Deep Learning with TensorFlow
http://cvml.ist.ac.at/courses/DLWT_W18

Lecture 4:
Word Vectors
Word Vectors

Learning Representations of Words and Phrases

Divyansh Gupta
Deep Learning with TensorFlow
2018-12-10
How to meaningfully represent text?

- Most basic unit of text encoding is a character (ascii/unicode)
- A character in itself carries very little meaning
  - ‘H’, ‘G’, ‘R’
- Words are the fundamental semantic and syntactic unit in language
  - ‘Help’, ‘garden’, ‘running’
- In machine learning, we usually represent quantities as vectors (or tensors). Strings are difficult to operate on.
- How can we have a vector for each word?
One - Hot vectors

- Only one entry is 1, rest are all 0s.

\[
\text{motel} = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]
\]

- Vector length = size of vocabulary (can be \(~1,000,000\) !)
- They are all orthogonal, no measure of similarity

\[
\text{motel} = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]
\]
\[
\text{hotel} = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]
\]

- Can we do better?
The Idea

Represent words as dense vectors that capture semantic and syntactic similarity.

That is, similar words should have similar vectors.
Distributional Semantics

- Guiding dogma of distributional semantics:
  
  “You shall know a word by the company it keeps”
  
  (J. R. Firth 1957)

- Use context information to learn meaningful vectors

...government debt problems turning into banking crises as happened in 2009...

...saying that Europe needs unified banking regulation to replace the hodgepodge...

...India has just given its banking system a shot in the arm...
Transfer Learning

- We can’t have a completely supervised way of training them, since we don’t actually know what these vectors should look like.

- So we train for a “proxy” task and use the representations for the actual task

- ‘Actual’ NLP tasks:
  - Machine Translation
  - Summarization
  - Text Classification
  - Question Answering
  - ...

![Transfer Learning Overview](image)
Methods

1. **Skip-gram** (Mikolov et al. 2013)
   Predict context words from center word

2. **CBOW**: Continuous Bag of Words (Mikolov et al. 2013)
   Predict center word from context words

3. **GloVe** (Socher et al. 2014)
   Incorporate co-occurrence counts into training

4. **FastText** (Bojanowski, et al. 2016)
   Uses morphological elements of words
Remember!

- We are not actually interested in the “fake” tasks that we are optimizing for.
- What we really care about are the intermediate representations that are learnt in the process.
- Also known as self-supervised learning as we are using implicit labels derived from the input data itself.
Skip-gram Training Data

Source Text

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

Training Samples

(the, quick)
(the, brown)
(quick, the)
(quick, brown)
(quick, fox)
(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)
(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)
Skip-gram Architecture

Input Vector

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

A ‘1’ in the position corresponding to the word “ants”

10,000 positions

Hidden Layer Linear Neurons

300 neurons

Hidden Layer Softmax Classifier

Output Layer Softmax Classifier

Probability that the word at a randomly chosen, nearby position is “abandon”

… “ability”

… “able”

… “zone”

Hidden Layer Weight Matrix

300 neurons

10,000 words

\[
[0 \ 0 \ 0 \ 1 \ 0] \times \begin{bmatrix}
17 & 24 & 1 \\
23 & 5 & 7 \\
4 & 6 & 13 \\
10 & 12 & 19 \\
11 & 18 & 25
\end{bmatrix} = [10 \ 12 \ 19]
\]
**Objective Function**

Objective function: $J(\theta) = \frac{1}{|\text{Text}|} \sum_{w \in \text{Text}} \sum_{c \in C(w)} \log P(c|w; \theta)$

$P(c|w; \theta) = \frac{\exp(u_c^T v_w)}{\sum_{c' \in V} \exp(u_{c'}^T v_w)}$

Where $\theta$ is a vector of parameters, and $v_w$ and $u_c$ are vectors representing word and output weights, respectively.

$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$
Upgrades! (Mikolov et al. 2013b)

1. Treating common word pairs or phrases as single “words” in their model. Example: New York
2. Subsampling frequent words to decrease the number of training examples. Probability of getting discarded:

\[
P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}
\]

3. Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update only a small percentage of the model’s weights.
Negative Sampling

- Softmax is too costly to compute
- So we convert the problem to binary classification of whether a given \((word, context\_word)\) pair belongs to the dataset, \(D\)

\[
J(\theta) = \sum_{(w,c)\in D} P(D = 1|w, c; \theta) = \sum_{(w,c)\in D} \log \frac{1}{1 + \exp(-u_c \cdot v_w)}
\]

- Has a trivial solution if \(P(D = 1|w, c; \theta) = 1\) for all \((w, c)\)
- So we randomly sample some negative examples that need not be in the text

\[
J(\theta) = \sum_{(w,c)\in D} P(D = 1|w, c; \theta) + \sum_{(w,c')\in D'} P(D = 0|w, c'; \theta)
\]
Negative Sampling (contd.)

\[ J(\theta) = \sum_{(w,c) \in D} P(D = 1|w, c; \theta) + \sum_{(w,c') \in D'} P(D = 0|w, c'; \theta) \]

\[ = \sum_{(w,c) \in D} \log \frac{1}{1 + \exp(-u_c \cdot v_w)} + \sum_{(w,c') \in D'} \log \frac{1}{1 + \exp(u_{c'} \cdot v_w)} \]

\[ = \sum_{(w,c) \in D} \log \sigma(u_c \cdot v_w) + \sum_{(w,c') \in D'} \log \sigma(-u_{c'} \cdot v_w) \]

- \(D'\) is constructed by randomly sampling \(c'\) from the following distribution

\[ P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} (f(w_j)^{3/4})} \]

- There are \(k\) (5-15) negative samples for each \((w, c) \in D\)
Skip-gram Summary

● The model needs to predict similar outputs for words that occur in similar contexts
  ○ This can be done by making their vectors similar
  ○ For example, intelligent and smart would appear in similar contexts
  ○ Similarly for contexts like “There are 11 players in a team”, and “Basketball is played by five members”...

● CBOW is just the opposite: predict center word from context words
A slightly different approach: co-occurrence

- Construct co-occurrence matrix
- Reduce with SVD (patented for IR in 1988)
- Example corpus:
  - I like deep. learning
  - I like NLP.
  - I enjoy flying.

<table>
<thead>
<tr>
<th>counts</th>
<th>I</th>
<th>like</th>
<th>enjoy</th>
<th>deep</th>
<th>learning</th>
<th>NLP</th>
<th>flying</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>like</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>enjoy</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>deep</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>learning</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NLP</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>flying</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
GloVe (2014) - Combining both methods

\[ J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2 \]

- $P_{ij}$ is the co-occurrence count of $w_i$ and $w_j$
- $f(P_{ij})$ is used to give less weight to infrequently occurring pairs
- Faster to train as it does not makes multiple passes through the corpus
Evaluation

Intrinsic

- Analyse the word vectors themselves to see what they represent
- For instance, Analogy task, similarity task, clustering

Extrinsic

- Use the generated representation on actual NLP tasks
- See which embedding does best on Part-of-speech tagging, machine translation etc.
Analogy Task

man:woman :: king:?

\[ d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|} \]

<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>
Other cool applications
Other cool applications

t-SNE visualization of the bilingual word embedding. Green is Chinese, Yellow is English.

(Socher et al. (2013a))
Implementation

- Pre-trained vectors available for all these methods!
  - Word2Vec (Skip-gram)
    https://code.google.com/archive/p/word2vec/
  - GloVe
    https://nlp.stanford.edu/projects/glove/
  - FastText (also for 156 other languages!)
embeddings = tf.Variable(tf.random_uniform([vocabulary_size,embedding_size],
-1.0, 1.0))
nce_weights = tf.Variable(tf.truncated_normal([vocabulary_size,
embedding_size],stddev=1.0 / math.sqrt(embedding_size)))
nce_biases = tf.Variable(tf.zeros([vocabulary_size]))

# Placeholders for inputs
train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])

# Embedding helper function:
embed = tf.nn.embedding_lookup(embeddings, train_inputs)
Compute the NCE loss, using a sample of the negative labels each time.

```
# Compute the NCE loss, using a sample of the negative labels each time.
loss = tf.reduce_mean(
    tf.nn.nce_loss(weights=nce_weights,
                    biases=nce_biases,
                    labels=train_labels,
                    inputs=embed,
                    num_sampled=num_sampled,
                    num_classes=vocabulary_size))
```

We use the SGD optimizer.

```
# We use the SGD optimizer.
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1.0).minimize(loss)
```

```
with tf.Session as session:
    for inputs, labels in generate_batch(...):
        feed_dict = {train_inputs: inputs, train_labels: labels}
        _, cur_loss = session.run([optimizer, loss], feed_dict=feed_dict)
```
References

1. Stanford NLP CS224n -
   http://web.stanford.edu/class/cs224n/syllabus.html


TensorBoard Demo

Adapted from:

http://www.cse.chalmers.se/~richajo/dit865/files/Word%20embeddings%20in%20Gensim.html

and

https://github.com/sudharsan13296/visualise-word2vec
Questions?