深層学習ベースの類似単語曖昧検索

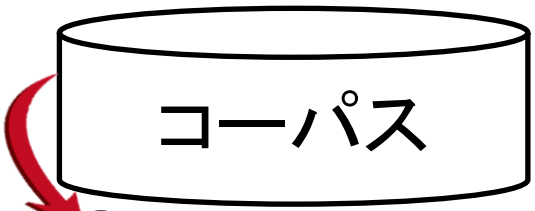
永田亮（甲南大学／理研）
川崎義史（東京大学）
内田諭（九州大学）

このパートでの内容

- **文（単語列）→単語ベクトルへの変換**
  - BERT：単語トークンベースのベクトル
  - 文脈付き単語ベクトル
- **単語ベクトルによる類似事例の検索**
  - 類似度：ベクトル間の類似度（余弦類似度）

プログラムの概要（仕様）

曖昧用例検索(再掲)



①対象単語の用例抽出

I have a bank account in Japan
They were sitting on the bank.
:

検索対象単語

He drew some bucks from the bank.

LM

LM

②文脈付きベクトル化

③文脈付きベクトル化

$$\begin{pmatrix}0.5\\:\\0.2\end{pmatrix}\begin{pmatrix}2.5\\:\\1.3\end{pmatrix}\cdots\begin{pmatrix}0.7\\:\\0.3\end{pmatrix}$$

④類似度計算

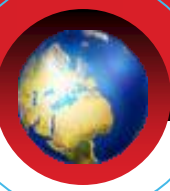
⑤高類似事例出力

$$\begin{pmatrix}0.6\\:\\0.3\end{pmatrix}$$

文(単語列)を単語ベクトルに変換

- ファイル読み込み(午前と同じ)
- 単語分割(トークン分割;トークナイザ)
  - BERT専用分割器
- 単語をIDへ変換
  - BERTに入力するための形式へ変換
- 単語ベクトルへ変換
  - ID→one-hot-ベクトル→単語ベクトル
- 出力:単語と単語ベクトルを画面に出力

単語ベクトルによる類似用例検索



- コーパスファイル読み込みとベクトル化
  - これまでの処理！
- 入力文中の対象単語のベクトル化
  - ここも上と同じ
- 対象単語とコーパス中の全単語の類似度計算
  - ベクトル間の類似度（＝余弦類似度）
- 類似度が高い順に並べ替え
  - ソート機能を使用
- 出力：上位5件を画面に出力

プログラム解説

LM（BERTの準備）

5_search_similar_word_usage.ipynb（前半）

```
!pip install transformers
import torch
import transformers
from transformers import AutoTokenizer
from transformers import BertModel

model_name = 'bert-base-cased' # 'bert-base-uncased' to ignore difference in upper/lower cases
bert_model = BertModel.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu') # CPU mode or GPU
bert_model.to(device)
bert_model.eval()
print('Working on:', device)
```

単語分割，ID化，単語ベクトル変換

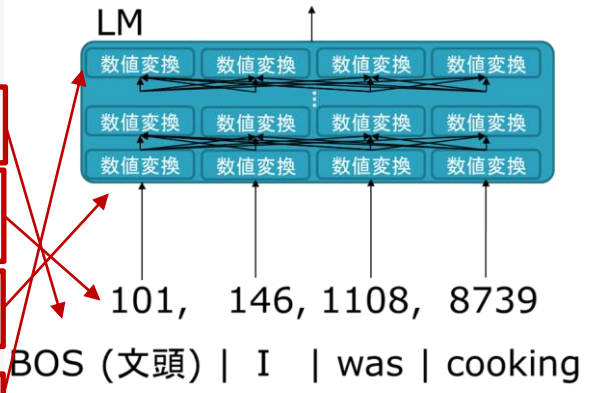
5_search_similar_word_usage.ipynb（中盤）

対象文中の全単語をベクトル化

```
import copy
vectors_with_indices = []
print('Device:', device)
for sentence in sentences_in_corpus:
  # token to ids
  tokenized = tokenizer(sentence,
                        return_tensors='pt', padding=True, truncation=True)
  token_ids = tokenized['input_ids'].to(device)
  mask_ids = tokenized['attention_mask'].to(device)

  bert_output = bert_model(token_ids, mask_ids)

  tokens = tokenizer.tokenize(sentence)
  last_hidden_state = bert_output.last_hidden_state[0]
  # 0番目は文頭トークン[CLS]のため除外．1番目からスタート
  for token_index in range(1, len(last_hidden_state)):
    token_vector  = last_hidden_state[token_index]
    token_vector = token_vector.to('cpu').detach().numpy().copy()
    vectors_with_indices.append([tokens,          #分割結果
                                 token_index-1, #単語番号（何番目の単語か）
                                 token_vector]) #単語ベクトル
```



検索対象語の位置(スパン)を特定

5_search_similar_word_usage.ipynb(中盤)

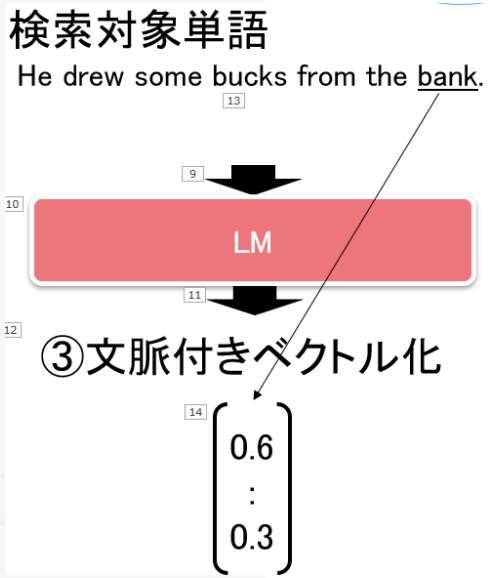
検索対象語の文中の位置を見つける処理

```
# 次の三行（3-5）は対象フレーズを見つけるためのプログラムの準備
# util.pyの中に具体的な処理が書かれている
import sys
sys.path.append('drive/My Drive/nlpseminar2024')
import util

#target_sentence = 'There is a large river bank.'
target_sentence = 'I drew some bucks from the bank.'
target_word = 'bank'  # 分析対象としてbankを指定

print('Target word:', target_word)
tokens_of_target_word =  tokenizer.tokenize(target_word)

target_words = tokenizer.tokenize(target_sentence)
print(target_words)
# (何番目の単語から，何番目の単語までか)
target_span = util.find_target_spans(target_words, tokens_of_target_word)
print(target_span, target_sentence)
```



検索語を単語ベクトルへ変換

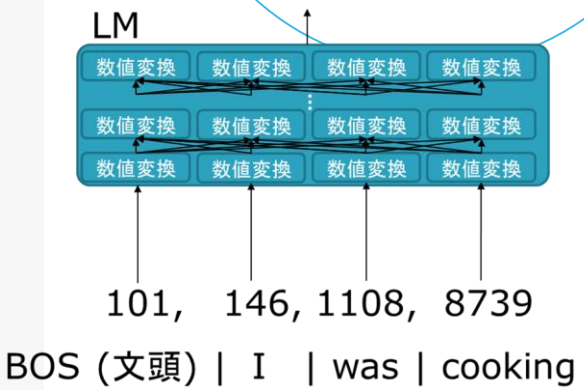
5_search_similar_word_usage.ipynb（中盤）

```
tokenized = tokenizer(target_sentence,
                      return_tensors='pt', padding=True, truncation=True)
token_ids = tokenized['input_ids'].to(device)
mask_ids = tokenized['attention_mask'].to(device)

# BERTに入力しベクトルに変換
bert_output = bert_model(token_ids, mask_ids)

# BERTの出力は文頭に特殊トークン[CLS]（文頭トークン）を含むため
# indexが一つずれることに注意（begin_index + 1）
begin_index, end_index = target_span[0]
target_vector = bert_output.last_hidden_state[0, begin_index+1:end_index+1][0]

# CPUで扱えるベクトルに型変換
target_vector = target_vector.to('cpu').detach().numpy().copy()
print(target_vector[0:10]) # ベクトルの0～9番目の値を表示
```



演習：プログラムを図に対応付けてみよう

検索語と全単語間の類似度を計算

5_search_similar_word_usage.ipynb（終盤）

```
import numpy as np  # ベクトル，行列計算モジュール
similarities = []
norm_of_target_vec = np.linalg.norm(target_vector)
for sentence_index, token_index, token_vector in vectors_with_indices:
  product =  np.dot(token_vector, target_vector) # 二つのベクトルの内積
  norm = np.linalg.norm(token_vector) # 類似度計算対象の単語のベクトルのノルム
  cos_sim =  product/(norm_of_target_vec*norm)   # 余弦類似度
  similarities.append((cos_sim, sentence_index, token_index))

sorted_sims = sorted(similarities, key=lambda x: x[0], reverse=True)
```

余弦類似度

$$\frac{v_1 \text{と} v_2 \text{の内積}}{v_1 \text{のノルム} \times v_2 \text{のノルム}}$$

結果の出力

5_search_similar_word_usage.ipynb（終盤）

結果の出力（top_n件）

```
print('------')
print('Target word:', target_word)
print('Target sentence:', target_sentence)
print('------')

top_n = 5 # 上位5件
tag = '*' # 検索された語をわかりやすく表示するためのタグ
for cos_sim, tokens, token_index in sorted_sims[:top_n]:
  output_tokens = tokens[:]
  output_tokens[token_index] = tag + output_tokens[token_index] + tag
  sentence_for_output = ' '.join(output_tokens)

  print(cos_sim, sentence_for_output)
```

演習：検索対象を変えてみよう

- 検索語を含む文：target_sentence
  - 例：There is a large river bank in the town.
- 検索語：target_word
  - 例：river

まとめと発展的内容

まとめ：LMと単語ベクトルを利用

- 曖昧検索を実現
  - トークン分割
  - ID化
  - ベクトル化
  - 類似検索
- わかりやすさを重視
  - 実行速度や使用メモリは度外視
  - それでも様々なことに応用可能

    例：用例グルーピング，可視化

まとめ：LMと単語ベクトルを利用

- 曖昧検索を実現
  - トークン分割
  - ID化
  - ベクトル化
  - 類似検索
- わかりやすさを重視
  - 実行速度や使用メモリは度外視
  - それでも様々なことに応用可能

    例：用例グルーピング，可視化

発展的内容

- 様々な項目に対する類似度
  - 単語，フレーズ
  - 文
  - パラグラフ
  - 文書
- 様々な言語現象
  - 意味用法の変化の検出
  - 文法化度の定量化

発展的内容

- 今回扱わなかった（避けた）項目
  - ミニバッチ化（実行速度向上）
  - サブワードの取り扱い（未知語の処理）