

ПОРОЖДАЮЩИЕ СОСТАЗАТЕЛЬНЫЕ СЕТИ

Сергей Николенко

Академия MADE – Mail.Ru

20 ноября 2021 г.

Random facts:

- 20 ноября в ООН – Всемирный день ребёнка; в этот день в 1959 г. была принята «Декларация прав ребёнка», а в 1989 — «Конвенция прав ребёнка»
- 20 ноября 1910 г. Франсиско Мадеро опубликовал «план Сан-Луис-Потоси», в котором объявлял результаты президентских выборов недействительными и призывал к борьбе с режимом Порфирио Диаса, правившего к тому времени 34 года, чем запустил Мексиканскую революцию; а в доме Ивана Озолина, начальника станции Астапово, в тот же день умер Лев Толстой
- 20 ноября 1947 г. в Вестминстерском аббатстве принцесса Елизавета вышла замуж за лейтенанта Филипа Маунтбэттена, который при этом стал герцогом Эдинбургским
- 20 ноября 1979 г. у стен Каабы появился человек и заявил, что настало время для исполнения древнего пророчества, во имя которого в пределах Запретной Мечети была пролита кровь; захват заложников длился до 4 декабря, погибло 255 человек
- 20 ноября 1985 г. вышла первая графическая операционная система от Microsoft, Windows 1.0

WAVENET

WAVENET И ПОРОЖДЕНИЕ ЗВУКА

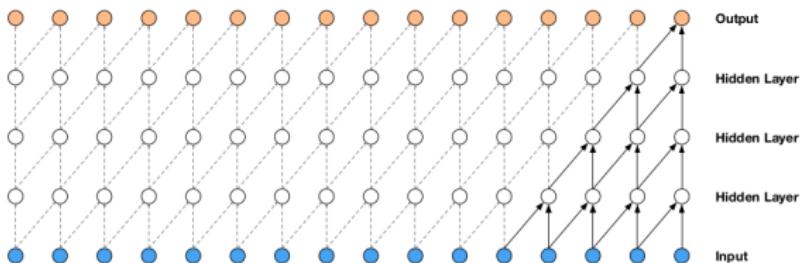
- Другой современный пример FVBN – WaveNet (van den Oord et al., 2016).
- Мы обычно мало говорим о звуке в DL, да и работ мало, WaveNet одна из немногих.
- Как породить, например, человеческую речь?
- Основная идея – будем моделировать условное распределение

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

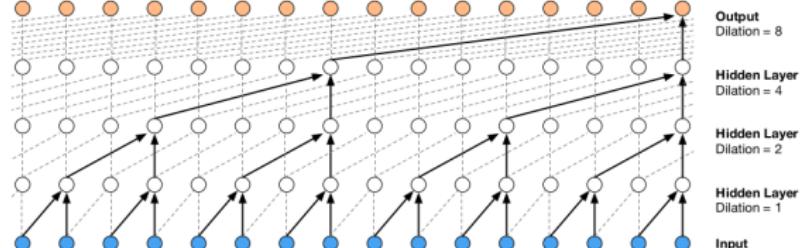
- На звуковых данных полезно делать одномерные свёртки, но свёртки не должны «забегать вперёд» по времени...

WAVENET И ПОРОЖДЕНИЕ ЗВУКА

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

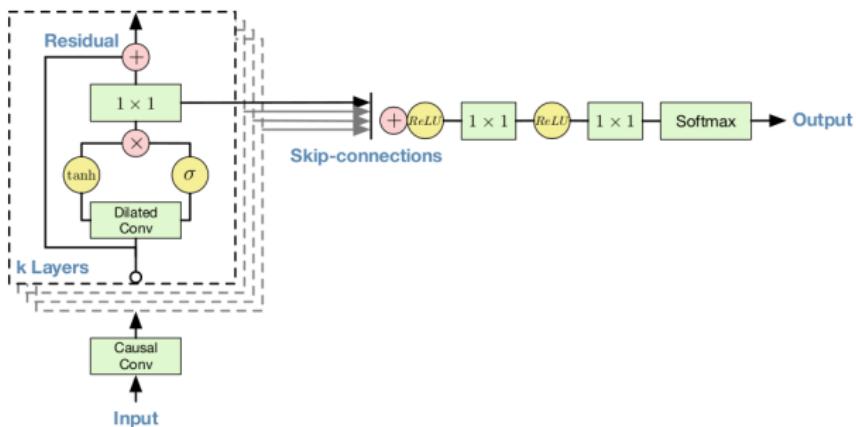


- Их лучше прореживать:



WAVENET И ПОРОЖДЕНИЕ ЗВУКА

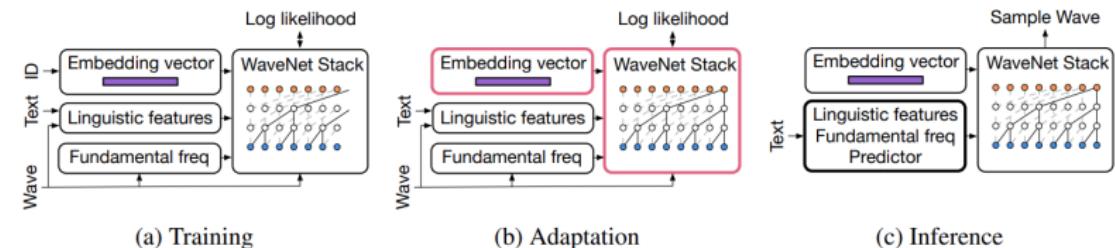
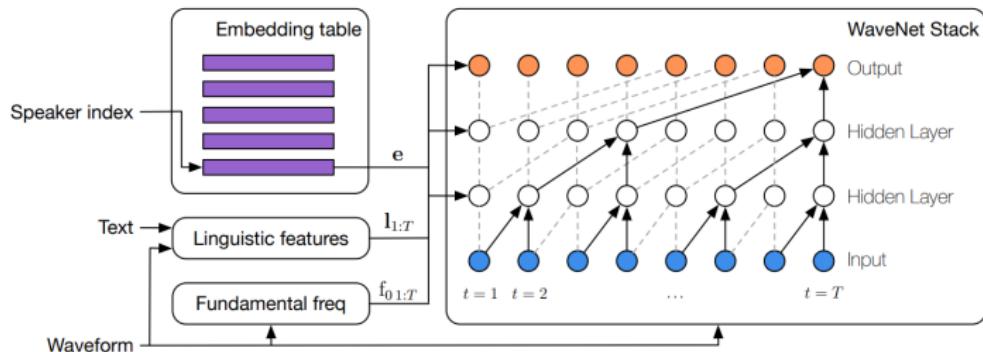
- А дальше архитектура с трюками, которые мы уже знаем – residual connections, skip-layer connections...



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

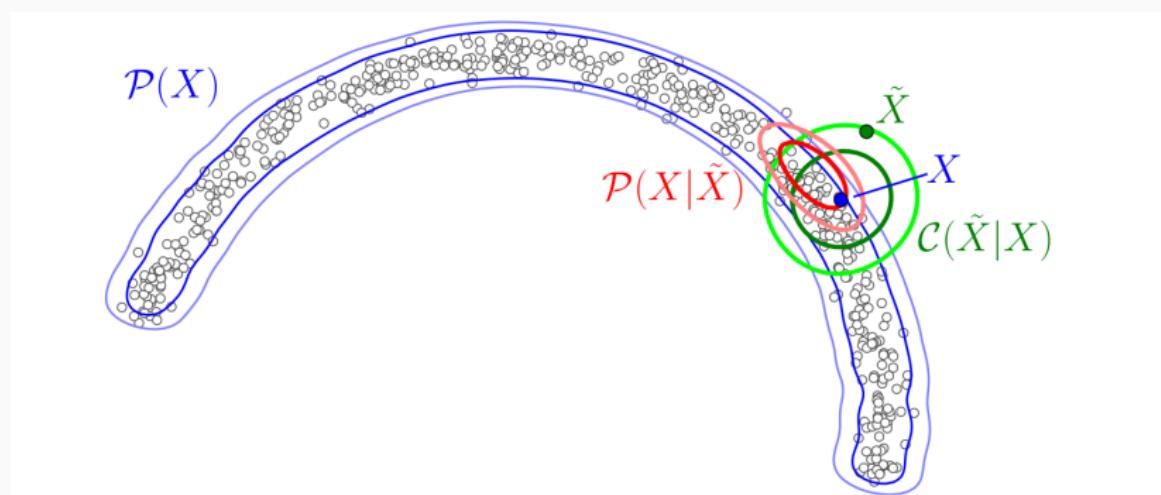
WAVENET И ПОРОЖДЕНИЕ ЗВУКА

- 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

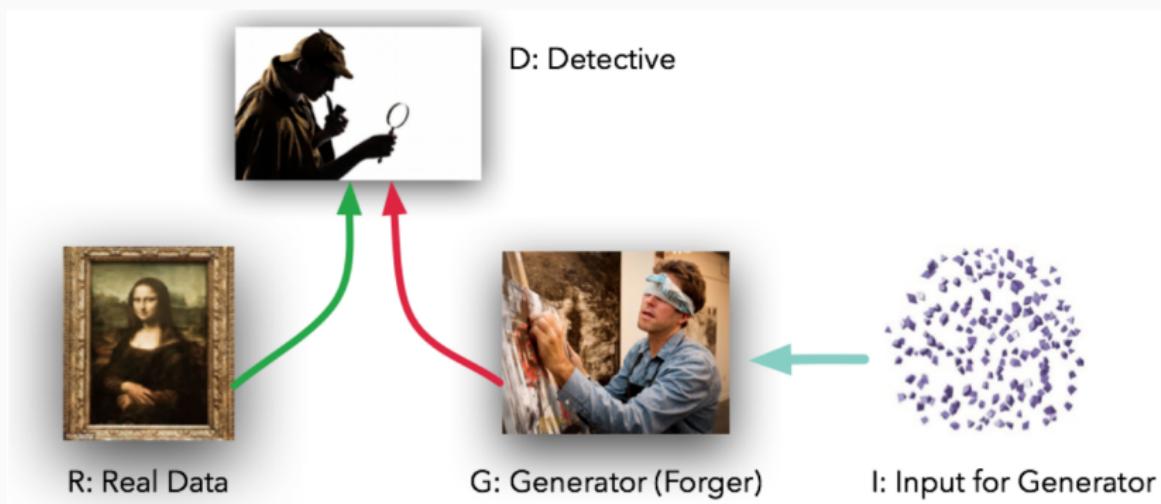
- А для неявных (implicit) порождающих моделей мы моделируем обычно процесс сэмплирования.
- Сэмплировать можно марковской цепью – если её моделировать нейронной сетью, получится Generative Stochastic Network (Alain et al., 2015):



- Но мы пойдём другим путём...

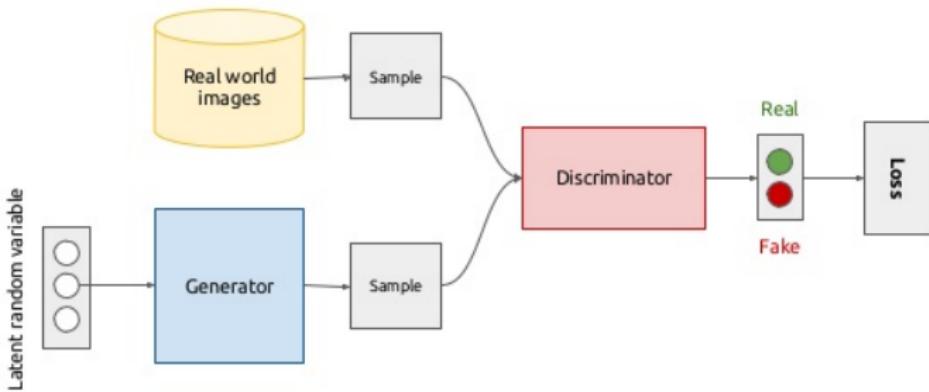
ПОРОЖДАЮЩИЕ СОСТАЗАТЕЛЬНЫЕ СЕТИ

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

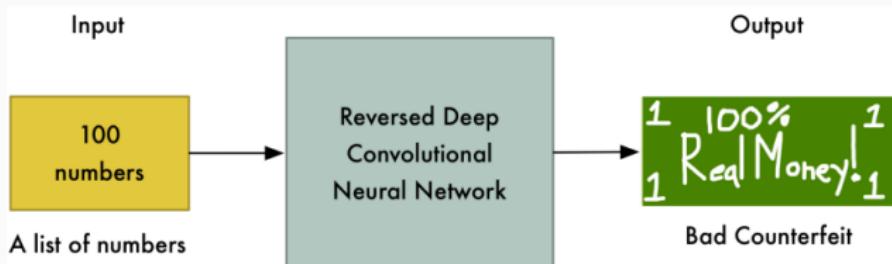


- Порождающие состязательные сети (Generative adversarial networks, GAN): генератор vs. дискриминатор, p_{data} vs. p_g .

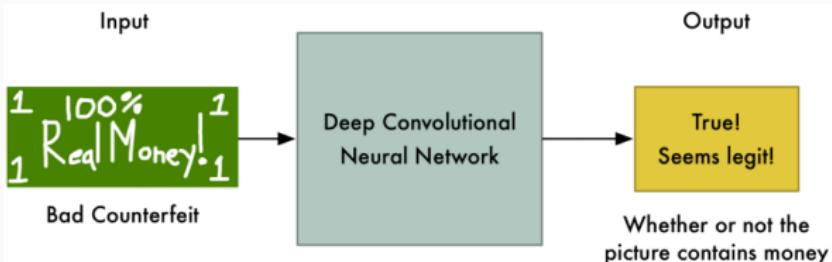
Generative adversarial networks (conceptual)



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

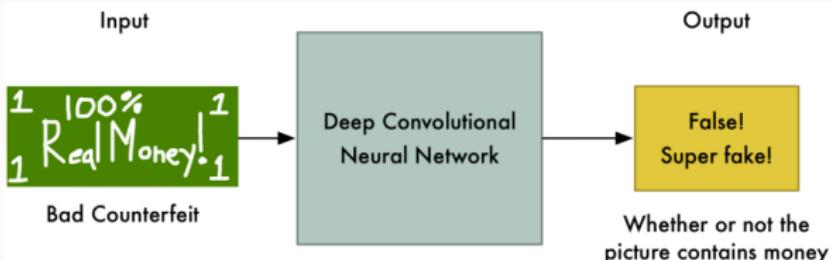


- но и дискриминатор ничего не умеет:

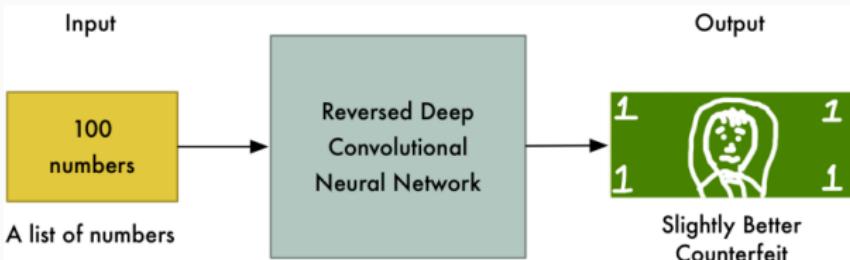


GAN

- 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

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

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

- Но это всё результаты для ошибки генератора $\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.
- 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

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

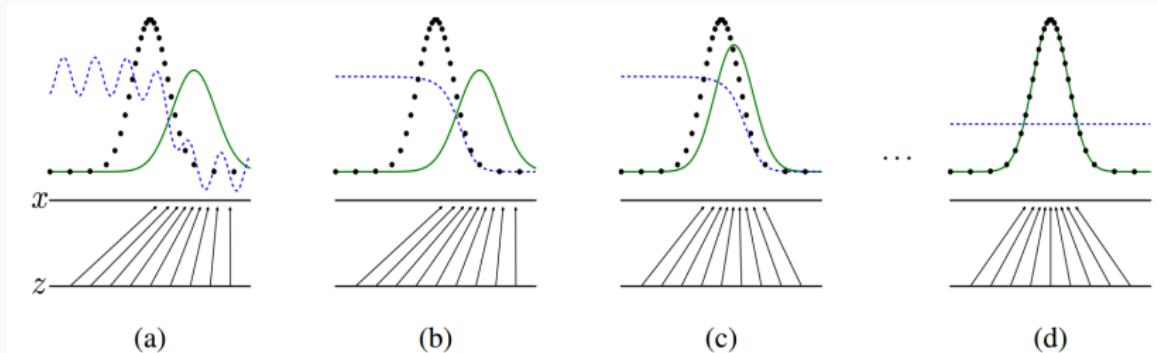
т.е. максимизируем вероятность того, что D ошибся, а не минимизируем вероятность его правильного ответа.

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

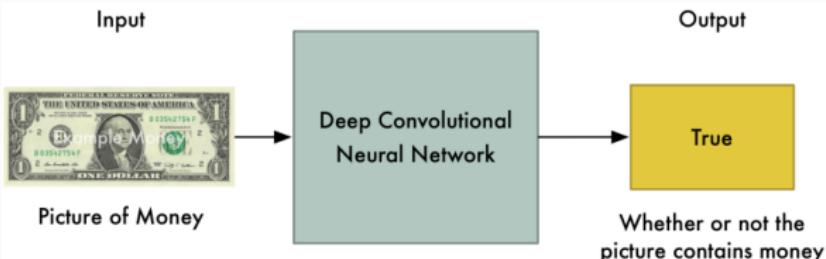
$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g(\mathbf{x})} [\log(1 - D(\mathbf{x}))],$$

- фиксируем D , обновляем генератор с функцией ошибки

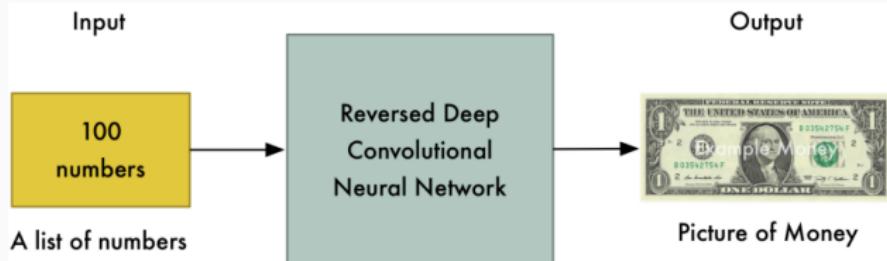
$$-\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})],$$



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



- а генератор хочет порождать:



GAN

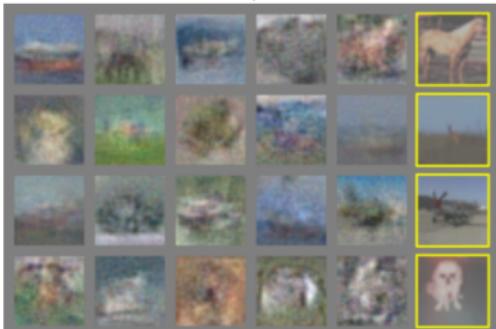
- Сэмплирование из GAN (Goodfellow et al., 2014):



a)



b)



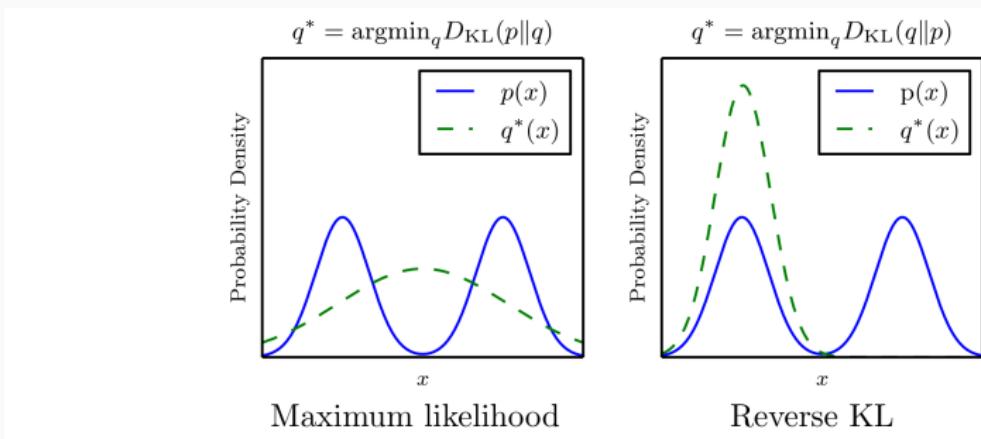
c)



d)

ПОЧЕМУ И КАК GAN РАБОТАЕТ?

- Почему GAN хорошо работает?
 - возможно, потому что минимакс эквивалентен не $\text{KL}(p_{\text{data}} \| p_{\text{model}})$, а дивергенции Йенсена–Шеннона, что больше похоже на $\text{KL}(p_{\text{model}} \| p_{\text{data}})$:



ПОЧЕМУ И КАК GAN РАБОТАЕТ?

- Почему GAN хорошо работает?
 - но нет, можно и другие функции ошибки брать (максимизировать правдоподобие, например), и всё равно хорошо;
 - ещё это напоминает обучение с подкреплением: генератор получает награду за действия; но так себе похоже: G знает градиенты D , а сам D сильно меняется со временем.

ПОЧЕМУ И КАК GAN РАБОТАЕТ?

- Практические советы из ранней истории GAN:
 - всегда помогают метки – сэмплы становятся гораздо лучше, если давать метки классов, причём даже если давать их только дискриминатору (Salimans et al., 2016);
 - сглаживание меток (label smoothing): D оценивает отношение распределений, а нейронные сети любят быть слишком уверены; поэтому цель для p_{data} меняем с 1 на 0.9, чтобы D не был слишком уверен; а на p_{model} не надо сглаживать.

ПОЧЕМУ И КАК GAN РАБОТАЕТ?

- Но хороши! И ещё лучше, если выполнять практические советы:
 - виртуальная нормализация по мини-батчам: вариации в мини-батче делают сэмплы скоррелированными; можно виртуально добавить reference batch и учитывать его в статистиках.

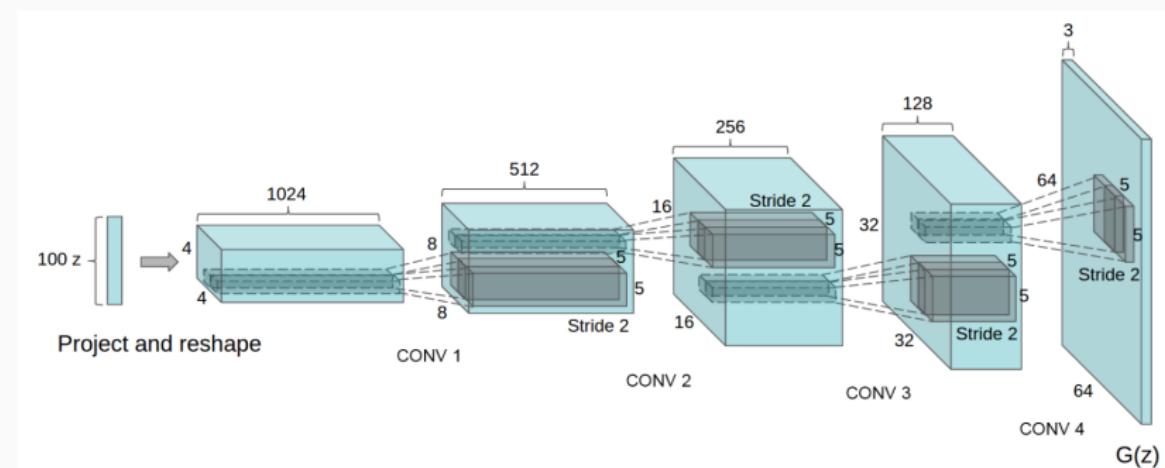


ПОЧЕМУ И КАК GAN РАБОТАЕТ?

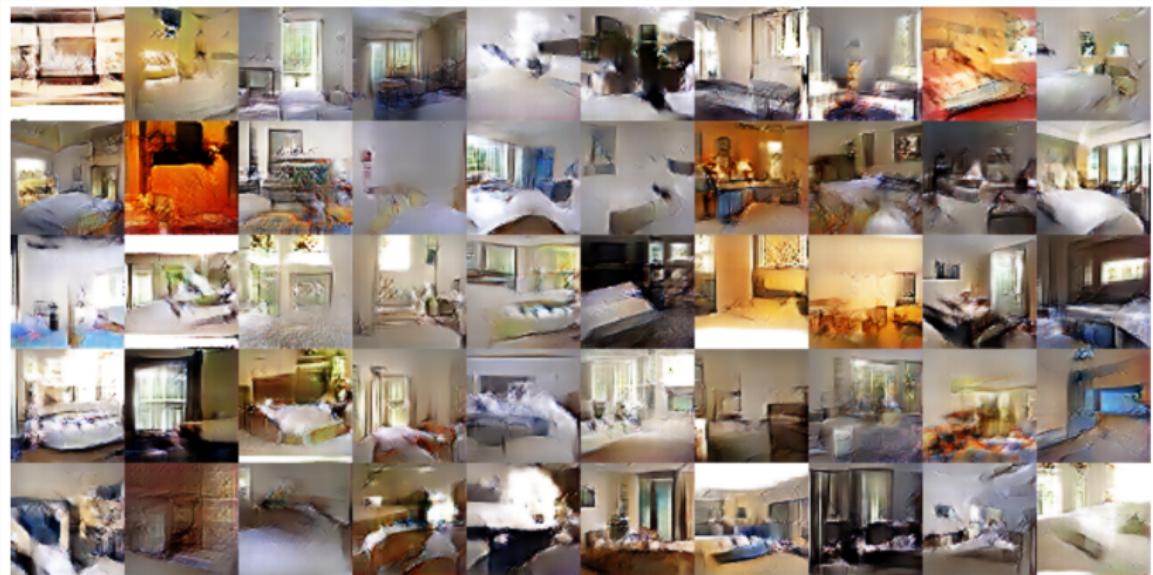
- Но хороши! И ещё лучше, если выполнять практические советы:
 - не надо балансировать G и D : мы обучаем отношение распределений, и оно хорошо оценивается, если D оптимален, т.е. пусть D будет «сильнее»;
 - на практике это значит, что D может быть более сложной сетью; просто дольше обучать D не особенно помогает;
 - пока не очень решённая проблема – diversity; GAN часто производит очень похожие сэмплы, mode collapse.
- Это советы из ранних статей, сейчас они всё ещё важны, но применимы не всегда.

DCGAN

- DCGAN – Deep Convolutional Generative Adversarial Networks (Radford et al., 2016).
- Несколько новых трюков (свёртки с пропусками вместо пулинга, везде batchnorm, убираем полно связные слои, Adam...)



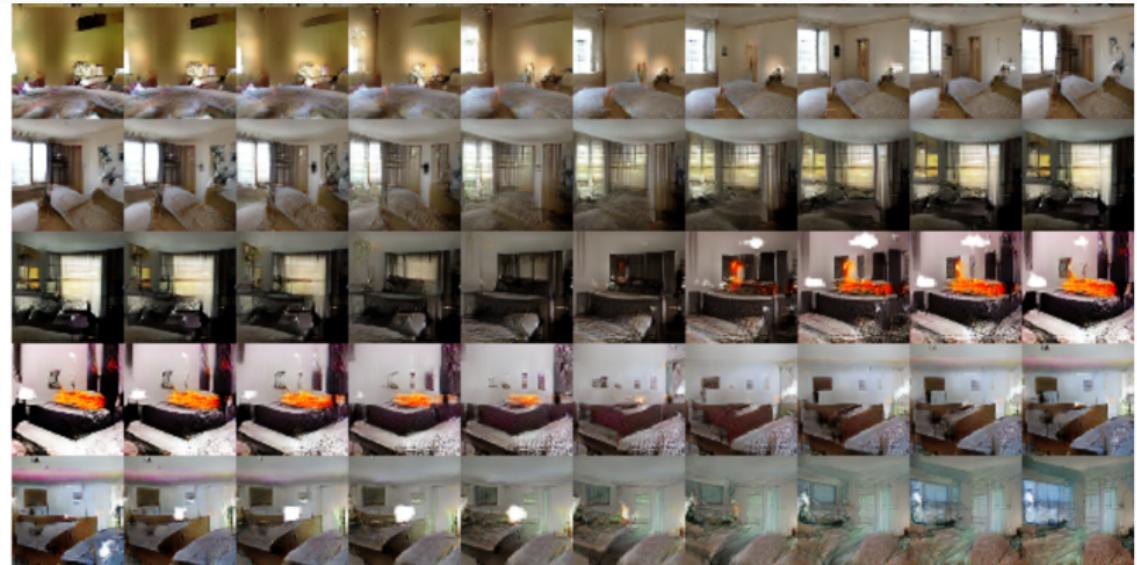
- Спальни (LSUN) после одной эпохи:



- Спальни (LSUN) после пяти эпох:



- Прогулка по пространству признаков:



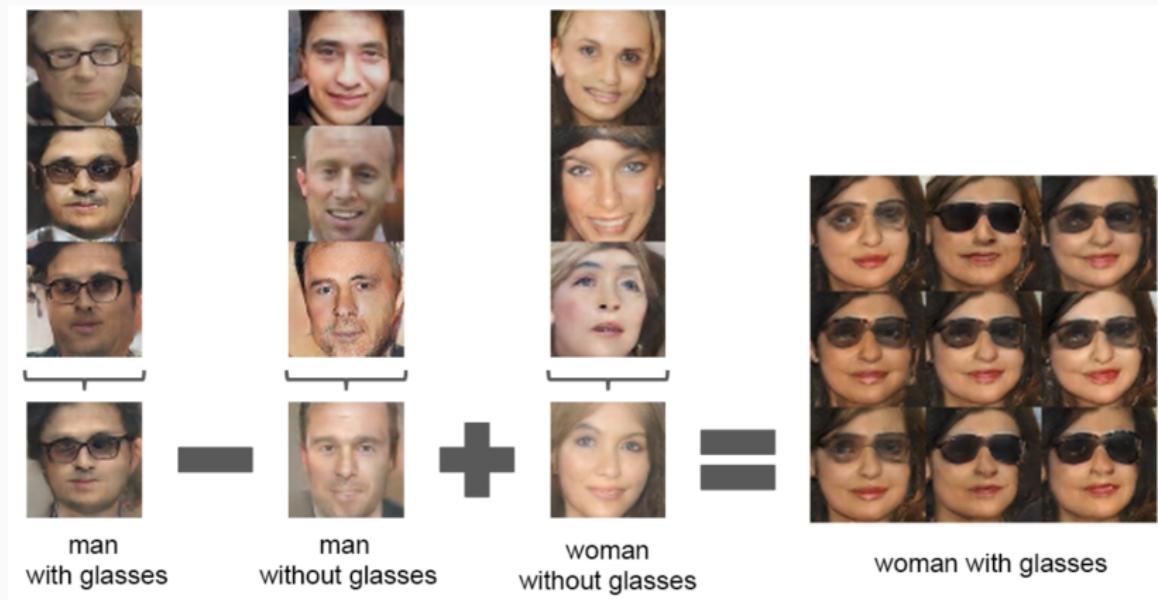
DCGAN

- Убираем фильтры, распознающие окна:



DCGAN

- Векторная арифметика (как в *word2vec*):



DCGAN

- (Geitgey, 2017): pixel art из игр NES

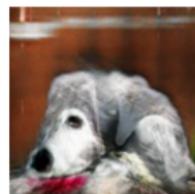
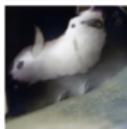


- АНИМАЦИЯ: <https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7>

- Но не всегда, конечно, работает – проблемы с подсчётом:



- Но не всегда, конечно, работает – проблемы с перспективой (right?):



ПРОГРЕСС

- Так работали самые первые GAN'ы.
- Но мы с тех пор прошли долгий путь:

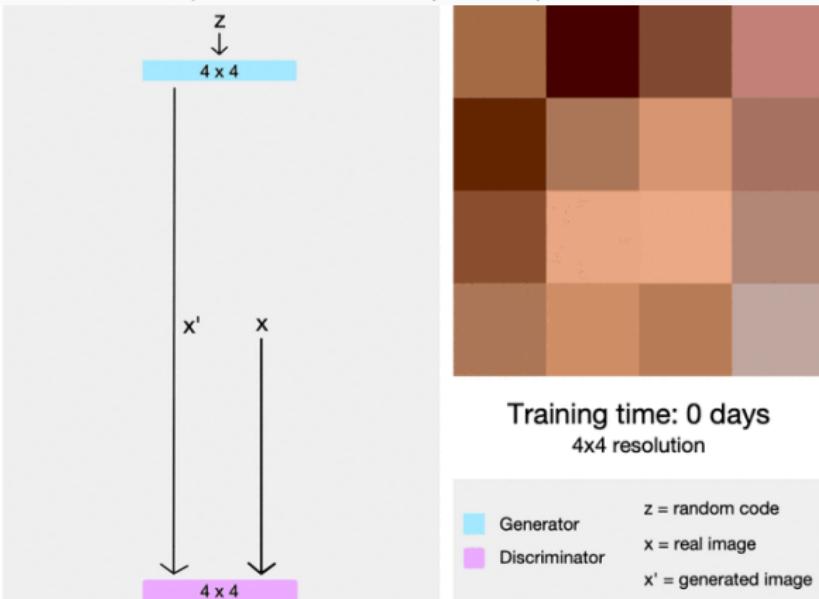


- Как же так получилось?

ENLARGE YOUR GAN

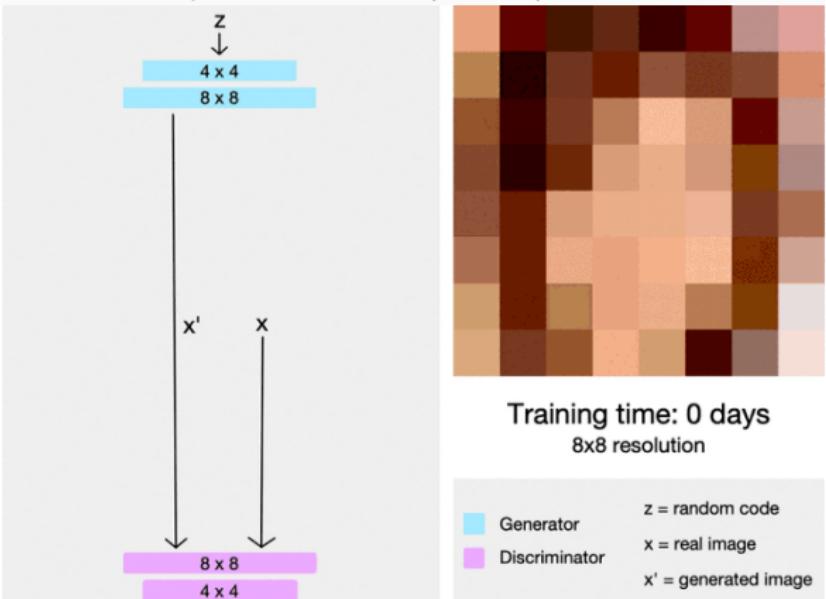
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



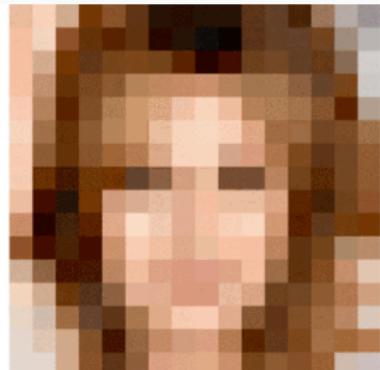
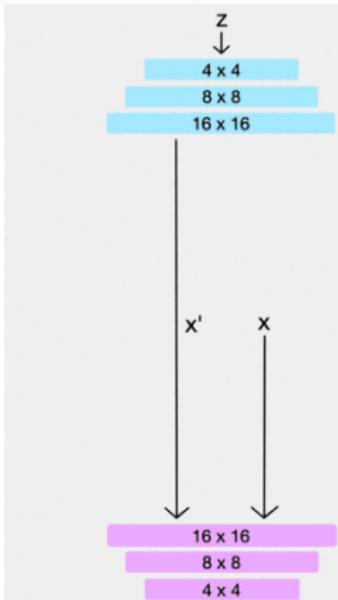
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024

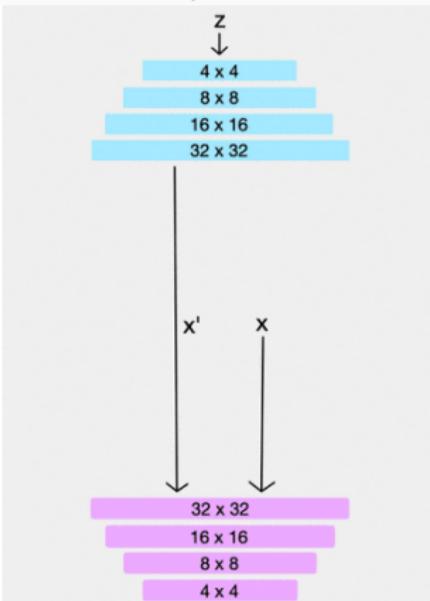


Training time: 0 days
16x16 resolution

Generator z = random code
Discriminator x = real image
 x' = generated image

ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024

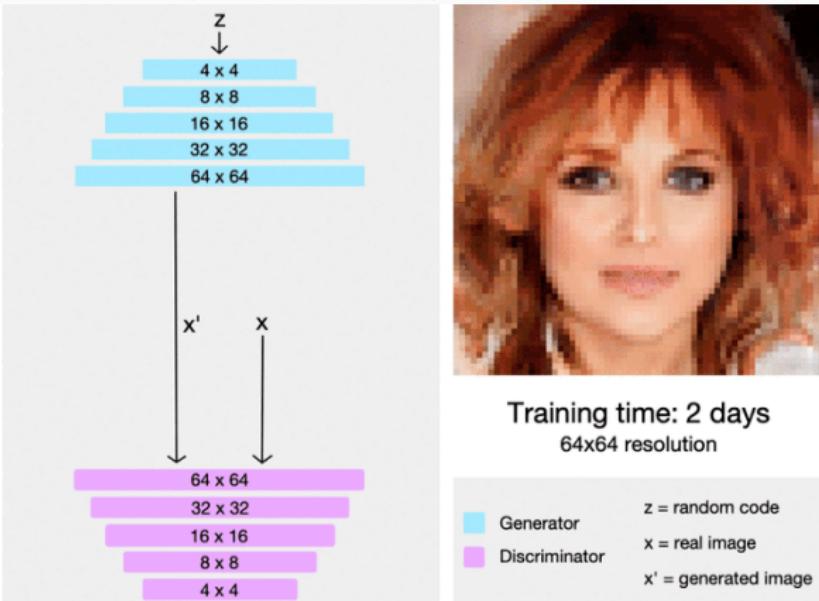


Training time: 1 day
32x32 resolution

Generator	z = random code
Discriminator	x = real image
	x' = generated image

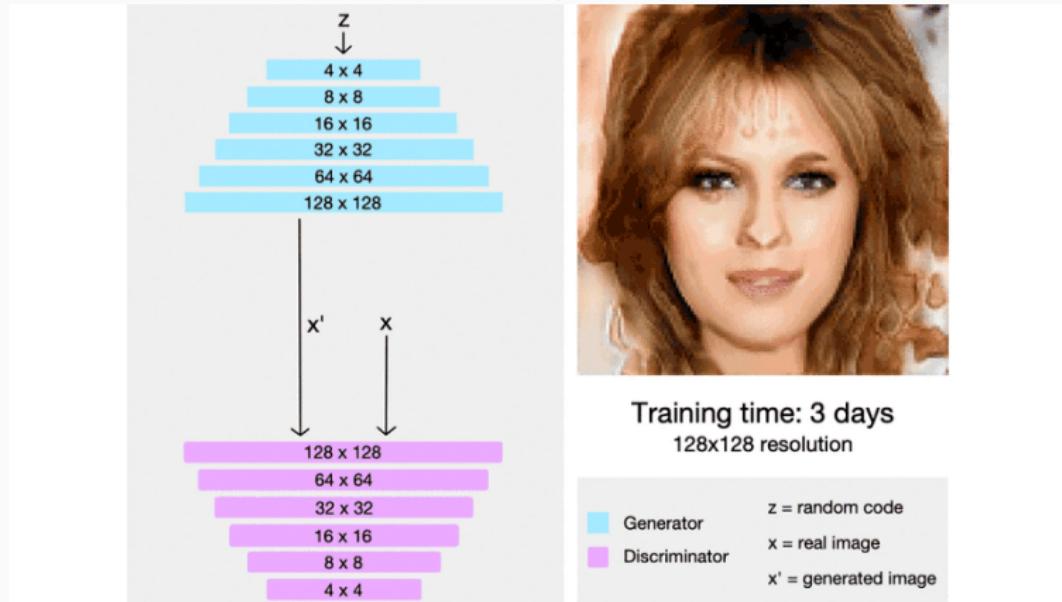
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



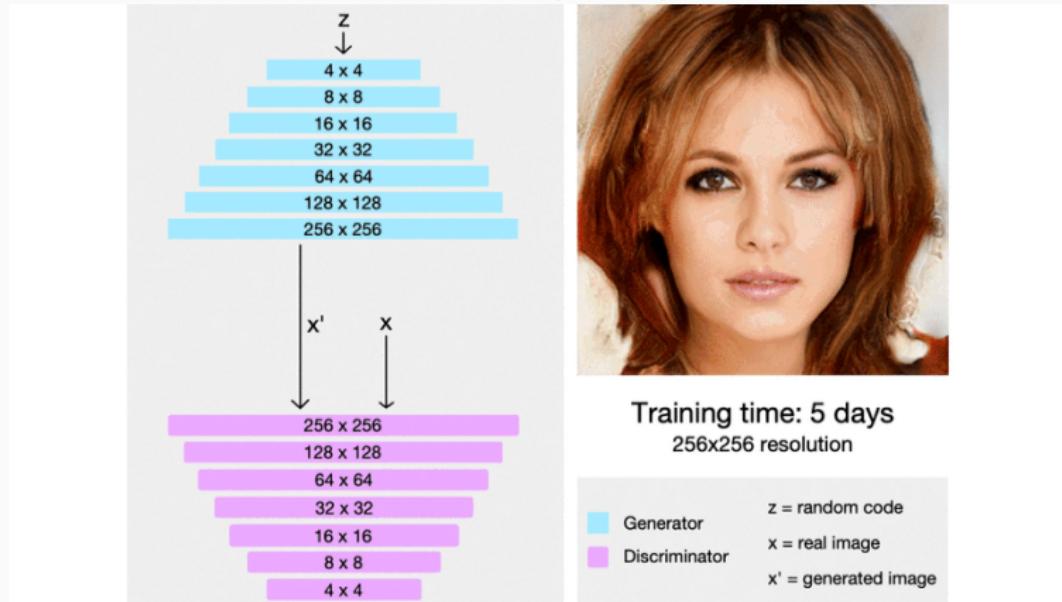
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



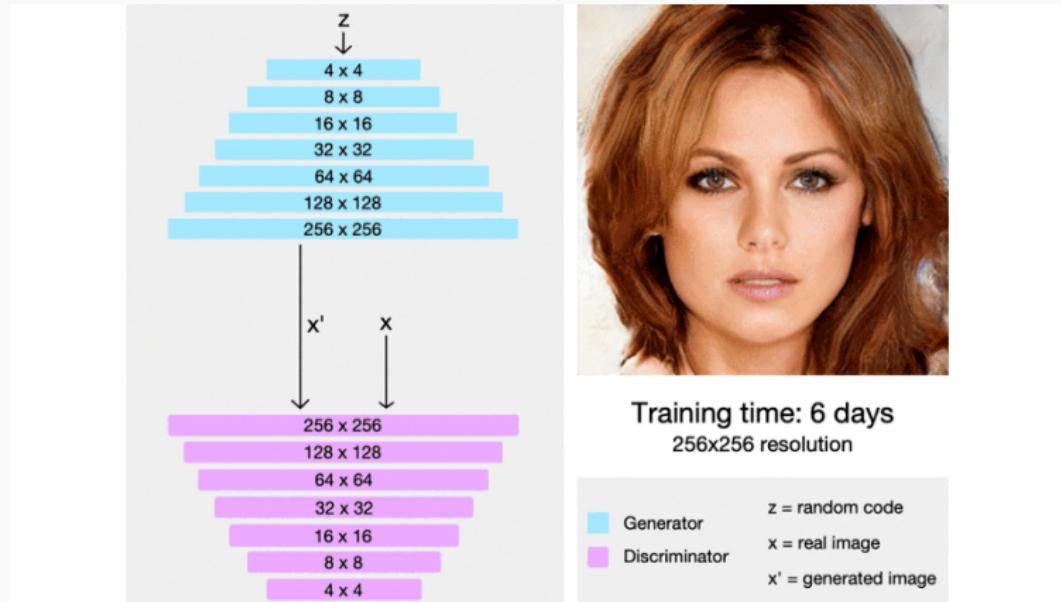
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



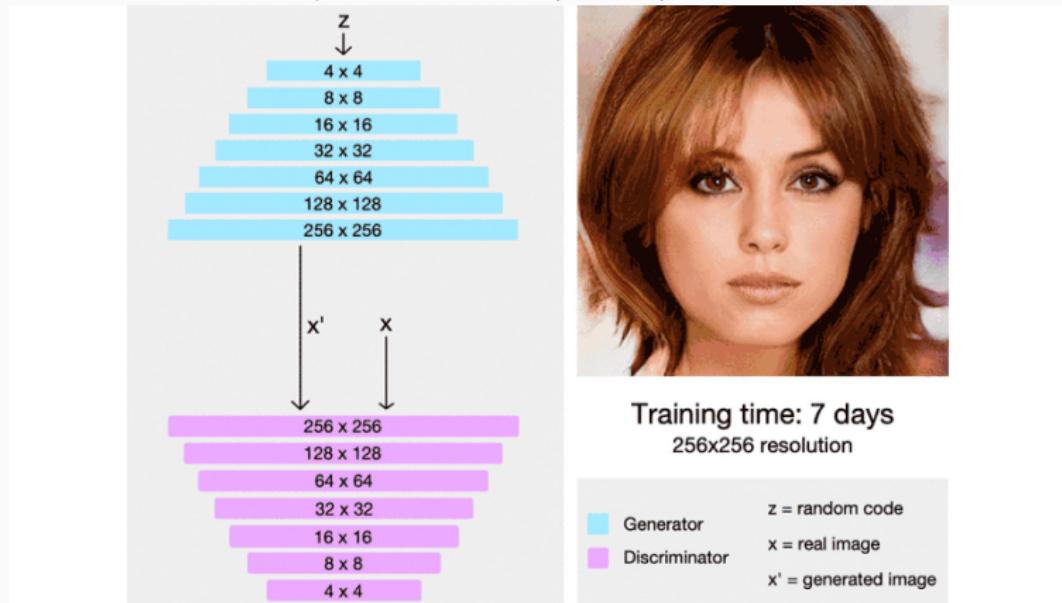
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



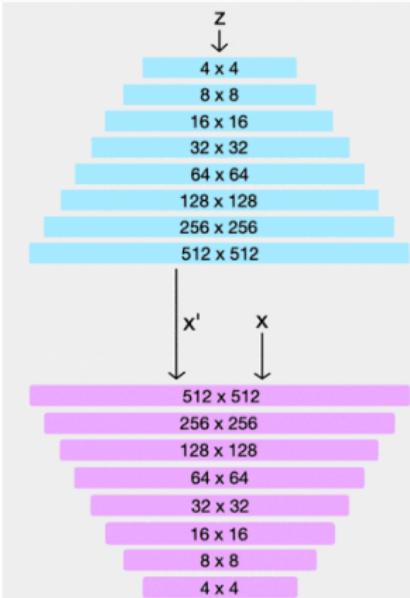
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



Training time: 8 days
512x512 resolution

Generator z = random code
Discriminator x = real image
 x' = generated image

ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



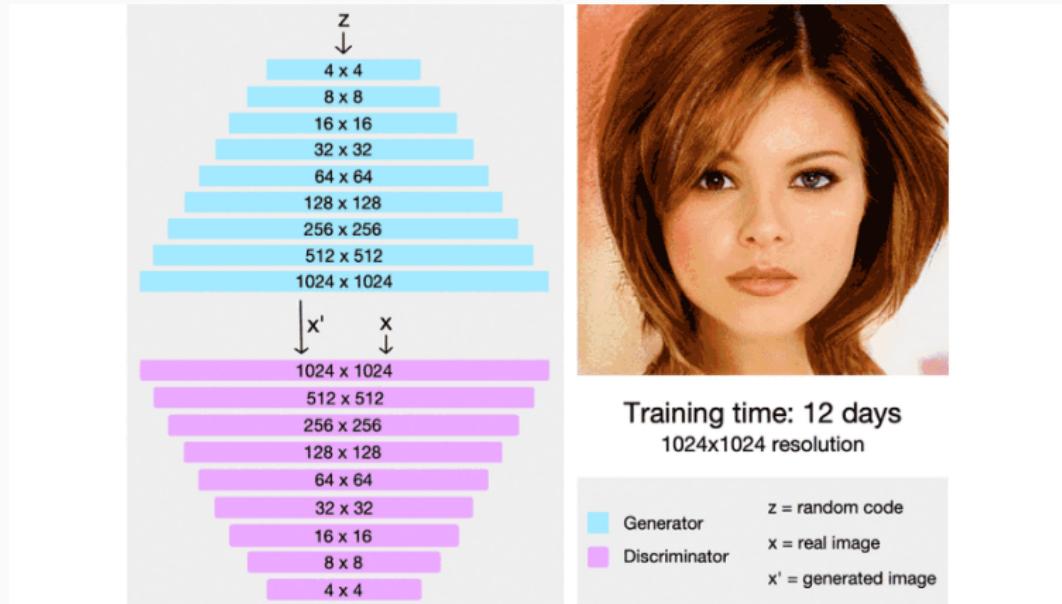
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



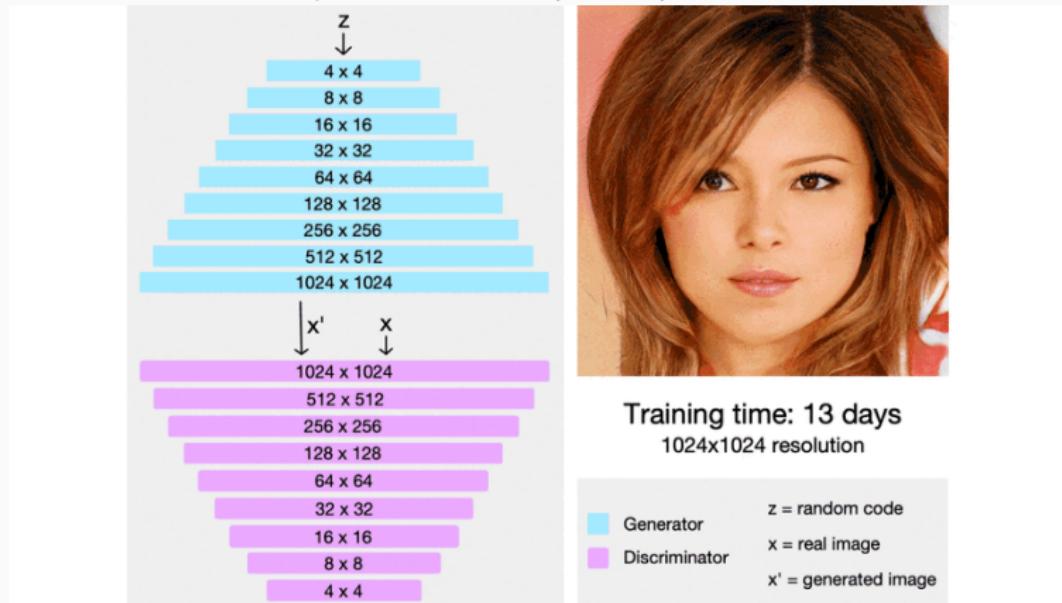
ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



ProGAN

- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024

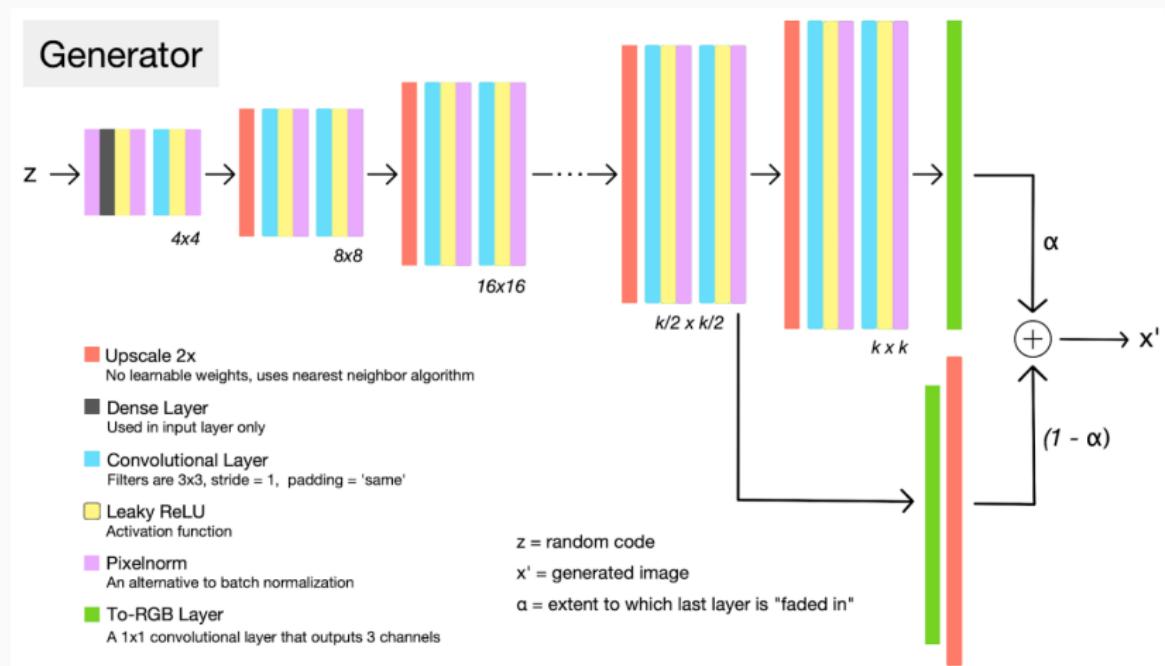


ProGAN

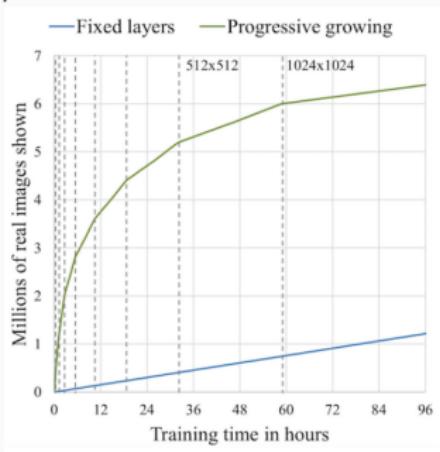
- Важный прорыв пришёл из NVIDIA (Karras et al., 2017)
- Progressive growing (ProGAN): на каждом этапе обучаем GAN создавать картинки вчетверо больше, начиная от 4×4
- И так постепенно увеличивают размерность до 1024×1024



- Именно добавляем новые слои каждый раз; дискриминатор примерно симметричен



- Это помогает экономить время обучения, показывать больше данных за то же время



- Давайте посмотрим:
 - <https://www.youtube.com/watch?v=G06dEcZ-QTg>
 - <https://www.youtube.com/watch?v=36lE9tV9vm0>
- На других категориях до 1024×1024 не доводили, но 256×256 выглядит неплохо

- На других категориях до 1024×1024 не доводили, но 256×256 выглядит неплохо

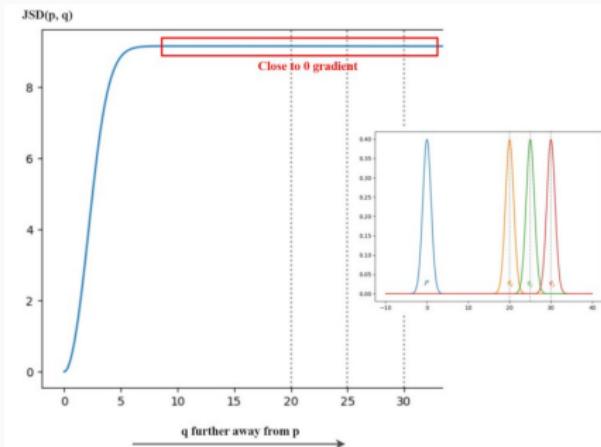


- И всё это обучается довольно стандартным способом: авторы сравнивают функции ошибки от least squares GAN и от Wasserstein GAN...
- ...wait, what??? Давайте разбираться...

ФУНКЦИИ ОШИБКИ В GAN

LSGAN

- (Mao et al., 2016): Least Squares GAN (LSGAN)
- Проблема с обычными GAN'ами: функция ошибки (дивергенция Йенсена-Шеннона) насыщается, когда распределение генератора далеко от правильного, и ничего не обучается.



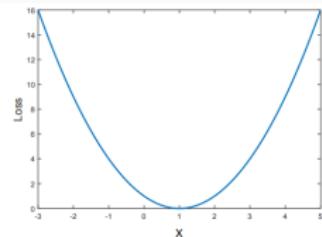
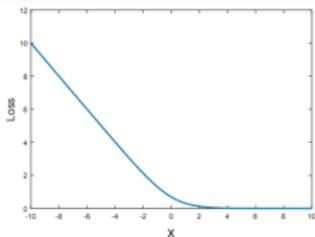
- Это потому, что дискриминатор в обычном GAN работает как классификатор с логистическим сигмоидом.

LSGAN

- Давайте попробуем перейти от сигмоидальной к квадратичной функции ошибки:

$$\min_D V_{\text{LSGAN}}(D) = \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [(D(\mathbf{x}) - b)^2] + \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}} [(D(G(\mathbf{z})) - a)^2],$$
$$\min_G V_{\text{LSGAN}}(G) = \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}} [(D(G(\mathbf{z})) - c)^2],$$

т.е. дискриминатор учится отвечать a для фейков и b для настоящих данных, а генератор хочет убедить его отвечать c на фейках. Обычно берут просто $a = 0$, $b = c = 1$.



LSGAN

- Mao et al. показали, что LSGAN устойчивее к изменениям архитектуры, меньше страдает от mode collapse; в целом, сейчас квадратичная функция ошибки применяется часто.



(a) LSGANs.



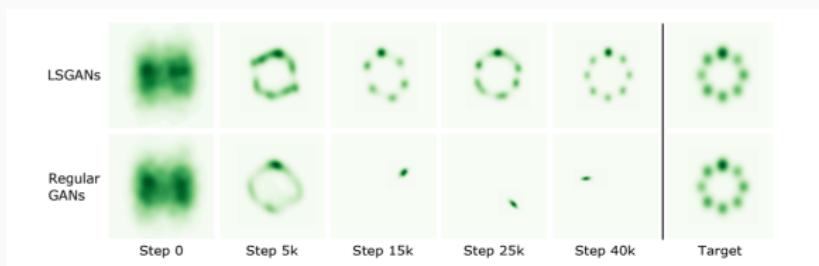
(b) Regular GANs.



(c) LSGANs.



(d) Regular GANs.



- (Chen et al., 2016): InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- Важное дополнение: было бы круто добавить disentanglement, чтобы разные признаки имели смысл
- В обычных GAN'ах используют просто вектор шума \mathbf{z} . В InfoGAN мы его разбиваем на части:
 - \mathbf{z} – это действительно несжимаемый шум, источник энтропии;
 - $\mathbf{c} = c_1 c_2 \dots c_L$ – это латентный код.
- Можно в обычный GAN попробовать это добавить, чтобы генератор стал $G(\mathbf{z}, \mathbf{c})$, но надо что-то добавить, чтобы \mathbf{c} было важно и G не мог бы просто забыть на него.

- InfoGAN: будем требовать, чтобы была большая взаимная информация (mutual information) $I(\mathbf{c}, G(\mathbf{z}, \mathbf{c}))$, т.е. в распределении $G(\mathbf{z}, \mathbf{c})$ должно оставаться много от \mathbf{c} .
- Формально – вариационная оценка:

$$\begin{aligned}
 I(c; G(z, c)) &= H(c) - H(c|G(z, c)) \\
 &= \mathbb{E}_{x \sim G(z, c)} [\mathbb{E}_{c' \sim P(c|x)} [\log P(c'|x)]] + H(c) \\
 &= \mathbb{E}_{x \sim G(z, c)} [\underbrace{D_{\text{KL}}(P(\cdot|x) \parallel Q(\cdot|x))}_{\geq 0} + \mathbb{E}_{c' \sim P(c|x)} [\log Q(c'|x)]] + H(c) \\
 &\geq \mathbb{E}_{x \sim G(z, c)} [\mathbb{E}_{c' \sim P(c|x)} [\log Q(c'|x)]] + H(c)
 \end{aligned}$$

- И эту нижнюю оценку мы через reparametrization trick параметризуем нейросетью Q ; она обычно почти целиком совпадает с D .

- Пример на MNIST с $\mathbf{c} = c_1 c_2 c_3$, где c_1 – дискретная метка с 10 категориями (без supervision! с равномерным априорным распределением), и $c_2, c_3 \sim U[-1, 1]$:

	
---	--

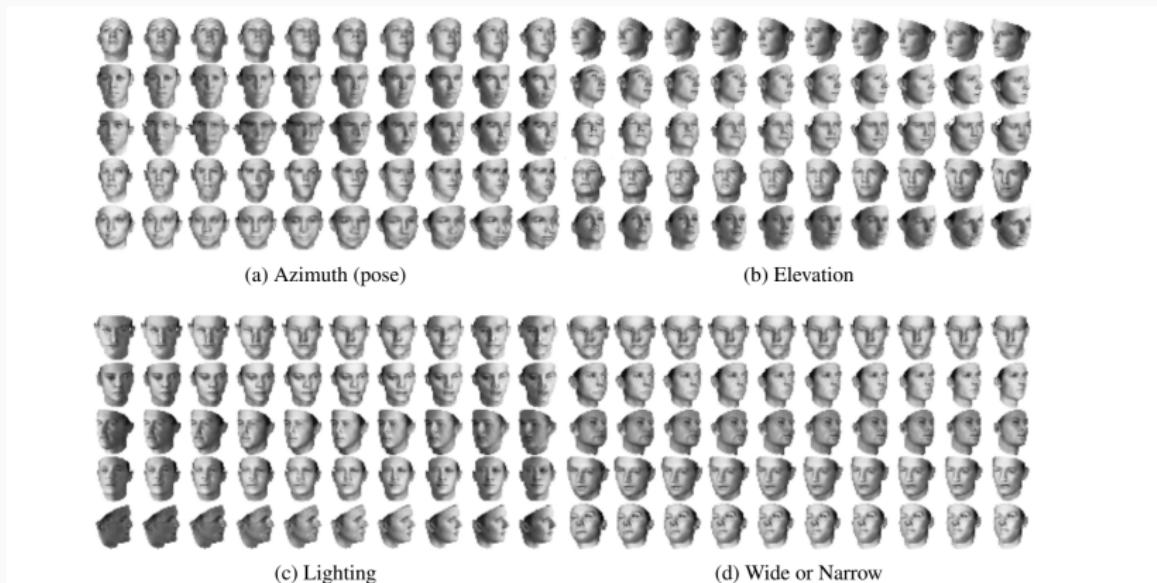
(a) Varying c_1 on InfoGAN (Digit type)(b) Varying c_1 on regular GAN (No clear meaning)

	
---	--

(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

- На самом деле по коду c_1 классификатор даёт ошибку 5% на MNIST без всякого supervision.

- 3D-лица:



- 3D-стулья:



(a) Rotation

(b) Width

- Wassershtein GAN (Arjovsky et al., 2017):
 - что такое обучить распределение вероятностей?
 - это значит минимизировать $KL(p_{\text{data}} \parallel p_{\text{model}})$;
 - но для этого нужно, чтобы p_{model} существовала, а это неверно, если распределение сосредоточено на подмногообразиях малой размерности; вряд ли многообразие p_{model} будет существенно пересекаться с многообразием p_{data} , и KL будет не определено (бесконечно);
 - обычно мы добавляем шум (гауссовский, например), все модели в ML с шумом;
 - но для порождающих моделей шум мешает, делает порождённые примеры размытыми;
 - поэтому обычно шум используют для подсчёта ML, но убирают во время порождения, т.е. шум – это неправильно, но надо для ML.

WASSERSHTEIN GAN

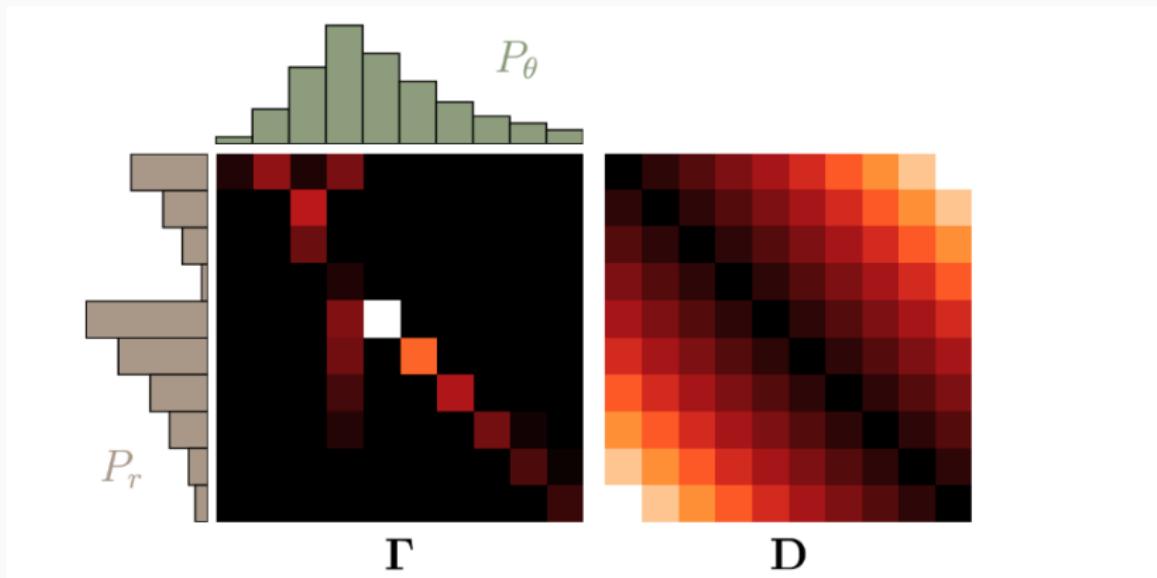
- Wassershtain GAN (Arjovsky et al., 2017): давайте посмотрим на другие метрики близости между p_{data} и p_{model} .
- Earth Mover distance, или Wassershtain-1:

$$W(p_{\text{data}}, p_{\text{model}}) = \inf_{\gamma \in \Pi(p_{\text{data}}, p_{\text{model}})} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \gamma} [\|\mathbf{x} - \mathbf{y}\|],$$

где $\Pi(p_{\text{data}}, p_{\text{model}})$ – множество совместных распределений $\gamma(\mathbf{x}, \mathbf{y})$, маргиналы которых – p_{data} и p_{model} .

WASSERSHTEIN GAN

- Т.е. $\gamma(\mathbf{x}, \mathbf{y})$ показывает, сколько надо «массы» перераспределить от \mathbf{x} к \mathbf{y} , чтобы превратить p_{data} в p_{model} .



- Пример: рассмотрим $Z \sim U[0, 1]$, \mathbb{P}_0 – распределение $(0, Z)$ на \mathbb{R}^2 , и $g_\theta(z) = (\theta, z)$, где θ – параметр. Тогда

- $W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|,$

- $JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$

- $KL(\mathbb{P}_\theta \| \mathbb{P}_0) = KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$

- and $\delta(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0. \end{cases}$

- Т.е. только ЕМ-расстояние действительно куда-то сходится при $\theta \rightarrow 0$, а это ведь очень похоже на обучение распределения, сосредоточенного на подмногообразии.

WASSERSHTEIN GAN

- Можно доказать, что $W(p_{\text{data}}, p_{\text{model}})$ – непрерывная функция от параметров θ распределения p_{model} при достаточно слабых условиях.
- А двойственность Монжа-Канторовича (Kantorovich-Rubinstein duality) говорит, что инфимум

$$W(p_{\text{data}}, p_{\text{model}}) = \inf_{\gamma \in \Pi(p_{\text{data}}, p_{\text{model}})} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \gamma} [\|\mathbf{x} - \mathbf{y}\|]$$

эквивалентен

$$W(p_{\text{data}}, p_{\text{model}}) = \sup_{\|f\|_L \leq 1} \left(\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_{\text{model}}} [f(\mathbf{x})] \right),$$

где супремум берётся по всем функциям с липшицевой константой ≤ 1 .

- А мы хотим обучить порождающую модель $p_{\text{model}} = g_\theta(\mathbf{z})$.
- Давайте всё параметризуем нейросетями.

WASSERSHTEIN GAN

- Сеть $f_{\mathbf{w}}$ для функции f и сеть для g_{θ} . Тогда:
 - для данного g_{θ} обучим веса $f_{\mathbf{w}}$, максимизируя

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_{\text{model}}} [f(\mathbf{x})];$$

- для данного $f_{\mathbf{w}}$ подсчитаем

$$\begin{aligned}\nabla_{\theta} W(p_{\text{data}}, p_{\text{model}}) &= \nabla_{\theta} \left(\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_{\text{model}}} [f(\mathbf{x})] \right) = \\ &= -\mathbb{E}_{\mathbf{z} \sim Z} [\nabla_{\theta} f_{\mathbf{w}}(g_{\theta}(\mathbf{z}))].\end{aligned}$$

- А чтобы $f_{\mathbf{w}}$ оставалось липшицевым, можно просто weight clipping сделать для градиентов.

- Итого алгоритм обучения:

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

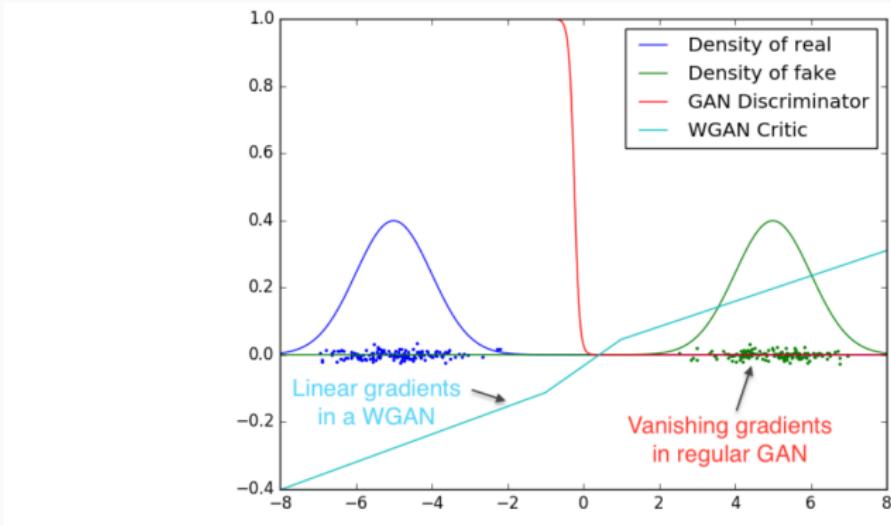
Require: : α , the learning rate. c , the clipping parameter. m , the batch size.
 n_{critic} , the number of iterations of the critic per generator iteration.

Require: : w_0 , initial critic parameters. θ_0 , initial generator's parameters.

```
1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  do
3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$ 
6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$ 
7:      $w \leftarrow \text{clip}(w, -c, c)$ 
8:   end for
9:   Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
10:   $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$ 
11:   $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$ 
12: end while
```

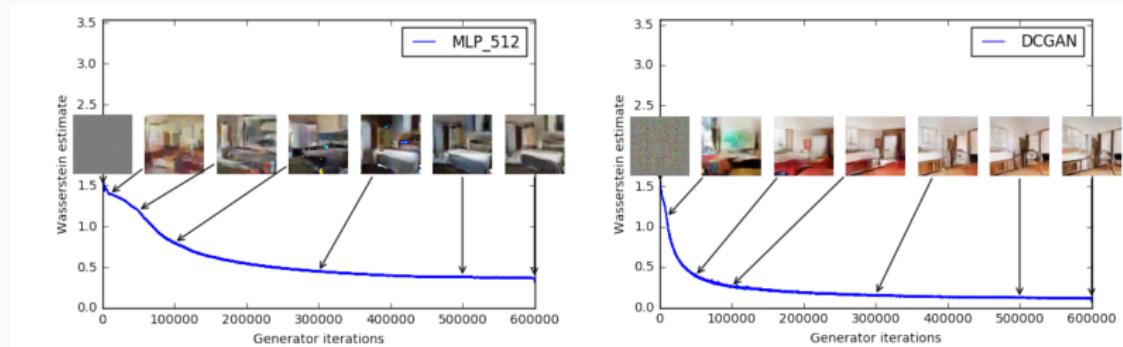
WASSERSHTEIN GAN

- Пример, на котором видно, что в WGAN можно спокойно обучать f_w до сходимости, а в обычном GAN дискриминатор, может, и не стоит так обучать:

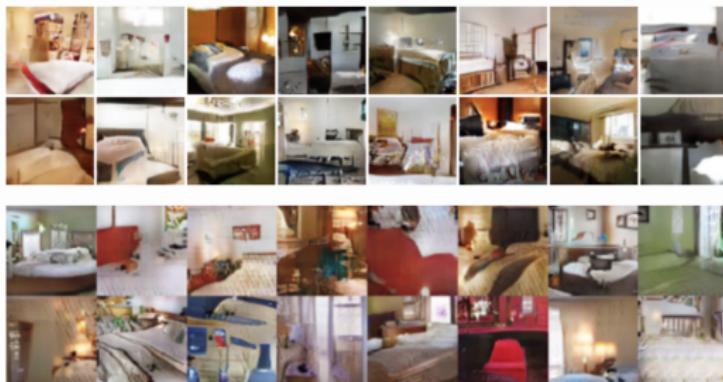


WASSERSHTEIN GAN

- Качество тесно коррелирует с функцией ошибки; слева тут просто полносвязная сеть в качестве генератора, справа – DCGAN:

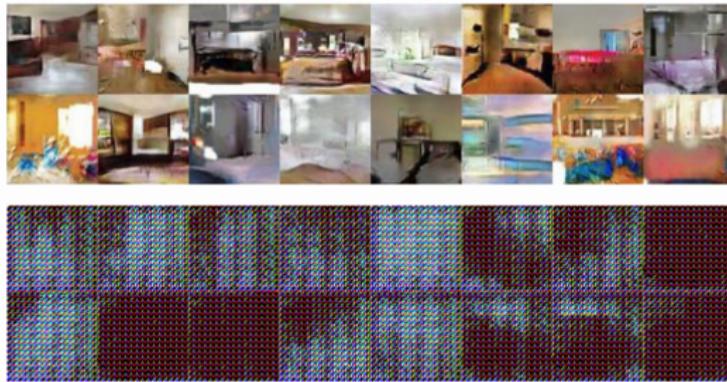


- WGAN устойчивее к изменению архитектуры; вот обычные WGAN и DCGAN (снизу):



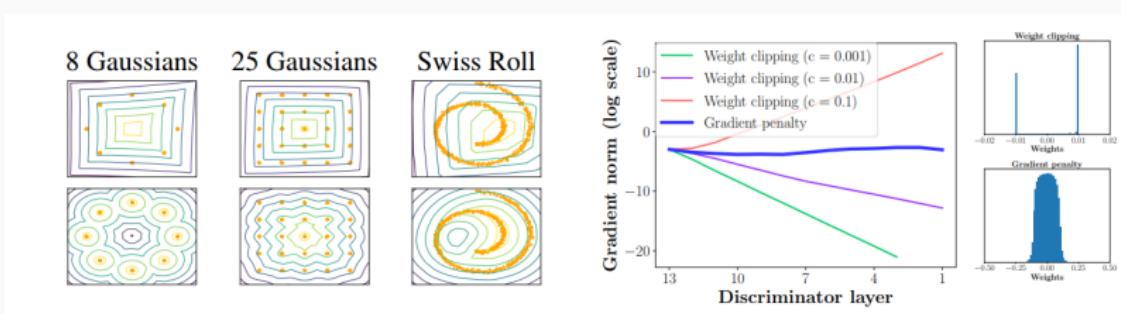
WASSERSHTEIN GAN

- А вот что будет, если убрать из генератора batchnorm и сделать фильтров поменьше (одинаковое число на каждом уровне):



WASSERSHTEIN GAN

- (Gulrajani et al., 2017): Improved Training of Wasserstein GANs
- Weight clipping в критике на самом деле ведёт к проблемам, не обучаются более высокие моменты распределений и градиенты взрываются/затухают
- Лучше делать мягкий регуляризатор на градиент, типа $\lambda \mathbb{E}_{\hat{\mathbf{x}} \sim P_{\hat{\mathbf{x}}}} [(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2]$



Спасибо!

Спасибо за внимание!

