732A92/TDDE16 Text Mining (2020)

Word embeddings

Marco Kuhlmann
Department of Computer and Information Science

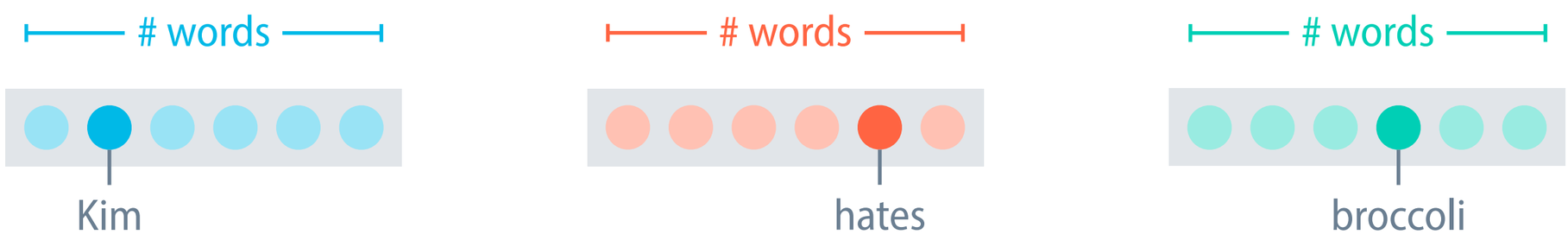
LINKÖPING UNIVERSITY

Reminder: The vector-space model for IR

	Scandal in Bohemia	Final problem	Empty house	Norwood builder	Dancing men	Retired colourman
Adair	0	0	14	0	0	0
Adler	13	0	0	0	0	0
Lestrade	0	0	10	51	0	0
Moriarty	0	20	15	1	0	0

One-hot vectors

- To process text using machine learning algorithms, we need to encode it into vectors of numerical values.
- For individual words, the most straightforward way to do this is to use **one-hot vectors** – all components but one are zero.



Problems with one-hot word representations

- One-hot vectors are long and sparse, which is challenging from a computational point of view.

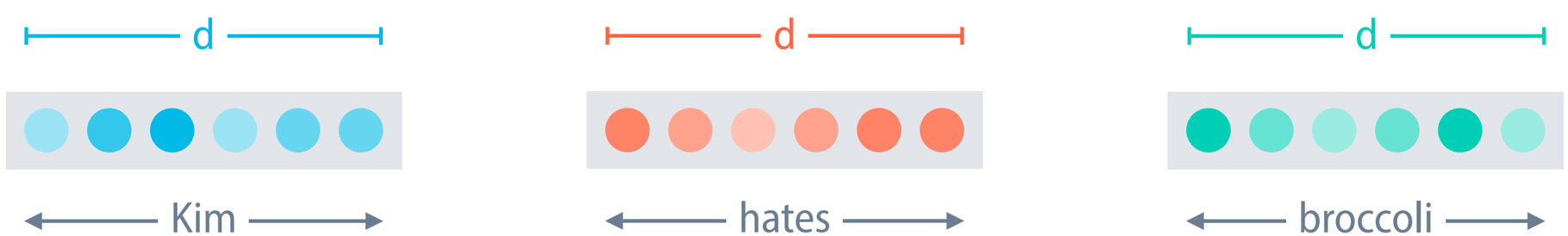
  hundreds of thousands of dimensions, zero almost everywhere

- One-hot vectors do not give rise to a useful notion of **similarity** between words, which is problematic for machine learning.

  bike and *bikes* as similar/dissimilar as *bike* and *carrot*

Word embeddings

- A **word embedding** is a mapping from a vocabulary of words V to a d-dimensional vector space, where $d \ll |V|$.
- In comparison to one-hot vectors, embedding vectors are typically much shorter, but dense.



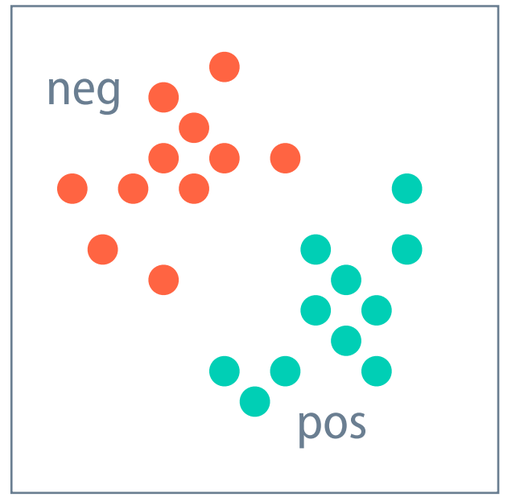
Desirable properties of word embeddings

What do we expect from words with similar embeddings?

- **Machine learning perspective**

  Words behave similarly in learning tasks.

  Example: sentiment classification



- **Linguistic perspective**

  Words have similar meanings.

  But how to operationalise this notion?

Words and contexts

What do the following sentences tell us about *garrotxa*?

- *Garrotxa* is made from milk.
- *Garrotxa* pairs well with crusty country bread.
- *Garrotxa* is aged in caves to enhance mold development.

Sentences taken from the English Wikipedia

The distributional principle

- The **distributional principle** states that words that occur in similar contexts tend to have similar meanings.

- ‘You shall know a word by the company it keeps.’

  Firth (1957)

- Based on this idea, we can try to operationalise the notion of *meaning similarity* as *distributional similarity*.

  similar vectors = similar co-occurrence patterns = similar meanings

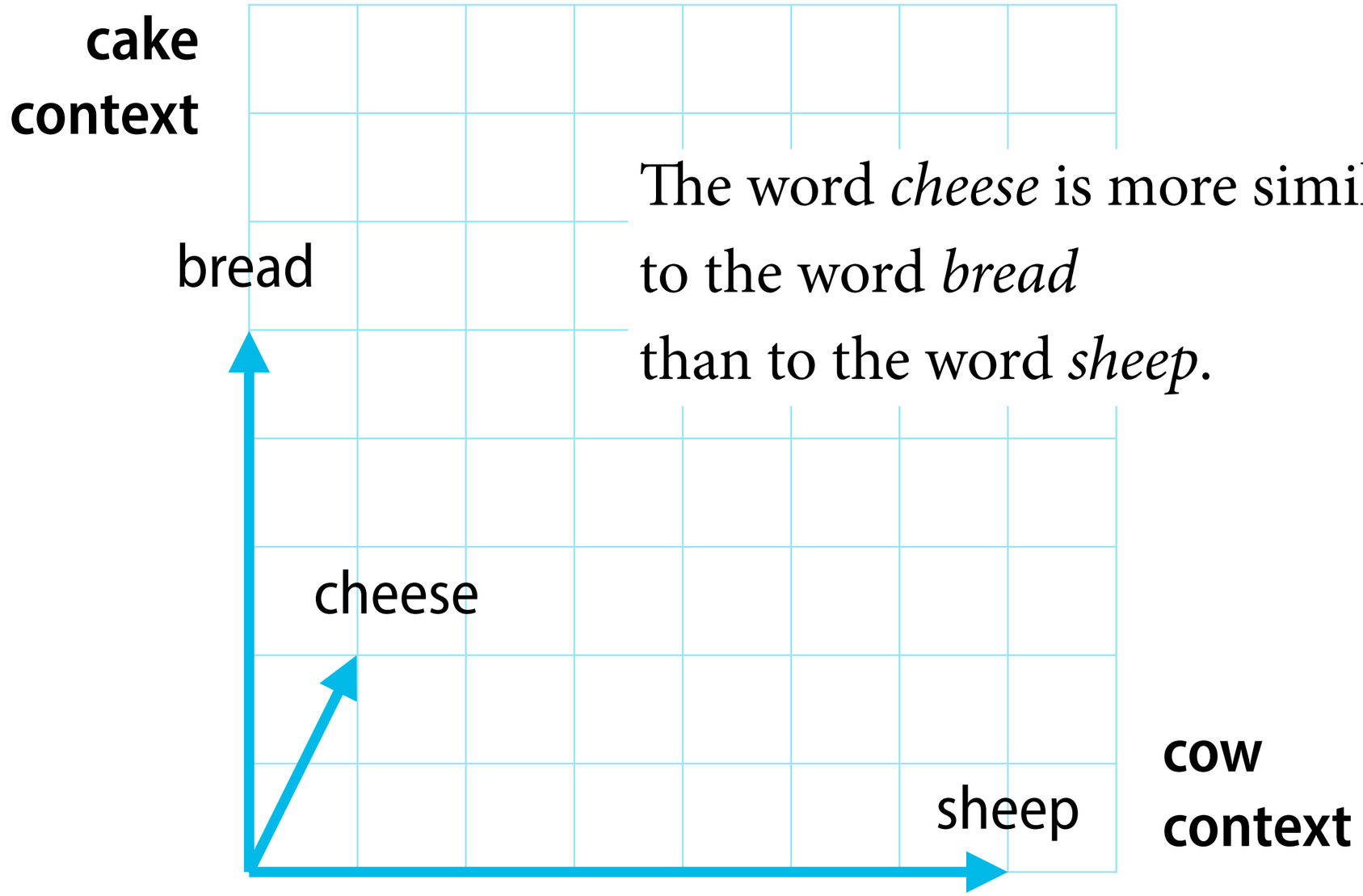
Co-occurrence matrix

context words

target words	butter	cake	cow	deer
cheese	12	2	1	0
bread	5	5	0	0
goat	0	0	6	1
sheep	0	0	7	5

word vector for *cheese*

From co-occurrences to word vectors



The word *cheese* is more similar to the word *bread* than to the word *sheep*.

Issues with co-occurrence vectors

- The row vectors of co-occurrence matrices are long; our word vectors would ideally be short.

  hundreds instead of hundreds of thousands of dimensions

- Raw observation counts give too much weight to stop words and common function words such as *the*, *she*, *has*.

  We should use other measures of association.

Obtaining word embeddings

- Word embeddings can either be trained 'from scratch' in the context of a specific task, or they can be pre-trained.

- In standard deep learning libraries, word embeddings are implemented via **embedding layers**.

  initialised randomly, updated through backpropagation

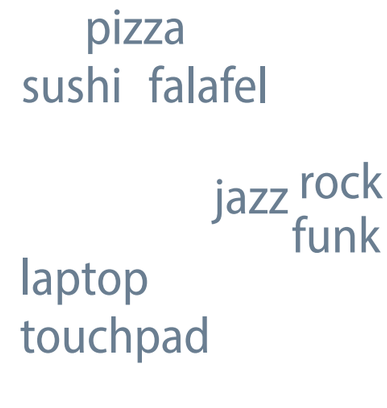
- Pre-trained word vectors for many different languages are available for download, and included in many NLP libraries.

  word2vec, GloVe, Polyglot project, spaCy

Evaluation of word embeddings

- visualisation of the embedding space

  requires dimensionality reduction (PCA, t-SNE)



- computing relative similarities

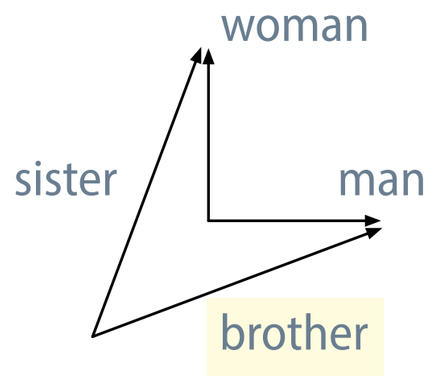
  standard measure: cosine similarity

- similarity benchmarks

  Example: odd one out – *breakfast lunch dinner surgery*

- analogy benchmarks

  woman is to *man* as *sister* is to *?*



Recognising textual entailment

Two doctors perform surgery on patient.

Entailment	Doctors are performing surgery.
Neutral	Two doctors are performing surgery on a man.
Contradiction	Two surgeons are having lunch.

Example from Bowman et al. (2015)

Limitations of word embeddings

- There are many different facets of 'semantic similarity'.

  Is a *cat* more similar to a *dog* or to a *tiger*?

- Text data does not reflect many trivial properties of words.

  more *black sheep* than *white sheep*

- Word vectors reflect social biases in the data used to train them.

  including gender and ethnic stereotypes

Goldberg (2017); Caliskan et al. (2017); Garg et al. (2018)

Embedding bias and occupation participation

Figure 1 from Garg et al. (2018)

This lecture

- Introduction to word embeddings
- Learning word embeddings via matrix factorisation
- Learning word embeddings with neural networks
- The skip-gram model
- Subword models
- Contextualized word embeddings

Learning word embeddings via matrix factorization

Co-occurrence matrix

context words

target words	butter	cake	cow	deer
cheese	12	2	1	0
bread	5	5	0	0
goat	0	0	6	1
sheep	0	0	7	5

word vector for *cheese*

Word embeddings via singular value decomposition

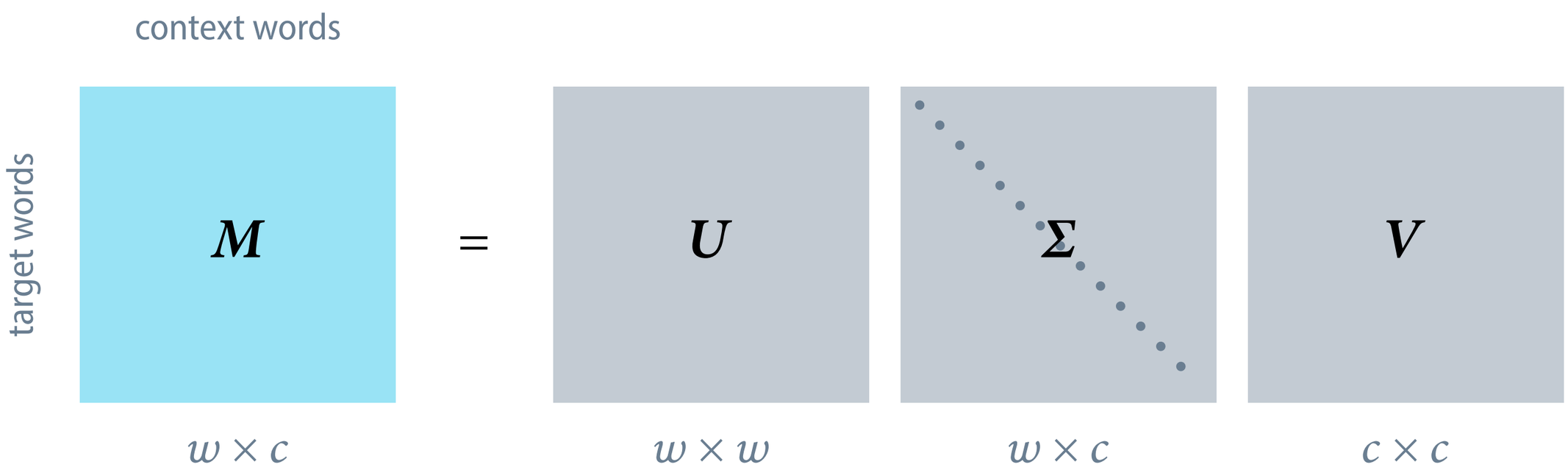
- The row vectors of co-occurrence matrices are long; our word vectors would ideally be short.

  hundreds instead of hundreds of thousands of dimensions

- One idea is to approximate the co-occurrence matrix by another matrix with fewer columns.

- This problem can be solved by computing the **singular value decomposition** of the co-occurrence matrix.

Singular value decomposition



Landauer and Dumais (1997)

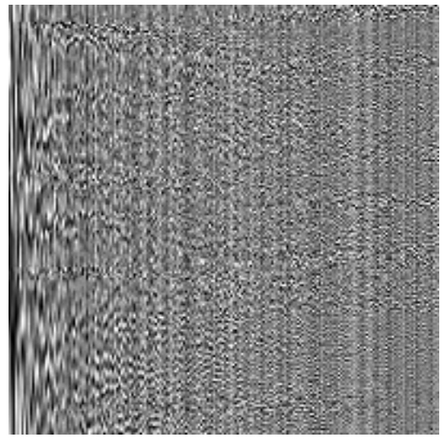
Truncated singular value decomposition

context words

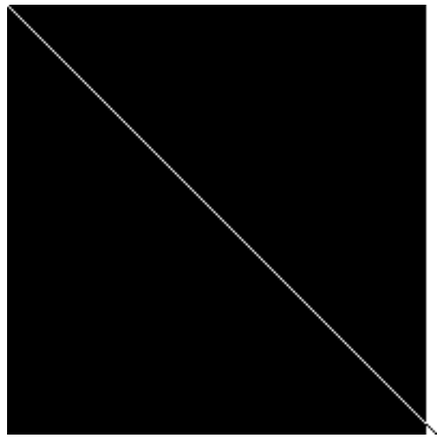
target words

$w \times c$

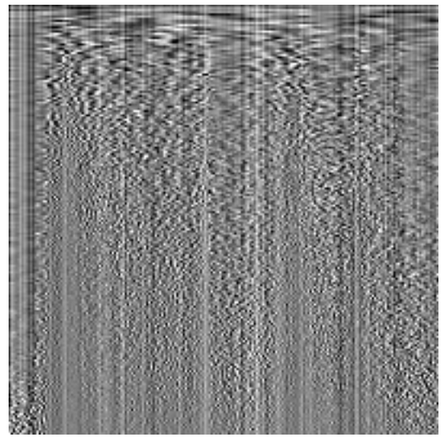
=



$w \times w$



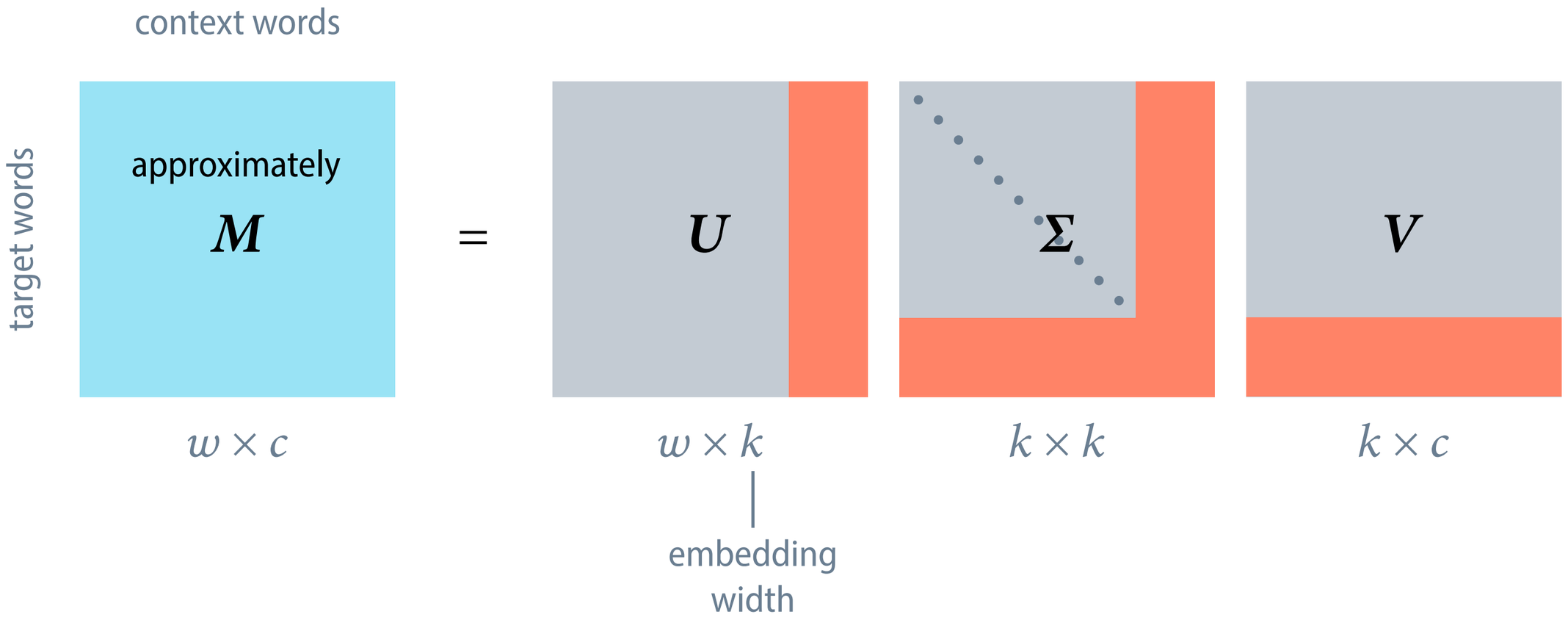
$w \times c$



$c \times c$

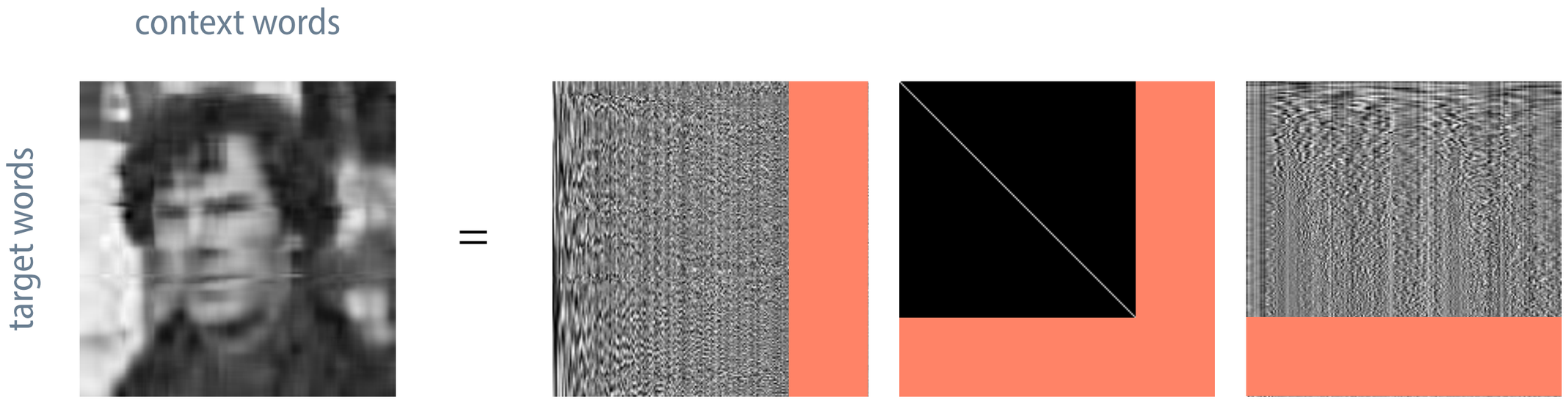
Landauer and Dumais (1997)

Truncated singular value decomposition



Landauer and Dumais (1997)

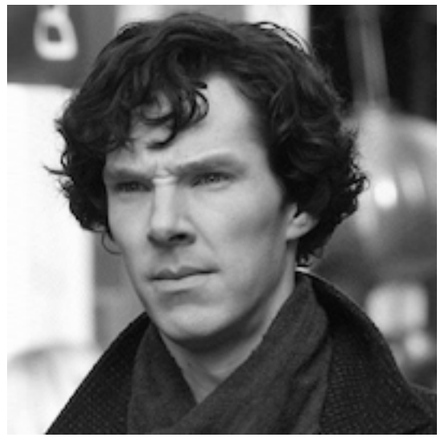
Truncated singular value decomposition



Landauer and Dumais (1997)

Truncated singular value decomposition

width 200



width 100

width 50

width 20

width 10

width 5

Problems with singular value decomposition

- Raw observation counts give too much weight to stop words and common function words such as *the*, *she*, *has*.

  Solutions: threshold, scaling, other association measures

- Computing the singular value decomposition for large matrices is expensive – and co-occurrence matrices *are* large!

  $O(mn^2)$ flops, assuming that $m \geq n$

- The decomposition needs to be recomputed whenever a new word is added to the vocabulary.

Pointwise mutual information

- We want a measure of association that emphasises word–context pairs that are more frequent than what we would expect.
- One well-known measure that satisfies this criterion is **pointwise mutual information (PMI)**:

$$\text{PMI}(x, y) = \log \frac{P(x, y)}{P(x)P(y)}$$

Pointwise mutual information

- We want to use PMI to measure the associative strength between a word w and a context c in a data set D:

$$\mathrm{PMI}(w, c) = \log \frac{P(w, c)}{P(w)P(c)}$$

- We can estimate the relevant probabilities by counting:

$$\mathrm{PMI}(w, c) = \log \frac{\#(w, c)/|D|}{\#(w)/|D| \cdot \#(c)/|D|} = \log \frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)}$$

Positive pointwise mutual information

- Note that PMI is infinitely small for unseen word–context pairs, and undefined for unseen target words.
- In **positive pointwise mutual information (PPMI)**, all negative and undefined values are replaced by zero:

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

- Because PPMI assigns high values to rare events, it is advisable to apply a count threshold, or to smooth the probabilities.

Computing PPMI on a word–context matrix

	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

Example from Jurafsky and Martin (2019)

Computing PPMI on a word–context matrix

	aardvark	computer	data	pinch	result	sugar
apricot	$\frac{0/19}{2/19 \cdot 0/19}$	$\frac{0/19}{2/19 \cdot 3/19}$	$\frac{0/19}{2/19 \cdot 7/19}$	$\frac{1/19}{2/19 \cdot 2/19}$	$\frac{0/19}{2/19 \cdot 5/19}$	$\frac{1/19}{2/19 \cdot 2/19}$
pineapple	$\frac{0/19}{2/19 \cdot 0/19}$	$\frac{0/19}{2/19 \cdot 3/19}$	$\frac{0/19}{2/19 \cdot 7/19}$	$\frac{1/19}{2/19 \cdot 2/19}$	$\frac{0/19}{2/19 \cdot 5/19}$	$\frac{1/19}{2/19 \cdot 2/19}$
digital	$\frac{0/19}{4/19 \cdot 0/19}$	$\frac{2/19}{4/19 \cdot 3/19}$	$\frac{1/19}{4/19 \cdot 7/19}$	$\frac{0/19}{4/19 \cdot 2/19}$	$\frac{1/19}{4/19 \cdot 5/19}$	$\frac{0/19}{4/19 \cdot 2/19}$
information	$\frac{0/19}{11/19 \cdot 0/19}$	$\frac{1/19}{11/19 \cdot 3/19}$	$\frac{6/19}{11/19 \cdot 7/19}$	$\frac{0/19}{11/19 \cdot 2/19}$	$\frac{4/19}{11/19 \cdot 5/19}$	$\frac{0/19}{11/19 \cdot 2/19}$

Example from Jurafsky and Martin (2019)

Computing PPMI on a word–context matrix

	aardvark	computer	data	pinch	result	sugar
apricot	undefined	log 0.00	log 0.00	log 4.75	log 0.00	log 4.75
pineapple	undefined	log 0.00	log 0.00	log 4.75	log 0.00	log 4.75
digital	undefined	log 3.17	log 0.68	log 0.00	log 0.95	log 0.00
information	undefined	log 0.58	log 1.48	log 0.00	log 1.38	log 0.00

Example from Jurafsky and Martin (2019)

Computing PPMI on a word–context matrix

	aardvark	computer	data	pinch	result	sugar
apricot	0.00	0.00	0.00	2.25	0.00	2.25
pineapple	0.00	0.00	0.00	2.25	0.00	2.25
digital	0.00	1.66	0.00	0.00	0.00	0.00
information	0.00	0.00	0.57	0.00	0.47	0.00

Example from Jurafsky and Martin (2019)

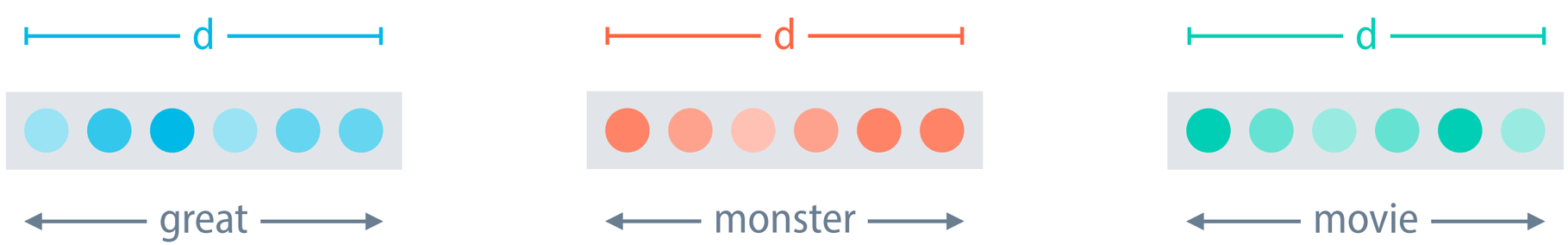
This lecture

- Introduction to word embeddings
- Learning word embeddings via matrix factorization
- Learning word embeddings with neural networks
- The skip-gram model
- Subword models
- Contextualized word embeddings

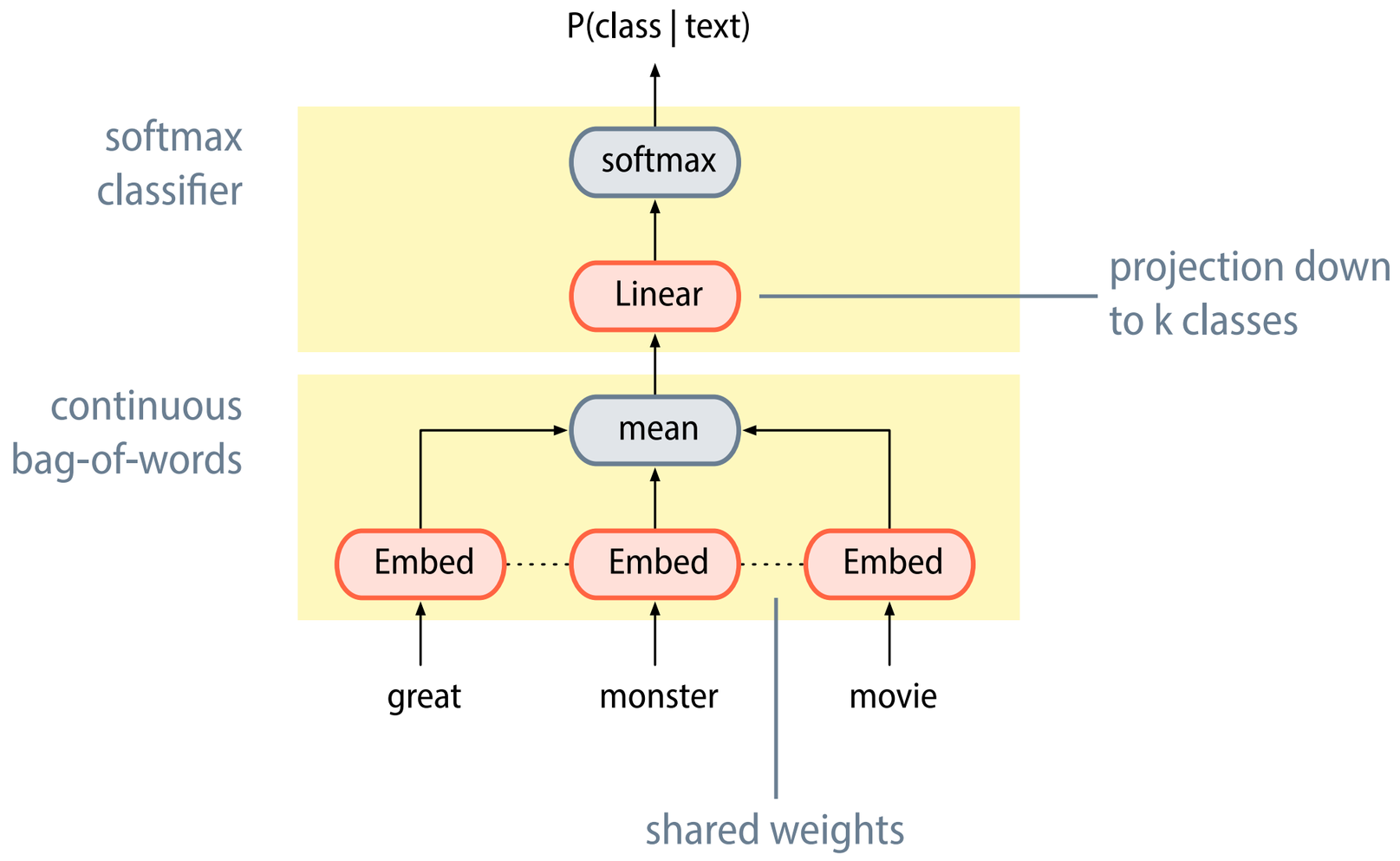
Learning word embeddings with neural networks

Embedding layers

- In neural networks, word embeddings are implemented through **embedding layers**.
- An embedding layer implements a mapping from a vocabulary of words V to a d-dimensional vector space.



Continuous bag-of-words classifier



Using embedding layers

- An embedding layer can be viewed as a linear transformation from one-hot vectors into the d-dimensional embedding space.

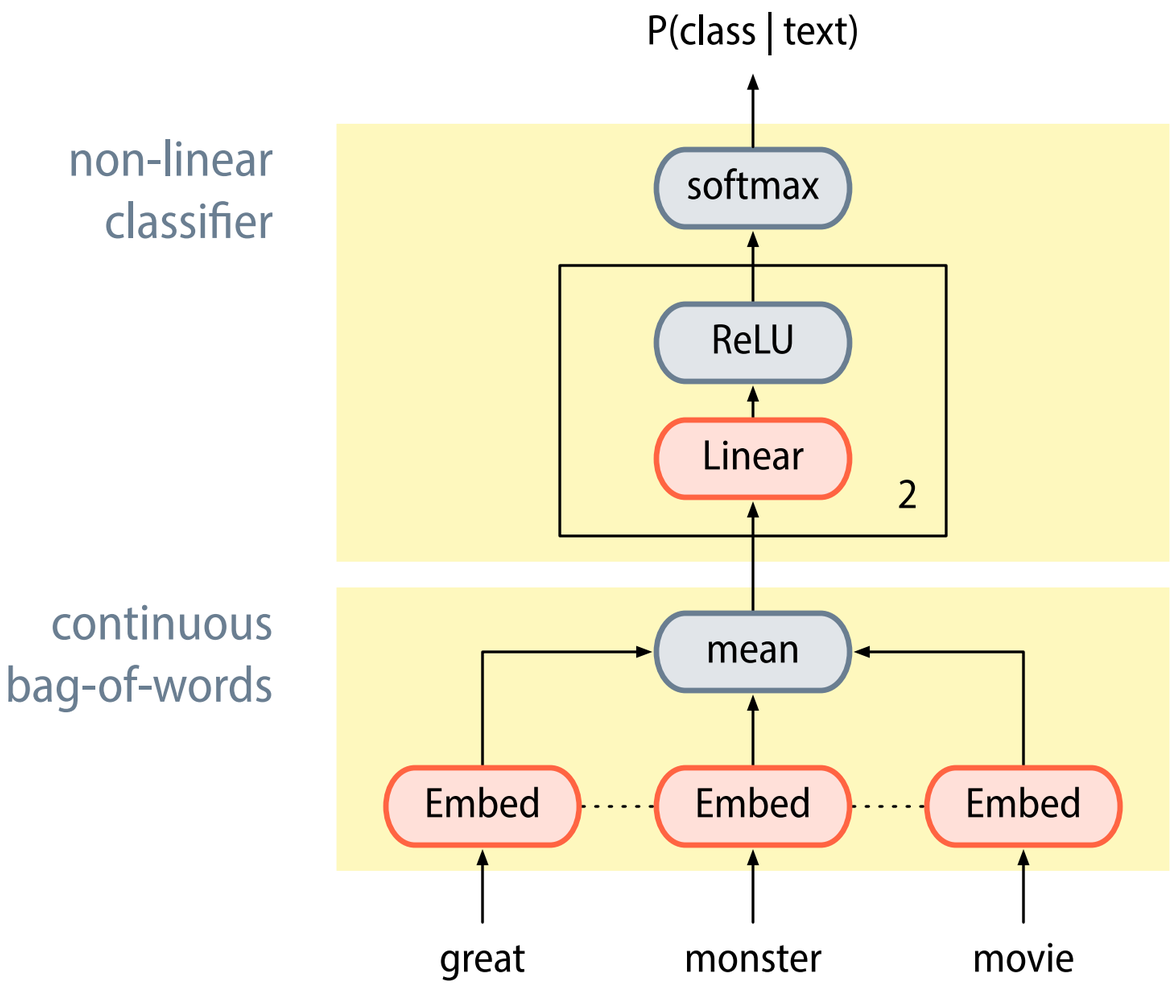
  values of the embedding vector = weights of the linear layer

- From a practical point of view, embeddings are more efficiently implemented using lookup tables.

- Embedding layers are initialised with random values, and then updated through backpropagation, as any other layer.

  typical choice for initialisation: $N(0, 1)$

Deep Averaging Network



Iyyer et al. (2015)

Task-specific word embeddings

- When we train a continuous bag-of-words classifier, the word embeddings are learned automatically.
- Through learning, the word embeddings are optimised for the task at hand – in this case, the classification task.
- Words can ‘mean’ different things in different tasks, and will therefore get different embeddings in different tasks.

  Example: sentiment classification – product classification

Transfer learning

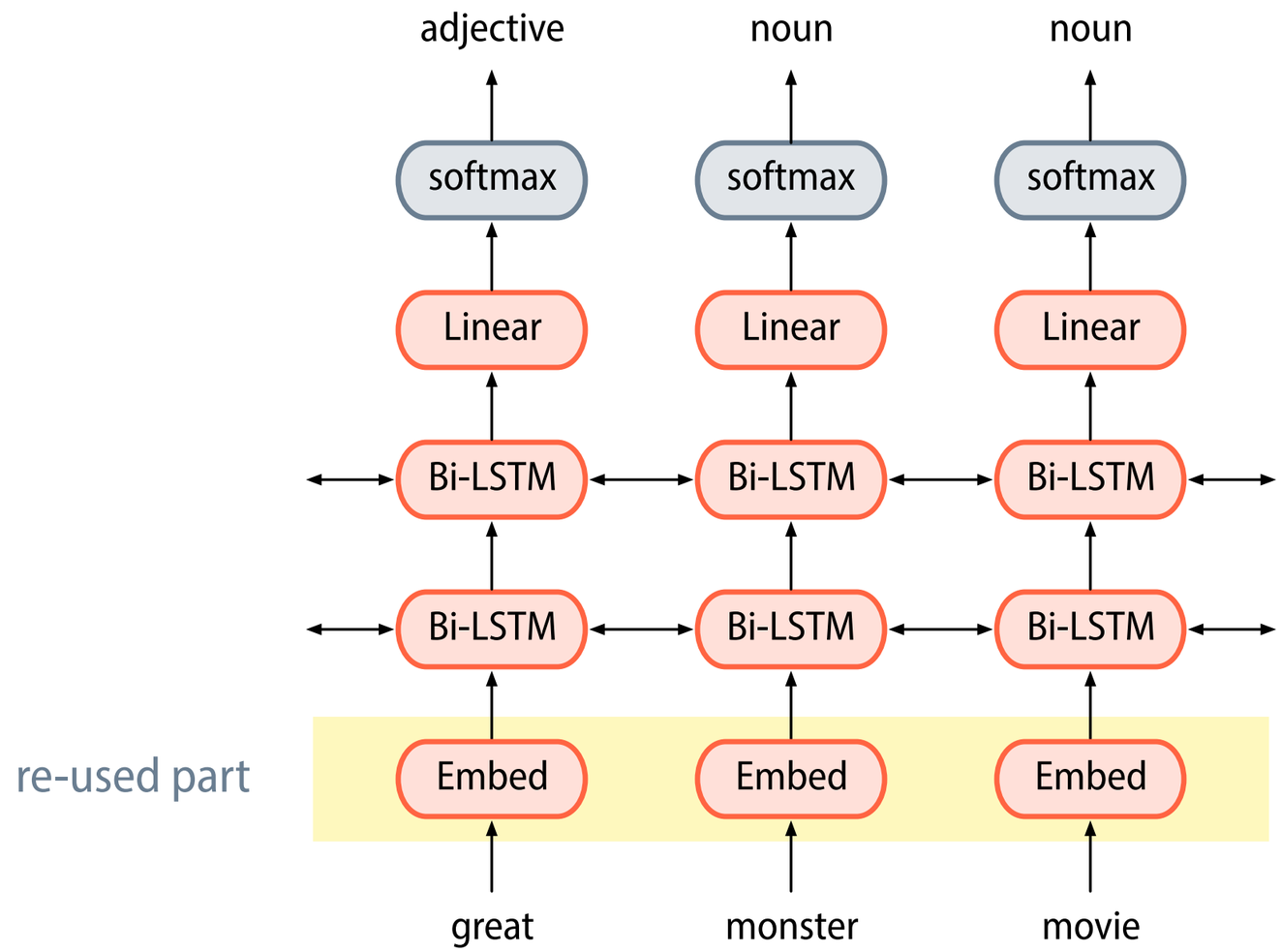
- **Transfer learning** focuses on re-using knowledge gained while solving some previous task when solving the next task.

  speed up training, reduce the need for training data

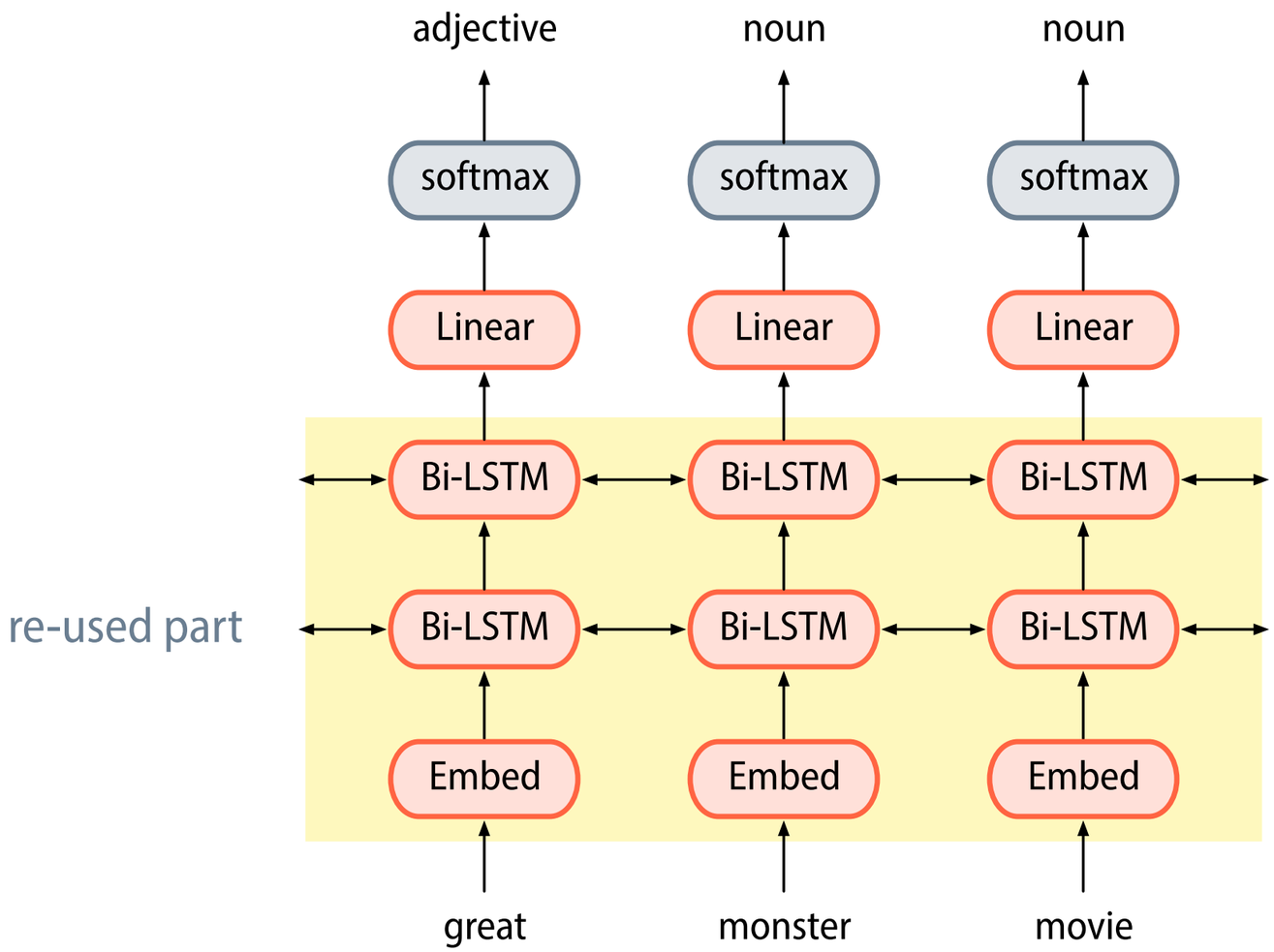
- In the context of deep learning, transfer learning is typically implemented by re-using some part of a trained model.

- In particular, we could try to re-use the embedding layers, instead of learning embeddings from scratch for each task.

Pre-training word embeddings



Pre-training word embeddings



What pre-training tasks should we use?

- We want to learn representations that are generally useful, so we prefer pre-training tasks that are general.
- We need to find training data for the pre-training tasks, so we prefer tasks for which data is abundant.

  ideal candidate: raw text

- The standard pre-training task for word embeddings is to predict co-occurrences.

  Remember the distributional principle!

Co-occurrence matrix

context words

target words	butter	cake	cow	deer
cheese	12	2	1	0
bread	5	5	0	0
goat	0	0	6	1
sheep	0	0	7	5

Re-using pre-trained word embeddings

- Use the pre-trained embeddings to initialise the embedding layers of the task-specific network.

- **Alternative 1:** Freeze the weights of the embedding layers, to prevent them from being updated during training.

- **Alternative 2:** Update the weights of the embedding layers, thereby fine-tuning the embeddings for the task at hand.

This lecture

- Introduction to word embeddings
- Learning word embeddings via matrix factorization
- Learning word embeddings with neural networks
- The skip-gram model
- Subword models
- Contextualized word embeddings

The skip-gram model

The skip-gram model

- The **skip-gram model** is one of two word embedding models implemented in Google's word2vec software.

  second model: continuous bag-of-words (CBOW)

- In the context of this model, a **skip-gram** is a pair of words from a text that is separated by at most k other words.

- The word embeddings are obtained as a by-product of the task to predict one word in the skip-gram from the other word.

Mikolov et al. (2013)

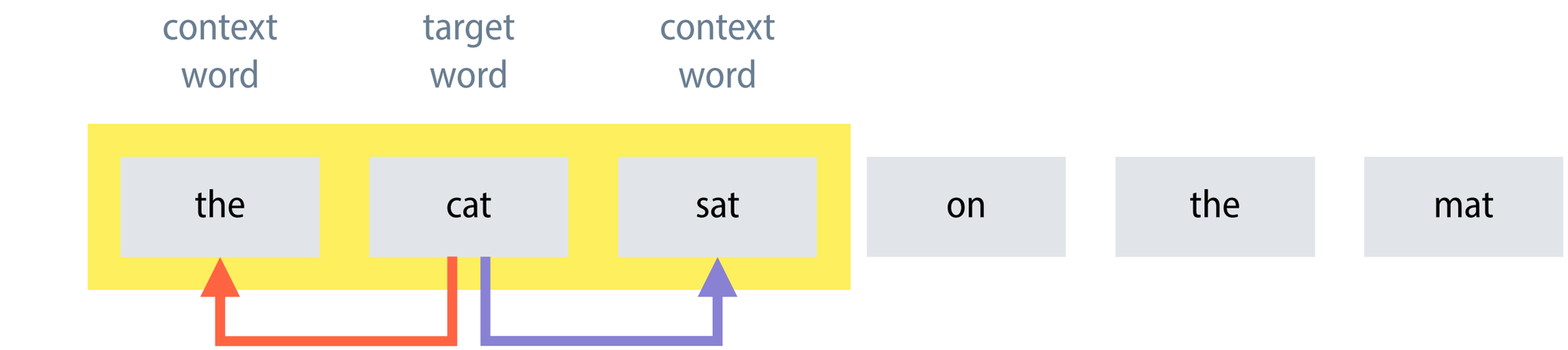
Overview of the skip-gram algorithm

- We move a small, symmetric window over the words in a text. Each window contains a target word w and context words c.
- At each position, we use the similarity of the current word vectors for w and c to calculate a conditional probability $P(c|w)$.
- We update the word vectors to maximise this probability.

  stochastic gradient descent

Mikolov et al. (2013)

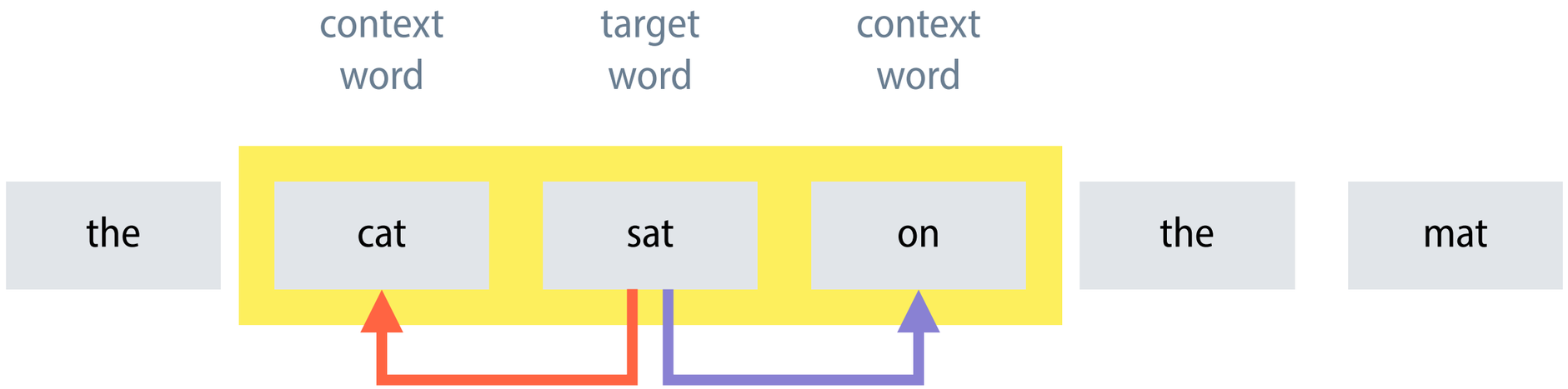
The skip-gram model



$P(\text{the} \mid \text{cat}) \propto \boldsymbol{v}'^{\top}_{\text{the}} \boldsymbol{v}_{\text{cat}}$ $P(\text{sat} \mid \text{cat}) \propto \boldsymbol{v}'^{\top}_{\text{sat}} \boldsymbol{v}_{\text{cat}}$

target word vectors	$\boldsymbol{v}_{\text{cat}}$	$\boldsymbol{v}_{\text{mat}}$	$\boldsymbol{v}_{\text{on}}$	$\boldsymbol{v}_{\text{sat}}$	$\boldsymbol{v}_{\text{the}}$
context word vectors	$\boldsymbol{v}'_{\text{cat}}$	$\boldsymbol{v}'_{\text{mat}}$	$\boldsymbol{v}'_{\text{on}}$	$\boldsymbol{v}'_{\text{sat}}$	$\boldsymbol{v}'_{\text{the}}$

The skip-gram model



$P(\text{cat} \mid \text{sat}) \propto \boldsymbol{v}'^{\top}_{\text{cat}} \boldsymbol{v}_{\text{sat}}$ $P(\text{on} \mid \text{sat}) \propto \boldsymbol{v}'^{\top}_{\text{on}} \boldsymbol{v}_{\text{sat}}$

target word vectors	$\boldsymbol{v}_{\text{cat}}$	$\boldsymbol{v}_{\text{mat}}$	$\boldsymbol{v}_{\text{on}}$	$\boldsymbol{v}_{\text{sat}}$	$\boldsymbol{v}_{\text{the}}$
context word vectors	$\boldsymbol{v}'_{\text{cat}}$	$\boldsymbol{v}'_{\text{mat}}$	$\boldsymbol{v}'_{\text{on}}$	$\boldsymbol{v}'_{\text{sat}}$	$\boldsymbol{v}'_{\text{the}}$

The basic skip-gram model in detail (1)

- We maintain two separate vector representations: one for target words and one for context words. Initially, they are random.
- The probability of a context word c given a target word w is defined using the softmax function:

vector representation for context words

vector representation for target words

$$P(c \mid w; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{v}'_c{}^{\top} \boldsymbol{v}_w)}{\sum_{x \in V} \exp(\boldsymbol{v}'_x{}^{\top} \boldsymbol{v}_w)}$$

all parameters of the model

The basic skip-gram model in detail (2)

- To maximise the conditional probabilities, we *minimise* the average negative log likelihood:

$$J(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P(w_{i+j} \mid w_i ; \boldsymbol{\theta})$$

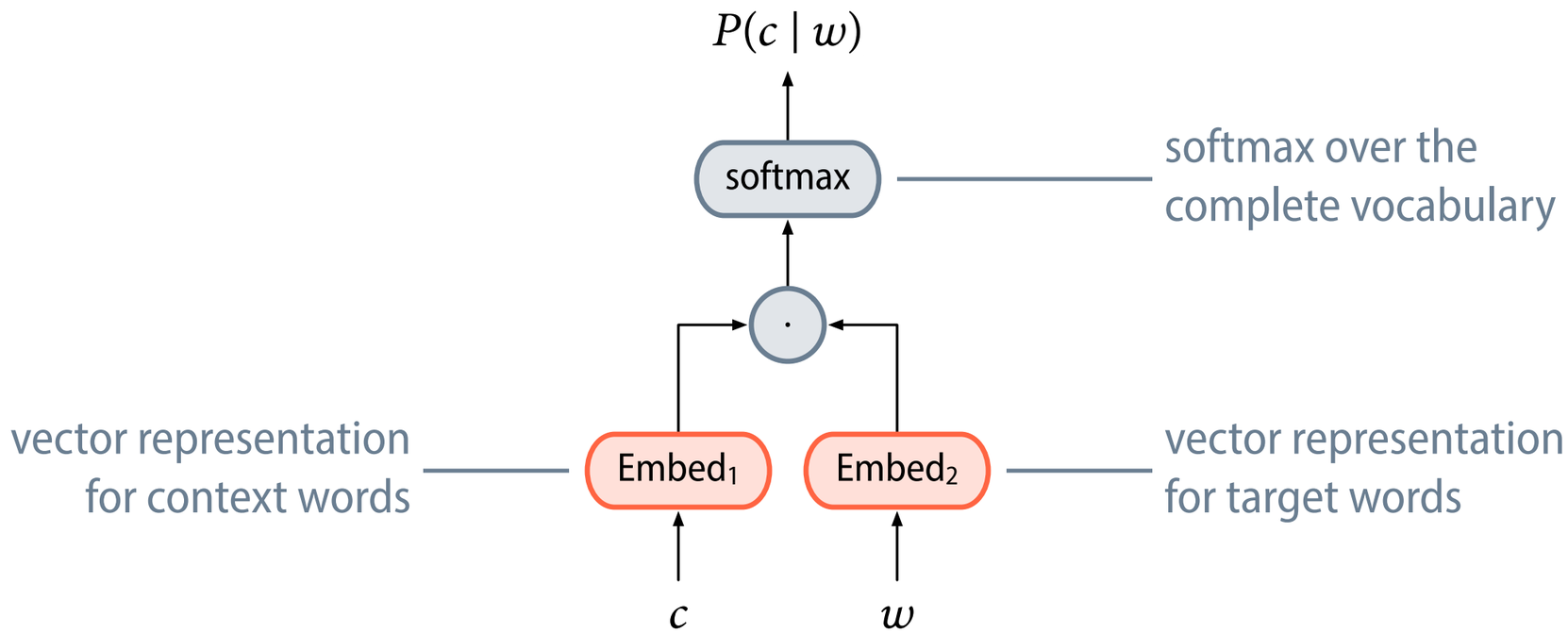
all parameters of the model

length of the text

size of each window

- As the final word vector for each word, we take the sum of its target-specific and its context-specific vector representation.

The skip-gram model as a neural network (1)



Problems with the basic skip-gram model

- Computing the softmax is impractical: For each position in the text, we need to sum over the complete vocabulary.

- **Idea 1:** Decompose the standard softmax computation into a tree-like structure of simpler computations.

  hierarchical softmax

- **Idea 2:** Instead of maximising the conditional probabilities directly, maximise simpler quantities that approximate them.

  negative sampling

Skip-gram with negative sampling (SGNS)

- Maximise the probability of actual word–context pairs, and minimise the probability of randomly drawn samples.

$$J(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{\substack{-m\le j\le m\\ j\neq 0}} \log\sigma\left(\boldsymbol{v}'_{w_{i+j}}{}^{\top}\boldsymbol{v}_{w_i}\right) + \sum_{c\sim D}\log\sigma\left(-\boldsymbol{v}'_{c}{}^{\top}\boldsymbol{v}_{w_i}\right)$$

length of the text — logistic function — pseudo-negative samples

- The pseudo-negative samples are drawn from $D(c) \propto \#(c)^{\alpha}$, where α is a hyperparameter (default value: 0.75).

SGNS in detail

- To reduce the influence of very frequent words (and speed up learning), discard the window around a word w with probability

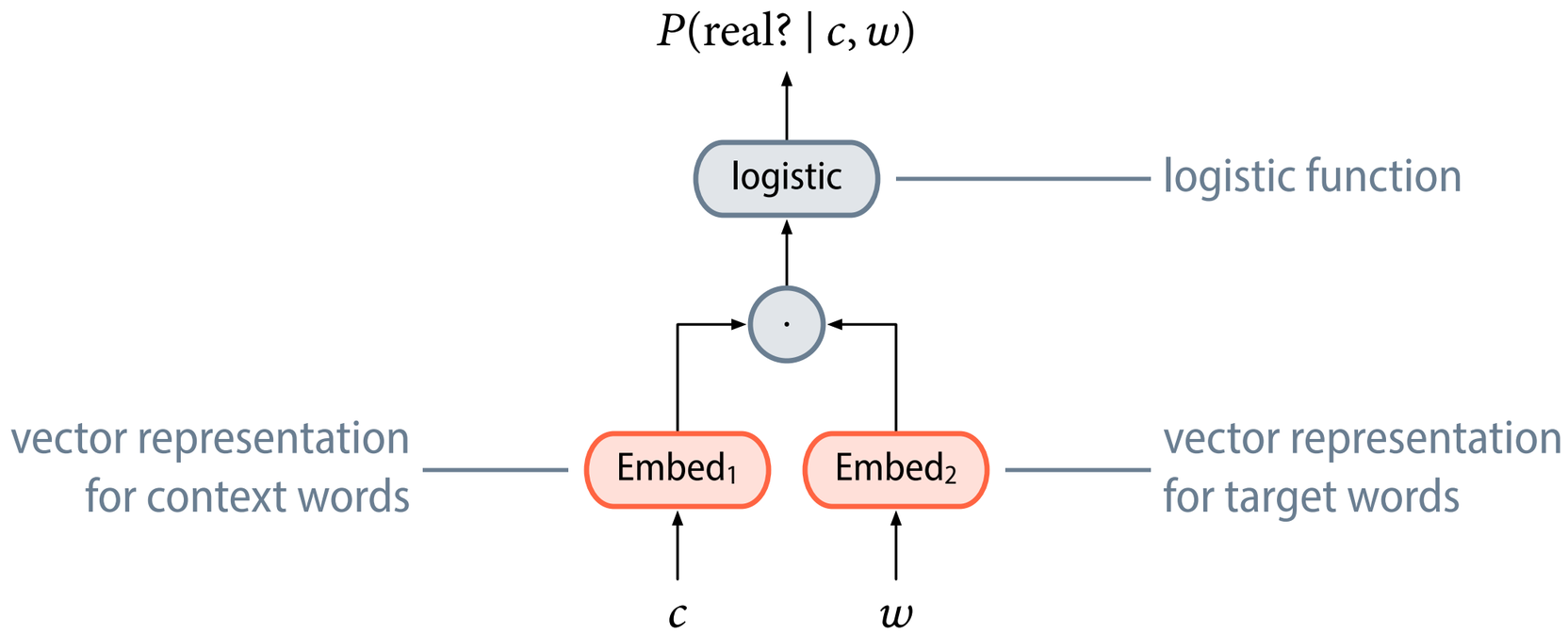
$$P(w) = \max\left(0, 1 - \sqrt{t/\#(w)}\right)$$

where t is a chosen threshold (default value: 10^{-3}).

- The window size is not constant; instead, window sizes up to the maximal size m are sampled with uniform probability.

  As a consequence, far-away context words will get less influence.

The skip-gram model as a neural network (2)



Connecting the two worlds

- Factorisation-based methods and neural networks take two seemingly very different approaches to word embeddings.
- However, a careful analysis reveals that the skip-gram model is implicitly factorising a PMI word–context matrix.

  Each entry is shifted by $-\log k$, where k is the number of negative samples.

Levy and Goldberg (2014)

This lecture

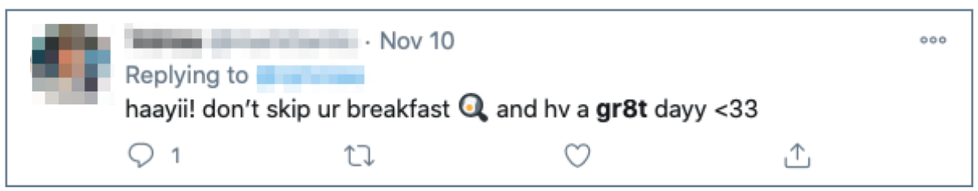
- Introduction to word embeddings
- Learning word embeddings via matrix factorization
- Learning word embeddings with neural networks
- The skip-gram model
- Subword models
- Contextualized word embeddings

Subword models

Subword models

- Word embeddings as we have covered them so far are essentially lookup tables with a fixed vocabulary.
- In practical applications, we will often encounter words that we do not have an embedding for.

  Remember Heap's law!



- One way to deal with this problem is to use models that can work at the subword level, such as character-based models.

Rationale for subword models

- Working with subword units makes sense from a linguistic point of view, as subword units resemble morphemes.

  Morpheme+s are the small+est mean+ing+ful unit+s of language.

- Features at the subword level have been shown to be very predictive in non-neural models for e.g. part-of-speech tagging.

  Does the word end in *-tion* or *-ism*? Then it's probably a noun!

Different types of subword models

- Use the same types of architectures that we find in word-based models, but apply them to subword units.
- Augment the architectures of word-based models with submodels that compose word representations from characters.
- Give up on word-based architectures altogether and process language as a connected sequence of characters.

WordPiece tokenisation in BERT

Source: The Muppet Wiki

Raw text

The history of morphological analysis dates back to the ancient Indian linguist Pāṇini, who formulated the 3,959 rules of Sanskrit morphology in the text Aṣṭādhyāyī by using a constituency grammar.

WordPiece tokenisation

The history of m ##or ##phological analysis dates back to the ancient Indian linguist P ##ā ##ṇ ##ini , who formulated the 3 , 95 ##9 rules of Sanskrit morphology in the text A ##ṣ ##ṭ ##ā ##dh ##y ##ā ##y ##ī by using a constituency grammar .

To obtain a word vector, take the average of the 9 word piece vectors.

Byte Pair Encoding algorithm

- Initialise the word unit vocabulary with all characters.

  plus a special end-of-word marker, here denoted by $

- Generate a new word unit by combining two units from the current vocabulary, increasing vocabulary size by one.
  Choose the new unit as the most frequent pair of adjacent units.

  WordPiece: maximise likelihood under a language model

- Repeat the previous step as long as the vocabulary size does not exceed a maximal size.

Sennrich et al. (2016); Schuster and Nakajima (2012)

Byte Pair Encoding: Example

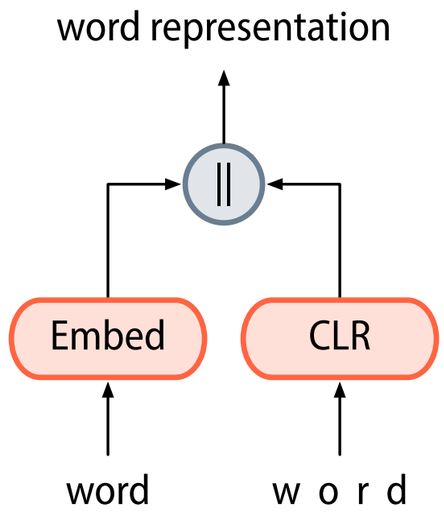
number of occurrences in data

Step	Merged pair	Words	Vocabulary size
0	–	low$/5 lower$/2 new**est**$/6 wid**est**$/3	11
1	es/9	low$ lower$ new**[es]t**$ wid**[es]t**$	12
2	[es]t/9	low$ lower$ new**[est]$** wid**[est]$**	13
3	[est]$/9	**lo**w$ **lo**wer$ new[est$] wid[est$]	14
4	lo/7	**[lo]w**$ **[lo]w**er$ new[est$] wid[est$]	15
5	[lo]w/7	[low]$ [low]er$ **ne**w[est$] wid[est$]	16

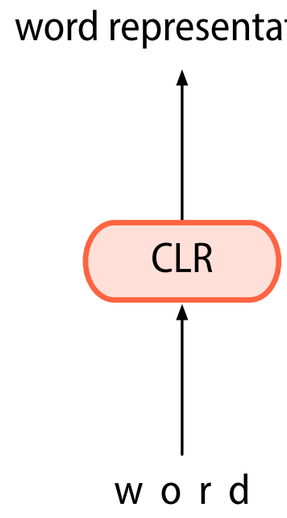
Example from Sennrich et al. (2016)

Composing word representations from characters

Character-level word representations are typically built using convolutional neural networks or recurrent neural networks.



combined (augmented) model



purely character-based model

Composing word representations using CNNs

<pad>	0.00 **0.50**	0.00 **0.10**	0.00 **0.10**
H	0.08 **1.00**	0.95 **0.20**	0.85 **0.20**
o	0.98 **0.50**	0.78 **0.10**	0.02 **0.10**
l	0.32	0.13	0.82
m	0.64	0.28	0.92
e	0.05	0.25	0.77
s	0.88	0.59	0.66
<pad>	0.00	0.00	0.00

1.010		
1.615		
1.520		
1.262		
1.259		
1.257		

Composing word representations using CNNs

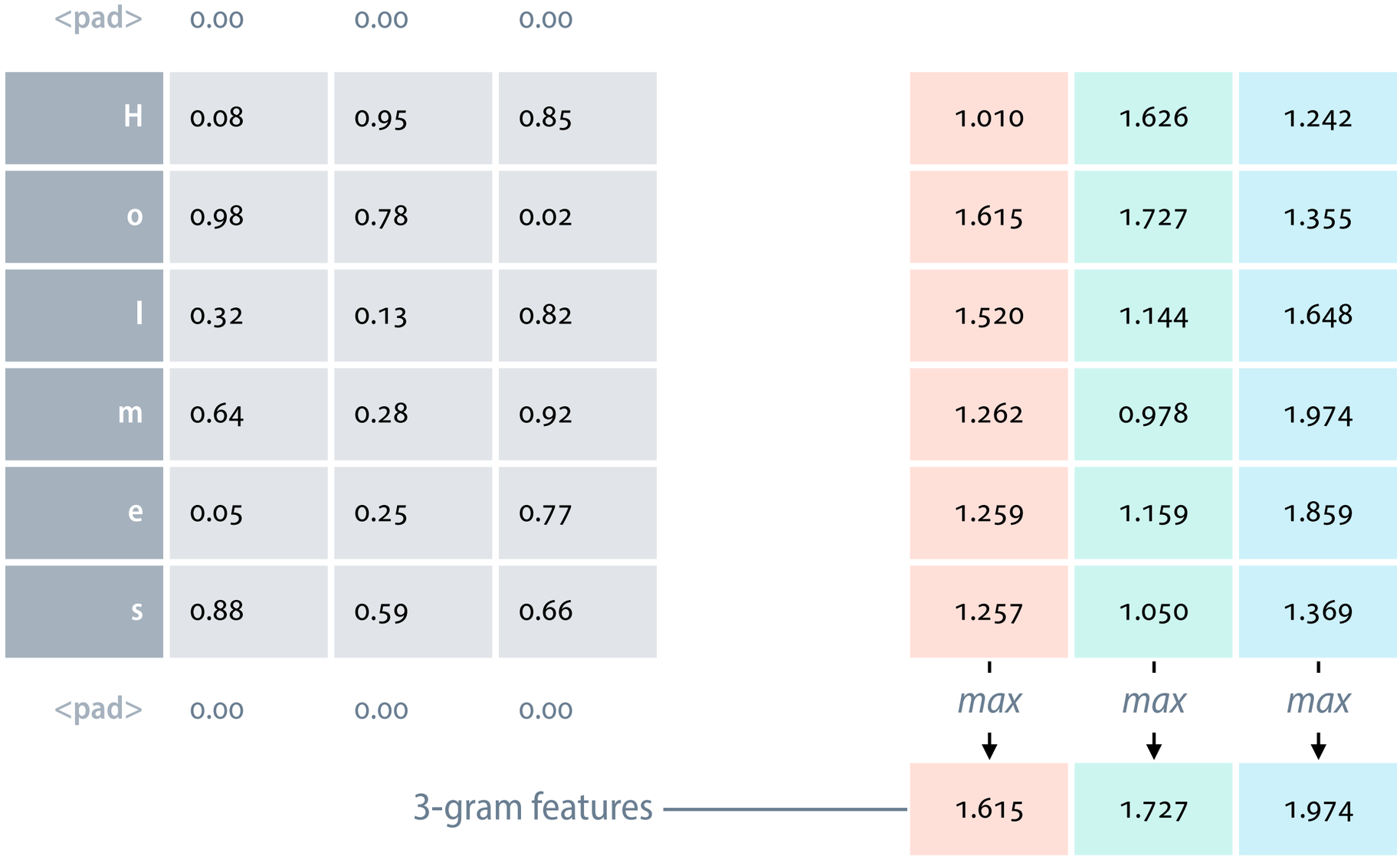
<pad>	0.00 **0.10**	0.00 **0.50**	0.00 **0.10**
H	0.08 **0.20**	0.95 **1.00**	0.85 **0.20**
o	0.98 **0.10**	0.78 **0.50**	0.02 **0.10**
l	0.32	0.13	0.82
m	0.64	0.28	0.92
e	0.05	0.25	0.77
s	0.88	0.59	0.66
<pad>	0.00	0.00	0.00

1.010	1.626	
1.615	1.727	
1.520	1.144	
1.262	0.978	
1.259	1.159	
1.257	1.050	

Composing word representations using CNNs

<pad>	0.00 **0.10**	0.00 **0.10**	0.00 **0.50**			
H	0.08 **0.20**	0.95 **0.20**	0.85 **1.00**	1.010	1.626	1.242
o	0.98 **0.10**	0.78 **0.10**	0.02 **0.50**	1.615	1.727	1.355
l	0.32	0.13	0.82	1.520	1.144	1.648
m	0.64	0.28	0.92	1.262	0.978	1.974
e	0.05	0.25	0.77	1.259	1.159	1.859
s	0.88	0.59	0.66	1.257	1.050	1.369
<pad>	0.00	0.00	0.00			

Composing word representations using CNNs



Training combined models

- In combined models, the character-based representations let us deal with unknown words at test time.
- When training these models, we need to make sure not to overfit to the embeddings of known words.
- In **word dropout**, we replace each word with a dummy ⟨UNK⟩ token with some dropout probability p, e.g.

$$p = \frac{\alpha}{\#(w) + \alpha} \qquad \text{where } \alpha \text{ is a small constant}$$

Kiperwasser and Goldberg (2016)

This lecture

- Introduction to word embeddings
- Learning word embeddings via matrix factorization
- Learning word embeddings with neural networks
- The skip-gram model
- Subword models
- Contextualized word embeddings

Contextualised word embeddings

Contextualised embeddings

- In standard word embeddings, each word is assigned a single word vector, independently of its context.
- Such a model cannot account for **polysemy**, the phenomenon that one and the same word may have multiple meanings.

  The children *play* in the park. The *play* premiered yesterday.

- In **contextualised embeddings**, each token is assigned a representation that is a function of its context.

ELMo – Embeddings from Language Models

- A token is represented as a task-specific, weighted sum of representations derived from a bidirectional language model.

  weights are learned for a specific task

- The basic ELMo model is frozen after pre-training and can complement or replace a standard word embedding layer.

- However, it is often beneficial to fine-tune a pre-trained ELMo model on task-specific data.

Peters et al. (2018)

Muppet character image from The Muppet Wiki

Language modelling

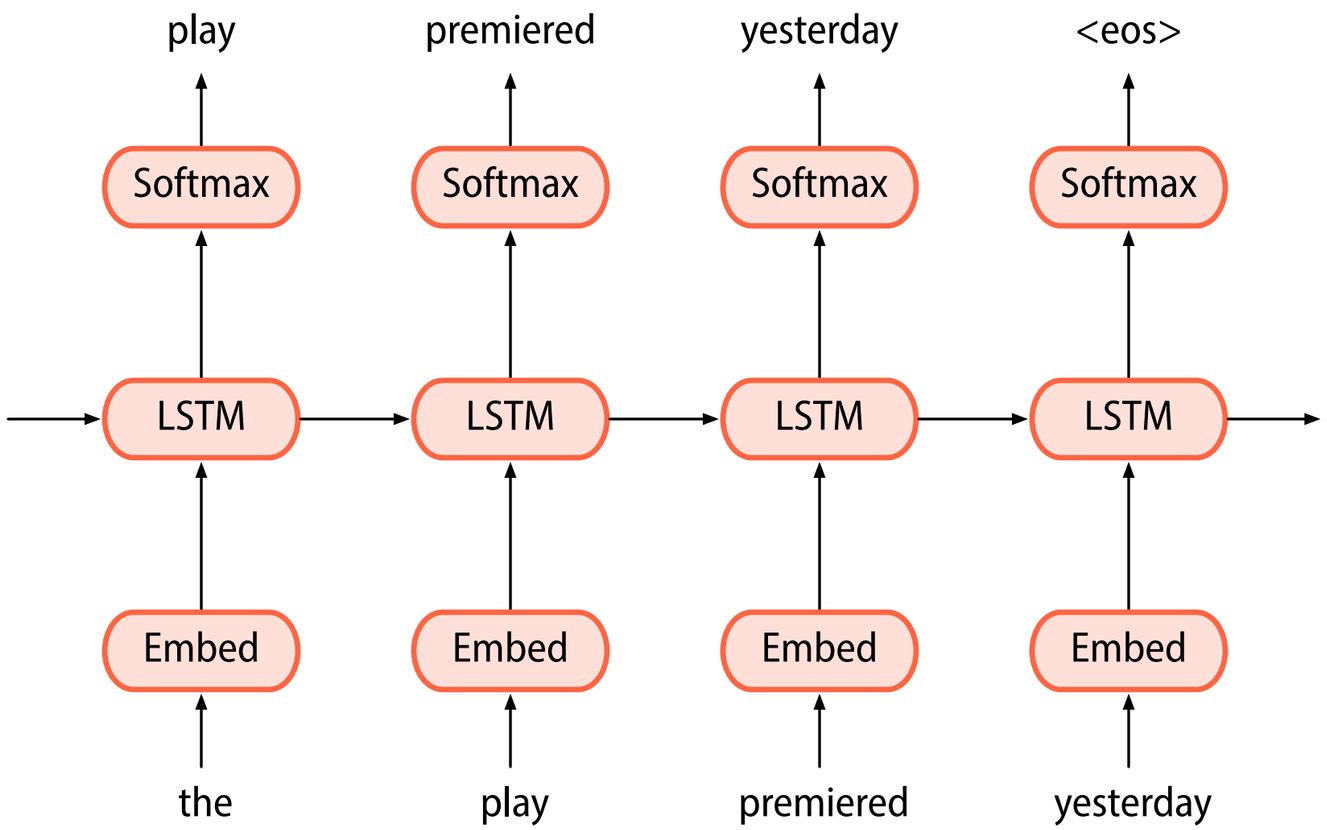
- **Language modelling** is the task of predicting which word comes next in a sequence of words.
- More formally, given a sequence of words $w_1, \ldots, w_t$, we want to know the probability of the next word, w_{t+1}:

$$P(w_{t+1} \mid w_1, \ldots, w_t)$$

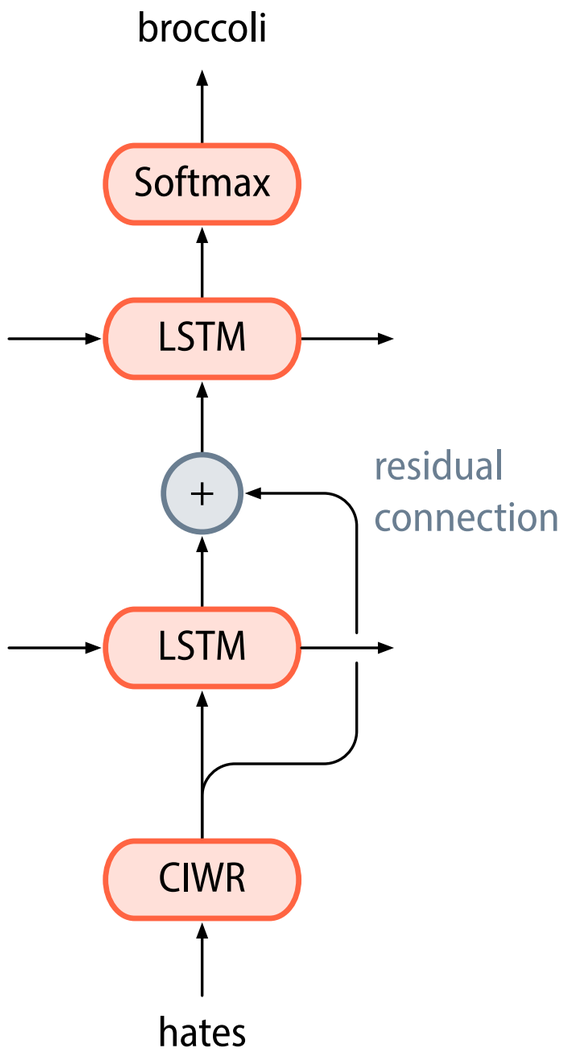
- Here we are assuming that w_{t+1} comes from a fixed vocabulary V.

  This allows language modelling to be treated as a classification task.

LSTM language model



ELMo architecture



- context-insensitive word representation using character convolutions

  followed by 2 highway layers, linear projection

- bidirectional LSTM layers with a residual connection between the layers

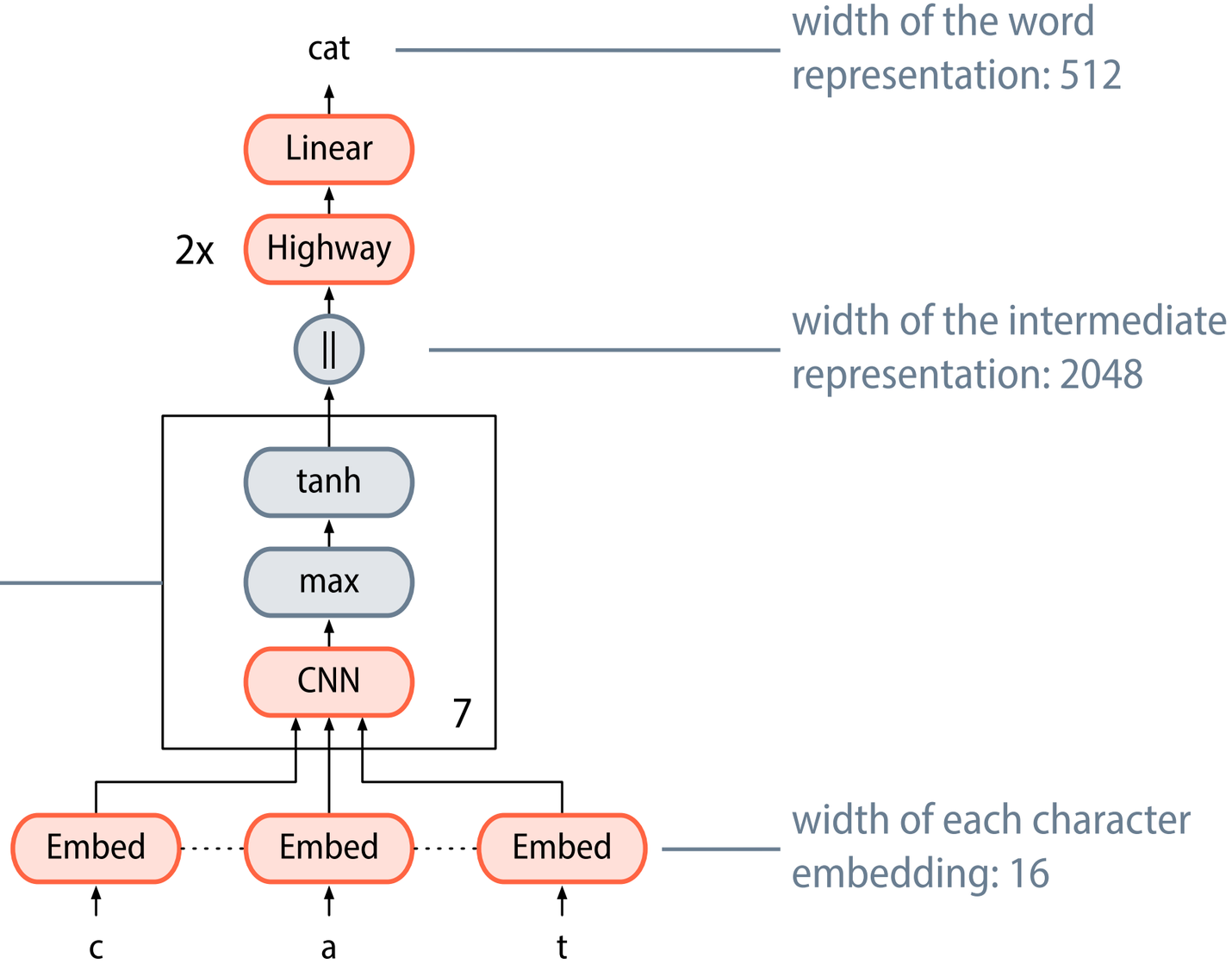
  4,096 hidden units; projected down to 512 units

- final softmax layer computes a probability distribution over the next tokens

Word representations in ELMo

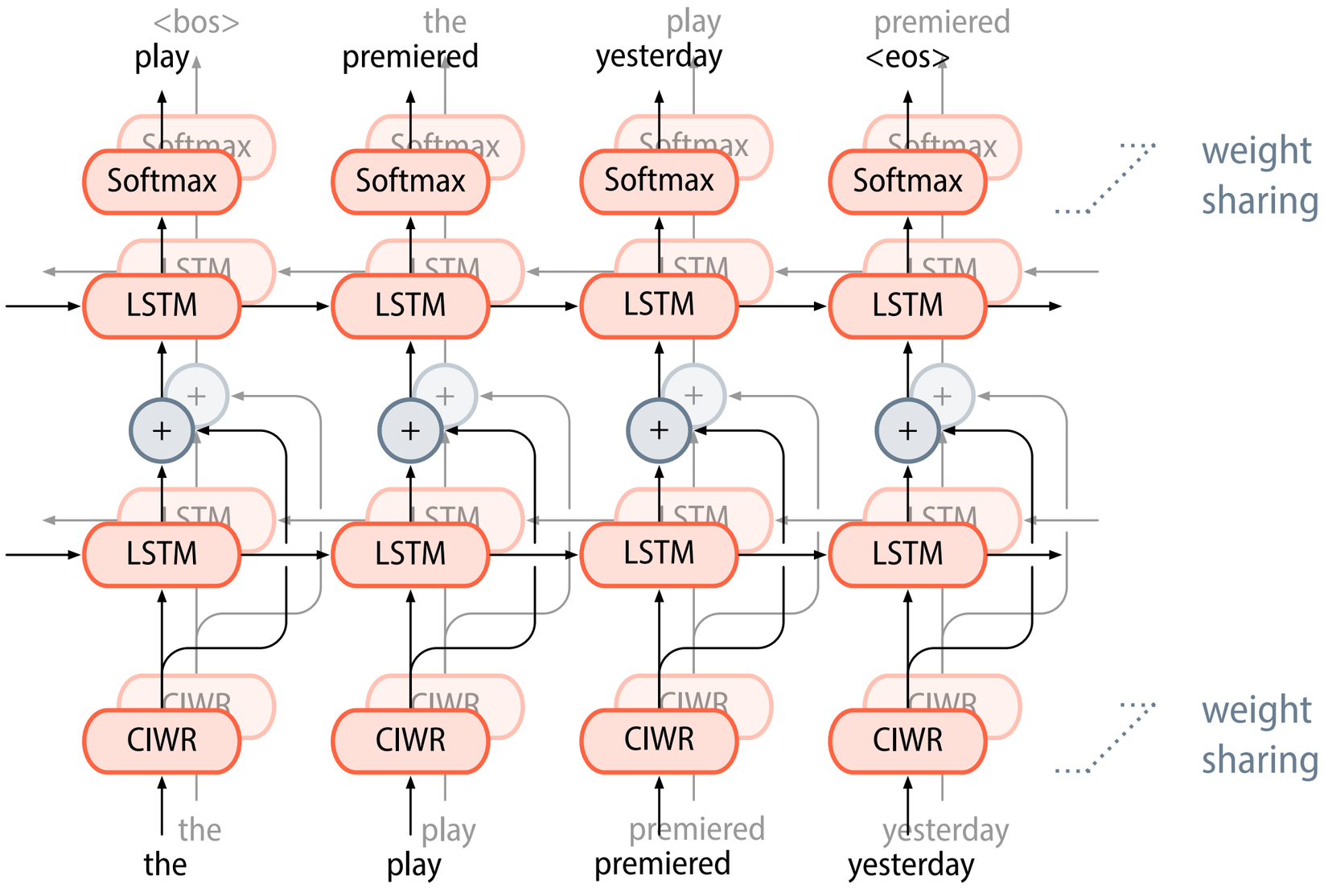
Filter specification

Width	Channels
1	32
2	32
3	64
4	128
5	256
6	512
7	1024



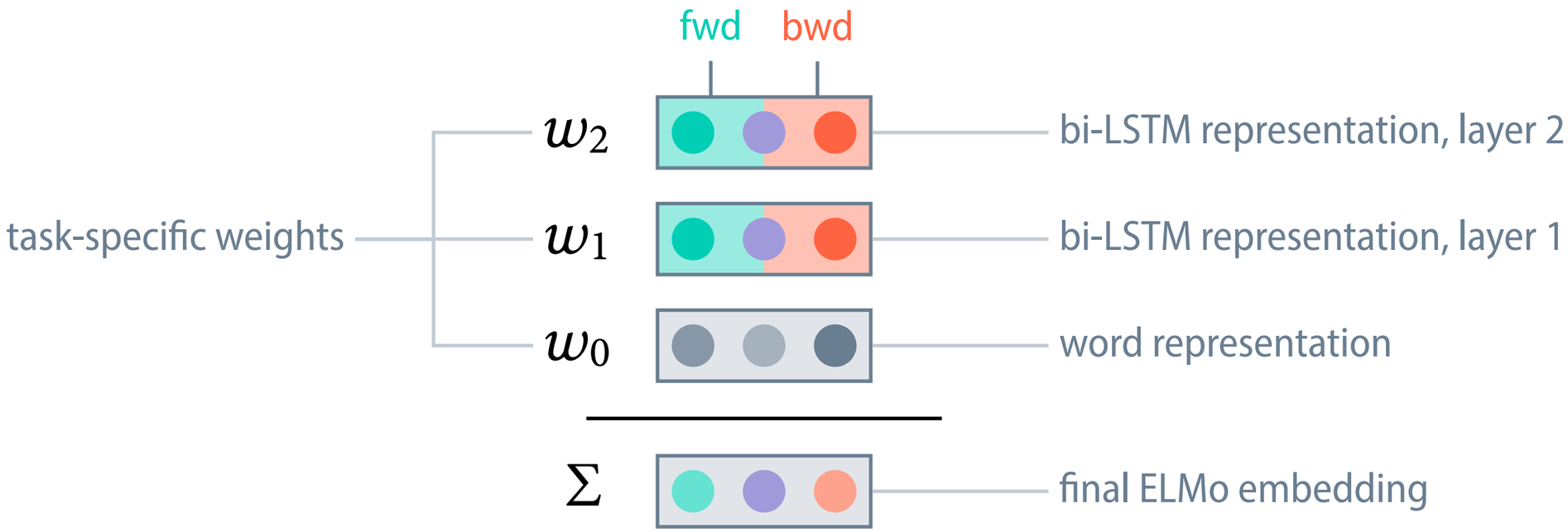
Peters et al. (2018)

Bidirectional language model



ELMo – Embeddings from Language Models

ELMo is a task-specific weighted sum of the
intermediate representations in the bidirectional language model.



Relative improvements by using ELMo embeddings

Task	Baseline	+ ELMo	Relative increase
Question answering (SQuAD)	81.1	85.8	24.9%
Coreference resolution (Coref)	67.2	70.4	9.8%
Sentiment analysis (SST-5)	51.4	54.7	6.8%
Textual entailment (SNLI)	88.0	88.7	5.8%

BERT – Bidirectional Encoder Representations from Transformers

- uses a transformer architecture instead of RNNs
- uses a masked language model pre-training objective
- significant improvements over existing models, including ELMo

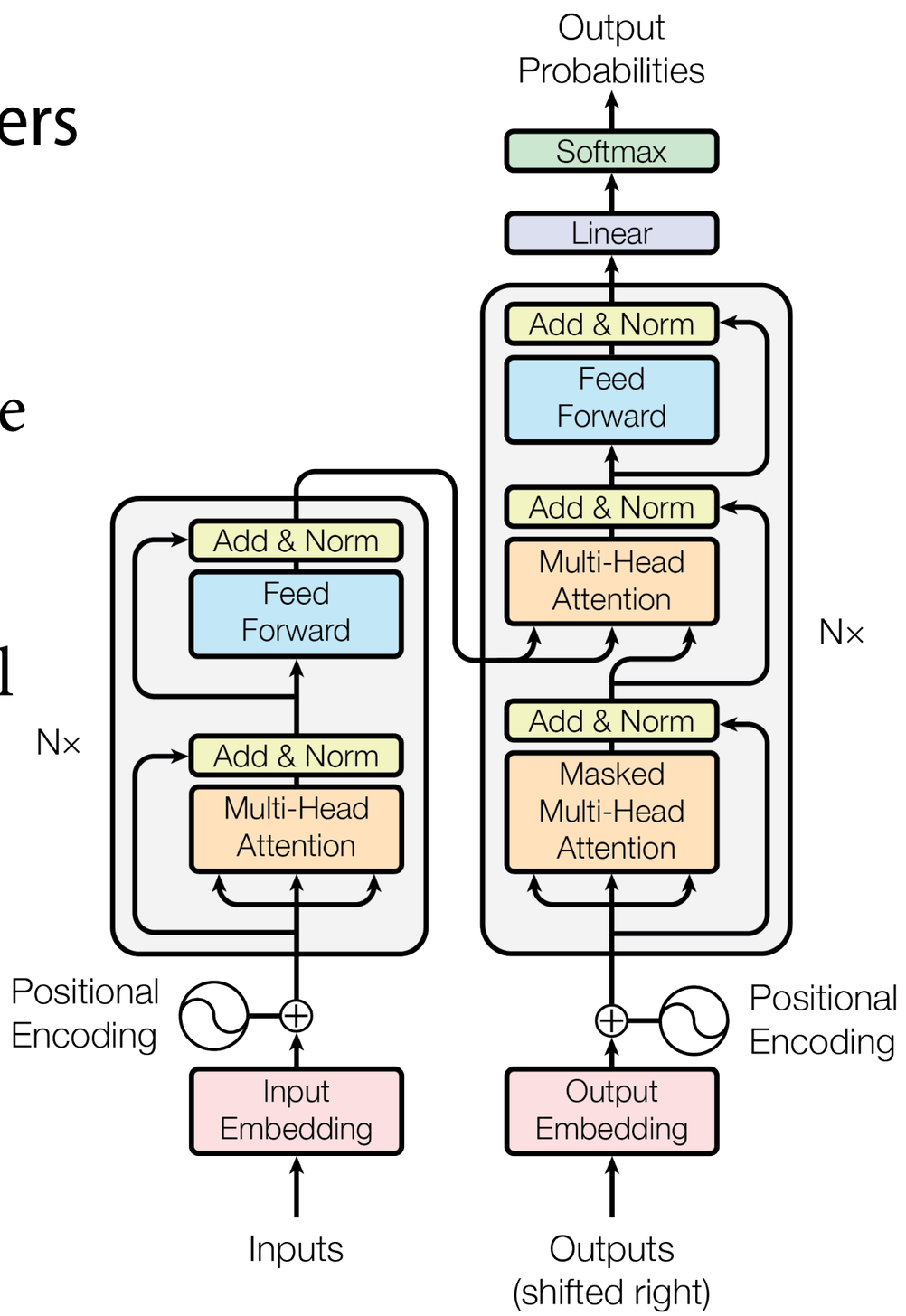


Figure from Vaswani et al. (2017)

Muppet character image from The Muppet Wiki

This lecture

- Introduction to word embeddings
- Learning word embeddings via matrix factorization
- Learning word embeddings with neural networks
- The skip-gram model
- Subword models
- Contextualized word embeddings