The "Deep Learning for NLP" Lecture Roadmap

~~Lecture 5: Text Vectorization~~
~~and the Bag-of-Words Model~~
~~Lecture 6: Embeddings~~

Lecture 7: Transformers (1/2)

Lecture 8: Transformers – (2/2)
Lecture 9-10: LLMs

15.S04: Hands-on Deep Learning
Spring 2024
Farias, Ramakrishnan

Transformers have proven to be an effective DNN architecture across a vast array of domains

Information Retrieval/Search

Machine Translation

Speech Recognition

Text-to-Speech

Computer Vision

Reinforcement Learning

Generative AI (LLMs, Text-to-image models,
Image Captioning, …)

Numerous special-purpose systems (e.g., AlphaFold)

…

We will use Search/Information Retrieval as the motivating use-case

- Find me all flights from BOS to LGA tomorrow morning
- How many customers abandoned their shopping carts?
- Find all contracts that are up for renewal next month

We will focus on this travel-related example today

"Find me all flights from BOS to LGA tomorrow morning"

We will focus on this travel-related example today

"Find me all flights from BOS to LGA tomorrow morning"

In these sorts of use-cases, a common approach is as follows: *Convert the natural language query into a structured query (i.e., SQL) that can be used to search/lookup info in a database.*

We will focus on this travel-related example today

"Find me all flights from BOS to LGA tomorrow morning"

In these sorts of use-cases, a common approach is as follows: *Convert the natural language query into a structured query (i.e., SQL) that can be used to search/lookup info in a database.*

To enable this, we need to automatically extract travel-related entities from the natural language query.

We will use the Airline Travel Information Systems (ATIS) dataset*

https://aclanthology.org/H90-1021/

Extracting “entities” from natural language

Given a query in natural language …

Input: I want to fly from boston at 7 am and arrive in denver at 11 in the morning

Dataset: Airline Travel Information Systems (ATIS)

Extracting “entities” from natural language

Given a query in natural language …

Input: I want to fly from boston at 7 am and arrive in denver at 11 in the morning

… classify each word in the query to a corresponding “slot”

Dataset: Airline Travel Information Systems (ATIS)

Classify each word in the query to a corresponding "slot" - Example

I want to fly from	O O O O O
boston	B-fromloc.city_name
at	O
7 am	B-depart_time.time I-depart_time.time
and arrive in	O O O
denver	B-toloc.city_name
at	O
11	B-arrive_time.time
in the	O O
morning	B-arrive_time.period_of_day

Slot Types in the ATIS dataset

```
'B-aircraft_code',
'B-airline_code',
'B-airline_name',
'B-airport_code',
'B-airport_name',
'B-arrive_date.date_relative',
'B-arrive_date.day_name',
'B-arrive_date.day_number',
'B-arrive_date.month_name',
'B-arrive_date.today_relative',
'B-arrive_time.end_time',
'B-arrive_time.period_mod',
'B-arrive_time.period_of_day',
'B-arrive_time.start_time',
'B-arrive_time.time',
'B-arrive_time.time_relative',
'B-city_name',
'B-class_type',
 ...
'I-round_trip',
'I-stoploc.city_name',
'I-time',
'I-today_relative',
'I-toloc.airport_name',
'I-toloc.city_name',
'I-toloc.state_name',
```

123 possible slots!

How can we solve this word-to-slot multi-class classification problem?

I want to fly from boston at 7 am and arrive in denver at 11 in the morning

O O O O O B-fromloc.city_name O B-depart_time.time I-depart_time.time O O O B-toloc.city_name O B-arrive_time.time O O B-arrive_time.period_of_day

How can we solve this word-to-slot multi-class classification problem?

I want to fly from boston at 7 am and arrive in denver at 11 in the morning

O O O O O B-fromloc.city_name O B-depart_time.time I-depart_time.time O O O B-toloc.city_name O B-arrive_time.time O O B-arrive_time.period_of_day

Each of the 18 words above must be assigned to one of 123 slot types!

If we could run the query sentence through a DNN and generate 18 outputs (one for each input word in the query), we could attach a 123-way softmax to each of those 18 outputs.

What must we take into account?

We want to generate an output that has the same length as the input (so that we can classify each output element to the right slot type)

What must we take into account?

We want to generate an output that has the same length as the input (so that we can classify each output element to the right slot type)

In addition, we would like to

- Take the surrounding context of each word into account
- Take the order of the words into account

Context matters

The meaning of a word can change dramatically depending on the context.
A single embedding – like GloVe - for all contexts a word can appear in isn’t good enough

Context matters

The meaning of a word can change dramatically depending on the context. A single embedding – like GloVe - for all contexts a word can appear in isn't good enough

Word	Example Contexts
	I will see you soon I will see this project to its end I see what you mean
bank	I went to the bank to apply for a loan I am banking on the job offer coming through I am standing on the left bank
it	The animal didn't cross the street because it was too tired The animal didn't cross the street because it was too wide
station	The train left the station on time The radio station was playing 60s hits I was stationed on a remote island in Polynesia

Order matters

<add your own examples 🙂>

The Transformer Architecture

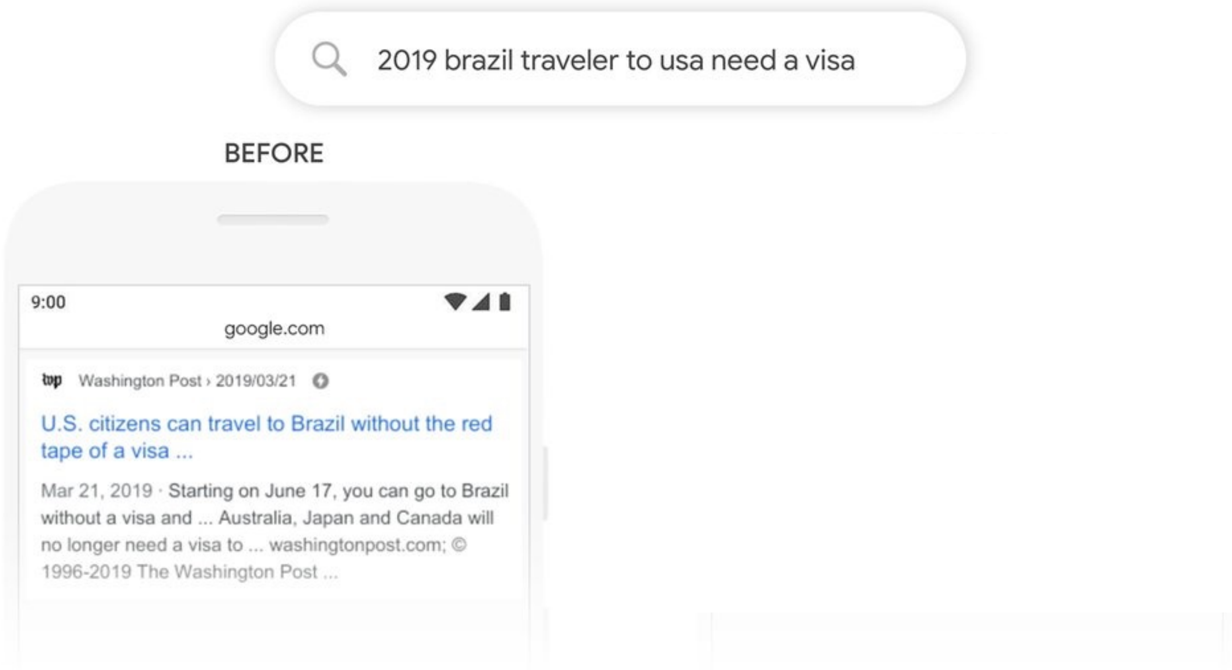
- Meets ALL the requirements we identified earlier
  - ✓ Takes the surrounding context of each word into account
  - ✓ Takes the order of the words into account
  - ✓ Can generate an output that has the same length as the input

The Transformer Architecture

- Meets ALL the requirements we identified earlier
  - ✓ Takes the surrounding context of each word into account
  - ✓ Takes the order of the words into account
  - ✓ Can generate an output that has the same length as the input

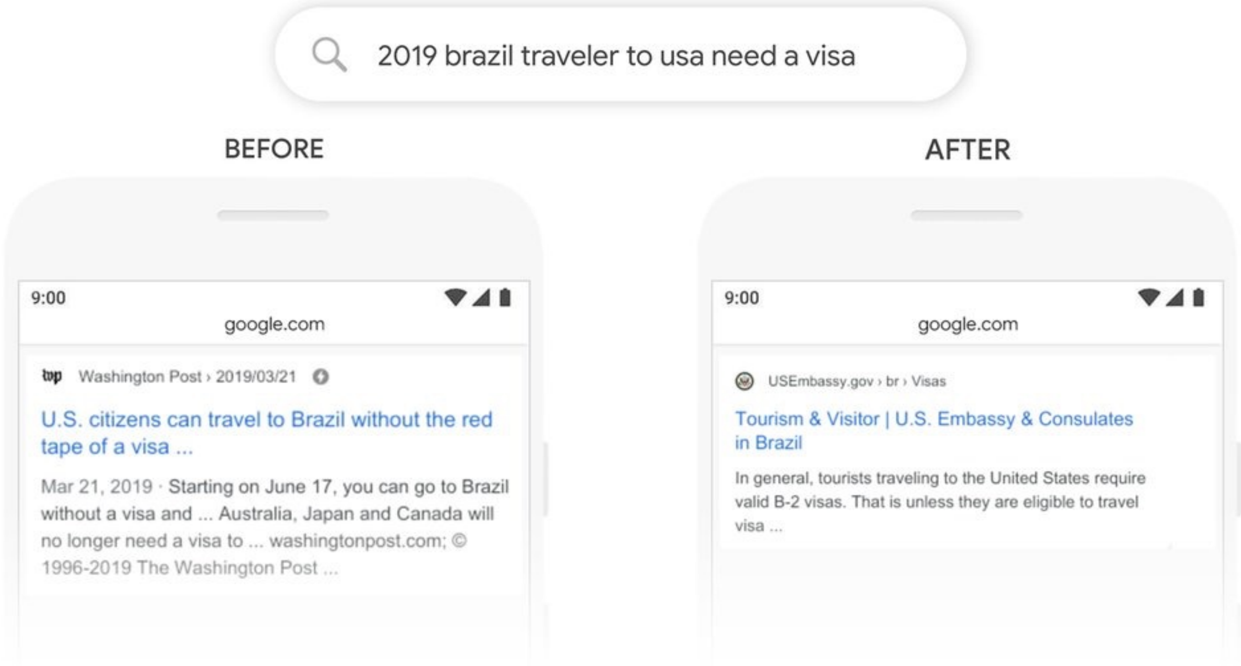
- Developed in 2017. Dramatic and on-going impact on DL

Effect of the Transformer on Google Search



https://blog.google/products/search/search-language-understanding-bert/

Effect of the Transformer on Google Search



https://blog.google/products/search/search-language-understanding-bert/

The Transformer Architecture

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

https://arxiv.org/abs/1706.03762

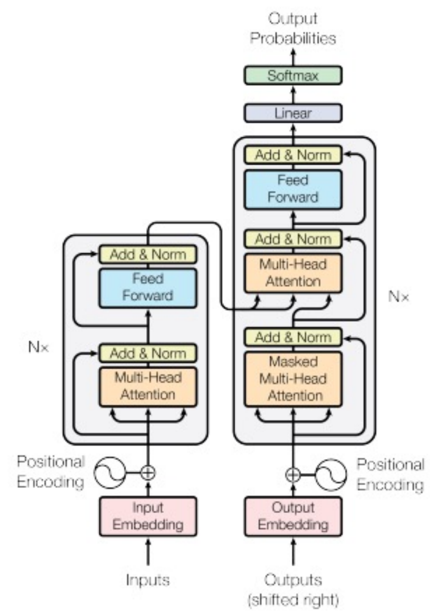


Figure 1: The Transformer - model architecture.

We will focus on this first

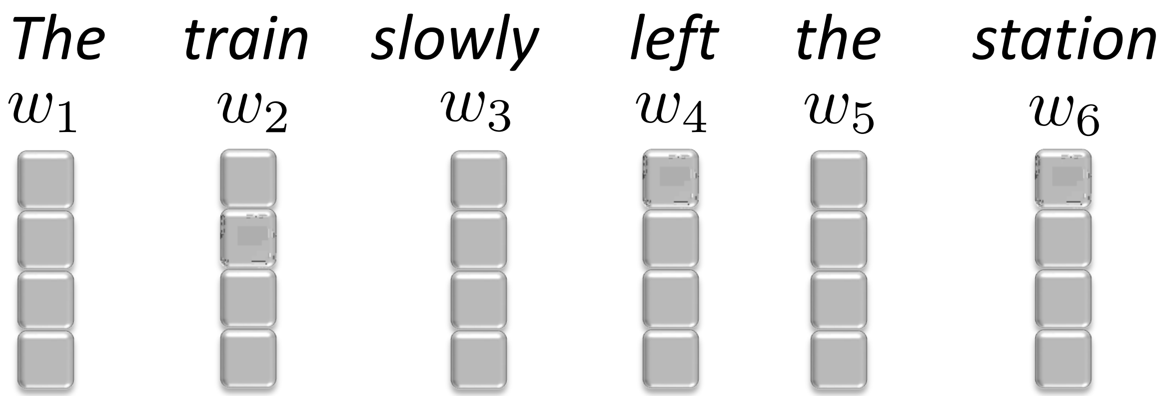
How to take the surrounding context of each word into account

How to take the order of the words into account

How to generate an output that has the same length as the input

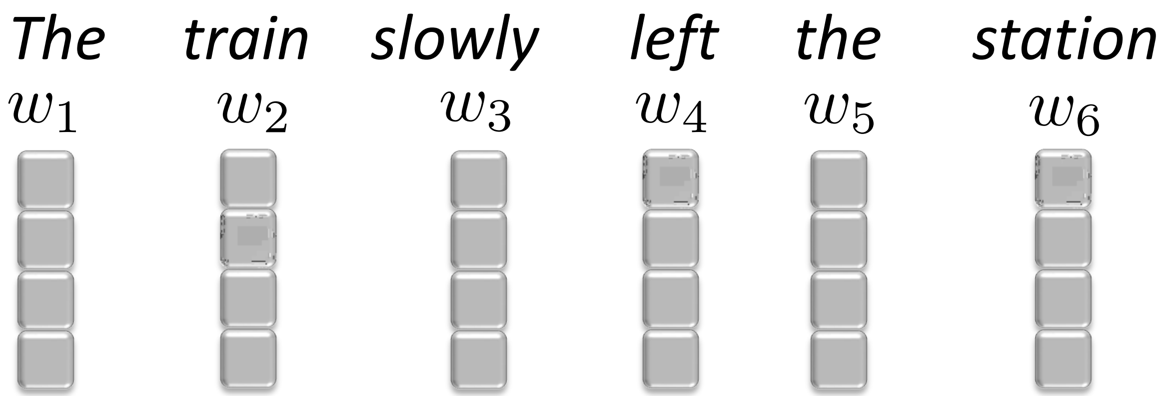
How to take the surrounding context of each word into account

From embeddings to *contextual embeddings*



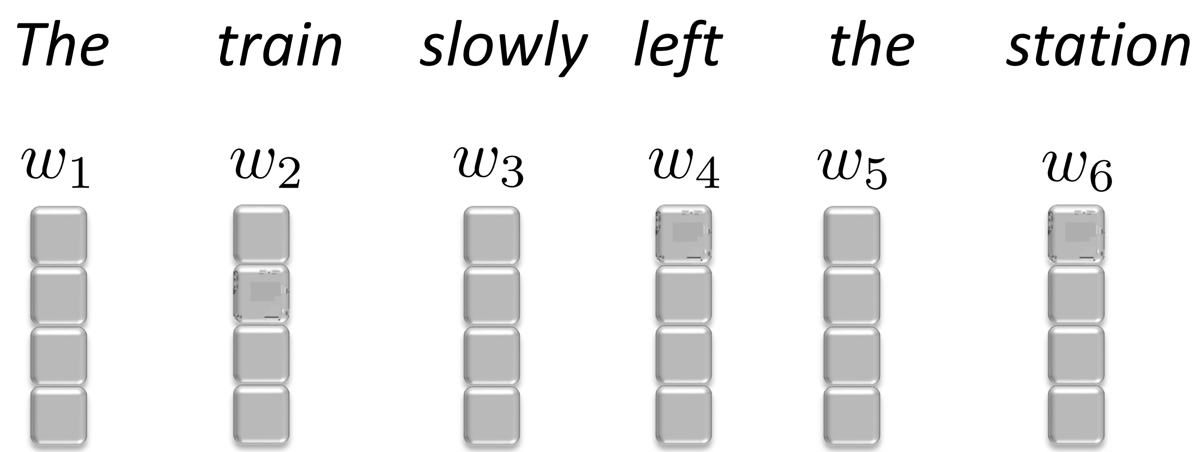
- We can easily get stand-alone embeddings for all the words

From embeddings to *contextual embeddings*



- We can easily get stand-alone embeddings for all the words
- How can we modify station’s embedding so that it incorporates the other words?

From embeddings to *contextual embeddings*

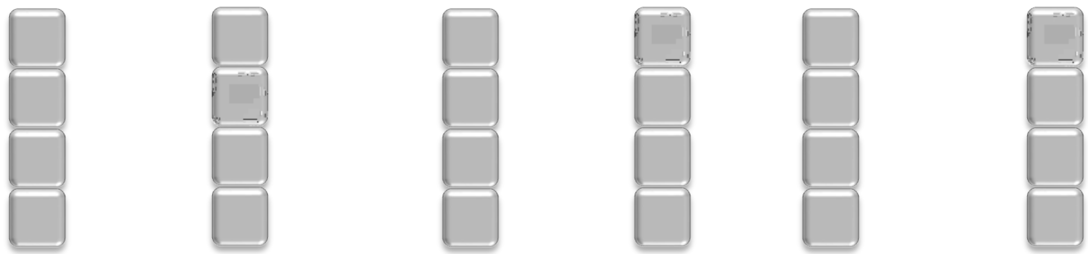


Imagine that we somehow know how much attention to give the other words i.e., how much weight to give the other words

From embeddings to *contextual embeddings*

The train slowly left the station

$w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5 \quad w_6$



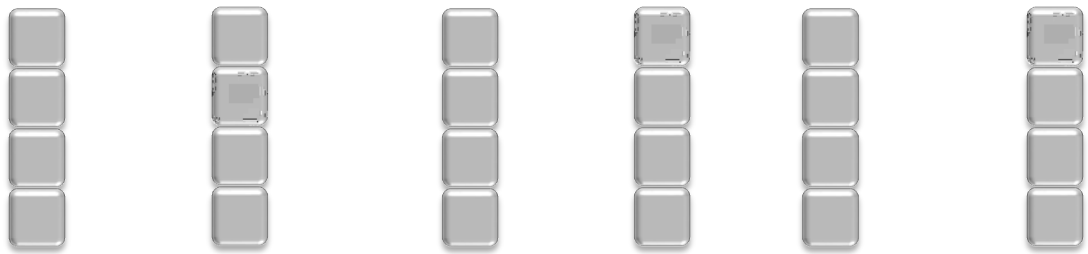
Imagine that we somehow know how much attention to give the other words i.e., how much weight to give the other words

Intuitively:
Which word(s) should get the most weight, which word(s) the least?

From embeddings to *contextual embeddings*

The *train* *slowly* *left* *the* *station*

w_1 w_2 w_3 w_4 w_5 w_6



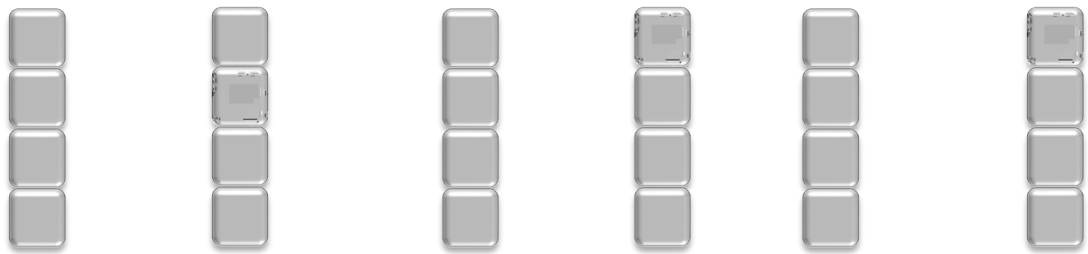
Imagine that we somehow know how much <u>attention</u> to give the other words i.e., how much <u>weight</u> to give the other words

We should give a lot of weight to ‘train’, a little to ‘slowly’ and ‘left’, and hardly anything to ‘the’.

From embeddings to *contextual embeddings*

The *train* *slowly* *left* *the* *station*

w_1 w_2 w_3 w_4 w_5 w_6



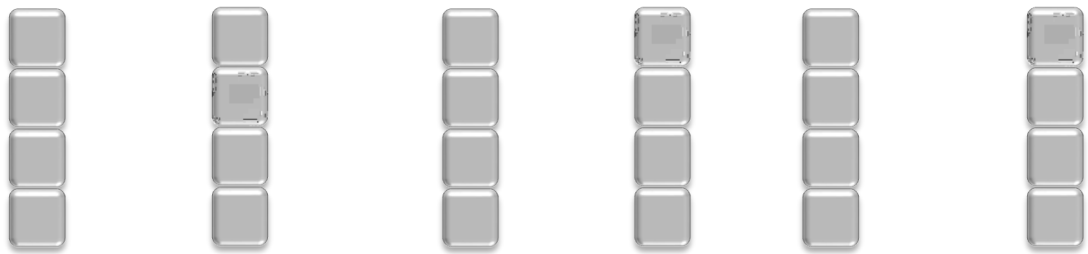
0.05	0.3	0.12	0.08	0.05	0.4

Maybe something like this?

From embeddings to *contextual embeddings*

The *train* *slowly* *left* *the* *station*

w_1 w_2 w_3 w_4 w_5 w_6

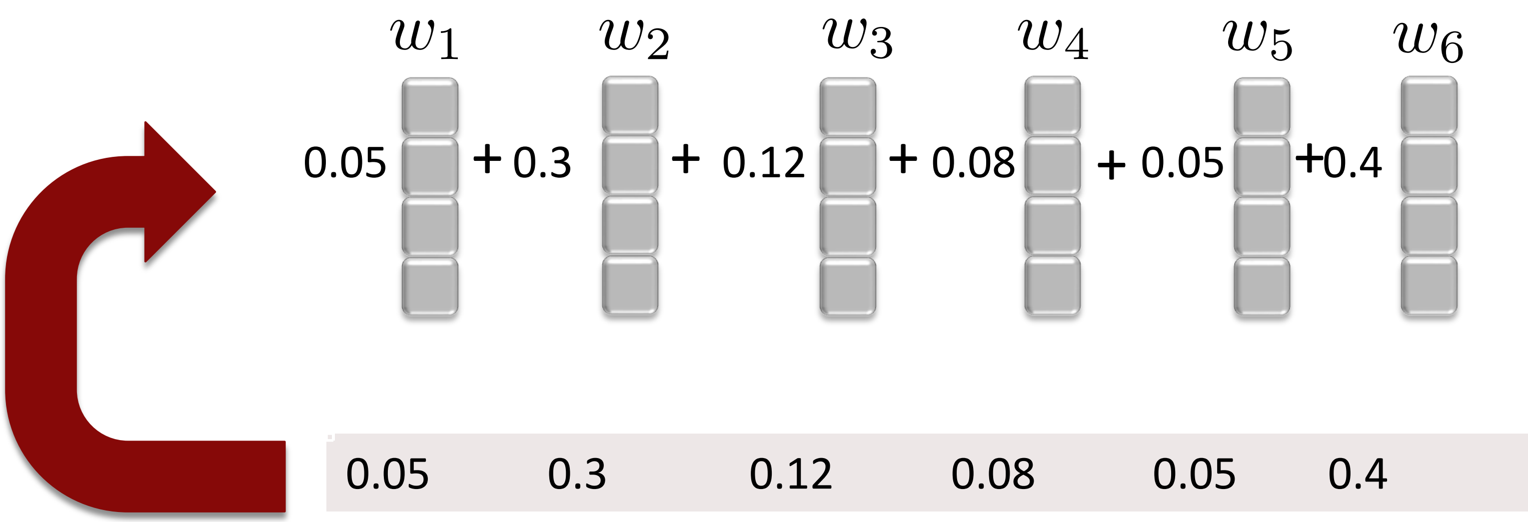


0.05	0.3	0.12	0.08	0.05	0.4

How can we use these weights to "contextualize" the stand-alone embedding w_6 for "station"?

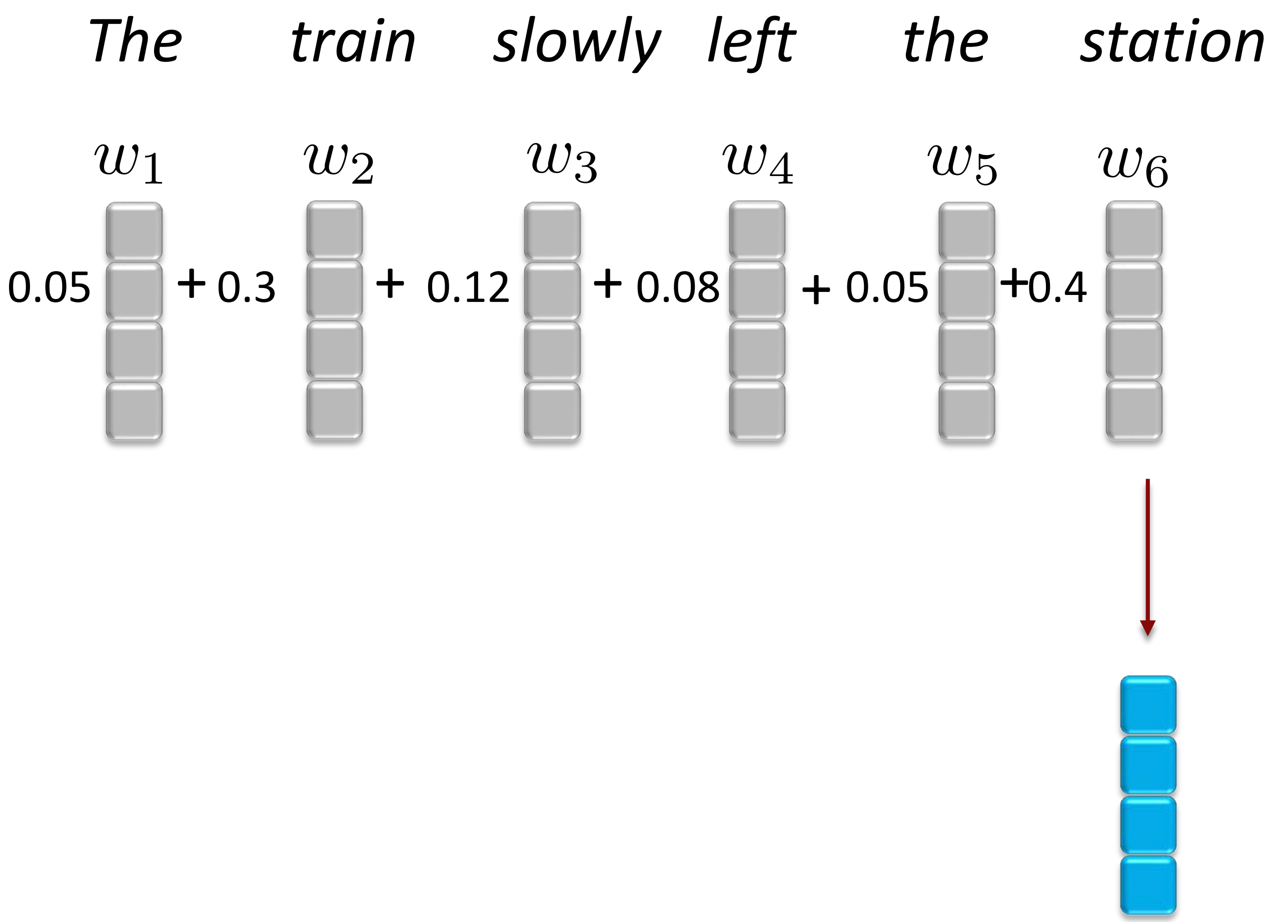
From embeddings to *contextual embeddings*

The train slowly left the station

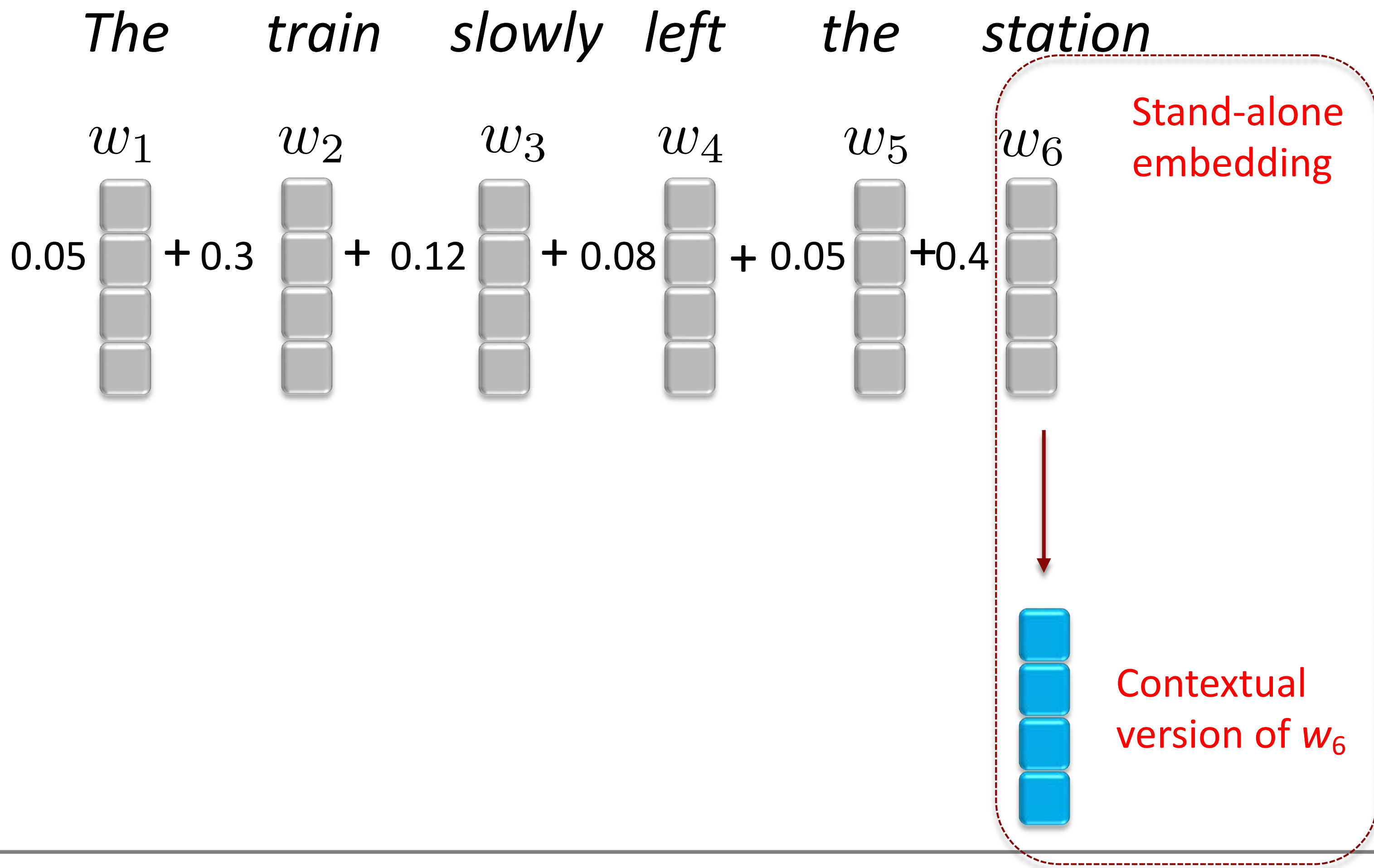


<u>Idea</u>: For each word, we can calculate a <u>weighted average</u> of the stand-alone embeddings of *all* the words in the sentence

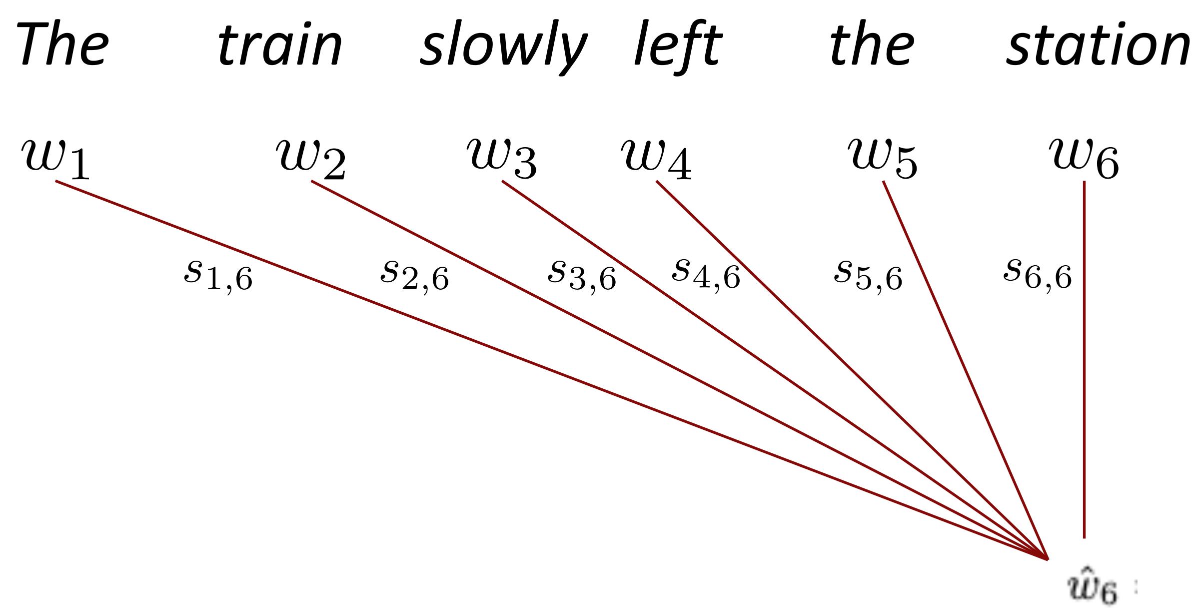
From embeddings to *contextual embeddings*



From embeddings to *contextual embeddings*



Let's write it more formally



Stand-alone embedding

$$\hat{w}_6 = s_{1,6}w_1 + s_{2,6}w_2 + s_{3,6}w_3 + s_{4,6}w_4 + s_{5,6}w_5 + s_{6,6}w_6$$

Contextual version of w_6

For a given word (e.g., ‘station’), how should the weights be chosen?

For a given word (e.g., ‘station’), how should the weights of the other words be chosen?

Intuition

- The weight of a word should be proportional to how related it is to the word “station”

For a given word (e.g., ‘station’), how should the weights of the other words be chosen?

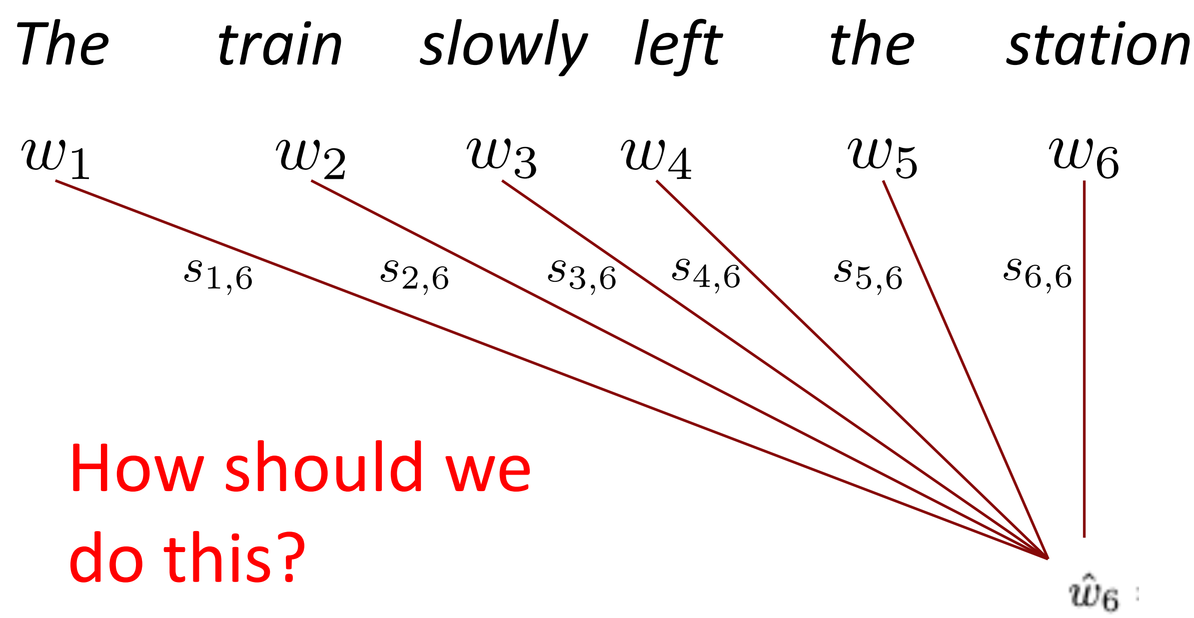
Intuition

- The weight of a word should be proportional to how related it is to the word “station”
- One way to quantify how “related” two words are: the *dot-product* of their stand-alone embeddings

How dot products measure “relatedness”

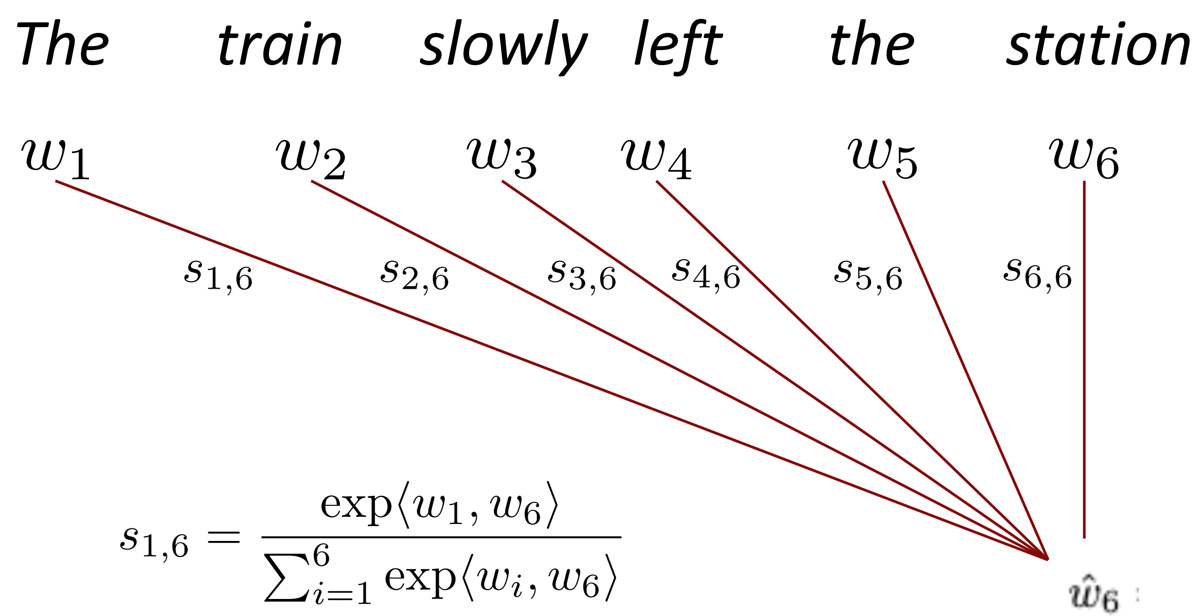
iPad

Dot-products between embeddings are a key ingredient but we need to do one more thing to make them proper* weights



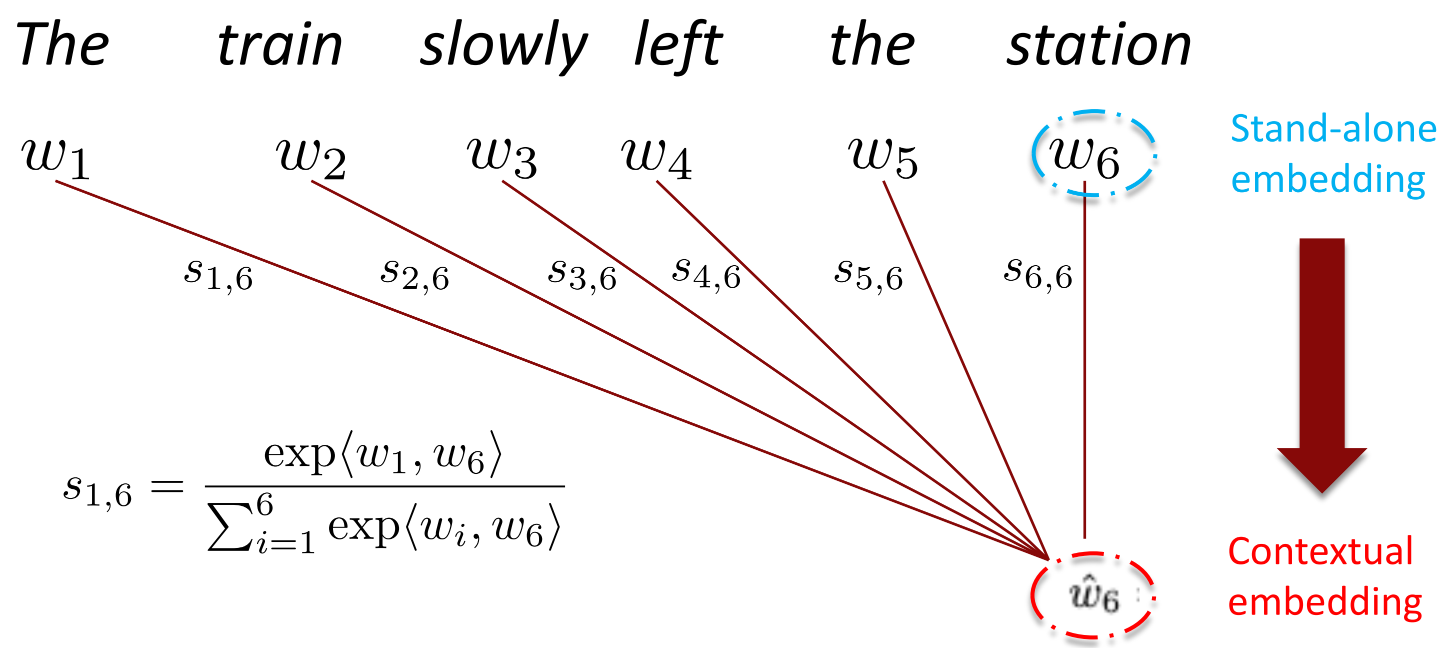
* non-negative, and summing to 1.0

Since dot-products can be negative, we can exponentiat them and then normalize (remember softmax?)



$$s_{1,6} = \frac{\exp\langle w_1, w_6 \rangle}{\sum_{i=1}^{6} \exp\langle w_i, w_6 \rangle}$$

Summary: From embeddings to *contextual* embeddings



$$s_{1,6} = \frac{\exp\langle w_1, w_6\rangle}{\sum_{i=1}^{6} \exp\langle w_i, w_6\rangle}$$

$$\hat{w}_6 = s_{1,6}w_1 + s_{2,6}w_2 + s_{3,6}w_3 + s_{4,6}w_4 + s_{5,6}w_5 + s_{6,6}w_6$$

By choosing weights in this manner, the embedding of a word moves closer to the embeddings of the other words in the current context, in proportion to how related they are

- The word ‘station’ has many contexts.

By choosing weights in this manner, the embedding of a word moves closer to the embeddings of the other words in the current context, in proportion to how related they are

- The word ‘station’ has many contexts.
  - In the current context, ‘train’ is closely related to ‘station’ and therefore exerts a strong “pull” on it

By choosing weights in this manner, the embedding of a word moves closer to the embeddings of the other words in the current context, in proportion to how related they are

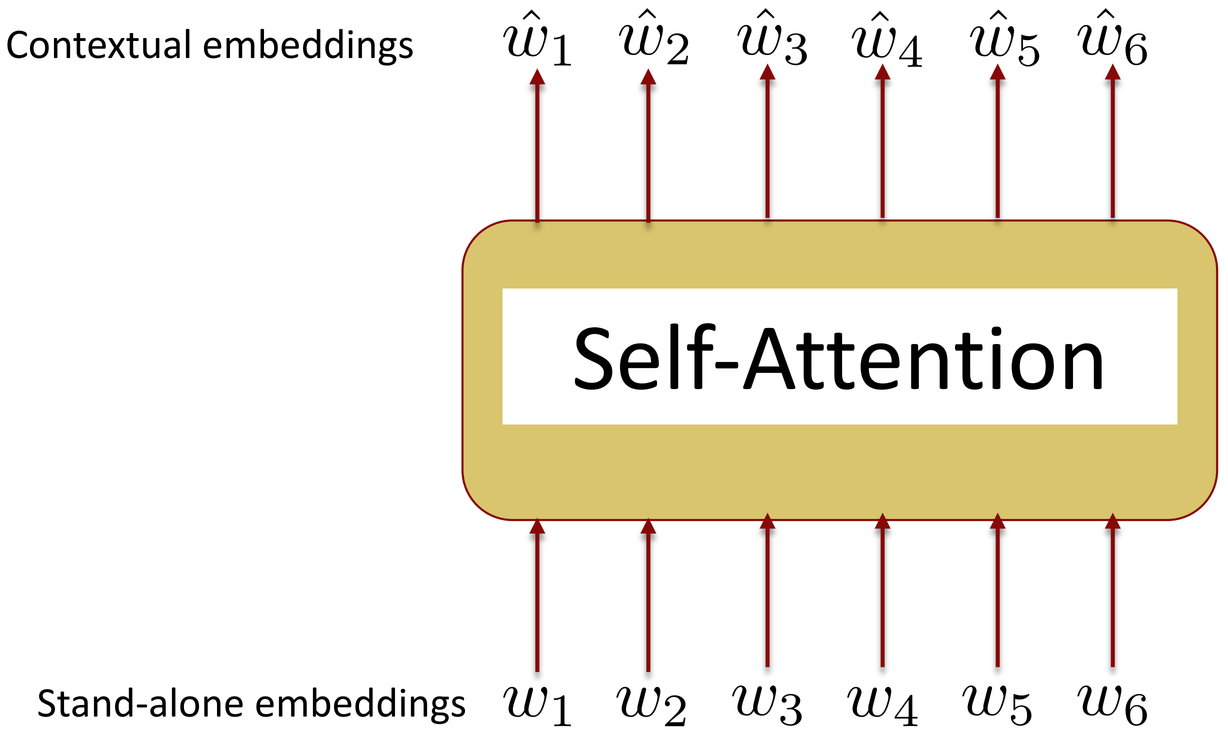
- The word ‘station’ has many contexts.
    - In the current context, ‘train’ is closely related to ‘station’ and therefore exerts a strong “pull” on it
    - ‘radio’ is also related to ‘station’ but doesn’t appear in the current context so (automatically) has zero weight

By choosing weights in this manner, the embedding of a word moves closer to the embeddings of the other words in the current context, in proportion to how related they are

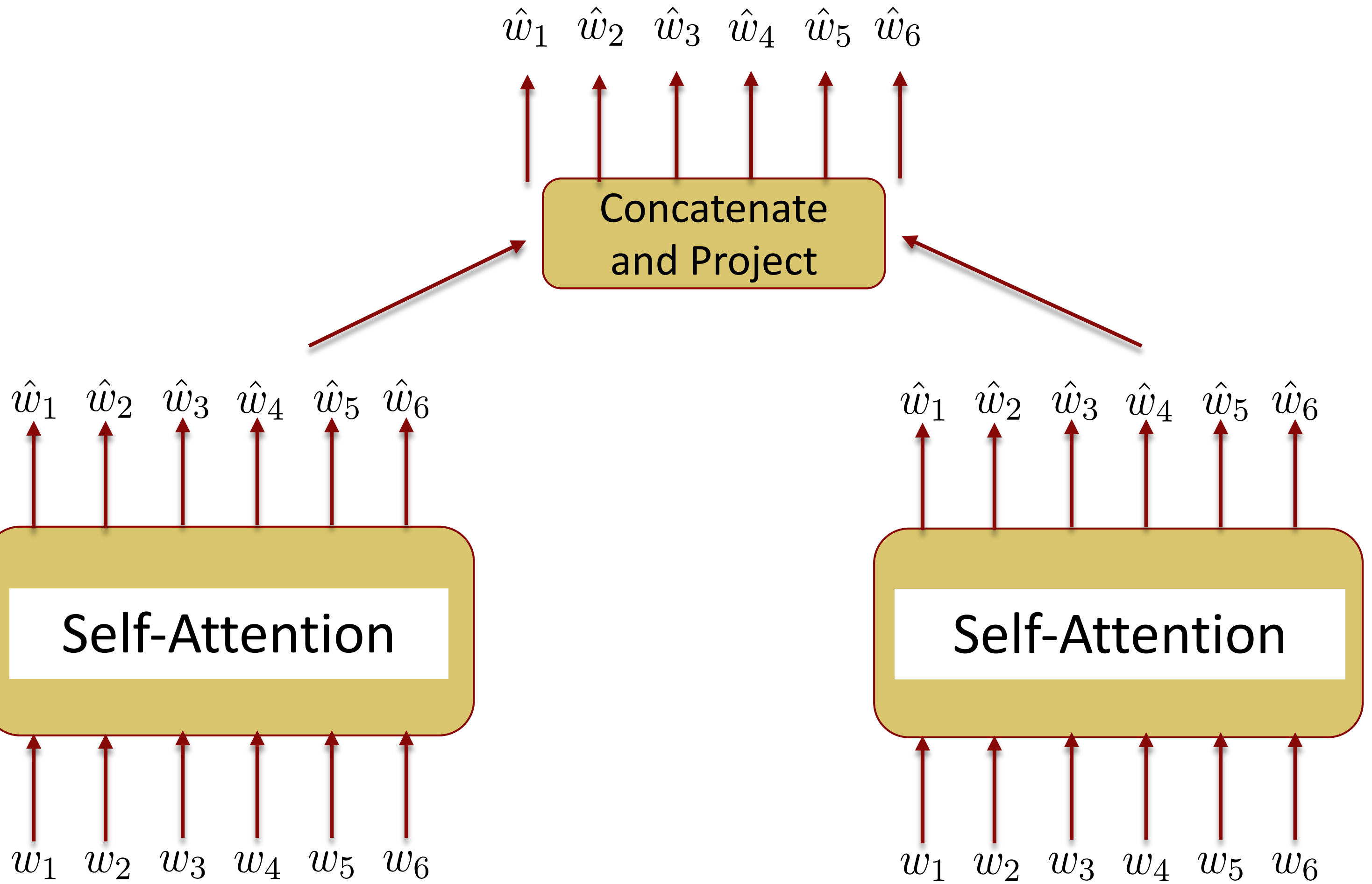
- The word ‘station’ has many contexts.
  - In the current context, ‘train’ is closely related to ‘station’ and therefore exerts a strong “pull” on it
  - ‘radio’ is also related to ‘station’ but doesn’t appear in the current context so (automatically) has zero weight

- By moving station closer to train (equivalently – paying more “attention” to train), we are contextualizing station’s embedding to the context of trains, platforms, departures, etc.

This operation is referred to as a 'Self Attention' layer and can be done very efficiently with matrix operations



Key Tweak: Multi*-Head Attention

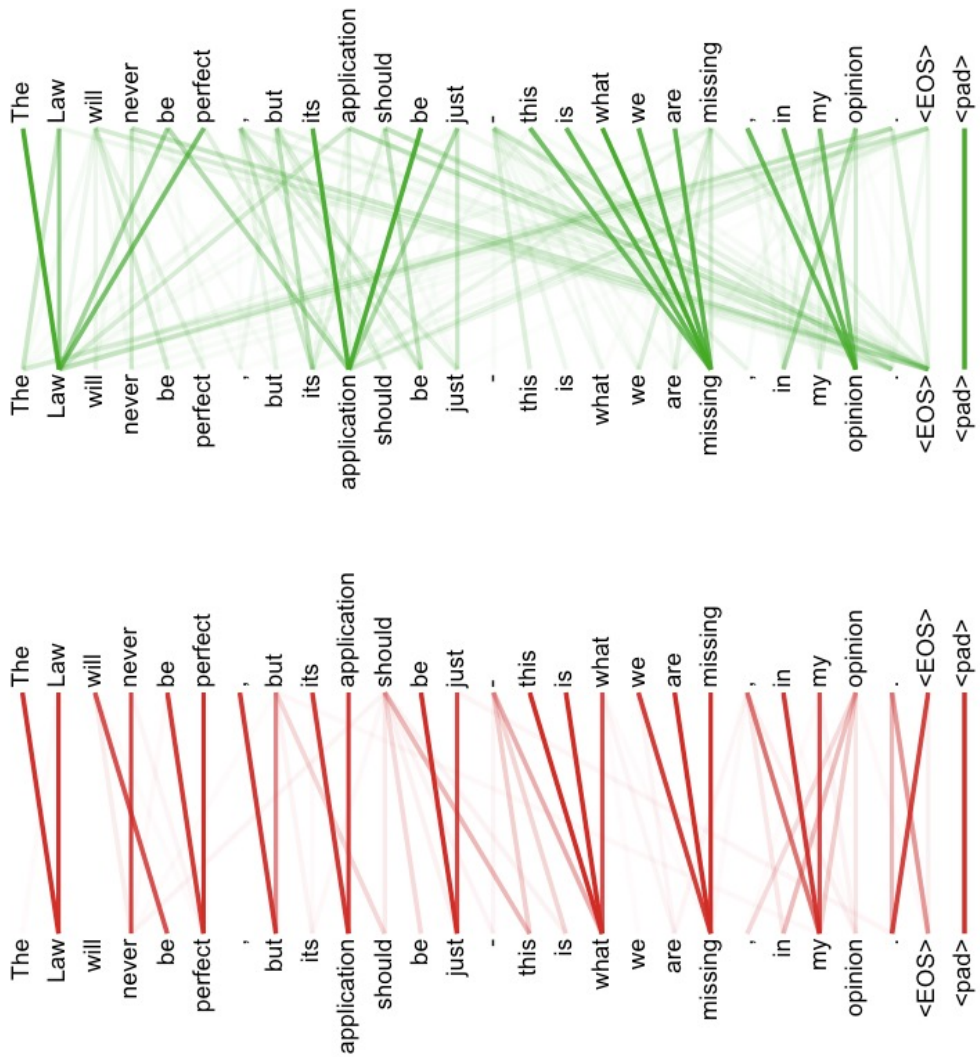


*Similar to having multiple filters in a convolutional layer

This helps us “attend to” the multiple patterns that may be present in a single sentence

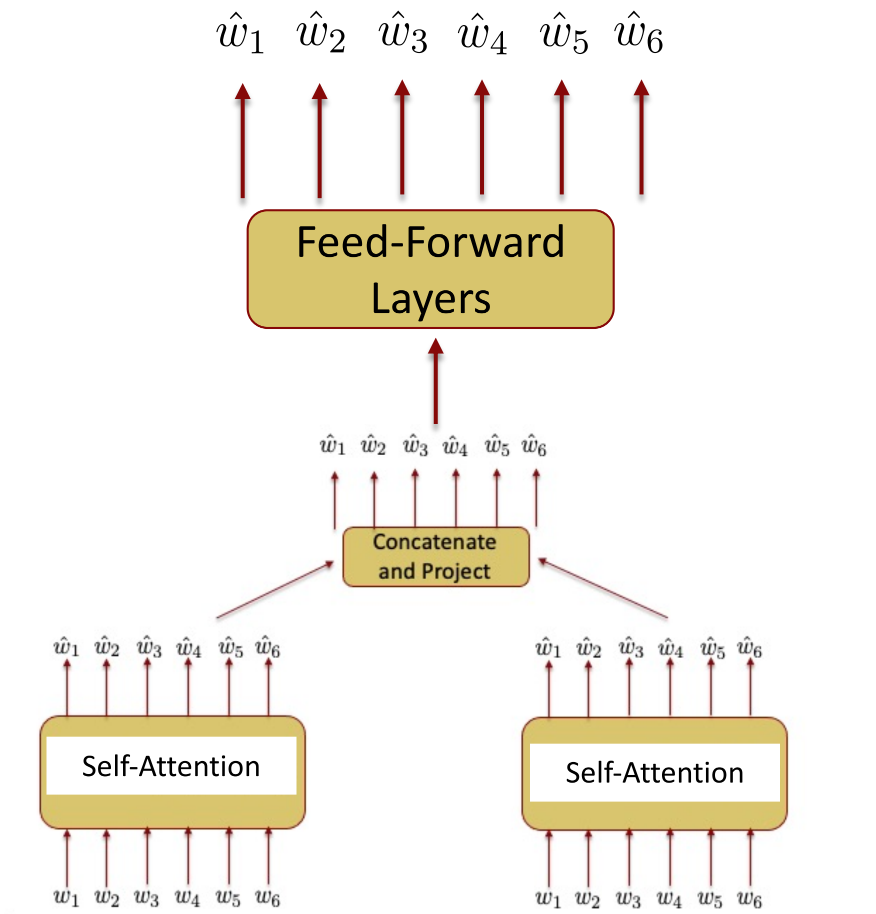
- Patterns related to “tense”
- Patterns related to “tone”
- Patterns related to the relationships between entities in the sentence
- …

Different attention ‘heads’ learn different patterns



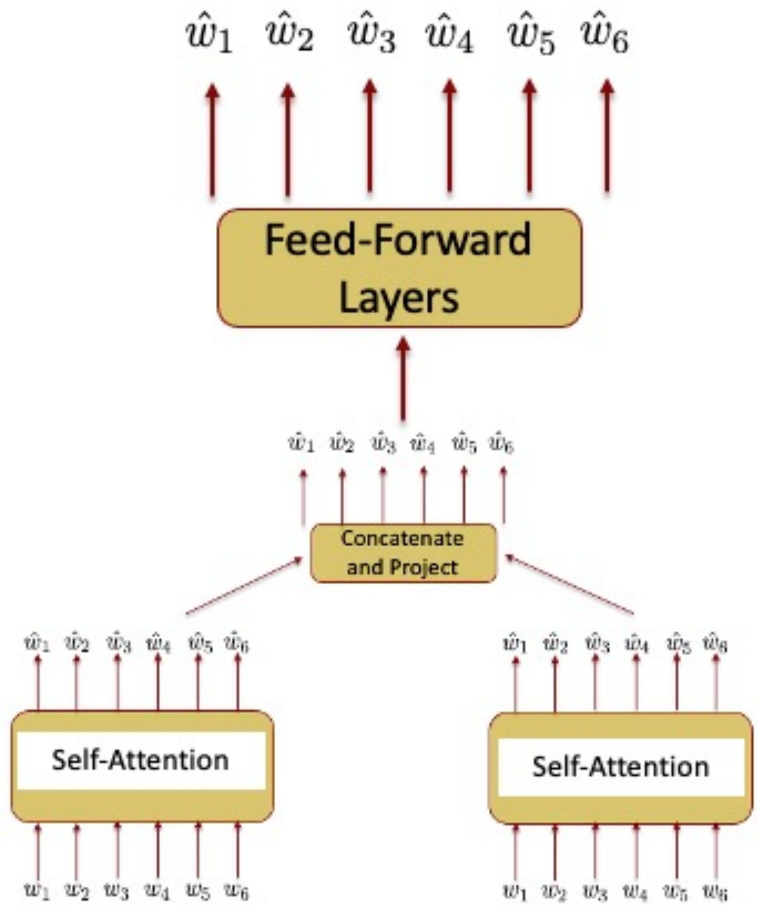
https://arxiv.org/abs/1706.03762

Key Tweak: Inject some non-linearity with feed-forward layers at the end



The story so far

End with contextual embeddings



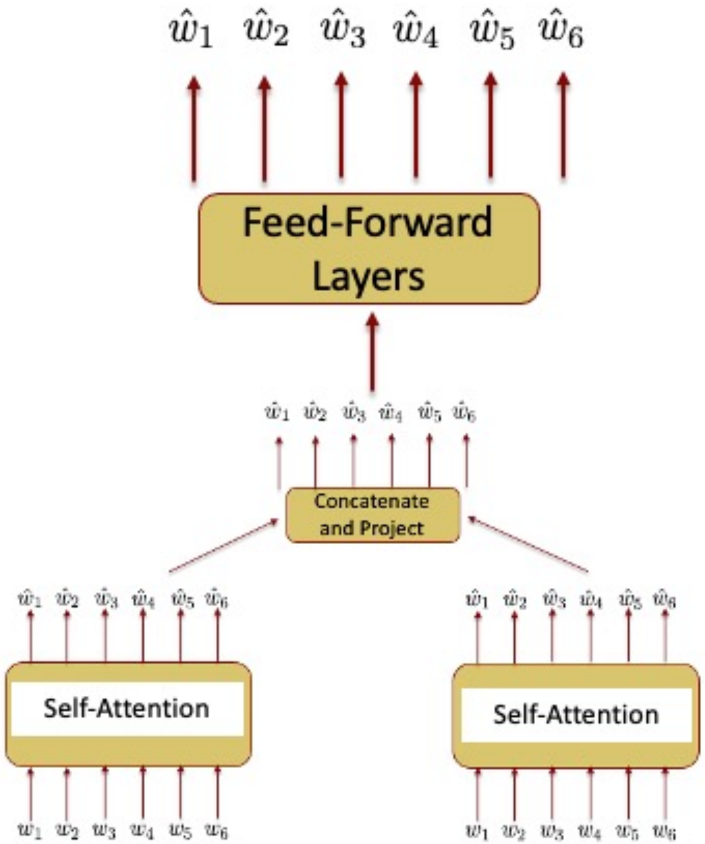
Start with random embeddings

We have satisfied 2 of the 3 requirements

✓Takes the surrounding context of each word into account

? Takes the order of the words into account

✓Can generate an output that has the same length as the input

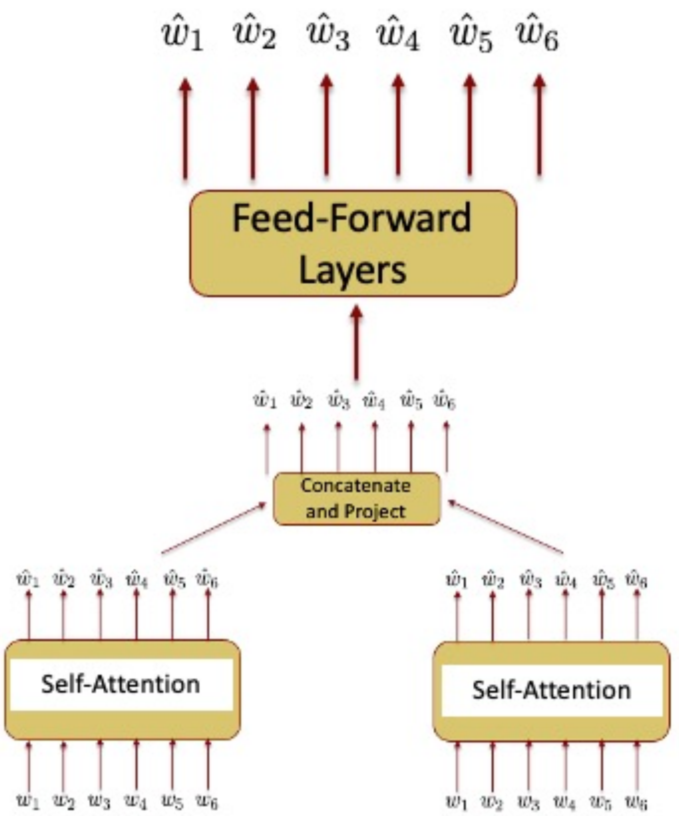


Does this architecture take word order into account?

✓Takes the surrounding context of each word into account

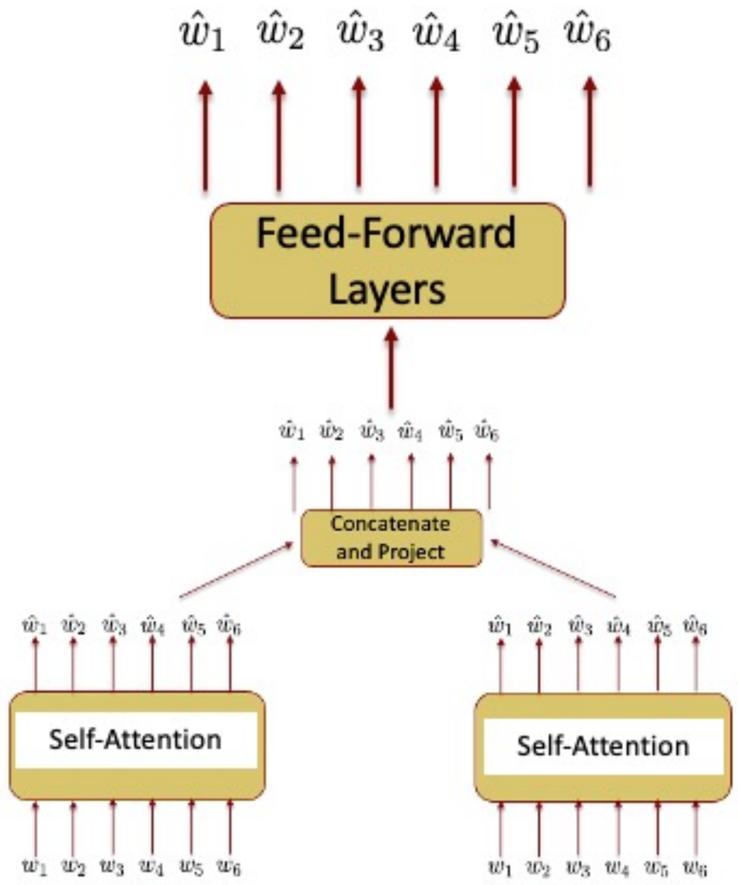
? Takes the order of the words into account

✓Can generate an output that has the same length as the input



The architecture does <u>not</u> take word order into account

We can scramble the order of the words in a sentence, and we'd get the exact same contextual embeddings at the end.



The Fix: Positional Encoding

- Add each word's position in the sentence to its stand-alone embedding.

- Our input word embeddings will be the sum of two things:
  - the usual "stand-alone" embedding +
  - a *position embedding*, which represents the position of the word in the sentence.

Positional Encoding - Example

Stand-alone embedding

Word	Dimension 1	Dimension 2
[UNK]	6.1	-3.2
cat	0.5	7.1
mat	-2	-3.1
I	0.1	3.4
sit	1.2	5.3
love	6.1	7.2
the	0.1	0.1
you	5.0	3.2
on	2.0	4.1

Position embedding

Position	Dimension 1	Dimension 2
0	1.3	3.9
1	6.3	3.7
2	0.6	8.1
3	-2.3	-4.1
4	0.14	5.4
5	1.29	3.3
6	6.12	-2.2
7	0.11	2.1
8	5.03	-5.2
9	2.05	-4.1

Positional Encoding - Example

Stand-alone embedding

Word		
[UNK]	6.1	-3.2
cat	0.5	7.1
mat	-2.0	-3.1
I	0.1	3.4
sat	1.2	5.3
love	6.1	7.2
the	0.1	0.1
you	5.0	3.2
on	2.0	4.1

Position embedding

Position		
0	1.3	3.9
1	6.3	3.7
2	0.6	8.1
3	-2.3	-4.1
4	0.14	5.4
5	1.29	3.3
6	6.12	-2.2
7	0.11	2.1
8	5.03	-5.2
9	2.05	-4.1

"cat sat mat"

cat	0.5	7.1
0	1.3	3.9
sum	1.8	11.0

Positional Encoding - Example

Stand-alone embedding

Word		
[UNK]	6.1	-3.2
cat	0.5	7.1
mat	-2.0	-3.1
I	0.1	3.4
sat	1.2	5.3
love	6.1	7.2
the	0.1	0.1
you	5.0	3.2
on	2.0	4.1

Position embedding

Position		
0	1.3	3.9
1	6.3	3.7
2	0.6	8.1
3	-2.3	-4.1
4	0.14	5.4
5	1.29	3.3
6	6.12	-2.2
7	0.11	2.1
8	5.03	-5.2
9	2.05	-4.1

“cat sat mat”

cat	0.5	7.1
0	1.3	3.9
sum	1.8	11.0

sat	1.2	5.3
1	6.3	3.7
sum	7.5	9.0

mat	-2.0	-3.1
2	0.6	8.1
sum	-1.4	5.0

Positional Encoding - Example

Stand-alone embedding

Word		
[UNK]	6.1	-3.2
cat	0.5	7.1
mat	-2.0	-3.1
I	0.1	3.4
sat	1.2	5.3
love	6.1	7.2
the	0.1	0.1
you	5.0	3.2
on	2.0	4.1

Position embedding

Position		
0	1.3	3.9
1	6.3	3.7
2	0.6	8.1
3	-2.3	-4.1
4	0.14	5.4
5	1.29	3.3
6	6.12	-2.2
7	0.11	2.1
8	5.03	-5.2
9	2.05	-4.1

"cat sat mat"

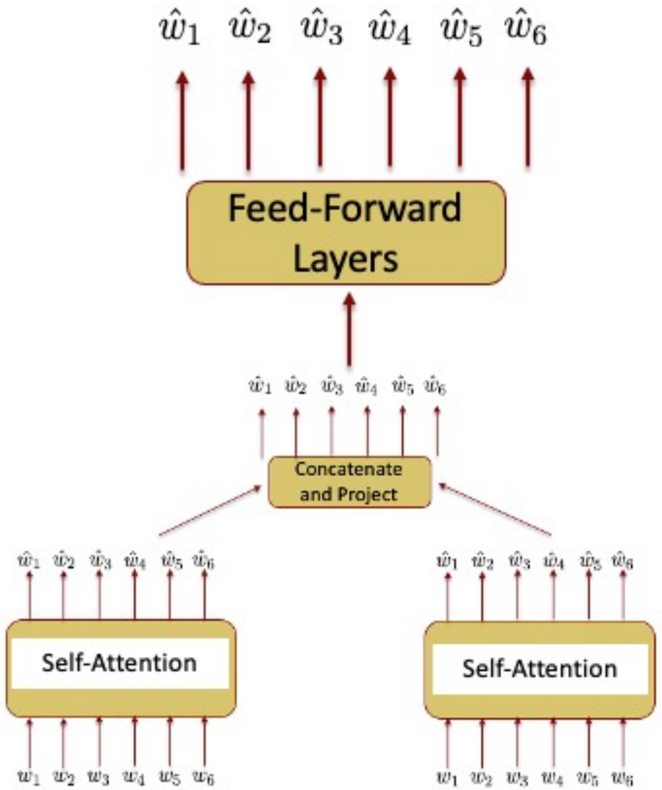
cat	0.5	7.1
0	1.3	3.9
sum	1.8	11.0

sat	1.2	5.3
1	6.3	3.7
sum	7.5	9.0

mat	-2.0	-3.1
2	0.6	8.1
sum	-1.4	5.0

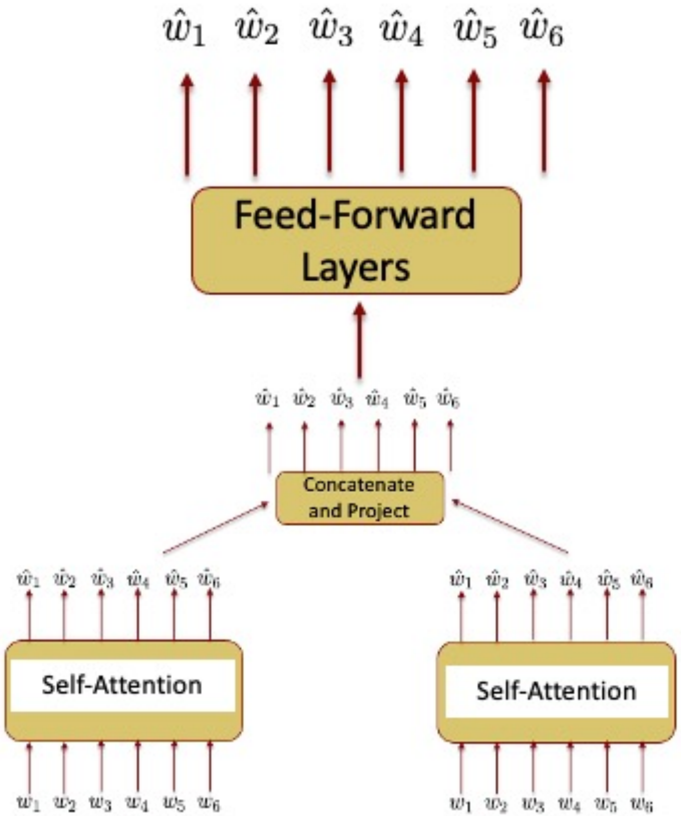
We have satisfied all 3 requirements

- ✓ Takes the surrounding context of each word into account
- ✓ Takes the order of the words into account
- ✓ Can generate an output that has the same length as the input



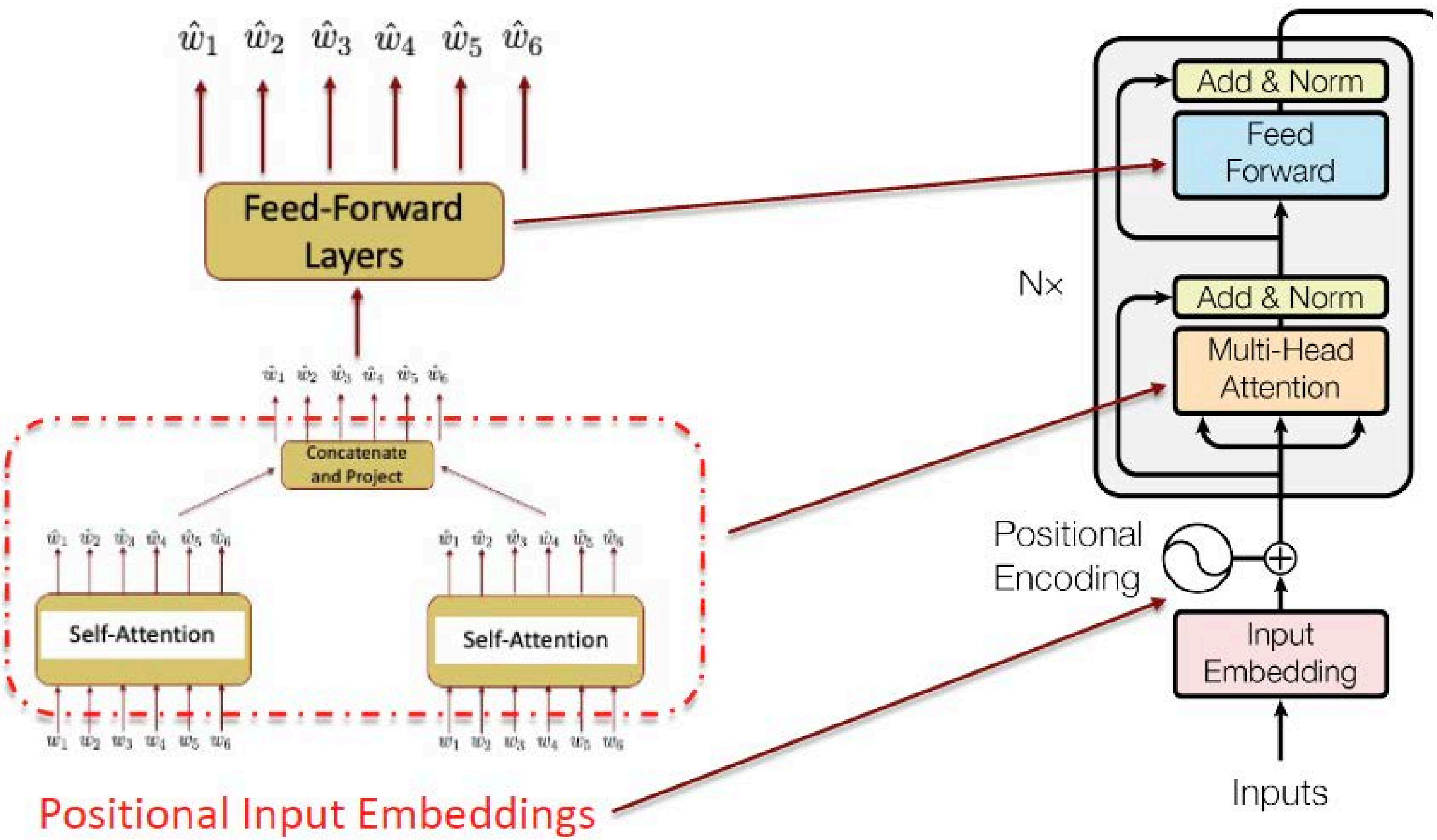
Positional Input Embeddings

This is called a Transformer Encoder



Positional Input Embeddings

This is called a Transformer Encoder*



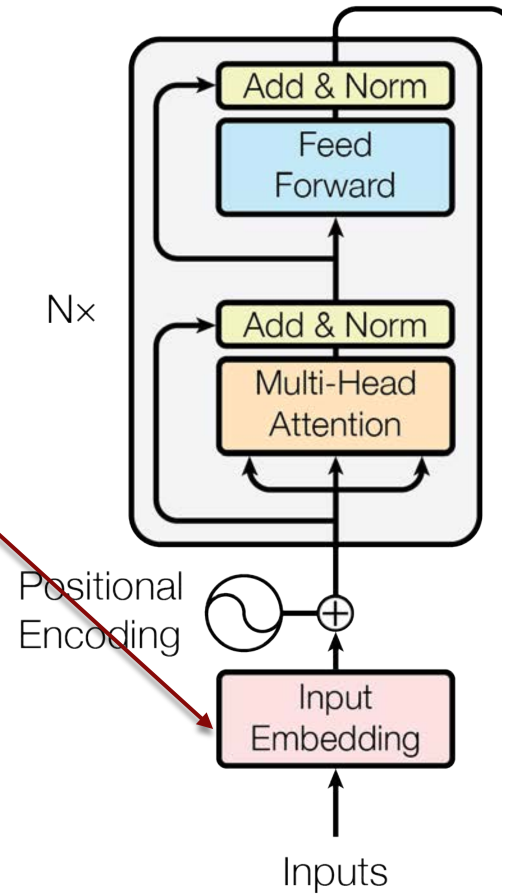
*with layernorm and residual connections (details in the next lecture)

https://arxiv.org/abs/1706.03762

This is called a Transformer Encoder

Summary

- The input embedding can simply be random embeddings or pretrained

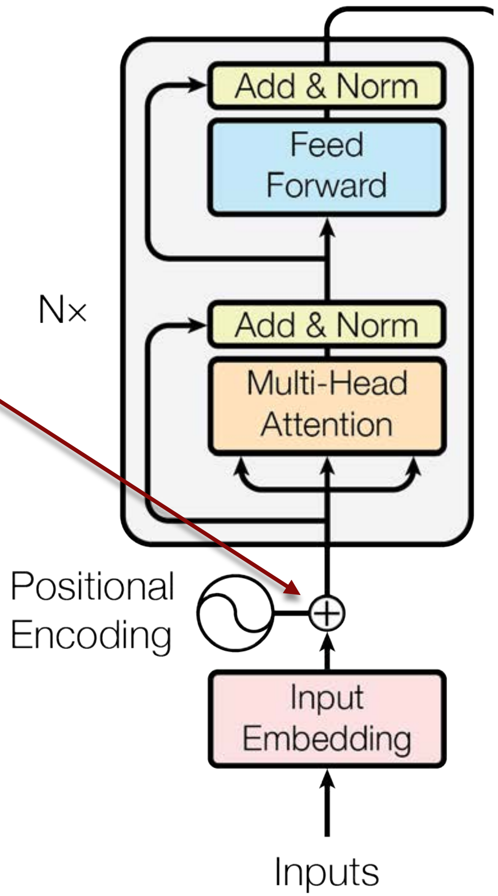


https://arxiv.org/abs/1706.03762

This is called a Transformer Encoder

Summary

- The input embedding can simply be random embeddings or pretrained
- Add in a position dependent embedding to represent the position of each word in sentence

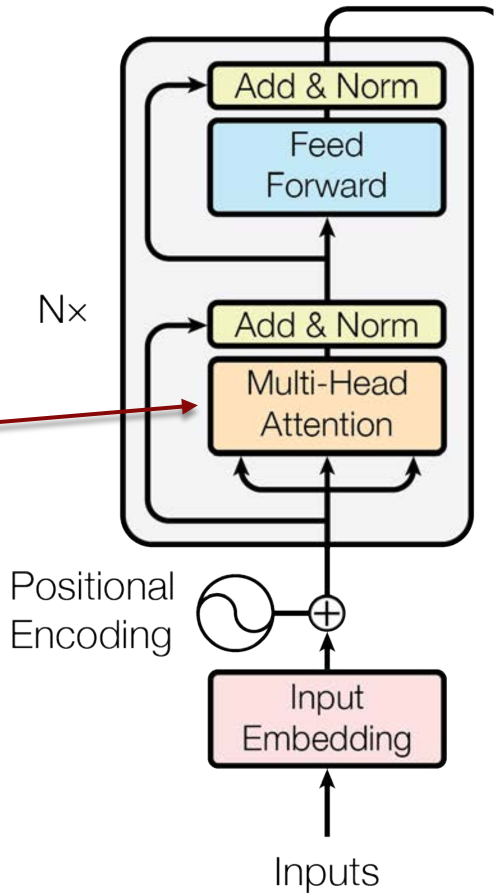


https://arxiv.org/abs/1706.03762

This is called a Transformer Encoder

Summary

- The input embedding can simply be random embeddings or pretrained
- Add in a position dependent embedding to represent the position of each word in sentence
- Pass through multi-headed attention to get a context dependent representation

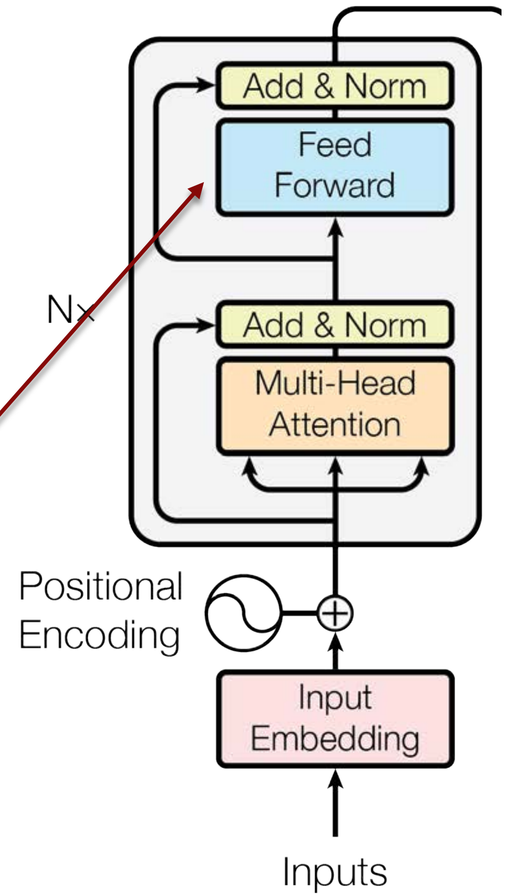


https://arxiv.org/abs/1706.03762

This is called a Transformer Encoder

Summary

- The input embedding can simply be random embeddings or pretrained
- Add in a position dependent embedding to represent the position of each word in sentence
- Pass through multi-headed attention to get a context dependent representation
- Finally, pass all this through a simple feed-forward network

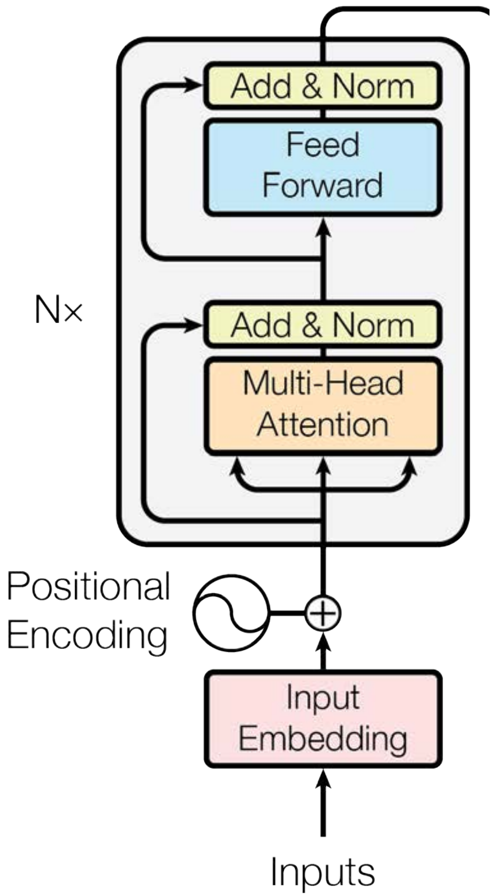


https://arxiv.org/abs/1706.03762

This is called a Transformer Encoder

Summary

- The input embedding can simply be random embeddings or pretrained
- Add in a position dependent embedding to represent the position of each word in sentence
- Pass through multi-headed attention to get a context dependent representation
- Finally, pass all this through a simple feed-forward network
- *Since encoders have the same-sized inputs and outputs, they can be daisy-chained (i.e., stacked) to get more modeling capacity*



https://arxiv.org/abs/1706.03762

Elements of the Transformer Encoder that will be covered in the next lecture*

- Linear projections of the incoming embeddings into three different spaces before the self-attention operation is carried out

- Residual connections

- Layer normalization

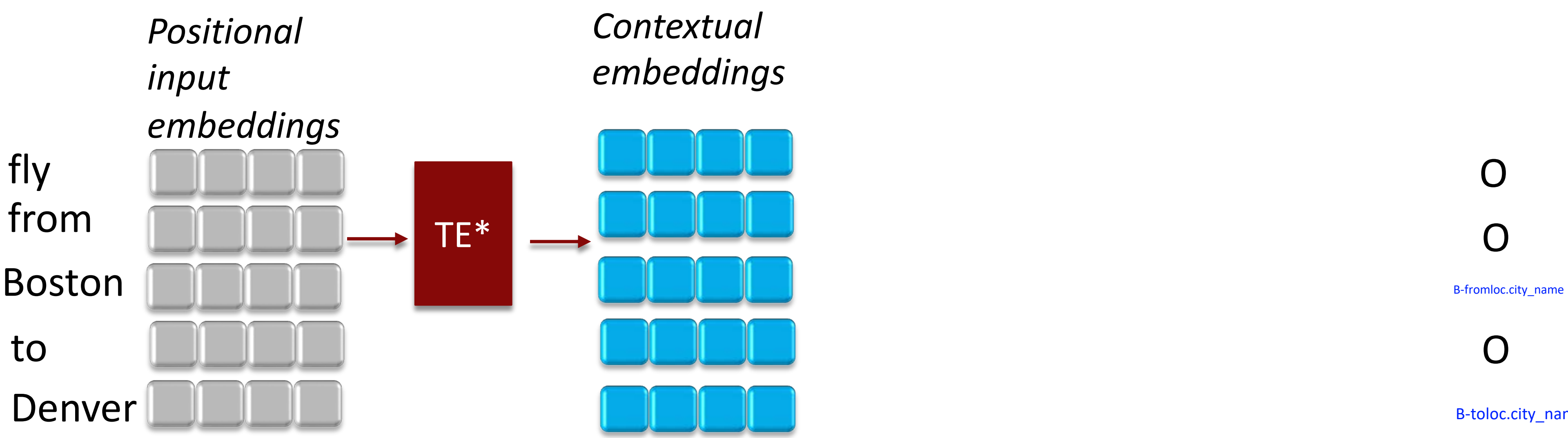
*please see the textbook if you can't stand the suspense 🙂

Let’s apply a Transformer Encoder to the word-to-slot problem!

Slot Filling with Transformers

fly	O
from	O
Boston	B-fromloc.city_name
to	O
Denver	B-toloc.city_nar

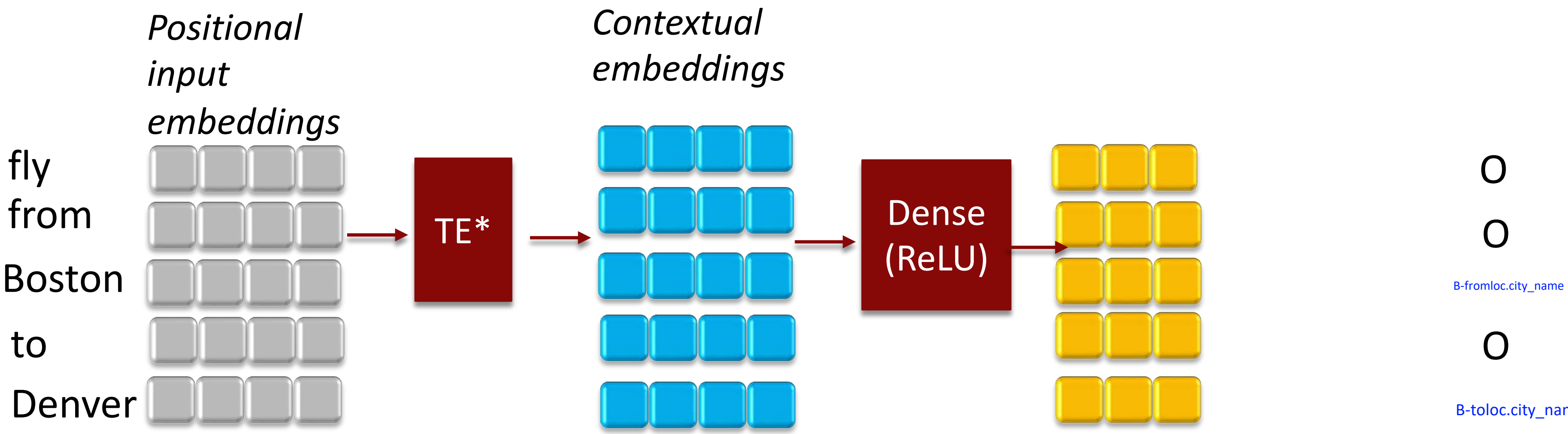
Slot Filling with Transformers



*Transformer Encoder

Indicate 4-element embedding vectors

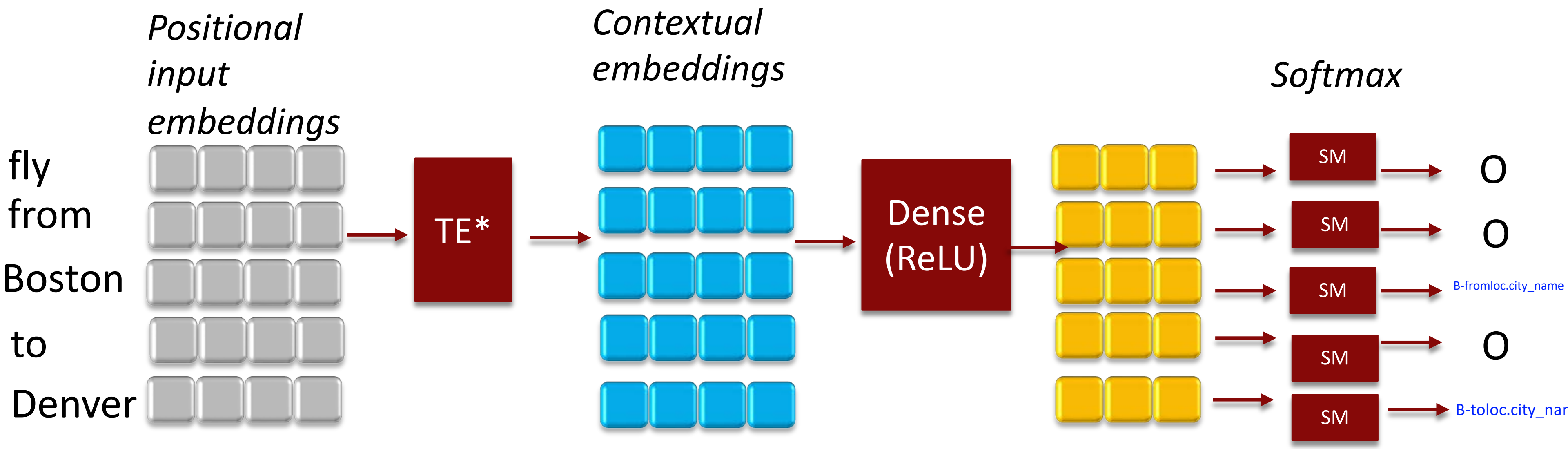
Slot Filling with Transformers



*Transformer Encoder

Indicate 4-element embedding vectors

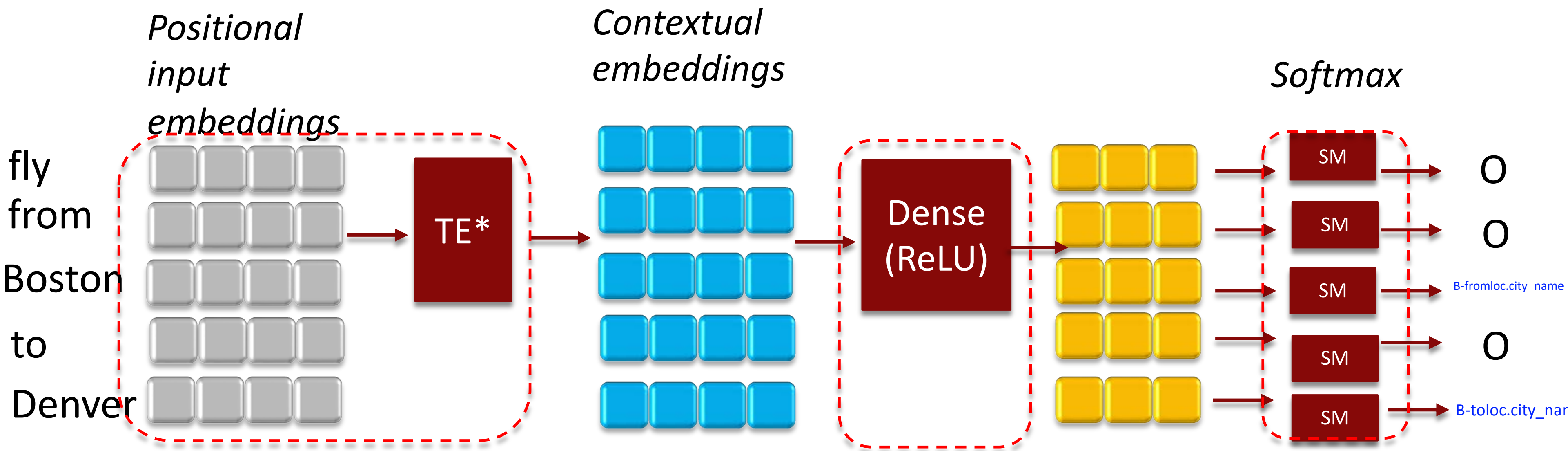
Slot Filling with Transformers



*Transformer Encoder

Indicate 4-element embedding vectors

Slot Filling with Transformers



The weights in all these layers will get optimized by backprop

*Transformer Encoder

Indicate 4-element embedding vectors

Colab

Link to colab