Fine-Tuning and Masked Language Models

Bidirectional Transformer Encoders

Bidirectional Transformer Encoder

We learned about causal language modeling

- Make autoregressive word prediction
- *Left-to-right transformers*

Now we introduce Bidirectional Transformer Encoder

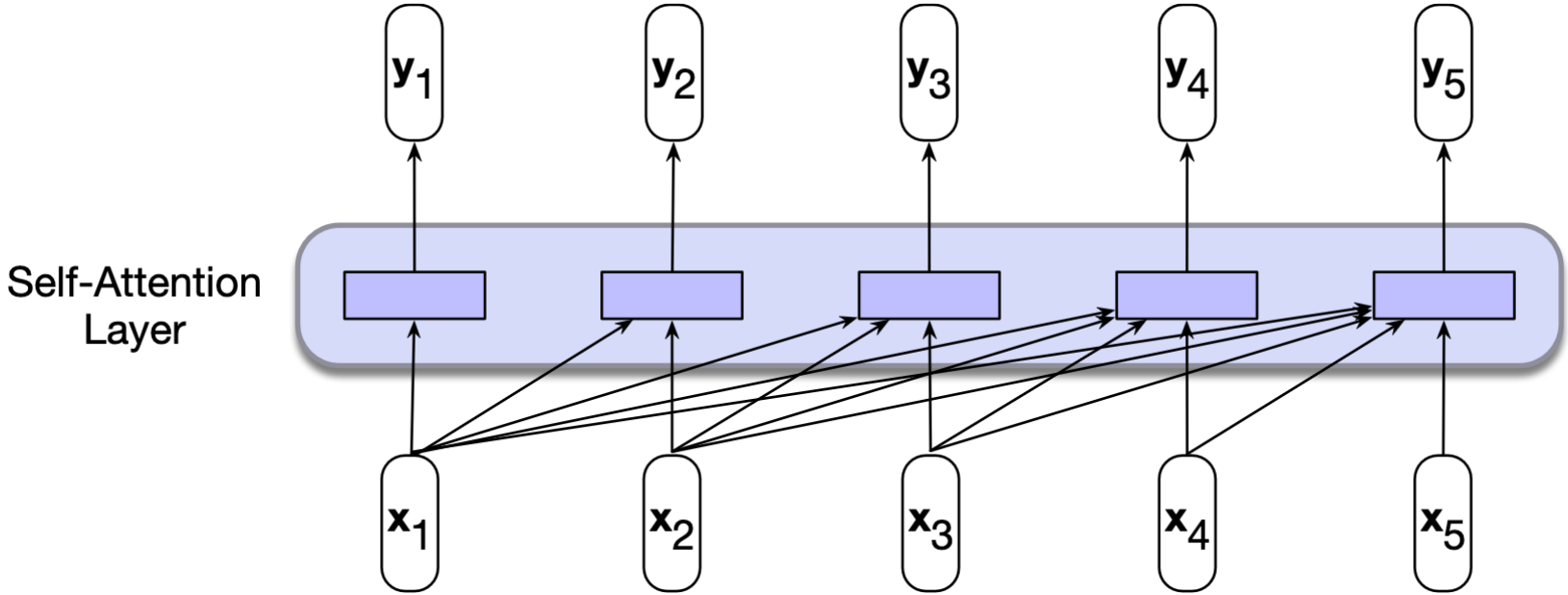
- Sees both left and right context
- Trained with *masked language modeling*

Bidirectional Transformer Encoder

BERT, RoBERTa, SpanBERT, XLM …

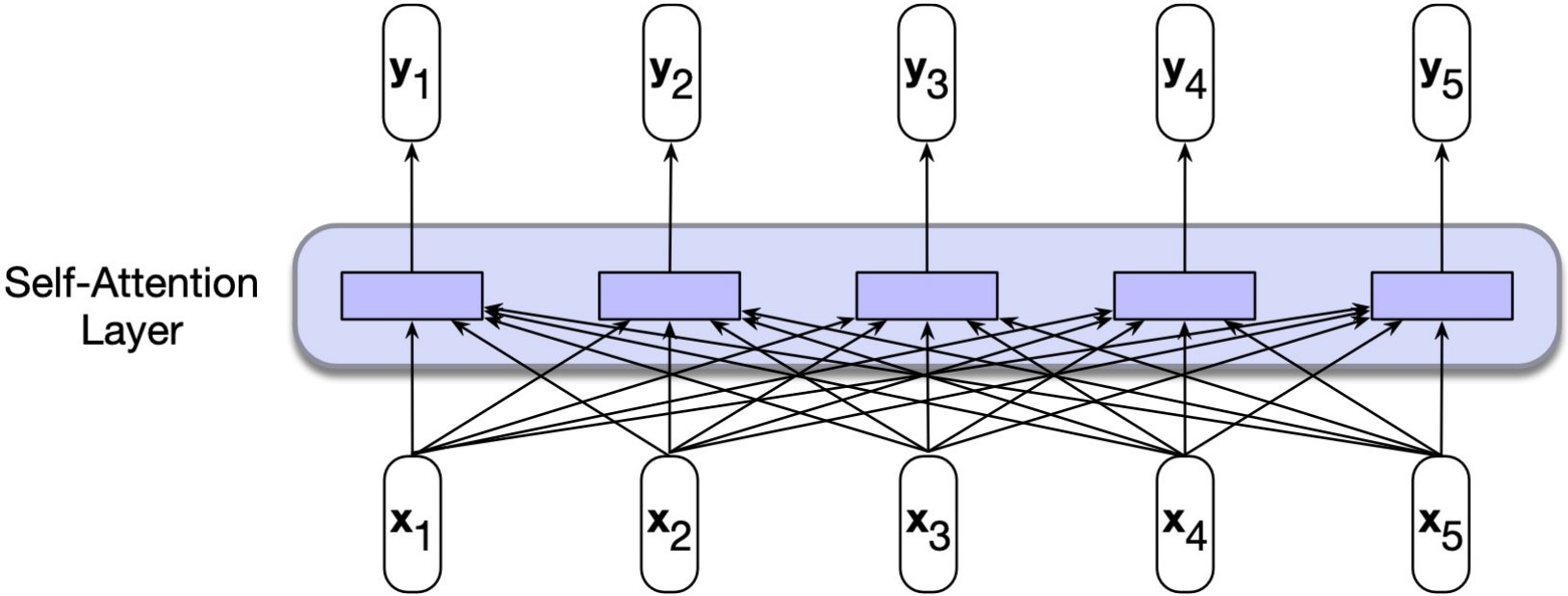
- They are used in downstream applications via *fine-tuning*
  - Pre-trained LM are used as input for a top layer network that is trained for the task
  - Pretraining/fine-tuning is a kind of *transfer learning*
- Embeddings generated by Masked Language models (MLM) are *contextual*

Bidirectional Transformer Encoder



Left-to-right transformer architecture

Bidirectional Transformer Encoder



Bidirectional transformer encoder architecture

Bidirectional Transformer Encoder

Classical self-attention mechanism

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i; \;\; \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i; \;\; \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

$$\mathbf{y}_i = \sum_{j=1}^{n} \alpha_{ij}\mathbf{v}_j \qquad \alpha_{ij} = \frac{\exp(\text{score}_{ij})}{\sum_{k=1}^{n}\exp(\text{score}_{ik})}$$

$$\text{score}_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$$

Bidirectional Transformer Encoder

Parallel implementation using matrix products

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}} \quad \mathbf{Q} \in \mathbb{R}^{N \times d_k}\ \mathbf{K} \in \mathbb{R}^{N \times d_k}\ \mathbf{V} \in \mathbb{R}^{N \times d_v}$$

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

The product $\mathbf{Q}\mathbf{K}^{\mathsf{T}}$ *computes also the scores*
for the keys that follow the query

q1•k1	q1•k2	q1•k3	q1•k4	q1•k5
q2•k1	q2•k2	q2•k3	q2•k4	q2•k5
q3•k1	q3•k2	q3•k3	q3•k4	q3•k5
q4•k1	q4•k2	q4•k3	q4•k4	q4•k5
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

N

Bidirectional Encoder Representations from Transformers (BERT)

Subword tokenization

- Word-Piece algorithm (includes BPE)
- 30,000 subwords vocabulary
- 512 token+positional embeddings fixed input to control computational complexity
  - Memory requirement grows quadratically with the input size

Bidirectional Encoder Representations from Transformers (BERT)

Multi-layer transformer blocks

- 768 units in heach hidden layer
- $\text{BERT}_{\text{BASE}}$: 12 transformer layers, 12 self-attention heads per layer, 110M params
- $\text{BERT}_{\text{LARGE}}$: 24 transformer layers, 16 self-attention heads per layer, 304M params

Fine-Tuning and Masked Language Models

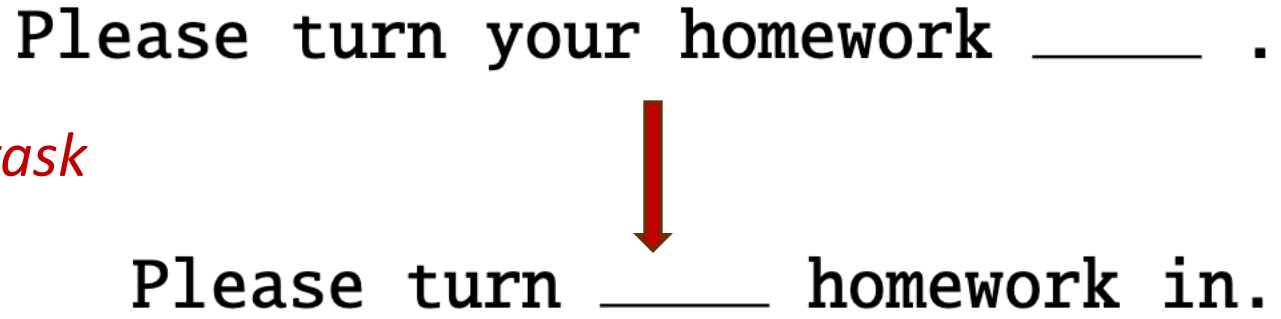
Training Bidirectional Encoders

Masked Language Modeling

Causal language modeling becomes trivial when we remove the causal mask

Masked Language Modeling uses a «fill-in-the-blank» approach

- The *cloze task*



Masked Language Modeling

During training the model is deprived of one or more elements of an input sequence

It must generate a probability distribution over the vocabulary for each of the missing items

We then use the cross-entropy loss from each of the model's predictions to drive the learning process

Masked Language Modeling

Masking can be generalized to any method that corrupt the training input and then asks the model to recover the original input.

- Masks
- Substitutions
- Reorderings
- Deletions
- Extraneous insertions into the training text

Masking words

BERT makes use of unannotated large text corpora

A random sample of tokens from each training sequence is selected for use in the learning task

Masking words

Once selected, a token is used in one of the following three ways:

- It is replaced with the unique vocabulary token `[MASK]`
- It is replaced with another token from the vocabulary, randomly sampled based on token unigram probabilities
- It is left unchanged

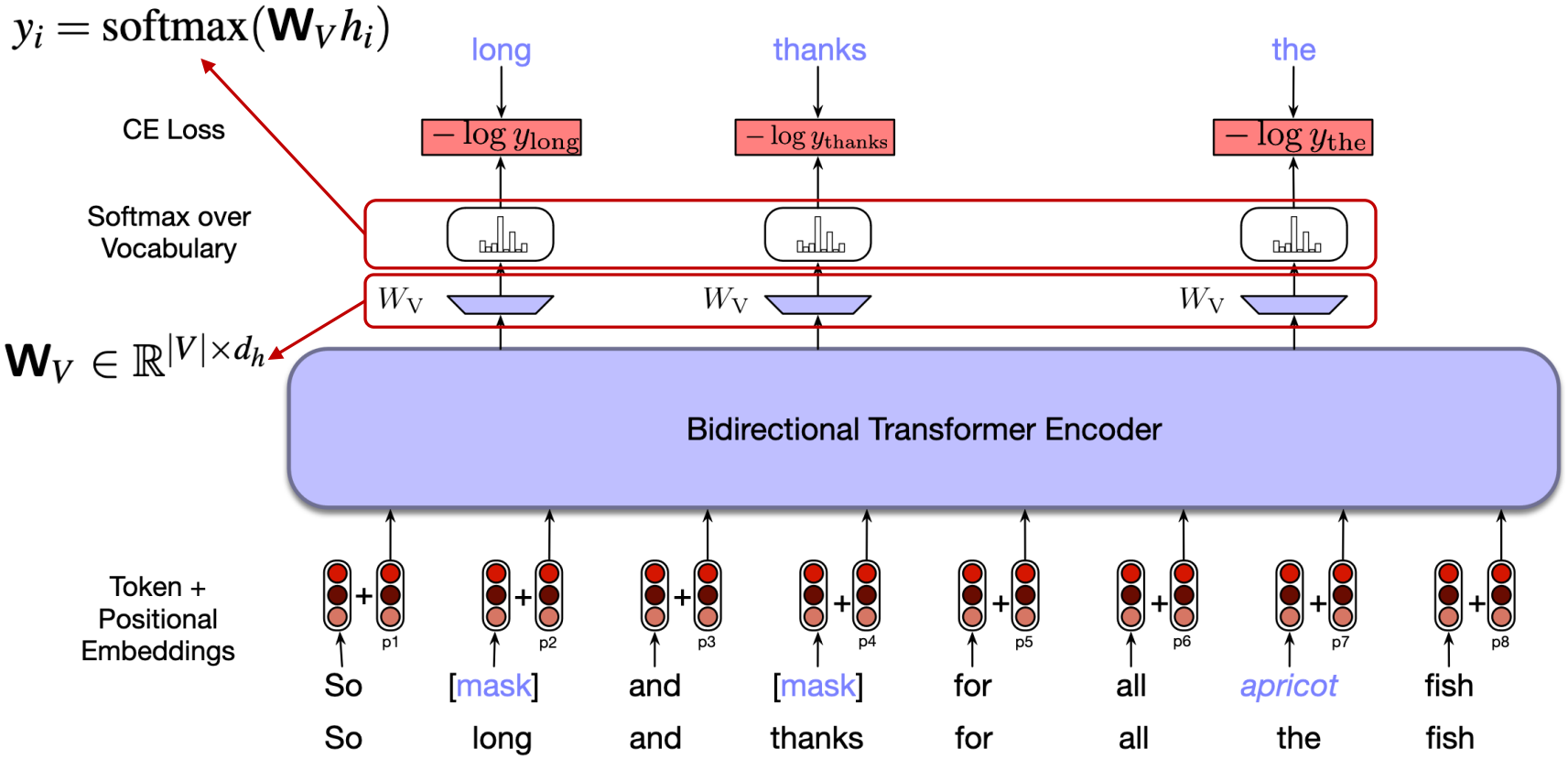
Masking words

In BERT 15% tokens are sampled for learning

- 80% is substitued by the `[MASK]` token
- 10% is replaced with a random token
- 10% is left unchanged

All the tokens are involved in self-attention, but only the selected ones are used in computing the loss through the softmax

Masking words



Masking spans

Many NLP tasks are oriented to phrases intead of single words

Span-orielted MLM is a better option in this respect

- *Span*: a contiguous sequence of one or more words selected from a training text, *prior to subword tokenization*

Masking spans

SpanBERT

- Same percentage as BERT of sampled tokens for training (15%)
- A span length is chosen from a geometric distribution biased towards short spans, and with an upper bound of 10

Masking spans

SpanBERT

- Given the span length, a starting position is sampled uniformly, that is consistent with the input length
- Elements in a span are substituted using the same 80%, 10%, 10% scheme as in BERT

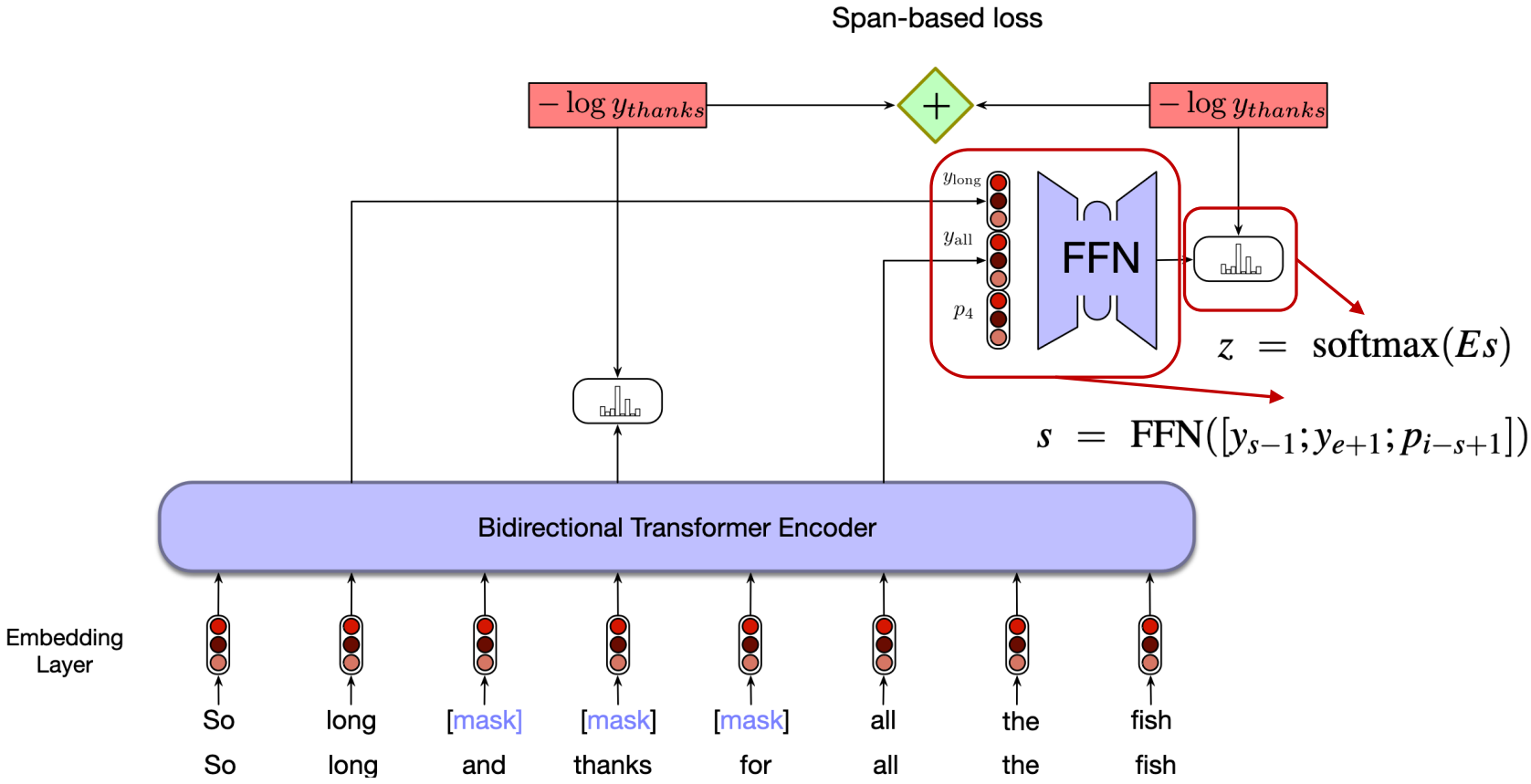
Masking spans

The loss function is modified to take into account the so called *Span Boundary Objective* (SBO)

$$L(x) = L_{MLM}(x) + L_{SBO}(x)$$
$$L_{SBO}(x) = -\log P\left(x \mid x_{s-1}, x_{e+1}, p_x\right)$$

L_{SBO} is learned through a 2-layer FFN

Masking spans



Next sentence prediction

Many NLP tasks involve determining the relation between pairs of sentences

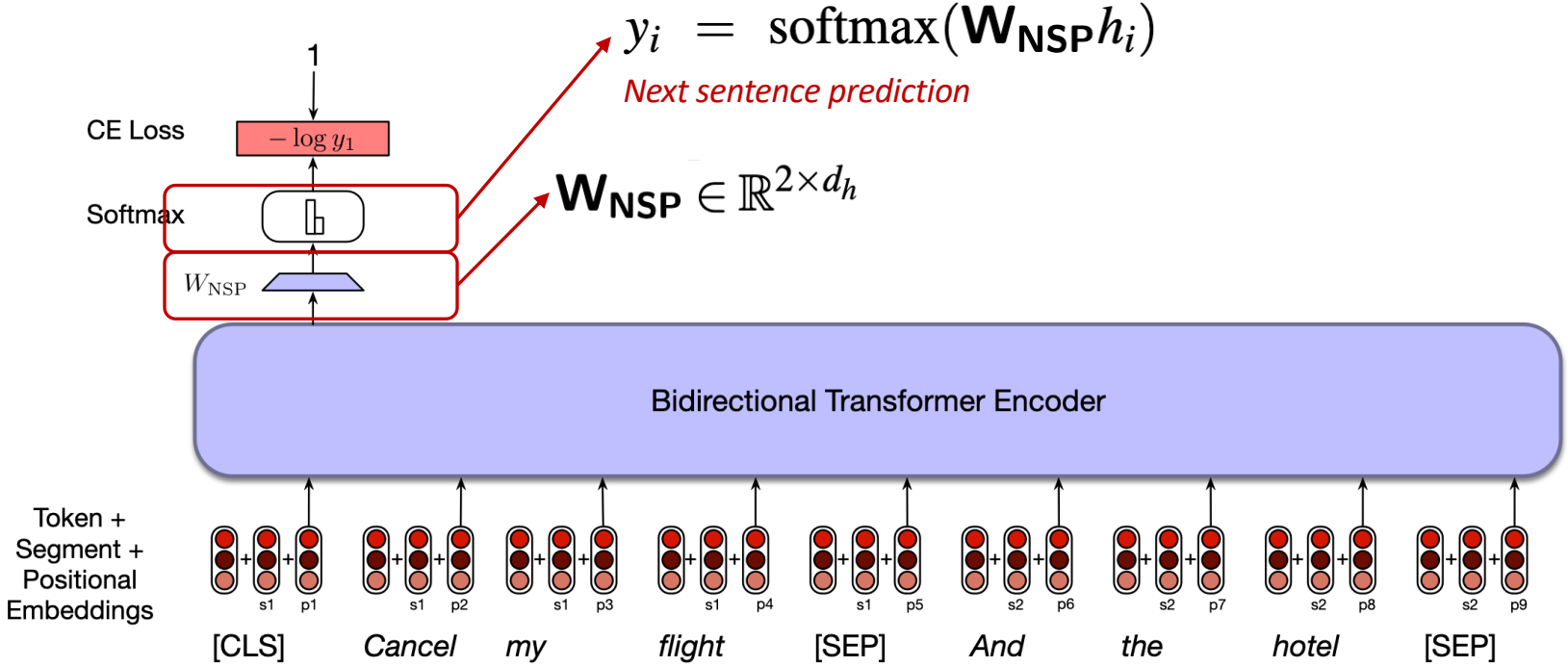
- Paraphrase detection
- Textual entailment
- Discourse coherence
- …

Next sentence prediction

BERT achieves Next Sentence Prediction (NSP) using the following training procedures

- 50% split between positive (related) and negative(unrelated) sentence pairs
- Two new tokens are introduced
  - `[CLS]` is prepended to the input sentence pair
  - `[SEP]` is placed between the sentences and after the final token

Next sentence prediction



BERT training

Trained on

- BookCorpus (800M words)
- English Wikipedia (2500M words)

NSP using the 50%/50% scheme

- Sentence pairs sampled to fit in the 512 token input size

40 epochs training using MLM plus NSP loss

Contextual embeddings

The output y_i provided by a pretrained LM for each token x_i of a novel sentence can be regarded as a *contextual embedding*

- Vectors representing some aspect of the meaning of a *token* in context
- Often, the c.e. is obtained by averaging the output of the last four layers of the model

- Static embeddings → meaning of *types*
- Contextual embeddings → meaning of *tokens*

Fine-Tuning and Masked Language Models

Transfer Learning through Fine-Tuning

Fine-tuning

Pretrained LMs are powerful at generalizing

- Useful for downstream applications

Fine-tuning creates an *interface* between the LM and the application (the application-specific model) to allow generalization

- Train the new parameters using labeled application data
- Freeze or make slight adjustments to the LM parameters

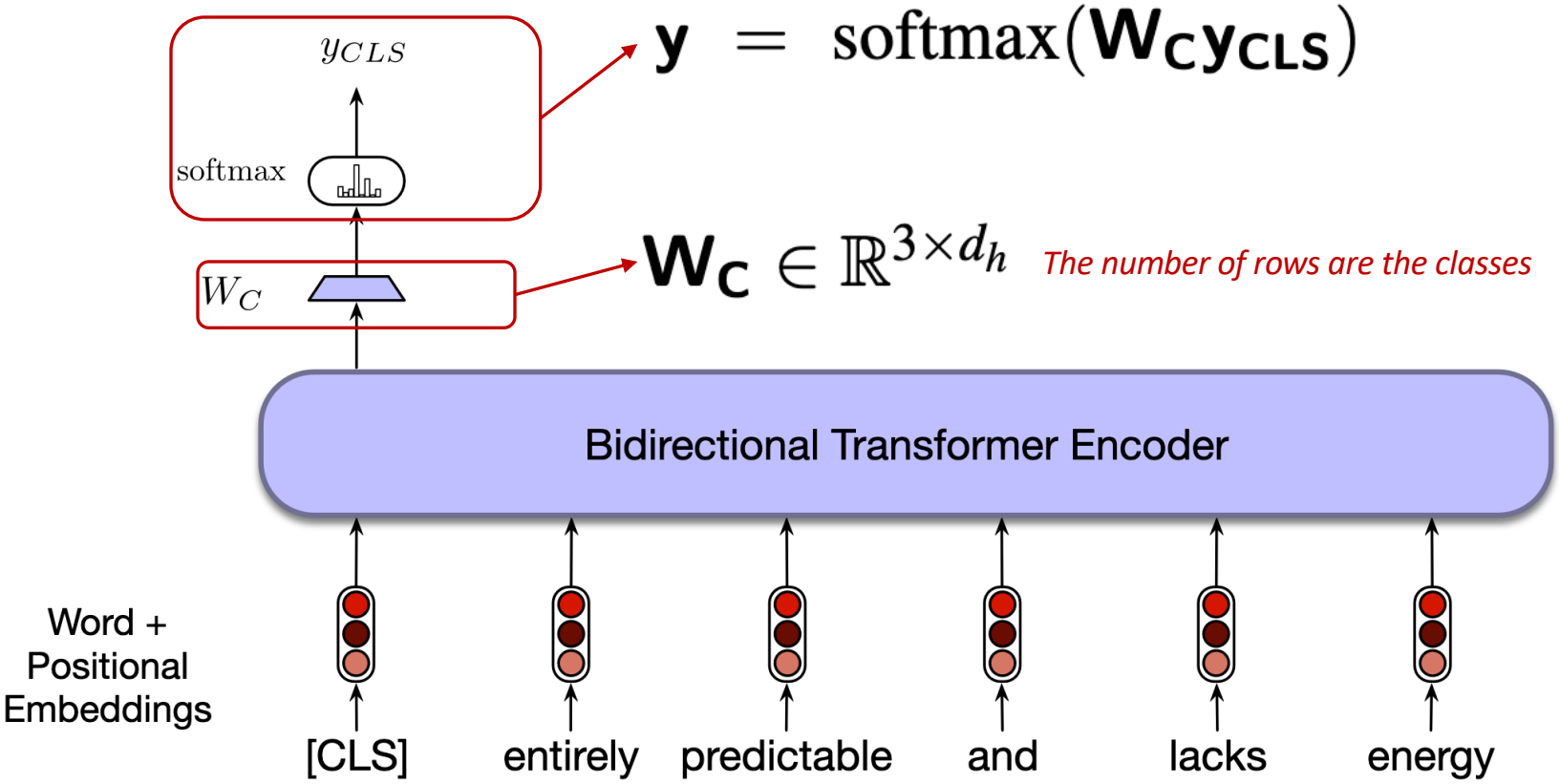
Sequence Classification

In sequence classification (i.e. sentiment analysis) a *sentence embedding* is often used to allow classification

In BERT the `[CLS]` token plays the role of sentence embedding

A dense layer $\mathbf{W}_c$ is fine-tuned with labeled data and cross-entropy loss

Sequence Classification



Pair-Wise Sequence Classification

Recognizing the relation between pairs of sentences: textual entailment, dicourse coherence …

Fine-tuning proceeds the same as NSP training

- Sentence pairs are presented to the model
- The `[CLS]` token head is trained with class labels

Pair-Wise Sequence Classification

Example: textual entailment with the MultiNLI corpus

- `Neutral`
  - a: Jon walked back to the town to the smithy.
  - b: Jon traveled back to his hometown.
- `Contradicts`
  - a: Tourist Information offices can be very helpful.
  - b: Tourist Information offices are never of any help.
- `Entails`
  - a: I'm confused.
  - b: Not all of it is very clear to me.

Sequence Labelling

Example: BIO-based named entity recognition

A dense layer $\mathbf{W_K}$ is trained on the output provided separately for each input token

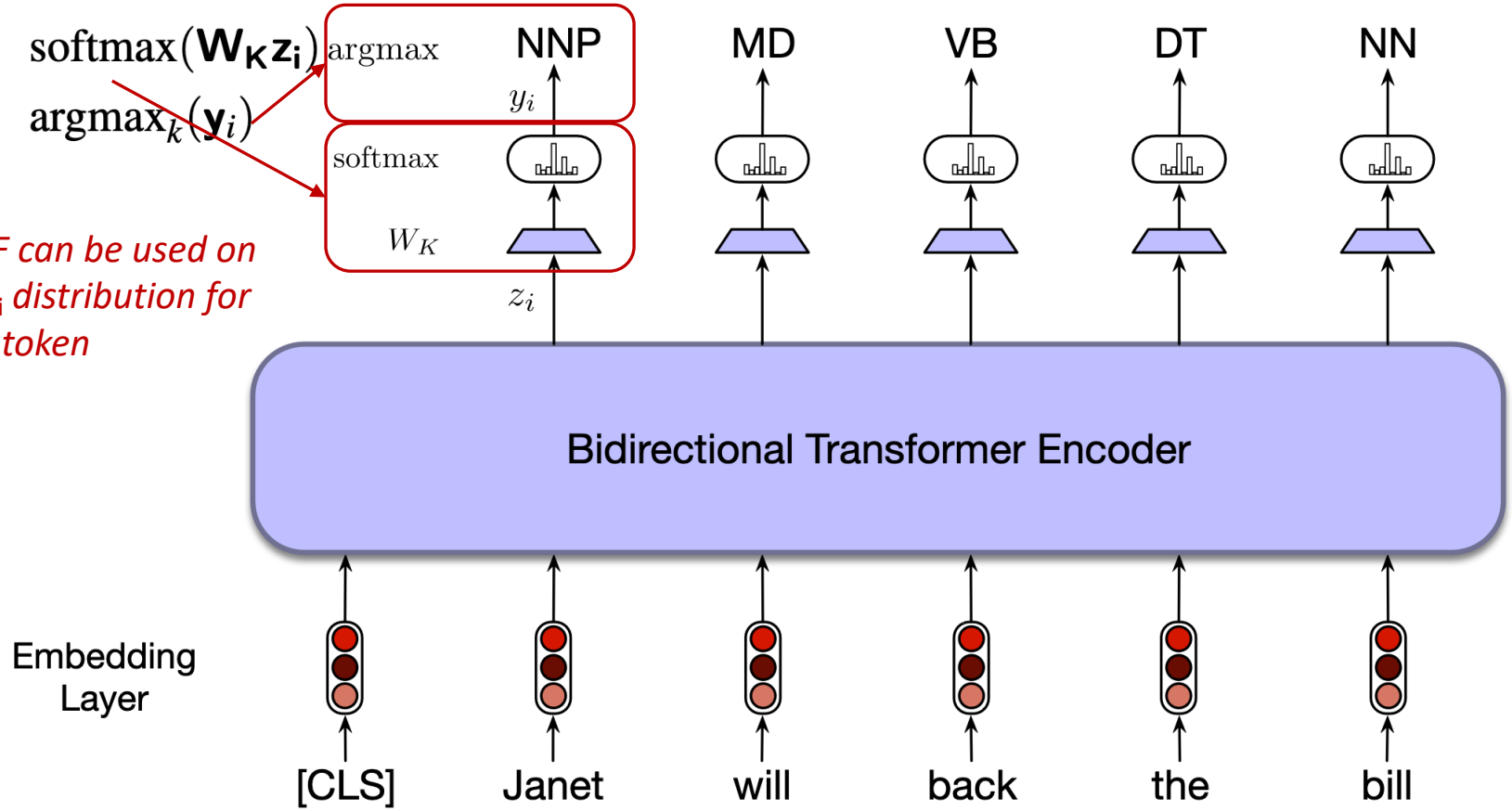
- $\mathbf{W_K}$ has *k* rows, one for each tag
- The *argmax* is used to select the output tag

Sequence Labelling

$$\mathbf{y_i} = \text{softmax}(\mathbf{W_K z_i})$$

$$\mathbf{t_i} = \text{argmax}_k(\mathbf{y}_i)$$

A CRF can be used on the $\mathbf{y_i}$ distribution for each token



Sequence Labelling

A problem: sub-word tokenization

Example:

[LOC **Mt. Sanitas**] is in [LOC **Sunshine Canyon**] .

Mt. Sanitas is in Sunshine Canyon .

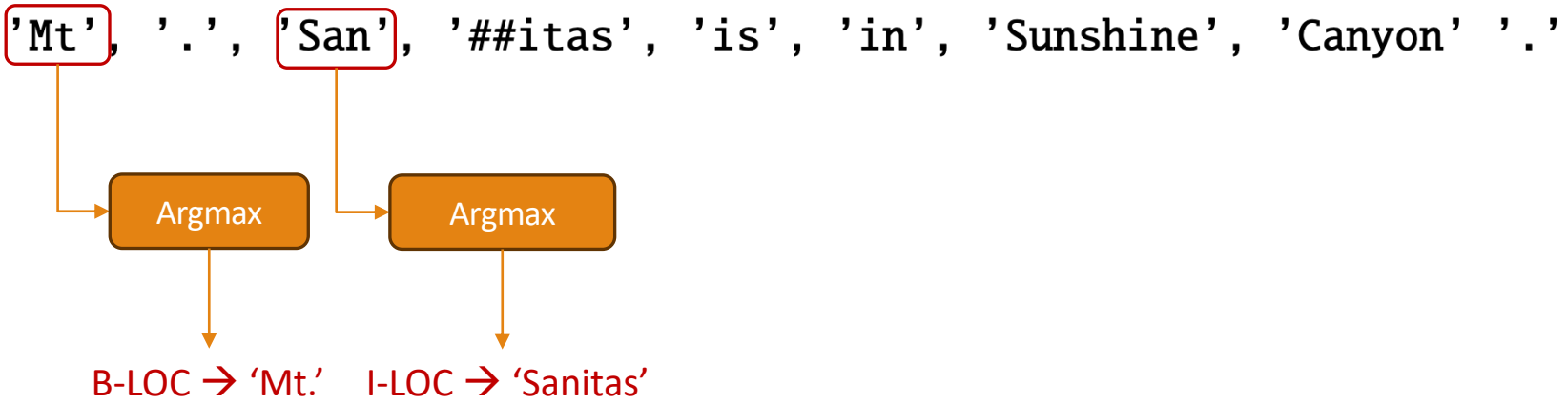
B-LOC I-LOC O O B-LOC I-LOC O

```
'Mt', '.', 'San', '##itas', 'is', 'in', 'Sunshine', 'Canyon' '.'
```

B-LOC B-LOC I-LOC I-LOC O O B-LOC I-LOC O

Sequence Labelling

Decoding through *argmax* of the tag assigned to the first sub-word token



Fine-tuning for Span-Based Applications

Span-oriented applications operate in the middle between sequence and token level tasks.

- Identifying spans of interest
- Classifying spans
- Determining relations between spans

Fine-tuning for Span-Based Applications

A span from a sequence x made by T tokens, starts at position $i \in [1, T]$, and ends at position $j \in [1, T]$

- $T(T - 1)/2$ possible spans
- A maximum length L can be set
- $S(x)$ will be the set of legal spans

Fine-tuning for Span-Based Applications

Many approaches for fine-tuning

All of them use embeddings where both boundary and span representation are included

- Boundary: the couple $\langle \mathbf{h}_i, \mathbf{h}_j \rangle$ or a FFN can be trained
  - $s_i = \text{FFN}(\mathbf{h}_i)$
  - $e_j = \text{FFN}(\mathbf{h}_j)$

Fine-tuning for Span-Based Applications

Many approaches for fine-tuning

All of them use embeddings where both boundary and span representation are included

- Span: $\mathbf{g}_{ij} = \text{avg}(\mathbf{h}_i,\ldots,\mathbf{h}_j)$ or better *self-attention*
  - $\mathbf{g}_{ij} = \text{SelfAttention}(\mathbf{h}_{i:j})$

Fine-tuning for Span-Based Applications

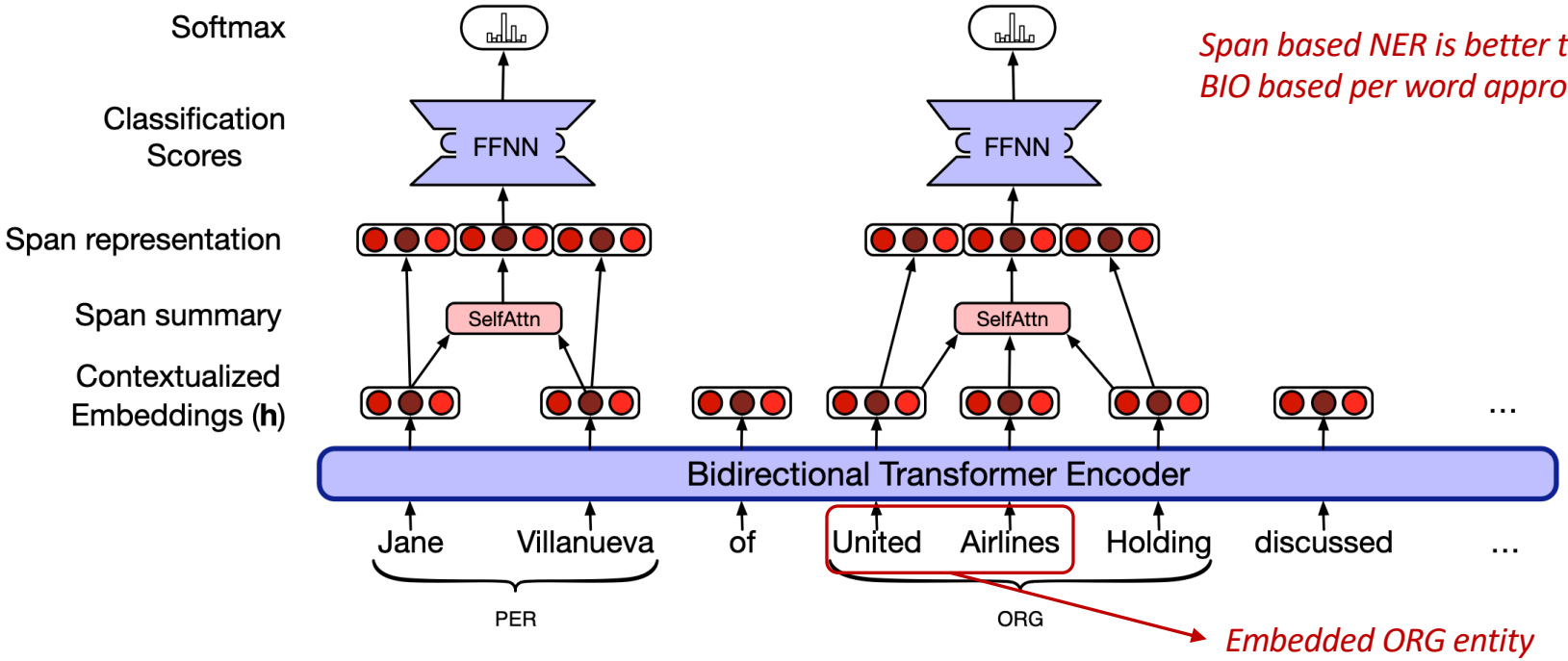
Many approaches for fine-tuning

All of them use embeddings where both boundary and span representation are included

- Embedding: $\mathbf{emb}_{ij} = \text{Concat}(\mathbf{h}_i, \mathbf{g}_{ij}, \mathbf{h}_j)$

Fine-tuning for Span-Based Applications

y_{ij} = softmax(FFN(**emb***ij*))



Fine-Tuning and Masked Language Models

BERT variants

Multilingual BERT

Trained on 104 languages at the same time to build a shared representation of the word meaning in each of them

110K token shared vocabulary for Word Piece

Not the best in cross-lingual tasks

RoBERTa

Developed by Facebook (now Meta)

Trained on five English corpora (160GB uncompressed text)

Effective pre-training with *dynamic masking*

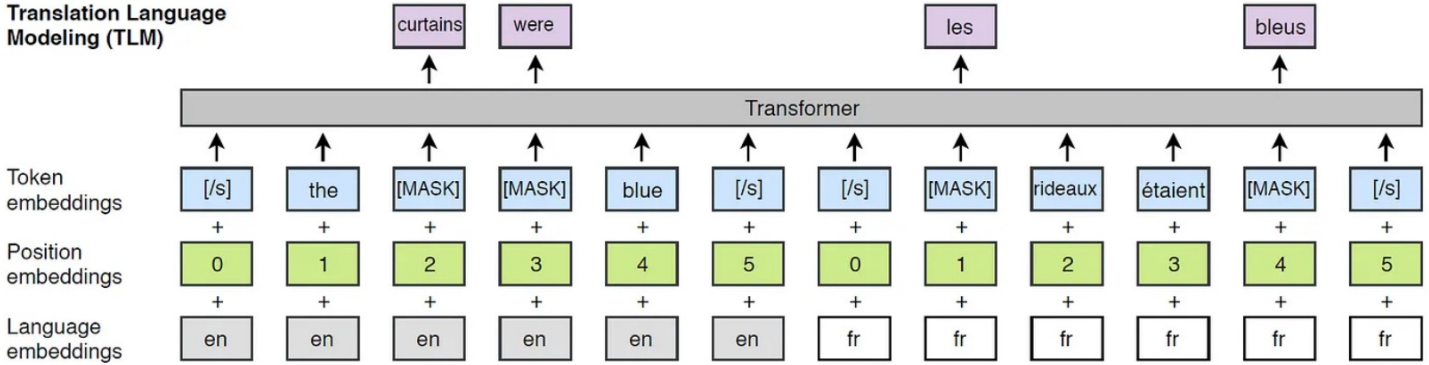
- Masking changes at each input sequence
- BERT makes the masking just once as preprocessing

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

XLM (croX lingual Language Model)

Developed by Facebook (pre-training on 64 GPUs)

Translation Language Modelling (TLM)



Lample, G., & Conneau, A. (2019). Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*.

XLM-RoBERTa

Developed by Facebook

Much more computational power (pre-training on 500 GPUs) and much more data

It doesn’t use language embedding to enable a fast language switchper

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., ... & Stoyanov, V. (2019). Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*.