Words

How Human Learn Language

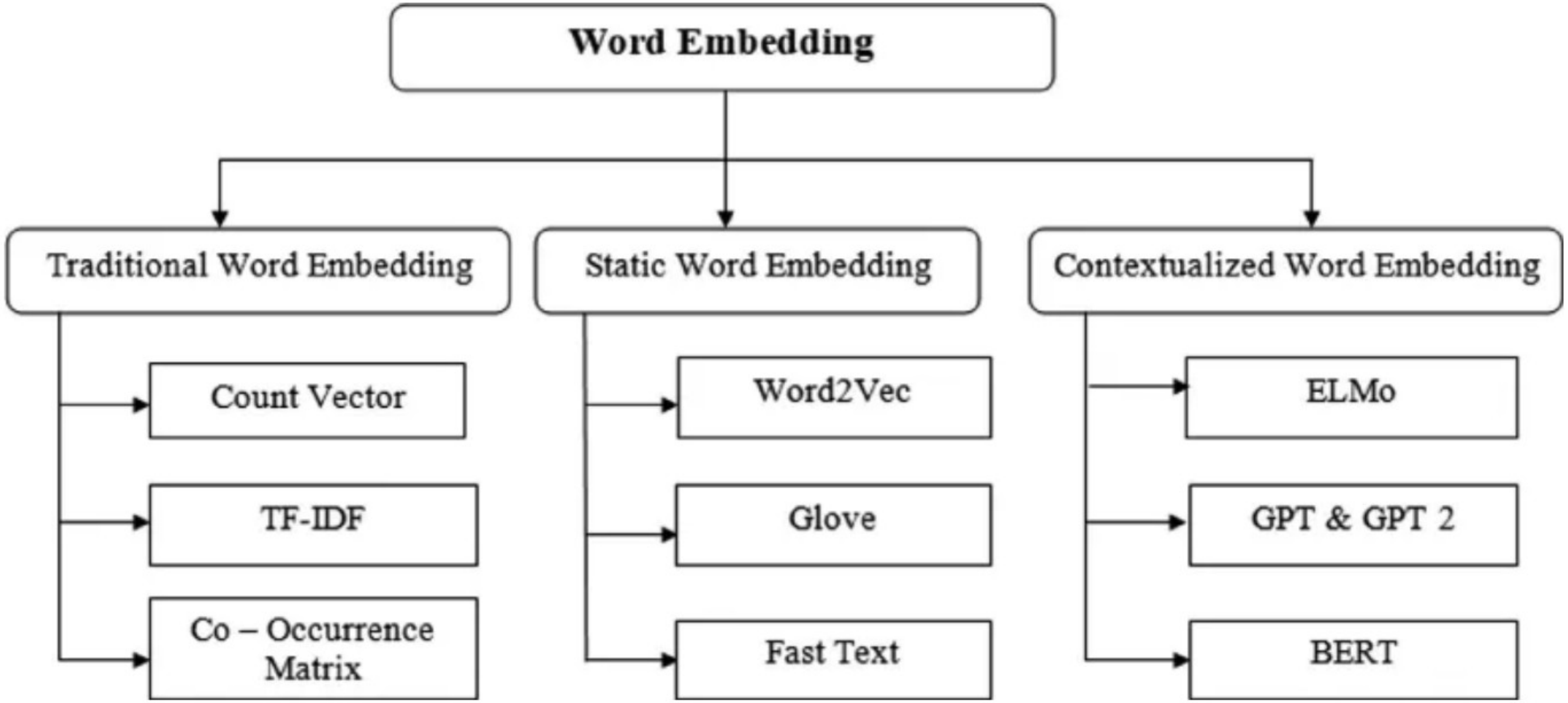
- Syntax
  - Syntax is the study of how words and morphemes combine to form larger units such as phrases and sentences
- Morphology
  - Morphology is the study of words, including the principles by which they are formed, and how they relate to one another within a language
- Semantics
  - Semantics and pragmatics are branches of linguistics concerned with meaning.
- Phonetics
  - Phonetics and phonology are branches of linguistics concerned with sounds (or the equivalent aspects of sign languages)

Representation

- Everything needs to be numbers for computer to process
- Encoding VS Embedding

- Encoding
  - Use vectors to present categorical data
  - One-hot encoding (0, 0, 0, 1)
- Embedding
  - Use vectors to capture semantics (meaning)
  - “Good” should be closer to “Excellent” than “University”

Word Embedding



Count-based Representation

- A **document** can be represented by a bag of words (BOW)
- Ignores word order and context

Document	*team*	*coach*	*hockey*	*baseball*	*soccer*	*penalty*	*score*	*win*	*loss*	*season*
Document1										
Document2										
Document3										
Document4										

Document representation – a simple way

- Term-frequency
  - Take the vocabulary
  - Count words

Document	*team*	*coach*	*hockey*	*baseball*	*soccer*	*penalty*	*score*	*win*	*loss*	*season*
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

- Term-existence
  - Binary: 0 means not exist and 1 means exist
  - To handle long document compared with short documents

Word count

- Can you guess what are the most frequent words in English?

- https://en.wikipedia.org/wiki/Most_common_words_in_English#:~:text=100%20most%20common%20words%20%20%20%20Word,%20Grade%201%20%20%2051%20more%20rows%20
- Those very frequent words are also known as “stop words”

Back to the word frequency

- Those words may appear in every documents with high frequency
  - Useless
- Not a good measurement of how important a word is
- **term frequency–inverse document frequency (TF-IDF)**
- **TF(term, document)= (raw count of “term” in a document)/(total number of words)**
- **IDF(term, document)=log (total number of documents/number of documents containing “term”)**
- **TF-IDF=TF*IDF**

Similarity Computation in Text

- A **document** can be represented by a bag of terms or a long vector, with each attribute recording the *frequency* of a particular term (such as word, keyword, or phrase) in the document

Document	*team*	*coach*	*hockey*	*baseball*	*soccer*	*penalty*	*score*	*win*	*loss*	*season*
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

- *Cosine measure*: If d_1 and d_2 are two vectors (e.g., term-frequency vectors), then

$$cos(d_1, d_2) = \frac{d_1 \bullet d_2}{\| d_1 \| \times \| d_2 \|} \qquad \text{cosine}(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \sqrt{\sum_{i=1}^{d} y_i^2}}$$

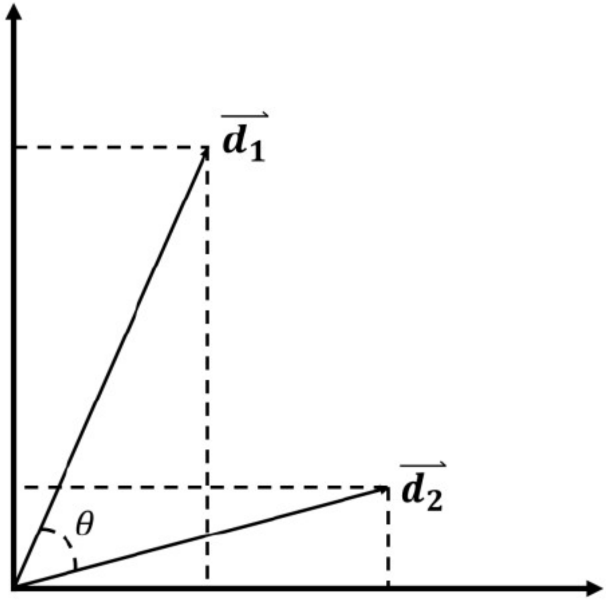
where $\bullet$ indicates vector dot product, $||d||$: the length of vector d

- *Jaccard coefficient is used in some analysis:*

$$\text{Jaccard}(S_x, S_y) = \frac{|S_x \cap S_y|}{|S_x \cup S_y|} \qquad \text{Jaccard}(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} x_i \cdot y_i}{\sum_{i=1}^{d} x_i^2 + \sum_{i=1}^{d} y_i^2 - \sum_{i=1}^{d} x_i \cdot y_i}$$

Text Representation: Vector Space Representation

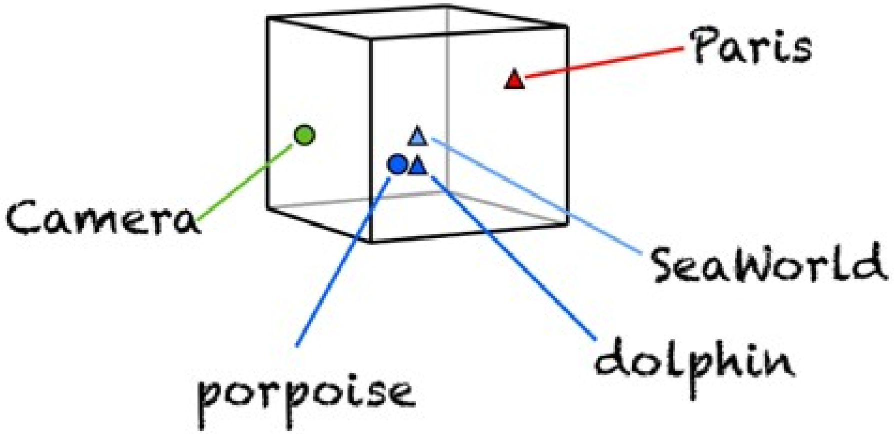
- Normalization (due to different doc length)
  - L-1 normalization: $N(\boldsymbol{w}) = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_1} = \frac{\boldsymbol{w}}{\sum_{i=1}^{V} w_i}$
  - L-2 normalization: $N(\boldsymbol{w}) = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2} = \frac{\boldsymbol{w}}{\sqrt[2]{\sum_i w_i^2}}$
  - L-infinity normalization: $N(\boldsymbol{w}) = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_\infty} = \frac{\boldsymbol{w}}{\max_i\{w_i\}}$



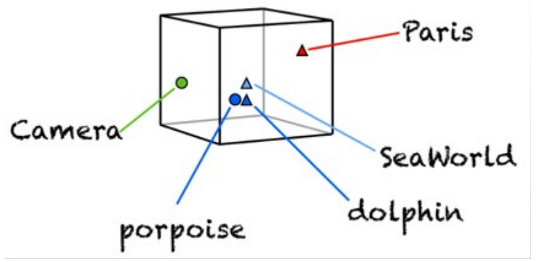
Big Data Challenge: The Curse of High-Dimensionality

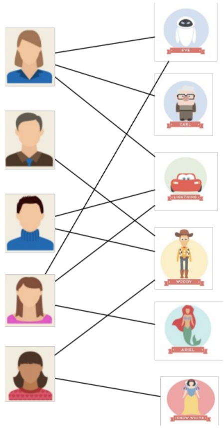
- Text: Word co-occurrence statistics matrix

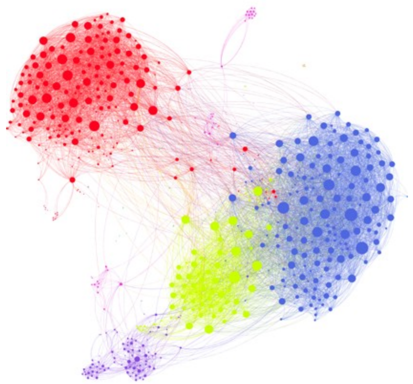
- High-dimensionality:
  - There are over **171k** words in English language
- Redundancy:
  - Many words share similar semantic meanings
    - Sea, ocean, marine..



Solution to Data & Network Challenge: Dimension Reduction



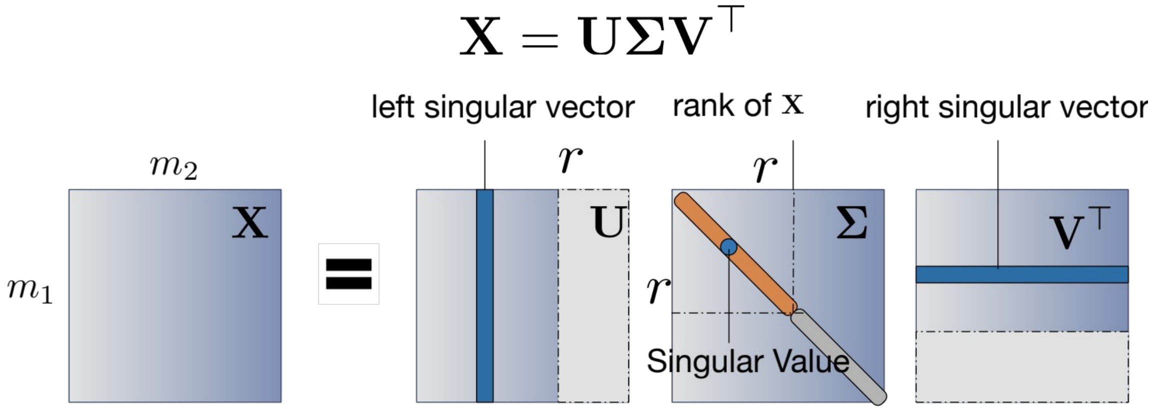


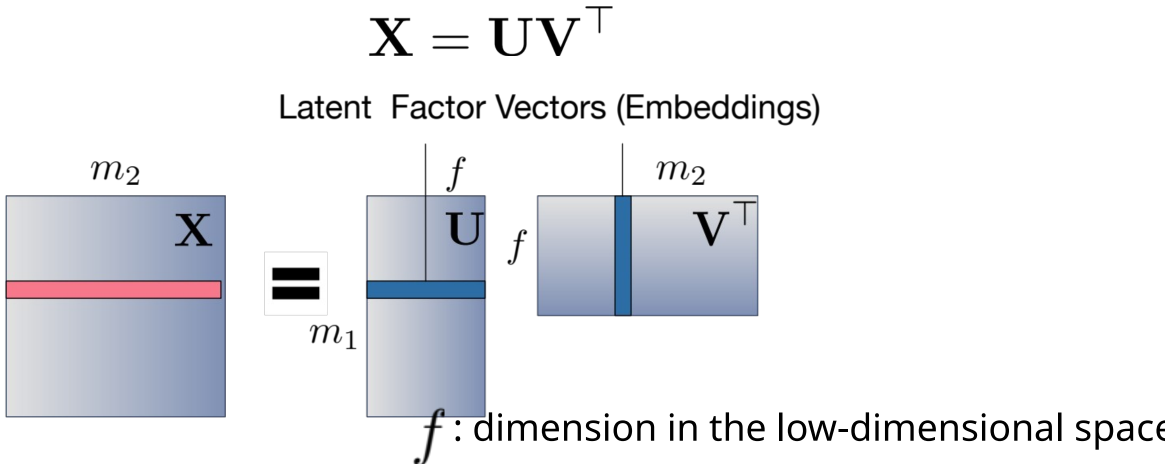


- Why Low-dimensional Space?
  - ❑ Visualization
  - ❑ Compression
  - ❑ Explanatory data analysis
  - ❑ Fill in (impute) missing entries (link/node prediction)
  - ❑ Classification and clustering
  - ❑ Identify / point

- How to automatically identify the lower-dimensional space that the high-dimensional data (approximately) lie in

Dimension Reduction Approaches: Low-rank Estimation vs. Embedding Learning



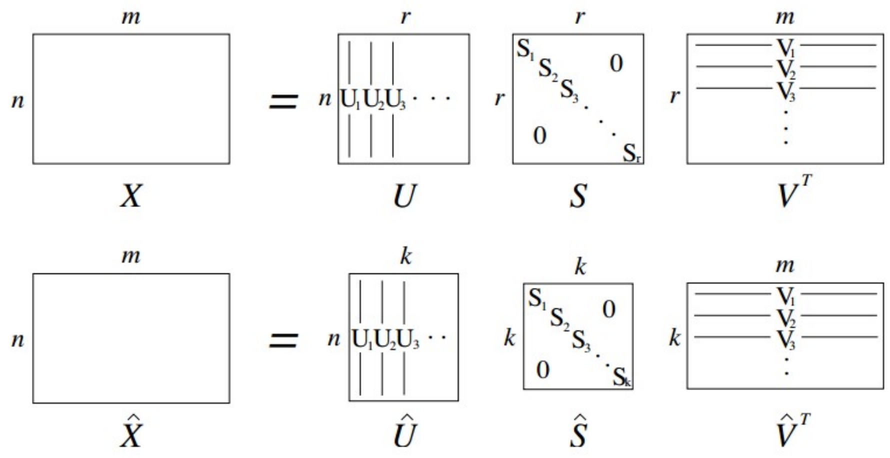


- Low-rank estimation
  - Data recovery
  - Imposing low-rank assumption
    - Regularization
  - Low-dimension vector space
    - Singular vectors (U)
    - = r
  - Low-rank Model

- Embedding Learning
  - Representation Learning
  - Project data into a low-dimensional space
  - Low-dimensional vector space
    - Spanned by columns of U
    - ≤ f
  - Generalized Low-rank Model

Example

- Store only “important” information in fixed, low dimensional vector.
- Singular Value Decomposition (SVD) on co-occurrence matrix
  - is the best rank *k* approximation to *X* , in terms of least squares
  - Motel = [0.286, 0.792, -0.177, -0.107, 0.109, -0.542, 0.349, 0.271]
- m = n = size of vocabulary
- is the same matrix as S except that it contains only the top largest singular values



Problems with SVD

- Computational cost scales quadratically for n x m matrix: $O(mn^2)$ flops (when $n<m$)
- Hard to incorporate new words or documents
- Does not consider order of words

Word2Vec and Word Embedding

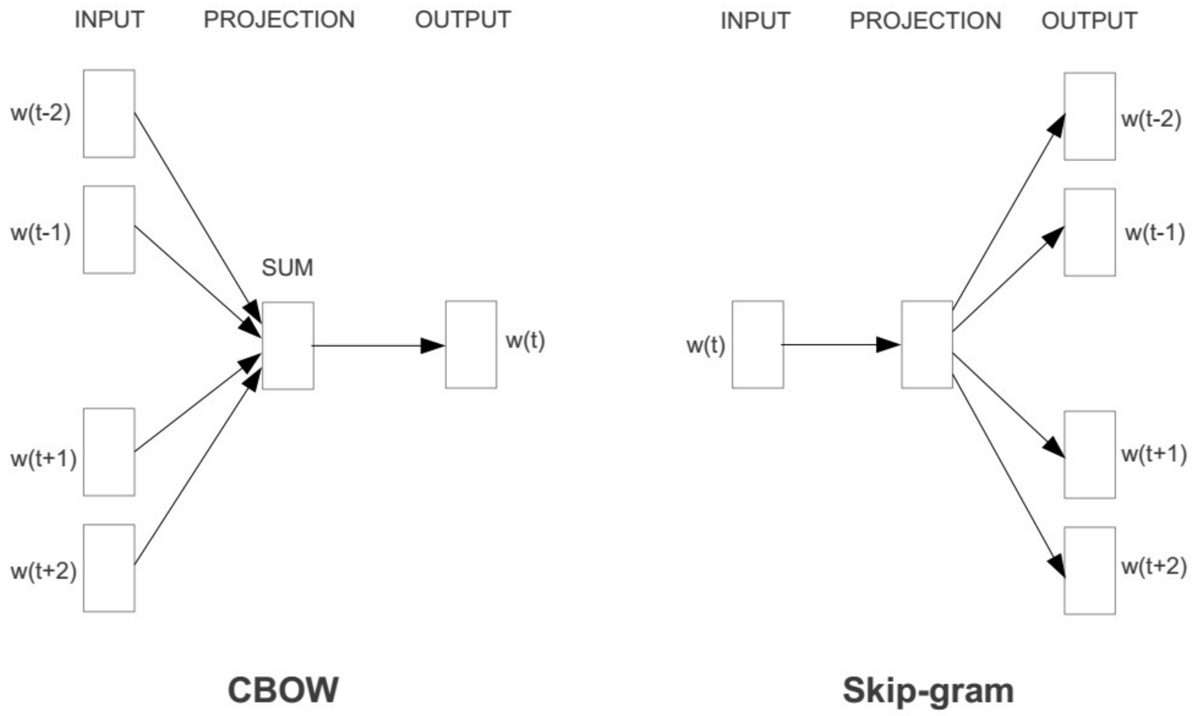
- Word2vec: A two-layer neural net that processes text
    - Invented by T. Mikolov et al. at Google (2013)
  - Input: a large text corpus
  - Output: A set of vectors, feature vectors for words in that corpus, of 10^2 dimensions
- Words sharing common contexts are embedded in close proximity in the vector space
- Embedding vectors created by Word2vec: better than LSA (Latent Semantic Analysis)
- Word2vec turns text into a numerical form that deep nets can understand
- Applications: Analysis of text, as well as genes, code, likes, playlists, social media graphs and other verbal or symbolic series

Word2vec approach to represent the meaning of word

- Represent each word with a low-dimensional vector
- Word similarity = vector similarity
- Key idea: Predict surrounding words of every word
- Faster and can easily incorporate a new sentence/document or add a word to the vocabulary

Word2Vec: CBOW vs. Ski-Gram Models

- Two model architectures:
  - Continuous bag-of-words (CBOW): use a window of word to predict the middle word
    - Order does not matter, faster
  - Continuous skip-gram: use a word to predict the surrounding ones in window.
    - Weigh nearby context words more heavily than more distant context words
    - Slower but better job for infrequent words



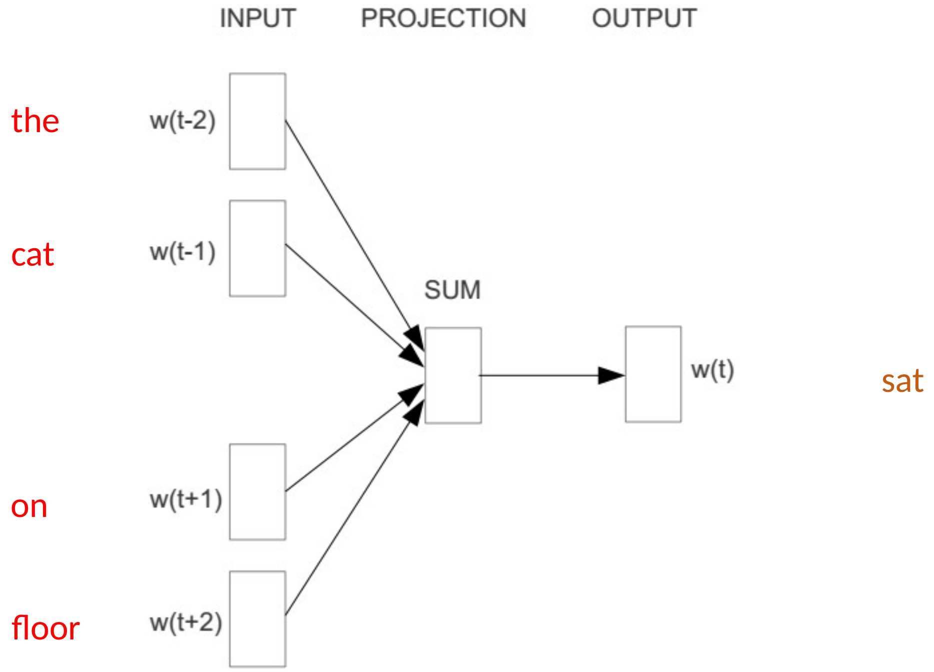
CBOW

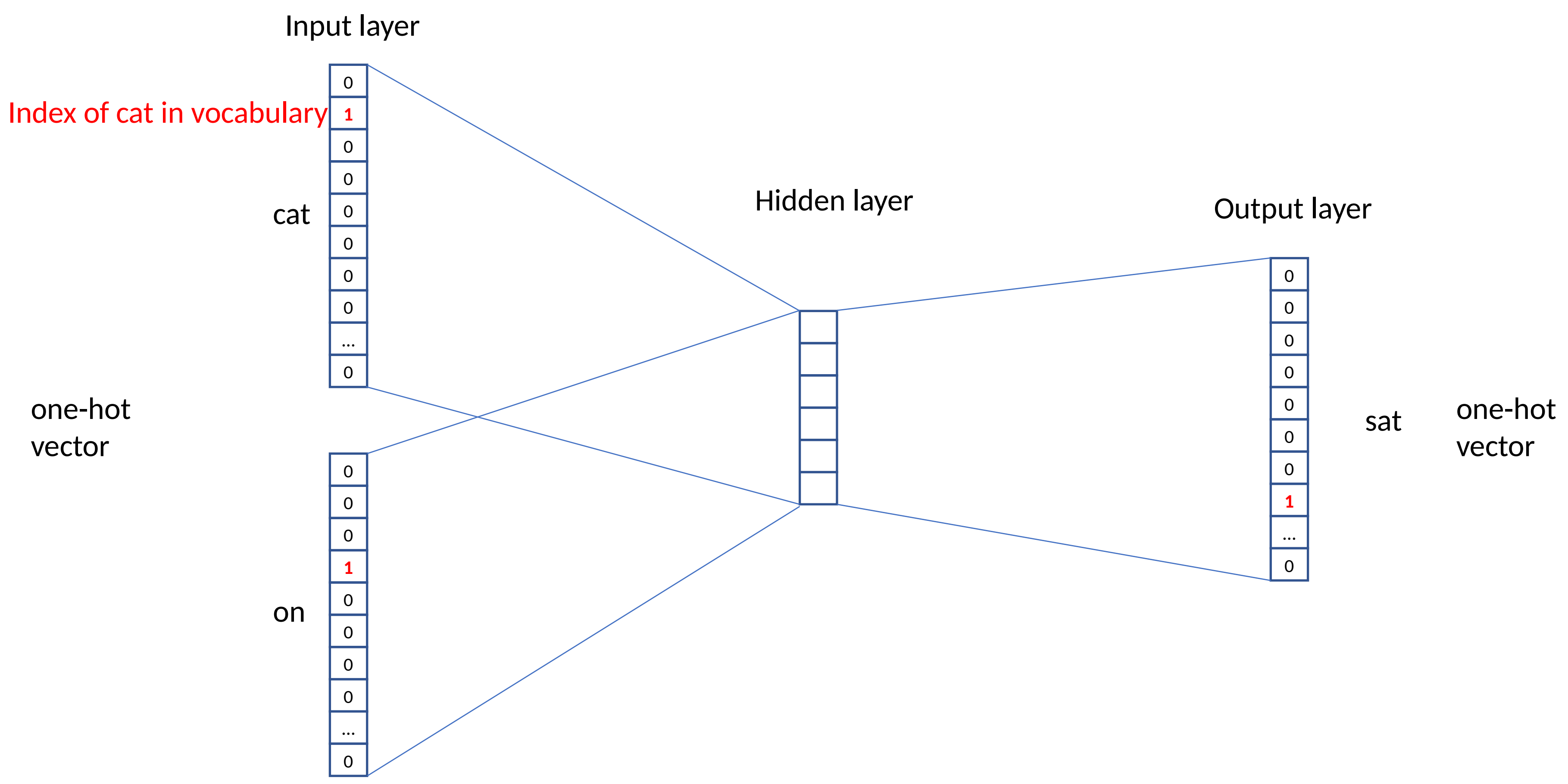
- The amount contributed ? the plan is
- income will be ? by investment performance.
- If you have ? regarding your eligibility

- The amount contributed to the plan is
- income will be determined by investment performance.
- If you have questions regarding your eligibility

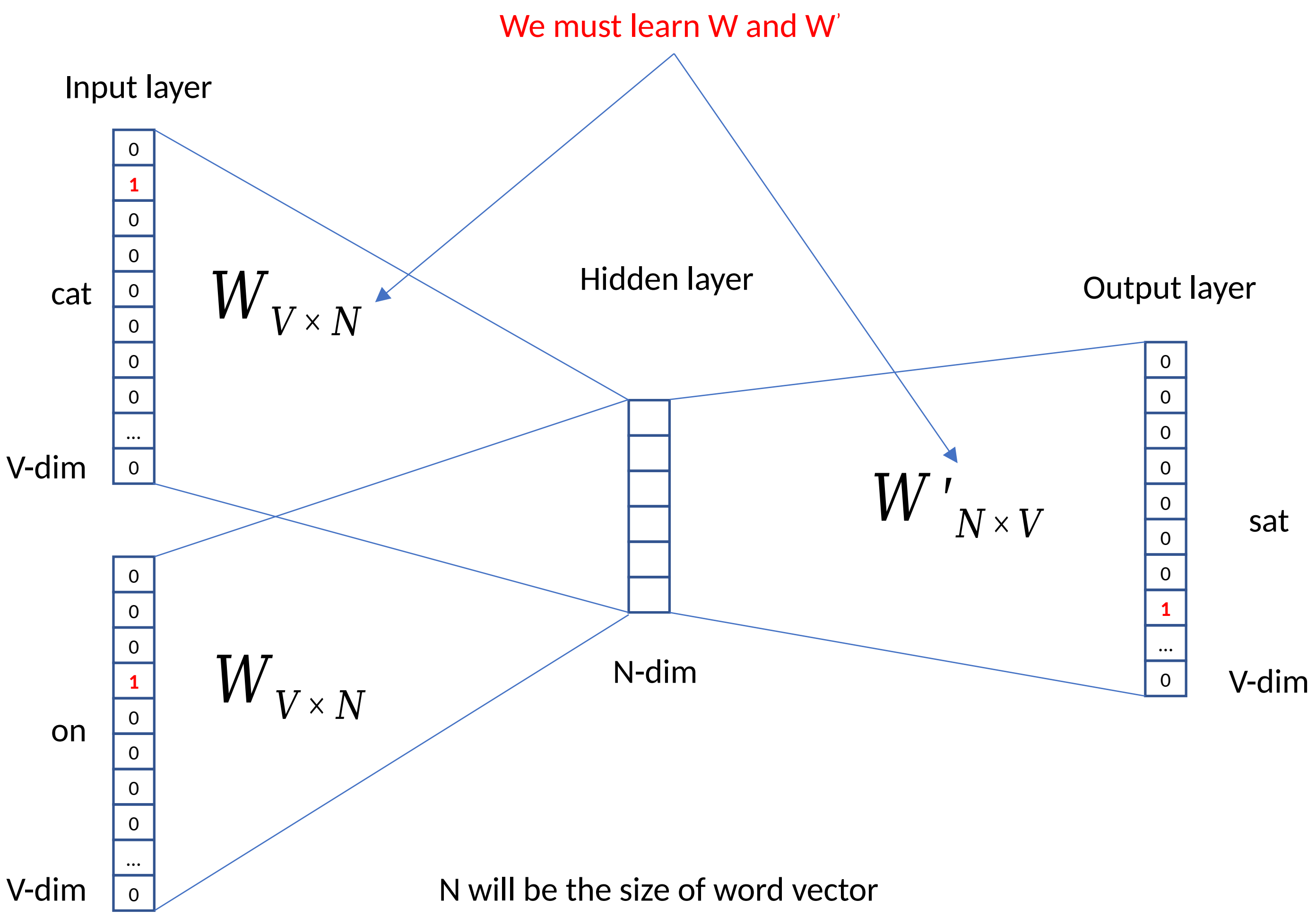
Word2vec – Continuous Bag of Word

- E.g. “The cat sat on floor”
  - Window size = 2

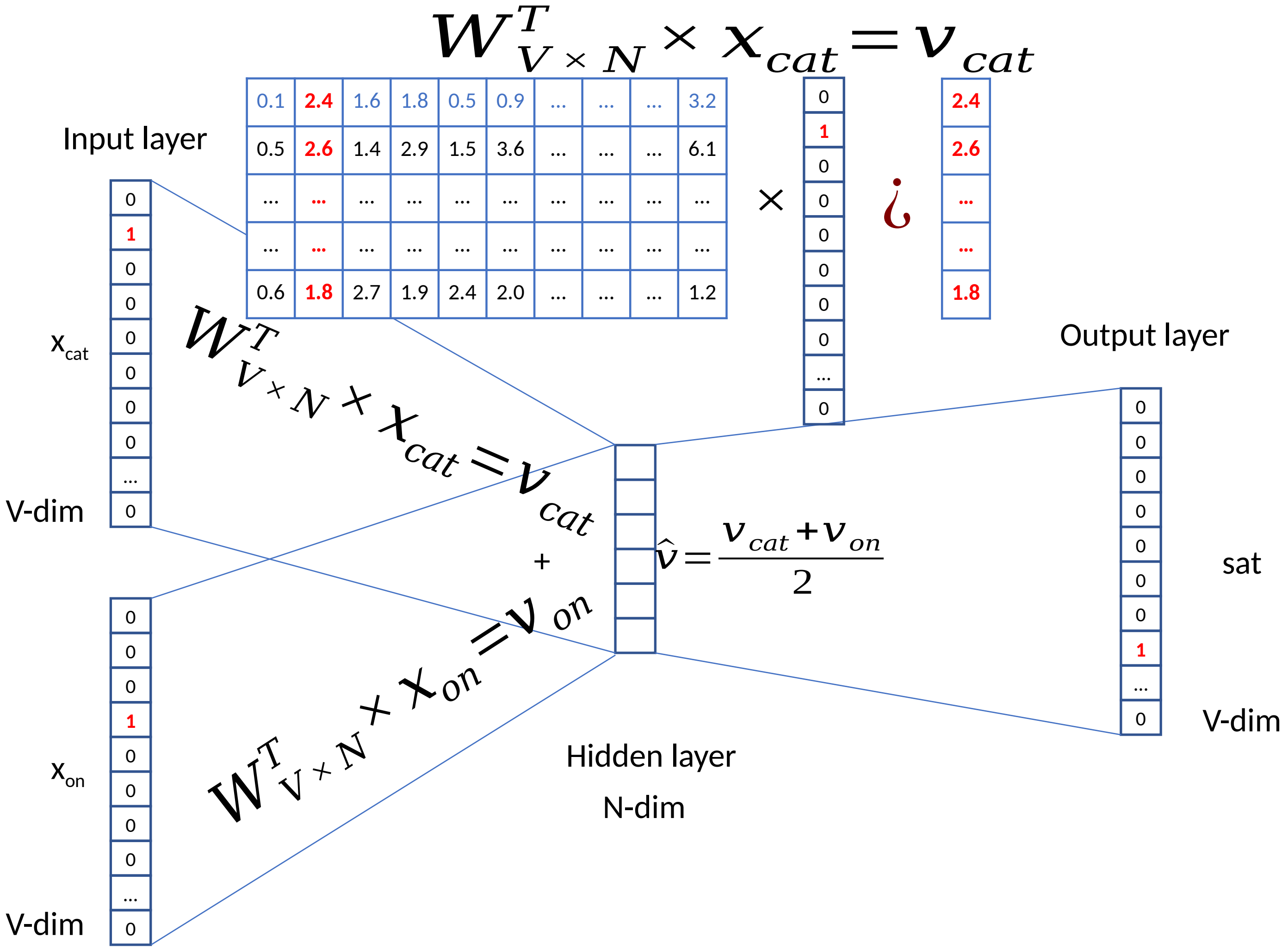



Input layer
Index of cat in vocabulary
cat
one-hot
vector
on
Hidden layer
Output layer
sat
one-hot
vector
0
1
0
0
0
0
0
0
...
0
0
0
0
1
0
0
0
0
...
0
0
0
0
0
0
0
0
1
...
0

22

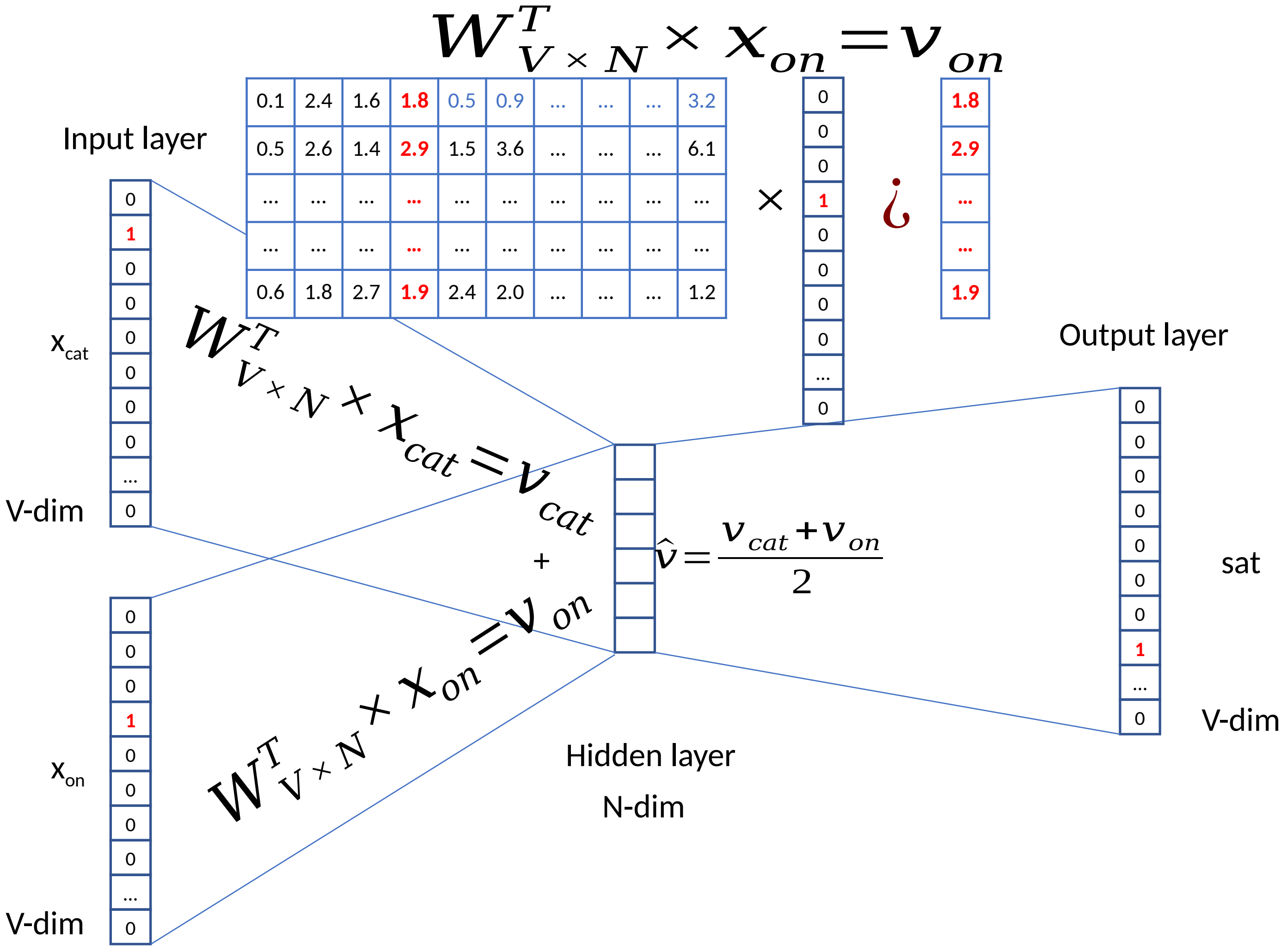

We must learn W and W'
Input layer
cat
V-dim
on
V-dim
$W_{V \times N}$
$W_{V \times N}$
Hidden layer
N-dim
$W'_{N \times V}$
Output layer
sat
V-dim
N will be the size of word vector

23

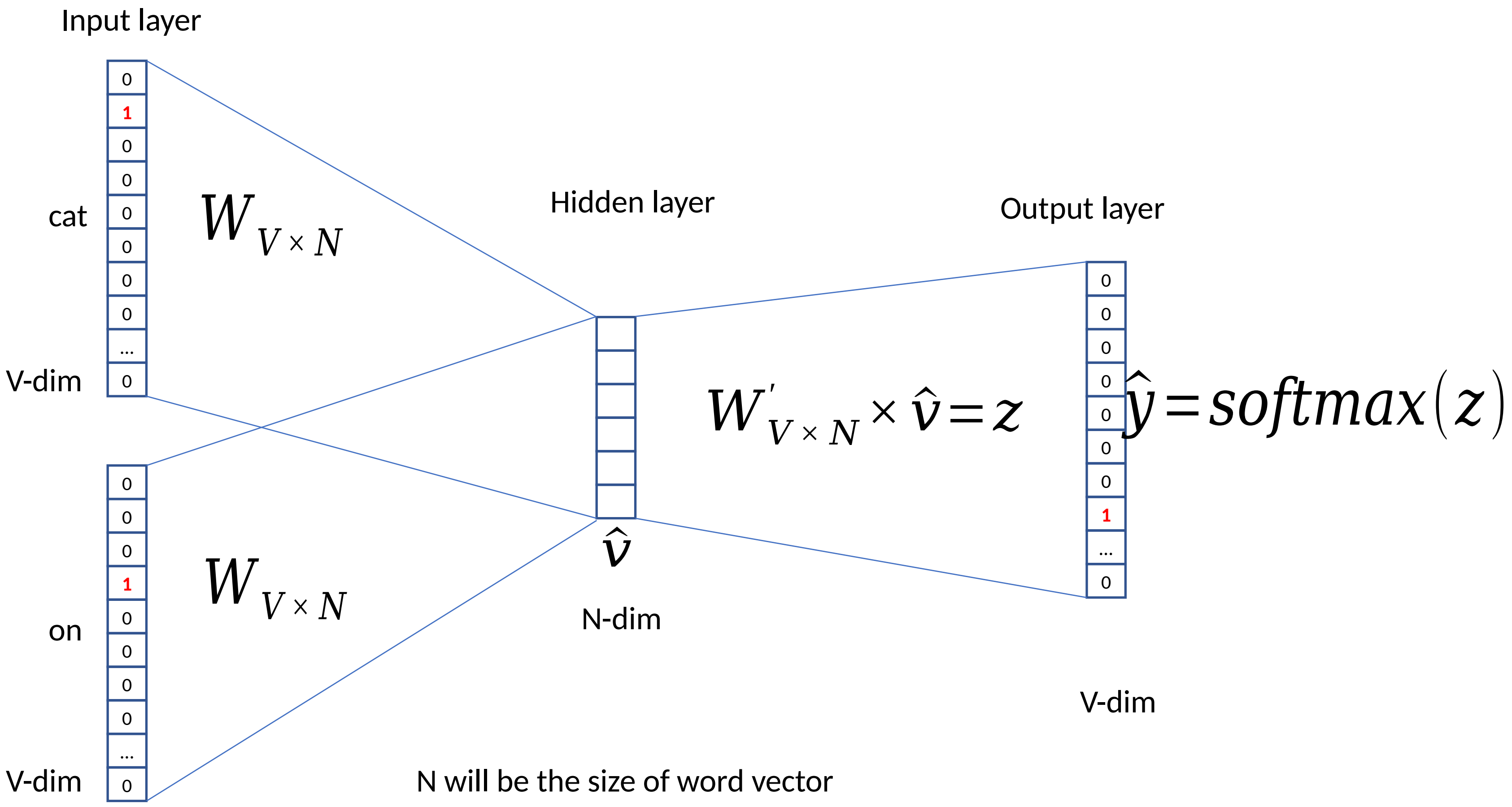


$W^T_{V\times N} \times x_{cat} = v_{cat}$
Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat}+v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim

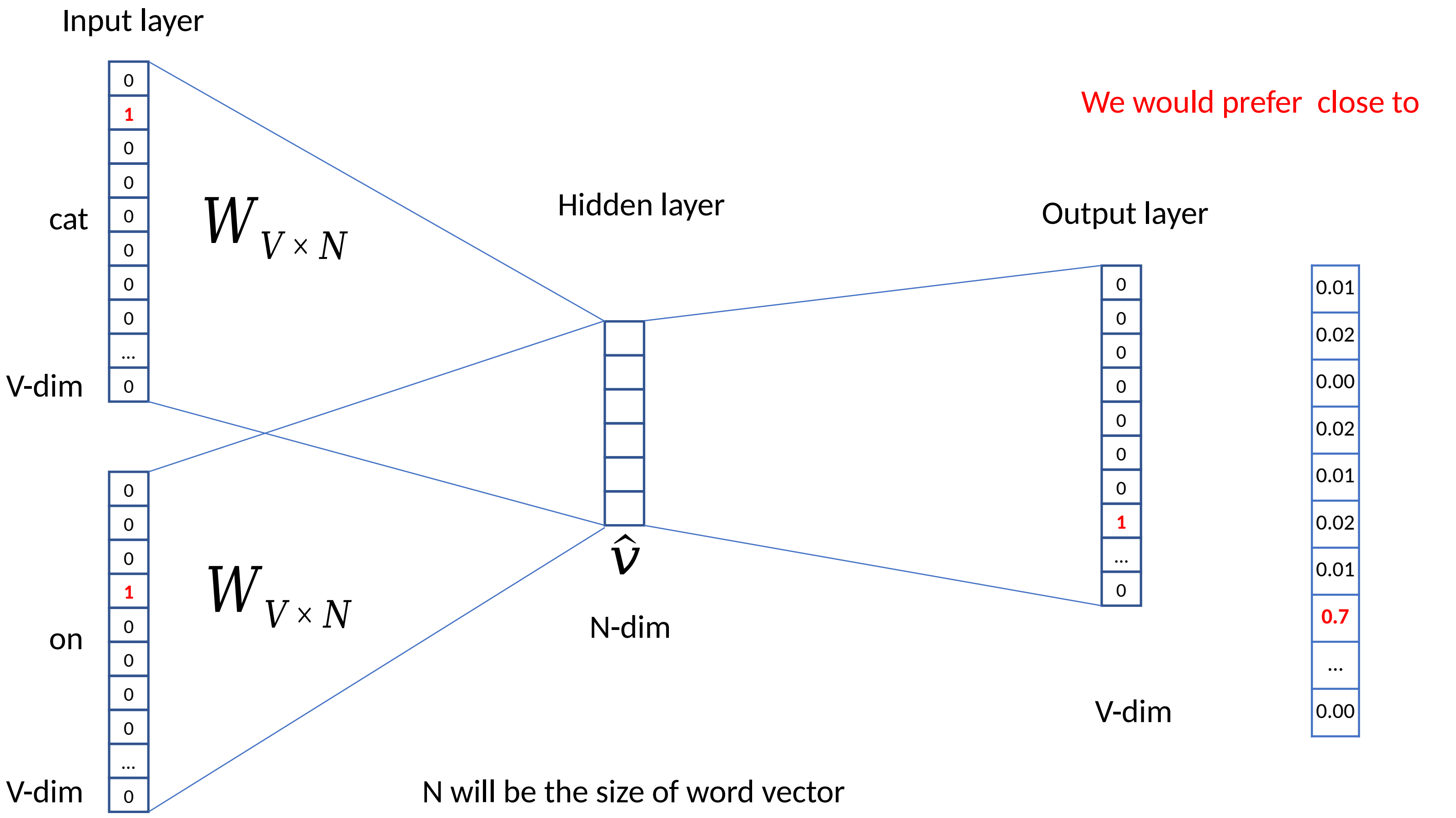
24



$W^T_{V\times N}\times x_{on}=v_{on}$
Input layer
0.1 2.4 1.6 1.8 0.5 0.9 … … … 3.2
0.5 2.6 1.4 2.9 1.5 3.6 … … … 6.1
0.6 1.8 2.7 1.9 2.4 2.0 … … … 1.2
x_{cat}
V-dim
$W^T_{V\times N}\times x_{cat}=v_{cat}$
+
$W^T_{V\times N}\times x_{on}=v_{on}$
x_{on}
V-dim
$\hat{v}=\frac{v_{cat}+v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim

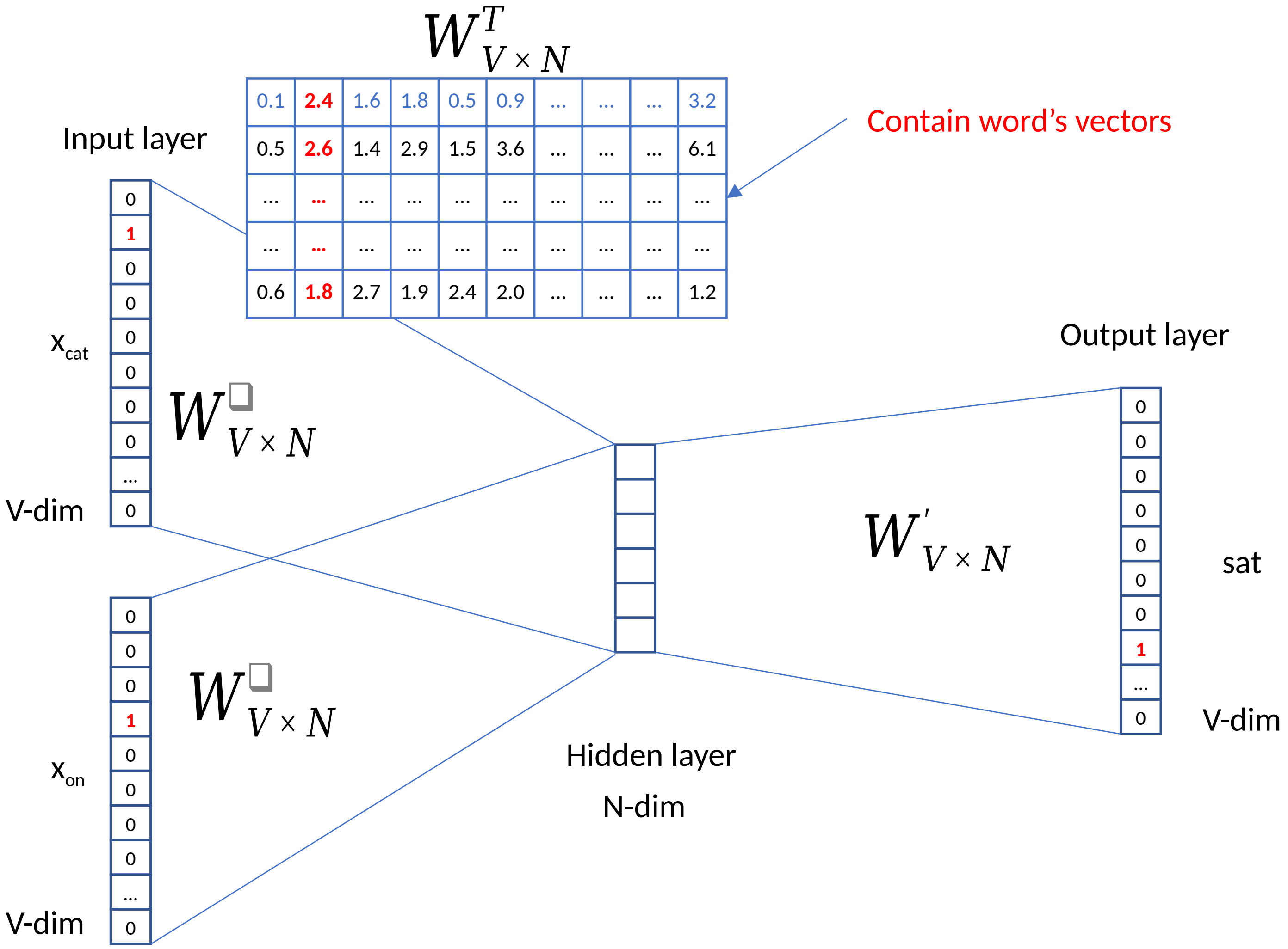

Input layer
cat
V-dim
on
V-dim
$W_{V \times N}$
$W_{V \times N}$
Hidden layer
$\hat{v}$
N-dim
Output layer
$W'_{V \times N} \times \hat{v} = z$
$\hat{y} = softmax(z)$
V-dim
N will be the size of word vector

26



Input layer
cat
V-dim
on
V-dim
$W_{V \times N}$
$W_{V \times N}$
Hidden layer
$\hat{v}$
N-dim
Output layer
V-dim
We would prefer close to
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 1 ... 0
0.01 0.02 0.00 0.02 0.01 0.02 0.01 0.7 ... 0.00
N will be the size of word vector

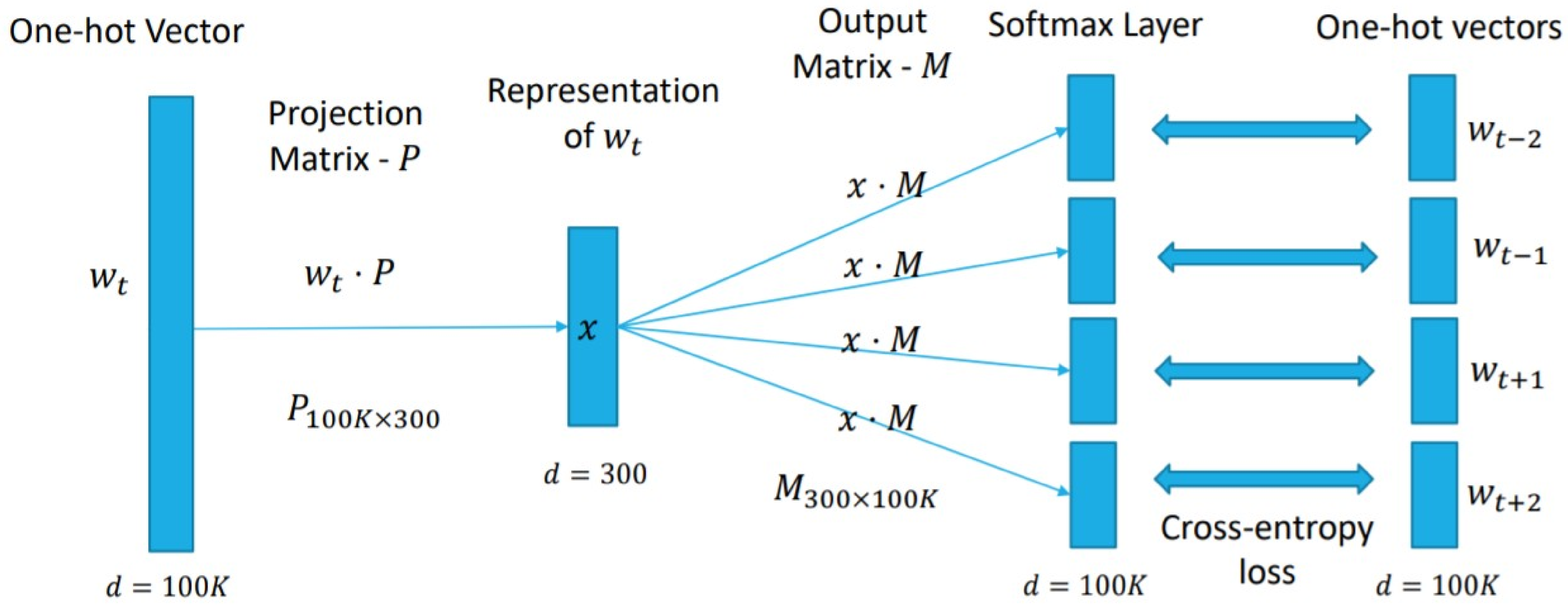
27



We can consider either W or W' as the word's representation.
Or even take the average.

28

Skip-gram – high level



Skip-gram details

- Given a sequence of training words , 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 using the softmax function:

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

v - input vector representations

v' - output vector representatior

cross-entropy loss

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

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

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

"positive" pair

"negative" pair

Negative Sampling

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

"positive" pair

"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:

$$\log \sigma(v'_{w_{t+j}} v_{w_t}) + \sum_{1=1}^{k} \log \sigma(-v'_{w_i} v_{w_t})$$

Replaces the term: $\log p(w_{t+j}|w_t)$ for each word in the training

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

Subsampling of Frequent Words

- 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 in the training set is discarded with probability:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

- This way frequent words are discarded more often
- This method improves the training speed and makes the word representations significantly more accurate

Some interesting results

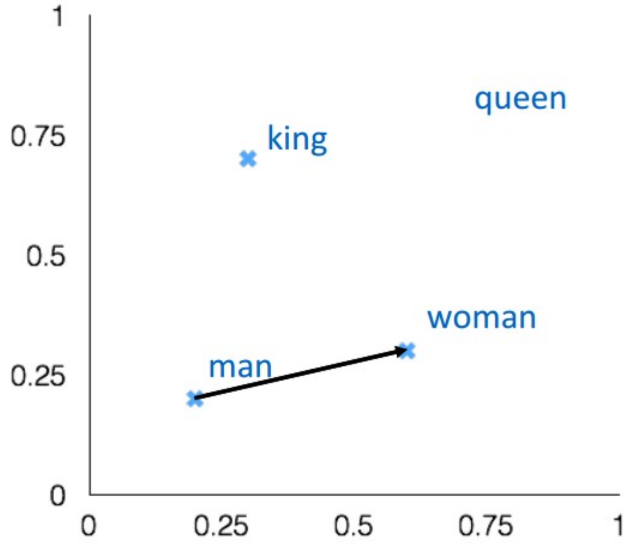
Word Analogies

Test for linear relationships, examined by Mikolov et al. (2014)

a:b :: c:? → $$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

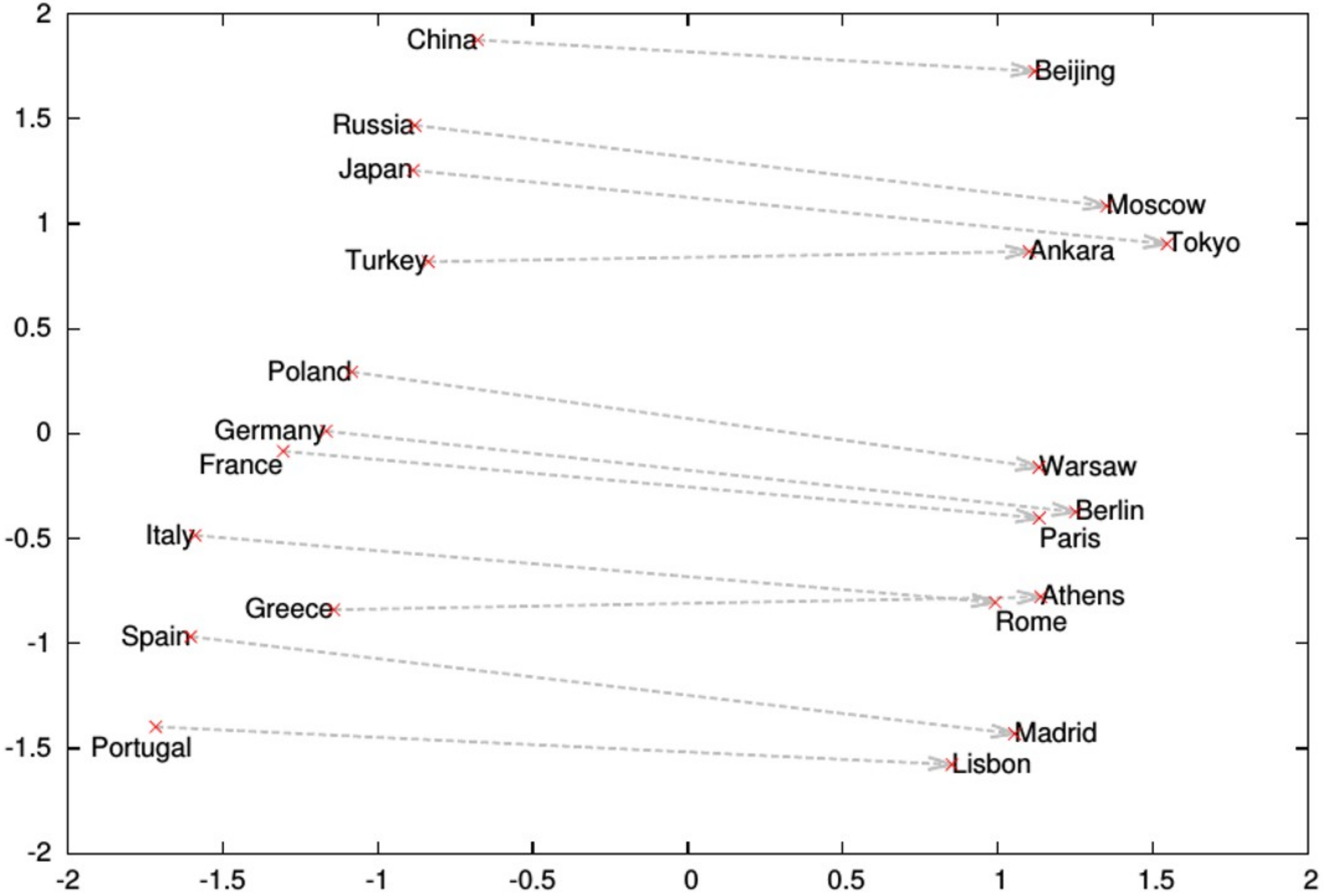
man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



34

Word analogies



Importance of Parameters – window size

Importance of Parameters – window size

Word: **walk**

Window size = 3

Word	Cosine distance
go	0.488083
snipe	0.464912
shoot	0.456677
fly	0.449722
sit	0.449678
pass	0.442459
climbs	0.440931
walked	0.436502
ride	0.434034
stumble	0.426750
bounce	0.425577
travelling	0.419419
walking	0.412107
walks	0.410949
trot	0.410418
leaping	0.406744
sneak	0.401918
climb	0.399793
move	0.396715
wait	0.394463
going	0.391639
shouted	0.388382
roam	0.388073
thrown	0.384087
get	0.383894

Window size = 30

Word	Cosine distance
walking	0.486317
walked	0.430764
walks	0.406772
stairs	0.401518
go	0.399274
sidewalk	0.385786
stand	0.380480
cortege	0.371033
wheelchair	0.362877
strapped	0.360179
hollywood	0.356544
carousel	0.356187
grabs	0.356007
swim	0.355027
breathe	0.354314
tripped	0.352899
cheer	0.352477
moving	0.350943
inductees	0.347791
walkway	0.347164
shout	0.346229
pounding	0.340554
blvd	0.339121
crowd	0.338731
levada	0.334899

Other word embedding

- Glove (Pennington et al.)
  - Based on ratios of co-occurrence probabilities
  - https://nlp.stanford.edu/projects/glove/
- Fast-text (Bojanowski et al.):
  - Each word is represented as a bag of character n-grams.
  - A vector representation is associated to each character n-gram, and words are represented as the sum of these representations
  - https://fasttext.cc/

GloVe

- Generate Co-occurrence matrix X (symmetric)
  - Take a context window (distance around a word, e.g. 10)
  - X(i,j) = # of times 2 words lie in the same context window
- Factorize X
- Ratio of co-occurrences

Encoding meaning in vector differences

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

Probe word

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

Encoding meaning in vector differences

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

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

Better distinguish power than raw probability

GloVe

- General model

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}},$$

  - F: function
  - w: embedding
  - P: probability
- A specific choice of F

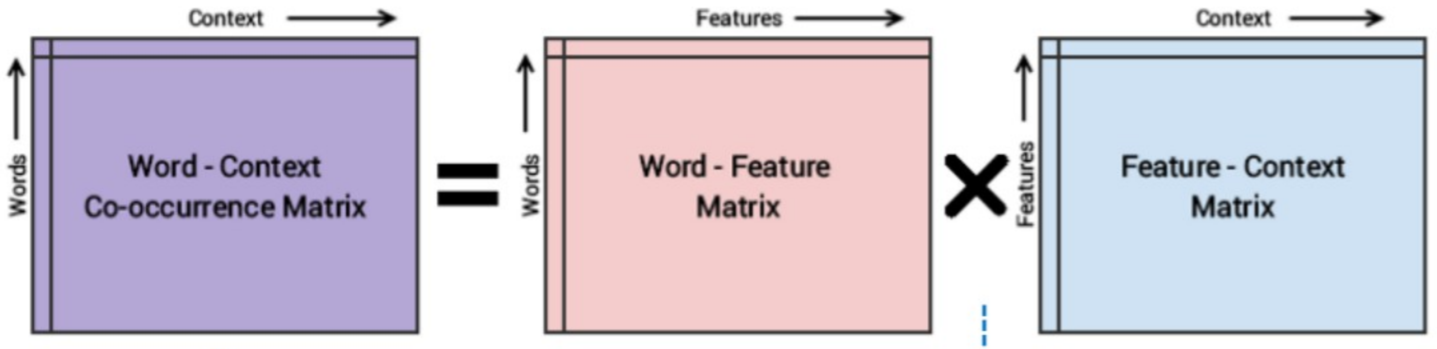
GloVe

Low-dimensional representations are obtained by solving a least-squares problem to “recover” the co-occurrence matrix

$$J = \sum_{i,j=1}^{V} \boxed{f\left(X_{ij}\right)} \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

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

Weighting function. Rare co-occurrence can be noisy



Sparse, high dimensional

Low dimensional representation

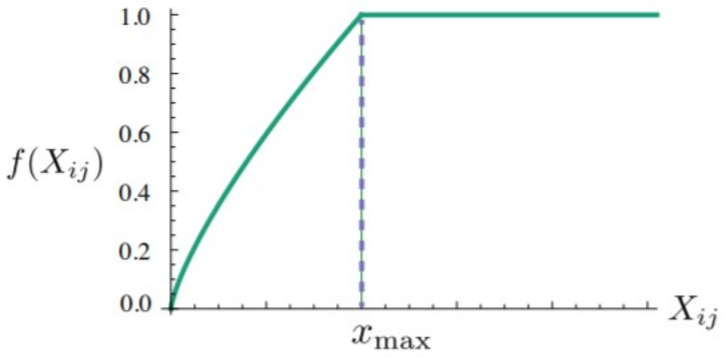
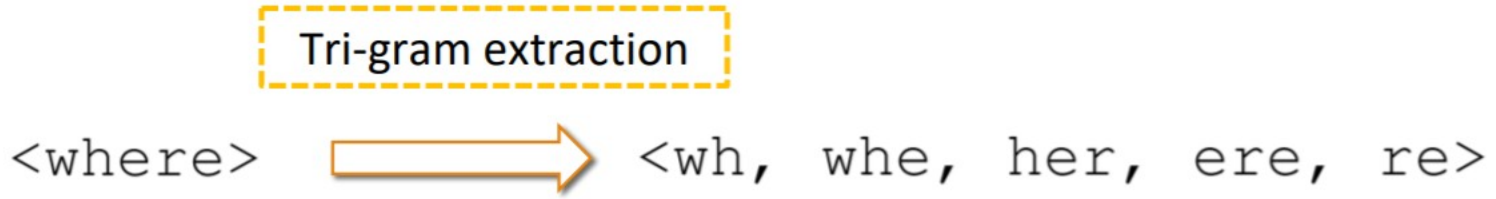


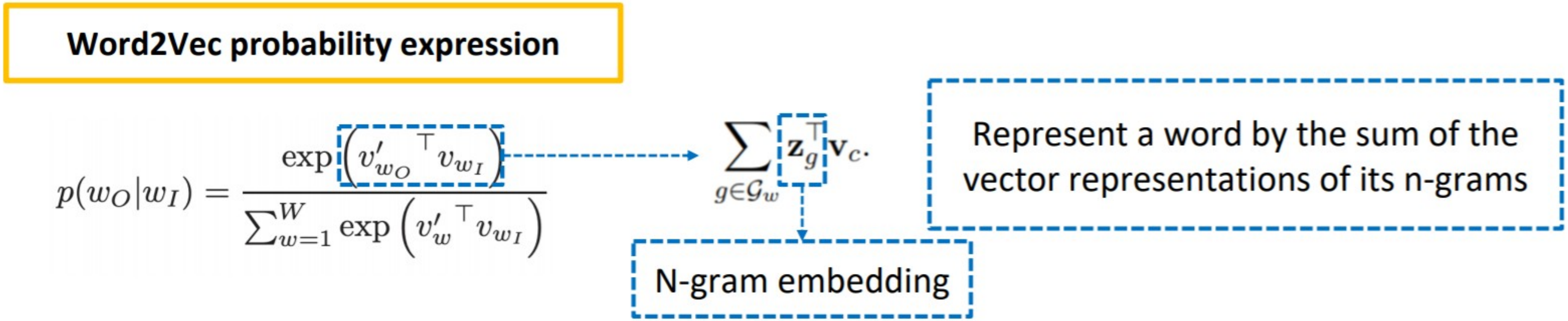
Figure 1: Weighting function f with $\alpha = 3/4$.

fastText

- fastText improves upon Word2Vec by incorporating subword informat

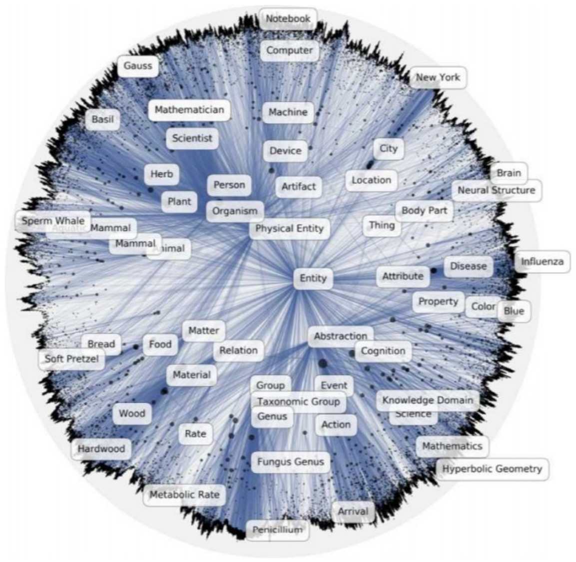


- fastText allows sharing subword representations across words, since



Hyperbolic Embedding: Poincaré embedding

- Why non-Euclidean embedding space?
  - Data can have specific structures that Euclidean-space models struggle to capture
- The hyperbolic space
  - Continuous version of trees
  - Naturally equipped to model hierarchical structures



- Poincaré embedding
  - Learn hierarchical representations by pushing general terms to the origin of the Poincaré ball, and specific terms to the boundary

$$d(\boldsymbol{u}, \boldsymbol{v}) = \operatorname{arcosh}\left(1 + 2\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{(1 - \|\boldsymbol{u}\|^2)(1 - \|\boldsymbol{v}\|^2)}\right)$$

Texts in Hyperbolic Space: Poincaré GloVe

- GloVe in hyperbolic space
- Motivation: latent hierarchical structure of words exists among text
  - Hypernym-hyponym
  - Textual entailment
- Approach: use hyperbolic kernels!
- Effectively model generality/specificity

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right) \left(\boxed{w_i^T \tilde{w}_j} + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

GloVe

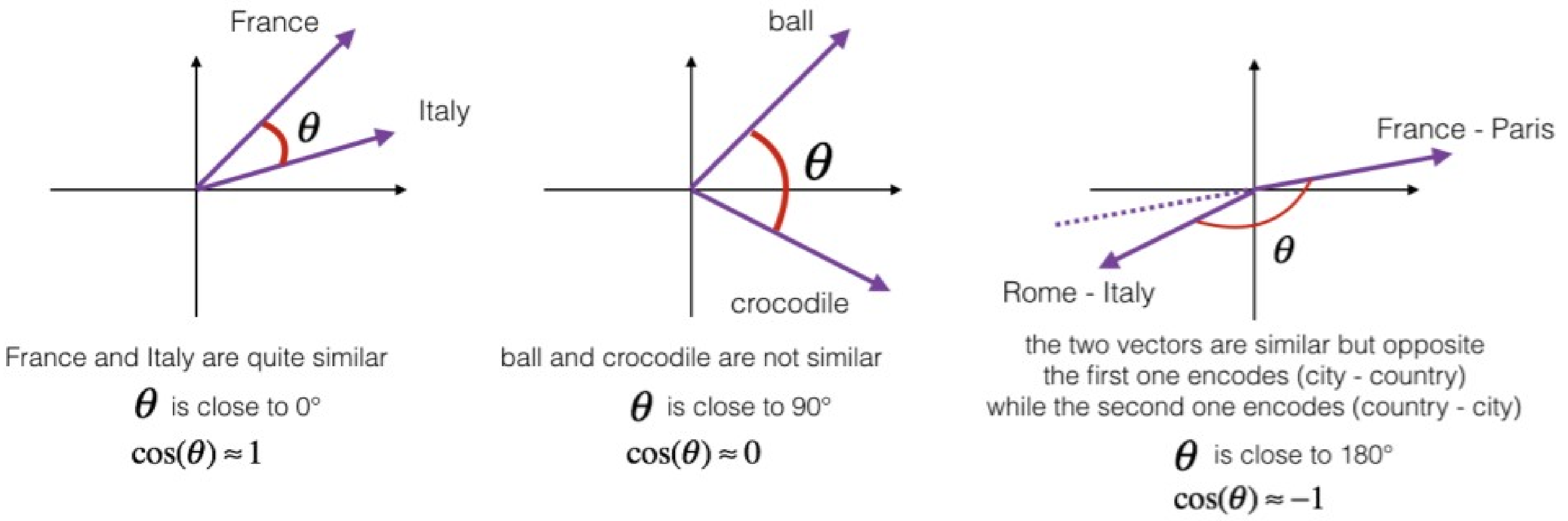
Hyperbolic metric

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(\boxed{-h(d(w_i, \tilde{w}_j))} + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Poincaré GloVe

Spherical Text Embedding

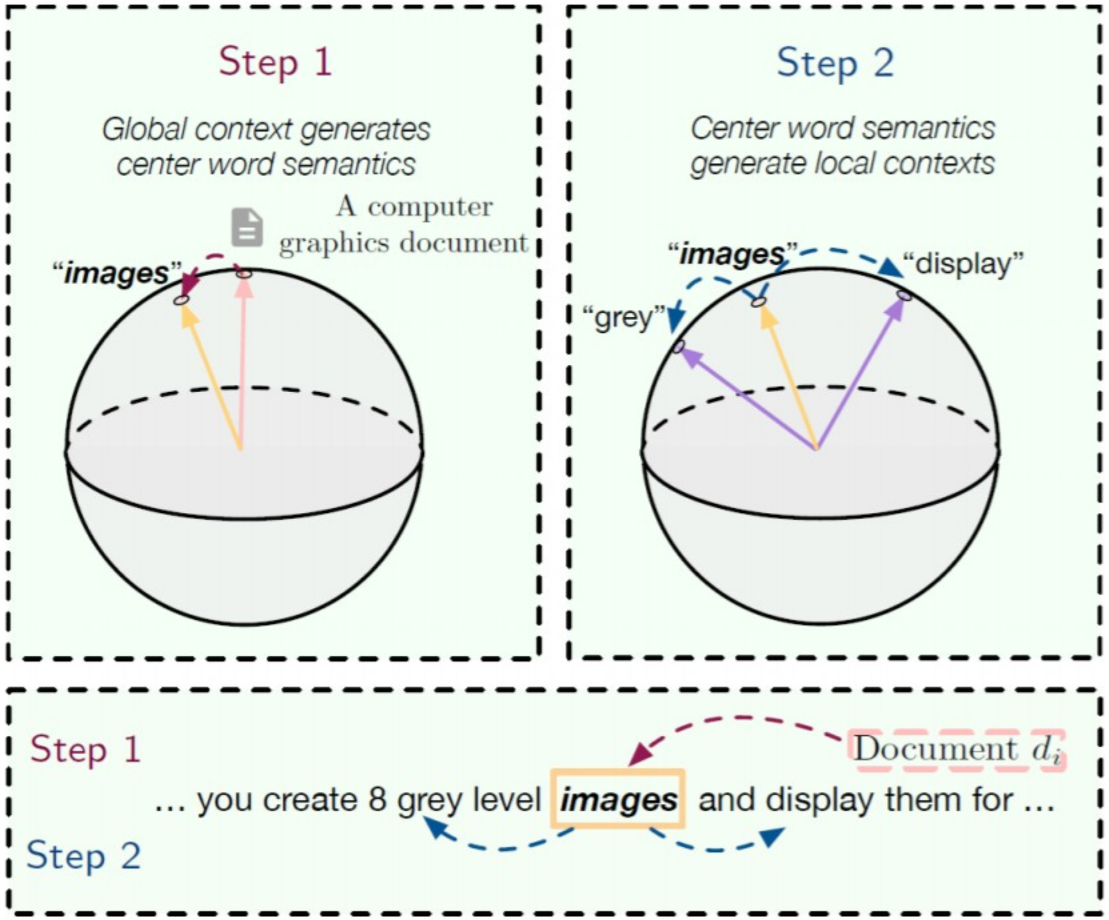
- Motivation
  - Word similarity is derived using cosine similarity

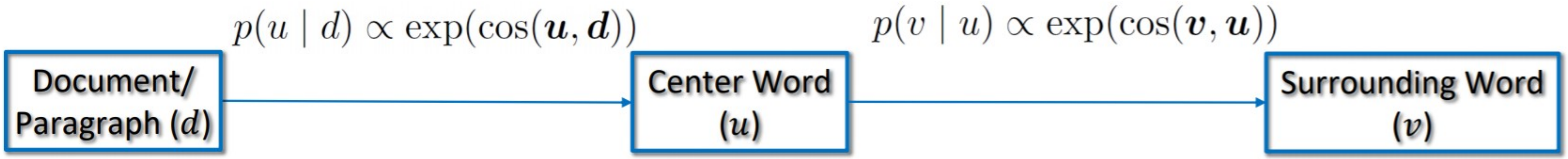


  - A gap between training space and usage space: Trained in Euclidean space but used on sphere

Spherical Text Embedding

- a generative model on the sphere that follows how humans write articles:
  - We first have a general idea of the paragraph/document, and then start to write down each word in consistent with not only the paragraph/document, but also the surrounding words
  - Assume a two-step generation process:





Spherical Text Embedding

- Training objective:
  - The final generation probability:

Von Mises-Fisher distribution

$$p(v, u \mid d) = p(v \mid u) \cdot p(u \mid d) = \boxed{\mathrm{vMF}_p}(\boldsymbol{v}; \boldsymbol{u}, 1) \cdot \mathrm{vMF}_p(\boldsymbol{u}; \boldsymbol{d}, 1)$$

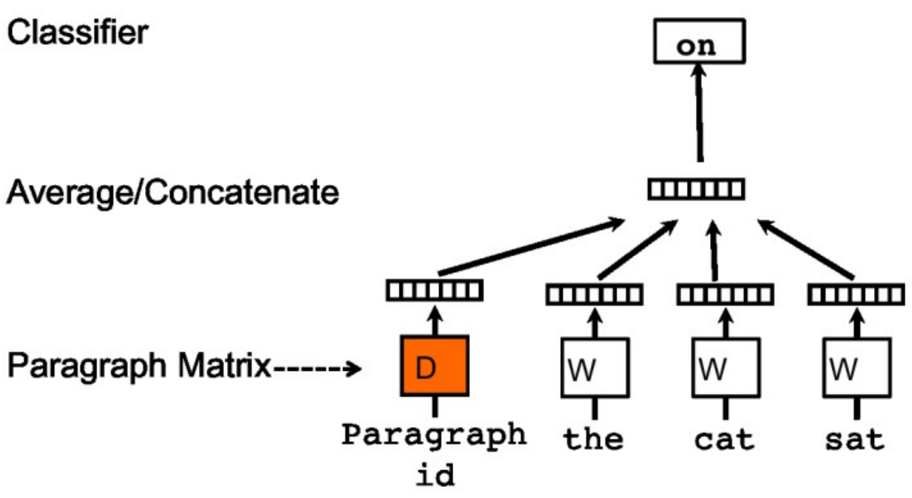
  - Maximize the log-probability of a real co-occurred tuple , while minimize that of a negative sample , with a max-margin loss:

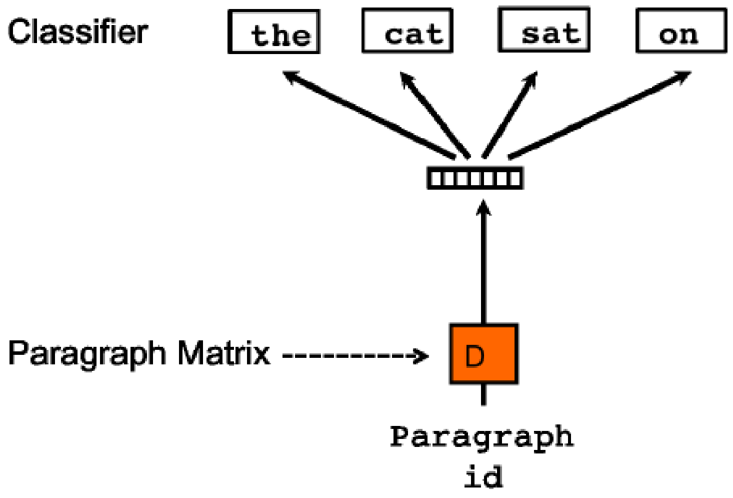
$$\begin{aligned}\mathcal{L}_{\text{joint}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{d}) &= \max\Big(0, m - \underbrace{\log\left(c_p(1)\exp(\cos(\boldsymbol{v}, \boldsymbol{u})) \cdot c_p(1)\exp(\cos(\boldsymbol{u}, \boldsymbol{d}))\right)}_{\text{Positive Sample}} \\ &\quad + \underbrace{\log\left(c_p(1)\exp(\cos(\boldsymbol{v}, \boldsymbol{u}')) \cdot c_p(1)\exp(\cos(\boldsymbol{u}', \boldsymbol{d}))\right)}_{\text{Negative Sample}}\Big) \\ &= \max\left(0, m - \cos(\boldsymbol{v}, \boldsymbol{u}) - \cos(\boldsymbol{u}, \boldsymbol{d}) + \cos(\boldsymbol{v}, \boldsymbol{u}') + \cos(\boldsymbol{u}', \boldsymbol{d})\right),\end{aligned}$$

- Riemannian optimization with Riemannian SGD

Represent the meaning of **sentence/text**

- Simple approach: take avg of the word2vecs of its words
- Another approach: Paragraph vector (2014, Quoc Le, Mikolov)
  - Extend word2vec to text level
  - Also two models: add paragraph vector as the input





49

Using word2vec

- Easiest way to use it is via the Gensim library for Python (tends to be slowish, even though it tries to use C optimizations like Cython, NumPy)

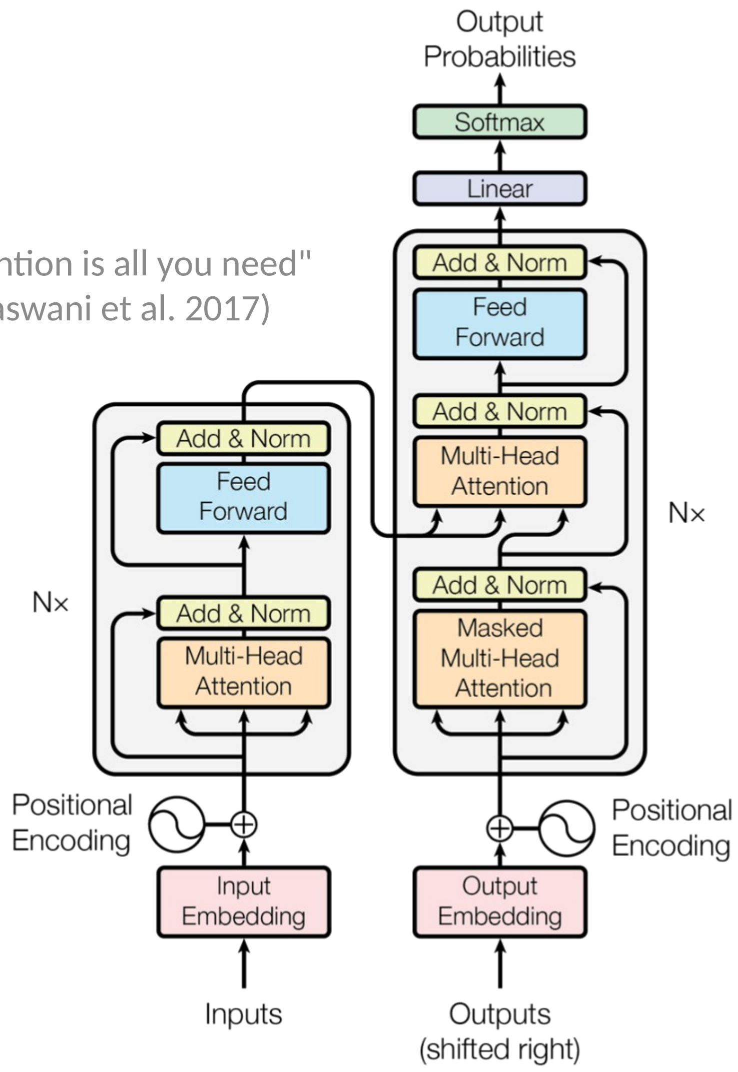
https://radimrehurek.com/gensim/models/word2vec.html

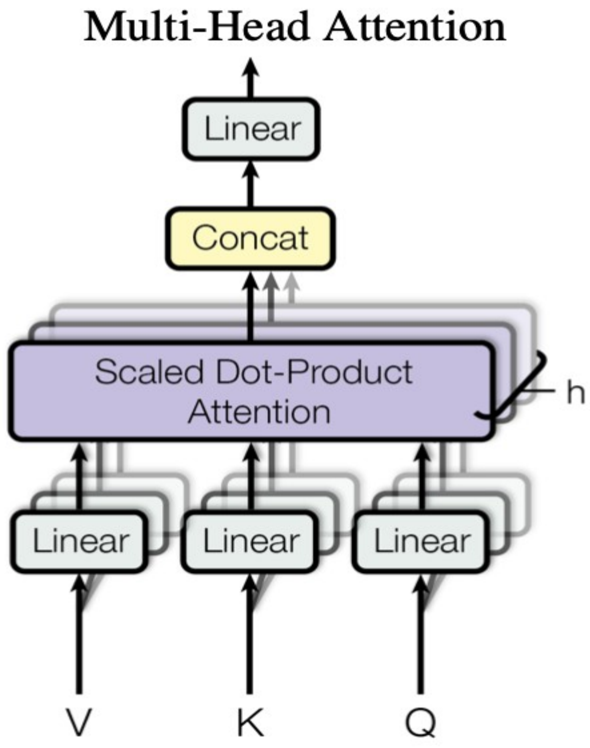
- Original word2vec C code by Google

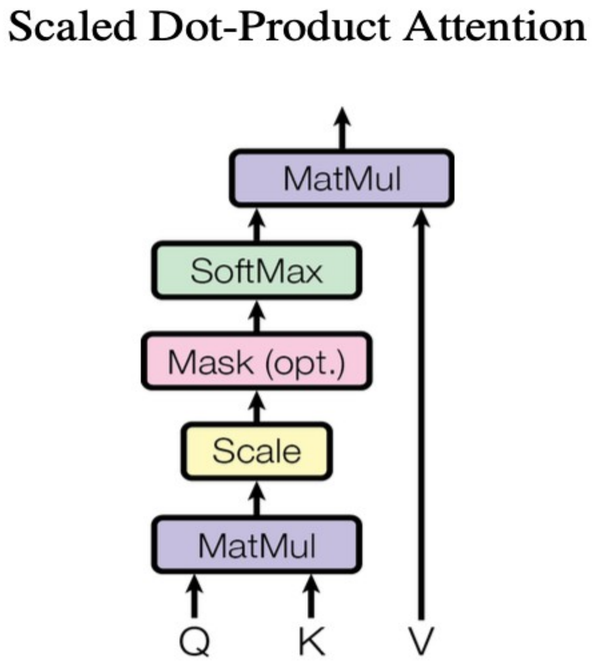
https://code.google.com/archive/p/word2vec/

Transformer

"Attention is all you need"
(Vaswani et al. 2017)

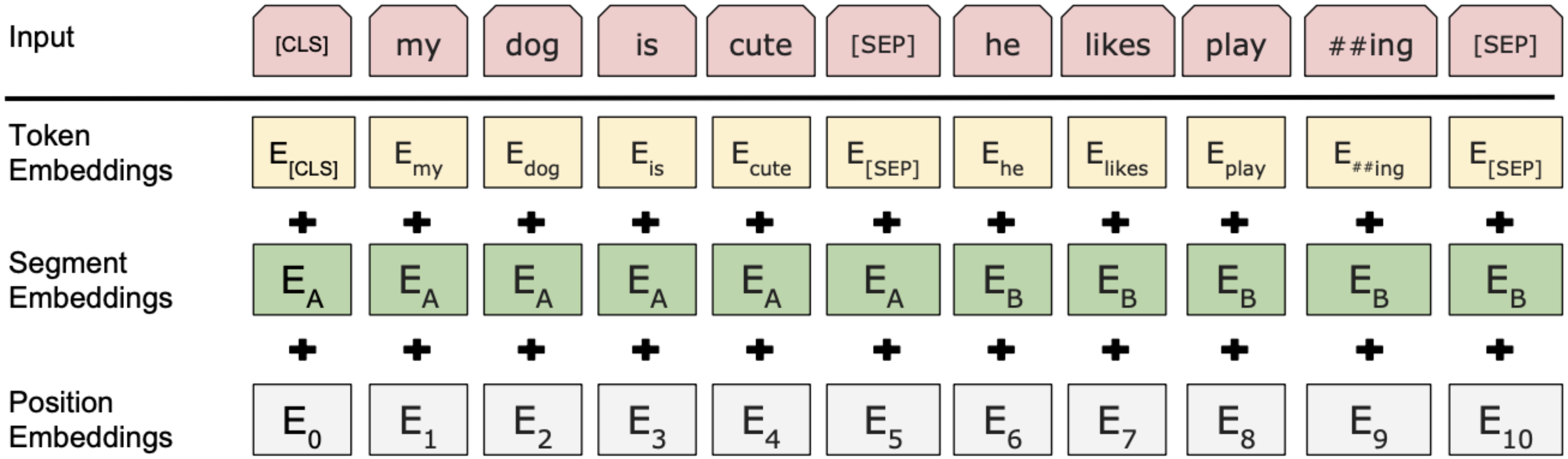






$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

BERT (Bidirectional Encoder Representations from Transformers)



- Token Embeddings: Pre-trained Word Piece embeddings
- Segment Embeddings: Sequence is divided into segments by [SEP]
- Position Embeddings: Learned Position Embeddings

BERT's architecture

- Multiple layers (e.g., L=12) of Transformer's encoder followed by an FF layer and a softmax layer.
- The input of BERT is a sequence of one or more sentences, separated by the [SEP] token. The first token is a special one [CLS] standing for Classification, e.g., [CLS] Here is a sentence. [SEP] Optionally, here is another one. [SEP]
- The context vector for the [CLS] token is the embedding for the entire input sequence.
- For each token, the final FF-softmax layer outputs a vector of size H (called hidden size). The number of attention heads for each token is denoted as A

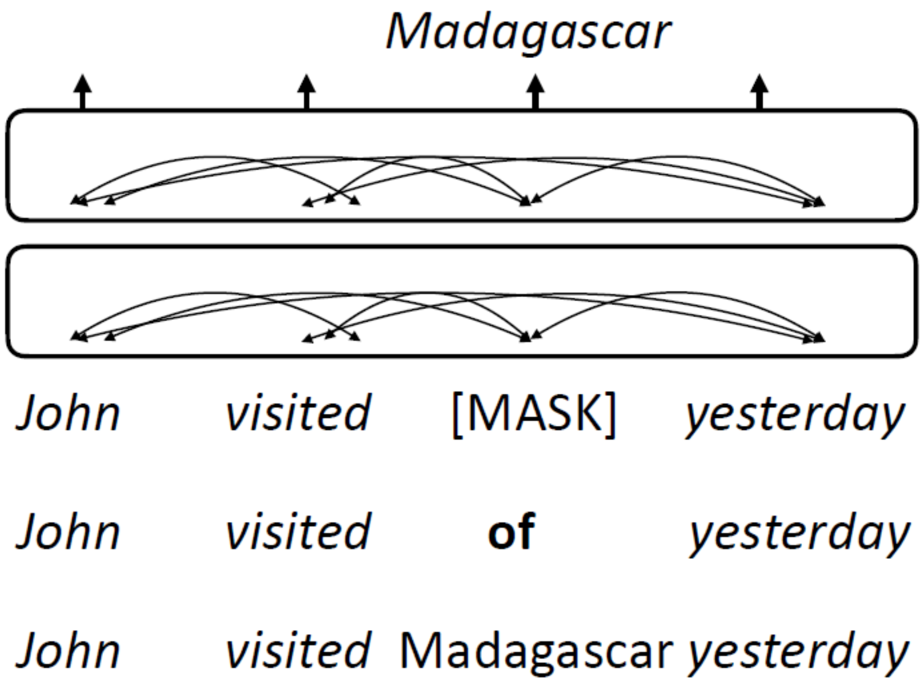
BERT

- Masked Language Model: The task is to predict the masked word based on its left and right context.
- Next Sentence Prediction: The task is to predict if B is the next sentence of A

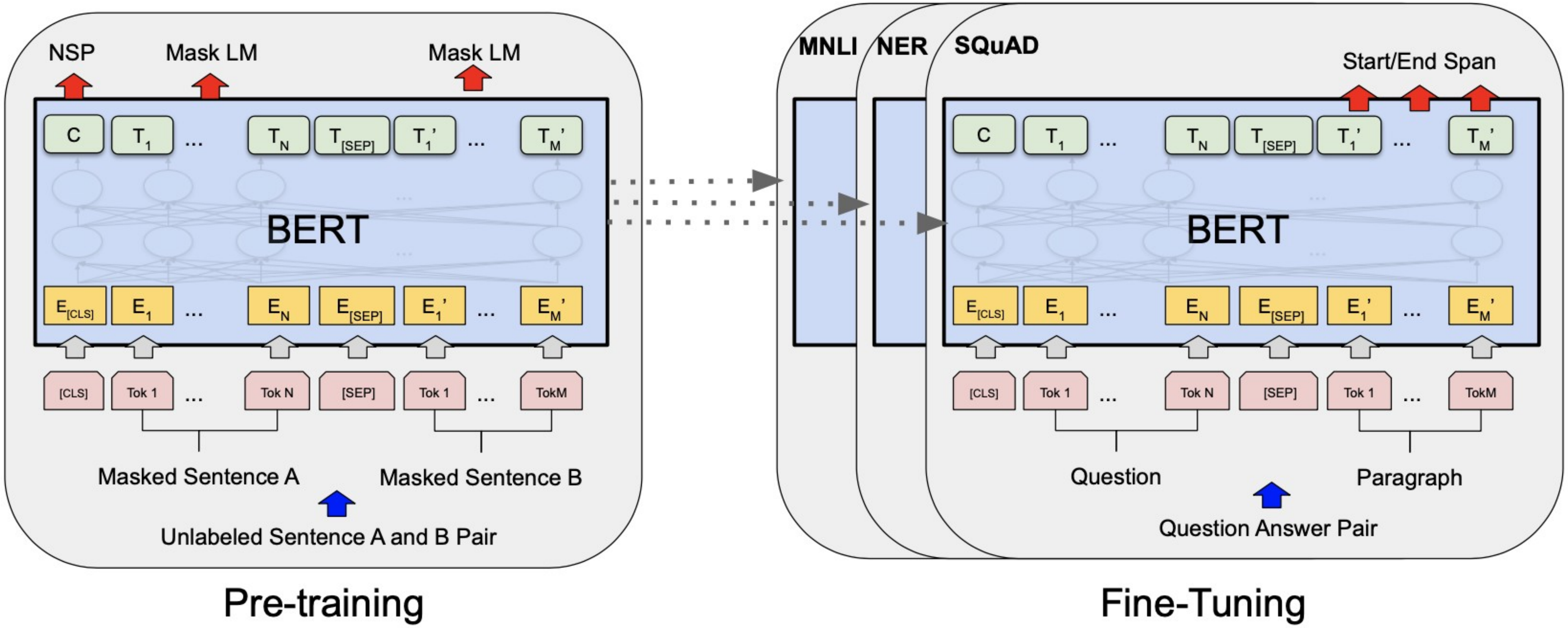
Input: [CLS] The man attended the [MASK] at Iowa state university [SEP] he was [MASK] by the lecture [SEP]
Input: [CLS] The man attended the [MASK] at Iowa state university [SEP] I [MASK] a book [SEP]

Masked Language Modeling

- BERT formula: take a chunk of text, predict 15% of the tokens
  - For 80% (of the 15%), replace the input token with [MASK]
  - For 10%, replace w/random
  - For 10%, keep same



Fine Tuning



NSP
Mask LM
Mask LM
C
T1
...
TN
T[SEP]
T1'
...
TM'
BERT
E[CLS]
E1
...
EN
E[SEP]
E1'
...
EM'
[CLS]
Tok 1
...
Tok N
[SEP]
Tok 1
...
TokM
Masked Sentence A
Masked Sentence B
Unlabeled Sentence A and B Pair
Pre-training
MNLI
NER
SQuAD
Start/End Span
C
T1
...
TN
T[SEP]
T1'
...
TM'
BERT
E[CLS]
E1
...
EN
E[SEP]
E1'
...
EM'
[CLS]
Tok 1
...
Tok N
[SEP]
Tok 1
...
TokM
Question
Paragraph
Question Answer Pair
Fine-Tuning

What can BERT do

- CLS token is used to provide classification decisions
- Sentence pair tasks (entailment): feed both sentences into BERT
- BERT can also do tagging by predicting tags at each word piece