

EDAN95

Applied Machine Learning

<http://cs.lth.se/edan95/>
Lecture 7: Recurrent Neural Networks

Pierre Nugues

Lund University
`Pierre.Nugues@cs.lth.se`
http://cs.lth.se/pierre_nugues/

November 25, 2019

Natural Language Processing

High-level applications:

- Spoken interaction: Apple Siri, Google Assistant, Amazon Echo
- Speech dictation of letters or reports: *Windows 10, macOS*
- Question answering: *IBM Watson and Jeopardy!*

The inner engines of these applications are powered by neural networks.
Big change from the

- 1980's (rules),
- 1990's (Bayes), and
- 2000's (SVM and logistic regression, still usable for most tasks).

Neural net expansion started in 2010. What in 2030?

Scope of this Course

Applications we will consider:

- 1 Text categorization
- 2 Word or segment categorization
- 3 Translation: *Google Translate, DeepL, Bing translator, etc.*

Text Categorization

- 1 spam/not spam
- 2 Language identification, French, English, or Spanish?
(<https://github.com/google/cld3>)
- 3 Sentiment analysis: Is this comment on my favorite tooth paste positive, neutral, or negative?
- 4 Newswire categorization.

Text Categorization: The Reuters Corpus

```
<title>USA: Tylan stock jumps; weighs sale of company.</title>
<headline>Tylan stock jumps; weighs sale of company.</headline>
<dateline>SAN DIEGO</dateline>
<text>
<p>The stock of Tylan General Inc. jumped Tuesday after the maker of
process-management equipment said it is exploring the sale of the company and
added that it has already received some inquiries from potential buyers.</p>
<p>Tylan was up $2.50 to $12.75 in early trading on the Nasdaq market.</p>
<p>The company said it has set up a committee of directors to oversee the sale and
that Goldman, Sachs & Co. has been retained as its financial adviser.</p>
</text>
<metadata>
<codes class="bip:topics:1.0">
<code code="C15"/>
<code code="C152"/>
<code code="C18"/>
<code code="C181"/>
<code code="CCAT"/>
</codes>
```

Text Categorization: The Categories

In total 103 topic categories:

| | | | |
|-------|----------------------|-------|-----------------------|
| C11 | STRATEGY/PLANS | C15 | PERFORMANCE |
| C12 | LEGAL/JUDICIAL | C151 | ACCOUNTS/EARNINGS |
| C13 | REGULATION/POLICY | C1511 | ANNUAL RESULTS |
| C14 | SHARE LISTINGS | C152 | COMMENT/FORECASTS |
| C15 | PERFORMANCE | C16 | INSOLVENCY/LIQUIDITY |
| C151 | ACCOUNTS/EARNINGS | C17 | FUNDING/CAPITAL |
| C1511 | ANNUAL RESULTS | C171 | SHARE CAPITAL |
| C152 | COMMENT/FORECASTS | C172 | BONDS/DEBT ISSUES |
| C16 | INSOLVENCY/LIQUIDITY | C173 | LOANS/CREDITS |
| C17 | FUNDING/CAPITAL | C174 | CREDIT RATINGS |
| C171 | SHARE CAPITAL | C18 | OWNERSHIP CHANGES |
| C172 | BONDS/DEBT ISSUES | C181 | MERGERS/ACQUISITIONS |
| C173 | LOANS/CREDITS | C182 | ASSET TRANSFERS |
| C174 | CREDIT RATINGS | C183 | PRIVATISATIONS |
| C11 | STRATEGY/PLANS | C21 | PRODUCTION/SERVICES |
| C12 | LEGAL/JUDICIAL | C22 | NEW PRODUCTS/SERVICES |
| C13 | REGULATION/POLICY | C23 | RESEARCH/DEVELOPMENT |
| C14 | SHARE LISTINGS | ... | |

Encoding Words

Neural networks can only handle numbers

We need then to encode the words or the characters with numbers.

Using ordinal numbers (a:1, b:2, c:3, d:4, etc) is impossible.

Is a closer to b , than c ?

The most simple encoding is the one-hot encoding (or contrast encoding), that we have seen in the 3rd lecture.

Matrix Notation

- A feature vector (predictors): \mathbf{x} , and feature matrix: \mathbf{X} ;
- The class: y and the class vector: \mathbf{y} ;
- The predicted class (response): \hat{y} , and predicted class vector: $\hat{\mathbf{y}}$

$$\mathbf{X} = \begin{bmatrix} \text{Sunny} & \text{Hot} & \text{High} & \text{False} \\ \text{Sunny} & \text{Hot} & \text{High} & \text{True} \\ \text{Overcast} & \text{Hot} & \text{High} & \text{False} \\ \text{Rain} & \text{Mild} & \text{High} & \text{False} \\ \text{Rain} & \text{Cool} & \text{Normal} & \text{False} \\ \text{Rain} & \text{Cool} & \text{Normal} & \text{True} \\ \text{Overcast} & \text{Cool} & \text{Normal} & \text{True} \\ \text{Sunny} & \text{Mild} & \text{High} & \text{False} \\ \text{Sunny} & \text{Cool} & \text{Normal} & \text{False} \\ \text{Rain} & \text{Mild} & \text{Normal} & \text{False} \\ \text{Sunny} & \text{Mild} & \text{Normal} & \text{True} \\ \text{Overcast} & \text{Mild} & \text{High} & \text{True} \\ \text{Overcast} & \text{Hot} & \text{Normal} & \text{False} \\ \text{Rain} & \text{Mild} & \text{High} & \text{True} \end{bmatrix}; \mathbf{y} = \begin{bmatrix} \text{N} \\ \text{N} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{N} \\ \text{P} \\ \text{N} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{N} \end{bmatrix}$$

Converting Symbolic Attributes into Numerical Vectors

Linear classifiers are numerical systems.

Symbolic – nominal – attributes are mapped onto vectors of binary values.

This is called a one-hot encoding

A conversion of the weather data set.

| Object | Attributes | | | | | | | | | | Class |
|--------|------------|----------|------|-------------|------|------|----------|--------|-------|-------|-------|
| | Outlook | | | Temperature | | | Humidity | | Windy | | |
| | Sunny | Overcast | Rain | Hot | Mild | Cool | High | Normal | True | False | |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | N |
| 2 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | N |
| 3 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | P |
| 4 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | P |
| 5 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | P |
| 6 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | N |
| 7 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | P |
| 8 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | N |
| 9 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | P |
| 10 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | P |
| 11 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | P |
| 12 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | P |
| 13 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | P |
| 14 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | N |

Code Example

Jupyter Notebook: Chollet 6.1 <https://github.com/fchollet/deep-learning-with-python-notebooks/6.1-one-hot-encoding-of-words-or-characters.ipynb>

Text Categorization Using Bags of Words

A first technique to carry out categorization is to represent documents as vectors in a space of words.

The word order plays no role and it is often called a *bag-of-word model*.

To represent the two documents:

D1: Chrysler plans new investments in Latin America.

D2: Chrysler plans major investments in Mexico.

We would have:

| D#\ Words | america | chrysler | in | investments | latin | major | mexico | new | plans | Category |
|-----------|---------|----------|----|-------------|-------|-------|--------|-----|-------|----------|
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | spam |
| 2 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | nospam |

Text Categorization Using TFIDF

For a collection of documents, we would have a word by document matrix. As parameter, (w_i, D_j) could contain the frequency of w_i in document D_j

| D#\Words | w_1 | w_2 | w_3 | ... | w_m | Category |
|----------|---------------|---------------|---------------|-----|---------------|----------|
| D_1 | $C(w_1, D_1)$ | $C(w_2, D_1)$ | $C(w_3, D_1)$ | ... | $C(w_m, D_1)$ | spam |
| D_2 | $C(w_1, D_2)$ | $C(w_2, D_2)$ | $C(w_3, D_2)$ | ... | $C(w_m, D_2)$ | nospam |
| ... | | | | | | |
| D_n | $C(w_1, D_n)$ | $C(w_2, D_n)$ | $C(w_3, D_n)$ | ... | $C(w_m, D_n)$ | spam |

Most of the time, the counts are replaced by the term frequency times the inverse document frequency: $tf \times idf$.

The term frequencies $tf_{i,j}$ are normalized by the sum of the frequencies of all the terms in the document:

$$tf_{i,j} = \frac{t_{i,j}}{\sum_i t_{i,j}},$$

The inverse document frequency is defined as:

$$idf_i = \log\left(\frac{N}{n_i}\right),$$

where N is the total number of documents in the collection

Code Example

Jupyter Notebooks: Chollet 3.5 and 3.6 <https://github.com/fchollet/deep-learning-with-python-notebooks>
3.5-classifying-movie-reviews.ipynb
3.6-classifying-newswires.ipynb

Dimension Reduction

One-hot encoding of TFIDF encoding can produce very long vectors: Imagine a vocabulary of one million words per language with 100 languages. A solution is to produce dense vectors also called word embeddings using a dimension reduction

This reduction is very close to principal component analysis or singular value decomposition

It can be automatically obtained through training or initialized with pretrained vectors

Principal Component Analysis

We will use a small dataset to explain principal component analysis: The characters in *Salammô*

| Ch. | a' | b' | c' | d' | e' | f' | g' | h' | i' | j' | k' | l' | m' | n' | o' | p' | q' | r' | s' | t' | u' | v' |
|-------|------|-----|------|------|------|------|------|------|------|-----|-----|------|------|------|------|------|-----|------|------|------|------|-----|
| 01_fr | 2503 | 365 | 857 | 1151 | 4312 | 264 | 349 | 295 | 1945 | 65 | 4 | 1946 | 726 | 1896 | 1372 | 789 | 248 | 1948 | 2996 | 1938 | 1792 | 414 |
| 02_fr | 2992 | 391 | 1006 | 1388 | 4993 | 319 | 360 | 350 | 2345 | 81 | 6 | 2128 | 823 | 2308 | 1560 | 977 | 281 | 2376 | 3454 | 2411 | 2069 | 499 |
| 03_fr | 1042 | 152 | 326 | 489 | 1785 | 136 | 122 | 126 | 784 | 41 | 7 | 816 | 397 | 778 | 612 | 315 | 102 | 792 | 1174 | 856 | 707 | 147 |
| 04_fr | 2487 | 303 | 864 | 1137 | 4158 | 314 | 331 | 287 | 2028 | 57 | 3 | 1796 | 722 | 1958 | 1318 | 773 | 274 | 2000 | 2792 | 2031 | 1734 | 422 |
| 05_fr | 2014 | 268 | 645 | 949 | 3394 | 223 | 215 | 242 | 1617 | 67 | 3 | 1513 | 651 | 1547 | 1053 | 672 | 166 | 1601 | 2192 | 1736 | 1396 | 315 |
| 06_fr | 2805 | 368 | 910 | 1266 | 4535 | 332 | 384 | 378 | 2219 | 97 | 3 | 1900 | 841 | 2179 | 1569 | 868 | 285 | 2205 | 3065 | 2293 | 1895 | 453 |
| 07_fr | 5062 | 706 | 1770 | 2398 | 8512 | 623 | 622 | 620 | 4018 | 126 | 19 | 3726 | 1596 | 3851 | 2823 | 1532 | 468 | 4015 | 5634 | 4116 | 3518 | 844 |
| 08_fr | 2643 | 325 | 869 | 1085 | 4229 | 307 | 317 | 359 | 2102 | 85 | 4 | 1857 | 811 | 2041 | 1367 | 833 | 239 | 2132 | 2814 | 2134 | 1788 | 437 |
| 09_fr | 2126 | 289 | 771 | 920 | 3599 | 278 | 289 | 279 | 1805 | 52 | 6 | 1499 | 619 | 1711 | 1130 | 651 | 187 | 1719 | 2404 | 1763 | 1448 | 348 |
| 10_fr | 1784 | 249 | 546 | 805 | 3002 | 179 | 202 | 215 | 1319 | 60 | 5 | 1462 | 598 | 1246 | 922 | 557 | 172 | 1242 | 1769 | 1423 | 1191 | 270 |
| 11_fr | 2641 | 381 | 817 | 1078 | 4306 | 263 | 277 | 330 | 1985 | 114 | 0 | 1886 | 900 | 1966 | 1356 | 763 | 230 | 1912 | 2564 | 2218 | 1737 | 425 |
| 12_fr | 2766 | 373 | 935 | 1237 | 4618 | 329 | 350 | 349 | 2273 | 65 | 2 | 1955 | 812 | 2285 | 1419 | 865 | 272 | 2276 | 3131 | 2274 | 1923 | 455 |
| 13_fr | 5047 | 725 | 1730 | 2273 | 8678 | 648 | 566 | 642 | 3940 | 140 | 22 | 3746 | 1597 | 3984 | 2736 | 1550 | 425 | 4081 | 5599 | 4387 | 3480 | 767 |
| 14_fr | 5312 | 689 | 1754 | 2149 | 8870 | 628 | 630 | 673 | 4278 | 143 | 2 | 3780 | 1610 | 4255 | 2713 | 1599 | 512 | 4271 | 5770 | 4467 | 3697 | 914 |
| 15_fr | 1215 | 173 | 402 | 582 | 2195 | 150 | 134 | 148 | 969 | 27 | 6 | 950 | 387 | 906 | 697 | 417 | 103 | 985 | 1395 | 1037 | 893 | 206 |
| 01_en | 2217 | 451 | 729 | 1316 | 3967 | 596 | 662 | 2060 | 1823 | 22 | 200 | 1204 | 656 | 1851 | 1897 | 525 | 19 | 1764 | 1942 | 2547 | 704 | 258 |
| 02_en | 2761 | 551 | 777 | 1548 | 4543 | 685 | 769 | 2530 | 2163 | 13 | 284 | 1319 | 829 | 2218 | 2237 | 606 | 21 | 2019 | 2411 | 3083 | 861 | 295 |
| 03_en | 990 | 183 | 271 | 557 | 1570 | 279 | 253 | 875 | 783 | 4 | 82 | 520 | 333 | 816 | 828 | 194 | 13 | 711 | 864 | 1048 | 298 | 94 |
| 04_en | 2274 | 454 | 736 | 1315 | 3814 | 595 | 559 | 1978 | 1835 | 22 | 198 | 1073 | 690 | 1771 | 1865 | 514 | 33 | 1726 | 1918 | 2704 | 745 | 245 |
| 05_en | 1865 | 400 | 553 | 1135 | 3210 | 515 | 525 | 1693 | 1482 | 7 | 153 | 940 | 571 | 1468 | 1586 | 517 | 17 | 1357 | 1646 | 2178 | 663 | 194 |
| 06_en | 2606 | 518 | 797 | 1509 | 4237 | 687 | 669 | 2254 | 2097 | 26 | 216 | 1239 | 763 | 2174 | 2231 | 613 | 25 | 1931 | 2192 | 2955 | 899 | 277 |
| 07_en | 4805 | 913 | 1521 | 2681 | 7834 | 1366 | 1163 | 4379 | 3838 | 42 | 416 | 2434 | 1461 | 3816 | 4091 | 1040 | 39 | 3674 | 4060 | 5369 | 1552 | 465 |
| 08_en | 2396 | 431 | 702 | 1416 | 4014 | 621 | 624 | 2171 | 2011 | 24 | 216 | 1152 | 748 | 2085 | 1947 | 527 | 33 | 1915 | 1966 | 2765 | 789 | 266 |
| 09_en | 1993 | 408 | 653 | 1096 | 3373 | 575 | 517 | 1766 | 1648 | 16 | 146 | 861 | 629 | 1728 | 1698 | 442 | 20 | 1561 | 1626 | 2442 | 683 | 208 |
| 10_en | 1627 | 359 | 451 | 933 | 2690 | 477 | 409 | 1475 | 1196 | 7 | 131 | 789 | 506 | 1266 | 1369 | 325 | 23 | 1211 | 1344 | 1759 | 502 | 181 |
| 11_en | 2375 | 437 | 643 | 1364 | 3790 | 610 | 644 | 2217 | 1830 | 16 | 217 | 1122 | 799 | 1833 | 1948 | 486 | 23 | 1720 | 1945 | 2424 | 767 | 246 |
| 12_en | 2560 | 489 | 757 | 1566 | 4331 | 677 | 650 | 2348 | 2033 | 28 | 234 | 1102 | 746 | 2125 | 2105 | 581 | 32 | 1939 | 2152 | 3046 | 750 | 278 |
| 13_en | 4597 | 987 | 1462 | 2689 | 7963 | 1254 | 1201 | 4278 | 3634 | 39 | 432 | 2281 | 1493 | 3774 | 3911 | 1099 | 49 | 3577 | 3894 | 5540 | 1379 | 437 |
| 14_en | 4871 | 948 | 1439 | 2799 | 8179 | 1335 | 1140 | 4534 | 3829 | 36 | 427 | 2218 | 1534 | 4053 | 3989 | 1019 | 36 | 3689 | 3946 | 5858 | 1490 | 539 |
| 15_en | 1119 | 229 | 335 | 683 | 1994 | 323 | 281 | 1108 | 912 | 9 | 112 | 579 | 351 | 924 | 1004 | 305 | 9 | 863 | 997 | 1330 | 310 | 108 |

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

Each chapter is modeled by a vector of characters. 

Character Counts

| | French | | | | | | | | | | | | | | | English | | | | | | | | | | | | |
|---|--------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|---------|------|------|------|------|------|------|------|------|------|------|----|----|
| | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 |
| a | 2503 | 2992 | 1042 | 2487 | 2014 | 2805 | 5062 | 2643 | 2126 | 1784 | 2641 | 2766 | 5047 | 5312 | 1215 | 2217 | 2761 | 990 | 2274 | 1865 | 2606 | 4805 | 2396 | 1993 | 1627 | 2375 | | |
| b | 365 | 391 | 152 | 303 | 268 | 368 | 706 | 325 | 289 | 249 | 381 | 373 | 725 | 689 | 173 | 451 | 551 | 183 | 454 | 400 | 518 | 913 | 431 | 408 | 359 | 437 | | |
| c | 857 | 1006 | 326 | 864 | 645 | 910 | 1770 | 869 | 771 | 546 | 817 | 935 | 1730 | 1754 | 402 | 729 | 777 | 271 | 736 | 553 | 797 | 1521 | 702 | 653 | 451 | 643 | | |
| d | 1151 | 1388 | 489 | 1137 | 949 | 1256 | 2398 | 1085 | 920 | 805 | 1078 | 1237 | 2273 | 2149 | 582 | 1316 | 1548 | 557 | 1315 | 1135 | 1509 | 2681 | 1416 | 1096 | 933 | 1364 | | |
| e | 4312 | 4993 | 1785 | 4158 | 3394 | 4535 | 8512 | 4229 | 3599 | 3002 | 4306 | 4618 | 8678 | 8870 | 2195 | 3967 | 4543 | 1570 | 3814 | 3210 | 4237 | 7834 | 4014 | 3373 | 2690 | 3790 | | |
| f | 264 | 319 | 136 | 314 | 223 | 332 | 623 | 307 | 278 | 179 | 263 | 329 | 648 | 628 | 150 | 596 | 685 | 279 | 595 | 515 | 687 | 1366 | 621 | 575 | 477 | 610 | | |
| g | 349 | 360 | 122 | 331 | 215 | 384 | 622 | 317 | 289 | 202 | 277 | 350 | 566 | 630 | 134 | 662 | 769 | 253 | 559 | 525 | 669 | 1163 | 624 | 517 | 409 | 644 | | |
| h | 295 | 350 | 126 | 287 | 242 | 378 | 620 | 359 | 279 | 215 | 330 | 349 | 642 | 673 | 148 | 2060 | 2530 | 875 | 1978 | 1693 | 2254 | 4379 | 2171 | 1766 | 1475 | 2217 | | |
| i | 1945 | 2345 | 784 | 2028 | 1617 | 2219 | 4018 | 2102 | 1805 | 1319 | 1985 | 2273 | 3940 | 4278 | 969 | 1823 | 2163 | 783 | 1835 | 1482 | 2097 | 3838 | 2011 | 1648 | 1196 | 1830 | | |
| j | 65 | 81 | 41 | 57 | 67 | 97 | 126 | 85 | 52 | 60 | 114 | 65 | 140 | 143 | 27 | 22 | 13 | 4 | 22 | 7 | 26 | 42 | 24 | 16 | 7 | 16 | | |
| k | 4 | 6 | 7 | 3 | 3 | 3 | 19 | 4 | 6 | 5 | 0 | 2 | 22 | 2 | 6 | 200 | 284 | 82 | 198 | 153 | 216 | 416 | 216 | 146 | 131 | 217 | | |
| l | 1946 | 2128 | 816 | 1796 | 1513 | 1900 | 3726 | 1857 | 1499 | 1462 | 1886 | 1955 | 3746 | 3780 | 950 | 1204 | 1319 | 520 | 1073 | 949 | 1239 | 2434 | 1152 | 861 | 789 | 1122 | | |
| m | 726 | 823 | 397 | 722 | 651 | 841 | 1596 | 811 | 619 | 598 | 900 | 812 | 1597 | 1610 | 387 | 656 | 829 | 333 | 690 | 571 | 763 | 1461 | 748 | 629 | 506 | 799 | | |
| n | 1896 | 2308 | 778 | 1958 | 1547 | 2179 | 3851 | 2041 | 1711 | 1246 | 1966 | 2285 | 3984 | 4255 | 906 | 1851 | 2218 | 816 | 1771 | 1468 | 2174 | 3816 | 2085 | 1728 | 1286 | 1833 | | |
| o | 1372 | 1560 | 612 | 1318 | 1053 | 1569 | 2823 | 1367 | 1130 | 922 | 1356 | 1419 | 2736 | 2713 | 697 | 1897 | 2237 | 828 | 1865 | 1586 | 2231 | 4091 | 1947 | 1698 | 1369 | 1948 | | |
| p | 789 | 977 | 315 | 773 | 672 | 868 | 1532 | 833 | 651 | 557 | 783 | 865 | 1550 | 1599 | 417 | 525 | 606 | 194 | 514 | 517 | 613 | 1040 | 527 | 442 | 325 | 486 | | |
| q | 248 | 281 | 102 | 274 | 166 | 285 | 468 | 239 | 187 | 172 | 230 | 272 | 425 | 512 | 103 | 19 | 21 | 13 | 33 | 17 | 25 | 39 | 33 | 20 | 23 | 23 | | |
| r | 1948 | 2376 | 792 | 2000 | 1601 | 2205 | 4015 | 2132 | 1719 | 1242 | 1912 | 2276 | 4081 | 4271 | 985 | 1764 | 2019 | 711 | 1726 | 1357 | 1931 | 3674 | 1915 | 1561 | 1211 | 1720 | | |
| s | 2996 | 3454 | 1174 | 2792 | 2192 | 3065 | 5634 | 2814 | 2404 | 1769 | 2564 | 3131 | 5599 | 5770 | 1395 | 1942 | 2411 | 864 | 1918 | 1646 | 2192 | 4060 | 1966 | 1626 | 1344 | 1945 | | |
| t | 1938 | 2411 | 856 | 2031 | 1736 | 2293 | 4116 | 2134 | 1763 | 1423 | 2218 | 2274 | 4387 | 4667 | 1037 | 2547 | 3083 | 1048 | 2704 | 2178 | 2955 | 5369 | 2765 | 2442 | 1759 | 2424 | | |
| u | 1792 | 2069 | 707 | 1734 | 1396 | 1895 | 3518 | 1788 | 1448 | 1191 | 1737 | 1923 | 3480 | 3697 | 893 | 704 | 861 | 298 | 745 | 663 | 899 | 1552 | 789 | 683 | 502 | 767 | | |
| v | 414 | 499 | 147 | 422 | 315 | 453 | 844 | 437 | 348 | 270 | 425 | 455 | 767 | 914 | 206 | 258 | 295 | 94 | 245 | 194 | 277 | 465 | 266 | 208 | 181 | 246 | | |
| w | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 653 | 769 | 254 | 663 | 568 | 733 | 1332 | 695 | 560 | 410 | 632 | | |
| x | 129 | 175 | 42 | 138 | 83 | 151 | 272 | 135 | 119 | 65 | 114 | 140 | 288 | 283 | 63 | 29 | 37 | 8 | 60 | 26 | 49 | 74 | 65 | 25 | 31 | 20 | | |
| y | 94 | 89 | 31 | 81 | 67 | 80 | 148 | 64 | 58 | 61 | 61 | 98 | 119 | 145 | 36 | 401 | 475 | 145 | 467 | 330 | 464 | 843 | 379 | 328 | 255 | 457 | | |
| z | 20 | 23 | 7 | 27 | 18 | 39 | 71 | 30 | 20 | 11 | 25 | 37 | 41 | 41 | 3 | 18 | 31 | 15 | 19 | 33 | 37 | 52 | 24 | 18 | 20 | 39 | | |
| ā | 128 | 136 | 39 | 110 | 90 | 131 | 246 | 130 | 90 | 73 | 101 | 129 | 209 | 224 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| ä | 36 | 50 | 9 | 43 | 67 | 42 | 50 | 43 | 24 | 18 | 40 | 33 | 55 | 75 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| æ | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 3 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| ç | 35 | 28 | 10 | 22 | 24 | 30 | 46 | 34 | 16 | 16 | 34 | 23 | 61 | 56 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| é | 102 | 147 | 49 | 138 | 112 | 122 | 232 | 119 | 99 | 68 | 108 | 151 | 237 | 260 | 58 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| ë | 423 | 513 | 194 | 424 | 367 | 548 | 966 | 502 | 370 | 304 | 438 | 480 | 940 | 1019 | 221 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| ê | 43 | 68 | 24 | 36 | 44 | 57 | 96 | 54 | 43 | 53 | 68 | 60 | 126 | 94 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| è | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| ï | 17 | 20 | 12 | 15 | 11 | 15 | 42 | 11 | 8 | 15 | 26 | 13 | 32 | 28 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| ï | 2 | 0 | 0 | 0 | 2 | 8 | 12 | 9 | 1 | 2 | 5 | 15 | 3 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| ô | 20 | 20 | 27 | 15 | 23 | 15 | 41 | 14 | 13 | 38 | 50 | 15 | 37 | 45 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| ö | 14 | 9 | 4 | 6 | 18 | 14 | 30 | 6 | 5 | 3 | 7 | 11 | 24 | 41 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| ü | 7 | 9 | 7 | 4 | 15 | 15 | 38 | 8 | 15 | 10 | 9 | 14 | 30 | 21 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| œ | 5 | 5 | 2 | 8 | 7 | 9 | 9 | 5 | 3 | 5 | 7 | 0 | 13 | 12 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |

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

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

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:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

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.

We keep the highest values and set the rest to zero.

Code Example

Jupyter Notebook 3.1-SVD

Word Embeddings

We can extend singular value decomposition from characters to words. The rows will represent the words in the corpus, and the columns, documents,

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:

$$w_{i-K}, w_{i-K+1}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+K-1}, w_{i+K},$$

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.

Word Embeddings

We store the word-context pairs (w_i, C_j) in a matrix.

Each matrix element measures the association strength between word w_i and context C_j , for instance mutual information.

Mutual information, often called pointwise mutual information (the strength of an association) is defined as:

$$I(w_i, w_j) = \log_2 \frac{P(w_i, w_j)}{P(w_i)P(w_j)} \approx \log_2 \frac{N \cdot C(w_i, w_j)}{C(w_i)C(w_j)}.$$

| D# \ Words | C_1 | C_2 | C_3 | ... | C_n |
|------------|----------------|----------------|----------------|-----|----------------|
| w_1 | $MI(w_1, C_1)$ | $MI(w_1, C_2)$ | $MI(w_1, C_3)$ | ... | $MI(w_1, C_n)$ |
| w_2 | $MI(w_2, C_1)$ | $MI(w_2, C_2)$ | $MI(w_2, C_3)$ | ... | $MI(w_2, C_n)$ |
| w_3 | $MI(w_3, C_1)$ | $MI(w_3, C_2)$ | $MI(w_3, C_3)$ | ... | $MI(w_3, C_n)$ |
| ... | ... | ... | ... | ... | ... |
| w_m | $MI(w_m, C_1)$ | $MI(w_m, C_2)$ | $MI(w_m, C_3)$ | ... | $MI(w_m, C_n)$ |

Word Embeddings

We compute the word embeddings with a singular value decomposition, where we truncate the $\mathbf{U}\Sigma$ matrix to 50, 100, 300, or 500 dimensions.

The word embeddings are the rows of this matrix.

We usually measure the similarity between two embeddings \vec{u} and \vec{v} with the cosine similarity:

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

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

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

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/>

ELMo: <https://allennlp.org/elmo>

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>

Semantic Similarity

Word embeddings mitigate the dimension problem relatively to one-hot encoding

In addition, similar words will have similar vectors

Demo: http://bionlp-www.utu.fi/wv_demo/

This enables to cope with words unseen in a training set

Text Categorization

Following Chollet, we will now train a network to categorize movie reviews. We will use embeddings and a feedforward network first. We will then use recurrent networks.

Structure of a Network

First, a network, where we train the embeddings (Chollet, Listing 6.7):

```
model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
model.summary()
history = model.fit(x_train, y_train,
                   epochs=10,
                   batch_size=32,
                   validation_split=0.2)
```

Using GloVe Embeddings

We create a dictionary, where the keys are the words and the value, the embedding vector

```
glove_dir = '/Users/pierre/Documents/Cours/EDAN20/programs/ch08'
embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.100d.txt'))

for line in f:
    values = line.strip().split()
    word = values[0]
    vector = np.array(values[1:], dtype='float32')
    embeddings_index[word] = vector
f.close()

print('Found %s word vectors.' % len(embeddings_index))
```

Initializing the Matrix

We create the embeddings matrix by using the GloVe embedding or the 0, if not in GloVe

```
embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

Building the Network

The embedding layer is set to the GloVe parameters.

```
model = Sequential()
model.add(Embedding(max_words, embedding_dim,
    input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

Complete Code Example

Jupyter Notebook: Chollet 6.1 <https://github.com/fchollet/deep-learning-with-python-notebooks/6.1-using-word-embeddings.ipynb>

Recurrent Neural Networks

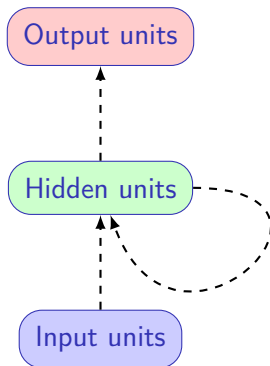
In feed-forward networks, predictions in a sequence of classifications are independent.

In many cases, given an input, the prediction also depends on the previous decision.

For instance, in weather forecast, if the input is the temperature and the output is rain/not rain, for a same temperature, if the previous output was rain, the next one is likely to be rain.

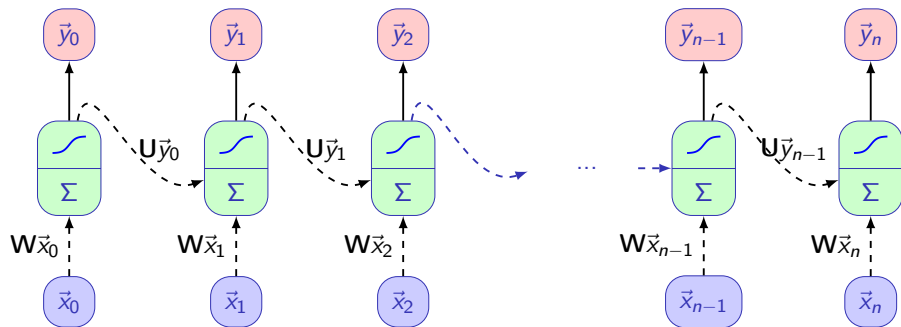
This is modeled by recurrent neural networks (RNN)

The RNN Architecture



A simple recurrent neural network; the dashed lines represent trainable connections.

The Unfolded RNN Architecture



The network unfolded in time. Equation used by implementations¹.

$$\mathbf{y}_{(t)} = \tanh(\mathbf{W} \cdot \mathbf{x}_{(t)} + \mathbf{U} \cdot \mathbf{y}_{(t-1)} + \mathbf{b})$$

¹See: <https://pytorch.org/docs/stable/nn.html#torch.nn.RNN>

Building a Simple RNN with Keras

```
model = Sequential()  
model.add(Embedding(max_features, 32))  
model.add(SimpleRNN(32))  
model.add(Dense(1, activation='sigmoid'))  
model.summary()
```

We can run them in both directions

LSTMs

Simple RNNs use the previous output as input. They have then a very limited feature context.

Long short-term memory units (LSTM) are an extension to RNNs that can remember, possibly forget, information from longer or more distant sequences.

Given an input at index t , \mathbf{x}_t , a LSTM unit produces:

- A short term state, called \mathbf{h}_t and
- A long-term state, called \mathbf{c}_t or memory cell.

The short-term state, \mathbf{h}_t , is the unit output, i.e. \mathbf{y}_t ; but both the long-term and short-term states are reused as inputs to the next unit.

LSTM Equations

A LSTM unit starts from a core equation that is identical to that of a RNN:

$$\mathbf{g}_t = \tanh(\mathbf{W}_g \mathbf{x}_t + \mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{b}_g).$$

From the previous output and current input, we compute three kinds of filters, or gates, that will control how much information is passed through the LSTM cell

The two first gates, \mathbf{i} and \mathbf{f} , defined as:

$$\begin{aligned}\mathbf{i}_t &= \text{activation}(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i), \\ \mathbf{f}_t &= \text{activation}(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f),\end{aligned}$$

model respectively how much we will keep from the base equation and how much we will forget from the long-term state.

LSTM Equations (II)

To implement this selective memory, we apply the two gates to the base equation and to the previous long-term state with the element-wise product (Hadamard product), denoted \circ , and we sum the resulting terms to get the current long-term state:

$$\mathbf{c}_t = \mathbf{i}_t \circ \mathbf{g}_t + \mathbf{f}_t \circ \mathbf{c}_{t-1}.$$

The third gate:

$$\mathbf{o}_t = \text{activation}(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o)$$

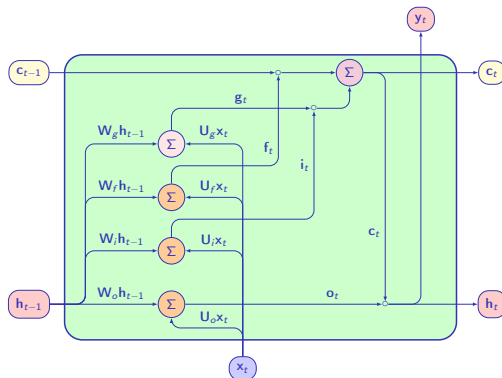
modulates the current long-term state to produce the output:

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t).$$

The LSTM parameters are determined by a gradient descent.
See also:

<https://pytorch.org/docs/stable/nn.html#torch.nn.LSTM>

The LSTM Architecture



An LSTM unit showing the data flow, where g_t is the unit input, i_t , the input gate, f_t , the forget gate, and o_t , the output gate. The activation functions have been omitted

Building a LSTM with Keras

```
model = Sequential()  
model.add(Embedding(max_features, 32))  
model.add(LSTM(32))  
model.add(Dense(1, activation='sigmoid'))  
model.summary()
```

Complete Code Example

Jupyter Notebook: Chollet 6.2 <https://github.com/fchollet/deep-learning-with-python-notebooks/6.2-understanding-recurrent-neural-networks.ipynb>