Lecture 6: Neural Networks for Representing Word Meaning

Mirella Lapata
School of Informatics
University of Edinburgh
mlap@inf.ed.ac.uk
February 7, 2017

Logistic Regression

- Input is a feature vector, output is one (binary classification) or many (multinomial distribution)
- Weights matrix (or vector) directly connects inputs and output
- Trained by gradient descent (neural network; no hidden units).

Logistic Functions

- Logistic Function: $f(x) = \frac{L}{1+e^{-k(x-x_0)}}$
- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Softmax function:
  
  $P(y = j|x) = \frac{e^{x^Tw_j}}{\sum_{k=1}^{K} e^{x^Tw_k}}$

- $e$ natural logarithm base, $x_0$ the $x$-value of the sigmoid's midpoint, $L$ the curve's maximum value, $k$ steepness of curve
- Sigmoid is special case of the logistic function
- Softmax is a generalization of the logistic function
- $x^Tw$ denotes the inner product of $x$ and $w$

Neural Networks

- $W$: input weights, matrix
- $I$: input signal, feature vector (one per example)
- Activation function: sigmoid, tanh
Training of Neural Networks

Forward Pass
- Input signal is presented first
- Hidden layer state is computed (vector times matrix operation and non-linear activation)
- Outputs are computed (vectors times matrix operation and usually non-linear activation)

Backpropagation
- To train the network, we need to compute gradient of the error
- The gradients are sent back using the same weights that were used in the forward pass

Gradient Descent
- Batch gradient descent: compute gradient from batch of \( N \) training examples
- Stochastic gradient descent: compute the gradient from 1 training example each time
- Minibatch: compute the gradient from minibatch of \( M \) training examples (\( M > 1, M < N \))

Learning Rate:
- Controls how much we change the weights
- Too little value will result in long training time, too high value will erase previously learned patterns
- We start with high learning rate and reduce it during training

Training Epochs: several passes over the training data are often performed (epoch: number of iterations over the data set in order to train the neural network).

Regularization:
- Network often overfits (it fails to generalize at test time)
- High weights are used to model only some small subset of data
- We can try to force the weights to stay small during training to avoid this problem (L1 & L2 regularization)

Model Parameters

Choice of the hyper-parameters has to be done manually:
- Type of activation function
- Choice of architecture (how many hidden layers, their sizes)
- Learning rate, number of training epochs
- What features are presented at the input layer
- How to regularize

N-grams

N-gram-based language models are often used to compute the probability of a sentence \( W \):

\[
P(W) = \prod_i (w_i | w_1 \ldots w_{i-1})
\]

Often simplified to trigrams:

\[
P(W) = \prod_i (w_i | w_{i-2}, w_{i-1})
\]

\[
P(this \ is \ a \ sentence) = P(this) \times P(is|this) \times P(a|this, is) \times P(sentence|is, a)
\]

\[
P(a|this, is) = \frac{C(this \ is \ a)}{C(this \ is)}
\]
One-hot Representations

Simple way how to encode discrete concepts, such as words; also known as 1-of-$N$ where $N$ would be the size of the vocabulary.

Let's consider a vocabulary containing the following words:

- Monday
- Tuesday
- is
- a
- today

The one-hot representations for these words would be:

- Monday = [1 0 0 0 0]
- Tuesday = [0 1 0 0 0]
- is = [0 0 1 0 0]
- a = [0 0 0 1 0]
- today = [0 0 0 0 1]

Bag-of-Words Representations

Ignores word order, sum of one-hot vectors. Can be extended to bag-of-$N$-grams to capture local ordering of words.

Let's consider the same vocabulary:

- Monday
- Tuesday
- is
- a
- today

We can represent sentences like:

- today is a Monday = [1 0 1 1 1]
- today is a Tuesday = [0 1 1 1 1]
- is a Monday today = [1 0 1 1 1]

A Basic Neural Language Model

![Diagram of neural language model]

- Bigram neural language model
- Previous word predicts current word via hidden layer
- As many outputs as there are words in the vocabulary
- Model learns compressed, continuous representations of words (usually the matrix of weights between input and hidden layer)

Another Neural Language Model

- Bengio et al. (2003, JMLR)
- Input layer, projection layer, hidden layer and output layer
- The projection layer is linear
- The hidden layer is non-linear
- Softmax at the output computes probability distribution over the whole vocabulary ($g(o) = e^{o_k} / \sum_k e^{o_k}$)
- Model is computationally very expensive
Continuous bag-of-words model (CBOW)

- Mikolov et al. (2013, ICLR)
- CBOW adds inputs from words within short window to predict the current word
- The weights for different positions are shared
- Computationally much more efficient than normal NNLM
- The hidden layer is just linear

Skip-gram NNLM

- We can reformulate the CBOW model by predicting surrounding words using the current word
- Find word representations useful for predicting surrounding words in a sentence or document.

The Hidden Layer

We wish to learn word vectors with 300 features. Hidden layer is a weight matrix with 10,000 rows (one for every word in vocabulary) and 300 columns (one for every hidden neuron).
Skip-gram Details

The basic Skip-gram formulation defines \( p(c|w; \theta) \) using the softmax function:

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

where \( v_c \) and \( v_w \) are vector representations for \( c \) and \( w \) and \( C \) is the set of all available contexts.

- Each word \( w \in W \) is associated with vector \( v_w \in \mathbb{R}^d \)
- Each context \( c \in C \) is associated with vector \( v_c \in \mathbb{R}^d \)
- \( W \) words vocabulary, \( C \) contexts vocabulary
- \( d \) embedding dimensionality
- Vector components are latent (parameters to be learnt).
- Objective function maximizes the probability of corpus \( D \) (set of all word and context pairs extracted from text):

\[
\argmax_{\theta} \prod_{(w,c) \in D} p(c|w; \theta)
\]

Training

- Stochastic gradient descent and backpropagation
- It is useful to sub-sample the frequent words (e.g., the, is, a)
- Words are thrown out proportional to their frequency (makes things faster, reduces importance of frequent words like IDF)
- Non-linearity does not seem to improve performance of these models, thus the hidden layer does not use activation function
- **Problem:** very large output layer - size equal to vocabulary size, can easily be in order of millions (too many outputs to evaluate)
- **Solution:** negative sampling (also Hierarchical softmax)
Negative Sampling

- Instead of propagating signal from the hidden layer to the whole output layer, only the output neuron that represents the positive class + few randomly sampled neurons are evaluated.
- The output neurons are treated as independent logistic regression classifiers.
- This makes the training speed independent of the vocabulary size (can be easily parallelized).

Model Evaluation

<table>
<thead>
<tr>
<th>Type of relationship</th>
<th>Word Pair 1</th>
<th>Word Pair 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common capital city</td>
<td>Athens</td>
<td>Greece</td>
</tr>
<tr>
<td></td>
<td>Astana</td>
<td>Karazakhstan</td>
</tr>
<tr>
<td></td>
<td>Oslo</td>
<td>Harare</td>
</tr>
<tr>
<td></td>
<td>Norway</td>
<td>Zimbabwe</td>
</tr>
<tr>
<td>Currency</td>
<td>Angola</td>
<td>kwanza</td>
</tr>
<tr>
<td></td>
<td>Harare</td>
<td>Iran</td>
</tr>
<tr>
<td>City-in-state</td>
<td>Chicago</td>
<td>Illinois</td>
</tr>
<tr>
<td>Man-Woman</td>
<td>brother</td>
<td>sister</td>
</tr>
<tr>
<td></td>
<td>grandson</td>
<td>granddaughter</td>
</tr>
<tr>
<td>Adjective to adverb</td>
<td>apparent</td>
<td>apparently</td>
</tr>
<tr>
<td></td>
<td>possibly</td>
<td>impossibly</td>
</tr>
<tr>
<td></td>
<td>rapid</td>
<td>rapidly</td>
</tr>
<tr>
<td>Opposite</td>
<td>great</td>
<td>greater</td>
</tr>
<tr>
<td>Comparative</td>
<td>easy</td>
<td>easiest</td>
</tr>
<tr>
<td>Superlative</td>
<td>read</td>
<td>reading</td>
</tr>
<tr>
<td>Present Participle</td>
<td>think</td>
<td>thinking</td>
</tr>
<tr>
<td>Nationality adjective</td>
<td>Switzerland</td>
<td>Swiss</td>
</tr>
<tr>
<td>Past tense</td>
<td>walking</td>
<td>walked</td>
</tr>
<tr>
<td></td>
<td>swimming</td>
<td>swam</td>
</tr>
<tr>
<td>Plural nouns</td>
<td>mouse</td>
<td>mice</td>
</tr>
<tr>
<td></td>
<td>dollar</td>
<td>dollars</td>
</tr>
<tr>
<td>Plural verbs</td>
<td>work</td>
<td>works</td>
</tr>
<tr>
<td></td>
<td>speak</td>
<td>speaks</td>
</tr>
</tbody>
</table>

- 20K questions, 6B words, 1M words vocabulary
- Find word closest to question word (e.g., Greece), consider it correct if it matches the answer.

Negative Sampling Objective

- Let $D$ denote dataset of observed $(w, c)$ pairs
- $p(D = 1|w, c)$ denotes probability that $(w, c)$ came from $D$
- $p(D = 0|w, c) = 1 - p(D = 1|w, c)$ that it didn’t

\[
\begin{align*}
\text{argmax}_{\theta} & \prod_{(w, c) \in D} p(D = 1|w, c; \theta) \prod_{(w, c) \in D'} p(D = 0|w, c; \theta) \\
\text{argmax}_{\theta} & \sum_{(w, c) \in D} \log p(D = 1|w, c; \theta) + \sum_{(w, c) \in D'} \log(1 - p(D = 1|w, c; \theta)) \\
\text{argmax}_{\theta} & \sum_{(w, c) \in D} \log \frac{1}{1+e^{-vc \cdot vw}} + \sum_{(w, c) \in D'} \log \frac{1}{1+e^{-vc \cdot vw}}
\end{align*}
\]

See Goldberg and Levy (2014) for full derivation.

Model Evaluation

Models trained on the same data (640-dimensional word vectors).

<table>
<thead>
<tr>
<th>Model Architecture</th>
<th>Semantic-Syntactic Word Relationship test set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Semantic Accuracy [%]</td>
</tr>
<tr>
<td>RNNLM</td>
<td>9</td>
</tr>
<tr>
<td>NNLM</td>
<td>23</td>
</tr>
<tr>
<td>CBOW</td>
<td>24</td>
</tr>
<tr>
<td>Skip-gram</td>
<td>55</td>
</tr>
</tbody>
</table>

Can you think of additional baseline comparisons?
Model Evaluation

Subtracting two word vectors, and add the result to another word.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Nearest token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris - France + Italy</td>
<td>Rome</td>
</tr>
<tr>
<td>bigger - big + cold</td>
<td>colder</td>
</tr>
<tr>
<td>sushi - Japan + Germany</td>
<td>bratwurst</td>
</tr>
<tr>
<td>Cu - copper + gold</td>
<td>Au</td>
</tr>
<tr>
<td>Windows - Microsoft + Google</td>
<td>Android</td>
</tr>
<tr>
<td>Montreal Canadiens - Montreal + Toronto</td>
<td>Toronto Maple Leafs</td>
</tr>
</tbody>
</table>

Check more examples out at:
http://rare-technologies.com/word2vec-tutorial/#app

Summary

Don’t count predict! Baroni et al. (ACL, 2014)

Why Does Skip-gram Work?

- We don’t really know!
- The objective tries to increase the quantity $v_w \cdot v_c$ for good word-context pairs, and decrease it for bad ones.
- Words that share many contexts will be similar to each other.
- Lots of parameters here too: contexts, subsampling, rare word pruning, dimension of vectors.

Resources

- Word2vec available at https://code.google.com/p/word2vec/
- Tool for training the word vectors using CBOW and skip-gram architectures, supports both negative sampling and hierarchical softmax
- Optimized for very large datasets (>billions of training words)
- Pre-trained on large datasets (100B words)