Deep Learning基礎講座

Natural Language Processing

Day 1

自然言語処理 (Natural Language Processing, NLP) は、
画像と共に研究が活発な分野のひとつ

- 特に、2018年は大きく盛り上がった年となった

講義シラバス

Day 1:　前処理、Word2Vec

Day 2:　Encoder-Decoderモデル

Day 3:　Transformerモデル

NLPにおける前処理について

基本的な前処理は２ステップ

文章の単語化（形態素化）

単語（形態素）のID化

文章の単語化（形態素化）

形態素とは

- 意味をもつ表現要素の最小単位
- それ以上分解したら意味をなさなくなるところまで分割して抽出された音素のまとまりのひとつひとつを指す

英語の形態素化： nltk　　（⇠ 英語にも形態素解析はある！）

日本語の形態素化： mecab

```
import nltk
nltk.download('punkt')

sentence = "I don't know."
nltk.word_tokenize(sentence)


=> ['I', 'do', "n't", 'know', '.']
```

```
import MeCab
tagger = MeCab.Tagger('-Ochasen')
tagger.parse('')

def parse(node):
    output = []
    for row in node.split('\n'):
        word = row.split('\t')[0]
        if word == 'EOS':
            break
        else:
            output.append(word)
    return output

sentence = "あのイーハトーヴォのすきとおった風、夏でも底に冷たさをもつ青いそら"
node = tagger.parse(sentence)
print(parse(node))
```

=> [‘あの', 'イーハトーヴォ', 'の', 'すきとおっ', 'た', '風', '、', '夏', 'で', 'も', '底', 'に', '冷た', 'さ', 'を', 'もつ', '青い', 'そら']

Mecab自身は、条件付き確率場 (Conditional Random Fields, CRF) を用いて形態素解析をしている

- 系列ラベリングに用いられる教師あり学習

- cf. 隠れマルコフモデル (Hidden Markov Model, HMM)

Mecabを自分の環境にインストールしたい場合

```
$ sudo apt install mecab
$ sudo apt install libmecab-dev
$ sudo apt install mecab-ipadic-utf8

$ pip install mecab-python3
```

※ 要参考： mecab-ipadic-neologd

(https://github.com/neologd/mecab-ipadic-neologd)

単語（形態素）のID化

形態素化した“単語”それぞれにIDを振っていく

その際、注意すべき点が２つ

- 予約語： <BOS>, <EOS>, <PAD>, <UNK>　※演習で確認

- 全単語にIDをつけると、データ量が多いケースではメモリ不足になる

⇢ 文字列から数値列となるので、機械学習の入力として使える形となる

では、実際の学習はどうするか？

シンプルなモデルは、単語IDをone-hotベクトルに変換したものを
入力し、そのまま隠れ層・出力層へ伝播させていくもの

…だが、one-hotベクトルをそのまま用いるのでは非効率的

単語IDのone-hotベクトル化の問題：

1. ベクトルの次元が巨大かつスパースなので、そのままDense層を
   つなげると無駄な計算が多くなってしまう（ほとんどゼロなので）

2. 機械的に振られたIDがベクトルになったものなので、
   可視化したとしても、そこから単語の関係性は分からない

そこで用いられるのがEmbedding

- 単語を密 (dense) なベクトルに変換する
- Embedding層では、one-hotベクトルで計算するのではなく、
  IDのまま行列をLookupすることで計算効率を上げる

Embeddingがやっていること自体はとてもシンプル

- スパースなone-hotベクトルを、密なベクトルに変換する

- すなわち、単語に重みをかける

$$\boldsymbol{h = Wx}$$

式（理論）としては、密なベクトル $\boldsymbol{h}$ は

one-hotベクトル $\boldsymbol{x}$ に行列 $\boldsymbol{W}$ をかけることで得られる

- $\boldsymbol{x}$ の次元は単語数、 $\boldsymbol{h}$ の次元は埋め込みたい数（ハイパーパラメータ）

… しかし実際の計算では、行列において
単語ID番目の列だけを抽出すればよい（圧倒的に計算が早い！）

- one-hotベクトルをかける、ということは列を抽出すること

（なので、ライブラリのEmbeddingに与える引数は単語ID列）

$$\boldsymbol{h = Wx}$$

- $\boldsymbol{W}$ (Embedding Matrix) はいわゆる重みだが、単語IDをルックアップして
  単語ベクトルに変換する行列なので、Lookup Tableとも呼ばれる

- $\boldsymbol{W}$ の値をうまく設定することができれば、密な単語ベクトル $\boldsymbol{h}$ は、
  意味のある数値を持つことになる

（すべての $\boldsymbol{h}$ は $\boldsymbol{W}$ が表しているので、 $\boldsymbol{W}$ は単語ベクトルの表現そのものであることにも注意！）

Embedding層における重み行列Wを求めることができれば、

それが密な単語ベクトルの表現そのもの

Word2Vec

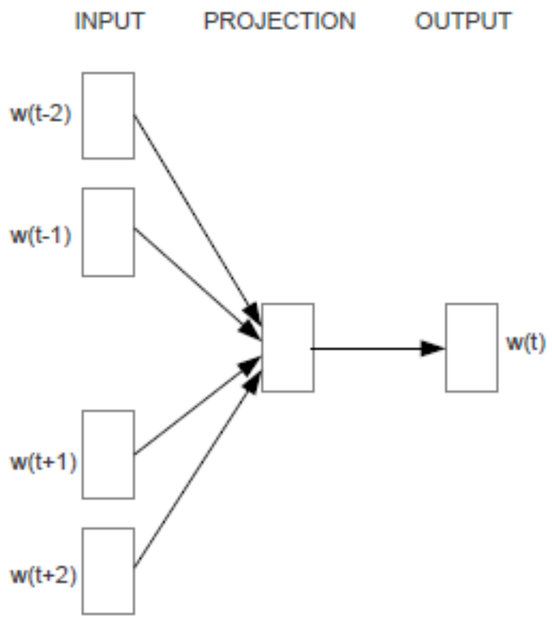
Word2Vecについて

Word2Vecの具体的な手法は２つ

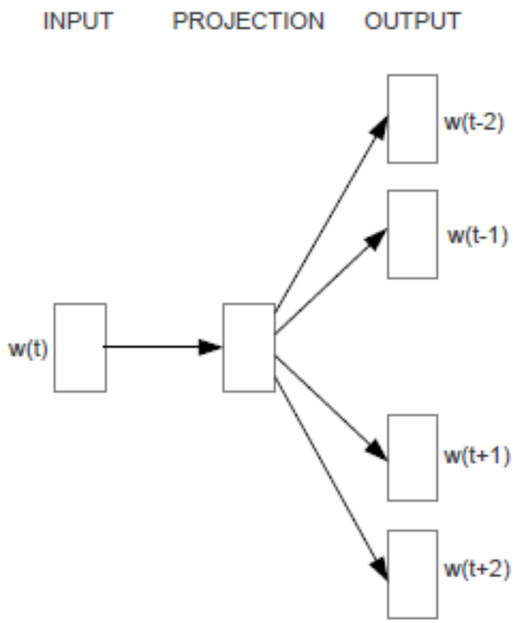
- CBOW (Continuous Bag of Words)

- Skip-Gram

いずれも、着目している単語と周辺単語を用いる



CBOW



Skip-gram

周辺単語が入力

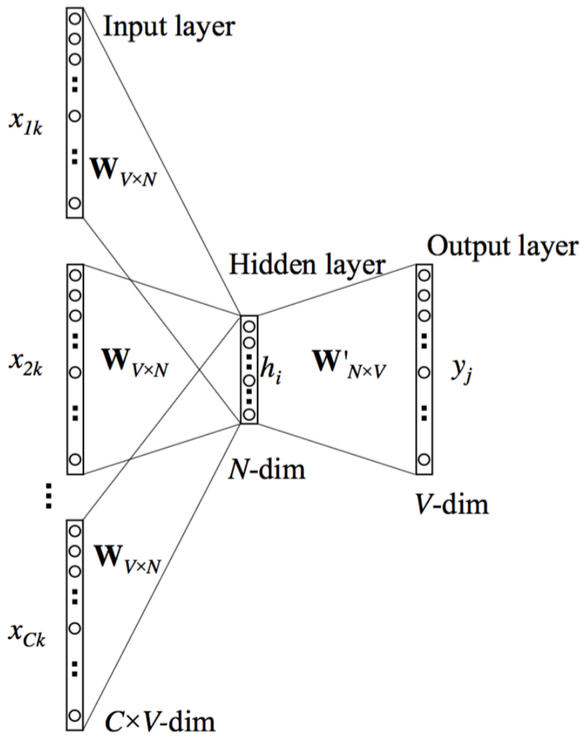
周辺単語が出力

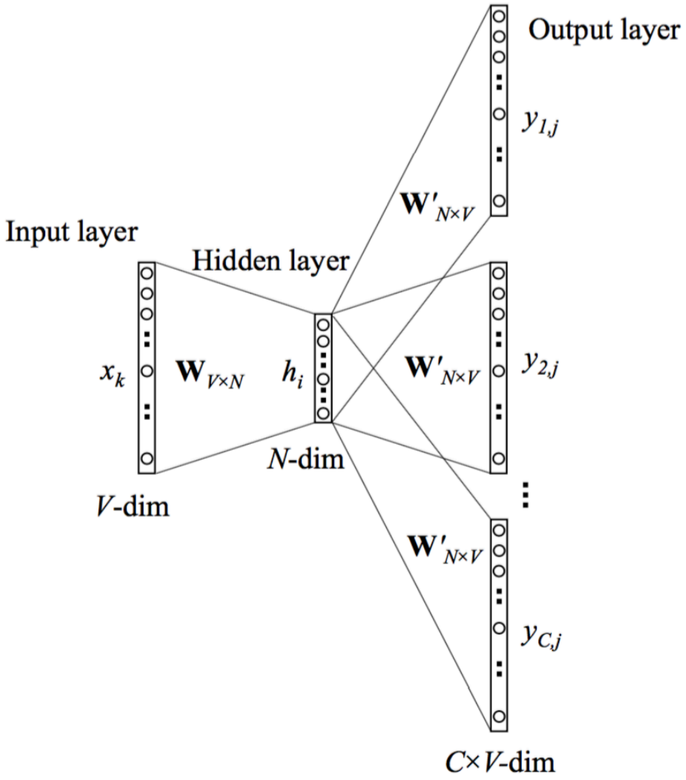
※ つまり、周辺単語を用いることで単語の関係性をベクトル表現で学習しようとしている

Layer構造は、いずれも

Embedding + Dense

非常にシンプルなネットワーク！





着目単語と周辺単語の同時出現確率を最大化したい（最尤推定）

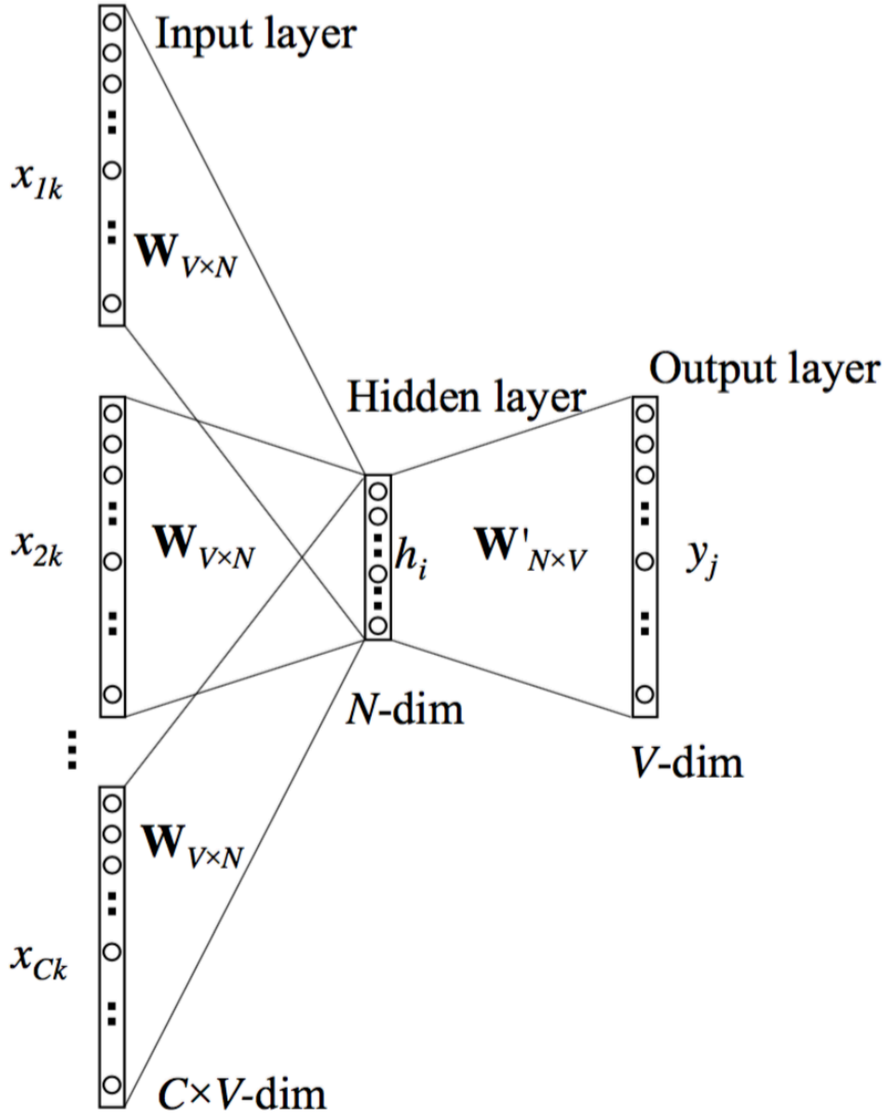
⇢ 誤差関数の設定

$$\mathcal{L}_{CBOW} = -\frac{1}{T}\sum_{t=1}^{T}\log p(w_t \mid w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c})$$

$$\mathcal{L}_{SG} = -\frac{1}{T}\sum_{t=1}^{T}\log p(w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c} \mid w_t)$$

$$\mathcal{L}_{CBOW} = -\frac{1}{T}\sum_{t=1}^{T}\log p(w_t \mid w_{t-c},\ldots,w_{t-1},w_{t+1},\ldots,w_{t+c})$$

$$= -\frac{1}{T}\sum_{t=1}^{T}\log(\text{softmax}((\sum_{-c\le j\le c, j\neq 0} \boldsymbol{v}_{t+j})^T \boldsymbol{W})_{s_t})$$

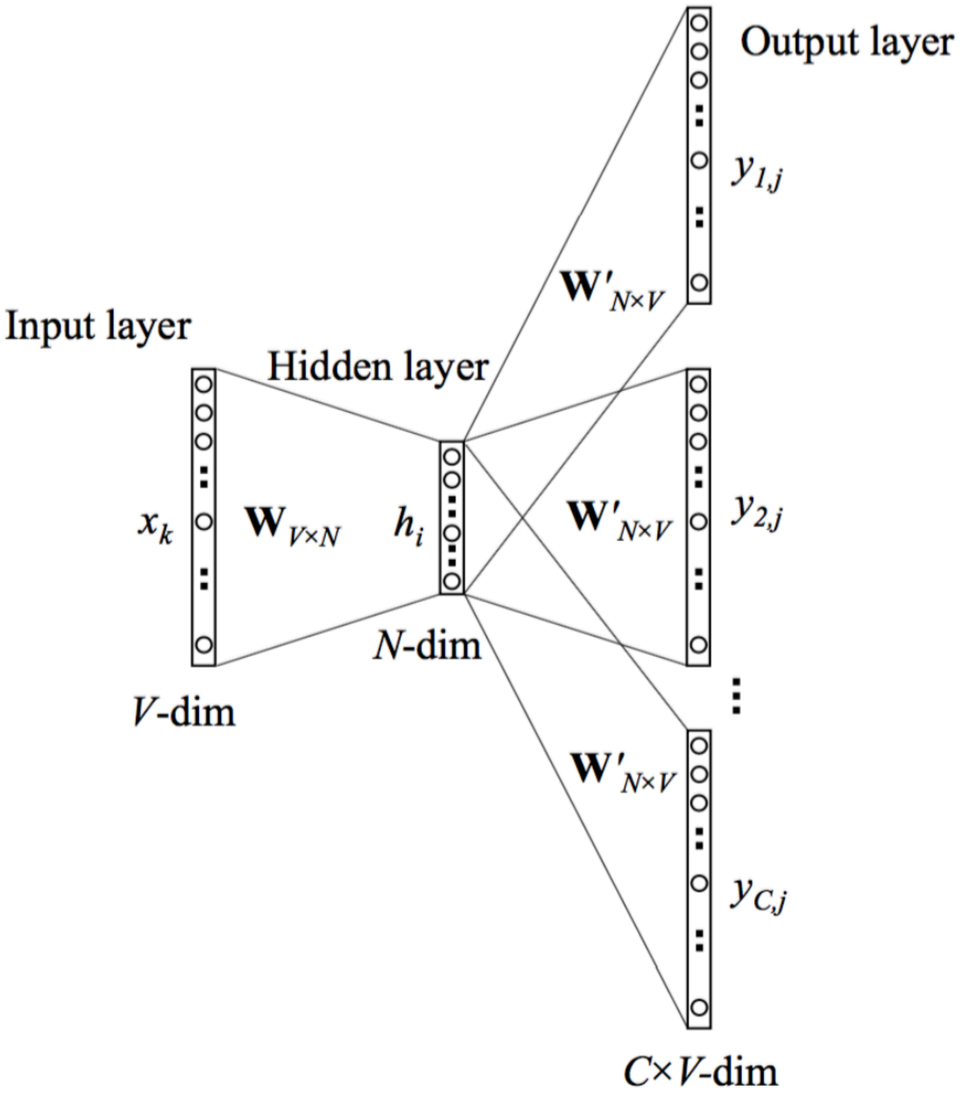


$$\mathcal{L}_{SG} = -\frac{1}{T}\sum_{t=1}^{T}\log p(w_{t-c},\ldots,w_{t-1},w_{t+1},\ldots,w_{t+c}\,|\,w_t)$$

$$= -\frac{1}{T}\sum_{t=1}^{T}\log\prod_{-c\le j\le c, j\neq 0} p(w_{t+j}|w_t)$$

$$= -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\neq 0}\log p(w_{t+j}|w_t)$$

$$= -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\neq 0}\log(\mathrm{softmax}(\boldsymbol{v}_t^T\boldsymbol{W})_{s_{t+j}})$$



CBOW, Skip-Gramでも理論的には問題ないが、
計算量が単語数に比例し、計算コストが高くつく

⇢ Skip-Gram with Negative Sampling の登場
（＝負例）

$$\mathcal{L}_{SG} = -\frac{1}{T}\sum_{t=1}^{T} \log p(w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c} \mid w_t)$$

$$= -\frac{1}{T}\sum_{t=1}^{T} \log \prod_{-c \le j \le c, j \neq 0} p(w_{t+j} | w_t)$$

$$= -\frac{1}{T}\sum_{t=1}^{T} \sum_{-c \le j \le c, j \neq 0} \log p(w_{t+j} | w_t)$$

$$= -\frac{1}{T}\sum_{t=1}^{T} \sum_{-c \le j \le c, j \neq 0} \log(\mathrm{softmax}(\boldsymbol{v}_t^T \boldsymbol{W})_{s_{t+j}})$$

softmaxに用いるWの計算コストが高い

⇢ Wではなく密な単語ベクトルのみを用いる

その代わりに、負例を用いて学習精度を高める

$$\mathcal{L}_{SGNS} = \frac{1}{T}\sum_{t=1}^{T} NLL(w_t)$$

負例が出ない確率を最大化

$$= -\frac{1}{T}\sum_{t=1}^{T} \log\left(\prod_{-c\leq j\leq c, j\neq 0} p(w_{t+j}|w_t) \prod_{w_n\in S} (1 - p(w_n|w_t)) \right)$$

$$= -\frac{1}{T}\sum_{t=1}^{T} \left(\sum_{-c\leq j\leq c, j\neq 0} \log p(w_{t+j}|w_t) + \sum_{w_n\in S} \log(1 - p(w_n|w_t)) \right)$$

$$= -\frac{1}{T}\sum_{t=1}^{T} \left(\sum_{-c\leq j\leq c, j\neq 0} \log \sigma(\boldsymbol{v}'^{T}_{t+j}\boldsymbol{v}_t) + \sum_{w_n\in S} \log(1 - \sigma(\boldsymbol{v}'^{T}_{n}\boldsymbol{v}_t)) \right)$$

softmaxの代わりにsigmoid

（Wを計算に用いない）

$$= -\frac{1}{T}\sum_{t=1}^{T} \left(\sum_{-c\leq j\leq c, j\neq 0} \log \sigma(\boldsymbol{v}'^{T}_{t+j}\boldsymbol{v}_t) + \sum_{w_n\in S} \log \sigma(-\boldsymbol{v}'^{T}_{n}\boldsymbol{v}_t) \right)$$

演習