



生成对抗网络

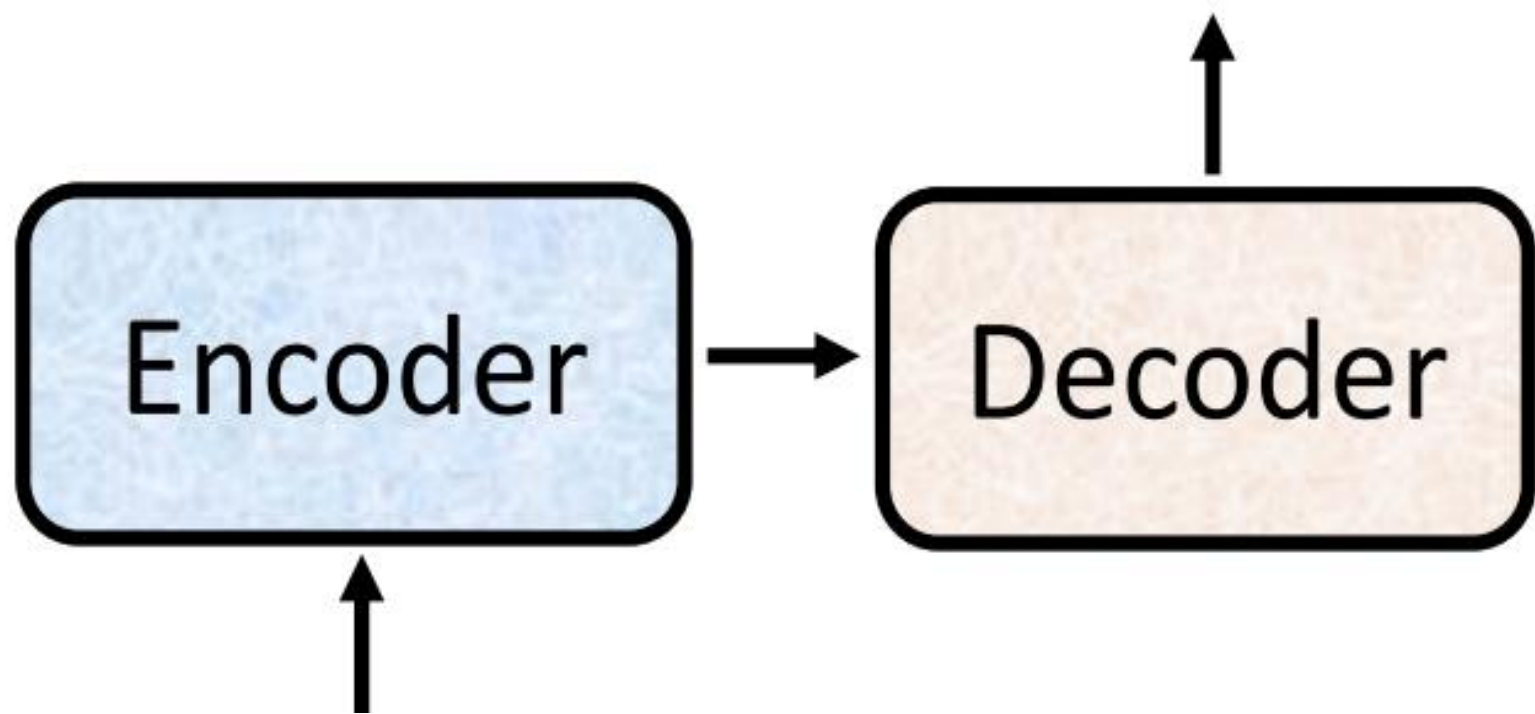
Generative Adversarial Network

《人工神经网络》第

10

讲

来源：《神经网络与深度学习》、GAN(v11).pptx、课程讲义



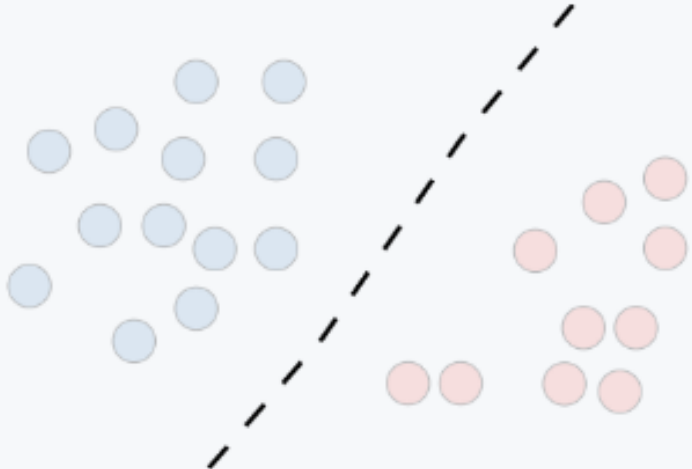
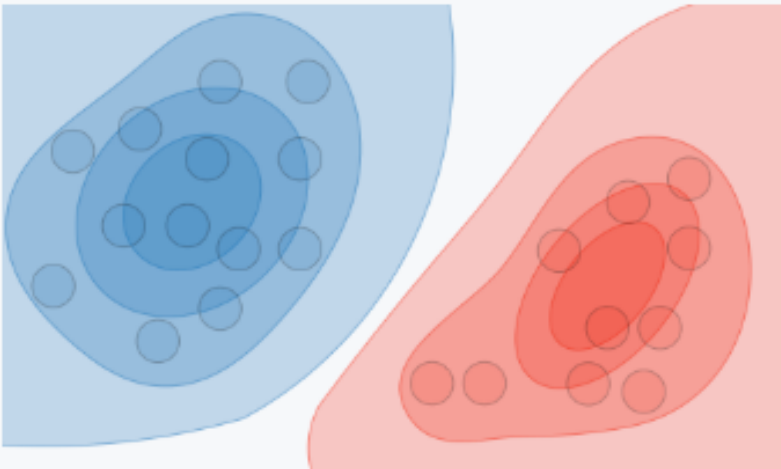
主要内容

1. 自编码器(AE)
2. 对抗生成网络 (GAN)
3. 一些应用举例



1 自编码器 & 生成模型

机器学习的两种范式

	Discriminative model	Generative model
Goal	Directly estimate $P(y x)$	Estimate $P(x y)$ to then deduce $P(y x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes



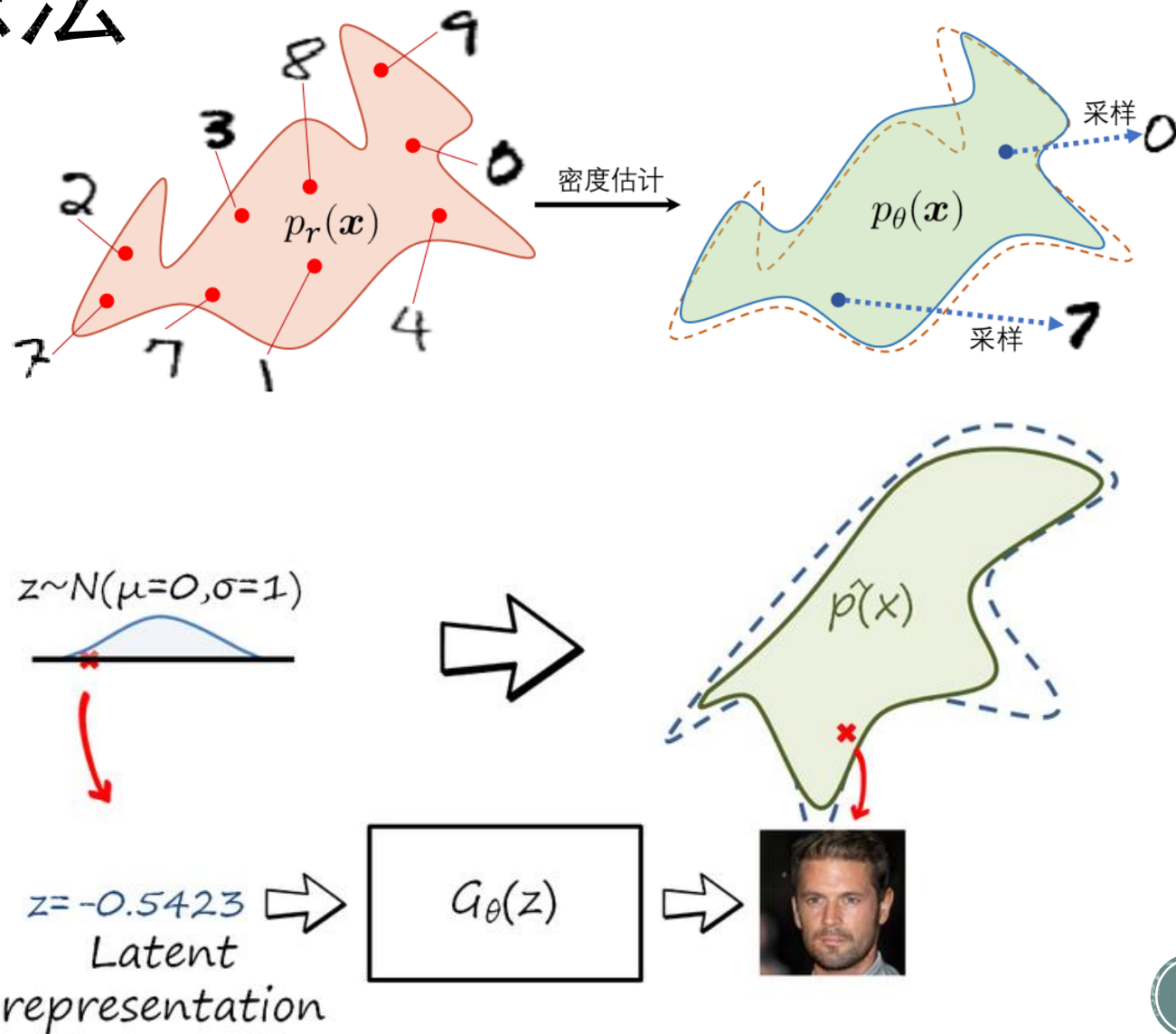
生成模型 - 基本想法

➤ 一系列用于随机生成可观测数据的模型

包含两个步骤：

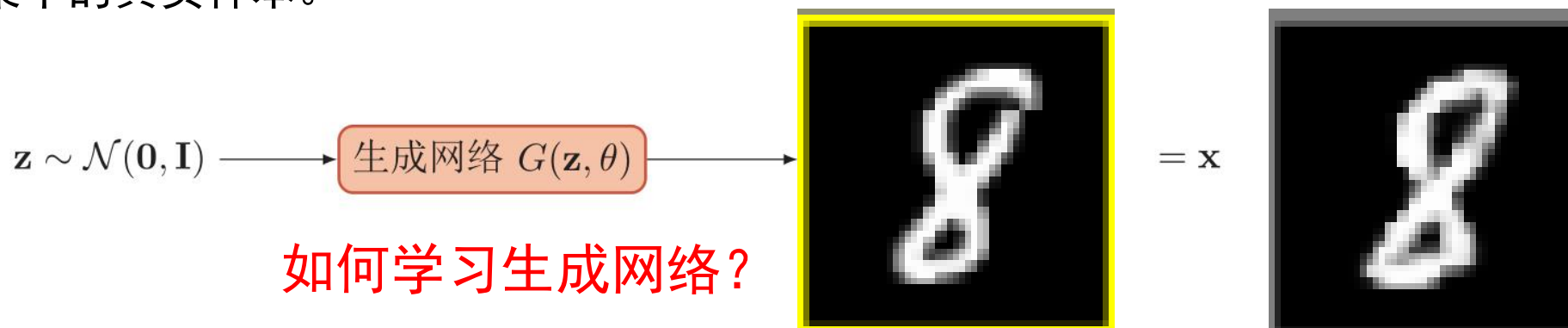
1. 密度估计
2. 采样

➤ 生成数据的另一种思路：密度估计



生成网络：基于神经网络的生成模型

- 生成网络从隐空间（latent space）中随机采样作为输入，其输出结果需要尽量模仿训练集中的真实样本。



深度生成模型可以认为是以某种方式寻找并表达数据的概率分布。

主流的深度生成模型：

1. VAE(Variational Auto Encoder)
2. GANs(Generative Adversarial Networks).



无监督学习 (Unsupervised Learning)

▶ 监督学习

- ▶ 建立映射关系 $f: x \rightarrow y$

▶ 无监督学习

- ▶ 指从无标签的数据中学习出一些有用的模式。

- ▶ 聚类：建立映射关系 $f: x \rightarrow y$

- ▶ 不借助于任何人工给出标签或者反馈等指导信息

- ▶ 特征学习

- ▶ 密度估计 $p(x)$

大脑有约 10^{14} 个突触，我们只能活大约 10^9 秒。所以我们有比数据更多的参数。这启发了我们必须进行大量无监督学习的想法，因为感知输入（包括本体感受）是我们可以获得每秒 10^5 维约束的唯一途径。

-- Geoffrey Hinton, 2014 AMA on Reddit

监督学习 VS 无监督学习

监督学习

1. 目标明确
2. 需要带标签的训练数据
3. 效果容易评估

无监督学习

1. 目标不明确
2. 不需要带标签的数据
3. 效果很难评估



主成份分析 (Principal Component Analysis, PCA)

▶ 一种最常用的数据降维方法，使得在转换后的空间中数据的方差最大。

▶ 样本点 $\mathbf{x}^{(n)}$ 投影之后的表示为

$$z^{(n)} = \mathbf{w}^\top \mathbf{x}^{(n)}$$

▶ 所有样本投影后的方差为

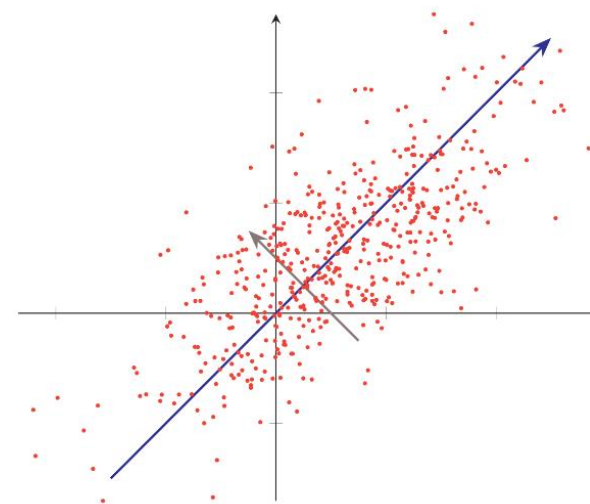
$$\begin{aligned}\sigma(\mathbf{X}; \mathbf{w}) &= \frac{1}{N} \sum_{n=1}^N (\mathbf{w}^\top \mathbf{x}^{(n)} - \mathbf{w}^\top \bar{\mathbf{x}})^2 \\ &= \frac{1}{N} (\mathbf{w}^\top \mathbf{X} - \mathbf{w}^\top \bar{\mathbf{X}})(\mathbf{w}^\top \mathbf{X} - \mathbf{w}^\top \bar{\mathbf{X}})^\top \\ &= \mathbf{w}^\top \Sigma \mathbf{w},\end{aligned}$$

▶ 目标函数

$$\max_{\mathbf{w}} \mathbf{w}^\top \Sigma \mathbf{w} + \lambda(1 - \mathbf{w}^\top \mathbf{w})$$

▶ 对目标函数求导并令导数等于 0，可得

$$\Sigma \mathbf{w} = \lambda \mathbf{w}$$



(线性) 编码

- ▶ 给定一组基向量 $A = [\mathbf{a}_1, \dots, \mathbf{a}_M]$ ，将输入样本 \mathbf{x} 表示为这些基向量的线性组合

$$\mathbf{x} = \sum_{m=1}^M z_m \mathbf{a}_m$$

$$= \mathbf{A}\mathbf{z},$$

字典 (dictionary)

编码 (encoding)

$$\begin{bmatrix} \text{green} \\ \text{blue} \\ \text{yellow} \\ \text{orange} \\ \text{blue} \\ \text{light blue} \end{bmatrix} = \begin{bmatrix} \text{grid of colored squares} \end{bmatrix} \begin{bmatrix} ? \\ ? \\ \cdot \\ \cdot \\ ? \\ ? \end{bmatrix}$$

$\mathbf{x} \qquad \qquad \mathbf{A} \qquad \qquad \mathbf{z}$

自编码器 - 定义

- 一种典型的生成模型，通过无监督学习使输出尽可能地和输入相同，从而学习数据的分布
- 包含两个部分：编码器和解码器，如图所示：
- 工作流程：

- 利用编码器 对 x 进行编码，即

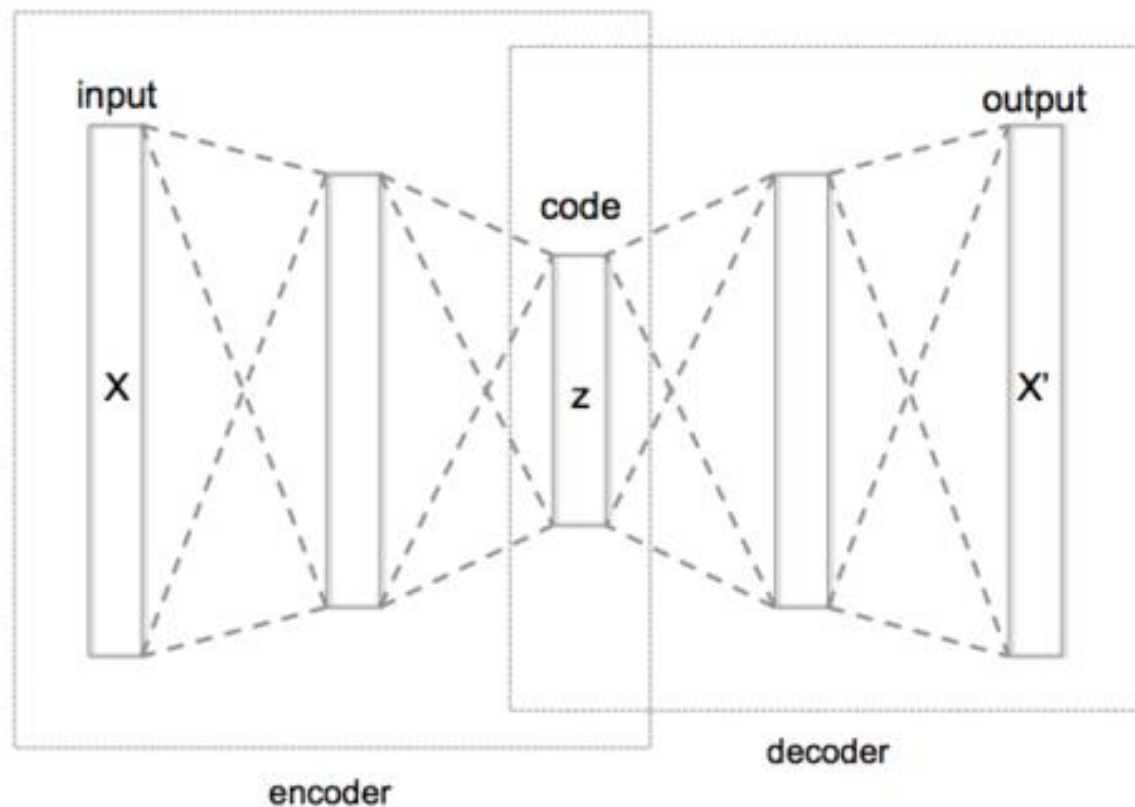
$$h = f(x),$$

形成所对应的编码 z

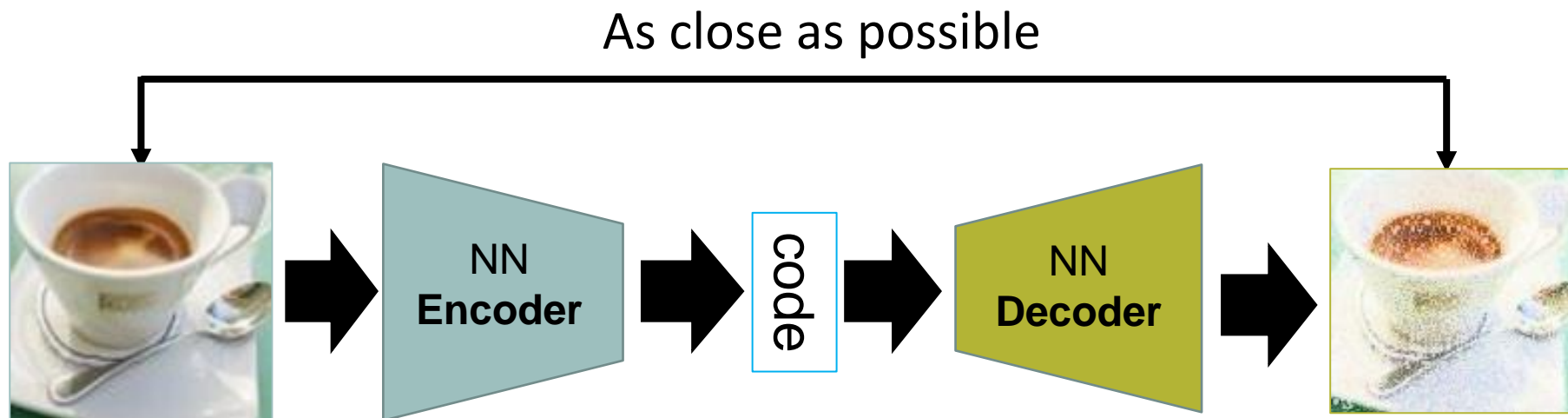
- 运用码器进行解码，获得重建向量 $x' = g(f(x))$
- 这是一个学习过程，它可以表示为

$$\min L(x, x')$$

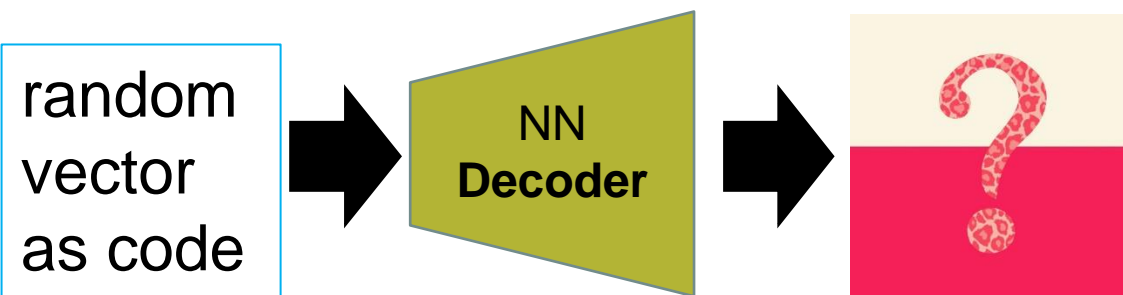
- 其中 L 为损失函数，通常采用均方误差。编码器的目标是使得： $x' \approx x$ --- 重建、降维、不相等！



自编码器 - 基本思想



- “压缩特征”？



自编码器 – PyTorch实现示例

```
class autoencoder1(nn.Module):
    def __init__(self, in_features, hidden_features):
        super(autoencoder1, self).__init__()
        self.encoder = nn.Sequential(nn.Linear(in_features, hidden_features), nn.ReLU(True))
        self.decoder = nn.Sequential(nn.Linear(hidden_features, in_features), nn.Sigmoid())

    def forward(self, x):
        en = self.encoder(x)
        de = self.decoder(en)
        return en, de
```

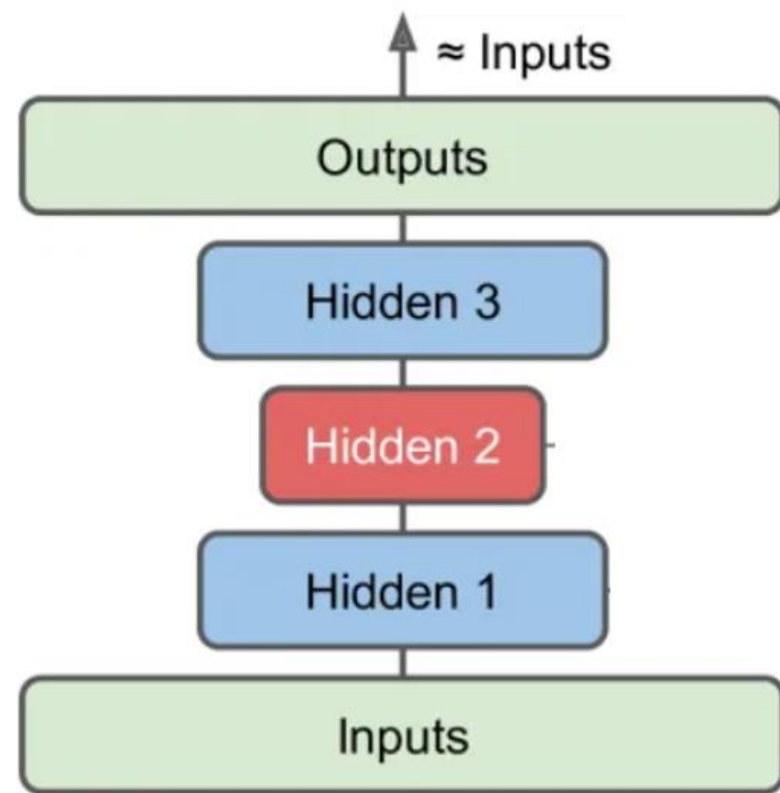
```
class autoencoder2(nn.Module):
    def __init__(self, in_features, hidden_features):
        super(autoencoder2, self).__init__()

        self.encoder = nn.Sequential(
            nn.Conv2d(in_features, hidden_features, kernel_size = 3, padding = 1),
            nn.ReLU(True), nn.MaxPool2d(kernel_size = 2), nn.ReLU(True))
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(hidden_features, hidden_features, kernel_size = 2, stride=2),
            nn.ReLU(True),
            nn.ConvTranspose2d(hidden_features, in_features, kernel_size=3, padding = 1),
            nn.Tanh())

    def forward(self, x):
        en = self.encoder(x)
        de = self.decoder(en)
        return en, de
```


典型的自编码器

1. 变分自编码器 (Variational AutoEncoder, VAE)
2. 稀疏自编码器 (Sparse AutoEncoder)
3. 降噪自编码器 (Denoising AutoEncoder)
4. 收缩自编码器 (Contractive AutoEncoder)
5. 堆叠自编码器 (Stacked AutoEncoder, SAE)

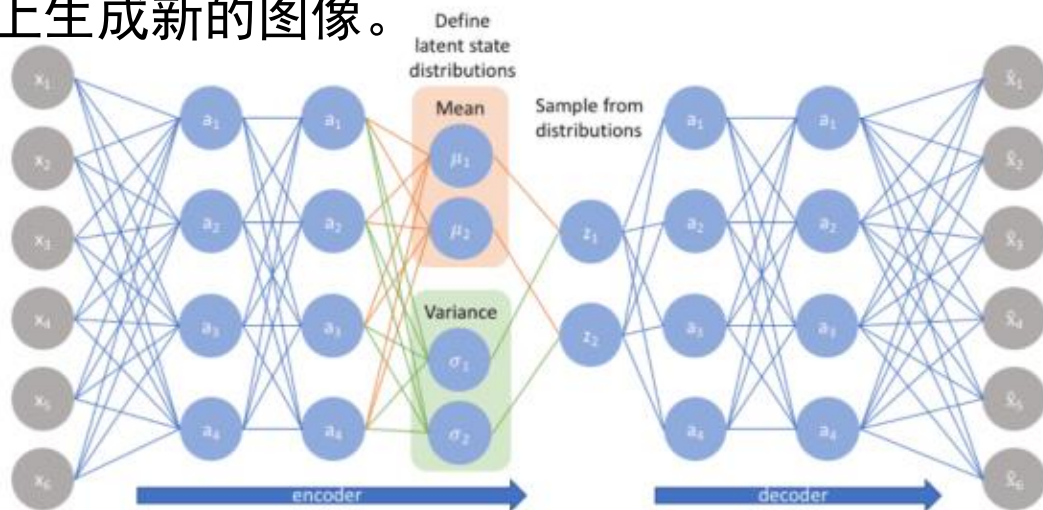


变分自编码器

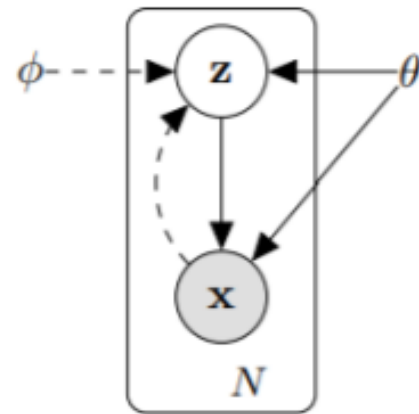
- VAE是一种基于神经网络的概率模型,任务是学习一个概率分布。损失函数

$$\mathcal{L}(\theta, \phi; x^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^J \left(1 + \log \left(\left(\sigma_j^{(i)} \right)^2 \right) - \left(\mu_j^{(i)} \right)^2 - \left(\sigma_j^{(i)} \right)^2 \right) + \frac{1}{L} \sum_{l=1}^L \log p_{\theta} \left(x^{(i)} \mid z^{(i,l)} \right),$$

- 用来衡量输入与输出之间的差距
 - 用KL散度来表示 $q(z|x)$ 和 $p(z)$ 之间的相似程度
- 目的: 使得网络能够在不给定原始图像的基础上生成新的图像。



```
1 import torch
2 import torch.nn as nn
3 from torch.nn.modules.loss import _Loss
4
5 class VAE_Loss(_Loss):
6     def __init__(self, reduction='sum'):
7         super(VAE_Loss, self).__init__()
8         self.bce_loss = torch.nn.BCELoss(reduction = reduction)
9
10    def forward(self, input, target, x_mean, x_logvar):
11        loss1 = self.bce_loss(input, target)
12        loss2 = self.KL_divergence(x_mean, x_logvar)
13        return loss1 + loss2
14
15    def KL_divergence(x_mean, x_logvar):
16        return - 0.5 * torch.sum(1 + x_logvar - torch.exp(x_logvar) - x_mean**2)
17
18 class VAE(nn.Module):
19     def __init__(self, in_features, hidden_features, out_features, n_num):
20         super(VAE, self).__init__()
21
22         self.encoder = nn.Sequential(nn.Linear(in_features, hidden_features),nn.ReLU(True))
23         self.mean=nn.Linear(hidden_features, n_num)
24         self.logvar=nn.Linear(hidden_features, n_num)
25
26         self.decoder = nn.Sequential(nn.Linear(n_num, hidden_features),nn.ReLU(True),nn.Linear(hidden_features,
27             out_features),nn.Tanh())
28
29     def forward(self, x):
30         x = self.encoder(x)
31         x_mean=self.mean(x)
32         x_logvar=self.logvar(x)
33         std = 0.5 * torch.exp(x_logvar)
34         z = torch.randn(std.size()) * std + x_mean
35         z = self.decoder(z)
36         return z, x_mean, x_logvar
37
38 Loss = VAE_Loss()
```



完备性

数学小知识 | 完备性

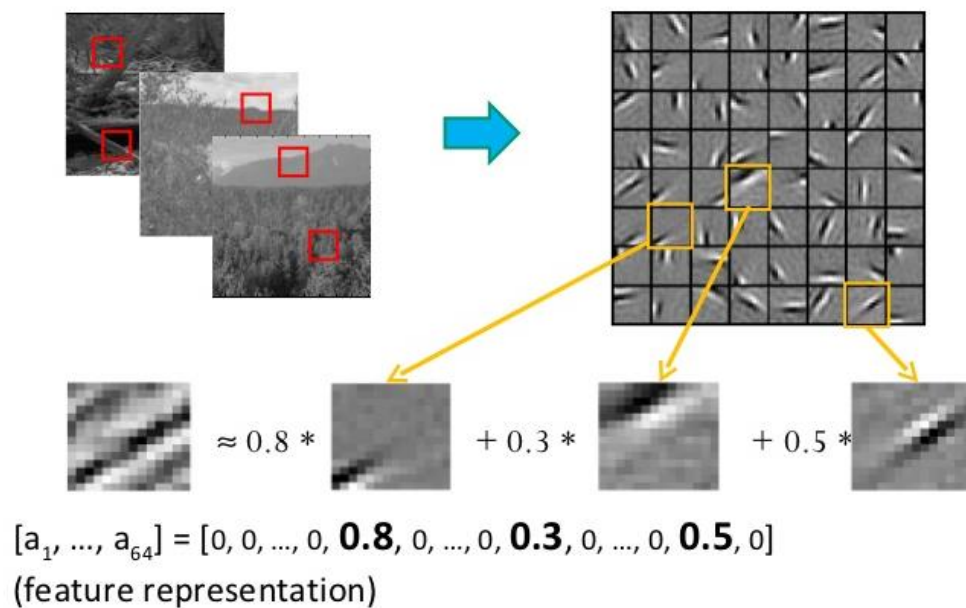
如果 M 个基向量刚好可以支撑 M 维的欧氏空间, 则这 M 个基向量是完备的. 如果 M 个基向量可以支撑 D 维的欧氏空间, 并且 $M > D$, 则这 M 个基向量是过完备的 (overcomplete)、冗余的.

“过完备”基向量是指基向量个数远远大于其支撑空间维度. 因此这些基向量一般不具备独立、正交等性质.

► 稀疏编码

- 找到一组“**过完备**”的基向量 (即 $M > D$) 来进行编码。

Sparse coding illustration



Slide credit: Andrew Ng

Compact & easily interpretable

稀疏编码 (Sparse Coding)

▶ 给定一组 N 个输入向量 $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}$, 其稀疏编码的目标函数定义为

$$\mathcal{L}(\mathbf{A}, \mathbf{Z}) = \sum_{n=1}^N \left(\left\| \mathbf{x}^{(n)} - \mathbf{A}\mathbf{z}^{(n)} \right\|^2 + \eta \rho(\mathbf{z}^{(n)}) \right)$$

▶ $\rho(\cdot)$ 是一个稀疏性衡量函数, η 是一个超参数, 用来控制稀疏性的强度。

$$\rho(\mathbf{z}) = \sum_{i=1}^p \mathbf{I}(|z_i| > 0)$$

$$\rho(\mathbf{z}) = \sum_{i=1}^p \log(1 + z_i^2)$$

$$\rho(\mathbf{z}) = \sum_{i=1}^p |z_i|$$

$$\rho(\mathbf{z}) = \sum_{i=1}^p -\exp(-z_i^2)$$

稀疏编码的优点

► 计算量

- 稀疏性带来的最大好处就是可以极大地降低计算量。

► 可解释性

- 因为稀疏编码只有少数的非零元素，相当于将一个输入样本表示为少数几个相关的特征。这样我们可以更好地描述其特征，并易于理解。

► 特征选择

- 稀疏性带来的另外一个好处是可以实现特征的自动选择，只选择和输入样本相关的最少特征，从而可以更好地表示输入样本，降低噪声并减轻过拟合。

训练过程

► 稀疏编码的训练过程:

一般用交替优化的方法进行。

- 1) 固定基向量 \mathbf{A} , 对每个输入 $\mathbf{x}^{(n)}$, 计算其对应的最优编码

$$\min_{\mathbf{z}^{(n)}} \left\| \mathbf{x}^{(n)} - \mathbf{A}\mathbf{z}^{(n)} \right\|^2 + \eta \rho(\mathbf{z}^{(n)}), \forall n \in [1, N].$$

- 2) 固定上一步得到的编码 $\{\mathbf{z}^{(n)}\}_{n=1}^N$, 计算其最优的基向量

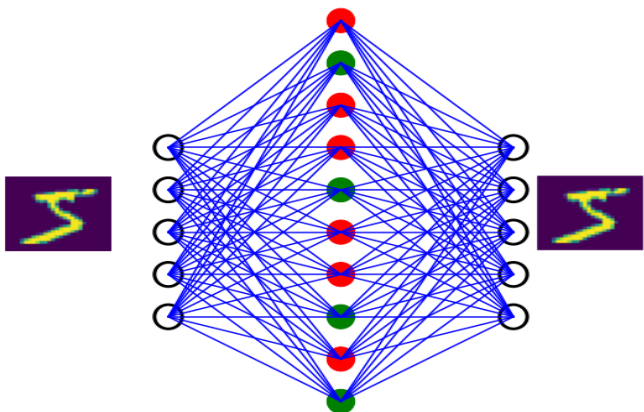
$$\min_{\mathbf{A}} \sum_{n=1}^N \left(\left\| \mathbf{x}^{(n)} - \mathbf{A}\mathbf{z}^{(n)} \right\|^2 \right) + \lambda \frac{1}{2} \|\mathbf{A}\|^2,$$

稀疏自编码器

- 通过给自编码器中隐藏层单元 z 加上稀疏性限制，自编码器可以学习到数据中一些有用的结构。

$$\mathcal{L} = \sum_{n=1}^N \|\mathbf{x}^{(n)} - \mathbf{x}'^{(n)}\|^2 + \eta \rho(\mathbf{Z}) + \lambda \|\mathbf{W}\|^2$$

- 目标函数
 - W 表示自编码器中的参数



- 和稀疏编码一样，稀疏自编码器的优点是有很高的可解释性，并同时进行了隐式的特征选择。

```

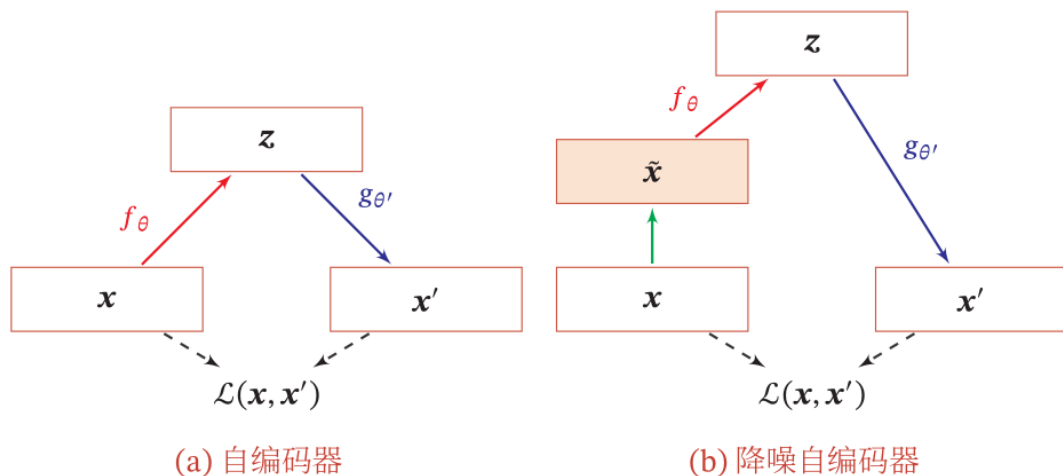
1 import torch
2 from torch.nn.modules.loss import _Loss
3
4 class SAE_Loss(_Loss):
5     def __init__(self, reduction='sum'):
6         super(SAE_Loss, self).__init__()
7         bce = torch.nn.BCELoss(reduction='sum')
8
9     def forward(self, input, target, rho, _beta):
10         loss1 = bce(input, target)
11         loss2 = KL(rho, input)
12         return loss1 + _beta * loss2
13
14
15 def KL_divergence(p, q):
16
17     q = torch.nn.functional.softmax(q, dim=0)
18     q = torch.sum(q, dim=0)/batch_size
19     s1 = torch.sum(p*torch.log(p/q))
20     s2 = torch.sum((1-p)*torch.log((1-p)/(1-q)))
21     return s1+s2

```



降噪自编码器

- 通过引入噪声来增加编码鲁棒性的自编码器
 - 对于一个向量 x ，我们首先根据一个比例 μ 随机将 x 的一些维度的值设置为0，得到一个被损坏的向量 \tilde{x} 。
 - 然后将被损坏的向量 \tilde{x} 输入给自编码器得到编码 z ，并重构出原始的无损输入 x 。



```
1 import torch
2 import torch.nn as nn
3 from torch.nn.modules.loss import _Loss
4 from torch.autograd import functional
5
6 class CAE_Loss(_Loss):
7     def __init__(self, reduction='sum'):
8         super(CAE_Loss, self).__init__()
9         self.MSE_Loss=torch.nn.MSELoss(reduce=True, size_average=False)
10
11     def forward(self, input, target, encoder, gamma):
12         loss1 = self.MSE_Loss(input, target)
13         loss2 = self.Jacobian(target, encoder)
14         loss = loss1 + gamma * loss2
15         return loss
16
17     def Jacobian(self, x, encoder):
18         J = functional.jacobian(encoder, x, create_graph = True)
19         J = sum([par.norm()**2 for par in J])
20         return J
21
22 # Test by autoencoder1
23 x = torch.randn(1,1,28,28)
24 x = torch.flatten(x, start_dim=1)
25 model = autoencoder1(28*28,64)
26 Loss = CAE_Loss()
27 en, de = model(x)
28 loss = Loss(de,x,model.encoder,0.05)
29 loss.backward()
```

收缩自编码器

- 添加基于Jacobian矩阵的敏感性惩罚项，能够使特征提取函数能抵抗极小的输入扰动。
- 对于数据x和编码过程f， $h = f(x)$ 为隐藏层的表示，则惩罚项可以表示为h的所有分量偏导的平方

$$\|J_f(x)\|_F^2 = \sum_{ij} \left(\frac{\partial h_j(x)}{\partial x_i} \right)^2$$

- 损失函数

$$L_{CAE} = \sum_{x \in D} (L(x, g(f(x))) + \lambda \|J_f(x)\|_F^2)$$

- 高阶收缩自编码器的 损失函数：

$$L_{CAE}(W) = \sum (L(x, g(f(x))) + \lambda \|J_f(x)\|_F^2 + \gamma E[\|J_f(x) - J(x + \epsilon)\|_F^2])$$

```
class CAE_Loss(_Loss):
    def __init__(self, reduction='sum'):
        super(CAE_Loss, self).__init__()
        self.MSE_Loss=torch.nn.MSELoss(reduce=True, size_average=False)

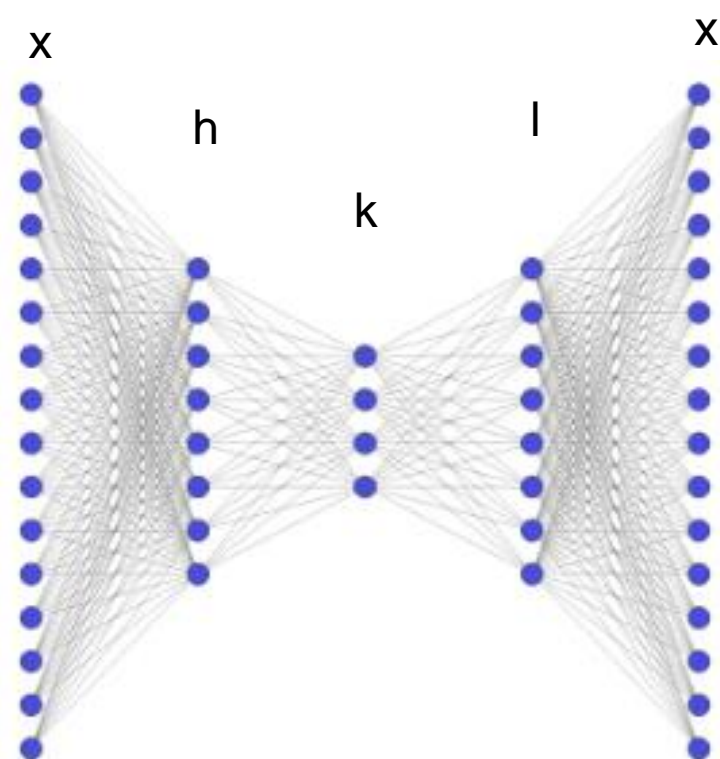
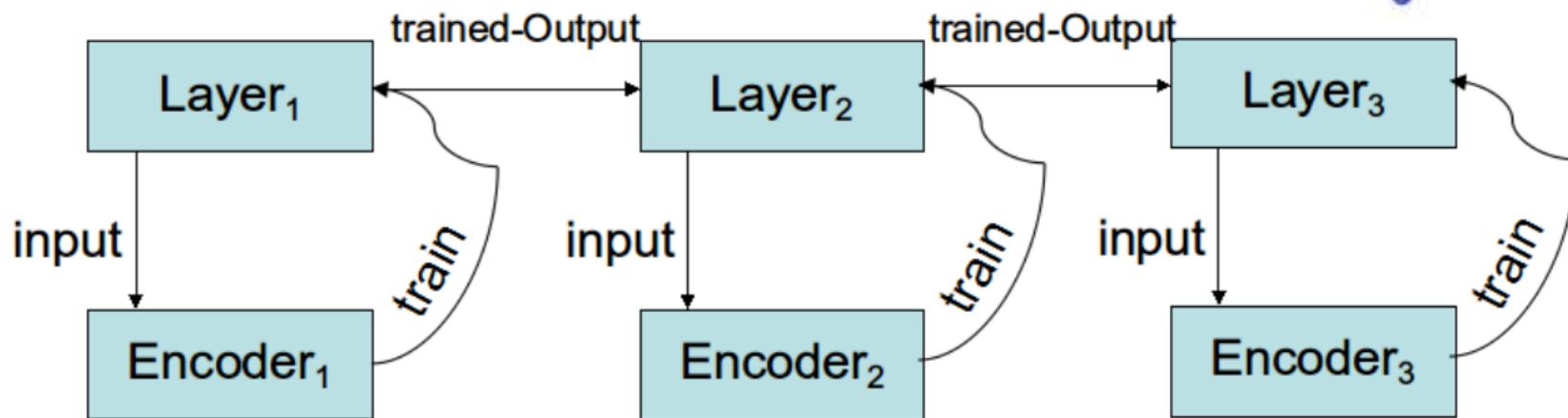
    def forward(self, input, target, encoder, gamma):
        loss1 = self.MSE_Loss(input, target)
        loss2 = self.Jacobian(target, encoder)
        loss = loss1 + gamma * loss2
        return loss

    def Jacobian(self, x, encoder):
        J = functional.jacobian(encoder, x, create_graph = True)
        J = sum([par.norm()**2 for par in J])
        return J

# Test by autoencoder1
x = torch.randn(1,1,28,28)
x = torch.flatten(x, start_dim=1)
model = autoencoder1(28*28,64)
Loss = CAE_Loss()
en, de = model(x)
loss = Loss(de,x,model.encoder,0.05)
loss.backward()
```


堆叠自编码器(SAE)

- 基本想法：假设一个含有三层隐藏层的自编码器的结构为 $x \rightarrow h \rightarrow k \rightarrow l \rightarrow x'$ ，其中 $x \rightarrow h \rightarrow k$ 是编码的过程， $k \rightarrow l \rightarrow x'$ 是解码的过程。那么可以把这个自编码器转化为先进行 $x \rightarrow h \rightarrow x'$ ，再进行 $h \rightarrow k \rightarrow h'$ ，这样通过训练两个单层的自编码器就实现了 $x \rightarrow h \rightarrow k$ ，可以得到需要的特征。



实施“无监督逐层贪婪预训练”，也称为深度自动编码器DeepAutoEncoder。参考代码[demo_SAE.ipynb](#)





Generative Adversarial Nets

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

Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

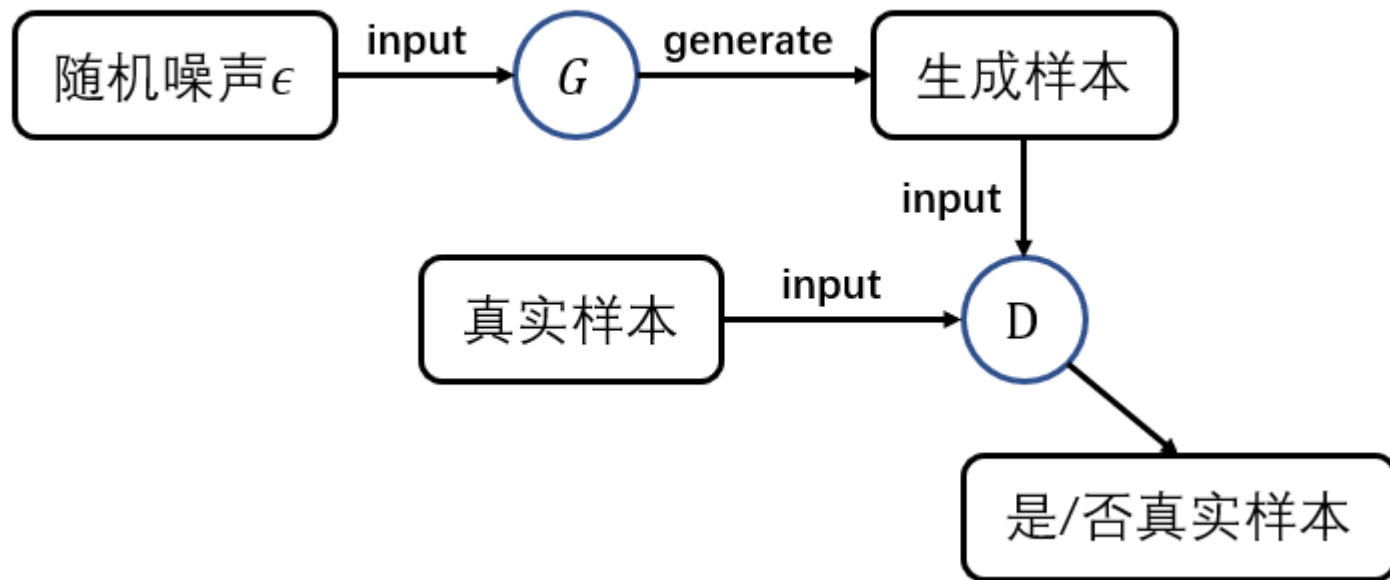
We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than 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 $\frac{1}{2}$ 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.

"Generative Adversarial Networks." Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. ArXiv 2014.

<https://github.com/goodfeli/adversarial>

GAN结构概览

生成对抗网络包含了2个子网络：生成网络(Generator, G)和判别网络(Discriminator, D)

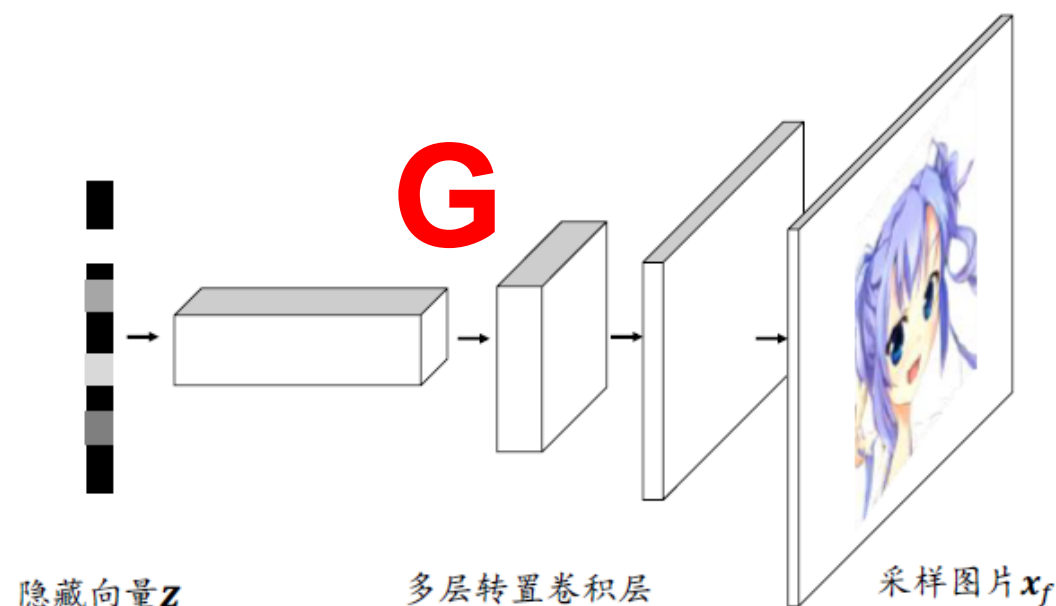
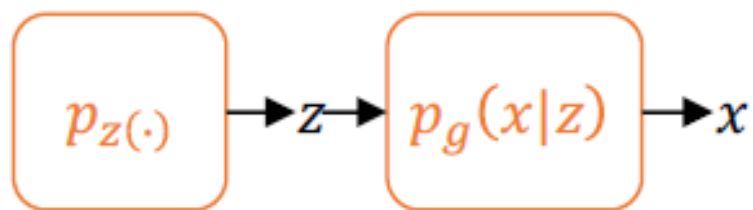


- 生成网络G负责学习样本的真实分布
- 判别网络D负责将生成网络采样的样本与真实样本区分开来



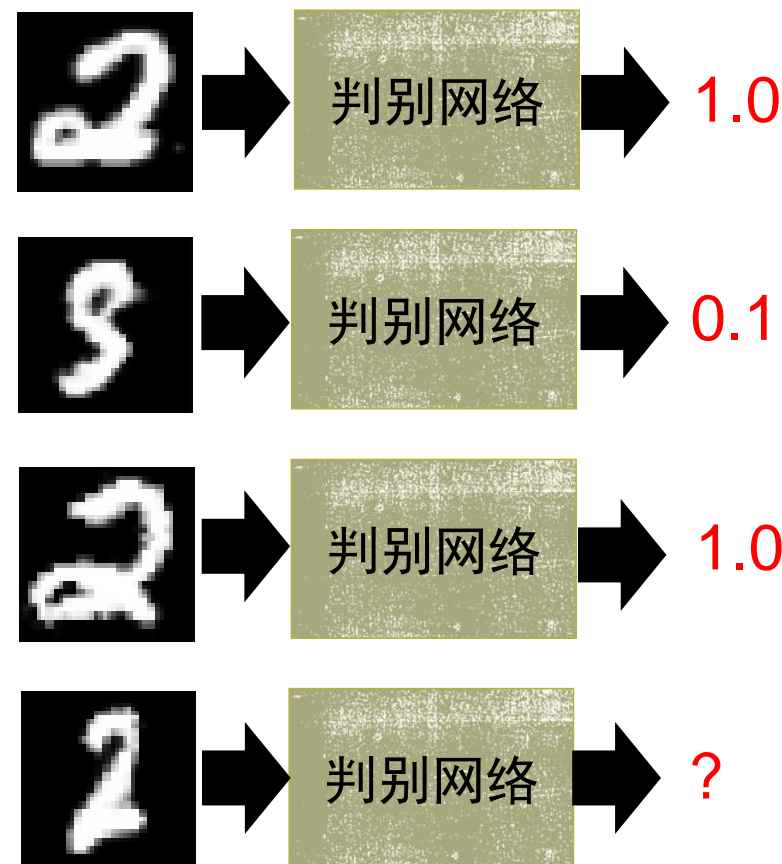
生成网络(**G**enerator)

- 与自编码器的Decoder功能类似，
从先验分布 $p_z(\cdot)$ 中采样隐藏变量
 $x \sim p_z(\cdot)$ ：通过**生成网络G**参数化的
 $p_g(x|z)$ 分布，获得生成样本



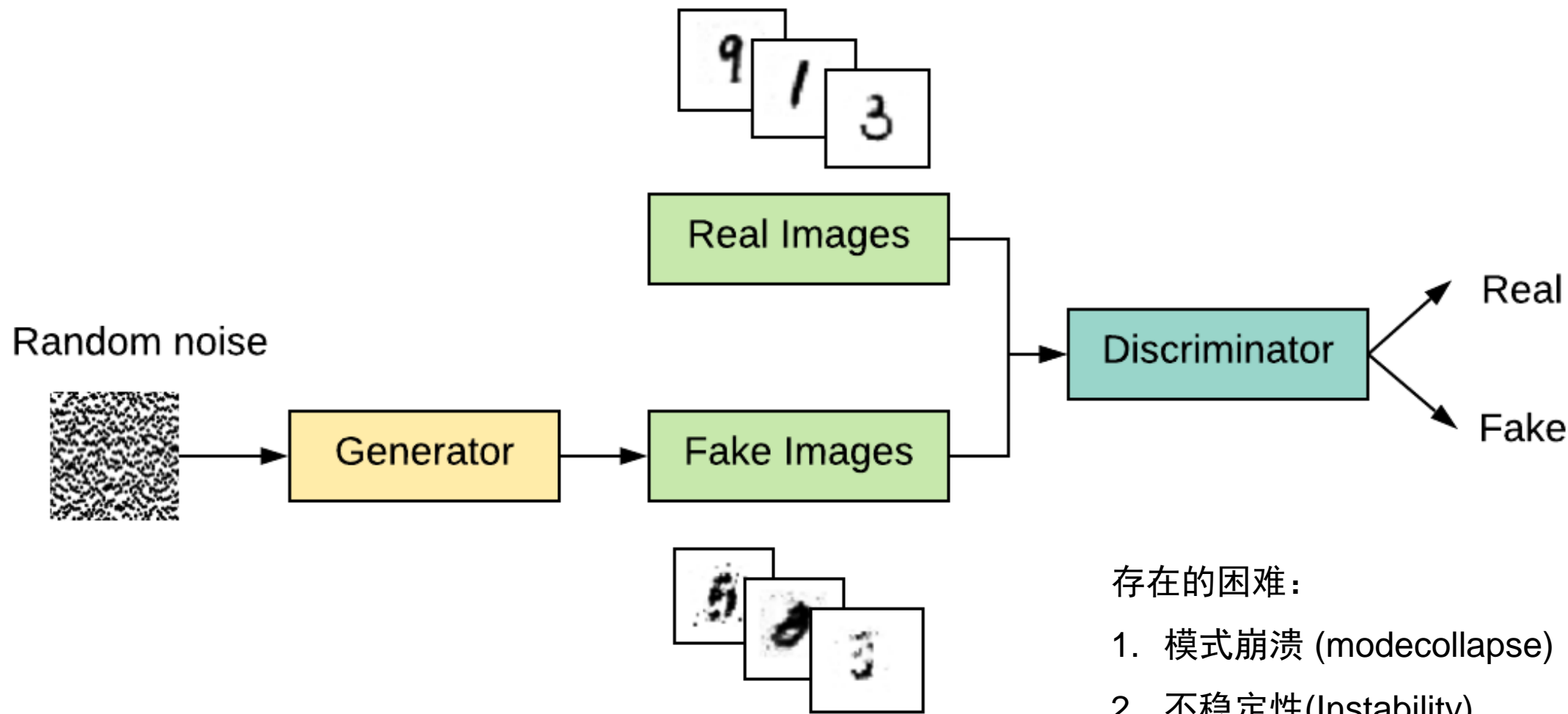
判别网络(Discriminator)

- 本质上是一个二分类网络：接受输入样本 x ，包含了采样自真实数据分布 $p_r(\cdot)$ 的样本 $x_r \sim p_r(\cdot)$ ，也包含了采样自生成网络的假样本 $x_f \sim p_g(x|z)$ ， x_r 和 x_f 共同组成了判别网络的训练集
 - 把所有真实样本 x_r 的标签标注为真(1)
 - 所有生成网络产生的样本 x_f 标注为假(0)
- 通过最小化判别网络预测值与标签之间的误差来优化判别网络参数. 即所谓：判别网络的输入则为真实样本或生成网络的输出，其目的是将生成网络的输出从真实样本中尽可能分辨出来。



1. 手写数字生成器

参考 [demo_gan_mnist.ipynb](#)



存在的困难:

1. 模式崩溃 (modecollapse)
2. 不稳定性 (Instability)



MinMax Game

- 对抗训练

- 生成网络: 尽可能地欺骗判别网络。

$$\max_{\theta} \left(\mathbb{E}_{z \sim p(z)} \left[\log D(G(z; \theta); \phi) \right] \right)$$

- 判别网络: 将生成网络生成的样本与真实样本中尽可能区分出来。

$$\max_{\phi} \mathbb{E}_{x \sim p_r(x)} \left[\log D(x; \phi) \right] + \mathbb{E}_{z \sim p(z)} \left[\log(1 - D(G(z; \theta); \phi)) \right]$$

两个网络不断调整参数，最终目的是使判别网络无法判断生成网络的输出结果是否真实。

- Minimax Game:

$$\min_{\theta} \max_{\phi} \left(\mathbb{E}_{x \sim p_r(x)} \left[\log D(x; \phi) \right] + \mathbb{E}_{z \sim p(z)} \left[\log(1 - D(G(z; \theta); \phi)) \right] \right)$$



Game理论

Proposition 1. *For G fixed, the optimal discriminator D is*

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

Proof. The training criterion for the discriminator D , given any generator G , is to maximize the quantity $V(G, D)$

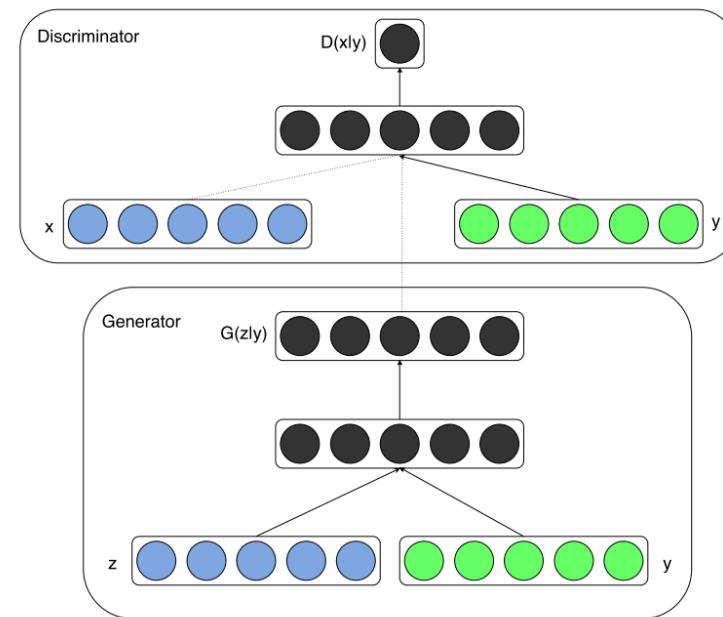
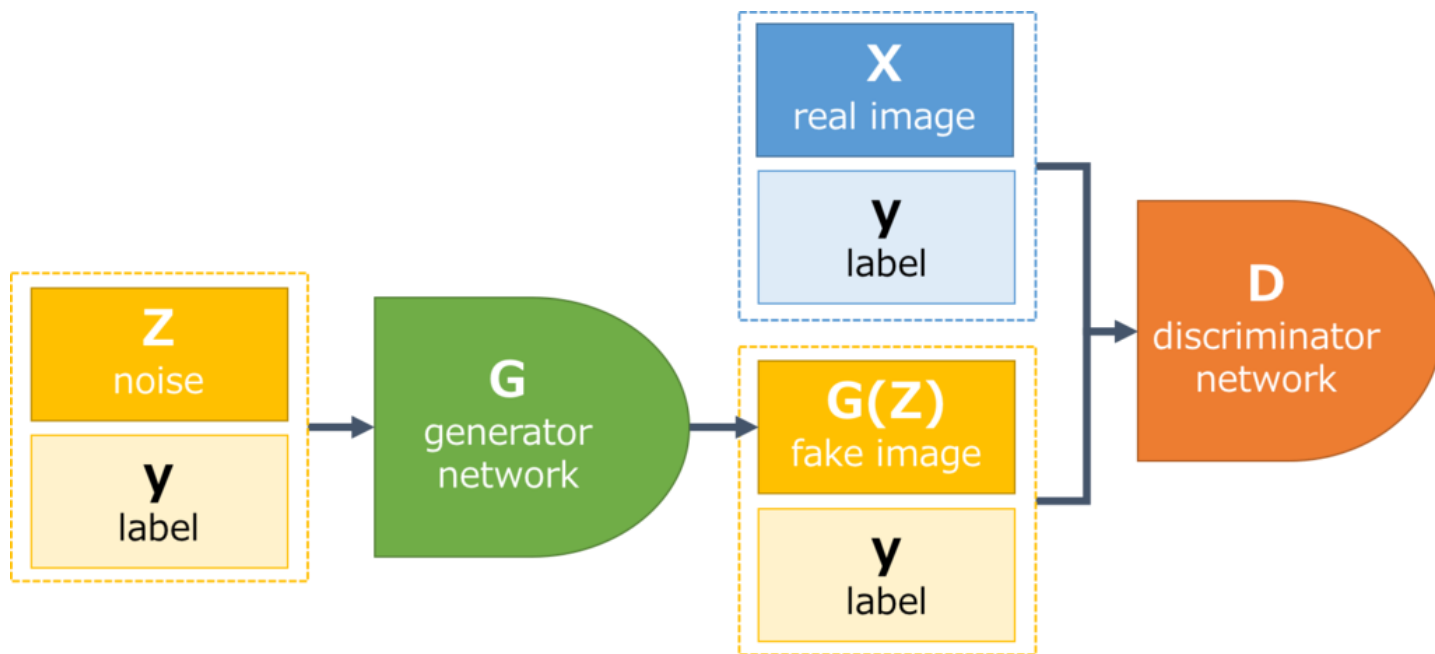
$$\begin{aligned} V(G, D) &= \int_{\mathbf{x}} p_{data}(\mathbf{x}) \log(D(\mathbf{x})) d\mathbf{x} + \int_{\mathbf{z}} p_z(\mathbf{z}) \log(1 - D(g(\mathbf{z}))) d\mathbf{z} \\ &= \int_{\mathbf{x}} p_{data}(\mathbf{x}) \log(D(\mathbf{x})) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x})) d\mathbf{x} \end{aligned} \quad (3)$$

For any $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$, the function $y \rightarrow a \log(y) + b \log(1 - y)$ achieves its maximum in $[0, 1]$ at $\frac{a}{a+b}$. The discriminator does not need to be defined outside of $Supp(p_{data}) \cup Supp(p_g)$, concluding the proof. \square

Theorem 1. *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$.*



2. Conditional GAN



- [demo_cgan_mnist.ipynb](#)
- 需要训练很长时间!
- CGANs 结果展示参考: [CGANmnist.gif](#)



潜在应用：条件生成

- 根据条件针对性的生成数据



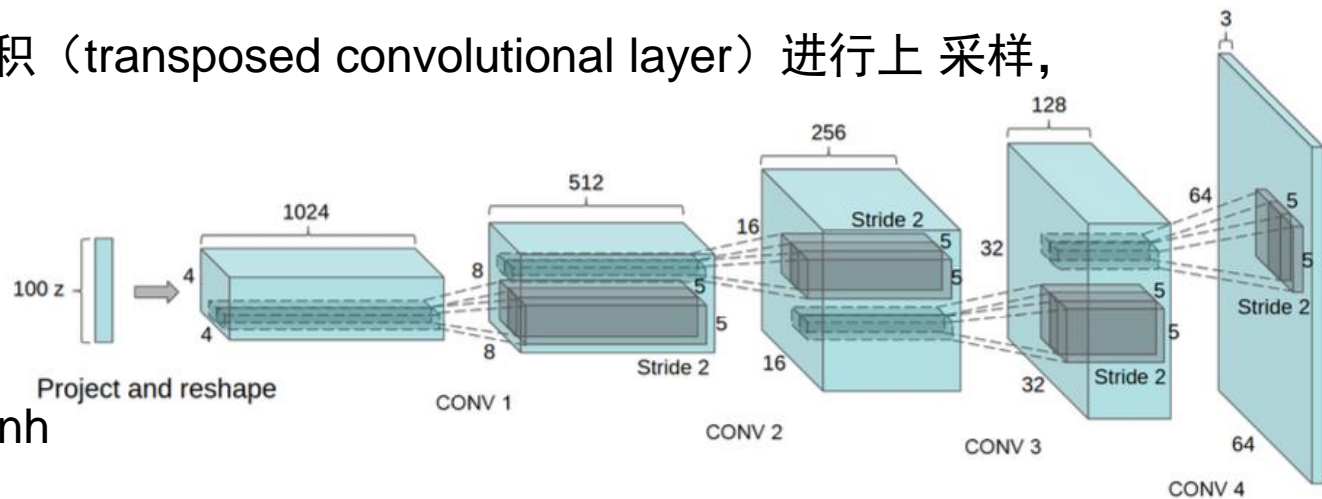
“Girl with red hair and red eyes”

“Girl with yellow ribbon”



3. 卷积GAN

- 针对图像生成应用，用CNN代替MLP，逻辑不变，DCGAN
- 对卷积神经网络的结构做了一些改变，以提高样本的质量和收敛的速度
 - 取消所有pooling层。G网络中使用转置卷积（transposed convolutional layer）进行上采样，D网络中用加入stride的卷积代替pooling。
 - 在D和G中均使用batch normalization
 - 去掉FC层，使网络变为全卷积网络
 - G网络中用ReLU为激活函数，最后一层tanh
 - D网络中使用LeakyReLU作为激活函数



- 参考论文：Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, Arxiv:1511.06434 .



卷积GAN案例

1. 参考gen_face/demo_genface.ipynb



<https://zhuanlan.zhihu.com/p/24767059>

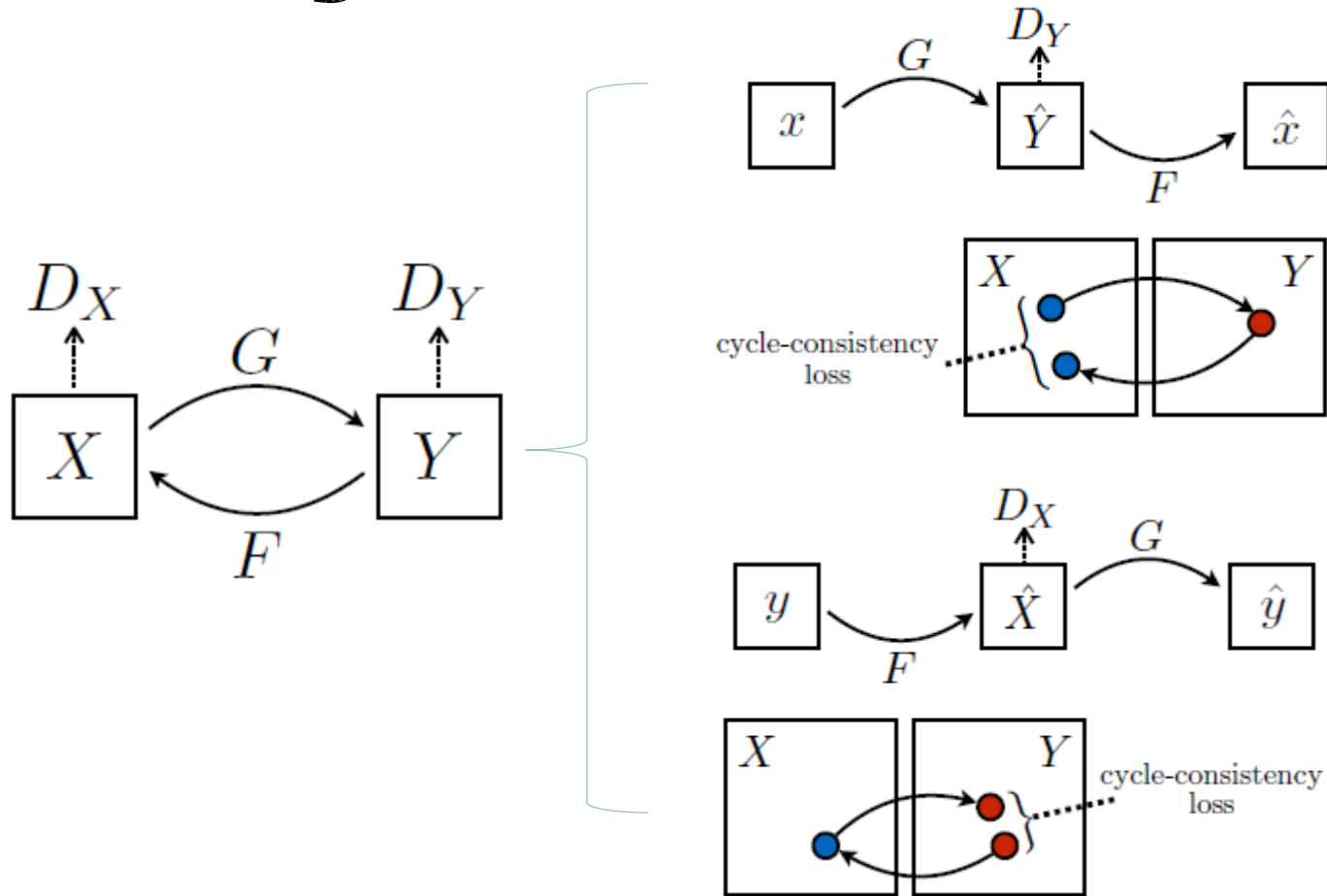


4. StyleGAN

- 实用模型简介: [stylegan2/demo_stylegan.ipynb](#)
- 近日, 一项专注于基于样式的生成模型的性能优化的研究引发了大家的关注。该研究分析了 StyleGAN2 中最困难的计算部分, 并对生成器网络提出了更改, 使得在边缘设备中部署基于样式的生成网络成为可能。
- 该研究提出了一种名为 MobileStyleGAN 的新架构。相比于 StyleGAN2, 该架构的参数量减少了约 71 %, 计算复杂度降低约 90 %, 并且生成质量几乎没有下降。新架构大大减少了基于样式 GAN 的参数量, 降低了计算复杂度。
 - 论文地址: <https://arxiv.org/pdf/2104.04767.pdf>
 - 项目地址: <https://github.com/bes-dev/MobileStyleGAN.pytorch>



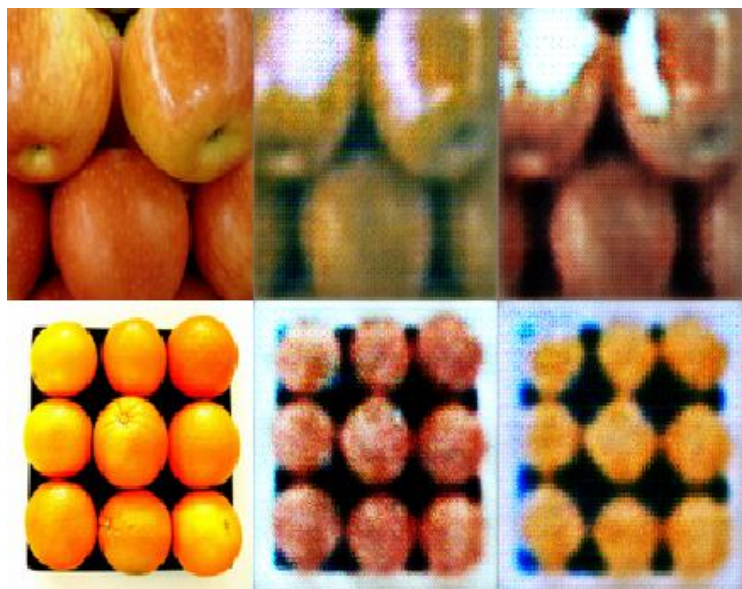
5. CycleGAN – demo in *.m



损失函数与算例(CycleGAN/CycleGAN.gif)

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = & \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ & + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ & + \lambda \mathcal{L}_{\text{cyc}}(G, F),\end{aligned}$$

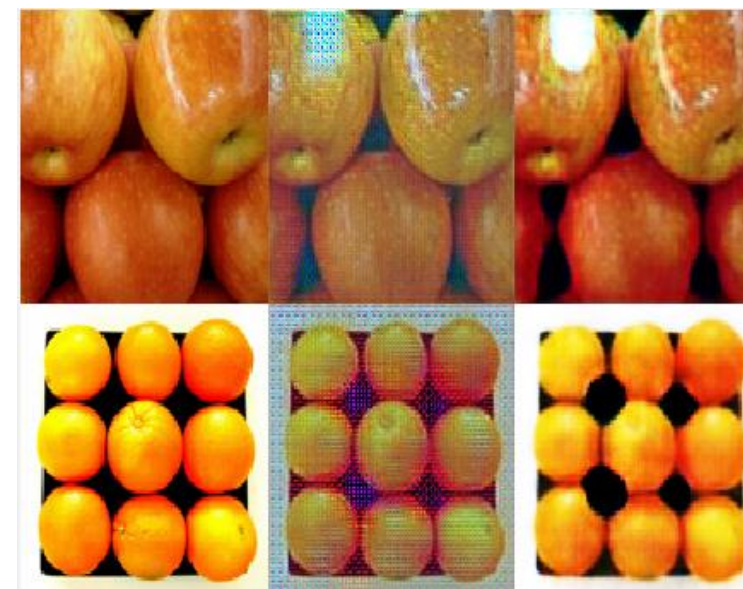
$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) = & \mathbb{E}_{x \sim p_{\text{data}}(x)} [\|F(G(x)) - x\|_1] \\ & + \mathbb{E}_{y \sim p_{\text{data}}(y)} [\|G(F(y)) - y\|_1].\end{aligned}$$



epoch 1



epoch 40



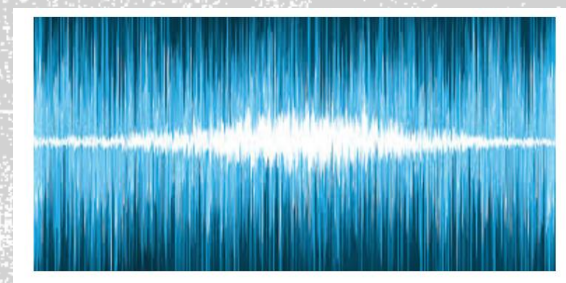
epoch 80



③ 其他应用示例

更多关于GAN的发展，社区中有很多科技工作者的贡献，
一个推荐的关于GAN论文和 代码的列表：

<https://gitcode.net/mirrors/zhangqianhui/AdversarialNetsPapers>



人脸伪造方法

- 最早的Deepfakes（2017）是基于自编码器的：

<https://github.com/deepfakes/faceswap>

- 基于GAN的换脸方法：Faceswap-GAN（2018）

<https://github.com/shaoanlu/faceswap-GAN>

- 基于扩散模型的换脸方法：DiffFace,

<https://hxngjee.github.io/DiffFace/>

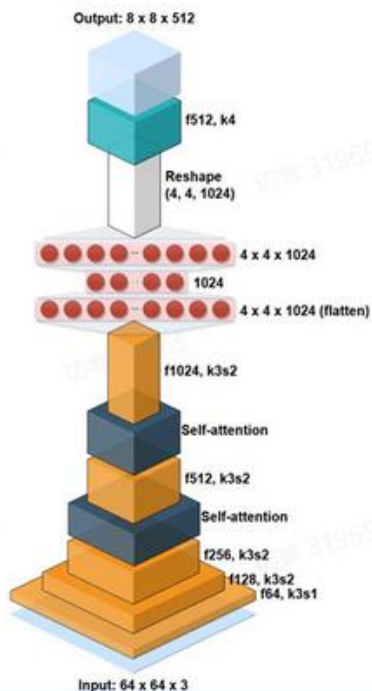


```
3 class Autoencoder(nn.Module):
4     def __init__(self):
5         super(Autoencoder, self).__init__()
6
7         self.encoder = nn.Sequential(
8             _ConvLayer(3, 128),
9             _ConvLayer(128, 256),
10            _ConvLayer(256, 512),
11            _ConvLayer(512, 1024),
12            Flatten(),
13            nn.Linear(1024 * 4 * 4, 1024),
14            nn.Linear(1024, 1024 * 4 * 4),
15            Reshape(),
16            _Upscale(1024, 512),
17        )
18
19        self.decoder_A = nn.Sequential(
20            _Upscale(512, 256),
21            _Upscale(256, 128),
22            _Upscale(128, 64),
23            Conv2d(64, 3, kernel_size=5, padding=1),
24            nn.Sigmoid(),
25        )
26
27        self.decoder_B = nn.Sequential(
28            _Upscale(512, 256),
29            _Upscale(256, 128),
30            _Upscale(128, 64),
31            Conv2d(64, 3, kernel_size=5, padding=1),
32            nn.Sigmoid(),
33        )
34
35        def forward(self, x, select='A'):
36            if select == 'A':
37                out = self.encoder(x)
38                out = self.decoder_A(out)
39            else:
40                out = self.encoder(x)
41                out = self.decoder_B(out)
42            return out
```

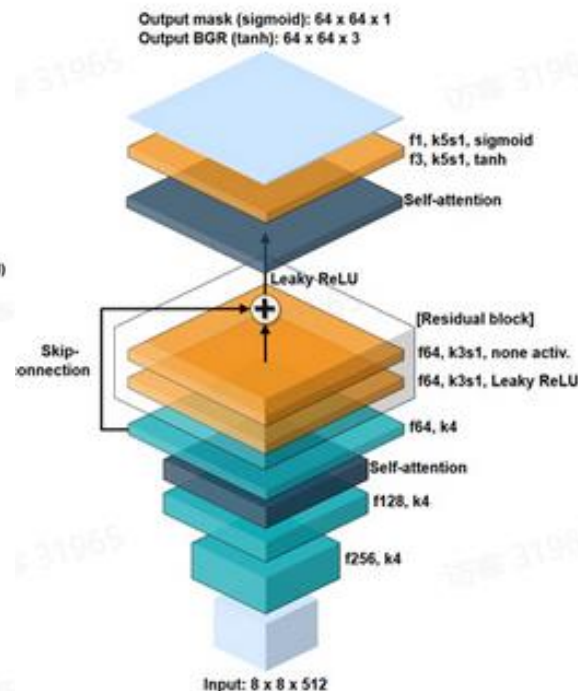
基于GAN的换脸方法

■ Faceswap-GAN (2018)

Encoder

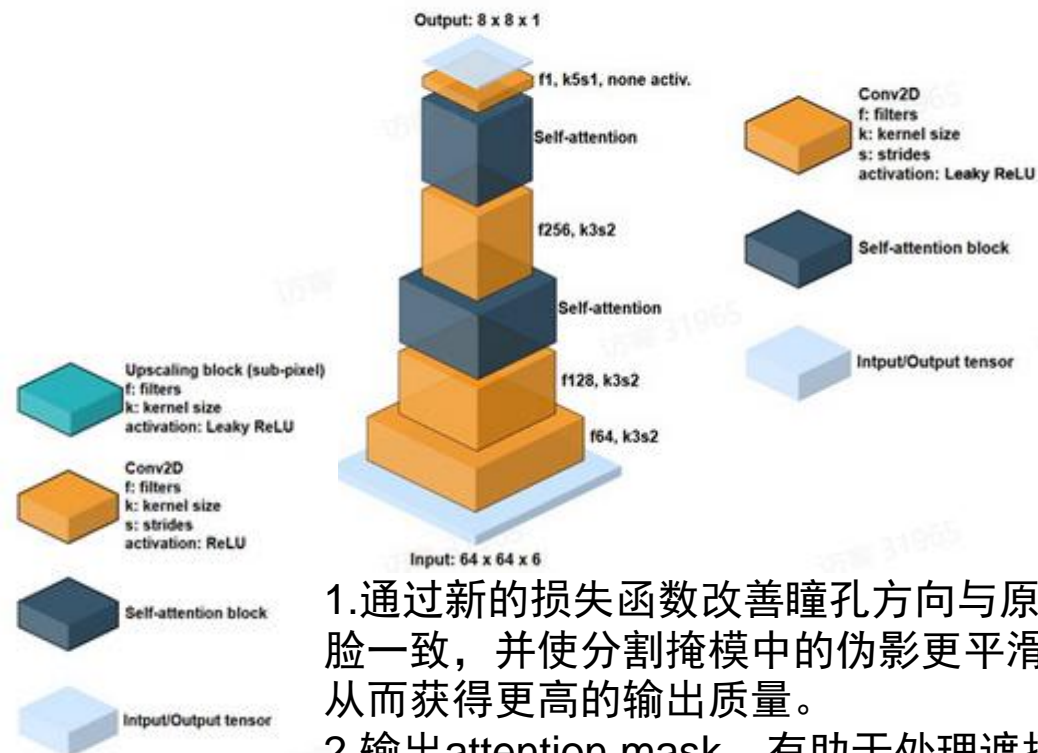


Decoder



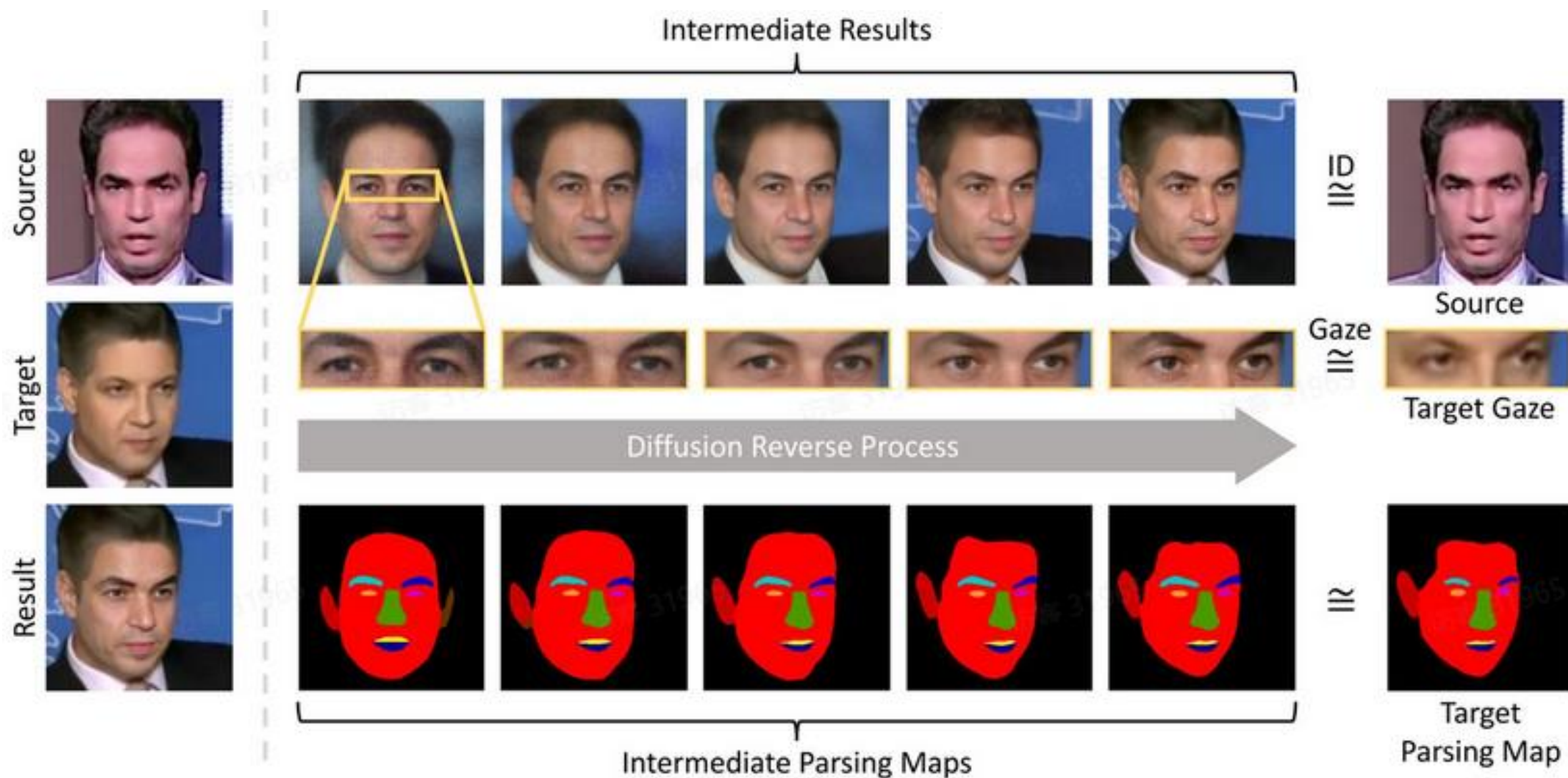
CycleGAN.....

Discriminator



- 1.通过新的损失函数改善瞳孔方向与原人脸一致，并使分割掩模中的伪影更平滑，从而获得更高的输出质量。
- 2.输出attention mask，有助于处理遮挡情况、消除伪影和产生自然肤色。
- 3.可配置的输入/输出分辨率：支持64x64、128x128和256x256输出分辨率。
- 4.使用MTCNN人脸关键点检测模型实现人脸对齐和跟踪，使用卡尔曼滤波降低视频中的抖动。
- 5.在眼睛区域引入高重建损失和边缘损失，引导模型生成逼真的眼睛。

基于扩散模型的换脸方法：DiffFace



人脸伪造检测数据集

- DFDC论文中根据提出时间、数据集规模和数据质量将目前公开的人脸伪造检测数据集分为三代：
- 第一代数据集包括：DF-TIMIT、UADFV和FaceForensics++，这一代数据集的总视频量少于1000个，视频帧数少于100万。数据集中的原视频大多来自Youtube，伪造视频由Deepfakes等方法生成，这些数据集通常没有征得视频中出现的当事人的同意。由于规模小，在FaceForensics++等数据集上训练的模型通常不能推广到真正的Deepfake视频。
- 第二代数据集包括：Google DFD、Celeb-DF、DFDC Preview Dataset。这一代数据集的视频总数少于10000个，视频帧数少于1000万，视频伪造质量比第一代更好。在这一代中，未经当事人同意而出现在数据集中的伦理问题被公开提出。
- 第三代数据集包括：DFDC、DeeperForensics-1.0。这一代数据集包含数万个视频和数千万帧。这两个数据集都使用了付费演员录制视频，解决了视频中当事人的权益问题。

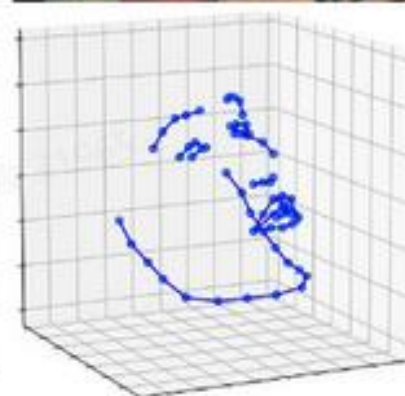
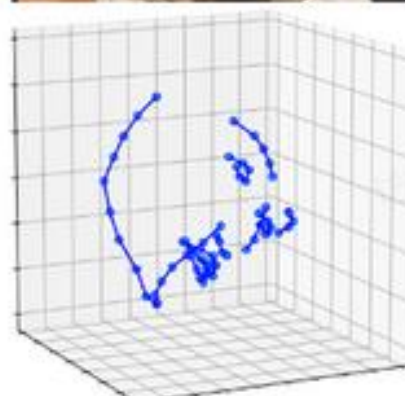
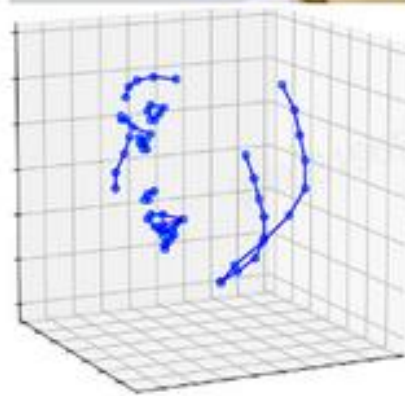
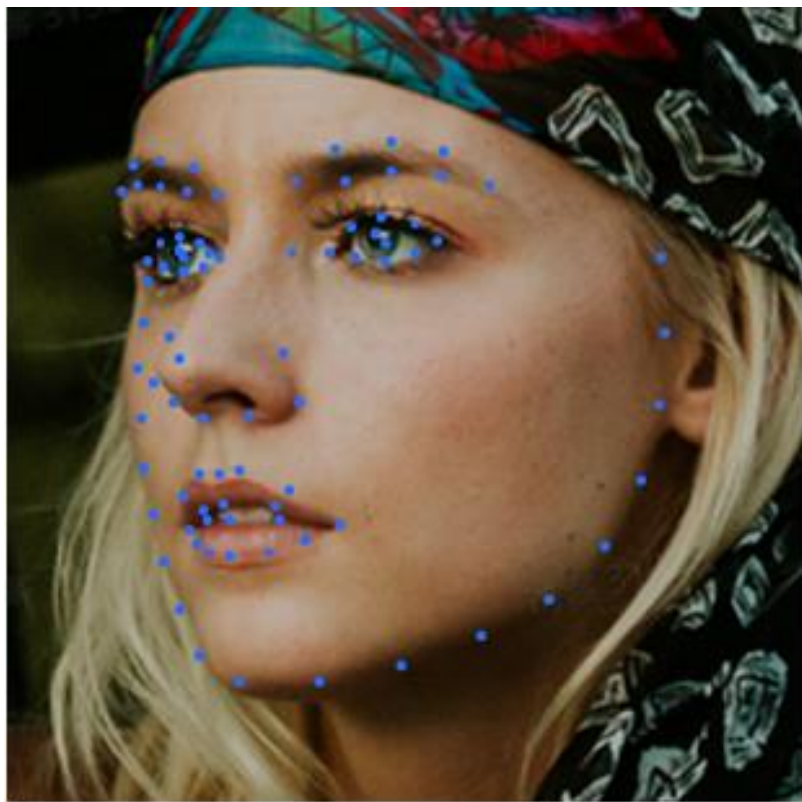


数据集	真实视频 数量	伪造视频 数量	人物数量	生成方法	获取源
DF-TIMIT ^[23]	320	320	32	faceswap-GAN	VidTIMIT 数据集
UADFV ^[24]	49	49	-	FakeAPP	网络
FaceForensics++ ^[25]	1000	1000	-	Face2Face, FaceSwap, DeepFakes, NeuralTextures	网络
DFD ^[26]	363	3068	28	DeepFakes	演员拍摄
Celeb-DF ^[27]	590	5639	59	改进的 DeepFakes	YouTube
DFDC Preview ^[28]	1131	4113	66	未公开	演员拍摄
DFDC ^[29]	3426	48190	960	DeepFakes, MM/NN face swap, NTH, FSGAN, StyleGAN	演员拍摄
DeeperForensics- 1.0 ^[30]	50000	10000	100	DF-VAE	演员拍摄



关键点位置定义和人工标注的准确性

- 被遮挡部分的ground truth如何标注？“脸部轮廓”的定义？高度镜片的折射？

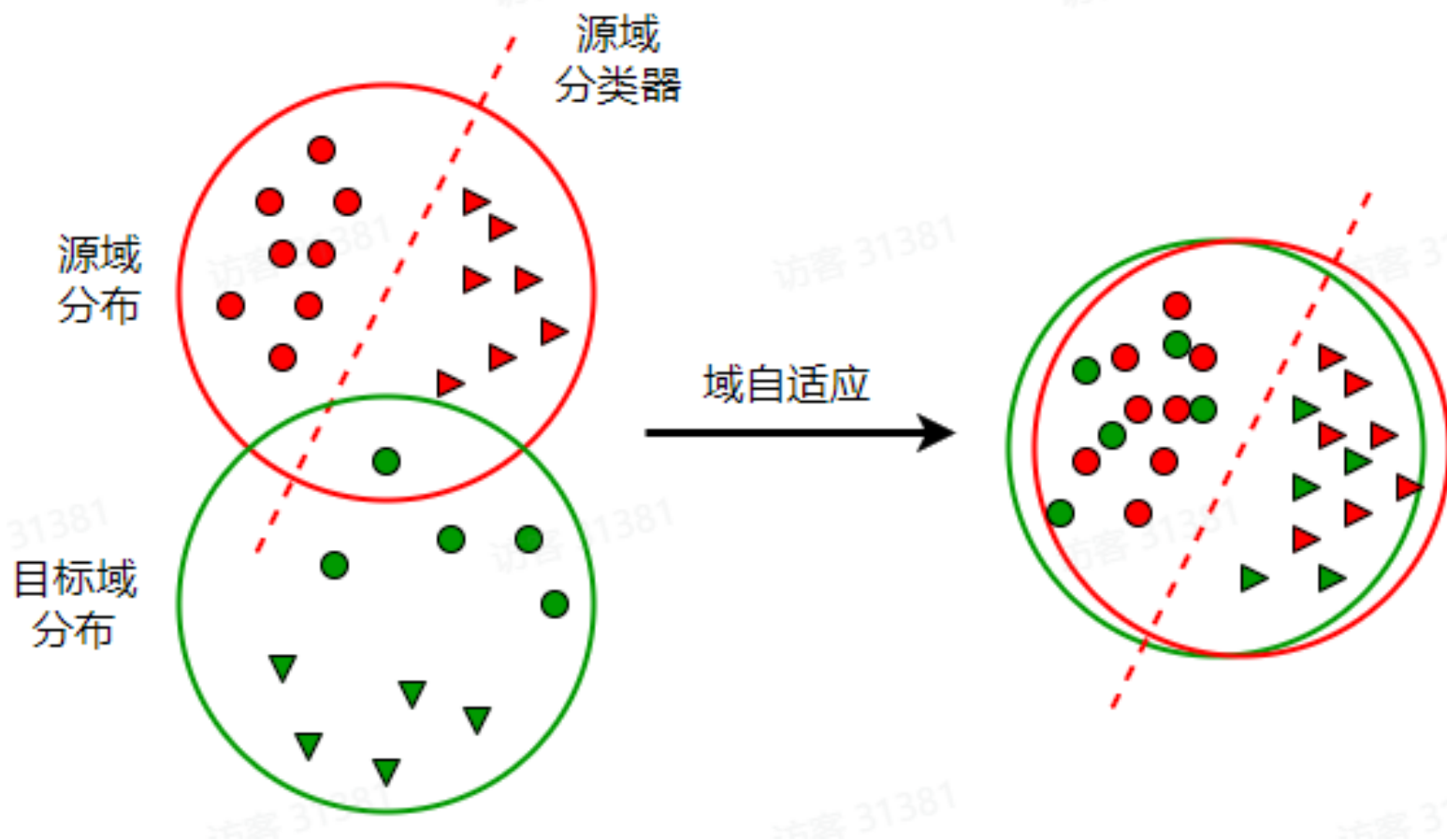


迁移学习

- 深度学习模型的训练需要 **大量数据 + 精确的标注**，获取成本极高。**迁移学习**可以被用于缓解深度学习模型的数据依赖问题，如：
 1. 预训练模型：通过预训练-微调的方式从具有大规模数据的上游任务中获得可转移的表示或模型
 2. 域自适应方法：通过减小源域和目标域之间数据分布的差异，将源域上训练的模型用到目标域
- 利用数据、任务或模型之间的相似性，旨在将旧领域学习过的模型和知识应用于新的领域。
- 与传统机器学习技术相比，迁移学习可以被理解为一种新的学习范式，用于解决现实需求：
 1. 应用场景数据量少
 2. 模型需要强鲁棒性
 3. 个性化和定制问题
 4. 保护用户隐私和数据安全等

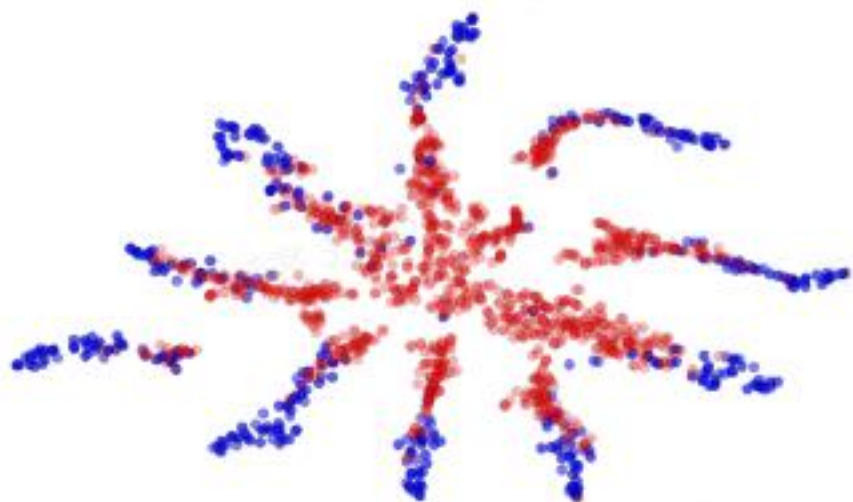


域自适应

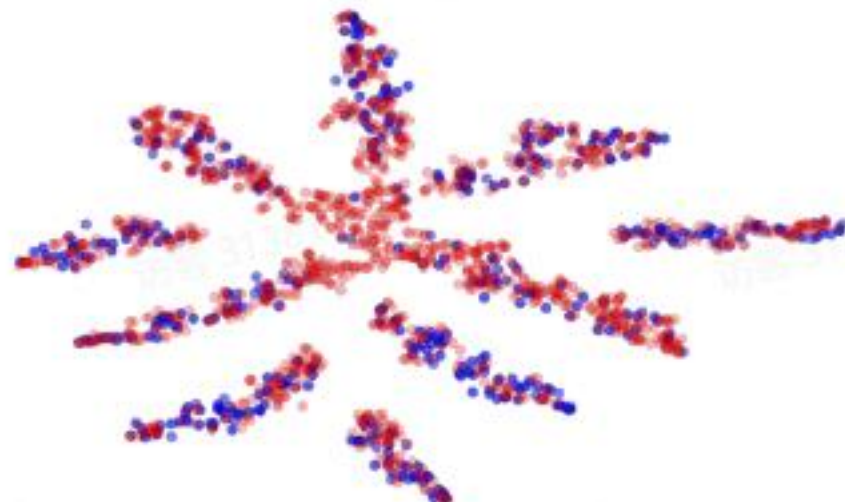


真实案例：车牌数字分类

SYN NUMBERS \rightarrow SVHN: last hidden layer of the label predictor



(a) Non-adapted



(b) Adapted

Jiang, Junguang, Yang Shu, Jianmin Wang, and Mingsheng Long.
2022. "Transferability in Deep Learning: A Survey." arXiv.



理论依据 - 1

- 建立源域（Source）分类误差和目标域（Target）分类误差的联系，利用源域分类误差和分布差异确定目标域误差上界。

Theorem 3 (Bound with Disparity) Assume that the loss function ℓ is symmetric and obeys the triangle inequality. Define the disparity between any two hypotheses h and h' on distribution \mathcal{D} as

$$\epsilon_{\mathcal{D}}(h, h') = \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}}[\ell(h(\mathbf{x}), h'(\mathbf{x}))]. \quad (14)$$

Then the target risk $\epsilon_{\mathcal{T}}(h)$ can be bounded by

$$\epsilon_{\mathcal{T}}(h) \leq \epsilon_{\mathcal{S}}(h) + [\epsilon_{\mathcal{S}}(h^*) + \epsilon_{\mathcal{T}}(h^*)] + |\epsilon_{\mathcal{S}}(h, h^*) - \epsilon_{\mathcal{T}}(h, h^*)|, \quad (15)$$

where $h^* = \arg \min_{h \in \mathcal{H}} [\epsilon_{\mathcal{S}}(h) + \epsilon_{\mathcal{T}}(h)]$ is the ideal joint hypothesis, $\epsilon_{ideal} = \epsilon_{\mathcal{S}}(h^*) + \epsilon_{\mathcal{T}}(h^*)$ is the ideal joint error, $|\epsilon_{\mathcal{S}}(h, h^*) - \epsilon_{\mathcal{T}}(h, h^*)|$ is the disparity difference between \mathcal{S} and \mathcal{T} .



理论依据 - 2

$\mathcal{H}\Delta\mathcal{H}$ 散度: 对 $|\epsilon_S(h, h^*) - \epsilon_T(h, h^*)|$ 中的变量 h, h^* 放缩。它可通过有限样本估计, 并用于确定目标域上界。

Definition 4 ($\mathcal{H}\Delta\mathcal{H}$ -Divergence) Define $\mathcal{H}\Delta\mathcal{H} \triangleq \{h | h = h_1 \otimes h_2, h_1, h_2 \in \mathcal{H}\}$ as the symmetric difference hypothesis space of \mathcal{H} , where \otimes stands for the XOR operator. Then the $\mathcal{H}\Delta\mathcal{H}$ -Divergence between \mathcal{S} and \mathcal{T} is

$$d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{S}, \mathcal{T}) \triangleq \sup_{h, h' \in \mathcal{H}} |\epsilon_S(h, h') - \epsilon_T(h, h')|.$$

For binary classification problem with the 01-loss, $\ell(y, y') = \mathbf{1}(y \neq y')$, we have

$$d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{S}, \mathcal{T}) = \sup_{\delta \in \mathcal{H}\Delta\mathcal{H}} |\mathbb{E}_S[\delta(\mathbf{x}) \neq 0] - \mathbb{E}_T[\delta(\mathbf{x}) \neq 0]|.$$

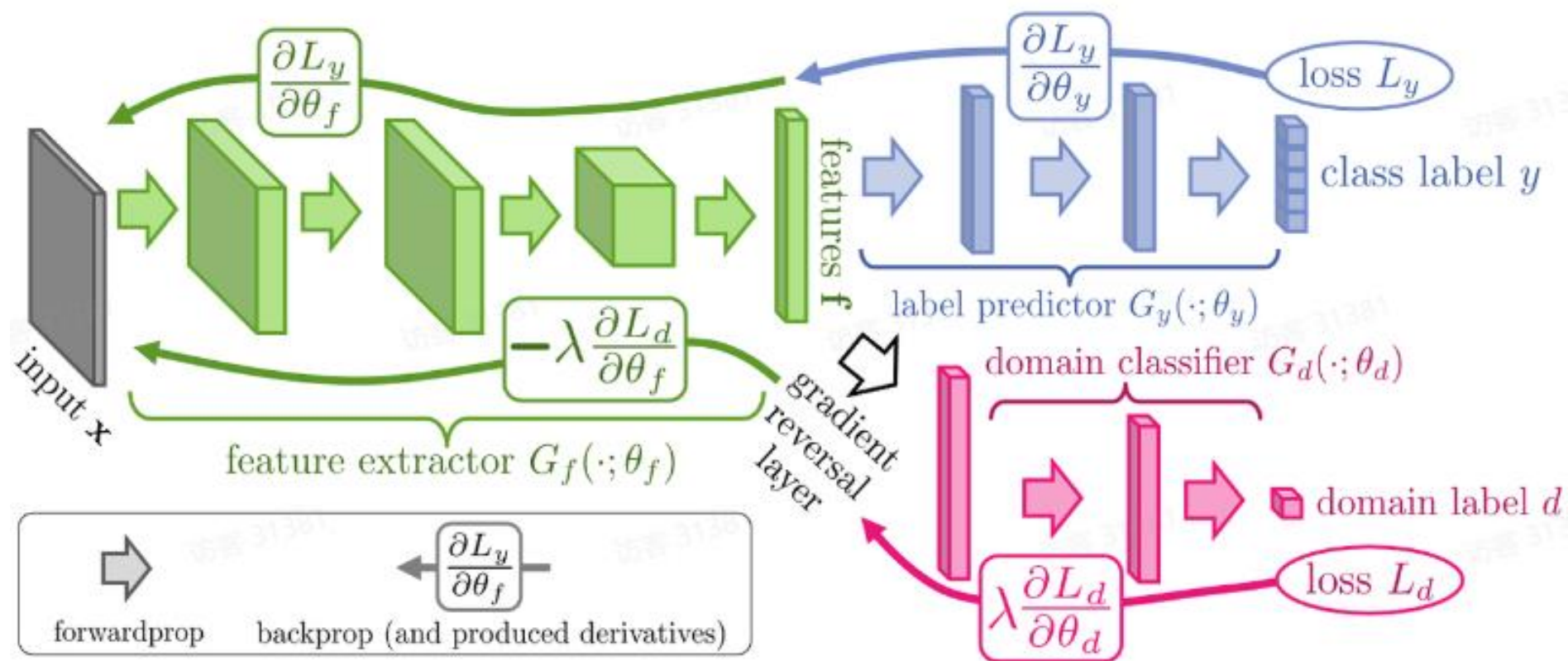
Theorem 5 (Ben-David et al. (2010a)) Let \mathcal{H} be a binary hypothesis space of VC dimension d . If $\hat{\mathcal{S}}$ and $\hat{\mathcal{T}}$ are samples of size m each, then for any $\delta \in (0, 1)$, with probability at least $1 - \delta$, for every $h \in \mathcal{H}$,

$$\epsilon_T(h) \leq \epsilon_S(h) + d_{\mathcal{H}\Delta\mathcal{H}}(\hat{\mathcal{S}}, \hat{\mathcal{T}}) + \epsilon_{ideal} + 4\sqrt{\frac{2d \log(2m) + \log(\frac{2}{\delta})}{m}}. \quad (16)$$



基于对抗的模型

- 受 $\mathcal{H} \Delta \mathcal{H}$ 散度启发的DANN：理论对算法设计的指导。



基于对抗的模型

As mentioned in Section 3.2, the upper bound of $\mathcal{H}\Delta\mathcal{H}$ -Divergence between feature distributions can be estimated by training the domain discriminator D ,

$$L_{\text{DANN}}(\psi) = \max_D \mathbb{E}_{\mathbf{x}^s \sim \hat{\mathcal{S}}} \log[D(\mathbf{z}^s)] + \mathbb{E}_{\mathbf{x}^t \sim \hat{\mathcal{T}}} \log[1 - D(\mathbf{z}^t)], \quad (28)$$

where $\mathbf{z} = \psi(\mathbf{x})$ is the feature representation for \mathbf{x} . The objective of the feature generator ψ is to minimize the source error as well as the $\mathcal{H}\Delta\mathcal{H}$ -Divergence bounded by Equation 28,

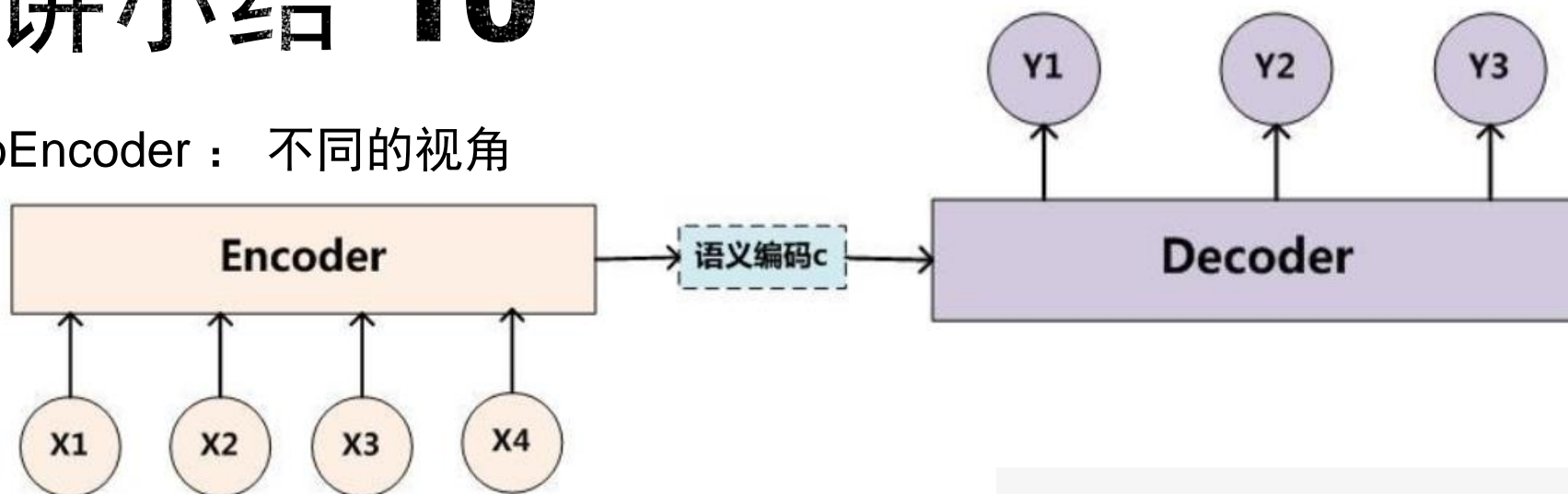
$$\min_{\psi, h} \mathbb{E}_{(\mathbf{x}^s, \mathbf{y}^s) \sim \hat{\mathcal{S}}} L_{\text{CE}}(h(\mathbf{z}^s), \mathbf{y}^s) + \lambda L_{\text{DANN}}(\psi), \quad (29)$$

where L_{CE} is the cross-entropy loss, λ is a hyper-parameter that trades off source error and domain adversary. Minimizing the cross-entropy loss will lead to discriminative representations while decreasing the domain adversarial loss will result in transferable representations.



本讲小结 10

- AutoEncoder : 不同的视角



- GAN:

1. 对抗训练机制
2. Condition GAN
3. Deep Convolutional GAN
4. Style GAN
5. Cycle GAN

