Statistical Natural Language Processing [COMP0087]

Manual feature engineering
Linear models and classification

Vasileios Lampos

Computer Science, UCL

lampos.net

About me

- Associate Professor at CS
- Vasileios 🇬🇷 or Bill 🇬🇧
- Website: lampos.net
- Research in ML / NLP methods for health
- Publications: scholar.google.com/citations?user=eXDONDEAAAAJ
- Tweets @ twitter.com/lampos
- 1.09D @ 90 High Holborn (UCL Centre for AI) / Meeting by appointment

About this lecture

- In this lecture:
  - Manual feature engineering for NLP applications
  - Introductory insights about supervised learning (*classification*)
  - A few introductory remarks about word representation in NLP

- **Reading:** Chapters 2 and 5 of "*Speech and Language Processing*" (SLP) by Jurafsky and Martin (2023) — web.stanford.edu/~jurafsky/slp3/

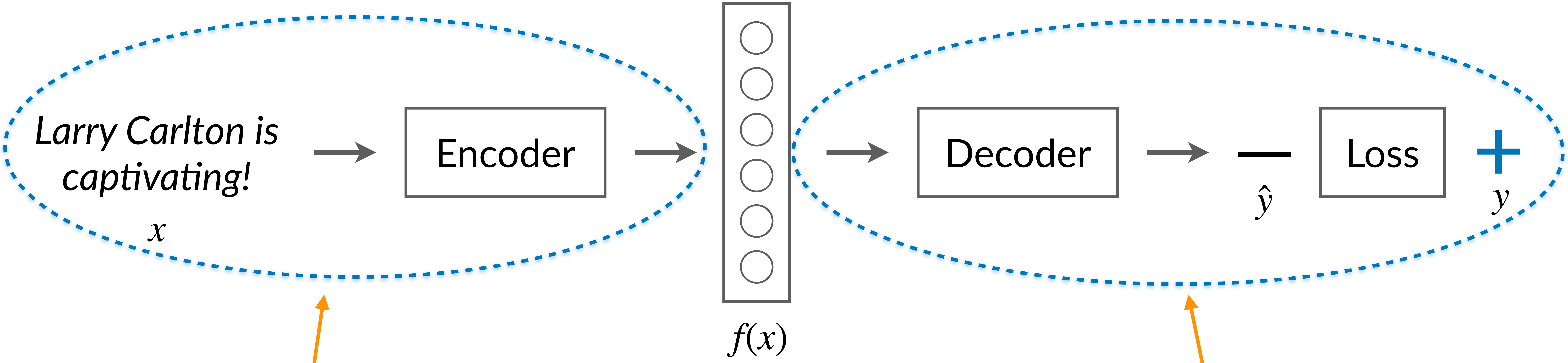
- **Acknowledgements:** Based on prior material from Pontus Stenetorp

Sentiment analysis as our NLP task paradigm

- A popular task / downstream NLP application

- *"A Clockwork Orange" is a cinematic masterpiece.* → +
- *No, I don't think this was Emma Stone's best performance, but overall it was still a decent one!* → +
- *Maybe I am too old, but I find any reference to "AI Music" quite irritating and aesthetically displeasing.* → −

The NLP view (*for today*)

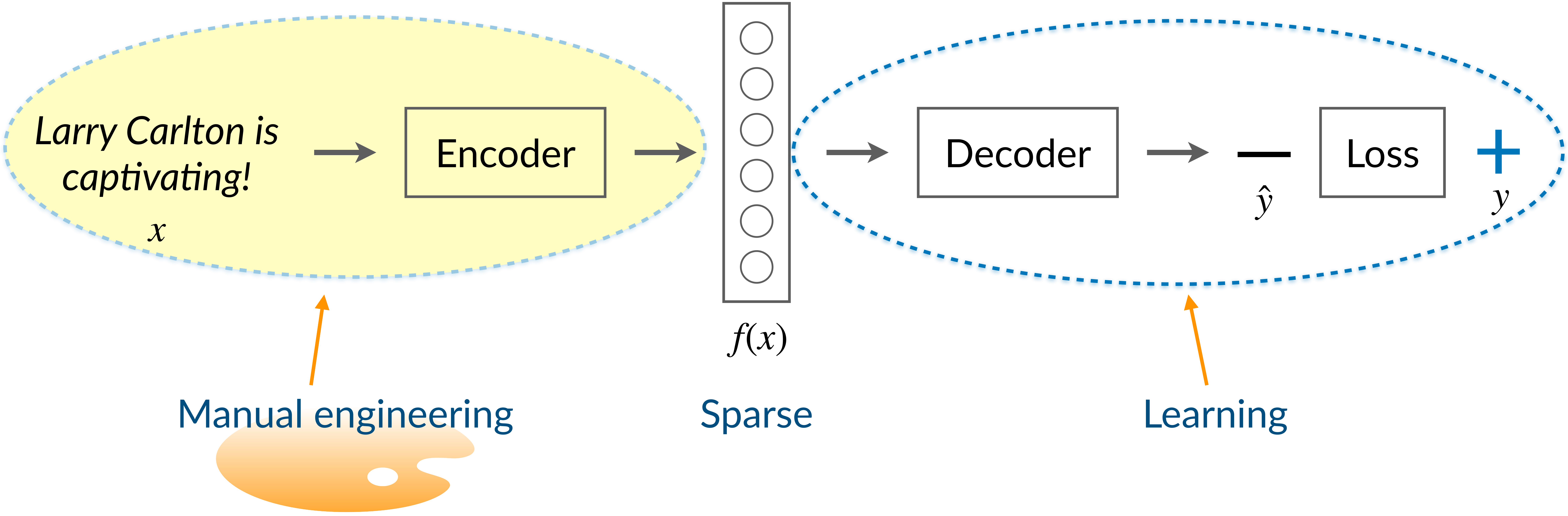


Manual engineering

Sparse

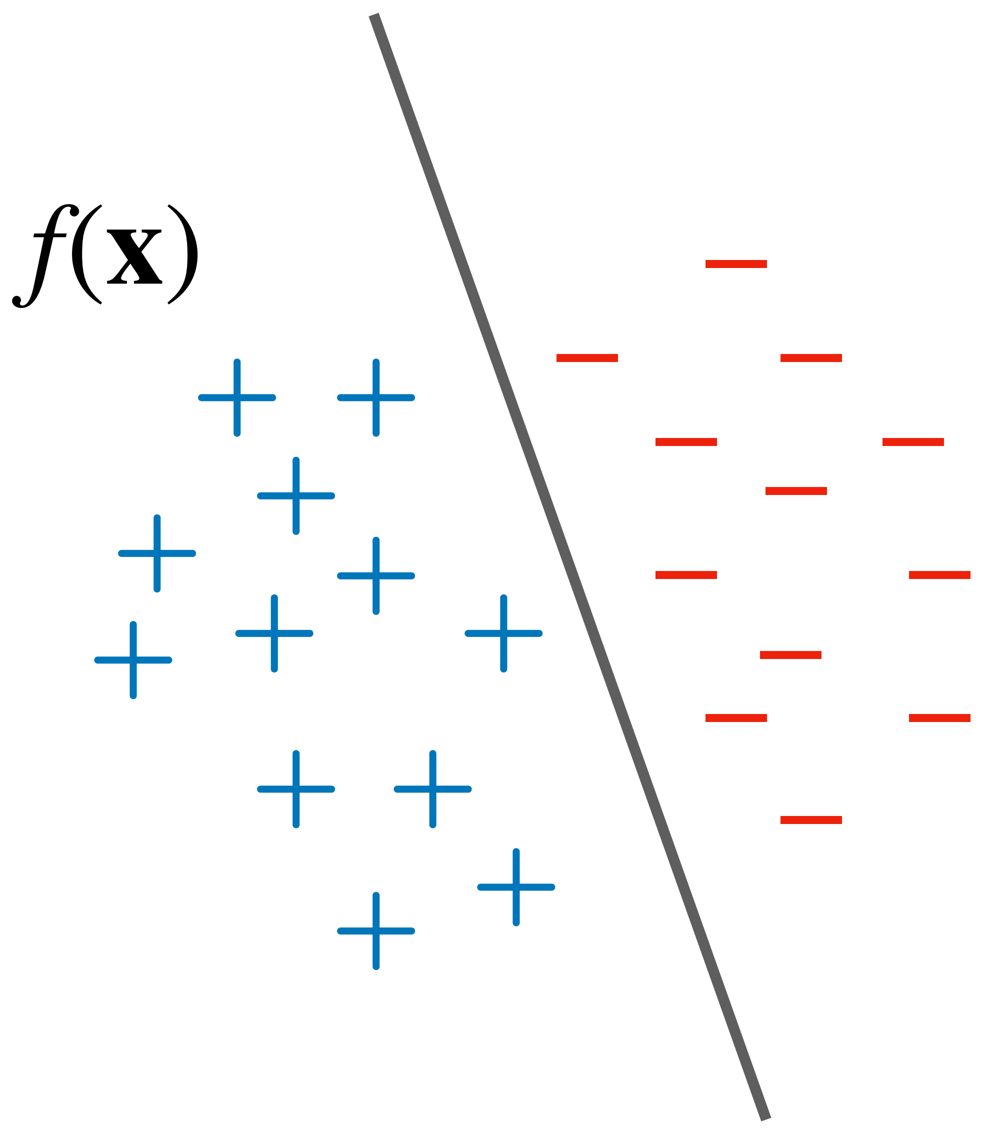
Learning

The NLP view (*for today*)



Data representation matters

- Machine learning methods become simpler when data representations are good
- But what is a “good” data representation?
  - Accurate / correct (*trivial if we take measurements, not trivial when we abstract*)
  - Good choice for a specific modelling task
- Then again, if it was always possible to obtain or have great data representations, advanced machine learning methods would not have been necessary
- More on some fundamental aspects of data representation in NLP later in this lecture!



Tokenisation

- A machine sees a string as a sequence of characters — no sense of “words”

In my rearview mirror, the sun is going down.

In ␣ my ␣ rearview ␣ mirror ␣ , ␣ the ␣ sun ␣ is ␣ going ␣ down ␣ .

Of course, mama’s gonna help build the wall!

Of ␣ course ␣ , ␣ mama ␣ ’s ␣ gonna ␣ help ␣ build ␣ the ␣ wall ␣ !

- Break up string into **tokens** ($\neq$ words)
  - Easy for well-structured English (white space plus a few other rules)
  - Not easy for some languages (e.g. Chinese, Japanese)
  - Not necessarily easy for unstructured (e.g. social media) or domain-specific (e.g. scientific) text

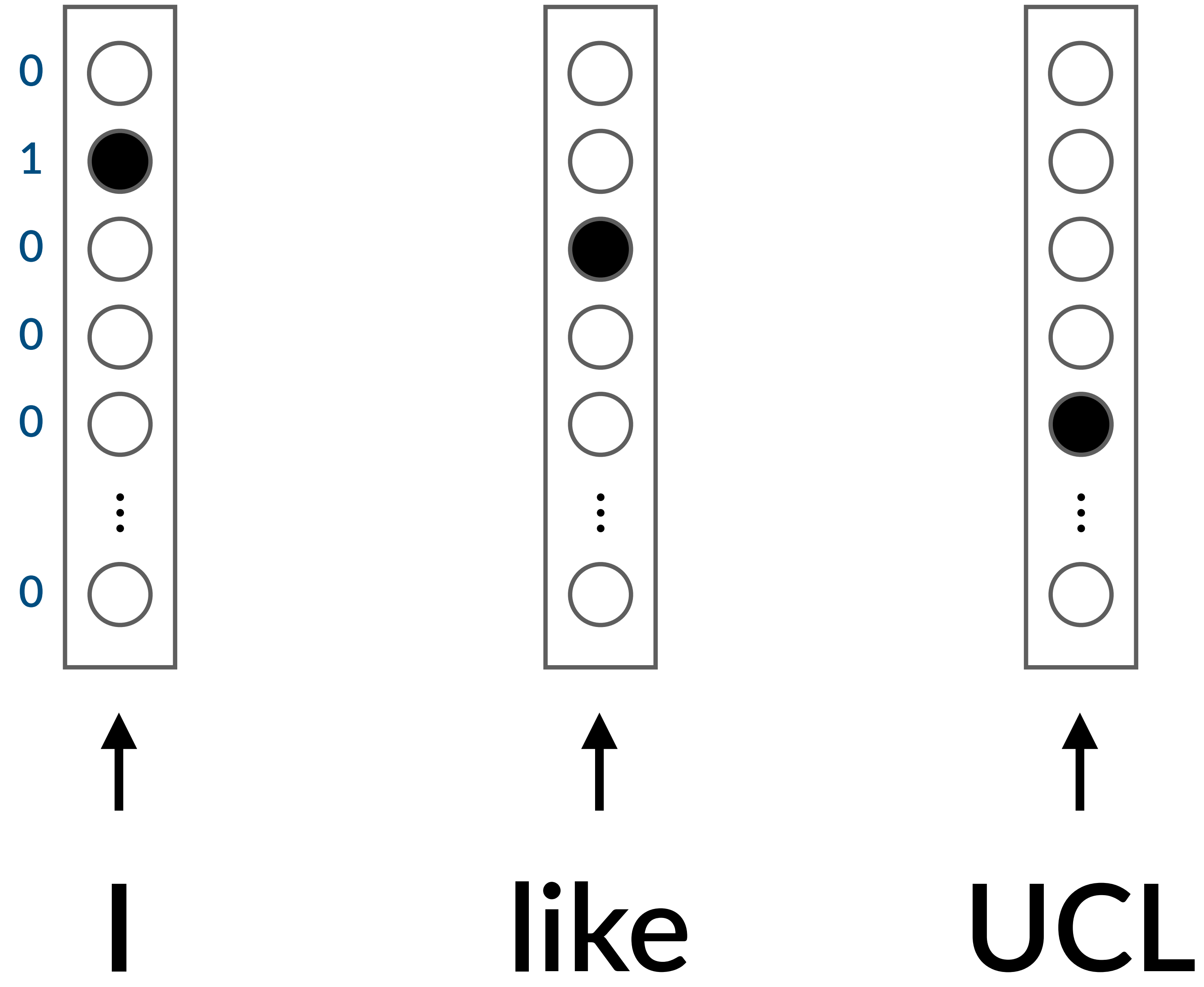
Tokenisation of English — *More challenging examples*

@RandomTwitterUser: Its another day of the week.also my bday! Feel xhausted !**@Helen0001781**, are U there?#goodMorningEveryone

The first example was the initial preparation of α,ω-diazido-terminated polystyrene-***b***-poly(ethylene oxide)-***b***-polystyrene followed by coupling with dipropargyl ether in dimethylformamide (DMF) in the presence of a CuBr/*N*,*N*,*N′*,*N″*,*N″*-pentamethyldiethylenetriamine catalyst.

From: K. Matyjaszewski, *Adv. Mater.* 2018, 30, 1706441.

Representing words (*tokens*) with one-hot vectors



vocabulary

$\mathcal{V} =$

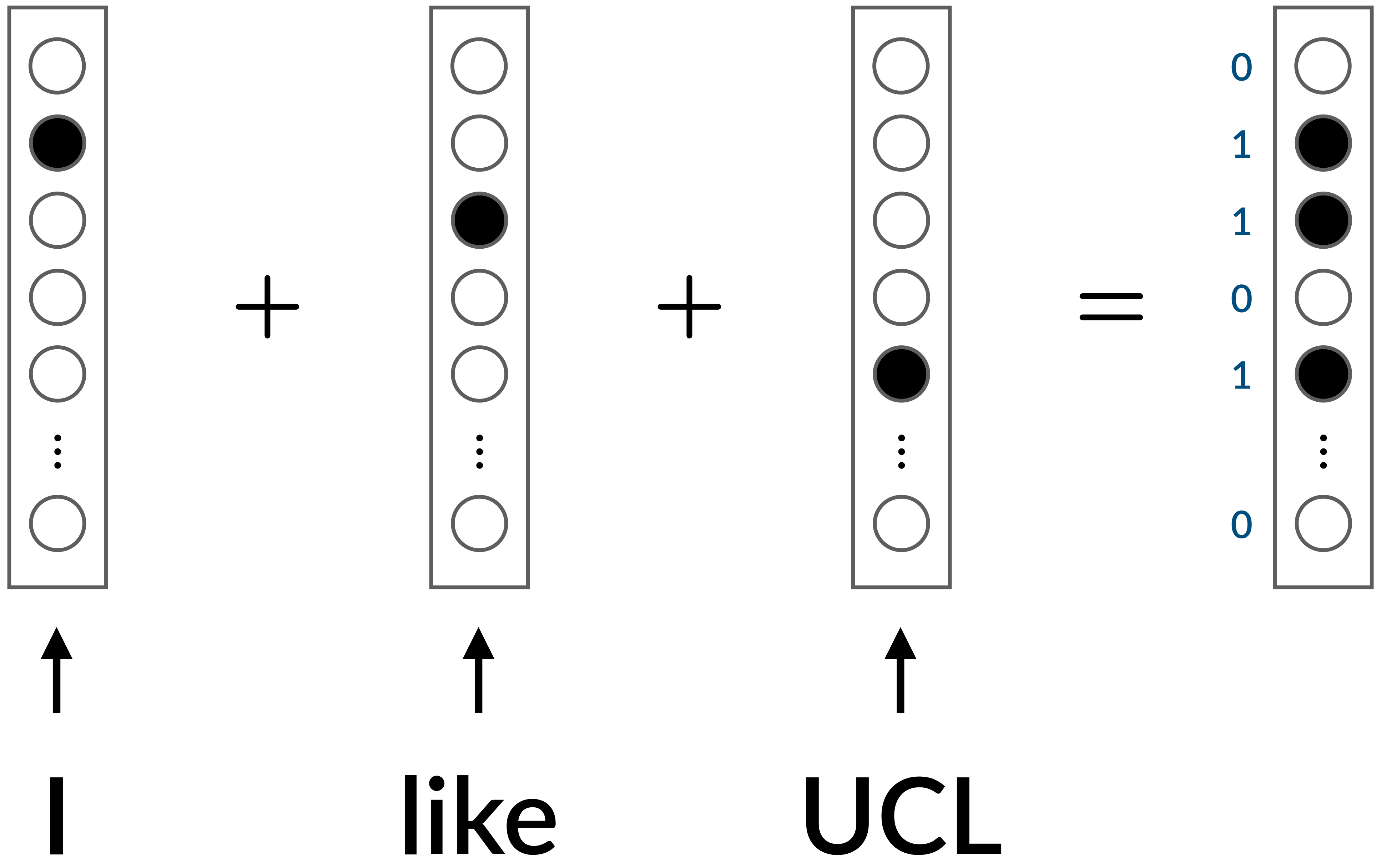
1	a
2	I
3	like
...	...
m	zoo

word (*token*)

$$f(w) \in \mathbb{R}^m, \text{ where } f_i(w) = \begin{cases} 1 & \text{if } w = \mathcal{V}_i \\ 0 & \text{otherwise} \end{cases}$$

Representing sentences with sum pooling

Sum Pooling



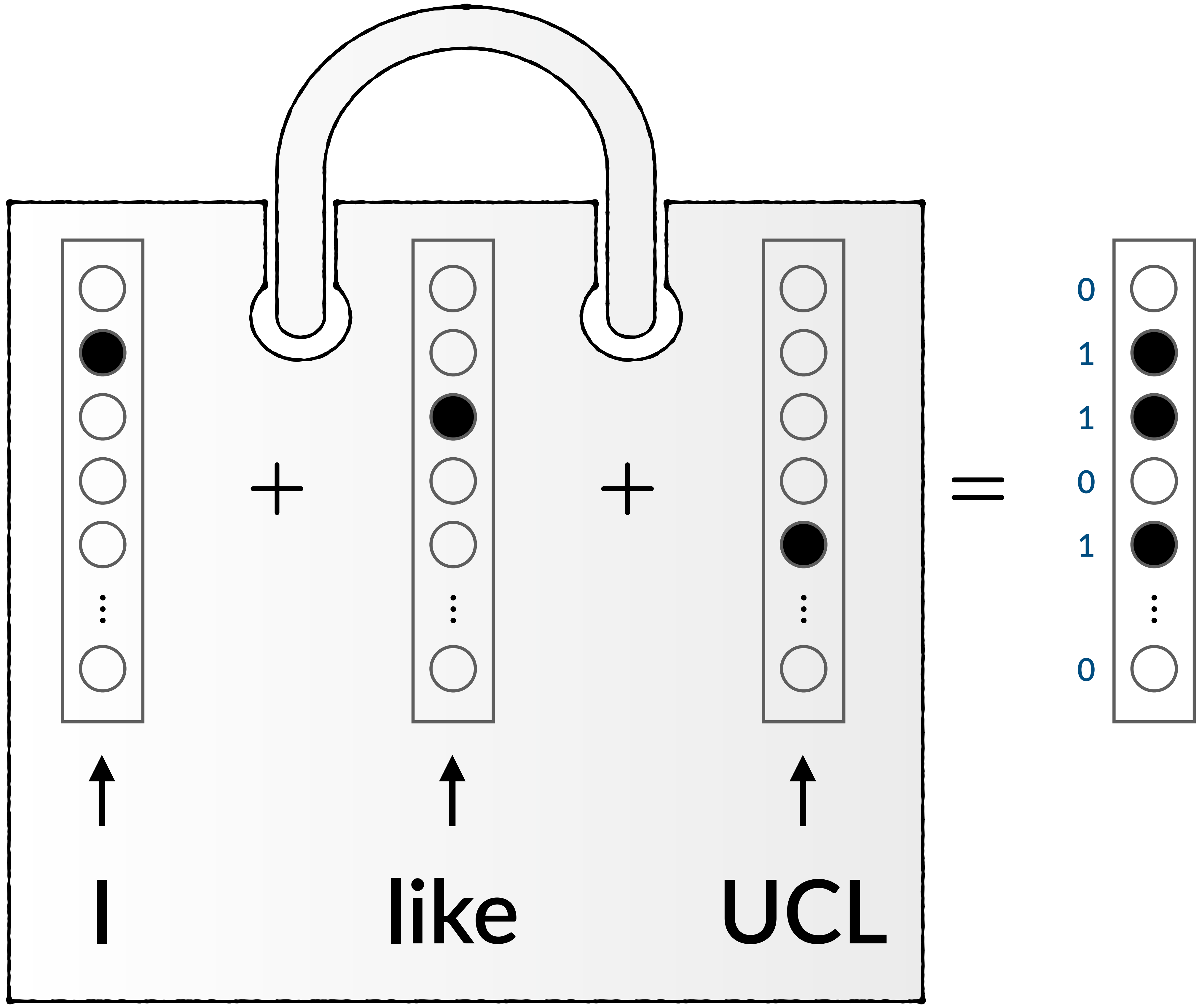
sentence

token representation

$$f(x) \in \mathbb{R}^m, \text{ where } f(x) = \sum_{w \in x} f(w)$$

“*Bag of words*” representation

Sum Pooling



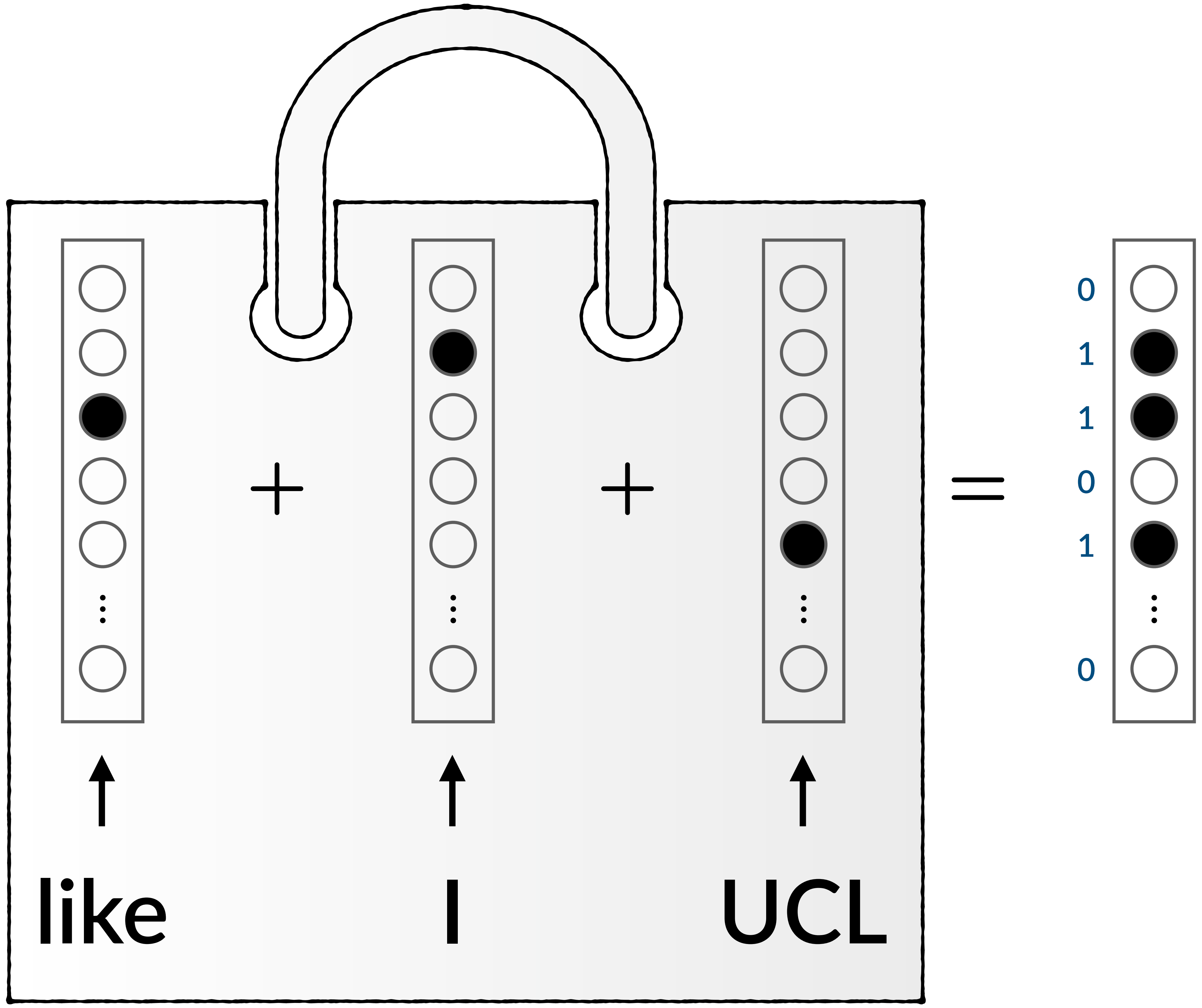
sentence

token representation

$$f(x) \in \mathbb{R}^m, \text{ where } f(x) = \sum_{w \in x} f(w)$$

Sum Pooling

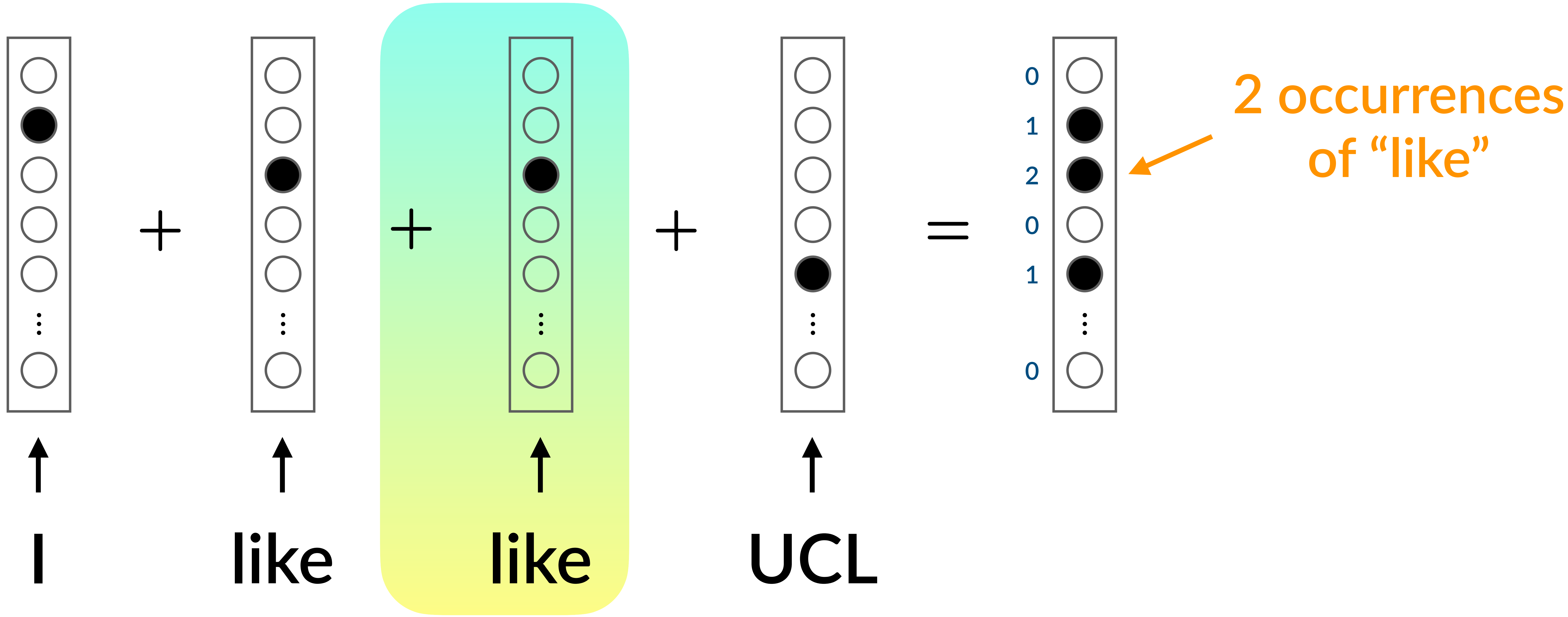
indifferent to token position



$$f(x) \in \mathbb{R}^m, \text{ where } f(x) = \sum_{w \in x} f(w)$$

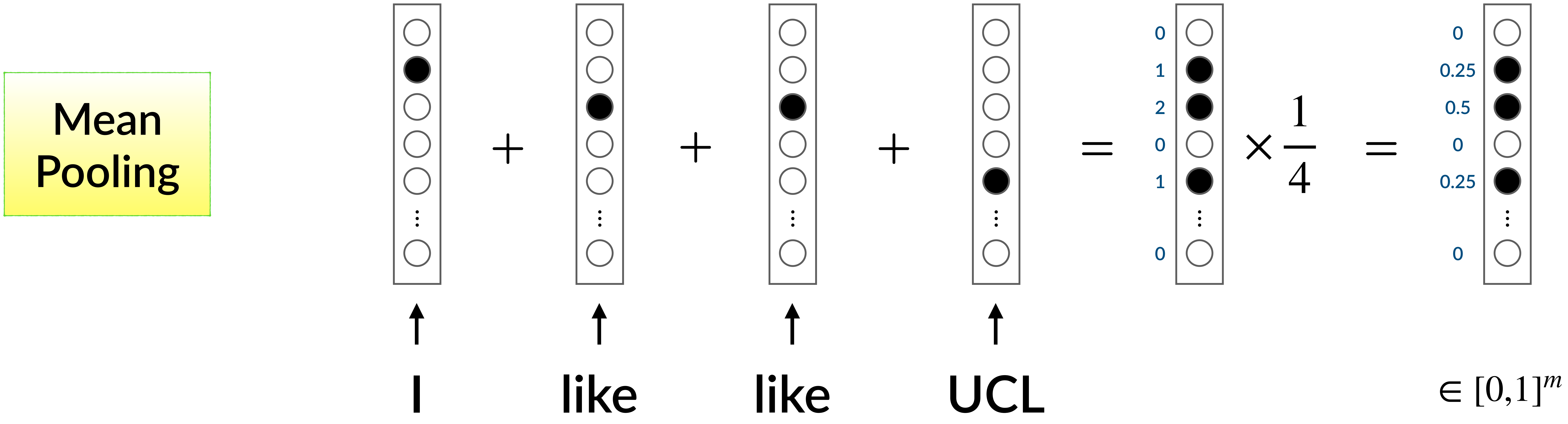
Representing sentences by sum pooling — Aggregation effect

Sum Pooling



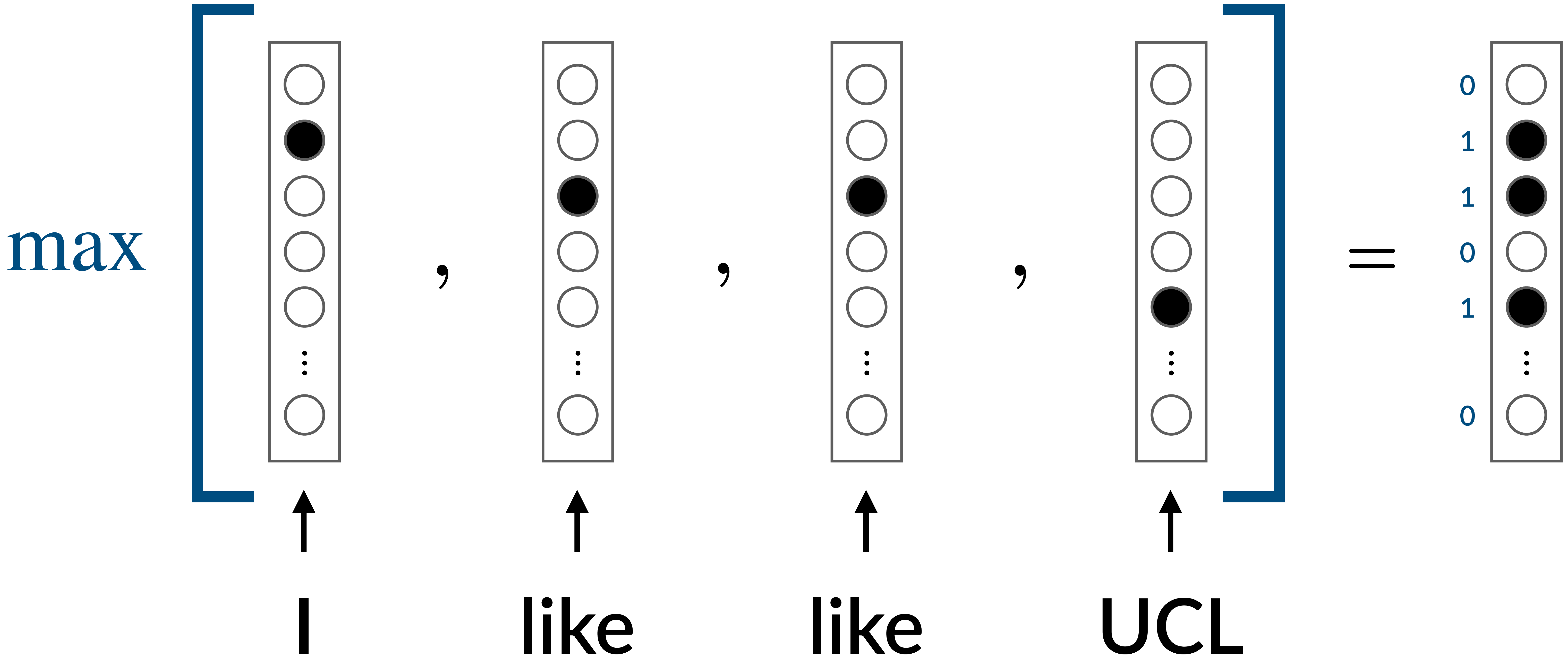
Sum pooling is sensitive to **sentence length**

Representing sentences by mean pooling



Mean pooling *corrects* the sentence length sensitivity of sum pooling

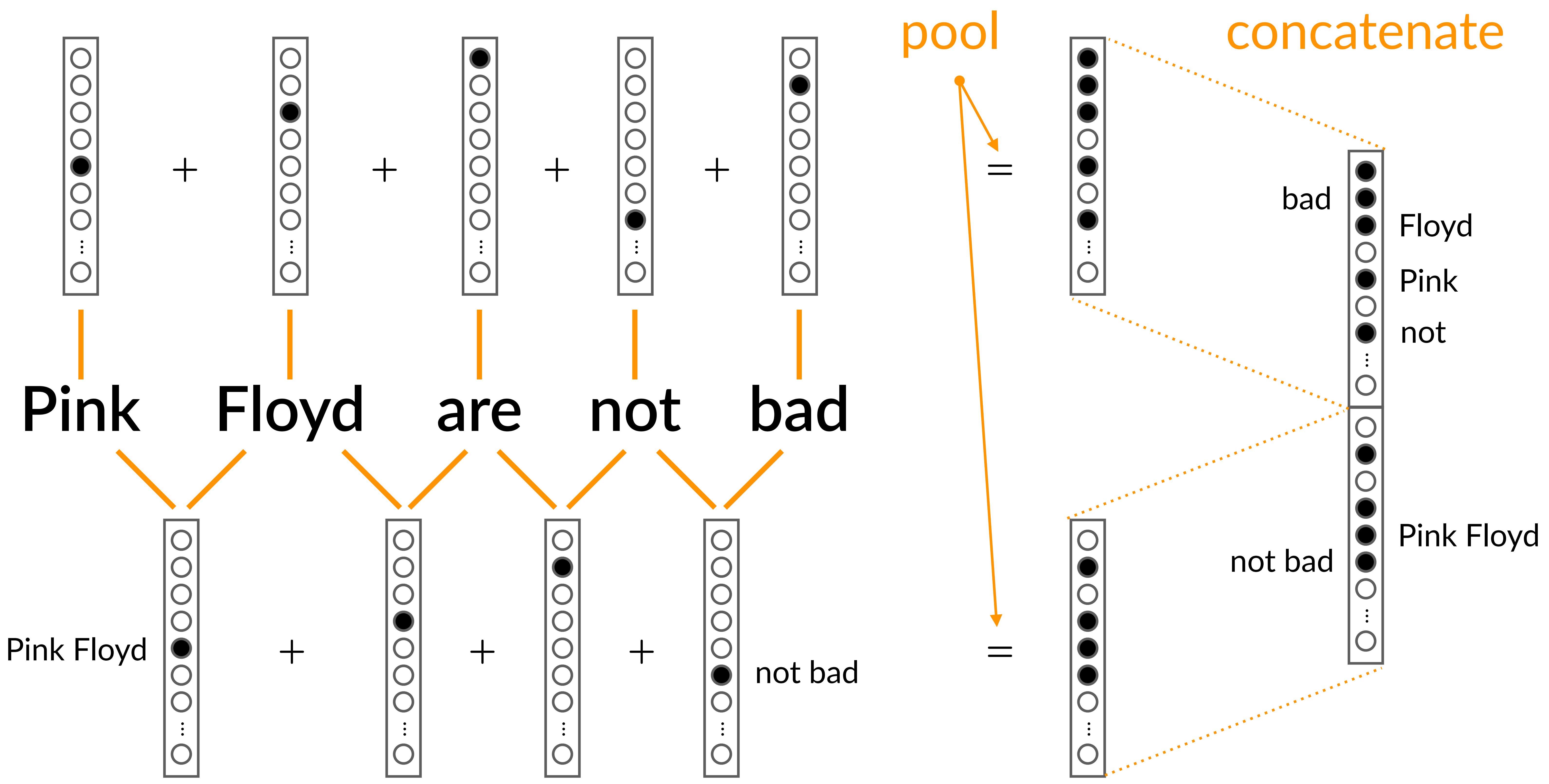
Representing sentences by max pooling



Max Pooling

Max pooling maintains a binary representation

More engineering — Using bi-grams



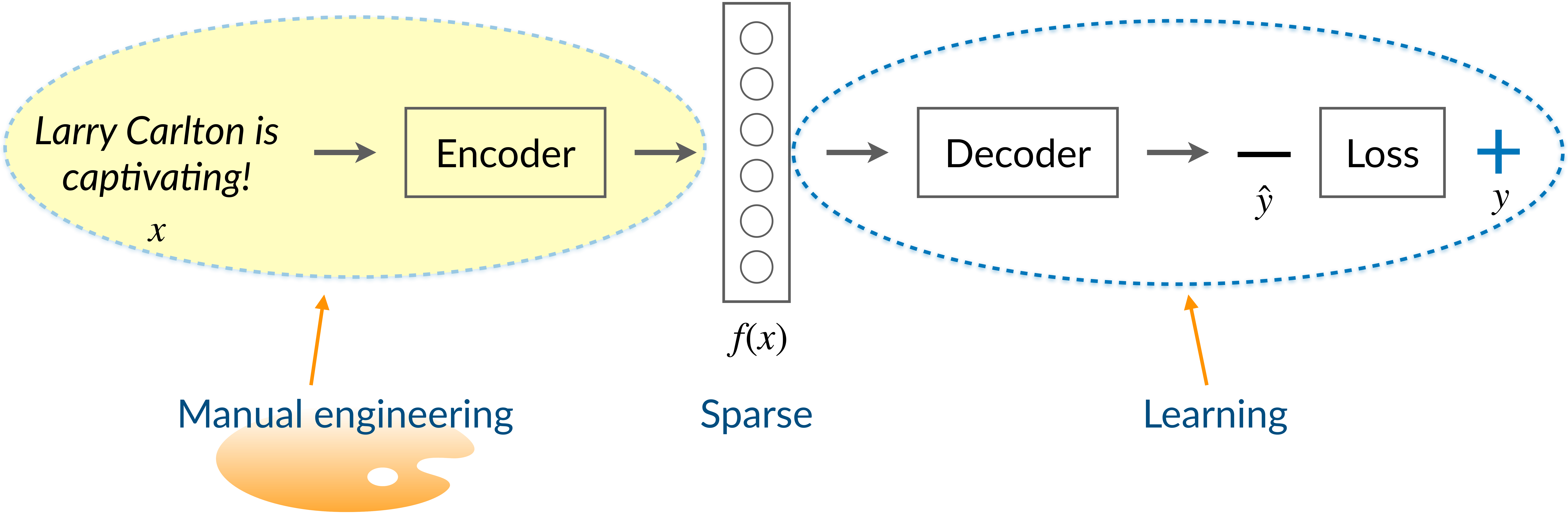
Uni-gram (1-gram) features may not be enough. Engineer more features!
bi-grams (2-grams) may capture more cohesive language patterns.

More (engineering) ideas?

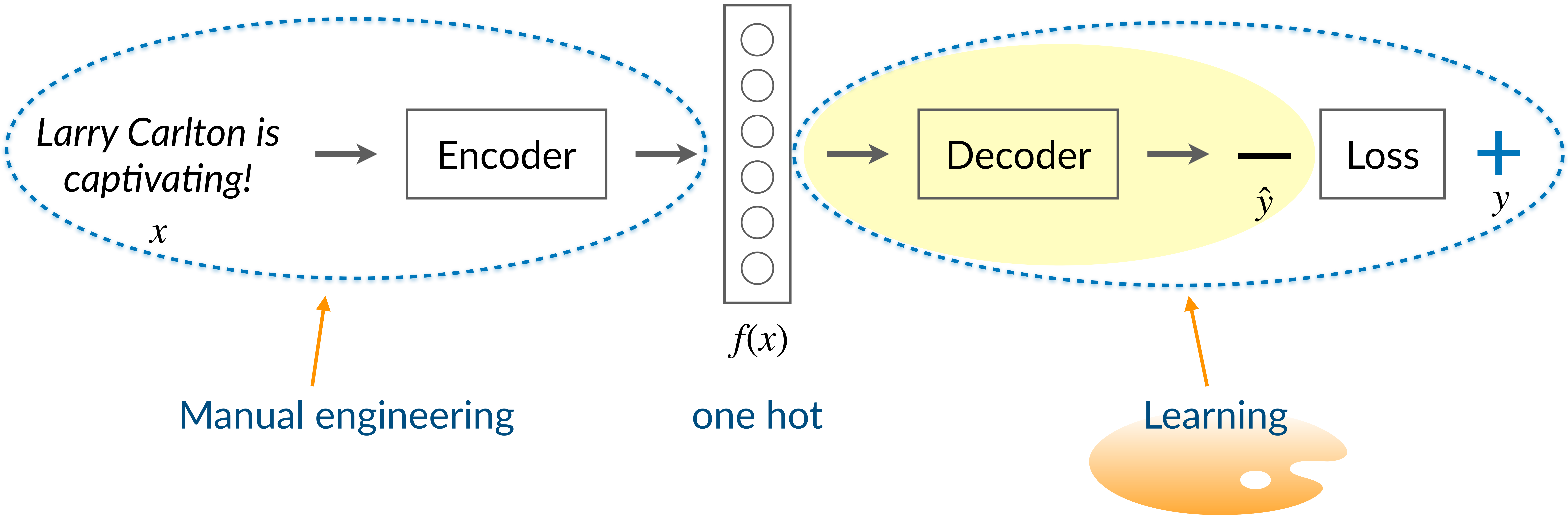
1. Use *dictionaries*?
2. Use *syntax*?
3. *Preprocessing*?

$\vdots$

The NLP view (*for today*)



The NLP view (*for today*)



Linear classification — Obtaining a classification score

For simplicity, let's now use $\mathbf{x} \in \mathbb{R}^m$ to represent $f(x)$

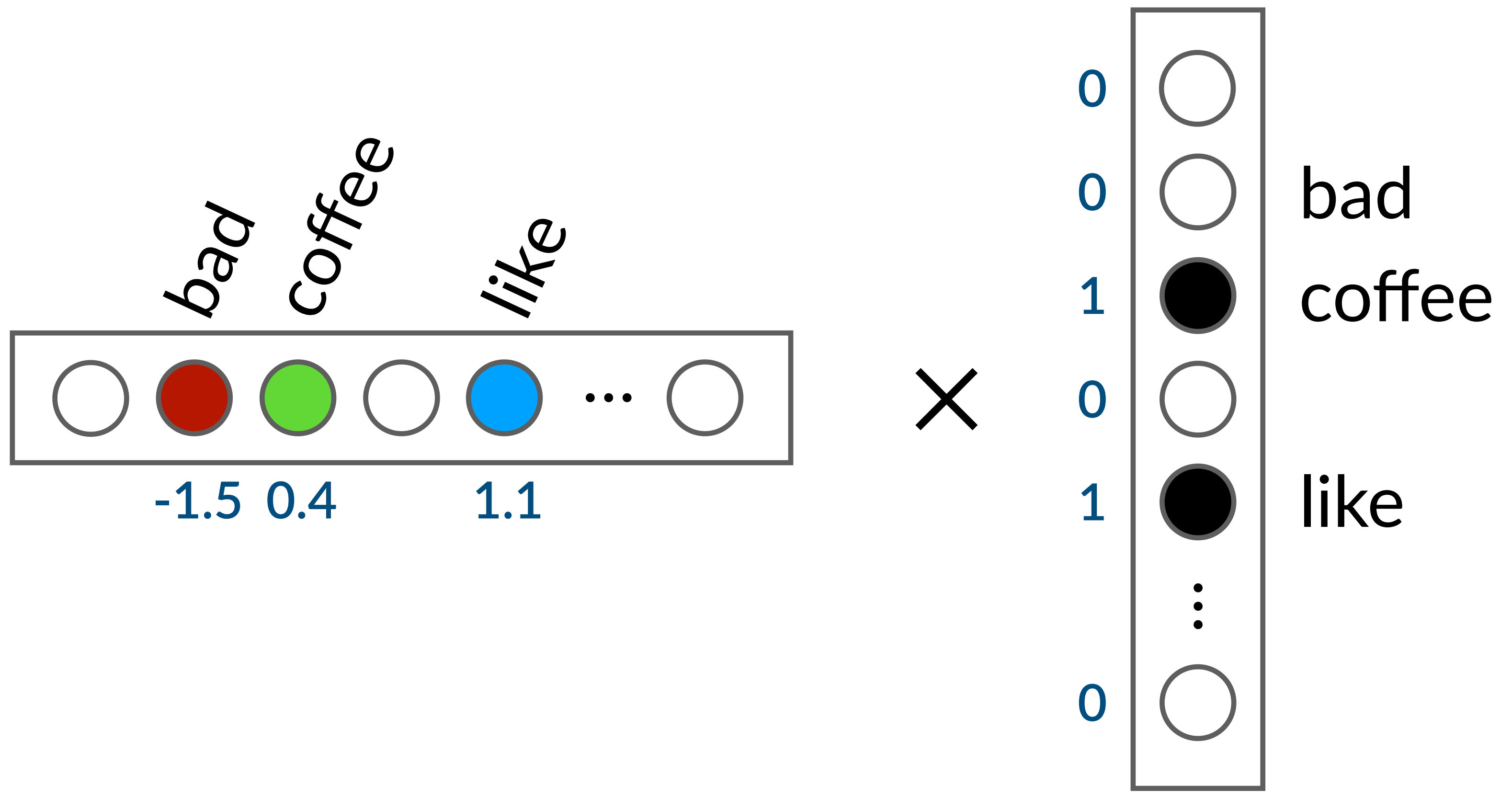
vector space representation for a set of tokens x

$$\mathbf{z}^\top \times \mathbf{x} = s_{\mathbf{z}}(\mathbf{x})$$

weights

representation

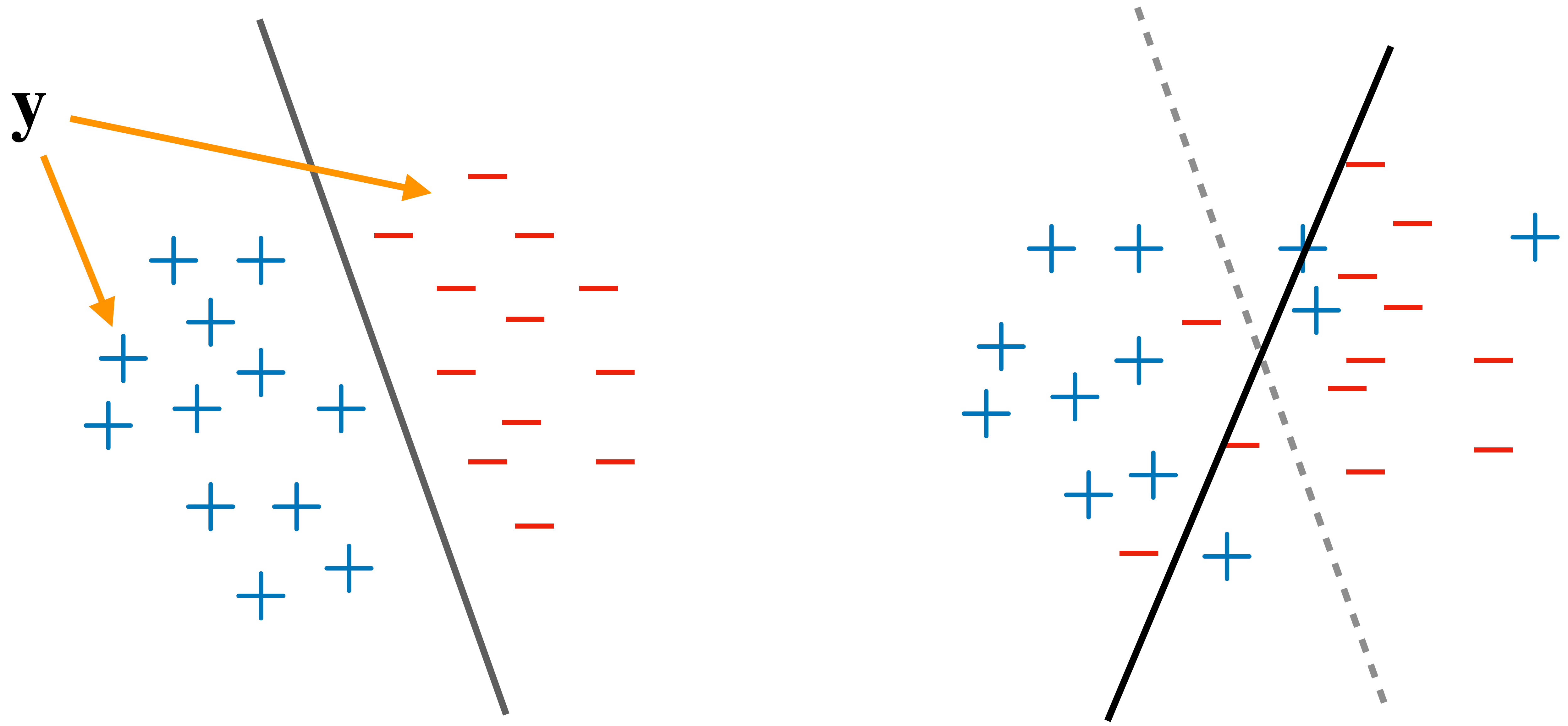
classification score, a.k.a. $\hat{y}$



$$= 0.4 \times 1 + 1.1 \times 1 = 1.5$$

classification outcome???

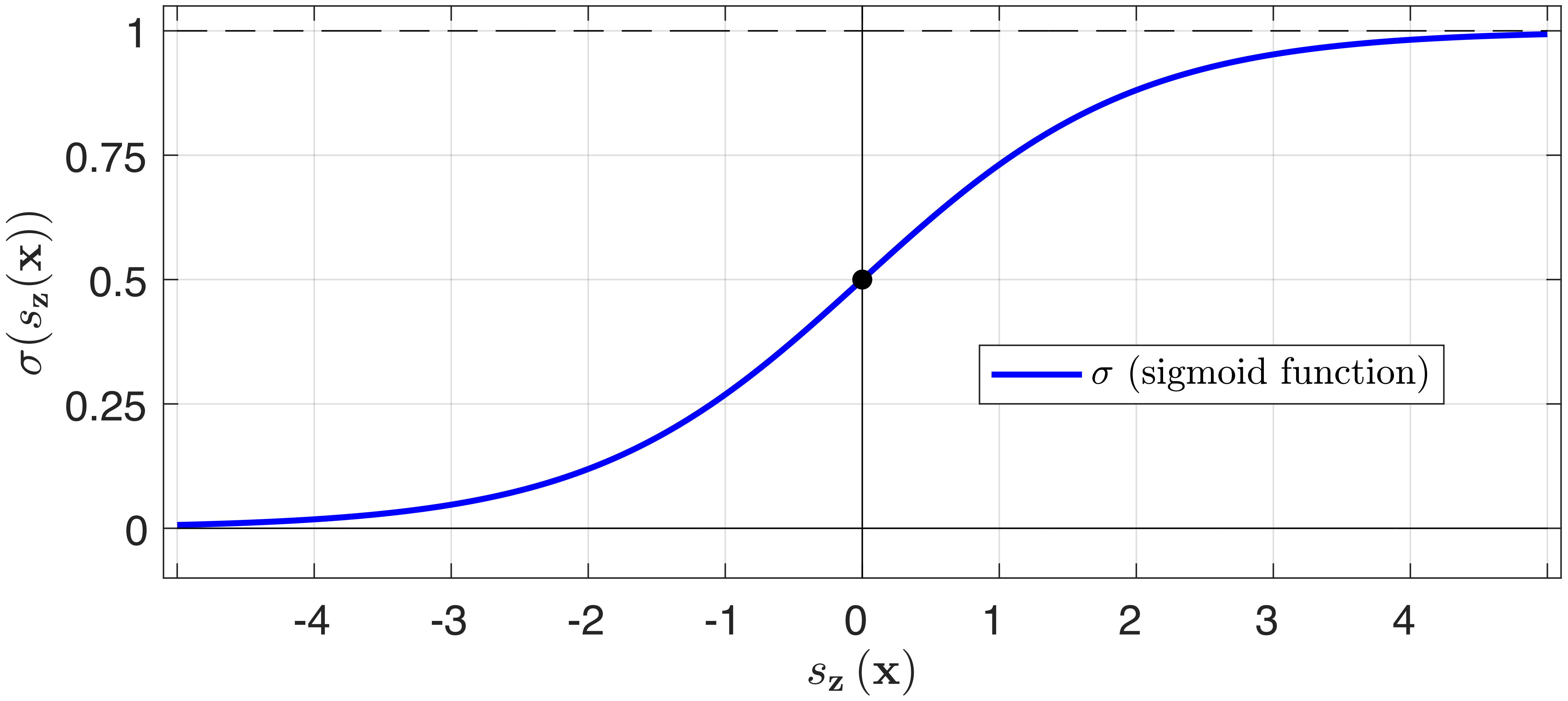
Classification decision boundary



We are "*learning*" a decision boundary that separates positives from negative examples.

If the range of scores is bounded, e.g. from [-1 to 1],
we may think a good boundary choice is 0. No learning!
However, on most occasions this is a sub-optimal decision.

Assign *pseudo*-probabilities to classes



$$p_{\mathbf{z}}\left(y=+\,|\,\mathbf{x}\right)=\sigma\left(s_{\mathbf{z}}(\mathbf{x})\right)=\frac{1}{1+e^{-s_{\mathbf{z}}(\mathbf{x})}}$$

$$p_{\mathbf{z}}\left(y=-\,|\,\mathbf{x}\right)=1-p_{\mathbf{z}}\left(y=+\,|\,\mathbf{x}\right)$$

Classification — *Search / final classification outcome*

Choose the label with the highest probability / score

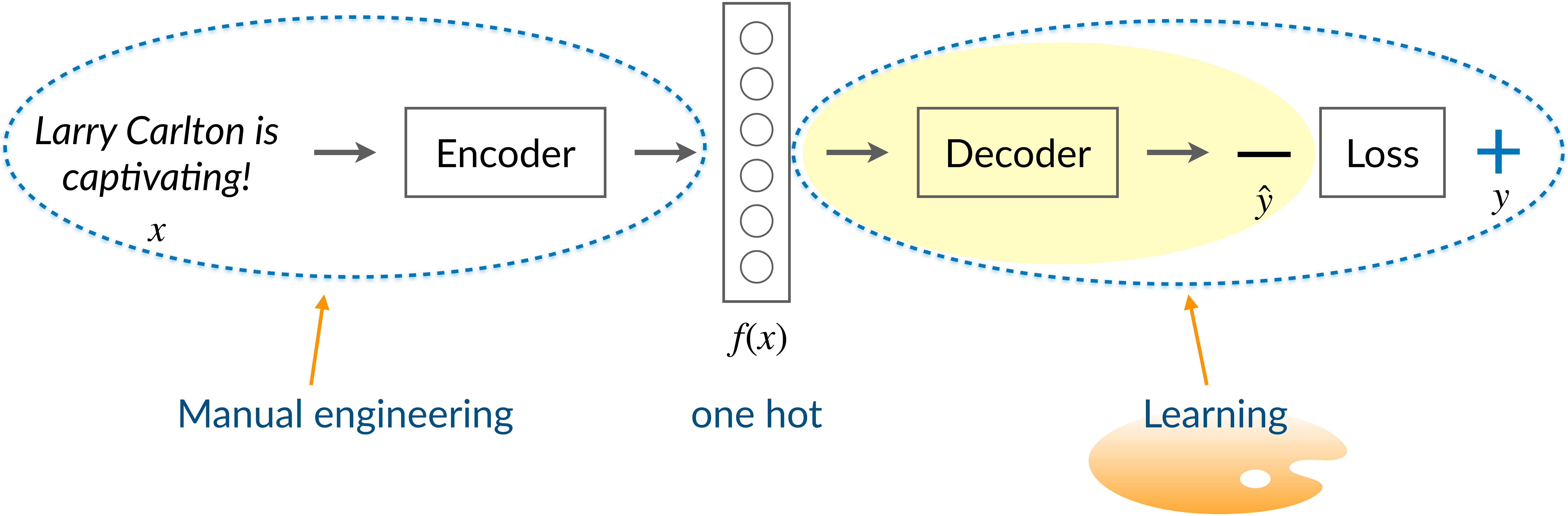
Trivial for binary classification (2 classes):

1. Calculate $p_{\mathbf{z}}\left(+ \mid \mathbf{x}\right)$
2. Calculate $p_{\mathbf{z}}\left(- \mid \mathbf{x}\right)$
3. Choose highest one!

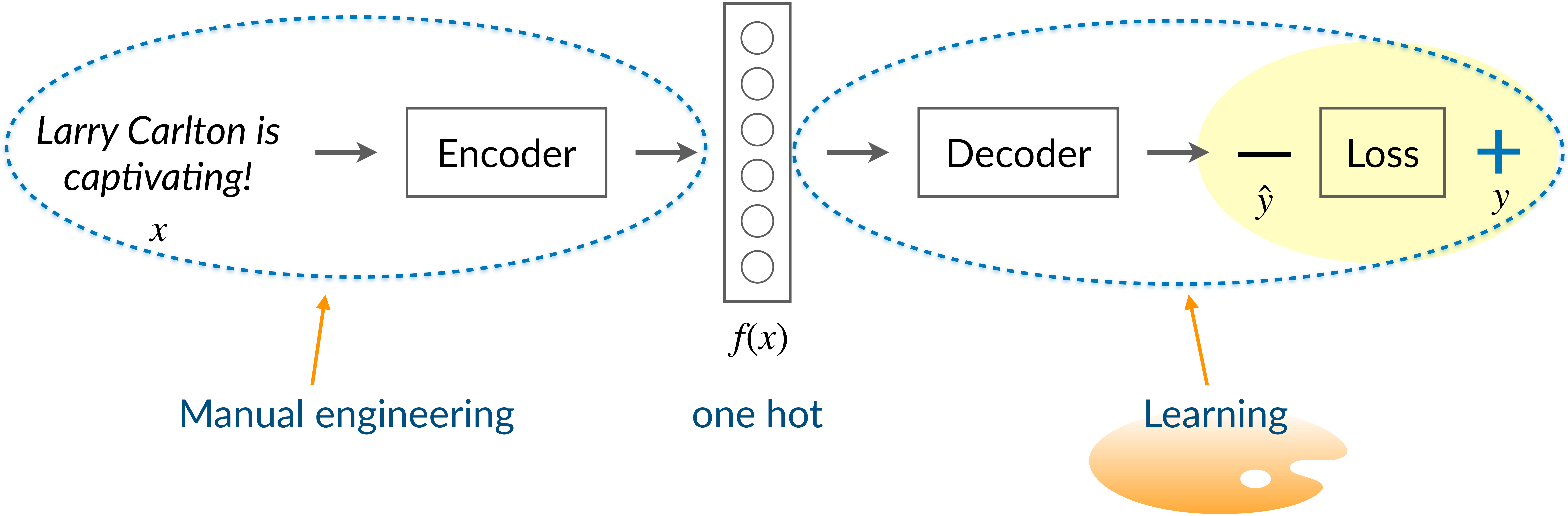
Formally: $$y^* = \underset{\hat{y}}{\operatorname{argmax}}\, p_{\mathbf{z}}\left(\hat{y} \in \{-\,,+\} \mid \mathbf{x}\right)$$

Less trivial when dealing with thousands of classes
(*machine translation*, *language models*)

The NLP view (*for today*)



The NLP view (*for today*)



Training loss

observations (*input*) labels (*output*)

Data set $\mathscr{D} = \left\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\right\}$

per instance loss

Loss function $L(\mathscr{D}, \mathbf{z}) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(\mathbf{x}_i, y_i, \mathbf{z}\right)$

conditional log-likelihood

$$\ell(\mathbf{x}, y, \mathbf{z}) = -\log p_{\mathbf{z}}\left(y \,|\, \mathbf{x}\right)$$

Cross-entropy loss (*logistic regression*)

expected output (*label*)

input

$$L_{\text{ce}}(\mathscr{D}, \mathbf{z}) = -\frac{1}{n}\sum_{i=1}^{n} \log p_{\mathbf{z}}\left(y_i \,|\, \mathbf{x}_i\right)$$

to simplify the notation (*see previous slide*)

$$\boxed{\sigma\left(s_{\mathbf{z}}\left(\mathbf{x}_i\right)\right) \rightarrow \sigma_{\mathbf{x}_i}}$$

Cross-entropy loss

Detailed explanation in Chapter 5 of SLP

Hint: y can be seen as a *Bernoulli* distribution

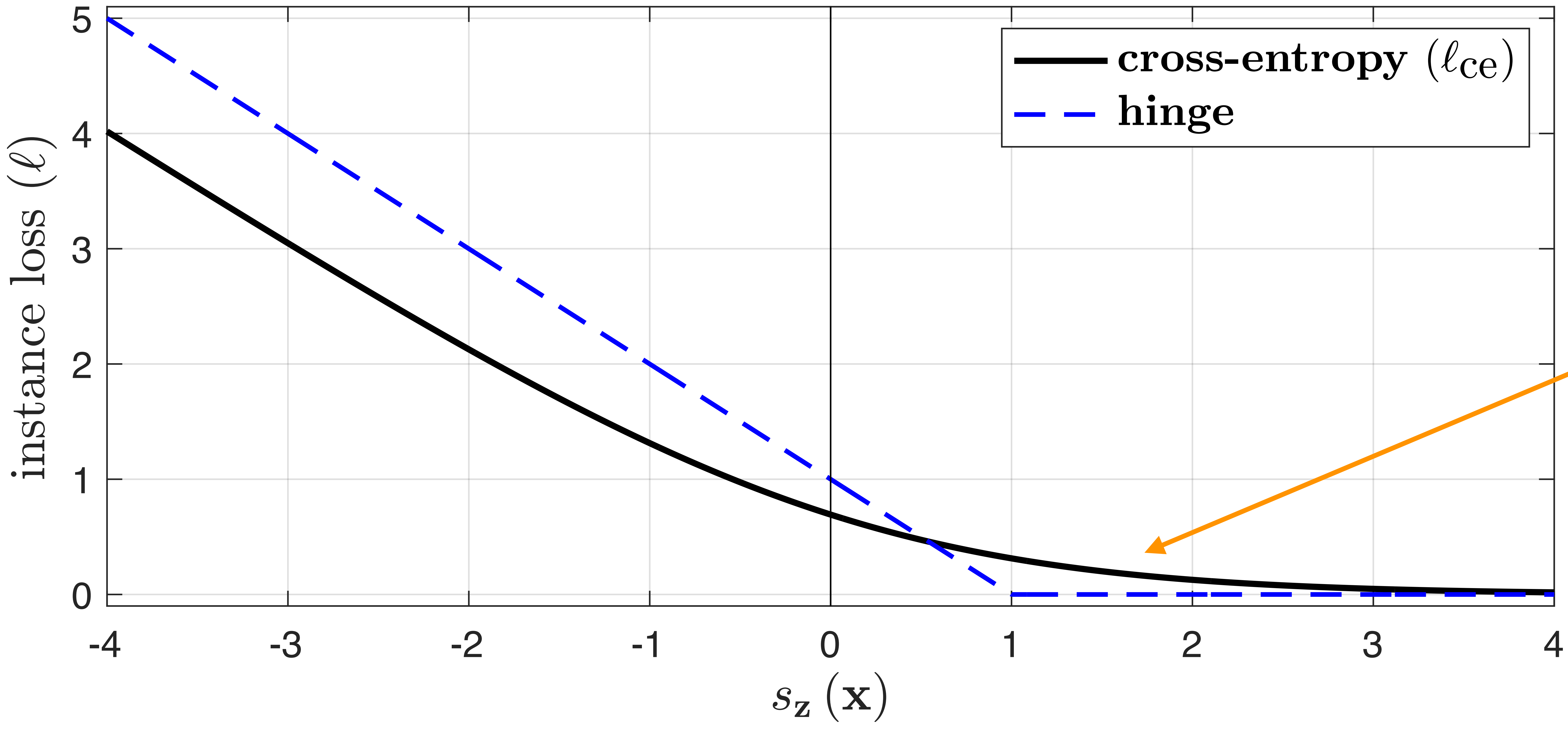
$$= -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log \sigma_{\mathbf{x}_i} + \left(1 - y_i\right) \log\left(1 - \sigma_{\mathbf{x}_i}\right)\right]$$

y_i can either be 1 (+ class) or 0 (−)

Intuition for the cross-entropy loss (*logistic regression*)

When $y_i = 1$ (or the + class)

the instance loss $\ell_{\text{ce}} = -\log \sigma_{\mathbf{x}_i}$



saturates but still rewards

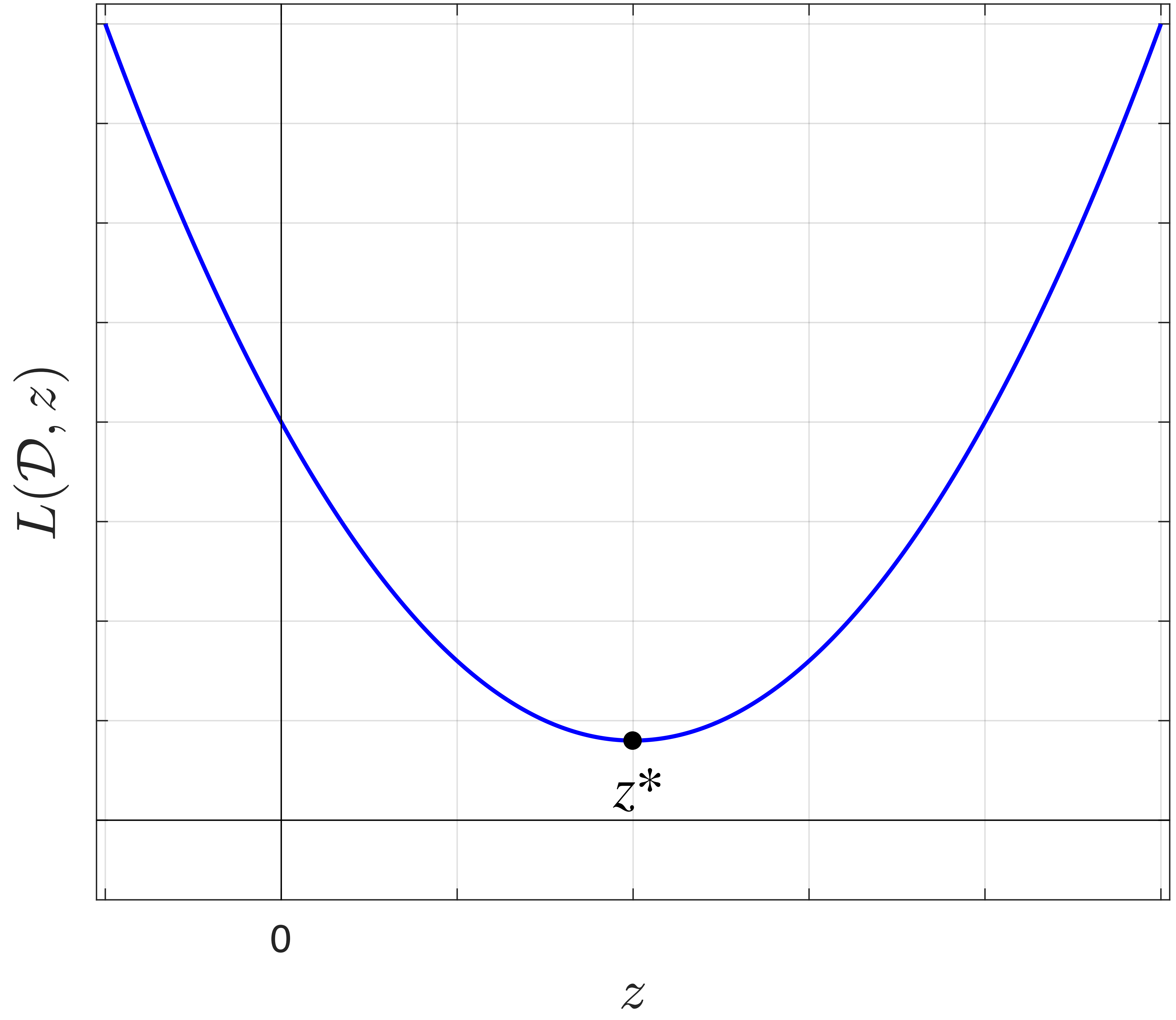
Training (*optimisation*)

Loss

$$\mathbf{z}^* = \arg\min_{\mathbf{z}\in\mathbb{R}^m} L\left(\mathscr{D}, \mathbf{z}\right)$$

best parameters
(*minimising the loss*)

$m = 1$



Gradient descent

$z_0 = \text{random};$

$i = 0;$

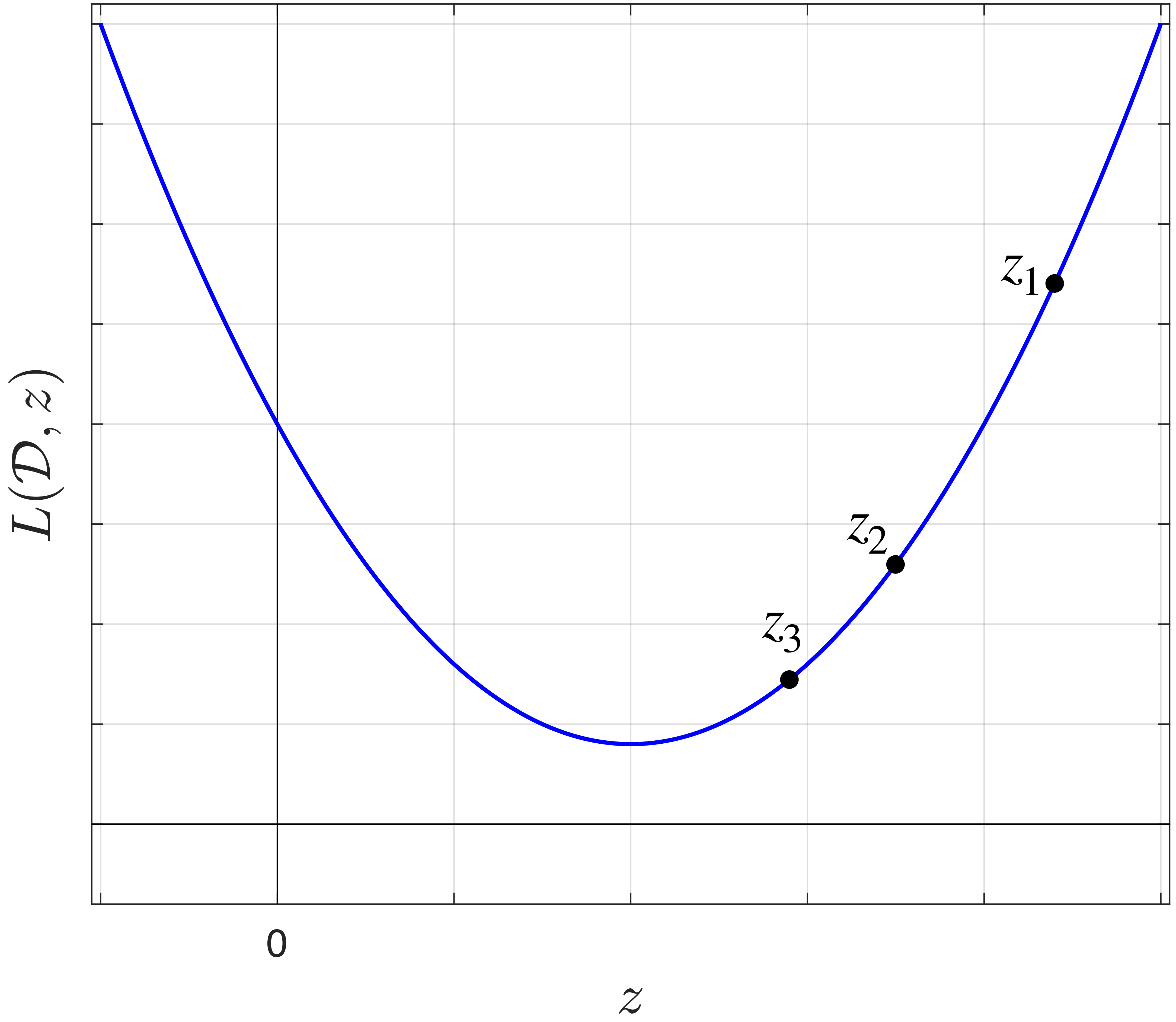
repeat until convergence:

$$z_{i+1} = z_i - \alpha \nabla_z L\left(\mathscr{D}, z_i\right);$$

$$i = i + 1;$$

small

learning rate



Training — Stochastic gradient descent

$$\nabla_{\mathbf{w}} L\left(\mathscr{D}, \mathbf{z}\right) = \nabla_{\mathbf{w}} \frac{1}{n} \left[\ell\left(\mathbf{x}_1, y_1, \mathbf{z}\right) + \ldots + \ell\left(\mathbf{x}_n, y_n, \mathbf{z}\right)\right]$$

Models with **many** parameters and large training sets → gradient descend updates one parameter at a time using *stale* values (for the rest), needs to iterate across all training samples, long time without update

Counter-measure: Approximate gradients via sampling a single training instance (*or in practice a small subset known as a **batch***)

$$\nabla_{\mathbf{w}} L\left(\mathscr{D}, \mathbf{z}\right) \approx \nabla_{w} \ell\left(\mathbf{x}_j, y_j, \mathbf{z}\right)$$

$$z_{i+1} = z_i - \alpha \nabla_{\mathbf{z}} \ell\left(\mathbf{x}_j, y_j, z_i\right)$$

Regularisation

$$
\begin{array}{ccl}
\mathbf{z}_1^* & \mathbf{z}_2^* & \\
\left[\begin{array}{c} 1 \\ -1 \\ 0.5 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{array}\right] &
\left[\begin{array}{c} 0 \\ -1 \\ 0 \\ \vdots \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{array}\right] &
\begin{array}{l} \text{good} \\ \text{bad} \\ \text{like} \\ \vdots \\ \text{good band} \\ \text{good music} \\ \text{good lyrics} \\ \vdots \\ \text{this is a great band} \\ \text{this was a great band} \end{array}
\end{array}
$$

$$
\begin{array}{rcc}
L(\mathscr{D}, \mathbf{z}^*) : & 0.02 & 0.02 \\
\|\mathbf{z}^*\|_2^2 : & 4.09 & 48.7
\end{array}
$$

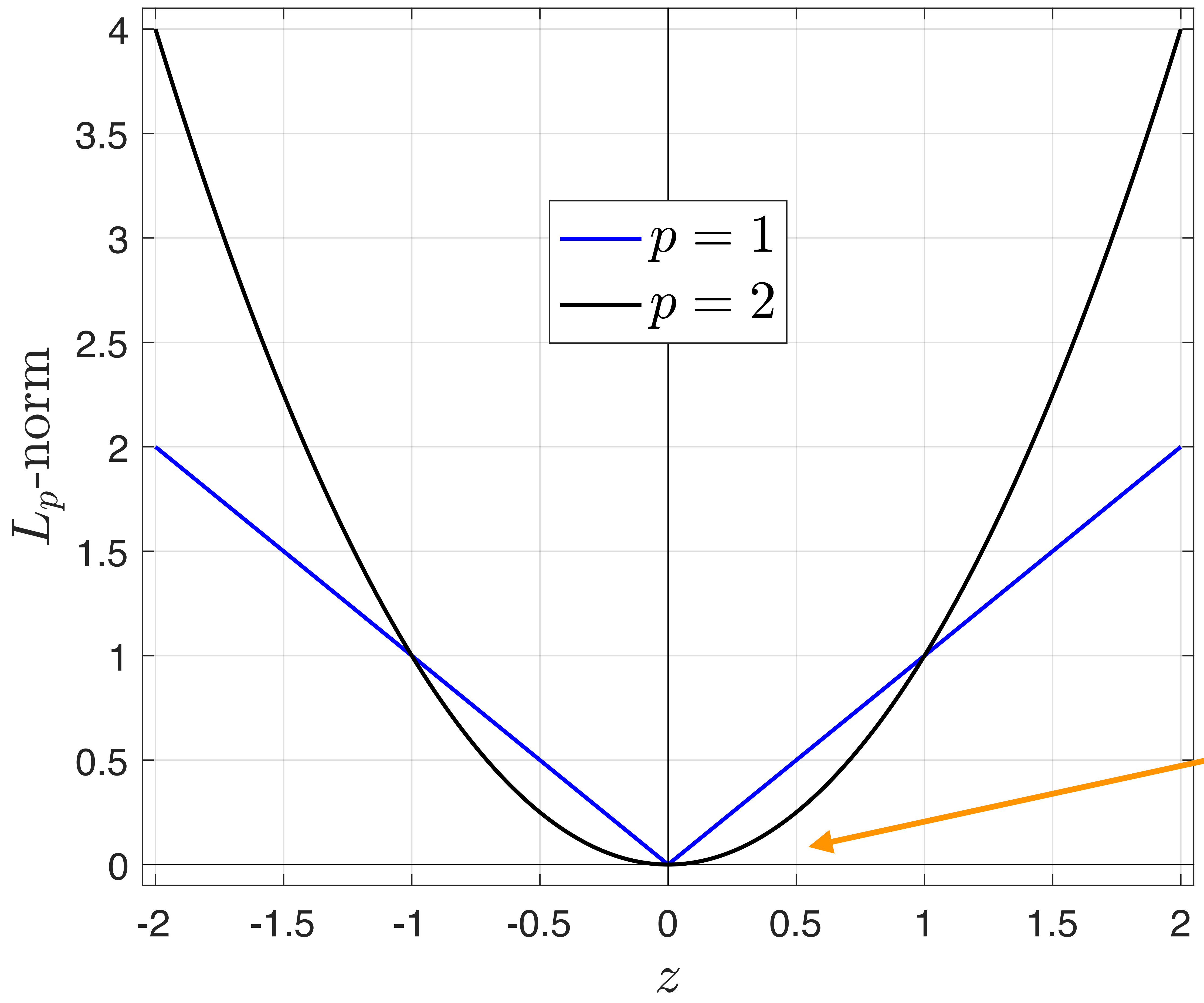
Which one of the two solutions might be **better**?

<u>L2-norm regularisation</u>

$$L_\lambda(\mathscr{D}, \mathbf{z}) = L(\mathscr{D}, \mathbf{z}) + \lambda \|\mathbf{z}\|_2^2$$

L2-norm vs L1-norm regularisation

1-dimensional parameter vector z



L2-norm regularisation

$$L_\lambda(\mathscr{D}, \mathbf{z}) = L(\mathscr{D}, \mathbf{z}) + \lambda ||\mathbf{z}||_2^2$$

L1-norm regularisation

$$L_\lambda(\mathscr{D}, \mathbf{z}) = L(\mathscr{D}, \mathbf{z}) + \lambda ||\mathbf{z}||_1$$

L2 easier to optimise

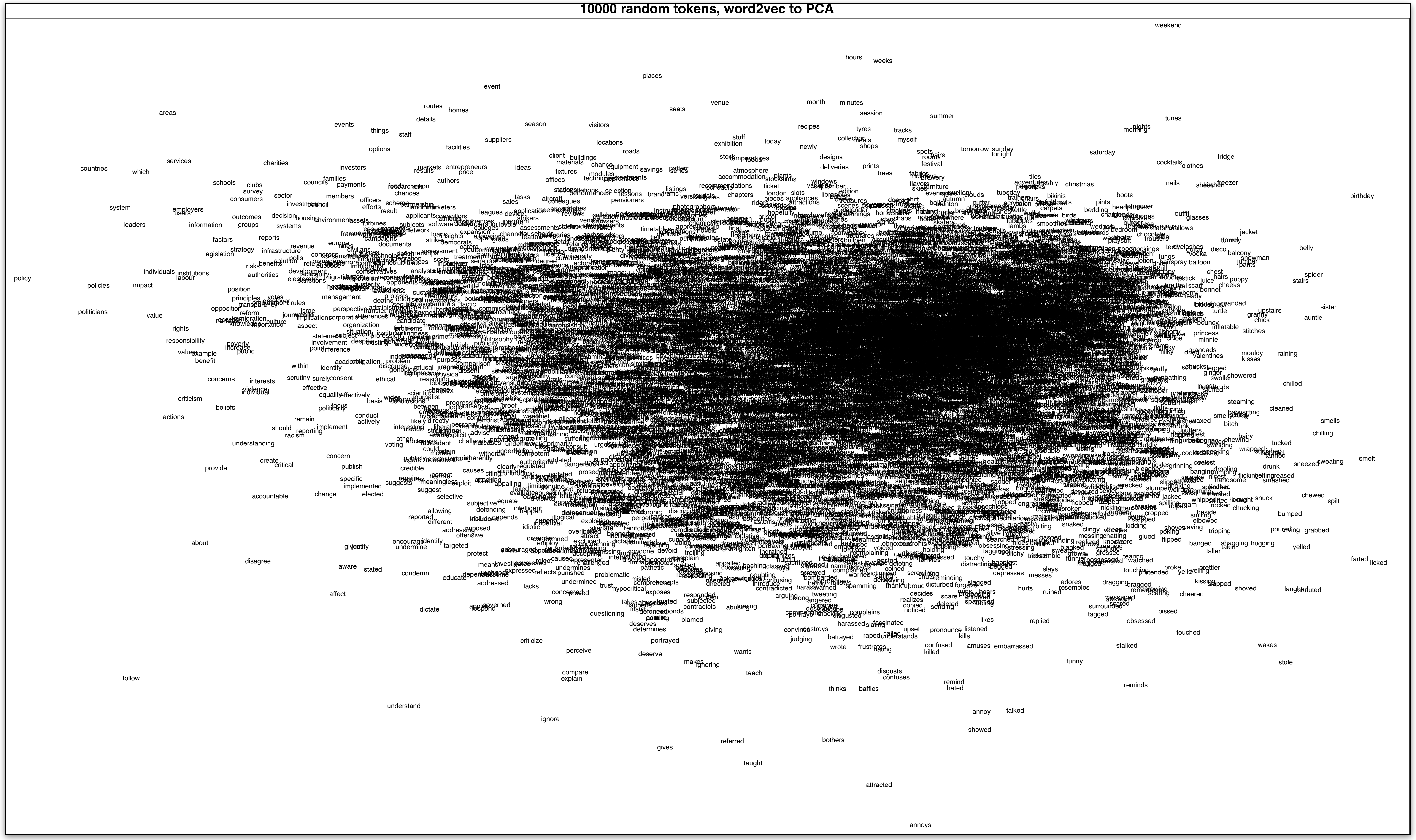
L1 non-continuous derivative at 0

L1 sparse, L2 weights are never 0

Desirable property?

Word (*token*) representation in NLP

figshare.com/articles/dataset/UK_Twitter_word_embeddings/4052331



based on tweets
~ 10 years old!

NB:
Uncensored!

Go to
lampos.net/img/fig-word-cloud.pdf
to zoom in

Why is word representation important?

- In a machine learning task (*if not 100%, then 99% of current NLP tasks*), feature representation is key — *sometimes, it is more important than the machine learning method itself!*
- Hence, better feature representation = better performance
- The main driving force for (*large*) language models

Word representation learning formalised

Words / tokens: w

Vocabulary: $\mathscr{V} = \{w_1, w_2, \ldots, w_n\}$

Learn / find representation function

$$f(w_i) = r_i\,, i = \{1,\ldots,n\}$$

Essential properties of a good word representation

A good word representation makes sure that:

- representations for different words are distinct
- similar words (*what is the definition of similar here?*) should have similar representations

Sparse binary representations

Map words to unique positive non-zero integers

$f(w) \in \mathbb{N}^n$

$$f_j(w_i) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{elsewhere} \end{cases}$$

one-hot vector

For example:

$$f(w_4) = \underbrace{[0 \quad 0 \quad 0 \quad 1 \quad \dots \quad 0]}_{n \text{ elements}}$$

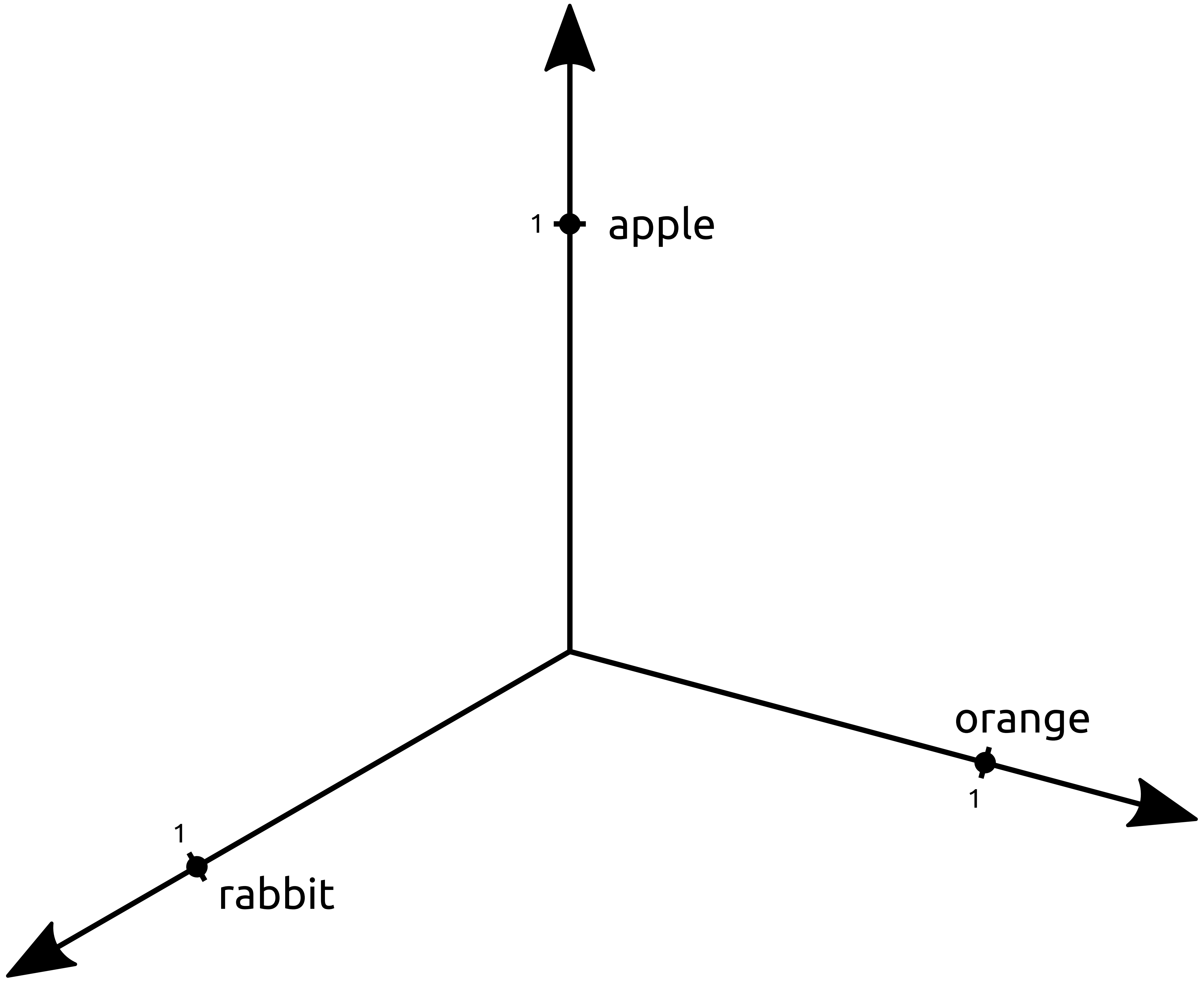
Sparse binary representation example

$$\mathscr{V} = \{\text{apple, orange, rabbit}\}$$

$$f(\text{apple}) = [1 \quad 0 \quad 0]$$

$$f(\text{orange}) = [0 \quad 1 \quad 0]$$

$$f(\text{rabbit}) = [0 \quad 0 \quad 1]$$



Cosine similarity

$$\text{cosine-sim}(\mathbf{w}, \mathbf{v}) = \frac{\sum_{i=1}^{n} w_i \cdot v_i}{\sqrt{\sum_{i=1}^{n} w_i^2} \cdot \sqrt{\sum_{i=1}^{n} v_i^2}} = \frac{\mathbf{w}^\top \mathbf{v}}{\|\mathbf{w}\|_2 \|\mathbf{v}\|_2} = \cos \phi$$

where ϕ is the angle between $\mathbf{w}$ and $\mathbf{v}$ in a vector space

ranges from $[-1,1]$, but for non-negative representations from $[0,1]$

$\text{cosine-sim} = 1 \rightarrow$ identical ($\phi = 0°$)

$\text{cosine-sim} = -1 \rightarrow$ opposites ($\phi = 180°$)

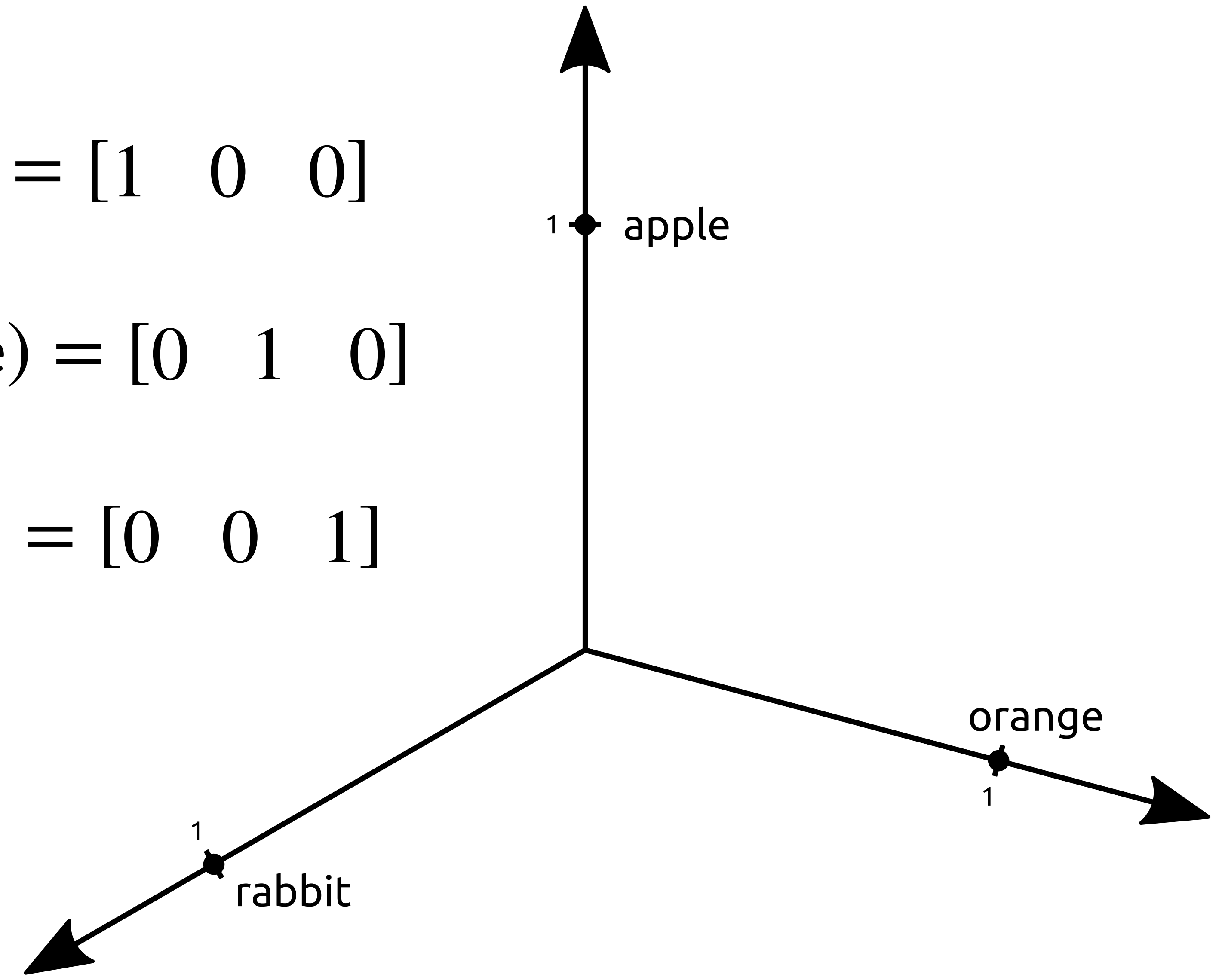
$\text{cosine-sim} = 0 \rightarrow$ orthogonal ($\phi = 90°$)

Sparse binary (*one-hot*) cosine similarities (*are irrelevant*)

$f(\text{apple}) = [1 \quad 0 \quad 0]$

$f(\text{orange}) = [0 \quad 1 \quad 0]$

$f(\text{rabbit}) = [0 \quad 0 \quad 1]$



$$\text{cosine-sim}\left(f(\text{apple}), f(\text{orange})\right) = 0$$

$$\text{cosine-sim}\left(f(\text{apple}), f(\text{rabbit})\right) = 0$$

$$\text{cosine-sim}\left(f(\text{orange}), f(\text{rabbit})\right) = 0$$

Dense continuous word representations

Vocabulary ($\mathcal{V}$) words / tokens are represented as matrix rows

$$\mathbf{W} \in \mathbb{R}^{|\mathcal{V}| \times d}$$

d: dimensionality of the continuous representation

The representation of a word w, $f(w)$, is now a row of $\mathbf{W}$:

$f(w) = \mathbf{W}_{i,:}$ or simply $\mathbf{w}_i \in \mathbb{R}^d$

Dense continuous word representations example

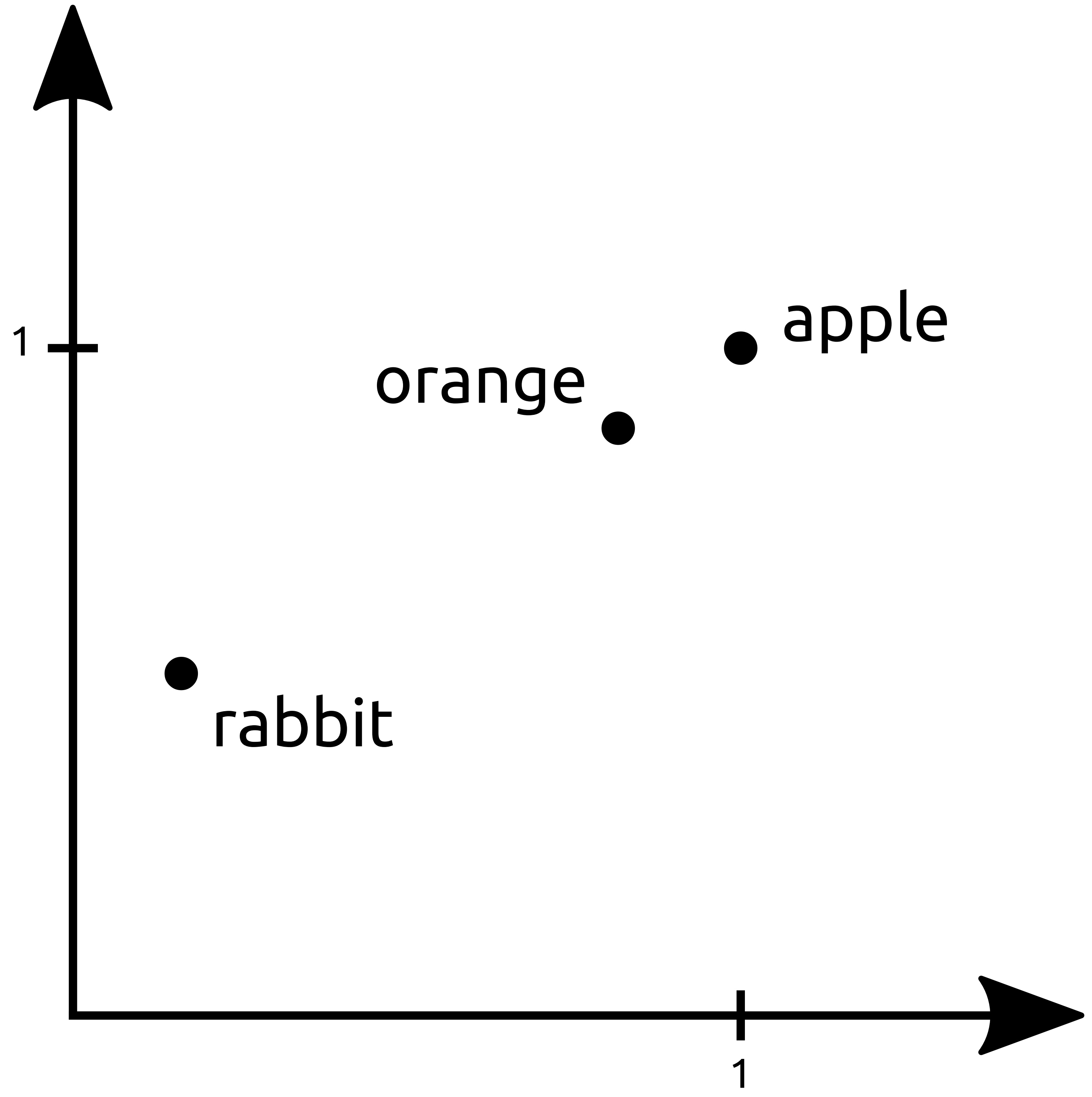
$\mathcal{V} = \{\text{apple, orange, rabbit}\}$

Assuming $d = 2$, $\mathbf{W} \in \mathbb{R}^{3 \times 2}$

$f(\text{apple}) = [1.0 \quad 1.0]$

$f(\text{orange}) = [0.9 \quad 1.0]$

$f(\text{rabbit}) = [0.1 \quad 0.5]$

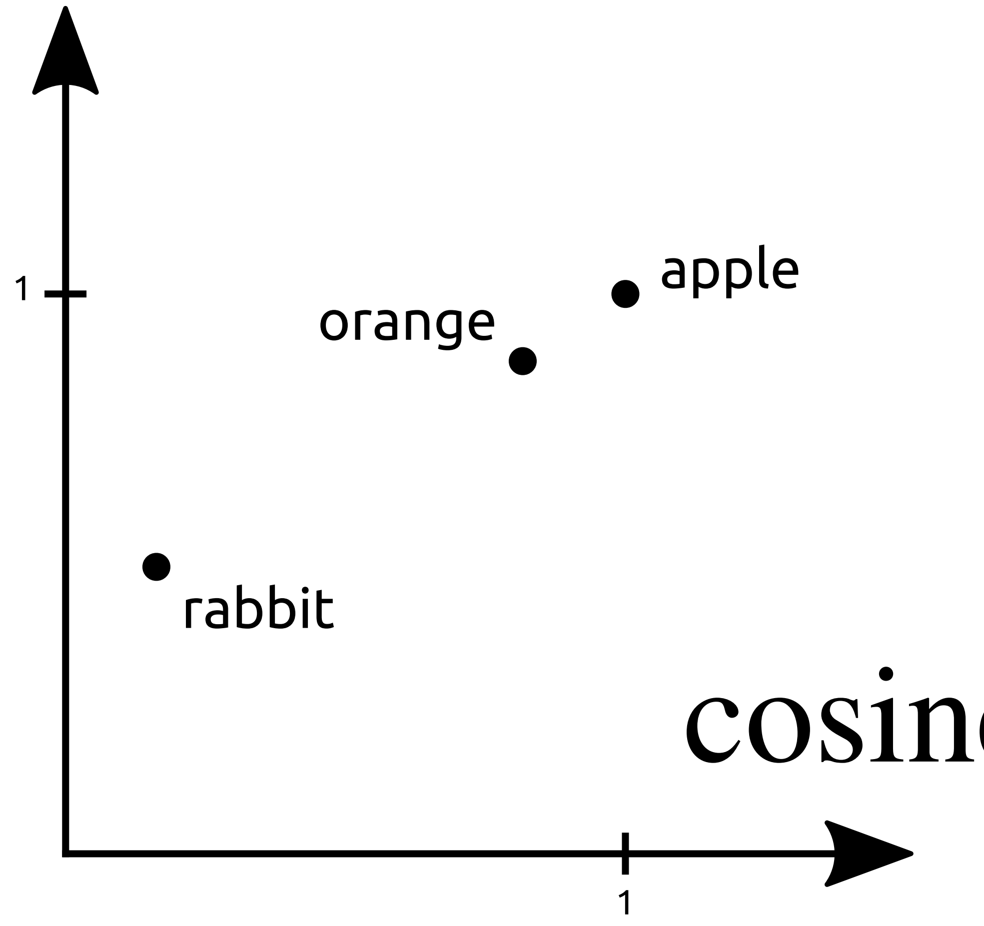


Dense continuous word similarities

$f(\text{apple}) = [1.0 \quad 1.0]$

$f(\text{orange}) = [0.9 \quad 1.0]$

$f(\text{rabbit}) = [0.1 \quad 0.5]$



$$\text{cosine-sim}\left(f(\text{apple}), f(\text{orange})\right) \approx 0.999$$

$$\text{cosine-sim}\left(f(\text{apple}), f(\text{rabbit})\right) \approx 0.8321$$

$$\text{cosine-sim}\left(f(\text{orange}), f(\text{rabbit})\right) \approx 0.8601$$

Learning representations

"***You shall know a word the company it keeps***"

John Rupert (J. R.) Firth (1957)

Word co-occurrences

"… comparing an ***apple*** to an ***orange***…"

"… an ***apple*** from Italy and an ***orange*** from Spain…"

"… my ***rabbit*** does not like ***orange*** juice…"

Sparse word co-occurrence representations

Record the number of times words co-occur
in a collection of documents (*corpus*)

$$\mathbf{C} \in \mathbb{N}^{|\mathcal{V}| \times |\mathcal{V}|} \quad \text{e.g. } \mathbf{C} = \begin{array}{c} \begin{array}{ccc} \text{apple} & \text{orange} & \text{rabbit} \end{array} \\ \begin{bmatrix} 2 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 1 \end{bmatrix} \end{array} \begin{array}{l} \text{apple} \\ \text{orange} \\ \text{rabbit} \end{array}$$

Similarities based on a co-occurrence matrix

$$\mathbf{C} \in \mathbb{N}^{|\mathscr{V}| \times |\mathscr{V}|} \quad \text{e.g. } \mathbf{C} = \begin{array}{c} \begin{array}{ccc} \text{apple} & \text{orange} & \text{rabbit} \end{array} \\ \begin{bmatrix} 2 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 1 \end{bmatrix} \end{array} \begin{array}{l} \text{apple} \\ \text{orange} \\ \text{rabbit} \end{array}$$

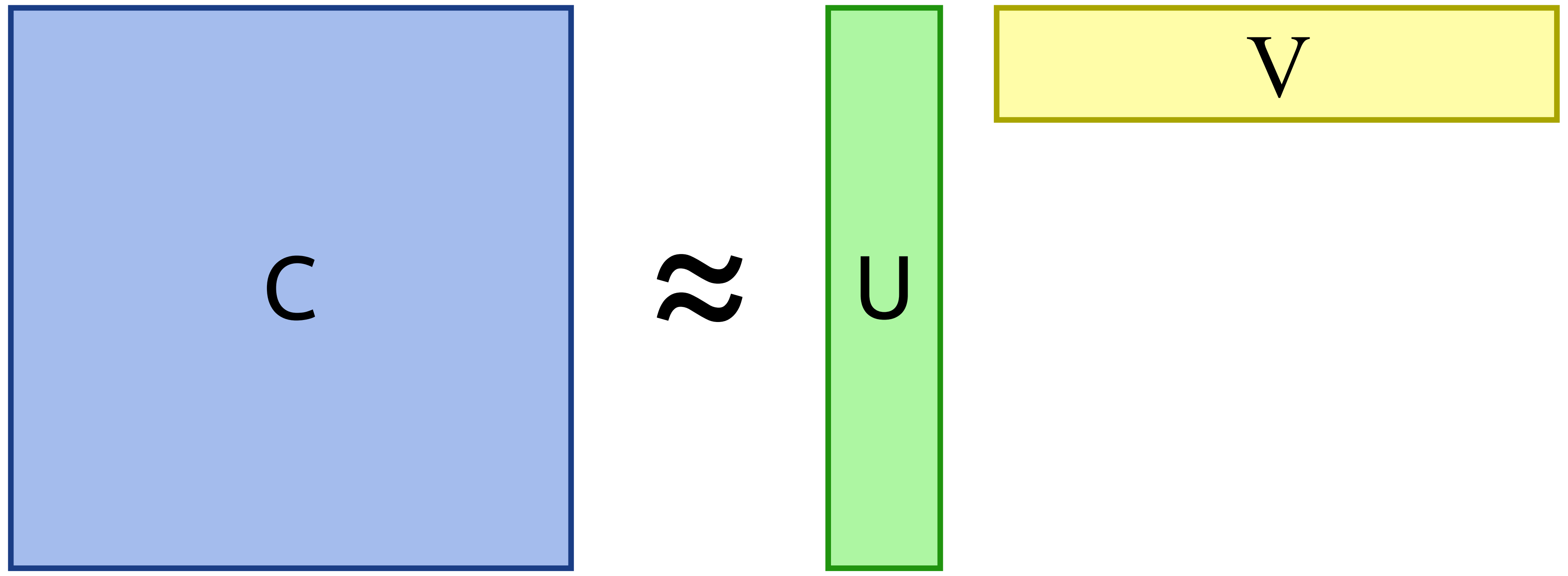
$$\text{cosine-sim}\left(f(\text{apple}), f(\text{orange})\right) \approx 0.995$$

$$\text{cosine-sim}\left(f(\text{apple}), f(\text{rabbit})\right) = 0.5$$

$$\text{cosine-sim}\left(f(\text{orange}), f(\text{rabbit})\right) \approx 0.756$$

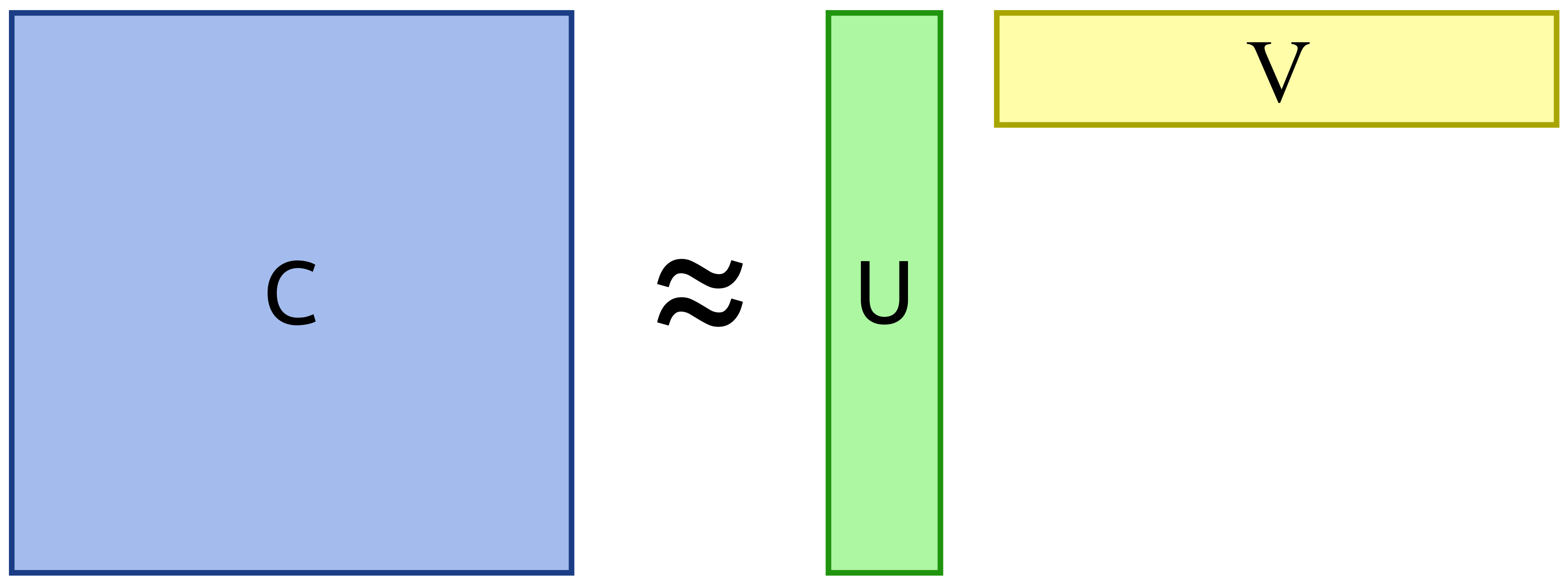
Dense continuous representations via matrix factorisation (SVD)

$$\mathbf{C} \in \mathbb{N}^{|\mathcal{V}| \times |\mathcal{V}|} \qquad \text{e.g. } \mathbf{C} = \begin{array}{cl} \begin{array}{ccc} \text{apple} & \text{orange} & \text{rabbit} \end{array} & \\ \begin{bmatrix} 2 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 1 \end{bmatrix} & \begin{array}{l} \text{apple} \\ \text{orange} \\ \text{rabbit} \end{array} \end{array}$$



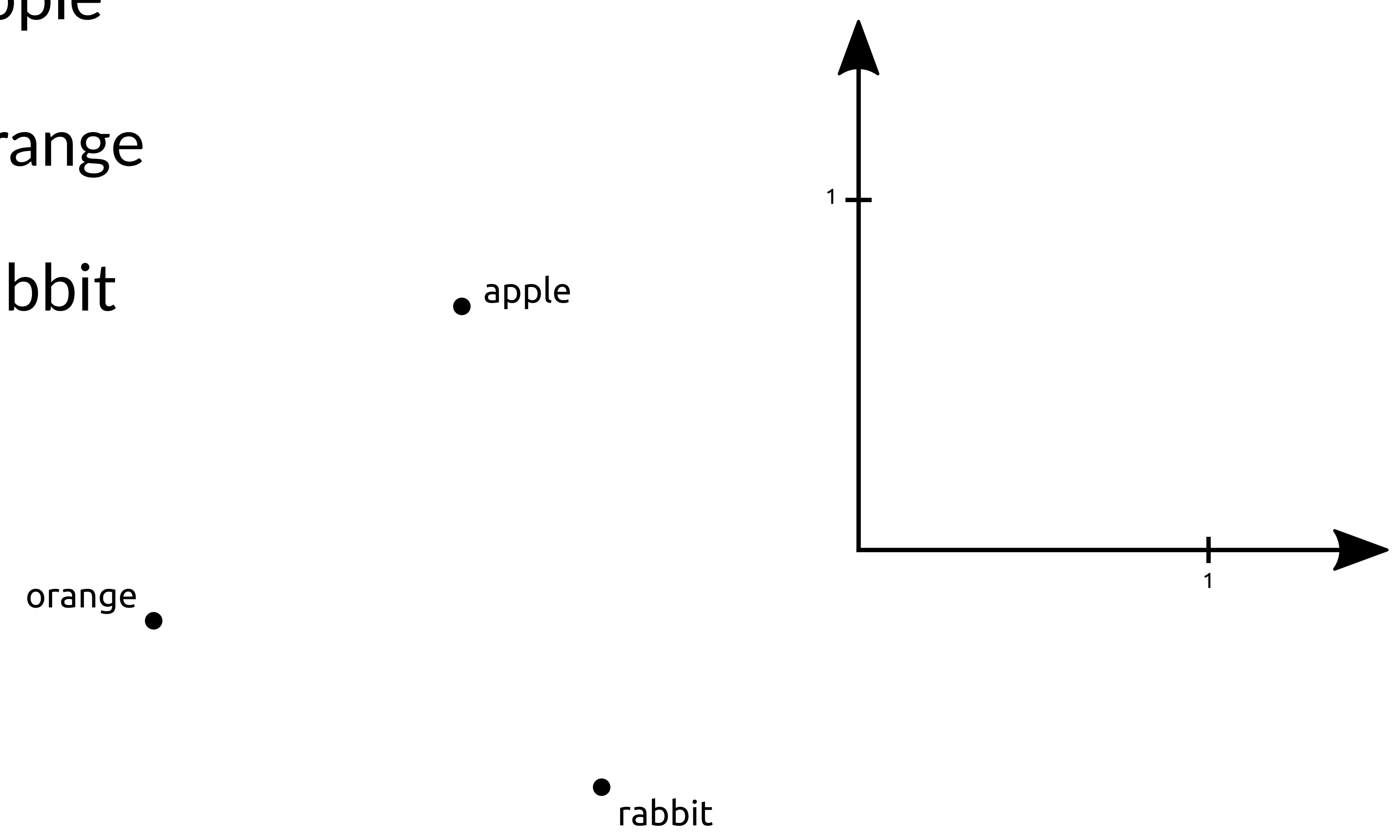
$$\mathbf{U} \in \mathbb{R}^{|\mathcal{V}| \times d}$$
$$\mathbf{V} \in \mathbb{R}^{d \times |\mathcal{V}|}$$

Dense continuous representations via matrix factorisation (SVD)

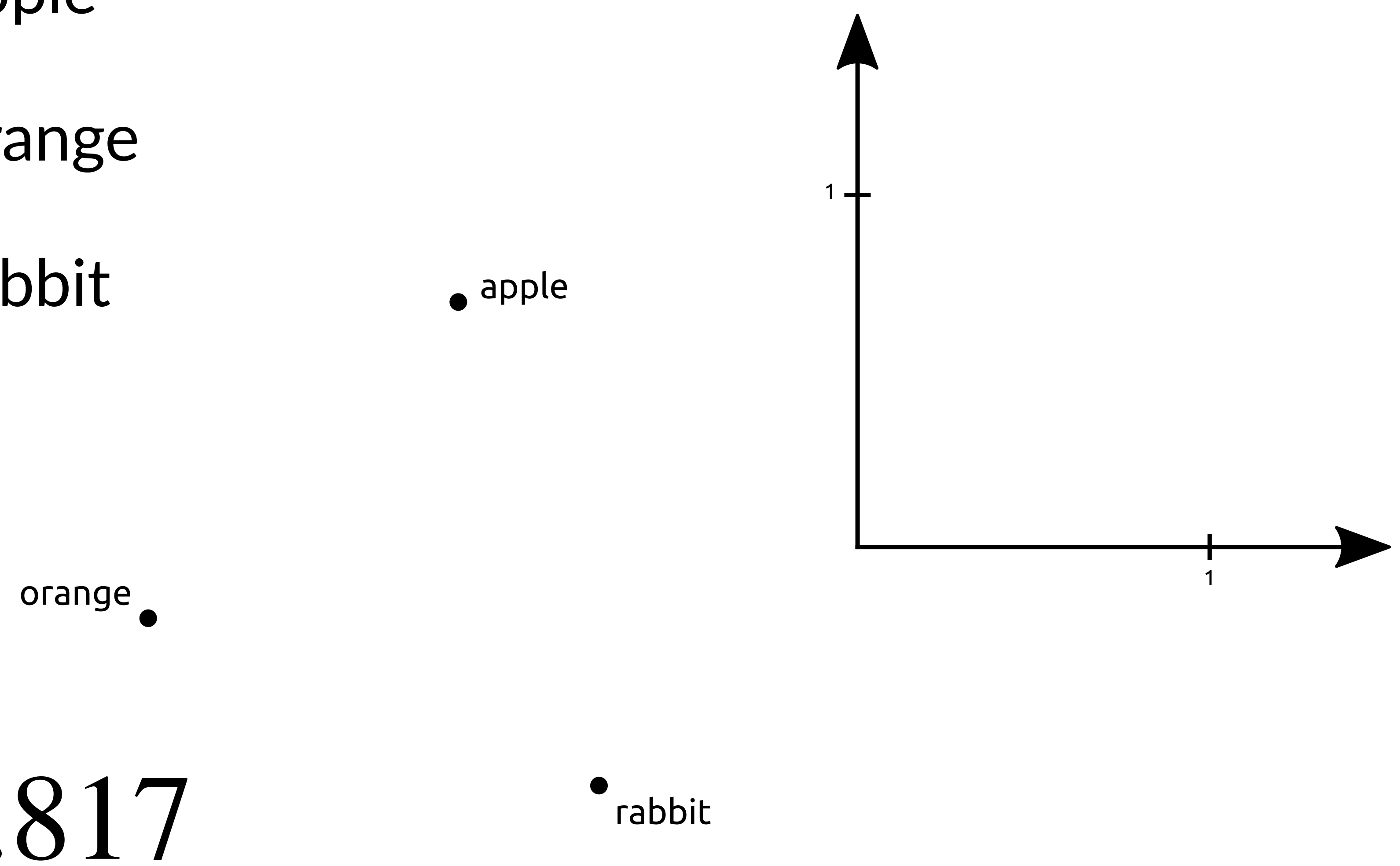


$$\mathbf{C} \in \mathbb{N}^{|\mathscr{V}| \times |\mathscr{V}|}$$
$$\mathbf{U} \in \mathbb{R}^{|\mathscr{V}| \times d}$$
$$\mathbf{V} \in \mathbb{R}^{d \times |\mathscr{V}|}$$

Let’s assume that $\mathbf{U} = \begin{bmatrix} -1.26 & 0.65 \\ -1.72 & -0.24 \\ -0.46 & -0.89 \end{bmatrix}\begin{array}{l} \text{apple} \\ \text{orange} \\ \text{rabbit} \end{array}$



Let's assume that
$$\mathbf{U} = \begin{array}{cl} \begin{bmatrix} -1.26 & 0.65 \\ -1.72 & -0.24 \\ -0.46 & -0.89 \end{bmatrix} & \begin{array}{l} \text{apple} \\ \text{orange} \\ \text{rabbit} \end{array} \end{array}$$



$$\text{cosine-sim}\left(f(\text{apple}), f(\text{orange})\right) \approx 0.817$$

$$\text{cosine-sim}\left(f(\text{apple}), f(\text{rabbit})\right) \approx 0.001$$

$$\text{cosine-sim}\left(f(\text{orange}), f(\text{rabbit})\right) \approx 0.578$$

Learning by slot filling — Word embeddings

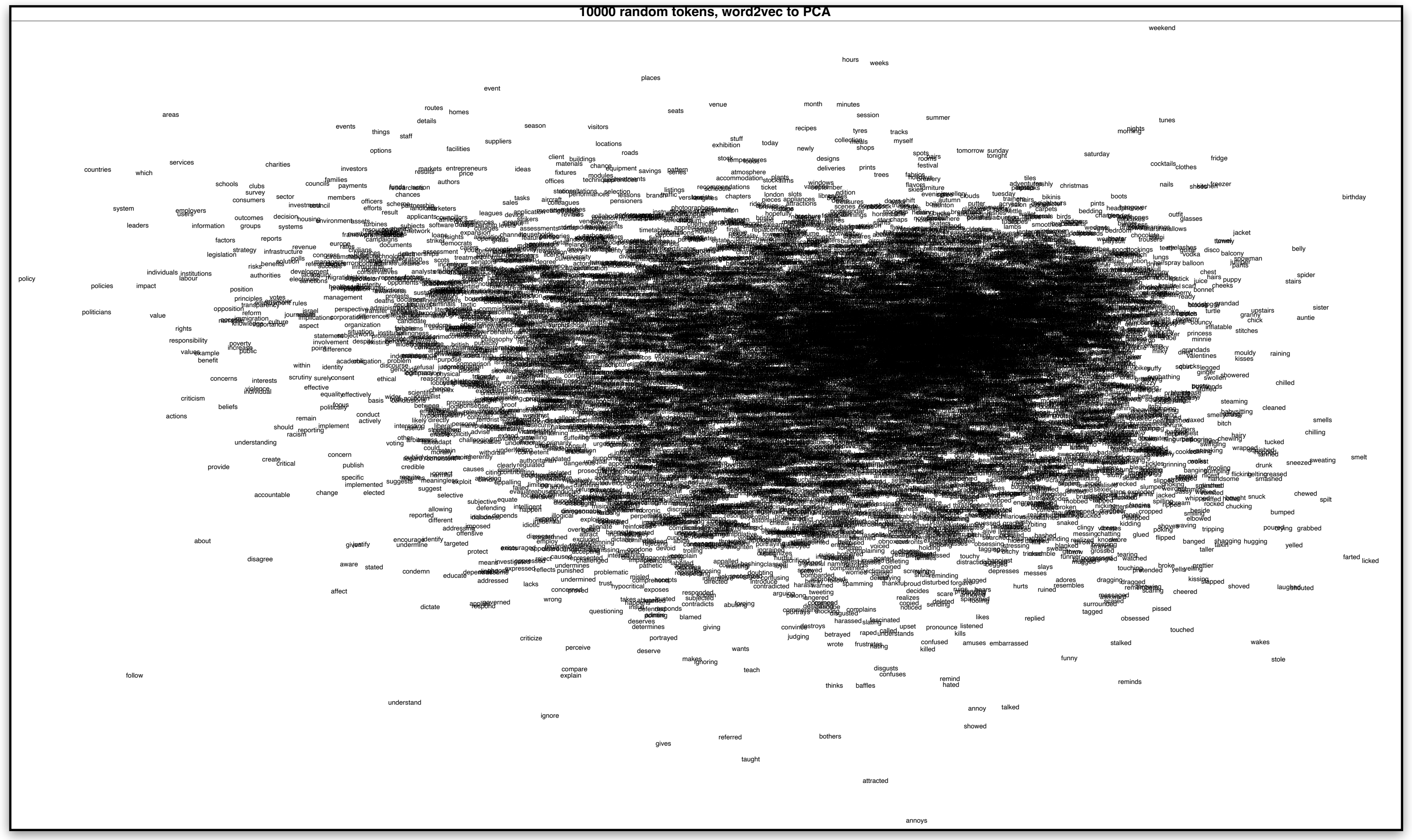
"I had ____ with milk for breakfast today."

- Good: *cereals*
- Acceptable (?): *pizza*
- Bad: *songs*

"Pink Floyd have become ____ numb."

- Good: *comfortably*
- Acceptable (?): *very*
- Bad: *dysfunctional*

Neural word representations — Cosine similarity



figshare.com/articles/dataset/UK_Twitter_word_embeddings/4052331

Go to

lampos.net/img/fig-word-cloud.pdf

to zoom in

$$\text{cosine-sim}\left(f(\text{apple}), f(\text{orange})\right) \approx 0.300$$

$$\text{cosine-sim}\left(f(\text{apple}), f(\text{rabbit})\right) \approx 0.094$$

$$\text{cosine-sim}\left(f(\text{orange}), f(\text{rabbit})\right) \approx 0.091$$