

Natural Language Processing (AI615)

Prepared by Dr. Ashraf Elsayed

Lecture 9



Word2vec

Lecture 9

Prepared by
Dr. Ashraf Elsayed

The objectives of this lecture is to:

- Understand Sparse versus dense vectors
- Define and understand Word2vec, GloVe, Fast Text
- Define Skip-Gram Classifier

Sparse versus dense vectors

- tf-idf (or PMI) vectors are
 - **long** (length $|V| = 20,000$ to $50,000$)
 - **sparse** (most elements are zero)
- Alternative: learn vectors which are
 - **short** (length 50-1000)
 - **dense** (most elements are non-zero)

Sparse versus dense vectors

- Why dense vectors?
 - Short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
 - Dense vectors may **generalize** better than explicit counts
 - Dense vectors may do better at capturing synonymy:
 - *car* and *automobile* are synonyms; but are distinct dimensions in sparse vectors
 - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
 - **In practice, they work better**

Common methods for getting short dense vectors

- “Neural Language Model”-inspired models
 - Word2vec (skipgram, CBOW), GloVe
 - Considered static embeddings (one fixed embedding for each word in the vocabulary)
- Alternative to these "static embeddings":
 - Contextual Embeddings (ELMo, BERT)
 - Compute distinct embeddings for a word in its context
 - Separate embeddings for each token of a word

Simple static embeddings you can download!

- Word2vec (Mikolov et al)
- <https://code.google.com/archive/p/word2vec/>
- GloVe (Pennington, Socher, Manning)
- <http://nlp.stanford.edu/projects/glove/>

Word2vec

- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: **predict** rather than **count**
- Word2vec provides various options. We'll cover:
 - **skip-gram with negative sampling (SGNS)**

Word2vec

- Instead of **counting** how often each word w occurs near "*apricot*"
 - Train a classifier on a binary **prediction** task:
 - Is w likely to show up near "*apricot*"?
- We don't actually care about this task
 - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
 - A word c that occurs near ***apricot*** in the corpus acts as the gold "correct answer" for supervised learning
 - No need for human labels
 - Neural Language Model (Bengio et al. (2003); Collobert et al. (2011))

Approach: predict if candidate word c is a "neighbor"

1. Treat the target word t and a neighboring context word c as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Skip-Gram Training Data

- Assume a +/- 2 word window, given training sentence:

• ...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

Skip-Gram Classifier

- (assuming a +/- 2 word window)

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

- Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair
 (apricot, jam)
 (apricot, aardvark)

...

- And assigns each pair a probability:

$$P(+ | w, c)$$

$$P(- | w, c) = 1 - P(+ | w, c)$$

Similarity is computed from dot product

- Remember: two vectors are similar if they have a high dot product
- So:
 - $\text{Similarity}(w,c) \propto w \cdot c$
- We'll need to normalize to get a probability
 - Use Sigmoid function

Turning dot products into probabilities

- $\text{Sim}(w, c) \approx w \cdot c$
- To turn this into a probability
- We'll use the sigmoid from logistic regression:

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

$$\begin{aligned} P(-|w, c) &= 1 - P(+|w, c) \\ &= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)} \end{aligned}$$

How Skip-Gram Classifier computes $P(+|w, c)$

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

- This is for one context word, but we have lots of context words.
- We'll assume independence and just multiply them:

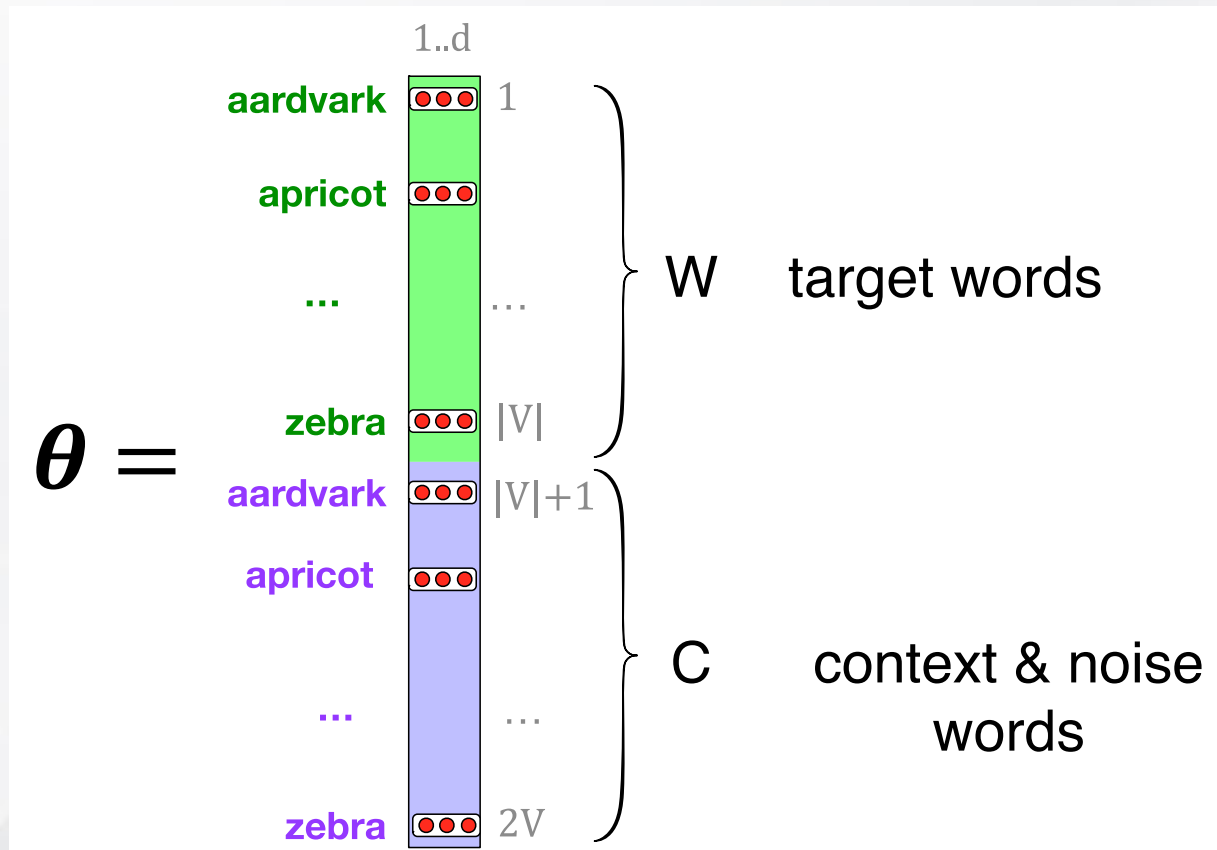
$$P(+|w, c_{1:L}) = \prod_{i=1}^L \sigma(c_i \cdot w)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^L \log \sigma(c_i \cdot w)$$

Skip-gram classifier: summary

- A probabilistic classifier, given
 - a test target word w
 - its context window of L words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).
- To compute this, we just need embeddings for all the words.

These embeddings we'll need: a set for w , a set for c



Vector Semantics & Embeddings

Word2vec: Learning the embeddings

Skip-Gram Training data

- Skip-gram learns embeddings by starting with random embedding vectors.
- Then iteratively shifting the embedding of each word w to be more like the embeddings of words that occur nearby in texts, and less like the embeddings of words that don't occur nearby.
- Let's start by considering a single piece of training data

This example has a target word w (apricot), and 4 context words in the $L = \pm 2$ window, resulting in 4 positive training instances:

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1

c2

[target]

c3

c4

positive examples +

t

c

apricot tablespoon

apricot of

apricot jam

apricot a

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1

c2 [target]

c3

c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

For each positive example we'll grab k negative examples, sampling by frequency

Parameter

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1

c2 [target]

c3

c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Word2vec: how to learn vectors

- Given the set of positive and negative training instances, and an initial set of embedding vectors
- The goal of learning is to adjust those word vectors such that we:
 - **Maximize** the similarity of the **target word**, **context word** pairs (w, c_{pos}) drawn from the positive data
 - **Minimize** the similarity of the (w, c_{neg}) pairs drawn from the negative data.

Loss function for one w with c_{pos} , $c_{neg1} \dots c_{negk}$

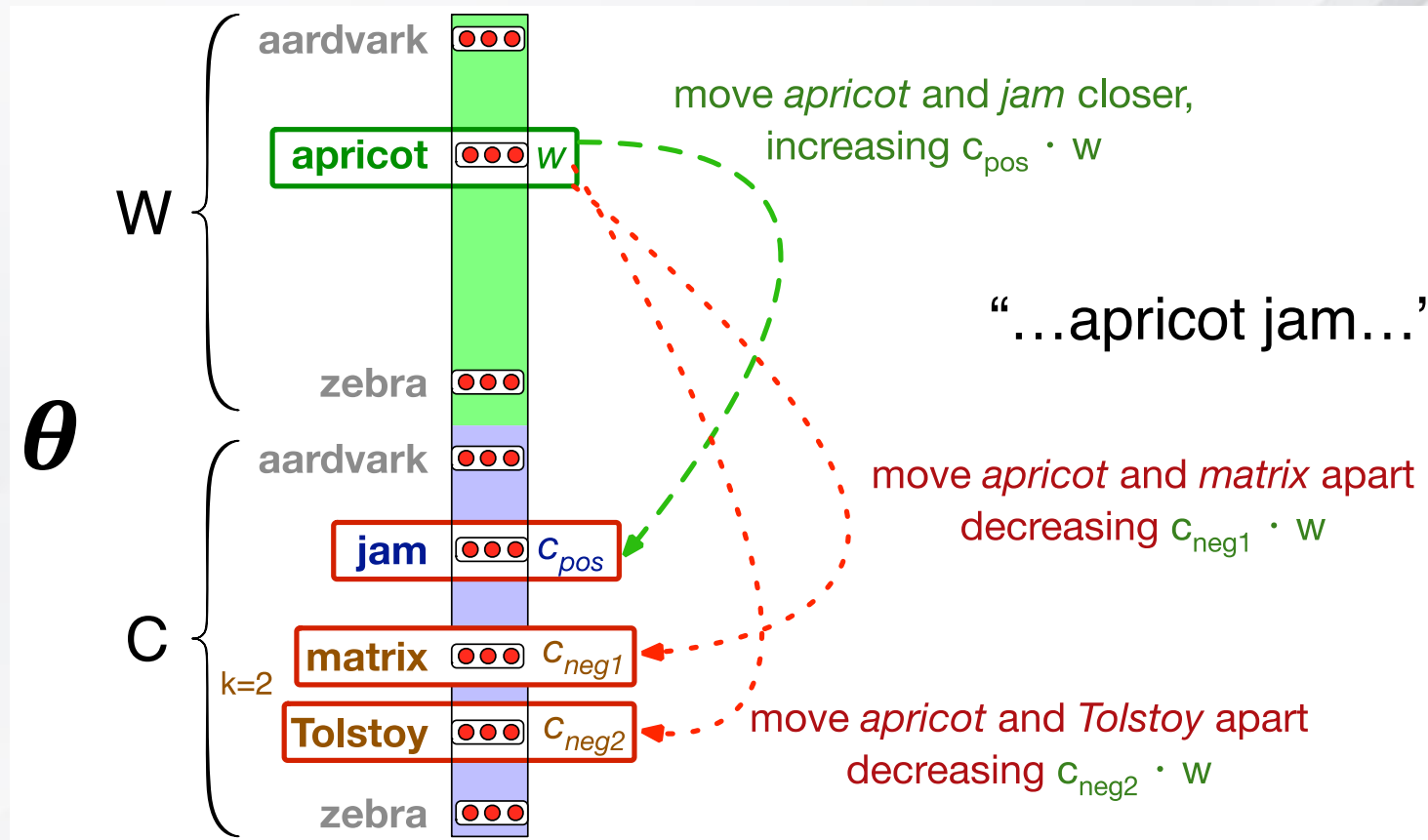
- Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the k negative sampled non-neighbor words.

$$\begin{aligned}
 L_{CE} &= -\log \left[P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \\
 &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\
 &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log (1 - P(+|w, c_{neg_i})) \right] \\
 &= - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right]
 \end{aligned}$$

Learning the classifier

- How to learn?
 - Stochastic gradient descent!
- We'll adjust the word weights to
 - make the positive pairs more likely
 - and the negative pairs less likely,
 - over the entire training set.

Intuition of one step of gradient descent



Reminder: gradient descent

- At each step
 - Direction: We move in the reverse direction from the gradient of the loss function
 - Magnitude: we move the value of this gradient $\frac{d}{dw} L(f(x; w), y)$ weighted by a **learning rate** η
 - Higher learning rate means move w faster

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

The derivatives of the loss function

$$L_{CE} = - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Update equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$c_{pos}^{t+1} = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w^t) - 1] w^t$$

$$c_{neg}^{t+1} = c_{neg}^t - \eta [\sigma(c_{neg}^t \cdot w^t)] w^t$$

$$w^{t+1} = w^t - \eta \left[[\sigma(c_{pos} \cdot w^t) - 1] c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w^t)] c_{neg_i} \right]$$

Two sets of embeddings

- SGNS learns two sets of embeddings
 - Target embeddings matrix W
 - Context embedding matrix C
- It's common to just add them together, representing word i as the vector $w_i + c_i$

Summary: How to learn word2vec (skip-gram) embeddings

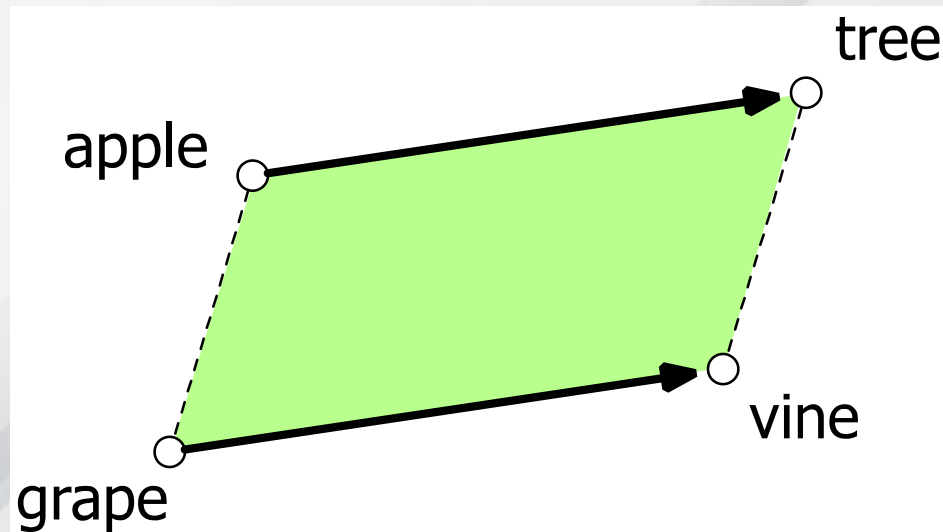
- Start with V random d -dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
 - Take a corpus and take pairs of words that co-occur as positive examples
 - Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

The kinds of neighbors depend on window size

- **Small windows** ($C = \pm 2$) : nearest words are syntactically similar words in same taxonomy
 - *Hogwarts* nearest neighbors are other fictional schools
 - *Sunnydale, Evernight, Blandings*
- **Large windows** ($C = \pm 5$) : nearest words are related words in same semantic field
 - *Hogwarts* nearest neighbors are Harry Potter world:
 - *Dumbledore, half-blood, Malfoy*

Analogical relations

- The classic parallelogram model of analogical reasoning
- To solve: "*apple is to tree as grape is to _____*"
- Add $\overrightarrow{\text{tree} - \text{apple}}$ to $\overrightarrow{\text{grape}}$ to get *vine*



Analogical relations via parallelogram

- The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)
- $\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$
- $\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}}$ is close to $\overrightarrow{\text{Rome}}$
- For a problem $a:a^*:b:b^*$, the parallelogram method is:

$$\hat{\mathbf{b}}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \operatorname{distance}(\mathbf{x}, \mathbf{b} - \mathbf{a} + \mathbf{a}^*)$$

Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

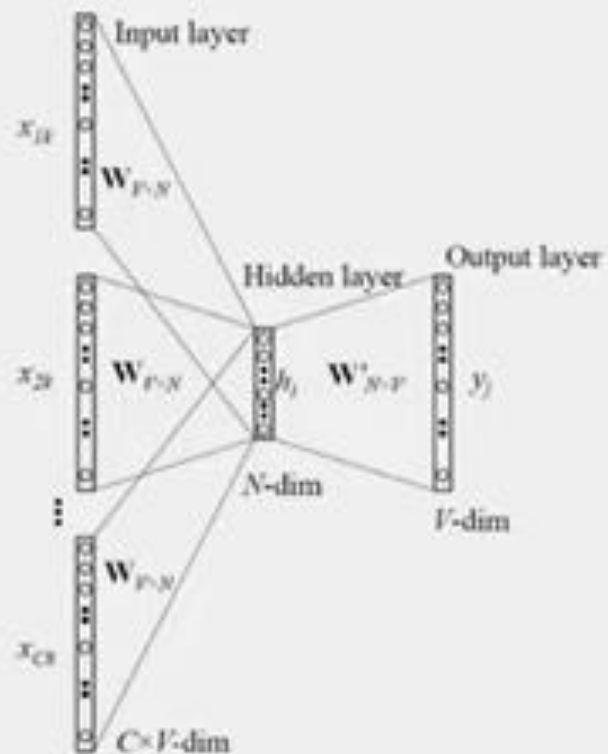
- Ask “Paris : France :: Tokyo : x”
 - x = Japan
- Ask “father : doctor :: mother : x”
 - x = nurse
- Ask “man : computer programmer :: woman : x”
 - x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Word2Vec

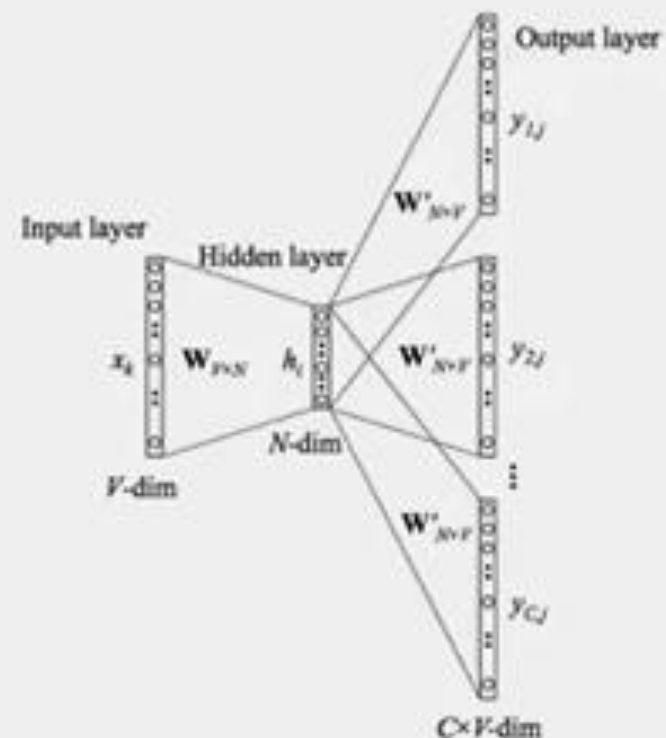
- Move a window over text to get C context words (V -dim one-hot encoded)
- Add hidden layer with N linear nodes, average pooling, and softmax layer(s)
- CBOW: predict word given context, use weights of last layer as embedding
- Skip-Gram: predict context given word, use weights of first layer as embedding (better for large corpora)

I like playing **football** with my friends



CBOW

predict word given context

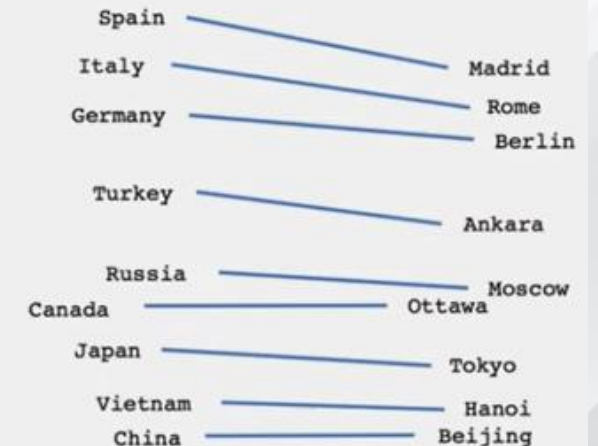
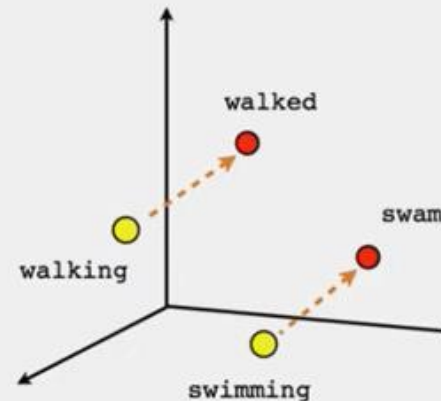
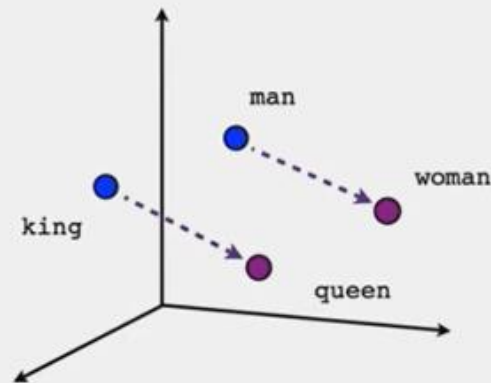


Skip-gram

predict context given word

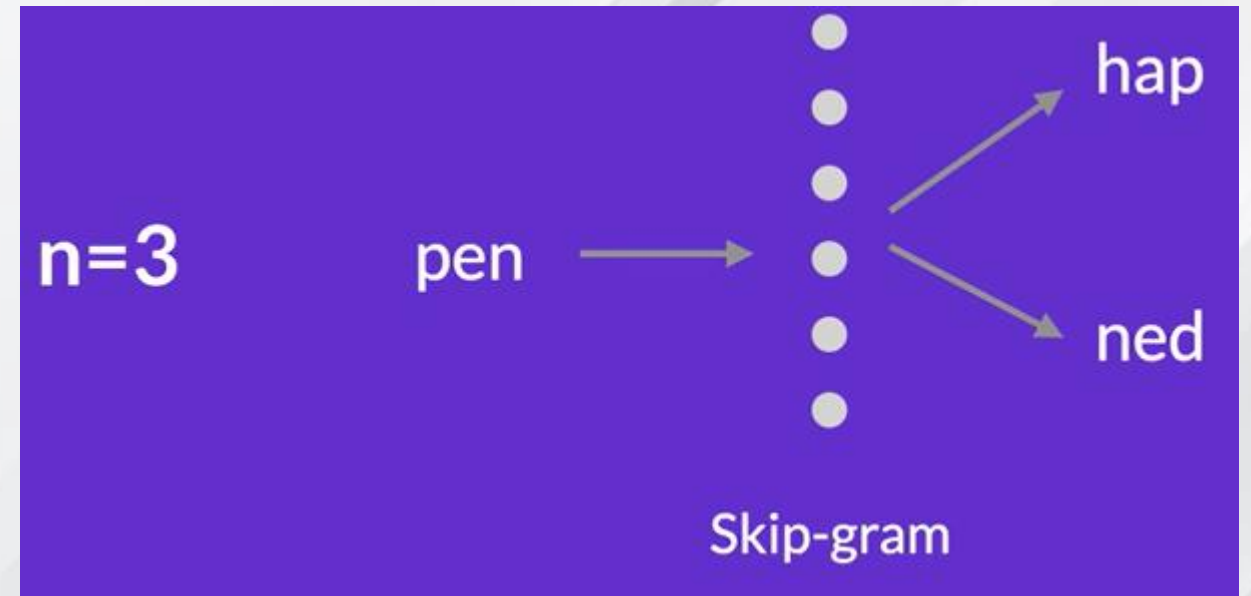
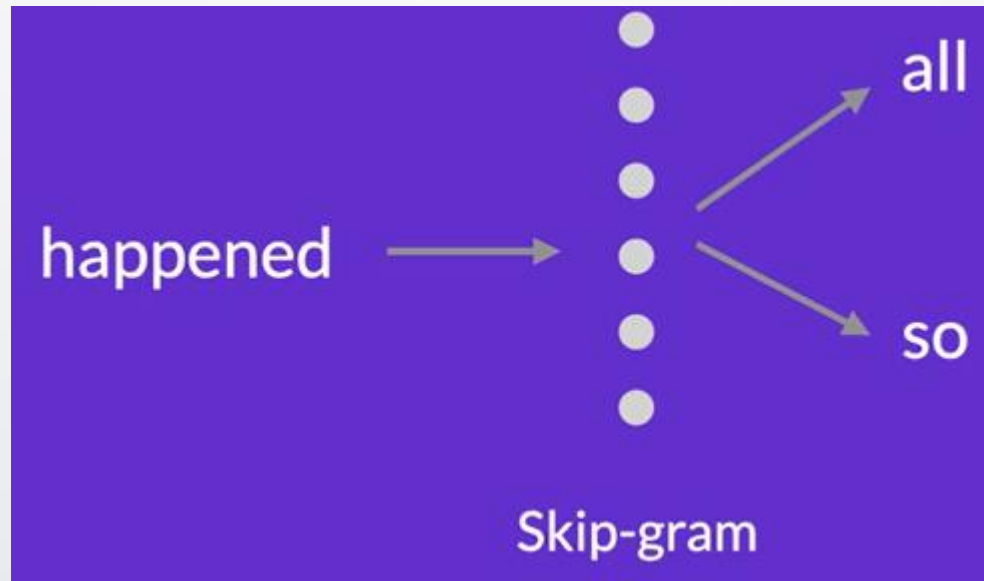
Word2Vec properties

- Word2Vec happens to learn interesting relationships between words
 - Simple vector arithmetic can map words to plurals, conjugations, gender analogies,...
 - e.g. Gender relationships: $vec_{king} - vec_{man} + vec_{woman} \sim vec_{queen}$
 - PCA applied to embeddings shows Country - Capital relationship
- Careful: embeddings can capture gender and other biases present in the data.
 - Important unsolved problem!



FastText

- Limitations of Word2Vec:
 - Cannot represent new (out-of-vocabulary) words
 - Words like 'meet' and 'meeting' are learned independently: less efficient (no parameter sharing)
- FastText addresses this by using character n-grams
 - Basic model can be Skip-gram or CBOW
 - Words are represented by all character n-grams of length 3 to 6
 - "football" 3-grams: <fo, foo, oot, otb, tba, bal, all, ll>
 - Because there are so many n-grams, they are hashed
 - Dimensionality V = bin size
 - Representation of word "football" is sum of its n-gram embeddings
- Training with positive examples (in-context words) and random negative examples



Global Vector model (GloVe)

- Builds a co-occurrence matrix $\mathbf{X}$
 - Counts how often 2 words occur in the same context (and how close)
- Learns a k-dimensional embedding w through matrix factorization with rank k
 - Actually learns 2 embeddings w and w' (differ in random initialization)
- Minimizes loss $\mathcal{L}$, where b_i and b'_i are bias terms and f is a weighting function

$$\mathcal{L} = \sum_{i,j=1}^V f(\mathbf{X}_{ij})(\mathbf{w}_i \mathbf{w}'_j + b_i + b'_j - \log(\mathbf{X}_{ij}))^2$$

I like playing football with my friends
This is the tallest tree I have ever seen
I have trained my dog well

	my	tallest	trained
football	1		
tree		1	
dog	1		1

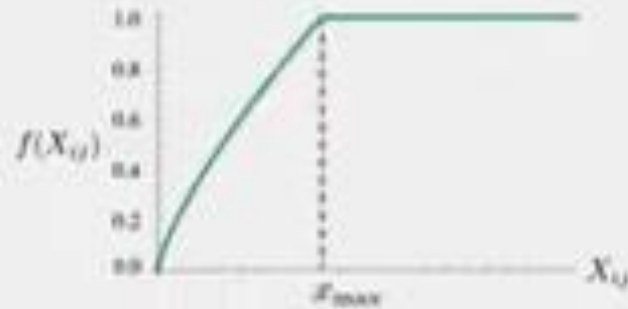
Co-occurrence matrix X

$$\log(X)$$

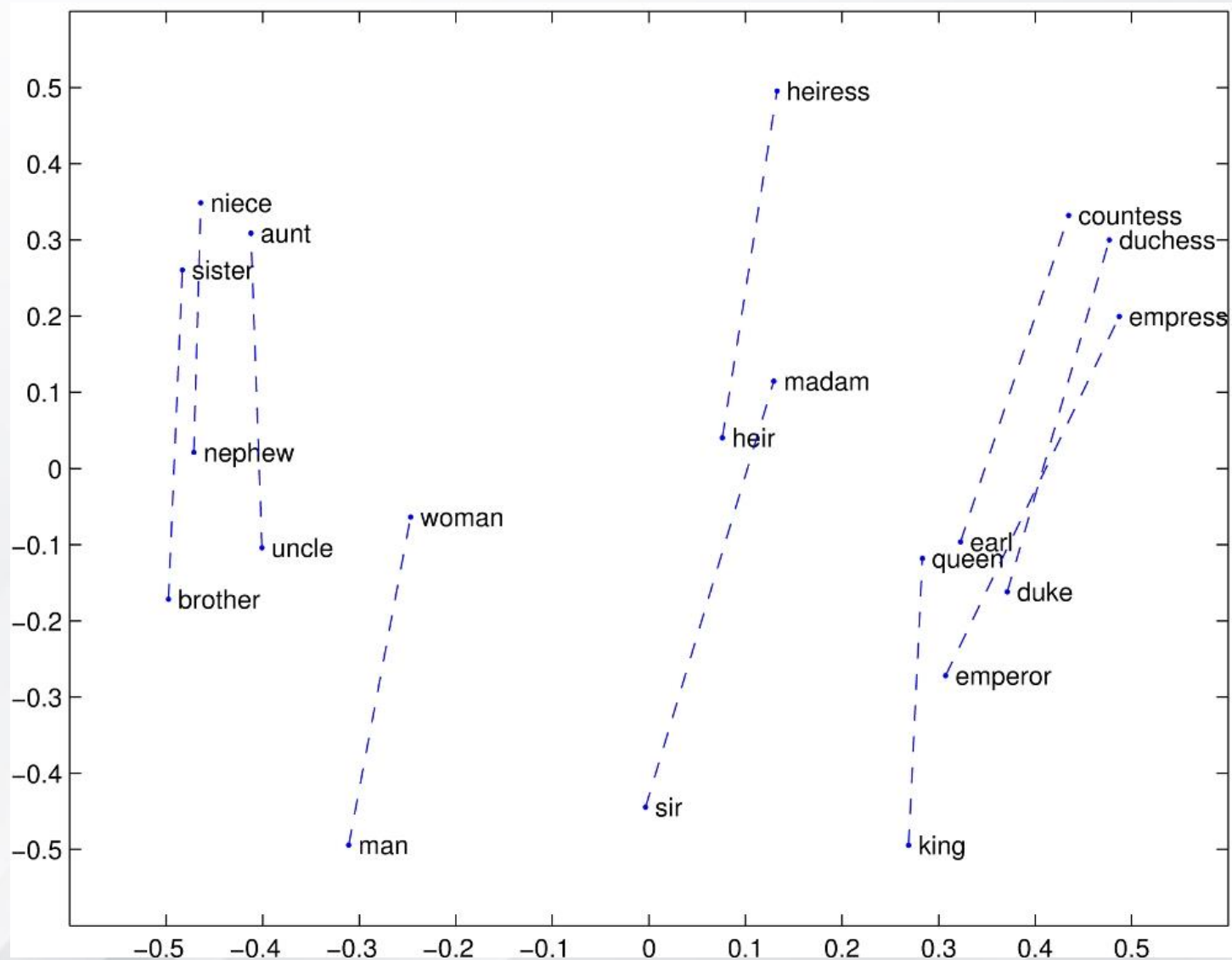
$\equiv$

$$\begin{matrix} K \\ \boxed{w} \end{matrix} \quad \begin{matrix} \boxed{w'} \\ K \end{matrix} + \begin{matrix} \boxed{b} \end{matrix} + \begin{matrix} \boxed{b'} \end{matrix}$$

word vectors (learned) bias terms (learned)

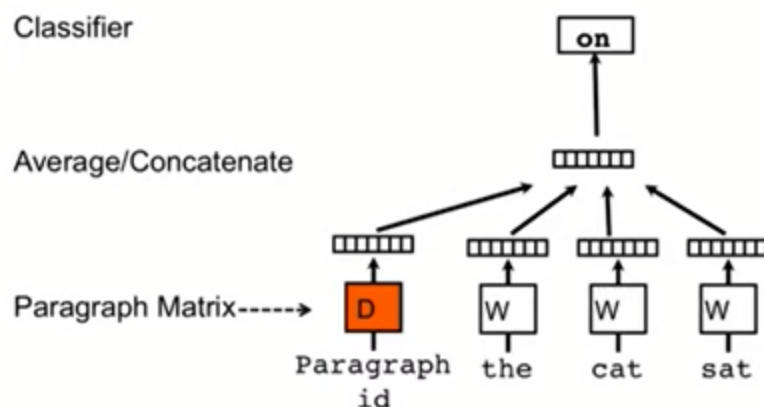


Structure in GloVe Embedding space



Document/paragraph embeddings

- Simplest approach to represent a document: sum or average of all word vectors
- Doc2Vec
 - Next to word embeddings, also learn an embedding for the document/paragraph
 - Acts as a memory that remembers what is missing from the current context (the topic of the paragraph)
 - Can be used to determine semantic similarity between documents.
 - Can be tricky to train.



Example

Skip-grams in Word2Vec

I want a glass of orange juice to go along with my cereal.

Context word

orange

orange

orange

Target/Neighbor

juice

glass

my

Model

Vocab size = 10,000

Context c (“orange”) $\longrightarrow$ Target t (“juice”)

$$o_c \longrightarrow E \longrightarrow e_c \xrightarrow{\text{softmax}} o \longrightarrow \hat{y}$$

Model

Vocab size = 10,000

Context c (“orange”) → Target t (“juice”)

$$o_c \longrightarrow E \longrightarrow e_c \xrightarrow{\text{softmax}} o \longrightarrow \hat{y}$$

$$\text{softmax: } p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

θ_t = parameters associated with context t

$$\mathcal{L}(\hat{y}, y) = - \sum_{i=1}^{10,000} y_i \log \hat{y}_i$$

Word2Vec Data Generation

“king brave man”

“queen beautiful woman”

window size = 1

Word	neighbor
king	brave
brave	king
brave	man
man	brave
queen	beautiful
beautiful	queen
beautiful	woman
woman	beautiful

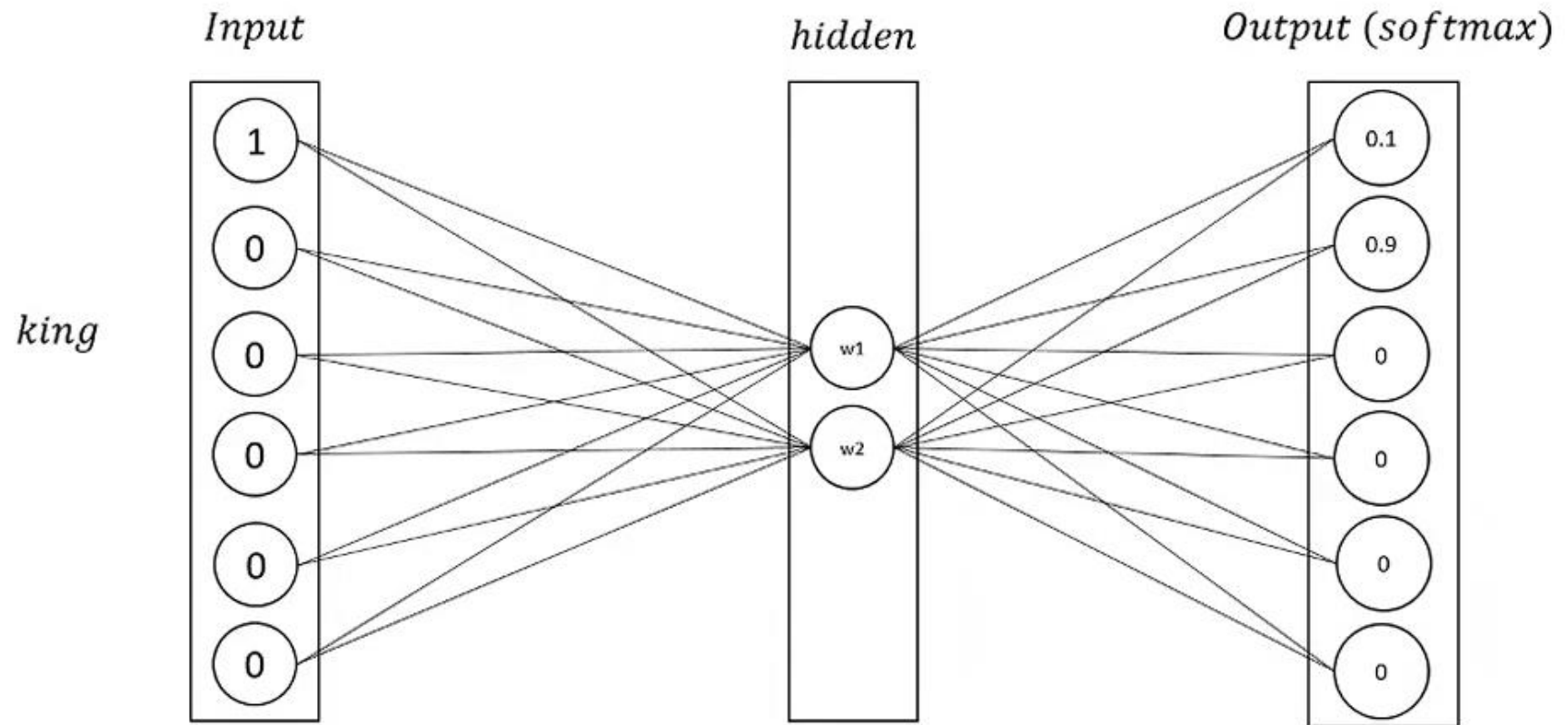
window size = 2

Word	neighbor
king	brave
king	man
brave	king
brave	man
man	king
man	brave
queen	beautiful
queen	woman
beautiful	queen
beautiful	woman
woman	queen
woman	beautiful

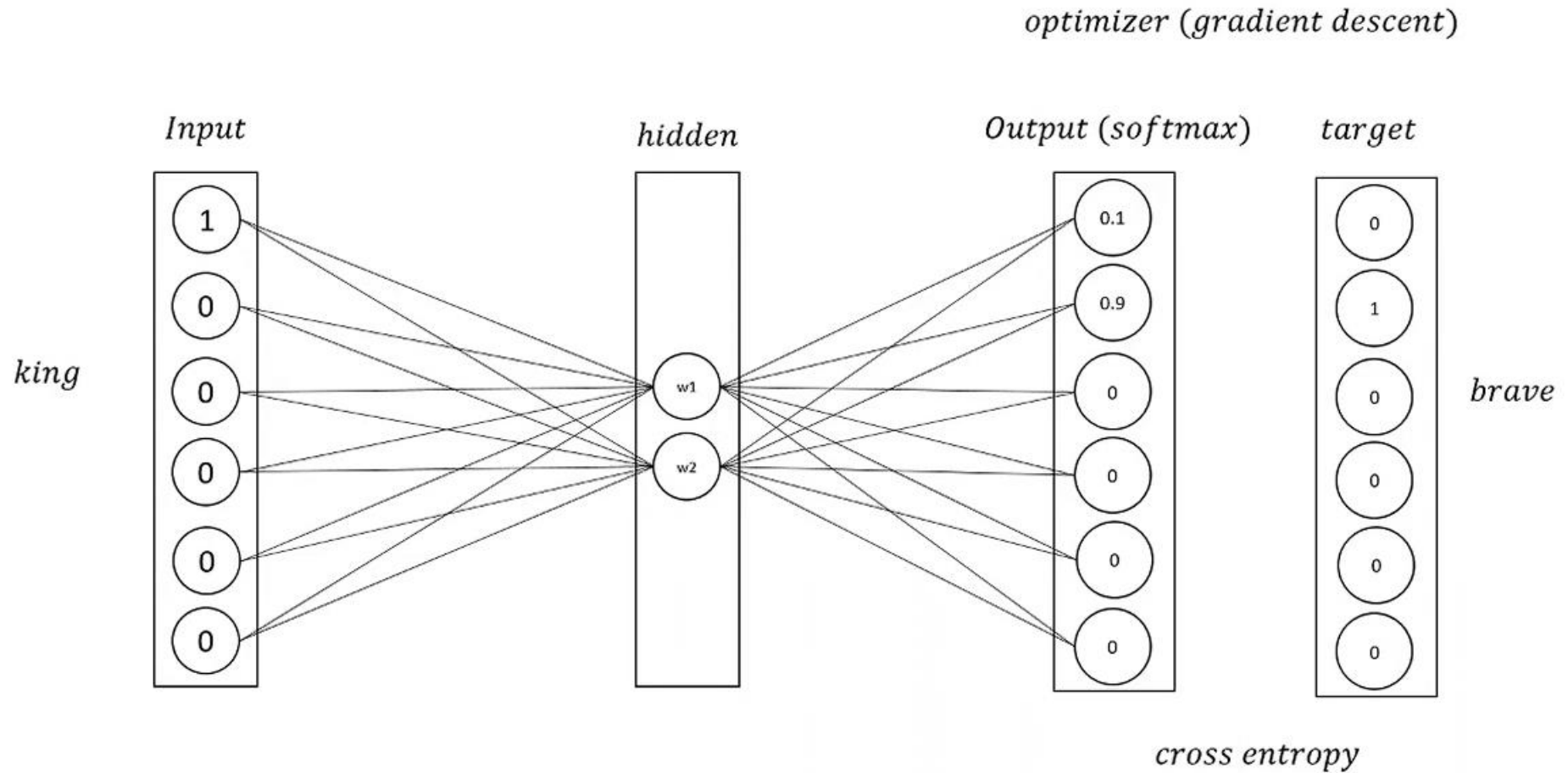
Word2Vec input and target

Word one-hot encoding	Neighbor one-hot encoding
[1,0,0,0,0,0]	[0,1,0,0,0,0]
[1,0,0,0,0,0]	[0,0,1,0,0,0]
[0,1,0,0,0,0]	[1,0,0,0,0,0]
[0,1,0,0,0,0]	[0,0,1,0,0,0]
[0,0,1,0,0,0]	[1,0,0,0,0,0]
[0,0,1,0,0,0]	[0,1,0,0,0,0]
[0,0,0,1,0,0]	[0,0,0,0,1,0]
[0,0,0,1,0,0]	[0,0,0,0,0,1]
[0,0,0,0,1,0]	[0,0,0,1,0,0]
[0,0,0,0,1,0]	[0,0,0,0,0,1]
[0,0,0,0,0,1]	[0,0,0,1,0,0]
[0,0,0,0,0,1]	[0,0,0,0,1,0]

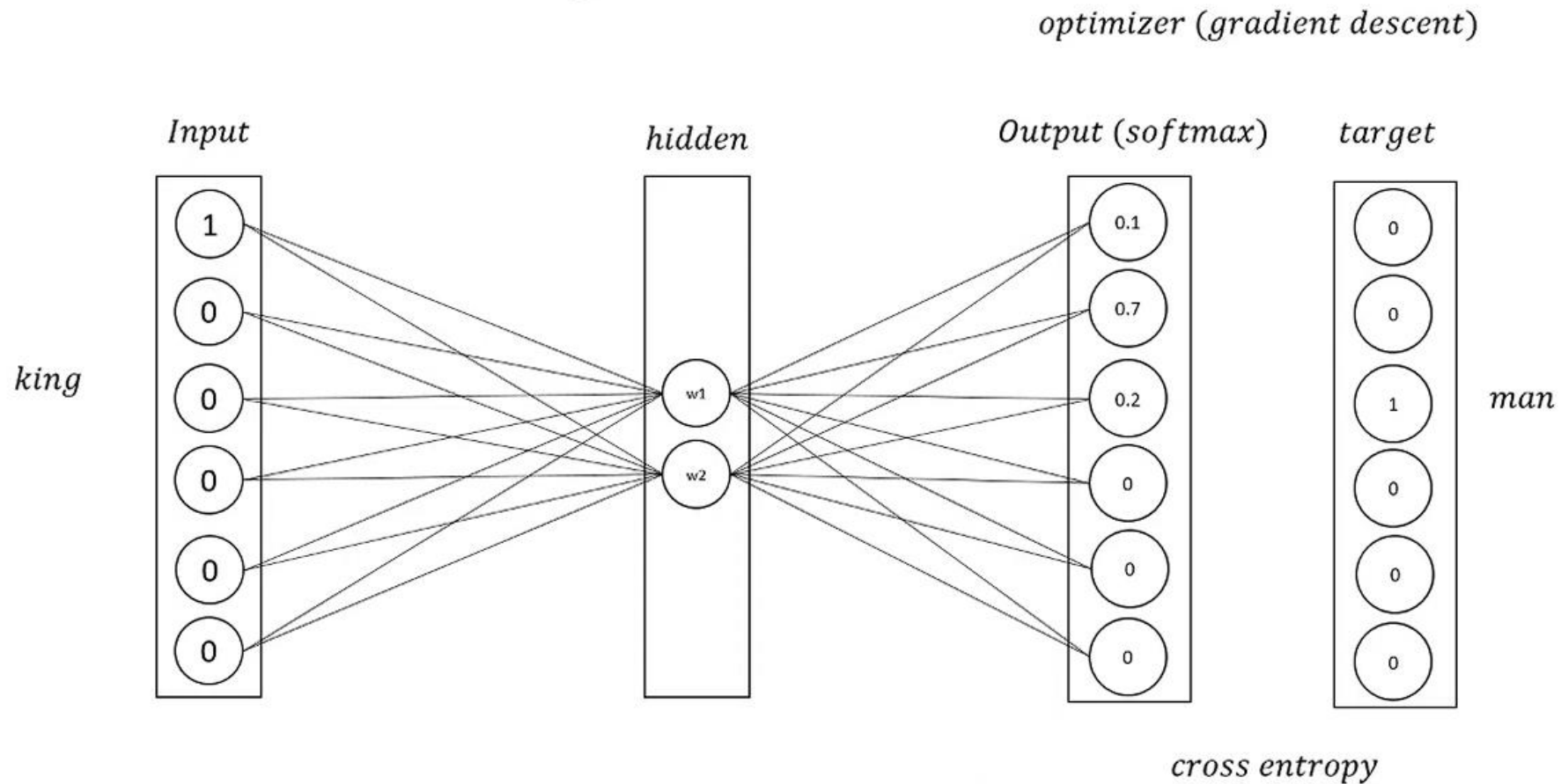
Word2Vec Training



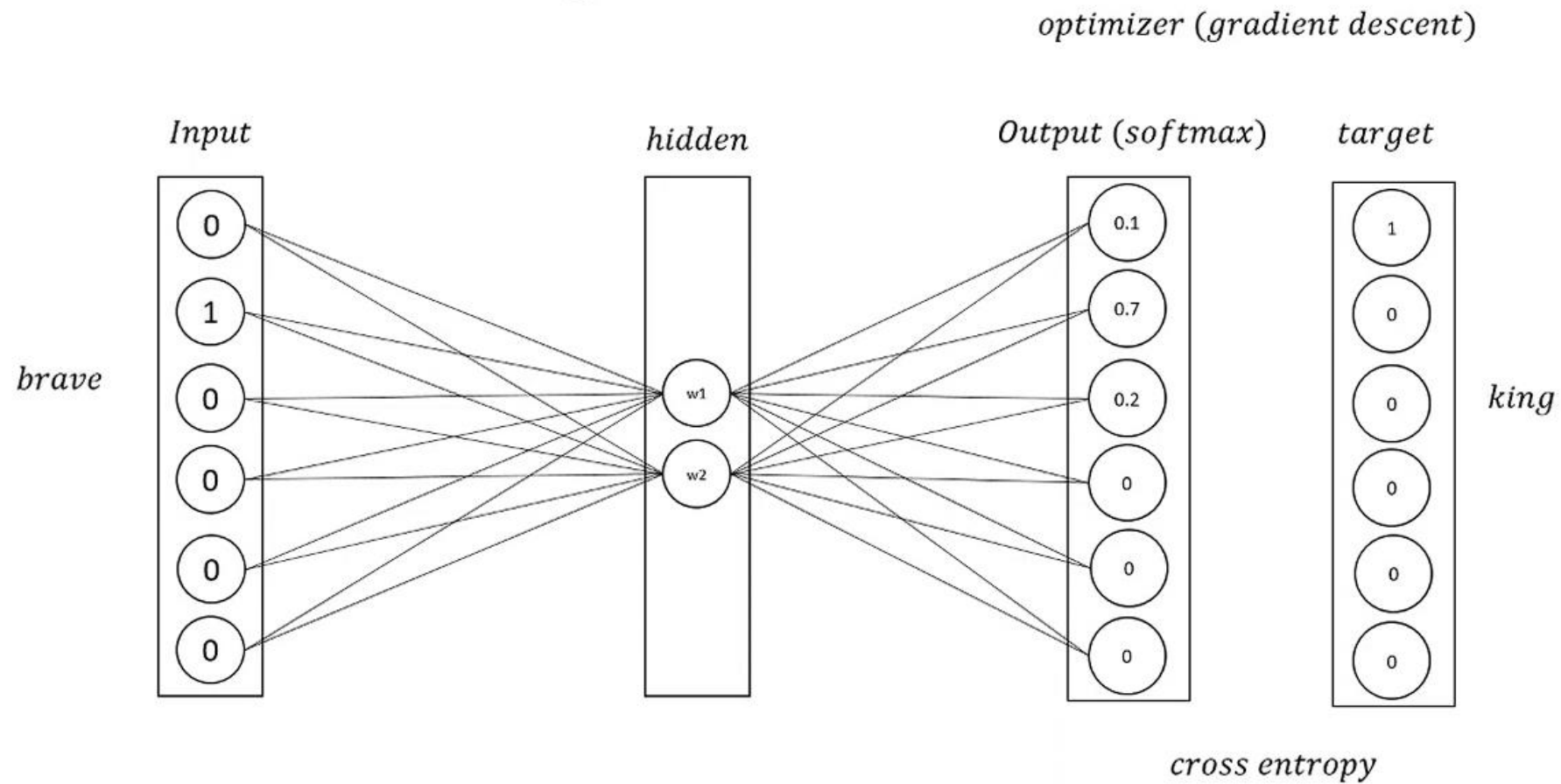
Word2Vec Training



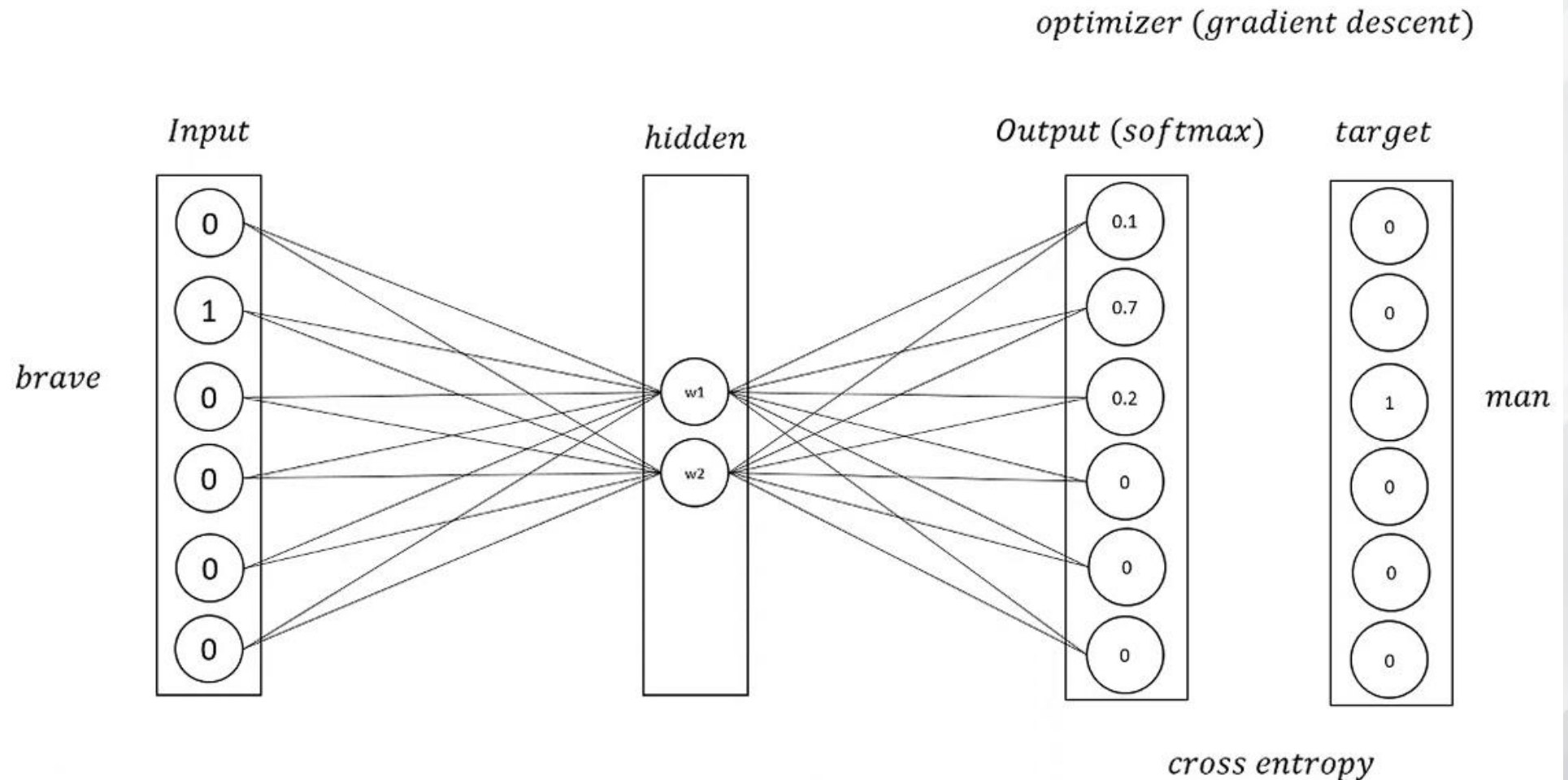
Word2Vec Training



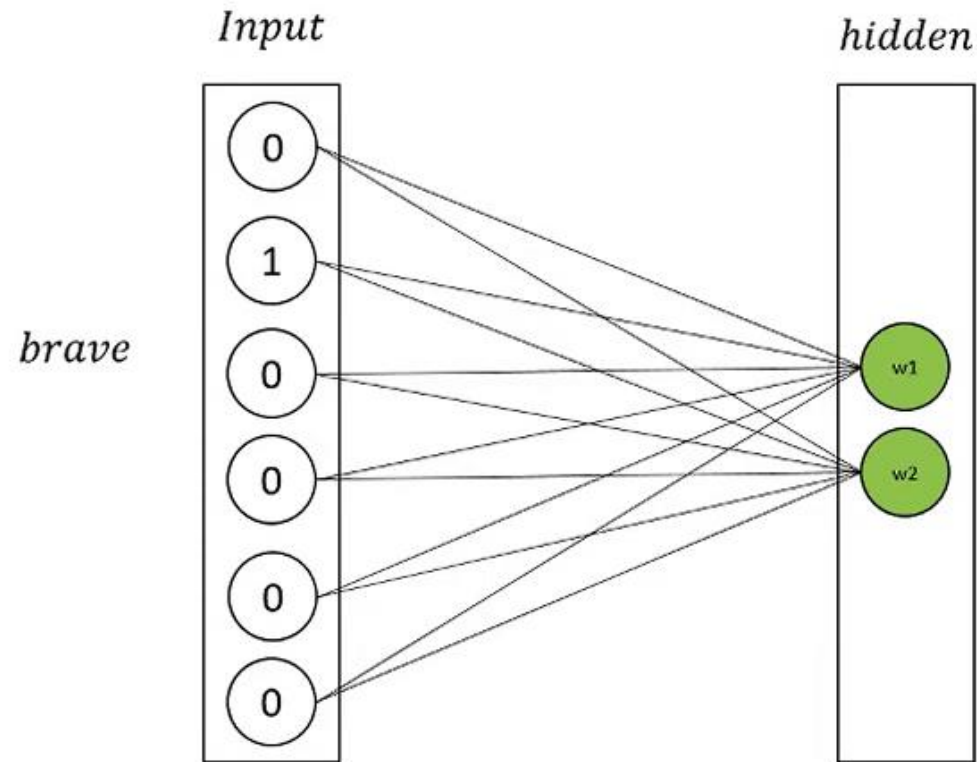
Word2Vec Training



Word2Vec Training

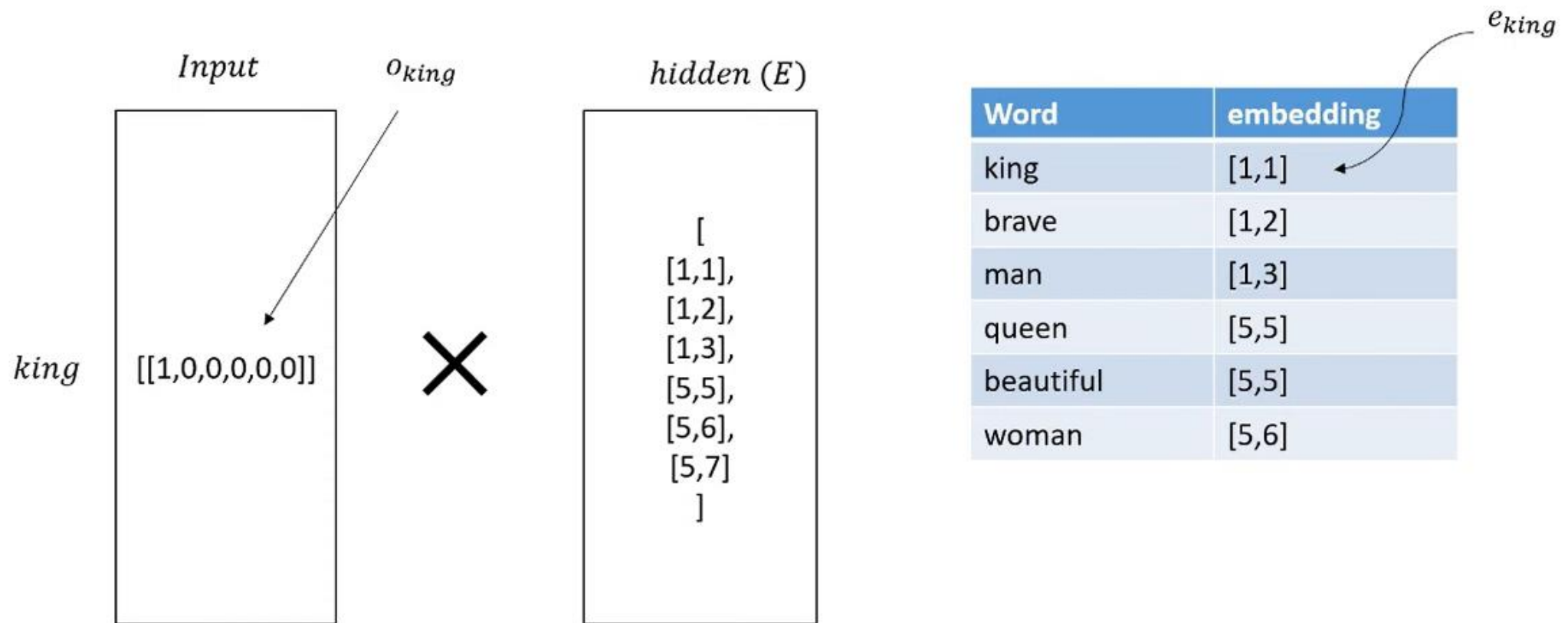


Word2Vec is hidden layer after training

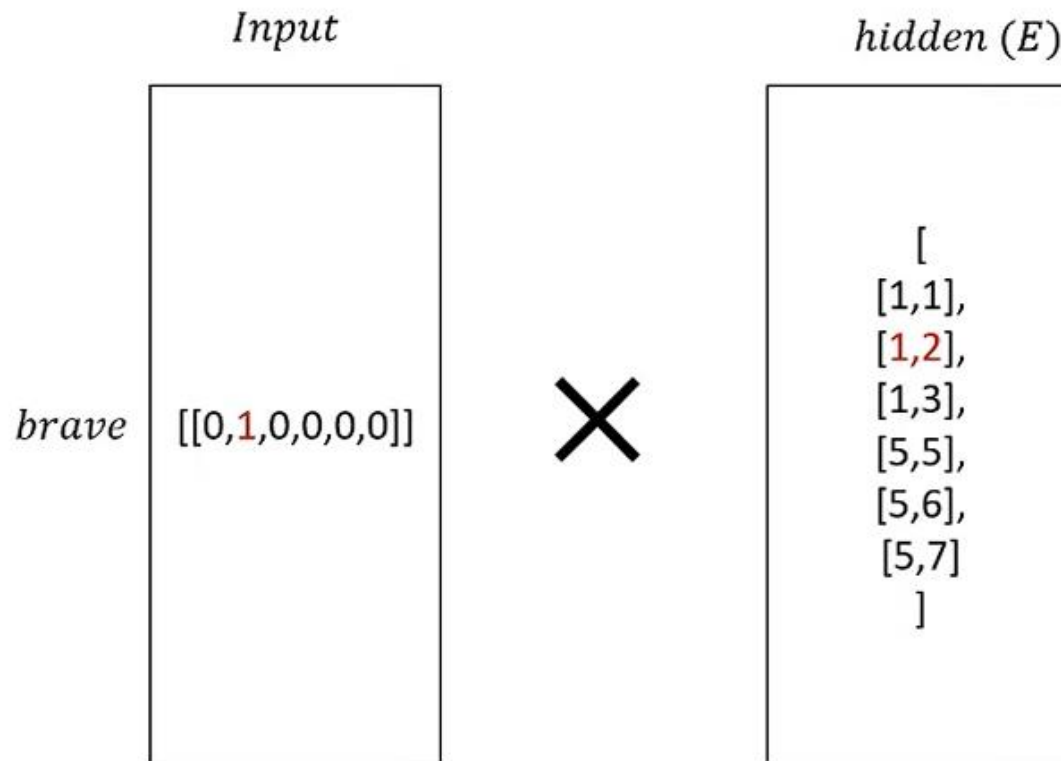


Word	embedding
king	[1,1]
brave	[1,2]
man	[1,3]
queen	[5,5]
beautiful	[5,5]
woman	[5,6]

Input one-hot encoding – hidden as lookup



Input one-hot encoding – hidden as lookup



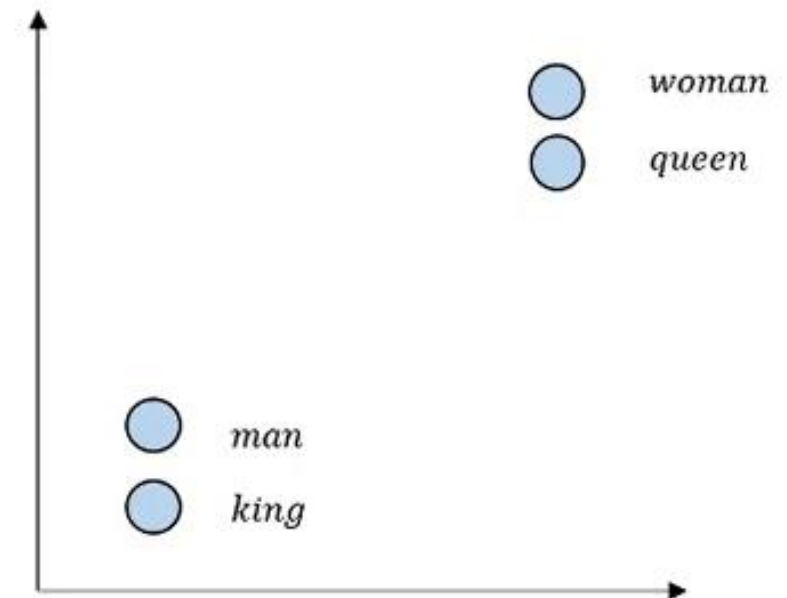
Word	embedding
king	[1,1]
brave	[1,2]
man	[1,3]
queen	[5,5]
beautiful	[5,5]
woman	[5,6]

Word2Vec gives similarity in vector representation

Word	encoding	Word2vec embedding
king	[1,0,0,0,0,0]	[1,1]
man	[0,0,1,0,0,0]	[1,3]
queen	[1,0,0,1,0,0]	[5,5]
woman	[1,0,0,0,0,1]	[5,7]

Word2Vec gives similarity in vector representation

Word	encoding	Word2vec embedding
king	[1,0,0,0,0,0]	[1,1]
man	[0,0,1,0,0,0]	[1,3]
queen	[1,0,0,1,0,0]	[5,5]
woman	[1,0,0,0,0,1]	[5,7]



Example

GloVe (global vectors for word representation)

I love NLP.

I love to make videos.

	I	love	NLP	to	make	videos	.
I	0	2	0	0	0	0	0
love	2	0	1	1	0	0	0
NLP	0	1	0	0	0	0	1
to	0	1	0	0	1	0	0
make	0	0	0	1	0	1	0
videos	0	0	0	0	1	0	1
.	0	0	1	0	0	1	0

GloVe (global vectors for word representation)

I love NLP.

I love to make videos.

X_{ij} = # times j appears in context of i

$X_{ij} = X_{ji}$

	I	love	NLP	to	make	videos	.
I	0	2	0	0	0	0	0
love	2	0	1	1	0	0	0
NLP	0	1	0	0	0	0	1
to	0	1	0	0	1	0	0
make	0	0	0	1	0	1	0
videos	0	0	0	0	1	0	1
.	0	0	1	0	0	1	0

$$\text{minimize } \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij}) (\theta_i^T e_j + b_i + b_j' - \log X_{ij})^2$$

Summary

- Understand Sparse versus dense vectors
- Define and understand Word2vec, GLoVe, Fast Text
- Define Skip-Gram Classifier



Thank You