Natural Language Processing with Deep Learning
Word Embeddings

Navid Rekab-Saz
navid.rekabsaz@jku.at

JOHANNES KEPLER
UNIVERSITY LINZ

Institute of
Computational
Perception

Agenda

- Introduction
- Count-based word representation
- Prediction-based word embedding

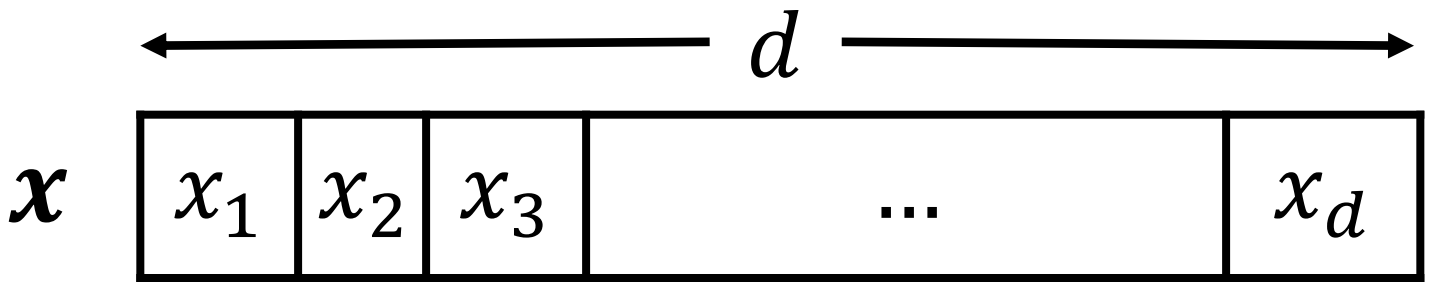
Agenda

- **Introduction**
- Count-based word representation
- Prediction-based word embedding

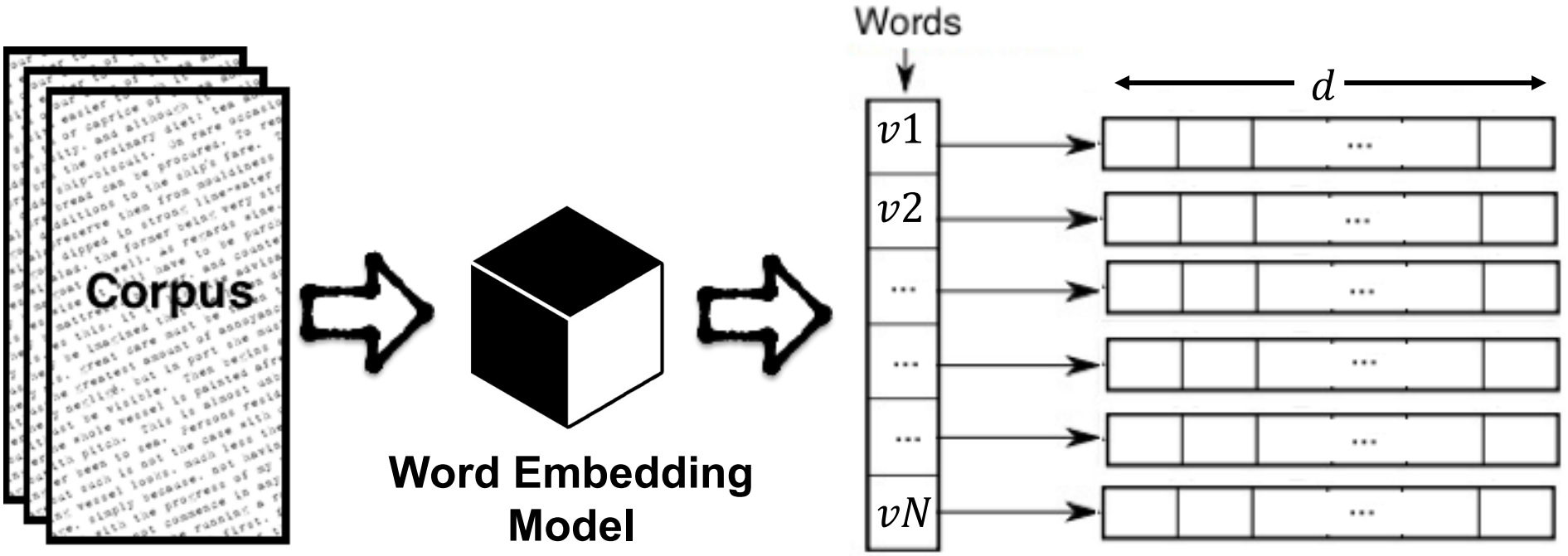
Distributional Representation

- An entity is represented with a vector of d dimensions

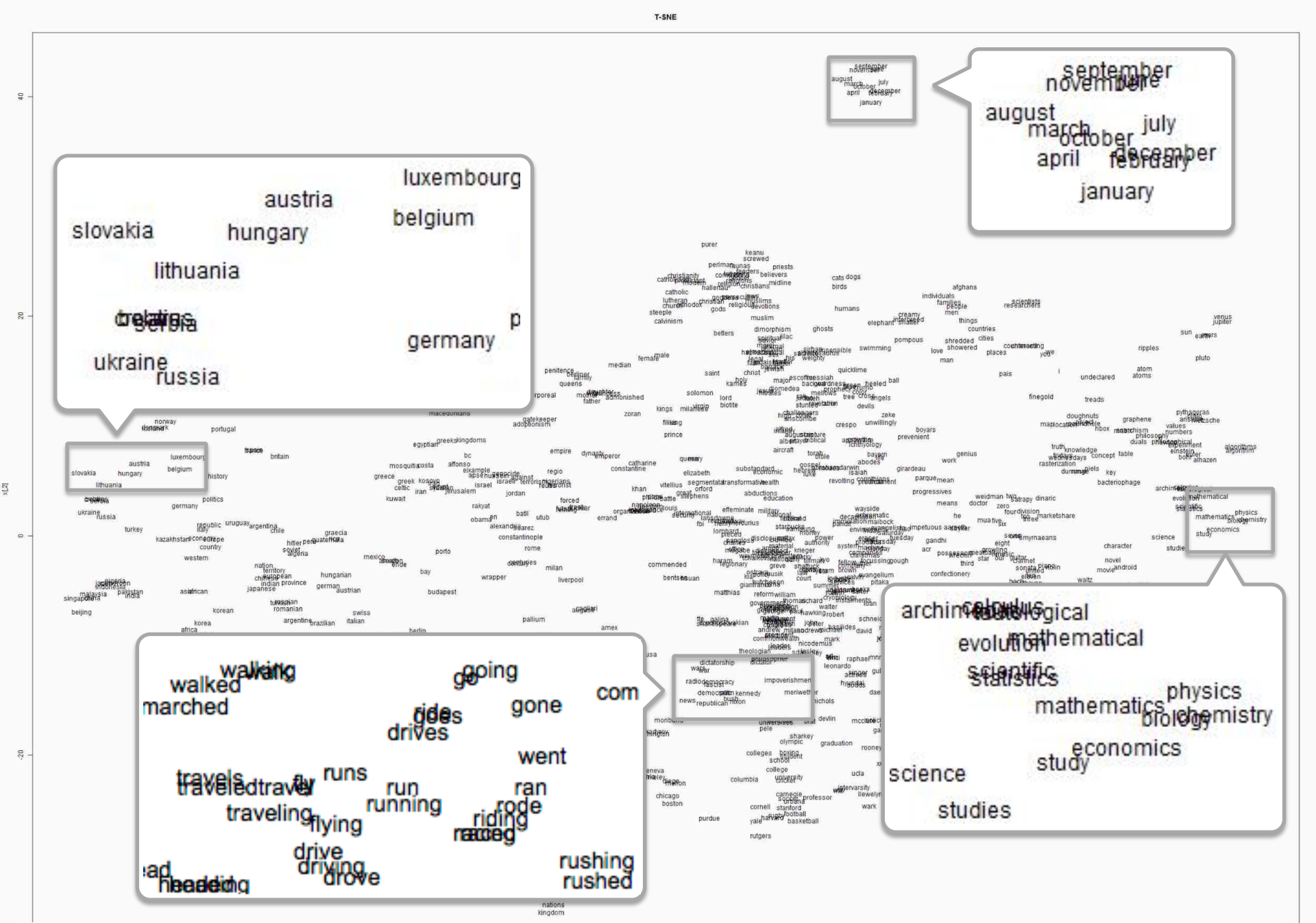
- Distributed Representations
  - Each dimension (units) is a feature of the entity
  - Units in a layer are not mutually exclusive
  - Two units can be ‘‘active’’ at the same time



Word Embedding Model



When vector representations are dense, they are often called embedding e.g. word embedding



Word embeddings projected to a two-dimensional space

Word Embeddings – Nearest neighbors

frog		book	asthma
frogs		books	bronchitis
toad		foreword	allergy
litoria		author	allergies
leptodactylidae		published	arthritis
rana		preface	diabetes

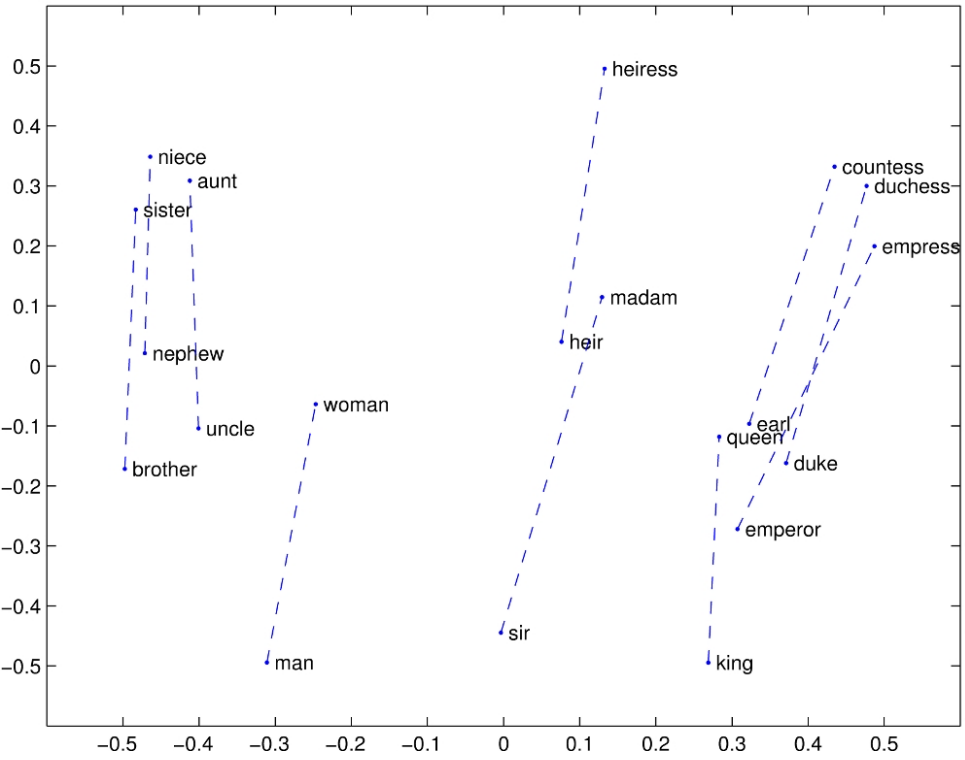
https://nlp.stanford.edu/projects/glove/

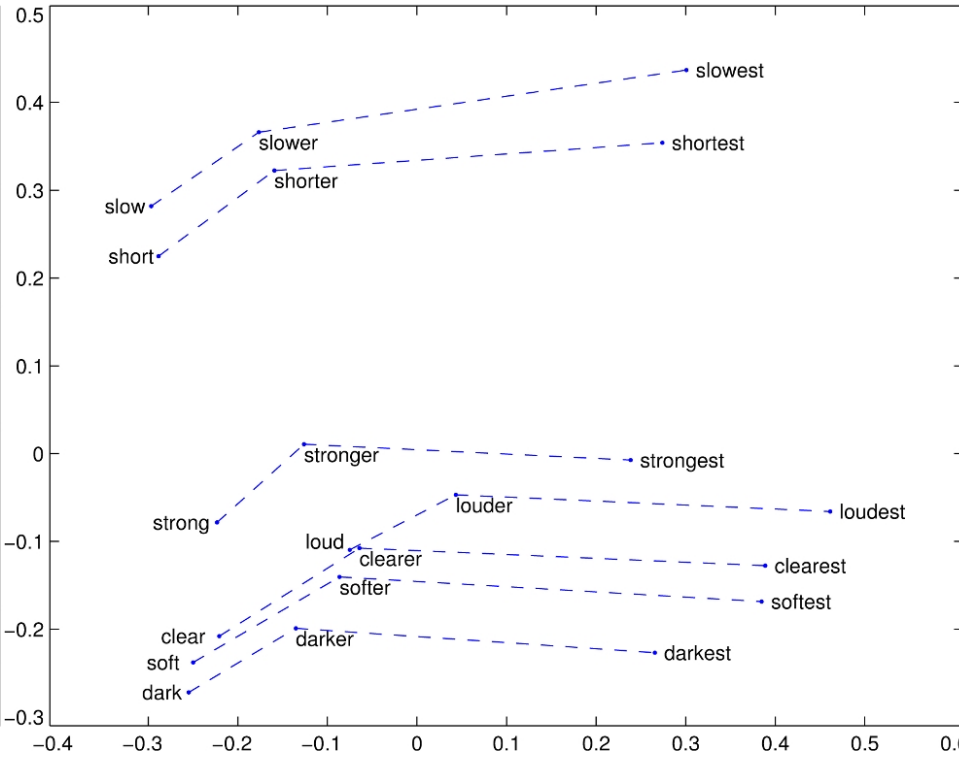
Word Embeddings – Linear substructures

- Analogy task:
  - *man* to *woman* is like *king* to *? (queen)*

$$\boldsymbol{x}_{\text{woman}} - \boldsymbol{x}_{\text{man}} + \boldsymbol{x}_{\text{king}} = \boldsymbol{x}^*$$

$$\boldsymbol{x}^* \approx \boldsymbol{x}_{\text{queen}}$$





https://nlp.stanford.edu/projects/glove/

Agenda

- Introduction
- **Count-based word representation**
- Prediction-based word embedding

Intuition for Computational Semantics

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

J. R. Firth, A synopsis of linguistic theory 1930–1955 (1957)

drink

sacred

alcoholic

Tesgüino

beverage

out of corn

fermented

Mexico

bottle

Nida[1975]

fermentation

bottle

grain

medieval

Ale

brew

pale

bar

drink

alcoholic

Tesgüino ⟷ Ale

Algorithmic intuition:

Two words are related when they have common context words

Word-Document Matrix – recap

- $\mathbb{D}$ is a set of documents (plays of Shakespeare)
$$\mathbb{D} = [d1, d2, \ldots, dM]$$
- $\mathbb{V}$ is the set of words (vocabularies) in dictionary
$$\mathbb{V} = [v1, v2, \ldots, vN]$$
- Words as rows and documents as columns
- Values: term count $\mathrm{tc}_{v,d}$
- Matrix size $N \times M$

	$d1$	$d2$	$d3$	$d4$
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0
...	...	…	...	...

Cosine

- Cosine is the normalized dot product of two vectors
  - Its result is between -1 and +1

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2} \cdot \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|_2} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

- $\boldsymbol{x} = [1 \quad 1 \quad 0]$ $\boldsymbol{y} = [4 \quad 5 \quad 6]$

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{1 * 4 + 1 * 5 + 0 * 6}{\sqrt{1^2 + 1^2 + 0^2}\sqrt{4^2 + 5^2 + 6^2}} = \frac{9}{\sim 12.4}$$

Word-Document Matrix

	$d1$	$d2$	$d3$	$d4$
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0
...	...	…	...	...

- Similarity between two words:

$$\text{similarity}(\text{soldier}, \text{clown}) = \cos(\boldsymbol{x}_{\text{soldier}}, \boldsymbol{x}_{\text{clown}})$$

Context

- Context can be
  - Document
  - Paragraph, tweet
  - **Window of (2-10) context words on each side of the word**

- Word-Context matrix
  - Every word as a unit (dimension)

$$\mathbb{C} = [c1, c2, \ldots, cL]$$

  - Matrix size: $N \times L$
  - Usually $\mathbb{C} = \mathbb{V}$ and therefore $L = N$

Word-Context Matrix

- Window context of 7 words

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

		$c1$	$c2$	$c3$	$c4$	$c5$	$c6$
		aardvark	computer	data	pinch	result	sugar
$v1$	apricot	0	0	0	1	0	1
$v2$	pineapple	0	0	0	1	0	1
$v3$	digital	0	2	1	0	1	0
$v4$	information	0	1	6	0	4	0

Word-to-Word Relations

		$c1$ aardvark	$c2$ computer	$c3$ data	$c4$ pinch	$c5$ result	$c6$ sugar
$v1$	apricot	0	0	0	1	0	1
$v2$	pineapple	0	0	0	1	0	1
$v3$	digital	0	2	1	0	1	0
$v4$	information	0	1	6	0	4	0

- First-order co-occurrence relation
  - Each cell of the word-context matrix
  - Words that appear in the proximity of each other
  - Like ***drink*** to ***beer***, and ***drink*** and ***wine***
- Second-order similarity relation
  - Cosine similarity between the representation vectors
  - Words that appear in similar contexts
  - Like ***beer*** to ***wine***, ***tesgüino*** to ***ale***, and ***frog*** to ***toad***

Point Mutual Information

- Problem with raw counting methods
  - Biased towards high frequent words (“and”, “the”) although they don’t contain much of information
- Point Mutual Information (PMI)
  - Rooted in information theory
  - A better measure for the first-order relation in word-context matrix in order to reflect the informativeness of co-occurrences
  - Joint probability of two events (random variables) divided by their marginal probabilities

$$\mathrm{PMI}(X, Y) = \log \frac{p(\mathrm{X}, \mathrm{Y})}{p(\mathrm{X})p(\mathrm{Y})}$$

Point Mutual Information

$$\mathrm{PMI}(v, c) = \log \frac{p(v, c)}{p(v)p(c)}$$

$$p(v, c) = \frac{\#\langle v, c \rangle}{S}$$

$$p(v) = \frac{\sum_{j=1}^{L} \#\langle v, c_j \rangle}{S} \qquad p(c) = \frac{\sum_{i=1}^{N} \#\langle v_i, c \rangle}{S}$$

$$S = \sum_{i=1}^{N} \sum_{j=1}^{L} \#\langle v_i, c_j \rangle$$

- Positive Point Mutual Information (PPMI)

$$\mathrm{PPMI}(t, c) = \max(\mathrm{PMI}, 0)$$

Point Mutual Information

		$c1$ aardvark	$c2$ computer	$c3$ data	$c4$ pinch	$c5$ result	$c6$ sugar
$v1$	apricot	0	0	0	1	0	1
$v2$	pineapple	0	0	0	1	0	1
$v3$	digital	0	2	1	0	1	0
$v4$	information	0	1	6	0	4	0

$$P(v = \text{information}, c = \text{data}) = {}^{6}/_{19} = .32$$

$$P(v = \text{information}) = {}^{11}/_{19} = .58$$

$$P(c = \text{data}) = {}^{7}/_{19} = .37$$

$$\text{PPMI}(v = \text{information}, c = \text{data}) = \max(0, \log \frac{.32}{.58 * .37}) = .39$$

Singular Value Decomposition - Recap

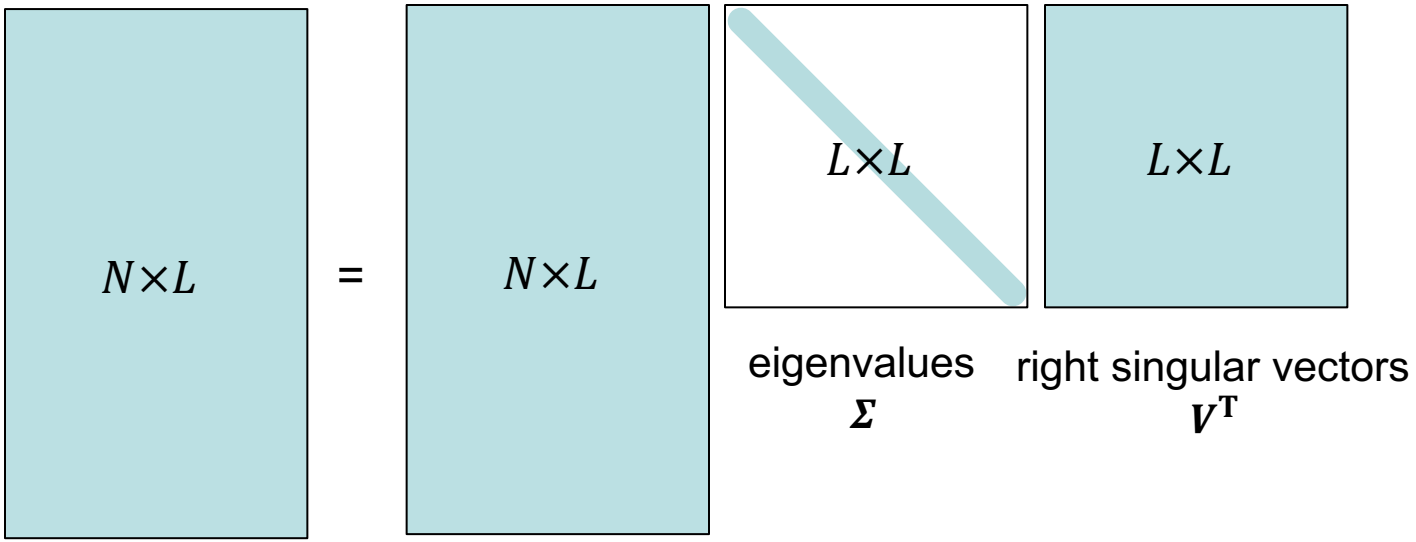
- An $N \times L$ matrix $\boldsymbol{X}$ can be factorized to three matrices:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$$

- $\boldsymbol{U}$ (left singular vectors) is an $N \times L$ unitary matrix
- $\boldsymbol{\Sigma}$ is an $L \times L$ diagonal matrix, diagonal entries
  - are eigenvalues,
  - show the importance of corresponding L dimensions in $\boldsymbol{X}$
  - are all positive and sorted from large to small values
- $\boldsymbol{V}^{\mathrm{T}}$ (right singular vectors) is an $L \times L$ unitary matrix

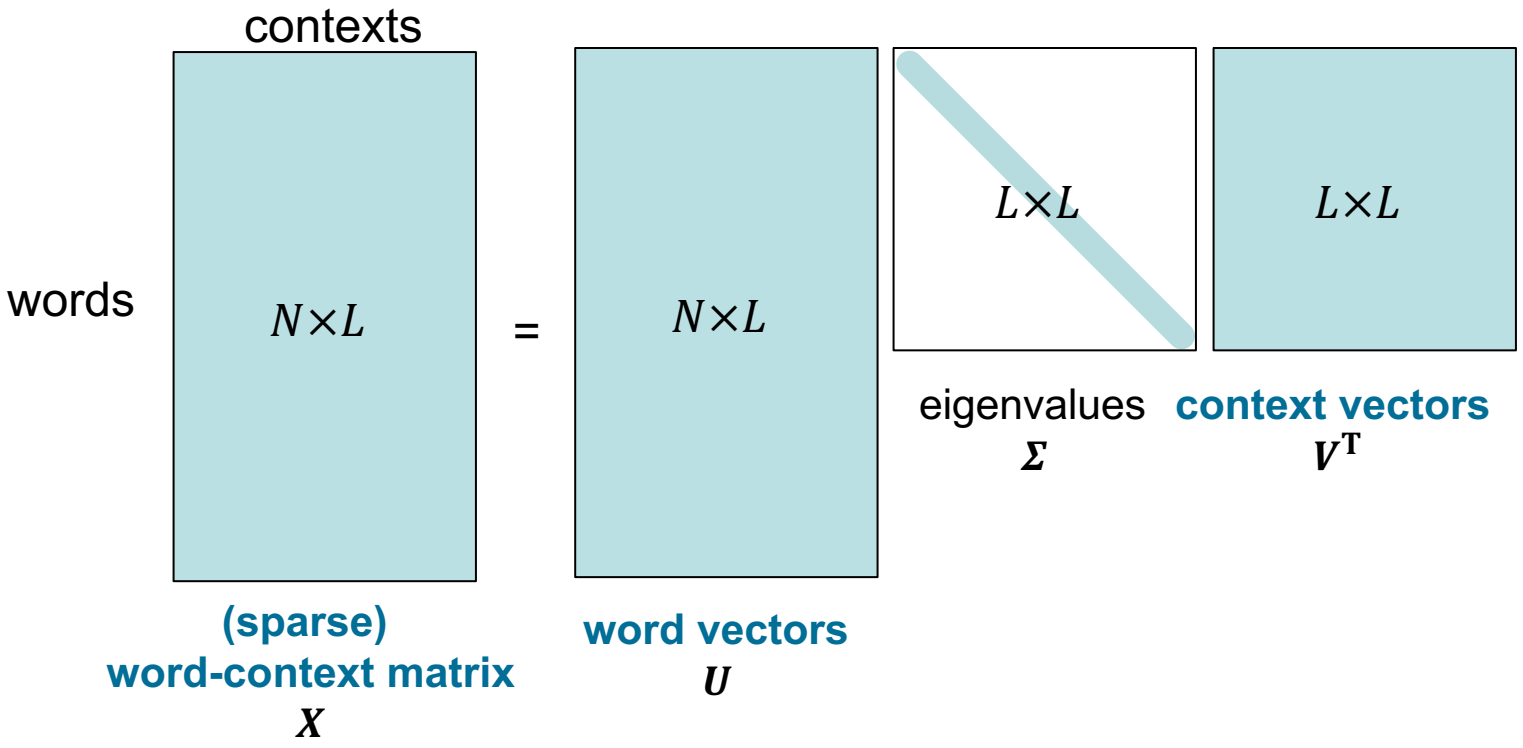
* The definition of SVD is simplified. Refer to https://en.wikipedia.org/wiki/Singular_value_decomposition for the exact definition

Singular Value Decomposition – Recap



Applying SVD to Word-Context Matrix

- Step 1: create a sparse PPMI matrix of the size $N \times L$,
- Apply SVD

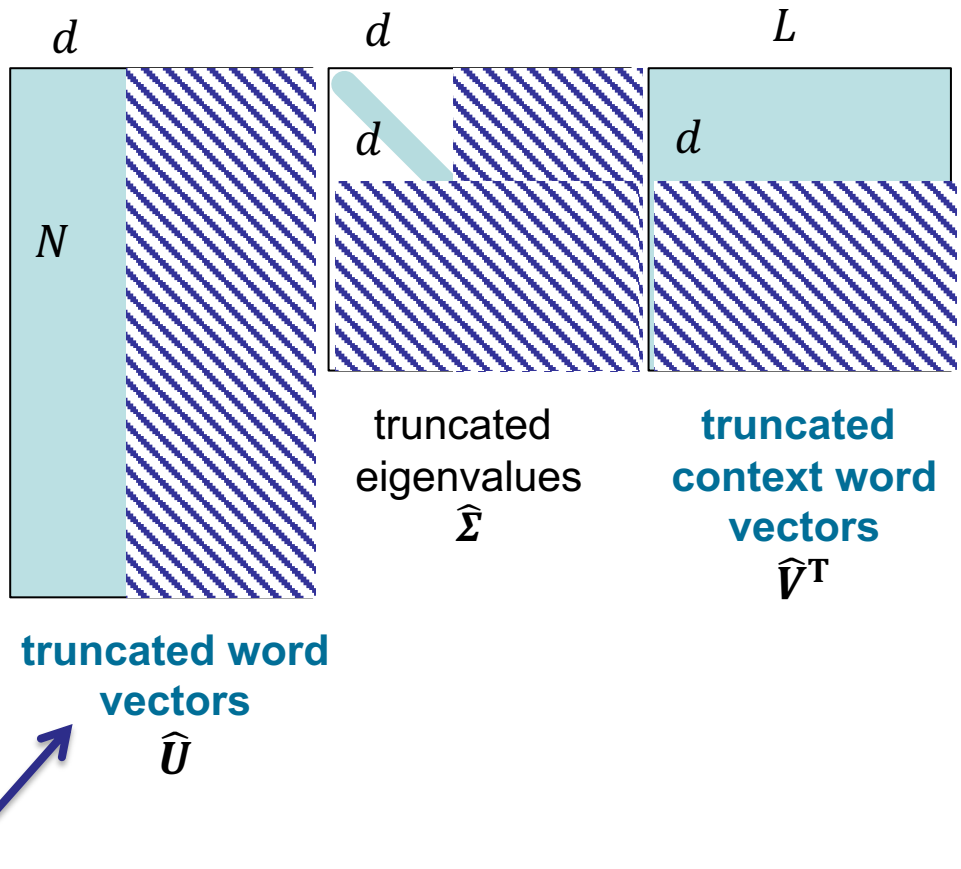


Applying SVD to Term-Context Matrix

- Step 2: keep only top d eigenvalues in $\boldsymbol{\Sigma}$ and set the rest to zero

- Truncate the $\boldsymbol{U}$ and $\boldsymbol{V}^{\mathbf{T}}$ matrices based on the changes in $\boldsymbol{\Sigma}$, called $\widehat{\boldsymbol{U}}$ and $\widehat{\boldsymbol{V}}^{\mathbf{T}}$ respectively

Applying SVD to Term-Context Matrix



- $\widehat{U}$ matrix is the dense low-dimensional word vectors

Agenda

- Introduction
- Count-based word representation
- **Prediction-based word embedding**

Word Embedding with Neural Networks

Recipe for creating (dense) word embedding with neural networks

- Design a neural network architecture!
- Loop over training data (v, c) for some epochs
  - Pass the word v as input and execute forward passing
  - Calculate the probability of observing context word c at output:

$$p(c|v)$$

  - Optimize the network to maximize this likelihood probability

Details come next!

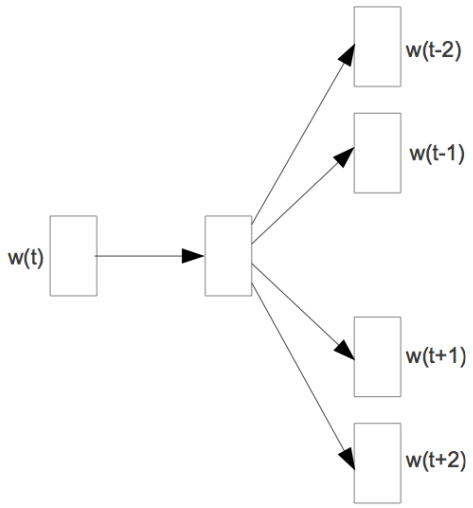
Training Data $\mathcal{D}$

Window of size 2

Source Text

Training Samples

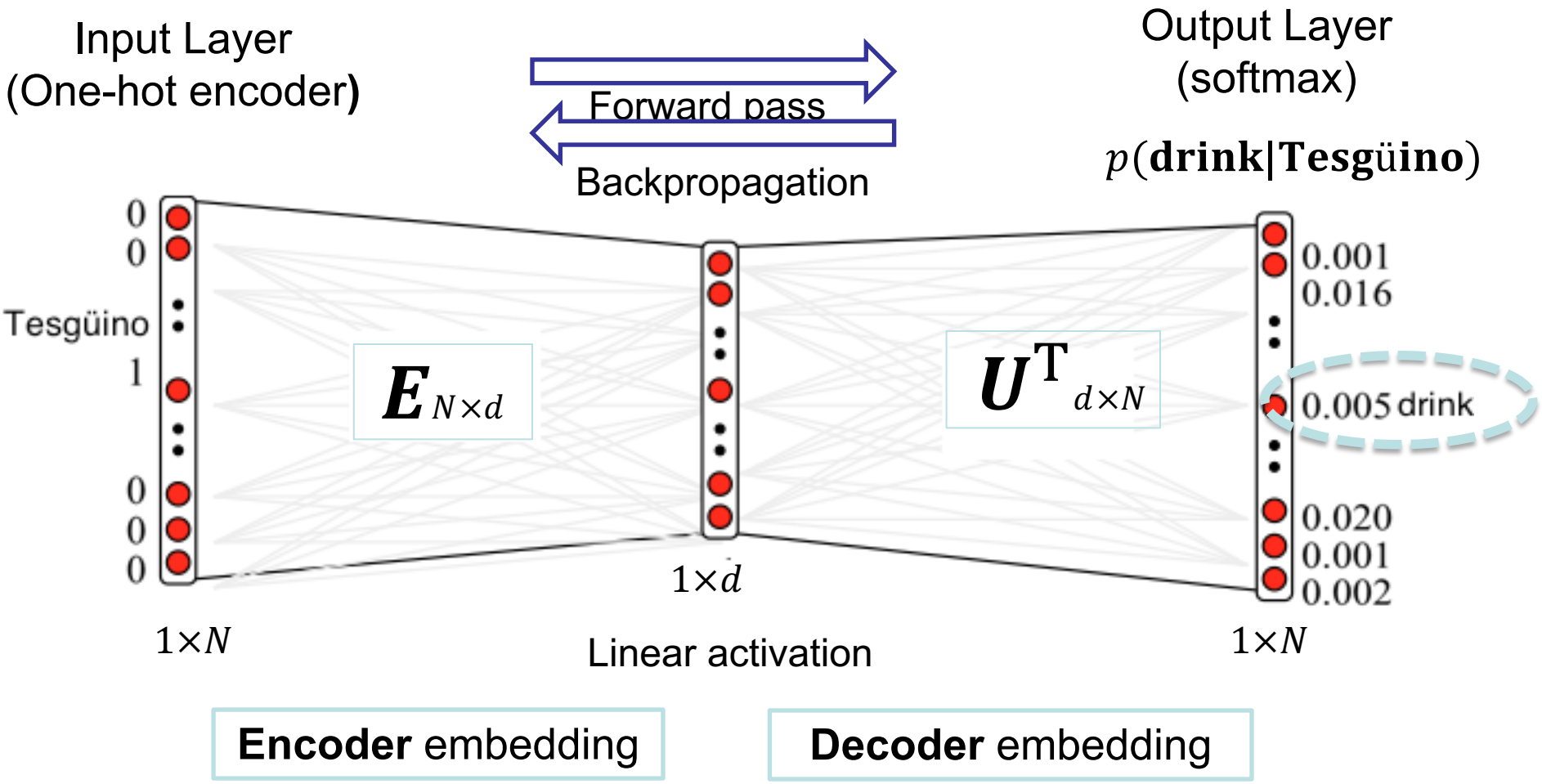
The quick brown fox jumps over the lazy dog.	(the, quick) (the, brown)
The **quick** brown fox jumps over the lazy dog.	(quick, the) (quick, brown) (quick, fox)
The quick **brown** fox jumps over the lazy dog.	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown **fox** jumps over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



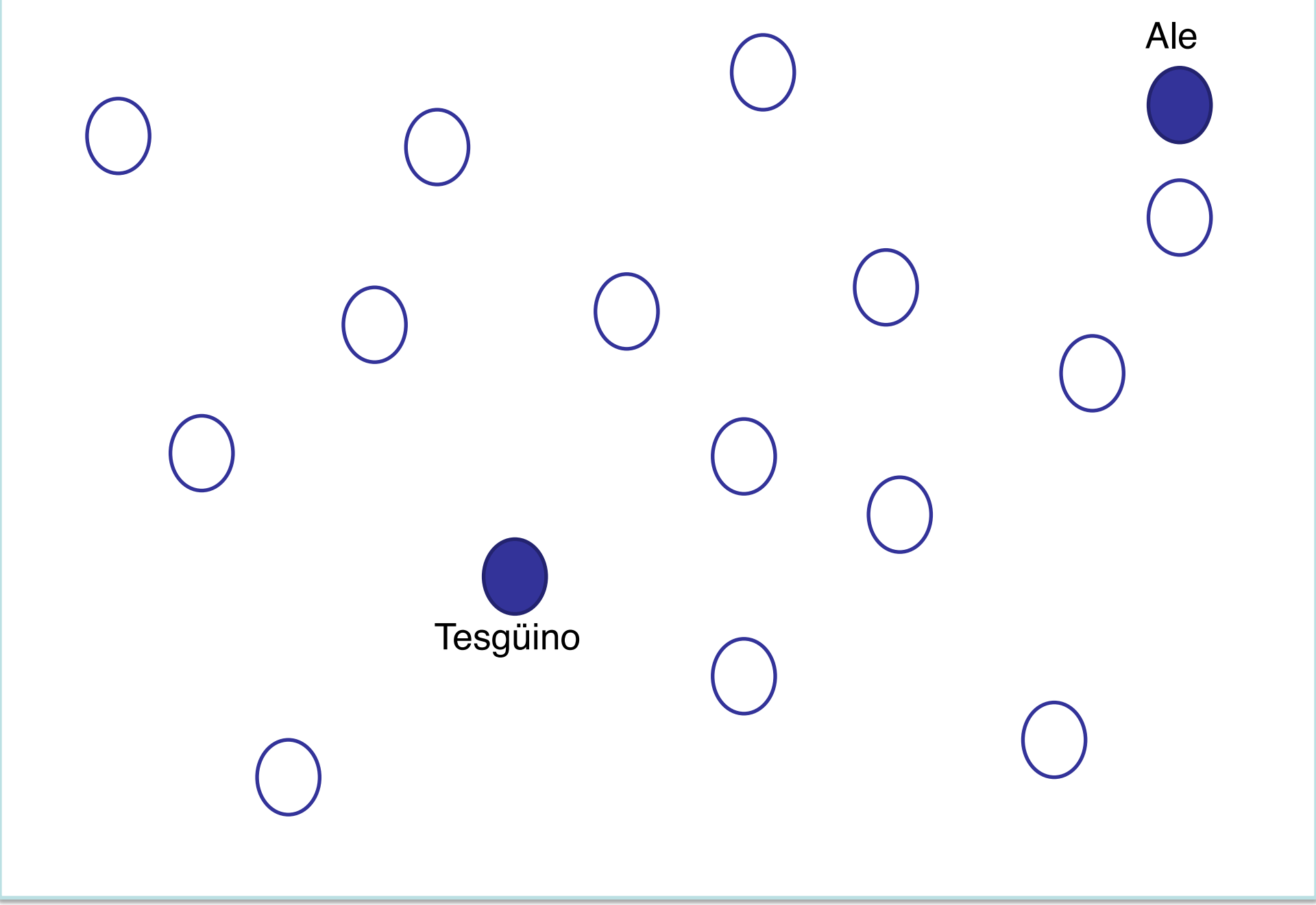
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Neural embeddings – architecture

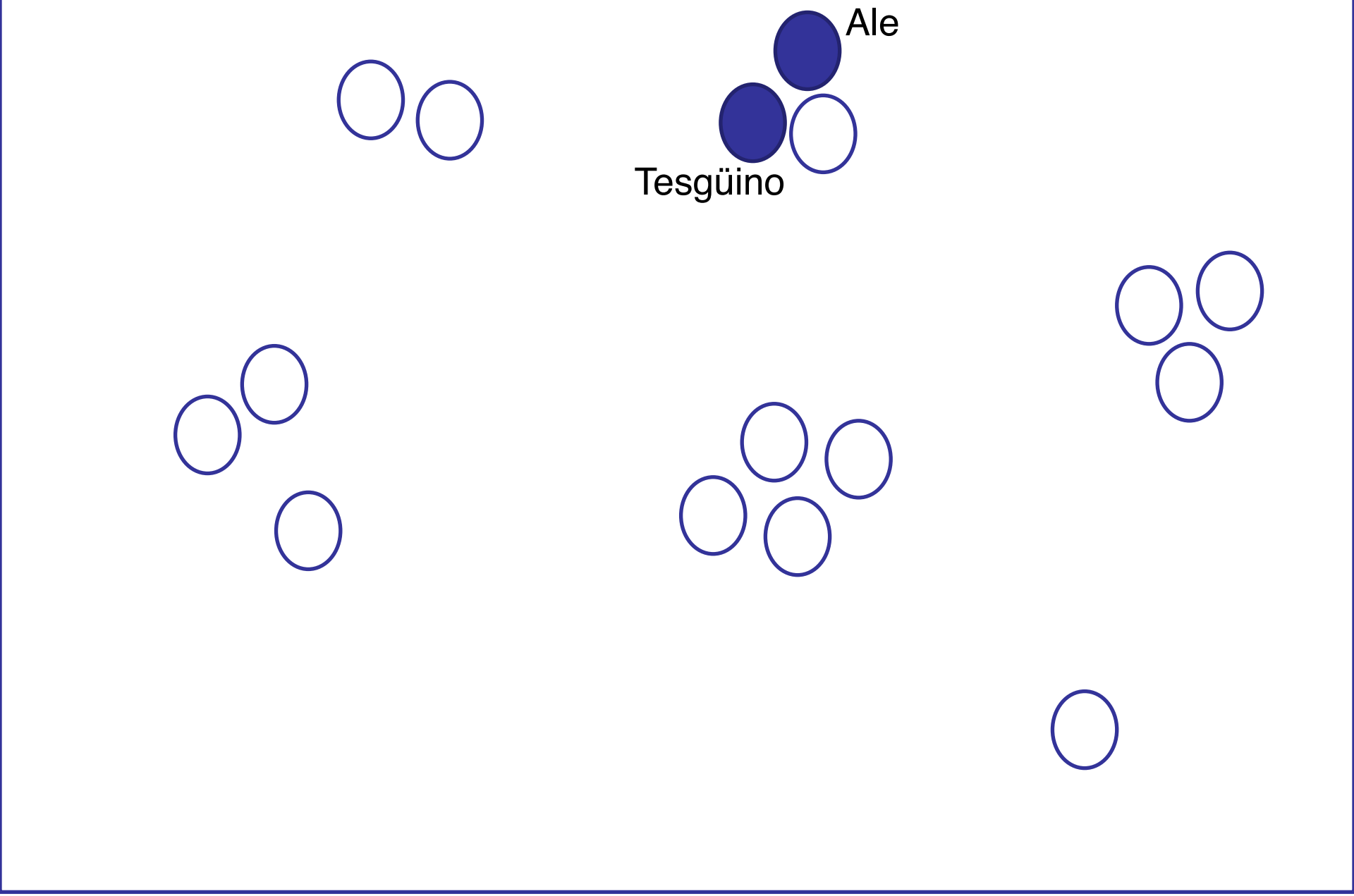
Train sample: (***Tesgüino***, ***drink***)



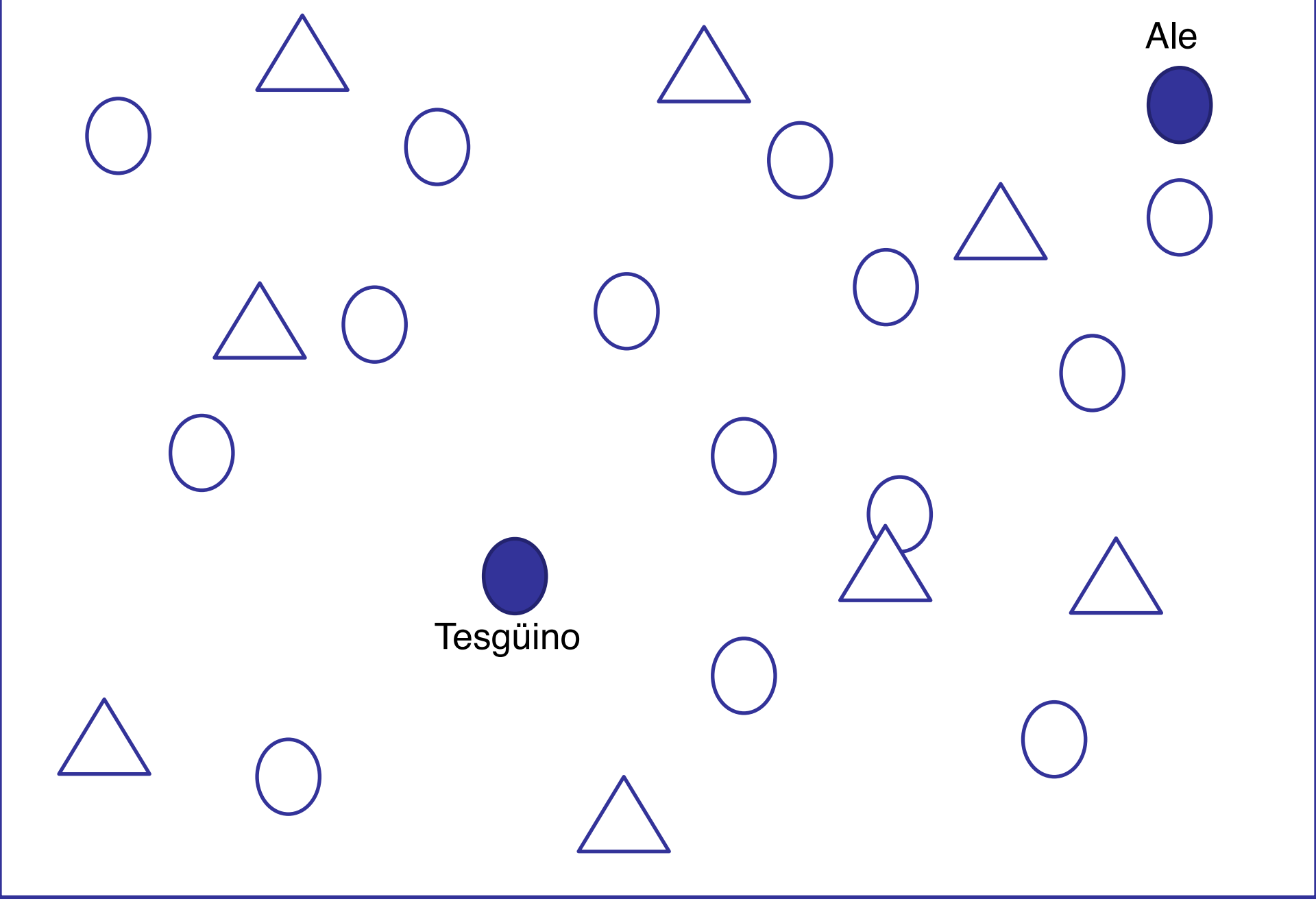
https://web.stanford.edu/~jurafsky/slp3/



Embedding vector

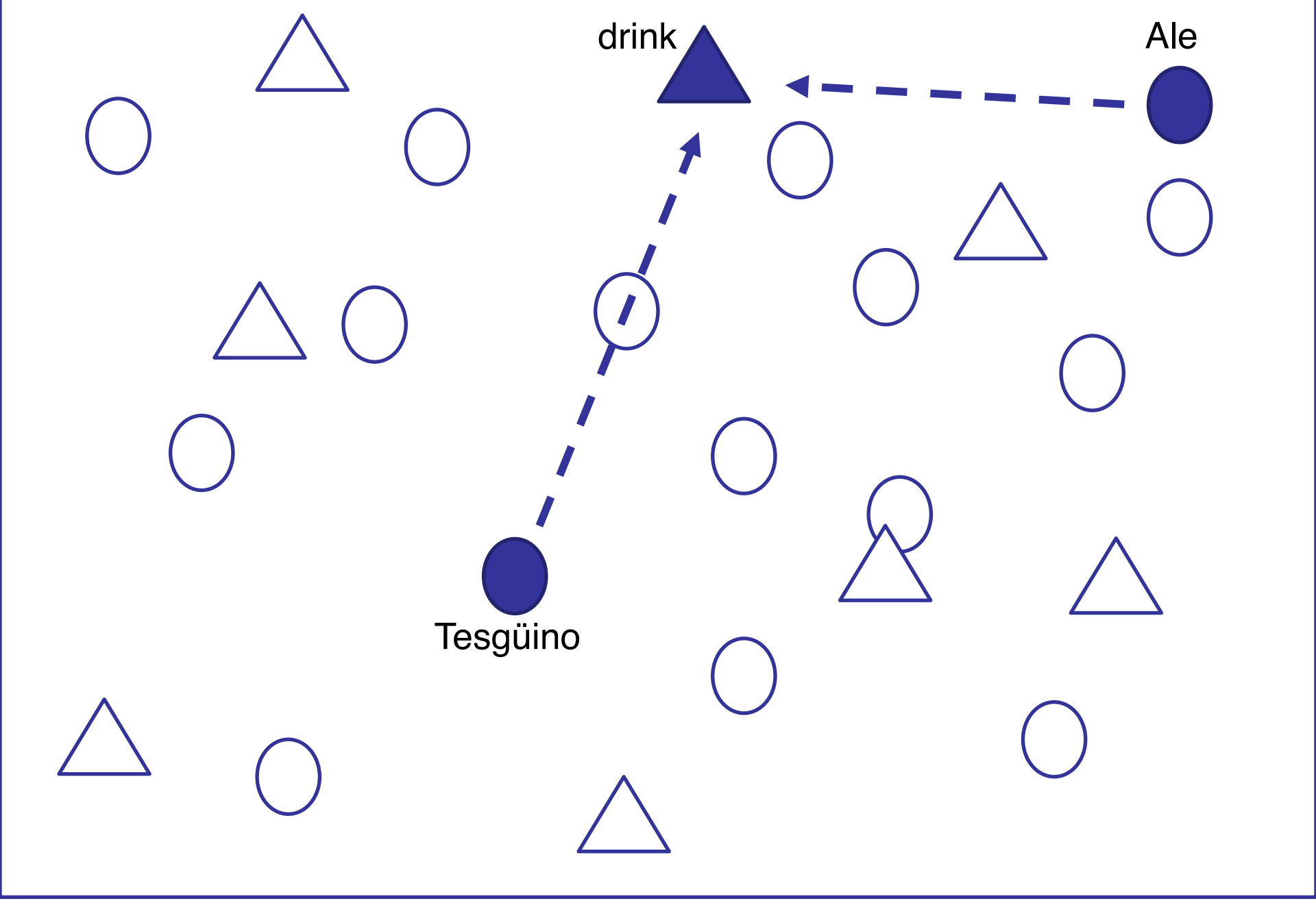


Embedding vector



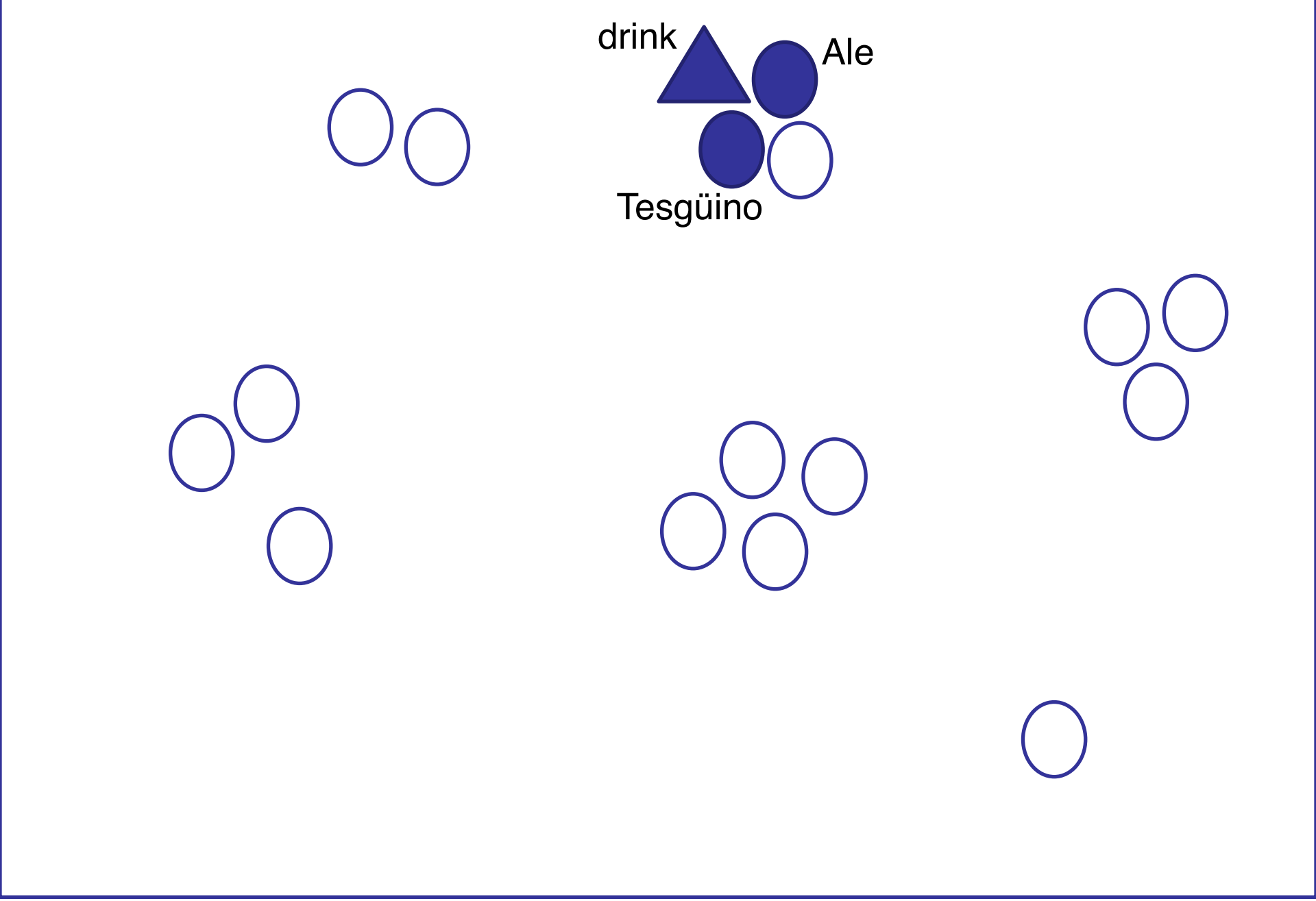
Embedding vector

Decoding vector



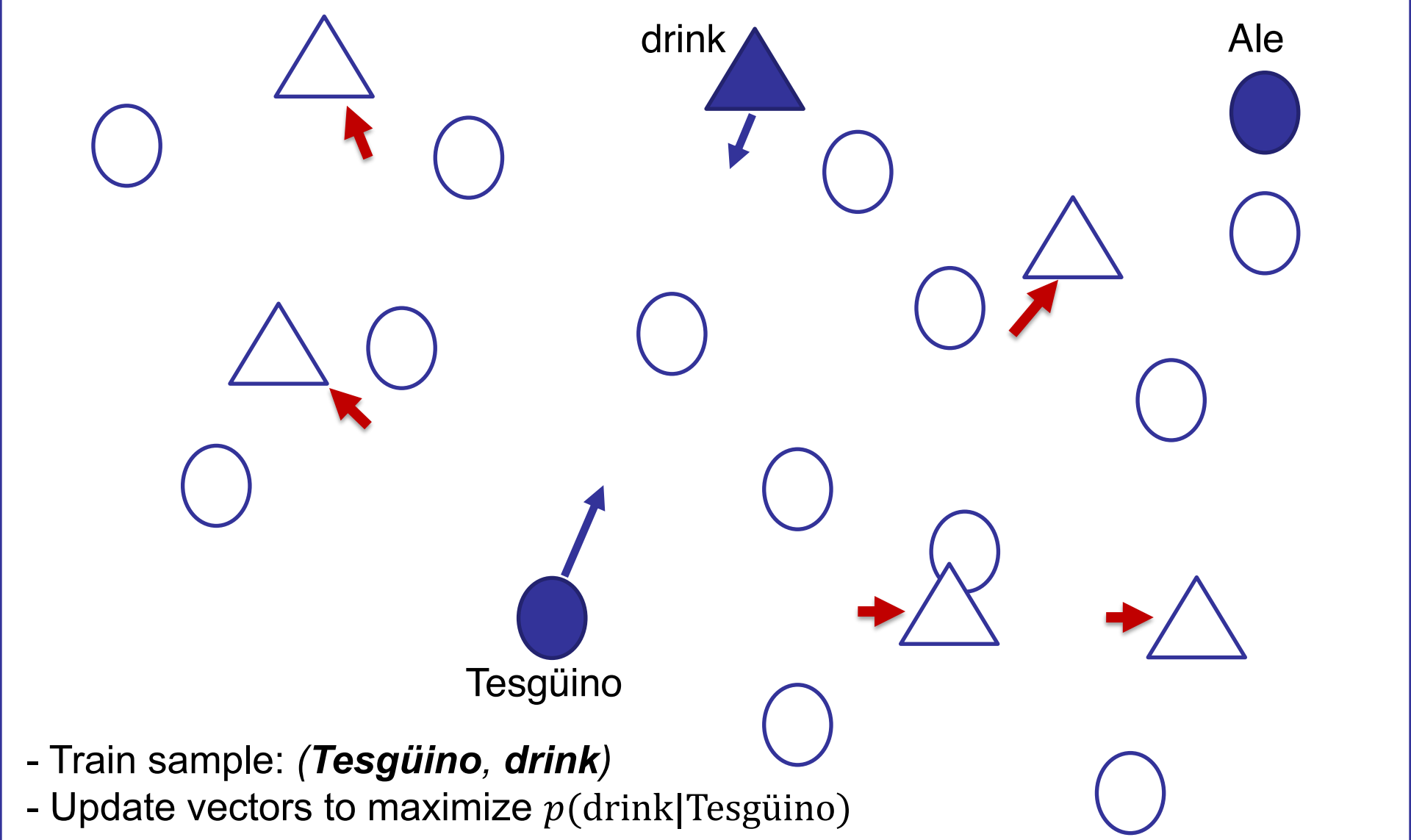
Embedding vector

Decoding vector



Embedding vector

Decoding vector



- Train sample: *(**Tesgüino**, **drink**)*
- Update vectors to maximize $p(\text{drink}|\text{Tesgüino})$

Embedding vector

Decoding vector

Neural embeddings – prediction

- Since hidden layer is linear, output vector is:

$$\boldsymbol{z} = \boldsymbol{e}_v \boldsymbol{U}^{\mathrm{T}}$$

- Probability distribution of output words using softmax:

$$p(c|v) = \mathrm{softmax}(\boldsymbol{z})_c = \frac{\exp\left(\boldsymbol{e}_v \boldsymbol{u}_c^{\mathrm{T}}\right)}{\sum_{\tilde{c}\in\mathbb{V}} \exp\left(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}}^{\mathrm{T}}\right)}$$

- In this example:

$$p(\mathrm{drink}|\mathrm{Tesgüino}) = \frac{\exp\left(\boldsymbol{e}_{\mathrm{Tesgüino}} \boldsymbol{u}_{\mathrm{drink}}^{\mathrm{T}}\right)}{\sum_{\tilde{c}\in\mathbb{V}} \exp\left(\boldsymbol{e}_{\mathrm{Tesgüino}} \boldsymbol{u}_{\tilde{c}}^{\mathrm{T}}\right)}$$

Denominator (normalization) is expensive!

Neural embeddings – loss

- Loss function with NLL over all training samples $\mathcal{D}$

$$\mathcal{L} = -\mathbb{E}_{(v,c)\sim\mathcal{D}} \log p(c|v)$$

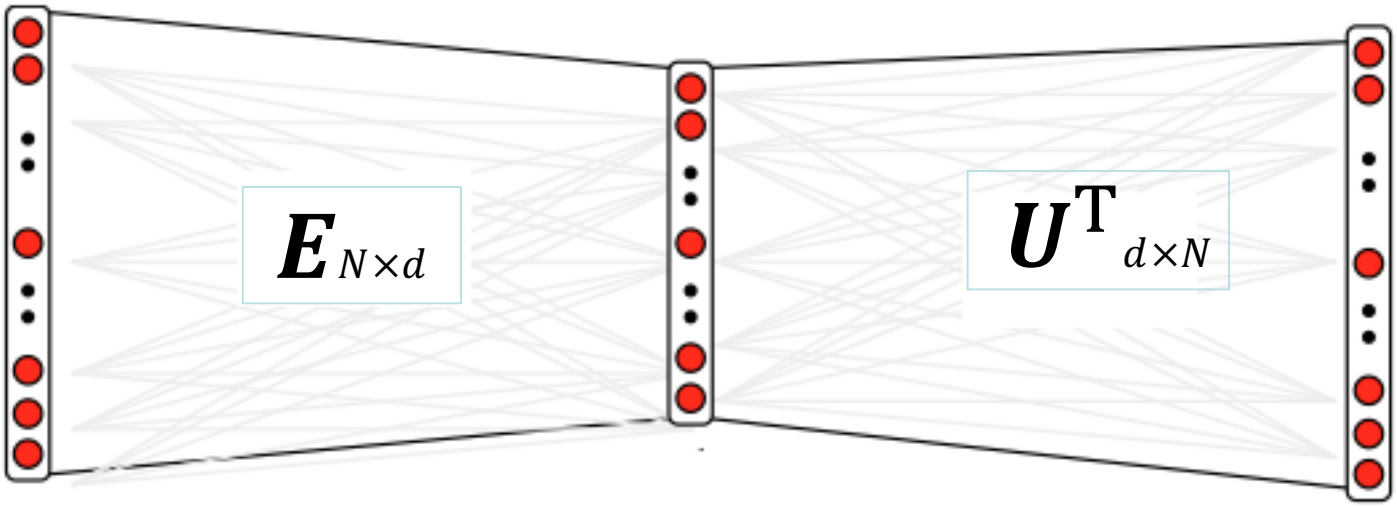
$$\mathcal{L} = -\mathbb{E}_{(v,c)\sim\mathcal{D}} \left[\log \frac{\exp\left(\boldsymbol{e}_v \boldsymbol{u}_c^{\mathrm{T}}\right)}{\sum_{\tilde{c}\in\mathbb{V}} \exp\left(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}}^{\mathrm{T}}\right)} \right]$$

$$\mathcal{L} = -\mathbb{E}_{(v,c)\sim\mathcal{D}} \left[\boldsymbol{e}_v \boldsymbol{u}_c^{\mathrm{T}} - \log \sum_{\tilde{c}\in\mathbb{V}} \exp\left(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}}^{\mathrm{T}}\right) \right]$$

Neural embeddings – word embedding

- word2vec uses the encoding matrix $\boldsymbol{E}$ as the final **word embeddings**

- Other possibilities
  - Mean of the vectors of $\boldsymbol{E}$ and $\boldsymbol{U}$ for each word
  - Concatenation of the vectors in $\boldsymbol{E}$ and $\boldsymbol{U}$ for each word
    - This results in vectors with $2d$ dimensions



word2vec (Skip-Gram) with Negative Sampling

- Word2vec is an efficient and effective algorithm that proposes Negative Sampling method to define loss

- In Negative Sampling instead of $p(c|v)$, the network estimates $p(y = 1|v, c)$: the probability that the co-occurrence of (v, c) comes from a genuine distribution, namely from training corpus

- $p(y = 1|v, c)$ is defined using sigmoid σ:

$$p(y = 1|v, c) = \sigma(\boldsymbol{e}_v \cdot \boldsymbol{u}_c^{\mathrm{T}})$$

word2vec (Skip-Gram) with Negative Sampling

- When two words (v, c) appear in the training data (genuine distribution), it is a positive sample

- Negative Sampling aims to distinguish between the co-occurrence probability of v in a positive sample

$$p(y = 1|v, c)$$

and the co-occurrences in k negative samples with context words $\tilde{c}$

$$p(y = 1|v, \tilde{c})$$

word2vec (Skip-Gram) with Negative Sampling

- Negative samples are drawn from the noisy distribution $\widetilde{\mathcal{D}}$
  - $\widetilde{\mathcal{D}}$ represents a randomly created corpus $\rightarrow$ words co-occur randomly!
  - Why a random co-occurrence can be assumed as a negative sample?

- The noisy distribution $\widetilde{\mathcal{D}}$ is defined using a *smooth* unigram distribution of words in the corpus,
  - In word2vec, $\widetilde{\mathcal{D}}$ is smoothed by raising unigram counts to the power of $\alpha = 0.75 \rightarrow$ Context Distribution Smoothing

- Number of negative samples k is usually between 2 to 20

word2vec with Negative Sampling – Objective Function

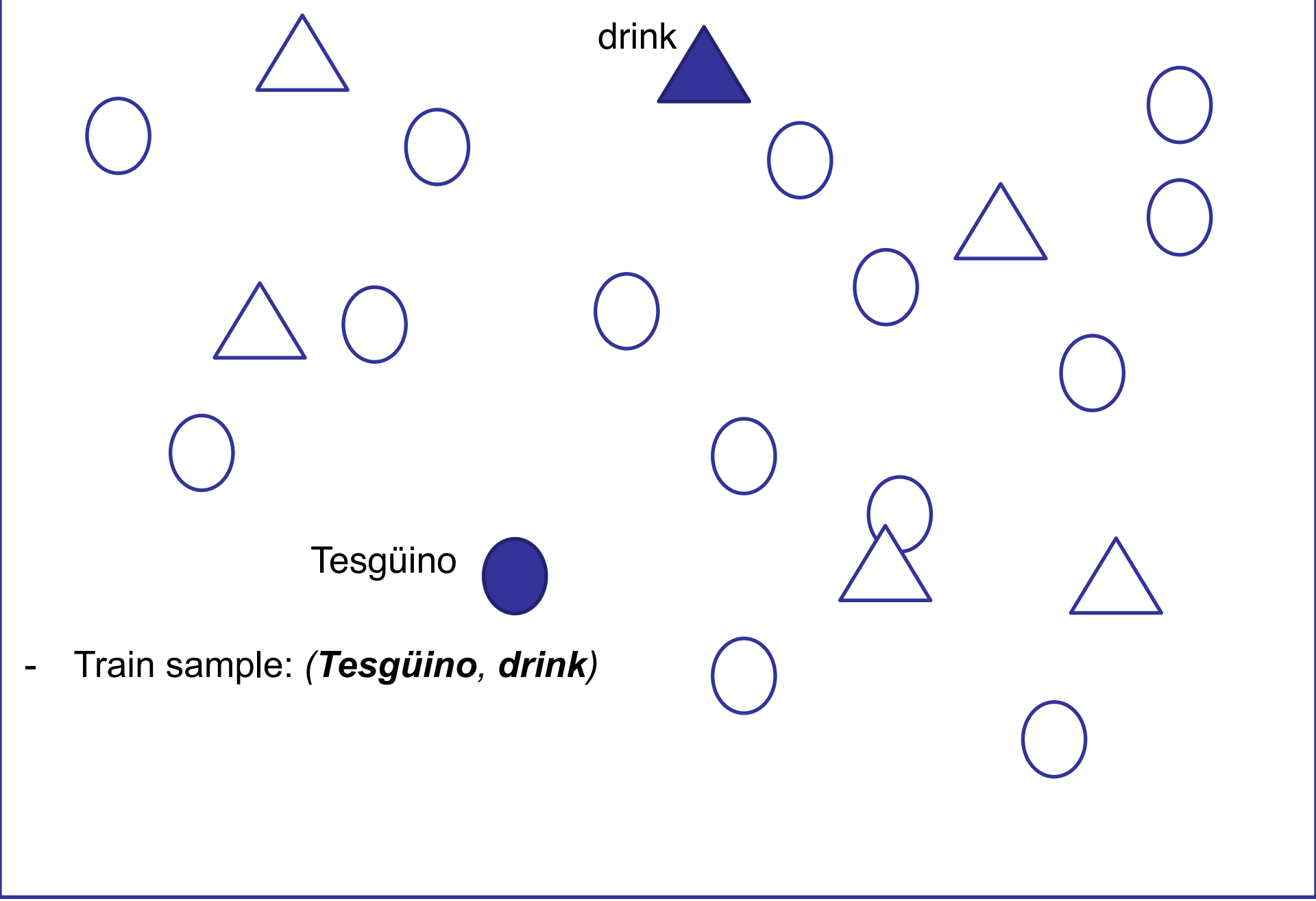
- The objective function
  - increases the probability for the positive sample (v, c)
  - decreases the probability for the k negative samples $(v, \tilde{c})$
- Loss function:

$$\mathcal{L} = -\mathbb{E}_{(v,c)\sim\mathcal{D}}\left[\log p(y=1|v,c) - \sum_{\substack{\tilde{c}\sim\tilde{\mathcal{D}} \\ k \text{ times}}} \log p(y=1|v,\tilde{c})\right]$$

$$\mathcal{L} = -\mathbb{E}_{(v,c)\sim\mathcal{D}}\left[\log \sigma\left(\boldsymbol{e}_v \boldsymbol{u}_c^{\mathrm{T}}\right) - \sum_{\substack{\tilde{c}\sim\tilde{\mathcal{D}} \\ k \text{ times}}} \log \sigma\left(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}}^{\mathrm{T}}\right)\right]$$

positive samples

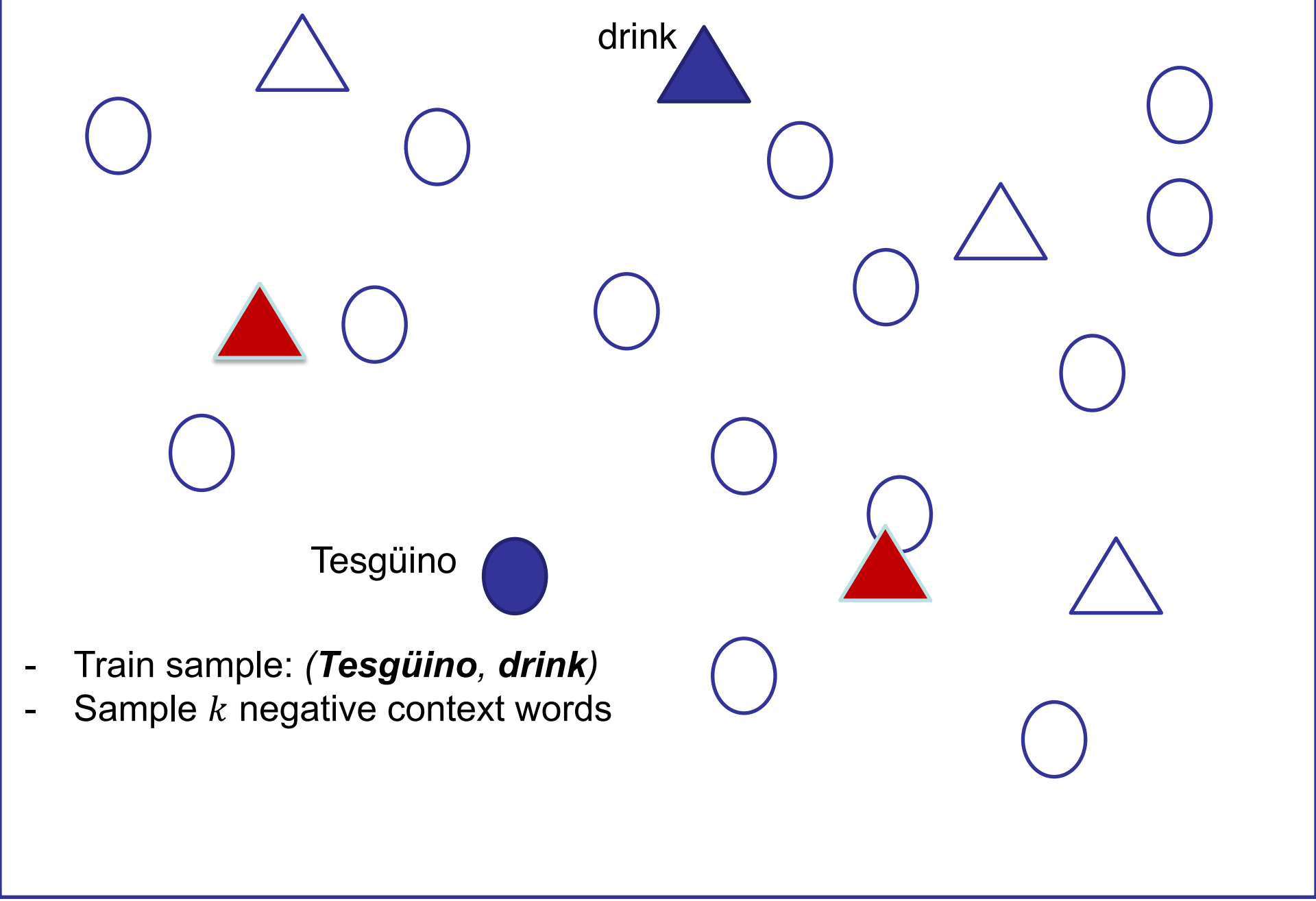
negative samples



- Train sample: *(**Tesgüino**, **drink**)*

Embedding vector

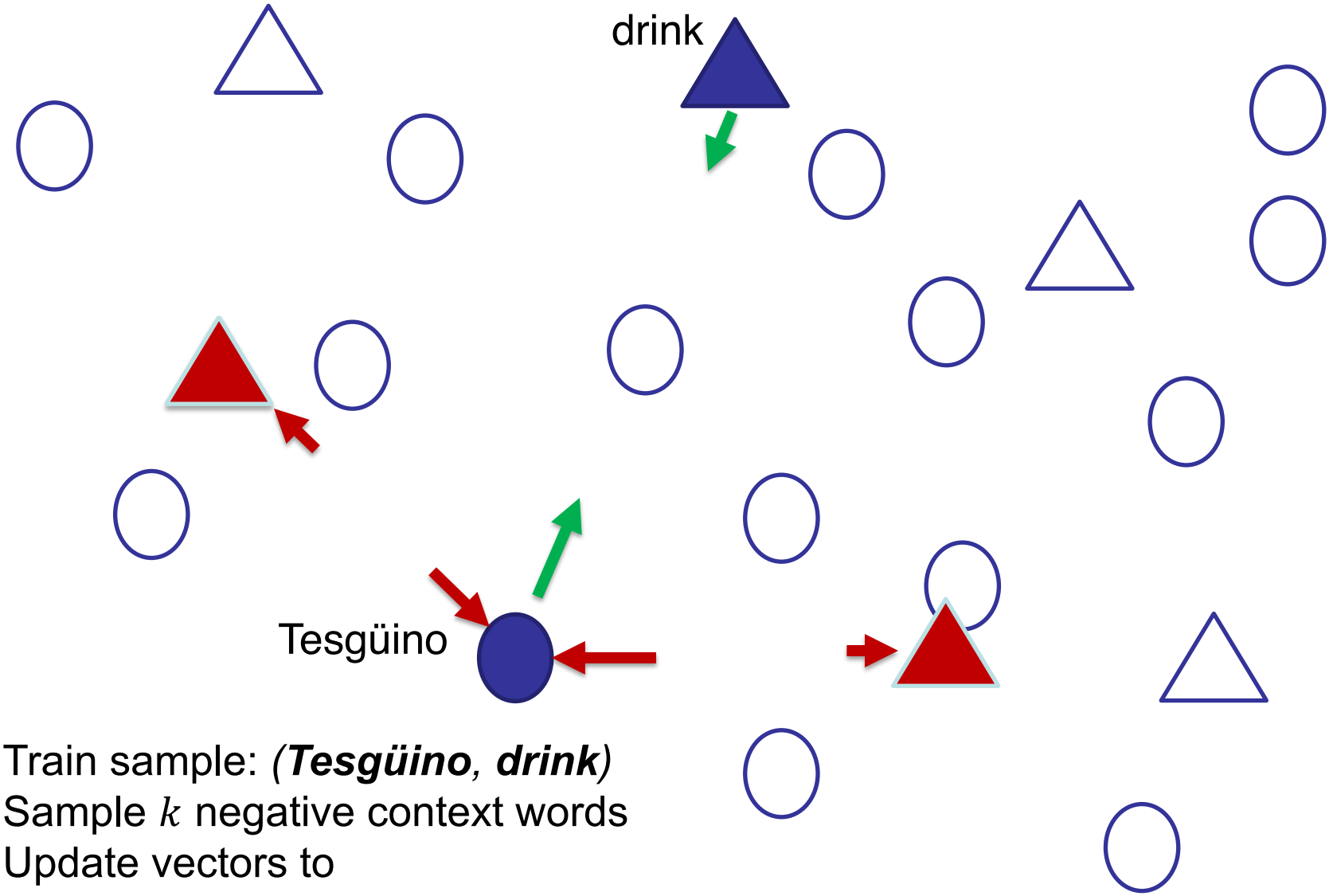
Decoding vector



- Train sample: *(**Tesgüino**, **drink**)*
- Sample k negative context words

Embedding vector

Decoding vector



- Train sample: *(**Tesgüino**, **drink**)*
- Sample k negative context words
- Update vectors to
  - Maximize $p(y = 1|\text{Tesgüino}, \text{drink})$
  - Minimize $p(y = 1|\text{Tesgüino}, \tilde{c})$

Embedding vector Decoding vector

Negative Sampling – final words!

- Negative Sampling turns the problem from multi-class classification to binary classification

- Negative Sampling is a biased approximation of softmax
  - Noisy Contrastive Estimation (the parent of Negative Sampling) is an unbiased approximation of softmax

- Softmax is a good choice for training language models, namely, to estimate $p(c|v)$

- word2vec and Negative Sampling aim to train good embeddings