

クレジット:

Mathematics and Informatics Center メディアプログラミング入門 2020 山肩洋子

ライセンス:

利用者は、本講義資料を、教育的な目的に限ってページ単位で利用することができます。特に記載のない限り、本講義資料はページ単位でクリエイティブ・コモンズ 表示-非営利-改変禁止 ライセンスの下に提供されています。

<http://creativecommons.org/licenses/by-nc-nd/4.0/>

本講義資料内には、東京大学が第三者より許諾を得て利用している画像等や、各種ライセンスによって提供されている画像等が含まれています。個々の画像等を本講義資料から切り離して利用することはできません。個々の画像等の利用については、それぞれの権利者の定めるところに従ってください。



メディアプログラミング入門

第4回：トピック分析とWord Embedding

火 5 @本郷 2020年6月23日

情報理工学系研究科 数理・情報教育研究センター

准教授 山肩 洋子

第3回の課題：

夏目漱石の「吾輩は猫である」っぽい文を生成してみよう！

課題 1：分かち書きの生成

- `text/wagahaiwa_nekodearu_org.txt`には、夏目漱石の小説「吾輩は猫である」の本文が記録されている
- このテキストを分かち書き文に変換して、`text/wagahaiwa_nekodearu_wakati.txt`に保存
- 出力ファイルの最初の1行を出力し、その状態で保存して提出

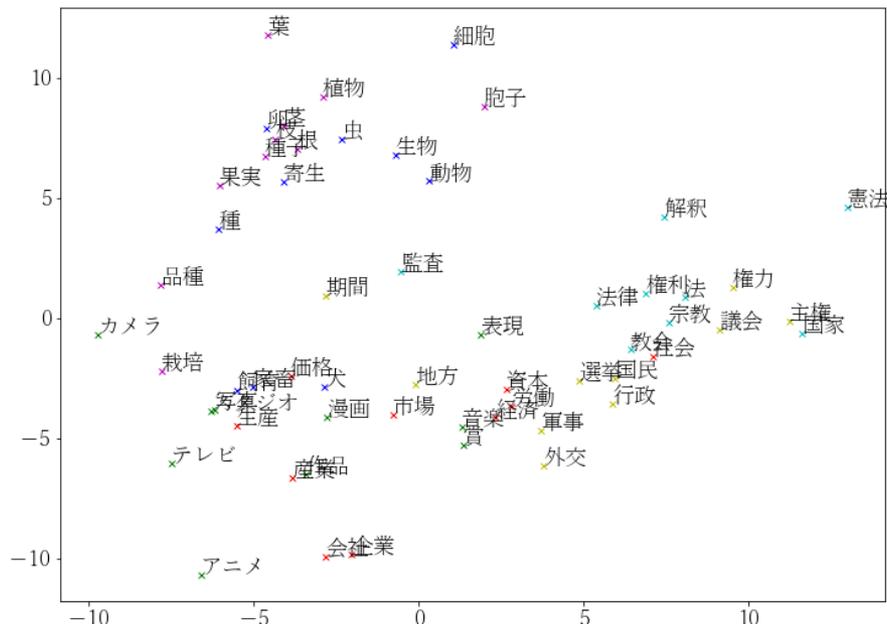
課題 2：統計的単語tri-gramによるランダム文生成

- 課題 1 で作成した`text/wagahaiwa_nekodearu_wakati.txt`を入力として、統計的tri-gramモデルを学習してください
- 学習モデルの変数名は`lm`とします
- また、2つ下のセルで、そのモデルを使って夏目漱石の小説「吾輩は猫である」っぽい文をランダムに生成してください
 - 生成文は「吾輩は」から始めて、「。」あるいは「」で終わる一文です
 - 生成した文が出力した状態でこのファイルを保存して、提出してください

第4回 テキスト解析2：自分の文章の特徴を調べよう

講義内容： Pythonにおける文書の取り扱いや解析、検索方法について学習する

演習内容： 自然言語処理ライブラリgensimを用いて、特徴語辞書作成やtf-idf、トピック分析 (LDA)、word2vecを試行する



Wikipediaの記事から重要語を抽出し、タグクラウドを生成（上図はWikipediaカテゴリ「動物」のタグクラウド）

- Newral networkの意味解析手法であるWord2vecによる単語マッピング
- 語彙をベクトル表現に変換すれば、機械学習に利用できる

本日の学習内容

- テキストのベクトル空間モデル
 - 単語のベクトル表現
 - 文書のベクトル表現
- トピック分析
 - 教師あり文書分類：tf-idf値による重要語抽出
 - 演習) Wikipedia記事のトピック分析
- Word Embedding
 - ニューラルネットワークによる単語の分散表現生成：Word2Vec
 - より進んだ手法：BERT
 - 演習) gensimによるWord2vecの学習と可視化

テキストのベクトル空間モデル

TopicAnalysis1.ipynb

テキストを機械学習で扱うには？

例えば...

- ある会社の株価が上がるか下がるかは、その会社のTwitterアカウントの文書が影響するかも！？
→ 株価の予測モデルにTwitterの情報も組み込みたい！
- 一般的な機械学習は数値の偏りをモデル化する
→ テキストを数値に変換する必要がある

例) 2019/4/10 @UTokyo_News

Tweet0 = “【イベント】GO GLOBAL 東大留学フェア2019、4月20日@駒場キャンパスにて開催！”



どうやって変換しましょう？

ベクトル表現に変換すれば学習できる

Tweet0 = [0.53, 0.45, 0.36, 0.26, 0.436, 0.463, ...]

何はともあれ単語分割

例) 2019/4/10 @UTokyo_News

Tweet0 = “【イベント】GO GLOBAL 東大留学フェア2019、4月20日@駒場キャンパスにて開催！”



Janomeなどの形態素解析器

```
Tweet0_words = ['【', 'イベント', '】', 'GO GLOBAL', ' ', '東大', '留学フェア', '2019', '、', ' ', '4', '月', '20', '日', '@', '駒場キャンパス', 'にて', '開催', '!']
```



文書は単語の集合とみなす : Bag-Of-Words



- 語順は考慮しない（しなくてもだいたいわかる）
 - 「イベント」→イベント関係の記事？
 - 「開催」→開催案内らしい...
- その文書に特定の単語が登場するかしないか？

辞書と単語のベクトル表現

語彙 (Vocabulary) : ある文書集合に現れるすべての単語

- 前回の演習で、宮沢賢治の文書集合の語彙サイズは22,793語
- 語彙にはIDが付いている→これを使っていい？ダメ！

語彙リスト

ID	表記
...	...
7727	カブ
7728	カマ
7729	カムチャッカ
7730	カムチャツカ
7731	カムパネルラ
7732	カメレオン
7733	カラ
7734	カラア
7735	カラカラ
...	...

「カムパネルラ」よりも「カムチャツカ」のほうが大きい？

「カメレオン」と「カムチャツカ」の平均は「カムパネルラ」！？

- IDは**名義尺度**であって大小関係を意味しない！
- しかし機械学習では大小関係を学習してしまう



one-hotベクトル

one-hotベクトルの使いどころ

アンケートの回帰分析

例) 所属学部から得意科目を予測しよう

好きな教科は何でしたか？
数字で教えてください。

3

科目ID: 1. 算数, 2. 国語, 3. 理科,
4. 社会, 5. 英語,...

	好きな科目	所属学部
東大花子	3	工学部
本郷太郎	1	文学部
駒場次郎	2	理学部

学習用データ :

工学部→3

文学部→1

理学部→2

で回帰モデルを学習すればいい？

ダメ！
IDは数値
じゃない！

one-hotベクトルの使いどころ

アンケートの回帰分析

例) 所属学部から得意科目を予測しよう

	好きな科目	所属学部
東大花子	3	工学部
本郷太郎	1	文学部
駒場次郎	2	理学部

学習用データ :

[算, 国, 理, 社, 英, ...]

[0, 0, 1, 0, 0, ...] → 工学部

[1, 0, 0, 0, 0, ...] → 文学部

[0, 1, 0, 0, 0, ...] → 理学部

というようにone-hotベクトルにするべき!

- ベクトルの加算
= その教科が好きな学生の数
- 「算数と国語が好きな人」
→ [1, 1, 0, 0, 0, ...]

データを表す様々な数値

- **名義尺度**

- 数値を名前ととらえたもの
- 例) ユーザID、郵便番号、性別、出身地

- **順序尺度**

- 順序を表す数値。大小はあるが、差や比に意味はない
- 例) 競争の順位、地震の震度、好き・普通・嫌い

- **間隔尺度**

- メモリが等間隔で、差に意味がある
- 例) 日付、温度（摂氏、華氏）、テストの点数

- **比例尺度**

- **原点があり、四則演算が使える**
- 例) 重さ、長さ、絶対温度、身長、体重、速度

機械学習では基本的にすべて**比例尺度**に変換する必要がある
特に**名義尺度は必須**！

one-hotベクトル

- ベクトルの要素数 = 語彙サイズ (22,793次元のベクトル)
- ある単語のベクトル表現は、その単語に該当するアドレスの値だけが1となり、残りが0

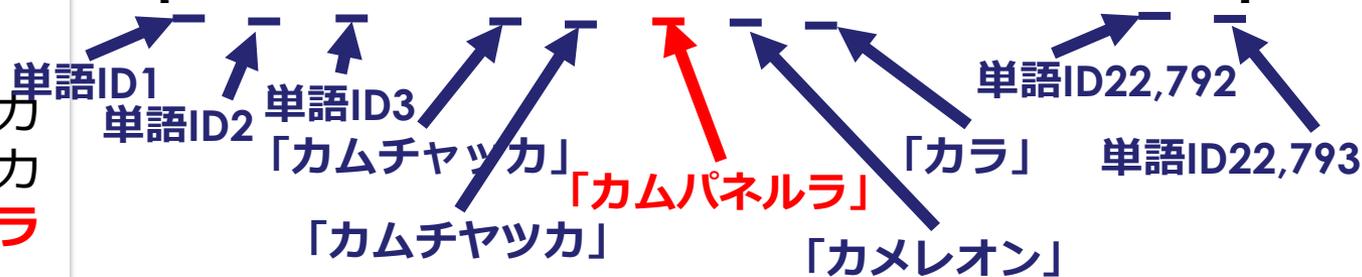
語彙リスト

ID	表記
...	...
7727	カブ
7728	カマ
7729	カムチャツカ
7730	カムチャツカ
7731	カムパネルラ
7732	カメレオン
7733	カラ
7734	カラア
7735	カラカラ
...	...

例) 「カムパネルラ」のone-hotベクトル

Word_vec['カムパネルラ']

= [0, 0, 0, ..., 0, 0, 1, 0, 0,, 0, 0]



22,793次元のベクトルのうち、「カムパネルラ」に対応する7731番目の要素だけが1で、それ以外はすべて0

ものすごくスパース (疎) !

文書のベクトル表現と類似度計算

目的：ある文書 d_A と d_B がどの程度似ているかを評価したい

- d_A と d_B 以外にもたくさんの文書が与えられているとする。
その文書集合 D における語彙を $V=[v_1, v_2, \dots, v_{|V|}]$ とする
- 単語 v_i の one-hot ベクトルの次元は $|V|$ (=語彙数)
- 文書 d の特徴ベクトル $d=[d_1, d_2, d_3, \dots, d_{|V|}]$
 - 次元は $|V|$
 - ある要素 d_j は単語 $v_i \in V$ のその文書における重要度
 - 重要度とは？
 - その文書における単語 v_i の出現回数
→つまり d は、その文書に含まれるすべての単語の one-hot ベクトルの合計
 - TF-IDF や LDA で求めた値とすることもできる
(後で出てきます)

演習：短い文書間の類似度を計算してみよう

文書 (8種類)

'私は本を読む。'
 '今日は晴天だ。'
 '私は私だ。'
 '本は本で本だ。'
 '私は本を読む。'
 '私本を読む。'
 '私は本を読む。本を私は読む。'
 '明日雨が降る！'



語彙リスト

ID	単語	ID	単語	ID	単語
0	。	5	を	10	私
1	が	6	今日	11	読む
2	だ	7	明日	12	降る
3	で	8	晴天	13	雨
4	は	9	本	14	！

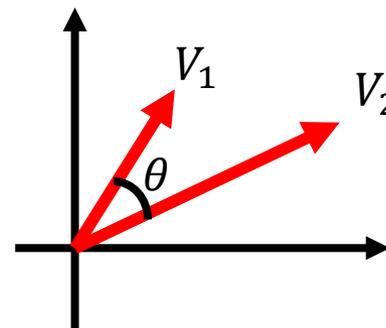
文書	。	が	だ	で	は	を	今日	明日	晴天	本	私	読む	降る	雨	！
「私は本を読む。」	1	0	0	0	1	1	0	0	0	1	1	1	0	0	0
「今日は晴天だ。」	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0
「私は私だ。」	1	0	1	0	1	0	0	0	0	0	2	0	0	0	0
「本は本で本だ。」	1	0	1	1	1	0	0	0	0	3	0	0	0	0	0
「私は本を読む。」	1	0	0	0	1	1	0	0	0	1	1	1	0	0	0
「私本を読む。」	1	0	0	0	0	1	0	0	0	1	1	1	0	0	0
「私は本を読む。 本を私は読む。」	2	0	0	0	2	2	0	0	0	2	2	2	0	0	0
「明日雨が降る！」	0	1	0	0	0	0	0	1	0	0	0	0	1	1	1

文書間類似度 = コサイン類似度 (cosine similarity)

- 文書間の類似度は、この文書ベクトル同士がどれだけ似通っているかで判断することができる
- ベクトル間の類似度の尺度はいろいろありますが、中でもよくつかわれるのが**コサイン類似度**

ベクトル V_1 とベクトル V_2 の類似度:

$$\text{cos_sim}(V_1 V_2) = \cos \theta = \frac{V_1 \cdot V_2}{|V_1| |V_2|}$$



ベクトルの大きさ (=文書の長さ) は関係ない!

「私は本を読む。」の文書ベクトルは $[1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0]$
「私は私だ。」の文書ベクトルは $[1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0]$

$$V_1 \cdot V_2 = 1 \times 1 + 0 \times 0 + 0 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 0 + 1 \times 2 + 1 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 = 4$$

$$|V_1| = \sqrt{1+1+1+1+1+1} = \sqrt{6}, \quad |V_2| = \sqrt{1+1+1+2**2} = \sqrt{7},$$

コサイン類似度計算結果

「は」と「。」しか一致しない

「私は本を読む。」と「今日は晴天だ。」の類似度: 0.365

完全に一致

「私は本を読む。」と「私は本を読む。」の類似度: 1.0

1単語だけ違うだけ

「私は本を読む。」と「私本を読む。」の類似度: 0.913

完全に一致しない

「私は本を読む。」と「明日雨が降る！」の類似度: 0.0

長さが倍で構成要素の比率は変わらない

「私は本を読む。」と「私は本を読む。本を私は読む。」
の類似度: 1.0

小説間の類似度

宮沢賢治の「銀河鉄道の夜」「風の又三郎」と太宰治の「人間失格」の3作品について、お互いがどの程度近いかを文書ベクトルのコサイン類似度で評価してみよう

語彙サイズ: 7721 → **文書ベクトルの次元は7721**

	銀河鉄道の夜	風の又三郎	人間失格
銀河鉄道の夜	1.00	0.94	0.88
風の又三郎	0.94	1.00	0.82
人間失格	0.88	0.82	1.00

- 「銀河鉄道の夜」は「人間失格」よりも「風の又三郎」のほうが似ている
- 「人間失格」は「風の又三郎」より「銀河鉄道の夜」のほうが似ている

トピック分析 tf-idfによる重要語抽出

演習 : TopicAnalysis2.ipynb

何が違う？重要な単語とは何？

名詞の頻出語

tf-idf法の重要語

法



動物



**どんな文書にも表れる単語は、
たとえその文書にたくさん現れていても
重要じゃない！**

重要な語とは何か？

- 先ほどの文書ベクトルは単語の出現回数を数えていた
- その文書にたくさん登場するからと言って、その単語が重要とは限らない！
 - 「の」はどんな文書でも最頻出単語
 - 「あなた」「私」などもどの小説にもたいてい現れる
 - 「カンパネラ」といえば・・・『銀河鉄道の夜』！

つまり文書において重要な単語とは...

- **その文書によくあらわれる**=これまでとほぼ同じ
 - **Tf (Term Frequency : 語の出現頻度)**
- **その単語が現れる文書が少ないこと！**
 - **iDf (Inverse Document Frequency: 逆文書頻度)**

tf-idf値

ある単語 $word_i$ のある文書 doc_j におけるtf-idf値を $tf-idf(word_i, doc_j)$ とすると、

$$\begin{aligned} &tf-idf(word_i, doc_j) \\ &= tf(word_i, doc_j) \cdot idf(word_i) \end{aligned}$$

Tf : 頻繁に表れる語のほど重要

$$tf(word_i, doc_j) = \frac{doc_j \text{ に } word_i \text{ が現れる回数}}{doc_j \text{ の総単語数}}$$

Idf : 他の文書には現れず、
その文書によくあらわれる語ほど重要

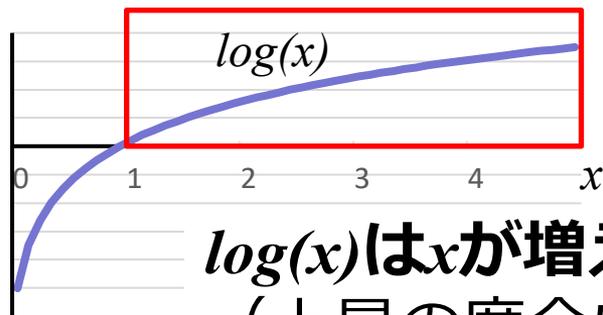
$$idf(word_i) = \log \frac{\text{文書の総数}}{word_i \text{ が現れる文書の総数}}$$

idfをもう少し詳しく

idf : 他の文書には現れず、
その文書によくあらわれる語ほど重要

$$\text{idf}(\text{word}_i) = \log \frac{\text{文書の総数}}{\text{word}_i \text{が現れる文書の総数}}$$

必ず1より大きい



$\log(x)$ は x が増えるにつれてなだらかに上昇
(上昇の度合いは \log の底によって調整可能)

つまりidfとは...

- その単語がすべての文書に現れるとき0=全く重要じゃない
- 他の文書に現れない単語ほど値が大きくなる

演習 : TopicAnalysis2.ipynb: tf-idfによる重要語抽出

分析対象 : Wikipediaの記事 (wikipedia_wakati.json)

- 6種類のカテゴリ [動物 (animal)、芸術 (art)、経済 (economy)、法 (law)、植物 (plant)、政治 (politics)]
- 分かち書きしてjsonファイルにまとめたもの

目的 : 各カテゴリに属する記事の集合を、それぞれ1つの文書とみなし、各文書 (=カテゴリ) の重要語を抽出

- Tf-idf値の高い単語を選べばよい
- 機械学習用モジュールscikit-learnのsklearn.feature_extraction.textモジュールのTfidfVectorizerという関数を使って計算

次元の呪い (Curse of dimensionality)

- **テキスト処理で扱うデータはスパースになりがち**
 - ほとんど1回しか現れないような単語は分析の役に立たない
 - 書き間違いもある→役に立たない次元が大量発生する！
- (数学的) 空間の次元が増えるのに対応して問題の算法が指数関数的に大きくなること (Wikipediaより)
 - 例: 『[フカシギの数え方](#)』は組み合わせ爆発による問題
 - 機械学習においてよいモデルを学習するためには、情報を欠落せずに次元を落とすことが極めて重要！
- 分析に有用な次元だけに絞り込む
 - 出現回数が上位10,000件以下の単語は無視
→ 生成される文書ベクトルは10,000次元となる
 - どの文書にも現れる単語はidfが低いので無視
(演習では6カテゴリ中5カテゴリ以下に現れる単語のみ考慮)
 - 特定の文書にしか現れない単語も特殊すぎるので無視
(演習では6カテゴリ中3カテゴリ以上に現れる単語のみ考慮)

検証 1 : tf-idf値はそのカテゴリにおけるその単語の重要度を表せているか？

ある単語に対する各カテゴリごとのtf-idf値

	「裁判官」	「画家」	「生育」	「細胞」	「融資」	「衆議院」
動物	0.0000	0.0006	0.0057	0.3792	0.0000	0.0000
芸術	0.0022	0.0890	0.0000	0.0025	0.0030	0.0000
経済	0.0012	0.0000	0.0007	0.0007	0.0576	0.0023
法	0.0750	0.0013	0.0000	0.0000	0.0000	0.0213
植物	0.0005	0.0032	0.2161	0.2527	0.0000	0.0000
政治	0.0298	0.0011	0.0006	0.0000	0.0022	0.0693

- 「裁判官」といえば法か政治（どちらかといえば法）
- 「画家」といえば芸術
- 「細胞」といえば動植物
- 「融資」といえば経済（でも芸術も少しだけ重要らしい...）
- 「衆議院」といえば政治（でも法も重要）

検証 2 :

tf-idf値が高い単語はそのカテゴリを特徴づけているか？

tf-idf値が高い単語上位10件

	1位	2位	3位	4位	5位	6位	7位	8位	9位	10位
動物	細胞	家畜	寄生	個体	飼育	哺乳類	ウシ	化石	昆虫	生殖
芸術	カメラ	スタジオ	漫画	フィルム	美術	デジタル	レンズ	映像	露出	バロック
経済	マルクス	資本	投資	景気	金融	会計	所得	貨幣	銀行	買収
法	監査	憲法	司法	教皇	立法	カトリック	裁判	公法	審査	権力
植物	品種	栽培	種子	細胞	cm	生育	花粉	果実	光合成	乾燥
政治	憲法	主権	監査	権力	議員	独裁	請求	立法	衛星	政党

- その分野に特有の単語が上位に現れている
 - 「品種」「マルクス」「スタジオ」
- 複数カテゴリで上位に挙がっている単語もある
 - 「細胞」は動物・植物
 - 「憲法」は法・政治
- 意外な発見もある？
 - 法に「教皇」「カトリック」

カテゴリ間の類似度

tf-idfベクトルによるカテゴリ間の類似度行列

	動物	芸術	経済	法	植物	政治
動物	1	0.077	0.042	0.026	0.389	0.022
芸術	0.077	1	0.114	0.076	0.064	0.082
経済	0.042	0.114	1	0.153	0.045	0.188
法	0.026	0.076	0.153	1	0.015	0.713
植物	0.389	0.064	0.045	0.015	1	0.019
政治	0.022	0.082	0.188	0.713	0.019	1

- 「動物」は「植物」と似ている
- 「芸術」はどれともあまり似てないけど、あえて言うなら「経済」
- 「経済」は「芸術」「法」「政治」と同程度似ている
- 「法」と「政治」は似ている

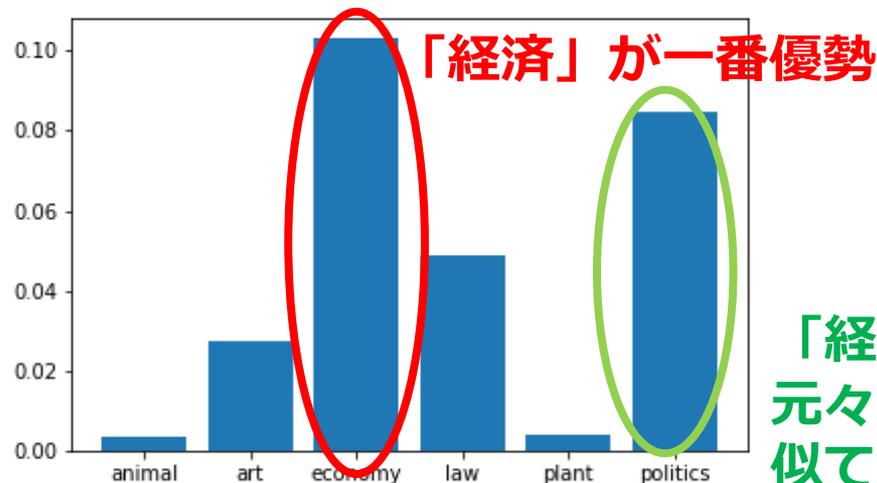
未知の文書はどのカテゴリの記事かを推定しよう

目的：新しいWikipediaの記事 $sample$ がどのカテゴリに入れるべき記事かを推定したい

Step 1: 学習済みのモデル (vectorizer) を使って $sample$ をベクトル化

Step 2: $sample$ のベクトルと各カテゴリから得た文書ベクトルとの類似をコサイン類似度で評価

Wikipedia 「分業 (ID: "204500")」 は経済カテゴリ



「経済」と「政治」は元々文書ベクトルが似ているのでこちらも大きく出る

教師なし学習によるトピック分類

- 先のtf-idfの課題ではすでにカテゴリごとに分類された記事が大量に用意されていた
 - 分類済みの文書集合を「教師 (supervisor)」として、モデルを教師あり学習 (supervised learning)
- 問題：**教師データがない場合**
 - 全く未分類の文書集合が与えられて、「これを適当に分類してね」と言われたら？
 - すでに分類済みであったとしても、「もっと他の分け方ないの？」と言われたら？

→**教師なし学習 (unsupervised learning)によるトピック分類**
- よく使われるのは**潜在的ディリクレ配分法 (LDA: Latent Dirichlet Allocation)**

Word Embedding

演習 : Word2vec.ipynb

Word Embeddingとは？

- 機械学習でテキストを学習させるためには何らかの方法でベクトル化する必要がある
- 1つの方法がone-hotベクトル
 - 大規模なコーパスになると語彙サイズは数万となる
→ 単語につき数万次元のベクトルになる
→ 次元が高いわりにスパースなので学習が効率が悪い
- 数万次元のone-hotベクトルを数百次元（たとえば200次元）のベクトルに**圧縮**する方法はないか？
 - しかし単に復元できればOKではない！
 - 機械学習では数値の大小関係を学習してしまう
 - ある単語Aのベクトルが別の単語Bのベクトルより大きい場合、それは妥当といえるようなベクトル表現でなければならない
 - 似た意味の単語のベクトル同士は近く、意味が大きく違う単語のベクトル同士は遠くあるべき

単語の意味情報を持たせたベクトルの圧縮

可逆圧縮と不可逆圧縮

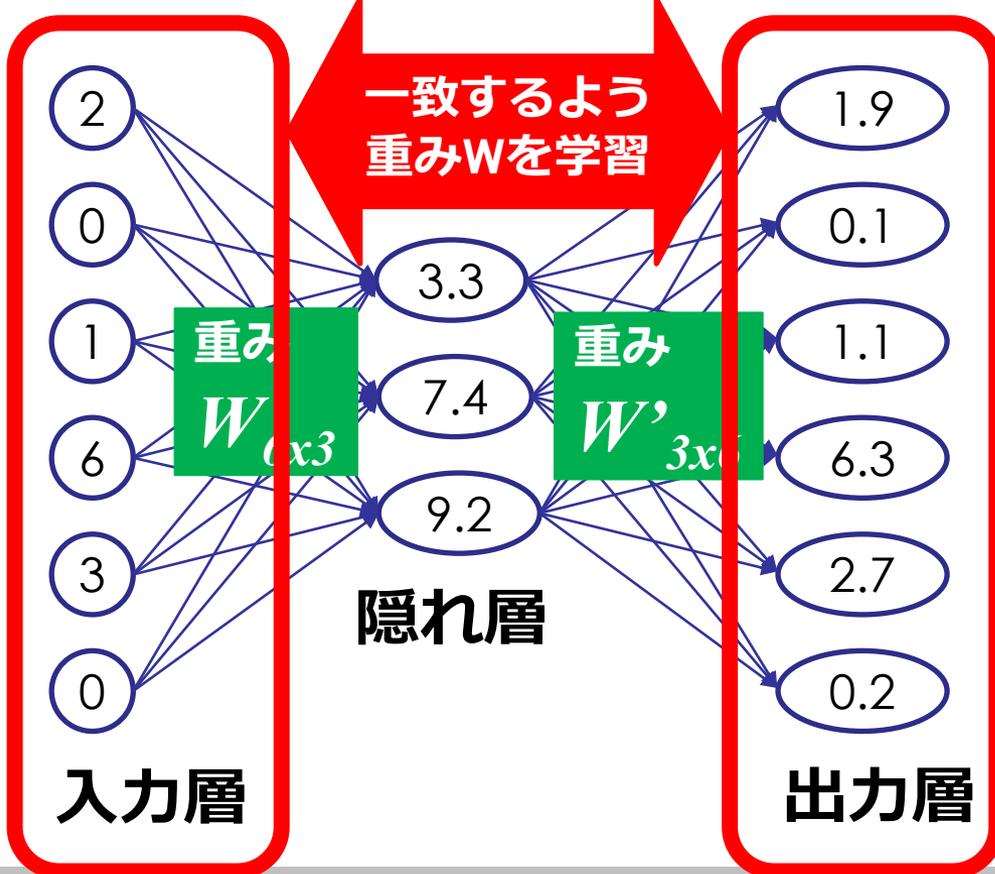
- 可逆圧縮：元のデータを完全に復元できる
例) ファイルの圧縮 zipなど
- **不可逆圧縮**：情報が欠損するため元のデータを完全には復元できない
例) jpeg圧縮（見た目に重要でない情報の欠損を許す）、
主成分分析（PCA）による圧縮



アーティ
ファクトが
現れる

Neural Networkによる不可逆圧縮: auto-encoder

様々なベクトル（例では6次元）について、
ネットワークで入力したベクトルと出力するベクトルが
似た値になるならば、
隠れ層（例では3次元）はそのベクトルの圧縮ベクトルである



例えば
[2, 0, 1, 6, 3, 0]を入力したら、
[1.9, 0.1, 1.1, 6.3, 2.7, 0.2]が出力される。
[5, 4, 2, 5, 3, 0]を入力したら、
[5.2, 3.8, 2.1, 5.1, 2.9, 0.1]が出力される。

ならば、**隠れ層の数値**
（左の例では3次元）がわかれば、
データを概ね再現できる！

学習フェーズ：
様々なデータを与えて重みWを更新

利用フェーズ：

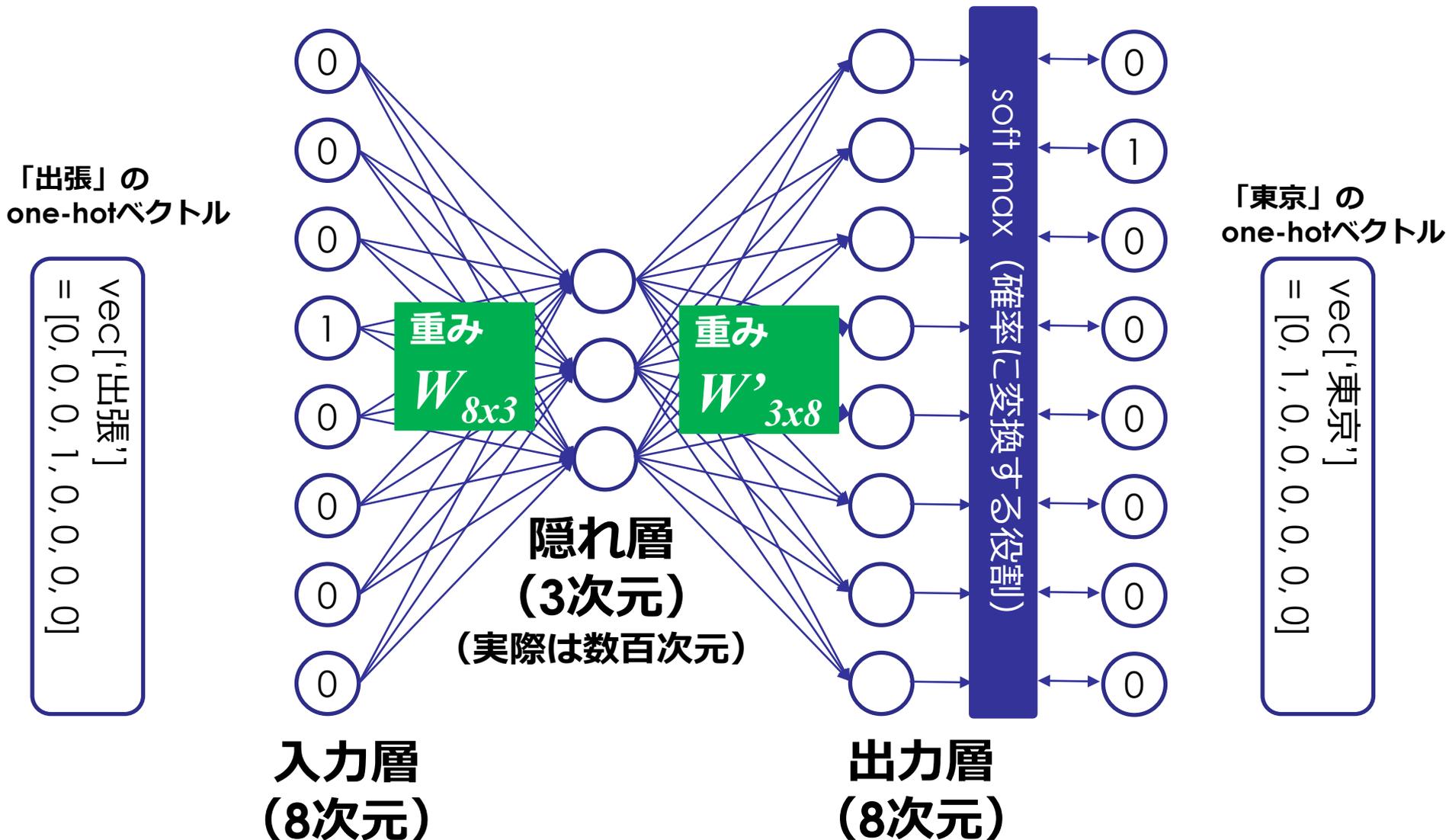
- 各データを3次元に圧縮
- あとはWが1つあれば元のデータを再現できる

単語の意味を持たせた圧縮を行うためには？

- 入力層と出力層に与えるベクトルは？
 - 文の単語穴埋め問題
 - Q) 「出張に行くため [mask] 駅から地下鉄に乗った。」
 - A) 「東京」
 - 入力層に「出張」や「行く」や「駅」や「地下鉄」を入れ、「東京」が出力されるよう重みを学習
 - これを何十万文の全単語に対して行う（同じ文も何度も学習）
- なぜその圧縮ベクトルが意味を表すのか？
 - ベクトルが意味を表すとは？
= 近い単語のベクトルは近く、遠い単語のベクトルは遠い
例) 「東京」と「大阪」は近いが「リンゴ」は遠い
 - 「東京」と意味が似た「大阪」の周辺単語は似ている
 - Q) 「出張に行くため [mask] 駅から地下鉄に乗った。」
 - A) 「大阪」
- **単語の意味が分からないと解けないようなQAを解けるよう学習すれば、圧縮ベクトルは意味情報を維持し、残りを捨てるよう圧縮される！**
 - auto-encoderでは何の情報も捨ててもいい
 - word2vecではQAを解くのに不要な情報を優先的に捨てていく

Word2vec: Skip-gram

語彙サイズが8だったとする（実際は数百万）

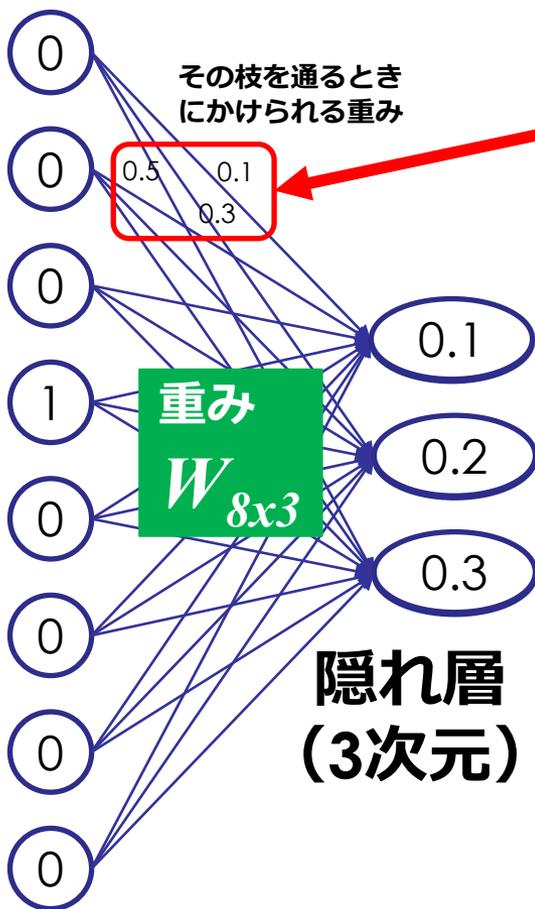


Word2vec: Skip-gram

たとえば

$$W_{8 \times 3} =$$

$[[0.1 \ 0.3 \ 0.5],$
 $[0.2 \ 0.4 \ 0.5],$
 $[0.5 \ 0.3 \ 0.2],$
 $[0.1 \ 0.2 \ 0.3],$
 $[0.6 \ 0.2 \ 0.3],$
 $[0.4 \ 0.1 \ 0.2],$
 $[0.5 \ 0.5 \ 0.1],$
 $[0.1 \ 0.3 \ 0.2]]$ なら



「出張」の one-hotベクトル

$vec[‘出張’]$
 $= [0, 0, 0, 1, 0, 0, 0, 0]$

入力層 (8次元)

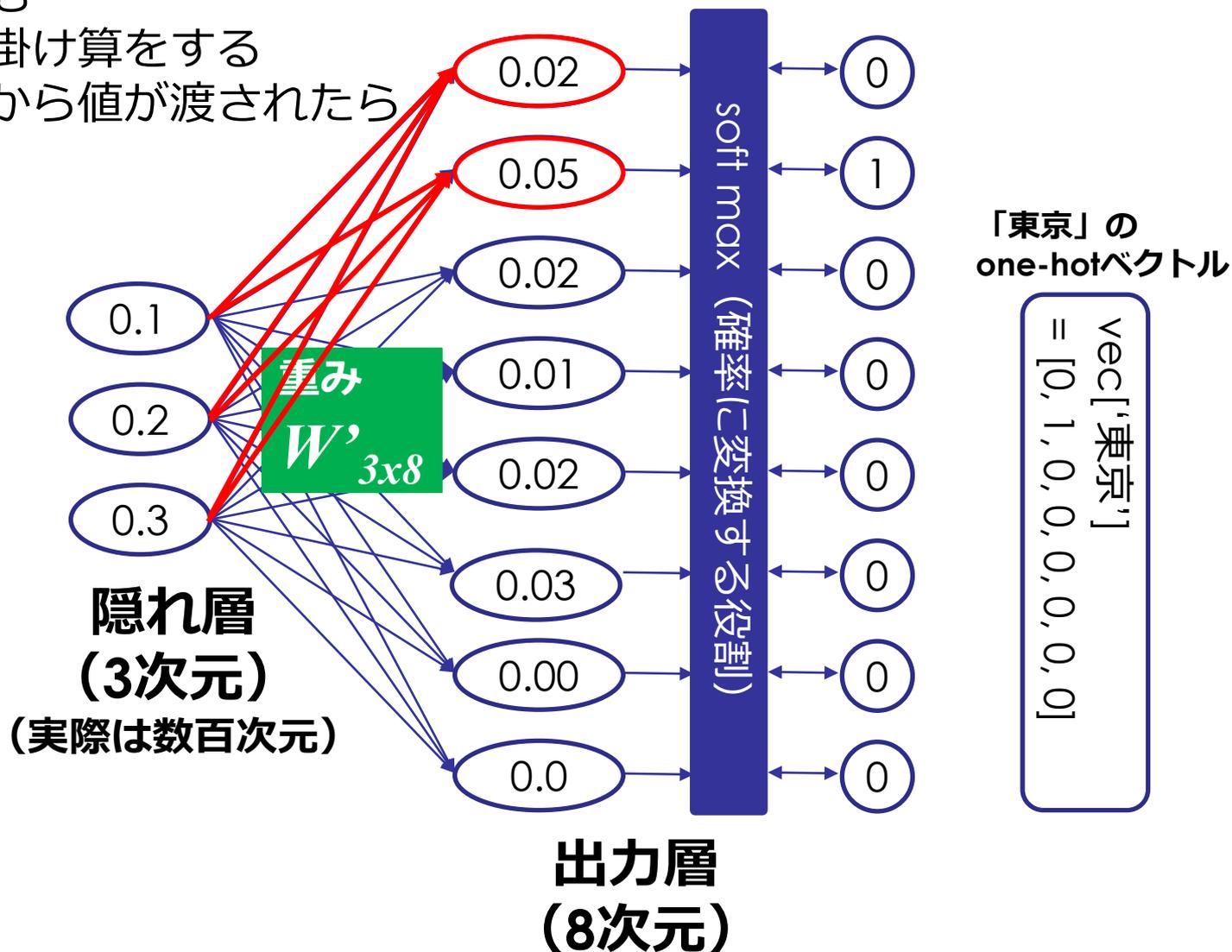
$$vec[‘出張’] \cdot W_{8 \times 3} =$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} [0.1 \ 0.3 \ 0.5], \\ [0.2 \ 0.4 \ 0.5], \\ [0.5 \ 0.3 \ 0.2], \\ [0.1 \ 0.2 \ 0.3], \\ [0.6 \ 0.2 \ 0.3], \\ [0.4 \ 0.1 \ 0.2], \\ [0.5 \ 0.5 \ 0.1], \\ [0.1 \ 0.3 \ 0.2] \end{bmatrix} = \begin{bmatrix} 0 \times [0.1 \ 0.3 \ 0.5], \\ 0 \times [0.2 \ 0.4 \ 0.5], \\ 0 \times [0.5 \ 0.3 \ 0.2], \\ 1 \times [0.1 \ 0.2 \ 0.3], \\ 0 \times [0.6 \ 0.2 \ 0.3], \\ 0 \times [0.4 \ 0.1 \ 0.2], \\ 0 \times [0.5 \ 0.5 \ 0.1], \\ 0 \times [0.1 \ 0.3 \ 0.2] \end{bmatrix}$$

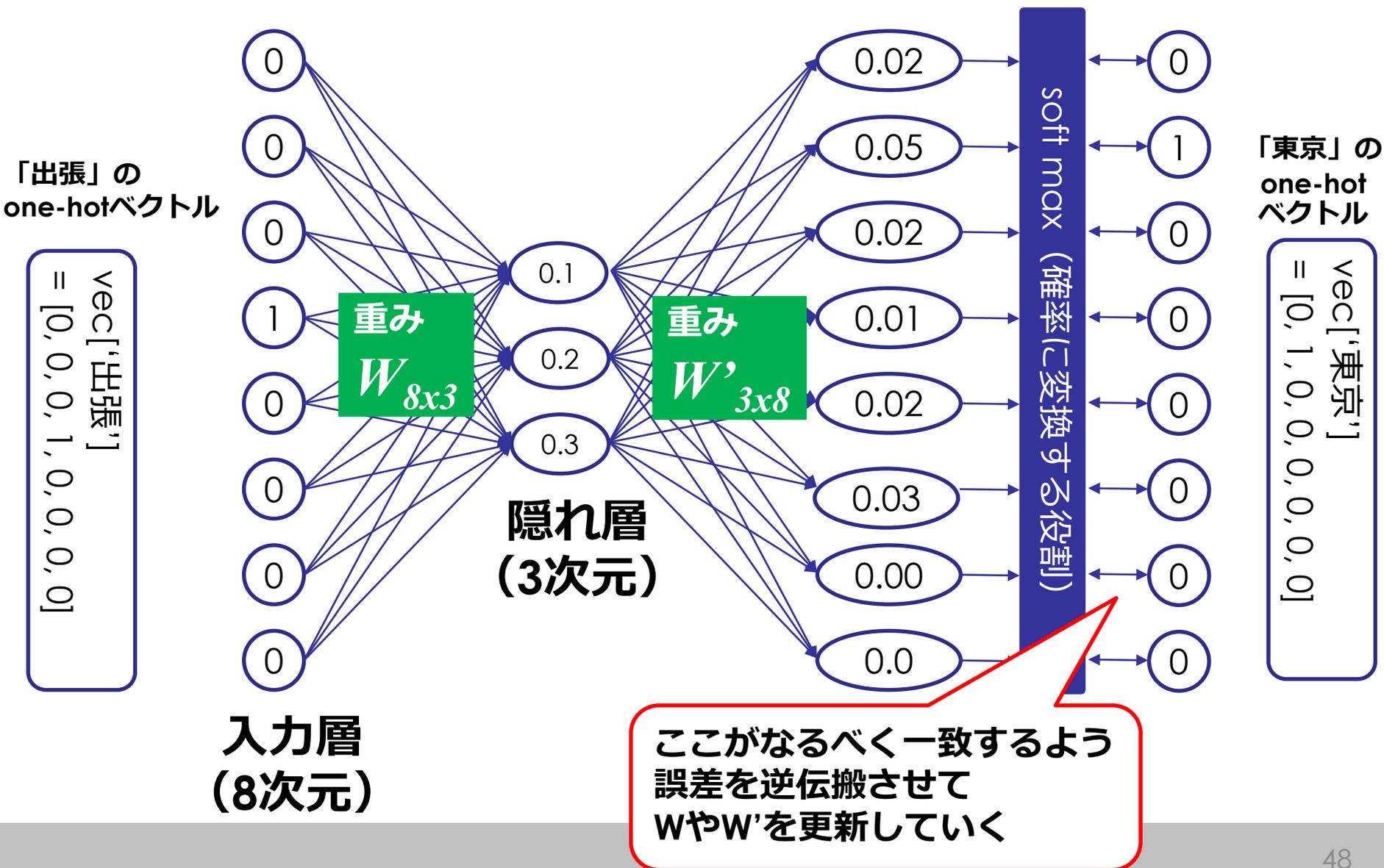
$$=[0.1 \ 0.2 \ 0.3]$$

Word2vec: Skip-gram

隠れ層から出力層も
同じように行列の掛け算をする
ただし、複数の枝から値が渡されたら
それらを加算する



Word2vec: Skip-gram

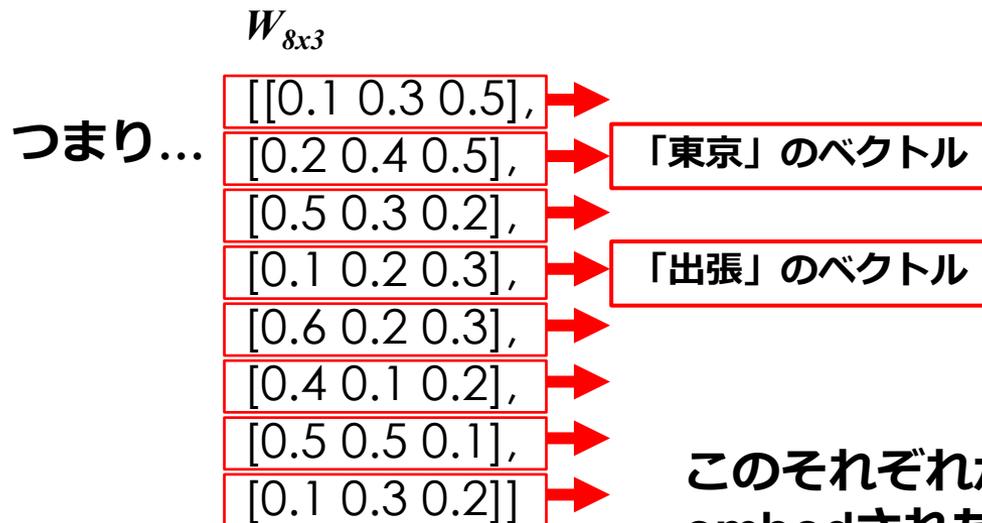


単語ベクトルはどこか？

$$\begin{aligned} \text{vec}[\text{「出張」}] \cdot W_{8 \times 3} &= \begin{matrix} \text{「出張」の} \\ \text{one-hotベクトル} \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} [0.1 & 0.3 & 0.5], \\ [0.2 & 0.4 & 0.5], \\ [0.5 & 0.3 & 0.2], \\ [0.1 & 0.2 & 0.3], \\ [0.6 & 0.2 & 0.3], \\ [0.4 & 0.1 & 0.2], \\ [0.5 & 0.5 & 0.1], \\ [0.1 & 0.3 & 0.2] \end{bmatrix} \\ &= \begin{matrix} 0 \times [0.1 & 0.3 & 0.5], \\ 0 \times [0.2 & 0.4 & 0.5], \\ 0 \times [0.5 & 0.3 & 0.2], \\ \boxed{1 \times [0.1 & 0.2 & 0.3]}, \\ 0 \times [0.6 & 0.2 & 0.3], \\ 0 \times [0.4 & 0.1 & 0.2], \\ 0 \times [0.5 & 0.5 & 0.1], \\ 0 \times [0.1 & 0.3 & 0.2] \end{matrix} \end{aligned}$$

$$= [0.1 \ 0.2 \ 0.3]$$

これは「出張」を表す
3次元のベクトルである！



このそれぞれが、その次元に対応する単語の
embedされた3次元のベクトルとなる

演習 : Word2vecによるWord Embedding

- トピック分析用ライブラリgensimを用いる
- wikipediaの6つのカテゴリから抽出した、計2101個の記事（分かち書き済み）をコーパスとする
- パラメータ
 - size=200: 200次元のベクトルを計算
 - mini_count=20: コーパスに20回以上現れる単語のみ考慮
 - batch_words=10000: コーパスを10000単語ごとに分けて学習
 - iter=5: 全コーパスを学習するのを5回繰り返す
 - window=15: 前後15単語との関係を考慮する
- 使ってみよう!
 - コーパスに含まれており、かつ学習対象となった単語についてしかベクトル化されていないので注意！

3. 単語ベクトルの活用

「動物」の類義語

	類似度
門	0.67
類	0.67
脊椎動物	0.64
爬虫類	0.63
節足動物	0.61
生物	0.60
類縁	0.59
有	0.59
ヌタウナギ	0.59
鳥類	0.58

「日本」の類義語

	類似度
中国	0.66
米国	0.63
沖縄	0.63
欧米	0.62
インドネシア	0.57
アメリカ合衆国	0.57
広く	0.56
琉球	0.56
列島	0.56
諸島	0.55

「カメラ」の類義語

	類似度
フィルム	0.95
デジタル	0.90
TTL	0.87
機種	0.84
撮影	0.84
素子	0.83
画像	0.83
レンズ	0.83
露出	0.83
撮像	0.82

3.2 ベクトルの加算・減算

「国会」 - 「日本」 + 「米国」

('下院', 0.8014905452728271)
('任期', 0.798736572265625)
('理事', 0.7881254553794861)
('大臣', 0.7878961563110352)
('会議', 0.7873814105987549)

「生育」 - 「植物」 + 「動物」

('飼育', 0.6654659509658813)
('獣', 0.6576670408248901)
('居住', 0.6255872249603271)
('放牧', 0.5914555191993713)
('ゾウ', 0.5872169733047485)

「ロンドン」 - 「イギリス」 + 「日本」

('東京', 0.7394543290138245)
('日経', 0.6854637265205383)
('サイト', 0.6852266192436218)
('毎年', 0.6691043376922607)
('記念', 0.6596246957778931)

「ロンドン」 - 「イギリス」 + 「フランス」

('パリ', 0.8611776828765869)
('ベルリン', 0.8477520942687988)
('ベルギー', 0.8415089845657349)
('結成', 0.8208703994750977)
('ブリュッセル', 0.8129469752311707)

Word2vecによる単語ベクトルが、加減算の成り立つ空間を構成していることを示す

新しいWord Embedding手法

新しい Word Embedding 手法

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- 2018年の自然言語処理の国際学会ACLで Googleが発表した汎用モデル
- BooksCorpus (800M 単語) とEnglish Wikipedia (2,500M words)で 事前学習したモデルを様々なタスクにfine-tuning
- 質問応答タスクや固有表現認識、センチメント分析など8つのタスクでState of the Artsを達成！

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

[Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"](#), pp. 4171–4186, ACL 2018.

SQuAD : 質問応答タスク

The Stanford Question Answering Dataset

F値

83.1

The first recorded travels by Europeans to China and back date from this time. The most famous traveler of the period was the Venetian Marco Polo, whose account of his trip to "Cambaluc," the capital of the Great Khan, and of life there astounded the people of Europe. The account of his travels, Il milione (or, The Million, known in English as the Travels of Marco Polo), appeared about the year 1299. Some argue over the accuracy of Marco Polo's accounts due to the lack of mentioning the Great Wall of China, tea houses, which would have been a prominent sight since Europeans had yet to adopt a tea culture, as well the practice of foot binding by the women in capital of the Great Khan. Some suggest that Marco Polo acquired much of his knowledge through contact with Persian traders since many of the places he named were in Persian.

Wikipediaのうち
質問に対する
回答文を含む段落

入力

How did some suspect that Polo learned about China instead of by actually visiting it?

質問文

マルコ・ポーロは中国について、実際に訪問するのではなくどのように学んだと疑われていますか？

Answer: through contact with Persian traders

ペルシャ人の商人との交流から

回答文
(該当する文の位置)

出力

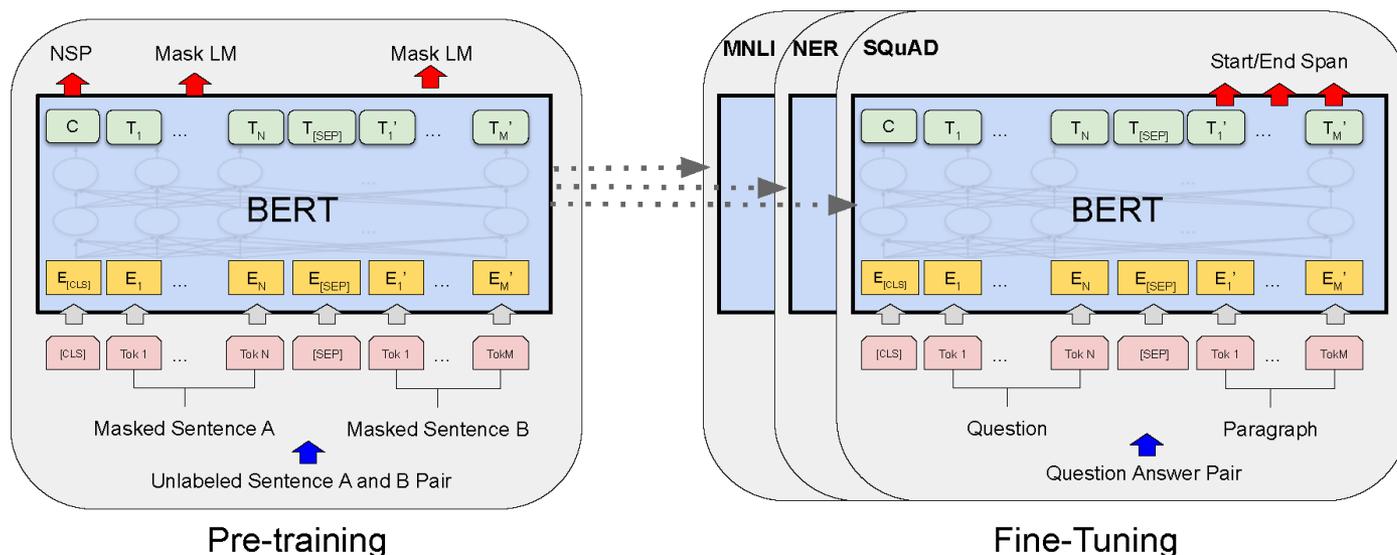
<https://rajpurkar.github.io/mlx/qa-and-squad/>

Fine-tuning : 事前学習モデルのタスク適用

- 事前学習モデルを初期値としてタスクを学習

質問応答タスクの場合 :

- 事前学習済みのモデルを初期値としてFine-tuning
- Fine-tuningでは質問文と、回答を含むWikipediaの段落を入力して、回答にあたる文の位置を推定



[Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", pp. 4171-4186, ACL 2018.](#)

[MASK]で隠された単語を予測するよう学習

Input: The man went to **[MASK]** store with **[MASK]** dog.

Target: **the** **his**

- 全体の15%の単語について、以下のルールに従って書き換えることにより汎用性を確保
 - 80%の単語は隠す
例) “my dog is hairy !” → “my dog is [MASK]”
 - 10%の単語はランダムに置き換え
例) “my dog is hairy !” → “my dog is apple”
 - 10%の単語は変更しない
例) “my dog is hairy !” → “my dog is hairy.”

- 以下の2文はつながりますか？

文1) [CLS] the man went to [MASK] store [SEP]

文2) he bought a gallon [MASK] milk [SEP]



IsNextを予想

文1) [CLS] the man [MASK] to the store [SEP]

文2) penguin [MASK] are flight ##less birds [SEP]

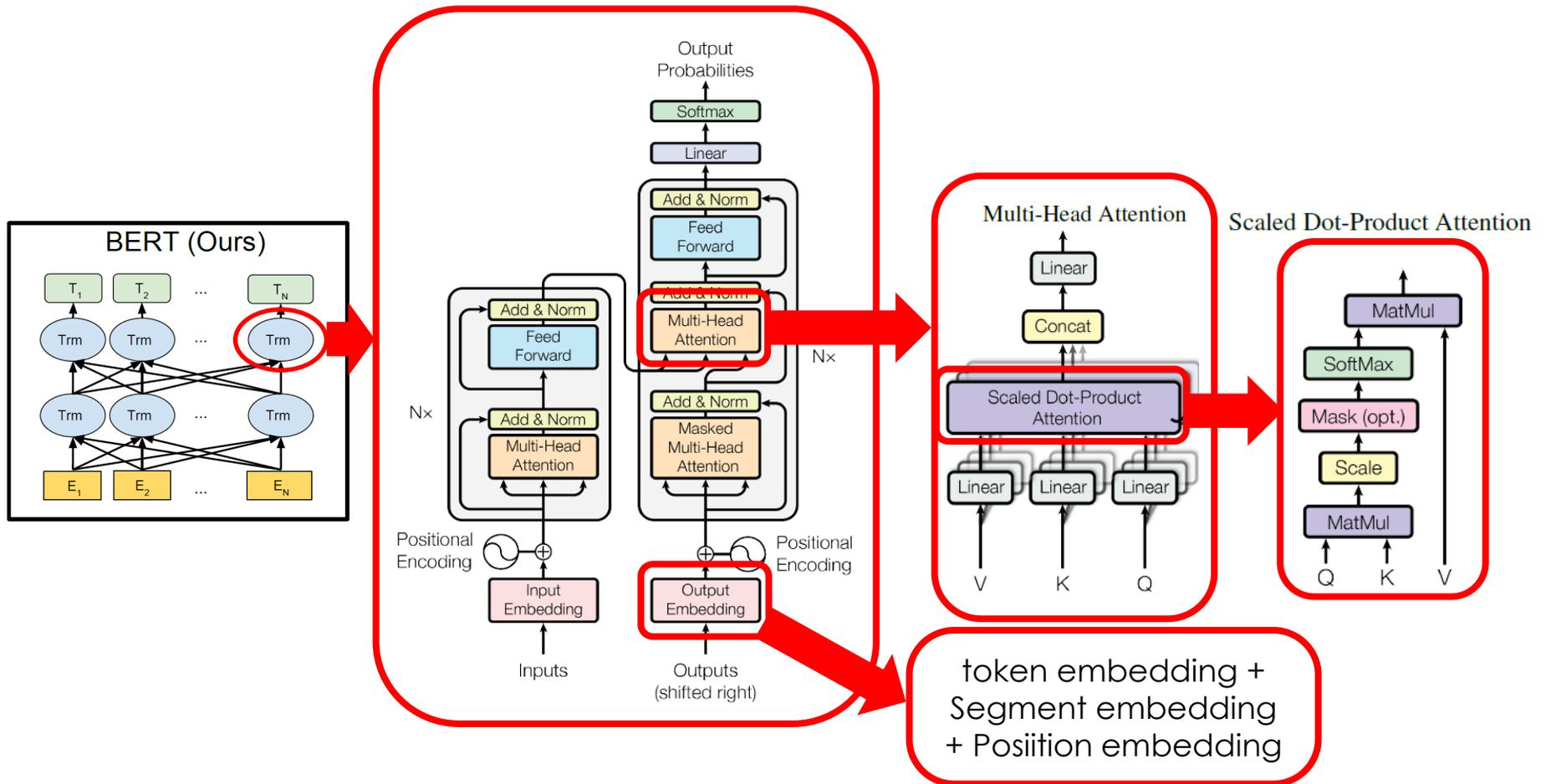


NotNextを予想

※"##less"は「-less」で終わる単語の接尾辞を表す

- 50%ではつながる、50%ではつながらない文

BERTのアーキテクチャ



[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, pp. 4171–4186, ACL 2018.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, “Attention Is All You Need”, NIPS2017

BERT日本語 事前学習済み (Pre-trained) モデル

- ライブラリはまだ整っていない
- 学習済みモデルは利用可能
 - [Jumanを使った単語分割ベース](#)
 - 日本語Wikipediaにより学習 (京大言語メディア研)
 - Sentencepieceによる単語分割ベース
 - [日本語Wikipediaにより学習](#) (菊田陽平氏)
 - [日本語Twitterにより学習](#) (株式会社ホットリンク)
 - [huggingfaceのtransformers](#)にpre-train Japanese modelとして登録
 - [東北大学 自然言語処理研究室](#)
- [BERT Word Embedding Tutorial](#)
 - BERTにより文や単語をベクトル化するPythonコードを解説

本日の学習のまとめ

- テキストのベクトル空間モデル
 - 単語のベクトル表現
 - 文書のベクトル表現
 - 文書のトピック分析の前処理・ストップワード
- トピック分析
 - 教師あり文書分類：tf-idf値による重要語抽出
 - 演習) Wikipedia記事のトピック分析
- Word Embedding
 - ニューラルネットワークによる単語の分散表現生成：Word2Vec
 - より進んだ手法：BERT
 - 演習) gensimによるWord2vecの学習と可視化