Natural Language Processing

Lecture 11
Contextual embeddings and transformers

2021

Contextual embeddings

Word embedding limitations

Traditional co-occurrence matrix-based word vectors and the first generation of neural word embeddings had several important limitations:

- *Context independence:* One surface form has only one representation. E.g., *bank* will have the same embedding in the sentences

  I went to my bank to withdraw some money.

  We explored the river bank.

  even though the two meanings are clearly different.

Word embedding limitations cont.

- *Words are black boxes:* Words have internal structure: they consist of characters and can be composed of several morphemes, but Word2vec, GloVe etc. ignore word internals.
- *No useful representation for unseen or rare words:* As words are treated as black boxes, these models cannot produce useful representations for words that do not occur or are very rare in the training corpus.
- *Good coverage requires huge model dimensions:* A word gets a meaningful representation only if it's explicitly included in the vocabulary of the model, but the memory consumption is typically a linear function of the covered vocabulary.

Contextual embeddings
Embedding limitations
Contextual embeddings
ELMo
FLAIR
Transformers
Transformer-based contextual embeddings
References

Word embedding limitations cont.

The problem of utilizing internal word structure, handling OOV words and reducing the vocabulary size was efficiently solved both by

- fastText embeddings, and, especially,
- subword embeddings,

but these embeddings were still *static*, i.e., mapped tokens of the same form to the same embedding vector. One of the most important recent developments in NLP has been the appearance of *contextual embeddings*, which, in contrast, can vary the embedding of the same surface form from context to context to reflect linguistic differences.

Contextual embeddings

Contextual embeddings are word or subword representations produced by deep networks (commonly LSTM or self-attention based) that are (pre)trained on self-supervised, broadly language modeling objectives.

Unlike static embeddings, these representations cannot be simply stored and deployed in the form of lookup tables, because they are *dynamically* computed for each token based on their context: For a $\mathbf{w} = \langle w_1, \dots, w_n \rangle$ input token sequence the network produces an

$$E(\langle w_1, \dots, w_n \rangle) = \langle E_{\mathbf{w}}(w_1), \dots, E_{\mathbf{w}}(w_n) \rangle$$

embedding sequence.

Contextual embeddings cont.

Because of this dynamic nature, the network itself has to be used as a *feature extractor module*.

The envisaged use of contextual embedding producing networks in NLP is similar to the role of processing pipelines in traditional NLP: they are supposed to produce feature vectors that are useful for downstream tasks, in fact, the hope is that only little further processing (e.g., in the form of shallow neural networks) is needed to build useful NLP models.

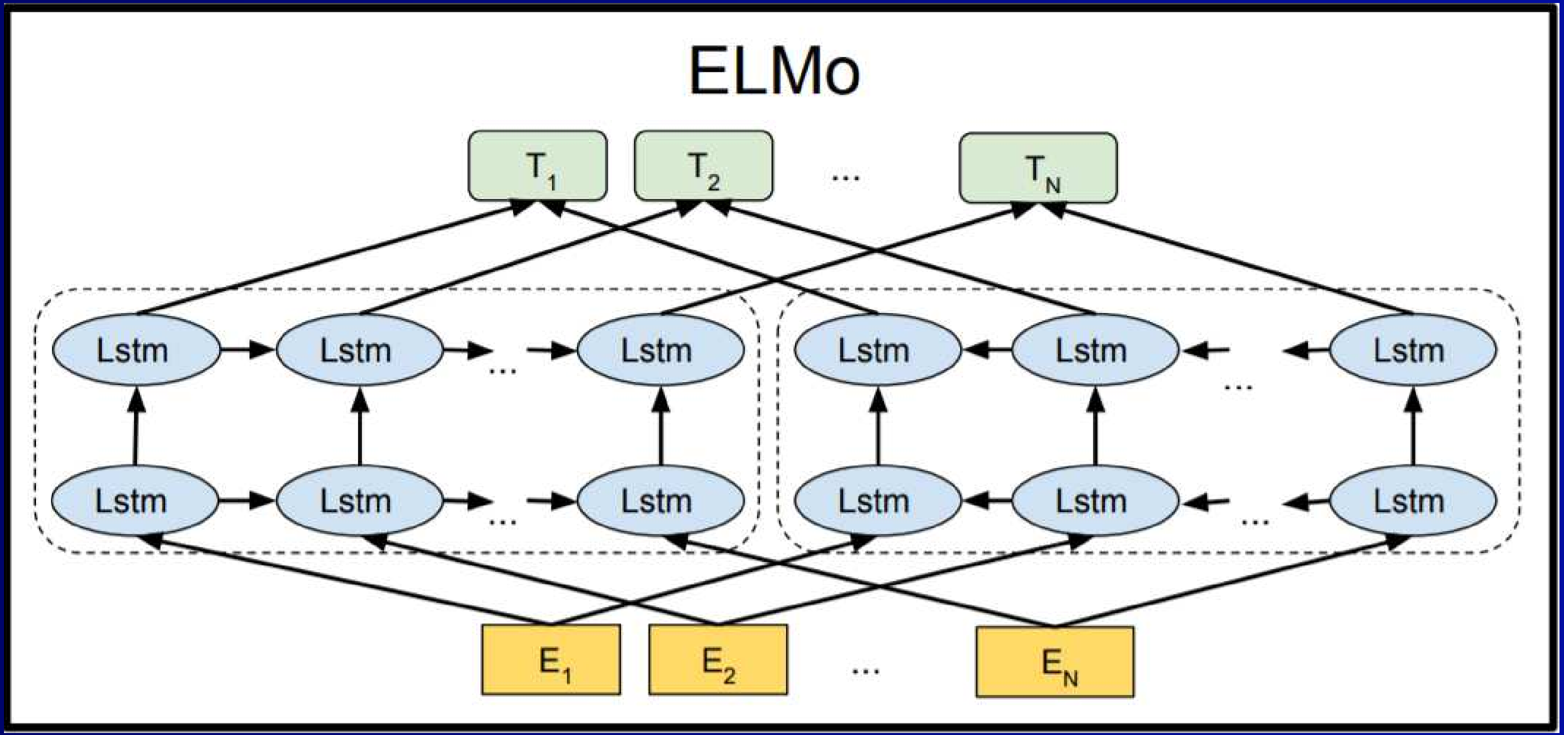
The huge difference is that contextual embeddings can be learned in a *self-supervised fashion*, without costly supervised training sets.

ELMo (Allen Institute for AI, 2018)

ELMo (Embeddings from Language Models, Peters et al. [2018]), the historically first important contextual embedding model, learns word representations via two standard one-directional language modeling tasks.

The architecture starts with producing context-independent embeddings using character-level convolutions, and then uses forward and backward bidirectional LSTM layers (their number, n, is a changeable hyperparameter) to predict the next/previous token via weight-shared softmax layers:

ELMo cont.



(Figure from Peters et al. [2018]).

At first approximation, the context dependent embeddings are all the $2n+1$ intermediate representations produced by the model ($2n$ contextual LSTM-based and one static character-based).

ELMo cont.

Although these vectors together can be considered the ”full” ELMo representation, for actual downstream NLP tasks ELMo’s creators actually suggest not to use this very high-dimensional representation, but a lower dimensional combination of these vectors. Solutions they suggest are

- simply taking the concatenation of the output of the top LSTM layers (forward and backward);
- learning a task-specific linear combination of the ELMO representations on the supervised task.

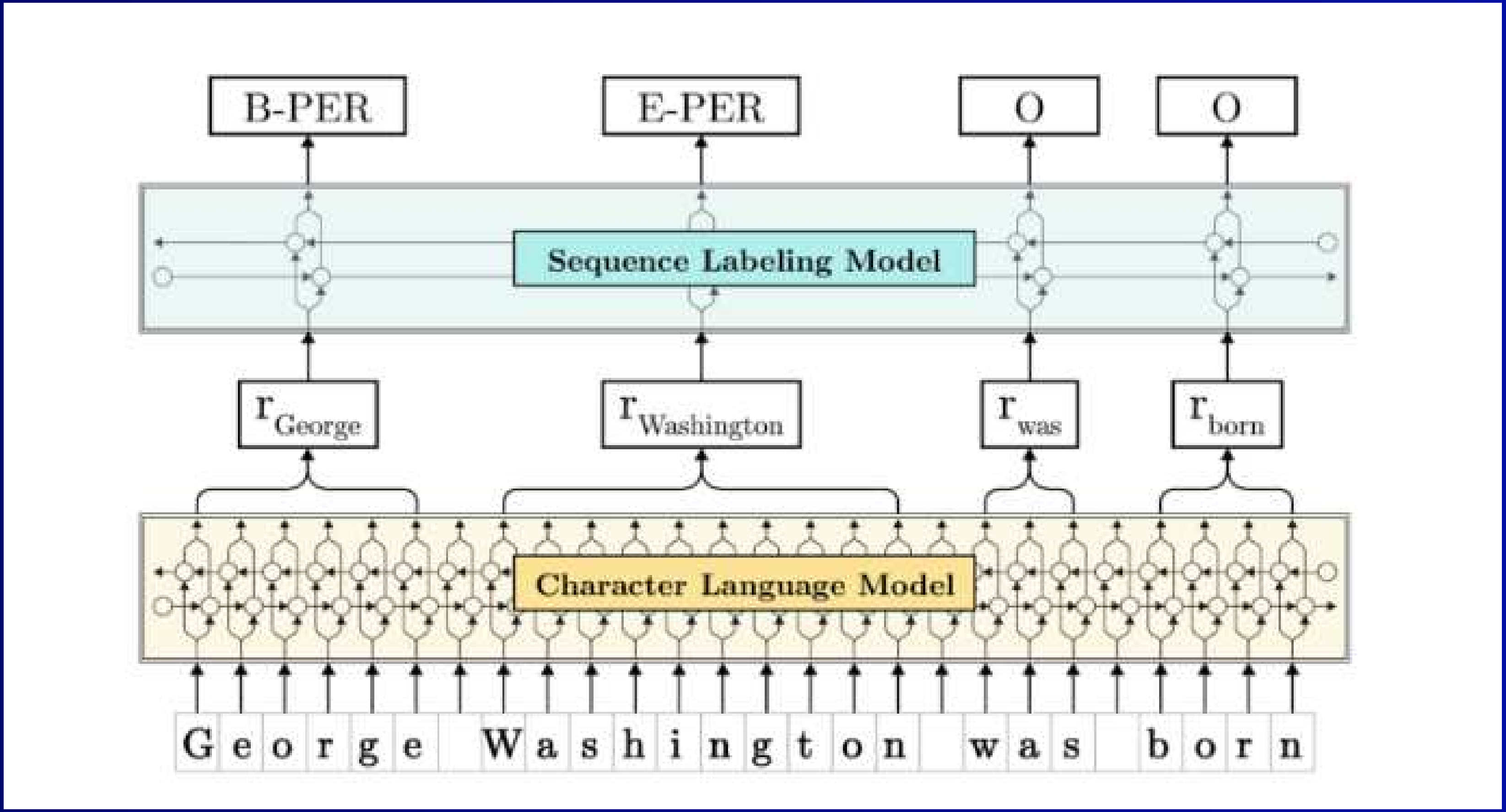
FLAIR (Zalando, 2018)

FLAIR [Akbik et al., 2018] is a contextual embedding model closely related to ELMo, but

- consists exclusively of recurrent *character level* language models (a forward and a backward one),
- produces token-level embeddings from the LSTM hidden states at the first and last character of tokens (first character from the backward and last from the forward LM).

FLAIR embeddings proved to be pretty useful in sequence tagging tasks, and shallow models using them are currently 2nd best (!) in NER and POS-tagging.

FLAIR cont.



(Figure from Akbik et al. [2018]).

Interlude: Transformers

Contextual embeddings and transformers

The first contextual embeddings were produced by RNNs, but later developments were made possible by the discovery of *transformers* [Vaswani et al., 2017], a new type of architecture using attention mechanisms as full-fledged layers, not as auxiliary RNN components.

Before discussing the recent transformer-based embedding models, we discuss shortly this new architecture and its building blocks:

- attention as soft dictionary lookup,
- self-attention layers, and
- transformer modules.

Attention as soft dictionary lookup

Recall that attention mechanisms provide *query-based aggregation* of an $\langle \mathbf{x}_1, \dots, \mathbf{x}_n \rangle$ vector sequence: given an $\mathbf{x}^*$ *query vector*, they calculate a sequence of relevance scores

$$\mathbf{s} = \langle s(\mathbf{x}_1, \mathbf{x}^*), \dots, s(\mathbf{x}_n, \mathbf{x}^*) \rangle$$

and returned the weighted sum

$$\mathrm{softmax}(\mathbf{s}) \cdot \langle \mathbf{x}_1, \dots, \mathbf{x}_n \rangle$$

as a summary or aggregation of the $\mathbf{x}_i$-s according to their relevance score.

Attention as soft dictionary lookup cont.

The $s(\cdot, \cdot)$ scoring function varies, we saw that one option is to use the scaled dot-product:

$$s(\mathbf{x}_i, \mathbf{x}^*) = \frac{\mathbf{x}_i \cdot \mathbf{x}^*}{\sqrt{d}}.$$

where d is the dimensionality of $\mathbf{x_i}$ and $\mathbf{x}^*$.

Building on this schema, the transformer attention mechanism makes a crucial change: treats $\langle \mathbf{x}_1, \dots, \mathbf{x}_n \rangle$ as a *dictionary*, for which there are $\mathcal{K}(\cdot)$ and $\mathcal{V}(\cdot)$ mappings that map each $\mathbf{x}_i$ to a corresponding $\mathcal{K}(\mathbf{x}_i)$ key and $\mathcal{V}(\mathbf{x}_i)$ value.

Attention as soft dictionary lookup cont.

Assuming also a $\mathcal{Q}(\cdot)$ *query* mapping which maps $\mathbf{x}^*$ to the range of $\mathcal{K}(.)$ (the “key-space”), scoring can be reformulated as calculating dot-product based similarity scores between the query and keys

$$s(\mathbf{x}_i, \mathbf{x}^*) = \frac{\mathcal{K}(\mathbf{x}_i) \cdot \mathcal{Q}(\mathbf{x}^*)}{\sqrt{d}},$$

(d is now the dimensionality of the “key-space”) and the retrieved value will be the weighted sum

$$\mathrm{softmax}(\langle s(\mathbf{x}_1, \mathbf{x}^*), \dots, s(\mathbf{x}_n, \mathbf{x}^*)\rangle) \cdot \mathcal{V}(\langle \mathbf{x_1}, \dots, \mathbf{x}_n\rangle).$$

Attention as a layer cont.

The outlined attention mechanism can be used as a standalone layer for transforming an input vector sequence $\mathbf{I} = \langle \mathbf{i}_1, \ldots, \mathbf{i}_n \rangle$:

Given another $\mathbf{X} = \langle \mathbf{x}_1, \ldots, \mathbf{x}_m \rangle$ sequence and $\mathcal{K}(\cdot), \mathcal{V}(\cdot), \mathcal{Q}(\cdot)$ mappings, for each $\mathbf{i_k}$ input, we can calculate the corresponding $\mathcal{Q}(\mathbf{i}_k)$ query, and use this with $\mathcal{K}$ and $\mathcal{V}$ to *attend to* $\mathbf{X}$ and compute a corresponding attention response $\mathbf{o}_k$.

The result is an $\mathbf{O} = \langle \mathbf{o}_1, \ldots, \mathbf{o}_n \rangle$ sequence of outputs, which collectively is the layer's output for the input $\mathbf{I}$.

Attention as a layer cont.

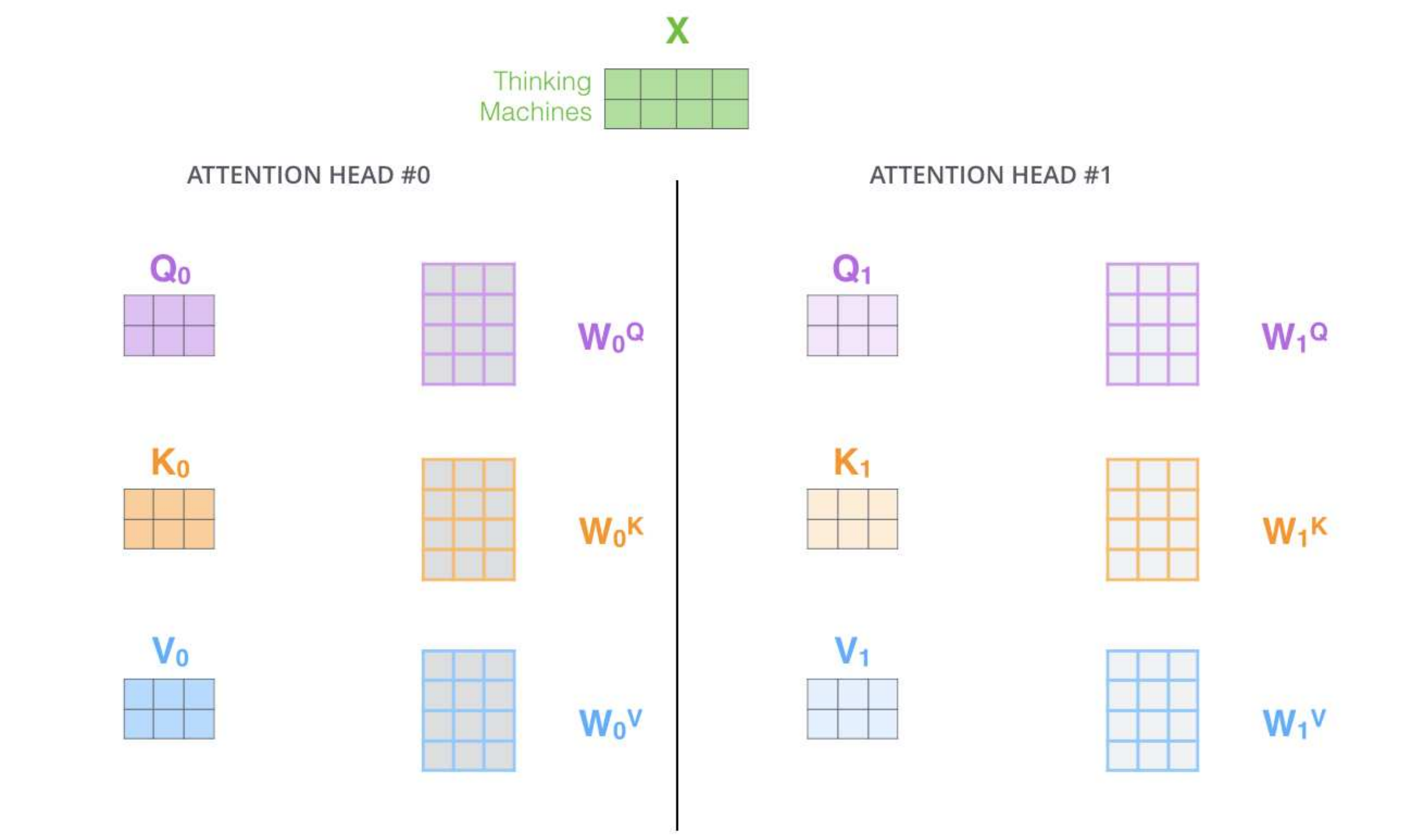
Depending on where the layer attends to (where $\mathbf{X}$ comes from) we can distinguish self- and outward attention layers.

- In *self-attention* layers, the queries generated from the input are used to query the input iself: $\mathbf{X} = \mathbf{I}$.
- In an *outward* attention layer, an external vector sequence is queried, e.g., in encoder-decoder transformer architectures a sequence created by an encoder.

As for the mappings $\mathcal{K}(\cdot), \mathcal{V}(\cdot), \mathcal{Q}(\cdot)$, all three are commonly implemented as *linear projections*, with learned weight matrices W_K, W_V, W_Q.

Multi-headed attention

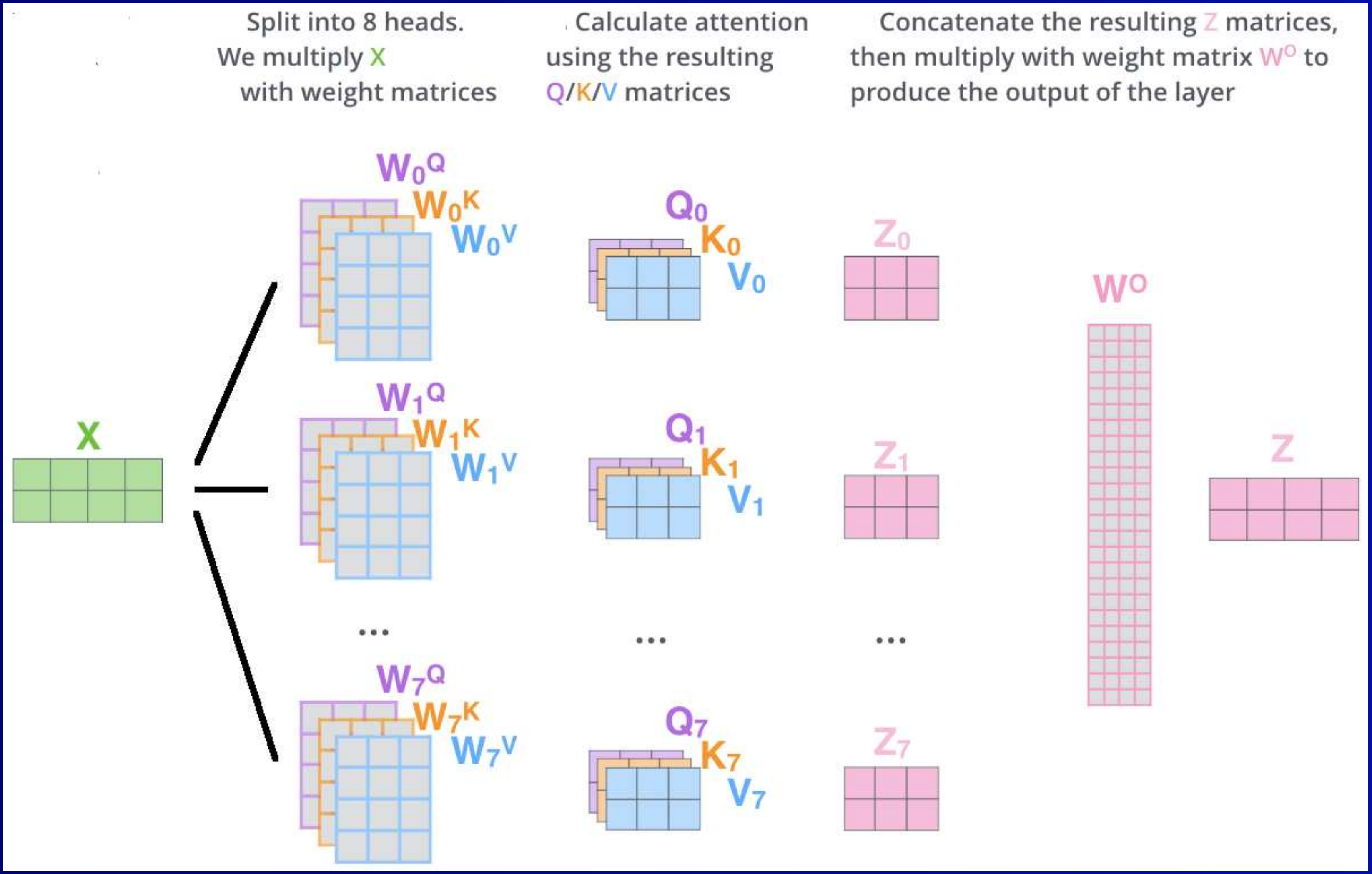
To be able to attend to multiple aspects of the input, the attention layers in transformers contain several parallel attention “heads” with different W_K, W_V, W_Q triplets:



(Figure from Alammar [2018].)

Multi-headed attention cont.

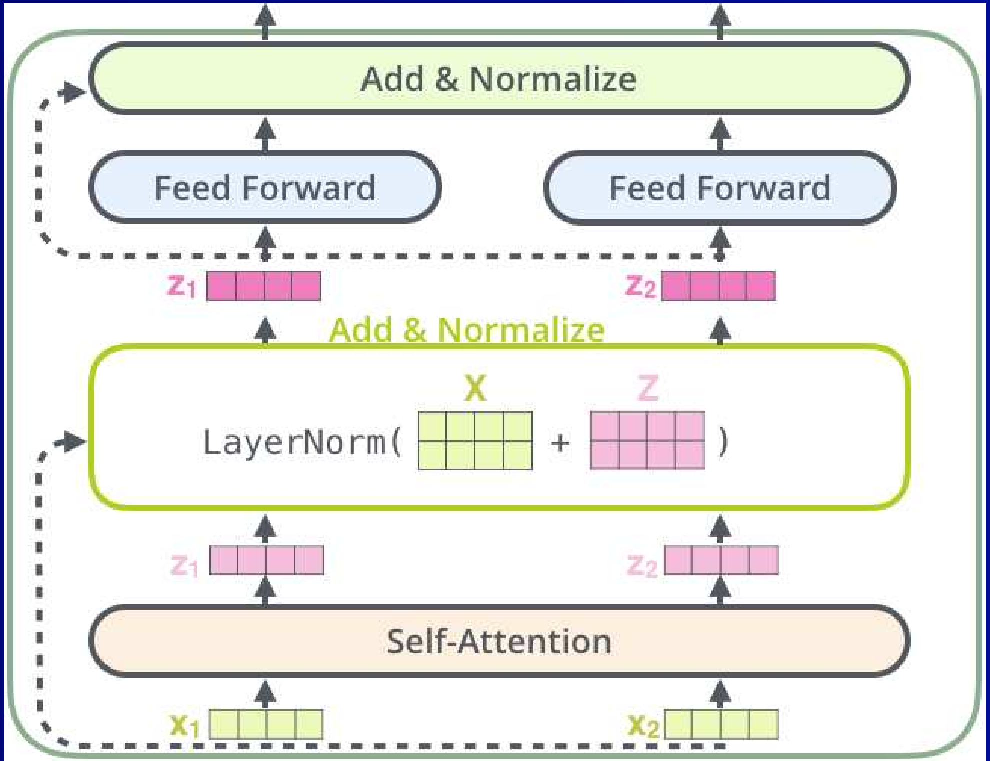
The head outputs are combined into a layer output:


Split into 8 heads.
We multiply X with weight matrices
Calculate attention using the resulting Q/K/V matrices
Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
X
W0^Q
W0^K
W0^V
W1^Q
W1^K
W1^V
...
W7^Q
W7^K
W7^V
Q0
K0
V0
Q1
K1
V1
...
Q7
K7
V7
Z0
Z1
...
Z7
W^O
Z

(Figure adapted from Alammar [2018].)

Transformer modules

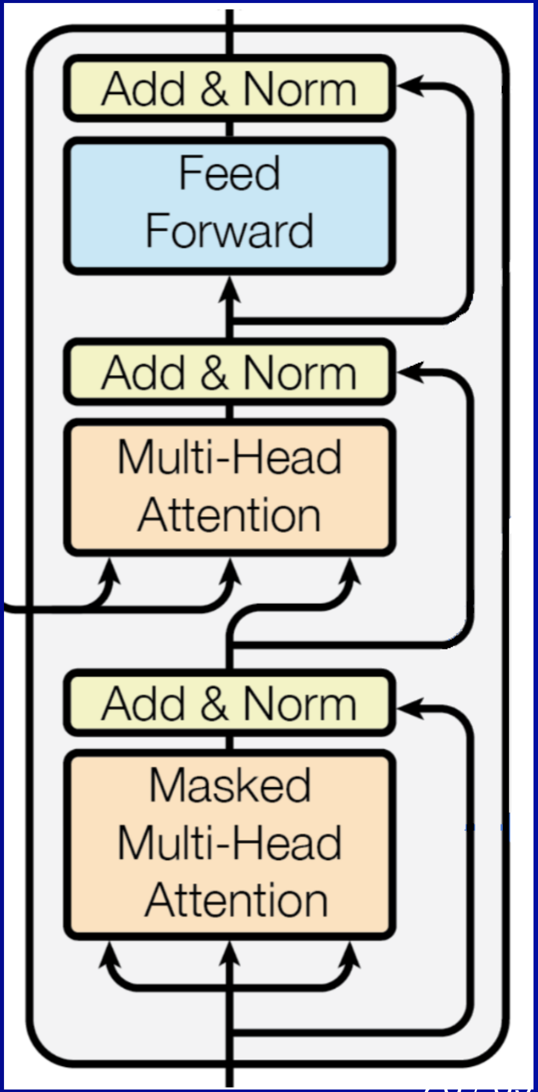
The building blocks of transformers are *transformer modules* consisting of attention and simple piecewise feedforward layers. The simplest variant contains only a single self-attention layer:



(Figure adapted from Alammar [2018].)

Transformer modules cont.

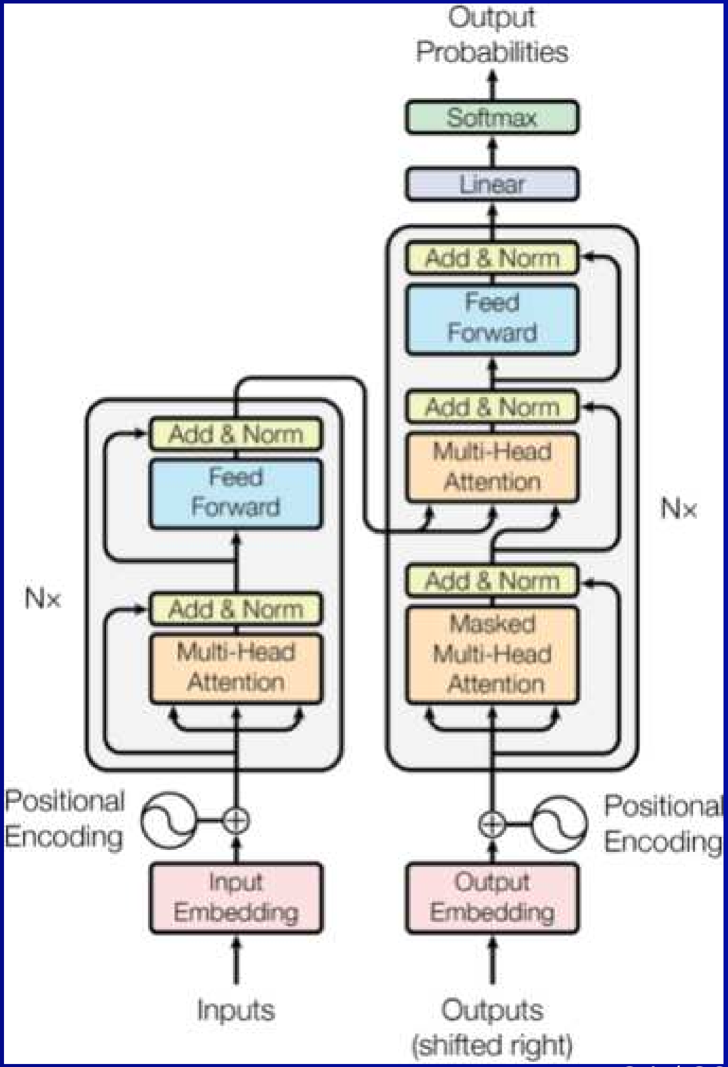
The *decoder* part of transformers is built from more complex modules: in addition to a self-attention layer, they also contains an outward attention layer, which attends to the output of the model's encoder. The self-attention layer is *masked*: attention queries from a decoder position cannot access information in later decoder positions to not use "future information" during training with teacher forcing.



(This and the next figure from Vaswani et al. [2017].)

Seq2seq transformer

The original *full transformer model* of [Vaswani et al., 2017] was a Seq2seq encoder-decoder model built exclusively of transformer blocks. During inference the decoder part predicts step-by-step, consuming already predicted output similary to RNNs, but during training it requires only a single forward pass with teacher forcing.



Transformer-based contextual embeddings

Contextual embeddings

Transformers

Transformer-based contextual embeddings

Introduction

GPT

BERT

Recent trends

Distillation

Sparse attention

Sparse attention

Few-shot learning

References

Transformer-based embeddings

The transformer architecture was first used for translation (2017), but starting from 2018, a series of transformer-based models producing contextual embeddings were developed. The most important research areas were

- finding self-supervised tasks conducive to learning high-quality representations;
- architectural improvements, importantly, finding more efficient attention variants;
- how to adapt/fine-tune the pretrained representation networks for downstream tasks.

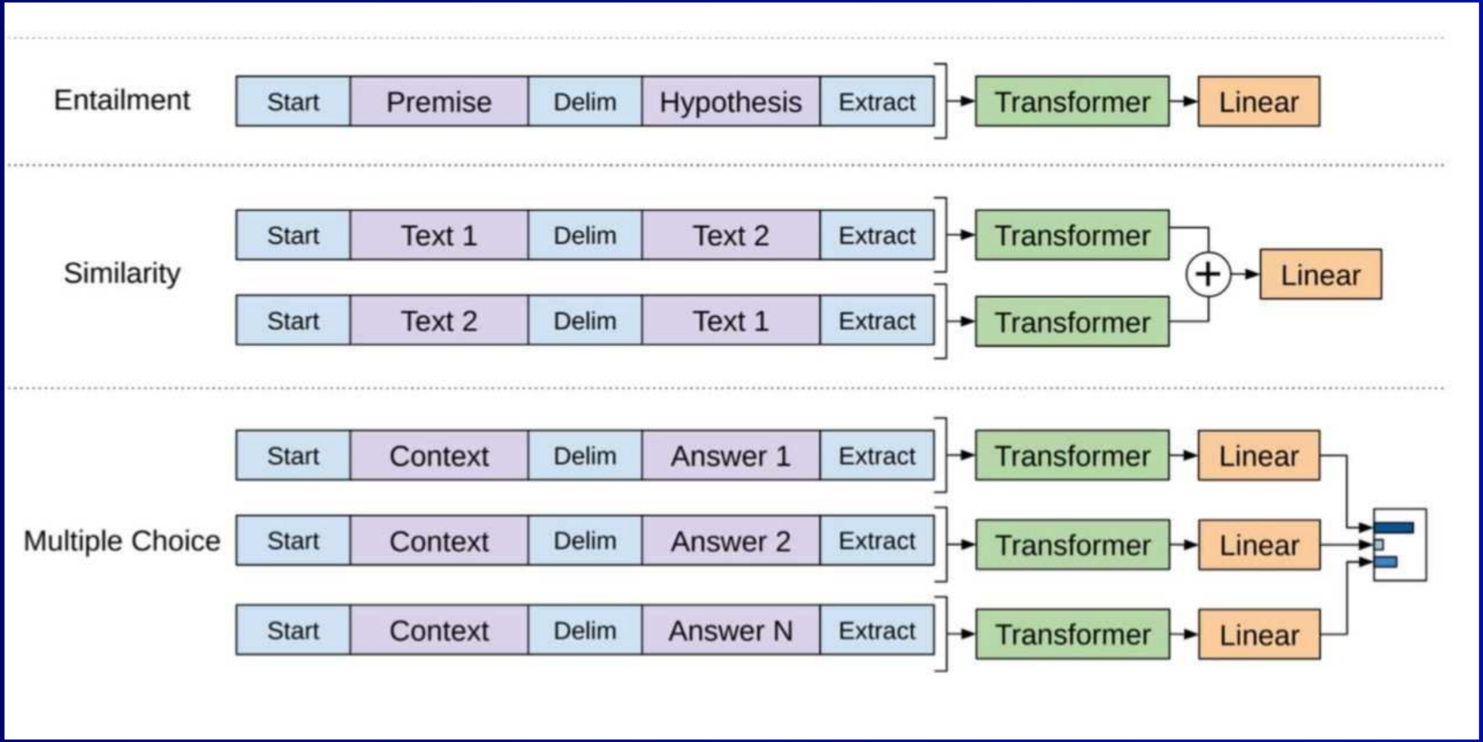
GPT (Generative Pre-Training, OpenAI, 2018)

GPT is a BPE-based, decoder only transformer model trained with a traditional ”predict the next token” language modeling objective. The contextual embeddings are simply the outputs of the top transformer module.

Similarly to ELMo, the main goal of GPT is to provide a useful pretrained ”feature extractor” module, which can be fine-tuned for supervised NLP tasks. Fine-tuning means changing also the pretrained GPT weights in an end-to-end fashion on the supervised downstream task.

GPT cont.

In most cases it's enough to add single linear layer(s) to get models with state-of-the-art performance (structured inputs are transformed into token sequences with special delimiter tokens [”delim”]):



(Figure from Radford et al. [2018].)

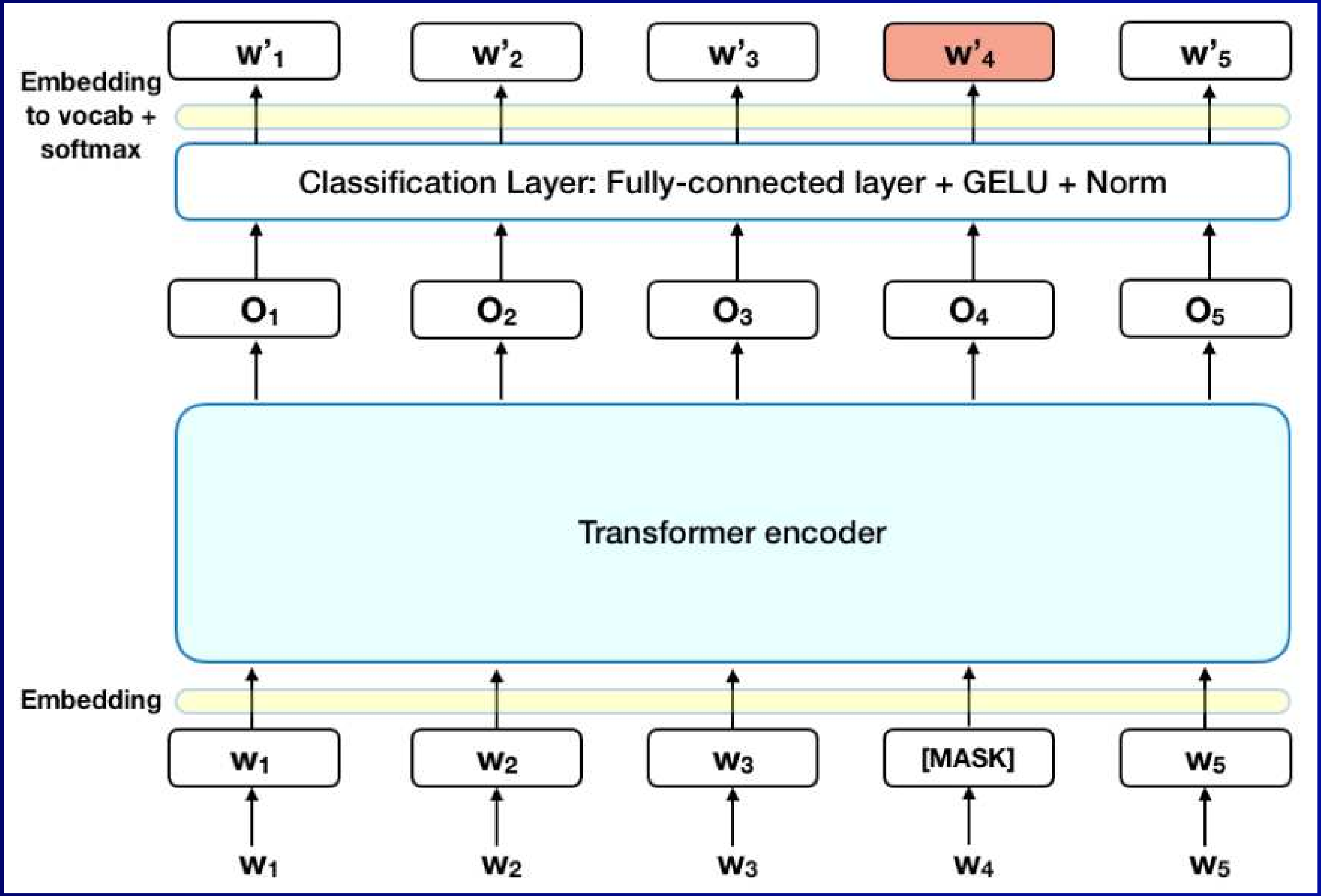
BERT (Google, 2018)

The next highly influential model was Google's BERT (Bidirectional Encoder Representations from Transformers, Devlin et al. [2018]), whose main innovations were

- the use of two new self-supervised objectives instead of traditional language-modeling:
  - masked language modeling, and
  - next sentence prediction (NSP); and
- a corresponding architectural change: the model is based on the *transformer encoder* architecture.

BERT: masked language modeling

The objective is to guess randomly masked tokens:


w'1
w'2
w'3
w'4
w'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5

(Figure from Horev [2018].)

BERT: next sentence prediction

The second objective was deciding whether two sentences followed each other in the training corpus or were randomly sampled:

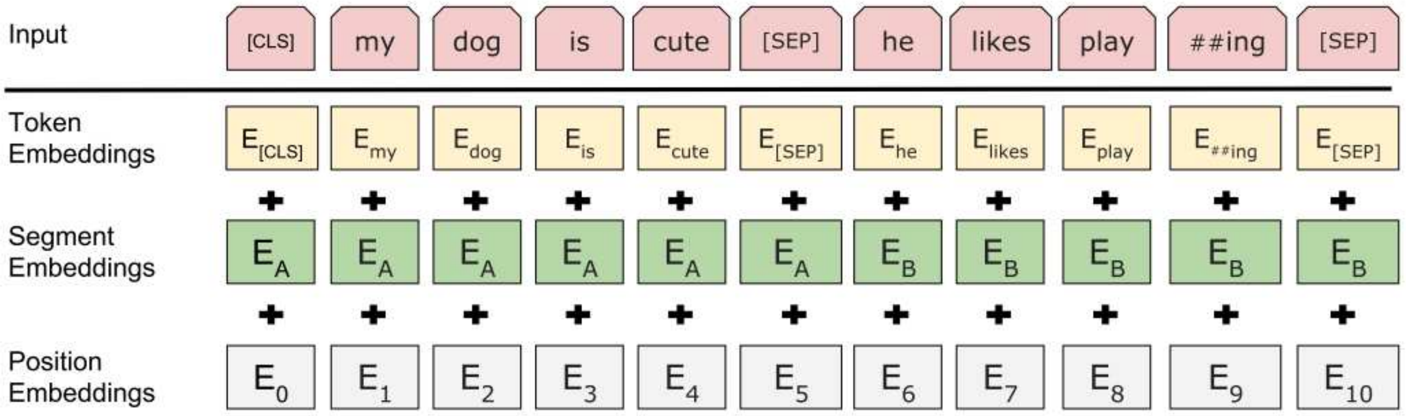


Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

(Figure from Devlin et al. [2018].)

Recent trends

Newer models have been able to improve on the state-of-the-art in NLP tasks again and again, but typically with an *increased number of parameters* and *on larger datasets*:

While the original ELMo model had 93.6 million parameters, GPT-3, the latest GPT version has 175 billion(!) parameters, and the dataset size increased from 800 million to 300 billion tokens.

Knowledge distillation

The huge increases in size led to intensive research into *knowledge distillation* methods to be able to produce smaller and more efficient models based on the original ones without significant performance loss.

A good example is *DistilBERT*, a distilled version of BERT trained to mimic BERT's output [Sanh et al., 2019]. DistilBERT retains 97% of BERT's performance, but with 40% fewer parameters and is 39% faster during inference.

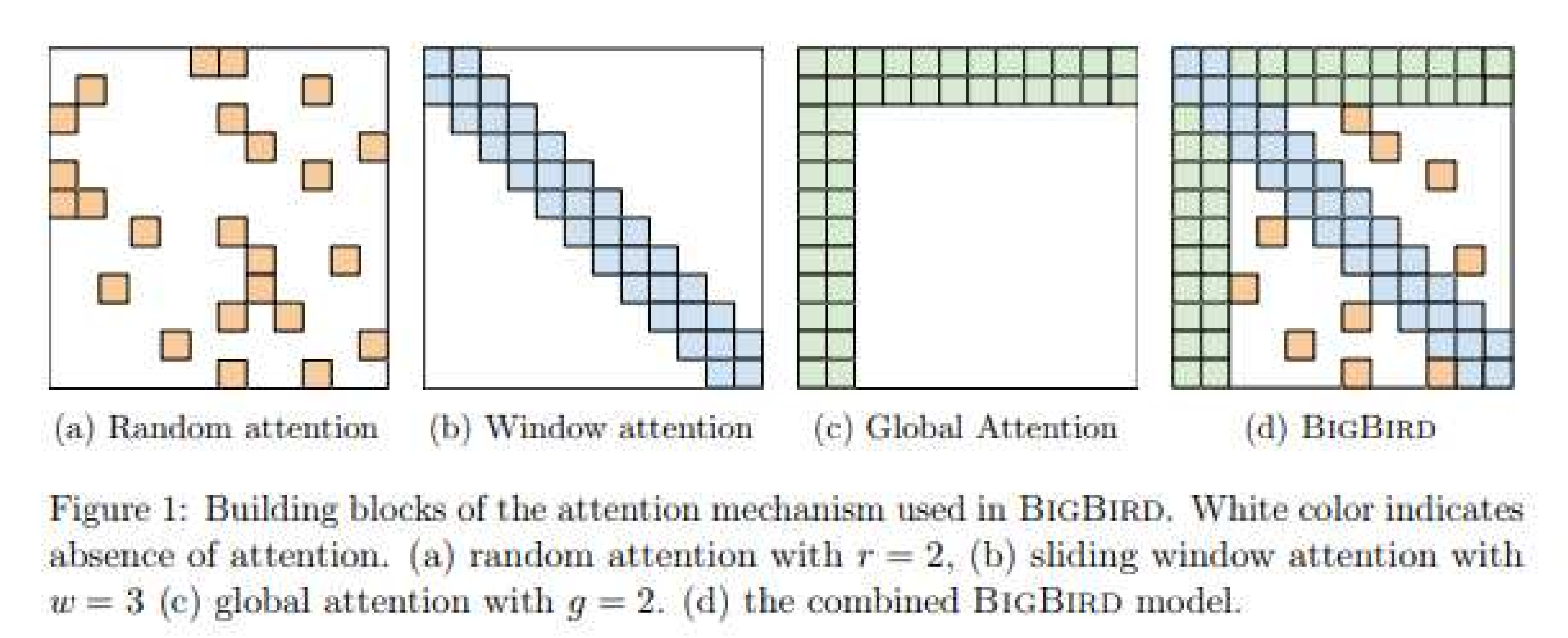
Sparse attention variants

Another way of improving efficiency has been reducing the scope of attention in the self-attention layers, since in full attention the number of dot products to be calculated is quadratic in the number of input tokens. Linear alternatives include

- *global attention*: a set of global tokens that attend to the whole sequence;
- *random attention*: for each query, a set of r random keys is calculated to which that query attends;
- *window attention*: only local neighbors within a fix radius are attended to.

Sparse attention variants cont.

The Big Bird contextual embedding model [Zaheer et al., 2020] combines all these linear attention types to increase the number of input tokens significantly without significant change in memory requirements:



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

(Figure from Zaheer et al. [2020].)

Few- one- and zero-shot learning

An interesting research direction is to try to use the model directly, without added layers and gradient updates on downstream tasks. Some recent models, most importantly, GPT-3 [Brown et al., 2020] can perform surprisingly well on various downstream tasks that are explained in the input. There are three learning settings:

- *zero-shot*: The input only contains a short description of the supervised task, and a concrete input instance prompt , e.g. “Translate English to French: cheese => ”.
- *one-* and *few-shot*: In addition to the short task description, the input also contains one or a few training examples before the prompt.

References

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018. URL http://alanakbik.github.io/papers/coling2018.pdf.

Jay Alammar. The illustrated transformer. Blog post. 2018. URL http://jalammar.github.io/illustrated-transformer/.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. URL https://arxiv.org/pdf/2005.14165.pdf.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. URL https://arxiv.org/abs/1810.04805.

Rani Horev. BERT – state of the art language model. Blog post. 2018. URL https://bit.ly/3bC6xNU.

References cont.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018. URL https://arxiv.org/pdf/1802.05365.pdf.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018. URL https://bit.ly/3qtMGop.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. URL https://arxiv.org/pdf/1706.03762.pdf.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020. URL https://arxiv.org/pdf/2007.14062.pdf.