Natural Language Processing

Lecture 6 : Contextualized Word Embeddings

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Lecture partially based on material originally developed by :
Elena Voita, University of Edinburgh

Review: transformer

©Marvel Productions

Review: transformer

The **transformer** is a neural network architecture modeling sequence to sequence transformation (seq2seq): it turns one sequence into another sequence.

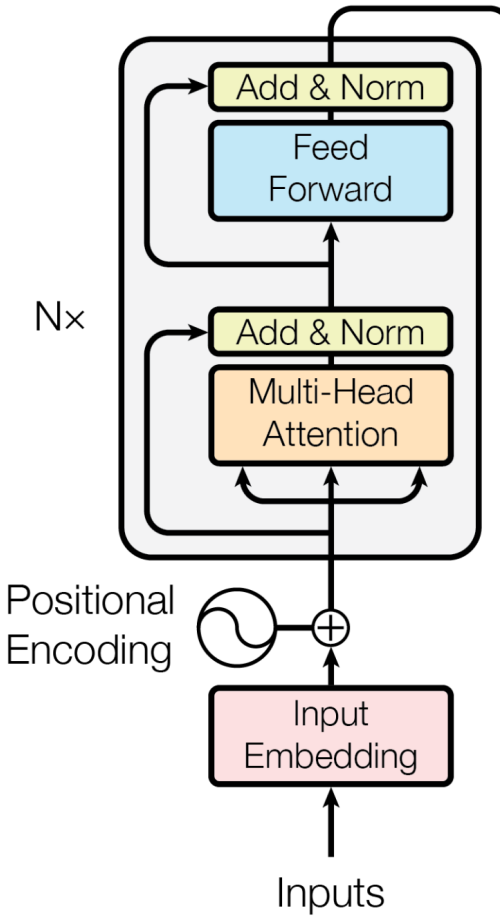
See Jurafsky & Martin §9 for an introduction to transformers.

The transformer model uses an **encoder-decoder** architecture.

The transformer computes representations of its input and output entirely based on **self-attention**, without relying on sequential computation.

High degree of parallelism: computation performed for each item is independent of all the other computations.

Review: transformer



Encoder

- tokenizer + embedding layer with positional encoding
- stack of N identical blocks with two layers each
- first layer is a multi-head self-attention mechanism
- second layer is a fully connected feed-forward network
- both layers employ residual connection + normalization

Review: transformer

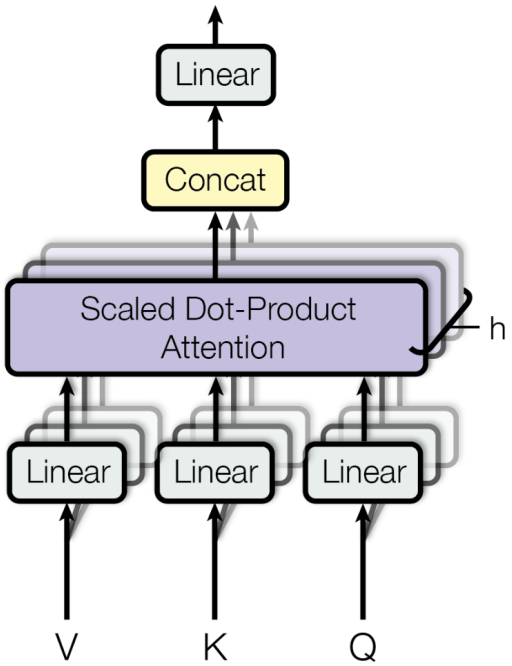
The encoder uses so-called **scaled dot-product attention**.

Attention is computed on the basis of three different roles for the input tokens, produced through linear transformation

- **query**: current focus in the comparison
- **key**: importance to the given query
- **value**: contribution to the given query

Query-key product needs to be scaled for gradient stability.

Review: transformer



Multi-head attention

- input tokens can relate to each other in many different ways simultaneously
- multi-head attention implements parallel layers of self-attention that reside at the same depth

Review: transformer

Residual connection allows information to skip a layer, improving learning.

Layer normalization improves performance by keeping the values of a hidden layer in a range that facilitates gradient-based training.

This uses mean, standard deviation, gain and offset.

Feed forward: a simple MLP applied to each token individually; uses GELU activation function.

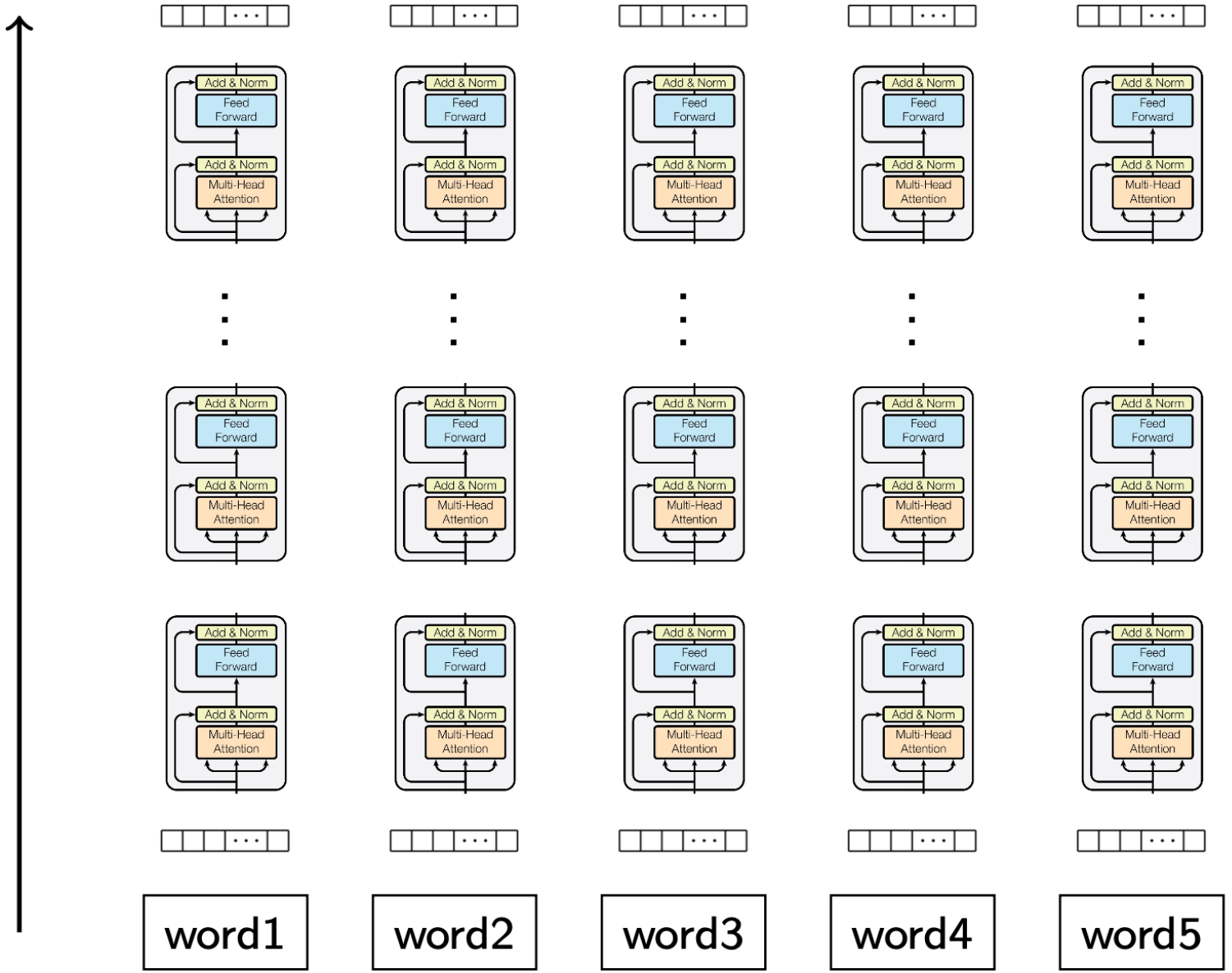
There's some weak evidence that this is where "world knowledge" is stored.

Review: transformer

Input length is **fixed**, since attention is quadratic.

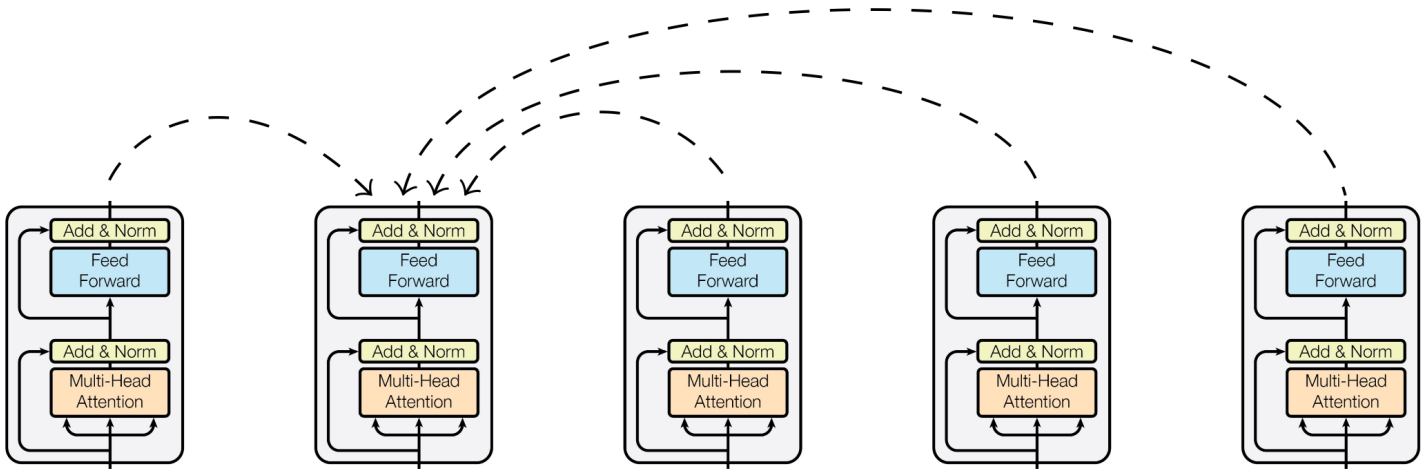
Encoder output is a high level, contextualized version of the input tokens, serving as a basis for decoder computation.

Review: transformer

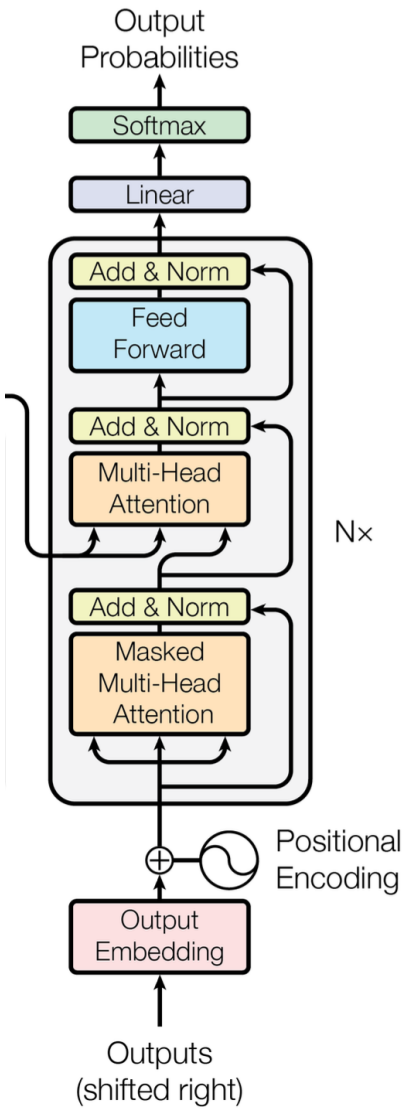


Review: transformer

Example of attention at second token:



Review: transformer



Decoder

- stack of N identical blocks with three layers each
- first layer as in encoder, but with masked attention (auto-regressive)
- second layer uses cross-attention (key and value of encoder stack)
- third layer as in encoder
- all layers employ residual connection + normalization

Review: transformer

At training time, decoder uses **masked** self-attention. In this way, each token only sees the already generated ones and computation can be carried out in parallel.

Also called autoregressive decoding.

At inference time we need to generate one token at a time. This is why autoregressive decoding is **much slower** than encoding.

Review: transformer

Moving layer normalization before multi-headed attention and feedforward layers stabilizes training; this is called prenorm.

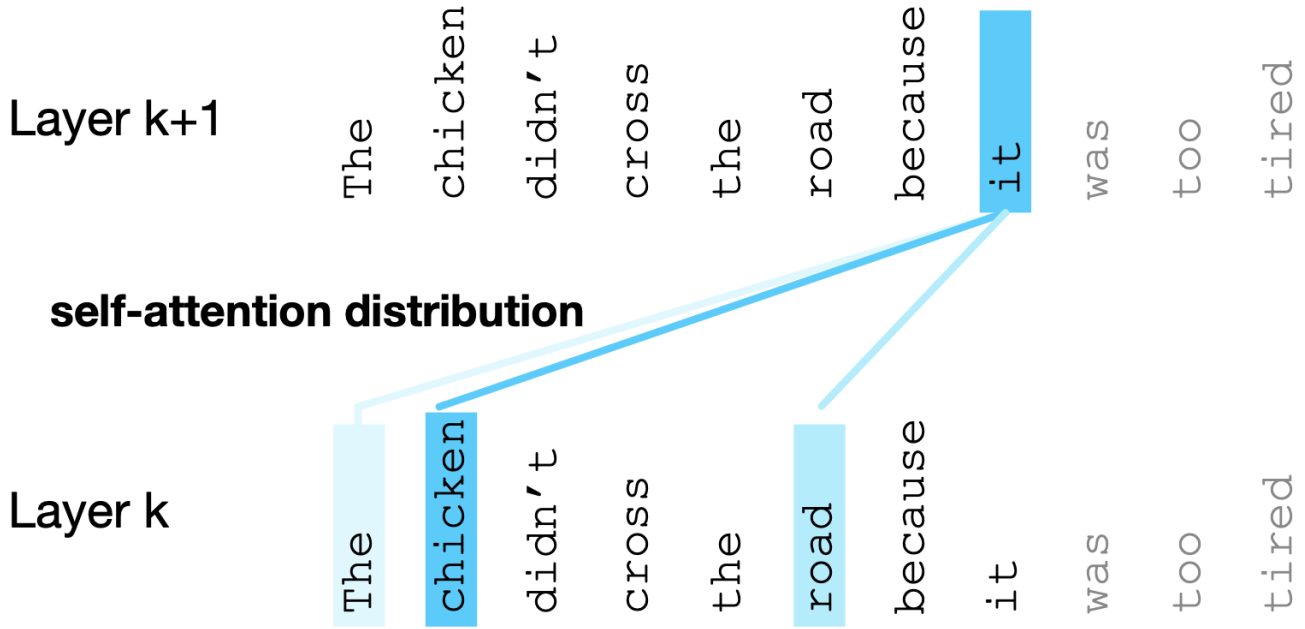
Fixed sinusoidal position embeddings sometimes replaced by unique, learned embeddings per position.

Some applications use encoder-only, or else decoder-only components.

Review: transformer

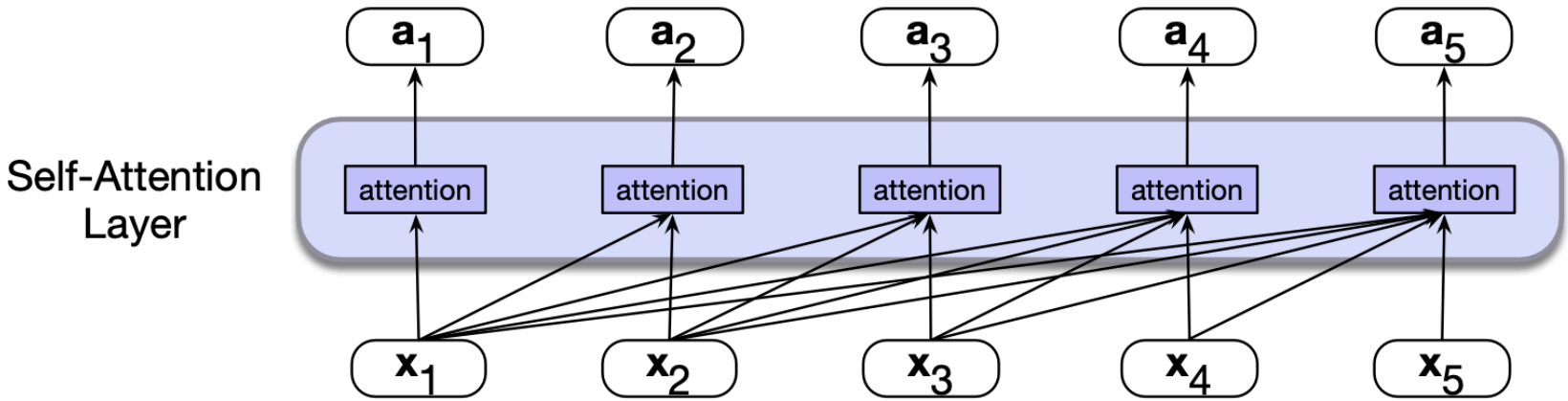
Masked (causal) self-attention

- combines the representations of tokens from layer $k-1$
- builds the representation of tokens at layer k



Review: transformer

Information flow in masked self-attention.



Review: transformer

Let $\mathbf{x}_i$ be the vector encoding of token at position i.

Single attention head computes **query**, **key**, and **value** vectors

$$\begin{aligned} \mathbf{q}_i &= \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i \\ \mathbf{k}_j &= \mathbf{W}^{\mathbf{K}}\mathbf{x}_j \\ \mathbf{v}_j &= \mathbf{W}^{\mathbf{V}}\mathbf{x}_j \end{aligned}$$

Review: transformer

Computation of attention logits and distribution

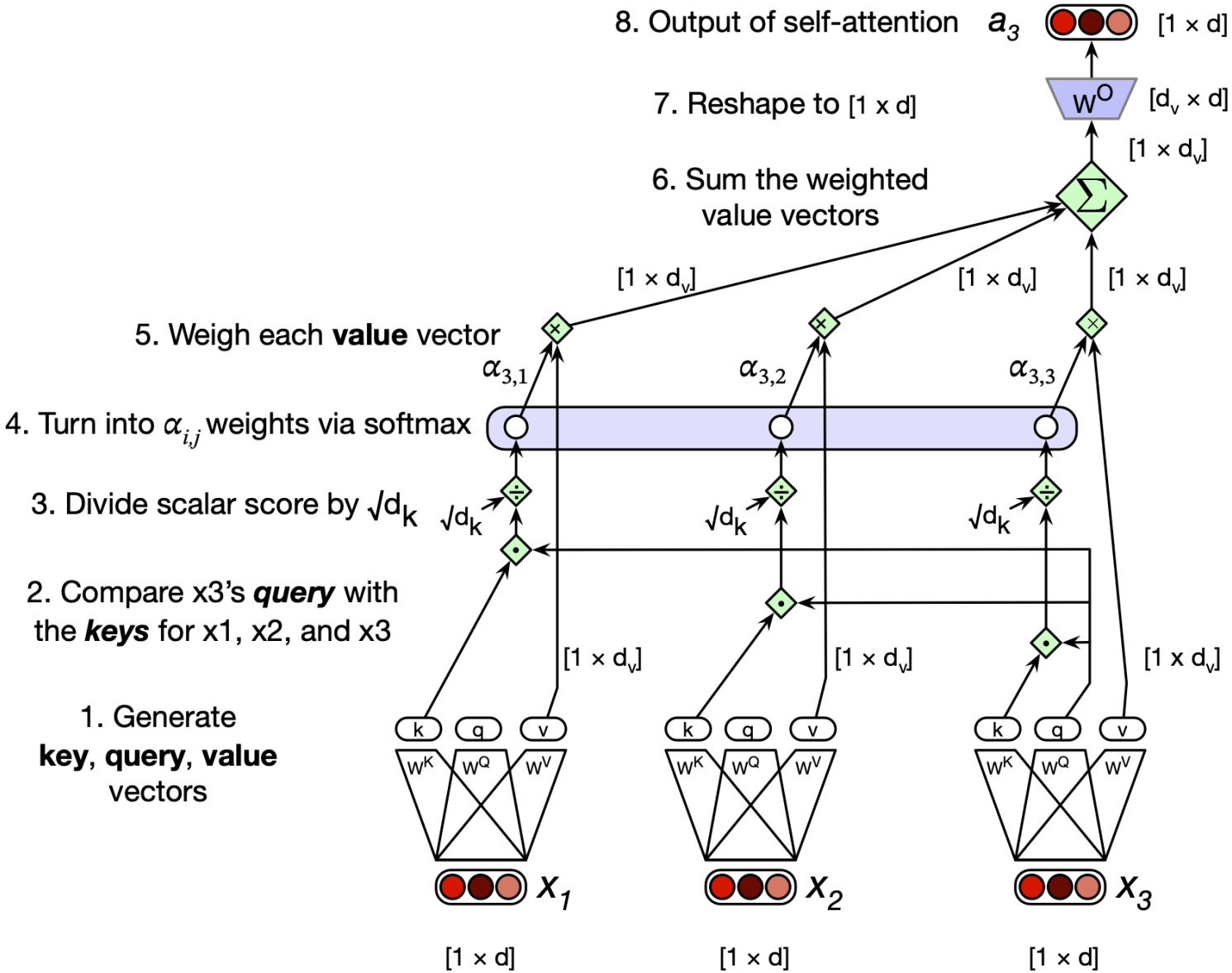
$$\begin{aligned} \text{score}(\mathbf{x}_i, \mathbf{x}_j) &= \frac{(\mathbf{q}_i)^\top \mathbf{k}_j}{\sqrt{d}} \\ \alpha_{i,j} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \qquad j \leqslant i \end{aligned}$$

Computation of output value at position i

$$\begin{aligned} \mathbf{head}_i &= \alpha_{i,j}\mathbf{v}_j \\ \mathbf{a}_i &= \mathbf{W}^{\mathbf{O}}\mathbf{head}_i \end{aligned}$$

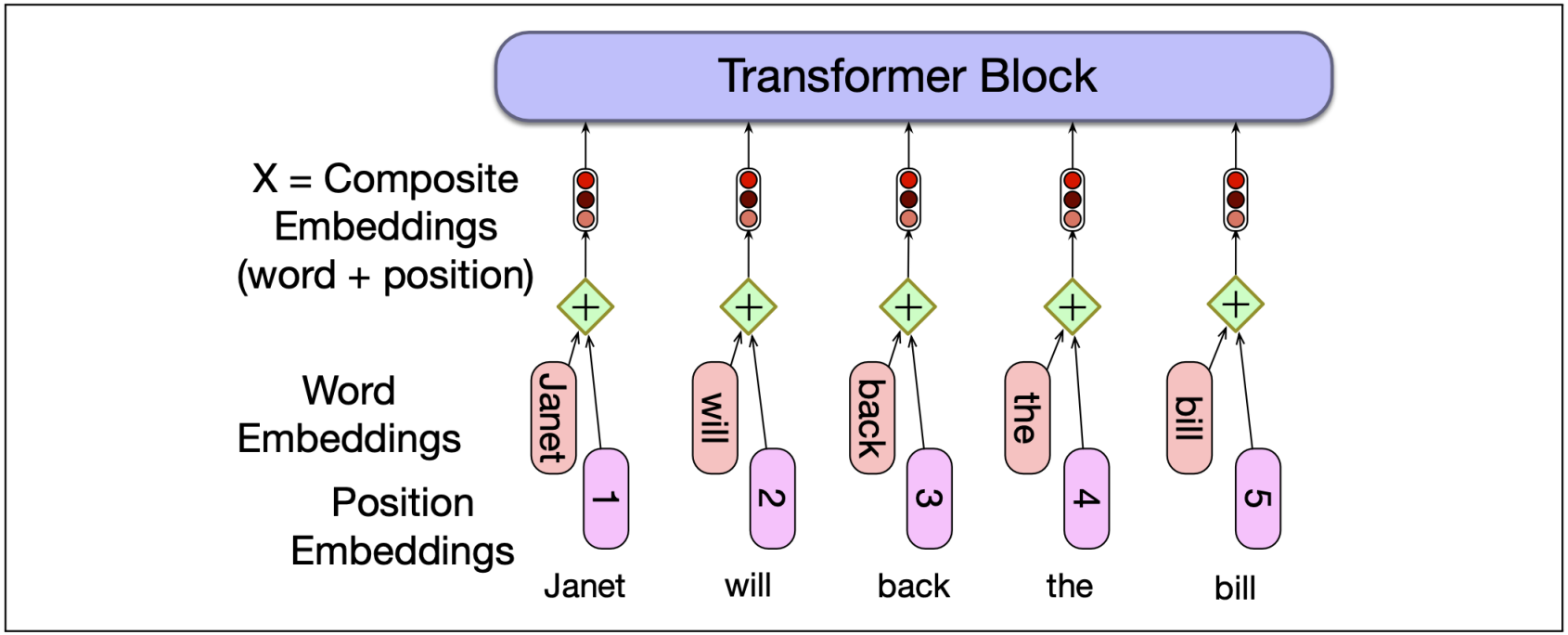
Review: transformer

Calculation of $\mathbf{a}_3$ using masked self-attention



Review: transformer

In input, each token embedding is combined with a **positional embedding**.

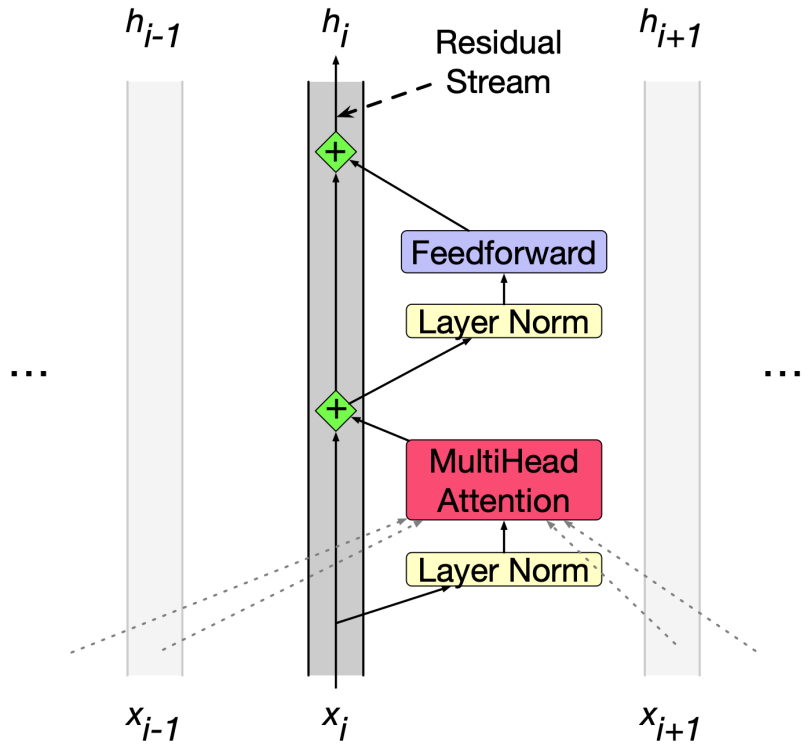


Review: transformer

Parallel computation of attention using a single matrix.

$$
\begin{aligned}
\mathbf{Q} &= \mathbf{W}^{\mathbf{Q}}\mathbf{X} \\
\mathbf{K} &= \mathbf{W}^{\mathbf{K}}\mathbf{X} \\
\mathbf{V} &= \mathbf{W}^{\mathbf{V}}\mathbf{X} \\
\mathbf{head} &= \text{softmax}\left(\text{mask}\left(\frac{(\mathbf{Q})\mathbf{K}^{\top}}{\sqrt{d}}\right)\right)\mathbf{V} \\
\mathbf{A} &= \mathbf{W}^{\mathbf{O}}\mathbf{head}
\end{aligned}
$$

Review: transformer



Residual stream view of the transformer (using prenorm)

- stream starts with the original input vector
- components read their input from the stream
- components add their output back into the stream

Contextualized word embeddings

Magda Ehlers on Pexels

Contextualized word embeddings

Word embeddings such as word2vec or GloVe learn a single vector for each **type** (unique word) in the vocabulary V. These are also called **static embeddings**.

However, in natural language words have rich linguistic relationships with other words that may be far away.

Example :
I walked along the **pond**, and noticed one of the trees along the **bank**

Example :
The **chicken** didn't cross the road because **it** was too tired
The chicken didn't cross the **road** because **it** was too wide

Contextualized word embeddings

By contrast, **contextualized word embeddings** represent each **token** (word occurrence) by a different vector, according to the context the token appears in.

Also called pre-trained language models, dynamic embeddings, large language models, or foundation model.

Incorporating context into word embeddings has proven to be a **watershed idea** in NLP, opening the way to transfer learning.

We discuss transfer learning and fine-tuning later on in this lecture.

Contextualized word embeddings

How can we learn to represent words along with the context they occur in?

We train a neural network combining ideas from word embeddings and language models (in historical order):

- predict a word from left and right context **independently**

  Bidirectional LSTM (ELMo)

- predict a word from left and right context **jointly**

  Transformer encoder (BERT)

- predict a word from left context only

  Transformer decoder (GPT-*n*)

©PBS/HBO Sesame Street

ELMo (embeddings from language model) looks at the entire sentence before assigning each word its embedding.

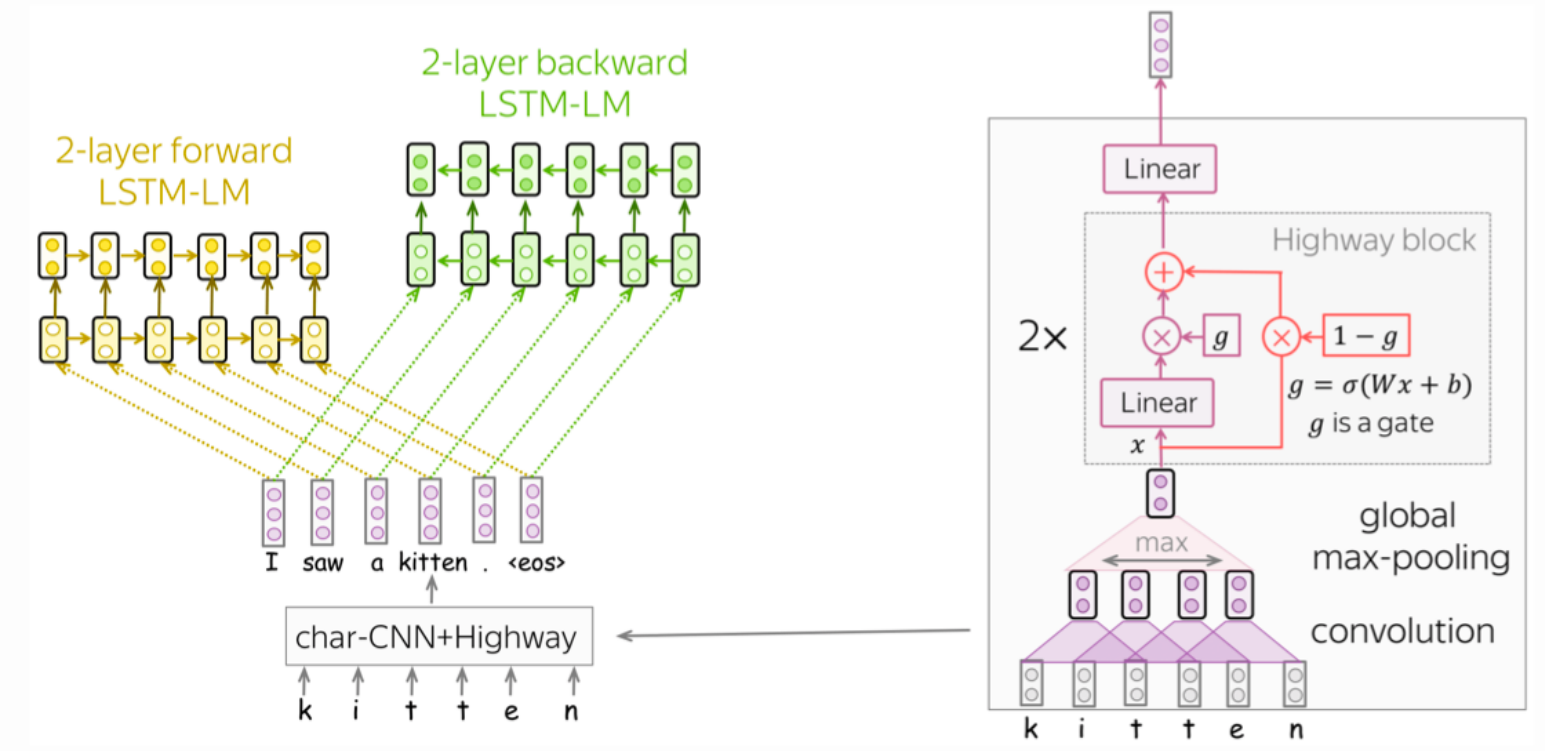
Created by researchers at the Allen Institute for Artificial Intelligence (AI2) and University of Washington, 2018.

Tokens are processed by a character-level convolutional neural network (CNN) producing word-level embeddings.

This captures word morphology and resolves OOV words.

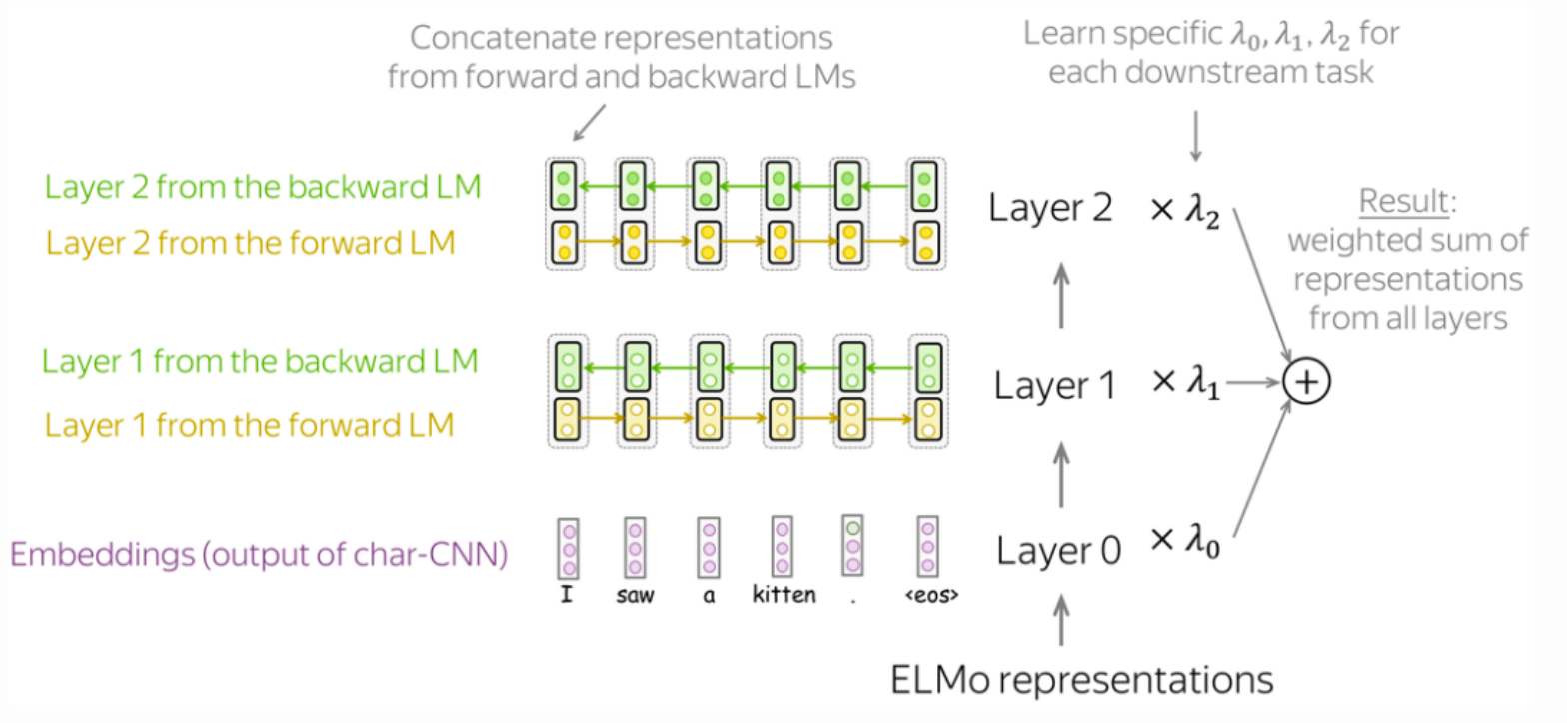
These embeddings are processed by a left-to-right and a right-to-left, 2-layers LSTM.

This is also called a bi-directional LSTM.



https://lena-voita.github.io/nlp_course/transfer_learning.html

For each word, the output embeddings at each layer (including the CNN layer) are combined, producing contextual embeddings.



https://lena-voita.github.io/nlp_course/transfer_learning.html

When **training**, add output layer with

- linear transformation to $|V|$ dimension
- softmax

and use cross-entropy as in neural language models.

Also recommended parameter dropout and L2 regularization.

When **encoding** the input

- ignore output layer
- retain the word embedding represented by the last hidden layer.

BERT

©PBS/HBO Sesame Street

BERT

BERT (bidirectional encoder representations from transformers) produces word representations by **jointly** conditioning on both left and right context.

Created by researchers at Google AI Language, 2018.

The model is based on the encoder component of the transformer neural network. Tokenizer uses WordPiece algorithm.

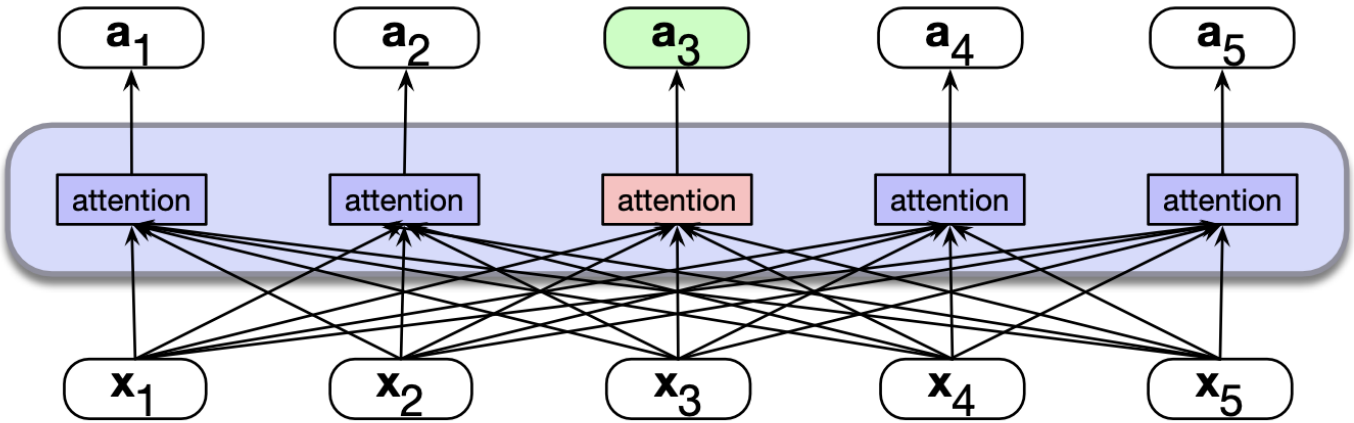
Two versions

- BERT Base: 12 layers, 12 attention heads, 768 embedding size, 110 million parameters
- BERT Large: 24 layers, 16 attention heads, 1024 embedding size, 340 million parameters

BERT

The bidirectional encoding in BERT is inherited from the self-attention mechanism of the transformer encoder.

Recall that the encoder does not use masking for the computation of attention.



Masked language modeling

The model learns to perform a fill-in-the-blank task, technically called the **cloze task**.

Example : Please turn _____ homework in.

The approach is called **masked language modeling** (MLM).
A random sample of 15% of the input tokens are

- replaced with the unique vocabulary token `[MASK]` (80%)
- replaced with another token randomly sampled with unigram probabilities (10%)
- left unchanged (10%)

  Used for transfer learning, presented later, in which token `[MASK]` does not appear.

Masked language modeling

Why the three possible manipulations?

Adding the `[MASK]` token creates a mismatch between training and inference.

If we just replaced tokens with the `[MASK]`, the model might only predict tokens when it sees a `[MASK]`.

However we want the model to try to always predict the input token.

Masked language modeling

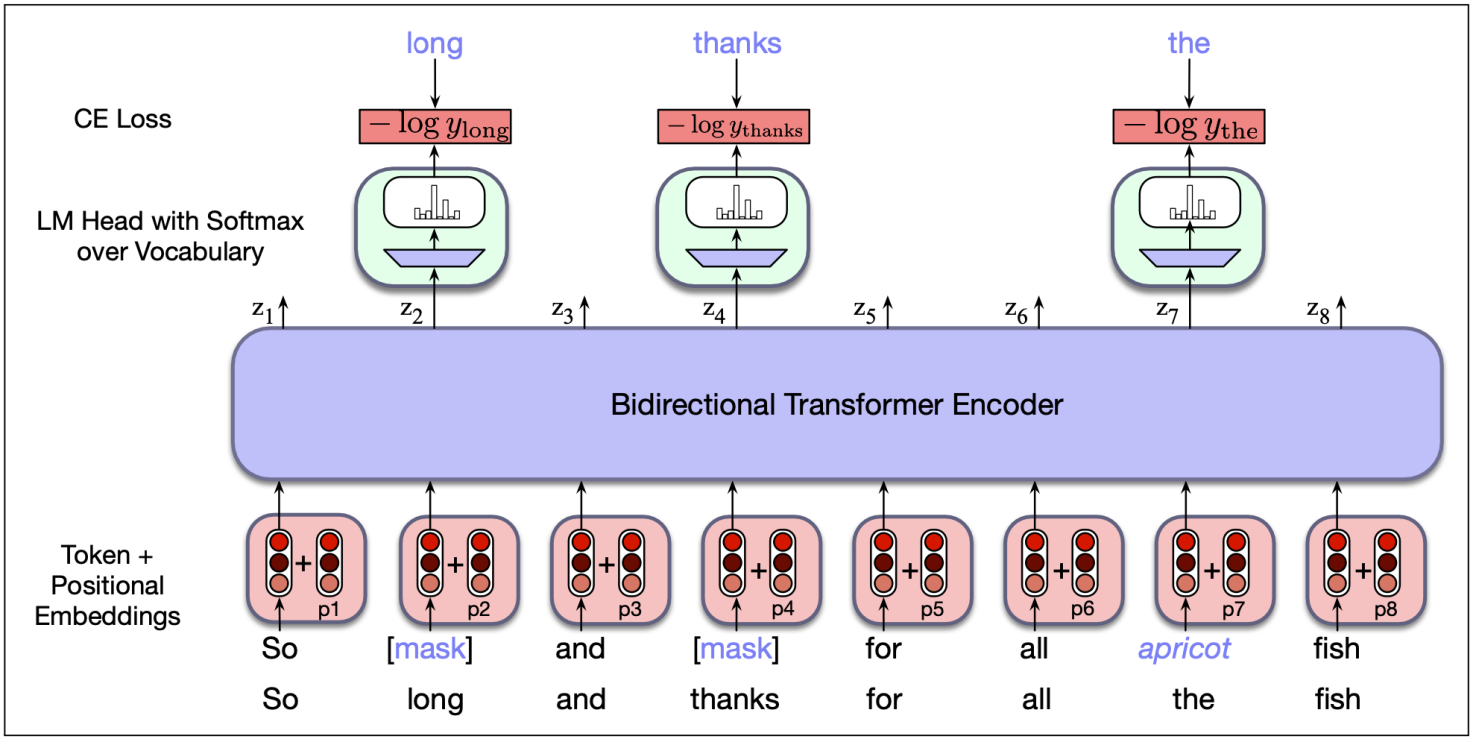
In masked language modeling, we train using a **language modeling head**, which

- takes the output vector $\mathbf{h}_i^L$ for each of the masked tokens
- multiplies it by the unembedding layer $\mathbf{E}^\top$ to produce the logits $\mathbf{u}$; in other words, language modeling head is tied to the embedding matrix $\mathbf{E}$.
- uses softmax to turn the logits into probabilities $\mathbf{y}$ over the vocabulary, to **predict** the masked word

$$\begin{aligned} \mathbf{u}_i &= \mathbf{h}_i^L \mathbf{E}^\top \\ \mathbf{y}_i &= \text{softmax}(\mathbf{u}_i) \end{aligned}$$

Masked language modeling

Training is carried out as usual for language modeling, using cross-entropy as loss function.



Masked language modeling

Masked language modeling can be generalized to any method that corrupts the training input and asks to recover the original.

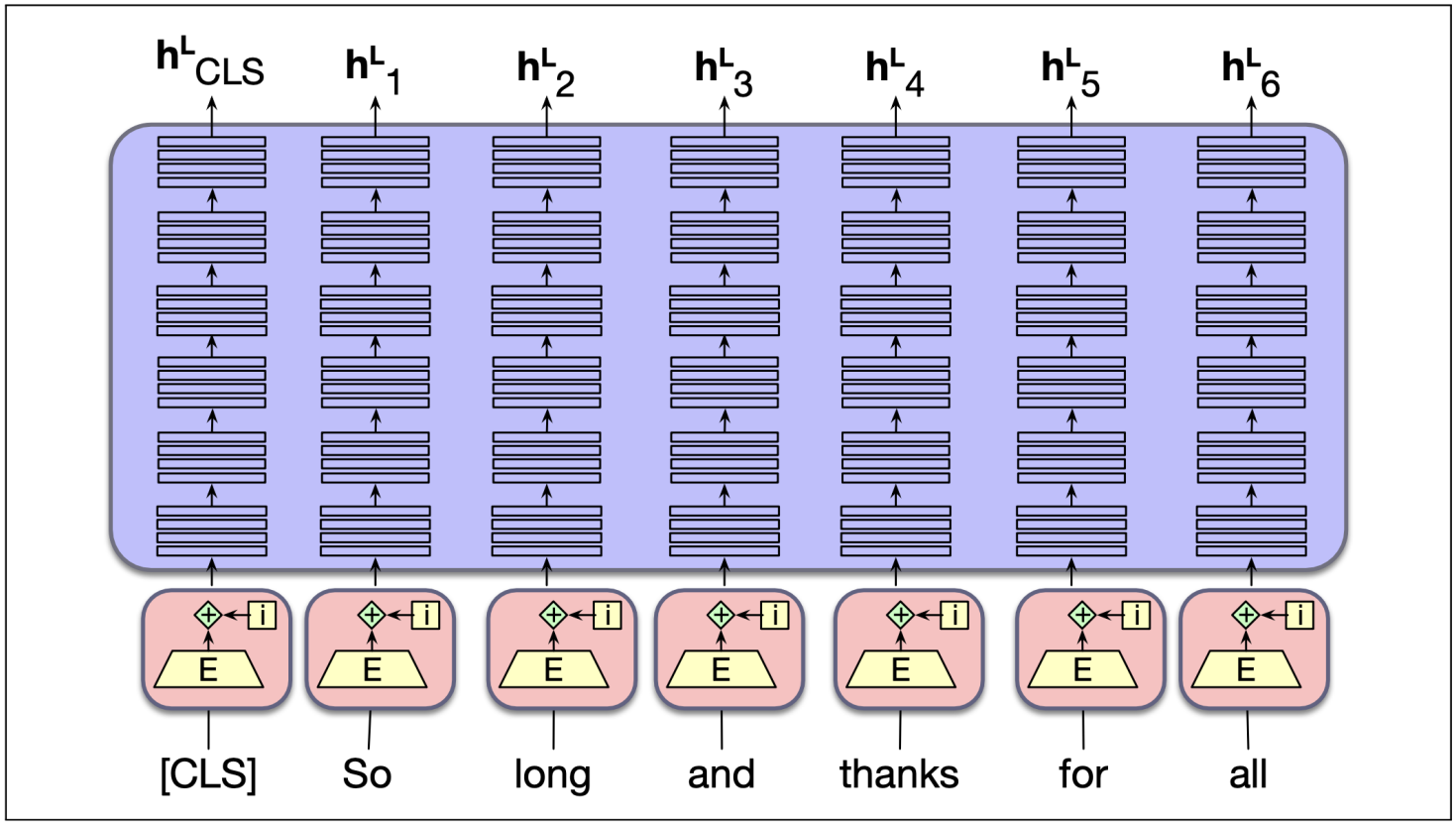
Examples include

- masking of several adjacent words
- reorderings of adjacent words
- insertion of extraneous words

The general name for this kind of training is **denoising**.

Masked language modeling

At inference time, the output of BERT is a **contextualized** embedding vector $\mathbf{h}_i^L$ for input token x_i.



Next sentence prediction

Many important downstream tasks involve relationship between pairs of sentences:

- **paraphrase detection**: detecting if two sentences have similar meanings, also called paraphrase identification
- **semantic textual similarity**: detecting the degree of semantic similarity between two sentences
- **sentence entailment**: detecting if the meanings of two sentences entail or contradict each other
- **answer sentence selection**: detecting if the answer to a question sentence is contained in the second sentence

This is not directly captured by language modeling.

Next sentence prediction

In order to train a model that understands sentence relationships, BERT introduces a second learning objective called **next sentence prediction** (NSP).

NSP is a **binary decision** task that can be trivially generated from any monolingual corpus.

NSP produces **sentence embeddings** that can be used for tasks involving relationship between pairs of sentences.

Next sentence prediction

In NSP, the model is presented with pairs of sentences with

- token `[CLS]` prepended to the first sentence
- token `[SEP]` placed between the two sentences and after the rightmost token of the second sentence
- **segment embeddings** marking the first and second sentences are added to each word and positional embeddings

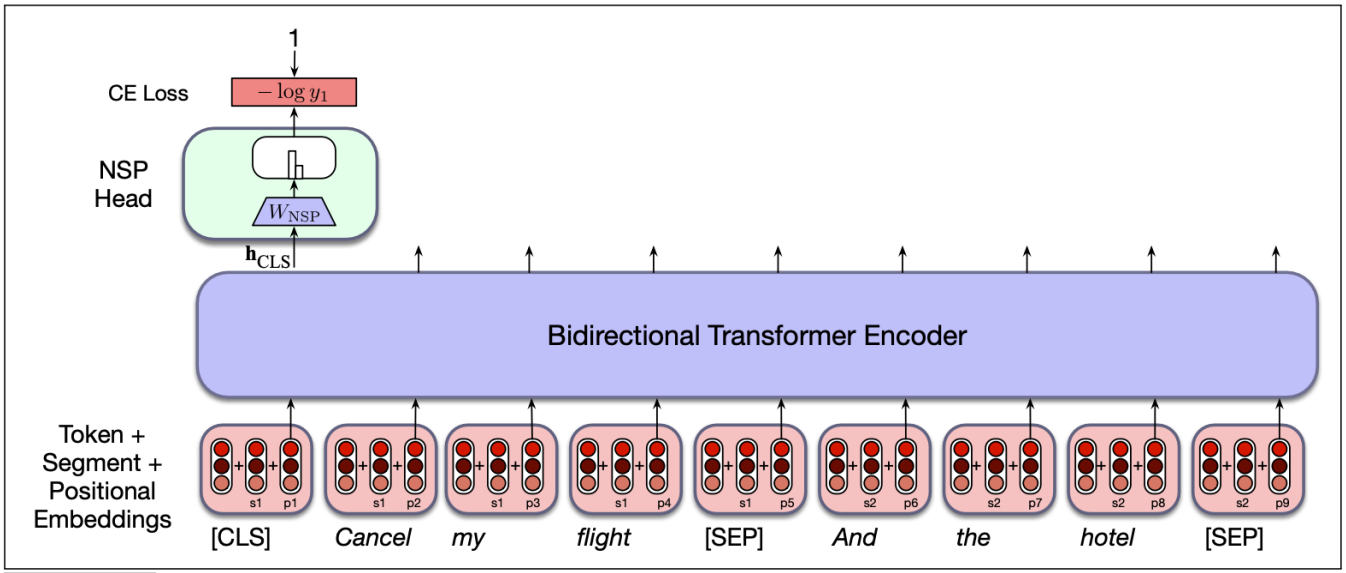
Each pair consists of

- an actual pair of adjacent sentences from the training corpus (50%)
- a pair of unrelated sentences randomly selected (50%)

  Used as in contrastive learning.

Next sentence prediction

The output embedding associated with the [CLS] token represents the next sentence prediction.



Practical issues

It is common to compute a representation for token w_t by **averaging** the output embeddings at t from each of the topmost layers of the model.

`[CLS]` embedding also used as **sentence embedding** for tasks involving single sentence classification.

Other models

BERT was significantly undertrained.

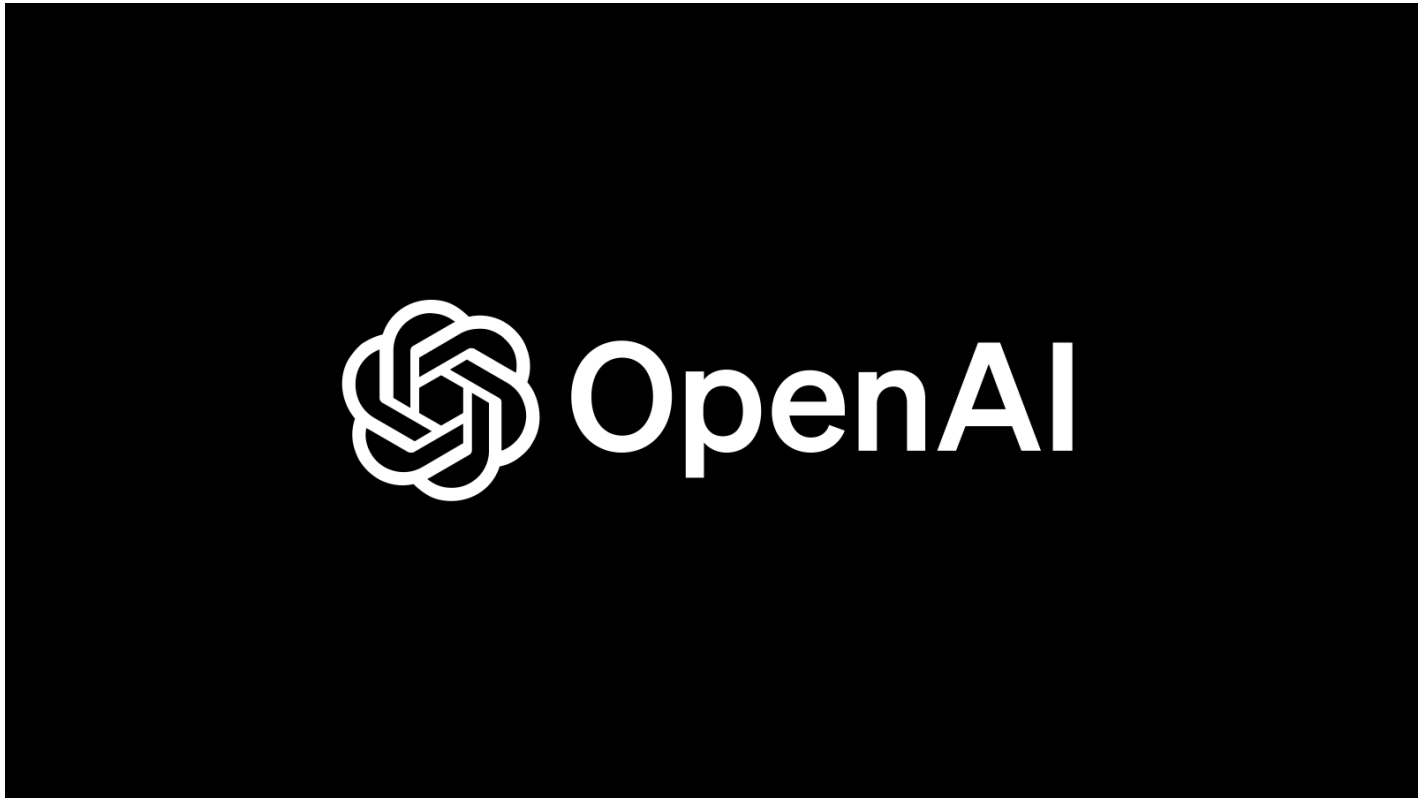
RoBERTa (Robustly Optimized BERT Approach) is based on a novel recipe for training BERT

- longer training over more data ($\times 10$) with larger batches
- removing NSP pre-training objective
- dynamically changing the masking pattern

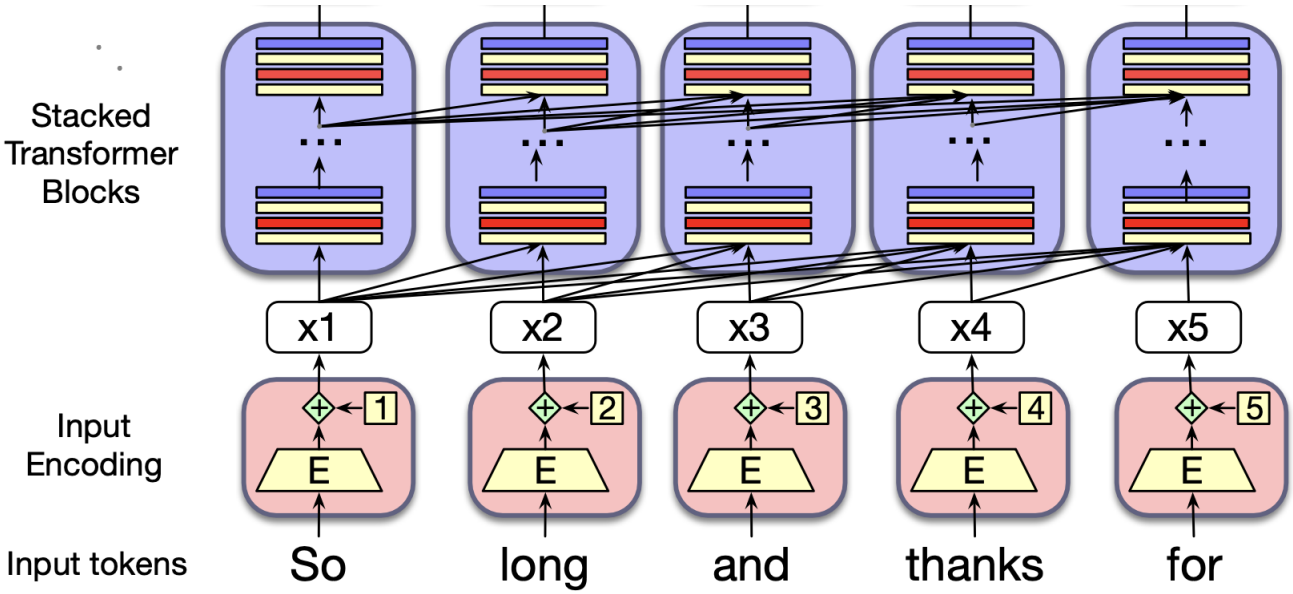
  BERT pre-training data are masked once for all
- using a byte level BPE tokenizer with larger vocabulary

Several more variants of BERT: ALBERT, BART, SpanBERT, SBERT, mBERT.

GPT-*n*



GPT (Generative Pre-training Transformer) produces word representations conditioning on left context only.



The architecture is based on the transformer decoder.

SBERT

Samuel Ferrara on Unsplash

SBERT

In some sentence-pair regression tasks, training BERT on all sentence pairs is computationally unfeasible.

Examples: semantic textual similarity, answer sentence selection.

Alternatively, individual **sentence embeddings** can be computed

- by averaging the BERT output layer, known as BERT embeddings, or
- by using the output of the `[CLS]` token

The above methods result in **bad** sentence embeddings, often worse than averaged GLoVE embeddings.

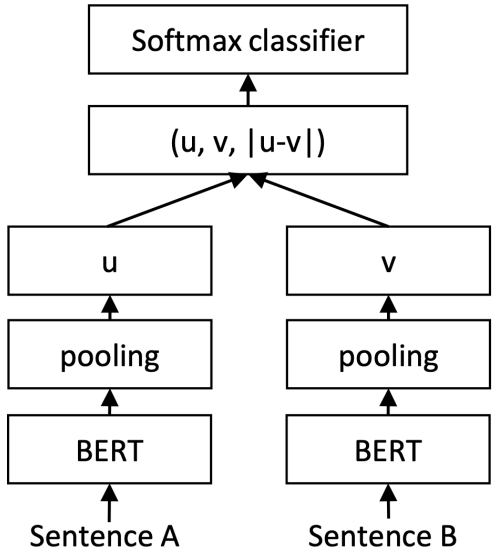
SBERT

Sentence-BERT, or SBERT for short, is an efficient and effective method to compute sentence embeddings.

SBERT uses a **siamese neural network**, a special network that contains two identical subnetworks sharing their parameters.

Siamese networks are most commonly used to compute **similarity scores** for the inputs, and have many applications in computer vision.

SBERT training



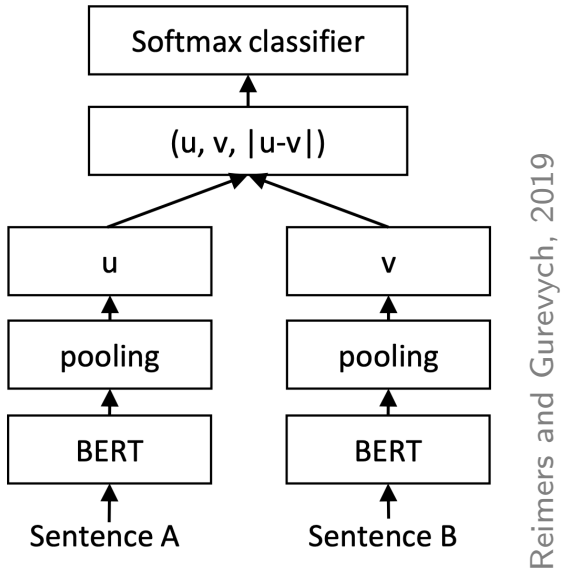
Reimers and Gurevych, 2019

SBERT concatenates vectors u, v, and the element-wise difference $|u - v|$ and uses trainable matrix $\mathbf{W} \in \mathbb{R}^{3n \times k}$ to compute

$$o = \text{softmax}(\mathbf{W}[u; v; |u - v|])$$

with n dimension of the embeddings and k number of labels.

SBERT training



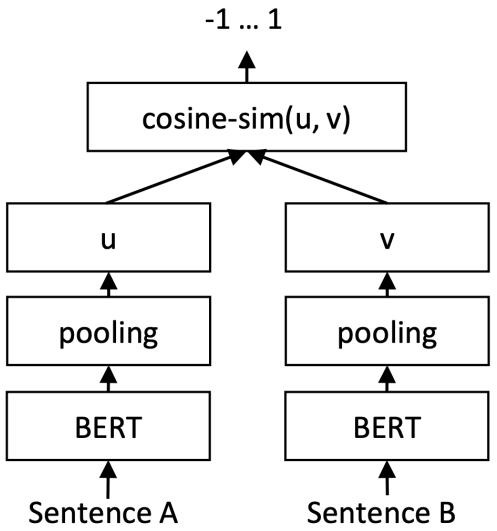
Reimers and Gurevych, 2019

BERT is **fine-tuned** (see second part of lecture).

Parameter updating is mirrored across both sub-networks.

Optimization uses cross-entropy loss.

SBERT inference



Reimers and Gurevych, 2019

The two embeddings are compared using a cosine similarity function, which will output a similarity score for the two sentences.