Text Mining (2019)

Word embeddings

Marco Kuhlmann
Department of Computer and Information Science

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)

Word embeddings

- A **word embedding** is a mapping of words to points in a vector space such that nearby words (points) are similar in terms of their distributional properties.

  … and hence, by the distributional principle, have similar meanings

- This idea is similar to the vector space model of information retrieval, where the dimensions of the vector space correspond to the terms that occur in a document.

  points = documents, nearby points = similar topic

Lin et al. (2015)

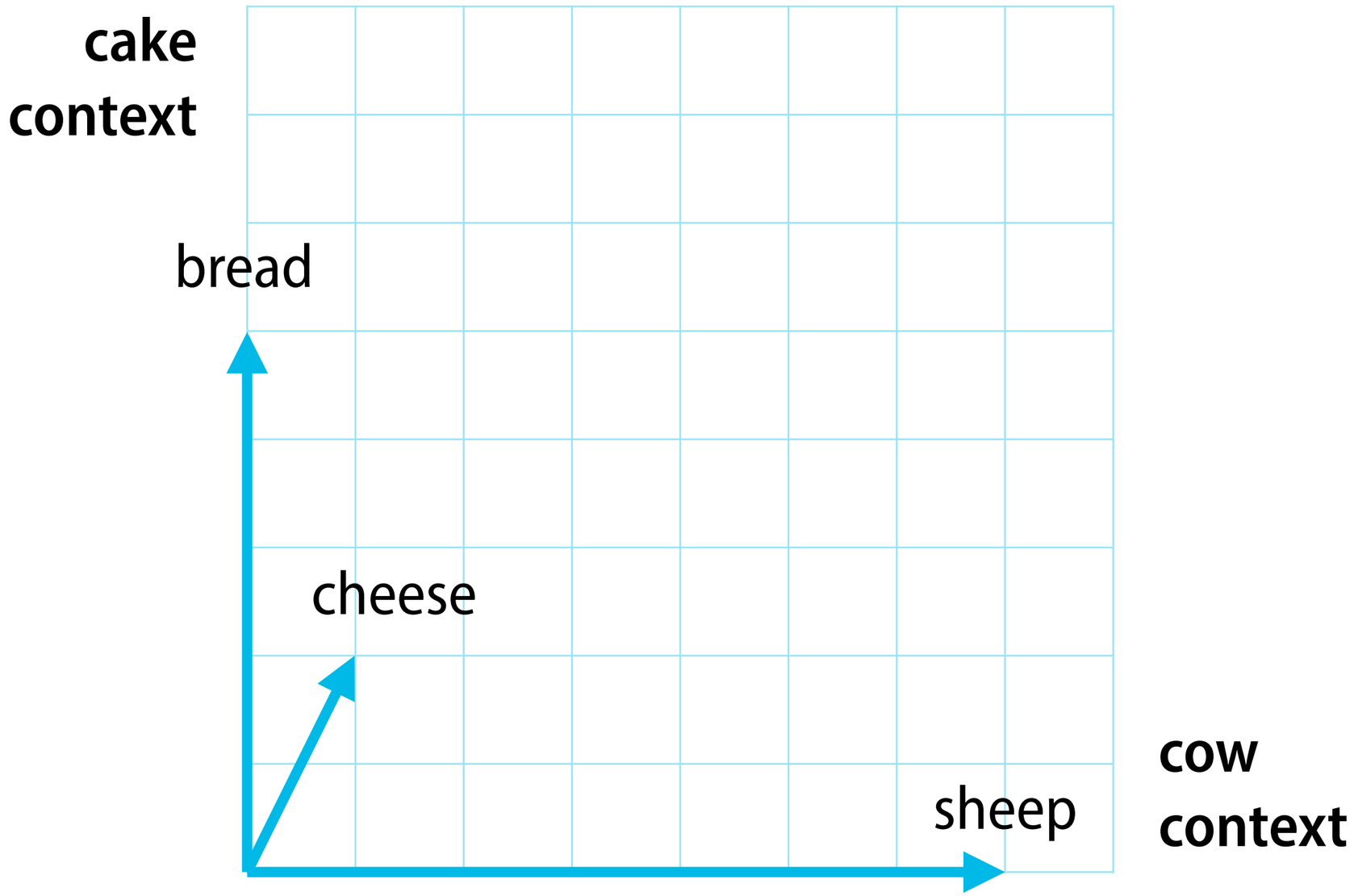
Co-occurrence matrix

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

Co-occurrence matrix

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

From co-occurrences to word vectors



Sparse vectors versus dense vectors

- The rows of co-occurrence matrices are long and sparse.

  length corresponds to number of context words = on the order of 10^4

- State-of-the-art word embeddings that are short and dense.

  length on the order of 10^2

- The intuition is that such vectors may be better at capturing generalisations, and easier to use in machine learning.

Simple applications of word embeddings

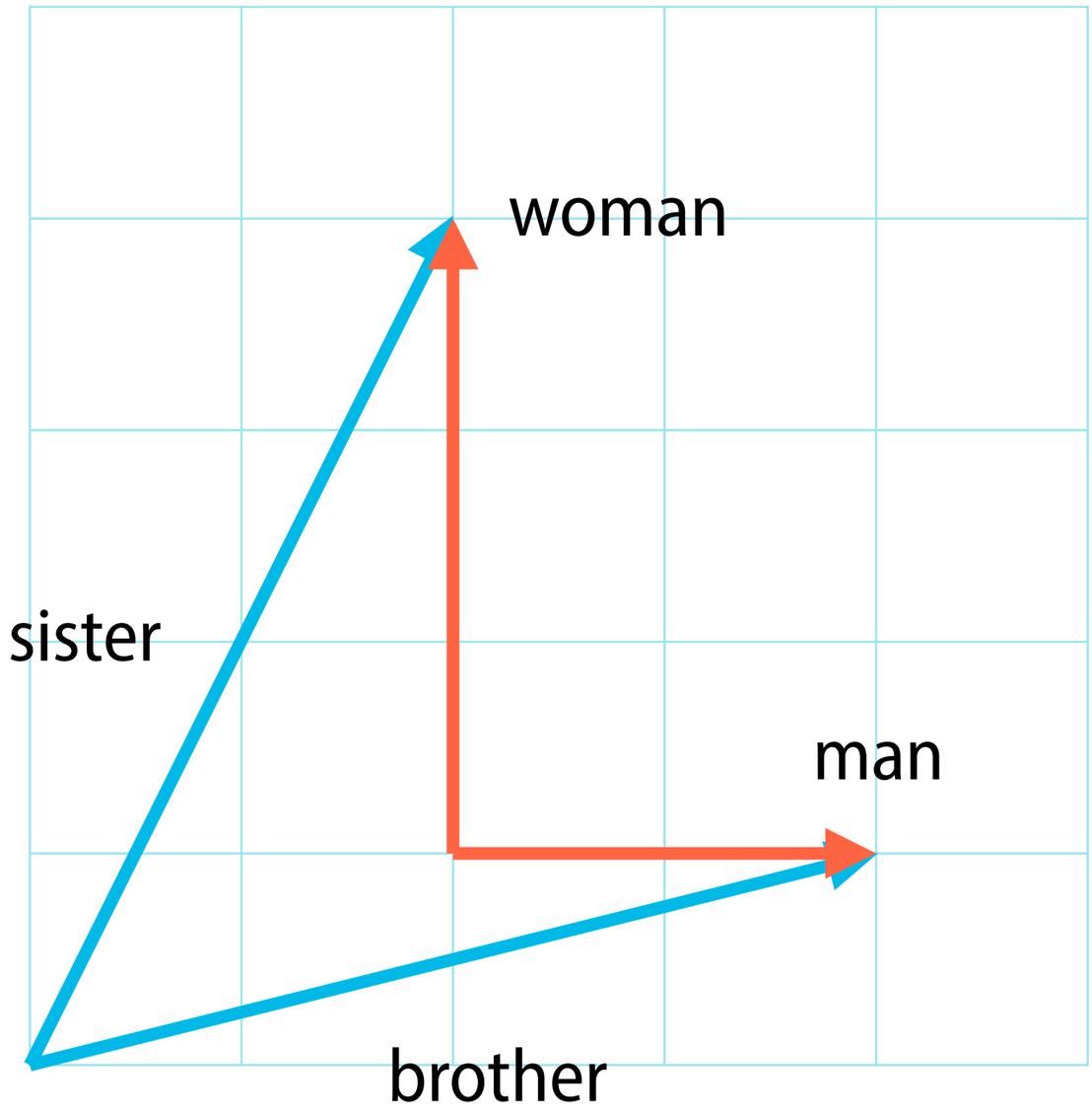
- finding similar words

  What words are most similar to *cheese*?

- answering ‘odd one out’-questions

  Which one is odd: *lunch*, *breakfast*, *dinner*, *car*?

Compositional structure of word embeddings



This week's lab: 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)

Obtaining word embeddings

- Word embeddings can be easily trained from any text corpus using available tools.

  word2vec, Gensim

- Pre-trained word vectors for English, Swedish, and various other languages are available for download.

  word2vec, GloVe, Polyglot project, spaCy

Limitations of word embeddings

- There are many different facets of ‘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 factorization
- Learning word embeddings via language models
- Contextualized word embeddings

Learning word embeddings via matrix factorization

Word embeddings via singular value decomposition

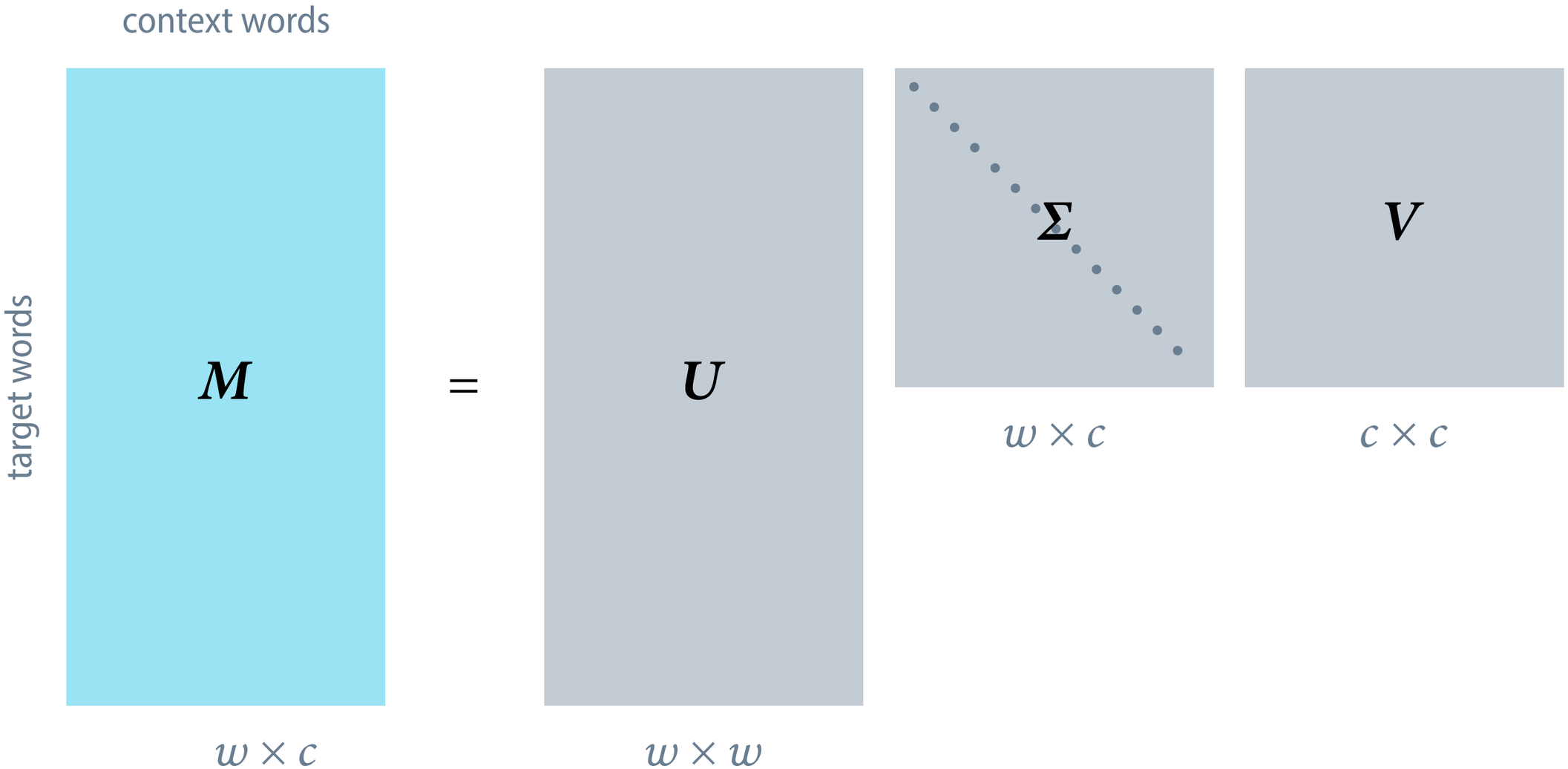
- The rows of co-occurrence matrices are long and sparse. Instead, we would like to have word vectors that are short and dense.

  length on the order of 10^2 instead of 10^4

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

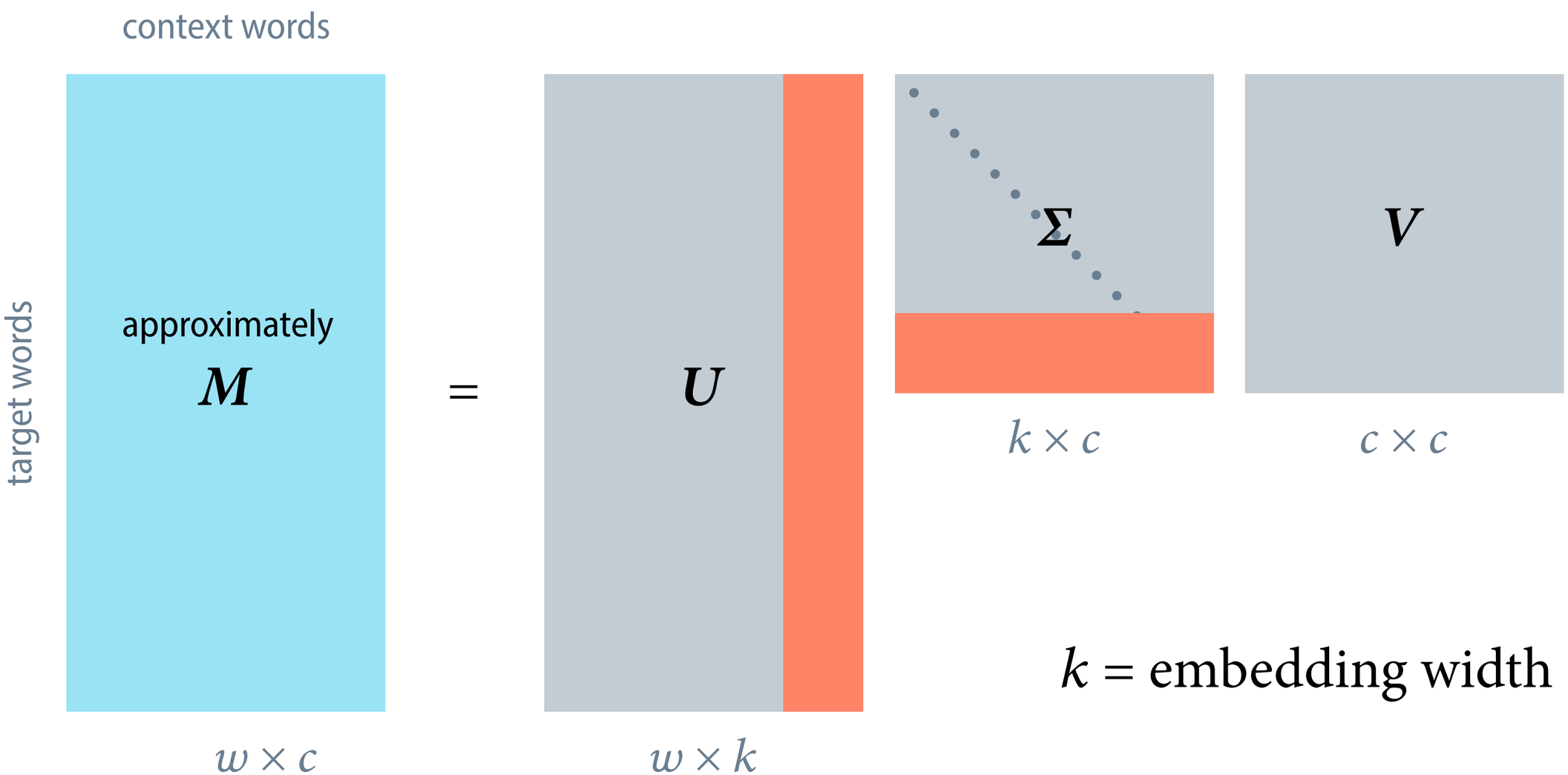
- This problem can be solved by computing the **singular value decomposition** of the co-occurrence matrix.

Singular value decomposition



Landauer and Dumais (1997)

Truncated singular value decomposition



k = embedding width

Truncated singular value decomposition

width 200

width 100

width 50

width 20

width 10

width 5

Photo credit: UMassAmherst

Word embeddings via singular value decomposition

- Each row of the (truncated) matrix $\boldsymbol{U}$ is a k-dimensional vector that represents the 'most important' information about a word.

  columns orded in decreasing order of importance

- A practical problem is that computing the singular value decomposition for large matrices is expensive.

  We are dealing with matrices with 10^8 entries.

Pointwise mutual information

- Raw counts favour pairs that involve very common contexts.

  the cat, *a cat* will receive higher weight than *cute cat*, *small cat*

- We want a measure that favours contexts in which the target word occurs more often than other words.

- A suitable measure is **pointwise mutual information (PMI)**:

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

Example from Goldberg (2017)

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 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)

Side track: PPMI can be used to find collocations

English collocations
close a deal
say a prayer
foot the bill
catch fire
sit a test
walk the dog
stir up emotions

Lexicographer's Mutual Information (LMI):

$$\text{LMI} = \#(x, y) \cdot \log \frac{P(x, y)}{P(x)P(y)}$$

Source: Snoder (2019), Appendix A

This lecture

- Introduction to word embeddings
- Learning word embeddings via matrix factorization
- Learning word embeddings via language models
- Contextualized word embeddings

Learning word embeddings via language models

Language models

- A **probabilistic language model** is a probability distribution over sequences of words in some language.

  Language models can also be defined at the character level.

- Recent years have seen the rise of **neural language models**, which are based on distributed representations of words.

Language models

- By the chain rule, the probability of a sequence of N words can be computed using conditional probabilities as

$$P(w_1 \cdots w_N) = \prod_{k=1}^{N} P(w_k \mid w_1 \cdots w_{k-1})$$

- To make probability estimates more robust, we can approximate the full history $w_1 \cdots w_N$ by the last few words:

$$P(w_1 \cdots w_N) = \prod_{k=1}^{N} P(w_k \mid w_{k-n+1} \cdots w_{k-1})$$

N-gram models

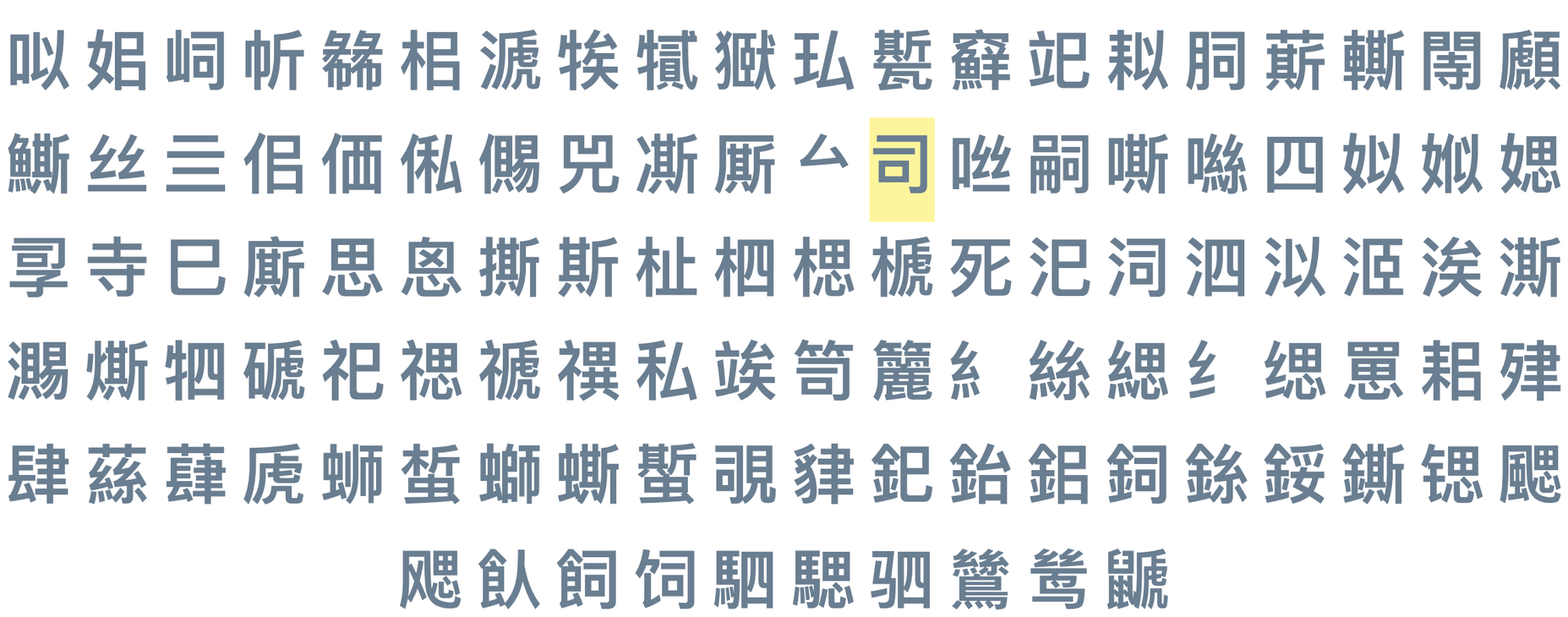
- An ***n*-gram** is a contiguous sequence of n words or characters.

  unigram (*Text*), bigram (*Text Mining*), trigram (*Text Mining course*)

- An ***n*-gram model** is a language model defined on n-grams – a probability distribution over sequences of n words.

Language models for predictive text input

ping guo gong si → 蘋果公司



Formal definition of an n-gram model

n	the model's order (1 = unigram, 2 = bigram, …)
V	a set of possible words (character); the vocabulary
$P(w\|u)$	a probability that specifies how likely it is to observe the word w after the context $(n-1)$-gram u

one value for each combination of a word w and a context u

Unigram model

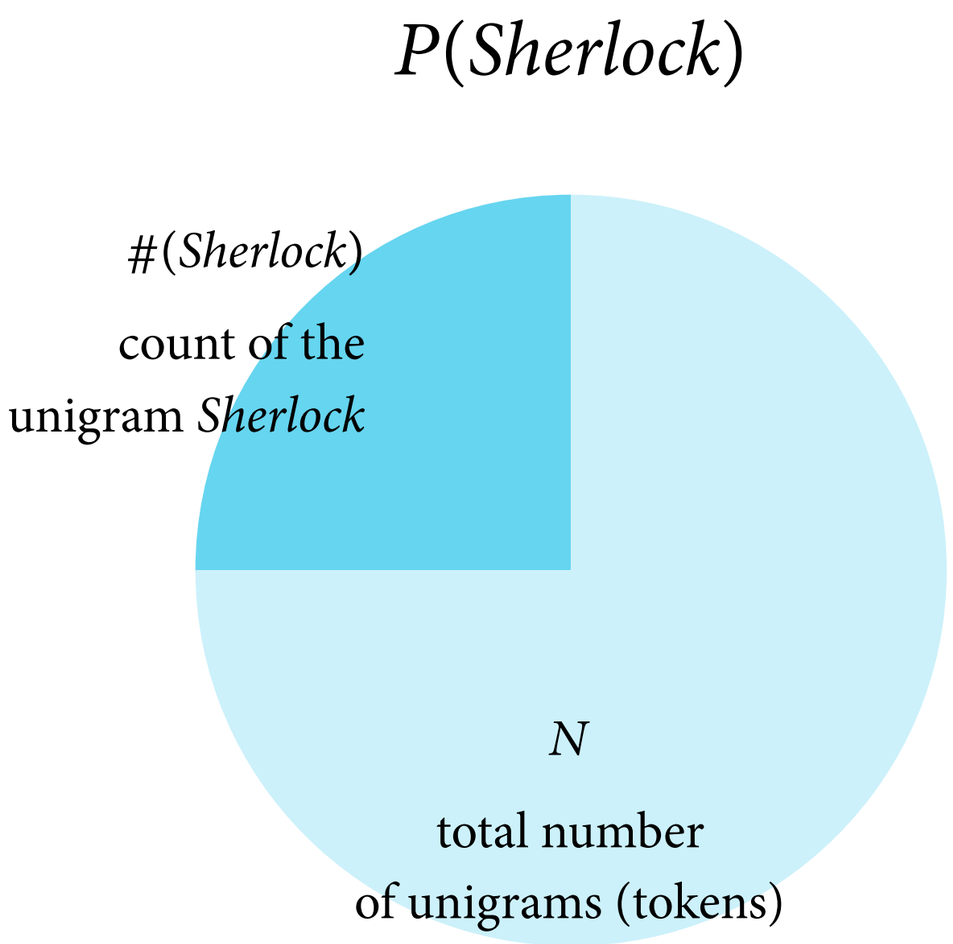
A **unigram language model** views a text as a 'bag of words':

$$P(w_1 \cdots w_N) = \prod_{k=1}^{N} P(w_k) = \prod_{w \in V} P(w)^{\#(w)}$$

text — count of w — vocabulary

Thus contexts are empty.

MLE of unigram probabilities



$$P(w) = \frac{\#(w)}{N}$$

Bigram models

A **bigram model** is a Markov model on sequences of words:

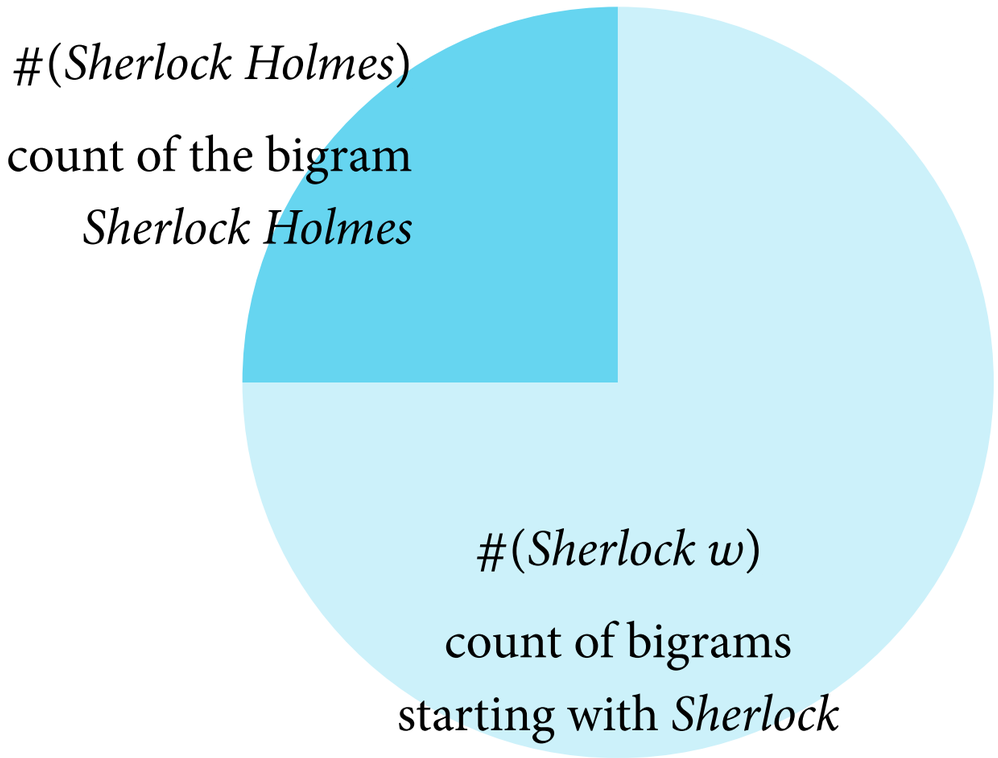
$$P(w_1 \cdots w_N) = P(w_1 \mid \text{BOS}) \cdot \prod_{i=2}^{N} \left[P(w_i \mid w_{i-1}) \right] \cdot P(\text{EOS} \mid w_N)$$

- $w_1 \cdots w_N$: text
- BOS: beginning-of-sequence marker
- EOS: end-of-sequence marker
- $P(w_i \mid w_{i-1})$: probability of a word given the previous word

Thus contexts are unigrams.

Estimating bigram probabilities

$$P(Holmes|Sherlock)$$



$$P(w \mid u) = \frac{\#(uw)}{\#(u\bullet)}$$

$$P(w \mid u) = \frac{\#(uw)}{\#(u)}$$

The case for smoothing

- Shakespeare's collected works contain ca. 31,000 word types. There are 961 million different bigrams with these words.
- In his texts we only find 300,000 bigrams. This means that 99.97% of all theoretically possible bigrams have count 0.
- Under an unsmoothed bigram model, each sentence containing one of those bigrams will receive a probability of zero.

  Zero probabilities destroy information!

Recursive smoothing

To smooth an n-gram model, we smooth it with the $(n-1)$-gram model, or with the uniform distribution in the case where $n = 1$.

$$P(w \mid u) = \lambda(u)\frac{\#(uw)}{\#(u\bullet)} + \big(1 - \lambda(u)\big)P(w \mid \bar{u})$$ Witten–Bell

$$P(w \mid u) = \frac{\max\big(0, \#(uw) - d\big)}{\#(u\bullet)} + \frac{dV_{1+}(u\bullet)}{\#(u\bullet)}P(w \mid \bar{u})$$ Absolute discounting

Here $\bar{u}$ denotes the $(n-2)$-gram suffix of u.

The problem of unknown words

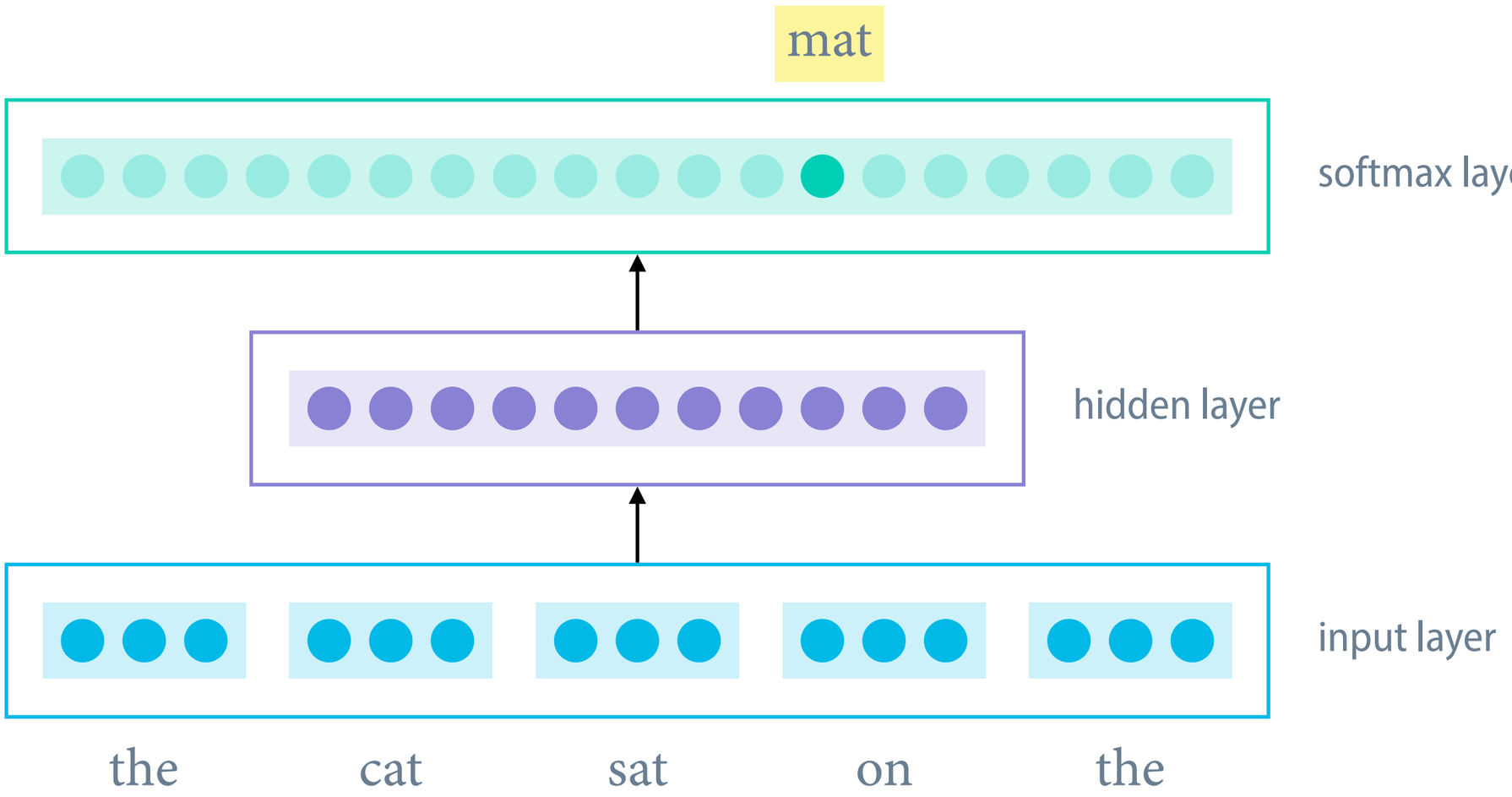
- In addition to unseen words, a new text may even contain **unknown words**. For these, smoothing will not help.

  out-of-vocabulary

- A simple way to deal with this is to introduce a special word type UNK, and to smooth it like any other word type in the vocabulary.

- When we compute the probability of a document, then we first replace every unknown word with UNK.

Neural networks as language models



Advantages of neural language models

- Neural models can achieve better perplexity than probabilistic models, and scale to much larger values of n.
- Words in different positions share parameters, making them share statistical strength.

  Everything must pass through the hidden layer.

- The network can learn that in some contexts, only parts of the n-gram are informative.

  implicit smoothing, helps with unknown words

Goldberg (2017)

Word embeddings via neural language models

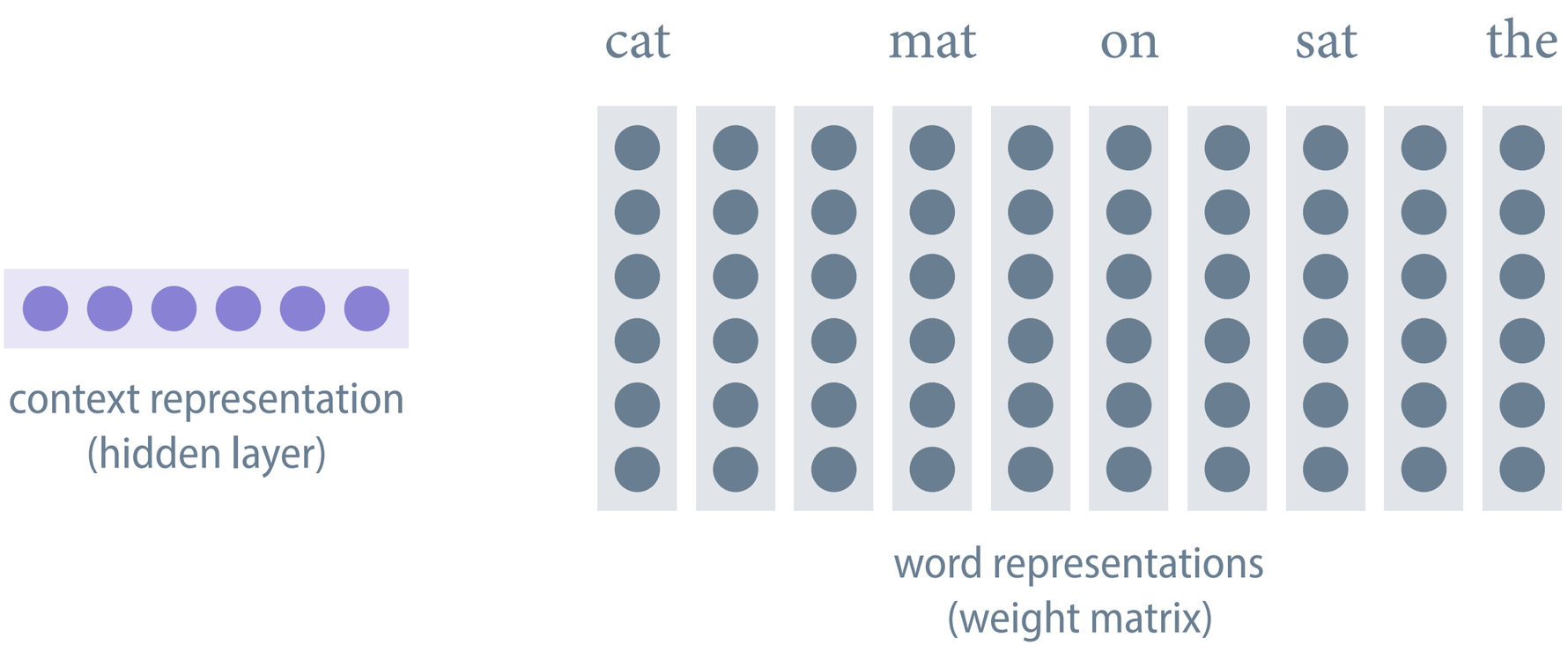
- The neural language model is trained to predict the probability of the next word being w, given the preceding words:

$$\hat{\boldsymbol{y}} = P(w \mid \text{preceding words}) = \text{softmax}(\boldsymbol{hW})$$

- Each column of the matrix $\boldsymbol{W}$ is a $\dim(\boldsymbol{h})$-dimensional vector that is associated with some vocabulary item w.

- We can view this vector as a representation of w that captures its compatibility with the context represented by the vector $\boldsymbol{h}$.

Network weights = word embeddings



Intuitively, words that occur in similar contexts
will be forced to have similar context representations.

Training word embeddings using a language model

- Initialise the word vectors with random values.

  typically by sampling from a uniform or normal distribution

- Train the network on large volumes of text.

  word2vec: 100 billion words

- Word vectors will be optimised to the prediction task.

  Words that tend to precede the same words will get similar vectors.

Google's word2vec

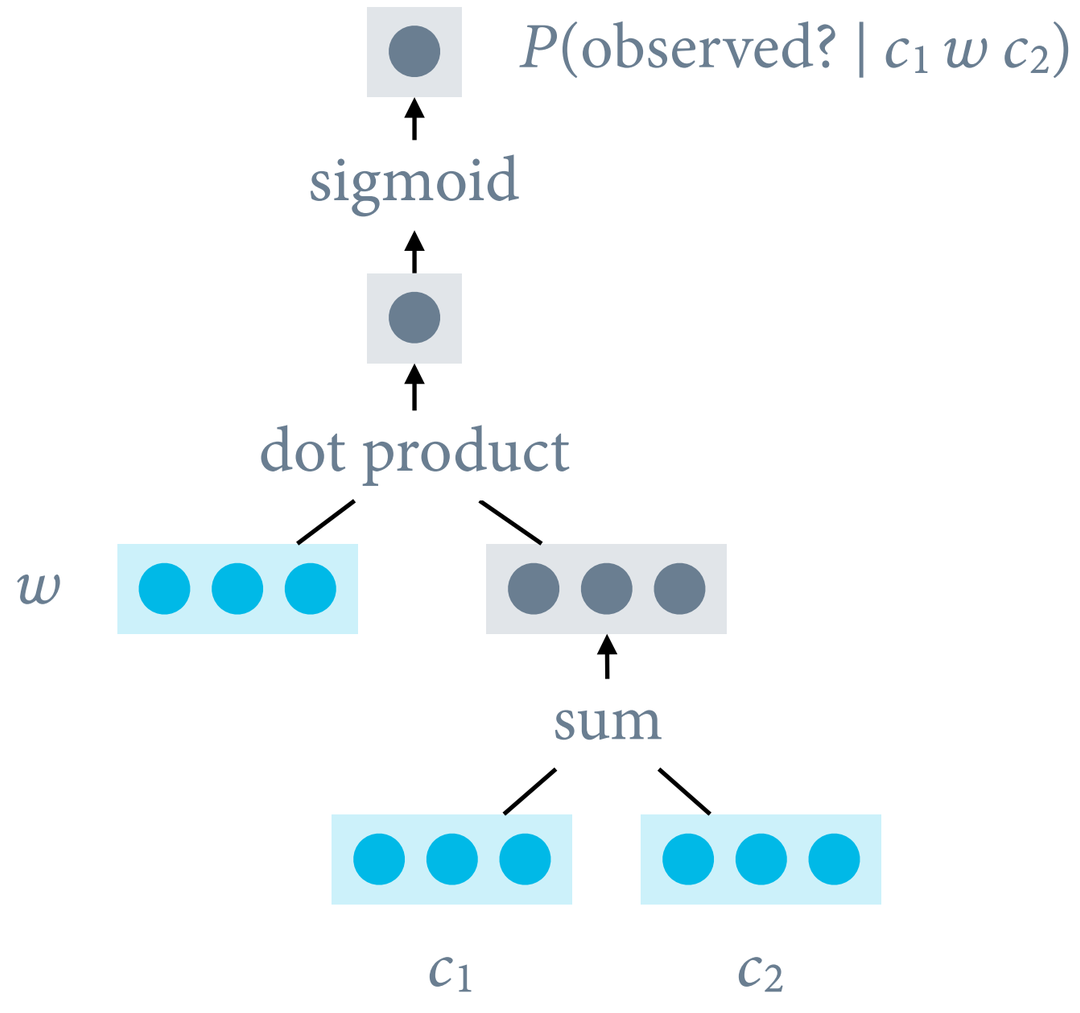
- Google's word2vec implements two different training algorithms for word embeddings: **continuous bag-of-words** and **skip-gram**.

- Both algorithms learn word embeddings as side products of a *binary* prediction task: 'Is this an actual word–context pair?'

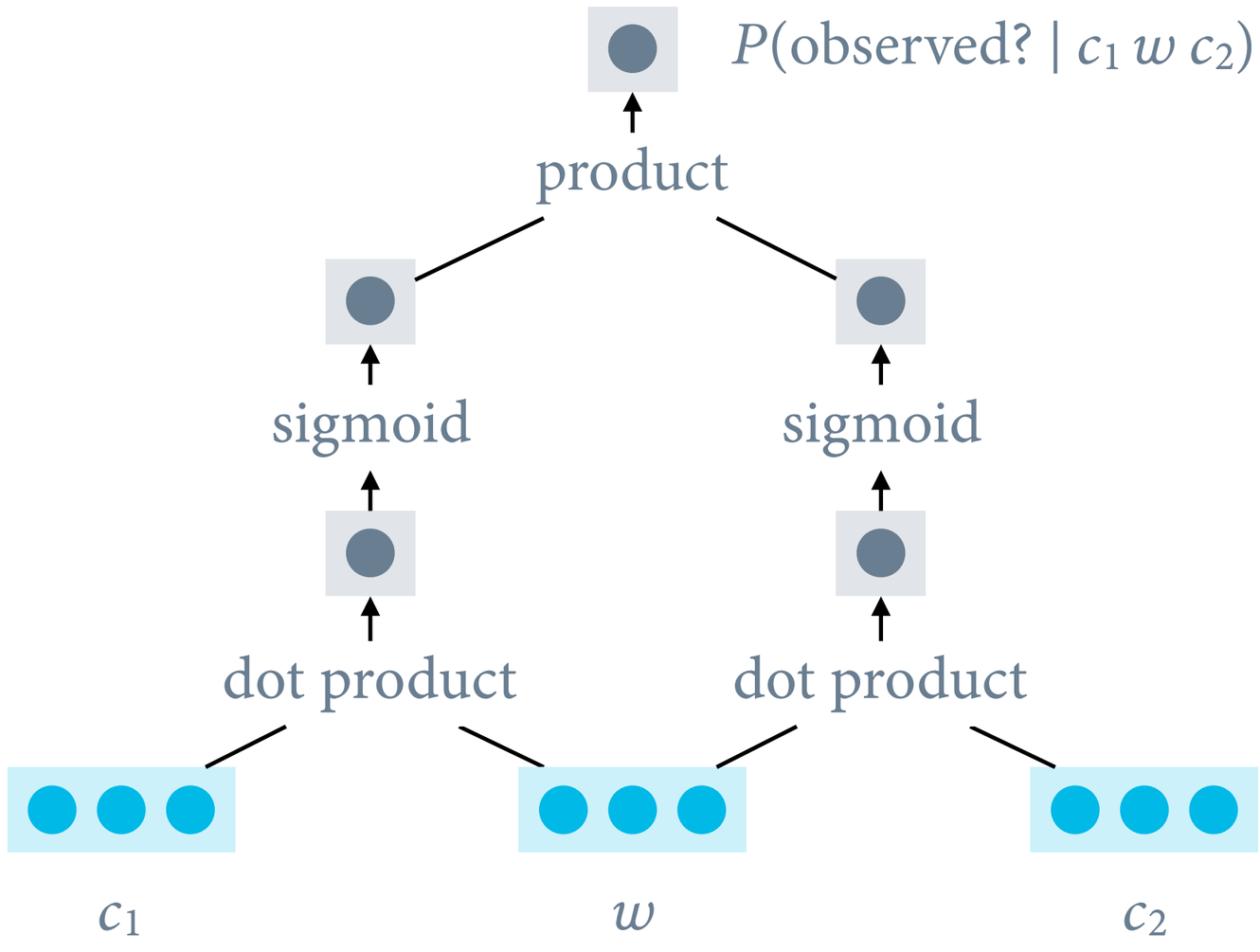
- Positive examples are generated from a corpus. Negative examples are generated by taking k copies of a positive example and randomly replacing the target word with some other word.

negative sampling

The continuous bag-of-words model



The skip-gram model



Connecting the two worlds

- The two algorithmic approaches that we have seen take two seemingly very different perspectives: 'count-based' and 'neural'.
- However, a careful analysis reveals that the skip-gram model is implicitly computing the already-decomposed PPMI matrix.

  Levy and Goldberg (2014)

This lecture

- Introduction to word embeddings
- Learning word embeddings via matrix factorization
- Learning word embeddings via language models
- Contextualized word embeddings

Contextualised word embeddings

Contextualised embeddings

- In standard word embedding models, 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.
- Using ELMo embeddings in supervised tasks involves learning weighting parameters for the linear combination.
- Additionally, it is often beneficial to fine-tune (continue training) a pre-trained ELMo model on task-specific data.

  domain transfer

Muppet character image from The Muppet Wiki

Properties of the ELMo architecture

- The parameters of the model are shared across all time steps.

  The number of parameters does not grow with the sequence length.

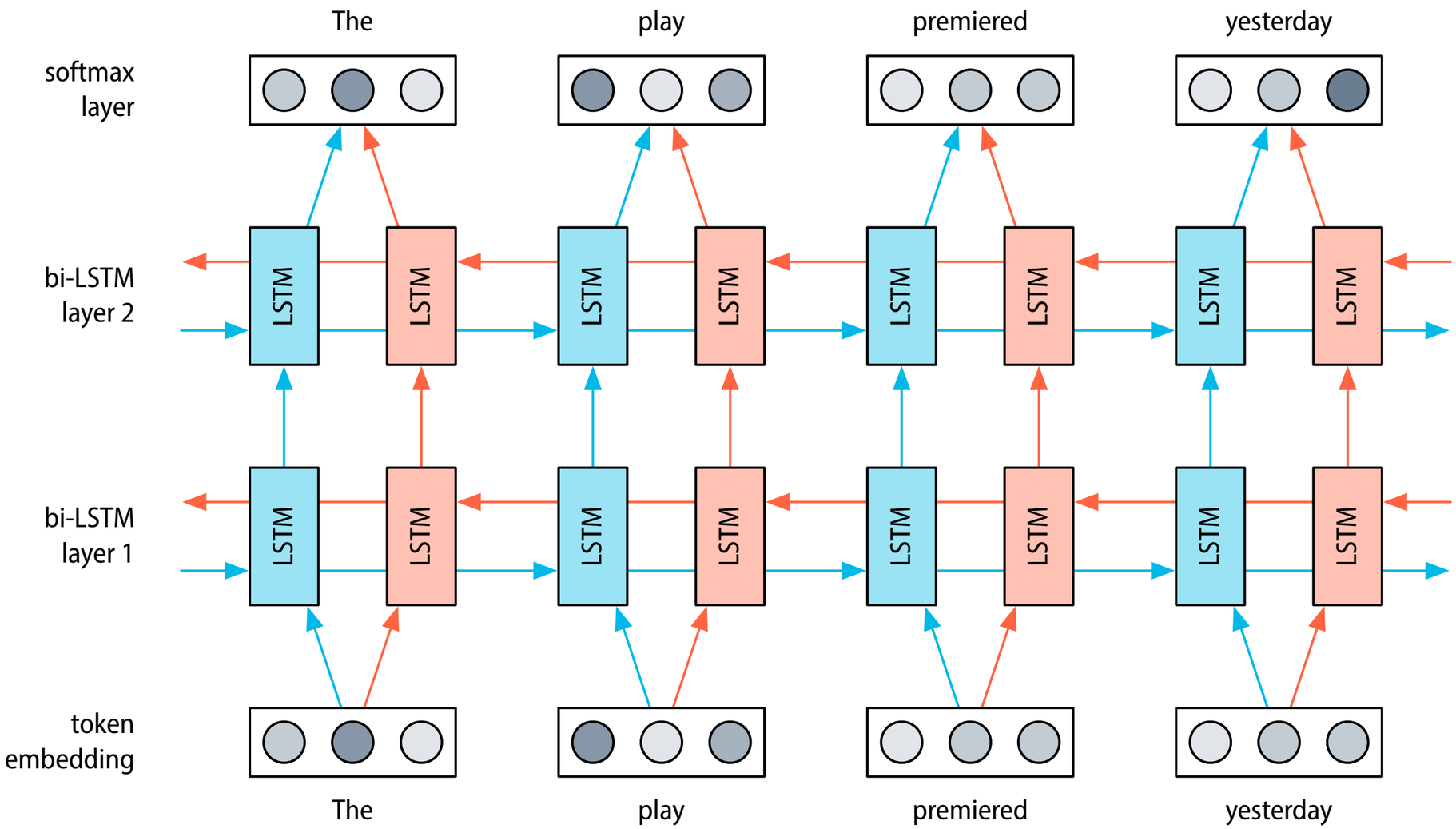
- The hidden state can be influenced by the entire input.

  Contrast this with the limited horizon assumption of n-gram models.

- The model cannot only 'look back', but also 'look forward'.

  central component: bidirectional LSTM

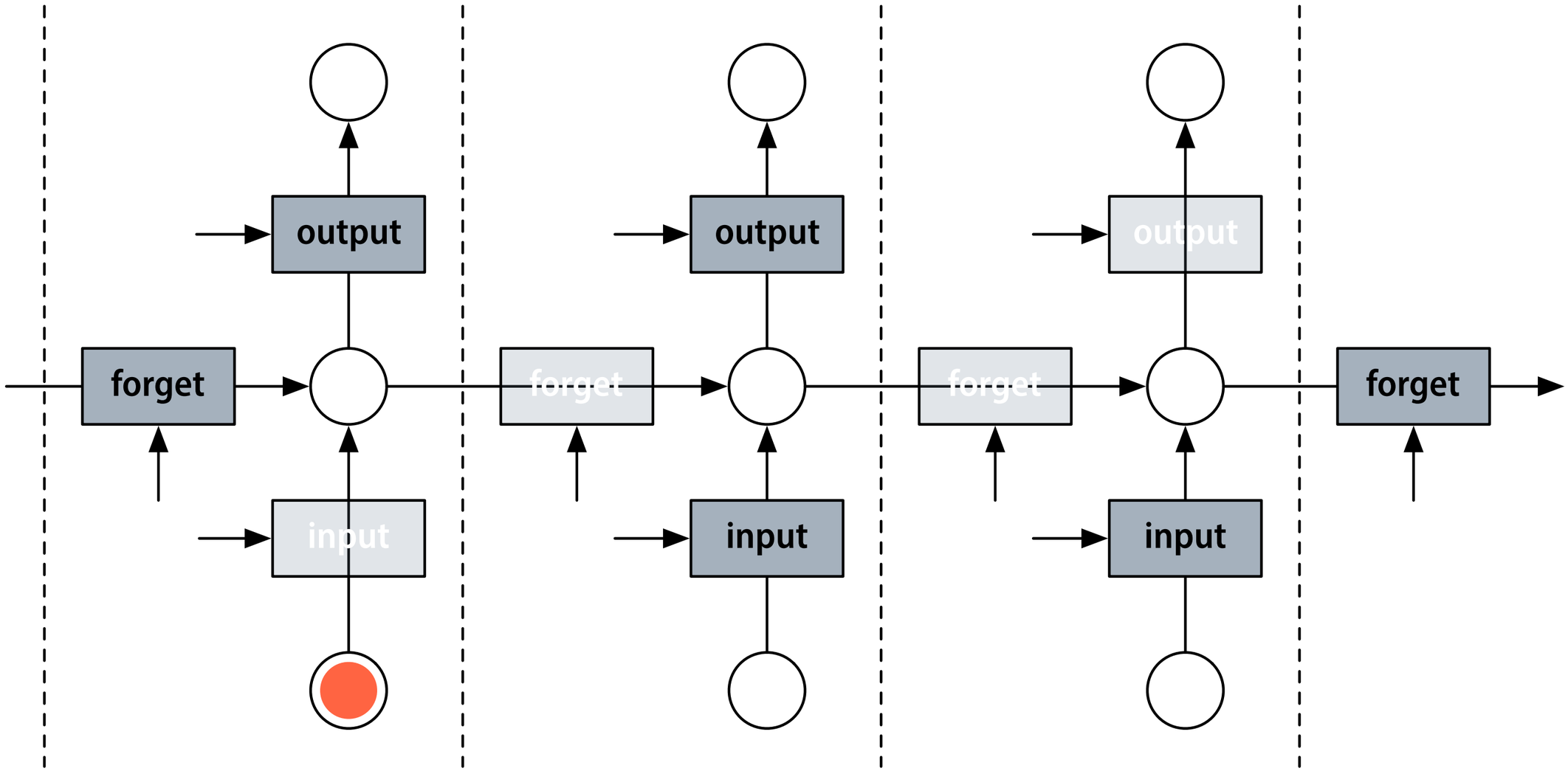
ELMo – Embeddings from Language Models



Long Short-Term Memory (LSTM)

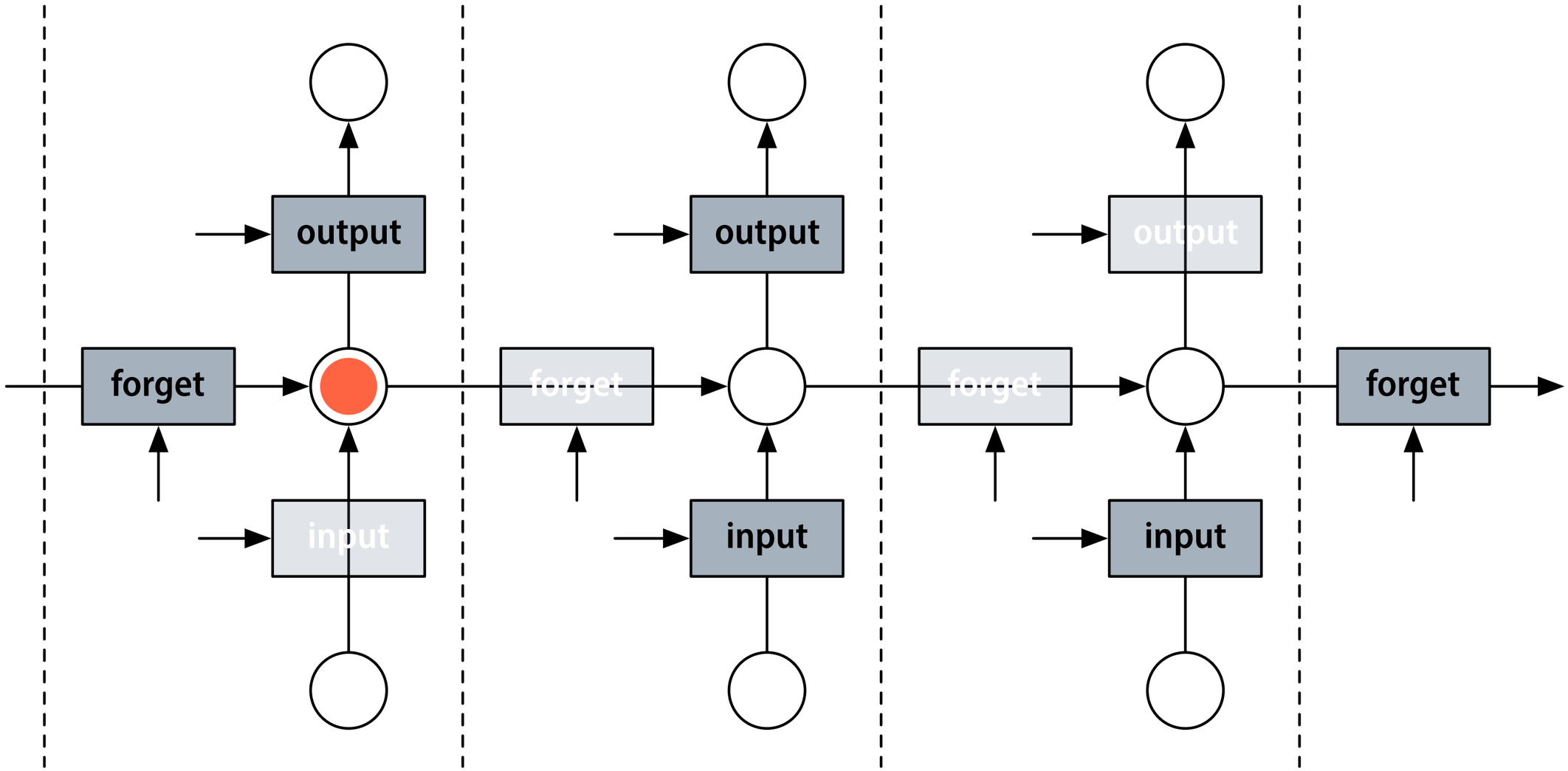
- The Long Short-Term Memory (LSTM) architecture was specifically designed to battle the vanishing gradients problem.
- Metaphor: The dynamic state of the neural network can be considered as a short-term memory.
- The LSTM architecture tries to make this short-term memory last as long as possible by preventing vanishing gradients.
- Central idea: gating mechanism

Information flow in an LSTM



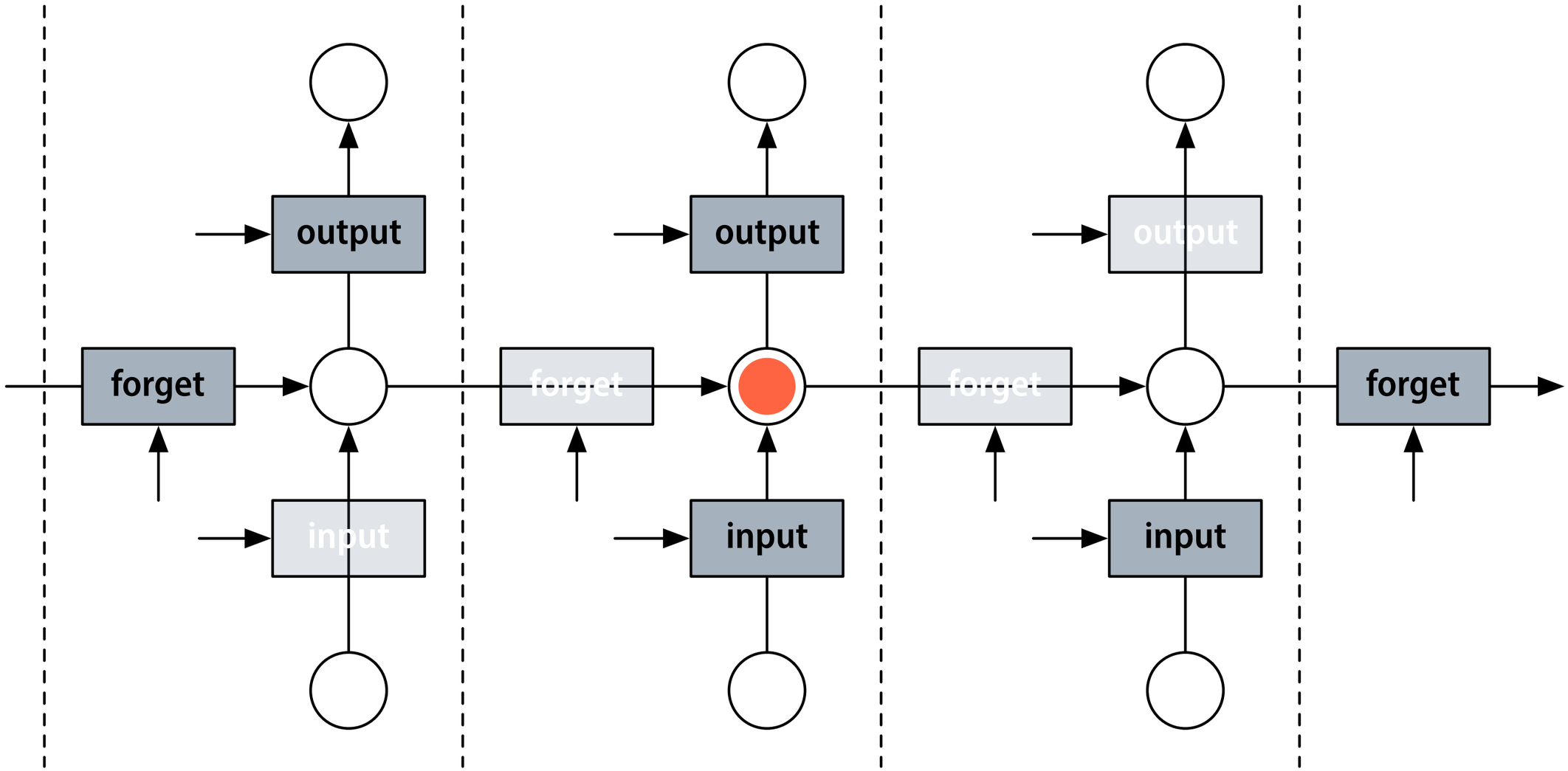
Based on an idea by Geoffrey Hinton

Information flow in an LSTM



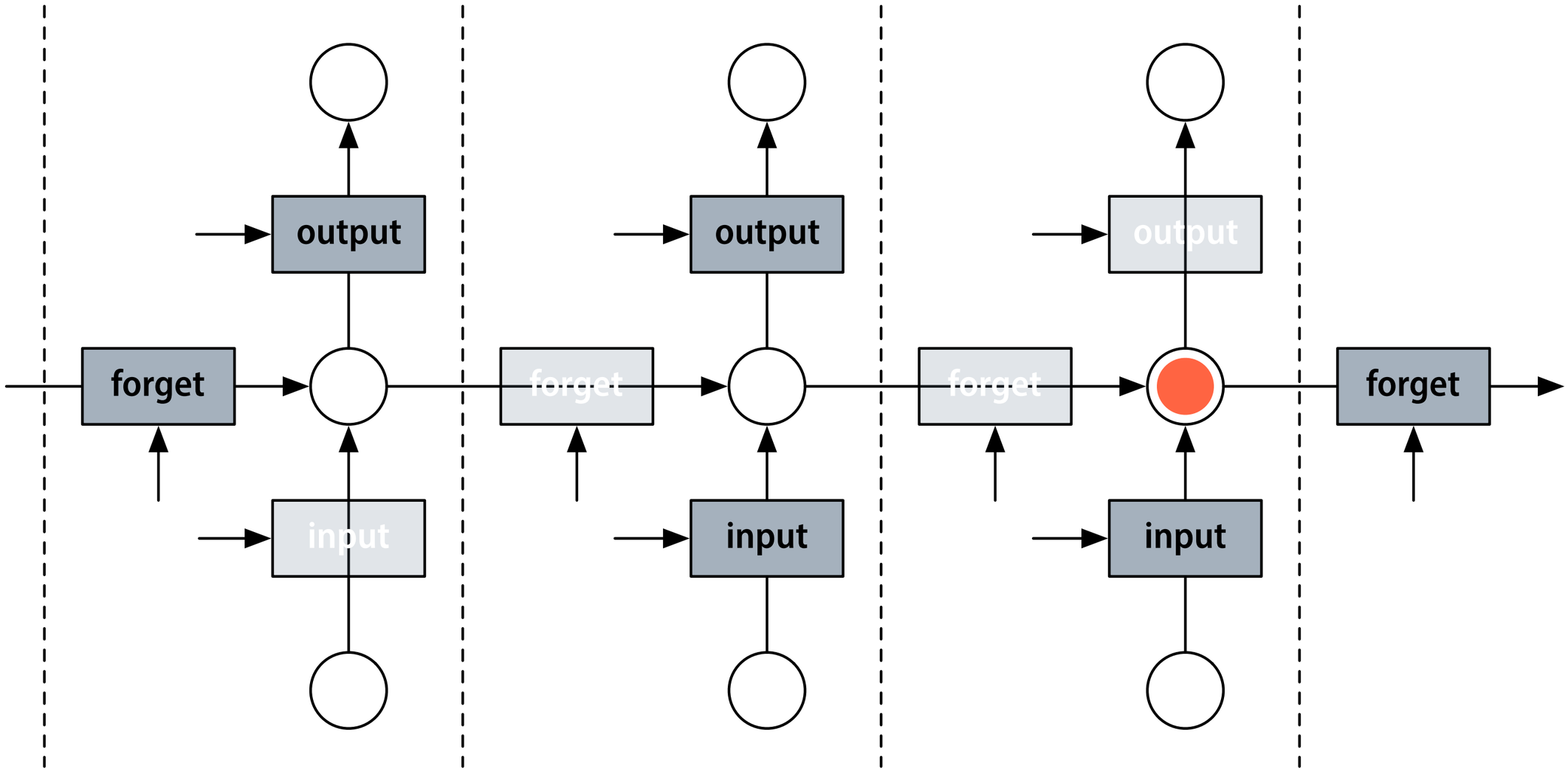
Based on an idea by Geoffrey Hinton

Information flow in an LSTM



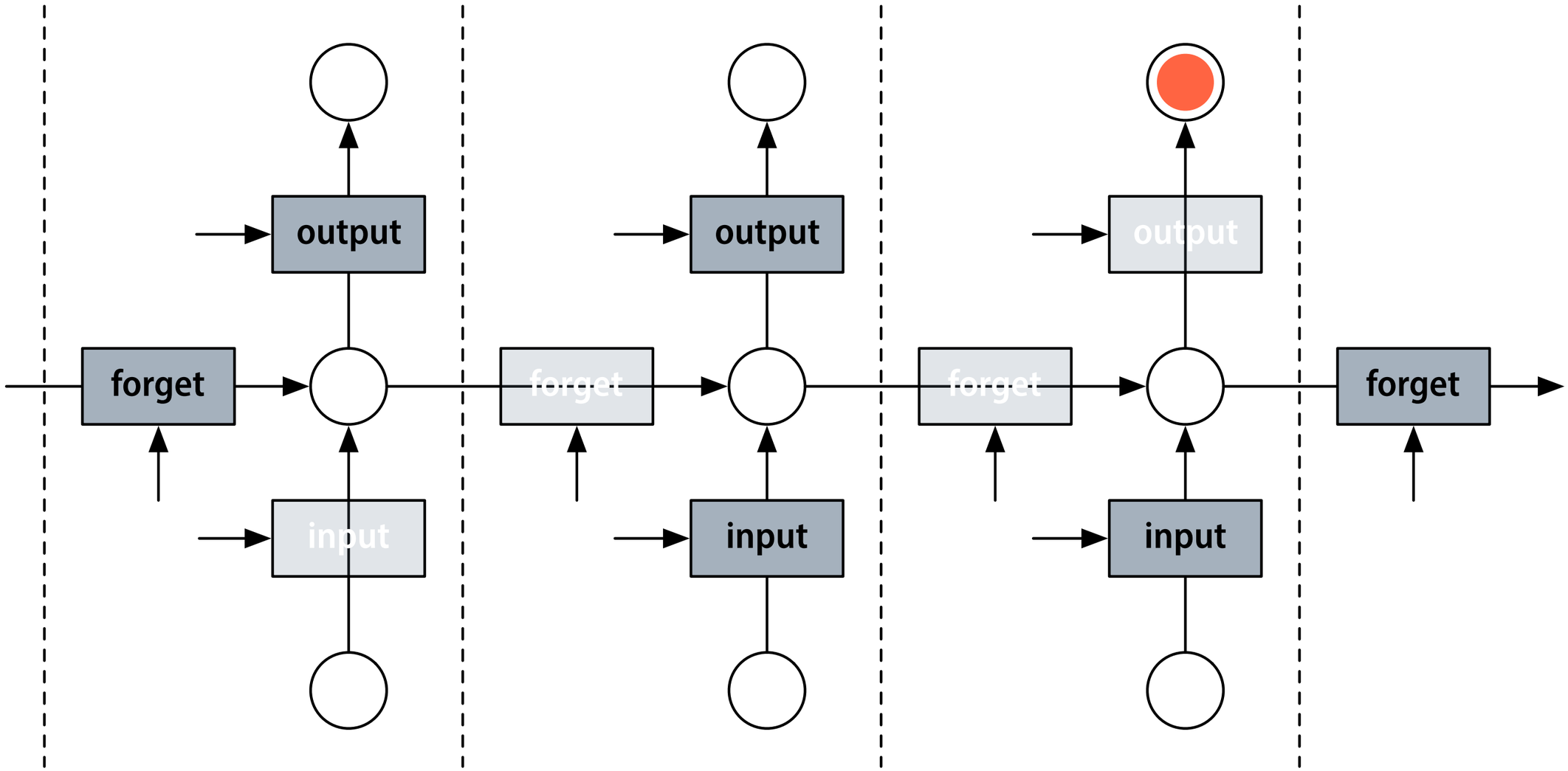
Based on an idea by Geoffrey Hinton

Information flow in an LSTM



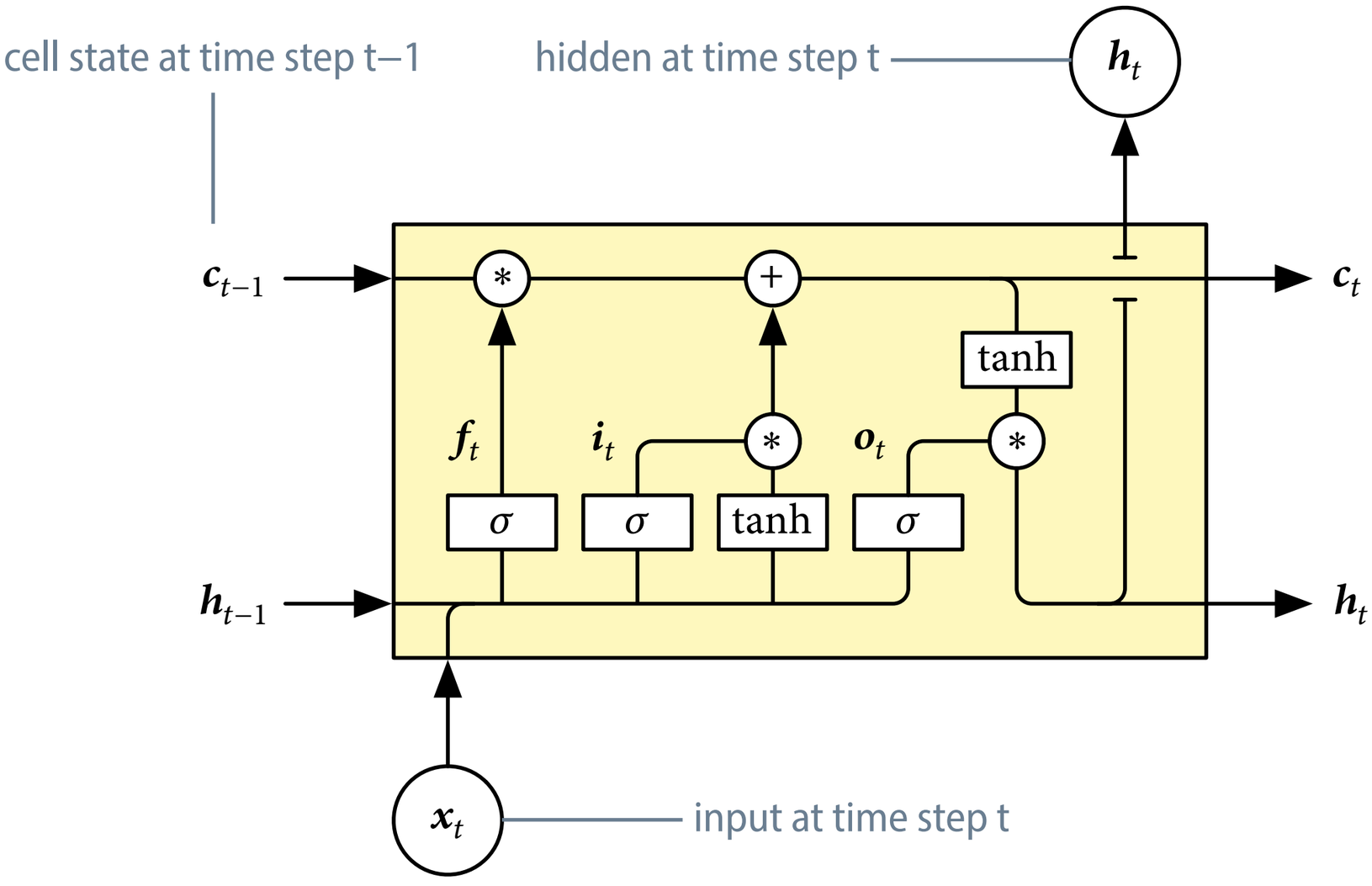
Based on an idea by Geoffrey Hinton

Information flow in an LSTM



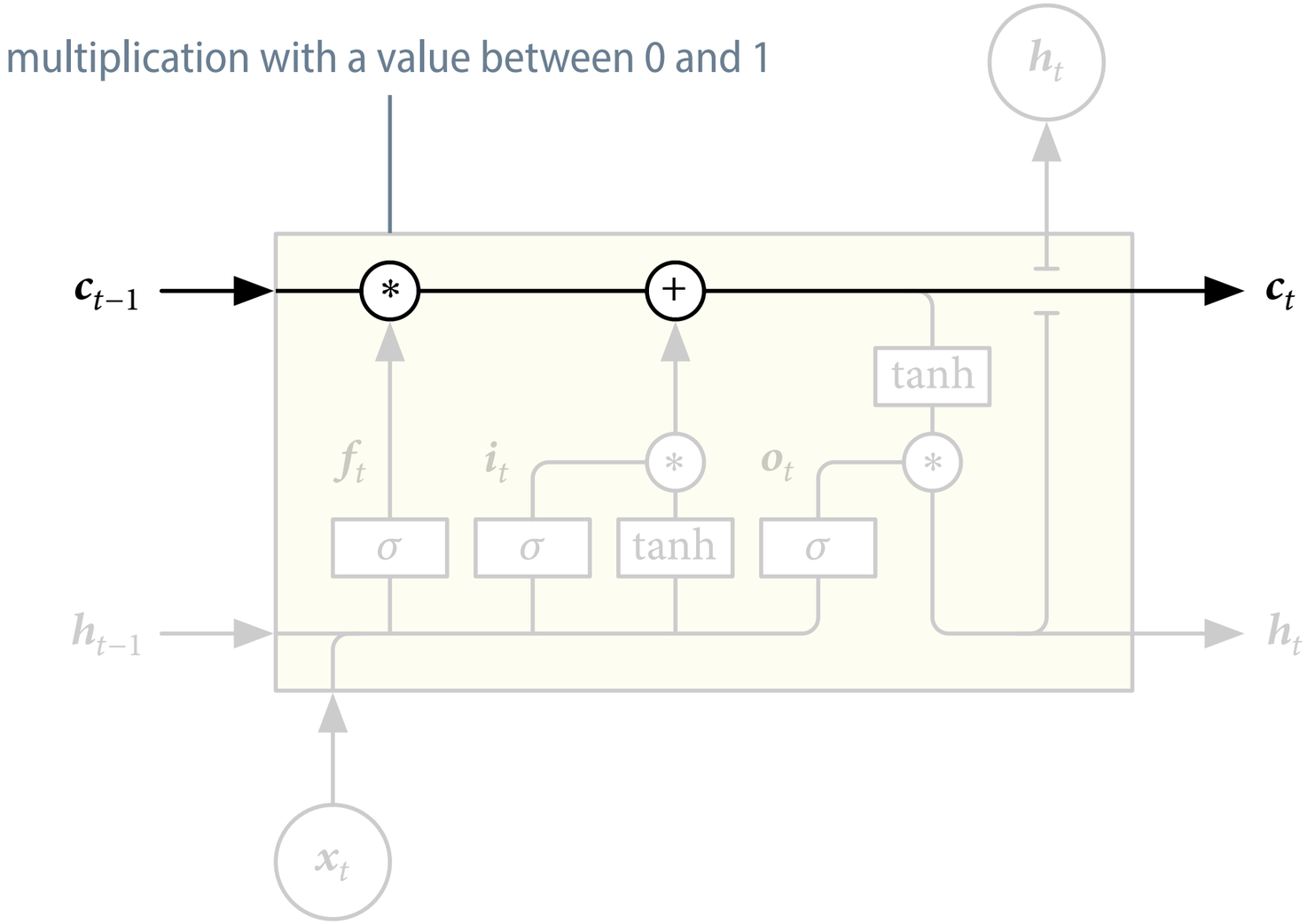
Based on an idea by Geoffrey Hinton

A look inside an LSTM cell

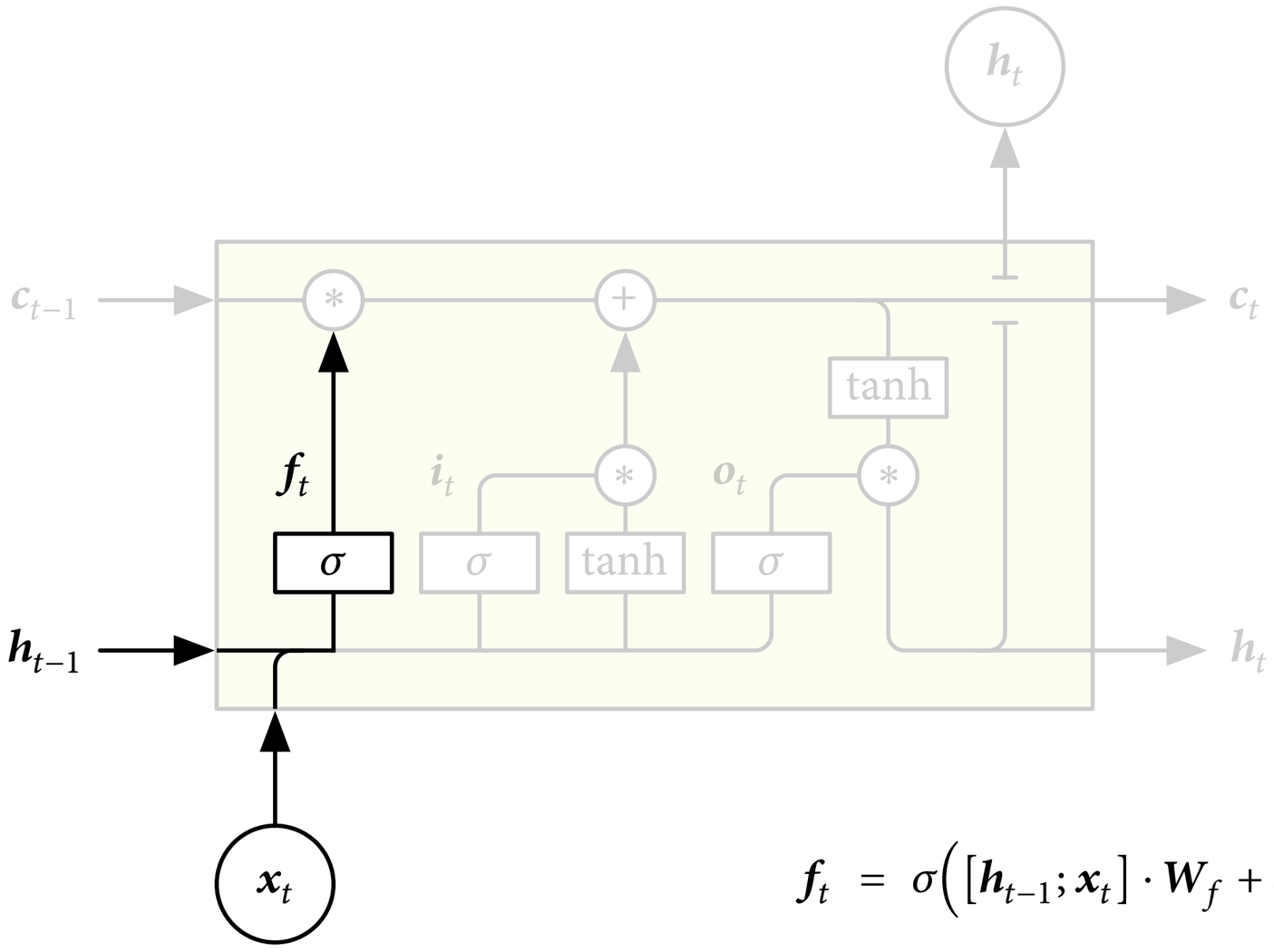


Adapted from Chris Olah (link)

The conveyor belt

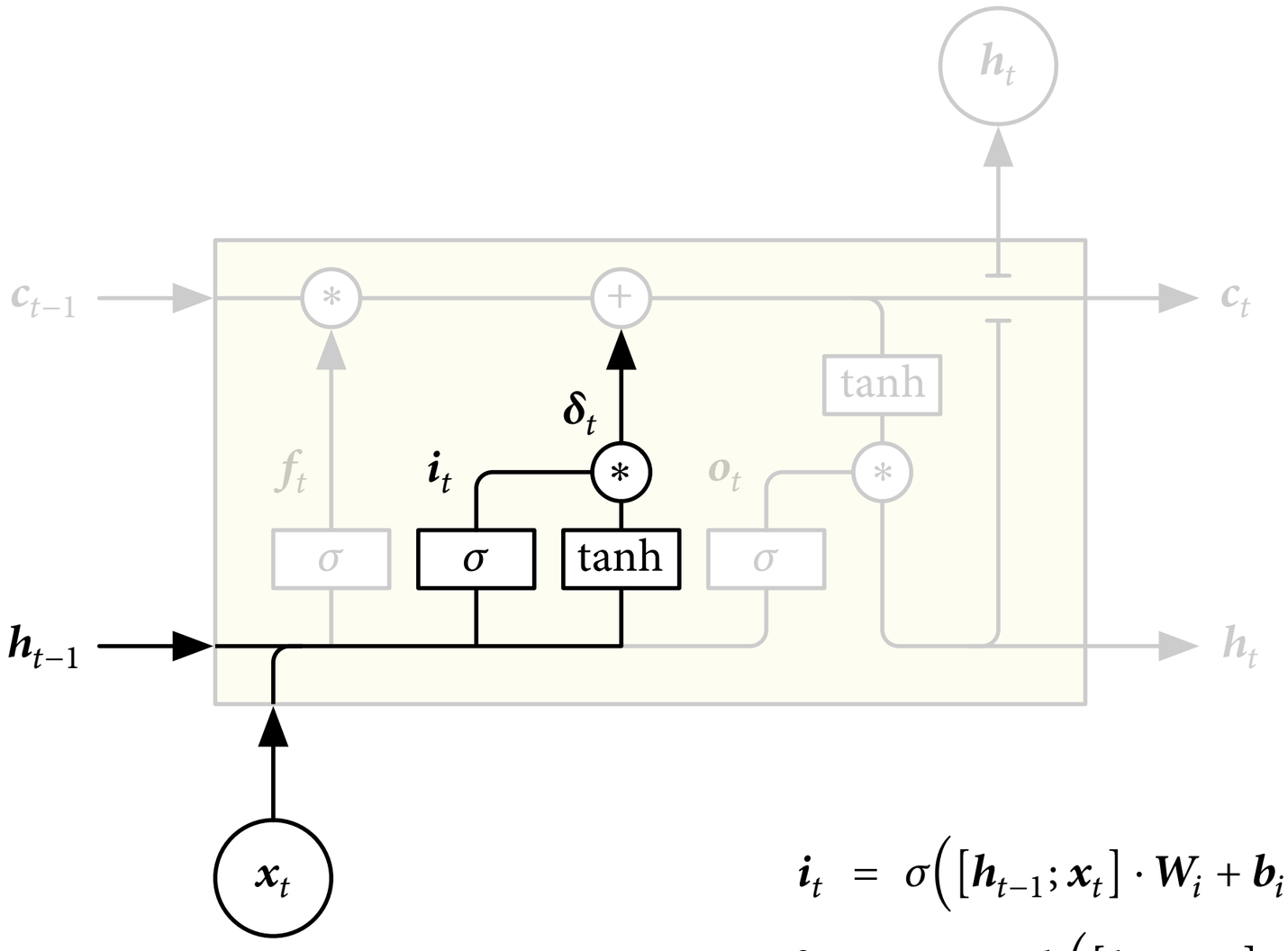


The ‘forget’ gate unit



$$f_t = \sigma\Big(\big[h_{t-1}; x_t\big] \cdot W_f + b_f\Big)$$

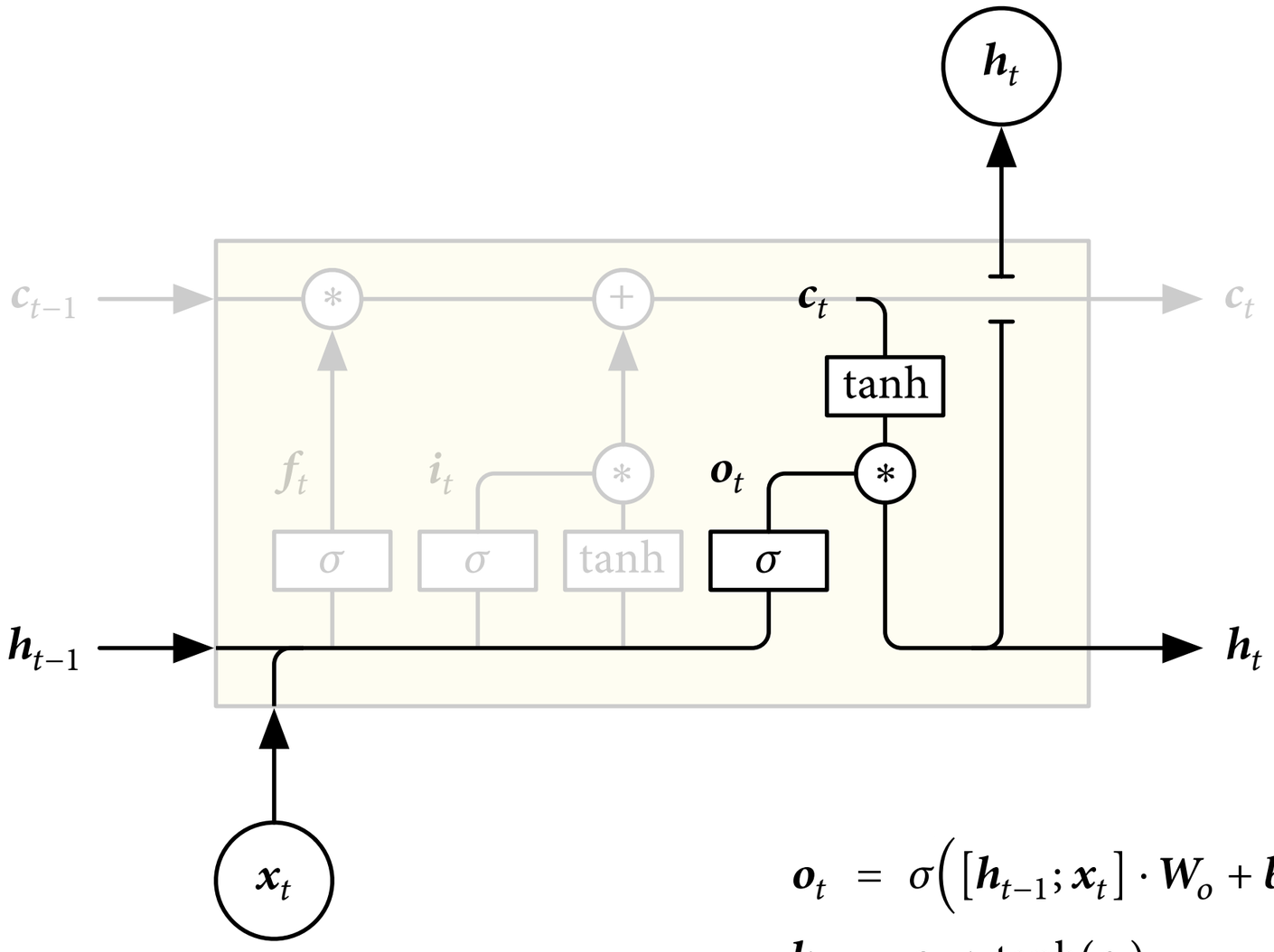
The ‘input’ gate unit



$$i_t = \sigma\Big(\big[h_{t-1}; x_t\big] \cdot W_i + b_i\Big)$$

$$\delta_t = i_t * \tanh\Big(\big[h_{t-1}; x_t\big] \cdot W_c + b_c\Big)$$

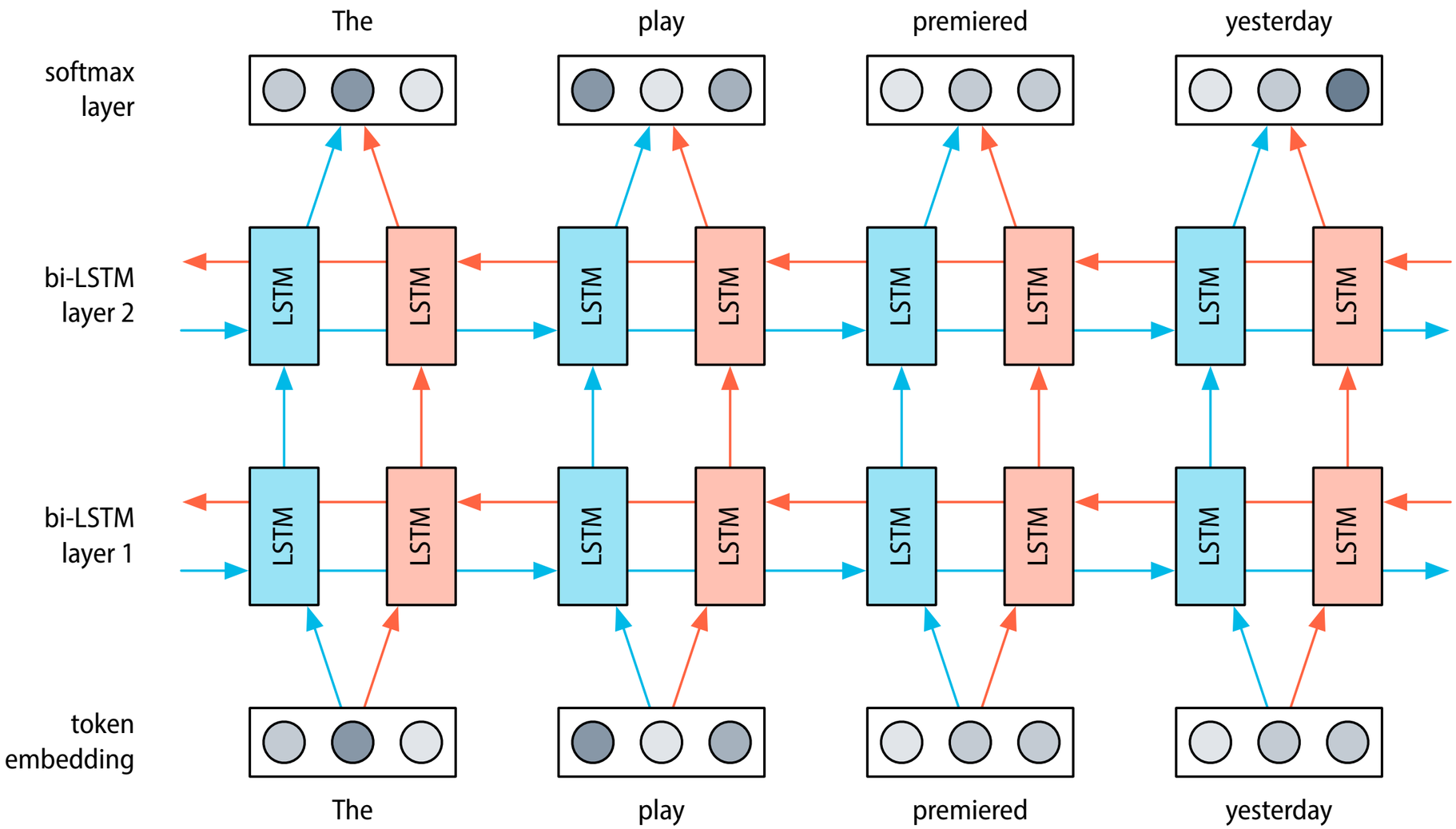
The 'output' gate unit



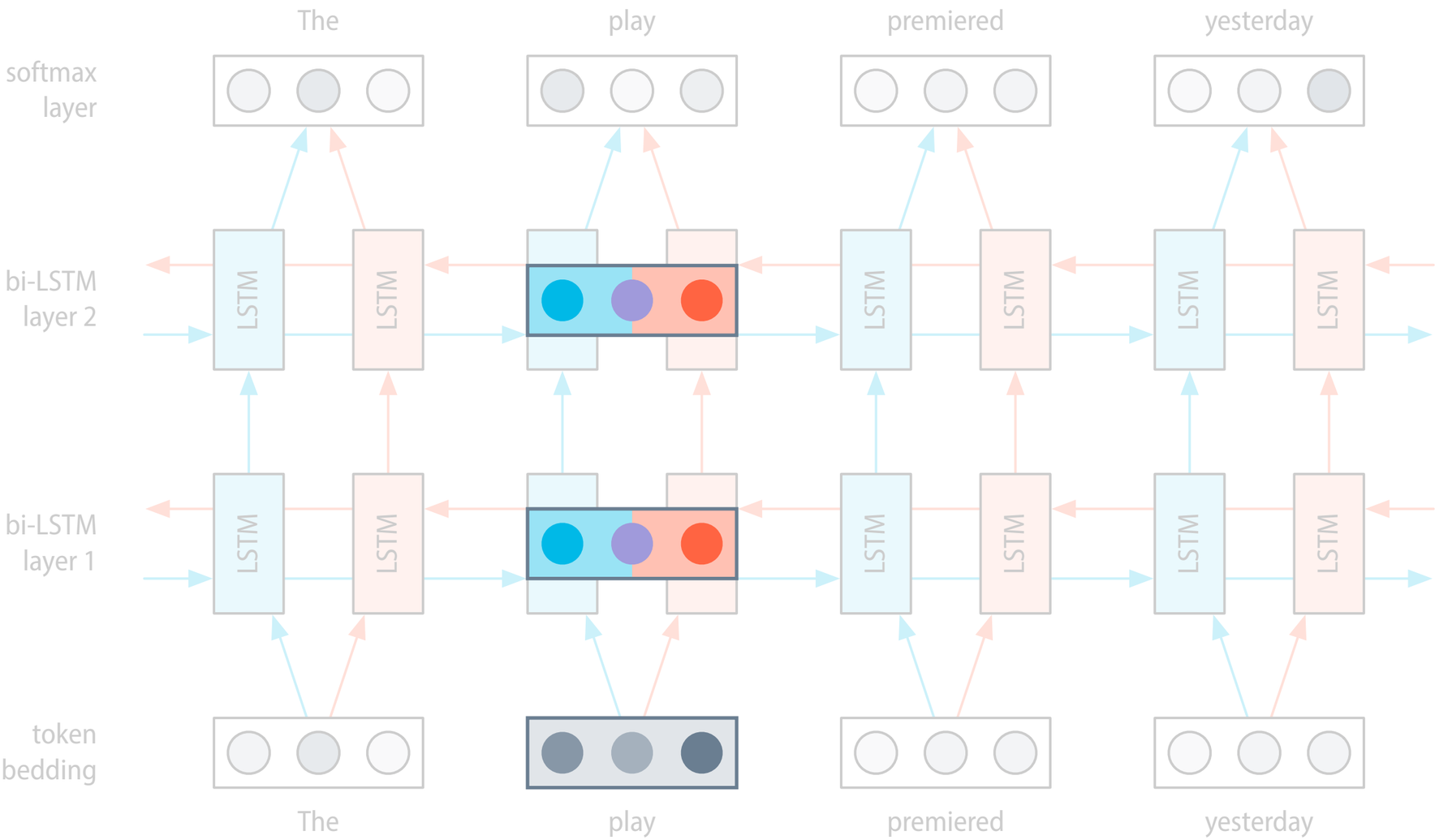
$$o_t = \sigma\Big(\big[h_{t-1}; x_t\big] \cdot W_o + b_o\Big)$$

$$h_t = o_t * \tanh(c_t)$$

ELMo – Embeddings from Language Models

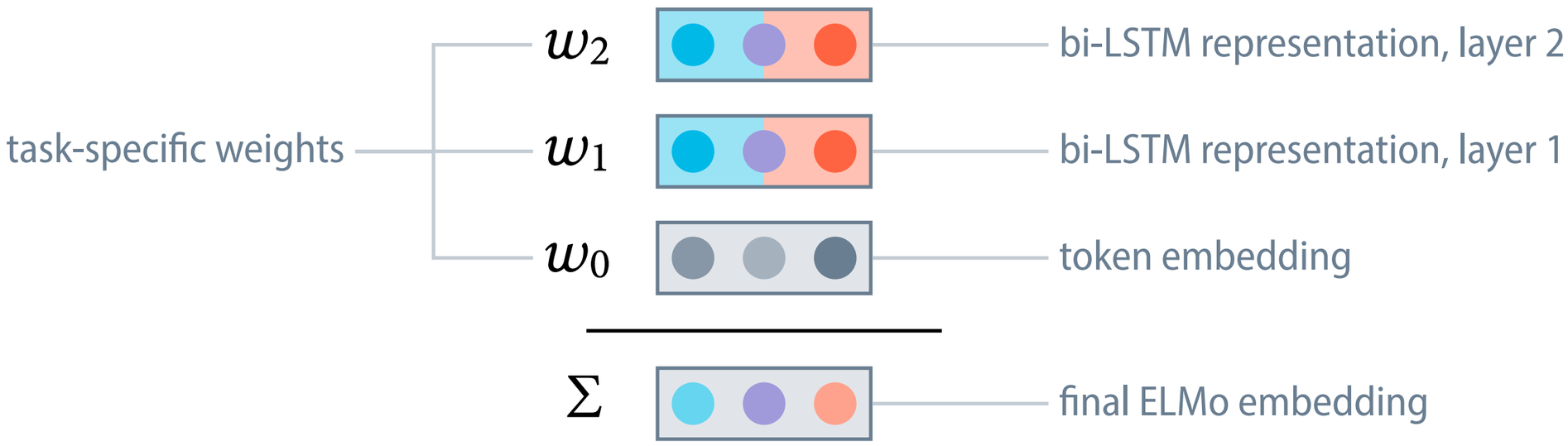


ELMo – Embeddings from Language Models



ELMo – Embeddings from Language Models

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



ELMo – Embeddings from Language Models

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

BERT – Bidirectional Encoder Representations from Transformers

- uses a masked language model pre-training objective
- makes central use of an attention mechanism
- significant improvements over existing models, including ELMo

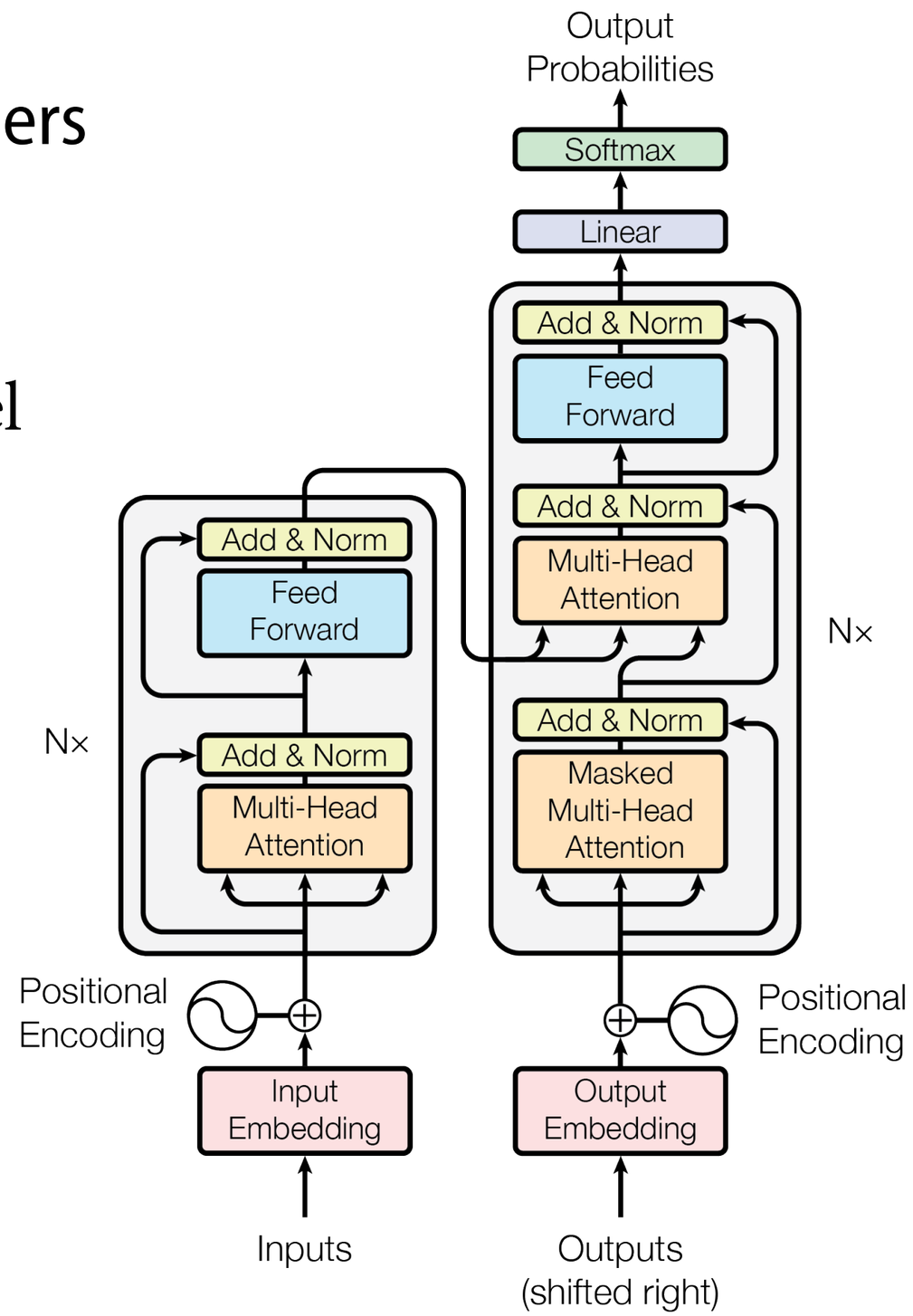


Figure from Vaswani et al. (2017)

Muppet character image from The Muppet Wiki

This lecture

- Introduction to word embeddings
- Learning word embeddings via matrix factorization
- Learning word embeddings via language models
- Contextualized word embeddings