Neural Methods for NLP

Course 3: embeddings

Master LiTL --- 2021-2022

chloe.braud@irit.fr

Content

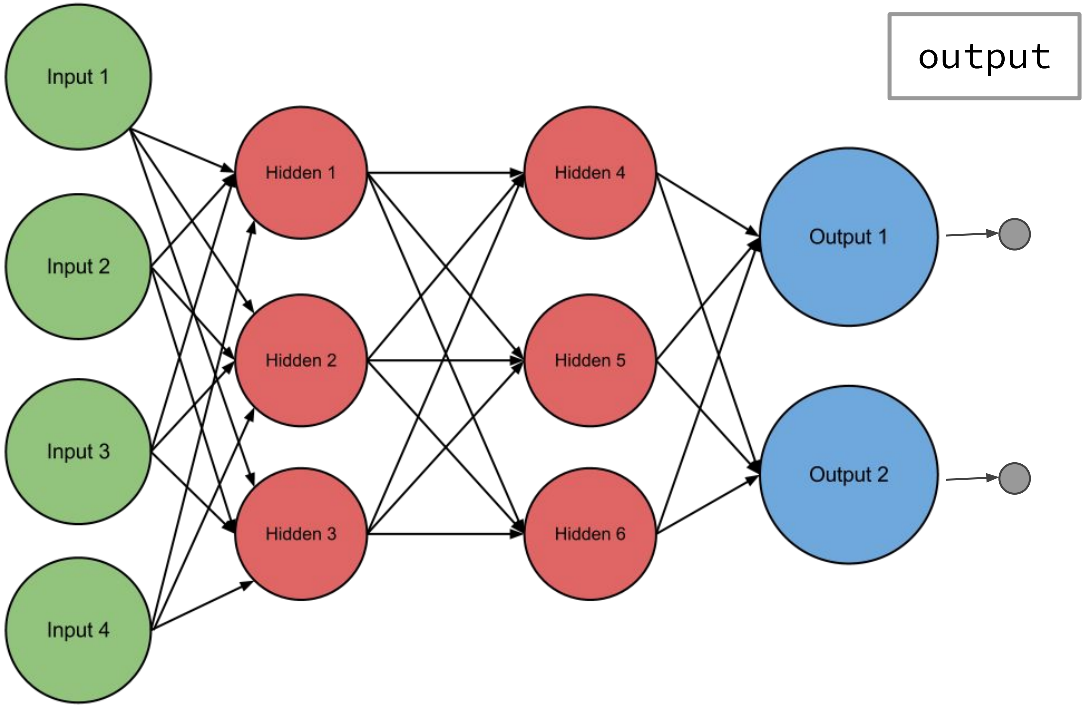
Cours 3 – embeddings

1. Why dense representations?
2. Word embeddings with DL
3. Pre-trained word embeddings
4. Feeding the network
5. Word embeddings in practice

The input layer

input

output



input layer

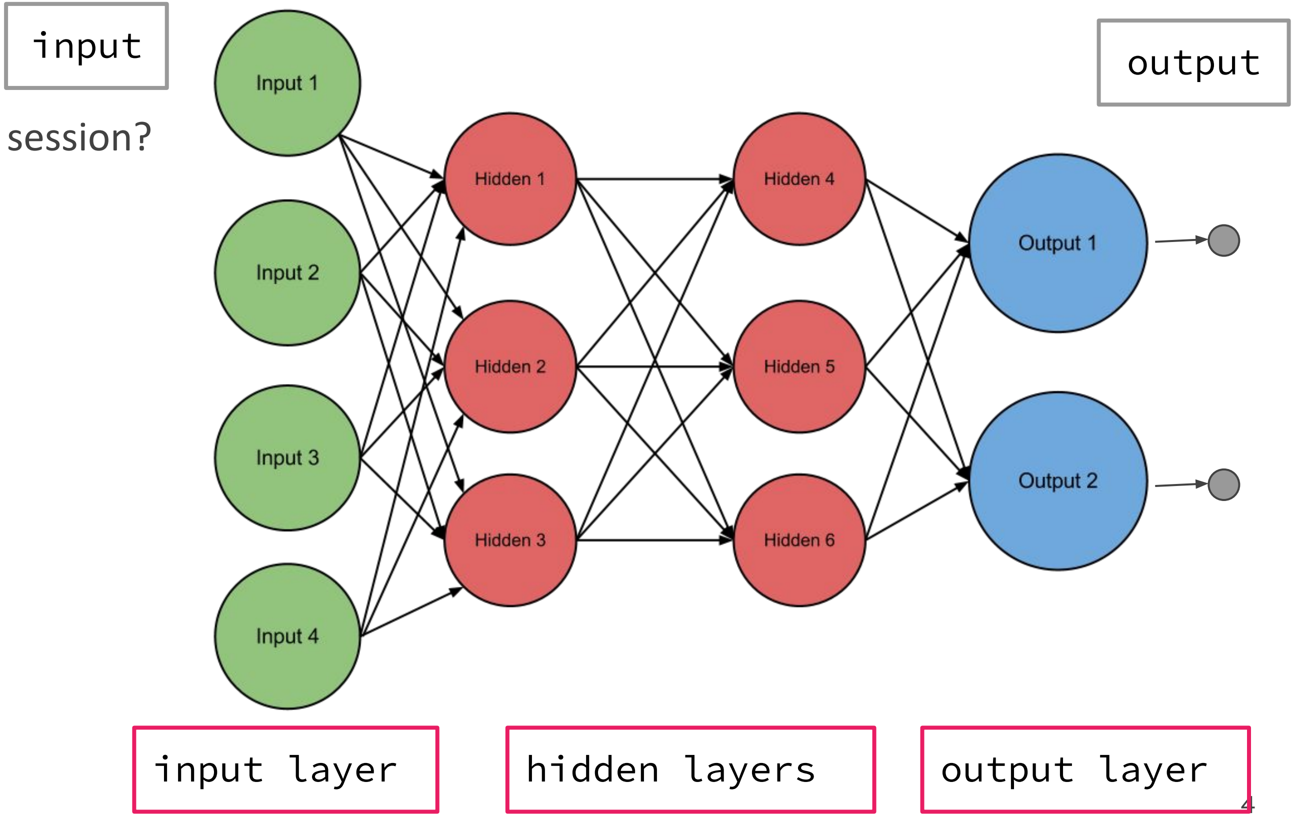
hidden layers

output layer

The input layer

What did we do during the last session?

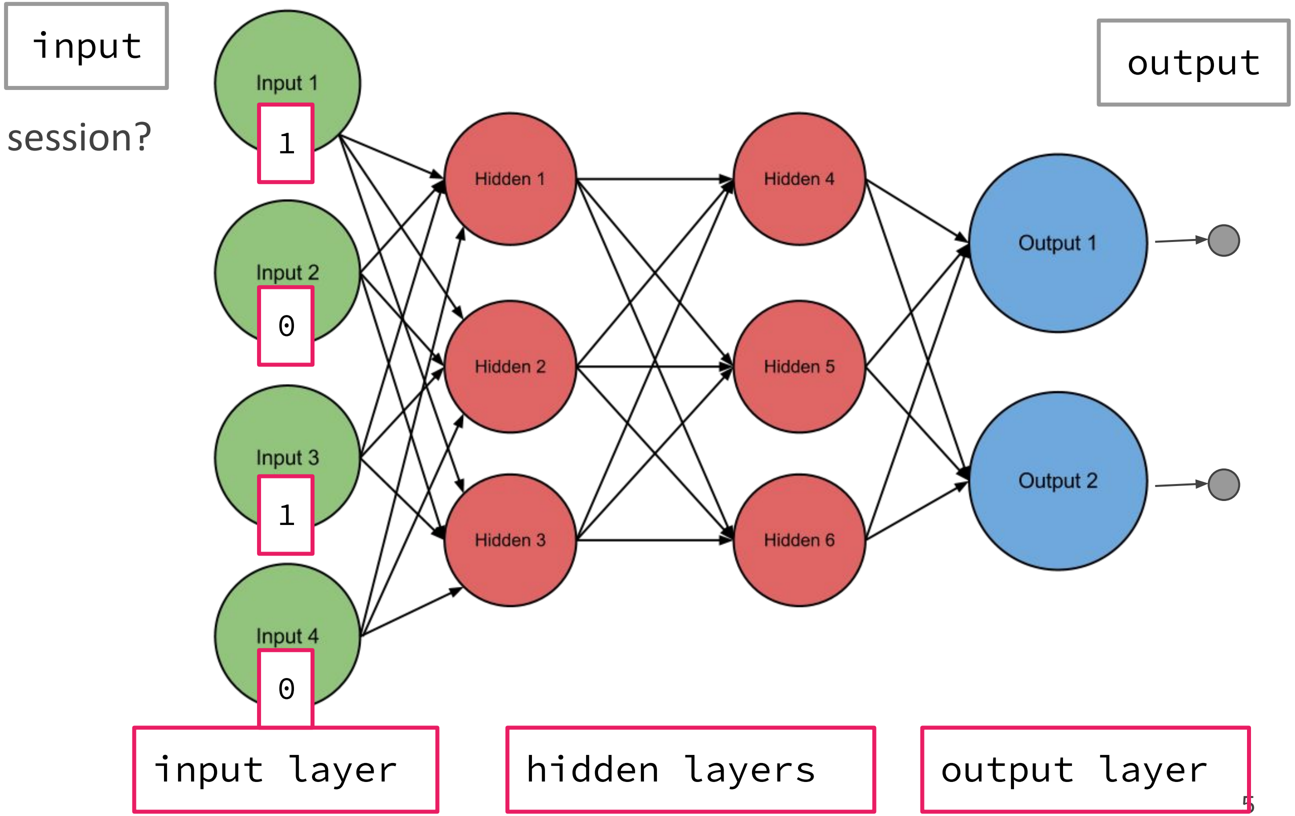
How was built the input?



The input layer

What did we do during the last session?

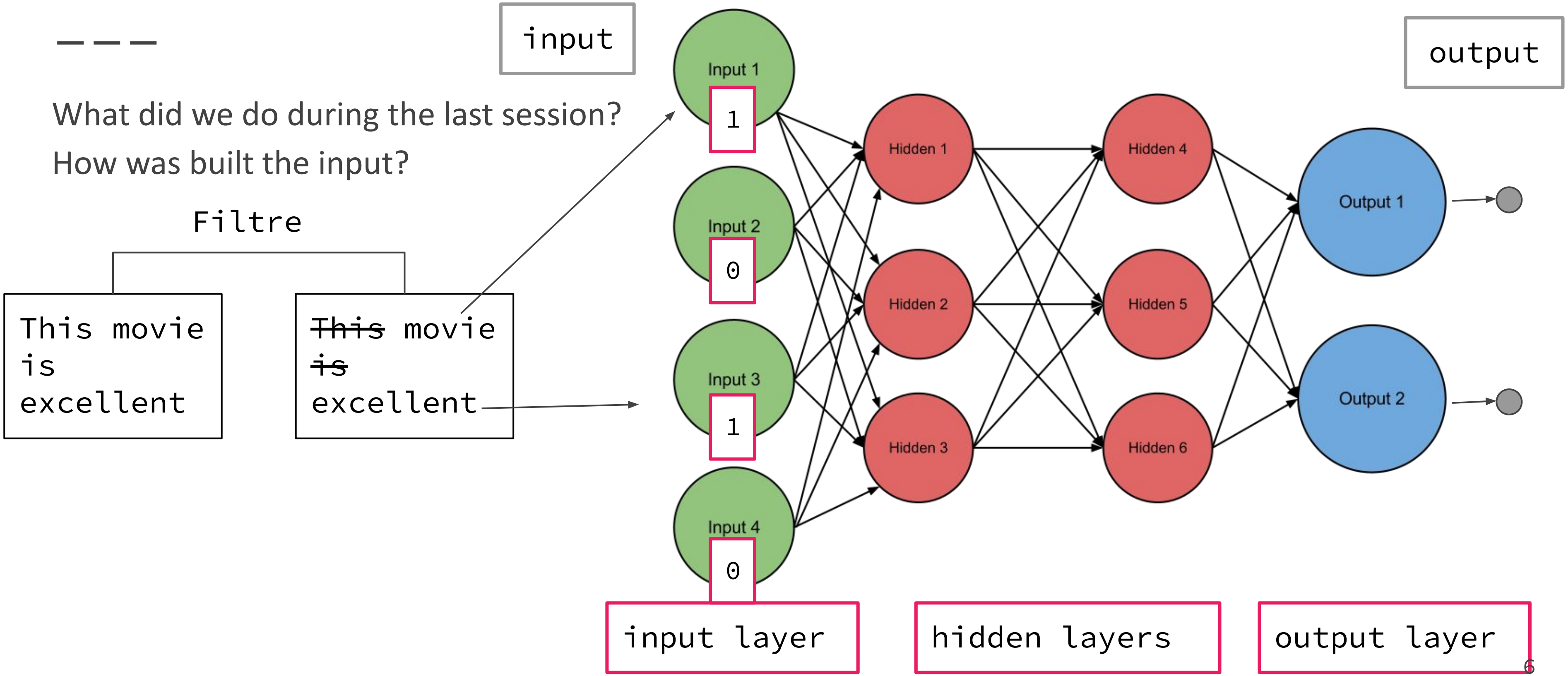
How was built the input?



The input layer

What did we do during the last session?

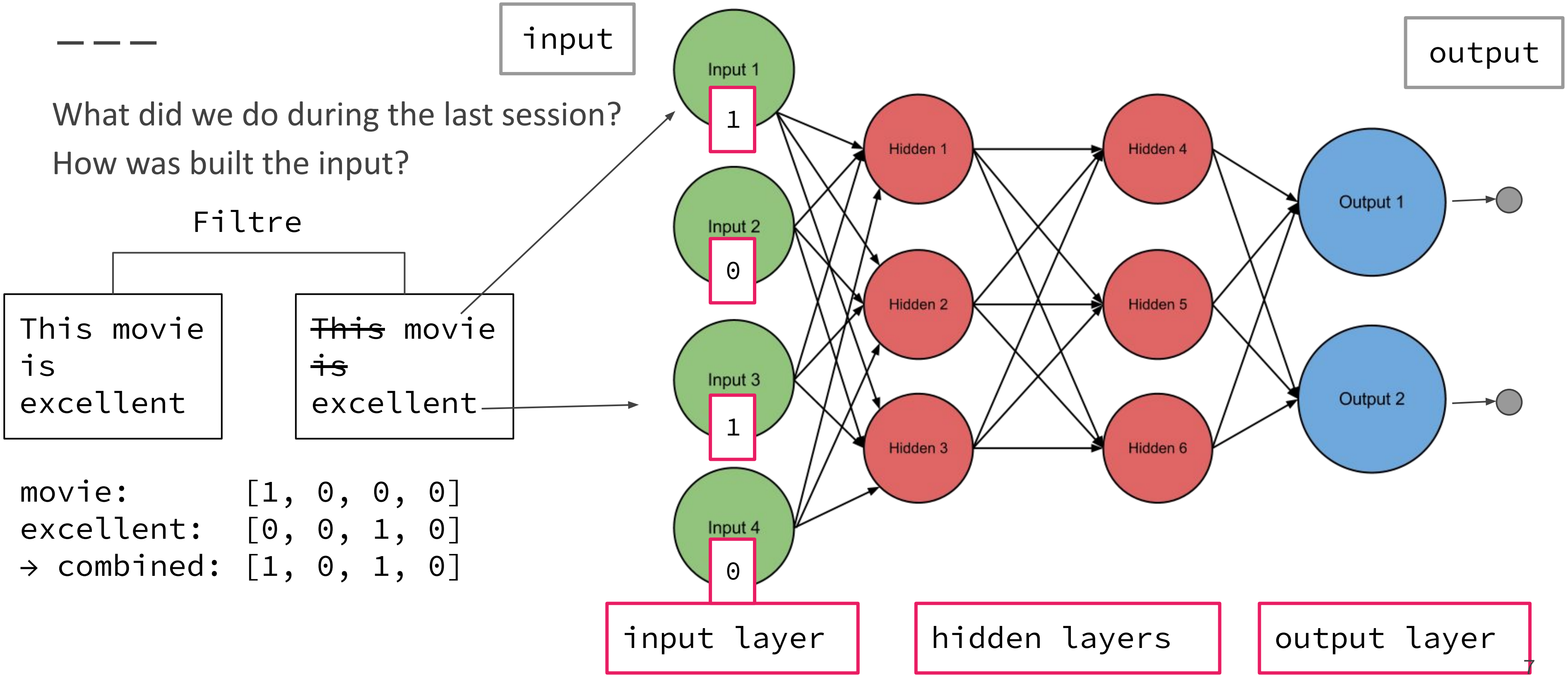
How was built the input?



The input layer

What did we do during the last session?

How was built the input?



Filtre

This movie is excellent

~~This~~ movie ~~is~~ excellent

```
movie:       [1, 0, 0, 0]
excellent:   [0, 0, 1, 0]
→ combined: [1, 0, 1, 0]
```

The input layer

What did we do during the last session?

How was built the input?

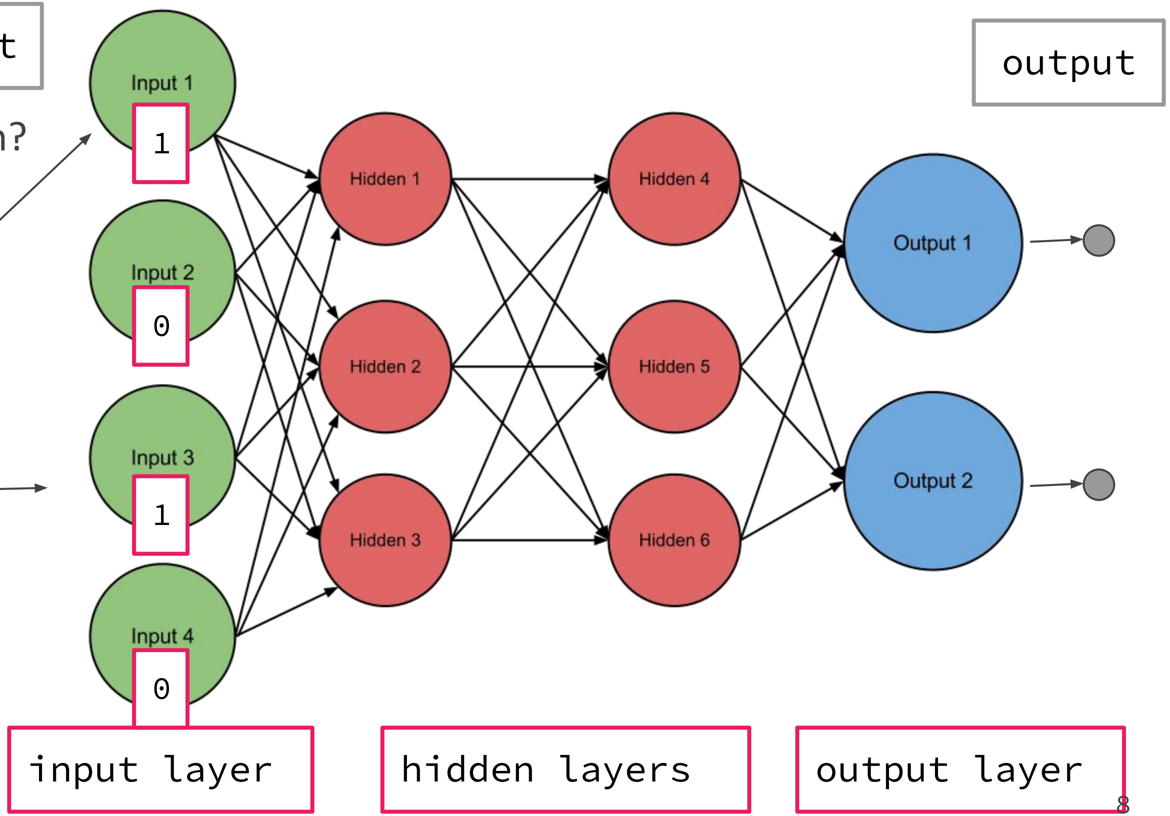
Filtre

This movie is excellent

~~This~~ movie ~~is~~ excellent

```
movie:     [1, 0, 0, 0]
excellent: [0, 0, 1, 0]
→ combined: [1, 0, 1, 0]
```

→ usual way: dense vectors



Standard vs neural approach

Standard approach:

- linear model trained over **high-dimensional but very sparse feature vectors**
- requires to manually specify the important features

Neural approach:

- non-linear neural networks **over dense input vectors**
- automatically induce important features

Standard vs neural approach

Standard approach:

- linear model trained over **high-dimensional but very sparse feature vectors**
- requires to **manually specify the important features**

Neural approach:

- non-linear neural networks **over dense input vectors**
- **automatically induce important features**

Feature representation

One-hot *vs* Dense

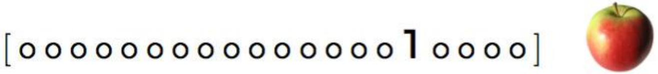
- **One-hot**: each feature is its own dimension
    - Dimensionality is same as number of features
    - Each feature is completely independent from one another
- **Dense**: each feature is a d-dimensional vector
    - Dimensionality is *d*
    - Similar features have similar vectors

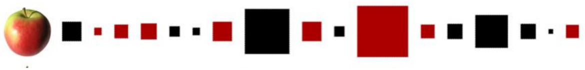
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

Feature representation

One-hot *vs* Dense

- **One-hot**: each feature is its own dimension
  - Dimensionality is same as number of features
  - Each feature is completely independent from one another
- **Dense**: each feature is a d-dimensional vector
  - Dimensionality is *d*
  - Similar features have similar vectors



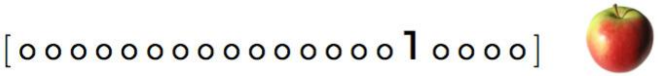


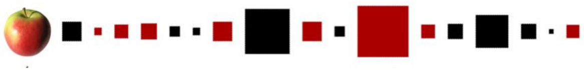
```
e.g. [1.9 2.5 38.4 0.01 12.42]
→ i.e. “smaller”, real-valued
```

Feature representation

One-hot *vs* Dense

- **One-hot**: each feature is its own dimension
  - Dimensionality is same as number of features
  - Each feature is completely independent from one another
- **Dense**: each feature is a d-dimensional vector
  - Dimensionality is *d*
  - **Similar features have similar vectors**





e.g. [1.9 2.5 38.4 0.01 12.42]
→ i.e. “smaller”, real-valued

Feature representation

Why dense?

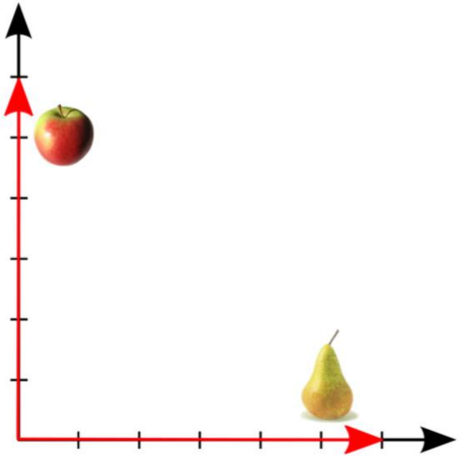
- Discrete approach often works surprisingly well in NLP
    - n-gram language models
    - POS-tagging, parsing
    - sentiment analysis
- Still, a very poor representation of word meaning
    - no notion of similarity
    - limited inference

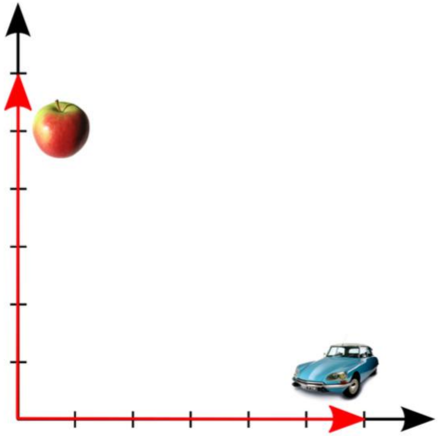
Word representation

- Discrete approach: no notion of similarity

[0000000000000001 0000]

[0000001 0000000000000]





Similarity measure: cosinus

a = apple [1,0,0]

p = pear   [0,1,0]

Cos(a,p) = a.p/(||a|| ||p||)

a.p = 1x0 + 0x1 + 0x0 = 0

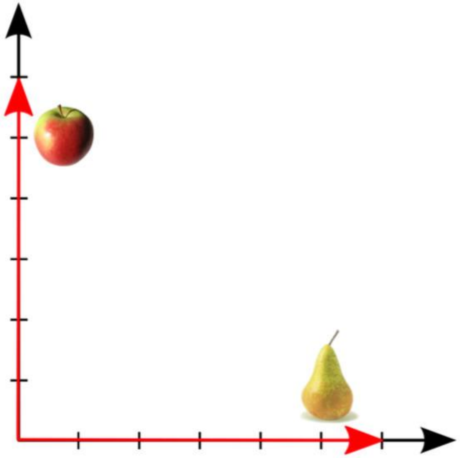
and cos(0) → angle 90°

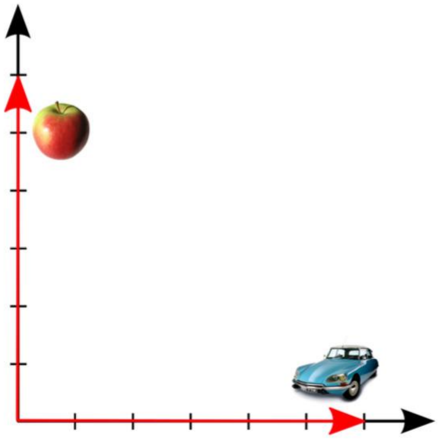
Word representation

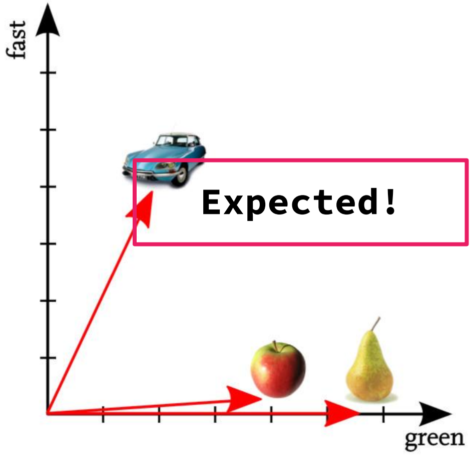
- Discrete approach: no notion of similarity

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]

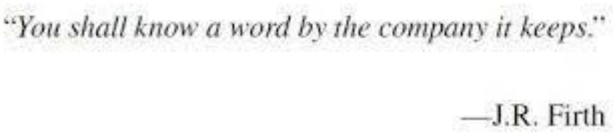






Expected!

Word distribution



"You shall know a word by the company it keeps."

—J.R. Firth

- Rather old idea: **distributional hypothesis** → 1950's!

Example (from Tim van der Cruys):

- delicious *sooluceps*
- sweet *sooluceps*
- stale *sooluceps*
- freshly baked *sooluceps*

→ Guess what is a *sooluceps* ?

Word distribution

> "You shall know a word by the company it keeps."
>
> —J.R. Firth

- Rather old idea: **distributional hypothesis** → 1950's!

Example (from Tim van der Cruys):

- delicious *sooluceps*
- sweet *sooluceps*
- stale *sooluceps*
- freshly baked *sooluceps*

Food!

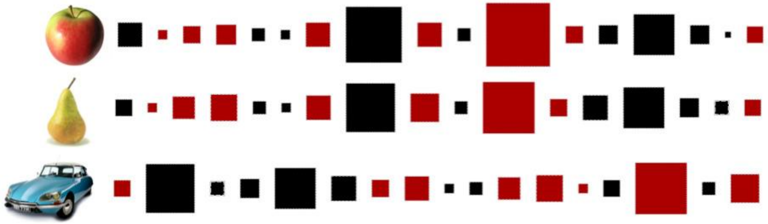
→ Guess what is a *sooluceps* ?

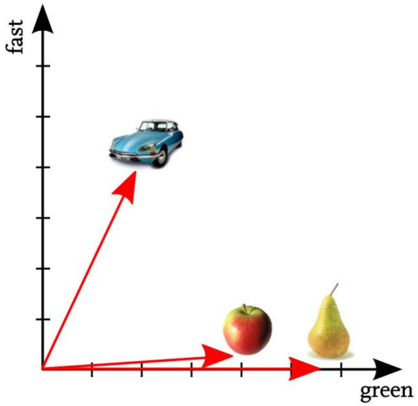
Looking at the context of use of a word, you can guess its meaning

Representing the meaning of words using context

Before neural networks:

- build a matrix over all the words appearing in a corpus
- count the number of time words appear together
- reduce the dimensions (e.g. PCA)





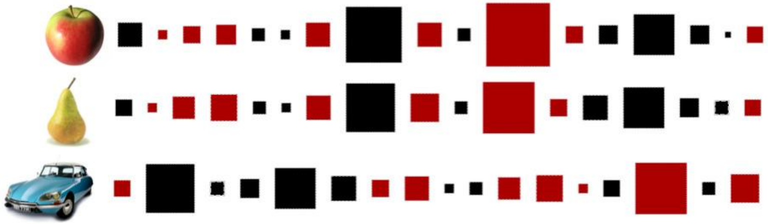
Representing the meaning of words using context

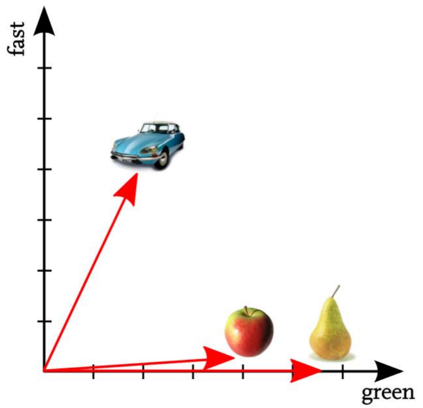
Before neural networks:

- build a matrix over all the words appearing in a corpus
- count the number of time words appear together
- reduce the dimensions (e.g. PCA)

Now: **Train a neural network to build a representation**

- massive amount of data
- task = predicting a linguistic unit (word, sentence…)

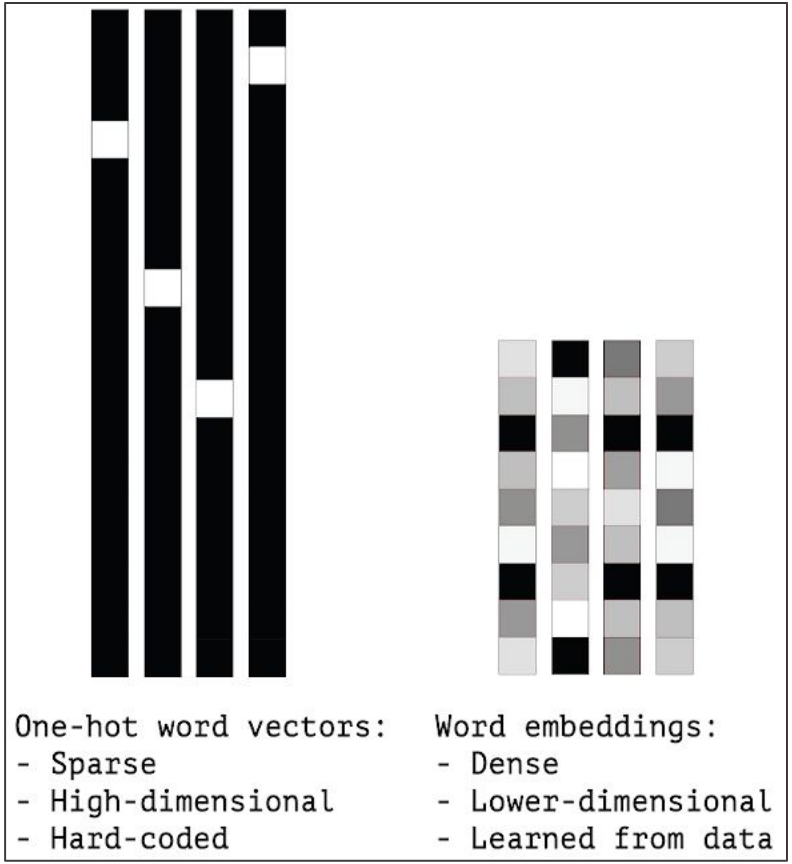




Feature representation

Why dense?

- Better representation of word meaning:
  - similar words have similar vectors
  - allows inference (talk about that later)
- What happens if we use a sparse vector as input of a NN?
  - the first layer = **learns a dense embedding vector over each input**



Feature combinations

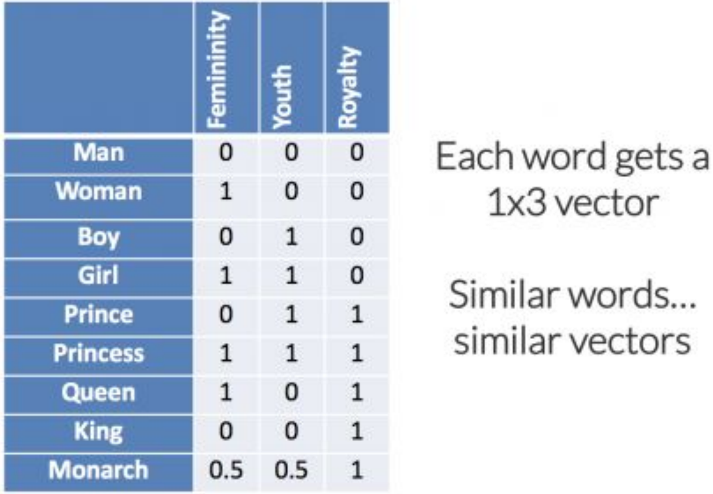
- Traditional NLP:
  - specify interactions of features
  - e.g. ‘word is *jump*, tag is *V* and previous word is *they*’
  - crucial because it introduces more dimensions → linearly separable
  - but the space of combinations is very large, time consuming
- Non-linear network:
  - only specify core features
  - **non-linearity takes care of finding indicative feature combinations**
  - (Note: it was also the case with non linear kernel methods, but with these methods, training becomes very slow when the size of the data increases, ie. scales linearly with the size of the training set *vs* NN scales linearly with the size of the network)

Word embeddings with Deep Learning

How can we **define vectors representing word meaning**?

⟶ we want to be able to represent similarity between words

We could use semantic attributes as dimensions of the vector, e.g. animated, animal, like coffee… ⟶ very complicated to find these attributes



	Femininity	Youth	Royalty
Man	0	0	0
Woman	1	0	0
Boy	0	1	0
Girl	1	1	0
Prince	0	1	1
Princess	1	1	1
Queen	1	0	1
King	0	0	1
Monarch	0.5	0.5	1

Each word gets a 1x3 vector

Similar words... similar vectors

Word embeddings with Deep Learning

How can we **define vectors representing word meaning**?

⟶ we want to be able to represent similarity between words

We could use semantic attributes as dimensions of the vector, e.g. animated, animal, like coffee… ⟶ very complicated to find these attributes

Central idea of DL: the neural network **learns representations** of the features, rather than requiring the programmer to design them

- let the word embeddings be parameters in our model,
- and then be updated during training

Word embeddings with Deep Learning

How can we **define vectors representing word meaning**?

⟶ we want to be able to represent similarity between words

We could use semantic attributes as dimensions of the vector, e.g. animated, animal, like coffee… ⟶ very complicated to find these attributes

Central idea of DL: the neural network **learns representations** of the features, rather than requiring the programmer to design them

- let the **word embeddings be parameters in our model**,
- and then be updated during training

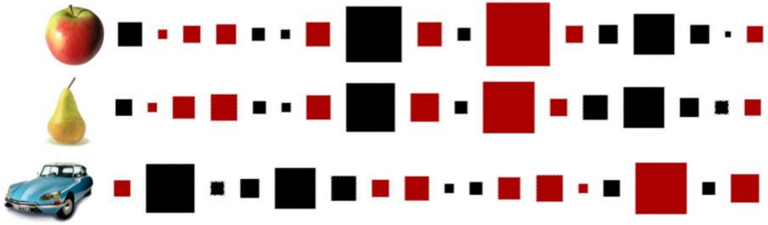
Word embeddings with Deep Learning

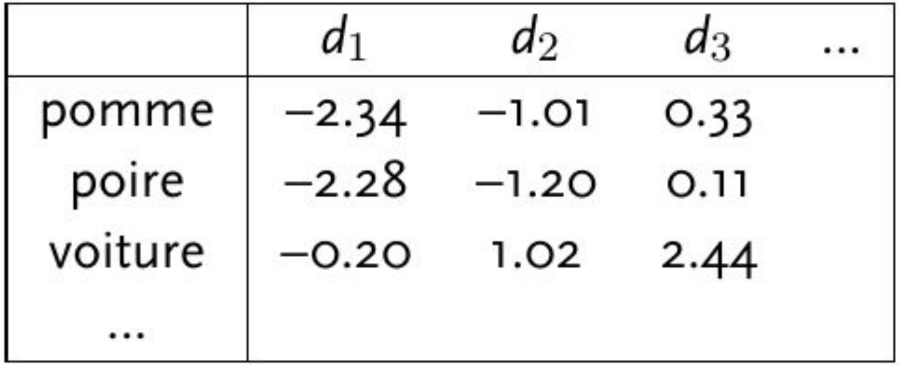
*Word embeddings are a **representation of the *semantics*** of a word, efficiently encoding semantic information that **might be relevant to the task** at hand.*

- dimensions = “latent semantic attributes”: but not directly interpretable
- “relevant to the task at hand”:
    - for example “bad” and “good” need to have opposite vectors for sentiment classification but it’s not crucial for POS tagging
    - the domain is crucial: typical example of “avocat” which is mostly used with one meaning for cooking and another one when referring to a lawyer
    - leads to an important issue: words often have several meanings...
- you can embed anything: POS, morphological information, parse tree etc

Word embeddings with Deep Learning

- Word embeddings: semantic representation of the words, used as basic features
- Similar words have similar embeddings
- Each word *i* is represented with a (unique) vector $\mathbf{v}_i \in R^d$
- *d* is typically between 50 and 1000



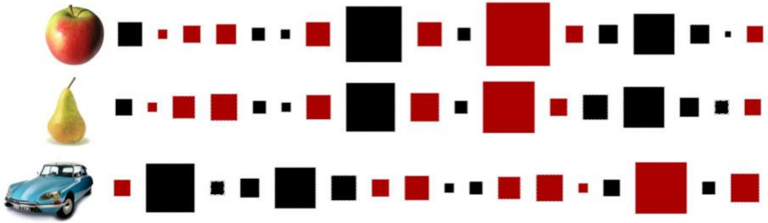


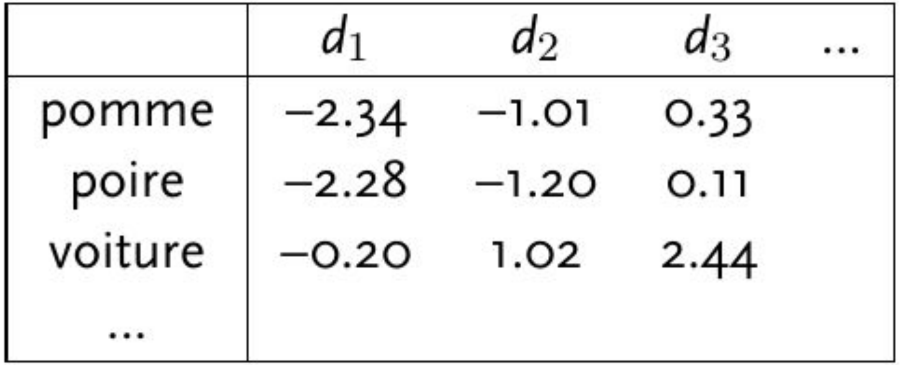
	d_1	d_2	d_3	...
pomme	−2.34	−1.01	0.33	
poire	−2.28	−1.20	0.11	
voiture	−0.20	1.02	2.44	
...				

Word embeddings with Deep Learning

one-hot: it’s an “embedding” of the words but high-dimensional and not dense, not real-valued

- Word embeddings: semantic representation of the words, used as basic features
- Similar words have similar embeddings
- Each word *i* is represented with a (unique) vector $\mathbf{v}_i \in R^d$
- *d* is typically between 50 and 1000





	d_1	d_2	d_3	...
pomme	−2.34	−1.01	0.33	
poire	−2.28	−1.20	0.11	
voiture	−0.20	1.02	2.44	
...				

Pre-trained word embeddings

- Considering the embeddings as trainable parameters is very cool: they can be updated while performing the task, thus adapted to the task
- But building good word representations require a massive amount of data: we need to see each word many times with varied contexts

→ probably not enough with your training set

Solution: pre-train word embeddings on massive amount of data and then use them as is / as initialization

Popular pre-trained word embeddings

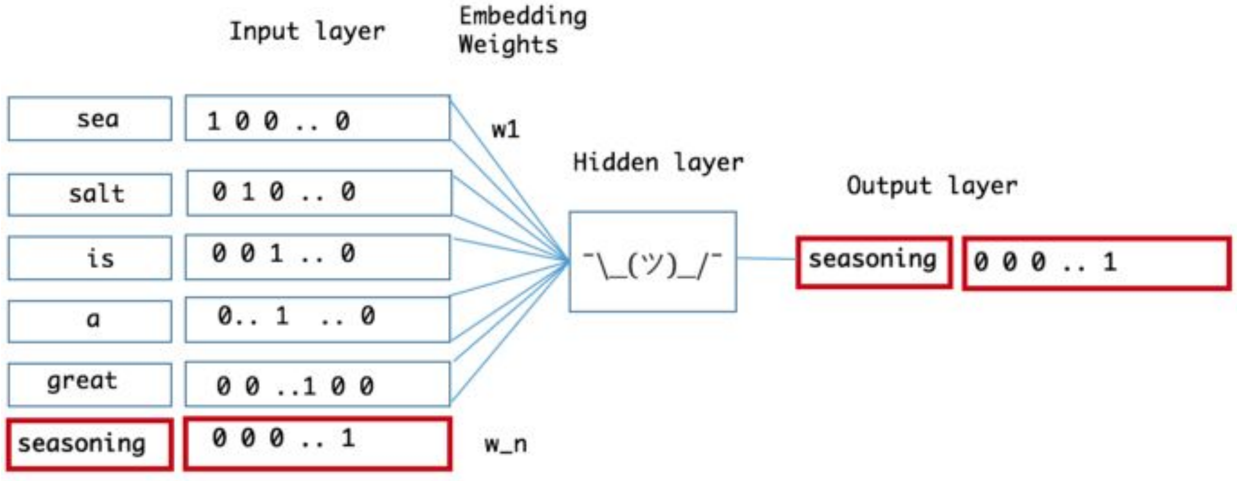
Word2Vec (Google, [Mikolov et al. 2013])

If two different words have very ***similar "contexts"*** *(that is, what words are likely to appear around them), then* ***our model needs to output very similar results*** *for these two words. And one way for the network to output similar context predictions for these two words is* ***if the word vectors are similar****. So, if two words have similar contexts, then our network is motivated to learn similar word vectors for these two words! Ta da!*

Pre-trained word embeddings

General idea:

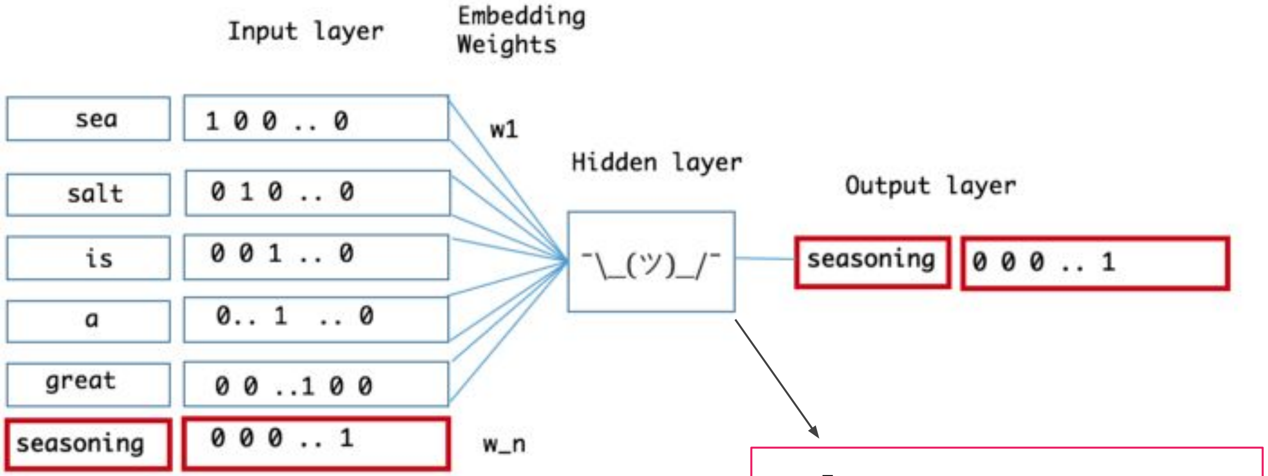
- Use words as input, with one-hot encoding
- Learn a task on words
- But we don’t care about the task
- Hidden layer = the new representation of the word = embeddings



Pre-trained word embeddings

General idea:

- Use words as input, with one-hot encoding
- Learn a task on words
- But we don’t care about the task
- **Hidden layer = the new representation of the word = embeddings**



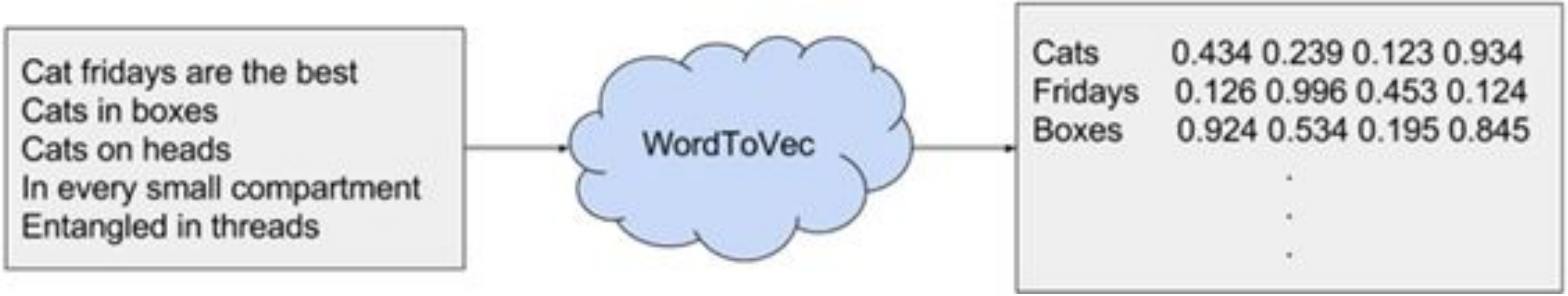
$$W=\begin{bmatrix} 0.4 & 9.2 & \dots & -4.3 \\ 1.3 & 5.4 & \dots & 6.7 \\ \dots & & & \\ -4.5 & 3.2 & \dots & -5.3 \end{bmatrix}$$

Word2vec

Idea: Use a classifier to predict which words appear in the context of a target word (or vice versa). This classifier induces a dense vector representation of words

- input: text corpus
- output: a vector representation for each word
- 2 flavors:
    - CBOW: uses each of these contexts to predict the current word *w*
    - SkipGram: use the current word *w* in order to predict its neighbors (i.e., its context)

→ To limit the number of words in each context, use a parameter called **window size**



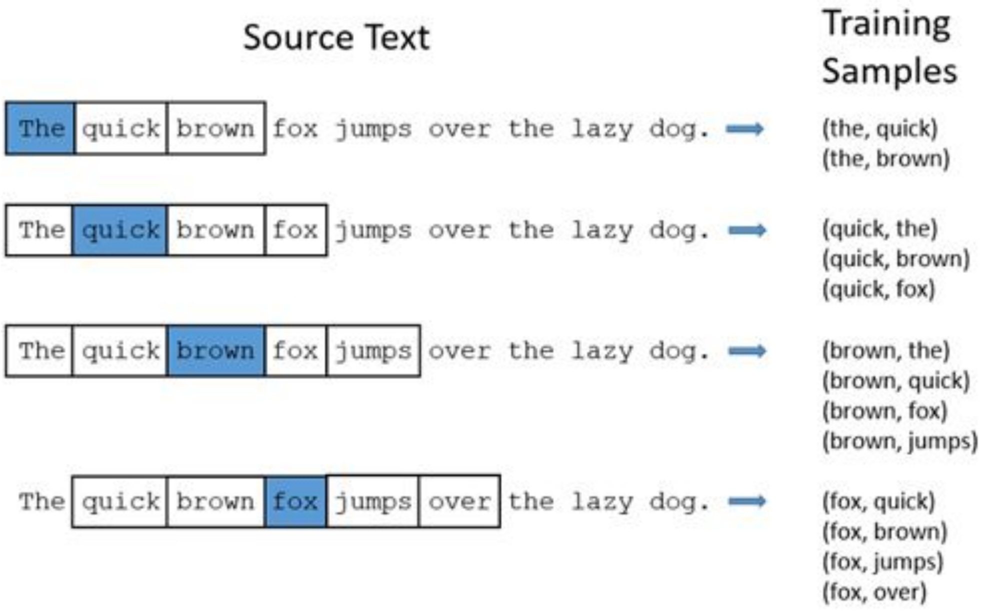
Cat fridays are the best
Cats in boxes
Cats on heads
In every small compartment
Entangled in threads

Cats	0.434 0.239 0.123 0.934
Fridays	0.126 0.996 0.453 0.124
Boxes	0.924 0.534 0.195 0.845
	.
	.
	.

Word2vec

Go through the text and:

- for each target word (in blue)
- consider some context words (here window = 5)



Source Text

Training Samples

The quick brown fox jumps over the lazy dog. ➡ (the, quick) (the, brown)

The quick brown fox jumps over the lazy dog. ➡ (quick, the) (quick, brown) (quick, fox)

The quick brown fox jumps over the lazy dog. ➡ (brown, the) (brown, quick) (brown, fox) (brown, jumps)

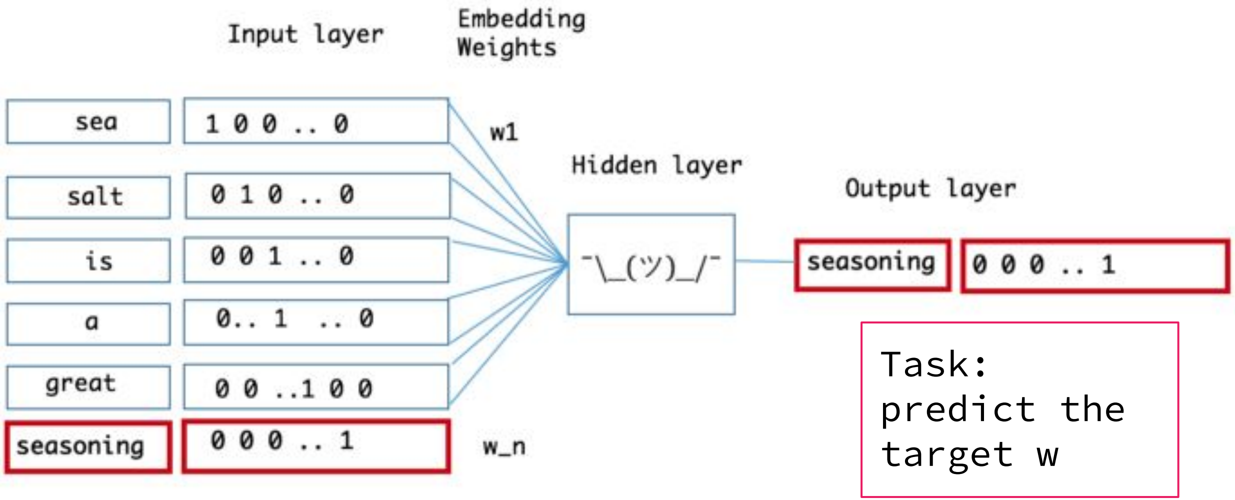
The quick brown fox jumps over the lazy dog. ➡ (fox, quick) (fox, brown) (fox, jumps) (fox, over)

Word2vec - CBOW

Continuous Bag-of-Words (CBOW):

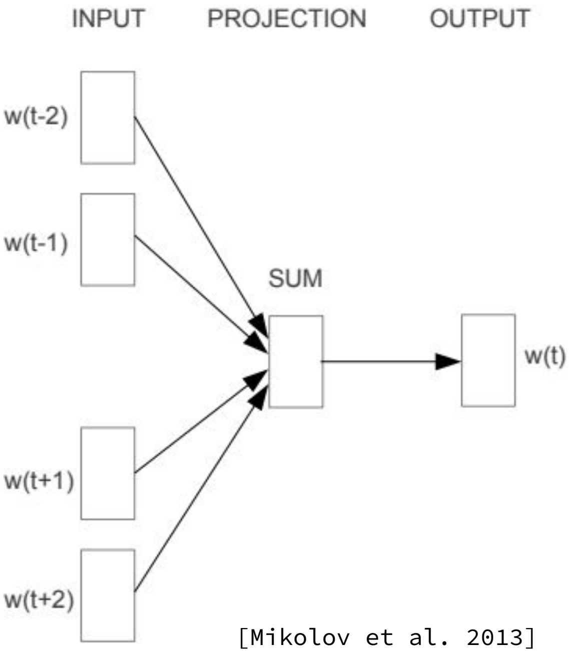
- **Task: predict the target word given the context**
- Resulting embeddings: the weights of the hidden layer are used as the representation of the target word

Context as input/features

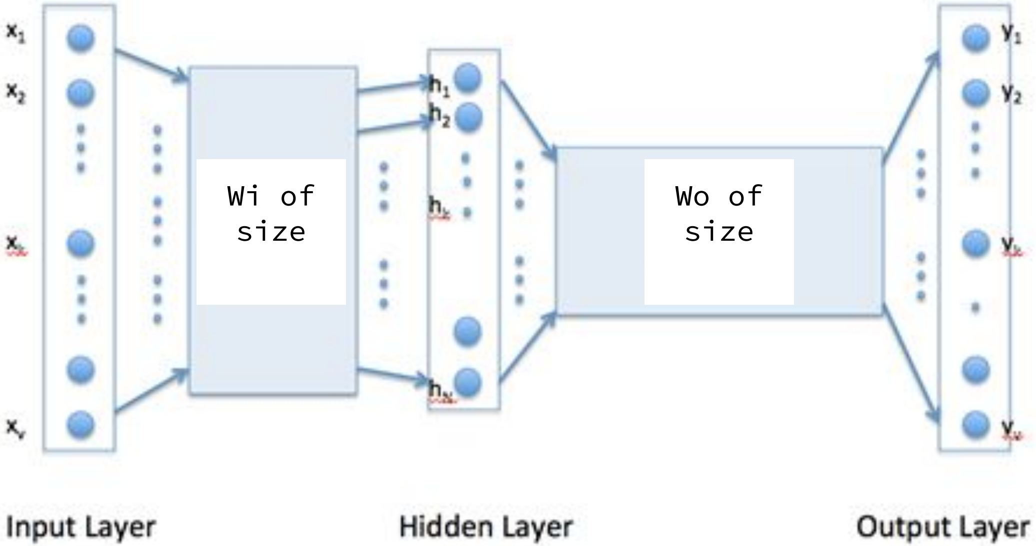


Task: predict the target w

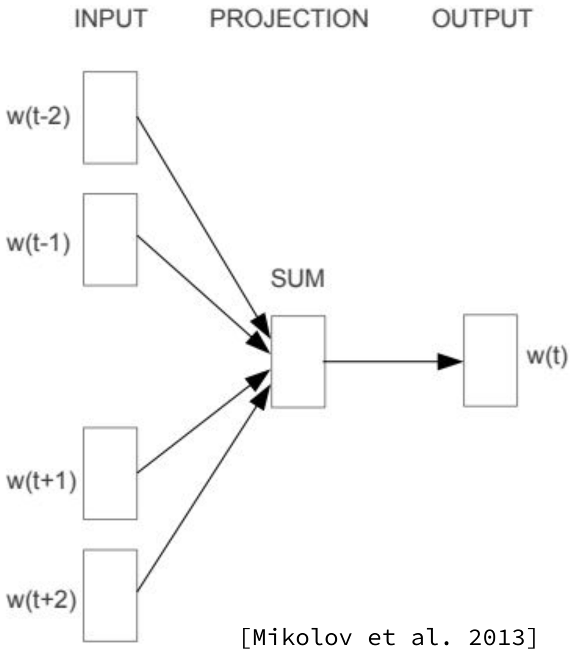
Word2vec - CBOW: architecture



[Mikolov et al. 2013]

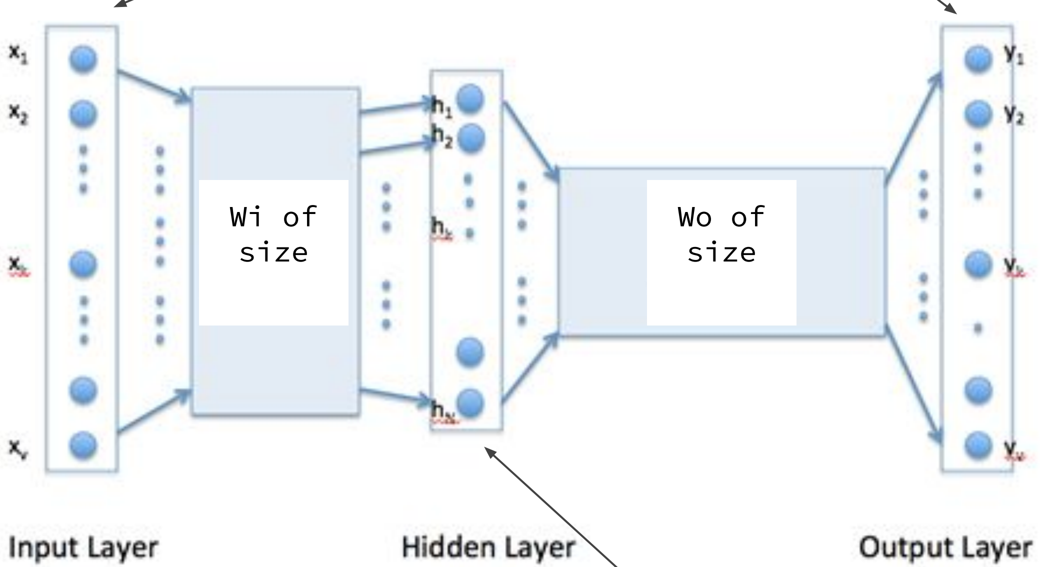


Word2vec - CBOW: architecture



[Mikolov et al. 2013]

V: size of the vocabulary
= input and output dimension

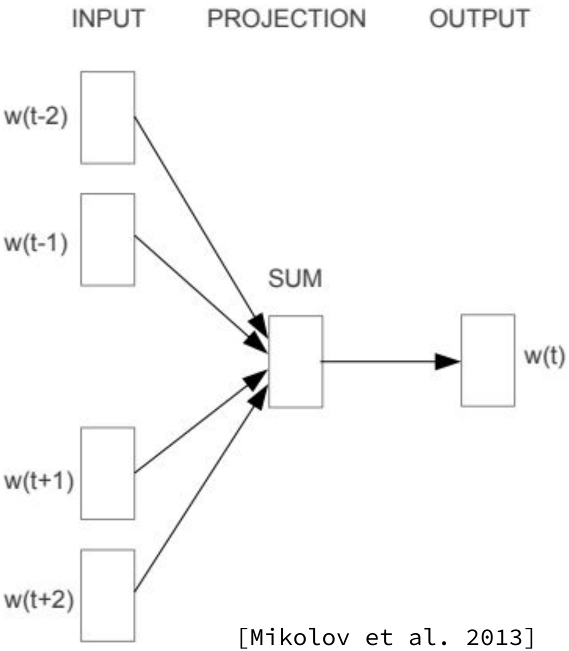


Wi of size

Wo of size

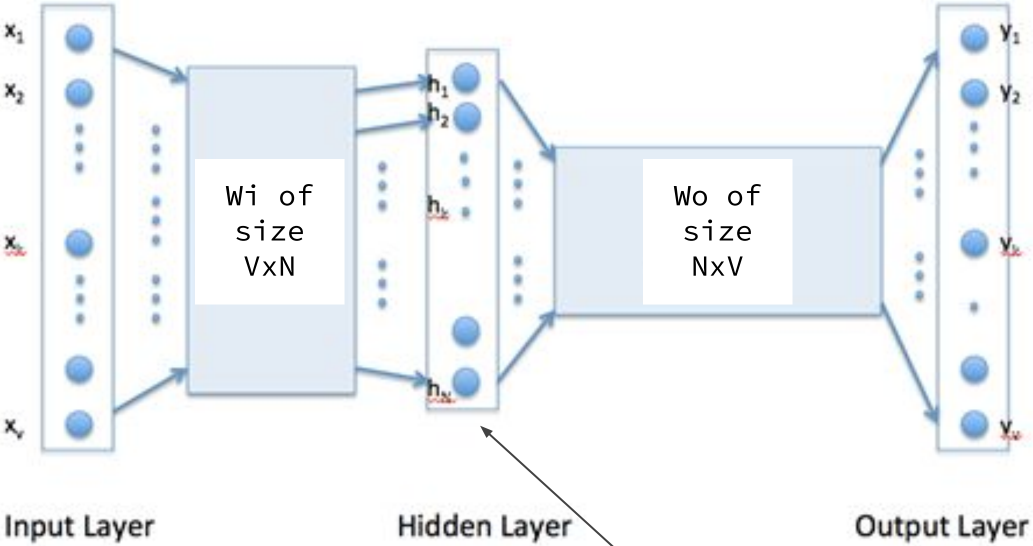
N: size of the desired output

Word2vec - CBOW: architecture



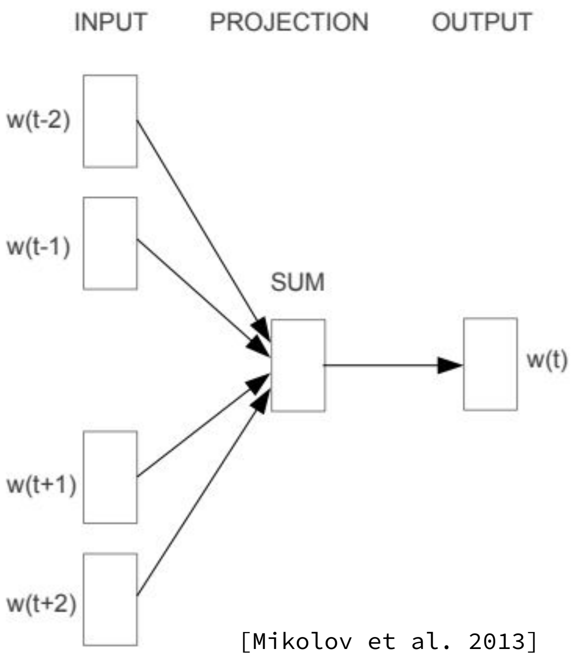
[Mikolov et al. 2013]

V: size of the vocabulary
= input and output dimension



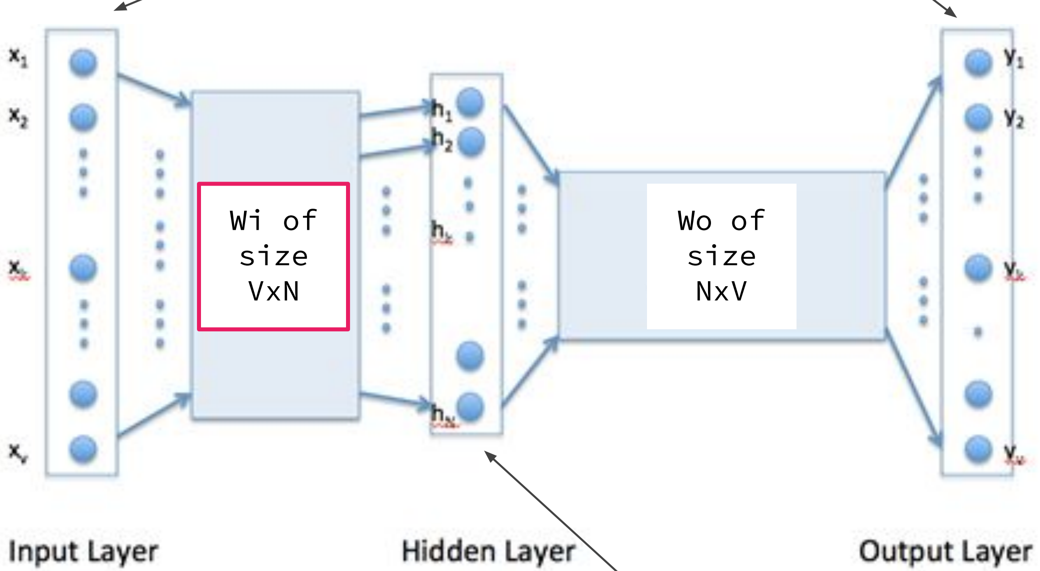
N: size of the desired output

Word2vec - CBOW: architecture



[Mikolov et al. 2013]

V: size of the vocabulary
= input and output dimension



Wi of size VxN

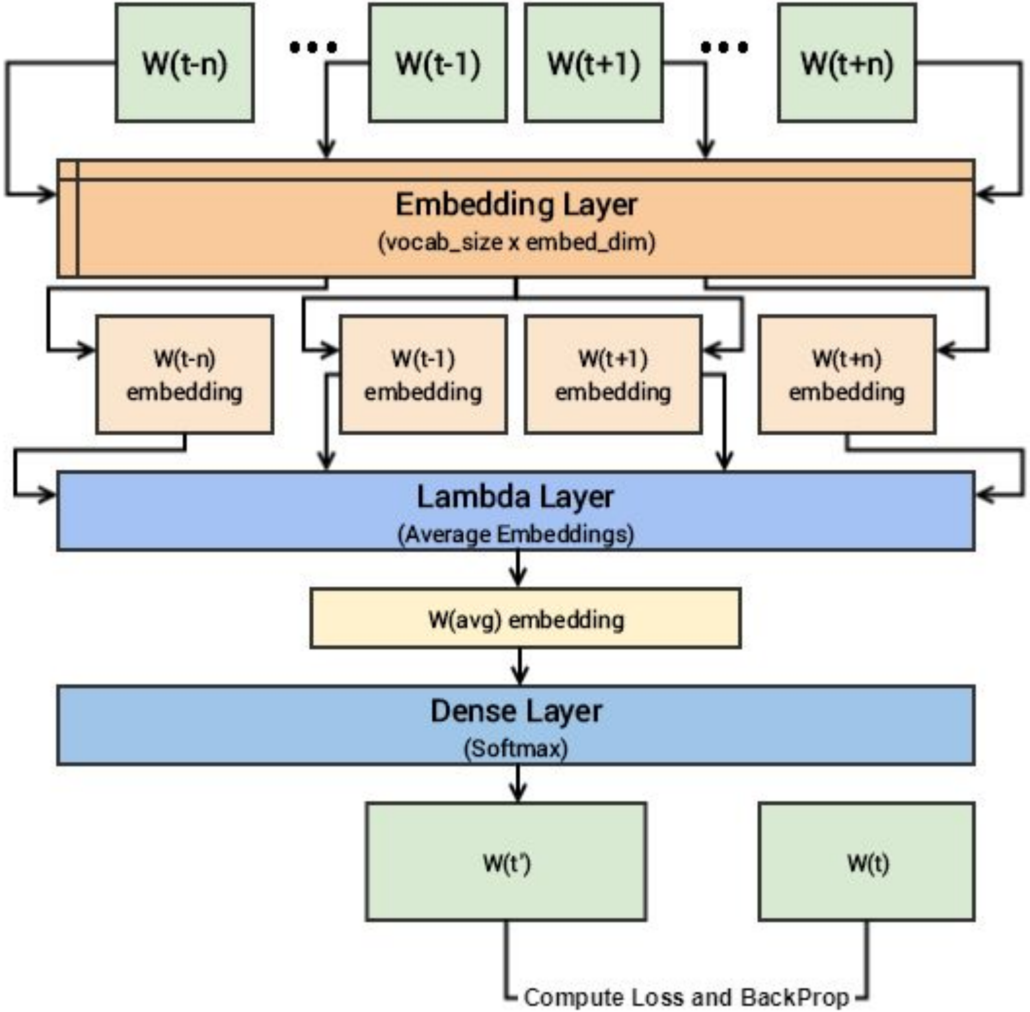
Wo of size NxV

N: size of the desired output

Word2vec - CBOW

Simplified NN → 1 "hidden layer": linear

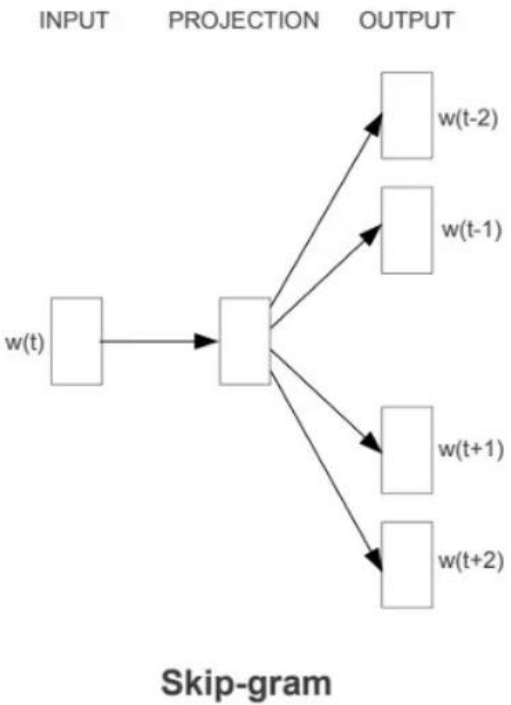
- embedding layer
- lambda layer
- output layer



Word2vec - Skip-Gram

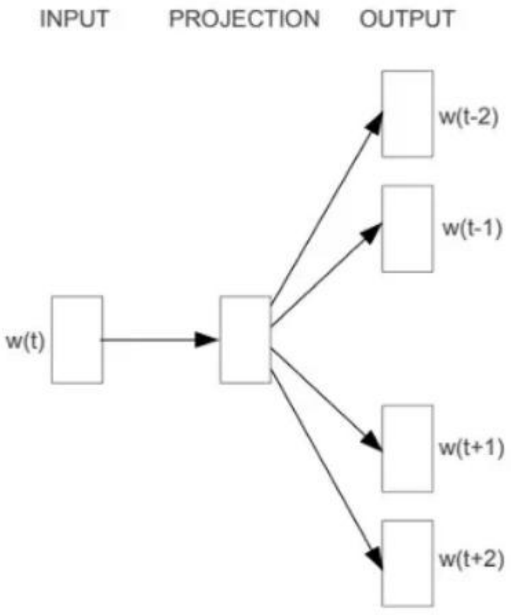
Task: predict the context words given the target word

- Input: one-hot vector of size N representing the target word
- Output: vector (also with N components) containing, for every word in our vocabulary, the probability that a randomly selected nearby word is that vocabulary word

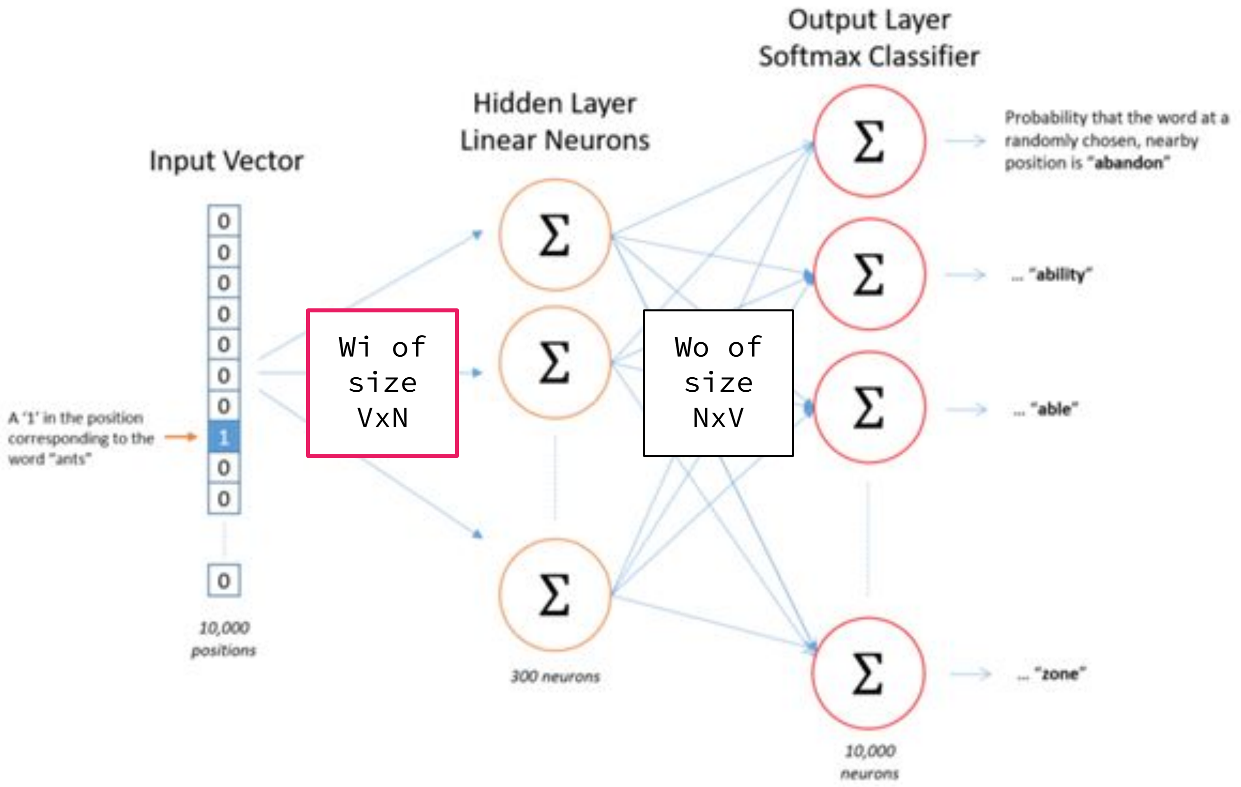


[Mikolov et al. 2013]

Word2vec - Skip-Gram



Skip-gram



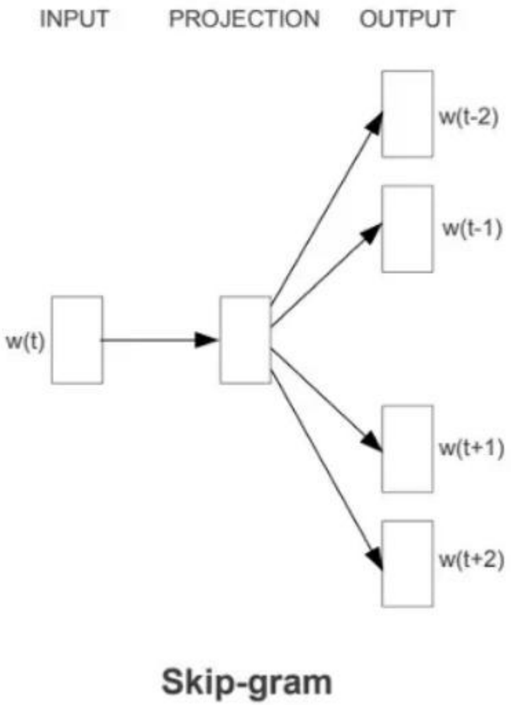
Wi of size VxN

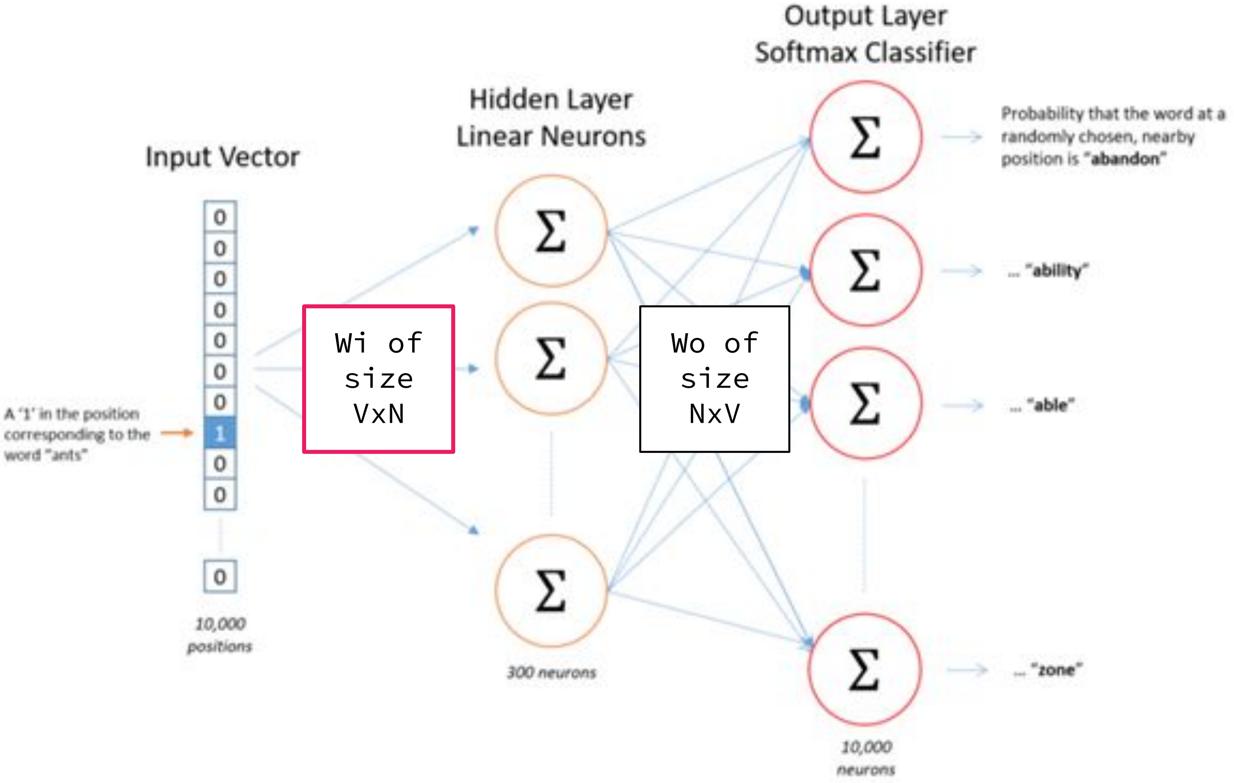
Wo of size NxV

Word2vec - Skip-Gram

Implementation tricks:

- negative sampling
- downsampling

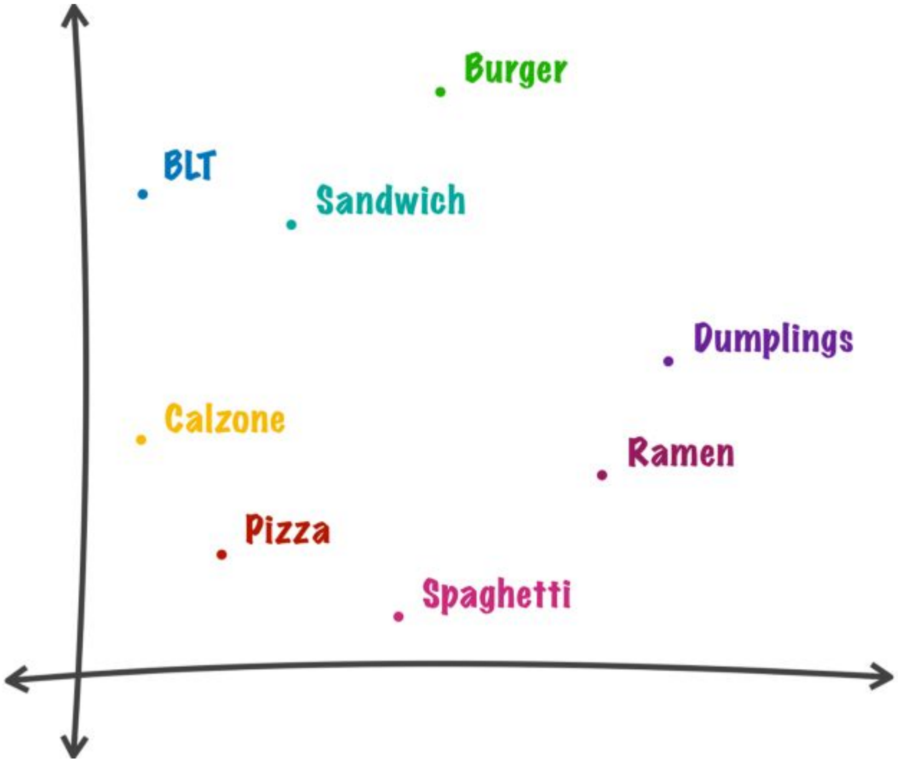



Input Vector
0
0
0
0
0
0
0
1
0
0
0
10,000 positions
A '1' in the position corresponding to the word "ants"
Hidden Layer Linear Neurons
Σ
Σ
Σ
300 neurons
Output Layer Softmax Classifier
Σ
Σ
Σ
Σ
10,000 neurons
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"

Wi of size VxN

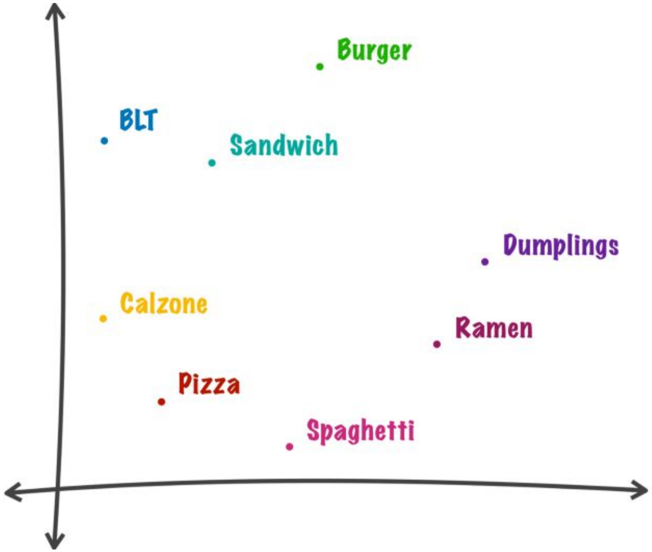
Wo of size NxV

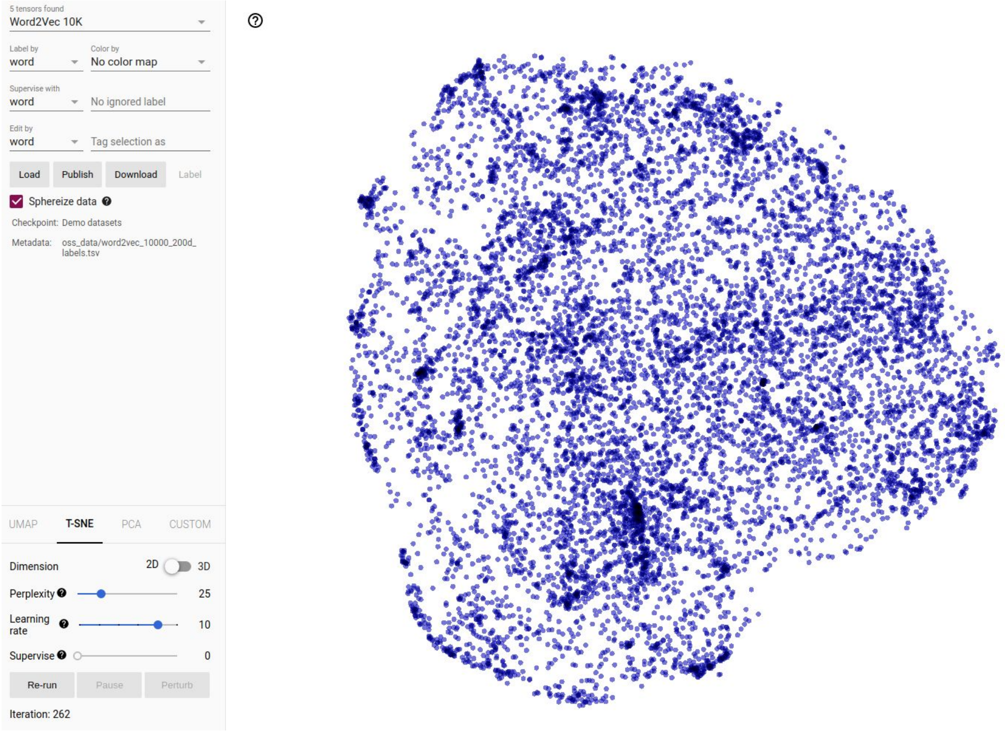
Visualizing embeddings



Visualizing embeddings

https://projector.tensorflow.org/





5 tensors found
Word2Vec 10K

Label by: word

Color by: No color map

Supervise with: word

No ignored label

Edit by: word

Tag selection as

☑ Sphereize data

Checkpoint: Demo datasets

Metadata: oss_data/word2vec_10000_200d_labels.tsv

Dimension: 2D 3D

Perplexity: 25

Learning rate: 10

Supervise: 0

Iteration: 262

Visualisation

Need for dimensionality reduction

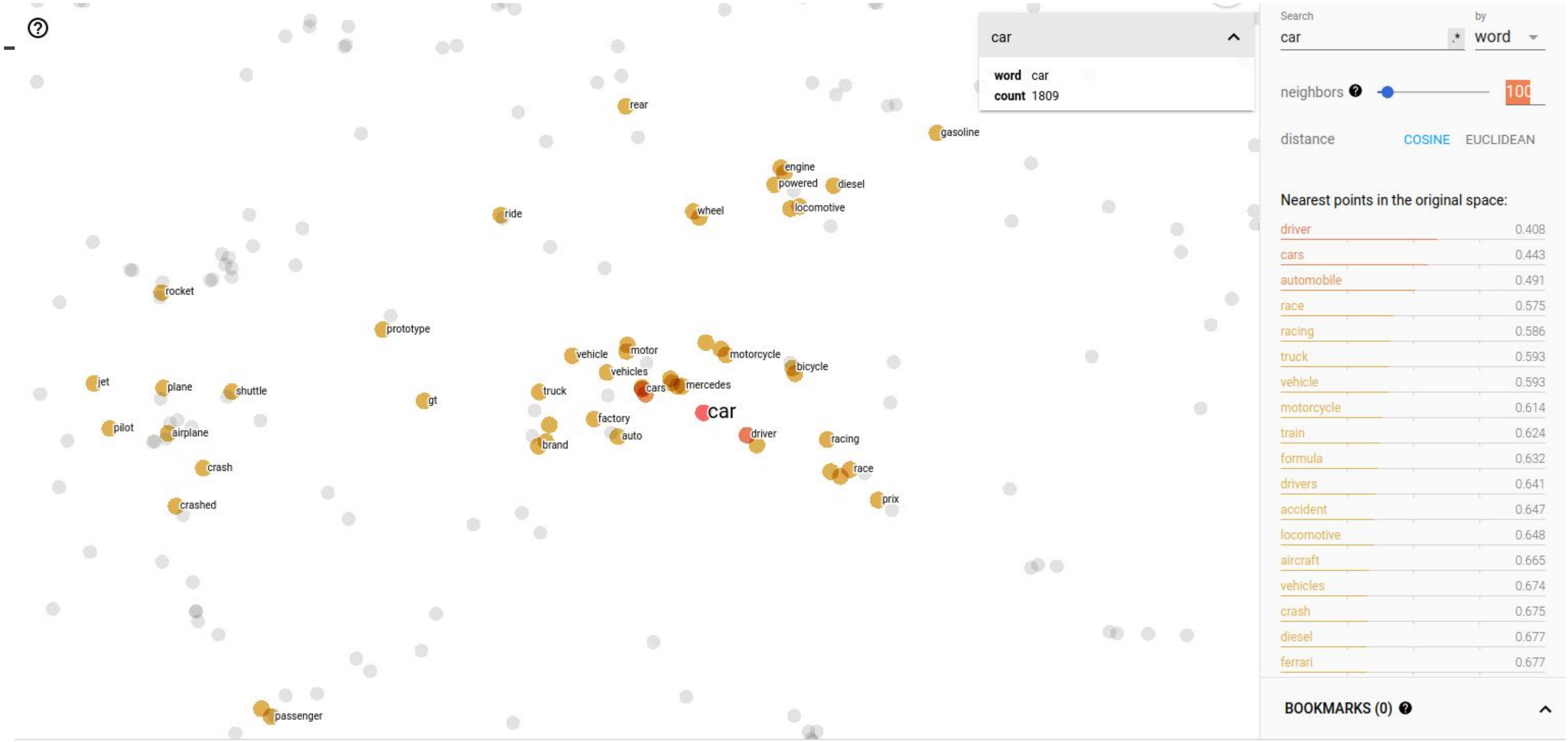
Algorithme t-SNE (*t-distributed stochastic neighbor embedding*):

- from high-dimensional space to 2 or 3 dimensions
- general idea: non-linear methods that keeps dstance, 2 points that were close/far in the original space must be close/far in the new projected space
- Using t-sne: https://distill.pub/2016/misread-tsne/

PCA (principal component analysis):

- from high-dimensional space to 2 or 3 dimensions
- general idea: from correlated data to uncorrelated data, in general keep the dimensions that explain 90-95% of the data (reduce dimensionality and redundancy)

Word similarity



car

word car
count 1809

Search
car
by word

neighbors 100

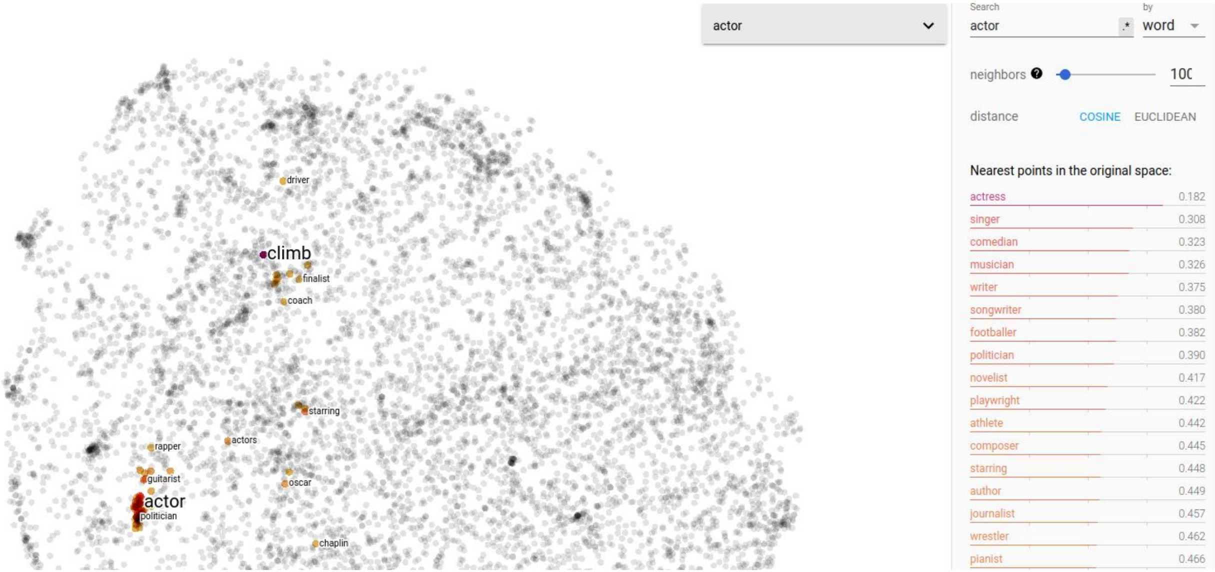
distance COSINE EUCLIDEAN

Nearest points in the original space:

driver	0.408
cars	0.443
automobile	0.491
race	0.575
racing	0.586
truck	0.593
vehicle	0.593
motorcycle	0.614
train	0.624
formula	0.632
drivers	0.641
accident	0.647
locomotive	0.648
aircraft	0.665
vehicles	0.674
crash	0.675
diesel	0.677
ferrari	0.677

BOOKMARKS (0)

Visualizing embeddings



actor

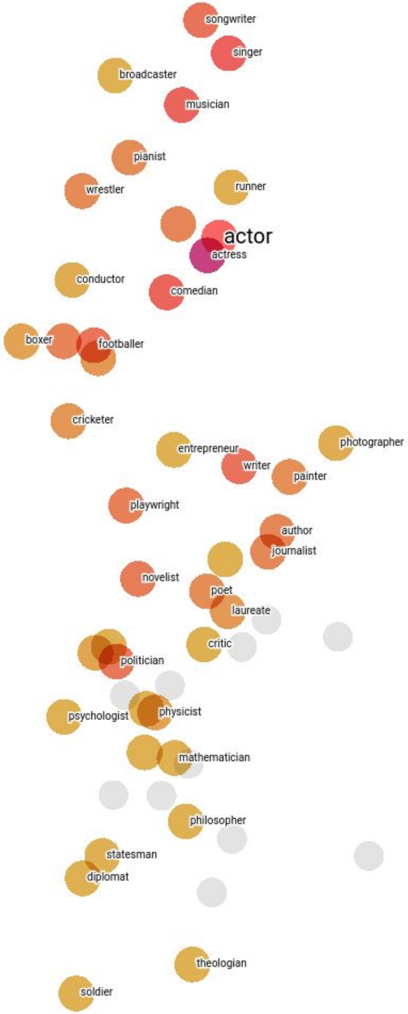
Search: actor .* by word

neighbors 100

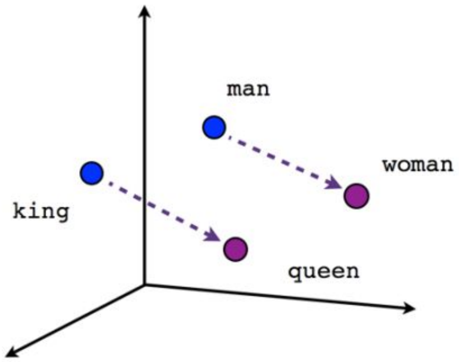
distance COSINE EUCLIDEAN

Nearest points in the original space:

actress	0.182
singer	0.308
comedian	0.323
musician	0.326
writer	0.375
songwriter	0.380
footballer	0.382
politician	0.390
novelist	0.417
playwright	0.422
athlete	0.442
composer	0.445
starring	0.448
author	0.449
journalist	0.457
wrestler	0.462
pianist	0.466



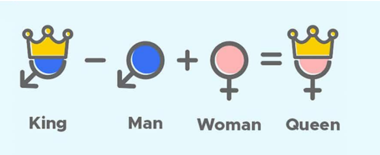
Word2vec

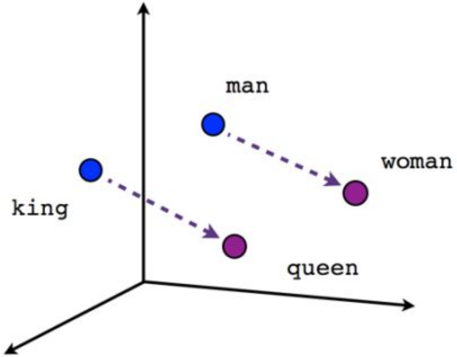


Male-Female

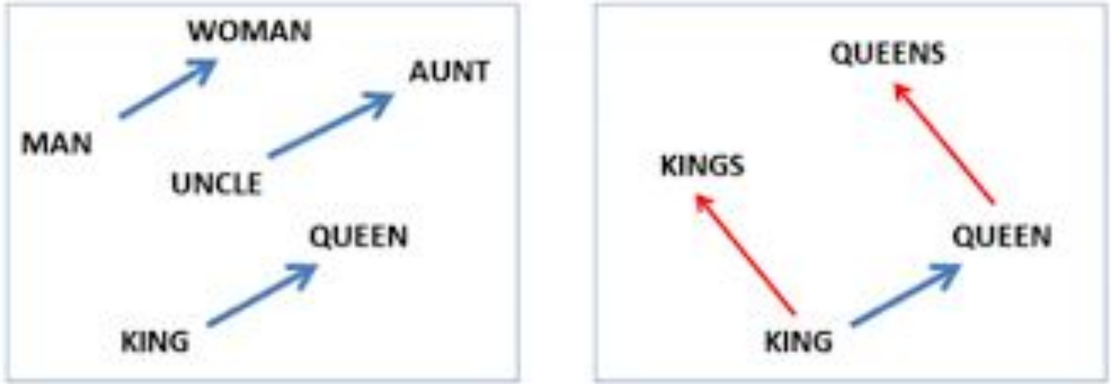
Word2vec

Allow inference:





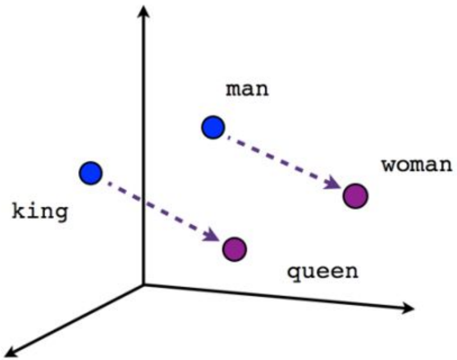
Male-Female



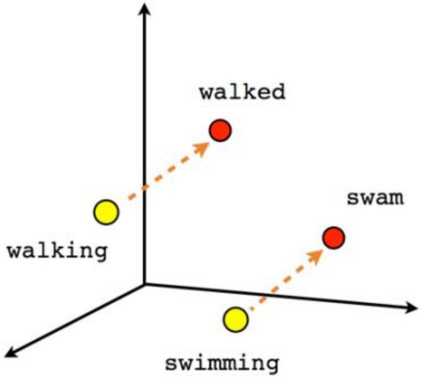
(Mikolov et al., NAACL HLT, 2013)

Word2vec

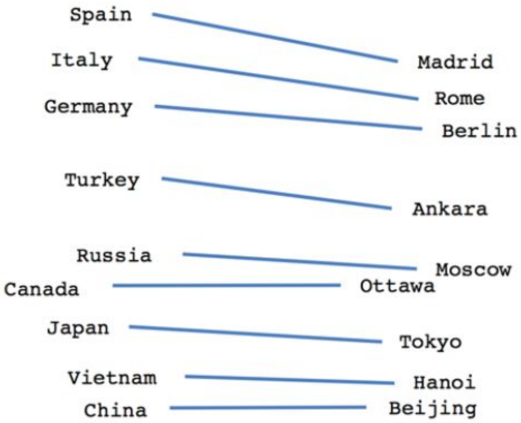
Allow inference:



Male-Female



Verb tense

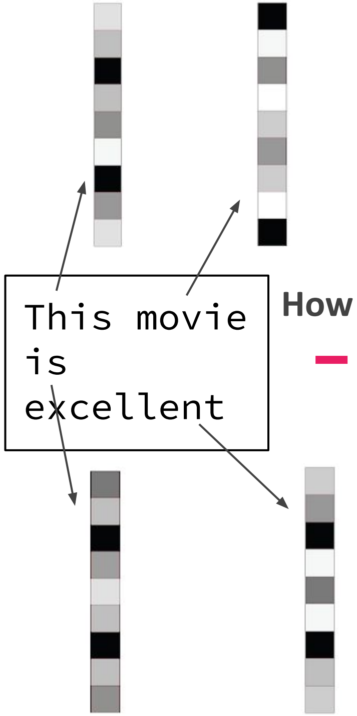


Country-Capital

Popular pre-trained word embeddings

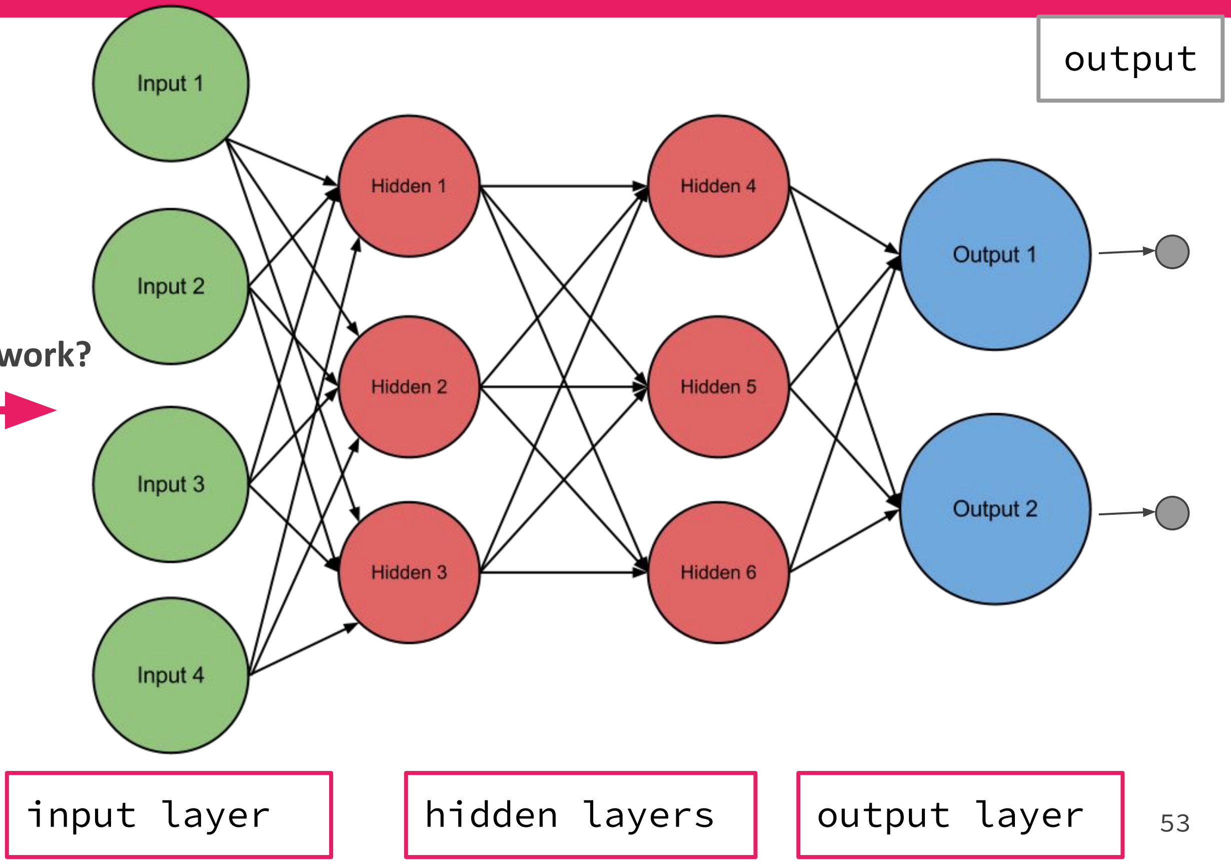
- Word2Vec (Google, [Mikolov et al. 2013])
- GloVe (Stanford, [Pennington et al. 2014]): GloVe is an approach to marry both the global statistics of matrix factorization techniques like LSA with the local context-based learning in word2vec. Rather than using a window to define local context, GloVe constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus. The result is a learning model that may result in generally better word embeddings. (https://machinelearningmastery.com/what-are-word-embeddings/)
- FastText (Facebook, [Bojanovski et al. 2016]): approach based on the skipgram model, where each word is represented as a bag of character n-grams (use subwords information)
- Talk later: context-sensitive embeddings
    - ELMo (AllenNLP, [Peters et al. 2018])
    - BERT (Google, [Devlin et al. 2018])

Input vector

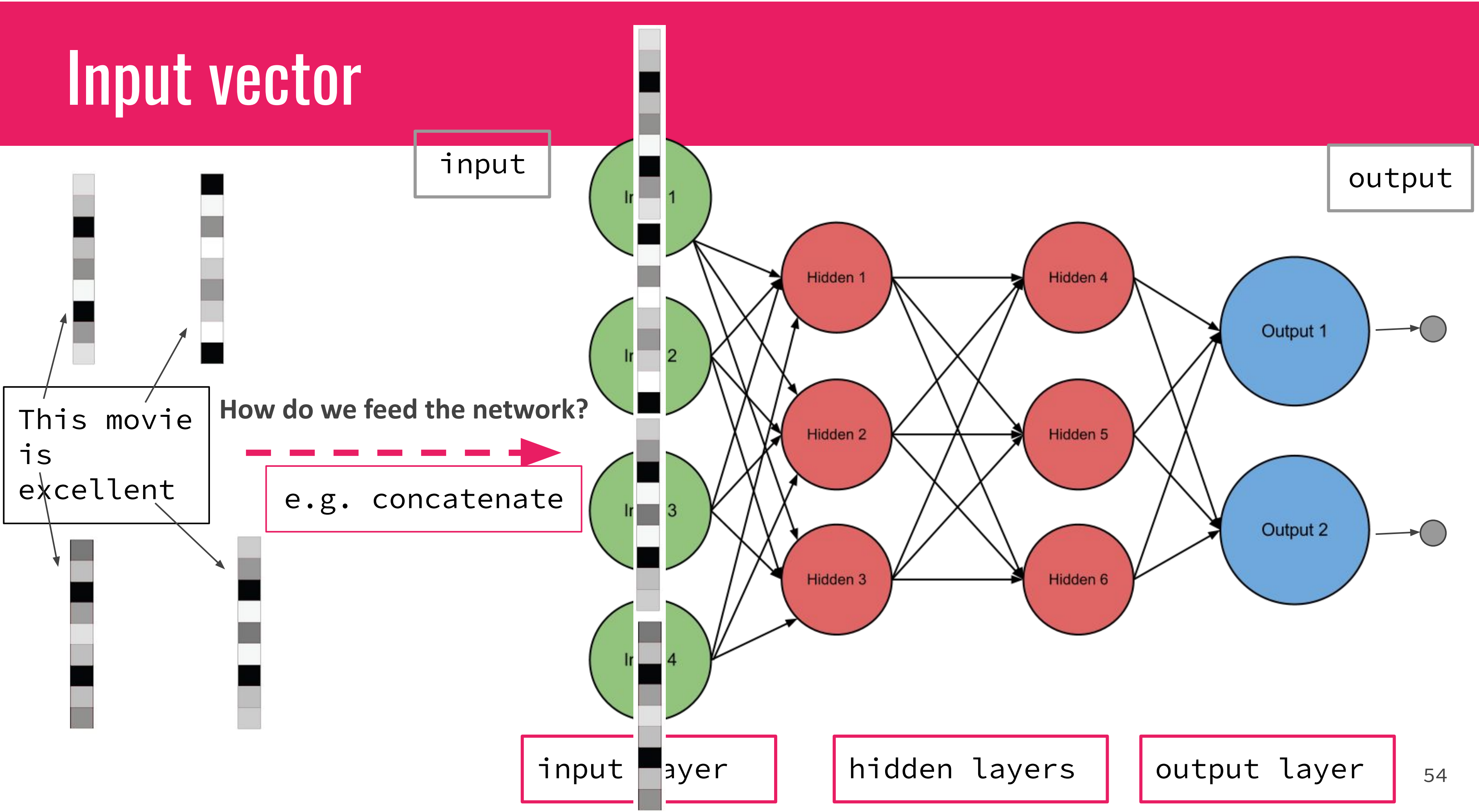


This movie is excellent

How do we feed the network?



Input vector



input

output

This movie is excellent

How do we feed the network?

e.g. concatenate

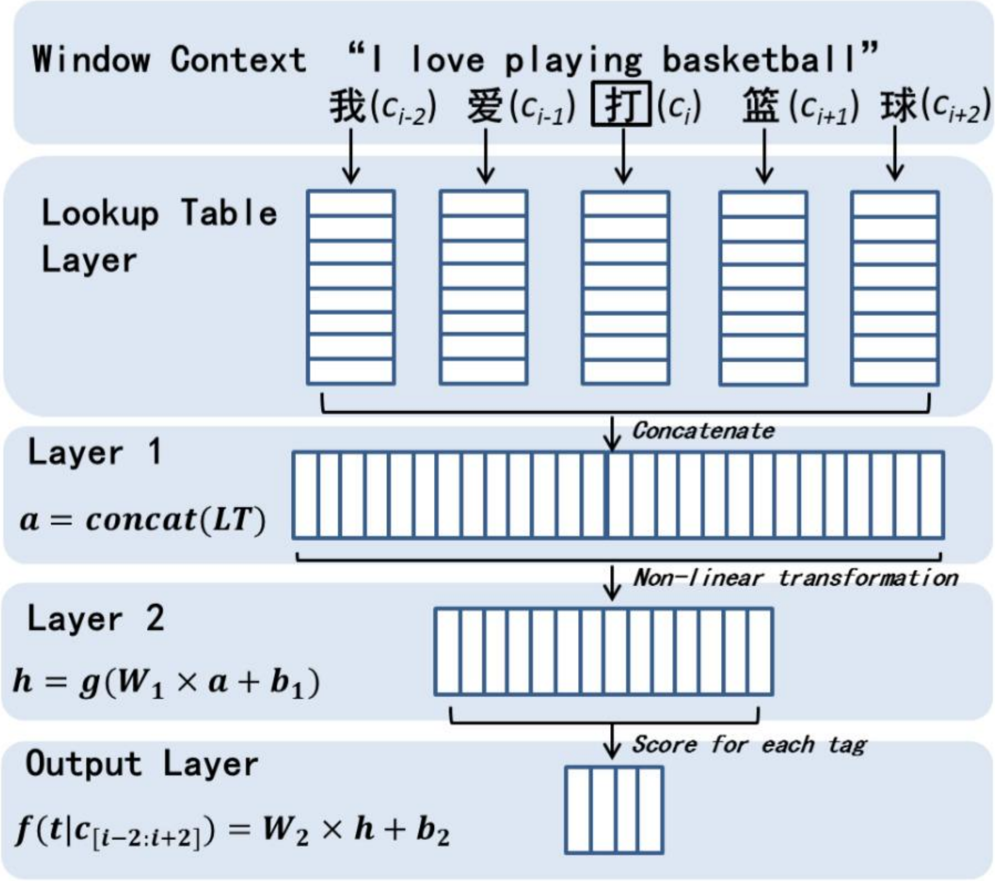
input layer

hidden layers

output layer

Input vector

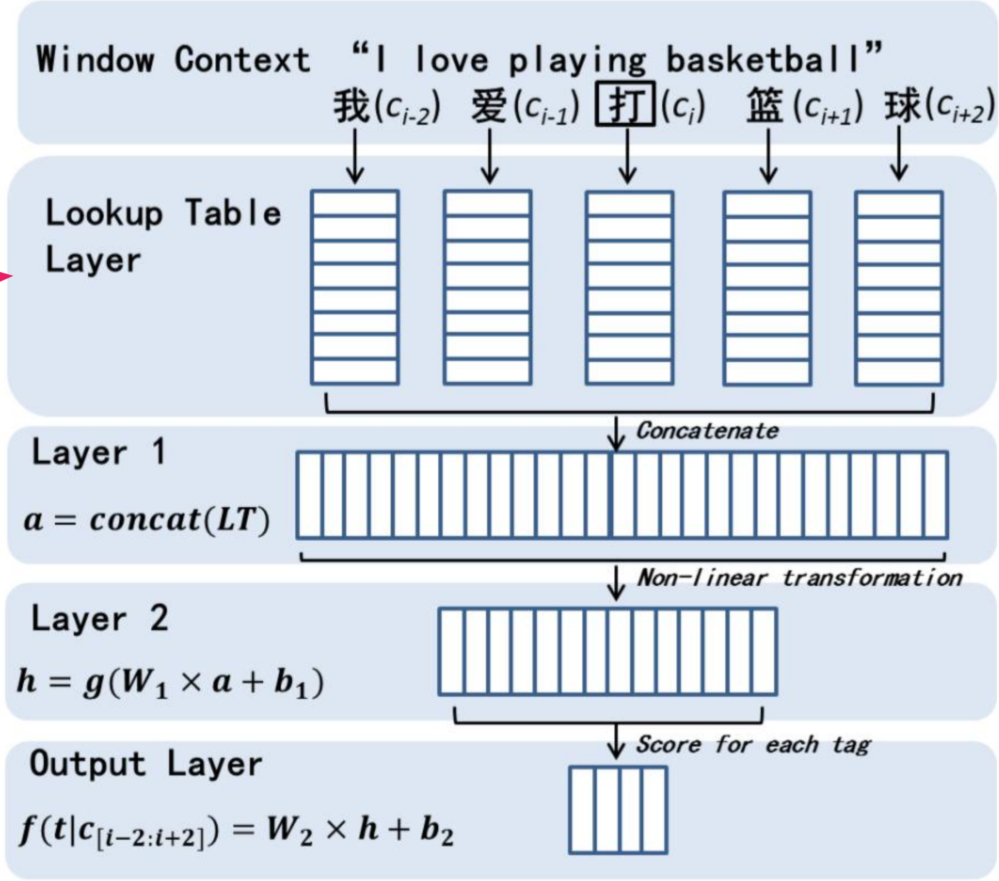
- Embedding lookup from embedding matrix
- Layer 1 = embedding layer:
    - e.g. mean or sum embeddings
- Layer 2 = hidden



Window Context “I love playing basketball”
我(c_{i-2}) 爱(c_{i-1}) 打(c_i) 篮(c_{i+1}) 球(c_{i+2})
Lookup Table Layer
Concatenate
Layer 1
$a = concat(LT)$
Non-linear transformation
Layer 2
$h = g(W_1 \times a + b_1)$
Score for each tag
Output Layer
$f(t|c_{[i-2:i+2]}) = W_2 \times h + b_2$

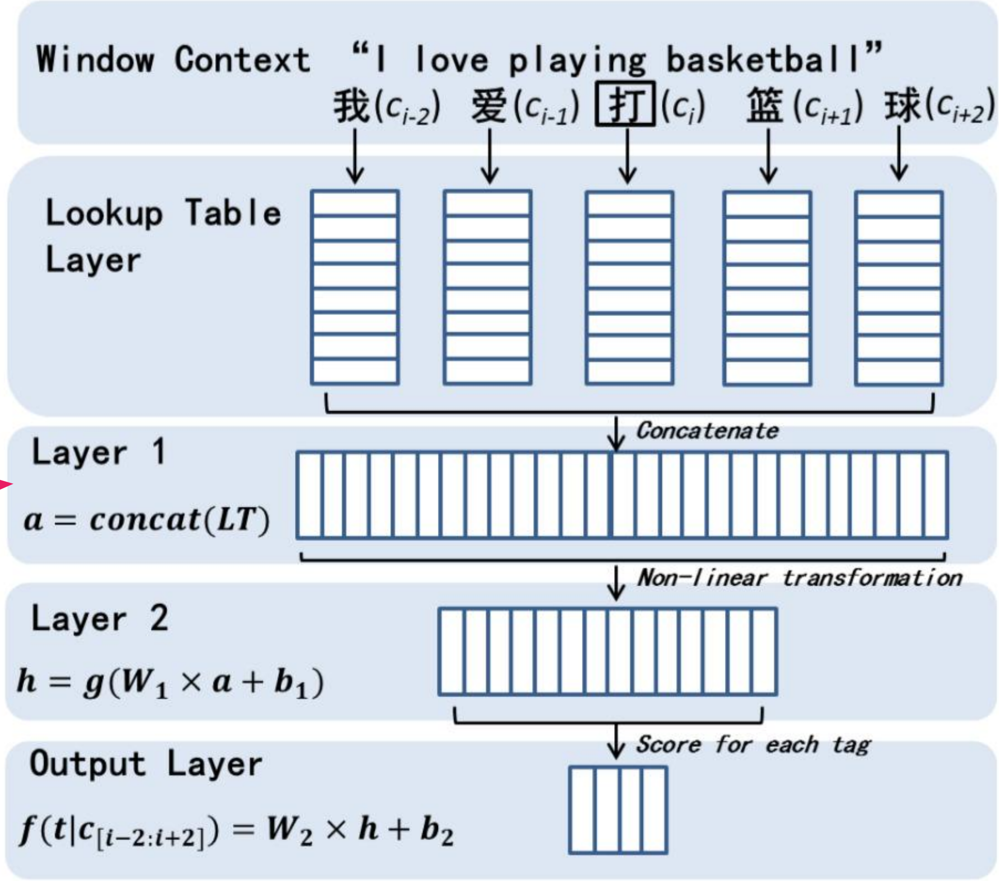
Input vector

- Embedding **lookup** from embedding matrix
- Layer 1 = embedding layer:
  - e.g. mean or sum embeddings
- Layer 2 = hidden


Window Context "I love playing basketball"
我(c_{i-2}) 爱(c_{i-1}) 打(c_i) 篮(c_{i+1}) 球(c_{i+2})
Lookup Table Layer
Concatenate
Layer 1
$a = concat(LT)$
Non-linear transformation
Layer 2
$h = g(W_1 \times a + b_1)$
Score for each tag
Output Layer
$f(t|c_{[i-2:i+2]}) = W_2 \times h + b_2$

Input vector

- Embedding lookup from embedding matrix
- Layer 1 = **embedding layer:**
  - e.g. mean or sum embeddings
- Layer 2 = hidden



Window Context "I love playing basketball"
我(c_{i-2}) 爱(c_{i-1}) 打(c_i) 篮(c_{i+1}) 球(c_{i+2})
Lookup Table Layer
Concatenate
Layer 1
$a = concat(LT)$
Non-linear transformation
Layer 2
$h = g(W_1 \times a + b_1)$
Score for each tag
Output Layer
$f(t|c_{[i-2:i+2]}) = W_2 \times h + b_2$

Using dense vectors in PyTorch

- the mapping from words to indices is a dictionary, generally named *word_to_ix*
- embeddings are stored as a |V| x d matrix, where *d* is the dimensionality of the embeddings, such that the word assigned index *i* has its embedding stored in the *i*'th row of the matrix

Tuto on embeddings : https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html

Tuto on classif using embeddings:

https://pytorch.org/tutorials/beginner/text_sentiment_ngrams_tutorial.html

Embeddings in PyTorch

https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html

CLASS torch.nn.Embedding(*num_embeddings*, *embedding_dim*, *padding_idx=None*, *max_norm=None*, *norm_type=2.0*, *scale_grad_by_freq=False*, *sparse=False*, *_weight=None*, *device=None*, *dtype=None*)

https://pytorch.org/docs/stable/generated/torch.nn.EmbeddingBag.html

CLASS torch.nn.EmbeddingBag(*num_embeddings*, *embedding_dim*, *max_norm=None*, *norm_type=2.0*, *scale_grad_by_freq=False*, *mode='mean'*, *sparse=False*, *_weight=None*, *include_last_offset=False*, *padding_idx=None*, *device=None*, *dtype=None*)

⟶Computes sums or means of ‘bags’ of embeddings, without instantiating the intermediate embeddings.

Embeddings in PyTorch

Single words as input

```
class Model(...):

     def __init__(self, vocab_size,
embedding_dim, …):

          self.embeddings =
nn.Embedding(vocab_size, embedding_dim)
          . . .

     def forward(self, inputs):
          embeds = self.embeddings(inputs)
          . . .
```

e.g. Text classification

```
class Model(...):

     def __init__(self, vocab_size,
embed_dim, ...):

          self.embedding =
nn.EmbeddingBag(vocab_size, embedding_dim)
          . . .

     def forward(self, inputs):
          embeds = self.embeddings(inputs)
          . . .
```

Embeddings in PyTorch

look-up

https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html

CLASS torch.nn.Embedding(*num_embeddings*, *embedding_dim*, *padding_idx=None*, *max_norm=None*, *norm_type=2.0*, *scale_grad_by_freq=False*, *sparse=False*, *_weight=None*, *device=None*, *dtype=None*)

embedding layer

https://pytorch.org/docs/stable/generated/torch.nn.EmbeddingBag.html

CLASS torch.nn.EmbeddingBag(*num_embeddings*, *embedding_dim*, *max_norm=None*, *norm_type=2.0*, *scale_grad_by_freq=False*, *mode='mean'*, *sparse=False*, *_weight=None*, *include_last_offset=False*, *padding_idx=None*, *device=None*, *dtype=None*)

Summary: Data representation

⟶ Before NN: expertise needed to find good data representations

⟶ Now: feed your NN with word embeddings! but….

- Setting:
  - which ones? GloVe, FastText, Word2Vec, ELMO, BeRT, RobeRTa, GPT-2, GPT-3, XLNet...
  - which size, window size, number of iterations?
- Other issues:
  - how to combine them into a sentence / document?
  - what about other information: POS / syntax / pragmatics?
  - what about different languages and domains?
  - problem with evaluation: e.g. natural language inference tasks seem inadequate
  - choice of the data / problem with models: bias and representativeness

⟶ expertise still needed

Practical Session

- Generating word embeddings: Gensim (Word2vec)
- Computing word similarity based on their embeddings
- Making analogical reasoning
- Vizualizing Word embeddings

https://colab.research.google.com/drive/1-WYhZxrL-y06Jz0qj-yiOnRTlzfpV13a?usp=sharing

Sources

- https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314
- https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-cbow.html
- https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1
- https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/
- https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/
- https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
- https://courses.engr.illinois.edu/cs546/sp2020/Slides/Lecture04.pdf
- https://towardsdatascience.com/how-to-solve-analogies-with-word2vec-6ebaf2354009
- https://gunjanagicha.medium.com/word-embeddings-ee718cd2b8b5