

2025年度 データサイエンティスト育成講座 第12回

単語埋め込み

~単語の意味をニューラルネットワークで表現 単語の類似性を計算する~

立命館大学 情報理工学研究科

データ工学研究室

M1 和田 仁聖

自然言語処理の復習

自然言語とは

- 人間が話したり書いたり **日常的に使っている言葉**のこと

自然言語処理とは

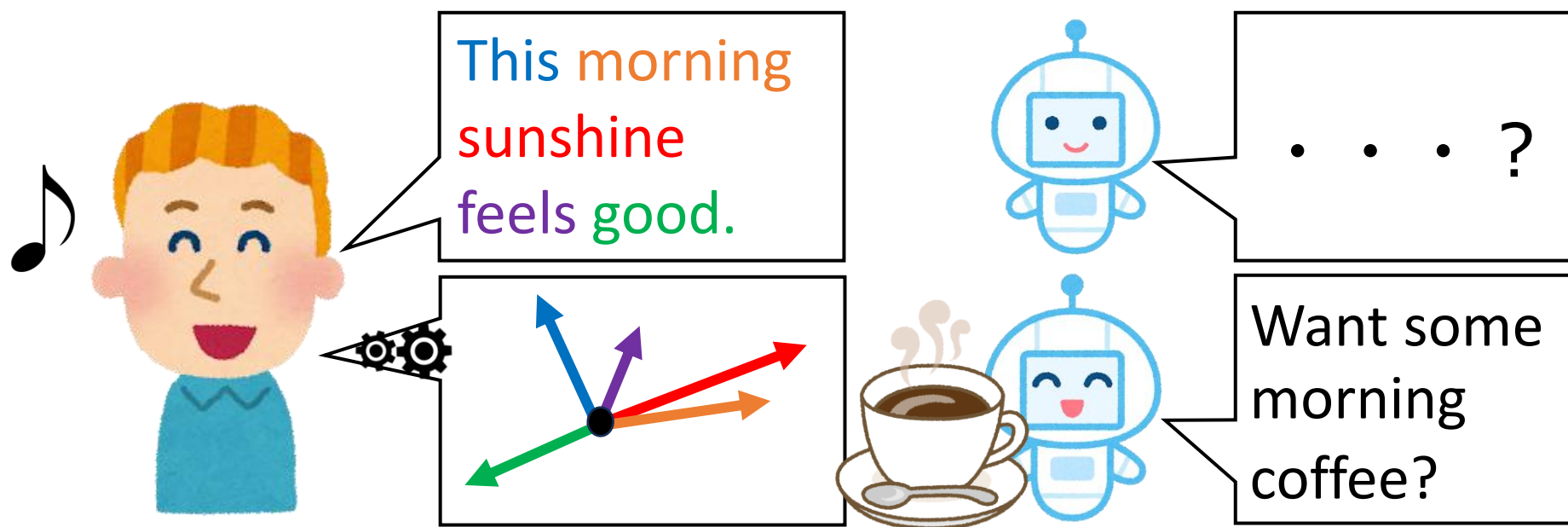
- 自然言語の **意味を解析し処理**する技術
- 形態素解析→構文解析→意味解析→文脈解析の順番で進む。

★今日の目的

コンピュータに人間の扱う単語の **意味**を理解してほしい

単語埋め込み(Word Embedding)

- 単語をコンピュータが理解しやすい数値ベクトルに変換する方法



単語同士の類似性を距離や角度から表現できる

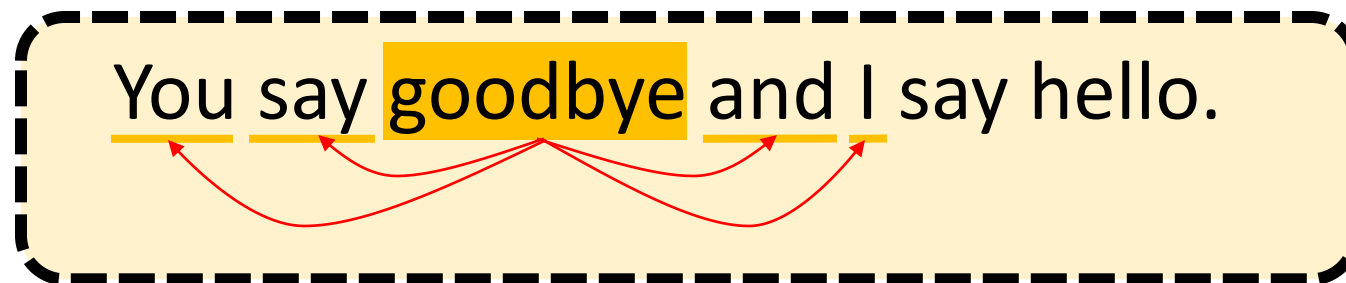
単語の分布仮説(distributional hypothesis)

- 意味が似ている単語は、似たような文脈で使われる

「I **drink** **beer**.」 「We **drink** **wine**.」 -> drinkの近くには飲み物が表れやすい

「I **guzzle** **beer**.」 「We **guzzle** **wine**.」 -> 多分guzzleはdrinkと近い意味

重要なのは**コンテキスト** (ある単語の周囲の単語)

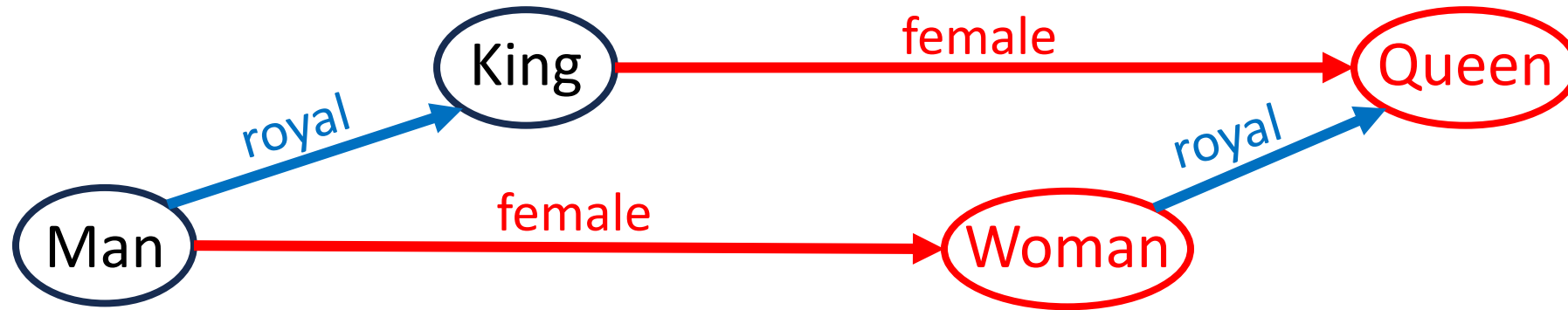


※ウィンドウサイズが2の場合、2つ隣まで見る

単語の分散表現(distributed representation)

臙脂色を出して -> わからない [185, 64, 71]を出して -> **こちらですね**
つまり...

- 単語を固定長のベクトルで表現



King - Man + Woman = Queen のような式が成り立つ

コンピュータが単語の類似性を理解できる

単語の局所表現 (local representation)

- 単語ごとに独立のベクトルを作る

One-Hotベクトル : 1単語につきベクトルの1要素のみ1

単語	One-Hotベクトル
Apple	[1, 0, 0]
Human	[0, 1, 0]
Player	[0, 0, 1]

問題点：

1. 単語の意味が反映できない
(単語は互いに独立)
2. 語彙が多いほどベクトルの次元が大きくなる

コンピュータに単語の意味を理解させる手法②

- カウントベースの手法

コーパス：目的をもって収集された大量の学習用テキストデータ

共起行列：コンテキストをカウントする

「You say goodbye and I say hello.」 (ウィンドウサイズ1の場合)

	You	say	goodbye	and	I	hello	.
You	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
I	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

- **Bag-of-Words** : 文書中に登場する単語の種類と出現回数だけを数えて文書を数値ベクトルで表現

文書1 : 「I love cats」

文書2 : 「Cats love milk」

文書	I	love	cats	milk
文書1	1	1	1	0
文書2	0	1	1	1

⇒ 語彙数次元のベクトル空間に各文書のベクトルを置ける

- **TF-IDF** : 単語の出現頻度 (TF) と単語の重要度 (IDF) を組み合わせて単語の重みを計算する

$$TF - IDF = \frac{\text{ある単語の文章内での出現回数}}{\text{文章内の全単語の出現回数}} \times \log \left(\frac{\text{総文書数}}{\text{ある単語が出現する文書数}} \right)$$

⇒ 「その文章で頻繁に出る + 全体では珍しい」 単語に大きなスコアがつく

カウントベースの手法 -> N-gram

- 連続するN個の単語のまとまりの並びを考慮することで、文章の局所的な文脈を捉える
 - 単語N-gram : 単語で分割する (英語だとこっちがやりやすい)
 - 文字N-gram : 文字で分割する (日本語だとこっちがやりやすい)

例) 今日はいい天気だ

分割数	単語N-gram	文字N-gram
Uni-gram(N=1)	今日/は/良い/天気/だ	今/日/は/良/い/天/気/だ
Bi-gram(N=2)	今日は/は良い/良い天気/天気だ	今日/日は/は良/良い/い天/天気/気だ
Tri-gram(N=3)	今日は良い/は良い天気/良い天気だ	今日は/日は良/は良い/良い天/い天気/天気だ

⇒ 単語の意味の違いを表現することができる

カウントベースの手法 まとめ

- 「単語の数や共起」を数えることで、テキストの特徴を数値化
- ベクトルの性質としてスパース（疎）になる

メリット

- 使うのが簡単
- シンプルな計算なので早い
- 文書分類（スパム判定）などに使える

デメリット

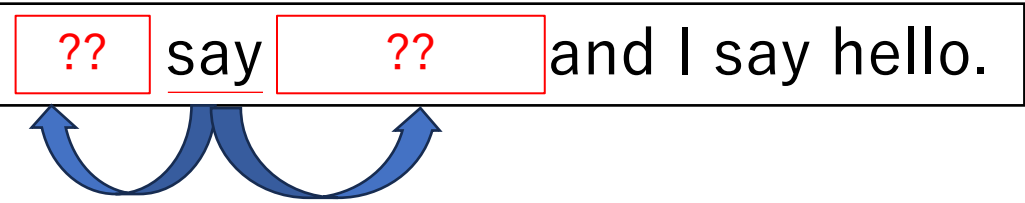
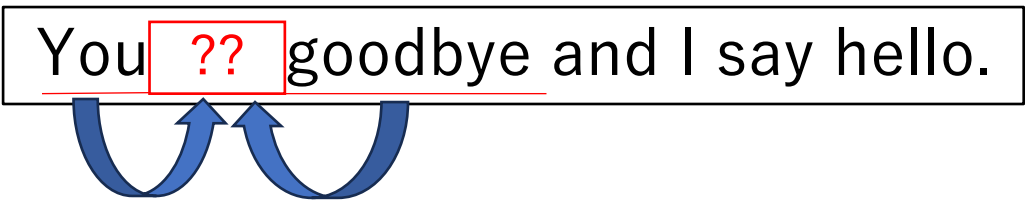
- 類義語・多義語の区別は難しい
- 語彙の数だけ次元が増えるので高次元になりやすい
- 長距離の文脈が分からない

コンピュータに単語の意味を理解させる手法③

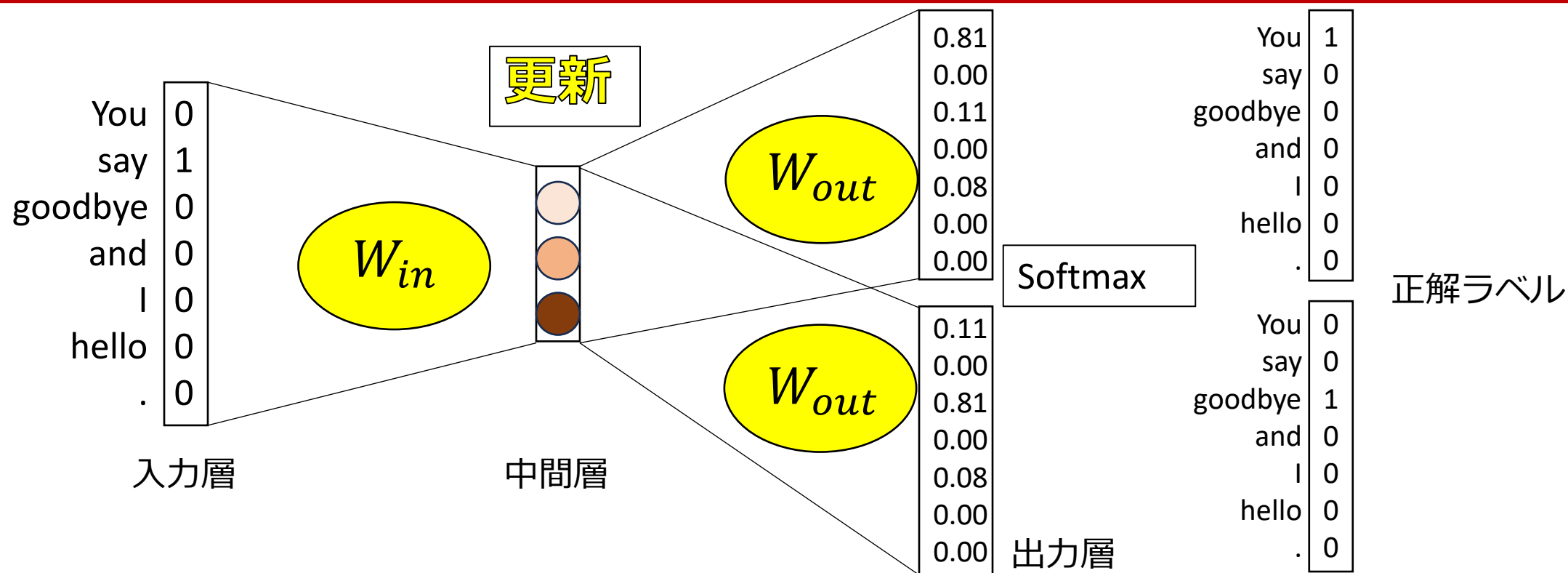
- 推論ベースの手法

Word2Vec：単語の出現パターンを予測するタスクを解きながら、その過程で単語のベクトルを学習する

学習に用いるニューラルネットワークには2種類ある

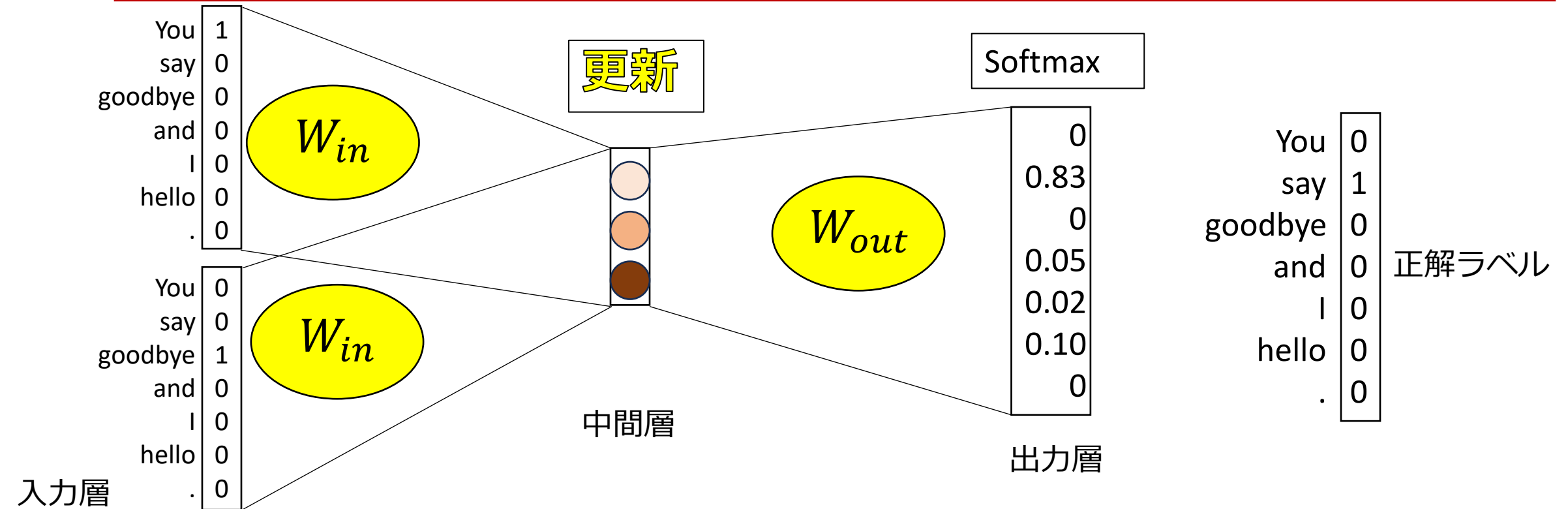
Skip-gram	CBow
注目している単語から周辺の単語を予測	周辺の単語から注目している単語を予測
<div>?? say ?? and I say hello.</div> 	<div>You ?? goodbye and I say hello.</div> 
<ul style="list-style-type: none">○ 低頻出の単語でも学習できる× 長い距離の依存関係の考慮なし	<ul style="list-style-type: none">○ 計算効率が高い× Skip-gramに比べて精度低

Skip-gramモデルの例



1. 語彙をOne-hotベクトルに変換する
2. 中心単語の周辺に出現しやすい単語との内積を大きくするように学習
3. 出力された周辺語のOne-hotベクトルと比較し、Skip-gramにfeedbackして最適化

CBoWモデルの例



1. 語彙をOne-hotベクトルに変換する
2. 周辺単語の中心に出現しやすい単語との内積を大きくするように学習
3. 出力された周辺語のOne-hotベクトルと比較し、CBoWにfeedbackして最適化

ベクトルの距離・類似度

- ユークリッド距離(Euclidean Distance)

$$d(\mathbf{u}, \mathbf{v}) = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 + \dots + (u_n - v_n)^2}$$

距離が遠い \Rightarrow 単語が似ていない

距離が近い \Rightarrow 単語が似ている

- コサイン類似度(Cosine Similarity)

$$\cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| * \|\mathbf{v}\|}$$

ベクトルの角度が 0°

$\Rightarrow 1$ (とても似ている)

ベクトルの角度が 90°

$\Rightarrow 0$ (全くの無関係)

ベクトルの角度が 180°

$\Rightarrow -1$ (反対の意味)

推論ベースの手法 まとめ

- 単語の出現関係を予測して単語の意味を学習する
- 自動的に分散表現（意味が内包された密なベクトル）を獲得

メリット

- 単語の意味をベクトルからとらえられる

デメリット

- 抽象度が高く学習にコストがかかる

Word2Vec以外の推論ベースの手法

- FastText : subword(N-gram) + Word2Vec で作るモデル、未知語にも強い
- ELMo : 双方向LSTMの文脈予測モデル、同じ単語でも文脈で使うベクトルを変える
- BERT : Transformerベースの双方向予測モデル、深い文脈理解が可能
- GPT系 : 次単語を順番に1つずつ予測する、自然な文章生成に特化

学習したモデルの精度の評価

- 内省的評価（主観的）
 - 単語ベクトル自体の「意味的な質」を直接測る方法
 - 単語間の類似度や、足し引きの結果を見て判定
- 外省的评价（客観的）
 - 分類問題など実際のタスクを単語分散表現により解くことでタスクの精度を評価する
 - かなりの量をテストしないといけなないので時間がかかる

Word2Vecなどの実装

Word2Vecの作成とパラメータ

```
from gensim.models import Word2Vec
```

```
model = Word2Vec(  
    sentences=sentences,  
    vector_size=100, . . .  
    window=5, . . . . .  
    min_count=1, . . . . .  
    sg=1, . . . . .  
    hs=0, . . . . .  
    negative=5, . . . . .  
    seed=42 . . . . .  
)
```

パラメータの内容	デフォルト
トークン化済みのリスト	必須
単語ベクトルの次元数	100
コンテキストの前後単語数	5
出現回数がこの値未満の単語を無視	5
0: CBOW（高速） / 1: Skip-gram（精度高）	0
1:階層的ソフトマックス（少語彙向け）	0
ネガティブサンプリング数（0で無効）	5
シード値	None

メソッド一覧

1. 単語ベクトルの取得

```
X = model.wv['単語']
```

2. 類似単語の取得

```
model.wv.most_similar('単語', topn=10)
```

3. 単語の類似度計算

```
model.wv.similarity('単語1', '単語2')
```

4. ベクトル計算

```
model.wv.most_similar(positive=['単語1', ...],  
                      negative=['単語2', ...])
```

5. 語彙に単語が含まれるか確認

```
'単語' in model.wv
```

6. 学習済みモデルの保存

```
model.save("file.model")
```

7. 学習済みモデルの読み込み

```
Word2Vec.load("file.model")
```

8. 語彙一覧の取得

```
model.wv.index_to_key[:10]
```

TF-IDFの実装

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

インスタンスの生成

```
vectorizer = TfidfVectorizer()
```

TF-IDFのベクトル化

```
tfidf_matrix = vectorizer.fit_transform(データ)
```

重要単語の抽出

```
feature_names = vectorizer.get_feature_names_out()
```

Question1

40分

- Word2Vecを用いて単語の分散表現を得る

使用するデータ

Livedoorニュースコーパス(ITライフハック)

Word2Vecを使って単語埋め込みについて理解してください
また、自然言語処理全般における前処理の重要性をこの演習
で理解してください

Question2

30分 + 30分

- Contextual Inquiry法にWord2Vecを用いる

使用するデータ

社会人講座第3回で作成したシナリオ

自由度が高く、負荷の高い考察になるので一度中断します

