Winter semester 2021/22

344.175 VL: Natural Language Processing
Neural Language Models & Word Embeddings

Navid Rekab-saz

navid.rekabsaz@jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

Agenda

- Neural *n*-gram Language Model
- Neural skip-gram Language Model
- word2vec

Agenda

- **Neural *n*-gram Language Model**
- Neural skip-gram Language Model
- word2vec

N-gram language modeling with neural networks

Recall

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

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

- We can use a feed forward neural network to estimate this probability
- Immediate benefits:
  - Smooth probability estimation
  - Exploiting the semantic space of word embeddings (probably better generalization)

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.

Neural *n*-gram LM – preparing training data

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

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

- Training data items would be:

```
(<bos> <bos> <bos>, a)
(<bos> <bos> a, fluffy)
(<bos> a fluffy, cat)
```

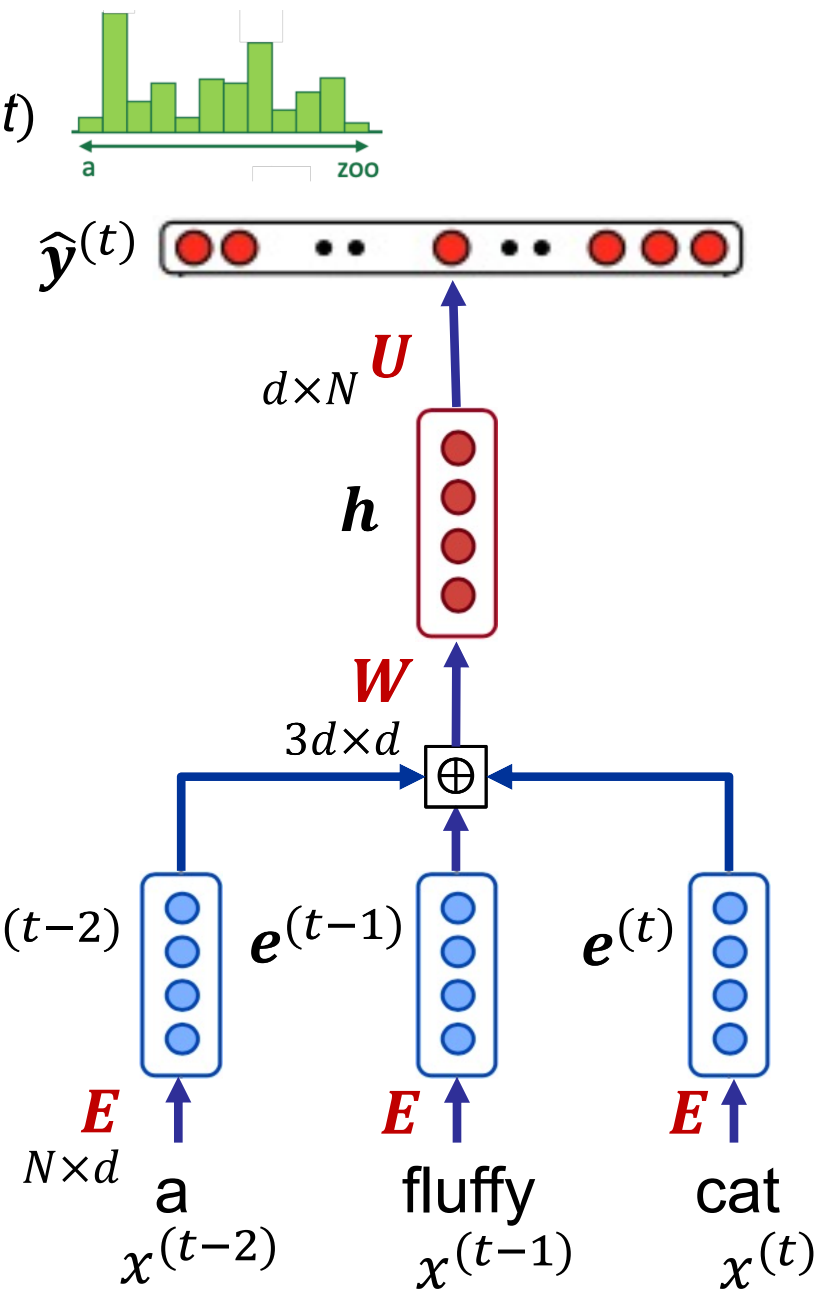
`(a fluffy cat, sunbathes)`

```
(fluffy cat sunbathes, on)
(cat sunbathes on, the)
(sunbathes on the, bank)
```

…

Neural *n*-gram Language Model – architecture

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



A data item in training data:

`(a fluffy cat, sunbathes)`

Formulation

Encoder

- From words to word embeddings:
  - One-hot vector of word $x^{(t)} \rightarrow \boldsymbol{x}^{(t)} \in \mathbb{R}^N$
  - Fetching word embedding $\rightarrow \boldsymbol{e}^{(t)} = \boldsymbol{x}^{(t)}\boldsymbol{E}$
    - In practice, $\boldsymbol{e}^{(t)}$ is achieved by fetching the vector of $x^{(t)}$ from $\boldsymbol{E}$ (no need for $\boldsymbol{x}^{(t)}$)
- 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 probabilities:
  - 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:

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

Model parameters

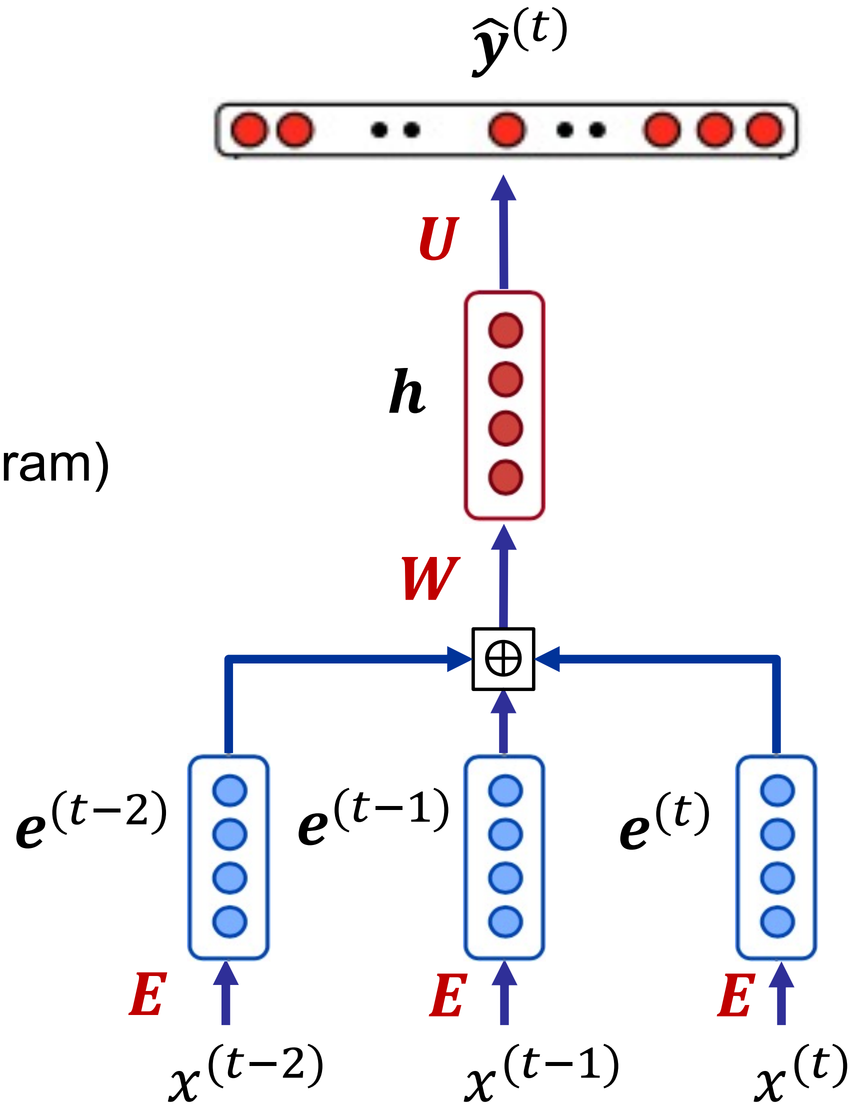
- $\boldsymbol{E} \rightarrow N{\times}h$
- $\boldsymbol{W} \rightarrow (n{\times}h){\times}h$
- $\boldsymbol{U} \rightarrow h{\times}N$

h embeddings dimension

n number of preceding words (n-gram)

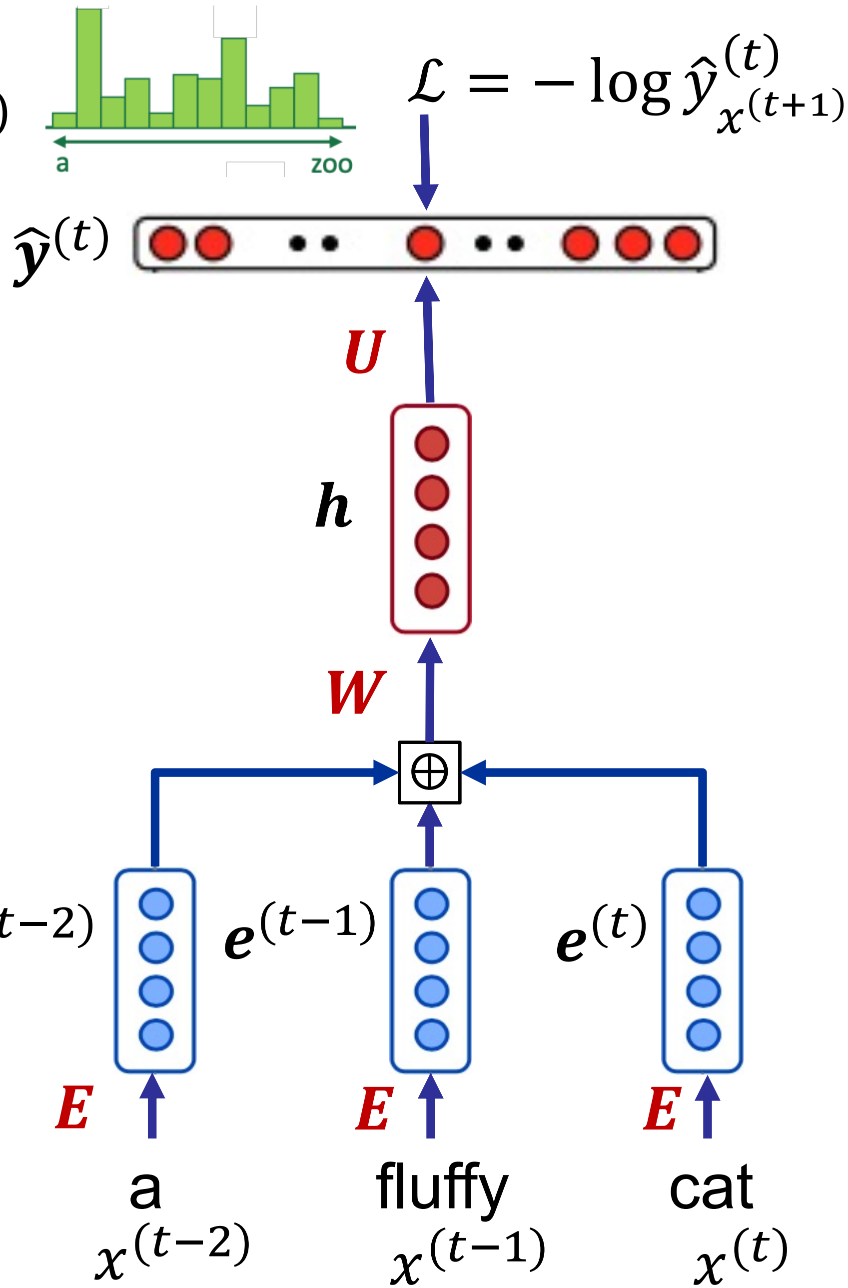
$\boldsymbol{E}$ is called encoder embedding or simply word embedding

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



Loss function

$P(sunbathes|a\ fluffy\ cat)$



A data item in training data:
`(a fluffy cat, sunbathes)`

Training procedure

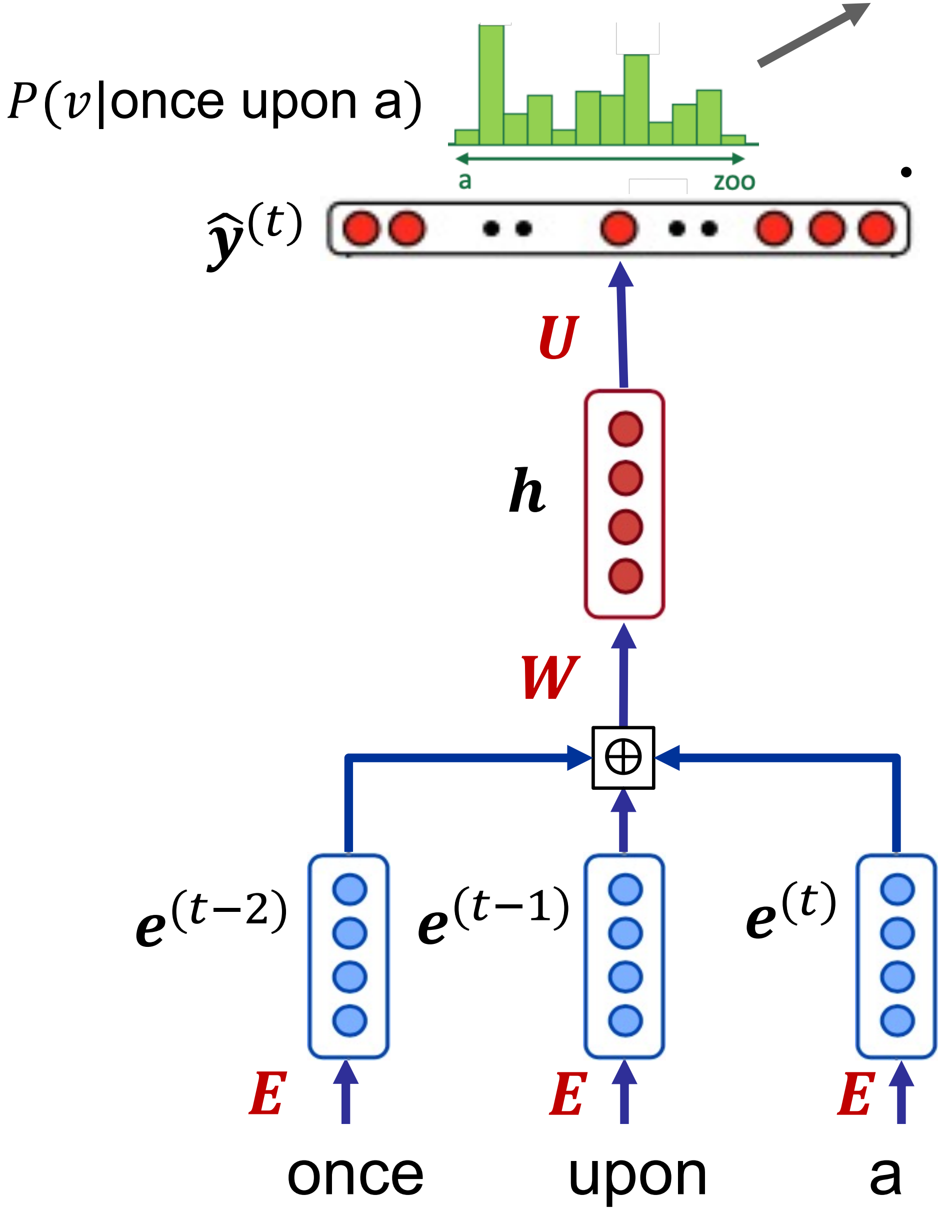
- Start with a large text corpus: $x^{(1)}, \dots, x^{(T)}$
- For every step t predict the output distribution $\hat{\boldsymbol{y}}^{(t)}$ given *n* previous words
- Loss function at t is Negative Log Likelihood of the predicted probability of the word at $x^{(t+1)}$

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

- Overall loss is the average over all time steps:

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

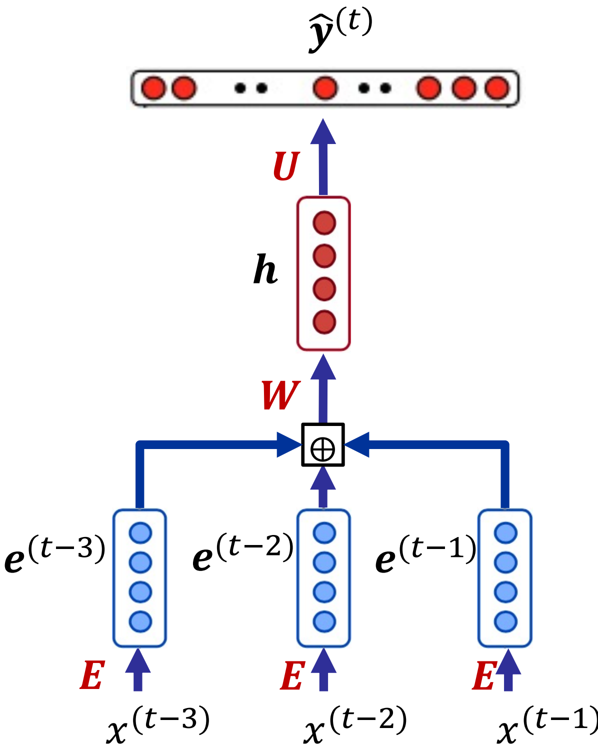
Generating text



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

Neural *n*-gram LMs – summary

- Neural *n*-gram LMs predict co-occurrence probabilities
  - In contrast, *n*-gram LMs count co-occurrences
- Neural *n*-gram LMs provide a smooth probability distribution
  - The predicted probabilities for different words are different
  - Count-based *n*-gram LMs may have the same probability for some words
- At inference time, neural *n*-gram LMs require a forward pass
  - At inference time, count-based *n*-gram LMs only fetch stored counts and estimate probabilities (generally faster)



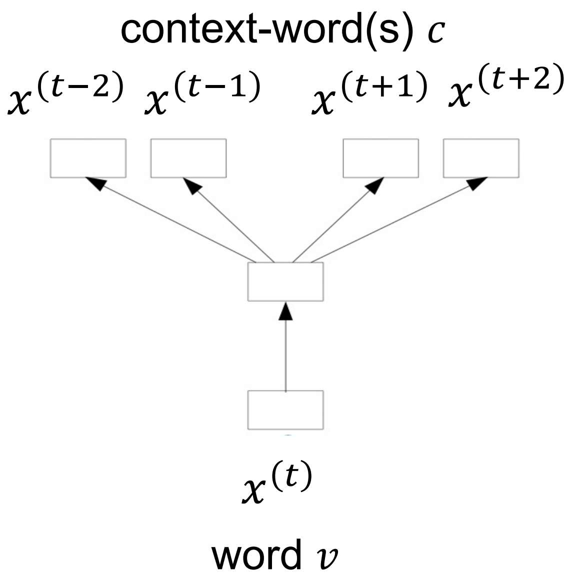
Agenda

- Neural n-gram Language Model
- **Neural skip-gram Language Model**
- word2vec

Neural skip-gram Language Model

- A skip-gram Language Model, …
  - instead of predicting the next word as in usual LMs, …
  - … predicts the probability of appearance of a context-word c in a window surrounding the word v

$$P(c|v)$$



$P\left(x^{(t-2)}\middle|x^{(t)}\right) =?$

$P\left(x^{(t-1)}\middle|x^{(t)}\right) =?$

$P\left(x^{(t+1)}\middle|x^{(t)}\right) =?$

$P\left(x^{(t+2)}\middle|x^{(t)}\right) =?$

drink

sacred

beer

Tesgüino

ritual

corn

fermented

Tarahumara people

Mexico

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

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 drink Tesgüino while following rituals …
```

```
(Tarahumara, people)
(Tarahumara, drink)
```

```
Tarahumara people drink Tesgüino while following rituals …
```

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

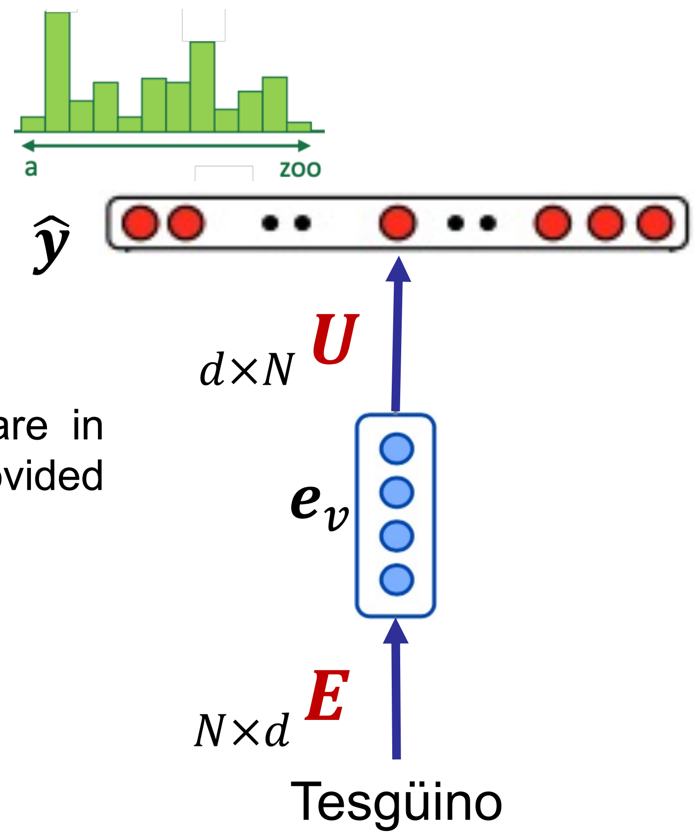
...

```
Tarahumara people drink Tesgüino while following rituals …
```

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

Neural word embeddings from neural skip-gram Language Model

$P(c|v) = P(drink|Tesgüino)$



The model's parameters $\boldsymbol{E}$ and $\boldsymbol{U}$ are in fact two sets of word embeddings, provided as the by-products of the model:

- $\boldsymbol{E} \rightarrow$ Encoder word embedding
- $\boldsymbol{U} \rightarrow$ Decoder word embedding

Training data: `(Tesgüino, drink)`

Formulation

Encoder

- From words to word embeddings:
  - One-hot vector of word $v \rightarrow \boldsymbol{v} \in \mathbb{R}^N$
  - Encoder word embedding $\rightarrow \boldsymbol{e}_v = \boldsymbol{v}\boldsymbol{E}$
    - In practice, $\boldsymbol{e}_v$ is achieved by fetching the embedding of v from $\boldsymbol{E}$ (no need for $\boldsymbol{v}$)

Decoder

- Predicted probabilities:
  - 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:

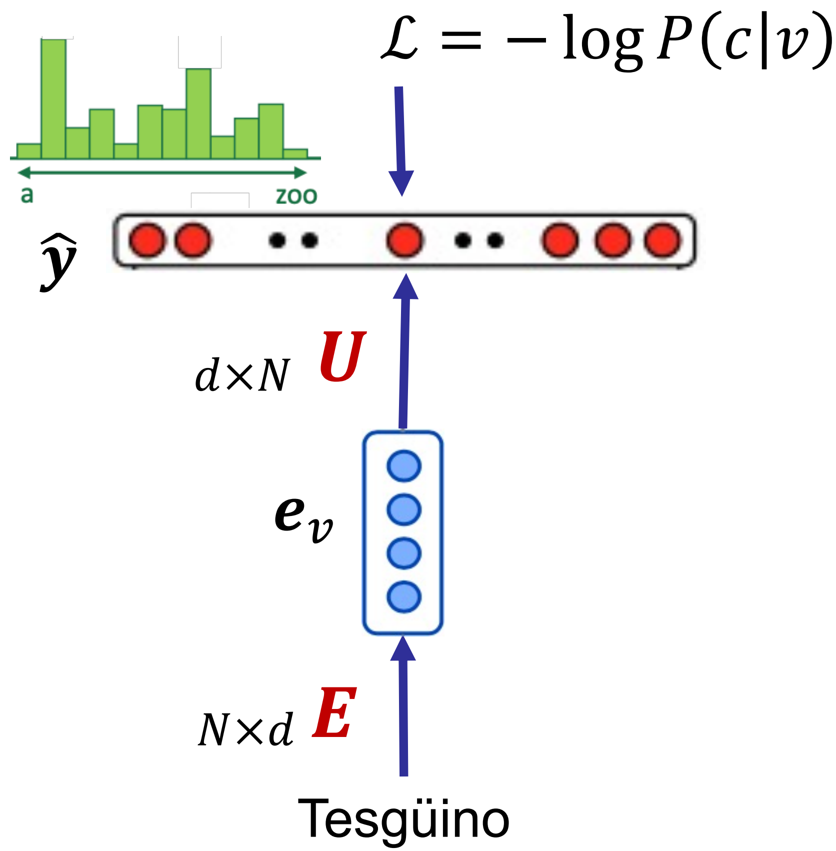
$$P(c|v) = \hat{y}_c$$

Putting all together:

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

Loss function

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



Training data: `(Tesgüino, drink)` v

Skip-gram Language Model – all together

- Probability distribution of output words:

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

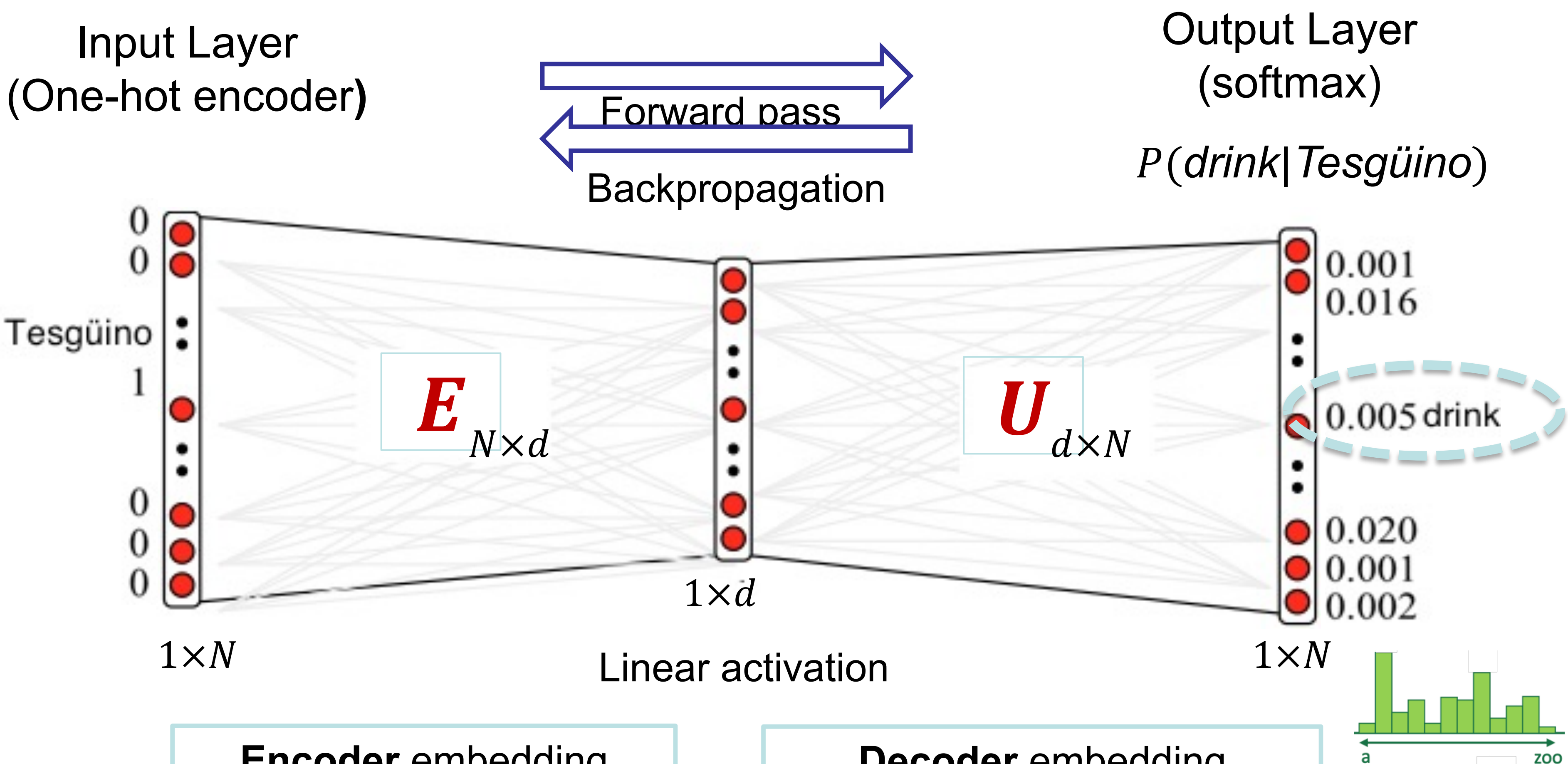
  - In the example: $P(\text{drink}|\text{Tesgüino}) = \frac{\exp(\boldsymbol{e}_{\text{Tesgüino}} \boldsymbol{u}_{\text{drink}})}{\sum_{\tilde{c} \in \mathbb{V}} \exp(\boldsymbol{e}_{\text{Tesgüino}} \boldsymbol{u}_{\tilde{c}})}$

- Loss is the NLL over all training data:

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

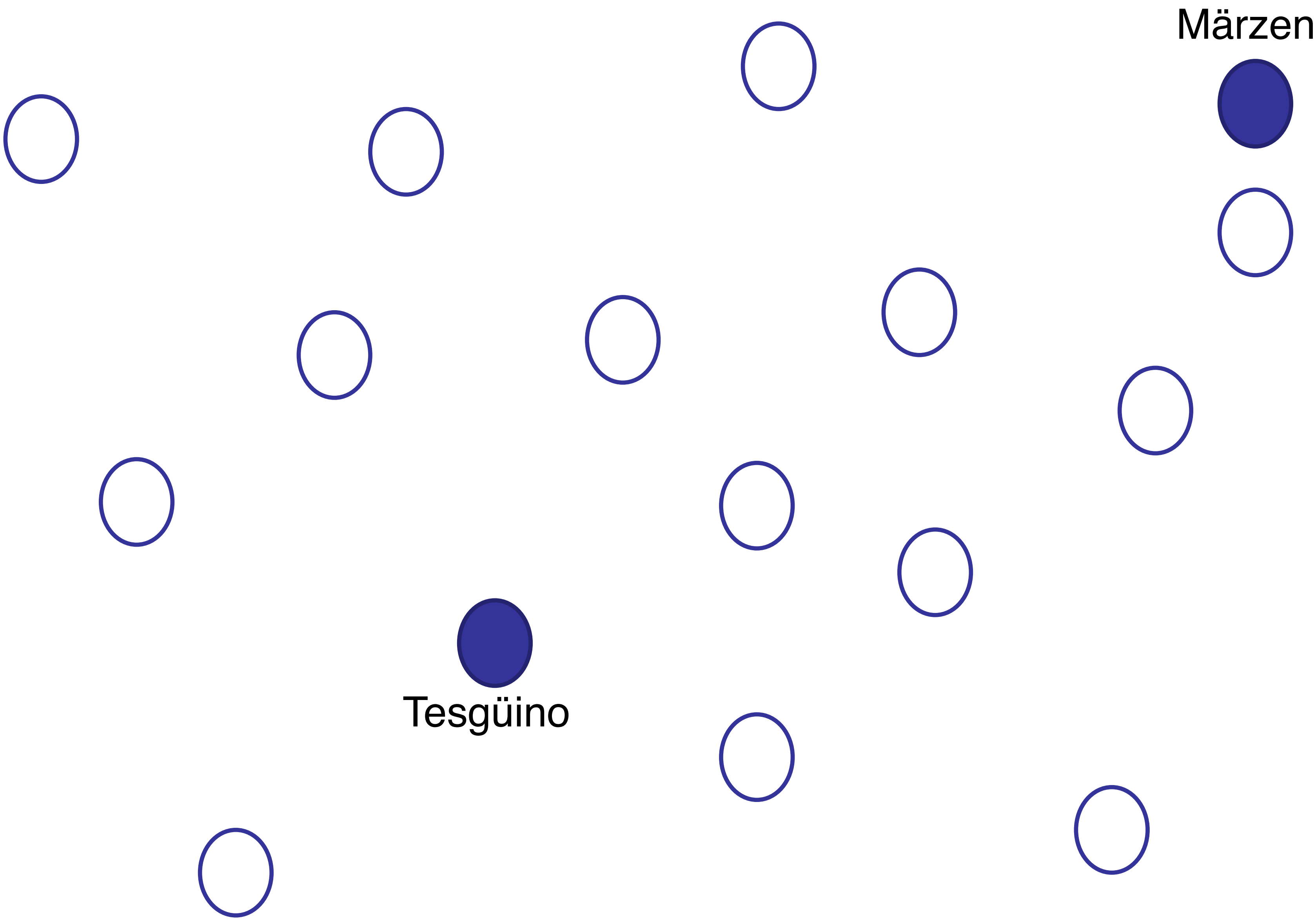
Another view!

Training data: `(Tesgüino, drink)`

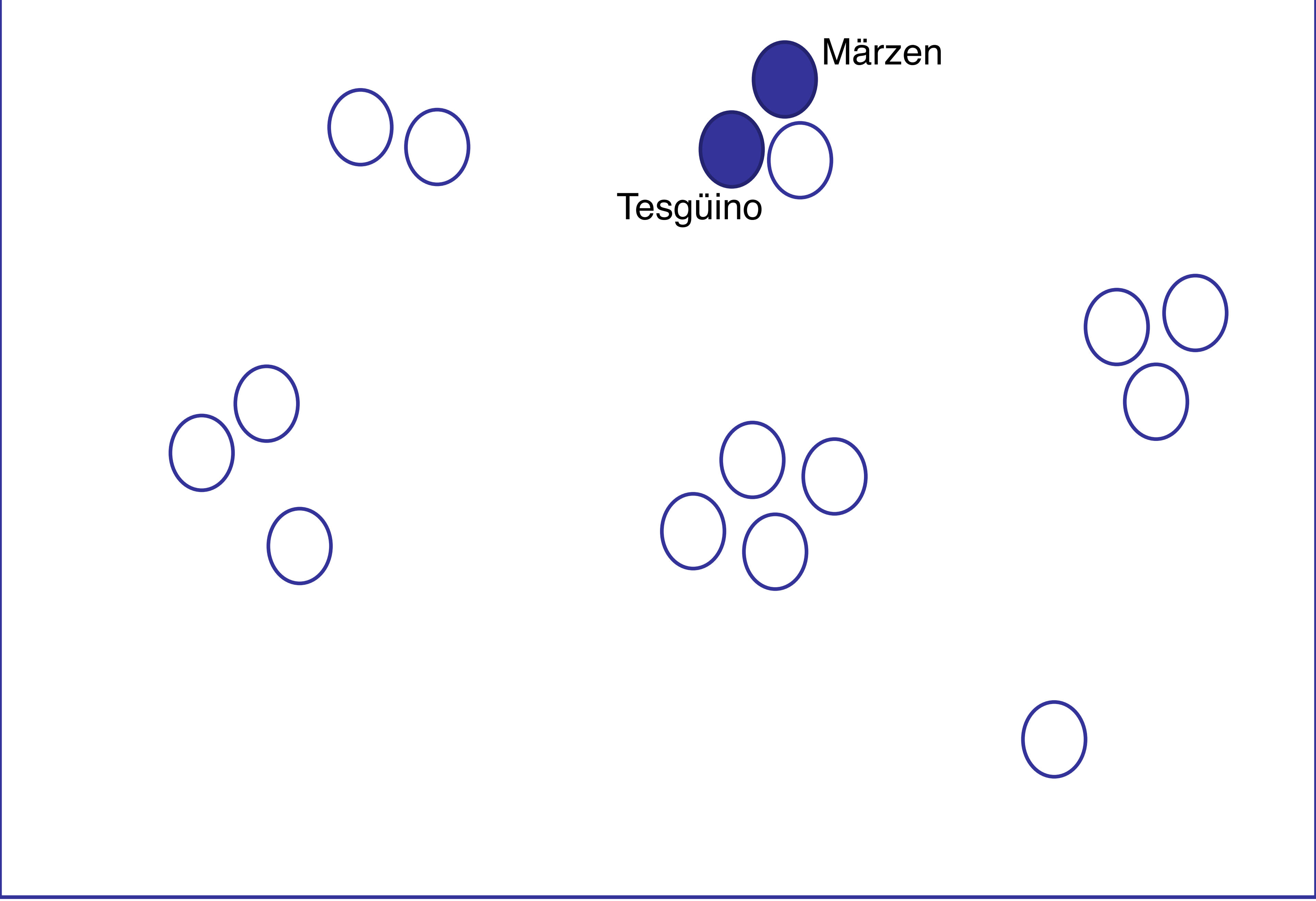


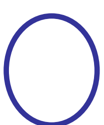
Encoder embedding
word embedding

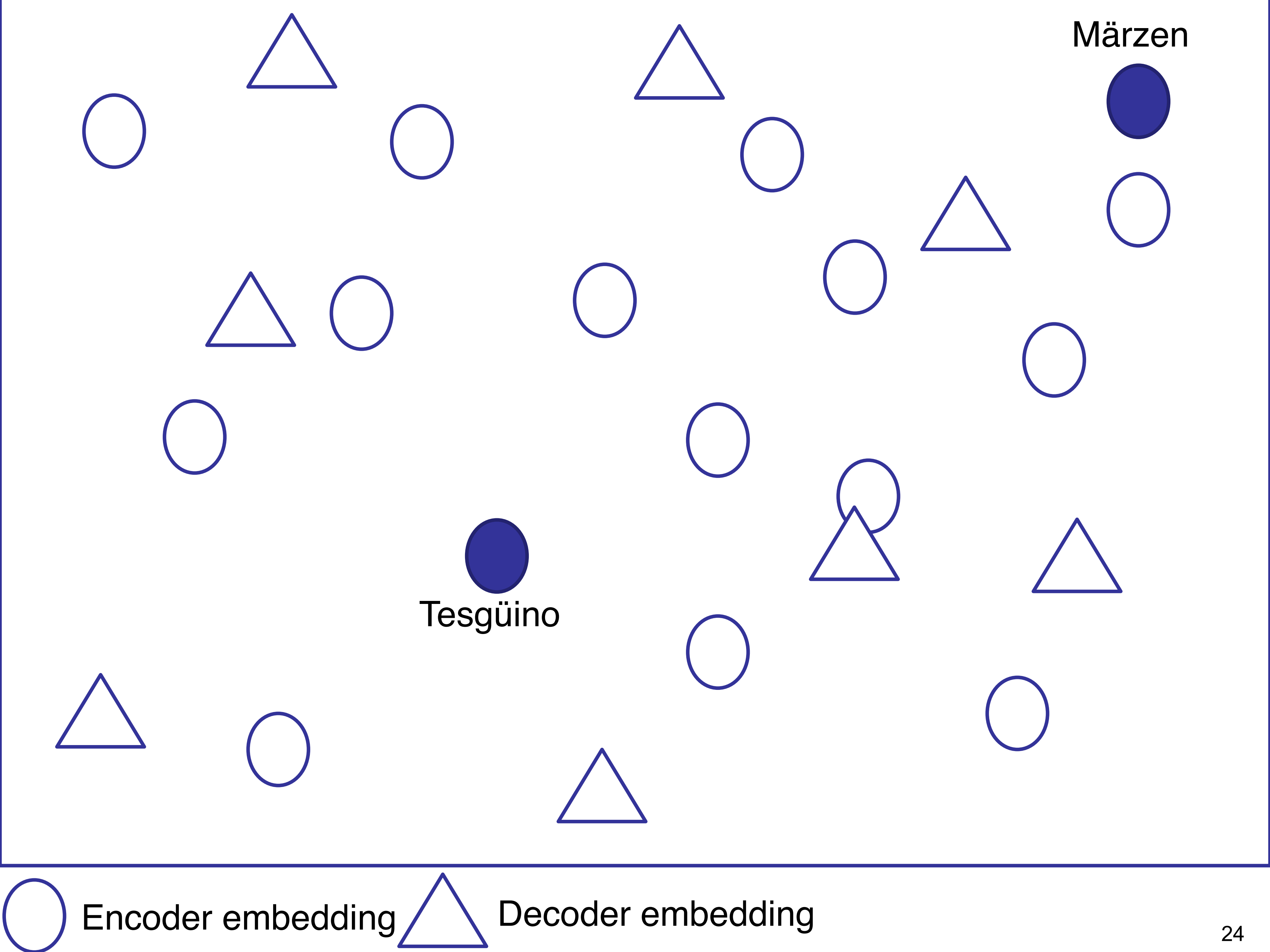
Decoder embedding
context-word embedding

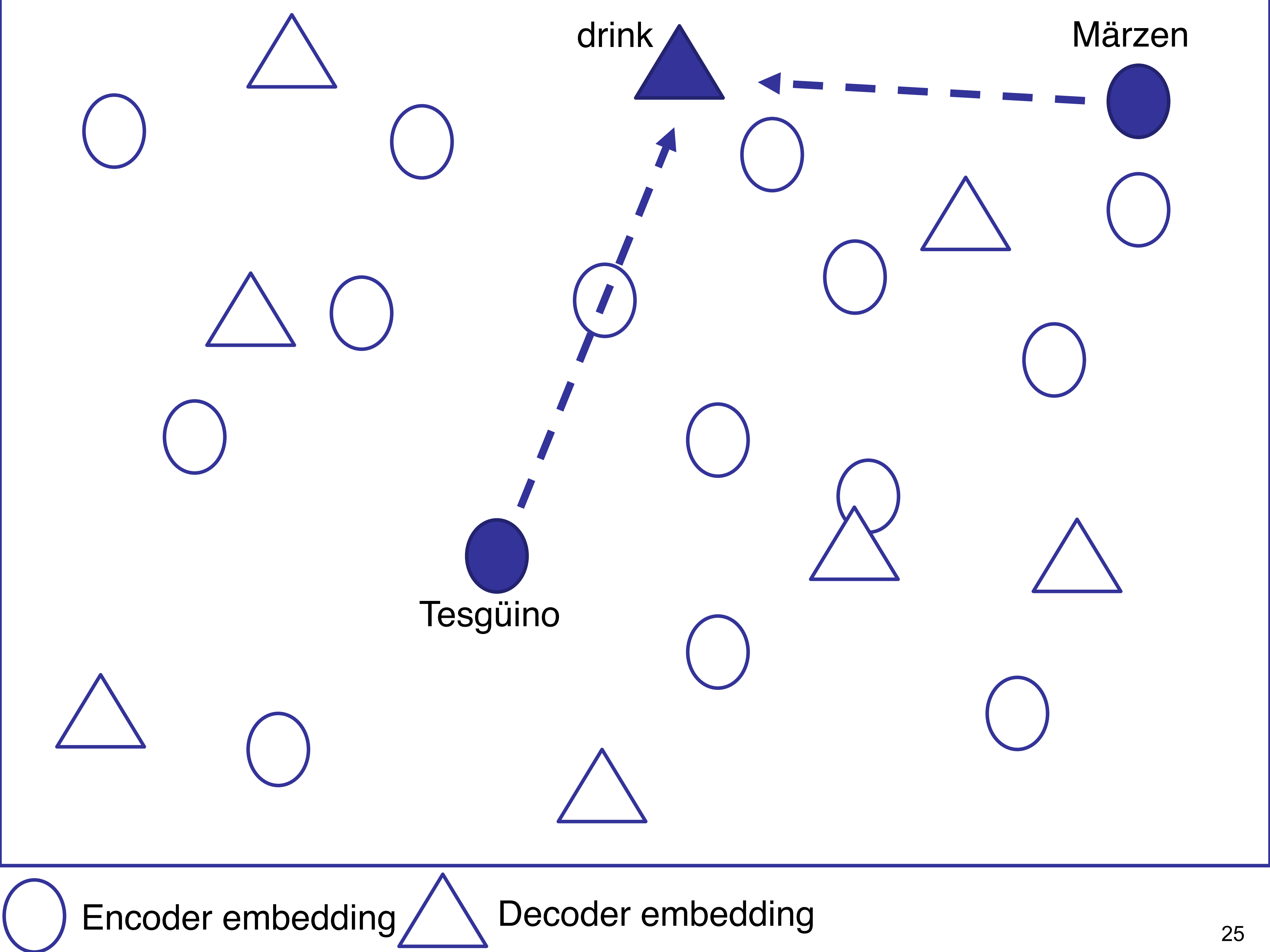


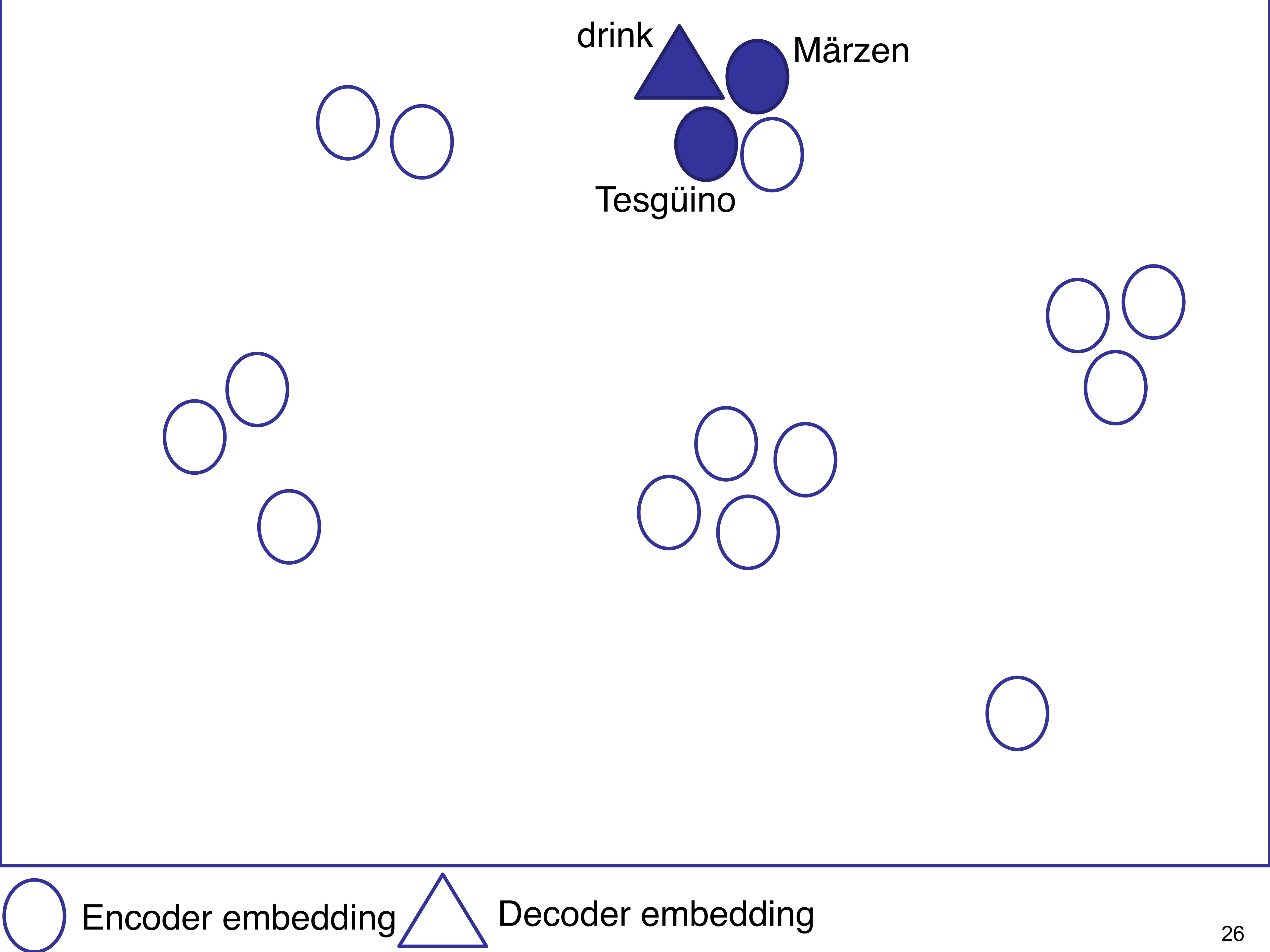
Encoder embedding

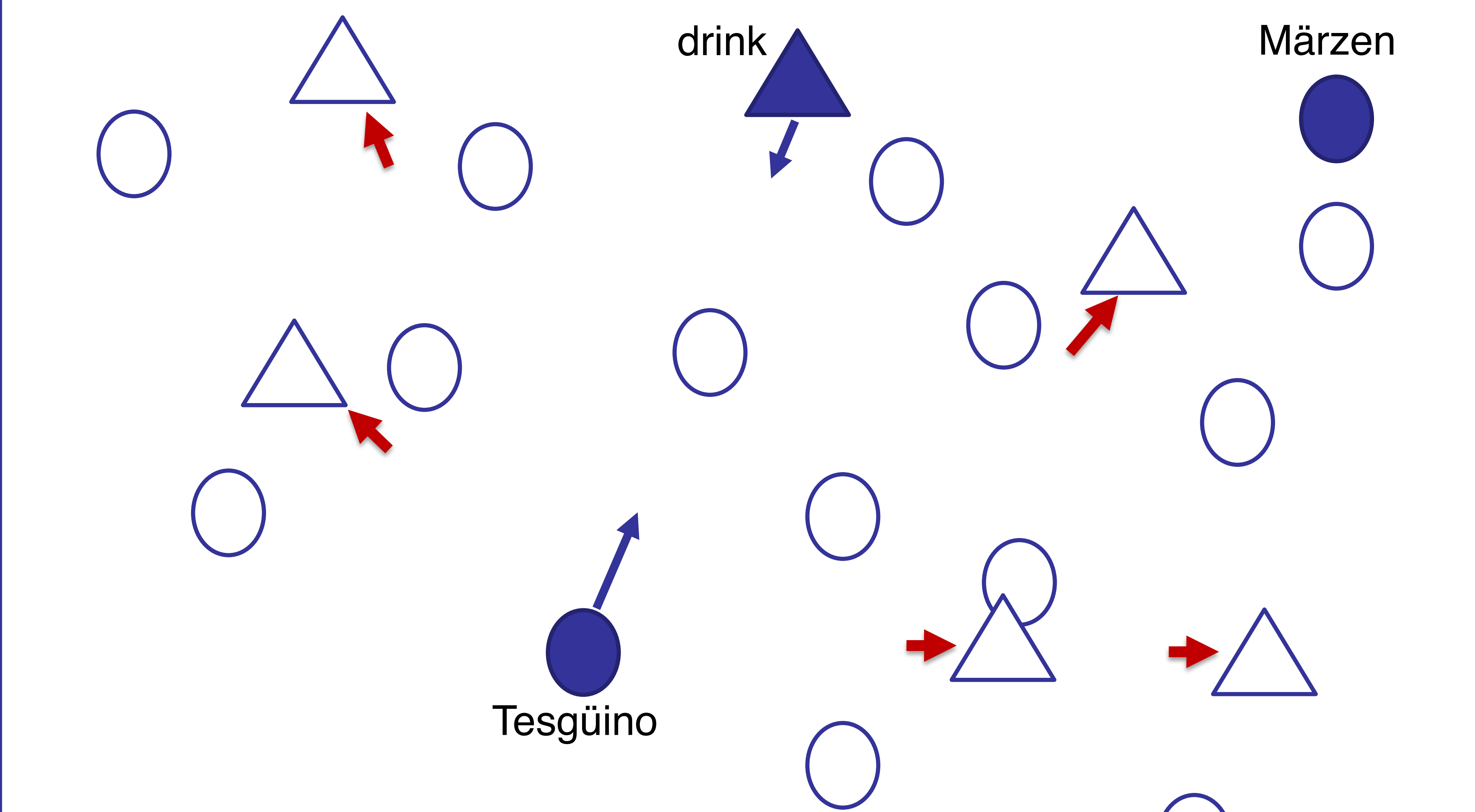


 Encoder embedding


Märzen
Tesgüino
Encoder embedding
Decoder embedding


drink
Märzen
Tesgüino
Encoder embedding
Decoder embedding


drink
Märzen
Tesgüino
Encoder embedding
Decoder embedding



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

Encoder embedding

Decoder embedding

Loss function – 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 can become a computation bottleneck!

when considering the very high number of the possible training data pairs in a corpus!

Agenda

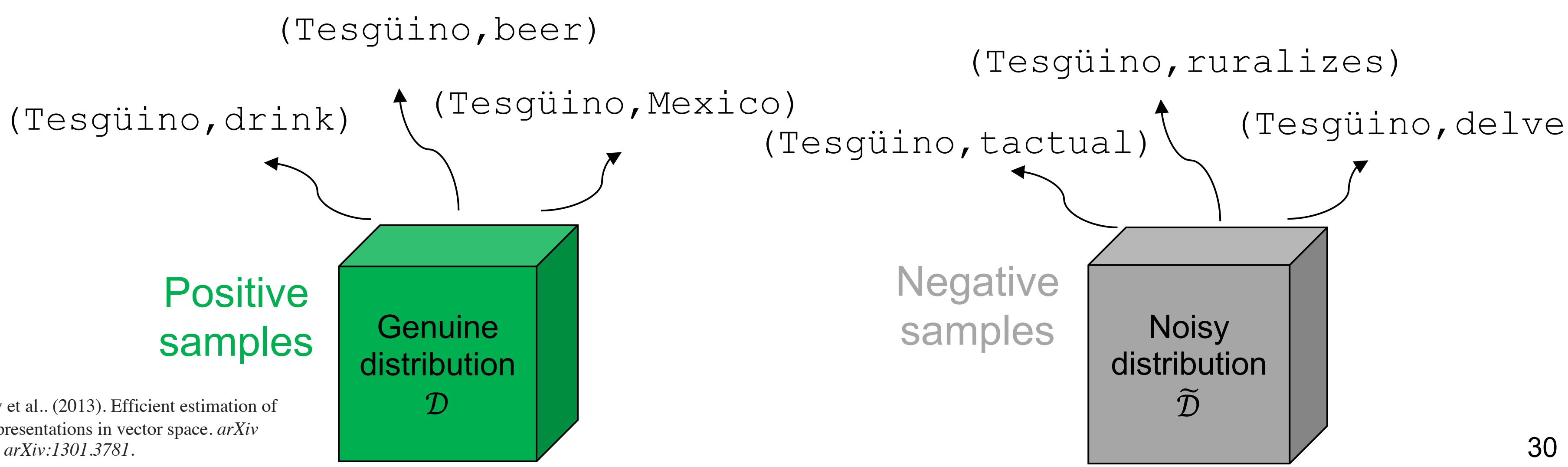
- Neural *n*-gram Language Model
- Neural skip-gram Language Model
- **word2vec**

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:

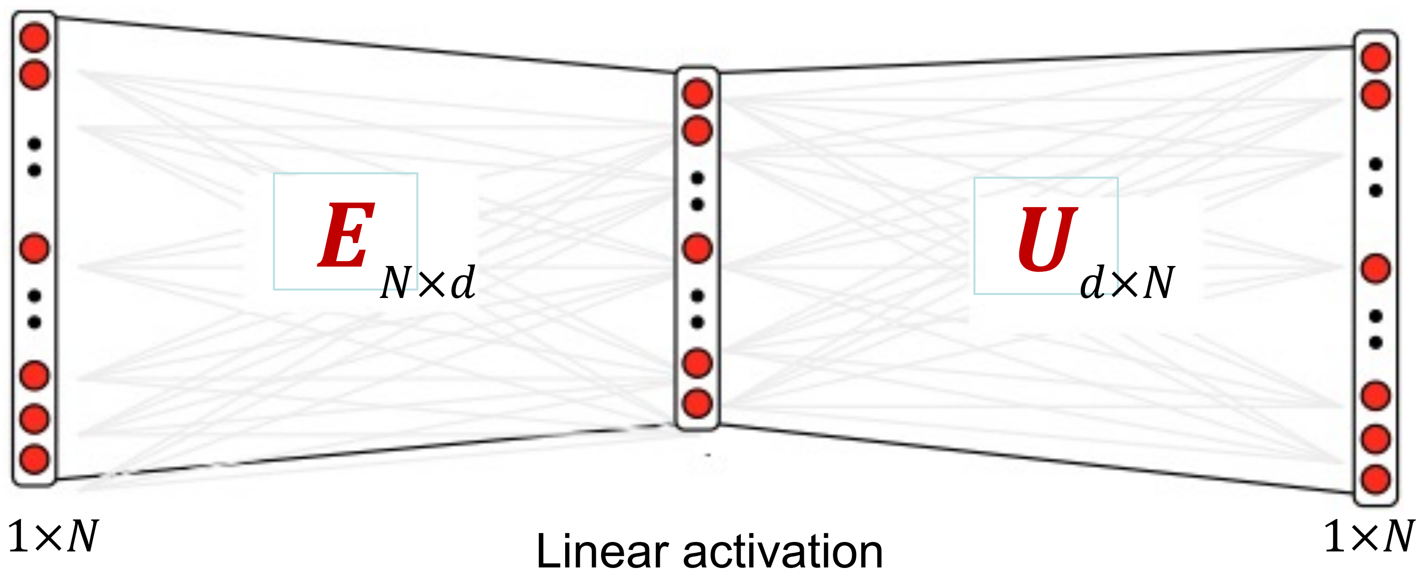
- Consider two data distributions that generate $(\text{word}\,,\text{context-}\,\text{word})$ pairs:
  1. A genuine distribution that generates the training data pairs $\rightarrow \mathcal{D}$
  2. A noisy distribution that generates random pairs $\rightarrow \widetilde{\mathcal{D}}$
- Objective: given a pair $(\text{word}\,,\text{context-}\,\text{word})$, the model should decide, whether the pair comes from the genuine or noisy distribution
  - Negative Sampling in fact turns the problem to a binary classification task



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

word2vec

- word2vec has the same architecture as the neural skip-gram Language Model



Encoder embedding
word embedding

Decoder embedding
context-word embedding

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

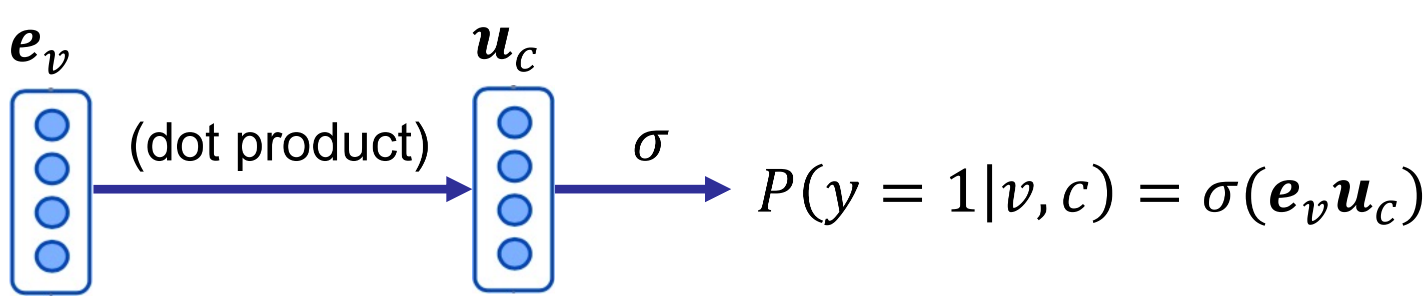
- Neural word embeddings' objective is $P(c|v)$
- word2vec instead calculates …

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

The probability that the pair (v,c) comes from the genuine data distribution

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

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



Training data

- To train the model, we use two sets of samples:
  - Positive sample: a pair (v, c) that comes from the genuine data distribution $\mathcal{D}$
    - $\mathcal{D}$ consists of every pair available in the training data
  - Negative sample: a pair $(v, \tilde{c})$ that is drawn from the noisy distribution $\widetilde{\mathcal{D}}$
    - $\widetilde{\mathcal{D}}$ consists of random pairs
      - Why can random pairs be considered as negative samples?
      - $\widetilde{\mathcal{D}}$ in word2vec is a *smoothed* unigram distribution of words in corpus. In word2vec's implementation, $\widetilde{\mathcal{D}}$ is further smoothed by raising unigram counts to the power of $\alpha = 0.75$

- Negative Sampling's objective is 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

Loss function

- Objective:
  - increase the probability of positive samples, $P(y = 1|v, c)$ and …
  - decrease the probabilities of k negative samples, $P(y = 1|v, \tilde{c})$

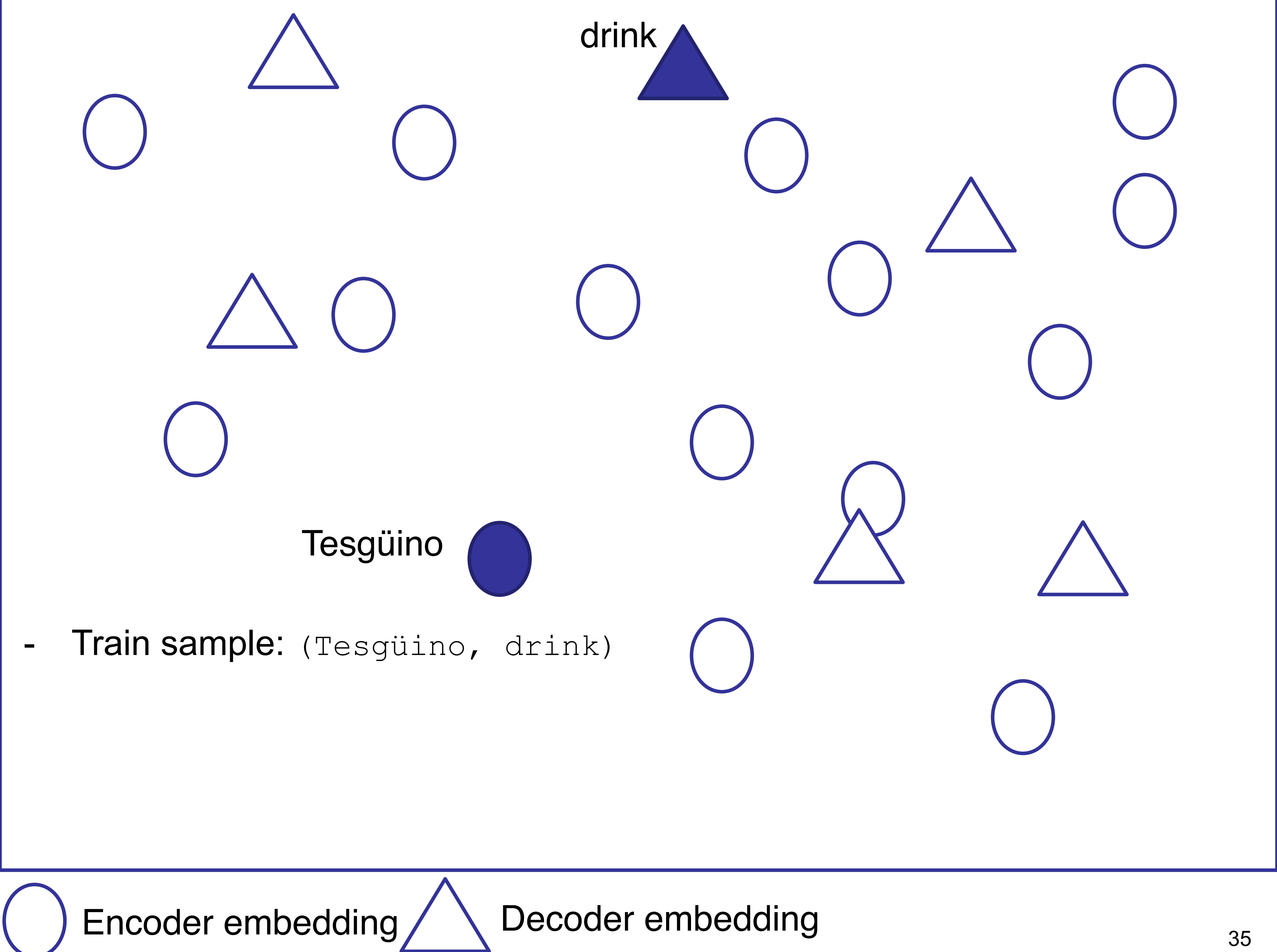
- Loss function:

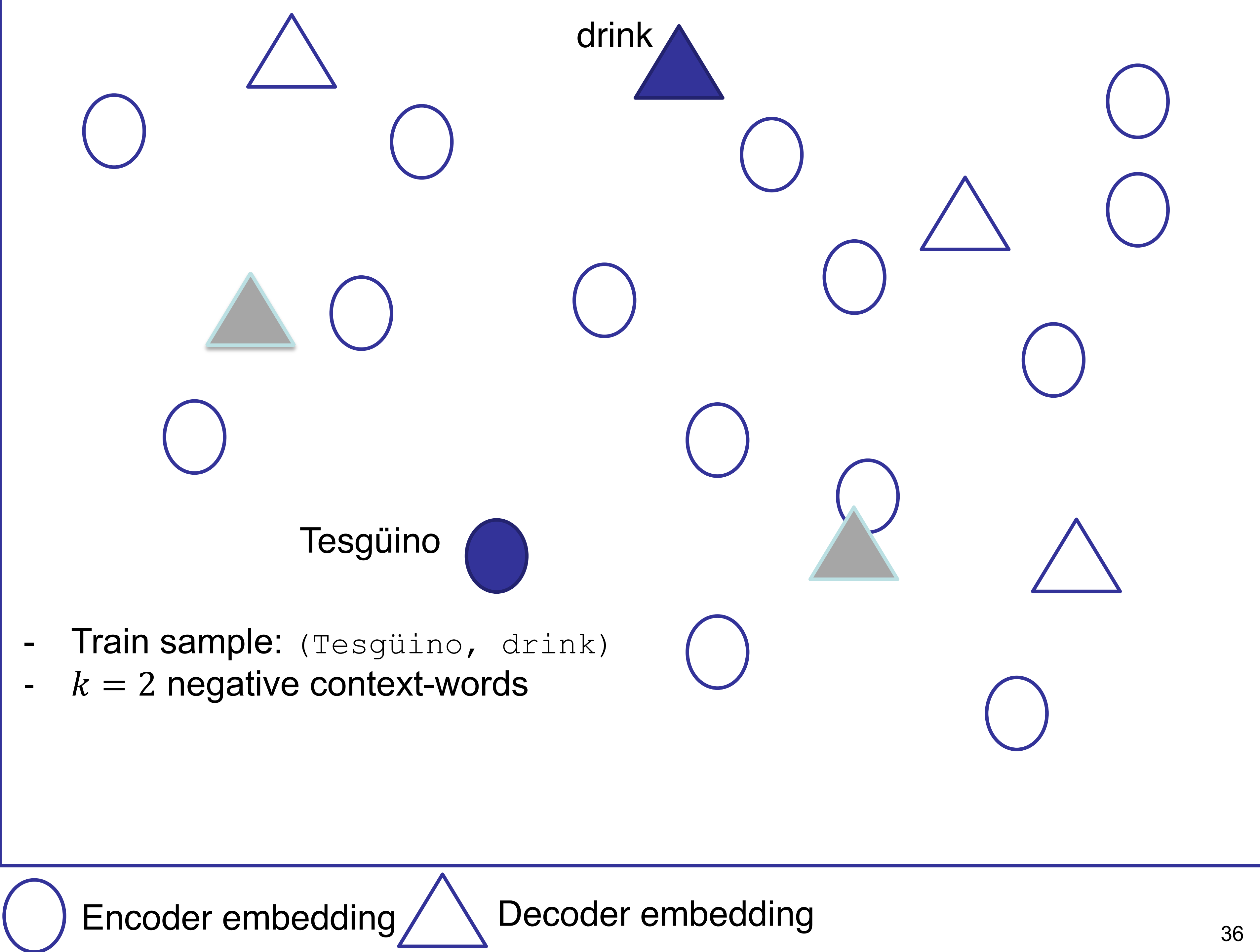
$$\mathcal{L} = -\mathbb{E}_{(v,c)\sim\mathcal{D}}\left[\log P(y = 1|v, c) - \sum_{\substack{\tilde{c}\sim\widetilde{\mathcal{D}} \\ k \text{ times}}} \log P(y = 1|v, \tilde{c})\right]$$

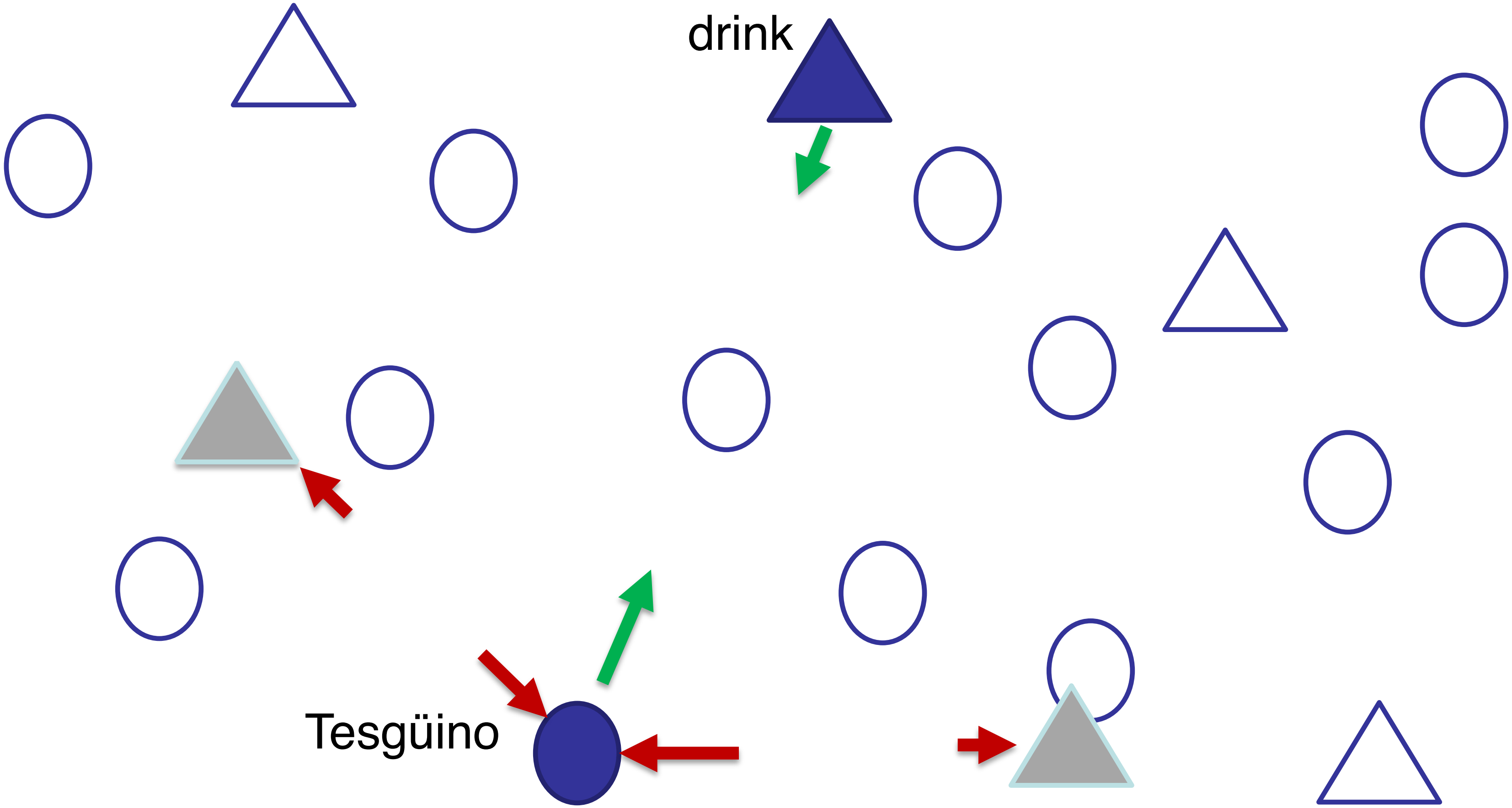
$$\mathcal{L} = -\mathbb{E}_{(v,c)\sim\mathcal{D}}\left[\log \sigma(\boldsymbol{e}_v \boldsymbol{u}_c) - \sum_{\substack{\tilde{c}\sim\widetilde{\mathcal{D}} \\ k \text{ times}}} \log \sigma(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}})\right]$$

positive samples

negative samples


drink
Tesgüino
- Train sample: (Tesgüino, drink)
Encoder embedding
Decoder embedding


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



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

Encoder embedding Decoder embedding

Final words!

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

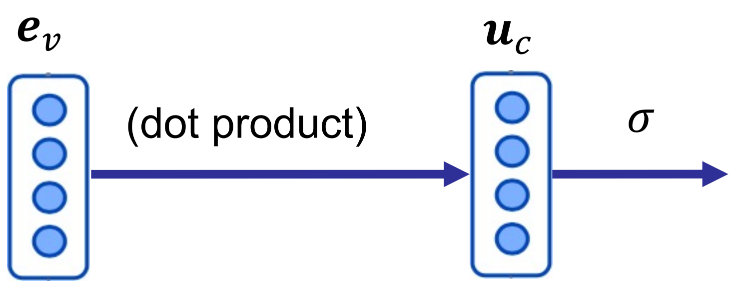
- Softmax is a good choice for training Language Models, namely to estimate $P(v|\text{context})$

- 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

word2vec skip-gram – summary

- word2vec creates word embeddings by …
  - following a skip-gram language modeling objective and …
  - exploiting Negative Sampling loss

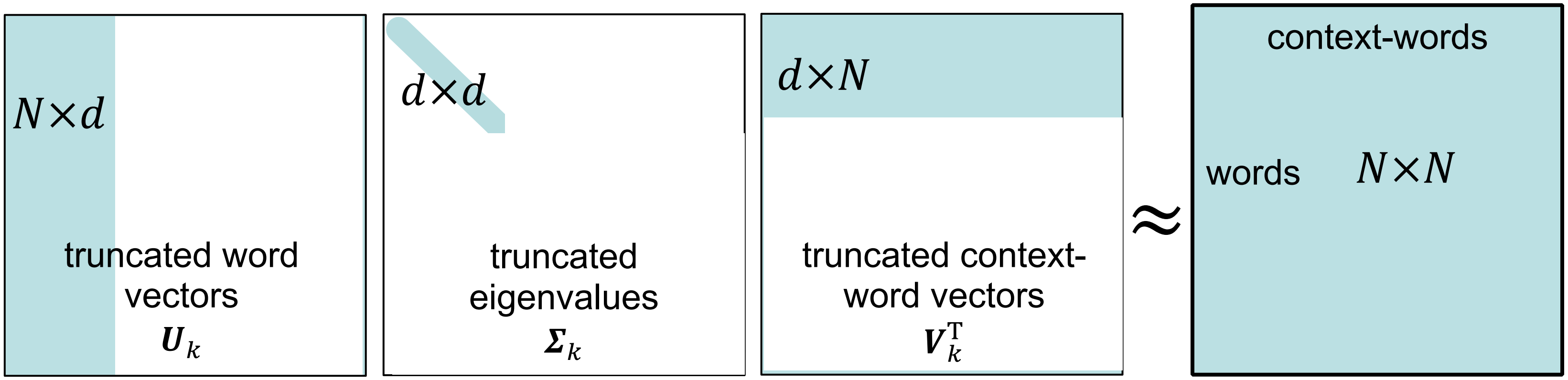


$$P(y = 1|v, c) = \sigma(\boldsymbol{e}_v \boldsymbol{u}_c)$$

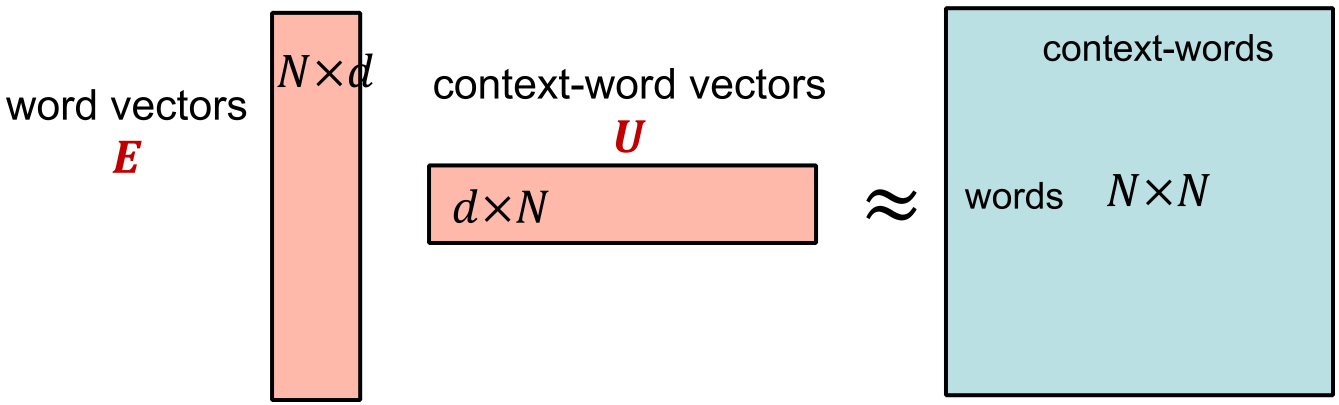
$$\mathcal{L} = -\left[\log \sigma(\boldsymbol{e}_v \boldsymbol{u}_c) - \sum_{\substack{\tilde{c} \sim \widetilde{\mathcal{D}} \\ k \text{ times}}} \log \sigma(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}})\right]$$

Three word embedding models in one frame!

PPMI+SVD:



GloVe:



word2vec skip-gram: