AI504: Programming for Artificial Intelligence

Week 6: Generative Adversarial Networks

Edward Choi
Grad School of AI
edwardchoi@kaist.ac.kr

Today’s Topic

- Generative Adversarial Network
- VAE vs Autoregressive vs GAN
- Evaluation
- Applications of GAN

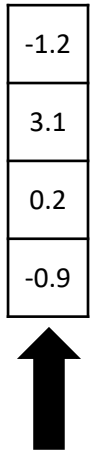
Generative Adversarial Network

VAE Recap

- Objective
  - Compress $\mathbf{x}$ to $\mathbf{z}$ which follows $P(\mathbf{Z} \mid \mathbf{X})$
  - Decompress $\mathbf{z}$ to reconstruct $\mathbf{x}$

Encoding (Compression)

$q_\theta(z \mid x_i)$



Decoding (Decompression)

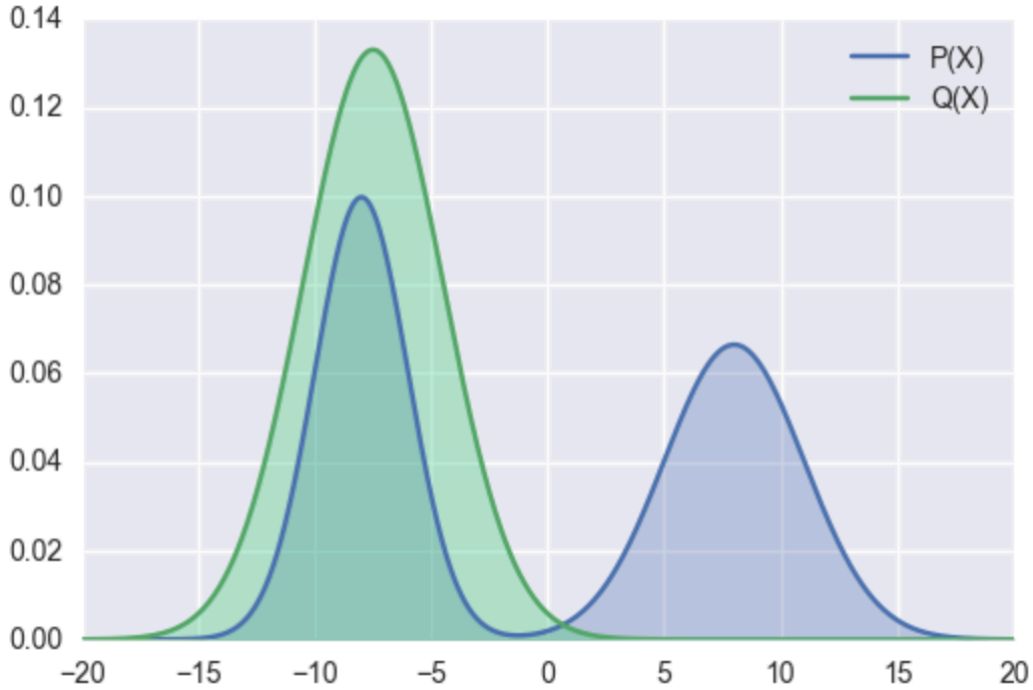
$p_\phi(x_i \mid z)$

This follows the distribution P (e.g. Gaussian $N(0, 1)$)

$$\mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$

VAE Recap

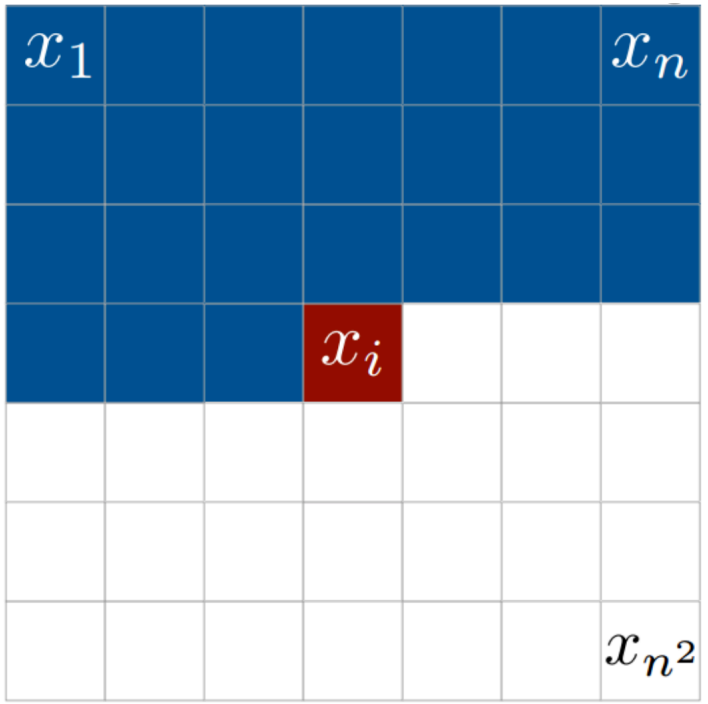
- This allows us to
  - Learn approximate posterior Q(Z|X) to mimic the true P(Z|X)
  - Sample $\mathbf{z}$ from $Q(\mathbf{Z})$, which can be converted to $\mathbf{x}$



https://wiseodd.github.io/techblog/2016/12/21/forward-reverse-kl/

Another Generative Model Family

- Pixel-CNN
  - Generate images one pixel at a time.
- WaveNet
  - Generate audio one frame at a time
- GPT-3
  - Generate text one word at a time



PixelCNN generates one pixel at a time

https://towardsdatascience.com/auto-regressive-generative-models-pixelrnn-pixelcnn-32d192911173

Another Generative Model

- Pixel-CNN
  - Generate images one pixel at a time.
- WaveNet
  - Generate audio one frame at a time
- GPT-3
  - Generate text one word at a time

Autoregressive Models

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

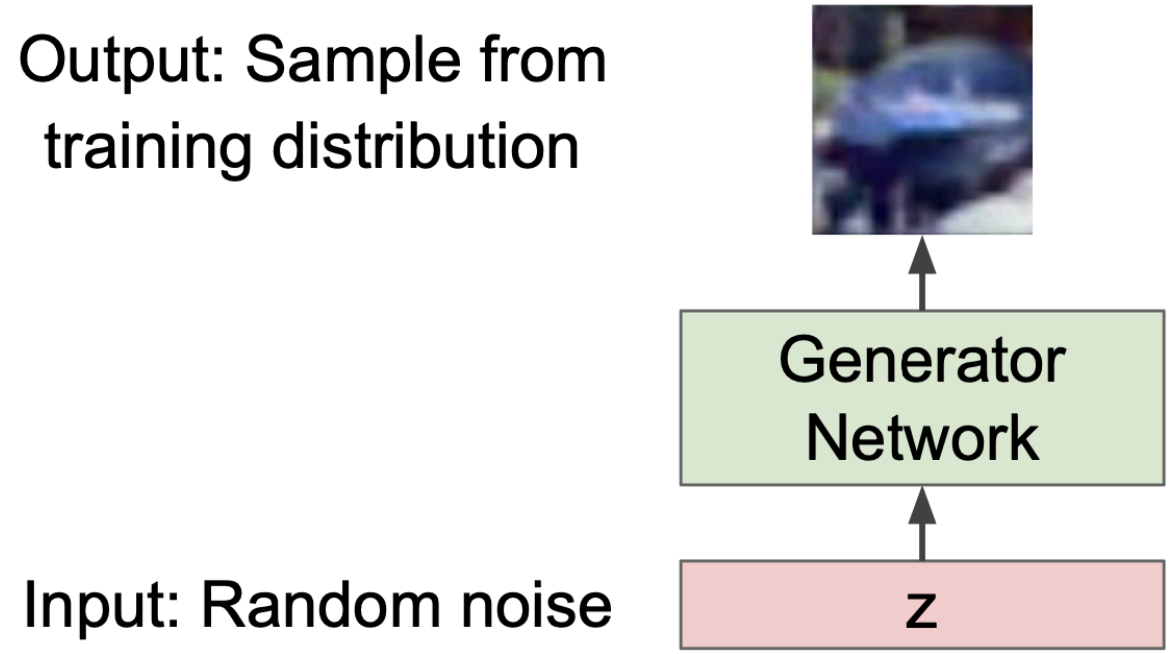
Works very well, but slow inference (sometimes impractically slow)

Yet Another Generative Model Family

- What if we don’t care about the data distribution?
- What if all we want is just good synthetic samples?

Yet Another Generative Model Family

- What if we don't care about the data distribution?
- What if all we want is just good synthetic samples?



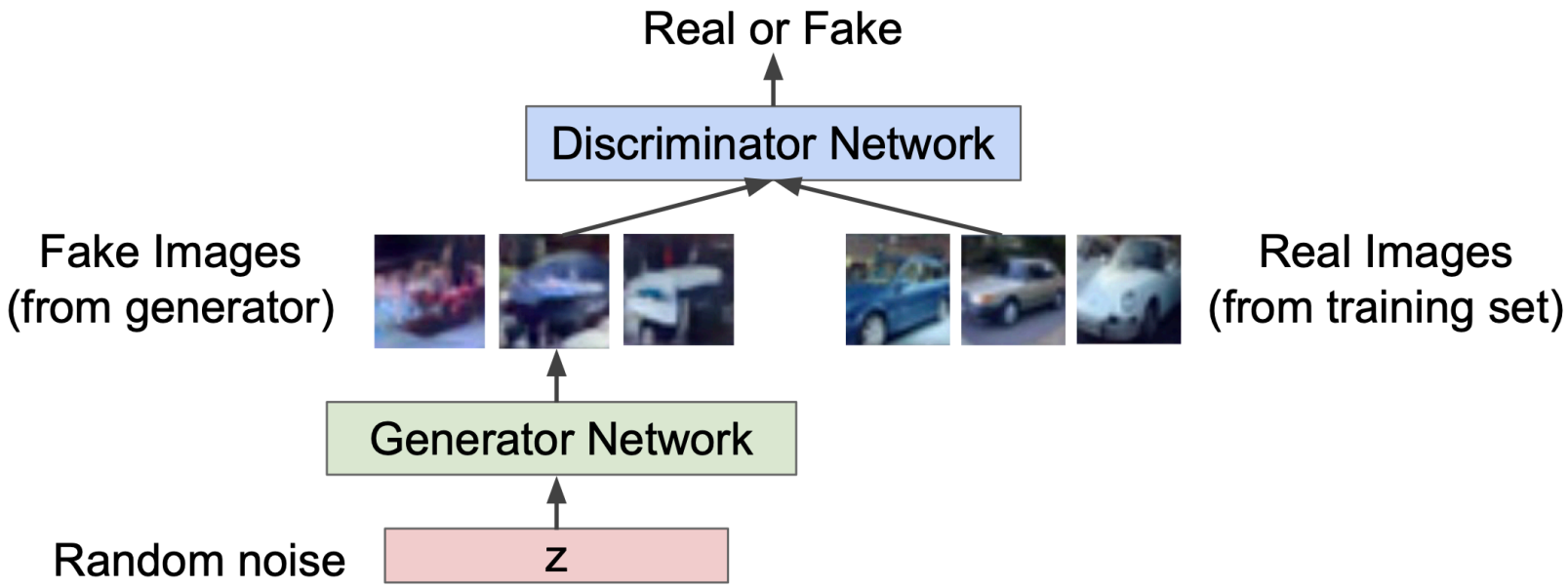
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Generative Adversarial Network

- Two person game
  - Game between Generator (G) and Discriminator (D)
  - Think of game theory
- Generator (G)
  - Tries to fool D with fake samples x’
- Discriminator (D)
  - Tries to discriminate between real samples x and fake samples x’

Generative Adversarial Network

- Generator (G)
  - Tries to fool D with fake samples x’
- Discriminator (D)
  - Tries to discriminate between real samples x and fake samples x’



Binary Classification

- GAN objective is basically binary classification
- Two different sub-goals
  - For G, fail the binary classification
  - For D, succeed the binary classification

Binary Classification

- GAN objective is basically binary classification
- Two different sub-goals
  - For G, fail the binary classification
  - For D, succeed the binary classification
- Remember Cross Entropy?

$$\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$$

Binary Classification

- Remember Cross Entropy?

$$\frac{1}{N}\sum_{n=1}^{N}\left[\underbrace{y_n \log \hat{y}_n}_{\text{Positive Samples}} + \underbrace{(1-y_n)\log(1-\hat{y}_n)}_{\text{Negative Samples}}\right]$$

$$= \frac{1}{N}\sum_{n=1}^{N/2}[\log \hat{y}_n] + \frac{1}{N}\sum_{n=1}^{N/2}[\log(1-\hat{y}_n)]$$

Separate positive samples and negative samples.
(Assuming the same number of positives & negatives)

$$= \frac{1}{2}\left[\frac{1}{N/2}\sum_{n=1}^{N/2}[\log \hat{y}_n] + \frac{1}{N/2}\sum_{n=1}^{N/2}[\log(1-\hat{y}_n)]\right]$$

$$= \frac{1}{2}\left[\mathbb{E}_{x\sim p_{positive}} \log D(x) + \mathbb{E}_{x\prime\sim p_{negative}} \log\left(1-D(x')\right)\right]$$

Sample mean is an unbiased estimator of population mean.

Binary Classification

- Remember Cross Entropy?

$$\mathbb{E}_{x\sim p_{positive}} \log D(x) + \mathbb{E}_{x\prime\sim p_{negative}} \log\left(1 - D(x')\right)$$ Lose the constant 1/2, since our goal is optimization.

$$= \mathbb{E}_{x\sim p_{real}} \log D(x) + \mathbb{E}_{x\prime\sim p_{fake}} \log\left(1 - D(x')\right)$$ Positive VS Negative ➔ Real VS Fake

$$= \mathbb{E}_{x\sim p_{real}} \log D(x) + \mathbb{E}_{z\sim p_z} \log\left(1 - D\left(G(z)\right)\right)$$ Fake samples come from random noise p(z) and Generator G.

$$= \mathbb{E}_{x\sim p_{real}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p_z} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)$$ Use parameterized functions for D and G.

Two Different Goals

- For D, succeed the binary classification

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{real}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p_z} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

- For G, fail the binary classification

$$\min_{\theta_g} \left[\mathbb{E}_{x \sim p_{real}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p_z} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

$$= \min_{\theta_g} \left[\mathbb{E}_{z \sim p_z} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

First term is not concerned with θ_g.

MinMax Game

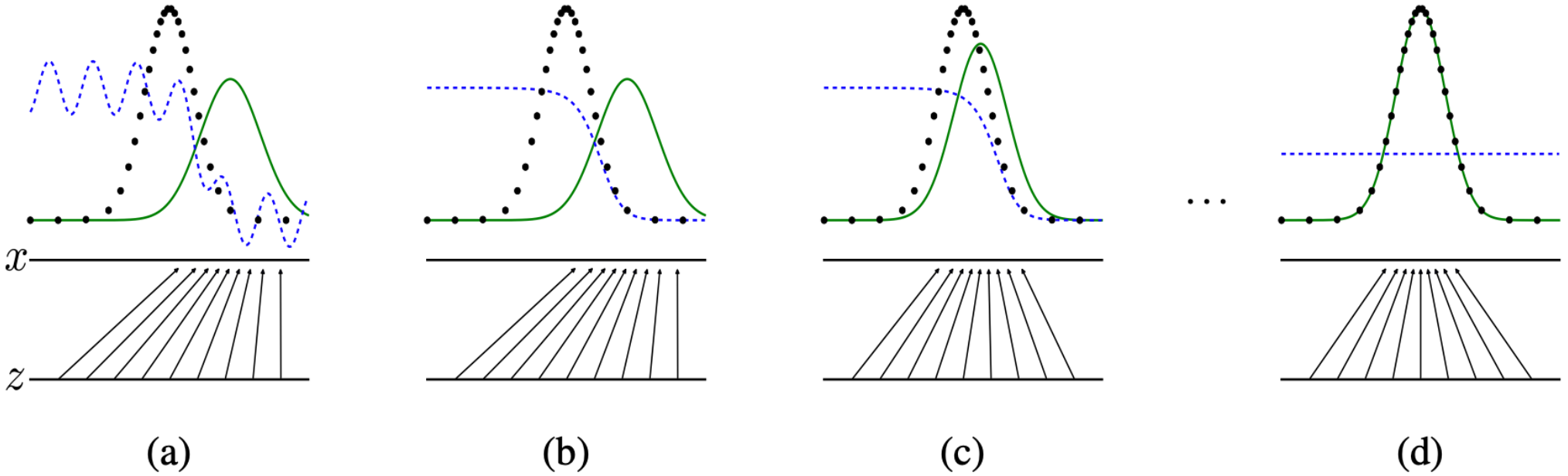
- GAN is a two-person MinMax game between D & G

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{real}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p_z} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Theoretically…

- When D & G reaches Nash Equilibrium,
  - D's accuracy is 50% (random guess)
  - G(z)'s distribution is the same as p(x)



Modified MinMax Game

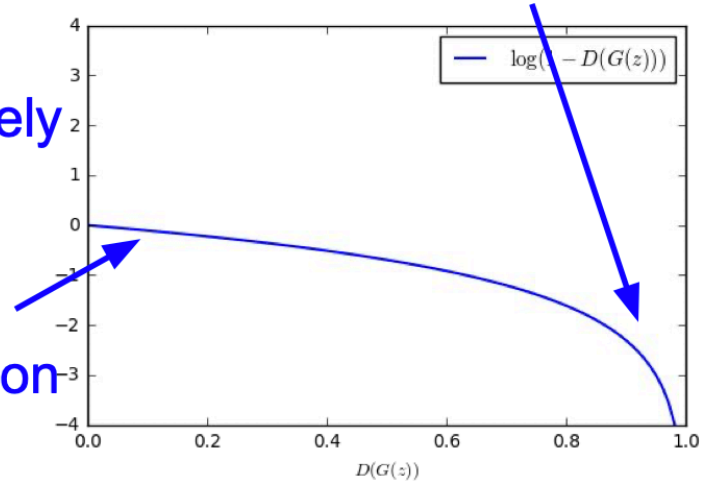
- In practice, G doesn't learn very well.
- G trying to fool D:

$$\min_{\theta_g}\left[\mathbb{E}_{z\sim p_z}\log\left(1-D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]$$

  - In the initial phase, D easily overpowers G
    - D(G(z)) will almost always be 0
  - The gradient of the above objective becomes zero

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



Modified MinMax Game

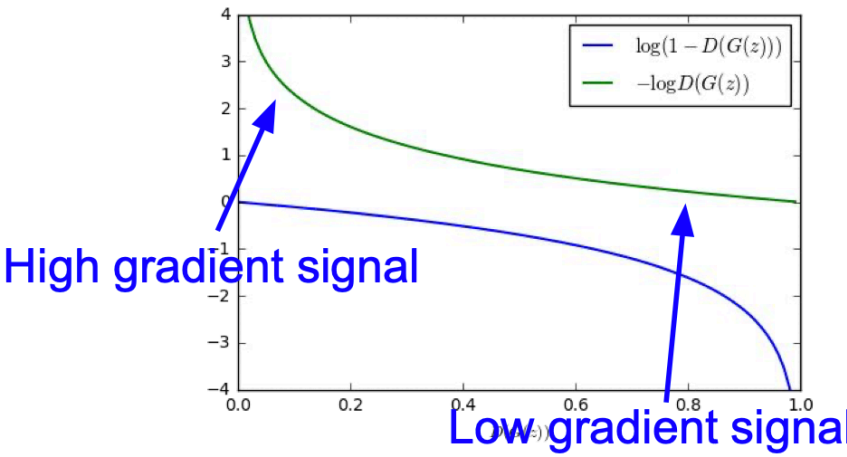
- Modified G objective:

$$\max_{\theta_g} \left[\mathbb{E}_{z \sim p_z} \log \left(D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.

But no longer MinMax game

➔ No Nash Equilibrium, but probably doesn't matter



Training GAN

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training GAN

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

What's the best value for *k*?

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

When to Stop Training

- There is no golden rule
  - Because D and G oscillates, we never reach Nash Equilibrium.
    - We use the modified G loss, so no Nash Equilibrium anyway.
- Typical way
  - Generate fake samples from time to time.
  - If there is no visual improvement, stop training.
    - You can use evaluation metric (FID score) as well.
- Use GAN variants
  - Ex) Wasserstein GAN provides a more indicative G loss.

Generating Fake Samples

- How do we generate fake samples?

Mode Collapse

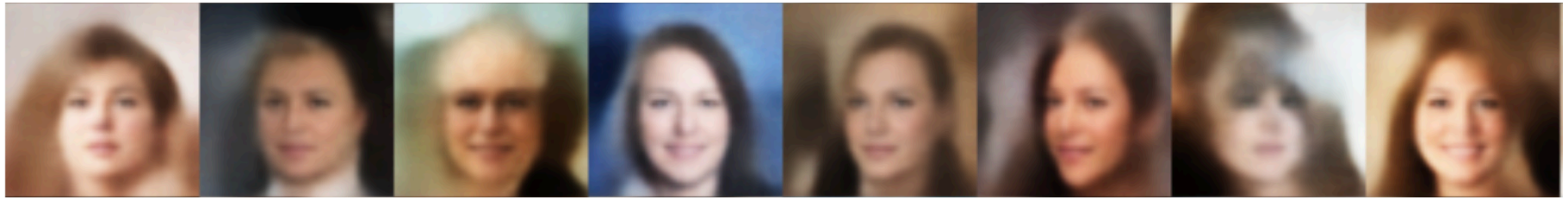
- What if G generates one realistic fake sample?
  - As long as G fools D…
- Mode Collapse
  - When G generates samples of limited diversity (often only a few)
  - (This means D is trapped in a bad local minimum)
- Remedy
  - Tune hyperparameters
  - Use GAN variants (Wasserstein GAN, Unrolled GAN)

VAE vs Autoregressive vs GAN

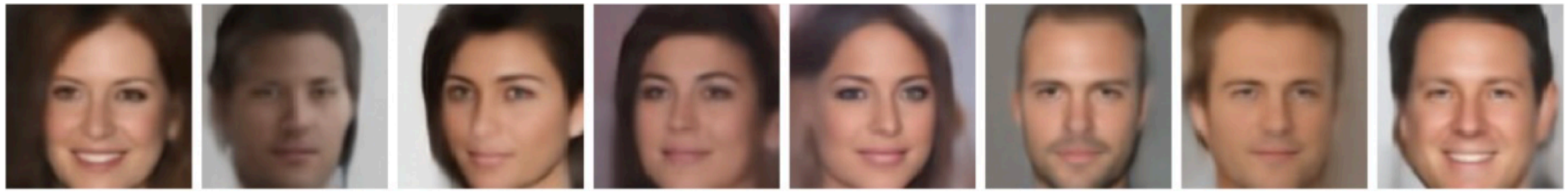
Generated Samples

- VAE vs PixelCNN vs PixelCNN

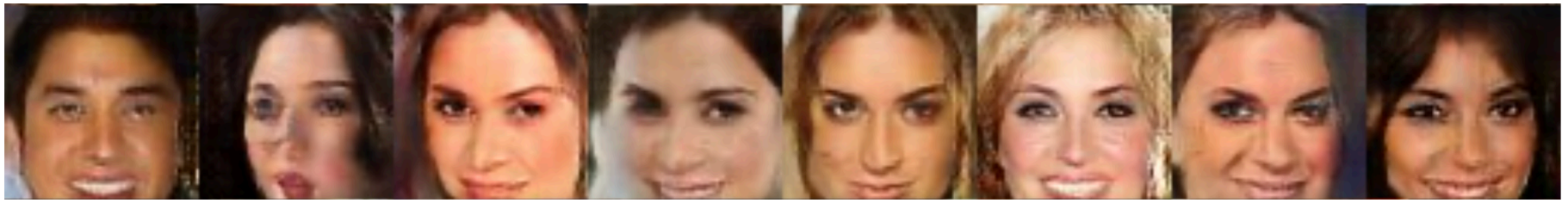
VAE



PixelCNN



GAN



VAE

- Generation
  - Sample from N(0, 1), then use decoder.
- Measuring performance
  - Can use reconstruction loss between x and x’.
- Image quality
  - Vanilla VAE generates blurry images
    - Due to Mean Squared Error loss
  - Modern VAEs generate better images

PixelCNN

- Generation
  - Given some initial pixels, generate rest of the pixels.
  - (Relatively) slow generation process
- Measuring performance
  - Can use the likelihood of the sample $p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i|x_1, ..., x_{i-1})$
- Image quality
  - Better than vanilla VAE
  - Modified autoregressive models exist to speed up the generation process.

GAN

- Generation
  - Sample from Uniform or Gaussian, then use Generator.
- Measuring performance
  - ???
- Image quality
  - Sharpest quality
    - StyleGAN generates 1024-by-1024 images successfully
  - Mode collapse lurks

Evaluation

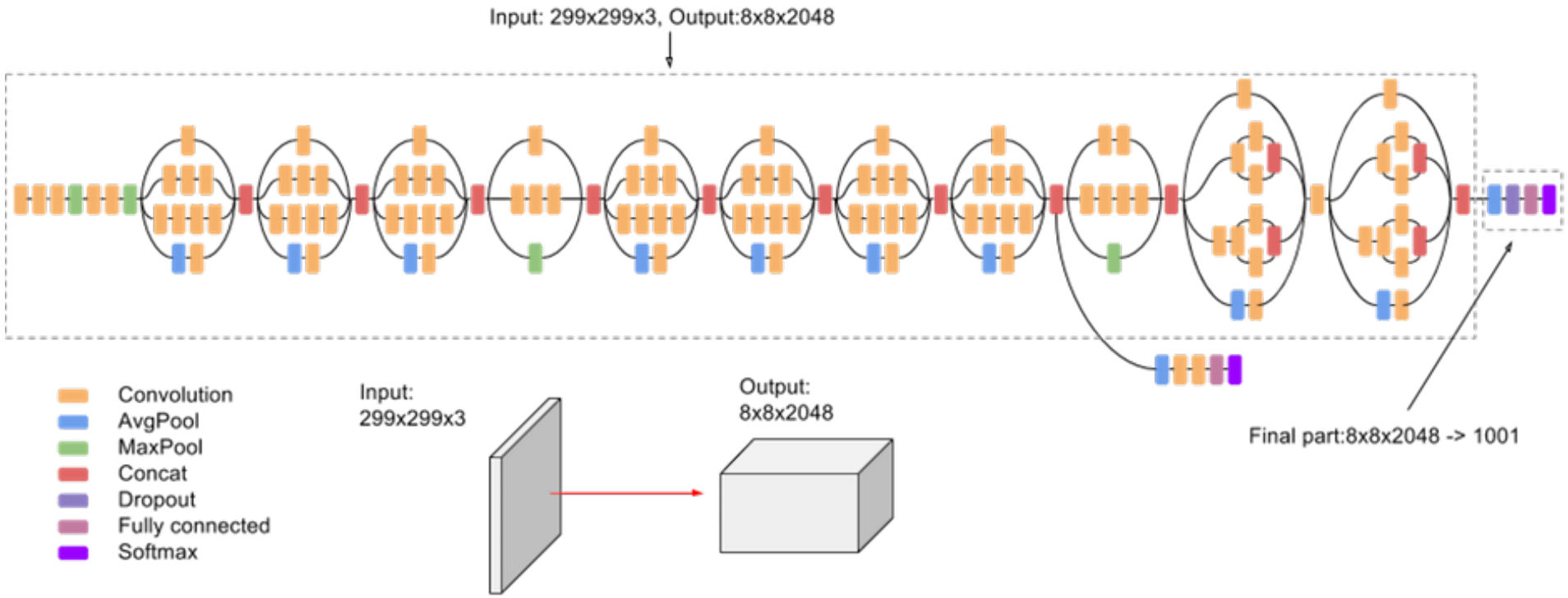
Evaluating GAN

- GAN uses an “implicit” distribution
  - Unlike VAE or PixelCNN
- Difficult to measure the quality of fake samples.

- Two popular metrics
  - Inception Score
  - Frechet Inception Distance (FID) Score

Inception Score

- Use a pre-trained model to evaluate fake images
  - Inception v3 model trained on ImageNet (1000 classes)



Inception Score

- Use a pre-trained model to evaluate fake images
  - Inception v3 model trained on ImageNet (1000 classes)
- Two criteria
  - Image quality
    - Do images look like a specific object (i.e. class)?
  - Image diversity
    - Is a wide range of objects generated?

https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/

Two Criteria

- Image quality
  - A single image should have a focused p(y|x)
    - A specific class should be preferred over other classes
  - p(y|x) of low entropy ➔ High image quality
- Image diversity
  - Marginal probability p(y) over all images should not focus on one class.
  - p(y) of high entropy ➔ Diverse images
- We want the relative entropy of p(y|x) and p(y) to be large
  - Large KL(p(y|x) || p(y))
- For specific algorithm, see appendix

Inception Score

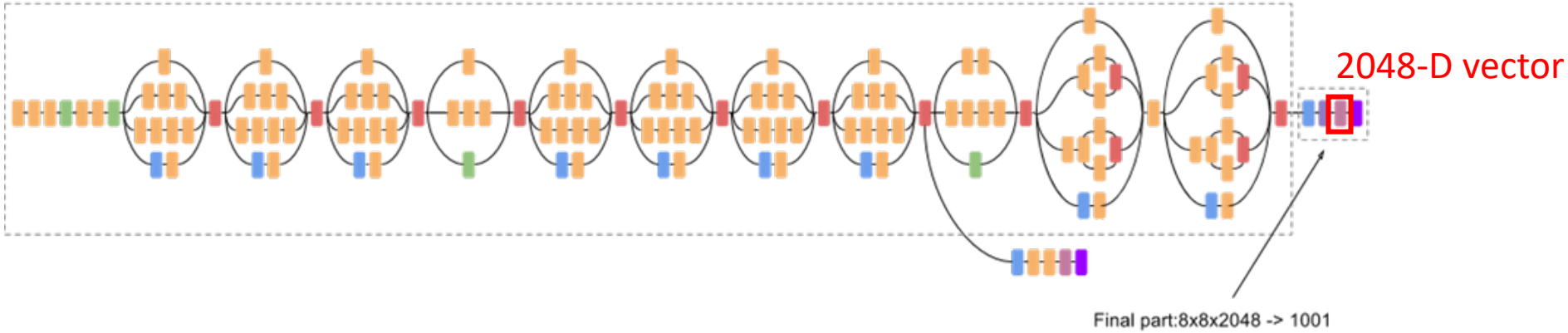
- Worst score: 1.0
- Best score: N
  - N being the number of output classes of the pre-trained model.
  - Inception v3: 1000 classes

[1, 0, 0], [0, 1, 0], [0, 0, 1] ➔ IS 3.0

[0.33, 0.33, 0.33], [0.33, 0.33, 0.33], [0.33, 0.33, 0.33] ➔ IS 1.0

Frechet Inception Distance (FID) Score

- Also uses Inception v3
  - Use the last pooling layer (i.e. image feature) instead of output probabilities



- Compare the distributions of the features between real and fake.
  - View the image features as Gaussian
  - Calculate the distance between real/fake Gaussian distributions
    - Using Frechet distance (also called Wasserstein-2 distance)

FID Score

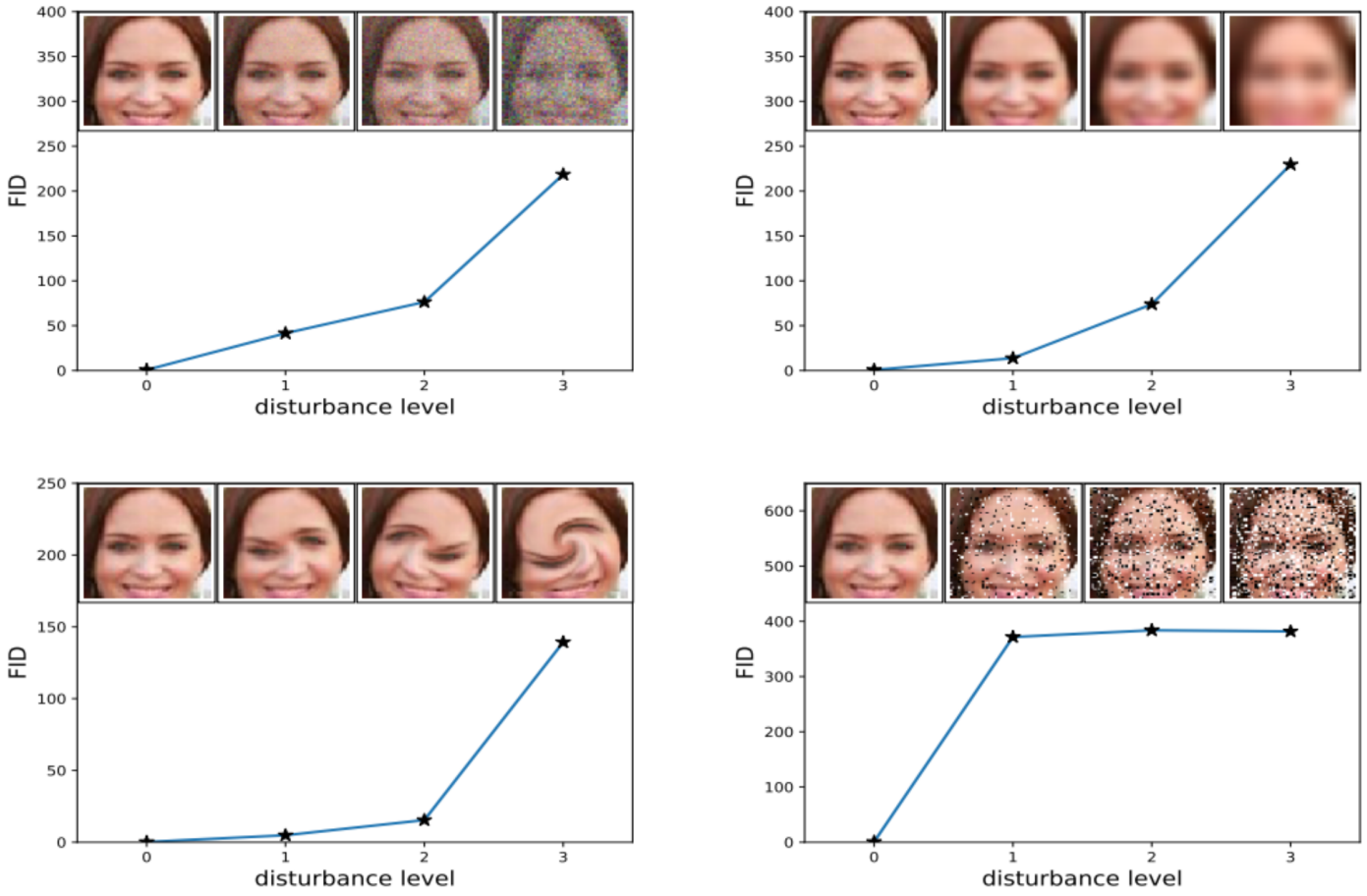
- Calculate the feature-wise (i.e. dimension-wise) statistics
  - Mean & Covariance for real/fake features
- Calculate Frechet distance between two Gaussians

$$d^2((\boldsymbol{m}, \boldsymbol{C}), (\boldsymbol{m}_w, \boldsymbol{C}_w)) = \|\boldsymbol{m} - \boldsymbol{m}_w\|_2^2 + \mathrm{Tr}\big(\boldsymbol{C} + \boldsymbol{C}_w - 2\big(\boldsymbol{C}\boldsymbol{C}_w\big)^{1/2}\big)$$

- For specific algorithm, see appendix

FID Score Example

- Lower FID ➔ Better sample quality



Applications of GAN

Text-to-Image GAN

- Reed et al. 2016
- Conditional image generation
- Inject text encoding
  - In the Generator
  - In the Discriminator

this small bird has a pink breast and crown, and black primaries and secondaries.

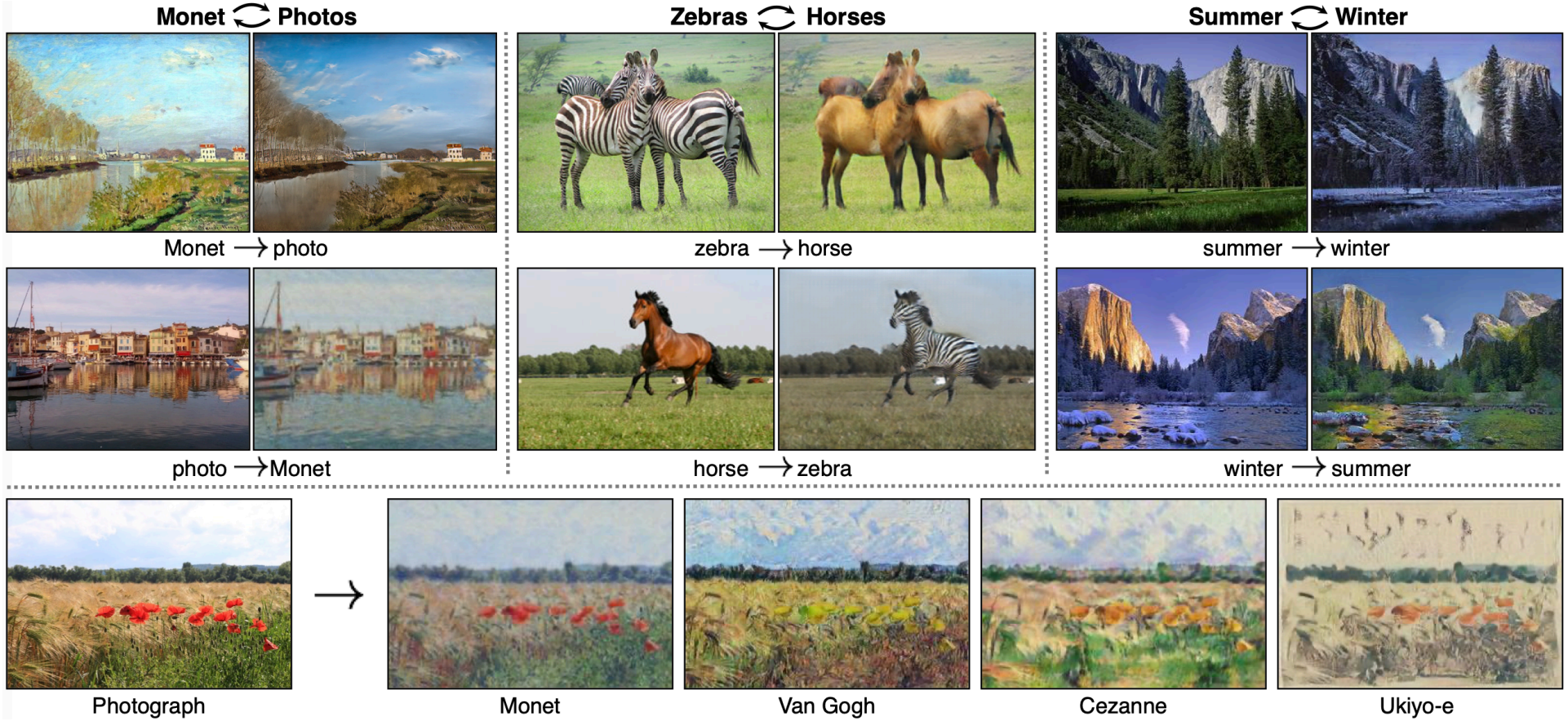
this magnificent fellow is almost all black with a red crest, and white cheek patch.

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

Cycle GAN

- Zhu et al. 2017
- Transfer style between two domains
  - Adversarial loss
  - Cycle consistency



Style GAN

- Karras et al. 2019
- High resolution samples
- Using various techniques
  - Put z through MLP
  - AdaIN
    - Evolved from BatchNorm
  - Bilinear upsampling
  - Layer-wise noise

AI504: Programming for Artificial Intelligence

Week 6: Generative Adversarial Networks

Edward Choi
Grad School of AI
edwardchoi@kaist.ac.kr

Inception Score Algorithm

```
# calculate the inception score for p(y|x)
def calculate_inception_score(p_yx, eps=1E-16):
    # calculate p(y)
    p_y = expand_dims(p_yx.mean(axis=0), 0)
    # kl divergence for each image
    kl_d = p_yx * (log(p_yx + eps) - log(p_y + eps))
    # sum over classes
    sum_kl_d = kl_d.sum(axis=1)
    # average over images
    avg_kl_d = mean(sum_kl_d)
    # undo the logs
    is_score = exp(avg_kl_d)
    return is_score
```

https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/

FID Score Algorithm

```python
# calculate frechet inception distance
def calculate_fid(act1, act2):
    # calculate mean and covariance statistics
    mu1, sigma1 = act1.mean(axis=0), cov(act1, rowvar=False)
    mu2, sigma2 = act2.mean(axis=0), cov(act2, rowvar=False)
    # calculate sum squared difference between means
    ssdiff = numpy.sum((mu1 - mu2)**2.0)
    # calculate sqrt of product between cov
    covmean = sqrtm(sigma1.dot(sigma2))
    # check and correct imaginary numbers from sqrt
    if iscomplexobj(covmean):
        covmean = covmean.real
    # calculate score
    fid = ssdiff + trace(sigma1 + sigma2 - 2.0 * covmean)
    return fid
```

https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/