□□□□□□□□□

□3□ (A)

□□□

nsformer を実行

ンを選ぶ処理を、自分で実装

単な応答プログラムを作成

デルをダウンロードし、自分の環境で実行

□□□□□

測も、選んでつなげるループも同一

回数の表が学習済みの関数になった点

2回のものをそのまま利用可能

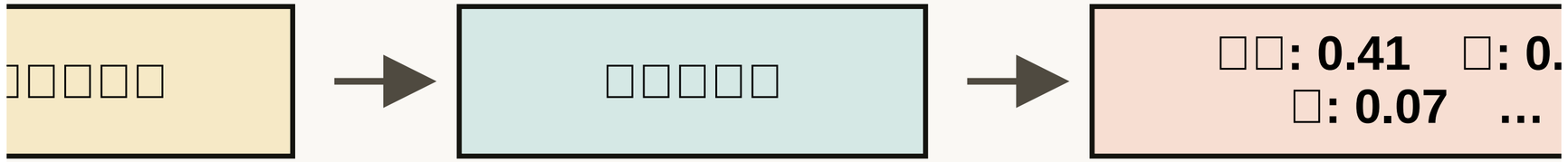
	第2回：マルコフ連鎖	今日：言語モデル
	`table[word]` を引く	`model(□□)` を呼ぶ
	数えた回数の表	学習した重み
	直前 1〜2 語	直前 数千トークン
	`random.choices` + 温度	サンプリング + 温度

□□□

こまでの文字列」

ではなく、語彙それぞれの確率

、1ステップの繰り返しで生成



のは、一度に1トークンしか返さないため

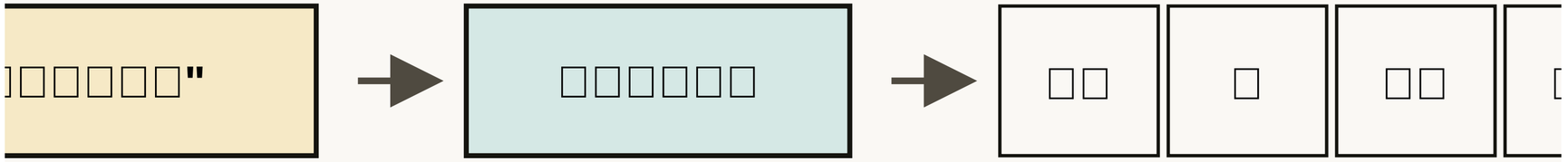
□□□□□□□□

、単語とも1文字とも限らない

文字列をまとめた、統計的な単位

`oftmax` を通すと確率

```
softmax(logits, dim=-1)   # □□□ 0 □□□□□ 1
```



□□□□□

ザで文字列を ID に変換

`s)` の形は (バッチ, 系列長, 語彙数)

は最後の位置なので `[0, -1]` を取得

```
ers import AutoTokenizer, AutoModelForCausalLM
nizer.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")
delForCausalLM.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")

]かし、あるところに", return_tensors="pt").input_ids
grad():
odel(ids).logits[0, -1]

softmax(logits, dim=-1)
k(probs, 5)
```

□□□

が最大のトークンを選択

ンが出たら停止

`ck_next` が `argmax` に変わっただけ

```
reedy(prompt, max_new_tokens=40):
prompt, return_tensors="pt").input_ids
ange(max_new_tokens):
orch.no_grad():
gits = model(ids).logits[0, -1]
d = torch.argmax(logits)
t_id.item() == tok.eos_token_id:
eak
torch.cat([ids, next_id.view(1, 1)], dim=1)
.decode(ids[0])
```

[illegible]

ら、いつも同じ出力

い語だけを拾うので、単調になりやすい

布からランダムに選択

```
reedy("□きな食べ物は")
]食べ物は何ですか? □きな食べ物は...
```

も拾うので、分布を整えてから選択

□□□□□□□

てから softmax

割れば差が開き、大きい数で割れば縮小

絞れば、低い語を候補から除外

```
t(logits, temperature=0.8, k=50):
rch.softmax(logits / temperature, dim=-1)
n.topk(probs, k)
n.multinomial(top.values, 1)
.indices[idx]
```

`** (1/T)` していた。logits は確率の対数なので、割り算が同じ役割

□□□□□

順に足し、合計が p に達したら停止

ときは候補が少なく、分散しているときは多い

あたりが定番。`k=1` なら greedy と同じ

```
softmax(logits / temperature, dim=-1)
sorted_idx = torch.sort(probs, descending=True)
.cumsum(sorted_probs, dim=-1)
- sorted_probs < p
ed_idx[mask][torch.multinomial(sorted_probs[mask], 1)]
```

本編では温度と top-k で十分

[illegible]

、「指示 → 応答」の書式を保持

`at_template` が、その書式の文字列を生成

つもの生成ループへ

```
system", "content": "あなたは簡潔に答えるアシスタントです。"},
user",   "content": "日本の首都は？"},

pply_chat_template(
tokenize=False, add_generation_prompt=True)
```

```
。人口は約1400万人で...
```

□□□□□□□□□

ンで、役割の区間を区切る

りとりを覚えているわけではない

えている」ように見えるのは、そのため

```
stem
トです。<|im_end|>
er
|>
sistant
```

どの生成ループへ

⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠

なると遅くなり、入力上限も超過

もっともらしい嘘が頻繁

`odel.generate(...)` を使用

	対処
上限	古いやりとりを削る
	温度を上げる / 既出語の logits を下げる
	事実確認に使わない
計算	本来は KV キャッシュで使い回す

、はるかに効率よく実行

API

は、次の1トークンの確率分布を返す関数

測 → 選ぶ → つなげる、のループ

歴を毎回すべて渡すことで成立

きさと、実装の速さだけ

ormer □□□

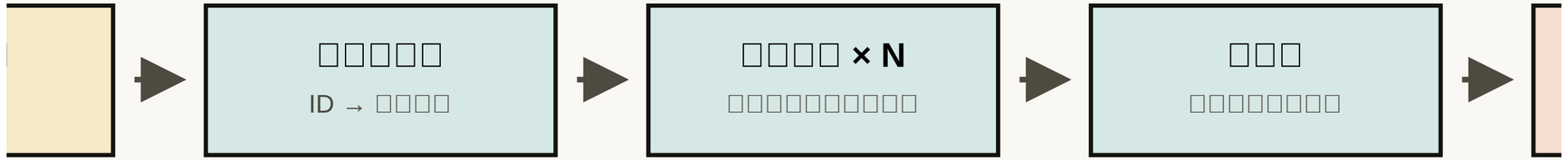
□□□□□□

]

ベクトルにし、注意機構で文脈を混合

サイズの logits を出力

学習の講義へ



くても、入出力だけ扱えれば生成ループは記述可能

☒☒☒☒☒☒☒☒☒

確率を目で確認

うと、温度と top-k を追加

ャットテンプレートで会話を生成

```
```

クンの上位10候補と確率を表示する

`te_greedy` を完成させ、同じ入力で結果が一致することを確かめる

top-k を実装し、温度 0.3 / 0.8 / 1.5 で比較する

テンプレートで3往復の会話を生成する

リング / 繰り返しペナルティ

□□□□□

開く」から課題を開き、「ドライブにコピーを保存」

レを実行し、各課題の `TODO` を埋める

消さない。最後に `score()` を実行してから提出