Deep Learning and Applications

DSA 5204 • Lecture 10

Dr Low Yi Rui (Aaron)
Department of Mathematics

NUS
National University of Singapore

Last Time

We introduced an important class of generative models known as *variational autoencoders*

Important ideas:

- Latent variables
- Reparameterization trick

$$\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}) \;\rightarrow\; \boldsymbol{x} = g(\boldsymbol{u}, \boldsymbol{\theta}), \boldsymbol{u} \sim p_0(\boldsymbol{u})$$

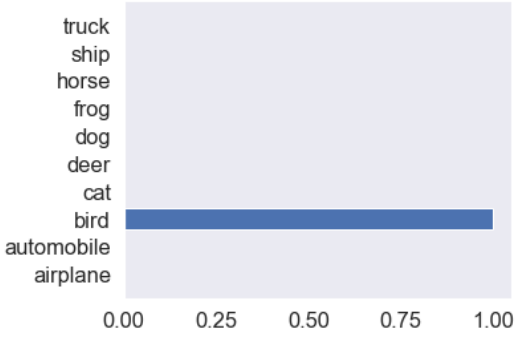
Today, we will discuss some other generative models that also shares some of these ideas

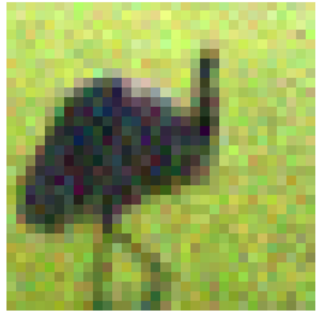
Generative Adversarial Networks

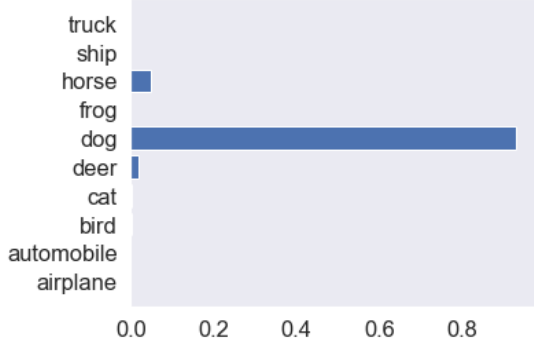
Review: Adversarial Training (Lecture 7)

Given a learned model $\hat{f}(x)$, we can fool the model by finding an *adversarial example*

$$x' = \underset{z, \|z-x\| \leq \delta}{\text{argmax}} \; L\left(\hat{f}(z), y\right)$$







Review: Adversarial Training (Lecture 7)

To ameliorate this problem, adversarial training is proposed, in the form of a min-max optimization problem

Adversarial Training via FGSM (Fast Gradient Sign Method)

Hyper-parameters: δ (adv size), J (#adv train) ϵ_1, ϵ_2 (learning rates)

For $k = 0,1, \ldots$ do

$\boldsymbol{z_0} = \boldsymbol{x}$

For $j = 0,1, \ldots, J-1$ do

$$\boldsymbol{z'} = \boldsymbol{z}_j + \epsilon_2 \text{Sign}\left(\nabla_{\boldsymbol{z}} L\left(\hat{y}\left(\boldsymbol{z_j}; \boldsymbol{\theta}_k\right), y\right)\right)$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon_1 \nabla_{\boldsymbol{\theta}} L\left(\hat{y}\left(\boldsymbol{z}_J; \boldsymbol{\theta}_k\right), y\right)$$

The Role of Adversaries

The key idea of adversarial training is game-theoretic:

Use the adversary to challenge the training, so that if training wins, then we can beat adversaries.

Adversarial Training via FGSM (Fast Gradient Sign Method)

Hyper-parameters: δ (adv size), J (#adv train) ϵ_1, ϵ_2 (learning rates)

For $k = 0,1, \ldots$ do

$\boldsymbol{z_0} = \boldsymbol{x}$

For $j = 0,1, \ldots, J-1$ do

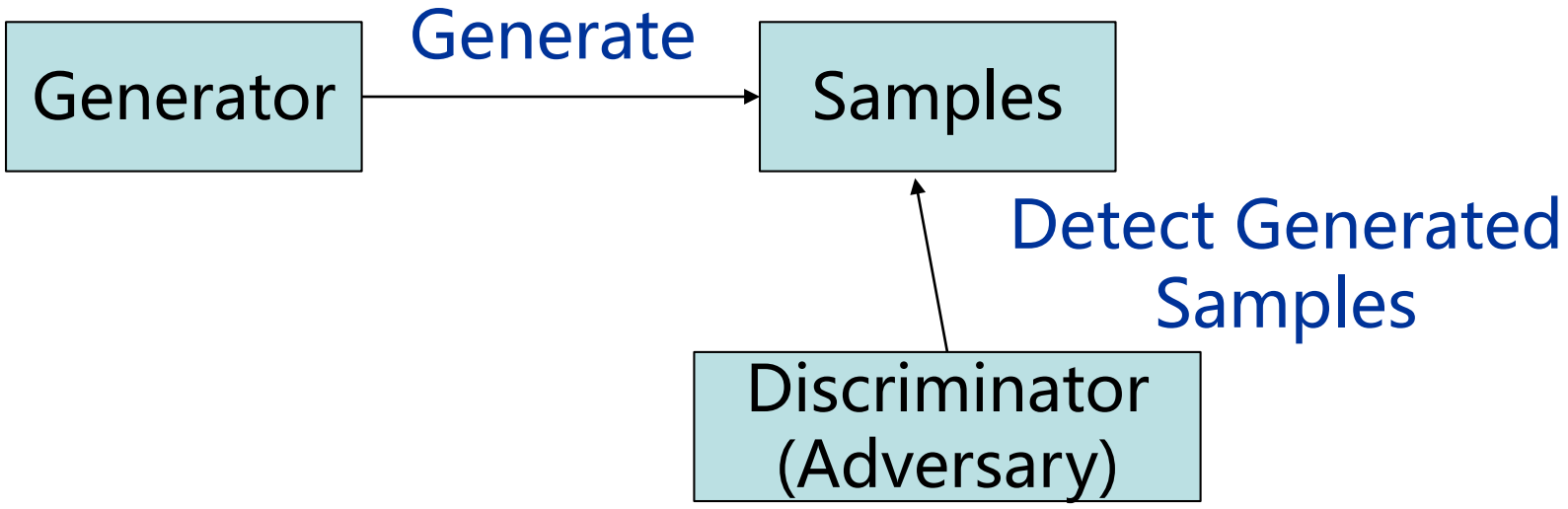
$$\boldsymbol{z'} = \boldsymbol{z_j} + \epsilon_2 \text{Sign}\left(\nabla_{\boldsymbol{z}} L\left(\hat{y}\left(\boldsymbol{z_j}; \boldsymbol{\theta}_k\right), y\right)\right)$$

Adversary

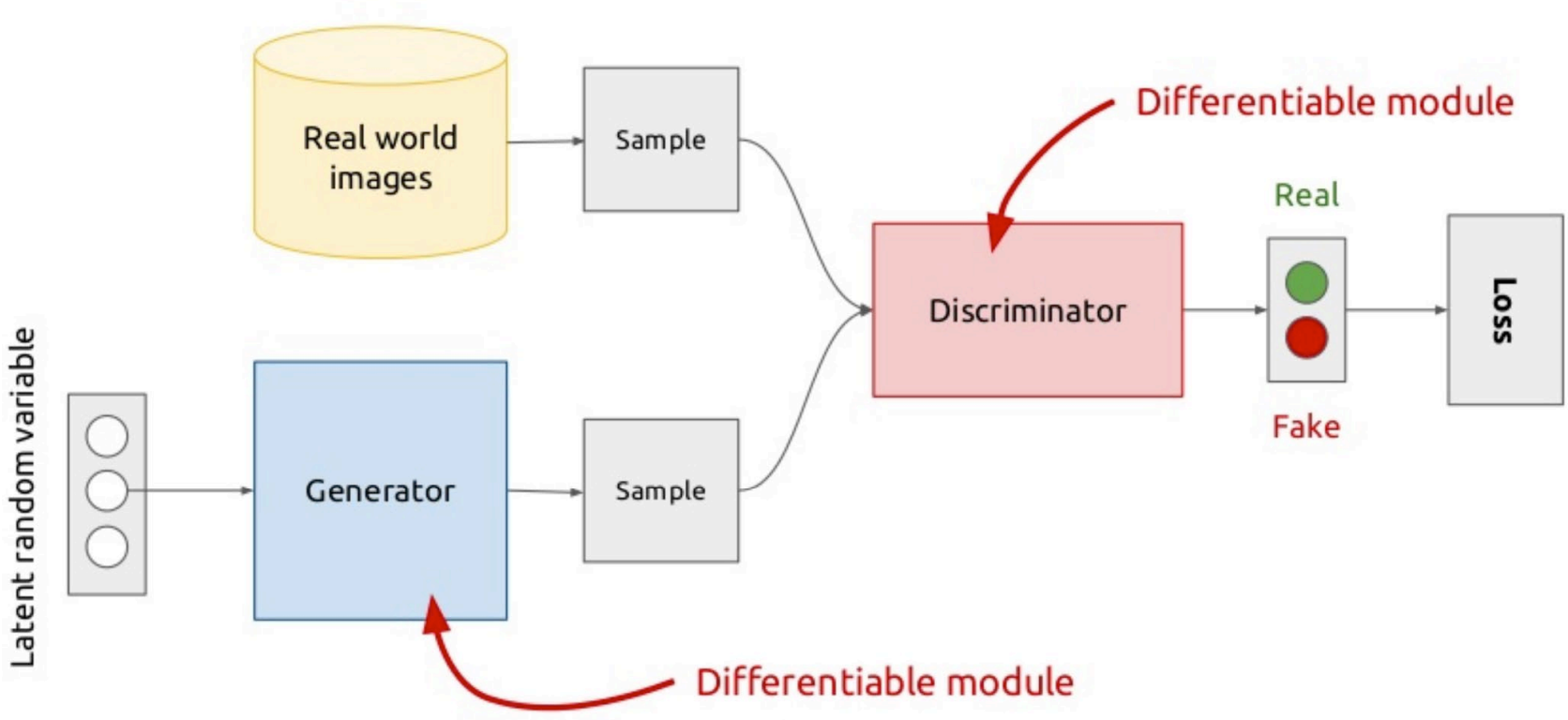
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon_1 \nabla_{\boldsymbol{\theta}} L\left(\hat{y}\left(\boldsymbol{z}_J; \boldsymbol{\theta}_k\right), y\right)$$

Adversaries for Generative Models

We make use of the previous idea of adversaries to generate outputs. This is known as *Generative Adversarial Networks (GAN)*

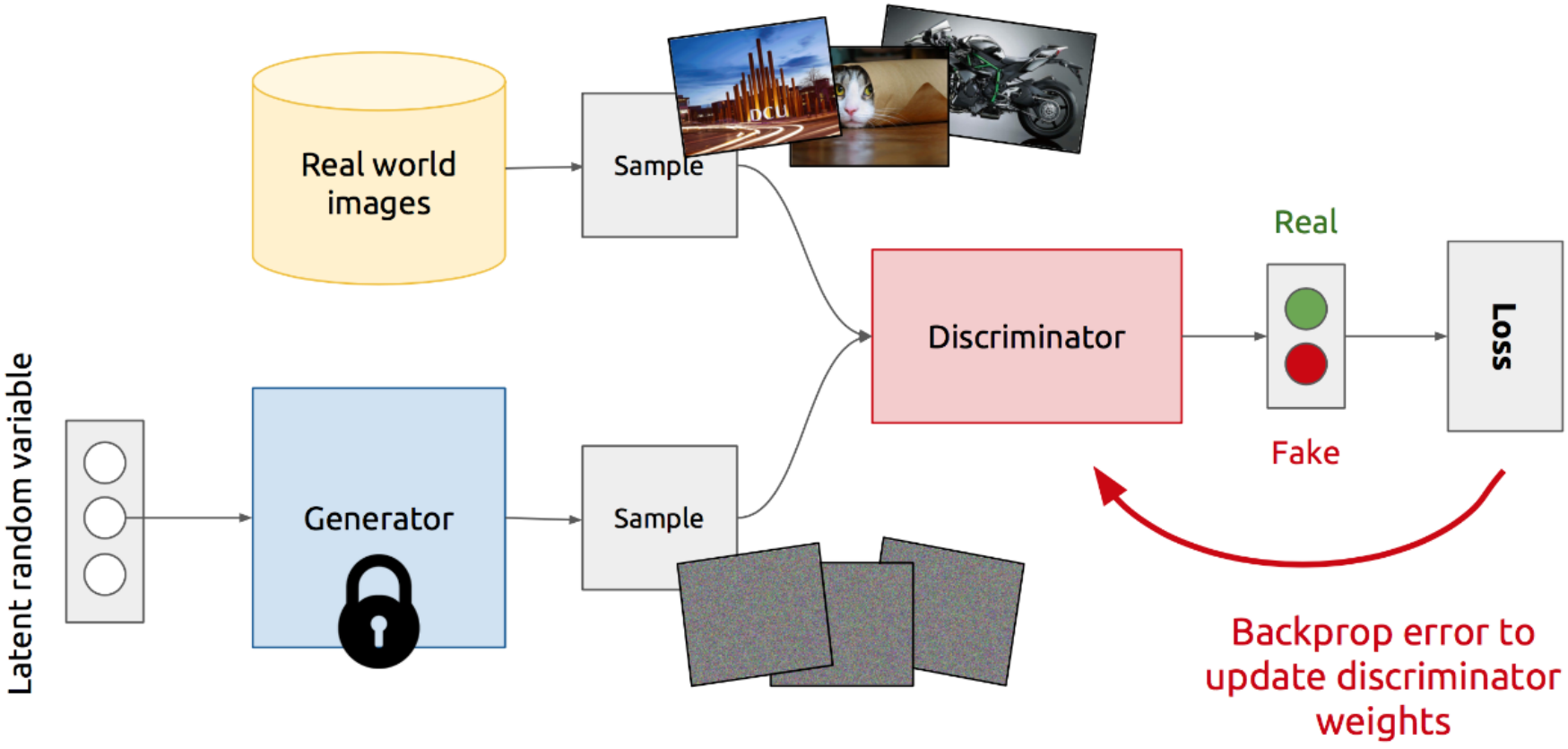


Detailed Architecture of GANs



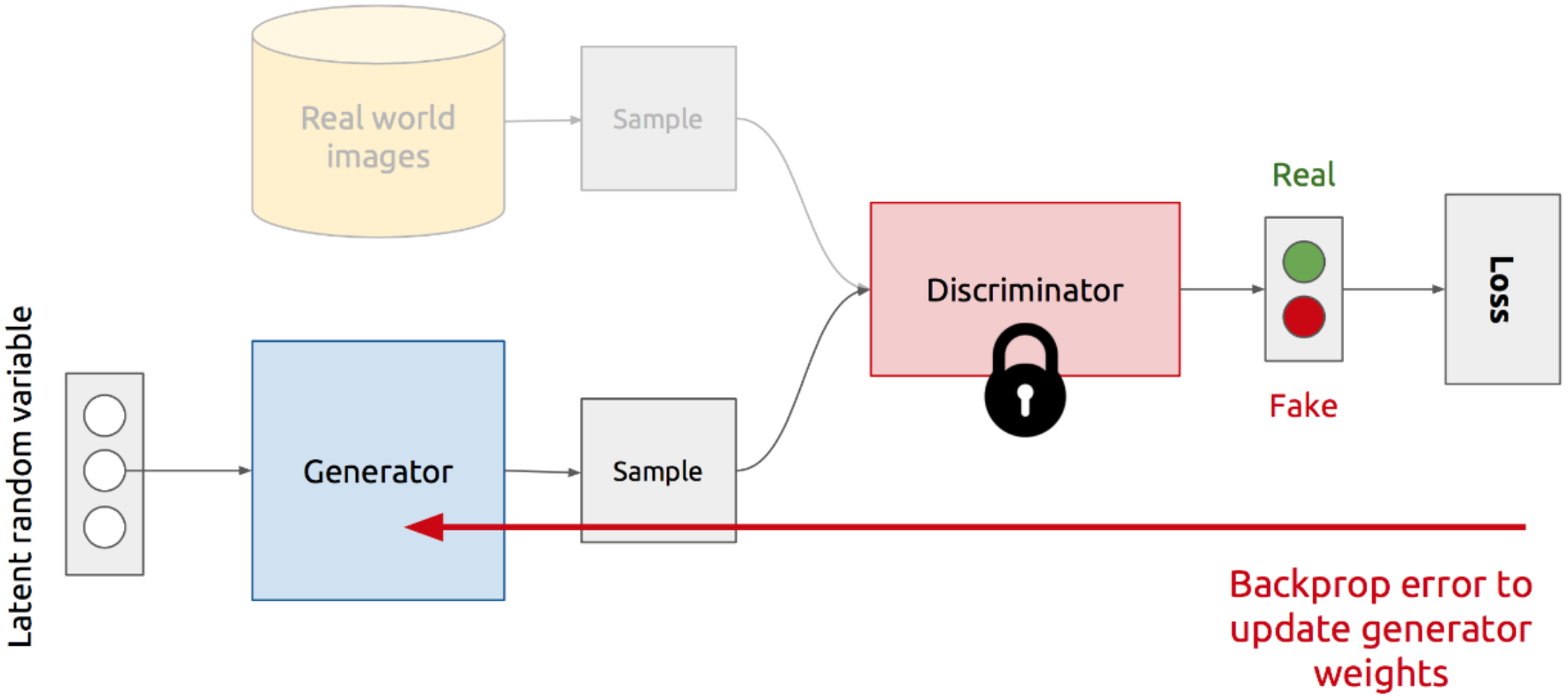
https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Discriminator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Generator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Mathematical Formulation of The Training Problem

Latent variable (n-dimensional)

$$\boldsymbol{z} \sim p_0 = \mathcal{N}(0, I_n)$$

Training data

$$\left\{\boldsymbol{x}^{(i)} \in \mathbb{R}^d : i = 1, \dots, N, \boldsymbol{x}^{(i)} \sim p^*\right\}$$

Generator:

$$G_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^d$$

Discriminator:

$$D_{\boldsymbol{\phi}} : \mathbb{R}^d \to [0,1]$$

where $D_{\boldsymbol{\phi}}(x)$ **is the predicted probability of x being a real sample.**

Mathematical Formulation of The Training Problem

Discriminator's cost function is to minimize the cross-entropy loss for real vs generated samples

Recall, given true binary labels $y \in \{0,1\}$, the (binary) cross-entropy loss is

$$-y \log D_{\boldsymbol{\phi}}(\boldsymbol{x}) - (1-y) \log\left(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x})\right)$$

- **If $y = 1$ (real sample), then the average loss is**

$$\mathbb{E}_{\boldsymbol{x} \sim p^*}\left[-\log D_{\boldsymbol{\phi}}(\boldsymbol{x})\right]$$

- **If $y = 0$ (generated sample), then the average loss is**

$$\mathbb{E}_{\boldsymbol{z} \sim p_0}\left[-\log\left(1 - D_{\boldsymbol{\phi}}\left(G_{\boldsymbol{\theta}}(\boldsymbol{z})\right)\right)\right]$$

Hence, the total (negative) loss the discriminator should maximize is

$$L(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{x} \sim p^*}\left[\log D_{\boldsymbol{\phi}}(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z} \sim p_0}\left[\log\left(1 - D_{\boldsymbol{\phi}}\left(G_{\boldsymbol{\theta}}(\boldsymbol{z})\right)\right)\right]$$

Min-Max Problem

Just like adversarial training, the adversary tries to maximize the loss, and the training then minimizes the maximized loss:

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} L(\boldsymbol{\theta}, \boldsymbol{\phi})$$

Writing this out (and negating), we have

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{x}\sim p^*}\left[\log D_{\boldsymbol{\phi}}(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim p_0}\left[\log\left(1 - D_{\boldsymbol{\phi}}\left(G_{\boldsymbol{\theta}}(\boldsymbol{z})\right)\right)\right]$$

This is called a

- *Min-max* (or minimax) formulation
- *Saddle-point* formulation
- *Zero-sum game* between $D_{\boldsymbol{\phi}}$ and $G_{\boldsymbol{\theta}}$

Relationship with Game Theory

When interpreted as a two-player zero-sum game, we can ask where the *Nash equilibrium* is (or are).

A Nash equilibrium is where neither players have incentive to change its strategy.

A Nash equilibrium

- The distribution of $G_{\boldsymbol{\theta}}(\boldsymbol{z})$ equals p^*
- $D_{\boldsymbol{\phi}}(\boldsymbol{x}) = \frac{1}{2}$ for all $\boldsymbol{x}$

Relationship with KL Divergence
<Lecture Notebook>

Let us forget about the explicit parameterizations as write

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p^*}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_0}\left[\log(1 - D\left(G(\boldsymbol{z})\right)\right]$$

We can then show:

- Denote by p_G the distribution of $G(\boldsymbol{z})$. Then, for *fixed* G, the optimal D is

$$D_G^*(\boldsymbol{x}) = \frac{p^*(\boldsymbol{x})}{p^*(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$

- Thus, the minimax problem can be framed as

$$\min_G D_{KL}\left(p^* \| \frac{p^* + p_G}{2}\right) + D_{KL}\left(p_G \| \frac{p^* + p_G}{2}\right) - 2\log 2$$

KL Divergence and JS Divergence

Recall the KL divergence

$$D_{KL}(p\|q) = \int p(\boldsymbol{x})\log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)\boldsymbol{dx}$$

The Jensen-Shannon (JS) divergence is defined as

$$D_{JS}(p\|q) = \frac{1}{2}D_{KL}\left(p\|\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q\|\frac{p+q}{2}\right)$$

Properties

- Like KL, JS divergence is non-negative and equals 0 if and only if $p = q$
- Unlike KL, JS divergence is *symmetric*
- In fact, $\sqrt{D_{JS}}$ is a metric on the space of distributions

Training GAN from the Divergence Point of View

The GAN training problem, interpreted from the divergence point of view, can be stated as

$$\min_G D_{JS}(p_G \| p^*) - 2\log 2$$

That is, ***we are minimizing the JS divergence (think "distance") between the generation distribution*** p_G ***and the true data distribution*** p^***.**

The Basic Training Algorithm for GAN

Repeat:

Train Discriminator

- For k steps do
  - Sample m noise samples $\{\boldsymbol{z}^{(1)}, \ldots \boldsymbol{z}^{(m)}\} \sim p_0$
  - Sample m real data samples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\} \in \mathcal{D}$
  - Update

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \epsilon_1 \nabla_{\boldsymbol{\phi}} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D_{\boldsymbol{\phi}}\left(G_{\boldsymbol{\theta}}\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$$

- Sample m noise samples $\{\boldsymbol{z}^{(1)}, \ldots \boldsymbol{z}^{(m)}\} \sim p_0$ and update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_2 \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D_{\boldsymbol{\phi}}\left(G_{\boldsymbol{\theta}}\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

Train Generator

Problem I: Vanishing Gradients

Loss function

$$L(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{x}\sim p^*}\left[\log D_{\boldsymbol{\phi}}(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim p_0}\left[\log\left(1 - D_{\boldsymbol{\phi}}\left(G_{\boldsymbol{\theta}}(\boldsymbol{z})\right)\right)\right]$$

The gradient with respect to θ controls the generator updates

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\phi}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{z}\sim p_0}\left[\log\left(1 - D_{\boldsymbol{\phi}}\left(G_{\boldsymbol{\theta}}(\boldsymbol{z})\right)\right)\right]$$

Observe if $\sigma = \mathrm{Sigmoid}$, then

- Since $\nabla_a \log\left(1 - \sigma(a)\right) = -\sigma(a)$
- Then, $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\phi}) \propto \mathbb{E}_{\boldsymbol{z}\sim p_0} D_{\boldsymbol{\phi}}\left(G_{\boldsymbol{\theta}}(\boldsymbol{z})\right)$

Problem I: Vanishing Gradients

We have shown that

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\phi}) \propto \mathbb{E}_{\boldsymbol{z} \sim p_0} D_{\boldsymbol{\phi}}\big(G_{\boldsymbol{\theta}}(\boldsymbol{z})\big) \approx 0 \textbf{ if } D_{\boldsymbol{\phi}} \approx 0$$

This means that

- Early in training, $G_{\boldsymbol{\theta}}$ generates very bad samples
- Then, $D_{\boldsymbol{\phi}}$ rejects them easily with high confidence, i.e. $D_{\boldsymbol{\phi}}\big(G_{\boldsymbol{\theta}}(\boldsymbol{z})\big) \approx 0$
- This causes $\nabla_{\boldsymbol{\theta}} L$ to vanish, preventing further learning of $\boldsymbol{\theta}$

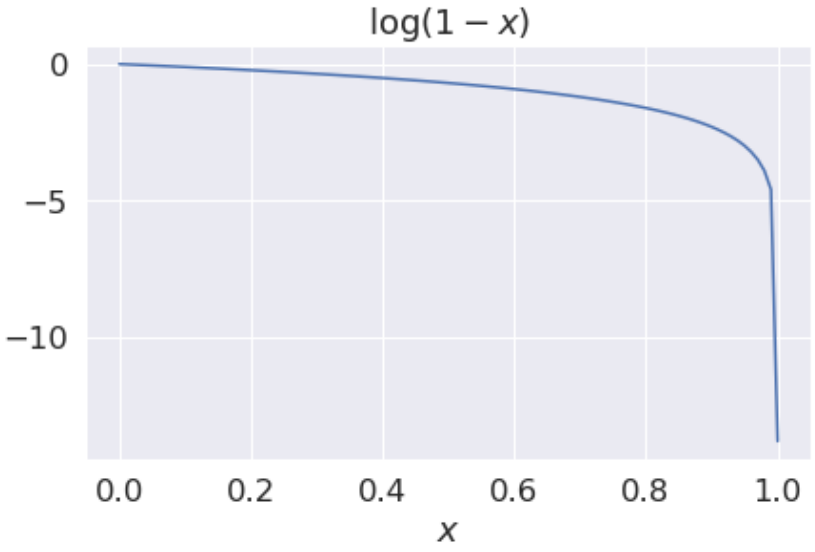
A Heuristic to Solve the Gradient Vanishing Problem

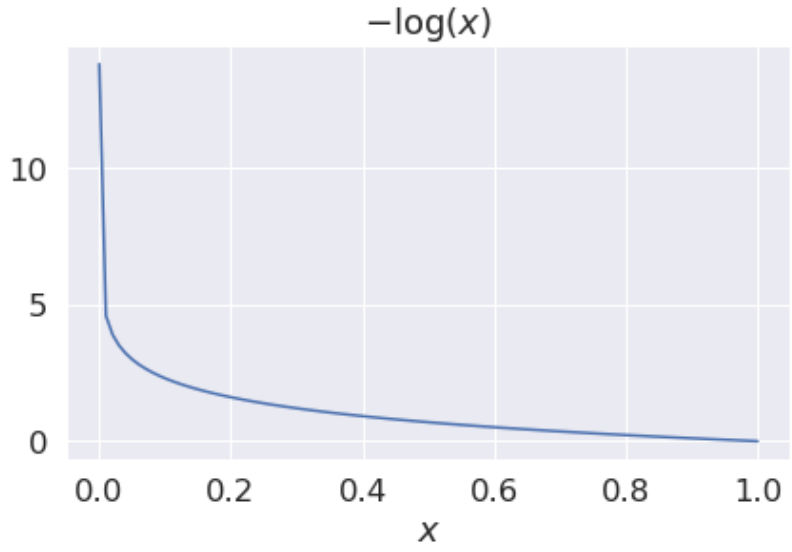
To solve the gradient vanishing problem for θ, we can replace

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{z} \sim p_0} \left[\log \left(1 - D_{\boldsymbol{\phi}} \left(G_{\boldsymbol{\theta}} (\boldsymbol{z}) \right) \right) \right]$$

by

$$\nabla_{\theta} \mathbb{E}_{\boldsymbol{z} \sim p_0} \left[- \log \left(D_{\boldsymbol{\phi}} \left(G_{\boldsymbol{\theta}} (\boldsymbol{z}) \right) \right) \right]$$





Problem II: Non-convergence
<Lecture Notebook>

Besides the gradient vanishing problem, in general GD is not designed to solve minimax problems.

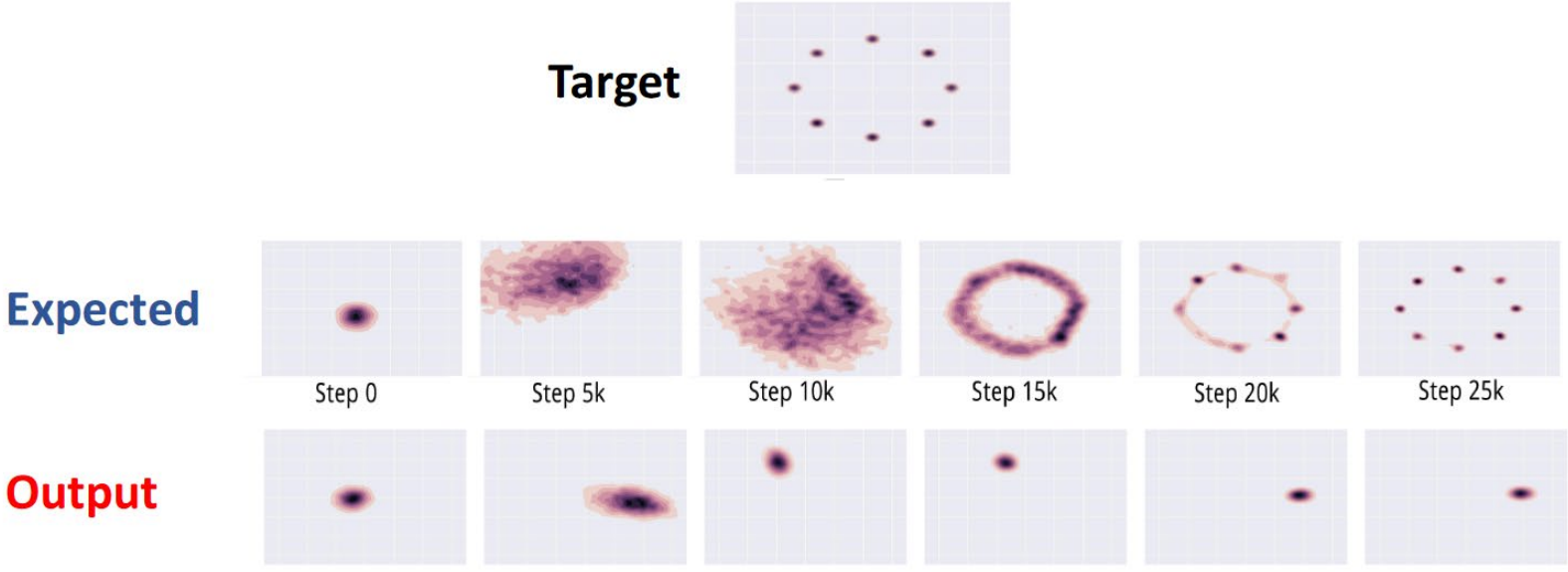
Consider solving

$$\min_x \max_y L(x, y) = xy$$

- The solutions are $x = 0,\ y \in \mathbb{R}$
- However, GD by alternating inner and outer loop will not find this solution!

Problem III: Mode Collapse

Mode collapse refers to the phenomenon that the generated samples are not varied and becomes very similar to a training sample.



Metz, Luke, et al. "Unrolled Generative Adversarial Networks."arXivpreprint arXiv:1611.02163 (2016).

Using Mini-Batch Statistics

Mode collapse can be detected, since generated samples are of small variance

To combat this, we can use *Mini-batch GAN*

- Generation happens in mini-batches, with batch statistics (e.g. variance, pair-wise L^2 norms) fed into discriminator
- Discriminator can tag an entire batch of examples as false if the variation is low
- Hence, generator is forced to produce varied samples

Wasserstein GANs

The Distance Viewpoint

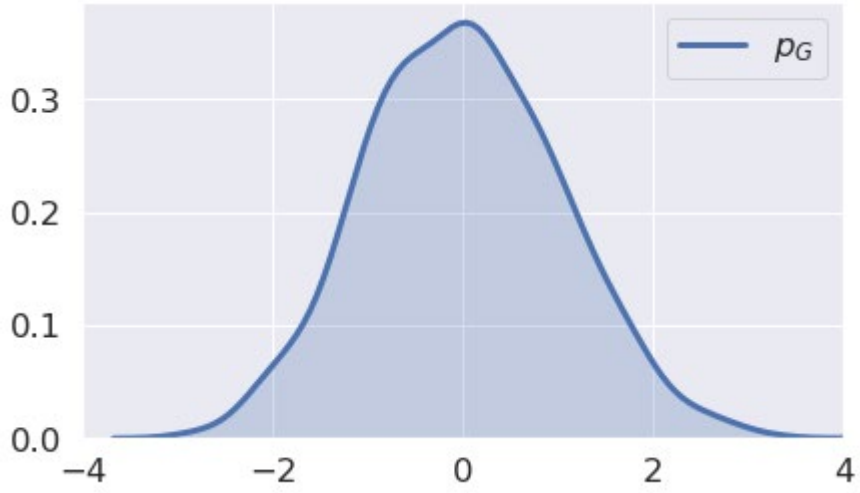
Let us return to the viewpoint of training GANs as *minimizing "distances"* between distributions.

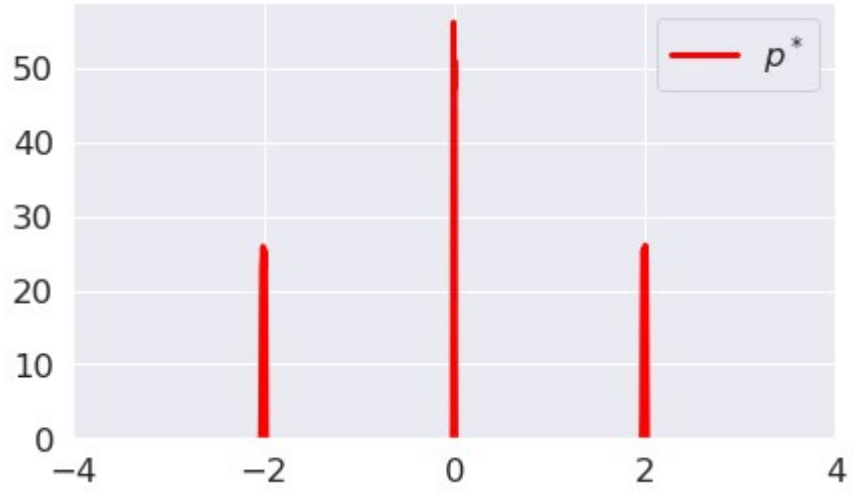
Recall that training GAN can be formulated as

$$\min_G D_{JS}(p_G \| p^*)$$
$$= \min_G \frac{1}{2} D_{KL}\left(p_G \| \frac{p^* + p_G}{2}\right) + \frac{1}{2} D_{KL}\left(p^* \| \frac{p^* + p_G}{2}\right)$$

What's wrong with KL/JS divergences?

KL/JS divergences don't play nice when the distributions are *singular*





Example: Singular KL Divergence

Let $p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$ **and** $q(x) = \frac{1}{2}\mathbb{I}_{x=1} + \frac{1}{2}\mathbb{I}_{x=-1}$

Then,

$$D_{KL}(p\|q) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

is undefined since for all $x \neq \pm 1$, $q(x) = 0$ **but** $p(x) \neq 0$.

In general, $D_{KL}(p\|q)$ **is finite if and only if** p **is *absolutely continuous* with respect to** q**, meaning that for any* subset** A**,**

$$\int_A q(x)dx = 0 \Rightarrow \int_A p(x)dx = 0$$

Otherwise, the D_{KL} **is infinite, and we say that** p **in *singular* with respect to** q**.**

Continuity and Distance

As in VAE and GAN, our primary idea of representing a distribution is to

- Sample latent $\boldsymbol{z} \sim p_0$ (simple distribution)
- Generate samples $G(\boldsymbol{z})$ with desired distribution p_G by adjusting G
- With parameterization: $G(\boldsymbol{z}) = G(\boldsymbol{z}; \boldsymbol{\theta})$ so $p_G = p_{\boldsymbol{\theta}}$

Then, the GAN training problem is

$$\min_{\boldsymbol{\theta}} \text{Distance}\,(p_{\boldsymbol{\theta}}, p^*)$$

In the usual GAN, the Distance function is the JS-divergence

Continuity and Distance

It is our hope to minimize $\text{Distance}(p_{\boldsymbol{\theta}}, p^*)$ **by gradient descent**

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} \text{Distance}(p_{\boldsymbol{\theta}}, p^*)$$

For this to work, it is essential that the following *continuity property* **be satisfied**

If we change θ *by a little bit, then* $\text{Distance}(p_{\boldsymbol{\theta}}, p^*)$ *changes also by only a little bit*

It turns out, that the JS-divergence (or KL-divergence) does not satisfy this property!

Example: Learning Parallel Lines
<Lecture Notebook>

Let $z \sim \mathcal{N}(0,1)$ and consider $\boldsymbol{x} \in \mathbb{R}^2$

- Let p^* be the distribution of $(0, z)$
- Let p_θ be the distribution of (θ, z)

Then, we can show

- $D_{KL}(p_\theta \| p^*) = \mathbb{I}_{\theta \neq 0}(+\infty)$
- $D_{JS}(p_\theta \| p^*) = \mathbb{I}_{\theta \neq 0}(\log 2)$

Both are not continuous at $\theta = 0$!

Modifying the GAN Problem

The natural question, is then, can we replace JS-divergence by another measure of distance so that the discontinuity issue is satisfied?

The answer is yes!

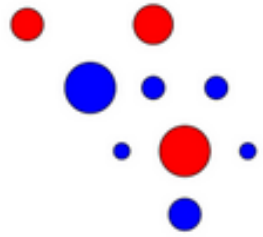
Consider the *Wasserstein distance*, or the *earth-mover's distance* (EMD)

$$W(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\gamma}[\|\boldsymbol{x}-\boldsymbol{y}\|]$$

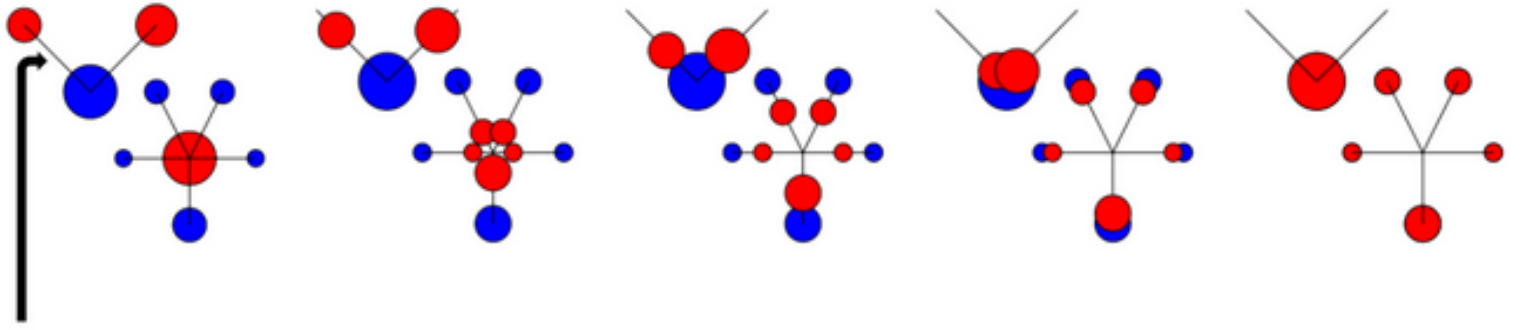
where $\Pi(p,q)$ **is the set of joint distributions whose marginals are** p **and** q **respectively, i.e.** $\gamma \in \Pi(p,q)$ **if and only if**

$$\int \gamma(\boldsymbol{x},\boldsymbol{y})\boldsymbol{dy} = p(\boldsymbol{x}) \quad and \quad \int \gamma(\boldsymbol{x},\boldsymbol{y})\boldsymbol{dx} = q(\boldsymbol{y})$$

EMD Visualized



- red distribution: “dirt”
- blue distribution: “holes”



The distance between points (ground distance) can be Euclidean distance, Manhattan…

Wasserstein GAN

The Wasserstein GAN transforms the problem

$$\min_{\boldsymbol{\theta}} D_{JS}(p_{\boldsymbol{\theta}} \| p^*) \rightarrow \min_{\boldsymbol{\theta}} W(p_{\boldsymbol{\theta}}, p^*)$$

Advantages

- Unlike D_{JS} (or D_{KL}), W is a true distance, i.e. a metric
- The mapping $\boldsymbol{\theta} \mapsto W(p_{\boldsymbol{\theta}}, p^*)$ is continuous (for the line problem, we have $W(p_\theta, p^*) = |\theta|$)
- The mapping $\boldsymbol{\theta} \mapsto W(p_{\boldsymbol{\theta}}, p^*)$ is differentiable almost everywhere if $\boldsymbol{\theta} \mapsto p_{\boldsymbol{\theta}}$ is Lipschitz

How to Compute W?

We need to compute

$$W(p_{\boldsymbol{\theta}}, p^*) = \inf_{\gamma \in \Pi(p_{\boldsymbol{\theta}}, p^*)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\gamma}[\|\boldsymbol{x} - \boldsymbol{y}\|]$$

and possibly also $\nabla_{\boldsymbol{\theta}} W(p_{\boldsymbol{\theta}}, p^*)$ for GD.

Direct computation via optimization is *not* tractable. However, we have a beautiful duality theory for W!

Kantorovich-Rubinstein Duality:

$$W(p_{\boldsymbol{\theta}}, p^*) = \sup_{f:\|f\|_L \leq 1} \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{\theta}}}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim p^*}[f(\boldsymbol{x})]$$

Here, $\|f\|_L$ denotes the Lipschitz constant for f (Think maximum value of $\|\nabla f\|$)

Wasserstein GAN

Owing to Kantorovich duality, we can approximate

$$W(p_{\boldsymbol{\theta}}, p^*) \approx \max_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{\theta}}}[f(\boldsymbol{x};\boldsymbol{\phi})] - \mathbb{E}_{\boldsymbol{x}\sim p^*}[f(\boldsymbol{x};\boldsymbol{\phi})]$$

with $f(x, \boldsymbol{\phi})$ a neural network ($\boldsymbol{\phi}$ are the weights).

In particular, if $\widehat{\boldsymbol{\phi}}$ attains this maximum then the gradient is

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} W &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{\theta}}}\left[f\left(\boldsymbol{x};\widehat{\boldsymbol{\phi}}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{z}\sim p_0} \nabla_{\boldsymbol{\theta}} f\left(G(\boldsymbol{z};\boldsymbol{\theta});\widehat{\boldsymbol{\phi}}\right)\end{aligned}$$

The last expectation can be computed using Monte-Carlo approximation.

Wasserstein GAN Algorithm

Train Dual

Repeat:

- For k steps do
  - Sample m noise samples $\{\boldsymbol{z}^{(1)}, \ldots \boldsymbol{z}^{(m)}\} \sim p_0$
  - Sample m real data samples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\} \in \mathcal{D}$
  - Update

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \epsilon_1 \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\phi}} \left[f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}\right) - f\left(G_{\boldsymbol{\theta}}\left(\boldsymbol{z}^{(i)}\right); \boldsymbol{\phi}\right) \right]$$

- Sample m noise samples $\{\boldsymbol{z}^{(1)}, \ldots \boldsymbol{z}^{(m)}\} \sim p_0$ and update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_2 \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} f\left(G_{\boldsymbol{\theta}}\left(\boldsymbol{z}^{(i)}\right); \boldsymbol{\phi}\right)$$

Train Generator

https://arxiv.org/pdf/1701.07875.pdf

Some Remarks on GANs

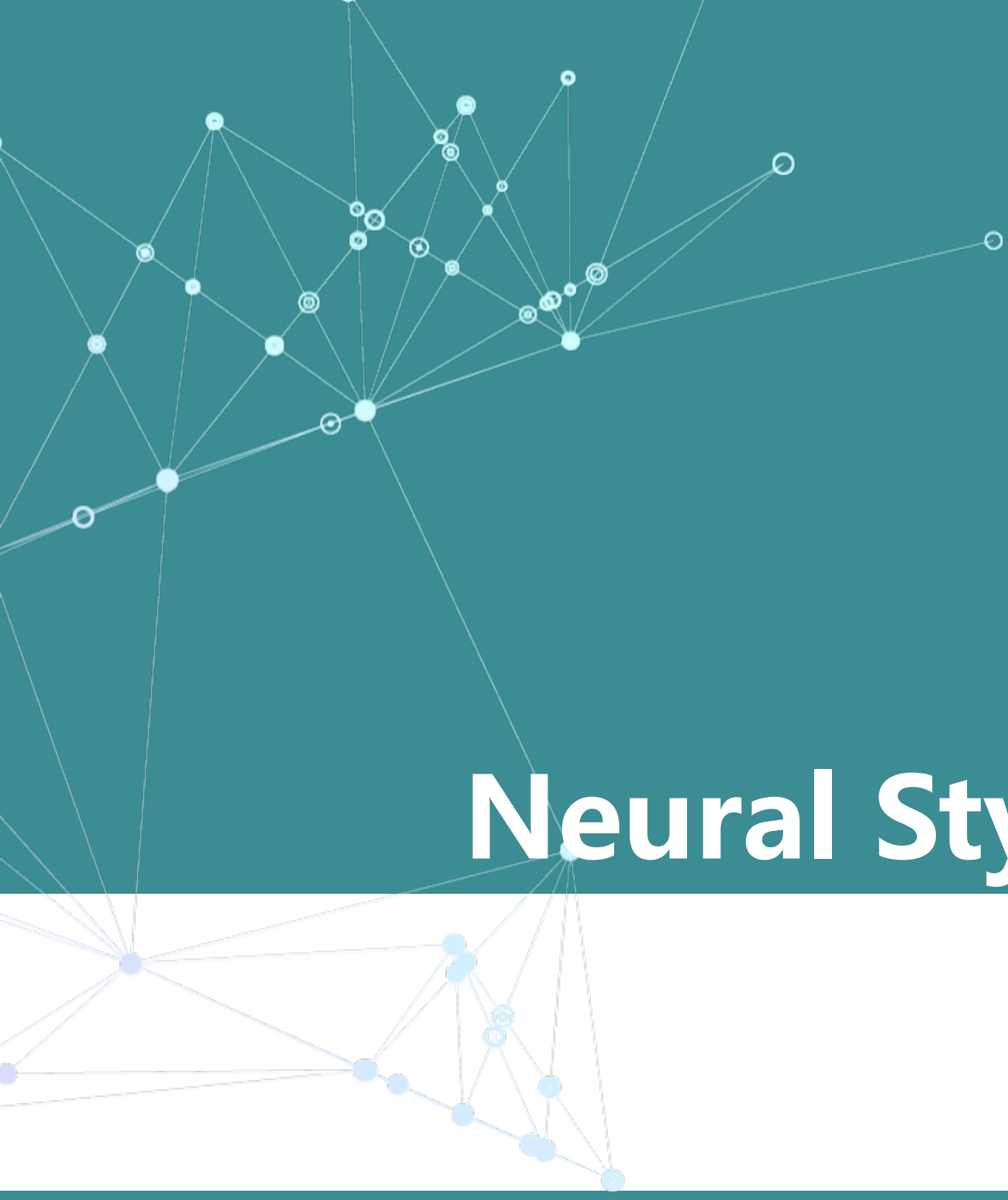
There are many variants of GAN architectures and training methods, and we have only touch on a few

Training GANs is not a solved problem!

Some resources:

- https://arxiv.org/pdf/2001.06937.pdf
- https://arxiv.org/abs/1906.01529.pdf

Demo: GAN for Image Generation



Neural Style Transfer

Transferring Styles

The style transfer task

- Given a content image x_c
- And a style image x_s
- Generate image x_{cs} with content x_c and style x_s

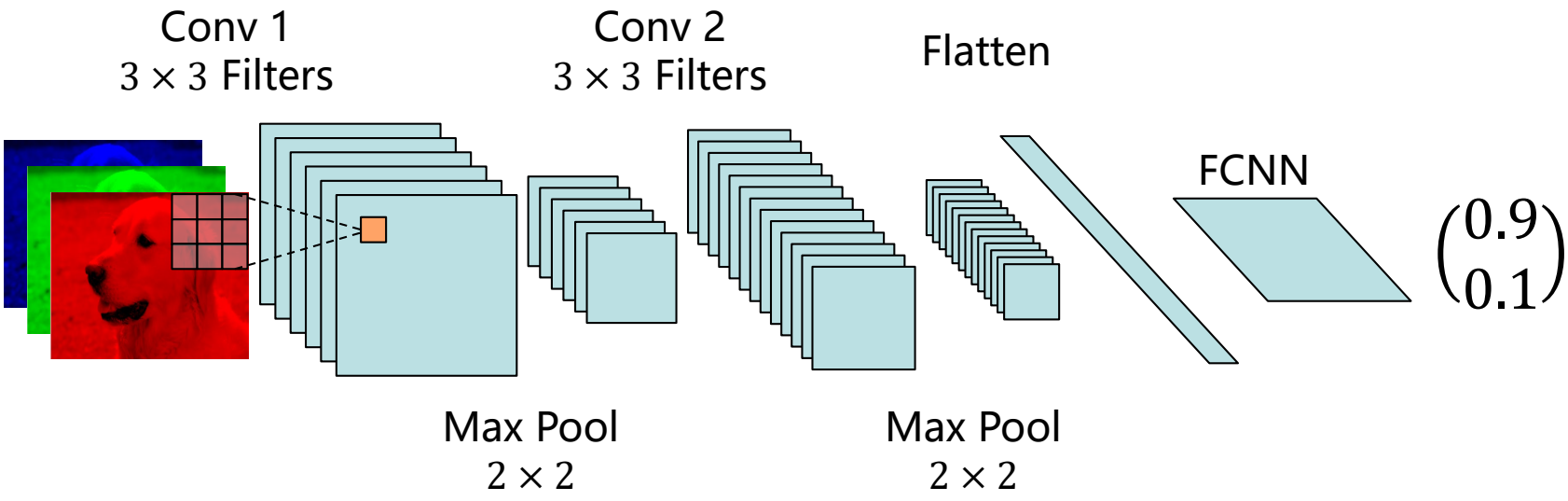
Content Image

+

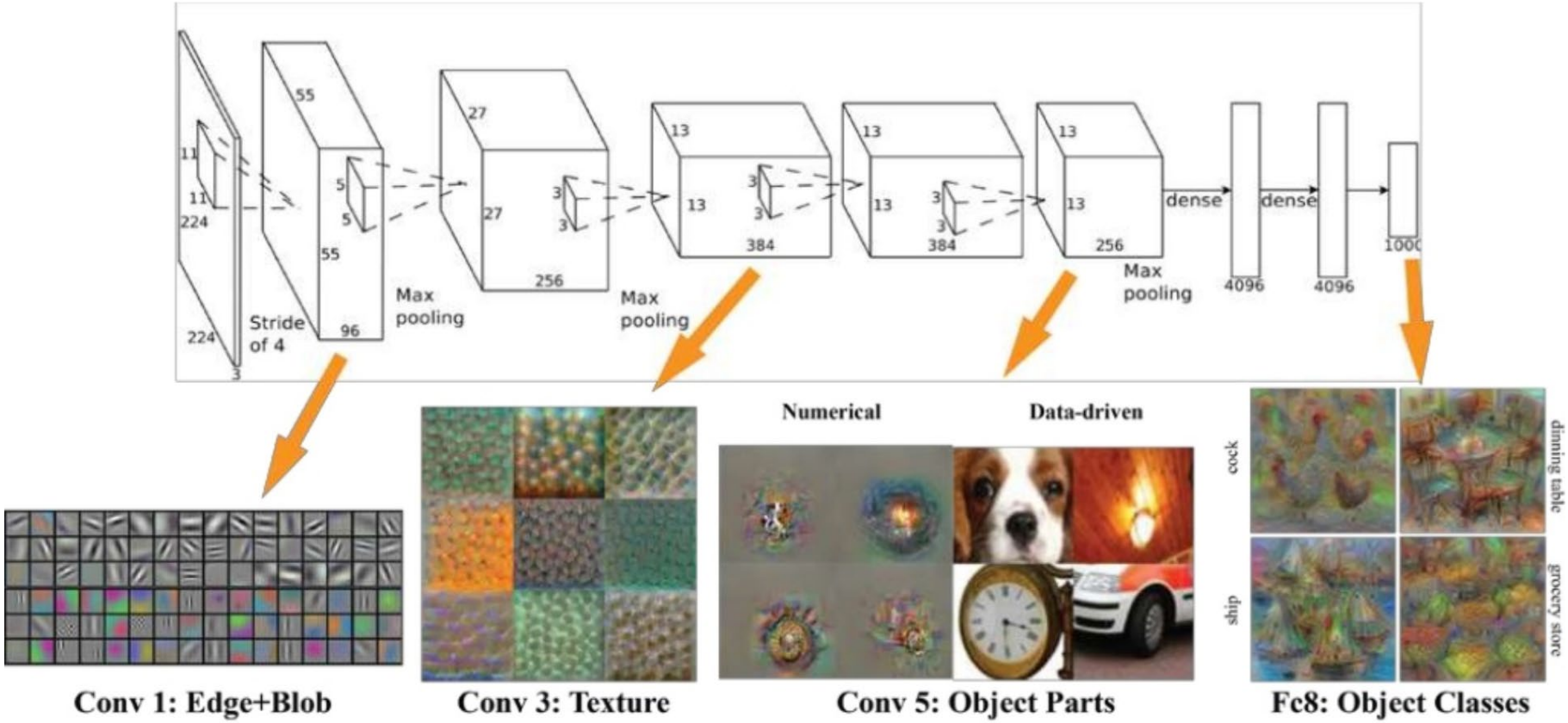
Style Image

Gatys et al, "A Neural Algorithm of Artistic Style", arXiv 2015

The Structure of CNNs



Features at Different Layers



Neural style transfer presentation done at Tensorflow summit Colombo 2018

The hierarchical representation of features in CNNs forms the basis of style transfer

Notation

We consider passing an input image x into a L-layer CNN.

The feature maps are

$$\boldsymbol{h}^{(\ell+1)} = \boldsymbol{f}^{(\ell)}\left(\boldsymbol{h}^{(\ell)}; \boldsymbol{\theta}^{(\ell)}\right), \qquad \boldsymbol{h}^{(0)} = \boldsymbol{x}$$

The function $\boldsymbol{f}^{(\ell)}$ is the ℓ^{th} convolutional block (e.g. Conv-BN-Activation-Pool).

The hidden feature maps are $\left\{\boldsymbol{h}^{(\ell)}\right\}$. We denote

$$h_{ij}^{(\ell)} = \mathrm{j}^{\mathrm{th}} \text{ pixel of } \mathrm{i}^{\mathrm{th}} \text{ channel feature at } \ell^{\mathrm{th}} \text{ layer}$$

To emphasize dependence on the initial input image x, we can write

$$h_{ij}^{(\ell)} = h_{ij}^{(\ell)}(\boldsymbol{x})$$

Combining Content and Style Losses

Our goal is to generate an image x with content x_c and style x_s

The idea is to perform minimization

$$\min_{\boldsymbol{x}} \alpha L_c(\boldsymbol{x}, \boldsymbol{x_c}) + \beta L_s(\boldsymbol{x}, \boldsymbol{x_s})$$

Here,

- L_c is the *content loss* that measures similarity in content
- L_s is the *style loss* that measures similarity in style

Content Loss

For the content loss we can take the square distance between the respective feature maps

$$E_c(\boldsymbol{x}, \boldsymbol{x_c}, \ell) = \frac{1}{2} \sum_{i,j} \left(h_{ij}^{(\ell)}(\boldsymbol{x}) - h_{ij}^{(\ell)}(\boldsymbol{x_c}) \right)^2$$

The overall content loss is a weighted sum

$$L_c(\boldsymbol{x}, \boldsymbol{x_c}) = \sum_{\ell \in \ell_c} E_c(\boldsymbol{x}, \boldsymbol{x_c}, \ell)$$

where ℓ_c **is the set of layers relevant to content**

Style Loss

To define the style loss, we first define the *Gram matrix*

Given a dataset matrix X (each row is a data vector), the Gram matrix is given by

$$G = XX^T$$

which has the dimensions $N \times N$

In fact, $G_{ij} = \left[x^{(i)}\right]^T x^{(j)}$, hence G measures the *correlations or similarities* amongst different data vectors, and is a summary of the statistics of our data

Style Loss

Therefore, we can define the layer-wise style loss

$$E_s(\boldsymbol{x}, \boldsymbol{x_s}, \ell) = \frac{1}{4N_l M_l} \sum_{i,j} \left(G_{ij}^{(\ell)}(\boldsymbol{x}) - G_{ij}^{(\ell)}(\boldsymbol{x_s}) \right)^2$$

where N_l is the number of channels, M_l is the image size and $G^{(\ell)}$ **is the Gram matrix of the features**

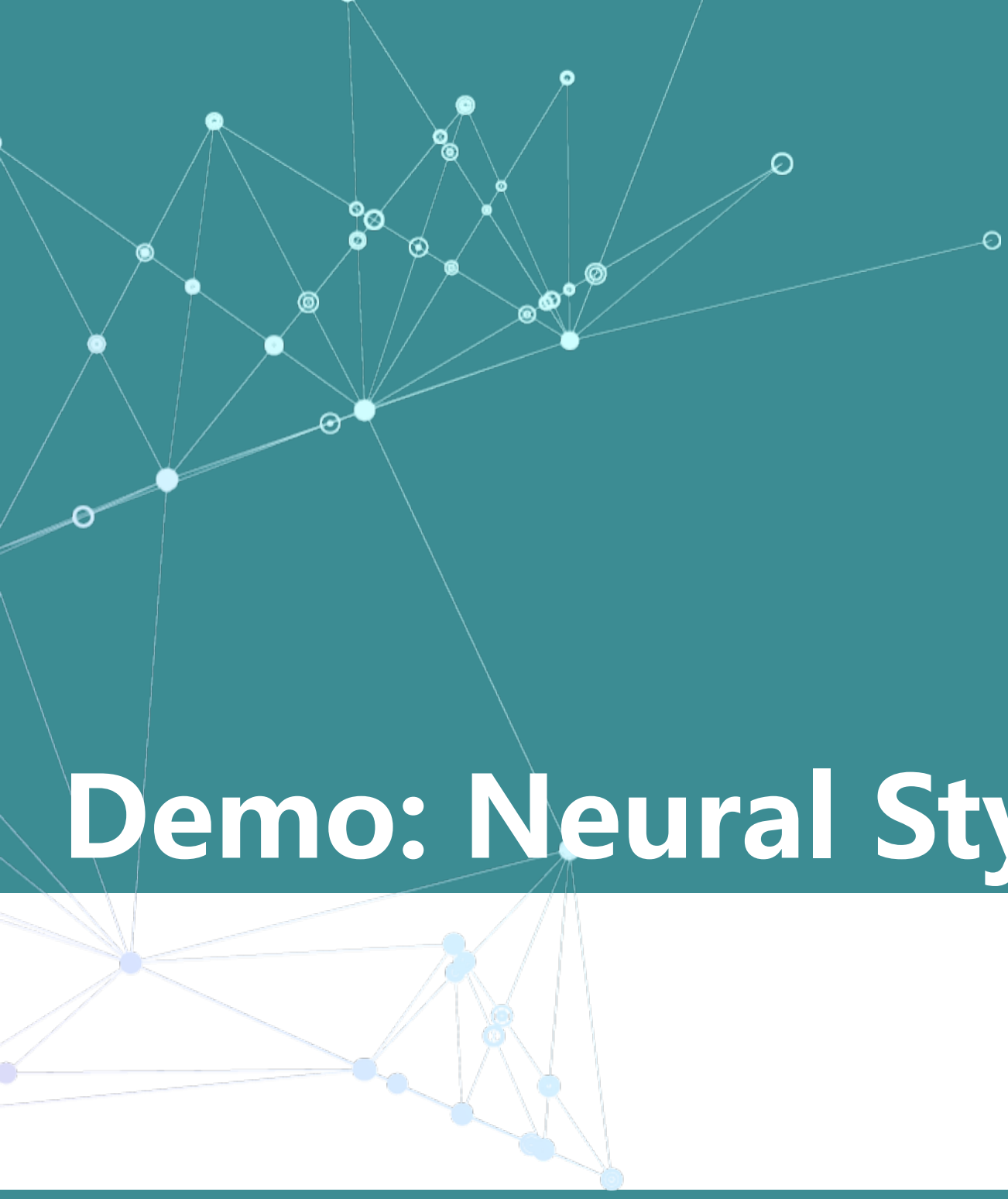
$$G_{ij}^{(\ell)}(\boldsymbol{x}) = \sum_k h_{ik}^{(\ell)}(\boldsymbol{x}) h_{jk}^{(\ell)}(\boldsymbol{x})$$

The total style loss is

$$L_s(\boldsymbol{x}, \boldsymbol{x_s}) = \sum_{\ell \in \ell_s} E_s(\boldsymbol{x}, \boldsymbol{x_s}; \ell)$$

Further Reading on NST

- https://arxiv.org/pdf/1705.04058.pdf (Review)
- https://arxiv.org/pdf/1508.06576.pdf (Original)



Demo: Neural Style Transfer

Summary

In this class, we introduced two other types of generative models

- Generative Adversarial Networks
- Neural Style Transfer

They represent a combination of ideas we have introduced previously

- Training with adversaries
- Generating samples using latent variables
- Using hierarchical representations in CNNs

Summary of DSA5204

This class is an overall introduction of deep learning

- Architectures (FCNNs, CNNs, RNNs)
- Optimization methods (SGD and variants)
- Regularization methods (batch-norm, data augmentation, dropout, etc.)

We have also seen a variety of applications

- Supervised learning (classification, regression)
- Unsupervised learning (compression, denoising)
- Generative modelling

Next Steps

This class is just an introduction, and there are many topics that are not covered, e.g. Bayesian NNs, attention models and many more!

How to learn more about deep learning?

- Read papers (NIPS, ICLR, ICML, CVPR, JMLR, etc)
- Follow ArXiV preprints
- Tensorflow/pytorch demonstrations
- Do an application project that interests you!