Applications in NLP

Marco Kuhlmann
Department of Computer and Information Science
What is natural language processing?

- **Natural language processing** develops techniques for the analysis and interpretation of natural language.

- Natural language processing is an interdisciplinary research area involving computer science, linguistics, and cognitive science.

  related names: language technology, computational linguistics
The Google Search index contains hundreds of billions of webpages and is well over 100,000,000 gigabytes in size.

Google, How Search Works
The Knowledge Gap

unstructured data (text)

natural language processing

structured data (database)
This Stanford University alumnus co-founded educational technology company Coursera.

SPARQL query against DBPedia

```
SELECT DISTINCT ?x WHERE {
  dbr:Coursera dbo:foundedBy ?x.
}
```
Syntactic structure, semantic relations

subject

Koller

co-founded

Coursera

object

dbr:Coursera dbo:foundedBy dbr:Daphne_Koller
Announcing SyntaxNet: The World’s Most Accurate Parser Goes Open Source
Thursday, May 12, 2016

Posted by Slav Petrov, Senior Staff Research Scientist

At Google, we spend a lot of time thinking about how computer systems can read and understand human language in order to process it in intelligent ways. Today, we are excited to share the fruits of our research with the broader community by releasing SyntaxNet, an open-source neural network framework implemented in TensorFlow that provides a foundation for Natural Language Understanding (NLU) systems. Our release includes all the code needed to train new SyntaxNet models on your own data, as well as Parsley McParseface, an English parser that we have trained for you and that you can use to analyze English text.
A very brief history

- 1950s: Fully automatic translation of Russian into English
- 1960s: Diaspora after funding cuts (ALPAC report)
- 1970s: Conceptual ontologies and chatterbots
- 1980s: Systems based on complex sets of hand-written rules
- 1990s: The surge of statistical techniques
- 2000s: Large corpora. Machine translation once more
- 2010s: Dominance of machine learning, neural networks
Ambiguity causes combinatorial explosion

'I only want to live in peace, plant potatoes, and dream!' – Moomin
Relative frequencies of tags per word

Data: UD English Treebank (training data)
Overview

- Introduction to natural language processing
- Introduction to word embeddings
- Learning word embeddings
Introduction to word embeddings
Words and contexts

What do the following sentences tell us about *garrotxa*?

- *Garrotxa* is made from milk.
- *Garrotxa* pairs well with crusty country bread.
- *Garrotxa* is aged in caves to enhance mold development.

The distributional principle

- The *distributional principle* states that words that occur in similar contexts tend to have similar meanings.

- ‘You shall know a word by the company it keeps.’

  Firth (1957)
Word embeddings

- A **word embedding** is a mapping of words to points in a vector space such that nearby words (points) are similar in terms of their distributional properties.
  
  distributional principle: have similar meanings

- This idea is similar to the vector space model of information retrieval, where the dimensions of the vector space correspond to the terms that occur in a document.
  
  points = documents, nearby points = similar topic

Lin et al. (2015)
<table>
<thead>
<tr>
<th>target words</th>
<th>butter</th>
<th>cake</th>
<th>cow</th>
<th>deer</th>
</tr>
</thead>
<tbody>
<tr>
<td>cheese</td>
<td>12</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>bread</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>goat</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>sheep</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

Co-occurrence matrix
# Co-occurrence matrix

<table>
<thead>
<tr>
<th>Target words</th>
<th>butter</th>
<th>cake</th>
<th>cow</th>
<th>deer</th>
</tr>
</thead>
<tbody>
<tr>
<td>cheese</td>
<td>12</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>bread</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>goat</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>sheep</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>
From co-occurrences to word vectors
Sparse vectors versus dense vectors

- The rows of co-occurrence matrices are long and sparse. Length corresponds to number of context words = on the order of $10^4$

- State-of-the-art word embeddings that are short and dense. Length on the order of $10^2$

- The intuition is that such vectors may be better at capturing generalisations, and easier to use in machine learning.
Simple applications of word embeddings

- finding similar words
- answering ‘odd one out’-questions
- computing the similarity of short documents
# Recognising textual entailment

<table>
<thead>
<tr>
<th>Entail</th>
<th>Doctors are performing surgery.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral</td>
<td>Two doctors are performing surgery on a man.</td>
</tr>
<tr>
<td>Contradict</td>
<td>Two surgeons are having lunch.</td>
</tr>
</tbody>
</table>

Example from Bowman et al. (2015)
Obtaining word embeddings

- Word embeddings can be easily trained from any text corpus using available tools.
  
  `word2vec, Gensim, GloVe`

- Pre-trained word vectors for English, Swedish, and various other languages are available for download.
  
  `word2vec, Swectors, Polyglot project, spaCy`
Compositional structure of word embeddings
Limitations of word embeddings

- There are many different facets of ‘similarity’.
  Is a *cat* more similar to a *dog* or to a *tiger*?

- Text data does not reflect many ‘trivial’ properties of words.
  more ‘black sheep’ than ‘white sheep’

- Text data does reflect human biases in the real world.
  king – man + woman = queen, doctor – man + woman = nurse

Goldberg (2017)
Questions about word embeddings

- How to measure association strength?
  positive pointwise mutual information

- How to measure similarity?
  cosine similarity

- How to learn word embeddings from text?
  matrix factorisation, direct learning of the low-dimensional vectors
Pointwise mutual information

- Raw counts favour pairs that involve very common contexts. *the cat, a cat* will receive higher weight than *cute cat, small cat*

- We want a measure that favours contexts in which the target word occurs more often than other words.

- A suitable measure is **pointwise mutual information (PMI)**:

\[
\text{PMI}(x, y) = \log \frac{P(x, y)}{P(x)P(y)}
\]
Pointwise mutual information

- We want to use PMI to measure the associative strength between a word \( w \) and a context \( c \) in a data set \( D \):

\[
PMI(w, c) = \log \frac{P(w, c)}{P(w)P(c)}
\]

- We can estimate the relevant probabilities by counting:

\[
PMI(w, c) = \log \frac{\#(w, c)/|D|}{\#(w)/|D| \cdot \#(c)/|D|} = \log \frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)}
\]
Positive pointwise mutual information

- Note that PMI is infinitely small for unseen word–context pairs, and undefined for unseen target words.

- In **positive pointwise mutual information (PPMI)**, all negative and undefined values are replaced by zero:

\[
\text{PPMI}(w, c) = \max(\text{PMI}(w, c), 0)
\]

- Because PPMI assigns high values to rare events, it is advisable to apply a count threshold or smooth the probabilities.
Questions about word embeddings

- How to measure association strength?
  positive pointwise mutual information

- How to measure similarity?
  cosine similarity

- How to learn word embeddings from text?
  matrix factorisation, direct learning of the low-dimensional vectors
Distance-based similarity

- If we can represent words as vectors, then we can measure word similarity as the distance between the vectors.
- Most measures of vector similarity are based on the dot product or inner product from linear algebra.
The dot product

\[
\begin{bmatrix}
  v_1 & v_2 & w_1 & w_2 \\
+2 & +2 & +2 & +1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  v_1 & v_2 & w_1 & w_2 \\
+2 & +2 & -2 & -1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  v_1 & v_2 & w_1 & w_2 \\
+2 & +2 & -2 & +2 \\
\end{bmatrix}
\]

\[
v \cdot w = 6
\]

\[
v \cdot w = -6
\]

\[
v \cdot w = \pm 0
\]
Problems with the dot product

- The dot product will be higher for vectors that represent words that have high co-occurrence counts or PPMI values.

- This means that, all other things being equal, the dot product of two words will be greater if the words are frequent.

- This makes the dot product problematic because we would like a similarity metric that is independent of frequency.
Cosine similarity

- We can fix the dot product as a metric by scaling down each vector to the corresponding unit vector:

\[
\cos(v, w) = \frac{v \cdot w}{|v| \cdot |w|} = \frac{v \cdot w}{|v||w|} = \frac{\sum_{i=1}^{d} v_i w_i}{\sqrt{\sum_{i=1}^{d} v_i^2} \sqrt{\sum_{i=1}^{d} w_i^2}}
\]

- This length-normalised dot product is the cosine similarity, whose values range from $-1$ (opposite) to $+1$ (identical).

cosine of the angle between the two vectors
Overview

- Introduction to natural language processing
- Introduction to word embeddings
- Learning word embeddings
Learning word embeddings
Learning word embeddings

- Word embeddings from neural language models
- word2vec: continuous bag-of-words and skip-gram
- Word embeddings via singular value decomposition
- Contextualised embeddings – ELMo and BERT
Neural networks as language models
Word embeddings via neural language models

- The neural language model is trained to predict the probability of the next word being $w$, given the preceding words:

\[
\hat{y} = P(w \mid \text{preceding words}) = \text{softmax}(hW)
\]

- Each column of the matrix $W$ is a $\text{dim}(h)$-dimensional vector that is associated with some vocabulary item $w$.

- We can view this vector as a representation of $w$ that captures its compatibility with the context represented by the vector $h$. 
Network weights = word embeddings

Intuitively, words that occur in similar contexts will have similar word representations.
Training word embeddings using a language model

- Initialise the word vectors with random values.
  typically by uniform sampling from an interval around 0

- Train the network on large volumes of text.
  word2vec: 100 billion words

- Word vectors will be optimised to the prediction task.
  Words that tend to precede the same words will get similar vectors.
Google’s word2vec

- Google’s word2vec implements two different training algorithms for word embeddings: **continuous bag-of-words** and **skip-gram**.

- Both algorithms obtain word embeddings from a *binary* prediction task: ‘Is this an actual word–context pair?’

- Positive examples are generated from a corpus. Negative examples are generated by taking $k$ copies of a positive example and randomly replacing the target word with some other word.
The continuous bag-of-words model

\[ P(\text{observed?} \mid c_1 \, w \, c_2) \]
The skip-gram model

\[ P(\text{observed?} \mid c_1 w c_2) \]

\[ \text{product} \]

\[ \text{sigmoid} \]

\[ \text{dot product} \]

\[ c_1 \]

\[ w \]

\[ c_2 \]
Word embeddings via singular value decomposition

- The rows of co-occurrence matrices are long and sparse. Instead, we would like to have word vectors that are short and dense. Length on the order of $10^2$ instead of $10^4$.

- One idea is to approximate the co-occurrence matrix by another matrix with fewer columns. In practice, use tf-idf or PPMI instead of raw counts.

- This problem can be solved by computing the singular value decomposition of the co-occurrence matrix.
Singular value decomposition

\[ \mathbf{M} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \]

- \( \mathbf{M} \): Context words
- \( \mathbf{U} \): Target words
- \( \mathbf{\Sigma} \): \( w \times c \) matrix
- \( \mathbf{V} \): \( c \times c \) matrix

Landauer and Dumais (1997)
Truncated singular value decomposition

\[ \mathbf{M} \mathbf{U} \mathbf{V}^\top = \begin{bmatrix} \mathbf{w} \times c \\ \mathbf{w} \times k \end{bmatrix} = \begin{bmatrix} k \times c \\ c \times c \end{bmatrix} \]

\( k = \text{embedding width} \)
Truncated singular value decomposition
Word embeddings via singular value decomposition

- Each row of the (truncated) matrix $U$ is a $k$-dimensional vector that represents the ‘most important’ information about a word. Columns ordered in decreasing order of importance.

- A practical problem is that computing the singular value decomposition for large matrices is expensive. But has to be done only once!
Connecting the two worlds

- The two algorithmic approaches that we have seen take two seemingly very different perspectives: ‘count-based’ and ‘neural’.

- However, a careful analysis reveals that the skip-gram model is implicitly computing the already-decomposed PPMI matrix.

  Levy and Goldberg (2014)
Contextualised embeddings: ELMo and BERT

- In the bag-of-words and skip-gram model, each word vector is obtained from a local prediction task.

- In ELMo and BERT, each token is assigned a representation that is a function of the entire input sentence.

- The final vector for a word is a linear combination of the internal layers of a deep bidirectional recurrent neural network (LSTM).

Muppet character images from The Muppet Wiki
The ELMo architecture
3.1 Encoder and Decoder Stacks

Encoder:
The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is $\text{LayerNorm}(x + \text{Sublayer}(x))$, where $\text{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Decoder:
The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position $i$ can depend only on the known outputs at positions less than $i$.

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.
Learning word embeddings

- Word embeddings from neural language models
- word2vec: continuous bag-of-words and skip-gram
- Word embeddings via singular value decomposition
- Contextualised embeddings – ELMo and BERT
Overview

- Introduction to natural language processing
- Introduction to word embeddings
- Learning word embeddings