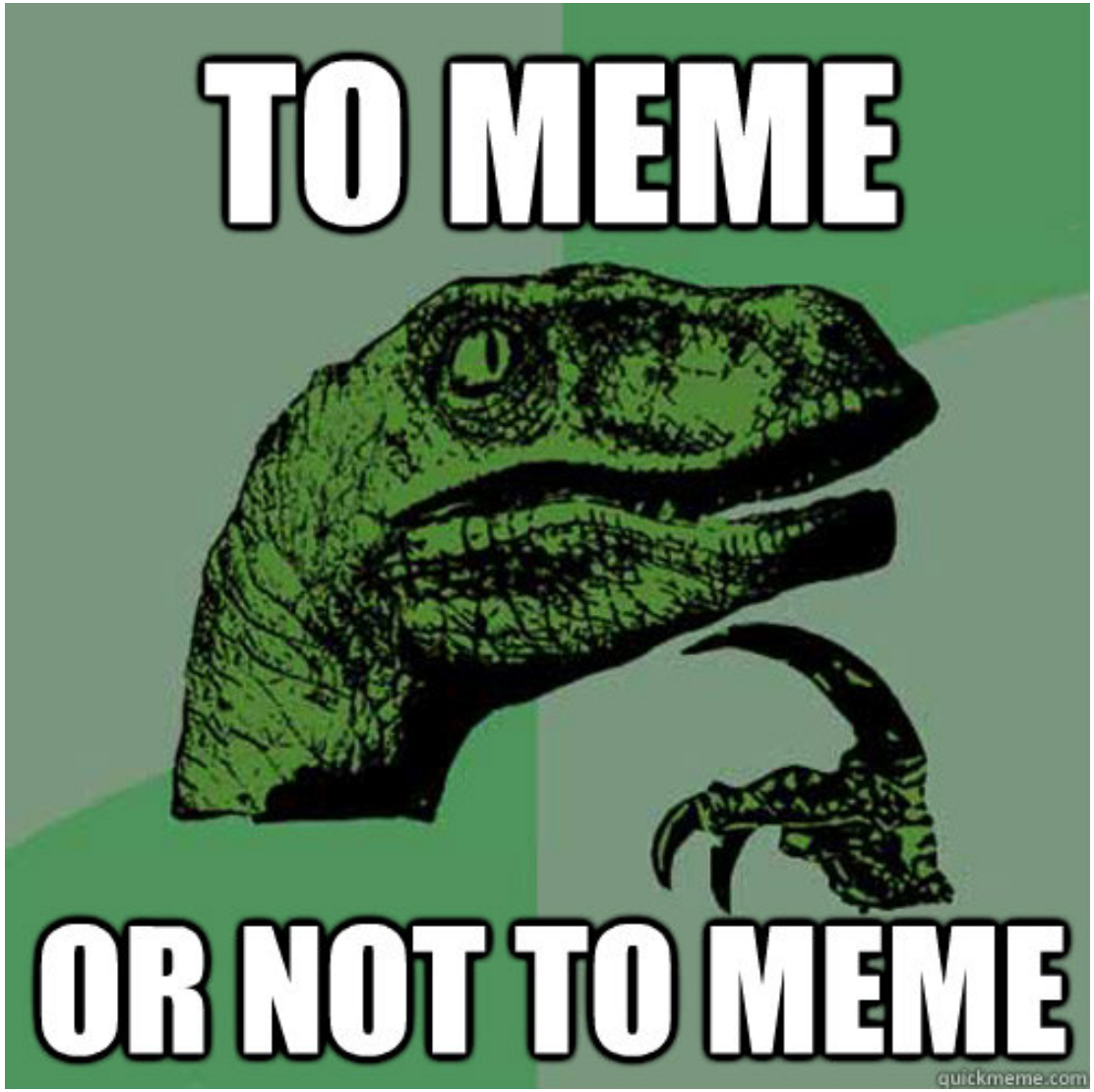
Embeddings and Data representation

Илья Захаркин

Факультет инноваций и высоких технологий
ФПМИ МФТИ

DLSchool, 2018


TO MEME
OR NOT TO MEME
quickmeme.com

Сегодня в программе

1. Представление данных
   - Зачем как-то кодировать данные?
2. Категориальные данные
   - Численное кодирование (LabelEncoding)
   - One-Hot Encoding (оно же DummyEncoding)
   - Умное кодирование
3. Текстовые данные
   - Bag of Words ("Мешок слов") (CountVectorizer)
   - Tf-idf (TfidfVectorizer)
   - Хеширование (HashingVectorizer)
4. Embeddings
   - Общая идея
   - Word2Vec
   - GloVe
   - Собственная нейронная сеть
5. *2Vec

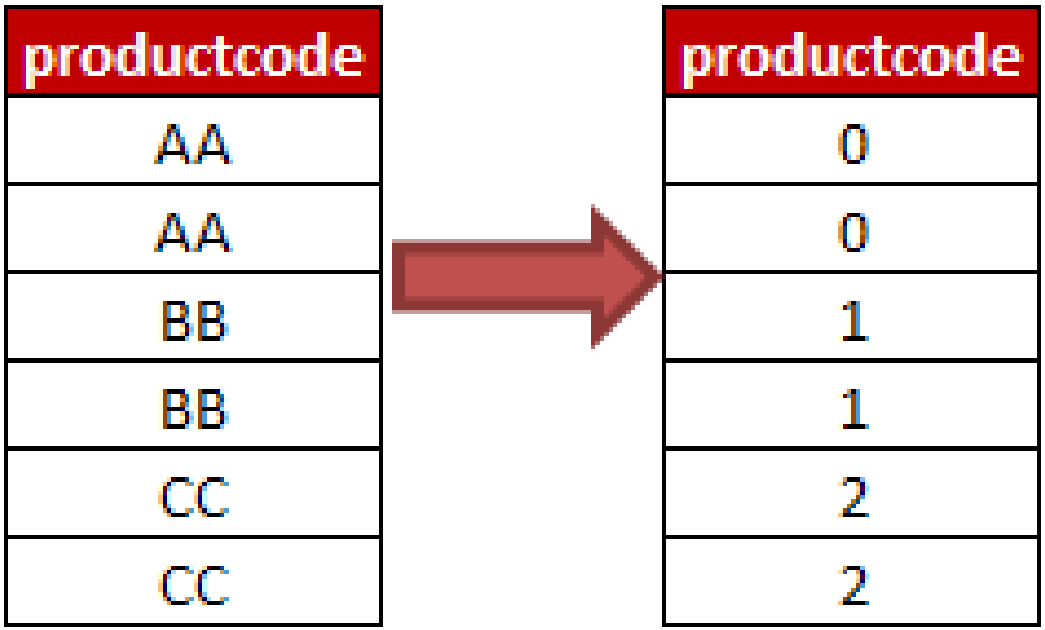
План

1. Представление данных
   - Зачем как-то кодировать данные?
2. Категориальные данные
   - Численное кодирование (LabelEncoding)
   - One-Hot Encoding (оно же DummyEncoding)
   - Умное кодирование
3. Текстовые данные
   - Bag of Words ("Мешок слов") (CountVectorizer)
   - Tf-idf (TfidfVectorizer)
   - Хеширование (HashingVectorizer)
4. Embeddings
   - Общая идея
   - Word2Vec
   - GloVe
   - Собственная нейронная сеть
5. *2Vec

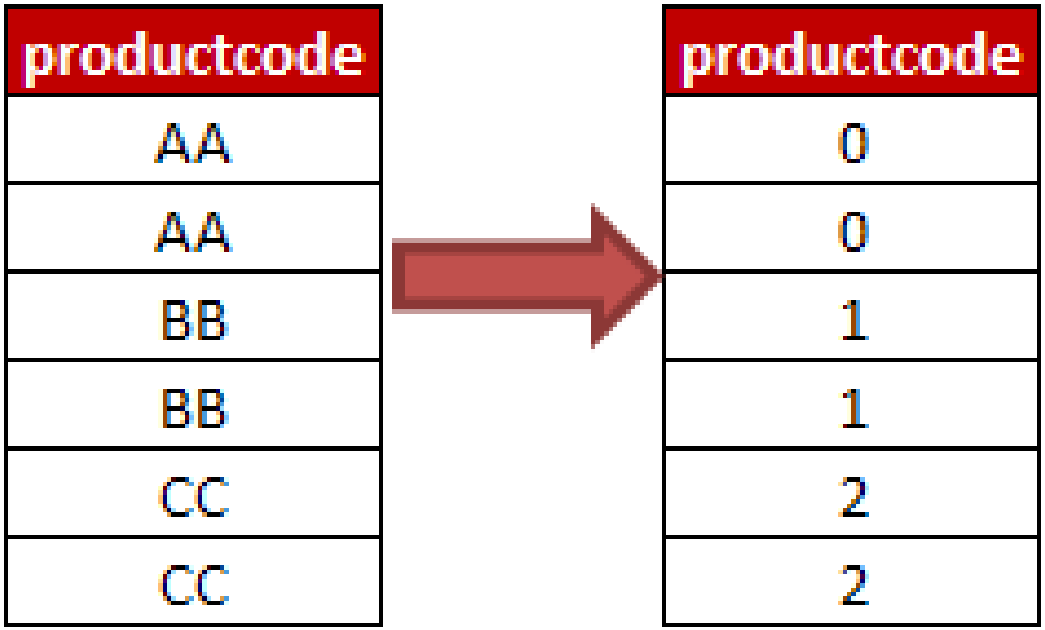
Because...

План

Численное кодирование

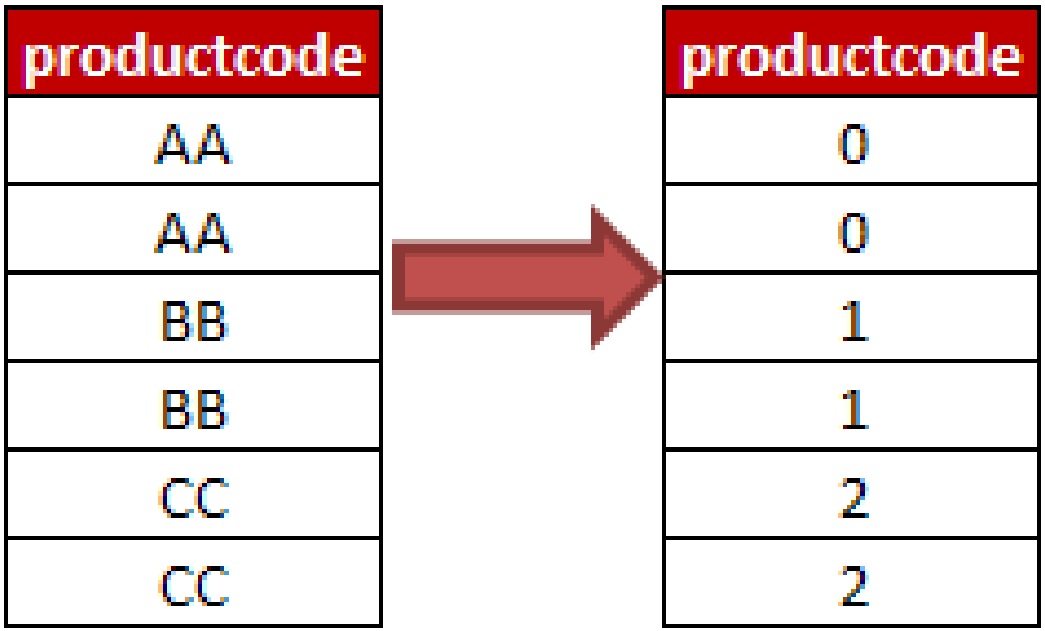


Численное кодирование



Кто видит недостаток?

Численное кодирование



Кто видит недостаток? **Недостаток:** Создаётся отношение порядка на категориальных признаках (а мы обычно не сравниваем, скажем, Москва $>$ Санкт-Петербург)

План

One-Hot (Dummy) кодирование

	A	B	C	D	E	F	G	H	I
1	Original data:			One-hot encoding format:					
2	id	Color		id	White	Red	Black	Purple	Gold
3	1	White		1	1	0	0	0	0
4	2	Red		2	0	1	0	0	0
5	3	Black		3	0	0	1	0	0
6	4	Purple		4	0	0	0	1	0
7	5	Gold		5	0	0	0	0	1
8									

One-Hot (Dummy) кодирование

	A	B	C	D	E	F	G	H	I
1	Original data:			One-hot encoding format:					
2	id	Color		id	White	Red	Black	Purple	Gold
3	1	White		1	1	0	0	0	0
4	2	Red		2	0	1	0	0	0
5	3	Black		3	0	0	1	0	0
6	4	Purple		4	0	0	0	1	0
7	5	Gold		5	0	0	0	0	1
8									

Кто видит недостаток?

One-Hot (Dummy) кодирование

	A	B	C	D	E	F	G	H	I
1	Original data:			One-hot encoding format:					
2	id	Color		id	White	Red	Black	Purple	Gold
3	1	White		1	1	0	0	0	0
4	2	Red		2	0	1	0	0	0
5	3	Black		3	0	0	1	0	0
6	4	Purple		4	0	0	0	1	0
7	5	Gold		5	0	0	0	0	1
8									

Кто видит недостаток? **Недостаток:** Становится очень много признаков (на каждое значение старого категориального признака создаётся по одному столбцу)

План

Target encoding

Пусть, скажем, у вас есть N категорий товара, и целевая переменная - какая-то вещественная.
Идея: закодировать каждую категорию товара его долей целевой переменной. То есть вместо "apple"писать в ячейку число, равное среднему целевой переменной по всей выборке для этой категории (для "apple"), и так для каждой категории.

Smoothed likelihood

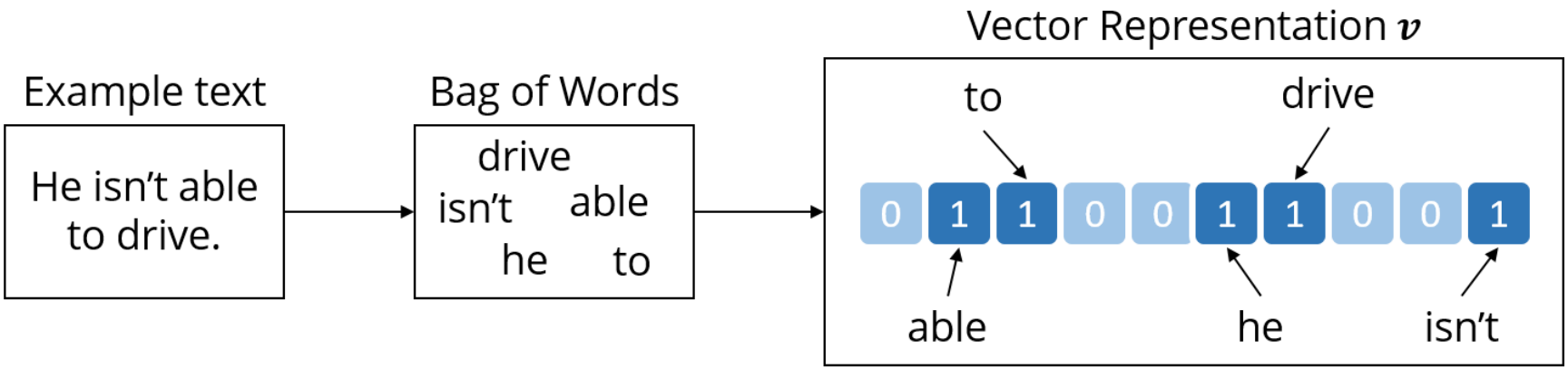
Видео от маэстро Kaggle:
https://www.youtube.com/watch?v=NVKDSNM702k

Tokenization

Сначала текст всегда преобразуется в **токены** - обычно это слова (то есть строка разбивается по пробелам на слова), но иногда задают более сложные правила токенизации (например, "не нравится"превращают в один токен "не_нравится" , чтобы подчеркнуть отрицание, а не просто разбивают на "не"и "нравится")

План

BoW


Example text
He isn't able to drive.
Bag of Words
drive
isn't
able
he
to
Vector Representation v
to
drive
0 1 1 0 0 1 1 0 0 1
able
he
isn't

BoW

```
>>> vectorizer = CountVectorizer()
>>> vectorizer
CountVectorizer(analyzer=...'word', binary=False, decode_error=...'strict',
        dtype=<... 'numpy.int64'>, encoding=...'utf-8', input=...'content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern=...'(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

```
>>> corpus = [
...     'This is the first document.',
...     'This is the second second document.',
...     'And the third one.',
...     'Is this the first document?',
... ]
>>> X = vectorizer.fit_transform(corpus)
>>> X
<4x9 sparse matrix of type '<... 'numpy.int64'>'
    with 19 stored elements in Compressed Sparse ... format>
```

```
>>> vectorizer.get_feature_names() == (
...     ['and', 'document', 'first', 'is', 'one',
...      'second', 'the', 'third', 'this'])
True

>>> X.toarray()
array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 2, 1, 0, 1],
       [1, 0, 0, 0, 1, 0, 1, 1, 0],
       [0, 1, 1, 1, 0, 0, 1, 0, 1]]...)
```

План

Tf-idf

Наблюдение: В больших корпусах текстов могут часто встречаться общие слова (такие как “the”, “a”, “is” в английском), несущие очень мало информации о самом тексте. Если мы сделаем "мешок слов то получим очень большие значения счётчиков у этих слов, что модель потом учтёт, а мы этого не хотим, т.к. есть более значимые для текстов слова, но встречающиеся реже.

Tf-idf

Наблюдение: В больших корпусах текстов могут часто встречаться общие слова (такие как “the”, “a”, “is” в английском), несущие очень мало информации о самом тексте. Если мы сделаем "мешок слов то получим очень большие значения счётчиков у этих слов, что модель потом учтёт, а мы этого не хотим, т.к. есть более значимые для текстов слова, но встречающиеся реже.

Решение: Перевзвесим слова в соответствие с частотой их появления во всех текстах, таким образом более общие слова будут иметь меньшую значимость (так как у них будет вес, меньший 1).

Tf-idf

Расшифровка Tf-idf:
term-frequency умножить на **inverse document-frequency**

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

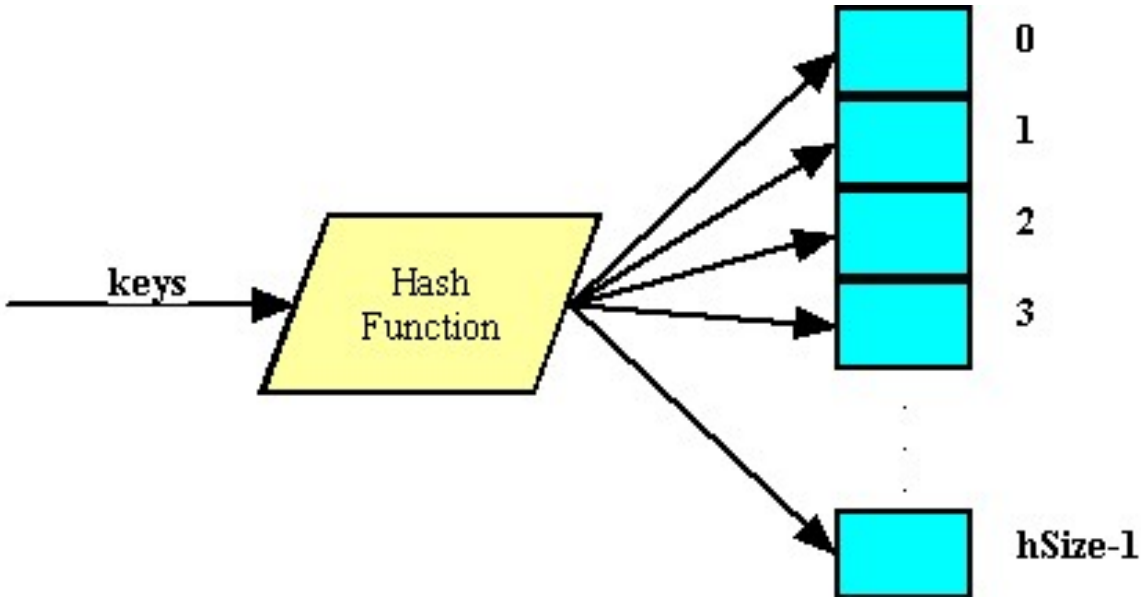
$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

План

Hashing




```
>>> hv = HashingVectorizer()
>>> hv.transform(corpus)
...
<4x1048576 sparse matrix of type '<... 'numpy.float64'>'
    with 19 stored elements in Compressed Sparse ... format>
```

Hashing

Плюсы:

- требует мало памяти, легко расширяется на большие датасеты - не надо хранить словарь
- может быть использован в режиме онлайн-кодирования или параллельно

Минусы:

- нельзя вычислить обратное преобразование хеш->слово, что мешает интерпретируемости
- иногда случаются **коллизии** (совпадение хешей разных слов) (однако на самом деле современные реализации свели вероятность подобного к минимуму)

План

Embeddings

Embedding - это функция, переводящая объекты (в нашем случае - слова) в векторное (числовое) пространство некоторой размерности (обычно довольно высокой, d = 200,500)
Эмбеддингами также часто называют просто сами представления слов в векторном (числовом) пространстве.

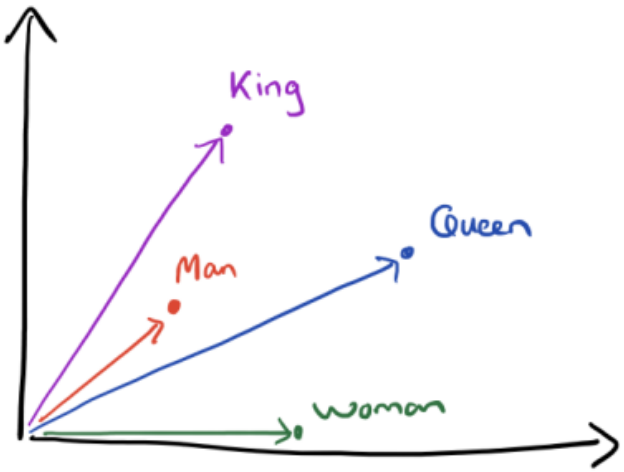
$$W(\text{“cat”}) = (0.2,\ \text{-}0.4,\ 0.7,\ \ldots)$$

$$W(\text{“mat”}) = (0.0,\ 0.6,\ \text{-}0.1,\ \ldots)$$

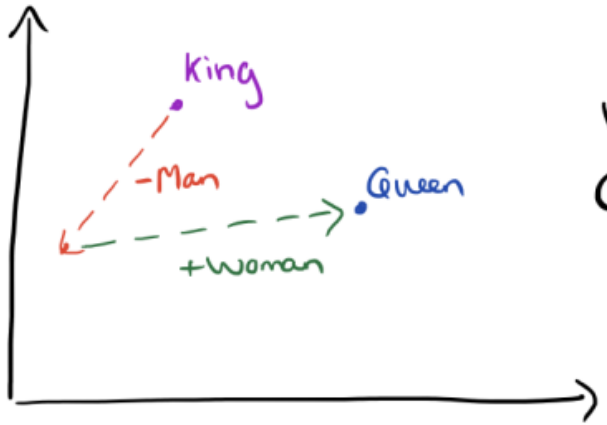
Embeddings

Идея: Мы хотим не просто перевести слова в числовые векторы, а сделать так, чтобы похожие слова были "близко а, скажем, антонимы были противоположны по знаку.

Embeddings



Word Vectors



Vector Composition

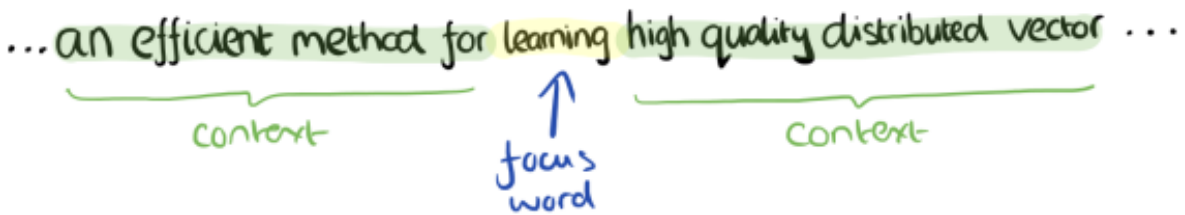
Distributive hypothesis

"Лингвистические единицы, встречающиеся в схожих контекстах, имеют близкие значения"

План

Word2Vec

Word2Vec использует информацию о контексте и тренирует нейросеть, чтобы различать группы слов, которые встречаются вместе, и группы слов, подобранные случайно.
На вход подаётся разреженное представление целевого слова и несколько слов из контекста (тоже в разреженном виде). Этот входной слой связан с одним скрытым слоем, который уже связан с выходным слоем (то есть в сети всего 3 слоя, считая вход и выход).



Word2Vec

- В одной версии алгоритма мы сэмплируем отрицательные примеры подменяя целевое слово случайным. Например: если есть позитивный пример "the plane flies можно подменить plane на jogging, получим негативный пример "the jogging flies".
- В другой версии мы берём пары (целевое слово, контекстное слово) и (для негативных примеров) подсовывем случайно выбранное слово вместо контекстного: (the, plane), (flies, plane) -> (compiled, plane), (who, plane). Алгоритм тренируется различать, какие пары действительно могут встречаться вместе, а какие нет.

Сам полученный классификатор не так интересен, как **веса скрытого слоя**, которые, по сути, и являтся embedding'ами входых слов (то есть веса, стоящие на связях входного слоя и скрытого слоя).

Word2Vec

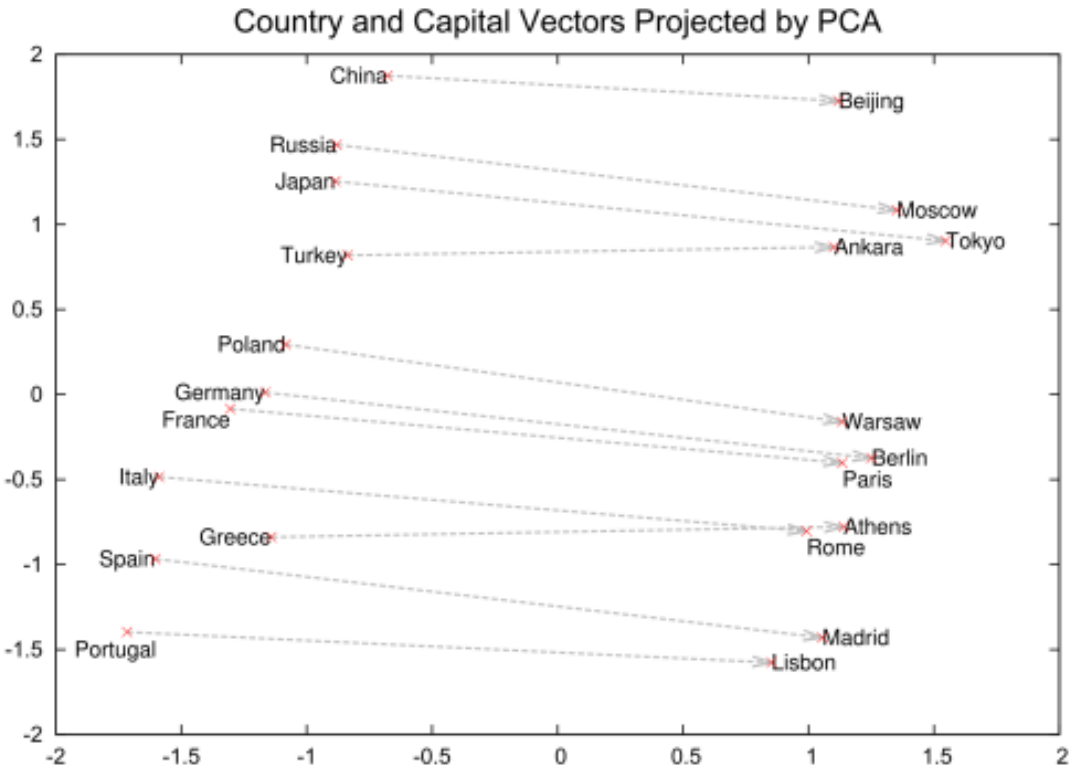
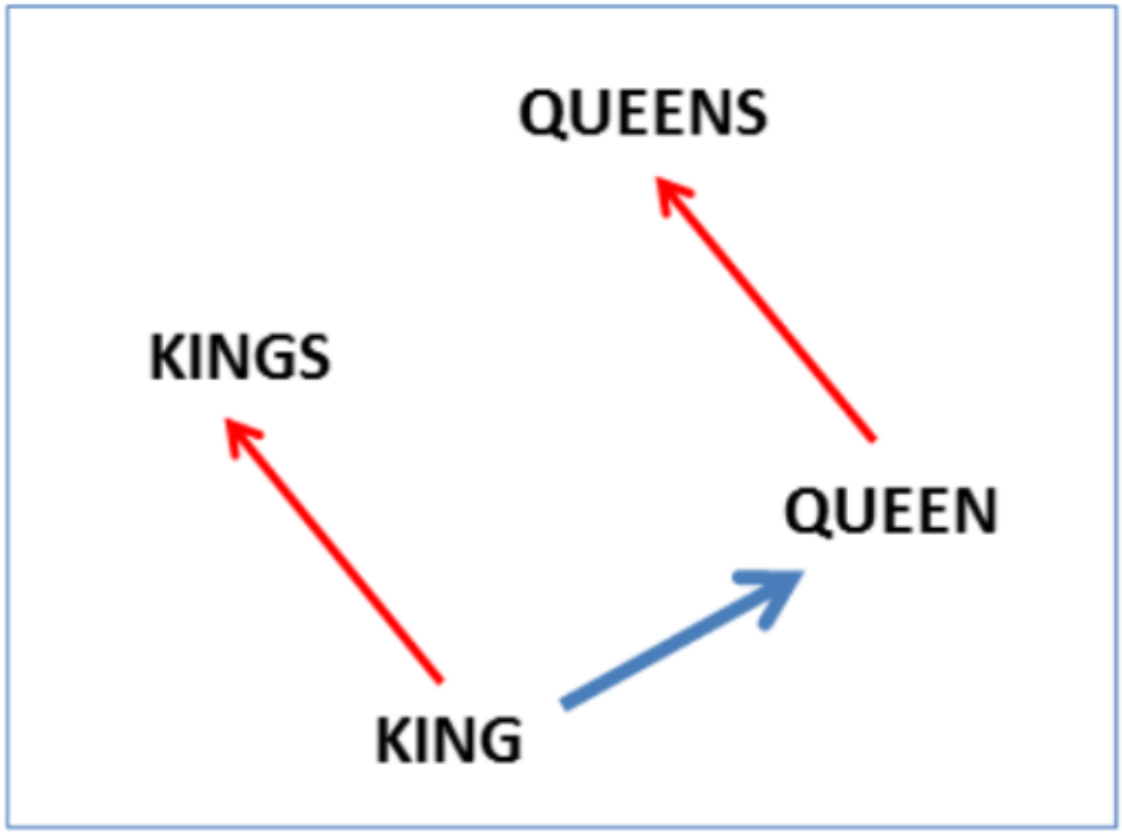


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

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

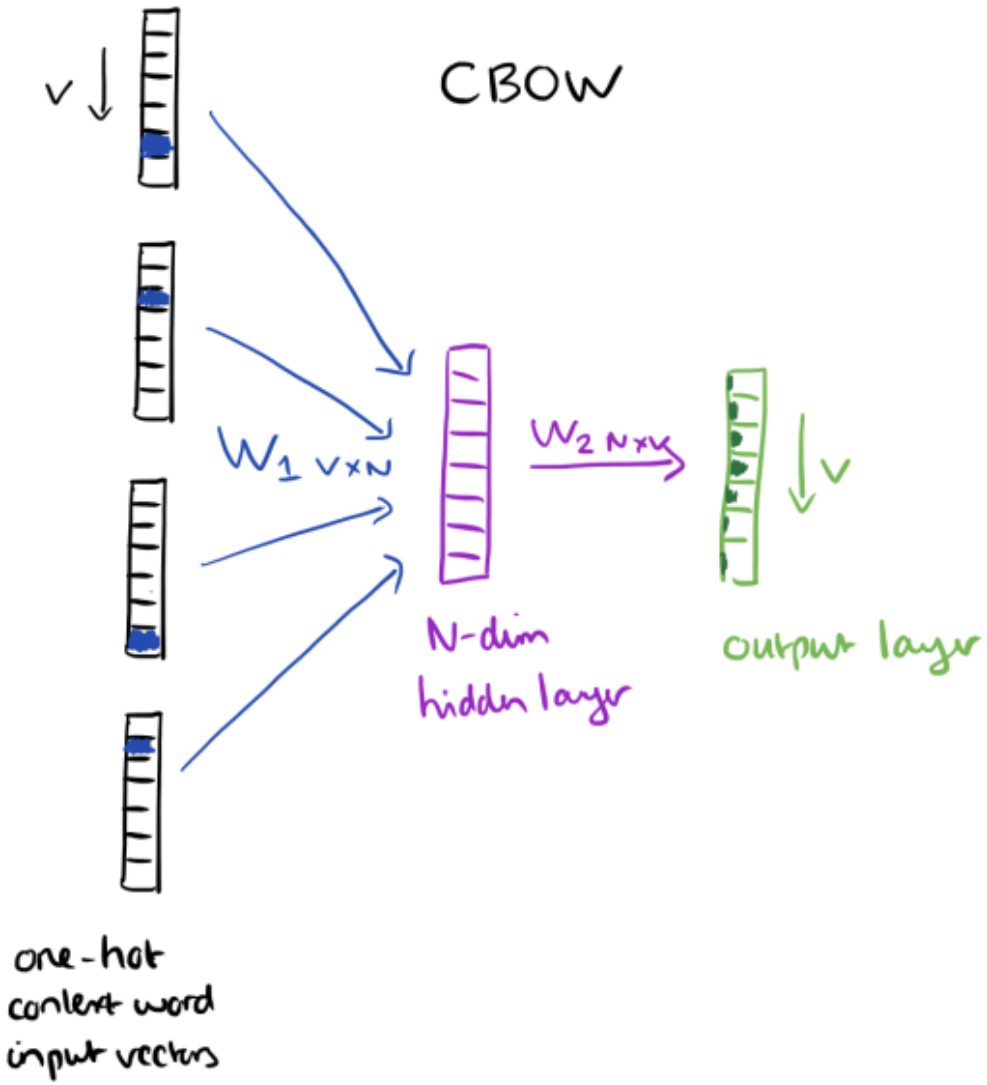
Word2Vec



Реализации Word2Vec

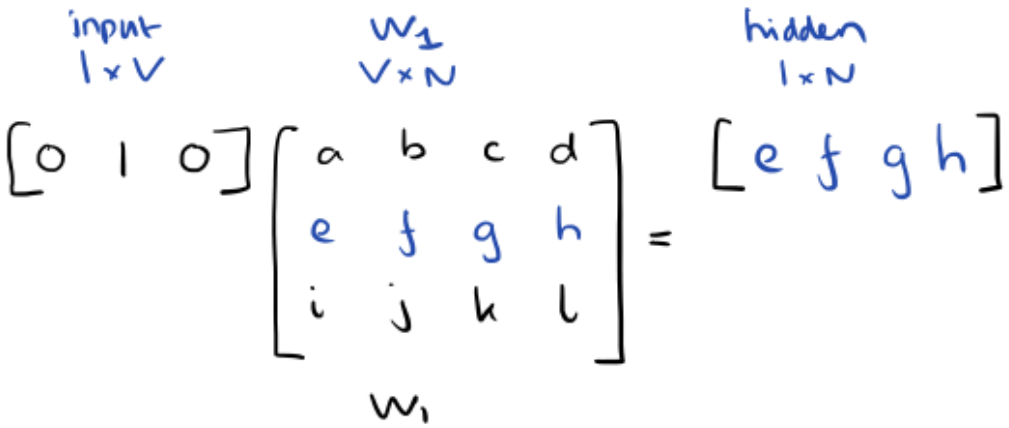
- Continuous Bag-of-Words (CBoW)
- Continuous Skip-gram

CBoW


CBOW
V
W_1 V×N
W_2 N×V
V
N-dim
hidden layer
output layer
one-hot
context word
input vectors

CBoW

Линейные активации



CBoW

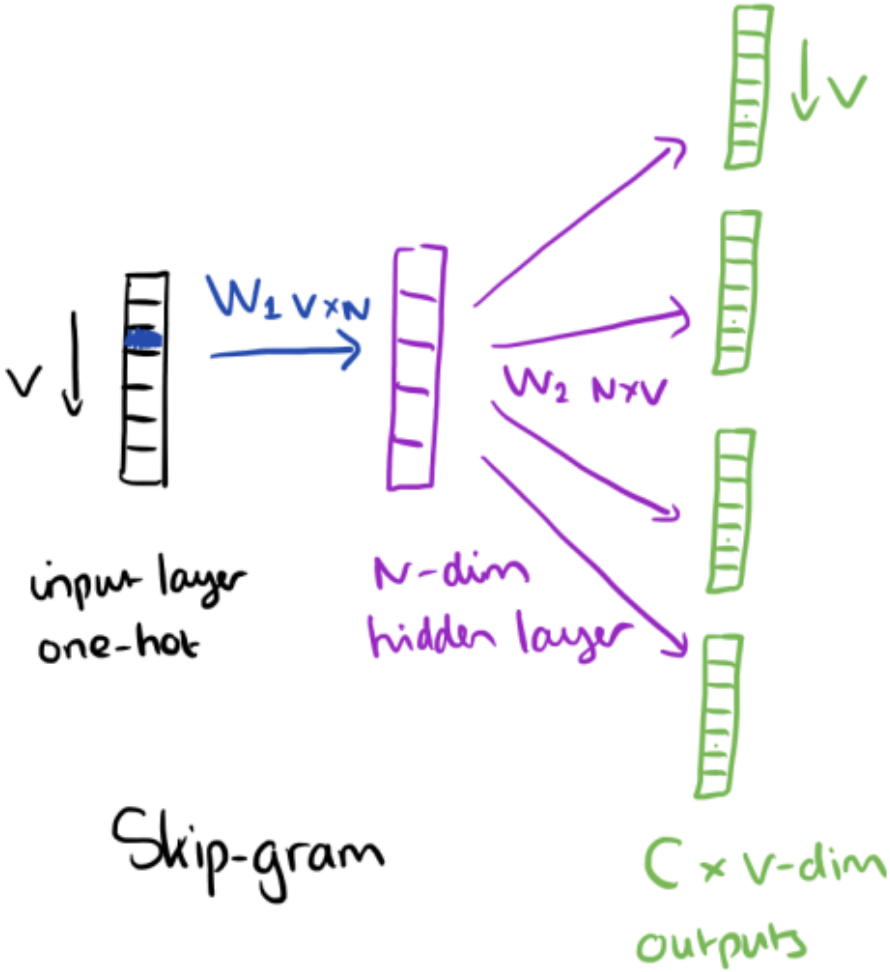
Оптимизируемый loss:

The CBOW model is as follows. Given a target word w_i and an N context window on each side, $w_{i-1}, \ldots, w_{i-N}$ and $w_{i+1}, \ldots, w_{i+N}$, referring to all context words collectively as C, CBOW tr to minimize

$$-\log p(w_i|C) = -\log \text{Softmax}(A(\sum_{w \in C} q_w) + b)$$

where q_w is the embedding for word w.

Skip-gram


V
W1 V×N
W2 N×V
V
input layer
one-hot
N-dim
hidden layer
Skip-gram
C × V-dim
outputs

План

GloVe

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning

Introduction

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Getting started (Code download)

- Download the code (licensed under the Apache License, Version 2.0)
- Unpack the files: unzip GloVe-1.2.zip
- Compile the source: cd GloVe-1.2 && make
- Run the demo script: ./demo.sh
- Consult the included README for further usage details, or ask a question
- The code is also available on GitHub

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data

Citing GloVe

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. [pdf] [bib]

План

1. Представление данных
   - Зачем как-то кодировать данные?
2. Категориальные данные
   - Численное кодирование (LabelEncoding)
   - One-Hot Encoding (оно же DummyEncoding)
   - Умное кодирование
3. Текстовые данные
   - Bag of Words ("Мешок слов") (CountVectorizer)
   - Tf-idf (TfidfVectorizer)
   - Хеширование (HashingVectorizer)
4. Embeddings
   - Общая идея
   - Word2Vec
   - GloVe
   - Собственная нейронная сеть
5. *2Vec

Embedding layer

Embedding [source]

```
keras.layers.Embedding(input_dim, output_dim, embeddings_initializer='uniform', embeddings_regularizer=None, act
```

Turns positive integers (indexes) into dense vectors of fixed size. eg. [[4], [20]] -> [[0.25, 0.1], [0.6, -0.2]]

This layer can only be used as the first layer in a model.

Example

```
model = Sequential()
model.add(Embedding(1000, 64, input_length=10))
# the model will take as input an integer matrix of size (batch, input_length).
# the largest integer (i.e. word index) in the input should be no larger than 999 (vocabulary size).
# now model.output_shape == (None, 10, 64), where None is the batch dimension.

input_array = np.random.randint(1000, size=(32, 10))

model.compile('rmsprop', 'mse')
output_array = model.predict(input_array)
assert output_array.shape == (32, 10, 64)
```

Arguments

- **input_dim:** int > 0. Size of the vocabulary, i.e. maximum integer index + 1.
- **output_dim:** int >= 0. Dimension of the dense embedding.
- **embeddings_initializer:** Initializer for the `embeddings` matrix (see initializers).
- **embeddings_regularizer:** Regularizer function applied to the `embeddings` matrix (see regularizer).
- **embeddings_constraint:** Constraint function applied to the `embeddings` matrix (see constraints).
- **mask_zero:** Whether or not the input value 0 is a special "padding" value that should be masked out. This is useful when using recurrent layers which may take variable length input. If this is `True` then all subsequent layers in the model need to support masking or an exception will be raised. If mask_zero is set to True, as a consequence, index 0 cannot be used in the vocabulary (input_dim should equal size of vocabulary + 1).
- **input_length:** Length of input sequences, when it is constant. This argument is required if you are going to connect `Flatten`

Embedding layer

```
# Author: Robert Guthrie

import torch
import torch.autograd as autograd
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

torch.manual_seed(1)
```

```
word_to_ix = {"hello": 0, "world": 1}
embeds = nn.Embedding(2, 5)  # 2 words in vocab, 5 dimensional embeddings
lookup_tensor = torch.LongTensor([word_to_ix["hello"]])
hello_embed = embeds(autograd.Variable(lookup_tensor))
print(hello_embed)
```

Out:

```
Variable containing:
 0.6614  0.2669  0.0617  0.6213 -0.4519
[torch.FloatTensor of size 1x5]
```

Our own NN

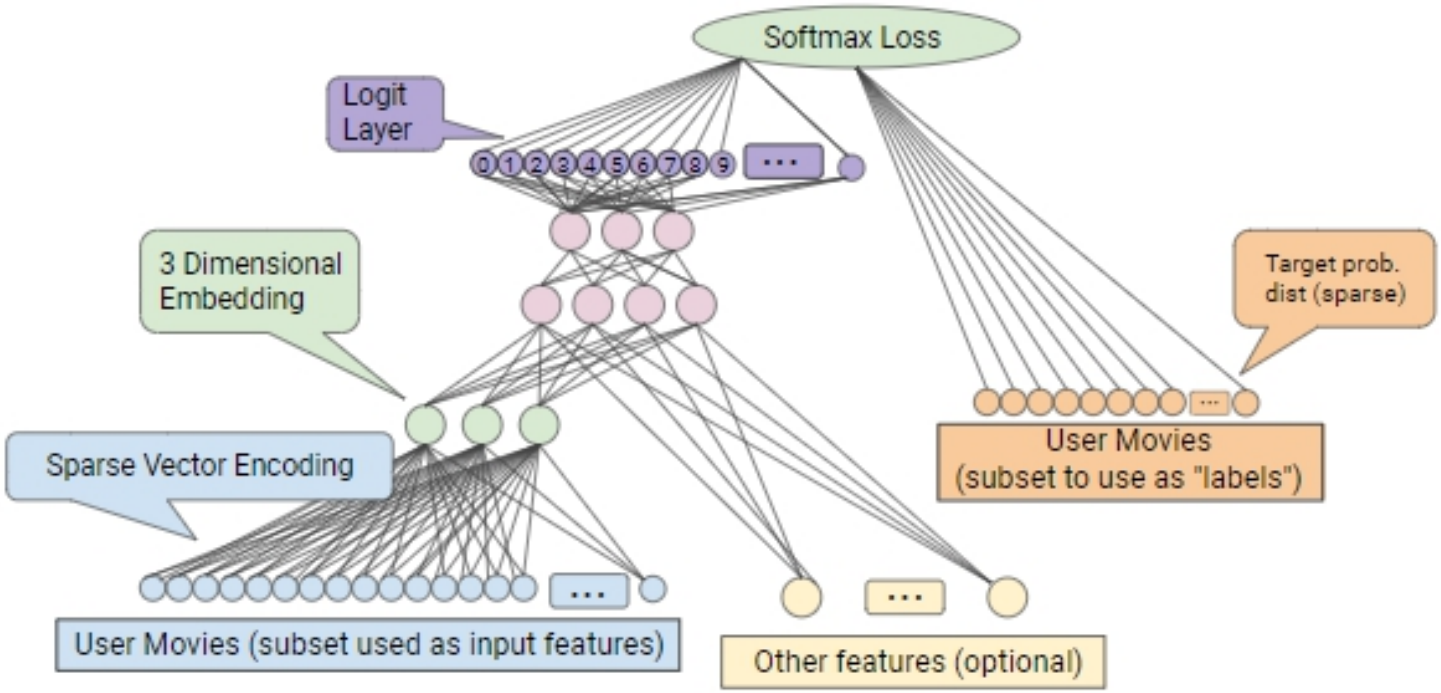


Figure 5. A sample DNN architecture for learning movie embeddings from collaborative filtering data.

*2Vec

https://gist.proxy.oonnnoo.com/nzw0301/333afc00bd508501268fa7bf40cafe4e

*2vec papers

1. word2vec https://arxiv.org/abs/1310.4546
2. sentence2vec, paragraph2vec, doc2vec http://arxiv.org/abs/1405.4053
3. tweet2vec http://arxiv.org/abs/1605.03481
4. tweet2vec https://arxiv.org/abs/1607.07514
5. author2vec http://dl.acm.org/citation.cfm?id=2889382
6. item2vec http://arxiv.org/abs/1603.04259
7. lda2vec https://arxiv.org/abs/1605.02019
8. illustration2vec http://dl.acm.org/citation.cfm?id=2820907
9. tag2vec http://ktsaurabh.weebly.com/uploads/3/1/7/8/31783965/distributed_representations_for_content-based_and_personalized_tag_recommendation.pdf
10. category2vec http://www.anlp.jp/proceedings/annual_meeting/2015/pdf_dir/C4-3.pdf
11. topic2vec http://arxiv.org/abs/1506.08422
12. image2vec http://arxiv.org/abs/1507.08818
13. app2vec http://paul.rutgers.edu/~qma/research/ma_app2vec.pdf
14. prod2vec http://dl.acm.org/citation.cfm?id=2788627
15. meta-prod2vec http://arxiv.org/abs/1607.07326

*2Vec

16. sense2vec http://arxiv.org/abs/1511.06388
17. node2vec http://www.kdd.org/kdd2016/papers/files/Paper_218.pdf
18. subgraph2vec http://arxiv.org/abs/1606.08928
19. wordnet2vec http://arxiv.org/abs/1606.03335
20. doc2sent2vec http://research.microsoft.com/apps/pubs/default.aspx?id=264430
21. context2vec http://u.cs.biu.ac.il/~melamuo/publications/context2vec_conll16.pdf
22. rdf2vec http://iswc2016.semanticweb.org/pages/program/accepted-papers.html#research_ristoski_32
23. hash2vec http://arxiv.org/abs/1608.08940
24. query2vec http://www.cs.cmu.edu/~dongyeok/papers/query2vec_v0.2.pdf
25. gov2vec http://arxiv.org/abs/1609.06616
26. novel2vec http://aics2016.ucd.ie/papers/full/AICS_2016_paper_48.pdf
27. emoji2vec http://arxiv.org/abs/1609.08359
28. video2vec https://staff.fnwi.uva.nl/t.e.j.mensink/publications/habibian16pami.pdf
29. video2vec http://www.public.asu.edu/~bli24/Papers/ICPR2016_video2vec.pdf
30. sen2vec https://arxiv.org/abs/1610.08078

*2Vec

31. content2vec http://104.155.136.4:3000/forum?id=ryTYxh5ll
32. cat2vec http://104.155.136.4:3000/forum?id=HyNxRZ9xg
33. diet2vec https://arxiv.org/abs/1612.00388
34. mention2vec https://arxiv.org/abs/1612.02706
35. POI2vec http://www.ntu.edu.sg/home/boan/papers/AAAI17_Visitor.pdf
36. wang2vec http://www.cs.cmu.edu/~lingwang/papers/naacl2015.pdf
37. dna2vec https://arxiv.org/abs/1701.06279
38. pin2vec https://labs.pinterest.com/assets/paper/p2p-www17.pdf, (cited blog)
39. paper2vec https://arxiv.org/abs/1703.06587
40. struc2vec https://arxiv.org/abs/1704.03165
41. med2vec http://www.kdd.org/kdd2016/papers/files/rpp0303-choiA.pdf
42. net2vec https://arxiv.org/abs/1705.03881
43. sub2vec https://arxiv.org/abs/1702.06921
44. metapath2vec https://ericdongyx.github.io/papers/KDD17-dong-chawla-swami-metapath2vec.pdf
45. concept2vec
    http://knoesis.cs.wright.edu/sites/default/files/Concept2vec__Evaluating_Quality_of_Embeddings_for_OntologicalConcepts%20%284%29.pdf

*2Vec

46. graph2vec http://arxiv.org/abs/1707.05005
47. doctag2vec https://arxiv.org/abs/1707.04596
48. skill2vec https://arxiv.org/abs/1707.09751
49. style2vec https://arxiv.org/abs/1708.04014
50. ngram2vec http://www.aclweb.org/anthology/D17-1023
51. hin2vec https://dl.acm.org/citation.cfm?id=3132953
52. edge2vec http://www.cys.cic.ipn.mx/ojs/index.php/CyS/article/view/2849/2380
53. place2vec https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/225625/41_gao.pdf?sequence=3
54. hyperedge2vec https://openreview.net/forum?id=rJ5C67-C-
55. mvn2vec https://arxiv.org/abs/1801.06597
56. onto2vec https://arxiv.org/abs/1802.00864

*2Vec

nzw0301 commented on 13 May 2016 • edited ▾ Owner

I cannot find some papers

- cv2vec https://medium.com/talla-inc/introducing-cv2vec-a-neural-model-for-candidate-similarity-e215b1b12472
- market2vec https://medium.com/@TalPerry/deep-learning-the-stock-market-df853d139e02#.akc6af77u
- food2vec https://altosaar.github.io/food2vec/
- entity2vec https://github.com/MultimediaSemantics/entity2vec

Must have for NLP

- sklearn (vectorizers, basic stuff)
- re (for regular expressions)
- gensim (Word2Vec, Doc2Vec)
- nltk
- pymorphy2 (с русским текстом)


IT'S OVER
IT'S DONE
quickmeme.com