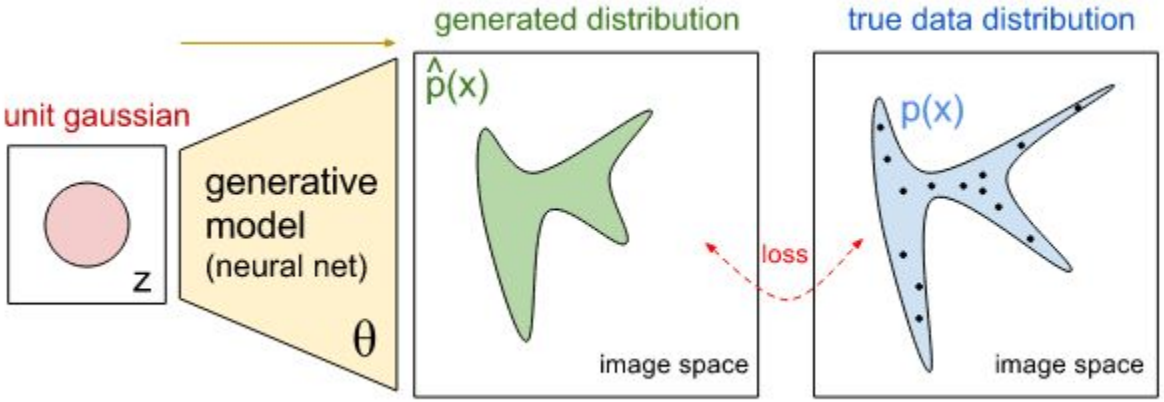
Генеративные модели: VAE, PixelCNN/PixelRNN, GAN

Алексей Илюхов

24 ноября 2020

Что?

Хотим получить функцию, которая из нормального распределения маленькой размерности делает распределение, близкое к реальному большой размерности



Зачем?

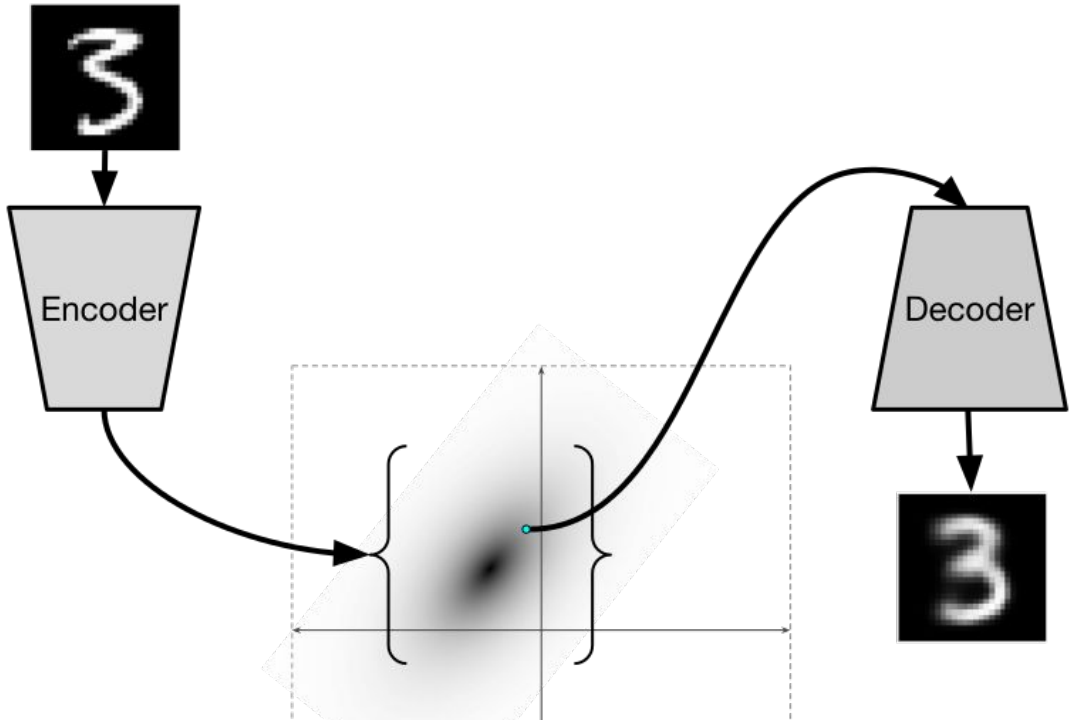
- Сжимать данные
- Извлекать скрытые ~~фичи~~ признаки
- Модифицировать данные
- Получать новые данные, похожие на реальные

Variational Autoencoder

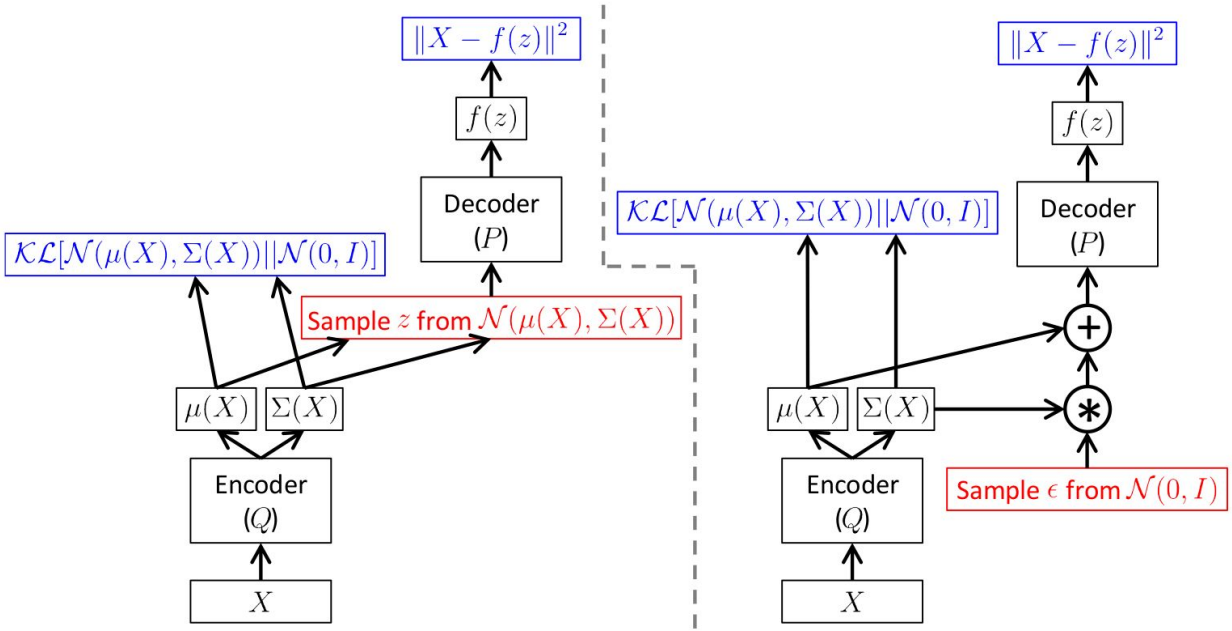
Обучим две модели:

Одна будет сжимать изображение в маленькую размерность

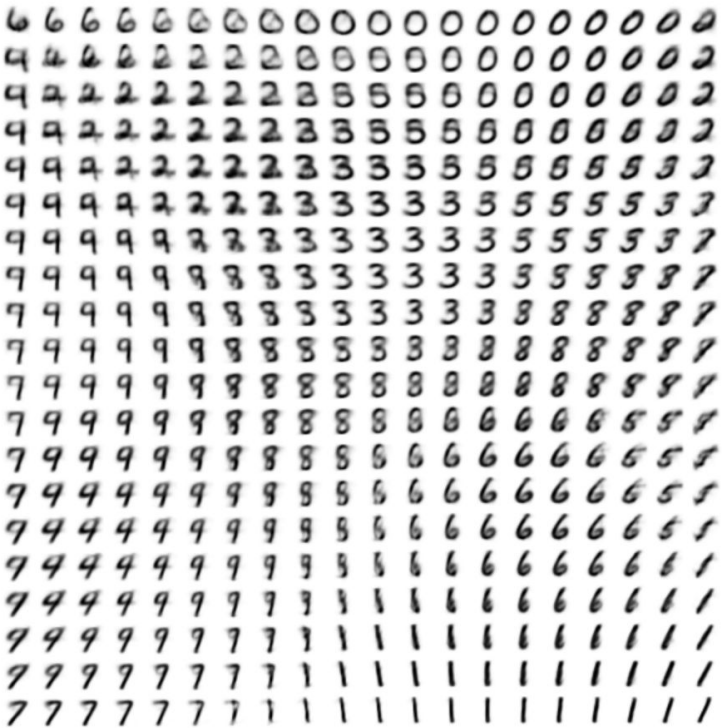
Другая будет пытаться восстановить это изображение

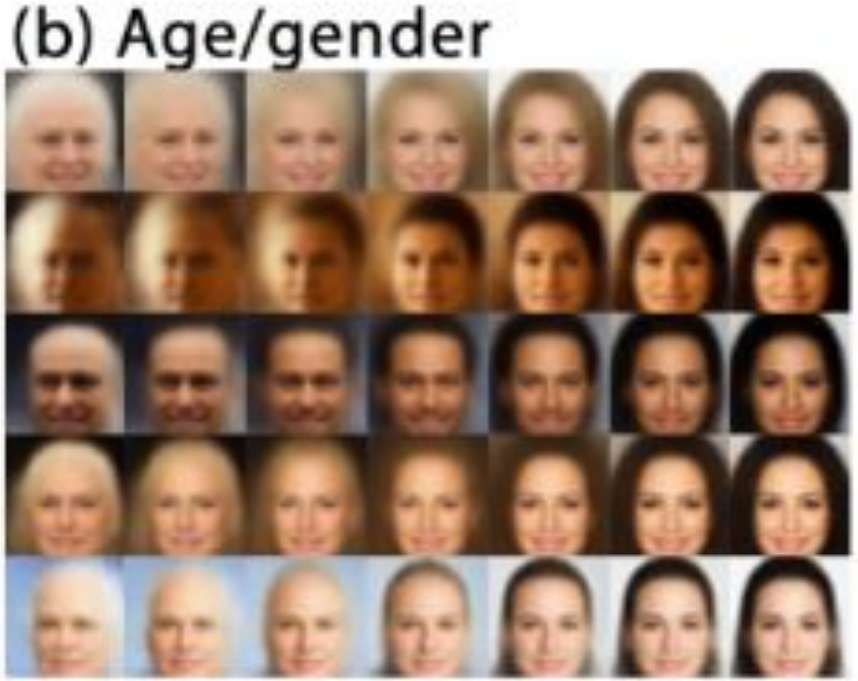


Как это учить?



Использование





Pixel RNN/Pixel CNN

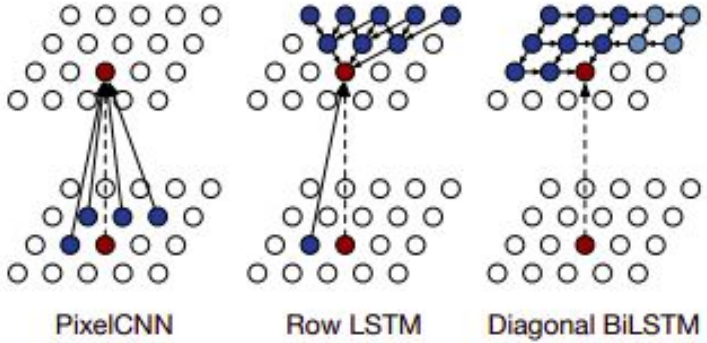
Скажем, что изображение — набор пикселей. Тогда хотим приблизить к реальному распределению

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

Для RGB преобразуется в

$$p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$

Возможные архитектуры



PixelCNN	Row LSTM	Diagonal BiLSTM
7×7 conv, mask A		
3×3 conv layers, mask B	Row LSTM layers Input-State: 3×1 conv, mask B State-State: 3×1 conv, no mask	Diagonal BiLSTM layers Input-State: 1×1 conv, mask B State-State: 1×2 conv, no mask
ReLU followed by 1×1 conv, mask B (2 layers)		
256-way Softmax for each RGB color (Natural images) or Sigmoid (MNIST)		

Использование

Figure 1. Image completions sampled from a PixelRNN.

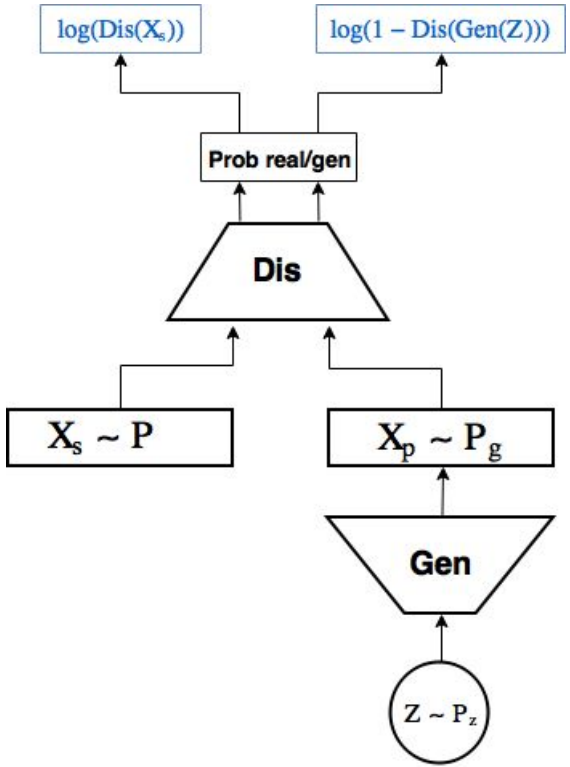
GAN’ы

Хотим обучить две модели

Первая модель будет генерировать изображение — генератор

Вторая будет пытаться отличать сгенерированное изображение от настоящего — дискриминатор

Итеративно делаем K шагов обучения дискриминатора, а потом шаг обучения генератора



Более формально

Целевая функция выглядит так:

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

Генератор пытается максимизировать D(G(x))

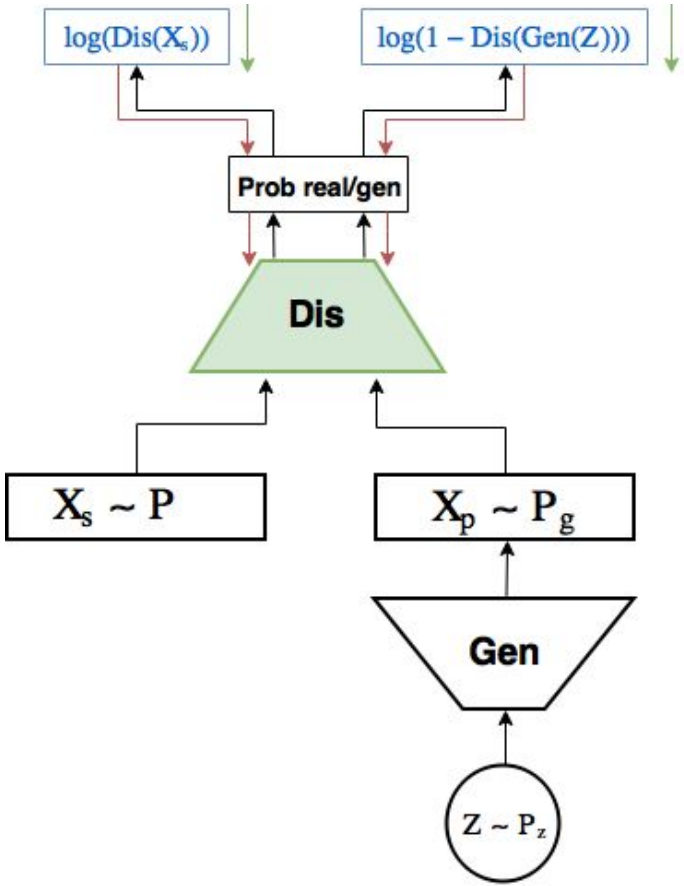
Дискриминатор пытается минимизировать D(G(x)) и максимизировать D(x)

Обучение дискриминатора

Дискриминатор пытается снизить вероятность ошибки на реальных и на сгенерированных данных

Шаг обучения:

$$\theta_d = \theta_d - \nabla_{\theta_d} \left(\log(D(X_s)) + \log(1 - D(G(Z))) \right)$$

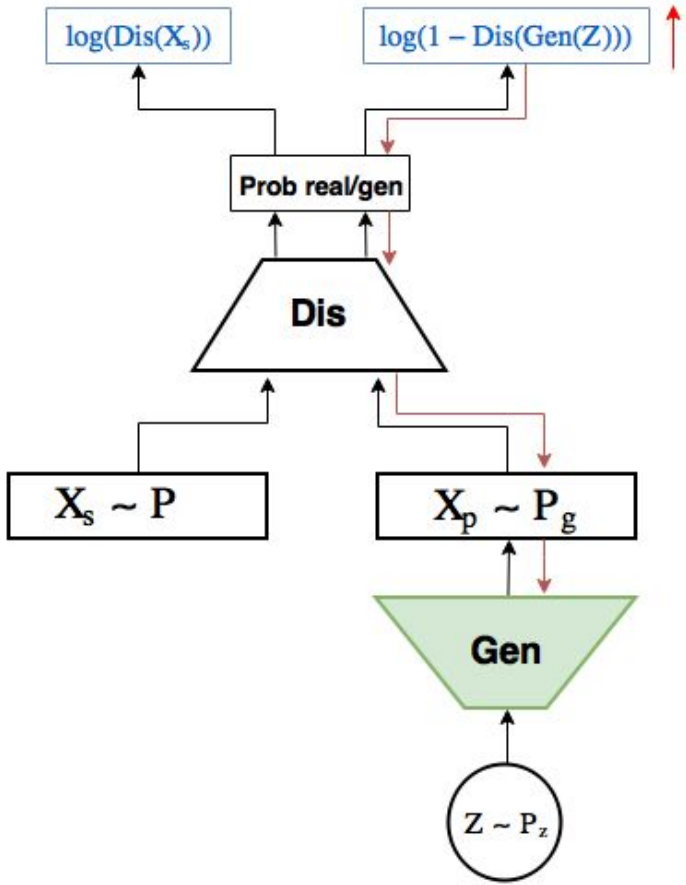


Обучение генератора

Генератор пытается максимизировать вероятность ошибки дискриминатора на сгенерированных данных

Шаг обучения выглядит так:

$$\theta_g = \theta_g + \nabla_{\theta_g} \log(1 - D(G(Z)))$$



Применения

Источники

https://habr.com/ru/post/331500/
https://habr.com/ru/post/331552/
https://habr.com/ru/company/wunderfund/blog/334568
https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
https://habr.com/ru/post/278425/
https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/
https://towardsdatascience.com/summary-of-pixelrnn-by-google-deepmind-7-min-read-938d9871d6d9#:~:text=In%20an%20image%2C%20generally%20a,generate%20pixels%20in%20the%20image.
https://www.machinelearningmastery.ru/auto-regressive-generative-models-pixelrnn-pixelcnn-32d192911173/
https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BA%D1%80%D1%91%D1%81%D1%82%D0%BD%D0%B0%D1%8F_%D1%8D%D0%BD%D1%82%D1%80%D0%BE%D0%BF%D0%B8%D1%8F
https://arxiv.org/abs/1703.10593