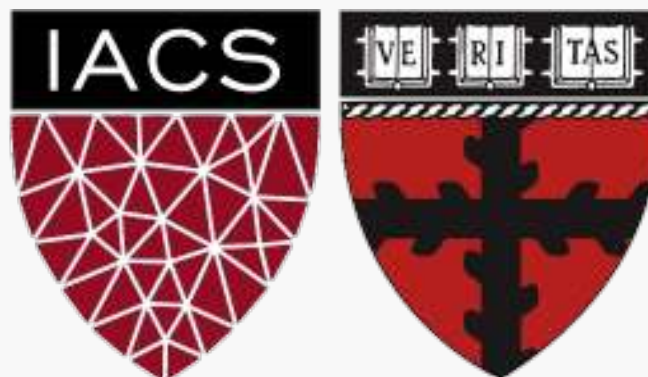


Lecture 30: Generative Adversarial Networks (GANS) – part 2

CS109B Data Science 2

Pavlos Protopapas, Mark Glickman, and Chris Tanner





Ian Goodfellow

@goodfellow_ian

Follow



One of my favorite samples from the Progressive GANs paper is this one from the "cat" category. Apparently some of the cat training photos were memes with text. The GAN doesn't know what text is so it has made up new text-like imagery in the right place for a meme caption.



11:41 AM - 3 Dec 2017



Lecture outline

- Review
- Generative Models
 - Mode collapse
- Evaluate GANs
 - Inception score
 - Train Synthetic Test Real
 - Fréchet Distance
- State of the Art GANs
 - Deep Convolutional GAN: DCGAN
 - Auxiliary Classifier GANs
 - Recent updates
- Concluding remarks



Lecture outline

- **Review**
- Generative Models
 - Mode collapse
- Evaluate GANs
 - Inception score
 - Train Synthetic Test Real
 - Fréchet Distance
- State of the Art GANs
 - Deep Convolutional GAN: DCGAN
 - Auxiliary Classifier GANs
 - Recent updates
- Concluding remarks



Quiz Q1

Consider the loss function from the original GAN (Goodfellow et al. 2014):

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

What would be its value for a D that is perfectly fooled by G?


Fooled means D can not tell
real from fake

Possible interpretations “perfectly fooled” as:

1. $D(x_{\text{real}})$ is whatever and $D(x_{\text{fake}}) = 1$. This would give $\log(0)$ which is $-\infty$. None of the answers work.
2. $D(x_{\text{real}}) = 0$ and $D(x_{\text{fake}}) = 1$ which again will result in infinities. No match.
3. $D(x_{\text{real}}) = \frac{1}{2}$ and $D(x_{\text{fake}}) = \frac{1}{2}$ which means the Discriminator has no idea what is going on.

Note: $\log(1/2) = -\log(2)$



 $-\log(4)$



Quiz Q2

Consider the min-max game described in the original GAN (Goodfellow et al. 2014):

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

Which of the following pair of losses is its equivalent "gradient **descent**" form?

Descent means minimize

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

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

G minimizes this: $\mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$



$$\begin{aligned} L_D &= \mathbb{E}_{\mathbf{x}} [-\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}} [-\log(1 - D(G(\mathbf{z})))] \\ L_G &= \mathbb{E}_{\mathbf{z}} [\log(1 - D(G(\mathbf{z})))] \end{aligned}$$

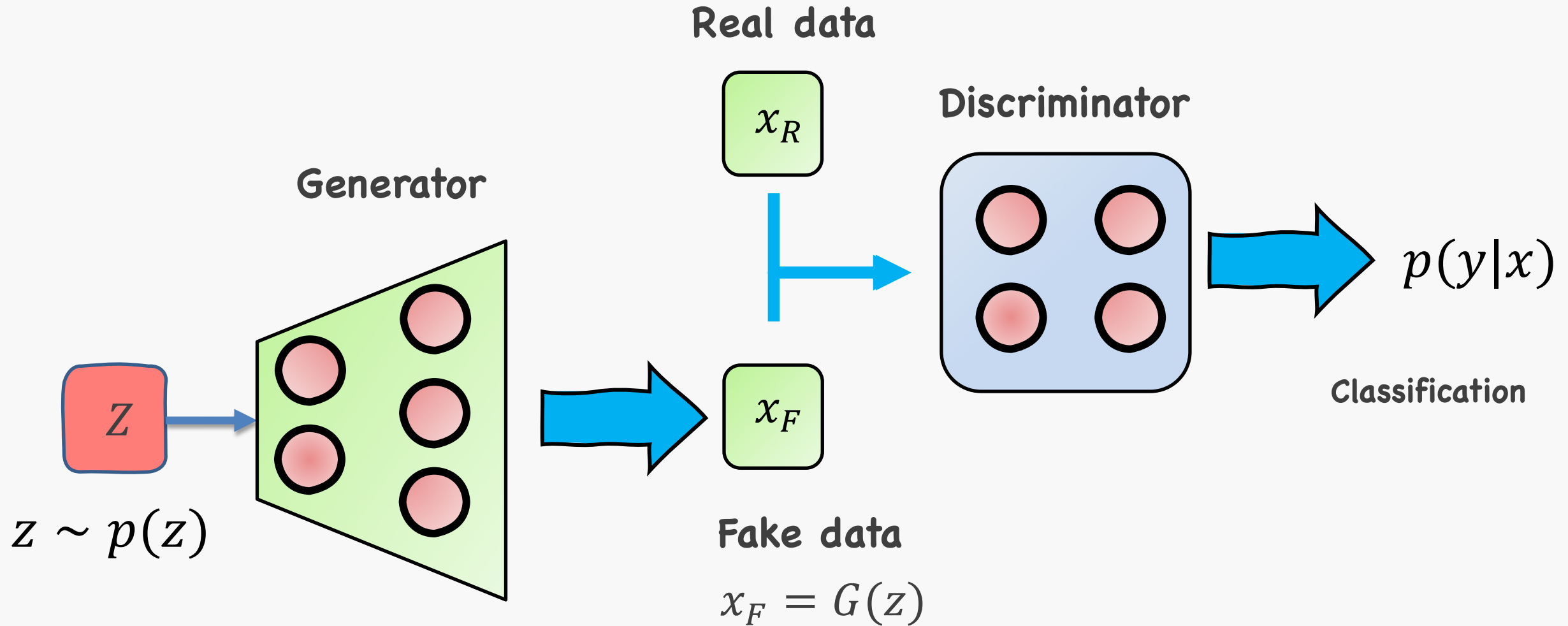


Image-to-Image Translation using Cycle-GANs

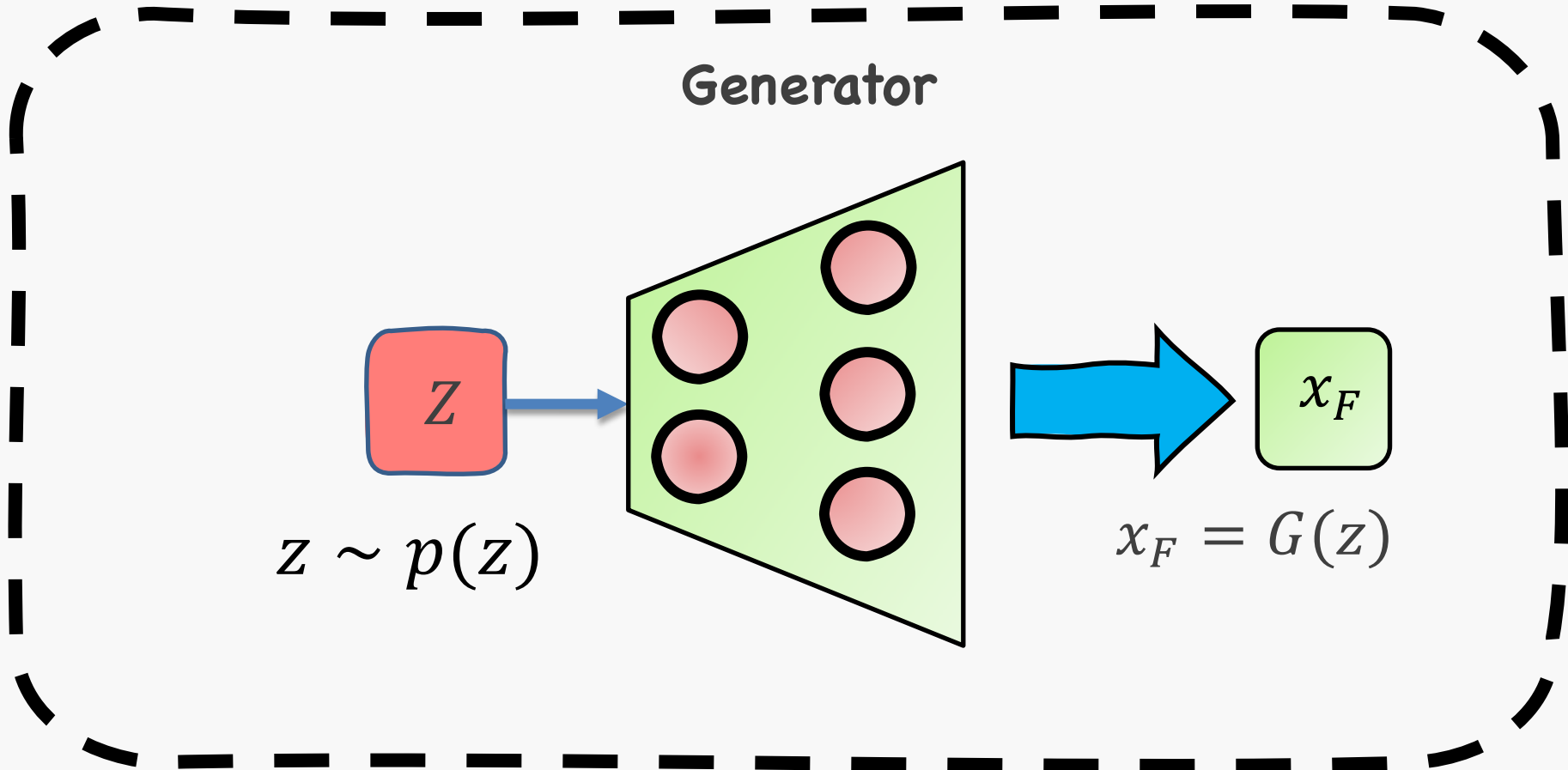


[\[Zhu et al. 2017\]](#)

How we build a generator from GANS



Generative Model



Game Theory

In some games there are unbounded resources. For example, in a game of poker, the pot can theoretically get larger and larger without limit.

Zero-sum game: Players compete for a fixed and limited pool of resources. Players compete for resources, claiming them and each player's total number of resources can change, but the total number of resources remain constant.

In zero-sum games each player can try to set things up so that the other player's best move is of as little advantage as possible. This is called a **minimax**, or **minmax**, technique.



Game Theory (cont)

Our goal in training the GAN is to produce two networks that are each as good as they can be. In other words, we don't end up with a “winner.”

Instead, both networks have reached their peak ability given the other network's abilities to prevent it. Game theorists call this state a **Nash equilibrium**, where each network is at its best configuration with respect to the other.



Challenges

Biggest challenge to using GANs in practice is their sensitivity to both structure and parameters.

If either the discriminator or generator gets better than the other too quickly, the other will never be able to catch up.

Also, there is no proof that they will **converge**.

GANs do seem to perform very well most of the time when we find the right parameters, but there's no guarantee beyond that.



Challenges: Using Big Samples

Trying to train a GAN generator to produce large images, such as 1000 by 1000 pixels can be problematic.

The problem is that with large images, it's easy for the discriminator to tell the generated fakes from the real images.

Many pixels can lead to error gradients that cause the generator's output to move in almost random directions, rather than getting closer to matching the inputs.

Compute power, memory, and time to process large numbers of these big samples.



Challenges: Using Big Samples (cont)

- Start by resizing the images: 512x512, 128x128, 64x64, ... ,4x4.
- Then build a small generator and discriminator, each with just a few layers of convolution.
- Train with the 4 by 4 images until it does well.
- Add a few more convolution layers to the end network, and now train them with 8 by 8 images. Again, when the results are good, add some more convolution layers to the end of each network and train them on 16 by 16 images.

This process takes much less time to complete than if we'd trained with only the full-sized images from the start.



Challenges: Mode collapse

I would like to use GAN to produce faces like the ones below from NVIDIA [Karras, Laine, Aila / Nvidia].



Challenges: Mode collapse (cont)

The generator somehow finds one image that fools the discriminator.



Challenges: Mode collapse (cont)



A generator could then just produce that image every time independently of the input noise.

The discriminator will always say it is real, so the generator has accomplished its goal and stops learning.

However: The problem is that every sample made by the generator is identical.

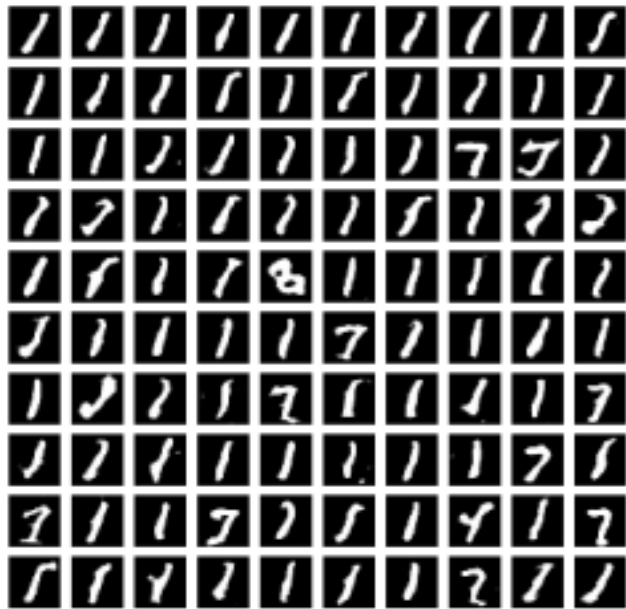
This problem of producing just one successful output over and over is called **mode collapse**.



Challenges: Modal collapse (cont)

Much more common is when the system produces the same few outputs, or minor variations of them.

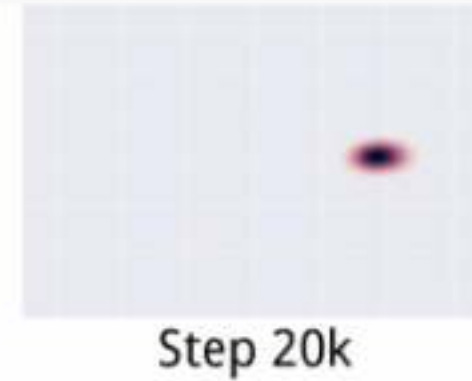
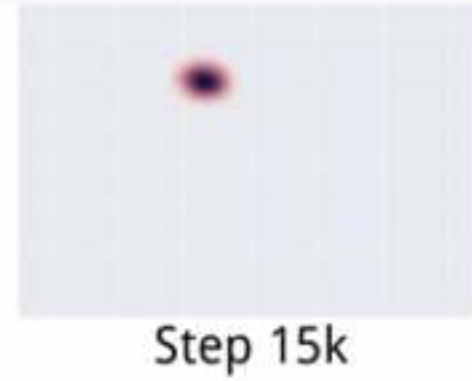
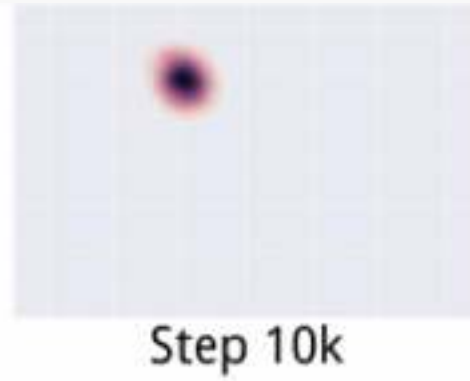
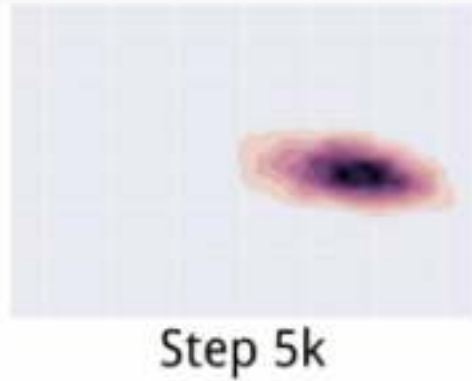
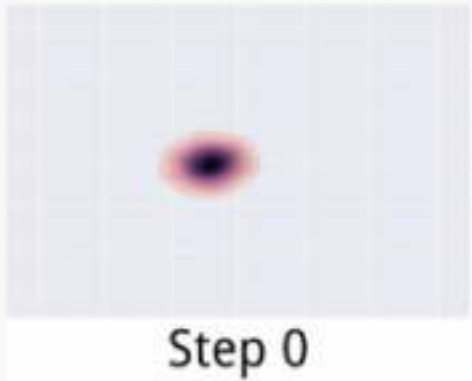
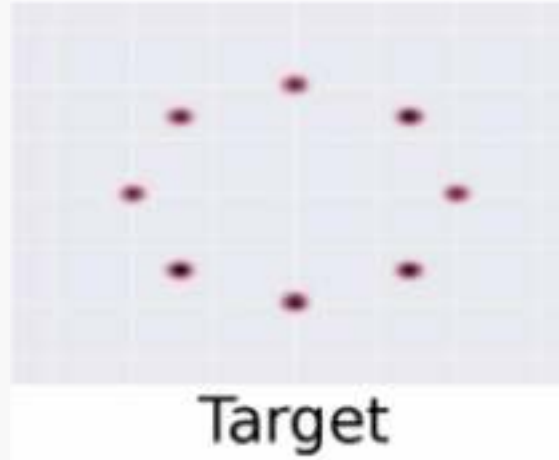
This is called partial mode collapse



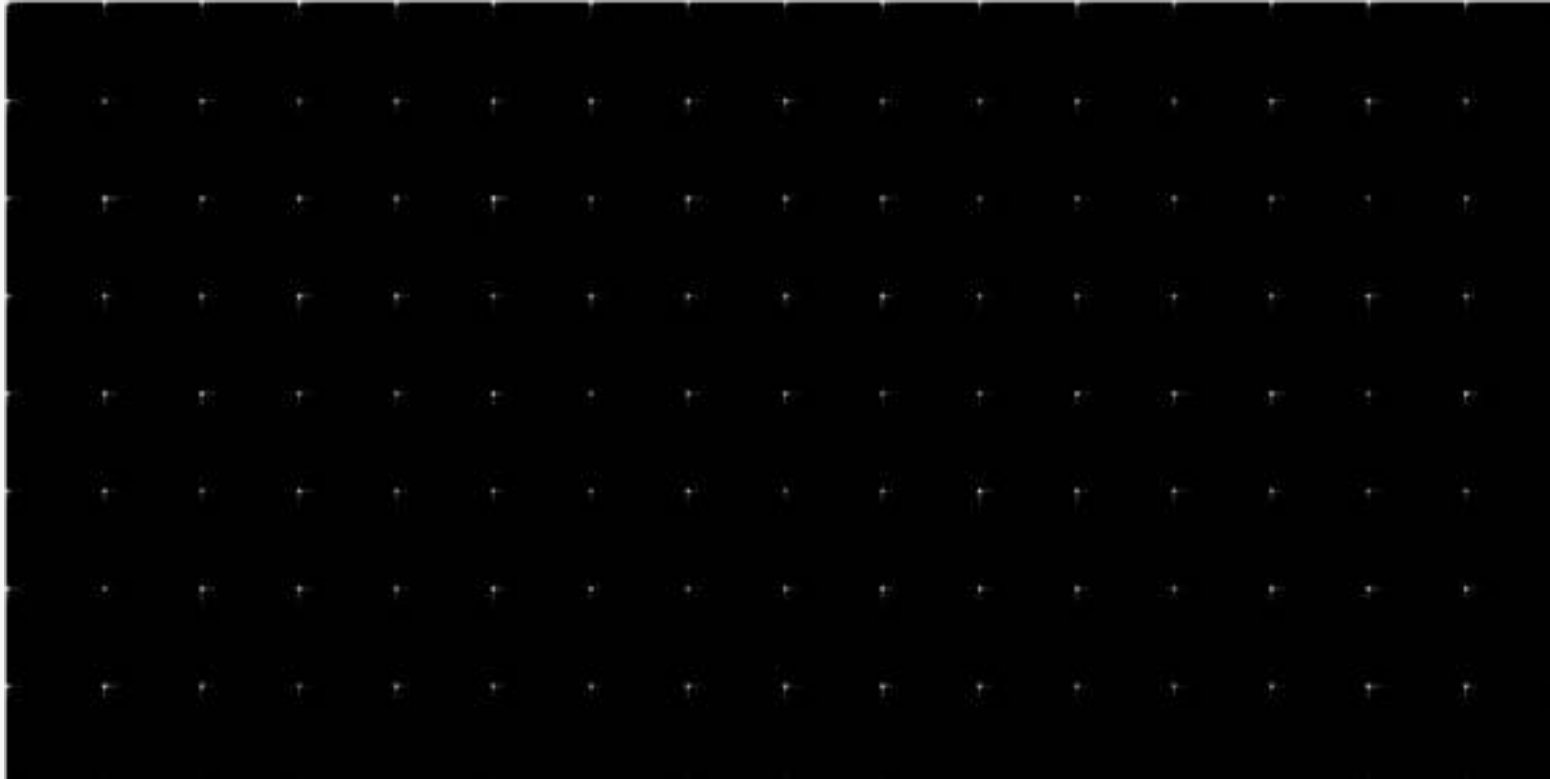
Solution:

- Extend the loss function with an additional terms to measure the diversity of the outputs produced.
- If the outputs are all the same, or nearly the same, the discriminator can assign a larger error to the result.
- The generator will diversify because that action will reduce the error.
- Use WGAN (see below)

Challenges: Modal collapse (cont)



Challenges: Modal collapse (cont)

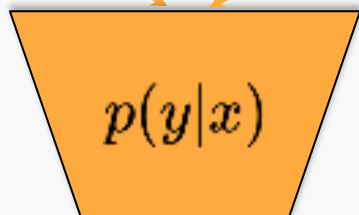


Credit: G. Garcia et al. 2018



Evaluating GANs: Why is it challenging

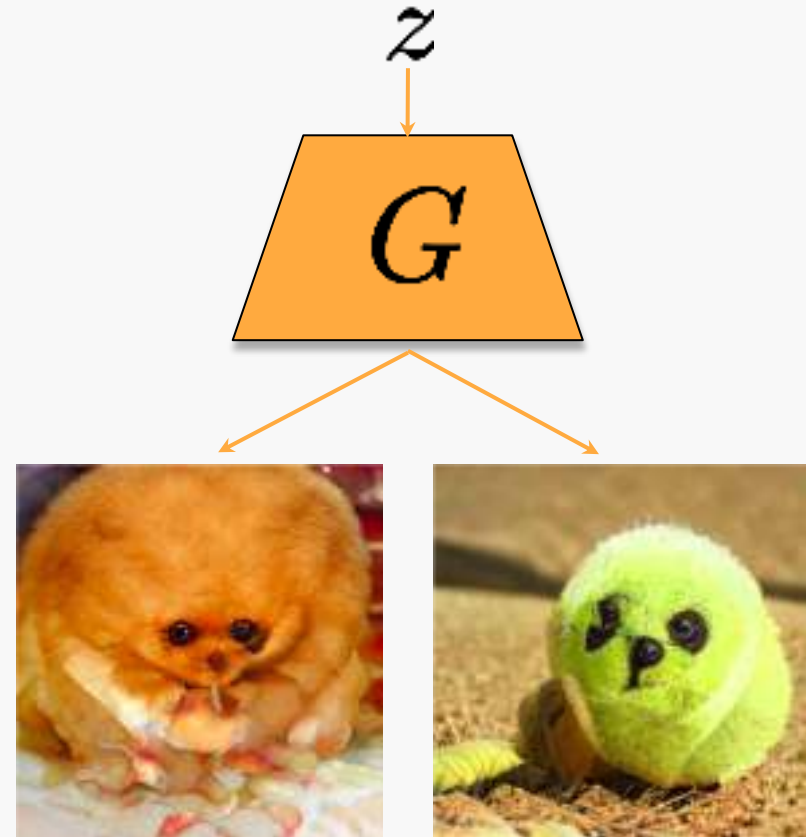
Supervised Learning



Pomeranian



GANs



??

~~Could we use D?~~

Evaluating GANs: What do we want from a Generator?

Fidelity: quality of images



Diversity: variety of images



Evaluating GANs

How to evaluate the models ?

- Human annotators
- inception Score (IS)
- Fréchet Inception Distance (FID)
- Others... (IS*, SWD)

Evaluation

Given a generative model, we generate images $x = G(z)$.

In the ideal case:

1. x has a diverse distribution i.e. it covers a wide range of original data distribution
2. x have good quality

We could try to classify x , and get $p(y|x)$.

If 1) holds: if we put together all the classifications, we could expect a uniform distribution $\rightarrow p(y)$ should be very wide

If 2) holds: $\rightarrow p(y|x)$ should be very narrow since there shouldn't be uncertainty when classifying

$p(y)$ and $p(y|x)$ should be very different !



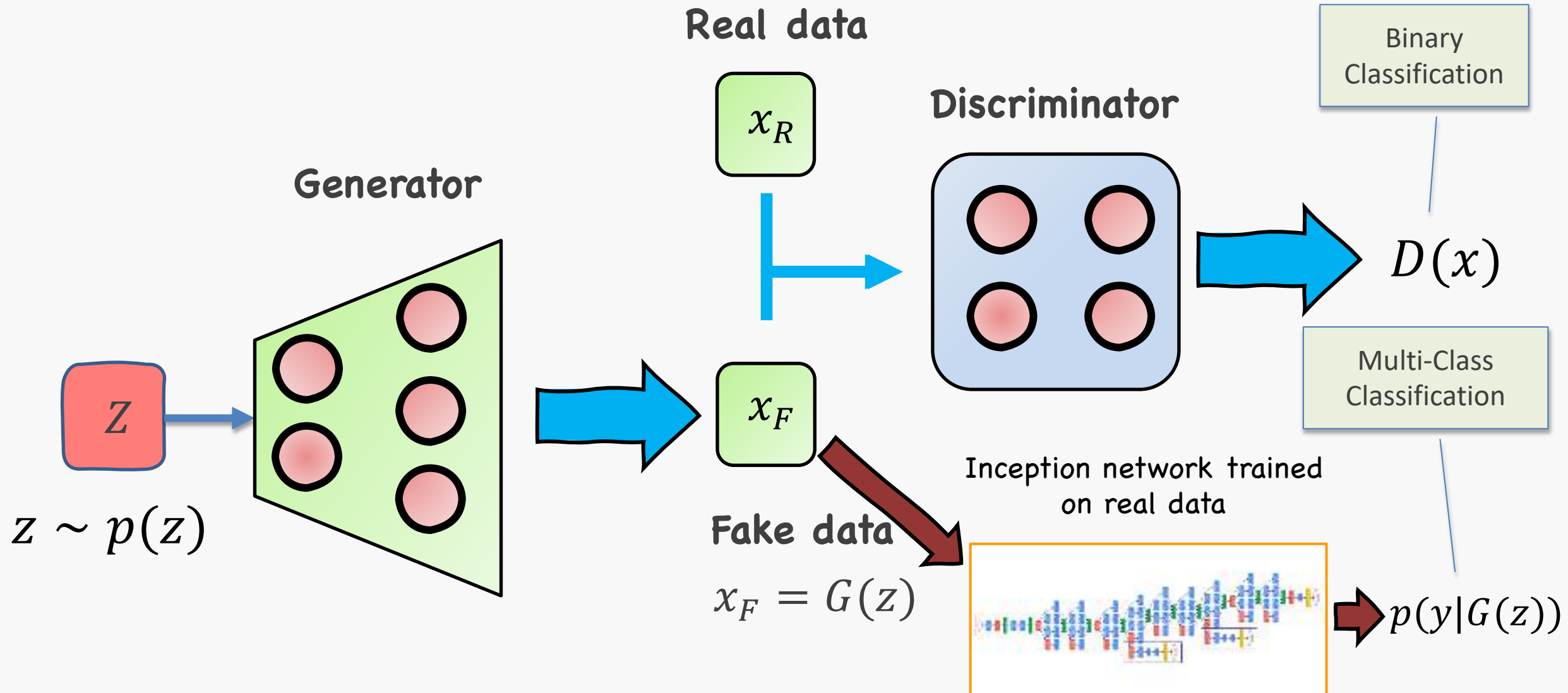
Evaluation (cont)

In order to do this, we need a good classifier.

In the context of images, why don't we use the inception network, and then call it inception score.



How we build a generator from GANS



Evaluation: Inception Score

Inception network trained on real data



$p(y|G(z))$ would be very high if the network is sure about the class (high entropy)

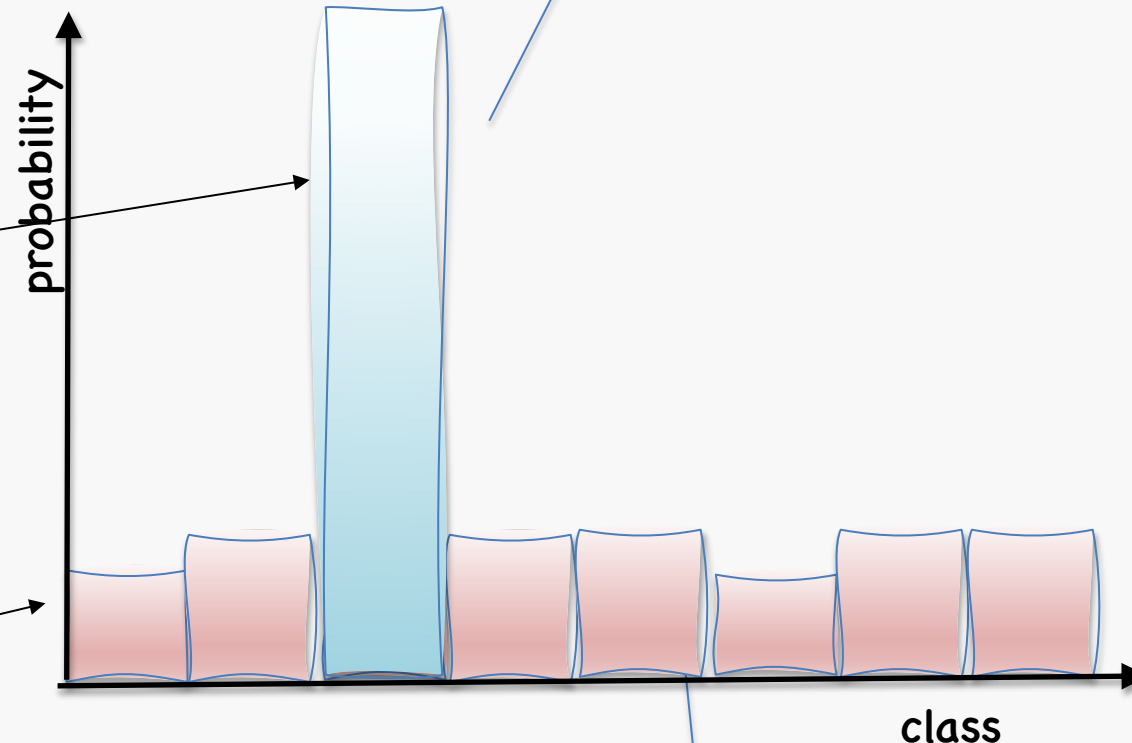
$\rightarrow p(y|G(z))$



$$p(y) = \int p(y|G(z))dz$$

High Inception Score means high quality and diverse generated data.

Distribution of labels which should be uniform (low entropy)



High quality means peaky distribution

KL measures the difference of the two distributions. If we want high quality and diverse, KL has to be high.

Diversity means flat distribution

Inception Score:

$$IS(G) = \exp(\mathbb{E}_{x_F \sim p_g} D_{KL}(p(y|x_F) || p(y)))$$



Evaluation: Inception Score

Inception Score is used for evaluation not for training.

Using IS for training, yields to weird results; see figure to the right.



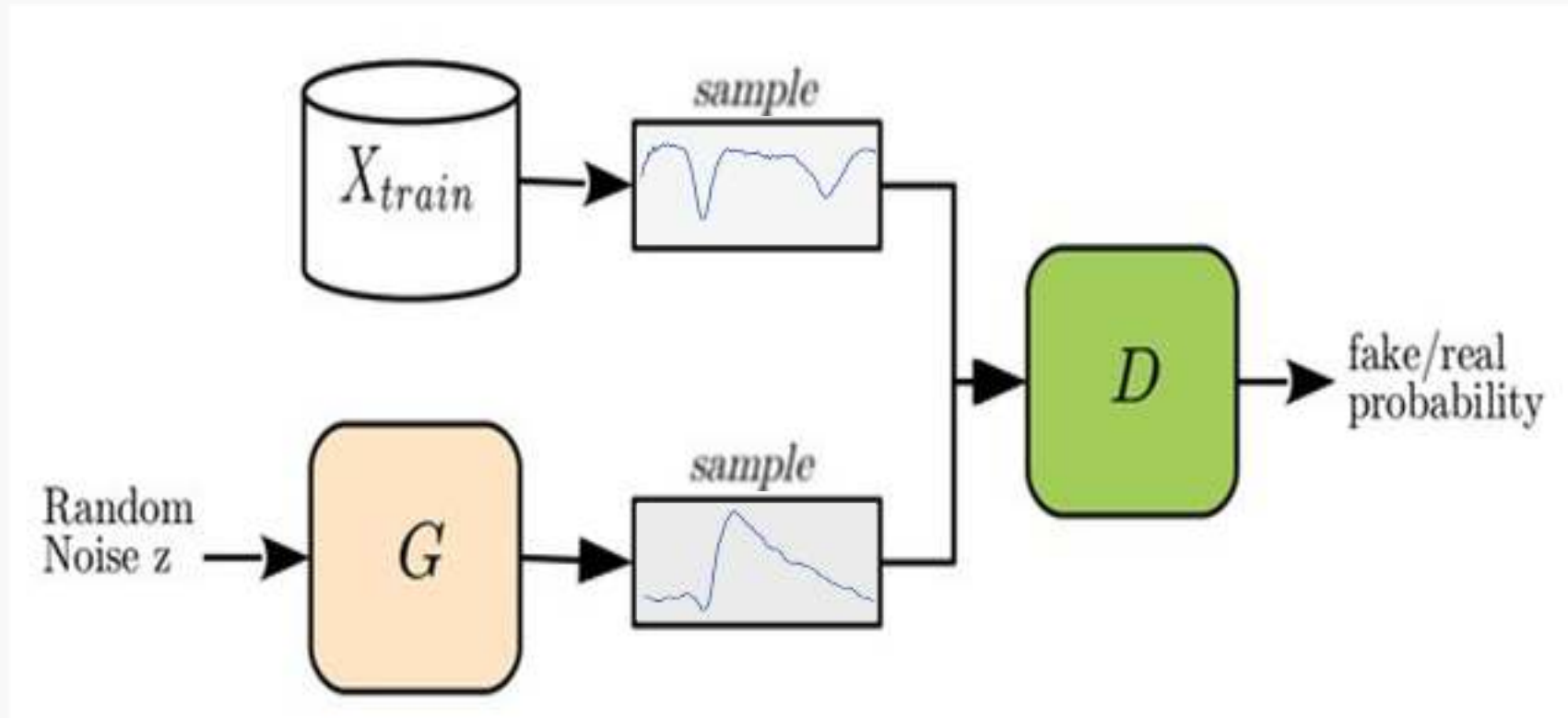
Figure 1. Sample of generated images achieving an Inception Score of 900.15. The maximum achievable Inception Score is 1000, and the highest achieved in the literature is on the order of 10.

Evaluation: Train Synthetic, Test Real: TSTR

Train classifier on synthetic, test on real (TSTR).

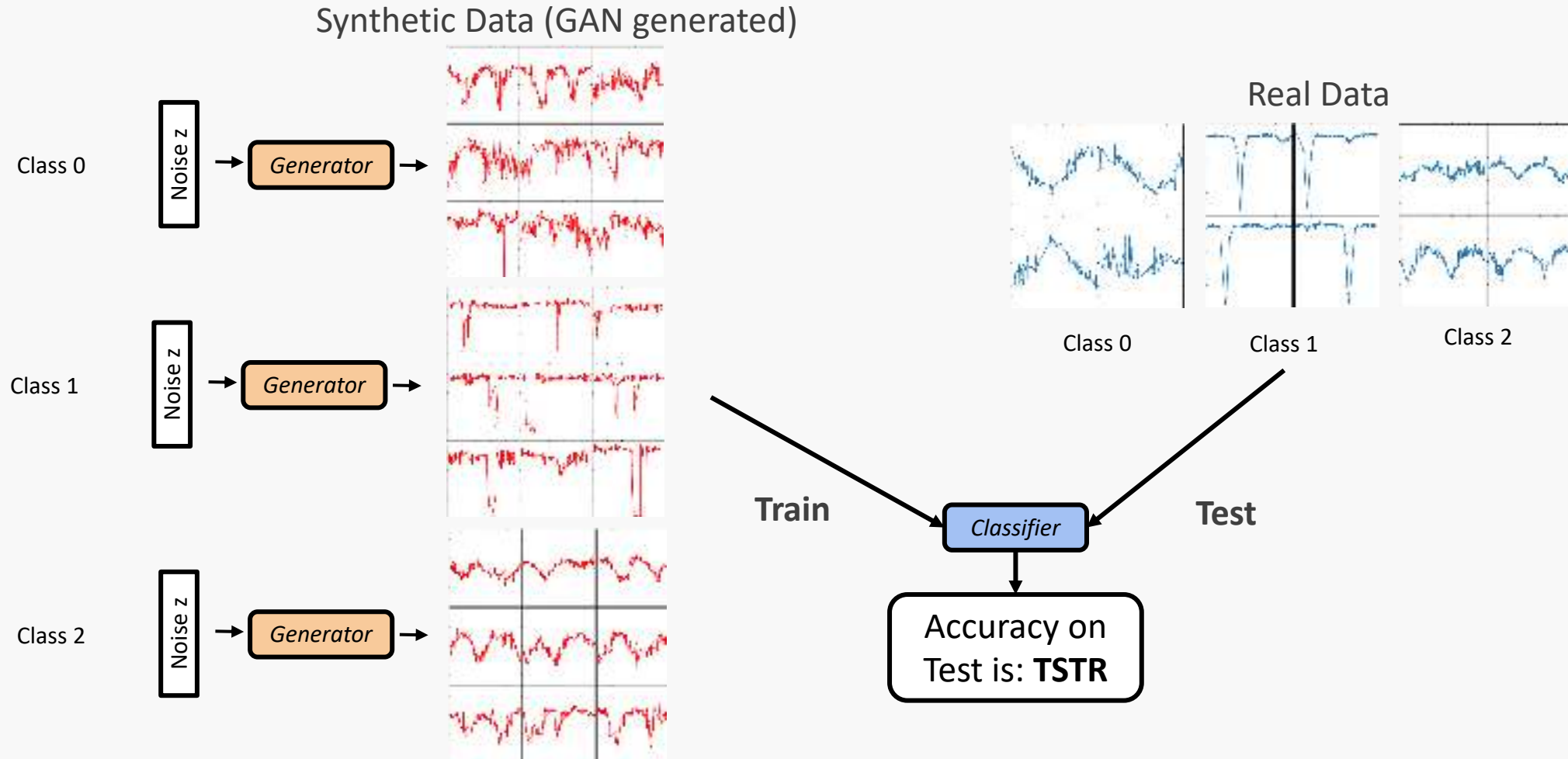
If synthetic data are of high quality then we expect $TSTR \geq TRTR$

Example: GAN model trained on time series data.



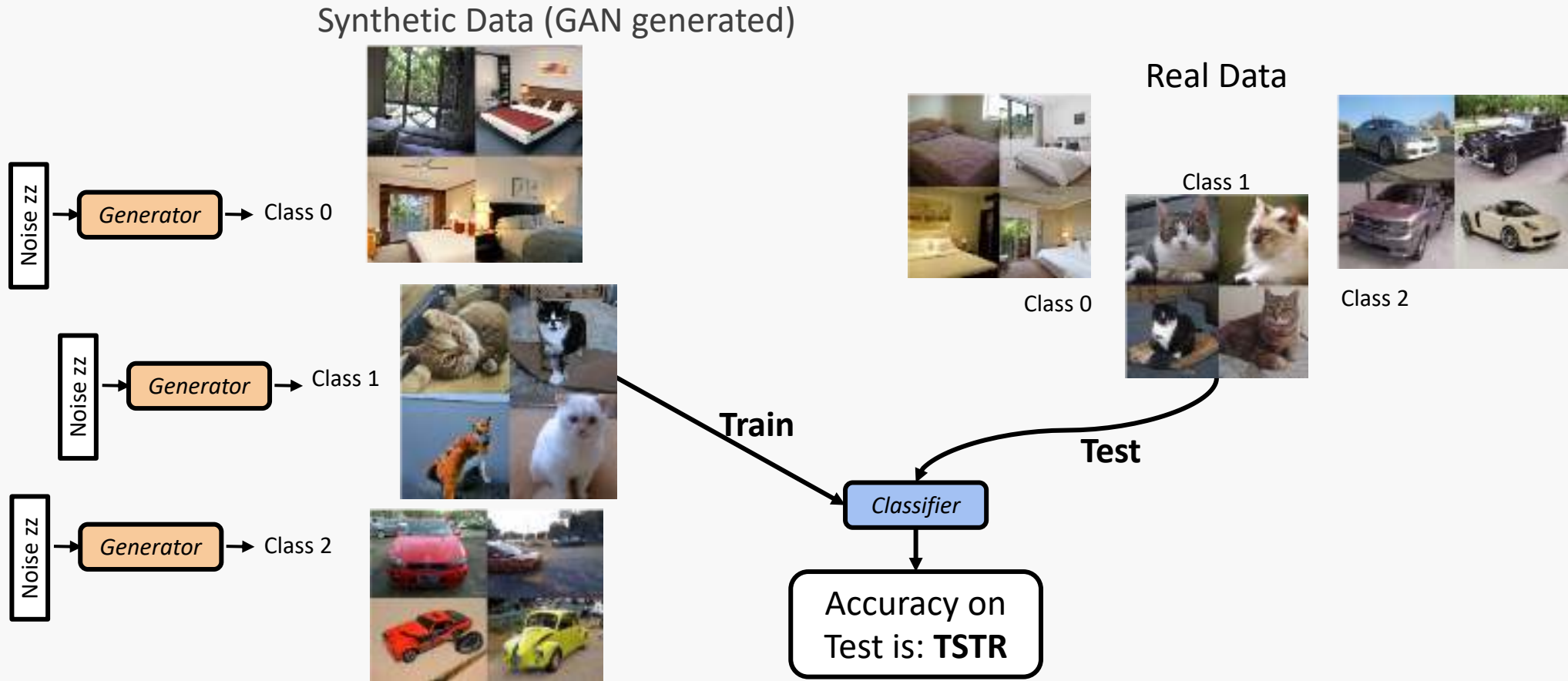
Evaluation: TSTR (cont)

Train on Synthetic Test on Real (TSTR)



Evaluation: TSTR (cont)

Test on Synthetic Train on Real (TSTR)

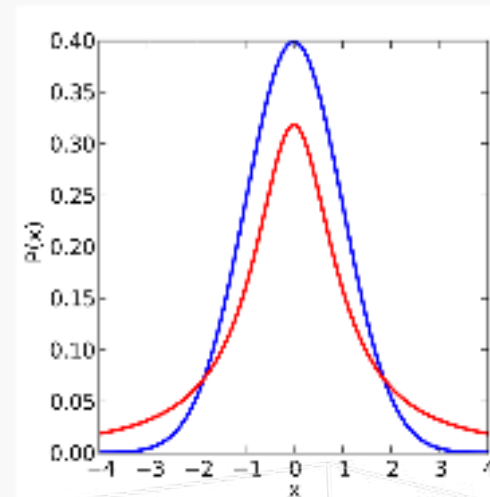


Evaluating GANs: **Fréchet Distance**

When we deal with distributions:

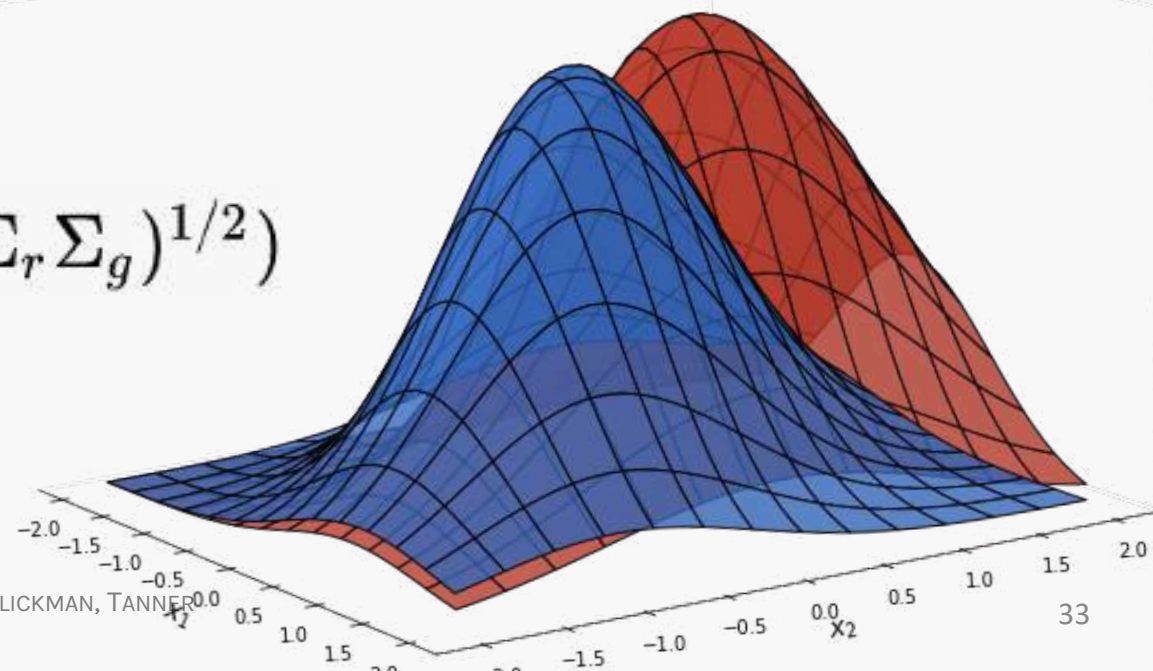
1-D normals:

$$d(X_r, X_g) = (\mu_r - \mu_g)^2 + (\sigma_r - \sigma_g)^2$$

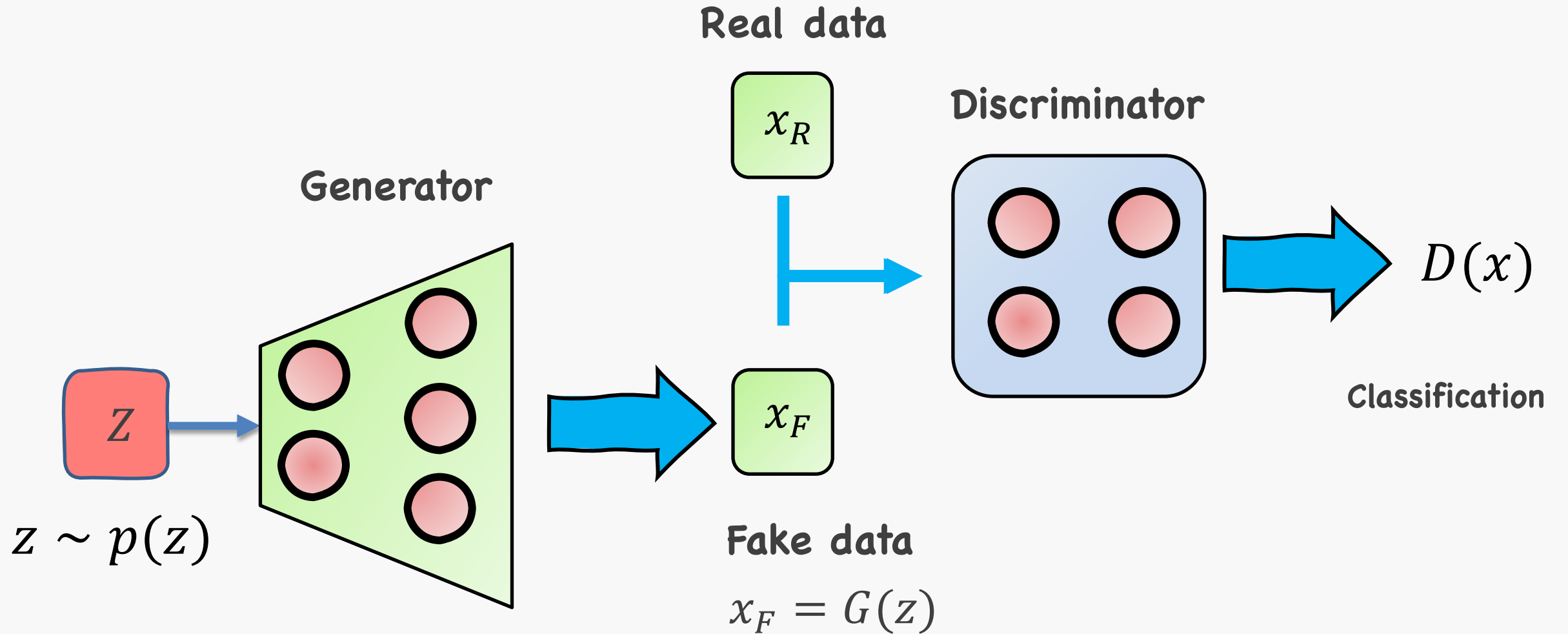


Multivariable normals:

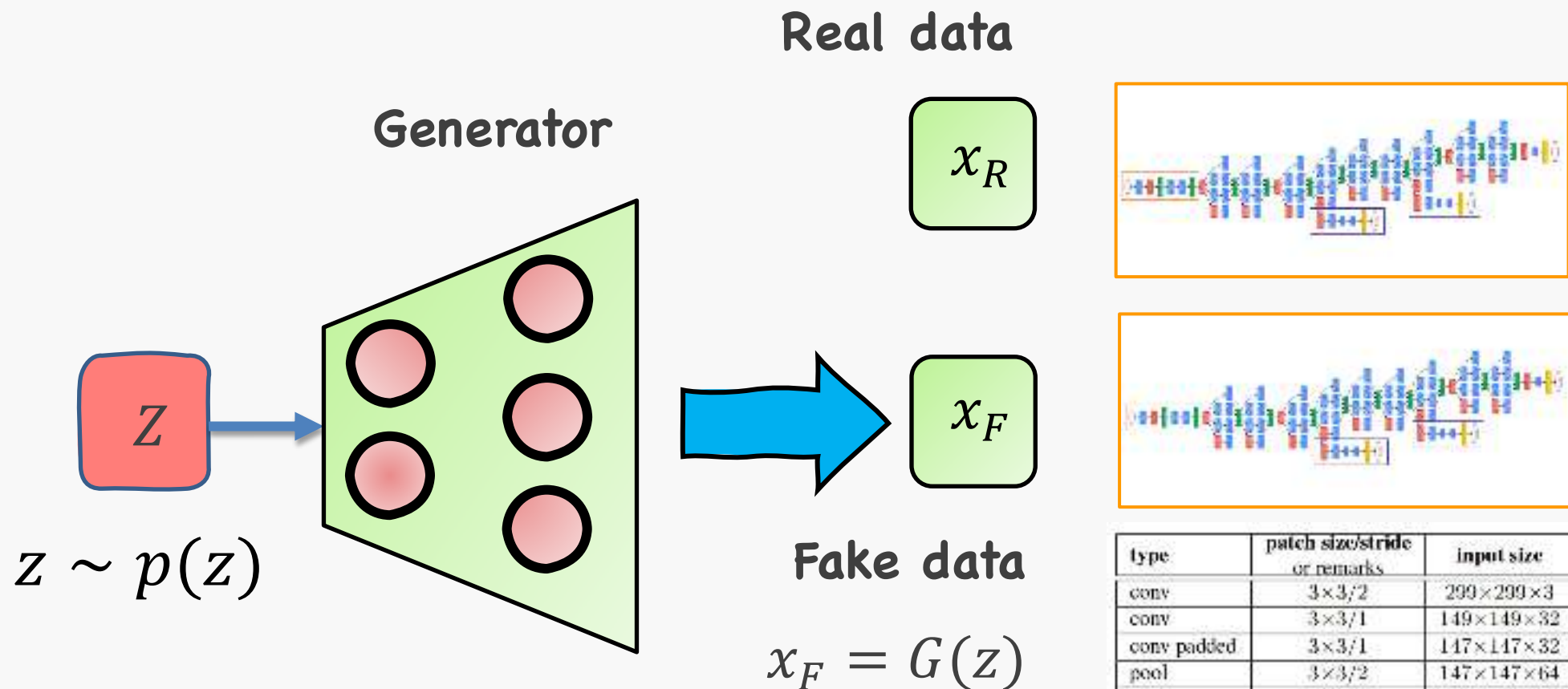
$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$



Evaluating GANs: Fréchet Inception Distance



Evaluating GANs: Fréchet Inception Distance

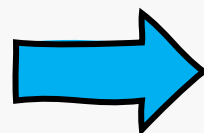


type	patch size/stride or remarks	input size
conv	$3 \times 3/2$	$299 \times 299 \times 3$
conv	$3 \times 3/1$	$149 \times 149 \times 32$
conv padded	$3 \times 3/1$	$147 \times 147 \times 32$
pool	$3 \times 3/2$	$147 \times 147 \times 64$
conv	$3 \times 3/1$	$73 \times 73 \times 64$
conv	$3 \times 3/2$	$71 \times 71 \times 80$
conv	$3 \times 3/1$	$35 \times 35 \times 192$
3×Inception	As in figure 5	$35 \times 35 \times 288$
5×Inception	As in figure 6	$17 \times 17 \times 768$
2×Inception	As in figure 7	$8 \times 8 \times 1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

Evaluating GANs: Fréchet Inception Distance

Real data

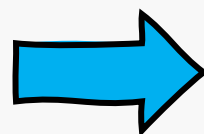
x_R



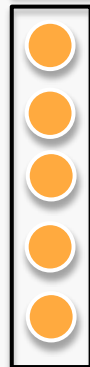
2048 dimensions



x_F



2048 dimensions



Fake data

$$x_F = G(z)$$

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8×8	8×8×2048
linear	logits	1×1×2048
softmax	classifier	1×1×1000

Assuming that the activations distribute as multivariate Gaussians, compute the W_2 distance between the distributions

$$X_r \sim \mathcal{N}(\mu_r, \Sigma_r)$$

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr} \left(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2} \right)$$

In practice, we use large batches, take μ_g, Σ_g and the average the μ_g, Σ_g 's over batches.



GANs: Brief timeline

Which one is real? Are you a good Discriminator?



GANs: Brief timeline

7 years of progress in faces



2014



2015



2016



2017



2018



2021

https://deepgenerativemodels.github.io/assets/slides/cs236_lecture9.pdf



GANs: Brief timeline

~2 years of progress in ImageNet



Odena et al
2016

Auxiliary



Miyato et al
2017

Spectral Normalization



Zhang et al
2018

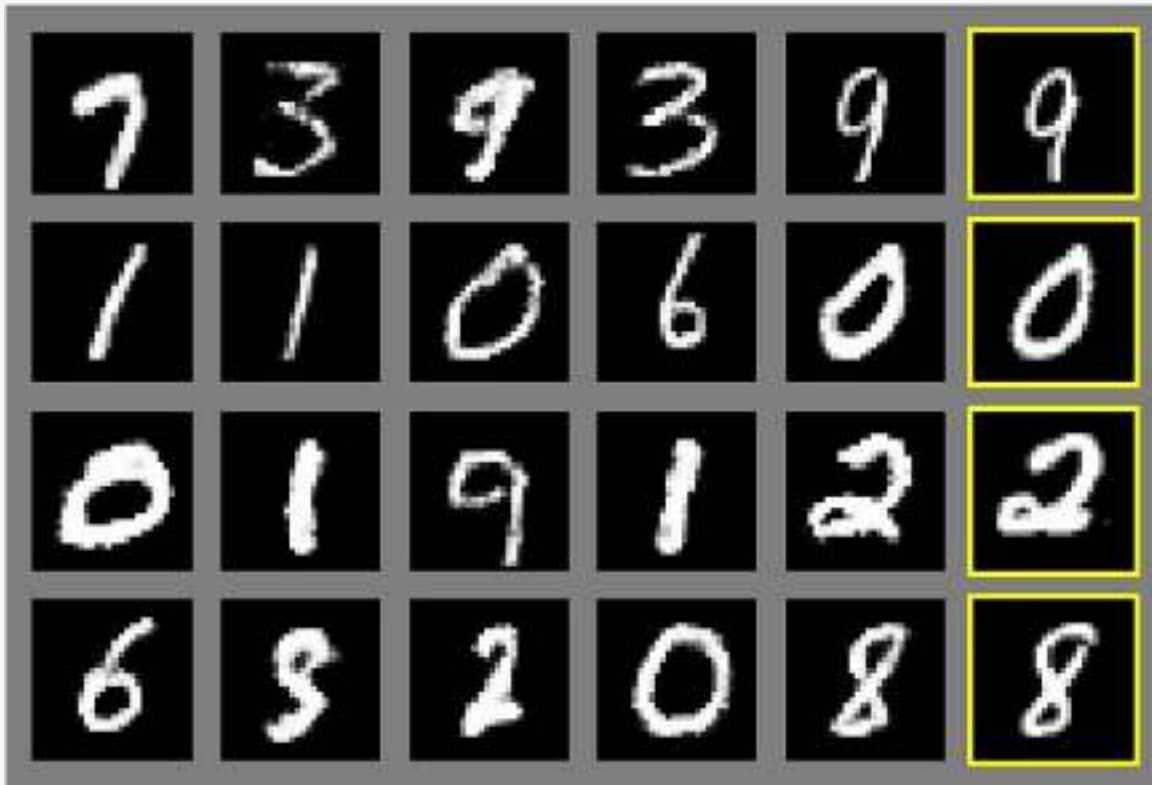
Self-Attention



Brock et al
2018

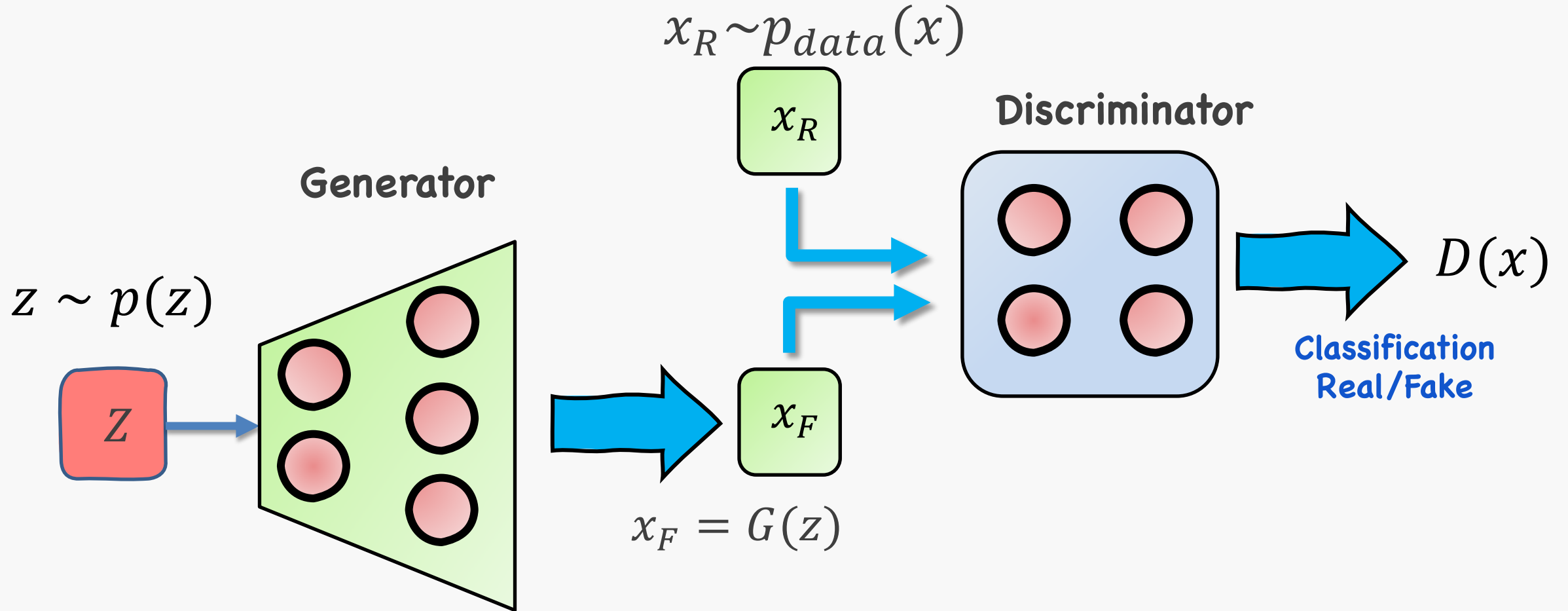
Large Scale GAN Training

GANs: Brief timeline – Original GAN

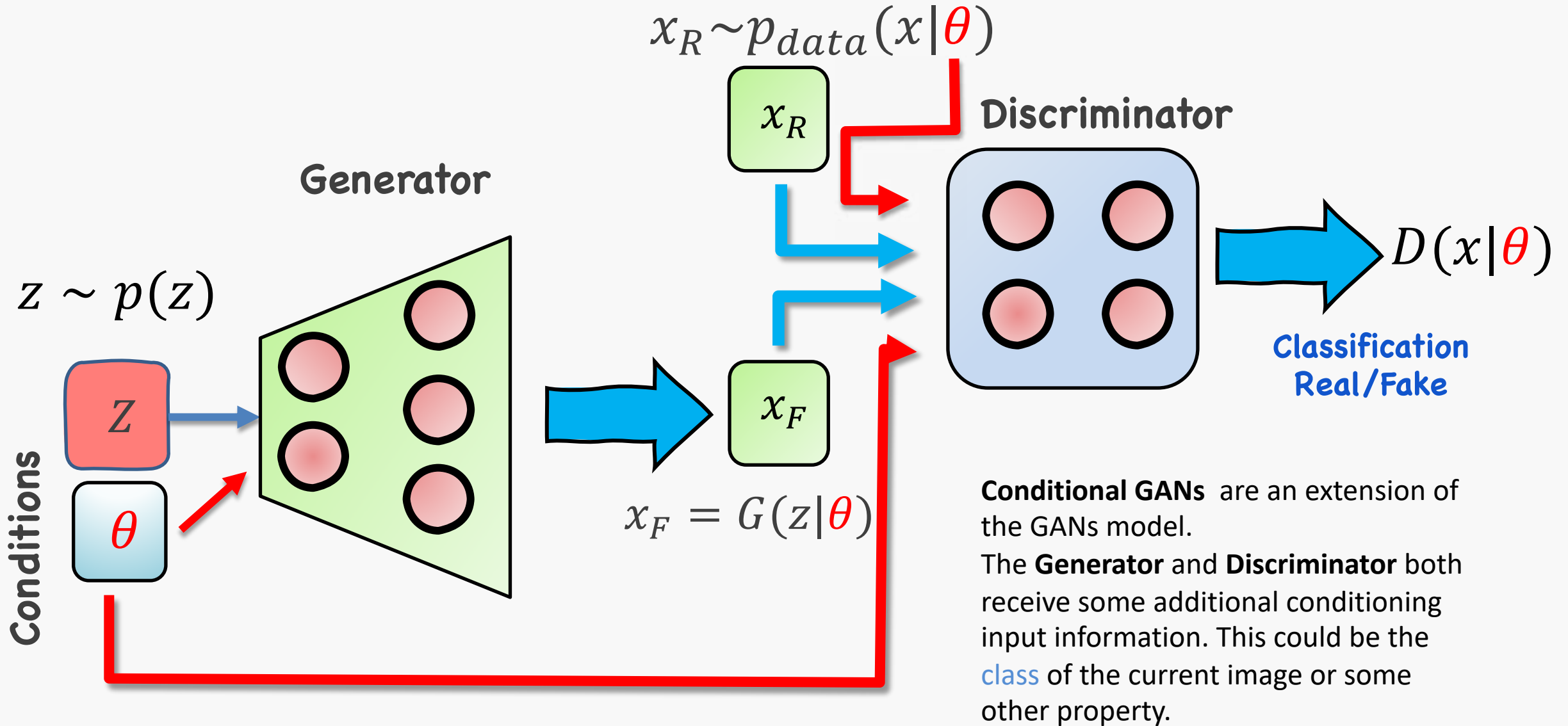


[\[Goodfellow et al. 2014\]](#)

GANs: Brief timeline: Conditional GAN



GANs: Brief timeline: Conditional GAN



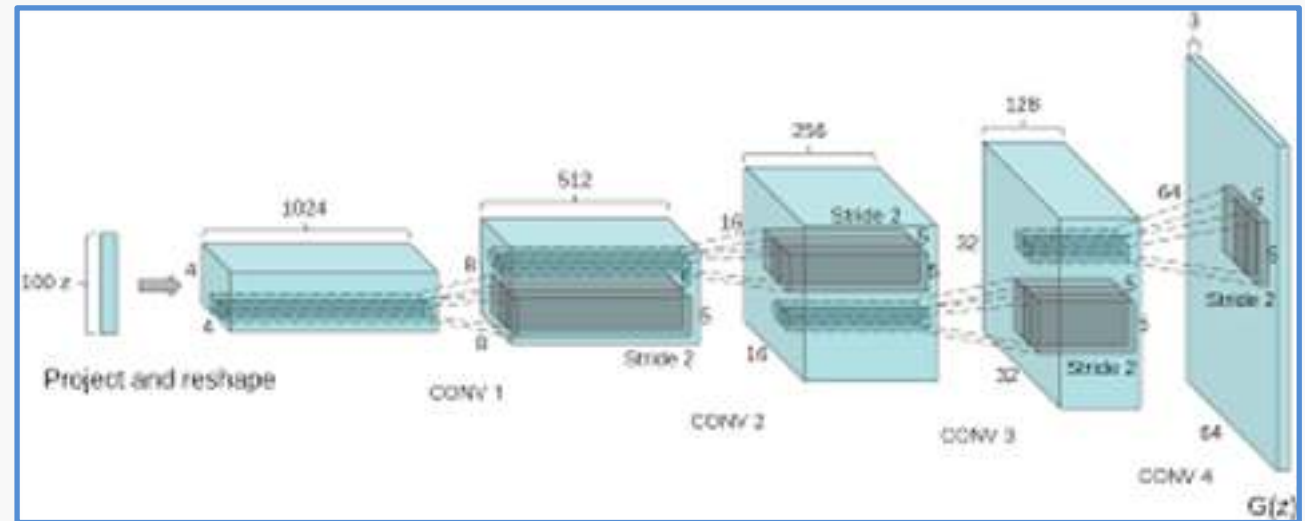
GANs: Brief timeline: Conditional GAN



[\[Mirza et al. 2014\]](#)

Deep Convolutional GAN: DCGAN

- Eliminate fully connected layers
- Replace all max pooling with convolutional stride
- Use transposed convolution or simply upsampling
- Batchnorm in G and D except for the input and output layer
- ReLU in G for all layers except output, which uses a tanh

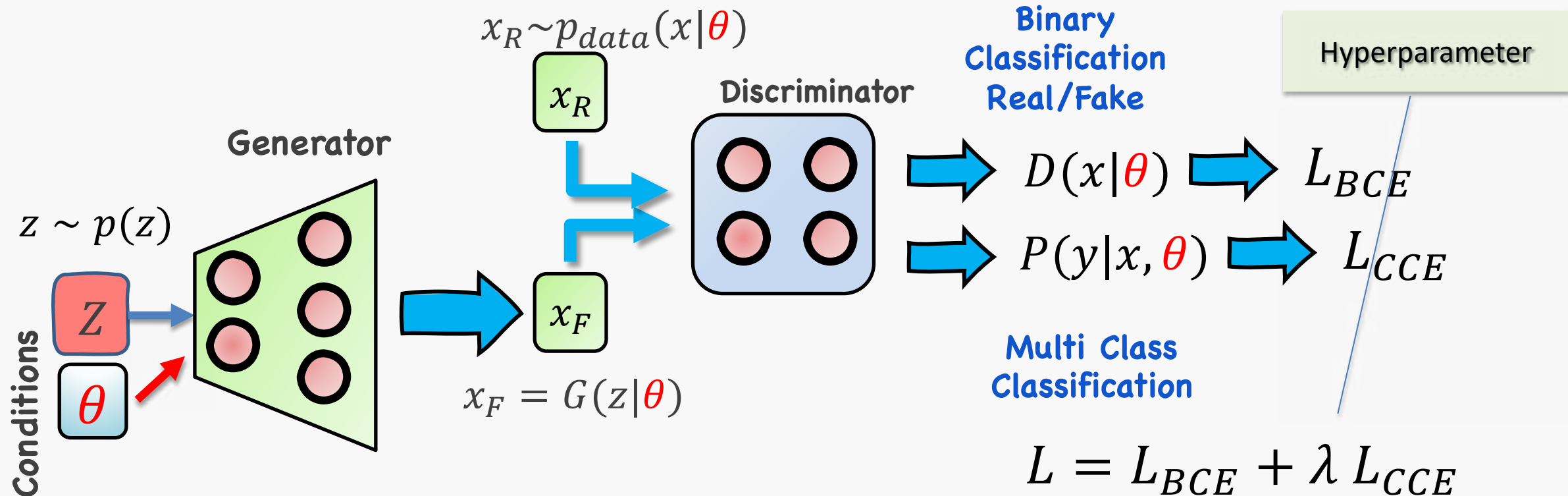


Deep Convolutional GAN: DCGAN

- Eliminate fully connected layers
- Replace all max pooling with convolutional stride
- Use transposed convolution or simply upsampling
- Batchnorm in G and D except for the input and output layer
- ReLU in G for all layers except output, which uses a tanh



Auxiliary Classifier Generative Adversarial Network



FALSE NEGATIVES: Discriminator is passed a fake and it calls it real but the wrong class. In vanilla GAN, the generator would be happy but in this case the Generator still updates.

Auxiliary Classifier Generative Adversarial Network



monarch butterfly



goldfinch



daisy



redshank

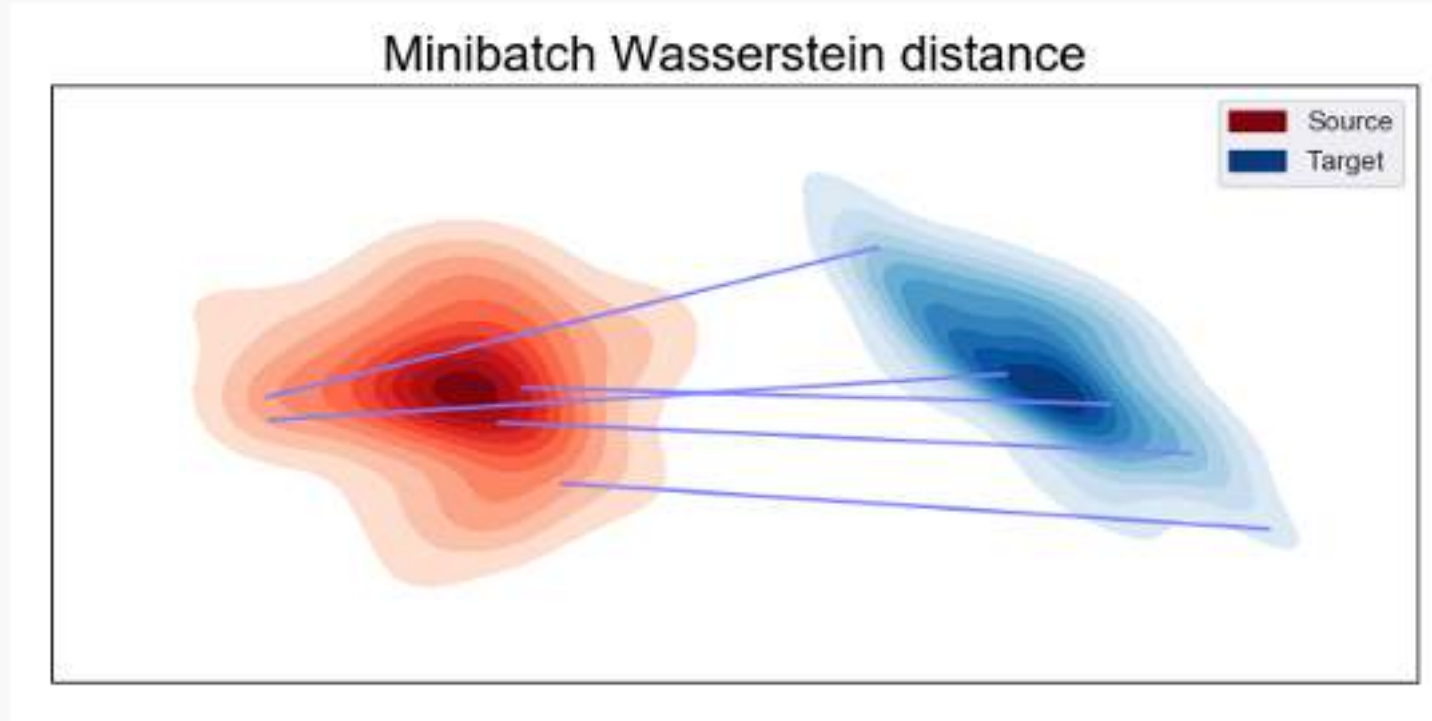


grey whale

[\[Odena et al. 2016\]](#)

Wasserstein GAN

They propose a GAN framework base on a metric that correlates with image quality: The Earth Mover Distance, or **Wasserstein Distance** [Arjovsky et al. 2017]



Wasserstein GAN

Distance is everything:

In general, generative models seek to minimize the distance between **real and learned distribution**.

Wasserstein (also EM, Earth-Mover) distance:

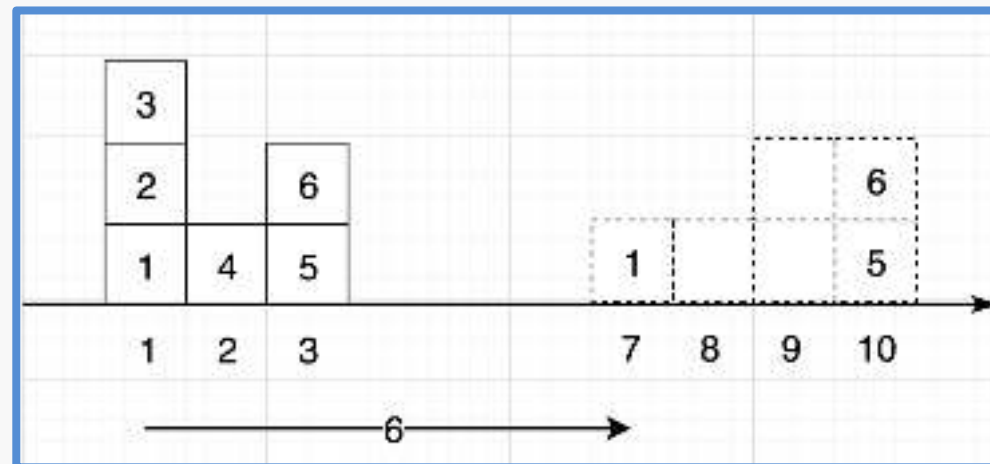
“Informally, if the distributions are interpreted as two different ways of piling up a certain amount of dirt over the region D , the **Wasserstein distance** is the **minimum cost** of turning one pile into the other; where the cost is assumed to be amount of dirt moved times the distance by which it is moved.”



Imagine we started with a distribution P_r and wanted to move mass around to change it into P_g . Moving mass m by distance d costs $m \cdot d$. To execute this, for all (x,y) , move $\gamma(x,y)$ mass from x to y

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|]$$

Among all the transport plans, the earth moving (EM) distance corresponds to the cost of the optimal transport plan.



Wasserstein GAN

- Exact computation is intractable.
- Idea: Use a NN to **approximate** Wasserstein distance.
- Here, we re-use the discriminator, whose outputs are now **unbounded**.
- We define a custom loss function:

$$W = \frac{1}{n} \sum_i y_{i,pred} y_{i,true}$$

y_{true} here is chosen from $\{-1, 1\}$ according to real/fake

Idea: make predictions for real as large as possible, and for fakes as small as possible



Wasserstein GAN

The authors claim:

- Higher stability during training, less need for carefully balancing generator and discriminator.
- Meaningful loss metric, correlating well with sample quality.
- Mode collapse is rare.
- Learns faster because it does not suffer as much from Vanishing gradients.



Wasserstein GAN

Tips for implementing Wasserstein GAN in Keras.

- Leave the discriminator output unbounded, i.e., apply linear activation.
- Initialize with small weights to not run into clipping issues from the start.
- Remember to run sufficient discriminator updates. This is crucial in the WGAN setup.
- You can use the Wasserstein surrogate loss implementation below.
- Clip discriminator weights by implementing your own keras constraint

```
def wasserstein_loss(y_true, y_pred):  
    return K.mean(y_true * y_pred)
```

```
class WeightClip(keras.constraints.Constraint):  
    def __init__(self, c):  
        self.c = c  
  
    def __call__(self, p):  
        return K.clip(p, -self.c, self.c)  
  
    def get_config(self):  
        return {'name': self.__class__.__name__, 'c': self.c}
```



GANs: Brief timeline - Spectral Normalization

Method to stabilize the training of the discriminator and restrict its capacity using a novel weight normalization technique:

$$\bar{W}_{SN} = W / \sigma(W)$$

Singular value is
very similar to
PCA

where $\sigma(W)$ is equivalent to the largest singular value of W .



GANs: Brief timeline – Self Attention



[\[Zhang et al. 2018\]](#)

GANs: Brief timeline – Large Scale GAN

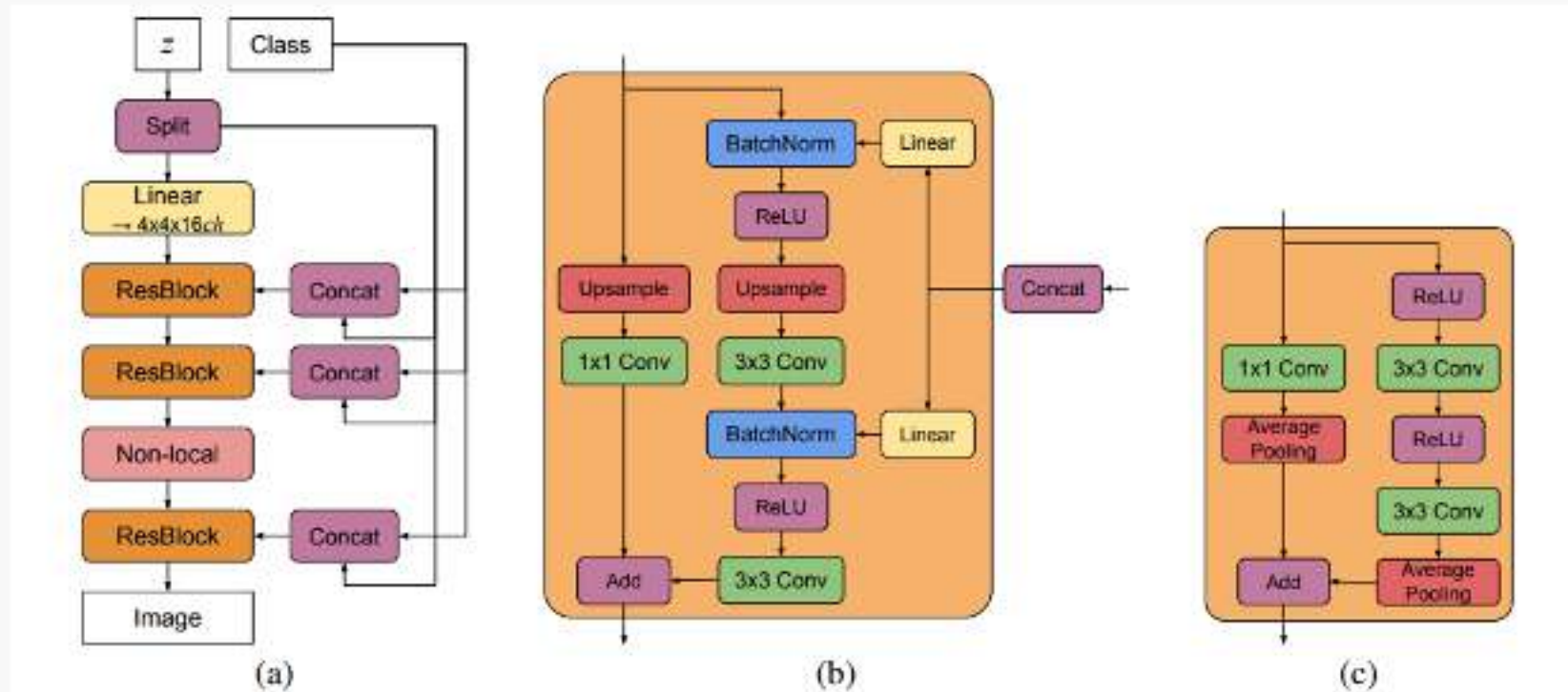


Figure 15: (a) A typical architectural layout for BigGAN's **G**; details are in the following tables. (b) A Residual Block (*ResBlock up*) in BigGAN's **G**. (c) A Residual Block (*ResBlock down*) in BigGAN's **D**.

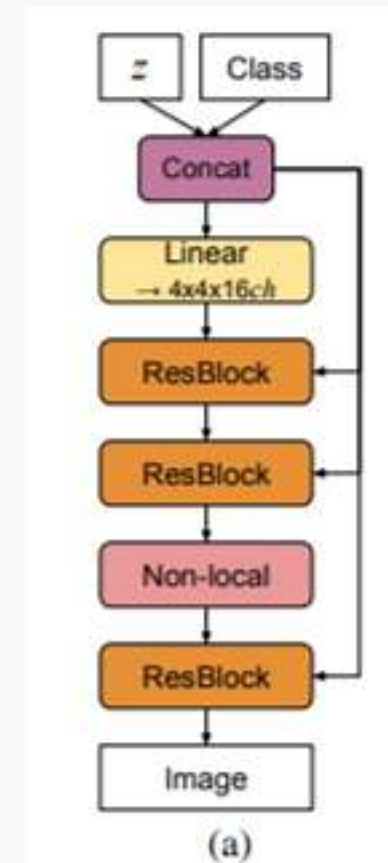


Figure 16: (a) A typical architectural layout for BigGAN-deep's **G**

GANs: Brief timeline – Large Scale GAN



2017: The explosion of GANs

The GAN Zoo

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorical GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- icGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks
- DEQGAN - [Differential Equation GAN](#)
- TCGAN - [Time Conditional GAN](#)

Normalize the inputs

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

Use Spherical Z

Don't sample from a Uniform 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](https://github.com/dribnet/plat) ref code <https://github.com/dribnet/plat> has more details



GAN Rules of Thumb (GANHACKs)

Batch Normalization

- 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).

Avoid Sparse Gradients: ReLU, MaxPool

- The stability of the GAN game suffers if you have sparse gradients
- LeakyReLU = good (in both G and D)
- For Downsampling, use: Average Pooling, Conv2d + stride
- For Upsampling, use: ConvTranspose2d + stride



GAN Rules of Thumb (GANHACKs)

Use Soft and Noisy Labels

- Label Smoothing, i.e. if you have two target labels: Real=1 and Fake=0, then for each incoming sample, if it is real, then replace the label with a random number between 0.7 and 1.2, and if it is a fake sample, replace it with 0.0 and 0.3 (for example).
 - Salimans et. al. 2016
- Make the labels noisy for the discriminator: occasionally flip the labels when training the discriminator.

See GANHACKs (<https://github.com/soumith/ganhacks>) for more tips.



Exercise:

Simple exercise to calculate the Fréchet distance.

