Introduction to Transformers

Michel RIVEILL
michel.riveill@univ-cotedazur.fr

General word/sentences representations

- Feature-based approaches (or static approach)
  - Non-neural word representations (BOW)
  - Neural embedding
    - Word embedding:
      - Word2Vec, Glove,...
    - Sentence embedding or Paragraph embedding,...
- Embeddings from Language Models
  - Replace static embeddings (lexicon lookup) with context-dependent embeddings (produced by a deep neural language model)
  - Each token's representation is a function of the entire input sentence, computed by a deep (multi-layer) bidirectional language model
  - Return for each token a (task-dependent) linear combination of its representation across layers.
  - Different layers capture different information

Embeddings from Language Models

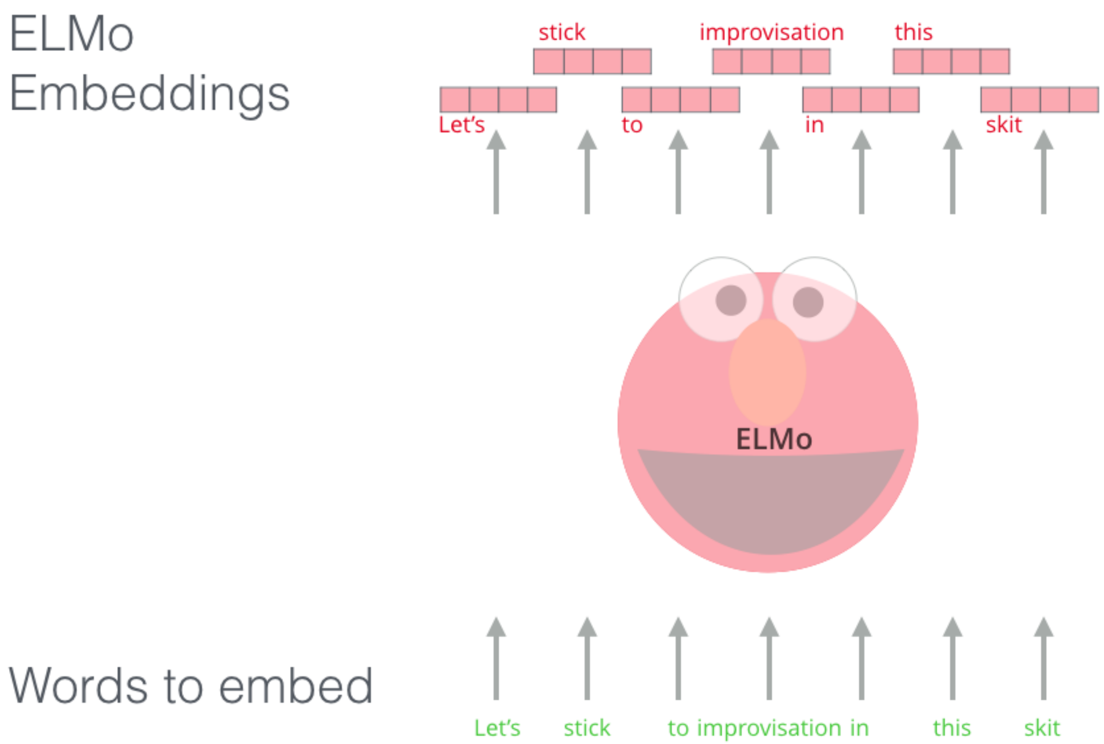
- Deep contextualised word representation
  - ELMo, Embeddings from Language Models, Peters et al., 2018

- Fine-tuning approaches
  - GPT
    - Generative Pre-trained Transformer, Radford et al., 2018
  - BERT
    - Bi-directional Encoder Representations from Transformers, Devlin et al., 2018

Elmo

ELMo: deep contextualised word representation (Peters et al., 2018)

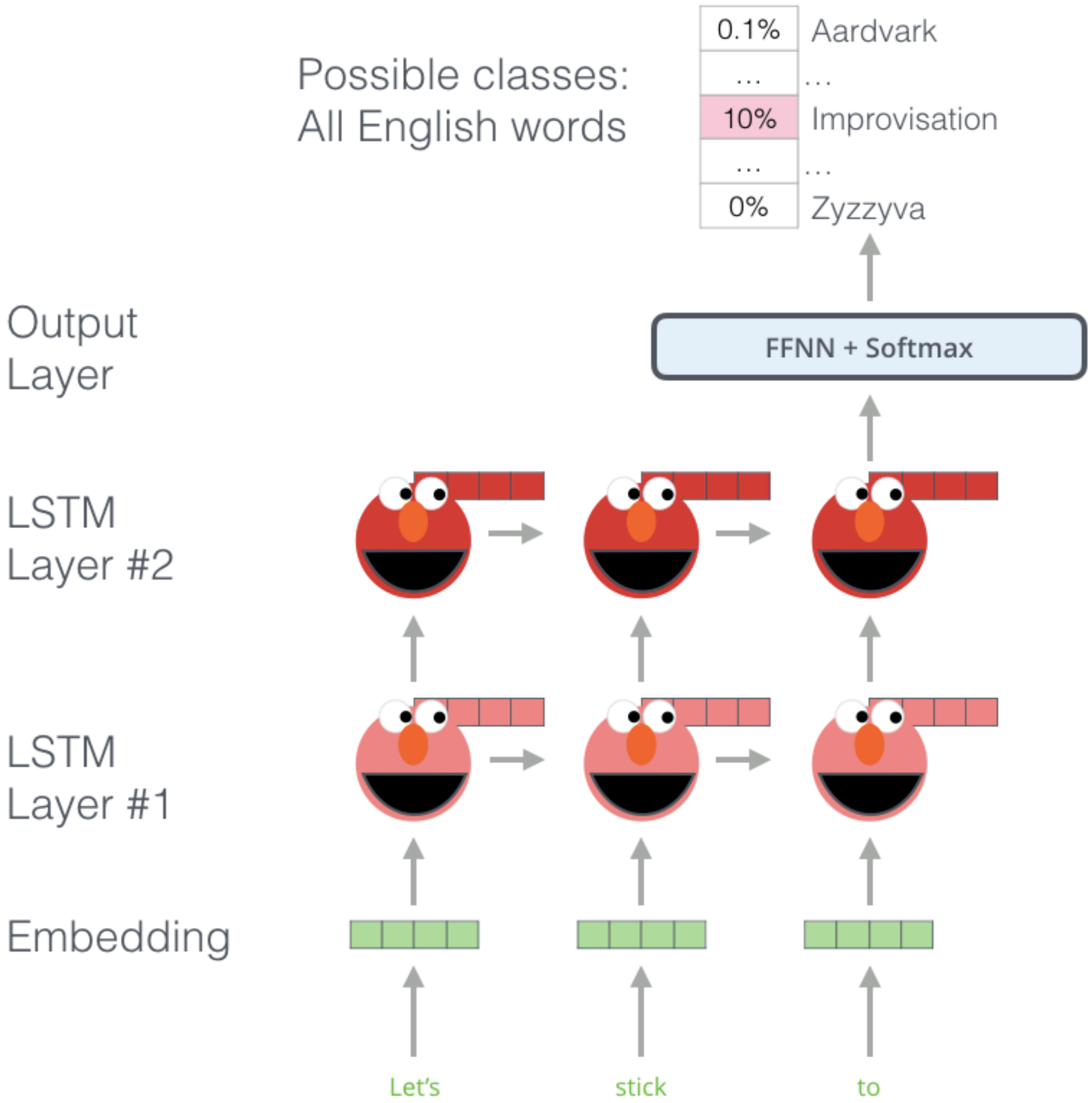
- "Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning each word in it an embedding."



ELMo's secret

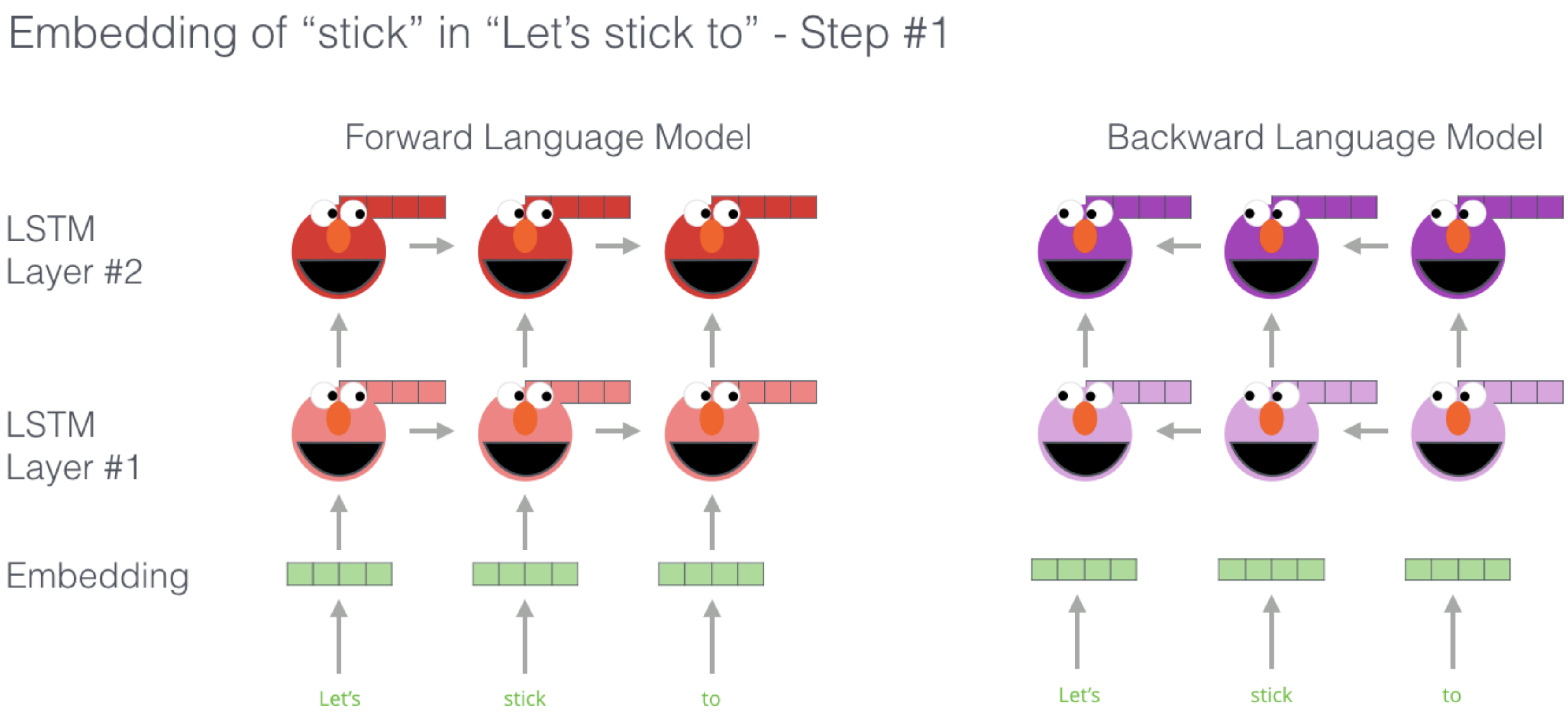
ELMo was trained in an unsupervised manner, like word2vec:
- predicting the next word in a sequence of words - a task called "language modeling".

This is useful because there is a large amount of unlabeled text data available.



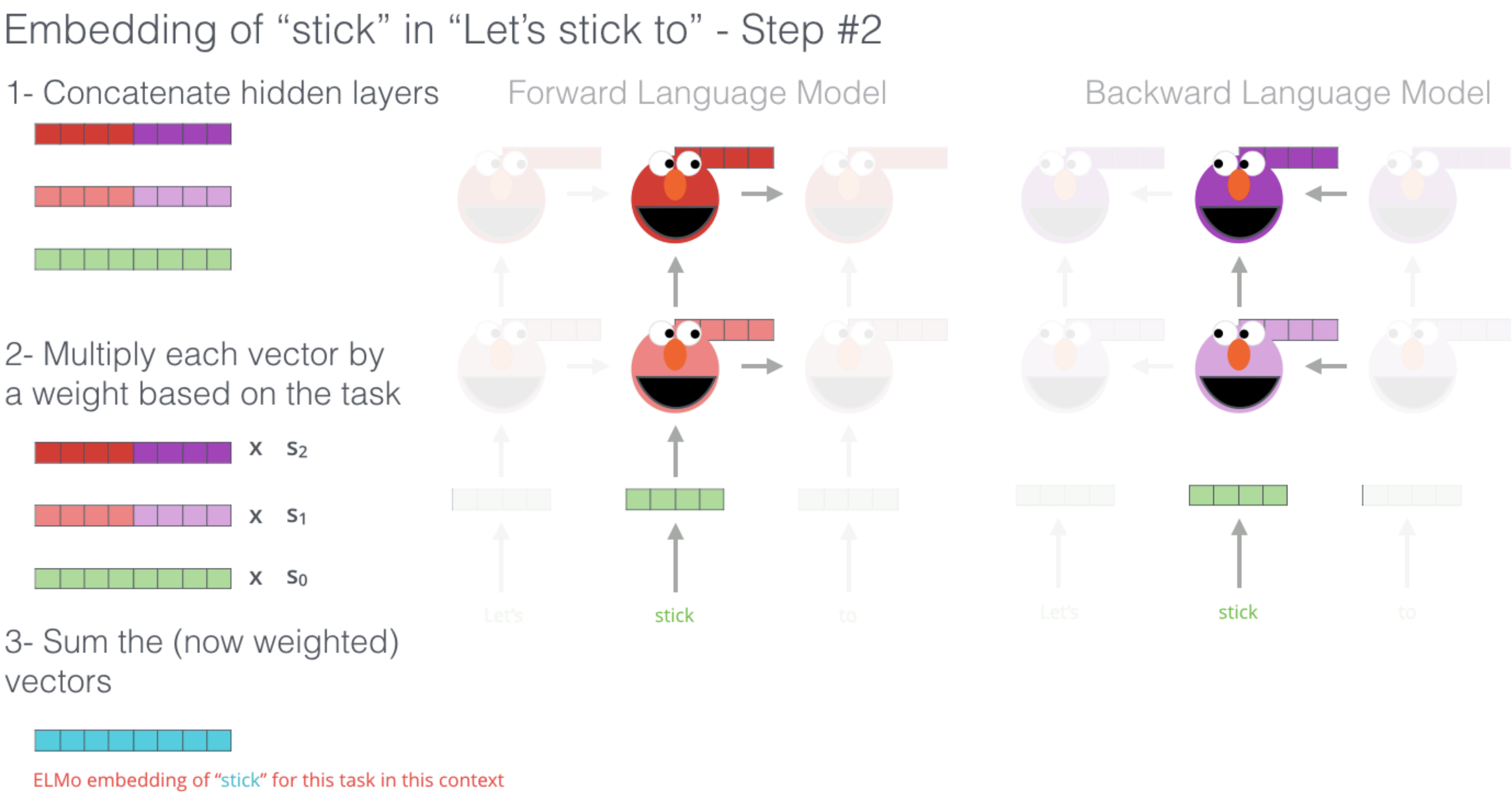
ELMo’s secret

- In practice ELMO uses a bi-directional MSTL.
- His linguistic model tries to capture the relationships within a sentence in both directions.



ELMo's secret

ELMo proposes a contextualized embedding by grouping hidden states (and initial integration) in a certain way (concatenation followed by a weighted summation).



ELMO architecture

- Input representation
  - input token representations are purely character-based: a character CNN, followed by linear projection to reduce dimensionality
  - "2048 characters n-gram convolutional filters with two highway layers, followed by a linear projection to 512 dimensions"
  - Advantage over using fixed embeddings: no UNK tokens, any word can be represented
- Train a multi-layer bidirectional language model with character convolutions on raw text
  - The forward LM is a deep LSTM that goes over the sequence from start to end to predict token t_k based on the prefix $t_1 \ldots t_{k-1}$
  - The backward LM is a deep LSTM that goes over the sequence from end to start to predict token t_k based on the suffix $t_{k+1} \ldots t_N$
- Each layer of this language model network computes a vector representation for each token
  - Train these LMs jointly, with the same parameters for the token representations and the softmax layer (but not for the LSTMs)
- Freeze the parameters of the language model
- For each task: train task-dependent softmax weights to combine
  - the layer-wise representations into a single vector for each token
  - *jointly* with a task- specific model that uses those vectors

How ELMo different from other word embeddings ?

- Suppose we have a couple of sentences:
  - I ***read*** the book yesterday.
  - Can you ***read*** the letter now?
- **Polysemy :** a word have multiple meanings or senses
  - "read" in the first sentence is in the past tense.
  - "read" in the second sentence is in the present tense

- Embedding of word read
  - With Keras embedding, word2vec, glove:
    - Read have always the same embedding
  - With ELMo:
    - Read have a contextualized embedding

What we can do with ELMo?

- ELMo allows to build an embedding for a list of sentences
  - It is then possible to couple this embedding with another LogisticRegression model, MLP, etc. for sentiment analysis tasks.
- It is of course possible to couple ELMO with other neural network-based models to perform more complex tasks.
  - Machine Translation
  - Language Modeling
  - Text Summarization
  - Named Entity Recognition
  - Question-Answering Systems
- It is possible to use embedding as or on the contrary, decide to make fine grained tuning

How to embed sentences with ELMo https://allennlp.org/elmo

- See notebook ELMO
  - Need the installation of tensorflow_hub

- elmo = hub.KerasLayer("https://tfhub.dev/google/elmo/2",
  trainable=False)

- text = ["A long sentence.",
  "Single-word",
  "URL http://example.com"]

- embeddings = elmo(tf.convert_to_tensor(np.asarray(text)))
- embeddings.shape

TensorShape([3, 1024])

- Reuse the embedding for another task

Transformers

Transformers

Originally

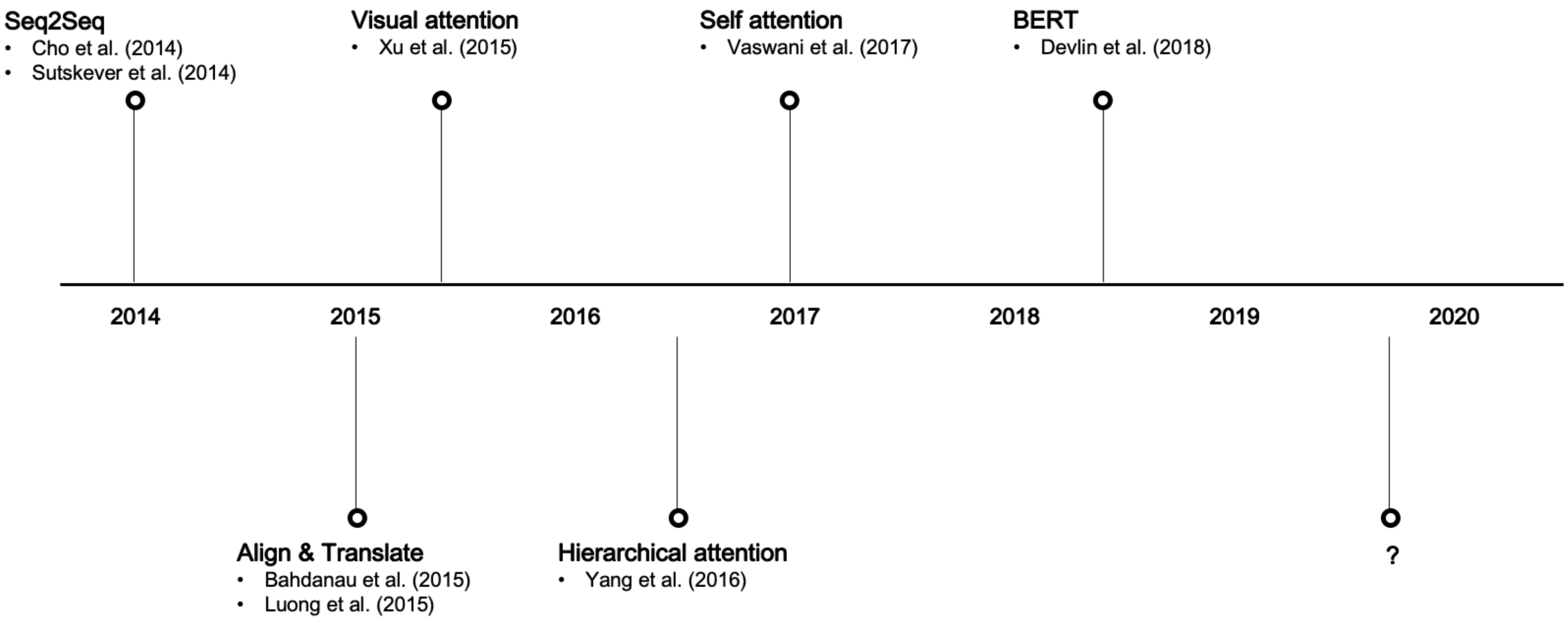
- Sequence transduction model based on attention
  - no convolutions or recurrence
  - easier to parallelize than recurrent nets
  - faster to train than recurrent nets
  - captures more long-range dependencies than CNNs (Convolutional Neural Nework) with fewer parameters

Now

- Transformers use stacked self-attention and pointwise, fully-connected layers for the encoder and decoder

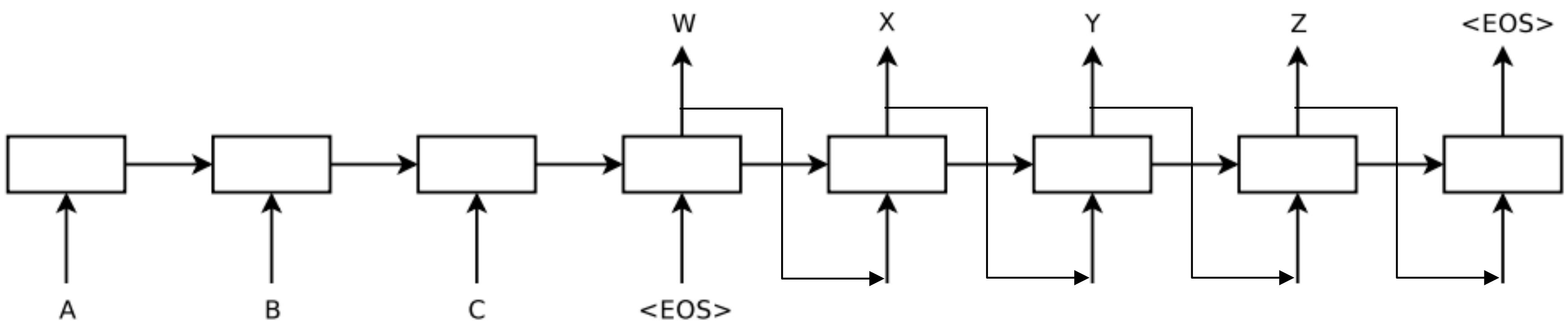
Seq2Seq architecture and Attention mechanism

- Brings many advances in NLP tasks

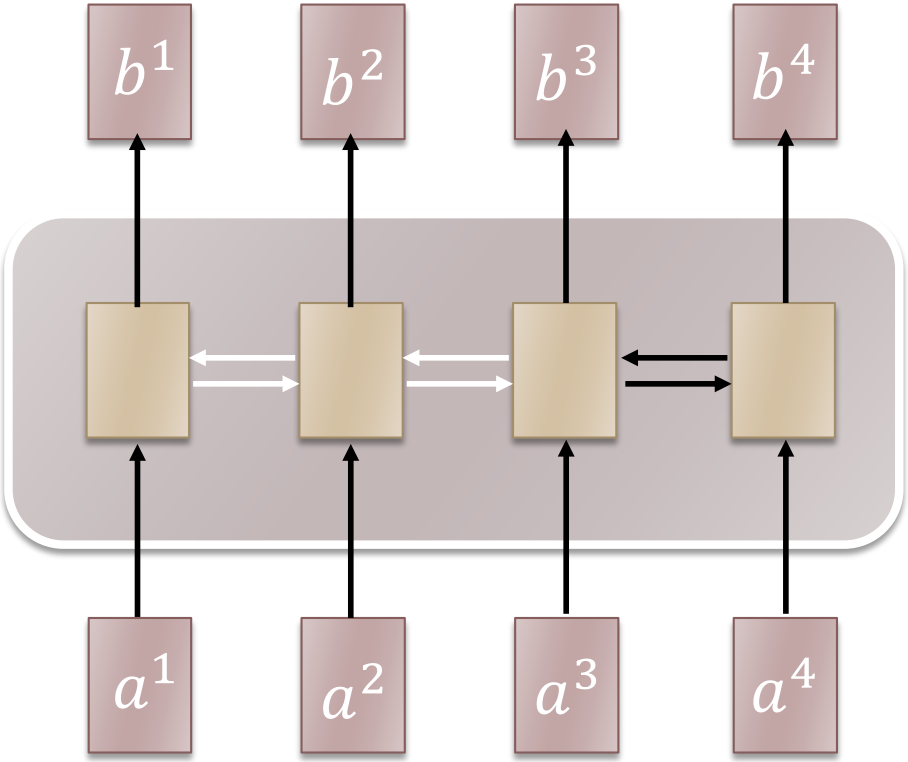


Seq2seq architecture

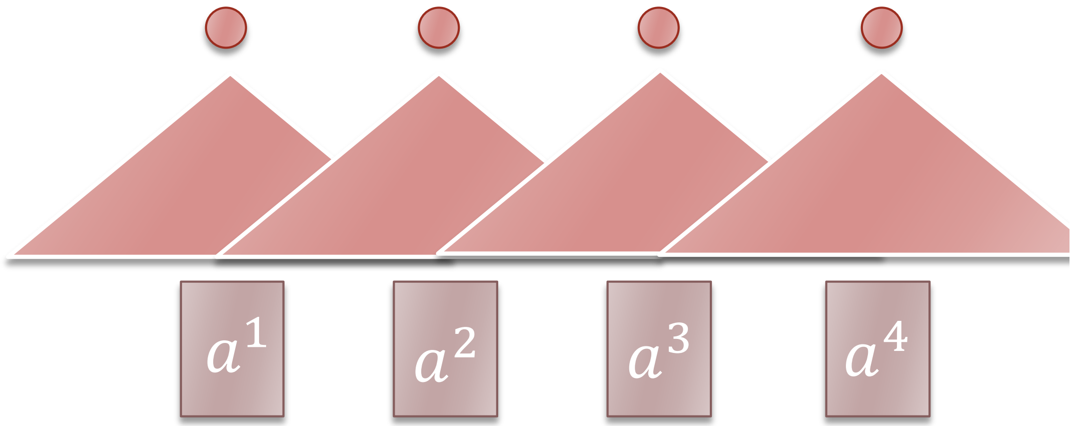
- Seq2Seq is a two-part deep learning architecture to map sequence inputs into sequence outputs
  - was initially proposed for the machine translation task
  - but can be applied for other sequence-to-sequence mapping
- Built using two Recurrent Neural Networks (RNNs), namely the encoder and the decoder
  - The encoder reads a sequence input with variable lengths, e.g., English words,
  - and the decoder produces a sequence output, e.g., corresponding French words, considering the hidden state from the encoder. The hidden state
- Main problem: sends source information from the encoder to the decoder, linking the two. Both the encoder and decoder consist of RNN cells or its variants such as LSTM and GRU.
  - difficult to parallelize, very long learning time



Replace Sequence by Self-attention



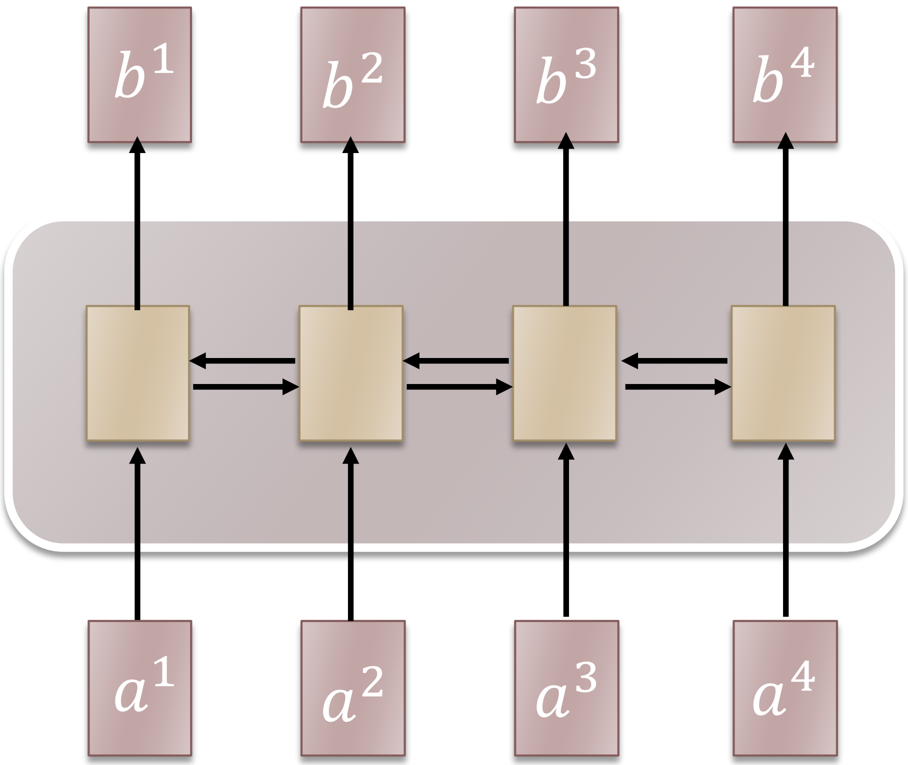
Difficult to parallelize with RNNs

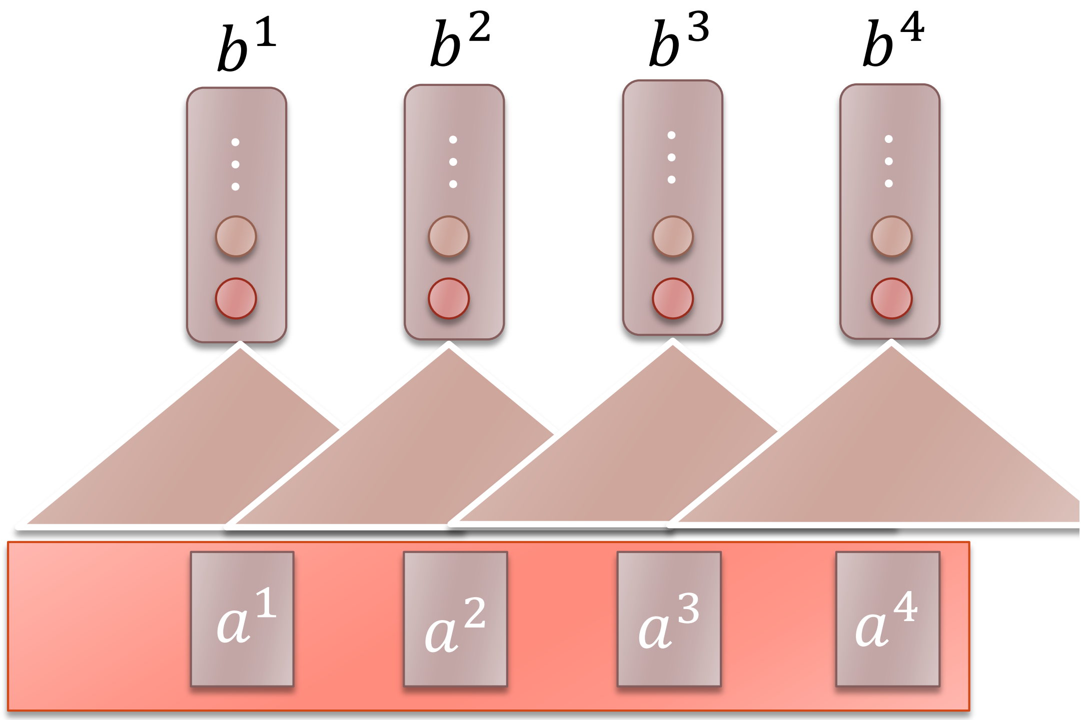


Idea: Replace RNN with CNNs (CNN can parallel)

Replace Sequence by Self-attention

b^1, b^2, b^3, b^4 can be parallelly computed.





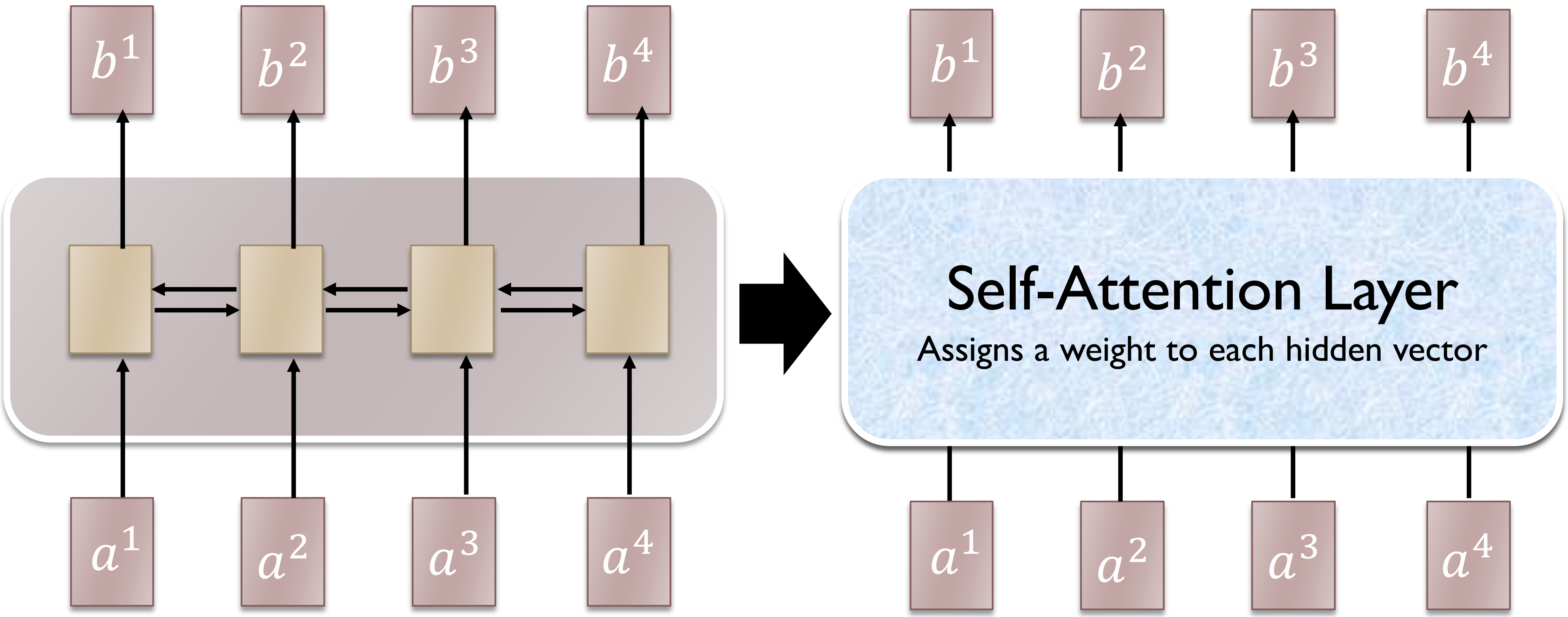
Difficult to parallelize with RNNs

Idea: Replace RNN with CNNs (CNN can parallel)

Replace Sequence by Self-attention

b^i is obtained based on the whole input sequence.

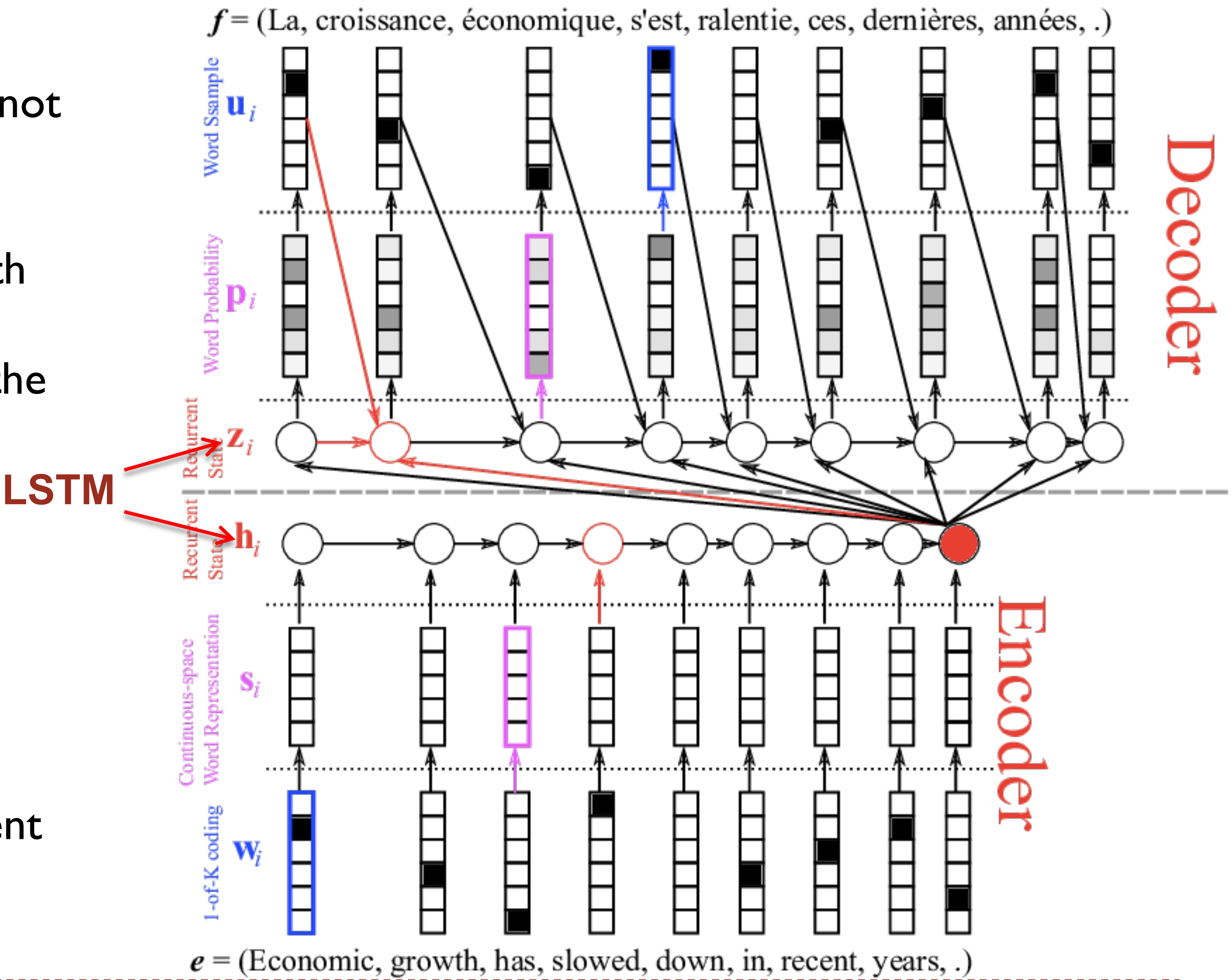
b^1, b^2, b^3, b^4 can be parallelly computed.



You can try to replace any thing that has been done by RNN with self-attention.

LSTM Encoder-decoder machine

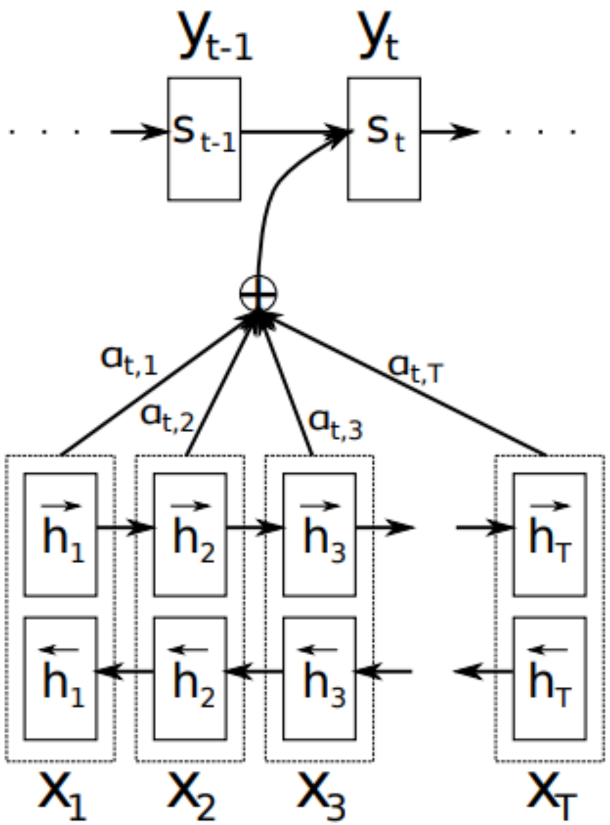
Seq2Seq architecture cannot capture all information by a single fixed length vector (i.e. the hidden state of the encoder)



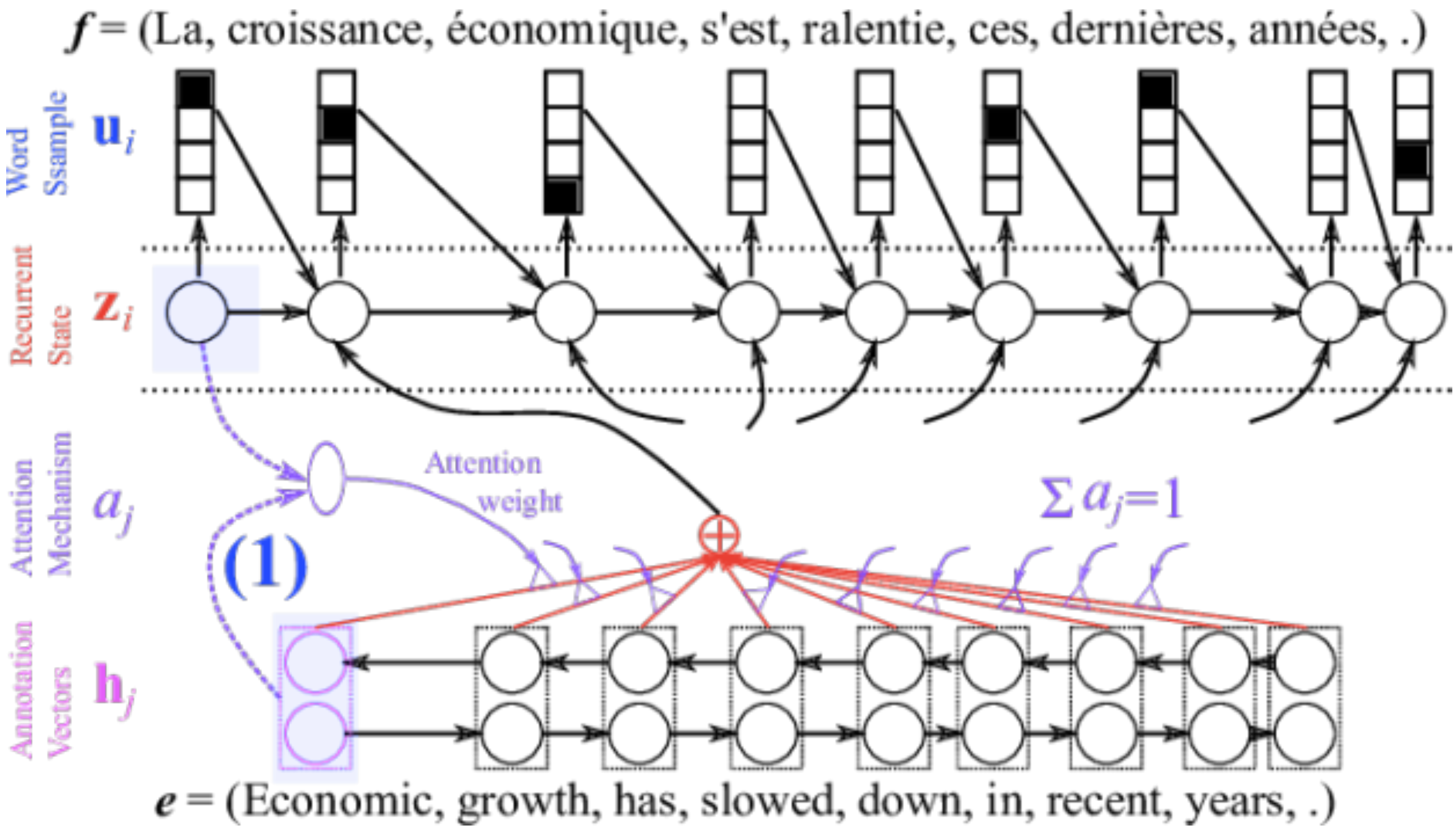
Problems when processing long sequences (vanishing gradient Problem)

Attention

- Attention proposes to use a context vector to represent the contributions of the source and the target (s_t)
- Context vector preserves the information of all hidden states of the encoder cells and aligns them with the current target output ($\sum_i \alpha_{t,i} . hi$)
  - $\sum_i \alpha_{t,i} = 1$
- Model to "take care" of a certain part of the source inputs
  - $\alpha_{t,i} \approx 1, important$
  - $\alpha_{t,i} \approx 0, not\ important$



Attention

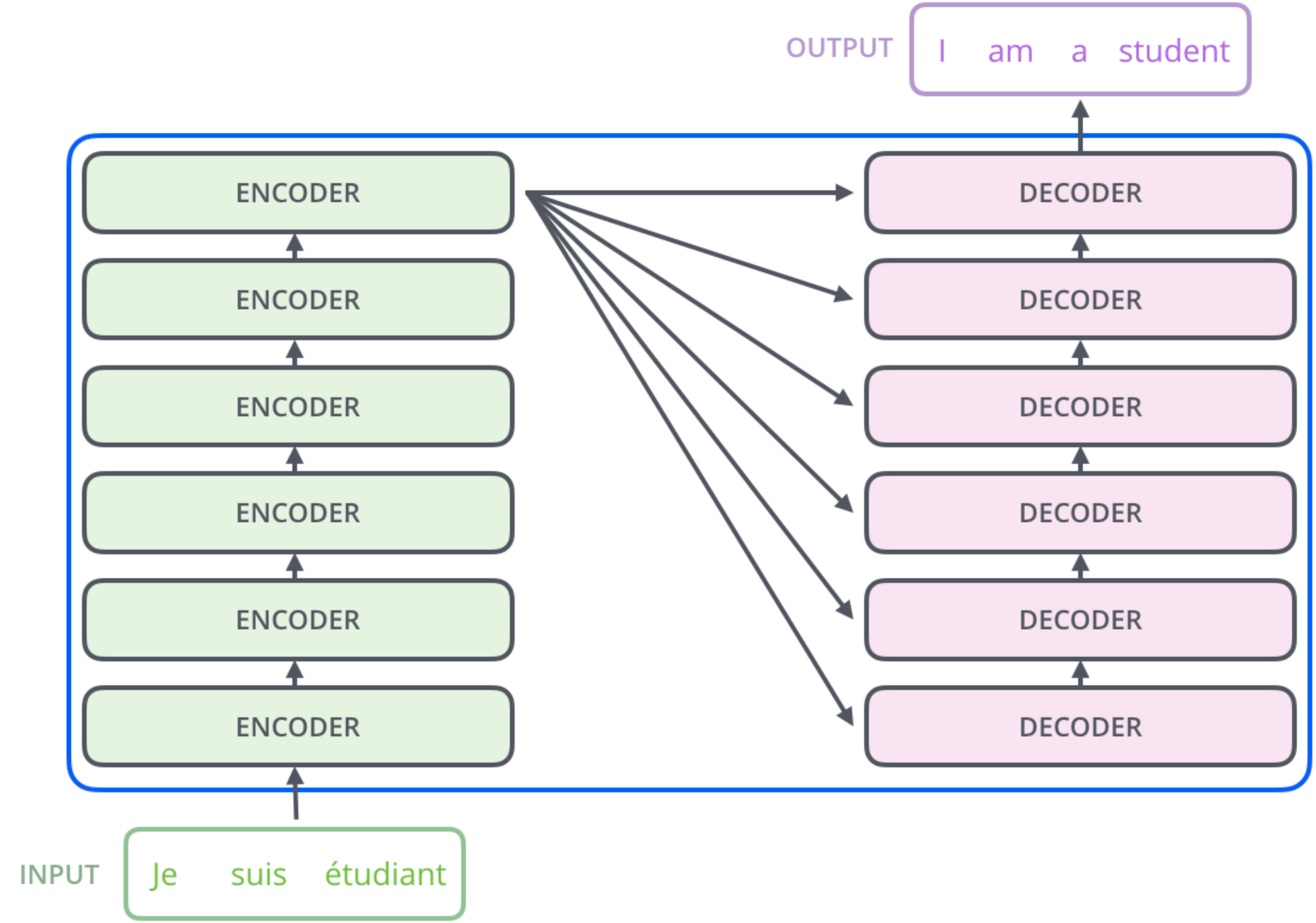


f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
$\mathbf{u}_i$
Recurrent State
$\mathbf{z}_i$
Attention Mechanism
a_j
Attention weight
$\Sigma a_j=1$
(1)
Annotation Vectors
$\mathbf{h}_j$
e = (Economic, growth, has, slowed, down, in, recent, years, .)

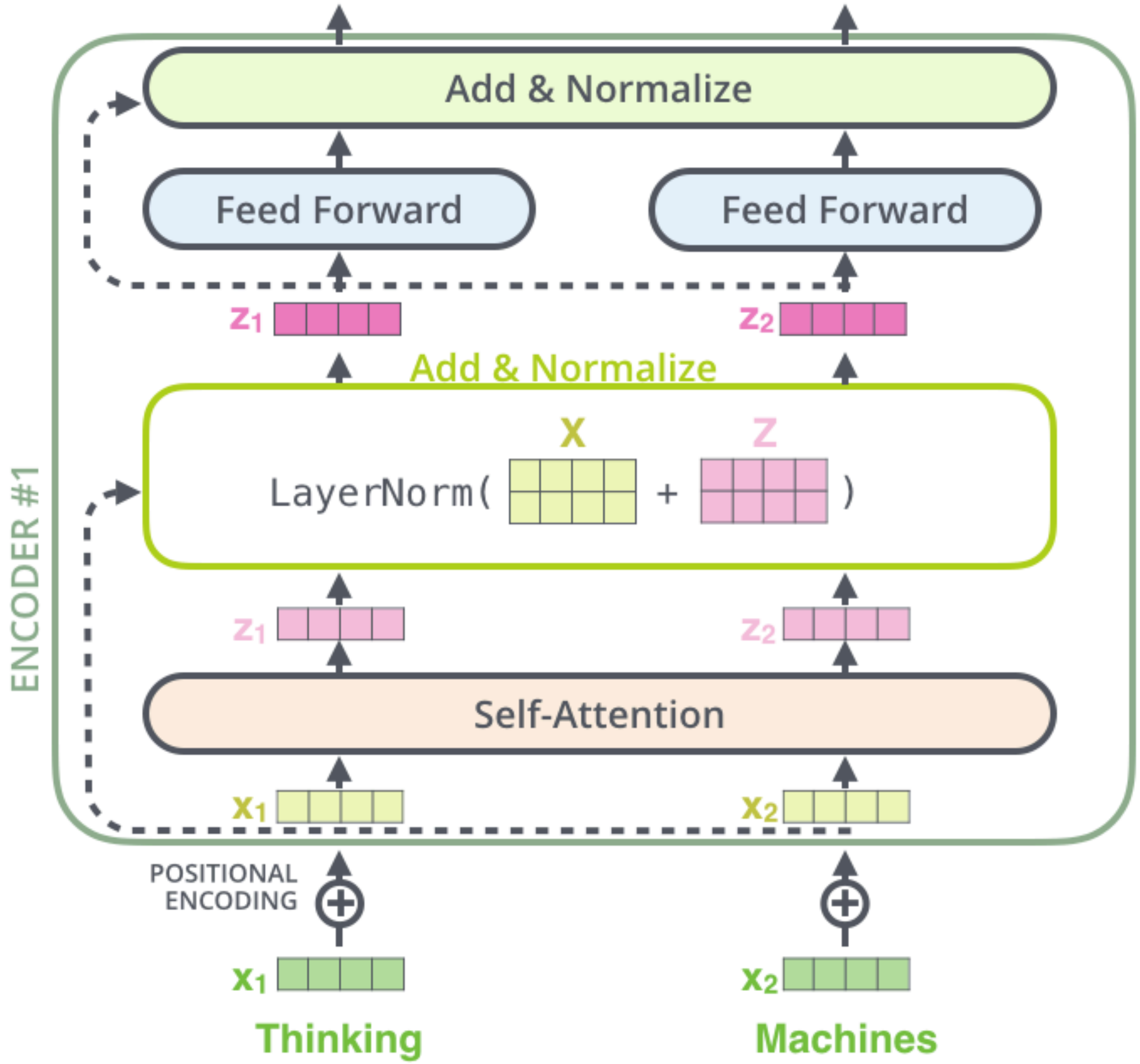
Transformers

- Models that use
  - Extend Seq2seq architecture with self attention
  - Use adapted self-attention to LSTM network
- Two main blocks
  - Encoder
  - Decoder

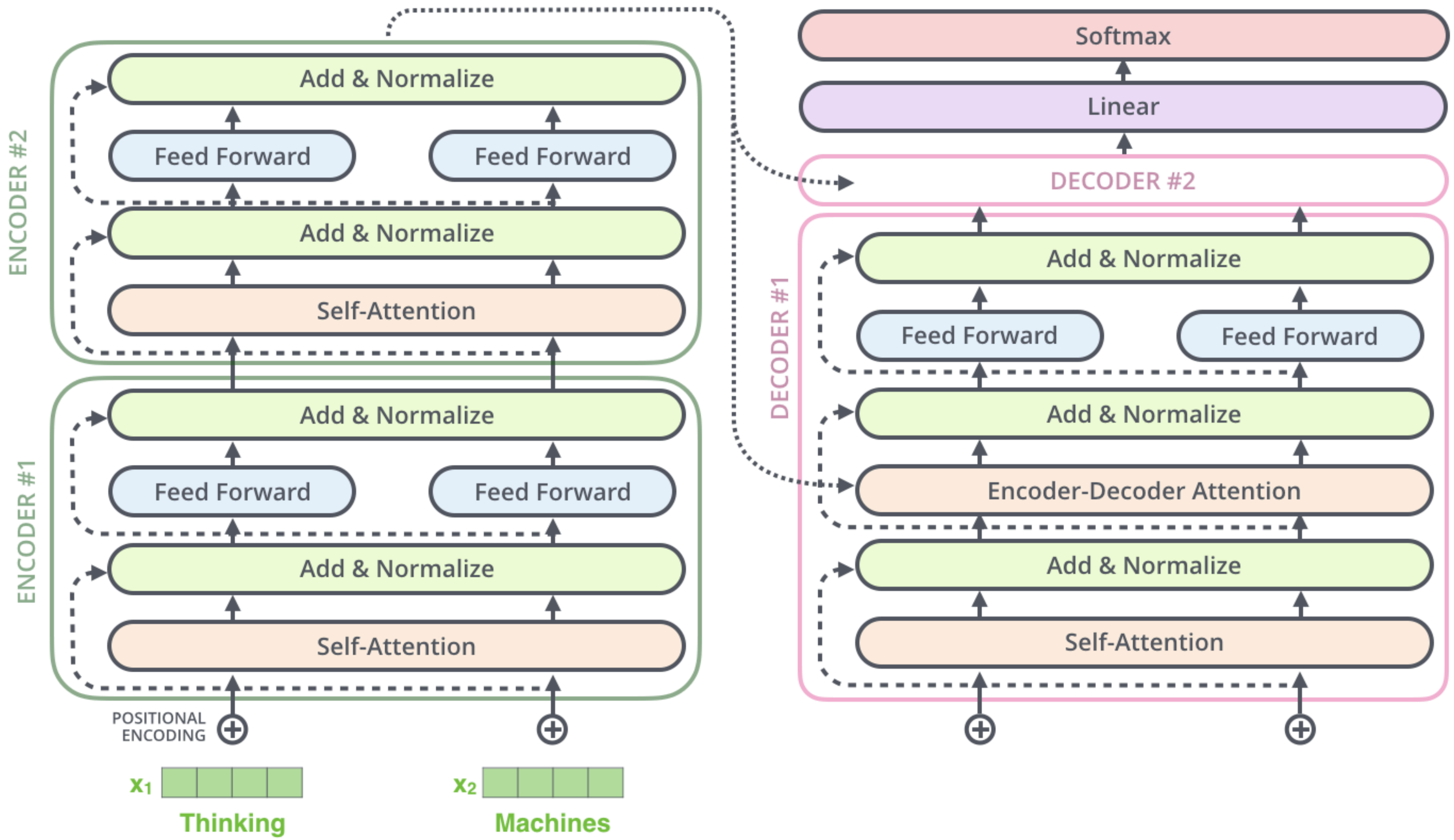
Transformer: Going beyond LSTMs



Transformer
The encoder



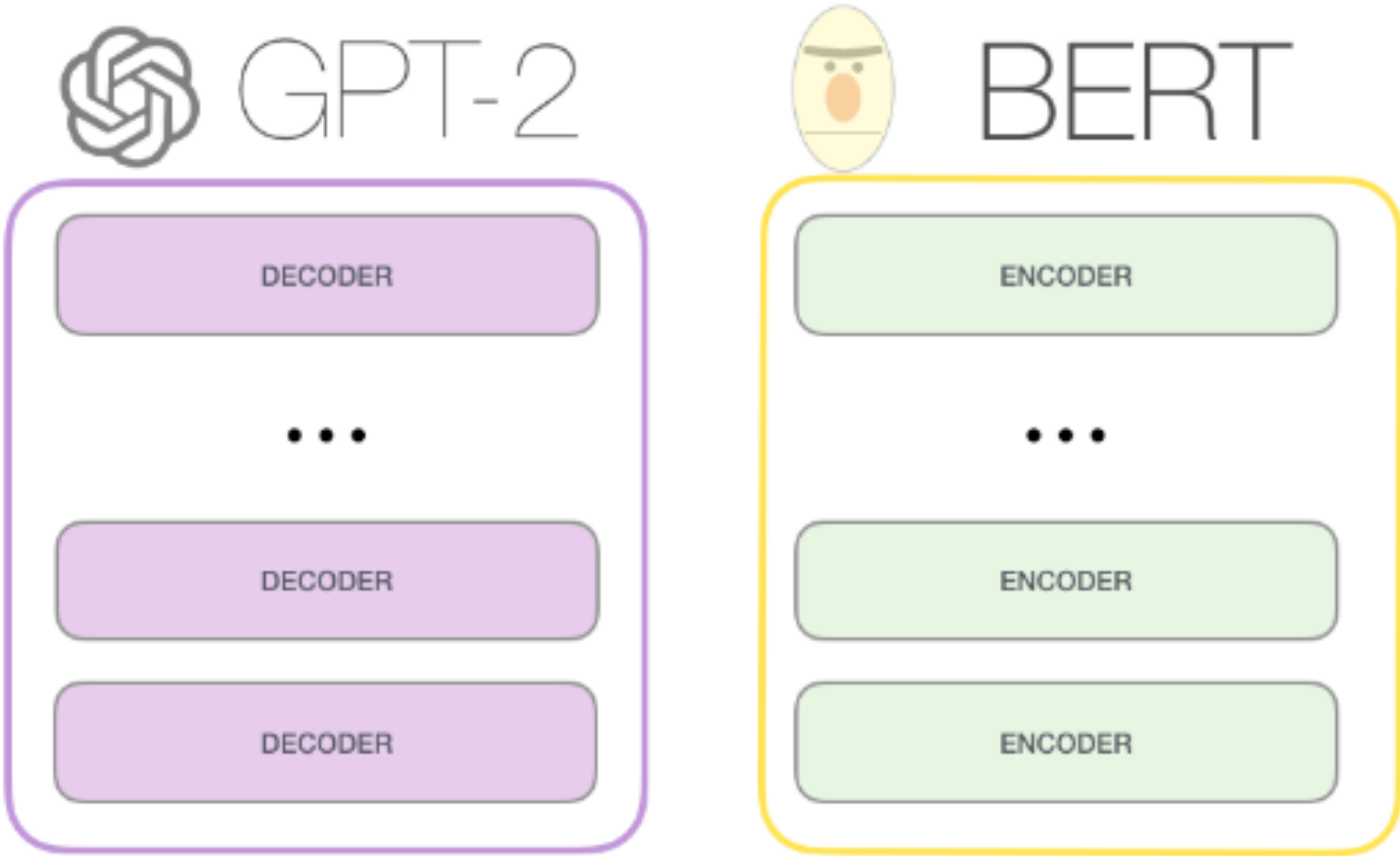
Transformer
The encoder



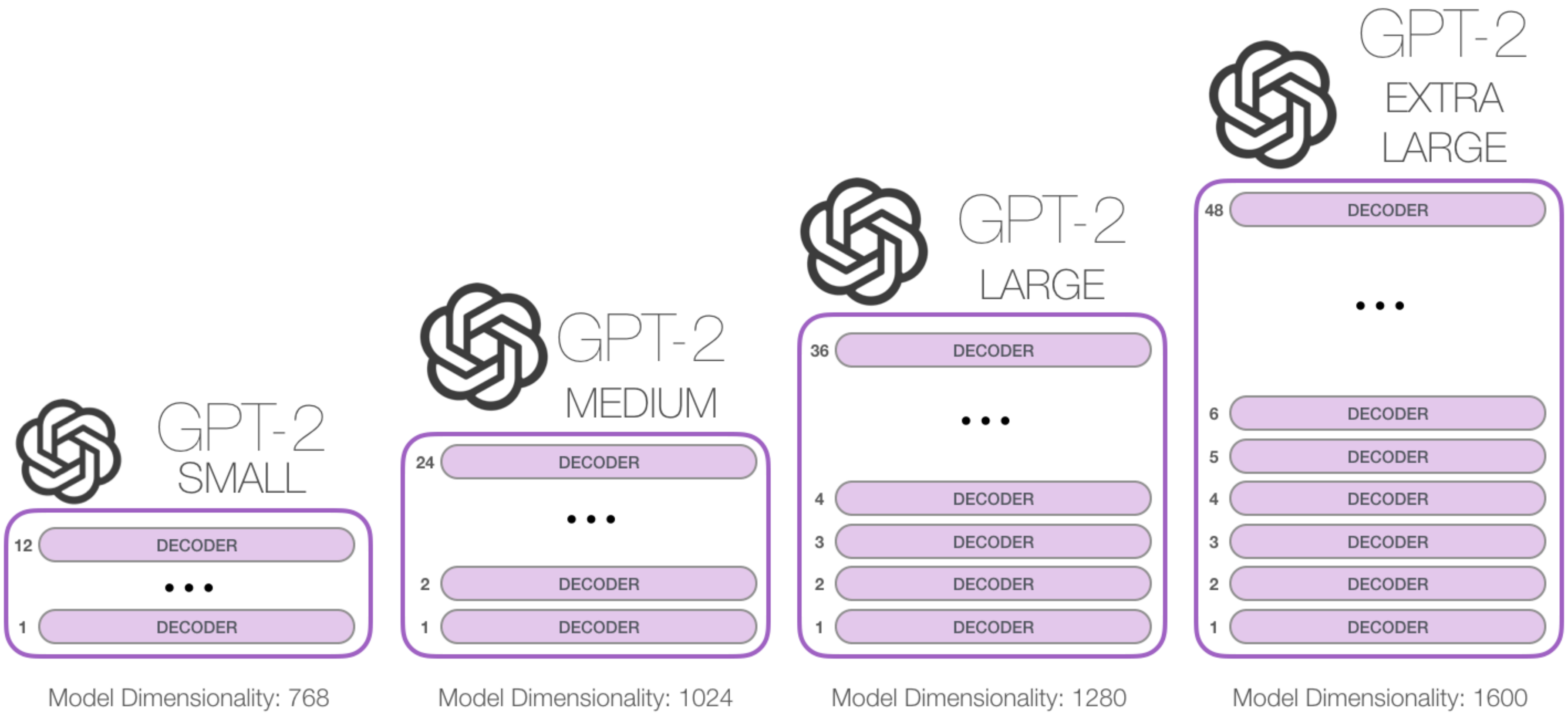
Transformers, GPT-2, and BERT

- A transformer uses an encoder stack to model the input, and a decoder stack to model the output.
- But if we don't have an input and we just want to model the "next word”
  - We can suppress the encoder side of a transformer and output the "next word" one by one
  - It gives us the GPT
- If we are only interested in forming a language model for input for other tasks
  - We don't need the transformer decoder,
  - It gives us BERT.

GPT 2 and BERT



GPT2

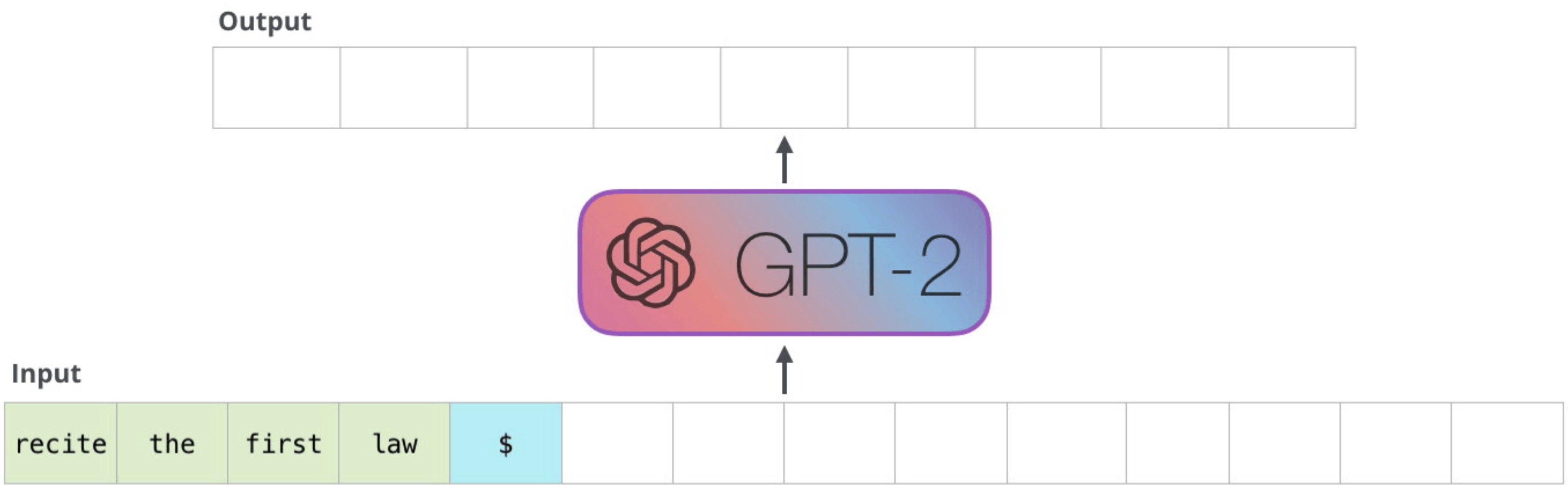


The embedding size varies according to the model (between 768 and 1600)

117 M parameters	345 M parameters	762 M parameters	1,542 M parameters

GPT2 in action

- Input: The GPT-2 can process 1024 tokens.
  - Each token flows through all the decoder blocks along its own path.
- Output: GPT2 produces one token at a time
  - This token is added to the input sequence to produce the following token



http://jalammar.github.io/illustrated-gpt2/

How to use GPT2

- The easiest way to use the transformers is through the API implemented by hugginface
  - To be preferred if you don't need to re-train the network.
  - huggingface.co
  - https://github.com/huggingface/transformers

  - Use
    - Text generation
    - Text Summarization

- You can also install from scratch
  - Use if you need to specialize the network (fine tunning)
  - https://medium.com/analytics-vidhya/gpt2-for-sentiment-analysis-38cd9832d5e9

GPT2Tokenizer

```
>>> from transformers import GPT2Tokenizer
>>> tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

>>> tokenizer("Hello world")['input_ids’]
[15496, 995]

>>> tokenizer(" Hello world")['input_ids’]
[18435, 995]
```

- Remark: The tokenizer treats spaces as parts of a token.
  - A word will not be encoded identically depending on whether it is at the beginning of the sentence (without space) or not.

Text generation with GPT2…

- **Go to GPT2 notebook**
- **Need the installation of transformers library**

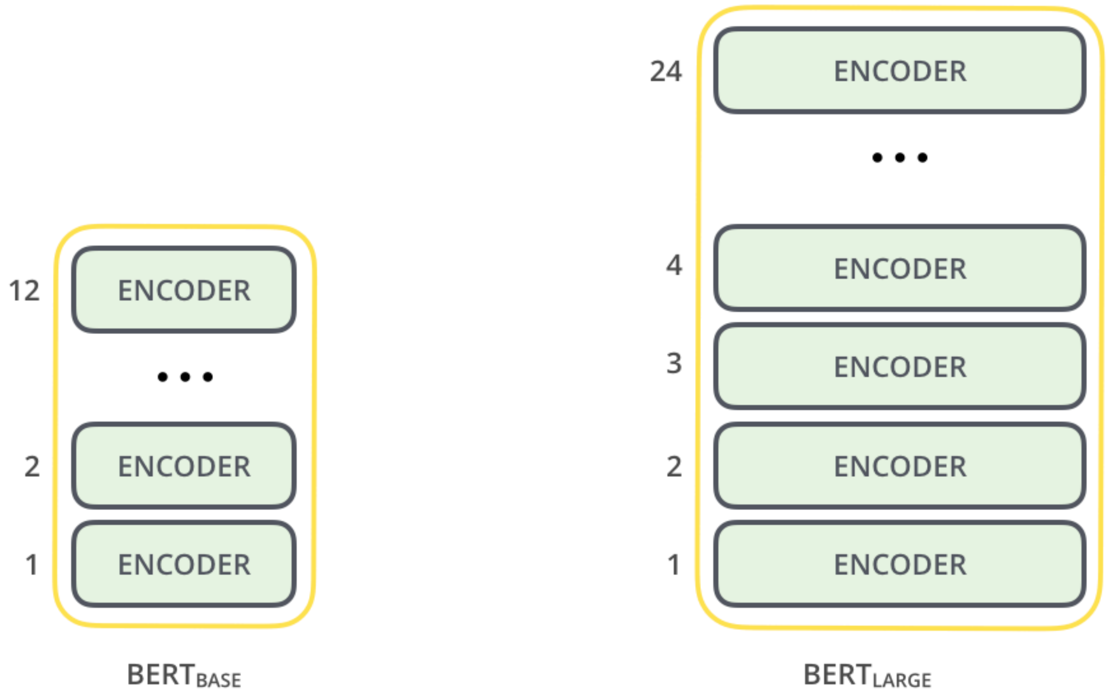
- from transformers import GPT2Tokenizer, GPT2LMHeadModel

- tokenizer = GPT2Tokenizer.from_pretrained('distilgpt2')
- indexed_tokens = tokenizer.encode(text)

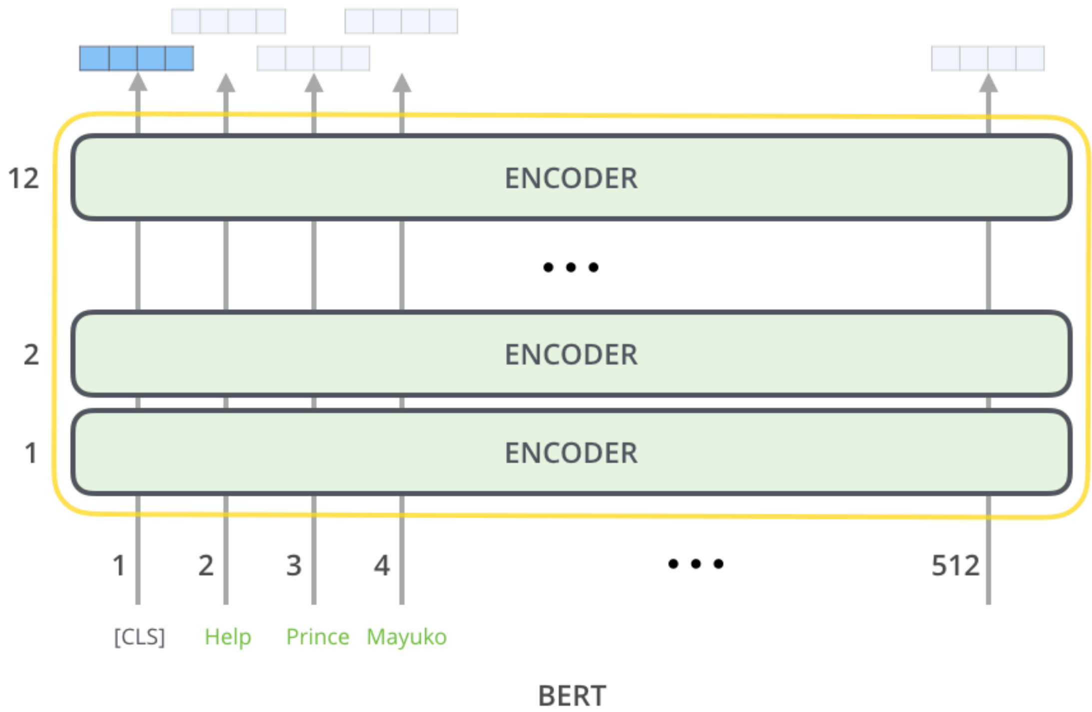
- model = GPT2LMHeadModel.from_pretrained('gpt2')
- outputs = model(tokens_tensor)

- predicted_index = torch.argmax(predictions[0, -1, :]).item()
- predicted_text = tokenizer.decode([predicted_index])

Bidirectional Encoder Representation from Transformers (BERT)

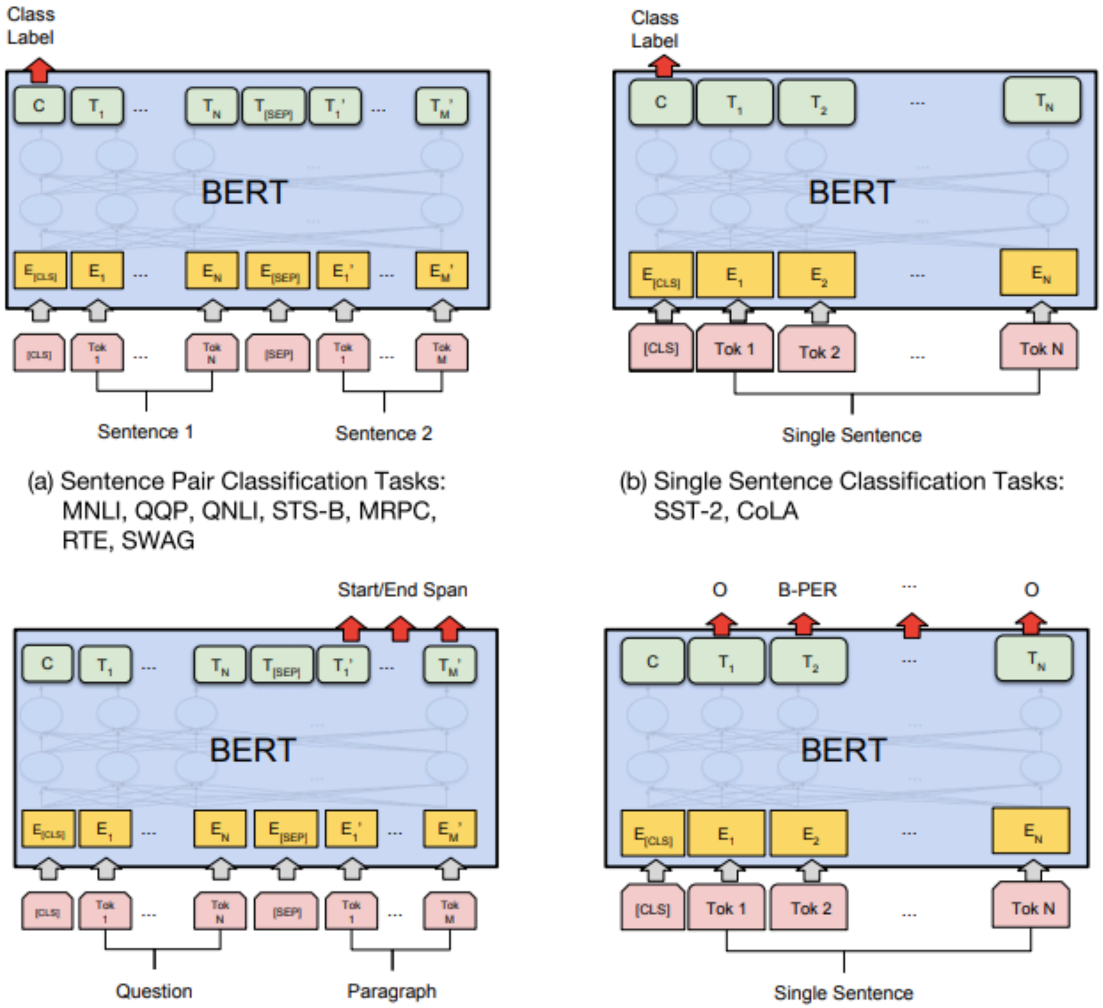


Model input and output



Input: 512 tokens
Output: 768 or 1024 regardings the model

Main BERT usage with/without fine training



(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

How to use BERT

- Like GPT2, the easiest way to use the transformers is through the API implemented by hugginface
  - To be preferred if you don't need to re-train the network.
  - huggingface.co
  - https://github.com/huggingface/transformers

  - Use
    - Text classification
    - NER task

BertTokenizer

>>> from transformers import BertTokenizer

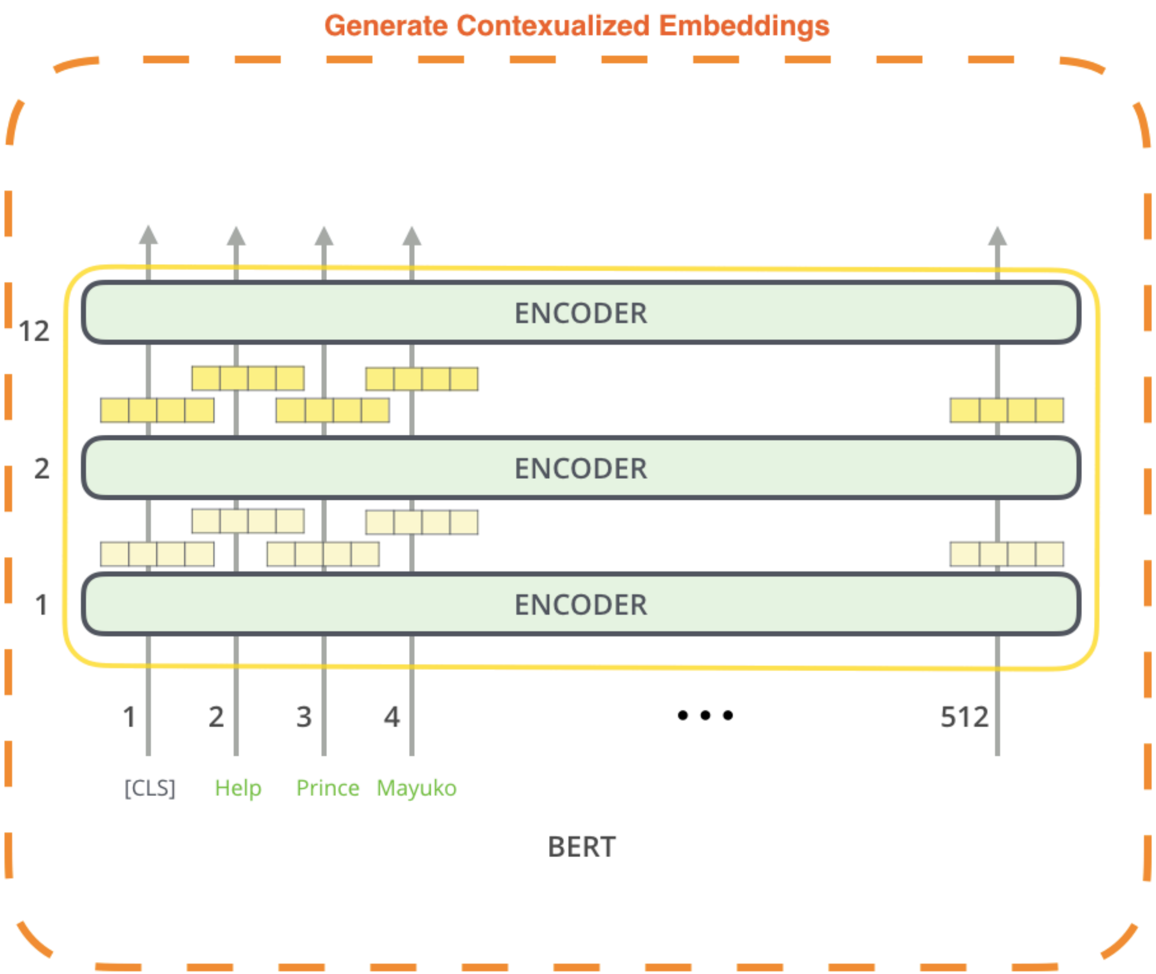
>>> tokenizer = BertTokenizer.from_pretrained(" bert-base-uncased ")

>>> tokenizer.tokenize("I take aspirin.")
['i', 'take', 'as', '##pi', '##rin', '.']

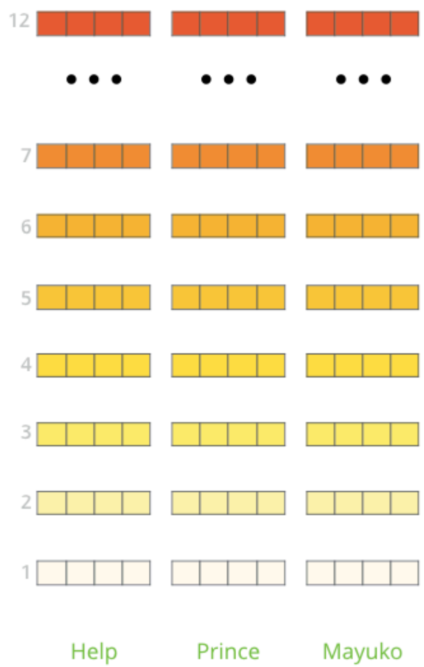
>>> tokenizer.tokenize("I like chocolate")
['i', 'like', 'chocolate']

- Remark: The tokenizer split OOV in sub-piece
  - the same token always has the same id
  - but the embedding changes

Feature extraction with BERT



The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

Bert embedding (feature extraction)

Use all encoder level

- from transformers import BertTokenizer, TFBertModel

- tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
- model = TFBertModel.from_pretrained('bert-base-uncased')

- tokenized_text = tokenizer.encode(review)
- input_ids = tf.constant(tokenized_text[:MAX_BERT_SIZE-2])
- outputs = model(input_ids[None, :])
- Prediction_scores, classification_scores = outputs[:2]

Prediction_scores : embedding of each word

- Shape: nb_sentences, nb_tokens, 768

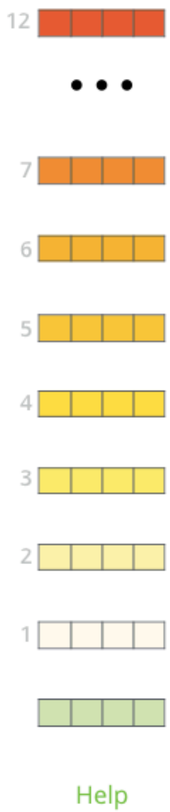
Classification_scores : last token (sentence embedding)

- Shape: nb_sentences, 768

Feature Extraction, which embedding to use?

What is the best contextualized embedding for "Help" in that context?

For named-entity recognition task CoNLL-2003 NER



	Layers used	Dev F1 Score
First Layer	Embedding	91.0
Last Hidden Layer	12	94.9
Sum All 12 Layers	12 + … + 2 + 1	95.5
Second-to-Last Hidden Layer	11	95.6
Sum Last Four Hidden	12 + 11 + 10 + 9	95.9
Concat Last Four Hidden	9, 10, 11, 12	96.1

Sentiment analysis with Bert

- **Go to Bert notebook**
- **Need the installation of transformers library**

- from transformers import BertTokenizer
- BertTokenizer.from_pretrained("bert-base-uncased")

- marked_text = "[CLS] " + "I take aspirin. I like chocolate" + " [SEP]"
- tokenized_text = tokenizer.tokenize(marked_text)

['[CLS]', 'i', 'take', 'as', '##pi', '##rin', '.', 'i', 'like', 'chocolate', '[SEP]']

- encoded_text = tokenizer.encode(tokenized_text)

Sentiment analysis with Bert

- outputs = model(tf.constant(encoded_text))[None, :])

- prediction_scores, classification_scores = outputs[:2]

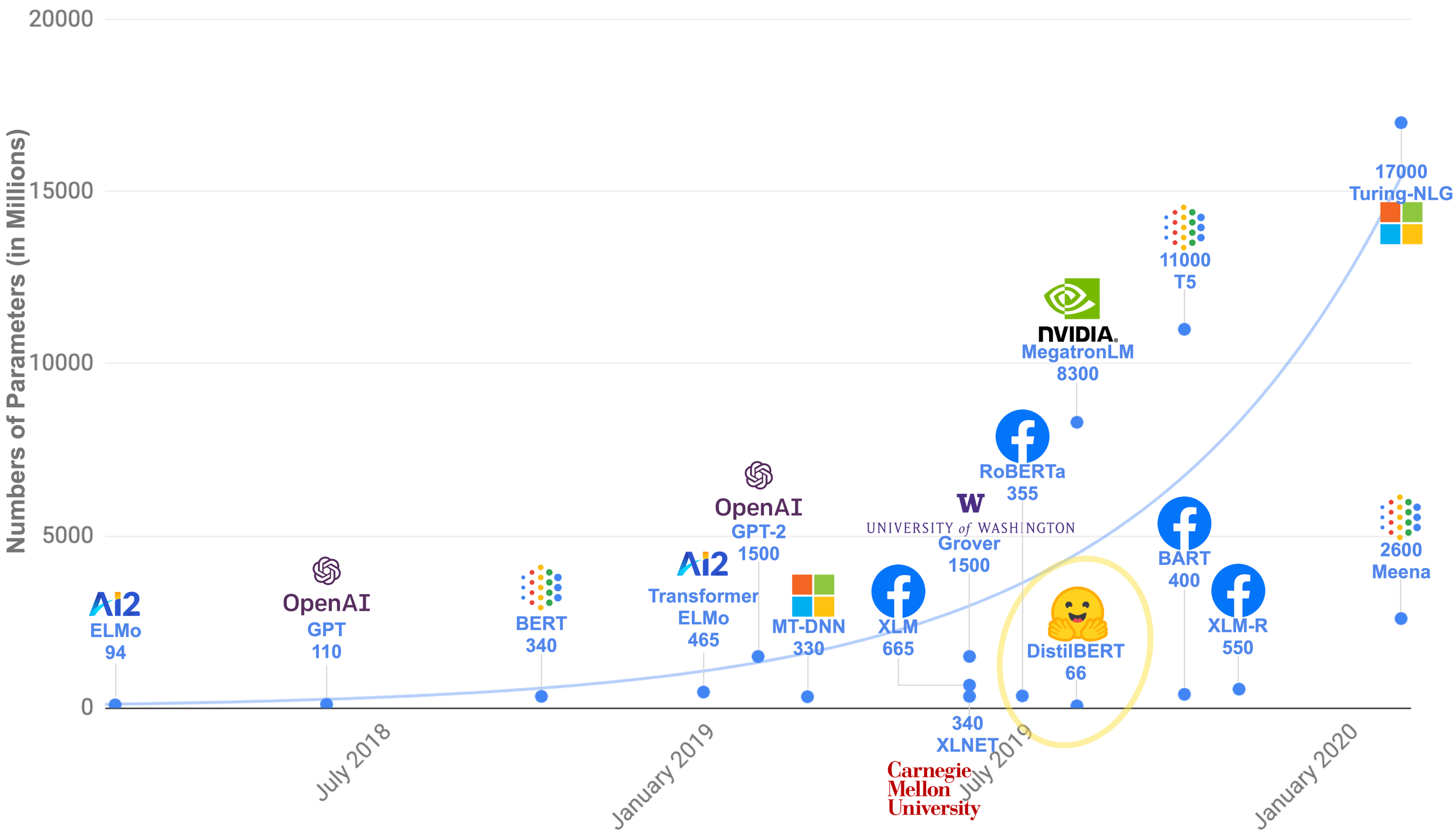
- prediction_scores.shape

(1, 10, 768)

- classification_scores.shape

(1, 768)

Summary



20000
15000
10000
5000
0
Numbers of Parameters (in Millions)
ELMo
94
OpenAI
GPT
110
BERT
340
Transformer
ELMo
465
OpenAI
GPT-2
1500
MT-DNN
330
XLM
665
340
XLNET
Carnegie Mellon University
UNIVERSITY of WASHINGTON
Grover
1500
RoBERTa
355
NVIDIA
MegatronLM
8300
DistilBERT
66
11000
T5
BART
400
XLM-R
550
17000
Turing-NLG
2600
Meena
July 2018
January 2019
July 2019
January 2020

Summary

- Transformer
  - Modelling language
  - Use the entire sentence before producing an output
    - The output could depend of the task

- LSTM is difficult to parallelize
- Self-Attention is a proposal to resolve the problems
- Transformer is sequence-to-sequence architecture
  - A set of encoders construct a latent representation of the input
  - A set of decoders could be use pour project the latent representation in. a new space

- Specific lecture next year

Summary

- Transformer use very large model
  - Not so easy to use
  - Need computational resources

- 2 easy way to use transformer model
  - Tensorflow_hub library: https://www.tensorflow.org/hub/installation
  - Transformers library: https://huggingface.co/transformers/

- ELMO use character embedding and a bi-LSTM in order to produce an embedding based on word prediction
- Bert uses mainly the encoder part and could be used for
  - Word / Sentence embedding
  - Text classification or Sentiment Analysis
  - NER
  - Q&A or Text translation
- GPT uses mainly the decoder part and could be used for
  - Text summarization
  - Text generation