

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).

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

- **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:

$$\cos(\mathbf{w}^{(i)}, \mathbf{w}^{(j)}) = \frac{\mathbf{w}^{(i)T} \mathbf{w}^{(j)}}{\|\mathbf{w}^{(i)}\|_2 \cdot \|\mathbf{w}^{(j)}\|_2}$$

- ▶ 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 $\mathbf{w}^{(\text{great})} \approx \mathbf{w}^{(\text{awesome})}$
- **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:

$$P(w_{[1]} \dots w_{[T]}) = \prod_{t=1}^T P(w_{[t]} | w_{[t-1]} \dots w_{[t-n+1]})$$

Word2Vec as a Bigram Language Model

- 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!

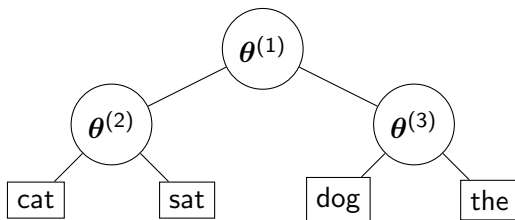
$$P(i|j) = \frac{\exp(\mathbf{w}^{(i)T} \mathbf{v}^{(j)})}{\sum_{k=1}^{|V|} \exp(\mathbf{w}^{(k)T} \mathbf{v}^{(j)})}$$

- **Question:** Problem with training softmax?
 - ▶ \Rightarrow Slow. Needs to compute dot products with the whole vocabulary for every single prediction.

Questions?

Speeding up Training: Hierarchical Softmax

- Context vectors \mathbf{v} are defined like before.
- Word vectors \mathbf{w} are replaced by a binary tree:

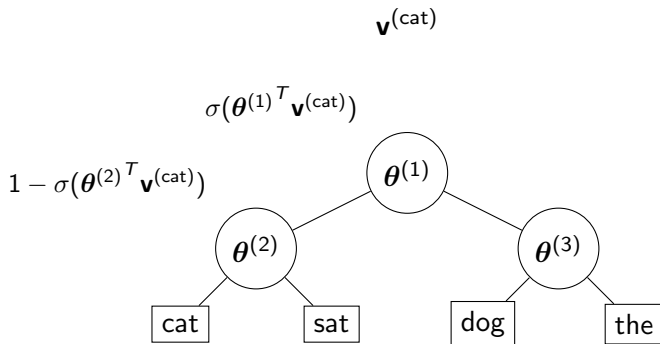


Hierarchical Softmax

- Each tree node l has parameter vector $\theta^{(l)}$
- Probability of going left at node l given context word j :
$$p(\text{left}|l, j) = \sigma(\theta^{(l)T} \mathbf{v}^{(j)})$$
- 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 $p(\text{sat}|\text{cat})$.



$$p(\text{sat}|\text{cat}) = \sigma(\theta^{(1)T} \mathbf{v}^{(\text{cat})})[1 - \sigma(\theta^{(2)T} \mathbf{v}^{(\text{cat})})]$$

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?

Speeding up Training: Negative Sampling

- 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: $\text{pos}(\mathcal{O})$
- ▶ Negative training set: $\text{neg}(\mathcal{O})$

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

- $P(\text{pos} | \mathbf{w}, \mathbf{v}) = \sigma(\mathbf{w}^T \mathbf{v})$
- $P(\text{neg} | \mathbf{w}, \mathbf{v}) = 1 - P(\text{pos} | \mathbf{w}, \mathbf{v})$
- **Question:** Why not just maximize $\prod_{(i,j) \in \text{pos}(\mathcal{O})} P(\text{pos} | \mathbf{w}^{(i)}, \mathbf{v}^{(j)})$?
 - ▶ Trivial solution: make all \mathbf{w}, \mathbf{v} identical

Word2Vec with negative sampling as classification

- Maximize likelihood of training data:

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

- \Leftrightarrow 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
 - ▶ θ Parameters \mathbf{v} , \mathbf{w}
 - ▶ $P(\dots)$ 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

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x)) \quad \frac{d\log(x)}{dx} = \frac{1}{x}$$

$$L(\mathbf{w}, \mathbf{v}, y) = -y \log(\sigma(\mathbf{w}^T \mathbf{v})) - (1 - y) \log(1 - \sigma(\mathbf{w}^T \mathbf{v}))$$

$$\frac{\partial L}{\partial \mathbf{w}} =$$

$$-y \frac{1}{\sigma(\mathbf{w}^T \mathbf{v})} \sigma(\mathbf{w}^T \mathbf{v})(1 - \sigma(\mathbf{w}^T \mathbf{v})) \mathbf{v}$$

$$- (1 - y) \frac{1}{1 - \sigma(\mathbf{w}^T \mathbf{v})} (-1) \sigma(\mathbf{w}^T \mathbf{v})(1 - \sigma(\mathbf{w}^T \mathbf{v})) \mathbf{v}$$

$$= (\sigma(\mathbf{w}^T \mathbf{v}) - y) \mathbf{v}$$

Same for \mathbf{v} :

$$\frac{\partial L}{\partial \mathbf{v}} = (\sigma(\mathbf{w}^T \mathbf{v}) - y) \mathbf{w}$$

Stochastic Gradient Descent

- One update step for one word pair i, j :

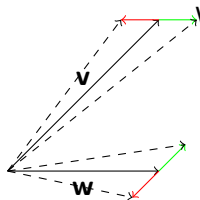
$$\mathbf{v}_{\text{updated}}^{(i)} \leftarrow \mathbf{v}^{(i)} - \eta \left(\sigma(\mathbf{w}^{(i)T} \mathbf{v}^{(j)}) - y \right) \mathbf{w}^{(j)}$$

$$\mathbf{w}_{\text{updated}}^{(j)} \leftarrow \mathbf{w}^{(j)} - \eta \left(\sigma(\mathbf{w}^{(i)T} \mathbf{v}^{(j)}) - y \right) \mathbf{v}^{(i)}$$

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

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

Absolute difference of y and $\sigma(\mathbf{w}^{(i)T} \mathbf{v}^{(j)})$



Speeding up Training: Negative Sampling

- 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

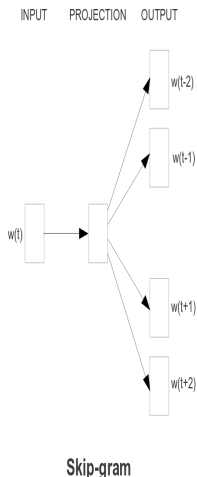
Questions

- **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 \mathbf{w} , \mathbf{v} parameters!



Skip-gram: Objective

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

$$p(w_{[t-c]} \dots w_{[t-1]} w_{[t+1]} \dots w_{[t+c]} | w_{[t]}) = \prod_{\substack{i \in \{-c \dots c\} \\ i \neq 0}} p(w_{[t+i]} | w_{[t]})$$

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

$$NLL = - \sum_{t=1}^N \sum_{\substack{i \in \{-c \dots c\} \\ i \neq 0}} \log p(w_{[t+i]} | w_{[t]})$$

- Using negative sampling as approximation:

$$\approx - \sum_{t=1}^N \sum_{\substack{i \in \{-c \dots c\} \\ i \neq 0}} \left[\log \sigma(\mathbf{w}_{[t+i]}^T \mathbf{v}_{[t]}) + \sum_{m=1}^M \log [1 - \sigma(\mathbf{w}^{(*)T} \mathbf{v}_{[t]})] \right]$$

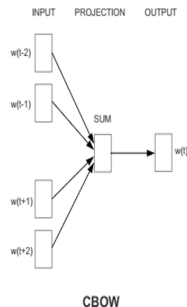
- $\mathbf{w}^{(*)}$ is the word vector of a random word, M is the number of negatives per positive sample

C(ontinuous) B(ag) o(f) W(ords)

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

$$p(w_{[t]} | w_{[t-c]} \dots w_{[t-1]} w_{[t+1]} \dots w_{[t+c]})$$

$$\propto \mathbf{w}_{[t]}^T \sum_{\substack{i \in -c \dots c \\ i \neq 0}} \mathbf{v}_{[t+i]}$$



```
./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:
- $\mathbf{w}^{(\text{remuneration})}$ should be similar to
 - ▶ $\mathbf{w}^{(\text{remunerate})}$ (same stem)
 - ▶ $\mathbf{w}^{(\text{iteration})}, \mathbf{w}^{(\text{consideration})} \dots$ (same suffix \approx same POS)

$$\text{known word: } \mathbf{w}^{(i)} = \frac{1}{|\text{ngrams}(i)| + 1} \left[\mathbf{u}^{(i)} + \sum_{n \in \text{ngrams}(i)} \mathbf{u}^{(n)} \right]$$

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

$$\text{ngrams}(\text{remuneration}) = \{\$re, rem, \$rem, \dots \text{ration}, \text{ation}\$ \}$$

FastText: Training

- ngrams 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

```
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

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

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

Resources

- <https://fasttext.cc/docs/en/crawl-vectors.html>
 - ▶ 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

$$\mathbf{w}^{(\text{Tokio})} - \mathbf{w}^{(\text{Japan})} + \mathbf{w}^{(\text{Poland})} \approx \mathbf{w}^{(\text{Warsaw})}$$

opposite

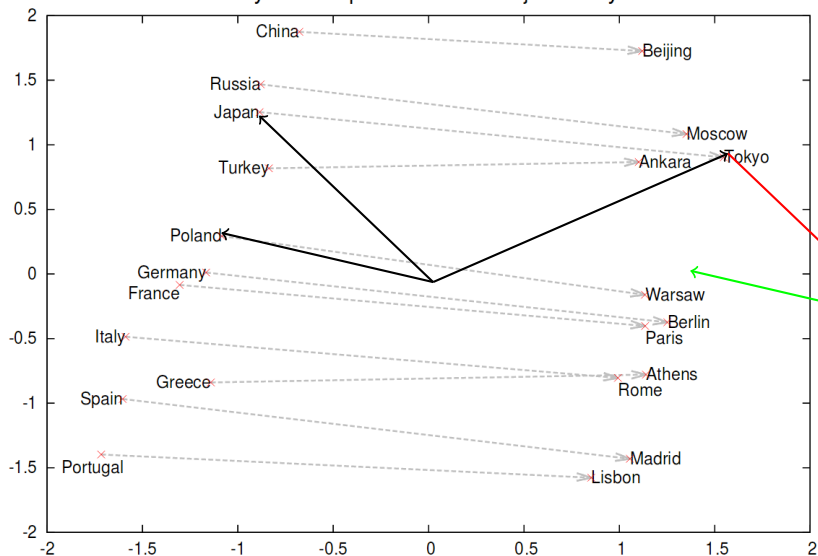
$$\mathbf{w}^{(\text{unacceptable})} - \mathbf{w}^{(\text{acceptable})} + \mathbf{w}^{(\text{logical})} \approx \mathbf{w}^{(\text{illogical})}$$

Nationality-adjective

$$\mathbf{w}^{(\text{Australian})} - \mathbf{w}^{(\text{Australia})} + \mathbf{w}^{(\text{Switzerland})} \approx \mathbf{w}^{(\text{Swiss})}$$

Analogy mining

Country and Capital Vectors Projected by PCA



$$\mathbf{w}^{(a)} - \mathbf{w}^{(b)} + \mathbf{w}^{(c)} = \mathbf{w}^{(?)}$$

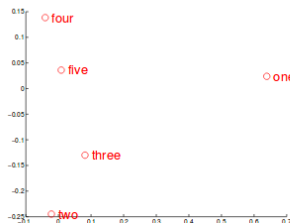
$$\mathbf{w}^{(d)} = \operatorname{argmax}_{\mathbf{w}^{(d')} \in \mathbf{W}} \cos(\mathbf{w}^{(?)}, \mathbf{w}^{(d')})$$

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).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

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}_{L1} , \mathbf{W}_{L2}
 - ▶ Dictionary D of a few known translations
- Learn function f , s.t.

$$\forall (i,j) \in D f(\mathbf{w}_{L1}^{(i)}) \approx \mathbf{w}_{L2}^{(j)}$$

- ▶ e.g., linear transformation: $f(\mathbf{w}_{L1}) = \mathbf{V}\mathbf{w}_{L1}$
- Given word k in L1 with unknown translation:
 - ▶ translate as L2 word l whose embedding $\mathbf{w}_{L2}^{(l)}$ minimizes cosine distance to $f(\mathbf{w}_{L1}^{(k)})$
- 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?