Automatic Generation of Context-specific Fake Reviews

Zhanghao Chen, Quanyan Zhu

Introduction

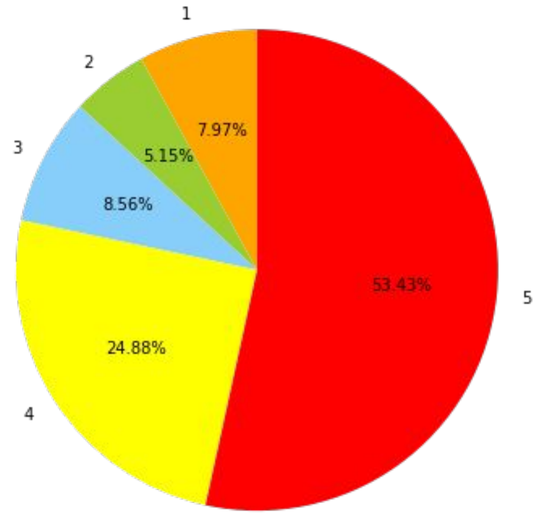
- Fake reviews are a major threat to online review systems by spreading misinformation
- Context-specific reviews are more convincing, thus more threatening
- Malicious crowdsourcing is limited by the cost of human labors
- This project:
  - Explores the possibility of using natural language generation (NLG) techniques for automatic generation of context-specific reviews
  - Reveals the potential threat to online review systems in the future

Prior work

- Yuanshun Yao, et al. (2017) uses a two-stage approach for generating context-specific fake reviews
- Two stages:
    - Initial Review Generation
    - Review Customization with domain-specific keywords replacement
- This project: end-to-end automatic generation

Data

- Use Yelp Open Dataset
- 5,996,996 reviews, 188,593 businesses, 280,992 pictures
- Most of the reviews are 4/5-star



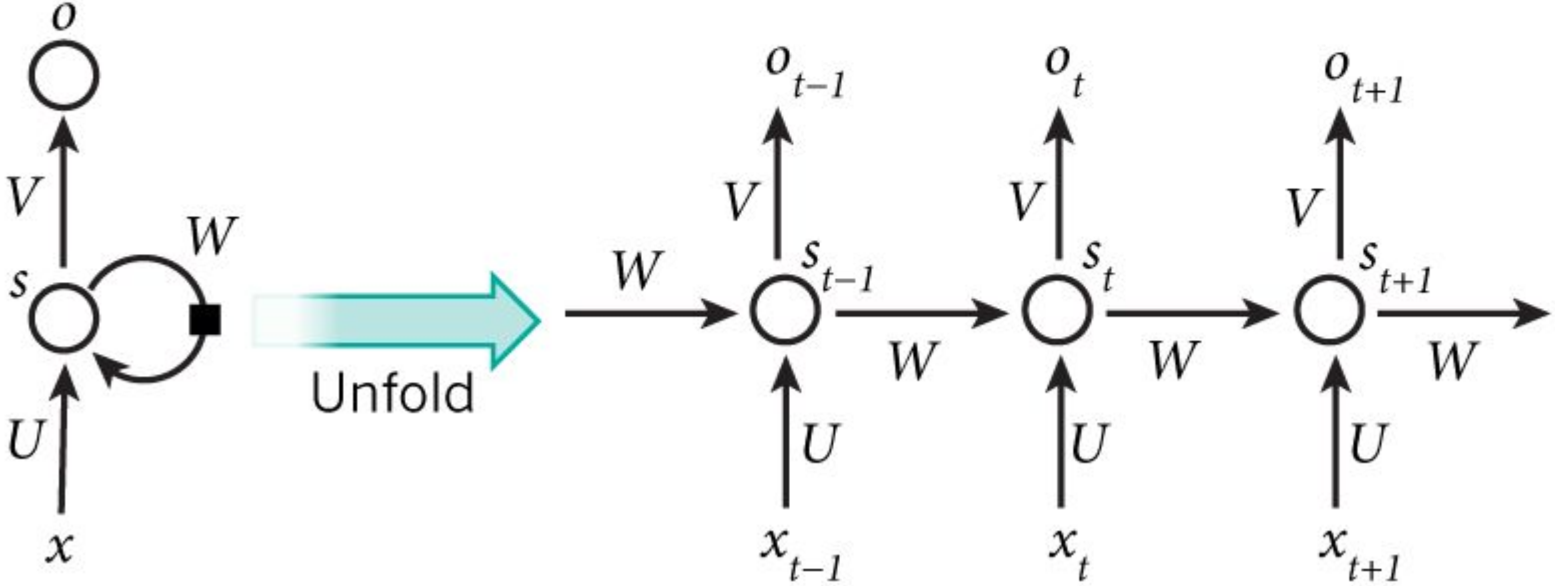
Methodology

- Adopt the seq2seq architecture for fake review generation, which is originally developed for machine translation by Cho et al.
- The encoder captures the context information of reviews (rating star, categories of reviewed restaurant) and the decoder decodes the context information to generate fake reviews.

Recurrent Neural Networks (RNNs)

- Traditional neural network (like a multi-layer perceptron network): all inputs are assumed to be independent from each other, same for outputs
- Many data are in nature sequential
- How to make use of sequential data?
- Idea: unfold neural network along time, each output is conditioned on previous computations

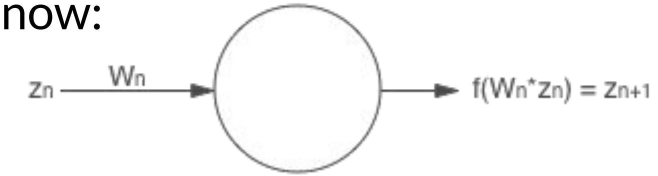
Recurrent Neural Networks (RNNs)

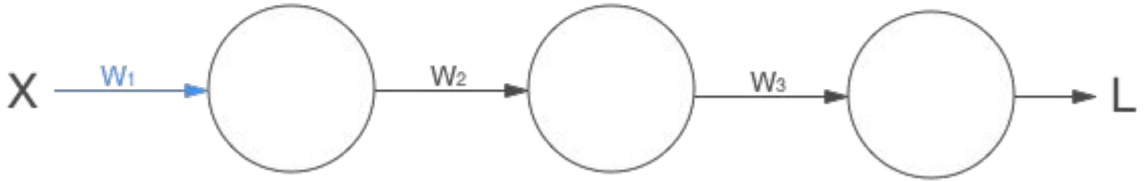


$$s_t = f(Ux_t + Vs_{t-1})$$
$$o_t = \text{softmax}(Ws_t)$$

Problem with Vanilla RNN

- Forget the input for now:





$$\frac{\partial Loss}{\partial W_1} = \frac{\partial Loss}{\partial f(z_3)} \cdot \frac{\partial f(z_3)}{\partial f(z_2)} \cdot \frac{\partial f(z_2)}{\partial f(z1)} \cdot \frac{\partial f(z_1)}{\partial W_1}$$

$$= \frac{\partial Loss}{\partial f(z_3)} \cdot f'(z_3) \cdot W_3 \cdot f'(z_2) \cdot W_2 \cdot f'(z_1) \cdot W_1$$

Problem with Vanilla RNN

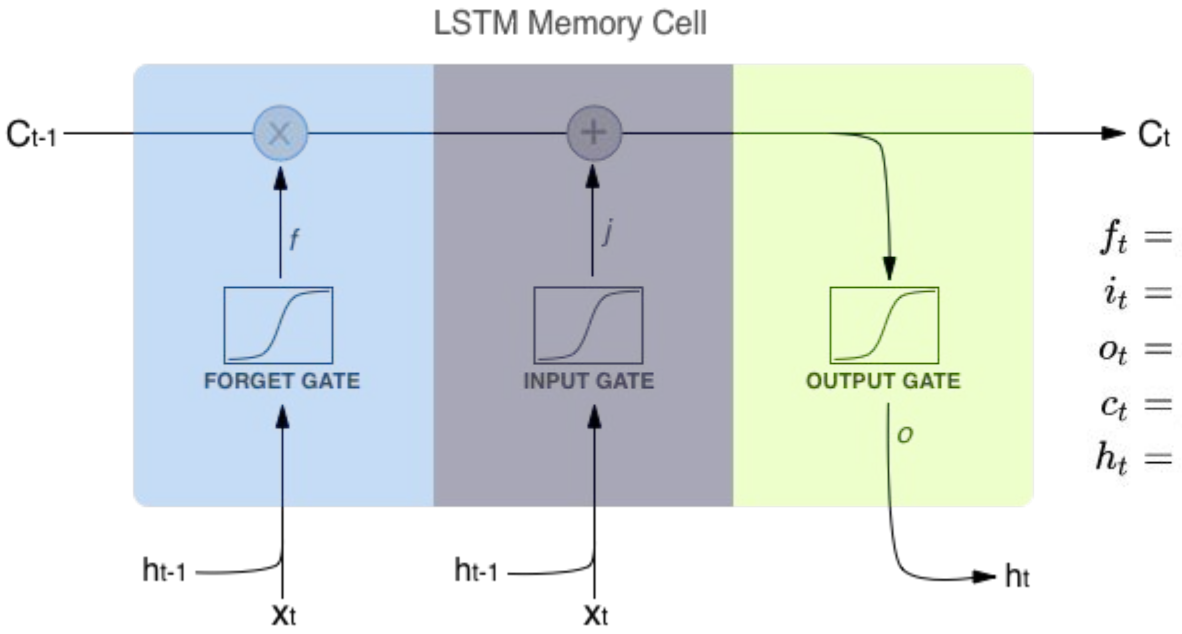
- Matrix W is kept constant along time
- Gradient Explosion/Vanishment can occur!

$$\frac{\partial Loss}{\partial W_1} = \frac{\partial Loss}{\partial f(z_3)} \cdot \frac{\partial f(z_3)}{\partial f(z_2)} \cdot \frac{\partial f(z_2)}{\partial f(z1)} \cdot \frac{\partial f(z_1)}{\partial W_1}$$

$$= \frac{\partial Loss}{\partial f(z_3)} \cdot f'(z_3) \cdot W_3 \cdot f'(z_2) \cdot W_2 \cdot f'(z_1) \cdot W_1$$

LSTM (Long Short-Term Memory, 1997)

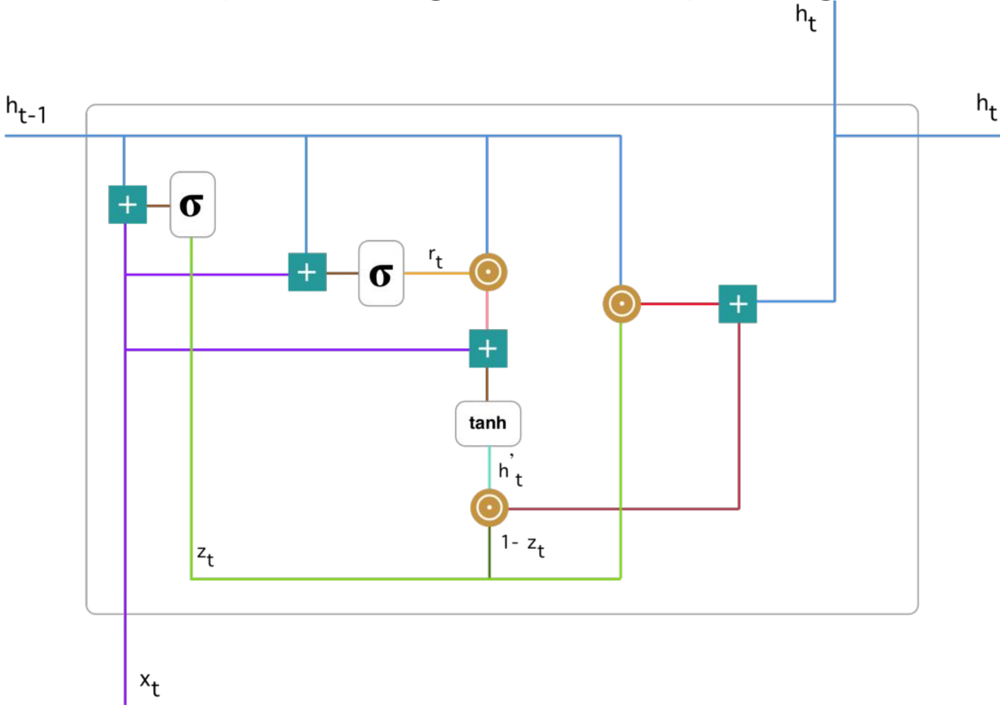
LSTM: add a cell state with three gates to selectively choose information which is remembered and passed to the next time step



$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

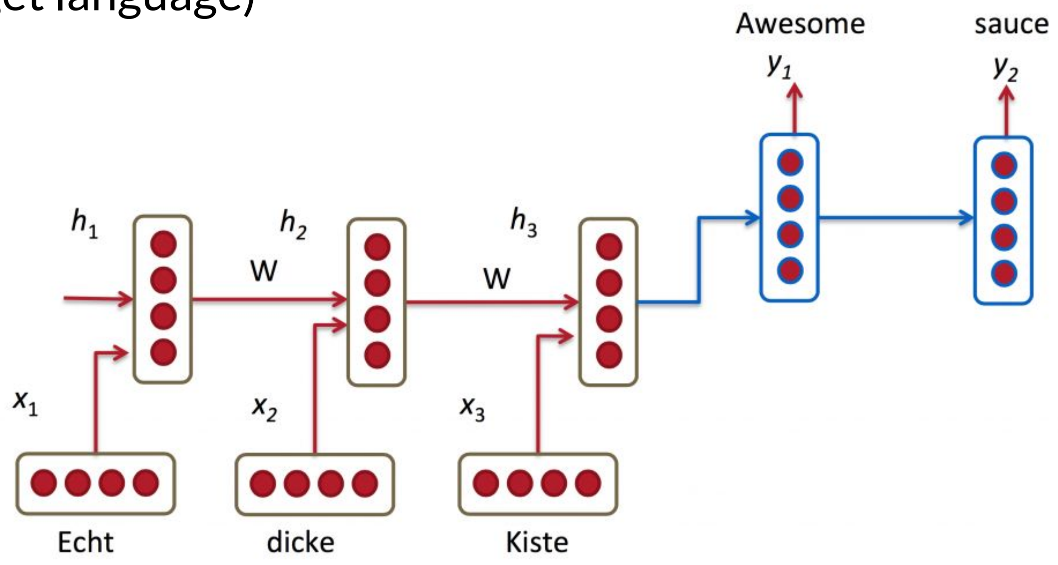
GRU (Gated Recurrent Unit, 2014)

GRU: redundant to keep both the cell state and hidden state in LSTM, why not merge them together? Only has two gates now: update gate and reset gate.



Seq2Seq Model

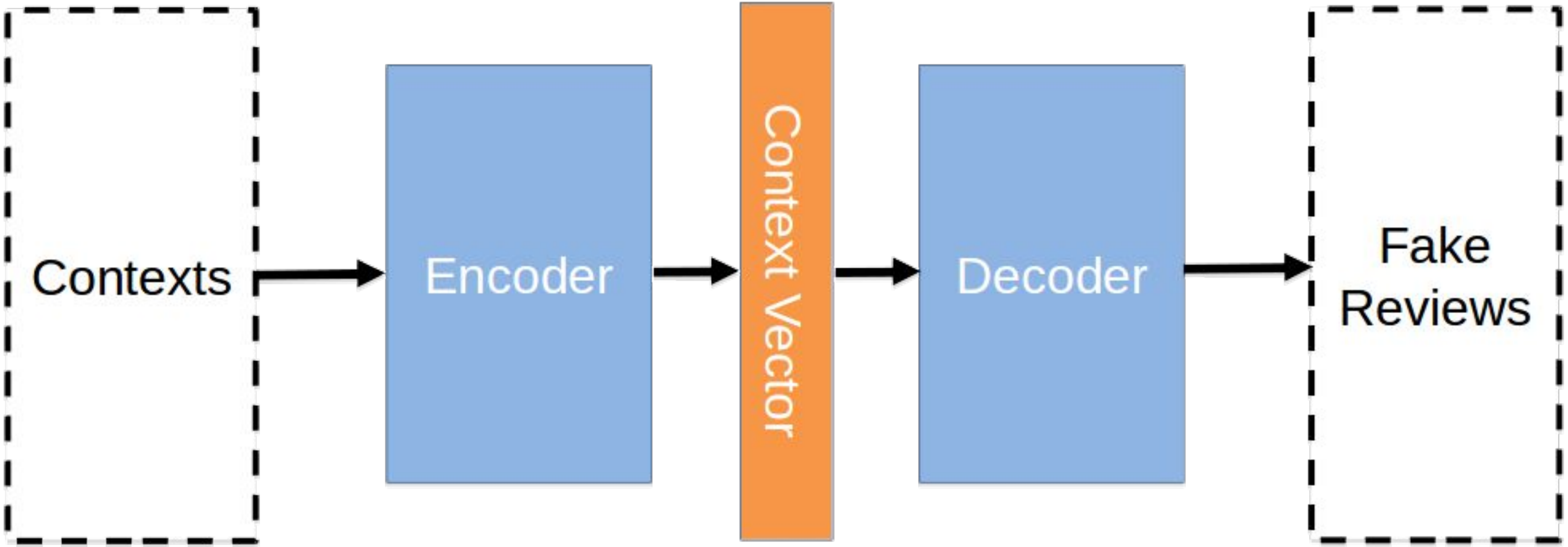
- Originally developed for machine translation in 2014
- Creates a mapping from the context space (source language) to the output space (target language)



Seq2Seq for Context-specific Review Generation

- Seq2Seq can be used to generate sentences that are conditioned on the given context information
- Here, the contexts are designed to be
  - Desired rating star
  - Categories of the restaurant (bar? Chinese style? Seafood? e.t.c.)
- Goal: generate reviews that are relevant to the given context

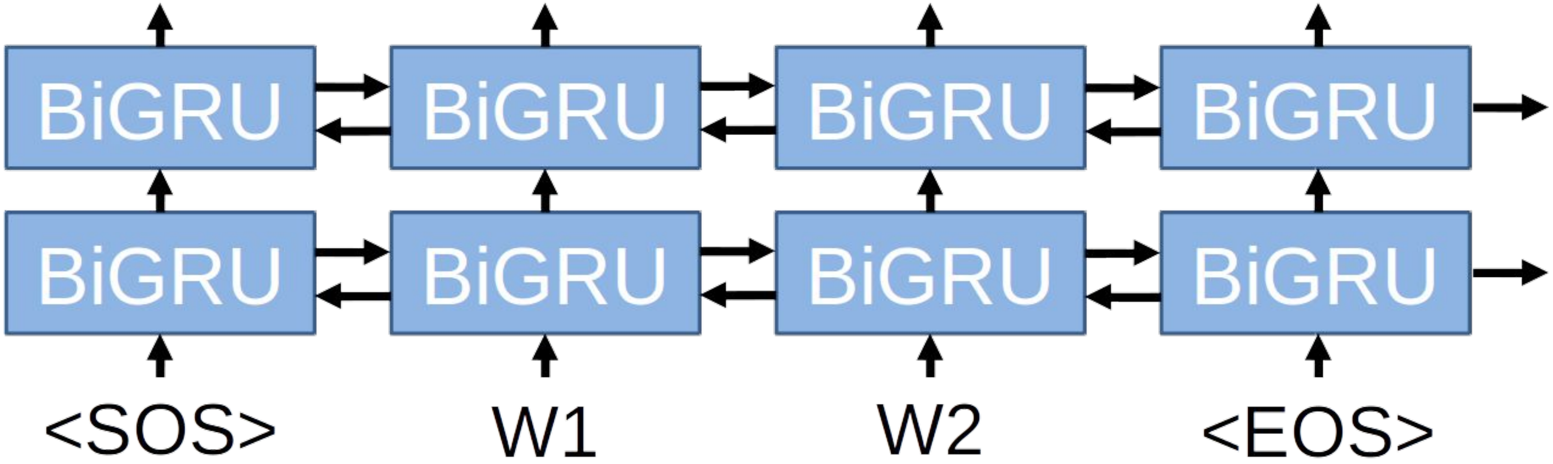
Model



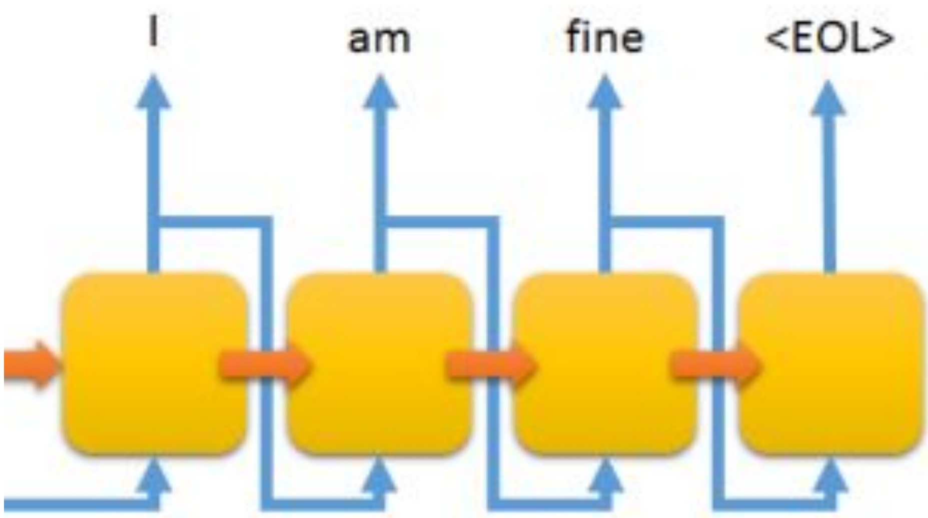
> 5.0 restaurants vietnamese

> its pretty good pho in a nice quiet location , charming .

Encoder — Stacked Bidirectional GRU



Decoder — Stacked Unidirectional GRU



Data Preprocessing

- Remove all redundant reviews
- Remove all non-ASCII characters
- Lowercase all letters
- Filter businesses that are non-restaurant or have more than 20 words of categories information
- Filter reviews with more than 30 words
- Filter reviews with characters @#%*[]+=<>~`^$&/
- Select 5000 reviews of each rating star, 25000 in all, as training data

Training Details

- Teacher forcing with a probability of 0.5
- Vocabulary size: 15000
- Hidden size: 500
- Dropout probability 0.1 between stacked GRU layers
- Batch size: 64
- Adam optimizer with lr 0.0001 for both encoder and decoder
- Criterion: NLL loss function
- Trained for 170 epochs, approx 6 hrs on one single Nvidia P40 GPU with PyTorch

Temperature Control

- Controls the “novelty” of generated text
- Temperatures lower than 1 amplifies the difference in the sampling probability for each word
- T = 0.5 is used by default

$$\sigma(z)_j = \frac{\exp(z_j/T)}{\sum_{k=1}^{K} \exp(z_k/T)}, j = 1 \ldots K$$

Results

> 5.0 mexican tacos tapas/small plates restaurants

=love everything about urban taco ! margaritas are well above average with lots of flavors . large wine list . food was unbelievably good ! very impressed overall !

< everything was delicious , happy hour menu rocked ! i recommend the short rib taco and cream corn with grilled onions and peppers <EOS>

< so so so delicious ! the will and def be the . and and the the and . the <EOS>

< so so so good ! i will be back . . flight the the . the ! and ! corn the my to very <EOS>

< so so so good ! i will rocked be back . the well , happy the happy hour prices are . the and and i . again . <EOS>

Results

> 3.0 korean japanese restaurants

= everything here is pretty inconsistent . service is terrible when it's crowded . the food is pretty good though , refills on sides cost extra .

< decent food . portions large small portions . price not good . menu is small bit for . not much <EOS>

< the food restaurant , the . the not . is . <EOS>

< ordered pork variety . quality got here in a row like the place and its decent if there are too many other casual dining restaurants . <EOS>

< decent food , but nothing special the in a bit restaurant . service is presentation , crowded and cute little slow . service was a bit slow . <EOS>

Results

Tested on 2500 hold-out test pairs:

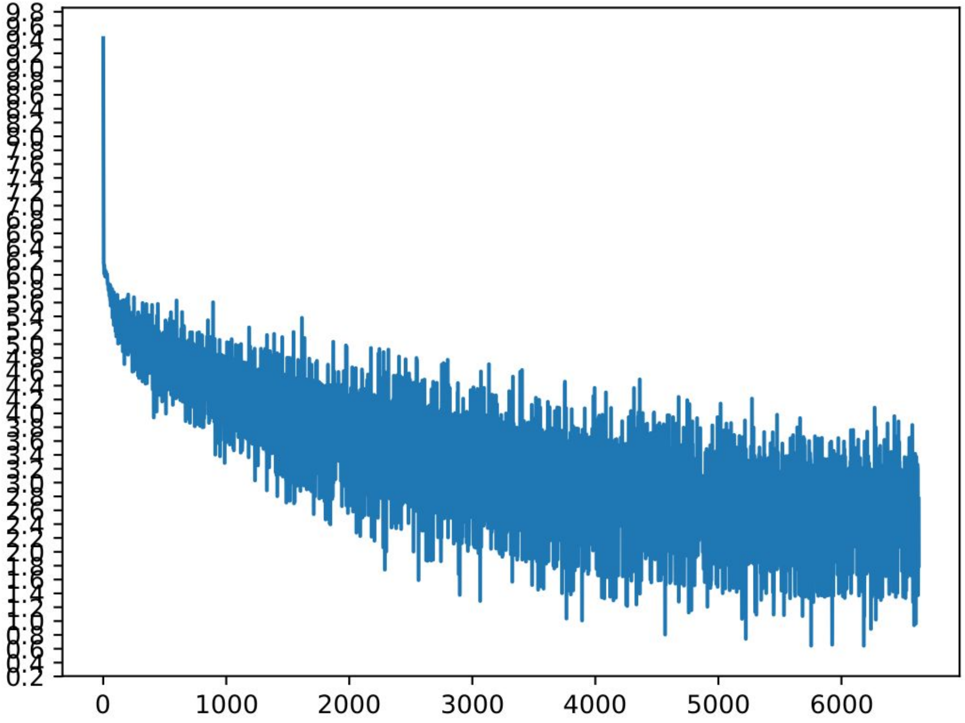
ROUGE1: 0.2396080807469998

ROUGE2: 0.1338730975358988

BLEU: 0.0910617092616161 9

BLEU_CLIP: 0.05604014057908484

Results



Demo

Code is public on:
https://github.com/X-czh/fake-review-generation

Observations

- Data preprocessing is more important than the model itself
- The context information here is “pseudo” sequential, but the model is for “real” sequential data
- Evaluation metric is a big issue
- Discriminating models are much stronger than generating models in this field
- Essential to save every experiment log properly

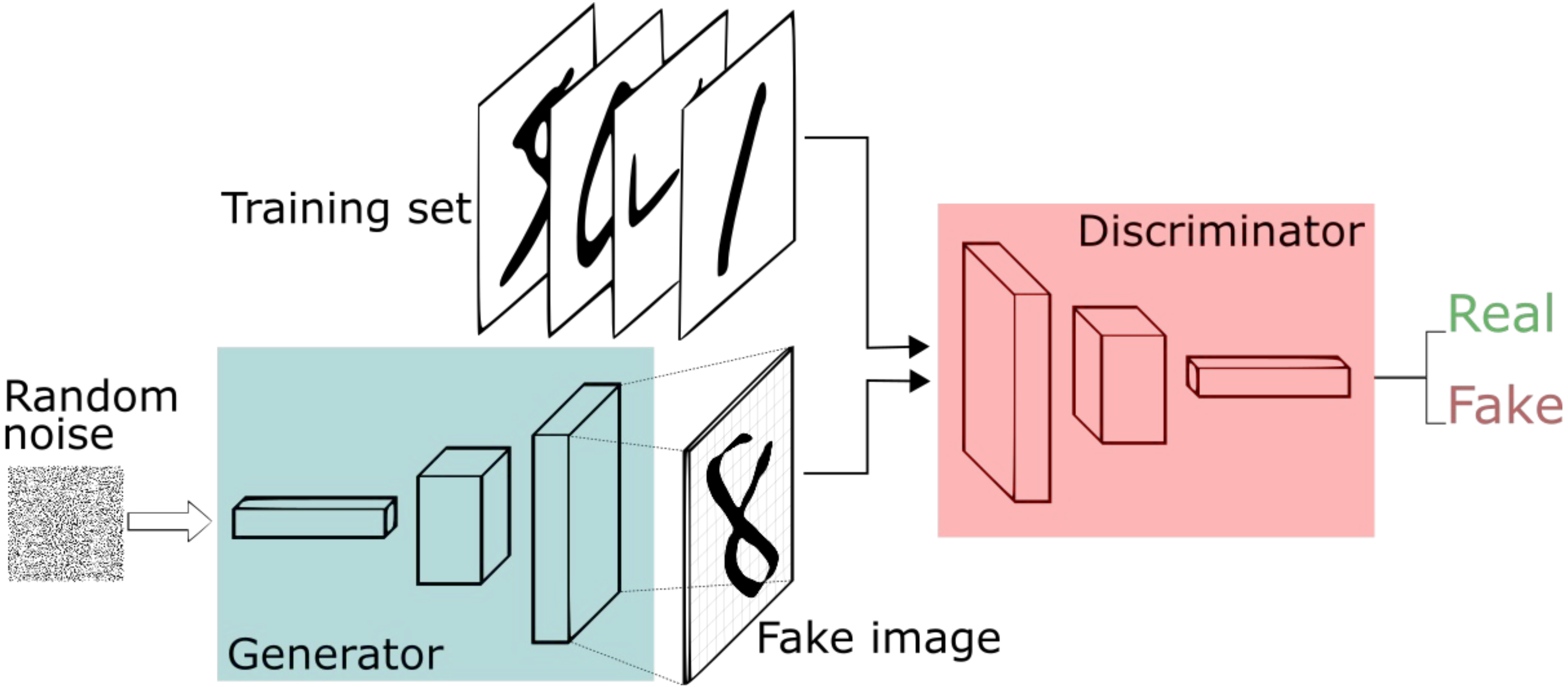
SeqGAN

GAN for Discrete Data

Generative Models

- Goal: build a model that generates real-looking samples which are indistinguishable from the real data
- How do we measure the distance between the real data distribution and the generated data distribution?
- What if we build a discriminator to judge whether a data instance is true or fake?
    - Leverage the strong power of deep learning based discriminative models

Generative Adversarial Networks (GANs)



Generative Adversarial Networks (GANs)

- Discriminator tries to distinguish the true data and the fake model-generated data
- Generator tries to fool the discriminator
- Two models are trained simultaneously to find a Nash equilibrium to a two-player non-cooperative game:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x\sim\mathbb{P}_r}[\log D(x)] + \mathbb{E}_{x\sim\mathbb{P}_G}[1 - \log D(x)]$$
$$= \int_x P_r(x)[\log D(x)] + G(x)[1 - \log D(x)]$$

Generative Adversarial Networks (GANs)

- In practice, we represent a limited family of P_G distributions via the function G(z; θ), z is sampled from a prior distribution

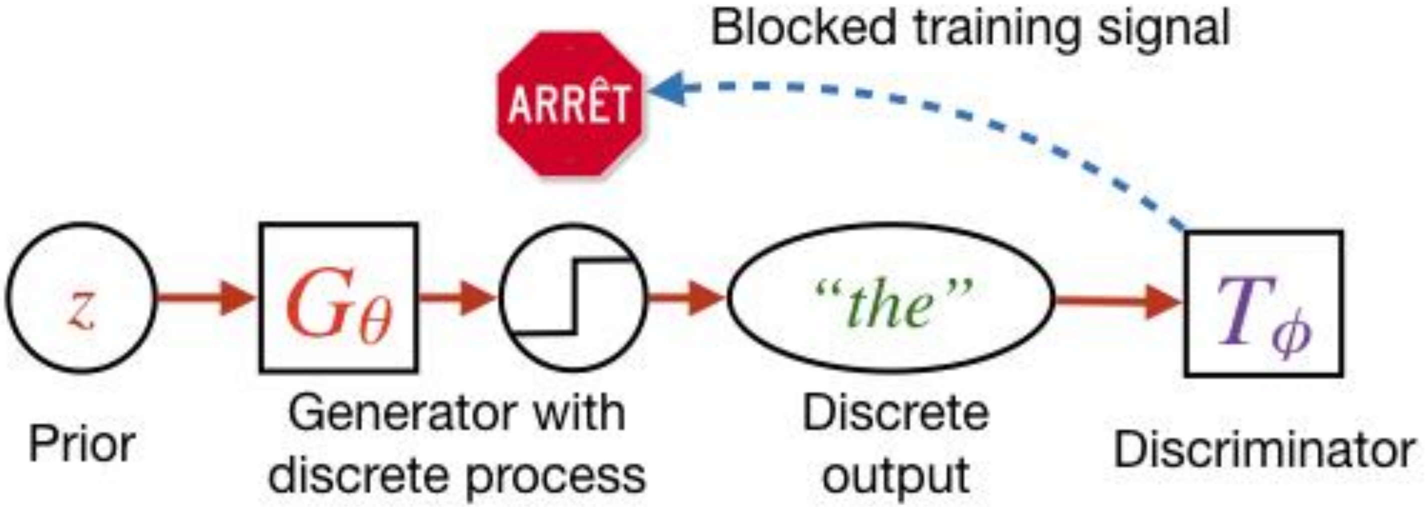
$$\min_G \max_D V(D, G)$$

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

$$= \int_x p_{\text{data}}(x) \cdot \log(D(x))dx + \int_z p_z(z) \cdot \log(1 - D(\boxed{G(z)}))dx$$

Directly optimize the differentiable mapping!

Problem with Discrete Data



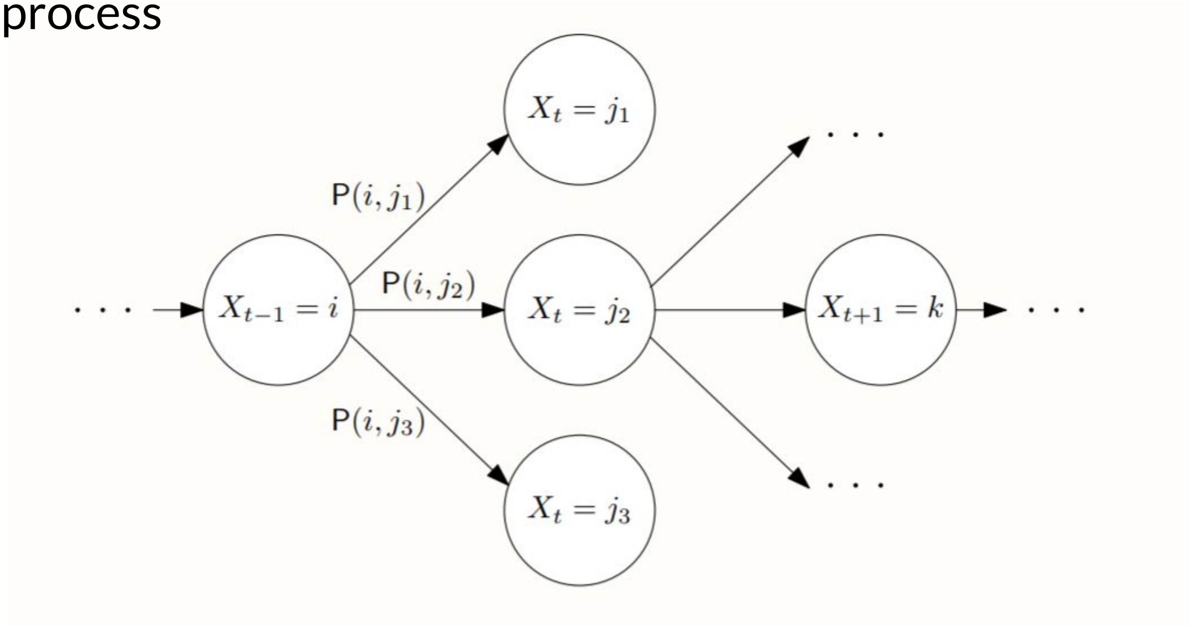
Problem with Discrete Data

- Directly optimize the p.d.f. G(x) with the guidance from discriminator instead of optimizing a non-differentiable transformation
- We could directly build a differentiable parametric distribution for discrete data using softmax function

$$\begin{aligned}\min_G \max_D V(D, G) &= \mathbb{E}_{x \sim \mathbb{P}_r}[\log D(x)] + \mathbb{E}_{x \sim \mathbb{P}_G}[1 - \log D(x)] \\ &= \int_x P_r(x)[\log D(x)] + G(x)[1 - \log D(x)]\end{aligned}$$

Problem with Sequential Discrete Data

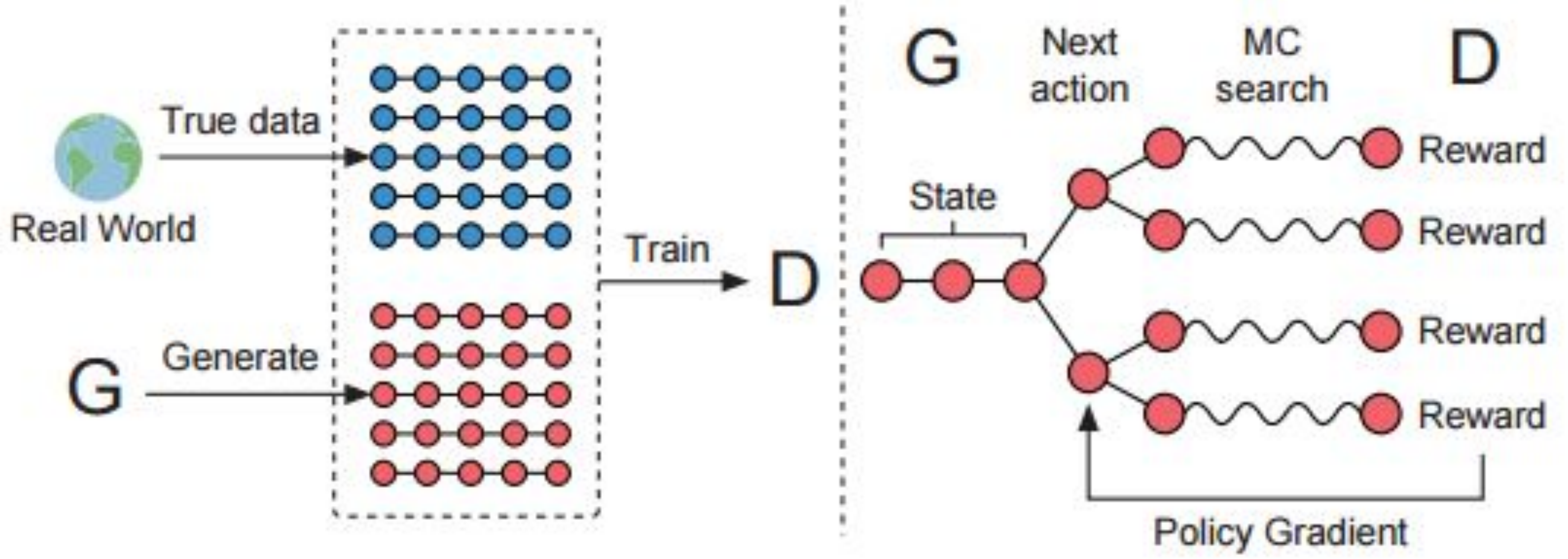
- Discriminator can only work on the whole sequence!
- Solution: model the generation of sequential discrete data as a sequential decision process



Problem with Sequential Discrete Data

- Introduce Reinforcement Learning ideas
- Generator is a reinforcement learning policy of generating a sequence
    - decide the next word to generate given the previous ones
- Discriminator provides the reward (i.e. the probability of being true data) for the whole sequence

Sequence GAN (Yu, Lantao, et al. 2017)

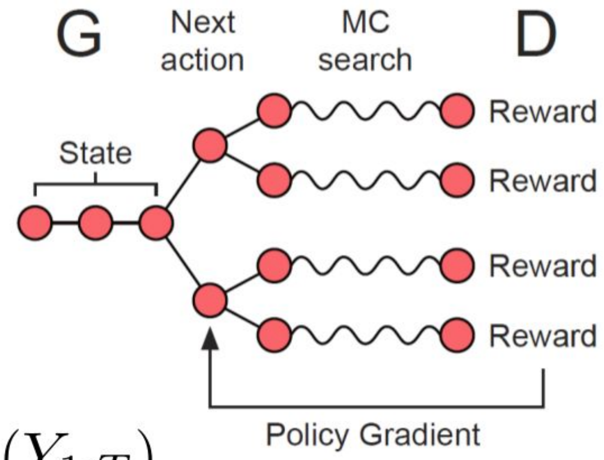


For generator:

- Objective: to maximize the expected reward

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

  - State-action value function $Q_{D_\phi}^{G_\theta}(s, a)$ is the expected accumulative reward that
    - Start from state *s*
    - Taking action *a*
    - And following policy *G* until the end
  - Reward is only on completed sequence (no immediate reward)



$$Q_{D_\phi}^{G_\theta}(s = Y_{1:T-1}, a = y_T) = D_\phi(Y_{1:T})$$

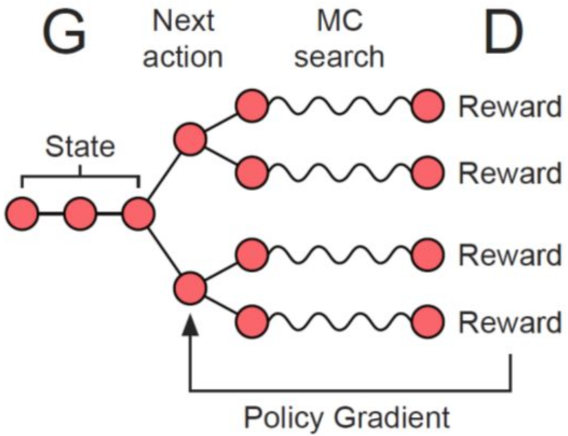
- Reward is only on completed sequence
    - No immediate reward
    - Then the last-step state-action value

$$Q_{D_\phi}^{G_\theta}(s = Y_{1:T-1}, a = y_T) = D_\phi(Y_{1:T})$$

- For intermediate state-action value
    - Use Monte Carlo search to estimate

$$\{Y_{1:T}^1, \ldots, Y_{1:T}^N\} = \mathrm{MC}^{G_\beta}(Y_{1:t}; N)$$

    - Following a roll-out policy G



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

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

Training Sequence Generator

Using Policy Gradient (REINFORCE):

[Richard Sutton et al. Policy Gradient Methods for Reinforcement Learning with Function Approximation. NIPS 1999.]

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

$$\theta \leftarrow \theta + \alpha_h \nabla_\theta J(\theta)$$

Training Sequence Discriminator

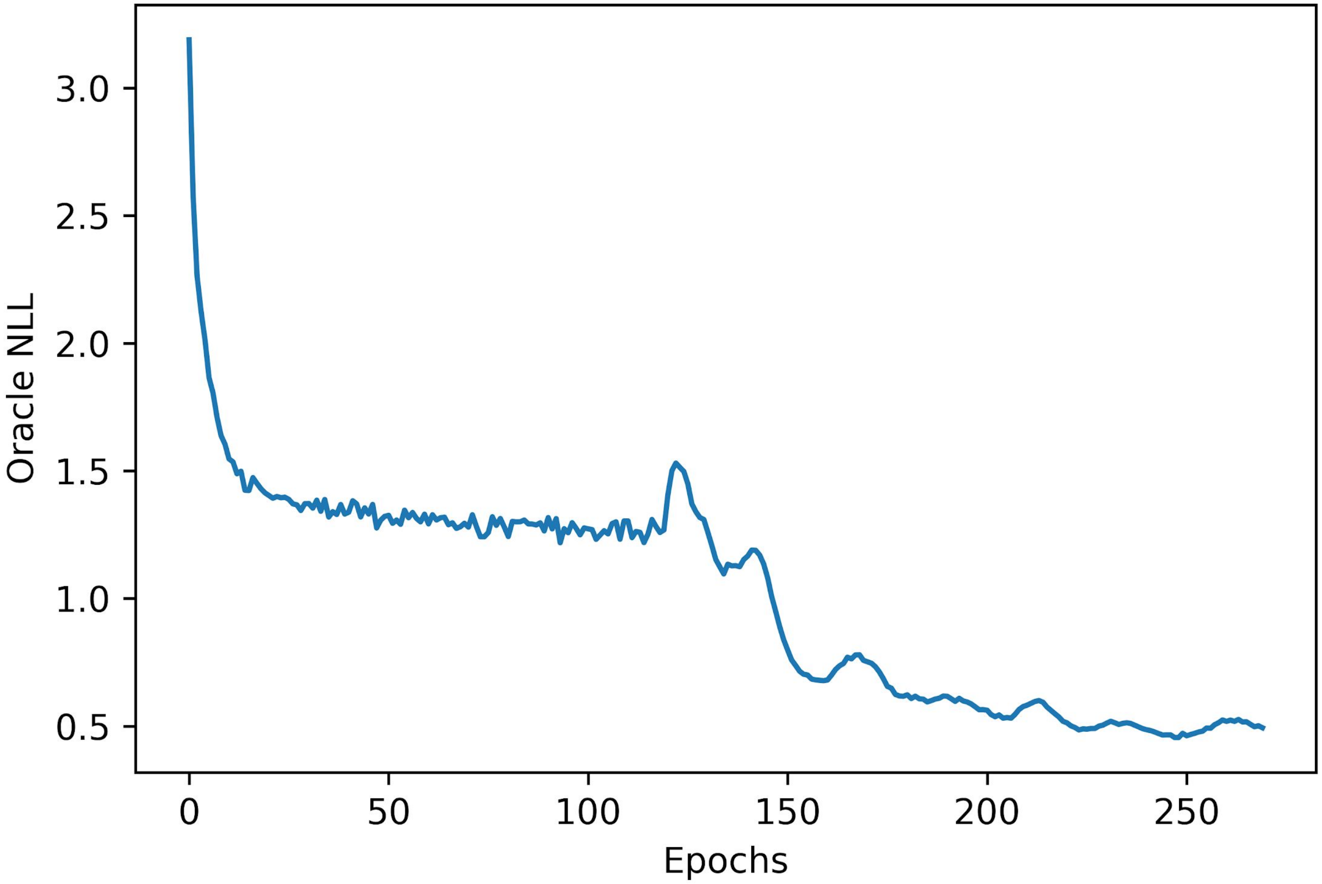
- Objective: standard bi-classification

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

Algorithm 1 Sequence Generative Adversarial Nets

Require: generator policy G_θ; roll-out policy G_β; discriminator D_ϕ; a sequence dataset $\mathcal{S} = \{X_{1:T}\}$

Initialize G_θ, D_ϕ with random weights θ, ϕ.
Pre-train G_θ using MLE on $\mathcal{S}$
$\beta \leftarrow \theta$
Generate negative samples using G_θ for training D_ϕ
Pre-train D_ϕ via minimizing the cross entropy
repeat
    for g-steps **do**
        Generate a sequence $Y_{1:T} = (y_1, \dots, y_T) \sim G_\theta$
        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_θ to generate negative examples and combine with given positive examples $\mathcal{S}$
        Train discriminator D_ϕ for k epochs by Eq. (5)
    end for
    $\beta \leftarrow \theta$
until SeqGAN converges



3.0
2.5
2.0
1.5
1.0
0.5
Oracle NLL
0
50
100
150
200
250
Epochs

Limitations

- Not capable of generating long sequences
    - I also failed to use it to generate relatively short reviews though (may be due to the high diversity of the corpus)
- Hierarchical Reinforcement Learning can be applied (Leaky GAN, 2017)

Thank you!