AIM 2: Artificial Intelligence in Medicine II

Embeddings and Transformers

HARVARD
MEDICAL SCHOOL

Carlos Morato, Phd.

Outline for today’s class

- Embeddings and their role in NLP
- Transformers and RNNs
- Stack-encoder and Stack-decoder architectures
- BERT and GPT
- Hugging Face library for NLP applications
- Clinical BERT and BioBERT
- LLM-based medical question-answering.

Text Representations

- Co-occurrence statistics
  - Brown Clusters
  - Count vectors, TF-IDF vectors, co-occurrence matrix decomposition
- Predictive
  - word2vec, GloVe, CBOW, Skip-Gram, etc
- Contextualized language models
  - Representation of word *changes* based on context
  - CoVE, ELMo, GPT, BERT, etc

Word Embeddings

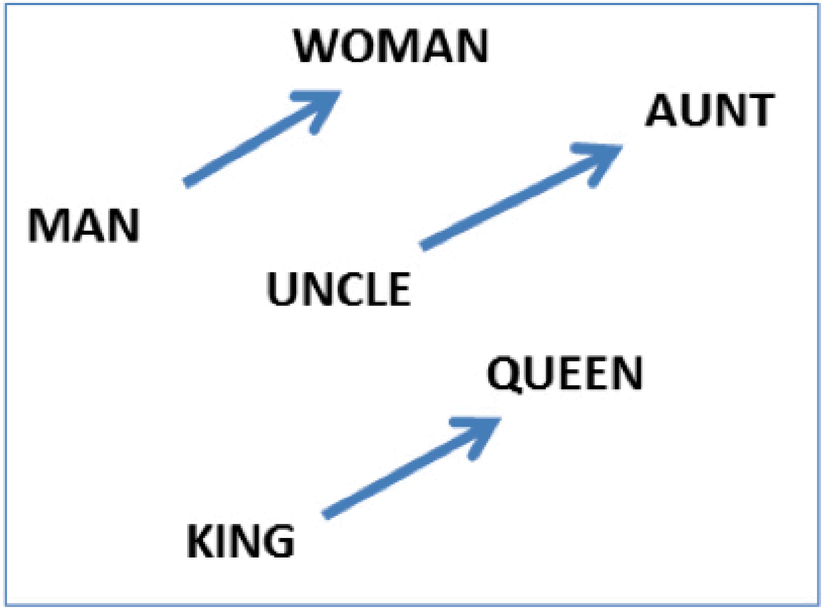
Representing words as vectors in a canonical space:

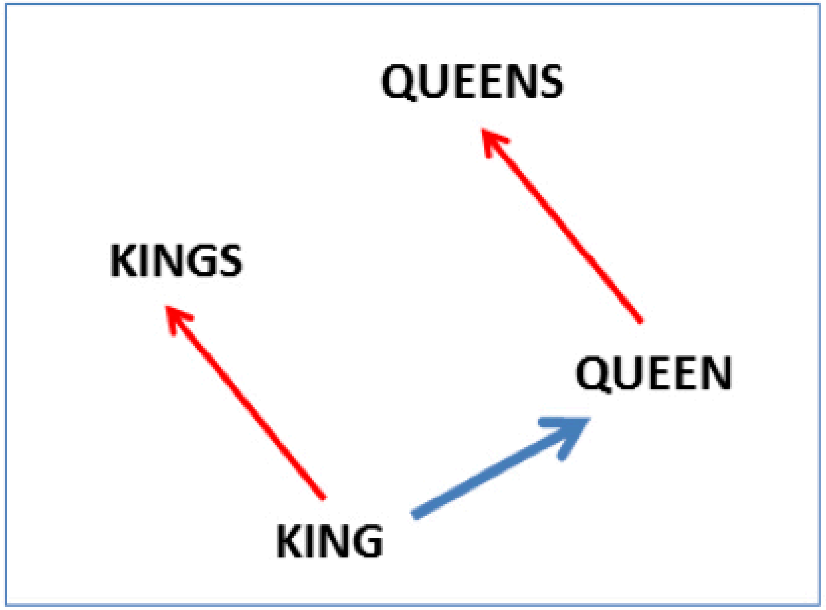
- Two distinct models
  - CBoW
  - **Skip-Gram (SG)**
- Various training methods
  - **Negative Sampling (NS)**
  - Hierarchical Softmax
- A rich preprocessing pipeline
  - Dynamic Context Windows
  - Subsampling
  - Deleting Rare Words

Embeddings capture relational meaning

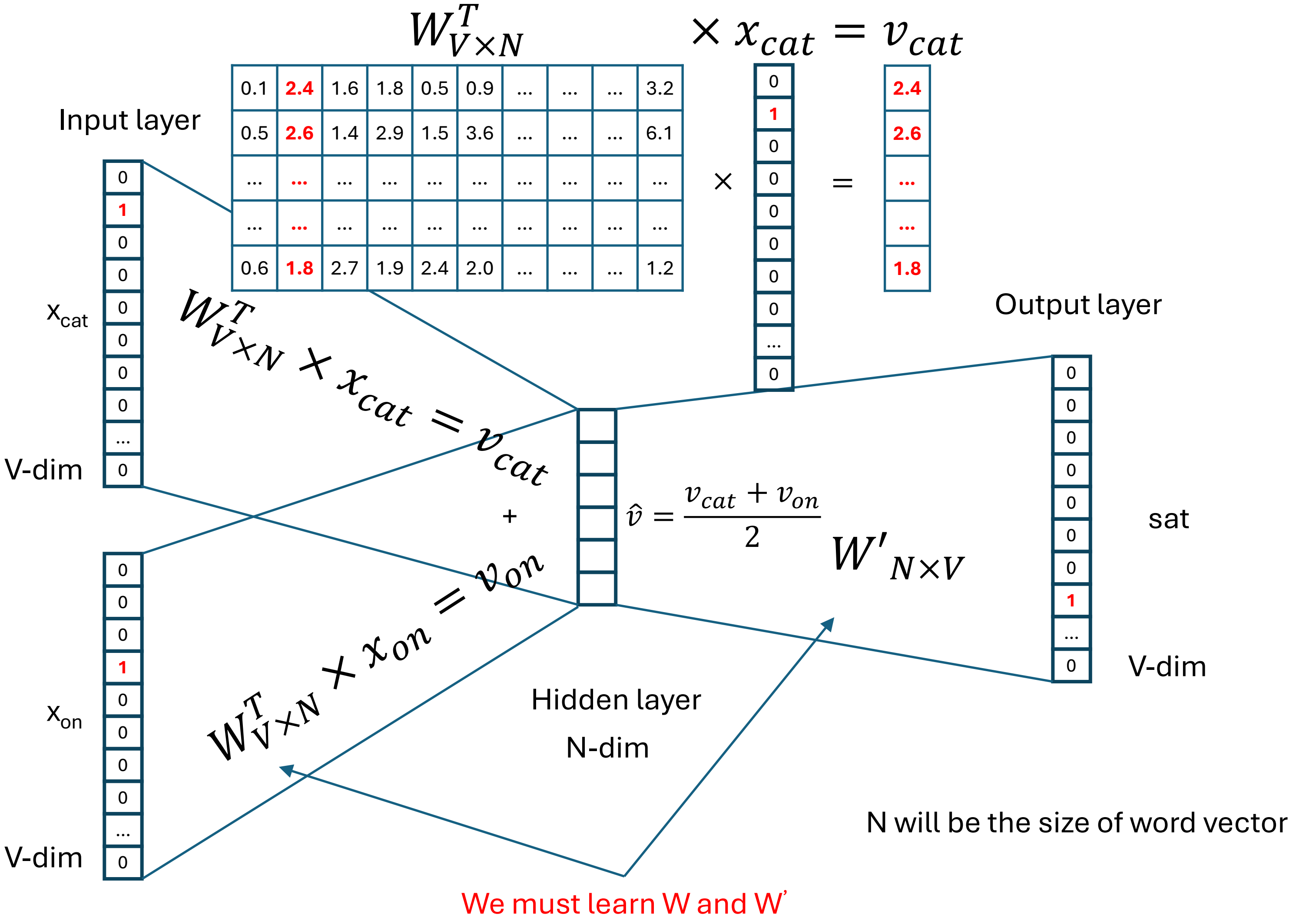
vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) ≈ vector(‘queen’)

vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) ≈ vector(‘Rome’)





Computing Embeddings



Word embedding applications

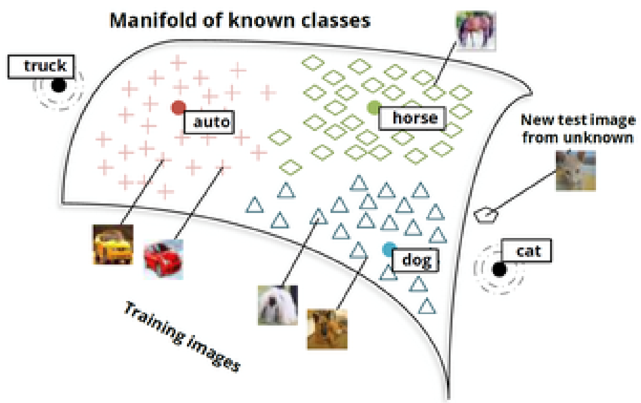
- The use of word representations… has become a key "secret sauce" for the success of many NLP systems in recent years, across tasks including named entity recognition, part-of-speech tagging, parsing, and semantic role labeling. (Luong *et al.* (2013))
- Learning a good representation on a task A and then using it on a task B is one of the major tricks in the Deep Learning toolbox.
    - Pretraining, transfer learning, and multi-task learning.
    - Can allow the representation to learn from more than one kind of data.

- Can learn to map multiple kinds of data into a single representation.

http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

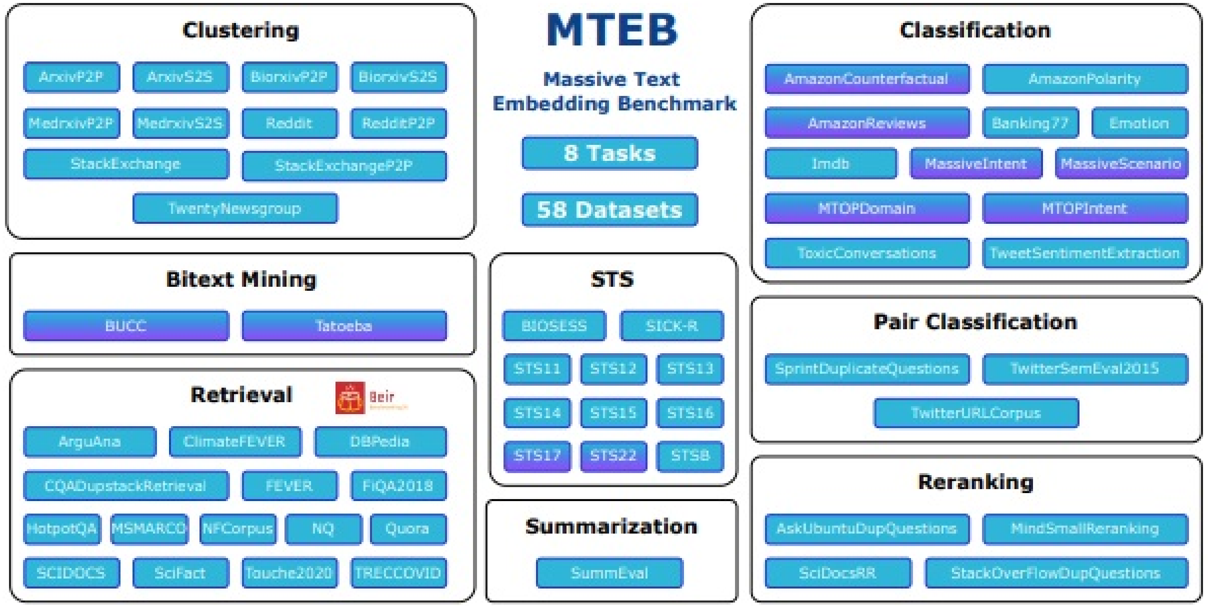
Word embedding applications

- Can apply to get a joint embedding of words and images or other multi-modal data sets.
- New classes map near similar existing classes: e.g., if ‘cat’ is unknown, cat images map near dog.

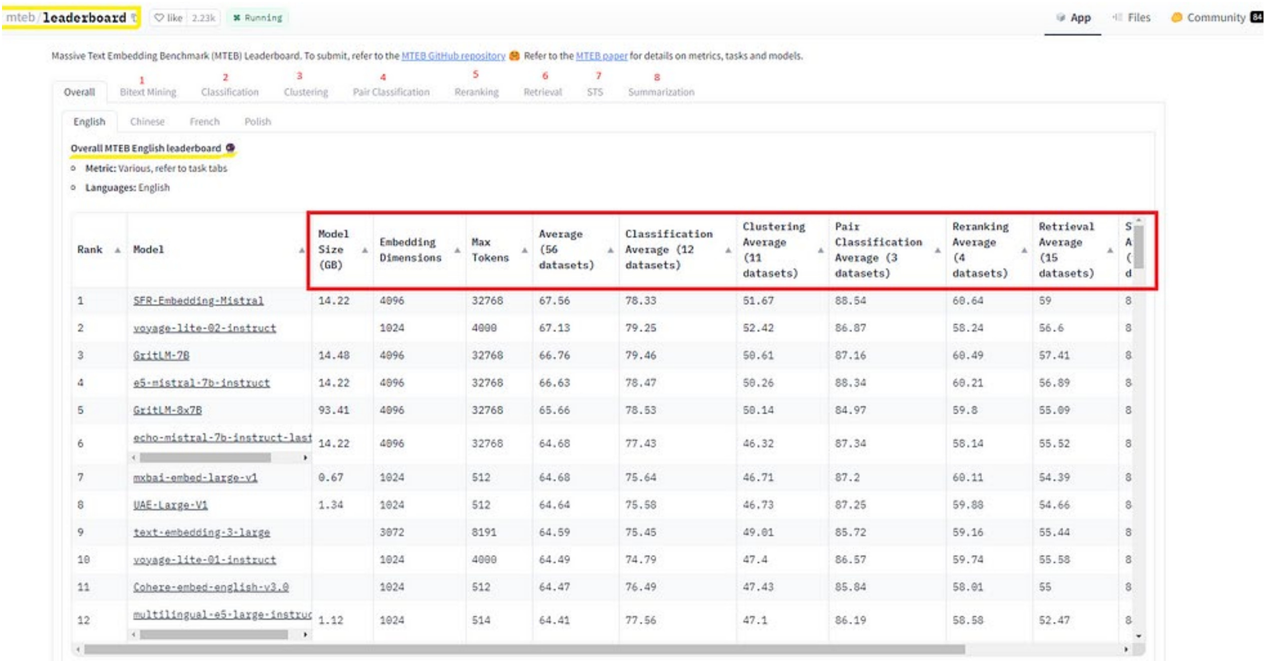


(Socher *et al.* (2013b))
http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

Massive text Embedding Benchmark



An overview of tasks and datasets in MTEB. Multilingual datasets are marked with a purple shade.


mteb/leaderboard like 2.23k Running App Files Community 84

Massive Text Embedding Benchmark (MTEB) Leaderboard. To submit, refer to the MTEB GitHub repository Refer to the MTEB paper for details on metrics, tasks and models.

Overall | 1 Bitext Mining | 2 Classification | 3 Clustering | 4 Pair Classification | 5 Reranking | 6 Retrieval | 7 STS | 8 Summarization

English | Chinese | French | Polish

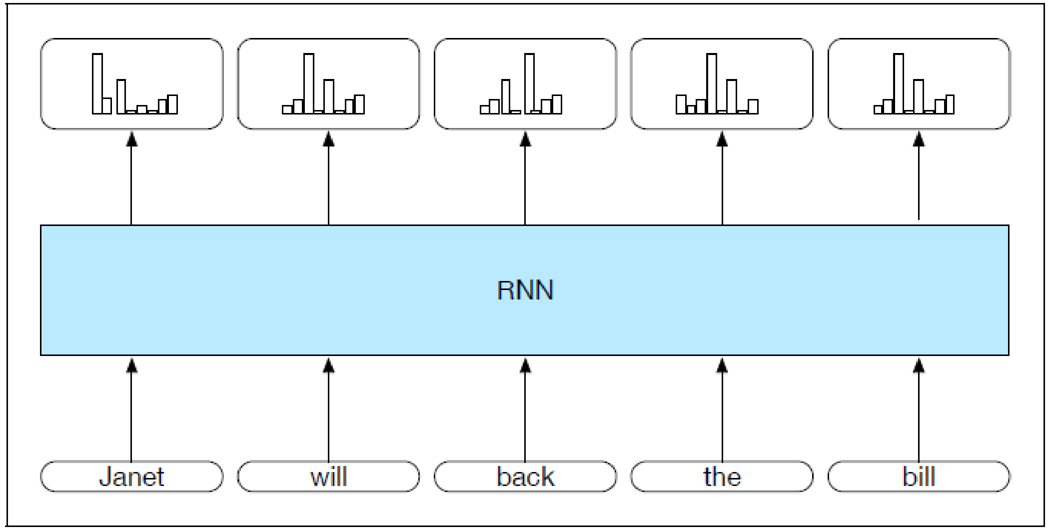
Overall MTEB English leaderboard

- Metric: Various, refer to task tabs
- Languages: English

Rank	Model	Model Size (GB)	Embedding Dimensions	Max Tokens	Average (56 datasets)	Classification Average (12 datasets)	Clustering Average (11 datasets)	Pair Classification Average (3 datasets)	Reranking Average (4 datasets)	Retrieval Average (15 datasets)	S A (d
1	SFR-Embedding-Mistral	14.22	4096	32768	67.56	78.33	51.67	88.54	60.64	59	8
2	voyage-lite-02-instruct		1024	4000	67.13	79.25	52.42	86.87	58.24	56.6	8
3	GritLM-7B	14.48	4096	32768	66.76	79.46	50.61	87.16	60.49	57.41	8
4	e5-mistral-7b-instruct	14.22	4096	32768	66.63	78.47	50.26	88.34	60.21	56.89	8
5	GritLM-8x7B	93.41	4096	32768	65.66	78.53	50.14	84.97	59.8	55.09	8
6	echo-mistral-7b-instruct-last	14.22	4096	32768	64.68	77.43	46.32	87.34	58.14	55.52	8
7	mxbai-embed-large-v1	0.67	1024	512	64.68	75.64	46.71	87.2	60.11	54.39	8
8	UAE-Large-V1	1.34	1024	512	64.64	75.58	46.73	87.25	59.88	54.66	8
9	text-embedding-3-large		3072	8191	64.59	75.45	49.01	85.72	59.16	55.44	8
10	voyage-lite-01-instruct		1024	4000	64.49	74.79	47.4	86.57	59.74	55.58	8
11	Cohere-embed-english-v3.0		1024	512	64.47	76.49	47.43	85.84	58.01	55	8
12	multilingual-e5-large-instruc	1.12	1024	514	64.41	77.56	47.1	86.19	58.58	52.47	8

https://huggingface.co/spaces/mteb/leaderboard

Encoder-Decoder



- **RNN:** input sequence is transformed into output sequence in a one-to-one fashion.

- **Goal:** Develop an architecture capable of generating *contextually appropriate*, *arbitrary length*, output sequences
- **Applications**:
  - Machine translation,
  - Summarization,
  - Question answering,
  - Dialogue modeling.

Simple recurrent neural network illustrated as a feed-forward network

Most significant change: new set of weights, U

- connect the hidden layer from the previous time step to the current hidden layer.
- determine how the network should make use of past context in calculating the output for the current input.

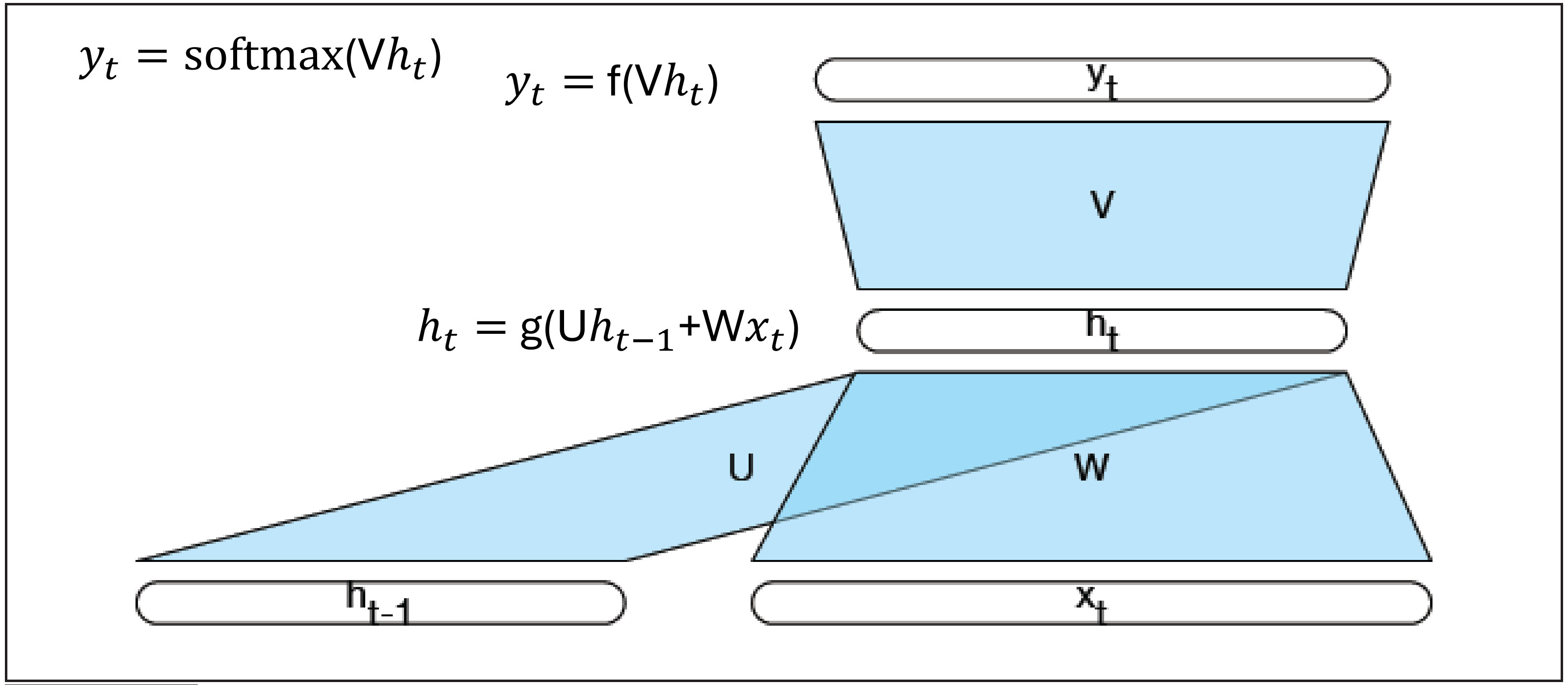
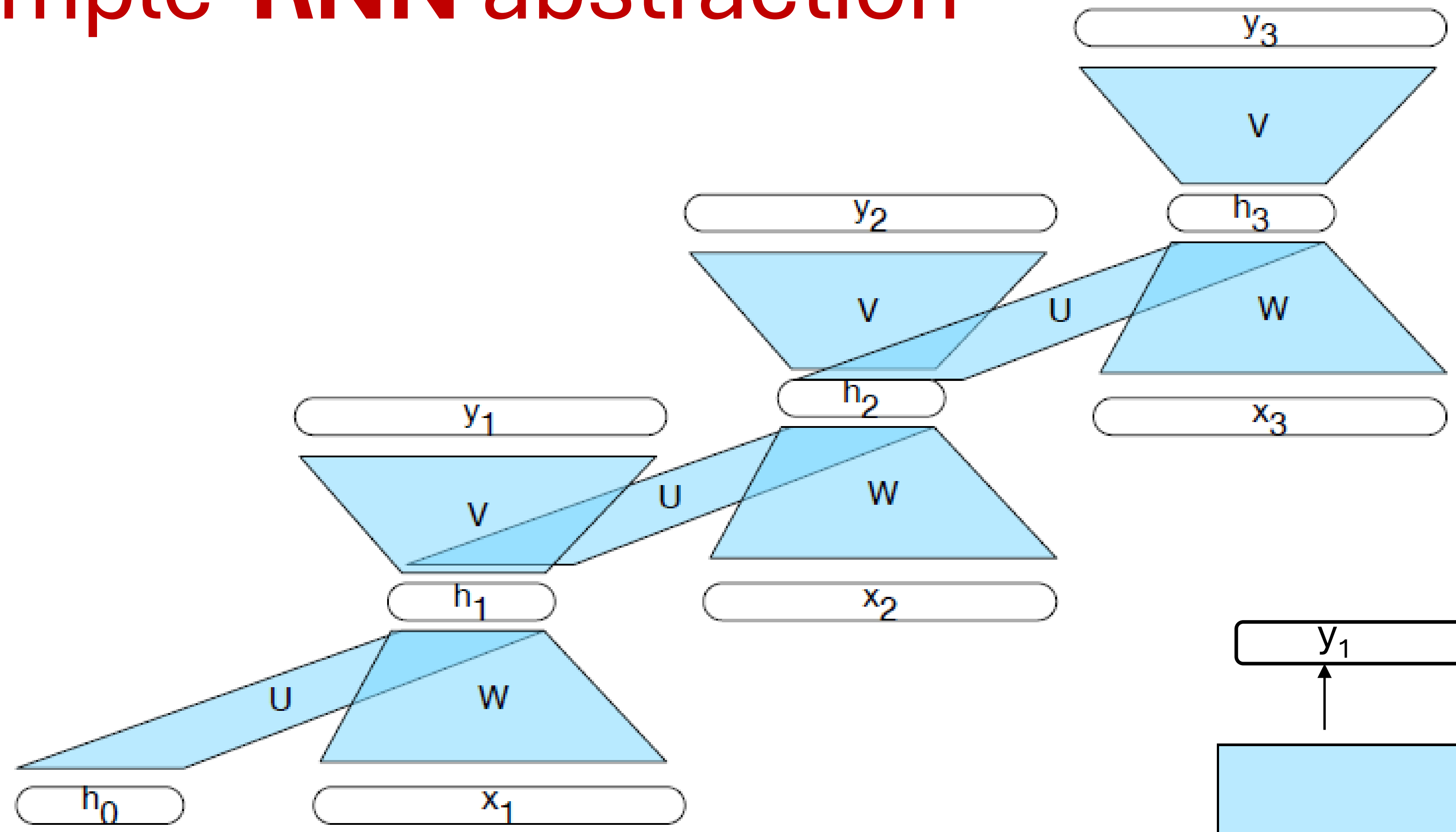
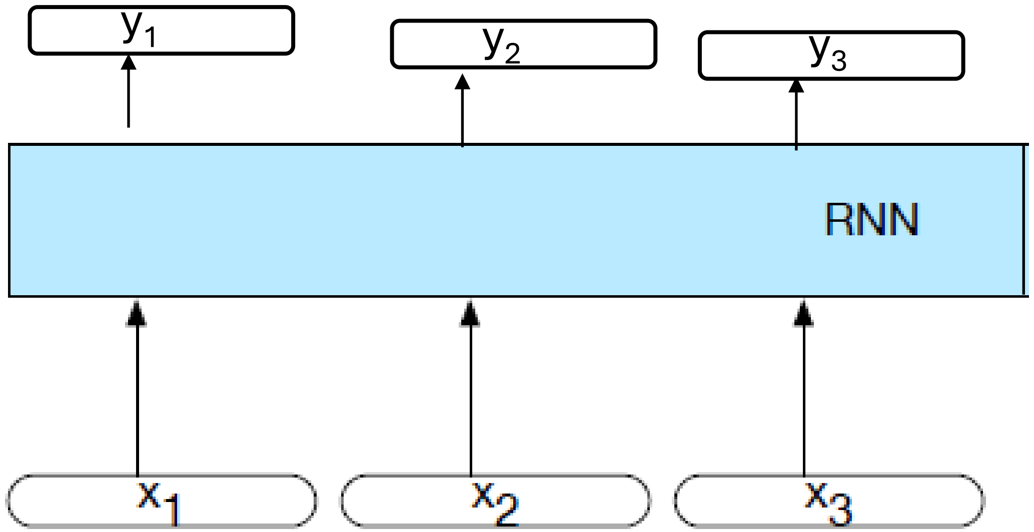


Figure 9.3 Simple recurrent neural network illustrated as a feed-forward network.

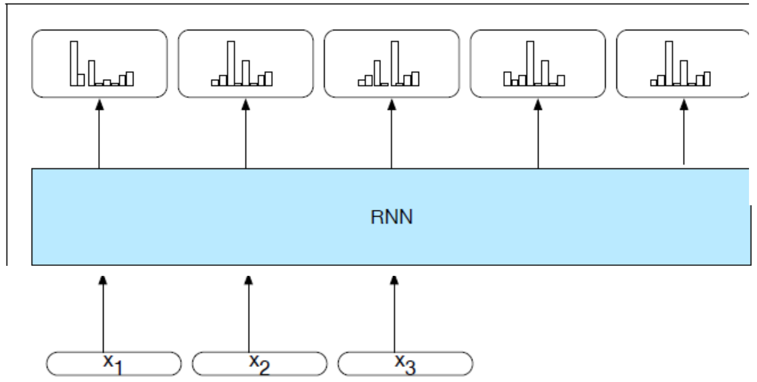
Simple-**RNN** abstraction

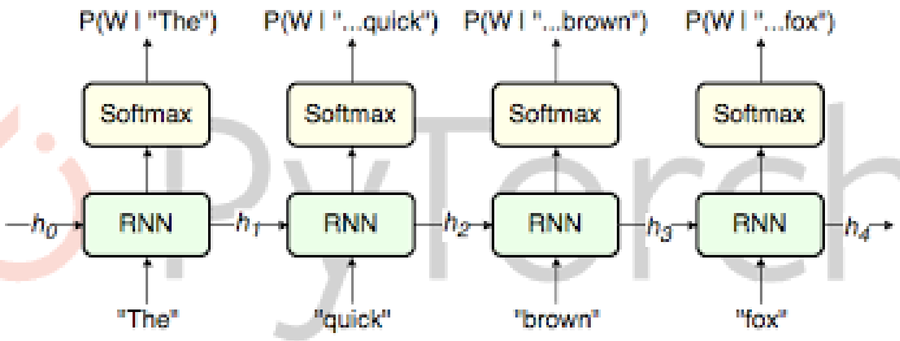




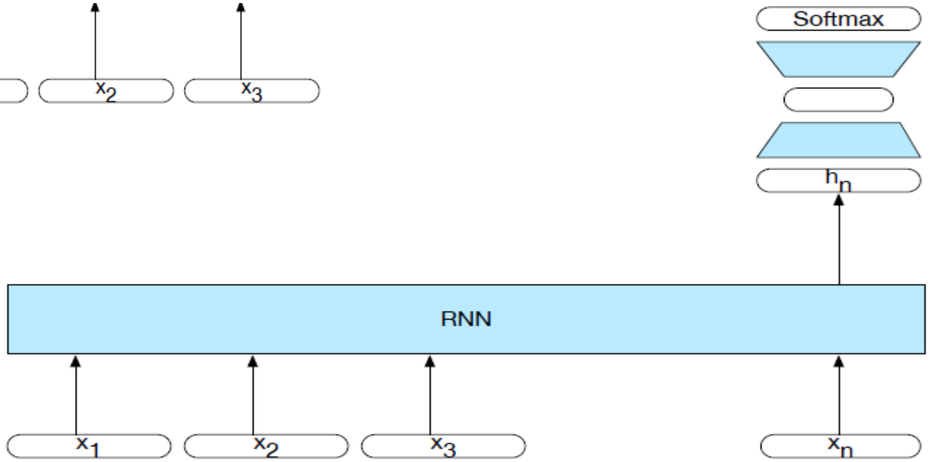
RNN Applications

- Language Modeling

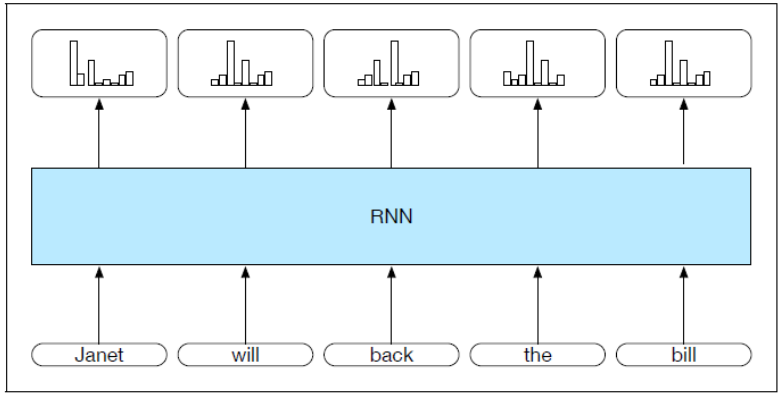




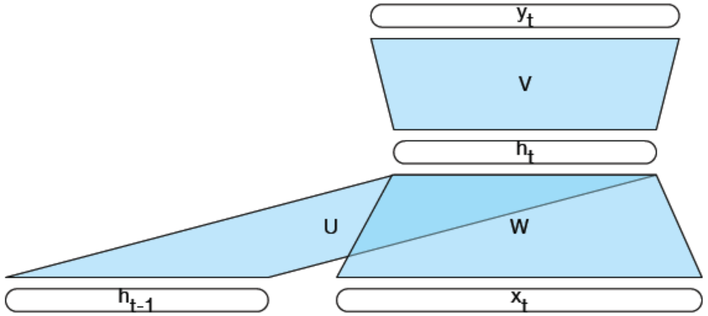
- Sequence Classification (Sentiment, Topic, intent, ..)



- Sequence to Sequence

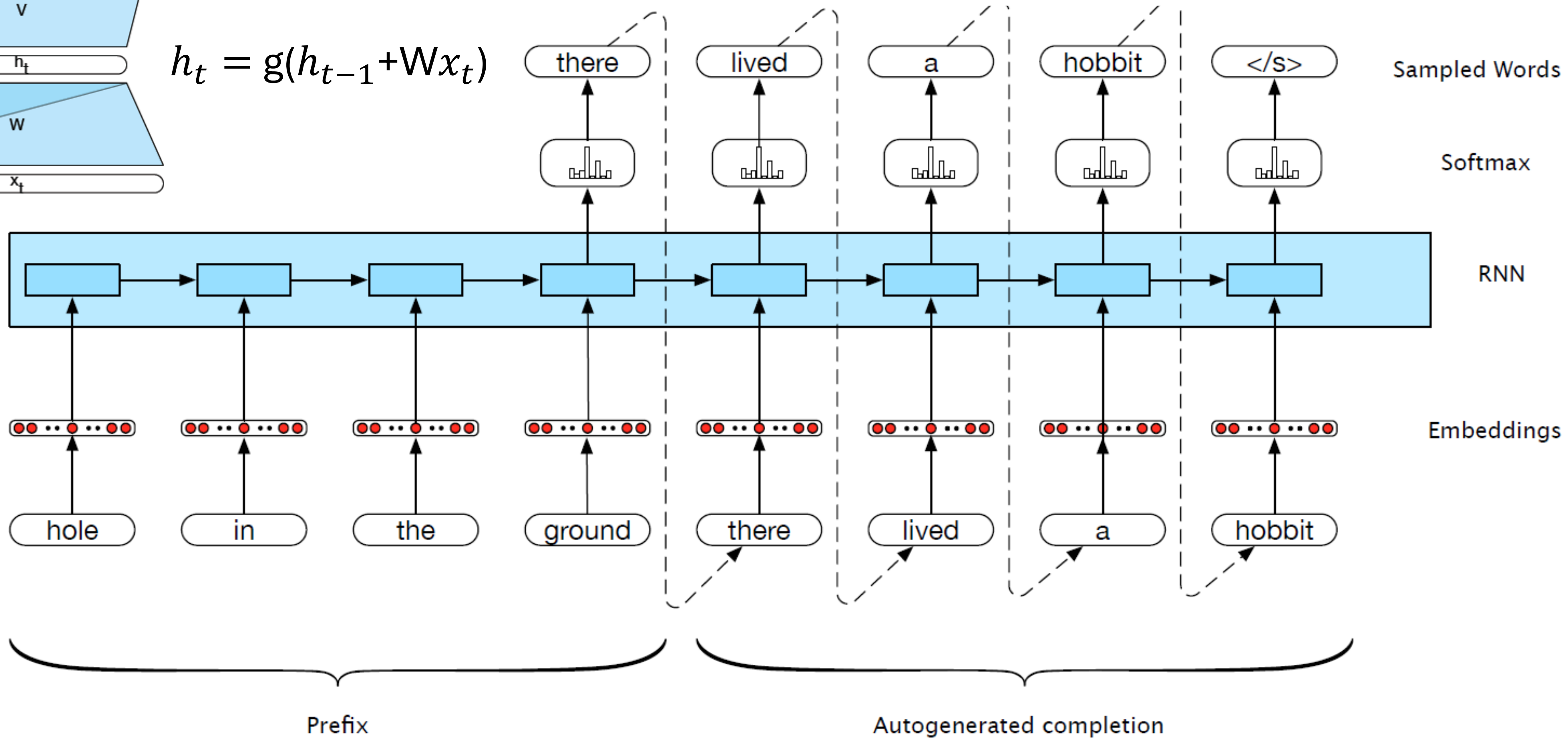


Sentence Completion using an RNN



$$y_t = \text{softmax}(Vh_t)$$

$$h_t = g(h_{t-1} + Wx_t)$$



- **Trained Neural Language Model** can be used to generate novel sequences
- Or to **complete** a given sequence (until end of sentence token <\s> is generated)

Extending (autoregressive) generation to Machine Translation

Autoregressive: word generated at each time step is conditioned on word from previous step.

- Training data are parallel text e.g., English / French

  there lived a hobbit *vivait un hobbit*

  ……..

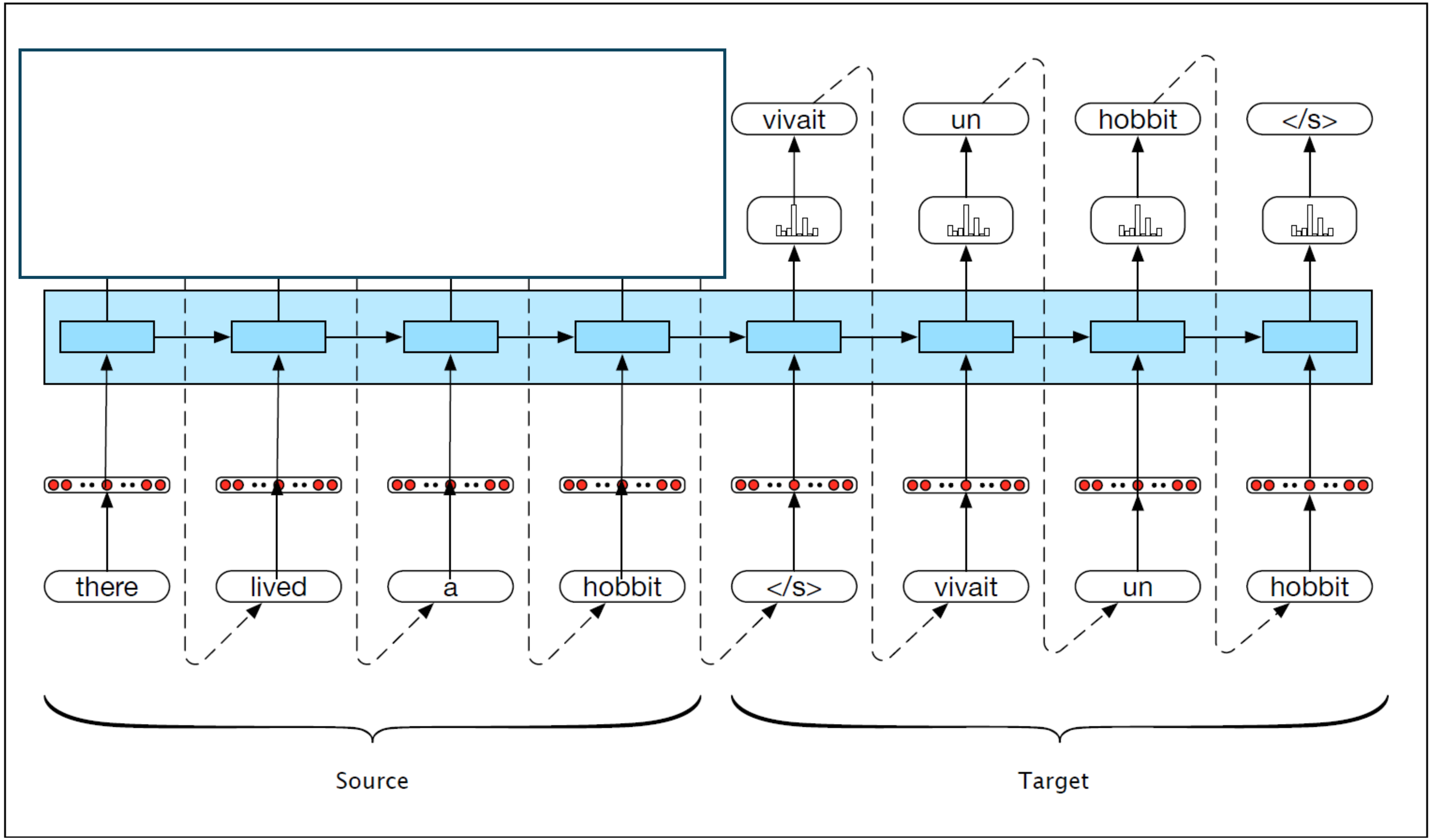
- Build an RNN language model on the concatenation of source and target

  there lived a hobbit <\s> *vivait un hobbit* <\s>

  ……..

Extending (autoregressive) generation to Machine Translation

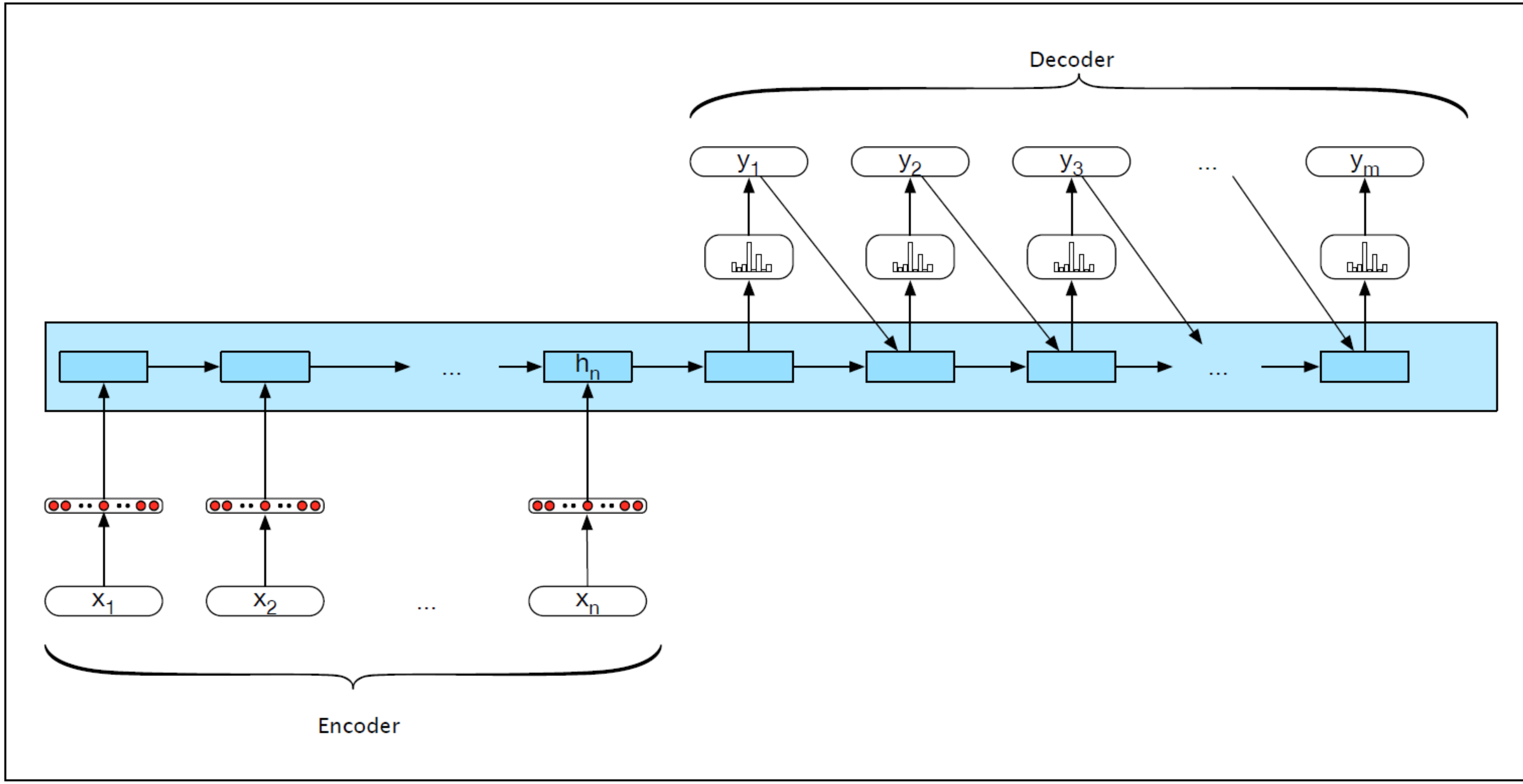
- Translation as Sentence Completion



(simple) **E**ncoder **D**ecoder Networks

Limiting design choices

- **E** and **D** assumed to have the same internal structure (here RNNs)
- Final state of the **E** is the only context available to **D**
- this context is only available to **D** as its initial hidden state.

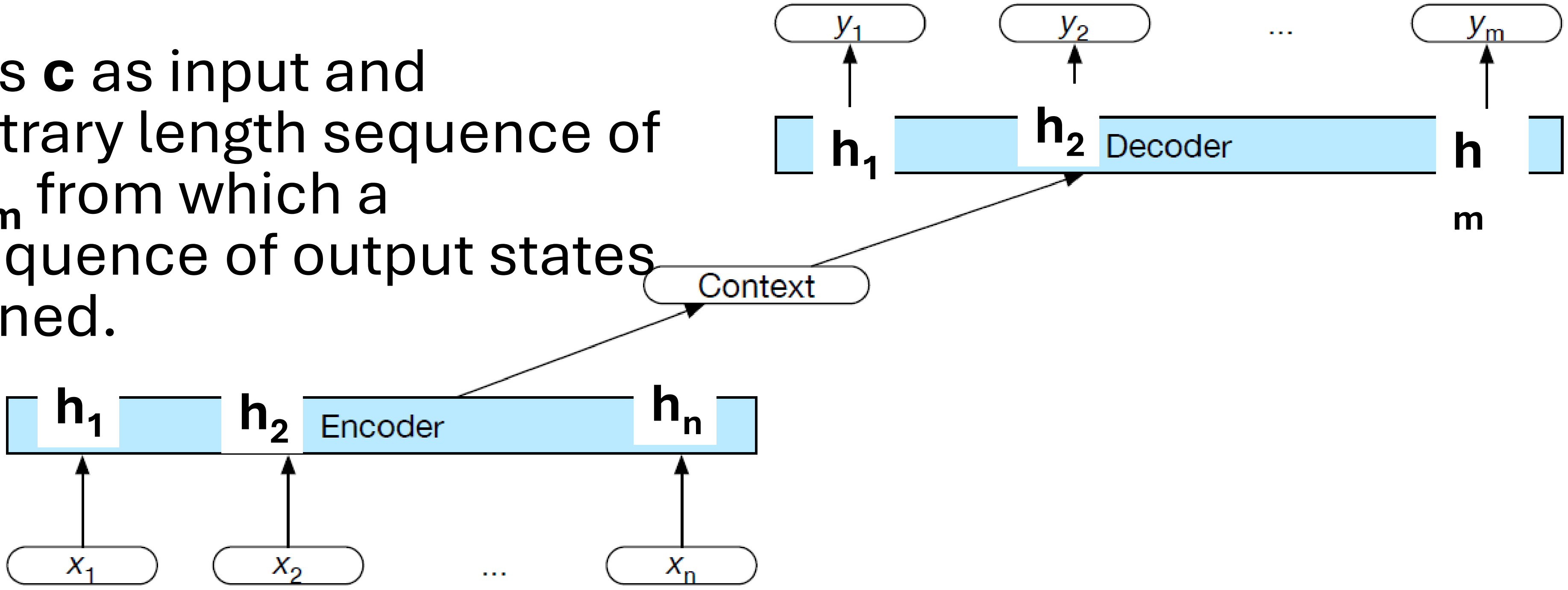


- Encoder generates a contextualized representation of the input (last state).
- Decoder takes that state and autoregressively generates a sequence of outputs

General Encoder Decoder Networks

Abstracting away from these choices

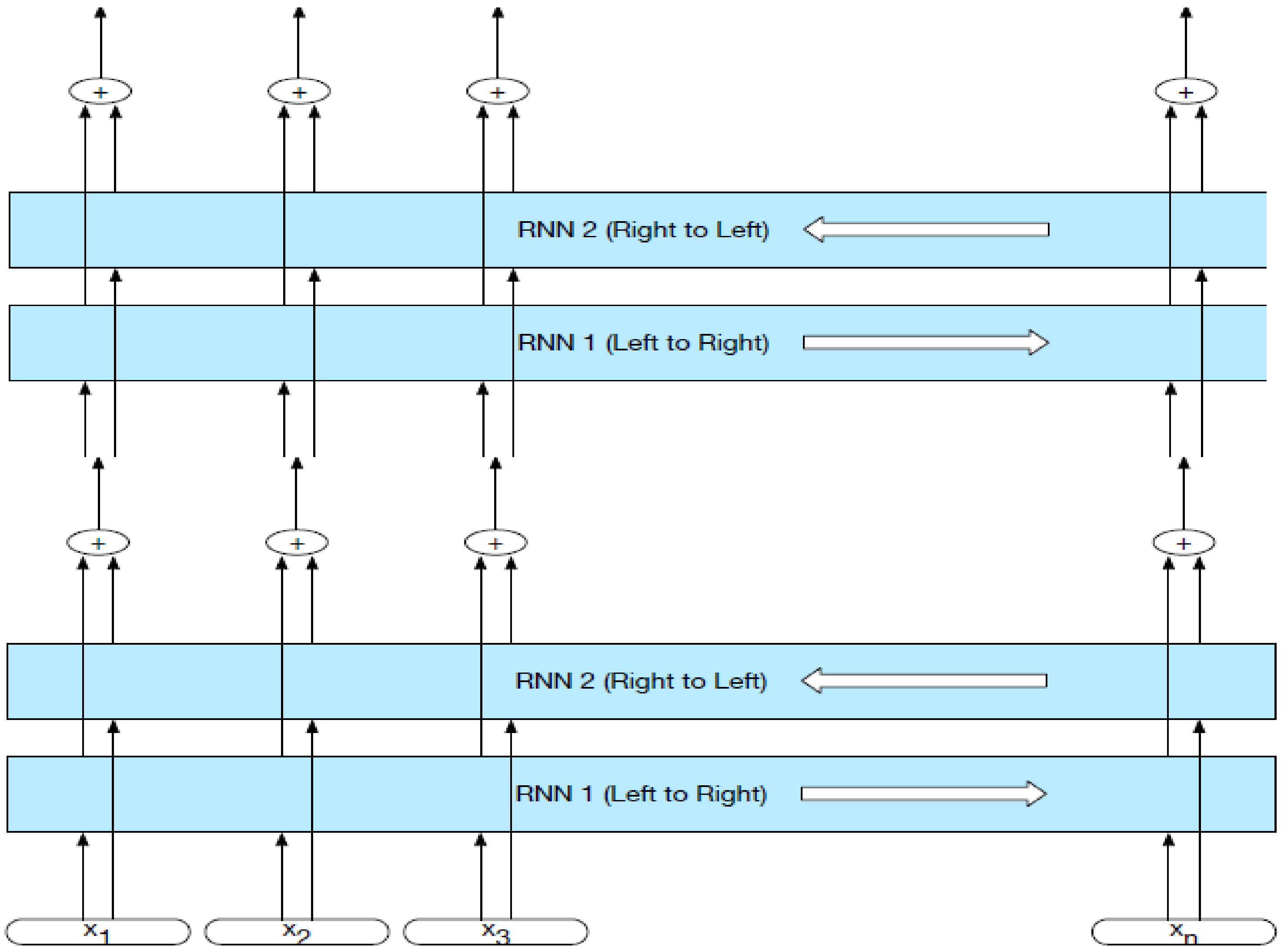
1. **Encoder**: accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}_{1:n}$
2. **Context vector c**: function of $\mathbf{h}_{1:n}$ and conveys the essence of the input to the decoder.
3. **Decoder**: accepts **c** as input and generates an arbitrary length sequence of hidden states $\mathbf{h}_{1:m}$ from which a corresponding sequence of output states $\mathbf{y}_{1:m}$ can be obtained.



Popular architectural choices: Encoder

Widely used encoder design: **stacked Bi-LSTMs**

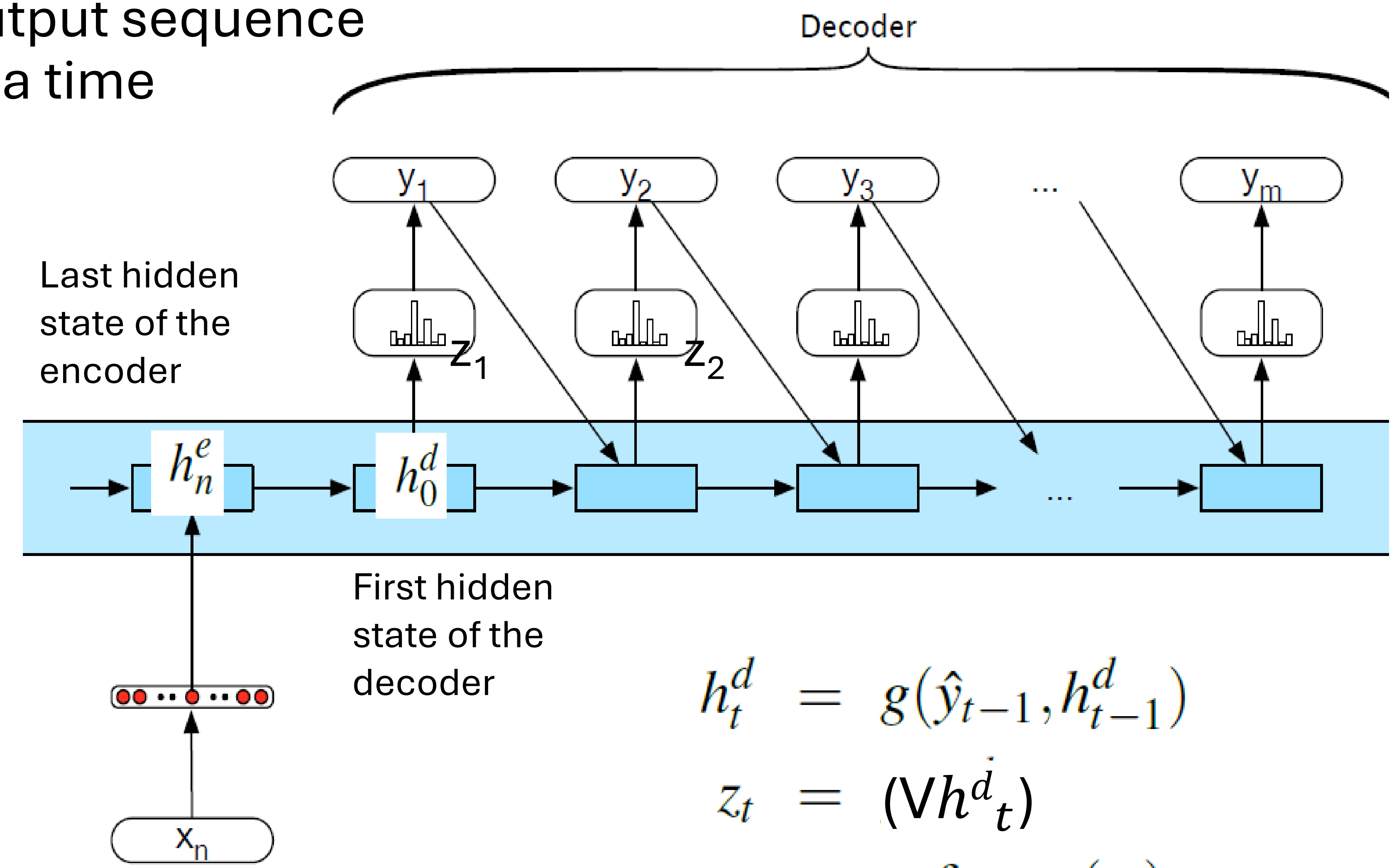
- Contextualized representations for each time step**: hidden states from top layers** from the forward and backward passes



Decoder Basic Design

- produce an output sequence an element at a time

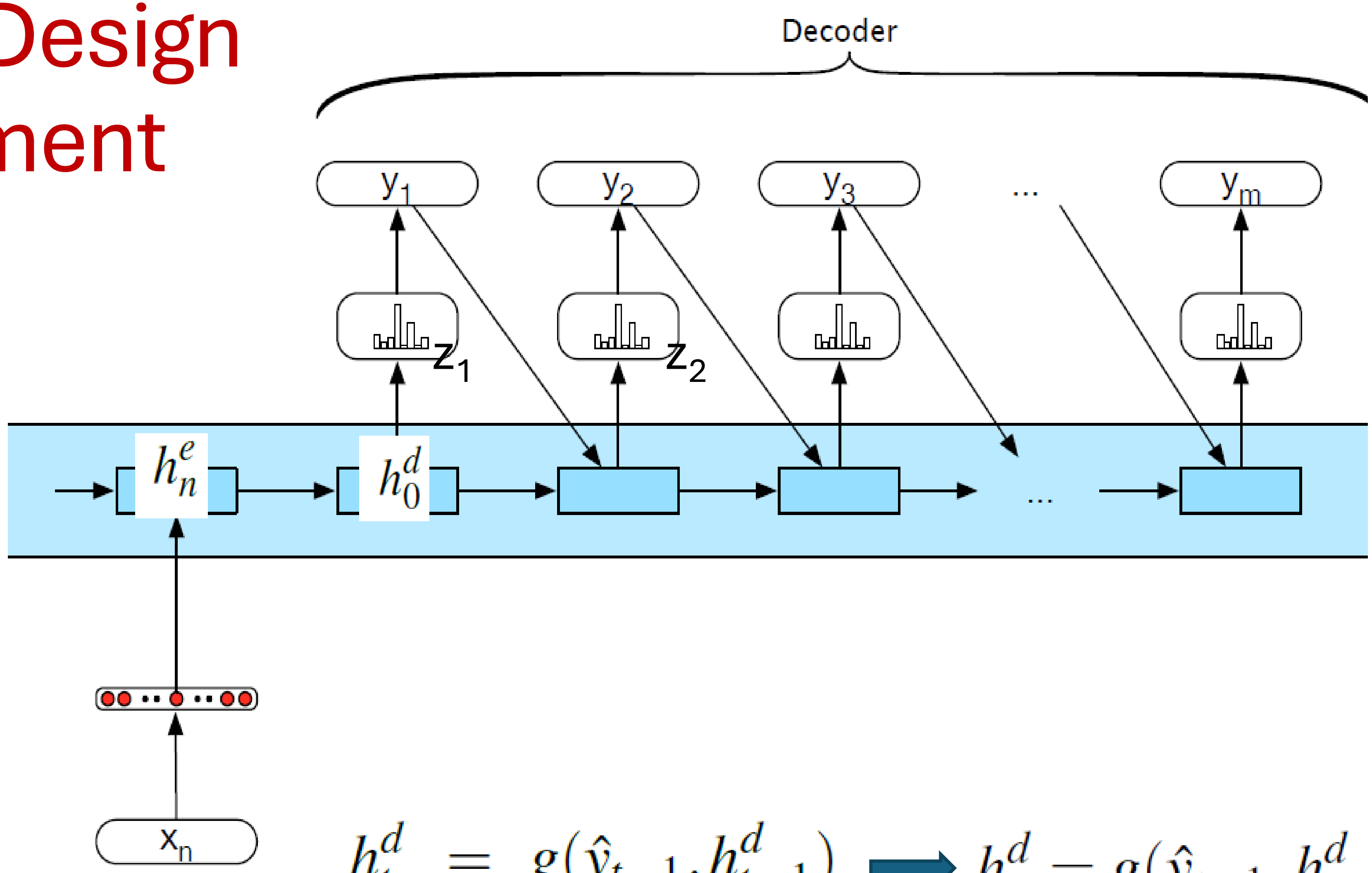
$$c = h_n^e$$
$$h_0^d = c$$



$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d)$$
$$z_t = (V h^d{}_t)$$
$$y_t = \text{softmax}(z_t)$$

Decoder Design Enhancement

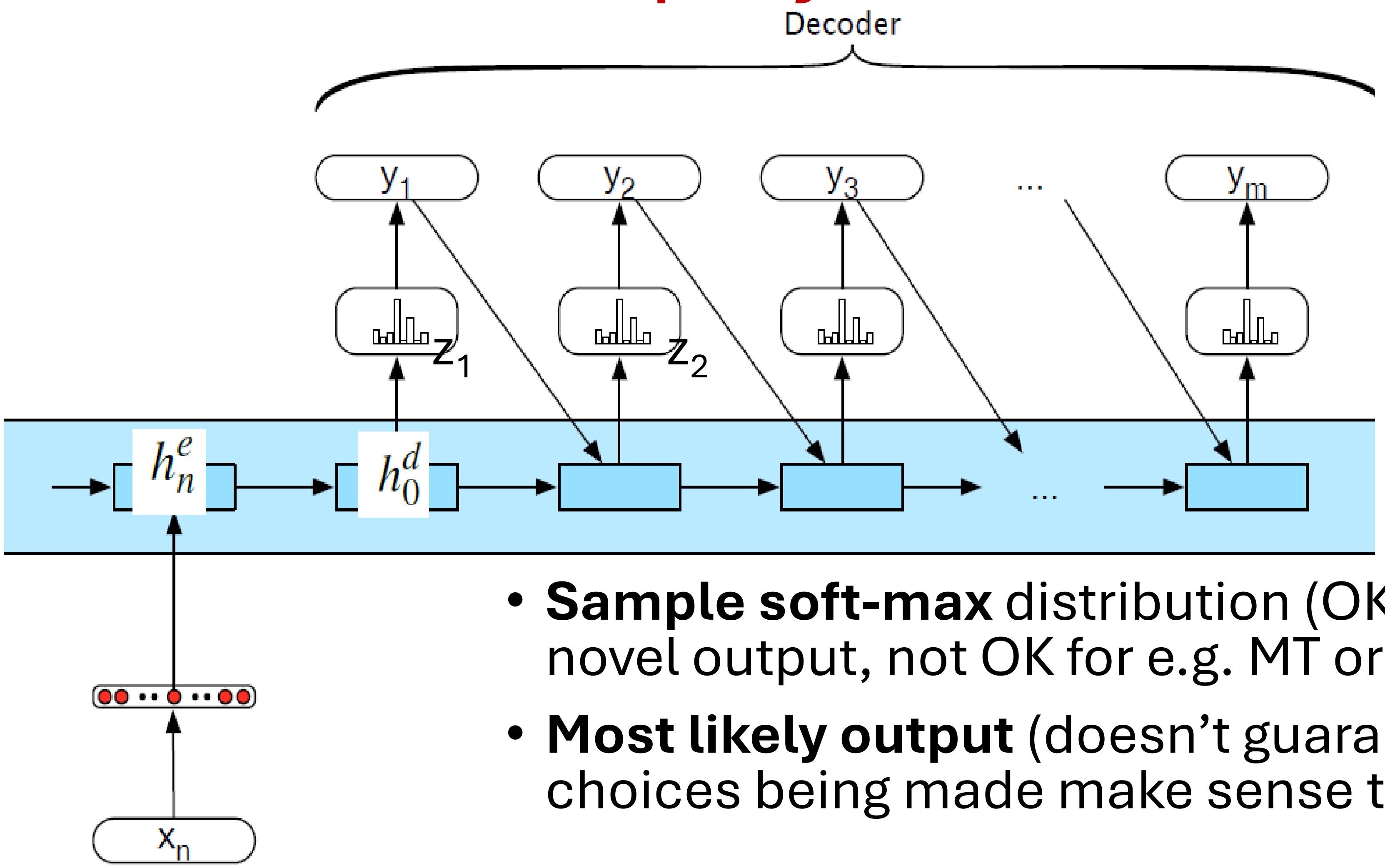
$$
\begin{aligned}
c &= h_n^e \\
h_0^d &= c
\end{aligned}
$$



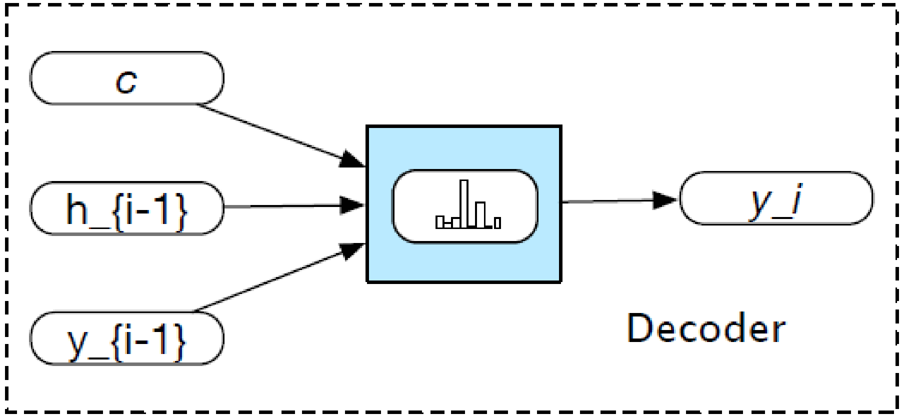
$$
\begin{aligned}
h_t^d &= g(\hat{y}_{t-1}, h_{t-1}^d) \longrightarrow h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c) \\
z_t &= f(h_t^d) \\
y_t &= \text{softmax}(z_t)
\end{aligned}
$$

Context available at each step of decoding

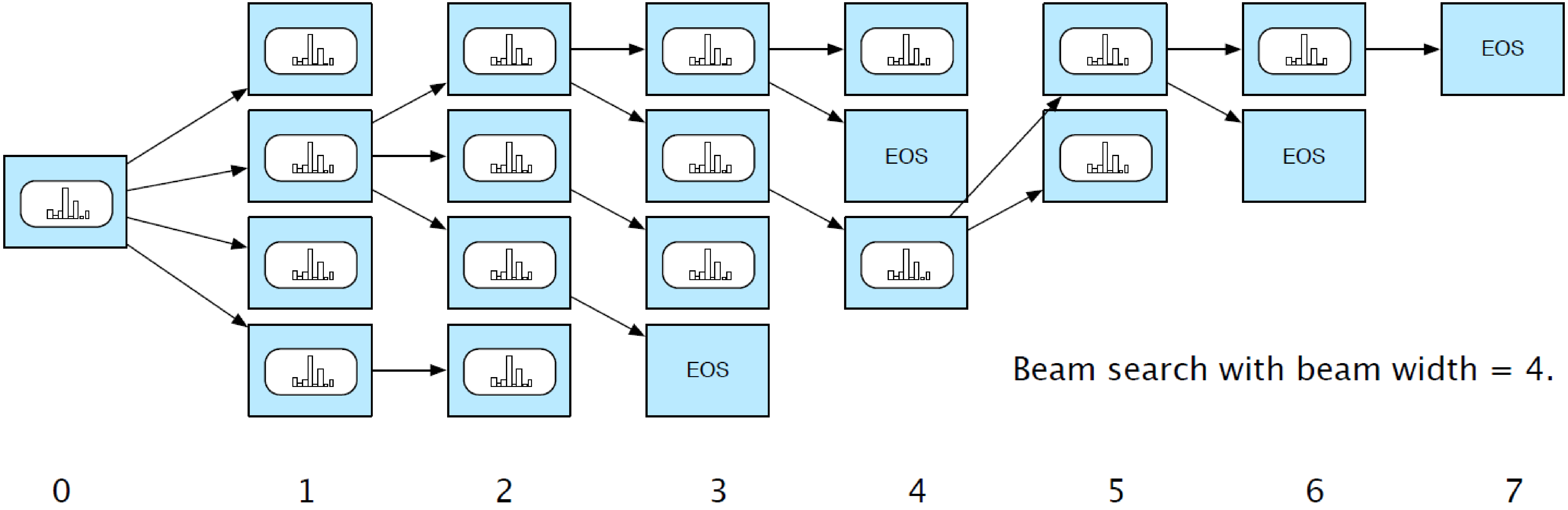
Decoder: How output *y* is chosen



- **Sample soft-max** distribution (OK for generating novel output, not OK for e.g. MT or Summ)
- **Most likely output** (doesn't guarantee individual choices being made make sense together)



- 4 most likely "words" decoded from initial state
- Feed each of those in decoder and keep most likely 4 sequences of two words
- Feed most recent word in decoder and keep most likely 4 sequences of three words ....
- When EOS is generated. Stop sequence and reduce Beam by 1



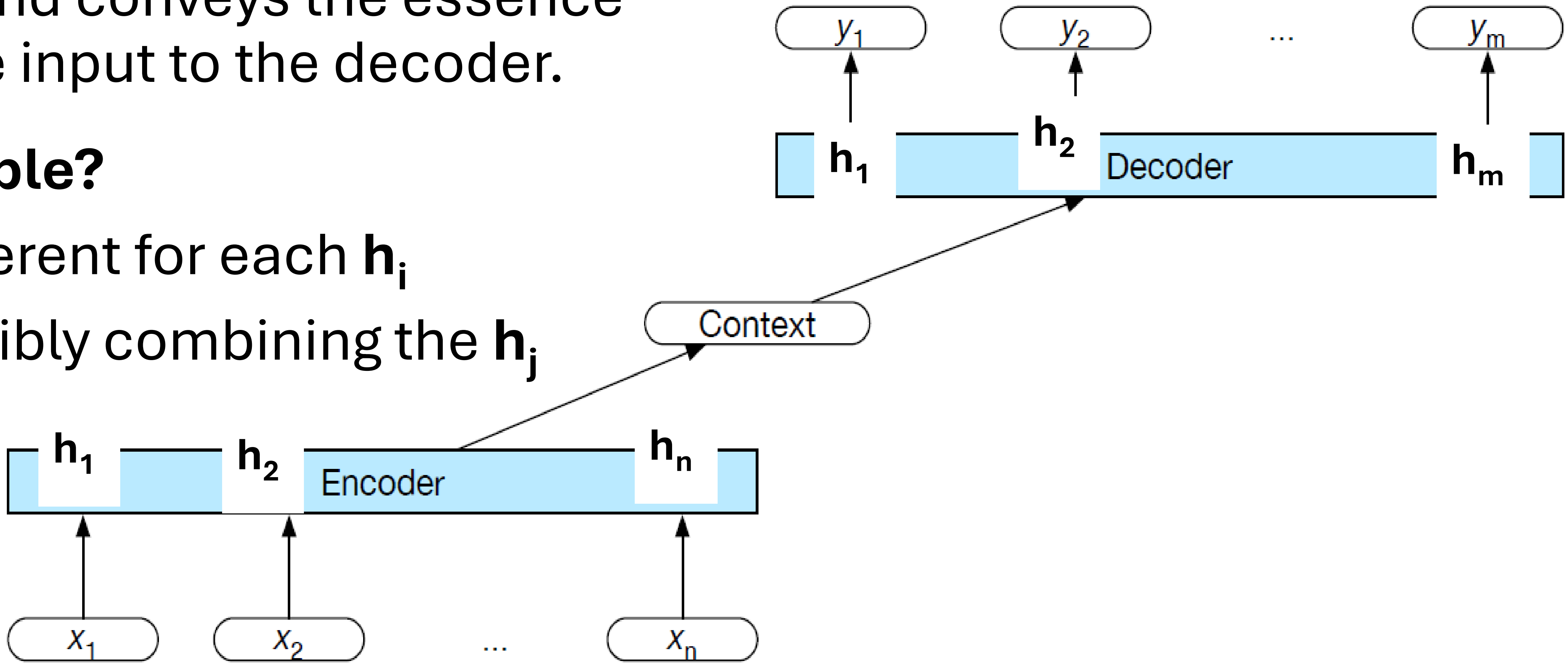
Beam search with beam width = 4.

Flexible context: Attention

Context vector c: function of $\mathbf{h}_{1:n}$ and conveys the essence of the input to the decoder.

Flexible?

- Different for each $\mathbf{h}_i$
- Flexibly combining the $\mathbf{h}_j$



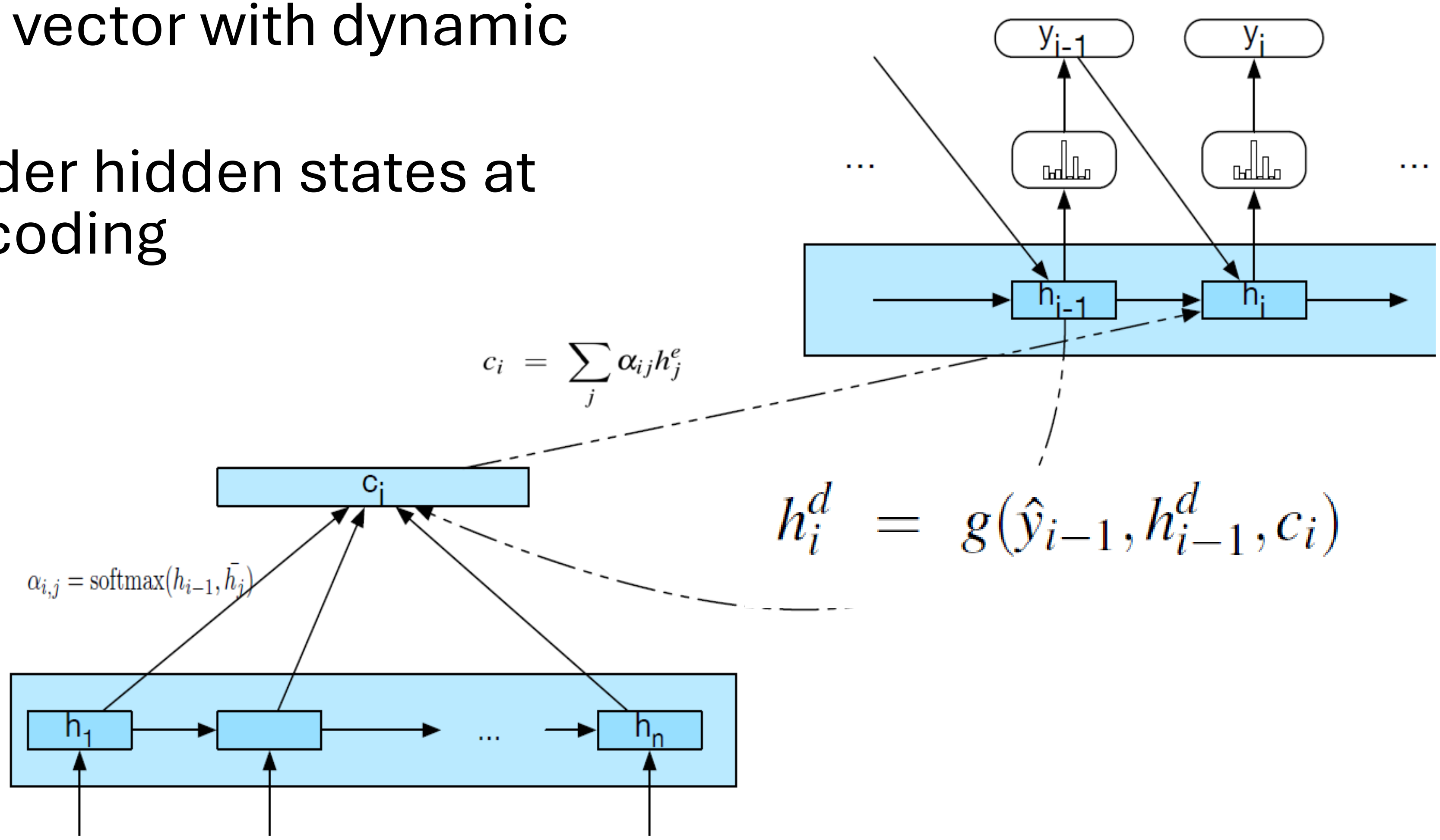
Attention (1): dynamically derived context

- Replace static context vector with dynamic $\mathbf{c_i}$
- derived from the encoder hidden states at each point i during decoding

Ideas:

- should be a linear combination of those states

$$c_i = \sum_j \alpha_{ij} h_j^e$$



- α_{ij} should depend on ?

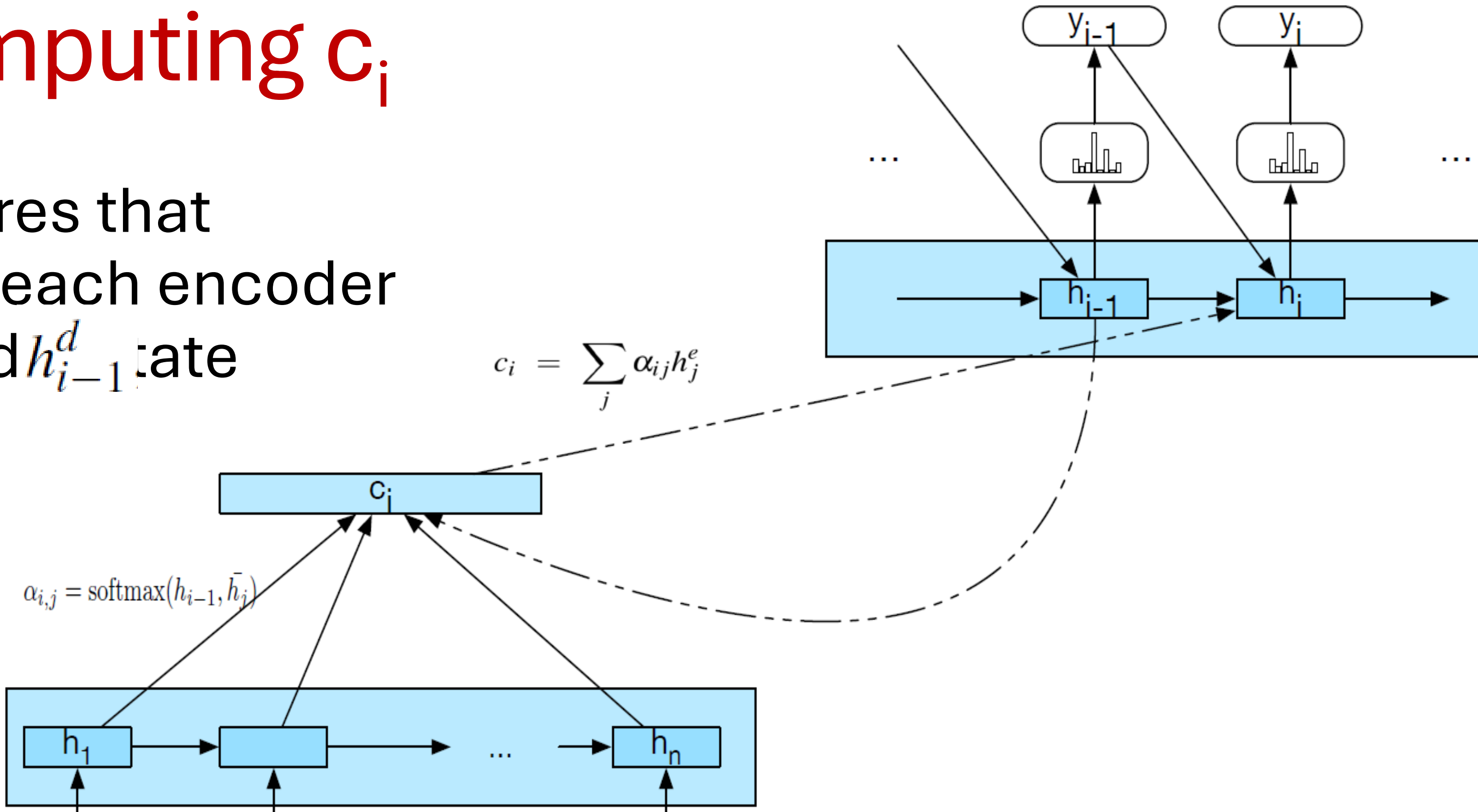
Attention (2): computing c_i

- Compute a vector of scores that capture the relevance of each encoder hidden state to the decod h^d_{i-1} state

$$score(h^d_{i-1}, h^e_j)$$

- Just the similarity

$$score(h^d_{i-1}, h^e_j) \; = \; h^d_{i-1} \cdot h^e_j$$



- Give network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application.

$$score(h^d_{i-1}, h^e_j) \; = \; h^d_{t-1} W_s h^e_j$$

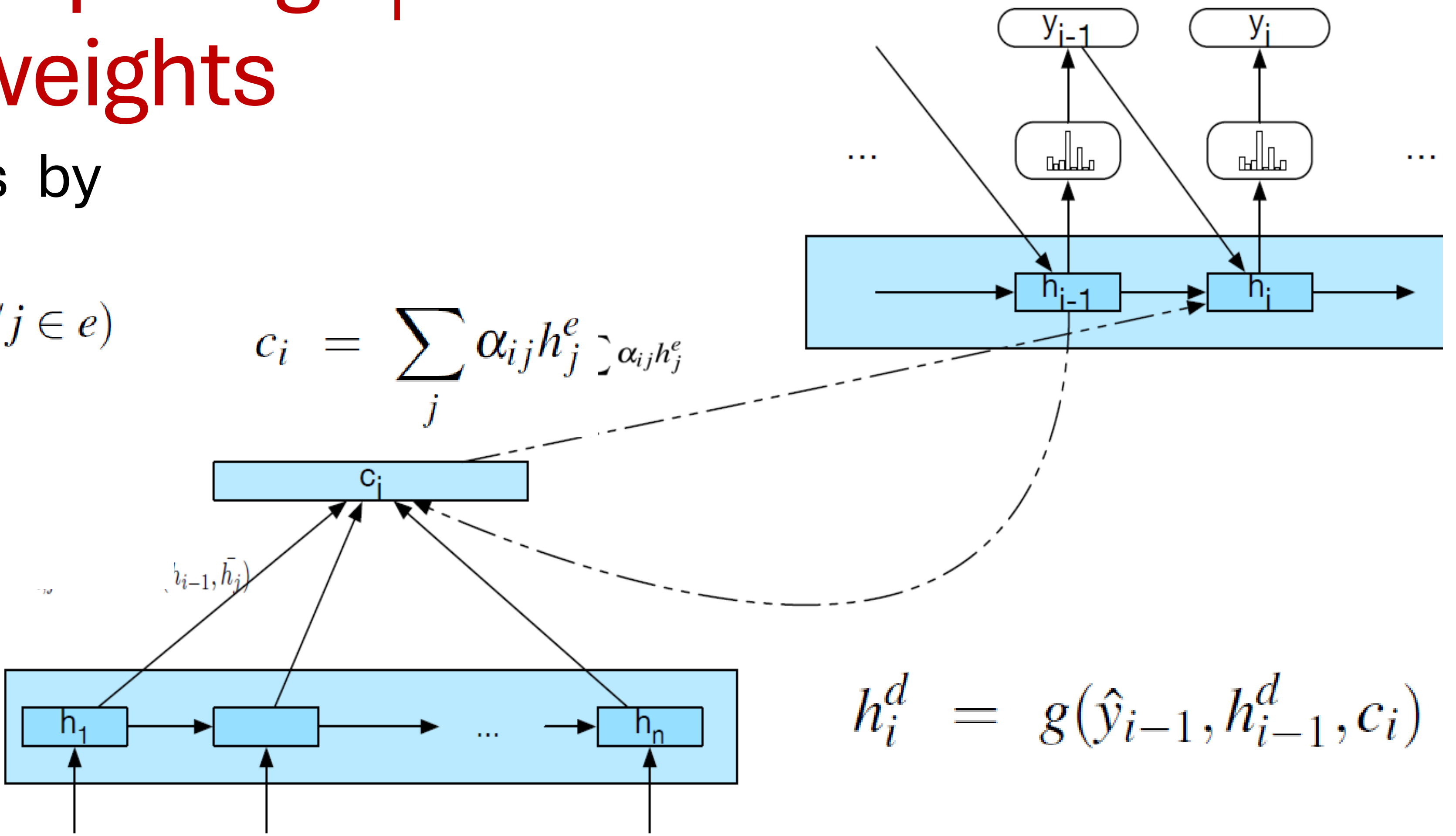
Attention (3): computing c_i From scores to weights

- Create vector of weights by normalizing scores

$$\alpha_{ij} = \text{softmax}(score(h^d_{i-1}, h^e_j)\ \forall j \in e)$$

$$= \frac{exp(score(h^d_{i-1}, h^e_j)}{\sum_k exp(score(h^d_{i-1}, h^e_k))}$$

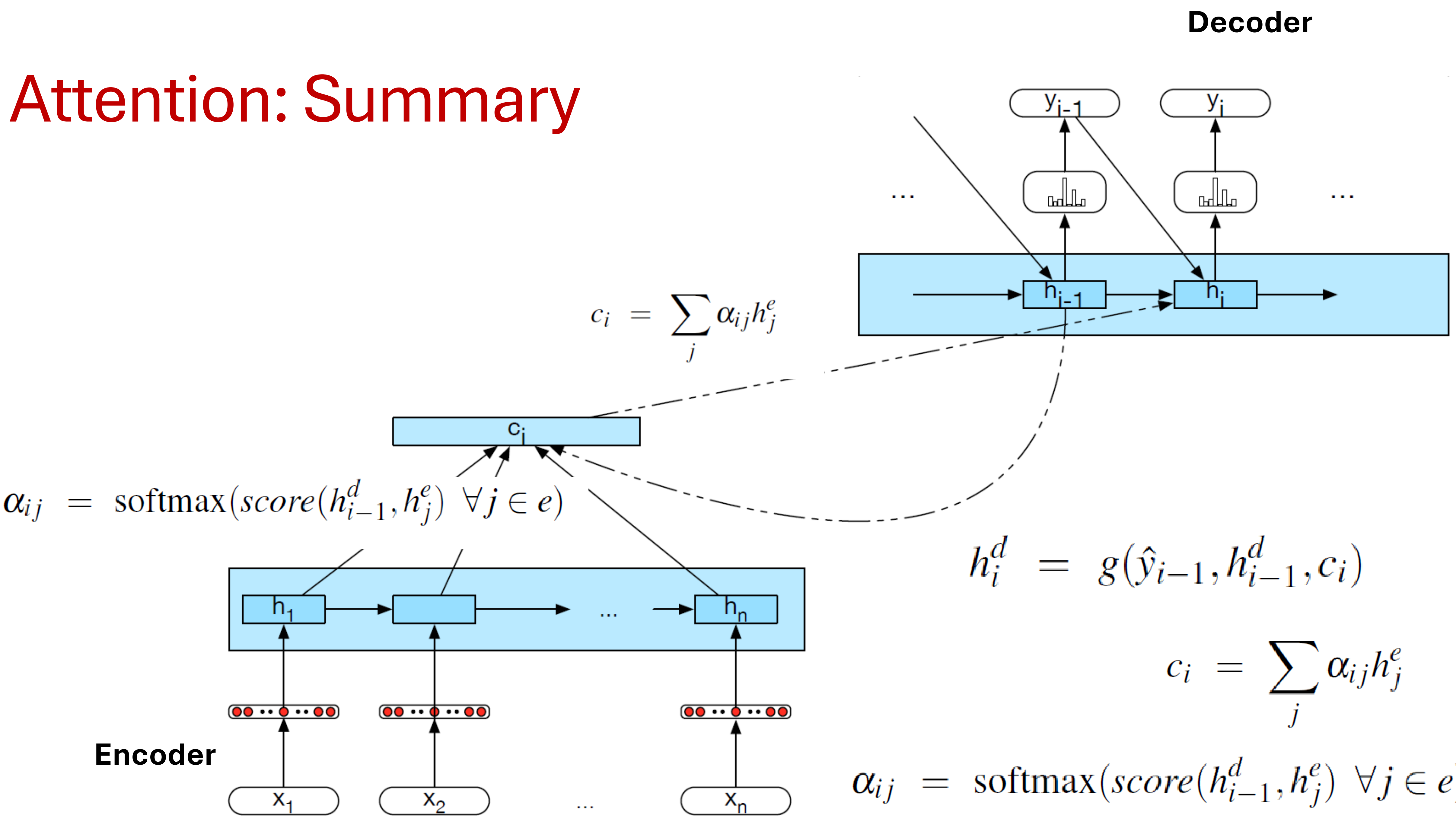
$$c_i = \sum_j \alpha_{ij} h^e_j$$



$$h^d_i = g(\hat{y}_{i-1}, h^d_{i-1}, c_i)$$

- **Goal achieved**: compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states.

Attention: Summary

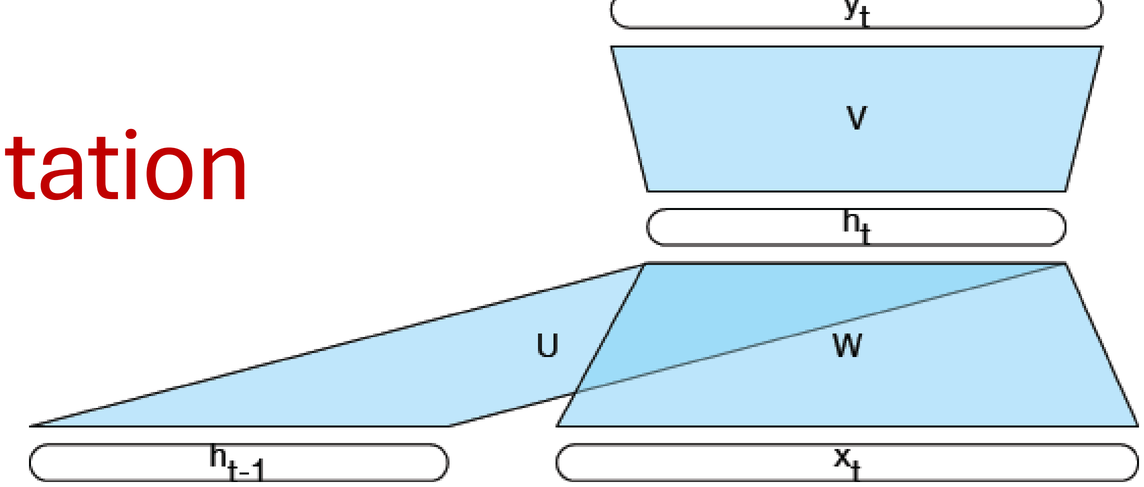


$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

$$c_i = \sum_j \alpha_{ij} h_j^e$$

$$\alpha_{ij} = \mathrm{softmax}(score(h_{i-1}^d, h_j^e) \; \forall j \in e)$$

Explain Y. Goldberg different notation



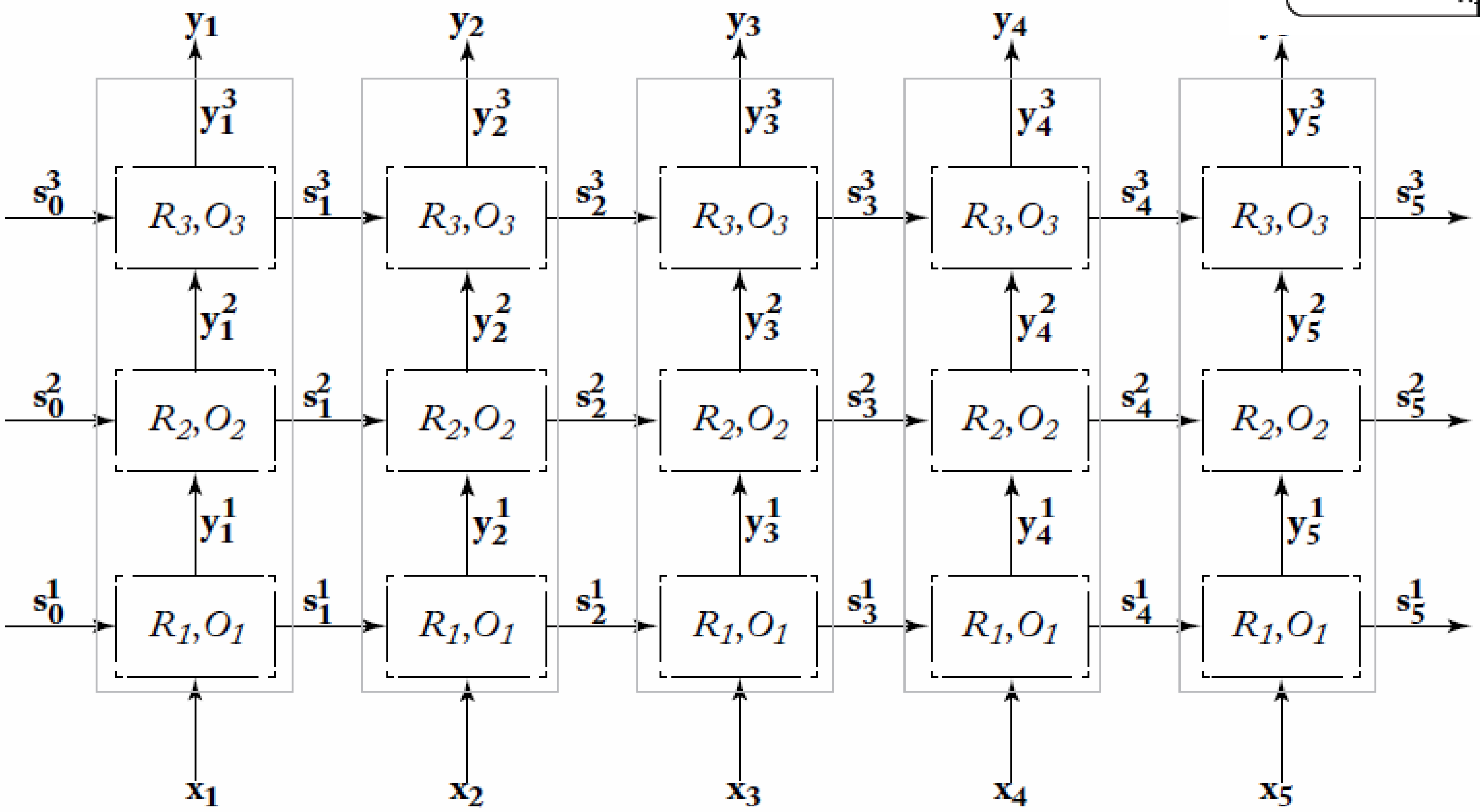


Figure 14.7: A three-layer ("deep") RNN architecture.

Transformers (Attention is all you need 2017)

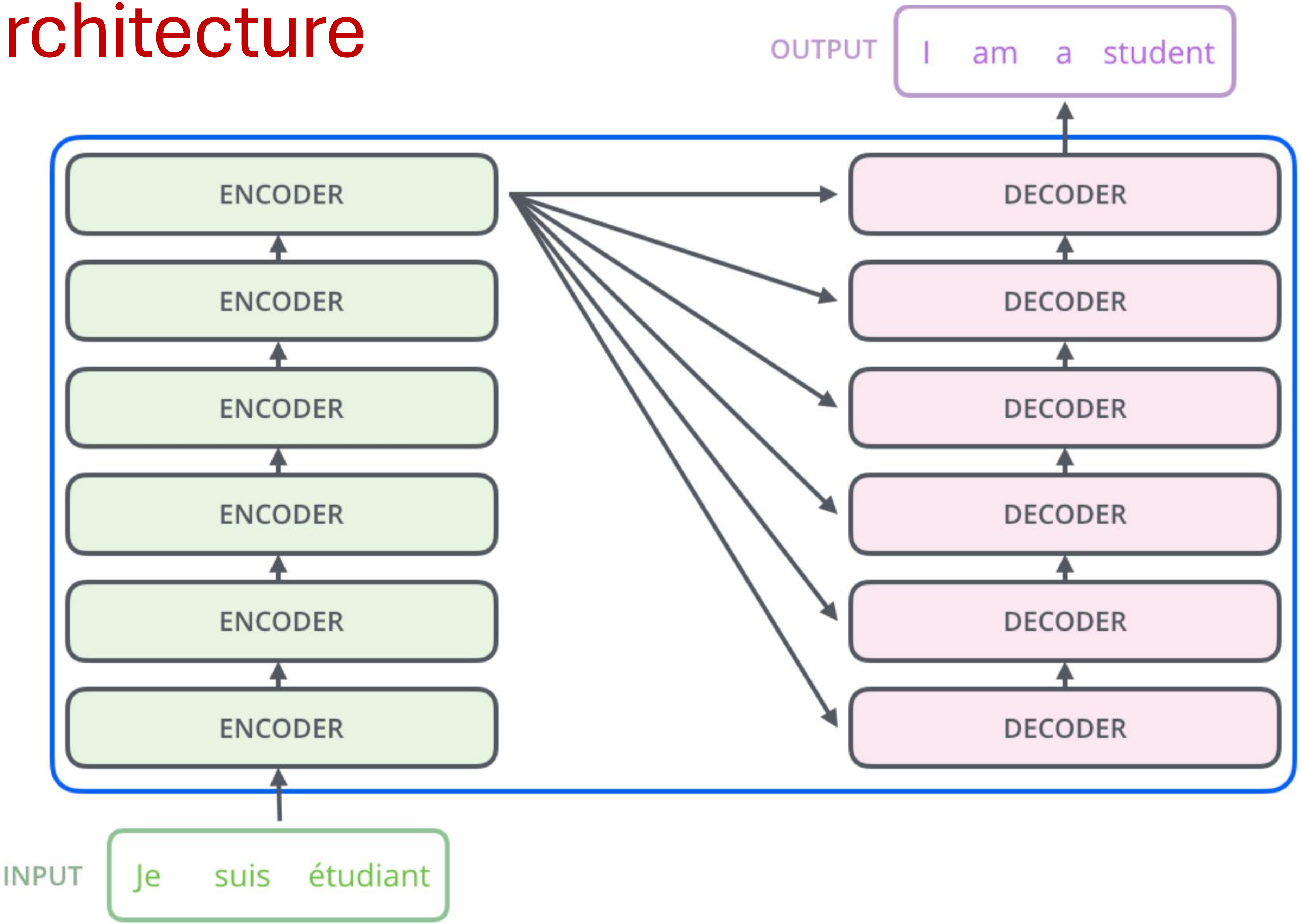
- **Just an introduction:** These are two valuable resources to learn more details on the architecture and implementation

- https://nlp.seas.harvard.edu/annotated-transformer/

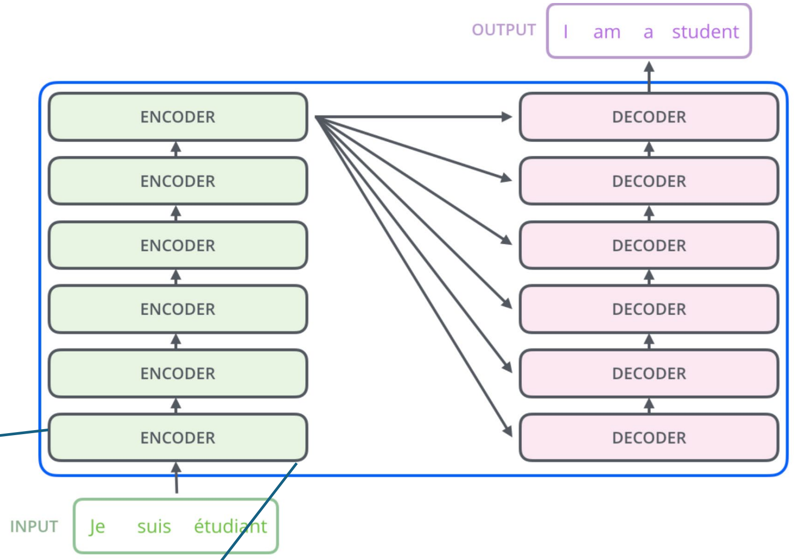
- https://jalammar.github.io/illustrated-transformer/ (slides come from this source)

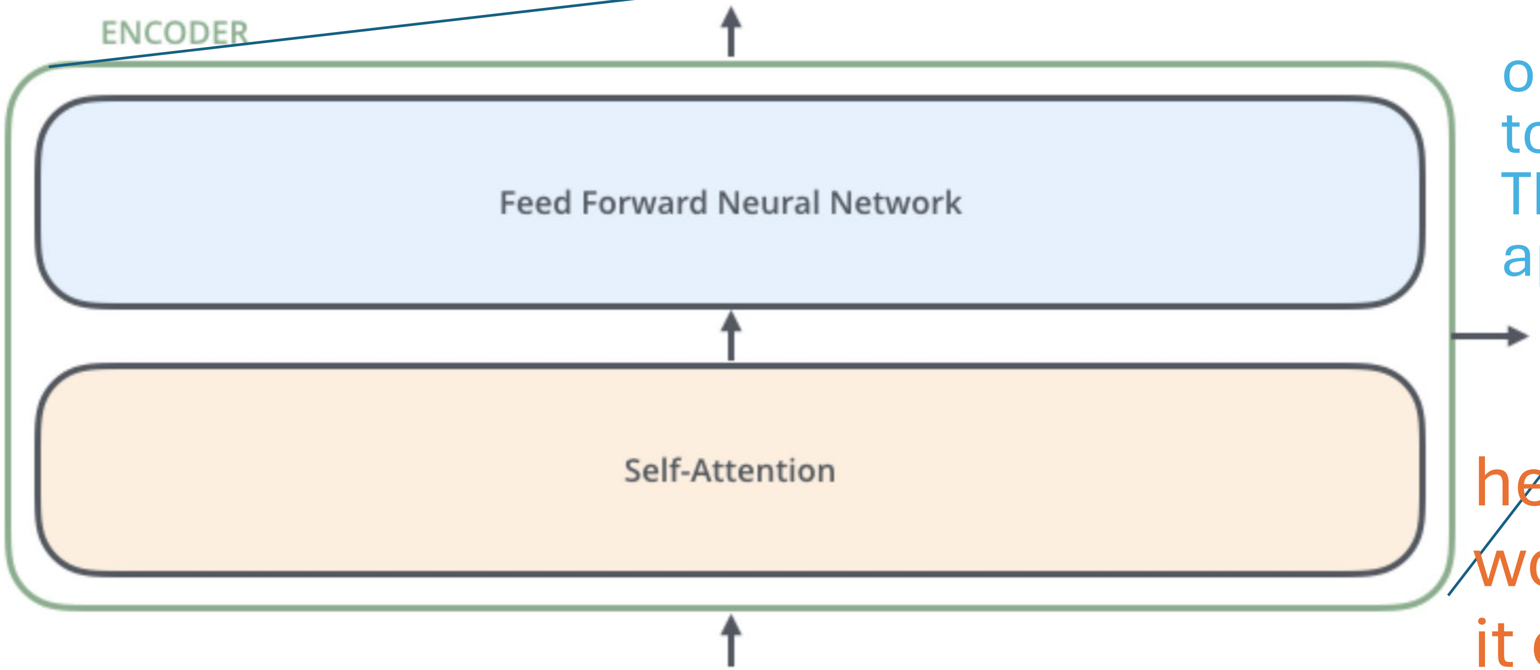
High-level architecture

- Will only look at the ENCODER(s) part in detail



The **encoders** are **all identical in structure** (yet they do not share weights). Each one is broken down into two sub-layers



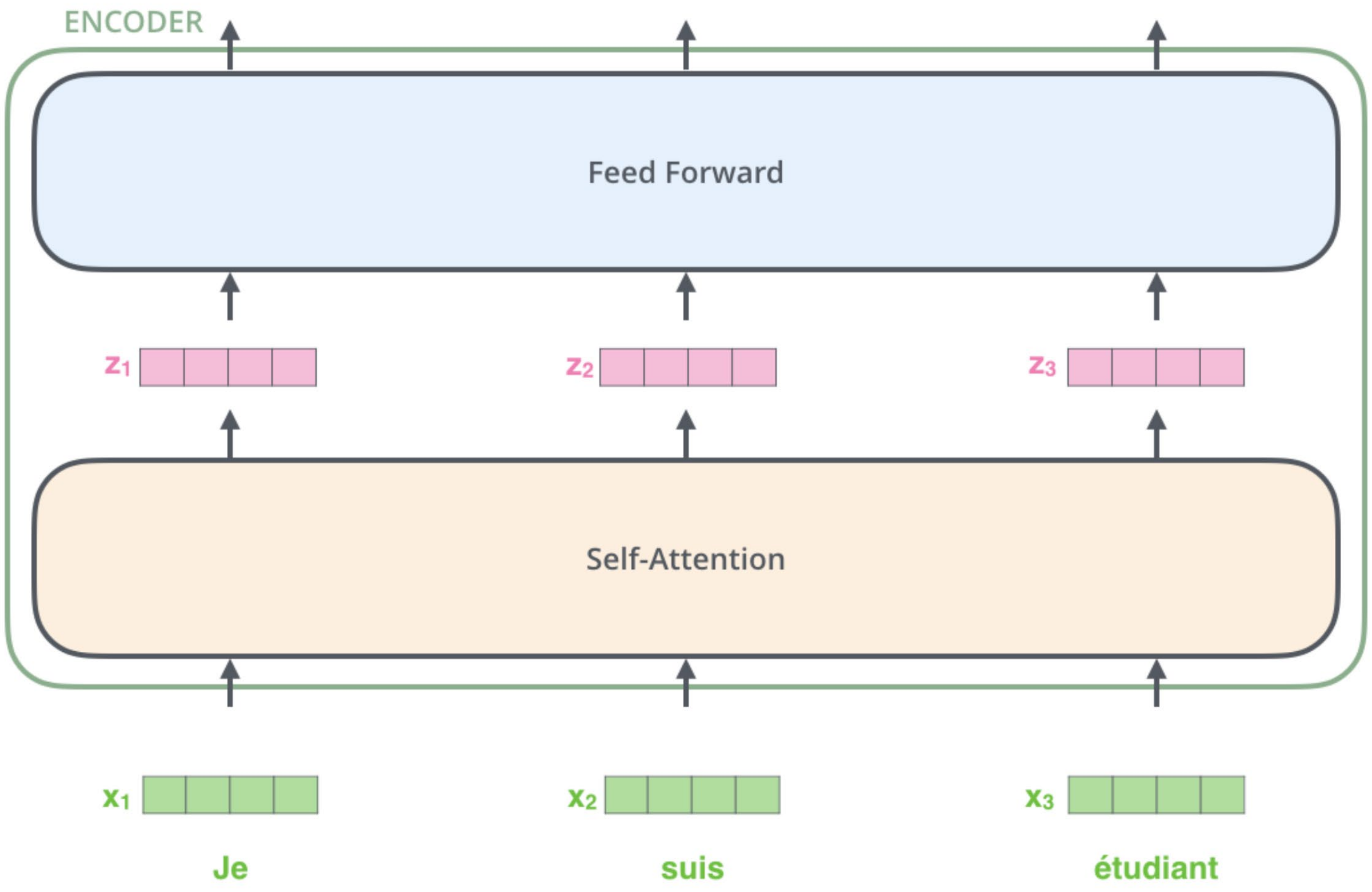


outputs of the self-attention are fed to a feed-forward neural network. The exact same one is independently applied to each position.

helps the encoder look at other words in the input sentence as it encodes a specific word.

Key property of Transformer: word in each position flows through its own path in the encoder.

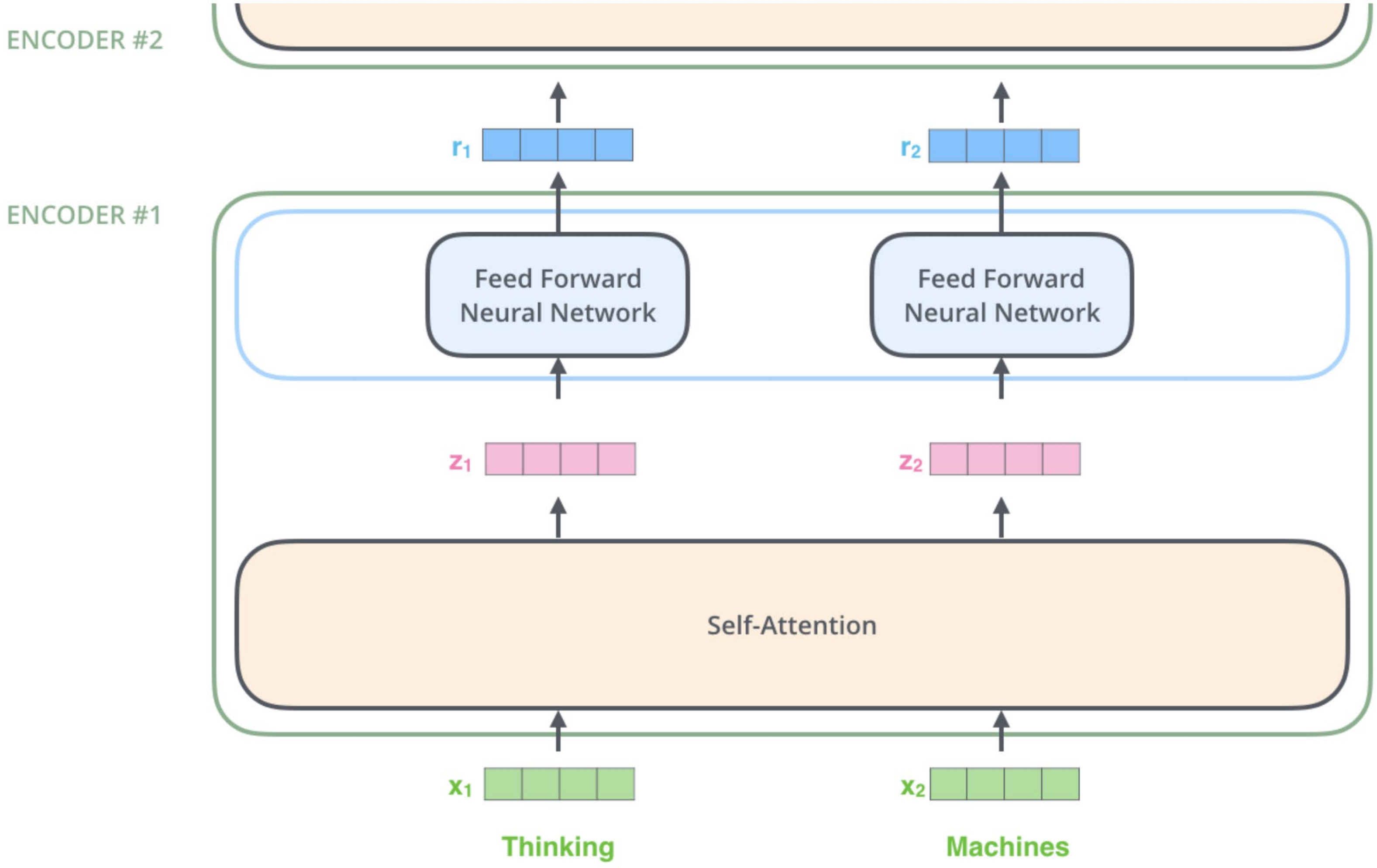
- There are dependencies between these paths in the self-attention layer.
- Feed-forward layer does not have those dependencies => various paths can be executed in parallel !



Word embeddings

Visually clearer on two words

- dependencies in self-attention layer.
- No dependencies in Feed-forward layer



Word embeddings

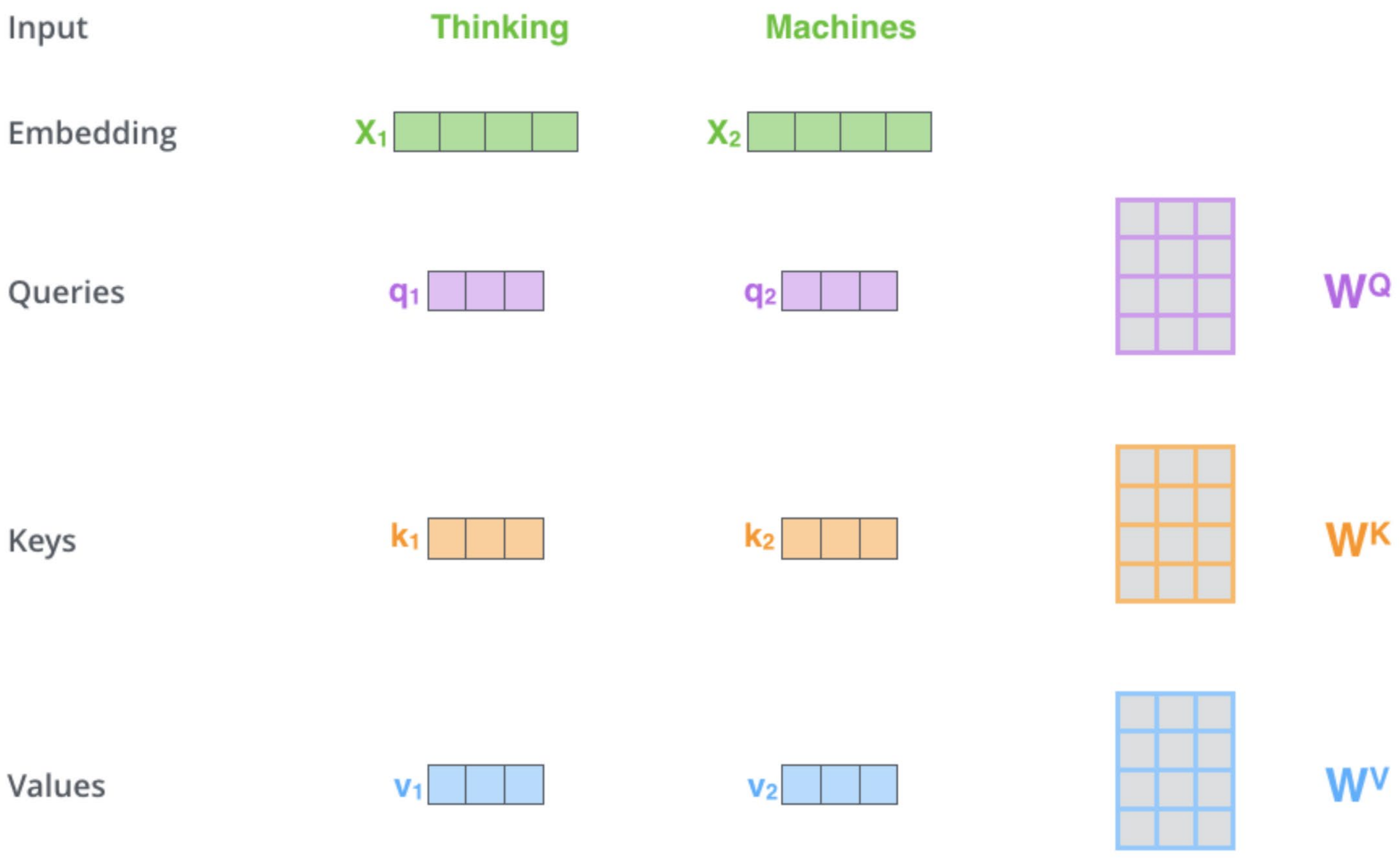
Self-Attention

While processing **each word** it allows to look at other positions in the input sequence for clues to build a better encoding for **this word**.

Step1: create three vectors from each of the encoder's input vectors:

Query, a Key, Value (typically smaller dimension).

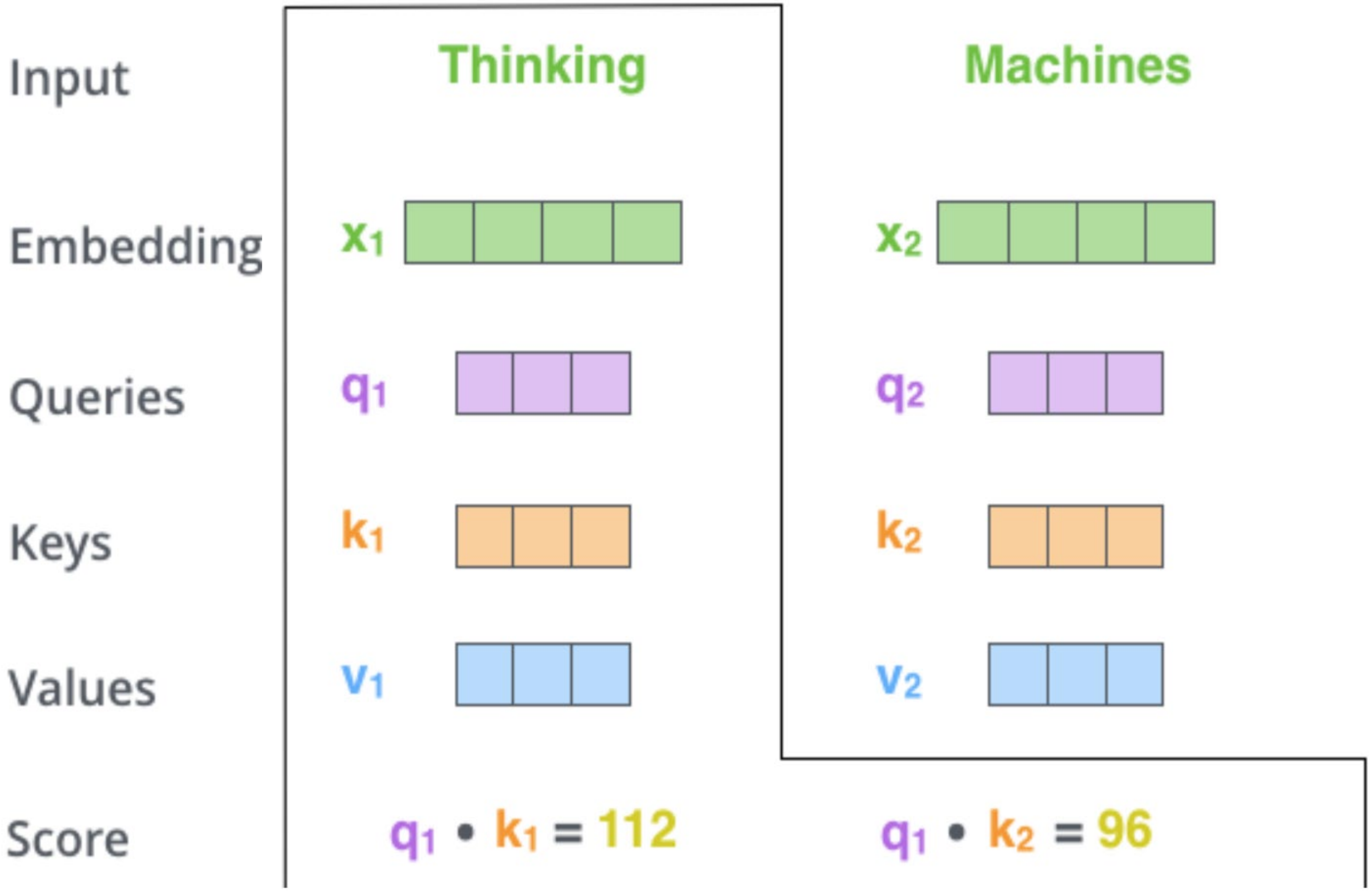
by multiplying the embedding by three matrices that we **trained** during the training process.



Self-Attention

Step 2: calculate a score (like we have seen for regular attention!) how much focus to place on other parts of the input sentence as we encode a word at a certain position.

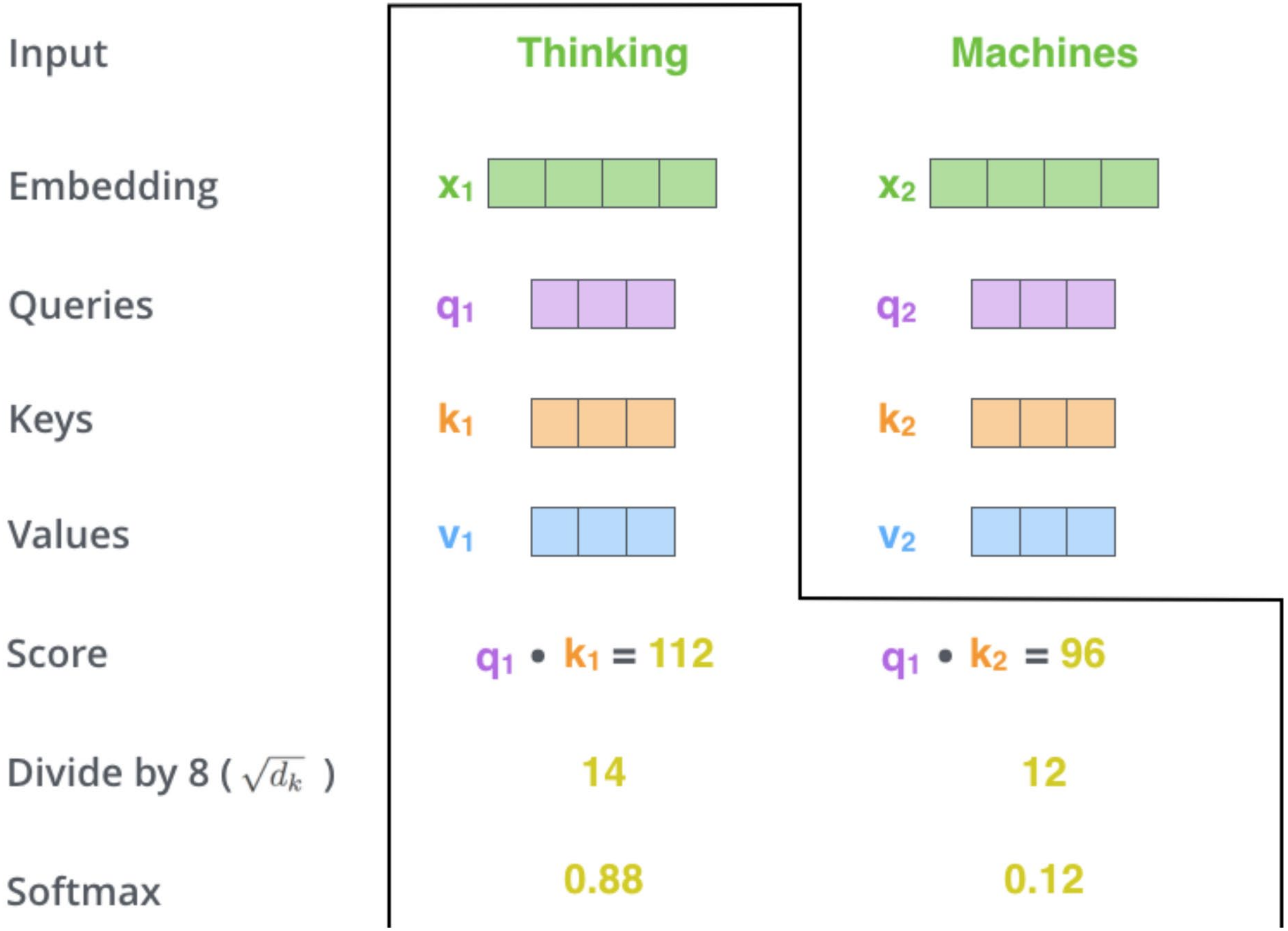
Take dot product of the query vector with the key vector of the respective word we're scoring.



E.g., Processing the self-attention for word "Thinking" in position #1, the first score would be the dot product of q1 and k1. The second score would be the dot product of q1 and k2.

Self Attention

- **Step 3** divide scores by the square root of the dimension of the key vectors (more stable gradients).
- **Step 4** pass result through a SoftMax operation. (all positive and add up to 1)



Intuition: SoftMax score determines how much each word will be expressed at this position.

Self Attention

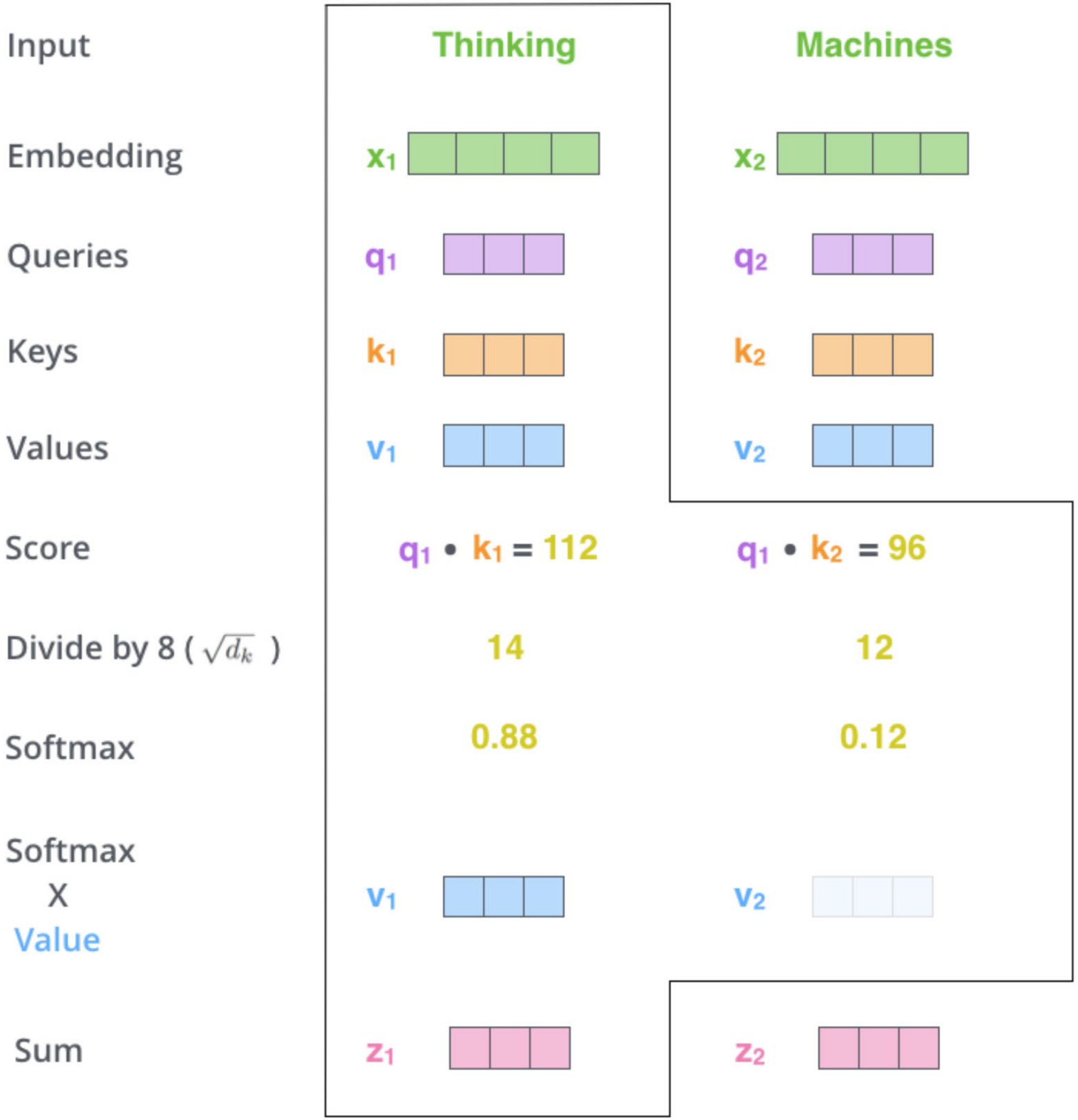
- **Step6** : sum up the weighted value vectors. This produces the output of the self-attention layer at this position

More details:

- What we have seen for a word is done **for all words** (using matrices)
- Need to **encode position** of words
- And improved using a mechanism called "**multi-headed**" attention

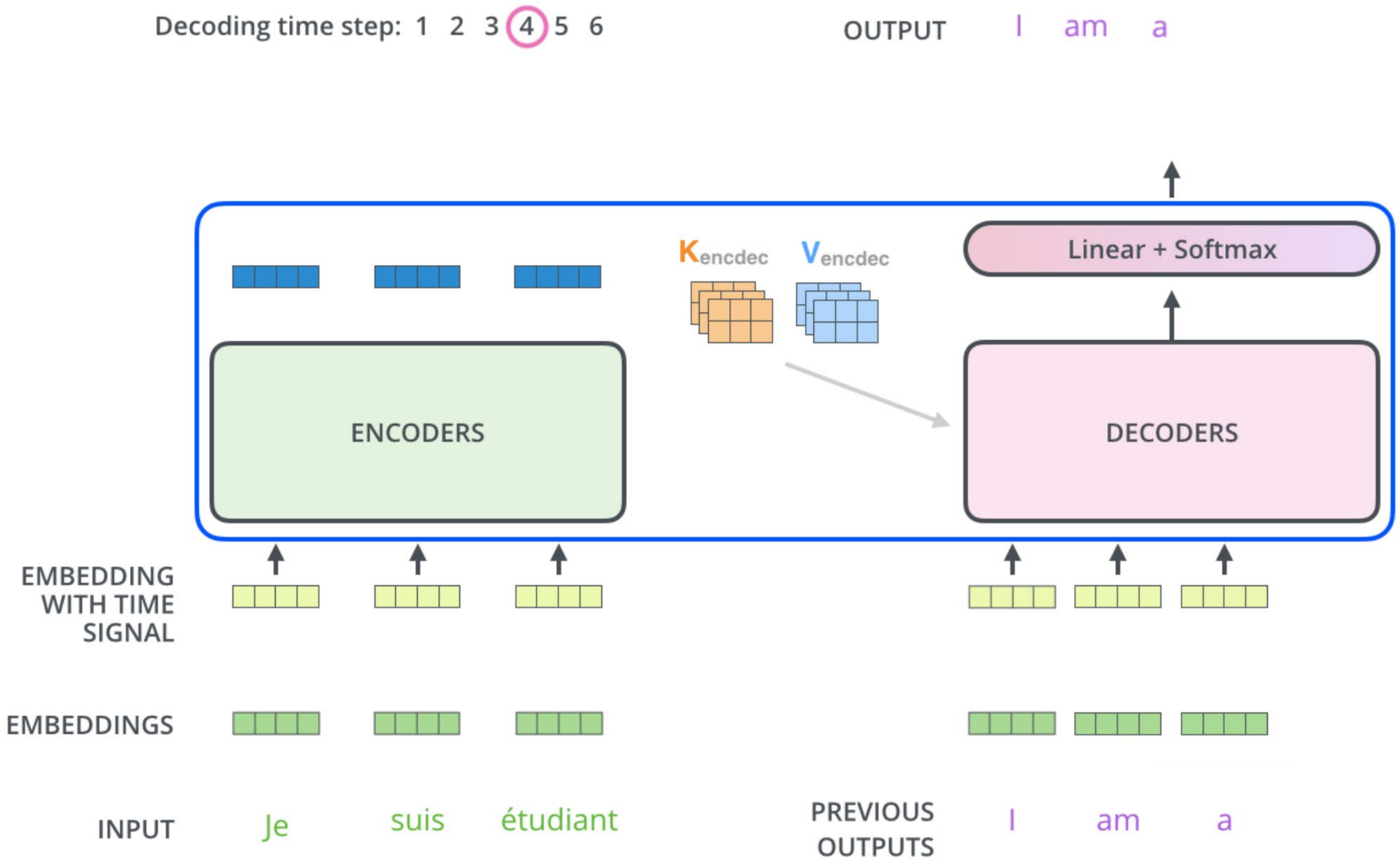
(kind of like multiple filters for CNN)

see https://jalammar.github.io/illustrated-transformer/



The Decoder Side

- Relies on most of the concepts on the encoder side
- See animation on https://jalammar.github.io/illustrated-transformer/



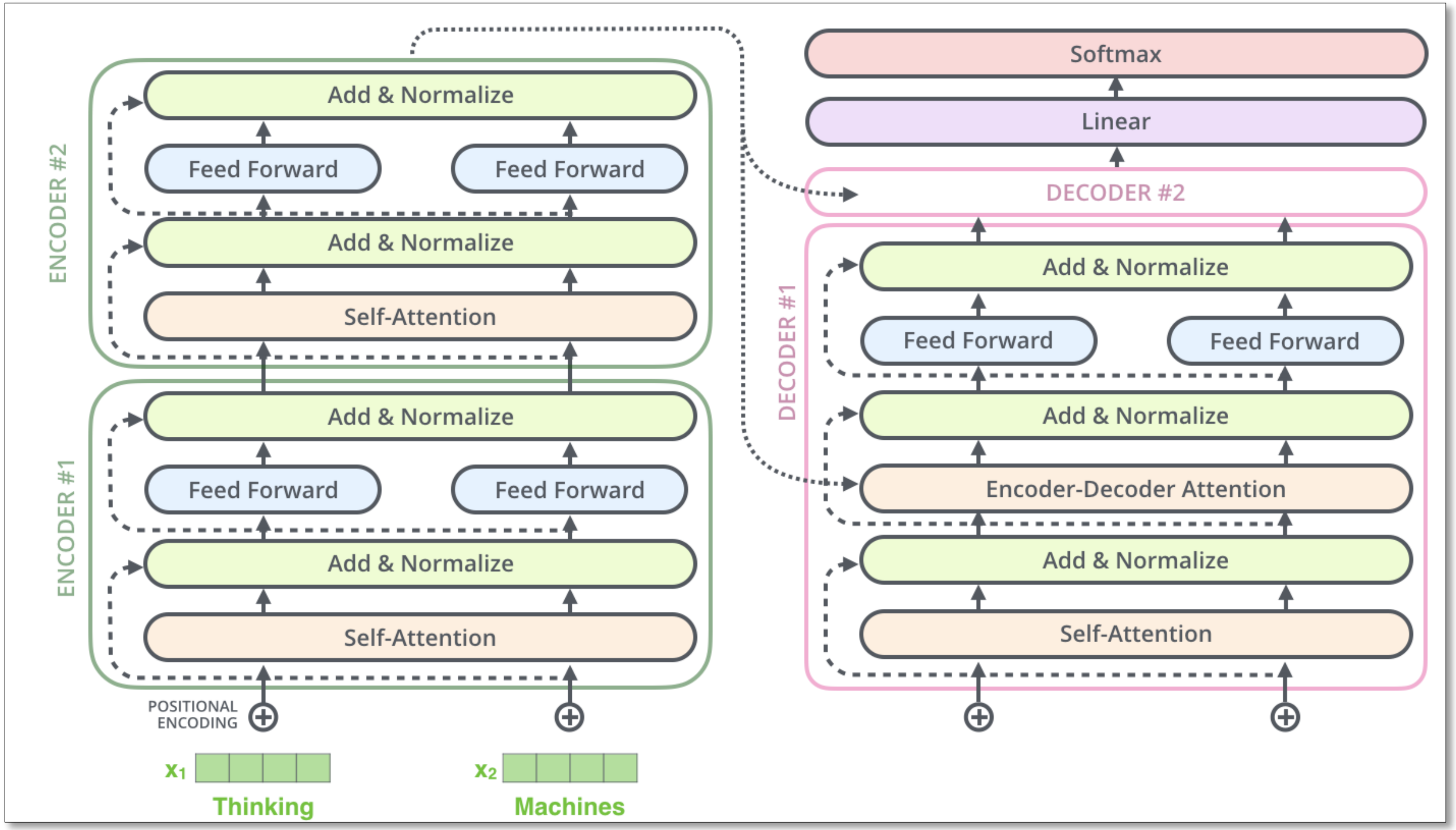
Stack for Decoder only and Stack for Encoder only

- The **RNN** and **LSTM** neural models were designed to process language and perform tasks like classification, summarization, translation, and sentiment detection
  - RNN: Recurrent Neural Network
  - LSTM: Long Short Term Memory
- In both models, layers get the next input word and have access to some previous words, allowing it to use the word's left context
- They used word embeddings where each word was encoded as a vector of 100-300 real numbers representing its meaning

Stack for Decoder only and **Stack for Encoder only**

- Transformers extend this to allow the network to process a word input knowing the words in both its left and right context
- This provides a more powerful context model
- Transformers add additional features, like **attention**, which identifies the important words in this context
- And break the problem into two parts:
  - An encoder (e.g., Bert)
  - A decoder (e.g., GPT)

Transformer model



Encoder (e.g., BERT)

Decoder (e.g., GPT)

Transformers, GPT-2, and BERT

1. A transformer uses an **encoder stack** to model input, and uses **decoder stack** to model output (using input information from encoder side)
2. If we do not have input, we just want to model the “next word”, we can get rid of the encoder side of a transformer and output “next word” one by one. This gives us **GPT**
3. If we are only interested in training a language model for the input for some other tasks, then we do not need the decoder of the transformer, that gives us **BERT**

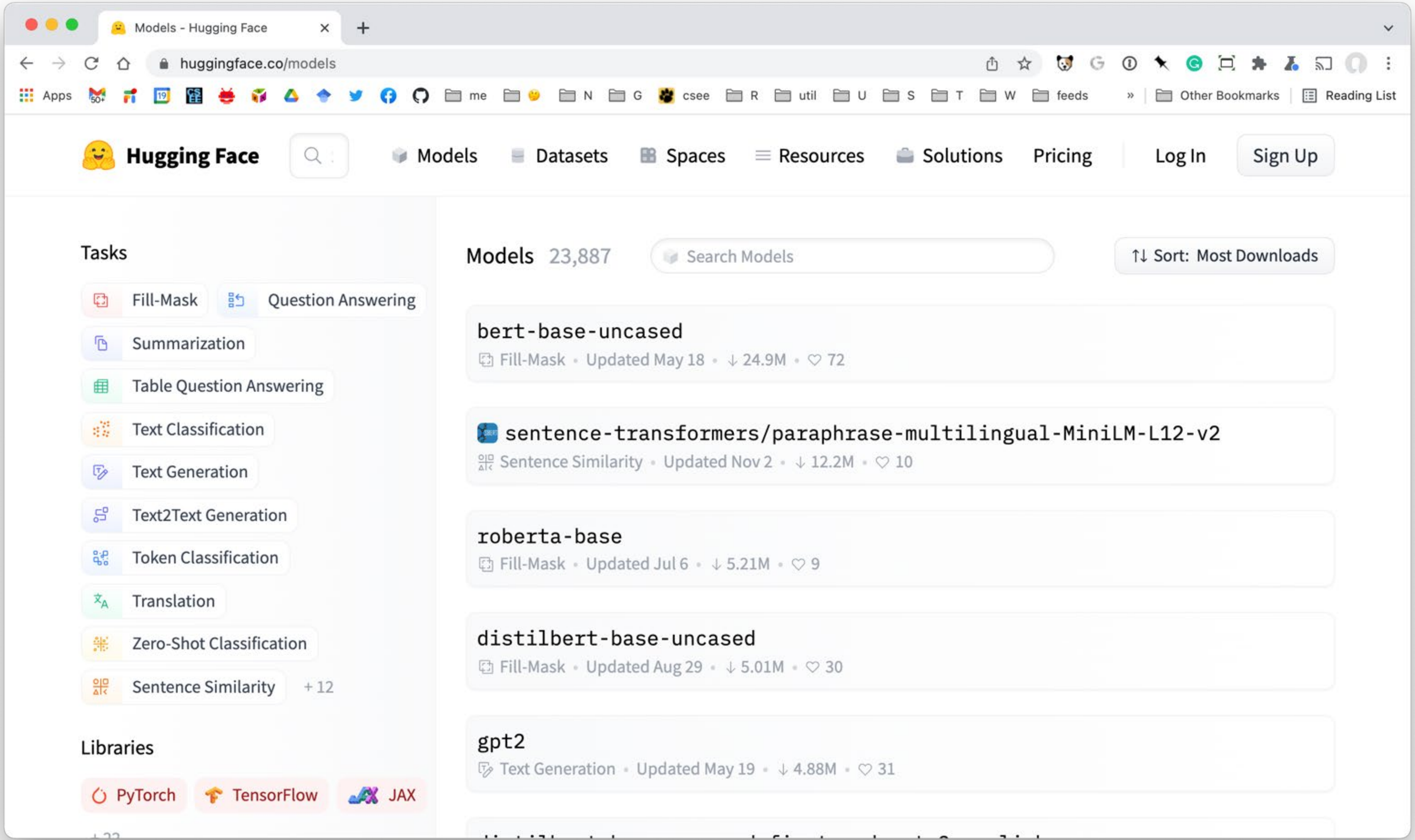
Training a Transformer

- Transformers typically use semi-supervised learning with
  - Unsupervised pretraining over a very large dataset of general text
  - Followed by supervised **fine-tuning** over a focused data set of inputs and outputs for a particular task
- Tasks for pretraining and fine-tuning commonly include:
  - language modeling
  - next-sentence prediction (aka completion)
  - question answering
  - reading comprehension
  - sentiment analysis
  - paraphrasing

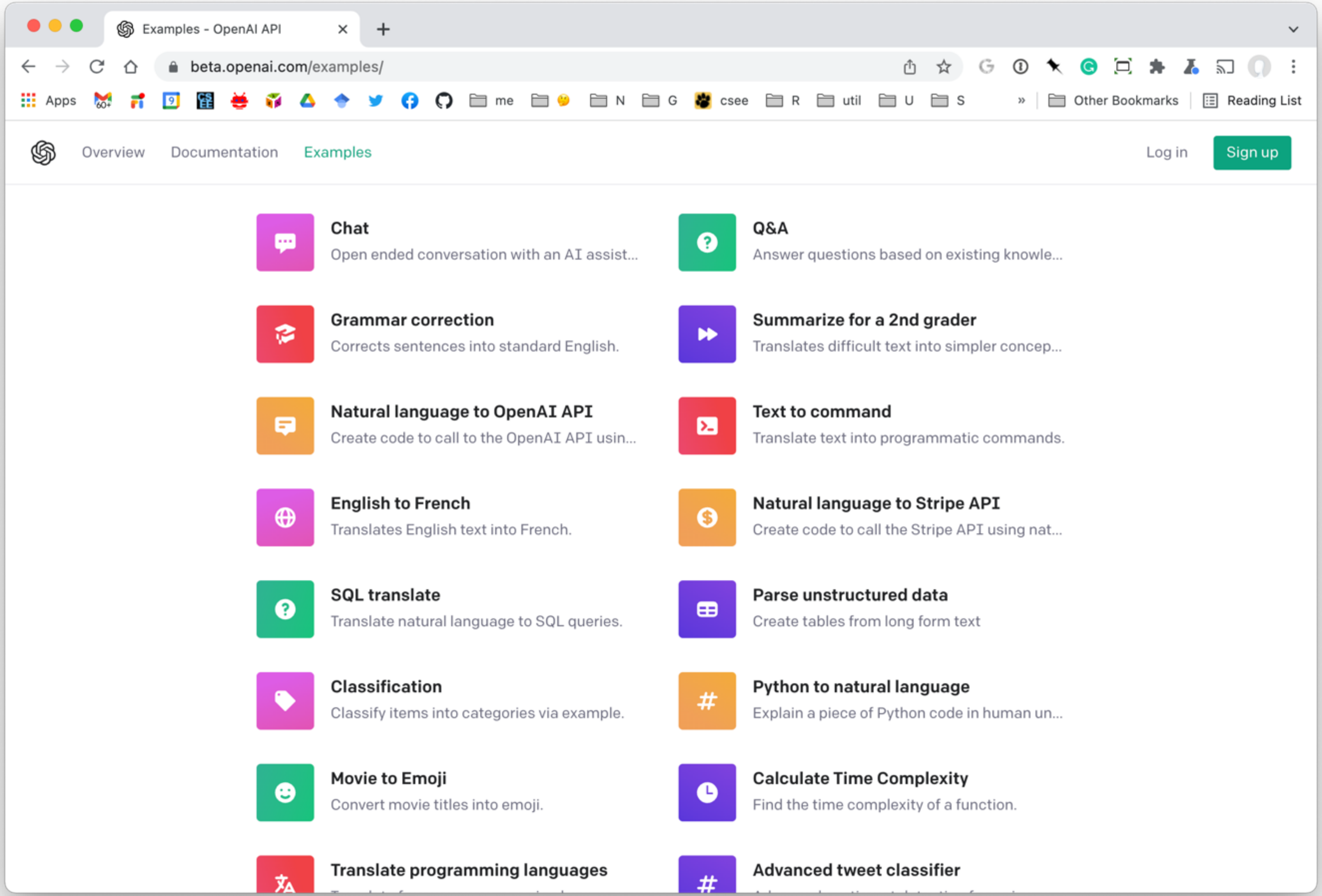
Pretrained models

- Since training a model requires huge datasets of text and significant computation, researchers often use common pretrained models
- Examples (circa December 2021) include
  - Google's BERT model
  - Huggingface's various Transformer models
  - OpenAI's and GPT-3 models

Huggingface Models



OpenAI Application Examples



Text Representations

- Co-occurrence statistics
  - Brown Clusters
  - Count vectors, TF-IDF vectors, co-occurrence matrix decomposition
- Predictive
  - word2vec, GloVe, CBOW, Skip-Gram, etc
- Contextualized language models
  - Representation of word *changes* based on context
  - CoVE, ELMo, GPT, BERT, etc

Who is BERT?

BERT is a 12 (or 24) layer Transformer language model trained on two pretraining tasks, masked language modeling (fill-in-the-blank) and next sentence prediction (binary classification), and on English Wikipedia and BooksCorpus.

About BERT and friends

Why this size and architecture?

-base, -large, -small, -xl, etc.

How much "language"?

linguistic probing tasks, attention, few-shot evaluation

BERT is a 12 (or 24) layer Transformer language model trained on two pretraining tasks, masked language modeling (fill-in-the-blank) and next sentence prediction (binary classification), and on English Wikipedia and BooksCorpus.

Why these tasks?

XLNet, ELECTRA, SpanBERT, LXMERT, etc.

Other languages?

mBERT, XLM, XLM-R, mT5, RuBERT, etc

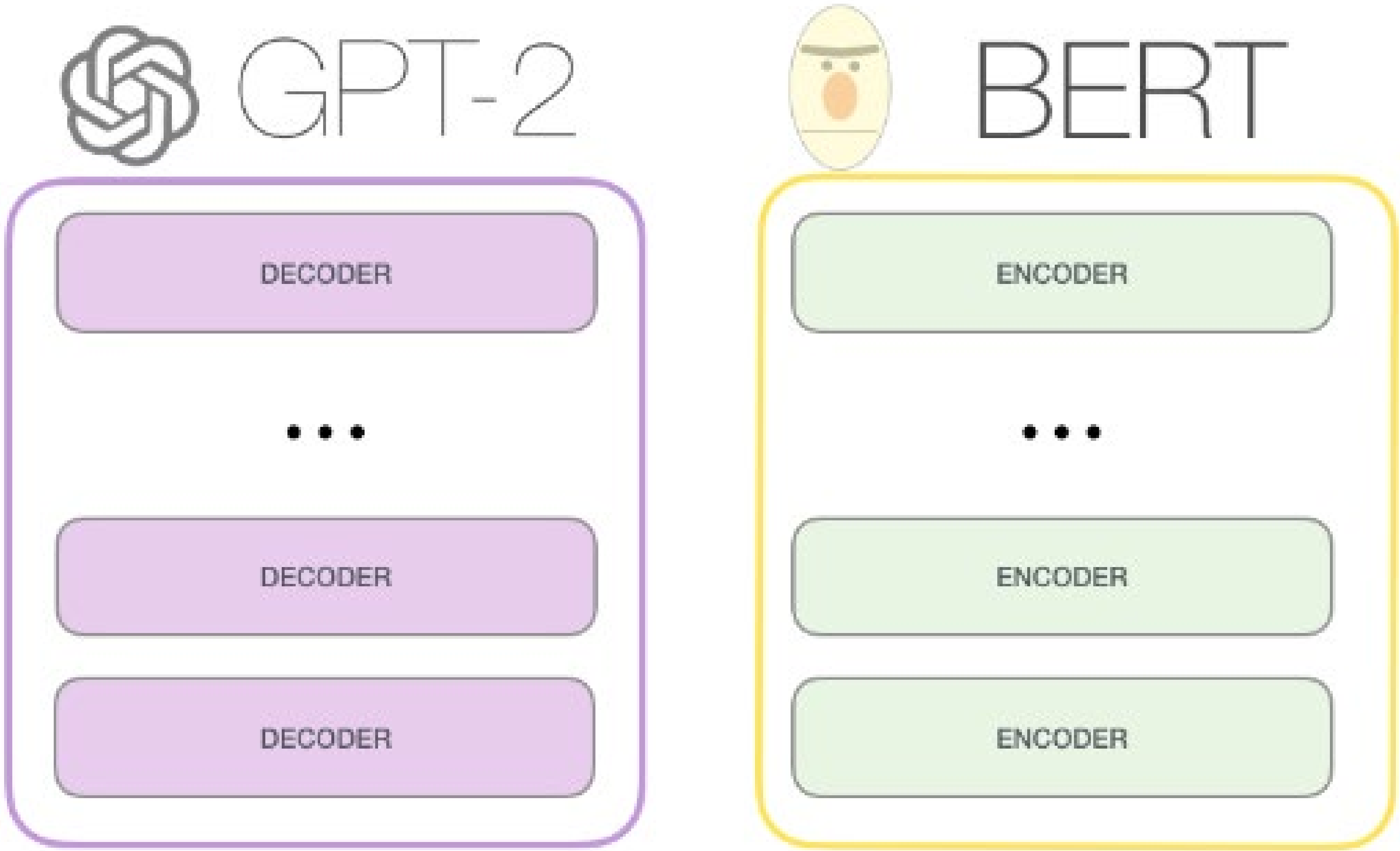
What's special about this data?

BioBERT, Covid-Twitter-BERT, etc

Using these *BERTs

- Pretrain → finetune
  - *Pretrain* encoders on pretraining tasks (high-resource/data, possibly unsupervised)
  - *Finetune* encoders on target task (low-resource, expensive annotation)
- Primary method of evaluation: Natural Language “Understanding” (NLU)
  - Question Answering and Reading Comprehension
  - Commonsense
  - Textual Entailments

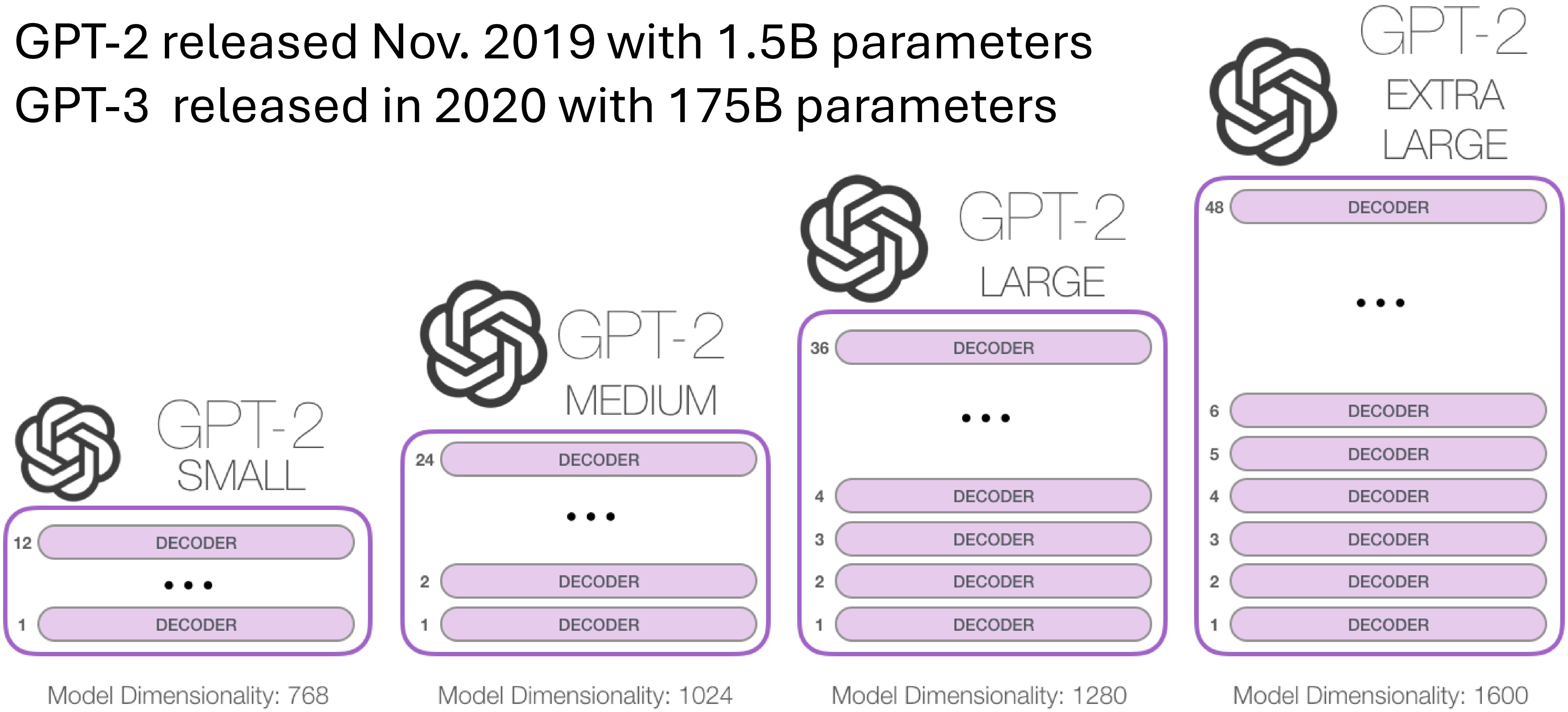
GPT-2 - BERT


GPT-2
DECODER
DECODER
DECODER
BERT
ENCODER
ENCODER
ENCODER

GPT released June 2018

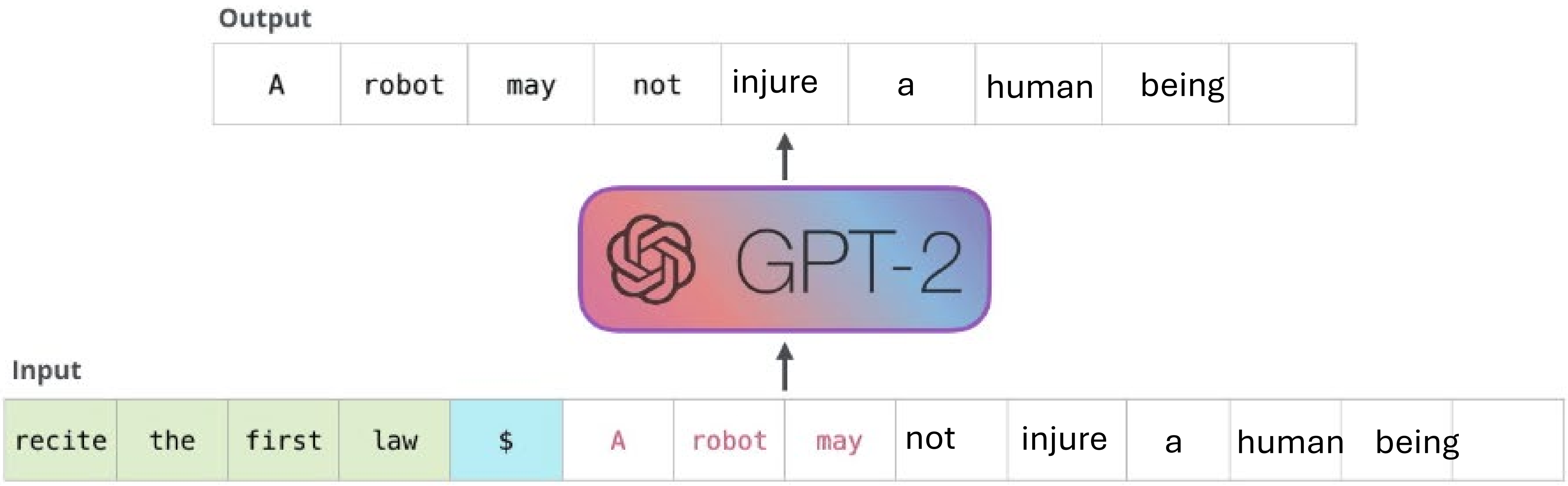
GPT-2 released Nov. 2019 with 1.5B parameters

GPT-3 released in 2020 with 175B parameters



117M parameters 345M 762M 1542M

GPT-2 in action



Byte Pair Encoding (BPE)

Word embedding sometimes is too high level, pure character embedding too low level. For example, if we have learned

old older oldest

We might also wish the computer to infer

smart smarter smartest

But at the whole word level, this might not be so direct. Thus the idea is to break the words up into pieces like er, est, and embed frequent fragments of words.

GPT adapts this BPE scheme.

Byte Pair Encoding (BPE)

GPT uses BPE scheme. The subwords are calculated by:

1. Split word to sequence of characters (add </w> char)
2. Joining the highest frequency pattern.
3. Keep doing step 2, until it hits the pre-defined maximum number of sub-words or iterations.

Example (5, 2, 6, 3 are number of occurrences)

{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t </w>': 6, 'w i d e s t </w>': 3 }

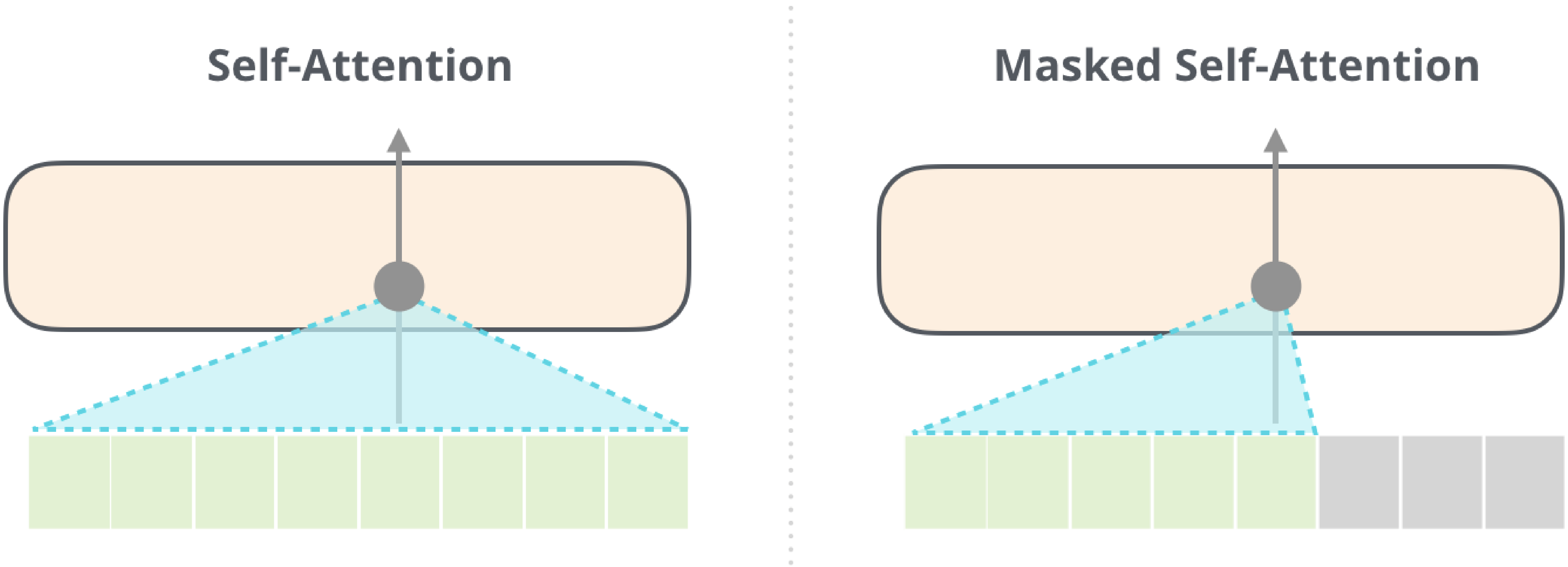
{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w es t </w>': 6, 'w i d es t </w>': 3 }

{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w est </w>': 6, 'w i d est </w>': 3 } (est freq. 9)

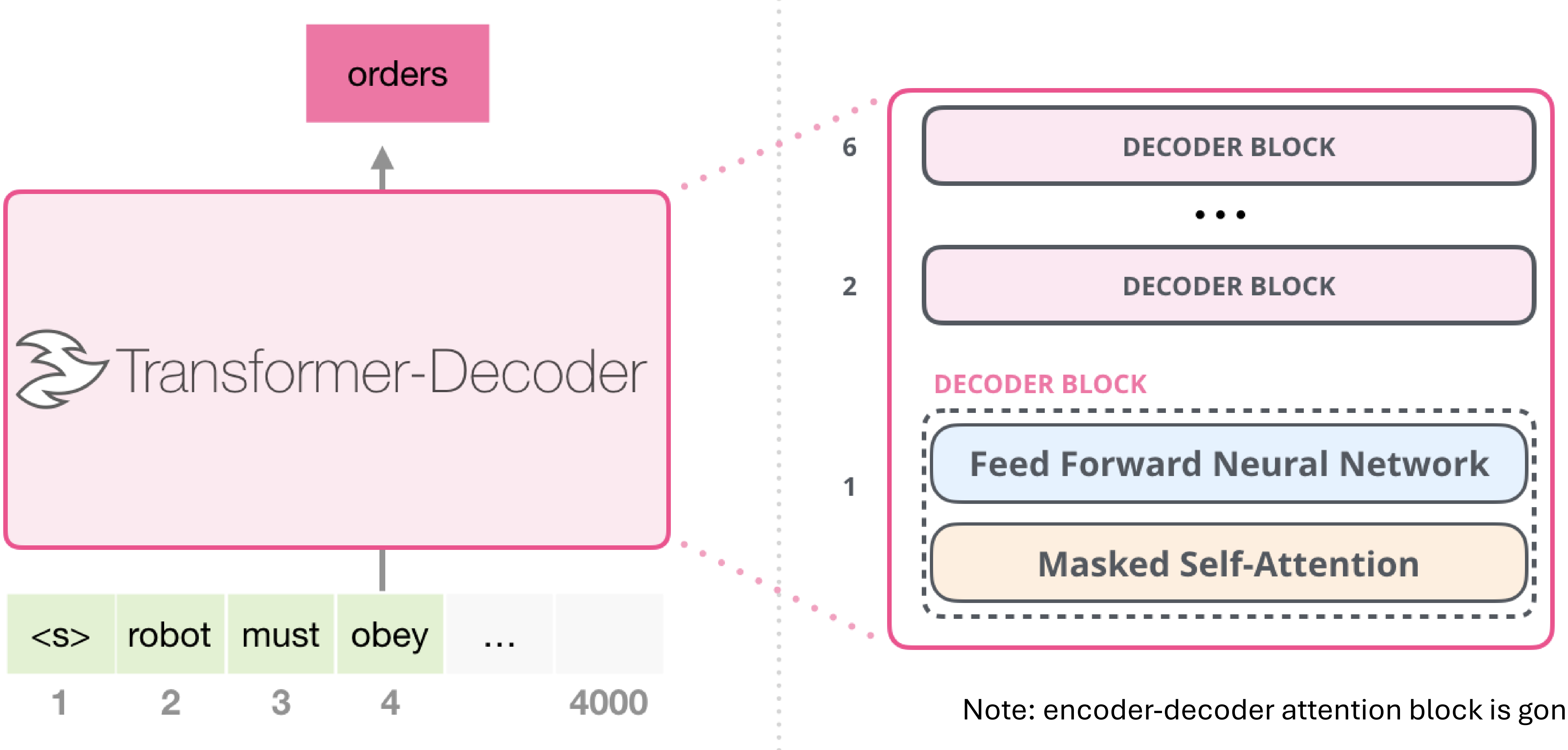
{'lo w </w>': 5, 'lo w e r </w>': 2, 'n e w est</w>': 6, 'w i d est</w>': 3 } (lo freq 7)

.....

Masked Self-Attention (to compute more efficiently)



Masked Self-Attention



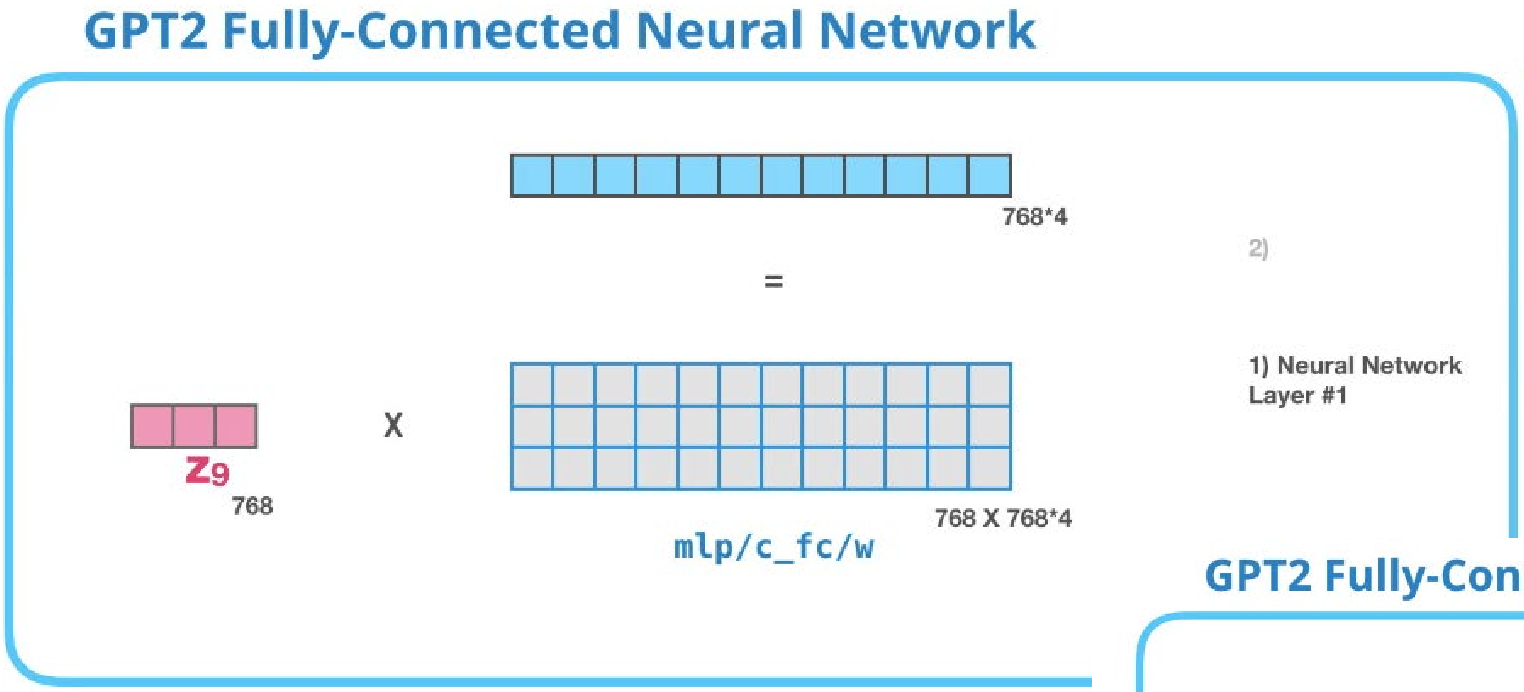
Note: encoder-decoder attention block is gone

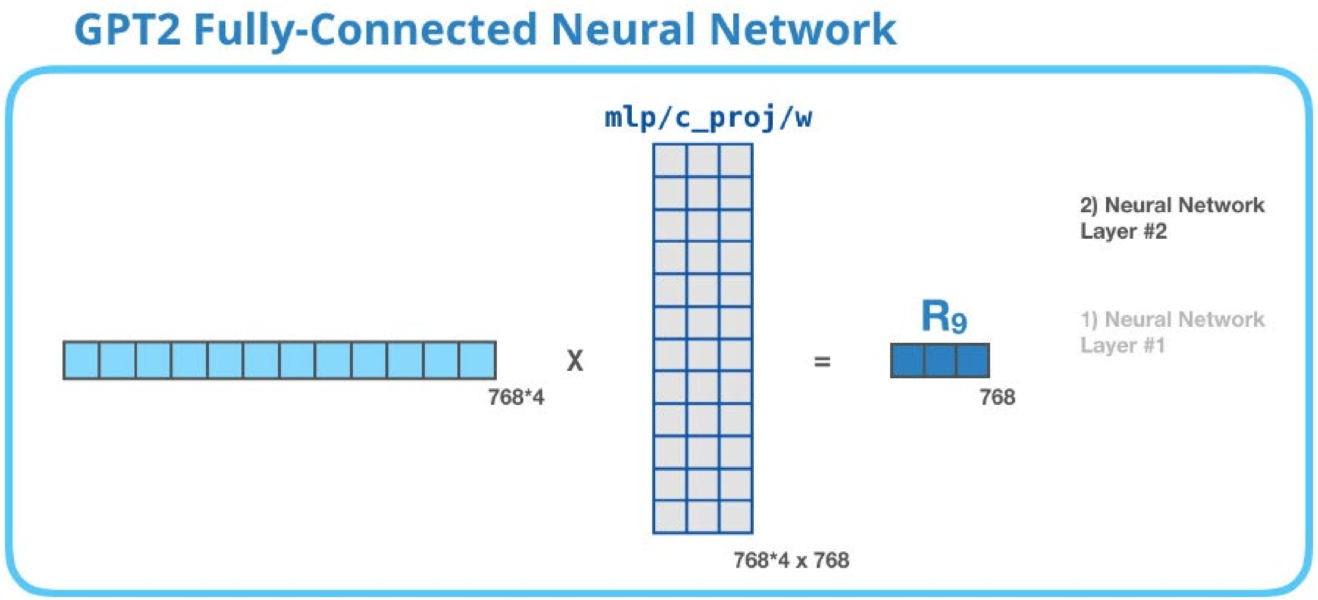
Masked Self-Attention Calculation

Re-use previous computation results: at any step, only need to results of q, k , v related to the new output word, no need to re-compute the others. Additional computation is linear, instead of quadratic.

Note: encoder-decoder attention block is gone

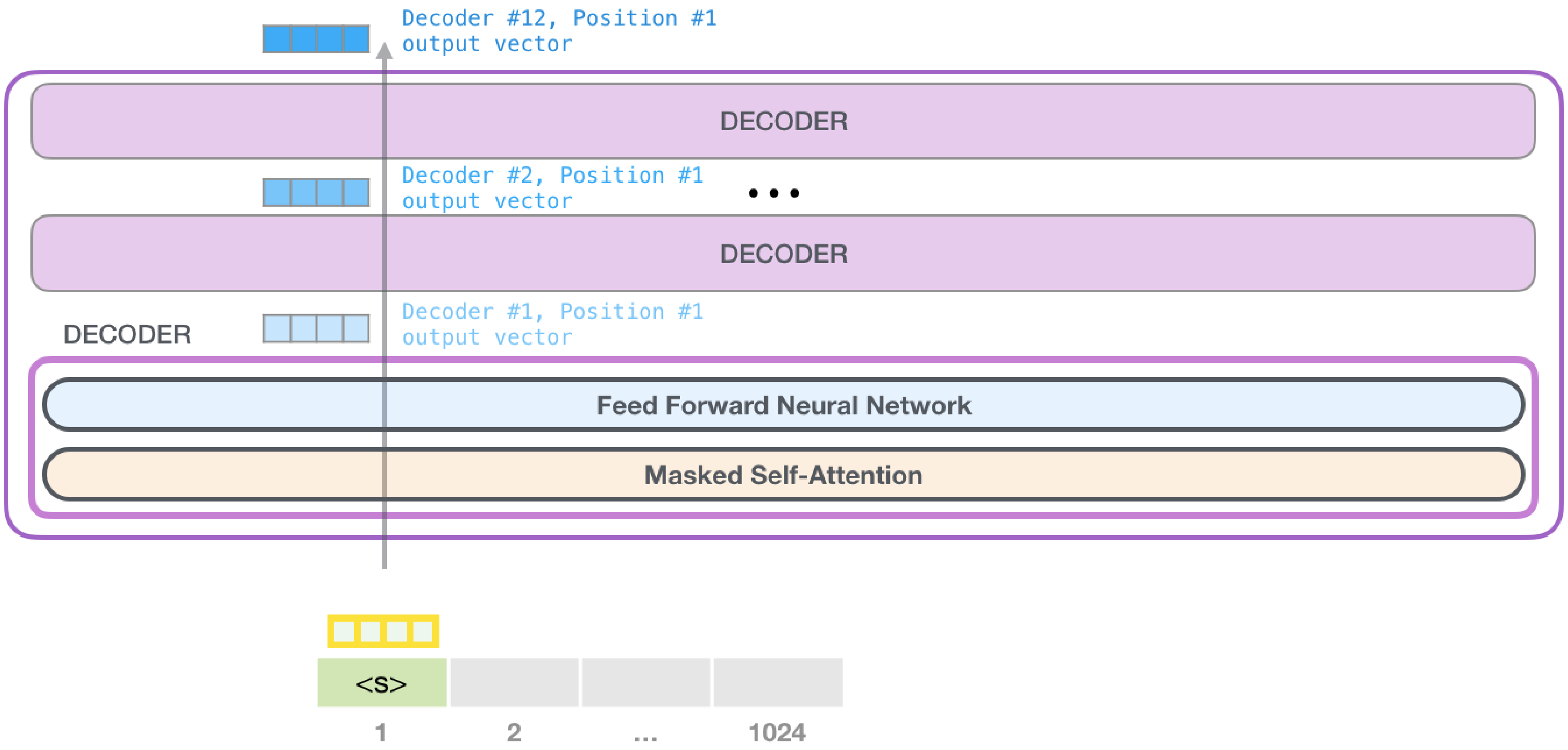
GPT-2 fully connected network has two layers (Example for GPT-2 small)





768 is small model size

GPT-2 has a parameter top-k, so that we sample words from top k (highest probability from SoftMax) words for each output

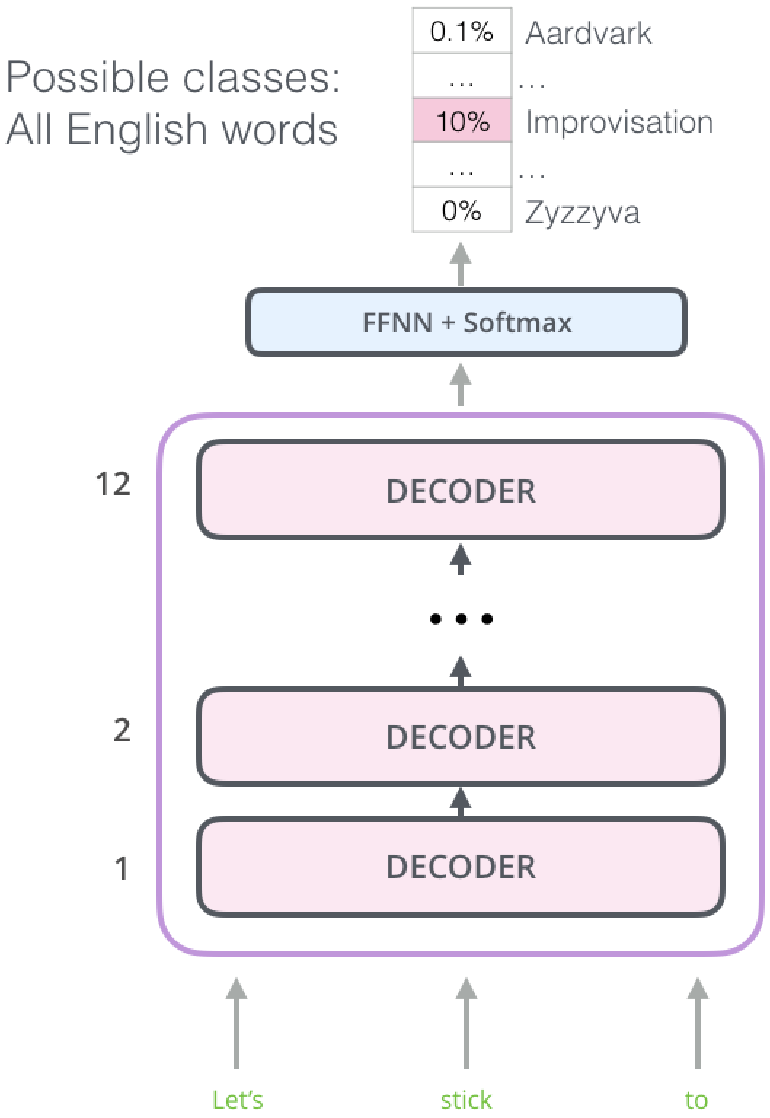


GPT Training

GPT-2 uses unsupervised learning approach to training the language model.

There is no custom training for **GPT-2**, no separation of pre-training and fine-tuning like **BERT**.

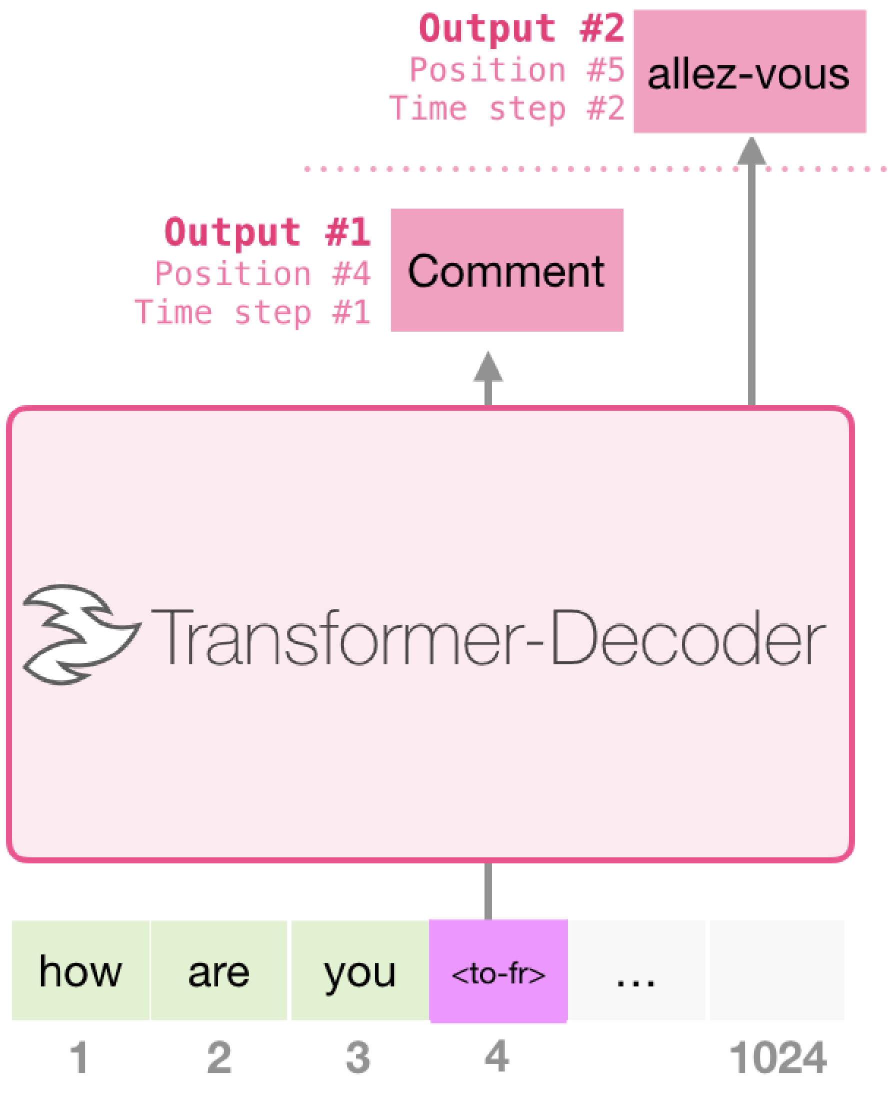
Transformer / GPT prediction



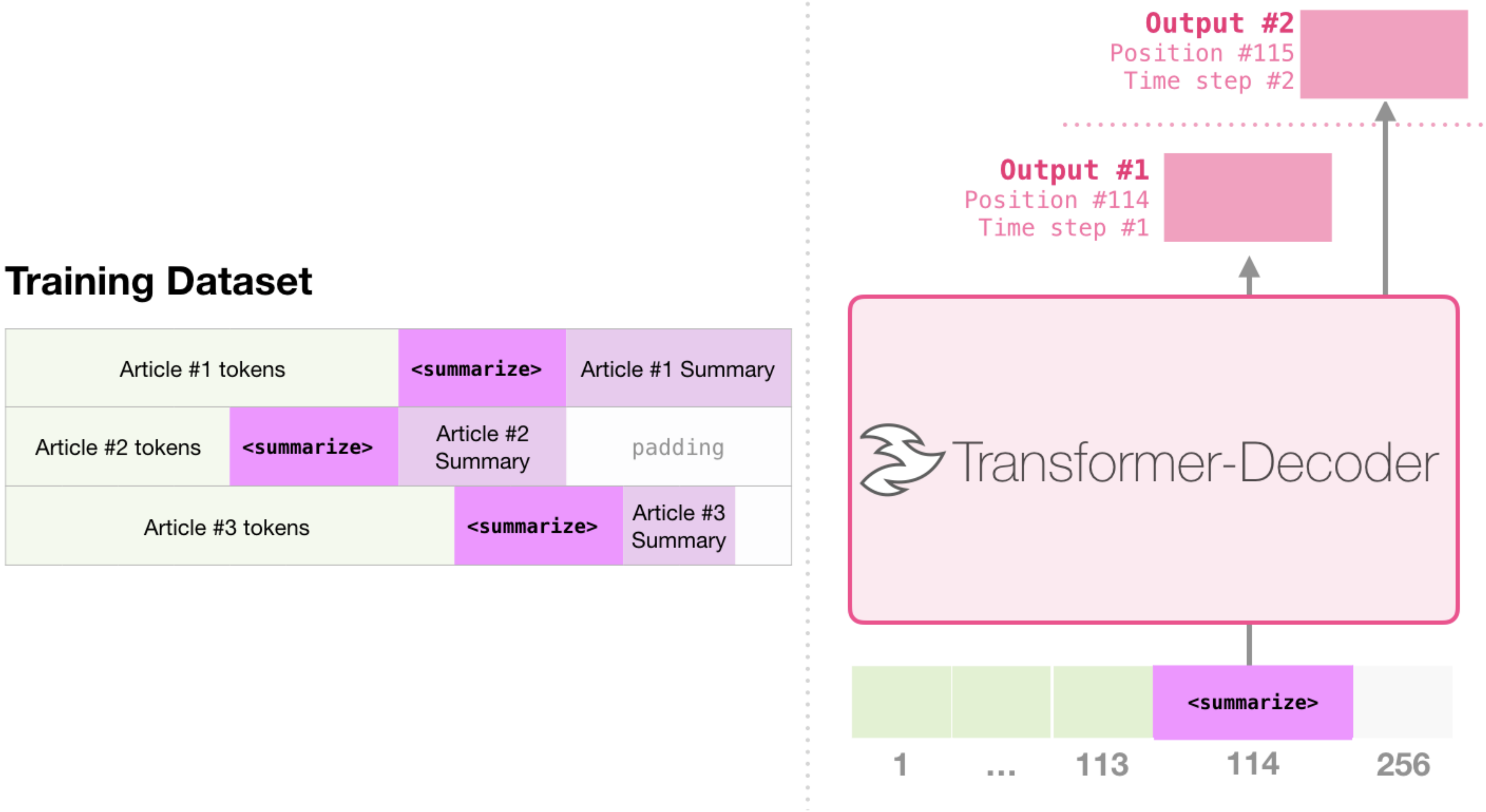
GPT-2 Application: Translation

Training Dataset

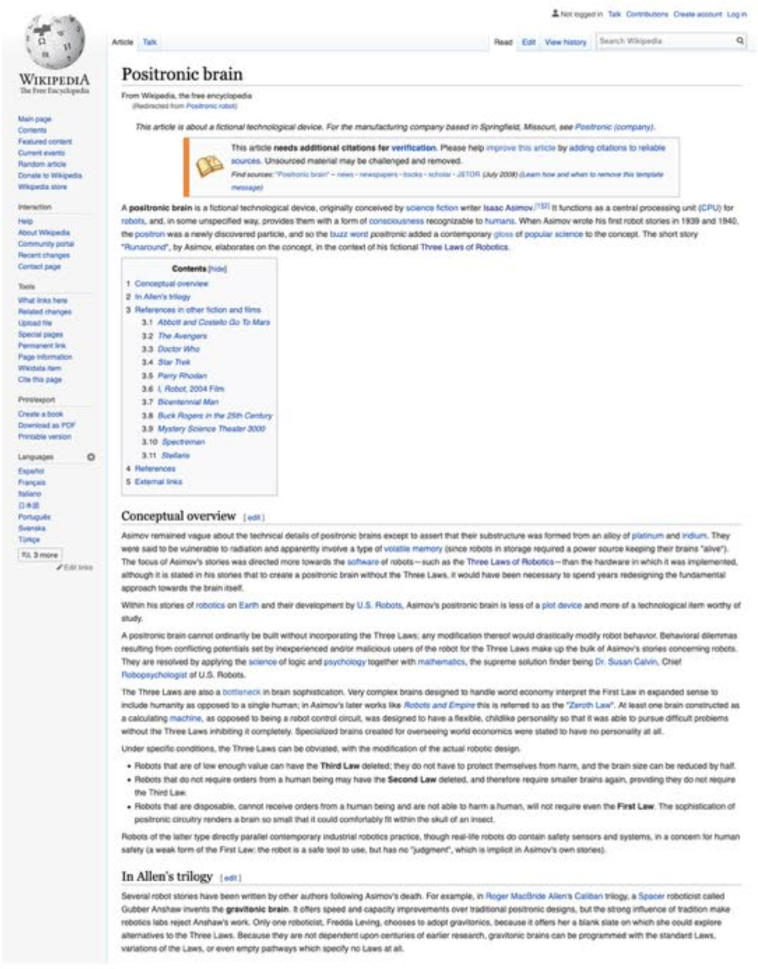
I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu'ils	mangent	de
good	morning	<to-fr>	Bonjour				



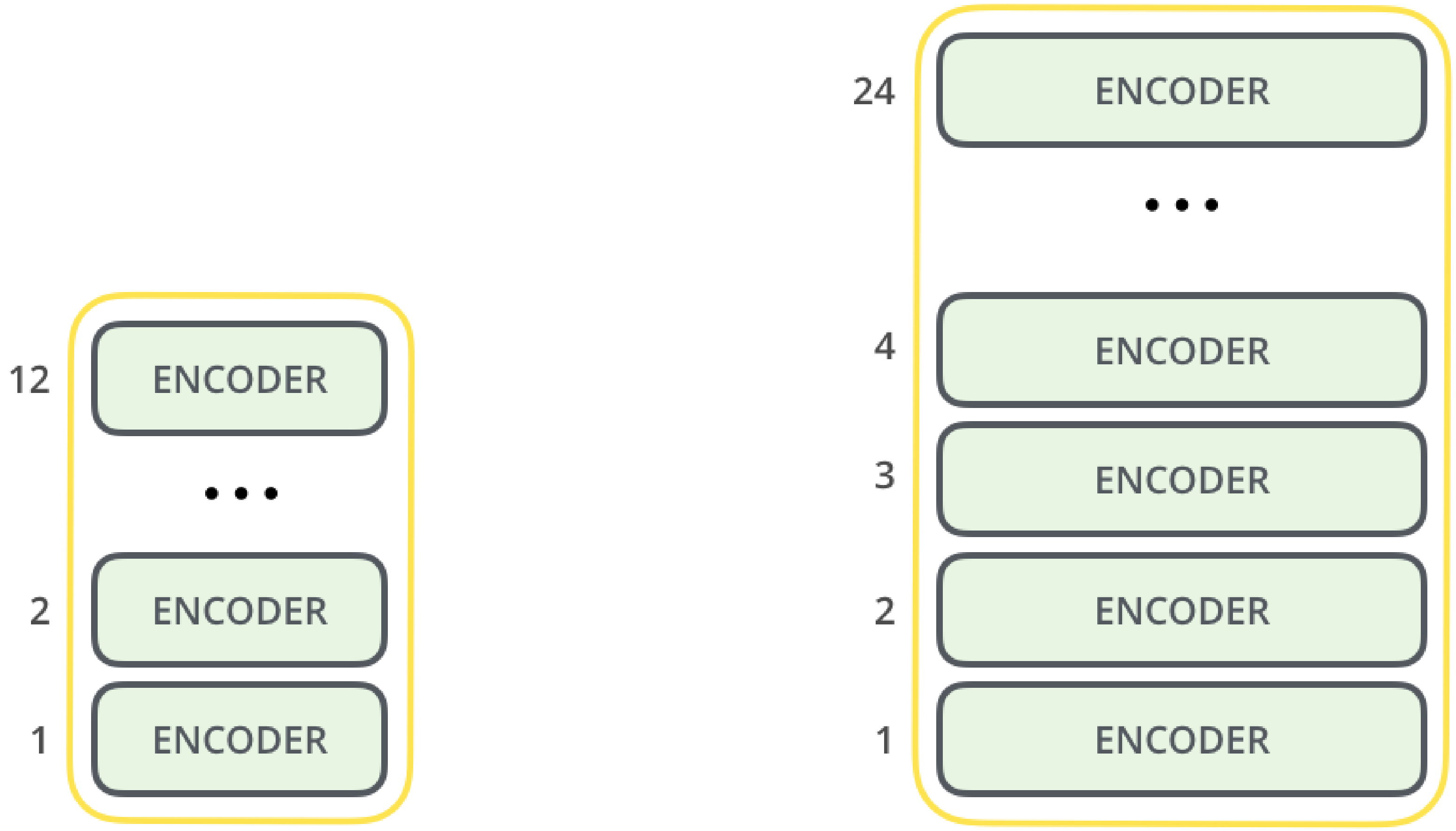
GPT-2 Application: Summarization



Using Wikipedia data



BERT (Bidirectional Encoder Representation from Transformers)

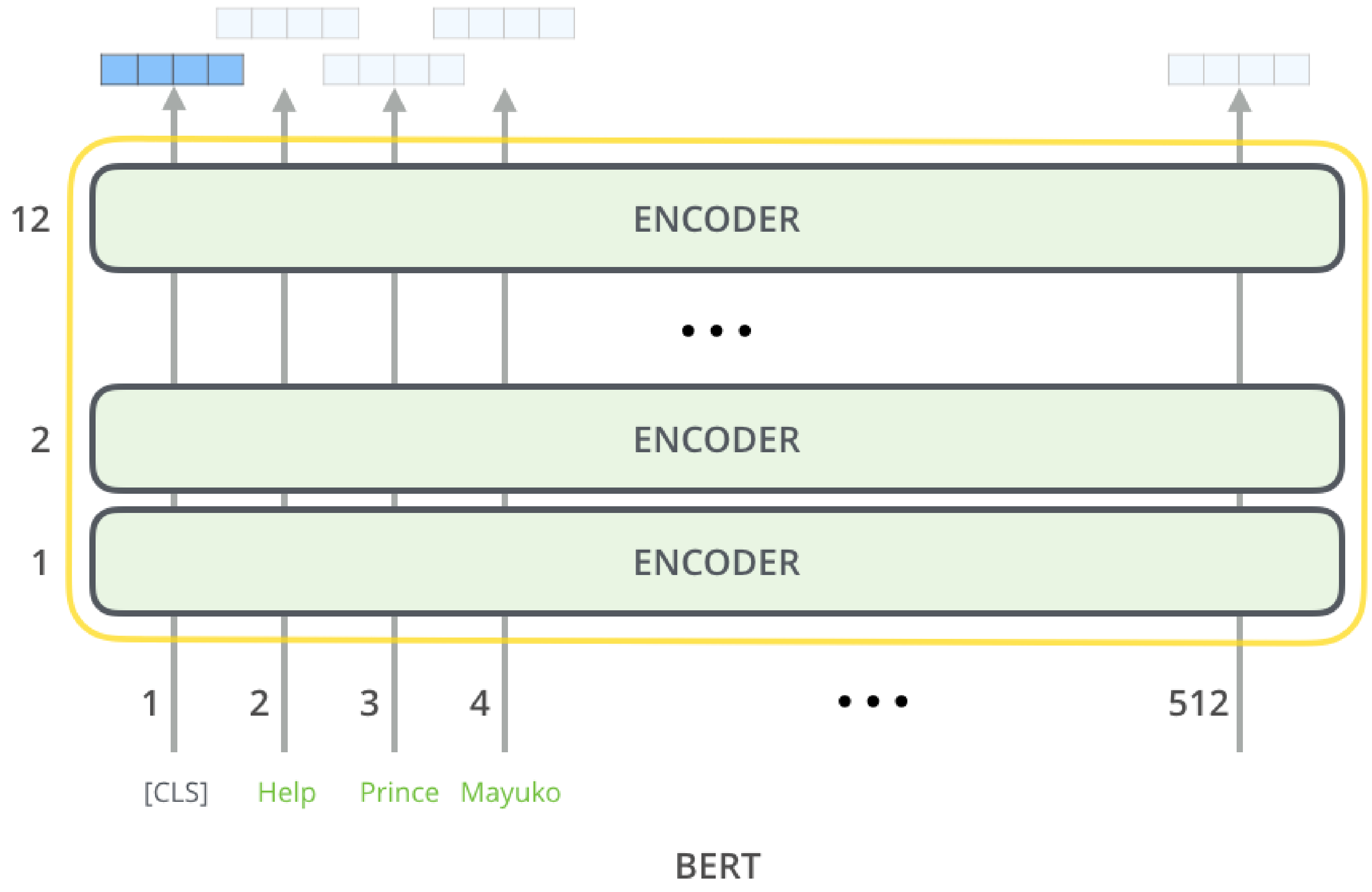


BERT BASE

BERT LARGE

Model input dimension 512

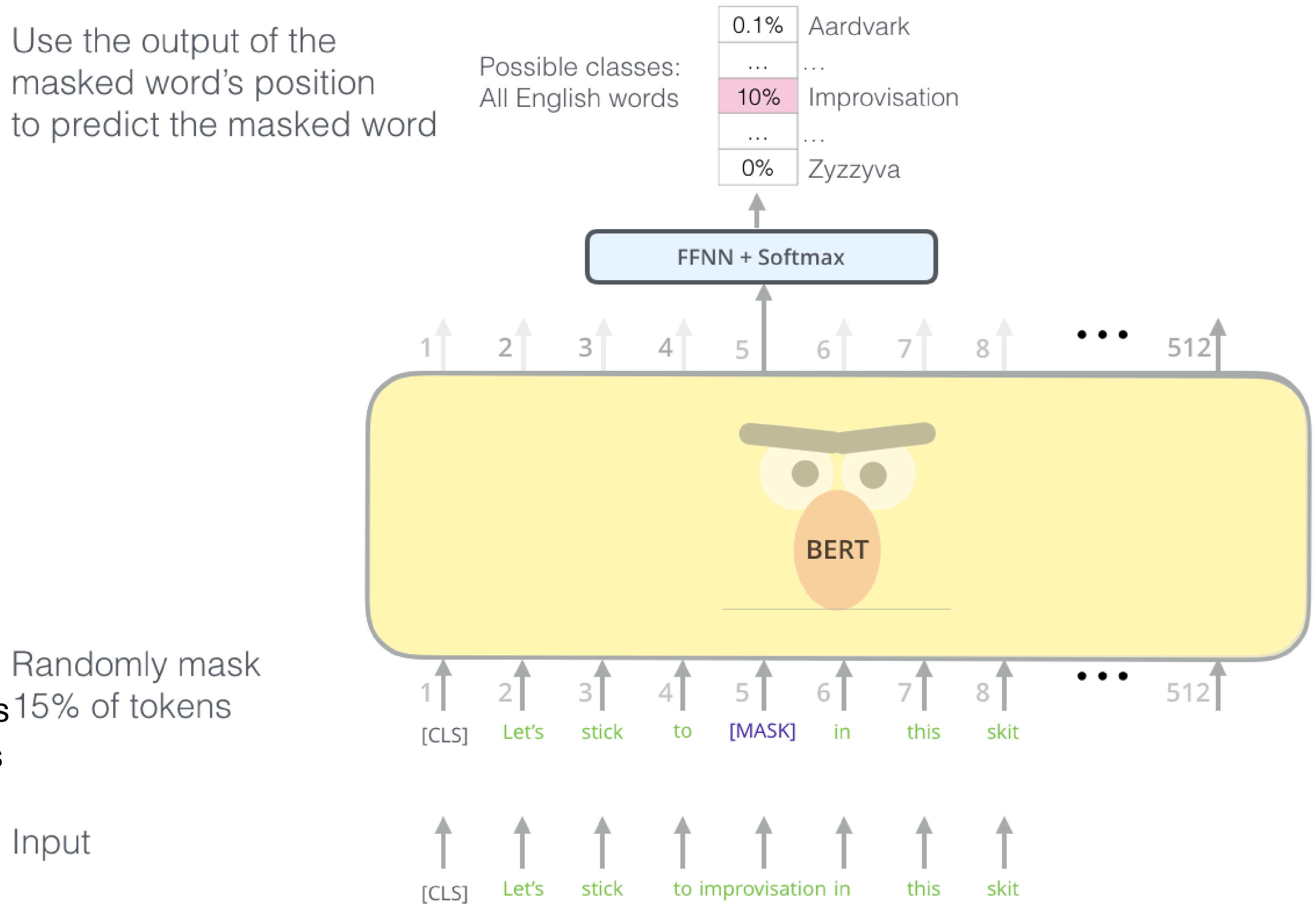
Input and output vector size



BERT pretraining

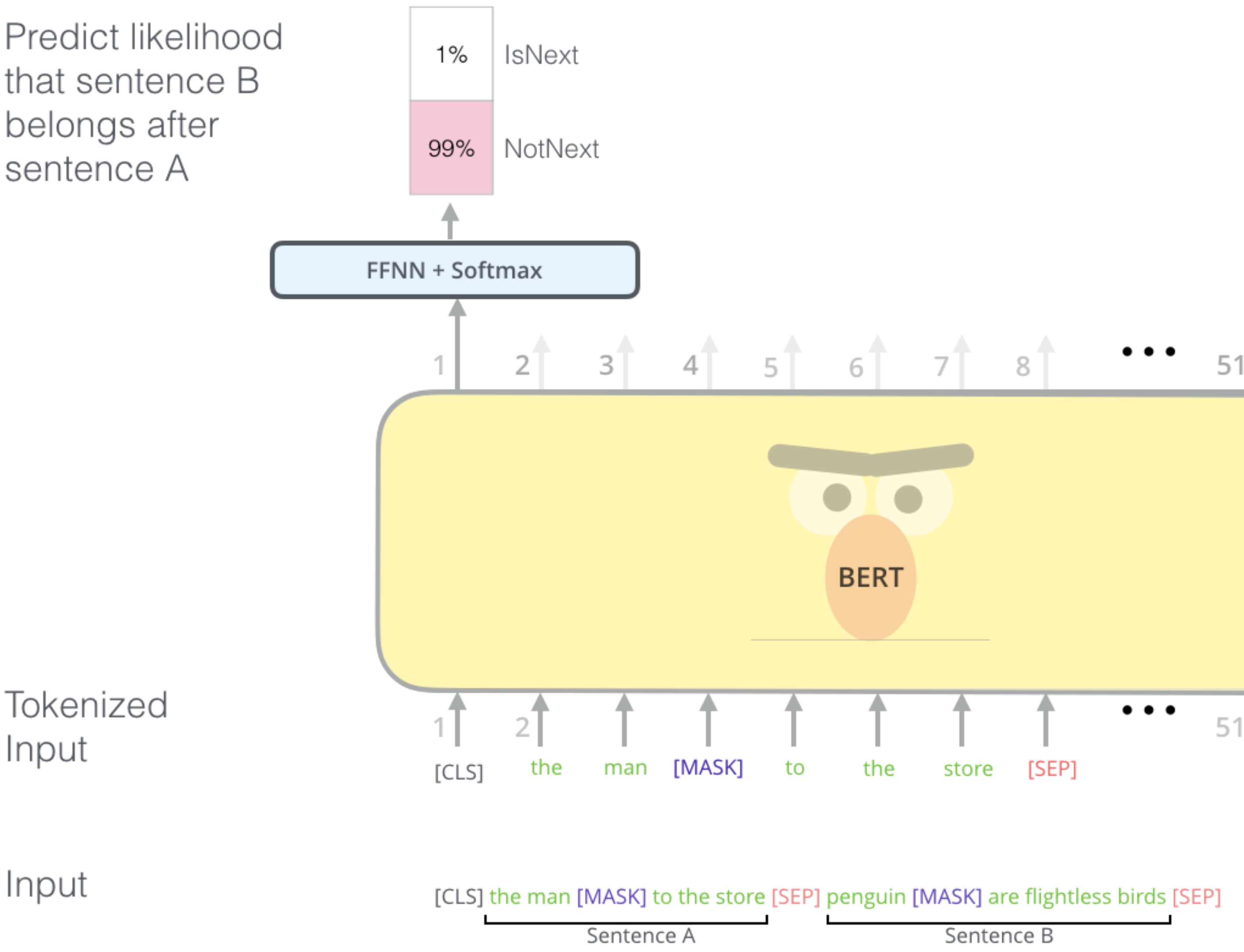
ULM-FiT (2018): Pre-training ideas, transfer learning in NLP.
ELMo: Bidirectional training (LSTM)
Transformer: Although used things from left, but still missing from the right.
GPT: Use Transformer Decoder half.
BERT: Switches from Decoder to Encoder, so that it can use both sides in training and invented corresponding training tasks: masked language model

BERT Pretraining Task 1: masked words

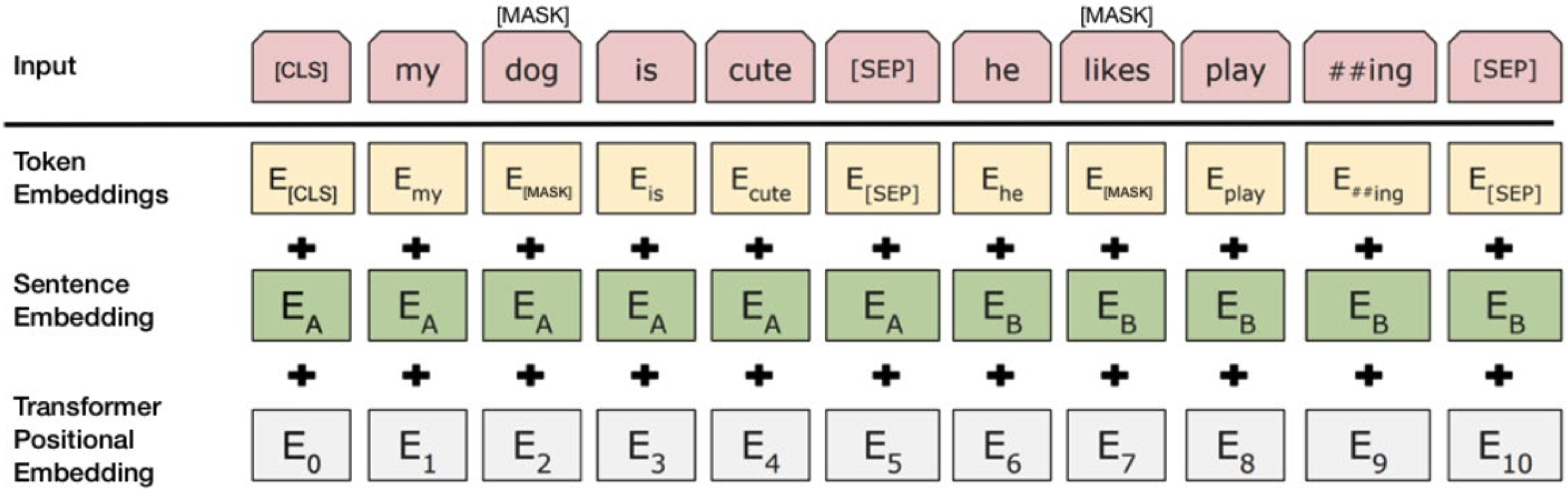


Out of this 15%,
80% are [Mask],
10% random words
10% original words

BERT Pretraining Task 2: two sentences



BERT Pretraining Task 2: two sentences

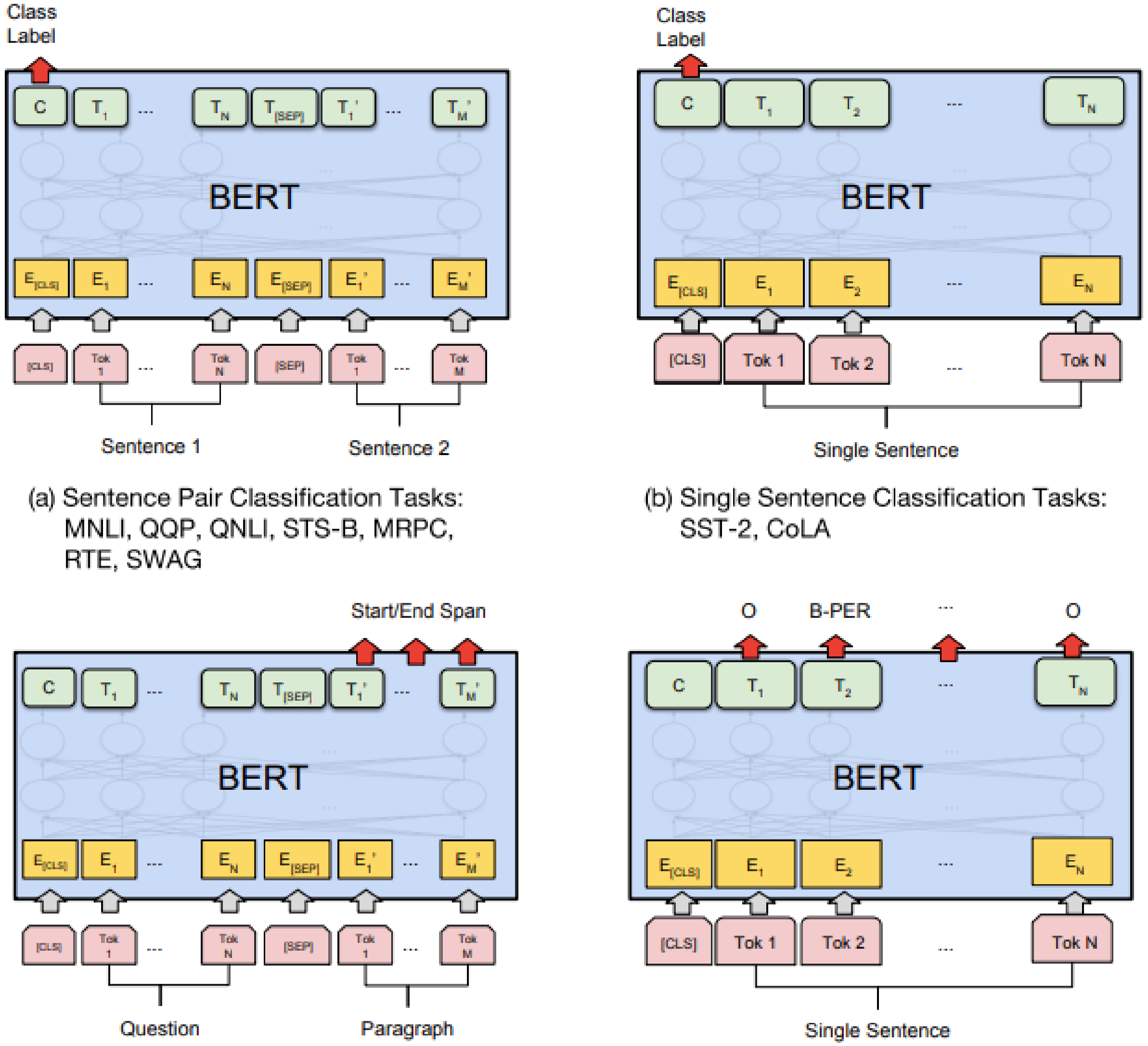


50% true second sentences

50% random second sentences

Fine-tuning BERT for other specific tasks

MNLI
QQP (Quaro Question Pairs)
 Semantic equivalence)
QNLI (NL inference dataset)
STS-B (texture similarity)
MRPC (paraphrase, Microsoft)
RTE (textual entailment)
SWAG (commonsense inference)
SST-2 (sentiment)
CoLA (linguistic acceptability
SQuAD (question and answer)



(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

SST (Stanford sentiment treebank): 215k phrases with fine-grained sentiment labels in the parse trees of 11k sentences.

NLP Tasks: Multi-Genre Natural Lang. Inference

MNLI: 433k pairs of examples, labeled by entailment, neutral or contraction

Met my first girlfriend that way.	FACE-TO-FACE **contradiction** C C N C	I didn't meet my first girlfriend until later.
8 million in relief in the form of emergency housing.	GOVERNMENT **neutral** N N N N	The 8 million dollars for emergency housing was still not enough to solve the problem.
Now, as children tend their gardens, they have a new appreciation of their relationship to the land, their cultural heritage, and their community.	LETTERS **neutral** N N N N	All of the children love working in their gardens.
At 8:34, the Boston Center controller received a third transmission from American 11	9/11 **entailment** E E E E	The Boston Center controller got a third transmission from American 11.
I am a lacto-vegetarian.	SLATE **neutral** N N E N	I enjoy eating cheese too much to abstain from dairy.
someone else noticed it and i said well i guess that's true and it was somewhat melodious in other words it wasn't just you know it was really funny	TELEPHONE **contradiction** C C C C	No one noticed and it wasn't funny at all.

Table 1: Randomly chosen examples from the development set of our new corpus, shown with their genre labels, their selected gold labels, and the validation labels (abbreviated E, N, C) assigned by individual annotators.

NLP Tasks (SQuAD -- Stanford Question Answering Dataset):

Sample: Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50.

Which NFL team represented the AFC at Super Bowl 50?

Ground Truth Answers: Denver Broncos

Which NFL team represented the NFC at Super Bowl 50?

Ground Truth Answers: Carolina Panthers

Add indices for sentences and paragraphs

SegaTron/SegaBERT

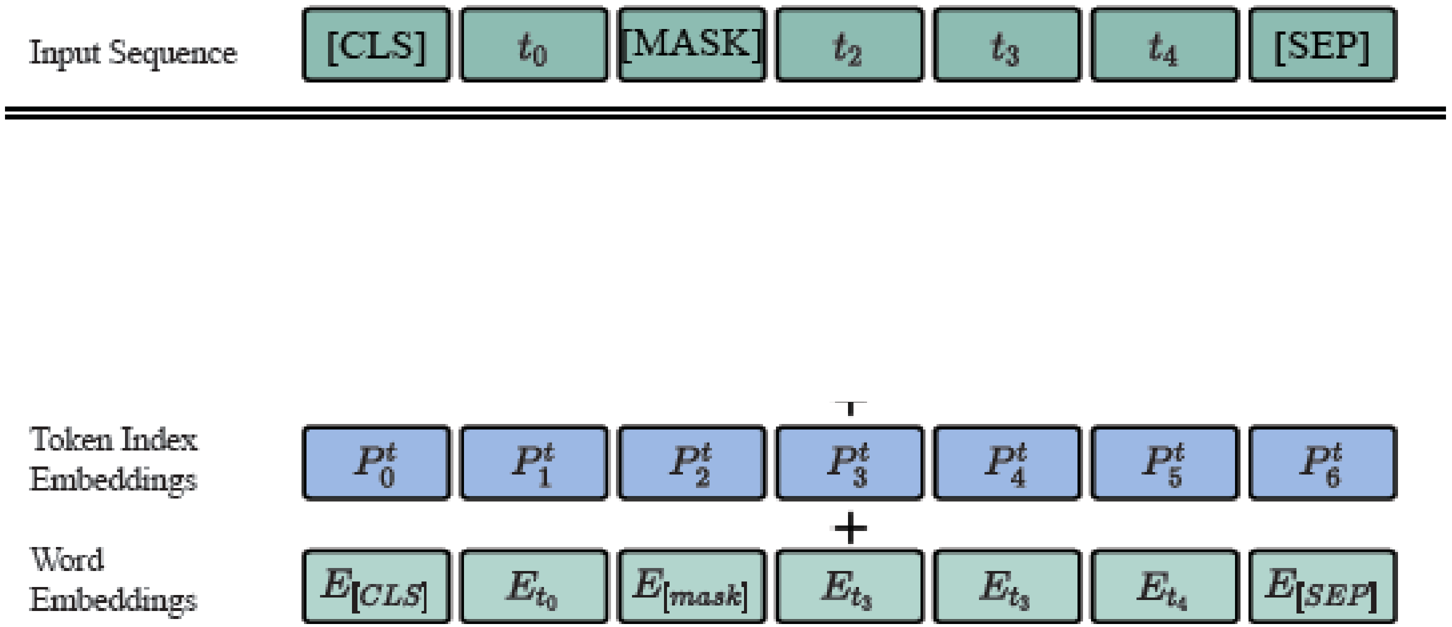
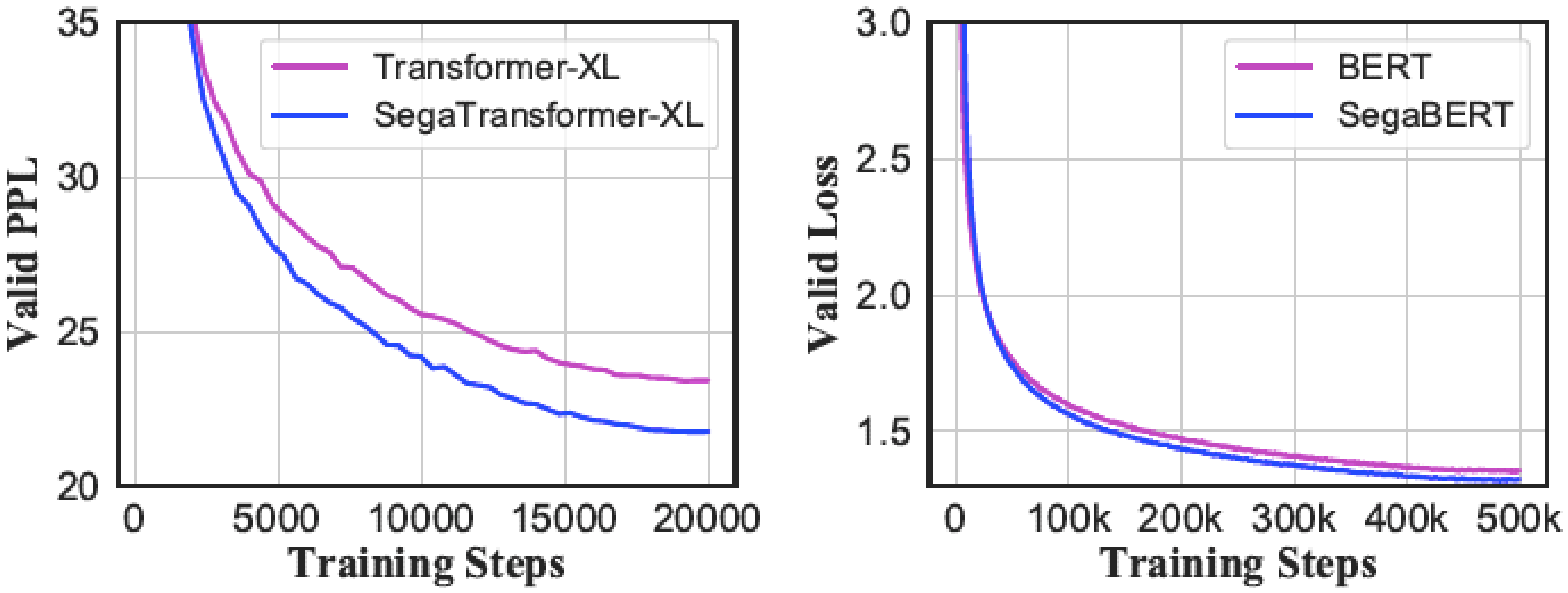


Figure 1: Input Representation of SegaBERT

H. Bai, S. Peng, J. Lin, L. Tan, K. Xiong, W. Gao, M. Li: SgaTron: Segment-aware transformer for language modeling and understanding. AAAI'2021

Conversion speed much faster:



(a) Language modeling perplexities

(b) Pre-training losses

Figure 2: Valid perplexities and losses during the training processes of language modeling and pre-training.

Testing on GLUE dataset

Task(Metrics)	BASE model(wikipedia 500K steps)				LARGE model(wikibooks 1000K steps)			
	dev		test		dev		test	
	BERT	SegaBERT	BERT	SegaBERT	BERT	SegaBERT	BERT	SegaBERT
CoLA (Matthew Corr.)	**55.0**	54.7	43.5	**50.7**	60.6	**65.3**	60.5	**62.6**
SST-2 (Acc.)	91.3	**92.1**	91.2	**91.5**	93.2	**94.7**	**94.9**	94.8
MRPC (F1)	**92.6**	92.4	88.9	**89.3**	-	92.3	89.3	**89.7**
STS-B (Spearman Corr.)	88.9	**89.0**	83.9	**84.6**	-	90.3	86.5	**88.6**
QQP (F1)	86.5	**87.0**	70.8	**71.4**	-	89.1	72.1	**72.5**
MNLI-m (Acc.)	83.2	**83.8**	82.9	**83.5**	86.6	**87.6**	86.7	**87.9**
MNLI-mm (Acc.)	83.4	**84.1**	82.8	**83.2**	-	87.5	85.9	**87.7**
QNLI (Acc.)	90.4	**91.5**	90.1	**90.8**	92.3	**93.6**	92.7	**94.0**
RTE (Acc.)	68.3	**71.8**	65.4	**68.1**	70.4	**78.3**	70.1	**71.6**
Average	82.2	**82.9**	77.7	**79.2**	-	86.5	82.1	**83.3**

Table 2: The results on GLUE benchmark. All base models are pre-trained by this work. Every result of the dev set is the average score of 4 times finetuning with different random seeds. Scores of BERT large dev are from (Sun et al., 2019) and scores of BERT large test are from (Devlin et al., 2018).

H. Bai, S. Peng, J. Lin, L. Tan, K. Xiong, W. Gao, M. Li: SgaTron: Segment-aware transformer for language modeling and understanding. AAAI’2021

Reading comprehension – SQUAD tasks

System	Dev	
	EM	F1
BERT base (Single)	80.8	88.5
BERT large (Single	84.1	90.9
BERT large (Single+DA)	84.2	91.1
KT-NET	85.2	91.7
StructBERT Large (Single)	85.2	92.0
SegaBERT base (Single)	83.2	90.2
SegaBERT large (Single)	85.3	92.4

Table 3: Evaluation results of SQUAD v1.1.

System	Dev	
	EM	F1
BERT base	72.3	75.6
BERT base (ours)	75.4	78.2
SegaBERT base	76.3	79.2
BERT large	78.7	81.9
BERT large wwm	80.6	83.4
SegaBERT large	81.8	85.2

Table 4: Evaluation results of SQUAD v2.0.

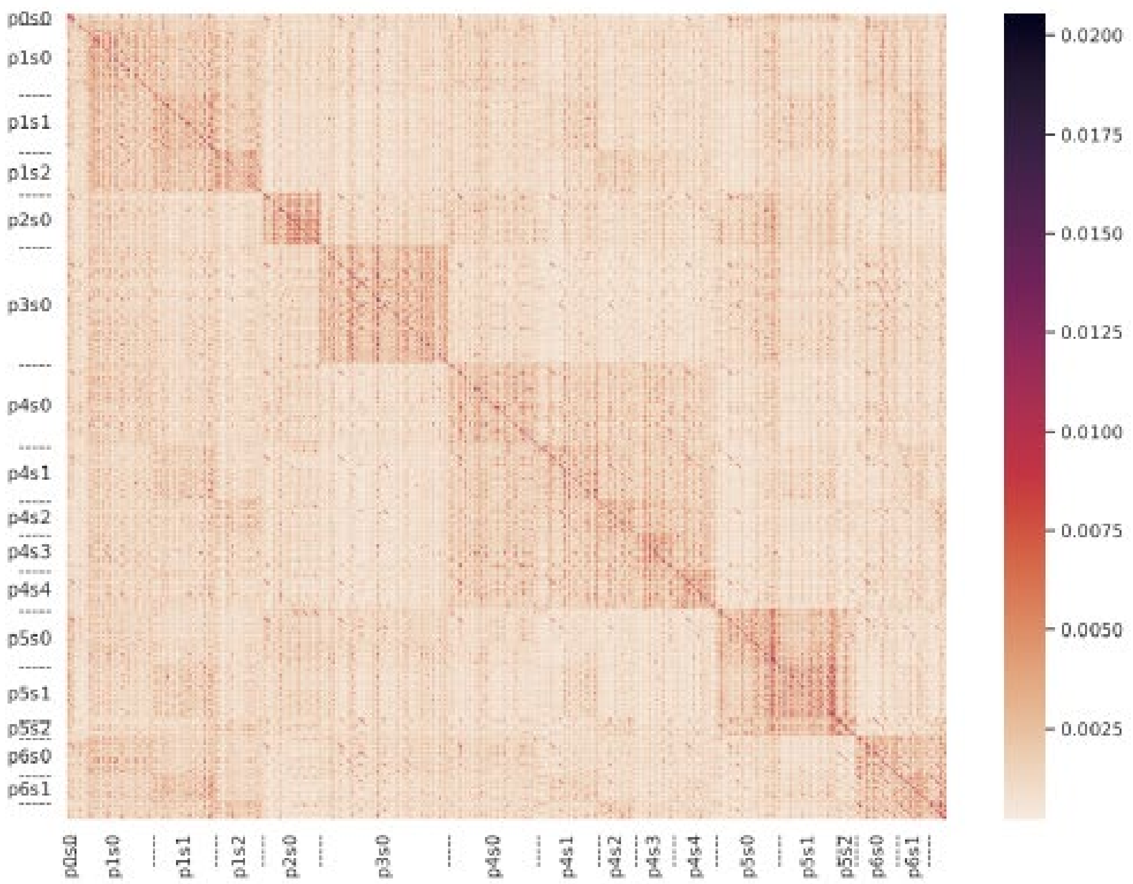
F1 = 2 (P*R) / (P+R), P is precision, R is recall, all in percentage, EM – exact match

Improving Transformer-XL

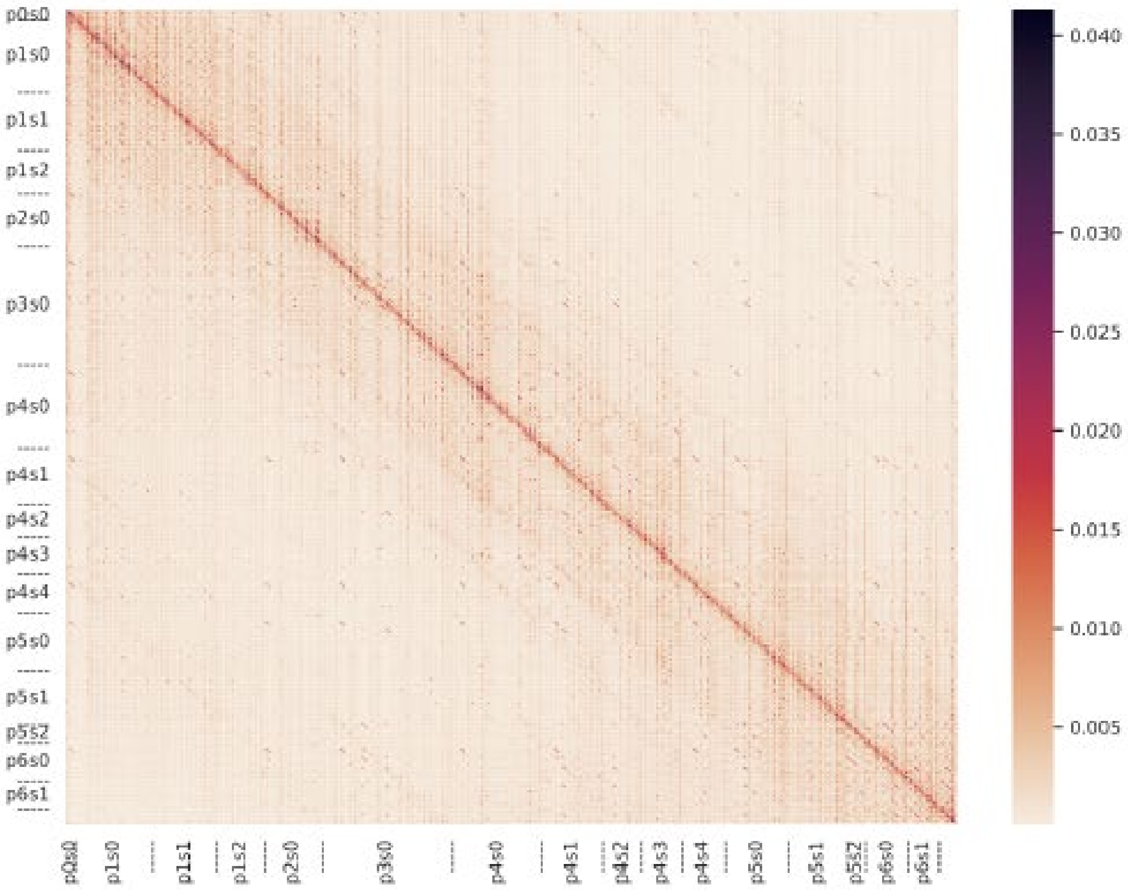
Model	#Param.	PPL
LSTM+Neural cache (Grave et al., 2017)	-	40.8
Hebbian+Cache (Rae et al., 2018)	-	29.9
Transformer-XL base, M=150 (Dai et al., 2019)	151M	24.0
Transformer-XL base, M=150 (ours)	151M	24.4
SegaTransformer-XL base, M=150	151M	**22.5**
Adaptive Input (Baevski and Auli, 2019)	247M	18.7
Transformer-XL large, M=384 (Dai et al., 2019)	257M	18.3
Compressive Transformer, M=1024 (Rae et al., 2020)	257M	17.1
SegaTransformer-XL large, M=384	257M	**17.1**

Table 1: Comparison with Transformer-XL and competitive baseline results on WikiText-103.

Looking at Attention

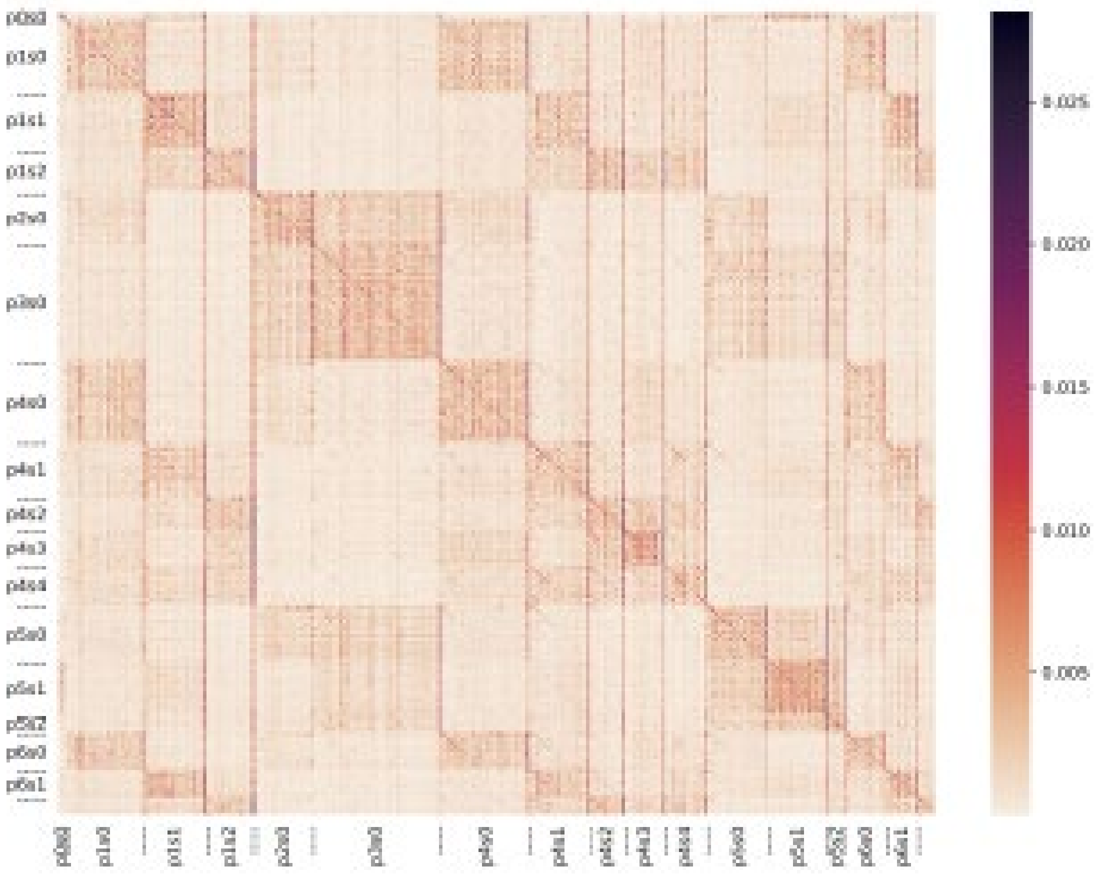


(a) SegaBERT-Layer 1

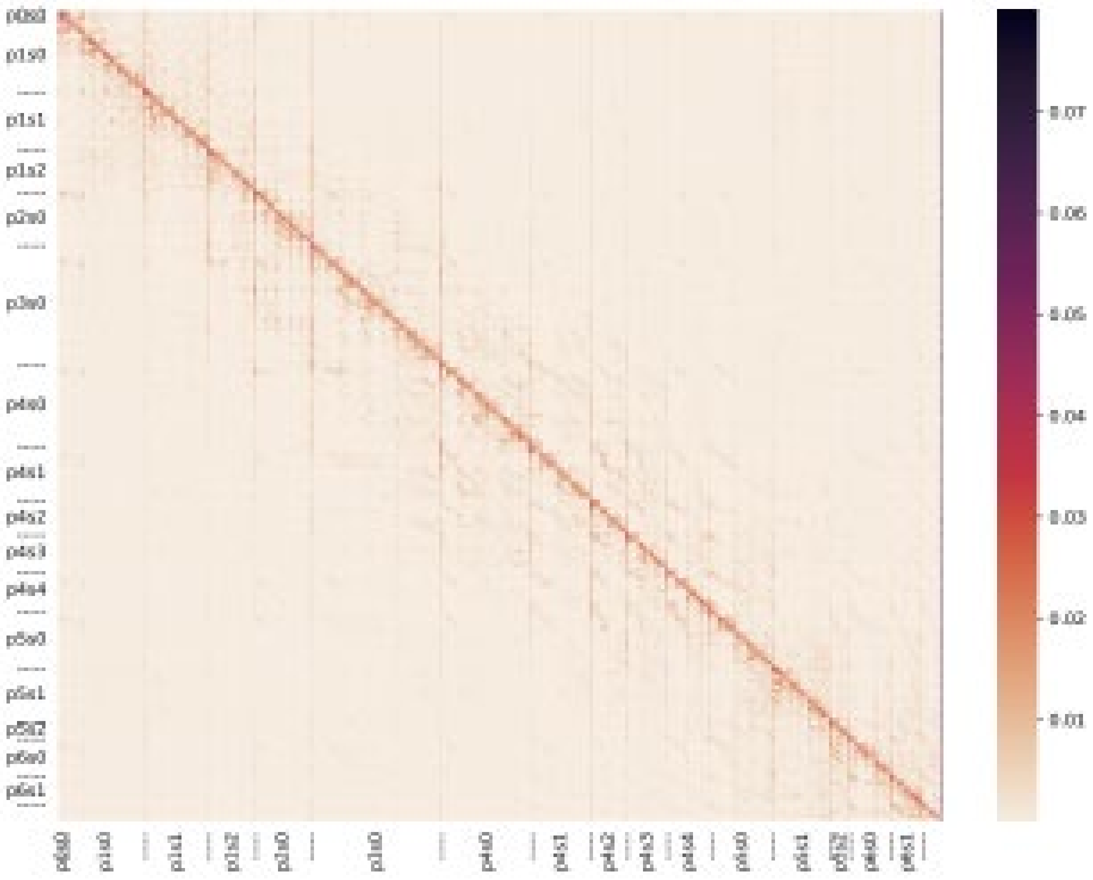


(b) BERT-Layer 1

Looking at Attention

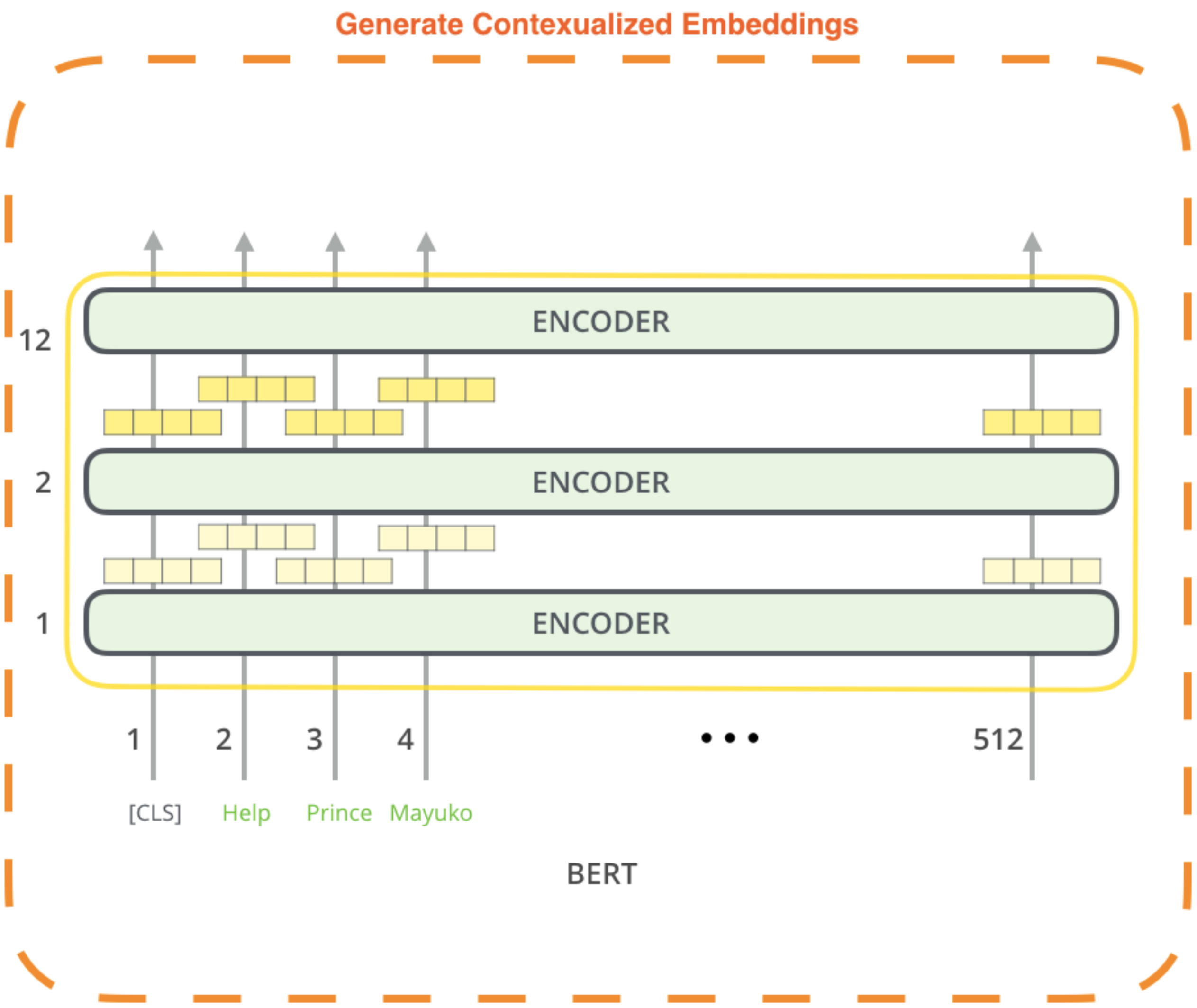


(a) SegaBERT-Layer 6

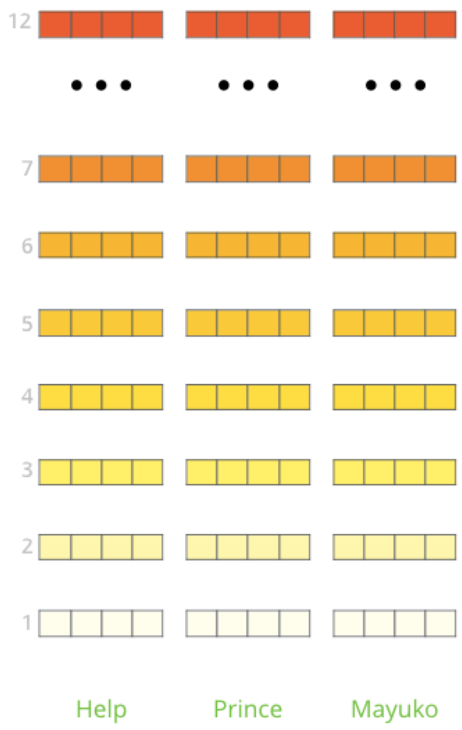


(b) BERT-Layer 6

Feature Extraction



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



We end up with some embedding for each word related to current input

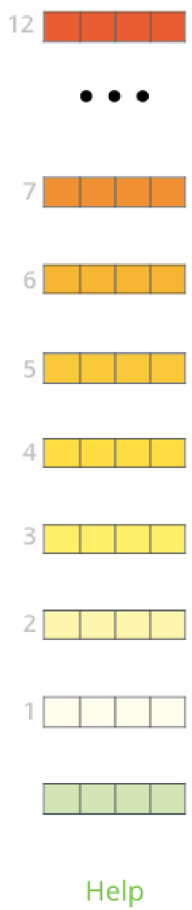
We start with independent word embedding at first level

But which one should we use?

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



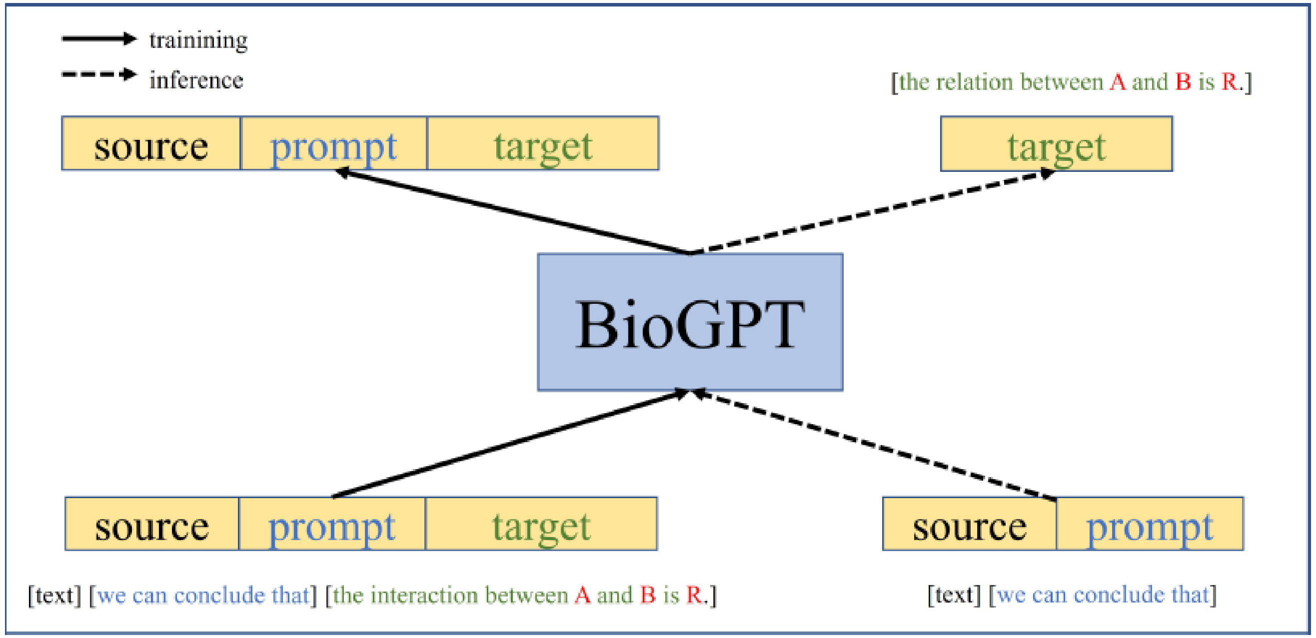
	Layers	Dev F1 Score
First Layer	Embedding	91.0
Last Hidden Layer	12	94.9
Sum All 12 Layers	12 + ... + 2 + 1 =	95.5
Second-to-Last Hidden Layer	11	95.6
Sum Last Four Hidden	12 + 11 + 10 + 9 =	95.9
Concat Last Four Hidden	9 10 11 12	96.1

BioGPT

BioGPT was fine-tuned and evaluated on several downstream tasks, including NER, QA, relation extraction, and document classification.

- Hard prompts: Hard prompts are manually designed discrete language phrases or templates that are prepended to the input text to guide the language model towards a specific task.

- Soft prompts, on the other hand, are continuous embeddings learned during the fine-tuning process.



"We have that [head entity] [relation] [tail entity]," "In conclusion, [head entity] [relation] [tail entity]," and "We can conclude that [head entity] [relation] [tail entity]."

Fig. 1. Framework of BioGPT when adapting to downstream tasks

BioBERT

- First, BioBERT is **intialized with weights from** BERT, which was pretrained on general domain corpora (English Wikipedia and BooksCorpus).
- Then, BioBERT is **pre-trained on biomedical domain corpora** (PubMed abstracts and PMC full-text articles).
- Finally, BioBERT is **fine-tuned** and evaluated on three popular biomedical text mining tasks (NER, RE and QA).

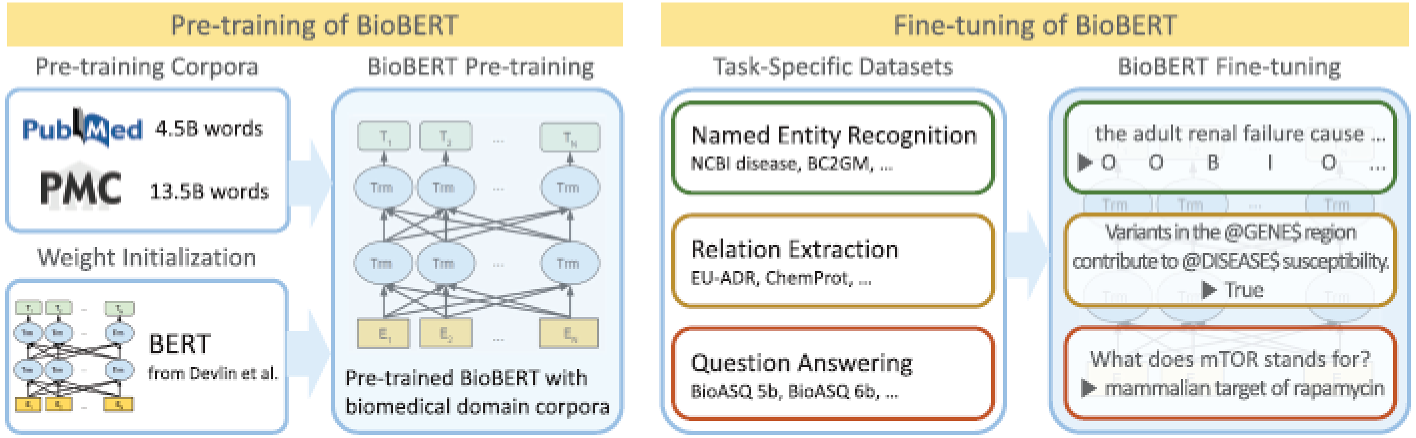


Fig. 1. Overview of the pre-training and fine-tuning of BioBERT

ClinicalBERT

The model is pre-trained on a large corpus of clinical notes from the Medical Information Mart for Intensive Care III (MIMIC-III) dataset, which contains de-identified electronic health records of patients admitted to the intensive care unit.

The main goal of ClinicalBERT is to improve the prediction of hospital readmission within 30 days based on the information contained in clinical notes.

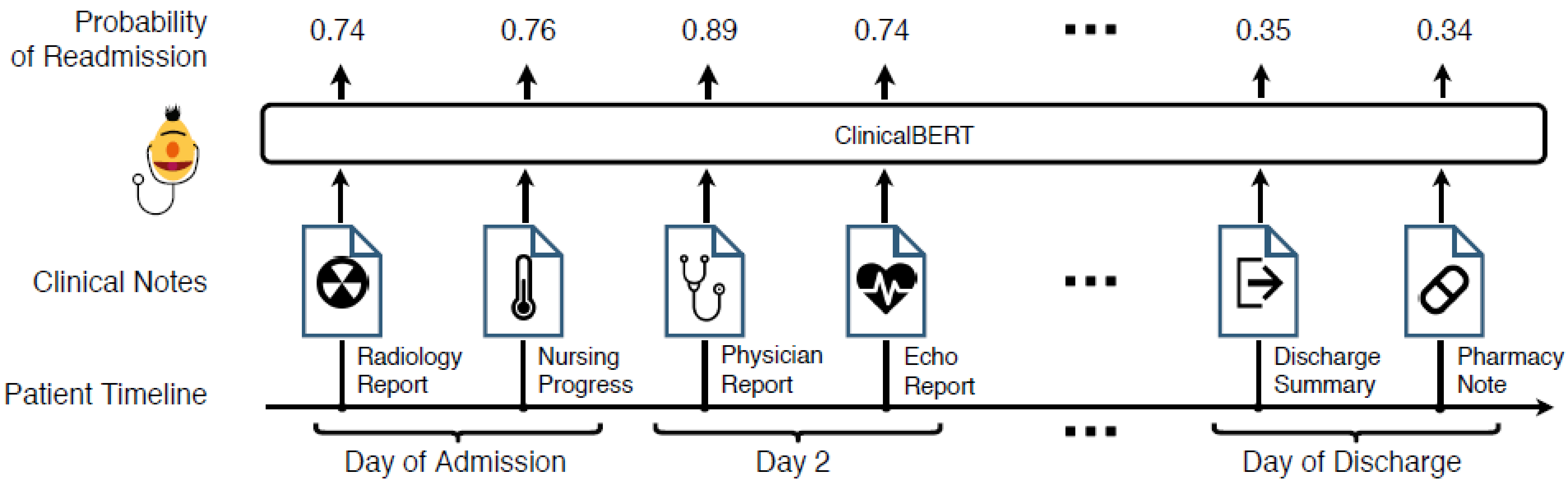


Figure 1: ClinicalBERT learns deep representations of clinical notes that are useful for tasks such as readmission prediction. In this example, care providers add notes to an electronic health record during a patient's admission, and the model dynamically updates the patient's risk of being readmitted within a 30-day window.

Thank you!

Literature & Resources for Transformers

Resources:

OpenAI GPT-2 implementation: https://github.com/openai/gpt-2

BERT paper: J. Devlin et al, BERT, pretraining of deep bidirectional transformers for language understanding. Oct. 2018.

ELMo paper: M. Peters, et al, Deep contextualized word representation, 2018

ULM-FiT paper: Universal language model fine-tuning for text classification. J. Howeard, S. Ruder., 2018

Jay Alammar, The illustrated GPT-2, https://jalammar.github.io/illustrated-gpt2/