

Language model

☑ Trắc nghiệm 24 ☑ Thực hành 3 🔗 Chạy trên Kaggle ⬇ PDF bài

Language model dự đoán token tiếp theo từ các token đứng trước, như bàn phím điện thoại gợi ý từ tiếp theo khi gõ “em học”. Lặp lại việc đó nhiều lần là sinh được cả đoạn văn. Bài này bắt đầu với bigram để tính xác suất bằng tay, rồi chuyển sang Transformer, học các cách chọn token khi sinh văn bản (decoding) và chỉ số perplexity. Cuối bài là vì sao một câu trôi chảy chưa chắc là câu đúng.

MỤC TIÊU

- Tính xác suất của một câu bằng tích các xác suất có điều kiện.
- Tính cross-entropy và perplexity.
- So các cách sinh: greedy, sampling, temperature, top-k, top-p, beam search.
- Biết giới hạn của language model khi phát hiện văn bản do máy viết.

Mỗi bước chọn từ một phân phối

Với chuỗi token $t_1, \dots, t_L$, quy tắc nhân xác suất cho

$$p(t_1, \dots, t_L) = \prod_{i=1}^L p(t_i \mid t_1, \dots, t_{i-1}).$$

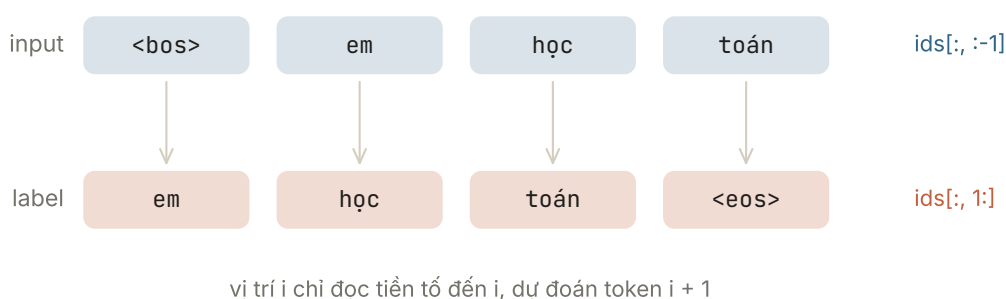
Ta thêm BOS (bắt đầu) và EOS (kết thúc) để model biết khi nào bắt đầu hoặc dừng. BOS thường là điều kiện đầu vào, EOS là một token cần dự đoán. Nếu không có EOS hoặc giới hạn số token, vòng sinh có thể không dừng.

Token ở đây theo tokenizer đã chọn, có thể là âm tiết, subword hoặc byte; “next word” chỉ là cách nói gần gũi, tên nhiệm vụ chính xác là next-token prediction. Đoán token tiếp theo, nối vào, rồi đoán tiếp là cách mọi model kiểu GPT sinh văn bản; gợi ý từ tiếp theo trên bàn phím điện thoại là phiên bản nhỏ của cùng ý.

Bigram chỉ nhìn một token trước: $p(t_i | t_{i-1})$; trigram nhìn hai token trước. N-gram dùng cửa sổ cố định nên dễ đếm và kiểm, nhưng không theo dõi được ngữ cảnh dài.

Bảng của nó còn phình rất nhanh: với vocabulary V , n-gram có $|V|^{n-1}$ ngữ cảnh, mỗi ngữ cảnh một phân phối trên $|V|$ token. Với $|V| = 10\,000$, trigram đã có 10^8 ngữ cảnh, mỗi ngữ cảnh cần 10^4 xác suất; một corpus vài triệu token chỉ gặp được một phần nhỏ trong số đó, nên phần lớn ô của bảng không bao giờ được đếm.

Causal transformer dùng self-attention trên tiền tố: tại vị trí i , nó không được đọc token tương lai dùng làm nhãn. BERT ở bài 6.3 nhìn hai chiều cho MLM; nó không phải causal language model chỉ cần gọi vòng sinh là thành GPT.



Input bỏ token cuối, label bỏ token đầu: vị trí i dự đoán token $i + 1$.

HỎI NHANH

Một model không dùng EOS, nên xác suất của một câu là tích xác suất của các token chữ. So câu hoàn chỉnh “em học” với câu “em học toán”, điều gì luôn đúng?

- A “em học toán” cao hơn, vì có thêm một từ hợp ngữ cảnh
- B “em học” không thấp hơn, vì nó là tiền tố của câu kia
- C Hai câu bằng nhau, vì có chung tiền tố “em học”
- D Không so được khi chưa biết $p(\text{toán} | \text{em học})$

Học một bigram trên corpus tự soạn

Ta dùng năm câu train. Từ vựng đích gồm bảy token chữ, EOS và UNK; BOS không nằm trong danh sách token được sinh. Vocabulary chỉ được rút từ train. Với token lạ ở evaluation, ta chuyển sang UNK, không âm thầm thêm cột mới.

Python

```
from collections import Counter, defaultdict
import numpy as np

train_sentences = ["em học toán", "em học tin", "em đọc sách", "bạn học toán", "bạn đọc sách"]
BOS, EOS, UNK = "<bos>", "<eos>", "<unk>"
target_vocab = sorted({token for text in train_sentences for token in text.split()} | {EOS, UNK})
bigram_counts = defaultdict(Counter)
for text in train_sentences:
    tokens = [BOS] + text.split() + [EOS]
    for previous, following in zip(tokens, tokens[1:]):
        bigram_counts[previous][following] += 1

print("vocabulary size:", len(target_vocab))
print("sau 'em':", dict(bigram_counts["em"]))
```

KẾT QUẢ

```
vocabulary size: 9
sau 'em': {'học': 2, 'đọc': 1}
```

Sau “em”, “học” xuất hiện hai lần và “đọc” một lần. Ước lượng không làm trơn cho $p(\text{học} \mid \text{em}) = 2/3$. Tương tự, “con” xuất hiện 200 lần và “con mèo” 30 lần cho $p(\text{mèo} \mid \text{con}) = 0.15$. Nhưng một cặp chưa thấy có xác suất 0, và chỉ một thừa số 0 cũng khiến cả tích chuỗi bằng 0. Ta dùng add- α smoothing, ở đây $\alpha = 1$:

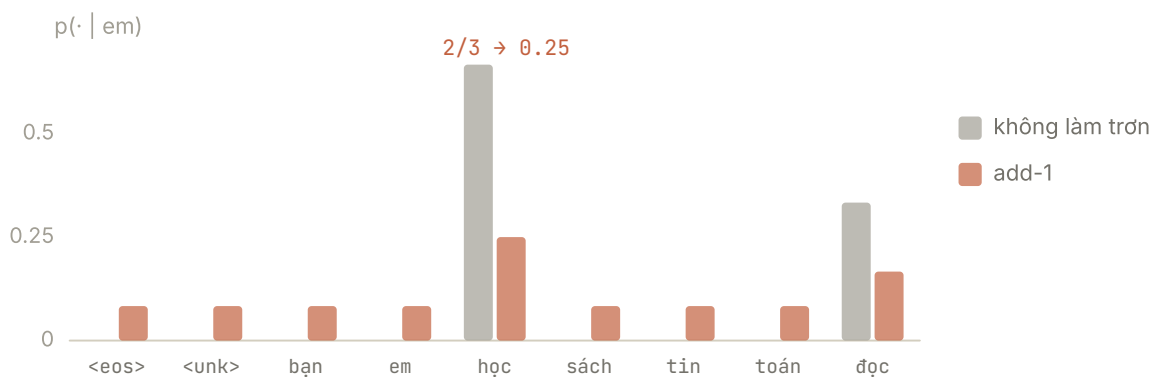
	<eos>	bạn	em	học	sách	tin	toán	đọc
<bos>		2	3					
em				2				1
bạn				1				1
học						1	2	
đọc					2			
toán	2							

ô trống: chưa gặp, xác suất 0 nếu không làm tròn

Bảng đếm bigram gần như toàn 0.

$$p(v | u) = \frac{c(u, v) + \alpha}{\sum_{w \in V} c(u, w) + \alpha |V|}$$

Mỗi token trong vocabulary được cộng cùng một số đếm giả α , rồi chia cho tổng mới nên phân phối vẫn cộng thành 1.



Không làm tròn, nhiều token có xác suất 0; add-1 chia cho mọi token một phần.

Python

```
alpha = 1.0
```

```
def next_distribution(previous):
    counts = bigram_counts.get(previous, Counter())
    denominator = sum(counts.values()) + alpha * len(target_vocab)
    return np.array([(counts[token] + alpha) / denominator for token in target_vocab])
```

```
em_distribution = next_distribution("em")
print("p(học|em):", round(em_distribution[target_vocab.index("học")], 3))
print("tổng xác suất:", round(em_distribution.sum(), 6))
```

KẾT QUẢ

p(học|em): 0.25
tổng xác suất: 1.0

Đối chiếu bằng tay: tử số là $2 + 1 = 3$, mẫu số $3 + 9 = 12$, nên xác suất giảm từ $2/3$ xuống 0.25. Add- α cho mỗi ngữ cảnh cùng một tổng số đếm giả $\alpha|V|$. Ngữ cảnh càng ít gặp và vocabulary càng lớn thì phần giả này càng lấn át số đếm thật, và phần lớn xác suất bị chia cho những token chưa từng đi sau ngữ cảnh đó.

Vì vậy với corpus thật, nếu dùng add- α thì thường chọn $\alpha < 1$ (chẳng hạn 0.5, 0.05 hay 0.01); dù vậy add- α vẫn kém cho language model, nên thực tế người ta dùng interpolation với n-gram ngắn hơn hoặc Kneser–Ney (xem Đọc thêm). α là hyperparameter phải chọn trên validation, không nhìn test rồi chọn số làm test đẹp nhất.

HỎI NHANH

Bỏ smoothing, tức $\alpha = 0$. Một câu validation có 20 cặp token. 19 cặp đã gặp trong train với xác suất cao, còn một cặp chưa gặp. NLL trung bình của câu bằng bao nhiêu?

- A Gần NLL của 19 cặp kia, vì một cặp chỉ chiếm 1/20
- B Bằng NLL của 19 cặp kia, vì cặp chưa gặp bị bỏ qua
- C Vô cùng, vì $-\log 0$ không hữu hạn
- D Lớn nhưng hữu hạn, vì xác suất ấy chỉ rất nhỏ

HỎI NHANH

Giữ đúng số đếm sau “em”, gồm “học” 2 lần và “đọc” 1 lần, nhưng vocabulary đích có 10 000 token. Với add-1, $p(\text{học} \mid \text{em})$ xấp xỉ bao nhiêu?

- A 0.0003
- B 0.25, như với vocabulary 9 token
- C 0.6, vì chỉ token đã gặp mới được cộng 1
- D 0.667, vì add-1 giữ nguyên tỉ lệ 2:1

Label dịch một vị trí, loss và perplexity

Trong huấn luyện next-token prediction, input và label lệch một vị trí. Với chuỗi [BOS, em, học, toán, EOS], input là [BOS, em, học, toán], label là [em, học, toán, EOS]. Token PAD trong label được bỏ khỏi loss. Khi dùng causal mask, các vị trí input được xử lý đồng thời trong train mà vẫn không đọc tương lai. Khi suy luận, ta chưa biết label thật nên phải dùng token vừa sinh làm input cho bước kế tiếp.

Average negative log-likelihood trên N token đích hợp lệ là

$$\mathcal{H} = -\frac{1}{N} \sum_{i=1}^N \log p(t_i \mid \text{tiền tố}), \quad \text{PPL} = e^{\mathcal{H}}.$$

Ta dùng log tự nhiên nên perplexity là $\exp(\text{loss})$. Nếu dùng log cơ số 2 thì là $2^{\mathcal{H}}$.

Perplexity có thể hiểu như số lựa chọn mà model đang phân vân: model cho xác suất đúng bằng 1/4 ở mọi bước có PPL bằng 4. Nó không phải số token sai hay số từ trong câu.

Cross-entropy 3 bit mỗi token cho $\text{PPL } 2^3 = 8$; model chia đều xác suất cho 50 000 token có PPL đúng 50 000. Ngay cả model hoàn hảo cũng không xuống dưới entropy của chính phân phối dữ liệu, nên PPL có một sàn lớn hơn 1.

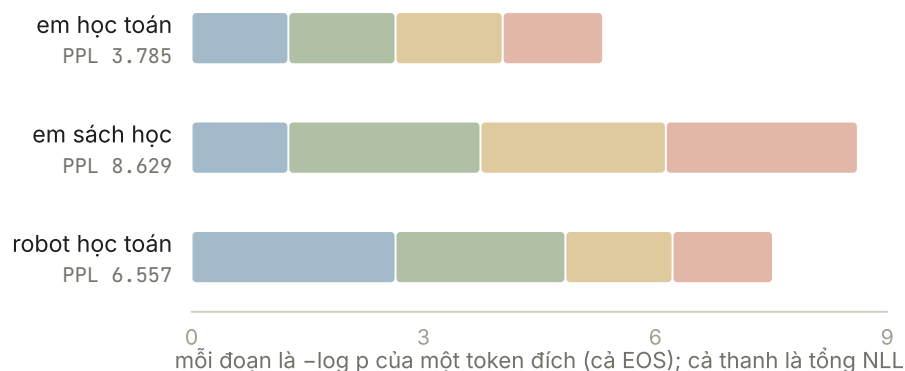
```
def sentence_nll(text):
    words = [word if word in target_vocab else UNK for word in text.split()]
    tokens = [BOS] + words + [EOS]
    log_probabilities = []
    for previous, following in zip(tokens, tokens[1:]):
        probability = next_distribution(previous)[target_vocab.index(following)]
        log_probabilities.append(np.log(probability))
    return -np.mean(log_probabilities), len(log_probabilities)

for text in ["em học toán", "em sách học", "robot học toán"]:
    nll, count = sentence_nll(text)
    print(text, "N=", count, "NLL=", round(nll, 3), "PPL=", round(np.exp(nll), 3))
print("PPL xác suất 1/4:", round(np.exp(-np.log(0.25)), 3))
```

KẾT QUẢ

```
em học toán N= 4 NLL= 1.331 PPL= 3.785
em sách học N= 4 NLL= 2.155 PPL= 8.629
robot học toán N= 4 NLL= 1.88 PPL= 6.557
PPL xác suất 1/4: 4.0
```

Số đếm N có cả EOS. So “em học toán” với “em sách học”: model đếm thích trật tự nó đã thấy hơn. Query có “robot” bị chuyển UNK, nên kết quả cũng phụ thuộc quy tắc xử lý từ lạ. Các câu trên là probe do ta đặt; câu đầu còn có trong train. Muốn bảo chất lượng language model, cần một corpus đánh giá độc lập, tách theo tài liệu hoặc thời gian.



Đảo trật tự hay gặp từ lạ đều làm perplexity tăng.

Causal neural language model tính cùng đại lượng theo cách khác: nó nhận cả chuỗi input một lần, trả một hàng logits cho mỗi vị trí, rồi so hàng thứ i với label thứ i bằng

cross-entropy. Bigram cũng viết được như vậy, vì hàng ở vị trí i chỉ phụ thuộc token i . Ta dùng log-xác suất làm logits: các xác suất đã cộng thành 1 nên softmax trả lại đúng phân phối cũ.

Python

```
import torch
import torch.nn.functional as F

def bigram_logits(tokens):
    # Hàng i chỉ đọc tokens[i]: một causal language model có ngữ cảnh dài 1.
    rows = [np.log(next_distribution(token)) for token in tokens]
    return torch.tensor(np.array(rows), dtype=torch.float32)

sequence = [BOS, "em", "học", "toán", EOS]
inputs, labels = sequence[:-1], sequence[1:]
logits = bigram_logits(inputs)
label_ids = torch.tensor([target_vocab.index(token) for token in labels])
print("input:", inputs)
print("label:", labels)
print("logits shape:", tuple(logits.shape))
print("cross-entropy:", round(F.cross_entropy(logits, label_ids).item(), 3))

# Thêm hai vị trí đệm; label -100 bị cross_entropy bỏ qua theo mặc định.
padded_logits = bigram_logits(inputs + [EOS, EOS])
padded_labels = torch.cat([label_ids, torch.tensor([-100, -100])])
print("thêm 2 vị trí đệm:", round(F.cross_entropy(padded_logits, padded_labels).item(), 3))
```

KẾT QUẢ

```
input: ['<bos>', 'em', 'học', 'toán']
label: ['em', 'học', 'toán', '<eos>']
logits shape: (4, 9)
cross-entropy: 1.331
thêm 2 vị trí đệm: 1.331
```

Cross-entropy 1.331 khớp NLL của “em học toán” ở trên: loss huấn luyện của language model chính là NLL trung bình trên các token đích. Hai vị trí đệm mang label -100 nên loss giữ nguyên, dù hàng logits ở đó chứa gì.

Với batch thật, input có shape (B, L) , logits có shape $(B, L, |V|)$, và label là chính chuỗi token dịch trái một vị trí, nên mỗi chuỗi L token cho $L - 1$ cặp (vị trí, token kế).

Quên dịch label, tức dùng chính `input_ids` làm nhãn ở cùng vị trí, thì model chỉ học chép lại token đang đọc: loss xuống rất thấp mà không học được gì về token tiếp theo.

HỎI NHANH

Batch `ids` có shape $(8, 128)$. Model trả về `logits` shape $(8, 128, V)$, trong đó `logits` ở vị trí i dự đoán token $i + 1$. Đưa cặp nào vào cross-entropy mới đúng?

A `logits[:, :-1]` với `ids[:, 1:]`

B `logits[:, 1:]` với `ids[:, :-1]`

C `logits[:, :-1]` với `ids[:, :-1]`

D `logits` với `ids`, không cắt gì

Không so PPL của hai tokenizer khác nhau như cùng một thước đo (bài 4.5): chuỗi bị chia khác làm N và phân phối đích đổi. Phải nói rõ tokenizer, vocabulary, cách tính BOS/EOS, truncation và đơn vị trung bình.

Loss phải cộng theo tổng số token hợp lệ; trung bình loss của các batch dài ngắn bằng trọng số ngang nhau có thể làm lệch phép đo.

HỎI NHANH

Hai câu dùng để đánh giá có 2 và 8 token hợp lệ. Muốn NLL trung bình theo token thì gộp thế nào?

A Lấy trung bình hai loss câu, mỗi câu trọng số 1/2

B Bỏ câu ngắn, vì 2 token quá ít để đo

C Cộng NLL của cả 10 token rồi chia cho 2 câu

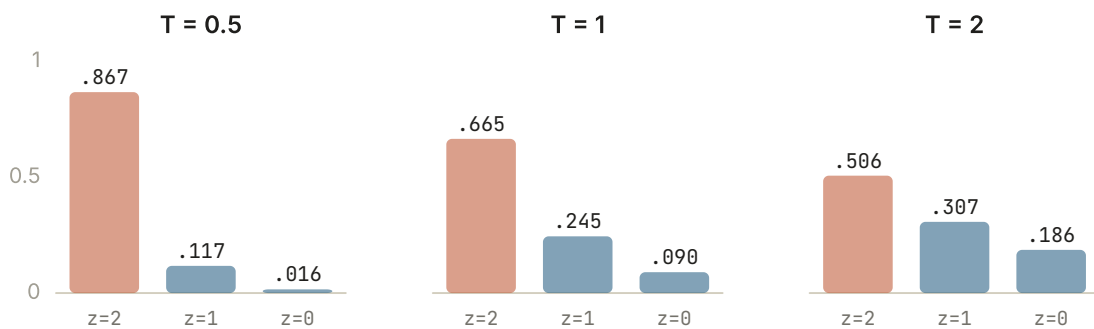
D Cộng NLL của cả 10 token rồi chia cho 10

Từ logits đến token được chọn

Causal neural language model trả logits z_v cho mọi token trong vocabulary. Với temperature $T > 0$, xác suất dùng khi sampling là

$$p_T(v) = \frac{e^{z_v/T}}{\sum_w e^{z_w/T}}.$$

T nhỏ làm phân phối nhọn, T lớn làm phẳng. Greedy luôn lấy token có logits lớn nhất; chia mọi logits cho cùng một $T > 0$ không đổi thứ tự, nên temperature không ảnh hưởng tới greedy. Không tính công thức với $T = 0$; “temperature 0” trong một số API là quy ước chuyển sang greedy, không phải phép chia toán học hợp lệ.



thứ tự không đổi; T nhỏ làm nhọn, T lớn làm phẳng

Temperature nhỏ làm phân phối nhọn, lớn làm phẳng; thứ tự giữ nguyên.

Python

```
def softmax_temperature(logits, temperature):
    if temperature <= 0:
        raise ValueError("Temperature phải dương trong công thức sampling")
    scaled = logits / temperature
    weights = np.exp(scaled - scaled.max())
    return weights / weights.sum()

example_logits = np.array([2.0, 1.0, 0.0])
for temperature in [0.5, 1.0, 2.0]:
    probability = softmax_temperature(example_logits, temperature)
    print("T=", temperature, "p=", probability.round(3))
```

KẾT QUẢ

$T = 0.5$ $p = [0.867 \ 0.117 \ 0.016]$

$T = 1.0$ $p = [0.665 \ 0.245 \ 0.09 \]$

$T = 2.0$ $p = [0.506 \ 0.307 \ 0.186]$

Token đầu vẫn có xác suất lớn nhất ở cả ba hàng; chỉ độ chênh thay đổi. Khi $T \rightarrow 0^+$, phân phối dồn về argmax nên sampling tiến tới greedy; khi $T \rightarrow \infty$, nó tiến tới phân phối đều trên cả vocabulary. Sampling lấy ngẫu nhiên theo phân phối, không nhất thiết chọn token đầu. Cố định seed giúp tái tạo cùng thí nghiệm trong cùng môi trường, nhưng không biến sampling thành kiểm chứng kiến thức.

Hai cách lọc trước khi rút:

- **Top-k** giữ đúng k token có điểm cao nhất rồi chuẩn hóa lại.
- **Top-p** sắp token theo xác suất giảm dần, giữ đoạn đầu ngắn nhất có tổng xác suất đạt ít nhất p , rồi chuẩn hóa lại. Số token giữ thay đổi theo độ nhọn; phải giữ cả token làm tổng vượt ngưỡng.

Python

```
def top_p_distribution(probability, threshold):
    if not 0 < threshold <= 1:
        raise ValueError("top-p nằm trong (0, 1]")
    order = np.argsort(-probability, kind="stable")
    sorted_probability = probability[order]
    # Giữ token khi tổng trước nó còn dưới ngưỡng, gồm cả token vượt ngưỡng.
    keep = np.cumsum(sorted_probability) - sorted_probability < threshold
    result = np.zeros_like(probability)
    result[order[keep]] = sorted_probability[keep]
    return result / result.sum()

base_probability = softmax_temperature(example_logits, 1.0)
filtered = top_p_distribution(base_probability, 0.8)
print("base:", base_probability.round(3))
print("top-p=0.8:", filtered.round(3))
print("số token giữ:", np.count_nonzero(filtered))
```

KẾT QUẢ

base: [0.665 0.245 0.09]

top-p=0.8: [0.731 0.269 0.]

số token giữ: 2

Top-p 0.8 giữ hai token: token đầu (0.665) chưa đạt 0.8, còn hai token đầu (0.91) đã đạt. Top-k giữ một số token cố định dù phân phối nhọn hay phẳng; top-p tự co lại khi model chắc chắn và nói ra khi model phân vân. Các thư viện như Hugging Face áp temperature vào logits trước, rồi mới lọc top-k và top-p trên phân phối đã đổi.

Ví dụ “và” 0.5, “nhưng” 0.3, “nên” 0.1, “thì” 0.1 với top-p 0.75: token đầu chưa đủ, hai token đầu đạt 0.8, nên chỉ chọn giữa “và” và “nhưng”.

Cũng vì sampling mà hỏi chatbot cùng một câu hai lần có thể nhận hai câu trả lời khác nhau. Việc cần chính xác như dịch văn bản pháp lý hợp với greedy hay beam search; viết truyện cần đa dạng thì hợp với sampling có temperature vừa phải.

HỎI NHANH

Vẫn logits (2, 1, 0), nhưng hạ temperature từ 1 xuống 0.5 rồi mới lọc. Top-p 0.8 và top-k 2 lần lượt giữ bao nhiêu token?

A Top-p giữ 2, top-k giữ 2

B Top-p giữ 1, top-k giữ 2

C Top-p giữ 1, top-k giữ 1

D Top-p giữ 2, top-k giữ 1

HỎI NHANH

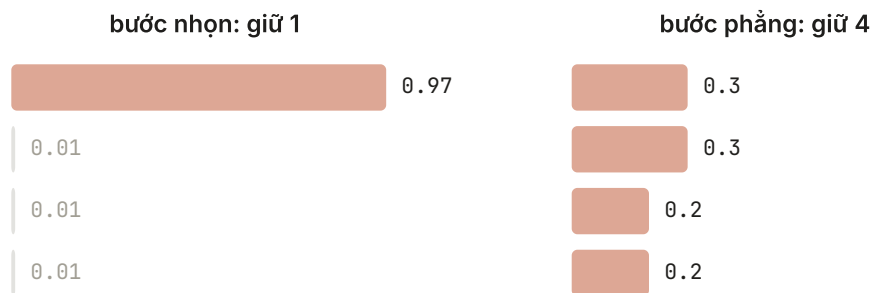
Hai bước sinh có phân phối đã sắp là (0.97, 0.01, 0.01, 0.01) và (0.3, 0.3, 0.2, 0.2). Top-p 0.9 giữ bao nhiêu token ở mỗi bước?

A 2 và 2, như top-k 2

B 4 và 1

C 1 và 4

D 1 và 3



top-p 0.9 giữ token cho tới khi tổng dồn đạt 0.9, kể cả token vượt ngưỡng

Top-p 0.9: bước nhọn giữ 1 token, bước phẳng giữ cả 4.

Sinh từng bước và vì sao greedy không tối ưu cả chuỗi

Ta cho bigram sinh đến EOS hoặc tối đa 6 token. Greedy dùng argmax, sampling dùng RNG seed 0. Khi hòa điểm, thứ tự target_vocab quyết định greedy, nên quy tắc đó phải được biết.

Python

```
def generate_bigram(sample=False, seed=0, max_tokens=6):  
    rng = np.random.default_rng(seed)  
    previous = BOS  
    generated = []  
    for _ in range(max_tokens):
```

```

probability = next_distribution(previous)
if sample:
    index = rng.choice(len(target_vocab), p=probability)
else:
    index = int(probability.argmax())
token = target_vocab[index]
if token == EOS:
    break
generated.append(token)
previous = token
return " ".join(generated)

```

```

print("greedy:", generate_bigram())
print("sampling seed 0:", generate_bigram(sample=True))

```

KẾT QUẢ

```

greedy: em học toán
sampling seed 0: em em

```

Greedy đi đúng đường được đếm nhiều nhất: “em” (3 lần sau BOS), “học”, “toán”, rồi EOS. Sampling với seed 0 ra “em em”, một cặp chưa từng xuất hiện: smoothing cho nó xác suất $1/12$, đủ để thỉnh thoảng được rút. Giới hạn số token giữ cho vòng sinh dừng, nhưng không sửa chất lượng. Chọn seed cho ra một câu hay rồi báo như thể model luôn sinh như vậy là chọn mẫu đẹp, giống chọn seed tốt nhất trên test.

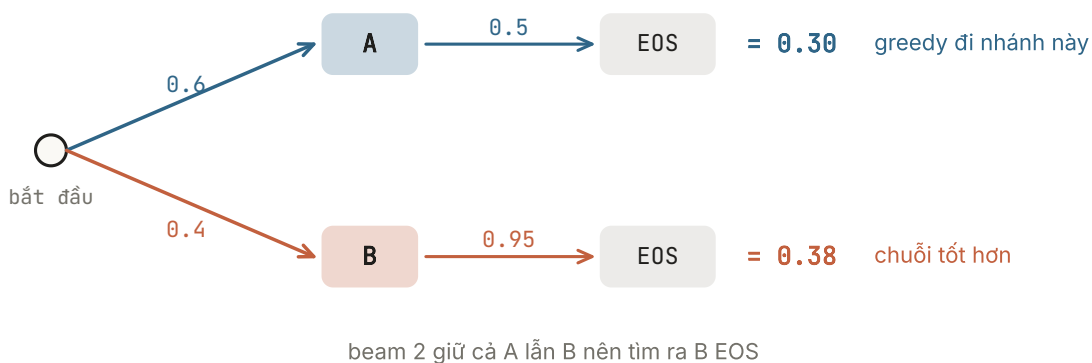
Câu trả lời bị cắt ngang giữa câu thường do chạm giới hạn số token sinh trước khi gặp EOS. Greedy còn hay rơi vào vòng lặp: khi cụm vừa sinh làm chính nó có xác suất cao nhất ở bước sau, argmax cứ chọn lại mãi; sampling hoặc phạt lặp (repetition penalty) phá vòng ấy.

HỎI NHANH

Bỏ giới hạn số token. Sinh greedy trên một bigram có 9 token đích, trong đó có EOS. Đã sinh 9 token mà chưa gặp EOS. Điều gì là chắc chắn?

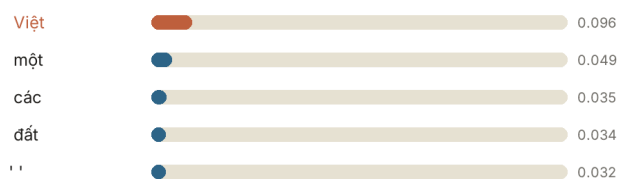
- A Sắp gặp EOS, vì smoothing cho EOS xác suất dương
- B Chưa kết luận được, phải chạy tiếp mới biết
- C Chỉ lặp mãi nếu có hai token liên nhau giống nhau
- D Sẽ không bao giờ gặp EOS, vì chuỗi đã vào một chu trình

Greedy tối ưu mỗi bước, không bảo đảm tối ưu cả chuỗi. Ví dụ bước đầu $p(A) = 0.6$, $p(B) = 0.4$. Tiếp theo $p(\text{EOS} | A) = 0.5$, còn $p(\text{EOS} | B) = 0.95$. Greedy chọn A, chuỗi A EOS có xác suất $0.6 \cdot 0.5 = 0.30$, còn B EOS có $0.4 \cdot 0.95 = 0.38$.



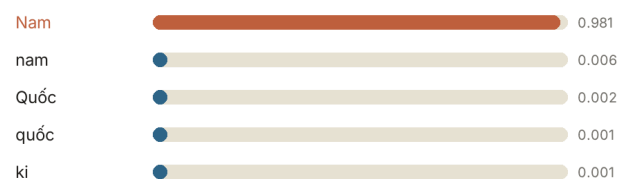
Greedy chọn A trước nên lỡ chuỗi B EOS có xác suất cao hơn; beam 2 tìm ra nó.

Hà Nội là thủ đô của



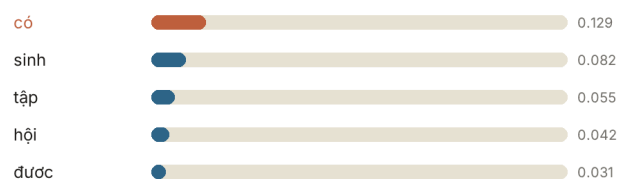
bước 1: 5 token có xác suất cao nhất; greedy chọn "Việt" (0.10)

Hà Nội là thủ đô của Việt



bước 2: 5 token có xác suất cao nhất; greedy chọn "Nam" (0.98)

Hà Nội là thủ đô của Việt Nam, là nơi



bước 6: 5 token có xác suất cao nhất; greedy chọn "có" (0.13)

Hà Nội là thủ đô của Việt Nam, là nơi có nhiều người dân

sau 9 bước greedy

Model viết tiếp từng token: mỗi bước lấy token có xác suất cao nhất rồi nối vào.

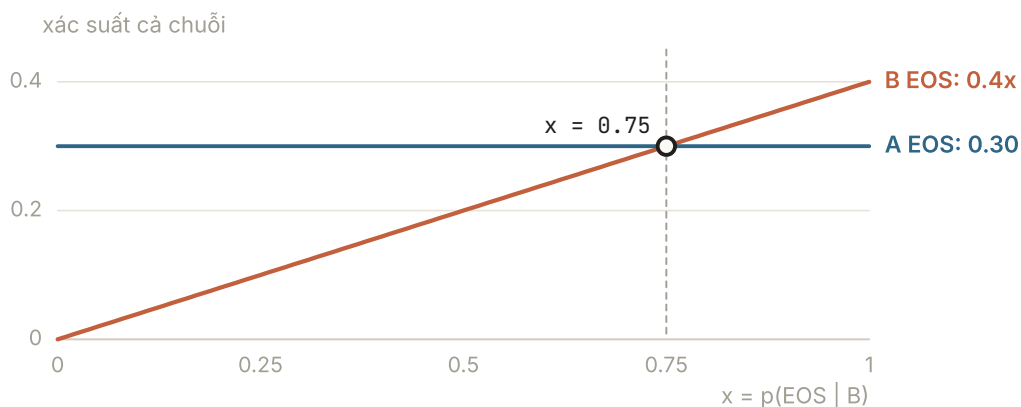
Beam search độ rộng k giữ k tiền tố có tổng log-xác suất cao nhất thay vì một. Mỗi bước, mỗi tiền tố được nối với mọi token, tạo $k|V|$ ứng viên, rồi chỉ k ứng viên tốt nhất được giữ; tiền tố vừa gặp EOS được cắt riêng thành câu hoàn chỉnh. Với $k = 2$, ví dụ trên giữ cả A lẫn B ở bước đầu nên tìm ra B EOS; với $k = 1$, beam search chính là greedy.

Beam hữu hạn vẫn chỉ tìm gần đúng: tiền tố bị loại ở một bước không bao giờ quay lại. Mỗi token nối thêm cộng một số hạng $\log p \leq 0$, nên tổng log không bao giờ tăng và beam thuận ưu ái câu ngắn; length penalty chia điểm theo độ dài và vì vậy đổi mục tiêu xếp hạng. Chuỗi xác suất cao nhất cũng không nhất thiết tốt nhất theo metric của nhiệm vụ.

HỎI NHANH

Giữ $p(A) = 0.6$, $p(B) = 0.4$ và $p(\text{EOS} | A) = 0.5$ như hình, nhưng đổi $p(\text{EOS} | B)$ thành x . Chỉ xét hai chuỗi A EOS và B EOS. Khi nào chuỗi greedy trả về không kém chuỗi còn lại?

- A Với mọi x , vì A thắng ở bước đầu
- B Khi $x \leq 0.5$, tức không vượt $p(\text{EOS} | A)$
- C Khi $x \leq 0.75$
- D Khi $x < 0.6$, tức nhỏ hơn $p(A)$



Greedy chỉ không kém khi $x \leq 0.75$.

Perplexity và văn bản bị máy sửa

Ta có thể dùng loss language model làm một đặc trưng để phát hiện văn bản do máy viết hoặc sửa: văn phong máy có thể khác corpus tham chiếu. Nhưng đây là một bài phân loại riêng, cần nhận biết chắc nguồn gốc của từng văn bản; không có luật “PPL thấp là do AI”. Một học sinh viết câu thông dụng có thể có loss thấp; một model sinh thuật ngữ hiếm có thể có loss cao. Dịch, sửa chính tả, tác giả và chủ đề còn là các yếu tố gây nhiễu.

Muốn thử, ta cần cặp bản gốc/bản sửa với cách tạo được ghi lại, chia theo nguồn văn bản để hai bản của cùng một đoạn không sang hai tập. Chọn ngưỡng bằng validation;

báo false positives và false negatives trên nhóm tác giả, chủ đề và cách sửa chưa thấy. Không dùng một score để kết luận gian lận của một người.

HỎI NHANH

Một detector gắn nhãn “máy viết” khi PPL dưới một ngưỡng. Ngưỡng được chọn trên validation gồm bài văn của học sinh chuyên Văn và văn bản do model sinh. Dem detector chấm bài tiếng Anh của học sinh mới học, câu ngắn, từ thông dụng. Rủi ro lớn nhất là gì?

- A False negative tăng, vì câu ngắn có PPL cao
- B False positive tăng, vì câu đơn giản có PPL thấp
- C Không có gì thay đổi, vì ngưỡng đã được chọn trên validation
- D Không có gì thay đổi, vì PPL không phụ thuộc người viết

Khi đi thi

- In một cặp input và label: label phải dịch đúng một vị trí, vị trí đệm mang -100.
- Đổi token cuối của input thì logits ở mọi vị trí trước phải giữ nguyên; không thì causal mask sai.
- So perplexity chỉ khi cùng tokenizer và cùng cách đếm token.

Tóm tắt

- Language model dự đoán token tiếp theo; xác suất chuỗi là tích các xác suất có điều kiện, và EOS biến một tiền tố thành câu hoàn chỉnh.
- N-gram đếm trên cửa sổ cố định nên số đếm thưa; add- α tránh xác suất 0 nhưng quá mạnh khi vocabulary lớn.

- Train dùng label dịch một vị trí và causal mask; loss là NLL trung bình trên token hợp lệ, perplexity là exp của nó và chỉ so được khi cùng giao thức.
- Temperature, top-k, top-p đổi phân phối lấy mẫu; greedy và beam search tìm chuỗi xác suất cao nhưng chỉ gần đúng.
- Xác suất cao không bảo đảm đúng; điểm language model không xác định chắc chắn ai viết một văn bản.

Kiểm tra cuối bài

CÂU 1

Khi train, một causal LM tính dự đoán token kế tiếp cho cả 512 vị trí của một chuỗi chỉ trong một lần forward. Dùng chính model này để sinh tiếp 512 token mới thì cần gì?

- A 512 bước nối tiếp nhau, vì bước sau cần token vừa sinh
- B 512 bước chạy song song, vì mỗi vị trí có mask riêng
- C Một lần forward, vì causal mask cho tính mọi vị trí cùng lúc
- D Một lần forward, nếu lưu lại phần đã tính cho tiền tố

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 239

CÂU 2

Dưới đây là ba cách tạo mask cho chuỗi dài L , dùng với `F.scaled_dot_product_attention`, nơi `True` nghĩa là được đọc.

```
a = ~torch.triu(torch.ones(L, L), diagonal=1).bool()
b = torch.tril(torch.ones(L, L)).bool()
c = torch.ones(L, L).bool()
for i in range(L):
    c[i, i+1:] = False
```

Cách nào cho đúng causal mask, tức vị trí i đọc được các vị trí $0, \dots, i$?

- A Chỉ c, vì chỉ vòng lặp mới giữ đúng đường chéo
- B Chỉ a và b, vì vòng lặp ở c bỏ mất đường chéo
- C Chỉ b và c, vì a bị đảo nên cấm cả quá khứ
- D Cả ba, vì đều là tam giác dưới gồm cả đường chéo

PHÒNG THEO NOAI 2026 Singapore · Chung kết · Câu 18

CÂU 3

Một đội train một causal LM nhỏ. Sau 300 bước, validation perplexity đã là 1.03, nhưng sinh greedy thì ra chuỗi vô nghĩa. Label đã được dịch đúng một vị trí. Nên kiểm gì trước?

- A Tăng temperature, vì greedy làm chuỗi lặp lại
- B Train thêm, vì perplexity chưa về đúng 1
- C Kiểm causal mask, vì PPL gần 1 gợi ý tương lai bị lộ
- D Đổi tokenizer, vì perplexity phụ thuộc tokenizer

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 231

CÂU 4

Một causal LM vừa khởi tạo, vocabulary có 32 000 token, và logits ở bước đầu gần như bằng nhau. Validation perplexity ở bước 0 khoảng bao nhiêu?

- A Khoảng $1/32\,000$, tức xác suất của mỗi token
- B Khoảng 32 000, bằng kích thước vocabulary
- C Khoảng 10.4, tức $\ln 32\,000$
- D Khoảng 4.5, tức $\log_{10} 32\,000$

CÂU 5

Một đội sinh 5 lời giải cho mỗi đề, chạy test và giữ lời giải qua được nhiều test nhất. Với `do_sample=True`, `temperature=0.2`, cả 5 lời giải gần như giống hệt nhau. Nên chỉnh gì?

- A Giữ temperature 0.2, chỉ đổi seed giữa 5 lần sinh
- B Tăng temperature, và chọn mức cụ thể trên tập dev
- C Chuyển sang greedy để 5 lần sinh ổn định hơn
- D Tăng `max_new_tokens` để các lời giải dài khác nhau

PHÒNG THEO [NOAI 2026 Singapore](#) · [Vòng sơ loại](#) · [Câu 240](#)

CÂU 6

Cấu hình sinh là `do_sample=True`, `top_k=1`, `temperature=1.5`. Chạy với 5 seed khác nhau, cả 5 output trùng nhau đến từng token. Vì sao?

- A Vì temperature 1.5 quá cao nên sampling bị tắt
- B Vì seed không được truyền vào bộ sinh nên RNG không đổi
- C Vì top-k 1 giữ một token ngẫu nhiên, cố định theo prompt
- D Vì top-k 1 chỉ giữ argmax, nên lần nào cũng là greedy

PHÒNG THEO [NOAI 2026 Singapore](#) · Vòng sơ loại · Câu 241

CÂU 7

Mọi câu có đúng hai token. Ở bước đầu, $p(A) = 0.4$, $p(B) = 0.35$, $p(C) = 0.25$. Token thứ hai tốt nhất sau mỗi nhánh có xác suất có điều kiện 0.6 sau A, 0.5 sau B và 1.0 sau C. Beam search độ rộng 2 trả về câu có xác suất bao nhiêu?

- A 0.24, vì câu tốt nhất đã bị loại ngay ở bước đầu
- B 0.24, và đó chắc chắn là câu tối ưu toàn cục
- C 0.25, vì beam search xét đủ mọi nhánh
- D 0.175, vì beam search bỏ nhánh mà greedy chọn

PHÒNG THEO [NOAI 2025 Singapore](#) · Phần 1 · Câu 18

CÂU 8

Hai đội báo PPL trên cùng tập validation, cùng cách tách từ. Đội X thay mọi từ xuất hiện dưới 5 lần trong train bằng UNK, còn vocabulary 2000, và được PPL 45. Đội Y giữ vocabulary 20 000, được PPL 80. Ta kết luận được gì?

- A X tốt hơn, vì PPL thấp hơn trên cùng một tập
- B Y tốt hơn, vì vocabulary lớn mô tả câu đầy đủ hơn
- C Chưa so được, vì đoán UNK dễ hơn đoán đúng từ hiếm
- D Chưa so được, vì phải chia PPL cho kích thước vocabulary

CÂU 9

Một detector văn bản bị máy sửa bắt được 90% số bài bị sửa, và gán nhầm 5% số bài do người viết. Hai tỉ lệ này đo trên dữ liệu cùng phân phối với nơi dùng. Trong một khối lớp, khoảng 2% số bài thật sự bị máy sửa. Trong những bài bị gán nhãn “máy sửa”, khoảng bao nhiêu phần trăm thật sự bị sửa?

- A Khoảng 2%, bằng tỉ lệ bài bị sửa trong khối
- B Khoảng 90%, bằng tỉ lệ bắt được bài bị sửa
- C Khoảng 95%, vì chỉ 5% bài người viết bị gán nhầm
- D Khoảng 27%

CÂU 10

Một bản top-p viết `keep = np.cumsum(sorted_probability) < threshold`, thiếu phần trừ xác suất của chính token như trong bài. Với phân phối (0.9, 0.06, 0.04) và top-p 0.8, hàm trả về gì?

- A Toàn nan, vì không token nào được giữ
- B (1, 0, 0), giống bản đúng
- C (0.9, 0.06, 0.04), không lọc gì
- D Báo ValueError, vì ngưỡng không hợp lệ

Đọc thêm

- Jurafsky, Martin — [N-gram Language Models](#). Đọc smoothing, evaluation và perplexity.
- Jurafsky, Martin — [Transformers and Pretraining](#). Nối next-token training với causal attention và decoding.
- Dive into Deep Learning — [Language models](#). Đọc cách tạo input/target dịch vị trí.
- Holtzman et al. — [The Curious Case of Neural Text Degeneration](#). Bài đề xuất top-p (nucleus sampling); đọc phần so greedy, beam và sampling về độ lặp và độ đa dạng.
- Hugging Face — [Generation](#). Kiểm ý nghĩa `do_sample`, `temperature`, `top_k`, `top_p` và giới hạn sinh của API.