Logical and Computational Structures for Linguistic Modeling
MPRI 2-27-1

B. Crabbé

2020-2021
Plan

1. General introduction
2. Linguistics basics
3. Machine learning for computational linguistics
Teachers: Benoît Crabbé (U Paris, UFR Linguistics, 1st half); P. de Groote (INRIA Nancy, 2nd half)

The class introduces to computational linguistics, that is computational modelling of language.

This contrasts with Natural Language Processing, that is more oriented towards engineering.

The whole class puts the focus on natural language understanding, that is methods for computing the meaning of written sentences or texts.
Tentative outline

- General introduction to linguistics and ML basics for computational linguistics
- Modelling Sequences
- Modelling syntax (representations)
- Parsing algorithms for computational linguistics
- Semantic representations
- Syntax/Semantics interface
- Montague semantics
- Discourse analysis
The fundamental hypothesis

Language has structure and symbolic (logical) structures are relevant for language modelling

- These structures can take the form of symbolic data structures such as trees, graphs... or can be expressed with logical expressions.
- This working hypothesis is common with traditional scientific hypotheses in linguistics.
- Within artificial intelligence subfields, this makes computational modelling of language substantially different from most other subfields (for instance computer vision where structure is less obvious)
Examples of hidden structures

- Natural language has an underlying syntactic structure (Chomsky, 1956):

```
S
  /\     /
NP  VP  
  |   |  /
D N V NP
 | | | |
A kid eats D N
 |  |  an apple
```

- Natural language semantics can be expressed by logical means:

\[
\exists x y \text{KID}(x) \land \text{APPLE}(y) \land \text{EAT}(x, y)
\]
Why do we care about structures?

- We want to perform **predictions** (or inferences)
- With **syntax**, this helps to define a theory of language:
  - We can use a grammar to check if a given sentence can be generated (grammaticality test).
  - We can update the grammar if needed
- With **semantics** beyond theory, some applications are typical:
  - We can test for logical entailments:
    
    \[
    \begin{align*}
    \text{A kid eats an apple} \\
    \text{All apples are rotten} \\
    \text{A kid eats a rotten apple}
    \end{align*}
    \]
  - More generally we are interested in performing textual inferences (for instance for finding paraphrases or for question answering), this is called textual entailment:
    
    - T His family has steadfastly denied the charges.
    - E The charges were denied by his family.
The structure is often considered as a first class citizen

- More surprisingly a structure can be a valuable resulting object in itself (without necessarily trying to perform inferences)
- Dependency structures are an example of such a case where the structure allows to extract easily the “who did what to whom” from a sentence:

```
  N  V  D  N
Peter eats a salad
```

This kind of representation is typically useful for information extraction
Inspiration from the semantics of formal languages

- One can think to reuse methods used in the design of programming languages.
- Example, the arithmetic expression:

\[ 3 + 7 \times 2 \]

RAW SYNTACTIC STRUCTURE

\[
\begin{array}{c}
+ \\
\downarrow \\
3 \times \\
\downarrow \\
7 \quad 2
\end{array}
\]

WITH DENOTATIONS

\[
\begin{array}{c}
+:17 \\
\downarrow \\
3:3 \times:14 \\
\downarrow \\
7:7 \quad 2:2
\end{array}
\]
If we omit the priority rules for arithmetic expressions, then the expression becomes ambiguous.

Example, the arithmetic expression:

\[ 3 + 7 \times 2 \]

becomes ambiguous:
It makes sense to draw inspirations from the study of formal languages for computational linguistics, one can design compositional methods for computing the semantics for natural language (Montague, 1974).

Example: A *kid eats a salad*

\[
\text{EAT} : \lambda P . \exists x y z. \text{EATS}(x, y) \land P(x) \land Q(y) \\
\text{KID} : \lambda x. \text{KID}(x) \\
\text{SALAD} : \lambda x. \text{SALAD}(x)
\]

\[
\begin{array}{c}
\text{KID} \quad \text{SALAD} \\
\hline
\text{RAW SYNTAX} \quad \text{DENOTATIONS} \\
\end{array}
\]

\[
\begin{array}{c}
\text{EAT} \quad \left[ \exists x y z. \text{EATS}(x, y) \land \text{KID}(x) \land \text{SALAD}(y) \right] \\
\end{array}
\]

Where the root node is a functor and the leaves its arguments. (You will see how to systematize that later and in the second part of the class)
Ambiguity

- Ambiguity is a massive issue in natural language processing and it cannot be easily fixed with priority rules, as for arithmetic expressions.
- Structural ambiguity:
  - The kid eats (a salad (with a tomato))
  - The kid eats (a salad) (with a knife)
  - (Fed) raises (interest rates)
  - (Fed raises) interest (rates)
- Ambiguity increases exponentially as sentences get longer (attachments and coordinations)
  - Mary drinks delicious (cookies and milk)  
  - Mary drinks (delicious cookies) and (milk)

Natural languages are unconstrained

One can design formal languages to avoid ambiguity as much as possible, but for natural languages one needs to take the language as it comes.
Disambiguation

- **Using world knowledge:**
  - The kid eats ( a salad (with a tomato ) )
  - The kid eats ( a salad ) ( with a knife )

- **Using discourse context:**
  - L’enfant mange (une salade (avec un avocat) )
  - L’enfant mange (une salade) (avec un avocat)
  - (Fed) raises (interest rates)
  - (Fed raises) interest (rates)

- **Using prosody (marked here with ‘,’) or regular patterns**
  - (La belle), (porte (le voile))
  - (La belle porte), (le voile)

**Knowledge of Language and Knowledge of the World**

Computational linguistics not only involves modelling language but also some knowledge of the world. (generic AI problem)
Are natural languages fundamentally flawed?

The function of ambiguity

- At first sight, this may seem odd that human languages come with a built-in design flaw, namely ambiguity.
- At second sight, instead of a bug, ambiguity can be rather seen as a feature that helps humans to communicate more efficiently (Zipf, 1949; Piantadosi et al., 2012), that is by compressing the message and by taking into account the discourse context.

  The core assumption can be summarized as:

  *The essential asymmetry is: inference is cheap, articulation expensive, and thus the design requirements are for a system that maximizes inference. Hence … linguistic coding is is to be thought of less like definitive content and more like interpretive clue (Levinson 2000).*

  In other words, writing or uttering words takes time, it is more efficient to reuse cheap short messages overloaded with potentially many senses provided that it is easy to disambiguate given the utterance context.

  *(?) actually show empirically on German, Dutch, English that short words/easy to pronounce words have more homophones and more lexical meanings than longer ones.*
Interim summary:

- Natural language cannot be syntactically as constrained as a programming language (we cannot enforce the grammar by design)
- Natural language is highly ambiguous
- Natural language is also full of irregularities, exceptions errors...
- Machine learning is used:
  - in the first place as a method to score competing textual interpretations from a large variety of features (including linguistic and world knowledge)
  - also for statistical inference, where such inferences can be compared to human behaviour for the purpose of cognitive modelling
Plan

1. General introduction
2. Linguistics basics
3. Machine learning for computational linguistics
Linguistics overview

- Linguistics is the science that studies language in all of its aspects:
  - **Sounds and phonemes**: phonetics and phonology
  - **Words**: morphology, lexicology
  - **Sentences**: syntax and semantics
  - **Discourse and dialogue**
  - **Variation**: *Typological and multilingual*, seeking descriptions and universals across languages, but also *Historical, Geographical, Social*

- With a variety of **methods**: experimental, computational and mathematical, fieldwork
Lexical properties

- As it stands, the word is a vague notion. Let’s consider:
  - **Word types**, that is the *set* of strings that are entries of the vocabulary (dictionary)
  - **Word tokens**, that is the *sequence* of words making up a corpus (a text). We can use tokens to count the number of occurrences of word types.

- Words are not just strings, they also have meaning and they can be polysemous:
  - *papier* ~ *papier* (polysemy, 1 word type ?)
  - *avocat* ~ *avocat* (homonymy, 2 word types ?)
  - *est* ~ *est* (categorical homonymy, 2 word types ?)

**Counting words from corpora is fundamentally flawed**

In practice, in computational linguistics, almost every statistical method counts badly defined strings (tokens) where the word identity remains approximative.
Other lexical properties

Compounds

Whitespace or punctuation is a poor word boundary indicator:

- Multiple words merged with evolution: *un bonhomme, deux bonshommes, trois bonshommes*
- Multiple words merged with punctuation: *aujourd’hui, quelqu’un*
- Hyphenated words: *arc-en-ciel, peut-être, abat-jour* (vs *viendrai-je*)
- Whitespace separated: *à temps, bande dessinée, eau de vie, machine à écrire, de sorte que, bien que*
- With potential lexical insertion: *avoir l’air, se rendre compte*...

Without mentioning languages where white space is not used as word boundary indicator (such as Chinese)

Counting words from corpora is fundamentally flawed

In practice, in computational linguistics, almost every statistical method counts badly defined strings (tokens) where the word identity remains approximative
Zipfian effect

Let $C(w)$ be a word type counting function in a textual corpus $D = w_1 \ldots w_n$ generated by a vocabulary $V$, then we define the ordering relation over word types:

$$w_1 \prec w_2 \iff C(w_1) > C(w_2)$$

A **ranked distribution** $R$ is a count distribution whose domain $Dom(R)$ is ordered by $\prec$.

Let $w^i$ be the rank of of $w$ in $Dom(R)$, then for a sufficiently large textual corpus, the following holds:

$$C(w^i) \approx \frac{C(w^1)}{w^i}$$

This is called the **Zipf law** of lexical frequencies.
Lexical pattern
Consequences for statistical modelling

- Words are a key building bricks in language modelling
- But it is hard to extract reliable counts from corpora, a large proportion of the vocabulary is seen only once
- This explains why extra-large corpora are used for performing lexical counts (billions and more recently several billions words)
Computational linguistics inherits from linguistics the use of discrete symbolic categories for modelling language. These categories do not appear from out of nowhere by some magical operation.

Let’s see how we can define categories such as nouns, verbs and related morphological classes.

Rather than attempting to define “conceptually” with some sort of long verbose definition those categories, we use a set of tests. A word that passes all tests related to some category belongs to that category, otherwise it does not. Example:

Mon ________ aime le cinema

we have that frère passes the test, on the other hand Pierre, il and celui-ci fail the test
Exercise

Consider the words *Pierre, frère, il, celui-ci* and the following tests:

________ aime le cinéma
Jeanne aime ________
Jeanne parle à mon ________
C'est ________ qui aime le cinéma
C'est mon ________ qui aime le cinéma
Qui aime le cinéma ? Mon ________.
Seul ________ aime le cinéma
Seul mon ________ aime le cinéma

Mon ________ aime le cinema
Jeanne parle à ________
Jeanne parle à mon ________
C'est ________, me semble-t-il, aime le cinéma
Mon, ________, me semble-t-il, aime le cinéma
Qui aime le cinéma ? ________.
_______, il aime le cinéma ?
_______ et (Jean | elle | celle-là | filles) aiment le cinéma
Aime(-t-) ________ le cinéma ?

According to the tests results, classify the words into categories and try to name those categories.

Can you abstract and generalize those tests beyond the *cinéma* ?
Tests and semantics

Given the following examples, abstract tests for distinguishing determiners and pronouns

- Chaque étudiant passera l’examen final
- Tous sont venus à la fête
- La fille se leva et et posa une question
- Jean la connait par coeur
- Certains étudiants sont déjà prêts pour l’examen
- Certains ne viendront pas à la fête
- Parmi les artistes de Paris, plusieurs sont connus dans le monde entier
- Plusieurs jeunes sont sortis dans la rue pour manifester

Given the examples, do you think that a semantic test such as a “the pronoun replaces a noun” would work straightforwardly?
Defining constituents

A constituent describes how we group words together. How do we decide which word sequences are constituents and which are not?

A constituent is a group of words that behave like a single word:
- *Le chef de la tribu* arrive ensuite
- *Géronimo* arrive ensuite

A constituent can be moved to some extent as whole:
- *Le chef de la tribu* arrive ensuite
- C’est ensuite qu’arrive *le chef de la tribu*
- *C’est ensuite que le chef arrive de la tribu* (autre sens au mieux)

Some constituent can be removed:
- Pierre arrive *le matin*
- Pierre arrive
- *Pierre arrive le*
Recursion

- Constituents can be embedded into other constituents, and can therefore be represented by phrase structure trees.

- Constituents can be seen as representation of the derivation of a context free grammar that generates the sentence.
Plan

1. General introduction
2. Linguistics basics
3. Machine learning for computational linguistics
## Traditional setup: a look at a classical data set

**Fisher's Iris**

<table>
<thead>
<tr>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
</tr>
<tr>
<td>5</td>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>6</td>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
</tr>
</tbody>
</table>

... 

#### Bullet Points:
- The $N$ lines represent the $N$ observations.
- The $K$ columns the variables.
- For the sake of providing an example, we use linear a model that predicts $y$ the Petal.Width given a vector $x$ of $k$ predictors (the Petal.Length, Sepal.Width and Sepal.Length).
Two problems of interest

- **Prediction.** Given a vector \( x \) of values for the \( k \) predictor variables predict a value for \( y \) the predicted variable. This problem assumes \( w \) to be known and simply evaluate the function for the given input. In the case of a linear model,

\[
\hat{y} = w^\top x + b
\]

- **Estimation.** Given a dataset \( D = (x_i, y_i)_{i=1}^N \) estimate the parameters \( w \) of the model.
  - Estimation amounts to seek a value for \( w \) such that the **loss function** is minimized.
  - The loss function measures the differences between model predictions and reference values on the whole dataset.
  - In other words, estimation amounts to solve the optimization problem:

\[
\hat{w} = \arg\min_{w \in \mathbb{R}^k} \sum_{i=1}^N \ell(y_i, \hat{y}_i; w)
\]
More details on the loss for linear models

- Given a dataset \( D = (x_i, y_i)_{i=1}^N \), the loss for linear models is:

\[
\hat{w} = \arg\min_{w \in \mathbb{R}^k} \sum_{i=1}^N \ell(y_i, \hat{y}_i; w) = \arg\min_{w \in \mathbb{R}^k} \sum_{i=1}^N (y_i - (w^T x_i + b))^2
\]

this is the \textit{least squares} loss function

- This problem is typically solved by gradient descent.
Consider the sentence annotated with word categories:

\[
Le/D \text{ vent/N souffle/V à/P l'/D est/N ./PT}
\]

This can also be viewed as a data set analog to the *Iris*:

<table>
<thead>
<tr>
<th>wordL</th>
<th>word</th>
<th>wordR</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>[none]</td>
<td>Le</td>
<td>vent</td>
<td>D</td>
</tr>
<tr>
<td>Le</td>
<td>vent</td>
<td>souffle</td>
<td>N</td>
</tr>
<tr>
<td>vent</td>
<td>souffle</td>
<td>à</td>
<td>V</td>
</tr>
<tr>
<td>souffle</td>
<td>à</td>
<td>l'</td>
<td>P</td>
</tr>
<tr>
<td>à</td>
<td>l'</td>
<td>est</td>
<td>D</td>
</tr>
<tr>
<td>l'</td>
<td>est</td>
<td>.</td>
<td>N</td>
</tr>
<tr>
<td>est</td>
<td>.</td>
<td>[none]</td>
<td>PT</td>
</tr>
</tbody>
</table>

With the key difference that data is now entirely categorical
Numerical coding of categorical data

One hot encoding and features

- Let $V$ be a categorical variable (a **finite** set of distinct symbolic values) then the **feature function** $\phi : V \mapsto \mathbb{R}^{|V|}$ maps each value $v \in V$ to a $|V|$ dimensional vector $\phi(v)$.

- Each such vector $\phi(v)$ is (by convention) **one-hot**: it is zero valued, except one value set to one. The coding scheme is designed to make sure that each distinct value $v \in V$ is one valued in a distinct dimension.

**Example**

Let $V = \{a, b, c, d\}$, then we may choose to design $\phi$ as:

<table>
<thead>
<tr>
<th>value</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1000</td>
</tr>
<tr>
<td>b</td>
<td>0100</td>
</tr>
<tr>
<td>c</td>
<td>0010</td>
</tr>
<tr>
<td>d</td>
<td>0001</td>
</tr>
</tbody>
</table>
The typical model for language

- Like for the earlier *Iris*, natural languages models take a vector $x$ of input symbolic variables. We note the $\Phi(x)$ the **feature vector** coding $x$ a sparse boolean vector (by concatenation of individual vector codes).
- The predicted variable is not a single scalar anymore, it is a $|Y|$–dimensional vector $y$ with each value $y \in Y$ mapped to a specific dimension.
- In a context where both inputs and outputs are categorical, the linear model becomes:

\[ y = W\Phi(x) + b \]

where $W$ and $b$ are a matrix and a vector of real parameters

**Some intuition**

Each row of the matrix $W$ codes the parameters related to the output value one-hot encoded to this dimension. Each column of the matrix codes the parameters related to the input value encoded in the corresponding dimension.
Multinomial logistic models for language

the softmax function

- The linear model for categorical variables described so far outputs a real valued vector $y$ with a real valued score corresponding to each of the values $y \in Y$.
- Most of the times, we use the softmax function to turn $y$ into a probabilistic vector $\hat{y}$. For each value of the vector:

$$\hat{y} = \text{SOFTMAX}(y) = \frac{\exp(y_i)}{\sum_{j=1}^{\|Y\|} \exp(y_j)} \quad (1 \leq i \leq |Y|)$$

- In other words, the softmax allows us to interpret the output $\hat{y}$ of the model as a conditional probability distribution $P(Y|x)$.
- Such models are called multinomial logistic models or simply logistic models when $|Y| = 2$ (in this last case, a less redundant formalisation can be given).
The loss for multinomial logistic models is the **cross entropy loss**. It compares \( \hat{y} \) and \( y \) as:

\[
\ell(y, \hat{y}; W) = -\sum_{j=1}^{|Y|} y_j \log(\hat{y}_j)
\]

Note that the loss is zero for all but one \( y_j \). In the case where \( y_j = 1 \) the loss is null when \( \hat{y}_j = 1 \) and maximal when \( \hat{y}_j \) is close to 0.

For a whole data set \( D = (x_i, y_i)_{i=1}^N \), the loss is then defined as:

\[
L(D; W) = \sum_{i=1}^{N} \ell(y_i, \hat{y}_i; W)
\]

Again parameter estimation amounts to minimize such a function and it is optimized by gradient descent.
Gradient descent in computational linguistics

Common practices (loss simplification)

- We assume a model of the form:

\[
\hat{y} = P(y|x; W, b) = \frac{\exp(w_y\Phi(x) + b)}{\sum_{y' \in Y} \exp(w_{y'}\Phi(x) + b)}
\]

where \( w_y \) is a row of \( W \)

- The loss is the cross entropy loss, that is:

\[
L(D; W) = -\sum_{i=1}^{N} \sum_{j=1}^{\mid Y \mid} y_{ij} \log(\hat{y}_{ij})
\]

\[
= -\sum_{i=1}^{N} y_{i} \log(\hat{y}_{i})
\]

\[
= -\sum_{i=1}^{N} \log(\hat{y}_{i})
\]

where \( y_{i} \) is the value at the single index \( j \) where \( y_{ij} = 1 \).
Gradient for NLL

- We expand $\hat{y}_i$ to get:

$$L(D; W) = - \sum_{i=1}^{N} \log \left( \frac{\exp(w_{y_i} \Phi(x) + b)}{\sum_{y' \in Y} \exp(w_{y'} \Phi(x) + b)} \right)$$

this is called the Negative Log Likelihood (NLL)

- The gradient is the vector of partial derivatives with respect to individual weights:

$$\frac{\partial L(D; W)}{\partial w_{op}} = - \sum_{i=1}^{N} ([y_i = y_o] - P(y_o|x_i, W, b))x_{ip}$$

where $[y_i = y_o]$ is an indicator (1 if the row of the weight $w_{op}$ is the reference row and 0 otherwise)
Gradient descent and stochastic gradient descent

- The classical gradient descent (GD) algorithm can be used to estimate the parameters:

  function $\text{GD}(\alpha, D = (x_i, y_i)_{i=1}^N)$
  
  $W \leftarrow 0$
  
  while non convergence do
  
  $W \leftarrow W - \alpha \nabla L(D, W)$
  
  end while

end function

- while Stochastic Gradient Descent (SGD) uses a single randomly chosen example at a time:

  function $\text{SGD}(\alpha, D = (x_i, y_i)_{i=1}^N)$
  
  $W \leftarrow 0$
  
  while non convergence do
  
  $i \leftarrow \text{UNIFORM}(1,N)$
  
  $W \leftarrow W - \alpha \nabla \ell(y_i, \hat{y}_i, W)$
  
  end while

end function
Word embeddings

Motivations

- With one-hot encoding, and with some metric of similarity such as euclidean distance, all vectors encoding words are equally distant.
- However some word pairs are more similar to others: for instance *monkey, gorilla, human, orangutan* are words that have more similar meanings than say *gorilla* and *rock* or say *gorilla* and *president*.
- The motivation for using encodings similar among semantically similar words also comes from data sparseness effects related to Zipf law. Obviously if we have designed some formal representation of *monkey* it is desirable to share some of the content with related *apes* given the few observations we have from corpora.
- **Word embeddings** are small dimensional vectors (compared to one-hot) for coding words that aim to get vector representations of semantically similar words close to each other given some similarity metric.
How to create word embeddings?

- Given an already existing model such as the softmax regression model, one can straightforwardly create embeddings by functional composition. That is:

$$\text{SOFTMAX}(\mathbf{W}_{out}\phi(\mathbf{x}) + \mathbf{b})$$

is reformulated as:

$$\text{SOFTMAX} \left( \mathbf{W}_{out} \begin{bmatrix} \mathbf{W}_{emb}\phi(\mathbf{x}_1); \\ \vdots \\ \mathbf{W}_{emb}\phi(\mathbf{x}_k); \end{bmatrix} + \mathbf{b} \right)$$

which is an instance of a neural network that can be trained with a cross entropy loss as before.

- $\mathbf{W}_{emb}$ has as many columns as the feature vector dimensionality and typically contains much fewer rows such that the product with the one hot vector $\mathbf{e} = \mathbf{W}_{emb}\phi(\mathbf{x}_i)$ yields a small dimensional embedding vector.
Although the previous method generally works fine, (Mikolov et al., 2013) describes two methods that explicitly aim to create a dictionary of word embeddings.

The methods are inspired by the distributional hypothesis (Firth 1957). We use the experimental example from McDonald and Ramscar (2001):

- He filled the **wampimuk**, passed it around and we all drunk some
- We found a little, hairy **wampimuk** sleeping behind the tree

This suggests that given the context we can somehow infer the semantic content of the unknown word.
Mikolov’s CBOW

- CBOW formalizes the intuition provided by the examples as follows: Let \( w_i \) be a target token and \( w_{i-1}, w_{i-2} \) its left context and \( w_{i+1}, w_{i+2} \) its right context.
- CBOW aims to predict the one hot encoding representation of \( w_i \) given its left and right contexts and by enforcing a low dimensional embedding representation \( h \):

\[
P(w_i | w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}) = \text{SOFTMAX}(W_{out}h + b)
\]

\[
h = \frac{1}{4} \sum_{w \in \{w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}\}} W_{\text{emb}\phi}(w)
\]

Once the model trained, observe that the embedding of a word \( w \) can be extracted by simply getting the result of \( W_{\text{emb}\phi}(w) \)

Alternative method

Note also the alternative method called skip-gram that "reverses" CBOW by attempting to predict the neighbouring words given \( w_i \)
Examples

- Cosine similarity
  \[ \text{SIM}(x, y) = \frac{x \cdot y}{||x|| \, ||y||} \]
  note that \( \text{SIM}(x, y) \in [-1, 1] \)

- Key: france, cosine similarity:
  ('canada', 0.66), ('germany', 0.65), ('spain', 0.64),
  ('barcelona', 0.61), ('mexico', 0.60), ('rome', 0.60)

- Key shocked, cosine similarity:
  ('horrified', 0.81), ('amazed', 0.77), ('astonished', 0.77),
  ('dismayed', 0.76), ('stunned', 0.76), ('apalled', 0.74)
Properties of word embeddings

- Generally a vocabulary is made of several hundred thousands of words, hence one-hot encoded vectors have very high dimensionality.
- A typical word embedding has 50/100/300/600 dimensions.
- Word embeddings are an instance of representation learning.
- Learning such representations requires very large amounts of texts (order of millions/billions of words).
- Word embeddings are representations built on strings. In case of polysemous words, senses are ’conflated’.
Embeddings and compositionality

- Vector based representations can be used to compute shallow semantics compositionally too.
- For instance, let the vector function:

\[ F(x, y) = W \begin{bmatrix} x; \\ y \end{bmatrix} + b \]

any function \( F : \mathbb{R}^k \times \mathbb{R}^k \mapsto \mathbb{R}^k \) is suitable for binary trees.
- By functional composition, such a function can be used to compute the vector representation for a binary constituent tree. This is also a perfectly valid structure for deep learning models where one typically adds an output at each node. Let \( h \in \mathbb{R}^k \) be the representation vector for some node, then we can output a symbol as before:

\[ \text{SOFTMAX}(W_{out} h + b_{out}) \]

the parameters \( W \) and \( b \) can be estimated with a cross entropy loss. A tree network of this form is called a recursive neural network.
The model is a recursive network that can be used to predict a sentiment ($\{+, -, 0\}$) at each node.

One can observe that the bag of words representation would probably predict a positive sentiment while the structural representation captures the scope of the negation and contributes to correctly predict the negative sentiment (Socher et al., 2013).
Vector semantics and logical semantics

- Neural networks provide an alternative to model theoretic semantics.
- The two methods capture different aspects of meaning, fundamentally
  - Model-theoretic semantics puts the language in relation with the world (the model), it is natural to express notions such as coreference
  - Vector based methods easily express similarities between words (or even constituents) but hardly make any reference to the world: Word embedding for anaphoric pronouns hardly catch anything substantial on pronouns.
Bibliography


