Word embeddings

X. Hilaire

ESIEE Paris, IT department
x.hilaire@esiee.fr

AIC-5102B Natural Language Processing

Outline

1. How to represent words ?
2. Contingency matrices. TF-IDF normalization
3. Latent Semantic Indexing (LSI)
4. Word2vec

Outline

1. How to represent words ?
2. Contingency matrices. TF-IDF normalization
3. Latent Semantic Indexing (LSI)
4. Word2vec

The issue

- A word is a sequence of characters
- But does the sequence conveys any useful information (apart from etymology)?
- Most of ML algorithms learn <u>numerical</u> data : either scalars, or vectors of finite dimensions
- First idea : map each word of the vocabulary to an an arbitrary numerical value, and learn numbers !

The issue

Queen Elizabeth Is Dead[1]

```
Queen Elizabeth II died "peacefully" at her home in Balmoral,
Scotland, on September 8, 2022, Buckingham Palace announced in
a statement. She was 96. "The Queen died peacefully at Balmora
this afternoon," the official statement read. The statement
continued, "The King and The Queen Consort will remain at
Balmoral this evening and will return to London tomorrow."
```

☞Hence: a $\leftrightarrow$ 1, afternoon $\leftrightarrow$ 2, and $\leftrightarrow$ 3, announced $\leftrightarrow$ 4, at $\leftrightarrow$ 5, Balmoral a $\leftrightarrow$ 6,...

Problems:

- 2022 $\leftrightarrow$???, 8 $\leftrightarrow$???, 96 $\leftrightarrow$???
- II $\leftrightarrow$???
- Should II be regarded as a new word or number ?

[1] source:https://stylecaster.com/queen-elizabeth-dead/

The issue

- ☞ One solution: map numbers to < 0 integers, real words to > 0 integers, and extend each side as new items appear
- ☞ So 8 $\leftrightarrow$ -1, 2022 $\leftrightarrow$ -2, 96 $\leftrightarrow$ -3

Not satisfactory:

- $8 - 2022 = -2014 \leftrightarrow -1 + 2 = 1$: norms are not preserved
- $8 < 2022 \leftrightarrow -1 > -2$: as well as order

More dissatisfaction:

- $|queen - king| = 23 - 17 = 5$, but $|london - balmoral| = 18 - 6 = 12$, $|tomorrow - afternoon| = 34 - 2 = 30$, $|the - a| = 29 - 1 = 28$
- $queen.king = 23 \times 17 = 391$, but $london.balmoral = 18 \times 6 = 108$, $tomorrow.afternoon = 34 \times 2 = 68$, $the.a = 29$

The issue

- Order matters !
- But a single dimension may not be sufficient to represent all semantical constraints
- Other extreme : if V is the vocabulary and $d = |V|$, then work in $\mathbb{R}^d$
- A word is then mapped to some $w \in \mathbb{R}^d$, such that
  - $\forall i \in \{1, ..., d\}, \quad w_i \in \{0, 1\}$
  - $\sum_{i=1}^{d} w_i = 1$
- Such vectors are called **one-hot** vectors : all their components are zero except one

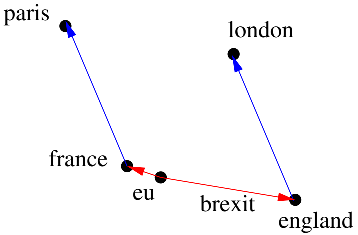
$$king = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ ... \\ 0 \end{pmatrix} \quad queen = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ ... \\ 0 \end{pmatrix}$$

The issue

Problems:

- All words are orthogonal: $w_i \cdot w_j = 0, \quad \forall i \neq j$
  - synonyms don't have a > 0 dot product
  - antonyms don't have a < 0 dot product
- All words are equidistant: $||w_i - w_j|| = 1$
- The resulting vector space is **sparse**
- Semantic relations like $king - male = queen - female$ or $paris - france = london - england$ are not encoded

However, we can **reduce** d to build a **dense** vector space, and **choose** w**'s coordinates** so that embedded words encode something – or at least, attempt to



Distributional hypothesis

- The **distributional hypothesis** of words assumes that words which are semantically bound tend to appear closely in text, i.e., in the same **context**
- Popularized by Firth [2] and Harris [3] in 1950's
- Discussed by Sahlgren [7] in 2008
- Does not distinguish whether words have identical or opposite meanings when they appear.
- Has served as a basis for the methods presented hereafter:
    - LSA
    - Word2vec
    - Glove

Outline

1. How to represent words ?
2. Contingency matrices. TF-IDF normalization
3. Latent Semantic Indexing (LSI)
4. Word2vec

Co-occurrence matrices

- Given an input text defined over a vocabulary V, and an integer $s \geq 1$, a **co-occurrence matrix** C is a $|V| \times |V|$ integer matrix such that C_{ij} counts the number of times words w_i and w_j were both seen within a sliding window with size s
- Example ($s = 2$):

  $\boxed{\text{I like}}$ swimming
  I like potatoes
  I do not like butter
  I like swimming

$$C = \left(\begin{array}{l|ccccccc} & \textit{butter} & \textit{do} & \textit{i} & \textit{like} & \textit{not} & \textit{potatoes} & \textit{swimming} \\ \hline \textit{butter} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \textit{do} & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \textit{i} & 0 & 1 & 0 & 3 & 0 & 0 & 0 \\ \textit{like} & 1 & 0 & 3 & 0 & 1 & 1 & 2 \\ \textit{not} & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ \textit{potatoes} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \textit{swimming} & 0 & 0 & 0 & 2 & 0 & 0 & 0 \end{array}\right)$$

Term-document matrices

- A natural extension to co-occurrence matrices is to allow the window to have variable length
- More precisely : length = 1 document (or even length = 1 book)
- We then count how many times a given term appears in documents
- This gives raise to **term-document matrices**
- Example (Reuter's corpus, cropped for pedagogical purposes):

$C =$

doc#	144	236	237	242	246	248	273	489	502	704
crude	0	1	0	0	0	0	5	0	0	0
dollars	0	2	1	0	0	3	2	1	1	0
last	1	4	3	0	2	1	7	0	0	0
million	4	4	1	0	0	3	9	2	2	0
oil	11	7	3	3	4	9	5	4	4	3
opec	10	6	1	2	1	6	5	0	0	0
prices	3	2	0	1	0	7	4	2	2	2
reuter	1	1	1	1	1	1	1	1	1	1
said	9	6	0	3	4	5	5	2	2	3
saudi	0	0	0	1	0	5	7	0	0	0

TF-IDF normalization

- Direct comparison of vectors (row or column) extracted from co-occurrence or term-document matrices is often not relevant
- Words like “the”, “of”, “some”, etc. are ubiquitous; their high frequencies dominate in norms or dot products, but they have limited or no discrimination power
- Idea #1 (TF = term frequency) : the relevance of a term should be proportional to the log of its frequency rather than its frequency itself

$$tf(t, d) = \log(1 + f(t, d))$$

  where $f(t, d)$ = number of times term t appears in document d
- Suggested by Luhn [4] in 1957, with no real justification
- *A posteriori* explanation provided by Zipf's law

TF-IDF normalization

- Idea #2 (IDF = inverse document frequency) : the relevance of a term should be inversely proportional to the document frequency, i.e, the proportion of documents in which the term appears
- Also normalized on a log scale to comply with *tf*

$$idf(t) = \log \frac{N}{n_t}$$

where N = total number of documents, and n_t = number of documents in which t appears

- To avoid division by 0 while keeping ≥ 0, one usually prefers

$$idf(t) = \log \frac{N+1}{1+n_t}$$

- Terms which appear in very few documents lead to an $idf \approx \log N$ and are highly discriminative.

TF-IDF normalization

- Terms which appear everywhere lead to an *idf* ≈ 0, and are not discriminative.
- Justified by Karen Jones in 1972 [8], drawing again on Zipf's law.
- Idea #3 Combine *tf* and *idf* in a single score to accounting for both

$$
\begin{aligned}
tfidf &= tf(t,d)\,idf(t) \\
&= \log\left(1 + f(t,d)\right)\log\frac{1+N}{1+n_t}
\end{aligned}
$$

- Used in about 80% of information retrieval systems [1]
- Remains empirical, though.

TF-IDF normalization

Example : non normalized version of the C matrix on truncated Reuter's corpus (10 documents only)

$$C = \left(\begin{array}{lcccccccccc}
\textit{doc\#} & 144 & 236 & 237 & 242 & 246 & 248 & 273 & 489 & 502 & 704 \\
\hline
\textit{crude} & 0 & 1 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \\
\textit{dollars} & 0 & 2 & 1 & 0 & 0 & 3 & 2 & 1 & 1 & 0 \\
\textit{last} & 1 & 4 & 3 & 0 & 2 & 1 & 7 & 0 & 0 & 0 \\
\textit{million} & 4 & 4 & 1 & 0 & 0 & 3 & 9 & 2 & 2 & 0 \\
\textit{oil} & 11 & 7 & 3 & 3 & 4 & 9 & 5 & 4 & 4 & 3 \\
\textit{opec} & 10 & 6 & 1 & 2 & 1 & 6 & 5 & 0 & 0 & 0 \\
\textit{prices} & 3 & 2 & 0 & 1 & 0 & 7 & 4 & 2 & 2 & 2 \\
\textit{reuter} & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\textit{said} & 9 & 6 & 0 & 3 & 4 & 5 & 5 & 2 & 2 & 3 \\
\textit{saudi} & 0 & 0 & 0 & 1 & 0 & 5 & 7 & 0 & 0 & 0 \\
\hline
\end{array}\right)$$

TF-IDF normalization

Example : TF-IDF normalized version of the C matrix on truncated Reuter's corpus (10 documents only)

$C =$

doc#	144	236	237	242	246	248	273	489	502	704
crude	0.00	0.90	0.00	0.00	0.00	0.00	2.33	0.00	0.00	0.00
dollars	0.00	0.50	0.31	0.00	0.00	0.63	0.50	0.31	0.31	0.00
last	0.31	0.73	0.63	0.00	0.50	0.31	0.94	0.00	0.00	0.00
million	0.51	0.51	0.22	0.00	0.00	0.44	0.73	0.35	0.35	0.00
oil	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
opec	0.76	0.62	0.22	0.35	0.22	0.62	0.57	0.00	0.00	0.00
prices	0.28	0.22	0.00	0.14	0.00	0.42	0.32	0.22	0.22	0.22
reuter	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
said	0.22	0.19	0.00	0.13	0.15	0.17	0.17	0.10	0.10	0.13
saudi	0.00	0.00	0.00	0.70	0.00	1.81	2.10	0.00	0.00	0.00

Outline

1. How to represent words ?
2. Contingency matrices. TF-IDF normalization
3. Latent Semantic Indexing (LSI)
4. Word2vec

Latent semantic indexing

- Recall that our aim is to map words into “dense” vectors of $\mathbb{R}^d$
- One way to do this is to perform a singular value decomposition (SVD) of the term-document matrices C
- Quick reminder about SVD :

$$\boldsymbol{C} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

  where
  - $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal : $\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$, $\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$
  - $\boldsymbol{C}$ and $\boldsymbol{\Sigma}$ have shape $m \times n$,
  - while $\boldsymbol{U}$ has shape $m \times m$, and $\boldsymbol{V}$ has shape $n \times n$.
- This is equivalent to writing

$$\boldsymbol{C} = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

  where r is the rank of $\boldsymbol{C}$,

Latent semantic indexing

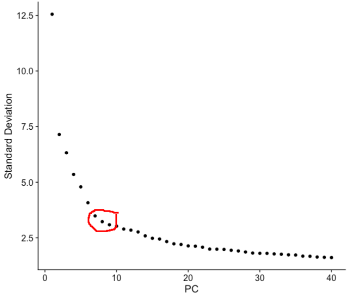
- Assuming $n > m$, the $\mathbf{\Sigma}$ matrix can be written as

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & & \vdots & 0 & \dots & 0 \\ \vdots & & \ddots & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & \sigma_r & 0 & \dots & 0 \end{pmatrix} = (\mathrm{diag}(\sigma_1, ..., \sigma_r); \mathbf{0})$$

- The σ_i's are the **singular values** of $\boldsymbol{C}$, ranged in decreasing order
- They are the squares of the r first eigenvalues of $\boldsymbol{CC}^T$

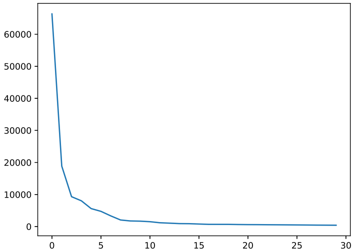
$$\begin{aligned} \boldsymbol{CC}^T &= \boldsymbol{U\Sigma V}^T(\boldsymbol{U\Sigma V}^T)^T \\ &= \boldsymbol{U\Sigma V}^T(\boldsymbol{\Sigma V}^T)^T\boldsymbol{U}^T \\ &= \boldsymbol{U\Sigma V}^T\boldsymbol{V\Sigma}^T\boldsymbol{U}^T \\ &= \boldsymbol{U\Sigma\Sigma}^T\boldsymbol{U}^T \end{aligned} \tag{1}$$

- By vanishing σ_{d+1} to σ_r, we get a $\tilde{\boldsymbol{\Sigma}}$ matrix, and by recomposing
$$\tilde{\boldsymbol{C}} = \boldsymbol{U}\tilde{\boldsymbol{\Sigma}}\boldsymbol{V}^T \tag{2}$$
whose rank is d
- This is the best d-rank approximation of $\boldsymbol{C}$ in the sense of the Fröbenius norm.. and precisely what we are looking for
- How large should be d ? ☞Like for PCA, answer typically given by the elbow method, since the remaining non null $\boldsymbol{\Sigma}$ explain the variance



Latent semantic indexing : toy example

- Original text : "Gulliver's travel", by Jonathan Swift
- 9056 different words found
- 2544 sentences
- Simple case study : 1 document = 1 sentence
- First 30 singular values are enough to explain more than 95% of variance ☞dimensionality reduction = 1-30/9056 = 99.6%



Latent semantic indexing

If $\boldsymbol{C}$ is a term-document matrix, then

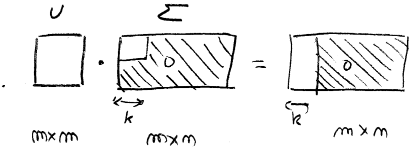
- The dot product $< t_i, t_j >$, where t_i and t_j are two rows of $\boldsymbol{C}$, expresses the correlation between terms t_i and t_j across documents.
- Matrix $\boldsymbol{C}\boldsymbol{C}^T$ contains all these dot products.
- The dot product $< d_i, d_j >$, where d_i and d_j are two columns of $\boldsymbol{C}$, expresses the correlation between documents d_i and d_j across terms.
- Matrix $\boldsymbol{C}^T\boldsymbol{C}$ contains all these dot products.

SVD does not really affect this

- by 1 and 2, we get
$$\tilde{\boldsymbol{C}}\tilde{\boldsymbol{C}}^T = \boldsymbol{U}\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{\Sigma}}^T\boldsymbol{U}^T$$
instead of
$$\boldsymbol{C}\boldsymbol{C}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T\boldsymbol{U}^T$$
- There are only slight errors on dot products, due to the approximation $\tilde{\boldsymbol{\Sigma}} \approx \boldsymbol{\Sigma}$

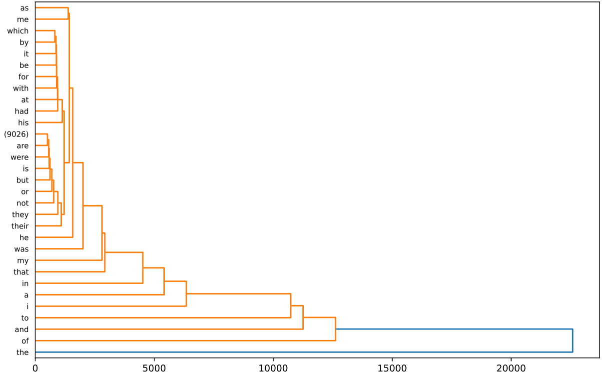
Latent semantic indexing

- Equality
$$\tilde{\boldsymbol{C}}\tilde{\boldsymbol{C}}^T = (\boldsymbol{U}\tilde{\boldsymbol{\Sigma}})(\tilde{\boldsymbol{\Sigma}}^T\boldsymbol{U}^T)$$
can be interpreted in terms of dot products of "pseudo-documents" whose DT matrix is $\boldsymbol{U}\tilde{\boldsymbol{\Sigma}}$.
- Thinking of rows d_i of $\boldsymbol{U}\tilde{\boldsymbol{\Sigma}}$ as "new" documents implies that "native" documents $\hat{d}_i$ satisfy $d_i = \boldsymbol{U}\tilde{\boldsymbol{\Sigma}}\hat{d}_i$, or $\hat{d}_i = (\boldsymbol{U}\tilde{\boldsymbol{\Sigma}})^{-1}d_i$ with slight abuse of notation (-1 = pseudo-inverse)
- Only the k first components of $\boldsymbol{U}\tilde{\boldsymbol{\Sigma}}$ are useful : others are 0 and can be ignored



Visualizing word similarities

- Word similarities can be visualized using hierarchical clustering and dendrograms.
- Example on "Gulliver's travel" data (thresholded, raw DT matrix)



Latent semantic indexing

Conclusion on DT matrices, co-occurrence matrices, TF-IDF, and LSI :

- Simple techniques to estimate correlations between words or terms
- TF-IDF is a must in case DT, or co-occurrences matrices are exploited directly
- LSI is very good at dimensionality reduction, and produce dense vectors
- DT and co-occurrence matrices are highly dependent on the size of the sliding window
- Word semantic is only weakly captured: distributional hypothesis of words means more than simple word proximity in sentences
- Techniques presented hereafter attempt to better capture local nature of words

Outline

1 How to represent words ?

2 Contingency matrices. TF-IDF normalization

3 Latent Semantic Indexing (LSI)

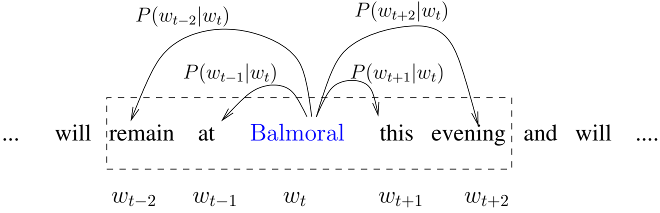
4 Word2vec

Word2vec

- Proposed by Mikolov et al. [5] in 2013.
- Comes in two flavors, depending on the language assumption :
  - skipgram assumption: any piece of text is explained by the conditional probabilities of observing context words given a center word
  - CBOW (continuous bag of words) assumption: any center word of a text is explained by the conditional probabilities of observing this word given its context words
- Both skipgram and CBOW use a window of size $2s + 1$
- The **center word** is the word at the center of the window
- Its **context words** are other words of the window
- CBOW is a vice-versa of skipgram

Word2vec, skipgram flavor

- Intuition of skipgram:



- The blue word at the center of the window is the center word.
- Other words within the window are context words.
- Assumption of skipgram:
  - The conditional probabilities of context words given center words are enough to explain the whole window
  - All conditional probabilities are independent (very naive assumption)
  - Hence, window's contents are also independent

Word2vec, skipgram flavor

- We work within a sliding window of fixed size $2s+1$ ($s=2$ in the example)[2]
- If all probabilities are independent (skipgram assumption), then the average log-likelihood over a text of size T is

$$
\begin{aligned}
L(\theta) &= \frac{1}{T}\log\prod_{t=1}^{T}\prod_{\substack{-s\leq j\leq s\\ j\neq 0}} P(w_{t+j}|w_t;\theta)\\
&= \frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-s\leq j\leq s\\ j\neq 0}} \log P(w_{t+j}|w_t;\theta) \qquad (3)
\end{aligned}
$$

where θ is the representation of all words we are precisely looking for

[2] It is often referred to as a *static embedding* method for this reason

Word2vec, skipgram flavor

- What should we use for P ? What is θ ?
- Any word w has indeed two representations :
  - u_w if it is a context word
  - v_w if it is a center word
- If all vectors live in $\mathbb{R}^d$, then θ can be thought of as a $2|V| \times d$ real matrix
- Given a context word o and its related center word c, Mikolov et al. use the softmax function to define the conditional P :

$$P(o|c) = \frac{\exp(u_o \cdot v_c)}{\sum_{x \in V} \exp(u_x \cdot v_c)} \tag{4}$$

- Why ?
  - Dot product is high for close vectors
  - Exponentiating makes everything positive
  - Nice mathematical properties (probability, smooth everywhere)

Word2vec, skipgram flavor

- Plugging (4) into (3) yields

$$
\begin{aligned}
L(\theta) = \frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-s \leq j \leq s \\ j \neq 0}} \left(u_{t+j} \cdot v_t - \log \sum_{x \in V} exp(u_x \cdot v_t) \right) \\
= \frac{1}{T}\sum_{t=1}^{T} v_t \sum_{\substack{-s \leq j \leq s \\ j \neq 0}} u_{t+j} \\
- \frac{2s}{T}\sum_{t=1}^{T} \log \sum_{x \in V} \exp(u_x \cdot v_t)
\end{aligned} \tag{5}
$$

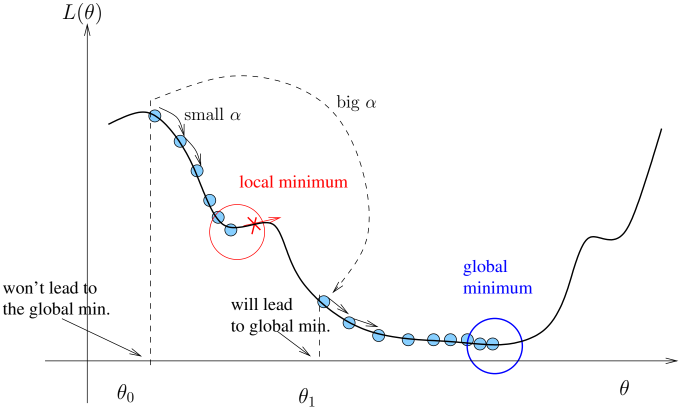
- How to optimize (5) ? ☞Use gradient descent

Word2vec, skipgram flavor

Gradient descent equation:

$$\boxed{\theta^{(i+1)} = \theta^{(i)} - \alpha \nabla_\theta L(\theta^{(i)})} \tag{6}$$

where $\alpha > 0$ is the **learning rate**



Word2vec, skipgram flavor

- Problem : equation 5 involves
  - a summation over the whole text (T words $= O(T)$)
  - a nested summation over the whole text and vocabulary (V words, $O(TV)$)
- Evaluating $L(\theta)$ even once would take a **huge** time !!
- We can't use conventional gradient descent for that reason, but stochastic gradient descent (SGD)
- SGD in a nutshell:
  1. Choose initial values for all u's and v's
  2. Randomly choose t : this will induce a window
  3. Compute partial derivatives in (5) according to the chosen t; discard all other terms
  4. Apply (6) to update the gradient
  5. Repeat to step 2 until the stop criterion is met

Word2vec, skipgram flavor

Exercise: derive the partial derivatives of (5) involved in SGD. Give the complete, resulting algorithm.

Implementation of skipgram

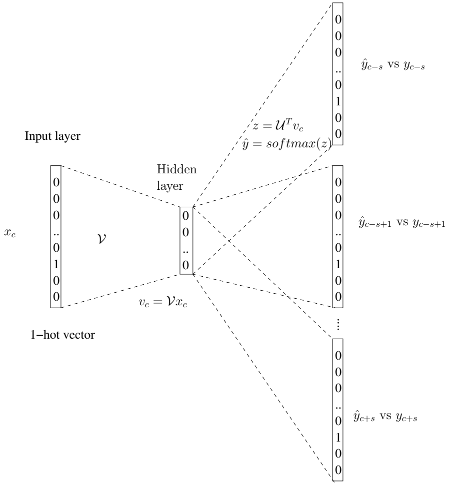
The implementation of skipgram using a 2-layer, forward only neural network is as follows:

$\mathcal{U} = d \times |V|$ matrix, whose i^{th} column is u_i

$\mathcal{V} = d \times |V|$ matrix, whose i^{th} column is v_i

$d =$ dimension of the embedding space

$x_c =$ 1-hot vector for word w_c



Implementation of skipgram

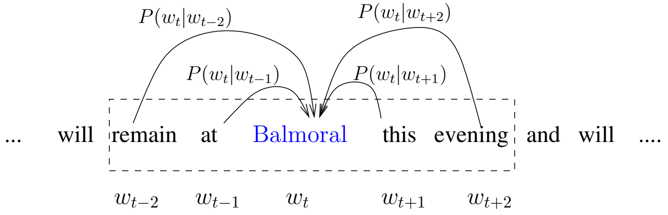
Explanations:

1. For a given word w_c, we generate its 1-hot vector x_c
2. We give this vector to the hidden layer, which computes $v_c = \mathcal{V} x_c$. This is the embedded representation of w_c. It is an $\mathbb{R}^d$ vector.
3. The output layer takes v_c as input, and computes $z = \mathcal{U}^T v_c$, then takes the softmax of each component of the resulting vector. These are the predicted probabilities of all possible words.

Short exercise: prove the result of step 3 above.

Word2vec, CBOW flavor

- For the CBOW flavor of word2vec, the setup is the following:



- As opposed to skipgram, given the context words, we now try to predict the center word.
- Analytical expressions of conditional probabilities (4) remain unchanged, c and o are just swapped:
- What does change, however, is that we are now trying to predict <u>one</u> center word given <u>several</u> context words
- This affects the summation in the likelihood expression of (3)

Word2vec, CBOW flavor

- Log-likelihood of eq. (3) remains identical, except that the $P(w_{t+j}|w_t;\theta)$ now becomes $P(w_t|w_{t+j};\theta)$

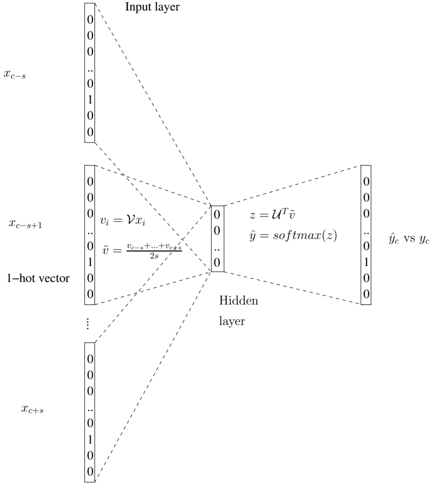
$$
\begin{aligned}
L(\theta) &= \frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-s \leq j \leq s \\ j \neq 0}} \log P(w_t|w_{t+j};\theta) \\
&= \frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-s \leq j \leq s \\ j \neq 0}} u_t \cdot v_{t+j} \\
&- \frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-s \leq j \leq s \\ j \neq 0}} \log \sum_{x \in V} \exp(u_x \cdot v_{t+j})
\end{aligned} \tag{7}
$$

Word2vec, CBOW flavor

- Partial derivatives are harder to compute, as a triple summation as emerged in (7)
- Complete derivation of partial derivatives left as an exercise
- CBOW is a bit more critical to implement due to this
- A simplification in neural implementation (see next slides) is that w_t is conditional not to every w_{t+j} for $j = -s, ..., s$, but to the average vector of these w_{t+j}.

Implementation of CBOW

The implementation of CBOW using a 2-layer, forward only neural network is as follows:



$\mathcal{U} = d \times |V|$ matrix, whose i^{th} column is u_i

$\mathcal{V} = d \times |V|$ matrix, whose i^{th} column is v_i

$d =$ dimension of the embedding space

$x_c =$ 1-hot vector for word w_c

Implementation of CBOW

Explanations:

1. For some (unknown) center word w_c, we have context words $w_{c-s}, \ldots, c_{c-1}, w_{c+1}, \ldots, w_{c+s}$, whose corresponding 1-hot vectors are $x_{c-s}, \ldots, x_{c-1}, x_{c+1}, \ldots, x_{c+s}$
2. For every $i = c-s, \ldots, c-1, c+1, \ldots, c+s$, we compute $v_i = \mathcal{V} x_i$, the corresponding embedded version of all context words
3. We average the v_i's in a single $\tilde{v}$ vector: $\tilde{v} = \frac{v_{c-s} + \ldots + v_{c-1} + v_{c+1} + \ldots + v_{c+s}}{2s}$
4. We generate $z = \mathcal{U}^T \tilde{v}$
5. We compute $\hat{y} = \text{softmax}(z)$, which is the predicted 1-hot vector for center word c

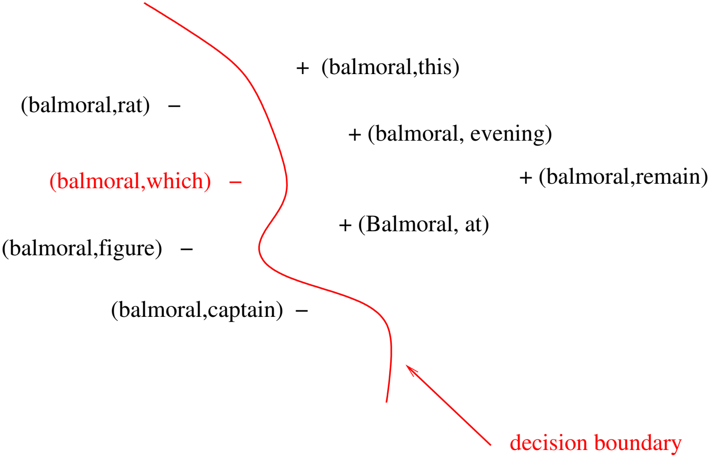
<u>Short exercise</u>: justify steps 3 and 5 above.

Word2vec, SGNS flavor

- To alleviate the huge number of terms involved in the skipgram likelihood (5), Mikolov et al. propose to negatively sample the corpus in [6]
- The resulting method is called **skipgram with negative sampling** (SGNS)
- Instead of directly optimizing (5), we will now use a classifier, that will separate between :
  - positive samples, which consists of pairs (w_c, w_o) = center word + context word observed in the corpus;
  - and negative samples, which are non existing, generated samples (w_c, c_o) from the vocabulary, all assumed non observable in the text
- This can be subject to criticism (see next slide), but still relieves the burden of (5)

Word2vec, SGNS flavor

Danger = some negative samples could perfectly have existed in real life, but unobserved in documents.



Word2vec, SGNS flavor

- Working under the skipgram assumption, let (w_c, w_o) be a pair formed of center word w_c and context word w_o.
- Their respective dense representations are v_c and u_o, and $\theta = (v_c, u_o)$, as before.
- Did this pair came from an existing document ? We denote by $P(D = 1|c, o)$ the probability it did, and by $P(D = 0|c, o)$ it did not.
- $P(D = 1|c, o)$ is modeled using the sigmoid function:

$$\begin{aligned} P(D = 1|c, o) &= P(D = 1|v_c, u_o) \\ &= \frac{1}{1 + \exp(-v_c \cdot u_o)} \end{aligned} \tag{8}$$

- $P(D = 0|c, o)$ is just 1 minus this probability

Word2vec, SGNS flavor

- Let $D^+ +$ be the set of all positive samples, and D^- that of the negative samples
- Then the log-likelihood of jointly observing the positive samples while not observing the negative ones is

$$L(\theta) = \log \left(\prod_{(c,o) \in D^+} P(D = 1 | c, o) \prod_{(c,o) \in D^-} P(D = 0 | c, o) \right)$$

- Using (8), $P(D = 0|c, o) = 1 - P(D = 1|c, o)$, and expending everything, we arrive at

$$L(\theta) = - \sum_{(c,o) \in D^+} \log(1 + \exp(-v_c \cdot u_o)) - \sum_{(c,o) \in D^-} \log(1 + \exp(v_c \cdot u_o)) \tag{9}$$

- How should D^- be built ?
- Mikolov assumes that to every observed, center word w_c should be associated K negative samples
- Under this assumption, D^- can be derived from D^+ as follows: the counterpart of any term (w_c, w_o) involved in D^+ in (9) is

$$\sum_{k=1}^{K} \exp(-v_c \cdot \tilde{u}_k)$$

in D^-, where $\tilde{u}_k$ is a vector negatively sampled from V.
- Hence, (9) can be rewritten as

$$L(\theta) = - \sum_{(c,o) \in D^+} \left[\log(1 + \exp(-v_c \cdot u_o)) + \sum_{k=1}^{K} \log(1 + \exp(-v_c \cdot \tilde{u}_k)) \right] \tag{10}$$

Some results

- Papers from Mikolov et al.
  - https://arxiv.org/abs/1301.3781
  - http://arxiv.org/abs/1310.4546
- "Word2Vec: A Comparison Between CBOW, SkipGram & SkipGramSI"
  https://kavita-ganesan.com/comparison-between-cbow-skipgram-subword/
- "Intrinsic and extrinsic evaluations of word embeddings"
  https://ojs.aaai.org/index.php/AAAI/article/download/9959/9818

Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger.
Research-paper recommender systems : a literature survey.
International Journal on Digital Libraries, 17(4):305–338, 2016.

J. Firth.
A synopsis of linguistic theory 1930-1955.
In Studies in Linguistic Analysis. Philological Society, Oxford, 1957. reprinted in Palmer, F. (ed. 1968) Selected Papers of J. R. Firth, Longman, Harlow.

Zellig S. Harris.
Distributional structure.
WORD, 10(2-3):146–162, 1954.

H. P. Luhn.
A statistical approach to mechanized encoding and searching of literary information.
IBM Journal of Research and Development, 1:309–317, 1957.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean.

Efficient estimation of word representations in vector space, 2013.

Tomás Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean.
Distributed representations of words and phrases and their compositionality.
CoRR, abs/1310.4546, 2013.

Magnus Sahlgren.
The distributional hypothesis.
Italian Journal of Linguistics, 20, 01 2008.

Karen Sparck Jones.
A statistical interpretation of term specificity and its application in retrieval.
Journal of documentation, 28(1):11–21, 1972.