344.075 KV: Natural Language Processing

Language Modeling & Neural Word Embedding

Navid Rekab-Saz

navid.rekabsaz@jku.at

JOHANNES KEPLER
UNIVERSITY LINZ

Institute of
Computational
Perception

Agenda

- *n*-gram language models
- Language modeling with neural networks
- word2vec

Agenda

- ***n*-gram language models**
- Language modeling with neural networks
- word2vec

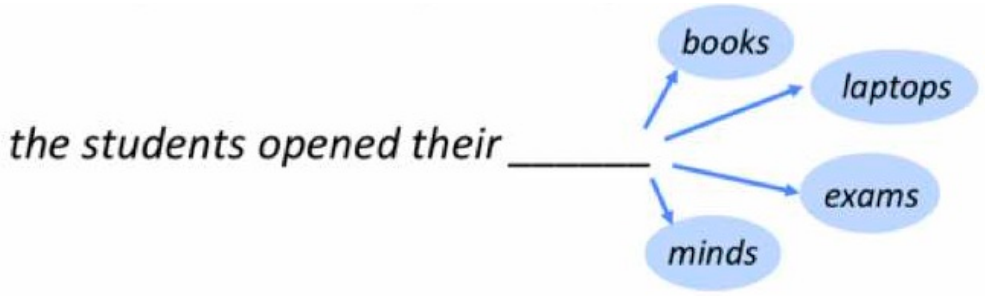
Some slides are adopted from http://web.stanford.edu/class/cs224n/

Language Modeling

- Language Modeling is the task of predicting a word (or a subword or character) given a context:

$$P(v|\text{context})$$

- A Language Model can answer the questions like



$P(v|\textit{the students opened their})$

Language Modeling – formal definition

- Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, a language model calculates the probability distribution of next word $x^{(t+1)}$ over all words in vocabulary

$$P(x^{(t+1)} | x^{(t)}, x^{(t-1)}, \ldots, x^{(1)})$$

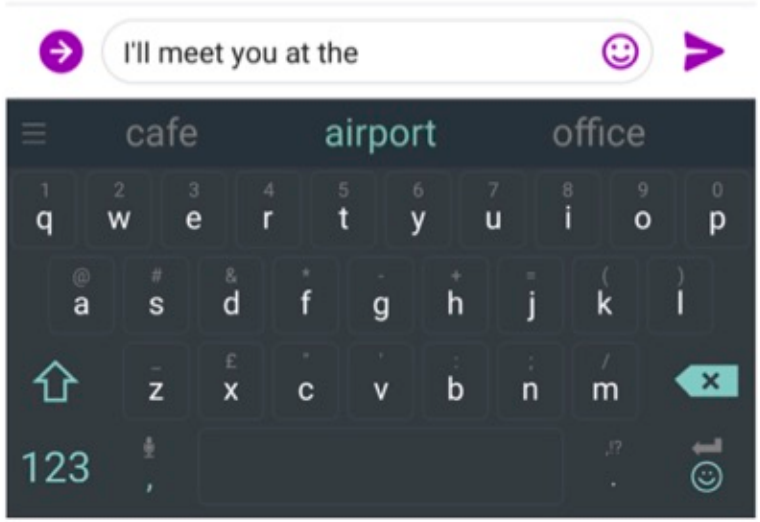
x is any word in the vocabulary $\mathbb{V} = \{v1, v2, \ldots, vN\}$

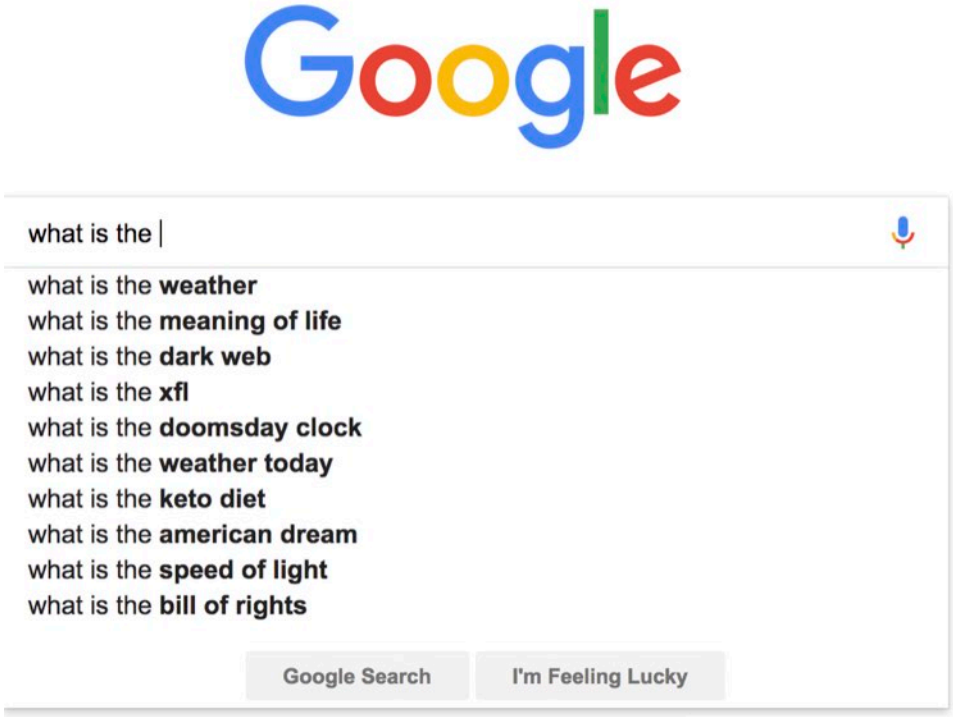
Why Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language

- Language Models is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Spelling/grammar correction
  - Speech recognition
  - Handwriting recognition
  - Machine translation
  - Summarization
  - Dialogue /chatbots
  - etc.

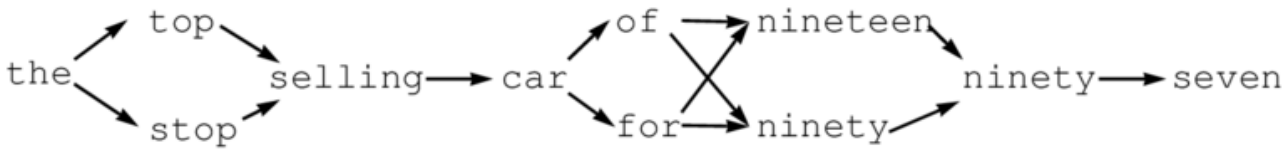
Language Modeling



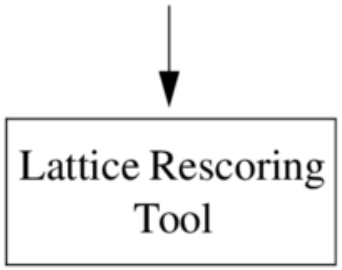


[link]

Language Modeling



Input Lattice




```
the top selling car of nineteen ninety seven
the top selling car of ninety ninety seven
the top selling car for ninety ninety seven
the stop selling car for nineteen ninety seven
```

N-Best List

n-gram Language Model

- Recall: a *n*-gram is a chunk of *n* consecutive words.

the students opened their _____

- unigrams: *"the", "students", "opened", "their"*
- bigrams: *"the students", "students opened", "opened their"*
- trigrams: *"the students opened", "students opened their"*
- 4-grams: *"the students opened their"*

- A *n*-gram Language Model collects frequency statistics of different *n*-grams in a corpus, and use these to calculate probabilities

n-gram Language Model

- Markov assumption: decision at time t depends only on the current state

- In *n*-gram Language Model: predicting $x^{(t+1)}$ depends on preceding *n-1* words

- Without Markovian assumption:

$$P(x^{(t+1)}|x^{(t)}, x^{(t-1)}, \dots, x^{(1)})$$

- *n*-gram Language Model:

$$P(x^{(t+1)}|\underbrace{x^{(t)}, x^{(t-1)}, \dots, x^{(t-n+2)}}_{n\text{-}1 \text{ words}})$$

n-gram Language Model

- Based on definition of conditional probability:

$$P\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(t-n+2)}\right) = \frac{P\left(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)}\right)}{P\left(x^{(t)}, \dots, x^{(t-n+2)}\right)}$$

- The *n*-gram probability is calculated by counting *n*-grams and [*n–1*]-grams in a large corpus of text:

$$P\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(t-n+2)}\right) \approx \frac{\text{count}\left(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)}\right)}{\text{count}\left(x^{(t)}, \dots, x^{(t-n+2)}\right)}$$

n-gram Language Model

- Example: learning a 4-gram Language Model

~~*as the exam clerk started the clock, the*~~ *students opened their* _____

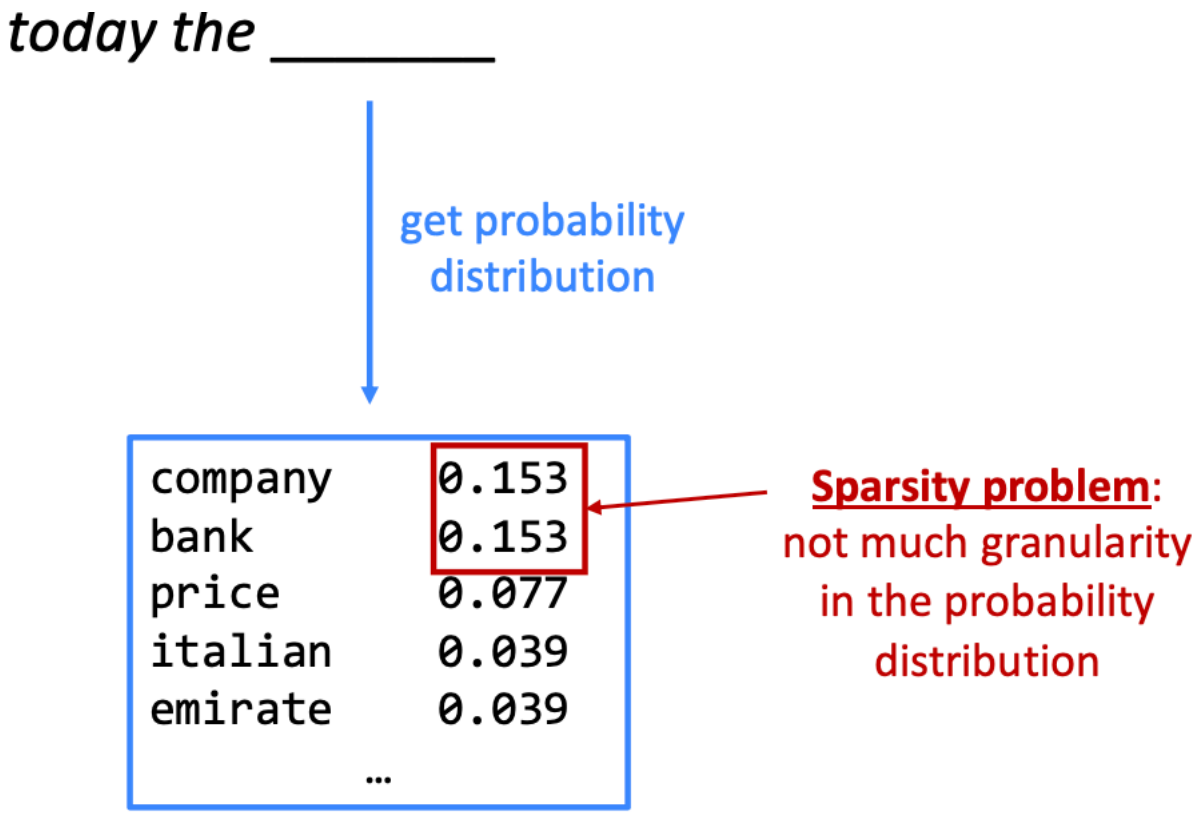
condition on this

$$P(v|\textit{students opened their}) = \frac{P(\textit{students opened their } v)}{P(\textit{students opened their})}$$

- For example, suppose that in the corpus:
  - "*students opened their*" occurred 1000 times
  - "*students opened their books*" occurred 400 times
    - P(*books* | *students opened their*) = 0.4
  - "*students opened their exams*" occurred 100 times
    - P(*exams* | *students opened their*) = 0.1

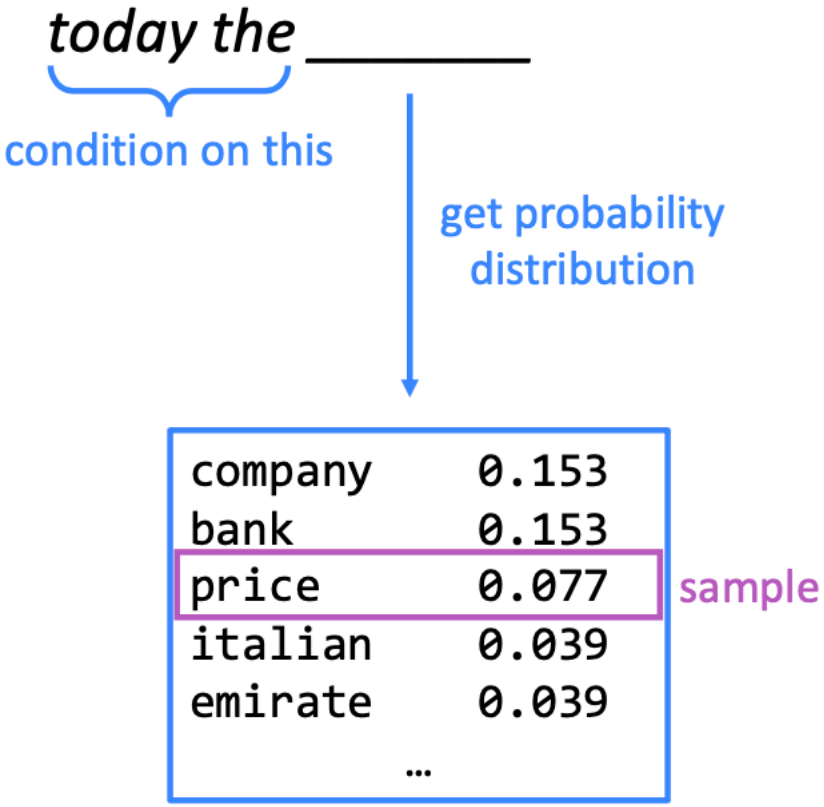
n-gram Language Models – generating text

- A trigram Language Model trained on Reuters corpus (1.7 M words)



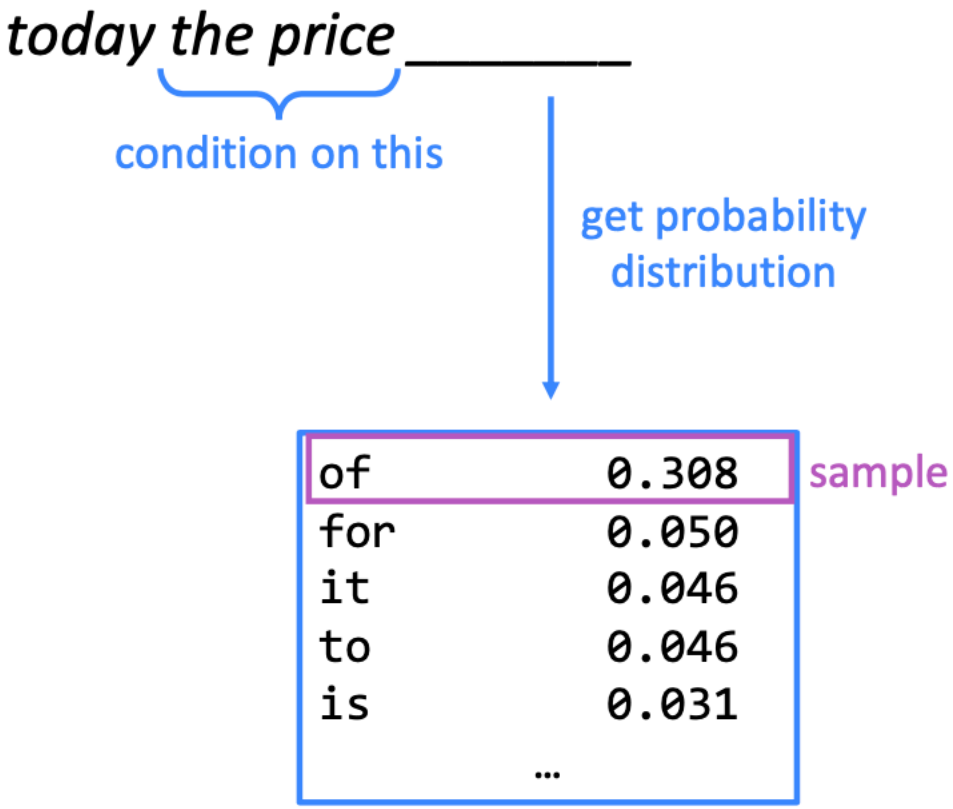
n-gram Language Models – generating text

- Generating text by sampling from the probability distributions



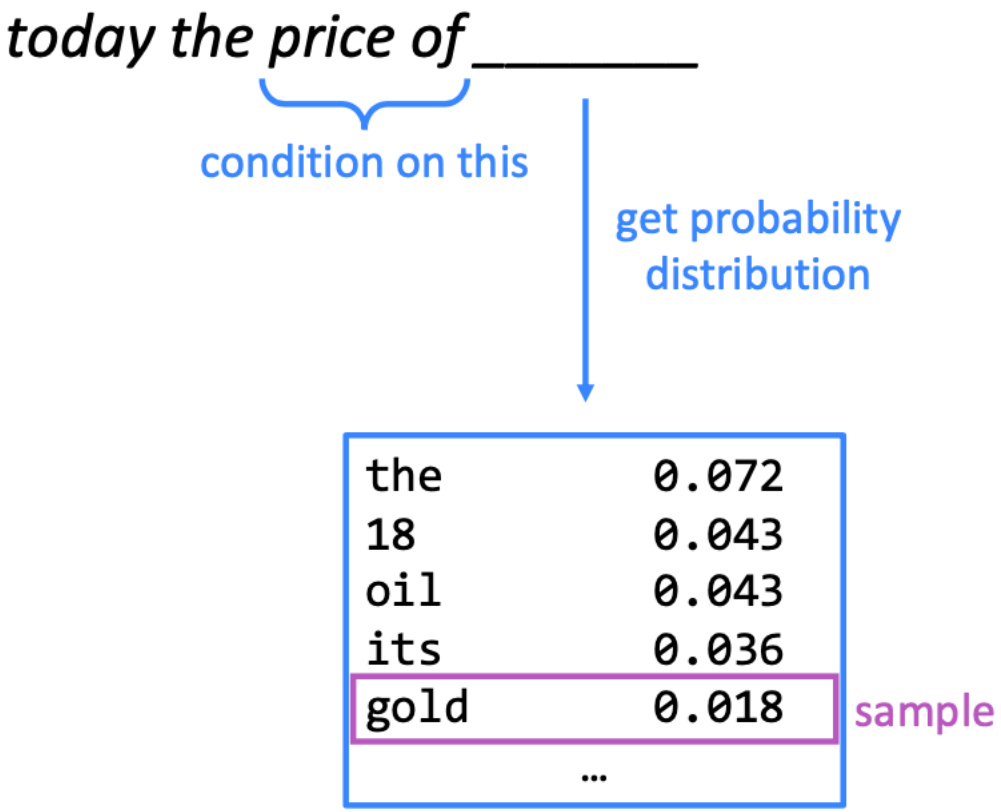
n-gram Language Models – generating text

- Generating text by sampling from the probability distributions



n-gram Language Models – generating text

- Generating text by sampling from the probability distributions



n-gram Language Models – generating text

- Generating text by sampling from the probability distributions

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

- Very good in syntax … but incoherent!
- Increasing *n* makes the text more coherent but also intensifies the discussed issues

n-gram Language Model – issues

- Sparsity
  - What if the nominator „*students opened their* v“ never occurred in corpus?
    - Smoothing: add small hyper-parameter δ to all words
  - What if the denominator „*students opened their*“ never occurred in corpus?
    - Backoff: condition on “*students opened*” instead
  - Increasing *n* makes the sparsity problem worse!
- Storage
  - The model needs to store all *n*-grams (from unigram to *n*-gram), observed in the corpus
  - Increasing *n* radically worsens the storage problem!

Agenda

- *n*-gram language models
- **Language modeling with neural networks**
- word2vec

n-gram Language Model with neural networks

Recall

- The aim of a *n*-gram Language Model is to calculate:

$$P\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(t-n+2)}\right)$$

- We can use a feed-forward neural network to calculate this probability
  - Smooth probability estimation
  - Benefitting from the semantic space of word embeddings

Basic idea from: Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, *3*(Feb), 1137-1155.

Feed-Forward Language Model – training data

- Creating training data for a 4-gram Feed-Forward Language Model in the form of $(\text{context}, \text{next word})$, namely $(x^{(t-2)}x^{(t-1)}x^{(t)}, x^{(t+1)})$:
- Text corpus:

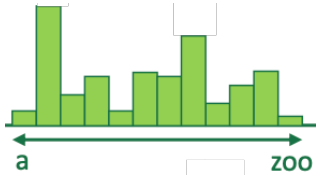
```
a fluffy cat sunbathes on the bank of river …
```

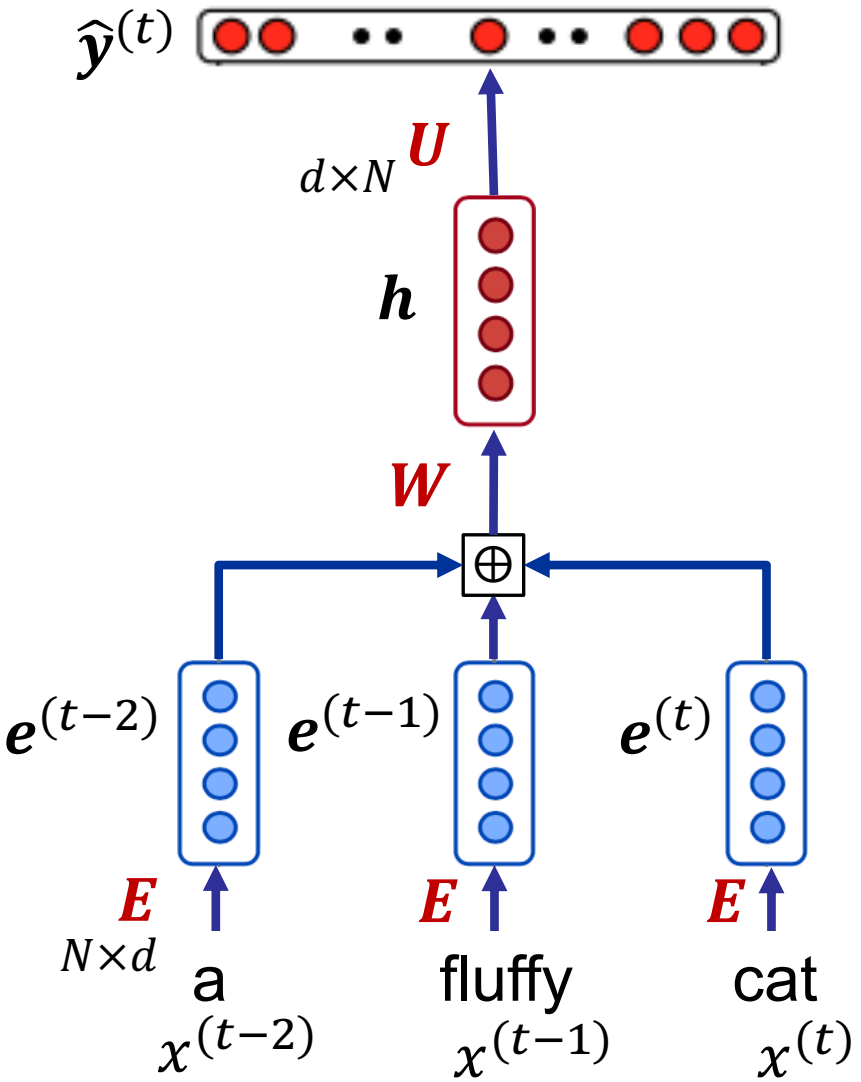
- Resulting training data items:

```
(<bos> <bos> <bos>, a)
(<bos> <bos> a , fluffy)
(<bos> a fluffy , cat)
(a fluffy cat , sunbathes)
(fluffy cat sunbathes , on)
(cat sunbathes on , the)
(sunbathes on the , bank)
…
```

Feed-Forward Language Model

$P(\textit{sunbathes}|\textit{a fluffy cat})$





Training data item:
(a fluffy cat , sunbathes)

Feed-Forward Language Model – formulation

- Encoder
  - One-hot vector of word $x^{(t)} \rightarrow \boldsymbol{x}^{(t)} \in \mathbb{R}^N$
  - Fetching word embedding $\rightarrow \boldsymbol{e}^{(t)} = \boldsymbol{x}^{(t)}\boldsymbol{E}$
- Concatenation of word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(t-2)}, \boldsymbol{e}^{(t-1)}, \boldsymbol{e}^{(t)}]$$

- Hidden layer

$$\boldsymbol{h} = \tanh(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b})$$

- Decoder
  - Predicted probability distribution:

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}) \in \mathbb{R}^N$$

  - Probability of any next word v at step t:

$$P\left(v \middle| x^{(t)}, \dots, x^{(t-n+2)}\right) = \hat{y}_v^{(t)}$$

Feed-Forward Language Model – parameters

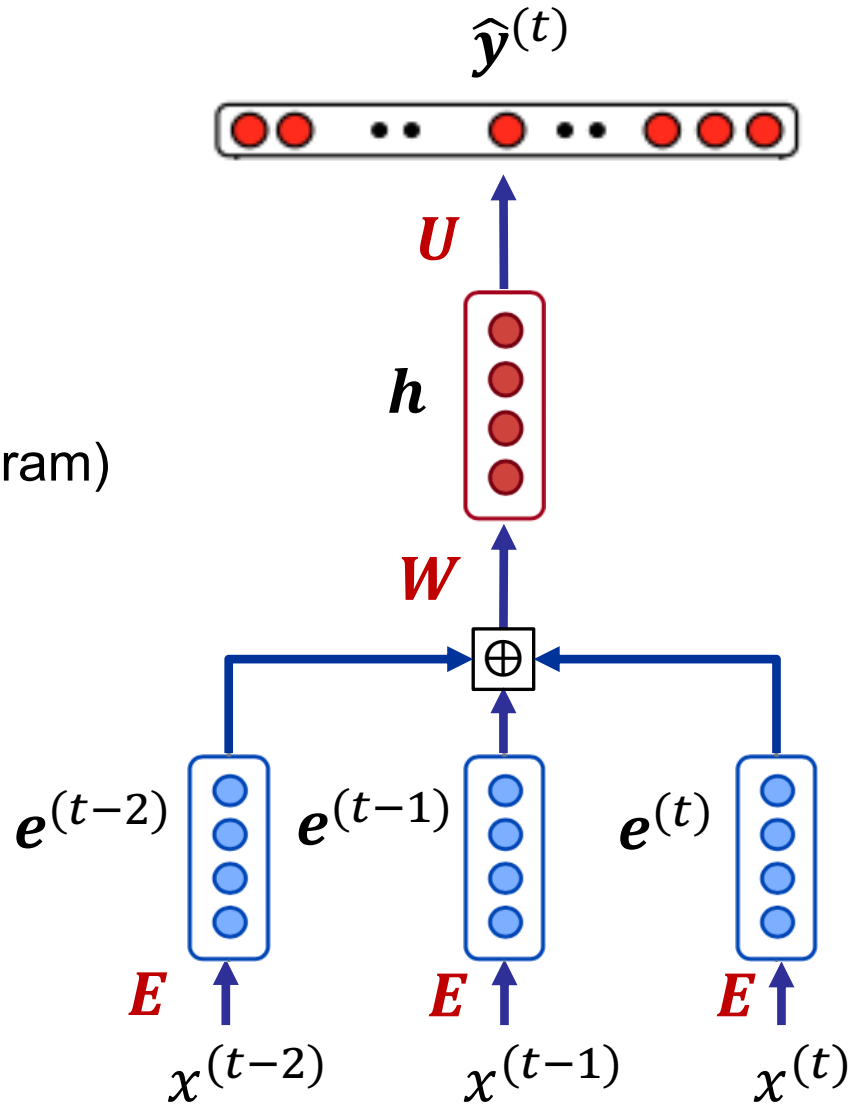
- $\boldsymbol{E} \rightarrow N \times h$
- $\boldsymbol{W} \rightarrow (n \times h) \times h$
- $\boldsymbol{U} \rightarrow h \times N$

h embeddings dimension

n number of preceding words (*n*-gram)

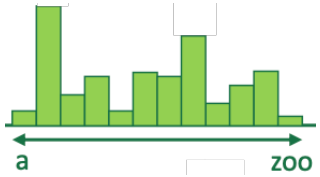
$\boldsymbol{E}$ is called encoder embedding or simply word embedding

$\boldsymbol{U}$ is called decoder embedding or output projection



Feed-Forward Language Model – loss

$P(\textit{sunbathes}|\textit{a fluffy cat})$



$$\mathcal{L} = -\log \hat{y}^{(t)}_{x^{(t+1)}}$$

$\hat{\boldsymbol{y}}^{(t)}$

$d \times N$ $\boldsymbol{U}$

$\boldsymbol{h}$

$\boldsymbol{W}$

$\boldsymbol{e}^{(t-2)}$ $\boldsymbol{e}^{(t-1)}$ $\boldsymbol{e}^{(t)}$

$\boldsymbol{E}$ $\boldsymbol{E}$ $\boldsymbol{E}$

$N \times d$

a fluffy cat

$x^{(t-2)}$ $x^{(t-1)}$ $x^{(t)}$

Training data item:
(a fluffy cat , sunbathes)

Training a Feed-Forward Language Model

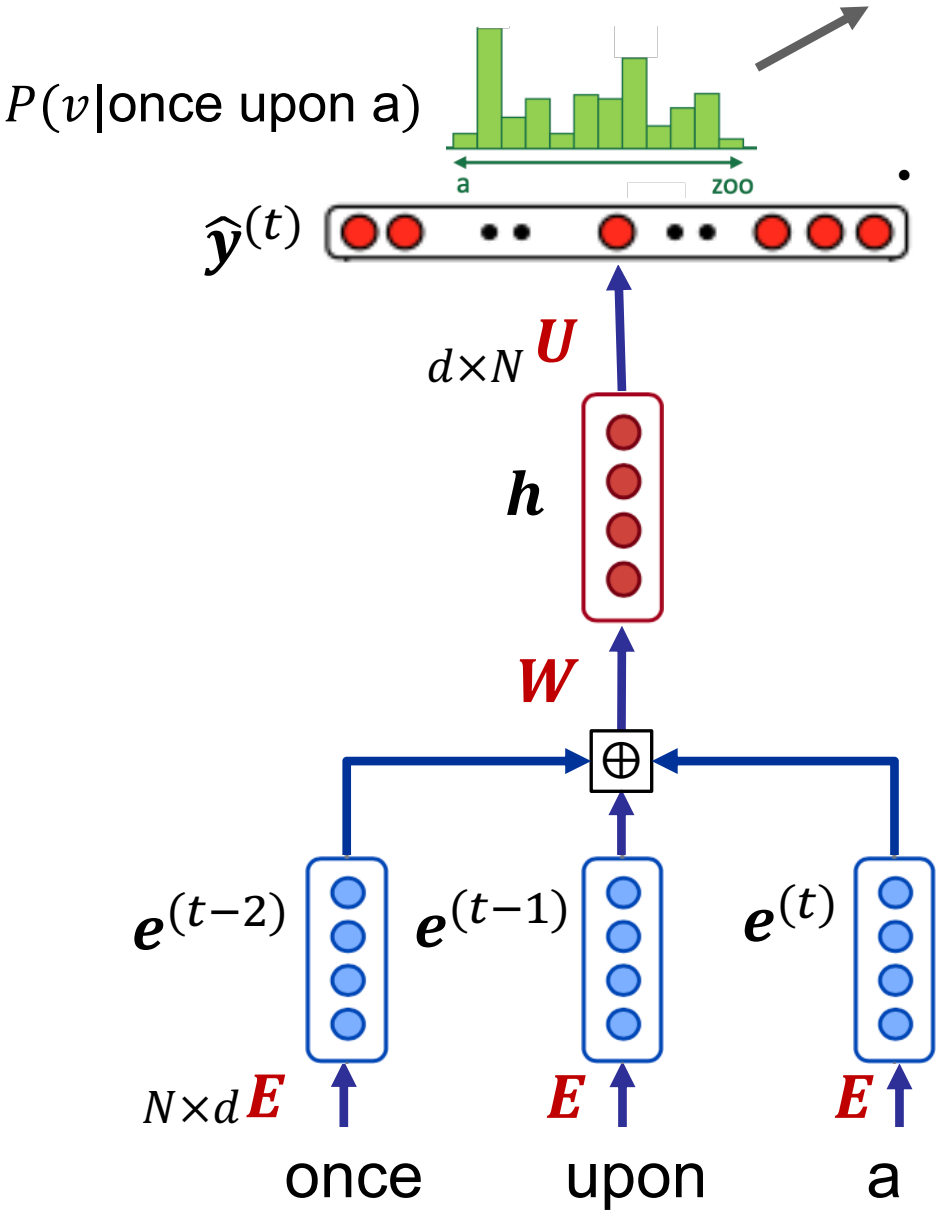
- Start with a large text corpus: $x^{(1)}, \ldots, x^{(T)}$
- For every step t predict the output distribution $\hat{\boldsymbol{y}}^{(t)}$ given *n* previous words
- Loss function at t is Negative Log Likelihood of the predicted probability of the word at $x^{(t+1)}$

$$\mathcal{L}^{(t)} = -\log \hat{y}^{(t)}_{x^{(t+1)}}$$

- Overall loss is the average over all time steps:

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}^{(t)}$$

Feed-Forward Language Model – text generating



- Generate the next word by sampling from the probability distribution (e.g. next word: ***time***)
- Use the generated word and continue generating next words

Generating text with Language Models

- Trained on Obama speeches

Jobs

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. The promise of the men and women who were still going to take out the fact that the American people have fought to make sure that they have to be able to protect our part. It was a chance to stand together to completely look for the commitment to borrow from the American people.

https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Generating text with Language Models

- Trained on Trump speeches

make the country
rich. it was terrible. but saudi arabia, they make a billion dollars a day. i was the king. i was the king. i was the smartest person in yemen, we have to get to business. i have to say, but he was an early starter. and we have to get to business. i have to say, donald, i can't believe it. it's so important. but this is what they're saying, dad, you're going to be really pro, growth, blah, blah. it's disgusting what's disgusting., and it was a 19 set washer and to go to japan. did you hear that character. we are going to have to think about it. but you know, i've been nice to me.

https://github.com/ppramesi/RoboTrumpDNN

Agenda

- *n*-gram language models
- Language modeling with neural networks
- **word2vec**

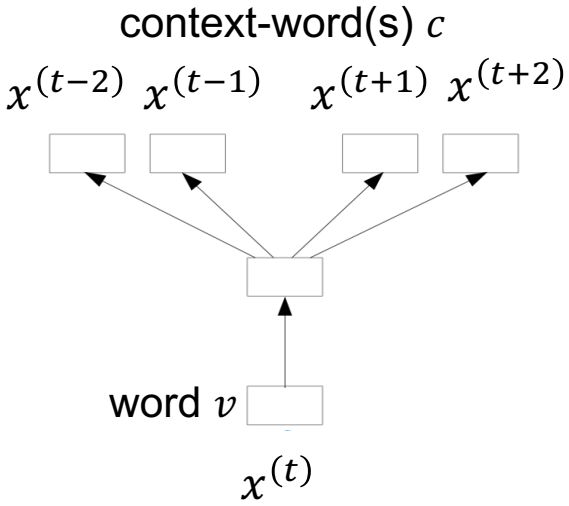
Word embedding with Neural Networks

- Word embeddings are a by-product of neural Language Models

- To create word embedding, we use a neural Language Model …
  - but instead of predicting the next word, …
  - … the model predicts the probability of appearance of a context-word c in a window around the word v

$$P(c|v)$$

- This approach is known as skip-gram



drink

sacred

beer

Tesgüino

ritual

corn

fermented

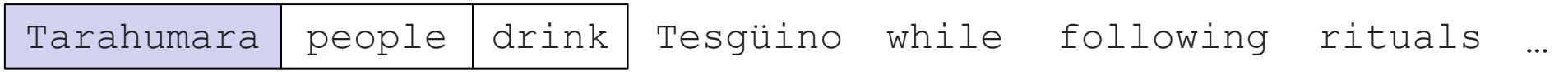
Mexico

Tarahumara people

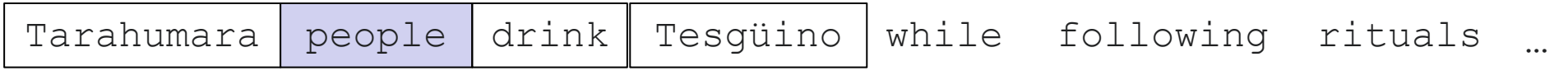
$$P(\text{drink}|\text{Tesgüino}) = ?$$

Training data $\mathcal{D}$

- Creating training data with a window size of 2 in the form of $(\text{word}, \text{context–word})$, namely (v, c) :

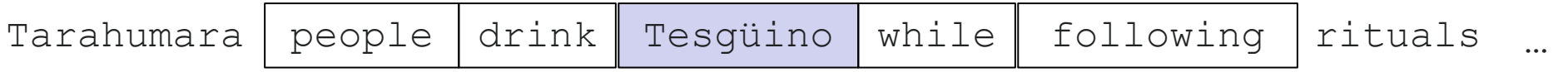


(Tarahumara , people)
(Tarahumara , drink)



(people , Tarahumara)
(people , drink)
(people , Tesgüino)

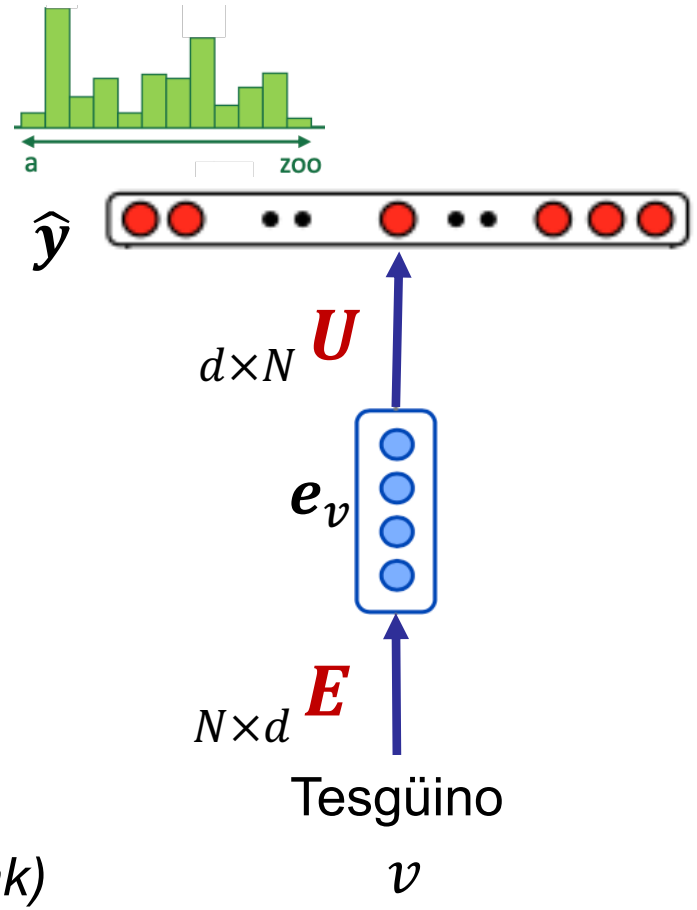
. . .



(Tesgüino , people)
(Tesgüino , drink)
(Tesgüino , while)
(Tesgüino , following)

Neural word embedding – architecture

$P(c|v) = P(\textit{drink}|\textit{Tesgüino})$



Training data: *(Tesgüino , drink)*

Neural word embedding – formulation

- Encoder
  - One-hot vector of word $v \rightarrow \ \boldsymbol{v} \in \mathbb{R}^N$
  - Fetching word embedding $\rightarrow \boldsymbol{e}_v = \boldsymbol{v}\boldsymbol{E}$
- Decoder
  - Predicted probability distribution:

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{e}_v) \in \mathbb{R}^N$$

  - Probability of an arbitrary context-word c given the word v:

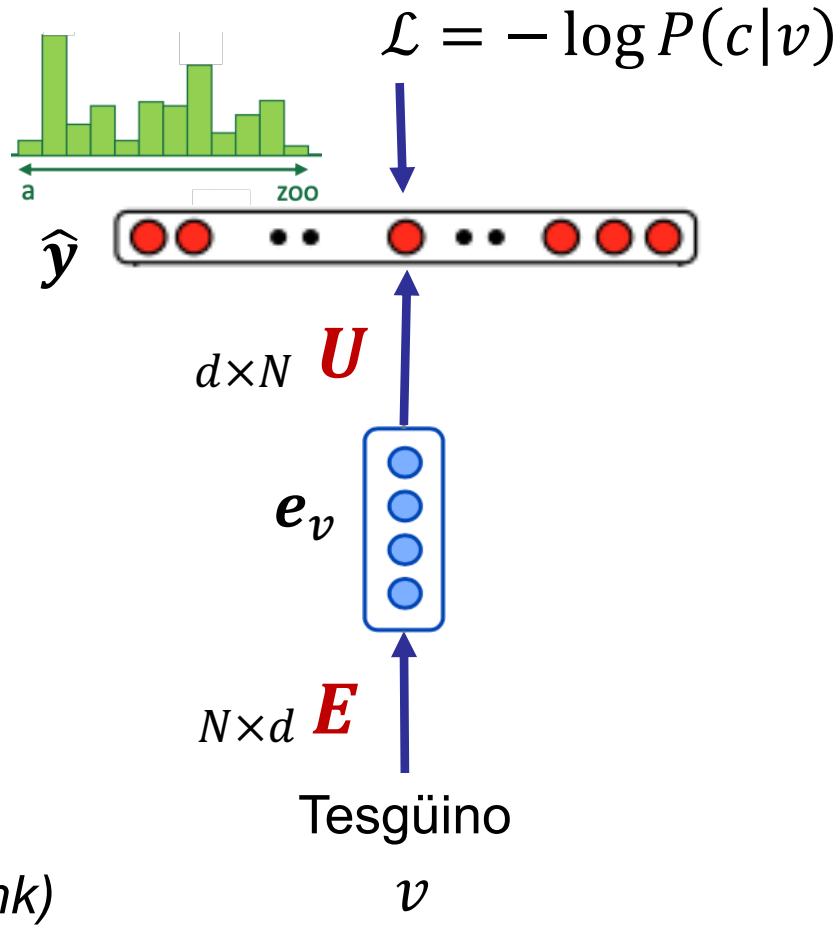
$$P(c|v) \ = \hat{y}_c$$

- Putting all together:

$$P(c|v) \ = \text{softmax}(\boldsymbol{U}\boldsymbol{e}_v)_c = \frac{\exp(\boldsymbol{e}_v \boldsymbol{u}_c)}{\sum_{\tilde{c} \in \mathbb{V}} \exp(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}})}$$

Neural word embedding – loss

$P(c|v) = P(\textit{drink}|\textit{Tesgüino})$



Training data: *(Tesgüino , drink)*

Neural word embedding – all together

- Probability distribution of output words:

$$P(c|v) = \frac{\exp(\boldsymbol{e}_v \boldsymbol{u}_c)}{\sum_{\tilde{c} \in \mathbb{V}} \exp(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}})}$$

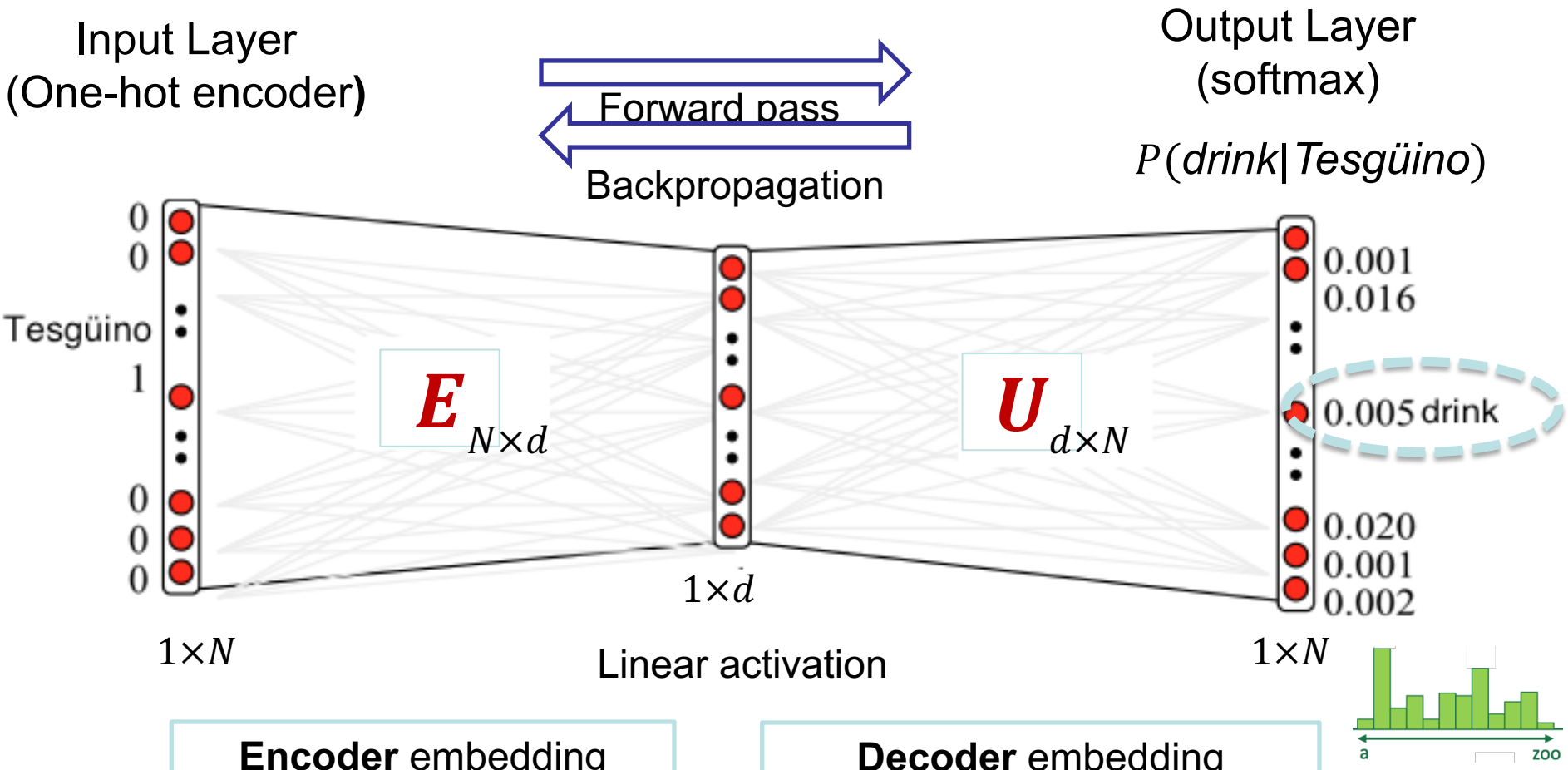
  - In the example: $P(\text{drink}|\text{Tesgüino}) = \frac{\exp(\boldsymbol{e}_{\text{Tesgüino}} \mathbf{u}_{\text{drink}})}{\sum_{\tilde{c} \in \mathbb{V}} \exp(\boldsymbol{e}_{\text{Tesgüino}} \boldsymbol{u}_{\tilde{c}})}$

- Loss is the NLL over all training data:

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

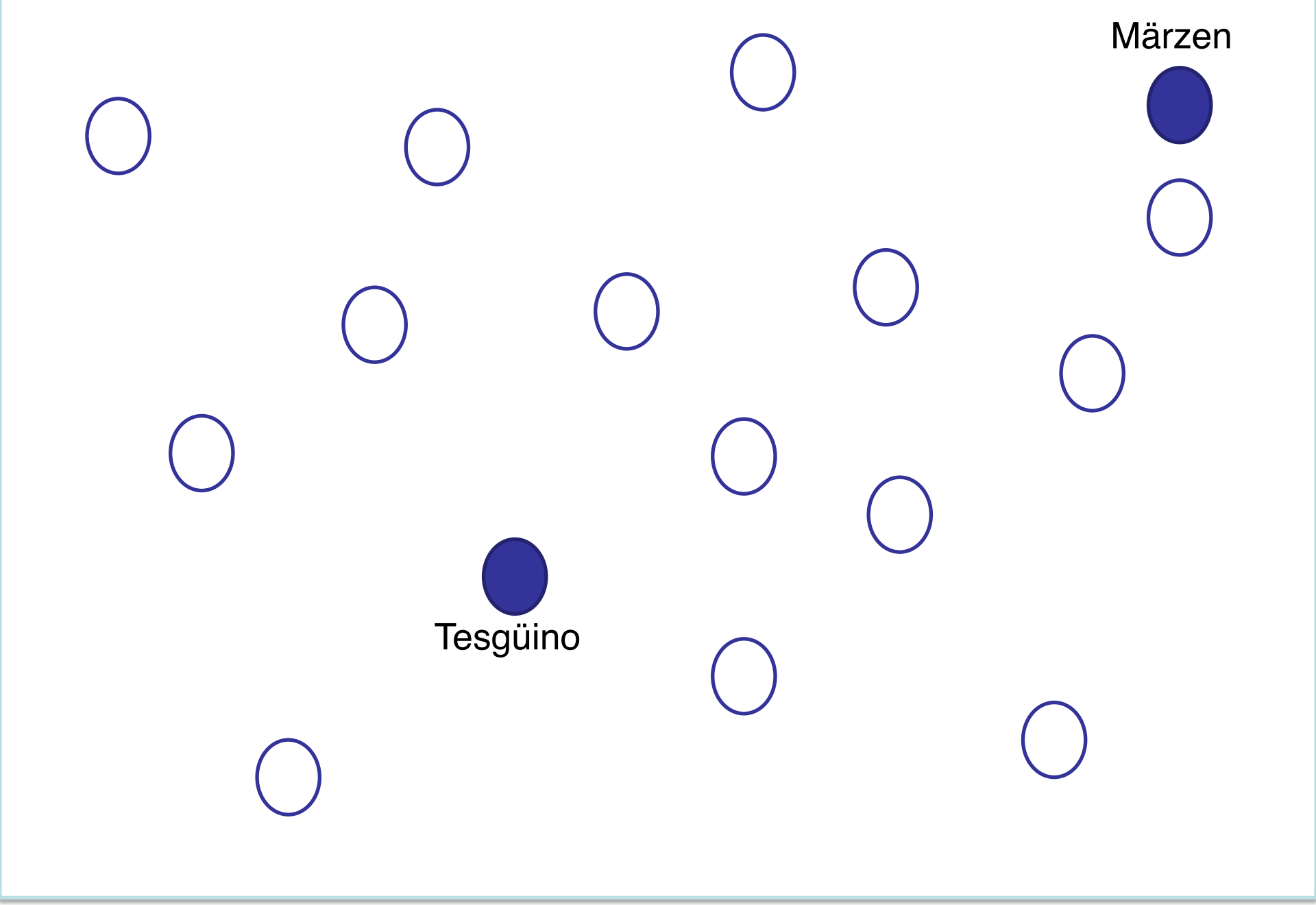
Neural word embedding – architecture (another view!)

Training data: (*Tesgüino*, *drink*)

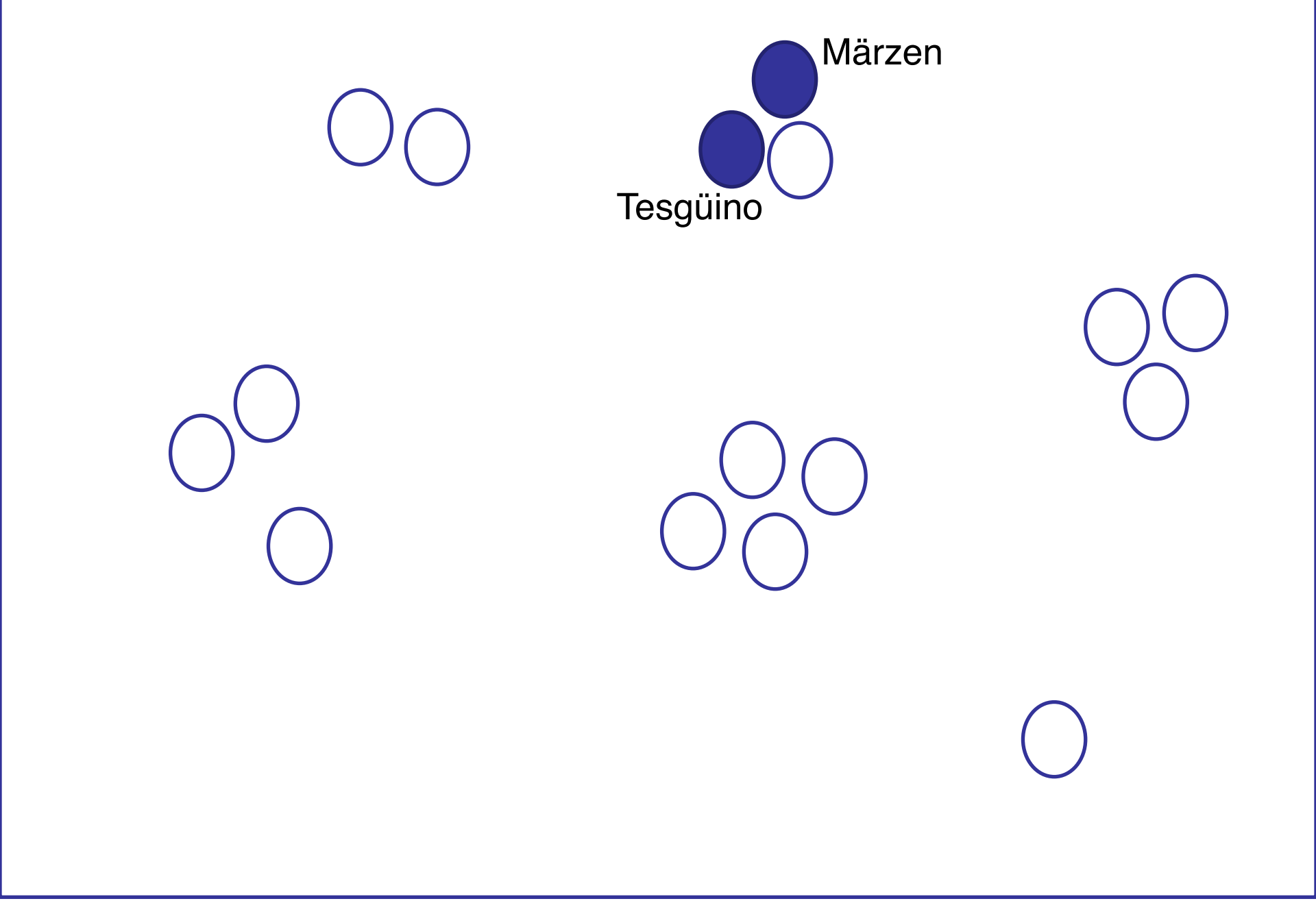


Encoder embedding
word embedding

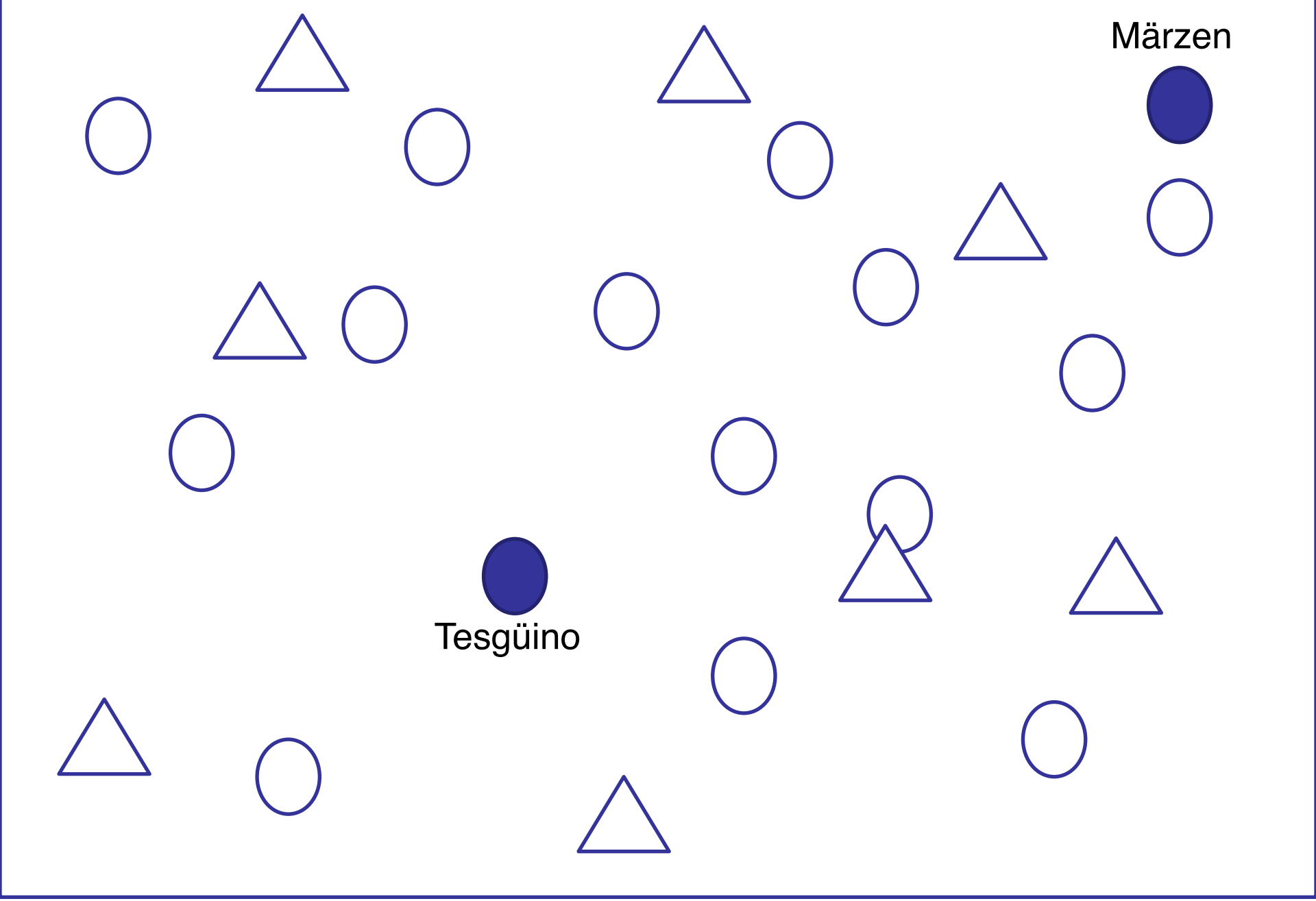
Decoder embedding
context-word embedding



Embedding vector

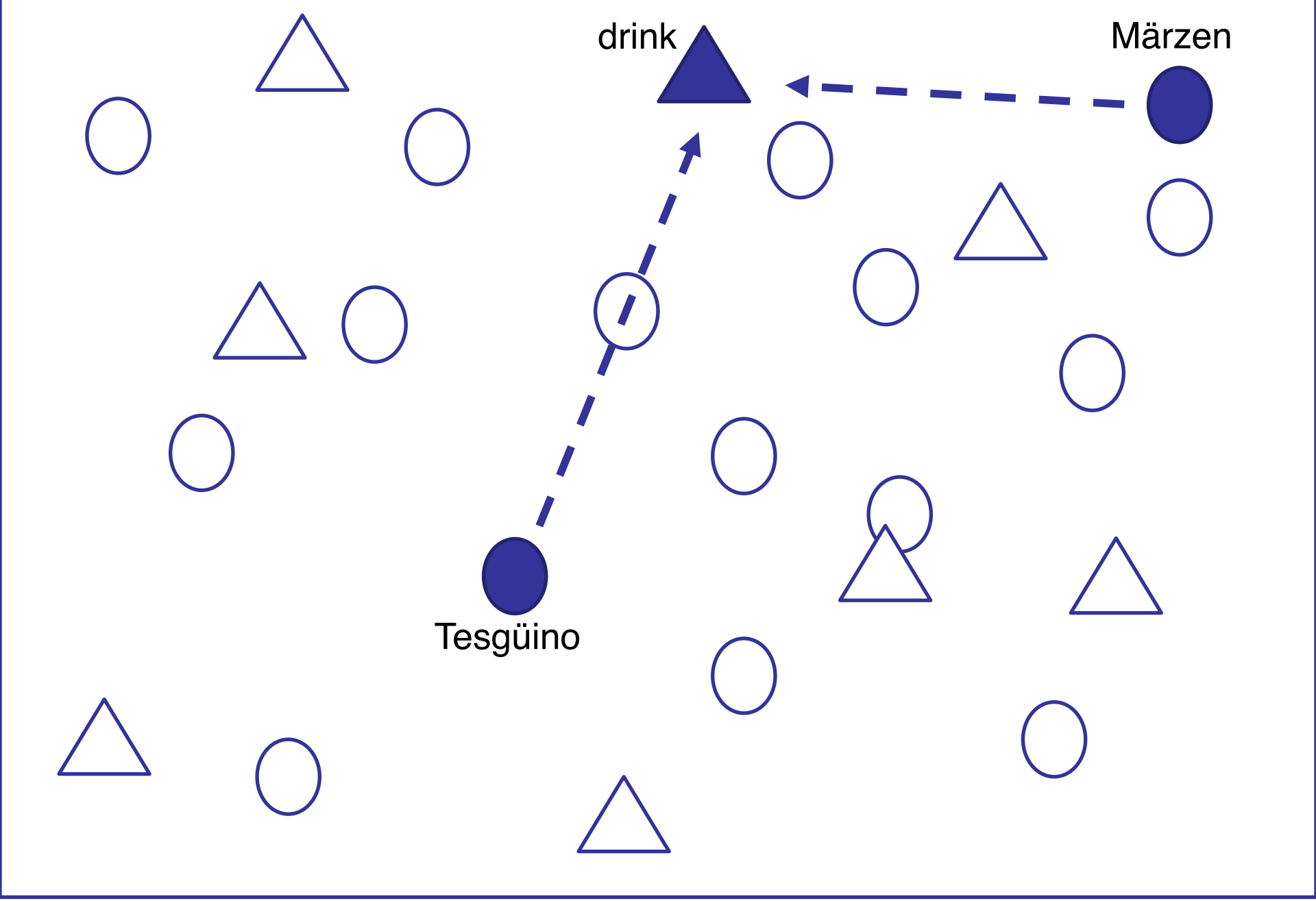


Embedding vector



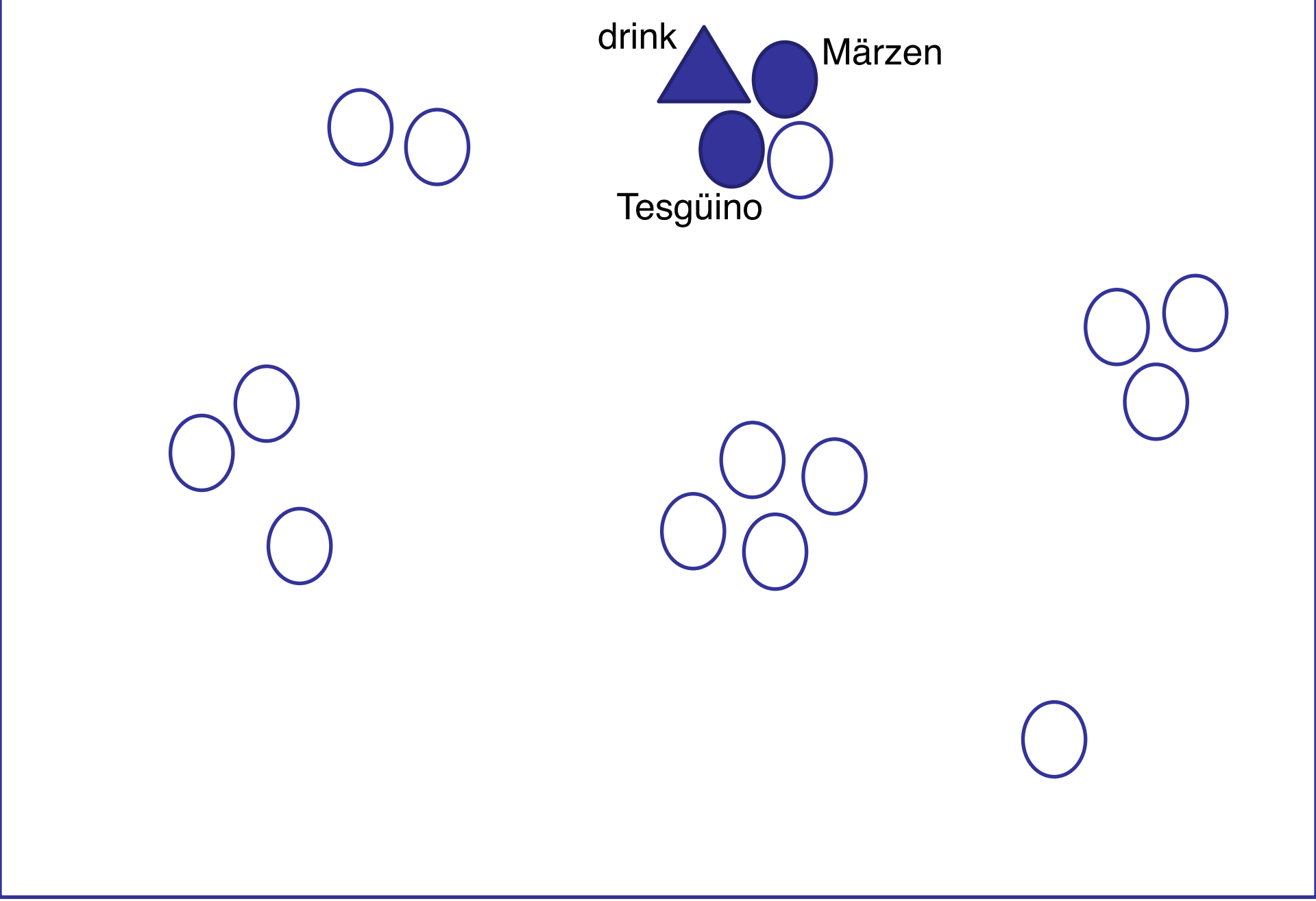
Embedding vector

Decoding vector



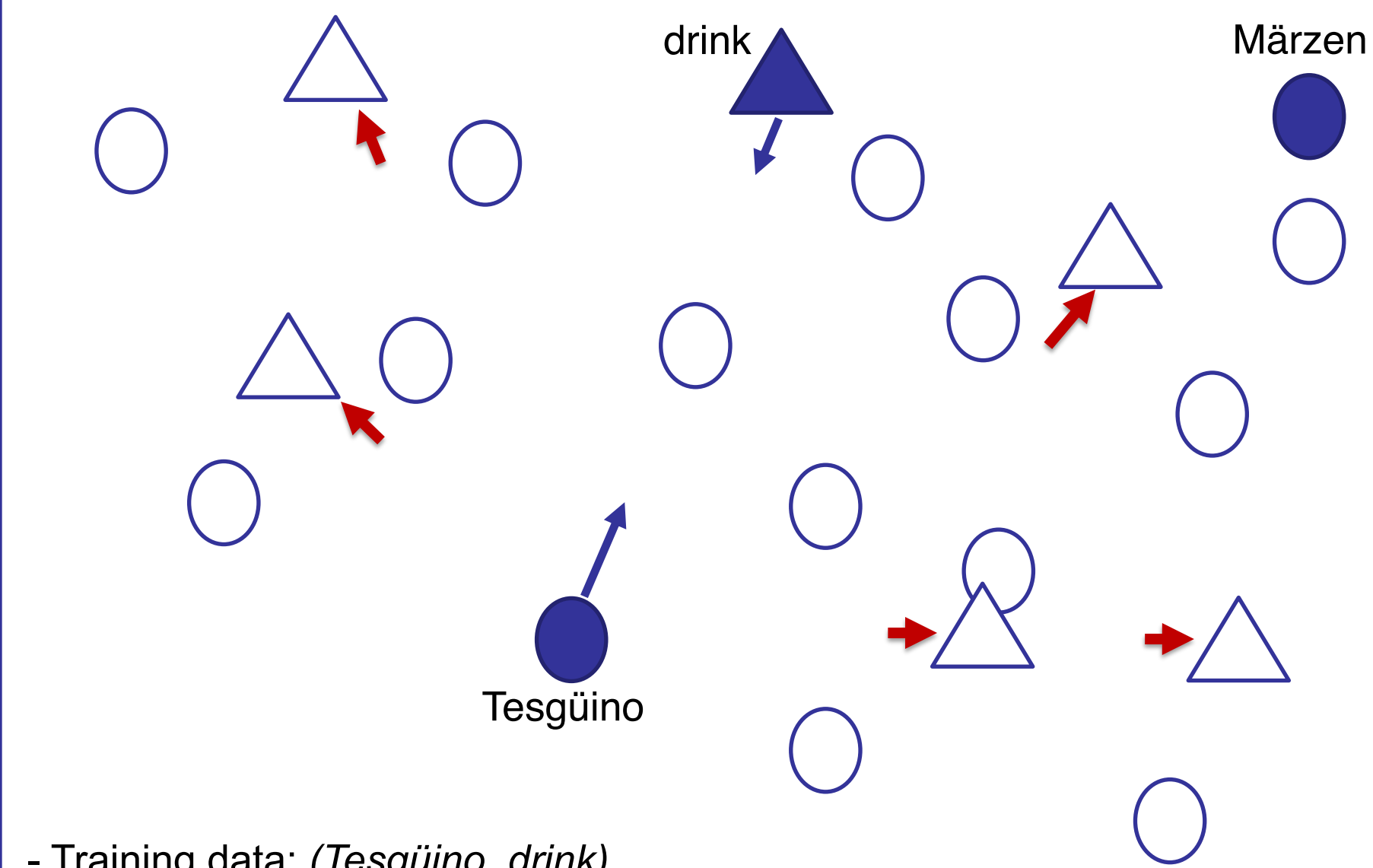
Embedding vector

Decoding vector



Embedding vector

Decoding vector



- Training data: *(Tesgüino, drink)*
- Update vectors to maximize $P(\textit{drink}|\textit{Tesgüino})$

Embedding vector

Decoding vector

Neural word embedding – NLL + softmax

$$P(c|v) = \frac{\exp(\boldsymbol{e}_v \boldsymbol{u}_c)}{\sum_{\tilde{c} \in \mathbb{V}} \exp(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}})}$$

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

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

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

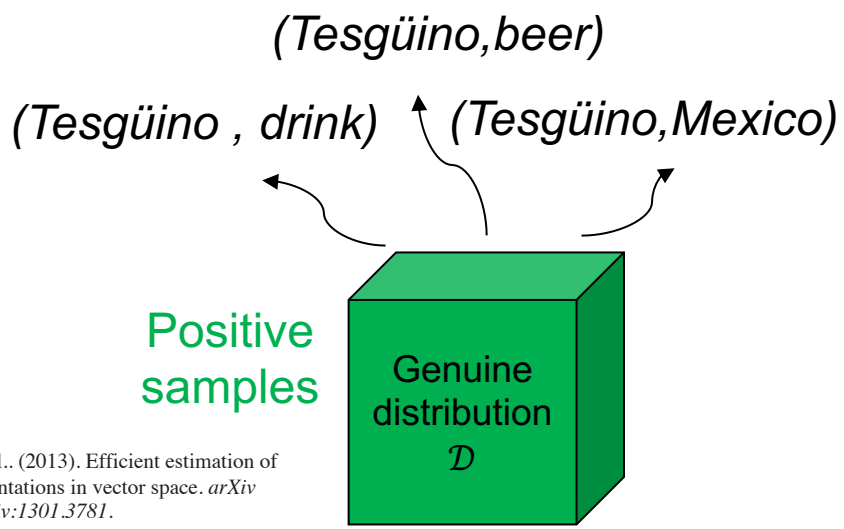
calculating this normalization term is expensive!

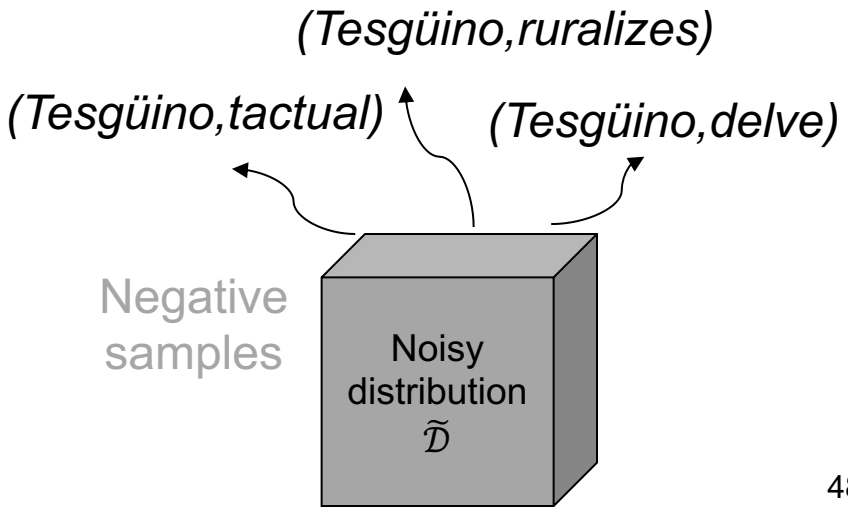
word2vec Skip-Gram with Negative Sampling

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

Central idea of Negative Sampling:

- Consider two data distributions that generate [word, context-word] pairs:
  1. The genuine one that comes from the training data $\rightarrow \mathcal{D}$
  2. The noisy one that generates (almost) random pairs $\rightarrow \widetilde{\mathcal{D}}$
- Aim: given a pair, the model should be able to distinguish whether it comes from the genuine distribution (binary classification)





Mikolov et al.. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Negative Sampling

- Instead of $P(c|v)$ as in neural word embedding, Negative Sampling calculates …
- $P(y = 1|v, c) \rightarrow$ probability that the pair (v, c) comes from the genuine data distribution

- Positive sample: pair (v, c) that comes from the genuine data distribution $\mathcal{D}$ (training data)

- Negative sample: pair $(v, \tilde{c})$ drawn from the noisy distribution $\widetilde{\mathcal{D}}$
  - $\widetilde{\mathcal{D}}$ generates pairs almost randomly!
    - Why random pairs can be considered as negative samples?
  - $\widetilde{\mathcal{D}}$ in word2vec is a *smoothed* unigram distribution of words in corpus
    - Implementation details: In word2vec, $\widetilde{\mathcal{D}}$ is smoothed by raising unigram counts to the power of $\alpha = 0.75$

Negative Sampling

- Negative Sampling loss aims to
  - increase the probability of positive samples, $P(y = 1|v, c)$ and …
  - decrease the probabilities of k negative samples, $P(y = 1|v, \tilde{c})$
    - k is usually between 2 to 20

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

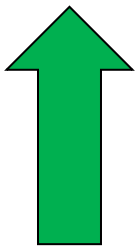
$$P(y = 1|v, c) = \sigma(\boldsymbol{e}_v \boldsymbol{u}_c)$$

word2vec – loss

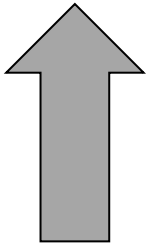
- Negative Sampling loss
  - increases the probability for the positive sample (v, c)
  - decreases the probability for the k negative samples $(v, \tilde{c})$
- Loss function:

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

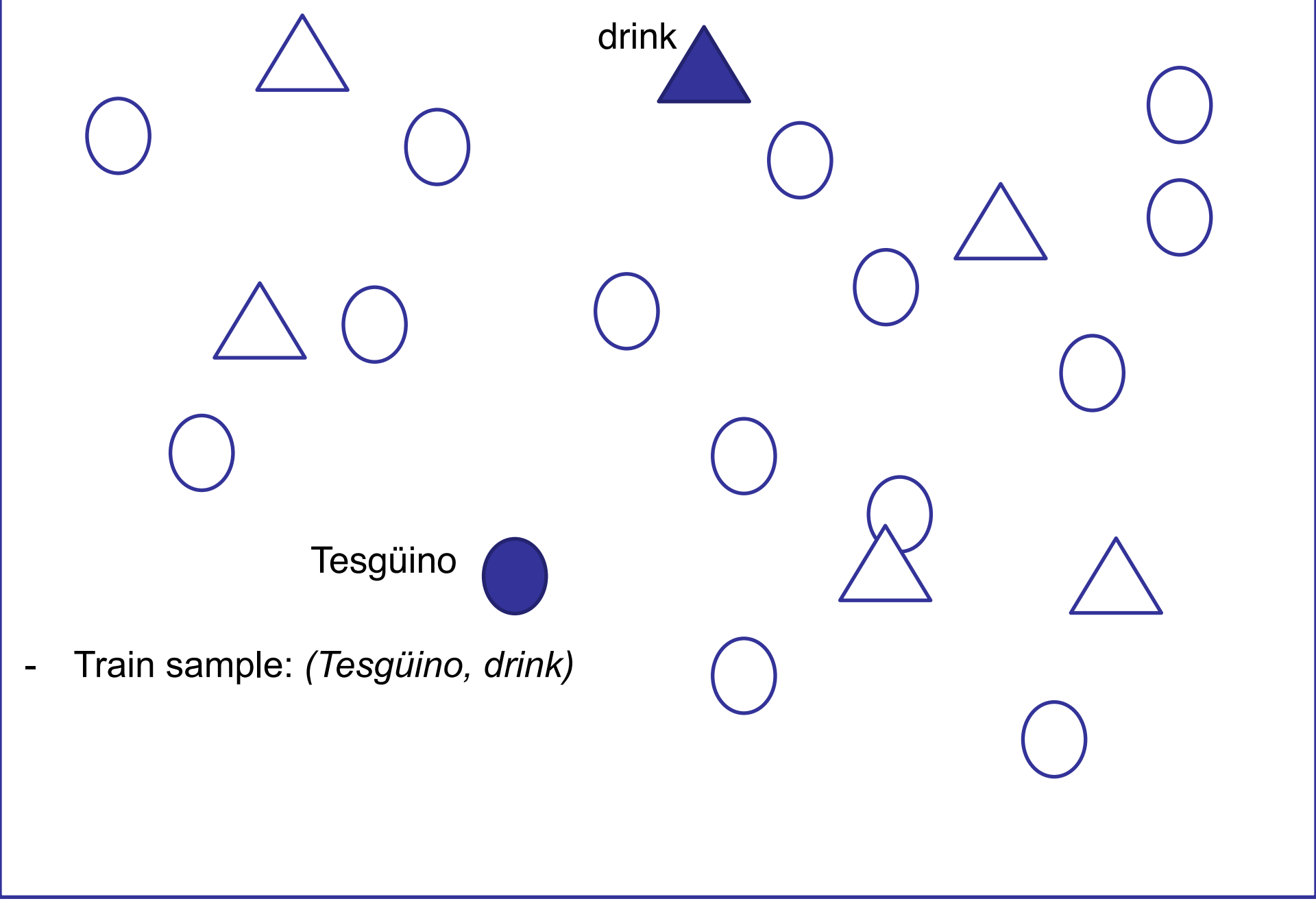
$$\mathcal{L} = -\mathbb{E}_{(v,c)\sim\mathcal{D}}\left[\log \sigma(\boldsymbol{e}_v \boldsymbol{u}_c) - \sum_{\substack{\tilde{c}\sim\tilde{\mathcal{D}} \\ k \text{ times}}} \log \sigma(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}})\right]$$



positive samples



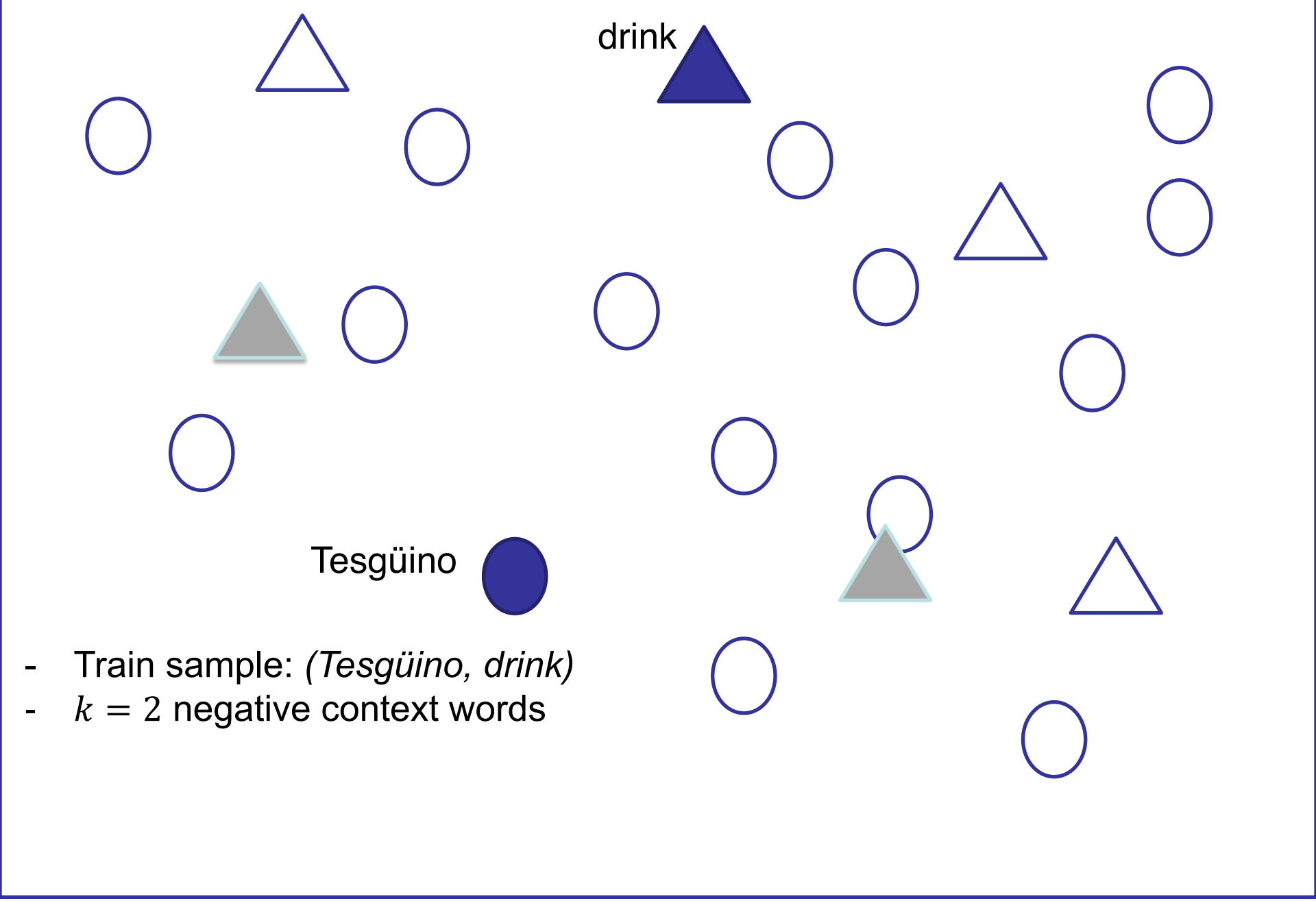
negative samples



- Train sample: *(Tesgüino, drink)*

Embedding vector

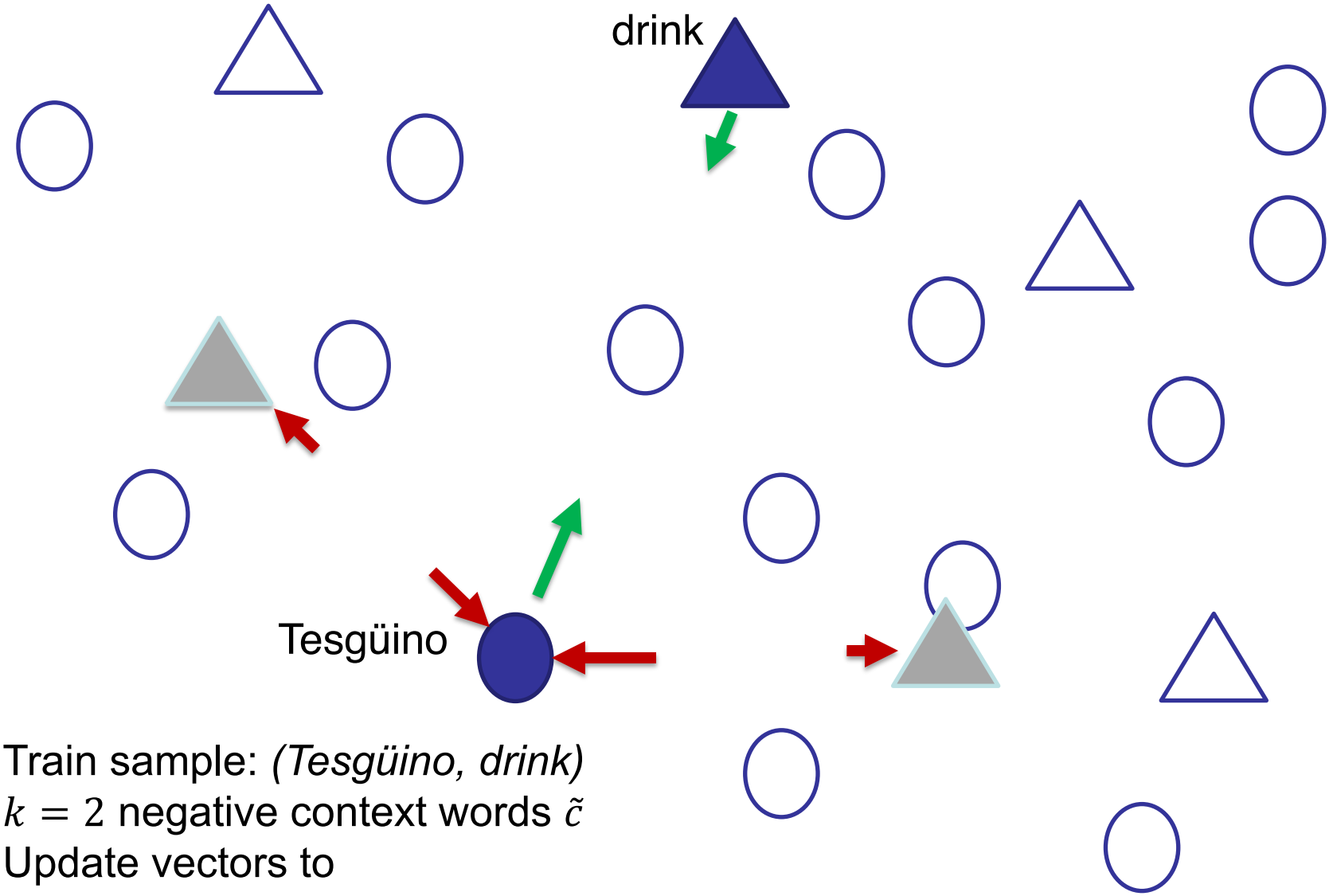
Decoding vector



- Train sample: *(Tesgüino, drink)*
- $k = 2$ negative context words

Embedding vector

Decoding vector



- Train sample: *(Tesgüino, drink)*
- $k = 2$ negative context words $\tilde{c}$
- Update vectors to
  - Increase $P(y = 1|\text{Tesgüino}, \text{drink})$
  - Decrease $P(y = 1|\text{Tesgüino}, \tilde{c})$

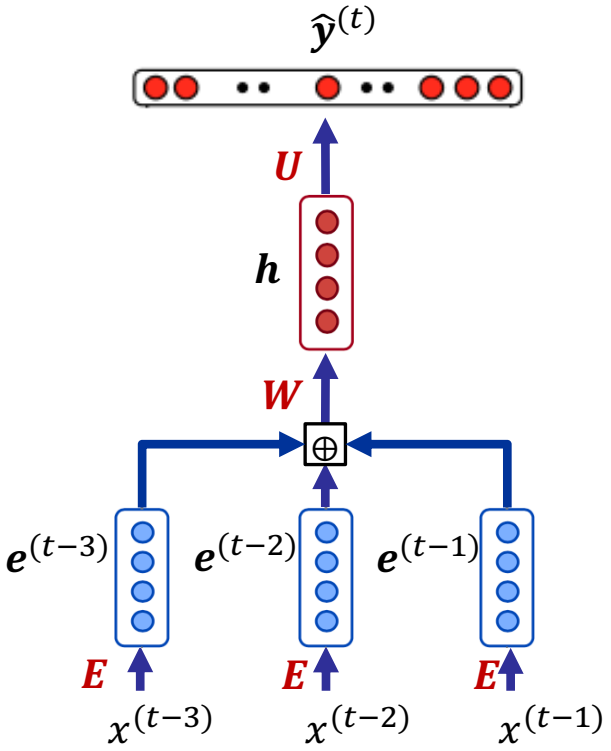
Embedding vector Decoding vector

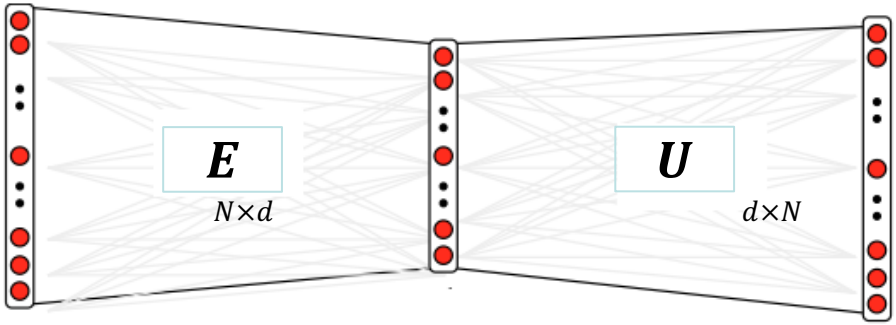
Negative Sampling – final words!

- Negative Sampling turns the problem from multi-class classification to binary classification
- Softmax is a good choice for training Language Models, namely to estimate $P(v|\text{context})$
- Negative Sampling is shown to be effective for training good embeddings
- Negative Sampling is a biased approximation of softmax
  - Noisy Contrastive Estimation (the parent of Negative Sampling) is an unbiased approximation of softmax

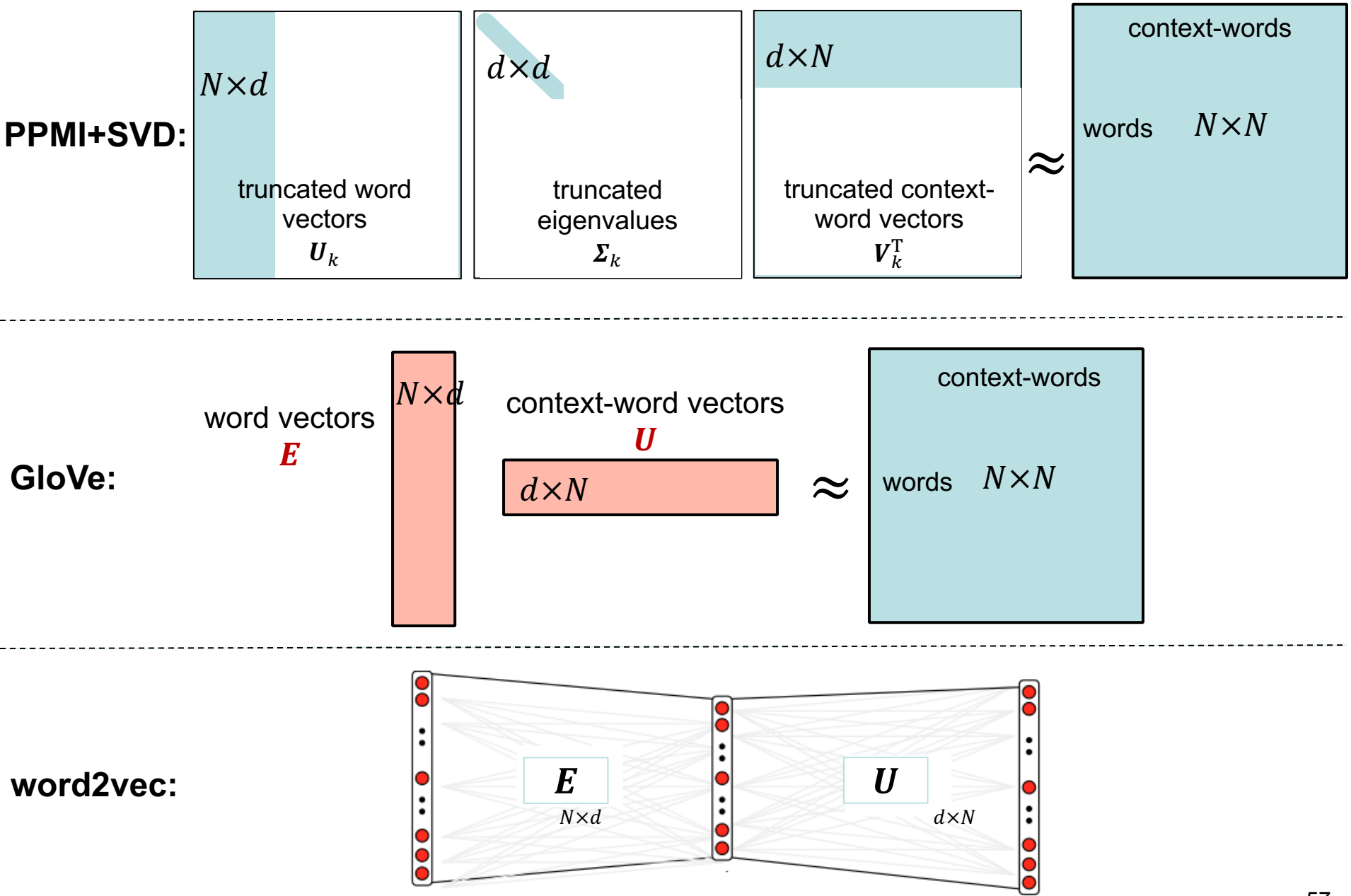
Summary

- *n*-gram Language Models count co-occurrences
- Neural *n*-gram Language Models predict co-occurrence probabilities
- word2vec creates word embedding by …
  - following a skip-gram Language Modeling objective and …
  - exploiting Negative Sampling loss





Word embedding models – in one frame!



Levy, et al. (2015). Improving distributional similarity with lessons learned from word embeddings. *TACL*