Distributional Semantics

Lecture 4. Prediction-based Distributional Models

Florian Gouret

March 16th, 2019

Lecture Plan

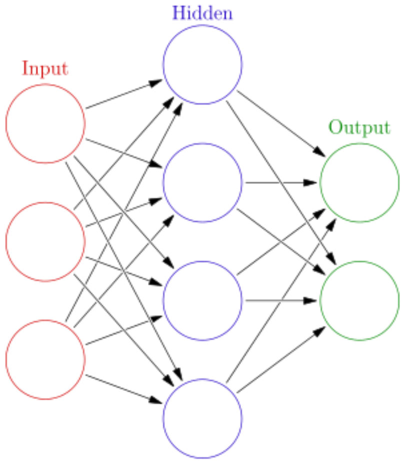
Lecture Plan

Introduction

Introduction: What do you think?

Prediction-based Distributional Models?

Neural Networks



- **[Artificial] neural network** consists of **layers**: input layer, output layer, and hidden layer(s).
- Each layer is a group of neurons. Neurons from different layers are connected.
- The input value of a neuron on the next layer is a weighted sum of the output values of the neurons on a previous layer: $h_j^{i+1} = \sum_k w_{kj}^i \cdot \sigma^i(h_k^i)$.
- σ^i are called **activation functions**, e. g. $\sigma^i(x) = x$ (identity function) or $\sigma^i(x) = x \cdot [x > 0]$ (ReLU).
- w_{kj}^i are called **weights** (parameters) and are usually **learned** during the model **training**, i. e. unknown a-priori.
- In a training process, model tries to minimize the **loss function** (MSE, log-likelihood, etc.) w.r.t. to its **parameters**.

History

History: Basement

To understand how much today prediction-based models evolved we have to first look at:

- Statistical Language Modeling
- Neural Probabilistic Language Model (NPLM)
- SENNA
- HLBL

Statistical Language Modeling

A **statistical model of language** can be represented by the **conditional probability** of the **next word** given all the previous ones, since

$$\hat{P}(w_1^T) = \prod_{t=1}^{T} \hat{P}(w_t \mid w_1^{t-1})$$

where w_t is the t-th word, and writing sub-sequence $w_i^j = (w_i, w_{i+1}, \ldots, w_{j-1}, w_j)$.

n-**gram models** construct (*n*-dimensional) **tables of conditional probabilities** for the next word, for each one of a large number of *contexts*, i. e. combinations of the last $n-1$ words:

$$\hat{P}(w_t \mid w_1^{t-1}) \approx \hat{P}(w_t \mid w_{t-n+1}^{t-1}).$$

Using this way of representing words and their probability, we would need tables with $O\left(|\mathcal{W}|^n\right)$ elements... This is called **curse of dimensionality**.
In practice, $n = 1, 2, 3$.

Distributed Representation of Words

A **distributed representation** is dense, low-dimensional, and real-valued. Distributed word representations are called **word embeddings**.
Each dimension of the embedding represents **a latent feature of a word**, hopefully capturing useful syntactic and semantic properties. A **distributed representation** is **compact**, in the sense that it can represent an exponential number of clusters in the number of dimensions.

A Neural Probabilistic Language Model

Facing the curse of dimensionality

A Neural Probabilistic Language Model

A **neural probabilistic language model (NPLM)** was proposed Bengio et al. in 2003.
The approach is as follows:

1. **associate** with **each word** in the vocabulary an **embedding** (a real-valued vector in $\mathbb{R}^d$);
2. **express** the **joint probability** function of word sequences in terms of the **embeddings** of these words in the sequence,
3. learn **simultaneously** the word **embeddings** and **parameters** of that probability function.

A Neural Probabilistic Language Model

The aim is to learn a good model for $f(w_t, \dots, w_{t-n+1}) = \hat{P}(w_t \mid w_1^{t-1})$.
To do so, the function is decomposed into two parts:

1. A **mapping** C from **any element** w of $\mathcal{W}$ to a **real vector** $C(w) \in \mathbb{R}^d$ (the distributed feature vectors associated with each word in the vocabulary). In practice, C is represented by a $|\mathcal{W}| \times d$ matrix of free parameters.
2. The **probability function over words** (expressed with C):
   $g : (i, C(w_{t-1}), \dots, C(w_{t-n+1})) \to \hat{P}(w_t = w^{(i)} \mid w_1^{t-1})$;

$$f(w^{(i)}, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1})).$$

A Neural Probabilistic Language Model

Loss function:

$$L = \frac{1}{T}\sum_{t=1}^{T} \log f(w_t, w_{t-1}, \dots, w_{t-n+1} \mid \boldsymbol{\Theta}) + R(\boldsymbol{\Theta}) \longrightarrow \min_{\boldsymbol{\Theta}},$$

where $\boldsymbol{\Theta}$ is a set of all parameters, and R is a regularizer.

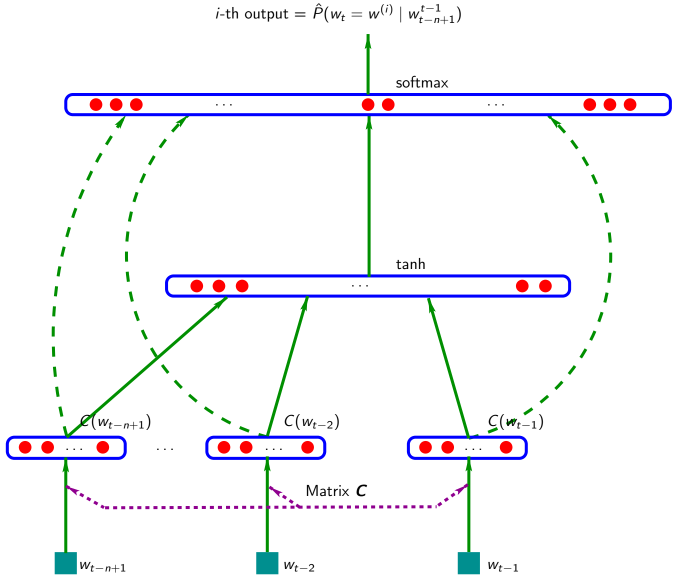
The probabilities are expressed by applying **SoftMax** to the **unnormalized log-probabilities** y_w:

$$\hat{P}(w_t \mid w_1^{t-1}) = \frac{\exp y_{w_t}}{\sum_{w \in \mathcal{W}} \exp y_w},$$

$$\boldsymbol{y} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{x} + \boldsymbol{U} \tanh(\boldsymbol{d} + \boldsymbol{H}\boldsymbol{x}),$$

where the hyperbolic tangent tanh is applied element by element, $\boldsymbol{W}$ is optionally zero (no direct connections), and $\boldsymbol{x}$ is the word features layer activation vector, which is the concatenation of the input word features from the matrix $\boldsymbol{C}$:

$$\boldsymbol{x} = [C(w_{t-1}), C(w_{t-2}), \dots, C(w_{t-n+1})].$$

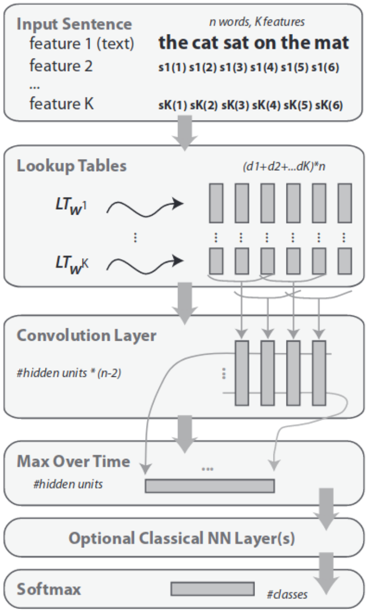


i-th output $= \hat{P}(w_t = w^{(i)} \mid w_{t-n+1}^{t-1})$
softmax
. . .
tanh
$C(w_{t-n+1})$
$C(w_{t-2})$
$C(w_{t-1})$
Matrix $\boldsymbol{C}$
w_{t-n+1}
w_{t-2}
w_{t-1}

SENNA

Collebert and Weston 2008: a **neural language model** that could be **trained over billions of words**, because the **gradient of the loss** was computed **stochastically** over a small sample of possible outputs, in a spirit similar to Bengio.
The architecture is as follows.

1. for each training update, we read an n-gram $w_{t-n+1}, \dots, w_t$ from the corpus and predict w_t using previous $n-1$ words.
2. the input sequence is transformed into embeddings $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{n-1} := \boldsymbol{W}_{w_{t-n+1},:}, \dots, \boldsymbol{W}_{w_{t-1},:}$
3. apply a convolution layer: $\boldsymbol{o}_s = \sum_{j=1-s}^{n-1-s} \boldsymbol{L}_j \boldsymbol{x}_{s+j}$
4. use max-pooling over time: $\boldsymbol{m} = \max_{s=1}^{n-1} \boldsymbol{o}_s$
5. apply a layer that gives the output of size $|\mathcal{W}|$ (e. g. linear layer): $\boldsymbol{y} = \boldsymbol{b} + \boldsymbol{H}\boldsymbol{m}$


Input Sentence
n words, K features
feature 1 (text)
the cat sat on the mat
feature 2
s1(1) s1(2) s1(3) s1(4) s1(5) s1(6)
...
feature K
sK(1) sK(2) sK(3) sK(4) sK(5) sK(6)
Lookup Tables
(d1+d2+...dK)*n
LT_{W^1}
LT_{W^K}
Convolution Layer
#hidden units * (n-2)
Max Over Time
#hidden units
Optional Classical NN Layer(s)
Softmax
#classes

HLBL

Log-bilinear model (LBL) is a probabilistic and linear neural model.
To **predict the next word** w_t given the context $w_{t-n+1}, \dots, w_{t-1}$, the model **computes** the **predicted feature vector** $\hat{u}_t$ for the next word by **linearly combining the context word feature vectors**:

$$\hat{\boldsymbol{u}}_t = \sum_{j=t-n+1}^{t-1} c_{j-t+n} \boldsymbol{u}_{w_j},$$

where c_i is the weight associated with the context position i.
Finally,

$$\hat{P}(w_t = w \mid w_{t-n+1}^{t-1}) = \frac{\exp\left(\hat{\boldsymbol{u}}_t^\top \boldsymbol{u}_w + b_w\right)}{\sum_{w' \in \mathcal{W}} \exp\left(\hat{\boldsymbol{u}}_t^\top \boldsymbol{u}_{w'} + b_{w'}\right)}.$$

Here b_w is the bias for word w, which is used to capture the context-independent word frequency.

HLBL

Mhin and Hinton (2009) speed up model evaluation during training and testing by using a hierarchy to exponentially filter down the number of computations that are performed. The model, combined with this optimization, is called the **hierarchical log-bilinear (HLBL)** model.
The **probability of the next word** being w is the **probability of making the sequence of binary decisions** specified by the word's code, given the context.

$$\hat{P}(w_t = w \mid w_{t-n+1}^{t-1}) = \prod_i \hat{P}(d_i \mid \boldsymbol{q}_i, w_{t-n+1}^{t-1})$$

where d_i is i-th digit in the code for word w, and $\boldsymbol{q}_i$ is the feature vector for the i-th node in the path corresponding to that code.
The probability of each decision is given by

$$\hat{P}(d_i = 1 \mid \boldsymbol{q}_i, w_{t-n+1}^{t-1}) = \sigma(\hat{\boldsymbol{u}}_t^\top \boldsymbol{q}_i + b_i)$$

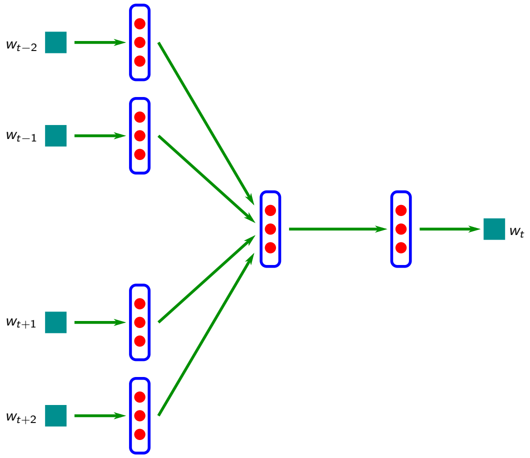
where $\sigma(x) = 1/(1 + e^{-x})$ is the logistic function, and b_i is the node's bias that captures the context-independent tendency to visit the left child when leaving this node.

Word2Vec

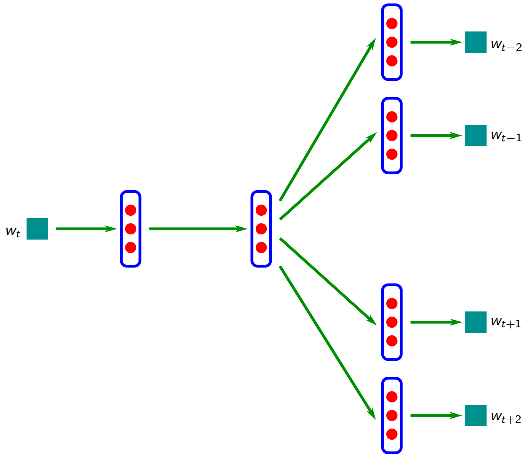
Word2Vec

Two particular models for learning word representations that can be **efficiently trained** on large amounts of text data are **Skip-gram (SG)** and **Continuous bag-of-words model (CBOW)** introduced by Mikolov et al. in 2013.

Word2Vec



Architecture of the CBOW model



Architecture of the Skip-gram model

Continuous Skip-gram Model

Given a sequence of training words $w_1, w_2, w_3, \ldots, w_T$, the objective of the Skip-gram model is to maximize the average log probability

$$L = \frac{1}{T} \sum_{t=1}^{T} \sum_{c \in \mathcal{C}_t} \log P(c \mid w_t),$$

where $\mathcal{C}_t = \cup_{j=-win, j\neq 0}^{win} w_{t+j}$ and *win* is the size of the training window (which can be a function of the center word w_t).

In the Skip-gram model, every word w is associated with two learnable parameter vectors, $\boldsymbol{u}_w$ and $\boldsymbol{v}_w$. They are the "input" and "output" vectors of the w respectively. The probability of correctly predicting the context word c given the central word w_t is defined as

$$P(c \mid w_t) = \frac{\exp\left(\boldsymbol{u}_{w_t}^{\top} \boldsymbol{v}_c\right)}{\sum_{c' \in \mathcal{W}} \exp\left(\boldsymbol{u}_{w_t}^{\top} \boldsymbol{v}_{c'}\right)}.$$

Hierarchical Softmax

$$P(c \mid w_t) = \frac{\exp\left(\boldsymbol{u}_{w_t}^\top \boldsymbol{v}_c\right)}{\sum_{c' \in \mathcal{W}} \exp\left(\boldsymbol{u}_{w_t}^\top \boldsymbol{v}_{c'}\right)}.$$

This formulation is impractical because the cost of computing $\nabla \log P(c \mid w_t)$ is proportional to the number of words in the vocabulary $|\mathcal{W}|$ (which can be easily in order of millions).

The **hierarchical softmax** uses a **binary tree representation** of the output layer with the $|\mathcal{W}|$ words as its leaves and, for each node, explicitly represents the relative probabilities of its child nodes. These define a **random walk** that assigns probabilities to words.

Hierarchical Softmax

Each word w can be reached from the root. Let $n(w,j)$ be the j-th node on the path from the root to w, and let $L(w)$ be the length of this path, so $n(w,1) = \mathrm{root}$ and $n(w, L(w)) = w$.
For any inner node n, let $\mathrm{ch}(n)$ be an arbitrary fixed child of n and let $[\![x]\!]$ be 1 if x is true and -1 otherwise. Then the hierarchical softmax defines $P(c \mid w_t)$ as follows:

$$P(c \mid w_t) = \prod_{j=1}^{L(c)-1} \sigma\left([\![n(c,j+1) = \mathrm{ch}(n(c,j))]\!] \boldsymbol{u}_{w_t}^{\top} \boldsymbol{v}_{n(c,j)}\right)$$

where $\sigma(x) = 1/(1+e^{-x})$.
This implies that the cost of computing $\log P(c \mid w_t)$ and $\nabla \log P(c \mid w_t)$ is proportional to $L(c)$, which on average is no greater than $\log_2 |\mathcal{W}|$.

Negative Sampling

The **noise contrastive estimation** (NCE) (2012) was introduced by Gutmann and Hyvarinen and applied to language modeling by Mnih and Teh. NCE follows the idea that a **good model** should be able to **differentiate data** from **noise** by means of **logistic regression**.
The Skip-gram model is only concerned with learning high-quality vector representations, so we are free to simplify NCE as long as the vector representations retain their quality. We define **negative sampling** by the objective

$$\log \sigma(\boldsymbol{u}_w^\top \boldsymbol{v}_c) + \sum_{j=1}^{k} \mathbb{E}_{c_{n,j} \sim P_n} \log \sigma(-\boldsymbol{u}_w^\top \boldsymbol{v}_{c_{n,j}})$$

which is used to replace every $\log P(c \mid w_t)$ term in the Skip-gram objective. Thus the task is to distinguish the target word c from draws c_n from the noise distribution $P_n(w)$ using logistic regression, where there are k negative samples for each data sample.

Subsampling of Frequent Words

In very large corpora, the most frequent words can easily occur hundreds of millions of times (e.g., “in”, “the”, and “a”). Such words usually provide less information value than the rare words.
To counter the imbalance between the rare and frequent words, the authors used a simple subsampling approach: each word w_t in the training set is discarded with probability computed by the formula

$$P(w_t) = 1 - \sqrt{\frac{r}{\#(w_t)}}$$

where r is a chosen threshold, typically around 10^{-5}.

Connection to Count-based Models

Connection to Count-based Models

SGNS embeds both words and their contexts into a low-dimensional space $\mathbb{R}^d$, resulting in the word and context matrices $\boldsymbol{U}$ and $\boldsymbol{V}$. Let $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{V}^\top$. An implicit matrix $\boldsymbol{M}$ of dimensions $|\mathcal{W}| \times |\mathcal{W}|$ is factorized into two smaller matrices, and $M_{w,c} = \boldsymbol{u}_w^\top \boldsymbol{v}_c$.

So, each cell of matrix $\boldsymbol{M}$ contains a quantity $f(w,c)$ reflecting the strength of association between that particular word-context pair. What can we say about the association function $f(w,c)$? In other words, which matrix is SGNS factorizing?

Levy and Goldberg (2014) has showh that $M_{w,c} = f(w,c) = \log 2 \cdot PMI_{w,c} - \log k$, where k is a number of negative samples.

Analogical Reasoning with Word Embeddings

Intuition

To compare the quality of different versions of word vectors, researchers typically used a table showing example words and their most similar words, and understand them intuitively.
Somewhat surprisingly, these questions can be answered by performing simple algebraic operations with the vector representation of words. To find a word that is similar to *small* in the same sense as *biggest* is similar to *big*, we can simply compute vector

$$v^* = v_{biggest} - v_{big} + v_{small}.$$

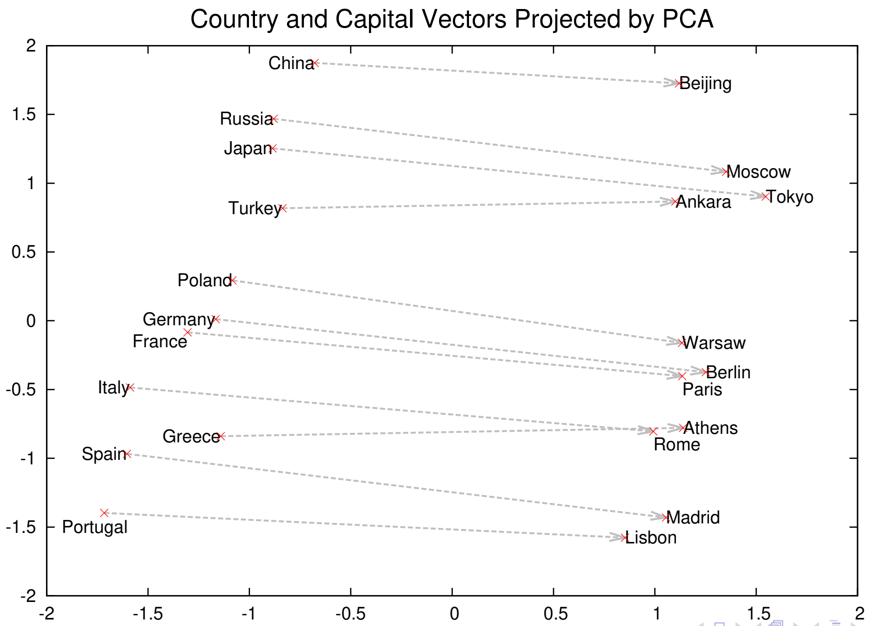
The relationship is defined by subtracting two word vectors, and the result is added to another word. Thus for example,

$$v_{Paris} - v_{France} + v_{Italy} = v_{Rome}.$$

Examples of the Learned Relationships

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Examples of the word pair relationships, using the best word vectors from Skip-gram model trained on 783M words with 300 dimensionality.


Country and Capital Vectors Projected by PCA
China
Beijing
Russia
Japan
Moscow
Tokyo
Turkey
Ankara
Poland
Germany
France
Warsaw
Berlin
Paris
Italy
Greece
Athens
Rome
Spain
Madrid
Portugal
Lisbon
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Questions?

Seminar

Word2Vec for Text Classification

1. TODO
2. Homework: implementation of CBOW or SG

TODO

▸ Link