

ПОРОЖДАЮЩИЕ МОДЕЛИ И GAN

Сергей Николенко

СПбГУ — Санкт-Петербург

23 октября 2025 г.

Random facts:

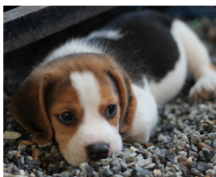


- 23 октября — День моля; он отмечается между 6:02 утра и 6:02 вечера в честь числа Авогадро 6.02×10^{23}
- 23 октября 1545 г. запорожские казаки, выйдя в море на 32 челнах-чайках, подошли к турецкой крепости Ачи-Кале (Очаков) и захватили её
- 23 октября 1748 г. открылась первая в Российской империи химическая лаборатория, основанная Ломоносовым, а 23 октября 1769 г. трёхколёсная паровая повозка впервые развила скорость в 4.5 км/ч
- 23 октября 1923 г. в Гамбурге началось коммунистическое восстание под руководством Эрнста Тельмана и Ханса Киппенбергера
- 23 октября 2002 г. группа из 40 вооружённых чеченских боевиков захватила (и затем три дня удерживала) 916 заложников из числа работников, зрителей и актёрского состава мюзикла «Норд-Ост» в здании Театрального центра
- 23 октября 2022 г. завершился XX съезд КПК, на котором генеральный секретарь ЦК КПК Си Цзиньпин был переизбран на третий срок

Порождающие и дискриминирующие модели в машинном обучении

Порождающие модели

- Итак, мы уже видели глубокое обучение и с учителем, и без.
- Но можем ли мы породить что-то новое?
- Модели бывают дискриминирующие (discriminative), которые моделируют $p(y | \mathbf{x})$:



Dog: 96%

Cat: 29%

Duck: 2%

Bird: 0%



Dog: 36%

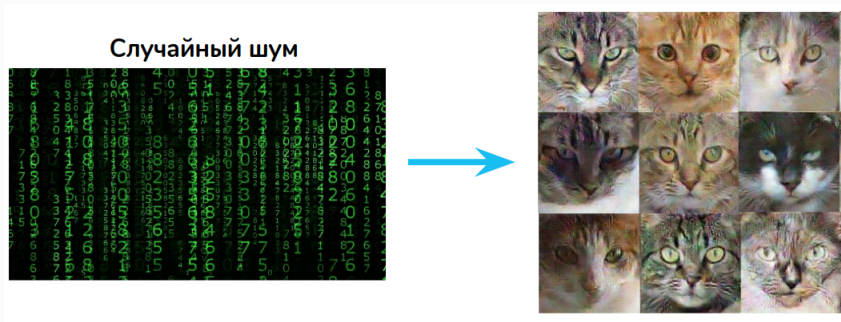
Cat: 94%

Duck: 2%

Bird: 1%

Порождающие модели

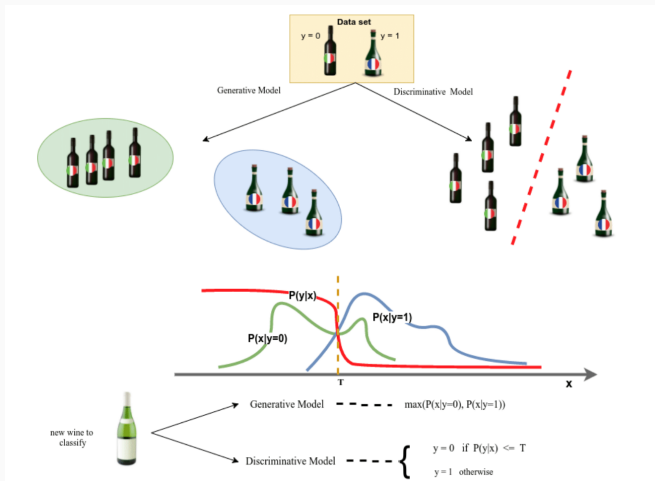
- А бывают порождающие (generative), которые моделируют $p(\mathbf{x}, y)$, и из них тогда можно сэмплировать:



- Давайте чуть подробнее. Во-первых, зачем это надо?
- $p(\mathbf{x}, y) = p(y | \mathbf{x})p(\mathbf{x})$, конечно, но разница есть.

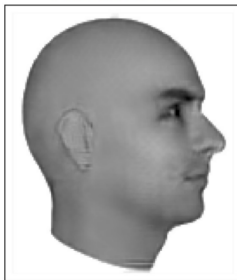
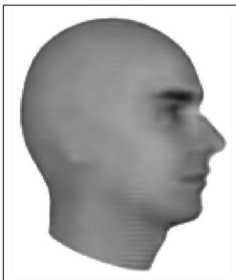
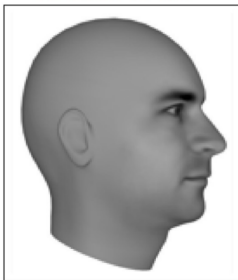
Порождающие модели

- Иллюстрация:



Порождающие модели

- Порождающие модели:
 - проверяют, насколько мы поняли распределение;
 - могут обучаться с недостатком данных и без разметки – semi-supervised learning (очень важно!);
 - могут обучаться мультимодальным выходам, когда есть несколько правильных ответов (см. ниже);
 - могут служить моделями окружающего мира в обучении с подкреплением (об этом позже);
 - ну и просто иногда действительно нужно именно порождать.



- Обычно порождающие модели максимизируют правдоподобие:

$$\theta^* = \arg \max_{\theta} \prod_{\mathbf{x} \in D} p_{\text{model}}(\mathbf{x}; \theta) = \arg \max_{\theta} \sum_{\mathbf{x} \in D} \log p_{\text{model}}(\mathbf{x}; \theta).$$

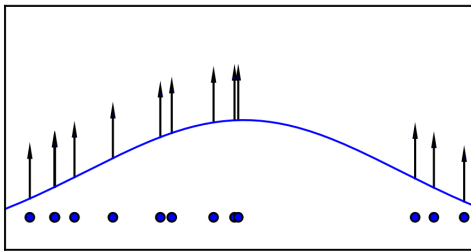
- Важный другой взгляд – это то же самое, что минимизировать KL между «распределением данных» p_{data} и p_{model} :

$$\theta^* = \arg \min_{\theta} \text{KL}(p_{\text{data}} \| p_{\text{model}}) = \arg \min_{\theta} - \int p_{\text{data}}(x) \ln \frac{p_{\text{model}}(x)}{p_{\text{data}}(x)} dx,$$

потому что p_{data} для нас дано в ощущениях через D , и это всё равно что дискретное равномерное распределение.

Порождающие модели

- Иначе говоря, точки данных «тянут» вверх распределение p_{model} .



$$\theta^* = \arg \max_{\theta} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(x \mid \theta)$$

- Только в большой размерности и не просто с гауссианами всё куда сложнее...
- Большая разница в том, предполагаем ли мы, что можем явно определить и посчитать плотность p_{model} .

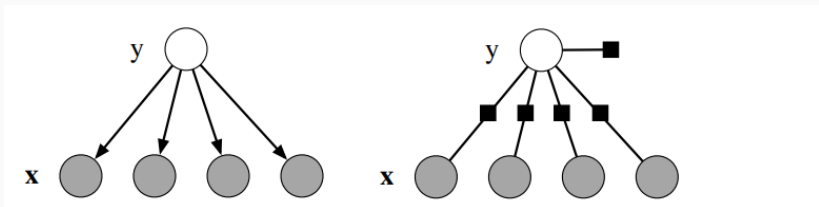
- Давайте рассмотрим пару примеров, которые дают понимание разницы между порождающими и дискриминирующими моделями.
- Заодно расскажу вам, что такое CRF, всё польза. :)
- Неожиданный заход: в чём разница между наивным Байесом и логистической регрессией?..

Порождающие и дискриминирующие модели

- Наивный Байес – это порождающая модель:

$$p(y, \mathbf{x}) = p(y) \prod_{k=1}^K p(x_k | y).$$

- Предположение в том, что признаки независимы.



ПОРОЖДАЮЩИЕ И ДИСКРИМИНИРУЮЩИЕ МОДЕЛИ

- В логистической регрессии предположение в том, что $\log p(y | \mathbf{x})$ – это линейная функция от $\mathbf{x}$:

$$p(y | \mathbf{x}) = \frac{1}{Z(\mathbf{x})} e^{\theta_y + \sum_{k=1}^K \theta_{y,k} x_k},$$

где θ – веса, θ_y – свободный член (похож на $p(y)$), $Z(\mathbf{x})$ – нормировочная константа.

- Можно, кстати, переписать в чуть более общем виде через признаки:

$$p(y | \mathbf{x}) = \frac{1}{Z(\mathbf{x})} e^{\sum_{k=1}^K \theta_k f_k(y, \mathbf{x})},$$

где $f_{k,j}(y, \mathbf{x}) = [k = y]x_j$ для векторов весов признаков и $f_k(y, \mathbf{x}) = [k = y]$ для bias weights.

- Но из наивного Байеса тоже можно получить $p(y | \mathbf{x})$, ведь $p(y, \mathbf{x}) = p(\mathbf{x})p(y | \mathbf{x})...$

Порождающие и дискриминирующие модели

- Более того, действительно получится то же самое. У них одно и то же пространство гипотез, и одно можно перевести в другое:
 - если обучать наивного Байеса максимизировать условное правдоподобие, получится логистическая регрессия;
 - а если интерпретировать логистическую регрессию как порождающую модель с

$$p(y, \mathbf{x}) = \frac{\exp \left(\sum_{k=1}^K \theta_k f_k(y, \mathbf{x}) \right)}{\sum_{y', \mathbf{x}'} \exp \left(\sum_{k=1}^K \theta_k f_k(y', \mathbf{x}') \right)},$$

то получится тот же классификатор, что из наивного Байеса.

- Наивный Байес и логистическая регрессия образуют generative-discriminative pair.

Порождающие и дискриминирующие модели

- Т.е. единственная разница в том, что наивный Байес – порождающая модель, а логистическая регрессия – дискриминирующая.
- Если бы мы могли получить идеальную модель $p^*(\mathbf{y}, \mathbf{x}) = p^*(\mathbf{y})p^*(\mathbf{y} | \mathbf{x})$, то не было бы никакой разницы.
- Но на практике есть разница, оценивать ли $p(\mathbf{y})p(\mathbf{x} | \mathbf{y})$, а затем вычислять из этого $p(\mathbf{y} | \mathbf{x})$, или сразу напрямую оценивать $p(\mathbf{y} | \mathbf{x})$.
- Порождающие модели могут больше, но у дискриминирующих больше свободы.

ПОРОЖДАЮЩИЕ И ДИСКРИМИНИРУЮЩИЕ МОДЕЛИ

- Например, пусть есть порождающая модель с параметрами θ :

$$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{y}; \theta) p_g(\mathbf{x} | \mathbf{y}; \theta).$$

- Можно её переписать как $p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{x}; \theta) p_g(\mathbf{x} | \mathbf{y}; \theta)$, где $p_g(\mathbf{x}; \theta) = \sum_{\mathbf{y}} p_g(\mathbf{y}, \mathbf{x}; \theta)$, $p_g(\mathbf{y} | \mathbf{x}; \theta) = p_g(\mathbf{y}, \mathbf{x}; \theta) / p_g(\mathbf{x}; \theta)$.
- А соответствующая ей дискриминирующая модель должна иметь априорное распределение $p(\mathbf{x})$, которое может получиться из неё, т.е. $p(\mathbf{x}) = p_c(\mathbf{x}; \theta') = \sum_{\mathbf{y}} p_g(\mathbf{y}, \mathbf{x}; \theta')$, и условное распределение, которое может получаться как $p_c(\mathbf{y} | \mathbf{x}; \theta) = p_g(\mathbf{y}, \mathbf{x}; \theta) / p_g(\mathbf{x}; \theta)$:

$$p_c(\mathbf{y}, \mathbf{x}) = p_c(\mathbf{x}; \theta') p_c(\mathbf{y} | \mathbf{x}; \theta).$$

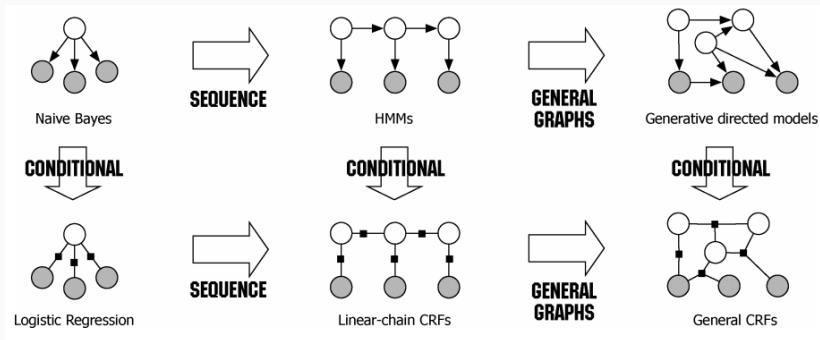
- Разница в том, что теперь не обязательно $\theta = \theta'$, и дискриминирующая модель более выразительна.

- В результате мы будем хуже моделировать $p(\mathbf{x})$ (на которое в классификации нам наплевать), и лучше подходить к $p(\mathbf{y} \mid \mathbf{x})$ (но и риск оверфиттинга выше).
- Другой пример порождающей модели – скрытая марковская модель: моделируем наблюдаемые $X = \{x_t\}_{t=1}^T$, предполагая, что есть скрытые состояния $Y = \{y_t\}_{t=1}^T$, и делаем предположения о независимости:

$$p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^T p(y_t \mid y_{t-1}) p(x_t \mid y_t).$$

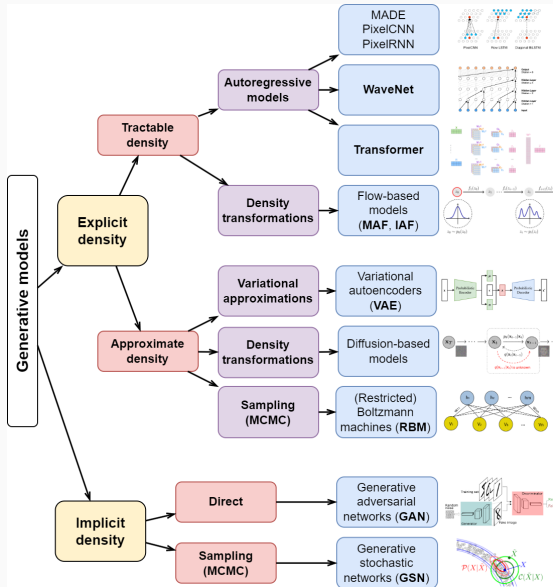
Порождающие и дискриминирующие модели

- Так вот, CRF – это дискриминирующие модели, и линейная CRF – это дискриминирующая модель, соответствующая HMM:



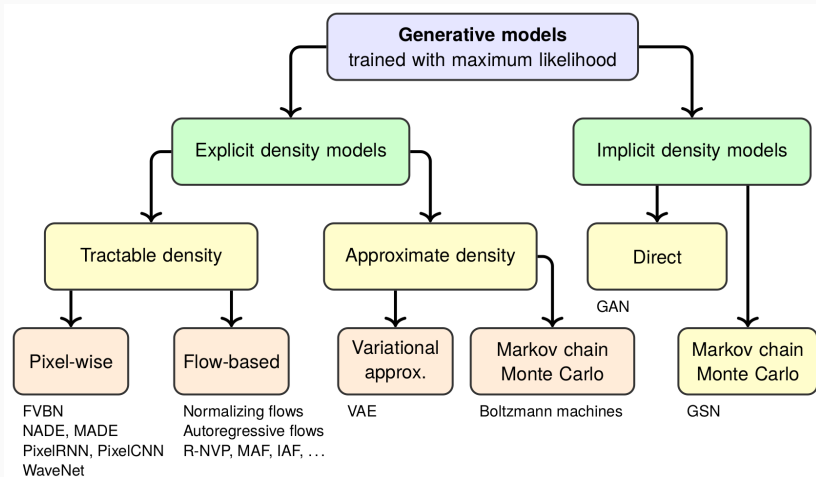
Порождающие модели в глубоком обучении

ГЛУБОКИЕ ПОРОЖДАЮЩИЕ МОДЕЛИ



Порождающие модели в DL

- Теперь возвращаемся к глубокому обучению. Общая таксономия:



- Если можем явно выразить порождающую плотность (explicit), то, например, FVBN (fully visible belief networks):

$$p_{\text{model}}(\mathbf{x}) = \prod_{i=1}^n p_{\text{model}}(x_i \mid x_1 \dots, x_{i-1}).$$

- А с одномерными распределениями уж как-нибудь разберёмся, например промоделируем нейронной сетью.
- Появились в конце 90-х (Frey et al., 1996; Frey, 1998).

- (Germain et al., 2015): MADE, Masked Autoencoder for Distribution Estimation
- Пример авторегрессионной модели, основанной на автокодировщиках:

$$\ell(\mathbf{x}) = \sum_{d=1}^D (-x_d \log \hat{\mathbf{x}}_d - (1 - x_d) \log(1 - \hat{\mathbf{x}}_d)),$$

где $\hat{\mathbf{x}}$ – бинарный выход, порождённый моделью.

- Теперь вопрос: можно ли сделать так, чтобы выход интерпретировался как вероятность?

- Можно! Начинаем с той же идеи:

$$p_{\text{model}}(\mathbf{x}) = \prod_{i=1}^n p(x_i \mid x_1 \dots, x_{i-1}) = p(x_i \mid \mathbf{x}_{<i}).$$

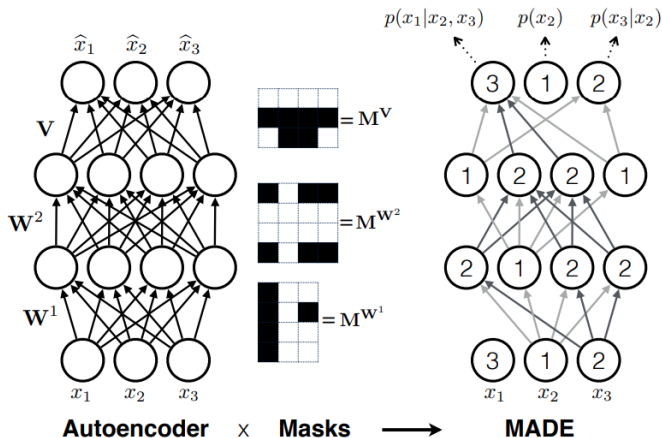
- И теперь возьмём $\hat{\mathbf{x}}_d = p(x_i = 1 \mid \mathbf{x}_{<i})$, и получится правдоподобие

$$-\log p(\mathbf{x}) = \sum_{d=1}^D (-x_d \log p(x_d = 1 \mid \mathbf{x}_{<d}) - (1 - x_d) \log p(x_d = 0 \mid \mathbf{x}_{<d})).$$

- Осталось позаботиться о том, чтобы на следующие пиксели не слишком смотреть.

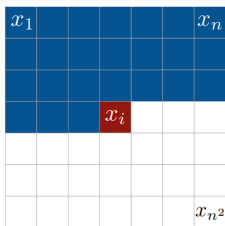
Порождающие модели в DL

- А этого можно достичь масками, причём их можно выбирать так, чтобы в случайном порядке раскладывая выход:

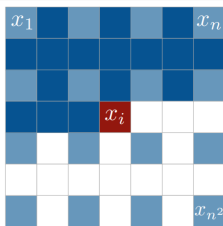


Порождающие модели в DL

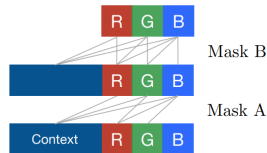
- (van den Oord, 2016a; 2016b): давайте порождать картинку пиксель за пикселем! Моделируем сеть распределение $p(x_i \mid x_1, \dots, x_{i-1})$



Context

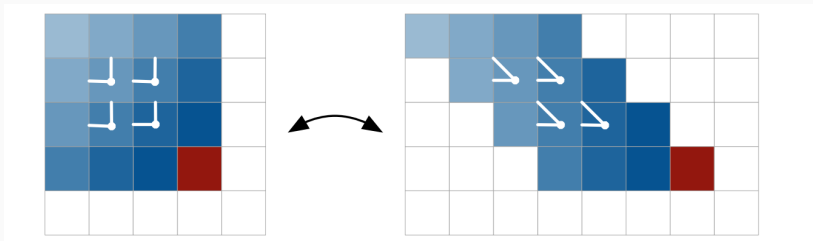
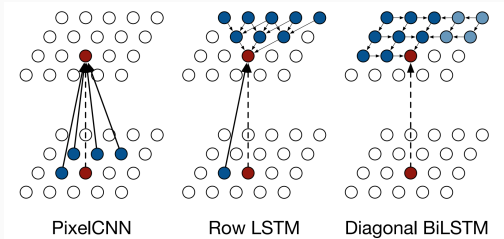


Multi-scale context



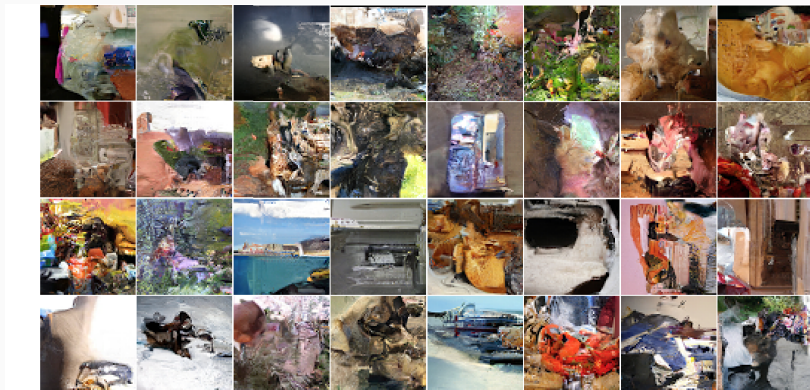
Порождающие модели в DL

- Можно моделировать свёрточную сеть или рекуррентной:



Порождающие модели в DL

- (van den Oord, 2016a) – PixelRNN:



ПОРОЖДАЮЩИЕ МОДЕЛИ В DL

- (van den Oord, 2016a) – PixelRNN:



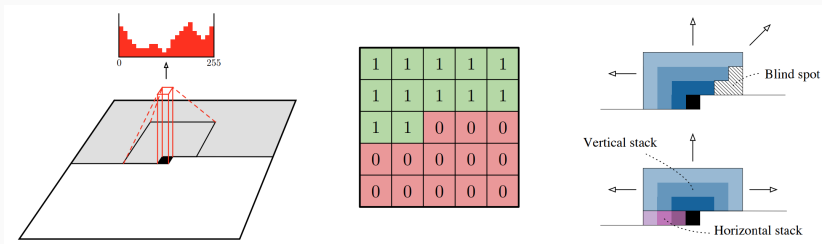
Порождающие модели в DL

- (van den Oord, 2016b) – PixelCNN; точно так же

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

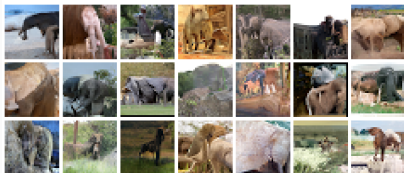
но теперь будем свёрточной сетью моделировать.

- Нужны маски, чтобы не забегать вперёд:

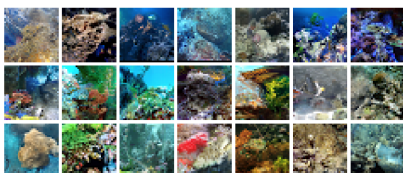


ПОРОЖДАЮЩИЕ МОДЕЛИ В DL

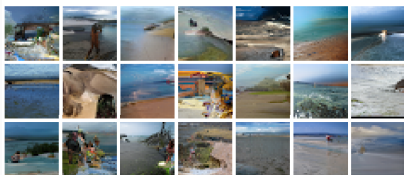
- (van den Oord, 2016b) – PixelCNN:



African elephant



Coral Reef



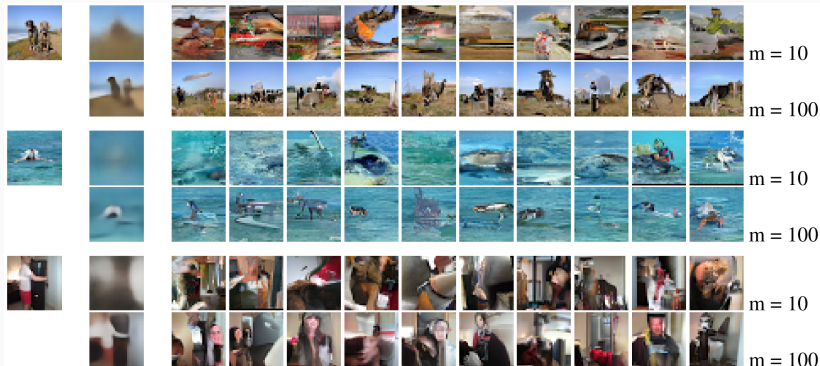
Sandbar



Sorrel horse

Порождающие модели в DL

- (van den Oord, 2016b) – PixelCNN autoencoder:



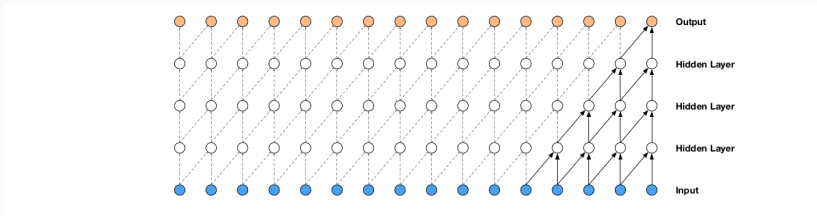
- Другой современный пример FVBN – WaveNet (van den Oord et al., 2016).
- Мы обычно мало говорим о звуке в DL, да и работ мало, WaveNet одна из немногих.
- Как породить, например, человеческую речь?
- Основная идея – будем моделировать условное распределение

$$p(\mathbf{x} \mid \mathbf{h}) = \prod_{t=1}^T p(x_t \mid x_1, \dots, x_{t-1}, \mathbf{h}).$$

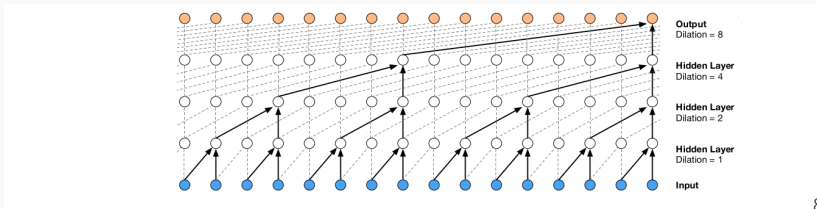
- На звуковых данных полезно делать одномерные свёртки, но свёртки не должны «забегать вперёд» по времени...

WaveNet и ПОРОЖДЕНИЕ ЗВУКА

- ...поэтому в WaveNet *каузальные свёртки* (causal convolutions):

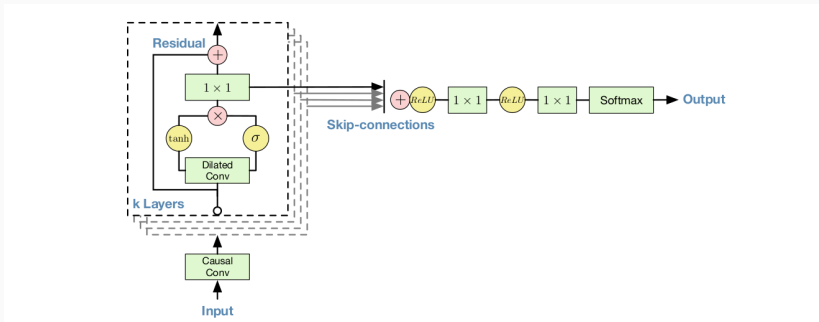


- Их лучше прореживать:



WaveNet и ПОРОЖДЕНИЕ ЗВУКА

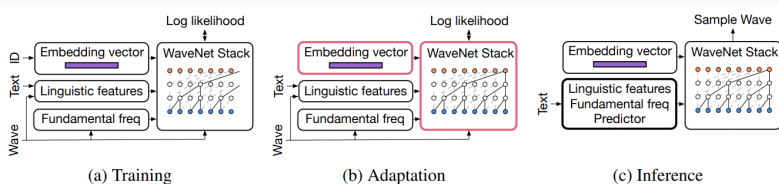
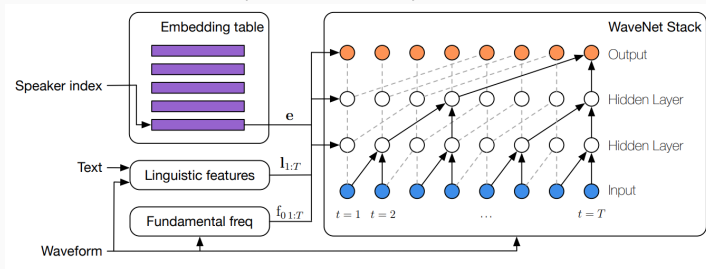
- А дальше архитектура с трюками, которые мы уже знаем – residual connections, skip-layer connections...



- Получалось очень хорошо ещё в 2016:
<https://deepmind.com/blog/wavenet-generative-model-raw-audio/>

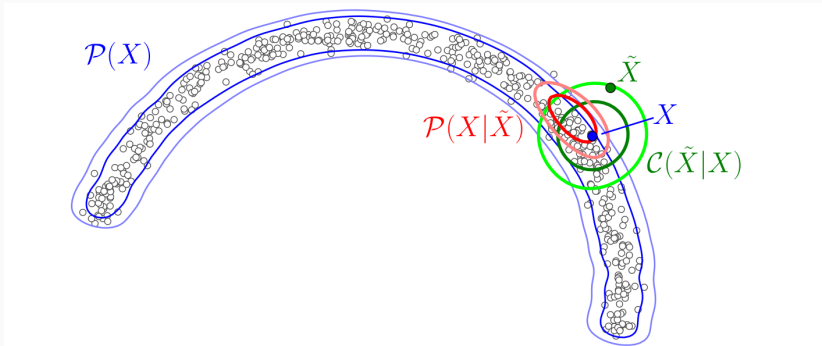
WaveNet и ПОРОЖДЕНИЕ ЗВУКА

- Сейчас есть несколько важных работ про latent representations для речи, основанные на WaveNet
- (Chen et al., 2019): adaptive text-to-speech



GSN КАК ПРИМЕР НЕЯВНЫХ МОДЕЛЕЙ

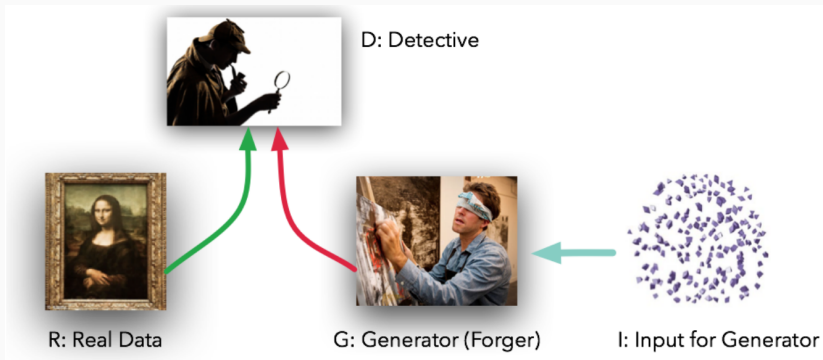
- А для неявных (implicit) порождающих моделей мы моделируем обычно *процесс сэмплирования*.
- Сэмплировать можно марковской цепью – если её моделировать нейронной сетью, получится Generative Stochastic Network (Alain et al., 2015):



- Но мы пойдём другим путём...

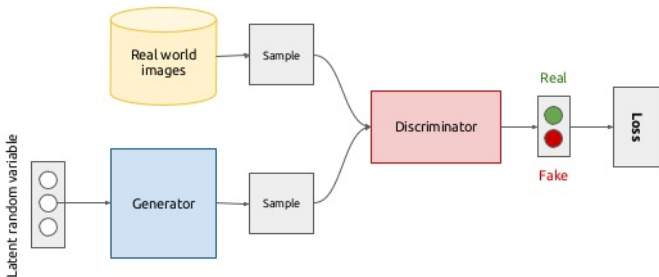
Порождающие состязательные сети

- Порождающие состязательные сети (Generative adversarial networks, GAN): генератор vs. дискриминатор, p_{data} vs. p_g .

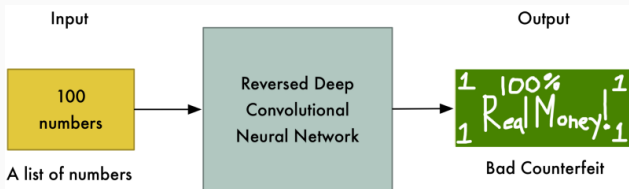


- Порождающие состязательные сети (Generative adversarial networks, GAN): генератор vs. дискриминатор, p_{data} vs. p_g .

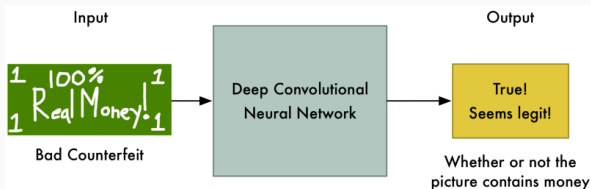
Generative adversarial networks (conceptual)



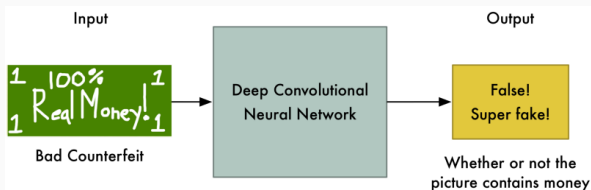
- В идеале должно быть примерно так (Geitgey, 2017):
 - сначала генератор плохой:



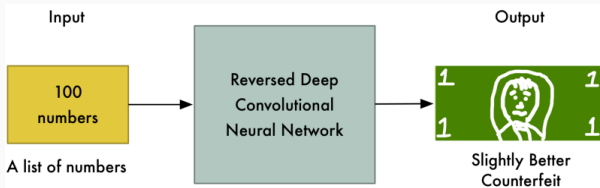
- но и дискриминатор ничего не умеет:



- В идеале должно быть примерно так (Geitgey, 2017):
 - поэтому сначала чуть обучим дискриминатор:



- ...и генератору придётся придумать что-то получше:



- И так они будут продолжать, пока всё не получится.

- Формально, генератор – это функция

$$G = G(\mathbf{z}; \theta_g) : Z \rightarrow X,$$

а дискриминатор – это функция

$$D = D(\mathbf{x}; \theta_d) : X \rightarrow [0, 1].$$

- Целевая функция для дискриминатора – это бинарная классификация:

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g(\mathbf{x})} [\log(1 - D(\mathbf{x}))],$$

где $p_g(\mathbf{x}) = G_{\mathbf{z} \sim p_z}(\mathbf{z})$ – распределение, порождённое G ; то есть мы берём два мини-батча, с метками 0 из генератора и с метками 1 из данных.

- А генератор обучается обманывать дискриминатор, то есть минимизировать

$$\mathbb{E}_{\mathbf{x} \sim p_g(\mathbf{x})} [\log(1 - D(\mathbf{x}))] = \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))] .$$

- И теперь, если их объединить, получится типичная минимакс-игра:

$$\min_G \max_D V(D, G), \text{ где}$$

$$V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))].$$

- Можно доказать, что $\max_D V(D, G)$ минимизируется именно тогда, когда $p_g = p_{\text{data}}$.
- Важное свойство: для фиксированного генератора G оптимальное распределение дискриминатора D — это распределение

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

- (Упражнение: попробуйте это доказать)

- Глобальный минимум критерия достигается тогда и только тогда, когда $p_g = p_{\text{data}}$; сам критерий для оптимального D эквивалентен

$$\text{KL} \left(p_{\text{data}} \left\| \frac{p_{\text{data}} + p_g}{2} \right\| \right) + \text{KL} \left(p_g \left\| \frac{p_{\text{data}} + p_g}{2} \right\| \right),$$

это т.н. *дивергенция Йенсена–Шеннона* (Jensen–Shannon divergence).

- Но это всё результаты для ошибки генератора $E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.
- А это не самая лучшая функция ошибки для генератора...

- Чем плохо минимизировать $E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$?
 - перекрёстная энтропия хороша в классификаторе: если ответ неправильный, она не насыщается, а если правильный, насыщается и не меняется;
 - тут получается, что когда дискриминатор хорошо работает, его градиент обнуляется...
 - ...и градиент генератора тоже обнуляется!
- Поэтому на самом деле минимизируют

$$-E_{z \sim p_z(z)}[\log D(G(z))],$$

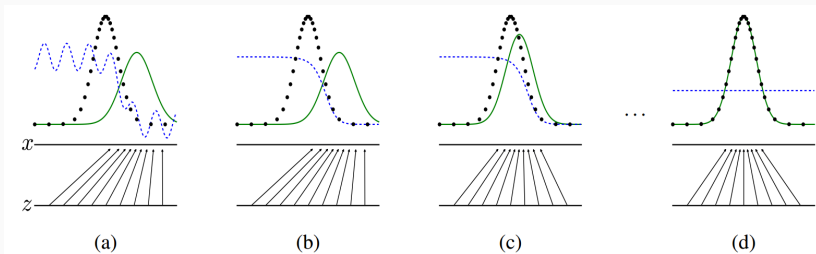
т.е. максимизируем вероятность того, что D ошибся, а не минимизируем вероятность его правильного ответа.

- Обучение похоже по сути на EM-алгоритм – попеременно:
 - фиксируем G , обновляем дискриминатор с функцией ошибки

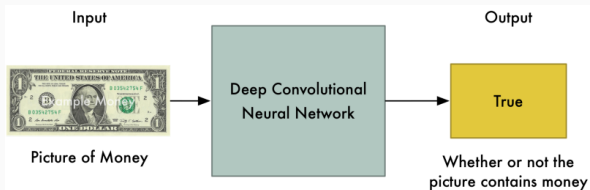
$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g(\mathbf{x})} [\log(1 - D(\mathbf{x}))],$$

- фиксируем D , обновляем генератор с функцией ошибки

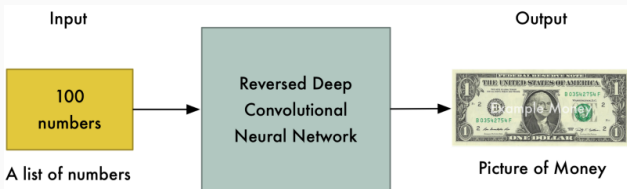
$$-\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})],$$



- В идеале должно быть примерно так (Geitgey, 2017):
 - дискриминатор пытается распознавать:



- а генератор хочет породить:



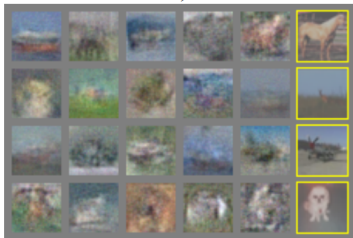
- Сэмплирование из GAN (Goodfellow et al., 2014):



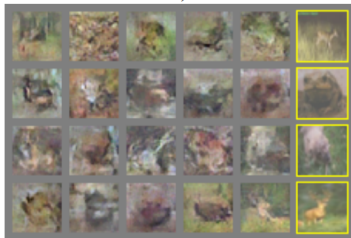
a)



b)



c)



d)

СПАСИБО!

Спасибо за внимание!

