Natural Language Processing

IFT6758 - Data Science

Sources:

http://demo.clab.cs.cmu.edu/NLP/

http://u.cs.biu.ac.il/~89-680/

And many more …

Announcements

- HM#1 grades are accessible in Gradescope!
- All the mid-term evaluation scores are on the scoreboard! 50 + (5 bonus points)
  - Task explanation: 2
  - Visualization: 3
  - Feature Engineering: 5
  - Approach: 15
  - Result: 10
  - Score on scoreboard: 15
  - Bonus point: 5
- Mid-term presentation are accessible in Gradescope!
- HM#2 grades will be out on Gradescope next week!
- Mid-term exam grades will be announced next week!
- Crash course on DL will be next Thursday!

Recall: Probabilistic Language Models

- **Goal**: Compute the probability of a sentence or sequences of words

$$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$$

- Related task: probability of an upcoming word:

$$P(w_5 | w_1,w_2,w_3,w_4)$$

- A model that computes either of the above is called a **language model**.

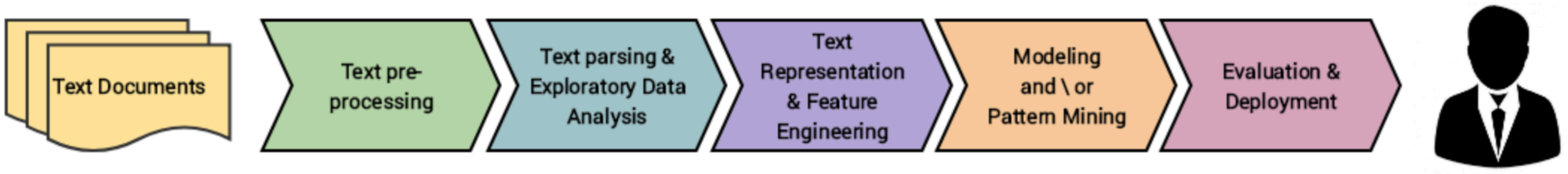
Recall: Parametric Language Model

~~Non-parametric estimator~~ $\longrightarrow$ parametric estimator

$$P\left(x_t | x_{t-n}, \ldots, x_{t-1}\right) = \xcancel{\frac{\text{count}\left(x_{t-n}, \ldots, x_t\right)}{\text{count}\left(x_{t-1}, \ldots, x_{t-1}\right)}}$$
$$= \boxed{f_\Theta\left(x_{t-n}, \ldots, x_{t-1}\right)}$$

Somehow, we need numerical representation for words... i.e. **Word vectors**

Representation

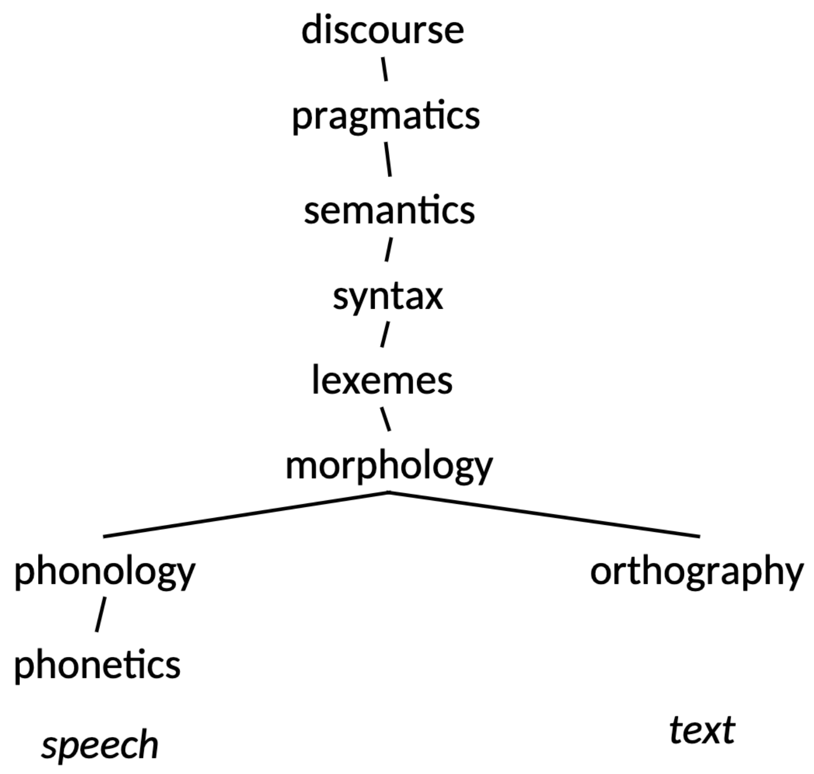

Text Documents
Text pre-processing
Text parsing & Exploratory Data Analysis
Text Representation & Feature Engineering
Modeling and \ or Pattern Mining
Evaluation & Deployment

A high-level standard workflow for any NLP project

- We can represent objects in different hierarchy levels:
  - Documents
  - Sentences
  - Phrases
  - Words

- We want the representation to be interpretable and easy-to-use

- **Vector representation** meets those requirements



Classic NLP word representation

- **Problems** with classic vector representation:
    - **Huge** – each of dimension |V| (the size of the vocabulary ~)
    - **Sparse** – most entries will be 0
- We want our vectors to be small and dense:

~~[0 1 0 0 0 0 0 0 0]~~ [0.315 0.136 0.831]

- **Similar words have similar vectors:** Capture semantic and morphologic similarity so that the features for "similar" words are "similar" (= their vectors are close to each other in the vector space)

Learning Dense embeddings

Matrix Factorization

Factorize word-context matrix.

	Context$_1$	Context$_1$	....	Context$_k$
Word$_1$				
Word$_2$				
⋮				
Word$_n$				

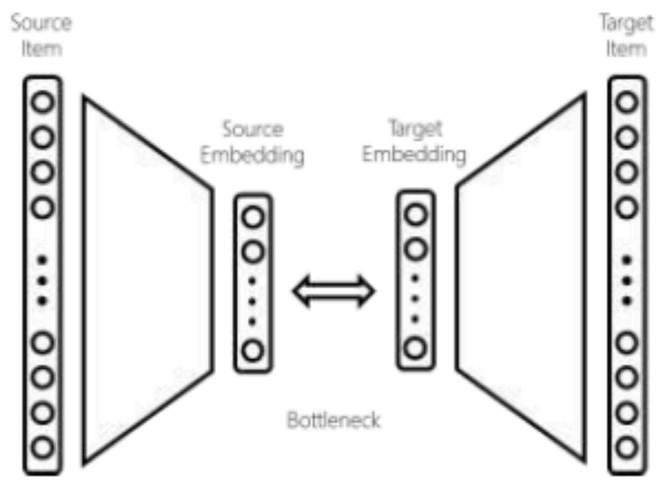
E.g.,

LDA (Word-Document),

GloVe (Word-NeighboringWord)

Neural Networks

A neural network with a bottleneck, word and context as input and output respectively.



E.g.,

Word2vec (Word-NeighboringWord)

Deerwester, Dumais, Landauer, Furnas, and Harshman, Indexing by latent semantic analysis, JASIS, 1990.
Pennington, Socher, and Manning, GloVe: Global Vectors for Word Representation, EMNLP, 2014.
Mikolov, Sutskever, Chen, Corrado, and Dean, Distributed representations of words and phrases and their compositionality, NIPS, 2013.

Word Embedding via SVD

GloVe

Word2Vec

Two representations

We represent how often a word occurs in a document

- **Term-document matrix**

Or how often a word occurs with another

- **Term-term matrix**

  (or **word-word co-occurrence matrix**

  or **word-context matrix**)

Term-Document Matrix

Documents

Words

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

- This matrix is the basis for computing the **similarity** between documents and queries
- This representation is used in information retrieval to calculate similarity between queries and documents

Term-Document Matrix

- We will decompose the term-document matrix into a product of matrices.

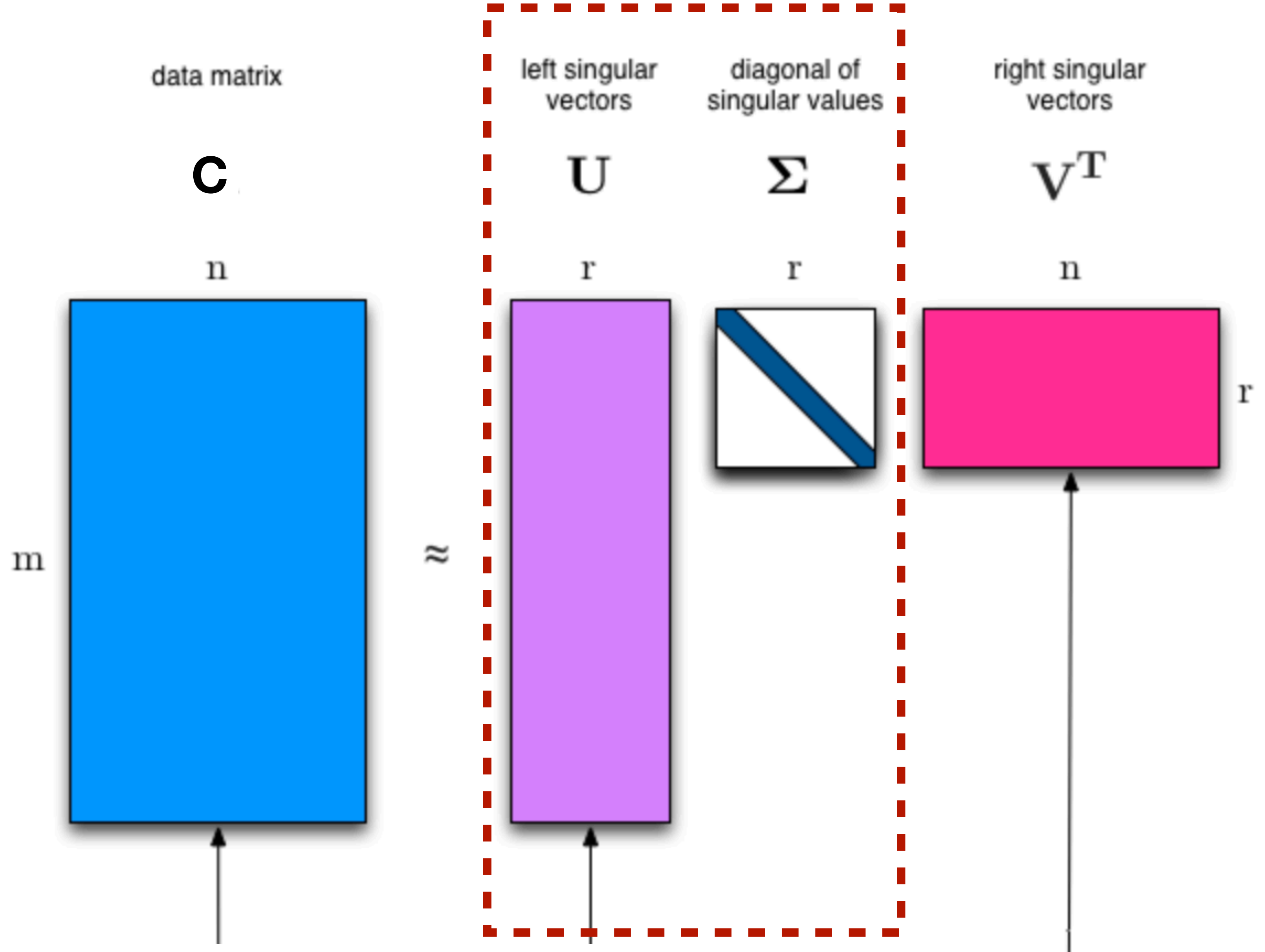
$$\begin{array}{ccccc} \text{data matrix} & & \text{left singular vectors} & \text{diagonal of singular values} & \text{right singular vectors} \\ \mathbf{C} & = & \mathbf{U} & \mathbf{\Sigma} & \mathbf{V}^{\mathrm{T}} \end{array}$$

- The particular decomposition we'll use: **singular value decomposition** (SVD).

- We will then use the SVD to compute a new, improved term-document matrix C′.

- We'll get better similarity values out of C′(compared to C).

Singular Value Decomposition

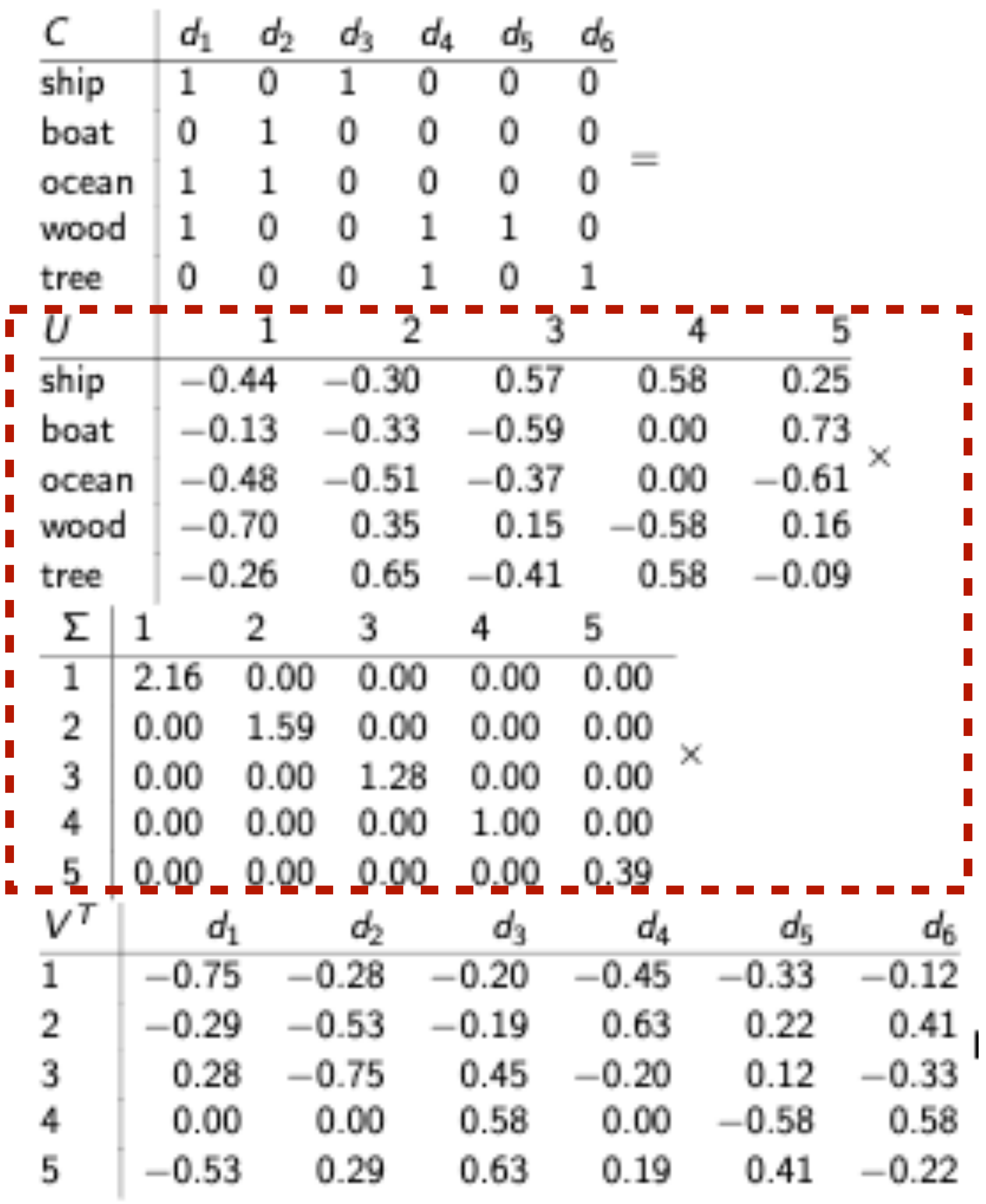


Word-document matrix

Word embedding matrix

Embedding of Documents

SVD decomposition



C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

=

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

×

Σ	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	1.28	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00
5	0.00	0.00	0.00	0.00	0.39

×

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

Word embedding matrix

Measuring similarity

Given 2 target words *v* and *w*

We'll need a way to measure their similarity.

Most measure of vectors similarity are based on the:

Dot product or **inner product** from linear algebra

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1w_1 + v_2w_2 + \ldots + v_Nw_N$$

- High when two vectors have large values in same dimensions.
- Low (in fact 0) for **orthogonal vectors** with zeros in complementary distribution

Similarity is preserved

$$
\begin{array}{ccccc}
\text{data matrix} & & \text{left singular vectors} & \text{diagonal of singular values} & \text{right singular vectors} \\
\mathbf{C} & = & \mathbf{U} & \mathbf{\Sigma} & \mathbf{V}^{\mathrm{T}}
\end{array}
$$

- Given a term-document matrix C, we can get a decomposition C' from SVD.
- The singular value decomposition will break C into best rank approximation C'.

$$
\begin{aligned}
C'C'^T &= (U\Sigma V^T)(U\Sigma V^T)^T \\
&= (U\Sigma V^T)(V\Sigma U^T) \\
&= U\Sigma\Sigma^T U^T \quad (\because V^T V = I) \\
&= U\Sigma(U\Sigma)^T
\end{aligned}
$$

Word embedding via SVD

- **Pros**:
    - Dense vector representation
    - Preserves similarity between words
- **Cons**:
    - The purpose of SVD is dimension reduction and it is not designed to learn the word embeddings
    - Computationally expensive

Word Embedding via SVD

GloVe

Word2Vec

Global Vectors for Word Representation (GloVe)

- Glove is based on **matrix factorization** techniques on the word-context matrix.

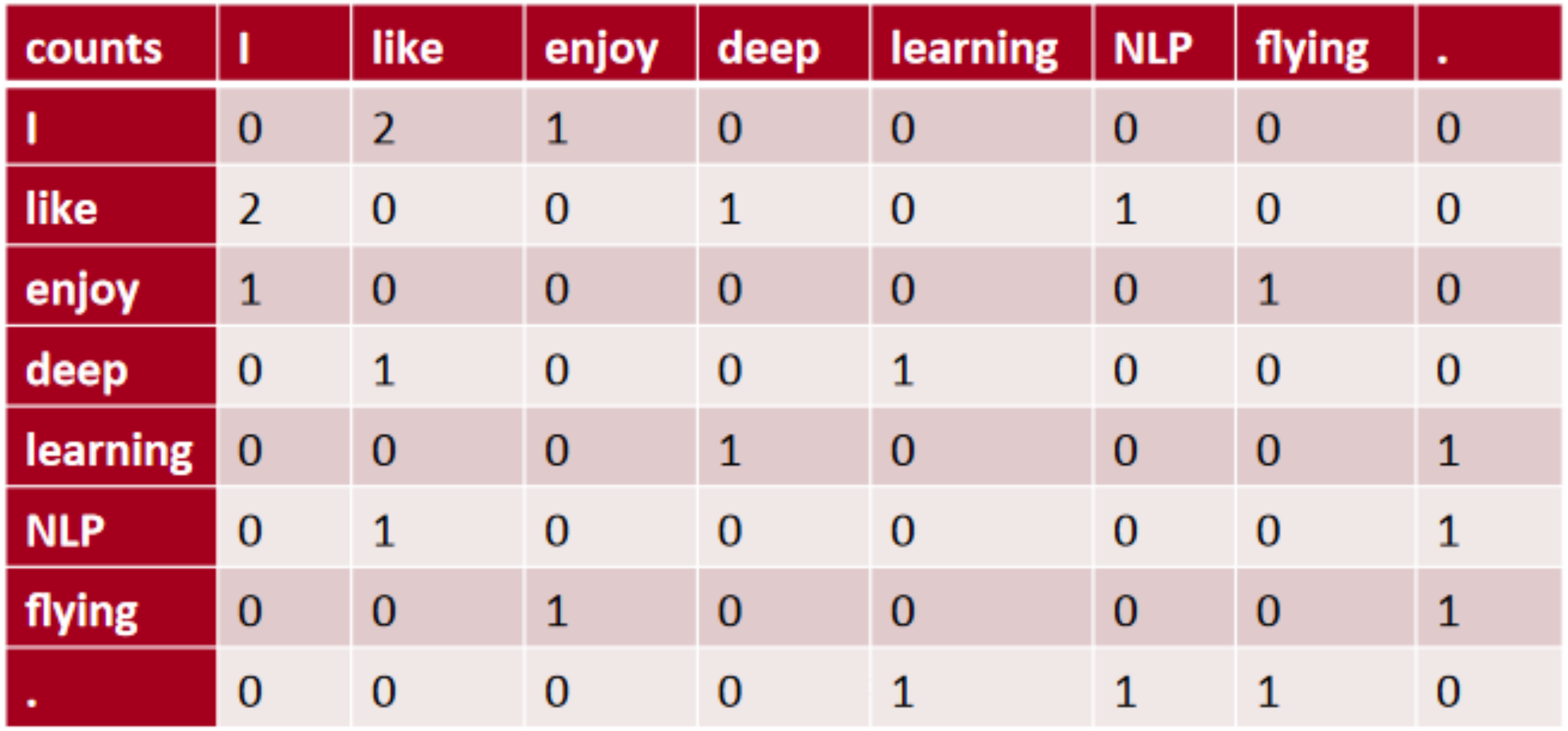
- It first constructs a large matrix of (words x context) co-occurrence information, i.e. for each “word” (the rows), you count how frequently we see this word in some “context” (the columns) in a large corpus.

- Uses **ratios of co-occurrence probabilities**, rather than the co-occurrence probabilities themselves.

Building a co-occurrence matrix

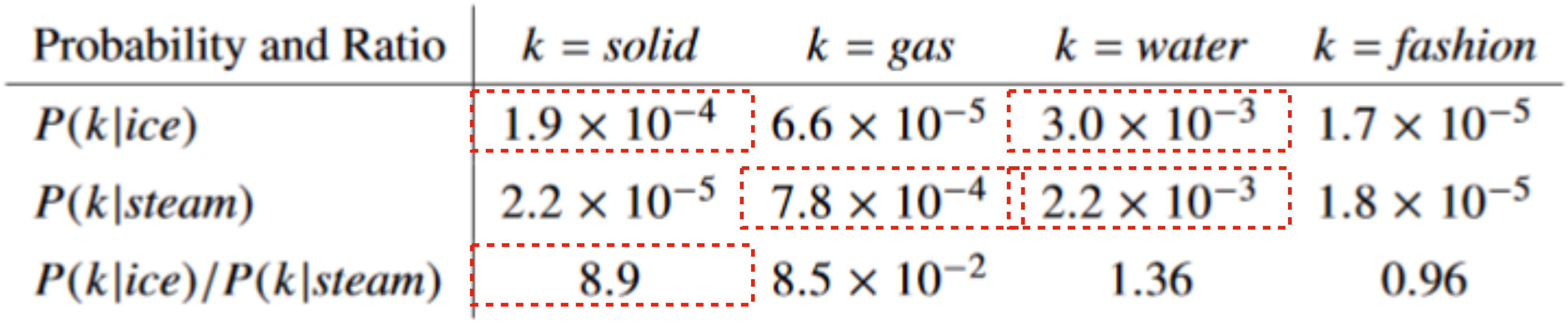
Corpus = {“I like deep learning”
“I like NLP”
“I enjoy flying”}

Context = previous word and next word



counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Intuition



Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Selected co-occurrence probabilities from a 6 billion token corpus

GloVe: Formal definition

- Let’s formulate this idea mathematically and then develop an intuition for it. Let X*ij* encodes important global information about the co-occurrence between *i* and *j.*

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{X_i}$$
$$X_{ij} = X_{ji}$$

- Our aim will be to learn word vectors which complies with computed probability on entire corpus. Essentially we are saying that we want word vectors **v***i* and **v***j* such that is faithful to the globally computed P (j|i).

$$\begin{aligned} v_i^T v_j &= \log P(j|i) \\ &= \log X_{ij} - \log(X_i) \end{aligned}$$

GloVe: Formal definition

- log(**X***i*) and log(**X***j*) depend only on the words *i* & *j* and we can think of them as word specific biases which will be learned. Formulate this problem in following way.

$$v_i^T v_j = \log X_{ij} - b_i - b_j$$

$$v_i^T v_j + b_i + b_j = \log X_{ij}$$

$$\min_{v_i, v_j, b_i, b_j} \sum_{i,j} (\underbrace{v_i^T v_j + b_i + b_j}_{\substack{\text{predicted value}\\ \text{using model}\\ \text{parameters}}} - \underbrace{\log X_{ij}}_{\substack{\text{actual value}\\ \text{computed from}\\ \text{the given corpus}}})^2$$

GloVe: Formal definition

- log(**X***i*) and log(**X***j*) depend only on the words *i* & *j* and we can think of them as word specific biases which will be learned. Formulate this problem in following way.

$$v_i^T v_j = \log X_{ij} - b_i - b_j$$

$$v_i^T v_j + b_i + b_j = \log X_{ij}$$

$$\min_{v_i, v_j, b_i, b_j} \sum_{i,j} (\underbrace{v_i^T v_j + b_i + b_j}_{\substack{\text{predicted value} \\ \text{using model} \\ \text{parameters}}} - \underbrace{\log X_{ij}}_{\substack{\text{actual value} \\ \text{computed from} \\ \text{the given corpus}}})^2$$

weights of all the co-occurrences are equal?

Weight should be defined in such a manner that neither rare or frequent words are over-weighted.

GloVe: Formal definition

- log(**X***i*) and log(**X***j*) depend only on the words *i* & *j* and we can think of them as word specific biases which will be learned. Formulate this problem in following way.

$$v_i^T v_j = \log X_{ij} - b_i - b_j$$

$$v_i^T v_j + b_i + b_j = \log X_{ij}$$

$$\min_{v_i, v_j, b_i, b_j} \sum_{i,j} \boxed{f(X_{ij})} (v_i^T v_j + b_i + b_j - \log X_{ij})^2$$

$$\Updownarrow$$

$$f(x) = \begin{cases} (\frac{x}{x_{max}})^\alpha, & \text{if } x < x_{max} \\ 1, & \text{otherwise} \end{cases}$$

GloVe

- **Pros**:
    - Provide dense vector
    - Easier/cheaper than SVD to compute
    - Extract semantic similarity between words
    - Pre-trained models are available online.
- **Cons**:
    - Computationally heavy and requires a lot of memory

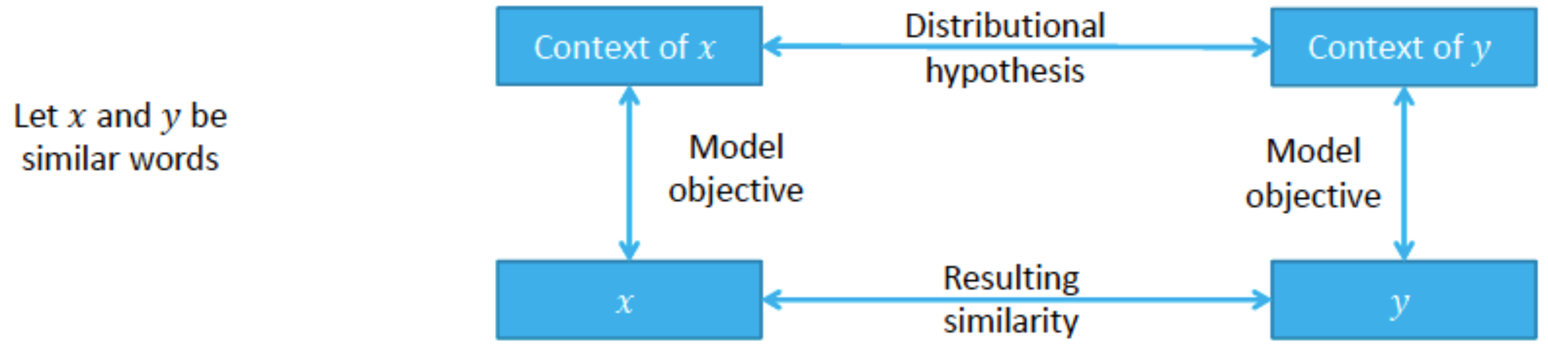
Word Embedding via SVD

GloVe

Word2Vec

Word2Vec

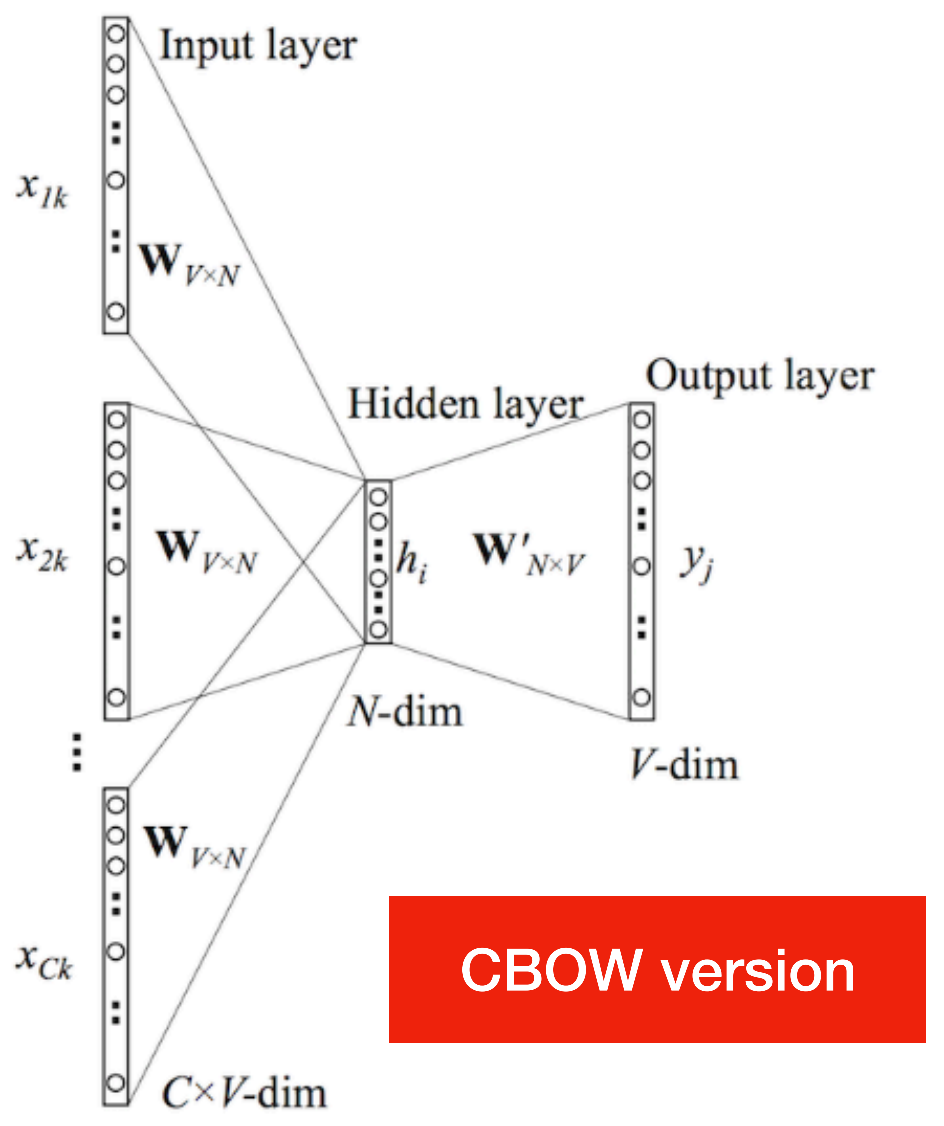
- Models for efficiently creating word embeddings
- Remember: our assumption is that **similar words appear with similar context**
- Intuition: two words that share similar contexts are associated with vectors that are close to each other in the vector space

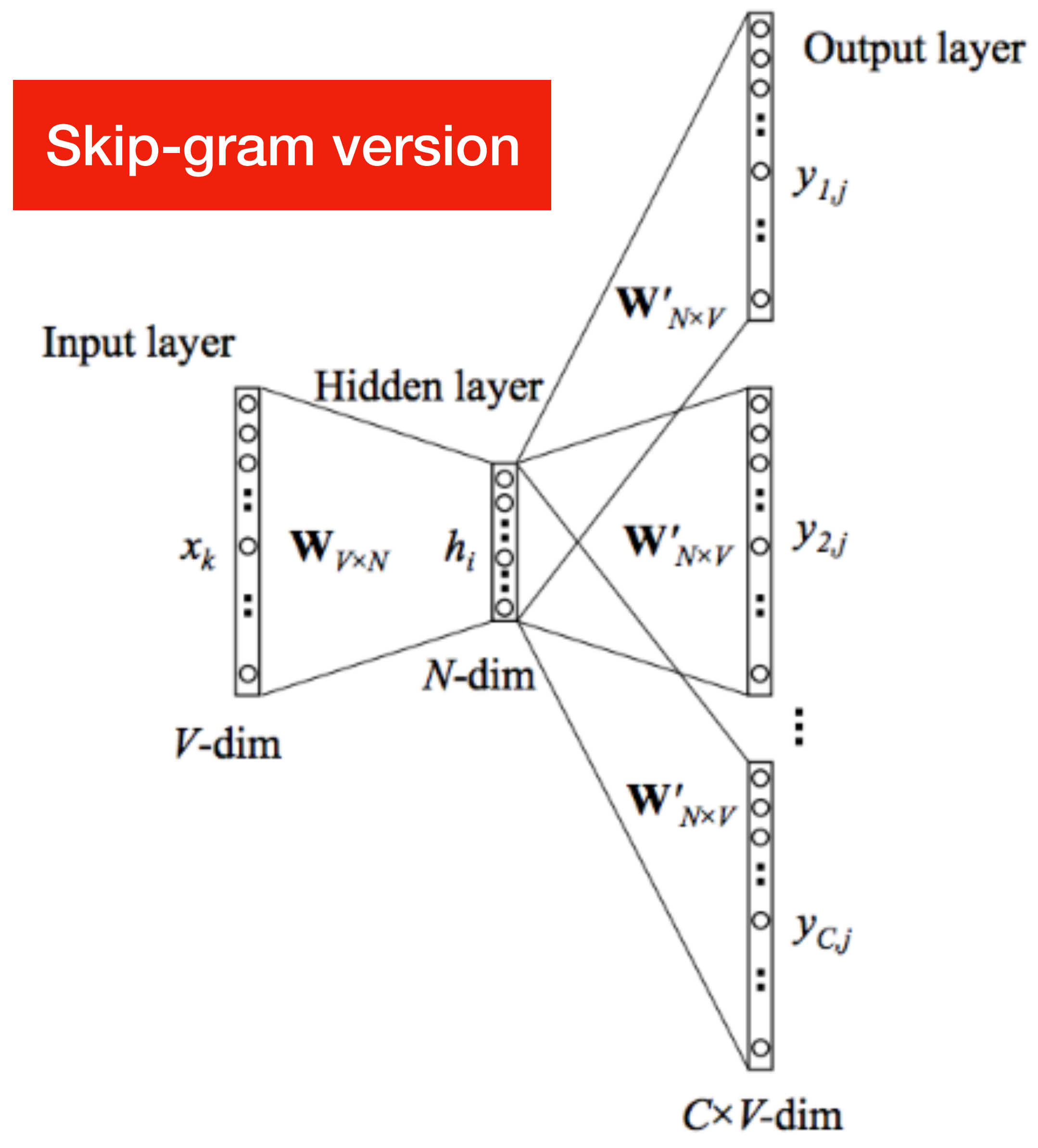


Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean, 2013. Distributed representations of words and phrases and theircompositionality. In Advances in neural information processing systems.

Word2Vec Embedding methods

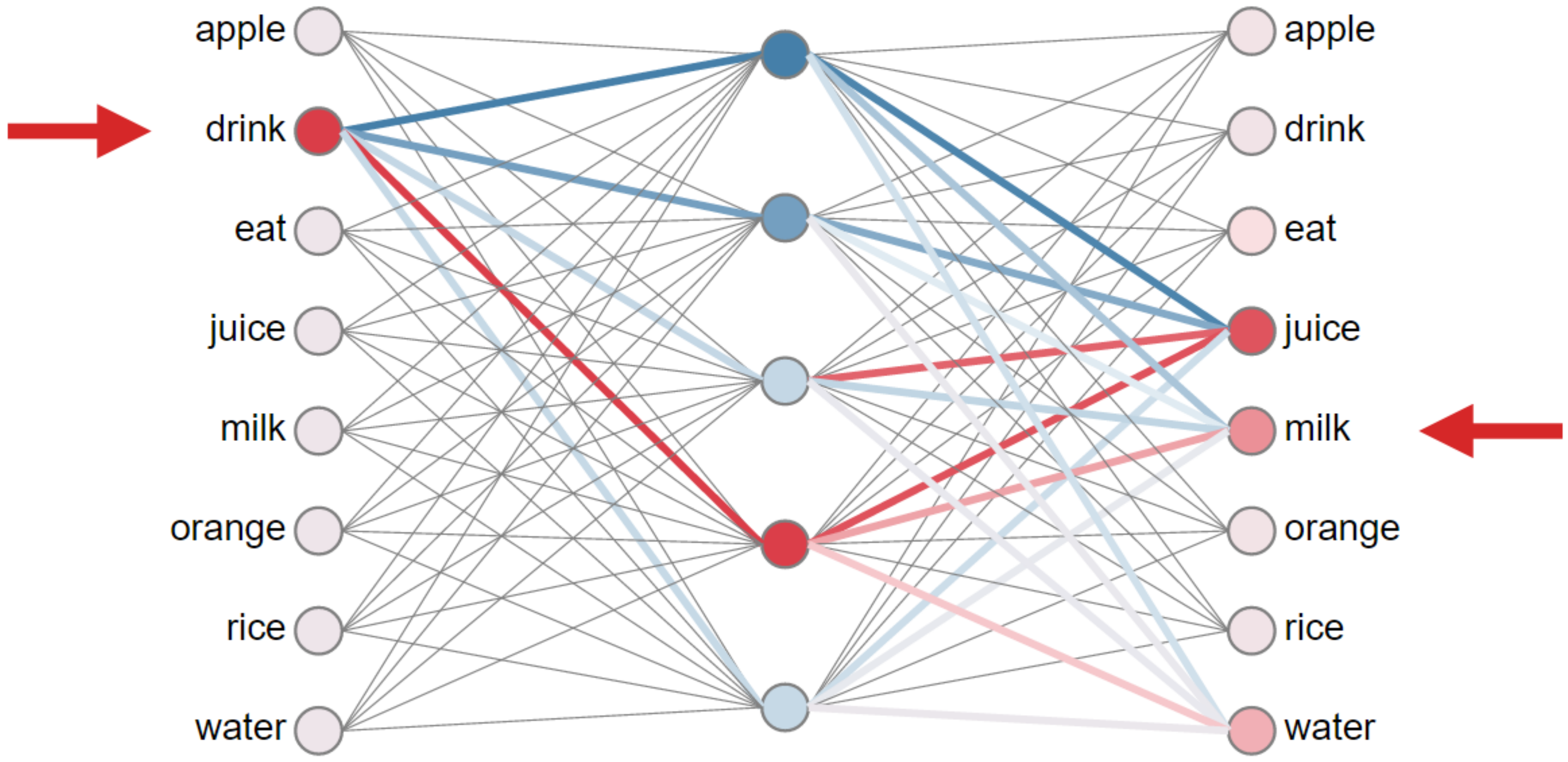




Mila

Université de Montréal

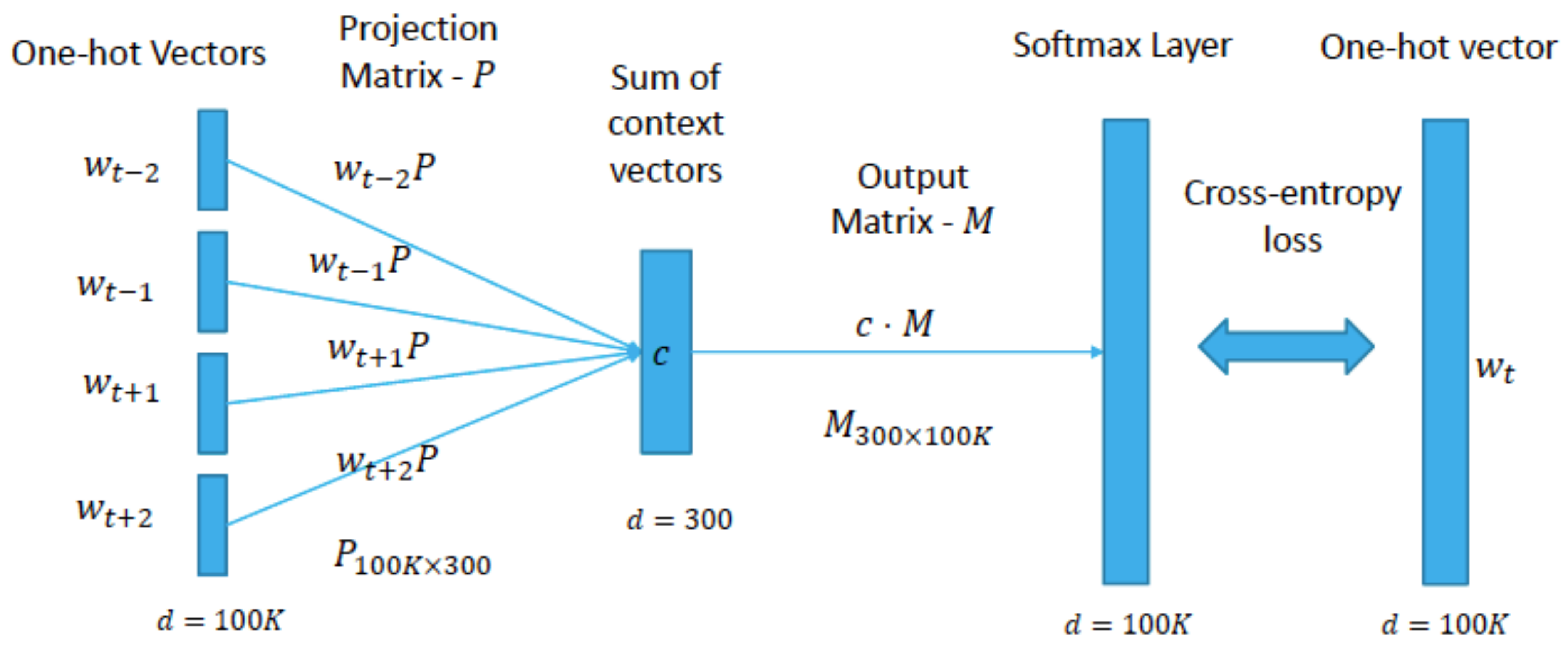
Main Goal


apple
drink
eat
juice
milk
orange
rice
water
apple
drink
eat
juice
milk
orange
rice
water

CBOW : Continuous bag-of-word (high level)

- Goal: Predict the middle **word** given the **words of the context**

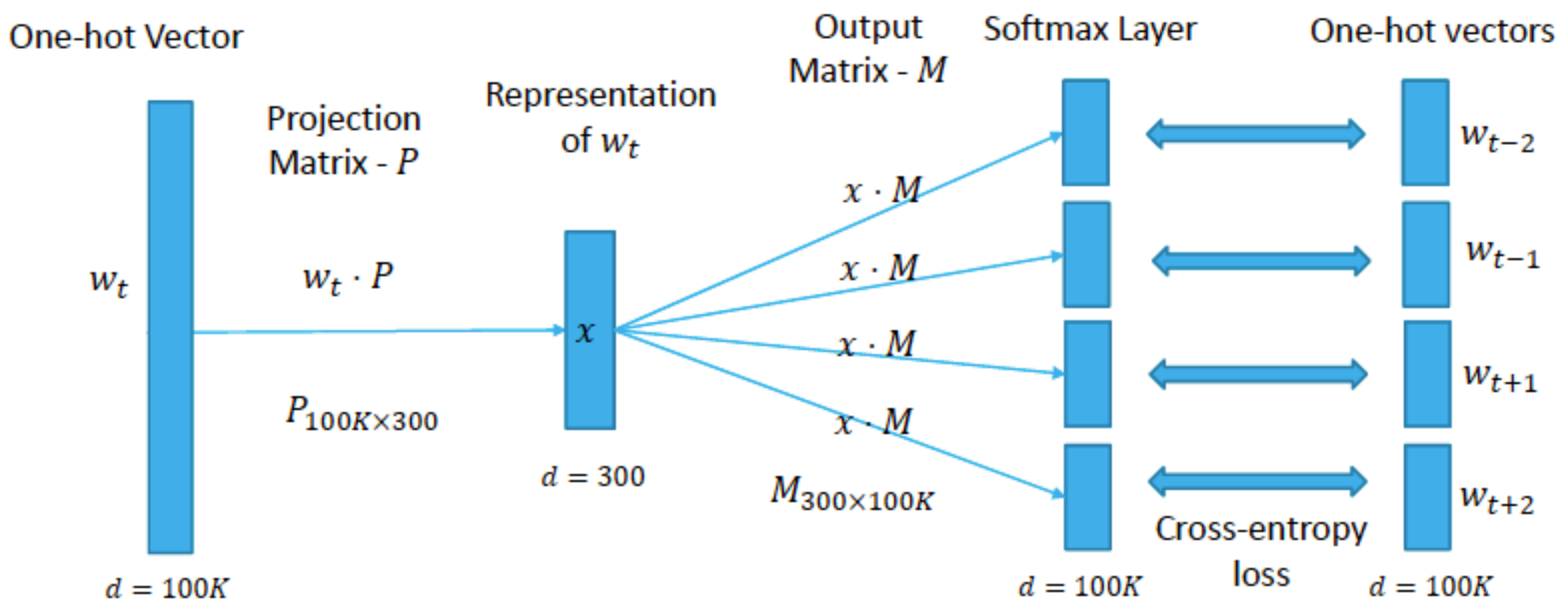
The resulting projection matrix P is the embedding matrix



Skip-gram (high level)

- Goal: Predict the **context words** given the middle **word**

The resulting projection matrix P is the embedding matrix



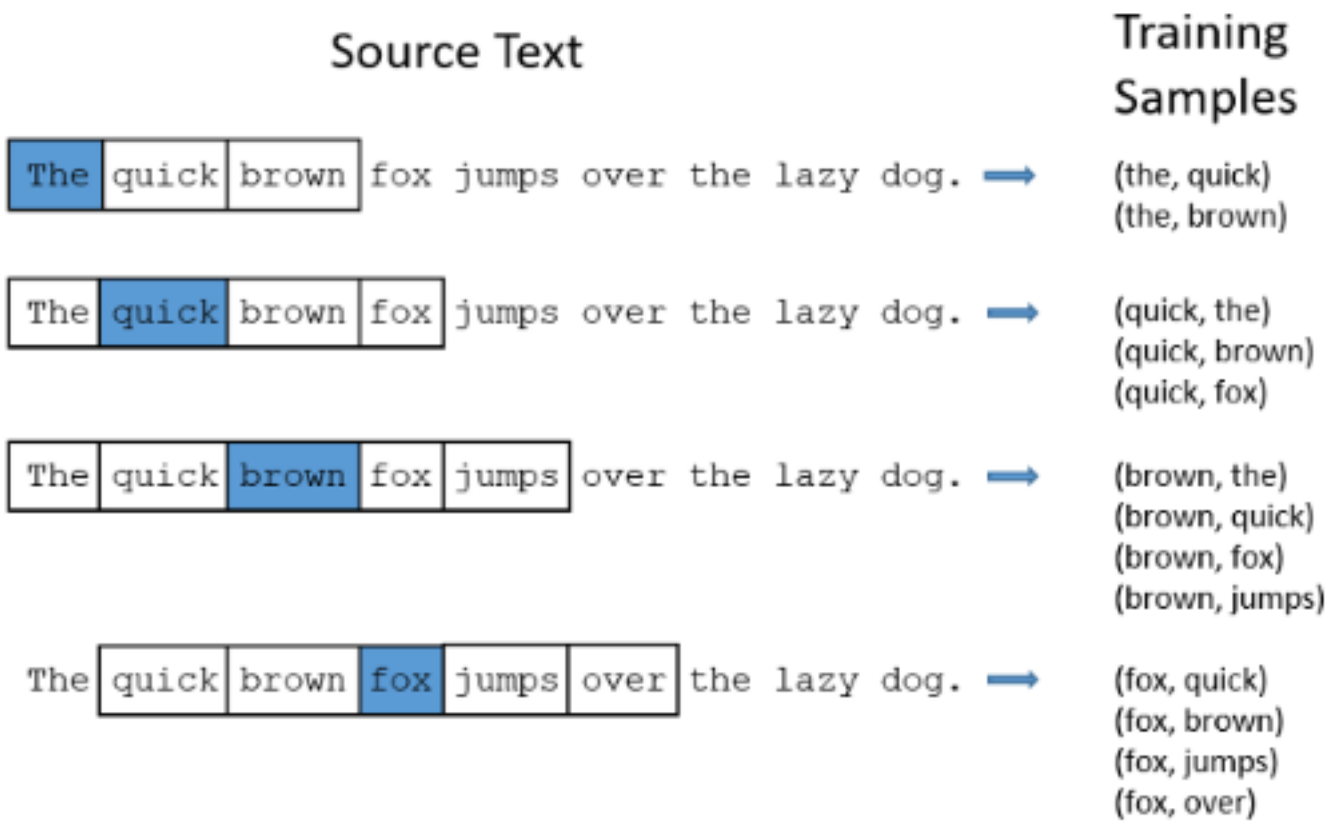
Mila

Université de Montréal

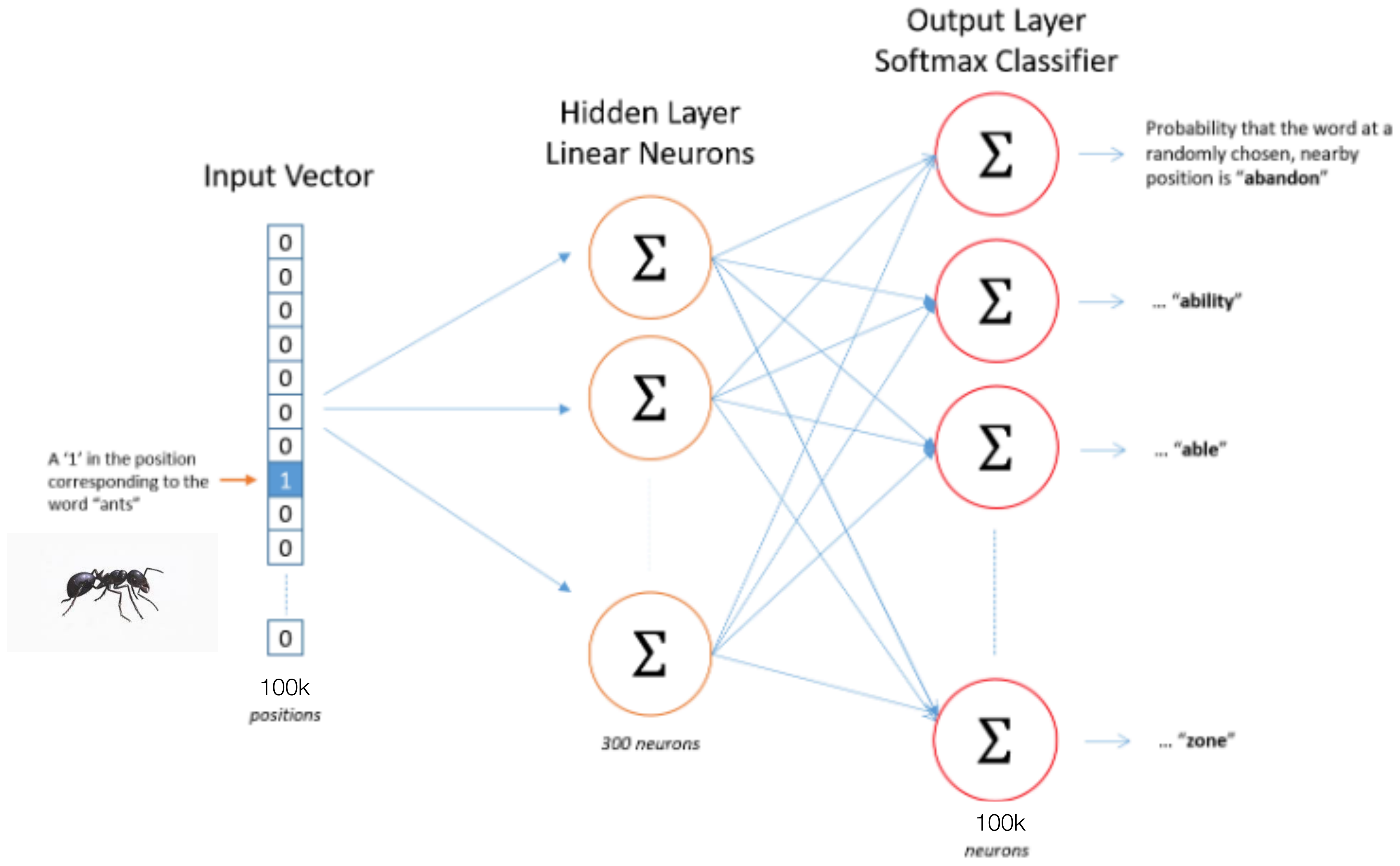
Training data

- Making context and target word pairs depends on the **window size** you take.
- To make the pairs, you need to look to the left and right of the context word for as many as window size words.

e.g.,
window size = 2

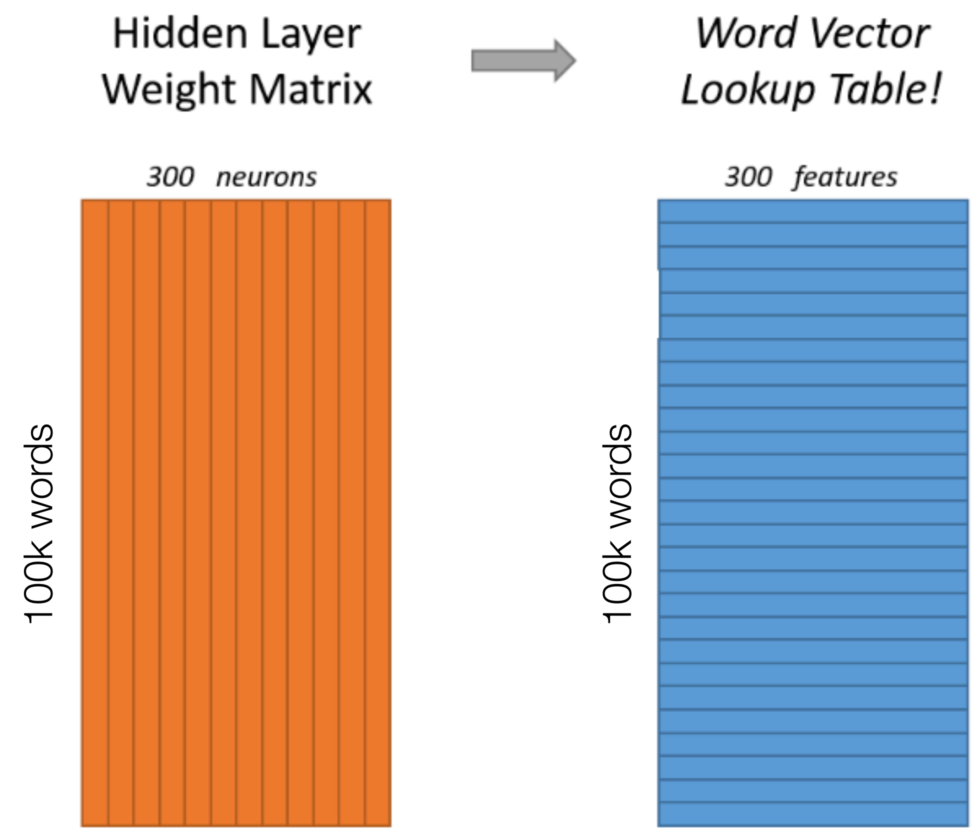


Architecture



Extract Word embedding

The rows of the hidden layer weight/projection matrix, are actually the word vectors (word embeddings).



$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

One-hot-vector of a word

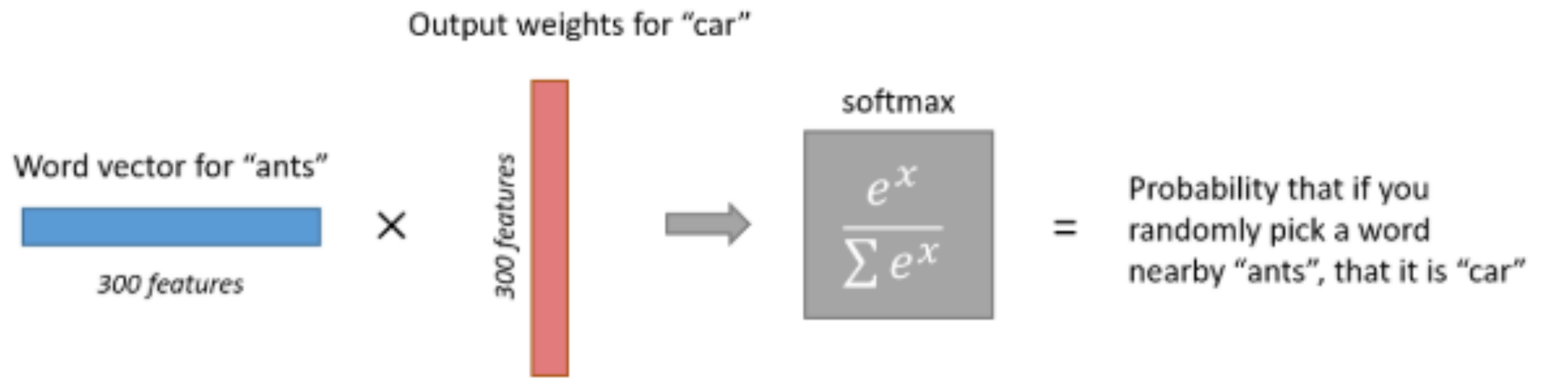
Hidden layer

Word embedding

If you multiply a 1 x 100,000 one-hot vector by a 100,000 x 300 matrix, it will effectively just select the matrix row corresponding to the '1'.

How to works?

- This is semi-supervised learning because we don’t have the direct labels associated with the words but we use the neighboring words (of a context word in a sentence) as the labels. How it works?



- **Softmax activation function**: is a function that takes as input a vector of *K* real numbers, and normalizes it into a probability distribution consisting of *K* probabilities proportional to the exponentials of the input numbers

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

Mila

Université de Montréal

Skip-gram: Formal definition

- Vector representations will be useful for predicting the **surrounding words.**
- Formally: Given a sequence of training words $w_1, w_2, ...w_T$ the objective of the Skip-gram model is to **maximize** the **average log probability**:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

- The basic Skip-gram formulation defines $p(w_{t+j}|w_t)$ using the **softmax** function:

$$p(w_{t+j}|w_t) = \frac{\exp(v'_{w_{t+j}} v_{w_t})}{\sum_{i=1}^{T}\exp(v'_{w_i} v_{w_t})}$$

v - input vector representations
v' - output vector representations

Negative Sampling (Technical details)

- Recall that for Skip-gram we want to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

- Which is equivalent to **minimizing the cross-entropy loss**:

$$L = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) = -\frac{1}{T}\boxed{\sum_{t=1}^{T}}\sum_{-c\leq j\leq c, j\neq 0}\log\frac{\exp(v'_{w_{t+j}}v_{w_t})}{\sum_{i=1}^{T}\exp(v'_{w_i}v_{w_t})}$$

- This is extremely **computational-expensive**, as we need to update all the parameters of the model for each training example…

Negative Sampling (Technical details)

- When looking at the loss obtained from a single training example, we get:

$$-\log p(w_{t+j}|w_t) = -\log \frac{\exp(v'_{w_{t+j}} v_{w_t})}{\sum_{i=1}^{T} \exp(v'_{w_i} v_{w_t})} = -\underbrace{v'_{w_{t+j}} v_{w_t}}_{\text{“positive” pair}} + \log \sum_{i=1}^{T} \exp\underbrace{(v'_{w_i} v_{w_t})}_{\text{“negative” pair}}$$

- When using negative sampling, instead of going through all the words in the vocabulary for negative pairs, we sample a modest amount of k words (around 5-20). The exact objective used:

$$\log \sigma(v'_{w_{t+j}} v_{w_t}) + \boxed{\sum_{1=1}^{k}} \log \sigma(-v'_{w_i} v_{w_t}) \longrightarrow \text{Replaces the term: } \log p(w_{t+j}|w_t) \text{ for each word in the training}$$

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

Context Sampling (Technical details)

- We want to give more weight to words closer to our target word
- For a given window size C, we sample R in [1, C] and try to predict only R words before and after our target word
- For each word in the training we need to perform 2*R word classifications (R is not fixed)

Subsampling of Frequent Words (Technical details)

- In order to eliminate the negative effect of very frequent words such as “in”, “the” etc. (that are usually not informative), a simple subsampling approach is used.

- Each word w in the training set is discarded with probability:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n}(f(w_j)^{3/4})}$$

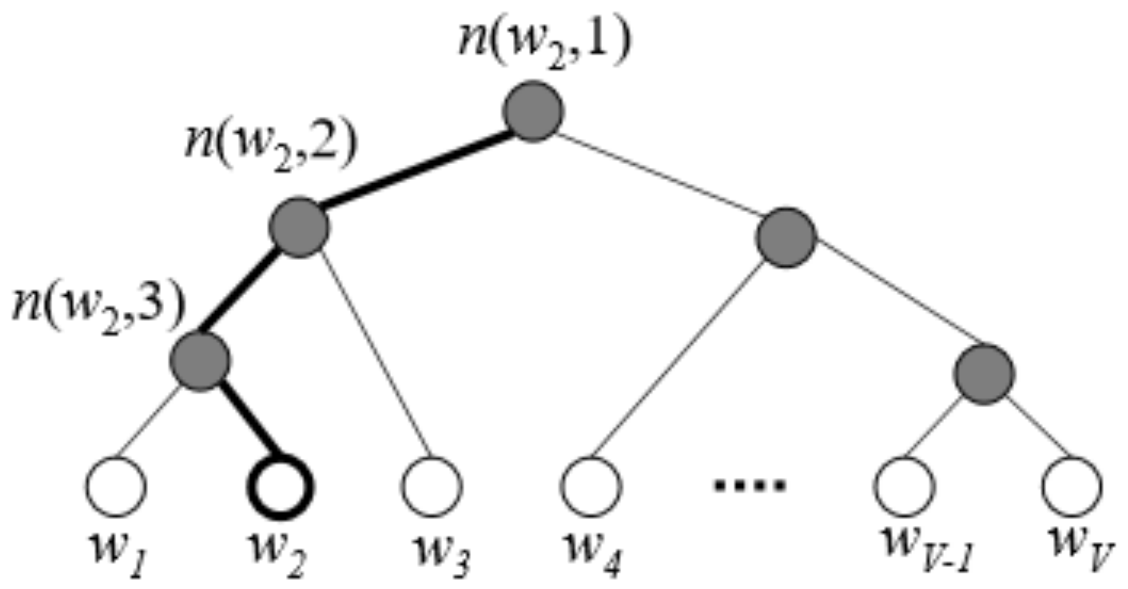
where *3/4* is the value found by taking experiments; *f(w)* is the frequency of the word in the corpus. This way frequent words are discarded more often.

- This method improves the training speed and makes the word representations significantly more accurate

Hierarchical softmax (Technical details)

- The size of the vocabulary can be quite large — datasets vocabulary size ranges between 30K, 82K, and 1M — which, iff implemented in the naive way, causes the output layer to become a bottleneck.

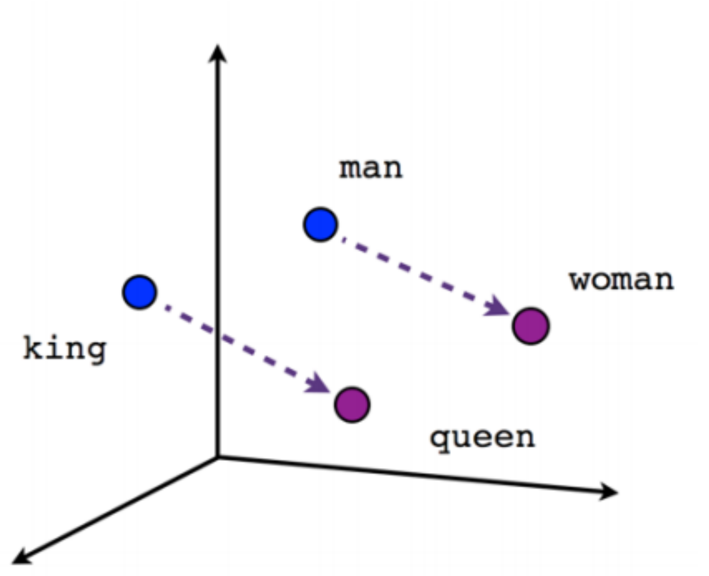
- So one option is to use **hierarchical softmax**, representing the vocabulary as a Huffman binary tree (more common words being closer to the root), which reduces the complexity to *O(log(V))*.



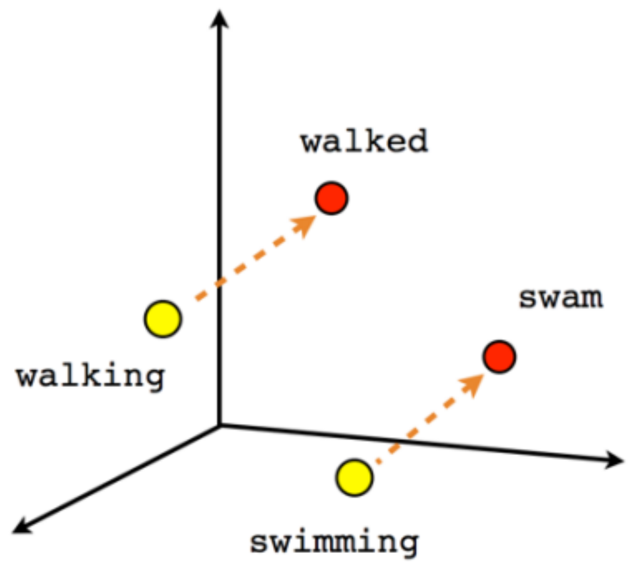
- There exists a unique path from the root node to a leaf node.

- In effect, the task gets formulated to the probability of predicting a word is the same as predicting the correct unique path from the root node to that word.

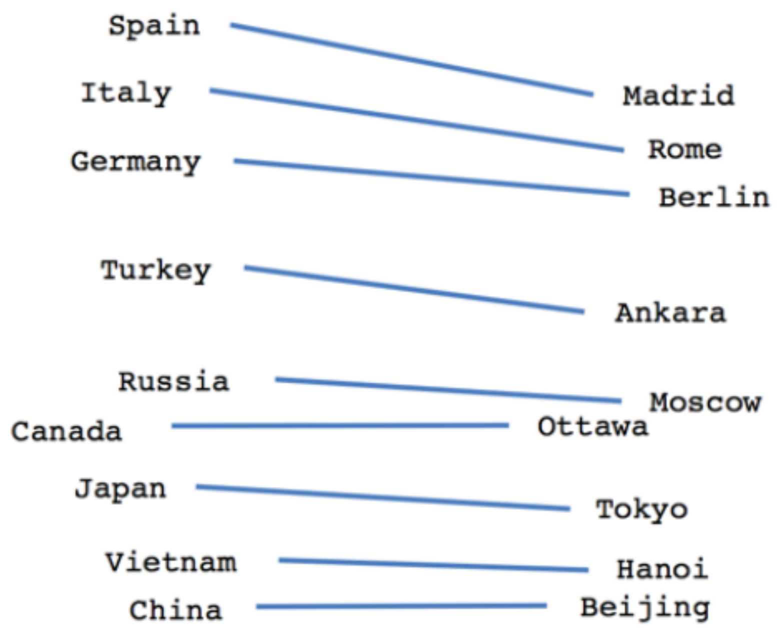
Regularities



Male-Female



Verb tense



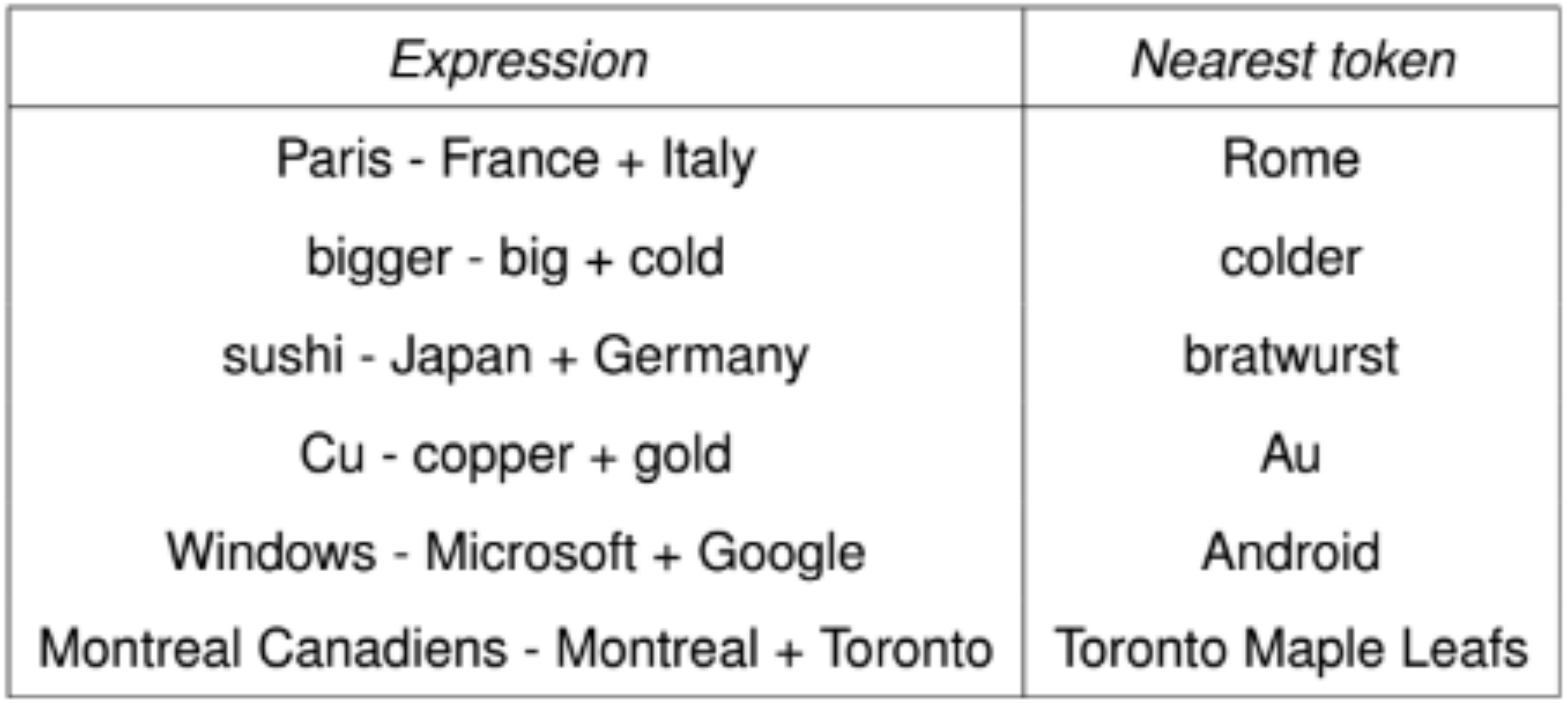
Country-Capital

vector[Queen] = vector[King] - vector[Man] + vector[Woman]

Mila

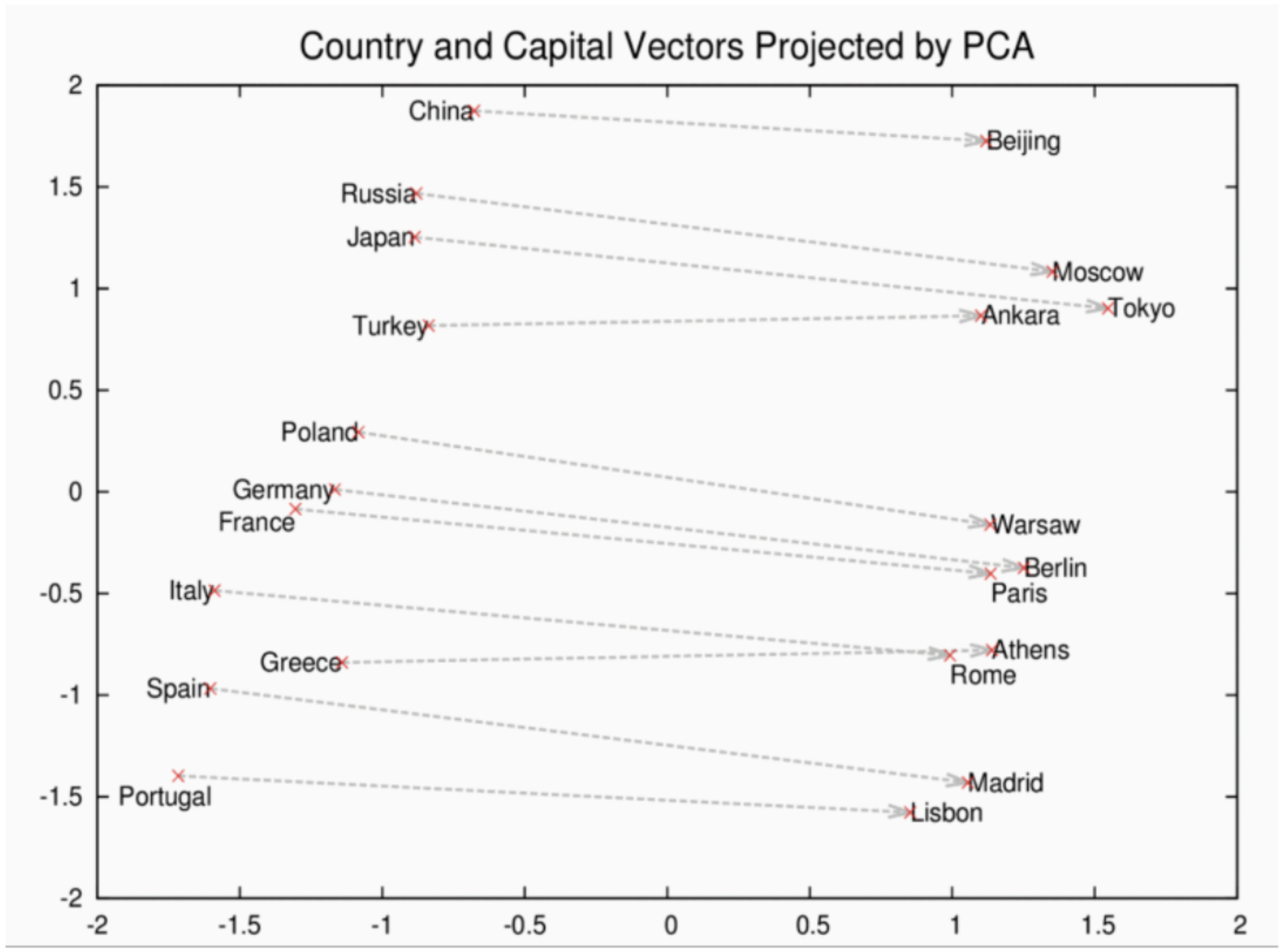
Université de Montréal

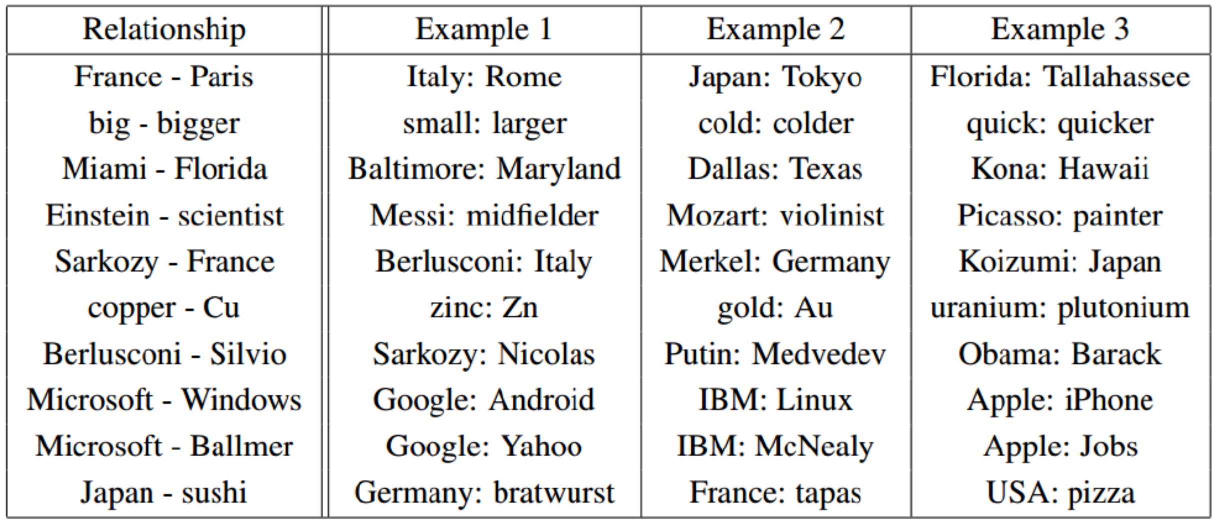
Regularities



Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Visualization in word space





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

- **Pros**:
  - Provide dense vector
  - Faster to train than GolVe
  - Extract semantic similarity between words
  - Pre-trained models are available online.
- **Cons**:
  - Cannot capture global context
  - It is not clear how to represent a sentence/document with word2vec

Summary

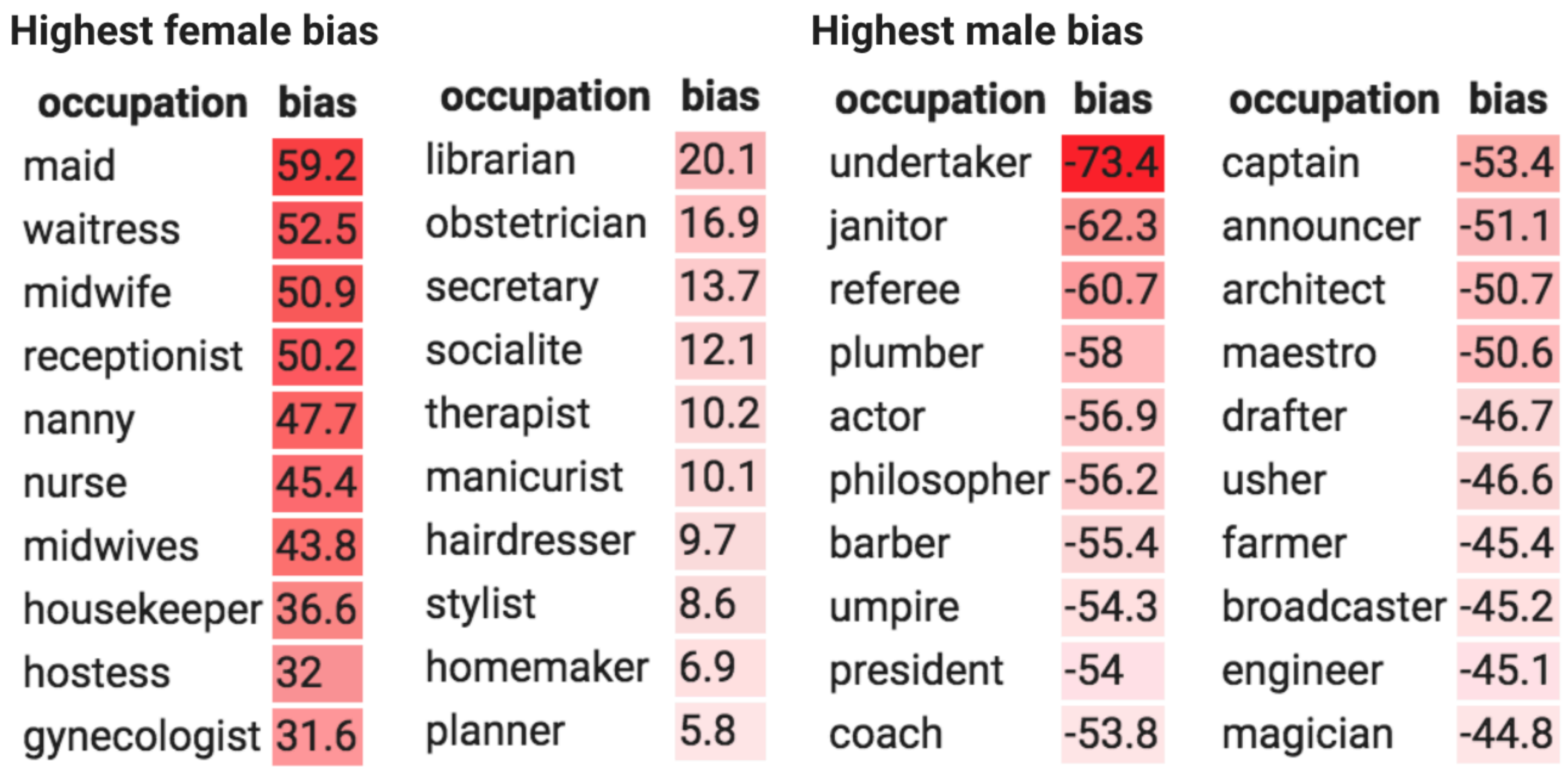
- Skip Gram works well with small amount of data and is found to represent rare words well. On the other hand, CBOW is faster and has better representations for more frequent words.
- GloVe uses a lot of memory but unlike Word2Vec consider global context as well as local context.
- GloVe optimizes directly so that the dot product of two word vectors equals the log of the number of times the two words will occur near each other.
- Word2Vec extracts the embeddings from a neural network that is designed to perform a surrogate task (predicting neighbouring words)
- GloVe and Word2Vec outperform each other on various tasks and there is no clear winner in terms of performance. Probably depends on the data/task.
- You can train your own word vector based on your own corpus using both approaches.

Word Embedding methods

- **Benefits**:
  - **Learns features** of each word on its own, given a text corpus.
  - **No heavy preprocessing** is required, just a corpus.
  - **Word vectors** can be used as **features** for lots of supervised
  - learning applications: POS, named entity recognition (NER), chunking, semantic role labeling with pretty much the **same network architecture**
  - Captures similarities and linear **relationships** between word vectors.

Bias in embeddings

The occupations with the highest female-biased scores (left) and the highest male-biased scores (right):



Highest female bias

occupation	bias	occupation	bias
maid	59.2	librarian	20.1
waitress	52.5	obstetrician	16.9
midwife	50.9	secretary	13.7
receptionist	50.2	socialite	12.1
nanny	47.7	therapist	10.2
nurse	45.4	manicurist	10.1
midwives	43.8	hairdresser	9.7
housekeeper	36.6	stylist	8.6
hostess	32	homemaker	6.9
gynecologist	31.6	planner	5.8

Highest male bias

occupation	bias	occupation	bias
undertaker	-73.4	captain	-53.4
janitor	-62.3	announcer	-51.1
referee	-60.7	architect	-50.7
plumber	-58	maestro	-50.6
actor	-56.9	drafter	-46.7
philosopher	-56.2	usher	-46.6
barber	-55.4	farmer	-45.4
umpire	-54.3	broadcaster	-45.2
president	-54	engineer	-45.1
coach	-53.8	magician	-44.8

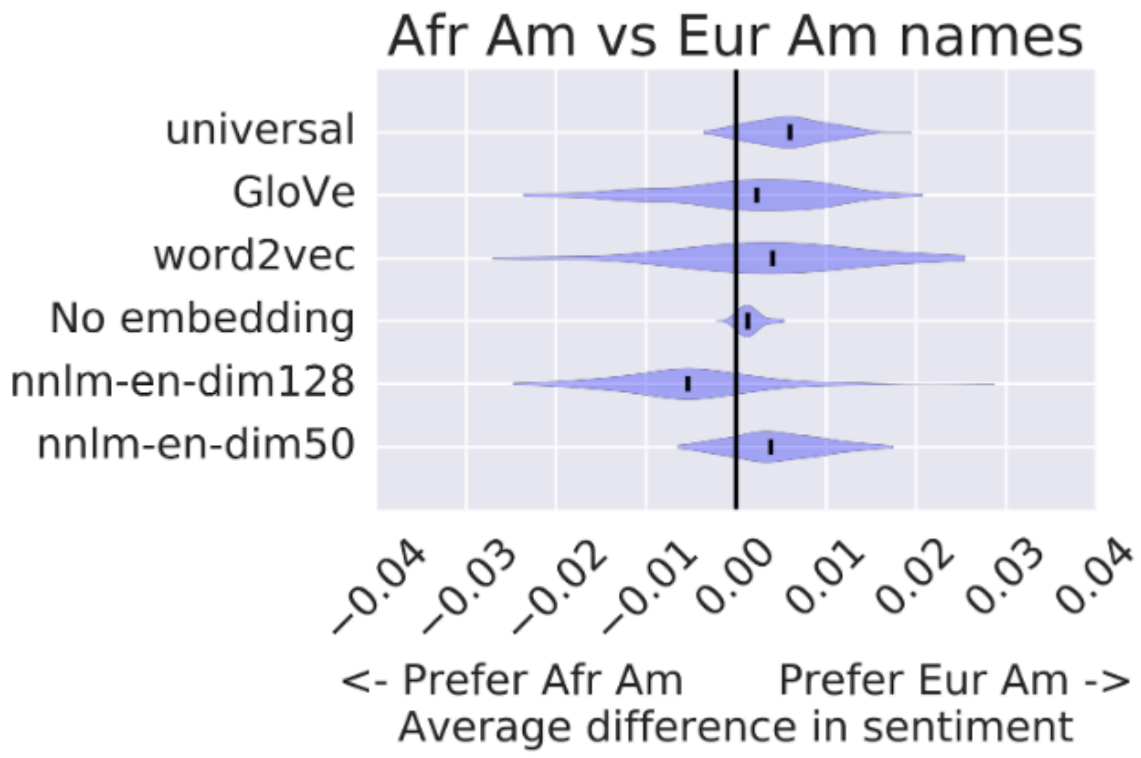
https://developers.googleblog.com/2018/04/text-embedding-models-contain-bias.html

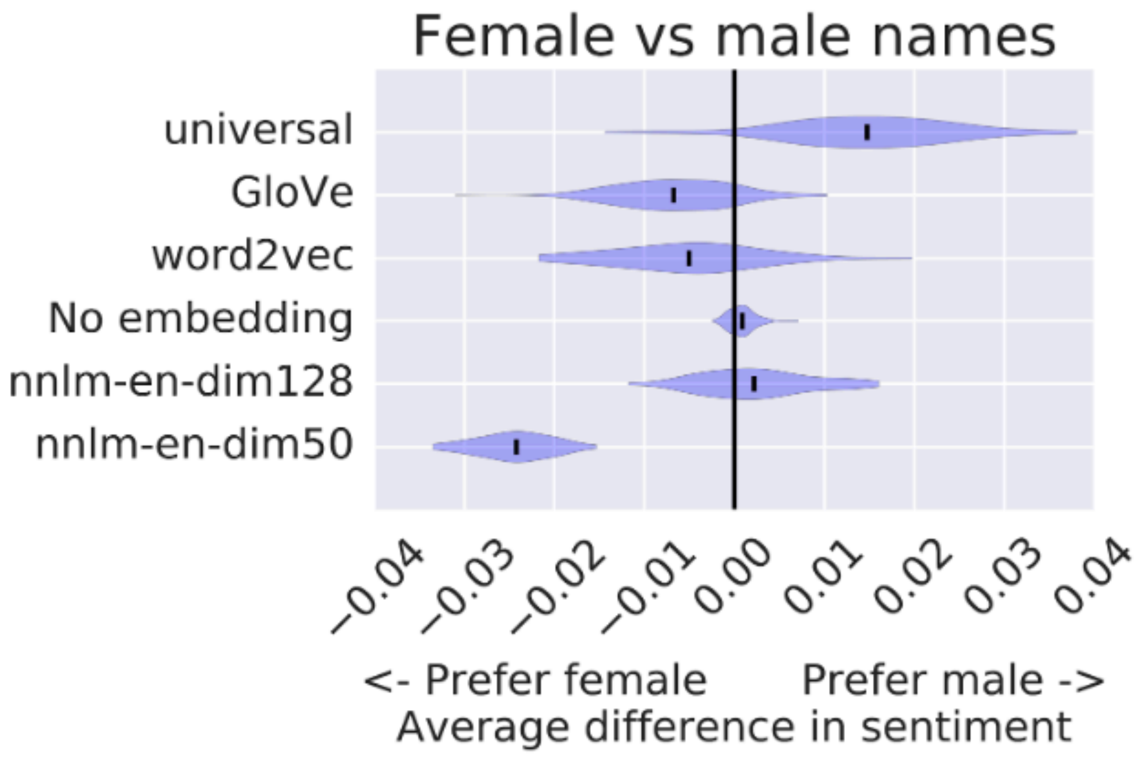
Mila

Université de Montréal

Bias in embeddings

Difference in average sentiment scores:





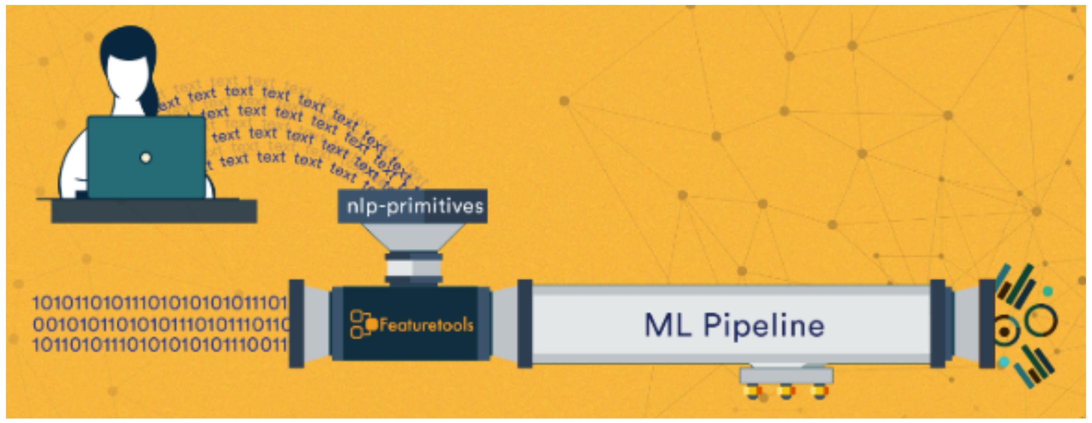
https://developers.googleblog.com/2018/04/text-embedding-models-contain-bias.html

Other Word Embedding approaches?

- **Static word embeddings**
  - Word2Vec (Google): https://code.google.com/archive/p/word2vec/
  - GloVe (Stanford): https://nlp.stanford.edu/projects/glove/
  - FastText: https://fasttext.cc/
- **Dynamic word embeddings**
  - ELMO: https://allennlp.org/elmo
  - FlairEmbeddings: https://github.com/zalandoresearch/flair
  - BERT: https://pypi.org/project/bert-embedding/

Representation learning

- Representation learning is a set of techniques that learn a feature: a transformation of the raw data input to a representation that can be effectively exploited in machine learning tasks.

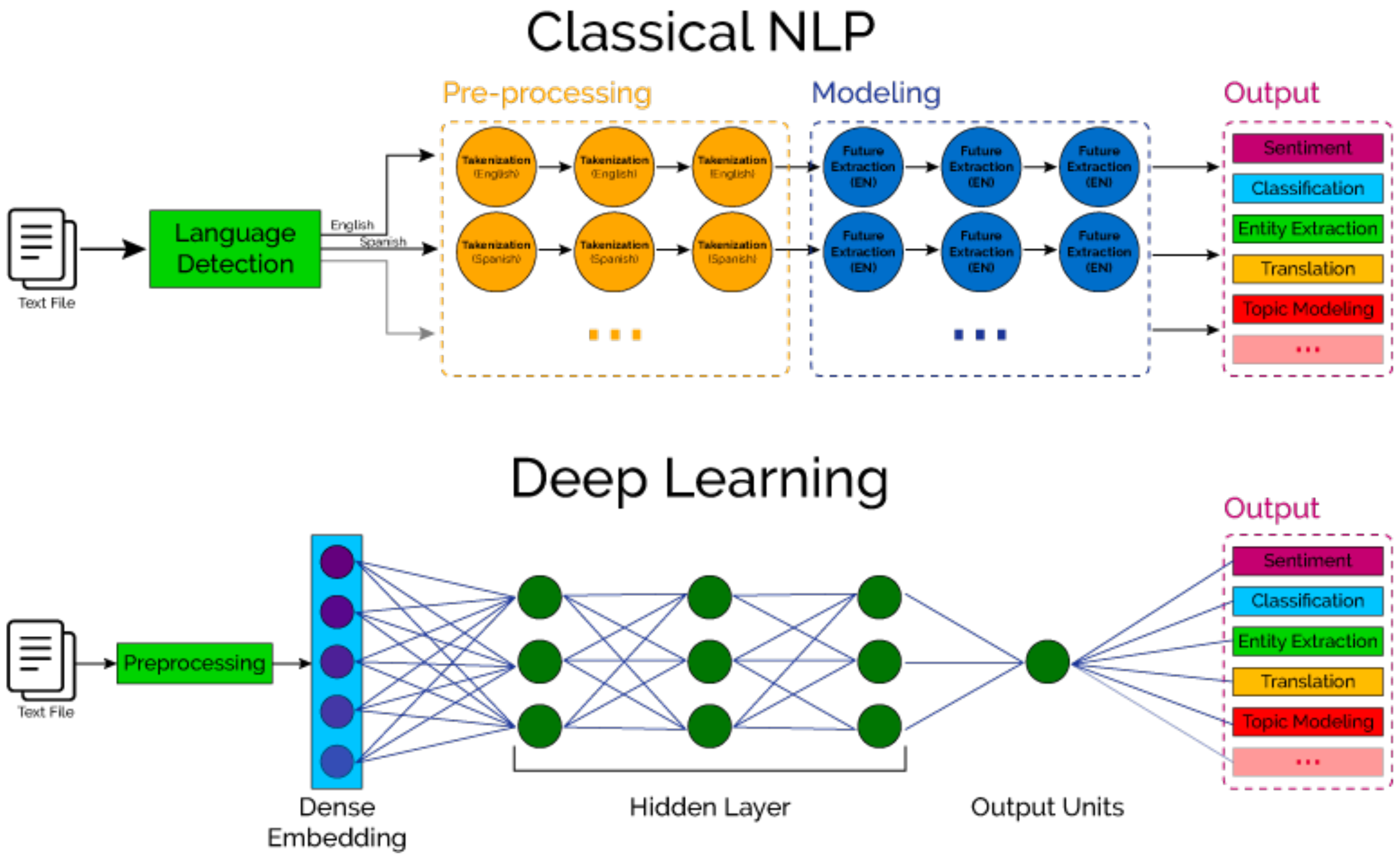


- Part of feature engineering/learning.

- Get rid of “hand-designed” features and representation

- Unsupervised feature learning - obviates manual feature engineering

Difference between classical NLP and Deep learning NLP

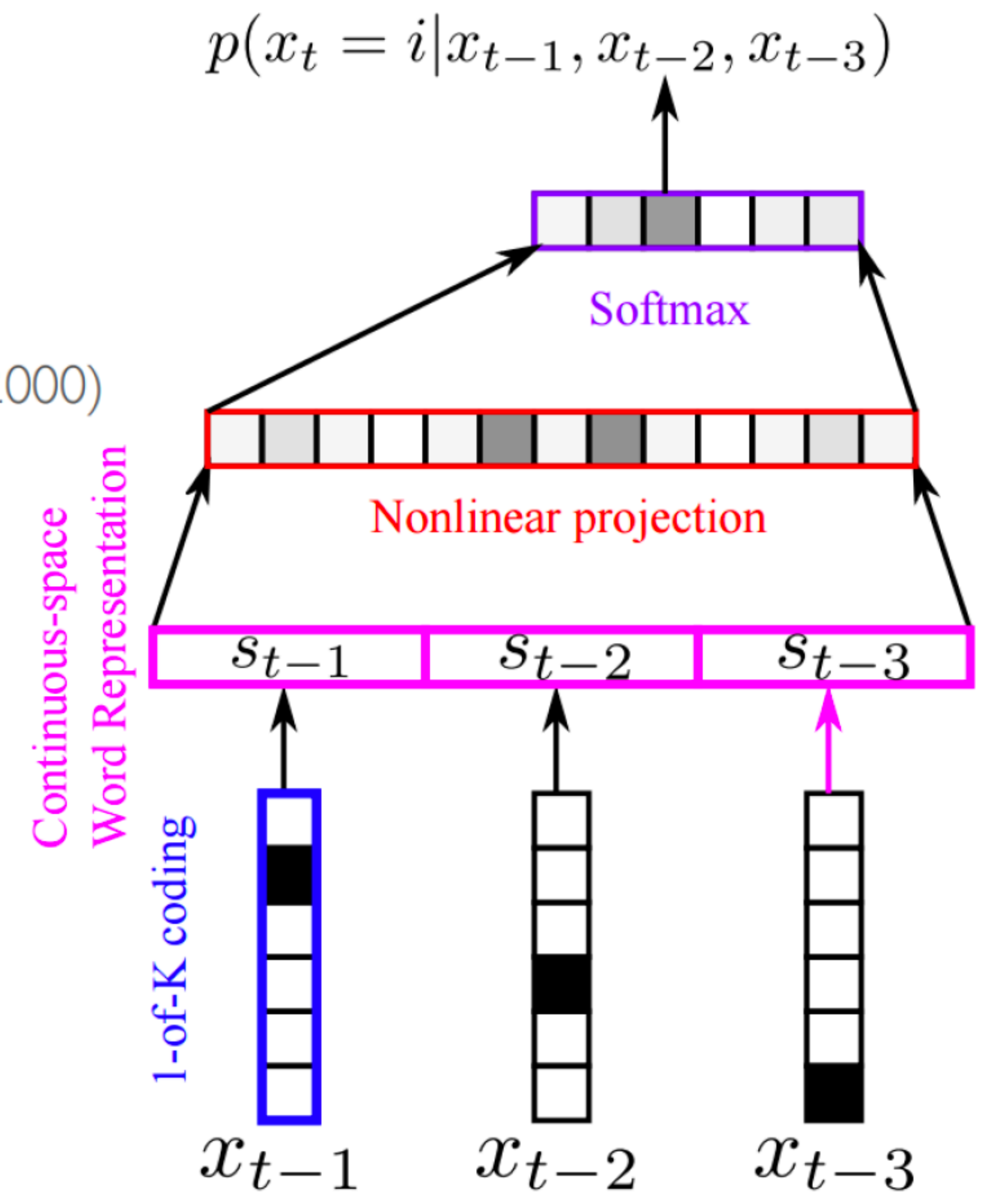


Neural network language model

Topics: Neural Language Modelling

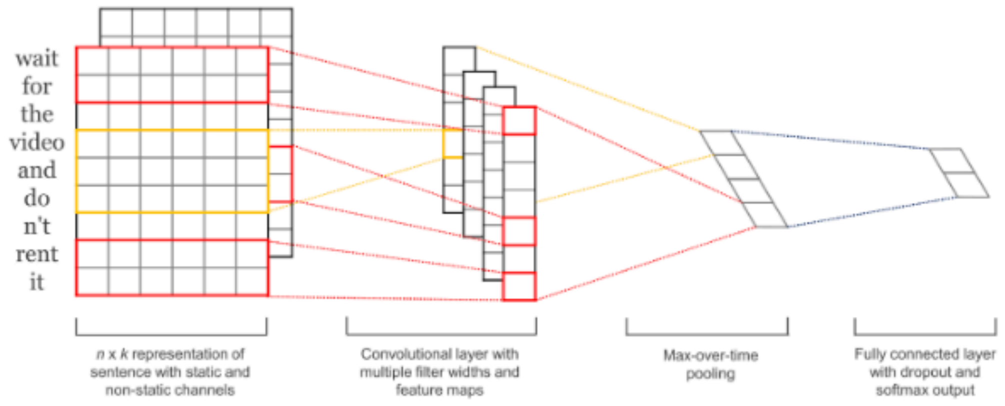
$$p(x_t | x_{t-n}, \ldots, x_{t-1}) = f_\Theta\left(x_{t-n}, \ldots, x_{t-1}\right)$$

- Building a neural language model (Bengio et al., 2000)



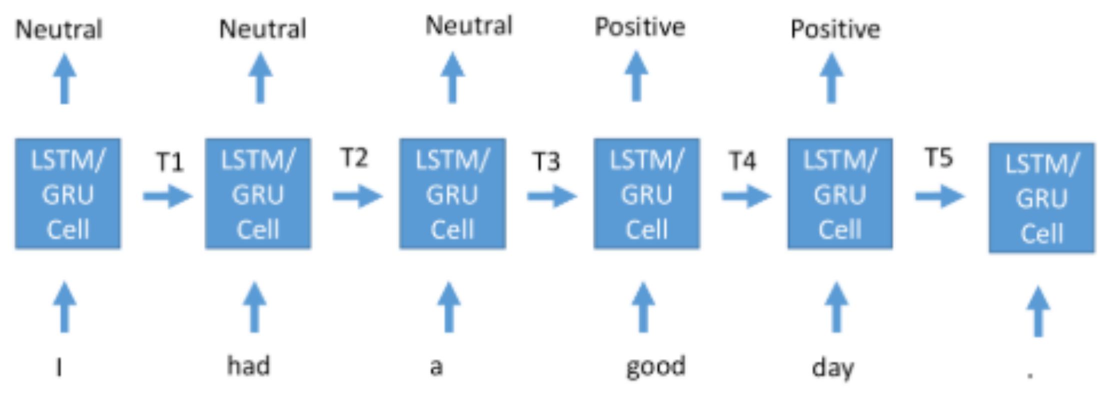
More on NN language modelling

Convolutional Neural Networks (CNNs)


wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification

Recurrent Neural Networks (RNNs)



Next Thursday!

Resources

- Live demo: https://ronxin.github.io/wevi/
- Language modeling: https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf
- C. Manning- Human Language & vector words: http://videolectures.net/deeplearning2015_manning_language_vectors/
- K. Cho - Deep Natural Language Understanding: http://videolectures.net/deeplearning2016_cho_language_understanding/
- John Arevalo, Language modeling and word embeddings

Conferences focusing on NLP

- Natural Language Processing
  ACL, NAACL, EACL, EMNLP, CoNLL, Coling, **TACL**
- Machine learning
  ICML, NIPS, ECML, AISTATS, ICLR, **JMLR**, **MLJ**
- Artificial Intelligence
  AAAI, IJCAI, UAI, **JAIR**