Word Sense Determination from Wikipedia Data Using a Neural Net

CS 297 Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San Jose State University

By

Qiao Liu

May 2017

Table of Contents

Introduction

Many words carry different meanings based on their context. For instance, apple could refer to a fruit, a company or a film. The ability to identify the entities (such as apple) based on the context where it occurs, has been established as an important task in several areas, including topic detection and tracking, machine translation, and information retrieval. Our aim is to build an entity disambiguation system.

The Wikipedia data set has been used as data set in many research projects. One of them, which is similar to our project is 'Large-Scale Named Entity Disambiguation Based on Wikipedia Data', by Silviu Cucerzan [1]. Information is extracted from the titles of entity pages, the titles of redirecting pages, the disambiguation pages, and the references to entity pages in other Wikipedia articles [1]. The disambiguation process employs a vector space model based on hypothesis, in which a vector representation of the processed document is compared with the vector representations of the Wikipedia entities [1].

In our project, we use the English Wikipedia dataset as a source of word sense, and word embedding to determine the sense of word within the given context. Word embedding were originally introduced by Bengio, et al, in 2000 [2]. A Word embedding is a parameterized function mapping words in some language to high-

dimensional vectors. Methods to generate this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix, probabilistic models, and explicit representation in terms of the context in which words appear. Using a neural network to learn word embedding is one of the most exciting area of research in deep learning now [3]. Unlike previous work, in our project, we will use neural network to learn word embeddings.

The following are the deliverables I have done in this semester to understand the essence of machine learning, neural network, word embedding and the work flow of TensorFlow. In Deliverable 1, I developed a program to recognize handwritten digits using TensorFlow, softmax and MNIST dataset. TensorBoard is practiced in deliverable 1 as well. In Deliverable 2, I present the introduction to word embedding and some thoughts about the approach of the project. In Deliverable 3, I created a dictionary of the ambiguous entities in Wikipedia and extract those pages to plain text file. More details of those three deliverables are discussed in the following sections.

Deliverable 1

My first deliverable is an example program implemented in TensorFlow. This program used softmax to recognize handwritten digits from the MNIST dataset. Prior to implementation, I studied machine learning, neural network and Python.

The MNIST Dataset

The MNIST data is hosted on Yann LeCun's website. MNIST consists of 70000 data point. Each data point consists of a label and an image of a handwritten digit. The label is a digit from 0 to 9. The image is 28 pixels by 28 pixels.

I split the 70000 data points to three groups. 55,000 data points in training set. 10,000 data points in test training set. 5,000 in validation set.

We can interpret the image as a 2D matrix. One process of the data in the program is flattening this 2D matrix to a 1D array of $28 \times 28 = 784$ numbers. This operation retains the feature of the image and keep it consistent with the image and label.

Softmax Regression

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes. In logistic regression, we assumed that the labels were binary: $y(i) \in \{0,1\}$. Softmax regression allows us to handle $y(i) \in \{1,\ldots,K\}$ where K is the number of classes [4], which is the case in this practice.

The Softmax function is given by

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

$\sigma(z)_j$ is in the range (0, 1), and $\sum\sigma(z)_j = 1$.

In this problem,

$$z_j = \sum_j W_{i,j} x_j + b_i$$

where W_i is the weights and b_i is the bias for class i, and j is an index for summing over the pixels in our input image.

Implementation

x is the flattened array of the image feature. W is the weight. b is the bias which is independent of the input. y is the classification outcome from the softmax model.

```
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b
```

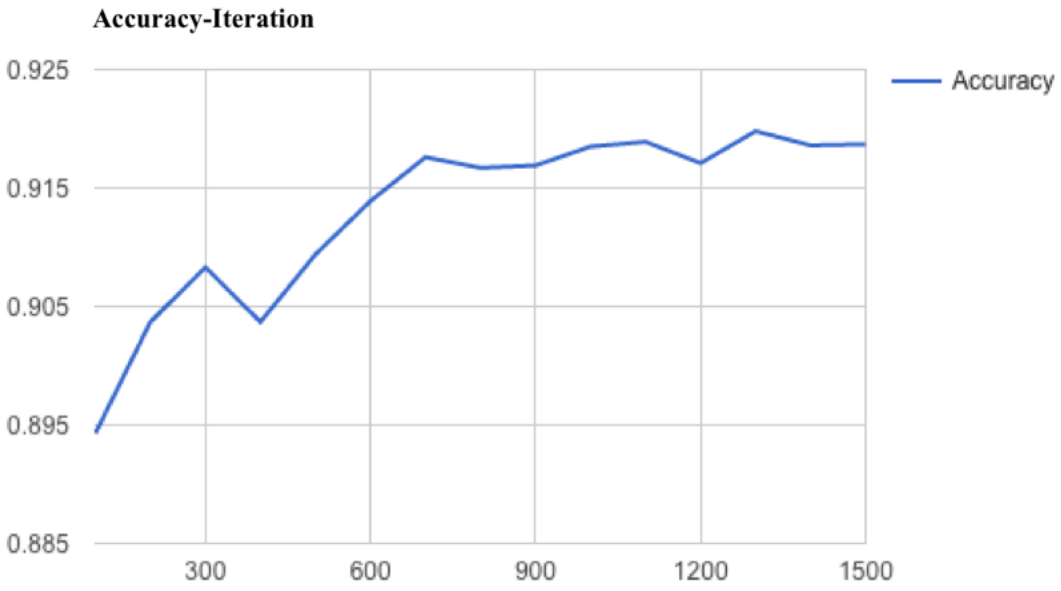
```
y_ = tf.placeholder(tf.float32, [None, 10])
```

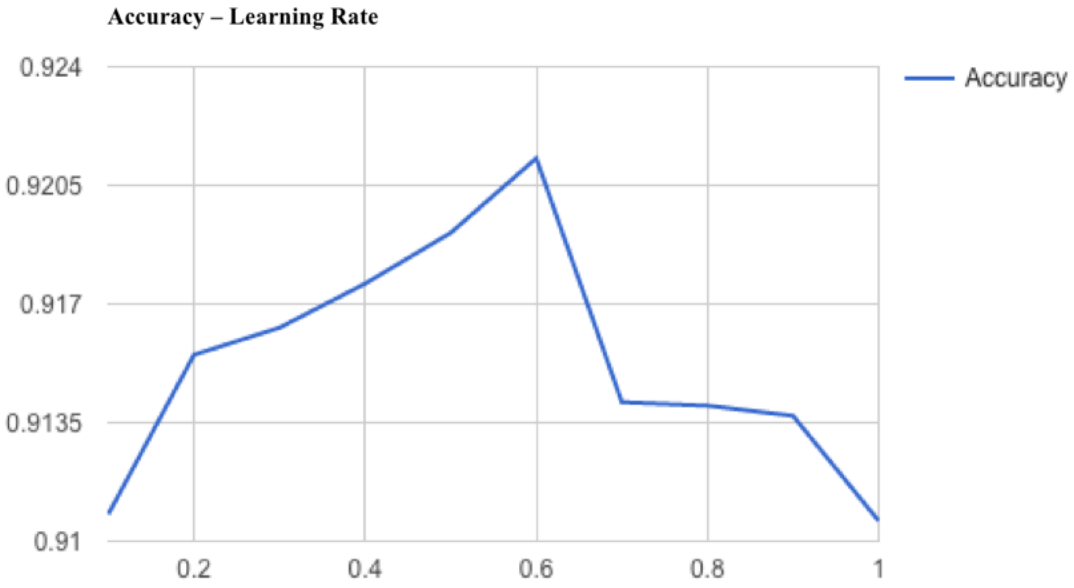
```
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
train_step = tf.train.GradientDescentOptimizer(FLAGS.learning_rate).minimize(cross_entropy)
```

Manually Visualize Learning

I trained the model with different numbers of gradient descent iterations and different learning rates, ad shown in the graphs below. The accuracy was between

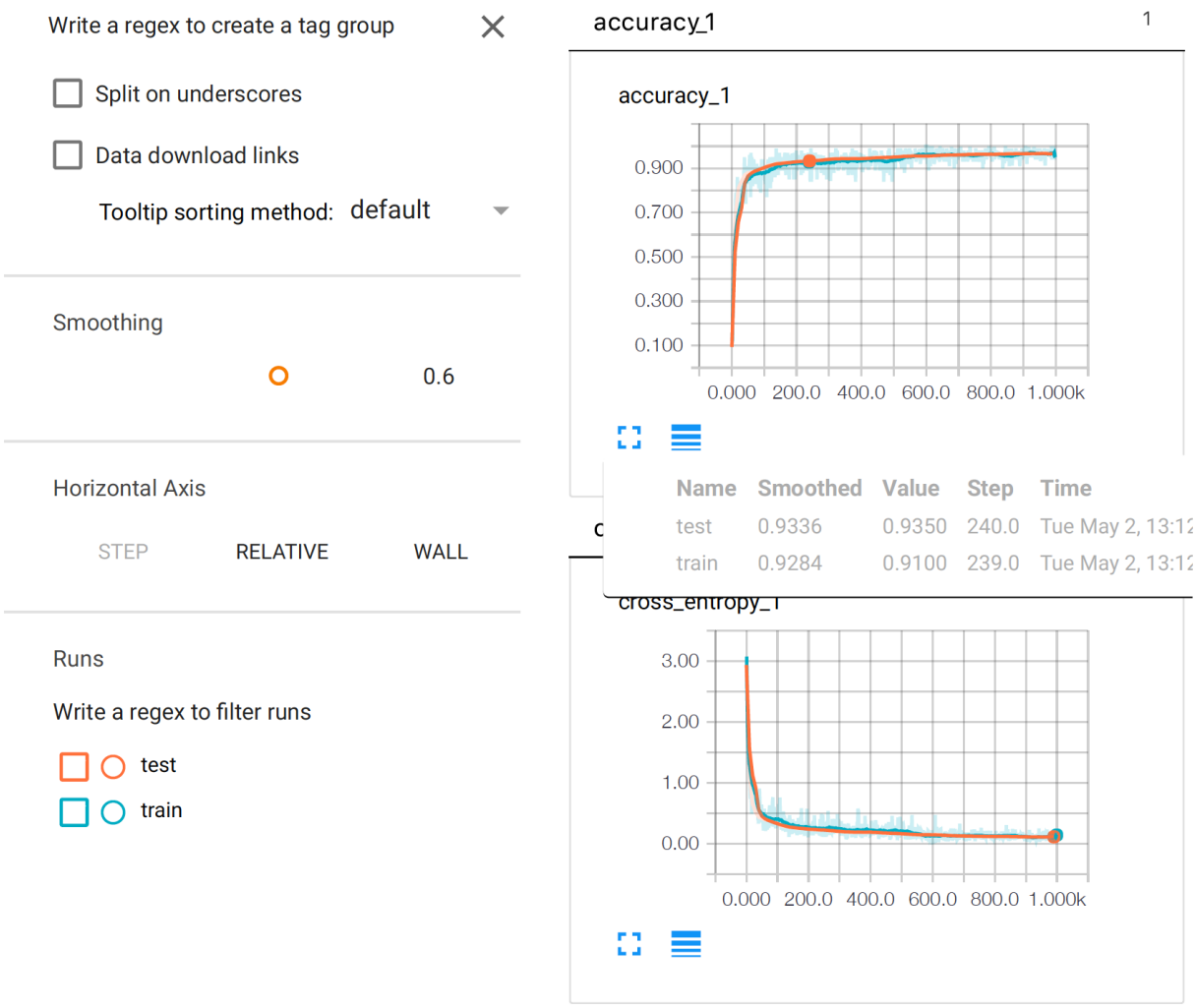
89.5% to 92%.


Accuracy-Iteration
0.925
0.915
0.905
0.895
0.885
300
600
900
1200
1500
Accuracy



TensorBoard

Beside manually visualizing the diagram above, there is a component named TensorBoard that facilitates visualized learning. It will be helpful to utilize this component in the future. Thus, I experimented using TensorBoard as well. TensorBoard operates by reading TensorFlow events files, which contain summary data that you can generate when running TensorFlow. There are various summary operations, such as scalar, histogram, merge_all, etc [5]. An example diagram created by TensorBoard shows as below.


Write a regex to create a tag group
Split on underscores
Data download links
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
test
train
accuracy_1
1
accuracy_1
0.900
0.700
0.500
0.300
0.100
0.000
200.0
400.0
600.0
800.0
1.000k
Name
Smoothed
Value
Step
Time
test
0.9336
0.9350
240.0
Tue May 2, 13:12
train
0.9284
0.9100
239.0
Tue May 2, 13:12
cross_entropy_1
3.00
2.00
1.00
0.00
0.000
200.0
400.0
600.0
800.0
1.000k

Deliverable 2

Word embedding is a parameterized function mapping words in some language to high-dimensional vectors. Methods to generate this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix, probabilistic models, and explicit representation in terms of the context in which words appear. My literature reviews focus on learning word embedding by a neural network.

Introduction to Word Embedding

A word embedding is sometimes called a word representation or a word vector. It maps words to a high dimensional vector of real number. The meaningful vector learned can be used to perform some task.

$$\text{word} \rightarrow R^n$$

W(“cat”) = [0.3, -0.2, 0.7, …]

W(“dog”) = [0.5, 0.4 -0.6, …]

Visualizing the representation of a word in a two-dimension projection, we can sometimes see its “intuitive sense”. For example, looking at Figure 1, digits are close together, and, there are linear relationship between words.

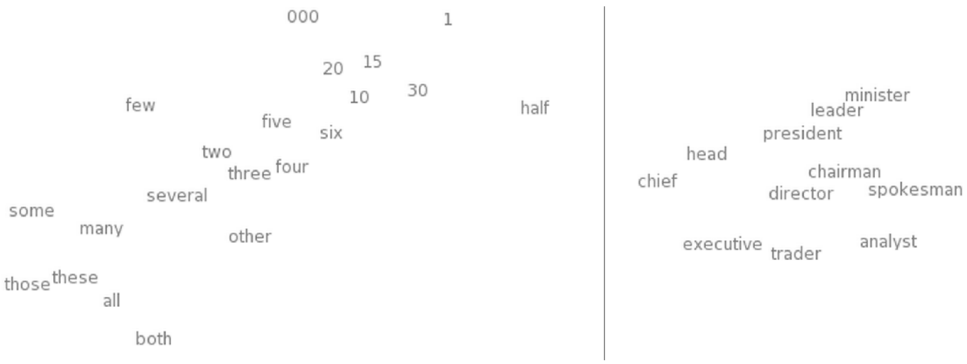
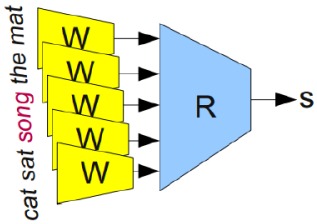


Figure 1. Two-dimension Projection [3]

One Application Example

One task we might train a network for is predicting whether a 5-gram (sequence of five words) is 'valid.' To predict these values accurately, the network needs to learn good parameters for both W and R [3].



Modular Network to
determine if a 5-gram is 'valid'
(From Bottou (2011)
(http://arxiv.org/pdf/1102.1808v

$$R(W(\text{``cat''}), W(\text{``sat''}), W(\text{``on''}), W(\text{``the''}), W(\text{``mat''})) = 1$$

$$R(W(\text{``cat''}), W(\text{``sat''}), W(\text{``song''}), W(\text{``the''}), W(\text{``mat''})) = 0$$

Apply to The Project

It turns out, though, that much more sophisticated relationships are also encoded in word embedding [3]. In this project, one possible approach is using pre-trained word vector W and works on R to find disambiguate words. Another possible approach is working on both W and R. The fully analysis, implement and evaluation of the learning algorithm will be done in CS298.

Deliverable 3

In Deliverable 3, I extracted pages of ambiguous words from Wikipedia data. Since the Wikipedia data is a huge bz2 file, I extracted pages of ambiguous words while decompressing the Wikipedia dump on the fly.

The Dictionary of Ambiguous Word

A word list was primarily extracted from the disambiguation data on http://wiki.dbpedia.org/downloads-2016-04#h26493-2. I created a main dictionary based on this file. There are plenty pages with words in the main dictionary as the title is redirected to another page. Thus, an additional dictionary is created while decompressing the Wikipedia bz2 file with the main dictionary as a filter. Then, the additional dictionary is used as a filter to decompress the Wikipedia bz2 file again.

Preprocessing Data

By decompressing the Wikipedia bz2 file twice, a file with only disambiguation page was output. Further data processing is needed in future after the requirement of data is clearer in CS298.

Conclusion

During CS297, I started by learning machine learning, neural networks, TensorFlow, and Python. I practiced on programming to solidify my understanding and gain experience. Literature review on disambiguation led me to understand the state of the art. Literature review on word embedding helped me understand what it is and how it can be used in my project. In CS297, I also started data preprocessing, however, most of the work of data processing will not be done until I figure out the requirements of the data when I work out how to create the model. In CS 298, I will work on how to define and build the model. To do this, I will need to gather a deeper understanding of how word embedding and neural networks work. Data processing will also be an important part of CS298 as well. Meanwhile, I will research on how to evaluation the outcome of the model as well.

References

Last Name, F. M. (Year). Article Title. *Journal Title*, Pages From - To.

1. Cucerzan, Silviu. (2007). Large-Scale Named Entity Disambiguation Based on Wikipedia Data

2. Bengio, Yoshua. and Ducharme, Réjean. and Vincent, Pascal. and Pascal, Christian (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, Pages 1137 - 1155.

3. Olah, Christopher. (2014). "Deep Learning, NLP, and Representations",

http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

4. UFLDL Tutorial, http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

5. TensorFlow Tutorial, https://www.tensorflow.org/get_started/summaries_and_tensorboard