Deep Learning: Word Embeddings

Lecture 11

Razvan C. Bunescu
School of Electrical Engineering and Computer Science
bunescu@ohio.edu
One-Hot Vector Representations

- **Sparse vector representation:**
  - \( V \) is the vocabulary
  - Each word \( w \) is mapped to a unique \( \text{id}(w) \) between 1 and \( |V| \).
    - i.e. the position of the word in the vocabulary.
  - Represent a word \( w \) using a “one-hot” vector \( \mathbf{w} \) of length \( |V| \):
    - \( \mathbf{w}[i] = 1 \), if \( i = \text{id}(w) \).
    - \( \mathbf{w}[i] = 0 \), otherwise

- **Example:**
  - Suppose \( \text{id}(\text{frog}) = 2 \) and \( \text{id}(\text{turtle}) = 4 \). Then:
    - \( \mathbf{w}(\text{frog}) = [0, 1, 0, 0, 0, ..., 0] \)
    - \( \mathbf{w}(\text{turtle}) = [0, 0, 0, 1, 0, ..., 0] \)
Sparse Representations of Words are Problematic for Machine Learning in NLP

1. **Document classification:**
   - Bag-of-words representation: each document is the sum of the vectors of all the words in the document, normalized to unit length.
   - Suppose we use softmax regression to classify into classes in C.
     - A parameter is needed for each (word, class) pair:
       - $|V| \times |C|$ parameters $\Rightarrow 100K \times 10 \Rightarrow 1M$ parameters.
       - The number of labeled documents needed to train these many parameters is unfeasible to obtain.
     - If *voleyball* does not appear in the training documents, but is mentioned in the test document, it will be completely ignored:
       - Even though *voleyball* is semantically close to *basketball*, which appeared many times in training documents from the *Sports* category.
Sparse Representations of Words are Problematic for Machine Learning in NLP

2. (Neural) Language Modeling:
   - Predict the next word in a sequence:
     • AI systems use deep ...... (dish? learning? about?, ...)
     • Need to compute $P(w | w_{-1}, w_{-2}, ...)$:
       - want $P(\text{learning} | \text{deep, use}) > P(\text{about} | \text{deep, use})$.
   - Predict the most likely word in a context:
     • AI systems use deep ...... algorithms. (dish? learning? about?, ...)
     • Need to compute $P(w | w_{-1}, w_{-2}, ..., w_1, w_2, ...)$.
   - Language modelling is useful for many tasks in NLP:
     • spell checking.
     • machine translation.
     • speech recognition.
(Neural) Language Modeling with Sparse Word Representations

2. (N)LM with (Naive) Softmax Regression:
AI systems use deep ...... algorithms

\[ P(w \mid \text{deep, algorithms}) \leq= \text{for each word } w \text{ in } V \]

- Need \(|W| = 2 \times |V| \times |V|\) parameters!
Sparse vs. Dense Representations of Words

- **Sparse representations:**
  - Each word $w$ is a sparse vector $w \in \{0,1\}^{|V|}$ or $\mathbb{R}^{|V|}$.
  - Using words as features leads to a large number of parameters.
    - $\text{sim}(\text{frog, turtle}) = 0$.

- **Dense representations:**
  - Each word $w$ is a dense vector $w \in \mathbb{R}^k$, where $k \ll |V|$.
  - Can use unsupervised learning:
    - Use Harris’ **Distributional Hypothesis** [Word, 1954]
      - Words that appear in the same contexts tend to have similar meanings.
    - $\text{sim}(\text{frog, turtle}) > \text{sim}(\text{frog, magpie}) > 0$
(Neural) Language Modeling with Dense Word Representations

- Softmax on top of a hidden layer of size $h$ per word:

$$P(w \mid \text{deep}, \text{algorithms}) \Leftarrow \text{for each word } w \text{ in } V$$

$$\text{#params} = |V| \times h + 2 \times h \times |V|$$

Lecture 11
Neural Language Modeling with Dense Word Representations

- **Neural Language Modeling:**
  - Associate each word \( w \) with its *distributed representation* \( Uw \).
    - \( w \) is the *sparse (one-hot) representation* of word \( w \).
    - \( Uw \) is the *dense representation* of word \( w \):
      - i.e. word embedding.
      - i.e. distributed representation.
    - \( U \) is the projection or embedding matrix:
      - its columns are the word embeddings.
      - Simultaneously learn the word embeddings (\( U \)) and the softmax parameters (\( W \)).
      - After training on large text corpus, throw away \( W \), keep only \( U \).
Neural Language Models for Learning Word Embeddings

- Softmax training of NLMs is expensive:
  - Maximum Likelihood $\Rightarrow$ minimize cross-entropy.
- Need to compute the normalization constant for each training example i.e. for each word instance in the corpus:

$$E_D(W) = -\ln \prod_{i=1}^{T} p(w_t | h_t) = -\sum_{t=1}^{T} \ln \frac{\exp(W[w_t:] \times h_t)}{Z(w_t)}$$

$$Z(w_t) = \sum_{v \in V} \exp(W[v:] \times h_t)$$

- Use Pairwise Ranking approach instead.
Neural Language Models for Learning Word Embeddings

- **Pairwise Ranking** approach:
  - Train such that $p(w_t | h_t) > p(w | h_t)$ i.e. $W[w_t:] \times h_t > W[w:] \times h_t$
  - $w$ is sampled at random from $V$.
  - give a higher score to the actual word $w_t$ than to random words.

\[
W, U = \text{argmin} \sum_{t=1}^{T} \sum_{w \in V} \max \{0, 1 - W[w_t:] \times h_t + W[w:] \times h_t\}
\]

\[
\text{minimize} \quad J(U, W) = \sum_{t=1}^{T} \sum_{w \in V} \xi_{t,w} + R(U) + R(W)
\]

subject to: $W[w_t:] \times h_t - W[w:] \times h_t \geq 1 - \xi_{t,w}$
Neural Language Models for Learning Word Embeddings

• **Pairwise Ranking** approach [Collobert et al., JMLR’11]:
  - Train using SGD on the ranking criterion.
  - Sample “negative” words from V for each $w_t$.

• Evaluation of learned embeddings:
  - Word similarity questions:
    - given seed word $w$, find word(s) $v$ with most similar embedding:
      $$\arg \max_{v \in V} \cos(Uw, Uv)$$
  - Analogy questions [Mikolov et al., NIPS’13].
### Table 7: Word embeddings in the word lookup table of the language model neural network LM1 trained with a dictionary of size 100,000. For each column the queried word is followed by its index in the dictionary (higher means more rare) and its 10 nearest neighbors (using
The Skip-gram Model

- Learn vector representations of words in order to predict surrounding words:
  - Logistic regression with negative sampling and subsampling of frequent words.

Mikolov et al., NIPS’13

Lecture 11

Read an array of *numbers* from a text file, using STL in C++

- icon
- browser
- problem
- graphics
- program

randomly sampled negative context words ($N_t$)
The Skip-gram Model

• Common words provide less information than rare words:
  – co-occurrence \((\text{France, Paris})\) more important than \((\text{France, the})\).
• try to counter imbalance between rare and frequent words.

1. **Negative sampling:**
   – Estimate unigram distribution \(U(w)\) of words from the training corpus.
   – Sample negatives according to distribution \(P_n(w) = U(w)^{3/4} / Z\).

2. **Subsampling of frequent words:**
   – Compute discounting distribution \(P_d(w) = 1 – \sqrt{10^{-5} / f(w)}\).
   – Discard (positive) examples \(w_t\) according to \(P_d(w_t)\).
The Skip-gram Model

• Use vector representations **both as features and parameters**:
  - **Parameters**: vector $w_t$ of current word.
  - **Features**: vector $w_k$ of word to be predicted.

\[
P(w_k \in C_t|w_t) = \sigma(w_t^T w_k) = (1 + \exp(-w_t^T w_k))^{-1}
\]

  - Original model use 2 versions of parameters for same word $w_i$:
    • Input version $w_i$ for parameters, output version $w_i'$ for features.

• Train on a sequence of $T$ words:
  - Use SGD to minimize negative log-likelihood objective:

\[
J(w) = \sum_{t=1}^{T} \sum_{w_+ \in C_t} \left( \log \sigma(w_t^T w_+) + \sum_{w_- \in N_t} \log \sigma(-w_t^T w_-) \right)
\]
Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.
The Skip-gram Model

Evaluation of normalized learned embeddings:

- Word similarity questions:
  - given seed word \( w \), find word(s) \( v \) with most similar embedding:
    \[
    v = \arg\max_{v \in V} \mathbf{w}^T \mathbf{v}
    \]

- Analogy questions:
  - find word \( x \) such that: \( w \) (e.g. Paris) is to \( v \) (e.g. France) what \( x \) is to \( u \) (e.g. Germany).
  - want \( \mathbf{w} - \mathbf{v} \) to be very similar to \( \mathbf{x} - \mathbf{u} \) \( \iff \mathbf{w} - \mathbf{v} + \mathbf{u} \) very similar to \( \mathbf{x} \).
    \[
    x = \arg\max_{x \in V} x^T (\mathbf{w} - \mathbf{v} + \mathbf{u})
    \]
## The Skip-gram Model

[Mikolov et al., ICLR’13](#)

### 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>gold: Au</td>
<td>uranium: plutonium</td>
</tr>
<tr>
<td>Berlusconi - Silvio</td>
<td>Sarkozy: Nicolas</td>
<td>Putin: Medvedev</td>
<td>Obama: Barack</td>
</tr>
<tr>
<td>Microsoft - Windows</td>
<td>Google: Android</td>
<td>IBM: Linux</td>
<td>Apple: iPhone</td>
</tr>
<tr>
<td>Microsoft - Ballmer</td>
<td>Google: Yahoo</td>
<td>IBM: McNealy</td>
<td>Apple: Jobs</td>
</tr>
<tr>
<td>Japan - sushi</td>
<td>Germany: bratwurst</td>
<td>France: tapas</td>
<td>USA: pizza</td>
</tr>
</tbody>
</table>

Lecture 11
Word Embeddings: Extensions

• “Efficient Non-parametric Estimation of Multiple Embeddings per Word in Vector Space” [Neelakantan et al., EMNLP’14].
• “Dependency-Based Word Embeddings” [Levy & Goldberg, ACL’14].
• Combine with co-occurrence/PPMI based word embeddings:
  – “Symmetric Pattern Based Word Embeddings for Improved Word Similarity Prediction” [Schwartz et al., CoNLL’15].
• Bilingual word embeddings:
  – “Bilingual Word Embeddings for Phrase-Based Machine Translation” [Zou et al., EMNLP’13].
  – “Multilingual Models for Compositional Distributed Semantics” [Hermann & Blunsom, ACL’14].
• Phrase, paragraph, and document embeddings:
  – “Distributed Representations of Sentences and Documents” [Le & Mikolov, ICML’14]
  – ”KATE: K-Competitive Autoencoder for Text” [Chen & Zaki, KDD’17]