Text Analysis for Social Sciences in Python

Week 14: Introduction to BERT

WINTER SEMESTER 21/22

DR. HUYEN NGUYEN

[HTTP://HUYENTTNGUYEN.COM](http://huyenttnguyen.com)

IMPORTANT! For your presentation & codes

Thursday 27.01 @ 23.59 CET: Submission of code and slides to Open Olat folder “**Codes & Presentation**”

AND fill in the team evaluation survey.

https://docs.google.com/forms/d/e/1FAIpQLSc9zv003qCbUHVrlpK-fnSwoy0iNFzxpqQpRYMQ2Ira7ODelg/viewform?usp=sf_link

Monday 31.01: Show up in the waiting room before your assigned time slots. The time slots and access Zoom links are here:

https://docs.google.com/spreadsheets/d/1E-lFBNDNk7-5GxALoYRfK-P8GO5aYJ8v9j0uEjUsZ5M/edit?usp=sharing

IMPORTANT! Have your student ID & personal IDs ready for legal identification purposes.

Today’s agenda

- BERT intuition: fine-tuned approach
- BERT data flow: inputs and outputs
- In practice: BERT & Transformer pipelines

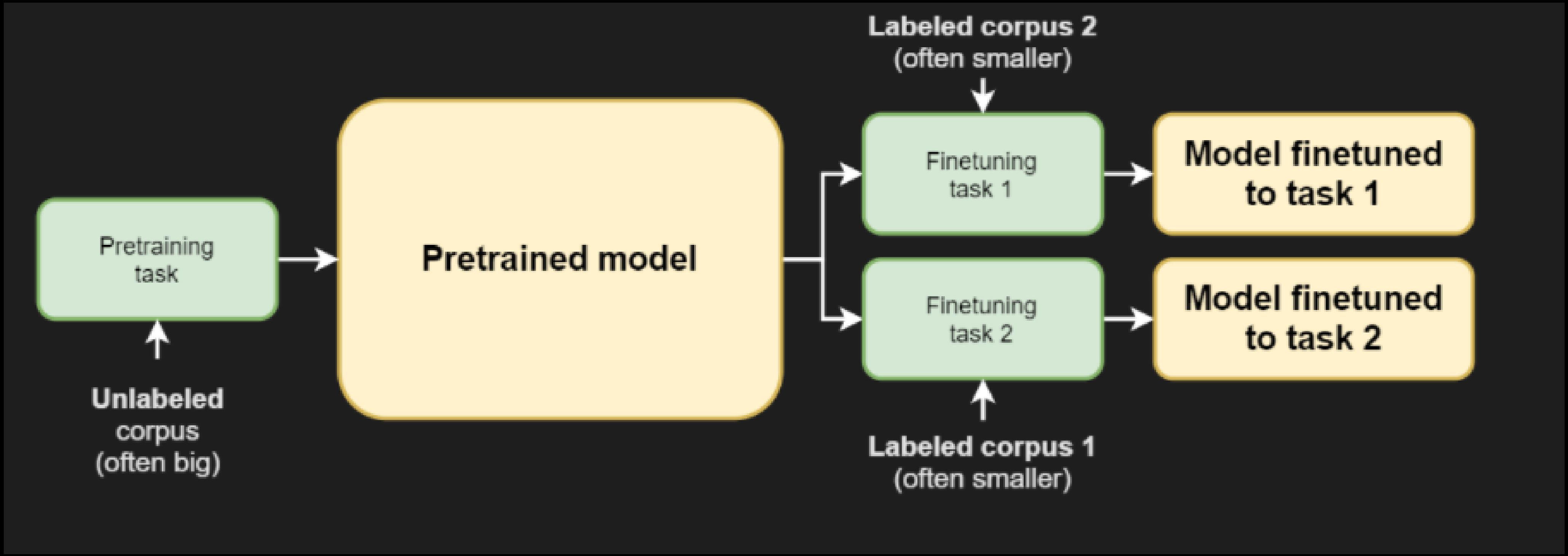
What is BERT?

BERT = **Bidirectional Encoder Representations from Transformers** (Devlin et al. 2018) follows the so-called **finetuning-based approach** in NLP.

BERT is widely used since its introduction (e.g. Google Search), and overcomes the issue of unidirectionality in previous Transformer approaches.

What is BERT? (cont)

What is fine-tuning?



What is BERT? (cont)

End result = a model that has been pretrained on the large unlabeled corpus AND finetuned to a specific language task, e.g. summarization, text generation in a particular domain, or translation.

Fine-tuned approach = train using the same model all the time AND allow us to use unlabeled data sets *(semi-supervised)*

What is BERT? (cont)

End result = a model that has been pretrained on the large unlabeled corpus AND finetuned to a specific language task, e.g. summarization, text generation in a particular domain, or translation.

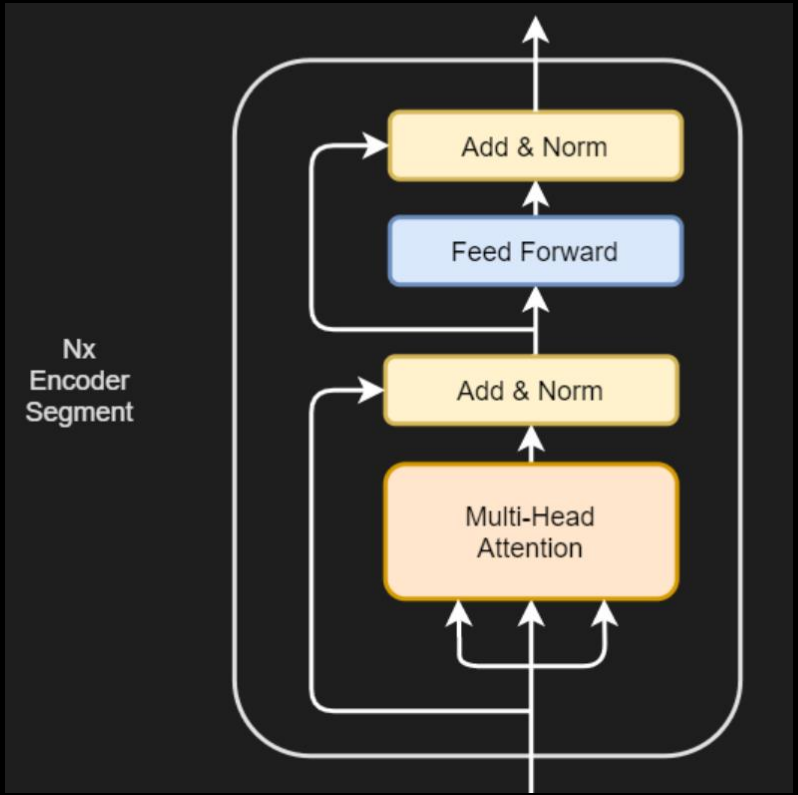
Fine-tuned approach = train using the same model all the time AND allow us to use unlabeled data sets *(semi-supervised)*

>> Feature-based approach = use pretrained models to generate features that are then used as features in a separate model.

→ BERT expands the choice of architectures that can be used during pre-training.

Transformer encoder segment

BERT utilizes the encoder segment of the original Transformer as the architecture.



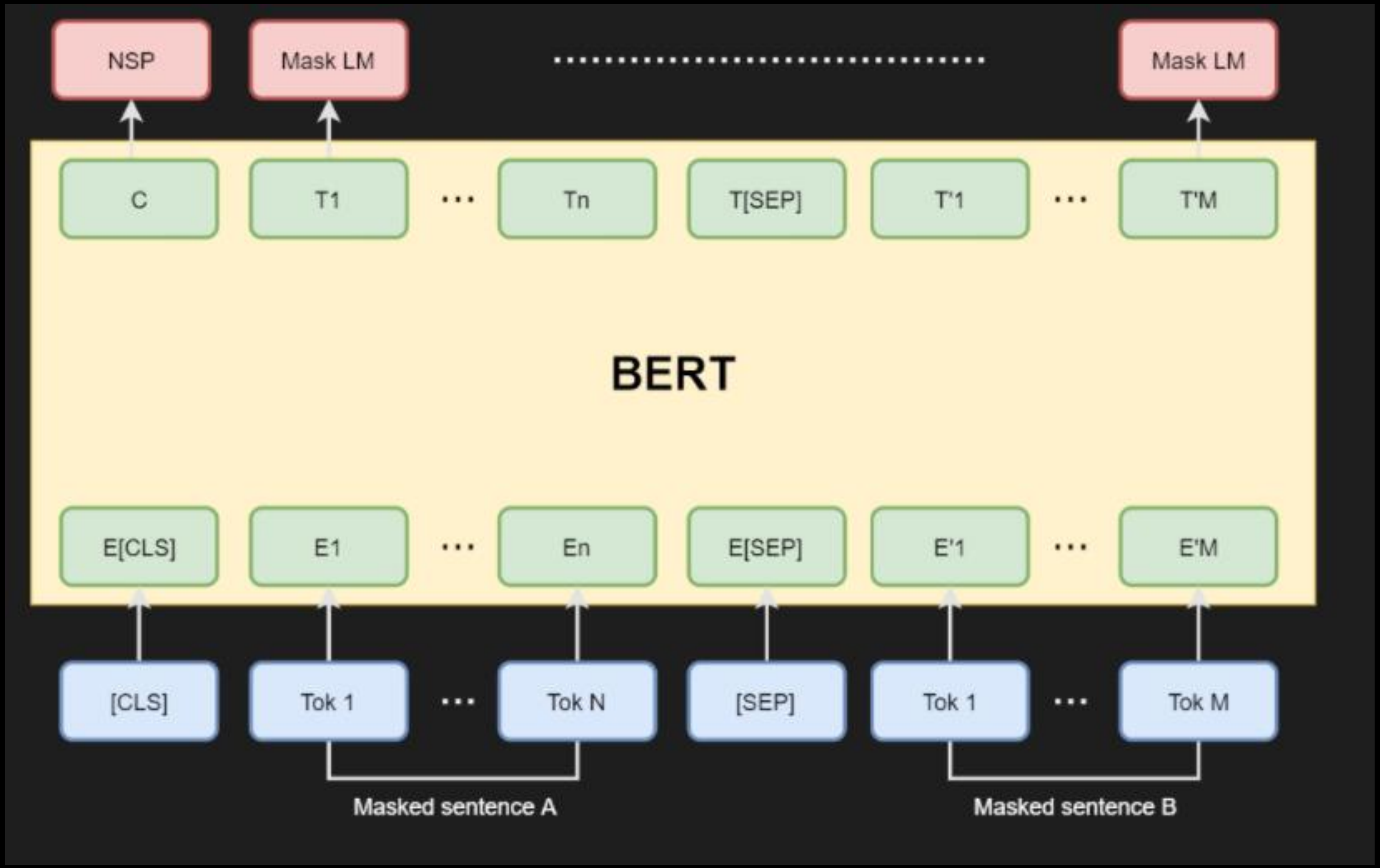
The core BERT models

BERT base (BERTBASE): has 12 Encoder Segments stacked on top of each other, has 768-dimensional intermediate state, and utilizes 12 attention heads (with hence 768/12 = 64-dimensional attention heads).

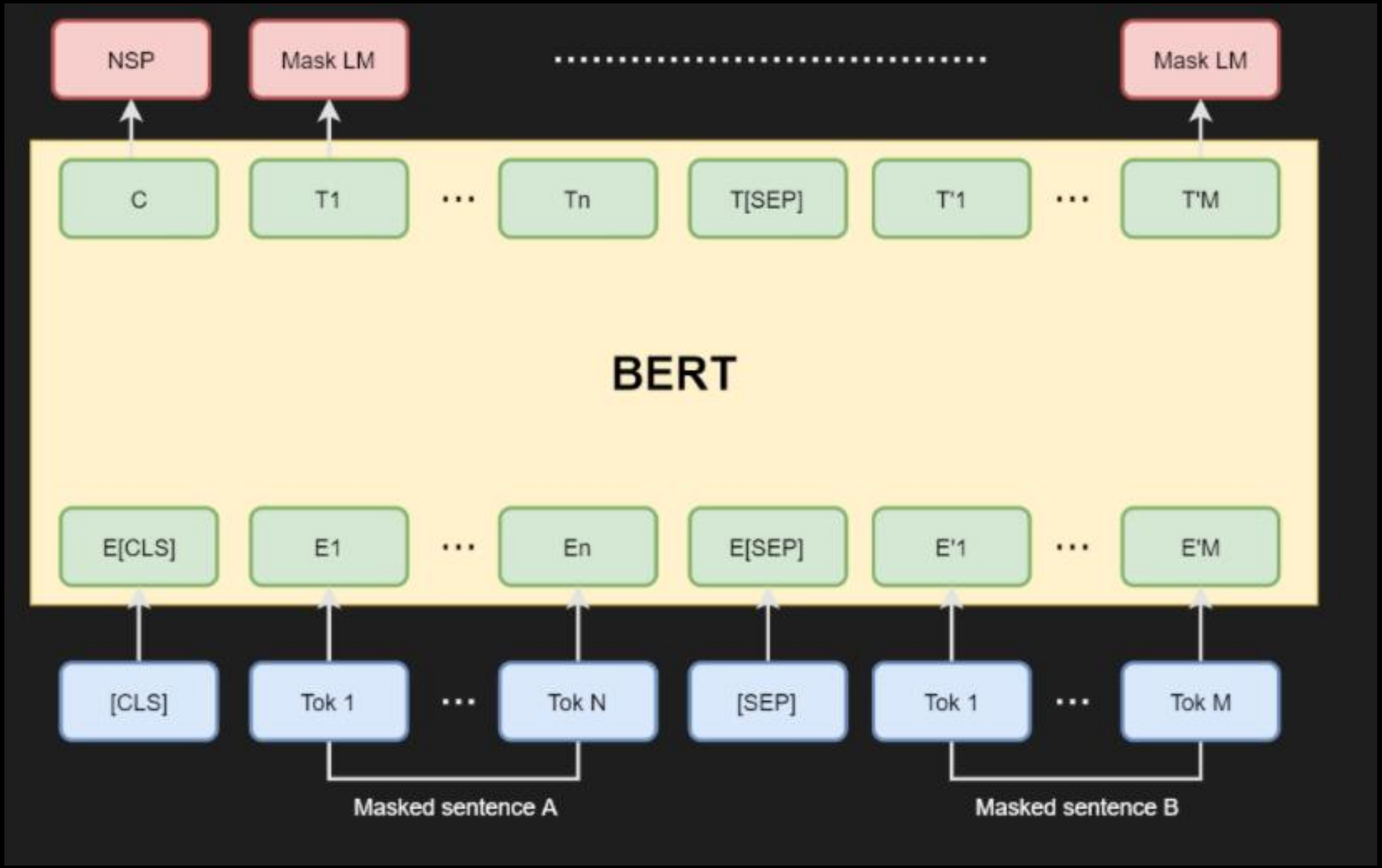
BERT large (BERTLARGE): has 24 Encoder Segments, 1024-dimensional intermediate state, and 16 attention heads (64-dimensional attention heads again).

The models are huge: the BERT base model has 110 million trainable parameters; the BERT large model has 340 million! (In comparison, classic vanilla ConvNets have hundreds of thousands to a few million)

BERT Dataflow: Inputs - Outputs



BERT Dataflow

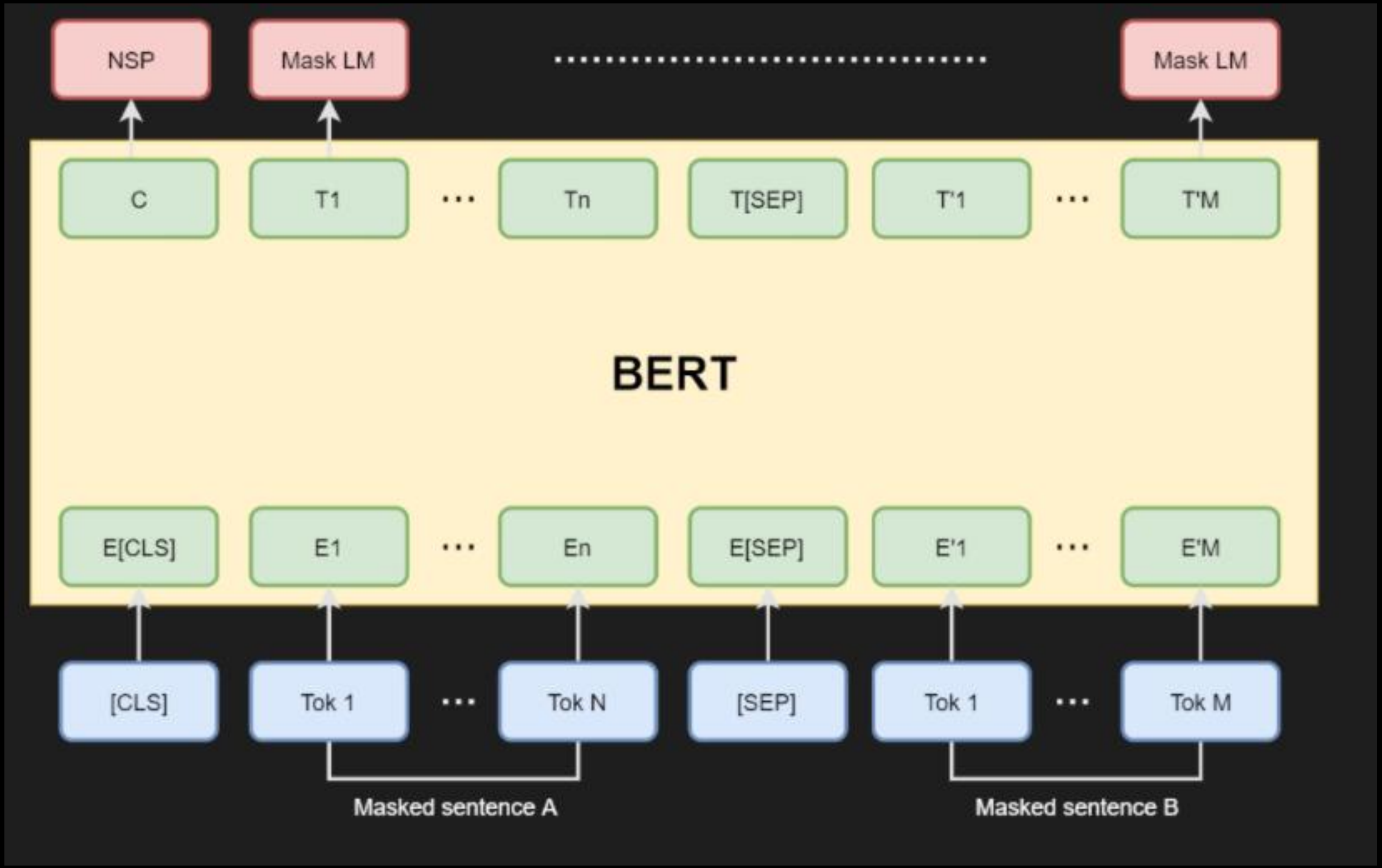


Pretraining needs 2 things: the tasks in pre-training, AND the dataset used.

During BERT pretraining of BERT, the model is trained for a combination of two tasks.

1) Masked Language Model (MLM) task
2) 2) Next Sentence Prediction (NSP) task.

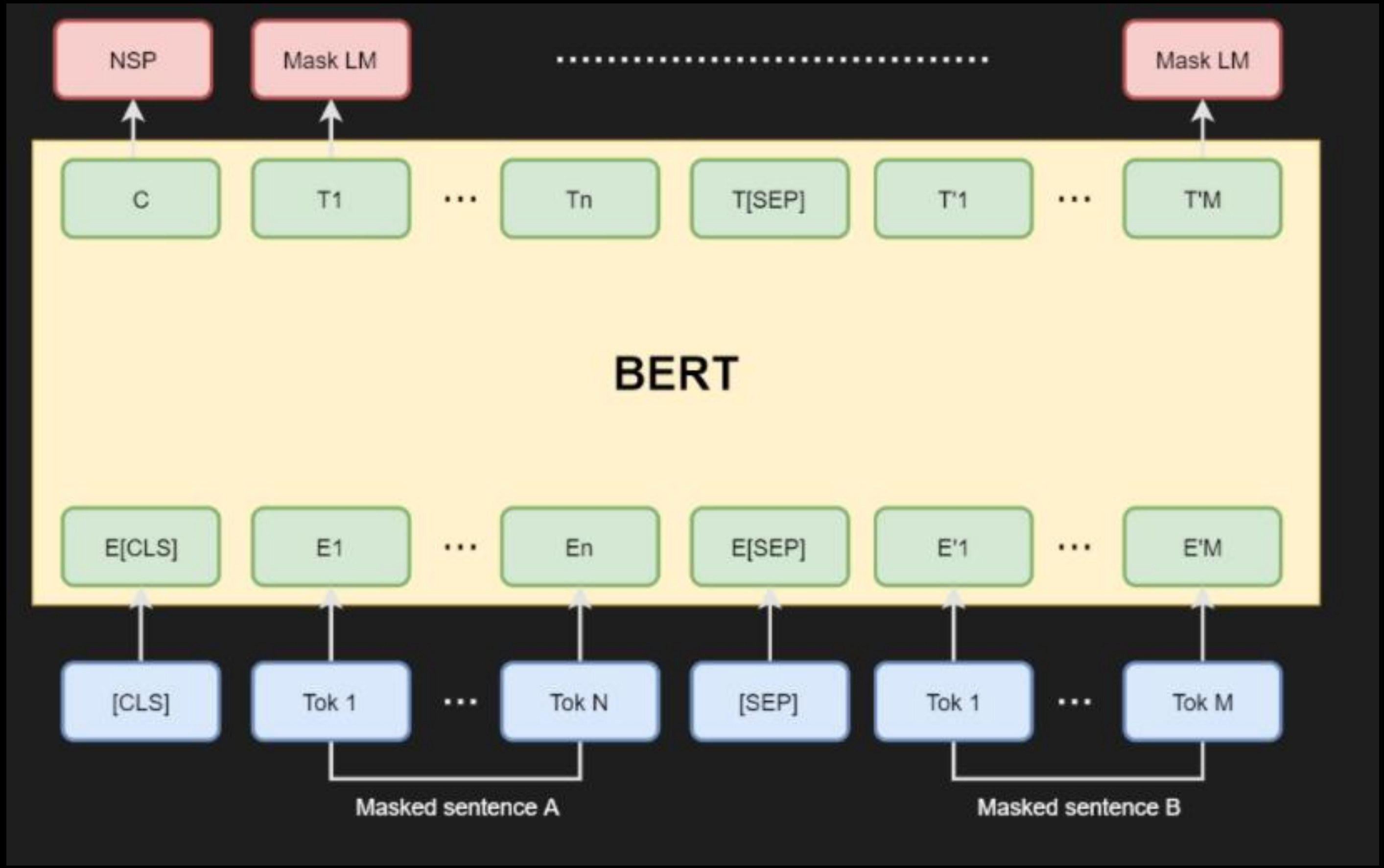
BERT Dataflow: Inputs



Inputs = either a whole sentence or two sentences packed together.

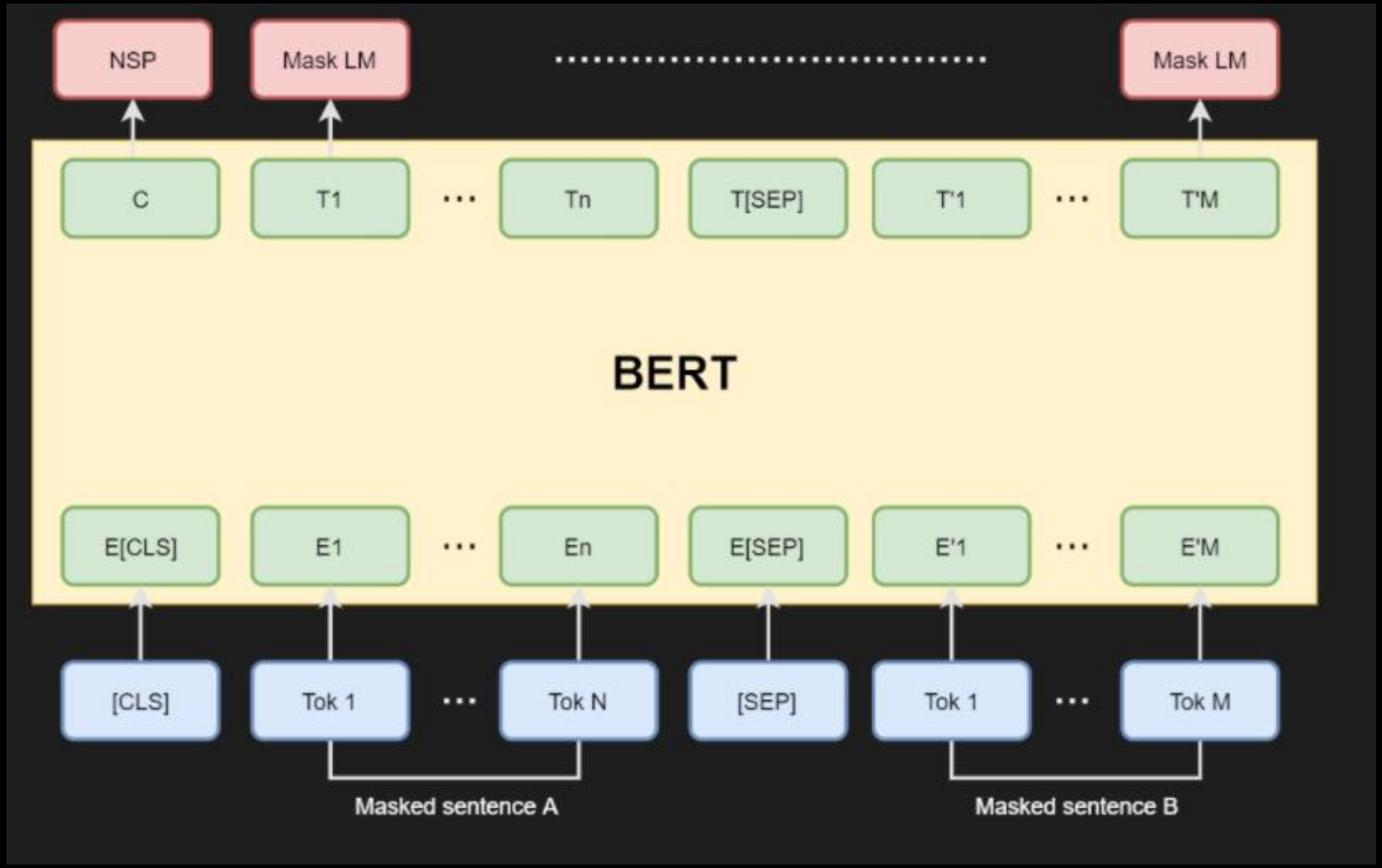
All the words are tokenized and (through BERT) converted into a word embedding.

BERT Dataflow: Inputs (cont)



[CLS] "classification" token = signals that a new combination is input and its output value can later be used for sentence-level predictions during fine-tuning, as it will learn to contain sentence-level information.

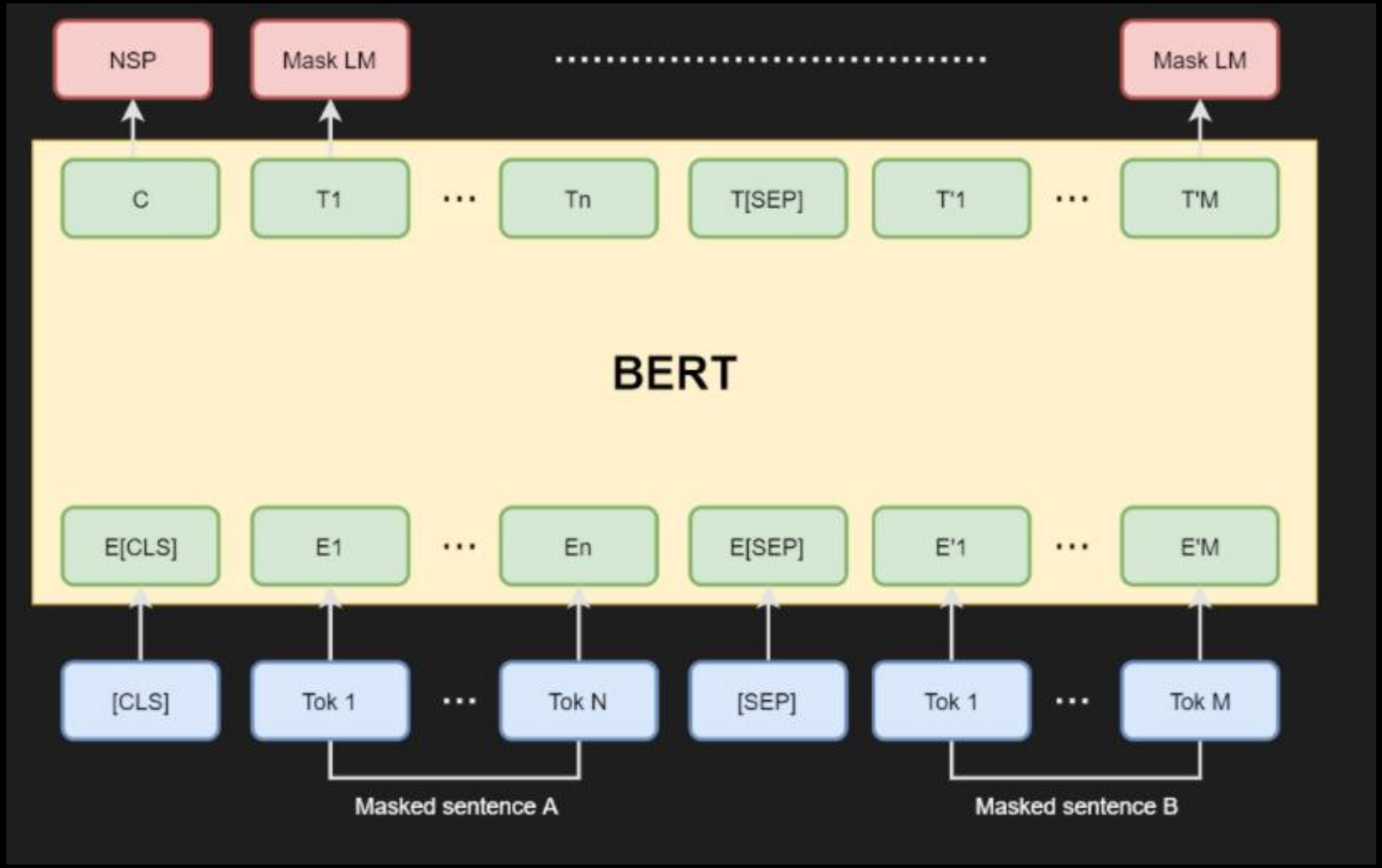
BERT Dataflow: Inputs (cont)



<u>[CLS] “classification” token</u> = signals that a new combination is input and its output value can later be used for sentence-level predictions during fine-tuning, as it will learn to contain sentence-level information.

<u>Tok 1 … Tok N ordered list of tokens</u> = all the tokens from the first sentence.

BERT Dataflow: Inputs (cont)

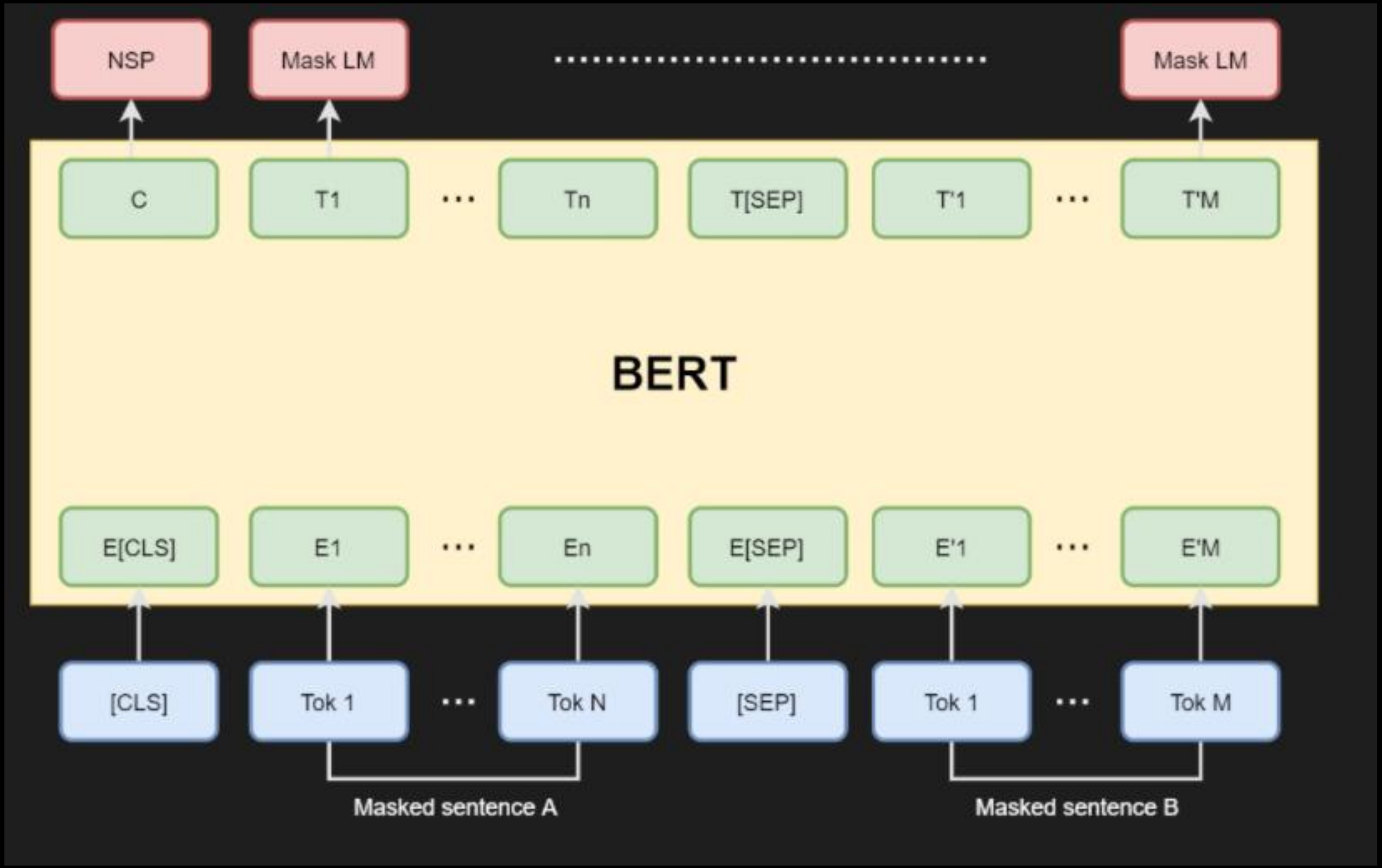


[CLS] "classification" token = signals that a new combination is input and its output value can later be used for sentence-level predictions during fine-tuning, as it will learn to contain sentence-level information.

Tok 1 ... Tok N ordered list of tokens = all the tokens from the first sentence.

[SEP] token = separates two sentences, if necessary.

BERT Dataflow: Inputs (cont)



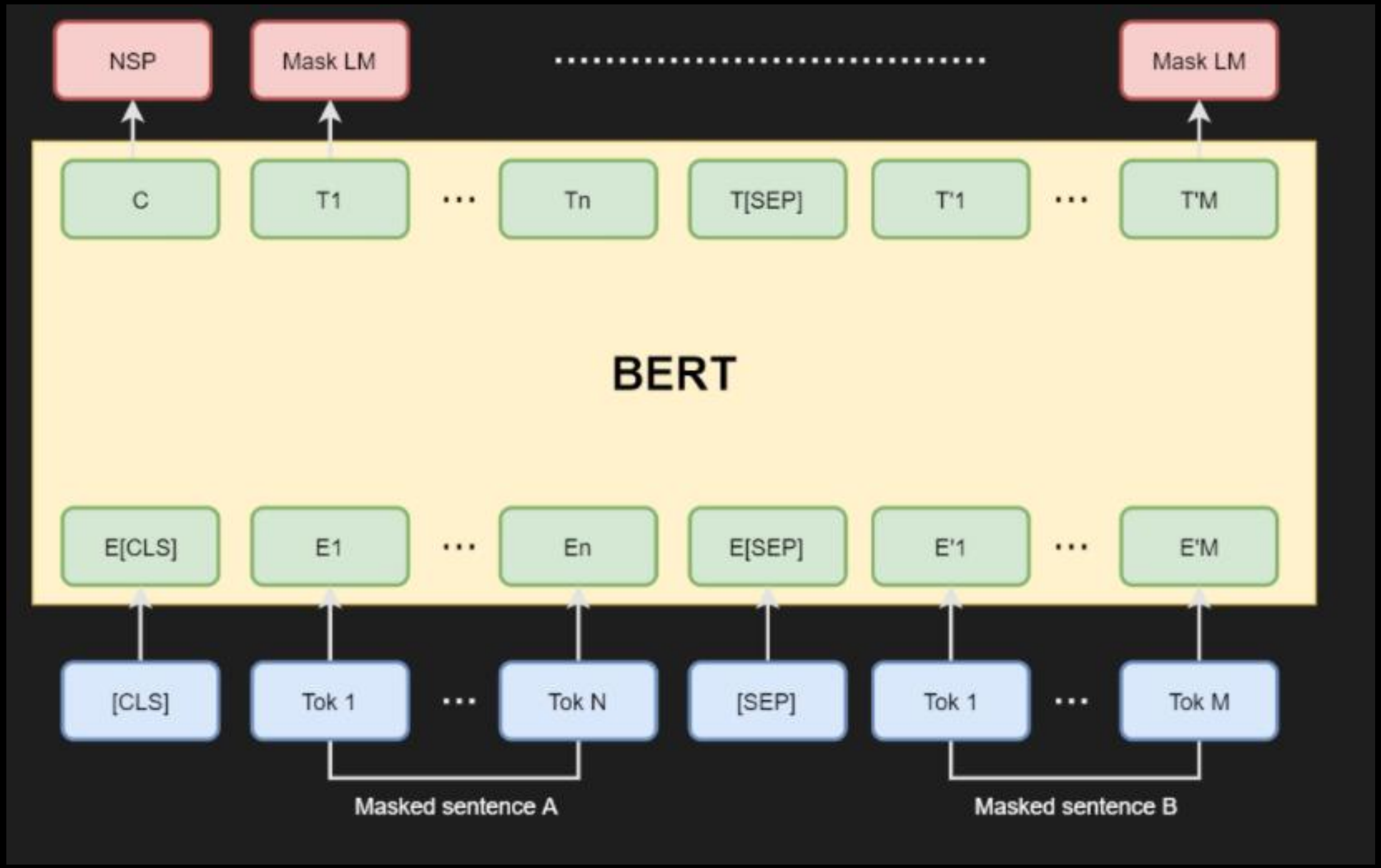
[CLS] “classification” token = signals that a new combination is input and its output value can later be used for sentence-level predictions during fine-tuning, as it will learn to contain sentence-level information.

Tok 1 ... Tok N ordered list of tokens = all the tokens from the first sentence.

[SEP] token = separates two sentences, if necessary.

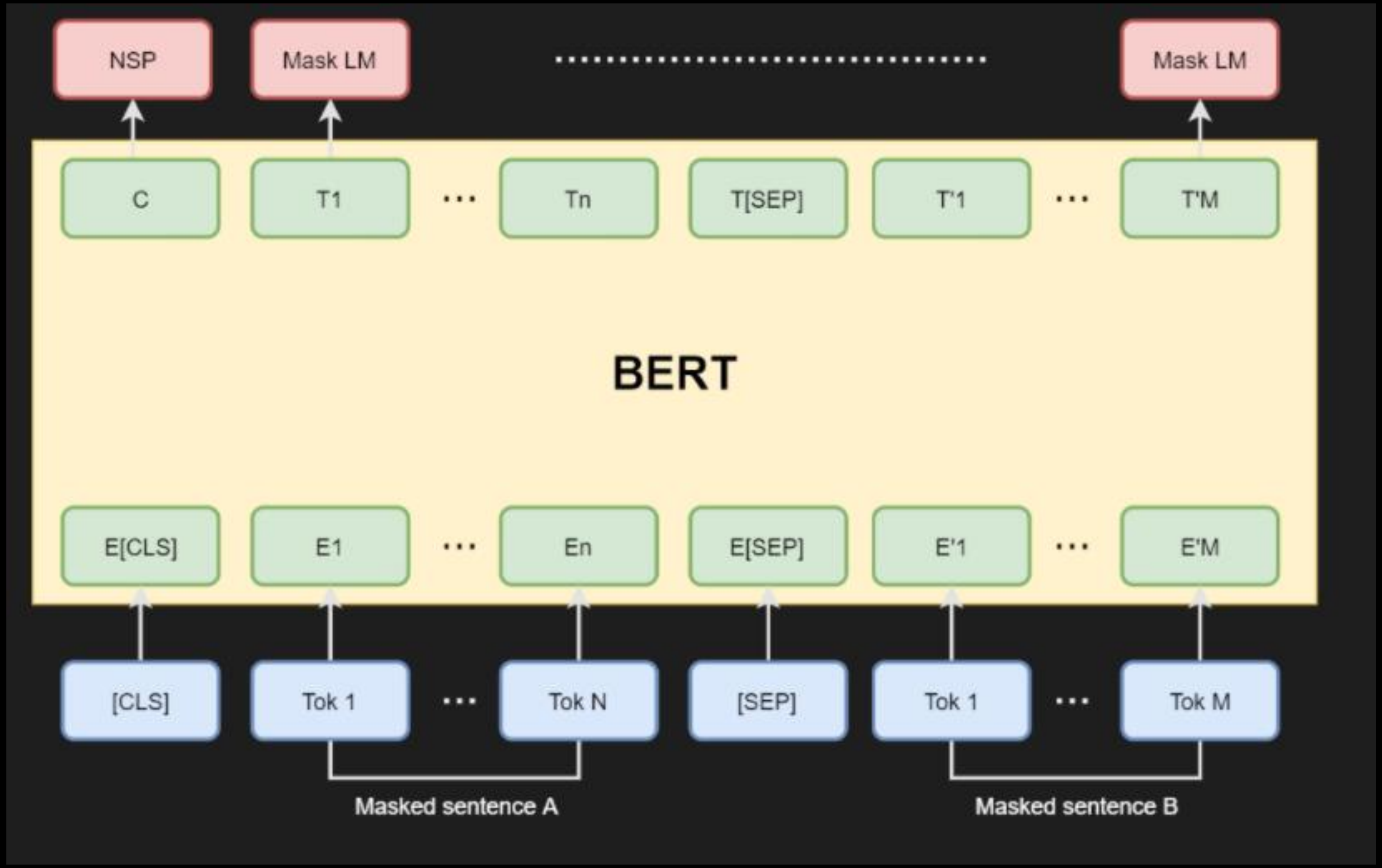
Tok 1 ... Tok M ordered list of tokens = all the tokens from the first sentence.

BERT Dataflow: from tokens to embeddings



1st component of decoder segment = word embedding

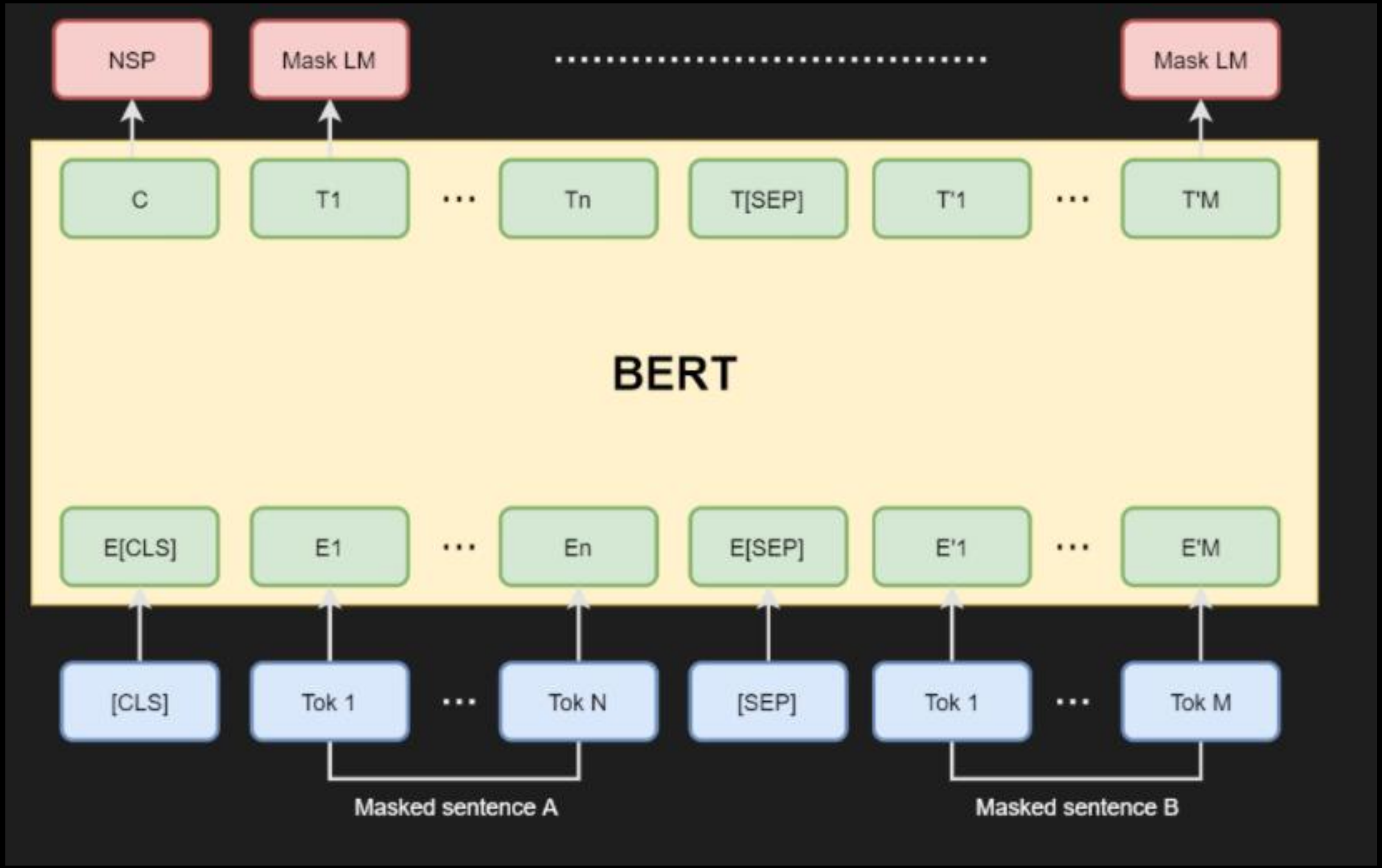
BERT Dataflow: from tokens to embeddings



1st component of decoder segment = word embedding

Word embeddings allow us to tokenized textual inputs (i.e., integers representing tokens) into vector-based format → ↓ dimensionality, ↑representation.

BERT Dataflow: from tokens to embeddings

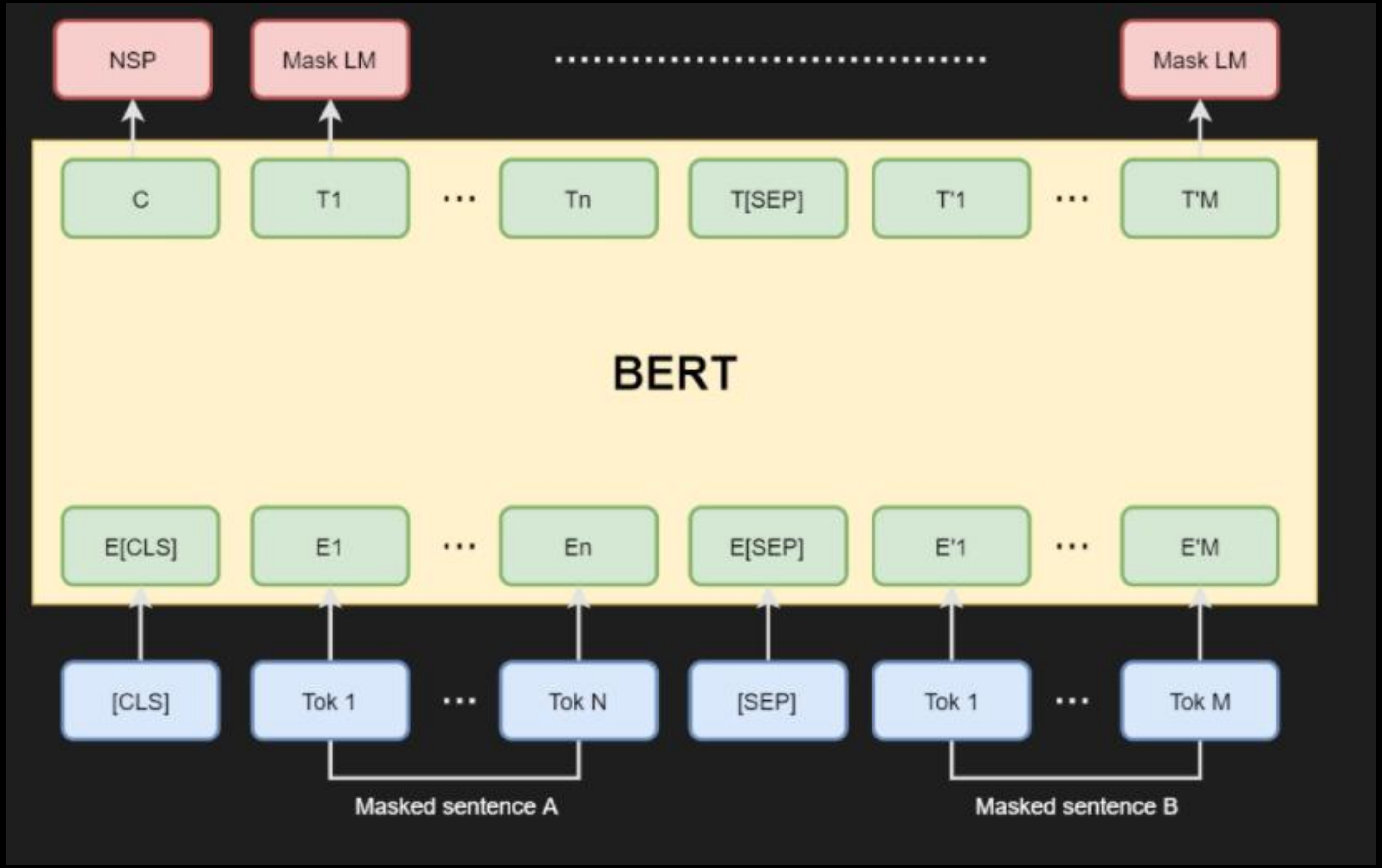


1st component of decoder segment = word embedding

Word embeddings allow us to tokenized textual inputs (i.e., integers representing tokens) into vector-based format → ↓ dimensionality, ↑representation.

We can also embed similar words together, (not possible with one-hot encoding).

BERT Dataflow: from tokens to embeddings



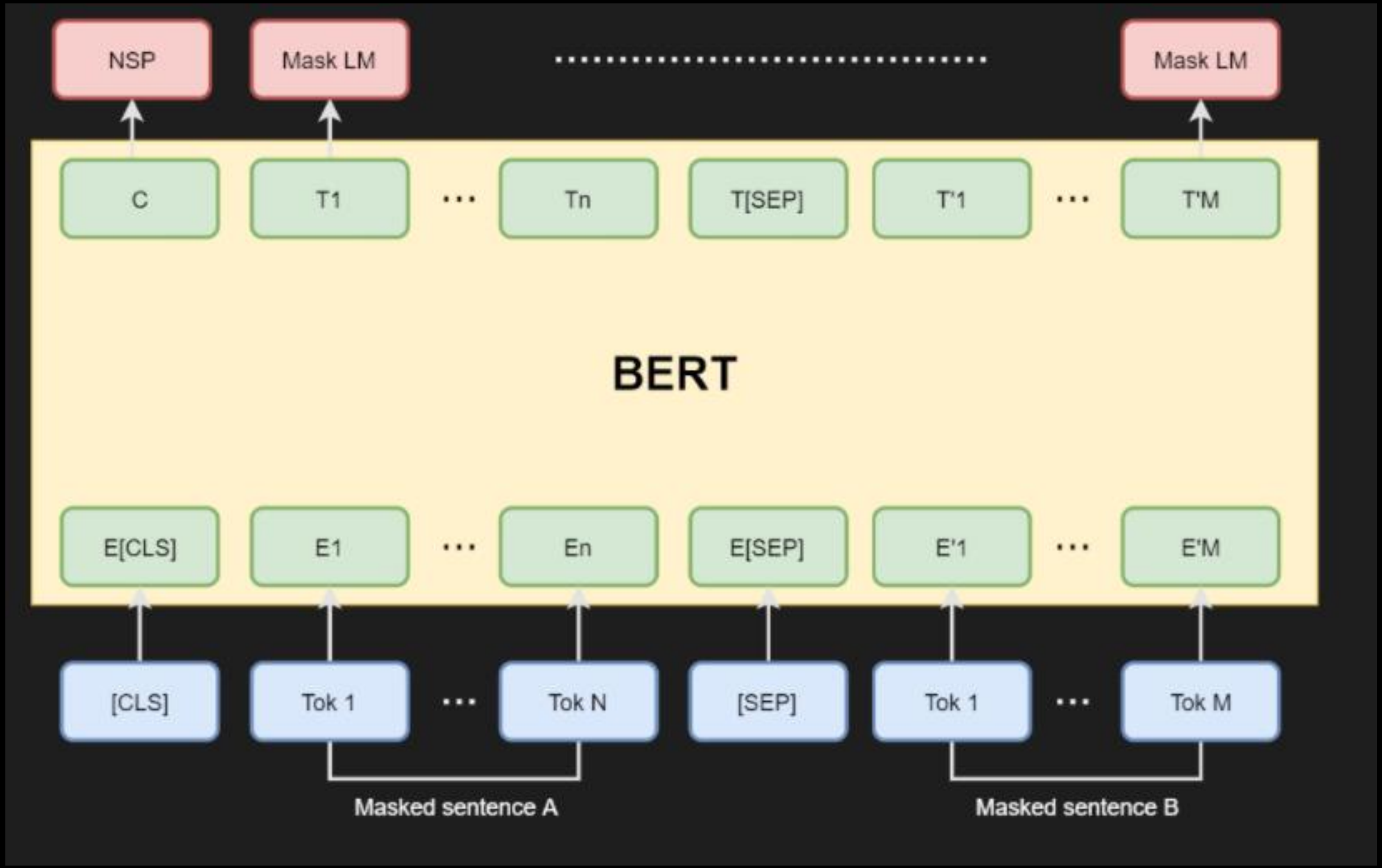
1st component of decoder segment = word embedding

Word embeddings allow us to tokenized textual inputs (i.e., integers representing tokens) into vector-based format → ↓ dimensionality, ↑representation.

We can also embed similar words together, (not possible with one-hot encoding).

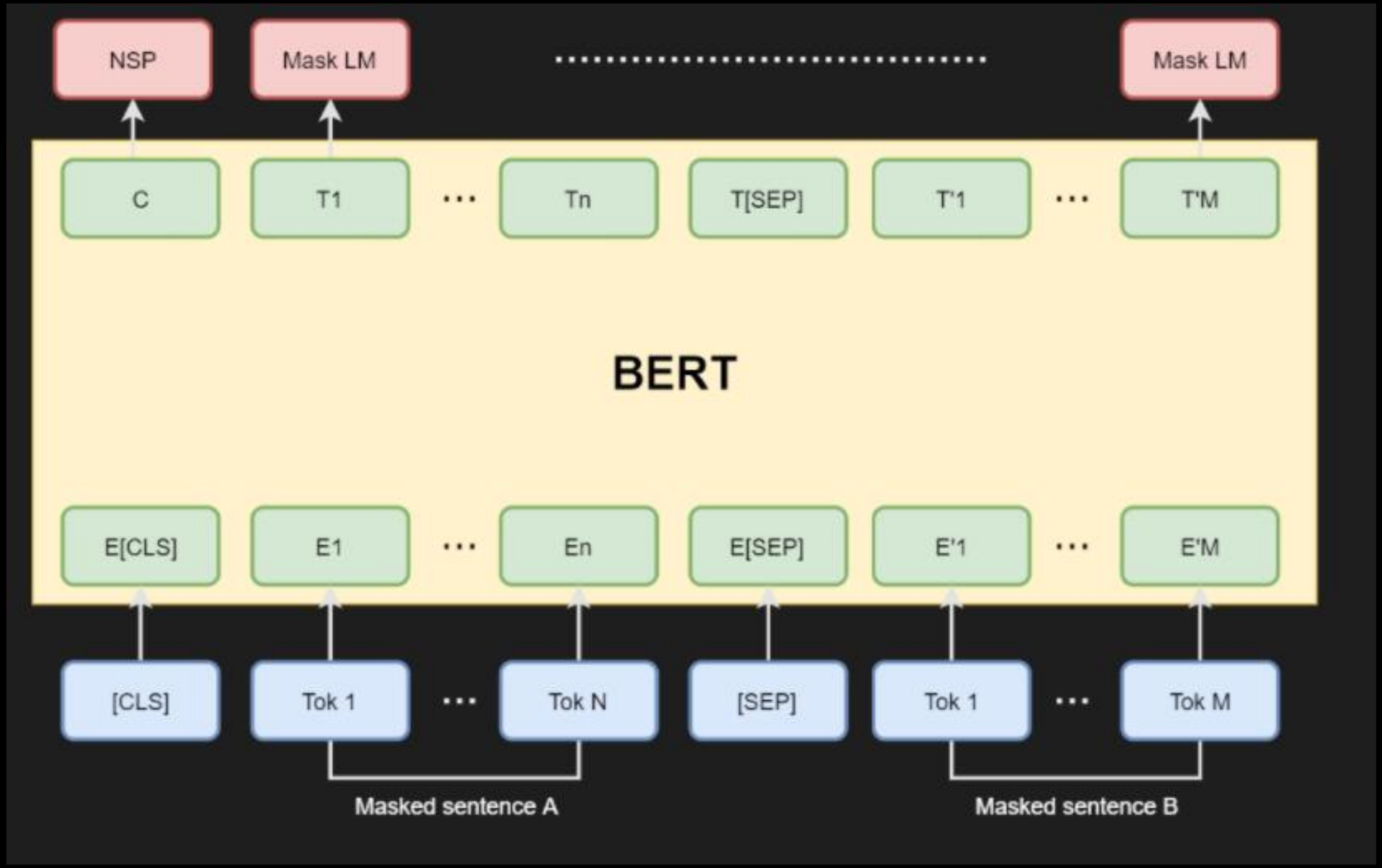
BERT uses WordPiece embeddings with a 30.000 size word vocabulary (Devlin et al., 2018)

BERT Dataflow: Outputs



BERT outputs some vectors T_i for every token.

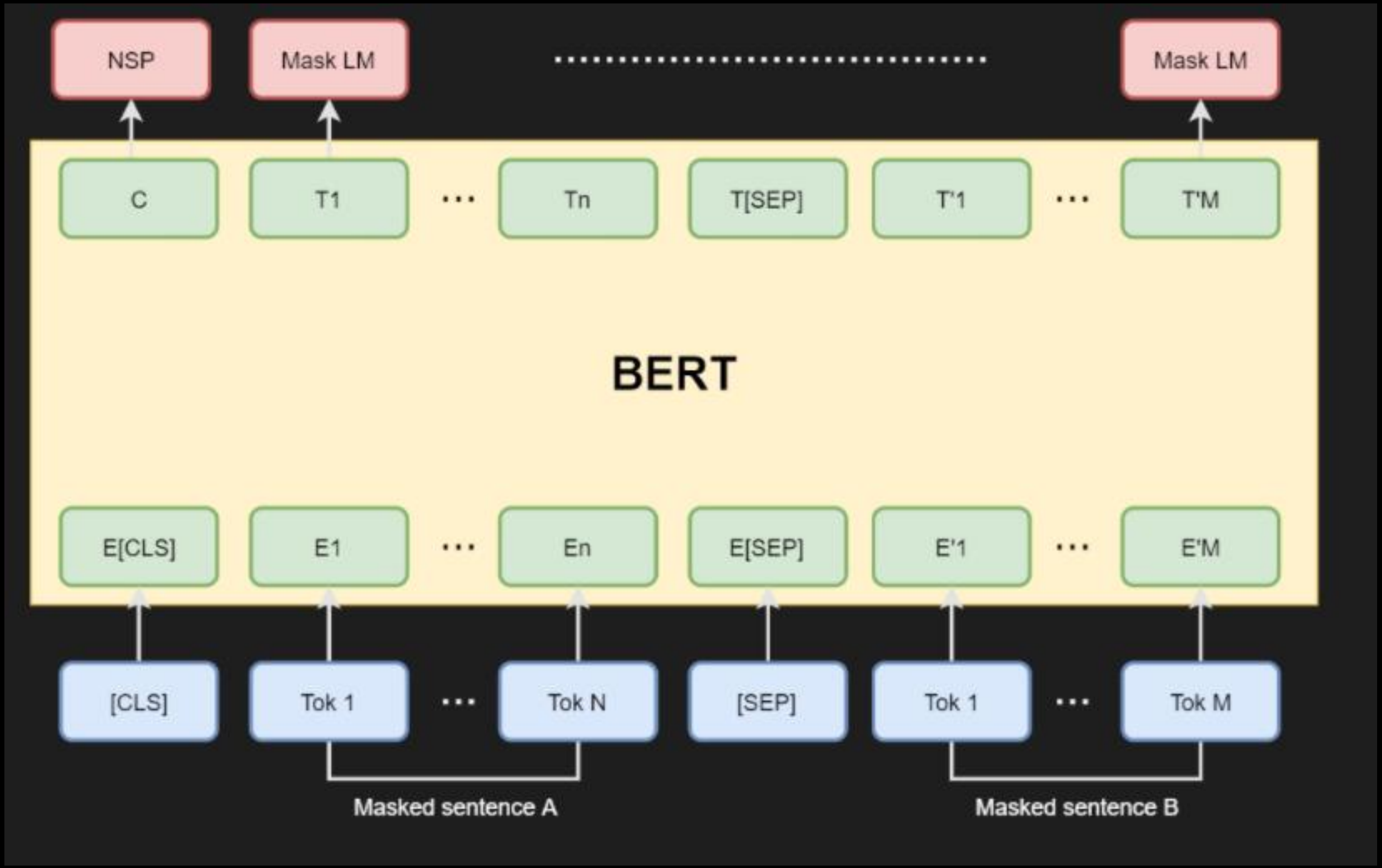
BERT Dataflow: Outputs



BERT outputs some vectors T_i for every token.

T_o= 1st vector, also called C, is the "class vector" that contains sentence-level information (or in the case of multiple sentences, information about the sentence pair).

BERT Dataflow: Outputs



BERT outputs some vectors T_i for every token.

T_o= 1st vector, also called C, is the “class vector” that contains sentence-level information (or in the case of multiple sentences, information about the sentence pair).

All other vectors are vectors representing information about the specific token.

BERT Advantages

BERT structure allows us to perform both sentence-level tasks and token-level tasks.

If we use BERT and want to work with sentence-level information, we build on top of the C token.

If we want to perform tasks related to tokens only, we can use the individual tokens.

→ High versatility ML model ☺

Setup BERT

Getting BERT downloaded and set up. We will be using the PyTorch version provided by Hugging Face and test out some Transformers pipeline.

Converting a dataset in the .csv format to the BERT .tsv format.

Loading the .tsv files into a notebook and converting the text representations to a feature representation (think numerical) that the BERT model can work with.

(ADV) Setting up a pretrained BERT model for fine-tuning.

(ADV) Fine-tuning a BERT model.

(ADV) Evaluating the performance of the BERT model

Finally…

If you haven’t yet, kindly take 5’ now to fill in the course evaluation survey – your inputs are much appreciated!

https://evasys-online.uni-hamburg.de/evasys/online.php?pswd=1S46L

Practice time ☺

Open your **Google Colab.**

https://github.com/httn21uhh/Text-Analysis-for-Social-Sciences-in-Python

→ W14_BERT.ipynb

- Download and put them in the same directory/environment.
- Work with your respective teammates.

This week…

- BERT in action
- Note that a lot of steps involving transformer modelling codes etc are advanced materials, so don't worry if you don't get the whole logic.
- Unmute yourself to ask questions at any point, including when you think things go too fast/slow for you.