91258 Natural Language Processing

Lesson 12. word2vec

Alberto Barrón-Cedeño

Alma Mater Studiorum-Università di Bologna
a.barron@unibo.it @_albarron_

07/11/2023

Previously

- Introduction to neural networks
- First Keras neural network
- Considerations when building/training a network

Table of Contents

Chapter 6 of Lane et al. (2019)

Introduction

Introduction

Previously

BoW Each token represents one dimension

TF–IDF Document- and corpus-level statistics

LSA Dimensional reduction for a dense representation

Drawbacks

- They ignore the (nearby) context of a word
- They ignore the overall meaning of a statement

Introduction

Word vectors. Numerical vector representations of word semantics, or meaning, including literal and implied meaning (Lane et al., 2019, p. 182)

Math with words

$q =$ "She was a key physics figure in Europe in the early 20th century"

```
answer_vector = wv['woman'] + wv['Europe'] + \
                wv['physics'] + wv['scientist']
```

Even better:

```
answer_vector = wv['woman'] + wv['Europe'] + \
                wv['physics'] + wv['scientist'] - \
                wv['male'] - wv['man']
```

Word Vectors

Word Vectors

Intuition

Word2vec (Mikolov et al., 2013)

- Learns the *meaning* of words by processing a large corpus[1]
- The corpus is not labeled
  → **unsupervised**

Can we train a NN to predict word occurrences near a target word w?

We don't care about the prediction (that's handy, but not important here). We care about the resulting **internal representation**

[1] For instance, 100B words from the Google News Groups

Word Vectors

Vector Algebra (again)

- word2vec transforms token-occurrence vectors into lower-dimensional vectors
- The dimension is usually in the 100s (e.g., 100, 200, 300)

Typical process
Input: Text
Output: Text

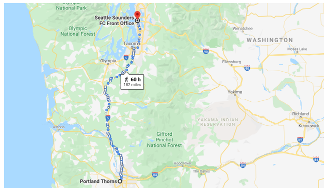
1. Compute vectors
2. Do algebra
3. Map back to text

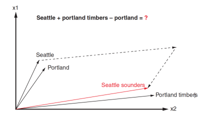
Word Vectors

Vector Algebra (again)

$$\text{Portland Timbers} + \text{Seattle} - \text{Portland} = ?$$

$$ourput\_vector = wv['\text{Seattle}'] + wv['\text{Portland Timbers}'] - wv['\text{Portland}']$$





[a]

[a] (Lane et al., 2019, p. 188)

Word2vec "knows" that

- $dist(\text{Portland}, \text{Portland Timbers}) \approx dist(\text{Seattle}, \text{Seattle Sounders})$
- The diffs between the pairs of vectors are roughly in the same direction

Word Vectors

Some "typical" operations/properties

Gender	king + woman − man → queen
Pl/Sg	$\vec{x}_{coffee} - \vec{x}_{coffees} \approx \vec{x}_{cup} - \vec{x}_{cups} \approx \vec{x}_{cookie} - \vec{x}_{cookies}$
Locations	San Francisco − California + Colorado → Denver
Culture	tortellini − Bologna + Valencia → paella **?**

Computing word2vec representations

The grand canal of Venice (Claude Monet, 1908)

Alternatives to Build word2vec Representations

skip-gram

Input one (target) word

Output context words

CBOW (continuous bag-of-words)

Input context words

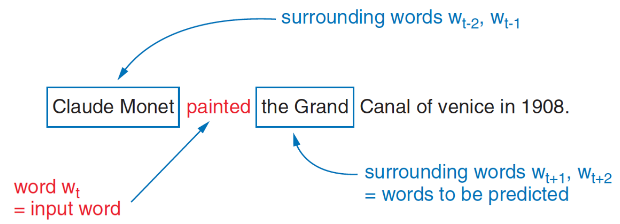
Output one target word

Skip-Gram

Definition Skip-grams are *n*-grams that contain gaps (skips over intervening tokens)

Input: one word

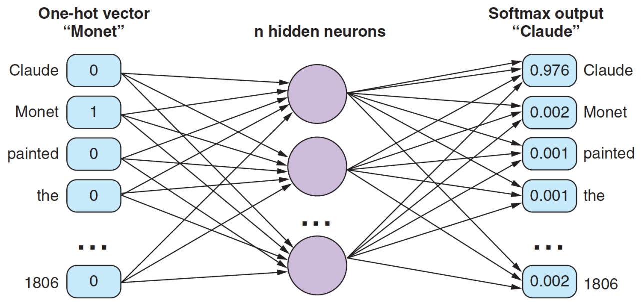
Output: context words



(Lane et al., 2019, p. 192)

Skip-Gram

Neural Network Structure



- n is the number of vector dimensions in the model
- M is the number of input/output neurons; $M = |\text{vocabulary}|$
- The output activation function is a **softmax**
  Typical in multi-class problems; $\sum_M = 1.0$

(Lane et al., 2019, p. 193)

Skip-Gram

Learning the Representations (1/3)

- **Window size**: 2 words $\rightarrow$ 5-grams
- **Input**: the token at time t: w_t
- **Output**: all context tokens on the left and right, one at a time

$$s = w_1\ w_2\ w_3\ w_4\ w_5\ w_6\ w_7\ w_8\ w_9\ w_{10}$$

$$[\ldots]\ w_{t-2}\ w_{t-1}\ \underline{w_t}\ w_{t+1}\ w_{t+2}\ [\ldots]$$

Skip-Gram
Learning the Representations (2/3)

Example: "Claude Monet painted the Grand Canal of Venice in 1908."

input	**expected output**			
w_t	w_{t-2}	w_{t-1}	w_{t+1}	w_{t+2}
Claude			Monet	painted
Monet		Claude	painted	the
painted	Claude	Monet	the	Grand
the	Monet	painted	Grand	Canal
Grand	painted	the	Canal	of
Canal	the	Grand	of	Venice
of	Grand	Canal	Venice	in
Venice	Canal	of	in	1908
in	of	Venice	1908	
1908	Venice	in		

(Lane et al., 2019, p. 194)

Skip-Gram
Learning the Representations (3/3)

Training

- Both input and output are a one-hot vector
- $n-1$ iterations when using n-grams:

$$[\ldots]\ w_{t-2}\ w_{t-1}\ \underline{w_t}\ w_{t+1}\ w_{t+2}\ [\ldots]$$

i	input	output	i	input	output	i	input	output
0	w_t	w_{t-2}	4	w_{t+1}	w_{t-1}	8	w_{t+2}	w_t
1	w_t	w_{t-1}	5	w_{t+1}	w_t	9	w_{t+2}	w_{t+1}
2	w_t	w_{t+1}	6	w_{t+1}	w_{t+2}	10	w_{t+2}	w_{t+3}
3	w_t	w_{t+2}	7	w_{t+1}	w_{t+3}	11	w_{t+2}	w_{t+4}

- To simplify the loss calculation, the softmax is converted to one-hot

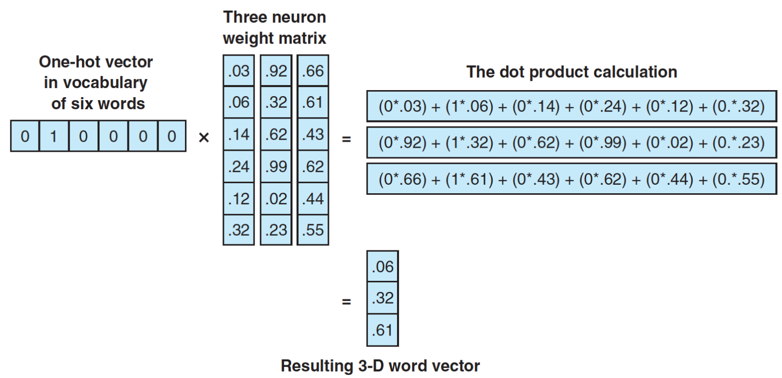
Skip-Gram
Outcome

- The output layer can be *ignored*[1]
- Semantically similar words end up with similar vectors —they were trained to **predict similar contexts**
- The weights from input to hidden layer are used to compute **embeddings**

$$wv_w = dot(one\,hot_w, W)$$

[1] Tweaking this procedure could result in a language model

Skip-Gram
Embedding Computation

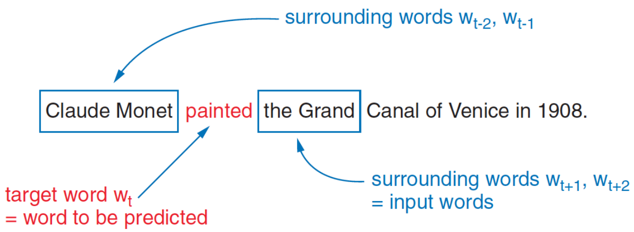


CBOW

Definition Continuous bag-of-words

Input: context words

Output: target (centre) word



(Lane et al., 2019, p. 196)

CBOW

Learning the Representations (1/3)

Window size: 2 words $\rightarrow$ 5-grams

Input: multi-hot vector (sum of all context one-hot vectors)

Output: one-hot vector

$$s = w_1\ w_2\ w_3\ w_4\ w_5\ w_6\ w_7\ w_8\ w_9\ w_{10}$$

$$[\ldots]\ \underline{w_{t-2}\ w_{t-1}}\ w_t\ \underline{w_{t+1}\ w_{t+2}}\ [\ldots]$$

CBOW

Learning the Representations (2/3)

Example: "Claude Monet painted the Grand Canal of Venice in 1908."

input				**expected output**
w_{t-2}	w_{t-1}	w_{t+1}	w_{t+2}	w_t
		Monet	painted	Claude
	Claude	painted	the	Monet
Claude	Monet	the	Grand	painted
Monet	painted	Grand	Canal	the
painted	the	Canal	of	Grand
the	Grand	of	Venice	Canal
Grand	Canal	Venice	in	of
Canal	of	in	1908	Venice
of	Venice	1908		in
Venice	in			1908

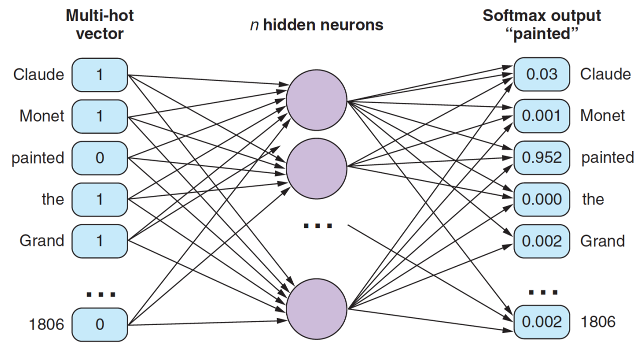
(Lane et al., 2019, p. 194)

CBOW

Learning the Representations (3/3)

Training

- The input is a multi-hot vector: $w_{t-2} + w_{t-1} + w_{t+2} + w_{t+2}$
- The output is a one-hot vector w_t



Final Remarks

Skip-gram

- Works well with **small** corpora
- High-frequency $[2, 3]$-grams can be added as single terms (e.g., New_York, Atlanta_Braves)
- High-frequency tokens are subsampled ($\sim$ to IDF over stopwords)
- Negative sampling. Not all weights are updated given a pair, just a few negative samples (much cheaper; roughly the same result)

CBOW

- Higher accuracy for frequent words
- Much faster to train

References

Lane, H., C. Howard, and H. Hapkem
2019. *Natural Language Processing in Action*. Shelter Island, NY: Manning Publication Co.

Mikolov, T., K. Chen, G. Corrado, and J. Dean
2013. Efficient estimation of word representations in vector space. In *Arxiv*.