# Анализ неструктурированных данных

Генерация текста

Национальныи Исследовательский Университет
Высшая Школа Экономики

3 декабря 2019 г.

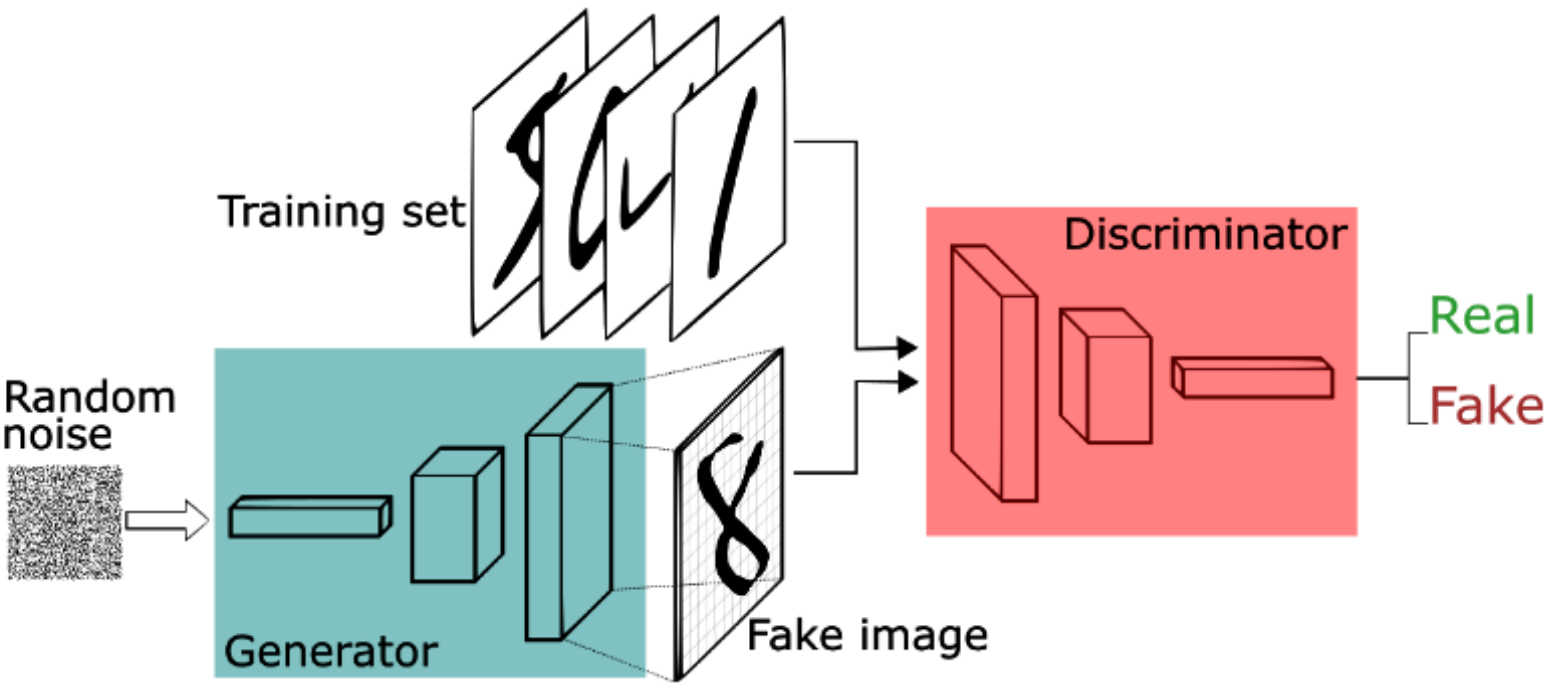
# Зачем порождать текст?

- развлечение
- развлечение
- ещё раз развлечение
- генерация сюжетов в играх (тоже развлечение)
- чат-бот для психологической помощи?

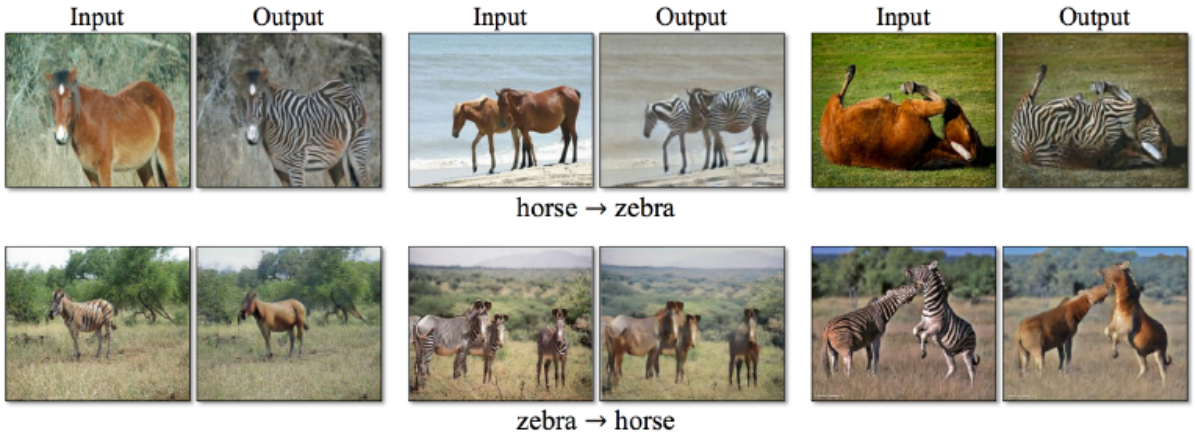
# Generative Adversarial Net

- система, состоящая из generator-a, discriminator-a. Generator порождает примеры, discriminator пытается отличить примеры, порожденные generator-ом от примеров, взятых их обучающей выборки (выдает вероятность того, что данный пример - реальный пример, а не "фальшивка").
- Процесс обучения можно описать как игру с нулевой суммой, в которой функция $v(\theta^g, \theta^d)$ - платеж discriminator-a, а генератор получает $-v(\theta^g, \theta^d)$ в качестве своего платежа.
- В процессе игры каждый пытается максмизировать свой платеж $g^* = argmin_g max_d v(g, d)$ (Идея в том, что не минимизируем лосс(y, p), а решаем другую оптимизационную задачу - максимизации функции успеха)

# Generative Adversarial Net



- Успешно применяются для картинок




- Тут можно поиграться с ганами в браузере
- Получится ли применить для текстов?

# GAN-ы для текстов: проблемы?

- есть ограничения, когда хотим генерировать последовательности дискретных токенов (пространство всех возможных токенов слишком мало; обсуждение с Ian Goodfellow на реддите, на которое ссылается статья)
- дискриминативная модель может оценить, насколько хорошо сгенерировано полное предложение, а по сгенерированной части нельзя понять ни текущий скор, ни будущий (когда полное предложение будет сгенрировано)

# Sequence generation procedure as a Sequential decision Making Process

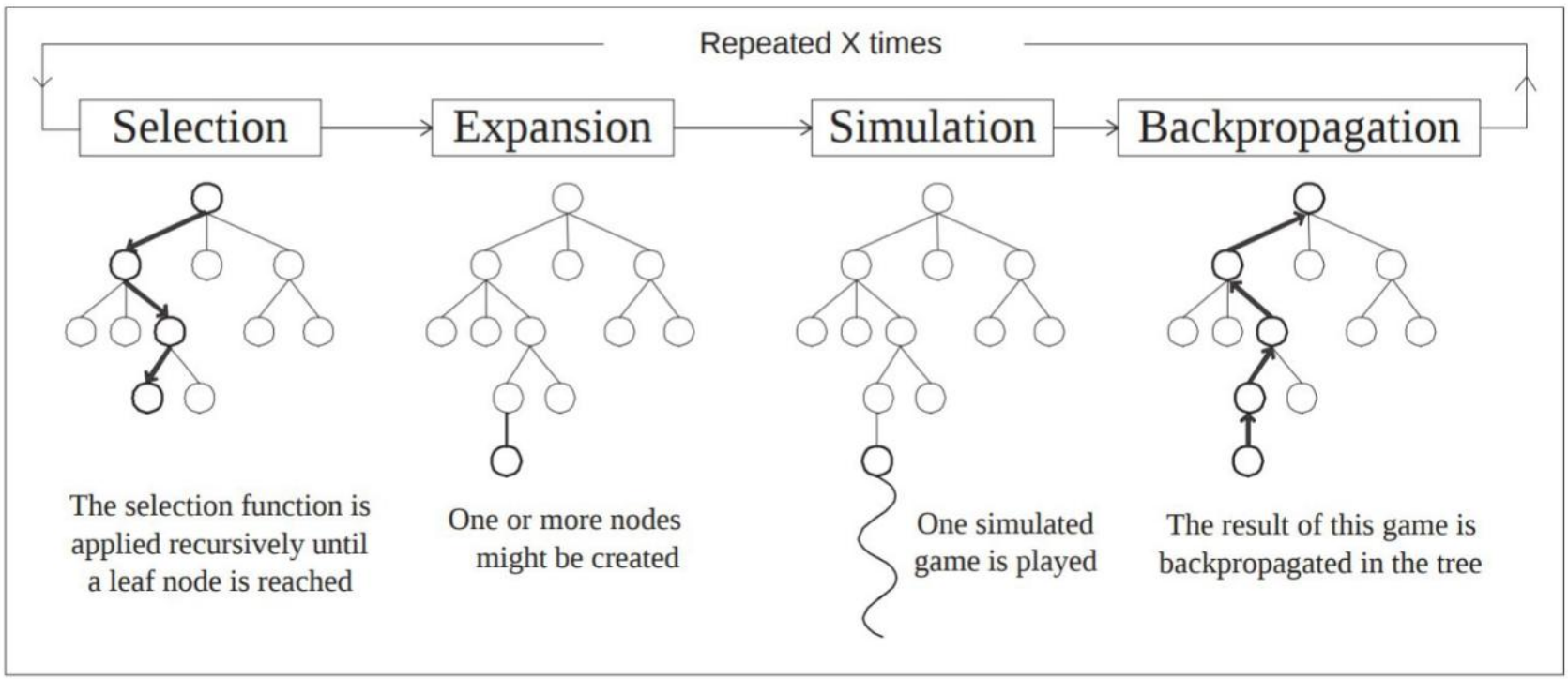
- Generator - агент in RL
- State - уже сгенерированные токены. Action - токен, который мы хотим сгенерировать.
- Reward приходит от discriminator-a (оцененный по полной последовательности)
- Поскольку в силу дискретности аутпут-а мы не можем пробросить градиент к генератору, исопльзуем stochastic parametrized policy. (С Monte-Carlo search для аппроксимации search-value action.)

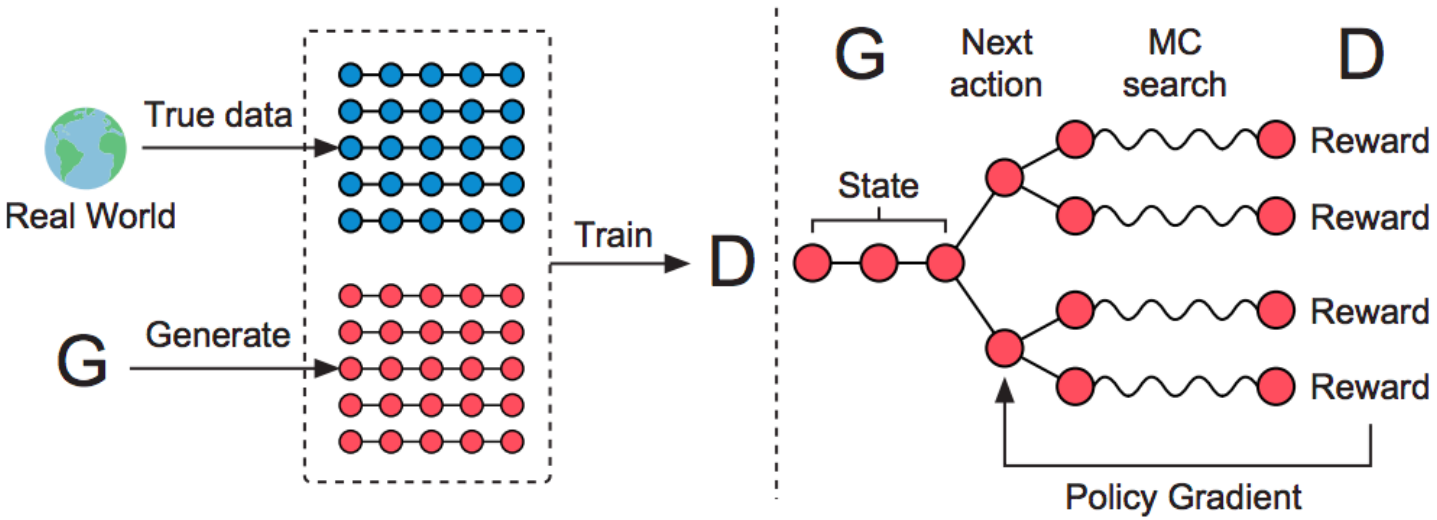
# Sequence GAN

- Обучаем модель $G_\theta$ генерировать последовательность $Y_{1:T} = (y_1, ..y_T), y_t \in Y$ - весь словарь.
- В момент времени $t$ текущее состояние - это $(y_1, ..y_{t-1})$, а действие - следующий токет $y_t$
- $G_\theta(y_t|Y_{1:t-1})$ - стохастическая
- State transition, после того, как действие выбрано, - детерминировано.
- Обучаем модель $D_\phi$
- $D_\phi(Y_{1:T})$ - вероятность того, что последовательность $Y_{1:T}$ - из исходного обучающей выборки (не сгенерированный генератором)

# Monte-Carlo tree search

- нужен, чтобы набрать статистику про самый выгодный путь, не обходя всё дерево
- используется, например, при обучении игре го




True data
Real World
G
Generate
Train
D
G
Next action
MC search
D
State
Reward
Reward
Reward
Reward
Policy Gradient

# SeqGAN via Policy Gradient

- Цель генератора - сгенерировать последовательность от начального состояние $s_0$, максимизировав ожидаемый конечный reward

$$J(\theta) = E[R_T|s_0, \theta] = \sum_{s_1 \in Y} G_\theta(y_1|s_0) Q_{D_\phi^\theta}(s_0, y_1),$$

- $R_T$ - reward за все предложение (от дискриминатора).
- $Q_{D_\phi}^{G_\theta}(s_0, y_1)$ - action-value function последовательности (матожидание реворда, если начать в состоянии s, приняв действие a, следуя стратегии $G_\theta$)
- Как оценить action-value funciton? REINFORCE алгоритм
- $Q_{D_\phi}^{G_\theta}(a = y_T, s = Y_{1:T-1}) = D_\phi(Y_{1:T})$

# SeqGAN via Policy Gradient

- Поскольку нам нужен хороший результат в долгосрочной перспективе (когда сгенерируем все предложение), будем использовать MC search
- Чтобы уменьшить дисперсию и лучше оценить action value, запустимся $N$ раз:

$$Q_{D_\phi}^{G_\theta}(s = Y_{1:t-1}, a = y_t) = \tag{4}$$

$$\begin{cases} \frac{1}{N}\sum_{n=1}^{N} D_\phi(Y_{1:T}^n),\ Y_{1:T}^n \in \mathrm{MC}^{G_\beta}(Y_{1:t}; N) & \text{for} \quad t < T \\ D_\phi(Y_{1:t}) & \text{for} \quad t = T, \end{cases}$$

# SeqGAN via Policy Gradient

- Обучаем дискриминатор:

$$\min_{\phi} -\mathbb{E}_{Y\sim p_{\text{data}}}[\log D_\phi(Y)] - \mathbb{E}_{Y\sim G_\theta}[\log(1 - D_\phi(Y))]$$

- Как только обучили дискриминатор, переходим к генератору:

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{Y_{1:t-1}\sim G_\theta}\Big[\sum_{y_t\in\mathcal{Y}} \nabla_\theta G_\theta(y_t|Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t)\Big]$$

# SeqGAN via Policy Gradient - Algorithm

**Require:** generator policy $G_\theta$; roll-out policy $G_\beta$; discriminator $D_\phi$; a sequence dataset $\mathcal{S} = \{X_{1:T}\}$

Initialize $G_\theta$, $D_\phi$ with random weights $\theta, \phi$.
Pre-train $G_\theta$ using MLE on $\mathcal{S}$
$\beta \leftarrow \theta$
Generate negative samples using $G_\theta$ for training $D_\phi$
Pre-train $D_\phi$ via minimizing the cross entropy
**repeat**
    **for** g-steps **do**
        Generate a sequence $Y_{1:T} = (y_1, \ldots, y_T) \sim G_\theta$
        **for** $t$ in $1 : T$ **do**
            Compute $Q(a = y_t; s = Y_{1:t-1})$ by Eq. (4)
        **end for**
        Update generator parameters via policy gradient Eq. (8)
    **end for**
    **for** d-steps **do**
        Use current $G_\theta$ to generate negative examples and combine with given positive examples $\mathcal{S}$
        Train discriminator $D_\phi$ for $k$ epochs by Eq. (5)
    **end for**
    $\beta \leftarrow \theta$
**until** SeqGAN converges

- Генератор — RNN: маппит инпут эмбеддингов токенов последовательности в последовательность скрытых состояний. Затем softmax маппит скрытые состояния в распределение над выходными токенами.
- Дискриминатор — CNN + max pooling, FC-layer, Softmax.

# Эксперименты

- На синтетических данных: используем случайную LSTM как универсальный источник знаний о мире; учим модели подгоняться под неё — и смотрим, как у них получается.
- На реальных данных: генерация китайских стихов, речей Обамы и музыки.

# Эксперимент на синтетических данных

LSTM — $G_{oracle}$:

- порождает данные для обучения генератора ($G_\theta$)
- оценивает, что порождает генератор, по формуле:

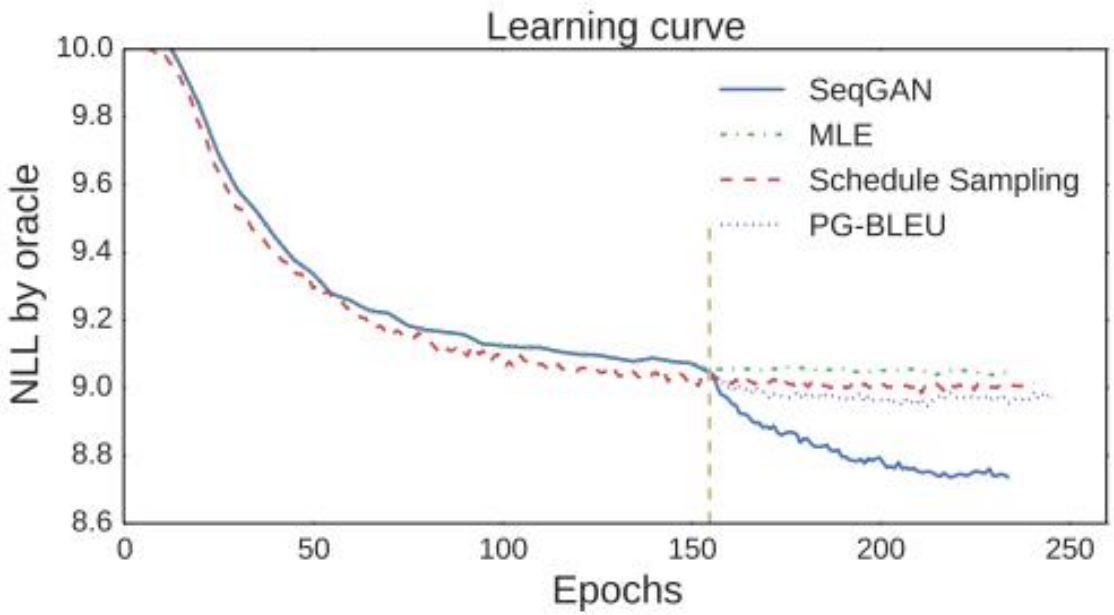
$$\mathrm{NLL}_{\mathrm{oracle}} = -\mathbb{E}_{Y_{1:T} \sim G_\theta} \left[ \sum_{t=1}^{T} \log G_{\mathrm{oracle}}(y_t | Y_{1:t-1}) \right],$$

В эксперименте сравниваются модели:

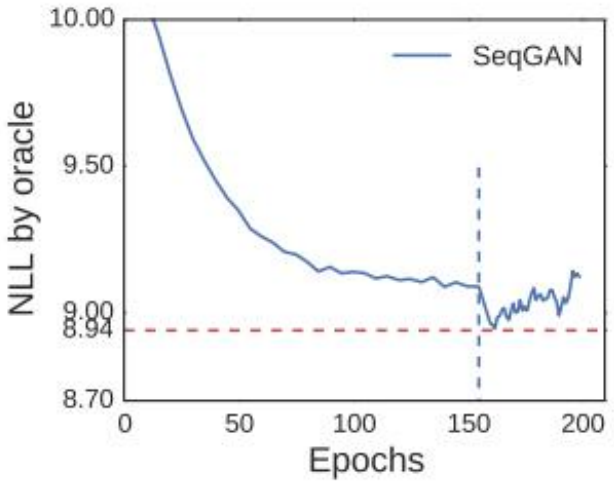
- случайная
- MLE (lstm + teacher forcing)
- scheduled sampling (lstm + не всегда teacher forcing)
- policy gradient + BLUE
- seqGAN

# Эксперимент на синтетических данных

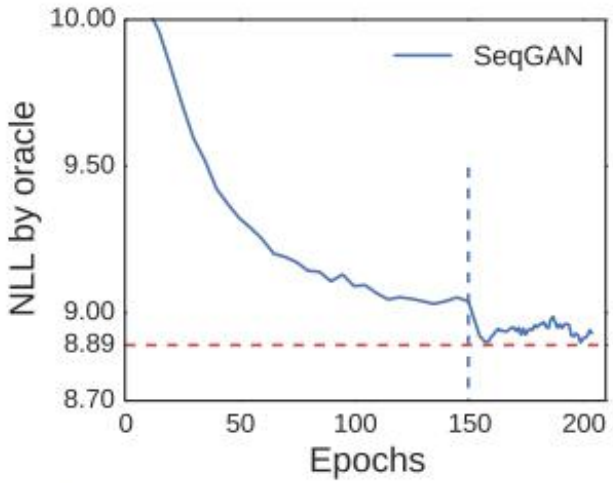
| Algorithm | Random | MLE | SS | PG-BLEU | SeqGAN |
|---|---|---|---|---|---|
| NLL | 10.310 | 9.038 | 8.985 | 8.946 | **8.736** |
| $p$-value | $< 10^{-6}$ | $< 10^{-6}$ | $< 10^{-6}$ | $< 10^{-6}$ | |



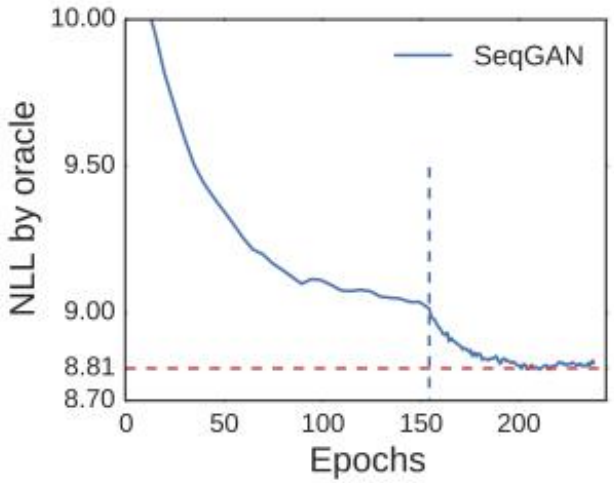
# Зависимость сходимости от гиперпараметров



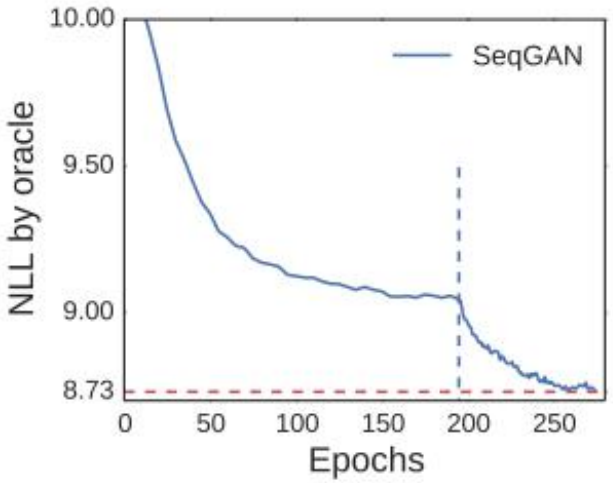

(a) g-steps=100, d-steps=1, $k$=10




(b) g-steps=30, d-steps=1, $k$=30




(c) g-steps=1, d-steps=1, $k$=10




(d) g-steps=1, d-steps=5, $k$=3

# Эксперимент на реальных данных

Table 2: Chinese poem generation performance comparison.

| Algorithm | Human score | $p$-value | BLEU-2 | $p$-value |
|---|---|---|---|---|
| MLE | 0.4165 | 0.0034 | 0.6670 | $< 10^{-6}$ |
| SeqGAN | **0.5356** | | **0.7389** | |
| Real data | 0.6011 | | 0.746 | |

Table 3: Obama political speech generation performance.

| Algorithm | BLEU-3 | $p$-value | BLEU-4 | $p$-value |
|---|---|---|---|---|
| MLE | 0.519 | $< 10^{-6}$ | 0.416 | 0.00014 |
| SeqGAN | **0.556** | | **0.427** | |

Table 4: Music generation performance comparison.

| Algorithm | BLEU-4 | $p$-value | MSE | $p$-value |
|---|---|---|---|---|
| MLE | 0.9210 | $< 10^{-6}$ | 22.38 | 0.00034 |
| SeqGAN | **0.9406** | | **20.62** | |

# В чем проблемы SeqGAN?

- Одного числа от дискриминатора - мало
- Хотим получать от дискриминатора какой-то набор фичей

# Материалы про seqGAN

- Пост на хабре
- SeqGAN (имплементация на PyTorch)
- Статья про SeqGAN (оригинальная версия)
- Статья про LeakGAN — тоже идея GANa для генерации текста

# Порождение с трансформером

GPT-2 — трансформер для порождения текста (используется только декодер с masked self-attention).

Несколько ссылок:

- Понятное объяснение в блоге Jay Alamar
- Код на гитхабе
- Talk to Transformer