Генеративные модели

Лекция 5: Векторная квантизация, *GANs*

Повторение

Проблемы вариационного вывода

Цель:

Хотим аппроксимировать сложное апостериорное распределение $p(\boldsymbol{z}|\mathbf{x}, \boldsymbol{\theta})$ с помощью более простого $q(\boldsymbol{z})$

***Проблема* 1**:

Пространство всех возможных функций $q(\boldsymbol{z})$ бесконечно

***Проблема* 2**:

Для датасета из N объектов $\{\mathbf{x}_1, .., \mathbf{x}_n\}$, нам нужно решать N отдельных задач оптимизации, чтобы найти $\{q_1(\mathbf{z}_1), .., q_n(\boldsymbol{z}_n)\}$

Амортизированный вариационный вывод

Решение 1, параметризация:

Ограничим поиск $q(\boldsymbol{z})$ по конкретным семействам распределений:

$$q(\boldsymbol{z}) \rightarrow q_{\boldsymbol{\phi}}(\boldsymbol{z})$$

Решение 2, амортизация ($\boldsymbol{Amortized\,Variational\,Inference}$):

Вместо отдельных параметров $\boldsymbol{\phi}$ для каждого $\mathbf{x}$ обучим единую ***нейросеть–кодировщик,*** которая по $\mathbf{x}$ будет предсказывать параметры для q:

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}) \rightarrow q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})$$

Амортизированный вариационный вывод

Введя эти 2 ограничения мы:

1. Ушли от необходимости считать вариационное распределение для каждого объекта $\mathbf{x}$
2. Введя параметры $\boldsymbol{\phi}$, мы признаем, что наше решение может быть не точным и KL-дивергенция между $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})$ и $p(\boldsymbol{z}|\mathbf{x}, \boldsymbol{\theta})$ может быть больше нуля

Теперь наша задача сводится к нахождению к нахождению оптимальных $\boldsymbol{\phi}$ и $\boldsymbol{\theta}$, которые максимизируют $ELBO$

- Е-шаг:

$$\boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1} + \eta \nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\phi}, \boldsymbol{\theta}_{k-1}}(\mathbf{x}) \Big|_{\boldsymbol{\phi} = \boldsymbol{\phi}_{k-1}}$$

- М-шаг:

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\phi}_k, \boldsymbol{\theta}}(\mathbf{x}) \Big|_{\boldsymbol{\theta} = \boldsymbol{\theta}_{k-1}}$$

Градиент на М-шаге

Считаем градиент по параметрам $\boldsymbol{\theta}$:

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\boldsymbol{z},\boldsymbol{\theta})] - KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p(\boldsymbol{z}))$$

KL-дивергенция не зависит от $\boldsymbol{\theta}$, так как q параметризуется $\boldsymbol{\phi}$, а априорное распределение $p(\boldsymbol{z})$ обычно фиксировано:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\boldsymbol{z},\boldsymbol{\theta})] = \nabla_{\boldsymbol{\theta}} \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}|\boldsymbol{z},\boldsymbol{\theta}) d\boldsymbol{z} = \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\boldsymbol{z},\boldsymbol{\theta}) d\boldsymbol{z}$$

Для оценки матожидания будем использовать метод Монте-Карло:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) \approx \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\boldsymbol{z}^*,\boldsymbol{\theta}), \qquad \boldsymbol{z}^* \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$$

Поскольку $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})$ обусловлено на конкретный $\mathbf{x}$, то его вероятностная масса будет сосредоточена в области пространства $\boldsymbol{z}$, которая наиболее вероятна для этого $\mathbf{x}$, а значит дисперсия будет сильно ниже, чем при наивном подходе

Reparameterization trick

Считаем градиент по параметрам $\boldsymbol{\phi}$ при фиксированных $\boldsymbol{\theta}$:

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})}[\log p(\mathbf{x}|\boldsymbol{z},\boldsymbol{\theta})] - KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})||p(\boldsymbol{z}))$$

Будем использовать ***трюк репараметризации*** (***reparameterization trick***) – считать матожидание по распределению, которое не зависит от $\boldsymbol{\phi}$

Предположим, что мы можем выразить $\boldsymbol{z}$ из распределения $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})$ как детерминированную функцию $\boldsymbol{g}$ от некоторой независимой случайной величины $\boldsymbol{\epsilon}$ из распределения $p(\boldsymbol{\epsilon})$:

$$\boldsymbol{z} = \boldsymbol{g}_{\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon}) \qquad\qquad \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$$

Пусть $p(\boldsymbol{\epsilon}) = \mathcal{N}(\mathbf{0},\boldsymbol{I})$ и функция репараметризации имеет вид:

$$\boldsymbol{z} = \boldsymbol{g}_{\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon}) = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\mathbf{x}) \cdot \boldsymbol{\epsilon}$$

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}),\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\mathbf{x}))$$

Градиент на E-шаге

Считаем градиент по параметрам $\boldsymbol{\phi}$ при фиксированных $\boldsymbol{\theta}$:

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\boldsymbol{z},\boldsymbol{\theta})] - KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p(\boldsymbol{z}))$$

Используем ***reparameterization trick***, чтобы внести оператор градиента внутрь интеграла:

$$\nabla_{\boldsymbol{\phi}} \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}|\boldsymbol{z},\boldsymbol{\theta}) d\boldsymbol{z} = \int p(\boldsymbol{\epsilon}) \nabla_{\boldsymbol{\phi}} \log p\left(\mathbf{x}\middle|\boldsymbol{g}_{\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon}),\boldsymbol{\theta}\right) d\boldsymbol{\epsilon}$$

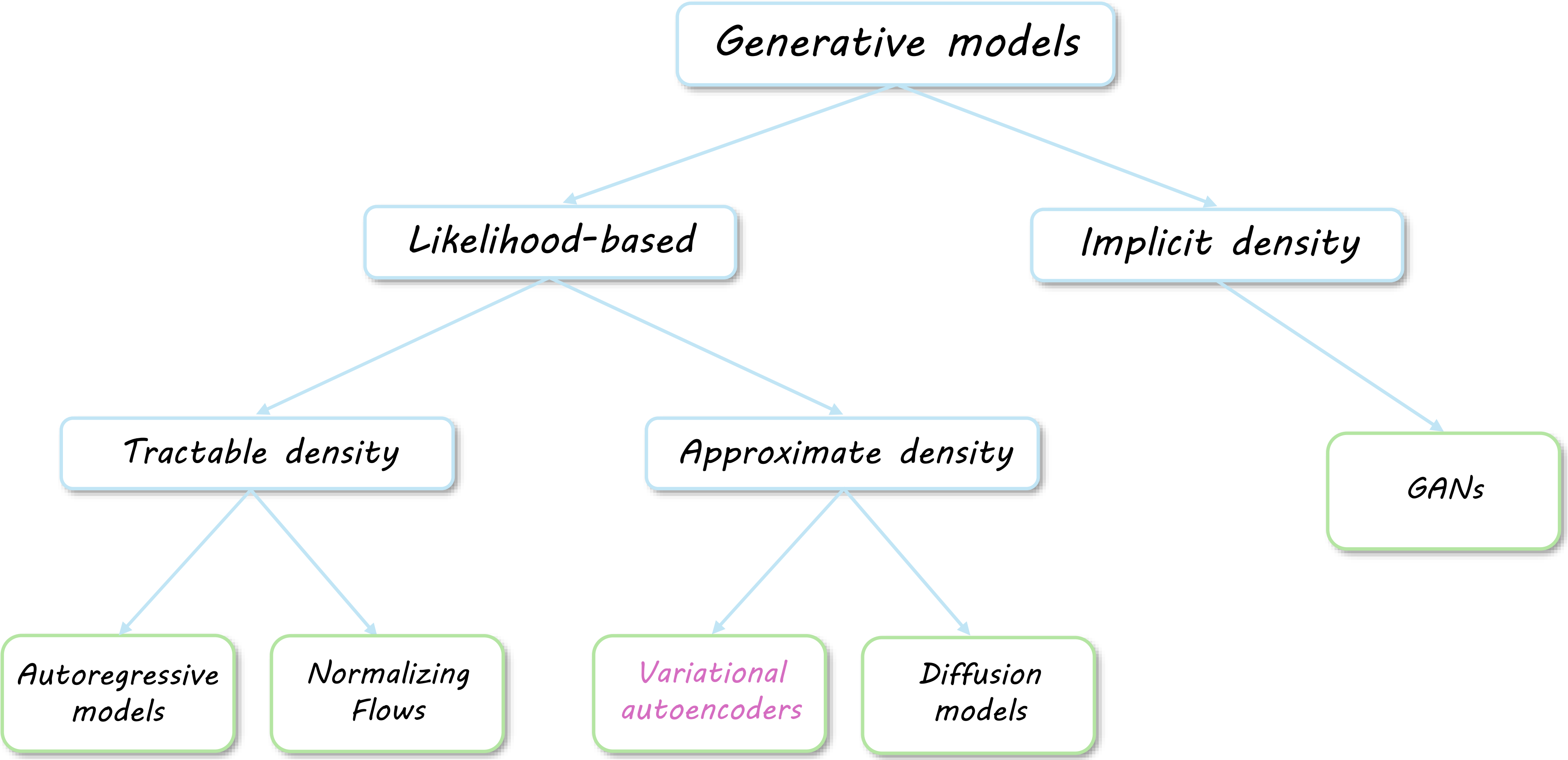
Оценим этот интеграл с помощью метода Монте-Карло:

$$\int p(\boldsymbol{\epsilon}) \nabla_{\boldsymbol{\phi}} \log p\left(\mathbf{x}\middle|\boldsymbol{g}_{\boldsymbol{\phi}}(\mathbf{x},\boldsymbol{\epsilon}),\boldsymbol{\theta}\right) d\boldsymbol{\epsilon} \approx \nabla_{\boldsymbol{\phi}} \log p\left(\mathbf{x}\middle|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\mathbf{x}) \cdot \boldsymbol{\epsilon}^*,\boldsymbol{\theta}\right), \qquad \boldsymbol{\epsilon}^* \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$$

Для KL –дивергенции между двумя нормальными распределениями существует аналитическая формула:

$$\nabla_{\boldsymbol{\phi}} KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p(\boldsymbol{z})) = \nabla_{\boldsymbol{\phi}} KL(\mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}),\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\mathbf{x})\right)||\mathcal{N}(\mathbf{0},\boldsymbol{I}))$$

Зоопарк генеративных моделей

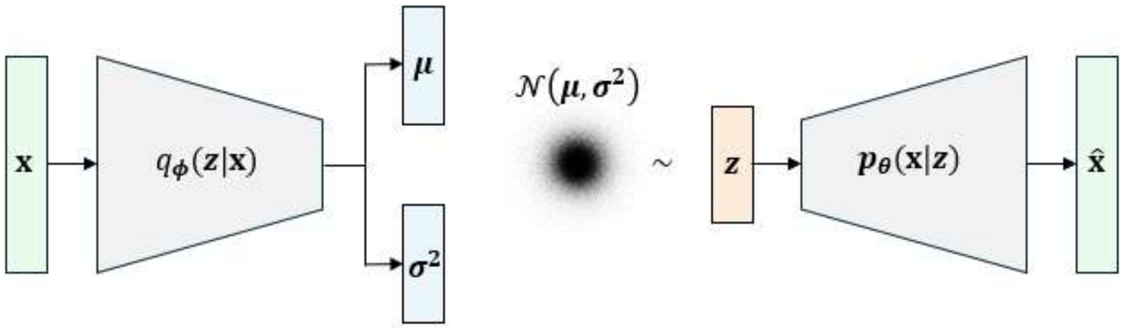


Variational Autoencoder

- ***VAE*** состоит из двух нейросетей: ***кодировщика*** и ***декодировщика***

Общая идея:

- Одному объекту соответствует не одна точка, а целое распределение $q_\phi(z|\mathbf{x})$
- Кодировщик $q_\phi(z|\mathbf{x})$ выдает не один вектор, а два – вектор средних $\boldsymbol{\mu}$ и дисперсий $\boldsymbol{\sigma^2}$
- Из этого распределения берется случайная точка $\boldsymbol{z}$, которая затем подается в декодер для восстановления



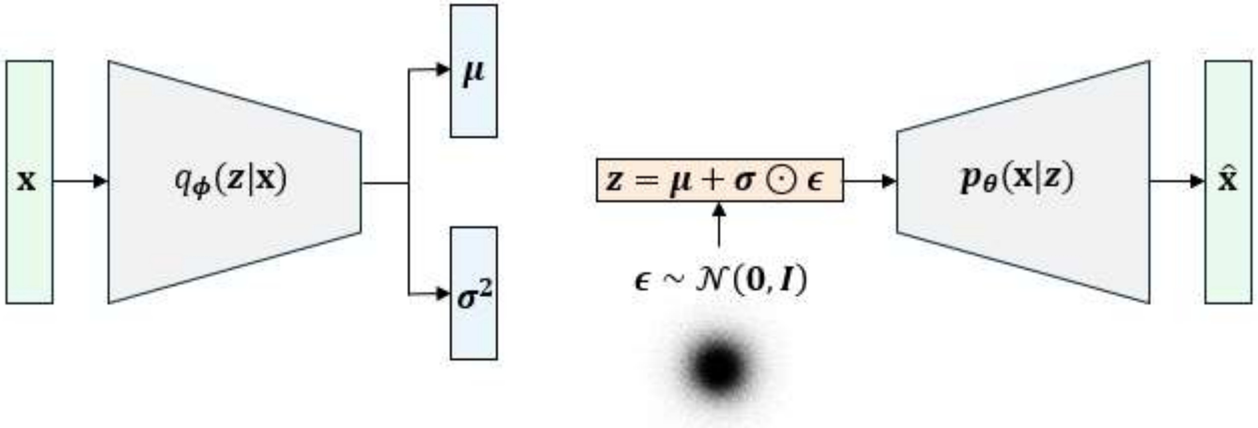
Обучение VAE

Алгоритм:

- Берем случайный объект $\mathbf{x}$
- Пропускаем его через кодировщик $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})$, получаем $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x})$ и $\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\mathbf{x})$
- Вычисляем $\boldsymbol{z}^*$ с помощью репараметризации:

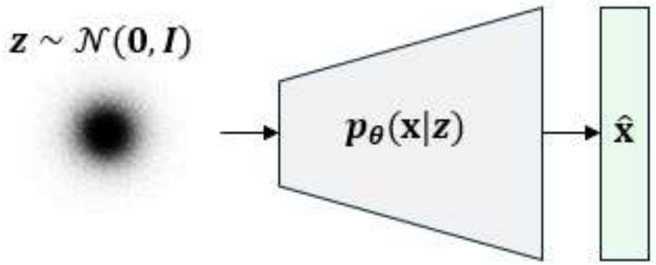
  Берем $\boldsymbol{\epsilon}^* \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  Считаем $\boldsymbol{z}^* = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\mathbf{x}) \cdot \boldsymbol{\epsilon}^*$

- Пропускаем $\boldsymbol{z}^*$ через декодер и получаем восстановленный $\hat{\mathbf{x}}$
- Считаем лосс, обновляем параметры $\boldsymbol{\phi}$ и $\boldsymbol{\theta}$



Генерация VAE

- Кодировщик $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})$ больше не нужен
- Берем случайный вектор $\boldsymbol{z}^*$ из априорного распределения $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$
- Подаем $\boldsymbol{z}^*$ на вход обученному декодеру



План

- Discrete VAE
- Vector Quantization
- Generative Adversarial Networks

Discrete VAE

Дискретный VAE

- Многие данные по своей природе дискретны
- Трансформеры работают именно с дискретными токенами

Пусть c – новая скрытая переменная является одной из K классов:

$$c \sim Categorical(\boldsymbol{\pi})$$

$\boldsymbol{\pi}$ – вектор вероятностей:

$$\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K), \qquad \pi_k = P(c = k), \qquad \sum_{k=1}^{K} \pi_k = 1$$

Выберем априорное распределение $p(c)$ самым простым – равномерным:

$$p(c) = \frac{1}{K}$$

Дискретный VAE

- Кодировщик $q_{\boldsymbol{\phi}}(c|\mathbf{x})$ теперь будет предсказывать вектор вероятностей $\boldsymbol{\pi}$

Преобразуем $ELBO$:

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(c|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|c)] - KL(q_{\boldsymbol{\phi}}(c|\mathbf{x})||p(c))$$

Распишем KL –дивергенцию:

$$KL(q_{\boldsymbol{\phi}}(c|\mathbf{x})||p(c)) = \sum_{k=1}^{K} q_{\boldsymbol{\phi}}(k|\mathbf{x}) \log \frac{q_{\boldsymbol{\phi}}(k|\mathbf{x})}{p(k)} =$$

$$\sum_{k=1}^{K} q_{\boldsymbol{\phi}}(k|\mathbf{x}) \log q_{\boldsymbol{\phi}}(k|\mathbf{x}) - \sum_{k=1}^{K} q_{\boldsymbol{\phi}}(k|\mathbf{x}) \log p(k) = -H\left(q_{\boldsymbol{\phi}}(c|\mathbf{x})\right) + \log K$$

Дискретный VAE

Итоговая формула $ELBO$ для дискретного VAE:

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(c|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|c)] + H\left(q_{\boldsymbol{\phi}}(c|\mathbf{x})\right) - log\ K$$

- Чтобы посчитать матожидание $\mathbb{E}_{q_{\boldsymbol{\phi}}(c|\mathbf{x})}$, нам нужно сэмплировать c из распределения, которое выдал кодировщик
- Эта операция недифференцируема
- Трюк репараметризации больше не работает, нужно искать аналог для дискретных распределений

Vector Quantization

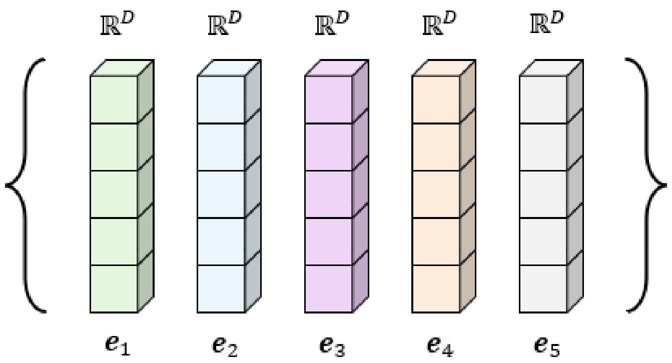
Векторное квантование

Идея ***векторной квантизации*** (***Vector Quantization, VQ***):

- Будем аппроксимировать непрерывное пространство конечным набором векторов

- Создадим кодовую книгу (***codebook***) – словарь из K обучаемых векторов $\{\boldsymbol{e}_k\}_{k=1}^{K}$

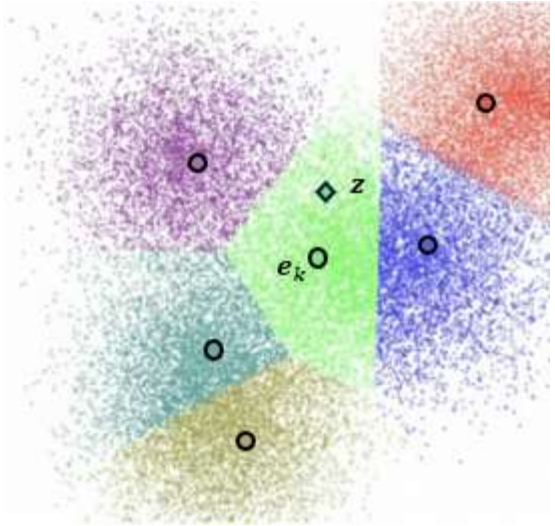
- Каждый $\boldsymbol{e}_k$ – вектор в пространстве $\mathbb{R}^D$:
  K – размер словаря
  D – размерность каждого вектора



Векторное квантование

Для вектора $\boldsymbol{z}_e \in \mathbb{R}^D$ находим ближайший к нему вектор из кодовой книги и получаем квантованный вектор $\boldsymbol{z}_q$:

$$\boldsymbol{z}_q = q(\boldsymbol{z}_e) = \boldsymbol{e}_{k^*}, \qquad k^* = \arg\min_k ||\boldsymbol{z}_e - \boldsymbol{e}_k||$$



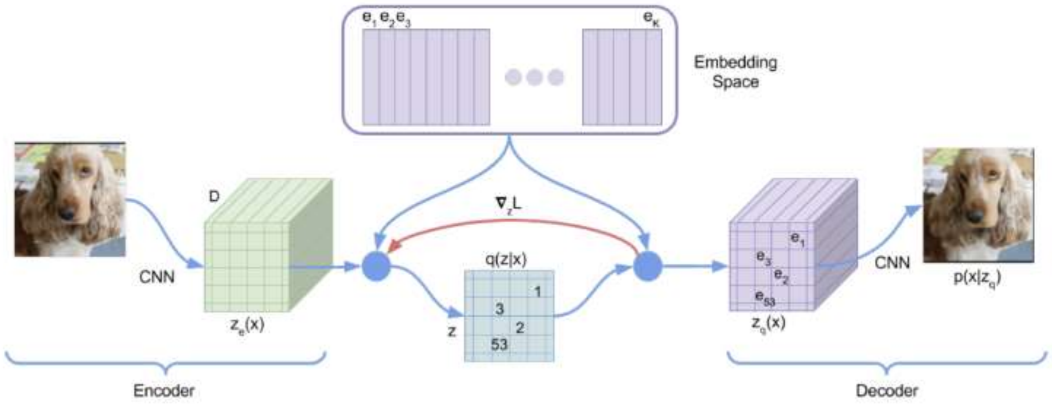
Это детерминированная операция, похожая на поиск ближайшего соседа в $\boldsymbol{K - Means}$

Векторное квантование

- $\boldsymbol{Encoder}$: принимает на вход объект **x**, сжимает его и выдает непрерывный вектор $\boldsymbol{z_e}$ (без $\boldsymbol{\mu}$ и $\boldsymbol{\sigma}$)

- $\boldsymbol{VQ}$: берёт вектор $\boldsymbol{z_e}$ и находит ближайший к нему вектор $\boldsymbol{e_{k^*}}$ из кодовой книги

- $\boldsymbol{Decoder}$: на вход декодеру приходит уже квантованный вектор $\boldsymbol{z_q} = \boldsymbol{e_{k^*}}$, из которого восстанавливается исходный объект



Векторное квантование

Особенностью $VQ - VAE$ является использование детерминированного вариационного апостериорного распределения:

$$q_{\boldsymbol{\phi}}(c = k^* | \mathbf{x}) = \begin{cases} 1 & \text{для} \quad k^* = \underset{k}{arg\,min}\, ||\boldsymbol{z}_e - \boldsymbol{e}_k|| \\ 0 & \text{иначе} \end{cases}$$

Такой выбор значительно упрощает $ELBO$:

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(c|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|c)] + \underbrace{H\left(q_{\boldsymbol{\phi}}(c|\mathbf{x})\right)}_{0} - log\, K$$

Оптимизация сводится к максимизации члена реконструкции:

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(c|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|c)] - log\, K = \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_q) - log\, K$$

$Straight - Through\ Estimator$

Операция квантования $k^* = \arg\min_k ||\boldsymbol{z}_e - \boldsymbol{e}_k||$ является недифференцируемой

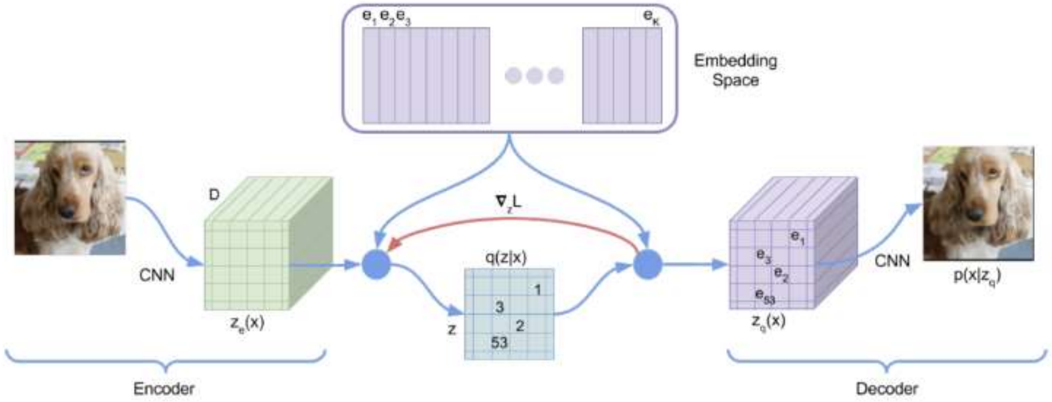
Авторы $VQ-VAE$ предложили ***прямое оценивание градиента*** ($\boldsymbol{Straight - Through\ Estimator}$, $\boldsymbol{STE}$)

Механизм $\boldsymbol{STE}$:

- На этапе обратного распространения градиент просто копируется с выхода недифференцируемого блока на его вход

- В $VQ-VAE$ градиент по $\boldsymbol{z}_q$ напрямую передается $\boldsymbol{z}_e$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_e} \approx \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_q}$$



$Straight - Through\ Estimator$

Математически это можно представить как аппроксимацию в $\boldsymbol{chain\,rule}$:

$$\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_q)}{\partial \boldsymbol{\phi}} = \frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_q)}{\partial \mathbf{z}_q} \cdot \frac{\partial \mathbf{z}_q}{\partial \mathbf{z}_e} \cdot \frac{\partial \mathbf{z}_e}{\partial \boldsymbol{\phi}} \approx \frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_q)}{\partial \mathbf{z}_q} \cdot \frac{\partial \mathbf{z}_e}{\partial \boldsymbol{\phi}}$$

- Выход кодировщика $\boldsymbol{z}_e$ и квантованный вектор $\boldsymbol{z}_q$ находятся рядом в одном и том же пространстве $\mathbb{R}^D$
- Градиент от декодера $\nabla_{z_q}\mathcal{L}$ показывает, куда сдвинуть $\boldsymbol{z}_q$, чтобы улучшить реконструкцию
- Кодировщик применяет это к $\boldsymbol{z}_e$ и подталкивает модель выбирать более подходящий вектор $\boldsymbol{e}_k$

Хотя такая оценка является смещенной, она часто указывает в направлении истинного градиента и хорошо работает на практике

$VQ-VAE$

Нам нужно обучать кодировщик $\boldsymbol{\phi}$, декодер $\boldsymbol{\theta}$ и кодовую книгу $\{\boldsymbol{e}_k\}_{k=1}^{K}$

В $VQ-VAE$ используется составная функция потерь:

$$\mathcal{L} = \underbrace{\log p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z}_q)}_{Reconstruction\ loss} + \underbrace{||sg[\boldsymbol{z}_e] - \boldsymbol{e}_{k^*}||^2}_{VQ-Loss} + \underbrace{\beta||\boldsymbol{z}_e - sg[\boldsymbol{e}_{k^*}]||^2}_{Commitment\ Loss}$$

$sg-$ оператор $\boldsymbol{stop-gradient}$

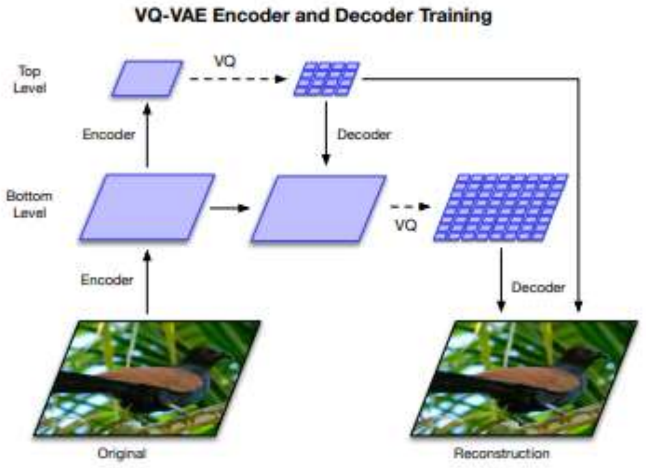
$\boldsymbol{VQ-Loss}$:
- Обновляет только $codebook$
- Заставляет выбранный вектор $\boldsymbol{e}_{k^*}$ сдвинуться ближе к $\boldsymbol{z}_e$

$\boldsymbol{Commitment-Loss}$:
- Обновляет только кодировщик
- Заставляет выход кодировщика $\boldsymbol{z}_e$ сдвинуться ближе к $\boldsymbol{e}_{k^*}$

$VQ - VAE2$

- $VQ - VAE$ использует один *codebook*
- Сразу хотим, чтобы модель умела улавливать и локальные, и глобальные детали

- $\mathbf{VQ - VAE2}$ использует несколько иерархических *codebooks* *(*верхний и нижний*)*
- Кодировщик выдает несколько карт признаков $\boldsymbol{z_e}$ с разным пространственным разрешением
- Карты признаков верхнего и нижнего уровня квантуются с помощью своих *codebooks*



Generative Adversarial Networks

Likelihood based models

- Мы хотим научиться воспроизводить объекты из истинного распределения $p_{data}(\mathbf{x})$
- Обычно мы просто максимизируем правдоподобие $p_{\boldsymbol{\theta}}(\mathbf{x})$

***Модель* 1**:

$$p_1(\mathbf{x}) = \sum_{i=1}^{n} \mathcal{N}(\mathbf{x}|\mathbf{x}_i, \epsilon \boldsymbol{I})$$

- Строим узкий гауссовский пик каждого $\mathbf{x}_i$
- Для малого ϵ при генерации модель выдаст образец, почти идентичный одному из обучающих
- Если взять тестовый образец, то его правдоподобие будет низким

***Модель* 2**:

$$p_2(\mathbf{x}) = 0.01p(\mathbf{x}) + 0.99p_{noise}(\mathbf{x})$$

- Модель с вероятностью 0.01 использует хорошую модель $p(\mathbf{x})$, и в 0.99 случаях генерируем шум
- Почти все сгенерированные объекты будут представлять шум
- Правдоподобие такой модели:

$$\log[0.01p(\mathbf{x}) + 0.99p_{noise}(\mathbf{x})] \geq \log[0.01p(\mathbf{x})] = \log p(\mathbf{x}) - \log 100$$

Likelihood based models

- Способность генерировать качественные объекты не гарантирует высокого правдоподобия модели

- Высокое правдоподобие не гарантирует, что модель будет генерировать качественные образцы

- Правдоподобие – не идеальная мера качества для генеративных моделей

- Правдоподобие часто может быть невычислимым

- Это заставляет задуматься о новой идее, свободной от правдоподобия – ***идее состязательного обучения***

$Likelihood\ free - learning$

- Мы отказываемся от прямой аппроксимации плотности $p_{data}(\mathbf{x})$
- Вместо этого будем учить функцию генератор $p_{\boldsymbol{\theta}}(\mathbf{x})$ создавать образцы, которые выглядят как будто они взяты и $p_{data}(\mathbf{x})$

У нас будет 2 набора данных:

- ***Реальные образцы*** $(\boldsymbol{real\ samples})$ $\{\mathbf{x}_i\}_{i=1}^{n_1}$ из распределения $p_{data}(\mathbf{x})$
- ***Сгенерированные образцы*** $(\boldsymbol{fake\ samples})$ $\{\mathbf{x}_i\}_{i=1}^{n_2}$ из распределения $p_{\boldsymbol{\theta}}(\mathbf{x})$

Вводим ***классификатор*** $(\boldsymbol{Discriminator})$, который будет учиться различать эти 2 набора:

- $p(y=1|\mathbf{x})-$ вероятность того, что образец $\mathbf{x}$ является реальным
- $p(y=0|\mathbf{x})-$ вероятность того, что образец $\mathbf{x}$ является сгенерированным

$Likelihood\ free - learning$

- Мы считаем, что генератор идеально изучил истинное распределение $p_{\boldsymbol{\theta}}(\mathbf{x}) = p_{data}(\mathbf{x})$, когда лучший из возможных дискриминаторов не может отличить реальные образцы от сгенерированных

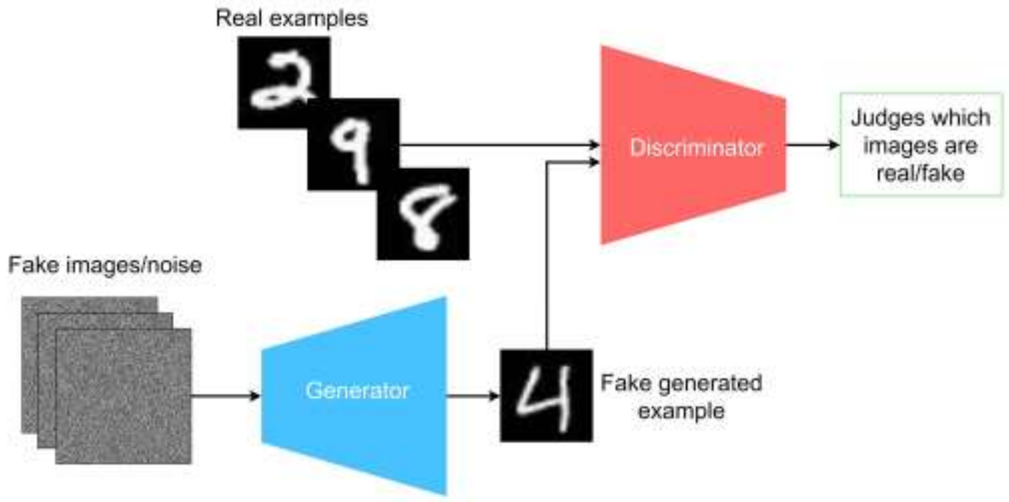
- В этой точке дискриминатор будет для любого $\mathbf{x}$ выдавать вероятность 0.5

- Наша цель – достичь этого равновесия

Generative Adversarial Networks

Обучение $GANs$ реализуется через состязание двух нейросетей*:*

- ***Генератор*** $\boldsymbol{G}$: создаёт реалистичные данные $\mathbf{x} = G(\boldsymbol{z})$ из вектора $\boldsymbol{z} \sim p(\boldsymbol{z})$
- ***Дискриминатор*** $\boldsymbol{D}$: бинарный классификатор, который получает на вход объект $\mathbf{x}$ и выдает вероятность $D(\mathbf{x})$, что $\mathbf{x} -$ реальный



Generator

По аналогии моделей со скрытыми переменными $\boldsymbol{z}$ – это латентный вектор, $p(\boldsymbol{z})$ – априорное распределение

Мы можем думать о генераторе, как о детерминированном декодере*:*

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{z}) \cdot p(\boldsymbol{z})$$

где $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \delta(\mathbf{x} - G_{\boldsymbol{\theta}}(\boldsymbol{z}))$

δ – дельа-функция Дирака, для одного $\boldsymbol{z}$ на выходе вся вероятность сосредоточена в одной точке $\mathbf{x} = \boldsymbol{G}_{\boldsymbol{\theta}}(\boldsymbol{z})$

Функция потерь

- Дискриминатор решает задачу бинарной классификации с метками $y = 1$ (реальный) и $y = 0$ (сгенерированный) и выдает вероятность $p(y = 1|\mathbf{x})$

- Хотим обучить D так, чтобы он максимизировал вероятность (правдоподобие) присвоения правильных меток всем классам:

$$\max_{p(y|\mathbf{x})} \left(\mathbb{E}_{p_{data}(\mathbf{x})} \log p(y = 1|\mathbf{x}) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})} \log p(y = 0|\mathbf{x})\right) =$$

$$= \max_{D} \left(\mathbb{E}_{p_{data}(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})} \log\big(1 - D(\mathbf{x})\big)\right)$$

- Генератор, наоборот, пытается минимизировать эту функцию:

$$\min_{G} \max_{D} \left(\mathbb{E}_{p_{data}(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})} \log\big(1 - D(\mathbf{x})\big)\right)$$

$$\min_{G} \max_{D} \left(\mathbb{E}_{p_{data}(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p(\boldsymbol{z})} \log\Big(1 - D\big(G(\boldsymbol{z})\big)\Big)\right)$$

Функция потерь 1

$$\min_{p(y|\mathbf{x})} \left(-\mathbb{E}_{p_{data}(\mathbf{x})}[\log p(y=1|\mathbf{x})] - \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}[\log p(y=0|\mathbf{x})]\right)$$

- $-\mathbb{E}_{p_{data}(\mathbf{x})}[\log p(y=1|\mathbf{x})]$ отвечает за реальные данные
- Хотим, чтобы $D(\mathbf{x}) = p(y=1|\mathbf{x})$ был равен 1

- Если $D(\mathbf{x}) = 1$, то $\log 1 = 0$
- Если $D(\mathbf{x}) = 0.01$, то $\log 0.01 \approx -4.6$

- $-\mathbb{E}_{p_{data}(\mathbf{x})}[\log p(y=1|\mathbf{x})]$ – это штраф за то, что дискриминатор назвал реальный объект сгенерированным

Функция потерь 2

$$\min_{p(y|\mathbf{x})} \left(-\mathbb{E}_{p_{data}(\mathbf{x})}[\log p(y=1|\mathbf{x})] - \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}[\log p(y=0|\mathbf{x})]\right)$$

- $-\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}[\log p(y=0|\mathbf{x})]$ отвечает за сгенерированные данные
- Хотим, чтобы $1 - D(\mathbf{x}) = p(y=0|\mathbf{x})$ был равен 1

- Если $D(\mathbf{x}) = 0$, то $\log 1 = 0$
- Если $D(\mathbf{x}) = 0.99$, то $\log 0.01 \approx -4.6$

- $-\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}[\log p(y=0|\mathbf{x})]$ – это штраф за то, что дискриминатор назвал сгенерированный объект реальным

Оптимальный дискриминатор

Найдем оптимальный дискриминатор D^* при любом фиксированном генераторе G

Хотим найти D, который максимизирует $V(G, D)$:

$$V(G, D) = \mathbb{E}_{p_{data}(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})} \log\bigl(1 - D(\mathbf{x})\bigr) =$$

$$\int \bigl[p_{data}(\mathbf{x}) \log D(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x}) \log\bigl(1 - D(\mathbf{x})\bigr)\bigr] d\mathbf{x}$$

$$\frac{dy(D)}{dD} = \frac{p_{data}(\mathbf{x})}{D(\mathbf{x})} - \frac{p_{\boldsymbol{\theta}}(\mathbf{x})}{1 - D(\mathbf{x})} = 0$$

$$D^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}$$

Генератор

Найдем цель генератора при условии оптимального дискриминатора D^*

Подставим D^* в нашу функцию $V(G, D)$:

$$V(G, D^*) = \int p_{data}(\mathbf{x}) \log D(\mathbf{x})\, d\mathbf{x} + \int p_{\boldsymbol{\theta}}(\mathbf{x}) \log\big(1 - D(\mathbf{x})\big)$$

$$= \int p_{data}(\mathbf{x}) \log \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})} d\mathbf{x} + \int p_{\boldsymbol{\theta}}(\mathbf{x}) \log\left(1 - \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}\right)$$

$$= KL\left(p_{data}(\mathbf{x}) \,||\, \frac{p_{data}(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}{2}\right) + KL\left(p_{\boldsymbol{\theta}}(\mathbf{x}) || \frac{p_{data}(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}{2}\right) - 2\log 2$$

$$= 2JSD\big(p_{data}(\mathbf{x}) \,||\, p_{\boldsymbol{\theta}}(\mathbf{x})\big) - 2\log 2$$

Оптимальный GAN

Дивергенция Йенсена-Шеннона:

$$JSD\left(p_{data}(\mathbf{x}) \,||\, p_{\boldsymbol{\theta}}(\mathbf{x})\right) = \frac{1}{2}\left[KL\left(p_{data}(\mathbf{x}) \,||\, \frac{p_{data}(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}{2}\right) + KL\left(p_{\boldsymbol{\theta}}(\mathbf{x}) || \frac{p_{data}(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}{2}\right)\right]$$

Истинная цель генератора – минимизация дивергенции Йенсена-Шеннона:

$$2JSD\left(p_{data}(\mathbf{x}) \,||\, p_{\boldsymbol{\theta}}(\mathbf{x})\right) - 2\log 2 \to \min_{\boldsymbol{\theta}}$$

- $JSD(p||q) = 0$ когда $p = q$
- В этой точке $D^*(\mathbf{x}) = \frac{1}{2}$, $V(G^*, D^*) = -2\log 2$

- Теоретическое равновесие достигается, когда генератор идеально воспроизводит данные, а дискриминатор не может отличить реальные объекты от сгенерированных

Проблемы GAN

Теоретическое равновесие требует, чтобы модели G и D были достаточно мощными (любыми функциями)

В действительности G и D — нейросети с конечным набором параметров

Из-за этой ограниченной мощности настоящее теоретическое равновесие для многих GAN архитектур может не существовать

Это и является основной причиной многих проблем при обучении GAN

Обучение GAN

Теперь наша $min\,max$ задача будет происходить над конкретными параметрами

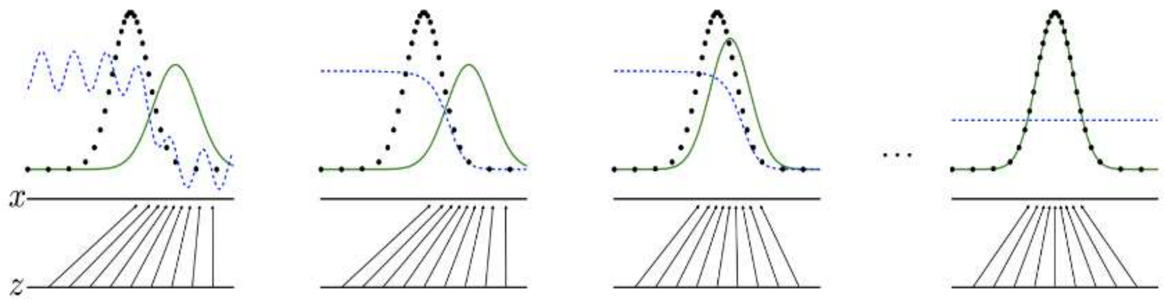
$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \left[\mathbb{E}_{p_{data}(\mathbf{x})} \log D_{\boldsymbol{\phi}}(\mathbf{x}) + \mathbb{E}_{p(\boldsymbol{z})} \log(\mathbf{1} - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{z})))\right]$$

Обучаем D:

- Фиксируем веса генератора $\boldsymbol{\theta}$
- Делаем шаг градиентного спуска по $\boldsymbol{\phi}$, чтобы максимизировать $V(G, D)$

Обучаем G:

- Фиксируем веса дискриминатора $\boldsymbol{\phi}$
- Делаем шаг градиентного спуска по $\boldsymbol{\theta}$, чтобы минимизировать $V(G, D)$



(a) (b) (c) (d)

Спасибо за внимание!