Lecture 10: Attention mechanism

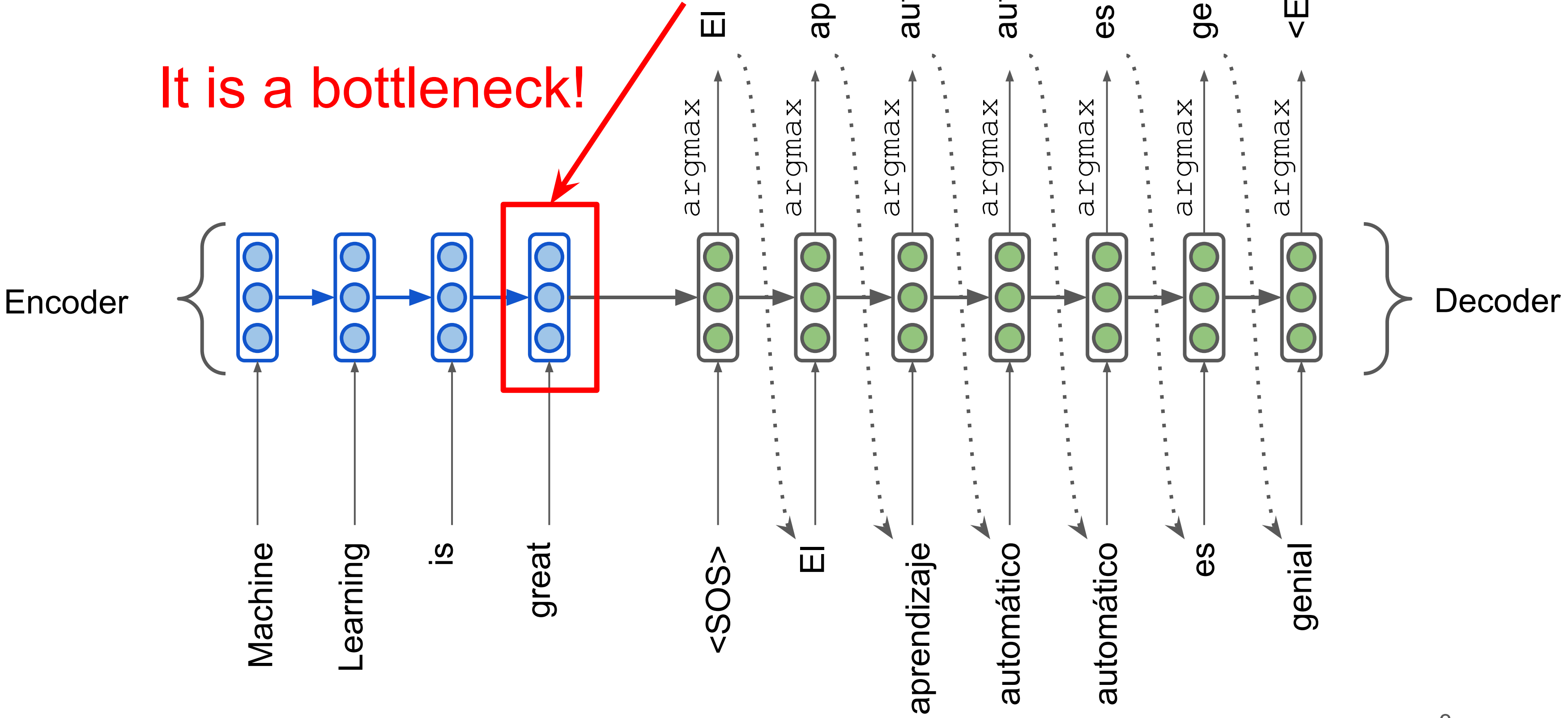
Radoslav Neychev

Attention

Seq2seq NMT

This state encodes the whole sentence

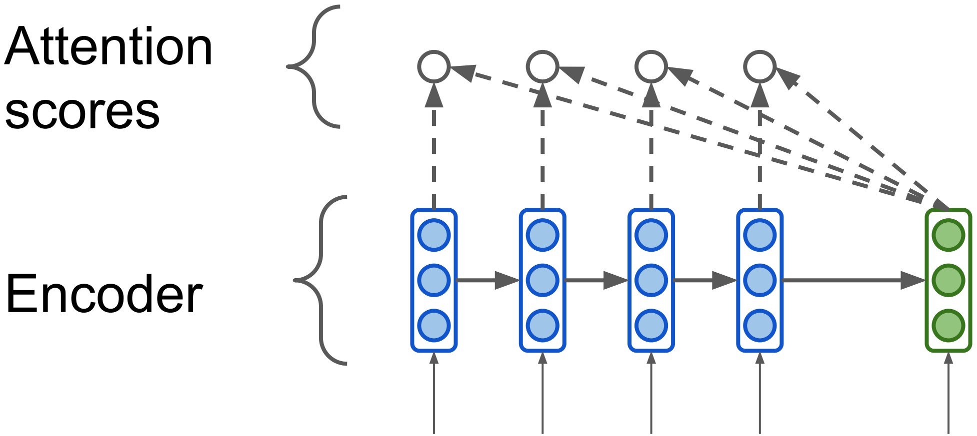
It is a bottleneck!



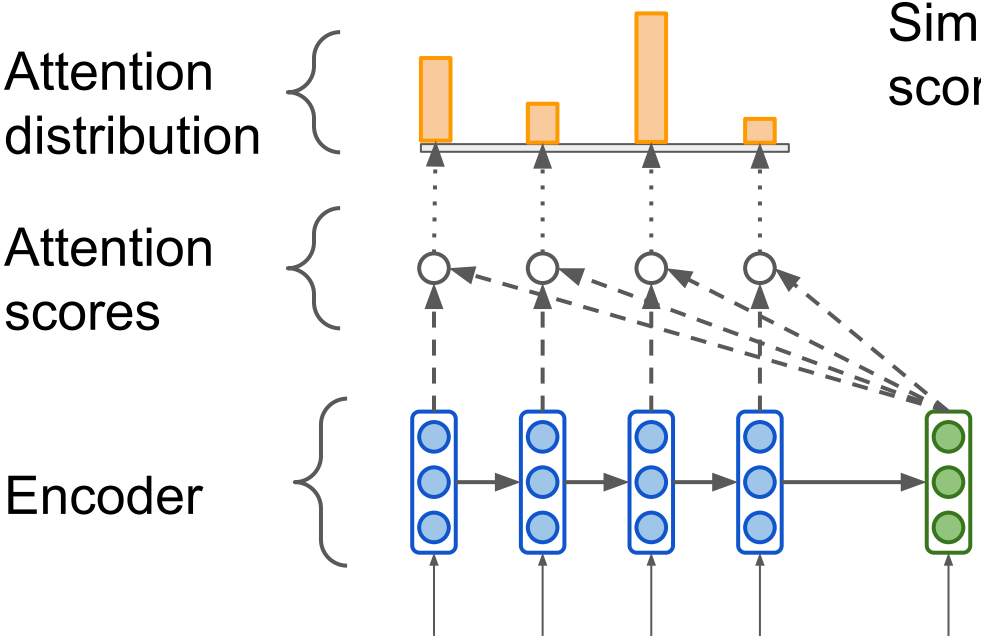
Main idea:

on each step of the **decoder**, use **direct connection to the encoder** to focus on a particular part of the source sequence

Seq2seq with attention

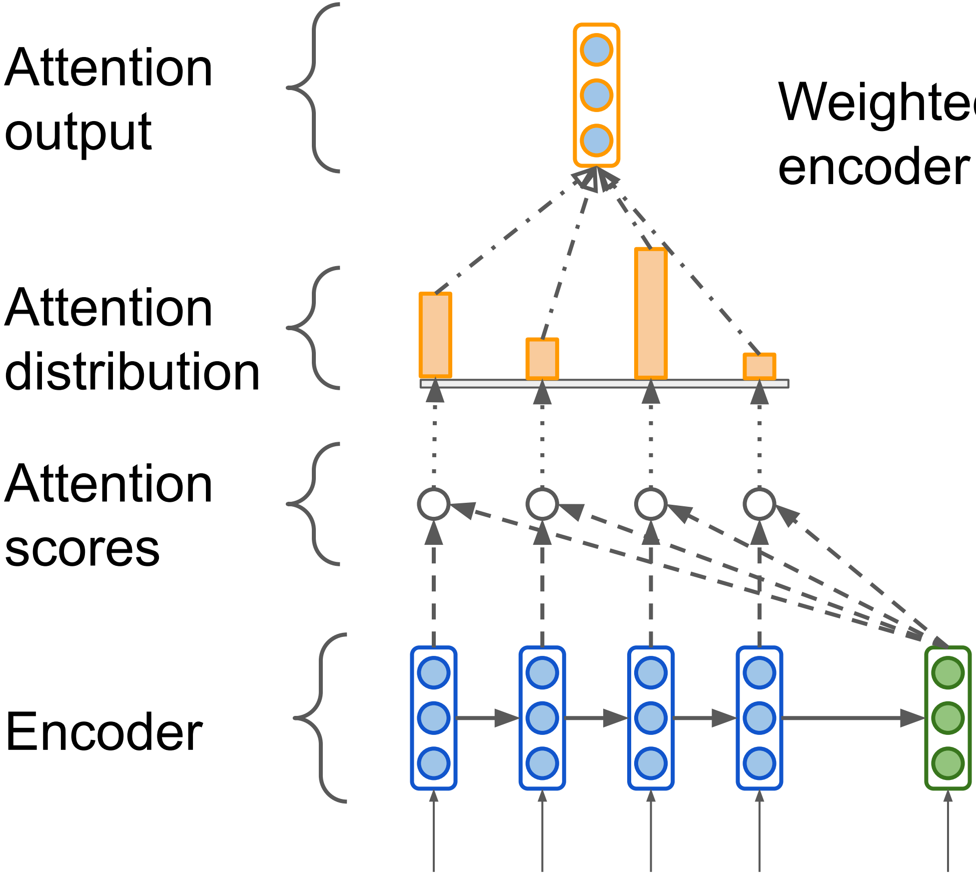


Seq2seq with attention



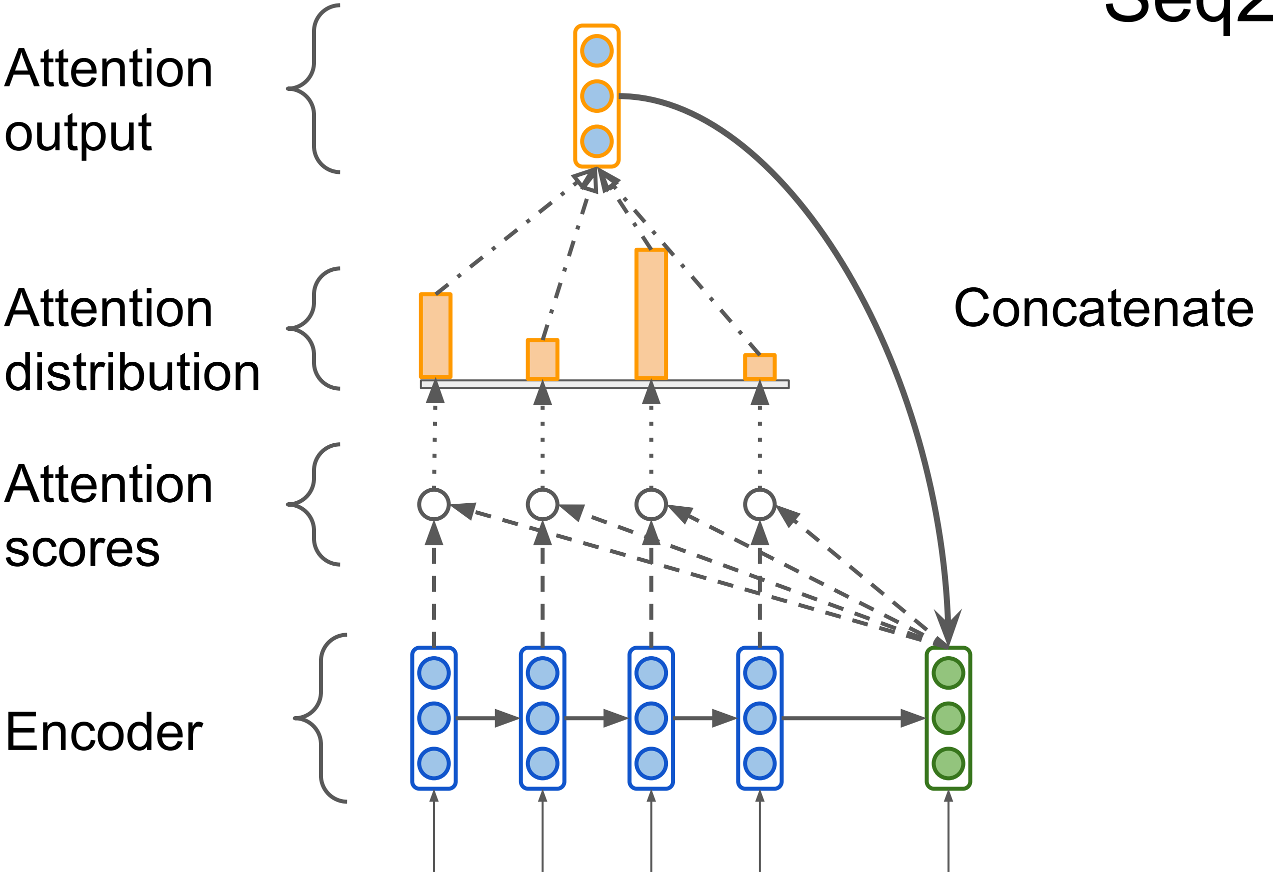
Simply apply softmax to scores

Seq2seq with attention

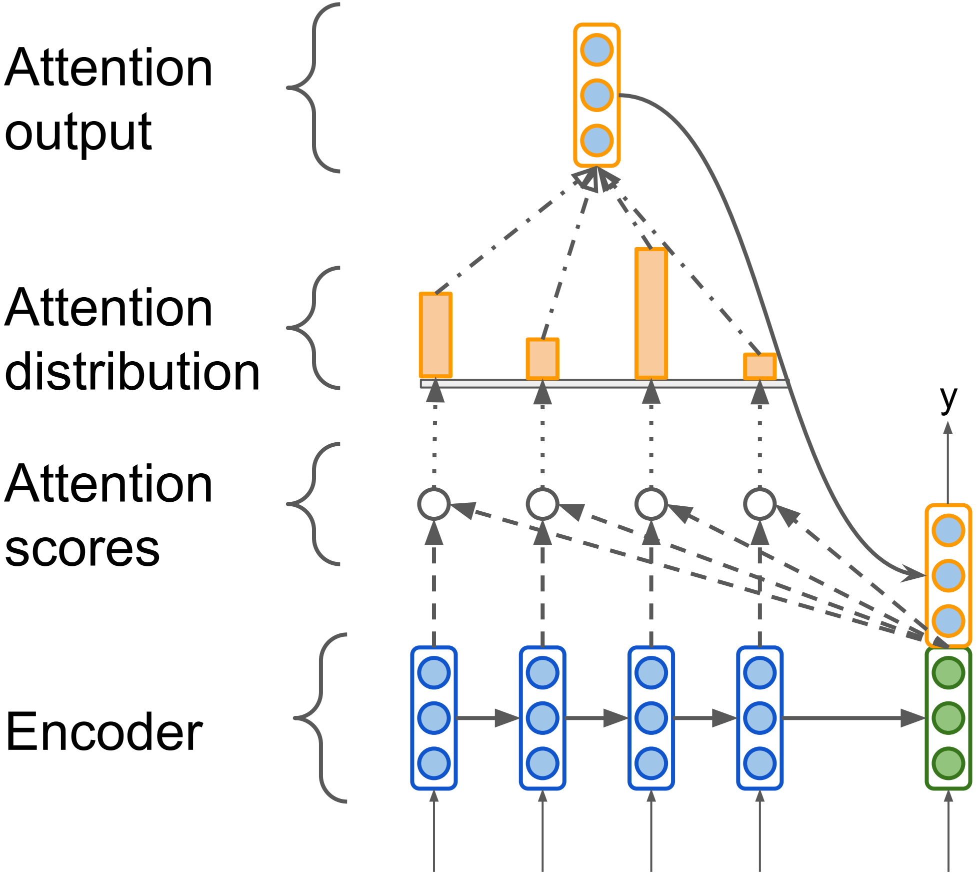


Weighted sum of all encoder states

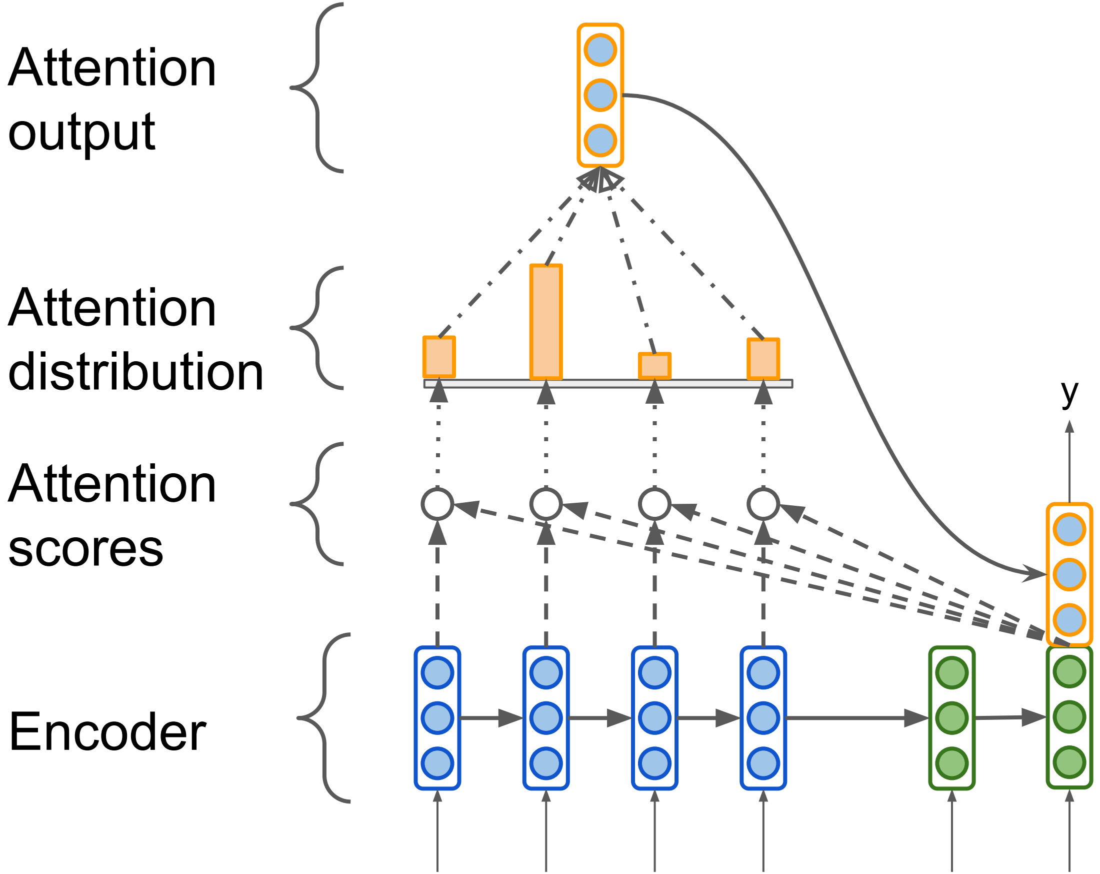
Seq2seq with attention



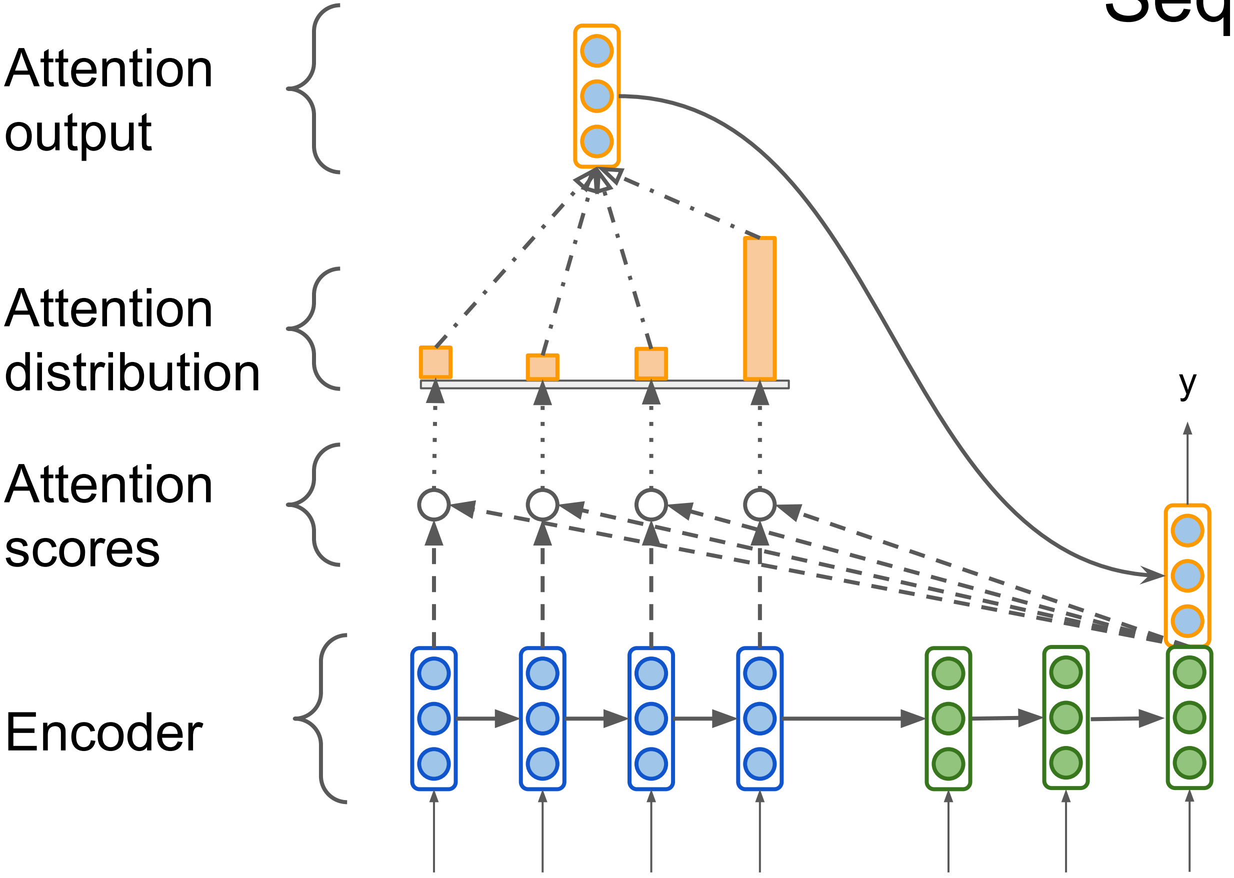
Seq2seq with attention



Seq2seq with attention



Seq2seq with attention



Attention in equations

Denote encoder hidden states $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \mathbb{R}^k$

and decoder hidden state at time step t $\mathbf{s}_t \in \mathbb{R}^k$

The attention scores $\mathbf{e}^t$ can be computed as dot product

$$\mathbf{e}^t = [\mathbf{s}^T \mathbf{h}_1, \ldots, \mathbf{s}^T \mathbf{h}_N]$$

Then the attention vector is a linear combination of encoder states

$$\mathbf{a}_t = \sum_{i=1}^{N} \boldsymbol{\alpha}_i^t \mathbf{h}_i \in \mathbb{R}^k$$, where $\boldsymbol{\alpha}_t = \mathrm{softmax}(\mathbf{e}_t)$

Attention provides interpretability

- We may see what the decoder was focusing on
- We get word alignment for free!

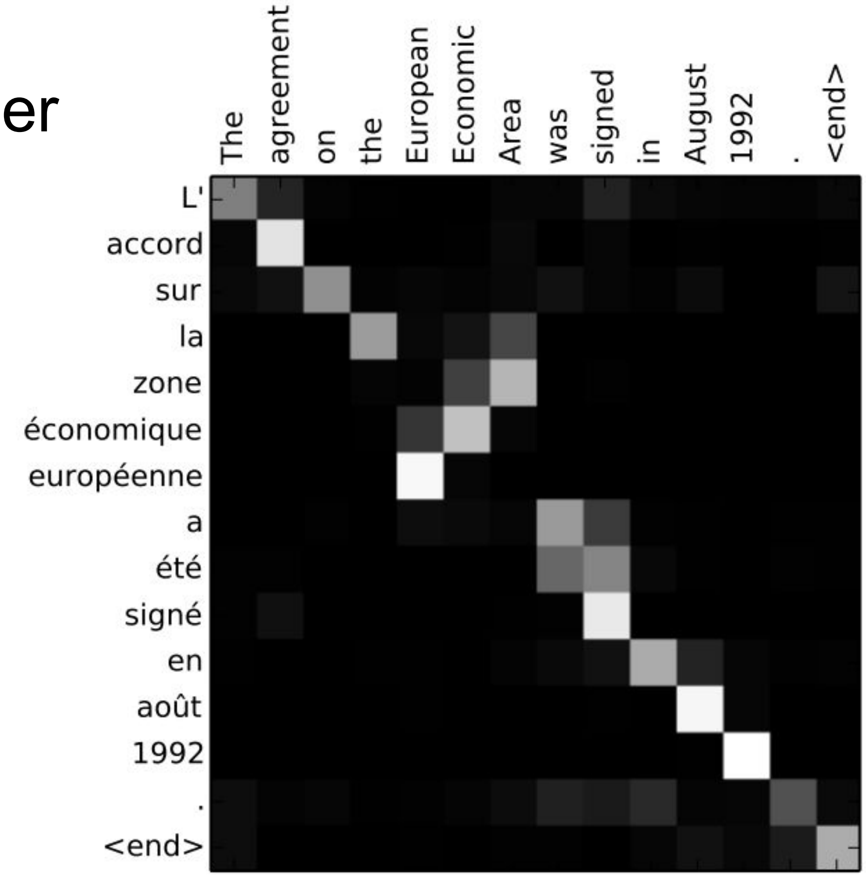
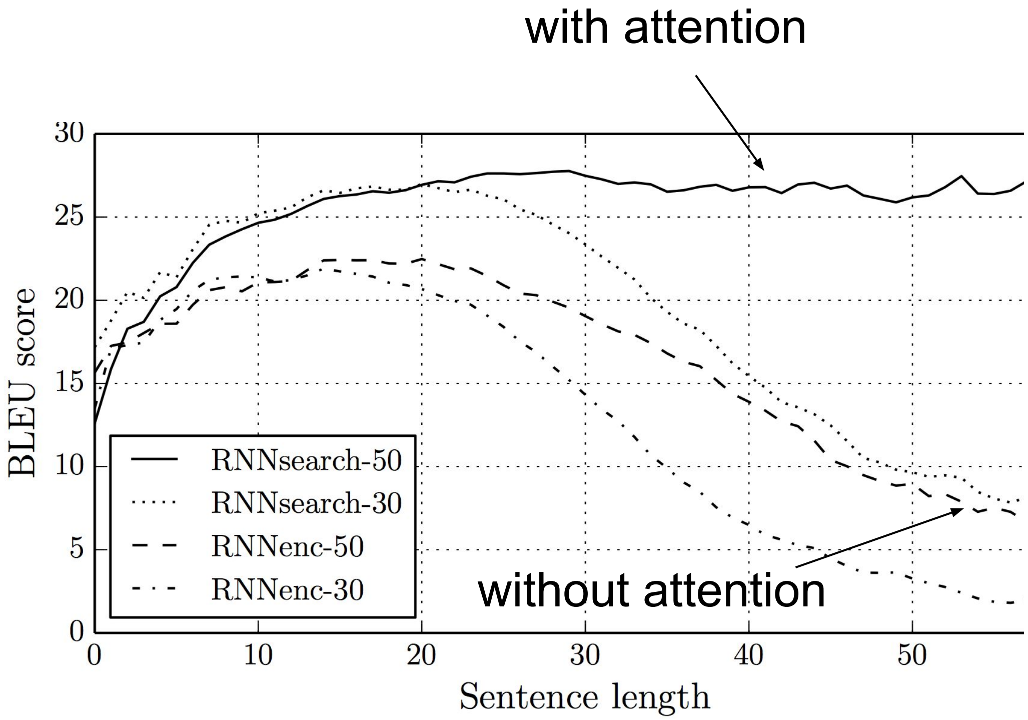


Image source: Neural Machine Translation by Jointly Learning to Align and Translate

Attention advantages

- “Free” word alignment
- Better results on long sequences



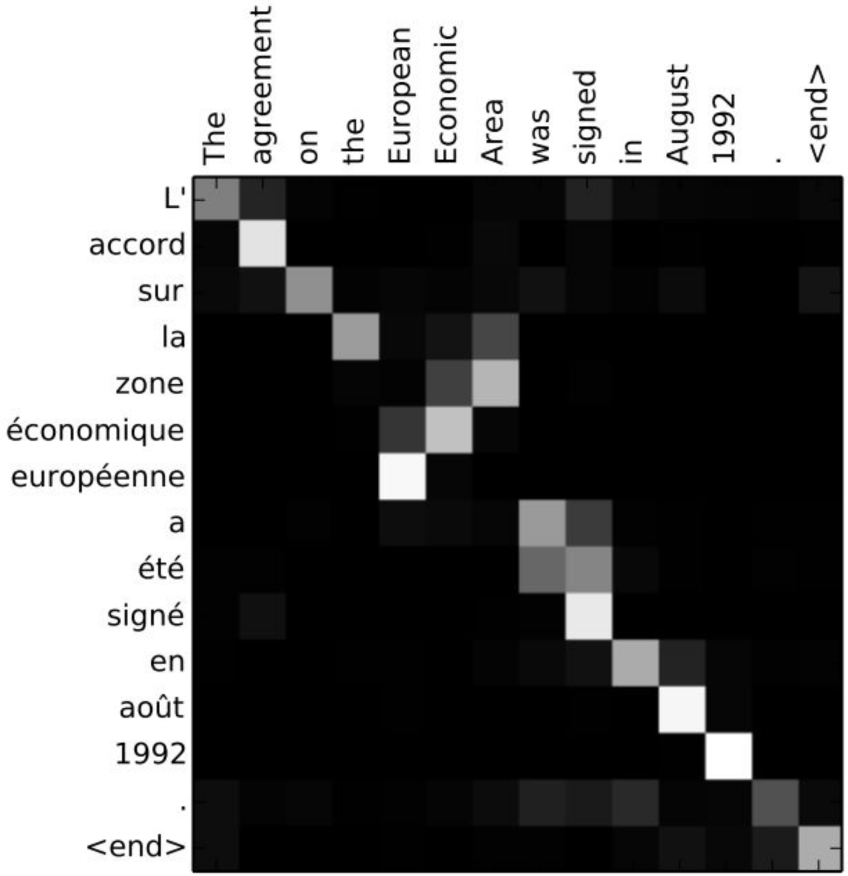


Image source: [Neural Machine Translation by Jointly Learning to Align and Translate]

Attention variants

- Basic dot-product (the one discussed before): $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
- Multiplicative attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
  - $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ - weight matrix
- Additive attention: $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$
  - $\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ - weight matrices
  - $\boldsymbol{v} \in \mathbb{R}^{d_3}$ - weight vector

Self-Attention

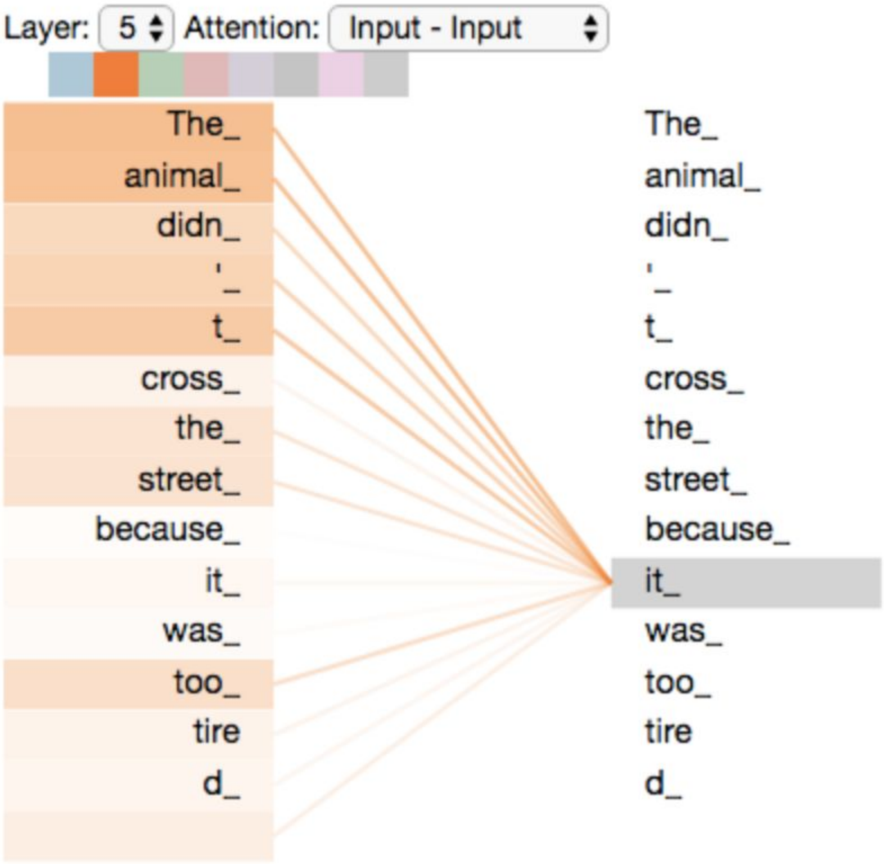
Self-Attention at a High Level

”`The animal didn't cross the street because it was too tired`”

- What does “it” in this sentence refer to?
- We want self-attention to associate “it” with “animal”

- Self-attention is the method the Transformer uses to bake the “understanding” of other relevant words into the one we’re currently processing

Self-Attention at a High Level



Self-Attention: detailed explanation

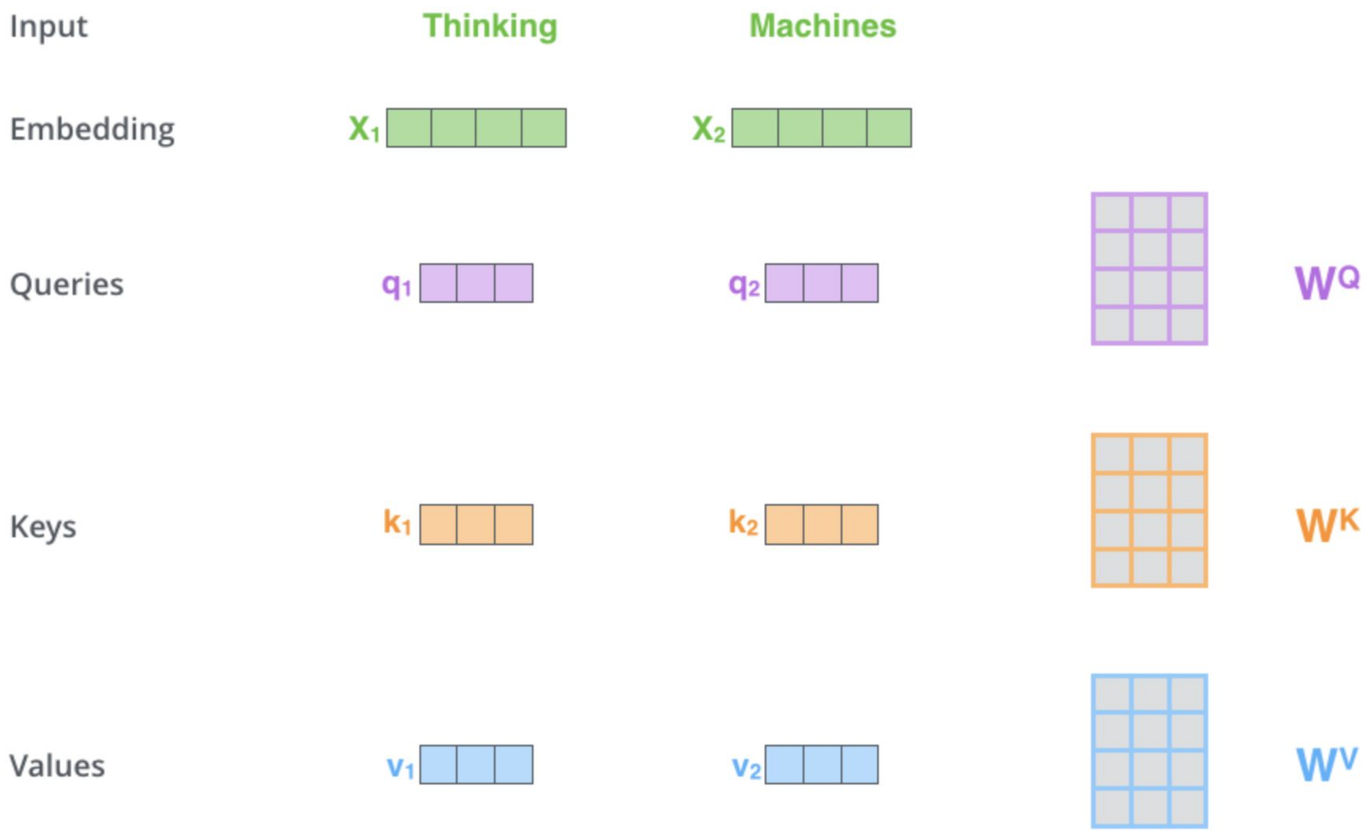


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 1:

create 3 vectors (**query**, **key**, **value**)

from each of the encoder's input vectors

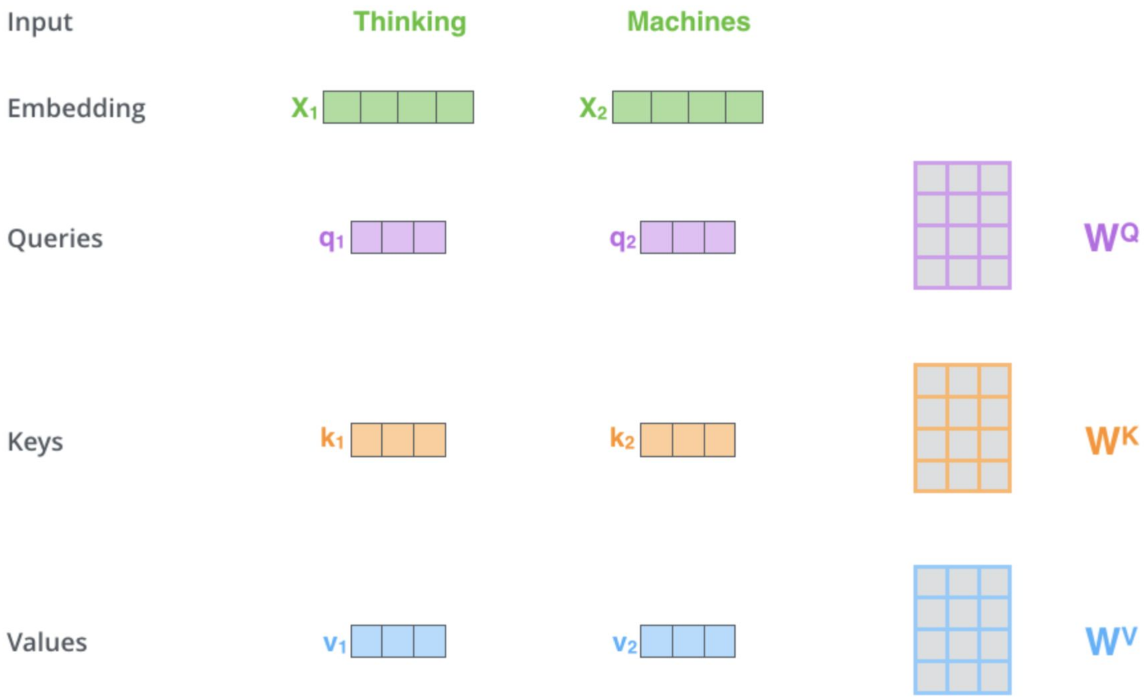


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

What are the **query**, **key**, **value** vectors?

They’re abstractions that are useful for

calculating and thinking about attention.

Self-Attention: detailed explanation

STEP 2:

calculate a score

(score each word of the input sentence against the current word)

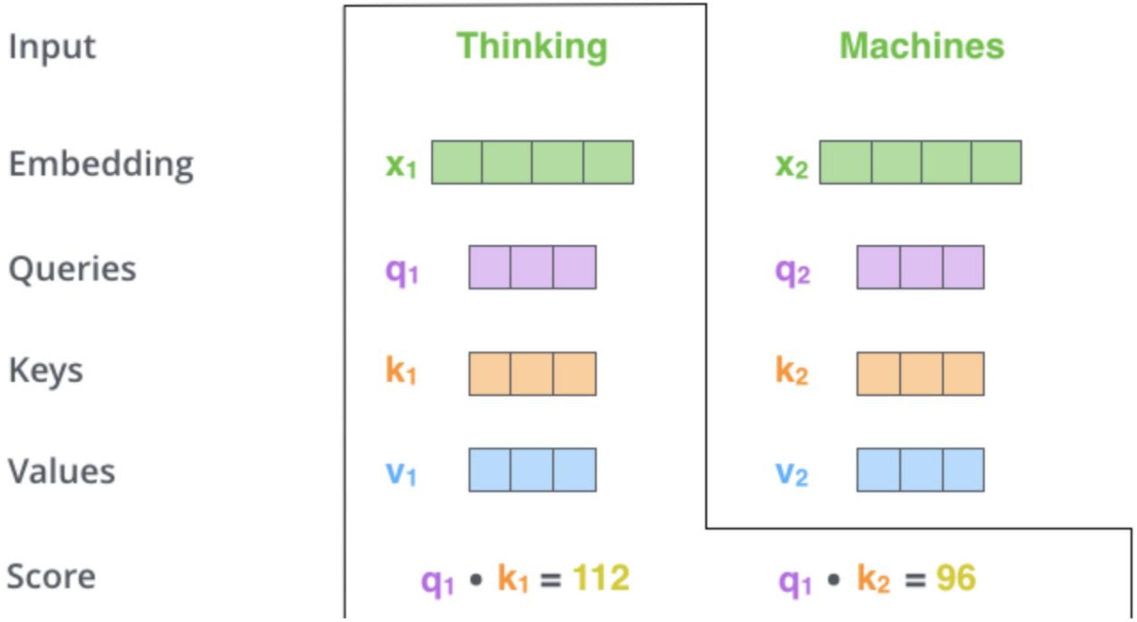


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 3:

divide the scores by 8

(the square root of the
dimension of the key vectors)

STEP 4:

softmax

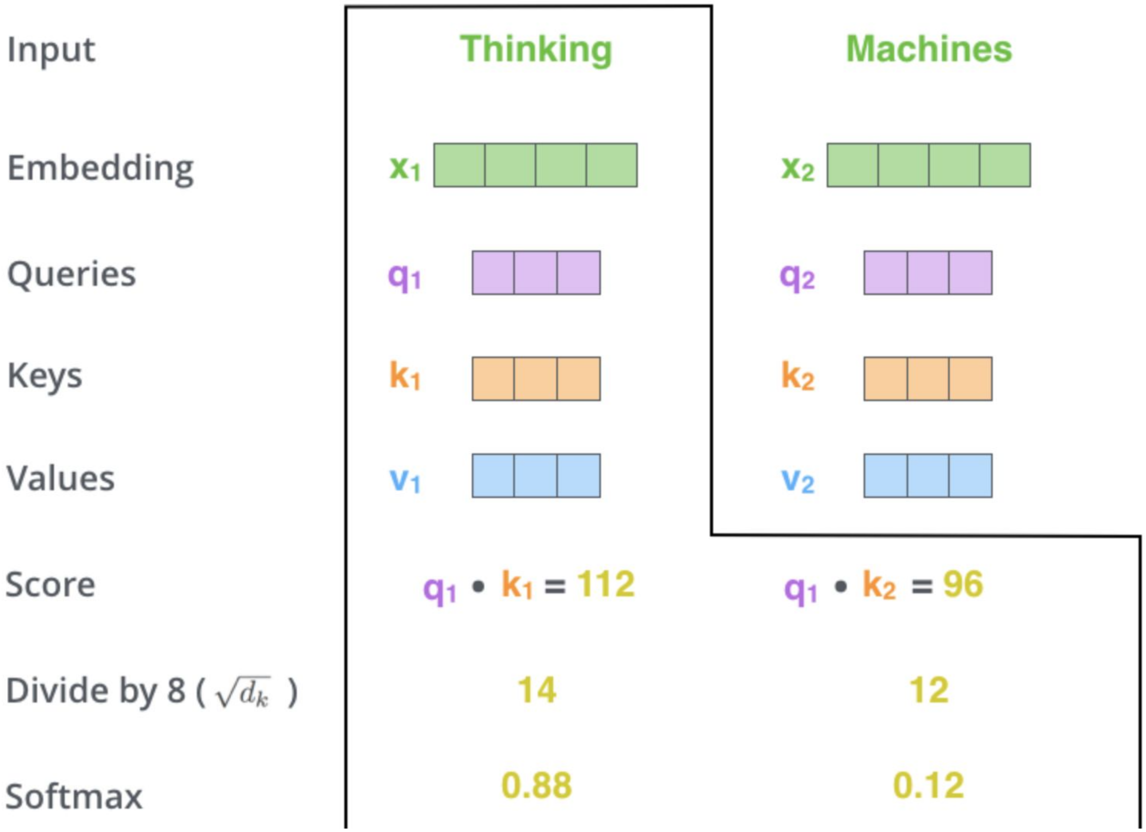


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 5:

multiply each value vector by the softmax score

STEP 6:

sum up the weighted value vectors

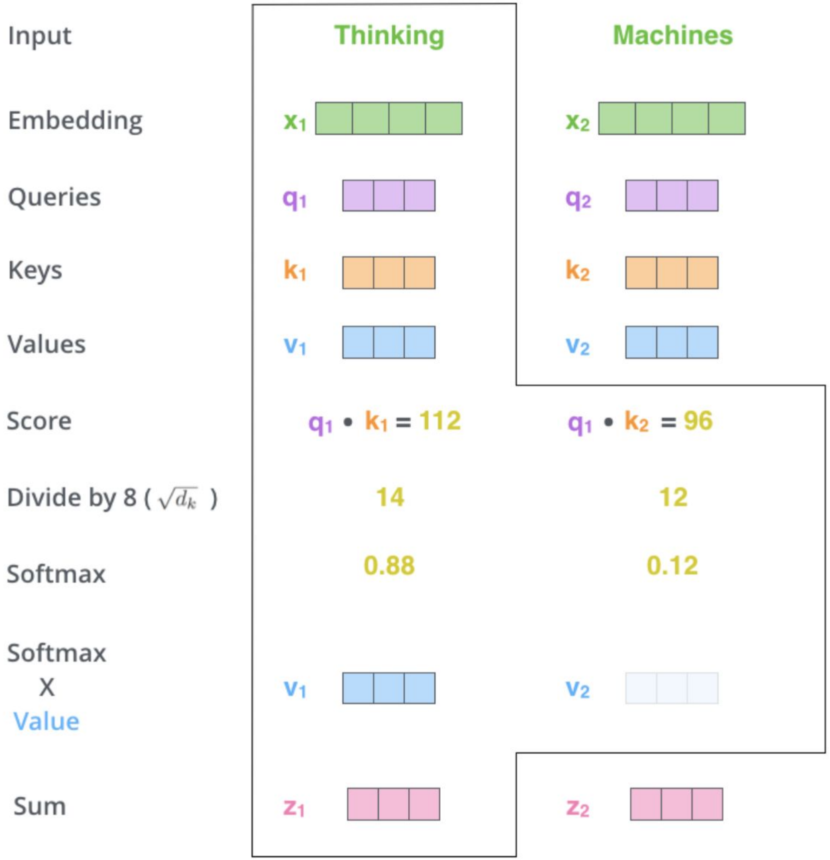
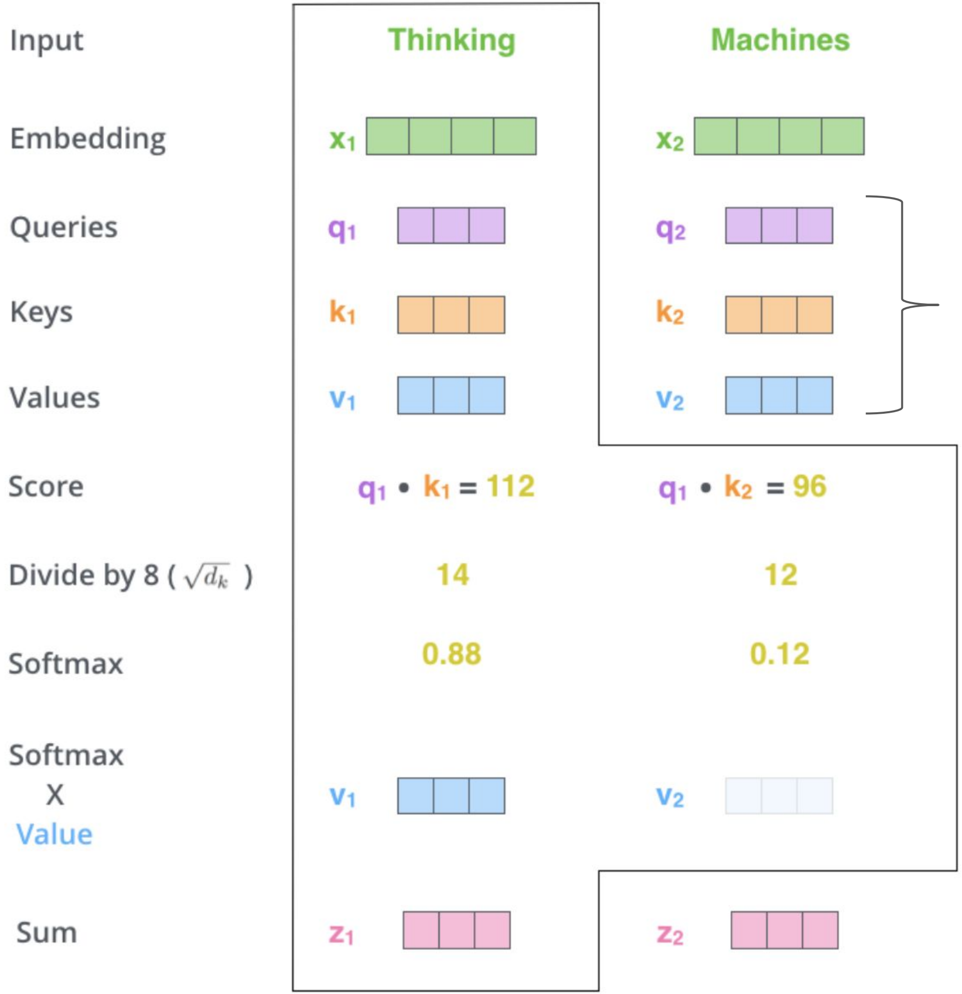


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention



STEP 1: create Query, Key, Value

STEP 2: calculate scores

STEP 3: divide by $\sqrt{d_k}$

STEP 4: softmax

STEP 5: multiply each value vector by the softmax score

STEP 6: sum up the weighted value vectors

Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: Matrix Calculation

Pack embeddings into matrix **X**

Multiply **X** by weight matrices we've trained (**Wk, Wq, Wv**)

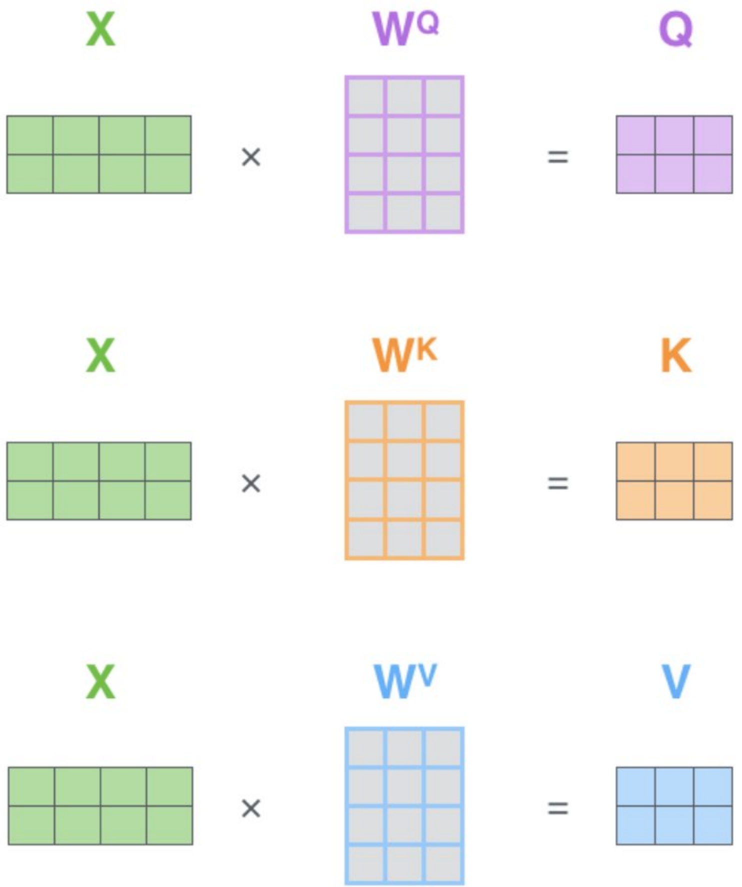


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: Matrix Calculation

$$\mathrm{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

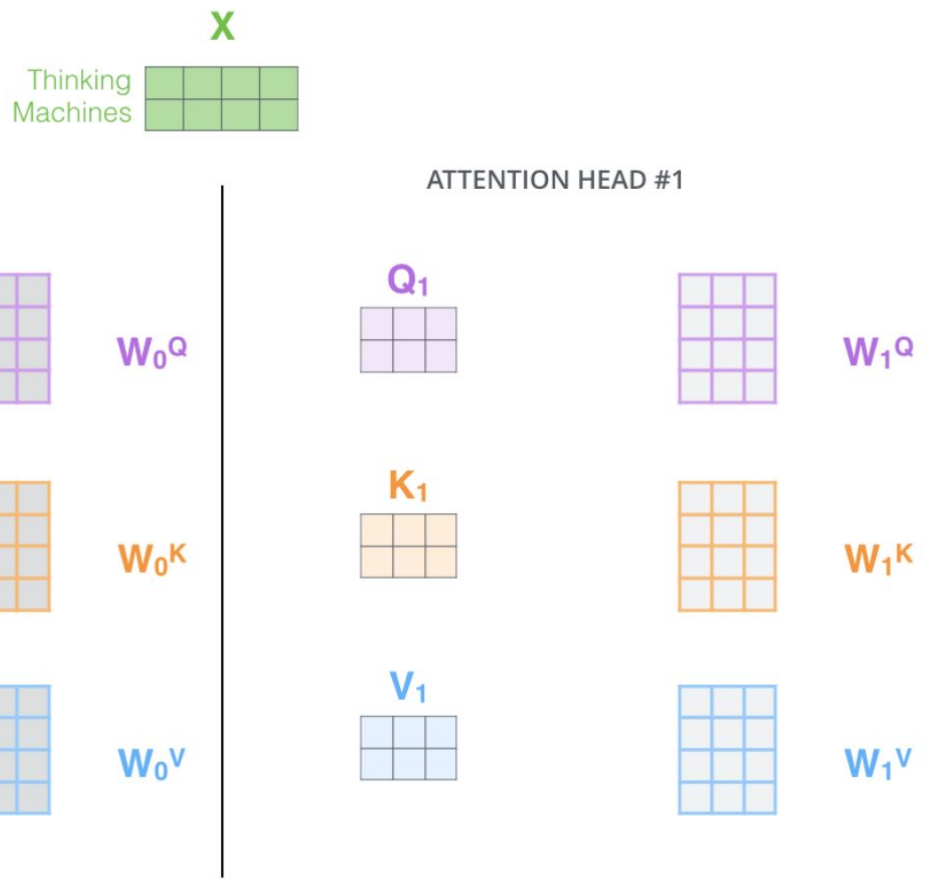


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

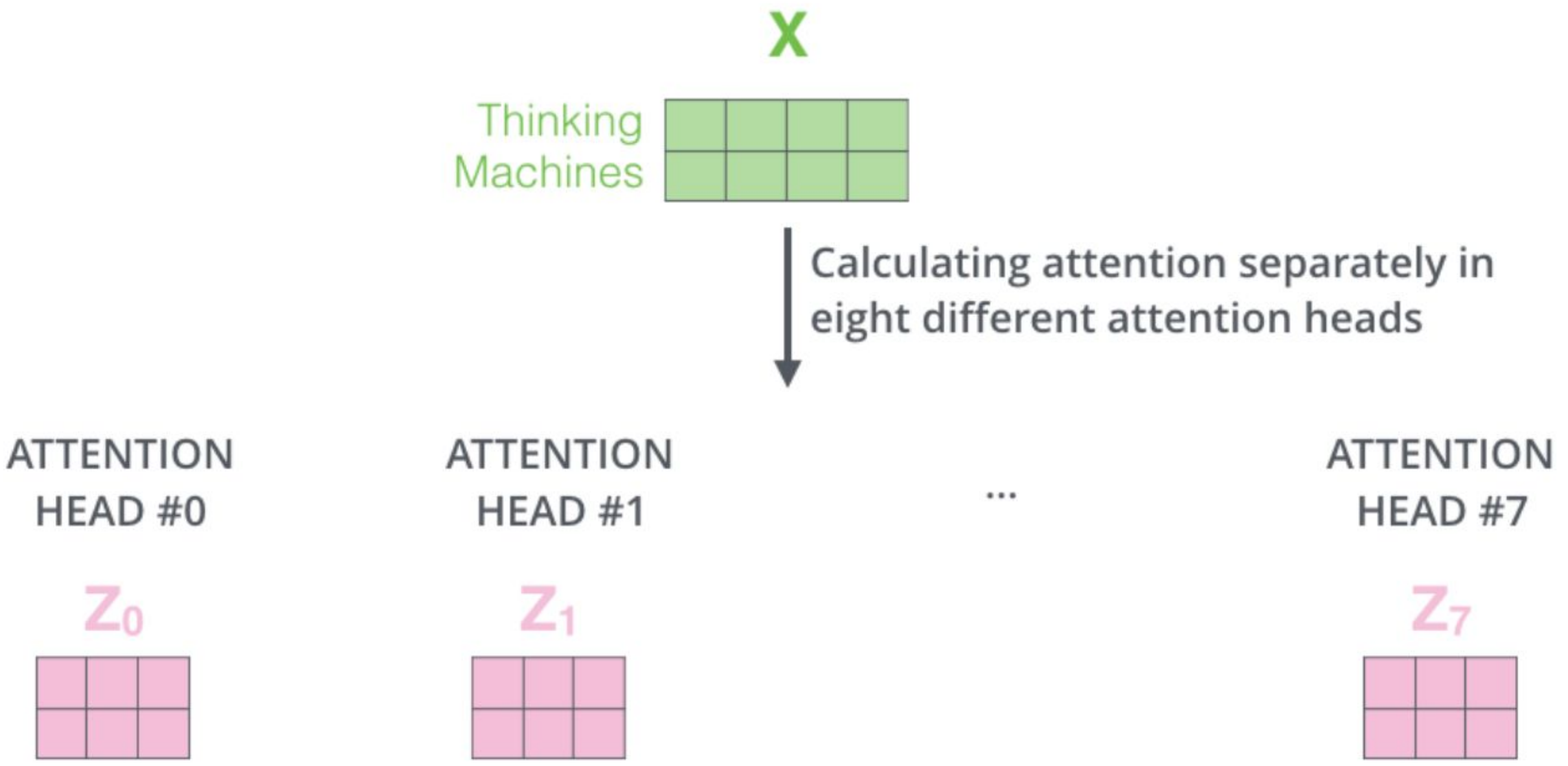


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

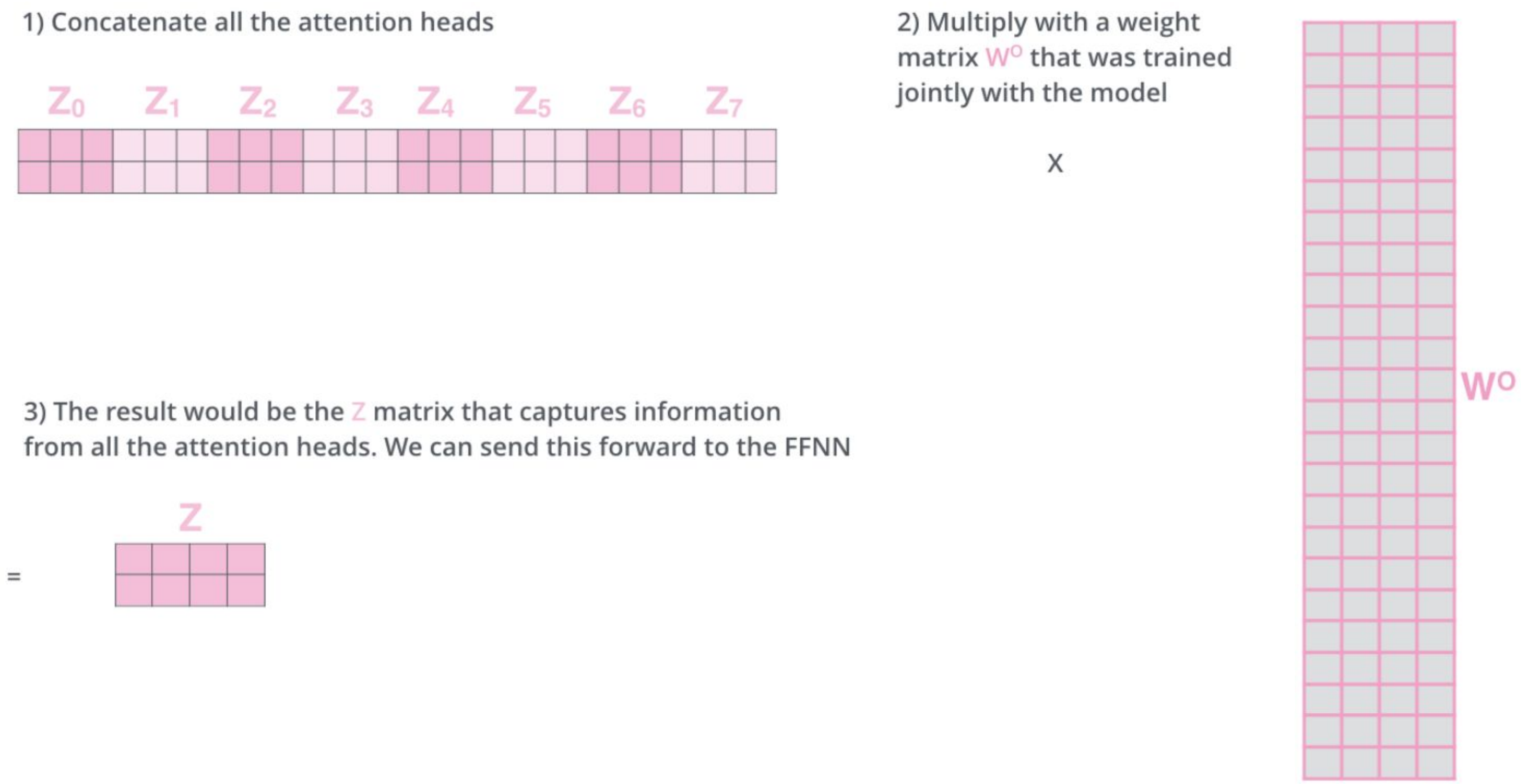


Image source: https://jalammar.github.io/illustrated-transformer/

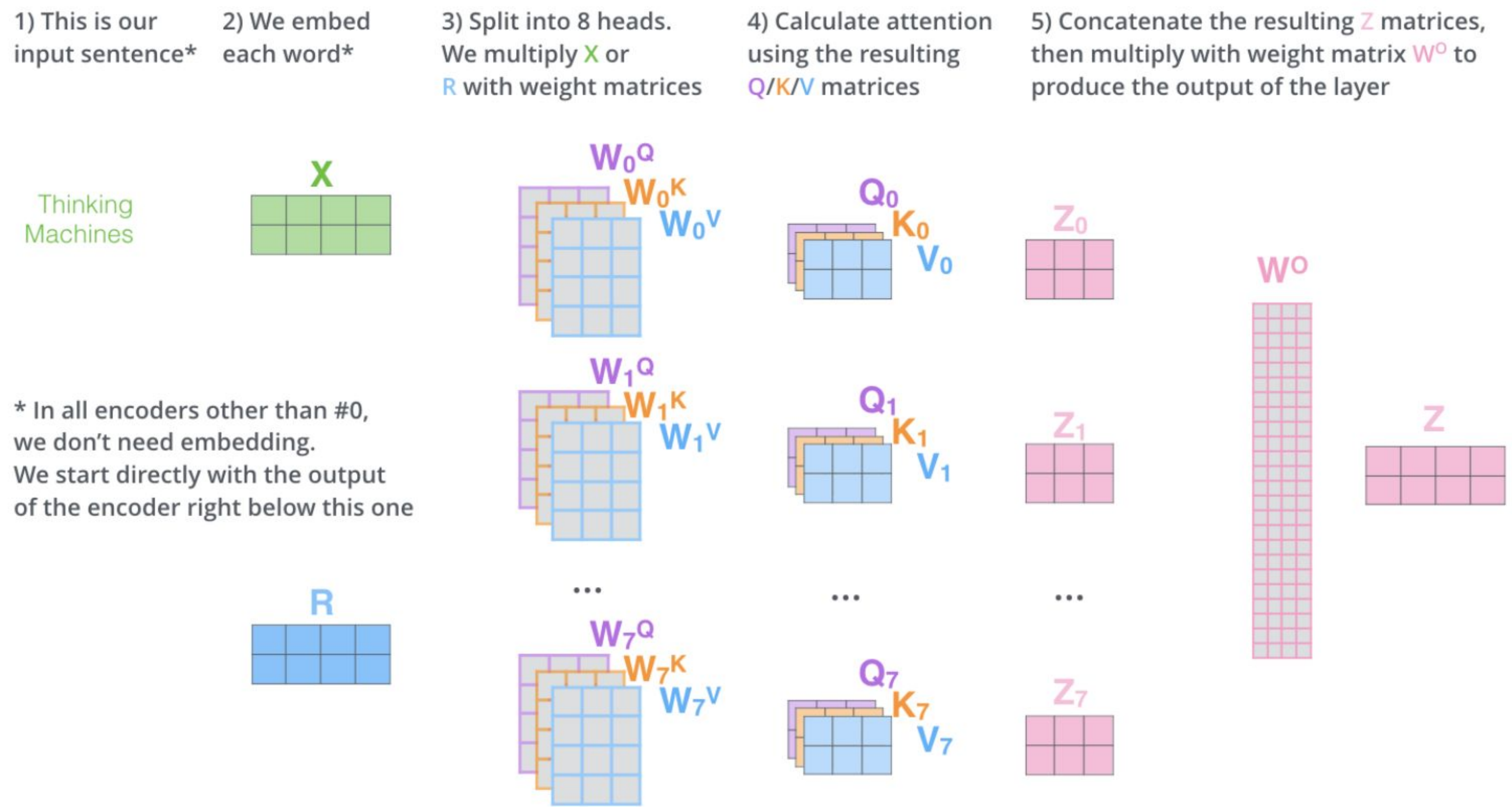
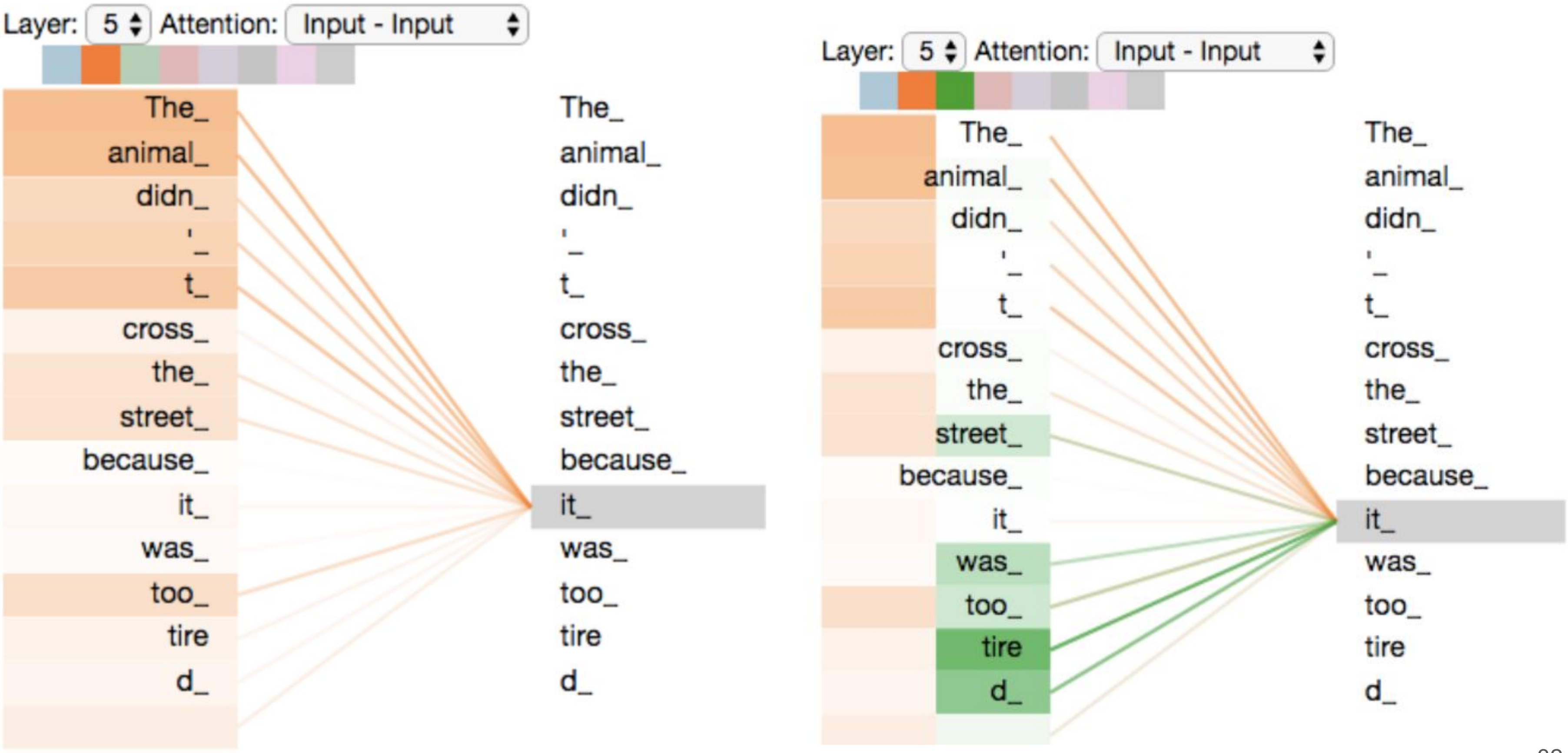
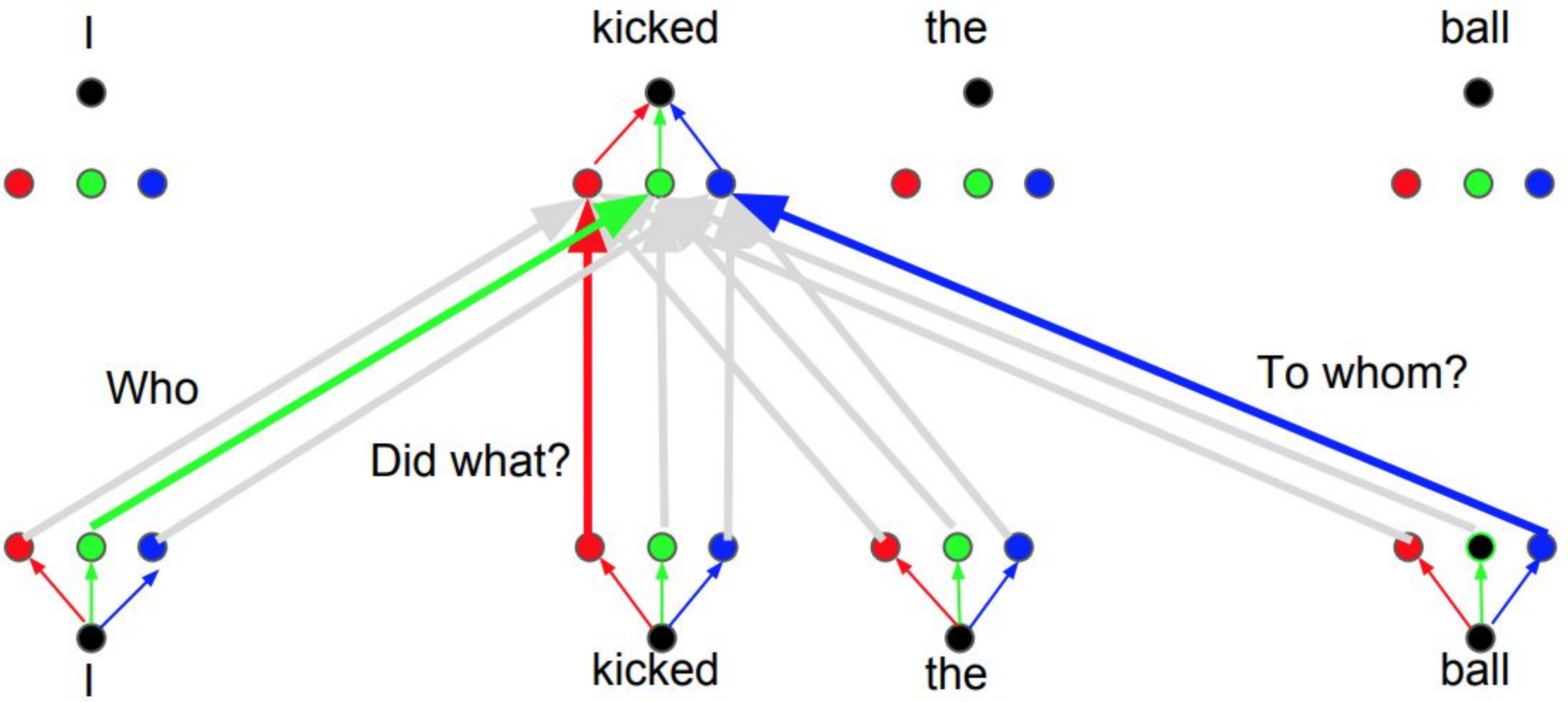


Image source: https://jalammar.github.io/illustrated-transformer/

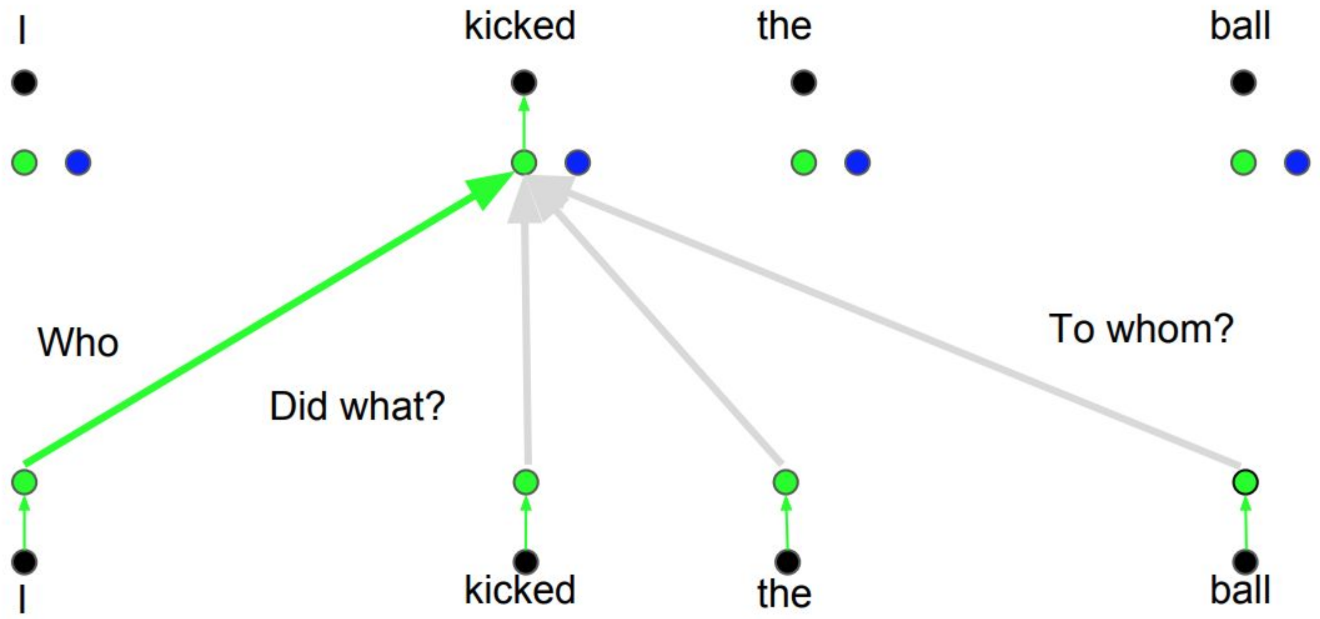
Multi-Head Attention



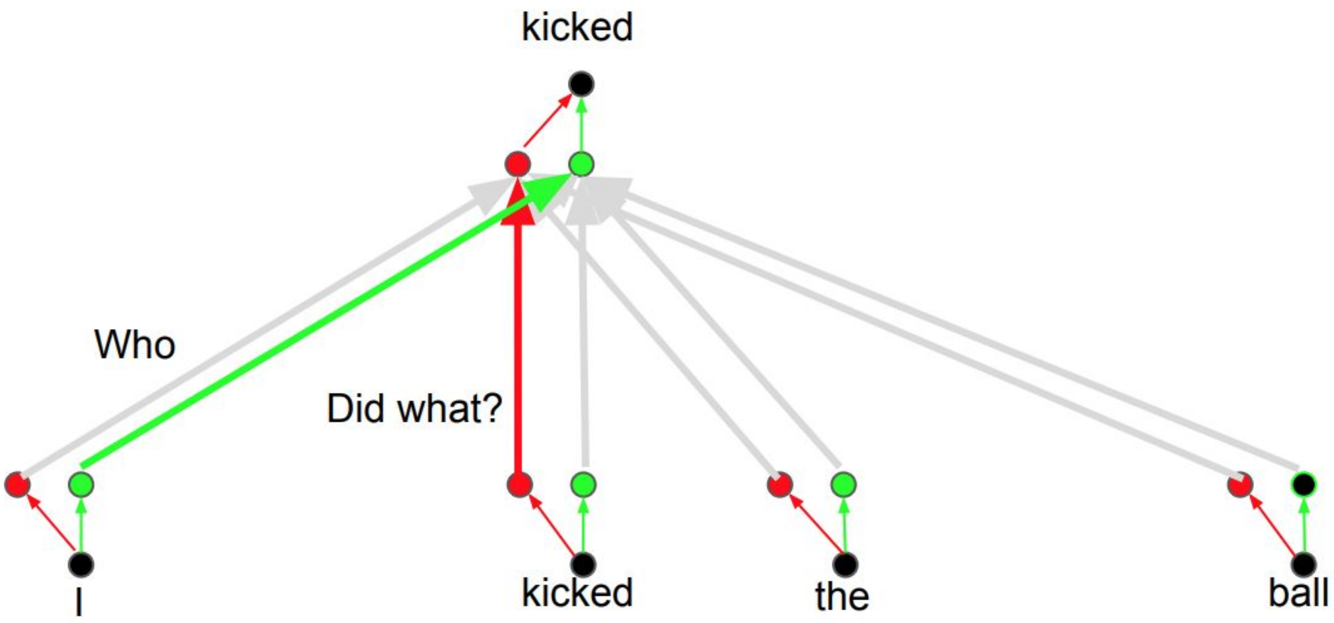
Why Multi-Head Attention?



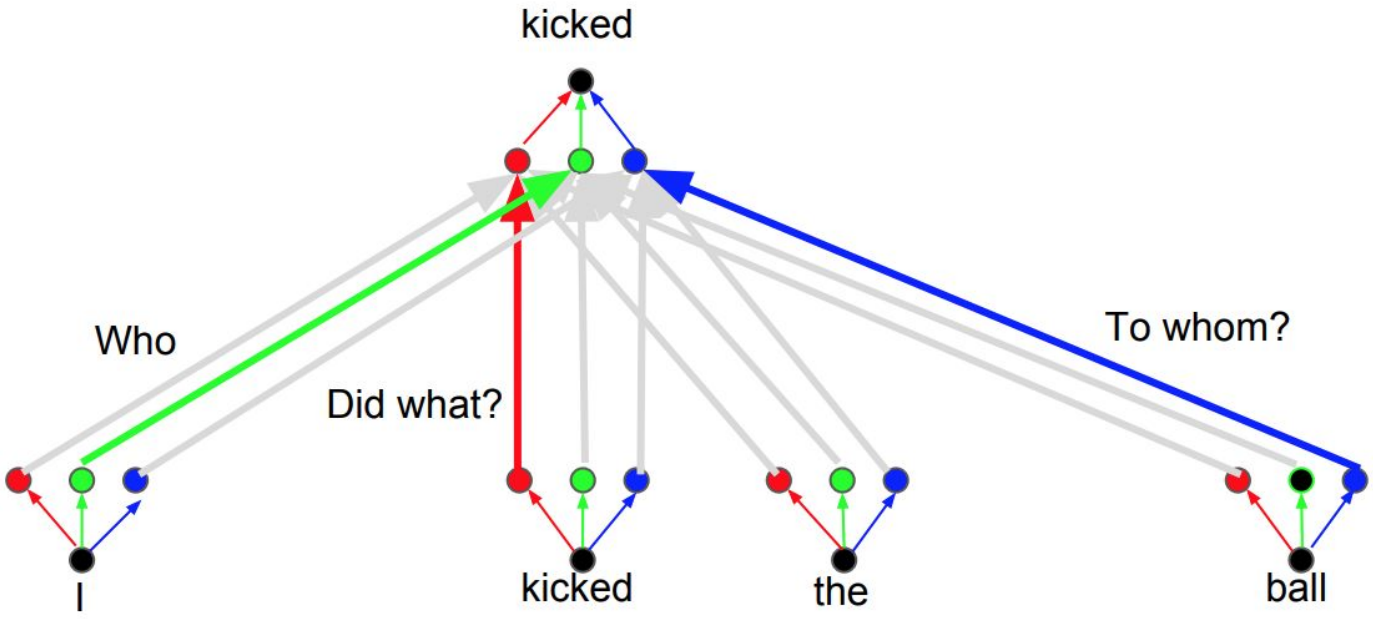
Attention head: Who



Attention head: Did What?

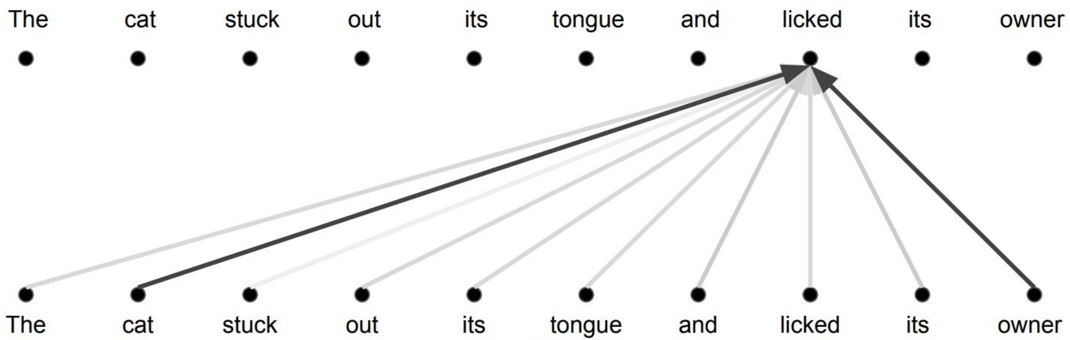


Attention head: To Whom?



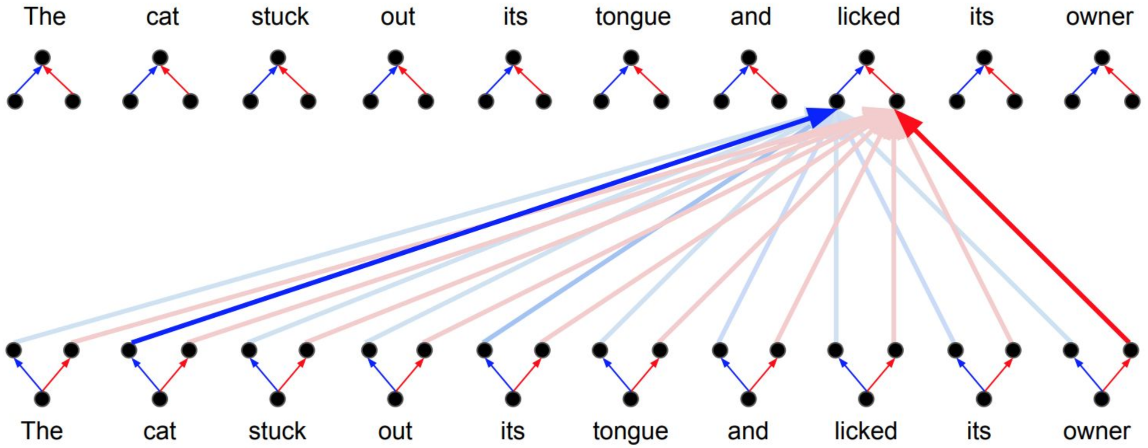
Attention vs. Multi-Head Attention

Attention: a weighted average



Multi-Head Attention:

parallel attention layers with different linear transformations on input and output.



Transformer outro

The Transformer

Image source: https://jalammar.github.io/illustrated-transformer/

The Transformer

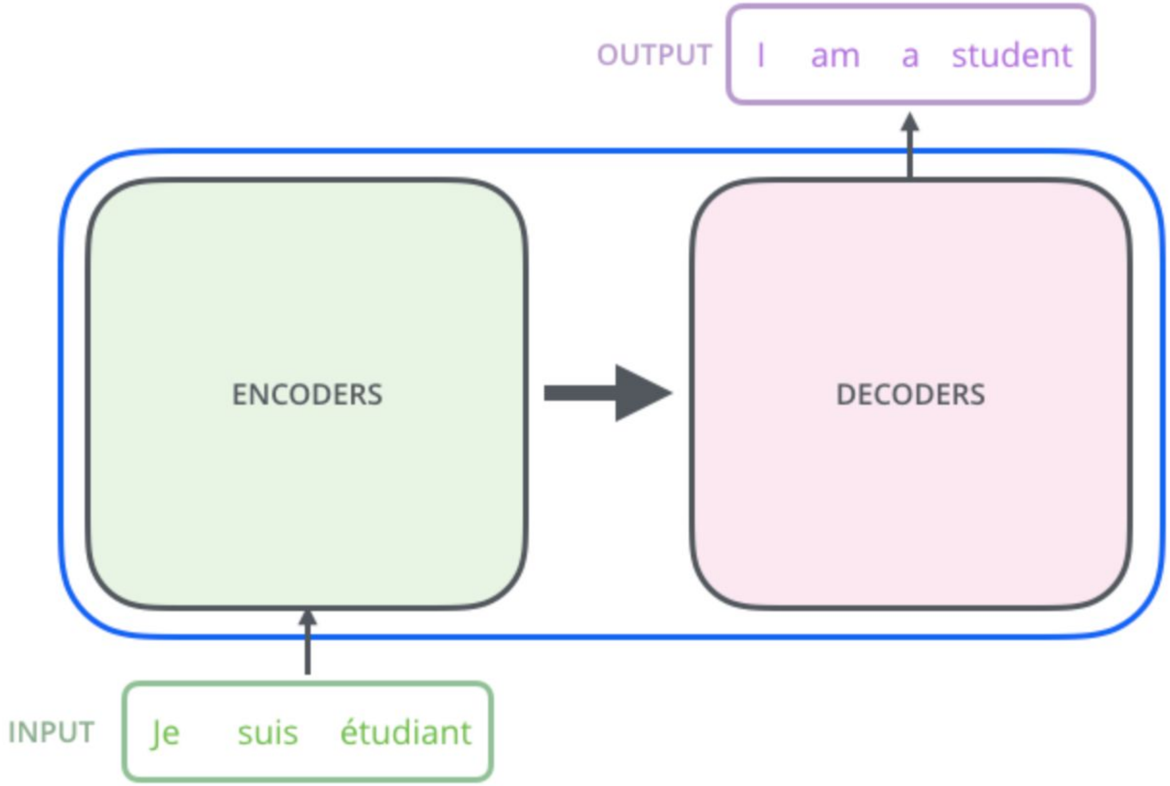


Image source: https://jalammar.github.io/illustrated-transformer/

The Transformer

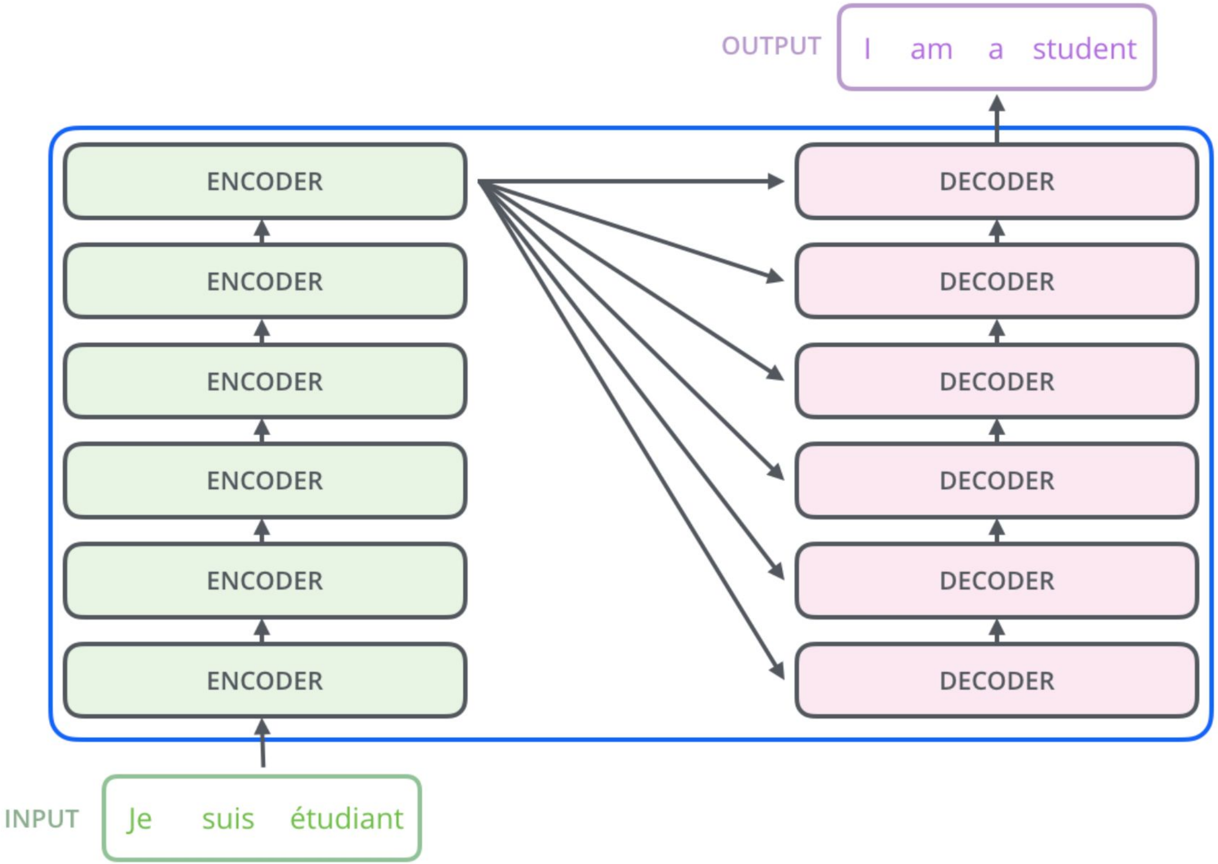
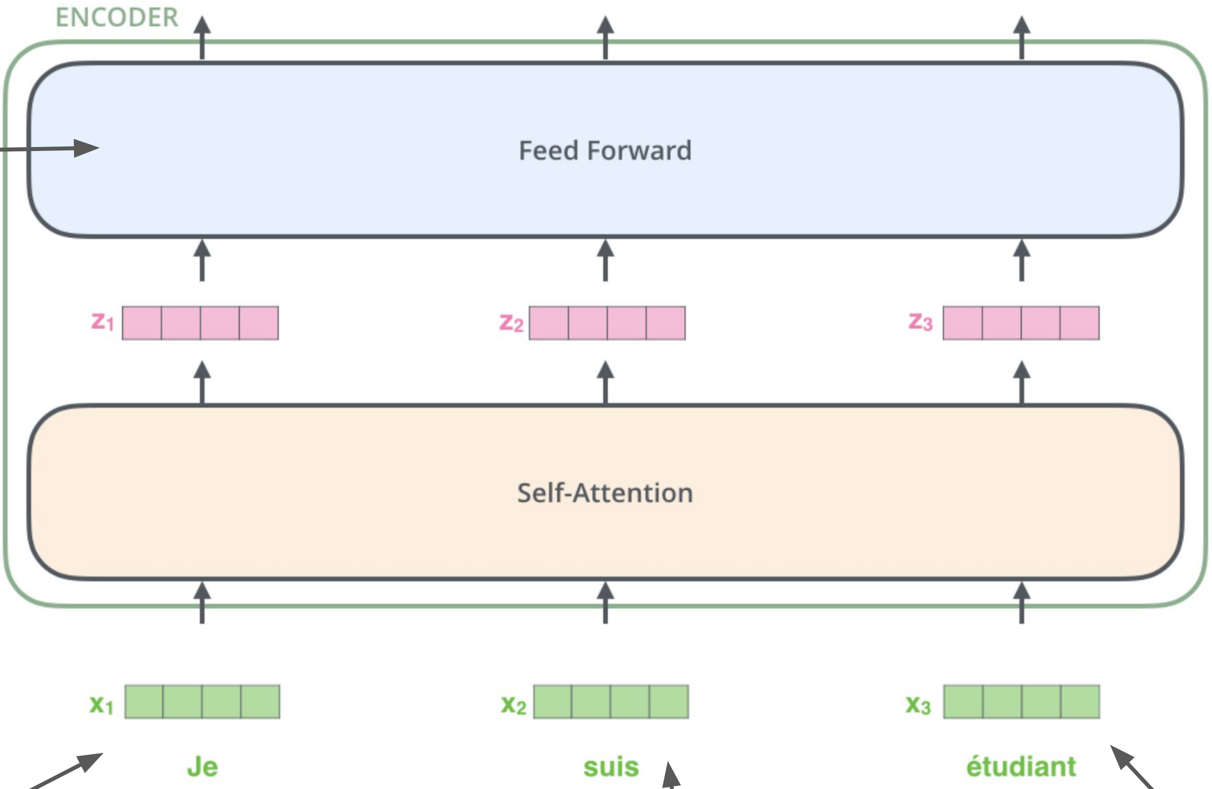


Image source: https://jalammar.github.io/illustrated-transformer/

The Encoder Side

Can be parallelized



the word in each position flows through its own path in the encoder

Image source: https://jalammar.github.io/illustrated-transformer/

BERT

Bidirectional Encoder Representations from Transformers

Model inputs

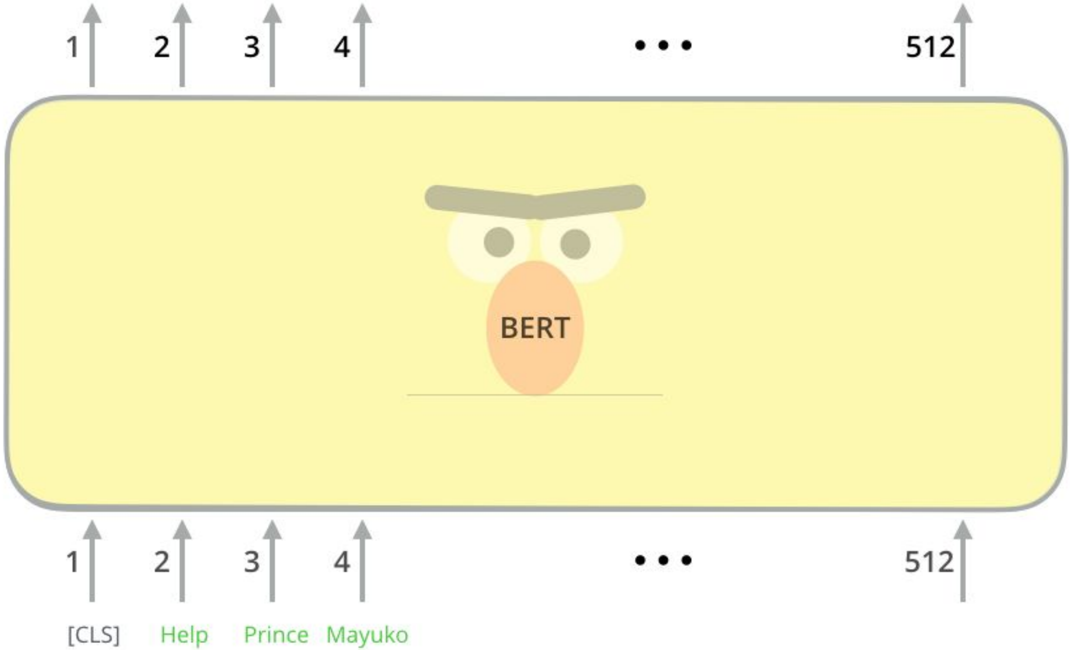
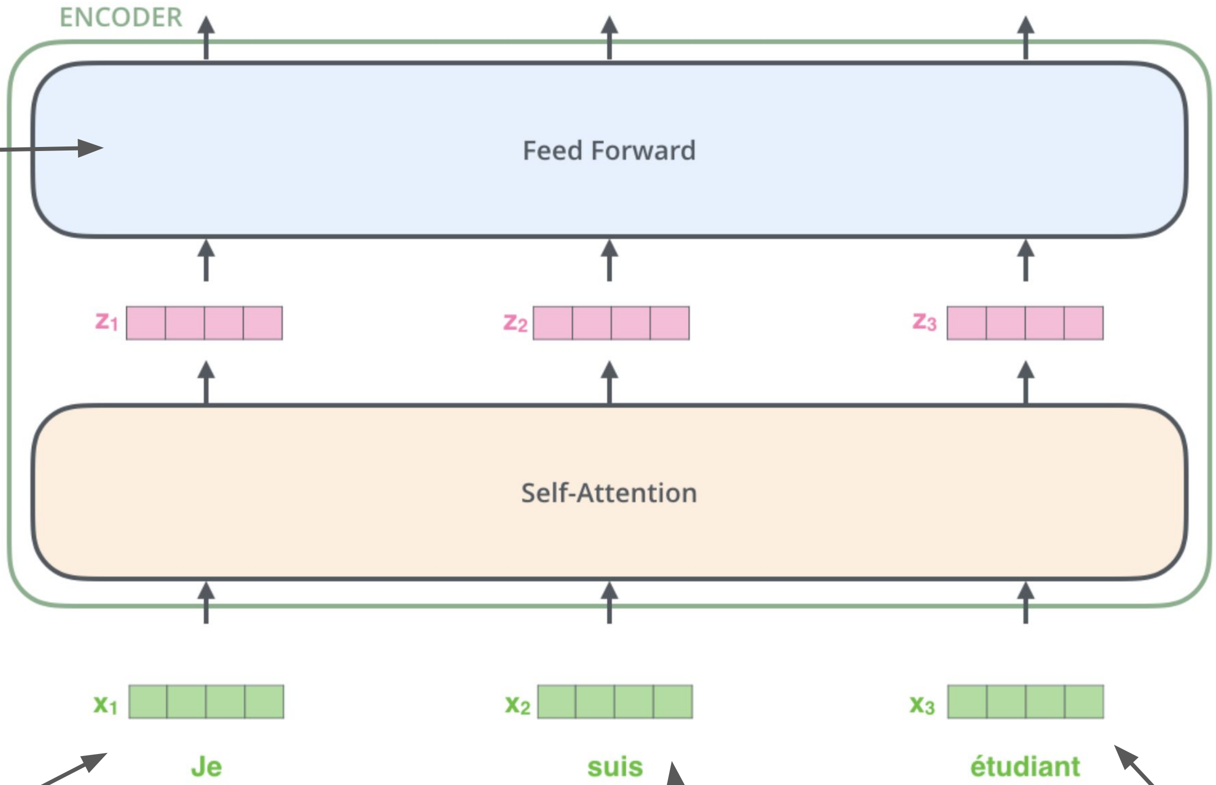


Image source: http://jalammar.github.io/illustrated-bert/

Transformer Block in BERT

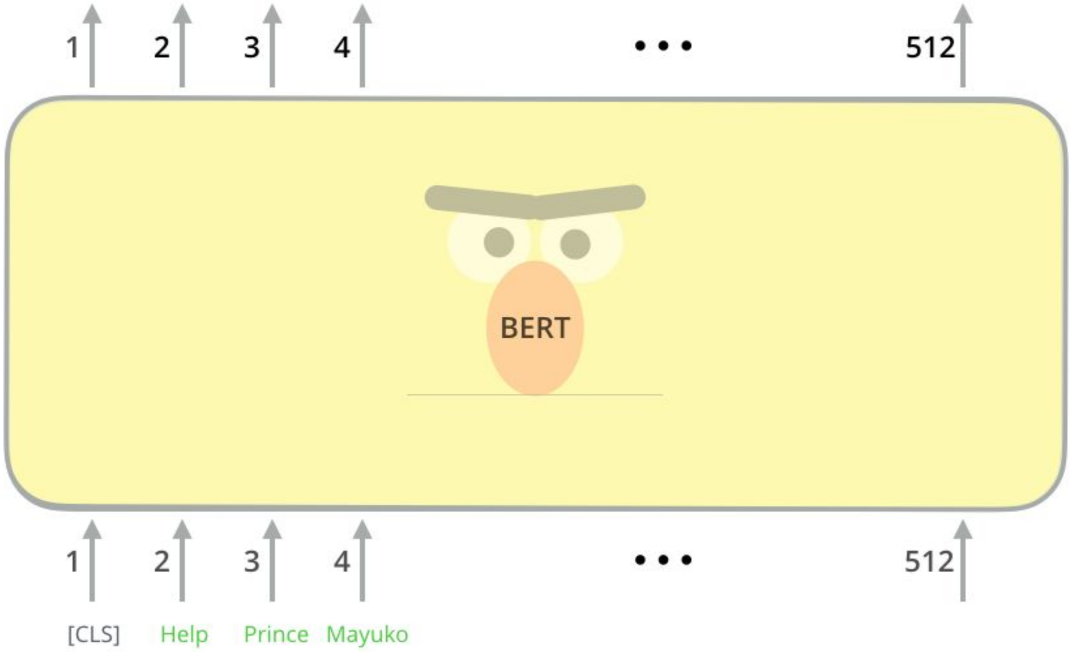
Can be parallelized



the word in each position flows through its own path in the encoder

Image source: https://jalammar.github.io/illustrated-transformer/

Model inputs

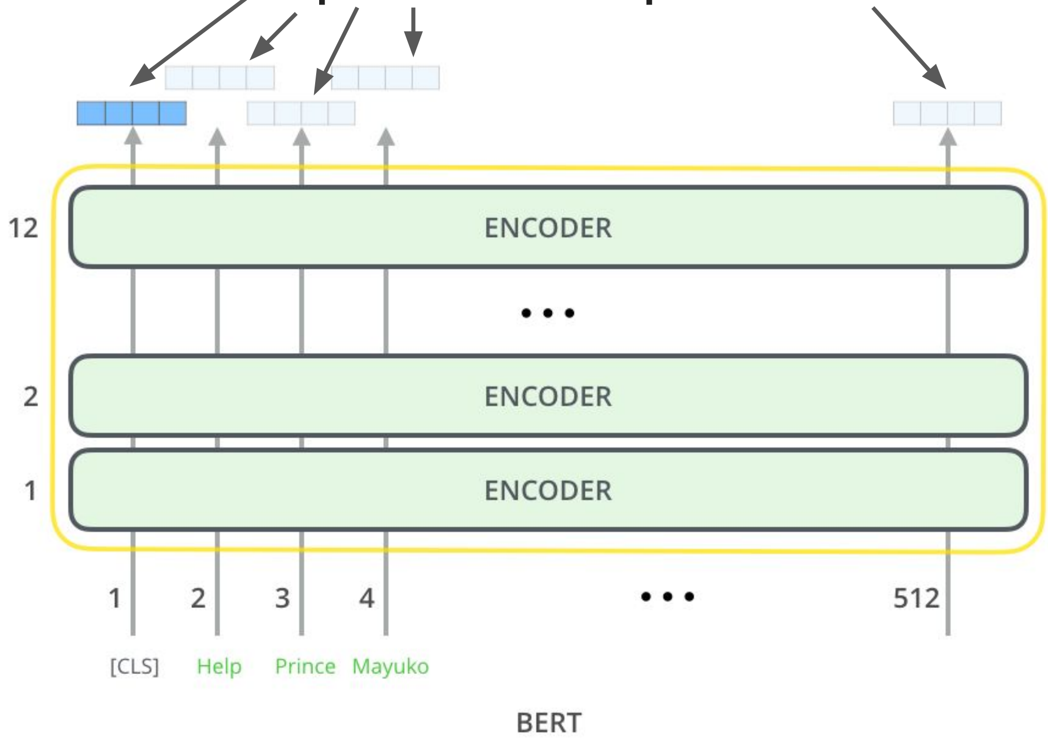


Identical to the Transformer up until this point

Why is BERT so special?

Model outputs

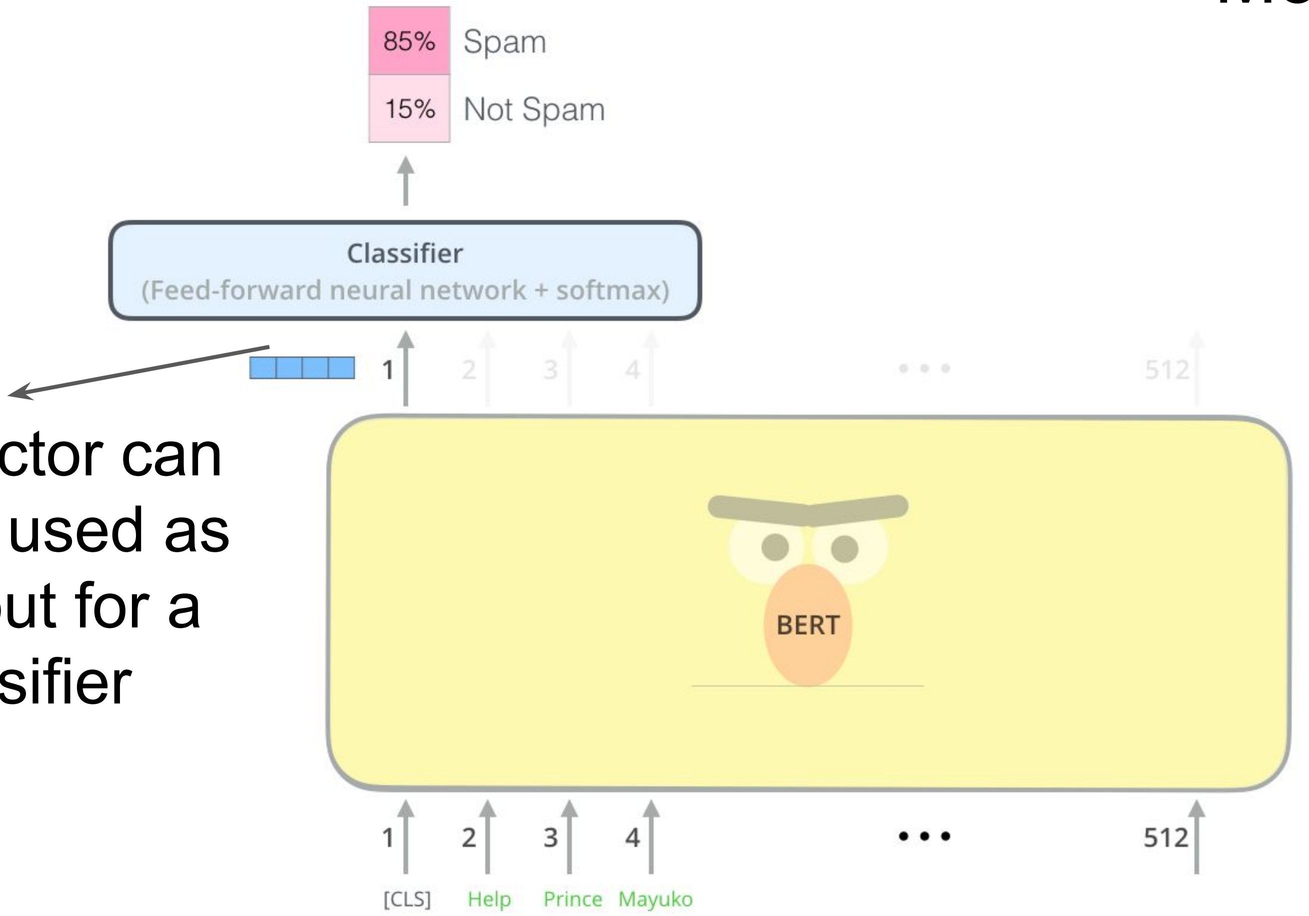
Each position outputs a vector



For sentence classification we focus on the first position (that we passed [CLS] token to)

Image source: http://jalammar.github.io/illustrated-bert/

Model inputs



This vector can now be used as the input for a classifier

Image source: http://jalammar.github.io/illustrated-bert/

Similar to CNN concept!

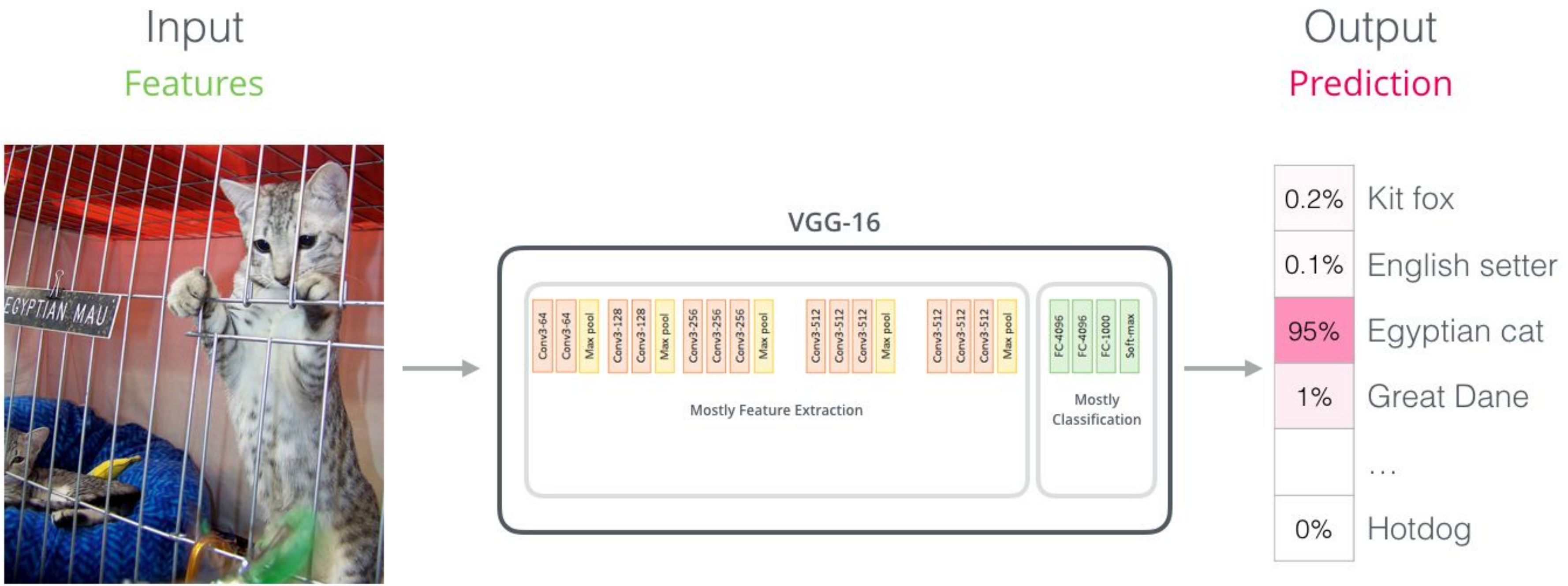


Image source: http://jalammar.github.io/illustrated-bert/

BERT: pre-training

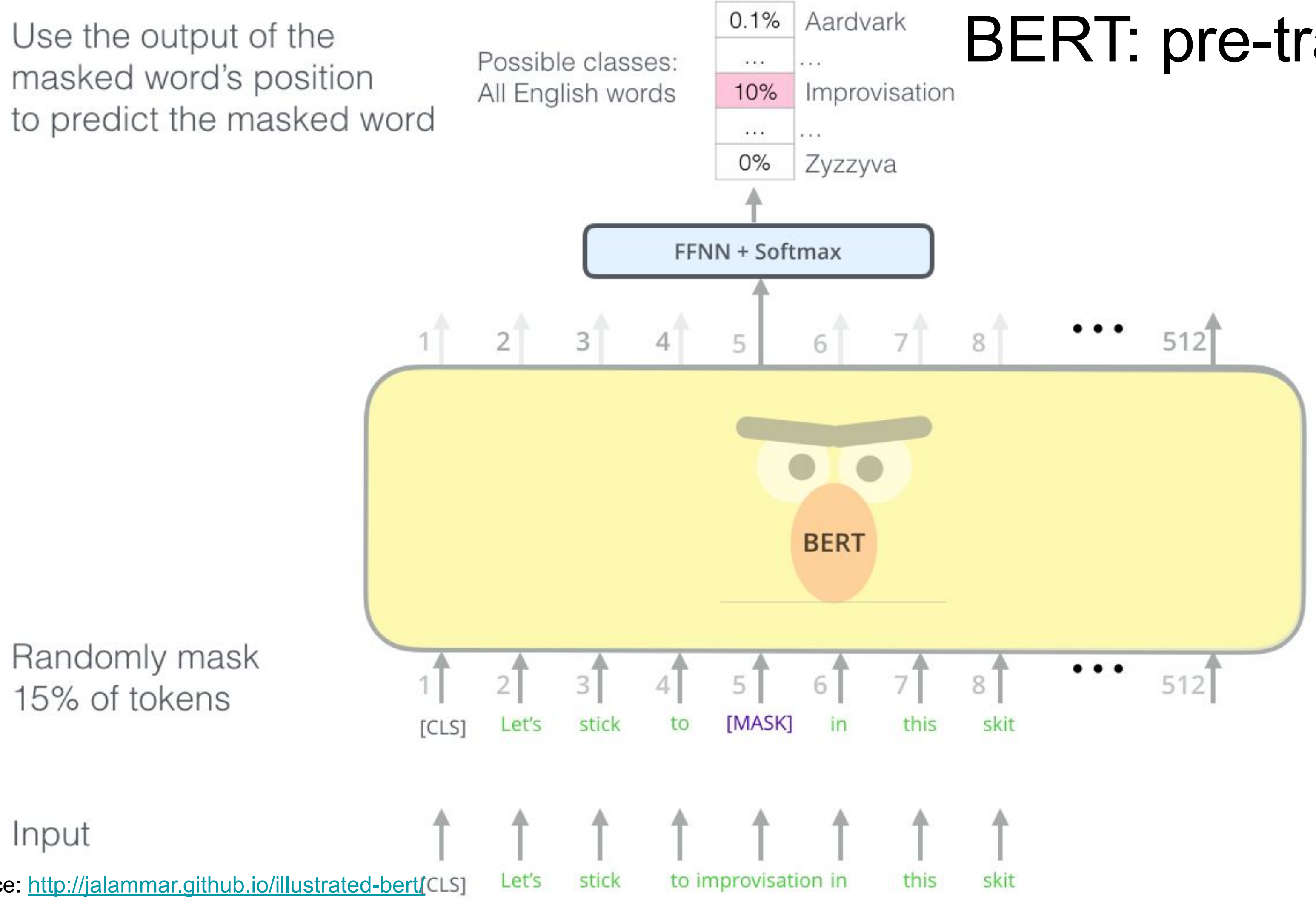


Image source: http://jalammar.github.io/illustrated-bert/

BERT: pre-training

- "Masked Language Model" approach

- To make BERT better at handling relationships between multiple sentences, the pre-training process includes an additional task:

  "Given two sentences (A and B), is B likely to be the sentence that follows A, or not?"

BERT: pre-training

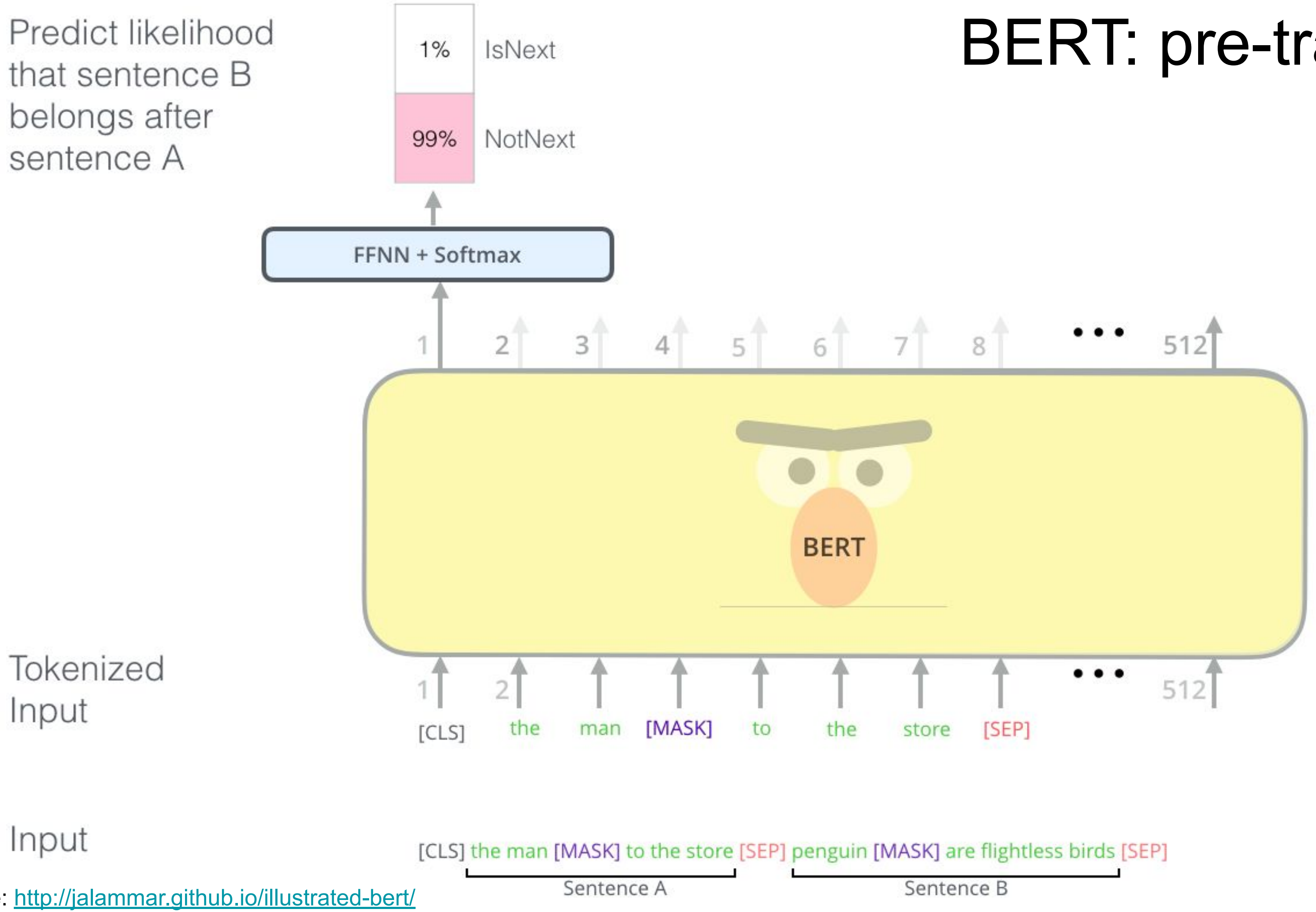


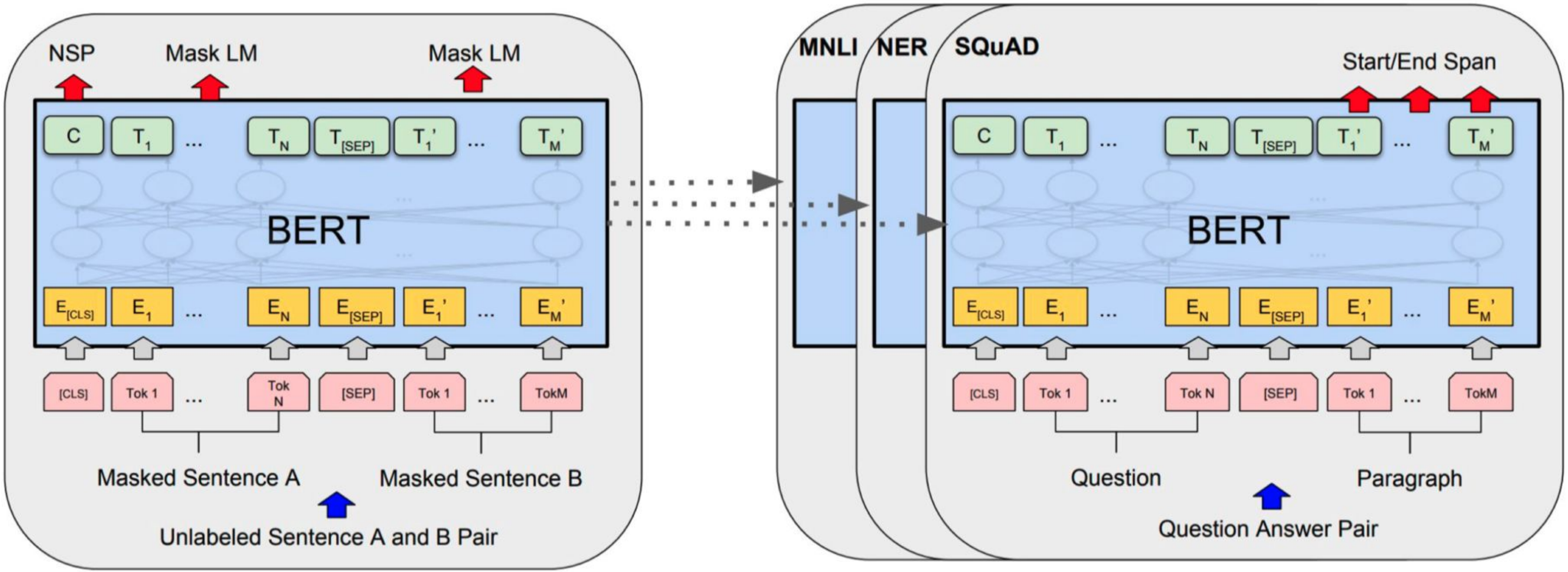
Image source: http://jalammar.github.io/illustrated-bert/

BERT: input data format

For each tokenized input sentence, we need to create:

- **input ids**: a sequence of integers identifying each input token to its index number in the BERT tokenizer vocabulary
- **segment mask**: a sequence of 1s and 0s used to identify whether the input is one sentence or two sentences long. For one sentence inputs, this is simply a sequence of 0s. For two sentence inputs, there is a 0 for each token of the first sentence, followed by a 1 for each token of the second sentence
- **attention mask**: a sequence of 1s and 0s, with 1s for all input tokens and 0s for all padding tokens

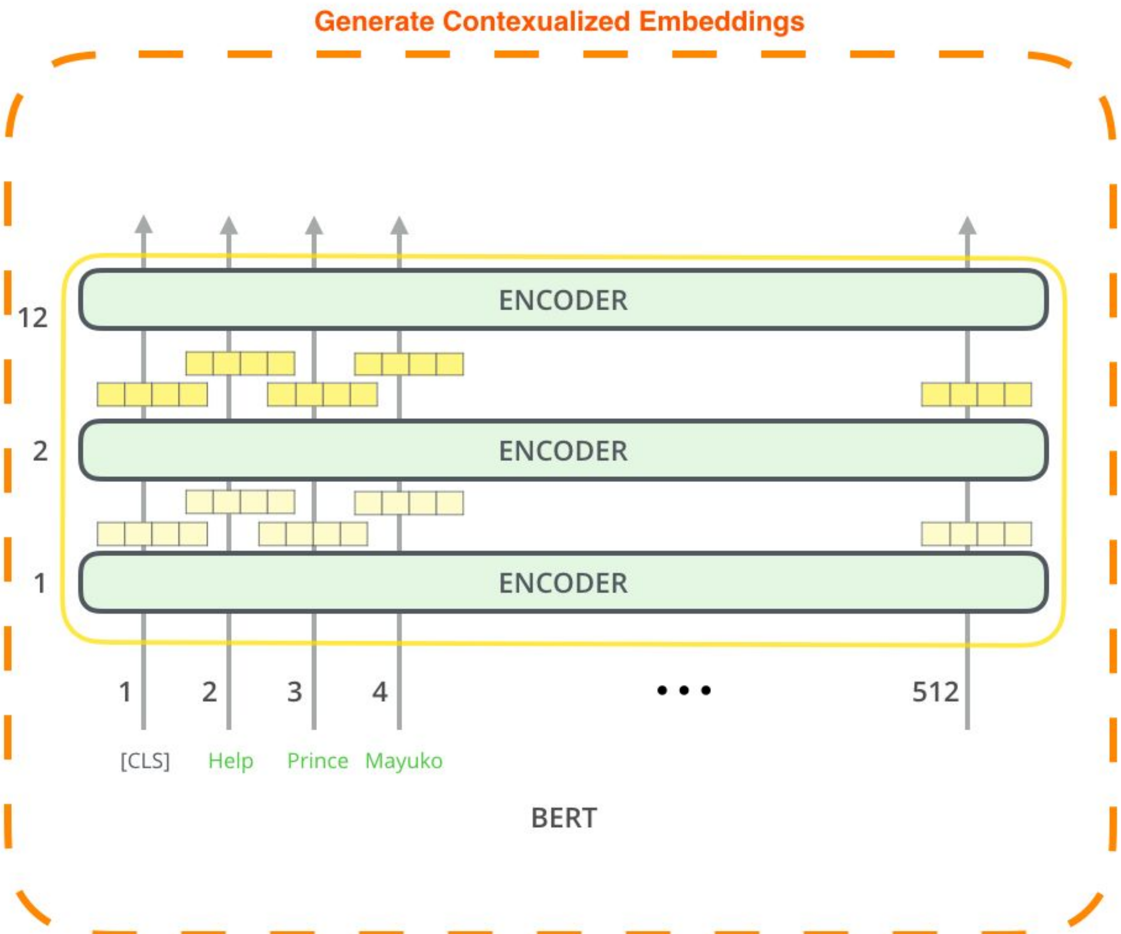
BERT: fine-tuning for different tasks



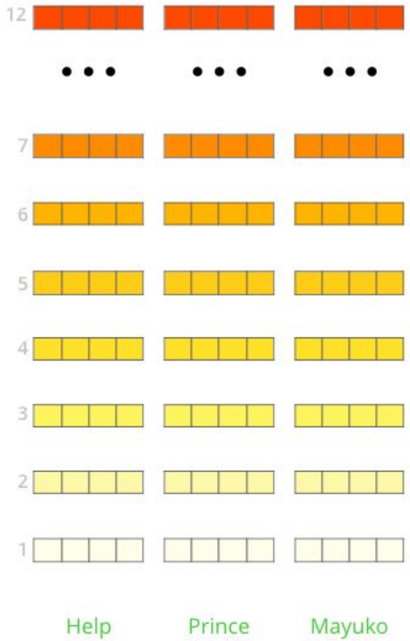
Pre-training Fine-Tuning

Image source: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture13-contextual-representations.pdf

BERT for feature extraction



The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

Image source: http://jalammar.github.io/illustrated-bert/

BERT for feature extraction

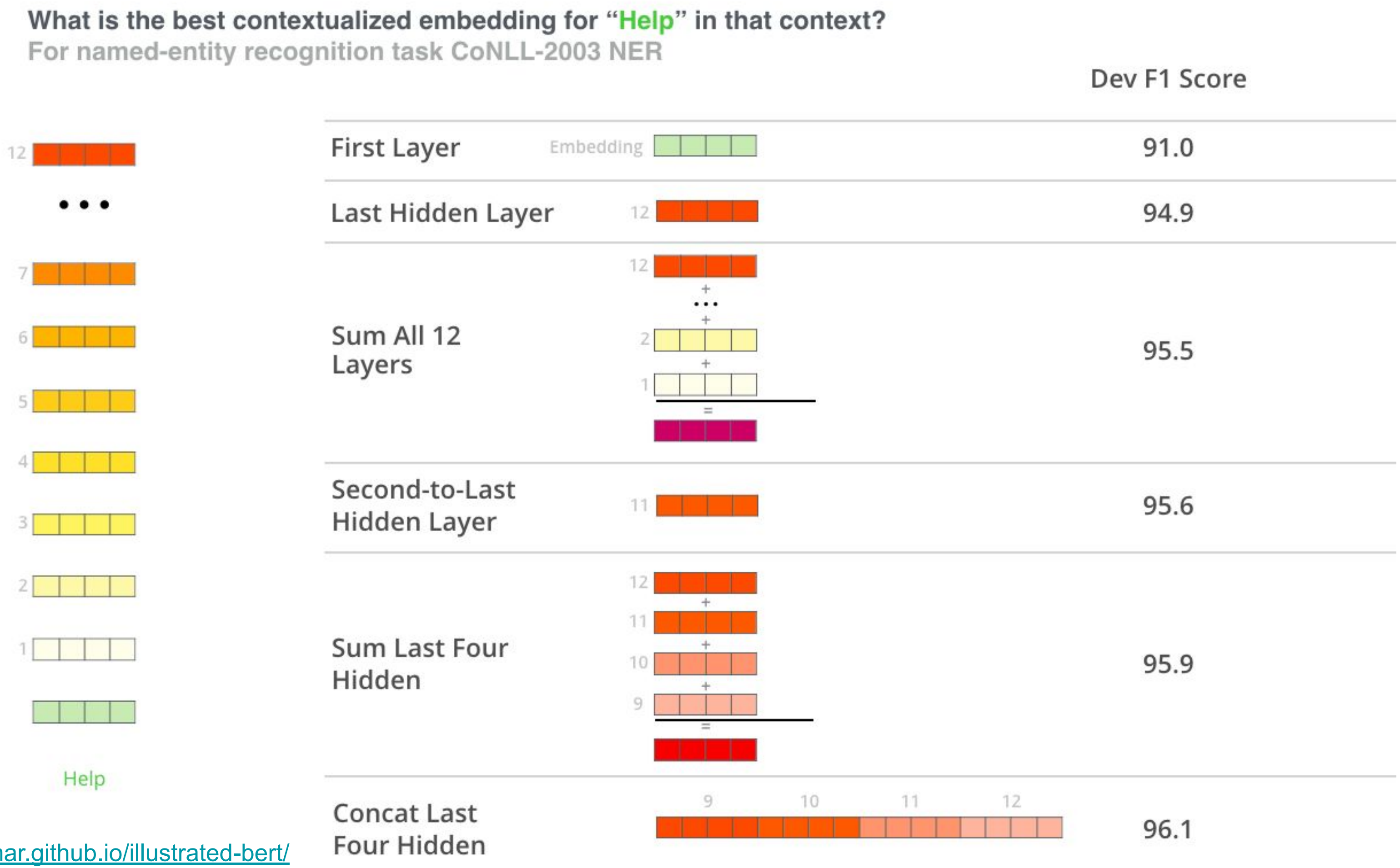


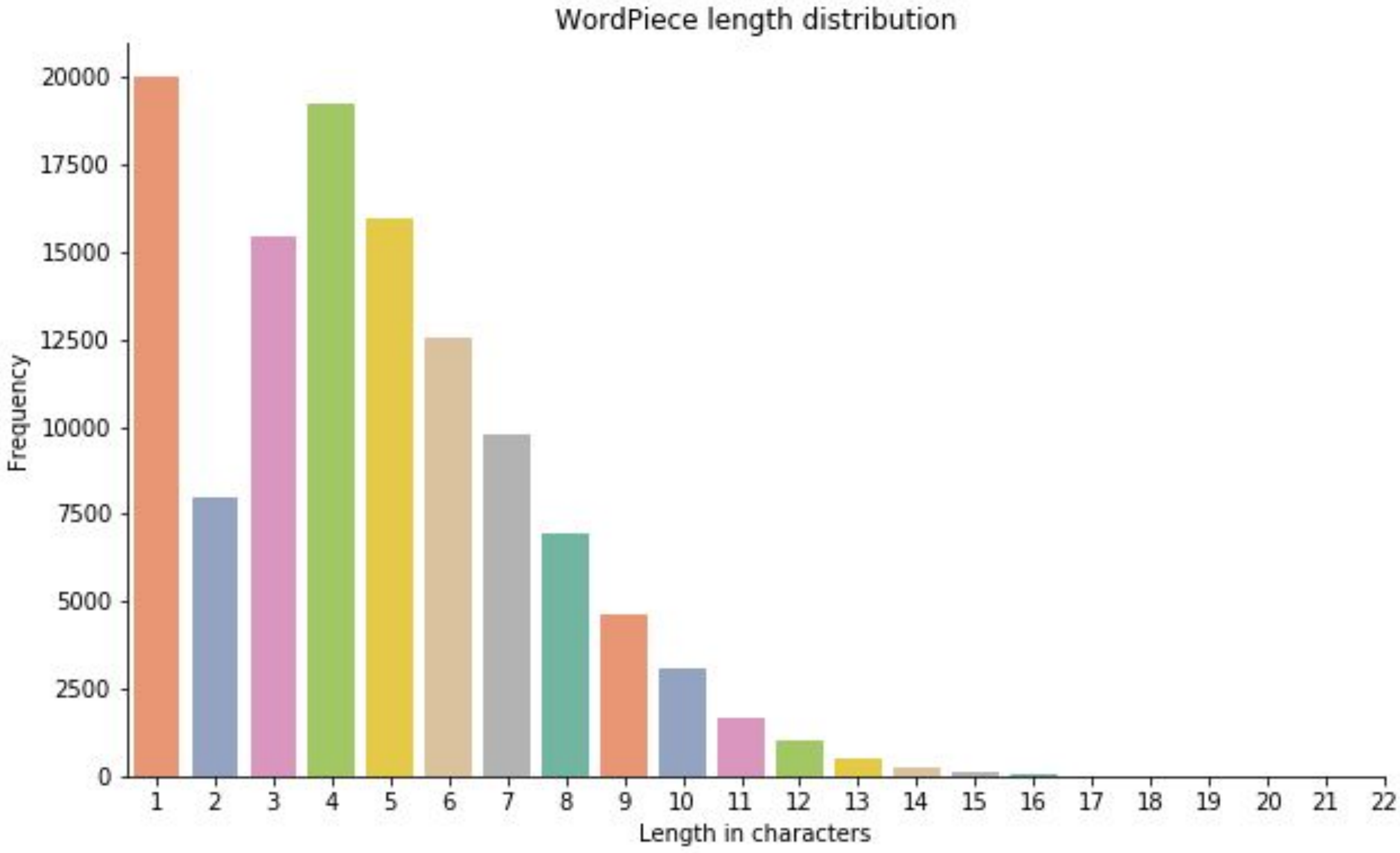
Image source: http://jalammar.github.io/illustrated-bert/

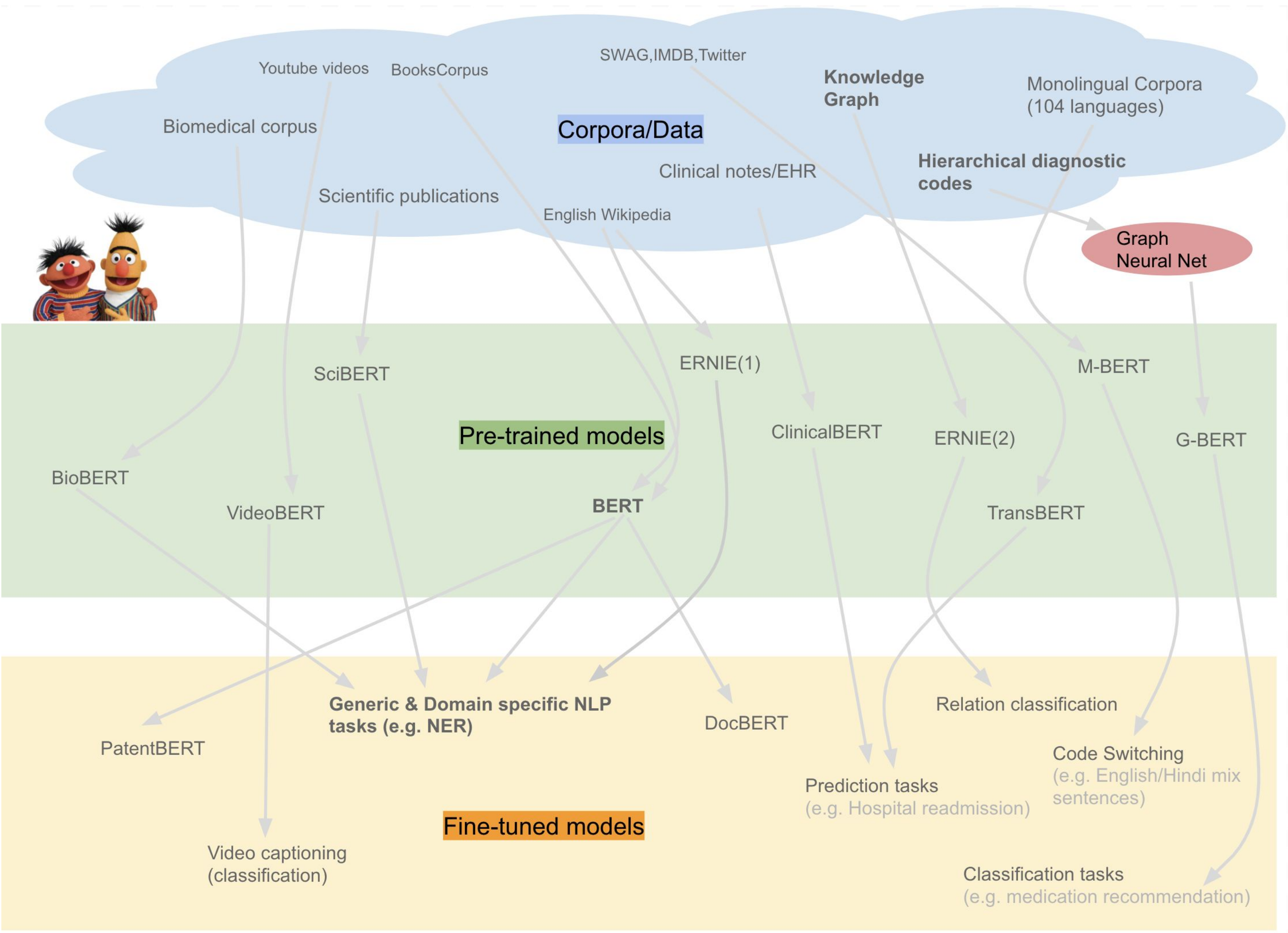
BERT: tokenization

<u>Example:</u> Unaffable -> un, ##aff, ##able

- Single model for 104 languages with a large shared vocabulary (119,547 WordPiece model)
- Non-word-initial units are prefixed with ##
- The first 106 symbols: constants like PAD and UNK
- 36.5% of the vocabulary are non-initial word pieces
- The alphabet consists of 9,997 unique characters that are defined as word-initial (C) and continuation symbols (##C), which together make up 19,994 word pieces
- The rest are multi character word pieces of various length.

BERT: tokenization




Corpora/Data
Youtube videos
BooksCorpus
SWAG,IMDB,Twitter
Knowledge Graph
Monolingual Corpora (104 languages)
Biomedical corpus
Clinical notes/EHR
Hierarchical diagnostic codes
Scientific publications
English Wikipedia
Graph Neural Net
Pre-trained models
SciBERT
ERNIE(1)
M-BERT
ClinicalBERT
ERNIE(2)
G-BERT
BioBERT
VideoBERT
BERT
TransBERT
Fine-tuned models
Generic & Domain specific NLP tasks (e.g. NER)
DocBERT
Relation classification
PatentBERT
Code Switching
(e.g. English/Hindi mix sentences)
Prediction tasks
(e.g. Hospital readmission)
Video captioning
(classification)
Classification tasks
(e.g. medication recommendation)

BERT: overview

- BERT repo
- Try out BERT on TPU
- WordPieces Tokenizer
- PyTorch Implementation of BERT

Outro

- Attention mechanism allows to “attend all positions” in the original sequence (or any other input with internal structure)
- Attention mechanism requires more computational resources than original seq2seq models
- Change of the model architecture affects the training procedure, so be careful with intuitive explanations