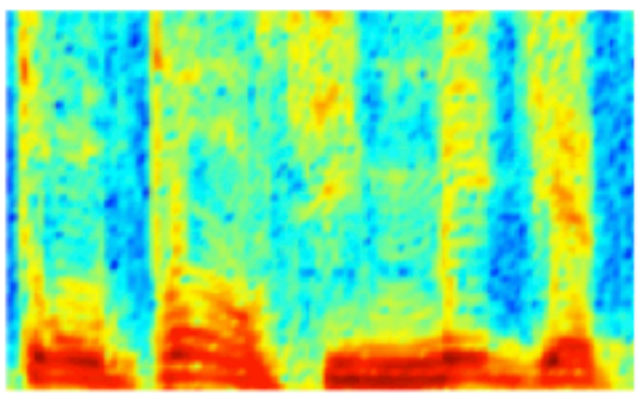
Embeddings

Subash Gandyer
Data Scientist, HealthChain

Why embeddings

AUDIO



Audio Spectrogram

DENSE

IMAGES

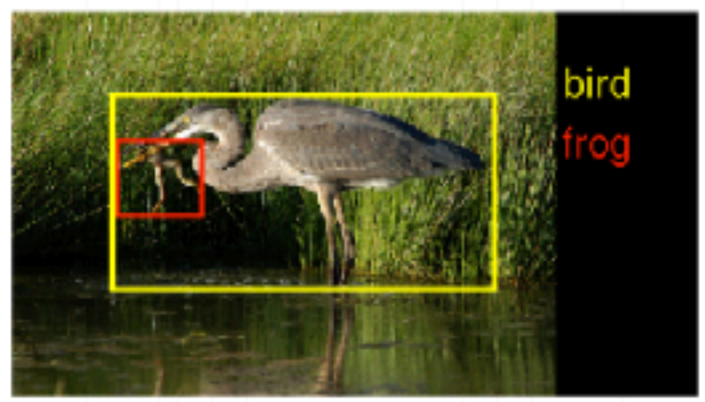
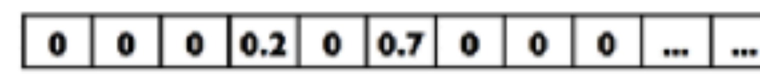


Image pixels

DENSE

TEXT



Word, context, or document vectors

SPARSE

Word Embedding

‘the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers’

the vector, which reflects the structure of the word in terms of morphology (Enriching Word Vectors with Subword Information) / word-context(s) representation (word2vec Parameter Learning Explained) / global corpus statistics (GloVe: Global Vectors for Word Representation) / words hierarchy in terms of WordNet terminology (Poincaré Embeddings for Learning Hierarchical Representations) / relationship between a set of documents and the terms they contain (Latent semantic indexing) / etc

- Idea: Capture morphological / semantic / context / hierarchical information from words

- Problem: Choosing which embedding works best

Embeddings

- One-hot encoding
- TF-IDF encoding
- Word2Vec
- Glove

One-hot encoding / CountVectorizer

- Idea: Collect a set of documents (words, sentences, paragraphs, articles) Count every occurrence of word

```
from sklearn.feature_extraction.text import CountVectorizer
# create CountVectorizer object
vectorizer = CountVectorizer()
corpus = [
          'Text of first document.',
          'Text of the second document made longer.',
          'Number three.',
          'This is number four.',
]
# learn the vocabulary and store CountVectorizer sparse matrix in X
X = vectorizer.fit_transform(corpus)
# columns of X correspond to the result of this method
vectorizer.get_feature_names() == (
    ['document', 'first', 'four', 'is', 'longer',
     'made', 'number', 'of', 'second', 'text',
     'the', 'this', 'three'])
# retrieving the matrix in the numpy form
X.toarray()
# transforming a new document according to learn vocabulary
vectorizer.transform(['A new document.']).toarray()
```

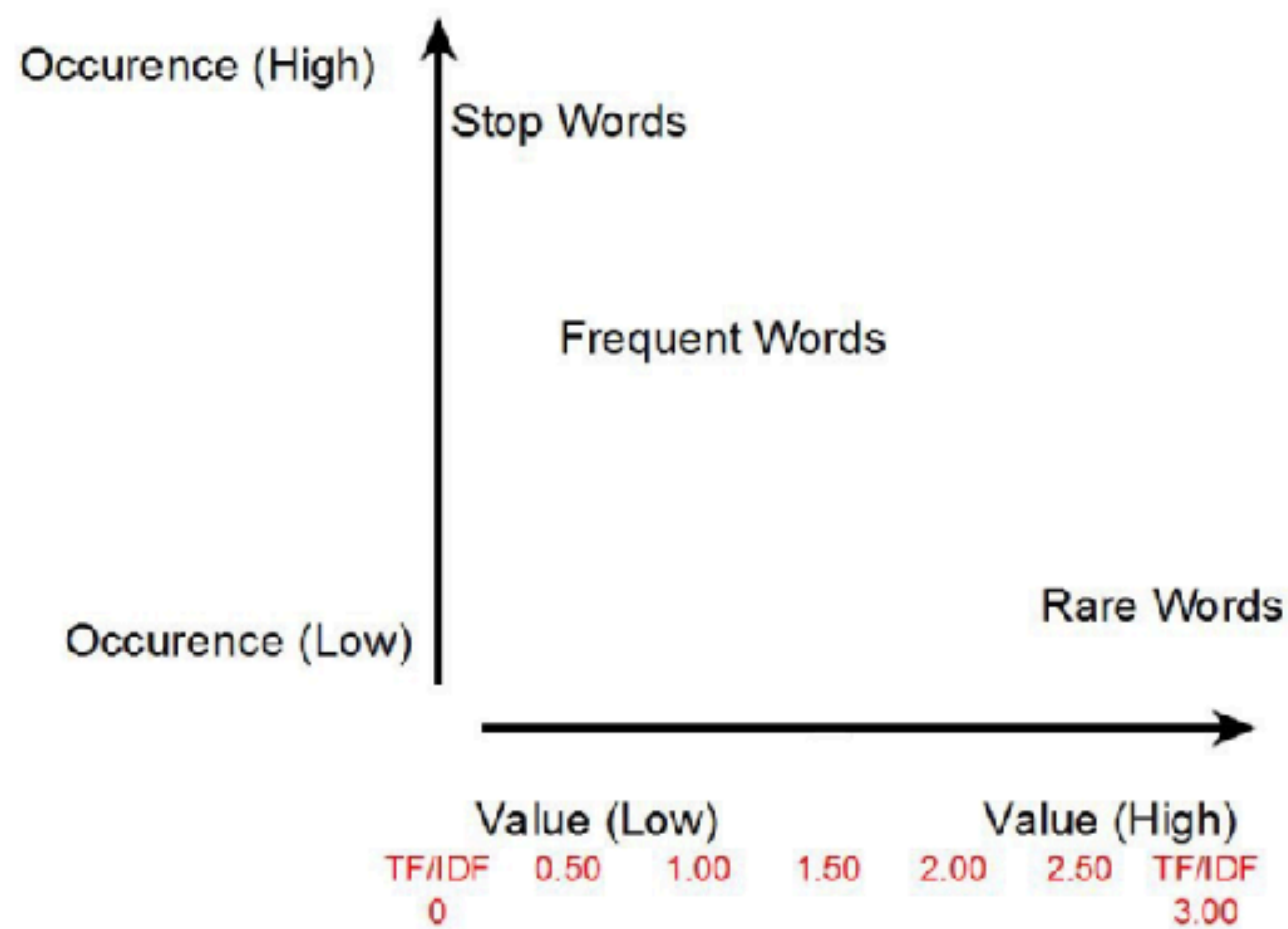
TF-IDF encoding

- Idea: Commonly occurring words don't carry useful information but rare words do

$$tfidf(term, document) = tf(term, document) \cdot idf(term)$$

$$tf(term, document) = \frac{n_i}{\sum_{k=1}^{V} n_k}$$

$$idf(term) = \log \frac{N}{n_t}$$



TF-IDF Encoding

```
from sklearn.feature_extraction.text import TfidfTransformer


# create tf-idf object
transformer = TfidfTransformer(smooth_idf=False)


# X can be obtained as X.toarray() from the previous snippet
X = [[3, 0, 1],
     [5, 0, 0],
     [3, 0, 0],
     [1, 0, 0],
     [3, 2, 0],
     [3, 0, 4]]


# learn the vocabulary and store tf-idf sparse matrix in tfidf
tfidf = transformer.fit_transform(counts)


# retrieving matrix in numpy form as we did it before
tfidf.toarray()
```

Uninterested

prefix stem suffix

Prefix/Suffix/Root	Meaning	Example	Prefix/Suffix/Root	Meaning	Example
1. acou-	hear	acoustics	41. fiss-	split	fission
2. aer-, aero-	air	aerodynamics	42. -flect	bend	reflection
3. alt-, alti-, alto-	height	altitude	43. flu-	flow	flux
4. ampli-	large	amplitude	44. for-, fort-	strong, strength	fortification
5. angl, angul-	angle	triangulation	45. -fract	break	fracture
6. ann-, annu-	year	annual	46. frict-	rub	friction
7. ap-, apo-	away from	apogee	47. fus-	melting	fusion
8. astr-, astro-, aster-	star	astronomical	48. -fy	to make	magnify
9. -ation, -ition, -tion	process, result	condensation	49. ge-, geo, -gee	earth	geothermal
10. atmo-	air	atmosphere	50. -gen	to produce, origin	hydrogen
11. -atude, -itude, -tude	condition, state of	amplitude	51. -gon	angle	hexagonal
12. aud-, audio-	hear	audiologist	52. grad-	step	gradient
13. bar-, baro-	heavy	barometric	53. -gram	written record	cardiogram
14. calor-	heat	calorimetry	54. -graph	recording	spectrograph
15. can-, cand-	glow white	incandescence	55. grav-	heavy, weighty	gravitational
16. cap-, capac-	hold	capacitor	56. gyr-	rotate	gyroscope
17. carb-, carbo-	carbon	carbohydrate	57. helio-	sun	heliocentric
18. -ced, -ceed, -cess	going	recessional	58. hemi-	half	hemisphere
19. -celer	swift, to hasten	accelerometer	59. hepta-	seven	heptathlon
20. centr-, centri-	center	centripetal	60. hetero-	unlike, different	heterotroph
21. chem-, chemo-	transmutation	chemical	61. hexa-	six	hexagon
22. chrom-, chromo-	color	chromatograph	62. homo-	same, like	homophone
23. chron-, chrono-	time	chronograph	63. hydr-, hydro-	water	hydrothermal
24. circ-, circl-	circle	circulation	64. -ic, -tic	person	lunatic
25. con-, com-	with, together	compound	65. -ical	pertaining to	mechanical
26. con-, coni-	cone	conical	66. -ician, -icist	specialist	physicist
27. cosm-, cosmo-	universe	cosmological	67. -ics, -tics-	skill	mechanics
28. cry-, cryo-	cold	cryogenic	68. ign-	fire	ignition
29. cycly-, cyclo-	circle	cyclone	69. -ile	pertaining to, thing	projectile
30. deca-	ten	dodecagon	70. infra-	below	infrasonic
31. di-	two	dipole	71. inter-	between	interstitial
32. -duc, -duce, -duct	to lead	aqueduct	72. intra-	within	intramural
33. dyn-, dynam-	power	hydrodynamic	73. iso-	equal	isobaric
34. elect-, electro-	electric, amber	electrician	74. -ist	person	chemist
35. -ence	state of	luminescence	75. -istry	skill	artistry
36. end-, endo-, ento-	inside, within	endothermic	76. -ject	to throw	projectile
37. equa-, equi-, equ-	equal	equilibrium	77. kin-, kine-	motion, movement	kinematics
38. erg-, ergo-	work	ergonomics	78. lept-	small	lepton
39. ex-, exo-	outside, out of	exogenic	79. lev-	to raise	levitation
40. ferro-	iron	ferromagnetic	80. libr-, libri-	weight	equilibrium

I love NLP and I like dogs

```
I = [_ _ _ _ _ _]
Love = [_ _ _ _ _ _]
NLP = [_ _ _ _ _ _]
And = [_ _ _ _ _ _]
Like = [_ _ _ _ _ _]
Dogs = [_ _ _ _ _ _]
```

Vector Filling Approaches

1. Word Count
2. Co-occurrence

and so on

I love NLP and I like dogs

	I	Love	NLP	And	Like	Dogs
I	0	1	0	1	1	0
Love	1	0	1	0	0	0
NLP	0	1	0	1	0	0
And	1	0	1	0	0	0
Like	1	0	0	0	0	1
Dogs	0	0	0	0	1	0

I = [0 1 0 1 1 0]

Love = [1 0 1 0 0 0]

NLP = [0 1 0 1 0 0]

And = [1 0 1 0 0 0]

Like = [1 0 0 0 0 1]

Dogs = [0 0 0 0 1 0]

Distributional Hypothesis

Words that appear in same context share semantic meaning

1. Count-based method (Latent Semantic Analysis)

Count-based methods compute the statistics of how often some word co-occurs with its neighbour words in a large text corpus, and then map these count-statistics down to a small, dense vector for each word.

2. Predictive method (Neural Probabilistic Language Model)

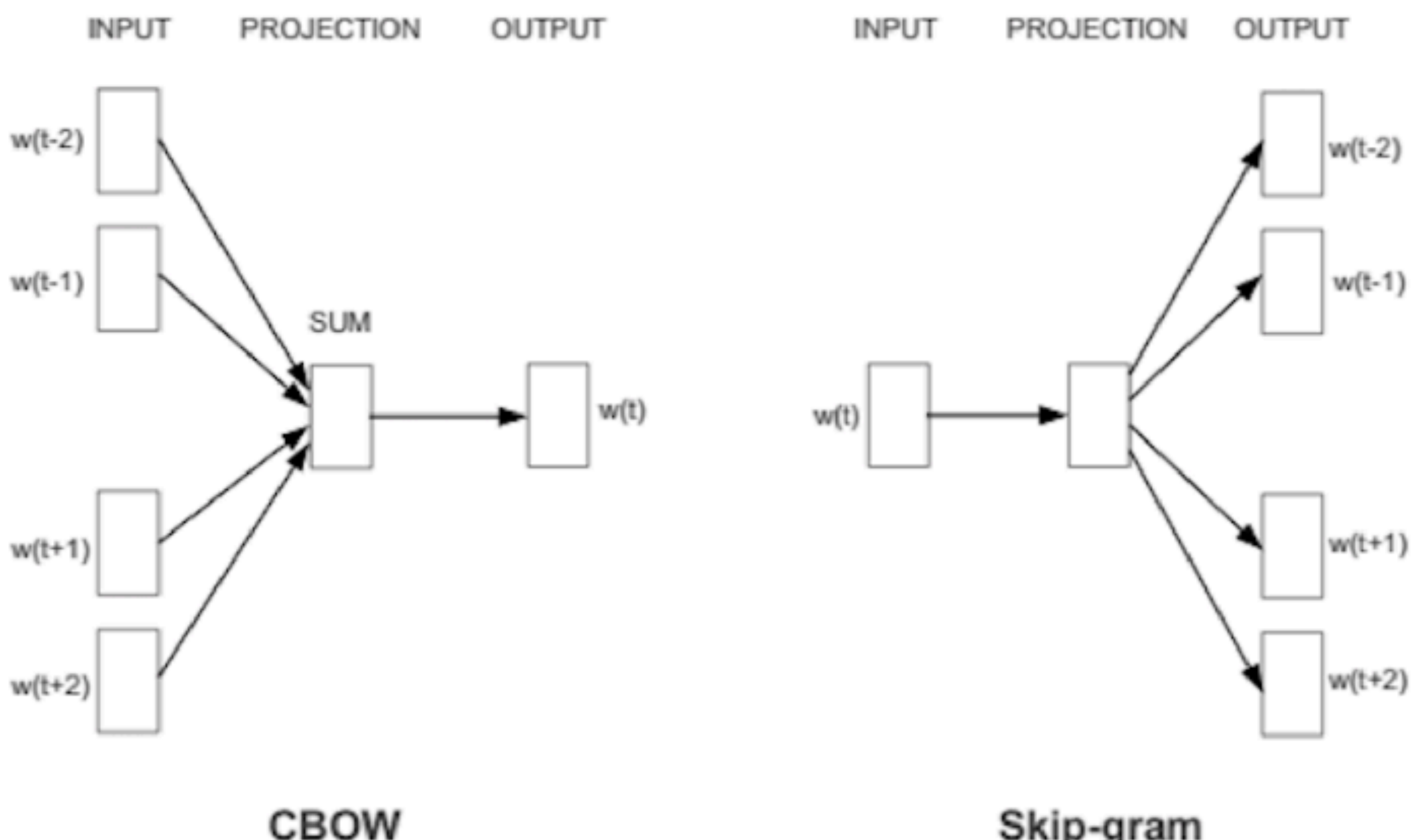
Predictive models directly try to predict a word from its neighbours in terms of learned small, dense embedding vectors.

Word2Vec —> Predictive Method

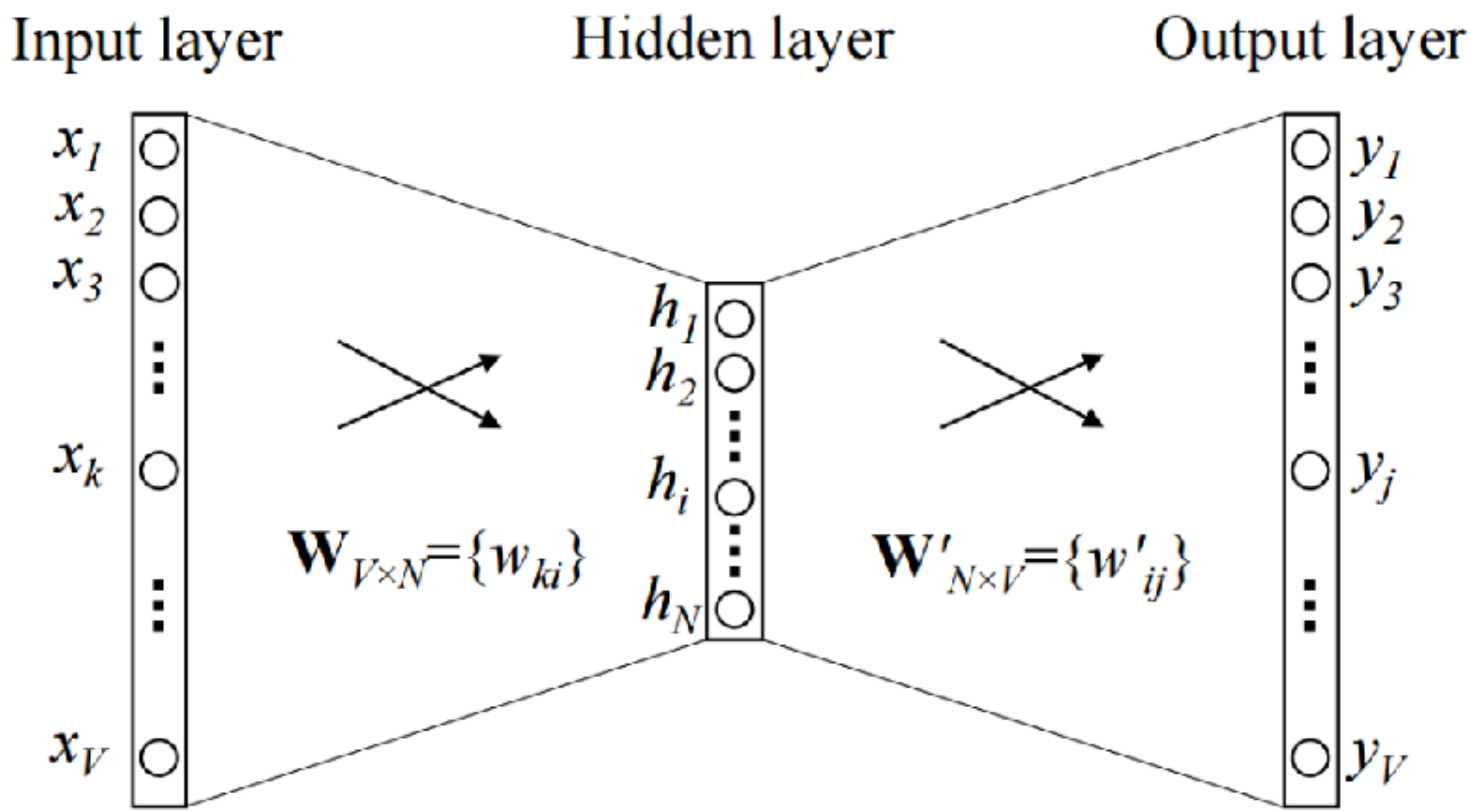
Word2Vec

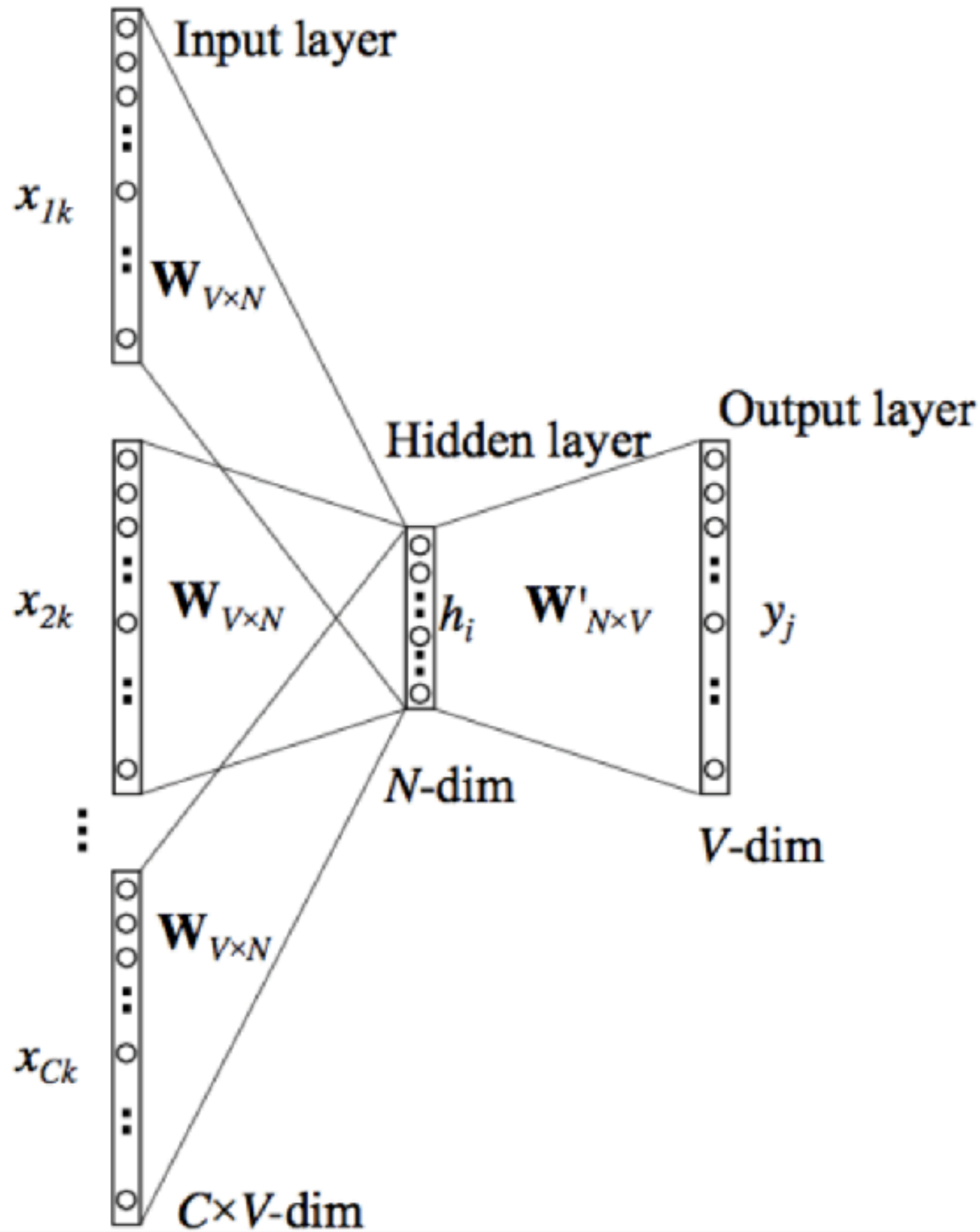
Word2vec is to group the vectors of similar words together in vectorspace. Word2vec creates vectors that are distributed numerical representations of word features, features such as the context of individual words.

Word2vec is similar to an autoencoder, encoding each word in a vector, but rather than training against the input words through reconstruction, as a restricted Boltzmann machine does, word2vec trains words against other words that neighbor them in the input corpus.



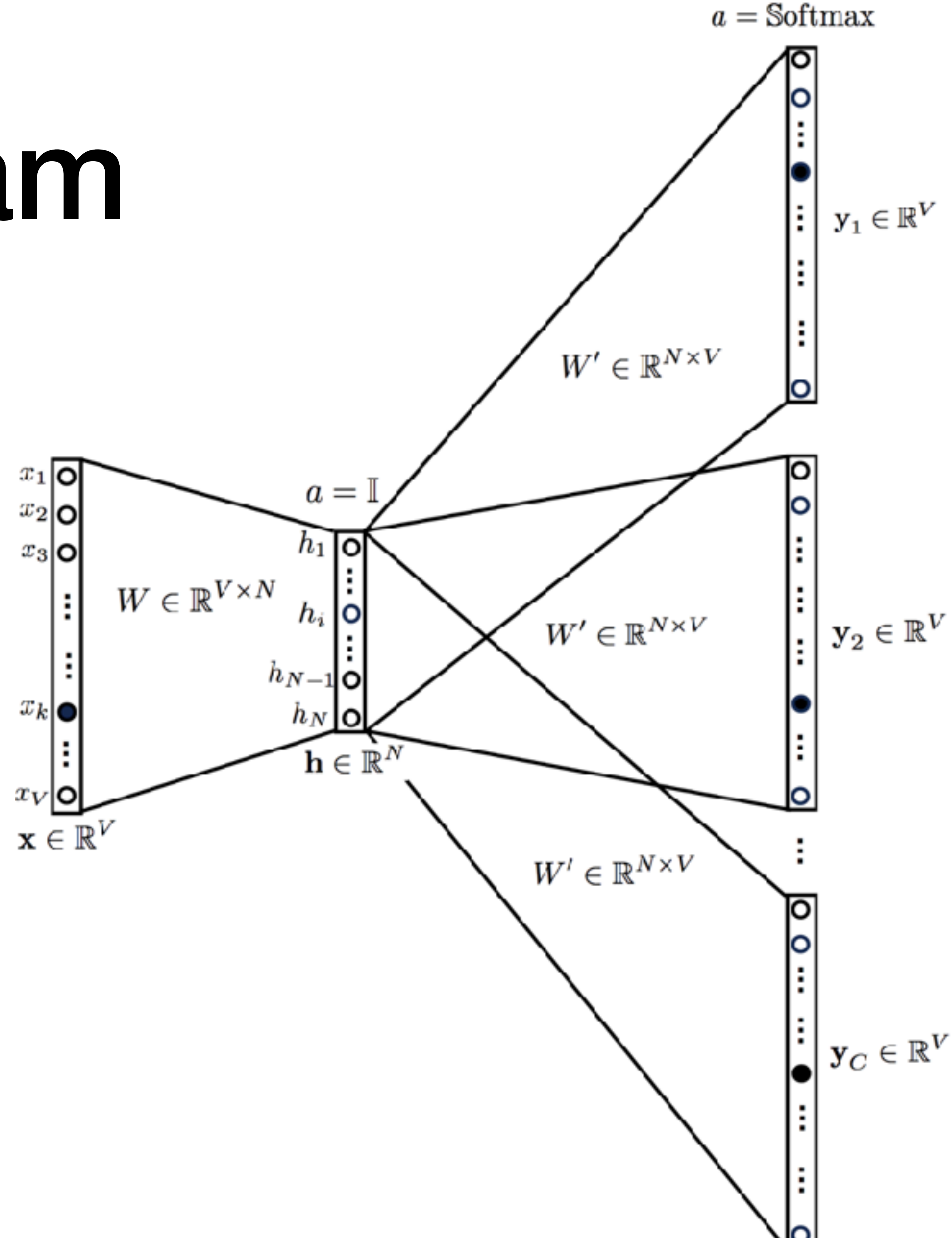
CBoW


Input layer
Hidden layer
Output layer
x_1
x_2
x_3
x_k
x_V
h_1
h_2
h_i
h_N
y_1
y_2
y_3
y_j
y_V
$\mathbf{W}_{V\times N}=\{w_{ki}\}$
$\mathbf{W}'_{N\times V}=\{w'_{ij}\}$



Input layer
x_{1k}
$\mathbf{W}_{V\times N}$
Hidden layer
Output layer
x_{2k}
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W}'_{N\times V}$
y_j
N-dim
V-dim
$\mathbf{W}_{V\times N}$
x_{Ck}
$C\times V$-dim

Skip-Gram


$a = \mathrm{Softmax}$
$\mathbf{y}_1 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
x_1
x_2
x_3
x_k
x_V
$\mathbf{x} \in \mathbb{R}^V$
$W \in \mathbb{R}^{V \times N}$
$a = \mathbb{I}$
h_1
h_i
h_{N-1}
h_N
$\mathbf{h} \in \mathbb{R}^N$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_2 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_C \in \mathbb{R}^V$

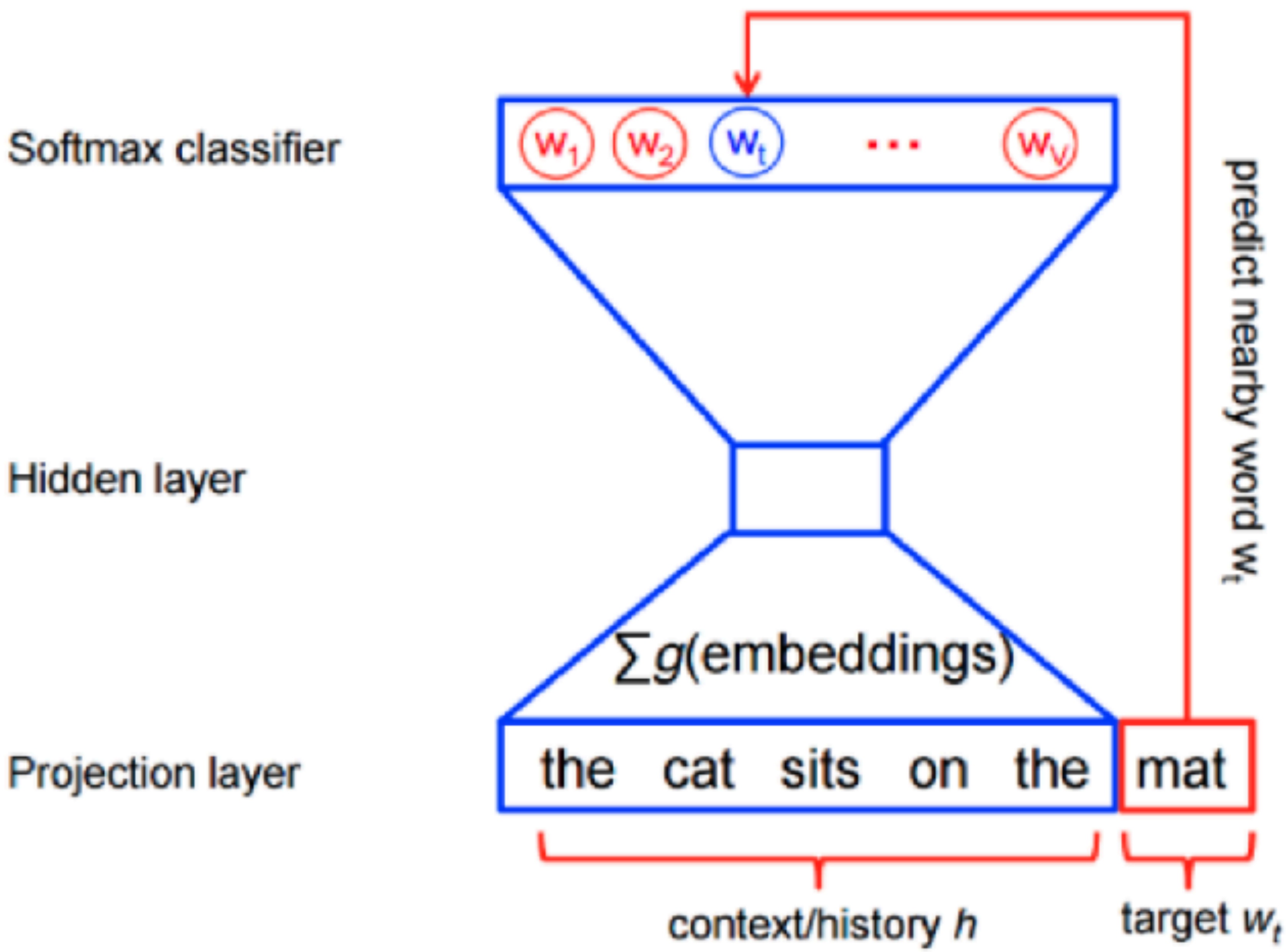
$$
\begin{aligned}
P(w_t|h) &= \text{softmax}(\text{score}(w_t, h)) \\
&= \frac{\exp\{\text{score}(w_t, h)\}}{\sum_{\text{Word w' in Vocab}} \exp\{\text{score}(w', h)\}}
\end{aligned}
$$

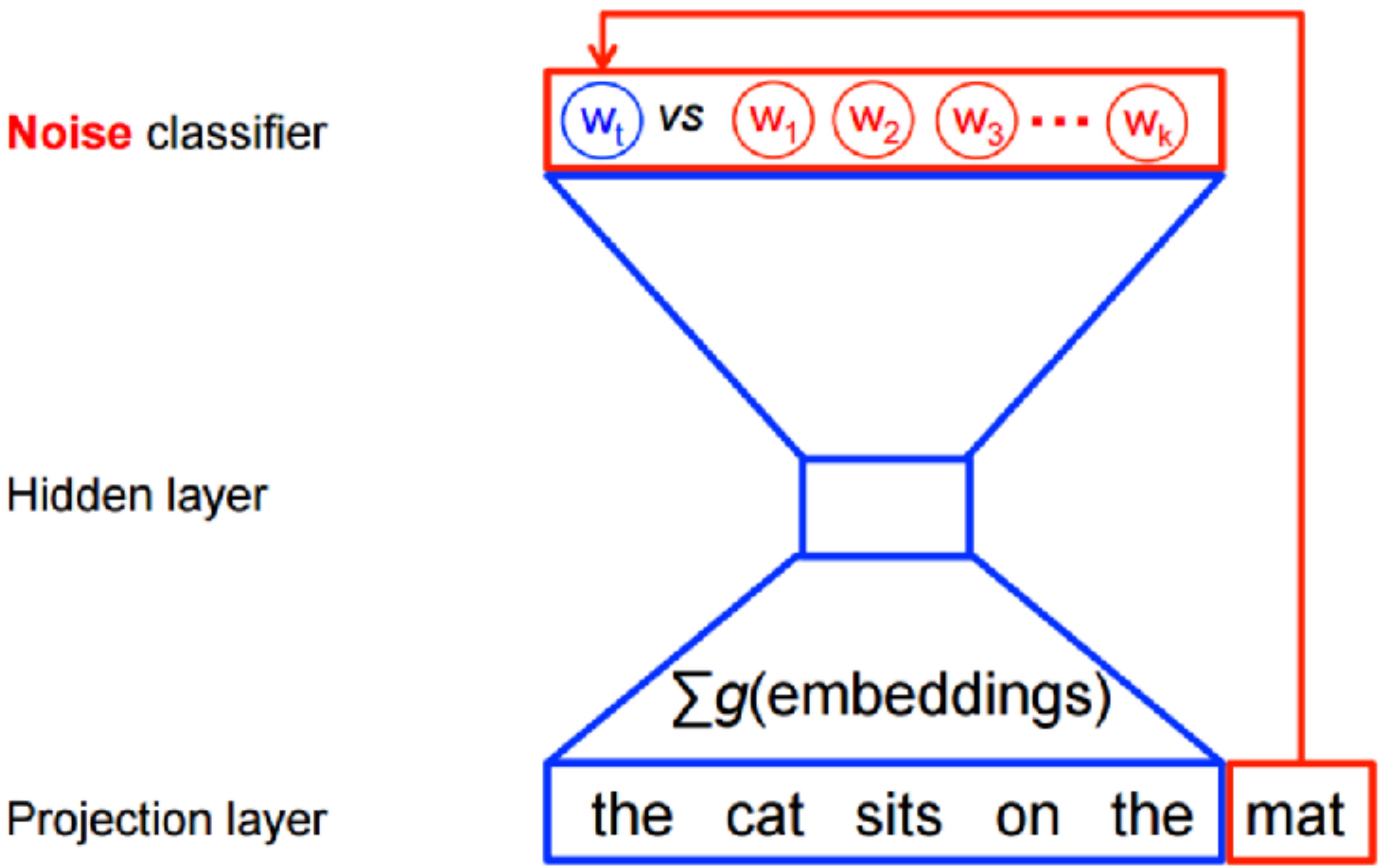
$$
\begin{aligned}
J_{\text{ML}} &= \log P(w_t|h) \\
&= \text{score}(w_t, h) - \log\left(\sum_{\text{Word w' in Vocab}} \exp\{\text{score}(w', h)\}\right)
\end{aligned}
$$

$$
argmax_{\theta} \frac{1}{T} \sum_{t=1}^{T} \sum_{j \in c, j != 0} logp(w_{t+j}|w_t; \theta)
$$

Embedding Lookup Matrix

$$p(w_i|w_t;\theta) = \frac{exp(\theta w_i)}{\sum_t exp(\theta w_t)}$$




Noise classifier
Hidden layer
Projection layer
w_t vs w_1 w_2 w_3 ... w_k
$\sum g$(embeddings)
the cat sits on the
mat

Negative Sampling

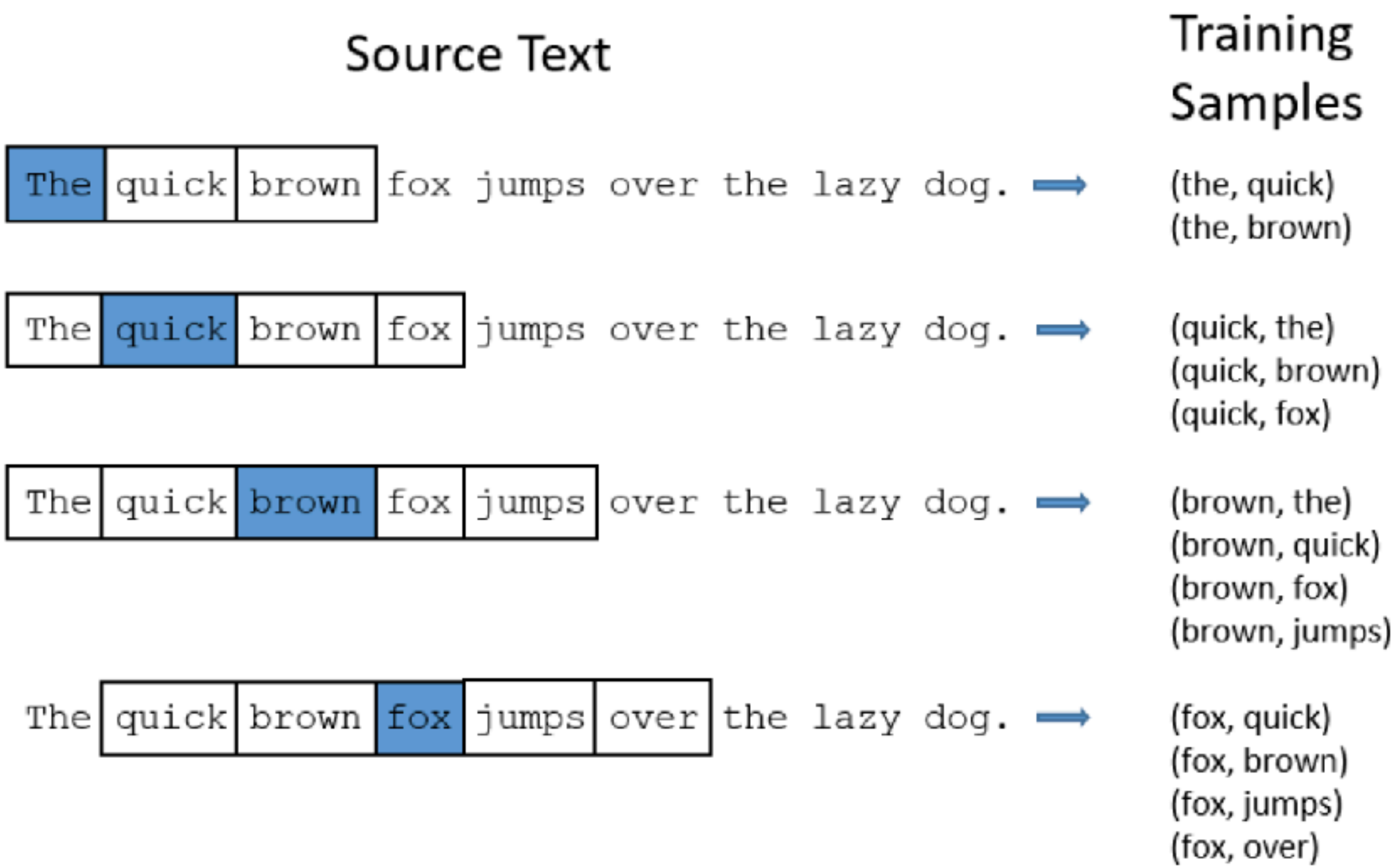
$$J_{\text{NEG}} = \log Q_\theta(D=1|w_t,h) + k \mathop{\mathbb{E}}_{\tilde{w}\sim P_{\text{noise}}} [\log Q_\theta(D=0|\tilde{w},h)]$$

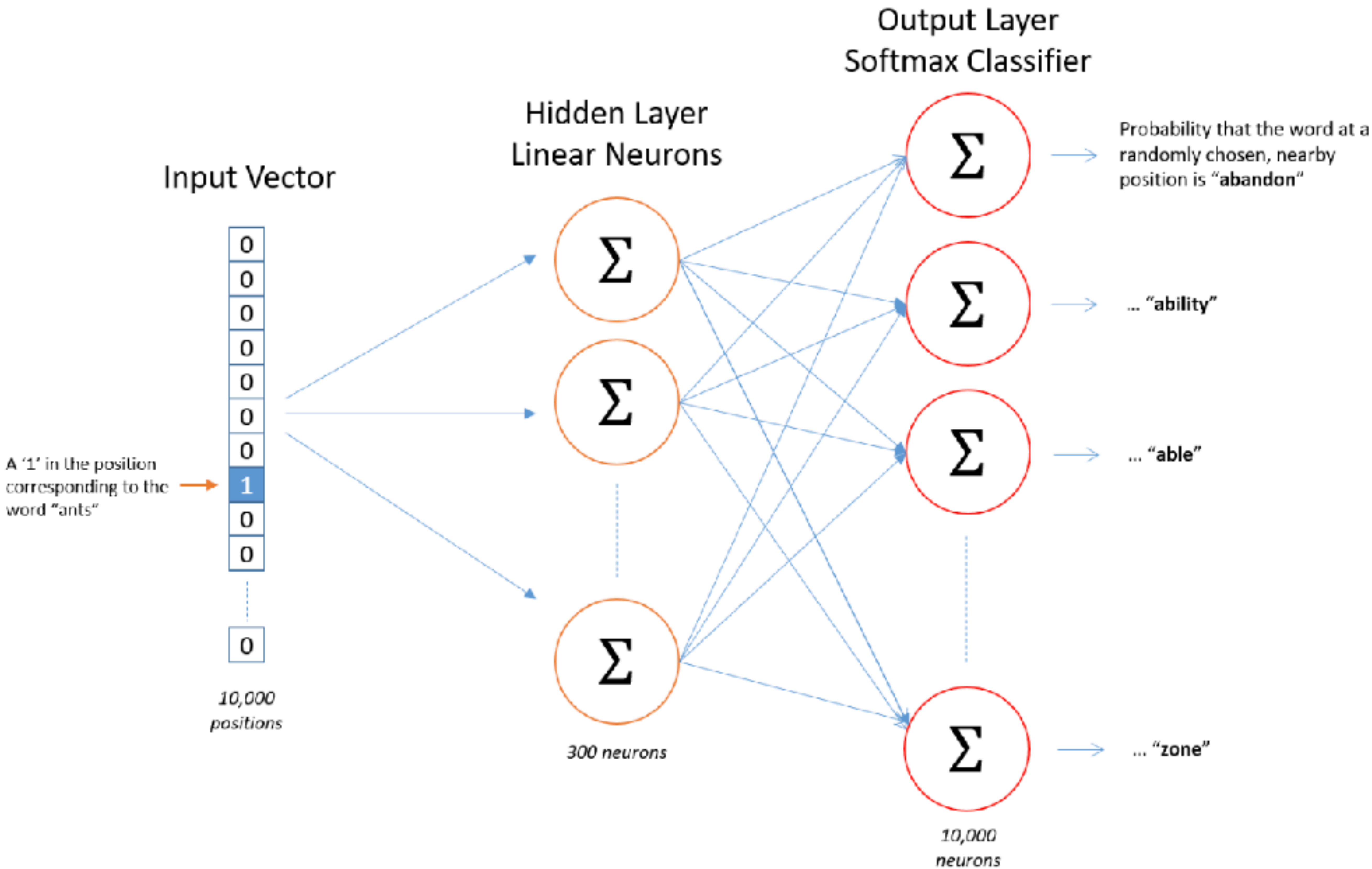
where $Q_\theta(D=1|w,h)$ is the binary logistic regression probability under the model of seeing the word w in the context h in the dataset D, calculated in terms of the learned embedding vectors θ. In practice we approximate the expectation by drawing k contrastive words from the noise distribution (i.e. we compute a Monte Carlo average).

$$J^{(t)}_{\text{NEG}} = \log Q_\theta(D=1|\text{the, quick}) + \log(Q_\theta(D=0|\text{sheep, quick}))$$

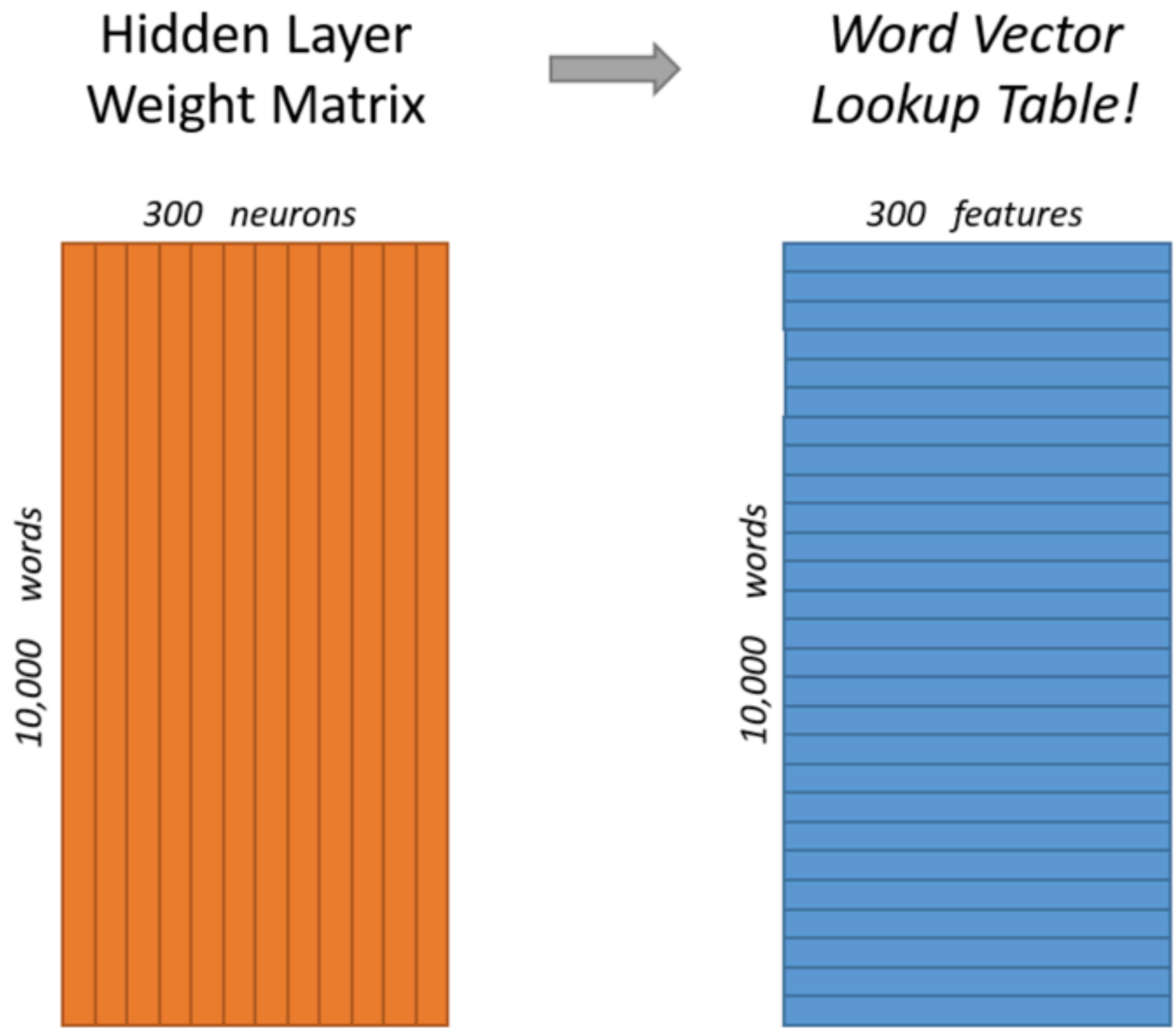
Word2Vec Steps

1. Take a 3 layer neural network. (1 input layer + 1 hidden layer + 1 output layer)
2. Feed it a word and train it to predict its neighbouring word.
3. Remove the last (output layer) and keep the input and hidden layer.
4. Now, input a word from within the vocabulary. The output given at the hidden layer is the '*word embedding*' of the input word.

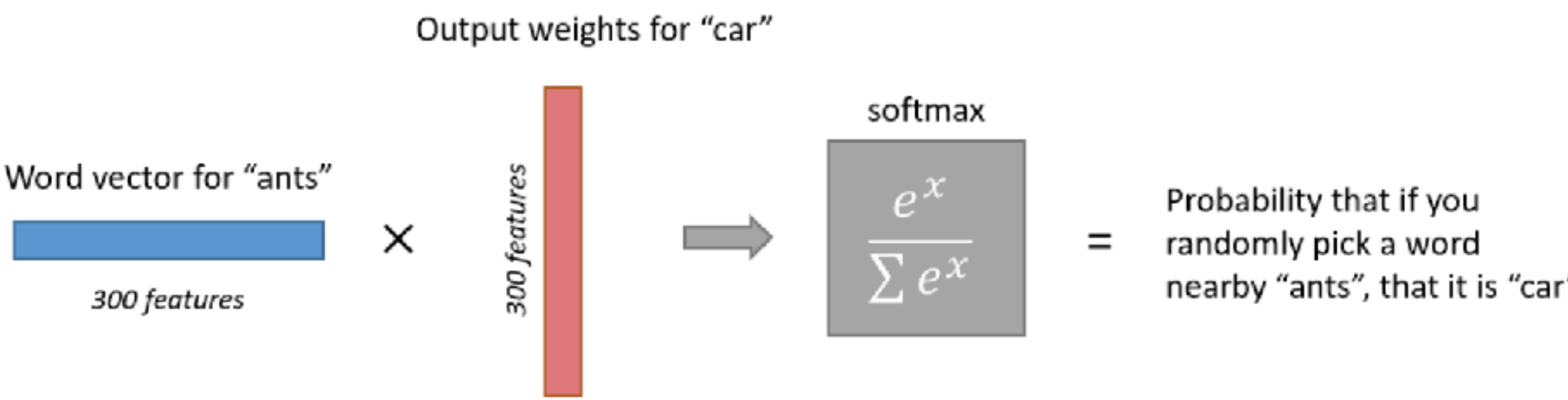

Source Text
Training Samples
The quick brown fox jumps over the lazy dog.
(the, quick)
(the, brown)
The quick brown fox jumps over the lazy dog.
(quick, the)
(quick, brown)
(quick, fox)
The quick brown fox jumps over the lazy dog.
(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)
The quick brown fox jumps over the lazy dog.
(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)



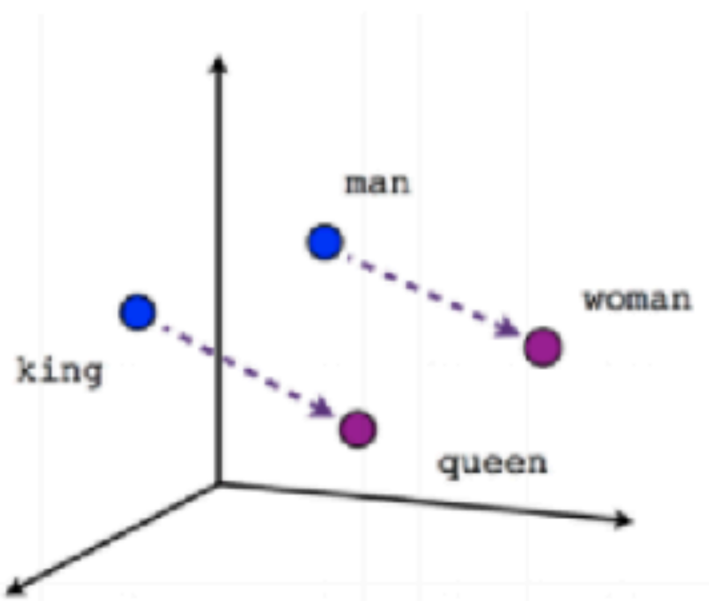
Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector
0
0
0
0
0
0
0
1
0
0
0
A '1' in the position corresponding to the word "ants"
10,000 positions
300 neurons
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"
10,000 neurons


Hidden Layer
Weight Matrix
Word Vector
Lookup Table!
300 neurons
10,000 words
300 features
10,000 words

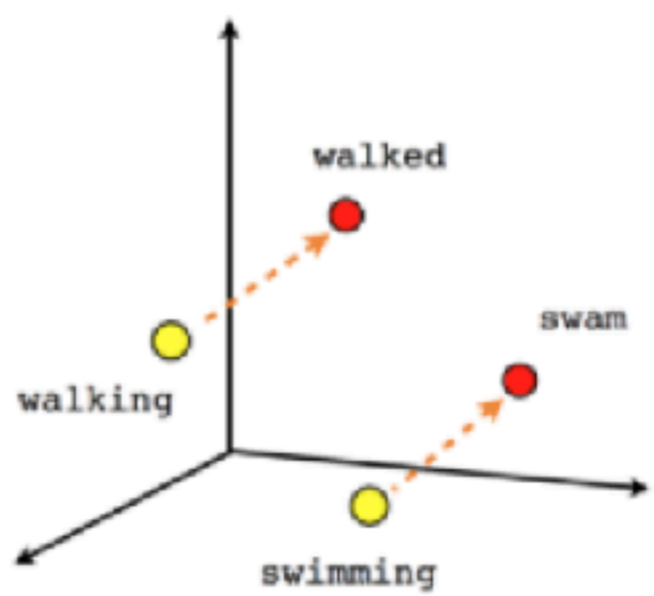
$$
\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}
$$



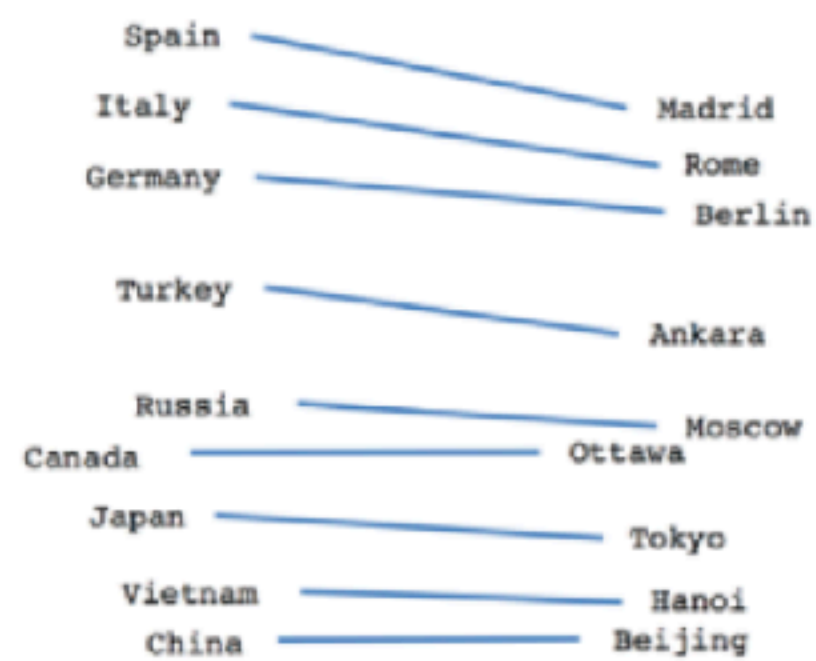
Word2Vec Relationships



Male-Female



Verb tense



Country-Capital

Implementation in Tensorflow

```
embeddings = tf.Variable(
    tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0))
```

```
nce_weights = tf.Variable(
  tf.truncated_normal([vocabulary_size, embedding_size],
                      stddev=1.0 / math.sqrt(embedding_size)))
nce_biases = tf.Variable(tf.zeros([vocabulary_size]))
```

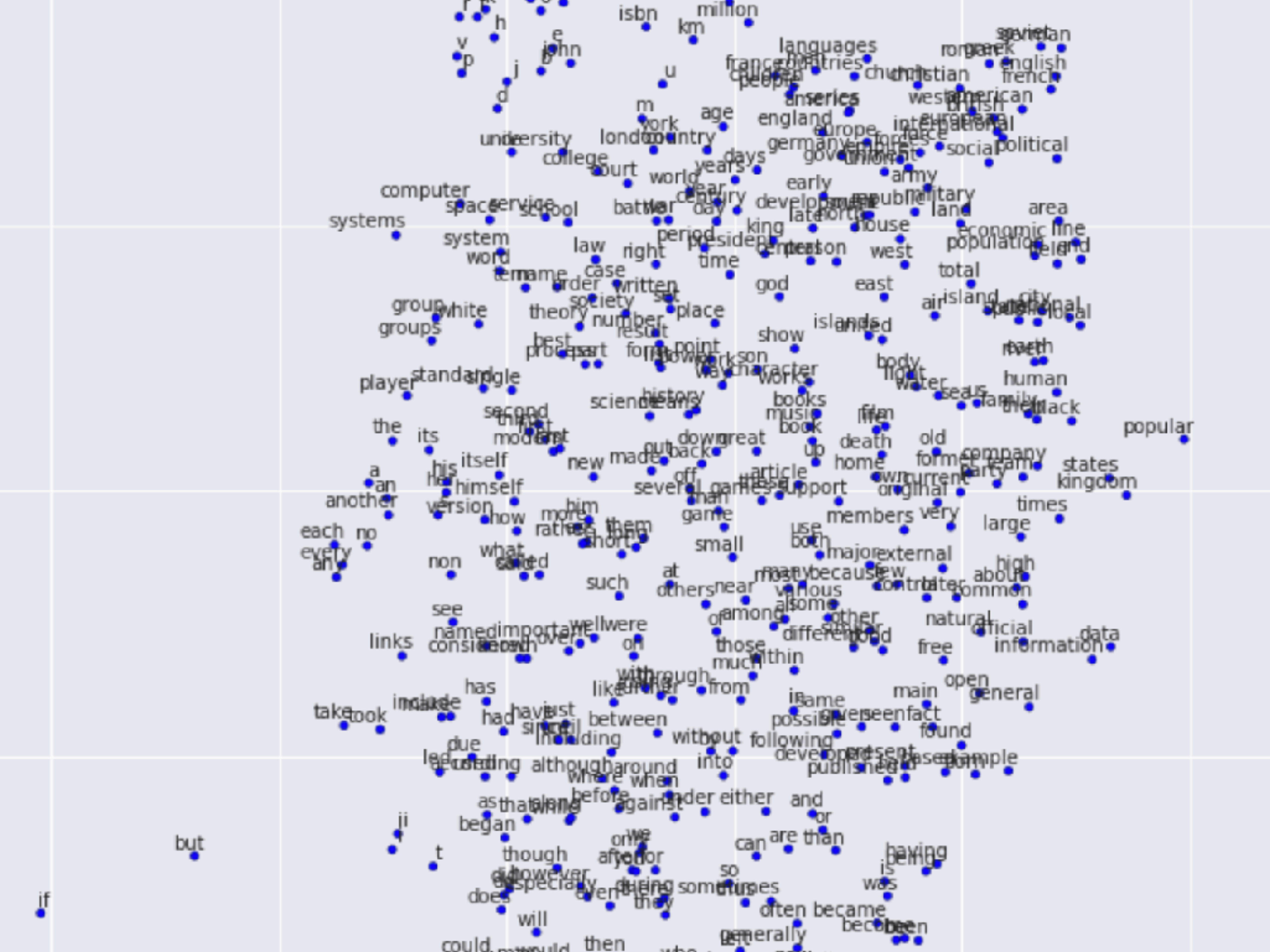
```
# Placeholders for inputs
train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
```

```
embed = tf.nn.embedding_lookup(embeddings, train_inputs)
```

```
# Compute the NCE loss, using a sample of the negative labels each time.
loss = tf.reduce_mean(
  tf.nn.nce_loss(weights=nce_weights,
                 biases=nce_biases,
                 labels=train_labels,
                 inputs=embed,
                 num_sampled=num_sampled,
                 num_classes=vocabulary_size))
```

```
# We use the SGD optimizer.
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1.0).minimize(loss)
```

```
for inputs, labels in generate_batch(...):
  feed_dict = {train_inputs: inputs, train_labels: labels}
  _, cur_loss = session.run([optimizer, loss], feed_dict=feed_dict)
```

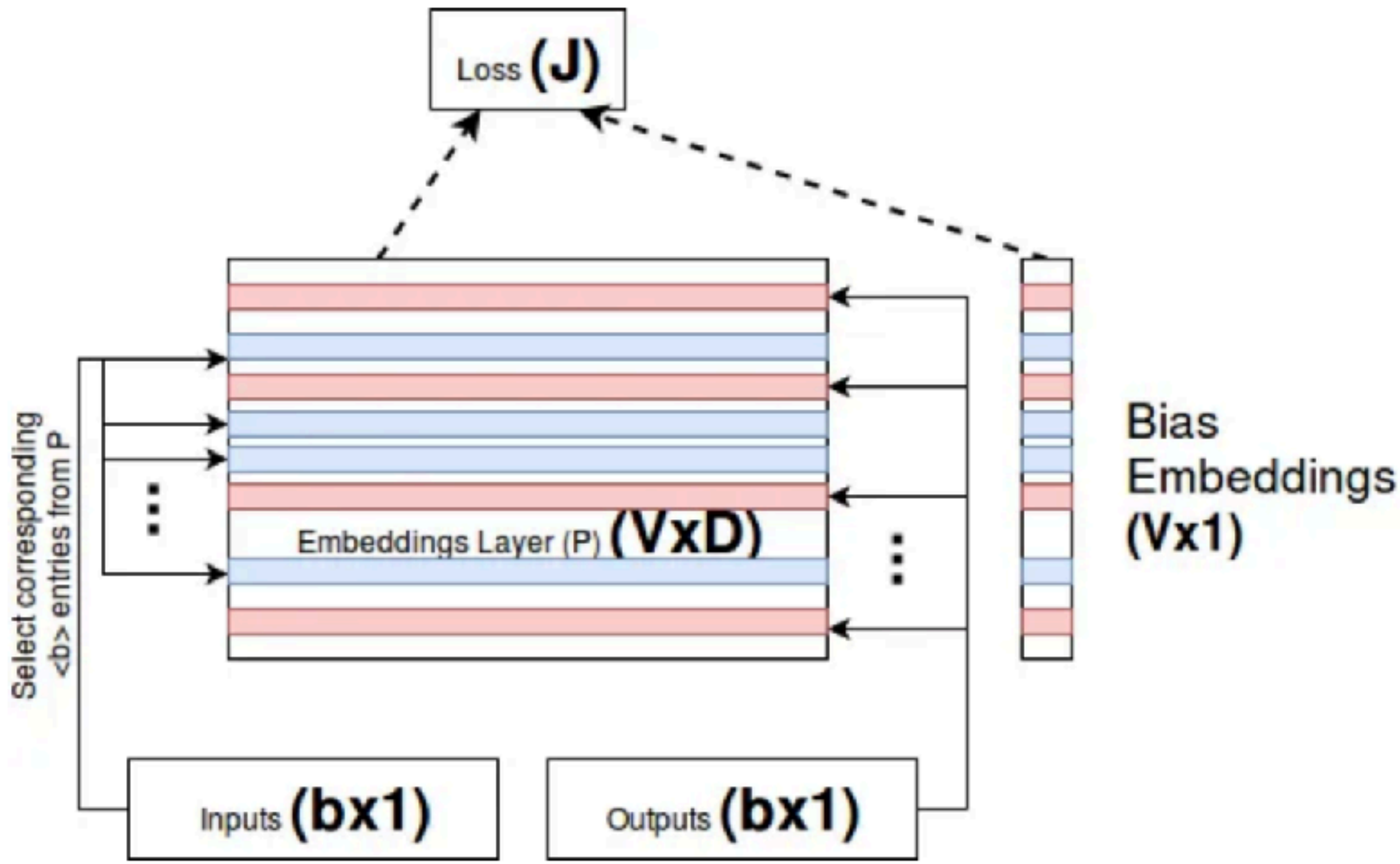


isbn
million
km
h
e
john
v
p
j
u
languages
english
french
d
m
american
age
york
england
international
political
social
london
country
days
college
court
world
years
early
army
computer
space
service
school
systems
military
land
area
king
house
period
system
law
right
word
west
population
time
case
written
god
east
total
group
white
theory
society
place
island
groups
number
result
show
islands
best
point
son
character
body
human
player
standard
single
works
water
science
history
books
music
second
the
its
modern
old
popular
death
down
great
new
made
back
home
company
itself
his
himself
article
support
original
states
kingdom
another
an
version
him
game
members
very
times
large
each
no
rather
them
short
small
both
major
external
every
any
non
what
such
at
others
near
various
few
high
about
common
see
among
some
natural
official
data
links
named
important
well
were
on
those
within
much
free
information
open
main
has
from
general
take
took
make
had
between
possible
fact
found
due
without
following
although
around
into
published
example
where
when
before
against
either
and
or
as
began
but
can
are
than
having
being
t
though
after
so
is
was
if
does
especially
sometimes
often
became
will
generally
been
could
then

GloVe

- Idea: Ratios between probabilities of words appearing next to each other carry more information that individual probabilities

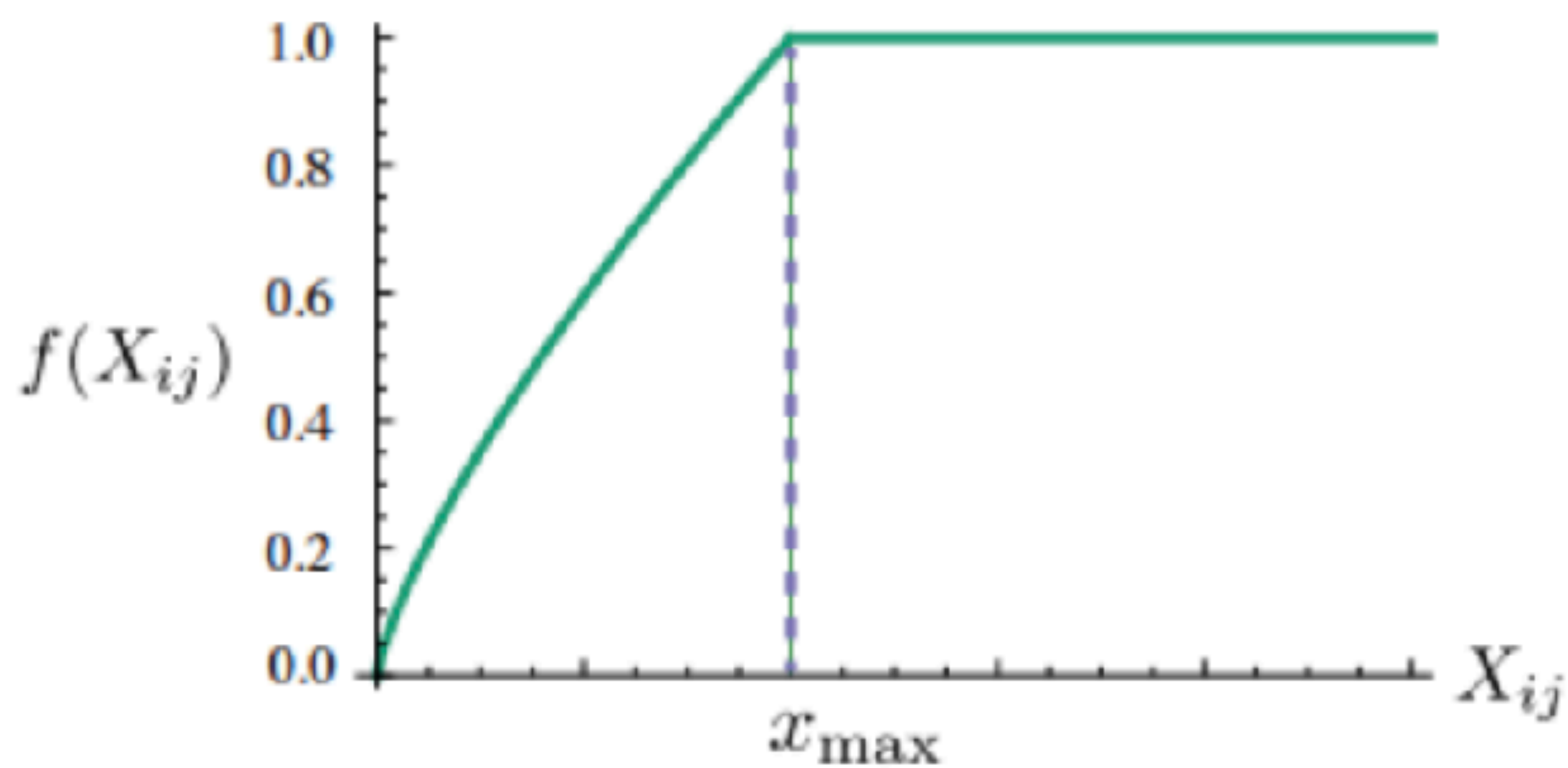
1. $P_{ice,solid}/P_{steam,solid}$ will be very high
2. $P_{ice,gas}/P_{steam,gas}$ is very low
3. $P_{ice,water}/P_{steam,water}$ will be higher than $P_{ice,fashion}/P_{steam,fashion}$



GloVe: High-level Architecture

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

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



Word Analogy

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**

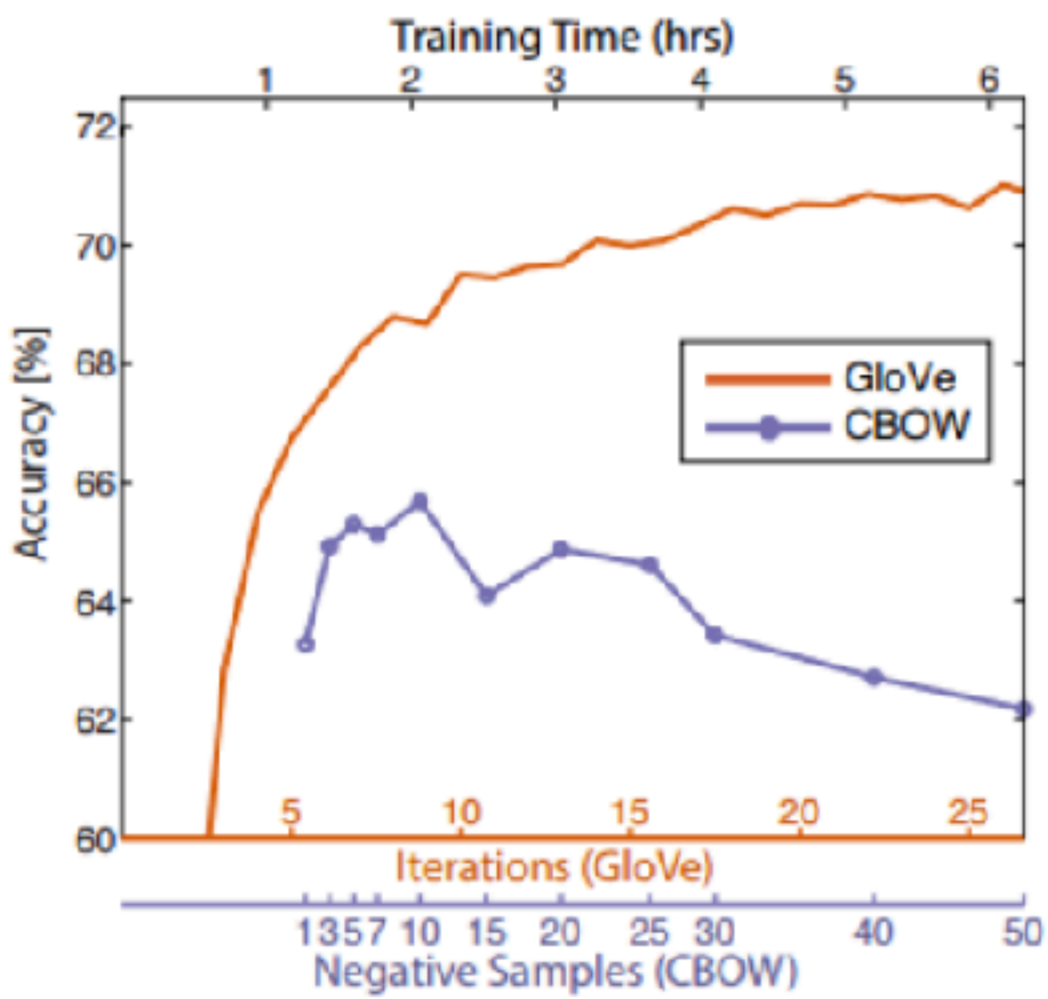
Word Similarity

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	<u>72.7</u>	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	<u>58.1</u>	37.2
GloVe	6B	<u>65.8</u>	<u>72.7</u>	<u>77.8</u>	53.9	<u>38.1</u>
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	<u>**75.9**</u>	<u>**83.6**</u>	<u>**82.9**</u>	<u>**59.6**</u>	<u>**47.8**</u>
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

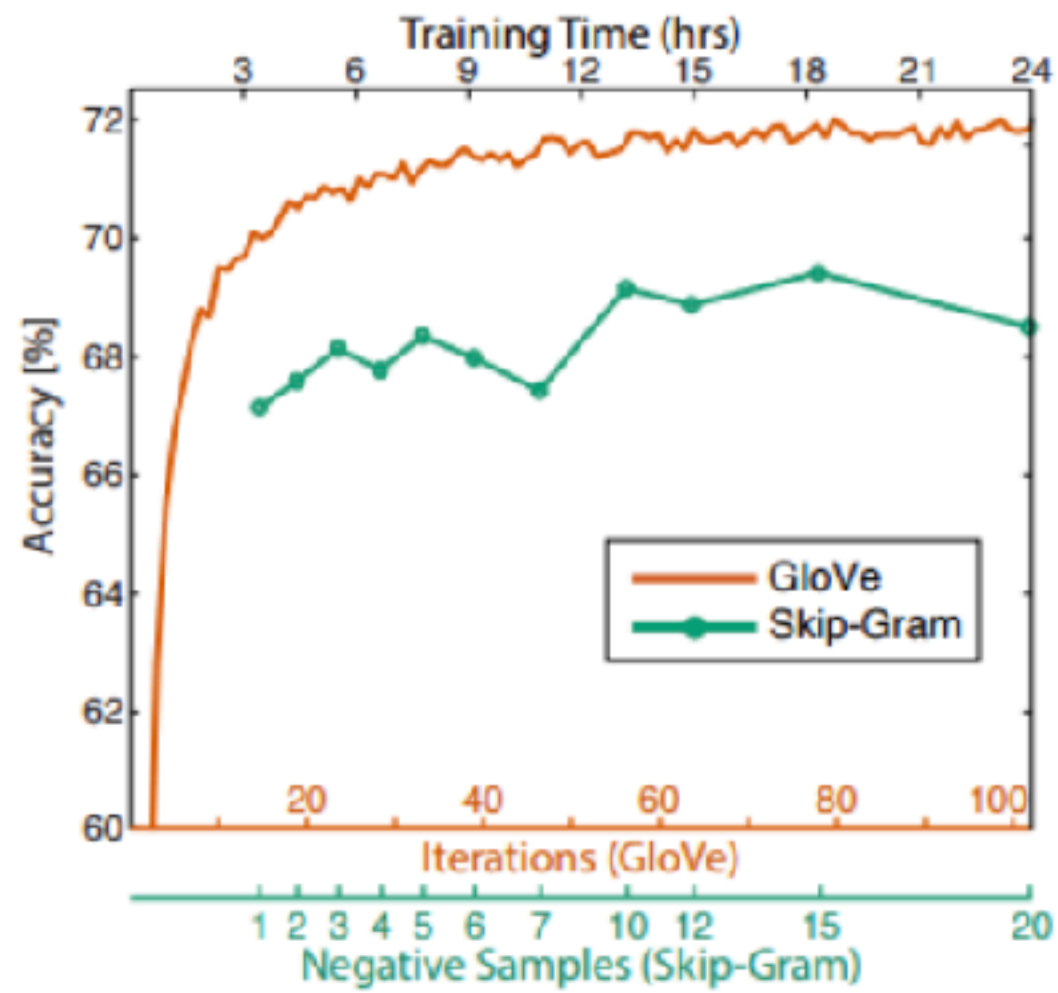
NER

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

GloVe Vs Word2Vec



(a) GloVe vs CBOW



(b) GloVe vs Skip-Gram