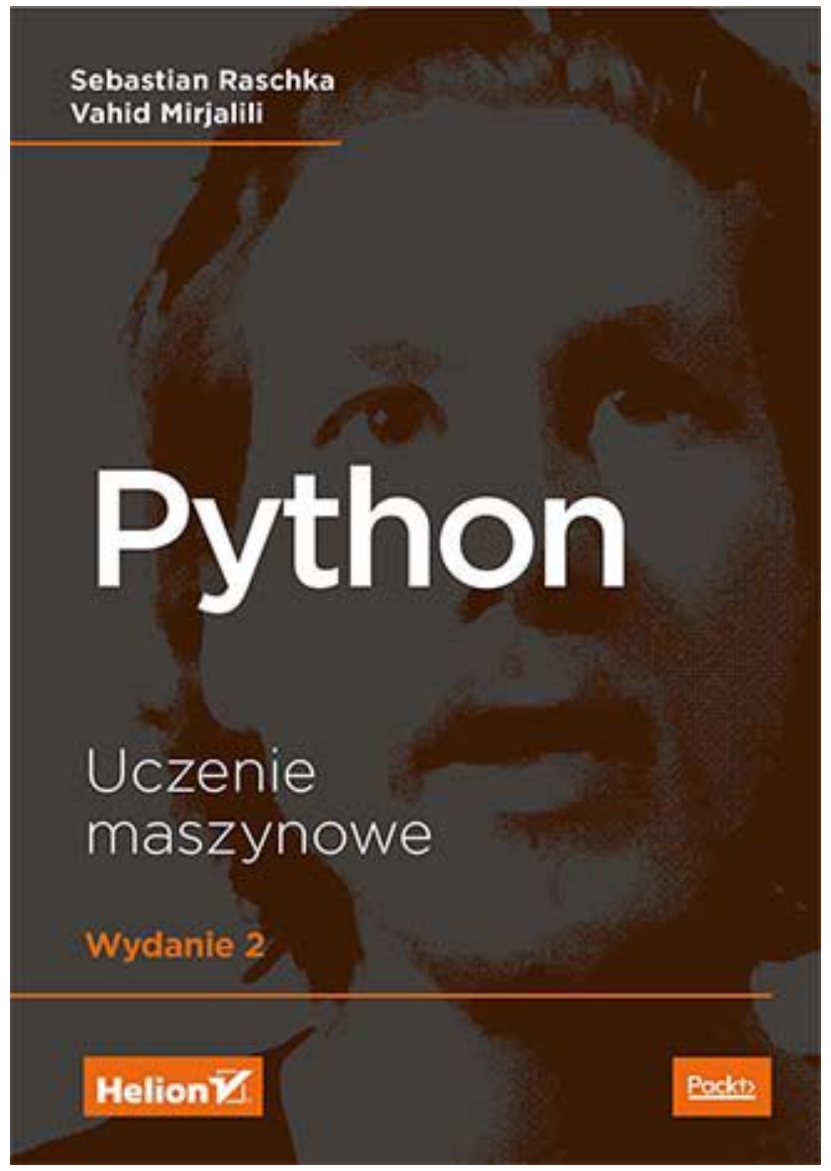
GANs are fun!



https://twitter.com/TheInsaneApp/status/1408516210099064833?s=20







https://sebastianraschka.com/books/

YouTube https://www.youtube.com/playlist?list=PLTKMiZHVd_2KJtIXOW0zFhFfBaJJilH51

@rasbt