Word Embeddings

Benjamin Roth, Nina Poerner

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilian-Universität München
beroth@cis.uni-muenchen.de

November 20, 2019
Motivation

- How to represent words in a neural network?
- Possible solution: indicator vectors of length $|V|$ (vocabulary size).

$$w^{(\text{the})} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \quad w^{(\text{cat})} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \quad w^{(\text{dog})} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

**Question:** Why is this a bad idea?

- Parameter explosion ($|V|$ might be $> 1M$)
- All word vectors are orthogonal to each other $\rightarrow$ no notion of word similarity
Motivation

- Learn one word vector $\mathbf{w}^{(i)} \in \mathbb{R}^D$ (“word embedding”) per word $i$
- Typical dimensionality: $50 \leq D \leq 1000 \ll |V|$
- Embedding matrix: $\mathbf{W} \in \mathbb{R}^{|V| \times D}$

**Question:** Advantages of using word vectors?

- We can express similarities between words, e.g., with cosine similarity:
  \[
  \cos(\mathbf{w}^{(i)}, \mathbf{w}^{(j)}) = \frac{\mathbf{w}^{(i)\top} \mathbf{w}^{(j)}}{\|\mathbf{w}^{(i)}\|_2 \cdot \|\mathbf{w}^{(j)}\|_2}
  \]

- Since the embedding operation is a *lookup operation*, we only need to update the vectors that occur in a given training batch.
Motivation

- Training from scratch: Initialize embedding matrix randomly and learn it during training phase
- $\rightarrow$ words that play similar roles w.r.t. task get similar embeddings
- e.g., from sentiment classification, we might expect $w_{\text{great}} \approx w_{\text{awesome}}$

**Question:** What could be a problem at test time?
- If training set is small, many words are unseen during training and therefore have random vectors

- We typically have more unlabelled than labelled data. Can we learn embeddings from the unlabelled data?
Motivation

- Distributional hypothesis: “a word is characterized by the company it keeps”’ (Firth, 1957)
- Basic idea: learn similar vectors for words that occur in similar contexts
- GloVe, Word2Vec, FastText
Questions?
Recap: Language Models

- **Question:** What is a Language Model?
  - Function to assign probability to a sequence of words.

- **Question:** What is an n-gram language Model?
  - Markov assumption: probability of word only depends on no more than $n - 1$ other (previous) words:

$$P(w_1 \ldots w_T) = \prod_{t=1}^{T} P(w_t | w_{t-1} \ldots w_{t-n+1})$$
Words in our vocabulary are represented as two sets of vectors:

- \( \mathbf{w}(i) \in \mathbb{R}^D \) if they are to be predicted
- \( \mathbf{v}(i) \in \mathbb{R}^D \) if they are conditioned on as context

Predict word \( i \) given previous word \( j \):

\[
P(i|j) = f(\mathbf{w}(i), \mathbf{v}(j))
\]

**Question:** What is a possible function \( f(\cdot) \)?
A Simple Neural Network Bigram Language Model

- Softmax!

\[ P(i|j) = \frac{\exp(w^{(i)}^T v(j))}{\sum_{k=1}^{|V|} \exp(w^{(k)}^T v(j))} \]

- Question: Problem with training softmax?
  - \( \Rightarrow \) Slow. Needs to compute dot products with the whole vocabulary for every single prediction.
Questions?
Context vectors $v$ are defined like before.

Word vectors $w$ are replaced by a binary tree:

```
  \theta^{(1)}
 /    \    /
|     \   /
\theta^{(2)} | \theta^{(3)}
  \   \   \   
  cat sat dog the
```
Hierarchical Softmax

- Each tree node $l$ has parameter vector $\theta^{(l)}$
- Probability of going left at node $l$ given context word $j$:
  \[ p(\text{left}|l, j) = \sigma(\theta^{(l)}^T v^{(j)}) \]
- Probability of going right: $p(\text{right}|l, j) = 1 - p(\text{left}|l, j)$
- Probability of word $i$ given $j$: product of probabilities on the path from root to $i$
Example

Calculate $p(sat|cat)$.

\[
p(sat|cat) = \sigma(\theta^{(1)}_T v^{(cat)}) [1 - \sigma(\theta^{(2)}_T v^{(cat)})]
\]
Questions

- **Question:** How many dot products do we need to calculate to get to $p(i|j)$? How does this compare to the naive softmax?
  - $\log_2 |V| \ll |V|$

- **Question:** Show that $\sum_{i'} p(i'|j)$ sums to 1.
Questions?
Another trick: **negative sampling** (aka *noise contrastive estimation*)

This changes the objective function, and the resulting model is not a language model anymore!

Idea: Instead of predicting probability distribution over whole vocabulary, make binary decisions for a small number of words.

Positive training set: Bigrams seen in the corpus.

Negative training set: Random bigrams (not seen in the corpus).
Negative Sampling: Likelihood

- **Given:**
  - Positive training set: \( \text{pos}(O) \)
  - Negative training set: \( \text{neg}(O) \)

\[
L = \prod_{(i,j) \in \text{pos}(O)} P(\text{pos}| w^{(i)}, v^{(j)}) \prod_{(i',j') \in \text{neg}(O)} P(\text{neg}| w^{(i')}, v^{(j')})
\]

- \( P(\text{pos}| w, v) = \sigma(w^T v) \)
- \( P(\text{neg}| w, v) = 1 - P(\text{pos}| w, v) \)

**Question:** Why not just maximize \( \prod_{(i,j) \in \text{pos}(O)} P(\text{pos}| w^{(i)}, v^{(j)}) \)?

- Trivial solution: make all \( w, v \) identical
Word2Vec with negative sampling as classification

- Maximize likelihood of training data:

\[
\mathcal{L}(\theta) = \prod_i P(y^{(i)}|x^{(i)}; \theta)
\]

- \iff minimize negative log likelihood:

\[
NLL(\theta) = -\log \mathcal{L}(\theta) = -\sum_i \log P(y^{(i)}|x^{(i)}; \theta))
\]

**Question:** What do these components stand for in Word2Vec with negative sampling?

- \(x^{(i)}\) Word pair, from corpus OR randomly created
- \(y^{(i)}\) Label: 1 = word pair is from positive training set, 0 = word pair is from negative training set
- \(\theta\) Parameters \(\mathbf{v}, \mathbf{w}\)
- \(P(\cdot)\) Logistic sigmoid: \(P(1|\cdot) = \sigma(\mathbf{w}^T \mathbf{v})\), resp. \(P(0|\cdot) = 1 - \sigma(\mathbf{w}^T \mathbf{v})\).
Stochastic Gradient Descent

\[
\begin{align*}
\frac{d\sigma(x)}{dx} &= \sigma(x)(1 - \sigma(x)) & \frac{d\log(x)}{dx} &= \frac{1}{x} \\
L(w, v, y) &= -y \log(\sigma(w^T v)) - (1 - y) \log(1 - \sigma(w^T v)) \\
\frac{\partial L}{\partial w} &= \\
&= -y \frac{1}{\sigma(w^T v)} \sigma(w^T v)(1 - \sigma(w^T v)) v \\
&\quad - (1 - y) \frac{1}{1 - \sigma(w^T v)} (-1)\sigma(w^T v)(1 - \sigma(w^T v)) v \\
&= (\sigma(w^T v) - y) v \\
\text{Same for } v: \quad \frac{\partial L}{\partial v} &= (\sigma(w^T v) - y) w
\end{align*}
\]
Stochastic Gradient Descent

- One update step for one word pair \( i, j \):
  \[
  v_{\text{updated}}^{(i)} \leftarrow v^{(i)} - \eta \left( \sigma (w^{(i)} T v^{(j)}) - y \right) w^{(j)}
  \]
  \[
  w_{\text{updated}}^{(j)} \leftarrow w^{(j)} - \eta \left( \sigma (w^{(i)} T v^{(j)}) - y \right) v^{(i)}
  \]

- \( \eta > 0 \) is learning rate, \( y \) is label \( \in \{0, 1\} \).
- When do the vectors of a pair become more/less similar, and why?
  - Let \( a = -\eta (\sigma (v^{(i)} T w^{(j)}) - y) \)
  - Positive (observed) word pair: \( y = 1 \Rightarrow a > 0 \).
    \* \( av^{(i)} \) is added to \( w^{(j)} \) and vice versa \( \rightarrow \) more similar.
  - Negative (random) word pair: \( y = 0 \Rightarrow a < 0 \).
    \* \( av^{(i)} \) is subtracted from \( w^{(j)} \) and vice versa \( \rightarrow \) less similar.

What does the update size \( a \) depend on (aside from \( \eta \))?

Absolute difference of \( y \) and \( \sigma (w^{(i)} T v^{(j)}) \)
Constructing a good negative training set can be difficult

Often it is some random perturbation of the training data (e.g. replacing the second word of each bigram by a random word).

The number of negative samples is often a multiple (1x to 20x) of the number of positive samples.

Negative sets are often constructed per batch.
**Question:** How many dot products do we need to calculate for a given word pair? How does this compare to the naive and hierarchical softmax?

- \( M + 1 \approx \log_2 |V| \ll |V| \)
- (for \( M = 20, |V| = 1,000,000 \))
Questions?
Skip-gram (Word2Vec)

- **Idea:** Learn many bigram language models at the same time.

- **Given word** $w_{[t]}$, predict words inside a window around $w_{[t]}$:
  - One position before the target word:
    $$p(w_{[t-1]}|w_{[t]})$$
  - One position after the target word:
    $$p(w_{[t+1]}|w_{[t]})$$
  - Two positions before the target word:
    $$p(w_{[t-2]}|w_{[t]})$$
  - ... up to a specified window size $c$.

- **Models share all** $w$, $v$ **parameters!**
Skip-gram: Objective

- Optimize the joint likelihood of the $2c$ language models:

$$p(w_{t-c} \ldots w_{t-1} w_{t+1} \ldots w_{t+c} | w_t) = \prod_{i \in \{-c \ldots c\}, i \neq 0} p(w_{t+i} | w_t)$$

- Negative Log-likelihood for whole corpus (of size $N$):

$$NLL = - \sum_{t=1}^{N} \sum_{i \in \{-c \ldots c\}, i \neq 0} \log p(w_{t+i} | w_t)$$

- Using negative sampling as approximation:

$$\approx - \sum_{t=1}^{N} \sum_{i \in \{-c \ldots c\}, i \neq 0} \left[ \log \sigma(w_{t+i}^T v_t) + \sum_{m=1}^{M} \log[1 - \sigma(w^{(*)T} v_t)] \right]$$

- $w^{(*)}$ is the word vector of a random word, $M$ is the number of negatives per positive sample
(ontinuous) Bag of Words

- Like Skipgram, but...
- Predict word $w[t]$, given the words inside the window around $w[t]$: 

$$ p(w[t] | w[t-c] \cdots w[t-1] w[t+1] \cdots w[t+c] ) $$

$$ \propto w_t^T \sum_{i \in -c \ldots c} v_{t+i} $$
./word2vec -train data.txt -output vec.txt
-window 5 -negative 20 -hs 0 -cbow 1
Questions?
Even if we train Word2Vec on a very large corpus, we will still encounter unknown words at test time.

Orthography can often help us:

- $w^{(\text{remuneration})}$ should be similar to
  - $w^{(\text{remunerate})}$ (same stem)
  - $w^{(\text{iteration})}, w^{(\text{consideration})}$ ... (same suffix $\approx$ same POS)
known word: \( w^{(i)} = \frac{1}{| \text{ngrams}(i) | + 1} \left[ u^{(i)} + \sum_{n \in \text{ngrams}(i)} u^{(n)} \right] \)

unknown word: \( w^{(i)} = \frac{1}{| \text{ngrams}(i) |} \sum_{n \in \text{ngrams}(i)} u^{(n)} \)

\[ \text{ngrams(remuneration)} = \{ \$re, rem, \$rem, \ldots ration, ation\} \]
FastText: Training

- n-grams typically contains 3- to 6-grams
- Replace \( \mathbf{w} \) in Skipgram objective with its new definition
- During backpropagation, loss gradient vector \( \frac{\partial J}{\partial \mathbf{w}(i)} \) is distributed to word vector \( \mathbf{u}^{(i)} \) and associated n-gram vectors \( \mathbf{u}^{(n)} \)
Summary

- Word2Vec as a bigram Language Model
- Hierarchical Softmax
- Negative Sampling
- Skipgram: Predict words in window given word in the middle
- CBOW: Predict word in the middle given words in window
- fastText: N-gram embeddings generalize to unseen words
- Any questions?
Initializing neural networks with pretrained embeddings

- Knowledge *transfer* from unlabelled corpus
- Design choice: Fine-tune embeddings on task or freeze them?
  - Pro: Can learn/strengthen features that are important for task
  - Contra: Training vocabulary is small subset of entire vocabulary → we might overfit and mess up topology w.r.t. unseen words

```python
pretrained = #load_some_embeddings()
frozen = Embedding(input_dim = pretrained.shape[0],
                      output_dim = pretrained.shape[1],
                      weights = [pretrained],
                      trainable = False)
finetunable = Embedding(input_dim = pretrained.shape[0],
                        output_dim = pretrained.shape[1],
                        weights = [pretrained],
                        trainable = True)
```

(keras)
Initializing neural networks with pretrained embeddings

<table>
<thead>
<tr>
<th>Model</th>
<th>MR</th>
<th>SST-1</th>
<th>SST-2</th>
<th>Subj</th>
<th>TREC</th>
<th>CR</th>
<th>MPQA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-rand (randomly initialized)</td>
<td>76.1</td>
<td>45.0</td>
<td>82.7</td>
<td>89.6</td>
<td>91.2</td>
<td>79.8</td>
<td>83.4</td>
</tr>
<tr>
<td>CNN-static (pretrained+frozen)</td>
<td>81.0</td>
<td>45.5</td>
<td>86.8</td>
<td>93.0</td>
<td>92.8</td>
<td>84.7</td>
<td>89.6</td>
</tr>
<tr>
<td>CNN-non-static (pretrained+fine-tuned)</td>
<td><strong>81.5</strong></td>
<td>48.0</td>
<td>87.2</td>
<td>93.4</td>
<td>93.6</td>
<td>84.3</td>
<td>89.5</td>
</tr>
<tr>
<td>CNN-multichannel (combination)</td>
<td>81.1</td>
<td>47.4</td>
<td><strong>88.1</strong></td>
<td>93.2</td>
<td>92.2</td>
<td><strong>85.0</strong></td>
<td>89.4</td>
</tr>
</tbody>
</table>

Table from Kim 2014: Convolutional Neural Networks for Sentence Classification.
Resources

  - Embeddings for 157 languages, trained on big web crawls, up to 2M words per language

- https://nlp.stanford.edu/projects/glove/
  - GloVe word vectors: Cooccurrence-count objective, not n-gram based
Analogy mining

country-capital

\[ \mathbf{w}^{(Tokio)} - \mathbf{w}^{(Japan)} + \mathbf{w}^{(Poland)} \approx \mathbf{w}^{(Warsaw)} \]

opposite

\[ \mathbf{w}^{(unacceptable)} - \mathbf{w}^{(acceptable)} + \mathbf{w}^{(logical)} \approx \mathbf{w}^{(illogical)} \]

Nationality-adjective

\[ \mathbf{w}^{(Australian)} - \mathbf{w}^{(Australia)} + \mathbf{w}^{(Switzerland)} \approx \mathbf{w}^{(Swiss)} \]
\[ w^{(a)} - w^{(b)} + w^{(c)} = w^{(d)} \]

\[ w^{(d)} = \arg\max_{w^{(d') \in W}} \cos(w^{(?)}, w^{(d')}) \]

Table 8: Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>France - Paris</td>
<td>Italy: Rome</td>
<td>Japan: Tokyo</td>
<td>Florida: Tallahassee</td>
</tr>
<tr>
<td>big - bigger</td>
<td>small: larger</td>
<td>cold: colder</td>
<td>quick: quicker</td>
</tr>
<tr>
<td>Miami - Florida</td>
<td>Baltimore: Maryland</td>
<td>Dallas: Texas</td>
<td>Kona: Hawaii</td>
</tr>
<tr>
<td>Einstein - scientist</td>
<td>Messi: midfielder</td>
<td>Mozart: violinist</td>
<td>Picasso: painter</td>
</tr>
<tr>
<td>Sarkozy - France</td>
<td>Berlusconi: Italy</td>
<td>Merkel: Germany</td>
<td>Koizumi: Japan</td>
</tr>
<tr>
<td>copper - Cu</td>
<td>zinc: Zn</td>
<td>Putin: Medvedev</td>
<td>uranium: plutonium</td>
</tr>
<tr>
<td>Berlusconi - Silvio</td>
<td>Sarkozy: Nicolas</td>
<td>IBM: Linux</td>
<td>Obama: Barack</td>
</tr>
<tr>
<td>Microsoft - Windows</td>
<td>Google: Android</td>
<td>IBM: McNealy</td>
<td>Apple: iPhone</td>
</tr>
<tr>
<td>Microsoft - Ballmer</td>
<td>Google: Yahoo</td>
<td>France: tapas</td>
<td>Apple: Jobs</td>
</tr>
<tr>
<td>Japan - sushi</td>
<td>Germany: bratwurst</td>
<td>USA: pizza</td>
<td>USA: pizza</td>
</tr>
</tbody>
</table>
Cross-lingual Embedding Spaces: A very short overview

- Embedding space: The space defined by the embeddings of all words in a language
- Hypothesis: Embedding spaces of different languages have similar structures

Mikolov et al. 2013: Exploiting Similarities among Languages for Machine Translation
Cross-lingual Embedding Spaces: A very short overview

- Given:
  - Monolingual embedding spaces of two languages: $\mathbf{W}_{L_1}$, $\mathbf{W}_{L_2}$
  - Dictionary $D$ of a few known translations
- Learn function $f$, s.t.
  \[ \forall (i,j) \in D f(\mathbf{w}_{L_1}^{(i)}) \approx \mathbf{w}_{L_2}^{(j)} \]

  - e.g., linear transformation: $f(\mathbf{w}_{L_1}) = \mathbf{Vw}_{L_1}$
- Given word $k$ in L1 with unknown translation:
  - translate as L2 word $l$ whose embedding $\mathbf{w}_{L_2}^{(l)}$ minimizes cosine distance to $f(\mathbf{w}_{L_1}^{(k)})$
- Used as initialization for unsupervised Machine Translation
Summary

- Applications of Word Embeddings:
- Word vector initialization in neural networks
- Analogy mining
- Word translation mining
- Any questions?