Winter semester 2022/23

344.175 VL: Natural Language Processing
Introduction to Large Language Models

Navid Rekab-saz

Email: navid.rekabsaz@jku.at
Office hours: https://navid-officehours.youcanbook.me

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

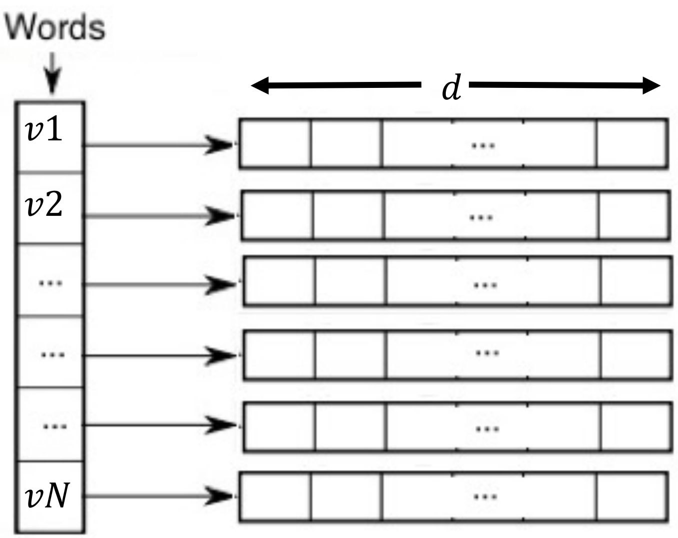
Agenda

- Background
  - Contextualized embeddings
  - Transformers – a shallow introduction
  - Subword tokenization – recap
- Large Encoder LMs with Transformers

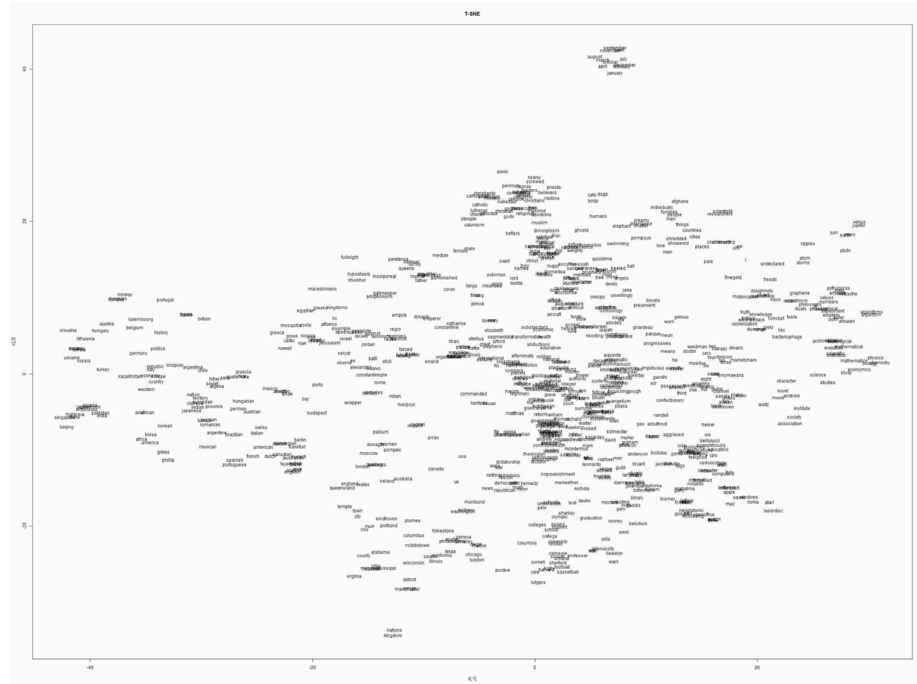
Agenda

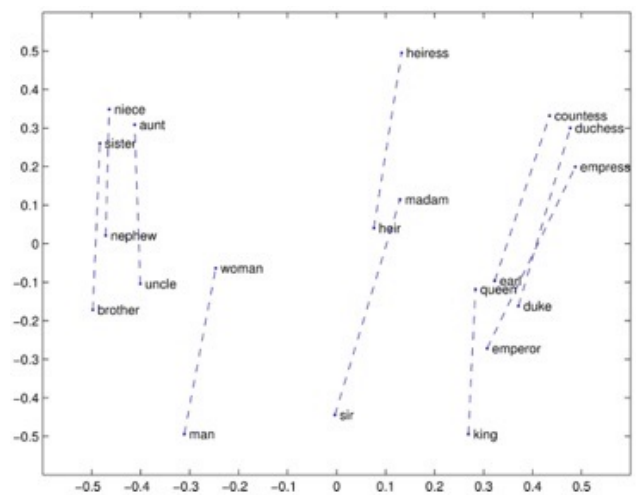
- **Background**
  - **Contextualized embeddings**
  - Transformers – a shallow introduction
  - Subword tokenization – recap
- Large Encoder LMs with Transformers

(Static) Word Embeddings



$$\boldsymbol{E} \rightarrow N \times d$$

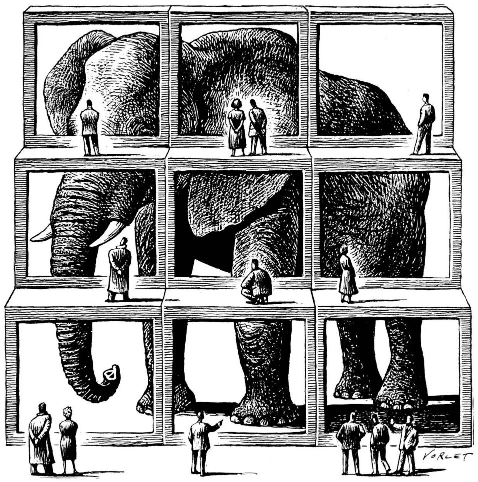




Context matters!

- Static word embeddings …
  - assign a fixed vector to each word, which in principle …
  - encodes all various meanings/relations of the word

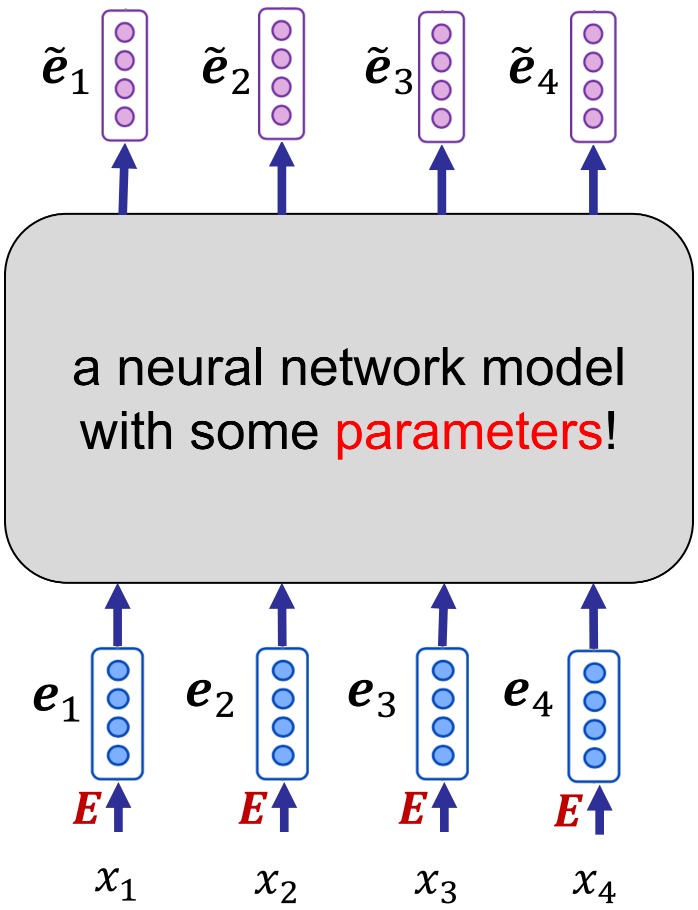
- However, the *right* meaning of a word though strongly depends on the contexts in which the word appears
  - E.g. *apple* as a word with multiple meaning can be disambiguated when considering its context:
    - *"eating an apple"* vs. "*share of the apple company*"



Contextualized Word/Token Embedding Models

- Contextualized word embeddings define the representation of a word according to the context in which the word appears
  - The contextualized embedding of a word/token can be different in different given sequences/contexts

- Input is a sequence of words/tokens, and their corresponding static word/token embeddings taken from matrix $\boldsymbol{E}$
- For each input word/token, the model "looks" at the embeddings of other words/tokens in the sequence
- The model outputs contextualized word embeddings, each corresponding to an input word/token

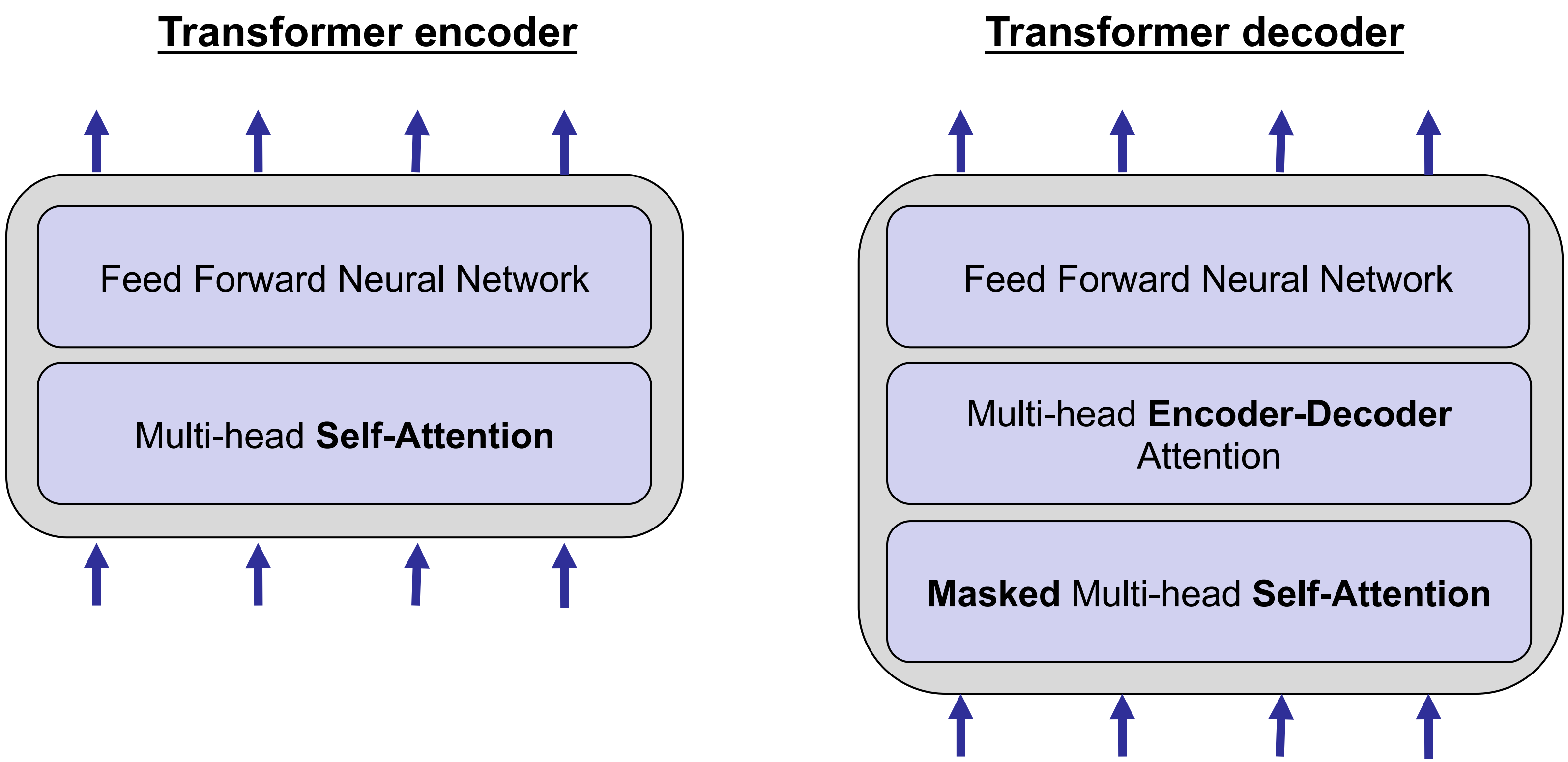


Agenda

- **Background**
  - Contextualized embeddings
  - **Transformers – a shallow introduction**
  - Subword tokenization – recap
- Large Encoder LMs with Transformers

Transformers

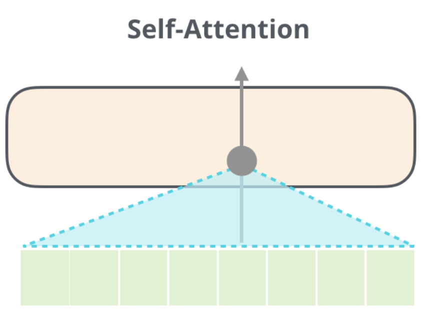
- Originally introduced in neural machine translation and now widely used for sequence encoding and decoding in various tasks

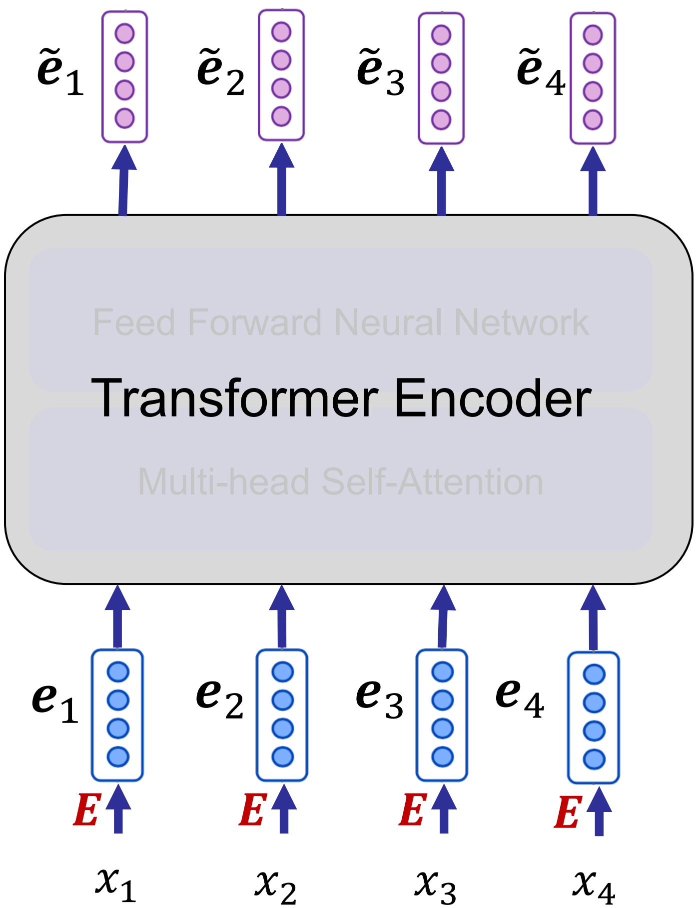


Attention is all you need. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Polosukhin, I. (2017). In *NeurIPS*.
Learn more: http://jalammar.github.io/illustrated-transformer/ https://jalammar.github.io/illustrated-gpt2/

Contextualized word/token embeddings with Transformer Encoder

- Each encoded vector is the contextual embedding of the corresponding input vector





Position embeddings

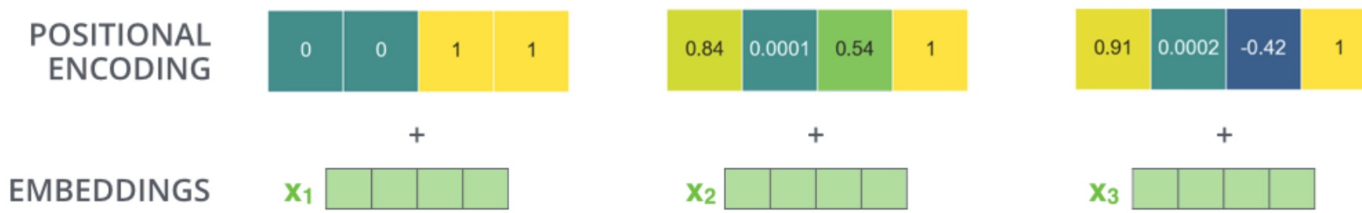
- Transformers are agnostic to the position of input tokens
  - For a given token, a context token in long-distance has the same effect as one in short-distance (no *locality bias*)
  - The positions of tokens in a sequence can be highly important in some tasks

Position embeddings – a common approach in Transformers:

- Create embeddings representing positions in a sequence, and add the values of the position embeddings to the token embeddings at corresponding positions
  - Position embedding is usually created using a sine/cosine function
    - It can also be learned end-to-end with the model parameters
  - Using position embeddings, the same token at different positions of a sequence will have different final representations

Position embeddings – examples

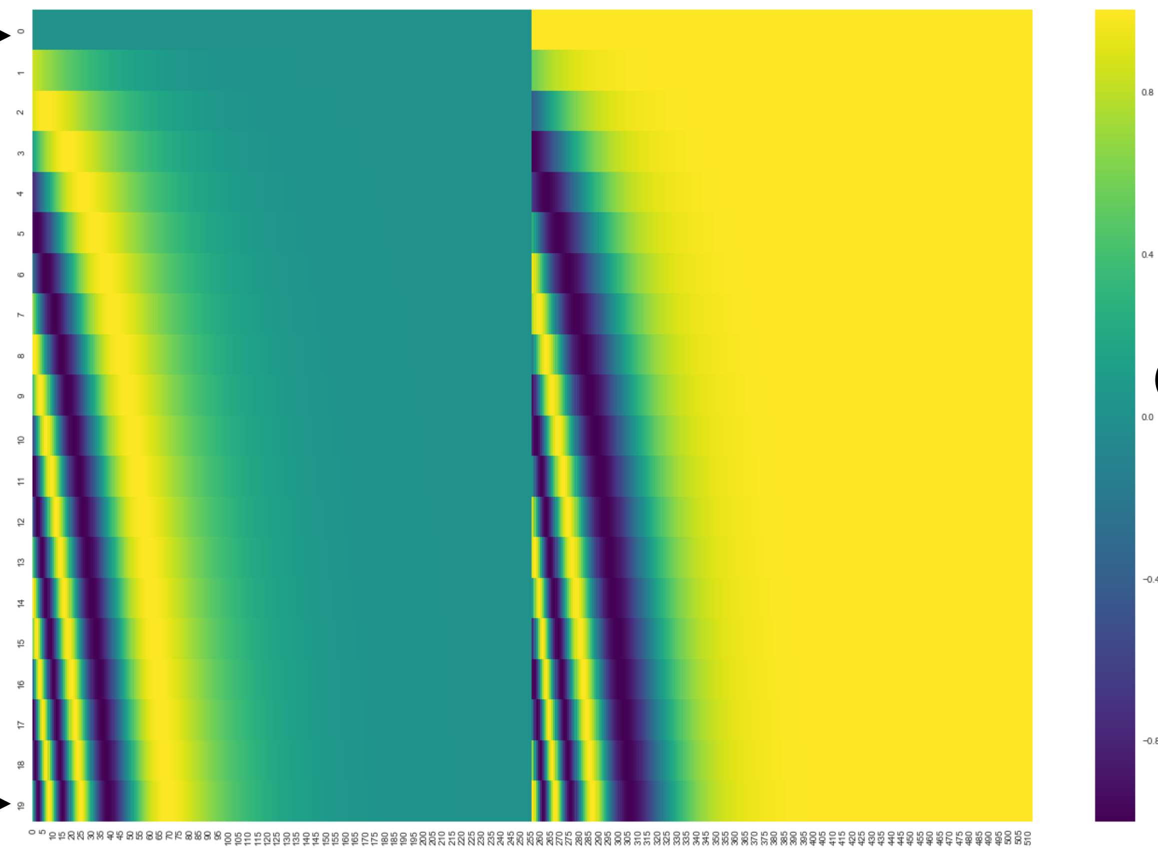
An example of embeddings with four dimensions:



Position embedding for location 0

Position embeddings

Position embedding for location 20



Values from -1 (dark) to +1 (light)

Dimensions (512)

Source: http://jalammar.github.io/illustrated-transformer/

Agenda

- **Background**
  - Transformers
  - Transformers – a shallow introduction
  - **Subword tokenization – recap**
- Large Encoder LMs with Transformers

Subwords – recap

- Words with low frequencies naturally observe a small number of contexts, and most probably end up with *worse* representations
  - E.g., "***structurally***" appears rarely, however, its meaning can be inferred from "***structure***" which may appear much more often in a corpus
    - Lemmatizers and stemmers turn "***structurally***" and "***structure***" to the same stem (like "***structur***") but they the differences between these two

- Ways to define subwords
  - Fixed-length like character tri-grams in fastText
  - Variable-length like Byte Pair Encoding, WordPiece, and SentencePiece

- Subword tokenizer
  - Training time: creates a vocabulary list of subwords using corpus statistics,
  - Tokenization/Decoding time: uses this vocabulary list to decompose a given word (or a given sequence) to subwords

Subword tokenization
Byte Pair Encoding (BPE)

- The core idea of BPE comes from information theory and compression

- BPE (or in general subword tokenizers) consist of two steps:
  1. **Training:** Learning a vocabulary list of subwords from a given corpus
  2. **Tokenization (decoding):** tokenize a given text using the stored subwords vocabulary list

Reference: Sennrich, R., Haddow, B., & Birch, A.. Neural Machine Translation of Rare Words with Subword Units. In *proceedings of ACL (2016)*

Byte Pair Encoding (BPE)

Sketch of training:

1. Pre-tokenize the training corpus, for instance simply by splitting on white spaces
2. Start from a vocabulary list with all single characters
3. Create a dictionary of words and counts from the pre-tokenized training corpus
4. Add special character "_" to the end of each word in the dictionary
5. Expand the vocabulary list:
    - Find the most frequent pair of characters in the dictionary of words
    - Merge the characters, and add them to the vocabulary list
    - Repeat step 5 till some limits on vocabulary size are reached

Learn more at "*Speech and Language Processing (3rd ed.) D. Jurafsky and J. H. Martin*", section 2.4.3.
https://web.stanford.edu/~jurafsky/slp3/2.pdf (the resource of the example)

Byte Pair Encoding – example

- Consider a tiny training corpus that leads to the following dictionary and vocabulary list

	dictionary	vocabulary
5	`l o w _`	`_, d, e, i, l, n, o, r, s, t, w`
2	`l o w e s t _`	
6	`n e w e r _`	
3	`w i d e r _`	
2	`n e w _`	

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

First merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_
2	l o w e s t _	
6	n e w e r_	
3	w i d e r_	
2	n e w _	

Next merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

Next merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew
2	l o w e s t _	
6	n ew er_	
3	w i d er_	
2	n ew _	

If we continue

Merge	Current Vocabulary
(n, ew)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new
(l, o'	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_, low_

WordPiece tokenization

- WordPiece is a descendent of BPE and has the following differences:

- Selecting character pairs for merging in WordPiece is based on "*minimizing the language model likelihood of the training data*"*

- WordPiece indicates internal subwords with "##" special symbol
  - E.g. *"**unavoidable**"* → [*"**un**"*, *"**##avoid**"*, *"**##able**"*]

* For more details look into the original paper:
Schuster, M., & Nakajima, K. (2012). Japanese and Korean voice search.
In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*

WordPiece tokenization

- Tokenization (decoding) is done using MaxMatch algorithm
  - A greedy longest-match-first algorithm
  - MaxMatch choses the longest token in the vocabulary that matches the given word
  - After a match, it repeats the previous step with the remainder of the word

```
function MaxMatch(string, dictionary) returns list of tokens T

    if string is empty
        return empty list
    for i ← length(sentence) downto 1
        firstword = first i chars of sentence
        remainder = rest of sentence
        if InDictionary(firstword, dictionary)
            return list(firstword, MaxMatch(remainder,dictionary) )
```

WordPiece tokenization

- WordPiece with MaxMatch decoding is used in some models such as BERT

Example

Original sequence:

"natural language processing"

pre-tokenization:

[*"natural"*, *"language"*, *"processing"*]

subword tokenization:

[*"natural"*, *"lang"*, *"##uage"*, *"process"*, *"##ing"*]

function MAXMATCH(string, dictionary) **returns** list of tokens T

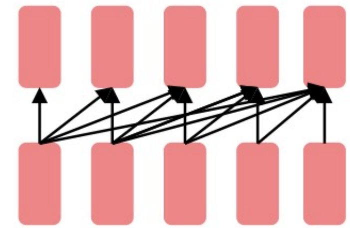
if string is empty
  return empty list
for $i \leftarrow length$(sentence) **downto** 1
  firstword = first *i* chars of *sentence*
  remainder = rest of *sentence*
  if InDictionary(*firstword*, *dictionary*)
    return list(firstword, MaxMatch(remainder,dictionary))

Agenda

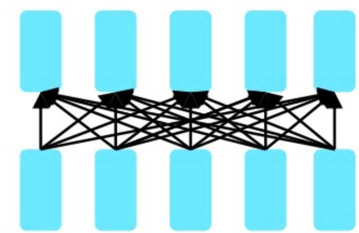
- Background
  - Contextualized embeddings
  - Transformers – a shallow introduction
  - Subword tokenization – recap
- **Large Encoder LMs with Transformers**

Large Language Models (LLMs)

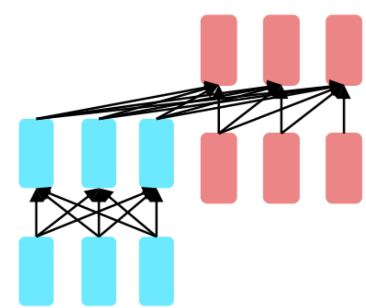
- Encoder LLMs
  - Model sees whole the sequence (past and future)
  - Input sequence is encoded into contextualized embeddings
  - Additionally, some models provide a sequence embedding or a pair-sequence embedding
  - Representative models: BERT, RoBERTa, XLM-*, ELMo



- Decoder LLMs
  - “Normal” LM objective: predict the next token conditioned on the previous tokens (unidirectional)
  - Training and inference is auto-regressive (one after each other)
  - Representative models: GPT-x

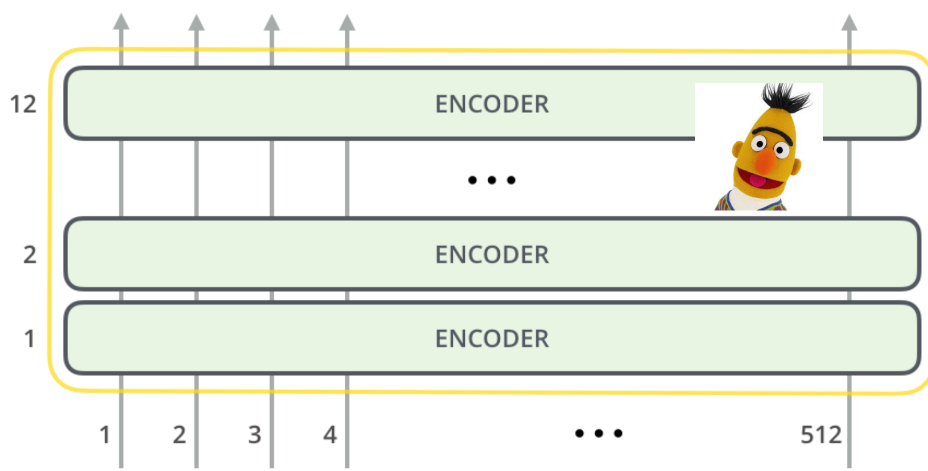


- Encoder-Decoder LLMs
  - The encoder encodes whole the input (bidirectional)
  - The decoder generates the output in auto-regressive fashion
  - Representative models: T5, BART



BERT
Bidirectional Encoder Representation from Transformers

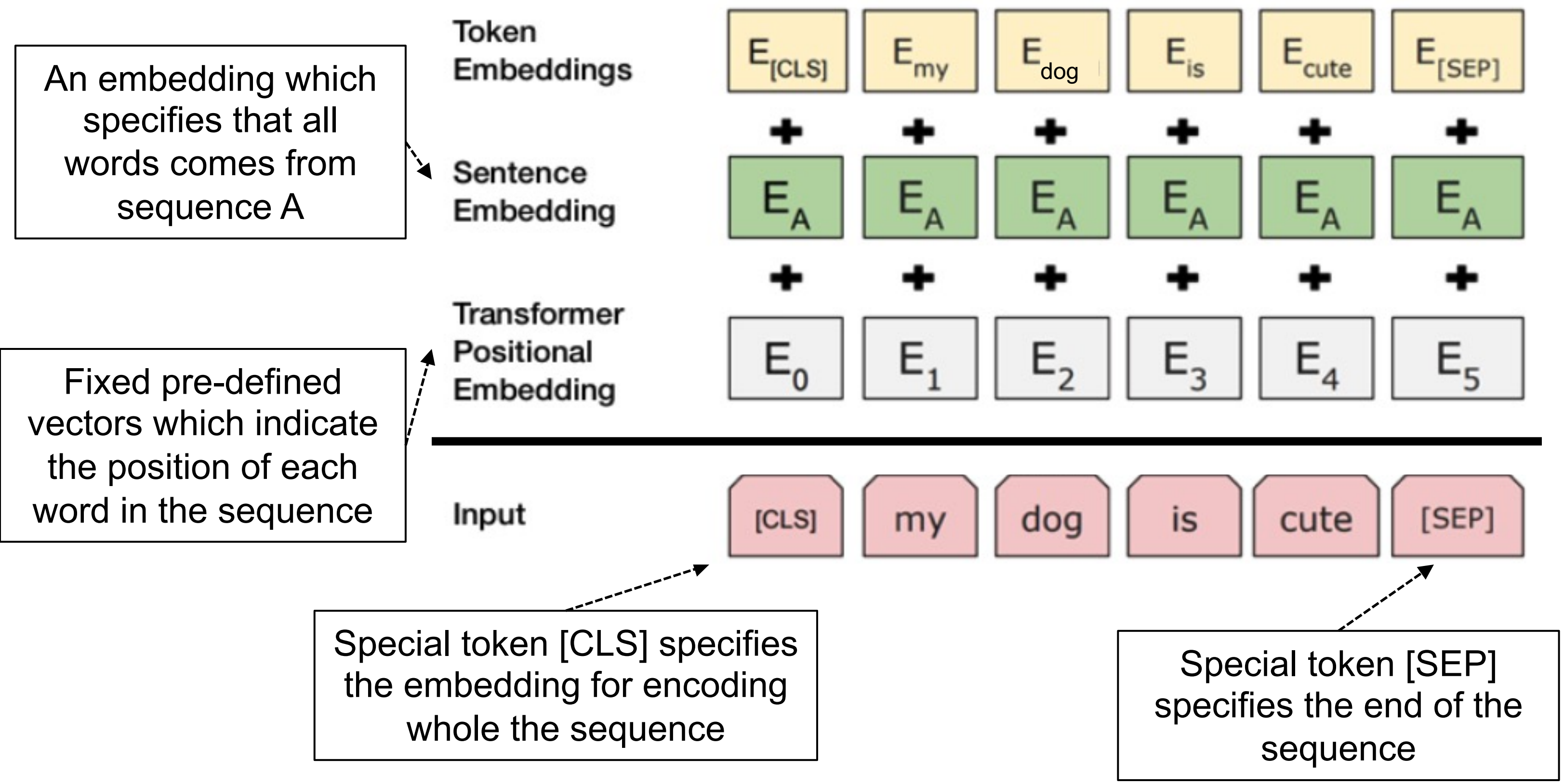
- BERT is a pre-trained Encoder LLM which …
  - is composed of multi-layers of Transformer Encoder,
  - uses WordPiece for tokenization,
  - trained with a Masked Language Model objective,
  - provides contextualized word embeddings …
  - as well as a sequence or pair-sequence embedding for one/multiple sequence(s) using sentence pair encoding



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, June). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. of NAACL-HLT (2019)*

Input embeddings (for one sequence)

- The input to Encoder Transformers is in fact the element-wise sum of three types of embeddings
  - Token embeddings, taken from a static subword embedding matrix (learnable)
  - Sentence embeddings (fixed)
  - Transformer position embeddings (fixed)



Masked Language Model (MLM)

- "Normal" language modeling objective: move from left to right (or right to left) and in each step predict the next token
  - The LM can see the full context on processing the last token
  - Good fit for language generation but suboptimal for sequence encoding

- Masked Language Model objective masks out $k\%$ of the tokens of input sequence, passes the masked sequence to the model, and predicts the masked words in output

Example

sequence: Jim made spaghetti for his girlfriend and he was very proud!

Input: Jim made [MASK] for his girlfriend and [MASK] was very proud!

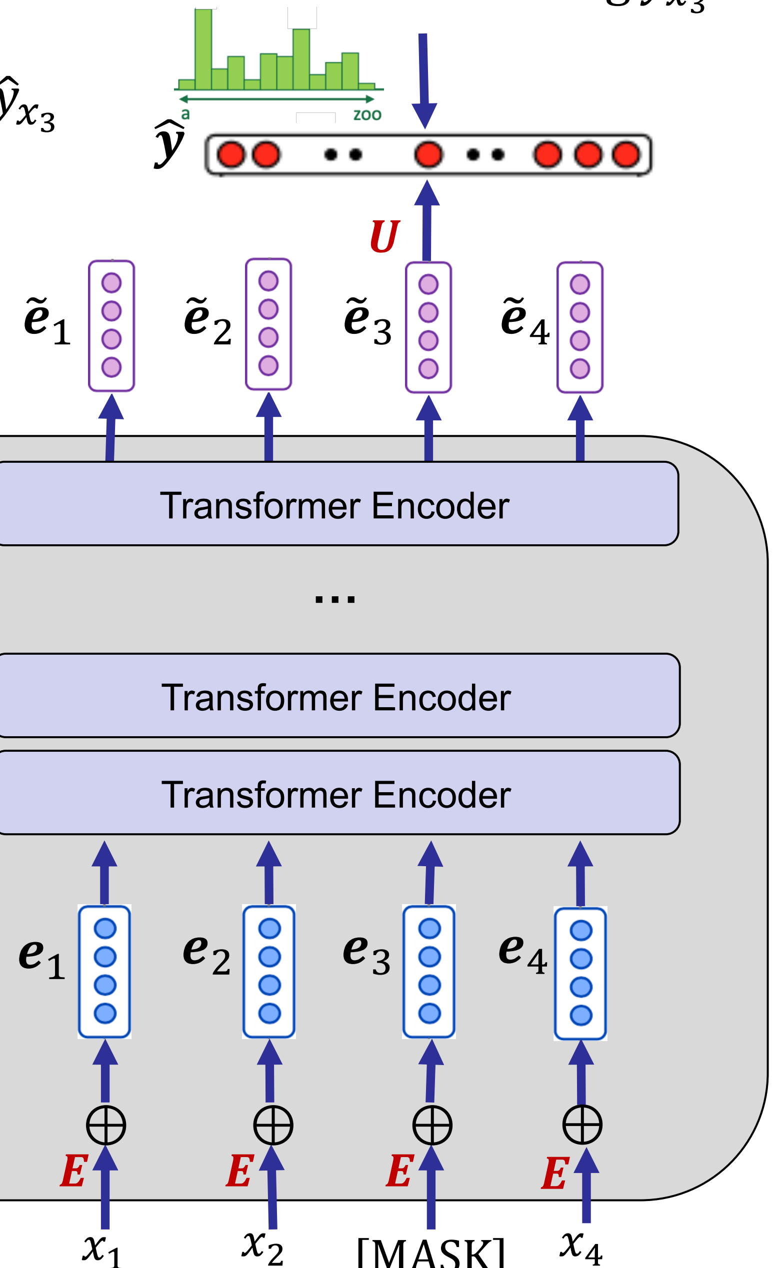
$$\downarrow \qquad\qquad \downarrow$$

predict: spaghetti he

MLM training

$$P(x_3 | \text{masked sequence}) = \hat{y}_{x_3}$$

$$\mathcal{L} = -\log \hat{y}_{x_3} = -\log \hat{y}_{\text{cat}}$$



Corpus:
`a fluffy cat sunbathes`
A training datapoint:
`a fluffy [MASK] sunbathes`

BERT Training setting

- Trained using MLM on Wikipedia + BookCorpus

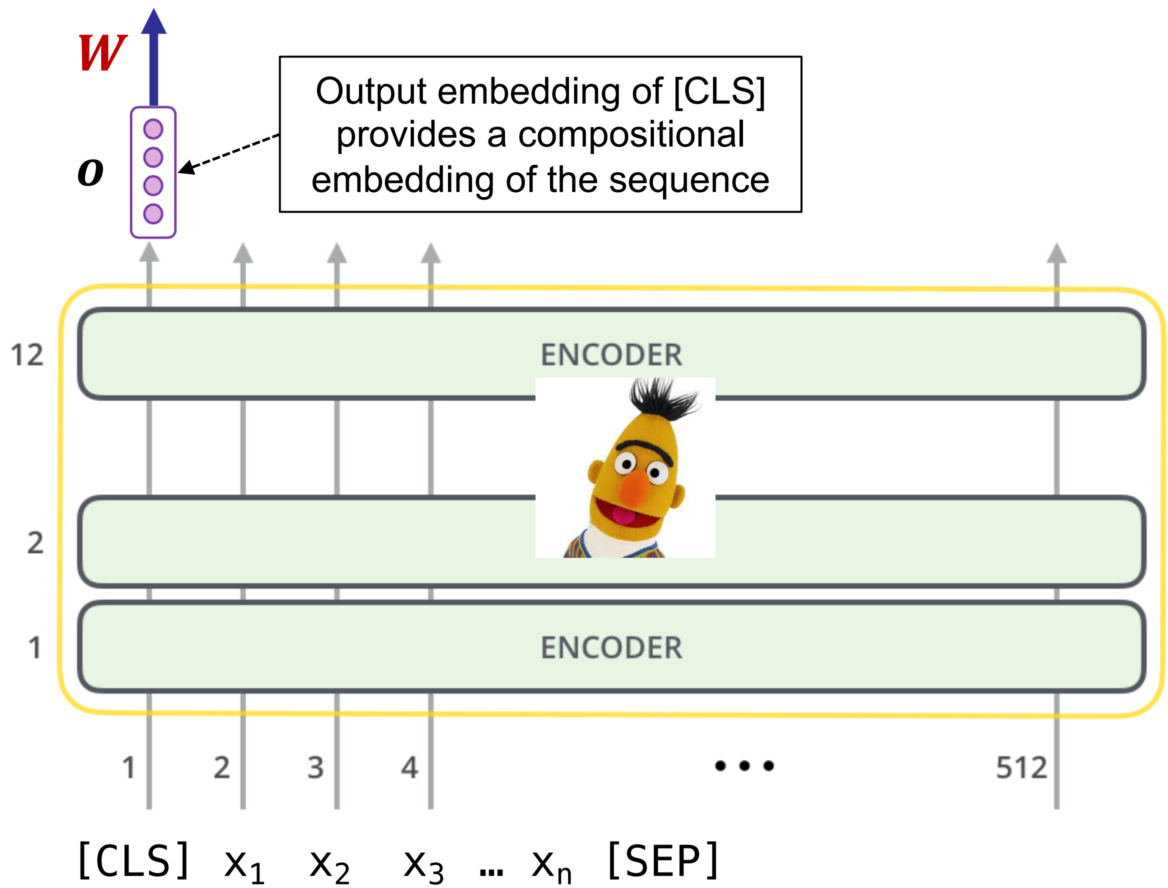
- Dictionary size is ~30K tokens (due to WordPiece subword tokenization)
- Specs of some provided pre-trained models:
  - BERT-Tiny: 2-layer, 128-hidden, 2-head, ~4M parameters*
  - BERT-Mini: 4-layer, 256-hidden, 4-head, ~11M parameters*
  - BERT-Base: 12-layer, 768-hidden, 12-head, ~110M parameters*
  - BERT-Large: 24-layer, 1024-hidden, 16-head, ~340M parameters*

- Some resources:
  - https://github.com/google-research/bert
  - Library to have BERT models in PyTorch: https://huggingface.co/transformers/

* For comparison, a (static) word embedding like word2vec with vocabulary size 200K and vector size 768 has 153M parameters

BERT fine-tuning for one sequence

$$\hat{\boldsymbol{y}} = P(Y|X) = \mathrm{softmax}(\boldsymbol{o}\boldsymbol{W} + \boldsymbol{b})$$



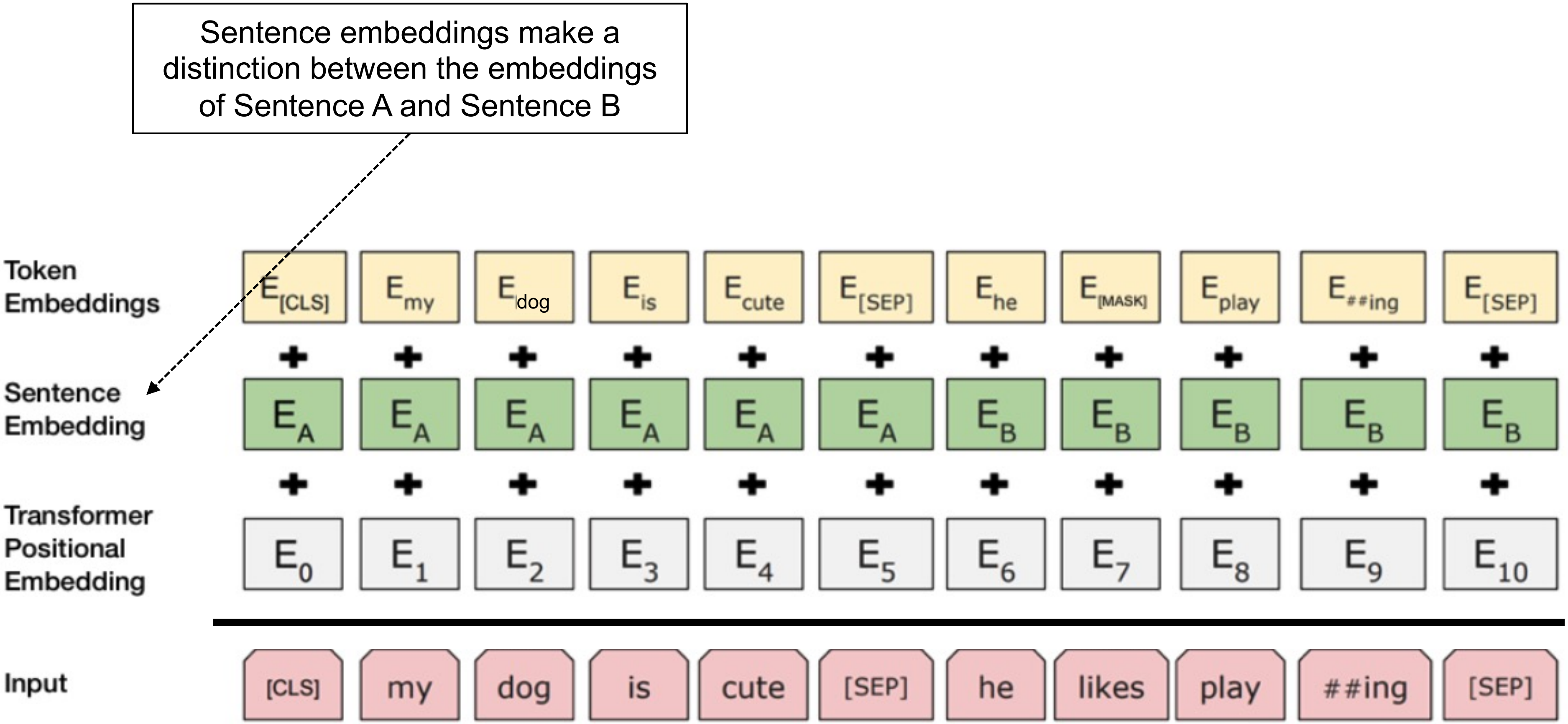
Sentence (Sequence) pair encoding

- Many NLP tasks need to calculate the relation between two sequences
  - E.g., question answering, information retrieval, natural language inference, paraphrasing, etc.

- During training, BERT also learns the relationships between two sequences using an additional binary classifier objective
  - The binary classifier take the output embedding of [CLS]
  - It predicts whether Sequence B is the actual sequence that proceeds Sequence A or a random sentence
  - This classifier is jointly optimized with the MLM objective

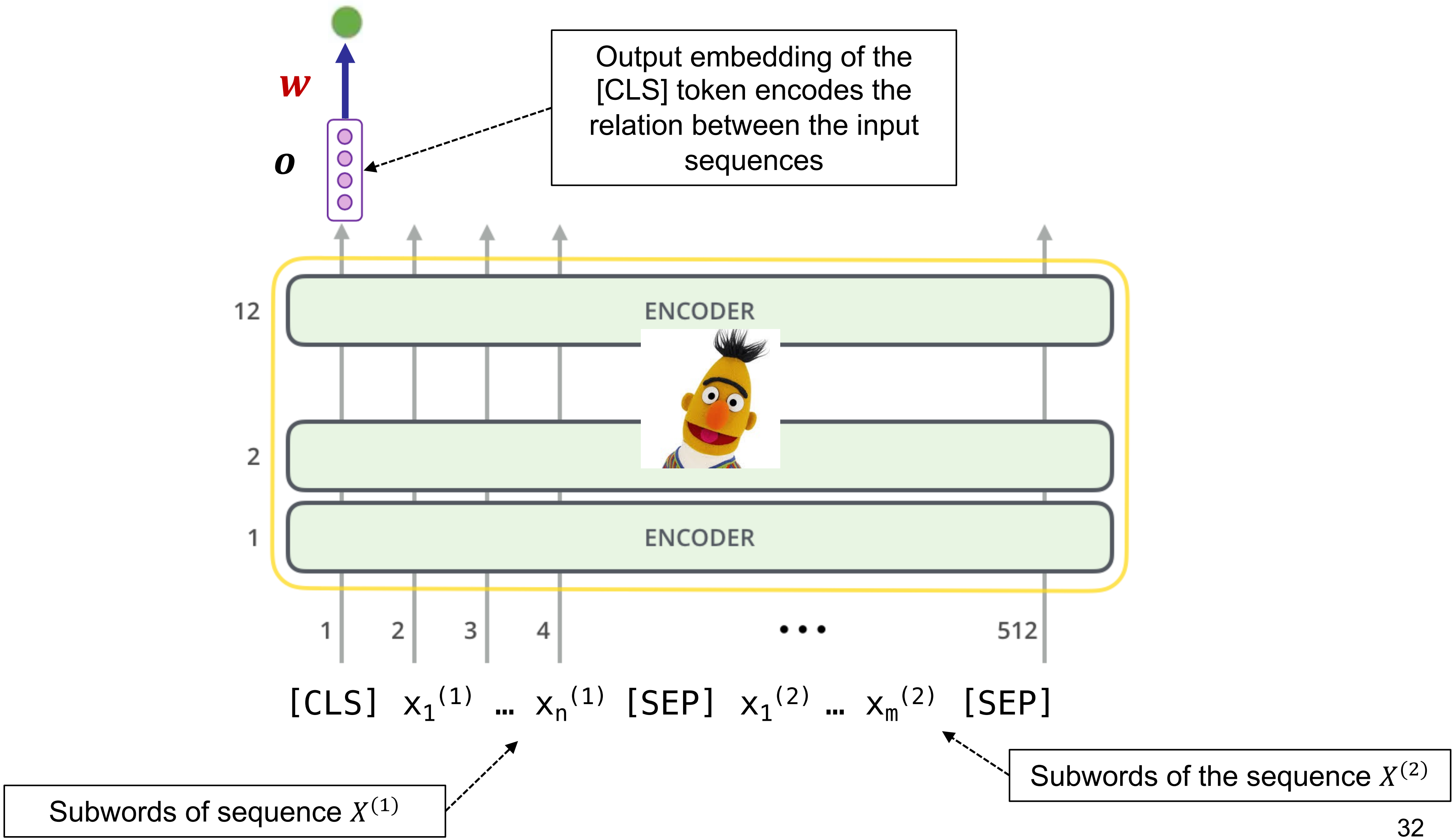
- If one sequence is given, the output of [CLS] is sequence embedding
- If two sequences are given, the output of [CLS] is the feature vector of the relation between the sequences

Input to BERT – two sequences

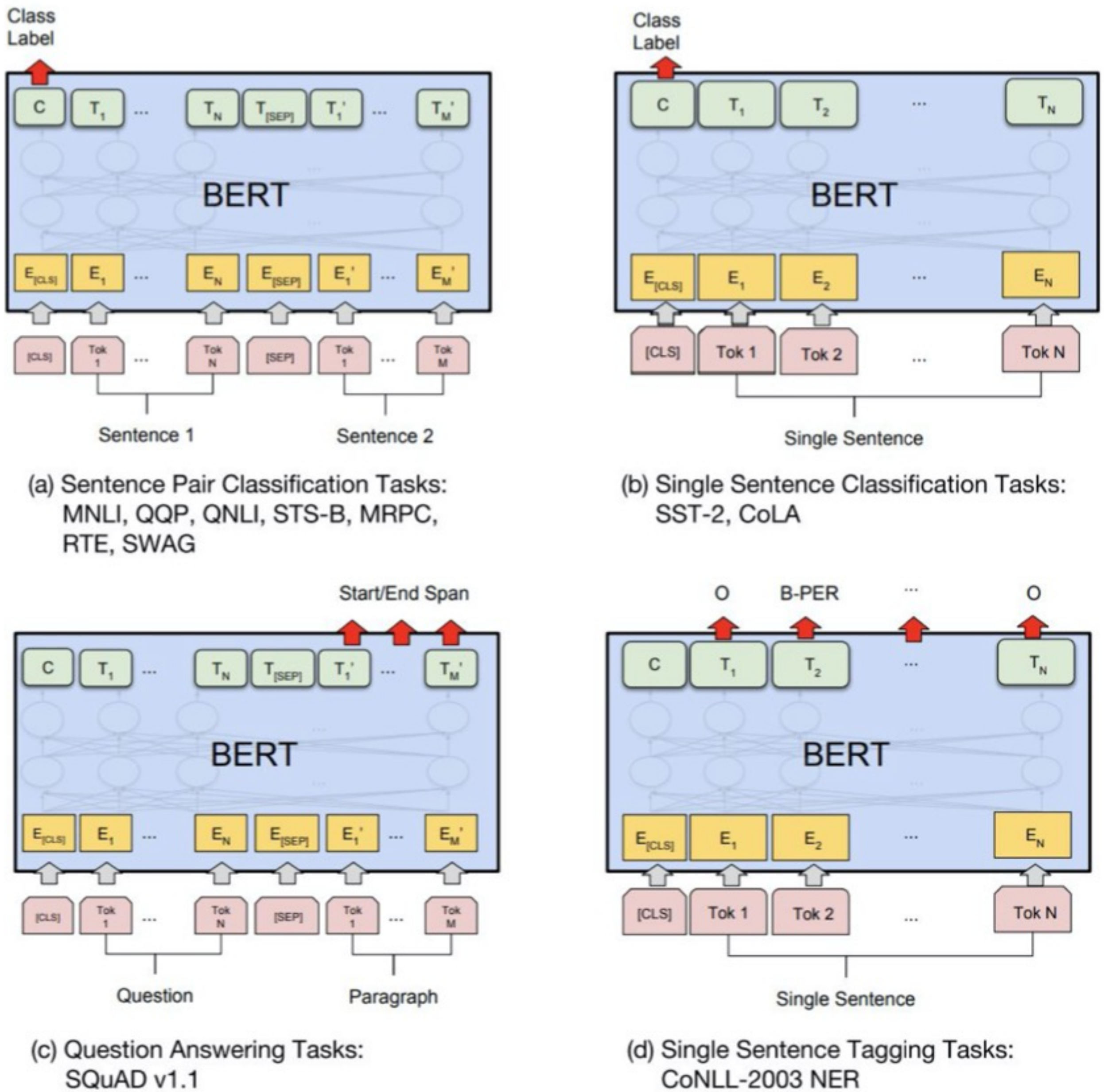


BERT Fine-tuning for Text Matching/Similarity tasks

$$\hat{y} = \text{similarity}\left(X^{(1)}, X^{(2)}\right) = \sigma(\boldsymbol{o}\boldsymbol{w})$$



Fine tuning – inputs in different scenarios



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Some evaluation results

- A generic, deep, pre-trained model that can simply be plugged in (almost) any NLP task!

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT$_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**