Автокодировщики и генеративно-состязательные сети

Лекция 6

План лекции

- Автокодировщики
- Архитектура Encoder-Decoder
- Генеративно-состязательные сети

Автокодировщики

- Задача извлечения признаков
  (обучение без учителя, unsupervised learning)
- Примеры:
  - представление изображений (например, цифр в MNIST)
  - представление текста
  - представление речи
  - ...
- Автокодировщики (autoencoders):
  - Rumelhart D.E., Hinton G.E., Williams R.J. Learning Internal Representations by Error Propagation // Parallel Distributed Processing. Vol 1: Foundations / Cambridge, MA, USA: MIT Press, 1986

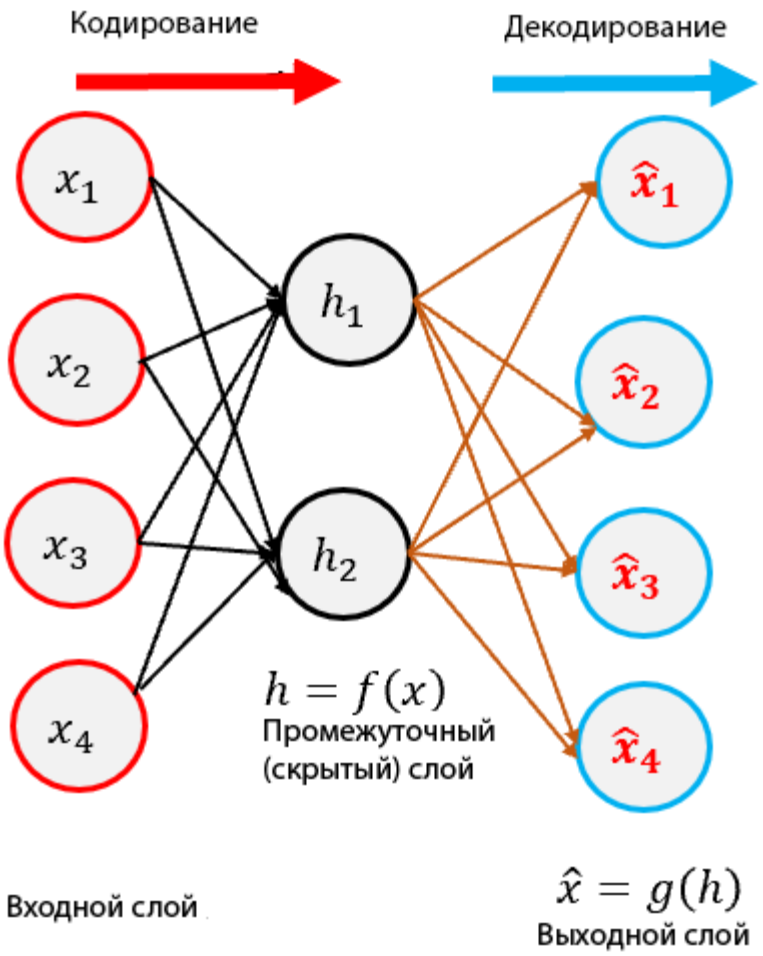
Автокодировщики

- Идея: превращение задачи обучения без учителя в задачу обучения с учителем – восстановление входа:

$$h = f(x)$$

$$\hat{x} = g(h) \approx x$$

- Простейший вариант: единичные веса и размер скрытого слоя совпадает с размером входа
  - Поэтому нужно накладывать ограничения на f и g



Автокодировщики

- Ограничение из оригинальной работы: уменьшение размерности скрытого слоя по сравнению с входом
  - MNIST: 784 входа $\rightarrow$ 100 нейронов
- Отображение пространства большей размерности в пространство меньшей размерности:

$$R^D \rightarrow R^d, \qquad D > d$$

  – значительная потеря информации?
- Допустимо, поскольку не приходится отображать все возможные объекты исходного пространства
- Это задача снижения размерности

Автокодировщики

Задача снижения размерности (dimensionality reduction):

- Отбор признаков (feature selection)
  - методы-фильтры (filters)
  - методы-обертки (wrappers)
  - встроенные методы (embedded)
- Выделение признаков (feature extraction) или проекция признаков (feature projection):
  - анализ главных компонент (principal component analysis, PCA)
  - неотрицательное матричное разложение (non-negative matrix factorization, NMF)
  - сингулярное разложение матриц (singular value decomposition, SVD)
  - линейный дискриминантный анализ (linear discriminant analysis, LDA)
  - **автокодировщики (autoencoders)**
  - стохастическое вложение соседей с t-распределением (t-distributed stochastic neighbor embedding, t-SNE)
  - uniform manifold approximation and projection, UMAP

Автокодировщики

Виды автокодировщиков:

- понижающие автокодировщики (undercomplete autoencoders)
- повышающие автокодировщики (overcomplete autoencoders)
- разреженные автокодировщики (sparse autoencoders)
- шумоподавляющие автокодировщики (denoising autoencoders)
- сжимающие автокодировщики (contractive autoencoders)
- вариационные автокодировщики (variational autoencoder, VAE)

Автокодировщики

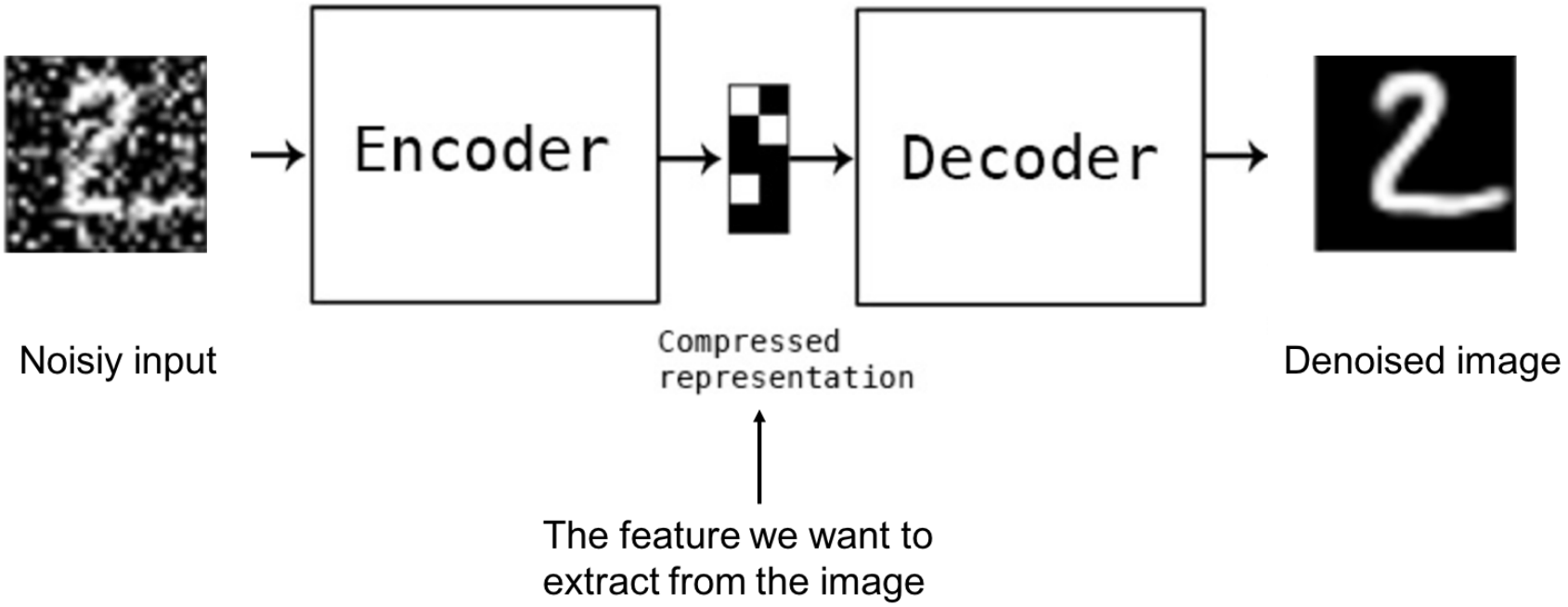
Способы регуляризации (борьба с переобучением):

- размерность скрытого слоя меньше, чем входного (undercomplete autoencoders)
- нелинейная функция активации
- дропаут
- разреженность активаций скрытых нейронов (sparse autoencoders)
  - добавить расстояние Кульбака-Лейблера между реальным и желаемым распределениями активации как регуляризатор
- восстановление зашумленного входа $\tilde{x}$ (denoising autoencoders):

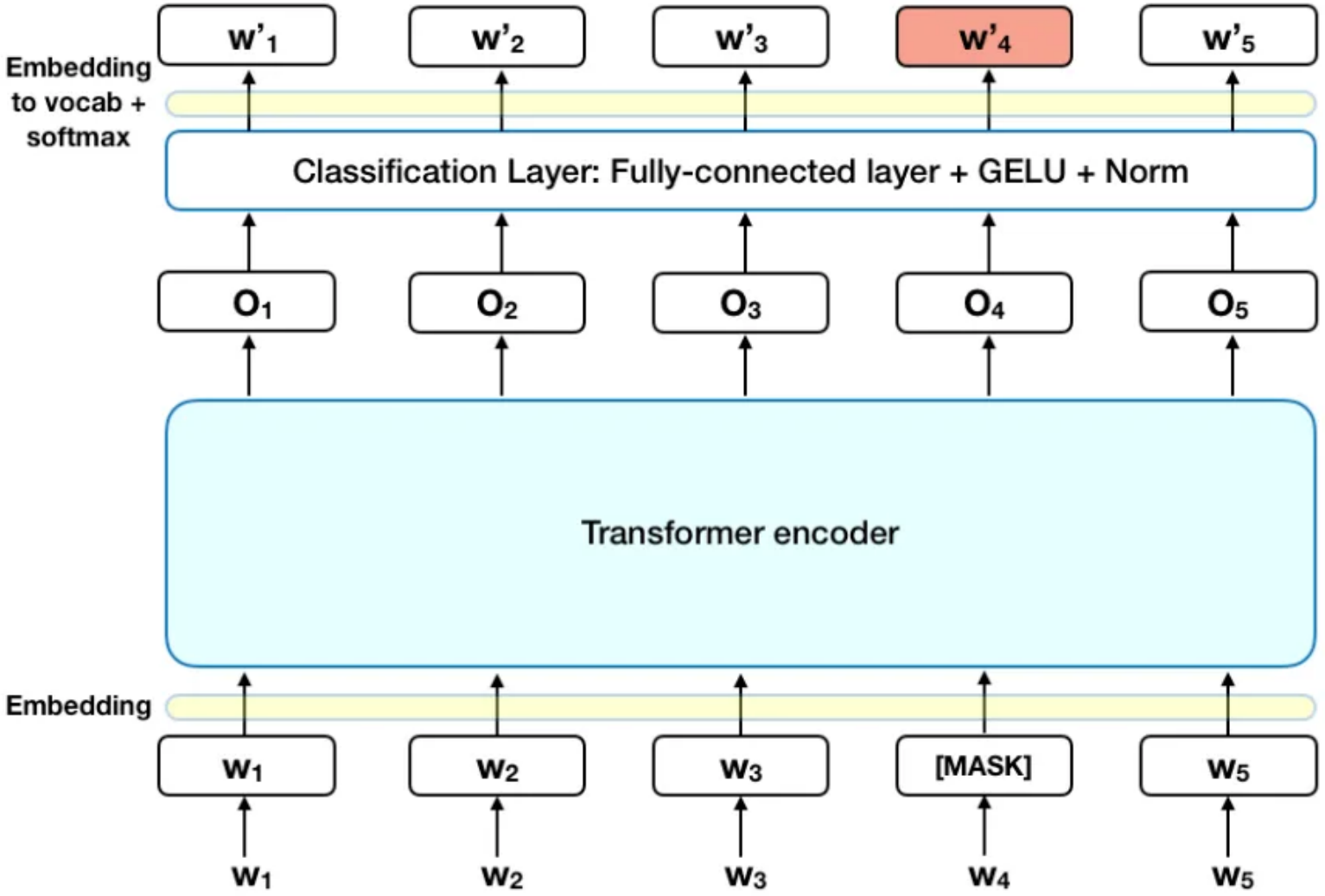
$$h = f(\tilde{x}), \qquad \hat{x} = g(h) \approx x$$

  - увеличение размера входа за счет различных преобразований

Шумоподавляющий автокодировщик



BERT – Bidirectional Encoder Representations from Transformers



См. [Николенко и др., 2018, стр. 219]

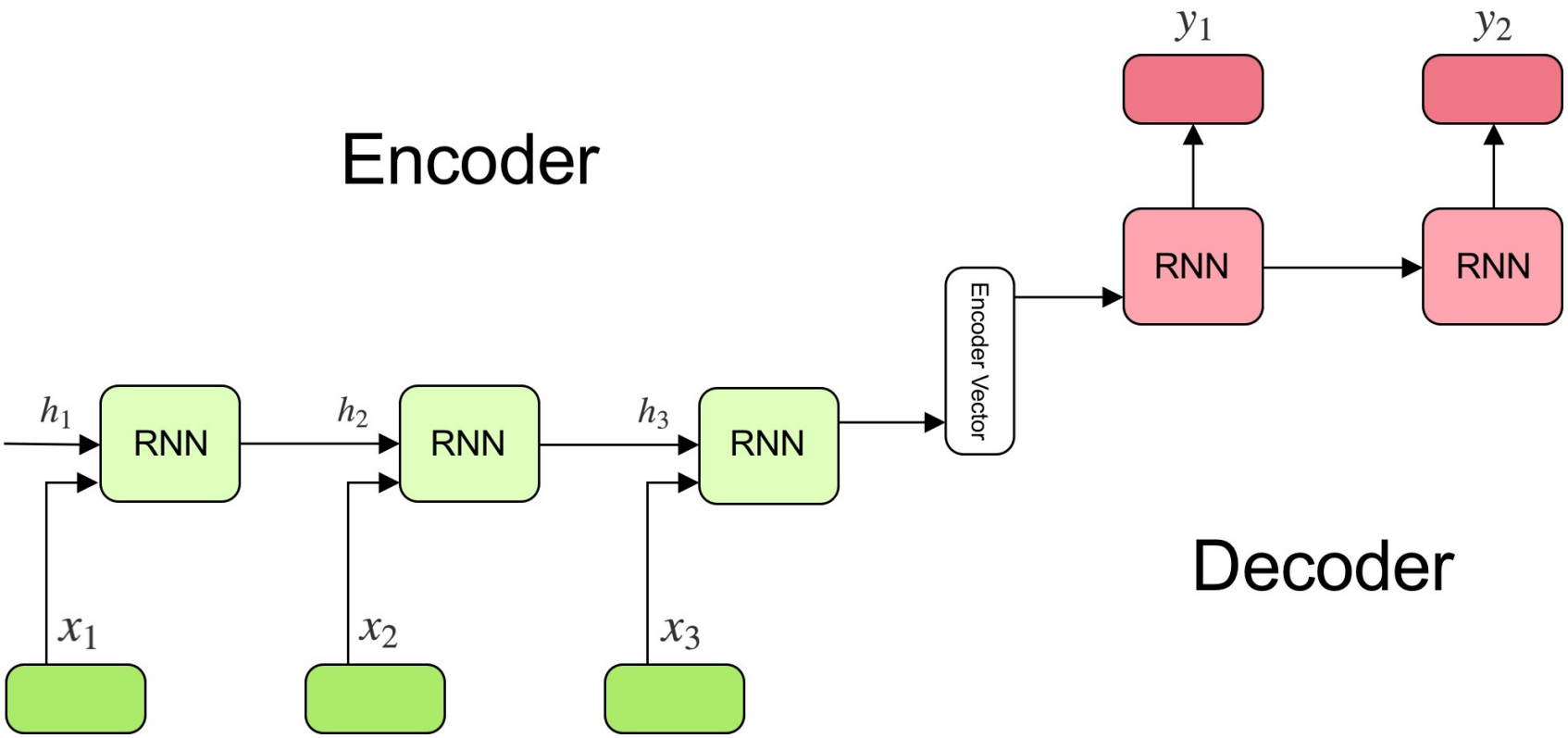
Автокодировщики

См. autoencoder.ipynb

```
class Autoencoder(nn.Module):
    def __init__(self, input_size, hidden_size):
        super().__init__()
        self.encoder_hidden_layer = nn.Linear(
            in_features=input_size, out_features=hidden_size)
        self.encoder_output_layer = nn.Linear(
            in_features=hidden_size, out_features=hidden_size)
        self.decoder_hidden_layer = nn.Linear(
            in_features=hidden_size, out_features=hidden_size)
        self.decoder_output_layer = nn.Linear(
            in_features=hidden_size, out_features=input_size)

    def forward(self, input):
        activation = self.encoder_hidden_layer(input)
        activation = torch.relu(activation)
        code = self.encoder_output_layer(activation)
        code = torch.relu(code)
        activation = self.decoder_hidden_layer(code)
        activation = torch.relu(activation)
        activation = self.decoder_output_layer(activation)
        reconstructed = torch.relu(activation)
        return reconstructed, code
```

Архитектура Encoder-Decoder



- Машинный перевод
- Диалоговые системы
- …

Генеративно-состязательные сети

- Генеративно-состязательные сети
  (Generative Adversarial Networks, GANs)
  - Goodfellow Ian et al. Generative Adversarial Networks // NIPS-2014
- Состоят из двух соревнующихся сетей:
  - *генератор* – порождает в пространстве данных объекты с распределением $p_z(z)$, стараясь воспроизвести распределение обучающей выборки $p_{data}(x)$ и обмануть дискриминатор
  - *дискриминатор* – учится отличать порожденные генератором объекты от объектов обучающей выборки
- Цель – научить генератор распределению обучающей выборки:

$$p_z(z) \approx p_{data}(x)$$

Генеративно-состязательные сети

- Как нейронная сеть (генератор) может порождать примеры?
  - На вход подаются случайные значения из некоторого распределения $p_z(z)$ (нормального или равномерного), которые генератор учится преобразовывать в обучающее распределение:

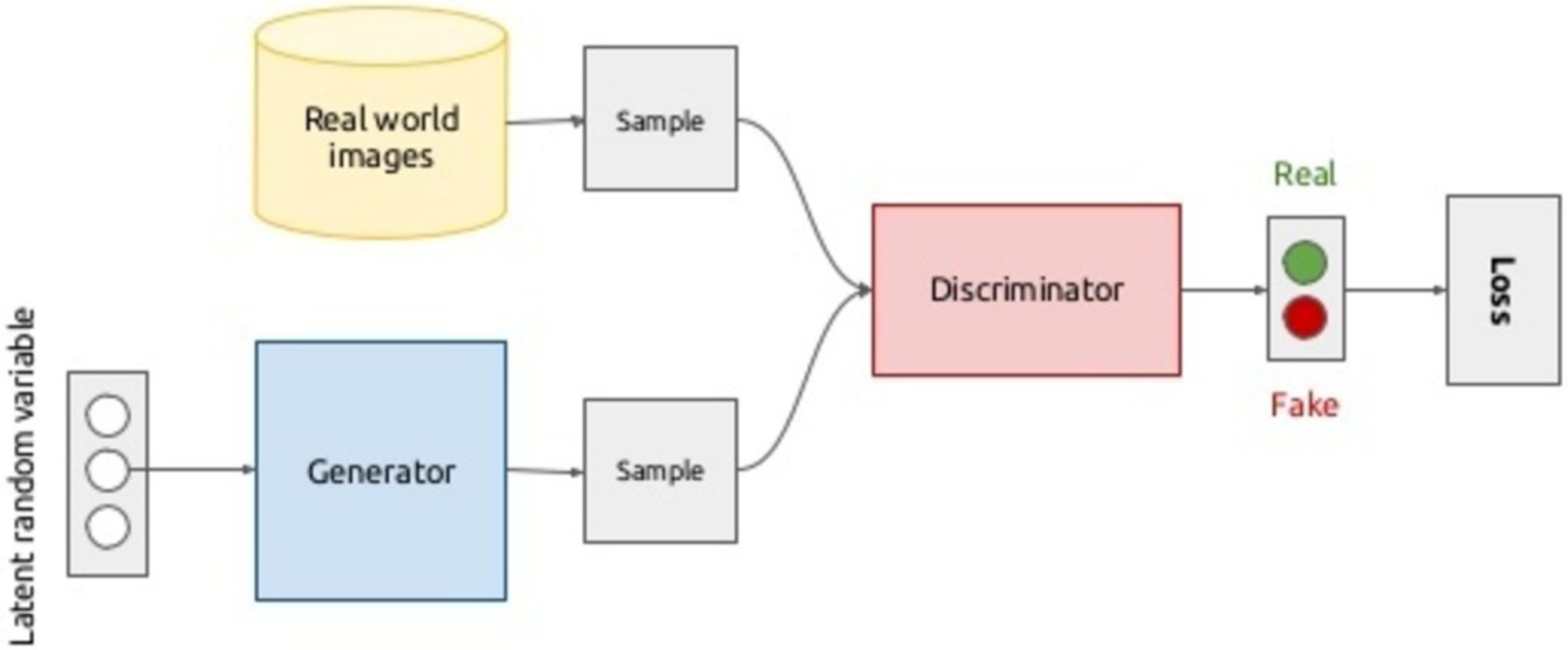
$$G = G(z, w_g): Z \to X$$

где Z – пространство скрытых факторов с заданным априорным распределением; w_g – параметры генератора

- Дискриминатор:

$$D = D(x, w_d): X \to [0, 1]$$

Генеративно-состязательные сети



https://poloclub.github.io/ganlab/

Генеративно-состязательные сети

Целевые функции

- Дискриминатор:

$$\mathop{\mathbb{E}}_{x\sim p_{data}(x)}[\log D(x)] + \mathop{\mathbb{E}}_{z\sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right] \to \max$$

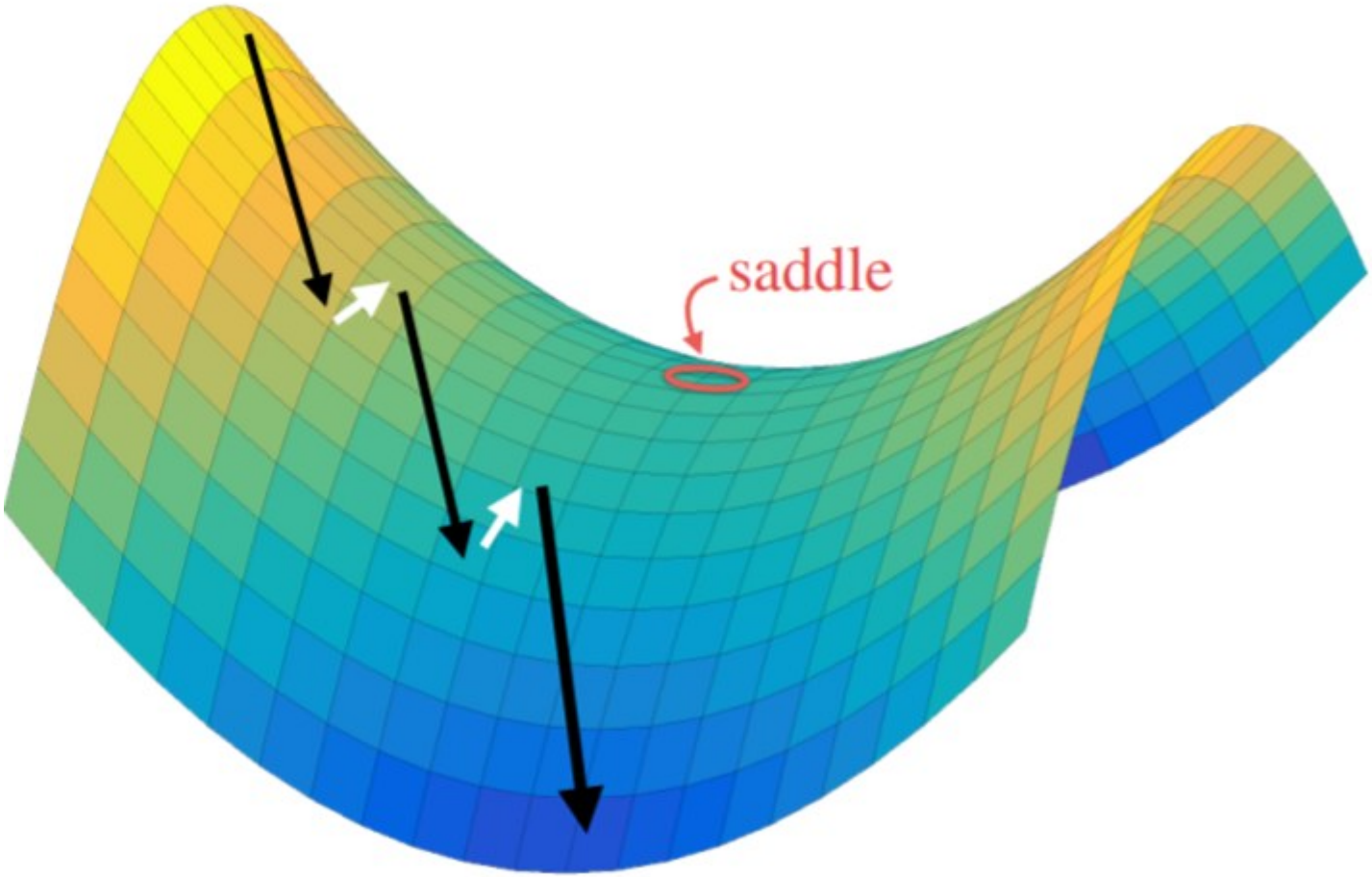
- Генератор:

$$\mathop{\mathbb{E}}_{z\sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right] \to \min$$

- Итоговая функция (*минимаксная игра*):

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

Генеративно-состязательные сети



Генеративно-состязательные сети

- Дискриминатор должен быть мощнее генератора
- Процесс обучения:
  - Один шаг обновления весов генератора
  - k шагов обновления весов дискриминатора
- Оптимальное решение дискриминатора (при фиксированном генераторе):

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

  - если x встречается только в реальных данных, то $D_G^*(x) = 1$
  - если x встречается только в сгенерированных данных, то $D_G^*(x) = 0$
  - если распределения совпадают ($p_{data} = p_z$), то $D_G^*(x) = 0.5$

Генеративно-состязательные сети

Процесс обучения:

// num_iteration – число итераций обучения
function GAN:
  for i = 1..num_iteration do
    for j = 1..k do
      //Получаем мини-батч $\{z_1, \ldots, z_m\}$ из распределения p_z
      z = getBatchFromNoisePrior(p_z)
      //Получаем мини-батч $\{x_1, \ldots, x_m\}$ из распределения p_{data}
      x = getBatchFromDataGeneratingDistribution(p_{data})
      //Обновляем дискриминатор в сторону возрастания его градиента

$$d_w \leftarrow \nabla_{\gamma_d} \frac{1}{m} \sum_{t=1}^{m} [logD(x_t)] + [log(1 - D(G(z_t))]$$

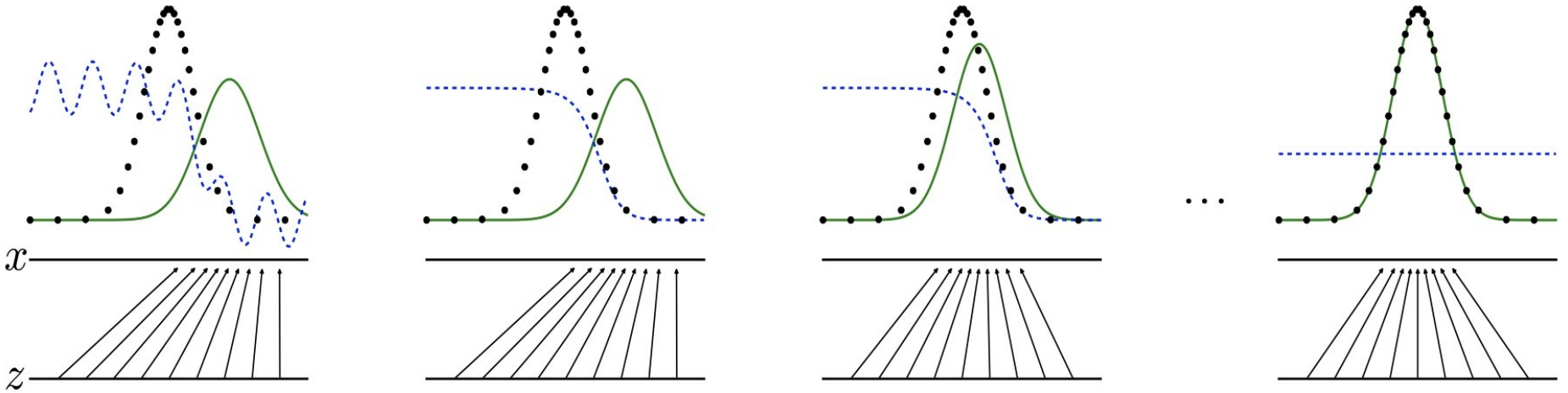
    end for
    //Получаем мини-батч $\{z_1, \ldots, z_m\}$ из распределения p_z
    z = getBatchFromNoisePrior(p_z)
    //Обновляем генератор в сторону убывания его градиента

$$g_w \leftarrow \nabla_{\gamma_g} \frac{1}{m} \sum_{t=1}^{m} [log(1 - D(G(z_t))]$$

  end for

Генеративно-состязательные сети

Процесс обучения:



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

Update G

$$p_{data}(x) = p_z(x), D = \frac{1}{2}$$

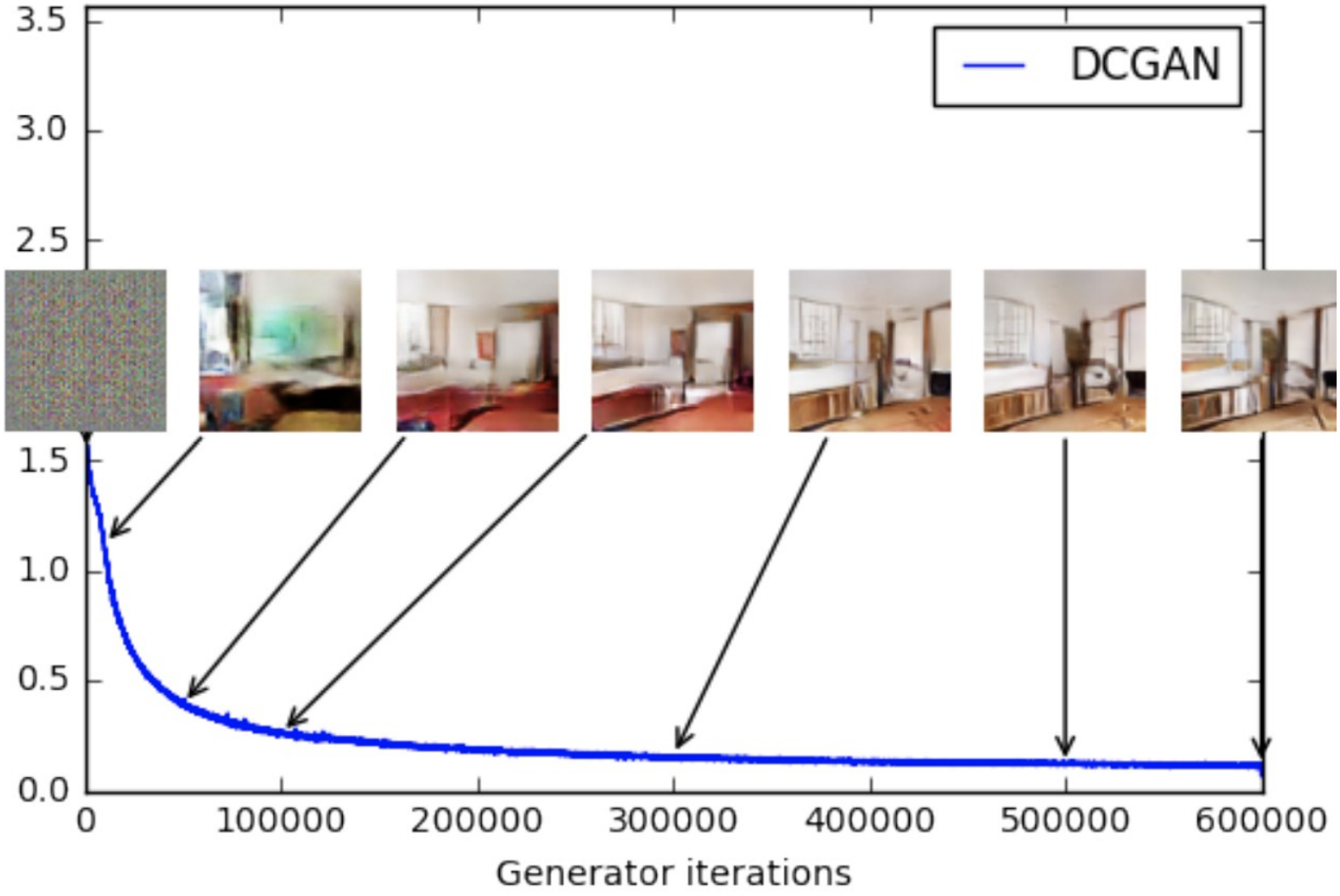
распределение данных

распределение генератора

распределение дискриминатора

Goodfellow et al. Generative Adversarial Networks (2014)

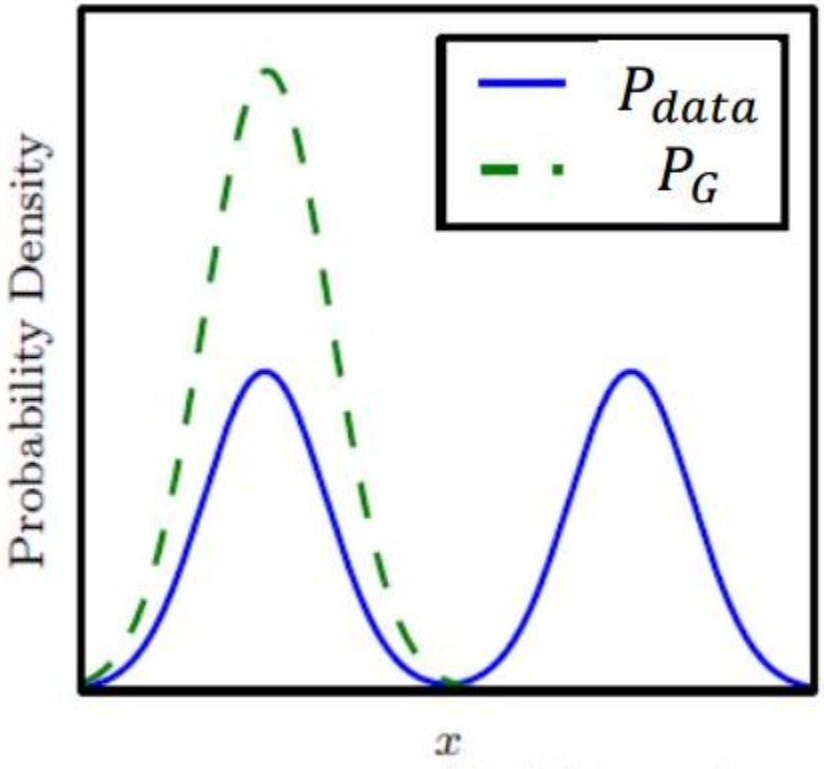
Генеративно-состязательные сети



Arjovsky et al. Wasserstein GAN (2017)

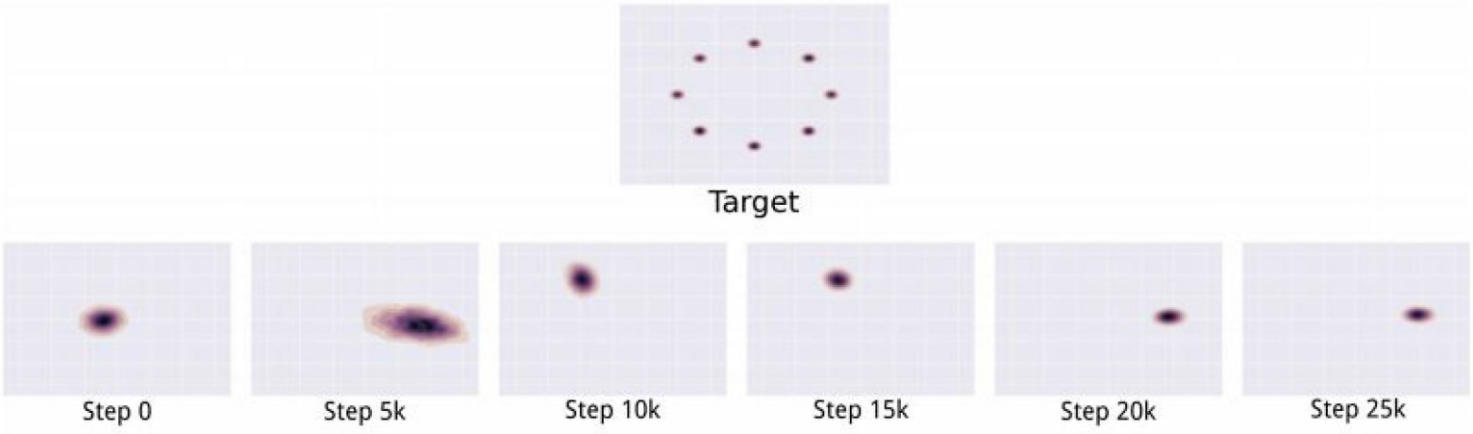
Генеративно-состязательные сети

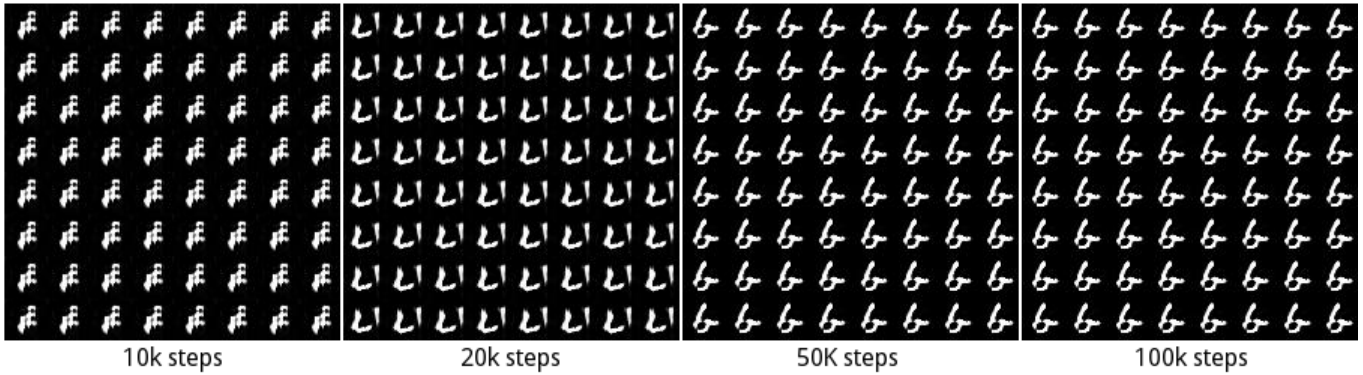
Проблема: схлопывание мод распределения (mode collapse)



Генеративно-состязательные сети

Проблема: схлопывание мод распределения (mode collapse)





Metz et al. Unrolled Generative Adversarial Networks (2016)

Генеративно-состязательные сети

Применение

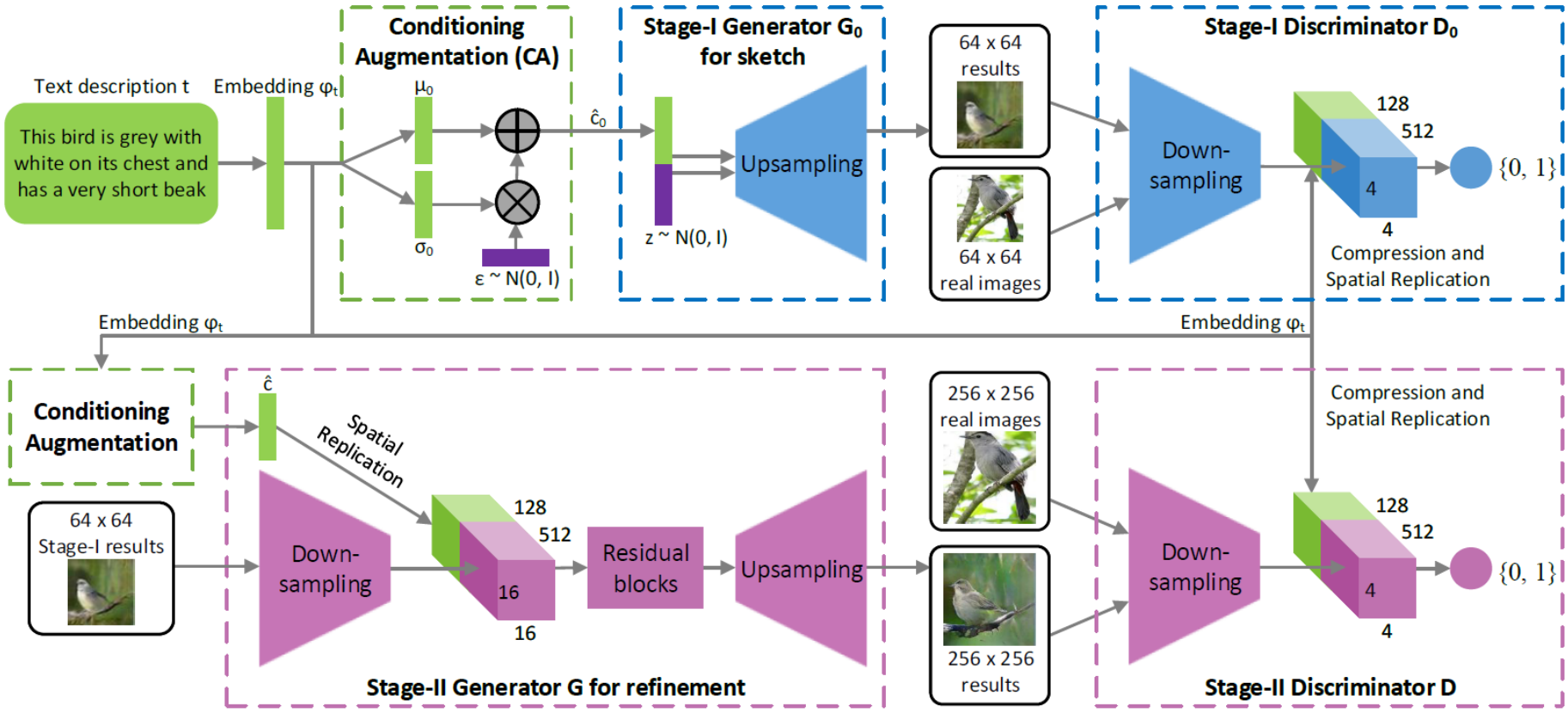
Domain	Topic	Reference
Image	Image translation	Pix2pix [52], PAN [127], CycleGAN [145], DiscoGAN [57]
	Super resolution	SRGAN [65]
	Object detection	SeGAN [28], Perceptual GAN for small object detection [69]
	Object transfiguration	GeneGAN [144], GP-GAN [132]
	Joint image generation	Coupled GAN [74]
	Video generation	VGAN [125], Pose-GAN [126], MoCoGAN [122]
	Text to image	Stack GAN [49], TAC-GAN [18]
	Change facial attributes	SD-GAN [23], SL-GAN [138], DR-GAN [121], AGEGAN [3]
Sequential data	Music generation	C-RNN-GAN [83], SeqGAN [141], ORGAN [41]
	Text generation	RankGAN [73]
	Speech conversion	VAW-GAN [48]
Others	Semi-supervised learning	SSL-GAN [104], CatGAN [115], Triple-GAN [67]
	Domain adaptation	DANN [2], CyCADA [47]
		Unsupervised pixel-level domain adaptation [12]
	Continual learning	Deep generative replay [110]
	Medical image segmentation	DI2IN [136], SCAN [16], SegAN [134]
	Steganography	Steganography GAN [124], Secure steganography GAN [109]

Hong Y. et al. How Generative Adversarial Networks and Their Variants Work: An Overview, 2019

Генеративно-состязательные сети

- StyleGAN и Progressive GAN (Nvidia):
  - https://thisxdoesnotexist.com
  - https://nvlabs.github.io/stylegan2/versions.html
  - https://www.youtube.com/watch?v=G06dEcZ-QTg
  - https://www.youtube.com/watch?v=36lE9tV9vm0
- StackGAN: генерация фотореалистичных изображений (256x256) по текстовому описанию
  - Первый этап – генерация скетча с примитивными формами и цветами, основанного на текстовом описании (64x64)
  - Второй этап – генерация изображения в высоком разрешении с фотореалистичными деталями (256x256) на основе результатов первого этапа и текстового описания

Генеративно-состязательные сети



Генеративно-состязательные сети