

Машинное обучение и нейросетевые модели: VAE с дискретным скрытым пространством

Лектор: Кравченя Павел Дмитриевич

Волгоград 2024

План лекции

1. Вариационные автокодировщики с дискретным латентным пространством.
 2. Модель VQ-VAE. Векторная квантизация. Кодовые вектора.
 3. ELBO и функция ошибки модели VQ-VAE. Обучение VQ-VAE.
 4. Проблемы семплирования в VQ-VAE. Обучение априорного распределения.
 5. Модель VQ-VAE-2. Иерархические латентные переменные.
 6. Функция ошибки VQ-VAE-2. Обновление кодовых векторов.
 7. Задача Text-to-Image Generation. Модель DALL-E.
 8. Дискретный VAE. Обучение дискретного VAE.
 9. Двухэтапное обучение модели DALL-E. Учет описания изображений.
 10. Генерация изображений с помощью DALL-E.
-

- Классический вариационный автокодировщик имеет непрерывное латентное пространство, часто представляемое в виде нормального распределения.
 - Однако, для решения *некоторых* задач более естественным является использование вероятностной модели с дискретными латентными переменными.
 - Дискретные представления часто естественным образом подходят для сложных рассуждений и предиктивных моделей.
 - Однако, использование вариационного автокодировщика с дискретным латентным пространством традиционно осложняется сложностью вычисления производных в такой модели.
-

- Один из способов реализации *вариационного автокодировщика* с *дискретным* латентным пространством является векторная квантизация, в ходе которой множество похожих векторов (выходов кодировщика) заменяется одним. Данная модель носит название VQ-VAE (*vector quantization VAE*).
- К преимуществам VQ-VAE можно отнести относительную простоту обучения, малую дисперсию и отсутствие проблемы коллапса апостериорного распределения.
- Данная модель обеспечивает *производительность*, схожую с ее непрерывным аналогом, и имеет *гибкость*, обеспечиваемую дискретным распределением латентного пространства.

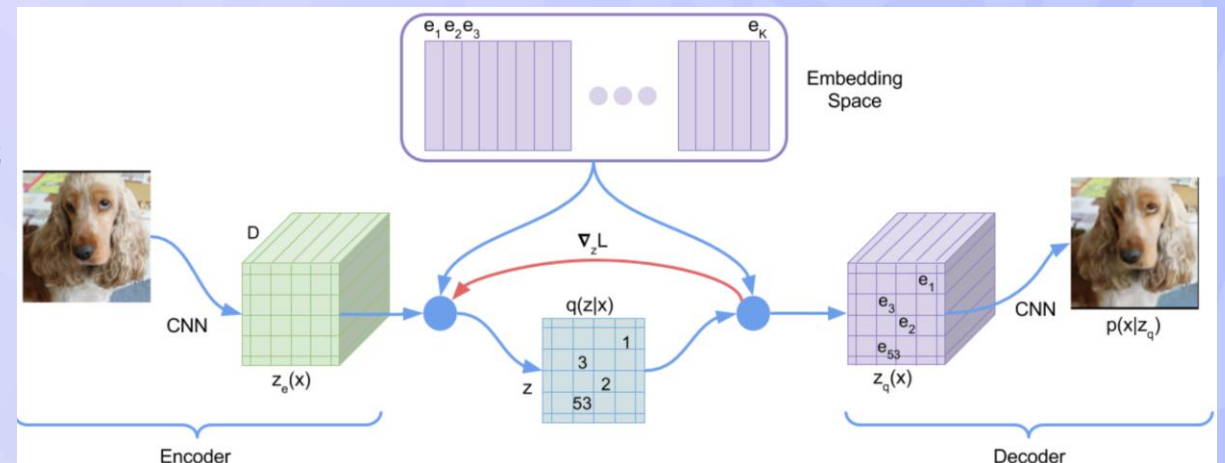
- Латентное пространство модели $\mathbf{e} \in \mathbb{R}^{K \times D}$ представляется множеством K *embedding*-векторов: $\mathbf{e}_i \in \mathbb{R}^D$, $i = 1..K$, называемых кодовыми векторами, или КОДАМИ.

$$\mathbf{z}_e(\mathbf{x}) = \text{Encoder}(\mathbf{x})$$

$$q(z = k | \mathbf{x}) = \begin{cases} 1 & \text{if } k = \arg \min_{j \in \{1..K\}} \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_j\|_2 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{z}_q(\mathbf{x}) = \mathbf{e}_k, \quad \text{где } k = \arg \min_{j \in \{1..K\}} \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_j\|_2$$

$$p(\mathbf{x} | \mathbf{z}_q) = \text{Decoder}(\mathbf{z}_q(\mathbf{x}))$$



- В качестве априорного распределения $p(\mathbf{z})$ в модели используется категориальное распределение с равными вероятностями классов:

$$p(\mathbf{z}) = \text{Cat} \left(\left[\frac{1}{K}, \frac{1}{K}, \dots, \frac{1}{K} \right] \right).$$

- Тогда:

$$\text{KL}(q(\mathbf{z} | \mathbf{x}) || p(\mathbf{z})) = - \sum_{k=1}^K q(z = k | \mathbf{x}) \log \left(\frac{p(\mathbf{z})}{q(z = k | \mathbf{x})} \right) = \log K.$$

- Поэтому, функция ошибки с учетом этого выражения примет вид:

$$\text{ELBO}(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z} | \mathbf{x})} [\log p_{\theta}(\mathbf{x} | \mathbf{z})] - \text{KL}(q(\mathbf{z} | \mathbf{x}) || p(\mathbf{z})) = \log p_{\theta}(\mathbf{x} | \mathbf{z}_q(\mathbf{x})) - \log K.$$

- Таким образом, второе слагаемое в ELBO при оптимизации можно не учитывать.

- Функция argmin не является дифференцируемой. Поэтому, при обучении VQ-VAE применяется следующий трюк: при обратном проходе градиент копируется напрямую из декодировщика в кодировщик. При этом слой, отображающий выходы кодировщика в кодовые векторы, пропускается. Данный способ расчета выполняется с помощью straight-through estimator.
- Данный подход позволяет передать градиенты кодировщику, однако, он не позволяет обучать сами кодовые векторы, поскольку по ним градиенты не вычисляются.

- Поэтому, функция ошибки VQ-VAE включает в себя три слагаемых:

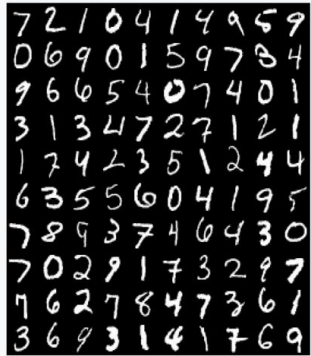
$$\mathcal{L}(\mathbf{x}) = \log p_{\theta}(\mathbf{x} | \mathbf{z}_q(x)) + \|\text{sg}[z_e(\mathbf{x})] - z_q(\mathbf{x})\|_2^2 + \beta \|z_e(\mathbf{x}) - \text{sg}[z_q(\mathbf{x})]\|_2^2.$$

- Под $\text{sg}[\cdot]$ понимается оператор остановки дифференцирования.

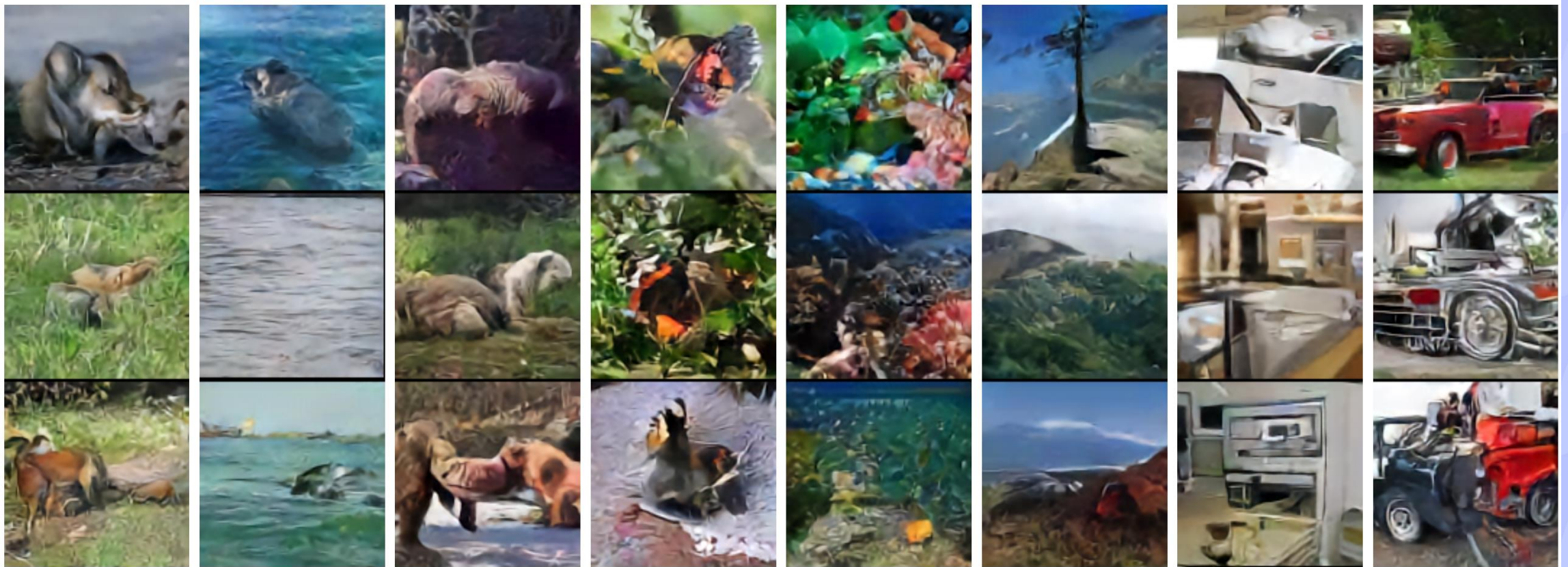
$$\mathcal{L}(\mathbf{x}) = \log p_{\theta}(\mathbf{x} | \mathbf{z}_q(x)) + \|\text{sg}[z_e(\mathbf{x})] - z_q(\mathbf{x})\|_2^2 + \beta \|z_e(\mathbf{x}) - \text{sg}[z_q(\mathbf{x})]\|_2^2.$$

- Первое слагаемое (*reconstruction loss*) представляет собой *ELBO* с *точностью до константы*.
- Второе слагаемое (*codebook loss*) отвечает за сдвиг кодовых векторов в сторону *выходов кодировщика* (поскольку первое слагаемое не позволяет обучить кодовые векторы).
- Третье слагаемое (*commitment loss*) ограничивает произвольный рост выхода кодировщика и обеспечивает ситуацию, при которой кодировщик стремится формировать векторы, близкие к кодовым векторам. *Значимость* слагаемого регулируется с помощью коэффициента β .

- При семплировании из категориального распределения с одинаковой вероятностью классов будут, скорее всего, формироваться зашумленные изображения, поскольку нет гарантий того, что обученные кодовые векторы будут распределены равномерно в латентном пространстве.
- Для решения проблемы можно с помощью дополнительной авторегрессионной модели выучить $p(\mathbf{z})$, который в дальнейшем подать на вход декодировщика VQ-VAE. Авторы VQ-VAE использовали PixelCNN для латентных векторов изображений и WaveNet – для звука.

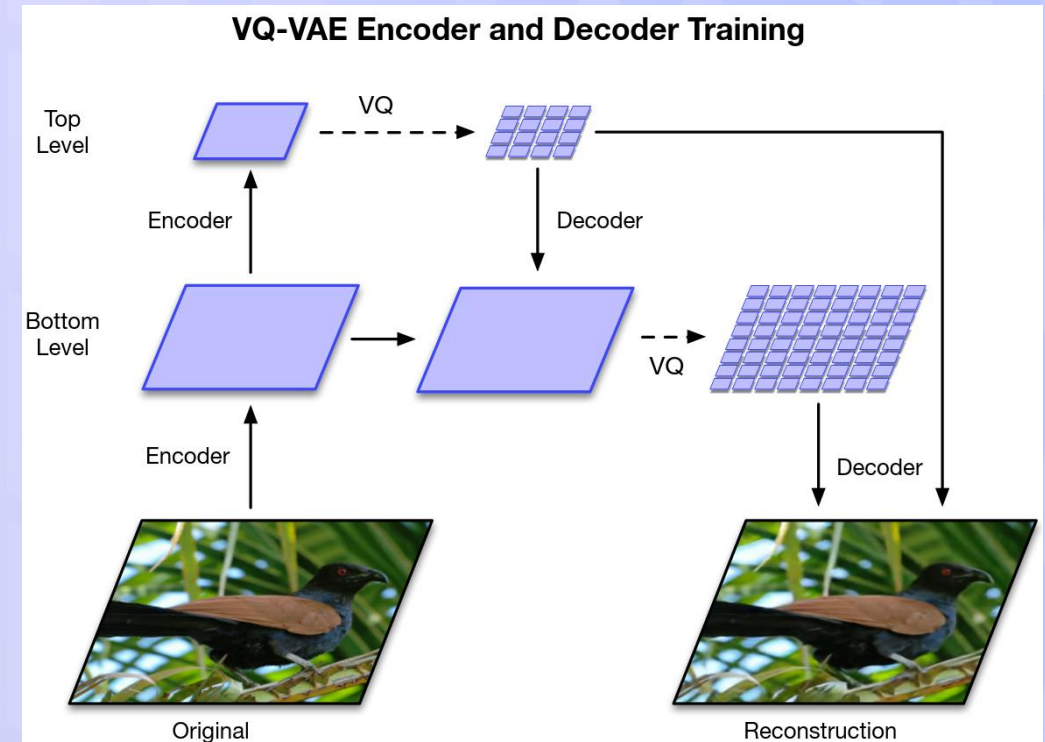
Testing data	Reconstruction	Random samples
		

- Последовательность действий для обучения и применения авторегрессионной модели:
 - Обучить VQ-VAE на выбранном датасете.
 - Закодировать объекты, пропустив их через кодировщик и сопоставив с кодовыми векторами.
 - Для каждого изображения получить матрицу индексов кодовых векторов, соответствующих изображению.
 - Обучить авторегрессионную модель на этом наборе данных.
 - Теперь можно генерировать последовательности из авторегрессионной модели и пропускать их через декодировщик VQ-VAE для получения новых объектов (например, изображений).



- Модель VQ-VAE-2 является *расширением* модели VQ-VAE. *Качество генерируемых изображений значительно повысилось.*
- Основное отличие модели VQ-VAE-2 от VQ-VAE – использование иерархических латентных переменных: архитектуре, в которой переменные расположены «по уровням».
- Каждому уровню в модели соответствуют свои *кодировщик, декодировщик и множество кодовых векторов*.
- Рассмотрим для примера модель с двумя уровнями: *верхним и нижним*.
- Обозначим кодировщики нижнего и верхнего уровня как E_{bottom} и E_{top} , а декодировщики – соответственно D_{bottom} и D_{top} .

- Процесс формирования реконструированного изображения в модели:
- Исходное изображение подается на вход E_{bottom} , который преобразует её и подает на вход E_{top} . Выходные векторы из E_{top} квантизируются в \mathbf{z}_{top} .
- \mathbf{z}_{top} передается в D_{top} , затем выходы E_{bottom} и D_{top} конкатенируются и далее квантизируются в \mathbf{z}_{bottom} .
- \mathbf{z}_{top} и \mathbf{z}_{bottom} конкатенируются и подаются на вход D_{bottom} , который уже преобразовывает их в *изображение*.



- Функция ошибки для модели VQ-VAE-2 аналогична используемой в VQ-VAE, **за исключением** второго слагаемого (*codebook loss*). В модели *codebook loss* заменяется на способ обновления кодовых векторов, базирующийся на расчете экспоненциального скользящего среднего.

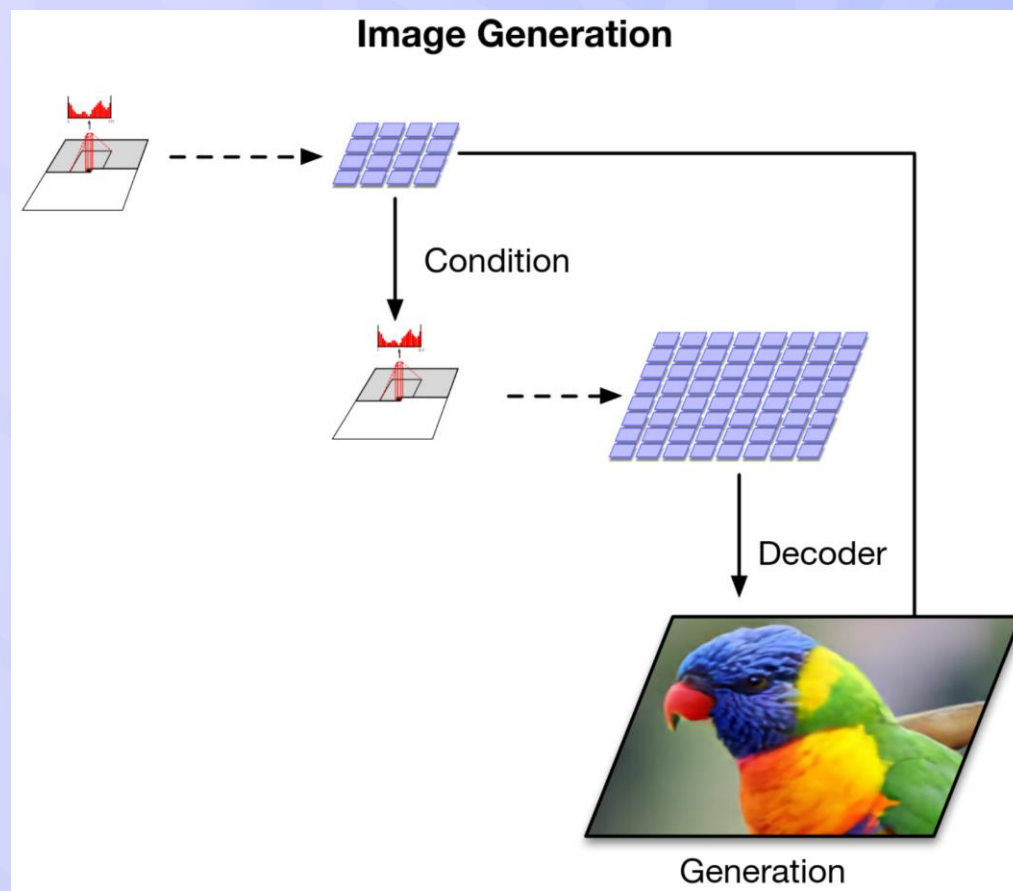
$$N_i^{(t)} = \gamma N_i^{(t-1)} + (1 - \gamma) n_i^{(t)}; \quad \mathbf{m}_i^{(t)} = \gamma \mathbf{m}_i^{(t-1)} + (1 - \gamma) \sum_{j=1}^{n_i^{(t)}} E(\mathbf{x})_{i,j}^{(t)}; \quad \mathbf{e}_i^{(t)} = \frac{\mathbf{m}_i^{(t)}}{N_i^{(t)}}.$$

Здесь $n_i^{(t)}$ – количество векторов, содержащихся в выходе декодировщика в мини-батче, которые будут квантизированы в вектора \mathbf{e}_i из латентного пространства. Параметр $\gamma \in [0,1]$ носит название decay parameter.

- Множество $\{E(\mathbf{x})_{i,j}^{(t)}\}_{i,j}$ – это множество из $n_i^{(t)}$ векторов, для которых на шаге t ближайшим оказался кодовый вектор $\mathbf{e}_i^{(t-1)}$.

- Априорное распределение $p(\mathbf{z})$ для VQ-VAE-2 обучается отдельно, уже после обучения основной модели, и также имеет иерархическую структуру. Оно используется для генерации.
- К априорному распределению для PixelCNN верхнего уровня добавляется метка класса изображения.
- На вход PixelCNN нижнего уровня подаётся метка класса изображения и вектор с верхнего уровня.

Обучение априорного распределения для модели VQ-VAE-2



Пример генерации изображений с помощью VQ-VAE-2

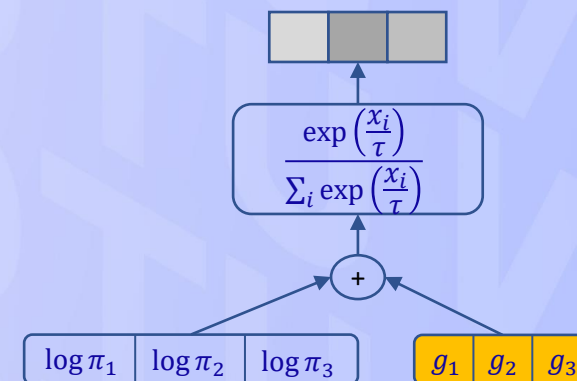
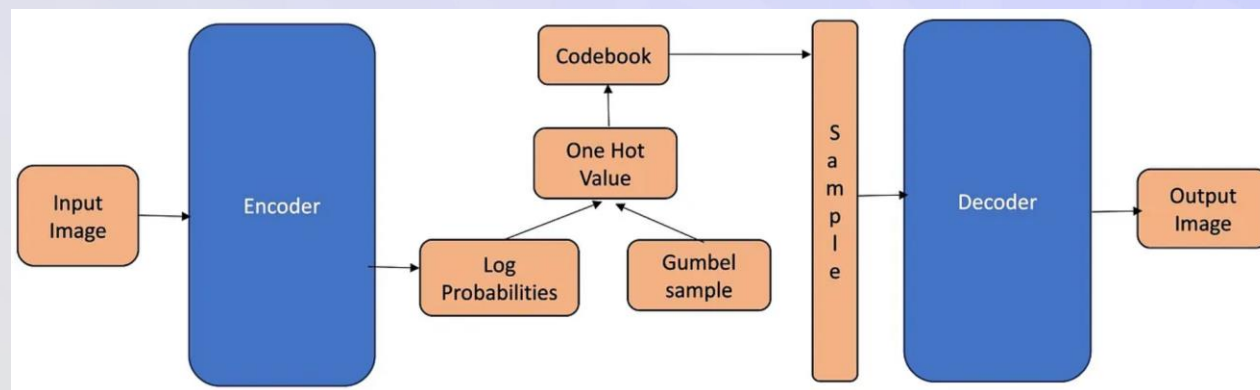
- На рисунках представлены выборки из трехуровневой иерархической модели, обученной на *FFHQ-1024×1024*.
- Модель позволяет создавать реалистичные лица, учитывая долгосрочные зависимости (совпадение цвета глаз или симметричные черты лица), охватывая при этом области данных с низкой плотностью.



- Решение задачи *Text-to-Image Generation* представляет собой создание изображения по его текстовому описанию.
- Ранние походы к решению данной задачи сводились к поиске архитектур и особенностей моделей, которые обучались на фиксированном датасете.
- Одна модель, посвященная решению этой задачи и основанная на VAE с дискретным латентным пространством, – это DALL-E от OpenAI.
- DALL-E развивает подход к генерации изображений, предложенный VQ-VAE: сначала обучаются кодировые векторы для изображений с помощью дискретного VAE, а потом обучается вспомогательная авторегрессионная модель на основе архитектуры трансформера, моделирующая совместное априорное распределение текстов и кодировых векторов.

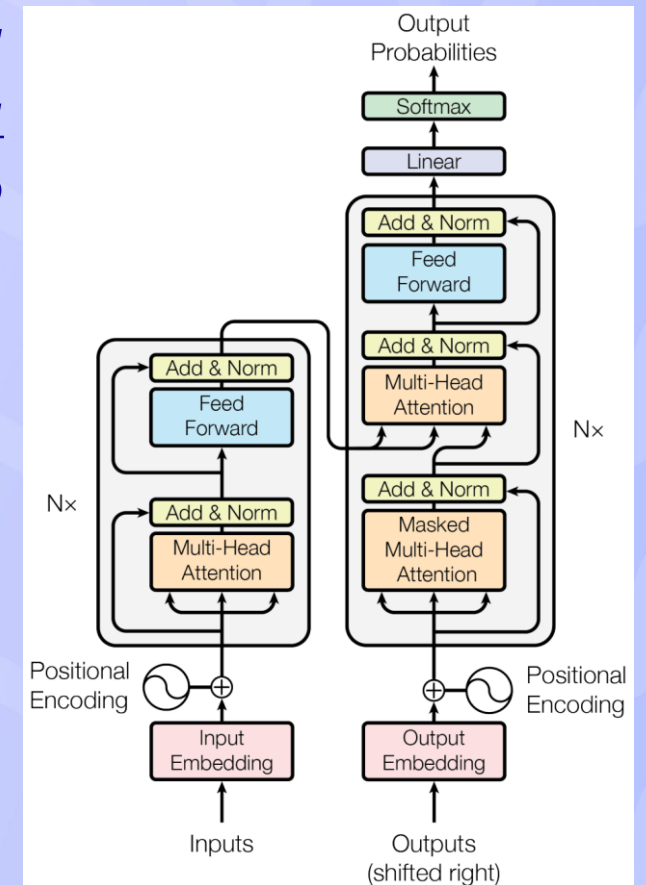
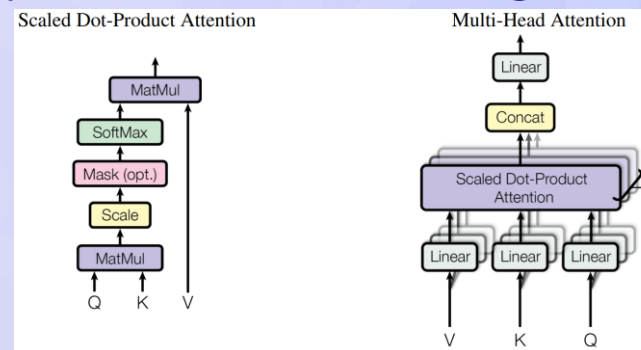
- Модель дискретного вариационного автокодировщика (dVAE) предполагает наличие VAE с дискретным латентным пространством, в котором каким-либо способом реализовано обратное распространение ошибки через дискретные скрытые переменные.
- Идея: если в декодировщик подаётся один из кодовых векторов, то почему бы с помощью кодировщика определять не вектор, близкий к кодовым векторам, а сразу вычислять вероятности их появления?
- Семплирование предполагает выбор одного из кодовых векторов. Данная операция недифференцируема. Идея: выполнить soft-sampling, который предполагает отбор не одного кодового вектора, а взвешенной суперпозиции кодовых векторов в соответствии с их вероятностями.

- Однако, данные операции нужно производить только в момент обучения. Во время генерации нужно явно семплировать один из кодовых векторов.
- Для организации такого обучения используется репараметризация распределения *Gumbel-Softmax*.
- Подобный подход позволяет распространить градиент и до кодовых векторов, что позволяет обучать их непосредственно в процессе.



- Трансформерная архитектура нейросетей основывается на реализации механизма внимания (*attention*), который позволяет вычислить корреляцию между любой парой признаков.
- Данная архитектура хорошо подходит для эффективной обработки последовательностей. Для этого требуется сформировать positional encoding.
- Блок self-attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V.$$



- Обучение модели проходит в две стадии:
 1. Обучается дискретный VAE для сжатия каждого 256 x 256 RGB-изображения в сетку кодовых векторов размером 32x32. При этом используется dVAE с параметризацией *Gumbel-Softmax*.
 2. Полученные 1024 кодовых вектора *конкатенируются* с текстовыми эмбедами и подаются на вход декодеру трансформерной модели, которая обучается по *последовательности* создавать её *продолжение*.
- Обозначим \mathcal{X} – множество изображений, \mathcal{Y} – множество текстовых описаний, \mathcal{Z} – множество латентных переменных, представленных кодовыми векторами закодированных RGB-изображений.

- Пусть $q_{\Phi}(\mathbf{z} | \mathbf{x})$ – распределение кодовых векторов, сгенерированных кодировщиком $dVAE$ для изображения \mathbf{x} .
- $p_{\theta}(\mathbf{x} | \mathbf{y}, \mathbf{z})$ обозначает распределение над RGB-изображениями, которые сгенерированы декодировщиком $dVAE$ для кодовых векторов \mathbf{z} .
- $p_{\Psi}(\mathbf{y}, \mathbf{z})$ представляет совместное распределение текста и кодовых векторов, моделируемое трансформерной моделью.
- Тогда совместное распределение модели может быть записано в факторизованном виде:

$$p_{\theta, \Psi}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = p_{\theta}(\mathbf{x} | \mathbf{y}, \mathbf{z}) \cdot p_{\Psi}(\mathbf{y}, \mathbf{z}).$$

- В процессе обучения максимизируется $ELBO$:

$$ELBO(\mathbf{x}) = \mathbb{E}_{\mathbf{z} \sim q_{\Phi}(\mathbf{z} | \mathbf{x})} [\log p_{\theta}(\mathbf{x} | \mathbf{y}, \mathbf{z})] - \beta \text{KL} \left(q_{\Phi}(\mathbf{y}, \mathbf{z} | \mathbf{x}), p_{\Psi}(\mathbf{y}, \mathbf{z}) \right).$$

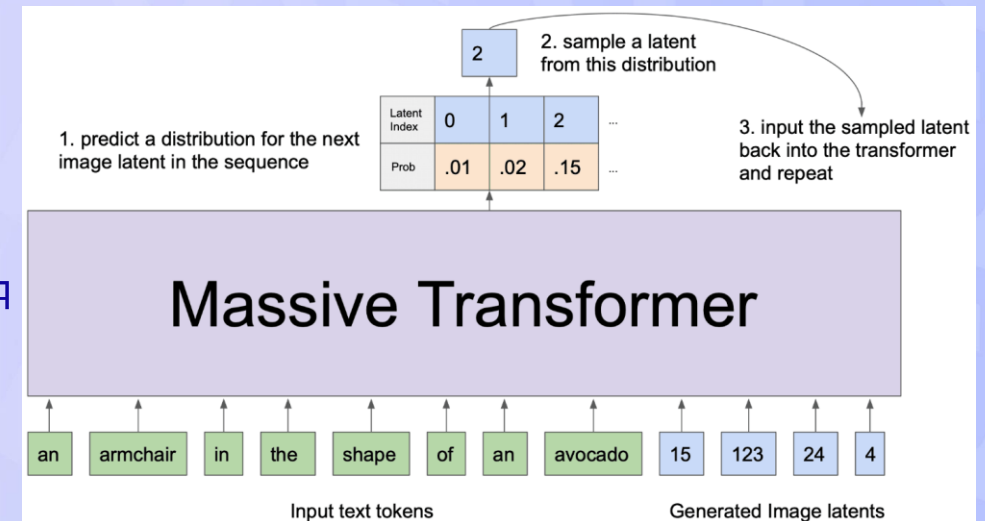
- При выборе распределений Гаусса и Лапласа для моделирования $p_{\theta}(\mathbf{x} | \mathbf{z})$ может возникать проблема, связанная с тем, что они определены на всей вещественной прямой, в то время как значения пикселей расположены в определенном интервале.
- Поэтому, при генерации изображений часть значений на выходе модели оказывается вне границ значений пикселей.
- Для исправления проблемы авторы *DALL-E* предлагают для моделирования выходов использовать распределение Logit-Laplace. Оно распределено на интервале (0,1) и выражается следующим образом:

$$p(x | \mu, b) = \frac{1}{2bx(1-x)} \exp\left(-\frac{|\text{logit}(x) - \mu|}{b}\right), \quad \text{logit}(x) = \frac{x}{1-x}.$$

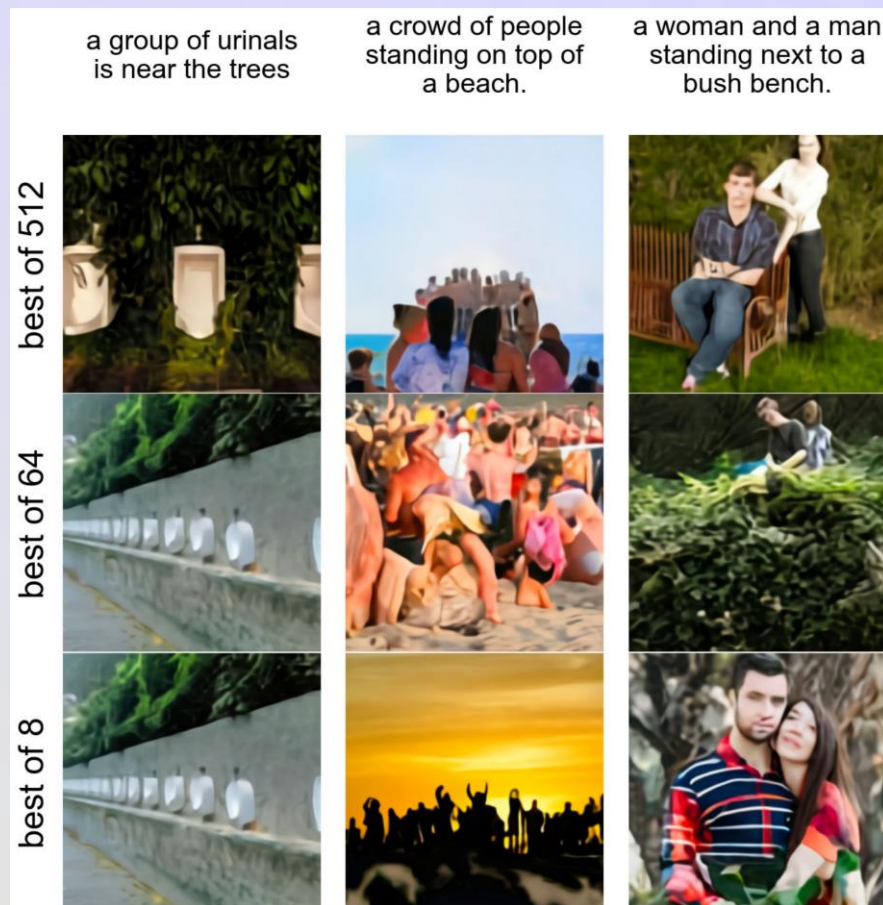
- Декодировщик выдает на выходе 6 значений: $(\mu_R, \mu_G, \mu_B, \log b_R, \log b_G, \log b_B)$.

Генерация новых изображений с помощью DALL-E

- В процессе генерации в трансформер подаются токены текстового описания изображения, на основе которых он авторегрессионно предсказывает кодовые векторы.
- Кодовые векторы затем подаются в декодер dVAE, который отображает их в финальное изображение.
- Для повышения качества предсказания сначала генерируют 512 картинок для каждого текстового описания, а затем выбирают лучшую картинку из них.
- Ранжирование полученных 512 картинок осуществляется с помощью модели CLIP.



Примеры генерации изображений с помощью DALL-E



Демонстрация практических примеров

Заключение

1. Рассмотрели подход к организации вариационных автокодировщиков: введение дискретного латентного пространства.
 2. Познакомились с моделями VQ-VAE и VQ-VAE-2, изучили их особенности.
 3. Поговорили про этап обучения априорного распределения VQ-VAE, рассмотрели роль авторегрессионной модели.
 4. Выяснили, как выполнять генерацию новых изображений в моделях VQ-VAE. На практическом примере разобрали процесс обучения и генерации.
 5. Поговорили про задачу Text-to-Image Generation, рассмотрели архитектуру DALL-E, поговорили про архитектуры dVAE и Transformer.
 6. Познакомились с процессами обучения и генерации изображений в DALL-E, разобрали особенности обучения и генерации.
-

Спасибо за внимание!

Волгоград 2024
