Language Technology

Chapter 11: Dense Vector Representations

Pierre Nugues

Pierre.Nugues@cs.lth.se

September 26, 2024

Dimension Reduction

- One-hot encoding with TFIDF can produce very long vectors: Imagine a vocabulary one million words per language with 100 languages.
- A solution is to produce dense vectors using a dimension reduction.
- Such vectors are also called **word embeddings**
- The reduction is similar to a principal component analysis (PCA) or a singular value decomposition (SVD)
- The embedding of a word can be constant (static) or depend on the context (contextual)

Reminder: Categorical Values

Linear classifiers only understand numbers

A collection of two documents D1 and D2:

> D1: Chrysler plans new investments in Latin America.
>
> D2: Chrysler plans major investments in Mexico.

How to represent these words or these documents?

Reminder: Categorical Values: One-hot encoding

One-hot encoding:

1. Assigns a unique index to each symbol. The number of indices corresponds to the number of symbols:

   ```
   {'Chrysler': 1, 'in': 2, 'investments': 3,
   'major': 4, 'Mexico': 5, 'plans': 6}
   ```

2. Represent a symbol of index i by a unit vector $\mathbf{x} = (x_1, x_2, ..., x_n)$, where n is the largest index and all the coordinates are 0, except $x_i = 1$

   ```
   'Chrysler':    (1, 0, 0, 0, 0, 0)
   'in':          (0, 1, 0, 0, 0, 0)
   'investments': (0, 0, 1, 0, 0, 0)
   ...
   ```

Reminder: Categorical Values: Multi-hot encoding

A collection of two documents D1 and D2:

D1: Chrysler plans new investments in Latin America.

D2: Chrysler plans major investments in Mexico.

Multi-hot encoding (also called a bag-of-words representation):

1. Creates a index of all the symbols (words) in all the documents
2. For each document, creates a set of its symbols (word)
3. Represents a document by a vector of 0s and 1s. $x_i = 1$ if the word of index i is in the document, or 0 otherwise.

D.	america	chrysler	in	investments	latin	major	mexico	new	plans
1	1	1	1	1	1	0	0	1	1
2	0	1	1	1	0	1	1	0	1

Reminder: Hashing

One-hot encoding leads to very large dimensions as there are billions of different words.

In the Tatoeba corpus, the number of unigrams, bigrams, and trigrams is (10854, 361766, 1536870). Note that these values change all the time

A solution is to hash the symbols and use the remainder of a division (modulo) as index

```
>>> hash('abc')
-6712881850779232724
>>> hash('abc') % 100
76   # Always less than 100
```

Hashing:

1. Reduces the vectors size and makes it manageable
2. Creates conflicts: two symbols can have the same hash numbers
3. Is usable in classification

Code Example

Experiment: hashing CLD3 n-grams Jupyter Notebook:
https://github.com/pnugues/edan20/tree/master/labs_2024

Reminder: Reproducible Hash Codes

```
def reproducible_hash(string):
    """
    reproducible hash on any string

    Arguments:
       string: python string object

    Returns:
       signed int64
    """

    # We are using MD5 for speed not security.
    h = hashlib.md5(string.encode("utf-8"),
                    usedforsecurity=False)
    return int.from_bytes(h.digest()[0:8], 'big', signed=True)
```

Dense Vectors

We can replace one-hot vectors by dense ones using embeddings

A dense representation is a trainable vector of 10 to 300 dimensions.

The vector parameters are learned in the fitting procedure.

Dimensionality reduction inside a neural network or another procedure.

Example: GloVe file 100d.

Many techniques, often based on language modeling, here CBOW

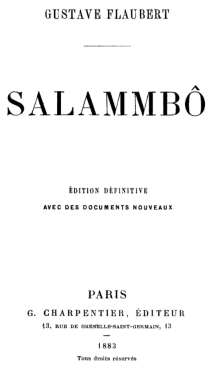
Creating Word Embeddings

1. We can derive word embeddings from corpora. Their construction is then similar to that of language models;
2. We can also introduce an embedding layer as input to a neural network. The embedding parameters are then trainable.
3. We can finally pretrain embeddings with a corpus and fine-tune them on an application.

A Dataset: *Salammbô*

A corpus is a collection – a body – of texts.

French original	English translation

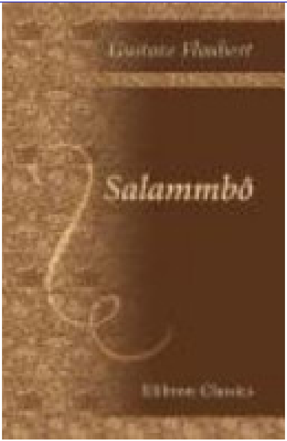


GUSTAVE FLAUBERT

SALAMMBÔ

ÉDITION DÉFINITIVE
AVEC DES DOCUMENTS NOUVEAUX

PARIS
G. CHARPENTIER, ÉDITEUR
13, RUE DE GRENELLE-SAINT-GERMAIN, 13
—
1883
Tous droits réservés



Letter Counts

Characters in *Salammbô*: A small dataset to explain PCA

Chapter	French		English	
	# characters	# *A*	# characters	# *A*
Chapter 1	36,961	2,503	35,680	2,217
Chapter 2	43,621	2,992	42,514	2,761
Chapter 3	15,694	1,042	15,162	990
Chapter 4	36,231	2,487	35,298	2,274
Chapter 5	29,945	2,014	29,800	1,865
Chapter 6	40,588	2,805	40,255	2,606
Chapter 7	75,255	5,062	74,532	4,805
Chapter 8	37,709	2,643	37,464	2,396
Chapter 9	30,899	2,126	31,030	1,993
Chapter 10	25,486	1,784	24,843	1,627
Chapter 11	37,497	2,641	36,172	2,375
Chapter 12	40,398	2,766	39,552	2,560
Chapter 13	74,105	5,047	72,545	4,597
Chapter 14	76,725	5,312	75,352	4,871
Chapter 15	18,317	1,215	18,031	1,119

Data set: Jupyter Notebook:
`https://github.com/pnugues/pnlp/tree/main/notebooks`

Representing Documents with Bags of Characters

Character counts per chapter, where the fr and en suffixes designate the language, either French or English

Ch.	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'i'	'j'	'k'	'l'	'm'	'n'	'o'	'p'	'q'	'r'	's'	't'	'u'	'v'	'w'	'x'	'y'	'z'	'à'	'â'
01_fr	2503	365	857	1151	4312	264	349	295	1945	65	4	1946	726	1896	1372	789	248	1948	2996	1938	1792	414	0	129	94	20	128	36
02_fr	2992	391	1006	1388	4993	319	360	350	2345	81	6	2128	823	2308	1560	977	281	2376	3454	2411	2069	499	0	175	89	23	136	50
03_fr	1042	152	326	489	1785	136	122	126	784	41	7	816	397	778	612	315	102	792	1174	856	707	147	0	42	31	7	39	9
04_fr	2487	303	864	1137	4158	314	331	287	2028	57	3	1796	722	1958	1318	773	274	2000	2792	2031	1734	422	0	138	81	27	110	43
05_fr	2014	268	645	949	3394	223	215	242	1617	67	3	1513	651	1547	1053	672	166	1601	2192	1736	1396	315	1	83	67	18	90	67
06_fr	2805	368	910	1266	4535	332	384	378	2219	97	3	1900	841	2179	1569	868	285	2205	3065	2293	1895	453	0	151	80	39	131	42
07_fr	5062	706	1770	2398	8512	623	622	620	4018	126	19	3726	1596	3851	2823	1532	468	4015	5634	4116	3518	844	0	272	148	71	246	50
08_fr	2643	325	869	1085	4229	307	317	359	2102	85	4	1857	811	2041	1367	833	239	2132	2814	2134	1788	437	0	135	64	30	130	43
09_fr	2126	289	771	920	3599	278	289	279	1805	52	6	1499	619	1711	1130	651	187	1719	2404	1763	1448	348	0	119	58	20	90	24
10_fr	1784	249	546	805	3002	179	202	215	1319	60	5	1462	598	1246	922	557	172	1242	1769	1423	1191	270	0	65	61	11	73	18
11_fr	2641	381	817	1078	4306	263	277	330	1985	114	0	1886	900	1966	1356	763	230	1912	2564	2218	1737	425	0	114	61	25	101	40
12_fr	2766	373	935	1237	4618	329	350	349	2273	65	2	1955	812	2285	1419	865	272	2276	3131	2274	1923	455	0	149	98	37	129	33
13_fr	5047	725	1730	2273	8678	648	566	642	3940	140	22	3746	1597	3984	2736	1550	425	4081	5599	4387	3480	767	0	288	119	41	209	55
14_fr	5312	689	1754	2149	8870	628	630	673	4278	143	2	3780	1610	4255	2713	1599	512	4271	5770	4467	3697	914	0	283	145	41	224	75
15_fr	1215	173	402	582	2195	150	134	148	969	27	6	950	387	906	697	417	103	985	1395	1037	893	206	0	63	36	3	48	20
01_en	2217	451	729	1316	3967	596	662	2060	1823	22	200	1204	656	1851	1897	525	19	1764	1942	2547	704	258	653	29	401	18	0	0
02_en	2761	551	777	1548	4543	685	769	2530	2163	13	284	1319	829	2218	2237	606	21	2019	2411	3083	861	295	769	37	475	31	0	0
03_en	990	183	271	557	1570	279	253	875	783	4	82	520	333	816	828	194	13	711	864	1048	298	94	254	8	145	15	0	0
04_en	2274	454	736	1315	3814	595	559	1978	1835	22	198	1073	690	1771	1865	514	33	1726	1918	2704	745	245	663	60	467	19	0	0
05_en	1865	400	553	1135	3210	515	525	1693	1482	7	153	949	571	1468	1586	517	17	1357	1646	2178	663	194	568	26	330	33	0	0
06_en	2606	518	797	1509	4237	687	669	2254	2097	26	216	1239	763	2174	2231	613	25	1931	2192	2955	899	277	733	49	464	37	0	0
07_en	4805	913	1521	2681	7834	1366	1163	4379	3838	42	416	2434	1461	3816	4091	1040	39	3674	4060	5369	1552	465	1332	74	843	52	0	0
08_en	2396	431	702	1416	4014	621	624	2171	2011	24	216	1152	748	2085	1947	527	33	1915	1966	2765	789	266	695	65	379	24	0	0
09_en	1993	408	653	1096	3373	575	517	1766	1648	16	146	861	629	1728	1698	442	20	1561	1626	2442	683	208	560	25	328	18	0	0
10_en	1627	359	451	933	2690	477	409	1475	1196	7	131	789	506	1266	1369	325	23	1211	1344	1759	502	181	410	31	255	20	0	0
11_en	2375	437	643	1364	3790	610	644	2217	1830	16	217	1122	799	1833	1948	486	23	1720	1945	2424	767	246	632	20	457	39	0	0
12_en	2560	489	757	1566	4331	677	650	2348	2033	28	234	1102	746	2125	2105	581	32	1939	2152	3046	750	278	721	35	418	40	0	0
13_en	4597	987	1462	2689	7963	1254	1201	4278	3634	39	432	2281	1493	3774	3911	1099	49	3577	3894	5540	1379	437	1374	77	673	49	0	0
14_en	4871	948	1439	2799	8179	1335	1140	4534	3829	36	427	2218	1534	4053	3989	1019	36	3689	3946	5858	1490	539	1377	90	856	49	0	0
15_en	1119	229	335	683	1994	323	281	1108	912	9	112	579	351	924	1004	305	9	863	997	1330	310	108	330	14	150	9	0	0

Each chapter (document) is modeled by a vector of 40 characters

Transposing the Matrix: Character Counts

Character counts per chapter in French, left part, and English, right part

	French														
	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
a	2503	2992	1042	2487	2014	2805	5062	2643	2126	1784	2641	2766	5047	5312	1215
b	365	391	152	303	268	368	706	325	289	249	381	373	725	689	173
c	857	1006	326	864	645	910	1770	869	771	546	817	935	1730	1754	402
d	1151	1388	489	1137	949	1266	2398	1085	920	805	1078	1237	2273	2149	582
e	4312	4993	1785	4158	3394	4535	8512	4229	3599	3002	4306	4618	8678	8870	2195
f	264	319	136	314	223	332	623	307	278	179	263	329	648	628	150
g	349	360	122	331	215	384	622	317	289	202	277	350	566	630	134
h	295	350	126	287	242	378	620	359	279	215	330	349	642	673	148
i	1945	2345	784	2028	1617	2219	4018	2102	1805	1319	1985	2273	3940	4278	969
j	65	81	41	57	67	97	126	85	52	60	114	65	140	143	27
k	4	6	7	3	3	3	19	4	6	5	0	2	22	2	6
l	1946	2128	816	1796	1513	1900	3726	1857	1499	1462	1886	1955	3746	3780	950
m	726	823	397	722	651	841	1596	811	619	598	900	812	1597	1610	387
n	1896	2308	778	1958	1547	2179	3851	2041	1711	1246	1966	2285	3984	4255	906
o	1372	1560	612	1318	1053	1569	2823	1367	1130	922	1356	1419	2736	2713	697
p	789	977	315	773	672	868	1532	833	651	557	763	865	1550	1599	417
q	248	281	102	274	166	285	468	239	187	172	230	272	425	512	103
r	1948	2376	792	2000	1601	2205	4015	2132	1719	1242	1912	2276	4081	4271	985
s	2996	3454	1174	2792	2192	3065	5634	2814	2404	1769	2564	3131	5599	5770	1395
t	1938	2411	856	2031	1736	2293	4116	2134	1763	1423	2218	2274	4387	4467	1037
u	1792	2069	707	1734	1396	1895	3518	1788	1448	1191	1737	1923	3480	3697	893
v	414	499	147	422	315	453	844	437	348	270	425	455	767	914	206
w	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
x	129	175	42	138	83	151	272	135	119	65	114	149	288	283	63
y	94	89	31	81	67	80	148	64	58	61	61	98	119	145	36
z	20	23	7	27	18	39	71	30	20	11	25	37	41	41	3
à	128	136	39	110	90	131	246	130	90	73	101	129	209	224	48
â	36	50	9	43	67	42	50	43	24	18	40	33	55	75	20
æ	0	1	0	0	0	0	1	0	2	0	0	0	3	0	2
ç	35	28	10	22	24	30	46	34	16	16	34	23	61	56	17
è	102	147	49	138	112	122	232	119	99	68	108	151	237	260	58
é	423	513	194	424	367	548	966	502	370	304	438	480	940	1019	221
ê	43	68	24	36	44	57	96	54	43	53	68	60	126	94	32
ë	1	0	0	0	1	0	2	0	1	0	1	0	0	0	0
î	17	20	12	15	11	15	42	11	8	15	26	13	32	28	12
ï	2	0	0	2	8	12	9	1	2	5	15	3	5	2	0
ô	20	20	27	15	23	15	41	14	13	38	50	15	37	45	24
ù	14	9	4	6	18	14	30	6	5	3	7	11	24	21	7
û	7	9	7	4	15	15	38	8	15	10	9	14	30	21	11
œ	5	5	2	8	7	9	9	5	3	5	7	0	13	12	6

	English											
	01	02	03	04	05	06	07	08	09	10	11	12
a	2217	2761	990	2274	1865	2606	4805	2396	1993	1627	2375	2560
b	451	551	183	454	400	518	913	431	408	359	437	489
c	729	777	271	736	553	797	1521	702	653	451	643	757
d	1316	1548	557	1315	1135	1509	2681	1416	1096	933	1364	1566
e	3967	4543	1570	3814	3210	4237	7834	4014	3373	2690	3790	4331
f	596	685	279	595	515	687	1366	621	575	477	610	677
g	662	769	253	559	525	669	1163	624	517	409	644	650
h	2060	2530	875	1978	1693	2254	4379	2171	1766	1475	2217	2348
i	1823	2163	783	1835	1482	2097	3838	2011	1648	1196	1830	2033
j	22	13	4	22	7	26	42	24	16	7	16	28
k	200	284	82	198	153	216	416	216	146	131	217	234
l	1204	1319	520	1073	949	1239	2434	1152	861	789	1122	1102
m	656	829	333	690	571	763	1461	748	629	506	799	746
n	1851	2218	816	1771	1468	2174	3816	2085	1728	1266	1833	2125
o	1897	2237	828	1865	1586	2231	4091	1947	1698	1369	1948	2105
p	525	606	194	514	517	613	1040	527	442	325	486	581
q	19	21	13	33	17	25	39	33	20	23	23	32
r	1764	2019	711	1726	1357	1931	3674	1915	1561	1211	1720	1939
s	1942	2411	864	1918	1646	2192	4060	1966	1626	1344	1945	2152
t	2547	3083	1048	2704	2178	2955	5369	2765	2442	1759	2424	3046
u	704	861	298	745	663	899	1552	789	683	502	767	750
v	258	295	94	245	194	277	465	266	208	181	246	278
w	653	769	254	663	568	733	1332	695	560	410	632	721
x	29	37	8	60	26	49	74	65	25	31	20	35
y	401	475	145	467	330	464	843	379	328	255	457	418
z	18	31	15	19	33	37	52	24	18	20	39	40
à	0	0	0	0	0	0	0	0	0	0	0	0
â	0	0	0	0	0	0	0	0	0	0	0	0
æ	0	0	0	0	0	0	0	0	0	0	0	0
ç	0	0	0	0	0	0	0	0	0	0	0	0
è	0	0	0	0	0	0	0	0	0	0	0	0
é	0	0	0	0	0	0	0	0	0	0	0	0
ê	0	0	0	0	0	0	0	0	0	0	0	0
ë	0	0	0	0	0	0	0	0	0	0	0	0
î	0	0	0	0	0	0	0	0	0	0	0	0
ï	0	0	0	0	0	0	0	0	0	0	0	0
ô	0	0	0	0	0	0	0	0	0	0	0	0
ù	0	0	0	0	0	0	0	0	0	0	0	0
û	0	0	0	0	0	0	0	0	0	0	0	0
œ	0	0	0	0	0	0	0	0	0	0	0	0

Each character is modeled by a vector of chapters.

Singular Value Decomposition

There are as many as 40 characters: the 26 unaccented letters from a to z and the 14 French accented letters: à, â, é, è, ê, ë, etc.

Singular value decomposition (SVD) reduces these dimensions, while keeping the resulting vectors semantically close

X is the $m \times n$ matrix of the letter counts per chapter, in our case, $m = 30$ and $n = 40$.

We can rewrite X as:

$$X = U\Sigma V^{\intercal},$$

where

- U is a matrix of dimensions $m \times m$,
- Σ, a diagonal matrix of dimensions $m \times n$, and
- V, a matrix of dimensions $n \times n$

The diagonal terms of Σ are called the **singular values** and are traditionally arranged by decreasing value.

To reduce the dimensions, we keep the highest values and set the rest to zero.

Code Example

Jupyter Notebook:
https://github.com/pnugues/pnlp/tree/main/notebooks

Vector Space Model

With the vector space model, we represent:

- Documents in a space of words (the words they contain) or
- Words in a space of documents (the documents that contain them).

Here, the rows are the words in the corpus, and the columns, the documents.

Words\D#	D_1	D_2	D_3	...	D_n
w_1	$C(w_1, D_1)$	$C(w_1, D_2)$	$C(w_1, D_3)$	...	$C(w_1, D_n)$
w_2	$C(w_2, D_1)$	$C(w_2, D_2)$	$C(w_2, D_3)$	...	$C(w_2, D_n)$
w_3	$C(w_3, D_1)$	$C(w_3, D_2)$	$C(w_3, D_3)$	...	$C(w_3, D_n)$
...	...	...	...	...	...
w_m	$C(w_m, D_1)$	$C(w_m, D_2)$	$C(w_m, D_3)$	...	$C(w_m, D_n)$

$C(w_i, D_j)$ can be Boolean values, counts, *tf* $\times$ *idf*, or a metric similar to *tf* $\times$ *idf* as in **latent semantic indexing**.

We can extend singular value decomposition from characters to words.

Word Embeddings

A PCA applied to this matrix will result in dense vectors representing the words.
Transposing the matrix, we represent documents with dense vectors
We compute the word embeddings with a singular value decomposition, where we truncate the matrix $\mathbf{U\Sigma}$ to 50, 100, 300, or 500 dimensions.
The word embeddings are the rows of this matrix.

Embeddings from Cooccurrences

We replace the documents with counts $C(w_i, w_j)$ in a context window

Words\Words	w_1	w_2	w_3	$\ldots$	w_n
w_1	$C(w_1, w_1)$	$C(w_1, w_2)$	$C(w_1, w_3)$	$\ldots$	$C(w_1, w_n)$
w_2	$C(w_2, w_1)$	$C(w_2, w_2)$	$C(w_2, w_3)$	$\ldots$	$C(w_2, w_n)$
w_3	$C(w_3, w_1)$	$C(w_3, w_2)$	$C(w_3, w_3)$	$\ldots$	$C(w_3, w_n)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
w_n	$C(w_n, w_1)$	$C(w_m, D_2)$	$C(w_m, D_3)$	$\ldots$	$C(w_n, D_n)$

We also extend singular value decomposition from characters to words.

Word Similarity

Contrary to one-hot encoder words, we can measure the similarity of two dense vectors **u** and **v**.
We usually measure the similarity between two embeddings **u** and **v** with the cosine similarity:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \, ||\mathbf{v}||},$$

ranging from -1 (most dissimilar) to 1 (most similar) or with the cosine distance ranging from 0 (closest) to 2 (most distant):

$$1 - \cos(\mathbf{u}, \mathbf{v}) = 1 - \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \, ||\mathbf{v}||}.$$

Embeddings in PyTorch

PyTorch has an embedding class. This is just a lookup table.
From PyTorch documentation:

```
embedding = nn.Embedding(10, 3)
input = torch.LongTensor([[1, 2, 4, 5], [4, 3, 2, 9]])
embedding(input)
```

```
embedding = nn.Embedding(10, 3, padding_idx=0)
input = torch.LongTensor([[0, 2, 0, 5]])
embedding(input)
```

Loading pretrained embeddings:

```
embeddings = nn.Embedding.from_pretrained(
  torch.FloatTensor(embedding_matrix),
  freeze=False,
  padding_idx=0)
```

Code Example

Jupyter Notebook:
`https://github.com/pnugues/pnlp/tree/main/notebooks`

Word2vec Embeddings

word2vec is another kind of embeddings that comes in two forms: CBOW and skipgrams.

CBOW uses a neural network architecture to predict a word given its surrounding context

The set up is similar to fill-the-missing-word questionnaires.

The missing word is called the focus word

CBOW embeddings corresponds to neural network parameters

The embeddings are trained on a corpus

Cloze Test

Guess a missing word given its context. Using the example:

Sing, O **goddess**, *the anger of Achilles son of Peleus,*

Cloze test: A reader, given the incomplete phrase:

```
Sing, O ____, the anger of Achilles
```

has to fill in the blank with the correct word, here **goddess**.

Word2vec Dataset

Using contexts of five words and training sentences such as:

Sing, O goddess, the anger of Achilles son of Peleus,

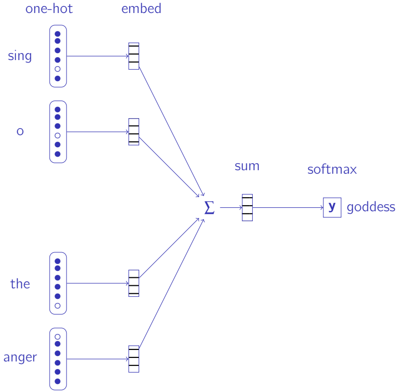
we generate a training set of contexts deprived of their focus word (X) and the focus word to predict ($\mathbf{y}$):

$$X = \begin{bmatrix} \text{sing} & \text{o} & \text{the} & \text{anger} \\ \text{o} & \text{goddess} & \text{anger} & \text{of} \\ \text{goddess} & \text{the} & \text{of} & \text{achilles} \\ \text{the} & \text{anger} & \text{achilles} & \text{son} \\ \text{anger} & \text{of} & \text{son} & \text{of} \\ \text{of} & \text{achilles} & \text{of} & \text{peleus} \end{bmatrix} ; \mathbf{y} = \begin{bmatrix} \text{goddess} \\ \text{the} \\ \text{anger} \\ \text{of} \\ \text{achilles} \\ \text{son} \end{bmatrix}$$

CBOW Architecture

We train a neural network to get the CBOW embeddings: N dimension of the embeddings, V size of the vocabulary.

1. Embedding table (V, N) and sum;
2. Linear layer (N, V) and softmax.



Embedding Bags in PyTorch

`EmbeddingBags` class creates embedding objects.

```
embedding_bag = nn.EmbeddingBag(MAX_CHARS, 64, mode='sum')
```

Given a list of embeddings (a list of rows) as input, an embedding bag returns the weighted sum of the embeddings.
We specify the weights with a `per_sample_weights` parameter.
`https://pytorch.org/docs/stable/generated/torch.nn.EmbeddingBag.html`

Programming Embedding Bags in PyTorch (I)

```
embedding_bag = nn.EmbeddingBag(MAX_CHARS, 64, mode='sum')

# Computes the sum of rows 1 and 2 and rows 3 and 4
# The result is a matrix of two rows
embedding_bag(torch.tensor([[1, 2], [3, 4]]))

embedding_bag(torch.tensor([[1, 2], [3, 4]]),
           per_sample_weights=torch.tensor([[0.5, 0.5],
           [0.2, 0.8]]))
```

Code Example

Jupyter Notebook:
https://github.com/pnugues/pnlp/tree/main/notebooks

Examples

Words closest to *he*, *she*, *London*, *table*, and *Monday* with CBOW embeddings trained on a corpus of Dickens novels:

```
he ['she', 'they', 'it', 'be', 'that']
she ['he', 'they', 'it', 'i', 'be']
london ['paris', 'england', 'town', 'india', 'dover']
table ['desk', 'counter', 'box', 'sofa', 'ground']
monday ['sunday', 'thursday', 'saturday', 'noon', 'wednesday']
```

Code Example

Jupyter Notebook:
https://github.com/pnugues/pnlp/tree/main/notebooks

Glove Embeddings

There are many kinds of word embeddings: Global vectors (GloVe) is one of them

We can replace documents by a context of a few words to the left and to the right of the focus word: w_i.

A context C_j is then defined by a window of $2K$ words centered on the word:

$$\begin{array}{ll} \text{Word:} & w_i, \\ \text{Context:} & w_{i-K}, w_{i-K+1}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+K-1}, w_{i+K}, \end{array}$$

where the context representation uses a bag of words.

We can even reduce the context to a single word to the left or to the right of w_i and use bigrams.

Glove Embeddings

We store counts of word pairs (w_i, w_j) in a matrix:

Words	w_1	w_2	w_3	...	w_n
w_1	X_{11}	X_{12}	X_{13}	...	X_{1n}
w_2	X_{21}	X_{22}	X_{23}	...	X_{2n}
w_3	X_{31}	X_{32}	X_{33}	...	X_{3n}
...	...	...	...	...	...
w_n	X_{n1}	X_{n2}	X_{n3}	...	X_{nn}

X_{ij} is the number of times word w_j occurs in the context of word w_i, for instance 10 words to the left and 10 to the right

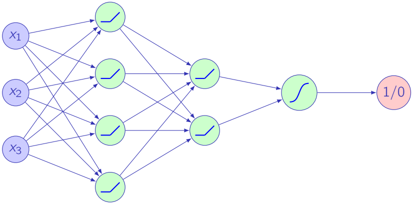
To train the embeddings, we minimize the loss (simplified):

$$J = \sum_{i,j=1}^{V} (\mathbf{w}_i \cdot \mathbf{w}_j - \log X_{ij})^2,$$

where $\mathbf{w}_i$, resp. $\mathbf{w}_j$, is the embedding vector of word of index i, resp. j.

Using Word Embeddings

We can use word embeddings to replace one-hot vectors as they will make the representation much more compact.



In a text categorization task, for instance, you would use a window of words (for instance the 200 first words of the document), where each word would be represented by its embedding.
The input layer is then called an **embedding layer.**
The embeddings are trainable parameters that you can initialize with pre-trained embeddings or random values.

Popular Word Embeddings

Embeddings from large corpora are obtained with iterative techniques

Some popular embedding algorithms with open source programs:

- word2vec: `https://github.com/tmikolov/word2vec`
- GloVe: Global Vectors for Word Representation
  `https://nlp.stanford.edu/projects/glove/`
- fastText: `https://fasttext.cc/`

To derive word embeddings, you will have to apply these programs on a very large corpus

Embeddings for many languages are also publicly available. You just download them

gensim is a Python library to create word embeddings from a corpus.
`https://radimrehurek.com/gensim/index.html`