Named Entity Recognition with Deep Learning

Presentation by:
Reza Ghaeini
Farhana Ferdousi Liza
Index

- Named Entities Recognition
- Deep Learning
- Word Representation
- Recurrent Neural Network
- Dataset
- Experimental Result
- Future works
Named Entity Recognition

Detecting named entities like name of people, locations, organization etc. in a sentence and text.

Example:

Input: Jim bought 300 shares of Acmpe Corp. in 2006.

Output: [Jim]_Person bought 300 shares of [Acmpe Corp.]_Organization in [2006]_Time
Deep Learning

Recently, it outperforms many tasks in different areas like NLP

It tries to capture deep features of data by itself (no feature extraction ...)

It uses pre-trained information (generally unsupervised) (i.e. word representation) to achieve better results
Deep Learning
Word Representation

Each word in vocabulary associated with n-dimensional vector •

Capture similarity between words in different aspects •

It can capture interesting relations too. •

\[ [WR]_{\text{king}} - [WR]_{\text{man}} + [WR]_{\text{woman}} \approx [WR]_{\text{queen}} \] (Mikolov et al. 2013) •
Representation of Text

Representation of text is very important for performance of many real-world applications. The most common techniques are:

Local representations
- N-grams
- Bag-of-words
- 1-of-N coding

Continuous representations
- Latent Semantic Analysis
- Latent Dirichlet Allocation

Distributed Representations
Distributed Representation

Distributed vector representations that capture a large number of precise syntactic and semantic word relationships.

Distributed representations of words can be obtained from various neural network based language models:

**Feedforward neural net language model**
Recurrent neural net language model
First Proposed Model

- Four-gram neural net language model architecture (Bengio 2001)

- The training is done using stochastic gradient descent and Backpropagation

- The training complexity of the feedforward NNLM is high:
  - Propagation from projection layer to the hidden layer
  - Softmax in the output layer

- Using this model just for obtaining the word vectors is very inefficient
Improving efficiency

The full softmax can be replaced by:

- Hierarchical softmax (Morin and Bengio)
- Hinge loss (Collobert and Weston)
- Noise contrastive estimation (Mnih et al.)
- Negative sampling (Mikolov et al.)

Mikolov et al. further removed the hidden layer: for large models, this can provide additional speedup 1000x

- Continuous bag-of-words model
- Continuous skip-gram model
Proposed Model

In this work, I am proposing to use continuous skip-gram and bag of words architecture with following extension:

I want to optimize the objective to project a common vector space to maximize correlation between the same category words.
Continuous Skip-gram and Bag of words- family of log linear language model

NLP is so varied and complex, even using a extremely large corpus, we can never model all string of words. Skip-gram is a technique that allows n-gram to be stored to model the language but it allows token to be skipped.

Example
the sentence "Hi fred how was the pizza?"
becomes:
Continuus bag of words: 3-grams {"Hi fred how", "fred how was", "how was the", ...}

Skip-gram 1-skip 3-grams: {"Hi fred how", "Hi fred was", "fred how was", "fred how the", ...}
Skip-gram

Objective Function

\[
\arg\max_{\theta} \prod_{w \in \text{Text}} \left[ \prod_{c \in C(w)} p(c|w; \theta) \right] \\
\arg\max_{\theta} \prod_{(w, c) \in D} p(c|w; \theta)
\]

\[
p(c|w; \theta) = \frac{e^{v_c \cdot v_w}}{\sum_{c' \in C} e^{v_{c'} \cdot v_w}}
\]

Hierarchical Softmax

\[
p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma([n(w, j + 1) = ch(n(w, j))] \cdot v'_{n(w,j)}^T v_{w_I})
\]
Some result by Milokov et al, 2013

<table>
<thead>
<tr>
<th>Model</th>
<th>Vector Dimensionality</th>
<th>Training words</th>
<th>Accuracy [%]</th>
<th>Training time [days x CPU cores]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Semantic</td>
<td>Syntactic</td>
</tr>
<tr>
<td>NNLM</td>
<td>100</td>
<td>6B</td>
<td>34.2</td>
<td>64.5</td>
</tr>
<tr>
<td>CBOW</td>
<td>1000</td>
<td>6B</td>
<td>57.3</td>
<td>68.9</td>
</tr>
<tr>
<td>Skip-gram</td>
<td>1000</td>
<td>6B</td>
<td>66.1</td>
<td>65.1</td>
</tr>
</tbody>
</table>

I did not run the model on big enough dataset because of the time constraint
Some snapshot of result

<table>
<thead>
<tr>
<th>Name</th>
<th>Cosine Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td>0.650896</td>
</tr>
<tr>
<td>Robert</td>
<td>0.646968</td>
</tr>
<tr>
<td>Thomas</td>
<td>0.633958</td>
</tr>
<tr>
<td>William</td>
<td>0.629634</td>
</tr>
<tr>
<td>Richard</td>
<td>0.617703</td>
</tr>
<tr>
<td>Peter</td>
<td>0.609647</td>
</tr>
<tr>
<td>George</td>
<td>0.591549</td>
</tr>
<tr>
<td>Paul</td>
<td>0.589056</td>
</tr>
<tr>
<td>Nicholas</td>
<td>0.581043</td>
</tr>
<tr>
<td>Anthony</td>
<td>0.573255</td>
</tr>
<tr>
<td>Hugh</td>
<td>0.567399</td>
</tr>
<tr>
<td>Henry</td>
<td>0.559341</td>
</tr>
<tr>
<td>Joseph</td>
<td>0.554239</td>
</tr>
<tr>
<td>Edward</td>
<td>0.553235</td>
</tr>
<tr>
<td>Andrew</td>
<td>0.549794</td>
</tr>
<tr>
<td>Reginald</td>
<td>0.549595</td>
</tr>
<tr>
<td>Michael</td>
<td>0.547040</td>
</tr>
<tr>
<td>Charles</td>
<td>0.542569</td>
</tr>
<tr>
<td>Archibald</td>
<td>0.538724</td>
</tr>
<tr>
<td>Martin</td>
<td>0.534698</td>
</tr>
<tr>
<td>Wilfred</td>
<td>0.533364</td>
</tr>
<tr>
<td>Nigel</td>
<td>0.533343</td>
</tr>
<tr>
<td>Stephen</td>
<td>0.531121</td>
</tr>
<tr>
<td>Arthur</td>
<td>0.526178</td>
</tr>
<tr>
<td>Dudley</td>
<td>0.525375</td>
</tr>
<tr>
<td>Patrick</td>
<td>0.524747</td>
</tr>
<tr>
<td>Alastair</td>
<td>0.524410</td>
</tr>
<tr>
<td>Evangelist</td>
<td>0.516333</td>
</tr>
<tr>
<td>Walter</td>
<td>0.513963</td>
</tr>
<tr>
<td>Knowles</td>
<td>0.511593</td>
</tr>
<tr>
<td>Samuel</td>
<td>0.510832</td>
</tr>
<tr>
<td>Kenneth</td>
<td>0.508504</td>
</tr>
<tr>
<td>Erskine</td>
<td>0.507395</td>
</tr>
</tbody>
</table>
Some snapshot of result-location
Recurrent Neural Network

Considering a memory for some nodes in a neural network.

Next result will be affected by previous state. (We have directed cycle in them)
Implemented Structure

Implemented from scratch ... •
Recurrent Neural Network with these flexibilities: •
   Structural (act like): •
      Simple Neural Network •
      Elman Neural Network (a RNN) •
      Jordan Neural Network (a RNN) •
      Elman & Jordan Neural Network (a RNN) •
      Non-Linear function •
         Sigmoid Function •
         Tanh Function •
      Meta parameters & weight initialization methods •
      input & hidden Layers •
      Number of features for each words •
      etc. •
Dataset

Two different datasets

- Informal tweets of tweeter: Detect Person, Location, Org. NE
- Stanford dataset for NER: Detect Person NE

- Very sparse dataset with majority of zeros
- Applying resampling with different manner and strategies

Standard Dataset: CoNLL-2003

- Not easily available! (some paperwork and waiting for at least 7 business days)
Learning

Using Gradient descent with $L_2$ Regularization •

Using Backpropagation method •

Many Local minimums, initialization has high effect on result •
  When it goes bad, start over with another initialization •

Better to use new optimization methods: L-BFGS, AdaGrad, etc. (serious lack of time!) •
Experimental Result

NER has high accuracy since it has lots of zeros

Resampling ones labels, reduce local minimums and preventing zero Recall

In both dataset unpredictable & unbelievable result!!!!:

<table>
<thead>
<tr>
<th>Test Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

About 10% unseen data in test sets.
It happens because of dataset
I’m not telling that we solved NER task for sure!!!!
Future Works

Using Standard Dataset to be able to compare results with other presented methods

Using handy-crafted features for improving result in standard dataset

The result definitely will decrease in standard dataset
Thanks for your attention and time