Extensions to the Skipgram Model

VL Embeddings

Uni Heidelberg

SS 2019
The SkipGram model

- Objective: Find word representations that are useful for predicting the surrounding words in a sentence or a document
- More formally:

\[-\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+1}|w_t)\] (1)

where \( p(w_o|w_c) = \frac{\exp(v^\top w_o v^\top w_c)}{\sum_{j=1}^{V} \exp(v^\top j v^\top w_c)} \)  

- All parameters need to be updated at every step
- Impractical: cost of computing \( p(w_o|w_c) \) is proportional to \( V \)
Hierarchical Softmax

Computationally efficient approximation of the full softmax

- First introduced by Morin and Bengio (2005)
- Instead of evaluating $V$ output nodes, we evaluate only $\log_2(V)$ nodes

- How does it work?
  - binary tree representation of output layer where all words in vocab $V$ are leaf nodes
  - for each node, represent the relative probabilities of its child nodes
  - random walk that assigns probabilities to words
Hierarchical Softmax

Binary tree representation of output layer where all words in vocab \( V \) are leaf nodes
Hierarchical Softmax

For each node, represent the relative probabilities of its child nodes: transition probabilities to the children are given by the proportions of total probability mass in the subtree of its left- vs its right child.
Hierarchical Softmax

Relative probabilities define a random walk that assigns probabilities to leaf nodes (words)
Hierarchical Softmax

- Probability for each word is result of a sequence of binary decisions
- For example

\[ p(time|C) = P_{n_0}(left|C) P_{n_1}(right|C) P_{n_2}(left|C) \]

where \( P_n(right|C) \) is the probability of choosing the right child when transitioning from node \( n \)

- There are only 2 outcomes, therefore

\[ P_n(right|C) = 1 - P_n(left|C) \]
Hierarchical Softmax

But where does the tree come from?

- Different approaches in the literature:
  - Morin and Bengio (2005)
    - binary tree based on the IS-A relation in WordNet
  - Mnih and Hinton (2009)
    - boot-strapping method: hierarchical language model with a simple feature-based algorithm for automatic construction of word trees from data
  - Mikolov et al. (2013)
    - Huffman tree
Hierarchical Softmax

But where does the tree come from?

- Different approaches in the literature:
  
  - Morin and Bengio (2005)
    - binary tree based on the IS-A relation in WordNet
  
  - Mnih and Hinton (2009)
    - boot-strapping method: hierarchical language model with a simple feature-based algorithm for automatic construction of word trees from data

  - Mikolov et al. (2013)
    - Huffman tree
Hierarchical Softmax

Huffman trees (Mikolov et al. 2013)

- often used for loss-less data compression (Huffman 1952)
  - minimise expected path length from root to leaf
  ⇒ thereby minimising the expected number of parameter updates
Hierarchical Softmax

Huffman trees (Mikolov et al. 2013)

- often used for loss-less data compression (Huffman 1952)
  - minimise expected path length from root to leaf
    ⇒ thereby minimising the expected number of parameter updates

<table>
<thead>
<tr>
<th>word</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>fat</td>
<td>3</td>
</tr>
<tr>
<td>fridge</td>
<td>2</td>
</tr>
<tr>
<td>zebra</td>
<td>1</td>
</tr>
<tr>
<td>potato</td>
<td>3</td>
</tr>
<tr>
<td>and</td>
<td>14</td>
</tr>
<tr>
<td>in</td>
<td>7</td>
</tr>
<tr>
<td>today</td>
<td>4</td>
</tr>
<tr>
<td>kangaroo</td>
<td>2</td>
</tr>
</tbody>
</table>
Hierarchical Softmax

Huffman trees (Mikolov et al. 2013)

- often used for loss-less data compression (Huffman 1952)
  - minimise expected path length from root to leaf
  ⇒ thereby minimising the expected number of parameter updates

<table>
<thead>
<tr>
<th>word</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>fat</td>
<td>3</td>
</tr>
<tr>
<td>fridge</td>
<td>2</td>
</tr>
<tr>
<td>zebra</td>
<td>1</td>
</tr>
<tr>
<td>potato</td>
<td>3</td>
</tr>
<tr>
<td>and</td>
<td>14</td>
</tr>
<tr>
<td>in</td>
<td>7</td>
</tr>
<tr>
<td>today</td>
<td>4</td>
</tr>
<tr>
<td>kangaroo</td>
<td>2</td>
</tr>
</tbody>
</table>

Hierarchical softmax reduces number of parameters from $V$ to $\log_2(V)$

Image from http://building-babylon.net/2017/08/01/hierarchical-softmax/
Hierarchical Softmax

- Each word $w$ can be reached by a path from the root node
- Average $L(w)$ is $\log(V)$
- Assigns short codes to frequent words → fast training

Old

\[
p(w_o|w_c) = \frac{\exp(v_{w_o}^T v_{w_c})}{\sum_{j=1}^V \exp(v_j^T v_{w_c})}
\]  \hspace{1cm} (2)

New

\[
p(w|w_c) = \prod_{j=1}^{L(w)-1} \sigma(v'_{n(w,j)}^T v_{w_c})
\]  \hspace{1cm} (3)
Hierarchical Softmax

- Each word $w$ can be reached by a path from the root node.
- Average $L(w)$ is $\log(V)$.
- Assigns short codes to frequent words $\rightarrow$ fast training.

Old

\[ p(w_o | w_c) = \frac{\exp(v_{w_o}^T v_{w_c})}{\sum_{j=1}^V \exp(v_j^T v_{w_c})} \] \hspace{1cm} (2)

- Two representations $(v_{w_c}, v_{w_o})$ for each word $w$.

New

\[ p(w | w_c) = \prod_{j=1}^{L(w)-1} \sigma(v'_{n(w,j)}^T v_{w_c}) \] \hspace{1cm} (3)

- One representation for each word $w$ and for each inner node $v'_n$. 
Hierarchical Softmax

\[ p(w|w_c) = \prod_{j=1}^{L(w)-1} \sigma(v_{n(w,j)}^\top v_{w_c}) \] (3)
Hierarchical Softmax

\[ p(w|w_c) = \prod_{j=1}^{L(w)-1} \sigma(v'_{n(w,j)}^\top v_{w_c}) \] (3)
Hierarchical Softmax

\[
p(w|w_c) = \prod_{j=1}^{L(w)-1} \sigma(v'_{n(w,j)}^\top v_{w_c})
\]

\[
\sum_{w=1}^{V} p(w|w_c) = 1
\]

⇒ implies that the cost of computing \( \log p(w_o|w_c) \) and \( \nabla \log p(w_o|w_c) \) is proportional to \( L(w_o) \), which, on average, is \( \log(V) \)

Image from http://building-babylon.net/2017/08/01/hierarchical-softmax/
Hierarchical Softmax – Sum-up

• Problem with Softmax:
  • cost of computing $p(w_o|w_c)$ is proportional to $V$

• Solution: Hierarchical Softmax
  • computationally efficient approximation of full Softmax
  • word2vec uses **Huffman trees** to implement Hierarchical Softmax
  • other tree representations are also possible (see Morin & Bengio 2005, Mnih & Hinton 2009)
Negative Sampling

Can we do better?

- Instead of summarising over all contexts in the corpus, create artificial negative samples

Goal: sample context words \( v_o \) that are unlikely to occur with \( v_c \)

- Generate the set of random \((v_c, v_o)\) pairs, assuming they are all incorrect \(\Rightarrow\) randomly sampled negative examples
Skip-Gram with Negative Sampling

• Given a pair \((v_c, v_o)\) of word and context
  • \(p(D = 1|v_c, v_o)\) if \((v_c, v_o) \in D\)
  • \(1 - p(D = 1|v_c, v_o)\) if \((v_c, v_o) \notin D\)

Goal: find parameters \(\theta\) that maximise the probability that all of the observed pairs are from \(D\): 
\[
\arg\max_{\theta} \prod_{(v_c, v_o) \in D} p(D = 1|v_c, v_o; \theta) = \arg\max_{\theta} \sum_{(v_c, v_o) \in D} \log p(D = 1|v_c, v_o; \theta)
\]
Skip-Gram with Negative Sampling

- Given a pair \((v_c, v_o)\) of word and context
  - \(p(D = 1|v_c, v_o)\) if \((v_c, v_o) \in D\)
  - \(p(D = 0|v_c, v_o) = 1 - p(D = 1|v_c, v_o)\) if \((v_c, v_o) \notin D\)

Goal: find parameters \(\theta\) that maximise the probability that all of the observed pairs are from \(D\):

\[
\arg\max_{\theta} \prod_{(v_c, v_o) \in D} p(D = 1|v_c, v_o; \theta) = \arg\max_{\theta} \sum_{(v_c, v_o) \in D} \log p(D = 1|v_c, v_o; \theta)
\]
Skip-Gram with Negative Sampling

- Given a pair \((v_c, v_o)\) of word and context
  - \(p(D = 1|v_c, v_o)\) if \((v_c, v_o) \in D\)
  - \(p(D = 0|v_c, v_o) = 1 - p(D = 1|v_c, v_o)\) if \((v_c, v_o) \notin D\)

- Goal: find parameters \(\theta\) that maximise the probability that all of the observed pairs are from \(D\):

\[
\arg\max_\theta \prod_{(v_c, v_o) \in D} p(D = 1|v_c, v_o; \theta) = 
\]
Skip-Gram with Negative Sampling

- Given a pair \((v_c, v_o)\) of word and context
  - \(p(D = 1|v_c, v_o)\) if \((v_c, v_o) \in D\)
  - \(p(D = 0|v_c, v_o) = 1 - p(D = 1|v_c, v_o)\) if \((v_c, v_o) \notin D\)

- Goal: find parameters \(\theta\) that maximise the probability that all of the observed pairs are from \(D\):

\[
\arg\max_\theta \prod_{(v_c, v_o) \in D} p(D = 1|v_c, v_o; \theta) = \\
\arg\max_\theta \sum_{(v_c, v_o) \in D} \log p(D = 1|v_c, v_o; \theta)
\]
Skip-Gram with Negative Sampling (II)

- We can define $p(D = 1|v_o, v_c; \theta)$:

$$p(D = 1|v_c, v_o; \theta) = \frac{1}{1 + e^{-v_o \cdot v_c}}$$ sigmoid function
We can define \( p(D = 1|v_o, v_c; \theta) \):

\[
p(D = 1|v_c, v_o; \theta) = \frac{1}{1 + e^{-v_o \cdot v_c}} \text{ sigmoid function}
\]

This gives us the objective:

\[
\arg\max_{v_c, v_o} \sum_{(v_c, v_o) \in D} \log \frac{1}{1 + e^{-v_o \cdot v_c}}
\]
Skip-Gram with Negative Sampling (II)

- We can define \( p(D = 1|v_o, v_c; \theta) \):
  \[ p(D = 1|v_c, v_o; \theta) = \frac{1}{1 + e^{-v_o \cdot v_c}} \text{ sigmoid function} \]

- This gives us the objective:
  \[ \arg\max_{v_c, v_o} \sum_{(v_c, v_o) \in D} \log \frac{1}{1 + e^{-v_o \cdot v_c}} \]

- Training objective with negative sampling:
  \[ \arg\max_{v_c, v_o} \left( \prod_{(v_c, v_o) \in D} p(D = 1|v_o, v_c) \prod_{(v_c, v_o) \in D'} p(D = 0|v_o, v_c) \right) = \]
Skip-Gram with Negative Sampling (II)

- We can define $p(D = 1|v_o, v_c; \theta)$:
  
  $$p(D = 1|v_c, v_o; \theta) = \frac{1}{1 + e^{-v_o \cdot v_c}}$$

  sigmoid function

- This gives us the objective:

  $$\arg\max_{v_c, v_o} \sum_{(v_c, v_o) \in D} \log \frac{1}{1 + e^{-v_o \cdot v_c}}$$

- Training objective with negative sampling:

  $$\arg\max_{v_c, v_o} \left( \prod_{(v_c, v_o) \in D} p(D = 1|v_o, v_c) \prod_{(v_c, v_o) \in D'} p(D = 0|v_o, v_c) \right) =$$

  $$\arg\max_{v_c, v_o} \left( \sum_{(v_c, v_o) \in D} \log \sigma(v_o \cdot v_c) + \sum_{(v_c, v_o) \in D'} \log \sigma(-v_o \cdot v_c) \right)$$
Skip-Gram with Negative Sampling (III)

- Online training using Stochastic Gradient Descent

\[ J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta) \]

\[ J_t(\theta) = \log \sigma(v_o^\top v_c) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-v_w^\top v_c)] \]
Skip-Gram with Negative Sampling (III)

- Online training using Stochastic Gradient Descent

\[ J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta) \]

\[ J_t(\theta) = \log \sigma(v_o^T v_c) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[\log \sigma(-v_{w_i}^T v_c)] \]

maximise probability of seen word pairs
minimise probability of unseen word pairs
Skip-Gram with Negative Sampling (IV)

How to generate the samples?

• For each \((v_c, v_o) \in D\) generate \(n\) samples \((v_c, v_o_1), \ldots, (v_c, v_o_n)\) where
  • \(n\) is a hyperparameter
  • each \(v_o_j\) is drawn according to its unigram distribution raised to the \(3/4\) power
    \[ P(w) = U(w) \frac{3}{4} \]
    (causes less frequent words to be sampled more often)

⇒ observed word pairs will have similar embeddings
⇒ unobserved word pairs will be scattered in space
How to generate the samples?

- For each \((v_c, v_o) \in D\) generate \(n\) samples \((v_c, v_{o1}), \ldots, (v_c, v_{on})\)
  where
  - \(n\) is a hyperparameter
  - each \(v_{oj}\) is drawn according to its unigram distribution raised to the 3/4 power \(P(w) = U(w)^{3/4} / Z\)
  (causes less frequent words to be sampled more often)
Skip-Gram with Negative Sampling (IV)

How to generate the samples?

• For each \((v_c, v_o) \in D\) generate \(n\) samples \((v_c, v_{o1}), \ldots, (v_c, v_{on})\) where
  
  • \(n\) is a hyperparameter
  • each \(v_{oj}\) is drawn according to its unigram distribution raised to the 3/4 power \(P(w) = U(w)^{3/4}/Z\) (causes less frequent words to be sampled more often)

\[ \Rightarrow \text{observed word pairs will have similar embeddings} \]
\[ \Rightarrow \text{unobserved word pairs will be scattered in space} \]
Skip-Gram with Negative Sampling (IV)

How many samples? Impact of sample size $k$

- 2 functions of $k$:
  1. better estimate of distribution of negative examples: higher $k$ means more data and better estimation
  2. $k$ acts as a prior on the probability of observing positive examples: higher $k \rightarrow$ negative examples more probable
Subsampling of frequent words

- In large corpora: Zipfian distribution
  - few words with very high frequency
  - many words with very low frequency
Subsampling of frequent words

- In large corpora: **Zipfian distribution**
  - few words with very high frequency
  - many words with very low frequency

- high-frequency words often provide less information than less frequent words:

  *France is the capital of Paris*

  *France, capital* → more informative than *the, of*
Subsampling of frequent words

- In large corpora: Zipfian distribution
  - few words with very high frequency
  - many words with very low frequency

- high-frequency words often provide less information than less frequent words:

  France is the capital of Paris

  \(France, \textit{capital} \rightarrow\) more informative than the, of

- Counter the imbalance between rare and frequent words
Subsampling of frequent words

- Simple subsampling approach:
  - Discard word $w_i$ in the training set with probability

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$  \hspace{1cm} (5)

where $f(w_i)$ is the frequency of word $w_i$ and $t$ is a threshold (typically around $10^{-5}$)
Subsampling of frequent words

- Simple subsampling approach:
  - Discard word $w_i$ in the training set with probability
    \[ P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \]  \hspace{1cm} (5)
    where $f(w_i)$ is the frequency of word $w_i$
    and $t$ is a threshold (typically around $10^{-5}$)

- Subsampling accelerates learning and significantly improves accuracy of embeddings for rare words
Sum-up: Extensions to the Skipgram model

Mikolov et al. (2013): Distributed Representations of Words and Phrases and their Compositionality

- More efficient training
- Higher quality word vectors

- Training with negative sampling results in faster training and better vector representations for frequent words
- Subsampling of frequent words improves training speed and accuracy for rare words
- Extension from word-based to phrase vectors (→ session on compositionality)


• Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean (2013): Efficient estimation of word representations in vector space. CoRR, abs/1301.3781


