Word Embeddings

THE UNIVERSITY OF UTAH

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

Word embeddings via language models

The goal: To find vector embeddings of words

High level approach:

1. Train a model for a surrogate task (in this case language modeling)
2. Word embeddings are a byproduct of this process

Neural network language models

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

Context = previous words in sentence

- 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

Context = previous and next words in sentence

This lecture

- The word2vec models: CBOW and Skipgram
- Connection between word2vec and matrix factorization
- GloVe

Word2Vec

[Mikolov et al ICLR 2013, Mikolov et al NIPS 2013]

- Two architectures for learning word embeddings
  - Skipgram and CBOW

- Both have two key differences from the older Bengio/C&W approaches
  1. No hidden layers
  2. Extra context (both left and right context)

- Several computational tricks to make things faster

This lecture

- The word2vec models: CBOW and Skipgram

- Connection between word2vec and matrix factorization

- GloVe

Continuous Bag of Words (CBOW)

Given a window of words of a length 2m + 1

- Call them: $x_{-m}, \cdots, x_{-1}\ x_0 x_1, \cdots, x_m$

Continuous Bag of Words (CBOW)

Given a window of words of a length 2m + 1

Call them: $x_{-m}, \cdots, x_{-1}\ x_0 x_1, \cdots, x_m$

Define a probabilistic model for predicting the middle word

$$P(x_0 \mid x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m)$$

Continuous Bag of Words (CBOW)

Given a window of words of a length 2m + 1

Call them: $x_{-m}, \cdots, x_{-1}\ x_0 x_1, \cdots, x_m$

Define a probabilistic model for predicting the middle word

$$P(x_0 \mid x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m)$$

Train the model by minimizing loss over the dataset

$$L = -\sum \log P(x_0 \mid x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m)$$

Continuous Bag of Words (CBOW)

Given a window of words of a length 2m + 1

Call them: $x_{-m}, \cdots, x_{-1}\ x_0 x_1, \cdots, x_m$

Define a probabilistic model for predicting the middle word

$$P(x_0 \mid x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m)$$

Need to define this to complete the model

Train the model by minimizing loss over the dataset

$$L = -\sum \log P(x_0 \mid x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m)$$

The CBOW model

The classification task

- Input: context words $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
- Output: the center word x_0
- These words correspond to one-hot vectors
  - Eg: `cat` would be associated with a dimension, its one-hot vector has 1 in that dimension and zero everywhere else

Notation:

- n: the embedding dimension (eg 300)
- V: The vocabulary of words we want to embed

Define two matrices:

1. $\mathcal{V}$: a matrix of size $n \times |V|$
2. $\mathcal{W}$: a matrix of size $|V| \times n$

The CBOW model

Input: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
Output: the center word x_0
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

Input: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
Output: the center word x_0
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

The CBOW model

1. Map all the context words into the n dimensional space using $\mathcal{V}$
   - We get 2m vectors $\mathcal{V}x_{-m}, \cdots, \mathcal{V}x_{-1}, \mathcal{V}x_1, \cdots, \mathcal{V}x_m$

The CBOW model

Input: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
Output: the center word x_0
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

1. Map all the context words into the n dimensional space using $\mathcal{V}$
   - We get 2m vectors $\mathcal{V}x_{-m}, \cdots, \mathcal{V}x_{-1}, \mathcal{V}x_1, \cdots, \mathcal{V}x_m$

2. Average these vectors to get a context vector

$$\hat{v} = \frac{1}{2m} \sum_{i=-m, i \neq 0}^{m} \mathcal{V}x_i$$

The CBOW model

Input: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
Output: the center word x_0
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

1. Map all the context words into the n dimensional space using $\mathcal{V}$
   – We get 2m vectors $\mathcal{V}x_{-m}, \cdots, \mathcal{V}x_{-1}, \mathcal{V}x_1, \cdots, \mathcal{V}x_m$

2. Average these vectors to get a context vector

$$\hat{v} = \frac{1}{2m} \sum_{i=-m, i\neq 0}^{m} \mathcal{V}x_i$$

3. Use this to compute a score vector for the output

$$score = \mathcal{W}\hat{v}$$

The CBOW model

Input: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
Output: the center word x_0
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

1. Map all the context words into the n dimensional space using $\mathcal{V}$
   - We get 2m vectors $\mathcal{V}x_{-m}, \cdots, \mathcal{V}x_{-1}, \mathcal{V}x_1, \cdots, \mathcal{V}x_m$

2. Average these vectors to get a context vector

$$\hat{v} = \frac{1}{2m} \sum_{i=-m, i \neq 0}^{m} \mathcal{V}x_i$$

3. Use this to compute a score vector for the output

$$score = \mathcal{W}\hat{v}$$

4. Use the score to compute probability via softmax

$$P(x_0 = \cdot \mid context) = softmax(\mathcal{W}\hat{v})$$

Input: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
Output: the center word x_0
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

The CBOW model

1. Map all the context words into the n dimensional space using $\mathcal{V}$
   – We get 2m vectors $\mathcal{V}x_{-m}, \cdots, \mathcal{V}x_{-1}, \mathcal{V}x_1, \cdots, \mathcal{V}x_m$

2. Average these vectors to get a context vector

$$\hat{v} = \frac{1}{2m} \sum_{i=-m, i\neq 0}^{m} \mathcal{V}x_i$$

Exercise: Write this as a computation graph

3. Use this to compute a score vector for the output

$$score = \mathcal{W}\hat{v}$$

4. Use the score to compute probability via softmax

$$P(x_0 = \cdot \mid context) = softmax(\mathcal{W}\hat{v})$$

The CBOW model

Input: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
Output: the center word x_0
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

1. Map all the context words into the n dimensional space using $\mathcal{V}$
   – We get 2m vectors $\mathcal{V}x_{-m}, \cdots, \mathcal{V}x_{-1}, \mathcal{V}x_1, \cdots, \mathcal{V}x_m$

2. Average these vectors to get a context vector

$$\hat{v} = \frac{1}{2m} \sum_{i=-m, i \neq 0}^{m} \mathcal{V}x_i$$

Word embeddings = Rows of the matrix corresponding to the output. That is, rows of $\mathcal{W}$

3. Use this to compute a score vector for the output

$$score = \mathcal{W}\hat{v}$$

4. Use the score to compute probability via softmax

$$P(x_0 = \cdot \mid context) = softmax(\mathcal{W}\hat{v})$$

The CBOW loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

The CBOW loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

- Step 1: Project a, b, d, e using the matrix $\mathcal{V}$. This gives us rows of the matrix: v_a, v_b, v_d, v_e.

The CBOW loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

- Step 1: Project a, b, d, e using the matrix $\mathcal{V}$. This gives us rows of the matrix: v_a, v_b, v_d, v_e.

- Step 2: Their average:

$$\hat{v} = \frac{v_a + v_b + v_c + v_d}{4}$$

The CBOW loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

- Step 1: Project a, b, d, e using the matrix $\mathcal{V}$. This gives us rows of the matrix: v_a, v_b, v_d, v_e.

- Step 2: Their average:

$$\hat{v} = \frac{v_a + v_b + v_c + v_d}{4}$$

- Step 3: The score = $\mathcal{W}\hat{v}$
  - Each element of this score corresponds to the score for a single word.

The CBOW loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

- Step 1: Project a, b, d, e using the matrix $\mathcal{V}$. This gives us rows of the matrix: v_a, v_b, v_d, v_e.

- Step 2: Their average:

$$\hat{v} = \frac{v_a + v_b + v_c + v_d}{4}$$

- Step 3: The score = $\mathcal{W}\hat{v}$
  - Each element of this score corresponds to the score for a single word.

- Step 4: the probability of a word being the center word

$$P(\cdot \mid a, b, d, e) = \text{softmax}(\mathcal{W}\hat{v})$$

The CBOW loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

- Step 4: the probability of a word being the center word

$$P(\cdot \mid a, b, d, e) = \text{softmax}(\mathcal{W}\hat{v})$$

The CBOW loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

- Step 4: the probability of a word being the center word

$$P(\cdot \mid a, b, d, e) = \mathrm{softmax}(\mathcal{W}\hat{v})$$

More concretely:

$$P(c \mid a, b, d, e) = \frac{\exp(w_c^T \hat{v})}{\sum_{i=1}^{|V|} \exp(w_i^T \hat{v})}$$

The CBOW loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

- Step 4: the probability of a word being the center word

$$P(\cdot \mid a, b, d, e) = \text{softmax}(\mathcal{W}\hat{v})$$

More concretely:

$$P(c \mid a, b, d, e) = \frac{\exp(w_c^T \hat{v})}{\sum_{i=1}^{|V|} \exp(w_i^T \hat{v})}$$

The loss requires the negative log of this quantity.

$$Loss = -w_c^T \hat{v} + \log \sum_{i=1}^{|V|} \exp(w_i^T \hat{v})$$

The CBOW loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

- Step 4: the probability of a word being the center word

$$P(\cdot | \, a, b, d, e\,) = \text{softmax}(\mathcal{W}\hat{v})$$

More concretely:

$$P(c\,|a, b, d, e) = \frac{\exp(w_c^T\hat{v})}{\sum_{i=1}^{|V|}\exp(w_i^T\hat{v})}$$

The loss requires the negative log of this quantity.

$$Loss = -w_c^T\hat{v} + \log\sum_{i=1}^{|V|}\exp(w_i^T\hat{v})$$

Exercise: Calculate the derivative of this with respect to all the w's and the v's

The CBOW loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

- Step 4: the probability of a word being the center word

$$P(\cdot|\ a, b, d, e\ ) = \mathrm{softmax}(\mathcal{W}\hat{v})$$

More concretely:

$$P(c\ |a, b, d, e) = \frac{\exp(w_c^T \hat{v})}{\sum_{i=1}^{|V|} \exp(w_i^T \hat{v})}$$

The loss requires the negative log of this quantity.

$$Loss = -w_c^T \hat{v} + \log \sum_{i=1}^{|V|} \exp(w_i^T \hat{v})$$

Note that this sum requires us to iterate over the entire vocabulary for each example!

Exercise: Calculate the derivative of this with respect to all the w's and the v's

This lecture

- The word2vec models: CBOW and **Skipgram**

- Connection between word2vec and matrix factorization

- GloVe

Skipgram

The other word2vec model

Given a window of words of a length 2m + 1

Call them: $x_{-m}, \cdots, x_{-1}\ x_0 x_1, \cdots, x_m$

Skipgram

Given a window of words of a length 2m + 1

Call them: $x_{-m}, \cdots, x_{-1}\ x_0 x_1, \cdots, x_m$

Define a probabilistic model for predicting each context word

$$P(x_{context} \mid x_0)$$

Skipgram

Given a window of words of a length 2m + 1

Call them: $x_{-m}, \cdots, x_{-1}\, x_0 x_1, \cdots, x_m$

Define a probabilistic model for predicting each context word

$$P(x_{context} \mid x_0)$$

Inverts the inputs and outputs from CBOW

As far as the probabilistic model is concerned:
Input: the center word
Output: all the words in the context

The Skipgram model

The classification task

- Input: the center word x_0
- Output: context words $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
- As before, these words correspond to one-hot vectors

Notation:

- n: the embedding dimension (eg 300)
- V: The vocabulary of words we want to embed

Define two matrices:

1. $\mathcal{V}$: a matrix of size $n \times |V|$
2. $\mathcal{W}$: a matrix of size $|V| \times n$

The Skipgram model

Input: the center word x_0

Output: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$

n: the embedding dimension (eg 300)

V: The vocabulary

$\mathcal{V}$: a matrix of size $n \times |V|$

$\mathcal{W}$: a matrix of size $|V| \times n$

Input: the center word x_0
Output: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

The Skipgram model

1. Map the center words into the n-dimensional space using $\mathcal{W}$
   – We get an n dimensional vector $w = \mathcal{W} x_0$

The Skipgram model

Input: the center word x_0
Output: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

1. Map the center words into the n-dimensional space using $\mathcal{W}$
   – We get an n dimensional vector $w = \mathcal{W}x_0$

2. For the i^{th} context position, compute the score for a word occupying that position as

$$v_i = \mathcal{V}w$$

The Skipgram model

Input: the center word x_0
Output: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

1. Map the center words into the n-dimensional space using $\mathcal{W}$
   – We get an n dimensional vector $w = \mathcal{W} x_0$

2. For the i^{th} context position, compute the score for a word occupying that position as

$$v_i = \mathcal{V} w$$

3. Normalize the score for each position to get a probability

$$P(x_i = \cdot \,|x_0) = \text{softmax}(v_i)$$

The Skipgram model

Input: the center word x_0
Output: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

1. Map the center words into the n-dimensional space using $\mathcal{W}$
   – We get an n dimensional vector $w = \mathcal{W} x_0$

2. For the i^{th} context position, compute the score for a word occupying that position as

$$v_i = \mathcal{V} w$$

3. Normalize the score for each position to get a probability

$$P(x_i = \cdot \mid x_0) = \text{softmax}(v_i)$$

Exercise: Write this as a computation graph

Input: the center word x_0
Output: context $x_{-m}, \cdots, x_{-1}, x_1, \cdots, x_m$
n: the embedding dimension (eg 300)
V: The vocabulary
$\mathcal{V}$: a matrix of size $n \times |V|$
$\mathcal{W}$: a matrix of size $|V| \times n$

The Skipgram model

1. Map the center words into the n-dimensional space using $\mathcal{W}$
   - We get an n dimensional vector $w = \mathcal{W} x_0$

2. For the i^{th} context position, compute the score for a word occupying that position as

$$v_i = \mathcal{V} w$$

3. Normalize the score for each position to get a probability

$$P(x_i = \cdot \mid x_0) = \text{softmax}(v_i)$$

Remember the goal of learning: Make this probability highest for the observed words in this context.

The Skipgram loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

The Skipgram loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

Step 1: Get the vector $w_c = \mathcal{W}c$

The Skipgram loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

Step 1: Get the vector $w_c = \mathcal{W}c$

Step 2: For every position compute the score for a word occupying that position as $v = \mathcal{V}w_c$

The Skipgram loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

Step 1: Get the vector $w_c = \mathcal{W}c$

Step 2: For every position compute the score for a word occupying that position as $v = \mathcal{V}w_c$

Step 3: Normalize the score for each position using softmax

$$P(\, x_i = \cdot \mid x_0 = c\,) = \text{softmax}(v)$$

The Skipgram loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

Step 1: Get the vector $w_c = \mathcal{W}c$

Step 2: For every position compute the score for a word occupying that position as $v = \mathcal{V}w_c$

Step 3: Normalize the score for each position using softmax

$$P(\, x_i = \cdot \mid x_0 = c\,) = \text{softmax}(v)$$

Or more specifically:

$$P(\, x_{-2} = a \mid x_0 = c\,) = \frac{\exp(v_a^T w_c)}{\sum_{i=1}^{|V|} \exp(v_i^T w_c)}$$

The Skipgram loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

Step 3: Normalize the score for each position using softmax

$$P(\, x_{-2} = a \mid x_0 = c \,) = \frac{\exp(v_a^T w_c)}{\sum_{i=1}^{|V|} \exp(v_i^T w_c)}$$

The Skipgram loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

Step 3: Normalize the score for each position using softmax

$$P(\, x_{-2} = a \mid x_0 = c \,) = \frac{\exp(v_a^T w_c)}{\sum_{i=1}^{|V|} \exp(v_i^T w_c)}$$

The loss for this example is the sum of the negative log of this over all the context words.

$$Loss = \sum_{k \in \{a,b,d,e\}} \left(-v_k^T w_c + \log \sum_{i=1}^{|V|} \exp\left(v_i^T w_c\right) \right)$$

The Skipgram loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

Step 3: Normalize the score for each position using softmax

$$P(\, x_{-2} = a \mid x_0 = c \,) = \frac{\exp(v_a^T w_c)}{\sum_{i=1}^{|V|} \exp(v_i^T w_c)}$$

The loss for this example is the sum of the negative log of this over all the context words.

$$Loss = \sum_{k \in \{a,b,d,e\}} \left(-v_k^T w_c + \log \sum_{i=1}^{|V|} \exp\left(v_i^T w_c\right) \right)$$

Exercise: Calculate the derivative of this with respect to all the w's and the v's

The Skipgram loss: A worked example

Consider the loss for one example with context size 2 on each side.
Denote the words by a b **c** d e with **c** being the output

Step 3: Normalize the score for each position using softmax

$$P(\, x_{-2} = a \mid x_0 = c\,) = \frac{\exp(v_a^T w_c)}{\sum_{i=1}^{|V|} \exp(v_i^T w_c)}$$

The loss for this example is the sum of the negative log of this over all the context words.

$$Loss = \sum_{k \in \{a,b,d,e\}} \left(-v_k^T w_c + \log \sum_{i=1}^{|V|} \exp\left(v_i^T w_c\right) \right)$$

Note that this sum requires us to iterate over the entire vocabulary for each example!

Exercise: Calculate the derivative of this with respect to all the w's and the v's

Negative sampling

$$\log \sum_{i=1}^{|V|} \exp\left(v_i^T w_c\right)$$

This sum requires us to iterate over the entire vocabulary for each example!

- Can we make it faster?

- Answer [Mikolov et al 2013]: change the objective function and define a new objective function that does not have the same problem
  - Negative Sampling

- The overall method is called Skipgram with Negative Sampling (SGNS)

Negative sampling: The intuition

- A new task: Given a pair of words (w, c), is this a valid pair or not?
  That is, can word c occur in the context window of w or not?

Negative sampling: The intuition

- A new task: Given a pair of words (w, c), is this a valid pair or not?
  That is, can word c occur in the context window of w or not?

- This is a binary classification problem
  - We can solve this using logistic regression
  - The probability of a pair of words being valid is defined as

$$P(\, c \mid w\,) = \sigma(v_c^T w_w) = \frac{1}{1 + \exp(-v_c^T w_w)}$$

Negative sampling: The intuition

- A new task: Given a pair of words (w, c), is this a valid pair or not?
  That is, can word c occur in the context window of w or not?

- This is a binary classification problem
  - We can solve this using logistic regression
  - The probability of a pair of words being valid is defined as

$$P(c \mid w) = \sigma(v_c^T w_w) = \frac{1}{1 + \exp(-v_c^T w_w)}$$

- Positive examples are all pairs that occur in data, negative examples are all pairs that don't occur in data, but this is still a massive set!

Negative sampling: The intuition

- A new task: Given a pair of words (w, c), is this a valid pair or not?
  - That is, can word c occur in the context window of w or not?

- This is a binary classification problem
  - We can solve this using logistic regression
  - The probability of a pair of words being valid is defined as

$$P(c \mid w) = \sigma(v_c^T w_w) = \frac{1}{1 + \exp(-v_c^T w_w)}$$

- Positive examples are all pairs that occur in data, negative examples are all pairs that don't occur in data, but this is still a massive set!

- Key insight: Instead of generating all possible negative examples, randomly sample k of them in each epoch of the learning loop
  - That is, there are only k negatives for each positive example, instead of the entire vocabulary

Negative sampling: The intuition

- A new task: Given a pair of words (w, c), is this a valid pair or not?
  - That is, can word c occur in the context window of w or not?
- This is a binary classification problem
  - We can solve this using logistic regression
  - The probability of a pair of words being valid is defined as

$$P(c \mid w) = \sigma(v_c^T w_w) = \frac{1}{1 + \exp(-v_c^T w_w)}$$

- Positive examples are all pairs that occur in data, negative examples are all pairs that don't occur in data, but this is still a massive set!
- Key insight: Instead of generating all possible negative examples, randomly sample k of them in each epoch of the learning loop
  - That is, there are only k negatives for each positive example, instead of the entire vocabulary

Word2vec notes

There are many other tricks that are needed to make this work and scale

- A scaling term in the loss function to ensure that frequent words do not dominate the loss
- Hierarchical softmax if you don’t want to use negative sampling
- A clever learning rate schedule
- Very efficient code

See readings for more details

This lecture

- The word2vec models: CBOW and Skipgram
- Connection between word2vec and matrix factorization
- GloVe

Recall: matrix factorization for embeddings

The general agenda

1. Construct a matrix word-word M whose entries are some function extracted from data involving words in context (e.g., counts, normalized counts, etc)
2. Factorize the matrix using SVD to produce lower dimensional embeddings of the words
3. Use one of the resulting matrices as word embeddings
   - Or some combination thereof

Word2vec and matrix factorization

[Levy and Goldberg, NIPS 2014]: Skipgram negative sampling is implicitly factorizing a specific matrix of this kind

Word2vec and matrix factorization

[Levy and Goldberg, NIPS 2014]: Skipgram negative sampling is implicitly factorizing a specific matrix of this kind

Two key points to note:

Word2vec and matrix factorization

[Levy and Goldberg, NIPS 2014]: Skipgram negative sampling is implicitly factorizing a specific matrix of this kind

Two key points to note:

1. The entries in the matrix are a shifted positive pointwise mutual information (SPPMI) between a word and its context word.

$$PMI(w,c) = \log \frac{p(w,c)}{p(w)p(c)}$$

These probabilities are computed by counting the data and normalizing them

Word2vec and matrix factorization

[Levy and Goldberg, NIPS 2014]: Skipgram negative sampling is implicitly factorizing a specific matrix of this kind

Two key points to note:

1. The entries in the matrix are a shifted positive pointwise mutual information (SPPMI) between a word and its context word.

$$PMI(w,c) = \log \frac{p(w,c)}{p(w)p(c)}$$

$$SPPMI(w,c) = \max(0, PMI(w,c) - \log k)$$

Word2vec and matrix factorization

[Levy and Goldberg, NIPS 2014]: Skipgram negative sampling is implicitly factorizing a specific matrix of this kind

Two key points to note:

2. The matrix factorization method is not truncated SVD.
   – It instead minimizes the objective function to compute the factorized matrices

This lecture

- The word2vec models: CBOW and Skipgram

- Connection between word2vec and matrix factorization

- GloVe [Pennington et al 2014]

What matrix to factorize?

If we are building word embeddings by factorizing a matrix, what matrix should we consider?

- Word counts [Rhode et al 2005]
- Shifted PPMI (implicitly) [Mikolov 2013, Levy & Goldberg 2014]

- Another answer: log co-occurrence counts [Pennington et al 2014]

Co-occurrence probabilities

Given two words i and j that occur in text, their co-occurrence probability is defined as the probability of seeing word i in the context of word j

$$P(j \mid i) = \frac{\text{count}(j \text{ in context of } i)}{\sum_k \text{count}(k \text{ in context of } i)}$$

Co-occurrence probabilities

Given two words i and j that occur in text, their co-occurrence probability is defined as the probability of seeing word i in the context of word j

$$P(j \mid i) = \frac{\text{count}(j \text{ in context of } i)}{\sum_k \text{count}(k \text{ in context of } i)}$$

Claim: If we want to distinguish between two words, it is not enough to look at their co-occurrences, we need to look at the ratio of their co-occurrences with other words

- Formalizing this intuition gives us an optimization problem

Co-occurrence ratios: An example

Words that can discriminate between ice and steam

Equally close to both ice and steam

Equally far to both ice and steam

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Example from Pennington et al 2014

Co-occurrence ratios: An example

Words that can discriminate between ice and steam

Equally close to both ice and steam

Equally far to both ice and steam

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

high

"The word solid occurs more frequently with ice than it does with steam"

low

"The word gas occurs less frequently with ice than it does with steam"

Close to one

"The words water and fashion occur about as frequently with ice than it does with steam"

Example from Pennington et al 2014

The GloVe objective

Notation:

- i : word, j : a context word
- w_i: The word embedding for i
- c_j: The context embedding for j
- b_i^w, b_j^c: Two bias terms: word and context specific
- X_{ij}: The number of times word i occurs in the context of j

The intuition:

1. Construct a word-context matrix whose $(i, j)^{th}$ entry is $\log X_{ij}$

2. Find vectors w_i, c_j and the biases b_i, c_j such that the dot product of the vectors added to the biases approximates the matrix entries

The GloVe objective

Notation:

- i : word, j : a context word
- w_i: The word embedding for i
- c_j: The context embedding for j
- b_i^w, b_j^c: Two bias terms: word and context specific
- X_{ij}: The number of times word i occurs in the context of j

Objective

$$J = \sum_{i,j=1}^{|V|} \left(w_i^T c_j + b_i + b_j - \log X_{ij}\right)^2$$

The GloVe objective

Notation:

- i : word, j : a context word
- w_i: The word embedding for i
- c_j: The context embedding for j
- b_i^w, b_j^c: Two bias terms: word and context specific
- X_{ij}: The number of times word i occurs in the context of j

Objective

$$J = \sum_{i,j=1}^{|V|} \left(w_i^T c_j + b_i + b_j - \log X_{ij}\right)^2$$

Problem: Pairs that frequently co-occur tend to dominate the objective.

The GloVe objective

Notation:

- i : word, j : a context word
- w_i: The word embedding for i
- c_j: The context embedding for j
- b_i^w, b_j^c: Two bias terms: word and context specific
- X_{ij}: The number of times word i occurs in the context of j

Objective

$$J = \sum_{i,j=1}^{|V|} \left(w_i^T c_j + b_i + b_j - \log X_{ij}\right)^2$$

Problem: Pairs that frequently co-occur tend to dominate the objective.

Answer: Correct for this by adding an extra term that prevents this

The GloVe objective

Notation:

- i : word, j : a context word
- w_i: The word embedding for i
- c_j: The context embedding for j
- b_i^w, b_j^c: Two bias terms: word and context specific
- X_{ij}: The number of times word i occurs in the context of j

Objective

$$J = \sum_{i,j=1}^{|V|} f(X_{ij})\left(w_i^T c_j + b_i + b_j - \log X_{ij}\right)^2$$

f : A weighting function that assigns lower relative importance to frequent co-occurrences

GloVe: Global Vectors

Essentially a matrix factorization method

Does not compute standard SVD though

1. Re-weighting for frequency
2. Two-way factorization, unlike SVD which produces U, Σ, V
3. Bias terms

Final word embeddings for a word: The average of the word and the context vectors of that word

Summary

- We saw three different methods for training word embeddings
- Many, many, many variants and improvements exist
- Various tunable parameters/training choices:
  - Dimensionality of embeddings
  - Text for training the embeddings
  - The context window size, whether it should be symmetric
  - And the usual stuff: Learning algorithm to use, the loss function, hyper-parameters
- See references for more details