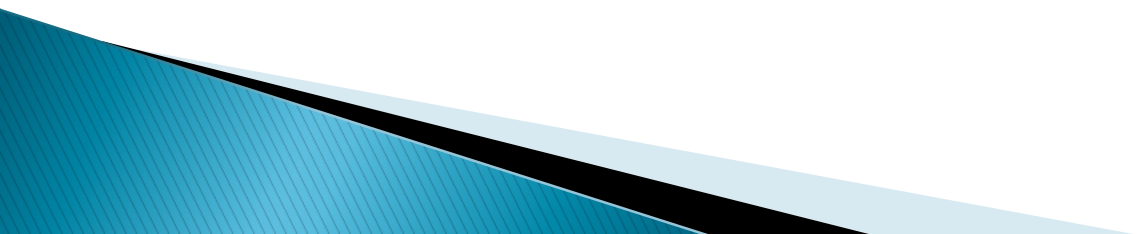
Transformers Part 2

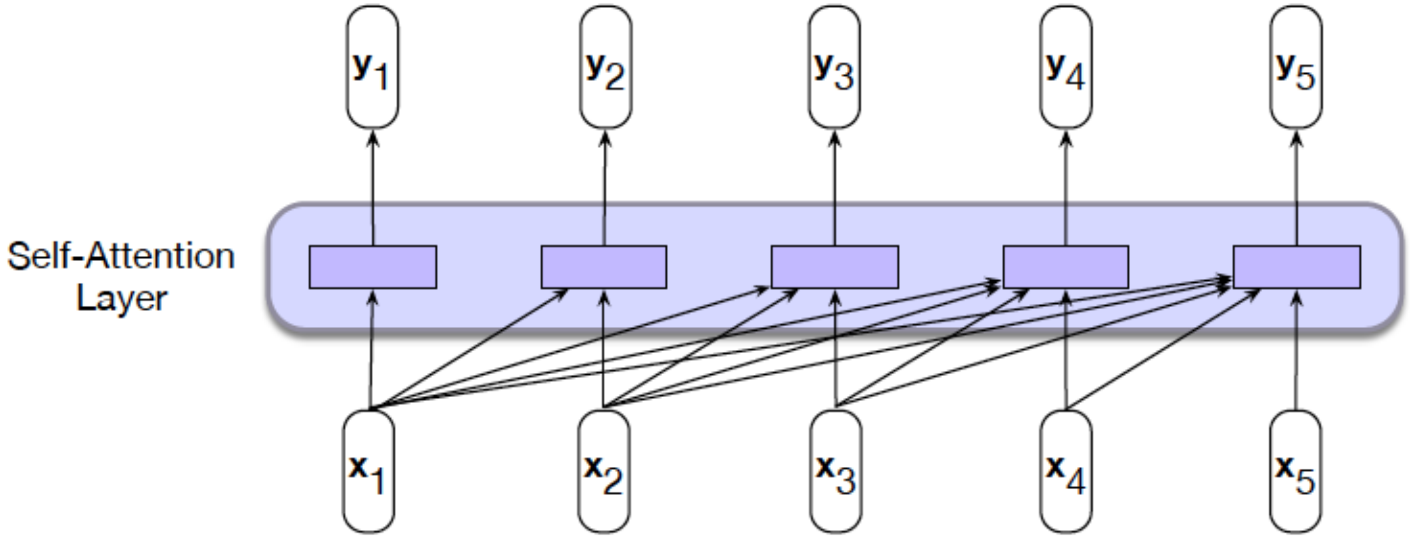
Lecture 18

Subir Varma

Language Models using Transformers



Transformer based Language Model



- The main difference between the Transformer used as Language Model, and the Transformer used for Classification is that when a sentence is fed into a Language Model Transformer during training, then the Attention calculations for word X_k cannot take into account words that occur after X_k.
- For example the Attention calculations for X_3 can take into account X_3 itself as well as X_1 and X_2, but not X_4 and X_5.

Self Attention Computation for a LM

$QK^T =$

N

q1•k1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
q2•k1	q2•k2	$-\infty$	$-\infty$	$-\infty$
q3•k1	q3•k2	q3•k3	$-\infty$	$-\infty$
q4•k1	q4•k2	q4•k3	q4•k4	$-\infty$
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

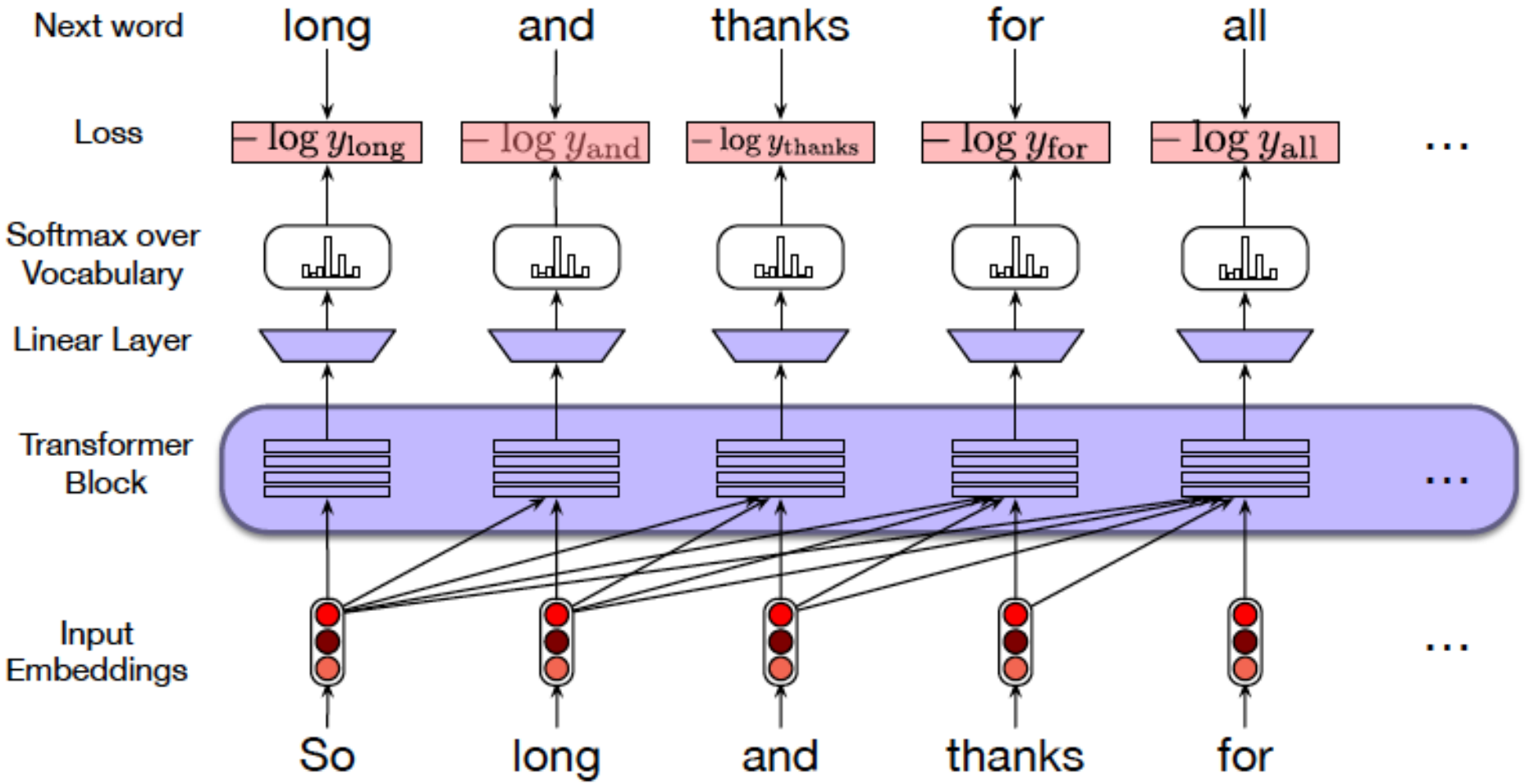
N

$$Z = softmax(\frac{QK^T}{\sqrt{d}})V$$

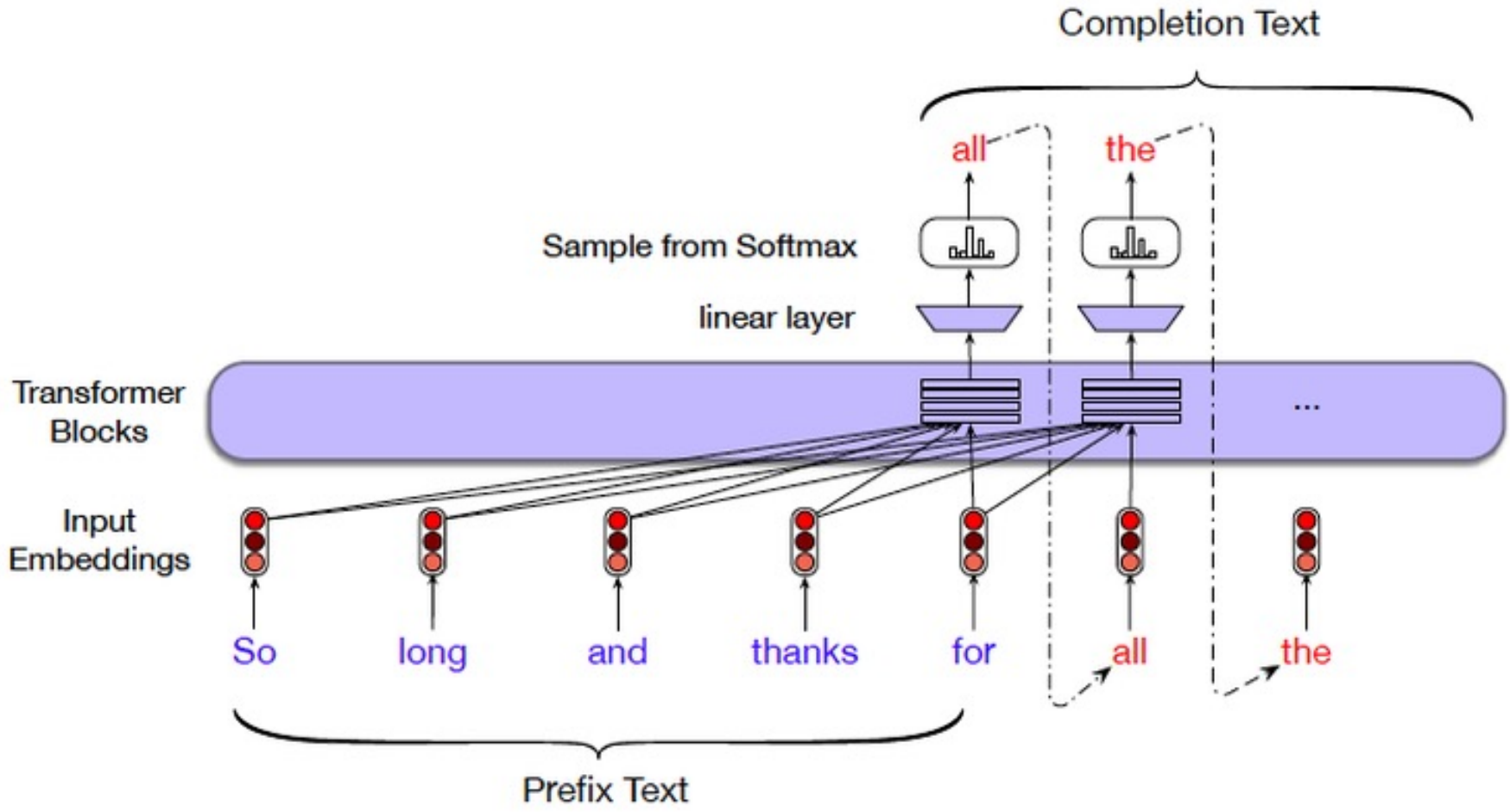
- If the upper half of this matrix is set to -infinity, then row i exhibits the correct dot product for computing the Self Attention for the i^{th} term in the input sequence.
- The mathematical operation described above is called masking, and is implemented in Keras using the *mask* argument.

Training

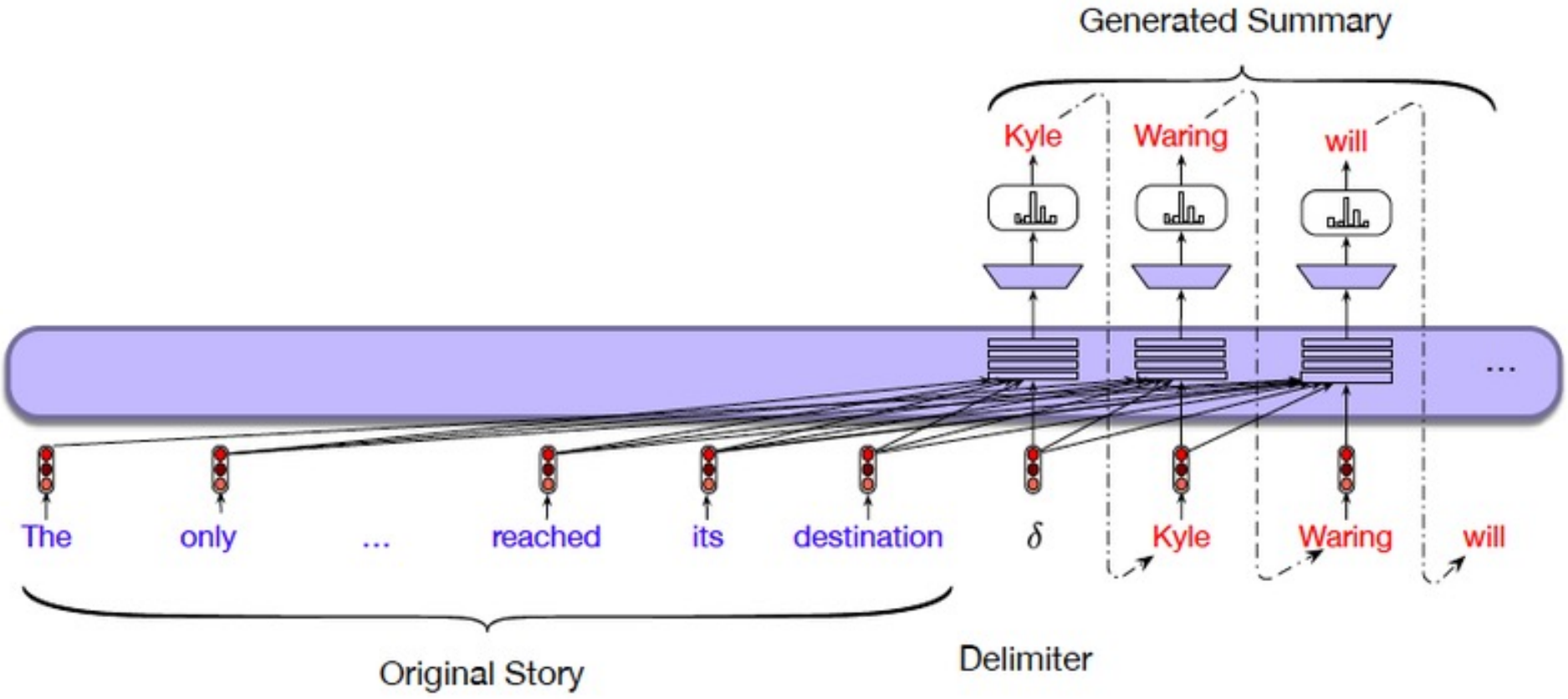
All the words are fed into the model together!



Inference: Text Completion



Text Summarization (or Translation)



GPT-3

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

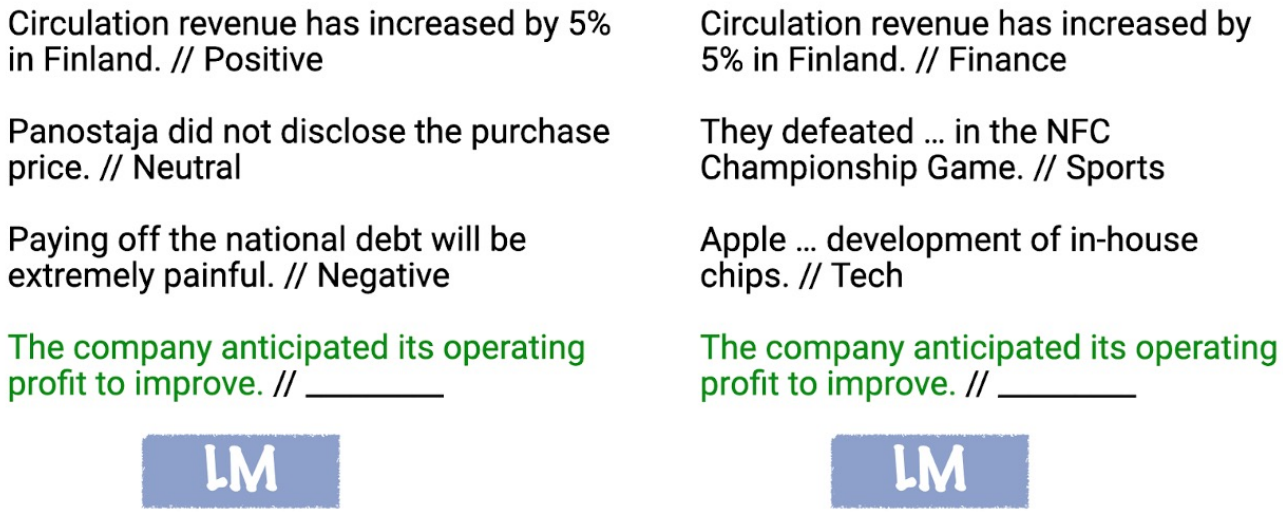
Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

- n_{params} is the total number of trainable parameters,
- n_{layers} is the total number of layers,
- d_{model} is the size of the embedding layer
- d_{head} is the dimension of each attention head.

All models use a context window of $n_{\text{ctx}} = 2048$ tokens.

In Context Learning

- In-context learning was popularized in the original GPT-3 paper as a way to use language models to learn tasks given only a few examples.
- During in-context learning, we give the LM a prompt that consists of a list of input-output pairs that demonstrate a task. At the end of the prompt, we append a test input and allow the LM to make a prediction just by conditioning on the prompt and predicting the next tokens.
- To correctly answer the two prompts below, the model needs to read the training examples to figure out the input distribution (financial or general news), output distribution (Positive/Negative or topic), input-output mapping (sentiment or topic classification), and the formatting.



Scaling Laws for Language Models

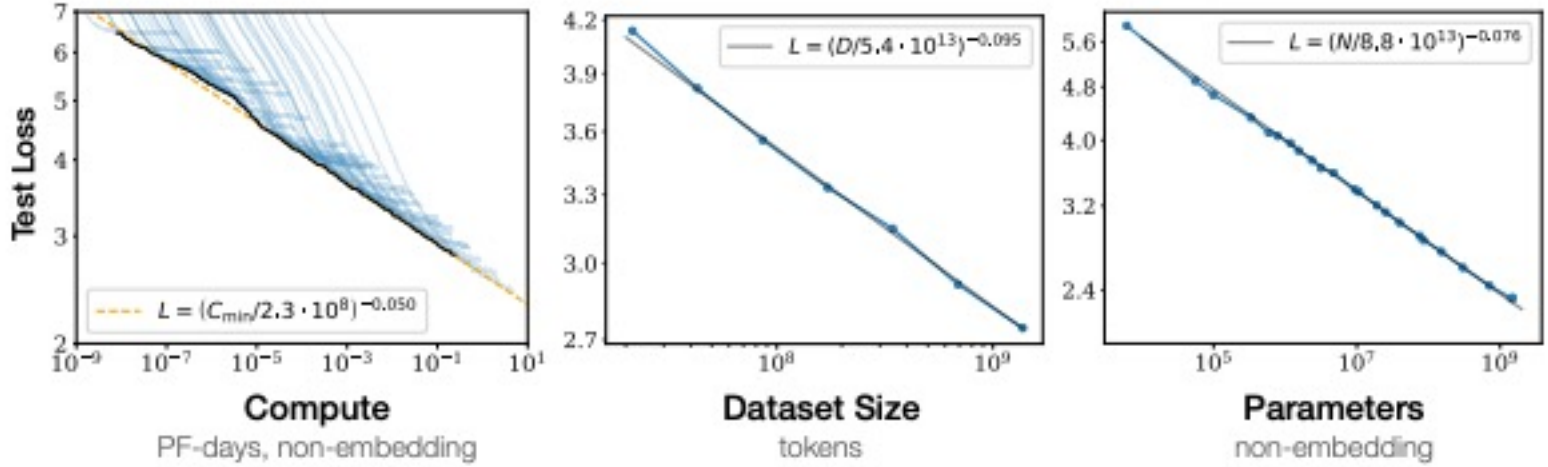


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Scaling Laws for Neural Language Models: https://arxiv.org/pdf/2001.08361.pdf
Training Compute Optimal LLMs: https://arxiv.org/pdf/2203.15556.pdf

Scaling Laws for Language Models

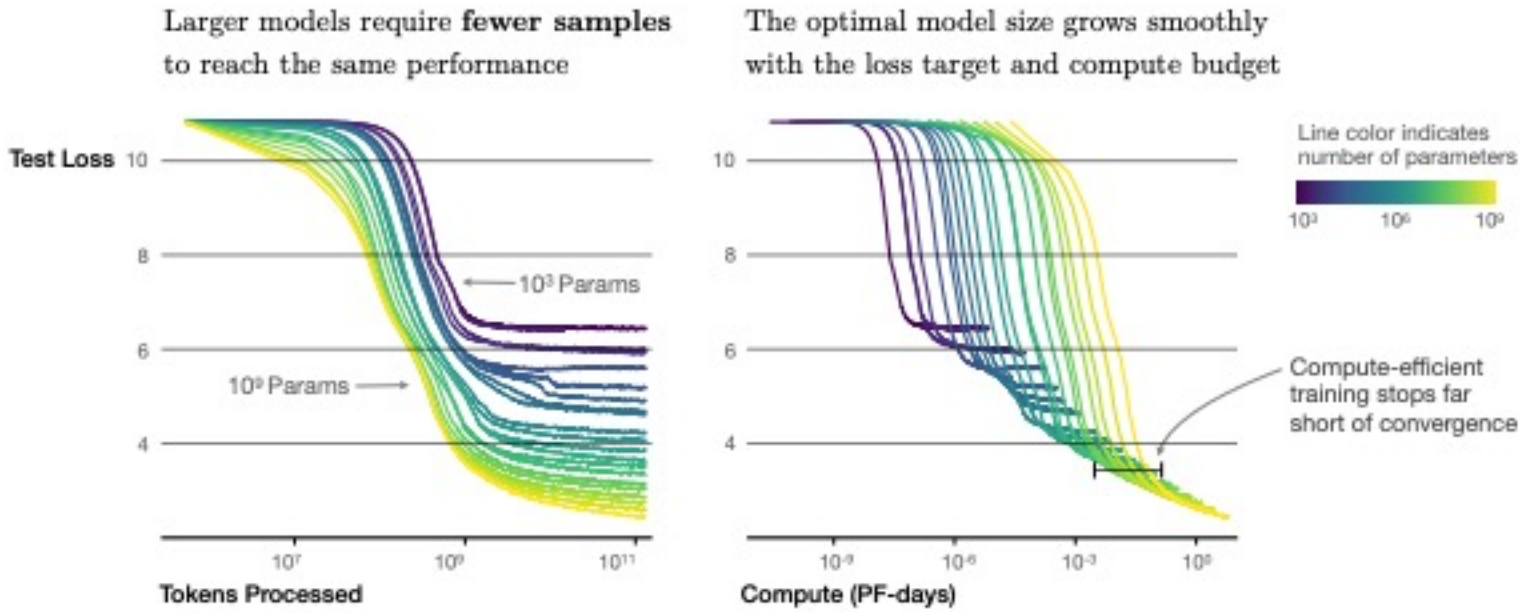
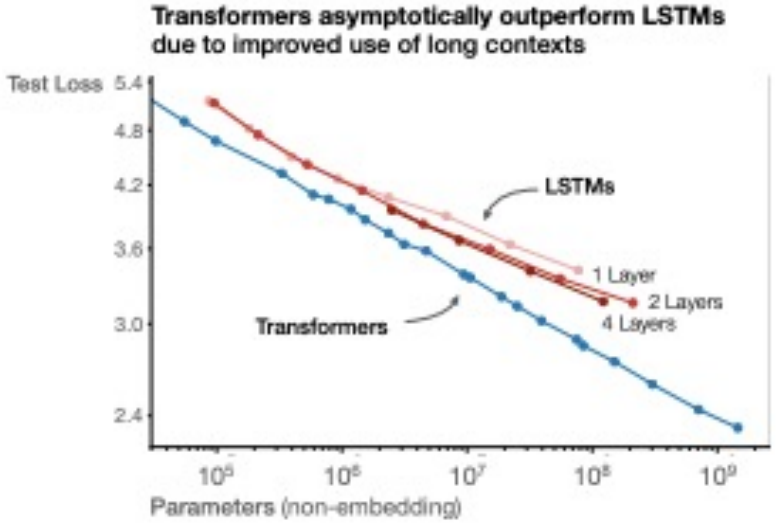
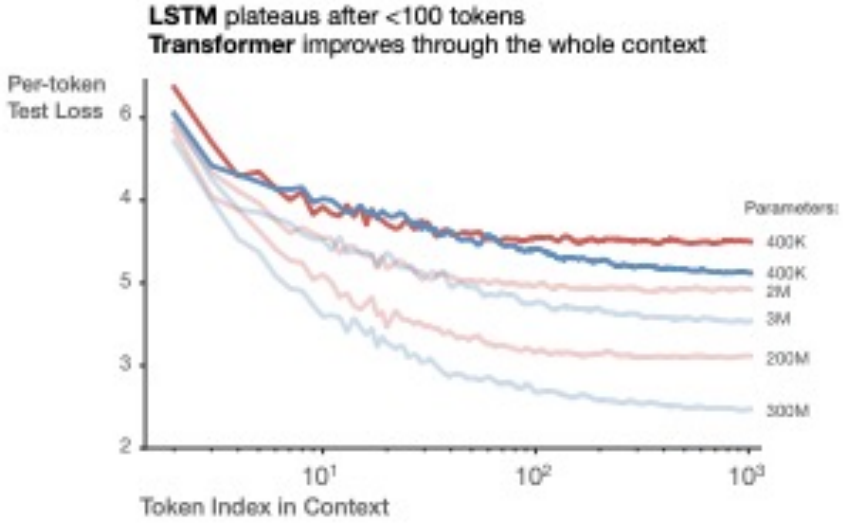


Figure 2 We show a series of language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings).

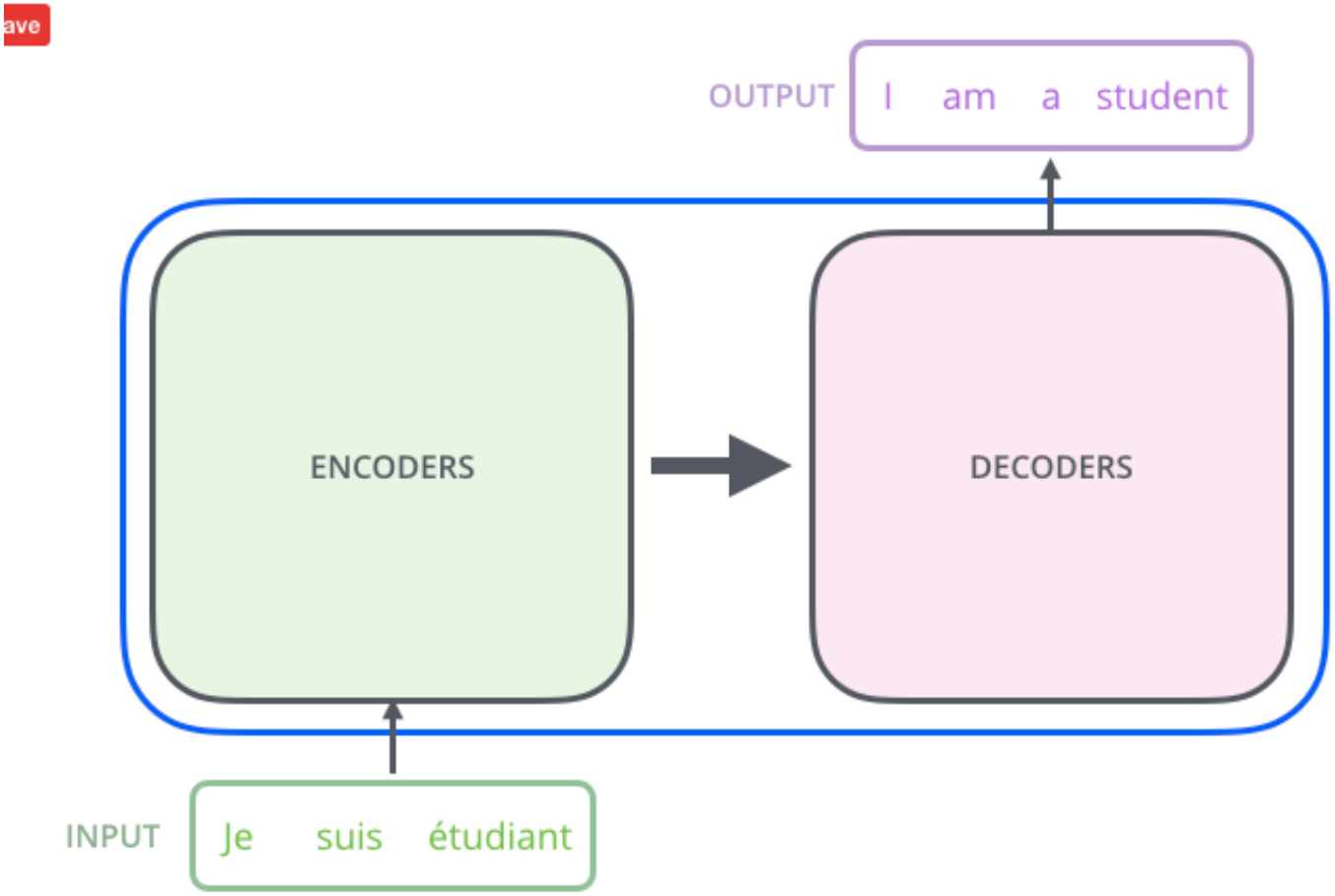
Comparison with LSTMs



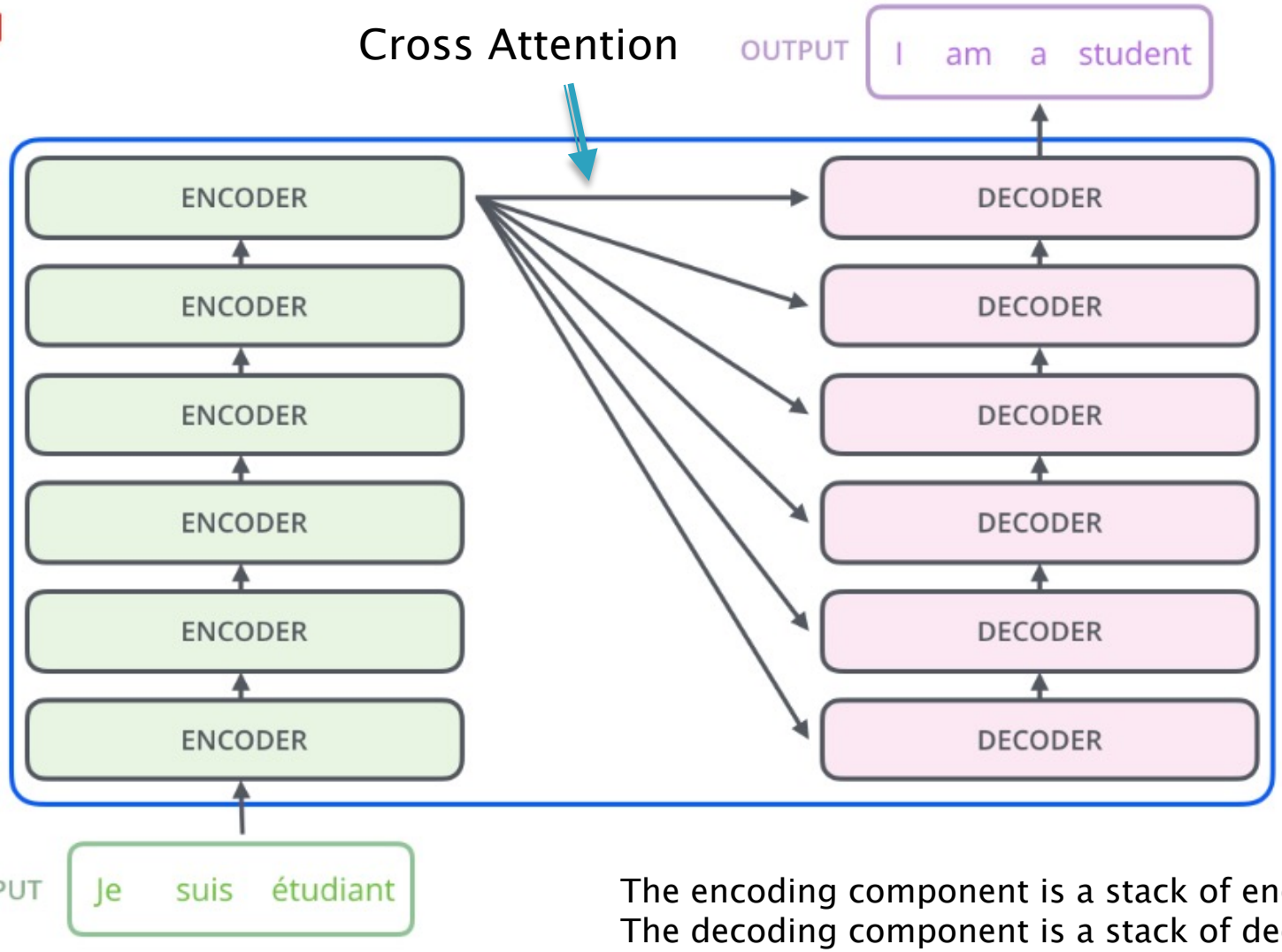


Encoder Decoders Using Transformers

Transformers: Encoder-Decoder

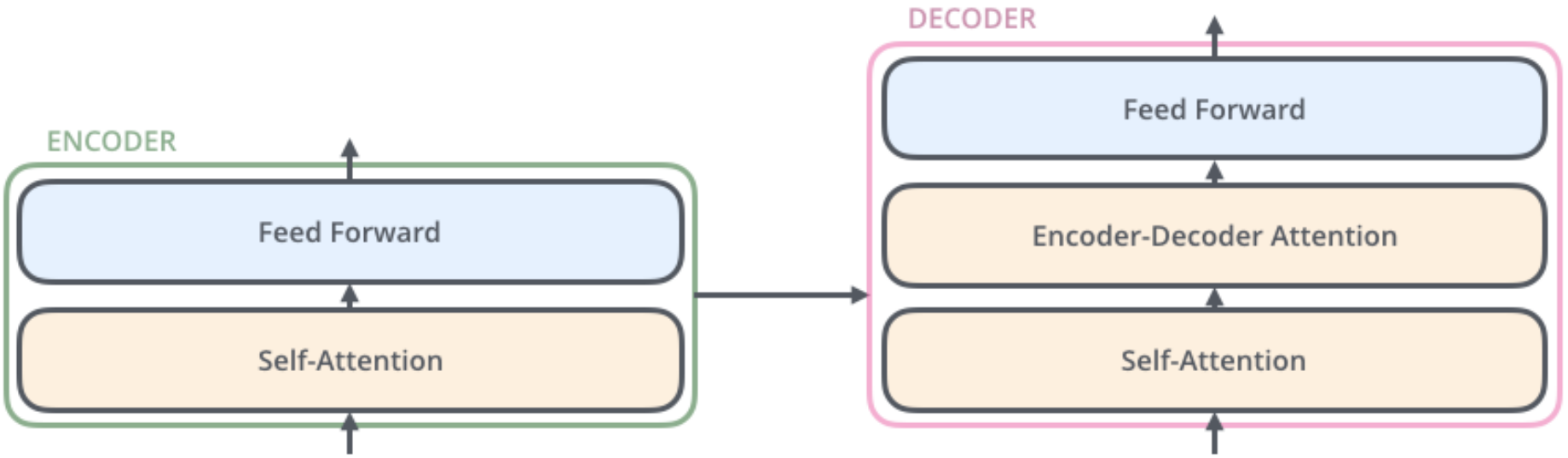


Transformers: Encoder-Decoder



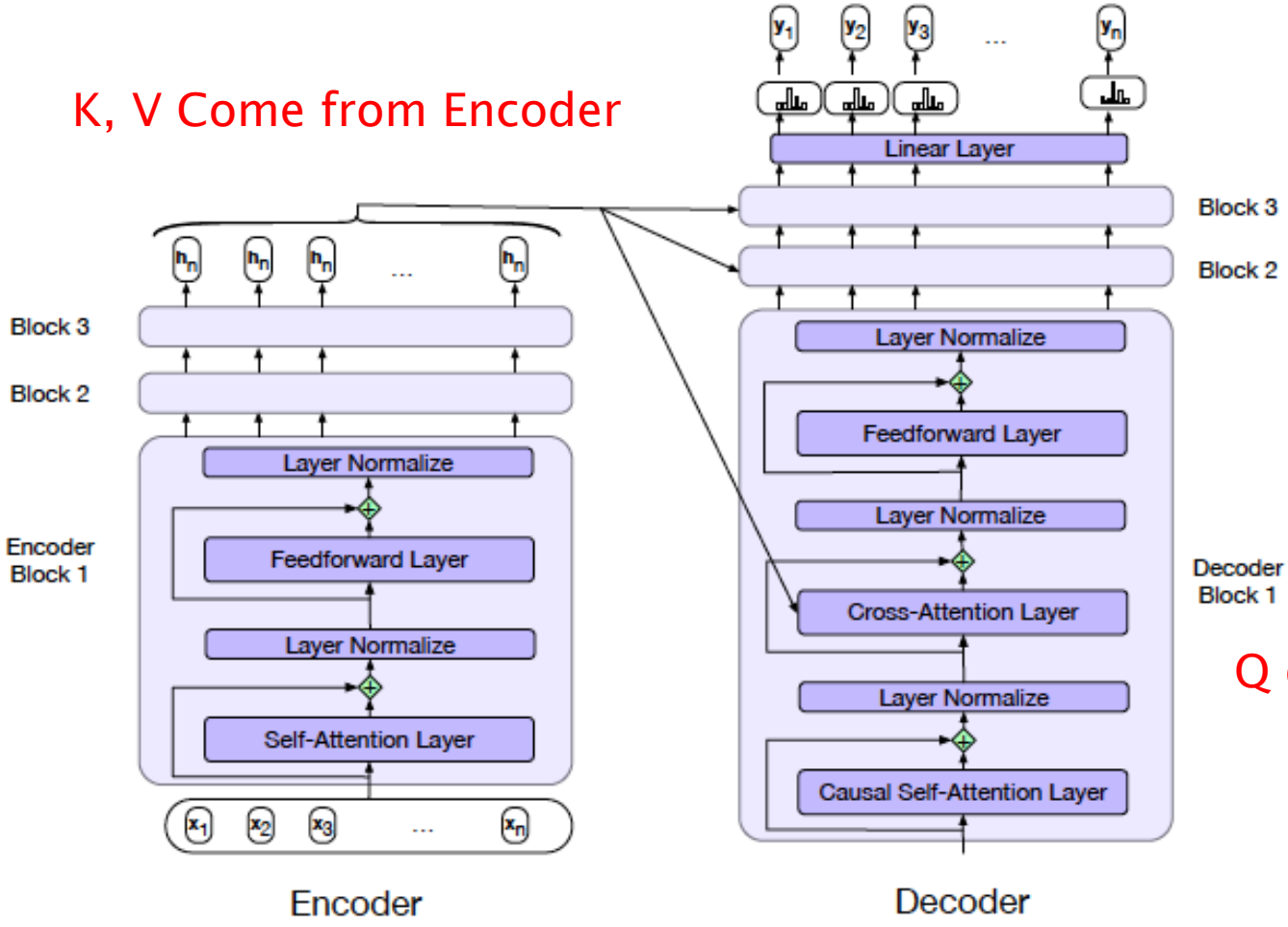
The encoding component is a stack of encoders. The decoding component is a stack of decoders of the same number.

Transformers – Decoders

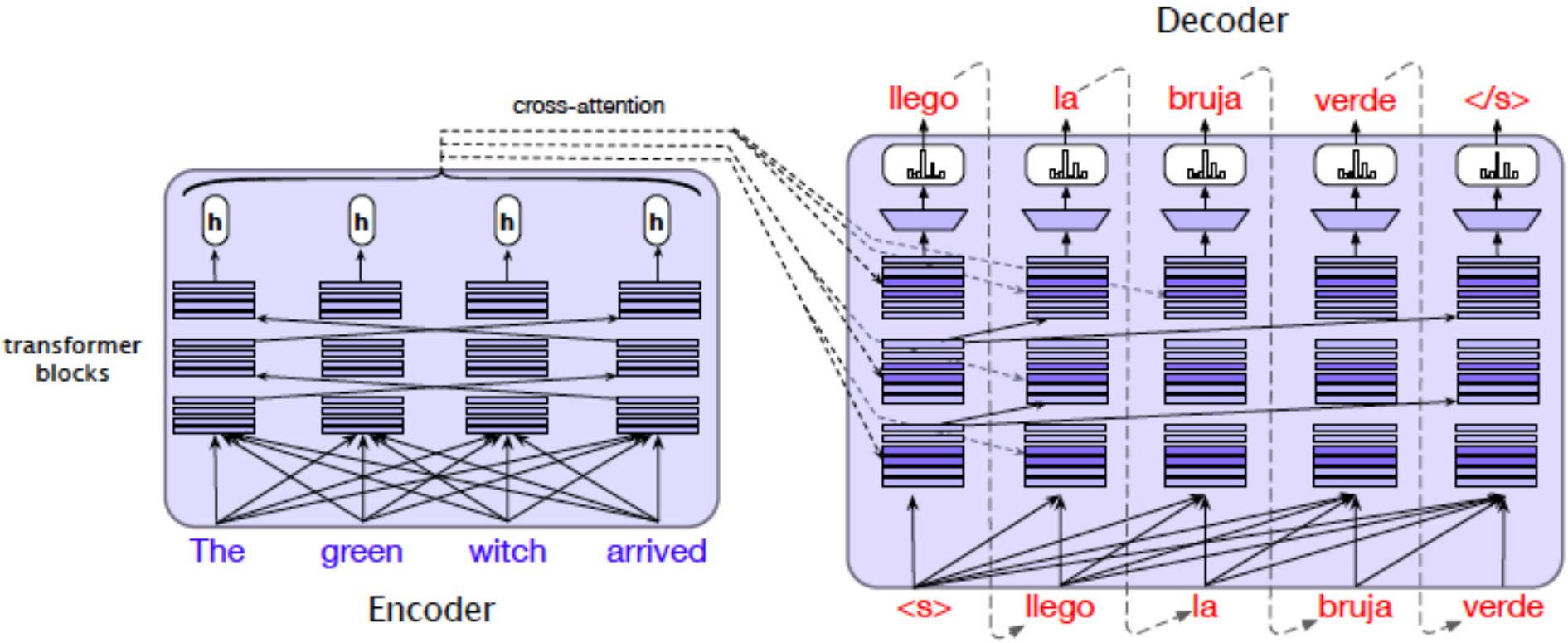


The decoder has an extra attention layer that helps it focus on relevant parts of the input sentence

Cross Attention



Encoder Decoder using Transformers



- The Queries come from the decoder layer
- The Keys and Values come from the output of the encoder.
  This allows every position in the decoder to attend over all positions in the input sequence

BERT

Bi-Directional Encoder Representations from Transformers (2019)

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

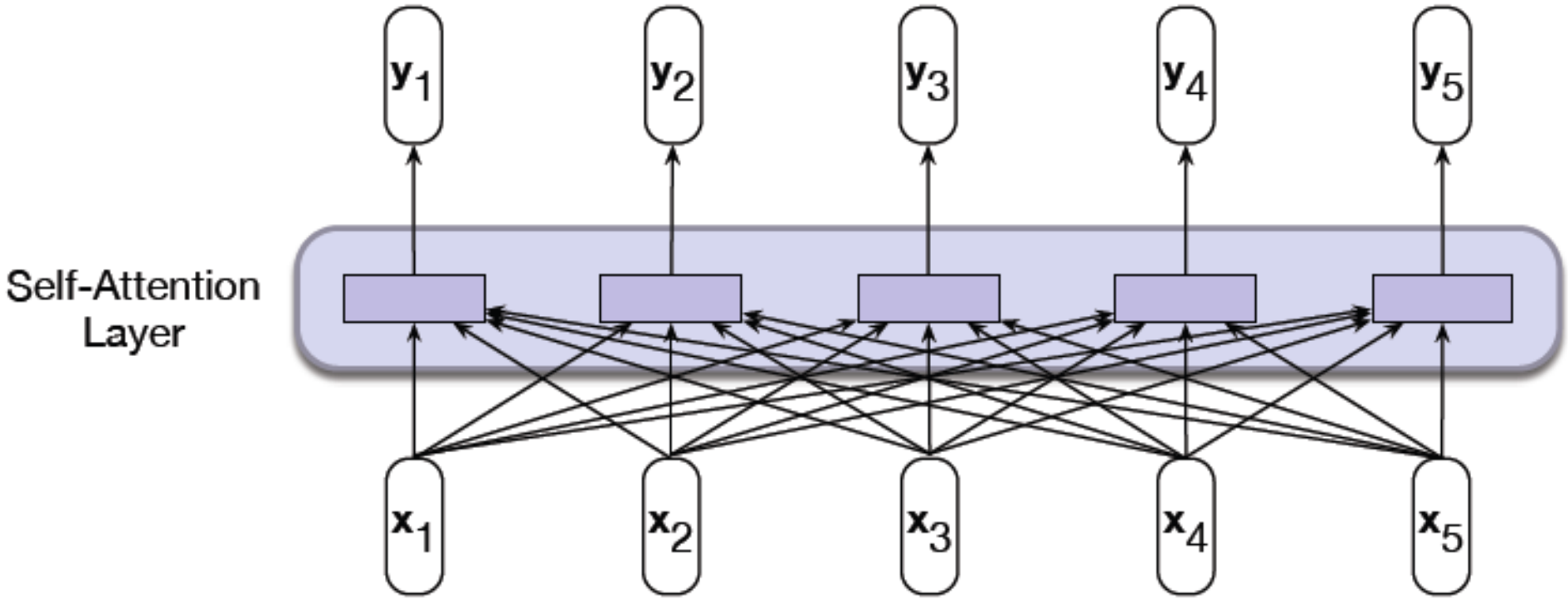
We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

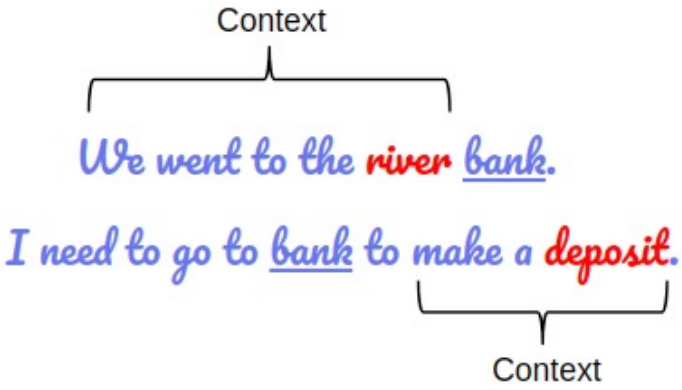
We argue that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token can only attend to previous tokens in the self-attention layers

BERT: Bi-Directional Language Models



BERT

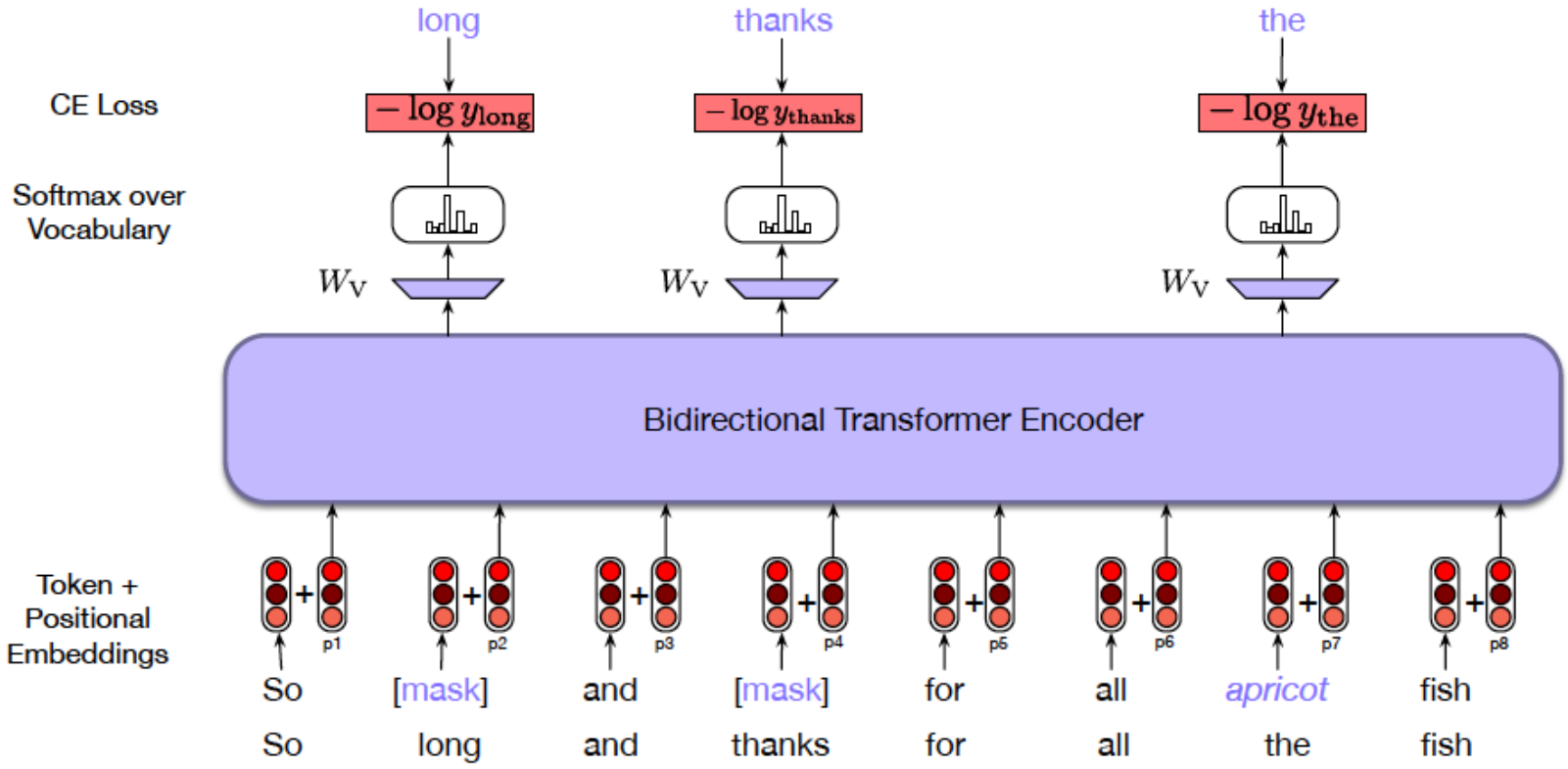
https://arxiv.org/pdf/1810.04805.pdf



Cannot use “Predict the Next Word” anymore

- The openAI transformer only trains a forward language model. Could we build a transformer-based model whose language model looks both forward and backwards
- Solution: BERT alleviates the unidirectionality constraint by using a “masked language model” (MLM) pre-training objective

BERT Training: Masked Language Model



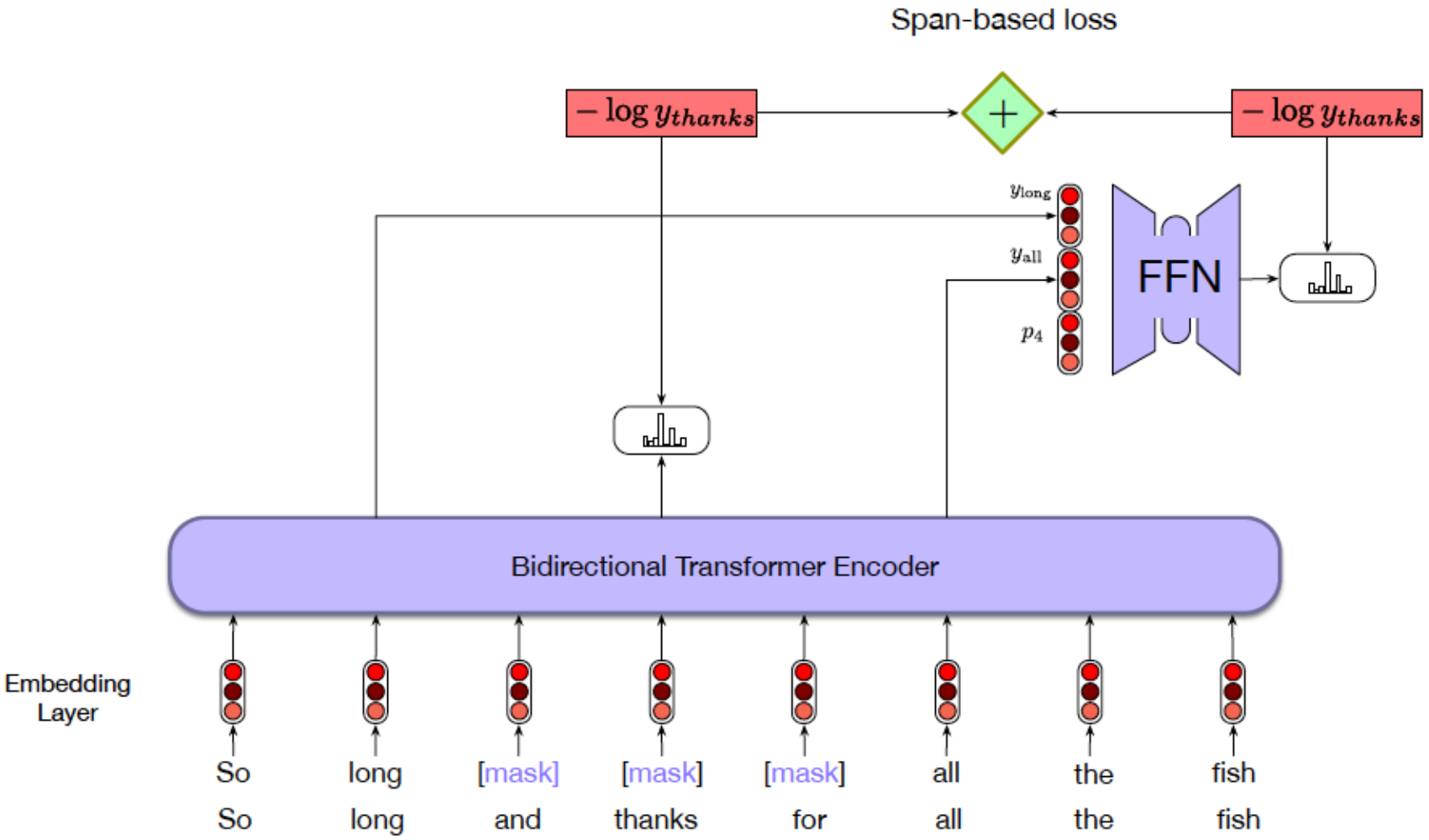
Up to 15% of the words in a sequence are randomly selected for prediction. Out of these, 80% of the words are replaced by a special MASK token, 10% of the words are left unchanged and the remaining 10% are replaced by a randomly selected word.

BERT Training: Span Based Loss

The span based training works as follows:

- The length of the span is chosen randomly by sampling from a geometric distribution, and is limited to 10 words or less.
- The start of the span is randomly selected using a uniform distribution. The Loss Function used to predict a word occuring within the span is the sum of two loss functions:
- The first Loss Function is simply the Cross Entropy Loss associated with the word being predicted
- The second Loss Function is computed using the word immediately preceding the span AND the word immediately following the span, augmented with a position token for the location of the missing word within the span.

BERT Training: Span based Loss

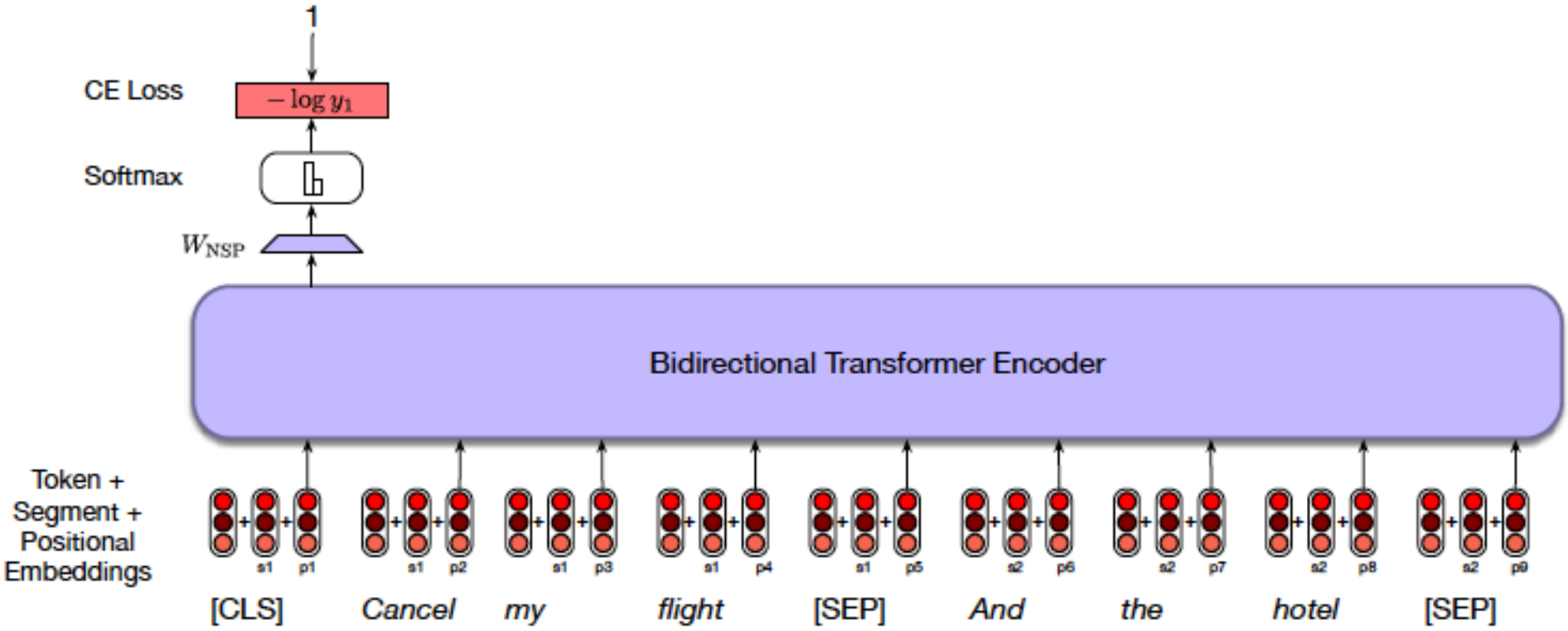


BERT Pre-Training: Next Sentence Prediction

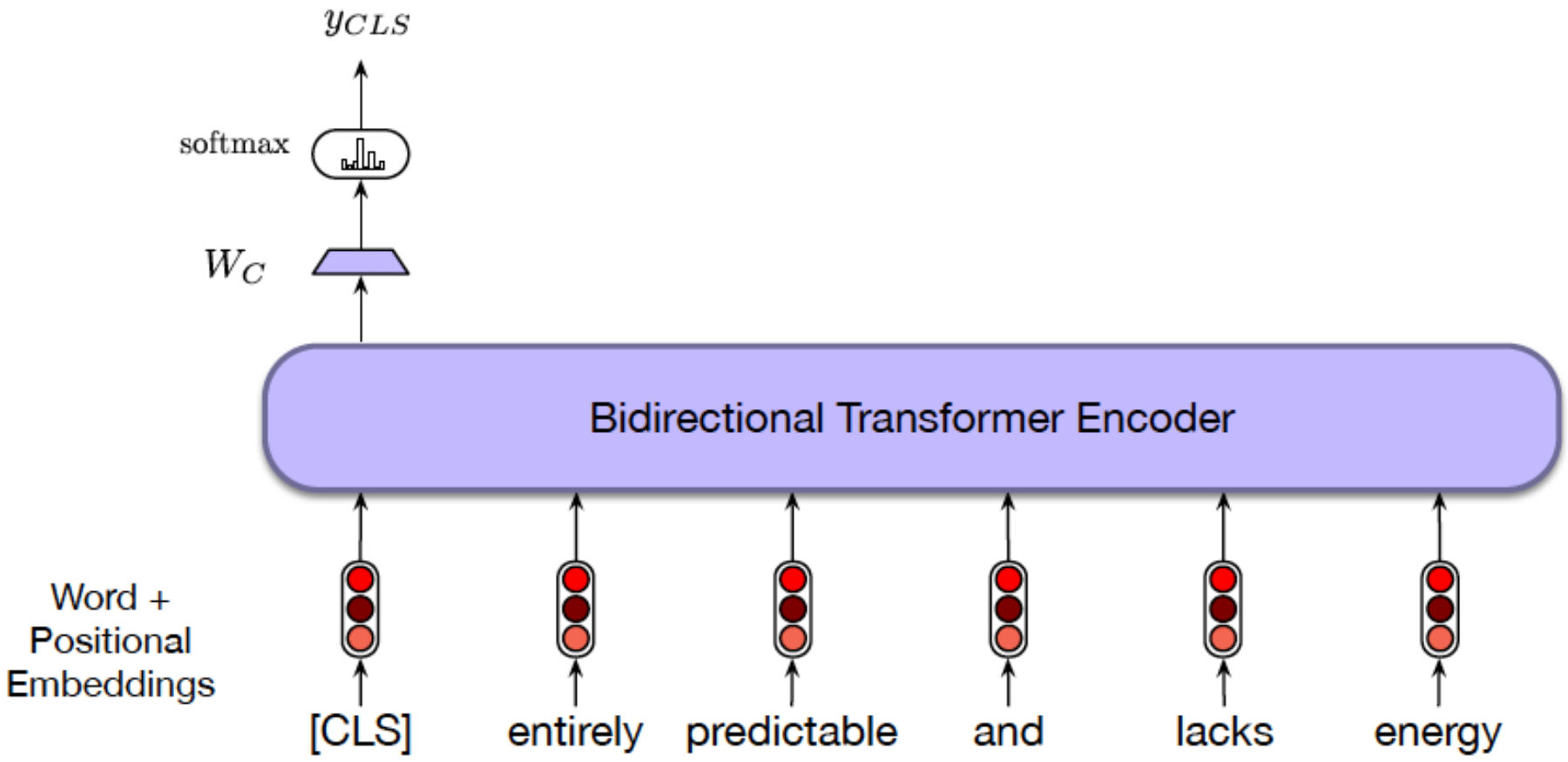
Consider that we have a text dataset of 100,000 sentences, so there will be 50,000 training examples or pairs of sentences as the training data.

- For 50% of the pairs, the second sentence would actually be the next sentence to the first sentence
- For the remaining 50% of the pairs, the second sentence would be a random sentence from the corpus
- The labels for the first case would be *'IsNext'* and *'NotNext'* for the second case

Next Sentence Prediction

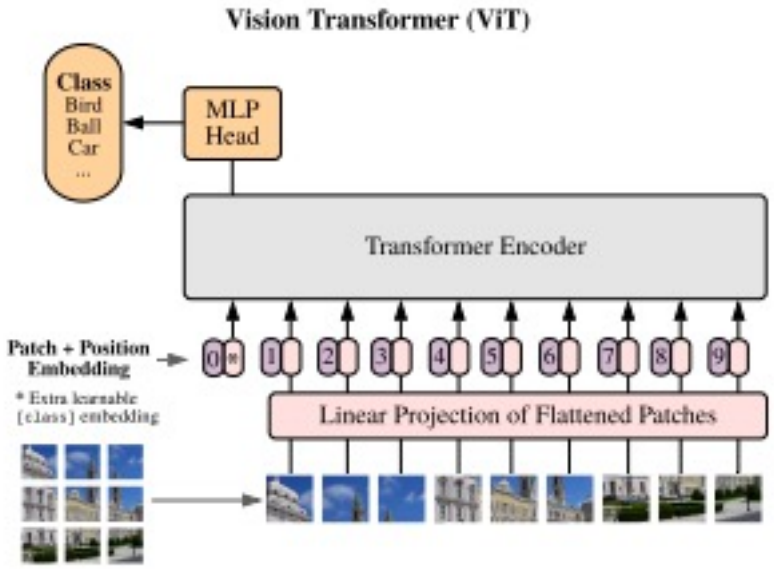


Classification Using a Pre-Trained BERT Model



Vision Transformers

Vision Transformers



The main idea behind ViT is quite simple and illustrated in Figure **trans18**. Given an input image X of shape $R^{H\times W\times C}$, where C is the number of channels and H and W are the dimensions of the image in pixels, sub-divide it into a sequence of flattened 2D patch vectors, which is of shape $R^{N\times P^2C}$. Each of the patch vectors is obtained by dividing the original image into image patches of size $P\times P\times C$ as shown in the figure, so that there are $N=\frac{HW}{P^2}$ image patches in all. Each image patch is then flattened to create N patch vectors of size P^2C. These patch vectors are then sent through a learnable embedding layer, and a position embedding is added to them, to create the input into the model. The Transformer model itself is exactly the same as was used for NLP.

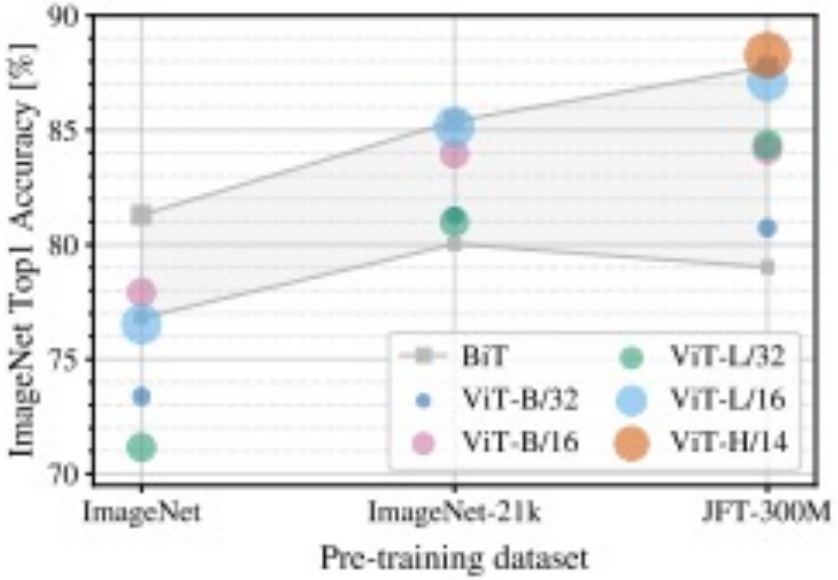
AN IMAGE IS WORTH 16X16 WORDS:
TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE https://arxiv.org/abs/2010.11929

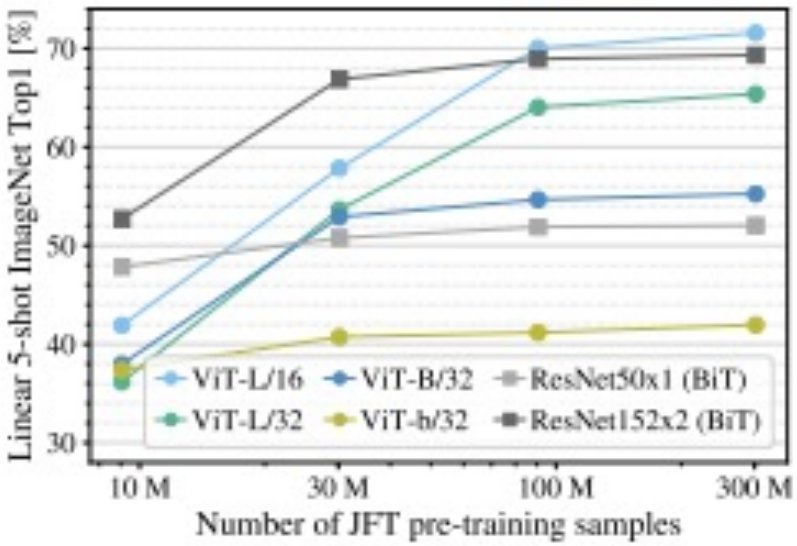
ViT Models

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

In order to test this new model, the researchers created three versions of ViT, as shown in Figure trans19, with increasing size. We use the notation ViT-L/16 to refer to the "Large" variant, with 16×16 input patch size. Note that decreasing the patch size increases the sequence length into the model, thus making it more computationally intensive.

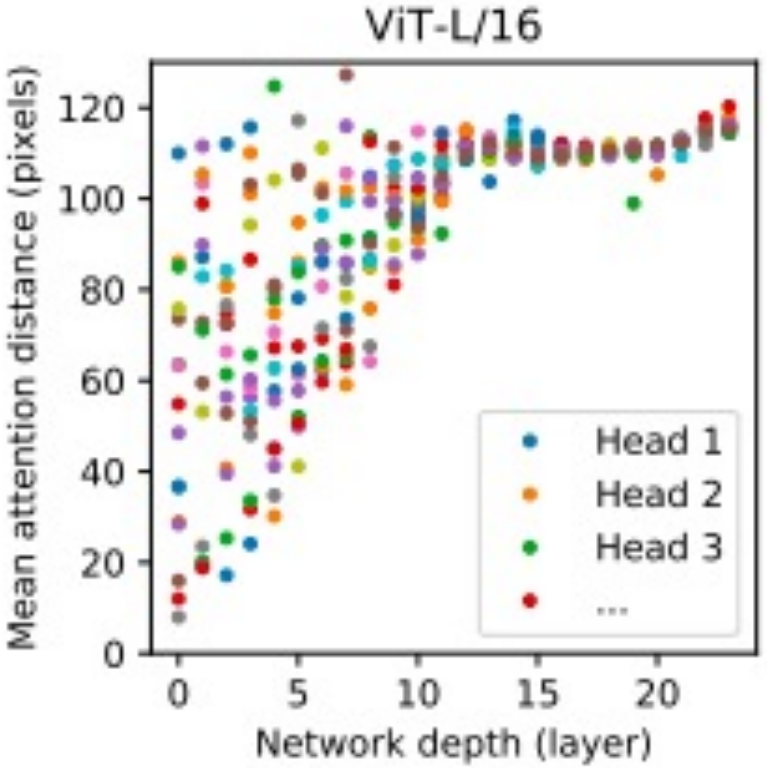
ViT Performance





The most interesting observation from this graph is that the performance of the ViT models varies very strongly as a function of the training dataset size. Indeed with smallest dataset (ImageNet), the ResNet models outperform all the ViT models. With the intermediate size dataset ImageNet-21k, their performances are about the same, while with the largest dataset, the best ViT model performs better than all the ResNet models. From this we can conclude that the strong inductive prior built into ResNet models, that feature representations are only influenced by nearby features in the neighborhood, works well when the training set is not very large. However for large training sets such as the JFT-300M, learning the relevant attention patterns from the data works as well or better.

Mean Attention Distance



The plot illustrates that in the early layers, the query patch is already paying attention to patches that are far it, indeed it seems to be paying attention across a broad spectrum of all the patches in the sequence. In later layers on the other hand, the Attention seems to focused more on patches that are further away. This behavior is in contrast to that in ConvNets, in which the convulation in the early layers is influnced solely by pixels that are in the immediate neighborhood.

Further Reading

- Das and Varma: Chapter Transformers
- Chollet (2nd Edition): Chapter 11, Section11.4