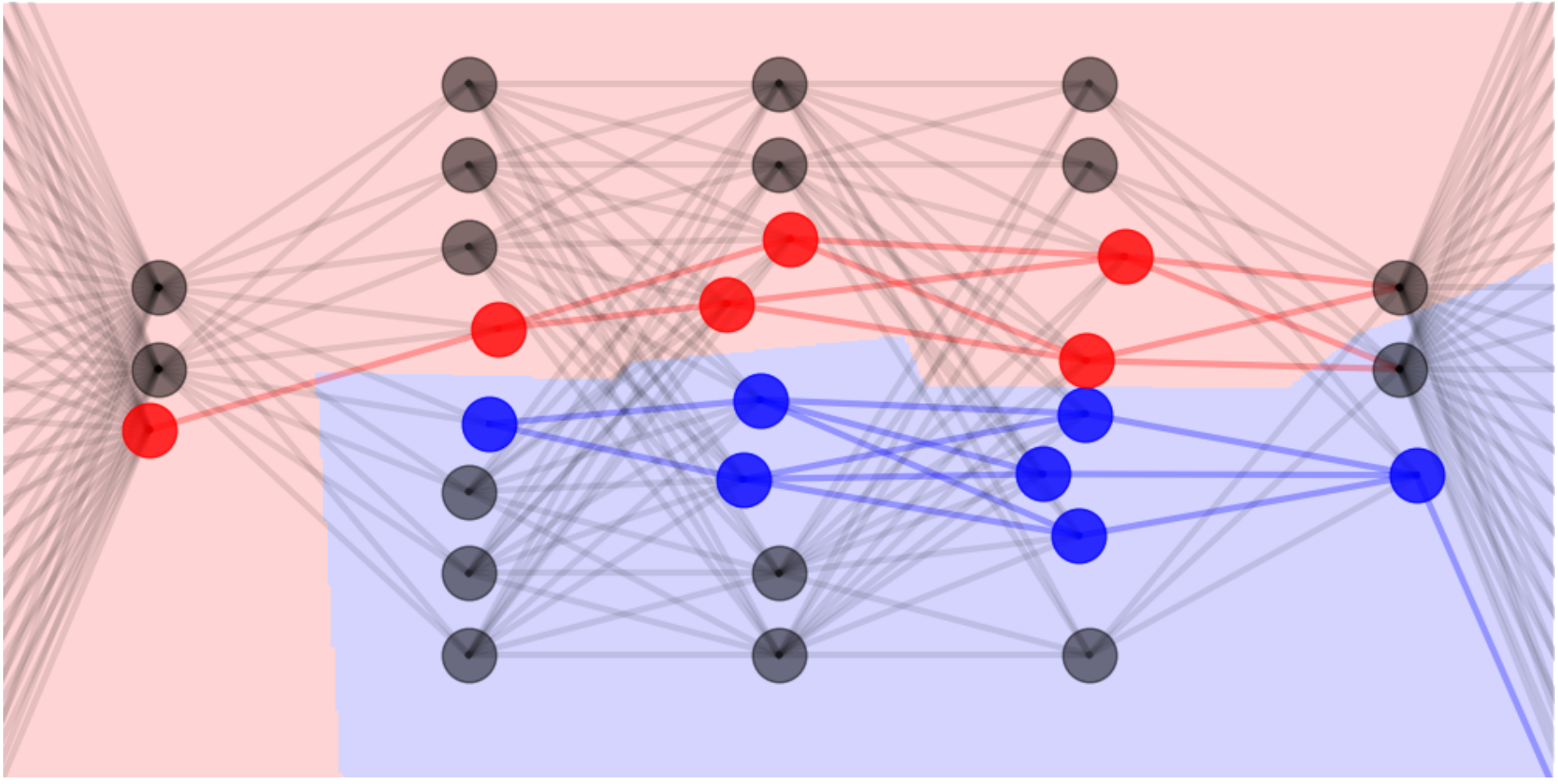
TU-Delft Machine and Deep Learning



Self-attention

Lecture 7

Lecturer: Jan van Gemert

Topic: Self-Attention

- Word embeddings
- Self-attention

D2L-book chapters: 11, 15.1
'Understanding Deep Learning' chapter 12 (https://udlbook.github.io/udlbook/),

Sequence models

Lets learn a model to predict the next word.

Consider these sentences:

The student, after reading the [old, thick, red, ...] book, was prepared.
The students, after reading the [old, thick, red, ...] book, were prepared.

Q: What is the difficulty here?

Sequence models

Lets learn a model to predict the next word.

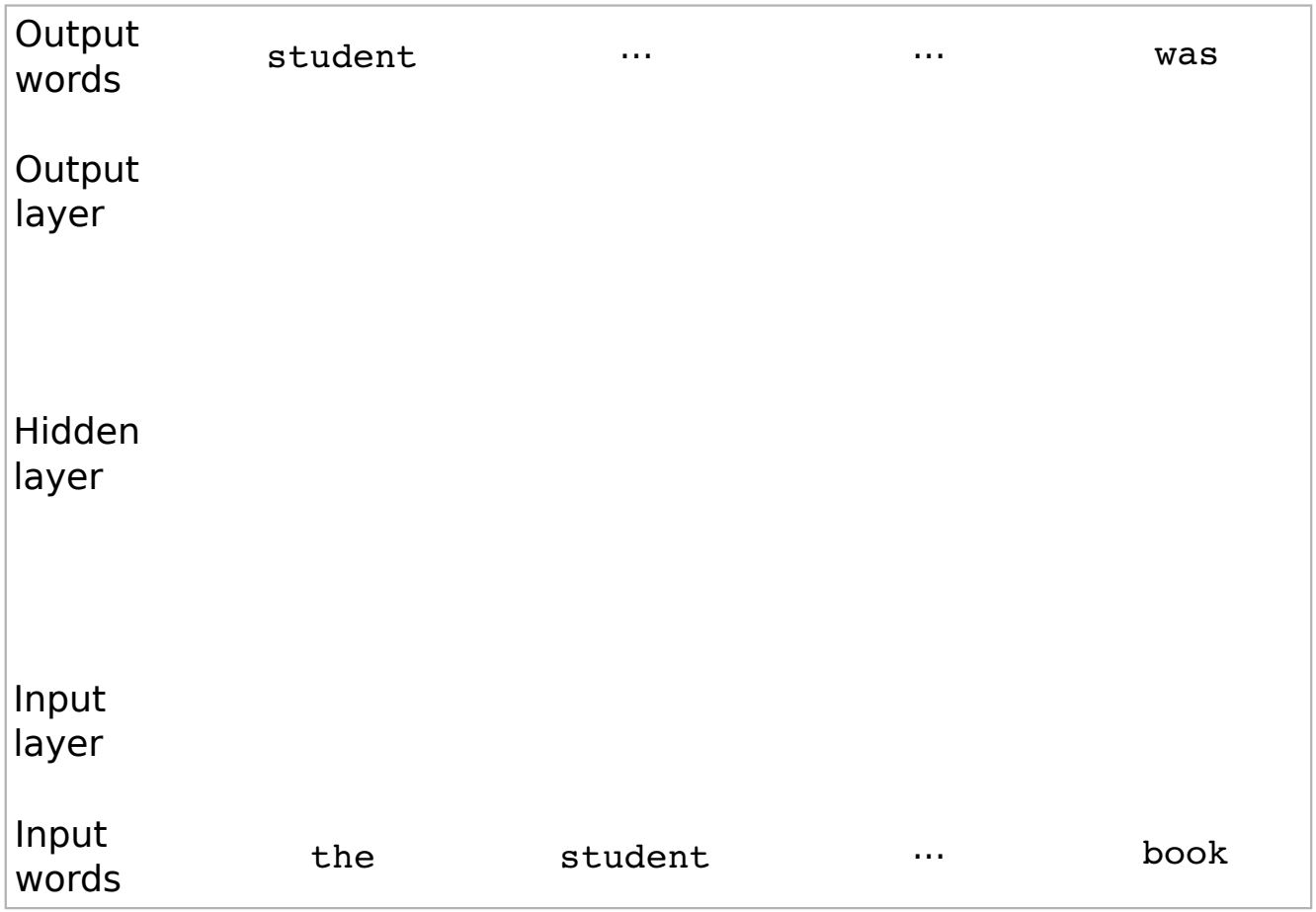
Consider these sentences:

The student, after reading the [old, thick, red, ...] book, was prepared.

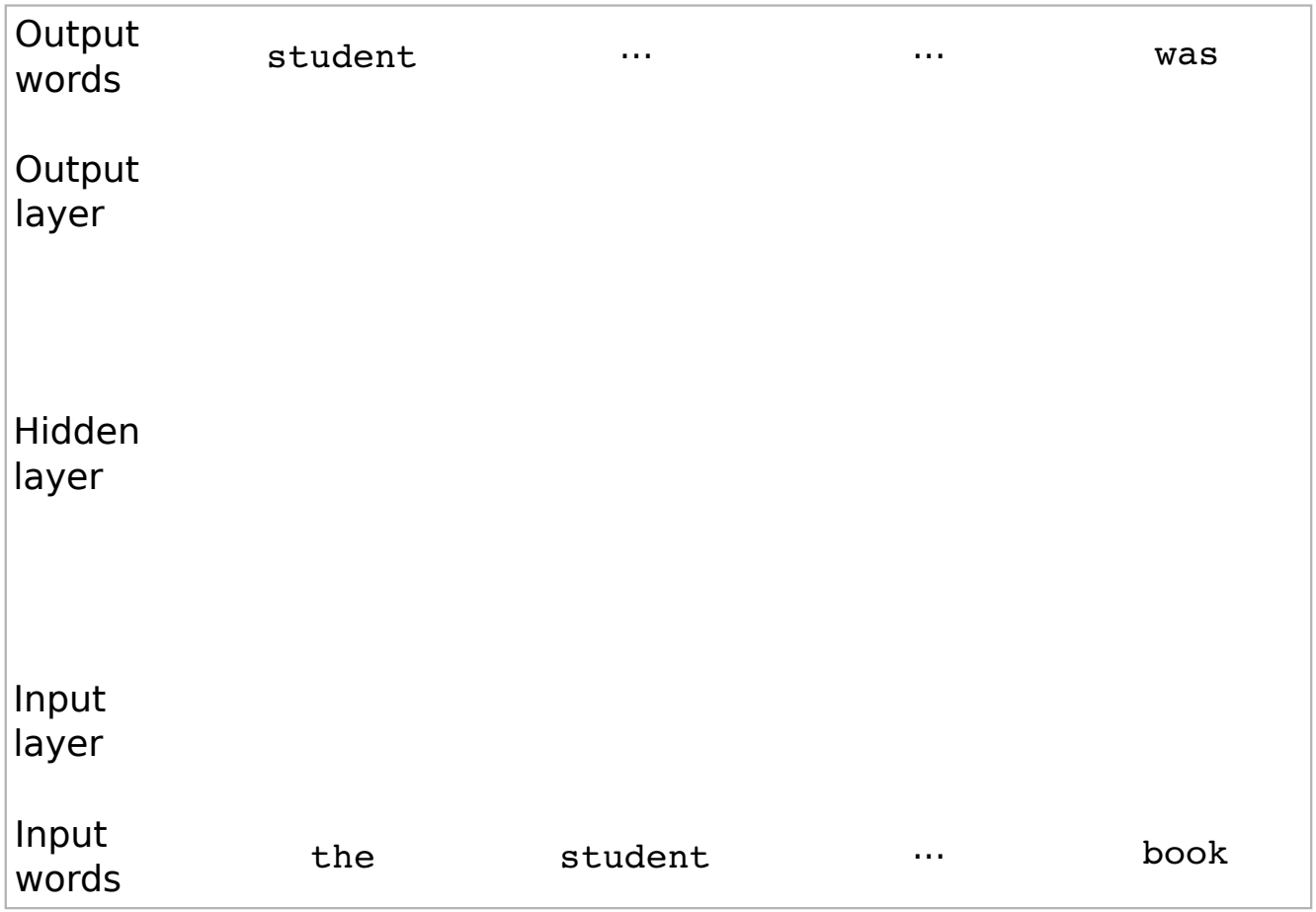
The students, after reading the [old, thick, red, ...] book, were prepared.

Q: What is the difficulty here?
A: Variable sized long distance relation between "student[s]" and "was[were]".

Recap: Predict the next word

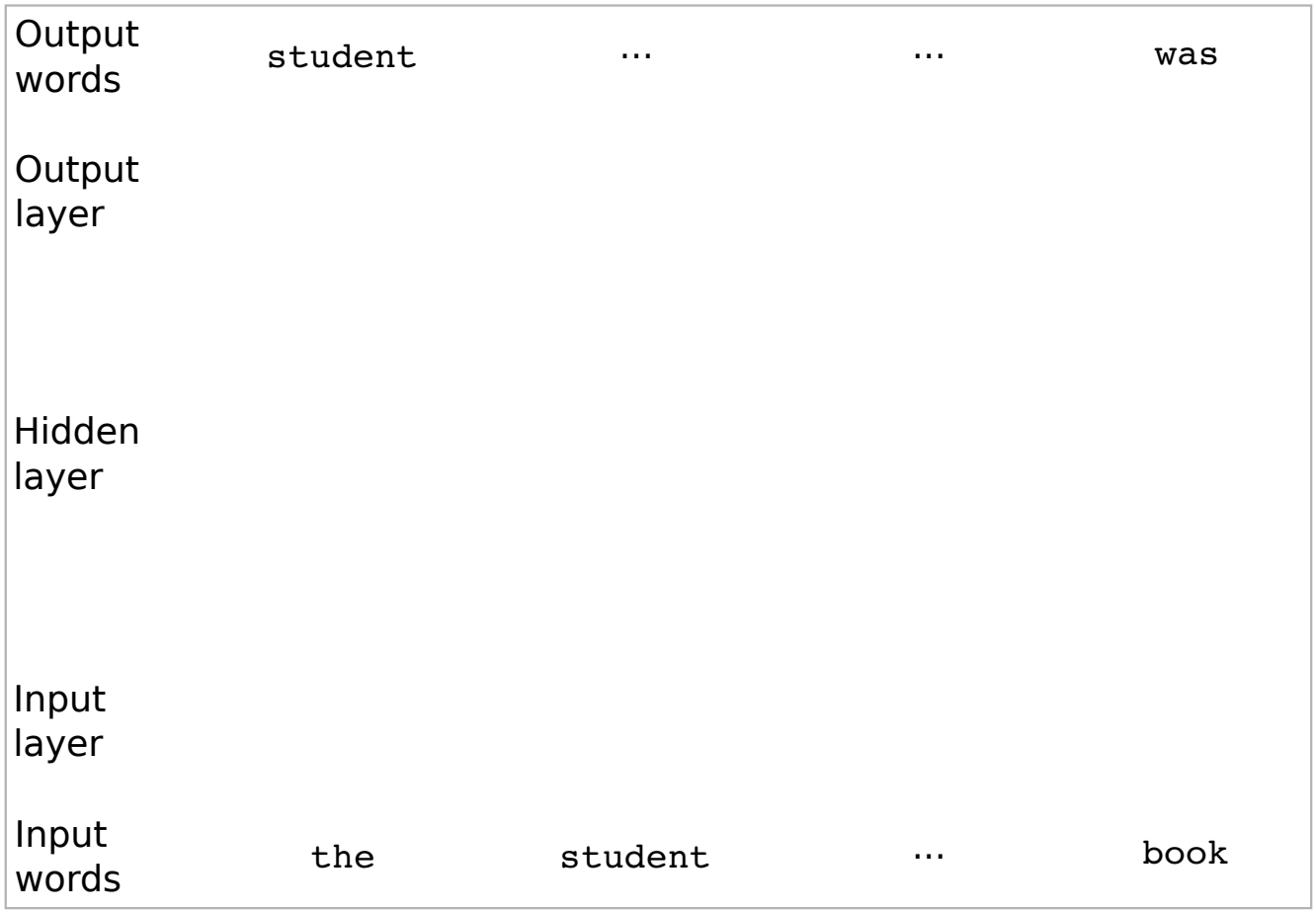


Recap: Predict the next word



2-layer network for word prediction.

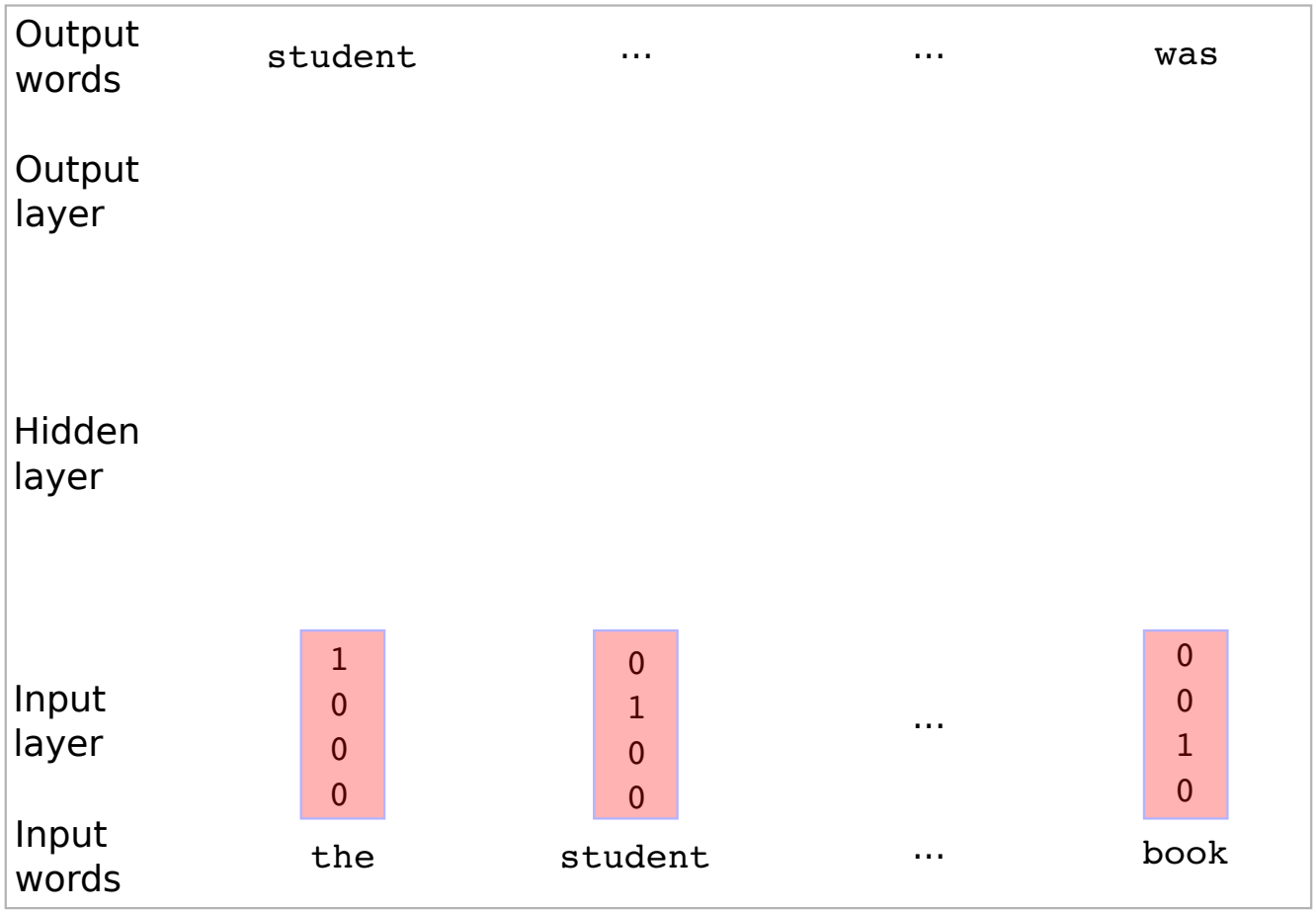
Recap: Predict the next word



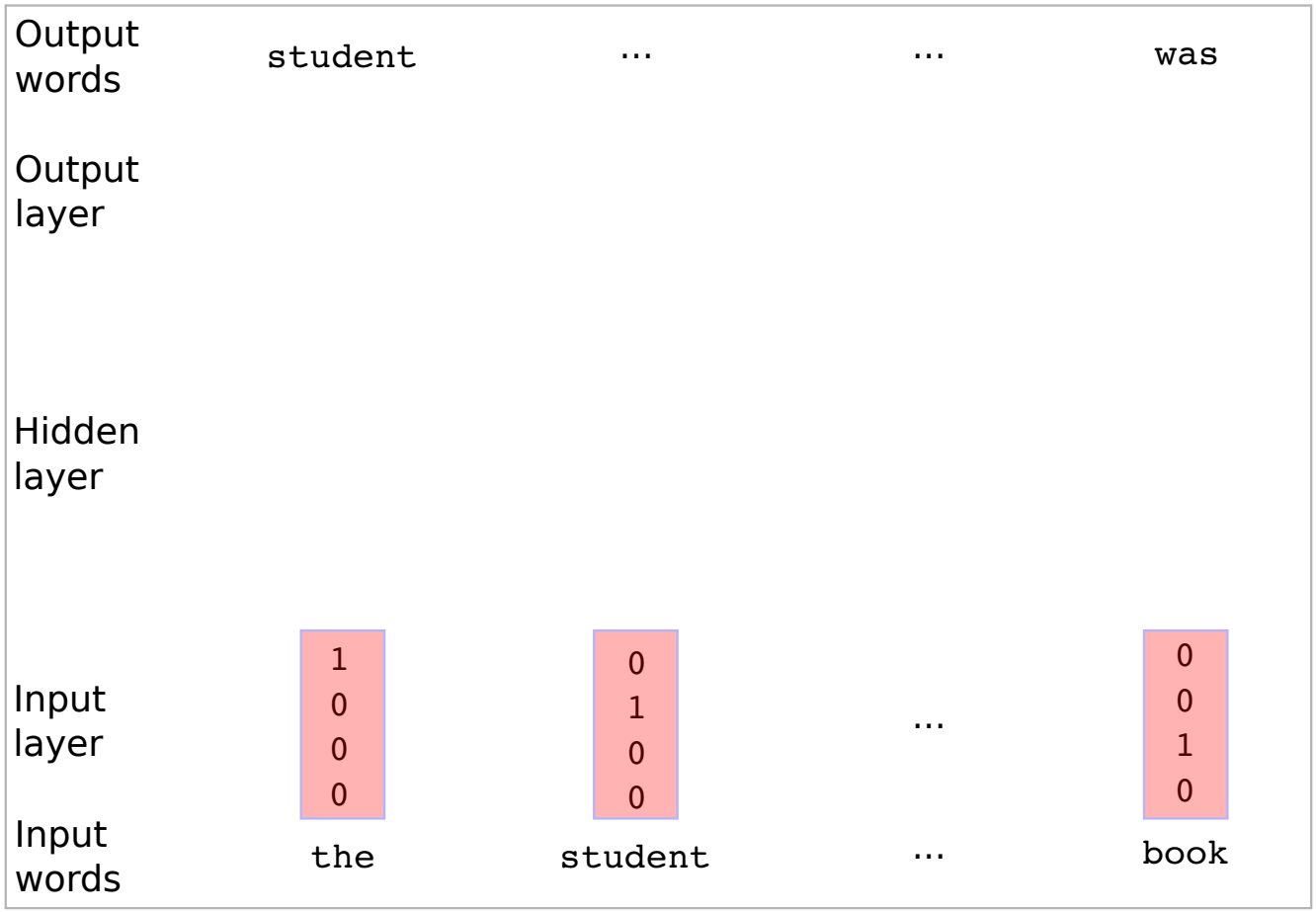
2-layer network for word prediction.

Q: How to represent input words?

Recap: Predict the next word

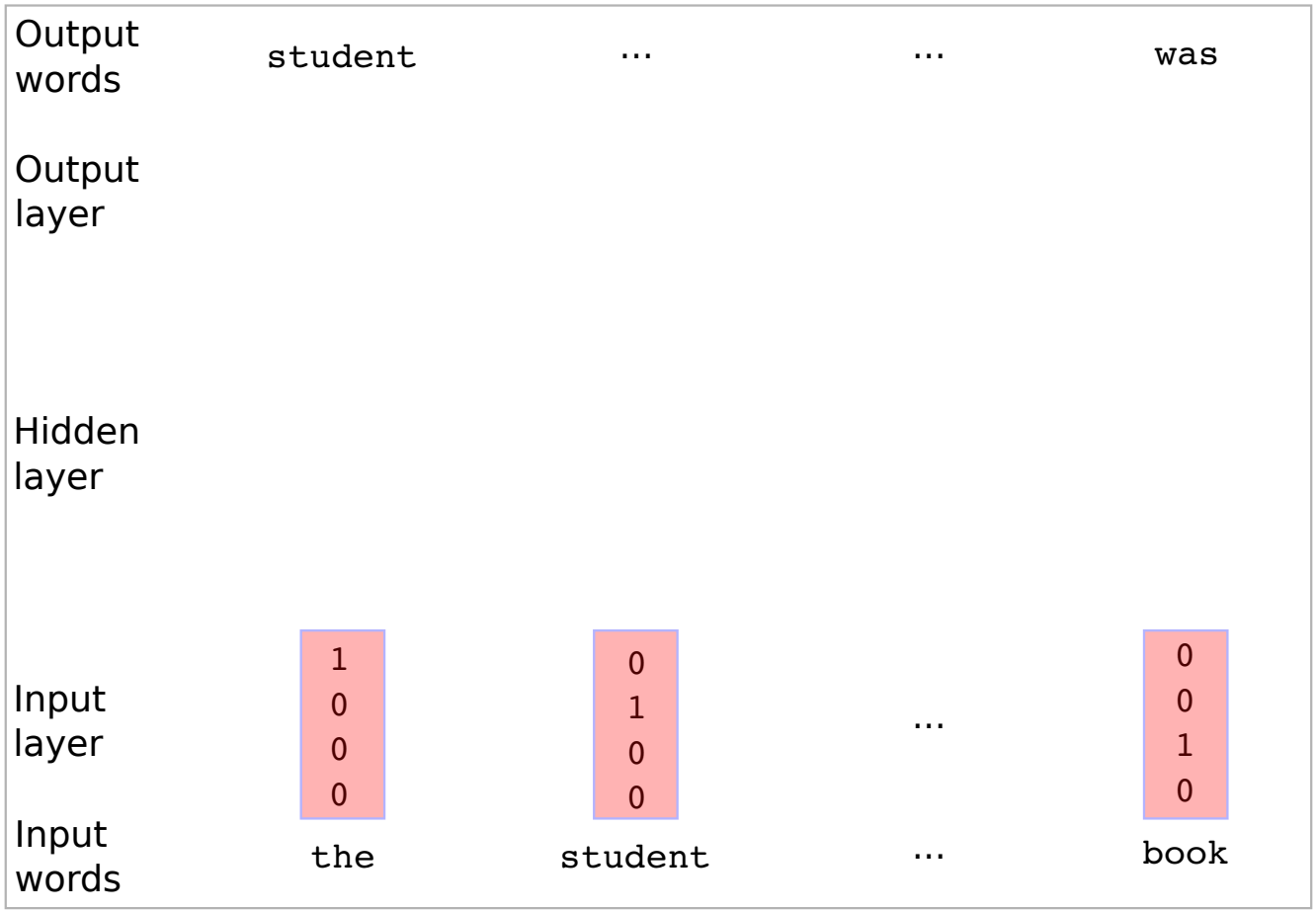


Recap: Predict the next word



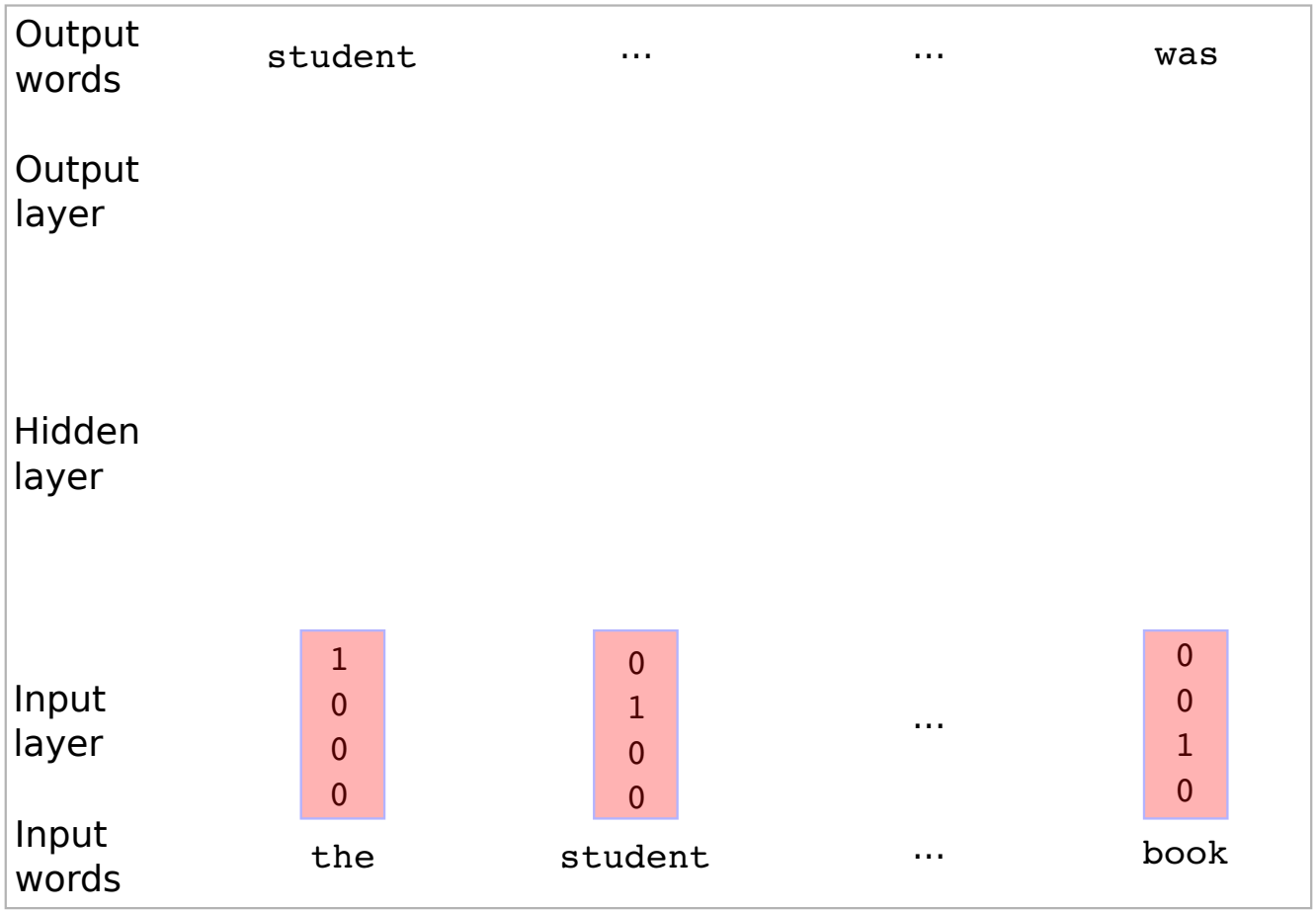
Q: What are the dot product similarities between input words?

Recap: Predict the next word



Q: What are the dot product similarities between input words? A: 0

Recap: Predict the next word



Q: What are the dot product similarities between input words? A: 0
Word embeddings: vector representation with semantic vector similarities.

Word2vec: Word embeddings

https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html
See for theoretical analysis: https://arxiv.org/abs/1402.3722

Word2vec: Word embeddings

https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html
See for theoretical analysis: https://arxiv.org/abs/1402.3722

Word2vec: turn a discrete word into a d-dimensional real valued feature vector.

Word2vec: Word embeddings

https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html
See for theoretical analysis: https://arxiv.org/abs/1402.3722

Word2vec: turn a discrete word into a d-dimensional real valued feature vector.

Word i from vocabulary $\mathbf{V}$ as a one-hot encoded vector $\mathbf{x}_i$: predict context

Word2vec: Word embeddings

https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html
See for theoretical analysis: https://arxiv.org/abs/1402.3722

Word2vec: turn a discrete word into a d-dimensional real valued feature vector.

Word i from vocabulary $\mathbf{V}$ as a one-hot encoded vector $\mathbf{x}_i$: predict context

Compute a word vector $\mathbf{h}_i = \mathbf{x}_i \mathbf{W}_{dv}$,

Word2vec: Word embeddings

https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html
See for theoretical analysis: https://arxiv.org/abs/1402.3722

Word2vec: turn a discrete word into a d-dimensional real valued feature vector.

Word i from vocabulary $\mathbf{V}$ as a one-hot encoded vector $\mathbf{x}_i$: predict context

Compute a word vector $\mathbf{h}_i = \mathbf{x}_i \mathbf{W}_{dv}$,
Use word vector $\mathbf{h}_i$ to predict context (surrounding words) $\mathbf{y}_i = \mathbf{W}_{vd} \mathbf{h}_i$

Word2vec: Word embeddings

https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html
See for theoretical analysis: https://arxiv.org/abs/1402.3722

Word2vec: turn a discrete word into a d-dimensional real valued feature vector.

Word i from vocabulary $\mathbf{V}$ as a one-hot encoded vector $\mathbf{x}_i$: predict context

Compute a word vector $\mathbf{h}_i = \mathbf{x}_i \mathbf{W}_{dv}$,
Use word vector $\mathbf{h}_i$ to predict context (surrounding words) $\mathbf{y}_i = \mathbf{W}_{vd} \mathbf{h}_i$

Two ways to predict context:

Word2vec: Word embeddings

https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html
See for theoretical analysis: https://arxiv.org/abs/1402.3722

Word2vec: turn a discrete word into a d-dimensional real valued feature vector.

Word i from vocabulary $\mathbf{V}$ as a one-hot encoded vector $\mathbf{x}_i$: predict context

Compute a word vector $\mathbf{h}_i = \mathbf{x}_i \mathbf{W}_{dv}$,
Use word vector $\mathbf{h}_i$ to predict context (surrounding words) $\mathbf{y}_i = \mathbf{W}_{vd} \mathbf{h}_i$

Two ways to predict context:

- From center word to k surrounding words (Skip gram)

Word2vec: Word embeddings

https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html
See for theoretical analysis: https://arxiv.org/abs/1402.3722

Word2vec: turn a discrete word into a d-dimensional real valued feature vector.

Word i from vocabulary $\mathbf{V}$ as a one-hot encoded vector $\mathbf{x}_i$: predict context

Compute a word vector $\mathbf{h}_i = \mathbf{x}_i \mathbf{W}_{dv}$,
Use word vector $\mathbf{h}_i$ to predict context (surrounding words) $\mathbf{y}_i = \mathbf{W}_{vd} \mathbf{h}_i$

Two ways to predict context:

- From center word to k surrounding words (Skip gram)
- From surrounding k words to center word (CBOW: Continuous bag-of-words)

Word2vec: Word embeddings

https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html
See for theoretical analysis: https://arxiv.org/abs/1402.3722

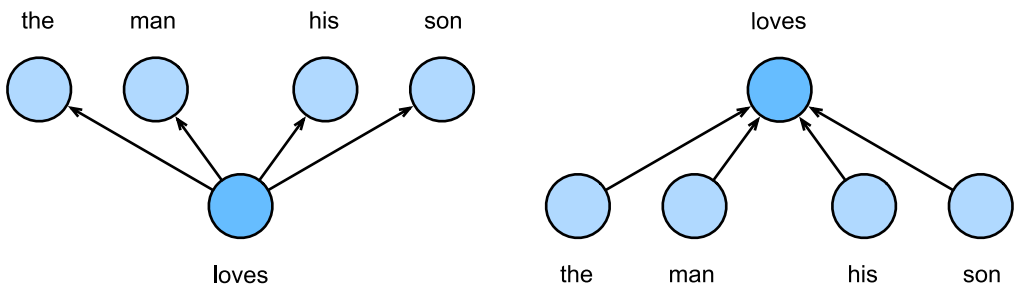
Word2vec: turn a discrete word into a d-dimensional real valued feature vector.

Word i from vocabulary $\mathbf{V}$ as a one-hot encoded vector $\mathbf{x}_i$: predict context

Compute a word vector $\mathbf{h}_i = \mathbf{x}_i \mathbf{W}_{dv}$,
Use word vector $\mathbf{h}_i$ to predict context (surrounding words) $\mathbf{y}_i = \mathbf{W}_{vd} \mathbf{h}_i$

Two ways to predict context:

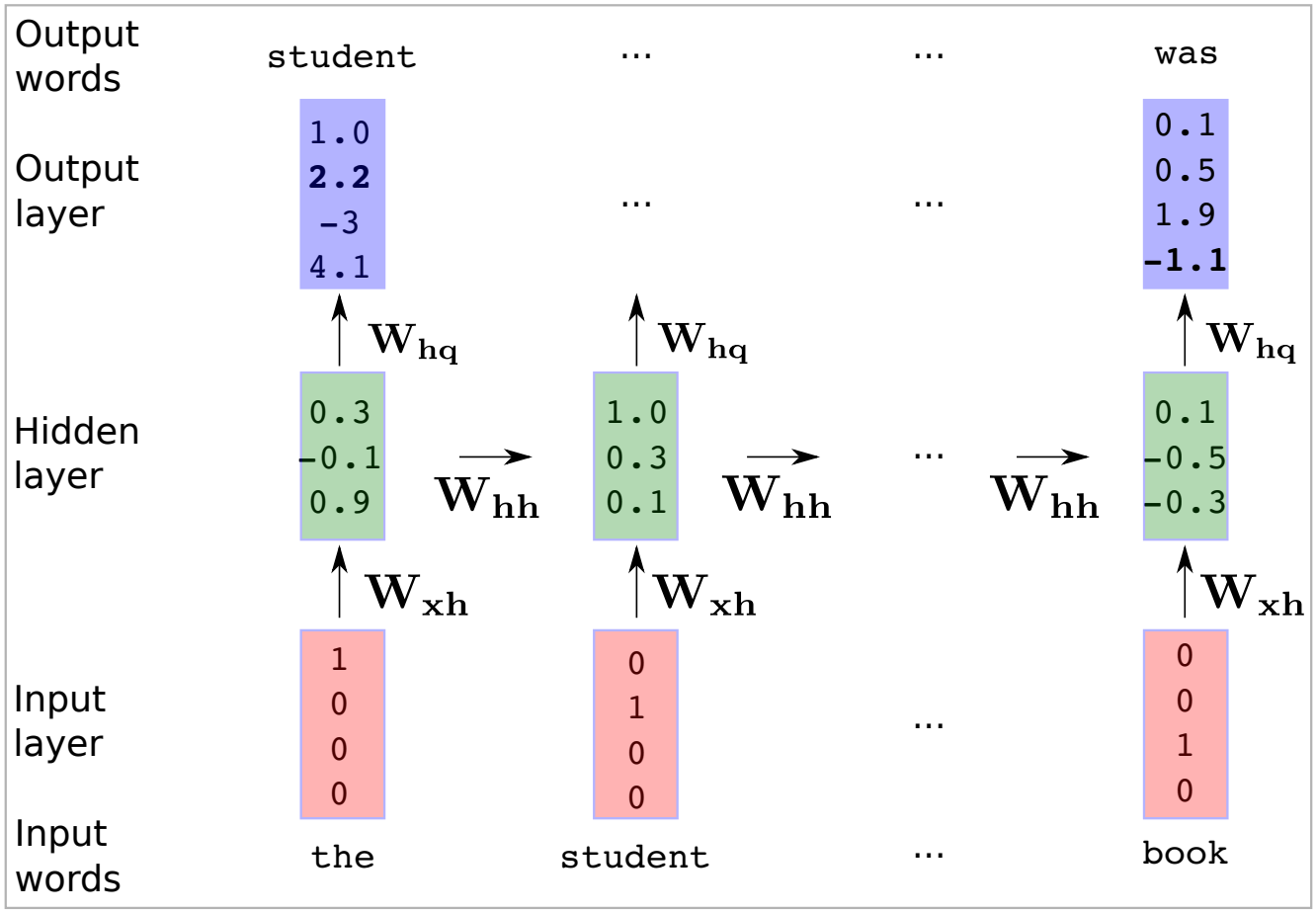
- From center word to k surrounding words (Skip gram)
- From surrounding k words to center word (CBOW: Continuous bag-of-words)



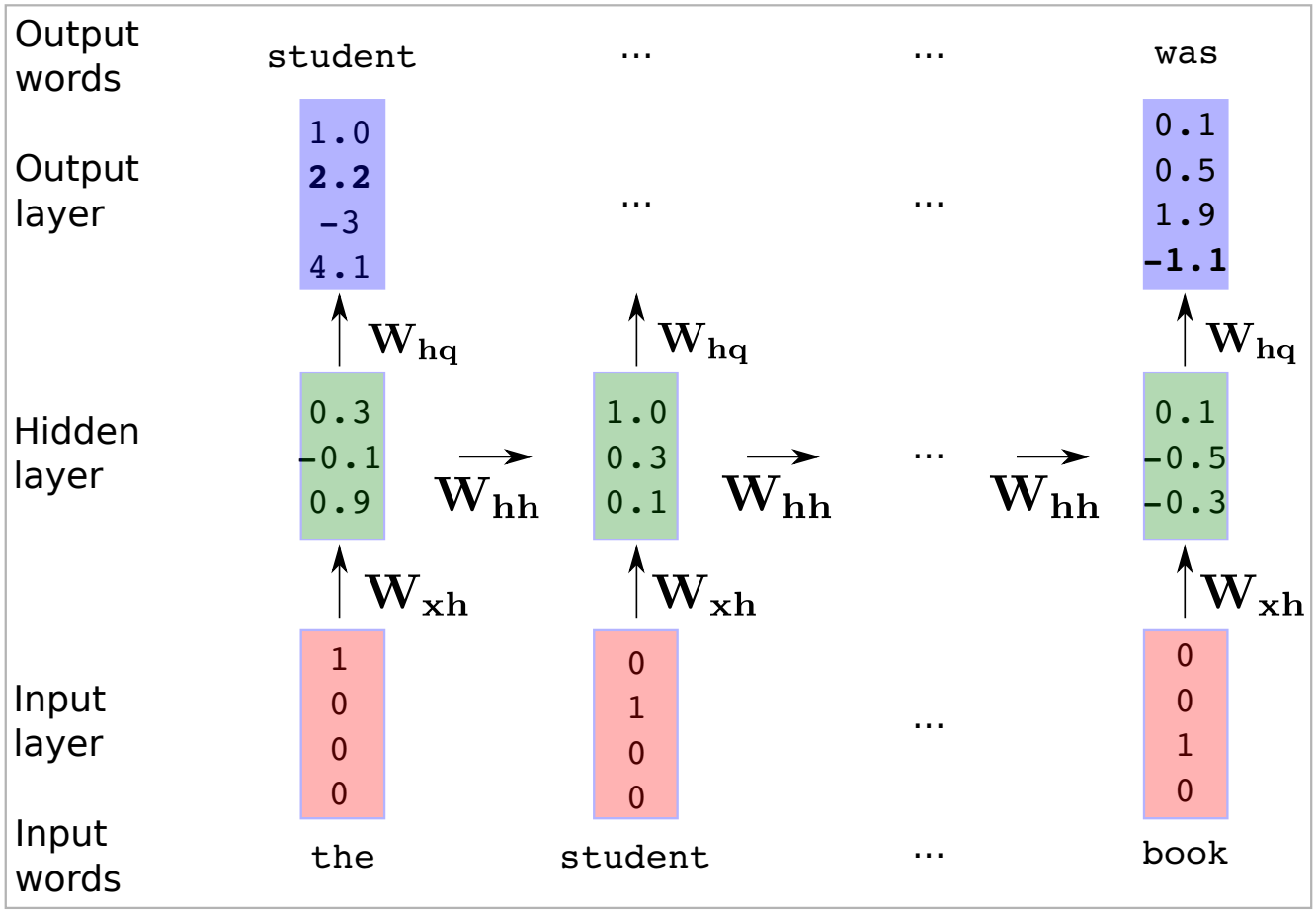
Skip gram CBOW

Questions?

Recap: Predict the next word with an RNN

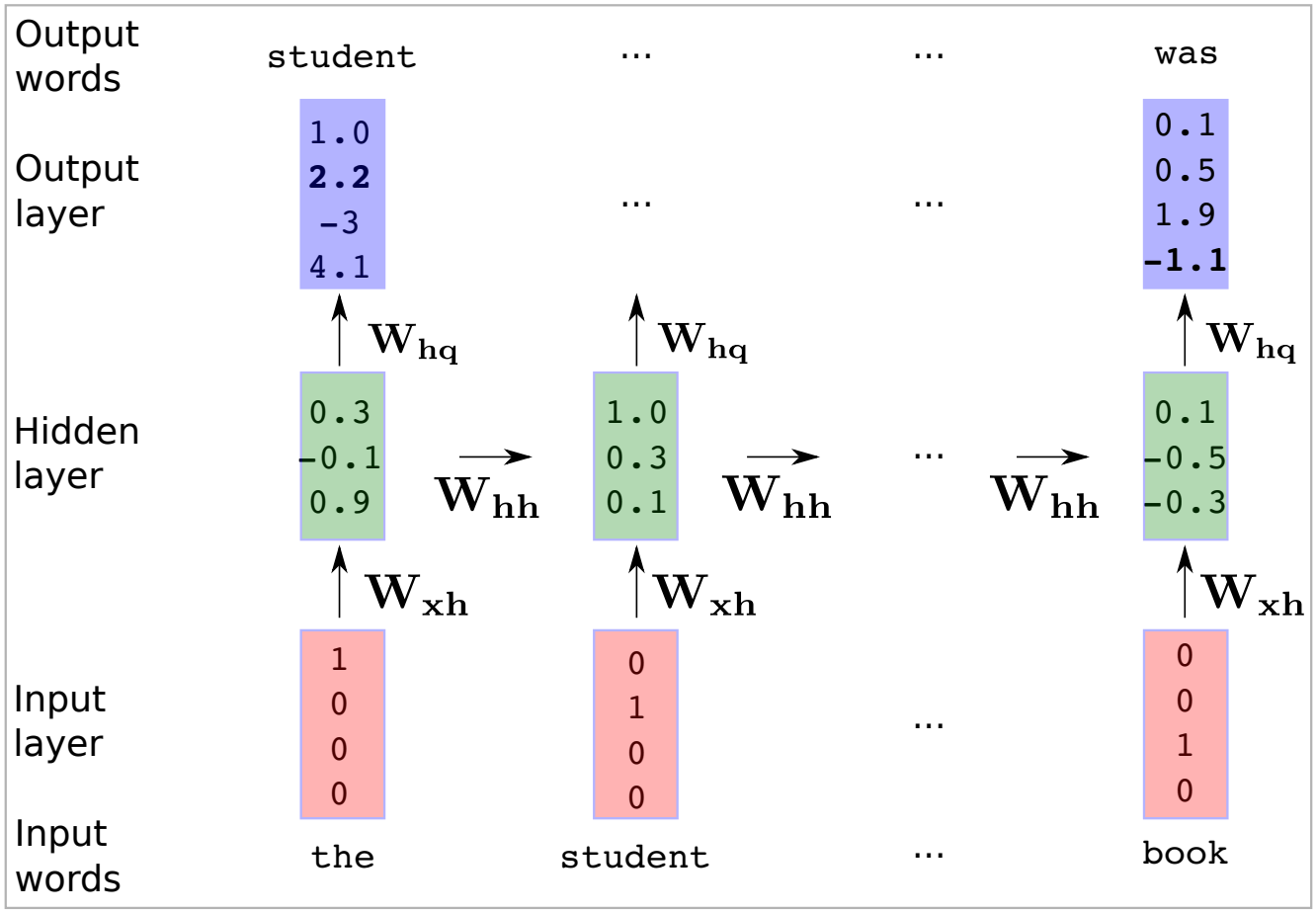


Recap: Predict the next word with an RNN



Recurrent net, with shared weights at each time step.

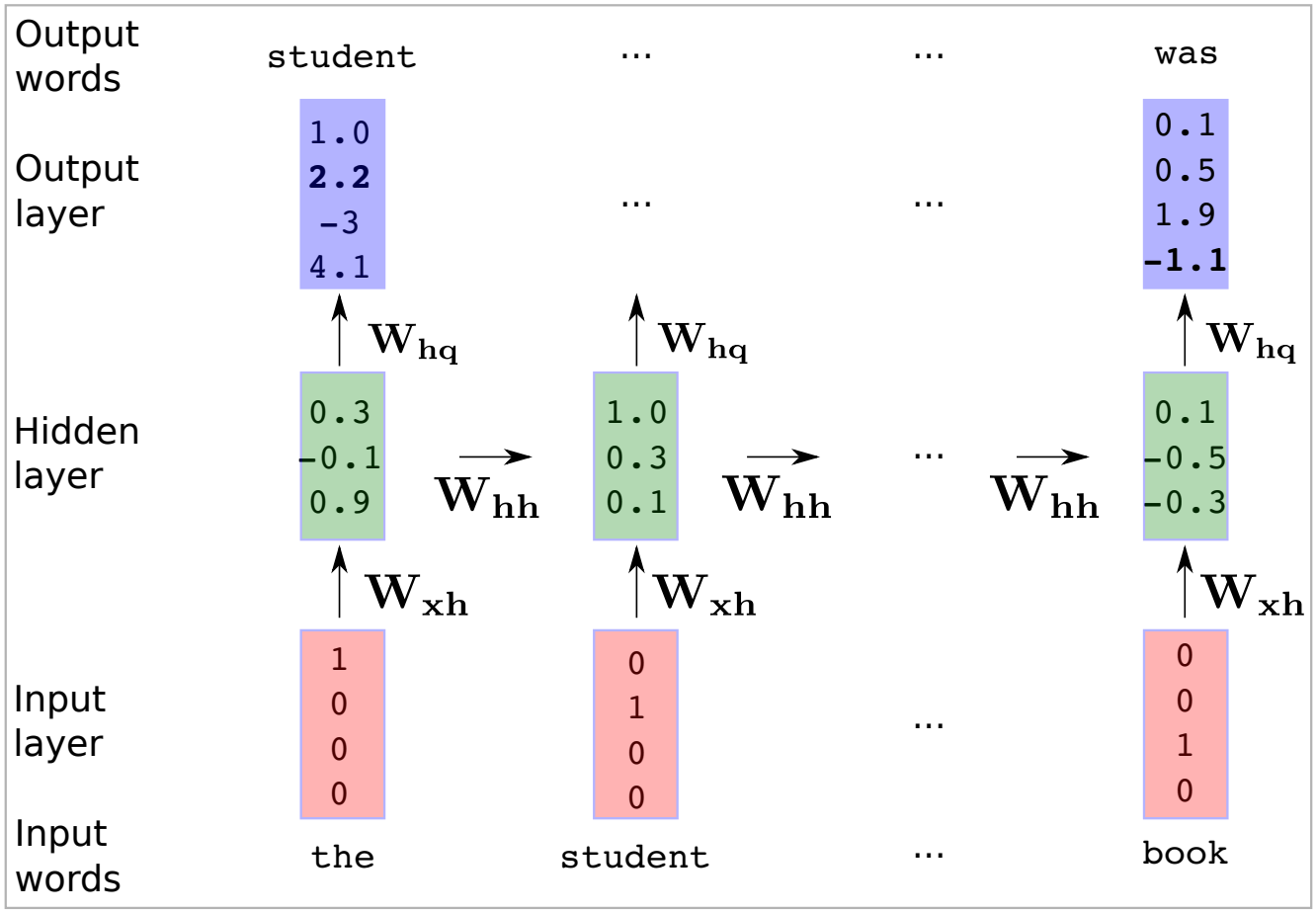
Recap: Predict the next word with an RNN



Recurrent net, with shared weights at each time step.

Q: Why are recurrent nets (RNN/GRU/LSTM) difficult to train in parallel?

Recap: Predict the next word with an RNN

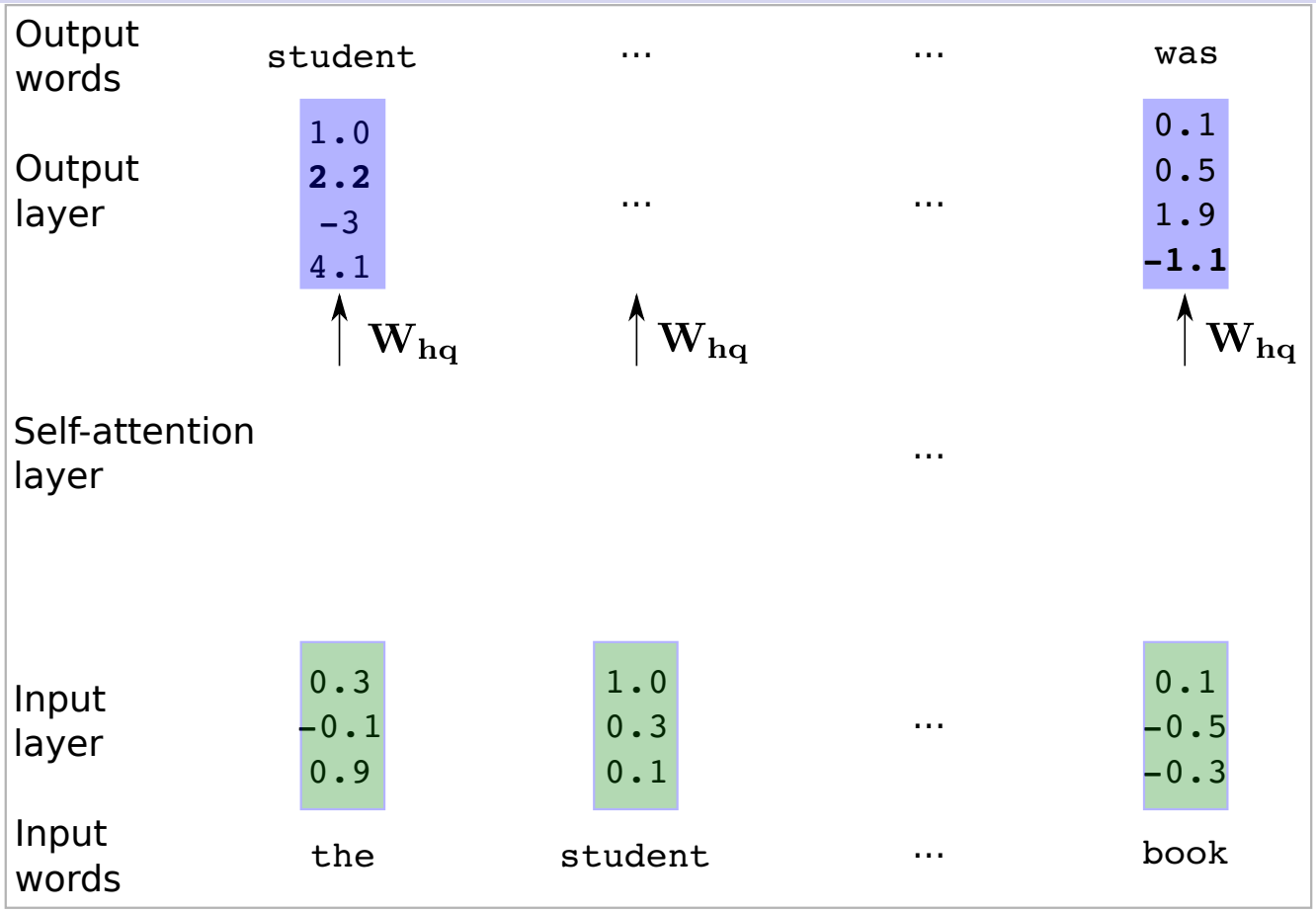


Recurrent net, with shared weights at each time step.

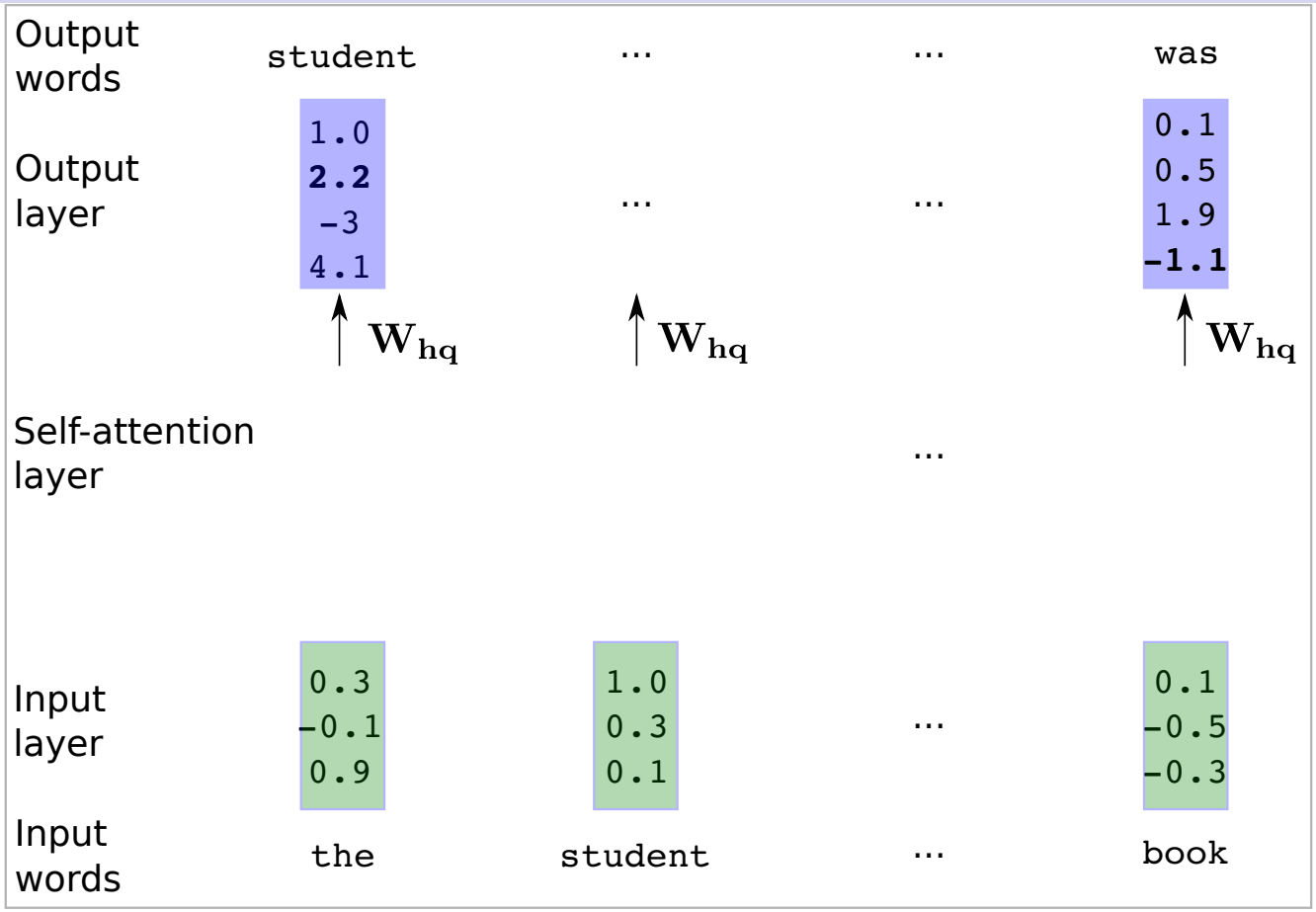
Q: Why are recurrent nets (RNN/GRU/LSTM) difficult to train in parallel?

A: Inherently sequential: Each step depends on the previous.

Self-attention: Removing the recurrence

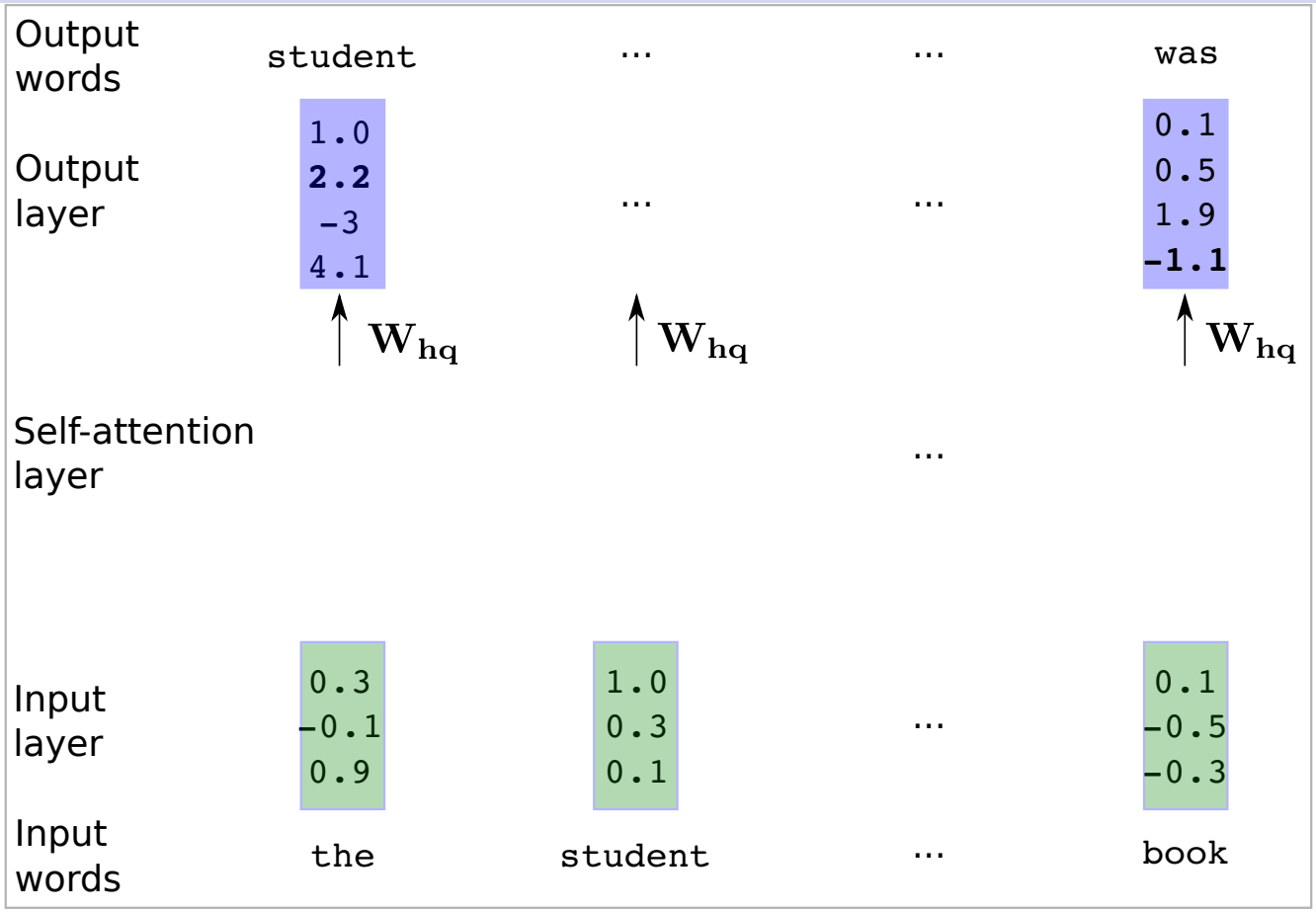


Self-attention: Removing the recurrence



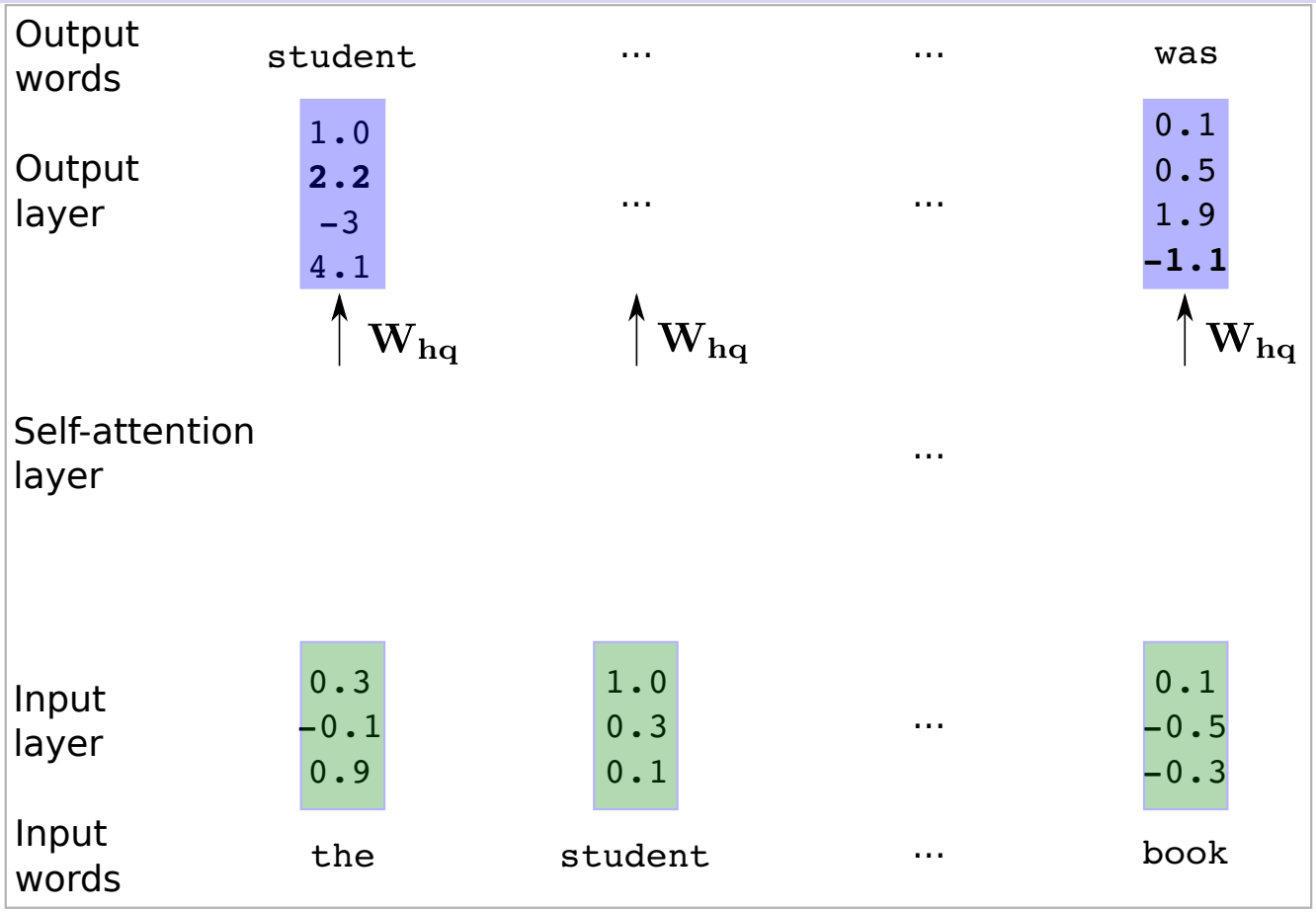
Q: Whats happening at input layer?

Self-attention: Removing the recurrence



Q: Whats happening at input layer? A: Word embedding vectors

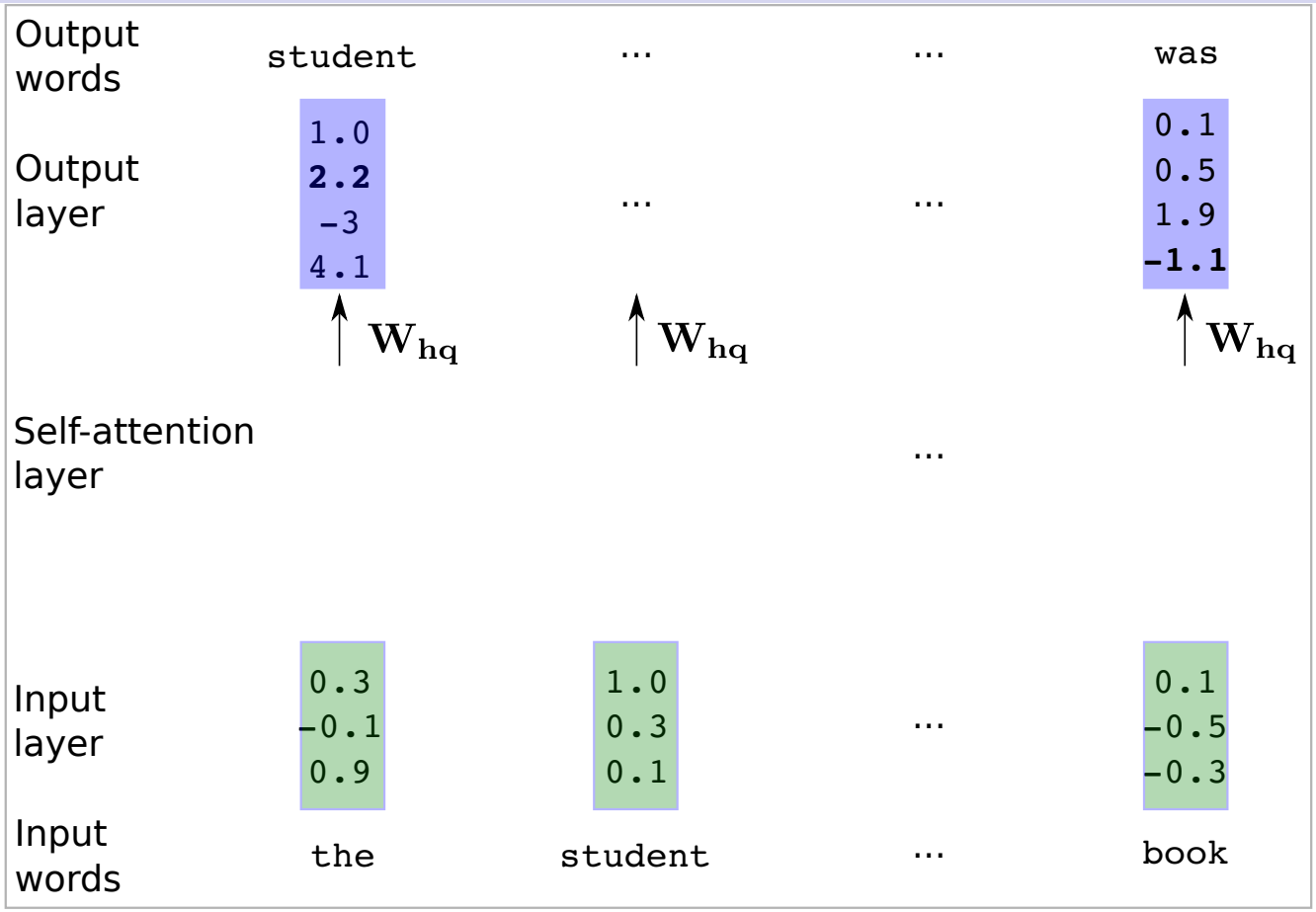
Self-attention: Removing the recurrence



Q: Whats happening at input layer? A: Word embedding vectors

Q: Whats happening at output layer?

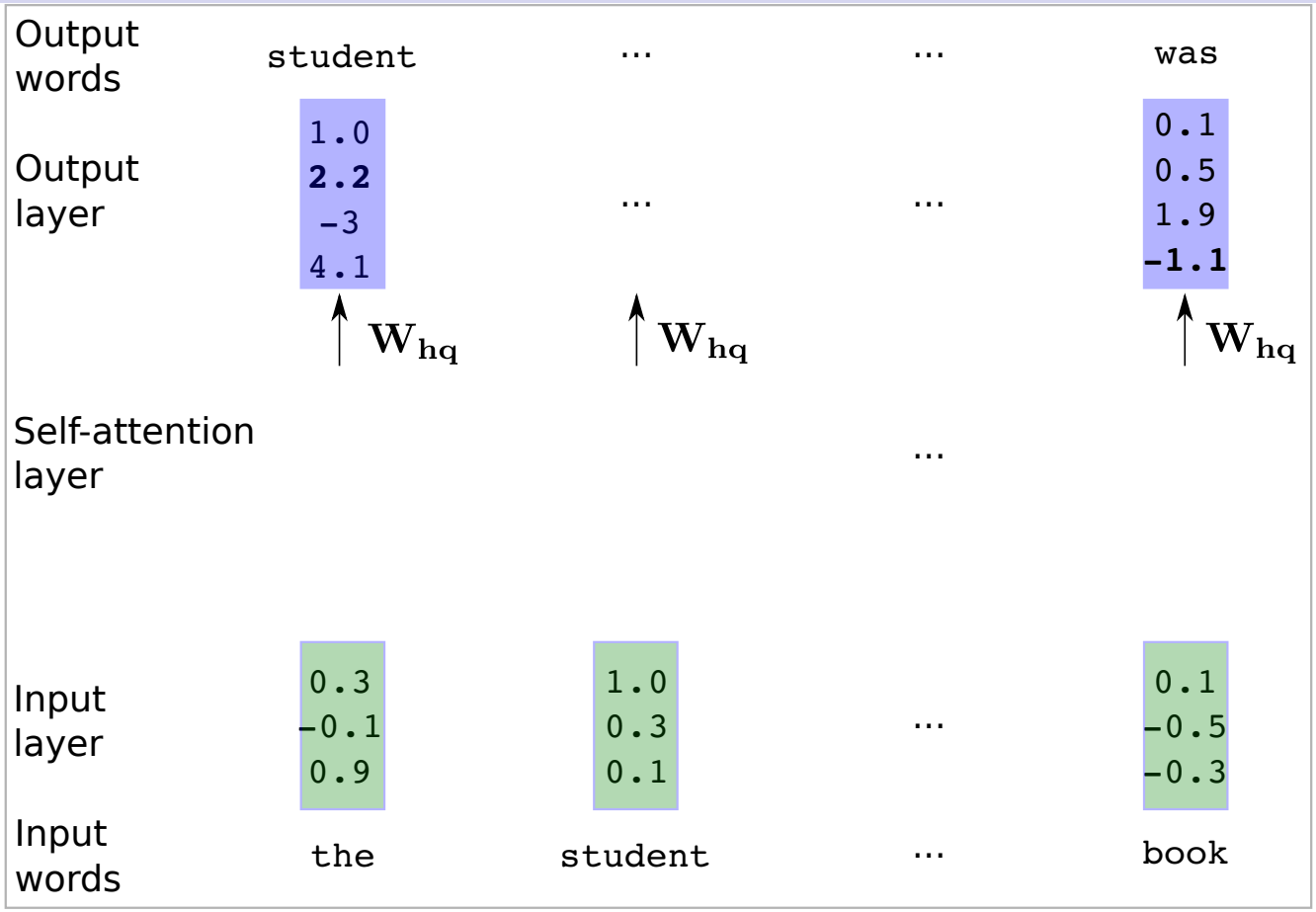
Self-attention: Removing the recurrence



Q: Whats happening at input layer? A: Word embedding vectors

Q: Whats happening at output layer? A: Shared (same) weights per word.

Self-attention: Removing the recurrence

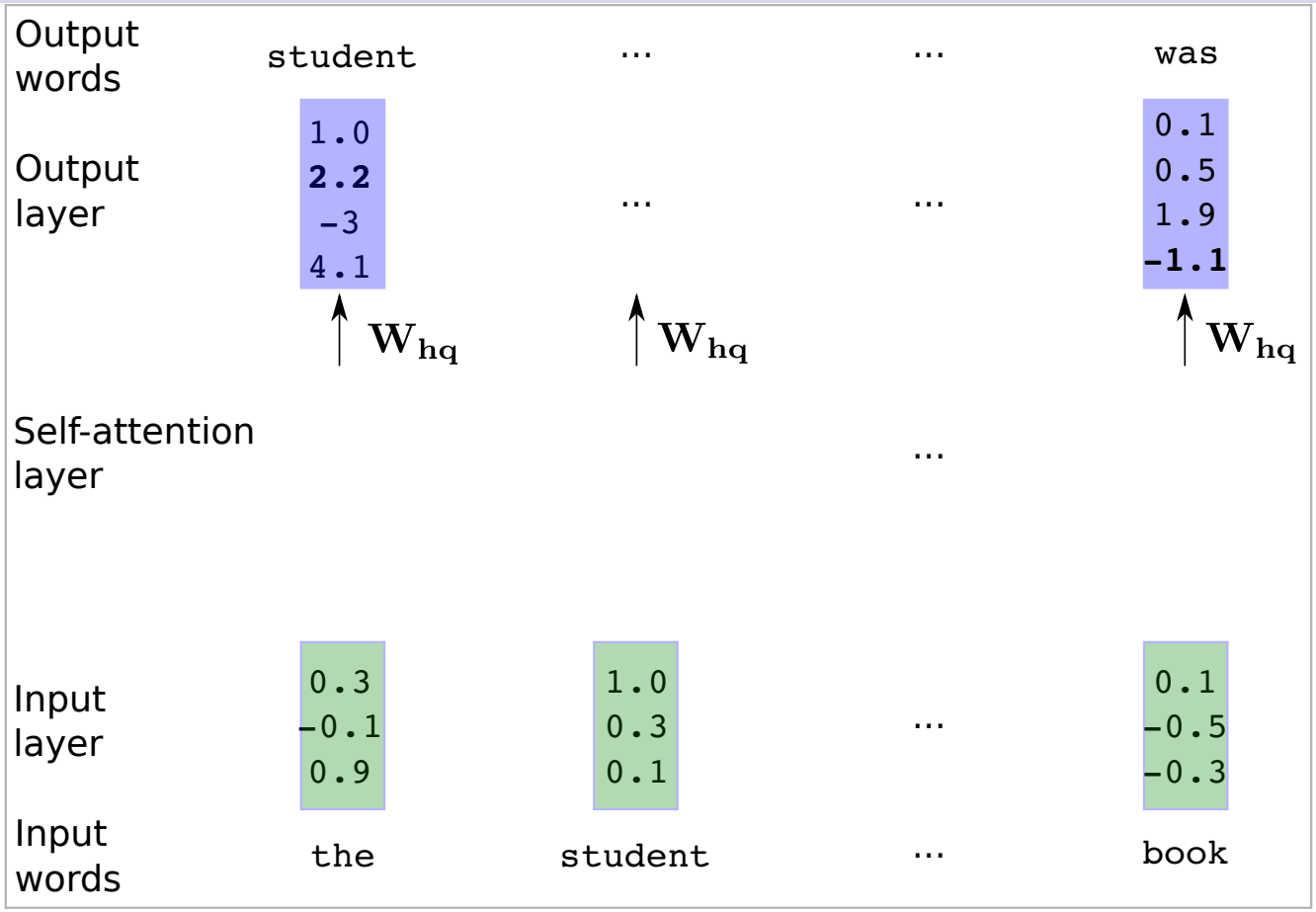


Q: Whats happening at input layer? A: Word embedding vectors

Q: Whats happening at output layer? A: Shared (same) weights per word.

How to remove recurrence, yet introduce word relationships...

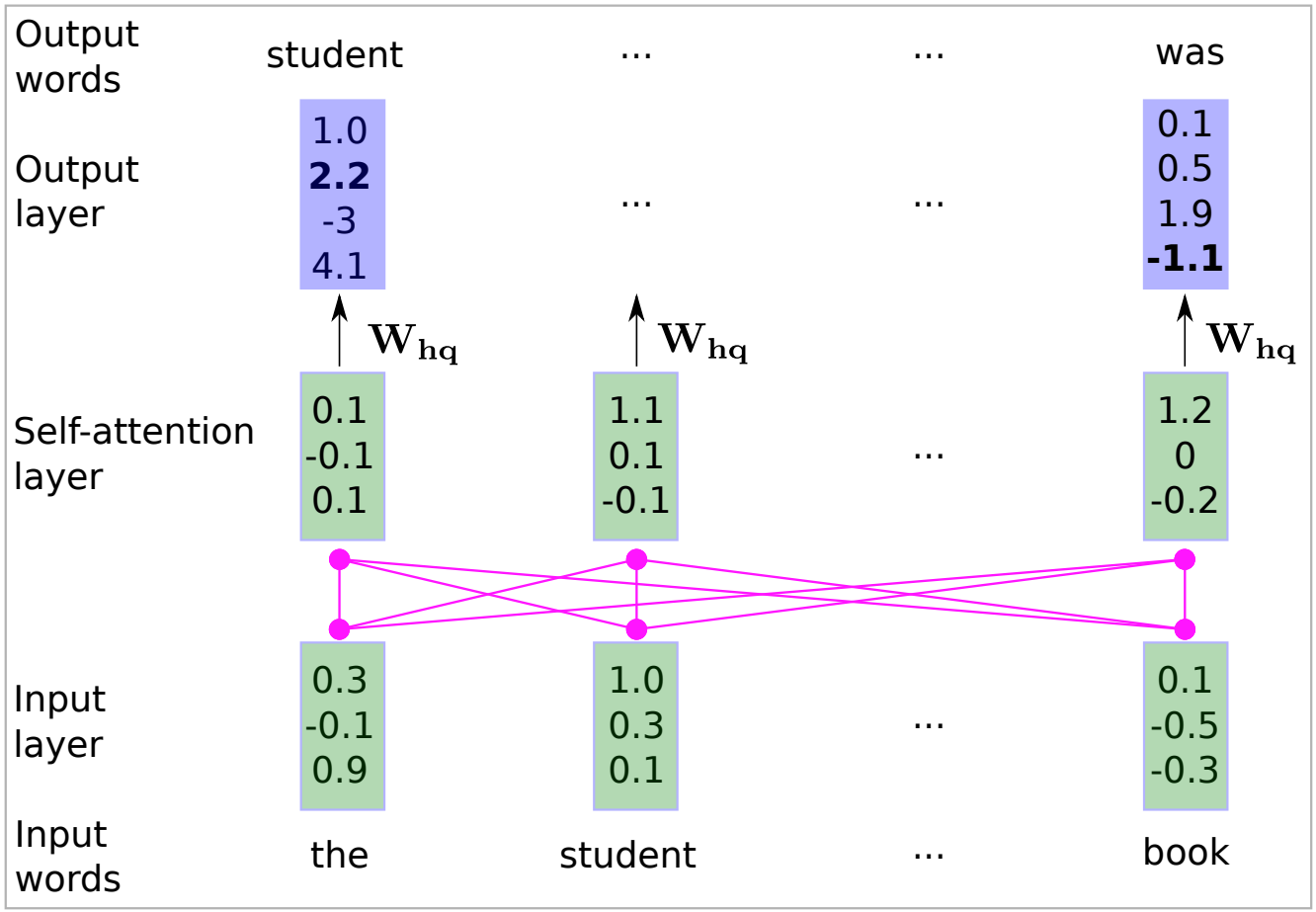
Self-attention: Removing the recurrence



Q: Whats happening at input layer? A: Word embedding vectors
Q: Whats happening at output layer? A: Shared (same) weights per word.

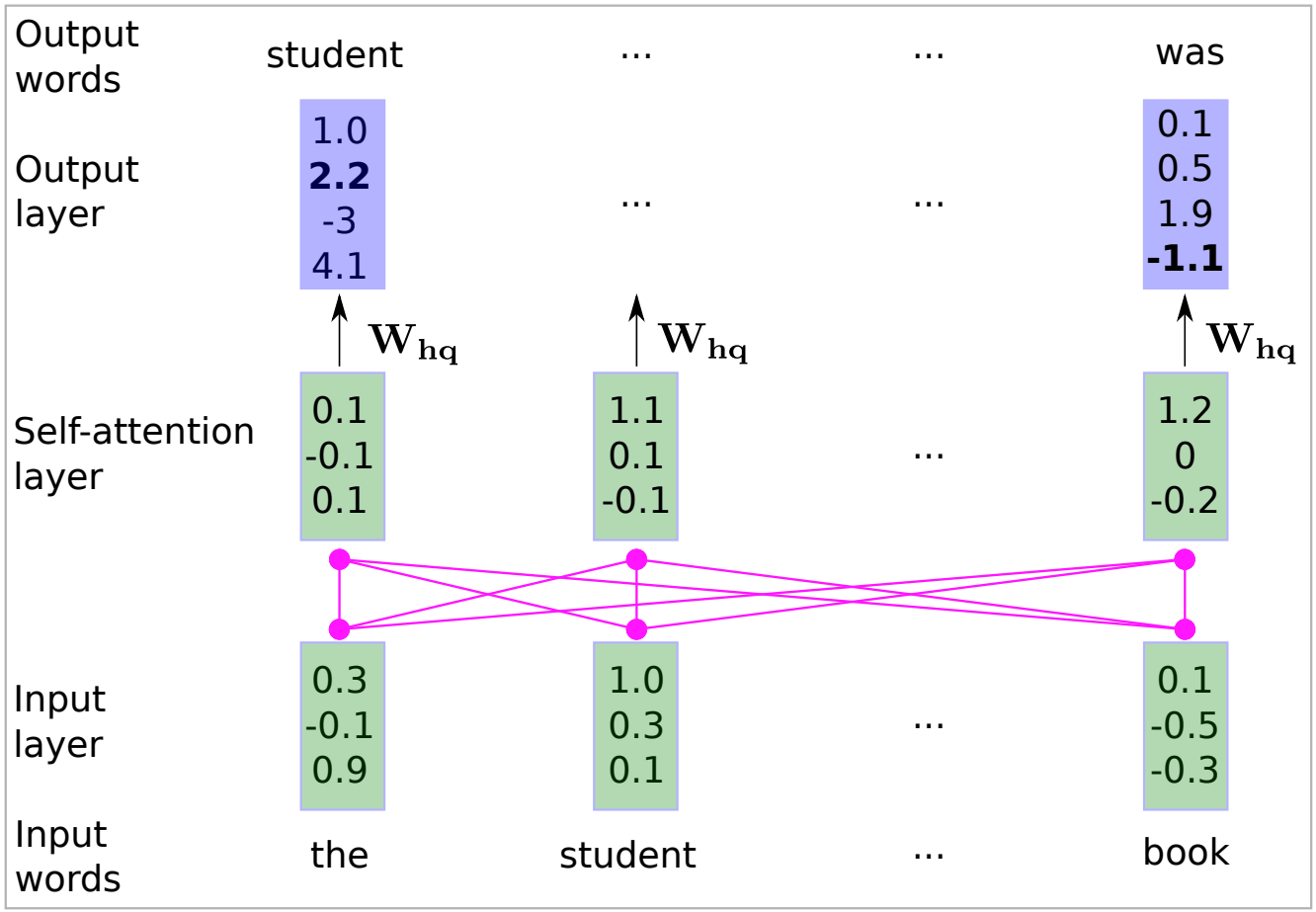
How to remove recurrence, yet introduce word relationships...
Somehow link all words to each other, without hard-coded positional weights.

Self-attention: Removing the recurrence



Note: purple lines do **not** represent individual weights; What they are will follow.

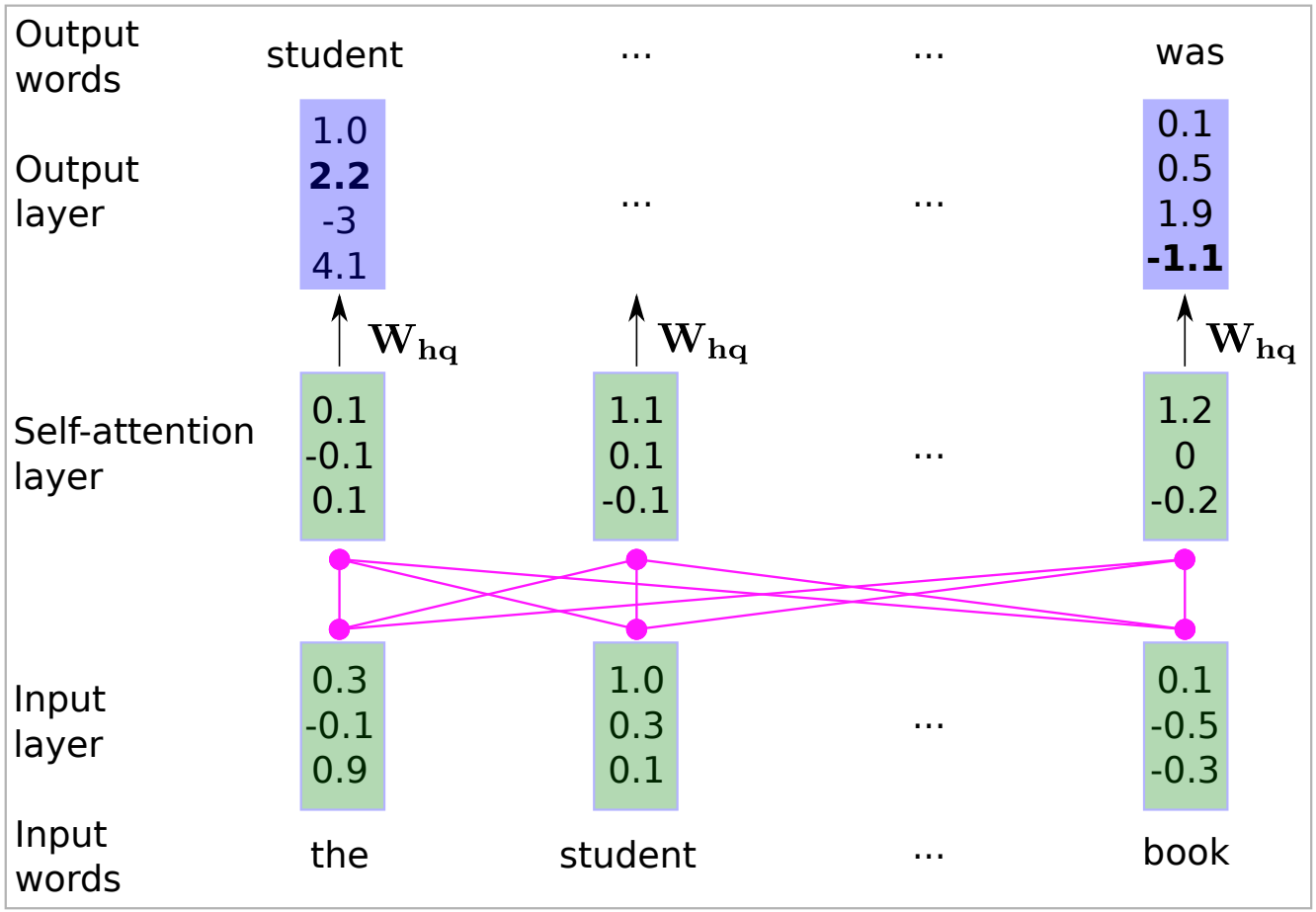
Self-attention: Removing the recurrence



Note: purple lines do **not** represent individual weights; What they are will follow.

Q: How many words influence a single word vector in the *self-attention* layer?

Self-attention: Removing the recurrence



Note: purple lines do **not** represent individual weights; What they are will follow.

Q: How many words influence a single word vector in the *self-attention* layer?

A: Influenced and changed by all other words.

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention: Parallelizable alternative to training recurrent nets

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention: Parallelizable alternative to training recurrent nets

Self attention is a sequence-to-sequence operation.

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention: Parallelizable alternative to training recurrent nets

Self attention is a sequence-to-sequence operation.

- Input vectors/tokens: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, of size d
- Output vectors/tokens: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t$, of size d

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention: Parallelizable alternative to training recurrent nets

Self attention is a sequence-to-sequence operation.

- Input vectors/tokens: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, of size d
- Output vectors/tokens: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t$, of size d

Start with words: *the, student, reads, a, book*

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention: Parallelizable alternative to training recurrent nets

Self attention is a sequence-to-sequence operation.

- Input vectors/tokens: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, of size d
- Output vectors/tokens: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention: Parallelizable alternative to training recurrent nets

Self attention is a sequence-to-sequence operation.

- Input vectors/tokens: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, of size d
- Output vectors/tokens: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention: Parallelizable alternative to training recurrent nets

Self attention is a sequence-to-sequence operation.

- Input vectors/tokens: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, of size d
- Output vectors/tokens: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Each self-attention layer transforms a sequence to another sequence.

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention: Parallelizable alternative to training recurrent nets

Self attention is a sequence-to-sequence operation.

- Input vectors/tokens: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, of size d
- Output vectors/tokens: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Each self-attention layer transforms a sequence to another sequence.

Self-attention: Learning how to re-weigh words with all other words.

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention: Parallelizable alternative to training recurrent nets

Self attention is a sequence-to-sequence operation.

- Input vectors/tokens: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, of size d
- Output vectors/tokens: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Each self-attention layer transforms a sequence to another sequence.

Self-attention: Learning how to re-weigh words with all other words.

Q: Why is self-attention parallelizable?

Self attention

https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

Self attention: Parallelizable alternative to training recurrent nets

Self attention is a sequence-to-sequence operation.

- Input vectors/tokens: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, of size d
- Output vectors/tokens: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t$, of size d

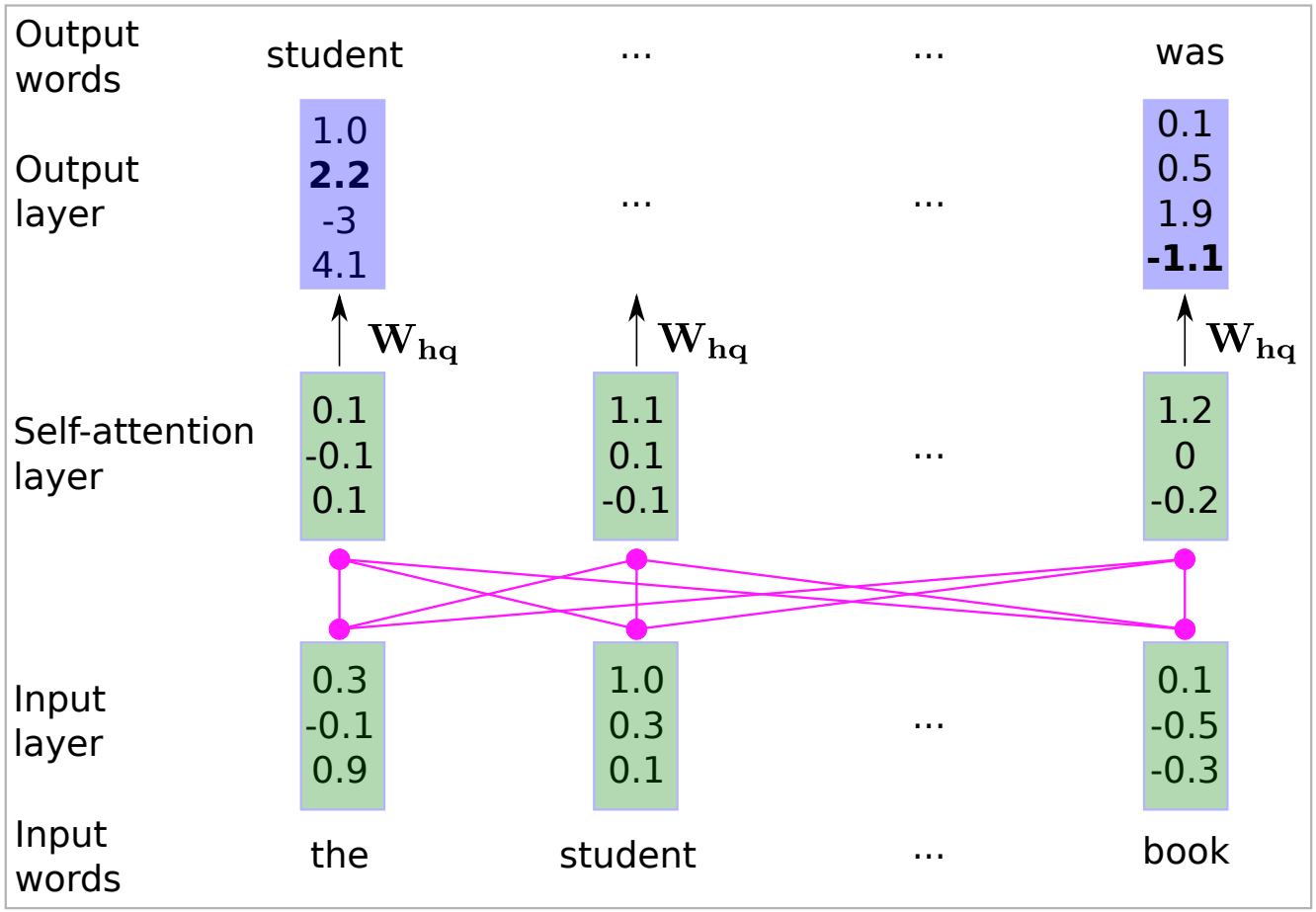
Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Each self-attention layer transforms a sequence to another sequence.

Self-attention: Learning how to re-weigh words with all other words.

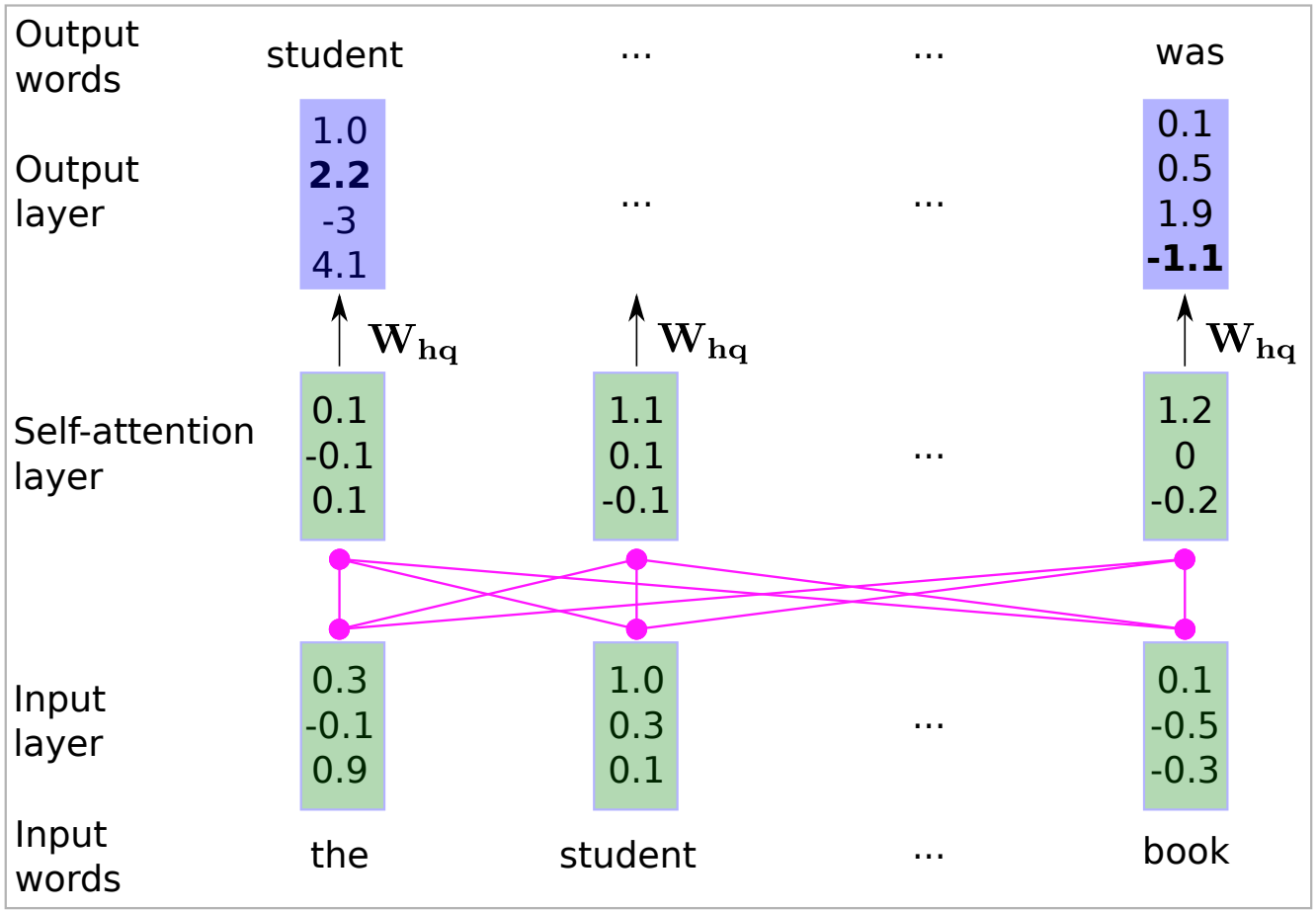
Q: Why is self-attention parallelizable?
A: No recurrence; each vector can be computed independently (ie: in parallel).

Self-attention: Learn to re-weight a word with all others



Note: purple lines do **not** represent individual weights; What they are will come now.

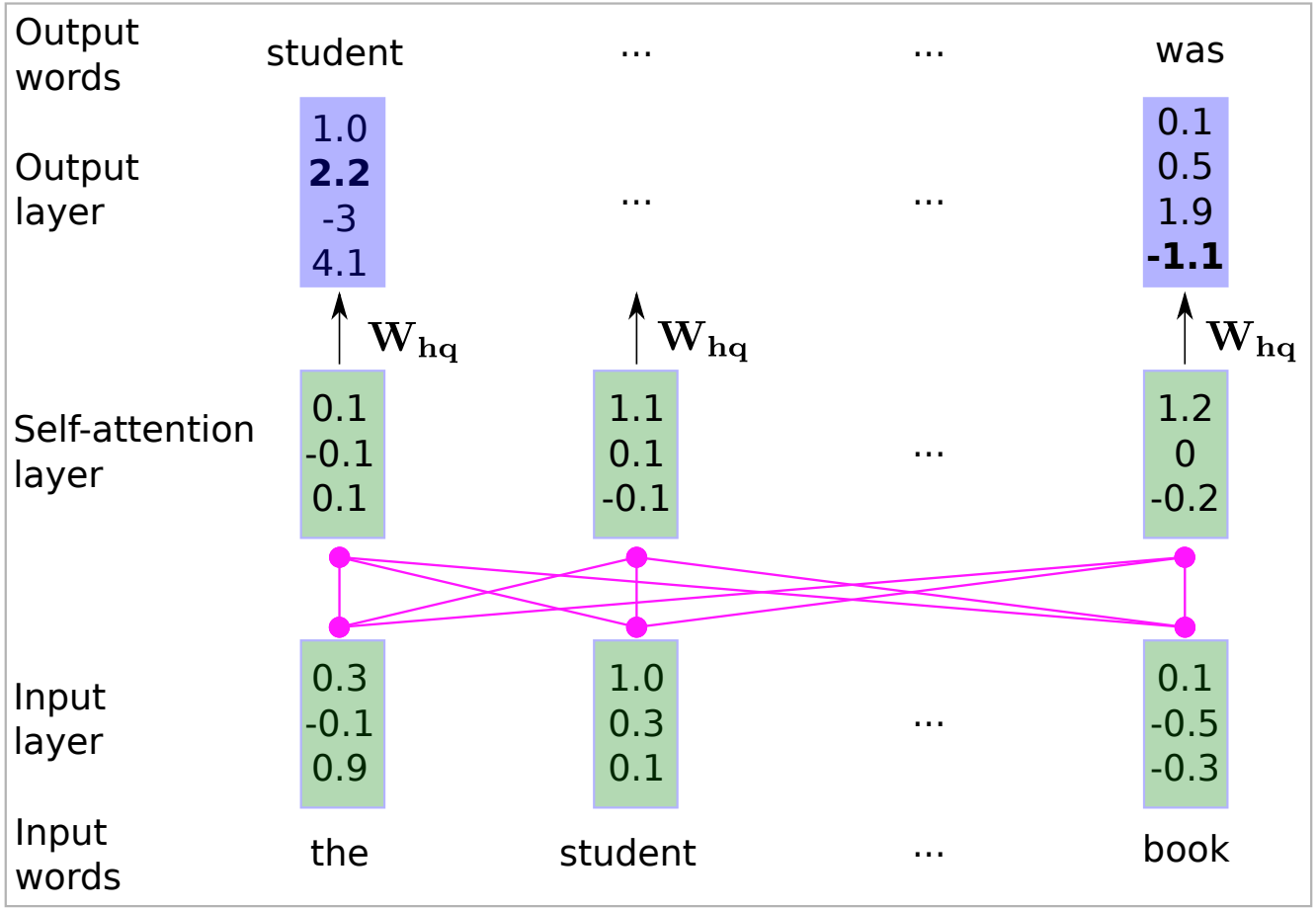
Self-attention: Learn to re-weight a word with all others



Note: purple lines do **not** represent individual weights; What they are will come now.

Make how much word A can re-weight a word B depend on their similarity

Self-attention: Learn to re-weight a word with all others

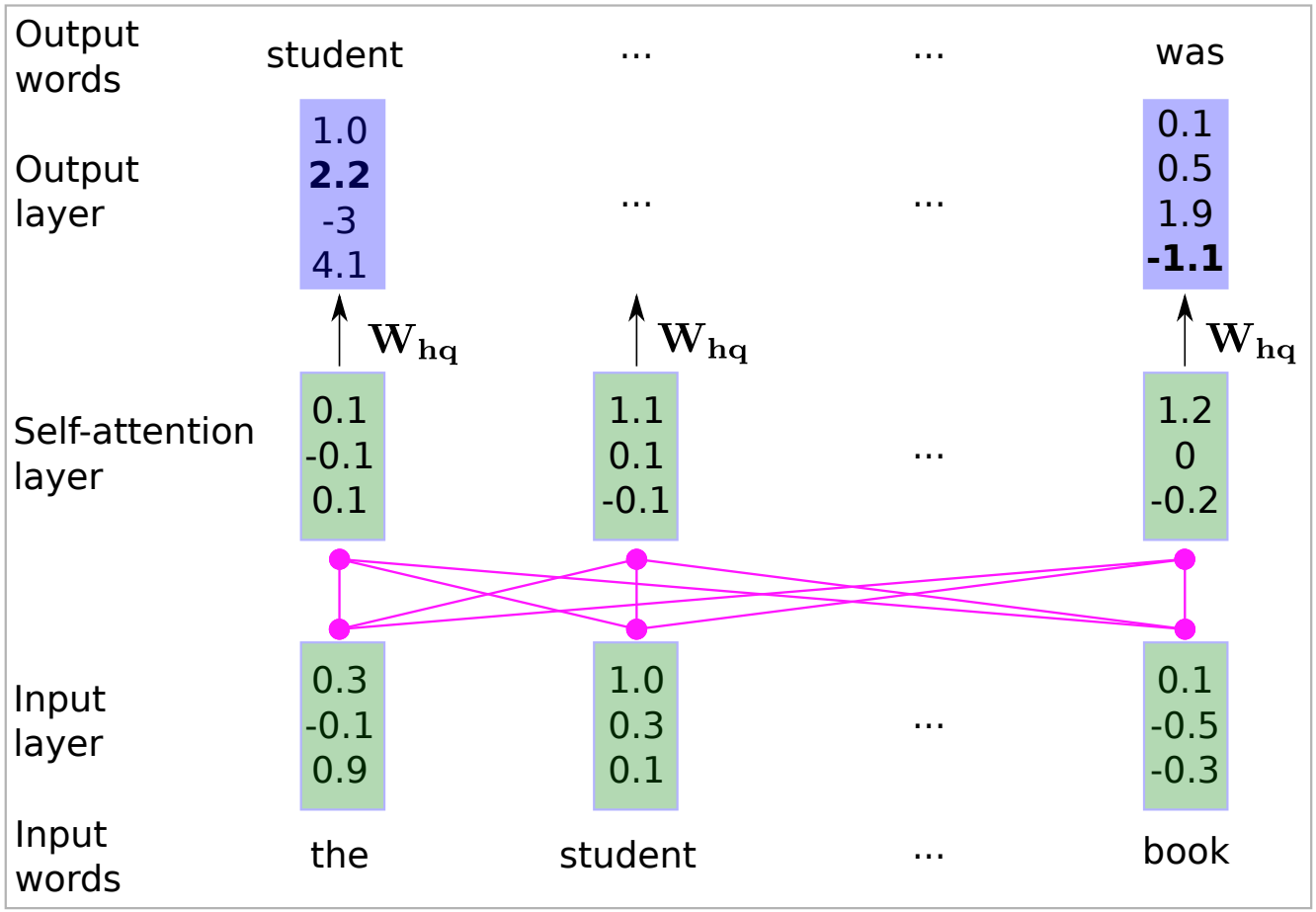


Note: purple lines do **not** represent individual weights; What they are will come now.

Make how much word A can re-weight a word B depend on their similarity

Q: On what would you compute the similarity between word A and word B?

Self-attention: Learn to re-weight a word with all others



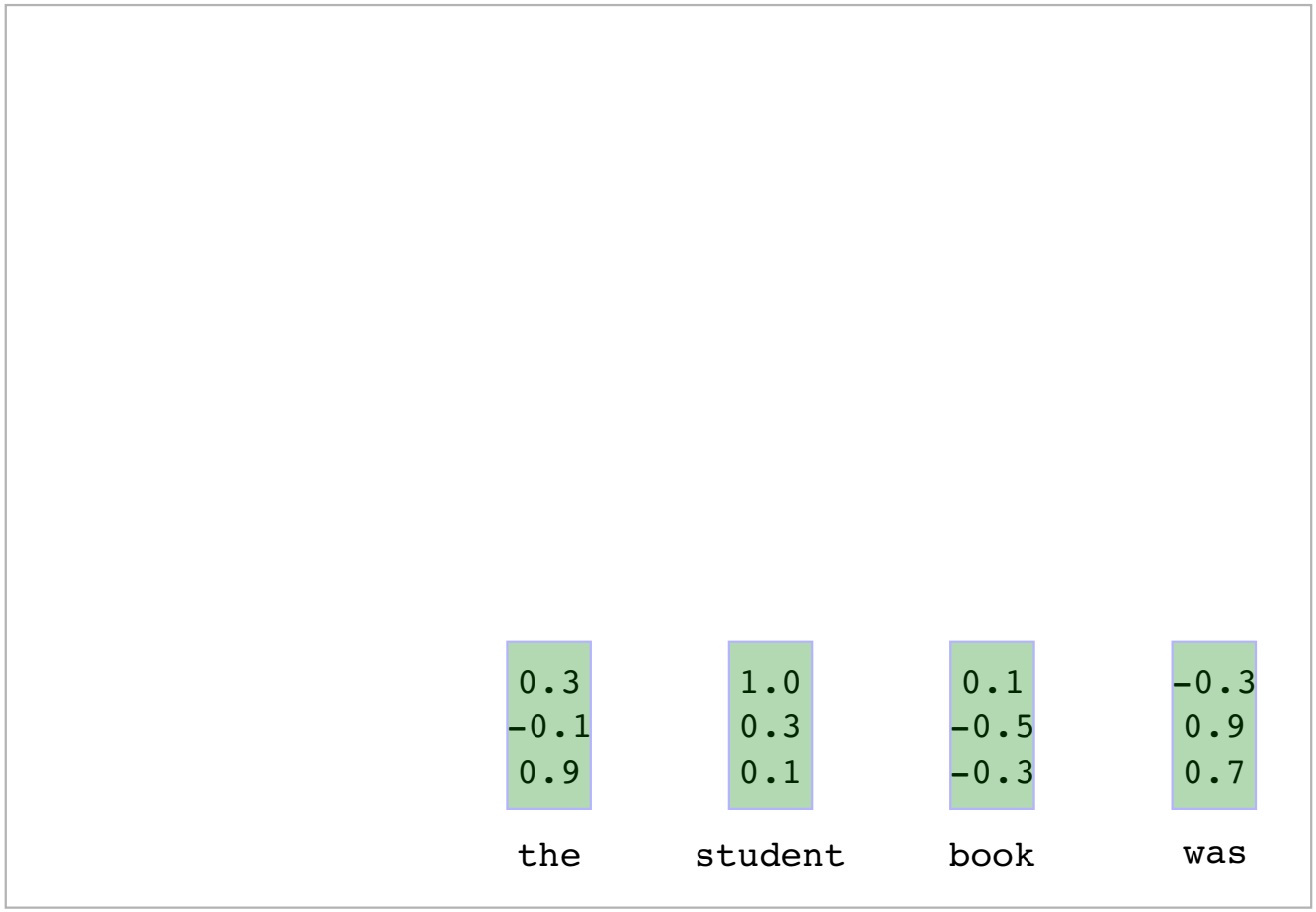
Note: purple lines do **not** represent individual weights; What they are will come now.

Make how much word A can re-weight a word B depend on their similarity

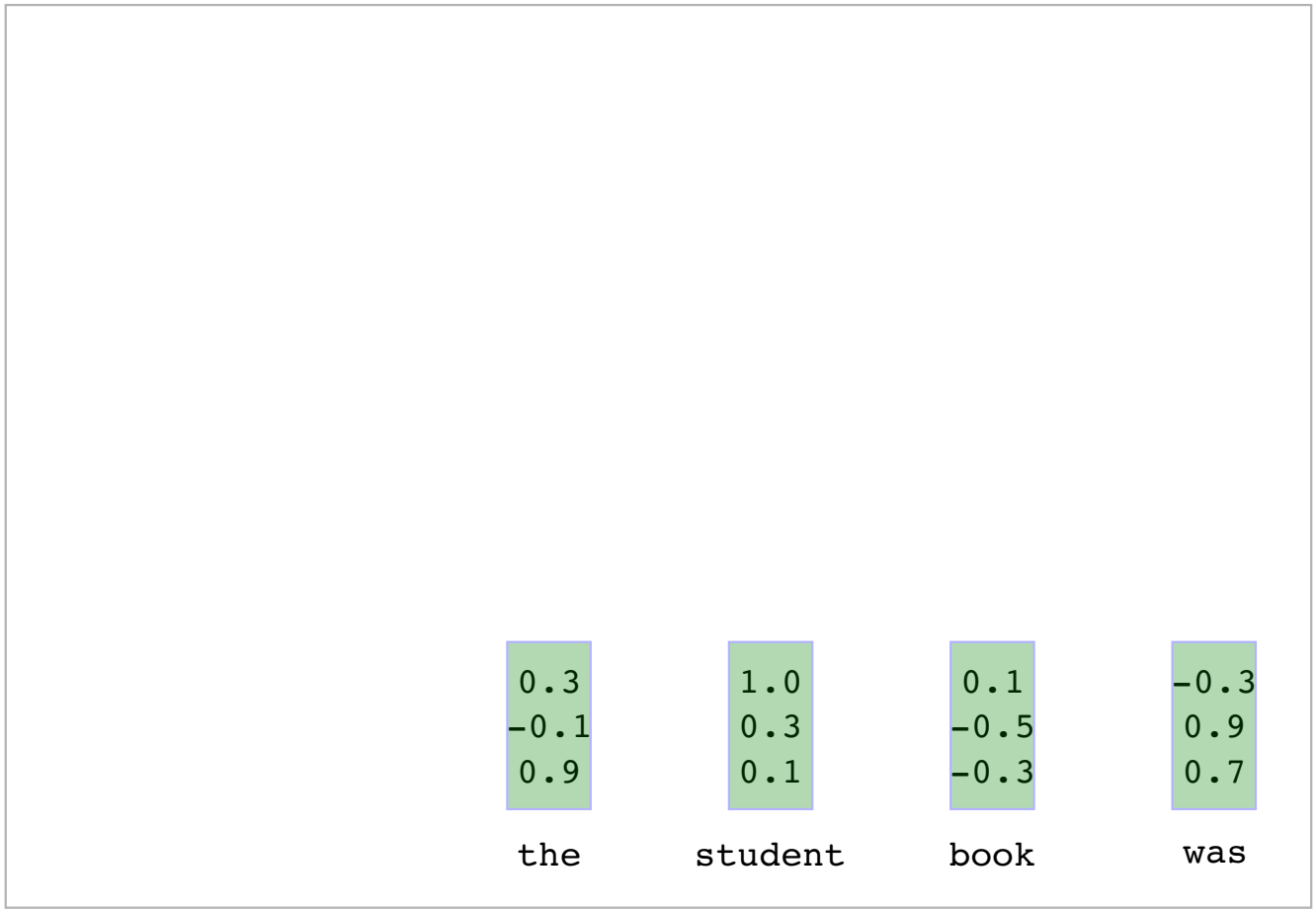
Q: On what would you compute the similarity between word A and word B?

A: Use similarities between their word embeddings.

Self-attention: re-weigh a word with all others

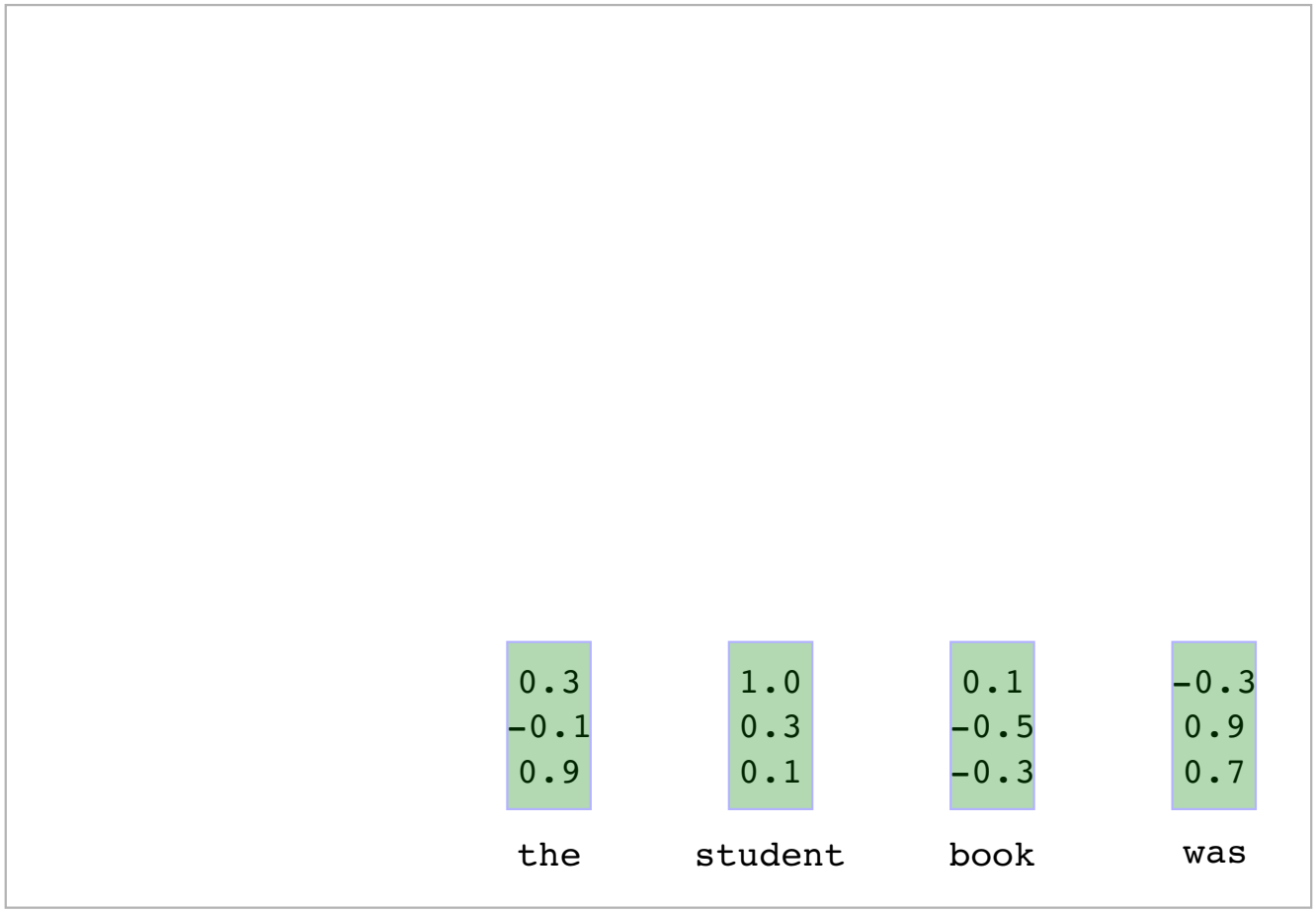


Self-attention: re-weigh a word with all others



Q: How many similarities?

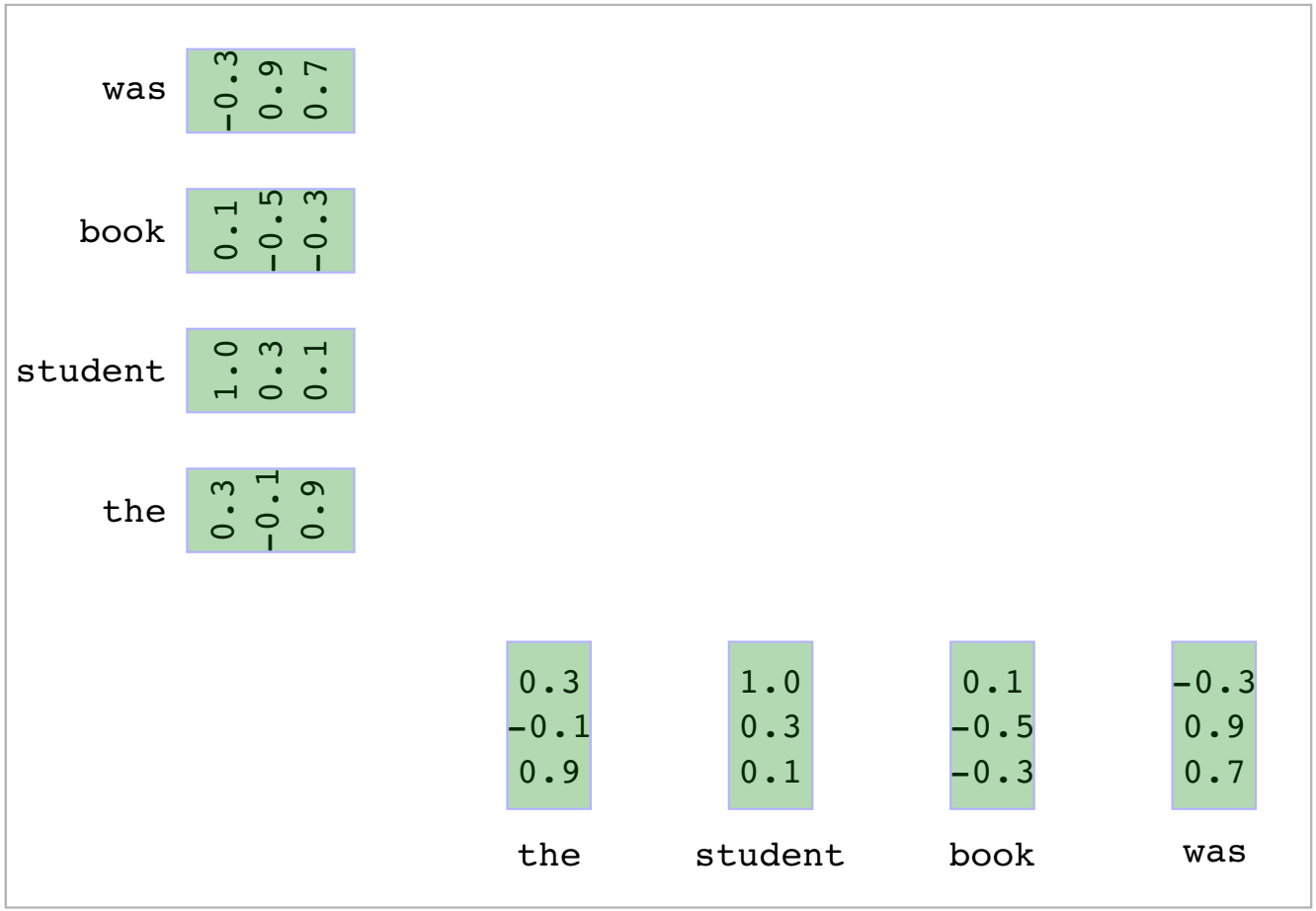
Self-attention: re-weigh a word with all others



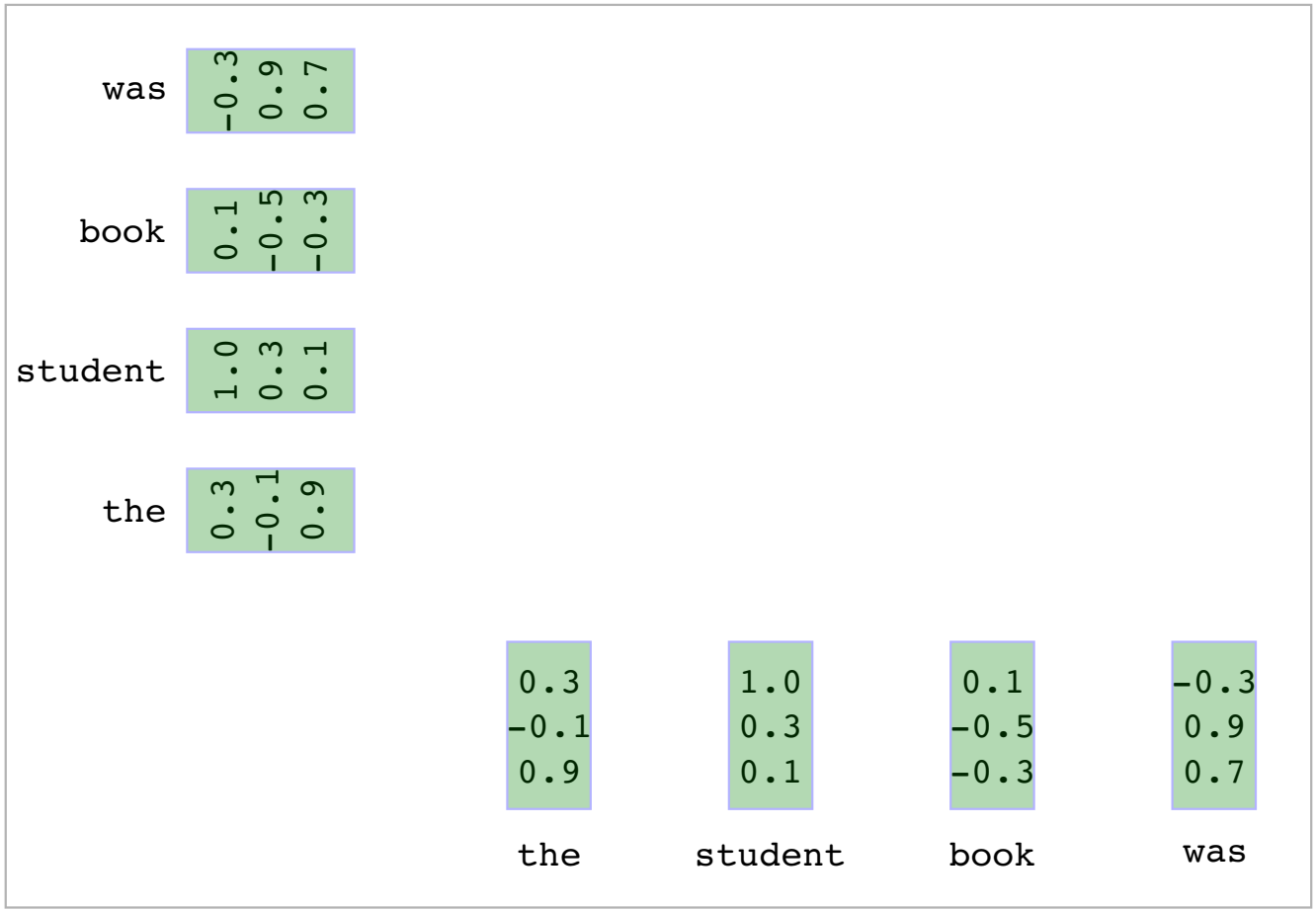
Q: How many similarities?

A: All words to all words; for n words, there are n^2 similarities.

Self-attention: re-weigh a word with all others

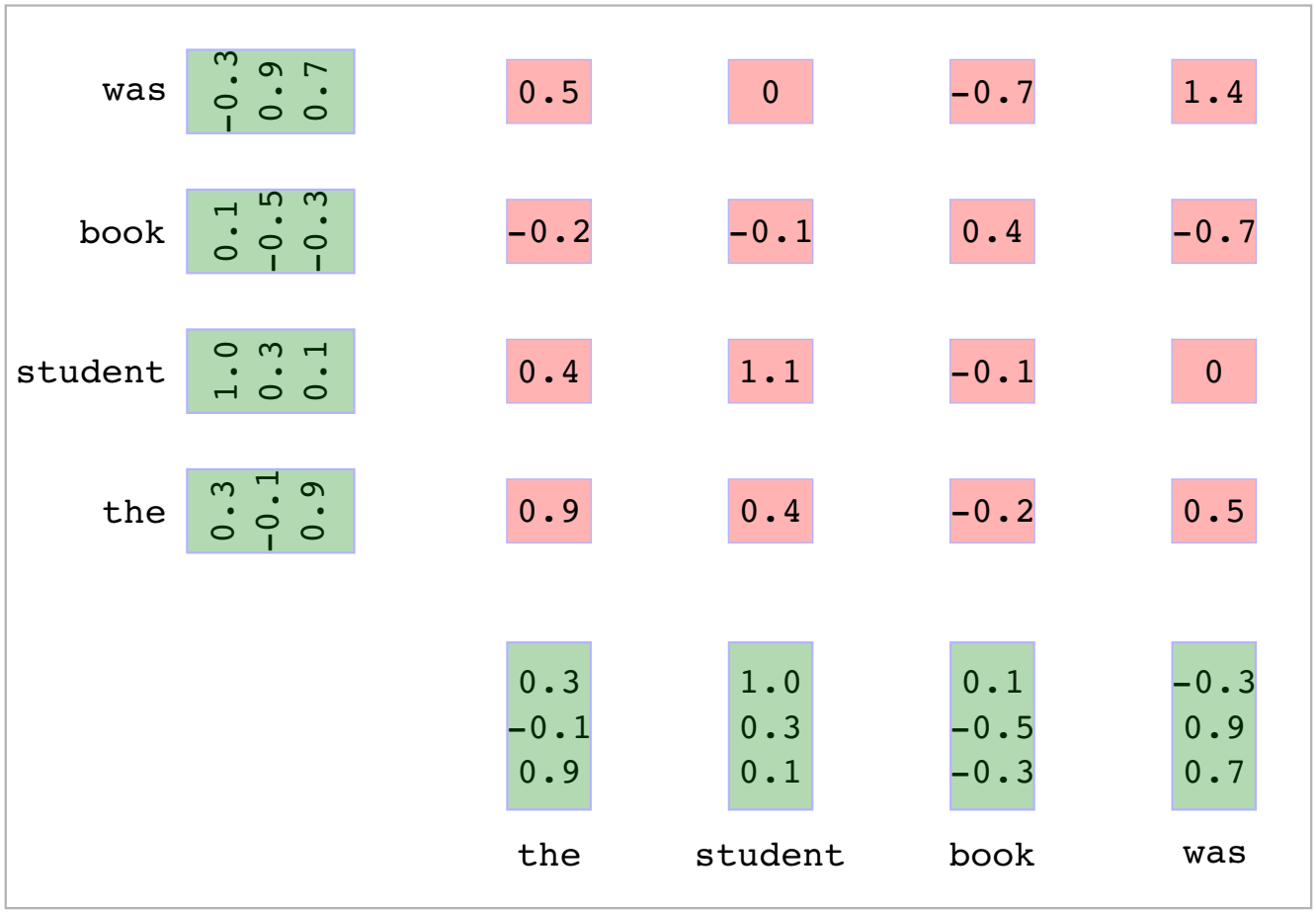


Self-attention: re-weigh a word with all others

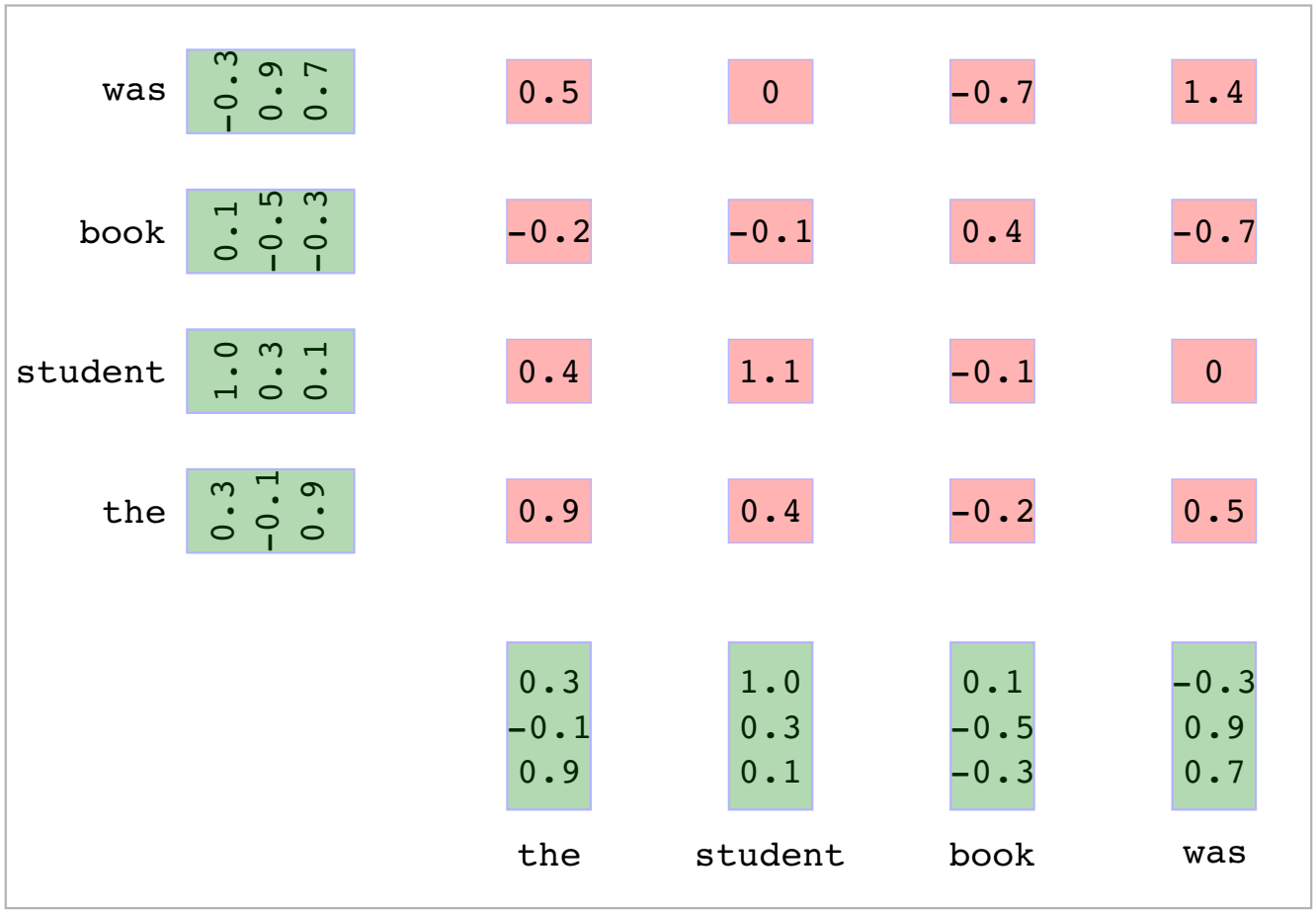


Q: Lets use the dot product $< \cdot, \cdot >$. Can you compute $< \mathbf{x}_{\text{the}}, \mathbf{x}_{\text{the}} >$?

Self-attention: re-weigh a word with all others

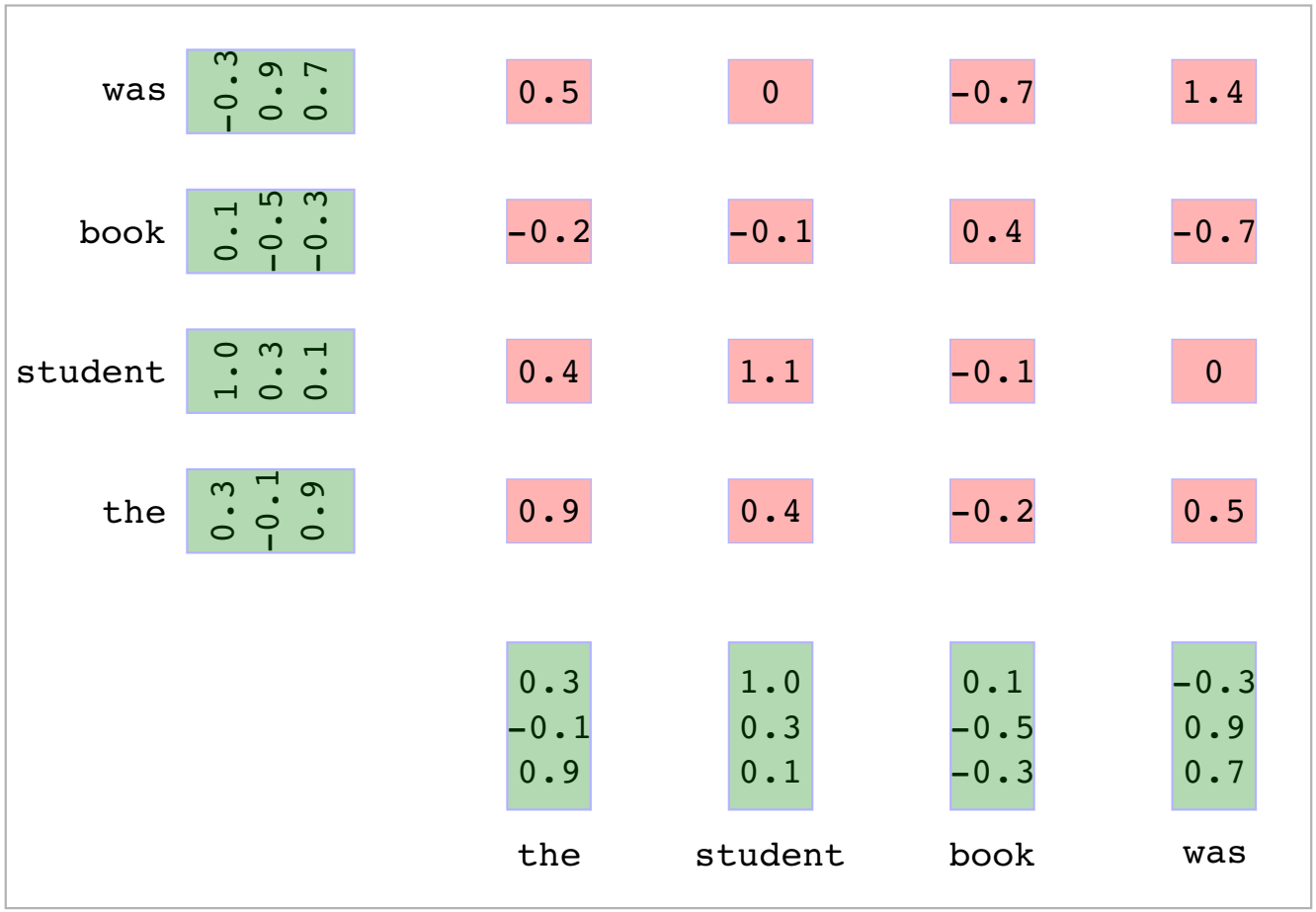


Self-attention: re-weigh a word with all others



Weighting dot products $\mathbf{x}_i^\intercal \mathbf{x}_j$ are unbounded in $(-\infty, \infty)$

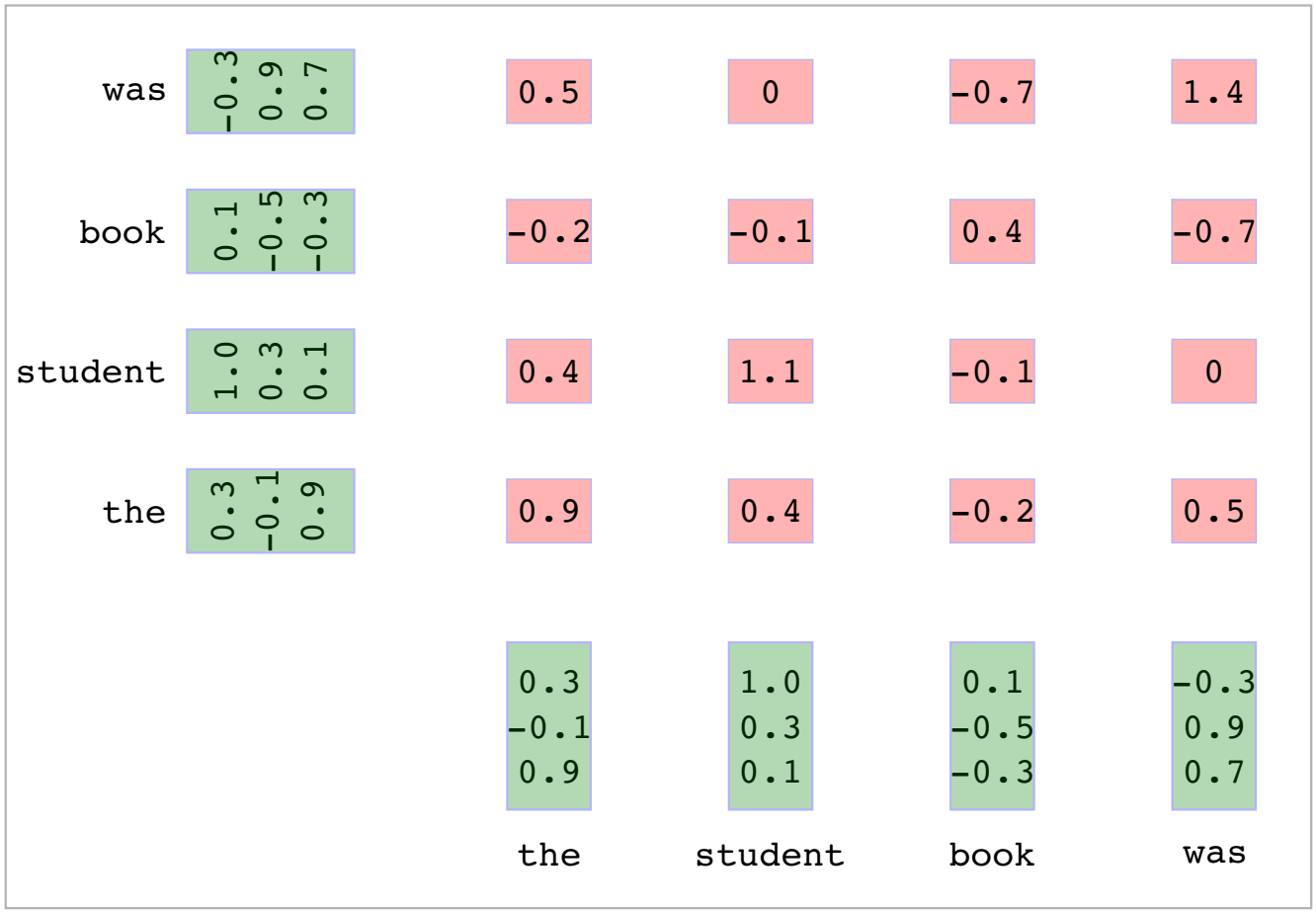
Self-attention: re-weigh a word with all others



Weighting dot products $\mathbf{x}_i^\intercal \mathbf{x}_j$ are unbounded in $(-\infty, \infty)$

Q: how to normalize that the similaries sum to 1 (and are not negative) ?

Self-attention: re-weigh a word with all others

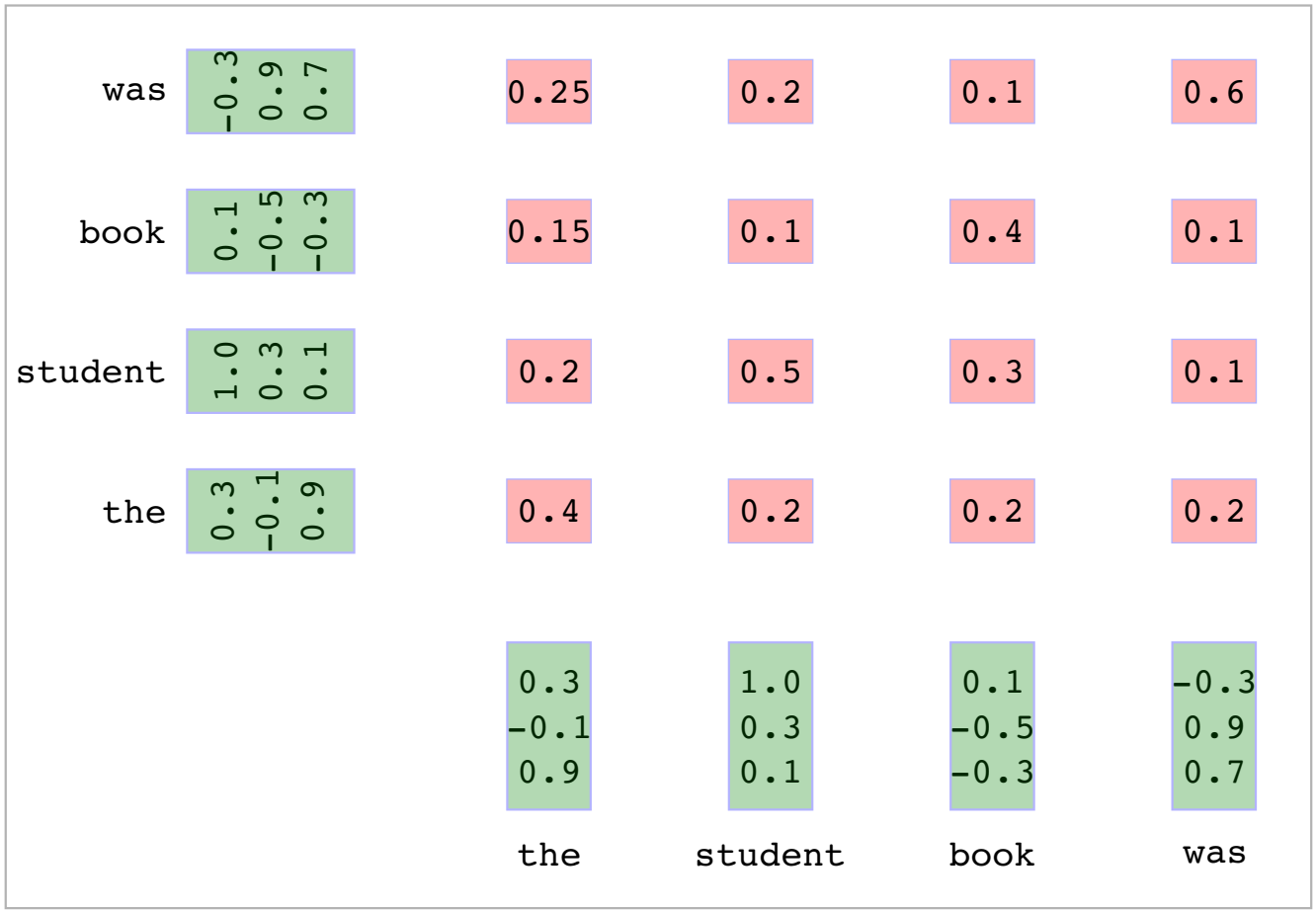


Weighting dot products $\mathbf{x}_i^\intercal \mathbf{x}_j$ are unbounded in $(-\infty, \infty)$

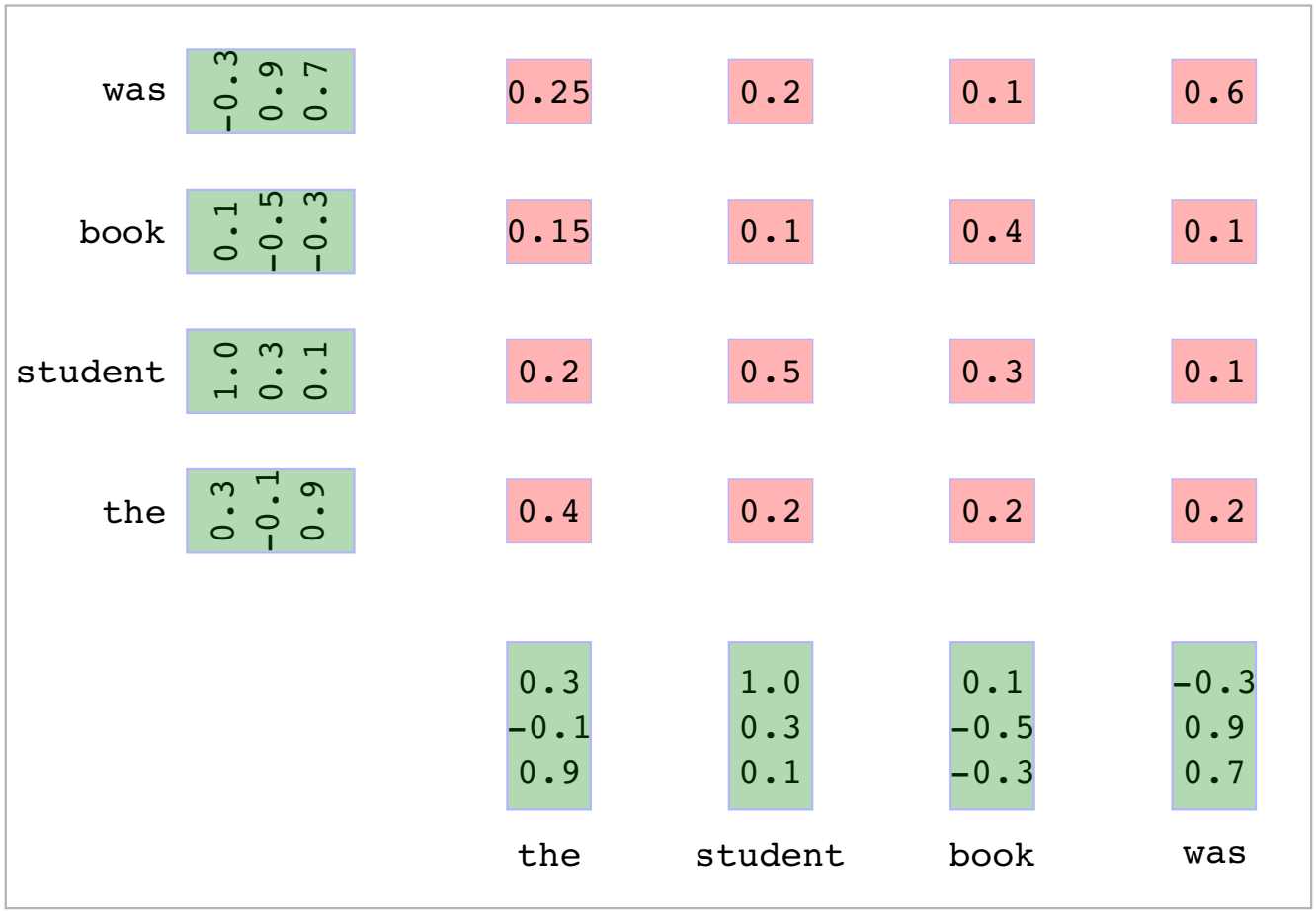
Q: how to normalize that the similaries sum to 1 (and are not negative) ?

A: Use the soft-max $w_{ij} = \frac{\exp \mathbf{x}_i^\intercal \mathbf{x}_j}{\sum_j \exp \mathbf{x}_i^\intercal \mathbf{x}_j}$, so $\sum_j w_{ij} = 1$

Self-attention: re-weigh a word with all others

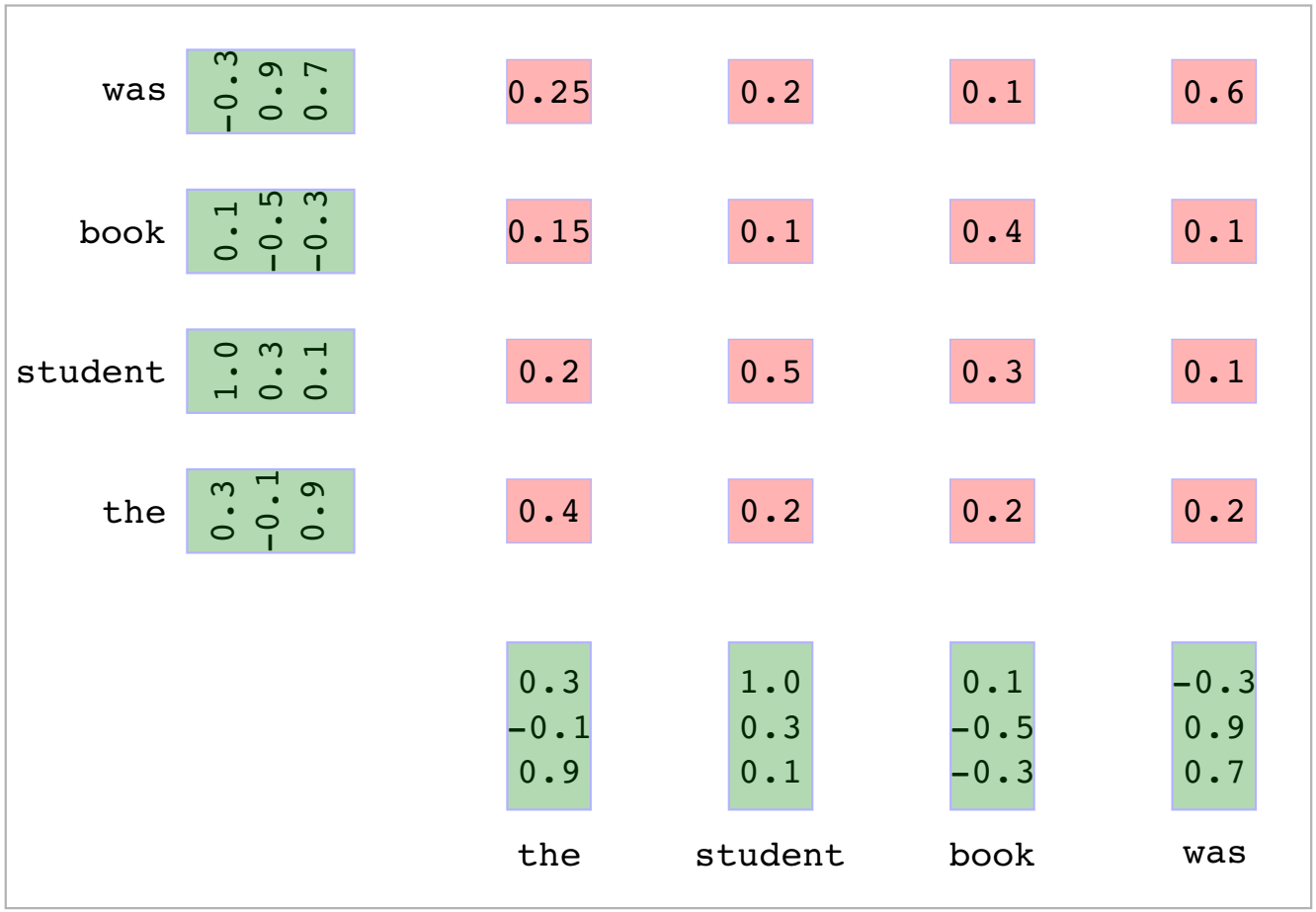


Self-attention: re-weigh a word with all others



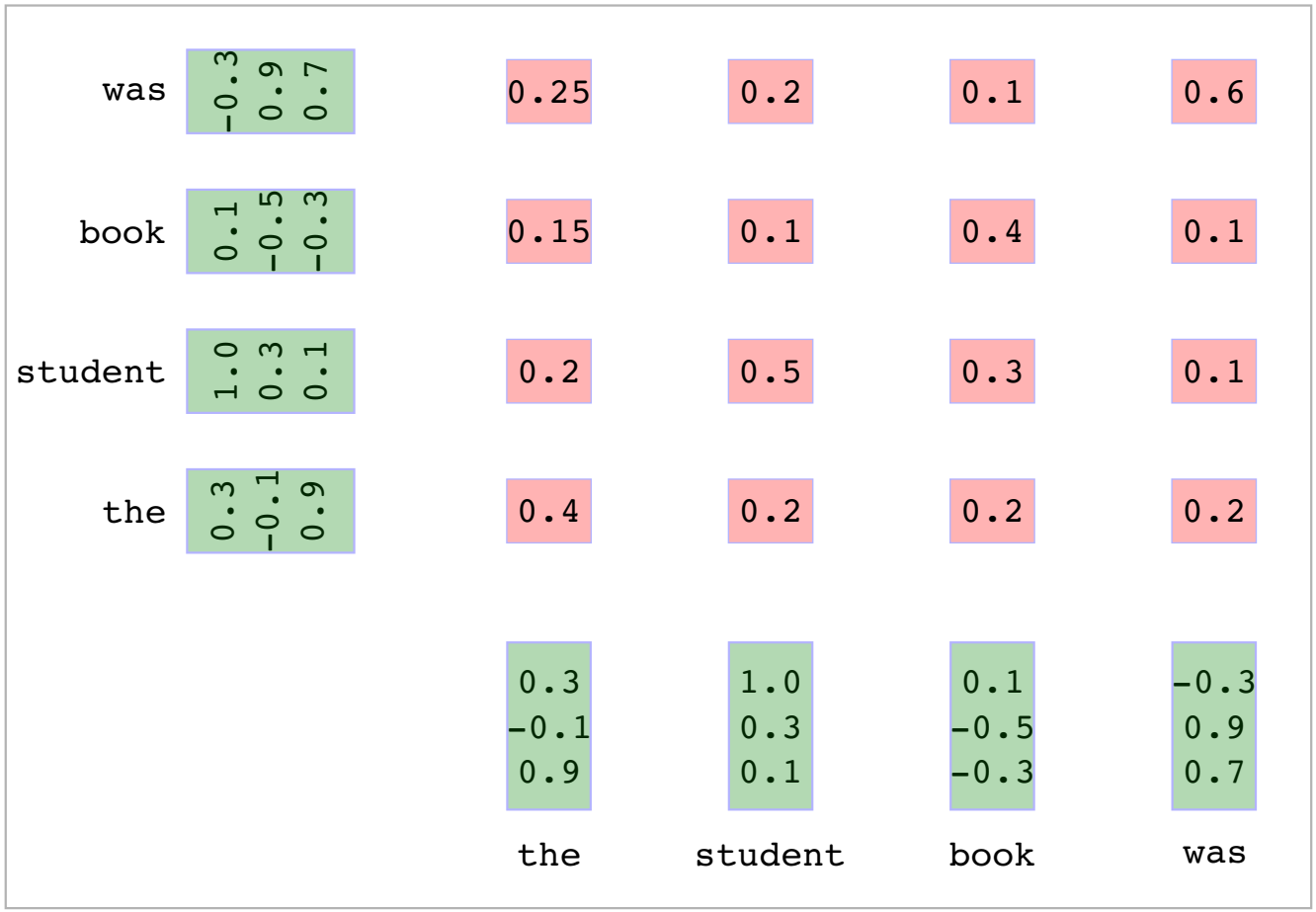
Q: Wait, but the rows do not sum to 1?

Self-attention: re-weigh a word with all others



Q: Wait, but the rows do not sum to 1? A: Softmax here was over the columns.

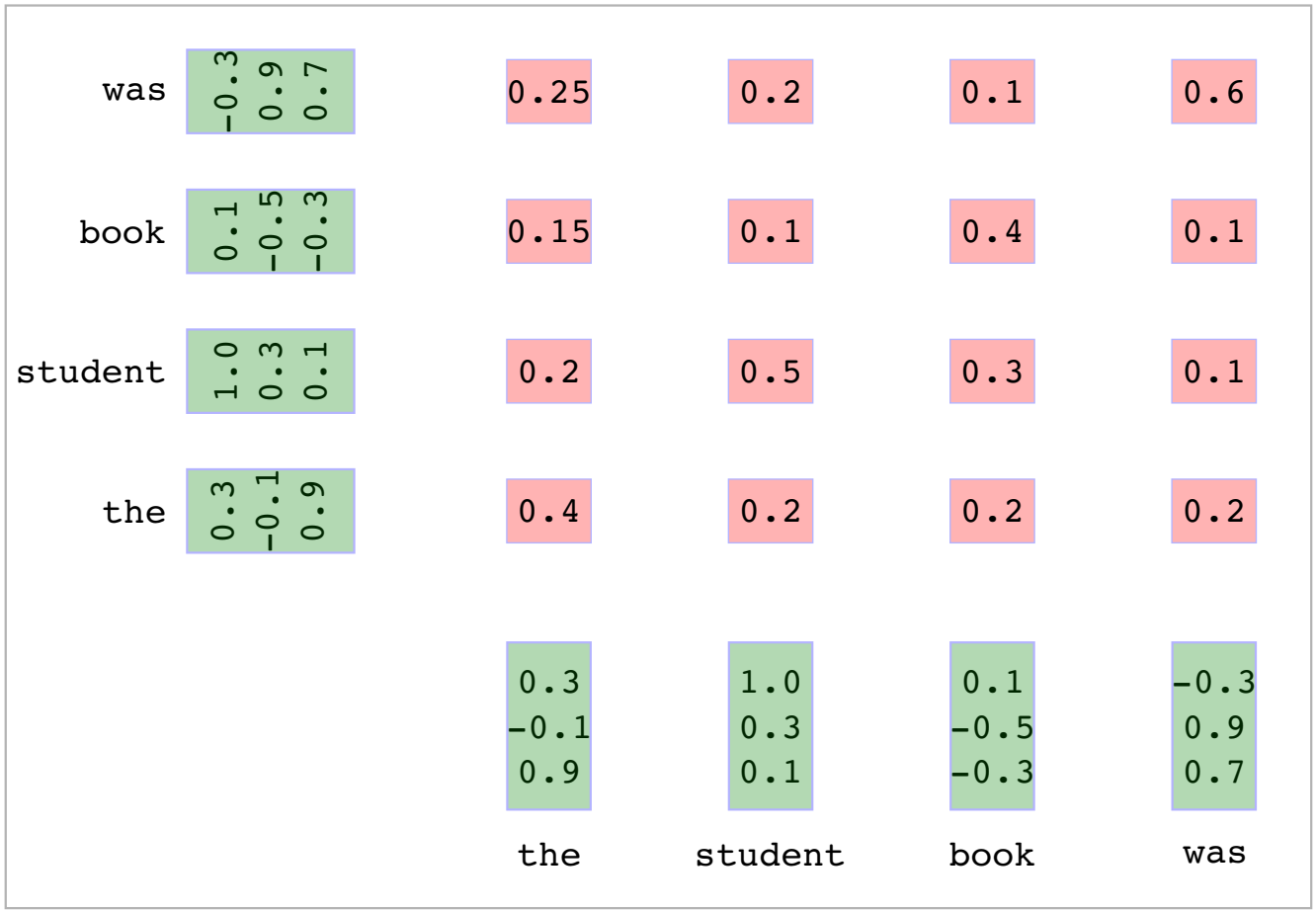
Self-attention: re-weigh a word with all others



Q: Wait, but the rows do not sum to 1? A: Softmax here was over the columns.

Q: How to compute new $\mathbf{x}_{\text{book}}$?

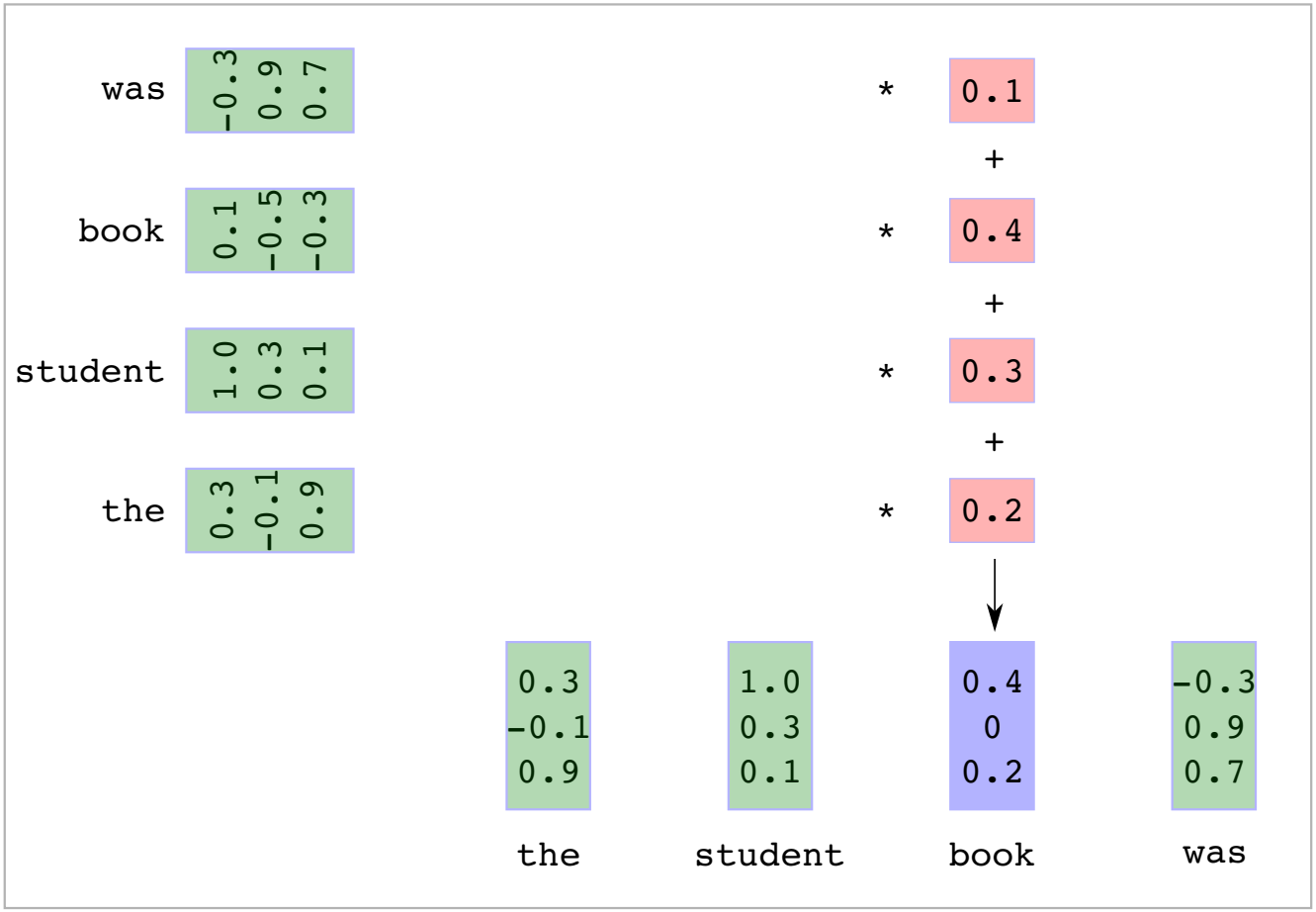
Self-attention: re-weigh a word with all others



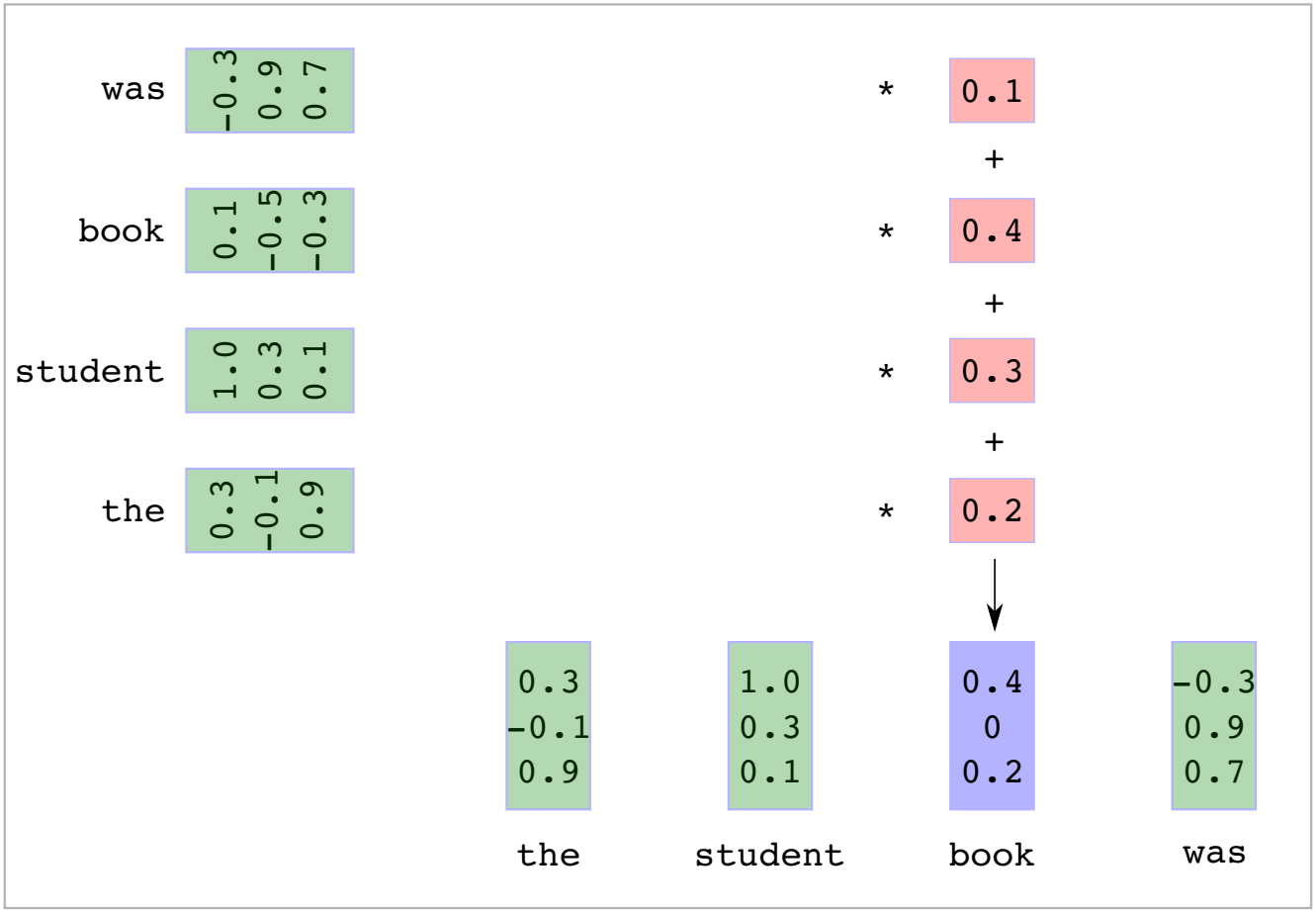
Q: Wait, but the rows do not sum to 1? A: Softmax here was over the columns.

Q: How to compute new $\mathbf{x}_{\text{book}}$? A: Weighted sum of word vectors.

Self-attention: re-weigh a word with all others



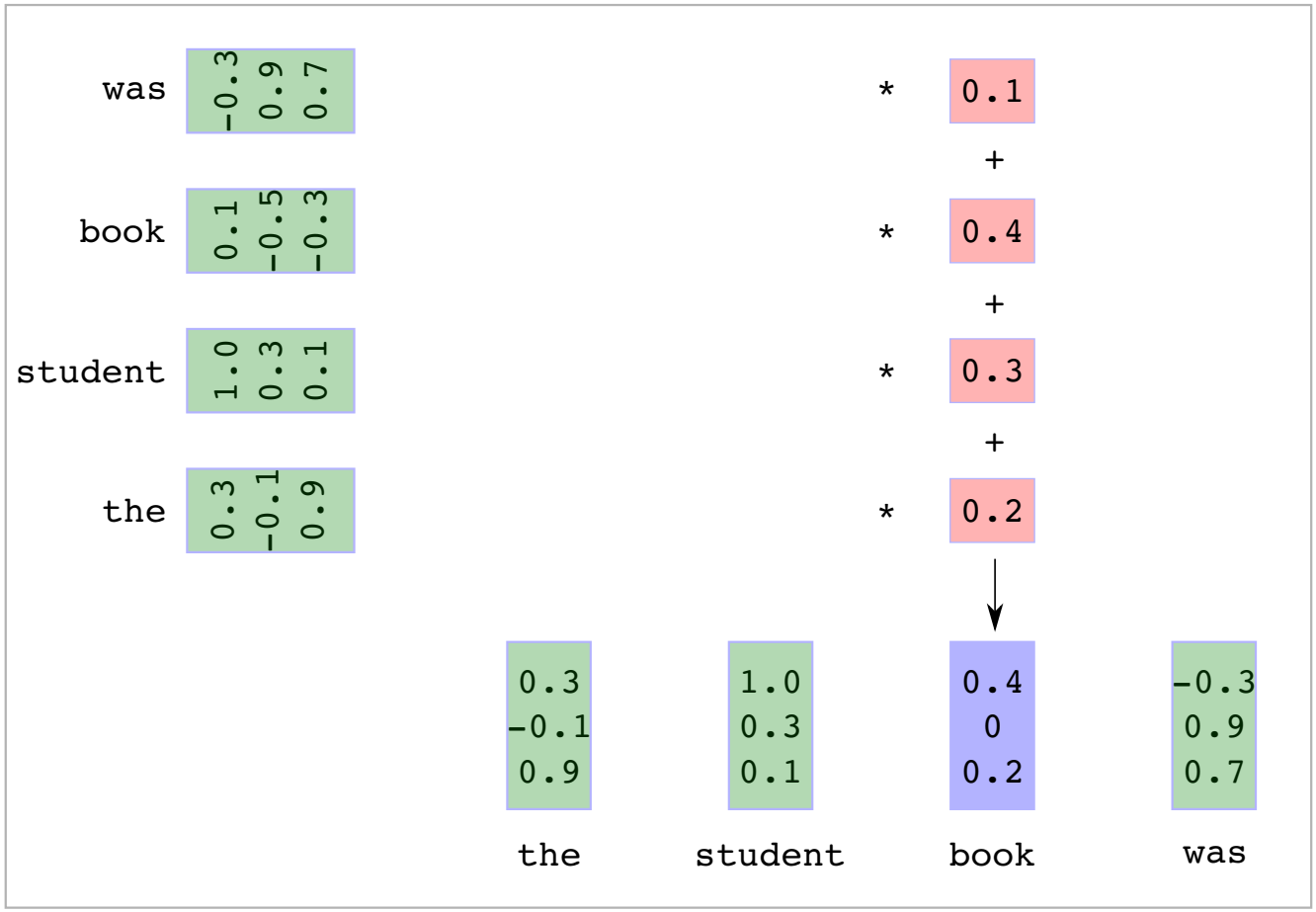
Self-attention: re-weigh a word with all others



Questions ?

Questions?

Self-attention: Roles of word vectors

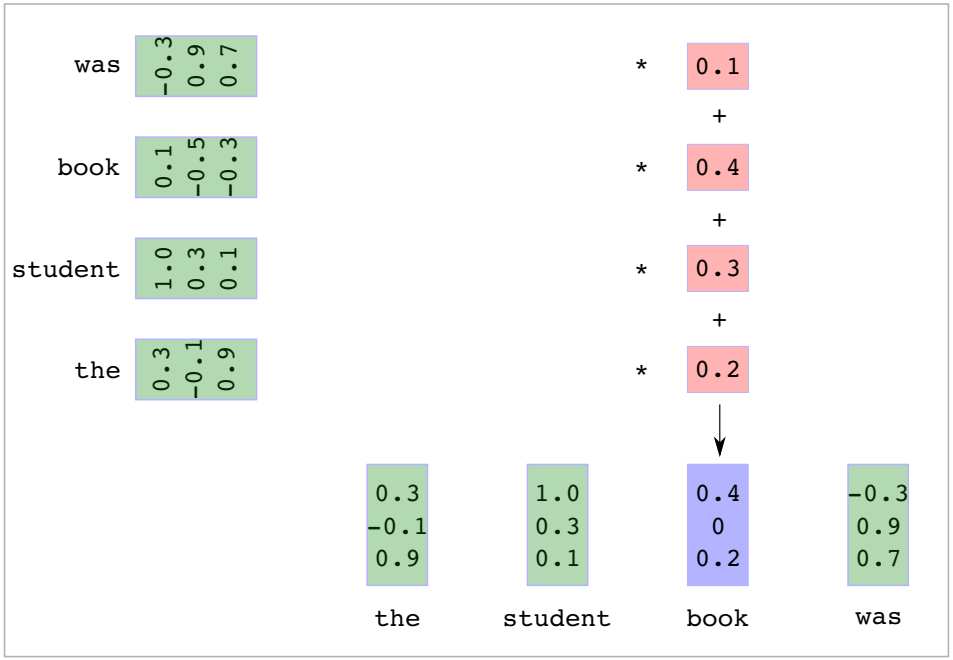


Word vectors are used in two roles:

1. Computing a weight for how much a word A influences a word B;
2. Weighted summing of all word vectors to a new vector.

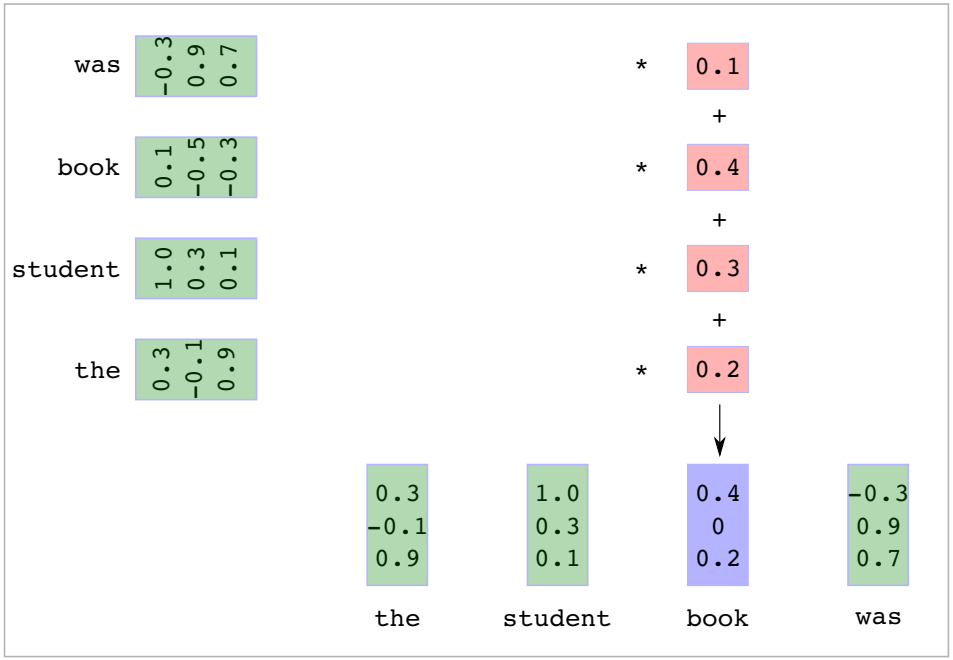
Self-attention: Roles of word vectors

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.



Self-attention: Roles of word vectors

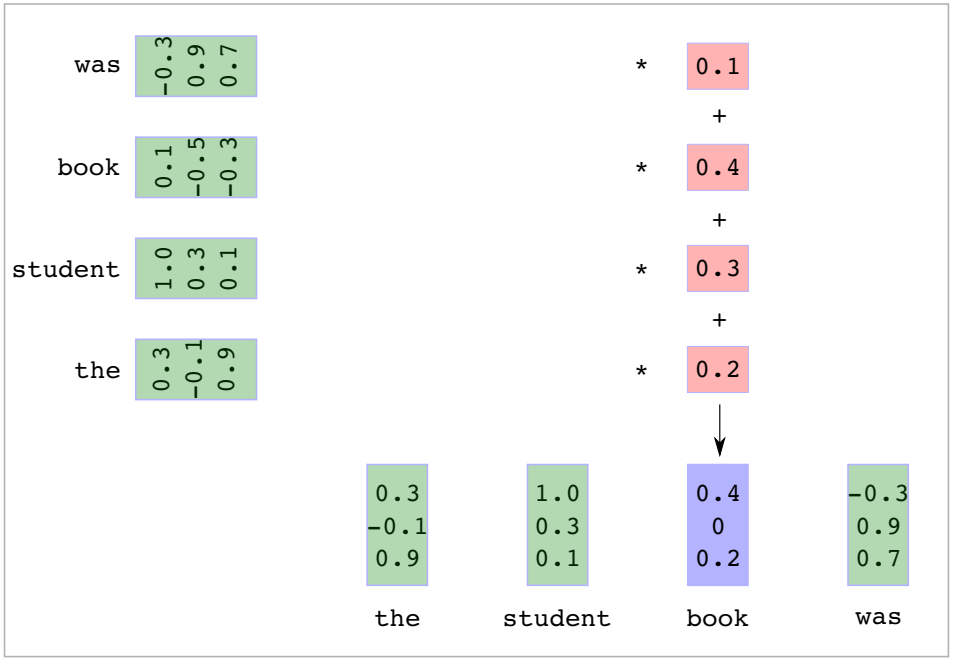
A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.



Q: Example of role 1? (which words would you expect to influence each other?):

Self-attention: Roles of word vectors

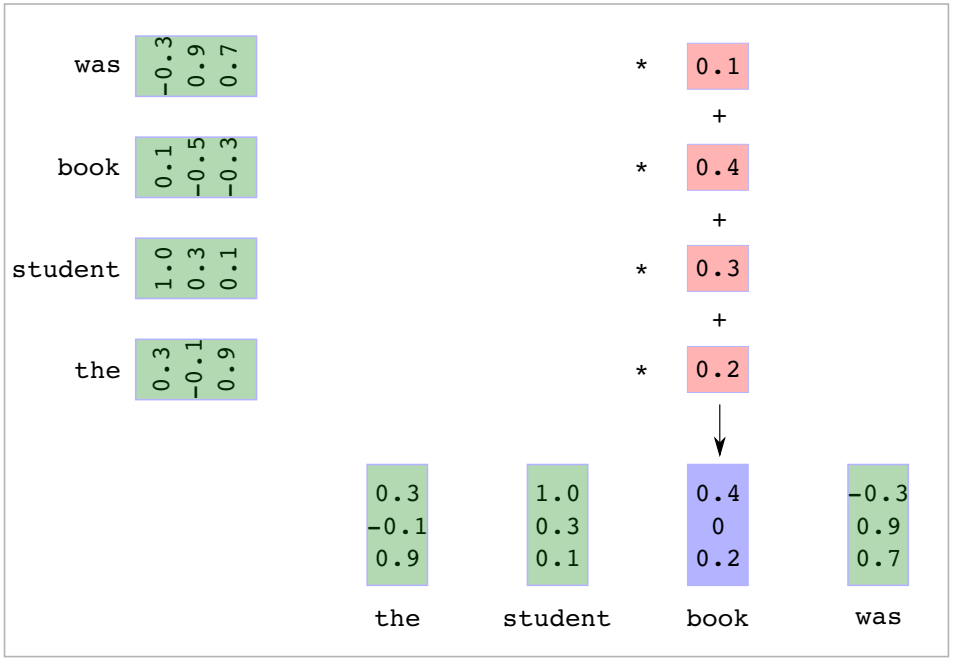
A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.



Q: Example of role 1? (which words would you expect to influence each other?):
A: *(the, student)*, *(the, book)*, *(student, was)*; not so much: *(the, was)*

Self-attention: Roles of word vectors

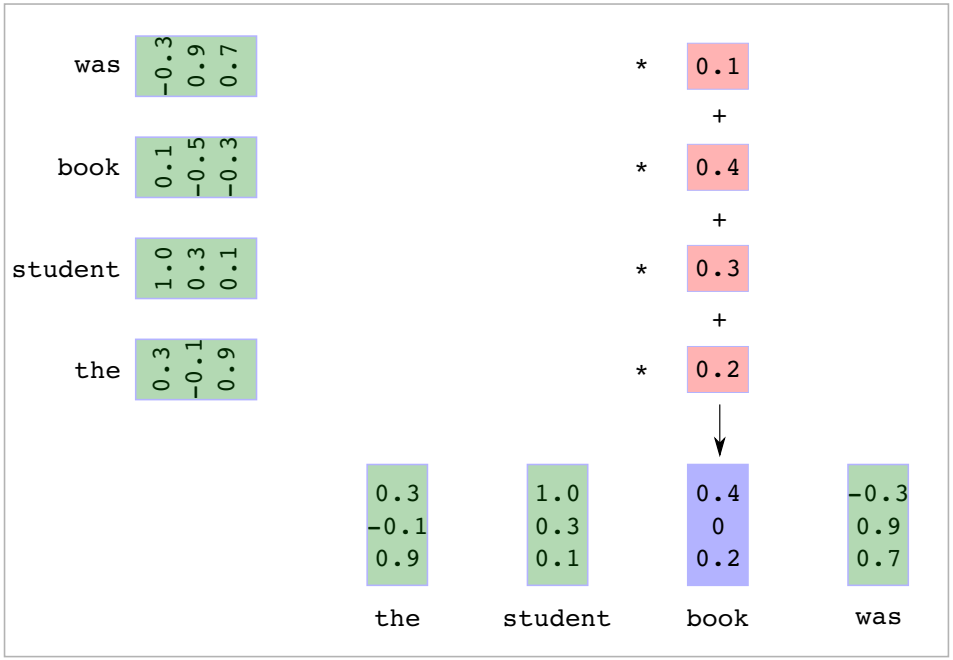
A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.



Q: Example of role 1? (which words would you expect to influence each other?):
A: *(the, student)*, *(the, book)*, *(student, was)*; not so much: *(the, was)*
Q: What is the "type" of words that influence each other?

Self-attention: Roles of word vectors

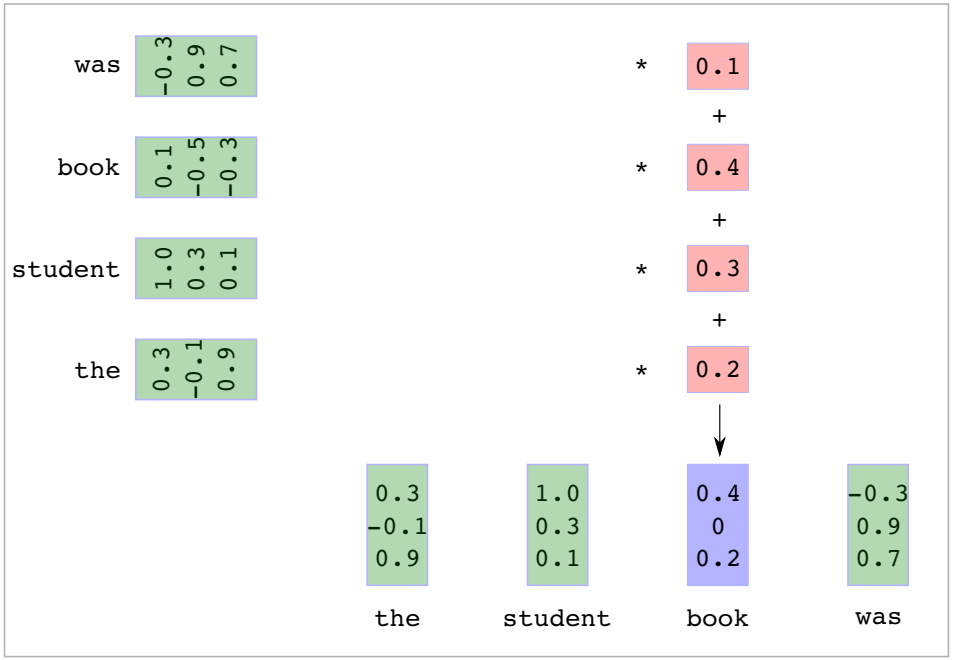
A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.



Q: Example of role 1? (which words would you expect to influence each other?):
A: *(the, student), (the, book), (student, was)*; not so much: *(the, was)*
Q: What is the "type" of words that influence each other?
A: (Noun, Verb), (Article, Noun); ...

Self-attention: Roles of word vectors

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.

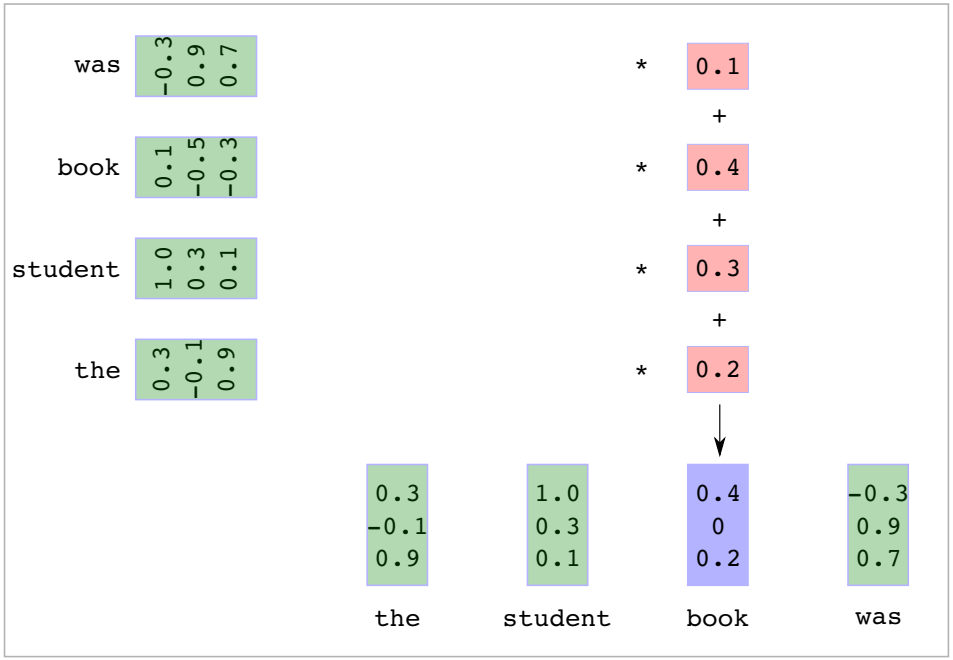


Q: Example of role 1? (which words would you expect to influence each other?):
A: *(the, student), (the, book), (student, was)*; not so much: *(the, was)*
Q: What is the "type" of words that influence each other?
A: (Noun, Verb), (Article, Noun); ...

Q: Example of role 2? (What to take from '*student[s]*' to determine '*was[were]*'?)

Self-attention: Roles of word vectors

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.



Q: Example of role 1? (which words would you expect to influence each other?):
A: *(the, student), (the, book), (student, was)*; not so much: *(the, was)*
Q: What is the "type" of words that influence each other?
A: (Noun, Verb), (Article, Noun); ...

Q: Example of role 2? (What to take from '*student[s]*' to determine '*was[were]*'?)
A: The concept of "singular/plurality".

Self attention: Queries, keys, values

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See: `http://www.peterbloem.nl/blog/transformers`

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.

Self attention: Queries, keys, values

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See: `http://www.peterbloem.nl/blog/transformers`

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.

Q: Is the required 'information' the same for role 1 and role 2?

Self attention: Queries, keys, values

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See: `http://www.peterbloem.nl/blog/transformers`

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.

Q: Is the required 'information' the same for role 1 and role 2?

A: No; different information per role:

Role 1: 'type' of word to influence (eg: noun, verb);

Role 2: what to 'give' from a word (eg: 'plurality').

Self attention: Queries, keys, values

D2L book: https://d2l.ai/chapter_attention-mechanisms/index.html
See: http://www.peterbloem.nl/blog/transformers

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.

Q: Is the required 'information' the same for role 1 and role 2?
A: No; different information per role:

Role 1: 'type' of word to influence (eg: noun, verb);
Role 2: what to 'give' from a word (eg: 'plurality').

2 roles; each word vector is used in 3 ways:

Self attention: Queries, keys, values

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See: `http://www.peterbloem.nl/blog/transformers`

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.

Q: Is the required 'information' the same for role 1 and role 2?
A: No; different information per role:

Role 1: 'type' of word to influence (eg: noun, verb);
Role 2: what to 'give' from a word (eg: 'plurality').

2 roles; each word vector is used in 3 ways:

- Role 1: Query: What 'type' am I ('noun-ness')?;

Self attention: Queries, keys, values

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See: `http://www.peterbloem.nl/blog/transformers`

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.

Q: Is the required 'information' the same for role 1 and role 2?
A: No; different information per role:

Role 1: 'type' of word to influence (eg: noun, verb);
Role 2: what to 'give' from a word (eg: 'plurality').

2 roles; each word vector is used in 3 ways:

- Role 1: Query: What 'type' am I ('noun-ness')?;
- Role 1: Key: How should my 'type' influence others ('verb-ness')?

Self attention: Queries, keys, values

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See: `http://www.peterbloem.nl/blog/transformers`

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.

Q: Is the required 'information' the same for role 1 and role 2?
A: No; different information per role:

Role 1: 'type' of word to influence (eg: noun, verb);
Role 2: what to 'give' from a word (eg: 'plurality').

2 roles; each word vector is used in 3 ways:

- Role 1: Query: What 'type' am I ('noun-ness')?;
- Role 1: Key: How should my 'type' influence others ('verb-ness')?
- Role 2: Value: What should I 'give' ('plurality')?

Self attention: Queries, keys, values

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See: `http://www.peterbloem.nl/blog/transformers`

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.

Q: Is the required 'information' the same for role 1 and role 2?
A: No; different information per role:

Role 1: 'type' of word to influence (eg: noun, verb);
Role 2: what to 'give' from a word (eg: 'plurality').

2 roles; each word vector is used in 3 ways:

- Role 1: Query: What 'type' am I ('noun-ness')?;
- Role 1: Key: How should my 'type' influence others ('verb-ness')?
- Role 2: Value: What should I 'give' ('plurality')?

Self attention: Queries, keys, values

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See: `http://www.peterbloem.nl/blog/transformers`

A word vector is used in 2 roles: 1. How A influences B; 2. Sum to a new vector.

Q: Is the required 'information' the same for role 1 and role 2?
A: No; different information per role:

Role 1: 'type' of word to influence (eg: noun, verb);
Role 2: what to 'give' from a word (eg: 'plurality').

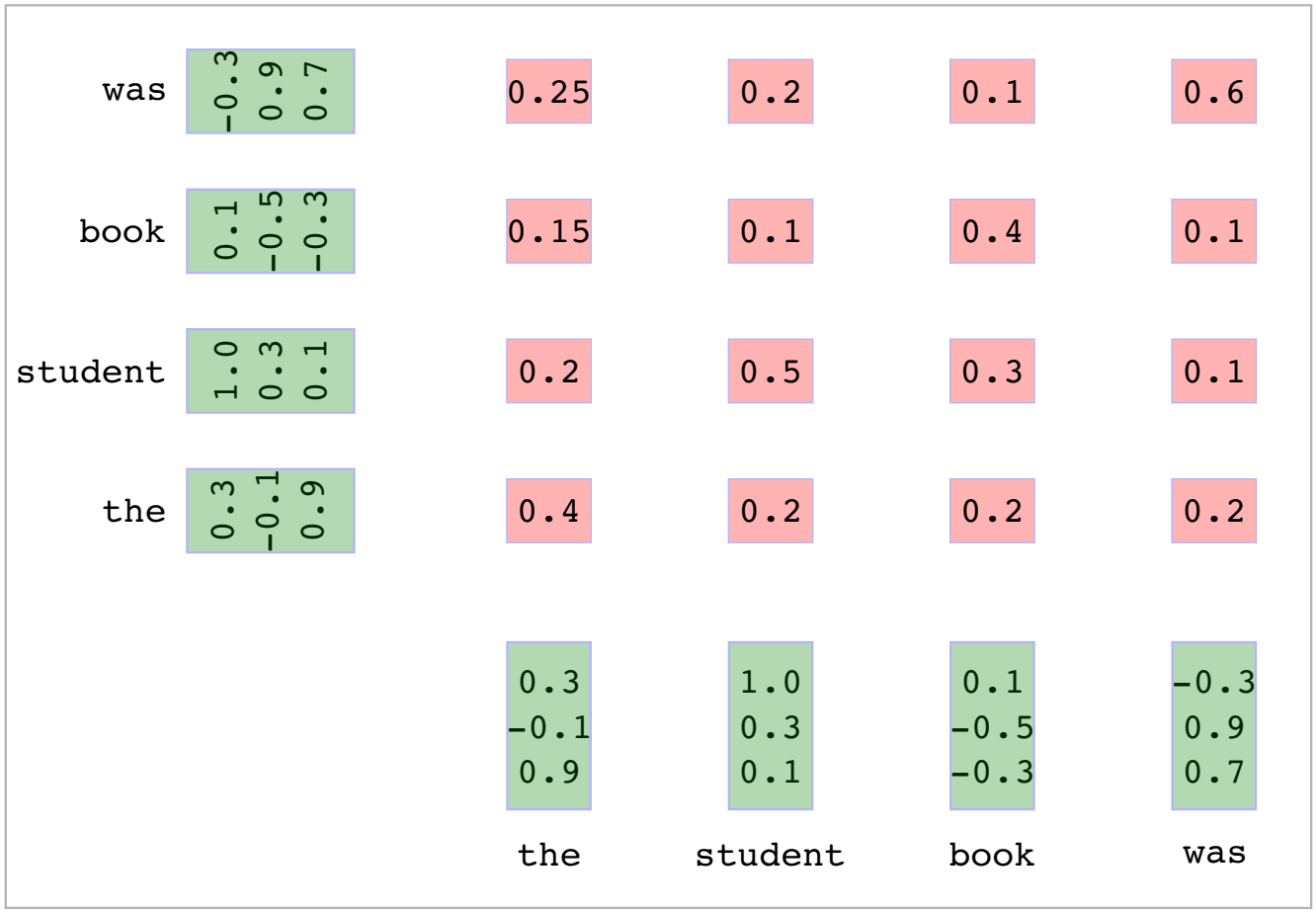
2 roles; each word vector is used in 3 ways:

- Role 1: Query: What 'type' am I ('noun-ness')?;
- Role 1: Key: How should my 'type' influence others ('verb-ness')?
- Role 2: Value: What should I 'give' ('plurality')?

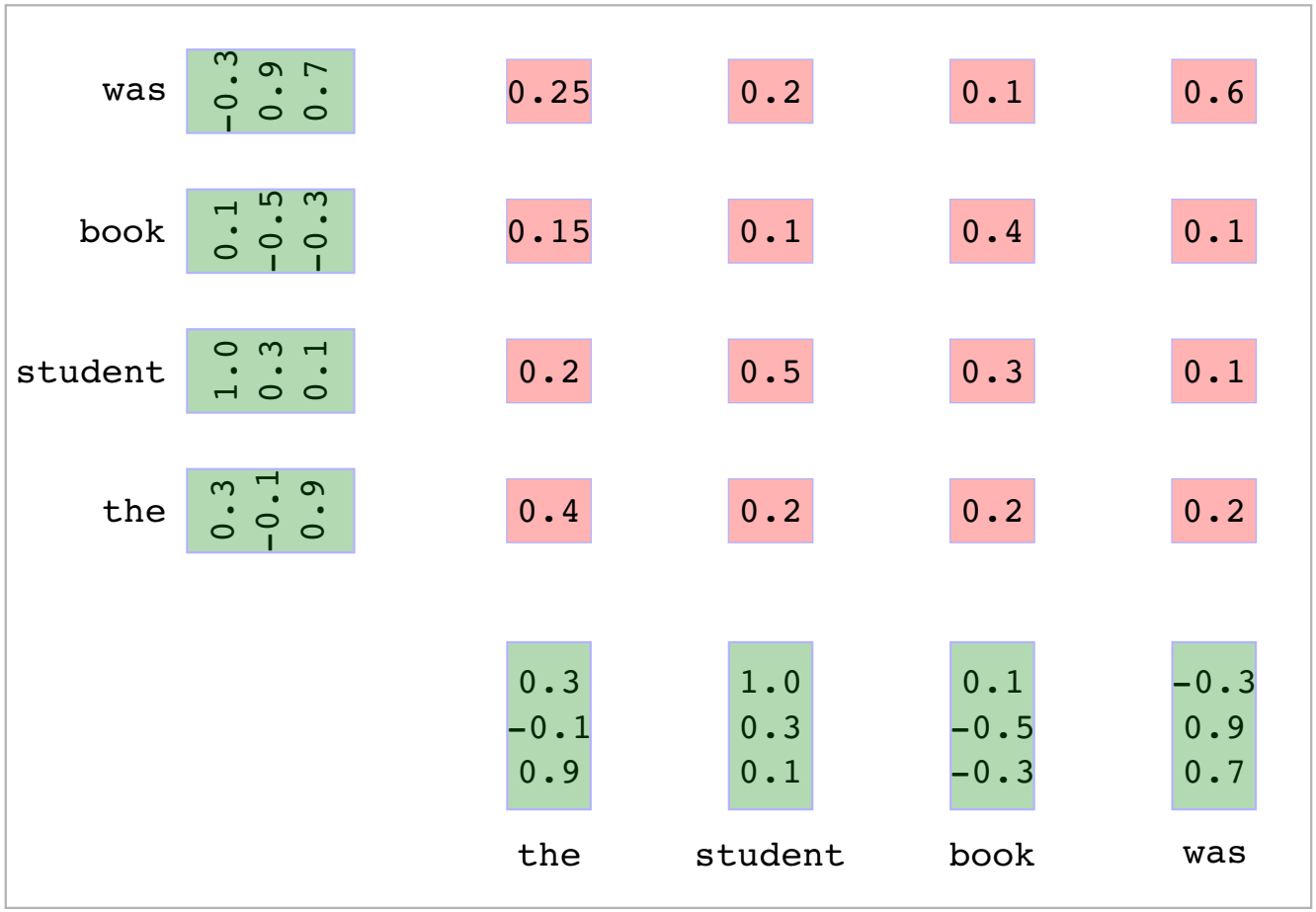
Each role has learnable parameters mapping a word vector to a new vector:

- Learnable Query matrix: $\mathbf{W}_q$
- Learnable Key matrix: $\mathbf{W}_k$
- Learnable Value matrix: $\mathbf{W}_v$

Self-attention: Word re-weighting

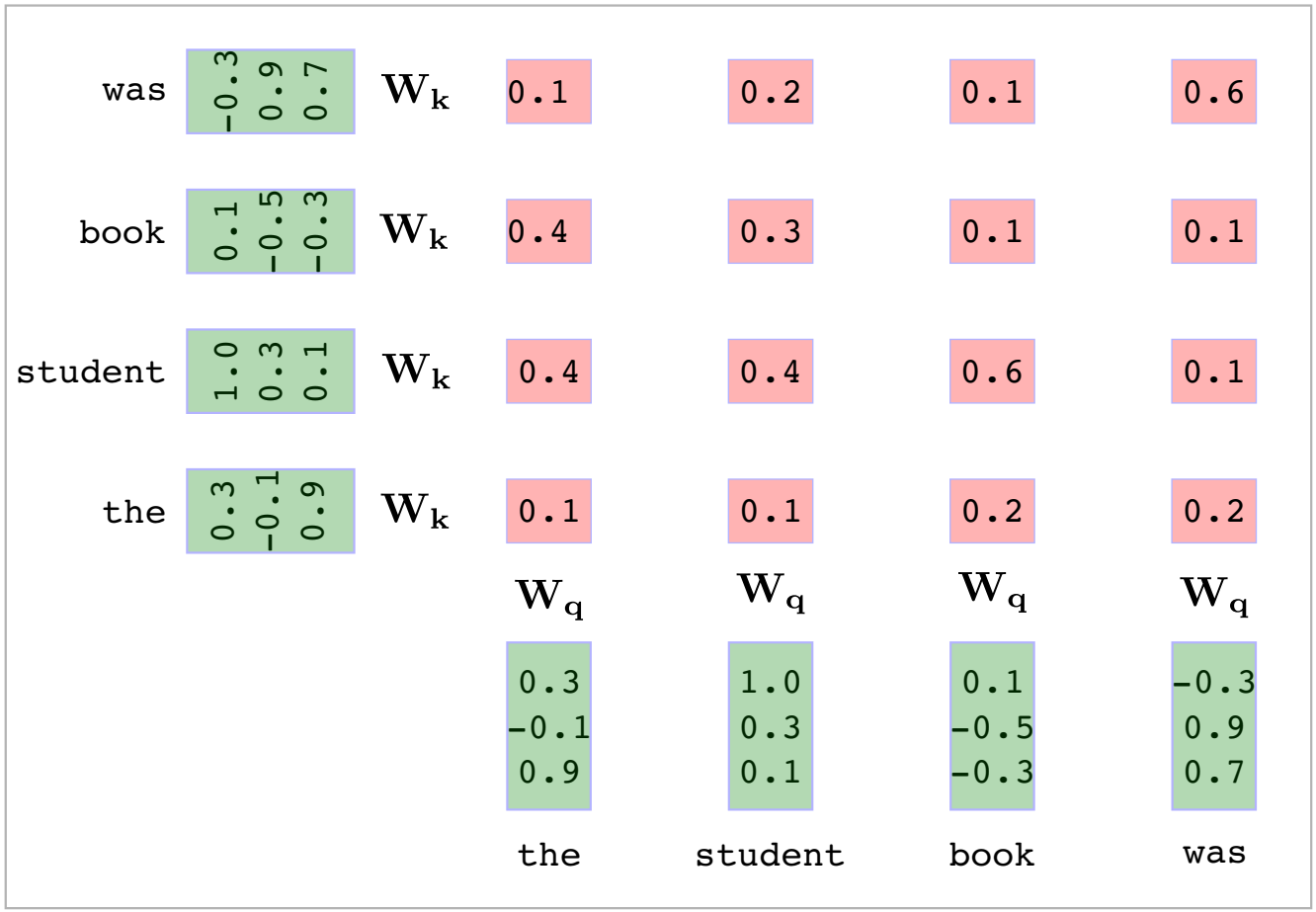


Self-attention: Word re-weighting

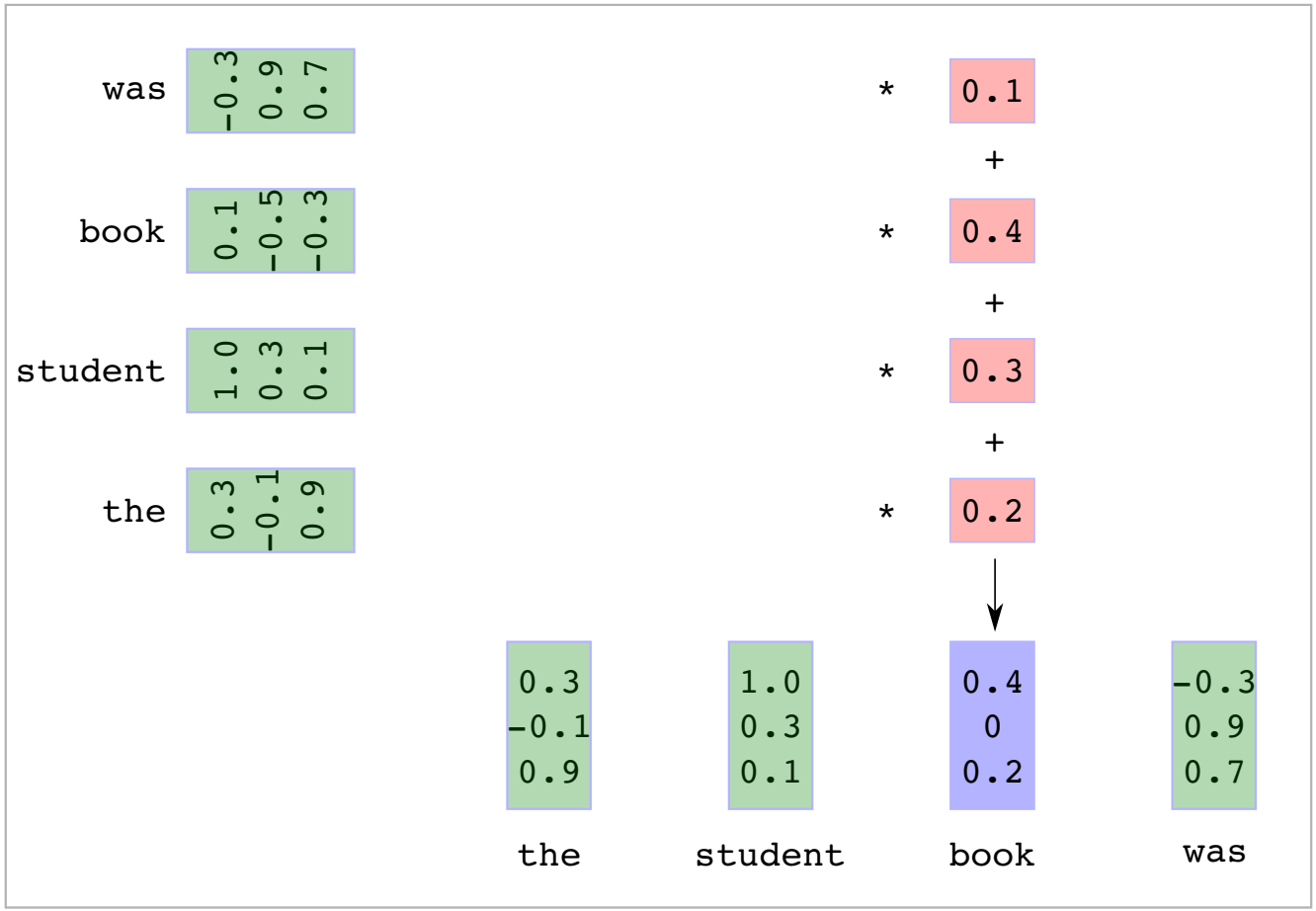


Q: Where would you add $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$?
(ie: the learnable parameters mapping a word vector to a new vector)

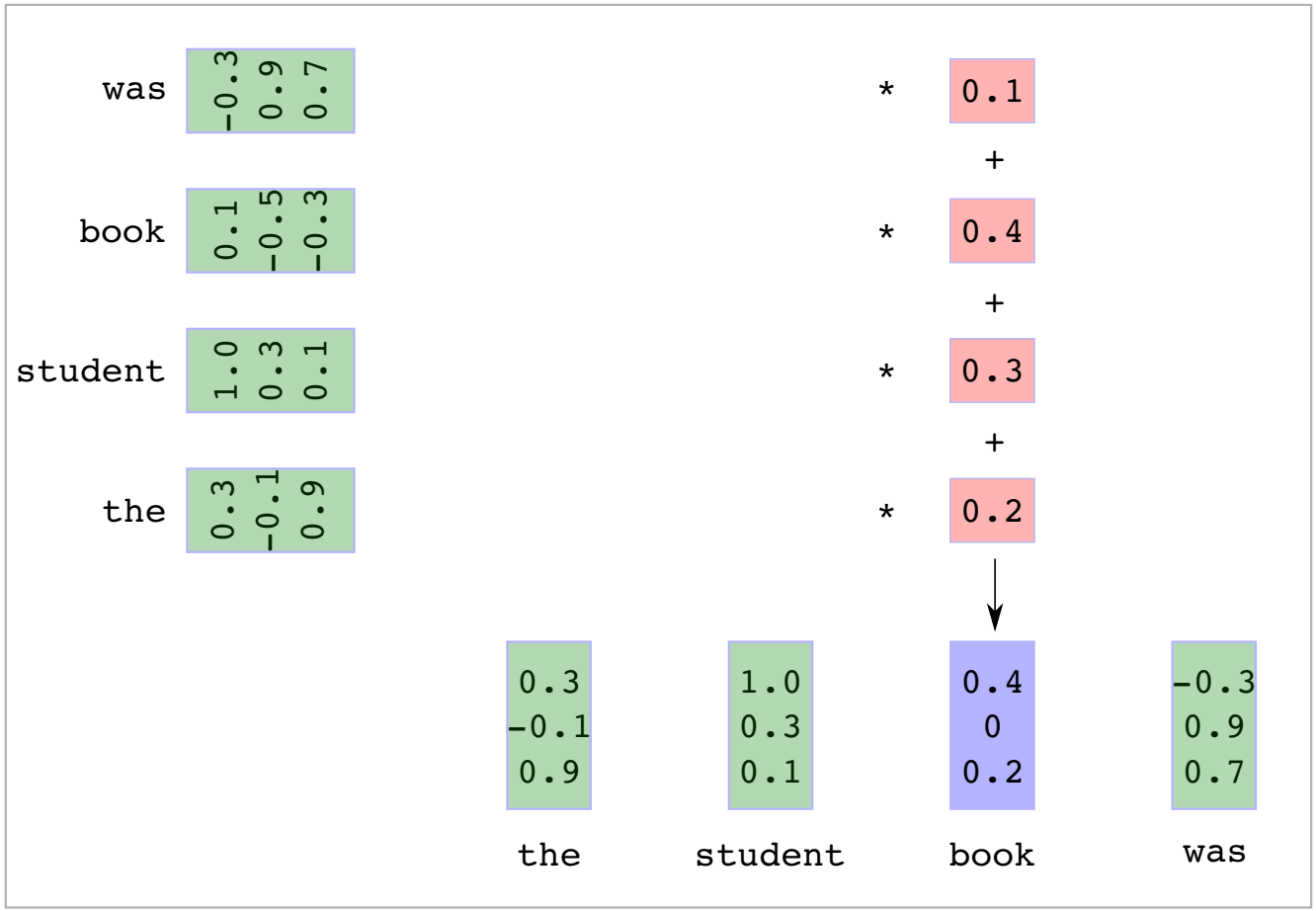
Self-attention: Word re-weighing



Self-attention: Word re-weighing

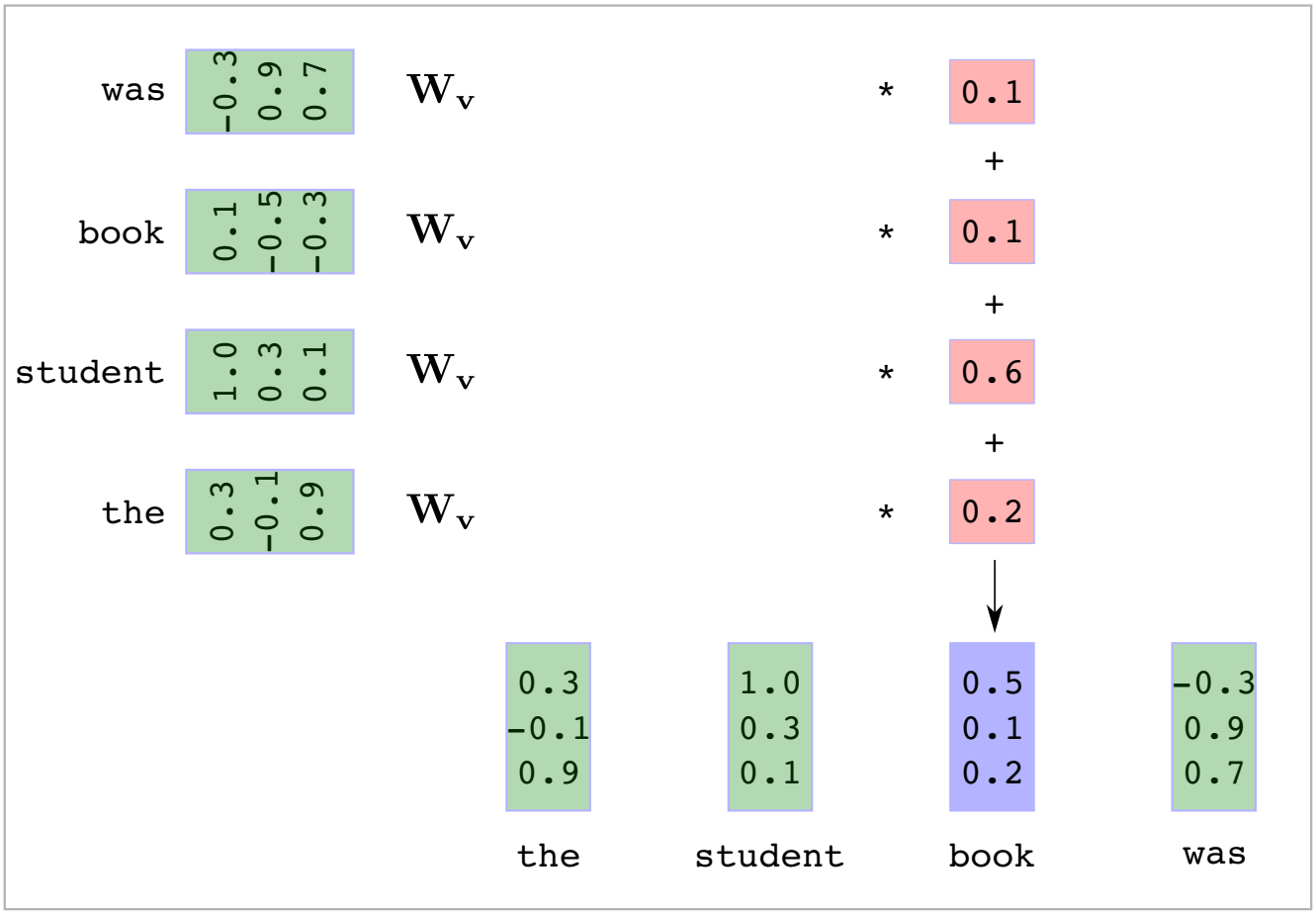


Self-attention: Word re-weighing



Q: Where would you add $\mathbf{W}_v$?

Self-attention: Word re-weighing



Formalizing the operations

Self attention is a sequence-to-sequence operation.

Formalizing the operations

Self attention is a sequence-to-sequence operation.

- Input vectors: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, of size d
- Output vectors: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, of size d

Formalizing the operations

Self attention is a sequence-to-sequence operation.

- Input vectors: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, of size d
- Output vectors: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Formalizing the operations

Self attention is a sequence-to-sequence operation.

- Input vectors: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, of size d
- Output vectors: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Self-attention: Learning how to re-weigh words with all other words.

Formalizing the operations

Self attention is a sequence-to-sequence operation.

- Input vectors: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, of size d
- Output vectors: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Self-attention: Learning how to re-weigh words with all other words.

Get Query, Key, and Value vectors: (Pre-compute this for each word $\mathbf{x}_i$:)

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i,$$

Formalizing the operations

Self attention is a sequence-to-sequence operation.

- Input vectors: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, of size d
- Output vectors: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Self-attention: Learning how to re-weigh words with all other words.

Get Query, Key, and Value vectors: Pre-compute this for each word $\mathbf{x}_i$:

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i,$$

Compute word similarities, i and j: Fill the n^2 matrix of all word pairs $\mathbf{q}_i$ and $\mathbf{k}_j$:

$$w'_{ij} = \mathbf{q}_i^T \mathbf{k}_j,$$

Formalizing the operations

Self attention is a sequence-to-sequence operation.

- Input vectors: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, of size d
- Output vectors: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Self-attention: Learning how to re-weigh words with all other words.

Get Query, Key, and Value vectors: Pre-compute this for each word $\mathbf{x}_i$:

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i,$$

Compute word similarities, i and j: Fill the n^2 matrix of all word pairs $\mathbf{q}_i$ and $\mathbf{k}_j$:

$$w'_{ij} = \mathbf{q}_i^T \mathbf{k}_j,$$

Normalize similarities (softmax): For all word pairs $\mathbf{q}_i$ and $\mathbf{k}_j$:

$$w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}},$$

Formalizing the operations

Self attention is a sequence-to-sequence operation.

- Input vectors: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, of size d
- Output vectors: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Self-attention: Learning how to re-weigh words with all other words.

Get Query, Key, and Value vectors:

Pre-compute this for each word $\mathbf{x}_i$:

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i,$$

Compute word similarities, i and j:

Fill the n^2 matrix of all word pairs $\mathbf{q}_i$ and $\mathbf{k}_j$:

$$w'_{ij} = \mathbf{q}_i^T \mathbf{k}_j,$$

Normalize similarities (softmax):

For all word pairs $\mathbf{q}_i$ and $\mathbf{k}_j$:

$$w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}},$$

Value-reweighted output vector:

The new vector $\mathbf{y}_i$ is a weighted sum of all value vectors $\mathbf{v}_j$:

$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{v}_j.$$

Formalizing the operations

Self attention is a sequence-to-sequence operation.

- Input vectors: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, of size d
- Output vectors: $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, of size d

Start with words:	*the, student, reads, a, book*
Learned embedding vectors:	$\mathbf{x}_{\text{the}}, \mathbf{x}_{\text{student}}, \mathbf{x}_{\text{reads}}, \mathbf{x}_{\text{a}}, \mathbf{x}_{\text{book}}$
Self attention converts this to:	$\mathbf{y}_{\text{the}}, \mathbf{y}_{\text{student}}, \mathbf{y}_{\text{reads}}, \mathbf{y}_{\text{a}}, \mathbf{y}_{\text{book}}$

Self-attention: Learning how to re-weigh words with all other words.

Get Query, Key, and Value vectors:

Pre-compute this for each word $\mathbf{x}_i$:

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i,$$

Compute word similarities, i and j:

Fill the n^2 matrix of all word pairs $\mathbf{q}_i$ and $\mathbf{k}_j$:

$$w'_{ij} = \mathbf{q}_i^T \mathbf{k}_j,$$

Normalize similarities (softmax):

For all word pairs $\mathbf{q}_i$ and $\mathbf{k}_j$:

$$w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}},$$

Value-reweighted output vector:

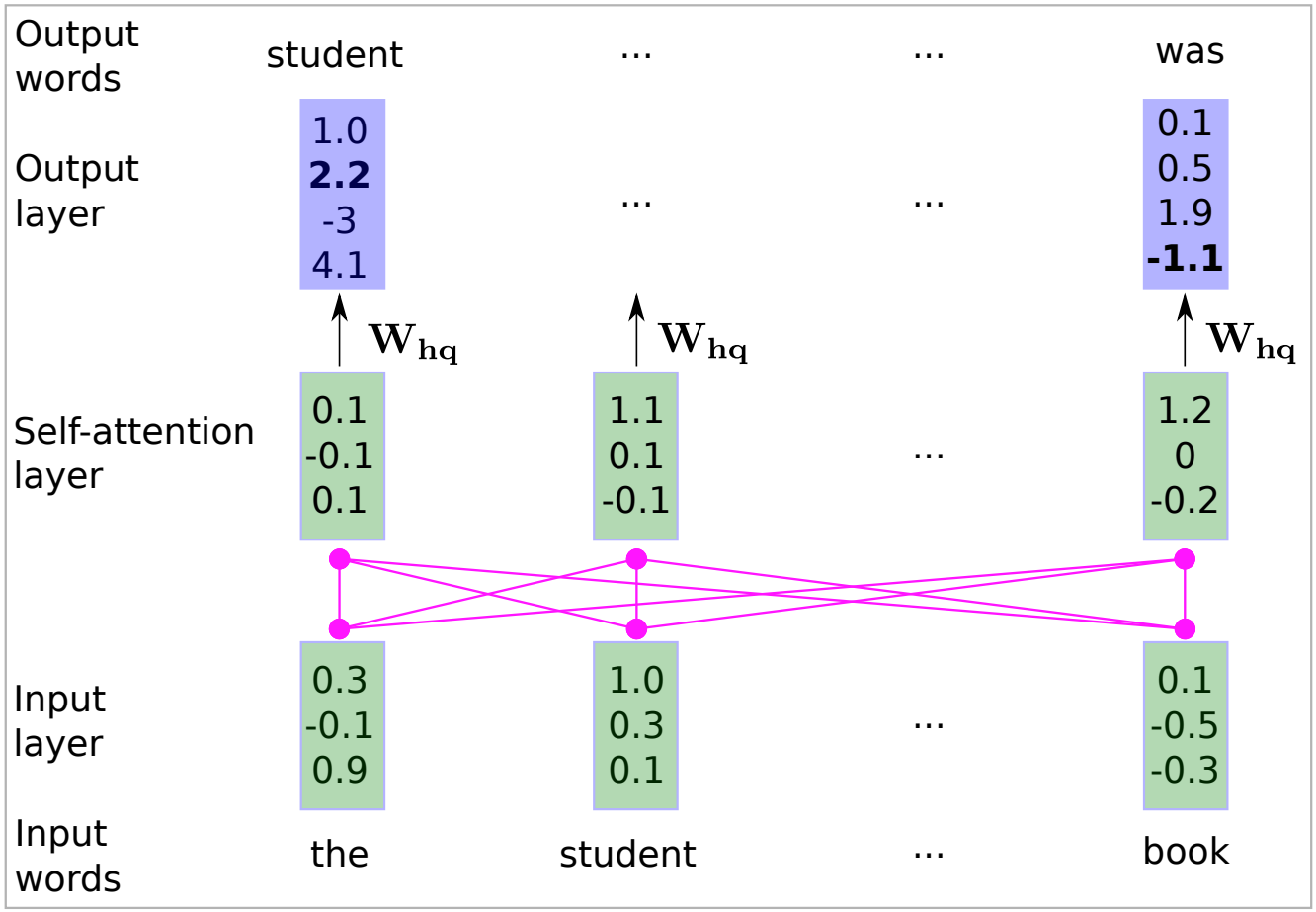
The new vector $\mathbf{y}_i$ is a weighted sum of all value vectors $\mathbf{v}_j$:

$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{v}_j.$$

Questions?

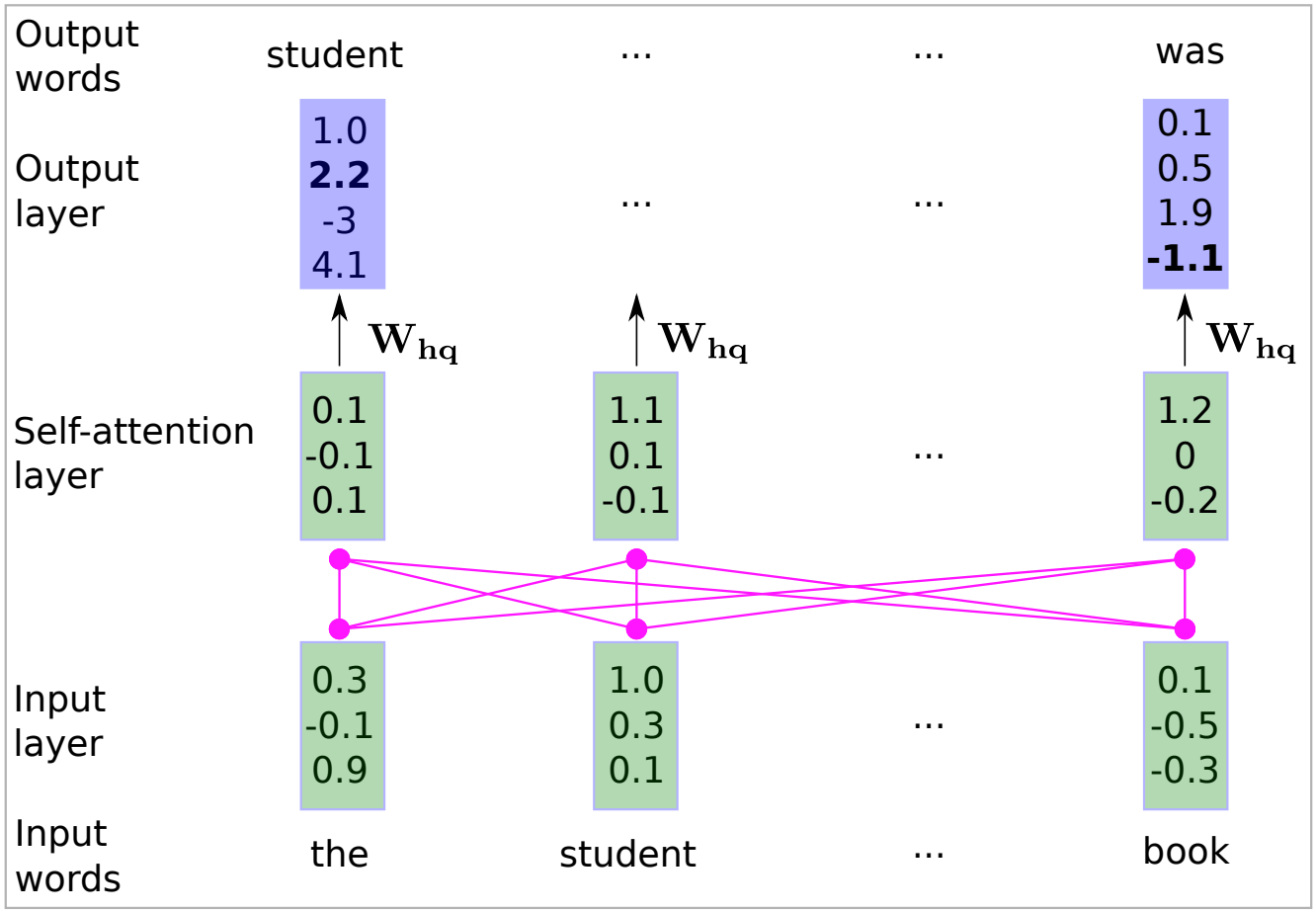
Questions?

Questions?



Self-attention: Learning how to re-weight words with all other words.

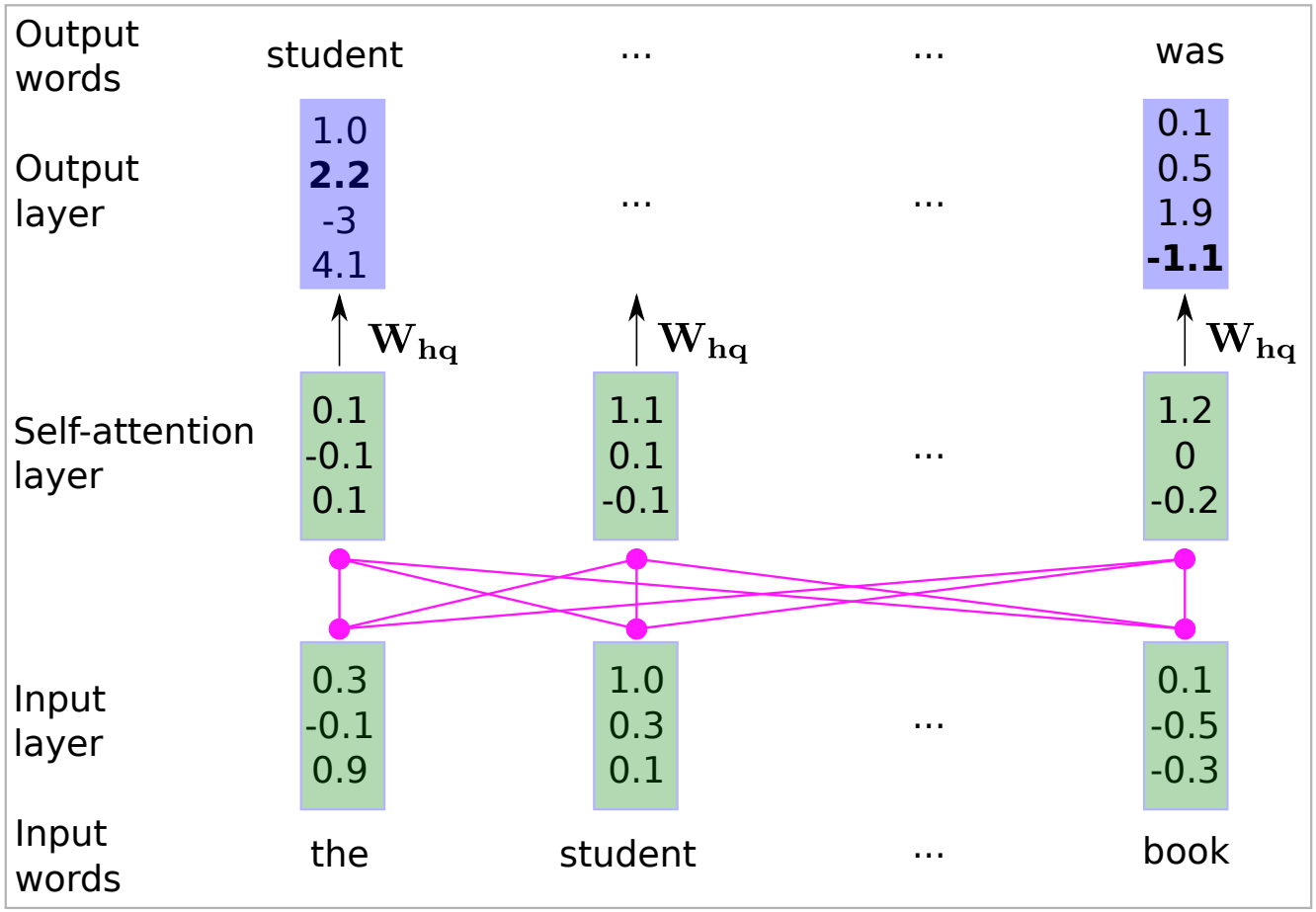
Questions?



Self-attention: Learning how to re-weight words with all other words.

Q: If the length of the sentence changes, how would the nr of parameters change?

Questions?



Self-attention: Learning how to re-weight words with all other words.

Q: If the length of the sentence changes, how would the nr of parameters change?

A: No change; all learnable parameters shared over each word.

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Q: What do you notice about this sentence?

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Q: What do you notice about this sentence?
A: (*eating shoots, and eating leaves*) or (*it eats, then it shoots, then it leaves*).

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Q: What do you notice about this sentence?
A: (*eating shoots, and eating leaves*) or (*it eats, then it shoots, then it leaves*).

Q: How is this currently modeled?

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Q: What do you notice about this sentence?
A: (*eating shoots, and eating leaves*) or (*it eats, then it shoots, then it leaves*).

Q: How is this currently modeled?
A: It is not. Currently, no difference is possible.

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Q: What do you notice about this sentence?
A: (*eating shoots, and eating leaves*) or (*it eats, then it shoots, then it leaves*).

Q: How is this currently modeled?
A: It is not. Currently, no difference is possible.

Use multiple 'heads': multiple variants of $\mathbf{W}_q^r, \mathbf{W}_k^r, \mathbf{W}_v^r$, indexed by r.

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Q: What do you notice about this sentence?
A: (*eating shoots, and eating leaves*) or (*it eats, then it shoots, then it leaves*).

Q: How is this currently modeled?
A: It is not. Currently, no difference is possible.

Use multiple 'heads': multiple variants of $\mathbf{W}_q^r, \mathbf{W}_k^r, \mathbf{W}_v^r$, indexed by r.
Each head can then learn different instantiations.

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Q: What do you notice about this sentence?
A: (*eating shoots, and eating leaves*) or (*it eats, then it shoots, then it leaves*).

Q: How is this currently modeled?
A: It is not. Currently, no difference is possible.

Use multiple 'heads': multiple variants of $\mathbf{W}_q^r, \mathbf{W}_k^r, \mathbf{W}_v^r$, indexed by r.
Each head can then learn different instantiations.

Option: Narrow multi-head attention

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Q: What do you notice about this sentence?
A: (*eating shoots, and eating leaves*) or (*it eats, then it shoots, then it leaves*).

Q: How is this currently modeled?
A: It is not. Currently, no difference is possible.

Use multiple 'heads': multiple variants of $\mathbf{W}_q^r, \mathbf{W}_k^r, \mathbf{W}_v^r$, indexed by r.
Each head can then learn different instantiations.

Option: Narrow multi-head attention

- Divide vector in k chunks; eg: turn single 256d vector in eight 32d heads

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Q: What do you notice about this sentence?
A: (*eating shoots, and eating leaves*) or (*it eats, then it shoots, then it leaves*).

Q: How is this currently modeled?
A: It is not. Currently, no difference is possible.

Use multiple 'heads': multiple variants of $\mathbf{W}_q^r, \mathbf{W}_k^r, \mathbf{W}_v^r$, indexed by r.
Each head can then learn different instantiations.

Option: Narrow multi-head attention

- Divide vector in k chunks; eg: turn single 256d vector in eight 32d heads

Option: Wide multi-head attention

Self attention: Multi-head attention

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Consider the sentence:

The, koala, eats, shoots, and, leaves.

Q: What do you notice about this sentence?
A: (*eating shoots, and eating leaves*) or (*it eats, then it shoots, then it leaves*).

Q: How is this currently modeled?
A: It is not. Currently, no difference is possible.

Use multiple 'heads': multiple variants of $\mathbf{W}_q^r, \mathbf{W}_k^r, \mathbf{W}_v^r$, indexed by r.
Each head can then learn different instantiations.

Option: Narrow multi-head attention

- Divide vector in k chunks; eg: turn single 256d vector in eight 32d heads

Option: Wide multi-head attention

- Apply each head independently, concatenate and project back; eg: Run 8x a 256d vector, project 2048d back to 256d

Self attention: Position embedding

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Q: If I change the word order, do I get different results?

Self attention: Position embedding

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Q: If I change the word order, do I get different results?
A: No; dot-product and sum is independent of position (permutation invariant)

Self attention: Position embedding

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Q: If I change the word order, do I get different results?
A: No; dot-product and sum is independent of position (permutation invariant)

Q: Can you think of a solution?

Self attention: Position embedding

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Q: If I change the word order, do I get different results?
A: No; dot-product and sum is independent of position (permutation invariant)

Q: Can you think of a solution?
A: Add position information to the input vectors.

Self attention: Position embedding

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Q: If I change the word order, do I get different results?
A: No; dot-product and sum is independent of position (permutation invariant)

Q: Can you think of a solution?
A: Add position information to the input vectors.

Option: Learned position embedding

Self attention: Position embedding

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Q: If I change the word order, do I get different results?
A: No; dot-product and sum is independent of position (permutation invariant)

Q: Can you think of a solution?
A: Add position information to the input vectors.

Option: Learned position embedding

- Embed the positions; like $\mathbf{x}_{\text{cat}}, \mathbf{x}_{\text{Susan}}$ learn $\mathbf{x}_{12}, \mathbf{x}_{25}$ and sum with word vectors

Self attention: Position embedding

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Q: If I change the word order, do I get different results?
A: No; dot-product and sum is independent of position (permutation invariant)

Q: Can you think of a solution?
A: Add position information to the input vectors.

Option: Learned position embedding

- Embed the positions; like $\mathbf{x}_{\text{cat}}, \mathbf{x}_{\text{Susan}}$ learn $\mathbf{x}_{12}, \mathbf{x}_{25}$ and sum with word vectors

Option: Position encoding

Self attention: Position embedding

D2L book: `https://d2l.ai/chapter_attention-mechanisms/index.html`
See also: `http://www.peterbloem.nl/blog/transformers`

Q: If I change the word order, do I get different results?
A: No; dot-product and sum is independent of position (permutation invariant)

Q: Can you think of a solution?
A: Add position information to the input vectors.

Option: Learned position embedding

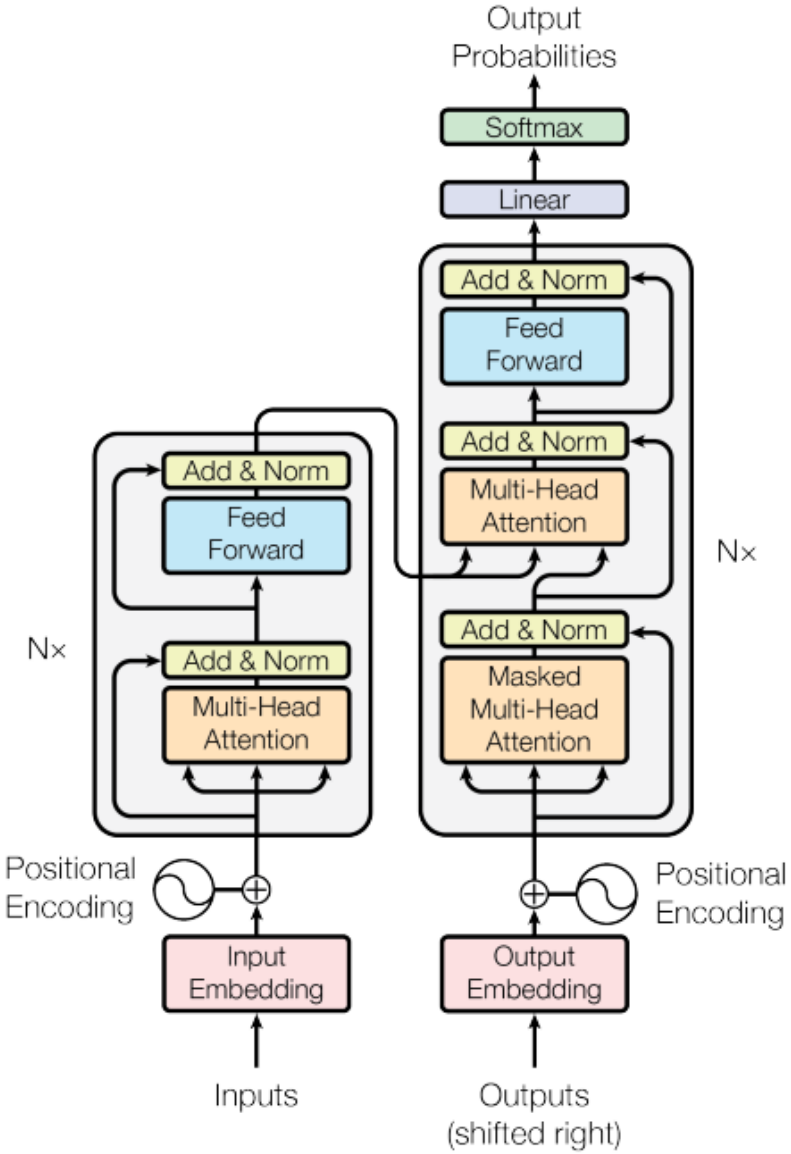
- Embed the positions; like $\mathbf{x}_{\text{cat}}, \mathbf{x}_{\text{Susan}}$ learn $\mathbf{x}_{12}, \mathbf{x}_{25}$ and sum with word vectors

Option: Position encoding

- Construct a function $f : \mathbb{N} \to \mathbb{R}^d$ and let the network learn with it.

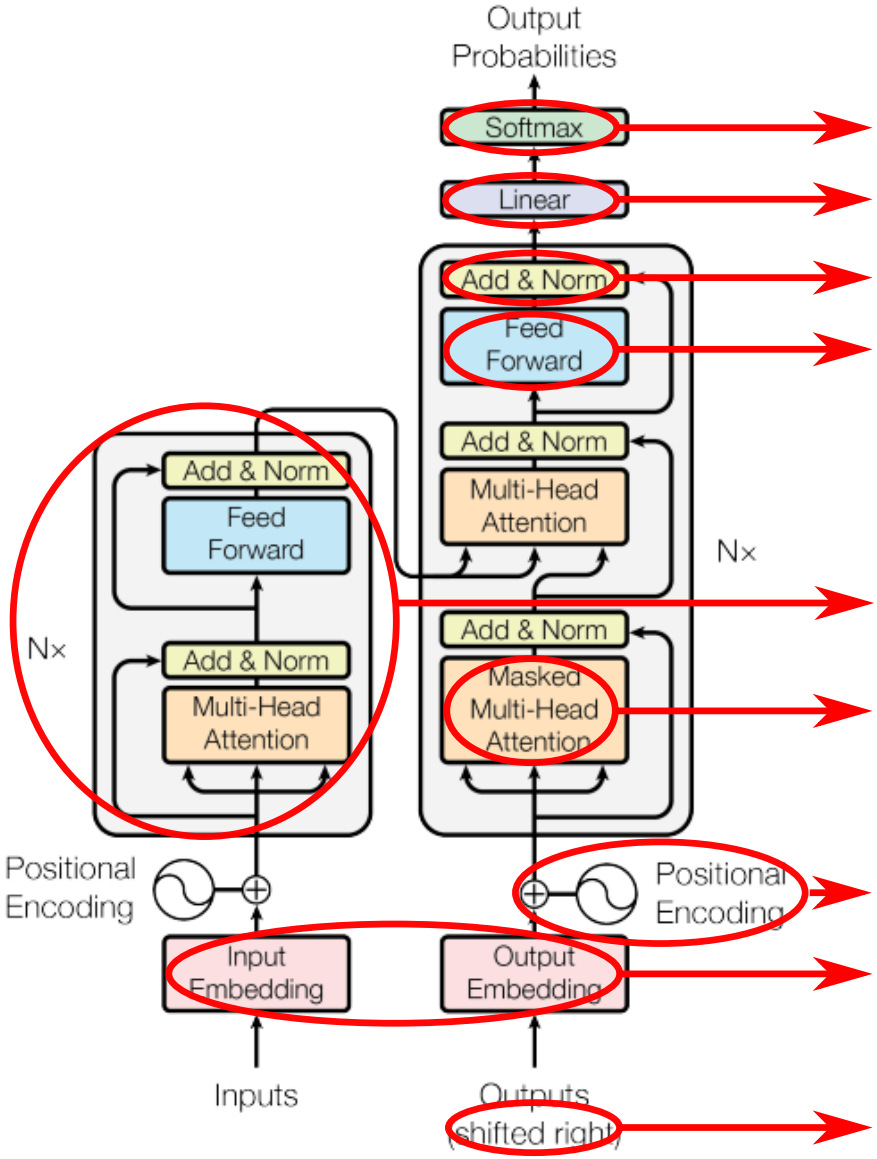
Attention is all you need: full *Transformer* architecture

Paper: `https://research.google/pubs/attention-is-all-you-need/`



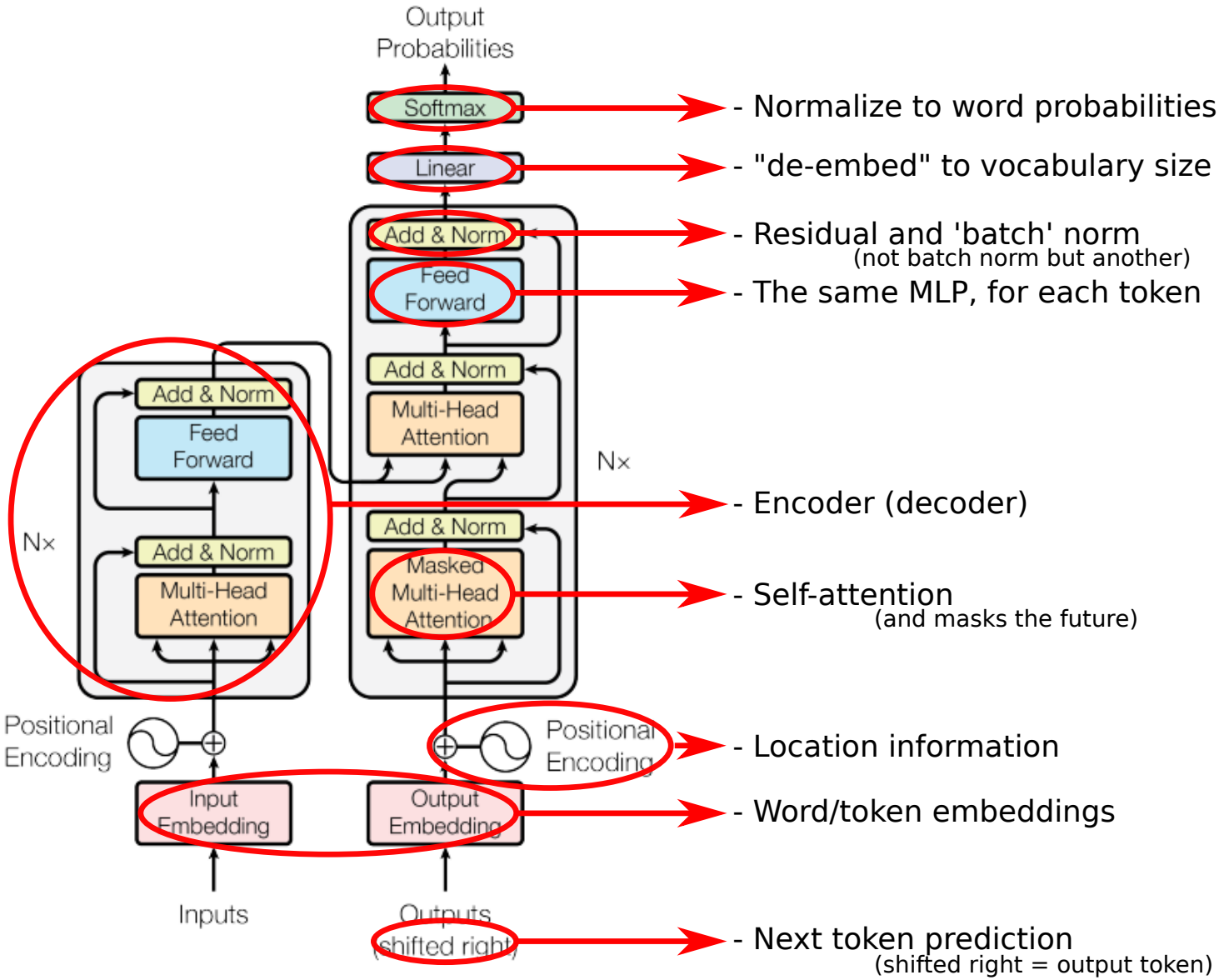
Attention is all you need: full *Transformer* architecture

Paper: `https://research.google/pubs/attention-is-all-you-need/`



Attention is all you need: full *Transformer* architecture

Paper: `https://research.google/pubs/attention-is-all-you-need/`



Questions?

Questions?

There are (too) many resources online.

I like these:

- From scratch:
  `http://www.peterbloem.nl/blog/transformers`
- Also from scratch:
  `https://e2eml.school/transformers.html`
- Illustrated self-attention
  `https://colab.research.google.com/github/mrm8488/shared_colab_notebooks/blob/master/basic_self_attention_.ipynb`
- 3Blue1Brown (Grant Sanderson) has great videos; also about self-attention:
  `https://www.3blue1rown.com/lessons/attention`

Summary

- Word embeddings
- Self attention
- Queries, keys, values
- Multi-head
- Position encodings