



BERT Transformer Architecture for NLP

Introduction to Natural Language
Processing with Next-Generation
Transformer Architectures

Agenda

Segment 0: Introduction

Segment 1: Introduction to Transformers and Language Models (30min + 15min Q/A break)

Segment 2: Introduction to BERT (60min + 15min Q/A break)

Segment 3: Fine-tuning BERT to perform NLP tasks (75min + 15min Q/A break)

Segment 4: Course wrap-up and next steps (30min + final Q/A)

Your Instructor

My name is **Sinan Ozdemir** (in/sinan-ozdemir + @prof_oz)

Director of Data Science at Directly in SF / Masters in Pure Math / Former lecturer of Business Analytics and CS at Johns Hopkins

Founder of [Kylie.ai](#) which created auto-generated chatbots (Acquired)

Author of several textbooks, including [The Principles of Data Science](#)



Our Goal for Today

Our goal is to help you understand

1. What BERT is
2. How BERT works
3. How to use BERT to perform several NLP tasks

Our Journey Today

BERT is derived from the

Transformer architecture which relies on the

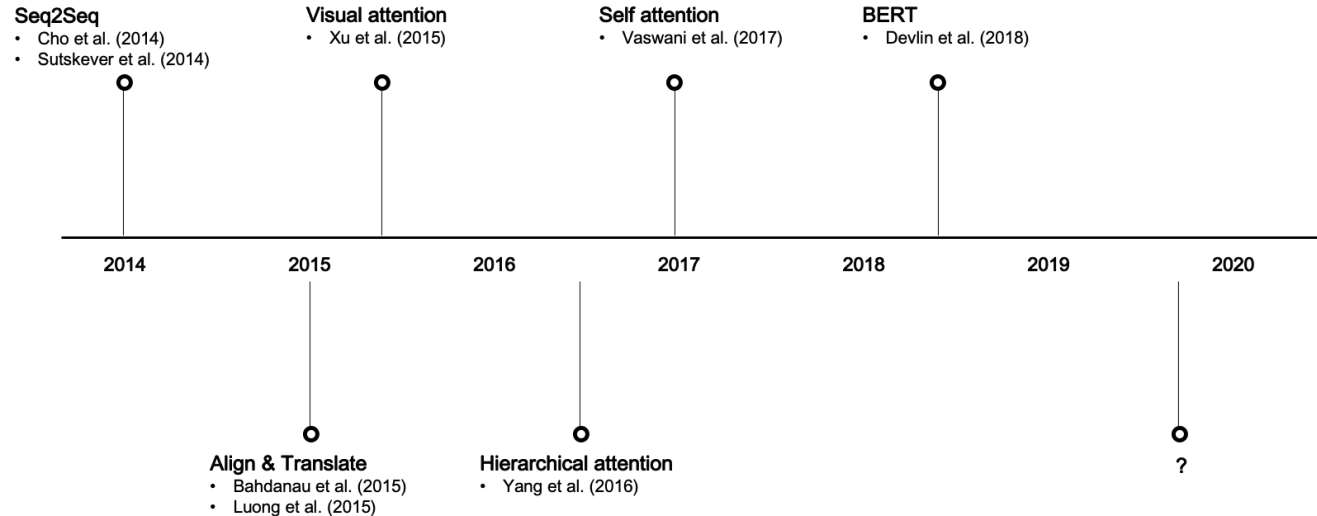
Attention mechanism



Introduction to Attention, Language Models, and Transformers

Attention in Neural Networks

The transformer model is based primarily on the idea of **attention** - a family of mechanisms designed to “attend to” parts of a sequence in the context of another sequence usually for the purpose of some prediction task



Attention Example - Image Captioning

Figure 4. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



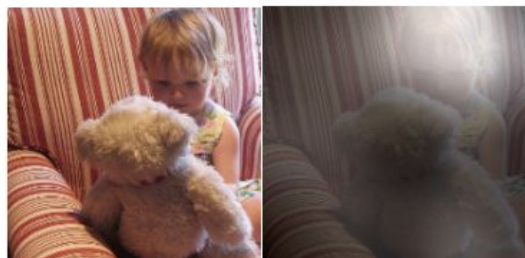
A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



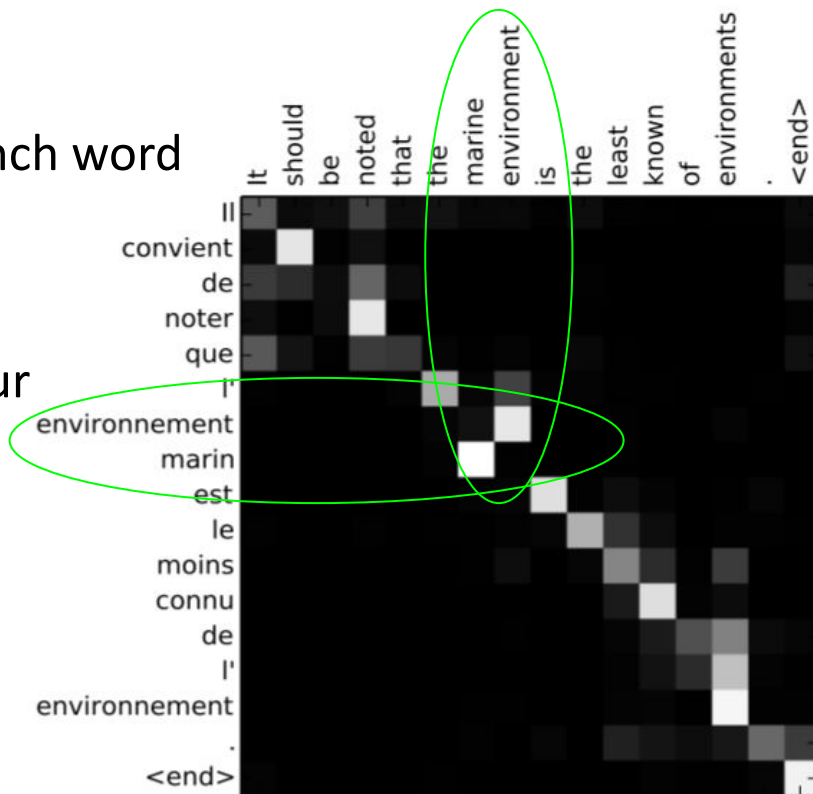
A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

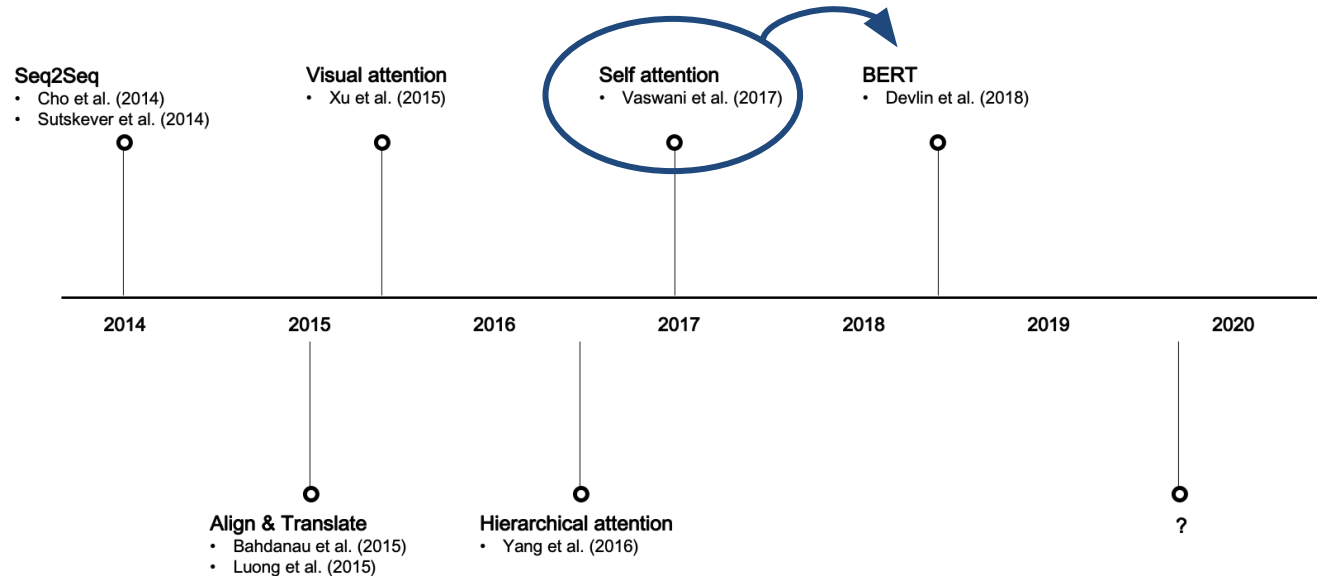
Attention Example - Language Translation

When translating English to French, we see that the French word **marin** is paying attention to the English word **marine** even though they don't occur at the same index



Attention in Neural Networks

Let's turn our focus to self-attention



Self-Attention

Consider the following phrase:

My friend told me about this class and I
love it so far! She was right.

- The pronoun “**I**” refers to the antecedent “**me**”
- The pronoun “**She**” refers to the antecedent “**friend**”

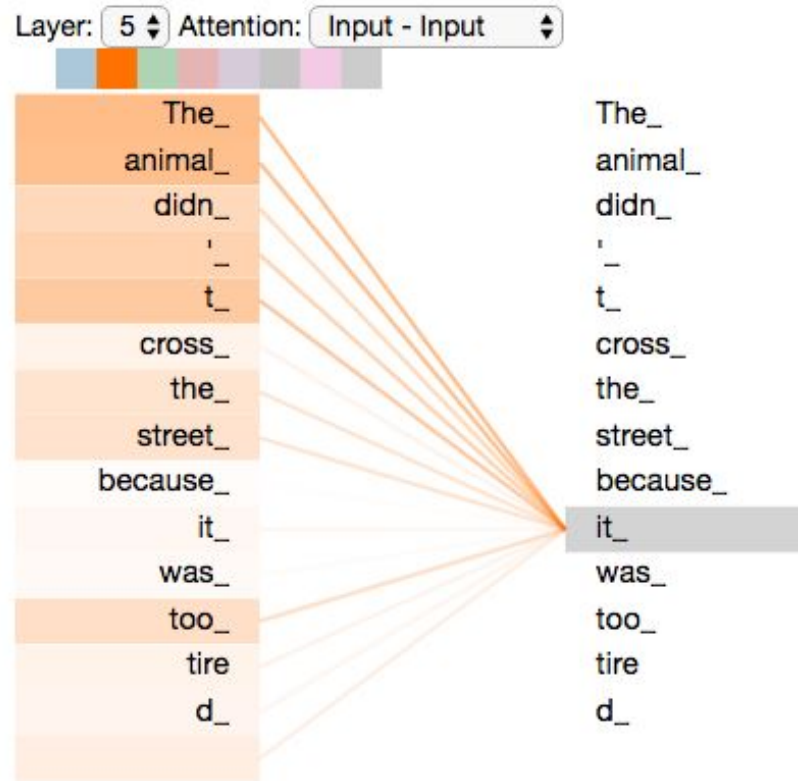
Self-Attention

Consider the following phrase:

My friend told me about this class and I
love it so far! She was right.

With self-attention, we alter the idea of attention to relate each word in the phrase with every other word in the phrase to teach the model the relationships between them

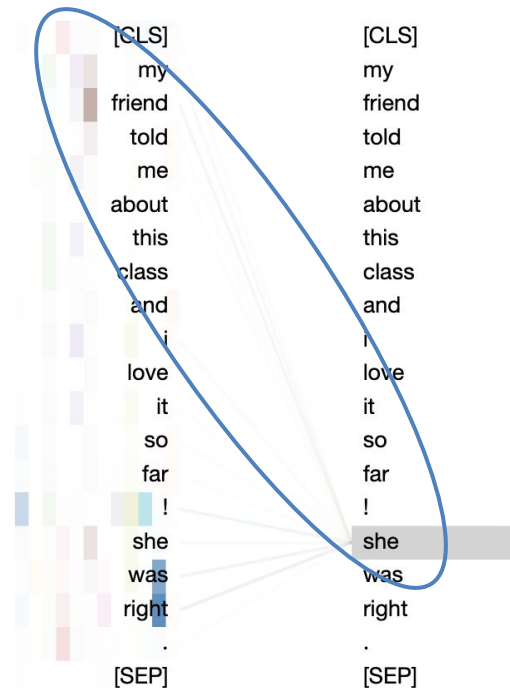
Self-Attention



Self-Attention

Consider the following phrase:

My **friend** told
me about this
class and **I**
love it so far!
She was right.



Attention in Neural Networks

We will re-visit attention in much more detail in a moment

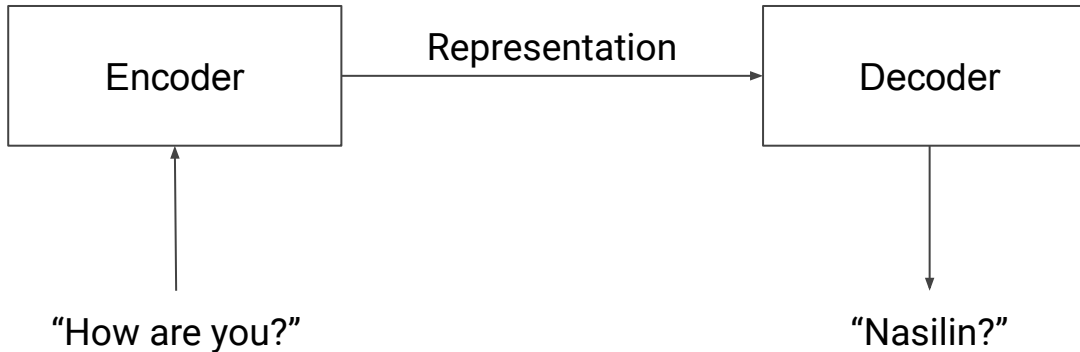
For now let's move on to the Transformer.

Transformers

The **Transformer** is a current state-of-the-art NLP model. It relies almost entirely on self-attention to model the relationship between tokens in a sentence rather than relying on recursion like RNNs and LSTMs do

Transformers

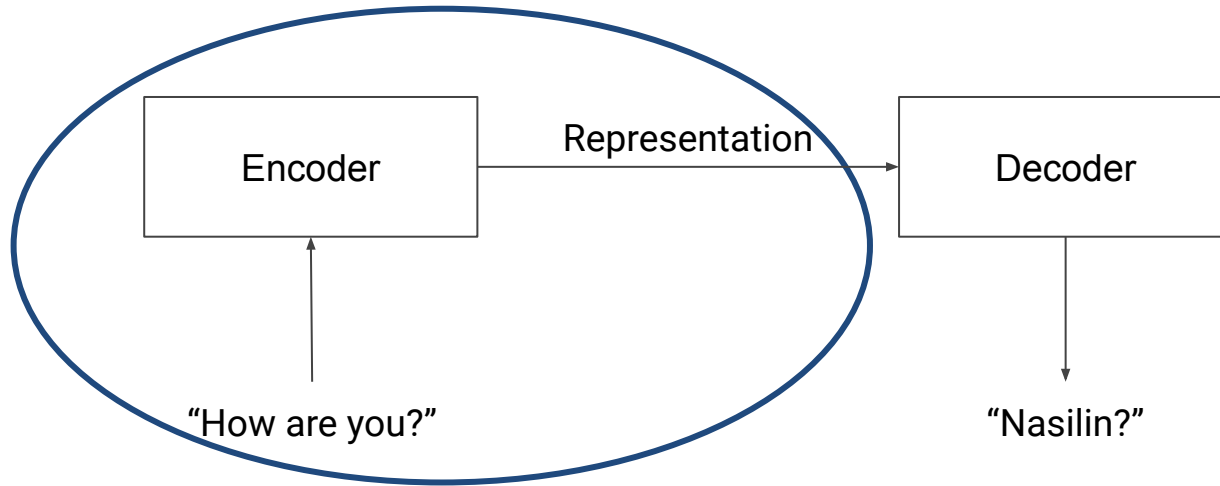
Transformers consists of an encoder and a decoder. The encoder takes in our **input** and outputs a matrix **representation** of that input. The decoder takes in that representation and iteratively generates an **output**



English -> Turkish
Language Translation
using a Transformer

Transformers

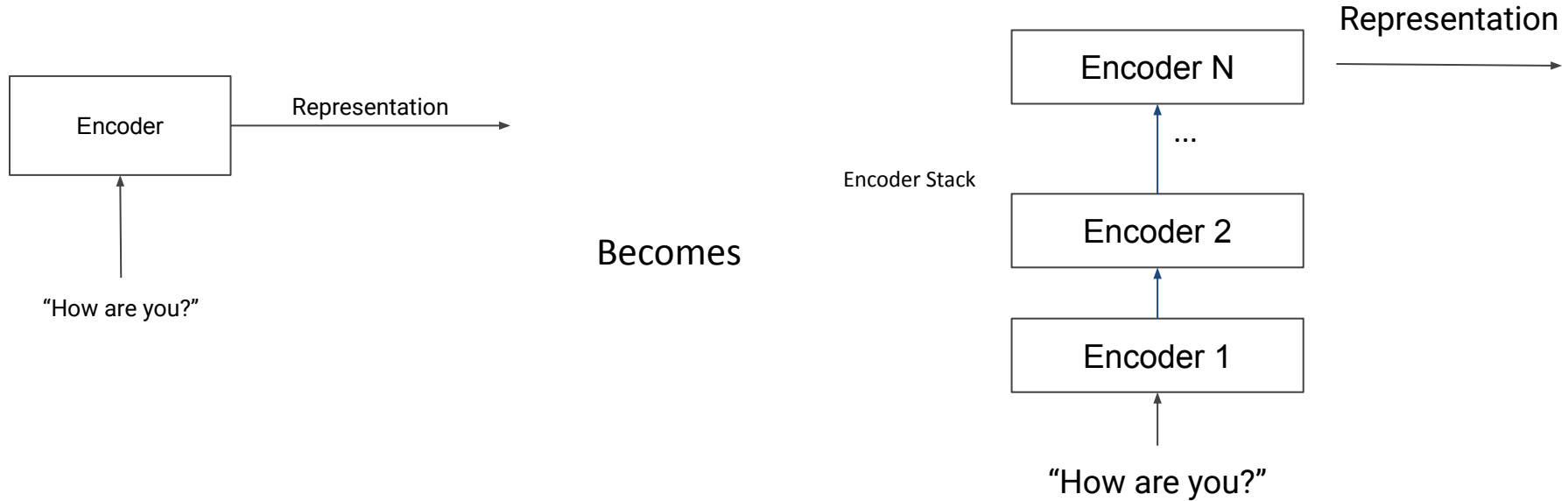
Let's zoom in on the encoder and the representation



English -> Turkish
Language Translation
using a Transformer

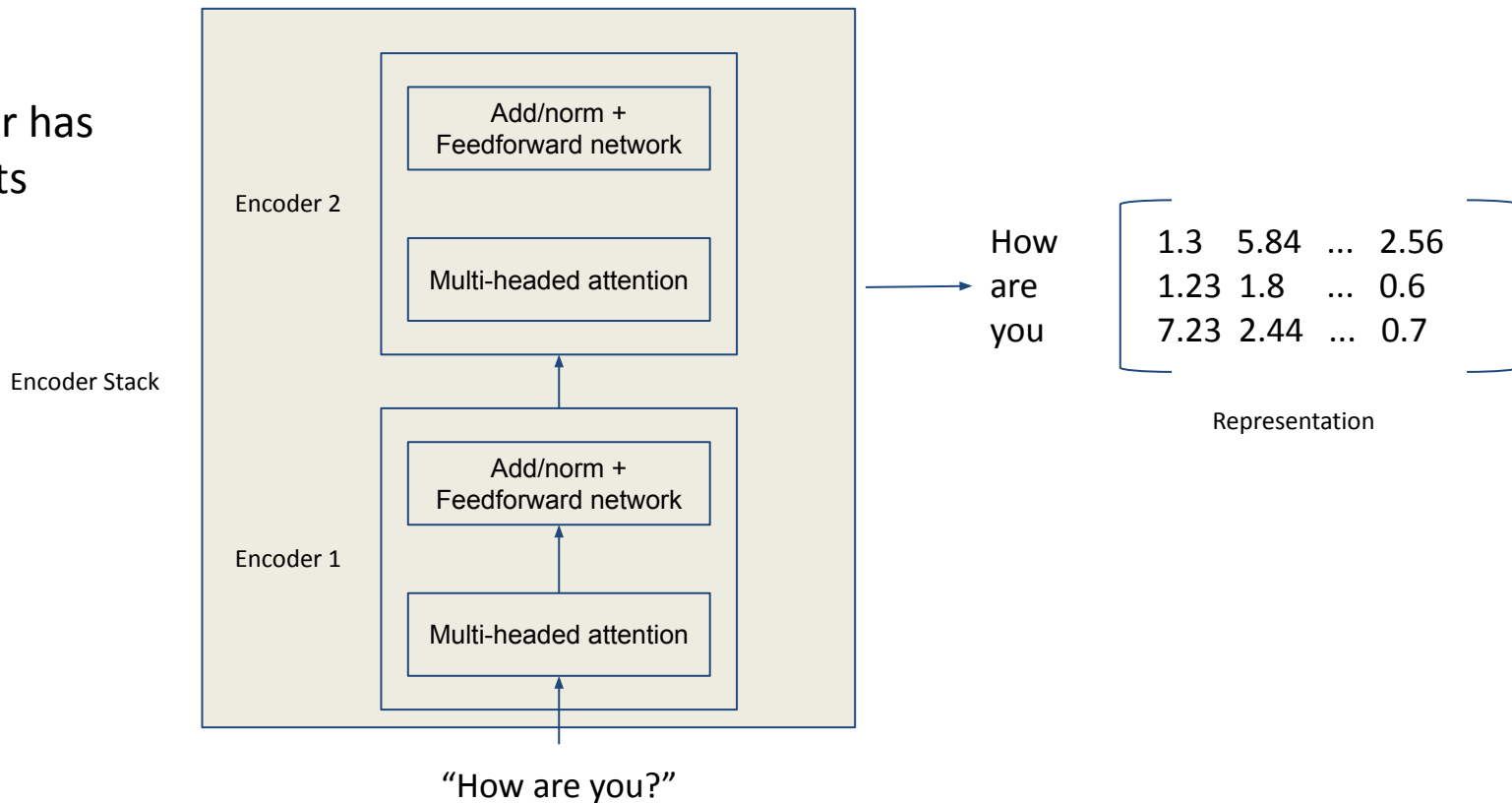
Transformers

The encoder is actually an **encoder stack** with multiple encoders



Transformers

Each encoder has multiple parts



Transformers

From the original
Transformer paper

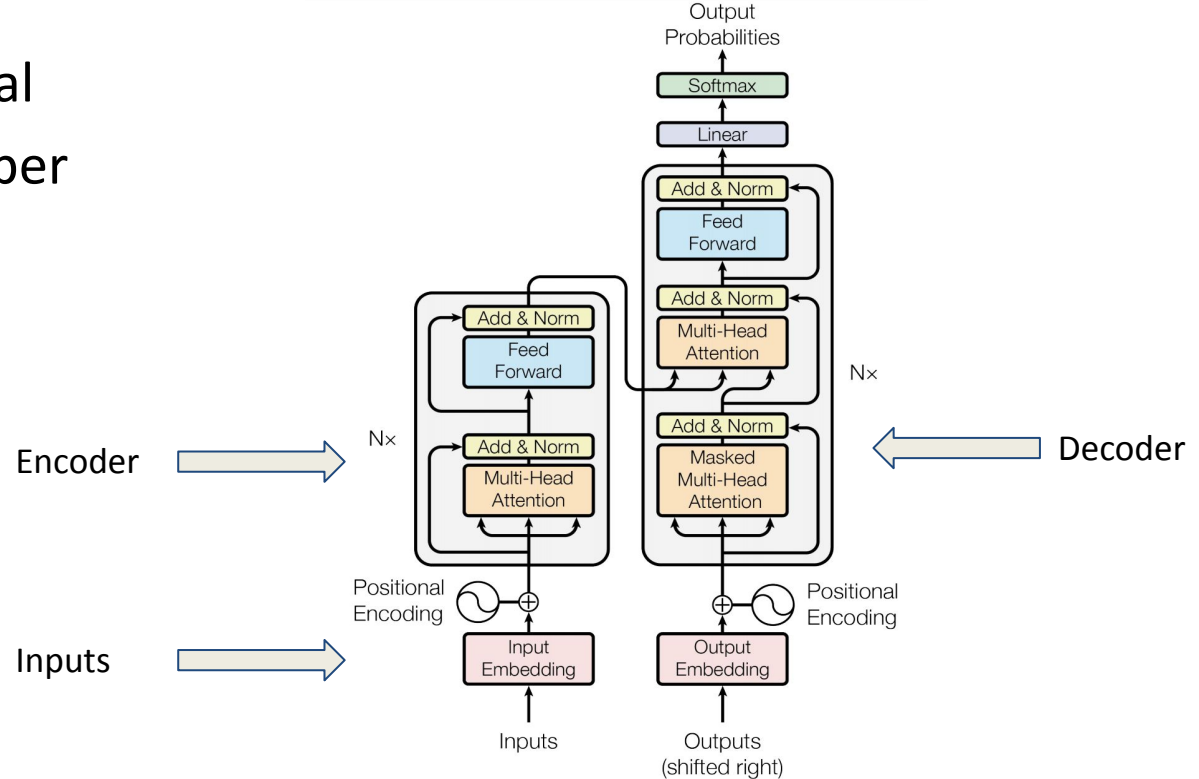


Figure 1: The Transformer - model architecture.

Language Models

In a **language modeling** task, A model is trained to predict a missing word in a sequence of words. In general, there are two types of language models:

1. Auto-regressive
2. Auto-encoding

Language Models

Consider the following example:

If you don't ____ at the sign, you will get a ticket.

Language Models

Consider the following example:

If you don't ____ at the sign, you will get a ticket.



95%



5%

Auto-__ Language Models

Auto-regressive Models

1. Goal is to predict a future token (word) given either the past tokens or the future tokens but not both

If you don't ____ (forward prediction)

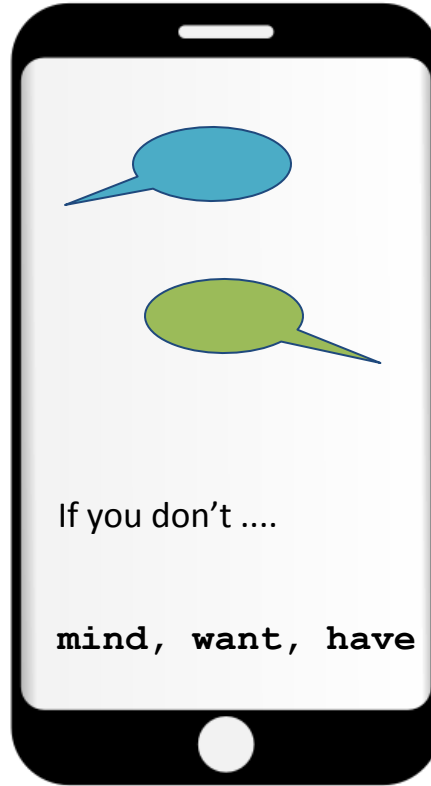
____ at the sign, you will get a ticket. (backward prediction)

Auto-encoding Models

1. Goal is to learn representations of the entire sequence by predicting tokens given both the past and future tokens

If you don't ____ at the sign, you will get a ticket.

Auto-regressive Use Case - word suggest



Auto-__ Language Model Use Cases

Auto-regressive Models

1. Predicting next word in a sentence (auto-complete)
2. Natural Language Generation (NLG)
3. GPT Family

Auto-encoding Models

1. Comprehensive understanding and encoding of entire sequences of tokens
2. Natural Language Understanding (NLU)
3. **BERT**

Recap - Attention + Transformers

Attention is a mechanism that relates sequences of tokens to other sequences of tokens

Self-attention is a special type of attention where both sequences are the same

Transformers are an encoder-decoder architecture that rely on self-attention to model relations in sequences to parse a sequence and output another sequence to produce a representation of an input sequence

What I skipped / glossed over

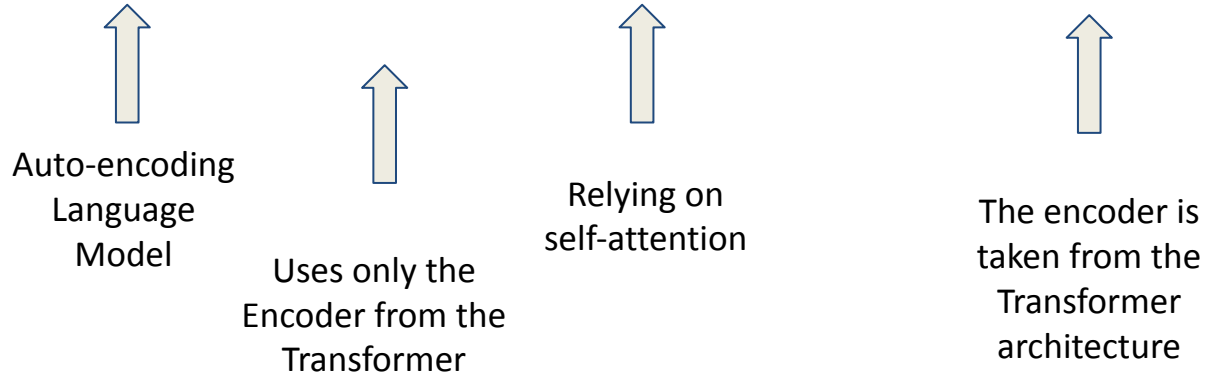
1. The Transformer's decoder. BERT doesn't use it so we won't focus on it
2. How the add/norm + dropout layers in the encoder stack function in depth

Q/A --> Break



Introduction to BERT

Bi-directional Encoder Representation from Transformers



BERT at a distance

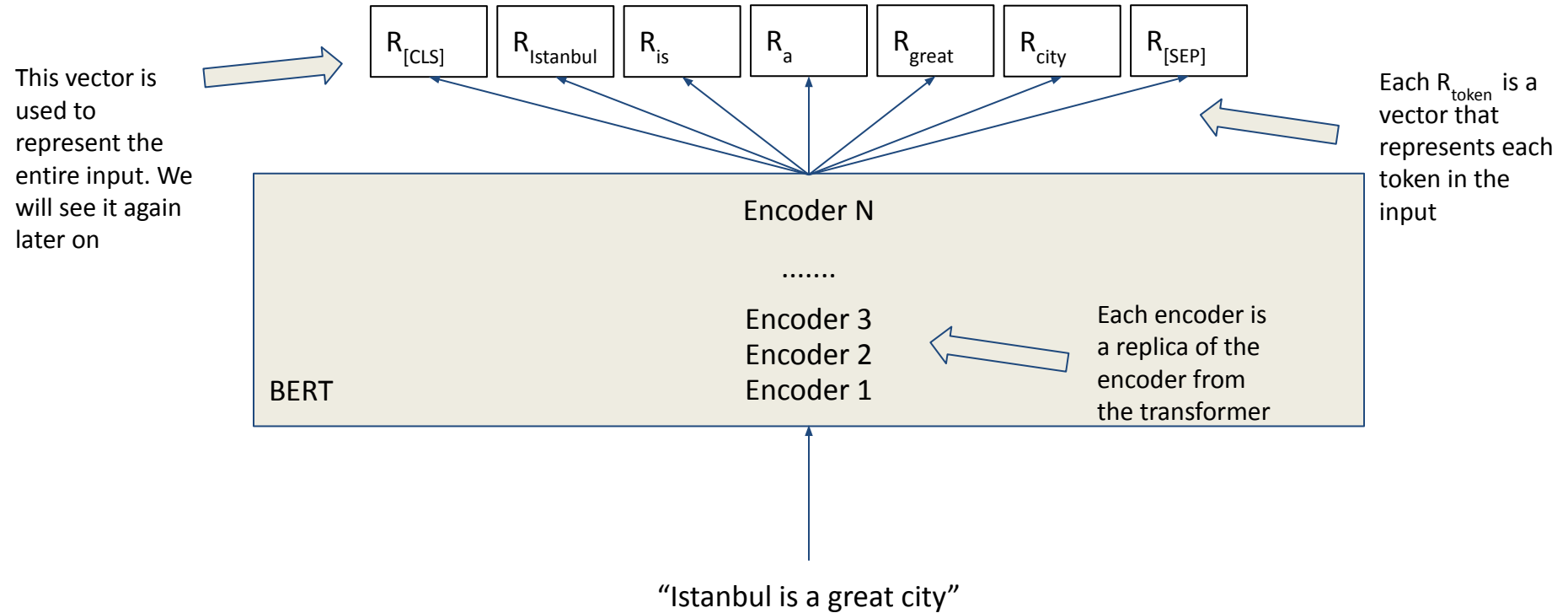
Consider the following sentence:

`'I love my pet Python'.`

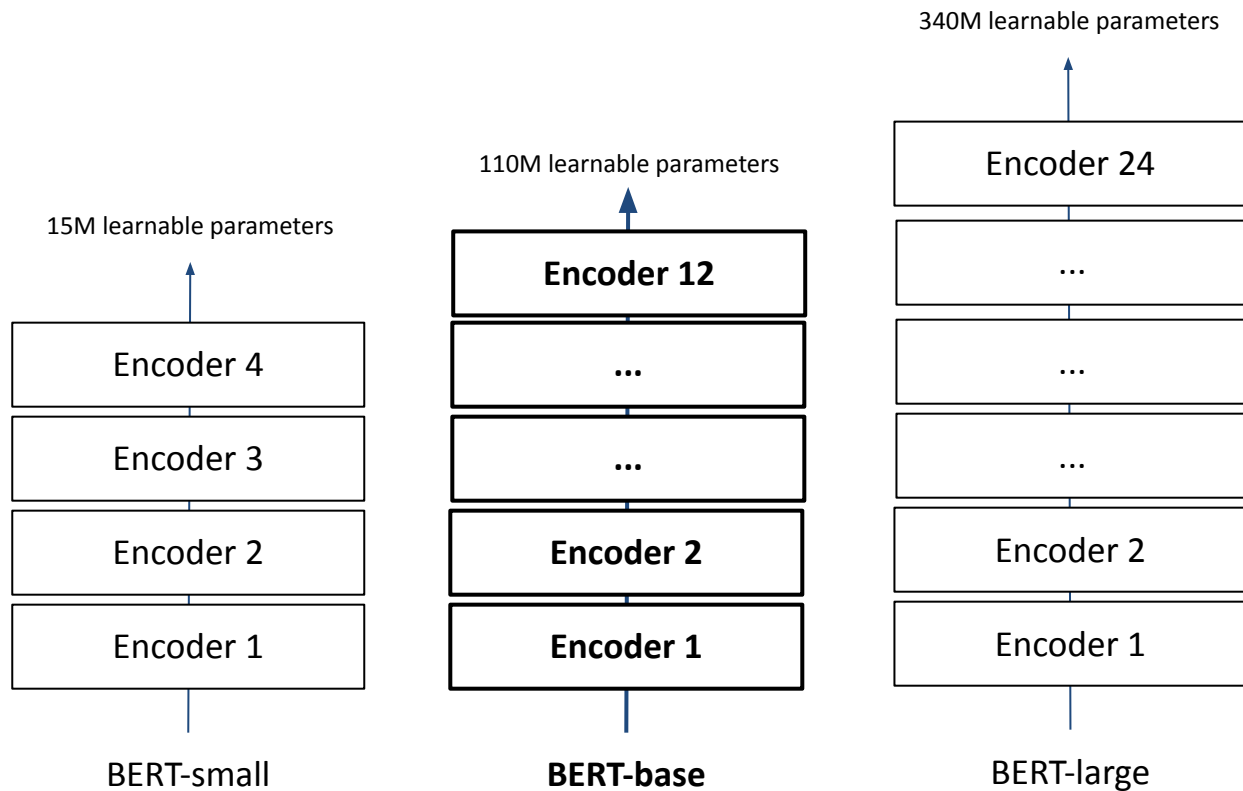
We feed this sentence into BERT to get a **contextual** representation (vector embedding) of **every** word in the sentence.

The encoder understands the context of each word in the sentence using a multi-head attention mechanism (which relates each word to every other word in the sentence)

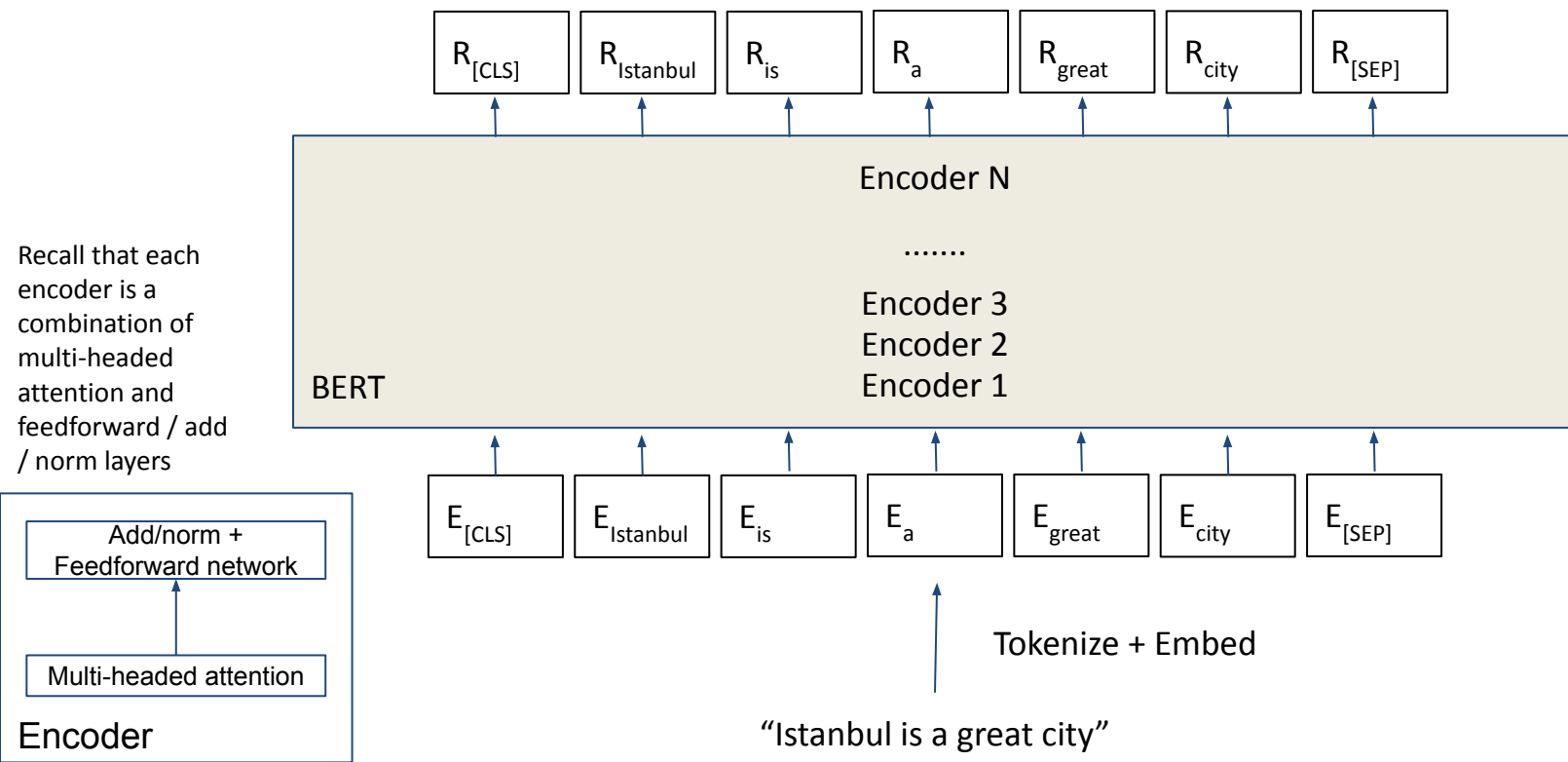
BERT at a distance



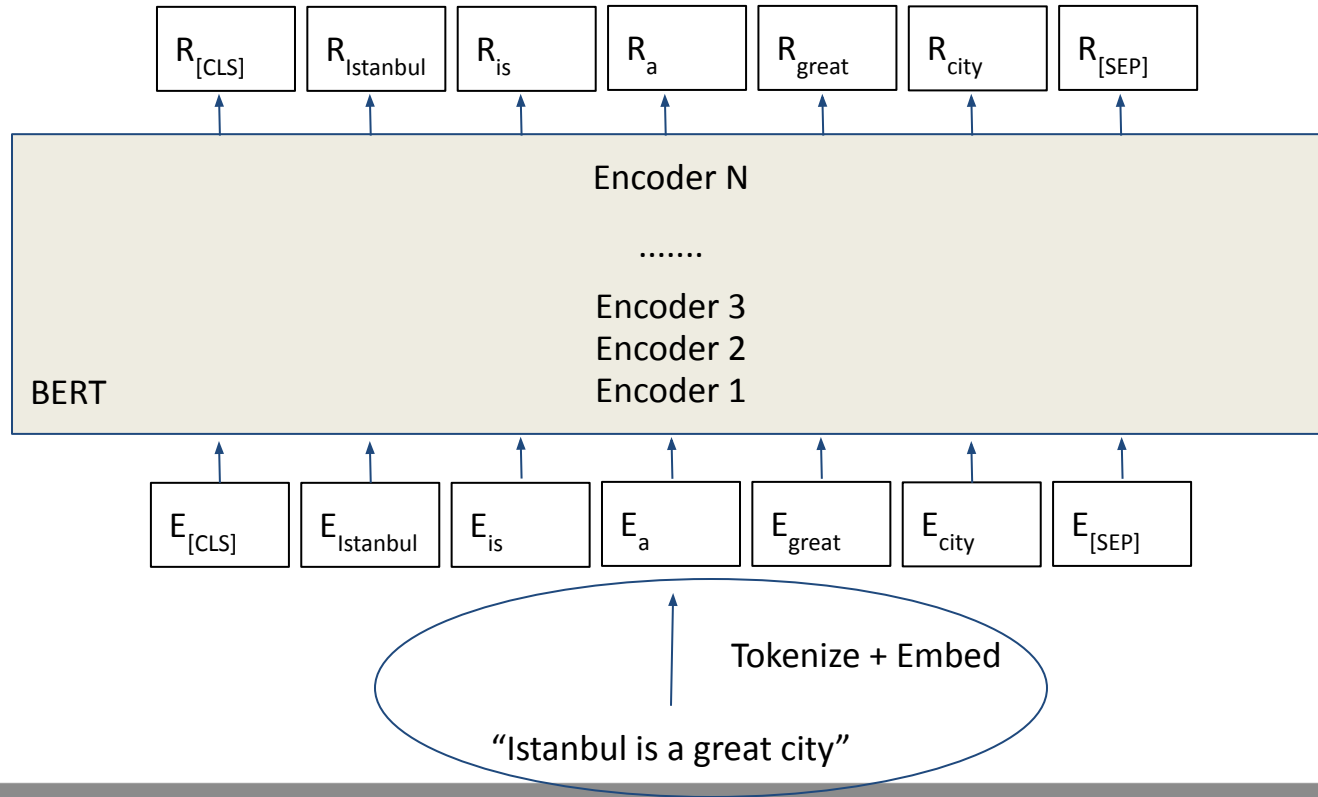
BERT comes in many sizes



BERT at a slightly closer distance



Tokenize + Embedding Layer



BERT's Wordpiece Tokenization

Consider the following sentence:

“Another beautiful day”

To tokenize this, we split into a list of tokens in our vocabulary over over 30,000 tokens. We also add two special tokens **[CLS]** at the beginning of the phrase and **[SEP]** at the end. [CLS] is meant to represent the entire sequence and [SEP] is meant to represent separation between sentences.

[“[CLS]”, “another”, “beautiful”, “day”, “[SEP]”]

BERT's Wordpiece Tokenization

Consider the following sentence:

“Sinan loves a beautiful day”

```
In [121]: 'sinan' in tokenizer.vocab
```

```
Out[121]: False
```

Story of my life..

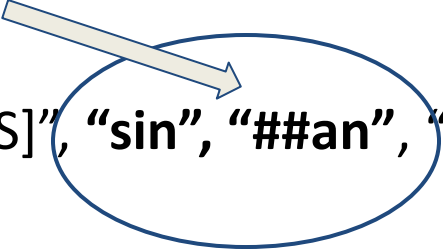
BERT's Wordpiece Tokenization

Consider the following sentence:

“Sinan loves a beautiful day”

the “##” indicates a subword

[“[CLS]”, “sin”, “##an”, “loves”, “a”, “beautiful”, “day”, “[SEP]”]



BERT's tokenizer is great at handling tokens that are OOV (out of vocabulary) by breaking them up into smaller chunks of known tokens

BERT's Wordpiece Tokenization

Note on tokenization:

BERT has a **maximum sequence length of 512 tokens**. This was implemented for the sake of efficiency.

Any sequence less than 512 tokens it will be padded to reach 512 and if it is over 512, the model may error out

Tokenization - Uncased vs Cased

Two terms we see a lot when working with BERT tokenization is **uncased** and **cased**.

Uncased

- Removes accents
- lower-cases the input

Café Dupont --> cafe dupont

Cased

- Does nothing to the input

Café Dupont --> Café Dupont

Tokenization - Uncased vs Cased

- Uncased tokenization is usually better for most situations because case generally doesn't contribute to context
- Cased tokenization works well in cases where case does matter (like Named Entity Recognition)
- Note that this has little to do with the BERT architecture, just in tokenization

BERT's Embedding Layer

BERT applies three separate types of embeddings to tokenized sentences:

1. Token Embeddings
 - a. Represents context-free meaning of each token
 - b. A lookup of 30,522 possible vectors
 - c. This is learnable during training
2. Segment Embeddings
 - a. Distinguishes between multiple inputs (for Q/A for example)
 - b. A lookup of 2 possible vectors (one for sentence A one for sentence B)
 - c. This is learnable during training
3. Position Embeddings
 - a. Used to represent the token's position in the sentence
 - b. This is not learnable

BERT's Embedding Layer

Input [CLS] Paris is a beautiful city [SEP] I love Paris [SEP] 11 tokens

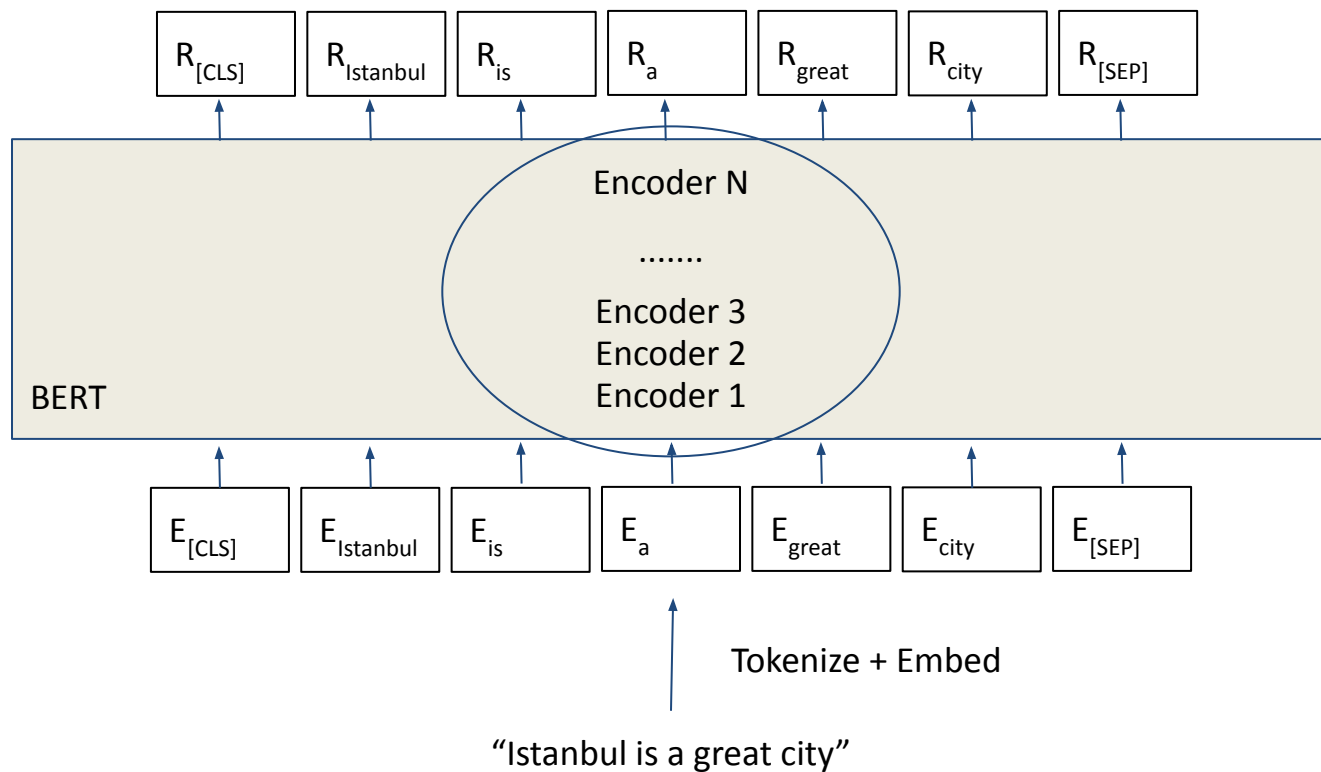
Token embeddings	$E_{[CLS]}$	E_{Paris}	E_{is}	E_a	$E_{\text{beautiful}}$	E_{city}	$E_{[SEP]}$	E_I	E_{love}	E_{Paris}	$E_{[SEP]}$	(11, 768)
	+	+	+	+	+	+	+	+	+	+	+	+
Segment embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	(11, 768)
	+	+	+	+	+	+	+	+	+	+	+	+
Position embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	(11, 768)

=

Each of these rectangles represents a vector of shape (1, 768) (assuming BERT-base)

Final processed input has shape (11, 768)

Re-visiting the Encoder's Attention



Re-visiting the Encoder's Attention

Standard attention is calculated using three matrices called the **query**, **key**, and the **value** matrices.

 **Linear Algebra Alert** 

Re-visiting Attention

X = Our tokenized input

each row is the token embedding for each token in the input.

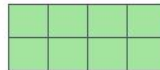
This input has already gone through BERT's preprocessing (token, segment, and position)

In this example, We have an input of **2** tokens

The matrix has shape:

number of tokens x embedding dimension

X



Re-visiting Attention

X = number of tokens \times embedding dimension

We perform three **linear transformations**
on the matrix X to project it onto
three new vector spaces called

the **query space**

the **key space**

the **value space**

$$\begin{matrix} X \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} W^Q \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} Q \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

$$\begin{matrix} X \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} W^K \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} K \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

$$\begin{matrix} X \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} W^V \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} V \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

Re-visiting Attention

X = number of tokens x embedding length

W^Q = embedding length x embedding dimension

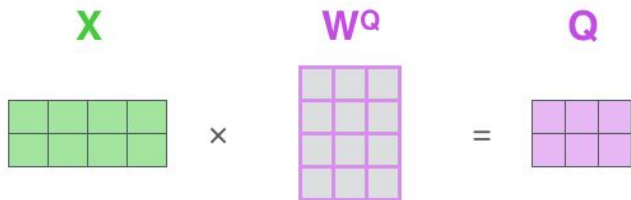
W^K = embedding length x embedding dimension

W^V = embedding length x embedding dimension

Q = number of tokens x embedding dimension

K = number of tokens x embedding dimension

V = number of tokens x embedding dimension



Re-visiting Attention

Q = number of tokens x embedding dimension

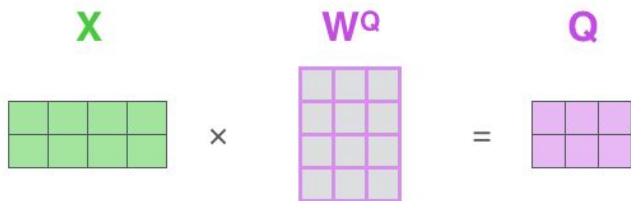
K = number of tokens x embedding dimension

V = number of tokens x embedding dimension

the Q (query) matrix intuitively represents the information we are looking for

the K (key) matrix intuitively represents the relevance of each word to our query

the V (value) matrix intuitively represents the **context-less** meaning of our input tokens



Re-visiting Attention

Q = number of tokens x embedding dimension

K = number of tokens x embedding dimension

V = number of tokens x embedding dimension

Q, K, and V all contain information regarding individual tokens. At this point each matrix contains a representation of each token in a new vector space

$$\begin{matrix} \text{X} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \text{W}^{\text{Q}} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \text{Q} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

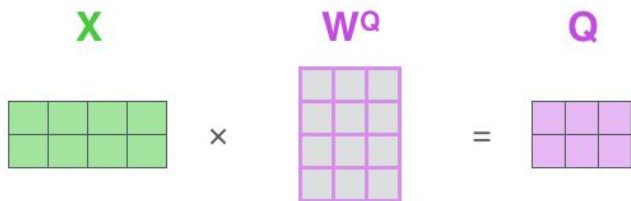
$$\begin{matrix} \text{X} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \text{W}^{\text{K}} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \text{K} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

$$\begin{matrix} \text{X} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \text{W}^{\text{V}} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \text{V} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

Re-visiting Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

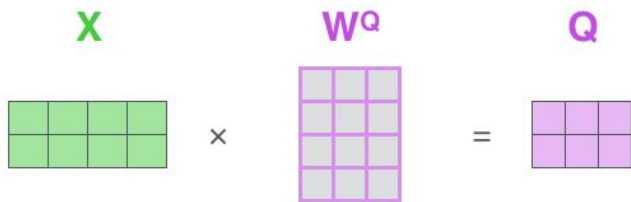
This is our attention equation. It takes the Q, K, and V matrices and combines them in a way that forces tokens to focus on each other and scales our context-less representations of our tokens, transforming them into context-ful representations



Re-visiting Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Let's break down this equation a bit..



Re-visiting Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

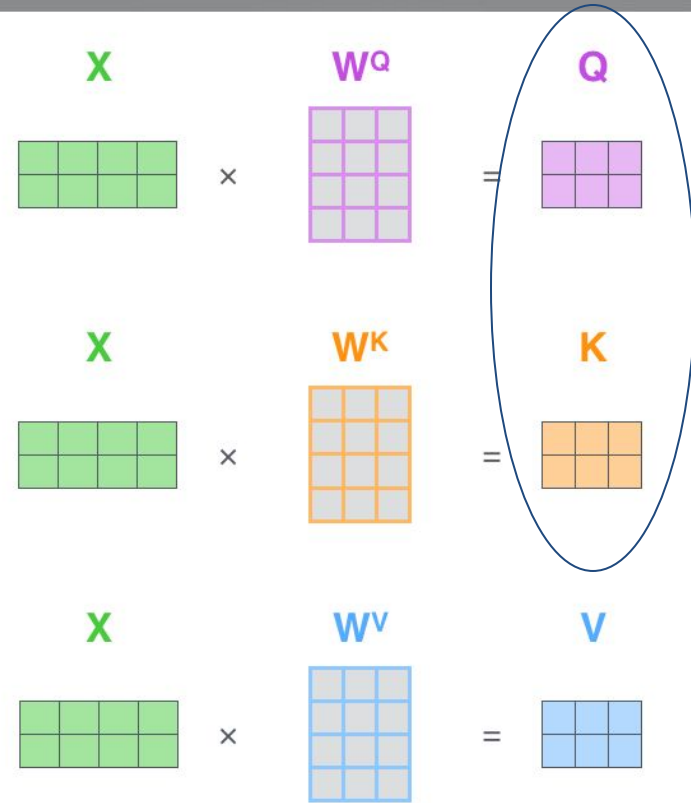
Let's break down this equation a bit..

QK^T = multiplying the query / key matrices

This is a square matrix that represents the **attention score** of the tokens

$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ = dividing the attention score by

the root of the embedding dimension to make the values more manageable and apply softmax to obtain probabilities

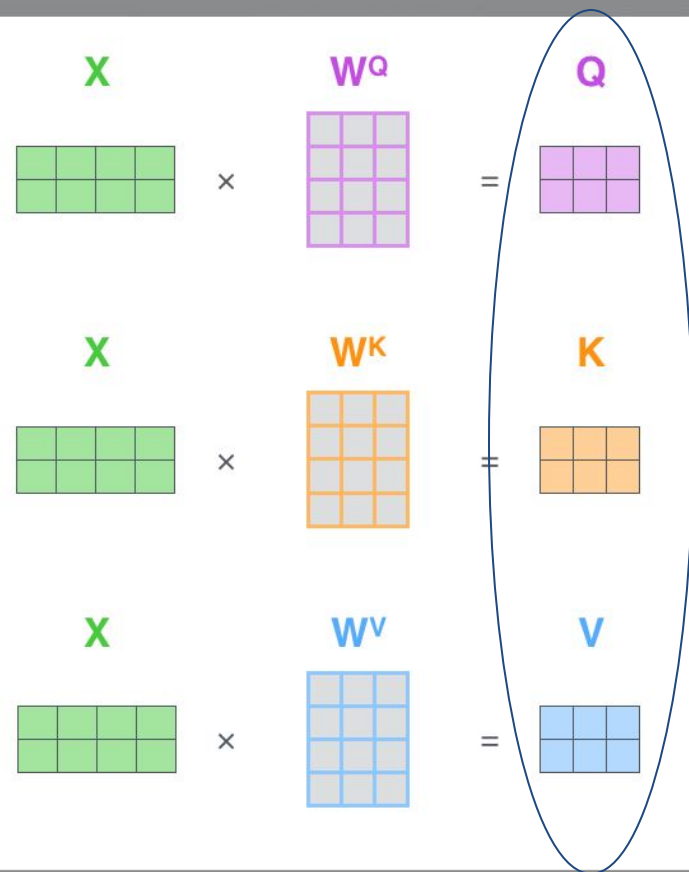


Re-visiting Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multiply our square matrix - which no longer holds a representation of each individual token - by the value matrix, which still represents each individual token. This way, we “scale” each token-value embedding with much we should be paying attention to it (as a percentage from our softmax calculation). We now have **context-ful representations of our input tokens**

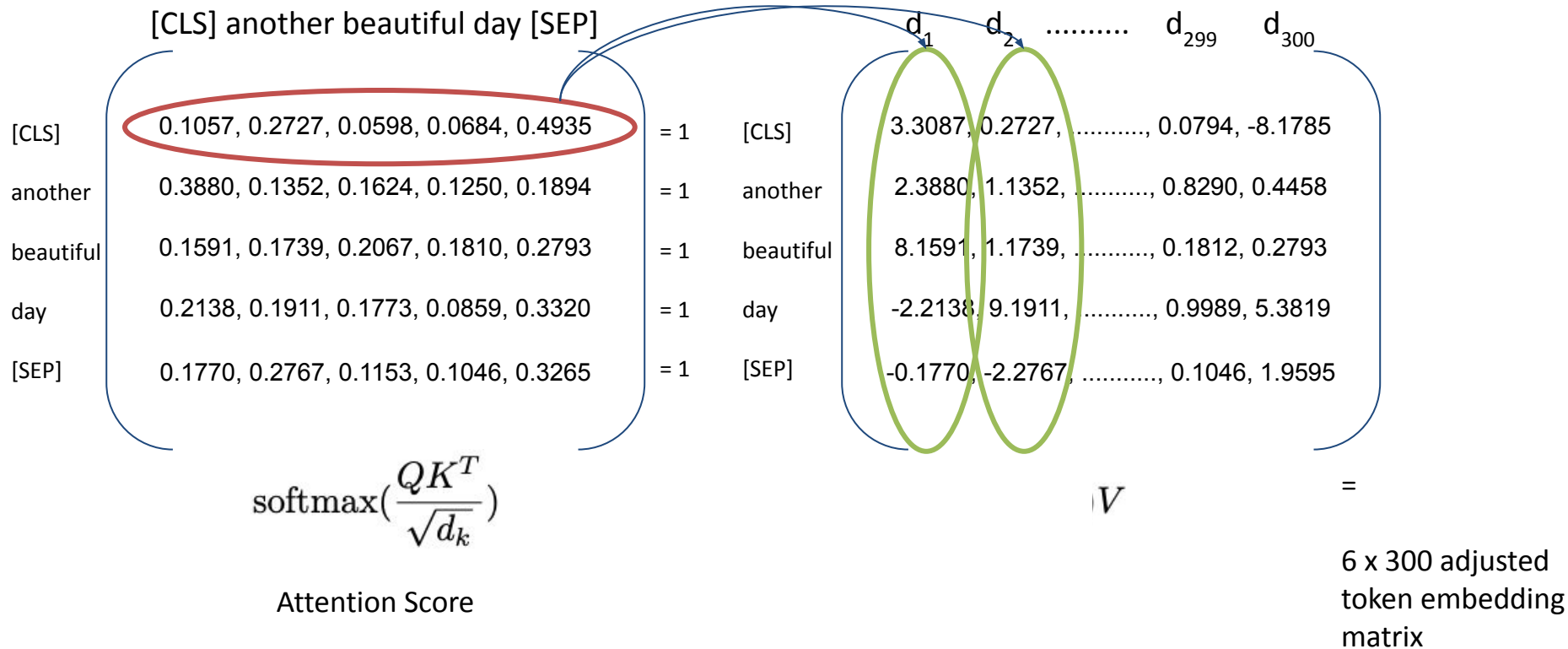


Re-visiting Attention

[CLS] another beautiful day [SEP]						d_1	d_2	d_{299}	d_{300}	
[CLS]	0.1057, 0.2727, 0.0598, 0.0684, 0.4935	= 1	[CLS]	3.3087, 0.2727,, 0.0794, -8.1785							
another	0.3880, 0.1352, 0.1624, 0.1250, 0.1894	= 1	another	2.3880, 1.1352,, 0.8290, 0.4458							
beautiful	0.1591, 0.1739, 0.2067, 0.1810, 0.2793	= 1	beautiful	8.1591, 1.1739,, 0.1812, 0.2793							
day	0.2138, 0.1911, 0.1773, 0.0859, 0.3320	= 1	day	-2.2138, 9.1911,, 0.9989, 5.3819							
[SEP]	0.1770, 0.2767, 0.1153, 0.1046, 0.3265	= 1	[SEP]	-0.1770, -2.2767,, 0.1046, 1.9595							
$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})$						V					=
Attention Score											6 x 300 adjusted token embedding matrix

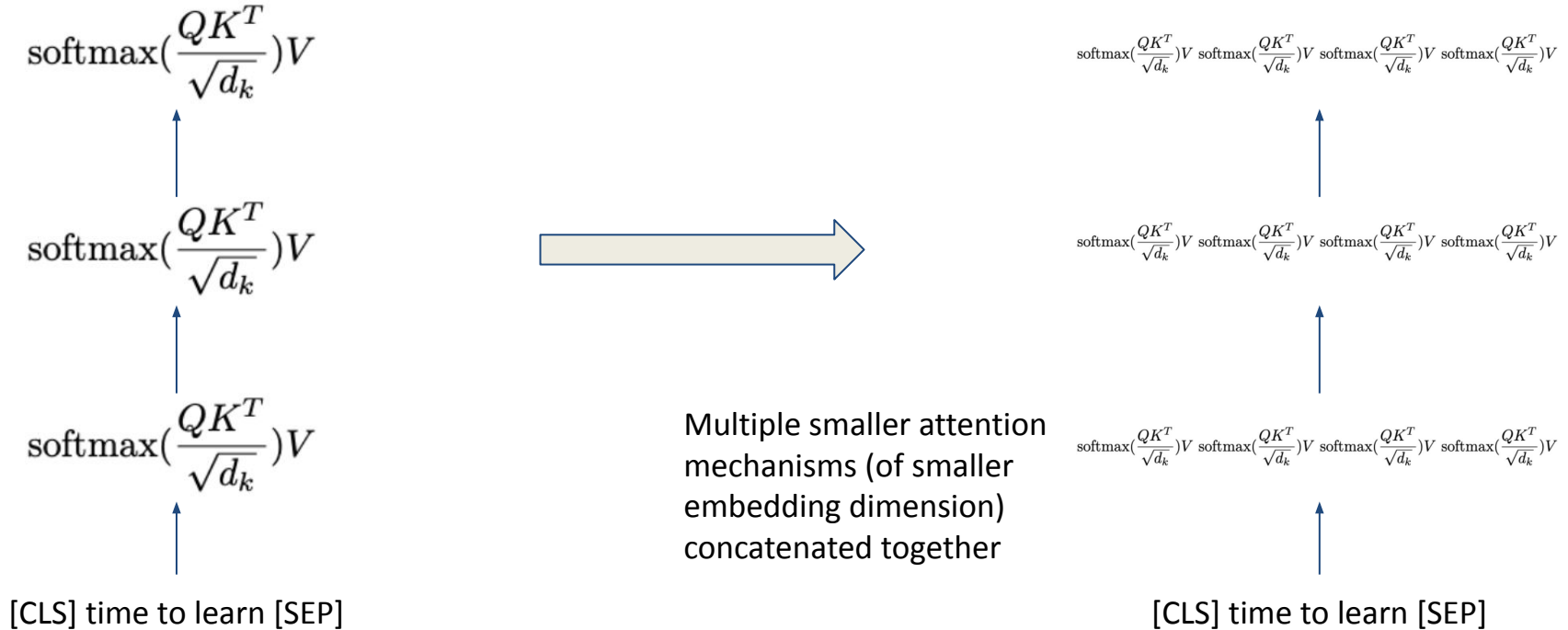
Re-visiting Attention

Adjust each dimension with attention



Multi-headed Self-Attention

Transformers + BERT use **multi-headed self-attention**



Multi-headed Self-Attention

In each encoder, we apply our multi-headed attention

In the first encoder we do the calculation on our initial X matrix like in our previous example

Every encoder afterwards uses the representation matrix outputted by the preceding encoder

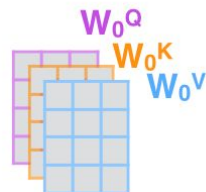
1) This is our input sentence*

Thinking
Machines

2) We embed each word*



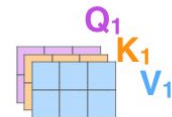
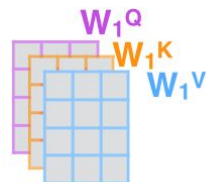
3) Split into 8 heads.
We multiply X or R with weight matrices



4) Calculate attention using the resulting $Q/K/V$ matrices



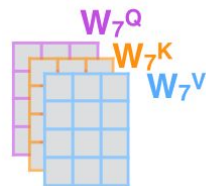
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



...

...

...



Multi-headed Self-Attention

In [34]: *#Let's take a look at the configuration of this BERT model*

```
config = BertConfig()  
config
```

Out[34]: BertConfig {
 "attention_probs_dropout_prob": 0.1,
 "gradient_checkpointing": false,
 "hidden_act": "gelu",
 "hidden_dropout_prob": 0.1,
 "hidden_size": 768,
 "initializer_range": 0.02,
 "intermediate_size": 3072,
 "layer_norm_eps": 1e-12,
 "max_position_embeddings": 512,
 "model_type": "bert",
 "num_attention_heads": 12,
 "num_hidden_layers": 12,
 "pad_token_id": 0,
 "position_embedding_type": "absolute",
 "transformers_version": "4.5.1",
 "type_vocab_size": 2,
 "use_cache": true,
 "vocab_size": 30522
}

BERT-base has 12
attention heads and 12
encoder layers

Multi-headed Self-Attention

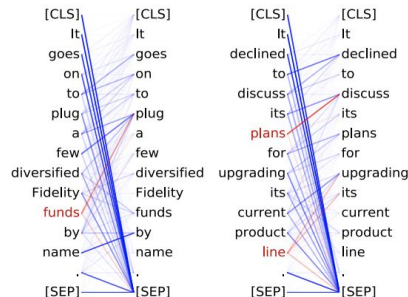
Transformers + BERT use **multi-headed self-attention**. Why?

1. Helps BERT focus on multiple relationships simultaneously
 - a. Eg. “Sinan ate the cereal and then he had a stomach ache”
 - b. the word **he** relates to Sinan as a pronoun
 - c. the word **he** also relates to **had**
2. Helps BERT learn different types of relations between words by transforming our inputs to multiple sub-spaces of Q/K/V

Multi-headed Self-Attention

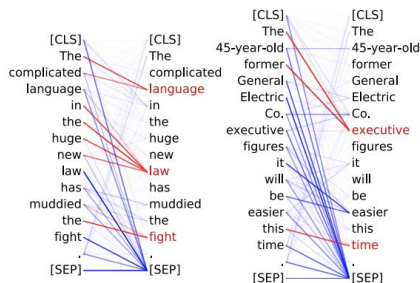
Head 8-10

- **Direct objects** attend to their verbs
- 86.8% accuracy at the **dobj** relation



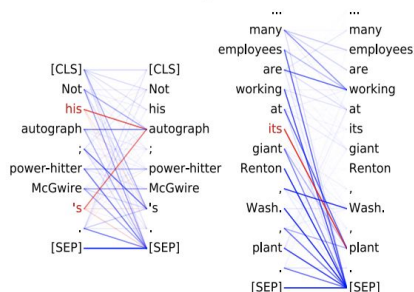
Head 8-11

- **Noun modifiers** (e.g., determiners) attend to their noun
- 94.3% accuracy at the **det** relation



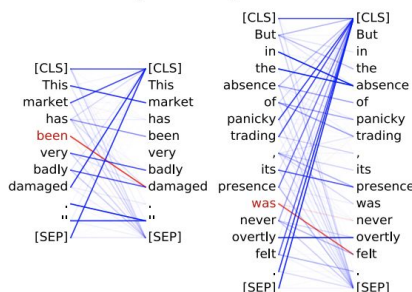
Head 7-6

- **Possessive pronouns** and apostrophes attend to the head of the corresponding NP
- 80.5% accuracy at the **poss** relation

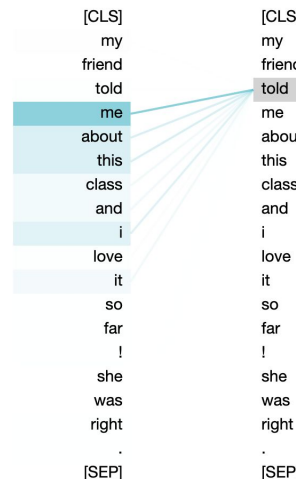


Head 4-10

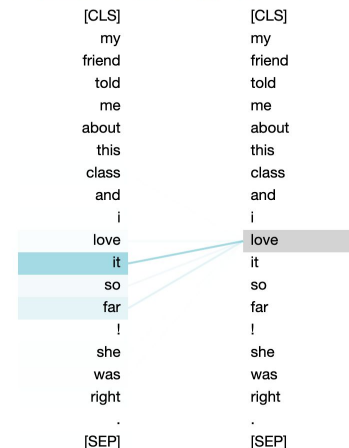
- **Passive auxiliary verbs** attend to the verb they modify
- 82.5% accuracy at the **auxpass** relation



Layer: 7



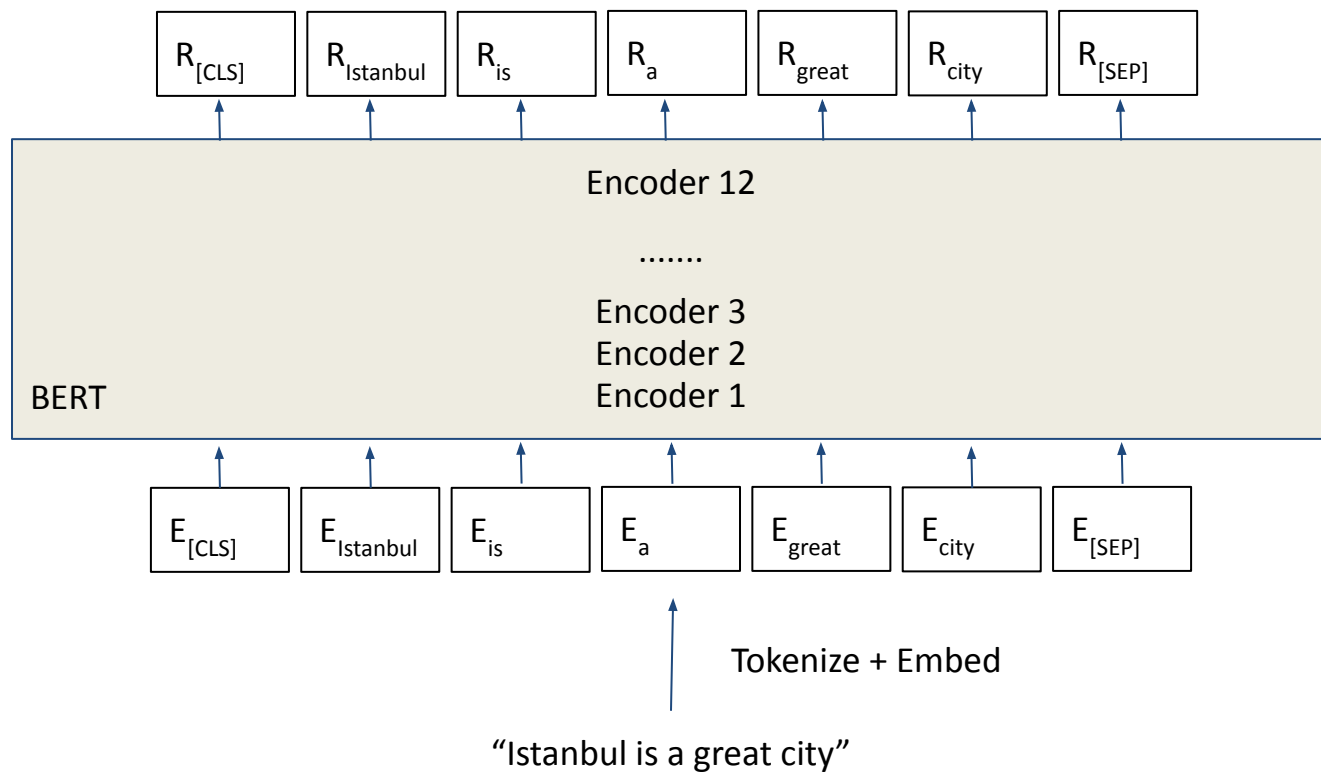
Layer: 7



Head 8-10 - direct objects to their verbs eg.
eg. love -> it

technically "told" is an indirect object because tell is acting as an intransitive verb

Putting it all together



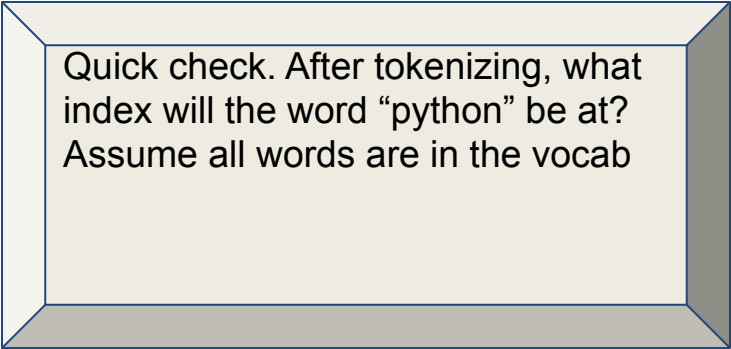
Representations of words with context

Consider the following sentences:

`'I love my pet Python'`

VS

`'I love coding in Python'`



Quick check. After tokenizing, what index will the word “python” be at?
Assume all words are in the vocab

Representations of words with context

Consider the following sentences:

'I love my pet Python'

VS

'I love coding in Python'

Quick check. After tokenizing, what index will the word “python” be at? Assume all words are in the vocab

5 (zero-index) Don't forget [CLS] would be 0!

['[CLS]', 'i', 'love', 'my', 'pet', '**python**', '[SEP]']

0 1 2 3 4 **5** 6

Representations of words with context

Consider the following sentences:

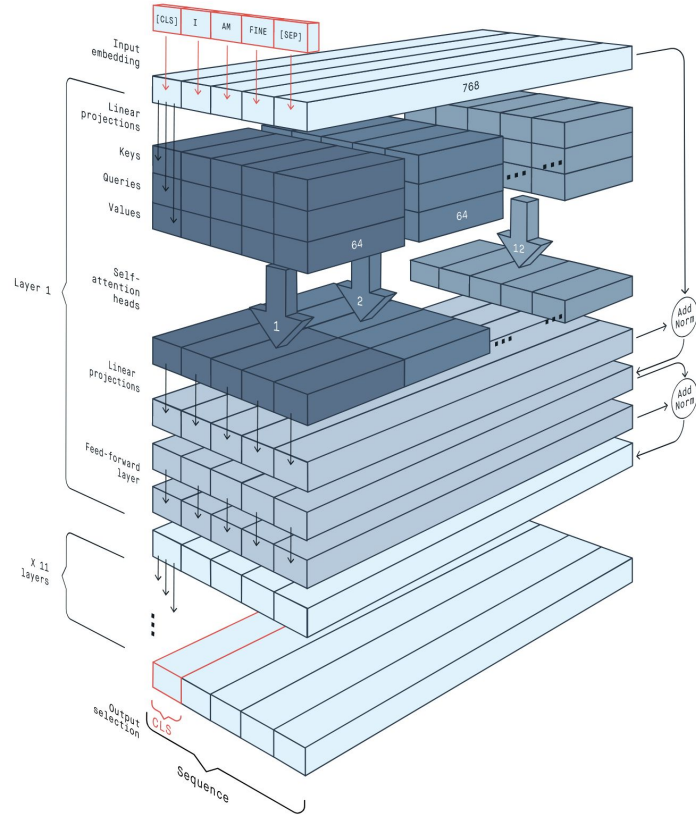
`'I love my pet Python'`

vs

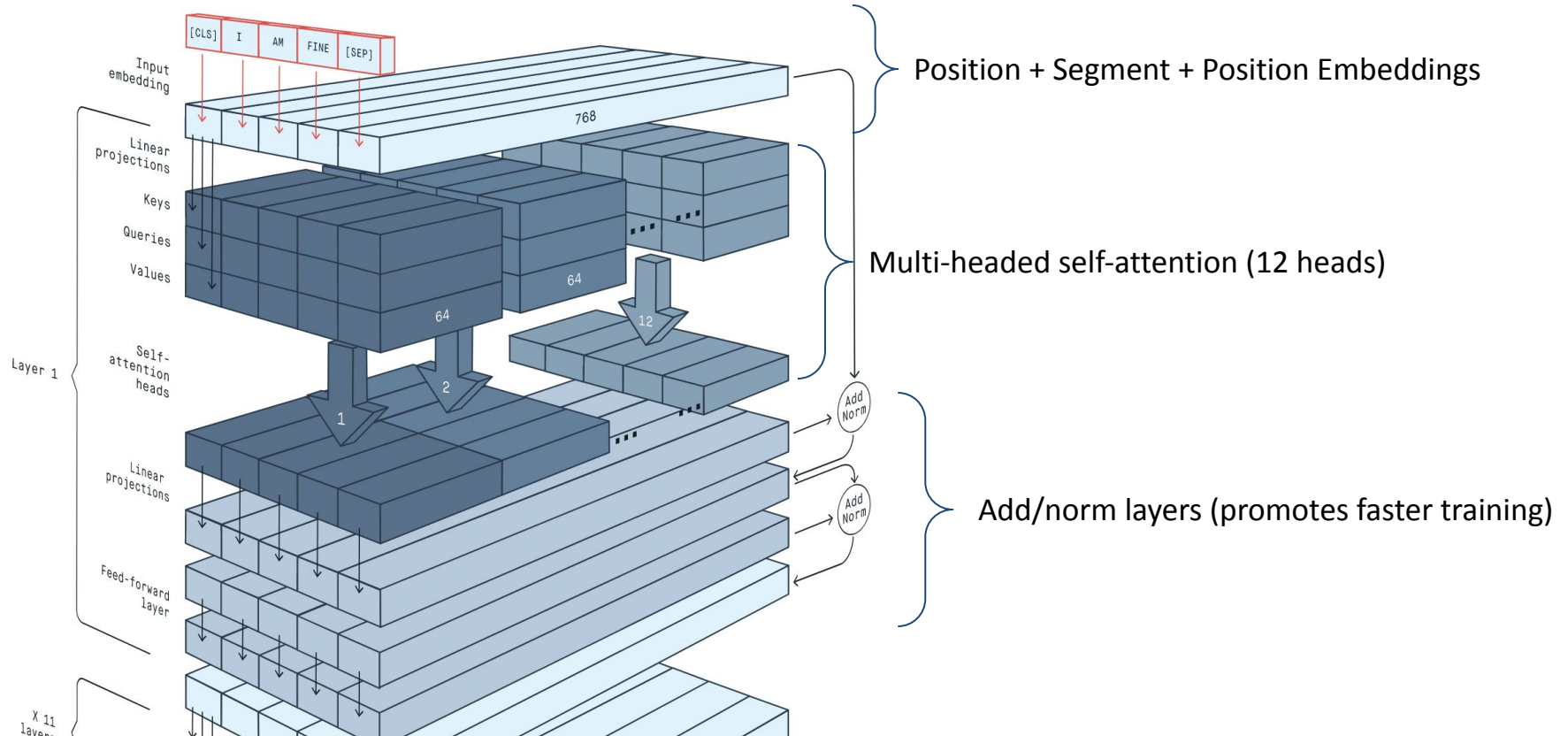
`'I love coding in Python'`

The token “python” will end up with a vector representation from each sentence via BERT. What’s interesting is that the vector representation “python” will be different for each sentence because of the surrounding words in the sentence. We will see this example in our upcoming code

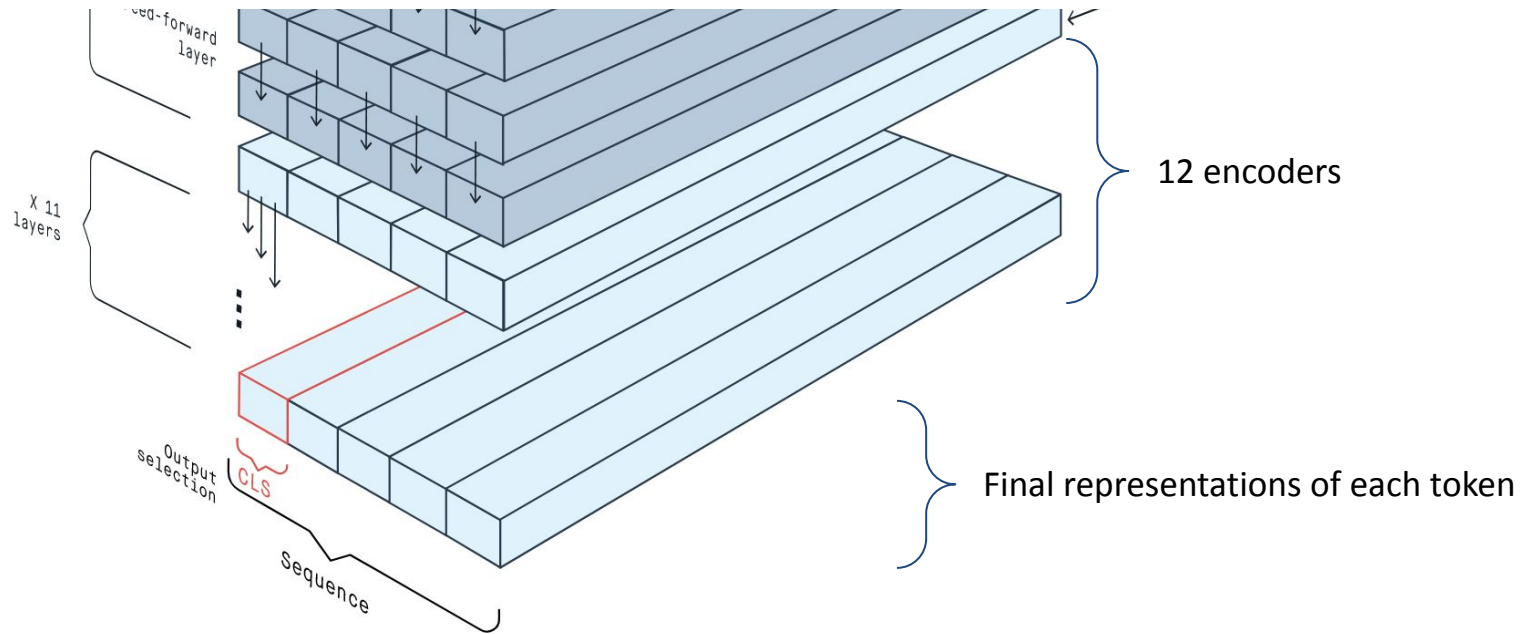
BERT Visualization



BERT Visualization



BERT Visualization



Code time!



Recap - Attention

- Linear transformations of the input via Query/Key/Value matrices allow for multiple representations of the same input
- Multi-headed attention allows for multiple types of attentions to train in parallel
- A lot of linear algebra

Recap - BERT's Architecture

- Pre-processing to represent meaning, segments, and position
- Stack of Encoders using multi-headed attention taken from the Transformers (12 in BERT-base)
- Add/norm + feed-forward layers to speed up training and promote generalizability

Q/A --> Break



Pre-training and Fine-tuning BERT to perform NLP tasks

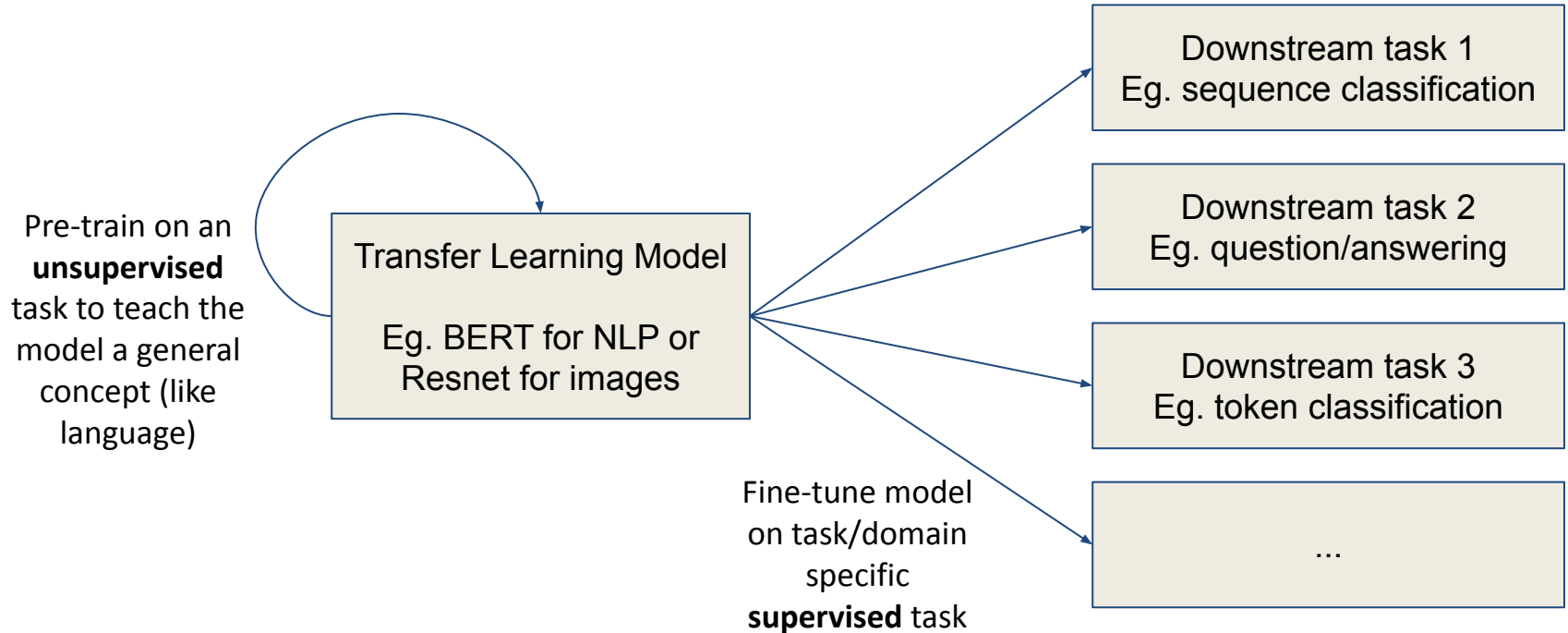
Transfer Learning

Transfer learning describe when a model is pre-trained on an unlabeled text corpora on an unsupervised task that generally doesn't have a "useful" objective. It is just meant to learn language/context in general

The model is then fine-tuned (updated) on a specific NLP "downstream" task using a labeled dataset that is usually quite small in comparison.

If we simply trained on the downstream smaller dataset without pre-training, we would never be able to achieve the same state of the art results

Transfer Learning



Pre-training BERT

Pre-training is where BERT really starts to stand out. BERT is pre-trained on two tasks:

1. The Masked Language Model
2. Next Sentence Prediction

These tasks are not generally “useful” tasks but they help BERT learn how words / language work in general

Pre-training BERT

Masked Language Modelling (MLM)

- Replace 15% of words in corpus with special [MASK] token and ask BERT to fill in the blank
- Think back to our “__ at the light” example. This is the MLM task

Next Sentence Prediction (NSP)

- Classification problem
- Given two sentences, did sentence B come **directly** after sentence A?
 - True or False

Pre-training BERT

Masked Language Modelling (MLM)

“Istanbul is a great [MASK] to visit”



Guess the word

Next Sentence Prediction (NSP)

A: “Istanbul is a great city to visit”

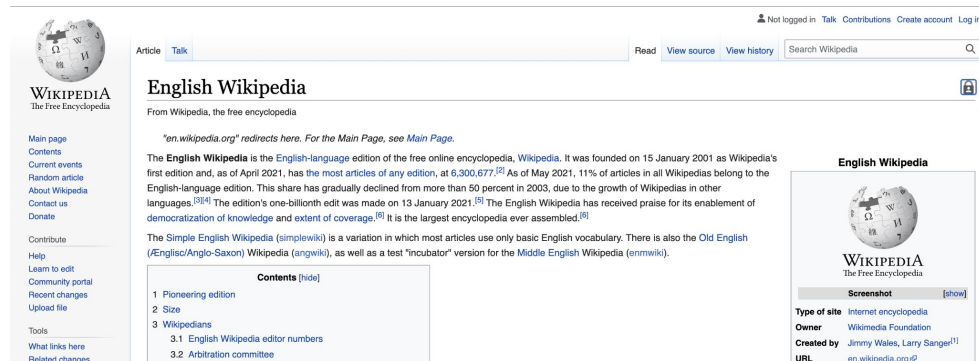
B: “I was just there.”

Did sentence B come directly after sentence A? Yes or No

Pre-training BERT - Corpus

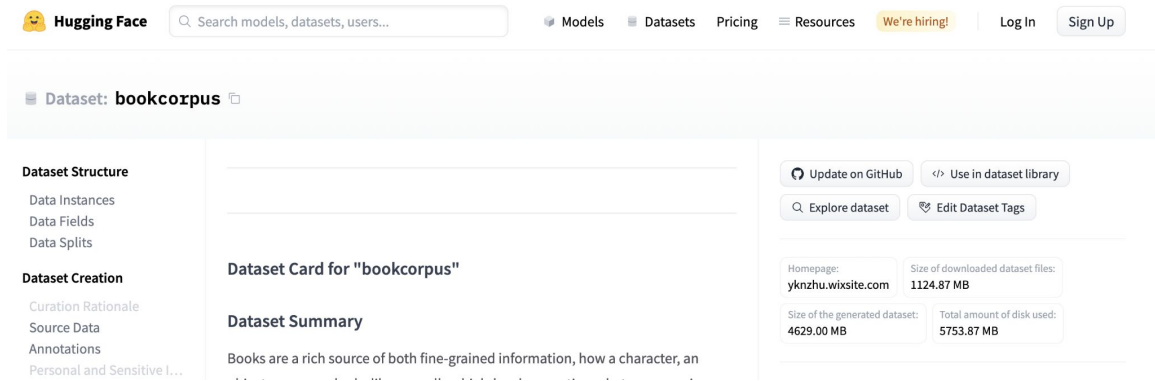
English Wikipedia (2.5B words)

https://en.wikipedia.org/wiki/English_Wikipedia

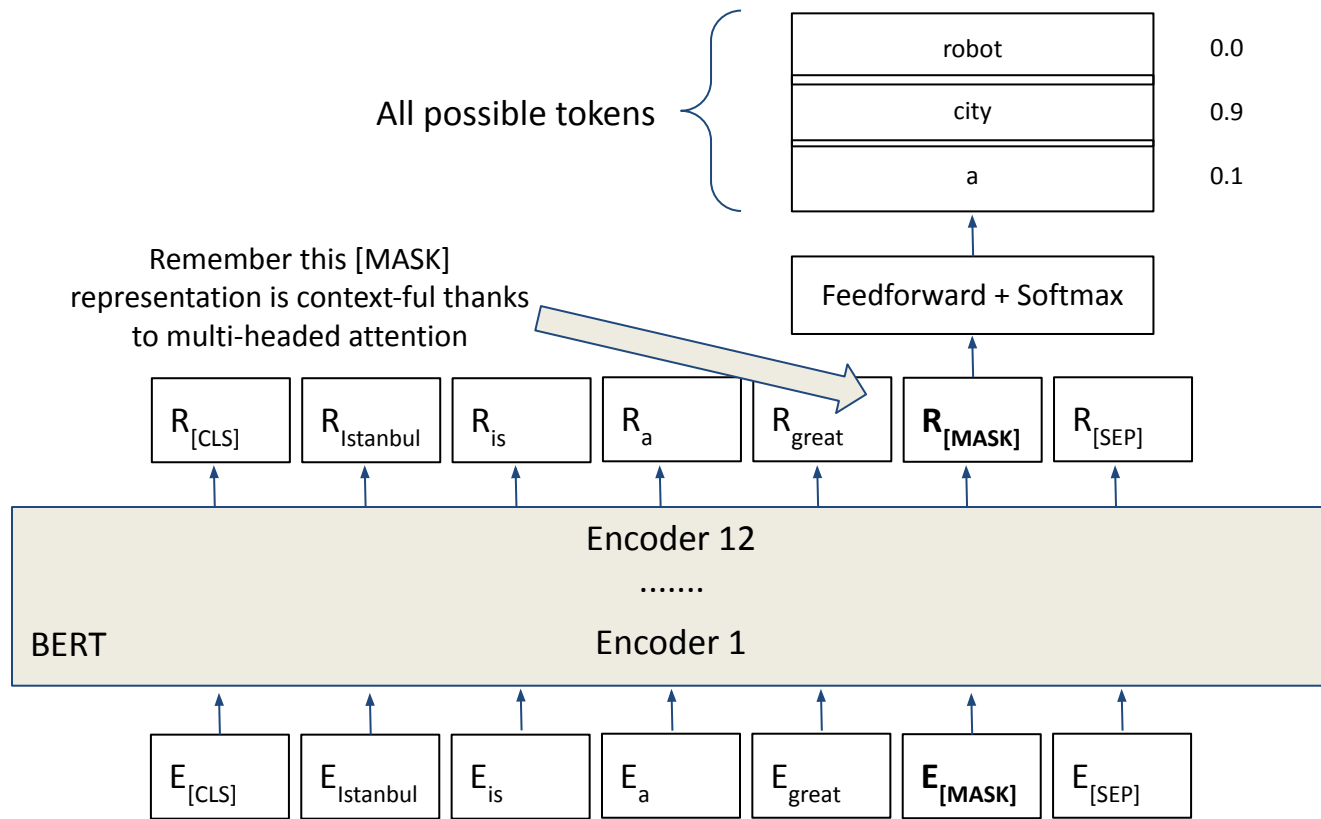


BookCorpus (800M words)

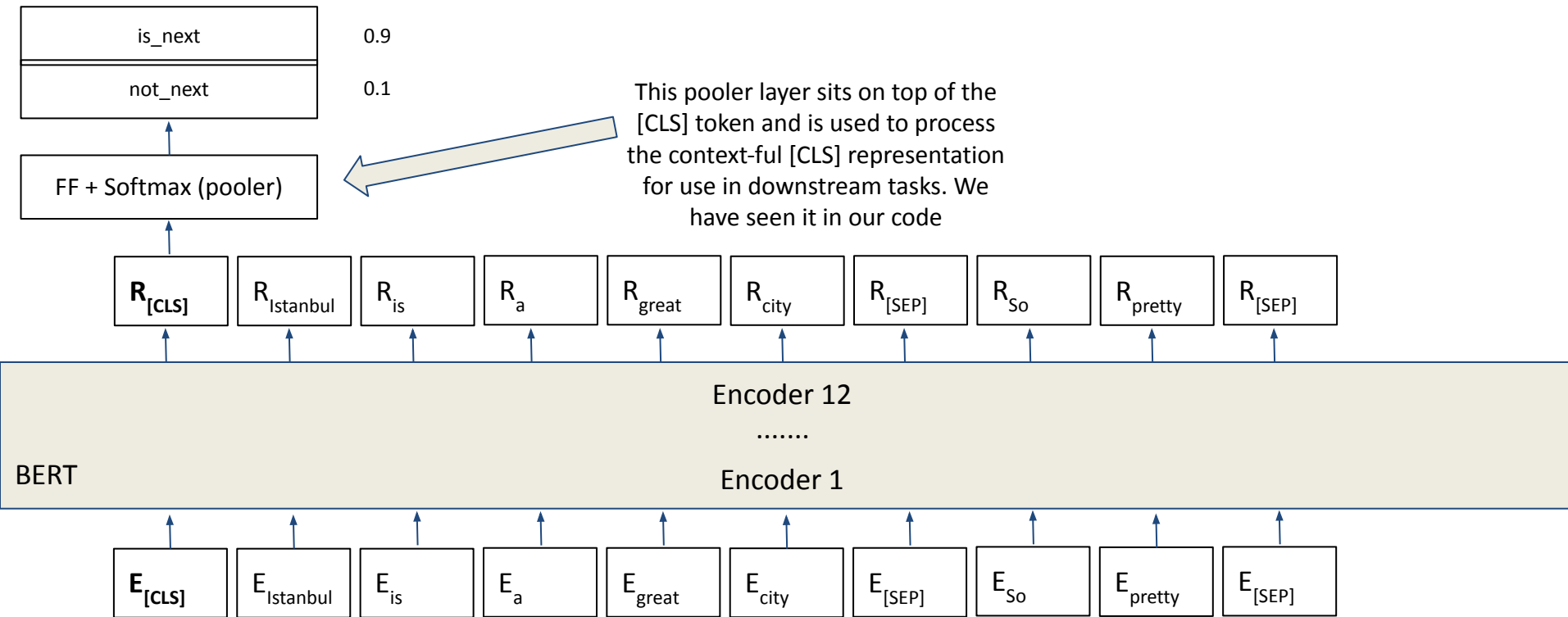
huggingface.co/datasets/bookcorpus



Pre-training BERT - MLM



Pre-training BERT - NSP



Transfer Learning - Fine-tuning

There are generally three approaches to fine-tuning:

1. Updating the whole BERT model (all encoder layers) on the labeled data + any additional layers added on top
2. Updating a subset of the BERT encoding layers to save computation time
3. Freezing all of the BERT encoding layers model and only training the additional layers added on top

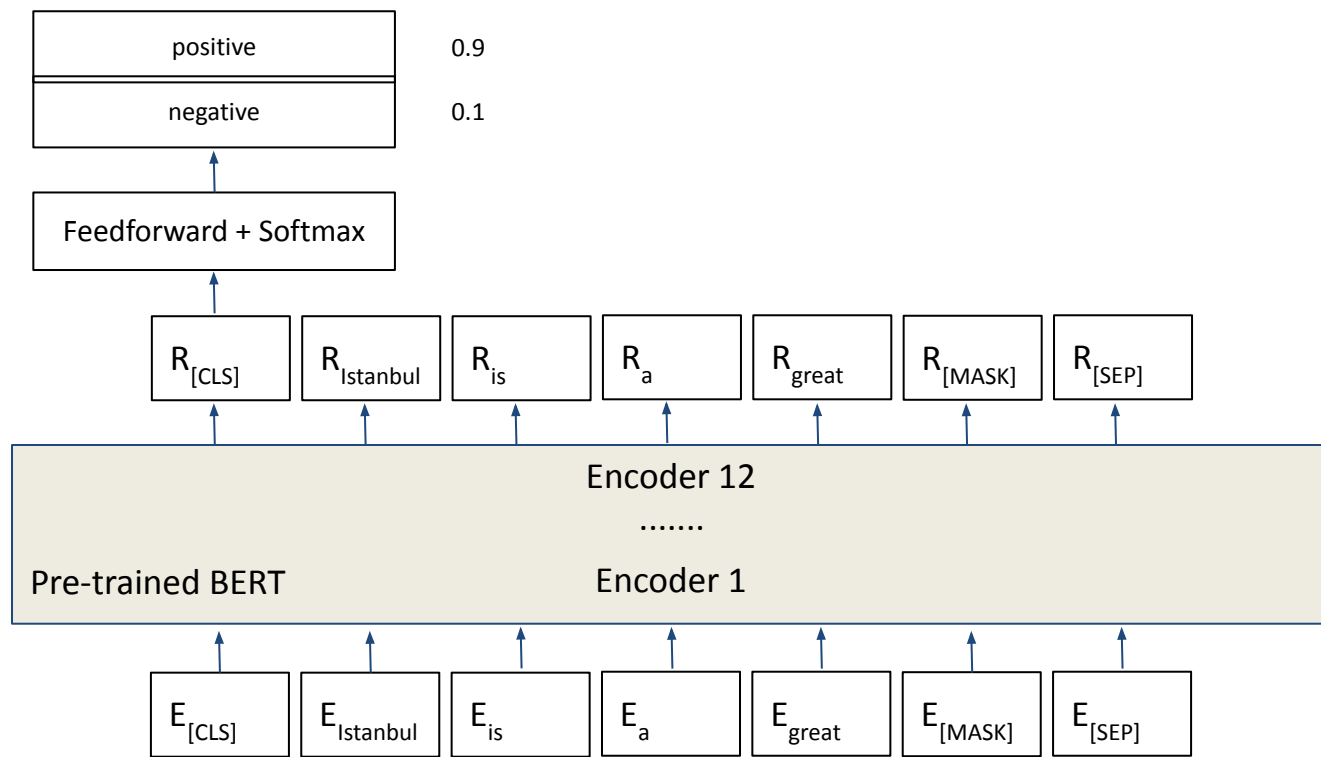
Transfer Learning - Fine-tuning

Once BERT has a general idea about how

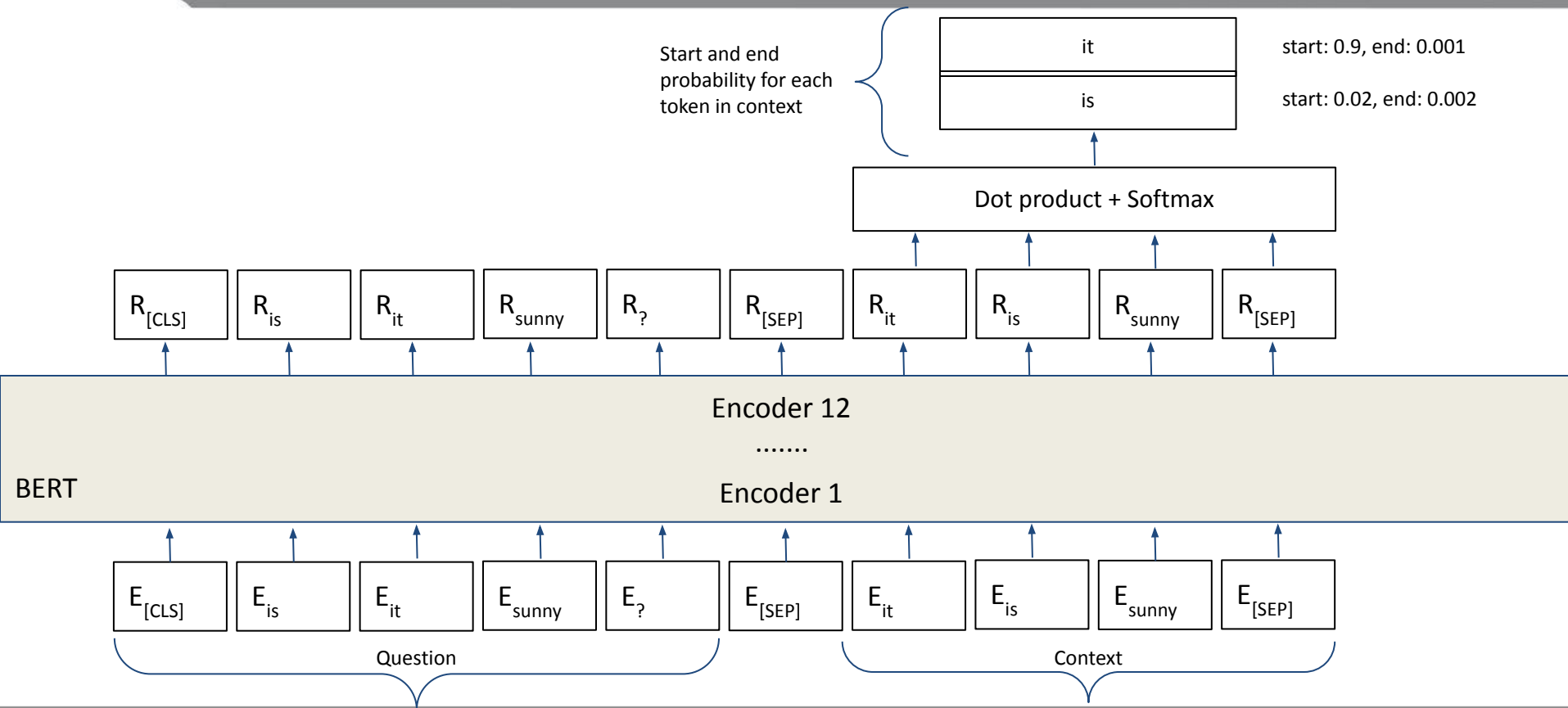
1. Words are used in sentences (MLM)
2. sentences are treated in a larger document (NSP)

It's time to take what BERT has learned and fine-tune it on a specific task of our choosing.

Fine-tuning BERT - Classification

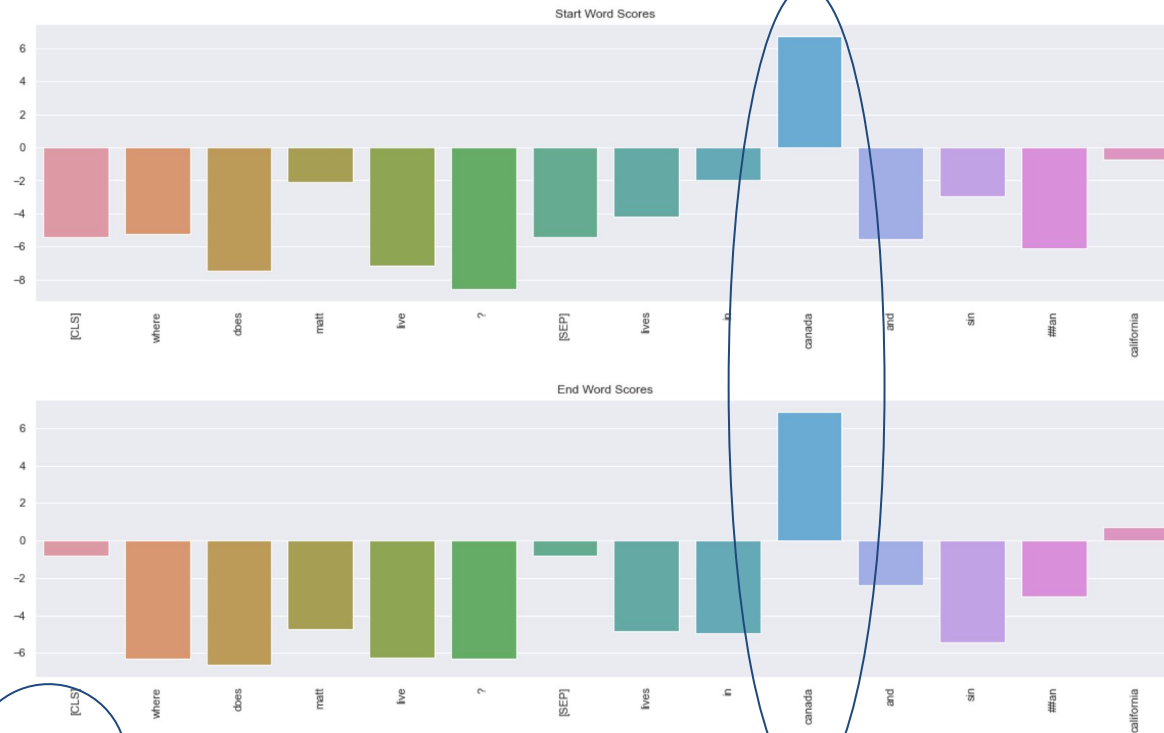


Fine-tuning BERT - Question/Answering



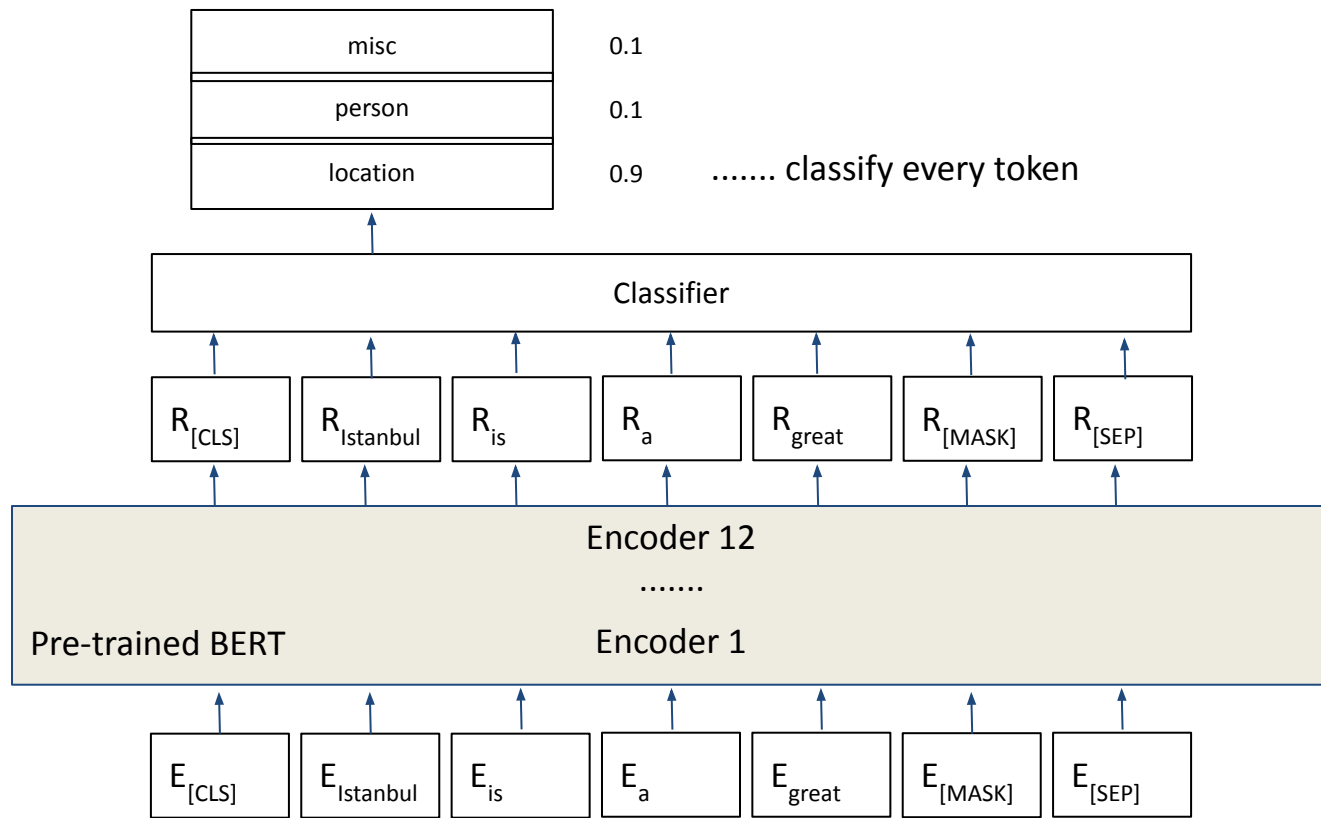
Fine-tuning BERT - Question/Answering

```
q_a("Where does Matt live?", "Matt lives in Canada and Sinan lives in California")
```



'canada'

Fine-tuning BERT - Token Classification



Code time!



Recap - Transfer Learning

- BERT is pre-trained on the masked language model and next sentence prediction task. This is an unsupervised task to teach BERT general language and context
- BERT is then fine-tuned on downstream NLP tasks using a supervised dataset

What I skipped / glossed over

1. Details on how the Segment + Position Embeddings work
2. The other kinds of downstream tasks BERT is used for
3. the “15% masking” is a bit more complicated with a few more steps I glossed over
4. Our code to fine-tune BERT was fairly surface level. Many other resources are available to show more custom fine-tuning capabilities

Q/A --> Break



Flavors of BERT + Next Steps

Flavors of BERT

BERT inspired several derivative architectures, each with their own specialties / drawbacks. Let's focus on a few of the more popular ones:

1. RoBERTa
2. DistilBERT
3. ALBERT

Each flavor attempts to enhance BERT by altering its architecture and/or its training

RoBERTa

1. **Robustly Optimized BERT Approach**
 - a. Authors claim BERT was vastly under-trained
2. 10x the training data (16GB -> 160GB) (training)
3. 15% more parameters (architecture)
4. Removed the next sentence prediction task (training)
 - a. claim is that the task is not useful
5. Dynamic Masking Pattern -> 4x the masking tasks to learn from (training)

DistilBERT

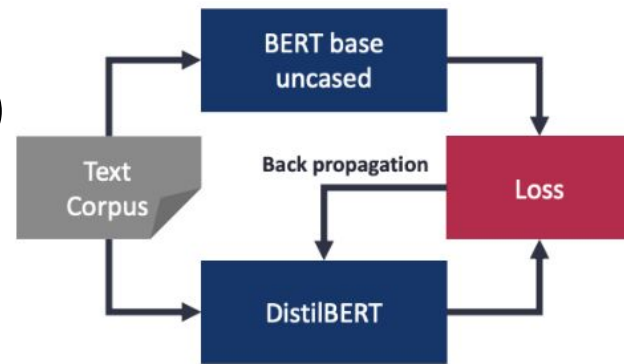
1. Distilled BERT

- a. Distillation is a technique to train a “student” model to replicate a “teacher” model

2. 40% fewer parameters, 60% faster (architecture)

- a. 97% of BERT’s performance!

3. Trained via knowledge distillation (training)



1. A Lite BERT

- a. An attempt to optimize model performance / number of parameters
- b. 90% fewer parameters

2. Factorized embedding parameterization (architecture)

- a. Factorize token embeddings to make them much smaller

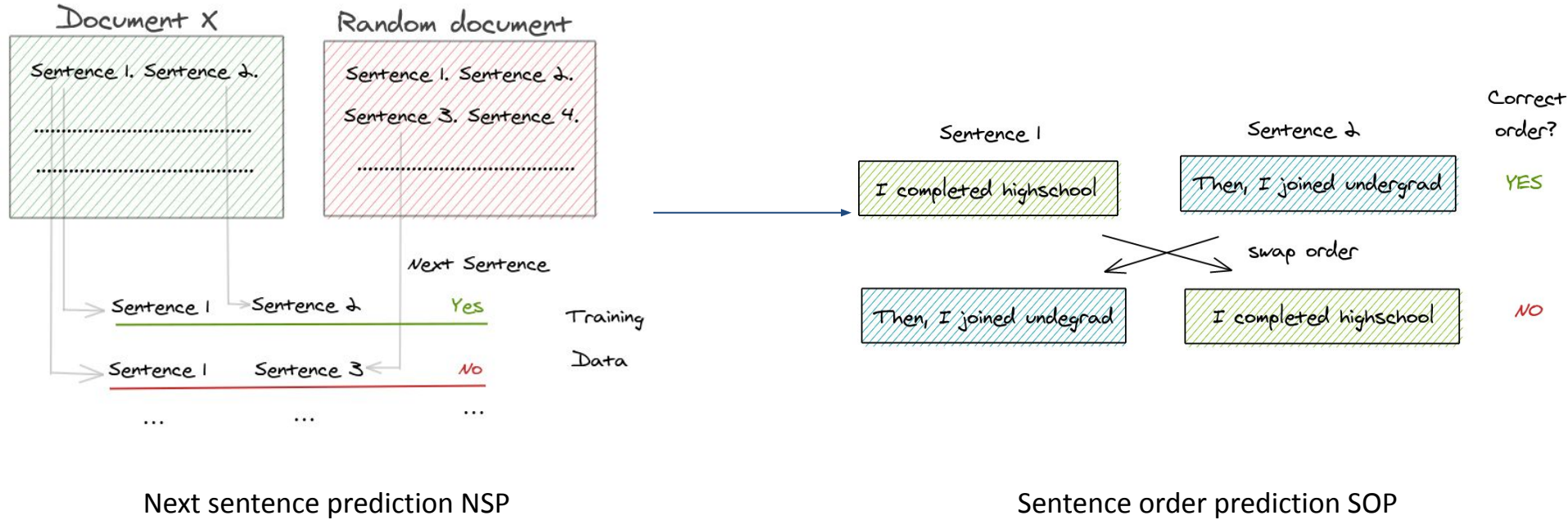
3. Cross-layer parameter sharing (architecture)

- a. share parameters across layers

4. Sentence order prediction > Next sentence prediction (training)

- a. Theory is that NSP and MLM are too similar of a task. SOP is harder
 - i. Take two consecutive parts from the same document as a positive class. Swap the order and use that as a negative example

ALBERT



Flavors of BERT

Sinan's
favorite to use
in production

Sinan's
favorite from
an academic
standpoint

Comparison	BERT October 11, 2018	RoBERTa July 26, 2019	DistilBERT October 2, 2019	ALBERT September 26, 2019
Parameters	Base: 110M Large: 340M	Base: 125M Large: 355M	Base: 66M	Base: 12M Large: 18M
Layers / Hidden Dimensions / Self-Attention Heads	Base: 12 / 768 / 12 Large: 24 / 1024 / 16	Base: 12 / 768 / 12 Large: 24 / 1024 / 16	Base: 6 / 768 / 12	Base: 12 / 768 / 12 Large: 24 / 1024 / 16
Training Time	Base: 8 x V100 x 12d Large: 280 x V100 x 1d	1024 x V100 x 1 day (4-5x more than BERT)	Base: 8 x V100 x 3.5d (4 times less than BERT)	[not given] Large: 1.7x faster
Performance	Outperforming SOTA in Oct 2018	88.5 on GLUE	97% of BERT-base's performance on GLUE	89.4 on GLUE
Pre-Training Data	BooksCorpus + English Wikipedia = 16 GB	BERT + CCNews + OpenWebText + Stories = 160 GB	BooksCorpus + English Wikipedia = 16 GB	BooksCorpus + English Wikipedia = 16 GB
Method	Bidirectional Transformer, MLM & NSP	BERT without NSP, Using Dynamic Masking	BERT Distillation	BERT with reduced parameters & SOP (not NSP)

Summary

Transformers are a next-generation architecture relying on stacks of attention mechanisms to encode and decode sequences of tokens

BERT is an auto-encoding language model and state of the art natural language understanding architecture that understands context via the transformer's encoder stack and pre-training on the MLM and the NSP tasks. It can be fine-tuned to several NLP tasks

Flavors of BERT have popped up ever since to address deficiencies with vanilla BERT like *ALBERT* and *DistilBERT*

Next Steps

1. There's always a deeper level to the math behind attention, transformers and BERT
2. More ways to fine-tune BERT
3. Different flavors of BERT - XLNET, etc
4. How Transformer decoders are used to perform Natural Language Generation (eg. GPT family)

Thank you + Final Q/A!

Please feel free to reach out (in/sinan-ozdemir + @prof_oz)

