Natural Language and Speech Processing

Lecture 11: Advanced Topic in Neural NLP

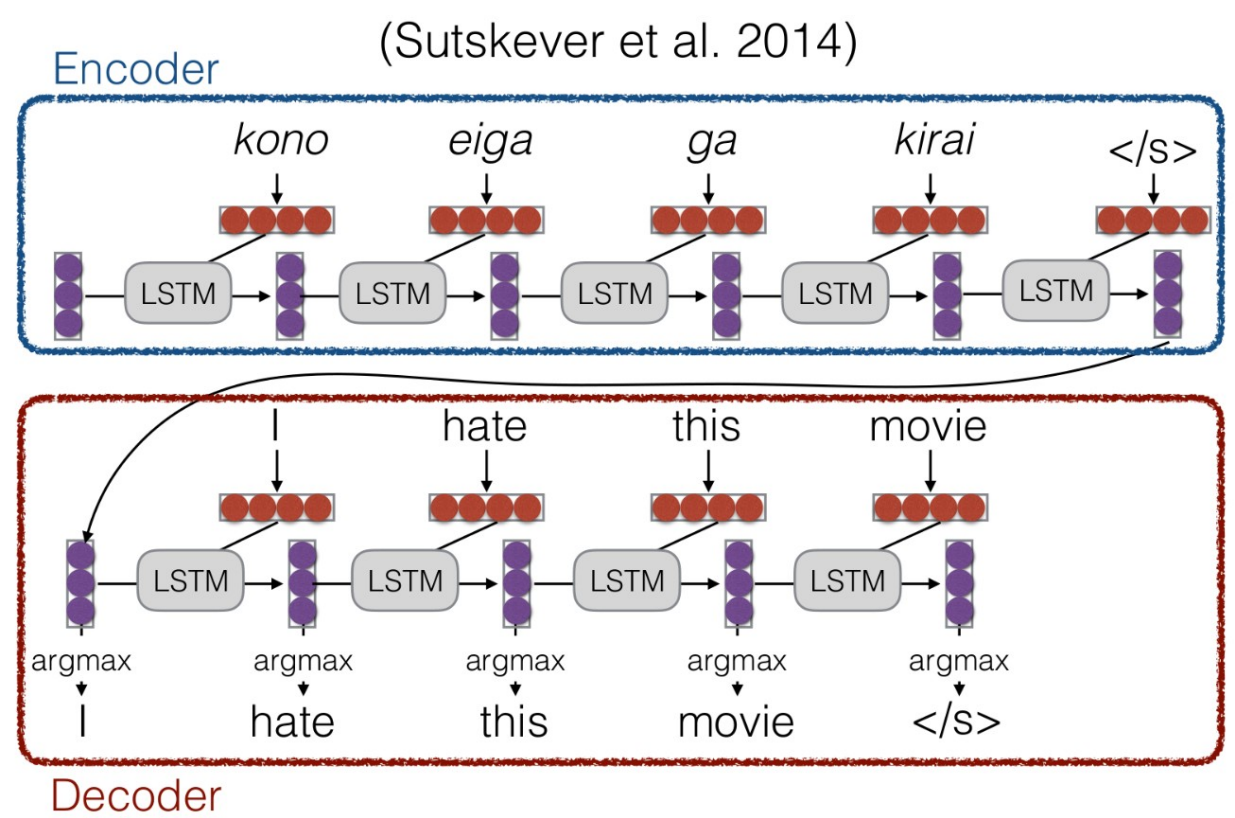
Tanel Alumäe

Contents

- Attention mechanism
- Transformer model
- BERT

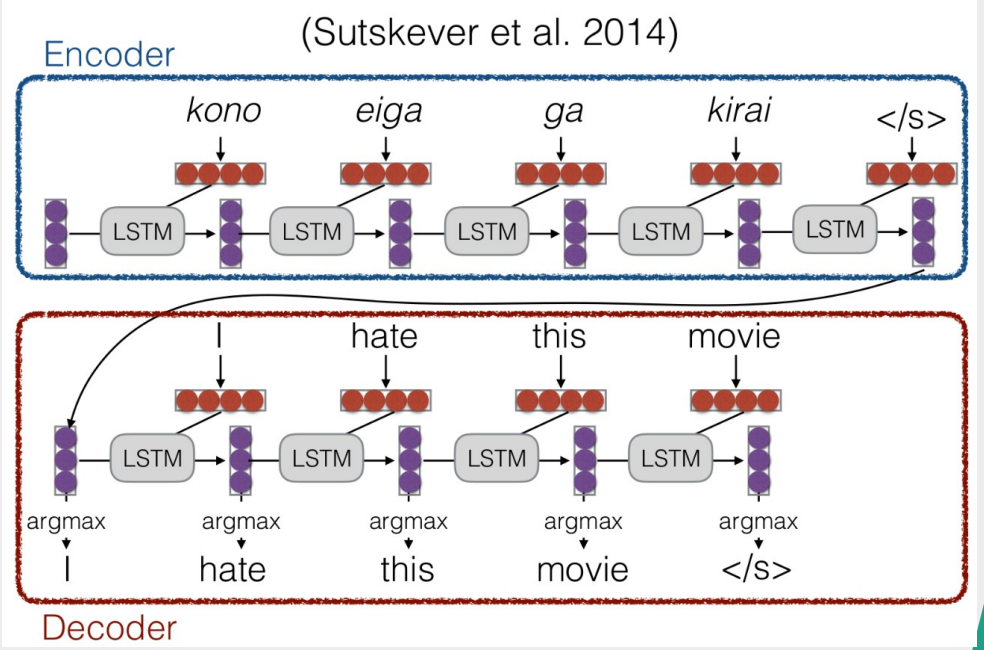
Machine translation

- Machine translation using the encoder-decoder neural architecture



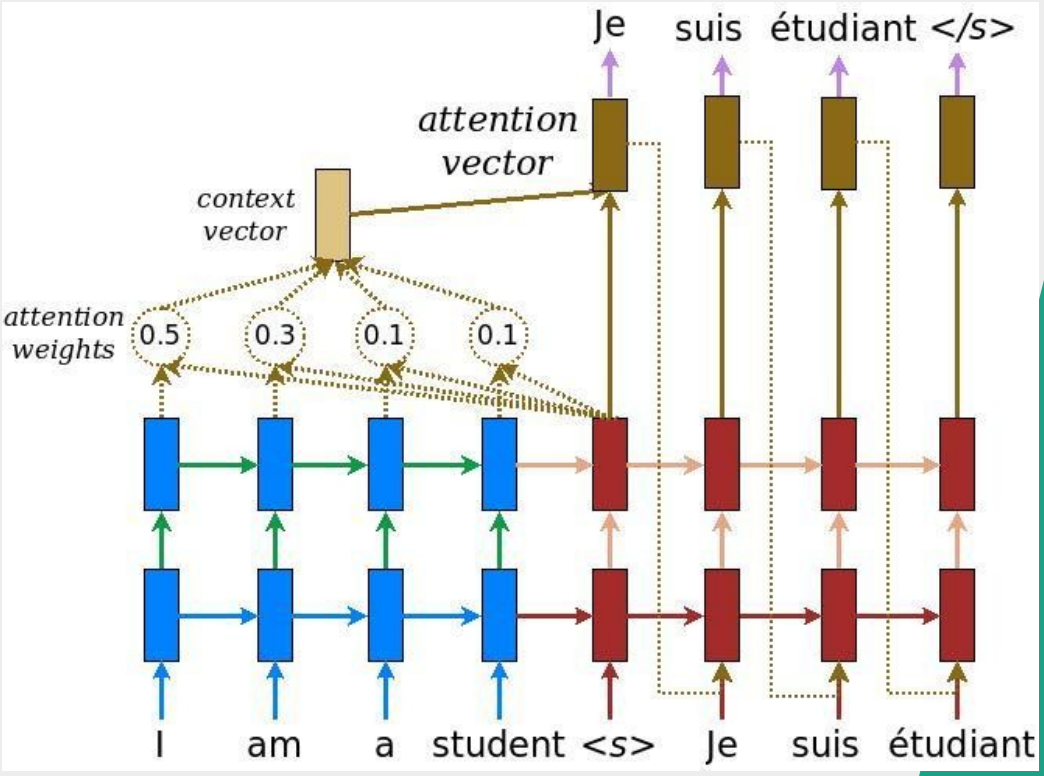
Problem with this architecture

- The whole source sentence is compressed into one vector
- The vector has fixed dimensions, regardless of the sentence
- “You can’t cram the meaning of a whole %&!$ing sentence into a single $&!*ing vector!” — Ray Mooney



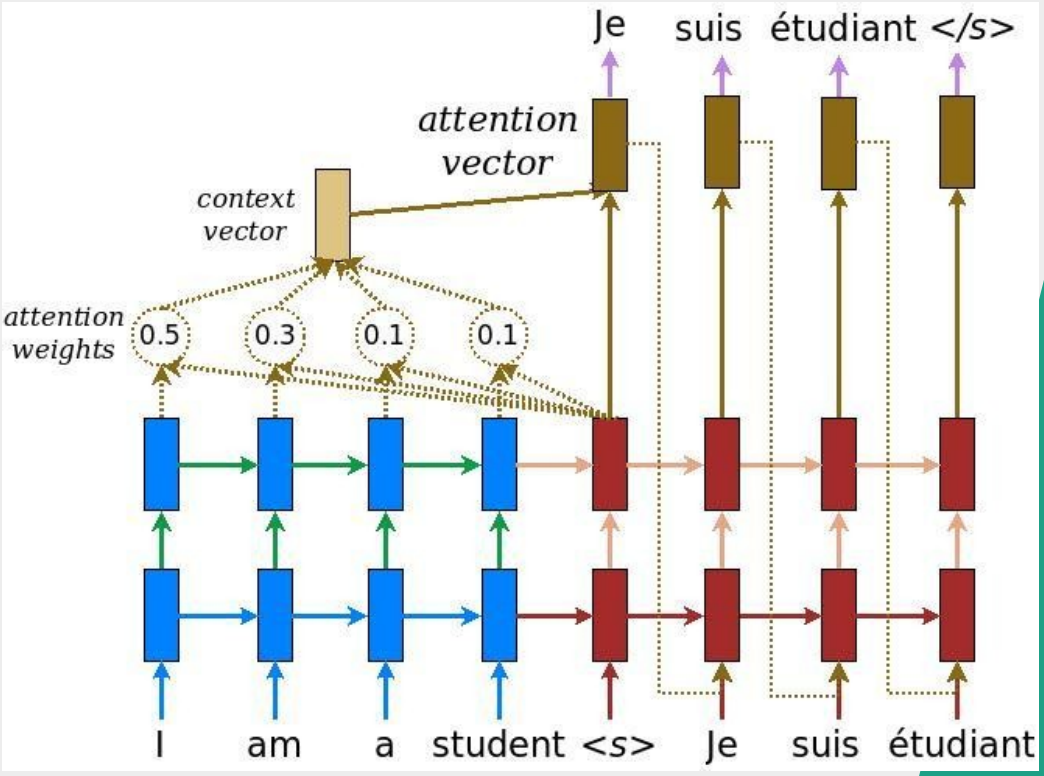
Attention

- Encode each word in the source sentence as a vector
- When generating the target sentence, perform a linear combination of those vectors to compute a **context vector**, using a different combination at each time step
- The context vector is used as addition input when generating the next word



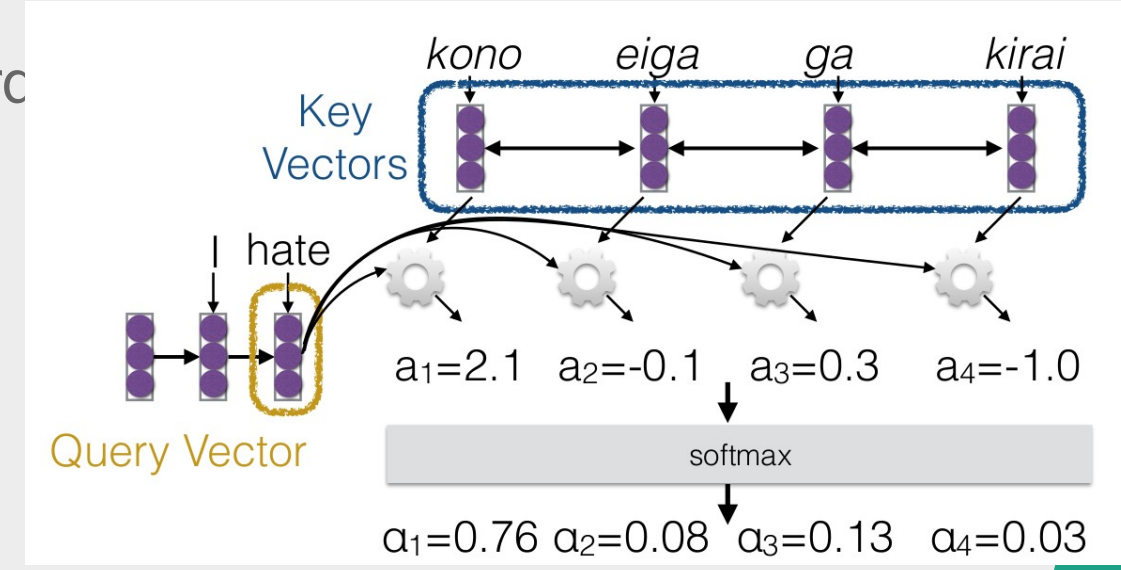
Why attention?

- This allows decoder to capture somewhat global information
- Rather than solely to generate based on one hidden state



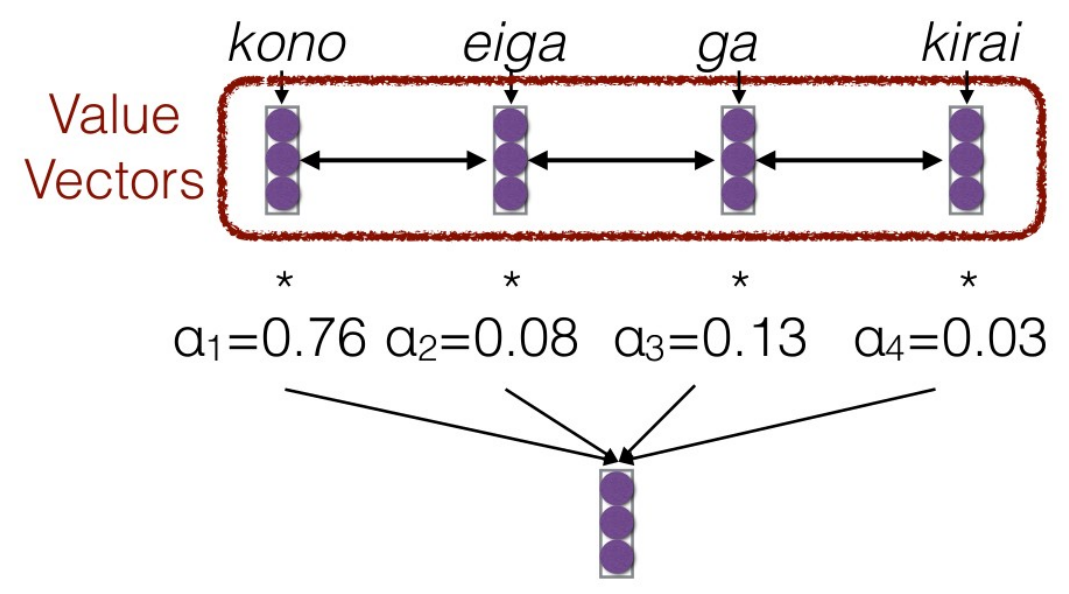
Calculating attention

- Decoder (generator) current hidden state vector is compared to all source word hidden states
- The comparison is done using cosine similarity or some learned function
- Each comparison results in some value
- The values are normalized using softmax, so that they sum to 1 and all are positive



Calculating attention, II

- The source word hidden states are now summed, using the weights
- The resulting vector can be used as addition input in the generator model (maybe after applying a nonlinearity, such as ReLU o rtanh)



Attention score function

- Attention score function is either fixed or trainable
- q is the current generator hidden state and k is the source word hidden state

- Dot product:

$$a(q,k)=q^T k$$

- Scaled dot product (scale by the size of the vector)

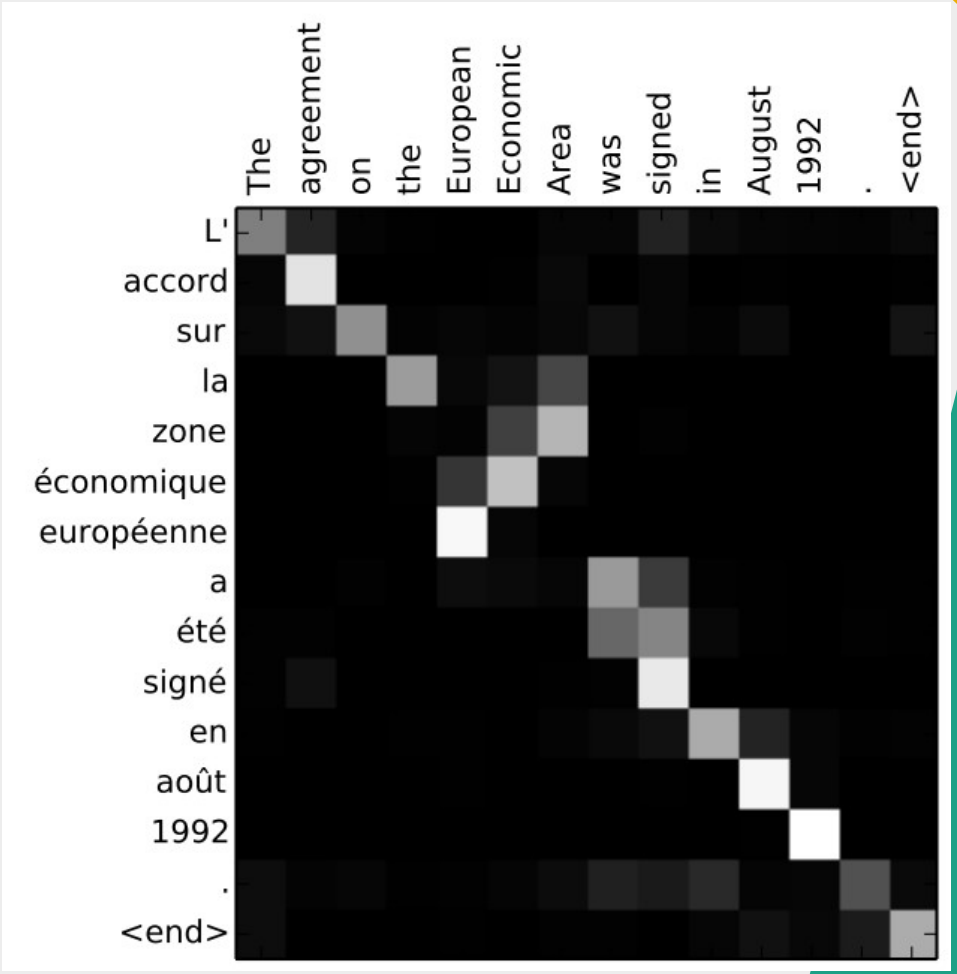
$$a(q,k)=\frac{q^T k}{\sqrt{|k|}}$$

- Small learned neural network

$$a(q,k)=\tanh(W[q;k])$$

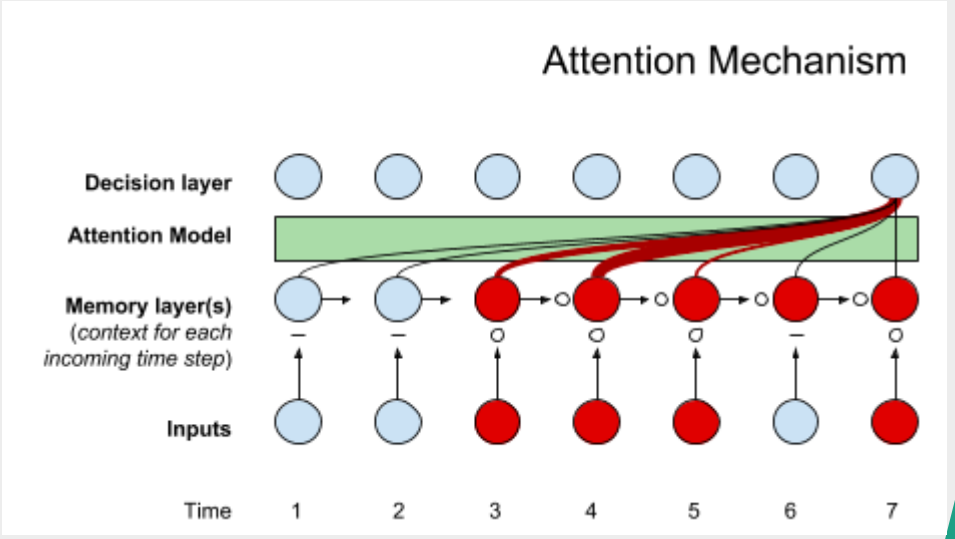
Visualizing attention

- We can check which words in the source sentence had highest weights for each target word
- In other words, which words were **attended to**
- You can see how the model paid attention correctly when outputing "European Economic Area"
- In French, the order of these words is reversed ("européenne économique zone") as compared to English



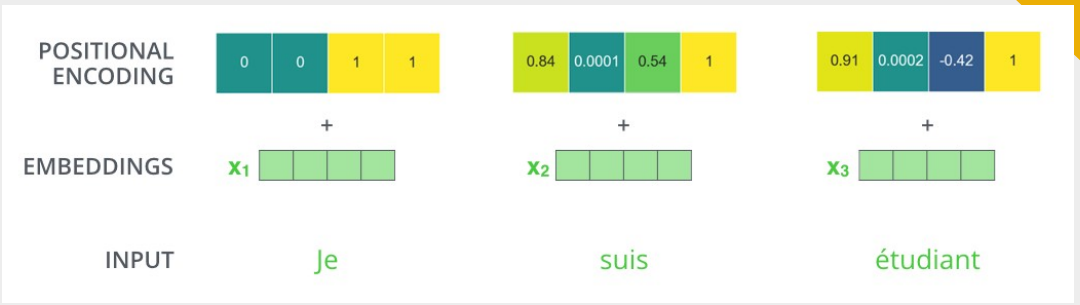
Self-attention

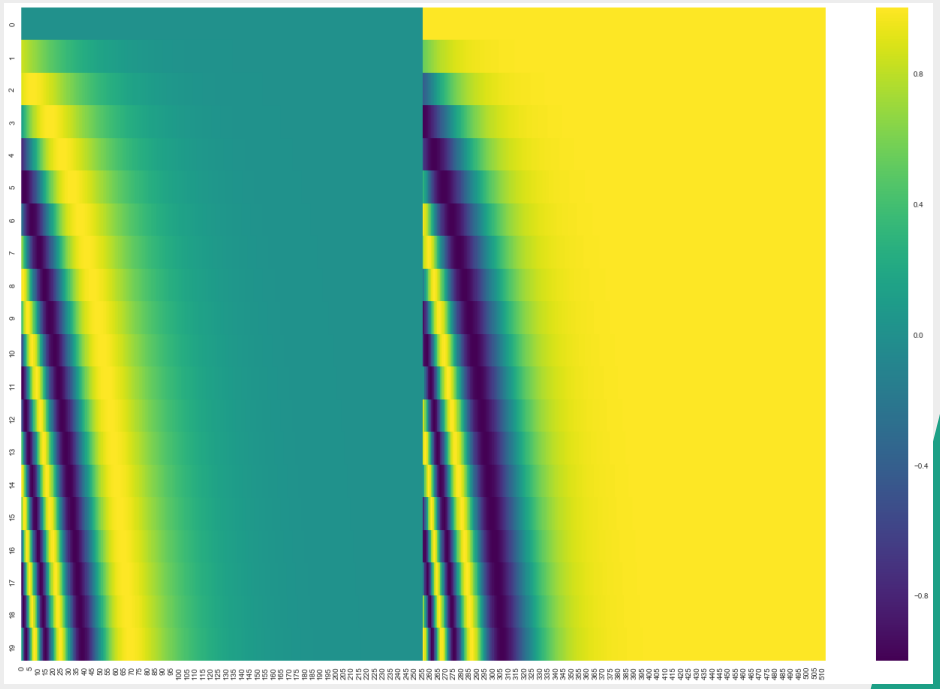
- Attention can be also used in other models besides encode-decoder architectures
- E.g., we can „attend” to previosly generated words when generating a new word
- Or we can attend to other words, when encoding a word in the encoder



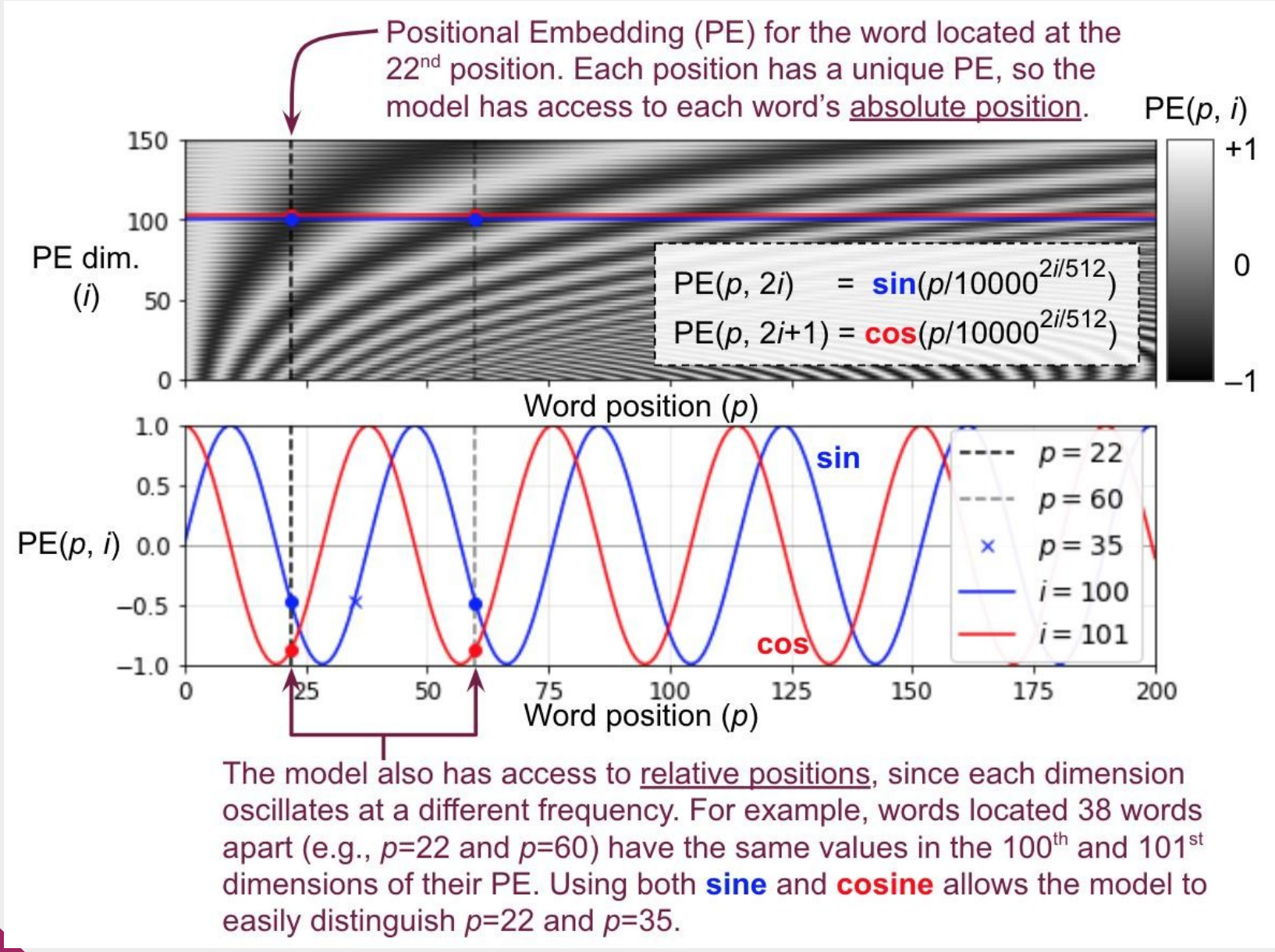
Positional encoding

- Problem: when computing the attention vector, word position in the sentence gets lost
- Solution: add information about word's absolute position to its embedding



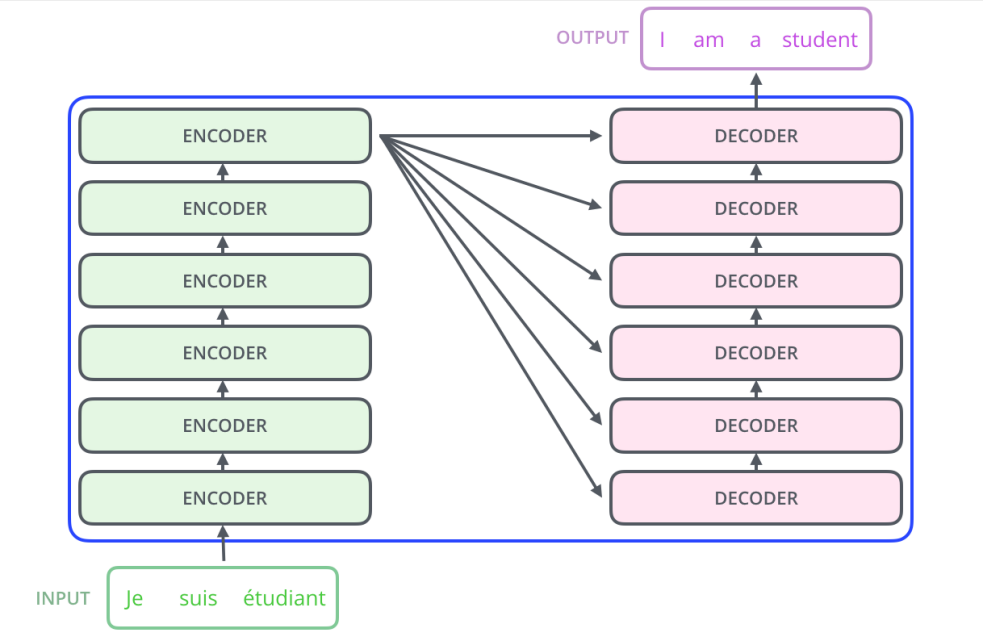


Positional encodings, cont.



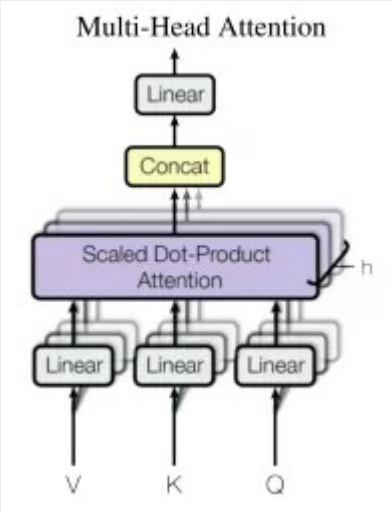
Attention is all you need!

- Now, self attention can be used to encode the whole history of the context into one vector
- This is similar to a recurrent neural network
- Attention is computationally more efficient, and words far in the history have a more fair chance of impacting the context
- Attention can also learn to mitigate the word relative positions using thje position encoding
- The encoder-decoder architecture with no recurrent layers, where only attention is used to encode the context, is called a **Transformer**



Multi-head attention

- Attention mechanism is used as a way for the model to focus on relevant information based on what it is currently processing
- It is difficult to capture various different aspects of the input, using a single attention weighted sum
- To solve this problem the Transformer uses the **Multi-Head Attention** block
- This block computes multiple attention weighted sums instead of a single attention pass over the values



The Transformer

- Transformer still uses the basic encoder-decoder design
- The initial inputs to the encoder are the embeddings of the input sequence,
- The initial inputs to the decoder are the embeddings of the outputs up to that point
- State-of-the-art results for many NLP tasks

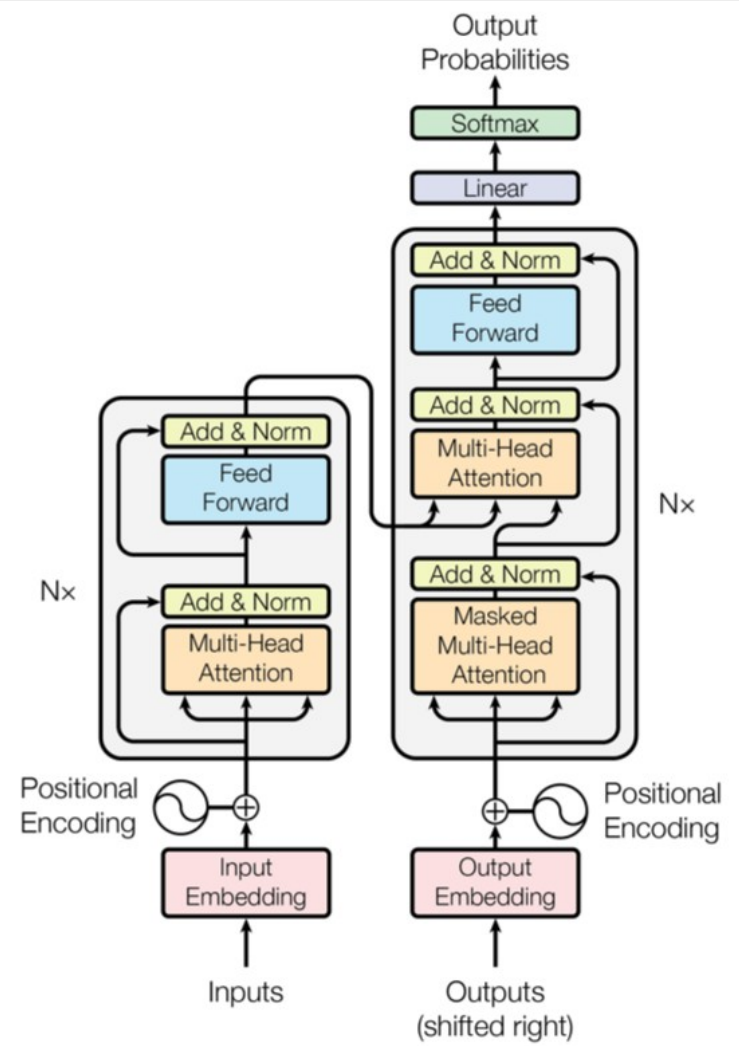


Figure 1: The Transformer - model architecture.

BERT

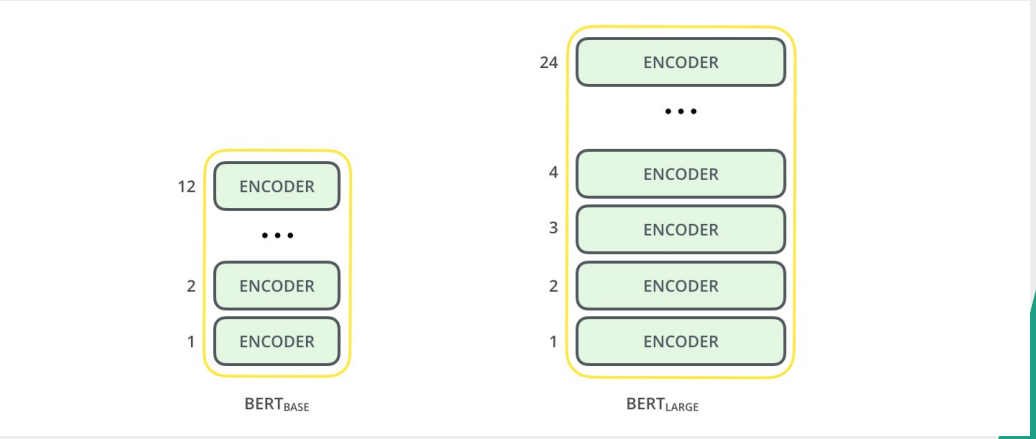
- BERT (Bidirectional Encoder Representations from Transformers, released in 2018) is a model that broke several records for how well models can handle language-based tasks
- BERT is a huge pretrained model for English (a multilingual version is also available)
- BERT can be used to generate contextual word embeddings
- Or to compute something based on a pair of texts

BERT applications

- Text classification
- Word classification
- Natural language understanding
  - Given: paragraph, question
  - Output: answer to the question based on the paragraph, or no answer if not available
- Almost any NLP task (in English)

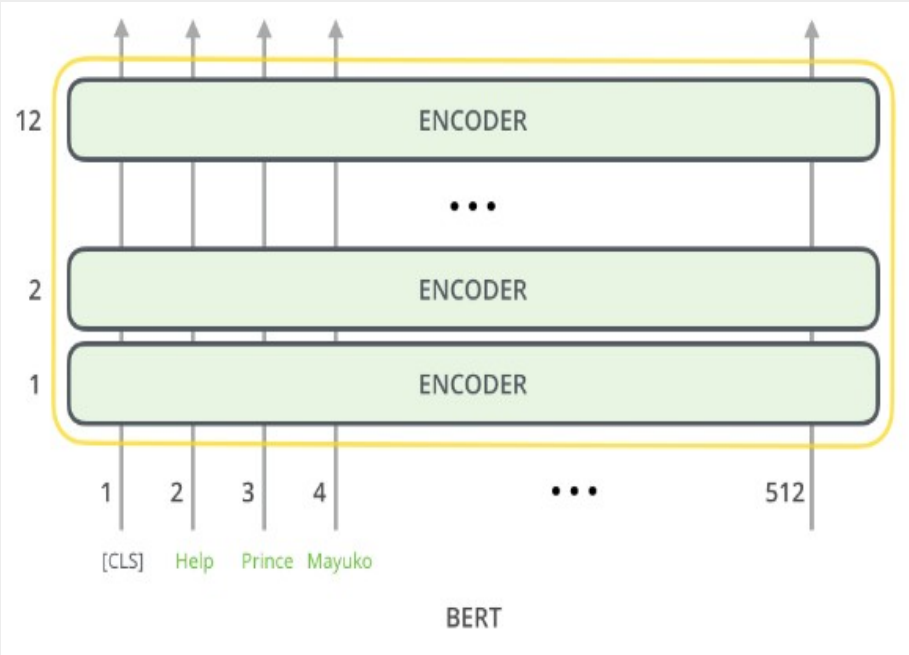
BERT architecture

- BERT is basically a trained Transformer Encoder stack
- BERT-base: 12 layers, 768-unit hidden layers
- BERT-large: 20 layers, 1024-unit hidden layers
- Each layer is a self-attention layer



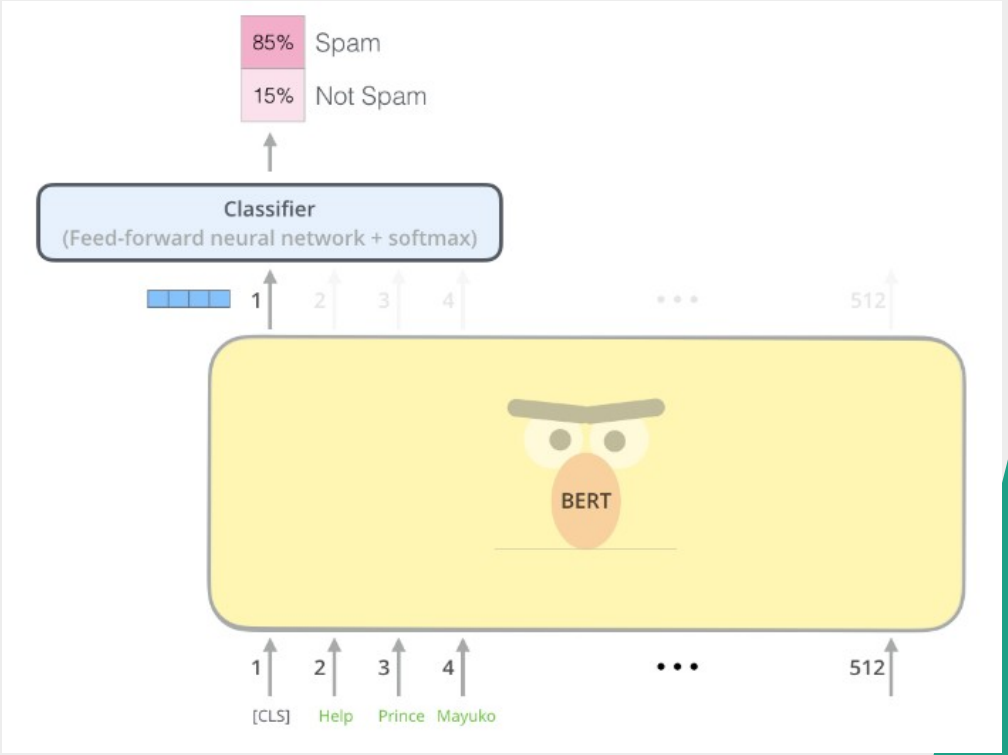
BERT model inputs

- The first input token is supplied with a special [CLS] token
- BERT takes a sequence of words as input which keep flowing up the stack
- Each layer applies self-attention, and passes its results through a feed-forward network, and then hands it off to the next encoder



BERT model outputs

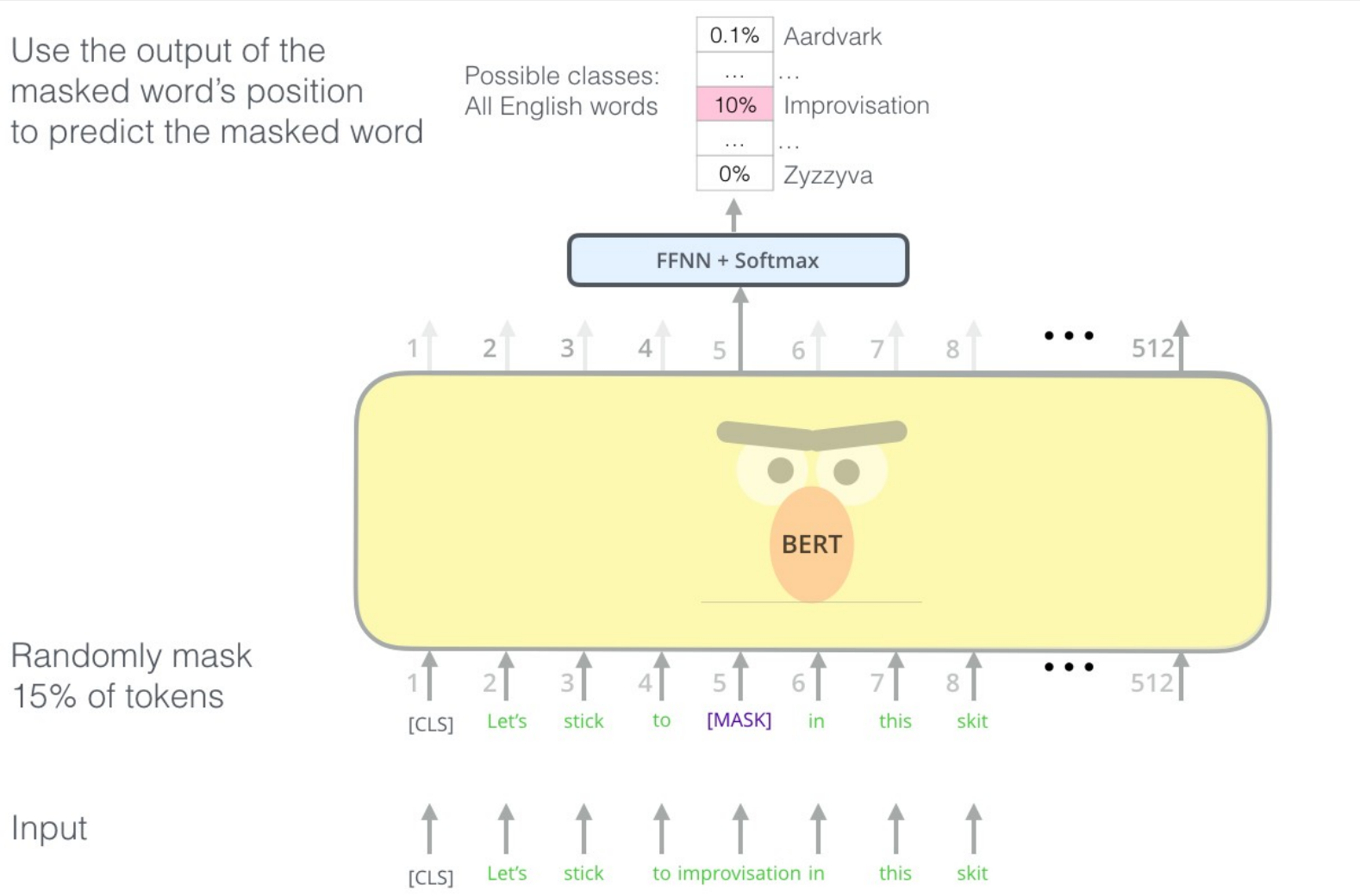
- Each position outputs a vector of size hidden_size (768 in BERT Base)
- For the sentence classification example we've looked at above, we focus on the output of only the first position (that we passed the special [CLS] token to)



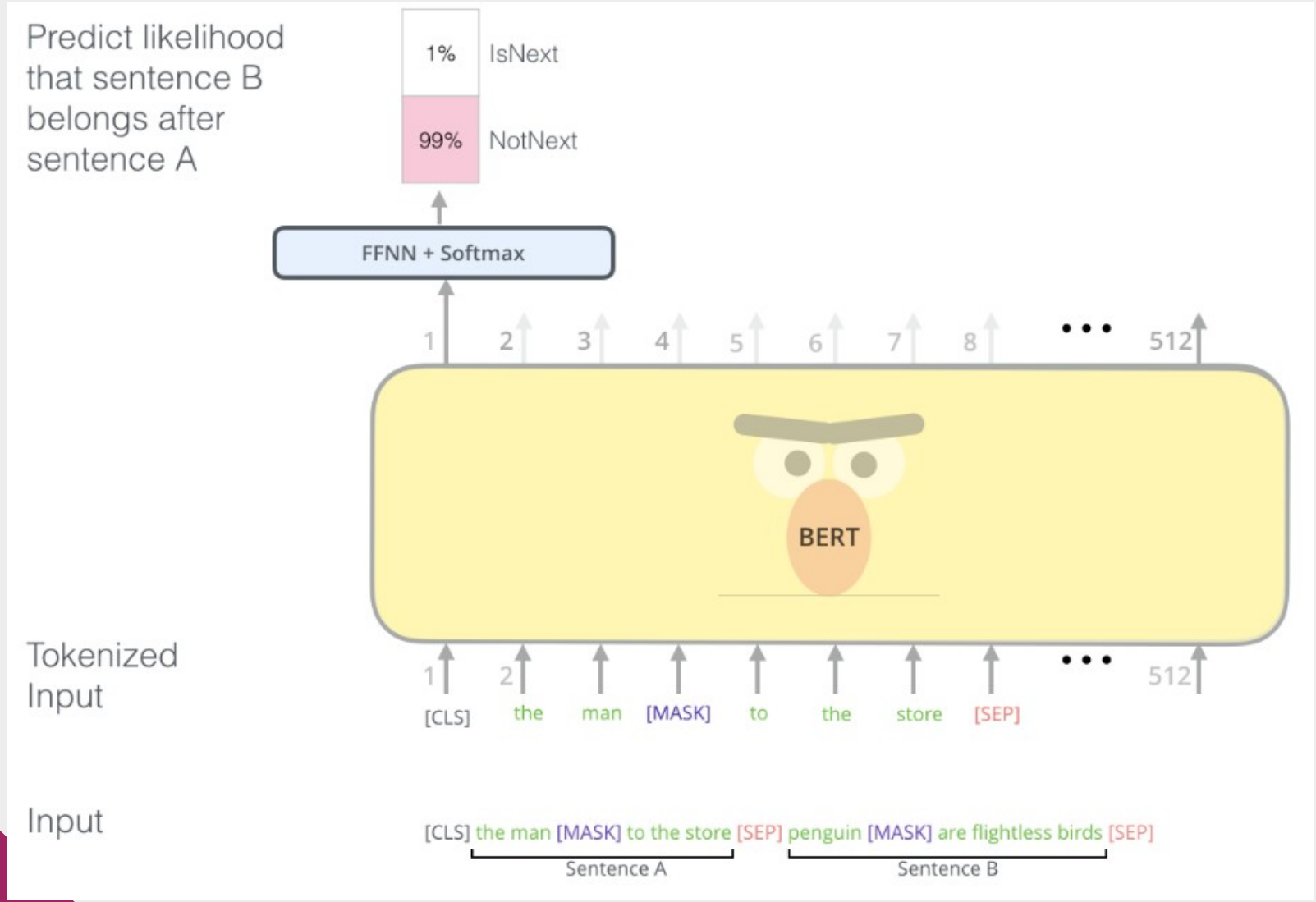
How BERT is trained

- BERT is trained using two tasks
- I.e., the model has two outputs, it shares most of the layers for both tasks, and training is done intermixed
- One training batch otpimizes one task and the next one the other task
- Model learns to optimize itself for both tasks, and thus learns regularities that are universal for both tasks
- This is called multitask learning

BERT training: masked LM



BERT training: does sentence A follow B?

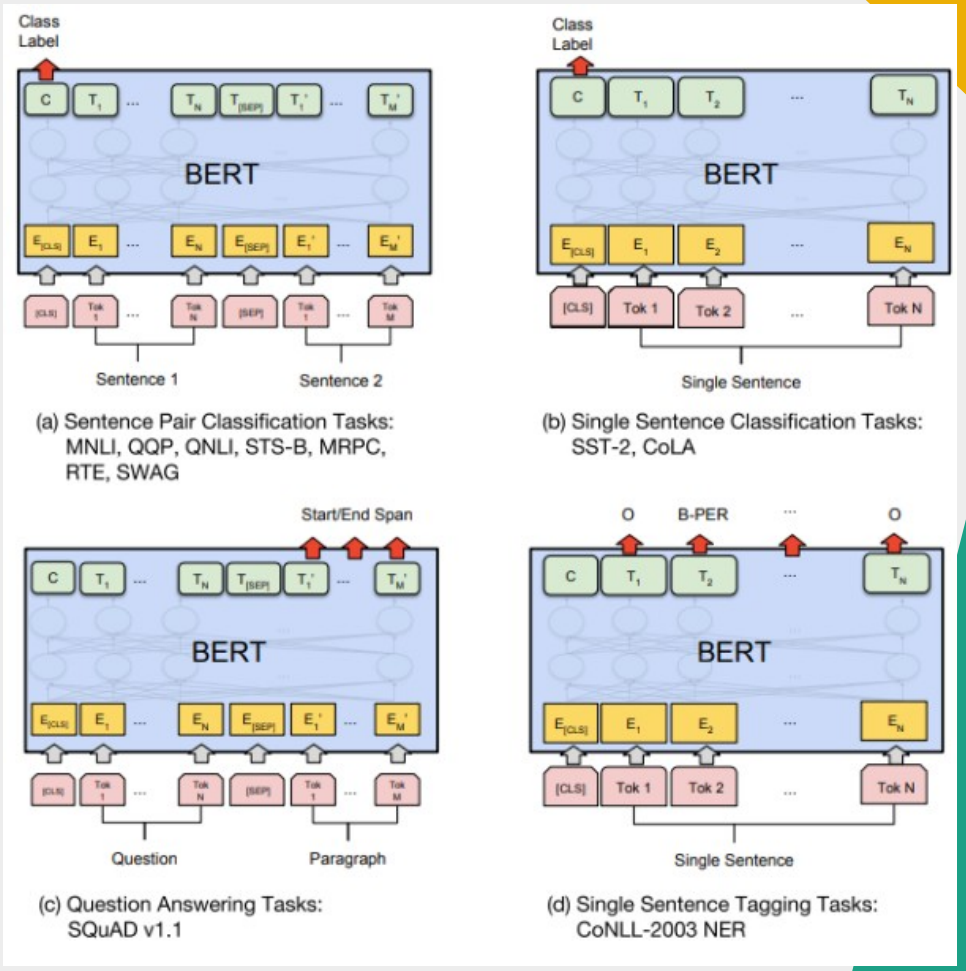


BERT training

- BERT is trained on Wikipedia and BooksCorpus
- BERT uses so-called WordPieces to tokenize input (to avoid having any out-of vocabulary words)
  - Common words are left as is but rare words are split into pieces
  - Original: `The chips from his wood pile refused to kindle a fire to dry his bed-clothes, and he had recourse to a more provident neighbor's to supply the deficiency.`
  - WordPieces: `['the', 'chips', 'from', 'his', 'wood', 'pile', 'refused', 'to', 'kind', '##le', 'a', 'fire', 'to', 'dry', 'his', 'bed', '-', 'clothes', ',', 'and', 'he', 'had', 'rec', '##ours', '##e', 'to', 'a', 'more', 'provide', '##nt', 'neighbor', "'", 's', 'to', 'supply', 'the', 'deficiency', '.']`

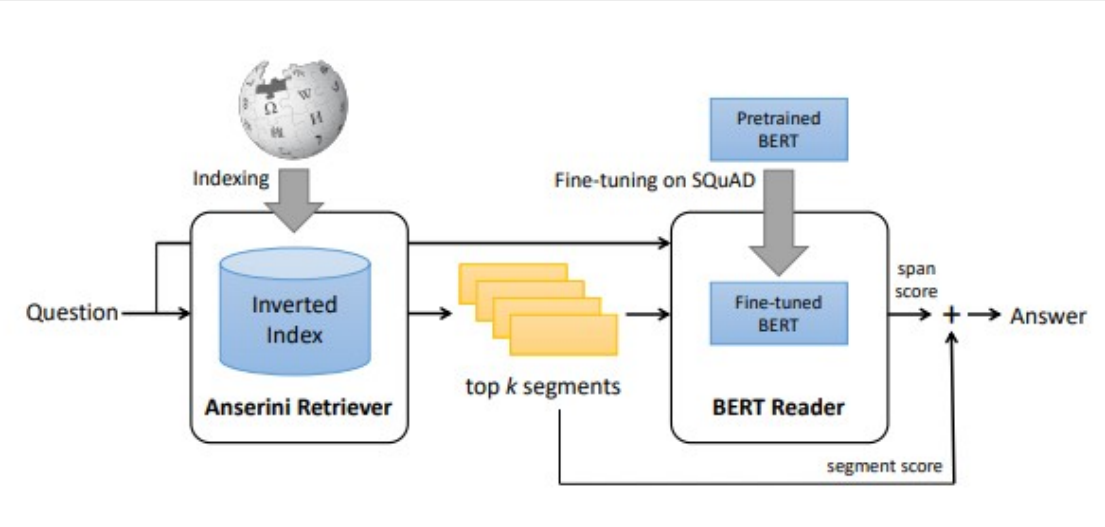
How to use BERT

- Use encoded words as is or finetune the pretrained model using your data
- Text classification: use the output for CLS tag, apply a single layer NN on top
- Word classification: use BERT word encodings as input to a sliding neural network
- Sentence similarity: give both sentences as input (seperated by [SEP]), and use the output of the CLS tag for classification
- Question answering: give question and paragraph as input (seperated by [SEP]), outputs corresponding to paragraph words are classified as „is this word part of the answer”



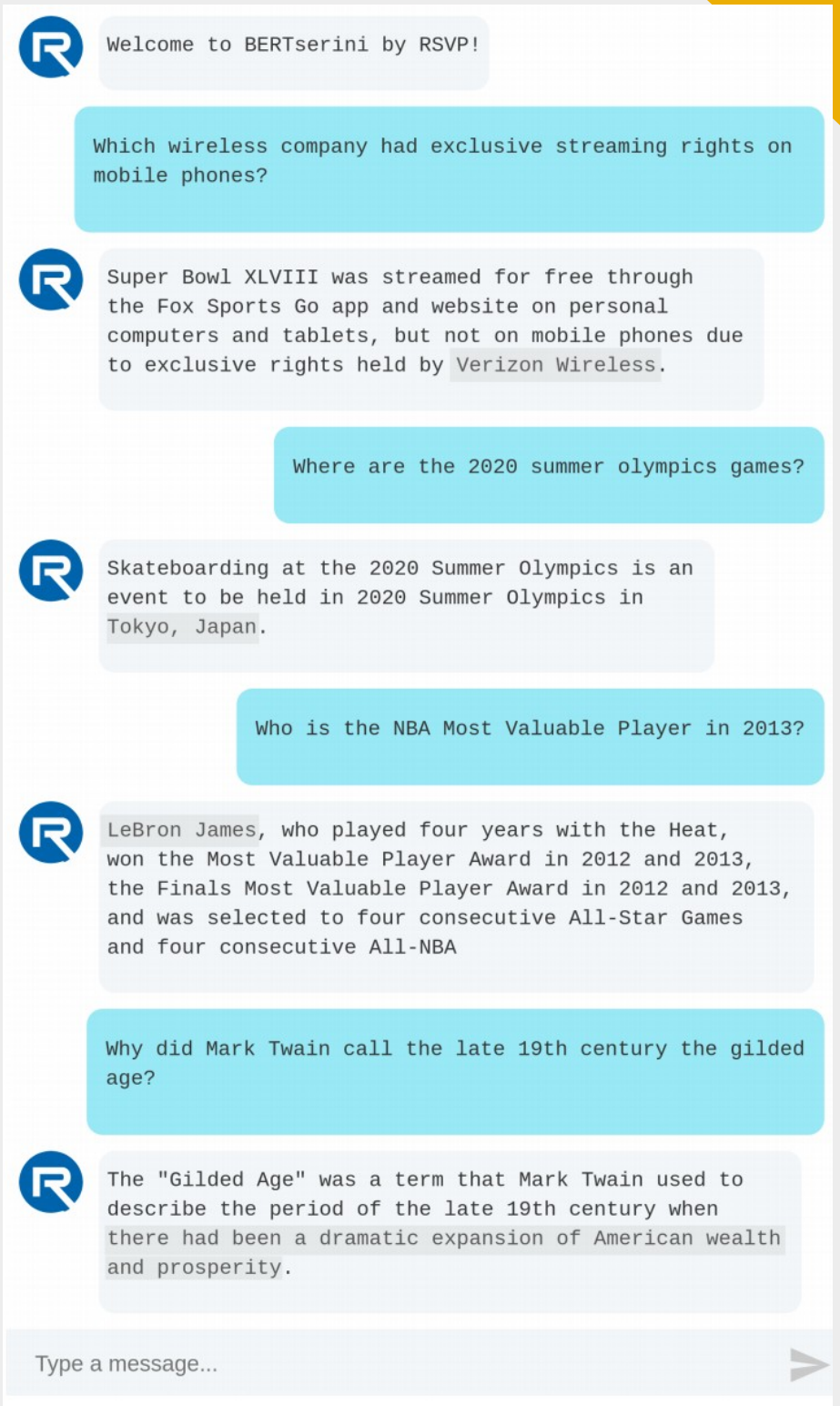
Open-domain question answering using BERT

- Yang et al, *End-to-End Open-Domain Question Answering with BERTserini*, 2019



- Method:
  - Retrieve top *k* paragraphs from Wikipedia using some simple method (e.g., bag-of-words based)
  - Use BERT (fine-tuned on a Question Answering corpus) to find most likely answer (or no answer)

- BERTserini in action



Use BERT

- Pretrained BERT is available for download, and it is relatively easy to use it for almost any NLP task
- If your data is in English, you should consider using it
  - However, it’s slow
- Probably new and better models will be available soon