Winter semester 2022/23

344.175 VL: Natural Language Processing
Neural Word and Sentence Embeddings

Navid Rekab-saz

Email: navid.rekabsaz@jku.at
Office hours: https://navid-officehours.youcanbook.me

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

Agenda

- word2vec
  - Neural skip-gram Language Model
  - Negative sampling
- fastText
- Sentence embedding with sent2vec

Agenda

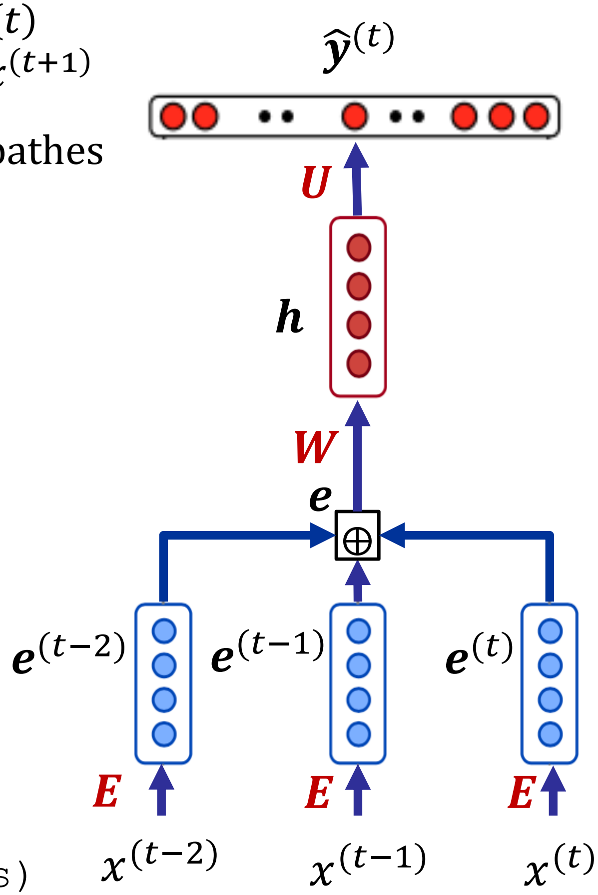
- **word2vec**
  - **Neural skip-gram Language Model**
  - Negative sampling
- fastText
- Sentence embedding with sent2vec

Neural *n*-gram Language Model – recap

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

$$P\left(x^{(t+1)} \middle| \text{a fluffy cat}\right) = \hat{y}^{(t)}_{x^{(t+1)}}$$

$$P(\text{sunbathes}|\text{a fluffy cat}) = \hat{y}^{(t)}_{\text{sunbathes}}$$

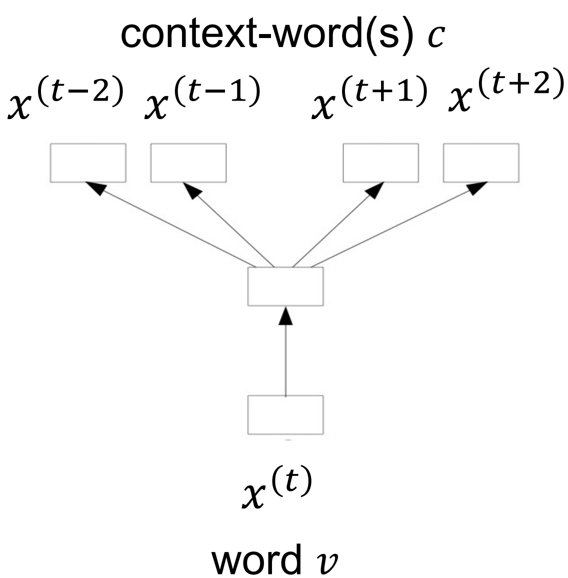


A data item in training data:
`(a fluffy cat, sunbathes)`

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

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	drink	Tesgüino	while	following	rituals	…

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

Tarahumara	people	drink	Tesgüino	while	following	rituals	…

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

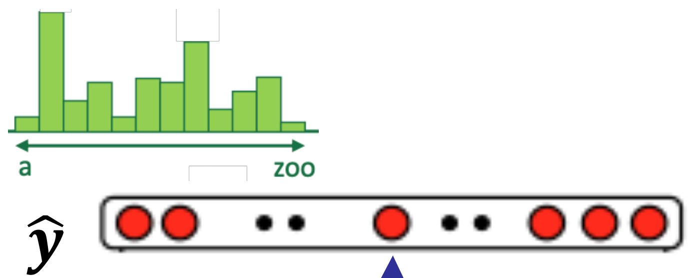
...

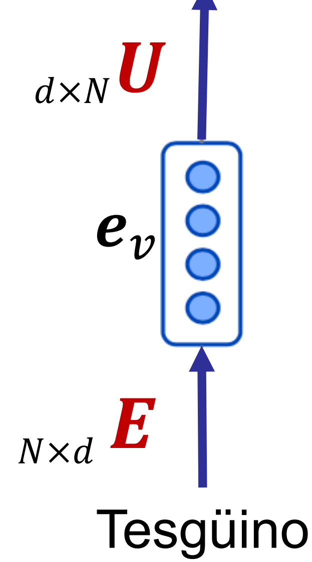
Tarahumara	people	drink	Tesgüino	while	following	rituals	…

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

Neural word embeddings from neural skip-gram Language Model

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





The model's parameters $\boldsymbol{E}$ and $\boldsymbol{U}$ are in fact two sets of word embeddings:

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

Training data: `(Tesgüino, drink)` v

N size of vocabulary
d embeddings dimension
Parameters are shown in red

Formulation

Encoder

- From words to word embeddings:
  - One-hot vector of word v is $\boldsymbol{v}$ vector: $\boldsymbol{v} \rightarrow 1 \times N$
    - In $\boldsymbol{v}$, all values are 0 and only the value corresponding to the word $\boldsymbol{v}$ is set to 1
  - Encoder word embedding: $\boldsymbol{e}_v = \boldsymbol{v}\boldsymbol{E}$ $\boldsymbol{e}_v \rightarrow 1 \times d$
    - In practice, $\boldsymbol{e}_v$ is achieved by fetching the embedding of v from $\boldsymbol{E}$ (no need for constructing $\boldsymbol{v}$)

$\boldsymbol{E} \rightarrow N \times d$

N size of vocabulary
d embeddings dimension
Parameters are shown in red

Formulation

Decoder

- Predicted logits:

$$\boldsymbol{z} = \boldsymbol{e}_v \boldsymbol{U} \qquad \boldsymbol{z} \rightarrow 1 \times N$$

- Predicted probability distribution:

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{z}) \qquad \hat{\boldsymbol{y}} \rightarrow 1 \times N$$

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

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

Putting all together:

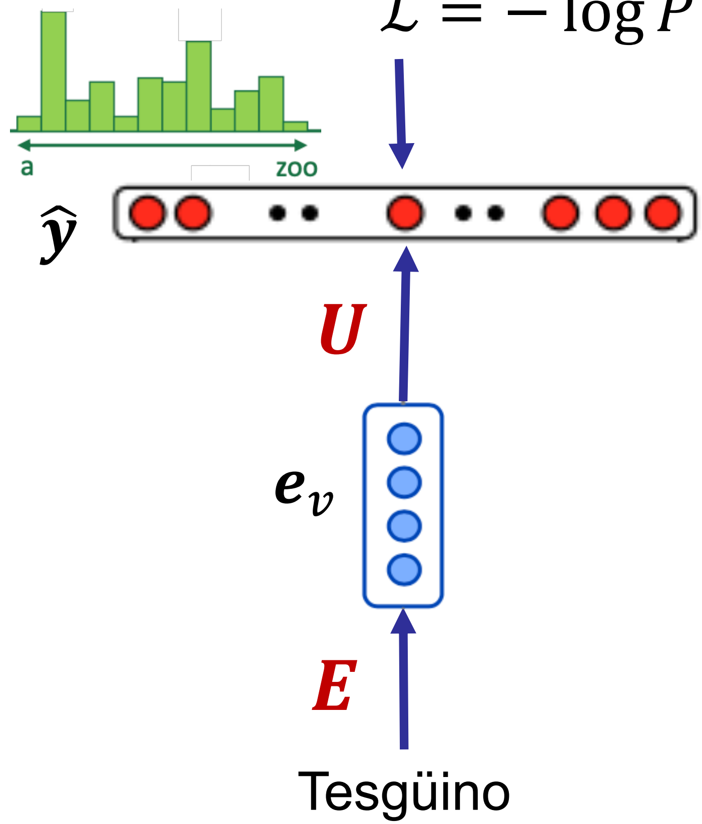
$$P(c|v) = \text{softmax}(\boldsymbol{e}_v \boldsymbol{U})_c = \frac{\exp(\boldsymbol{e}_v \boldsymbol{u}_c)}{\sum_{\tilde{c} \in \mathbb{V}} \exp(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}})}$$

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

Loss function

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

$\mathcal{L} = -\log P(c|v)$



Training data: `(Tesgüino, drink)` v

Skip-gram Language Model – all together

- Probability distribution of output words:

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

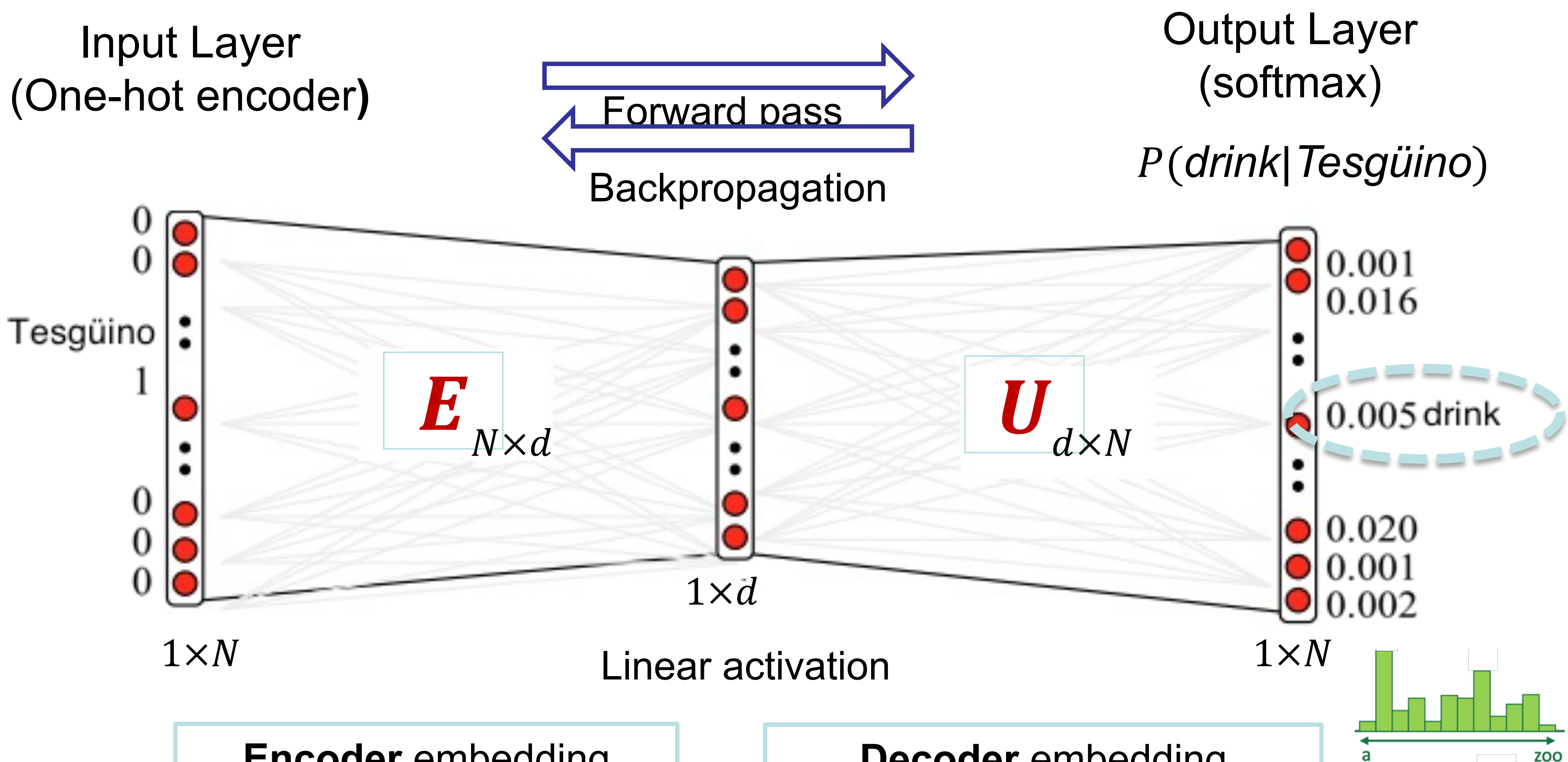
  - In the example: $P(\text{drink}|\text{Tesgüino}) = \frac{\exp(\boldsymbol{e}_{\text{Tesgüino}} \boldsymbol{u}_{\text{drink}})}{\sum_{\tilde{c} \in \mathbb{V}} \exp(\boldsymbol{e}_{\text{Tesgüino}} \boldsymbol{u}_{\tilde{c}})}$

- 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)`

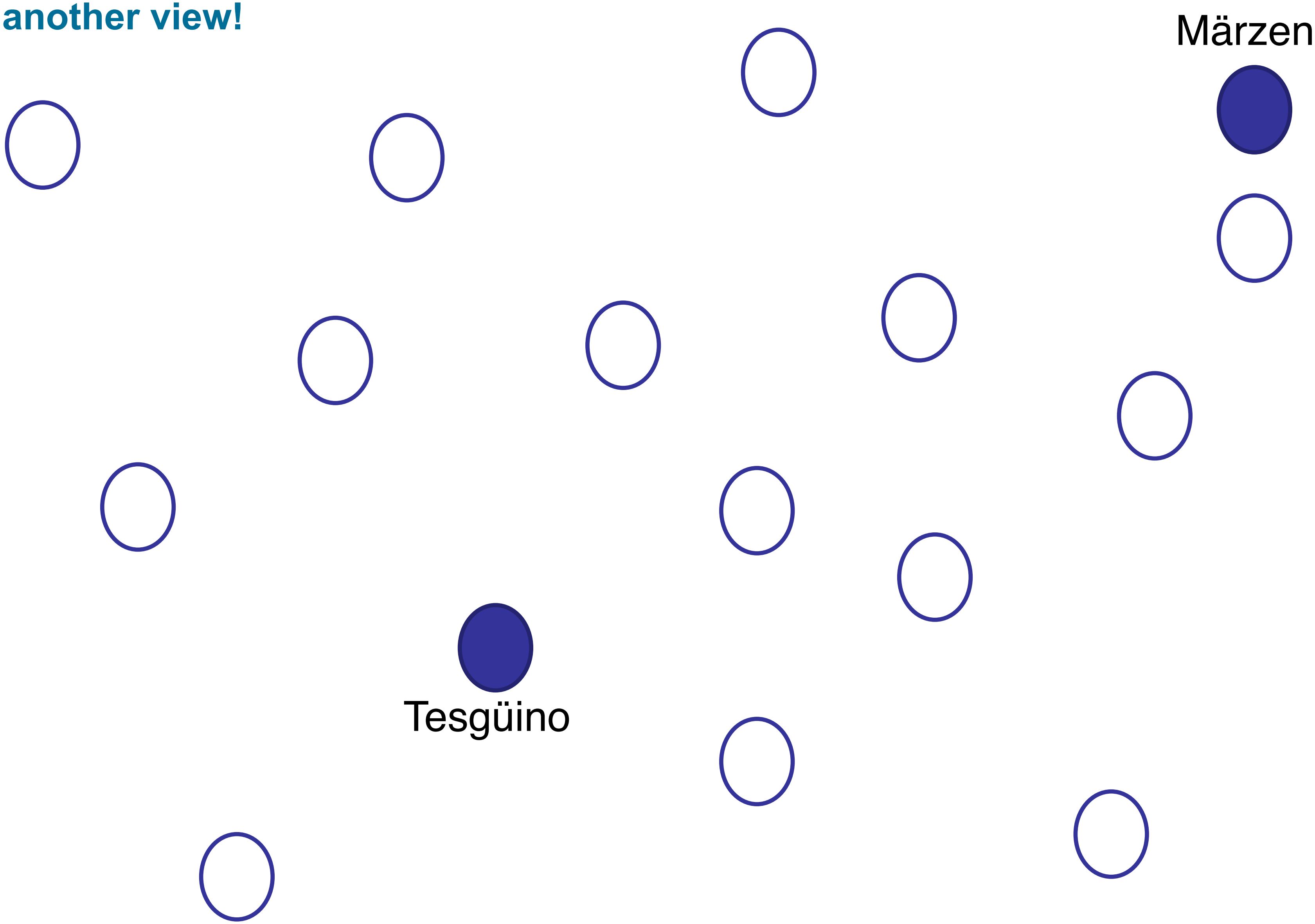


Encoder embedding
word embedding

Decoder embedding
context-word embedding

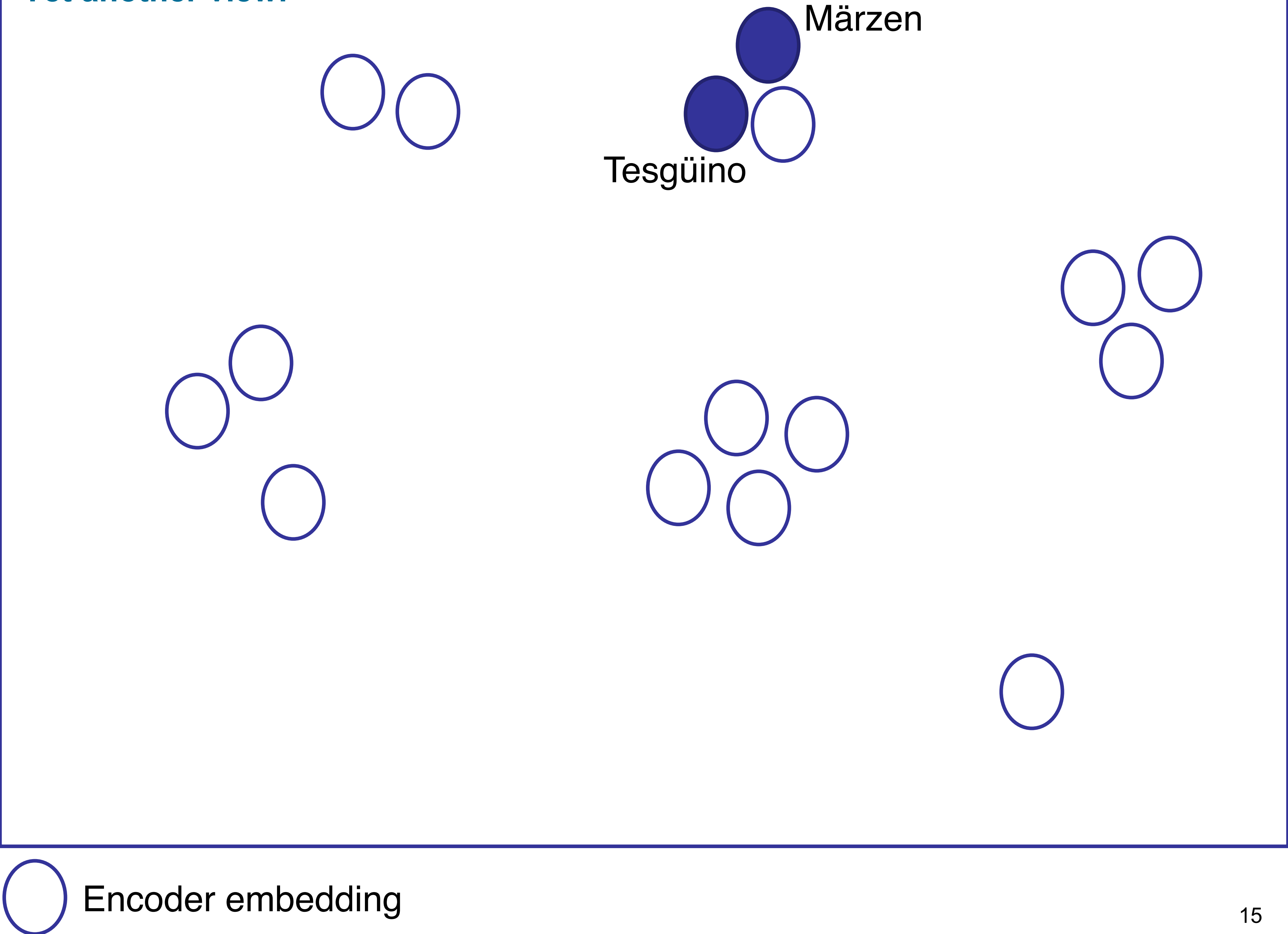
https://web.stanford.edu/~jurafsky/slp3/

Yet another view!

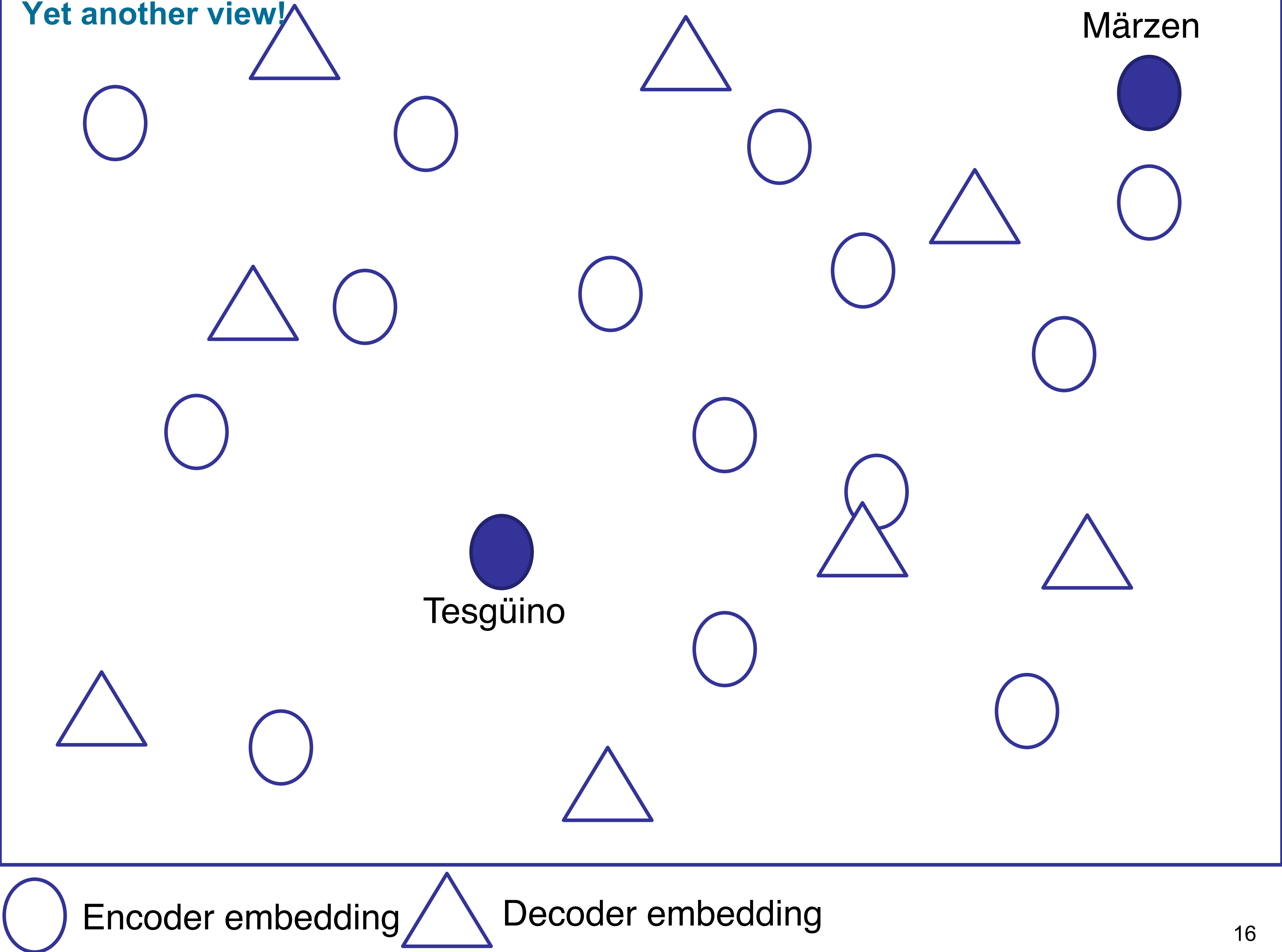


Encoder embedding

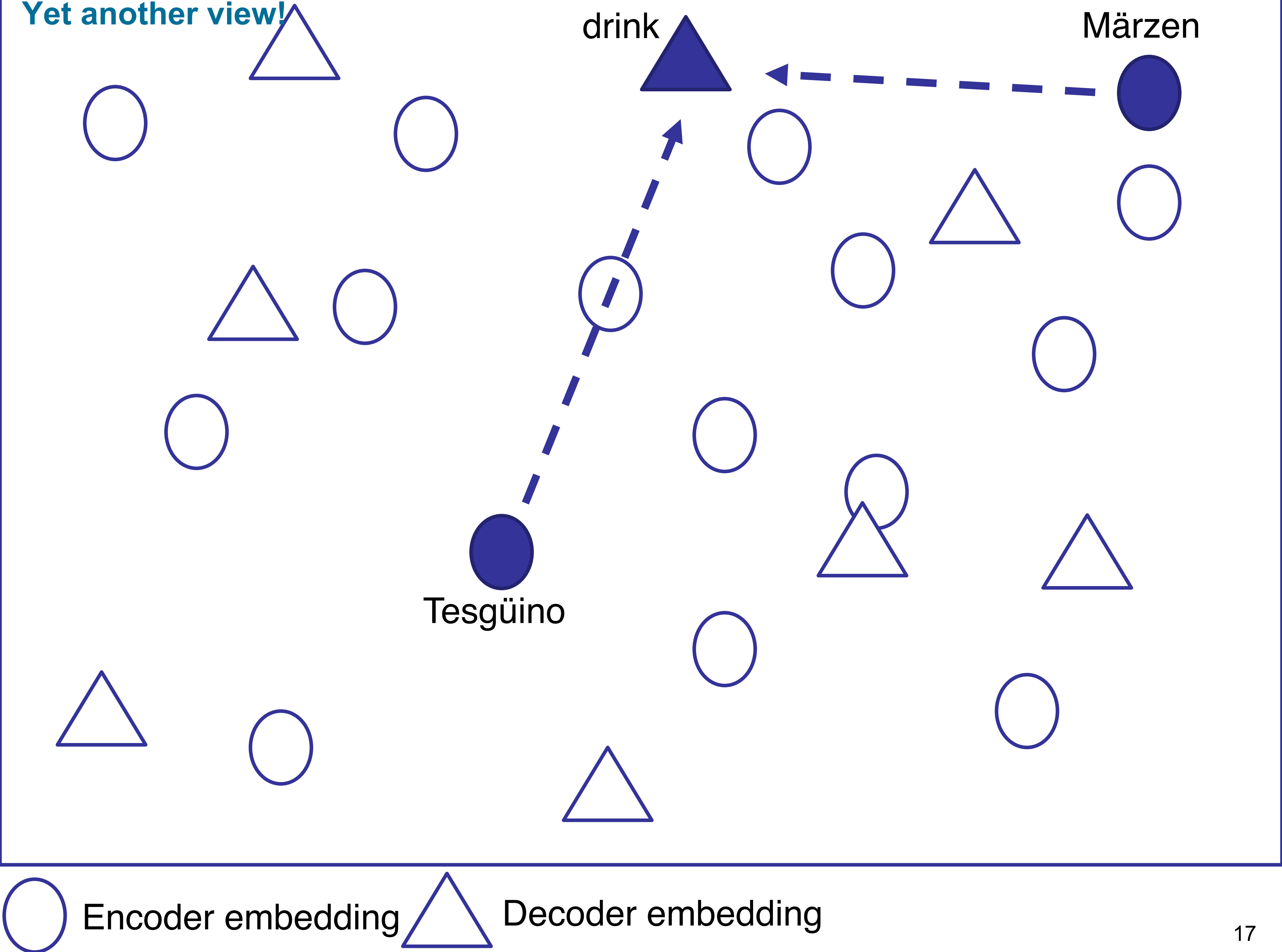
Yet another view!



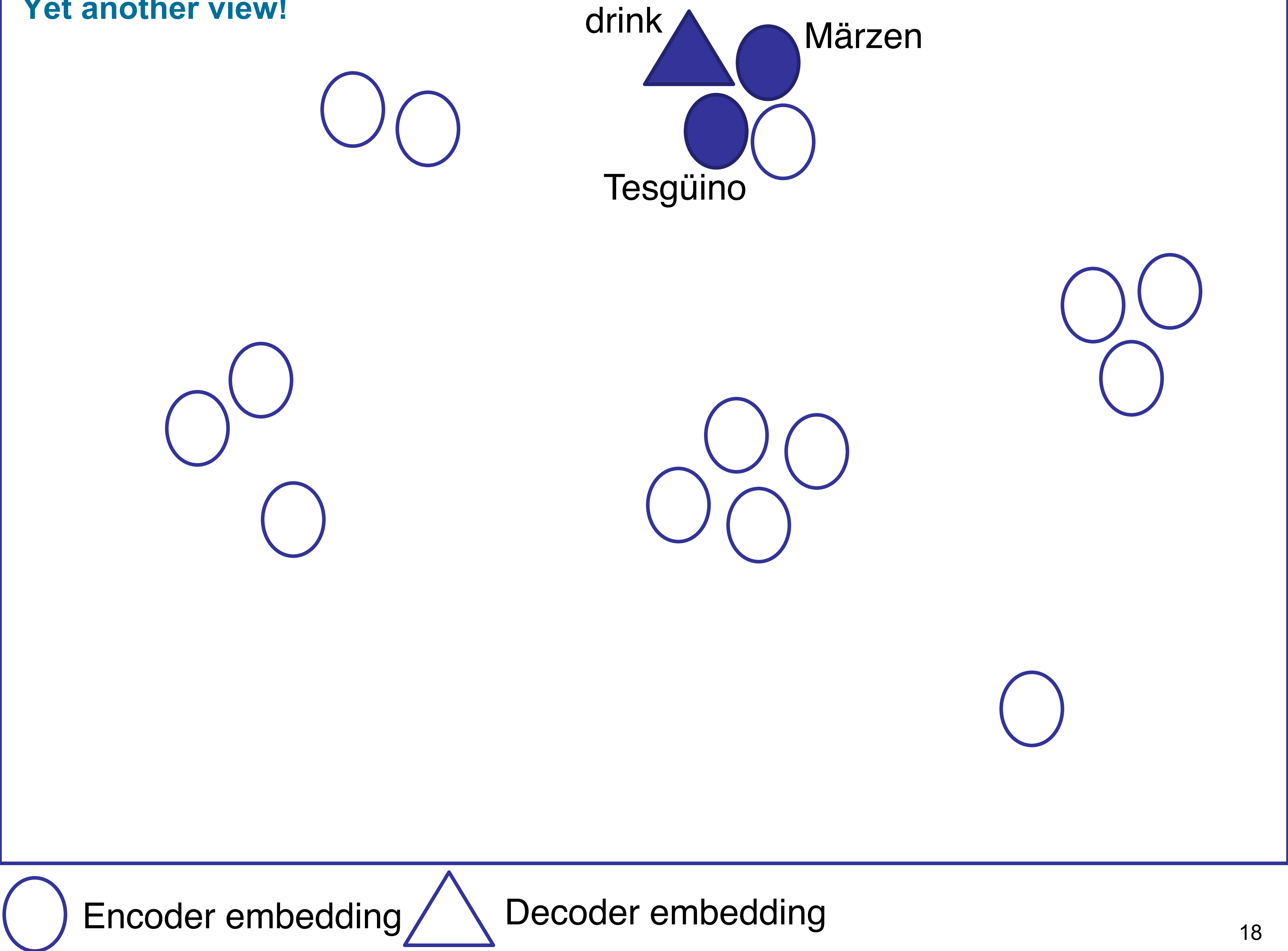
Yet another view!



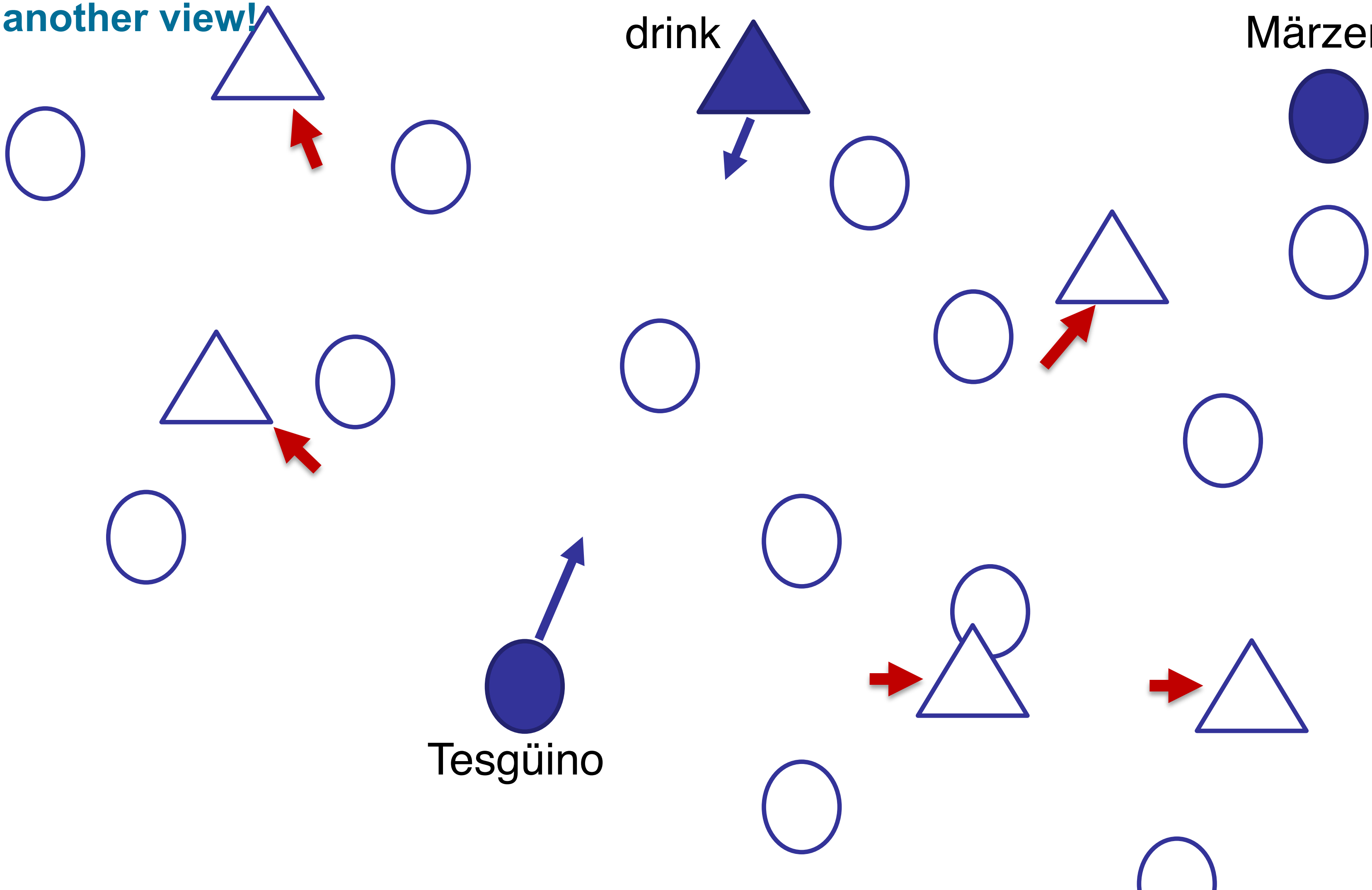
Yet another view!



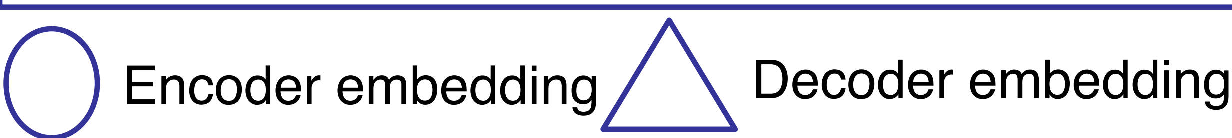
Yet another view!



Yet another view!



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



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

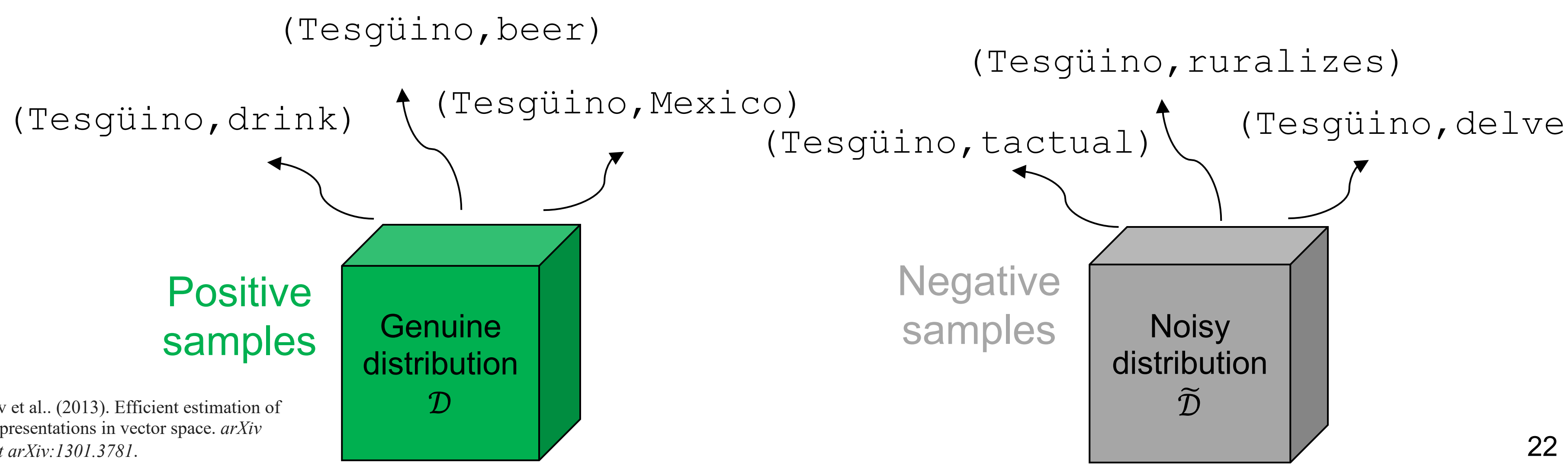
- **word2vec**
  - Neural skip-gram Language Model
  - **Negative sampling**
- fastText
- Sentence embedding with sent2vec

word2vec: skip-gram with Negative Sampling

- word2vec is an efficient and effective algorithm that proposes Negative Sampling method to define loss

Core idea in Negative Sampling (a form of contrastive learning):

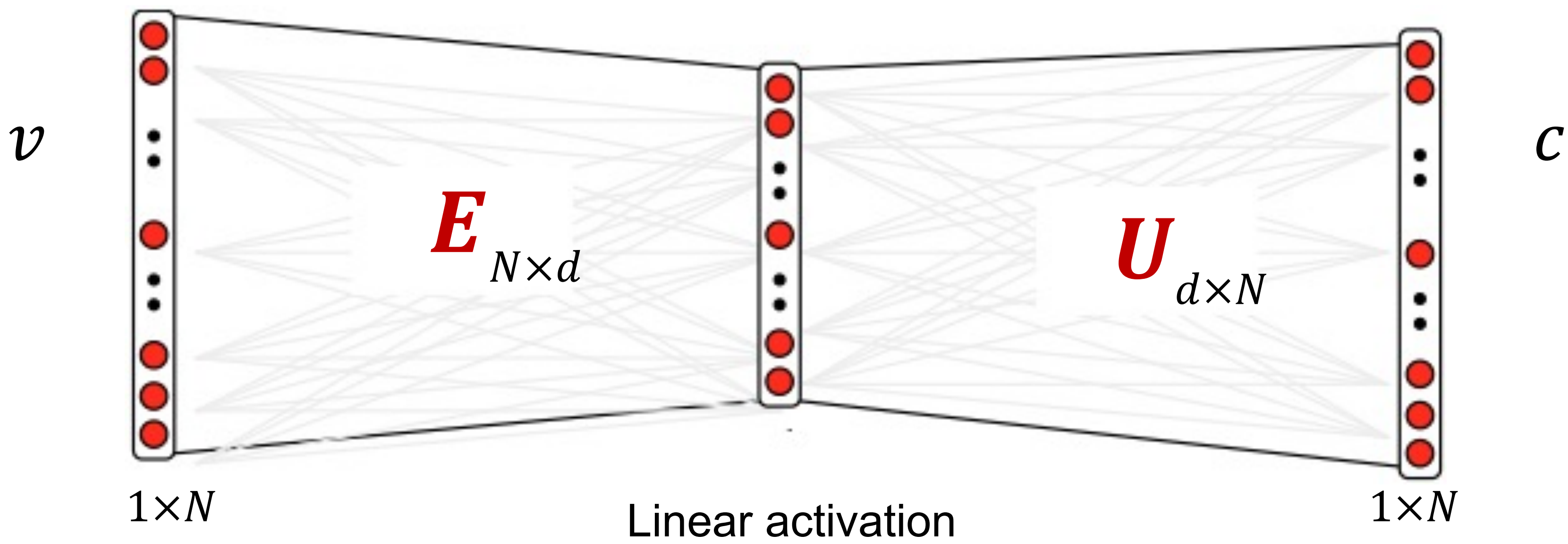
- Consider two data distributions that generate $(\text{word}\,,\text{context-word})$ pairs:
  1. A genuine distribution that generates the training data pairs $\rightarrow \mathcal{D}$
  2. A noisy distribution that generates random pairs $\rightarrow \widetilde{\mathcal{D}}$
- Objective: given a pair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



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

word2vec – architecture

- Starting from neural skip-gram Language Model



Encoder embedding
word embedding

Decoder embedding
context-word embedding

Negative Sampling – prediction probability

- Standard skip-gram approach calculates $P(c|v)$
  - Probability for a multi-class classification task
- Negative Sampling instead calculates the probability below

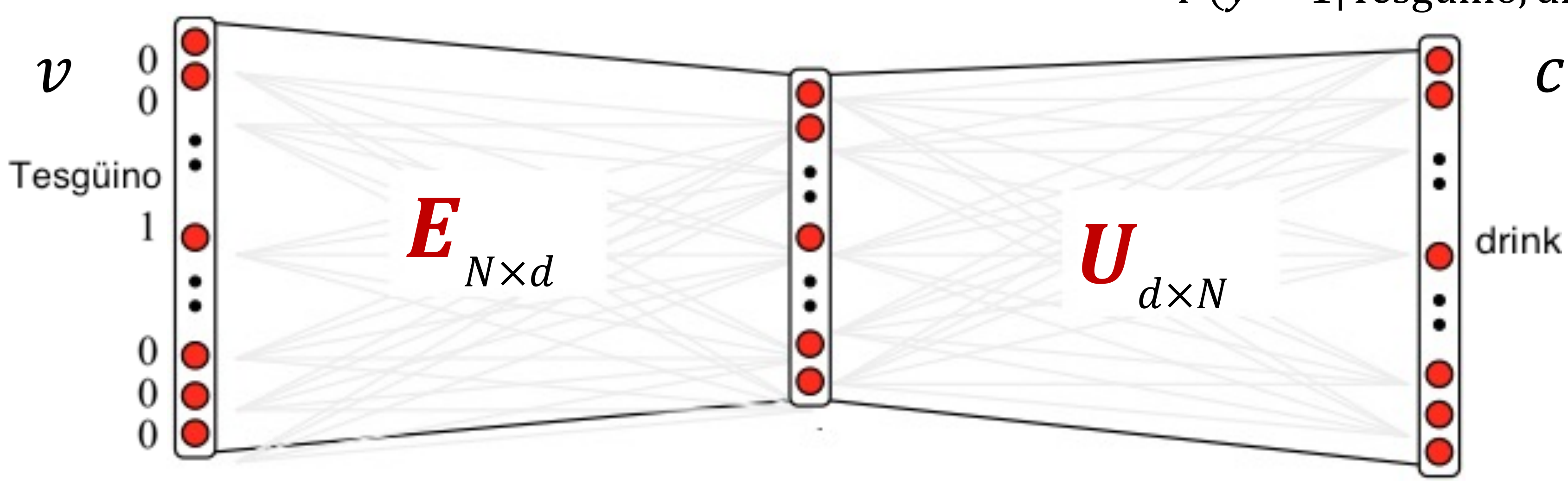
$$P(y = 1|v, c)$$

Probability that (v, c) comes from the genuine data distribution

Probability for a binary classification task

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

$$P(y = 1|\text{Tesgüino}, \text{drink})$$



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

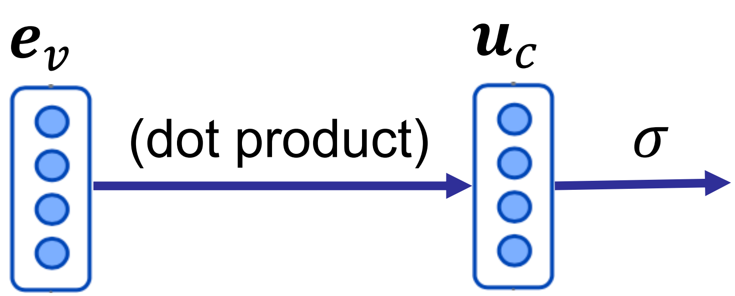
$$P(y = 1|v, c)$$

Probability that (v, c) comes from the genuine data distribution

Probability for a binary classification task

- $P(y = 1|v, c)$ is defined as sigmoid σ of the logit $\boldsymbol{e}_v \boldsymbol{u}_c$:

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



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

Negative Sampling training data and objective
(or generally in contrastive learning)

Training data

- Training data consists of two sets of samples:
  - Positive sample: a pair (v, c) from the genuine distribution $\mathcal{D}$
    - $\mathcal{D}$ is the set of all (v, c) pairs appearing in the training corpus
  - Negative sample: a pair $(v, \tilde{c})$ from the noisy distribution $\widetilde{\mathcal{D}}$
    - $\widetilde{\mathcal{D}}$ is a set of randomly selected $(v, \tilde{c})$ pairs (why?)
    - $\widetilde{\mathcal{D}}$ is created by randomly sampling from a *smoothed* unigram distribution of the words in the training corpus

Objective

- Train a model that distinguishes between the positive and negative samples, namely:
  - 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, $P(y=1|v,\tilde{c})$

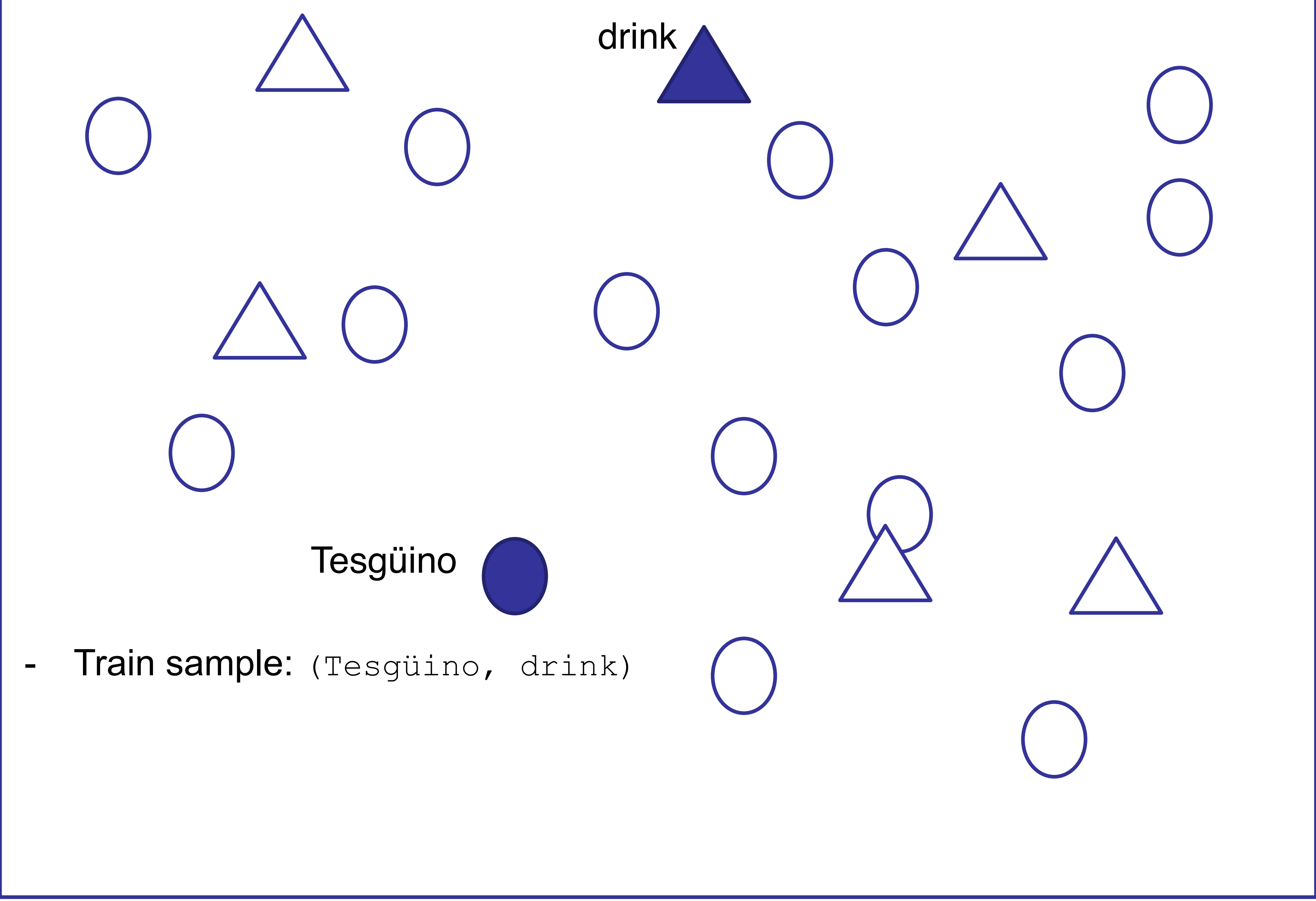
- Loss function:

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

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

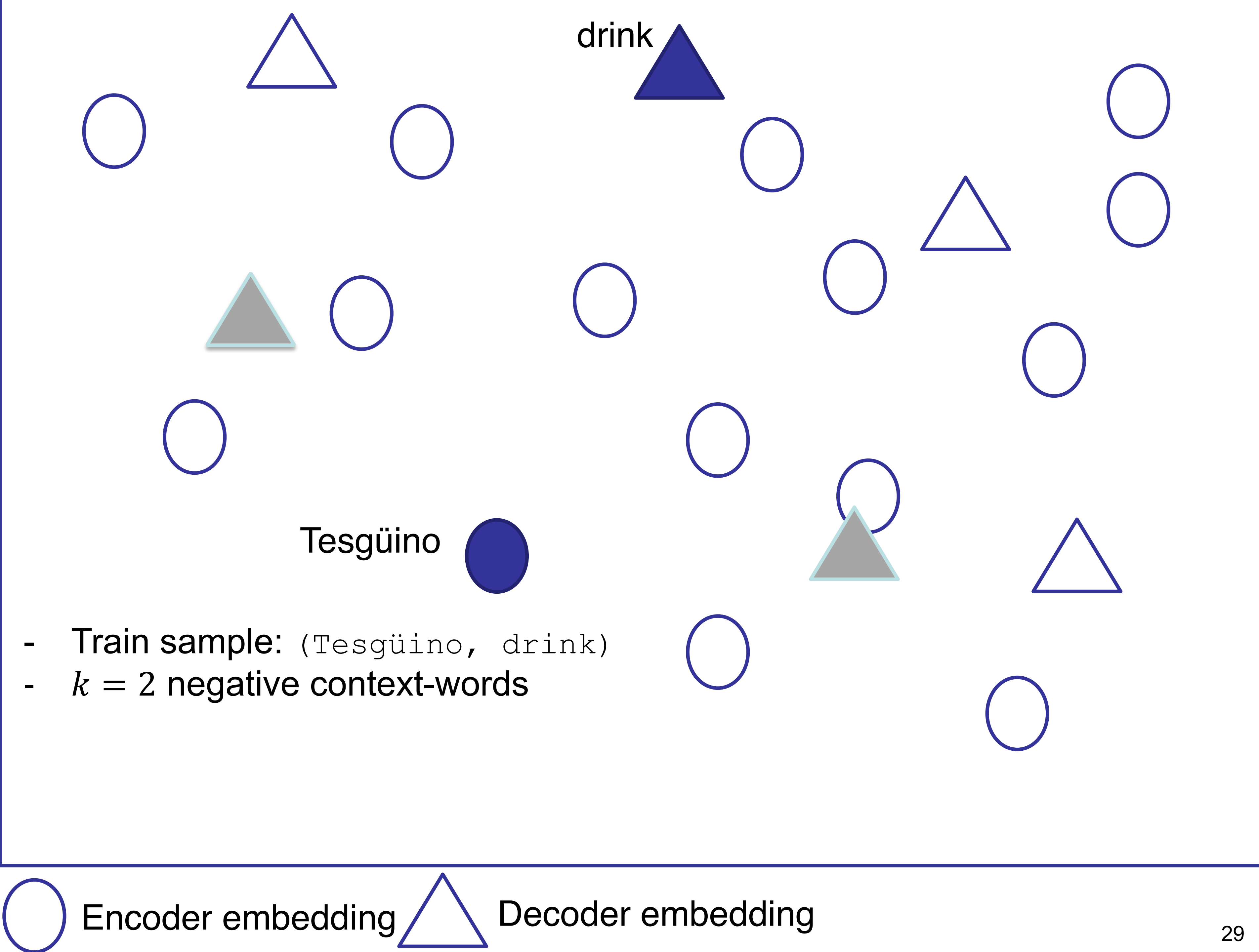
positive samples

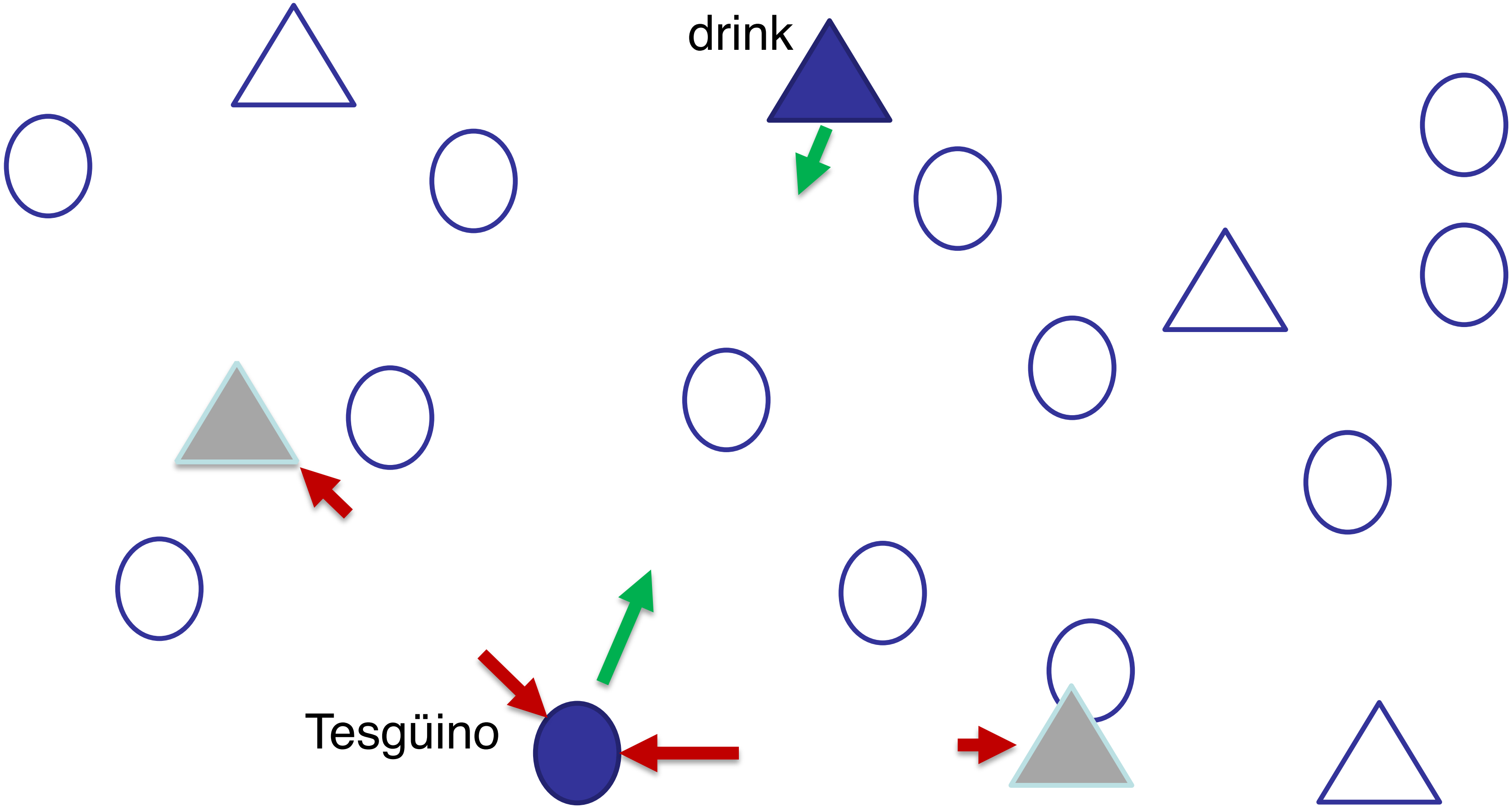
negative samples



- 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

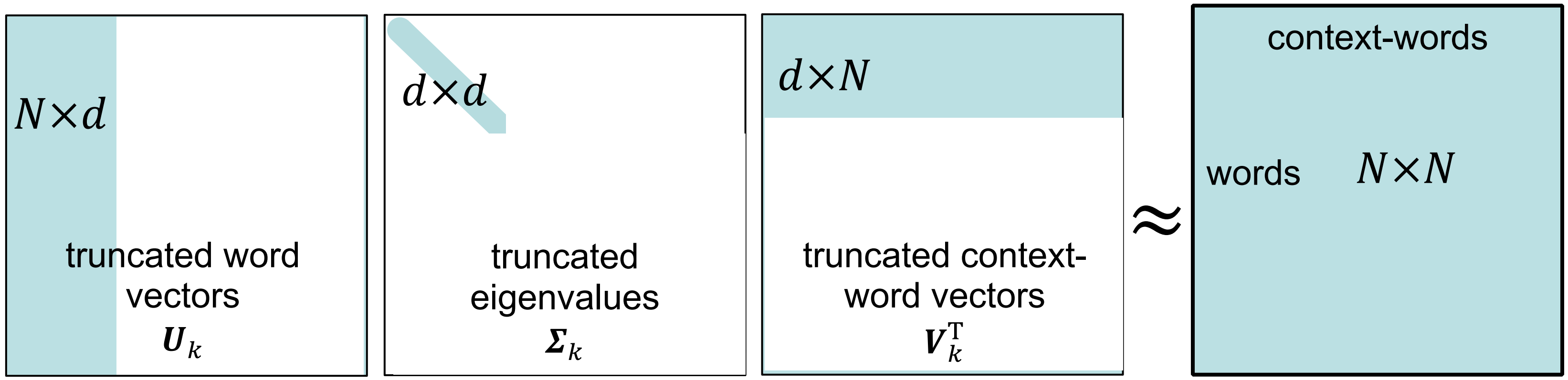
Final words!

- 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

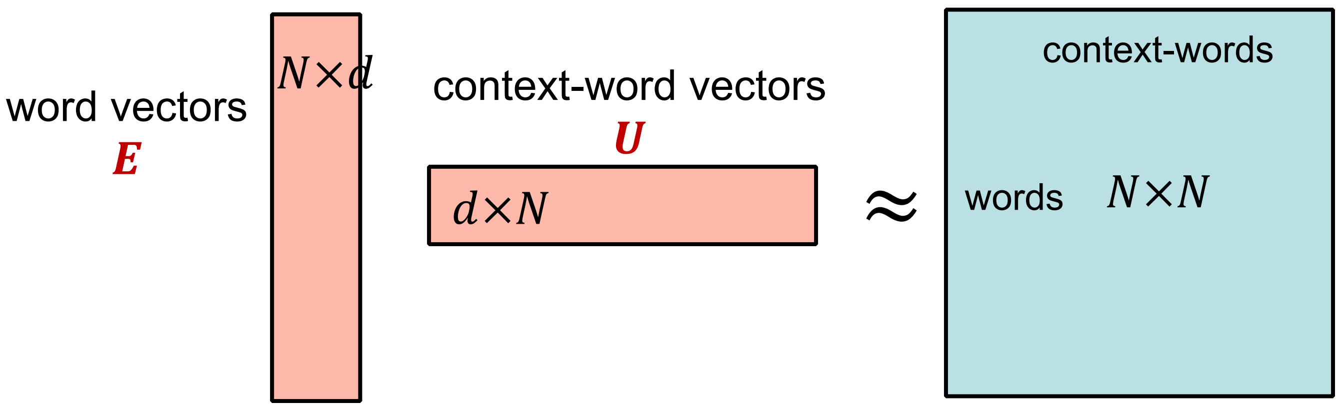
- Negative Sampling is a biased approximation of softmax
  - Noisy Contrastive Estimation (the parent of Negative Sampling) is an unbiased approximation of softmax

Three word embedding models in one frame!

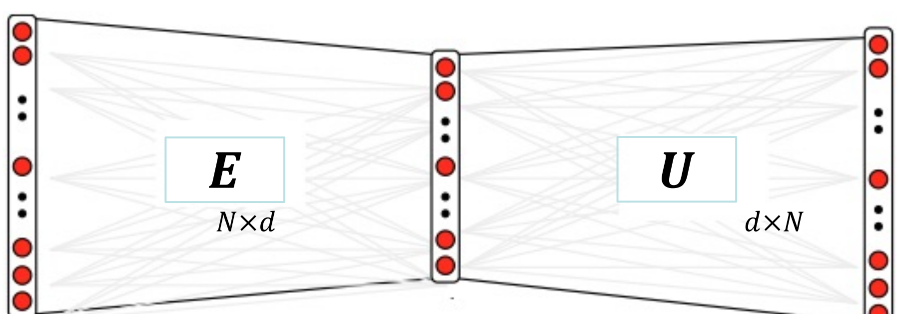
PPMI+SVD:



GloVe:



word2vec skip-gram:



Agenda

- word2vec
  - Neural skip-gram Language Model
  - Negative sampling
- **fastText**
- Sentence embedding with sent2vec

From words to subwords embeddings

- word2vec and the other word embeddings so far define one vector representation for every word in the defined dictionary

- However, words with low frequencies naturally observe a small number of contexts, and therefore most probably end up with *weaker* semantic representations
  - For example, the word "***structure***" will probably have a *better* representation than a word like "***structurally***" which typically appears less frequently in corpora

- The discussed word embeddings also do not have a principled way to approach out-of-vocabularies (OOV)

- One way to approach these limitations is by using subwords

Subwords embeddings

Principle idea of subwords embeddings

- Using the statistics of the corpus, create a dictionary of subwords
- Assign an embedding to each subword
- Given a word, first break it into its subwords
- Compose the embedding of the word from the embeddings of its subwords

Pros:

- Subword embeddings may provide better word embeddings due to a better generalization, particularly when a word lacks sufficient training data
  - Inferring the embedding of "***structurally***" from "***structur***", "***al***", and "***ly***"
- OOVs also have embeddings, composed from their subwords

Cons:

- Composing words from subwords may lead to some errors and ambiguities
  - E.g., unseen named entities (like the name of a city) are also provided with the semantic vector, composed from its subwords. This may imply wrong semantic relations

fastText – subwords

- fastText defines the set of subwords of a word as the *n*-gram characters of the word
  - Start and end of the word are indicated with < and >
  - The word itself is also added to the set of subwords of the word
  - 3-gram is used in practice

Examples based on 3-gram characters:

- Word v:

  where

- $\mathbb{G}_v$ – the set of subwords of v:

  {<wh, whe, her, ere, re>, <where>}

- Word v:

  Highest

- $\mathbb{G}_v$:

  {<hi, hig, igh, ghe, hes, est, st>, <highest>}

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135-146. https://aclanthology.org/Q17-1010.pdf

fastText – formulation

- Process the corpus to create the dictionary of subwords
  - The dictionary consists 3-gram characters plus the words themselves
- Create subword encoder embeddings $\boldsymbol{E}$ for all the subwords
- The encoder embedding of a word is calculated as the sum of its encoder subword embeddings:

$$\boldsymbol{e}_v = \sum_{x \in \mathbb{G}_v} \boldsymbol{e}_x$$

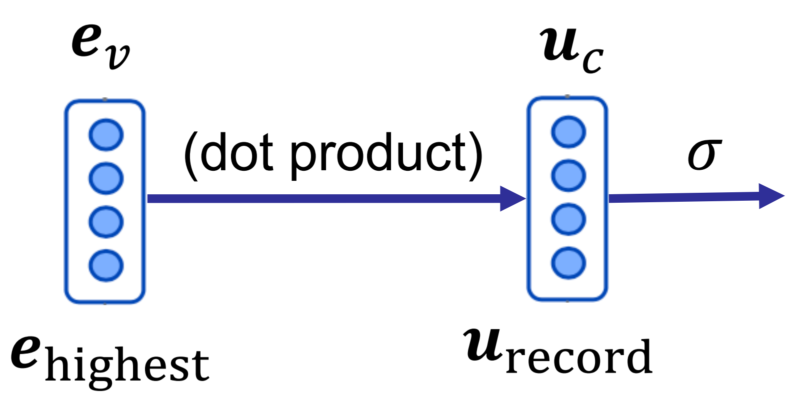
- Decoder word embeddings $\boldsymbol{U}$ remain the same as word2vec, namely a set embeddings for the words in the corpus
- Model training is also the same way as word2vec using Negative Sampling

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135-146. https://aclanthology.org/Q17-1010.pdf

word2vec skip-gram – recall

Training data:

$(v = \texttt{highest}\,, c = \texttt{record})$



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

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

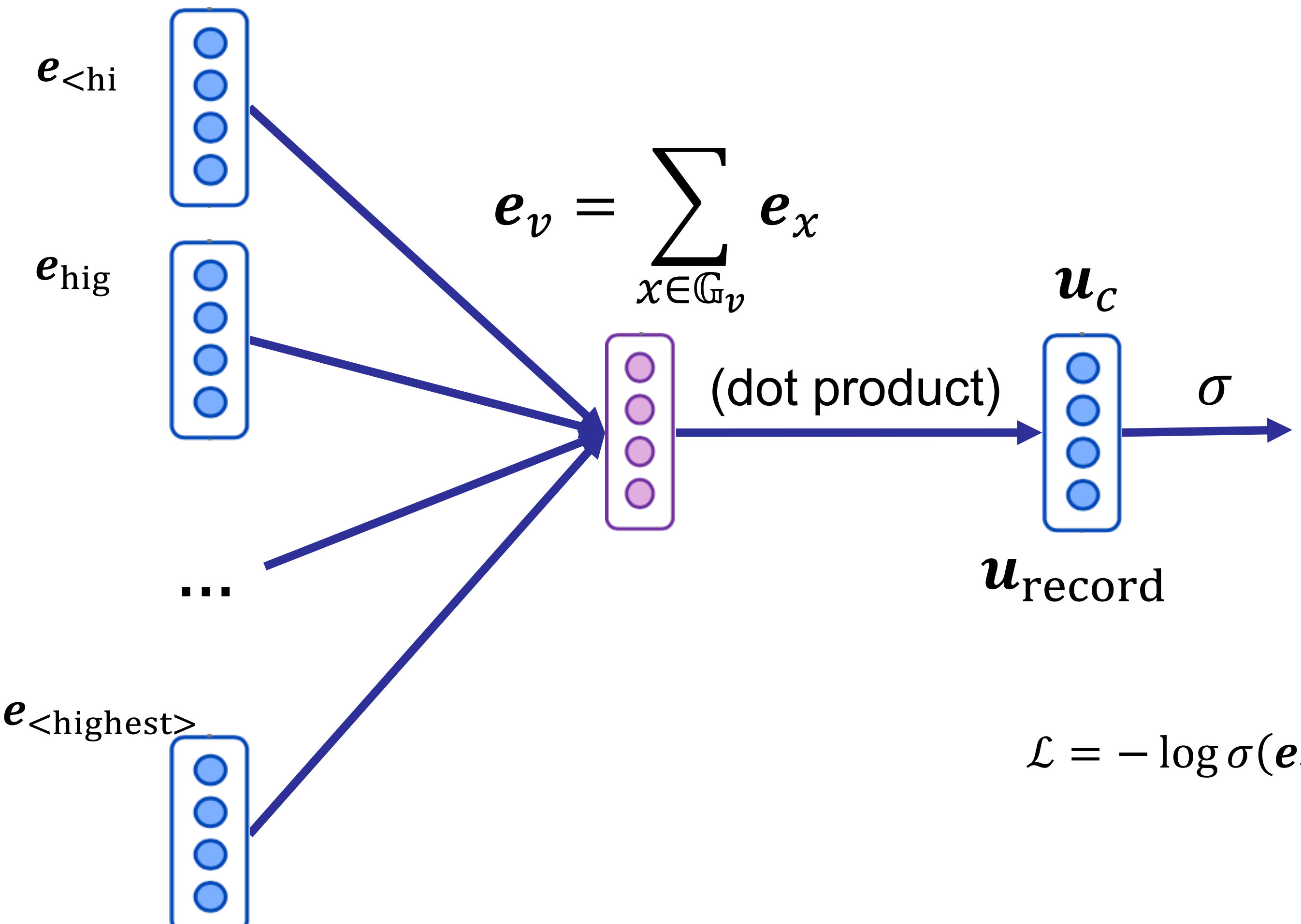
fastText – architecture

Training data:

$(v = \texttt{highest}\,,\, c = \texttt{record})$

$(\mathbb{G}_v = \{\texttt{<hi}, \texttt{hig}, \texttt{igh}, \texttt{ghe}, \texttt{hes}, \texttt{est}, \texttt{st>}, \texttt{<highest>}\}\,,\, c = \texttt{record})$

Embeddings of the subwords in $\mathbb{G}_v$

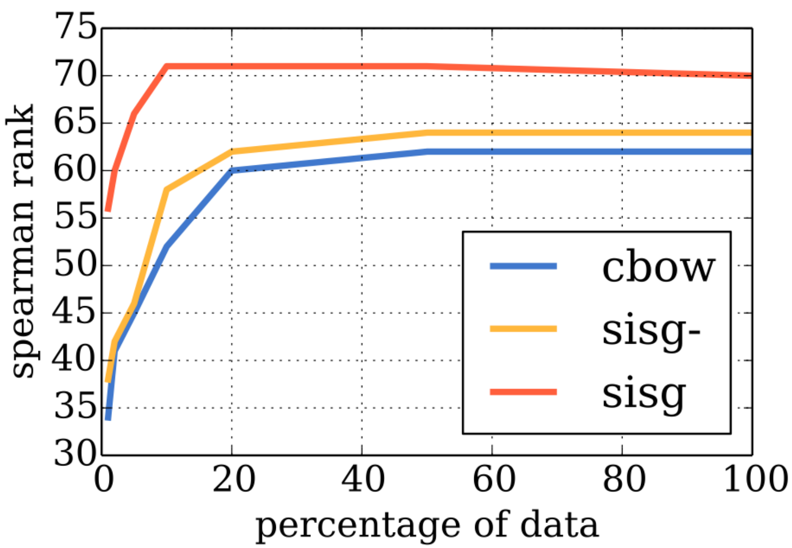


$P(y = 1 | v, c) = \sigma(\boldsymbol{e}_v \boldsymbol{u}_c)$

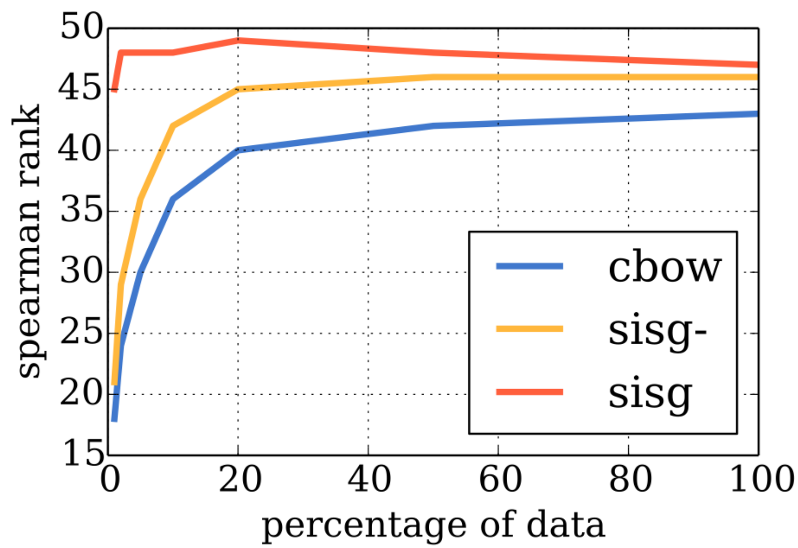
$$\mathcal{L} = -\log \sigma(\boldsymbol{e}_v \boldsymbol{u}_c) - \sum_{\substack{\tilde{c} \sim \widetilde{\mathcal{D}} \\ k \text{ times}}} \log \sigma(\boldsymbol{e}_v \boldsymbol{u}_{\tilde{c}})$$

Better generalization with fastText

- In comparison with word2vec, fastText …
  - generalizes faster in training
  - generally provides better embeddings



(a) DE-GUR350



(b) EN-RW

See details:
Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135-146. https://aclanthology.org/Q17-1010.pdf

Agenda

- word2vec
  - Neural skip-gram Language Model
  - Negative sampling
- fastText
- **Sentence embedding with sent2vec**

Sentence embedding

Problem definition

- Given a "sentence" S with length $|S|$, consisting of the words

$$v_1, v_2, \dots, v_{|S|}$$

with corresponding word vectors

$$\boldsymbol{e}_{v_1}, \boldsymbol{e}_{v_2}, \dots, \boldsymbol{e}_{v_{|S|}}$$

create the sentence embedding: $\boldsymbol{e}_S$

- "Sentence" here can refer to
  - Any sequence of words with any arbitrary length
  - An actual sentence in language

Sentence embedding

- First approach … simply average!

$$\boldsymbol{e}_S = \frac{1}{|S|} \sum_{v \in S} \boldsymbol{e}_v$$

  - As done in Assignment 2 and 3

- What are the possible limitations of this approach?
  - The word embeddings are not trained for the purpose of creating a sentence embeddings

sent2vec

- A simple and efficient method for creating sentence representations

- sent2vec starts from <u>subword/word embeddings</u> and calculates a sentence embedding as the <u>average</u> of subword/word embeddings:

$$\boldsymbol{e}_S = \frac{1}{|S|}\sum_{v \in S} \boldsymbol{e}_v$$

- sent2vec trains subword/word embeddings ($\boldsymbol{E}$) in the way that they fulfill the objective of creating effective <u>sentence embeddings</u>

Pagliardini, Matteo, Prakhar Gupta, and Martin Jaggi. "Unsupervised Learning of Sentence Embeddings Using Compositional n-Gram Features." *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2018.

Training

- Parameters of sent2vec – similar to fastText/word2vec – consists of subword/word embeddings ($\boldsymbol{E}$) and context-word embeddings ($\boldsymbol{U}$)

- Given a sentence S, a training data point is defined as the pair of:

(set of subwords in S while putting out the word v , left-out word v)

$$(S\backslash\{v\}\,,v)$$

- During training, $\boldsymbol{e}_{S\backslash\{v\}}$, the sentence embedding without the left-out word, aims to predict the left-out word v

- The optimization is done with Negative Sampling

Training data

- Training data is in the form of $(S \backslash \{v\}, v)$

$S =$ `Tarahumara people drink Tesgüino during the rituals`

<u>Some training data points in $\mathcal{D}$:</u>

(`people drink Tesgüino during the rituals`, `Tarahumara`)
(`Tarahumara drink Tesgüino during the rituals`, `people`)
(`Tarahumara people Tesgüino during the rituals`, `drink`)
(`Tarahumara people drink during the rituals`, `Tesgüino`)
(`Tarahumara people drink Tesgüino the rituals`, `during`)
…

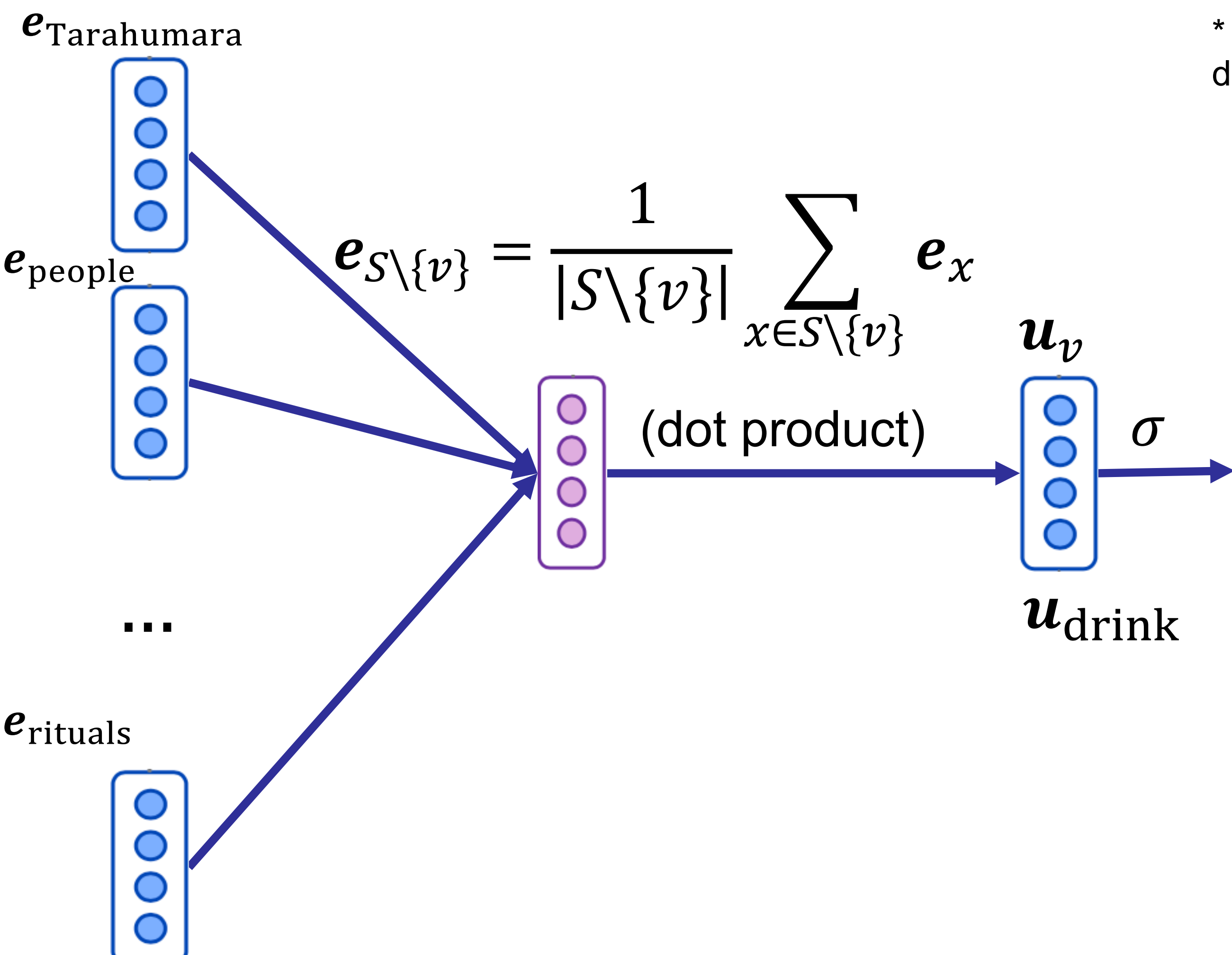
Architecture

Training data:

$(S \backslash \{v\} =$ `Tarahumara people Tesgüino during the rituals` $, v =$ `drink`$)$

$(S \backslash \{v\} = \{$`Tarahumara,people,Tesgüino,during,the,rituals`$\}^*, v =$ `drink`$)$

* Encoder tokenization can also be done in subword level as in fastText



$$\boldsymbol{e}_{S\backslash\{v\}} = \frac{1}{|S\backslash\{v\}|} \sum_{x \in S\backslash\{v\}} \boldsymbol{e}_x$$

$$P(y = 1 | S\backslash\{v\}, v) = \sigma(\boldsymbol{e}_{S\backslash\{v\}} \boldsymbol{u}_v)$$

Negative Sampling loss

- Negative Sampling loss
  - increases $P(y = 1|S\setminus\{v\}, v)$ probability for positive sample $(S\setminus\{v\}, v)$
  - decreases $P(y = 1|S\setminus\{v\}, \tilde{v})$ probability for k negative samples $(S\setminus\{v\}, \tilde{v})$

- Loss function:

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

positive sample

k negative samples