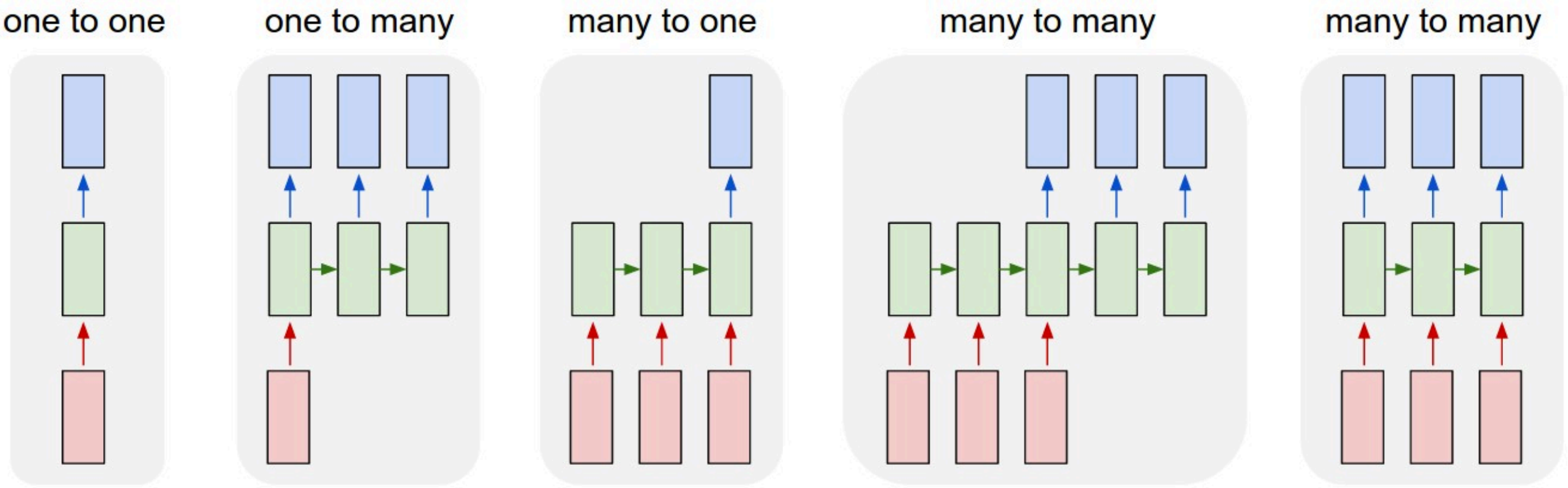
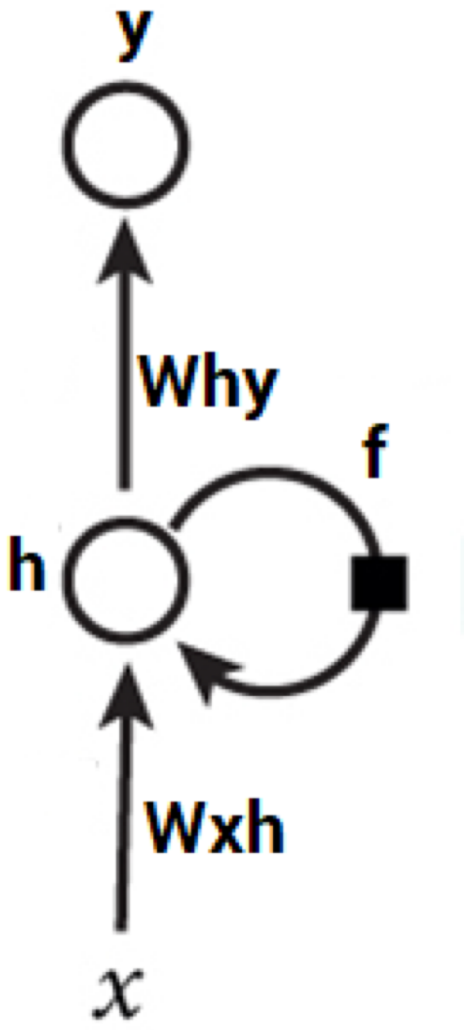
# Attention, Transformers, BERT, and Other Modern Sequence Processing

# Review – Sequence Problems



# Recurrent Neural Networks




A sequence of vectors, **x**, can be processed by applying the following recurrence relation at each time step:

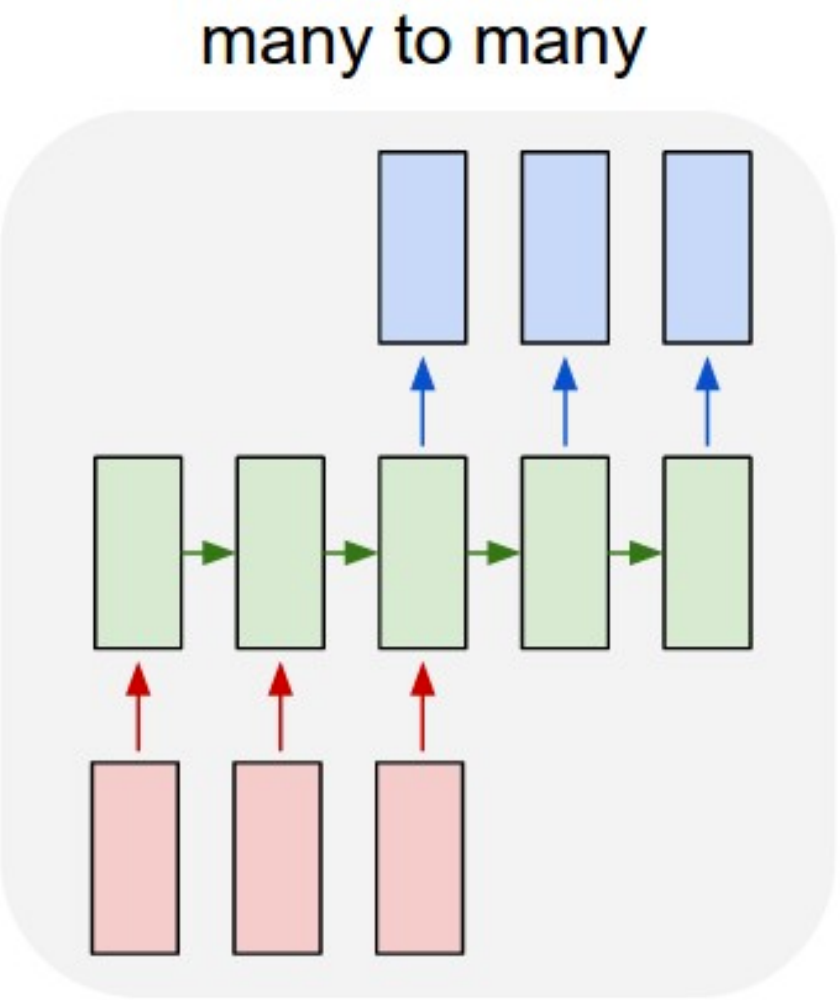
$$h_t = f_W(h_{t-1}, x_t)$$

Key idea: the same parameters are used at each time step

# Problem

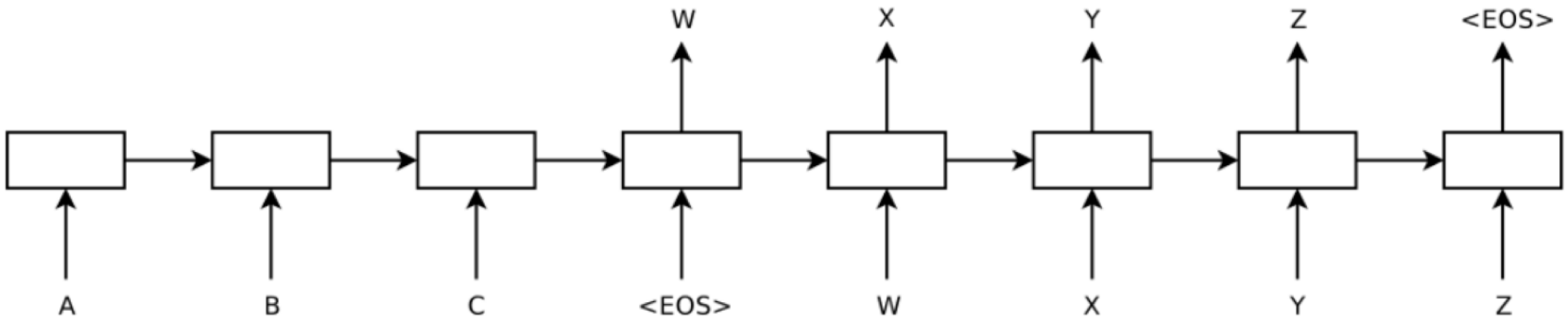
- Recurrent neural networks are difficult to train on long sequences
  - Machine Translation – difficult to summarize an entire sentence in one feature vector

- Today we will introduce “Attention”, a mechanism to improve long-term relationships in sequences



# Review – Sequence to Sequence Translation

- Encoder/Decoder model used for machine translation:



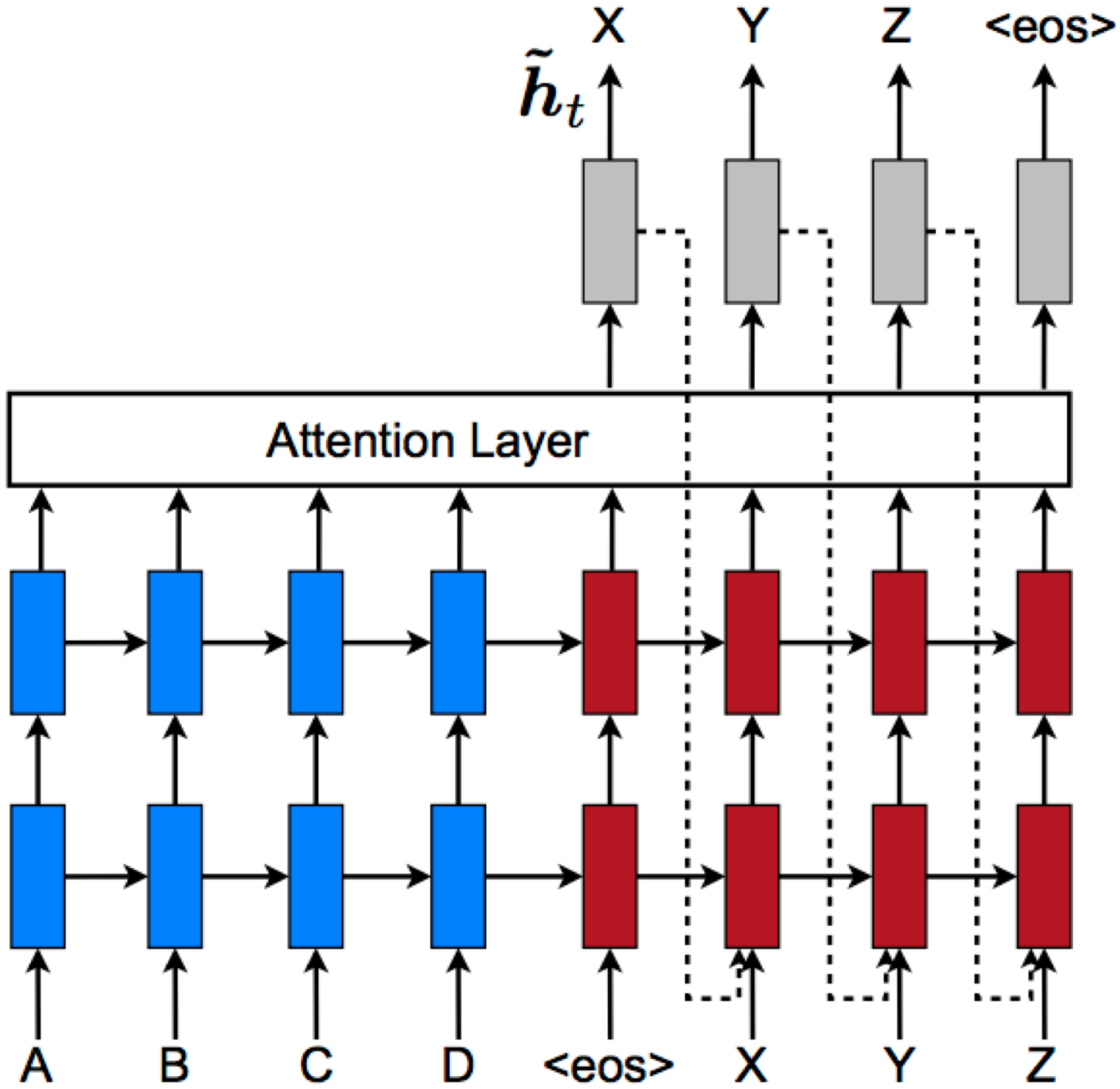

- Network reads a sentence and stores the information in the hidden units

# Attention

- Define relationship between points in the input and output sequence

- During encoding, each word is represented as a vector
  - Similar to hidden state in previous RNN models
- During decoding, input to hidden state is linear combination of input states
  - Corresponding to a weighted average of the important of each input state for that particular output state

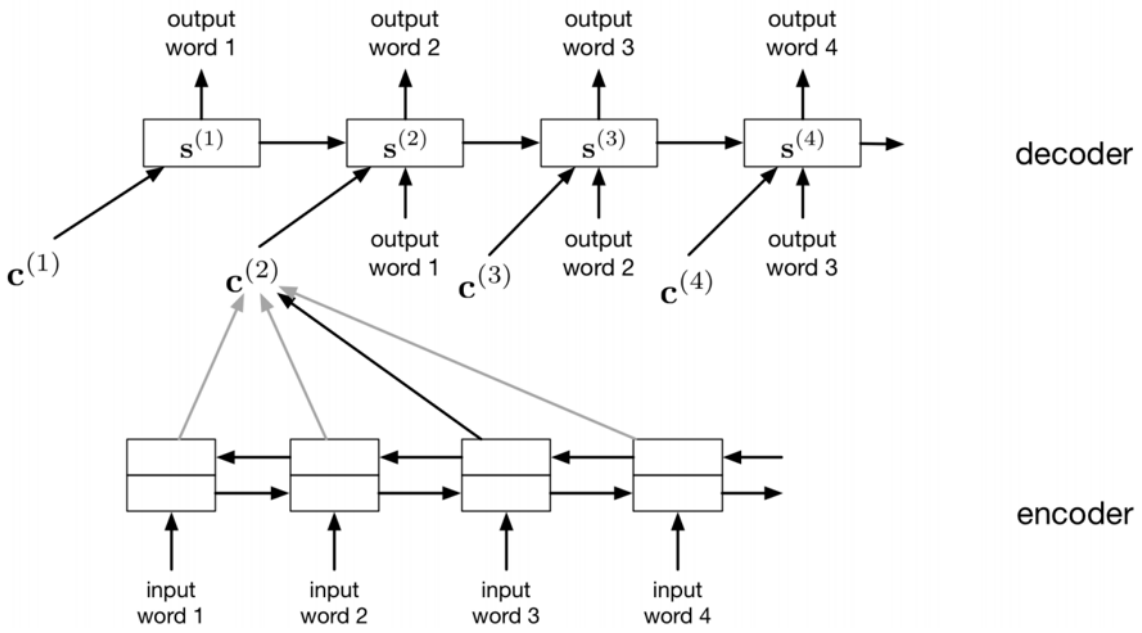
# Attention



# Attention Layer

- The decoder network is an RNN. Like the encoder/decoder translation model, it makes predictions one word at a time, and the predictions are fed back in as inputs.

- In the attention layer, a context vector is used as the input with the other information.



# Attention

- The context vector is computed as a weighted average of the encoder's annotations

$$c^i = \sum_j a_{ij} h^j$$

- Attention weights are computed as a softmax where the inputs annotations depend on the annotation and decoder states:

$$a_{ij} = \frac{\exp e_{ij}}{\sum'_j e_{ij'}}$$

$$e_{ij} = a\left(s^{i-1}, h^j\right)$$

- The attention function ($a(s, h)$) depends on the annotation vector rather than the position in the sentence

# Attention layer calculation

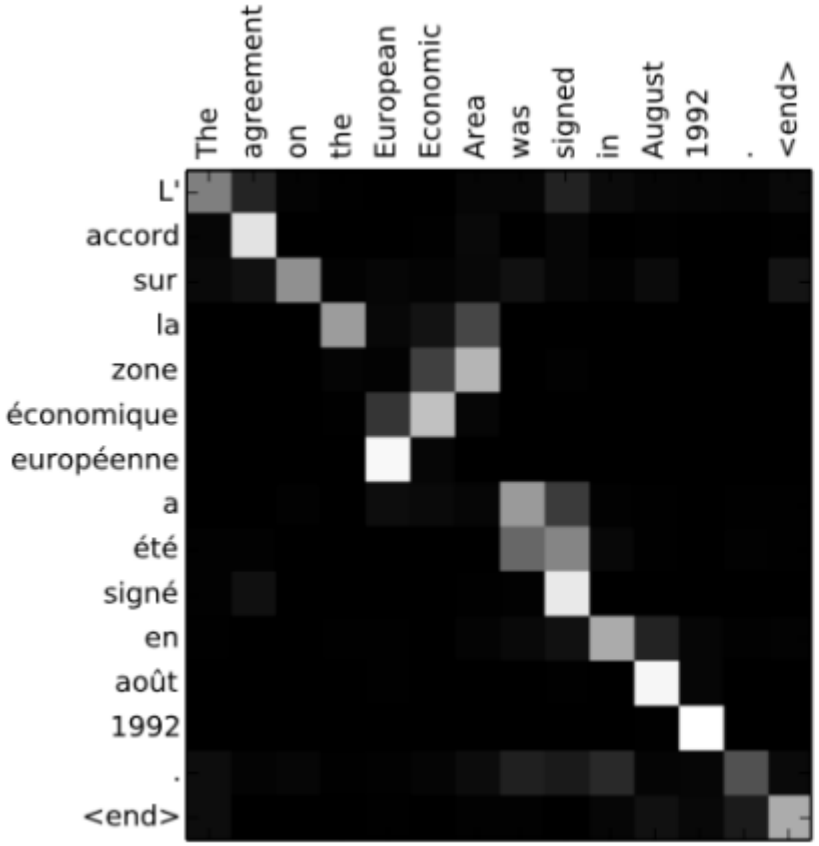
- Attention weights are computed as a softmax where the inputs annotations depend on the annotation and decoder states:

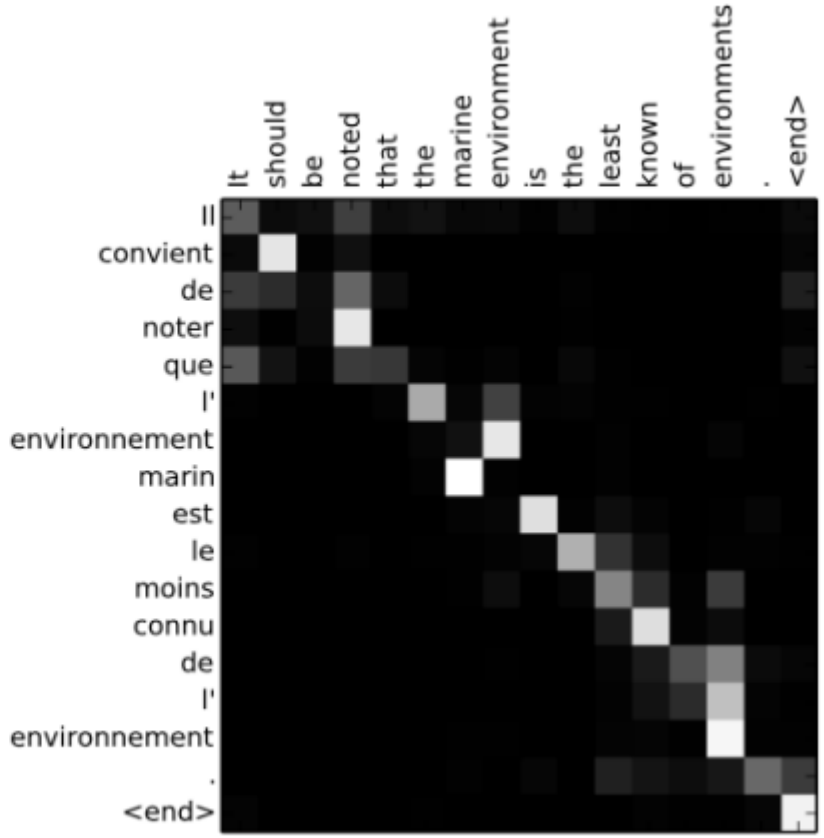
$$a_{ij} = \frac{\exp e_{ij}}{\sum'_j e_{ij'}}$$

$$e_{ij} = a\left(s^{i-1}, h^j\right)$$

- $a\left(s^{i-1}, h^j\right)$ can be any type of network (commonly a fully-connected network) where the output is a score.
  - Score is how strong the relationship between $j$ in the input and $i$ in the output

# Attention in practice

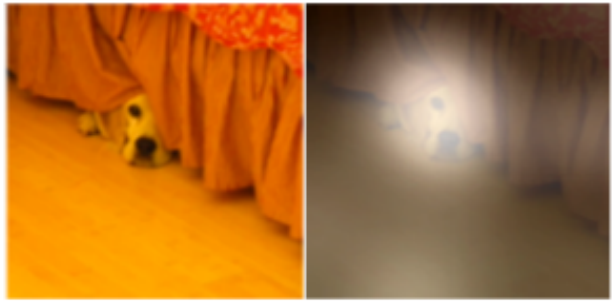



In French to English translation, the relative position of adjectives and nouns can change. Attention can account for that in the model.

# Attention in Other Contexts


A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.

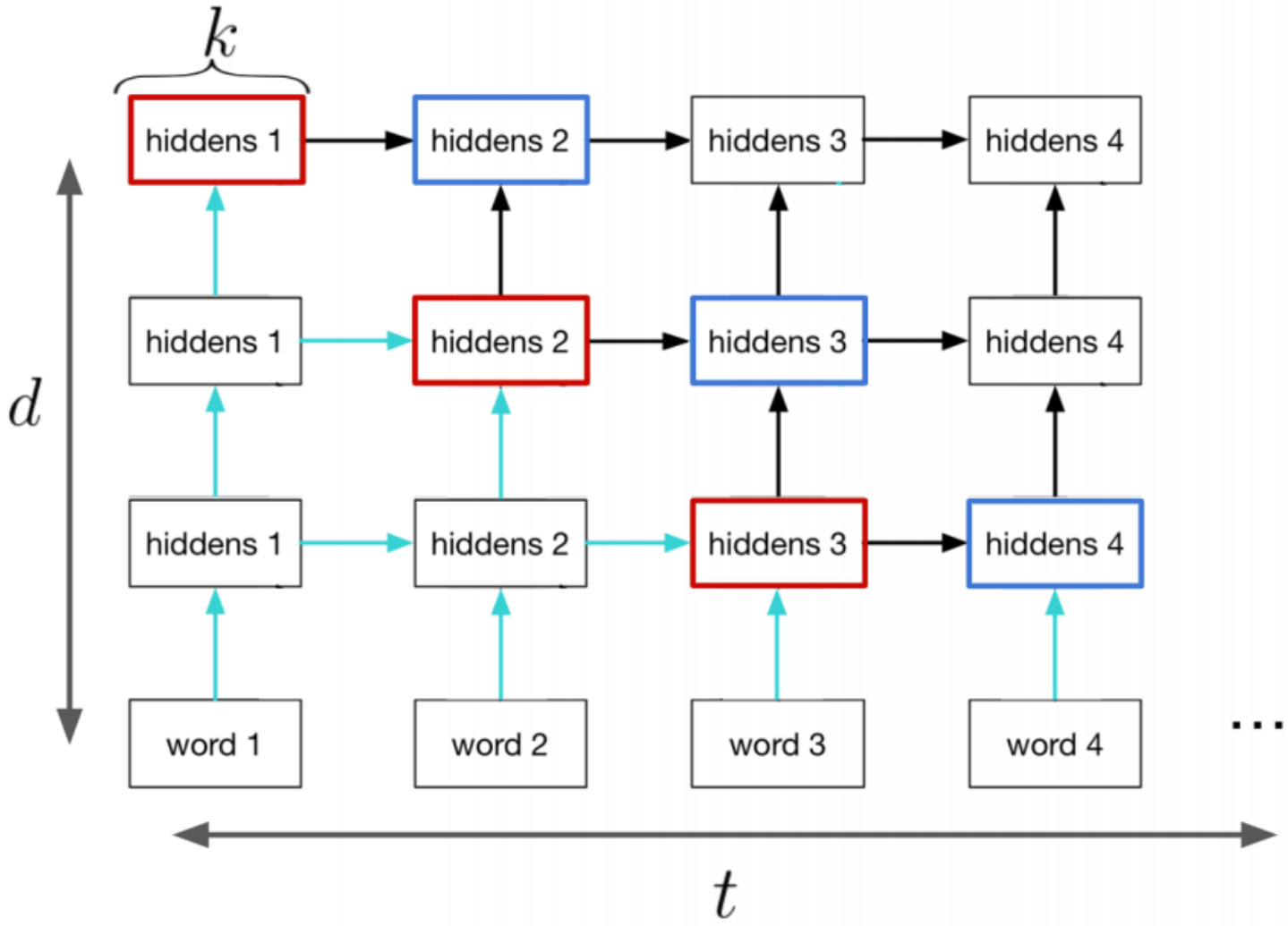

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

# Challenge: RNNs cannot work well in parallel



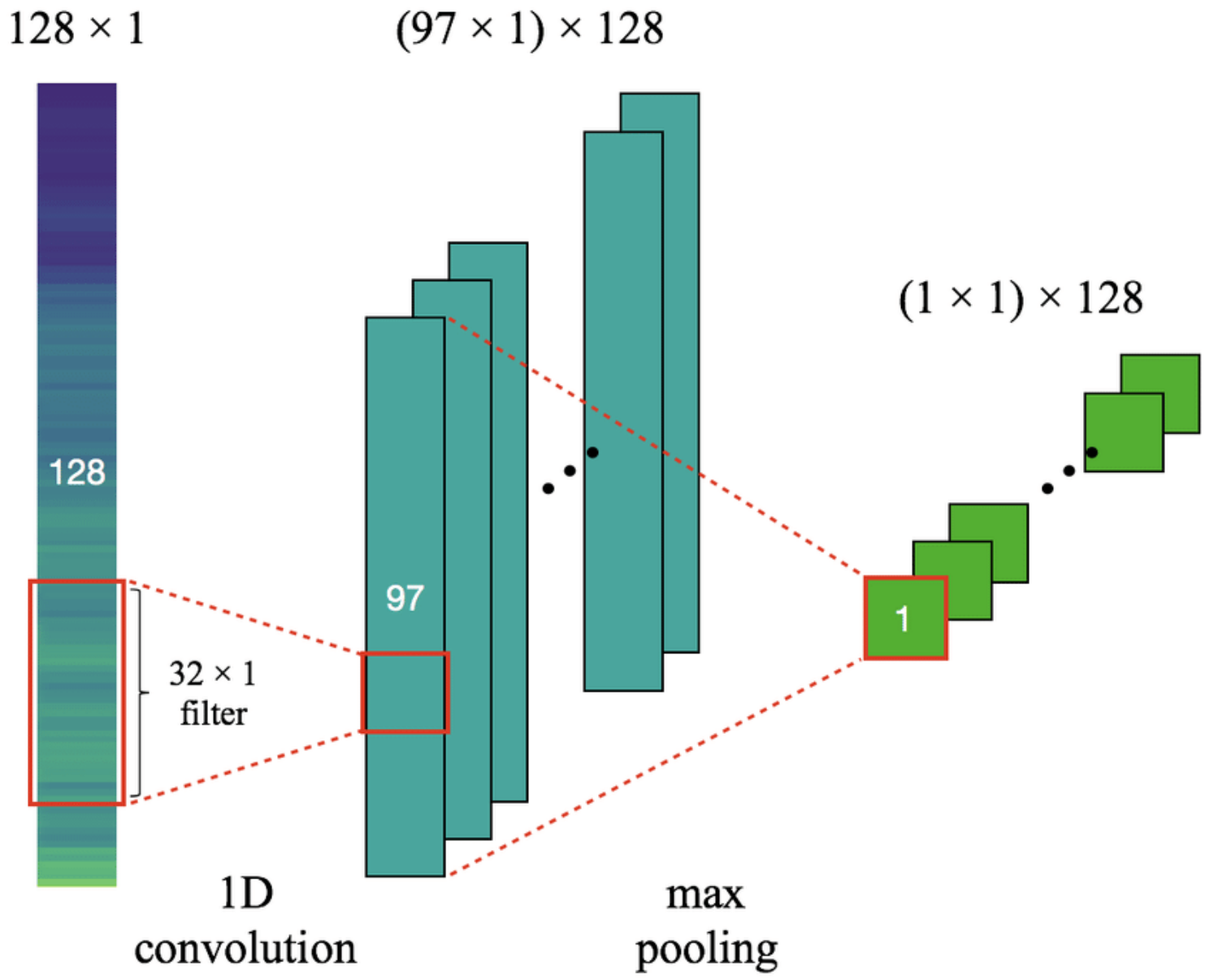
# RNNs can be complex, and not support parallel operations

- Inputs: t: length of sequence, d: # of layers, k: # of neurons

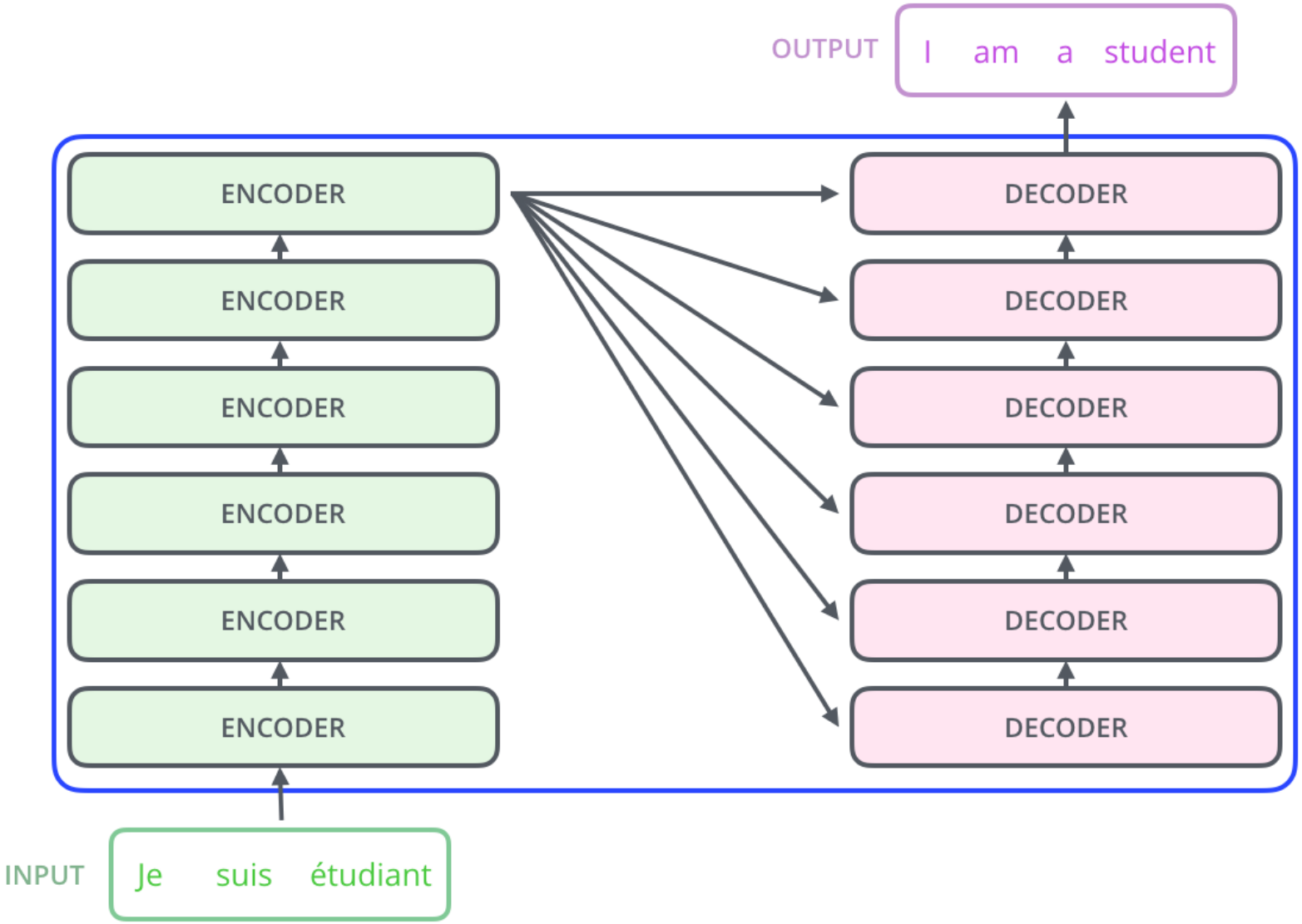
| Model | Training Complexity | Training Memory | Testing Complexity | Testing Memory |
|---|---|---|---|---|
| RNN | $t \times k^2 \times d$ | $t \times k \times d$ | $t \times k^2 \times d$ | $k \times d$ |
| RNN + Attn | $t^2 \times k^2 \times d$ | $t^2 \times k \times d$ | $t^2 \times k^2 \times d$ | $t \times k \times d$ |

- Attention requires additional information to recompute at every time step

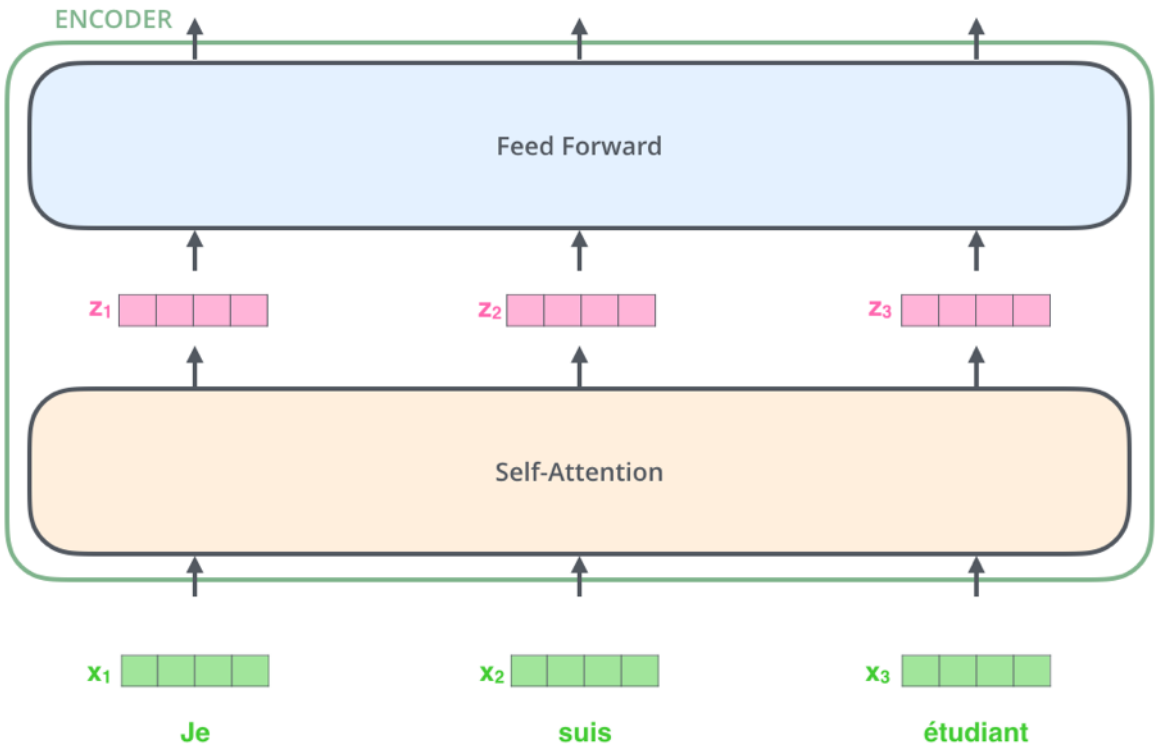
# 1D Convolution

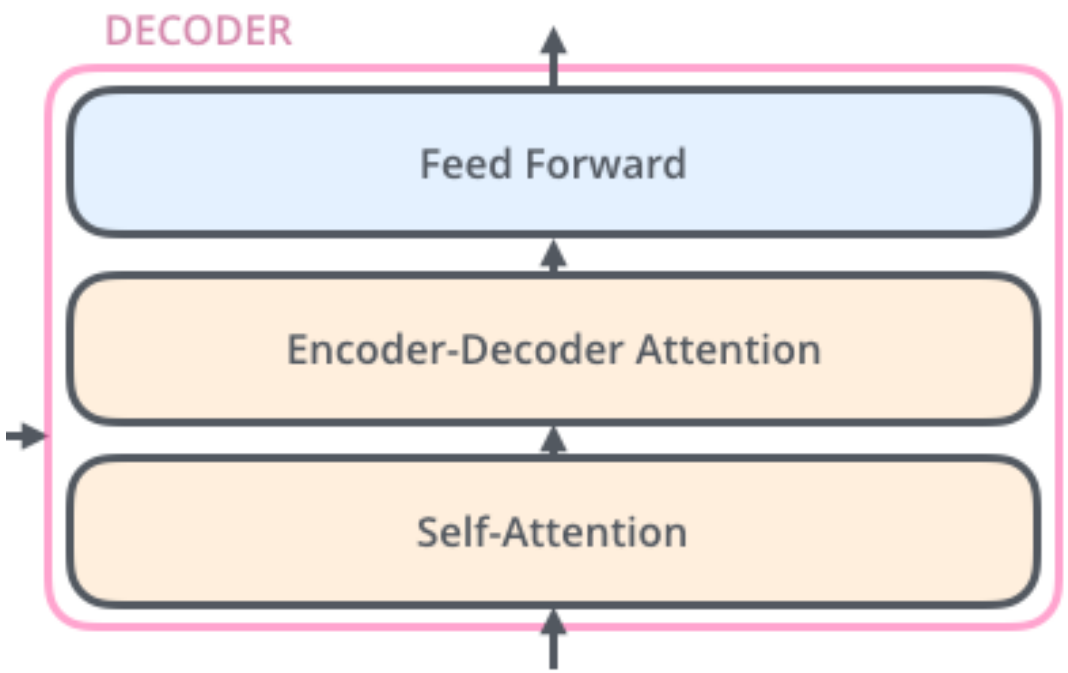


# Transformer Networks

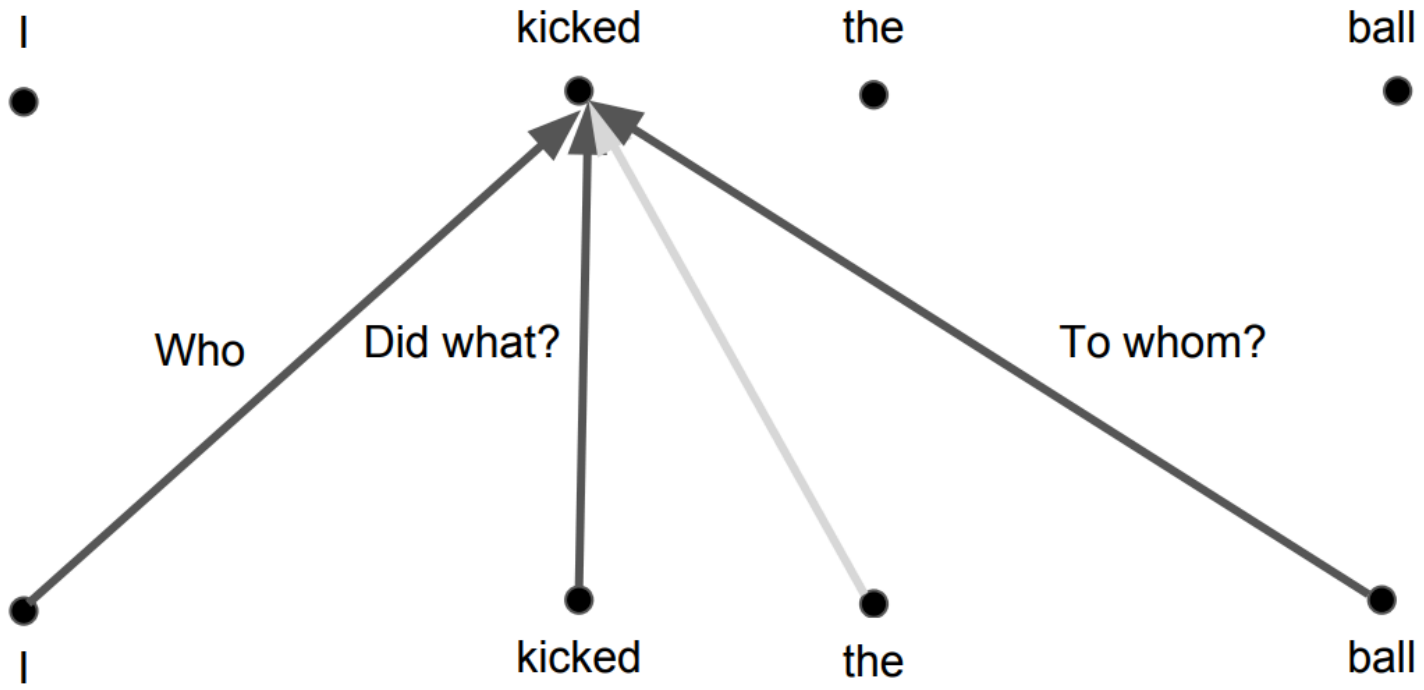


# Transformer Networks

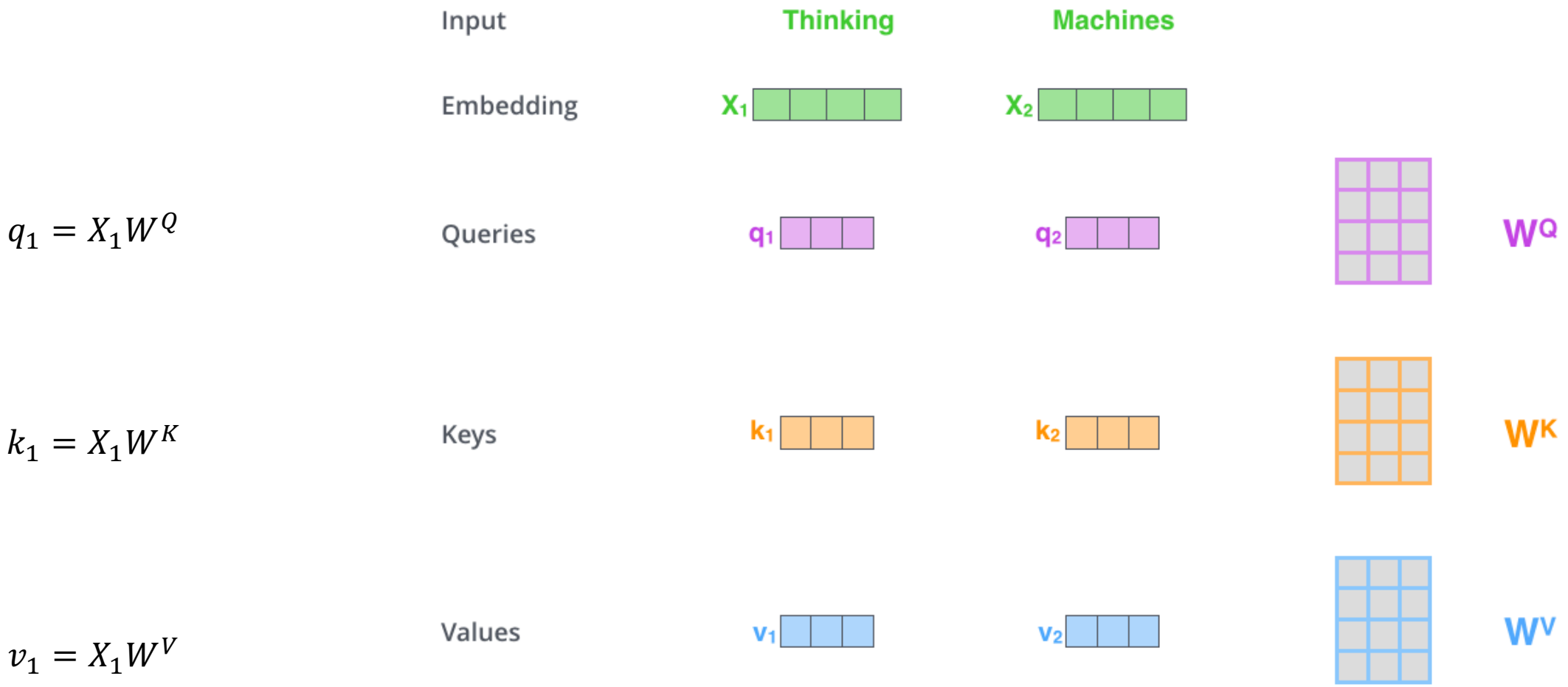


# Encoder Block – Self Attention



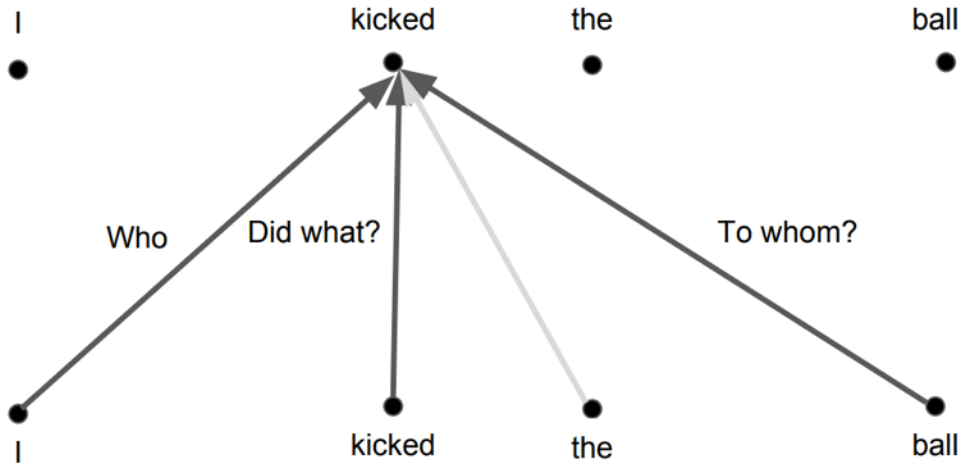
# Encoder Block – Self Attention



# Encoder Block – Self Attention

- Conceptually
  - $q_i \rightarrow$ represents a word we are scoring
  - $k_i \rightarrow$ represents a word we are assessing against $q_i$
  - $v_i \rightarrow$ output vector for term $i$

- $q$ and $k$ are used to assess the similarity of two words, $v$ is the output function for the weighted attention scoring



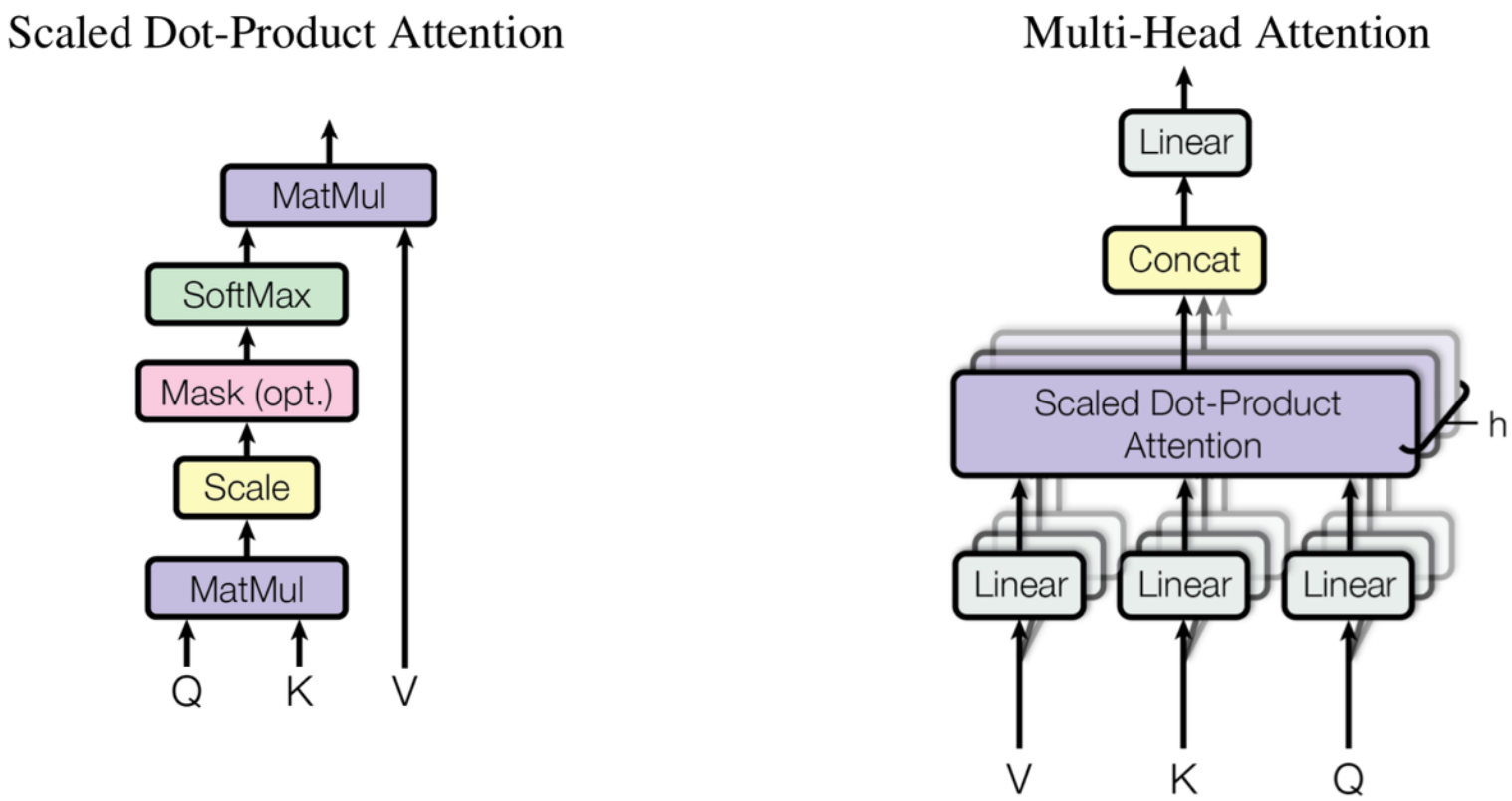
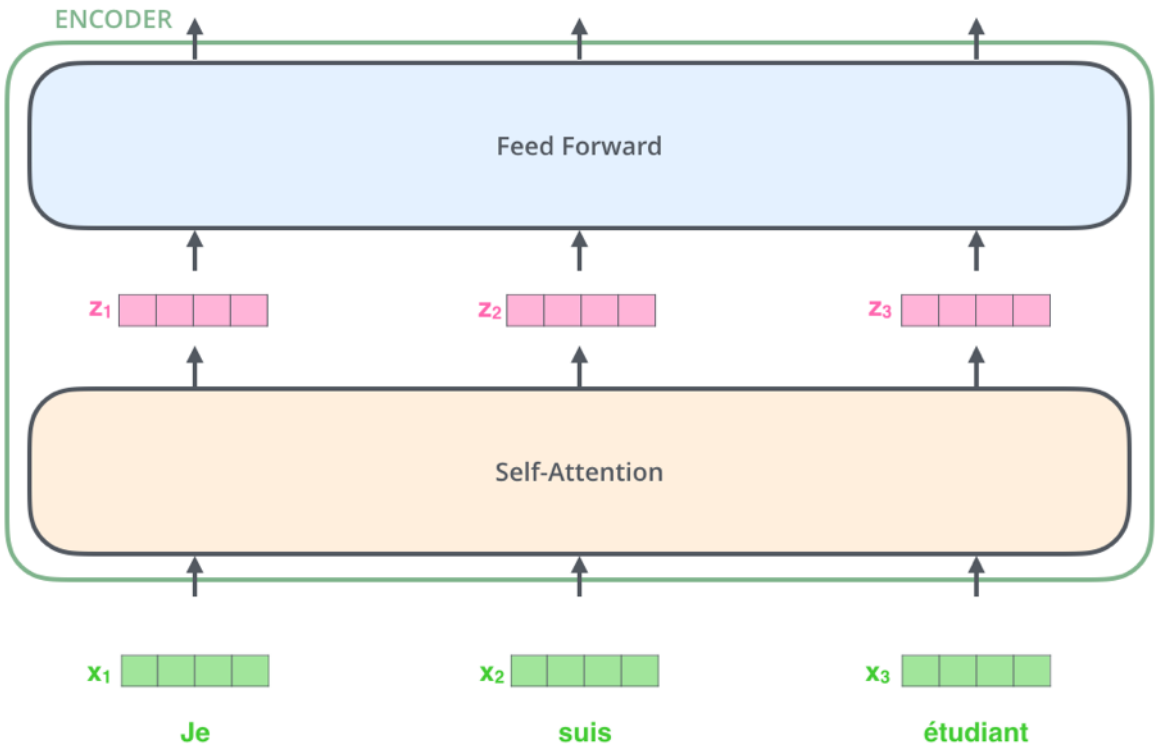
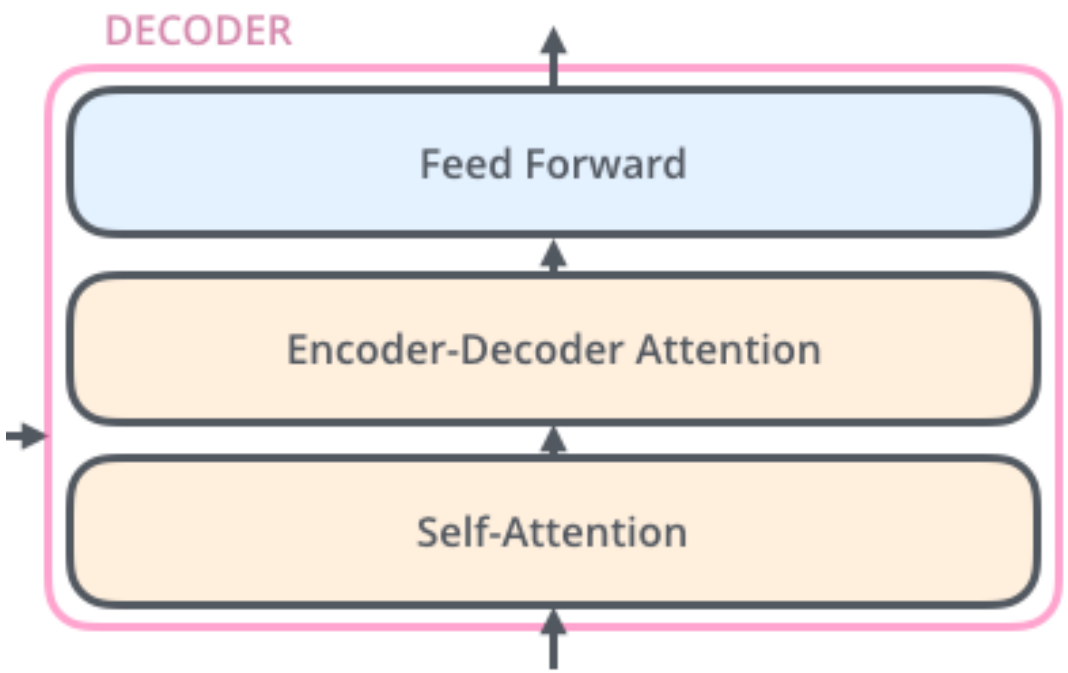


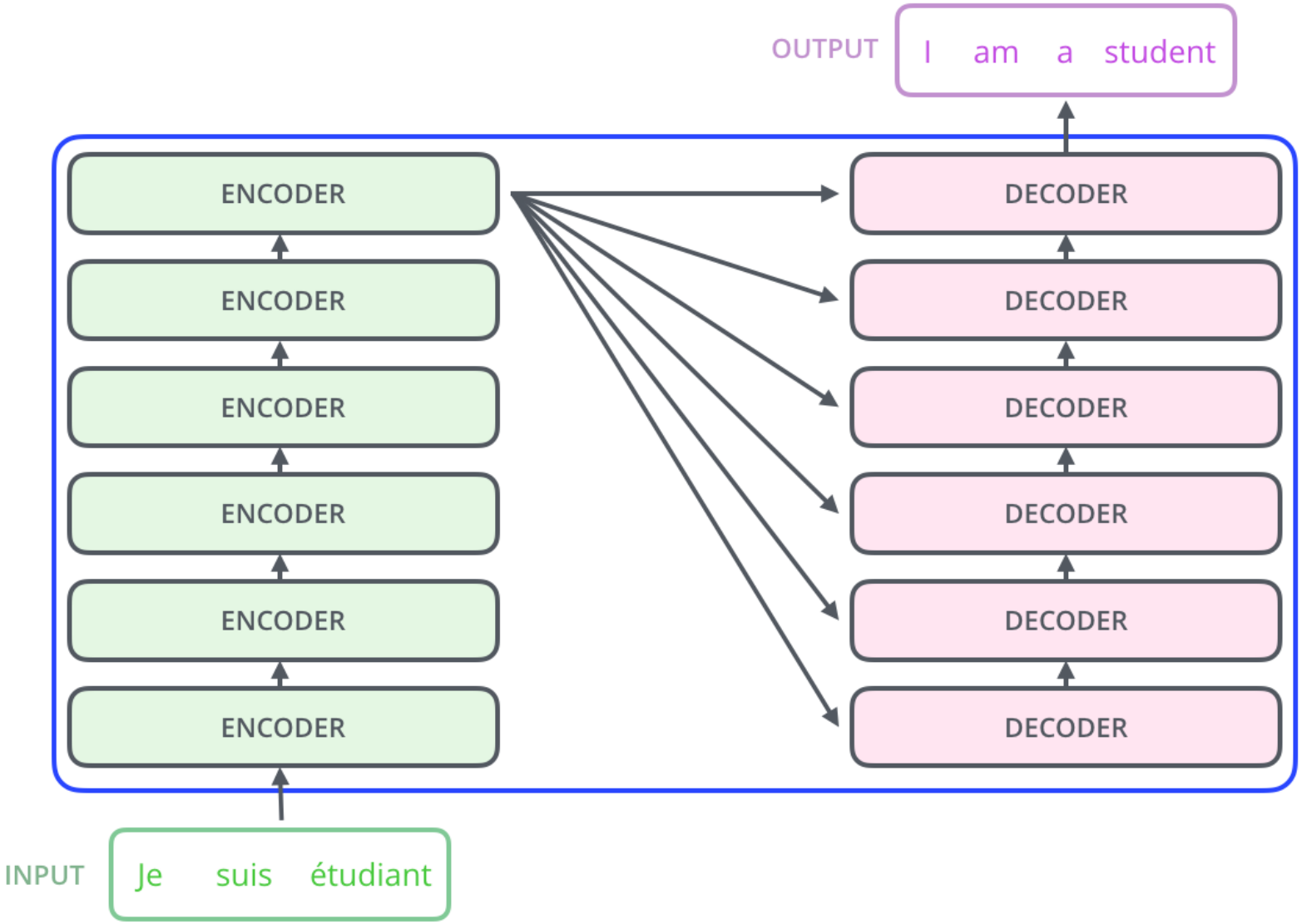

Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

# Transformer Networks

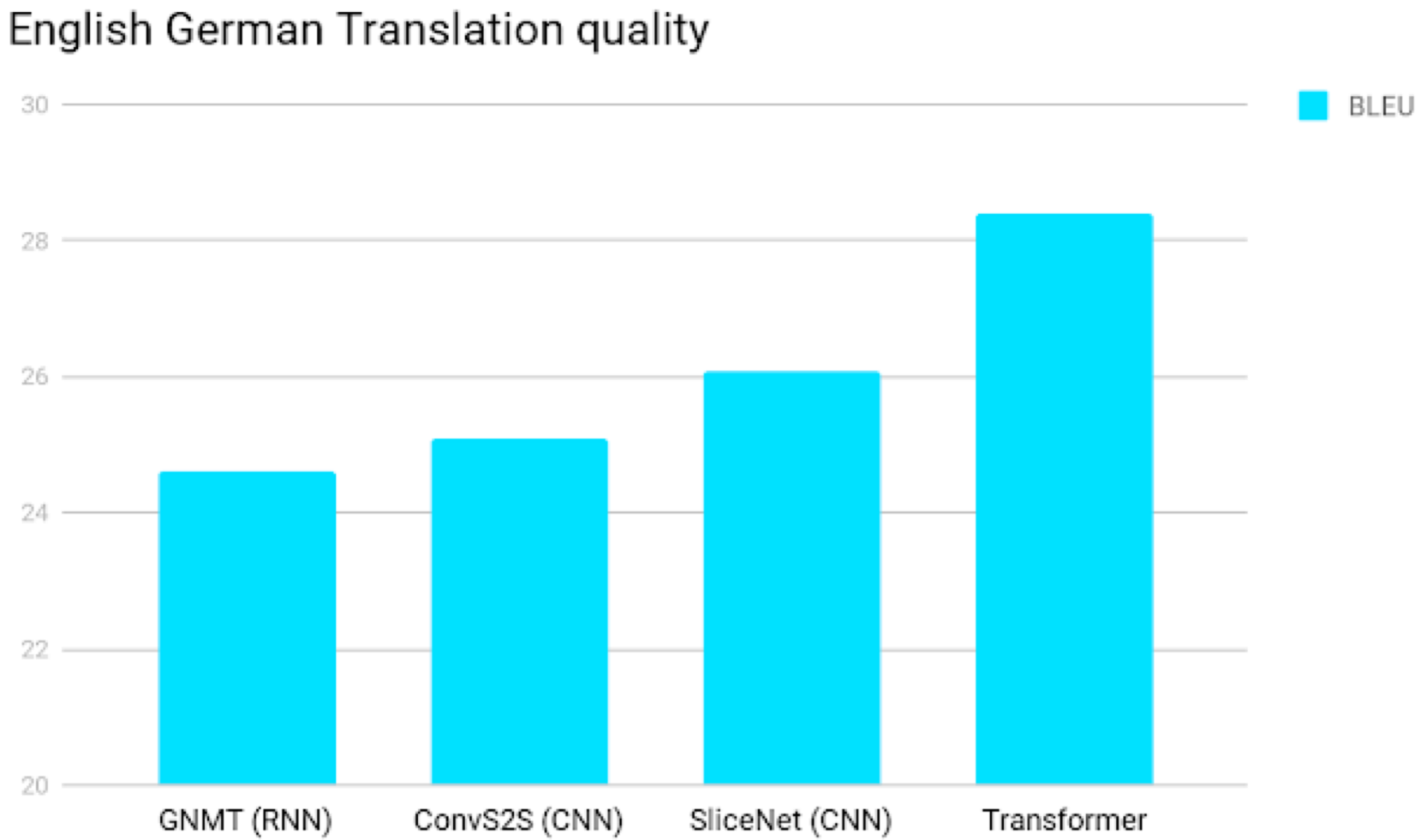


# Transformer Networks

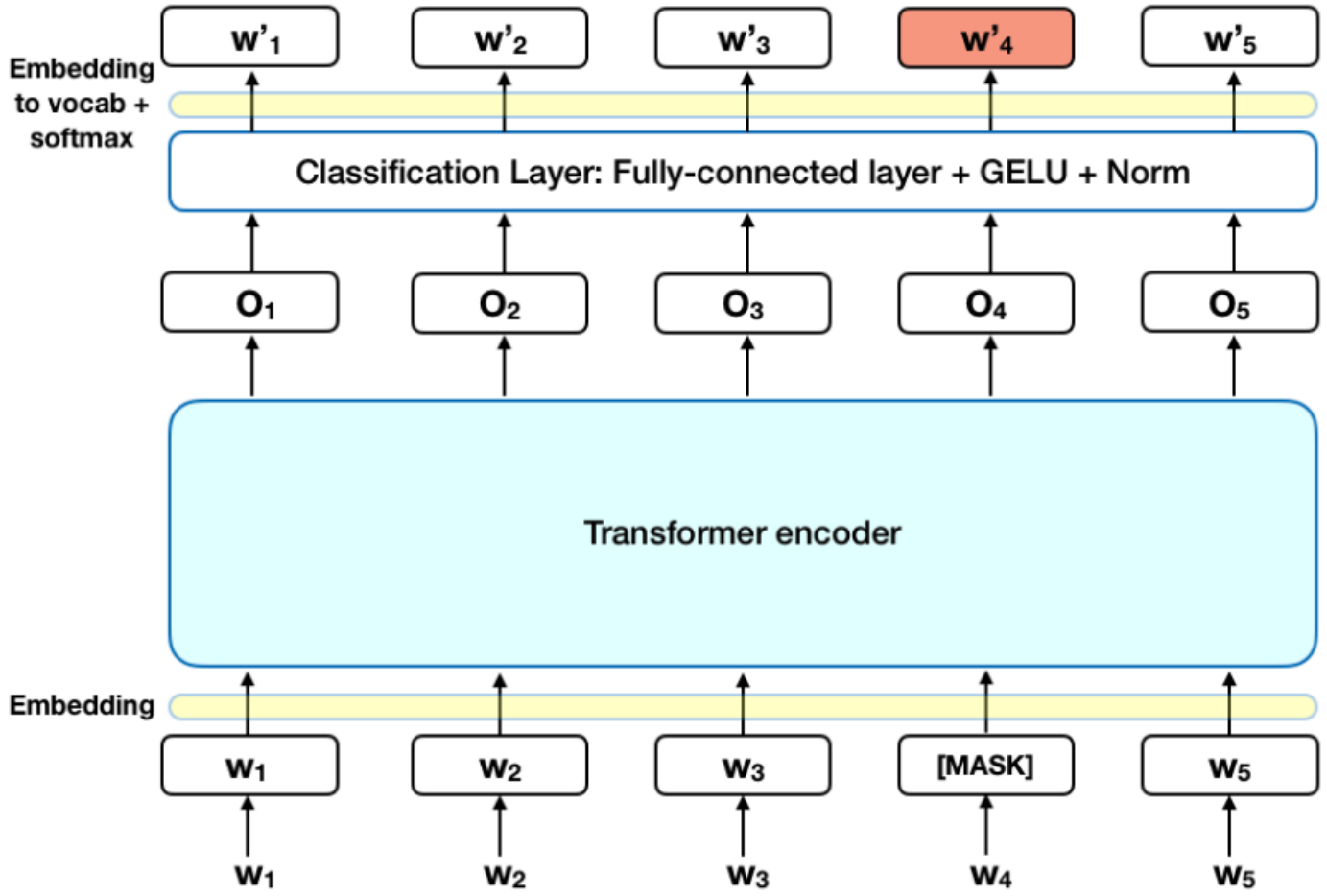


# Transformer Results



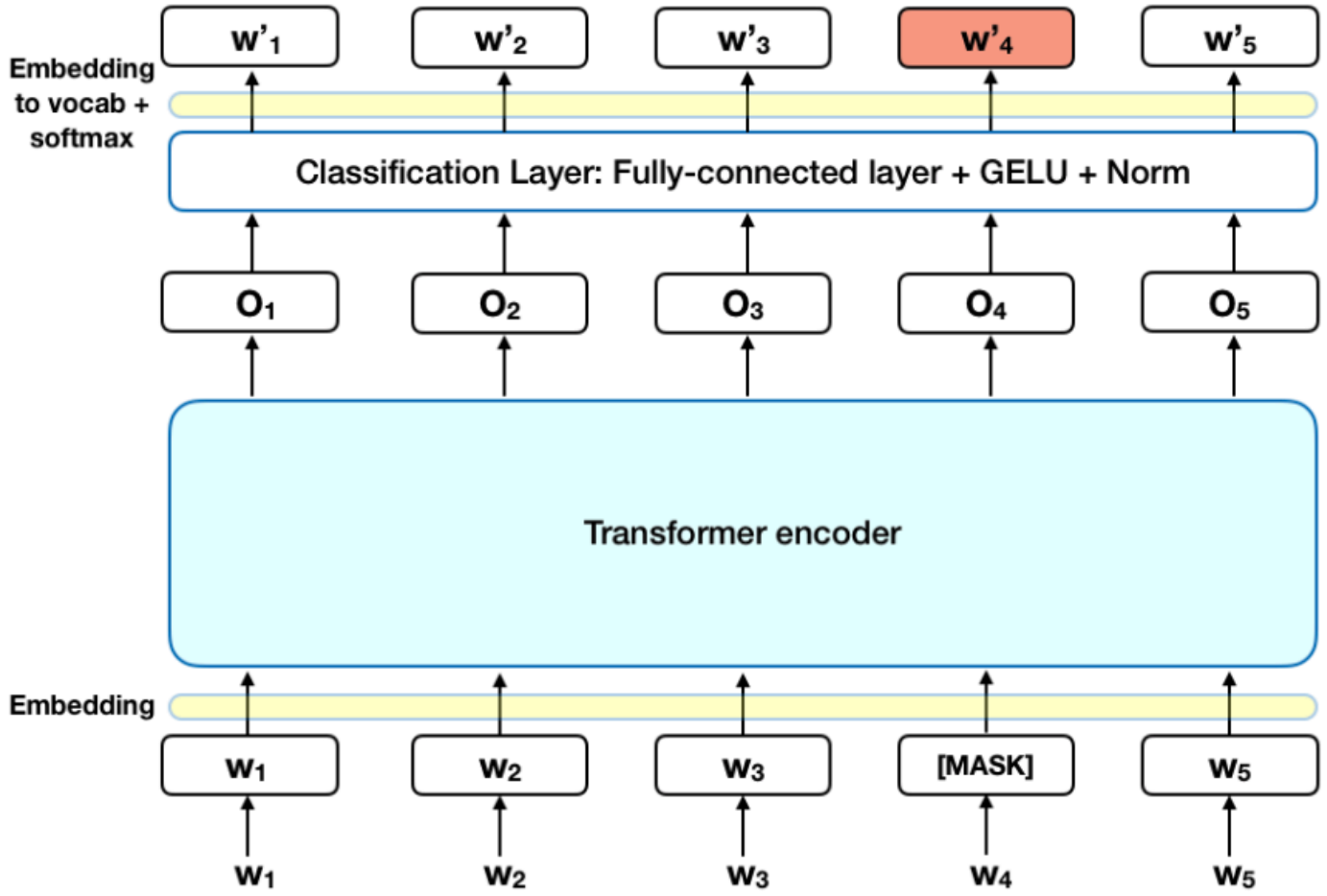
# BERT (similarly OpenAI GPT)

- BERT -
  Bidirectional Encoder Repre
  sentations
  from Transformers
- Trained as a deeply bidirectional, unsupervised text representation
  - Trained by reading Wikipedia



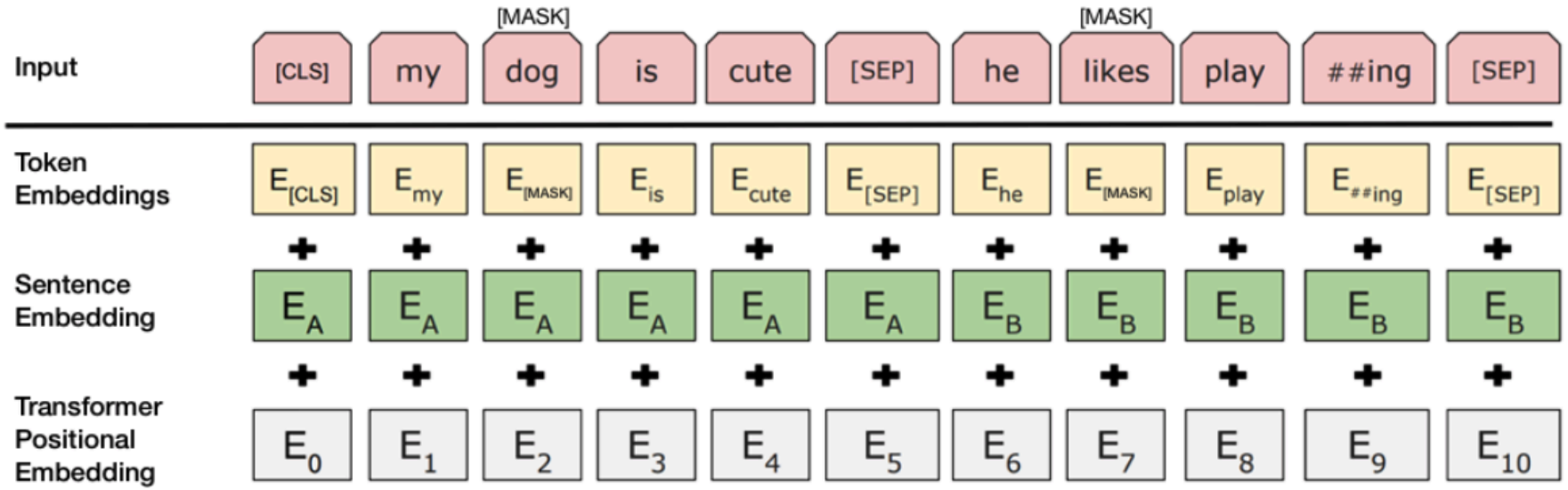
# BERT Training

- Training process – sentences are entered in pairs into BERT
  - 50% of the time the first sentence proceeds the second sentence, 50% of the time it does not
  - Goal is to predict if the two sentences are related



# BERT Training

- The entire input sequence goes through the Transformer model.
- The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
- Calculating the probability of IsNextSequence with softmax.



# Using BERT today

- https://github.com/google-research/bert

- Classification tasks such as sentiment analysis are done similarly to Next Sentence classification, by adding a classification layer on top of the Transformer output for the [CLS] token.
- In Question Answering tasks (e.g. SQuAD v1.1), the software receives a question regarding a text sequence and is required to mark the answer in the sequence. Using BERT, a Q&A model can be trained by learning two extra vectors that mark the beginning and the end of the answer.
- In Named Entity Recognition (NER), the software receives a text sequence and is required to mark the various types of entities (Person, Organization, Date, etc) that appear in the text. Using BERT, a NER model can be trained by feeding the output vector of each token into a classification layer that predicts the NER label.

# Other BERT Takeaways

1. *Model size matters, even at huge scale.* BERT_large, with 345 million parameters, is the largest model of its kind. It is demonstrably superior on small-scale tasks to BERT_base, which uses the same architecture with "only" 110 million parameters.
2. *With enough training data, more training steps == higher accuracy.* For instance, on the MNLI task, the BERT_base accuracy improves by 1.0% when trained on 1M steps (128,000 words batch size) compared to 500K steps with the same batch size.

| 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 |
| $\text{BERT}_{\text{BASE}}$ | 84.6/83.4 | 71.2 | 90.1 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| $\text{BERT}_{\text{LARGE}}$ | **86.7/85.9** | **72.1** | **91.1** | **94.9** | **60.5** | **86.5** | **89.3** | **70.1** | **81.9** |