Introduction to word embeddings

Agenda

- language modeling
- limitations of traditional n-gram language models
- Bengio et al. (2003)’s NNLM
- Google’s word2vec (Mikolov et al. 2013)

Antoine Tixier, DaSciM team, LIX
November 2015
Language model

• Goal: determine $P(s = w_1 \ldots w_k)$ in some domain of interest

$$P(s) = \prod_{i=1}^{k} P (w_i \mid w_1 \ldots w_{i-1})$$

e.g., $P(w_1w_2w_3) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1w_2)$

• Traditional n-gram language model assumption:
  “the probability of a word depends only on context of $n - 1$ previous words”

$$\Rightarrow \hat{P}(s) = \prod_{i=1}^{k} P (w_i \mid w_{i-n+1} \ldots w_{i-1})$$

• Typical ML-smoothing learning process (e.g., Katz 1987):
  1. compute $\hat{P}(w_i \mid w_{i-n+1} \ldots w_{i-1}) = \frac{\#w_{i-n+1} \ldots w_{i-1}w_i}{\#w_{i-n+1} \ldots w_{i-1}}$ on training corpus
  2. smooth to avoid zero probabilities
Traditional n-gram language model

Limitation 1): curse of dimensionality

• Example
  - train a 10-gram LM on a corpus of 100,000 unique words
  - space: 10-dimensional hypercube where each dimension has 100,000 slots
  - model training ↔ assigning a probability to each of the $100,000^{10}$ slots
  - probability mass vanishes → more data is needed to fill the huge space
  - the more data, the more unique words! → vicious circle
  - what about corpuses of $10^6$ unique words?

• → in practice, contexts are typically limited to size 2 (trigram model)
  e.g., famous Katz (1987) smoothed trigram model

• → such short context length is a limitation: a lot of information is not captured
Traditional n-gram language model

*Limitation 2): word similarity ignorance*

- We should assign similar probabilities to Obama speaks to the media in Illinois and the President addresses the press in Chicago.

- This does not happen because of the “one-hot” vector space representation:

  \[
  \begin{align*}
  \text{obama} &= [0 \ 0 \ 0 \ 0 \ \ldots \ 0 \ 1 \ 0 \ 0] \\
  \text{president} &= [0 \ 0 \ 0 \ 1 \ \ldots \ 0 \ 0 \ 0 \ 0] \\
  \text{speaks} &= [0 \ 0 \ 1 \ 0 \ \ldots \ 0 \ 0 \ 0 \ 0] \\
  \text{addresses} &= [0 \ 0 \ 0 \ 0 \ \ldots \ 0 \ 0 \ 1 \ 0] \\
  \text{illinois} &= [1 \ 0 \ 0 \ 0 \ \ldots \ 0 \ 0 \ 0 \ 0] \\
  \text{chicago} &= [0 \ 1 \ 0 \ 0 \ \ldots \ 0 \ 0 \ 0 \ 0]
  \end{align*}
  \]

  \[
  \begin{align*}
  \text{obama} \cdot \text{president} &= 0 \\
  \text{speaks} \cdot \text{addresses} &= 0 \\
  \text{illinois} \cdot \text{chicago} &= 0
  \end{align*}
  \]

- In each case, word pairs share no similarity.
- This is obviously wrong.
- We need to encode *word similarity* to be able to *generalize*.
Word embeddings: distributed representation of words

- Each unique word is mapped to a point in a real continuous m-dimensional space
- Typically, $|V| > 10^6$, $100 < m < 500$

$$w_i \in V \xrightarrow{\text{mapping } C} \mathbb{R}^m$$

$w_1 \quad \text{obama} \quad w_{|V|}$

obama $= [0 \ldots \ldots 1 \ldots \ldots 0]$  

$|V|$  

“one-hot” vector

$\text{feature}_1 \quad \text{feature}_m$

obama $= [0.12 \ldots -0.25]$  

$m \ll |V|$  

feature vector

- Fighting the curse of dimensionality with:
  - compression (dimensionality reduction)
  - smoothing (discrete to continuous)
  - densification (sparse to dense)

- Similar words end up close to each other in the feature space
Neural Net Language Model (Bengio et al. 2003)

For each training sequence: input = (context, target) pair: \((w_{t-n+1} \ldots w_{t-1}, w_t)\)

objective: minimize \(E = -\log \hat{P}(w_t | w_{t-n+1} \ldots w_{t-1})\)
**NNLM Projection layer**

- Performs a simple table lookup in $C_{|V|, m}$: concatenate the rows of the shared mapping matrix $C_{|V|, m}$ corresponding to the context words

Example for a two-word context $w_{t-2}w_{t-1}$:

\[
\begin{align*}
C_{|V|, m} & \rightarrow \begin{bmatrix}
C(w_1) \\
\vdots \\
C(w_{t-2}) \\
C(w_{t-1}) \\
\vdots \\
C(w_{|V|})
\end{bmatrix}
\end{align*}
\]

Concatenate ① and ② → $C(w_{t-2})$ and $C(w_{t-1})$

- $C_{|V|, m}$ is **critical**: it contains the weights that are tuned at each step. After training, it contains what we’re interested in: the **word vectors**
NNLM hidden/output layers and training

• Softmax (log-linear classification model) is used to output positive numbers that sum to one (a multinomial probability distribution):

\[
\widehat{P}(w_i = w_t \mid w_{t-n+1} \ldots w_{t-1}) = \frac{e^{yw_i}}{\sum_{i'=1}^{\mid V \mid} e^{yw_{i'}}}
\]

where:
- \( y = b + U \cdot \tanh(d + H \cdot x) \)
- \( \tanh \) : nonlinear squashing (link) function
- \( x \) : concatenation \( C(w) \) of the context weight vectors seen previously
- \( b \) : output layer biases (\( |V| \) elements)
- \( d \) : hidden layer biases (\( h \) elements). Typically \( 500 < h < 1000 \)
- \( U : |V| \times h \) matrix storing the hidden-to-output weights
- \( H : (h \times (n - 1)m) \) matrix storing the projection-to-hidden weights

\( \rightarrow \theta = (b, d, U, H, C) \)

• Complexity per training sequence: \( n \times m + n \times m \times h + h \times |V| \)

computational bottleneck: **nonlinear hidden layer** (\( h \times |V| \) term)

• **Training** is performed via stochastic gradient descent (learning rate \( \varepsilon \)):

\[
\theta \leftarrow \theta + \varepsilon \cdot \frac{\partial E}{\partial \theta} = \theta + \varepsilon \cdot \frac{\partial \log \widehat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})}{\partial \theta}
\]

(weights are initialized randomly, then updated via backpropagation)
NNLM facts

- tested on Brown (1.2M words, $|V| \approx 16K$, 200K test set) and AP News (14M words, $|V| \approx 150K$ reduced to 18K, 1M test set) corpuses
- Brown: $h = 100$, $n = 5$, $m = 30$
- AP News: $h = 60$, $n = 6$, $m = 100$, 3 week training using 40 cores
- 24% and 8% relative improvement (resp.) over traditional smoothed n-gram LMs in terms of test set perplexity: geometric average of $1/\hat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$

- Due to complexity, NNLM can’t be applied to large data sets → poor performance on rare words

- Bengio et al. (2003) initially thought their main contribution was a more accurate LM. They let the interpretation and use of the word vectors as future work

- On the opposite, Mikolov et al. (2013) focus on the word vectors
Google’s word2vec (Mikolov et al. 2013a)

• Key idea of word2vec: achieve better performance not by using a more complex model (i.e., with more layers), but by allowing a simpler (shallower) model to be trained on much larger amounts of data

• Two algorithms for learning words vectors:
  - CBOW: from context predict target (focus of what follows)
  - Skip-gram: from target predict context

• Compared to Bengio et al.’s (2003) NNLM:
  - no hidden layer (leads to 1000X speedup)
  - projection layer is shared (not just the weight matrix)
  - context: words from both history & future:
    “You shall know a word by the company it keeps” (John R. Firth 1957:11):

...Pelé has called Neymar an excellent player...
...At the age of just 22 years, Neymar had scored 40 goals in 58 internationals...
...occasionally as an attacking midfielder, Neymar was called a true phenomenon...

These words will represent Neymar
word2vec’s Continuous Bag-of-Words (CBOW)

For each training sequence: \( \text{input} = (\text{context, target}) \text{ pair: } (w_{t-n/2} \ldots w_{t-1} w_{t+1} \ldots w_{t+n/2}, w_t) \)

objective: minimize \( E = -\log \hat{P}(w_t | w_{t-n/2} \ldots w_{t-1} w_{t+1} \ldots w_{t+n/2}) \)

**OUTPUT LAYER**

hierarchical softmax. \( \text{t}^{\text{th}} \text{ output } = P(w_i = w_t | w_{t-n/2} \ldots w_{t-1} w_{t+1} \ldots w_{t+n/2}) \)

\(|V|\) probabilities that sum to 1

**PROJECTION LAYER**

linear \( \frac{1}{n} \cdot C([\cdot]) \)

averaging

100 < \( m \) < 1000 typically

table lookup in shared \( C_{|V|,m} \)

**INPUT LAYER**

\( \vec{c} = 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \)

\(|V|\)

input context:

\( n/2 \) history words: \( w_{t-n/2} \ldots w_{t-1} \)
\( n/2 \) future words: \( w_{t+1} + \ldots + w_{t+n/2} \)

\( n \cong 8 \) typically
Weight updating intuition

- For each (context, target=\(w_t\)) pair, only the word vectors from matrix \(C\) corresponding to the context words are updated.
- Recall that we compute \(P(w_i = w_t \mid \text{context})\) \(\forall w_i \in V\). We compare this distribution to the true probability distribution (1 for \(w_t\), 0 elsewhere).
- If \(P(w_i = w_t \mid \text{context})\) is **overestimated** (i.e., \(> 0\), happens in potentially \(|V| - 1\) cases), some portion of \(C'(w_i)\) is **subtracted** from the context word vectors in \(C\), proportionally to the magnitude of the error.
- Reversely, if \(P(w_i = w_t \mid \text{context})\) is **underestimated** (< 1, happens in potentially 1 case), some portion of \(C'(w_i)\) is **added** to the context word vectors in \(C\).

→ at each step the words move away or get closer to each other in the feature space → clustering → analogy with a **spring force** layout. See online [demo](#) with Chrome.
word2vec facts

- Complexity is $n \times m + m \times \log|V|$ (Mikolov et al. 2013a)
- On Google news 6B words training corpus, with $|V| \sim 10^6$:
  - CBOW with $m = 1000$ took **2 days** to train on **140 cores**
  - Skip-gram with $m = 1000$ took **2.5 days** on **125 cores**
  - NNLM (Bengio et al. 2003) took **14 days** on **180 cores**, for $m = 100$ only!
    (note that $m = 1000$ was not reasonably feasible on such a large training set)
- word2vec training speed $\approx 100K$-$5M$ words/s
- Quality of the word vectors:
  - $\uparrow$ significantly with **amount of training data** and **dimension of the word vectors** ($m$),
    with diminishing relative improvements
  - measured in terms of accuracy on 20K semantic and syntactic association tasks.
    e.g., words in **bold** have to be returned:

<table>
<thead>
<tr>
<th>Capital-Country</th>
<th>Past tense</th>
<th>Superlative</th>
<th>Male-Female</th>
<th>Opposite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Athens: <strong>Greece</strong></td>
<td>walking: <strong>walked</strong></td>
<td>easy: <strong>easiest</strong></td>
<td>brother: <strong>sister</strong></td>
<td>ethical: <strong>unethical</strong></td>
</tr>
</tbody>
</table>

Adapted from Mikolov et al. (2013a)

- Best NNLM: 12.3% overall accuracy. Word2vec (with Skip-gram): 53.3%

[https://code.google.com/p/word2vec/](https://code.google.com/p/word2vec/)
Remarkable properties of word2vec’s word vectors

regularities between words are encoded in the difference vectors e.g., there is a constant country-capital difference vector
Remarkable properties of word2vec’s word vectors

constant female-male difference vector

Remarkable properties of word2vec’s word vectors

constant **male-female** difference vector

- Vector operations are supported and make intuitive sense:
  
  \[ w_{king} - w_{man} + w_{woman} \cong w_{queen} \]
  
  \[ w_{paris} - w_{france} + w_{italy} \cong w_{rome} \]

  \[ w_{windows} - w_{microsoft} + w_{google} \cong w_{android} \]

  \[ w_{einstein} - w_{scientist} + w_{painter} \cong w_{picasso} \]
  
  \[ w_{his} - w_{he} + w_{she} \cong w_{her} \]
  
  \[ w_{cu} - w_{copper} + w_{gold} \cong w_{au} \]

- Online [demo](http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd) (scroll down to end of tutorial)

Applications

• High quality word vectors boost performance of all NLP tasks, including document classification, machine translation, information retrieval…

• Example for English to Spanish machine translation:

About 90% reported accuracy (Mikolov et al. 2013c)
Application to document classification

With the BOW representation $D_1$ and $D_2$ are at equal distance from $D_0$. Word embeddings allow to capture the fact that $D_1$ is closer.

Resources

Papers:


**Google word2vec webpage** (with link to C code):
https://code.google.com/p/word2vec/

**Python implementation:**
https://radimrehurek.com/gensim/models/word2vec.html

**Kaggle tutorial on movie review classification with word2vec:**
https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-2-word-vectors

**Insightful blogpost:** http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/