

Word Embeddings



Overview

- Representing meaning
- Word embeddings: Early work
- Word embeddings via language models
- Word2vec and Glove
- Evaluating embeddings
- Design choices and open questions

Overview

- Representing meaning
- Word embeddings: Early work
- Word embeddings via language models
- Word2vec and Glove
- Evaluating embeddings
- Design choices and open questions

Learning word embeddings

- Bengio et al 2003: Define a neural language model that embedded words along the way
- Collobert & Weston 2008: Showed that word embeddings can actually help many NLP tasks
- Mikolov 2013: word2vec
 - Two families of widely used models

A brief detour: Language models

Given a sequence of words so far $(w_1, w_2, \dots, w_{n-1})$, what is the probability of the next word w_n ?

– Eg: $w_1, w_2, \dots, w_{n-1} = \text{Once upon a ...}$

A brief detour: Language models

Given a sequence of words so far $(w_1, w_2, \dots, w_{n-1})$, what is the probability of the next word w_n ?

– Eg: $w_1, w_2, \dots, w_{n-1} = \text{Once upon a ...}$

$$P(w_n \mid w_1, w_2, \dots, w_{n-1})$$

A brief detour: Language models

Given a sequence of words so far $(w_1, w_2, \dots, w_{n-1})$, what is the probability of the next word w_n ?

– Eg: $w_1, w_2, \dots, w_{n-1} = \text{Once upon a ...}$

$$P(w_n | w_1, w_2, \dots, w_{n-1})$$

Before neural networks, this involved counting

$$P(w_n | w_1, w_2, \dots, w_{n-1}) = \frac{\text{count}(w_1, w_2, \dots, w_{n-1}, w_n)}{\text{count}(w_1, w_2, \dots, w_{n-1})}$$

A brief detour: Language models

Given a sequence of words so far $(w_1, w_2, \dots, w_{n-1})$, what is the probability of the next word w_n ?

– Eg: $w_1, w_2, \dots, w_{n-1} = \text{Once upon a ...}$

$$P(w_n | w_1, w_2, \dots, w_{n-1})$$

Before neural networks, this involved counting

$$P(w_n | w_1, w_2, \dots, w_{n-1}) = \frac{\text{count}(w_1, w_2, \dots, w_{n-1}, w_n)}{\text{count}(w_1, w_2, \dots, w_{n-1})}$$

Typically there are many ways to smooth this distribution

Eg, five-gram models ($n=5$), with Kneser Ney smoothing

A brief detour: Language models

Given a sequence of words so far $(w_1, w_2, \dots, w_{n-1})$, what is the probability of the next word w_n ?

– Eg: $w_1, w_2, \dots, w_{n-1} = \text{Once upon a ...}$

$$P(w_n | w_1, w_2, \dots, w_{n-1})$$

Bengio et al 2003: What if this probability is defined by a neural network?

A brief detour: Language models

Given a sequence of words so far $(w_1, w_2, \dots, w_{n-1})$, what is the probability of the next word w_n ?

– Eg: $w_1, w_2, \dots, w_{n-1} = \text{Once upon a ...}$

$$P(w_n | w_1, w_2, \dots, w_{n-1})$$

With neural networks, we can define this probability as

$$P(w_n | w_1, w_2, \dots, w_{n-1}) = \text{softmax}(f(\mathbf{w}_1, \dots, \mathbf{w}_n))$$

A brief detour: Language models

Given a sequence of words so far $(w_1, w_2, \dots, w_{n-1})$, what is the probability of the next word w_n ?

– Eg: $w_1, w_2, \dots, w_{n-1} = \text{Once upon a ...}$

$$P(w_n | w_1, w_2, \dots, w_{n-1})$$

With neural networks, we can define this probability as

$$P(w_n | w_1, w_2, \dots, w_{n-1}) = \text{softmax}(f(\mathbf{w}_1, \dots, \mathbf{w}_n))$$

$\mathbf{w}_1, \dots, \mathbf{w}_n$ are vectors for each word that are learned by backpropagation

A brief detour: Language models

Given a sequence of words so far $(w_1, w_2, \dots, w_{n-1})$, what is the probability of the next word w_n ?

– Eg: $w_1, w_2, \dots, w_{n-1} = \text{Once upon a ...}$

$$P(w_n | w_1, w_2, \dots, w_{n-1})$$

With neural networks, we can define this probability as

$$P(w_n | w_1, w_2, \dots, w_{n-1}) = \text{softmax}(f(\mathbf{w}_1, \dots, \mathbf{w}_n))$$

$\mathbf{w}_1, \dots, \mathbf{w}_n$ are vectors for each word that are learned by backpropagation

Many variants on this theme – e.g.,
left and right context could be involved

Training a neural language model

Given a sentence w_1, w_2, \dots, w_T , we can write its probability as

$$P(w_1, w_2, \dots, w_T) = \prod P(w_t \mid w_{t-1}, \dots, w_{t-n+1})$$

This gives us a natural definition for log loss

Training a neural language model

Given a sentence w_1, w_2, \dots, w_T , we can write its probability as

$$P(w_1, w_2, \dots, w_T) = \prod P(w_t \mid w_{t-1}, \dots, w_{t-n+1})$$

The language model, in this case, a neural network

This gives us a natural definition for log loss

Training a neural language model

Given a sentence w_1, w_2, \dots, w_T , we can write its probability as

$$P(w_1, w_2, \dots, w_T) = \prod P(w_t \mid w_{t-1}, \dots, w_{t-n+1})$$

This gives us a natural definition for log loss

$$J(\text{params}) = \sum \log P(w_t \mid w_{t-1}, \dots, w_{t-n+1})$$

Training a neural language model

Given a sentence w_1, w_2, \dots, w_T , we can write its probability as

$$P(w_1, w_2, \dots, w_T) = \prod P(w_t \mid w_{t-1}, \dots, w_{t-n+1})$$

This gives us a natural definition for log loss

$$J(\text{params}) = \sum \log P(w_t \mid w_{t-1}, \dots, w_{t-n+1})$$

One question left: What is a good neural network architecture for this problem?

Neural network language models

- A multi-layer neural network [Bengio et al 2003]
 - Words → embedding layer → hidden layers → softmax
 - Cross-entropy loss

Neural network language models

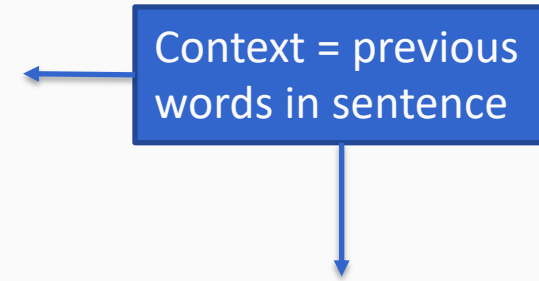
- A multi-layer neural network [Bengio et al 2003]
 - Words → embedding layer → hidden layers → softmax
 - Cross-entropy loss
- Instead of producing probability, just produce a score for the next word (no softmax) [Collobert and Weston, 2008]
 - Ranking loss
 - Intuition: Valid word sequences should get a higher score than invalid ones

Neural network language models

- A multi-layer neural network [Bengio et al 2003]
 - Words → embedding layer → hidden layers → softmax
 - Cross-entropy loss
- Instead of producing probability, just produce a score for the next word (no softmax) [Collobert and Weston, 2008]
 - Ranking loss
 - Intuition: Valid word sequences should get a higher score than invalid ones
- No need for a multi-layer network, a shallow network is good enough [Mikolov, 2013, word2vec]
 - Simpler model, fewer parameters
 - Faster to train

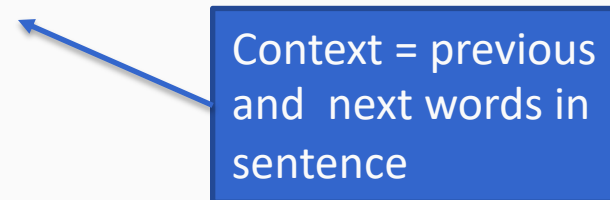
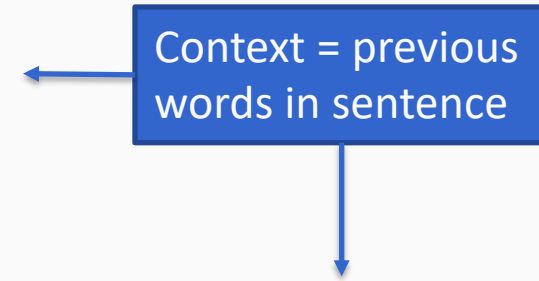
Neural network language models

- A multi-layer neural network [Bengio et al 2003]
 - Words → embedding layer → hidden layers → softmax
 - Cross-entropy loss
- Instead of producing probability, just produce a score for the next word (no softmax) [Collobert and Weston, 2008]
 - Ranking loss
 - Intuition: Valid word sequences should get a higher score than invalid ones
- No need for a multi-layer network, a shallow network is good enough [Mikolov, 2013, word2vec]
 - Simpler model, fewer parameters
 - Faster to train



Neural network language models

- A multi-layer neural network [Bengio et al 2003]
 - Words → embedding layer → hidden layers → softmax
 - Cross-entropy loss
- Instead of producing probability, just produce a score for the next word (no softmax) [Collobert and Weston, 2008]
 - Ranking loss
 - Intuition: Valid word sequences should get a higher score than invalid ones
- No need for a multi-layer network, a shallow network is good enough [Mikolov, 2013, word2vec]
 - Simpler model, fewer parameters
 - Faster to train



Coming up...

- The word2vec models: Skipgram and CBOW
- Connection between word2vec and matrix factorization
- Glove
- Evaluating word embeddings