STAT 453: Introduction to Deep Learning and Generative Models

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching

Lecture 19

RNNs and Transformers for Sequence-to-Sequence Modeling

Lecture Topics

1. Sequence Generation with RNNs
2. Character RNN in PyTorch
3. RNNs with Attention
4. Attention is All We Need
5. Transformer Models
6. Transformer in PyTorch

Many-to-Many RNNs for Generating Text

1. **Sequence Generation with RNNs**
2. Character RNN in PyTorch
3. RNNs with Attention
4. Attention is All We Need
5. Transformer Models
6. Transformer in PyTorch

Different Types of Sequence Modeling Tasks

Previously, we built an
(Word-level) RNN classifier

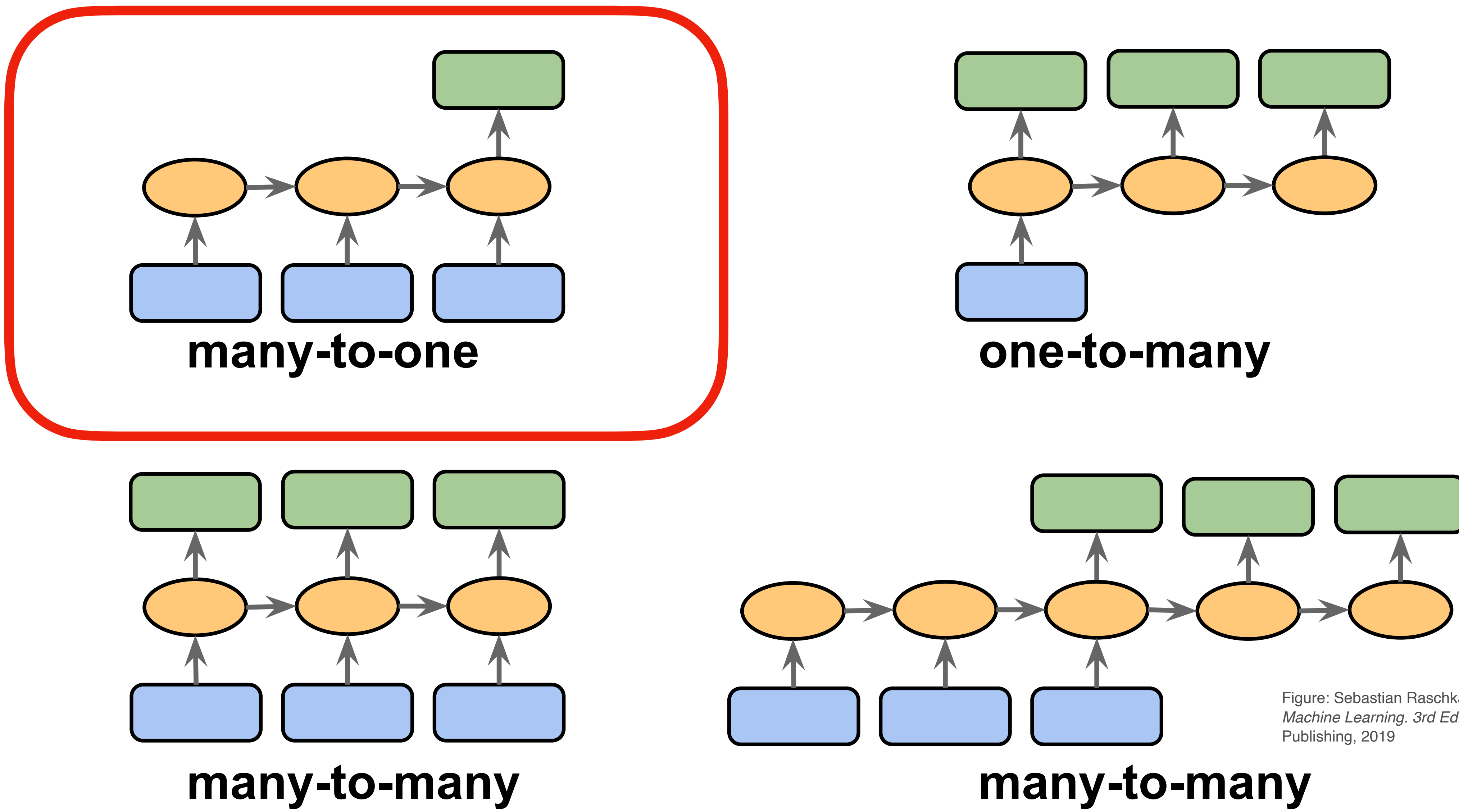
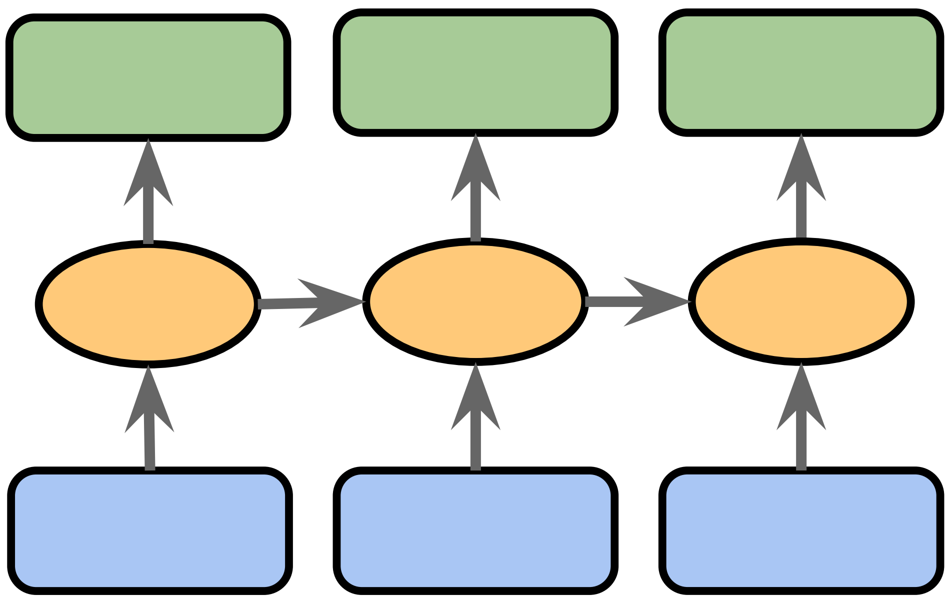


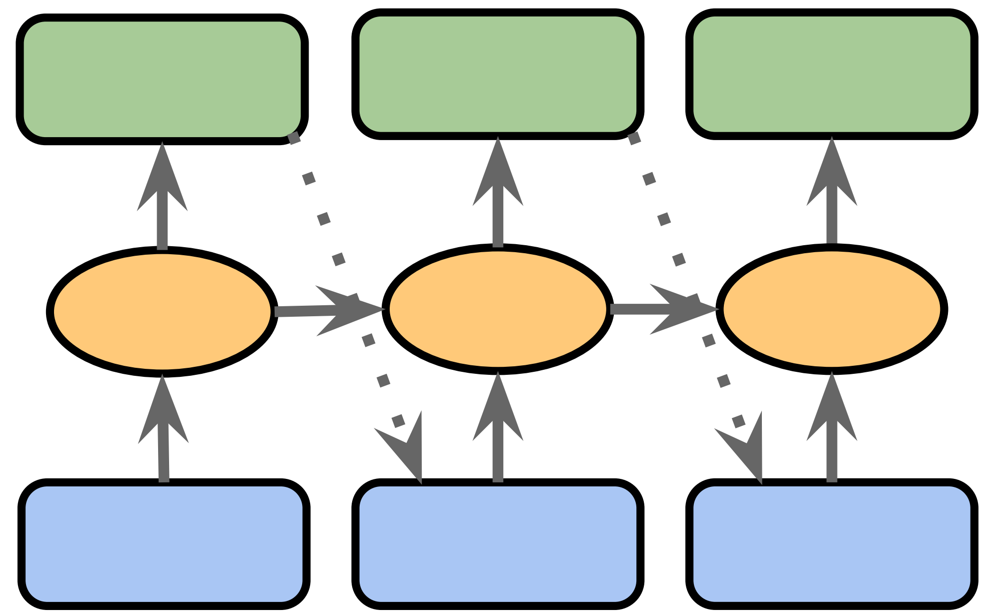
Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019

Figure based on:
The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy (http://karpathy.github.io/2015/05/21/rnn-effectiveness/)



many-to-many

"training"

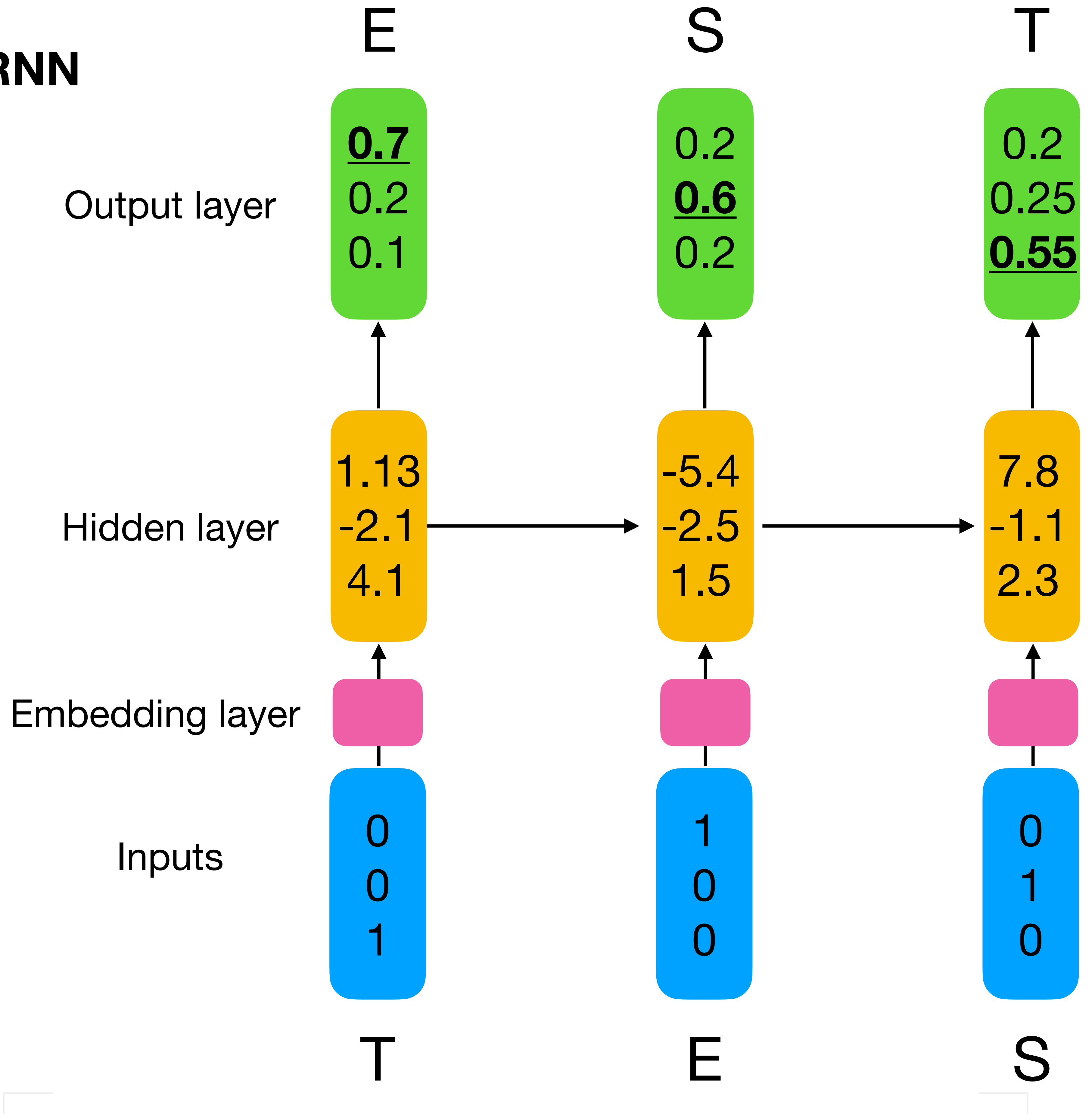


~~many~~-to-many

"one"

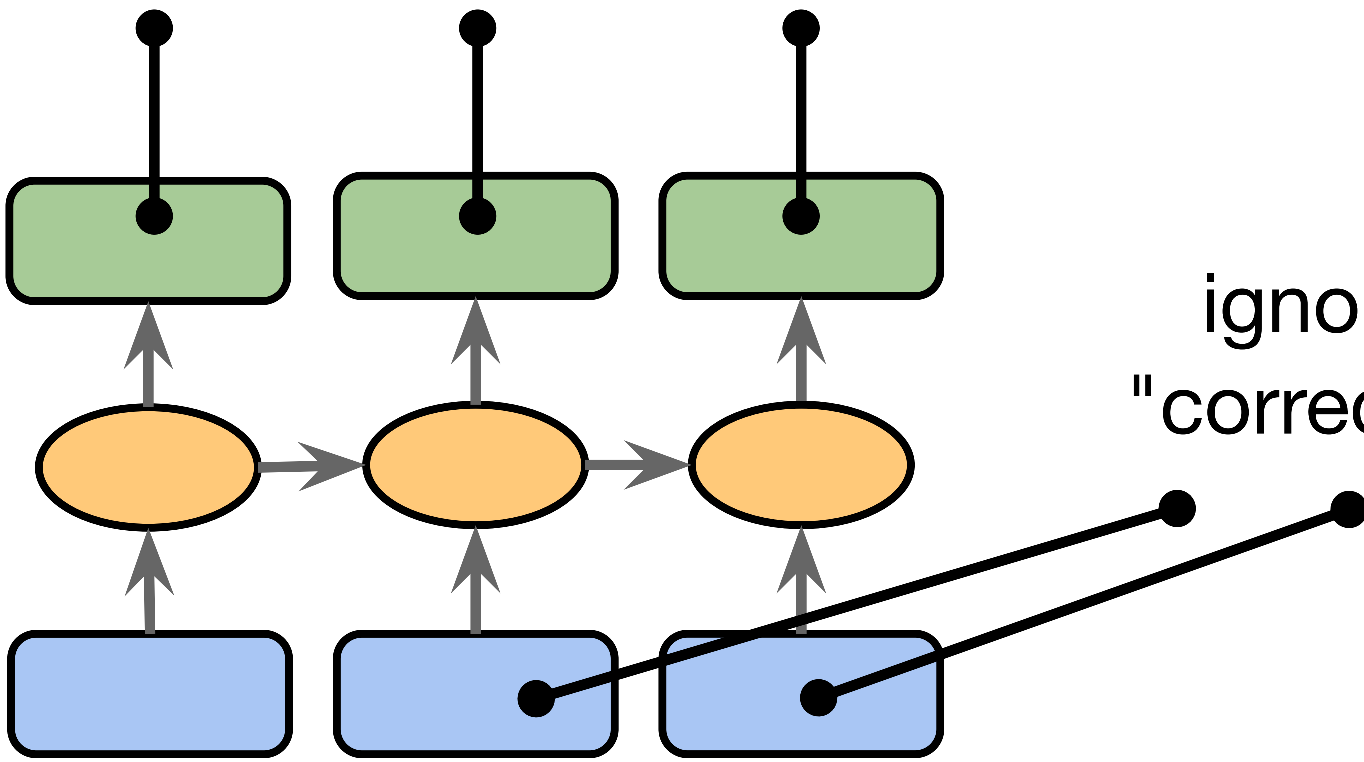
"generating new text"

Character RNN



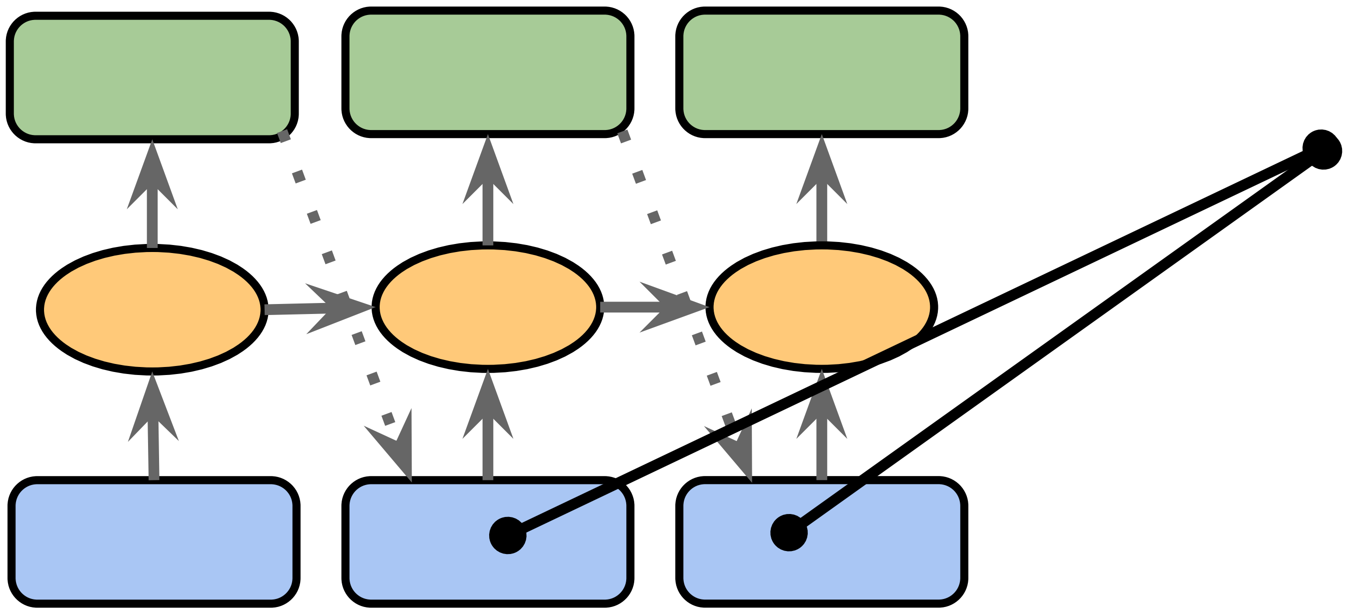
At each time step
Softmax output (probability)
for each possible"next letter"

For the next input,
ignore the prediction but use the
"correct" next letter from the dataset



many-to-many

"training"

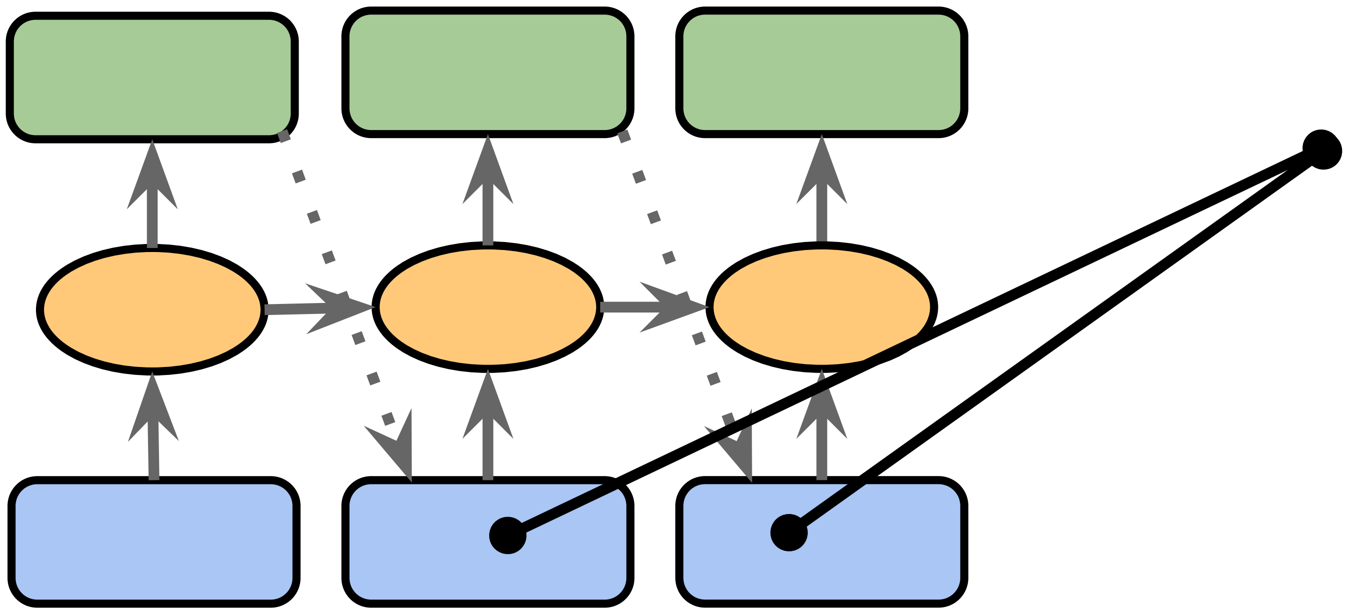


To generate new text, now, sample from the softmax outputs and provide the letter as input to the next time step

~~many~~-to-many

"one"

"generating new text"



~~many~~-to-many
"one"
"generating new text"

To generate new text, now, sample from the softmax outputs and provide the letter as input to the next time step

Note that this approach works with both Word- and Character-RNNs

Advantages and Disadvantages of Character RNNs over Word RNNs

+ Character embeddings (only 24 letters plus punctuation in English language) require less memory compared to word embeddings
+ Smaller output layers for the same reason as above
- Can create weird & nonsense words
- Worse at capturing long-distance dependencies

Implementing Character RNNs in PyTorch

1. Sequence Generation with RNNs
2. **Character RNN in PyTorch**
3. RNNs with Attention
4. Attention is All We Need
5. Transformer Models
6. Transformer in PyTorch

LSTM Class

https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a *stacked LSTM*, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional LSTM. Default: `False`
- **proj_size** – If `> 0`, will use LSTM with projections of corresponding size. Default: 0

Examples:

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```

LSTM Class

Examples:

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```

(number of layers)

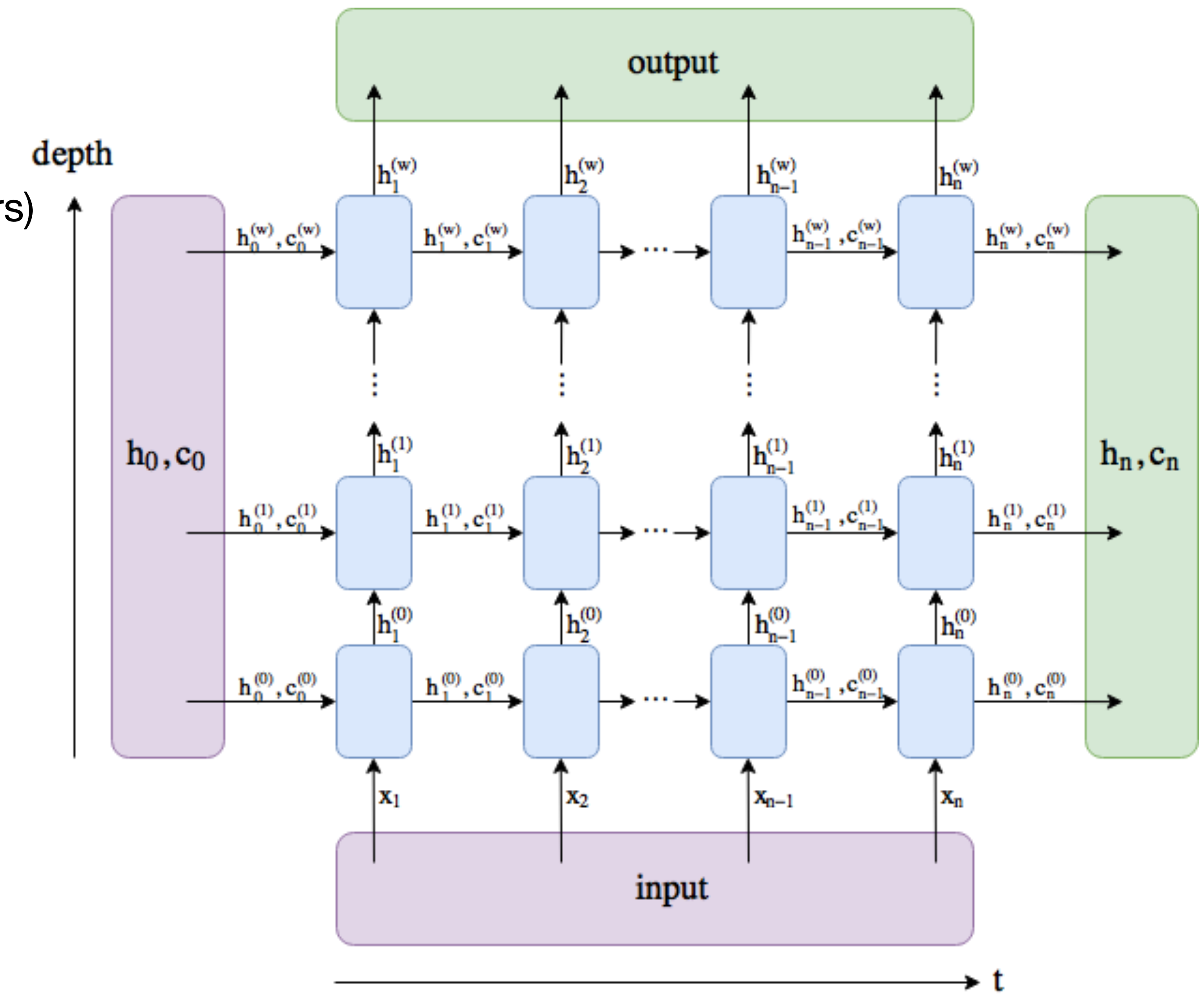


Image source: https://stackoverflow.com/questions/48302810/whats-the-difference-between-hidden-and-output-in-pytorch-lstm

LSTMCell Class

https://pytorch.org/docs/stable/generated/torch.nn.LSTMCell.html

Inputs: input, (h_0, c_0)

- **input** of shape (*batch, input_size*): tensor containing input features
- **h_0** of shape (*batch, hidden_size*): tensor containing the initial hidden state for each element in the batch.
- **c_0** of shape (*batch, hidden_size*): tensor containing the initial cell state for each element in the batch.

  If (*h_0, c_0*) is not provided, both **h_0** and **c_0** default to zero.

Outputs: (h_1, c_1)

- **h_1** of shape (*batch, hidden_size*): tensor containing the next hidden state for each element in the batch
- **c_1** of shape (*batch, hidden_size*): tensor containing the next cell state for each element in the batch

Examples:

```
>>> rnn = nn.LSTMCell(10, 20) # (input_size, hidden_size)
>>> input = torch.randn(2, 3, 10) # (time_steps, batch, input_size)
>>> hx = torch.randn(3, 20) # (batch, hidden_size)
>>> cx = torch.randn(3, 20)
>>> output = []
>>> for i in range(input.size()[0]):
        hx, cx = rnn(input[i], (hx, cx))
        output.append(hx)
>>> output = torch.stack(output, dim=0)
```

LSTMCell Class

Examples:

```
>>> rnn = nn.LSTMCell(10, 20) # (input_size, hidden_size)
>>> input = torch.randn(2, 3, 10) # (time_steps, batch, input_size)
>>> hx = torch.randn(3, 20) # (batch, hidden_size)
>>> cx = torch.randn(3, 20)
>>> output = []
>>> for i in range(input.size()[0]):
        hx, cx = rnn(input[i], (hx, cx))
        output.append(hx)
>>> output = torch.stack(output, dim=0)
```

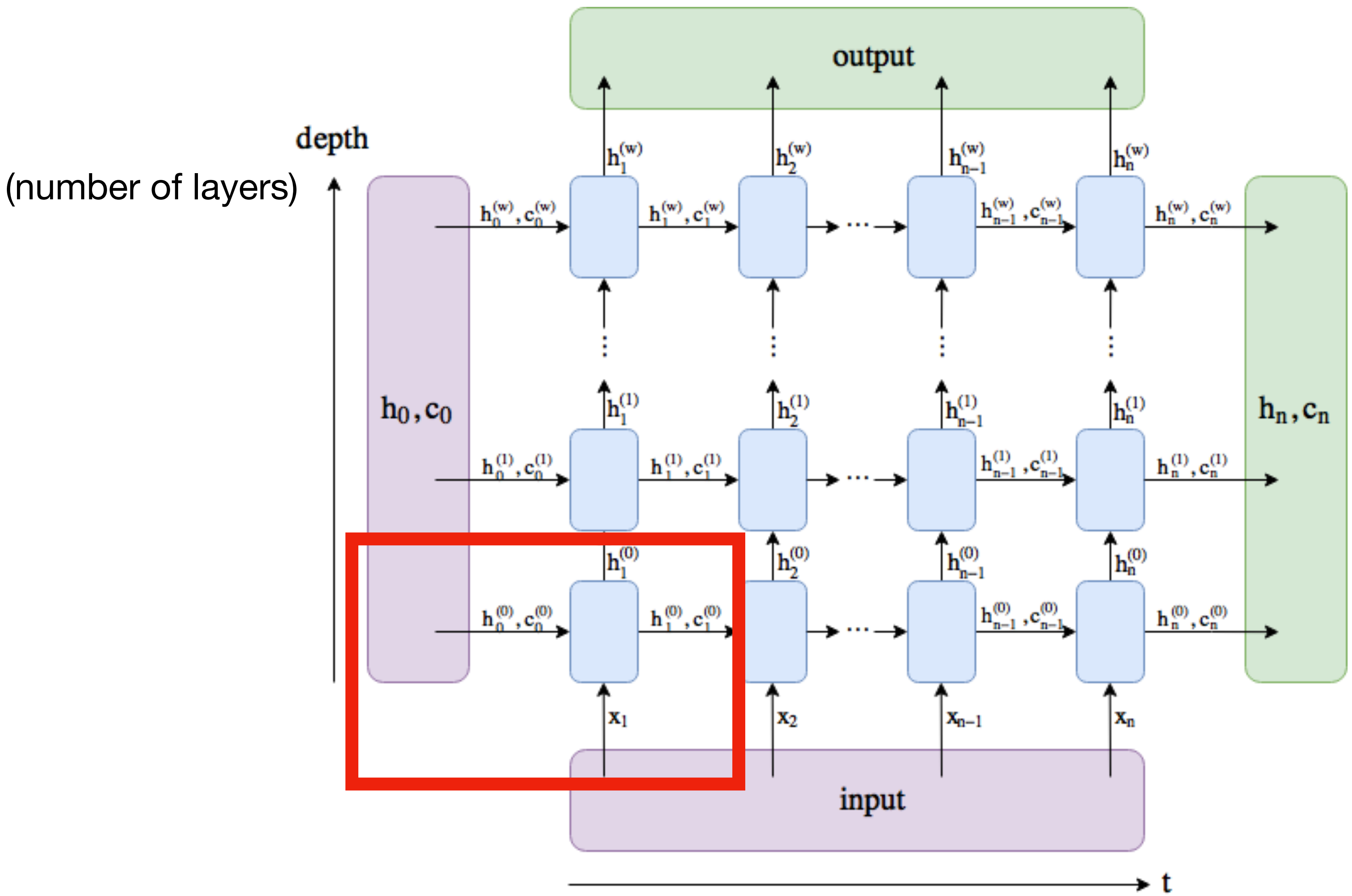
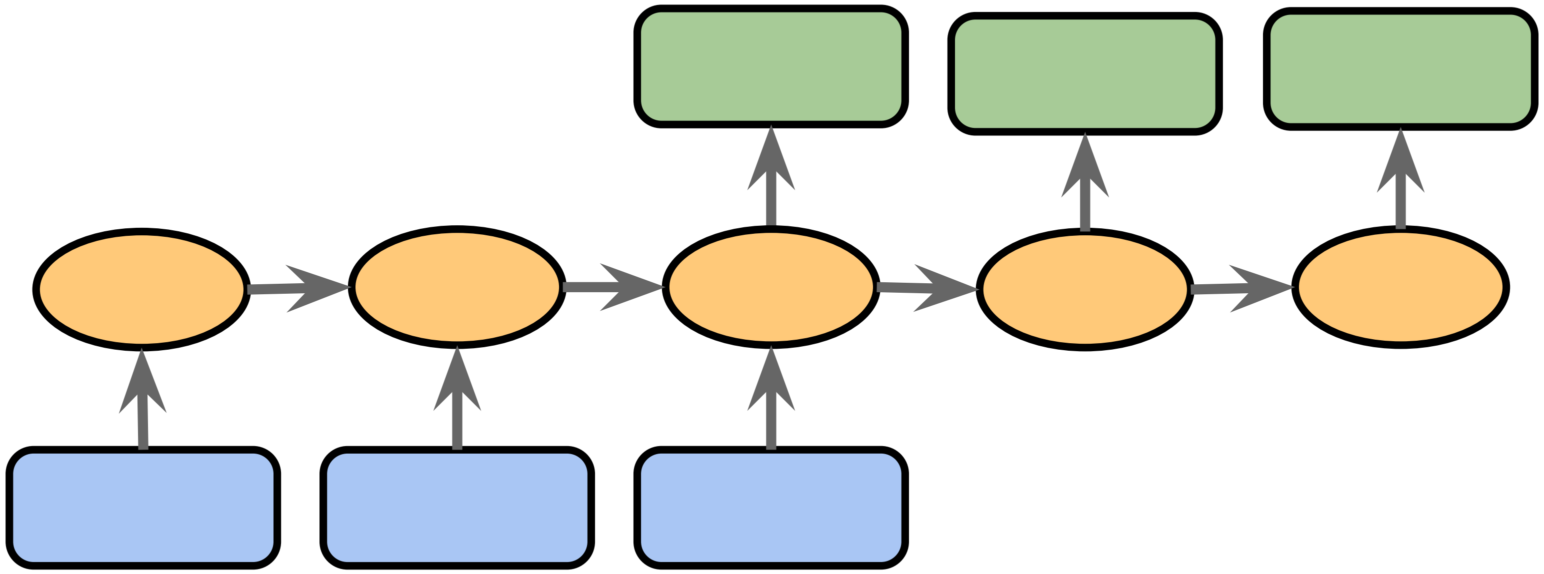


Image source: https://stackoverflow.com/questions/48302810/whats-the-difference-between-hidden-and-output-in-pytorch-lstm



many-to-many

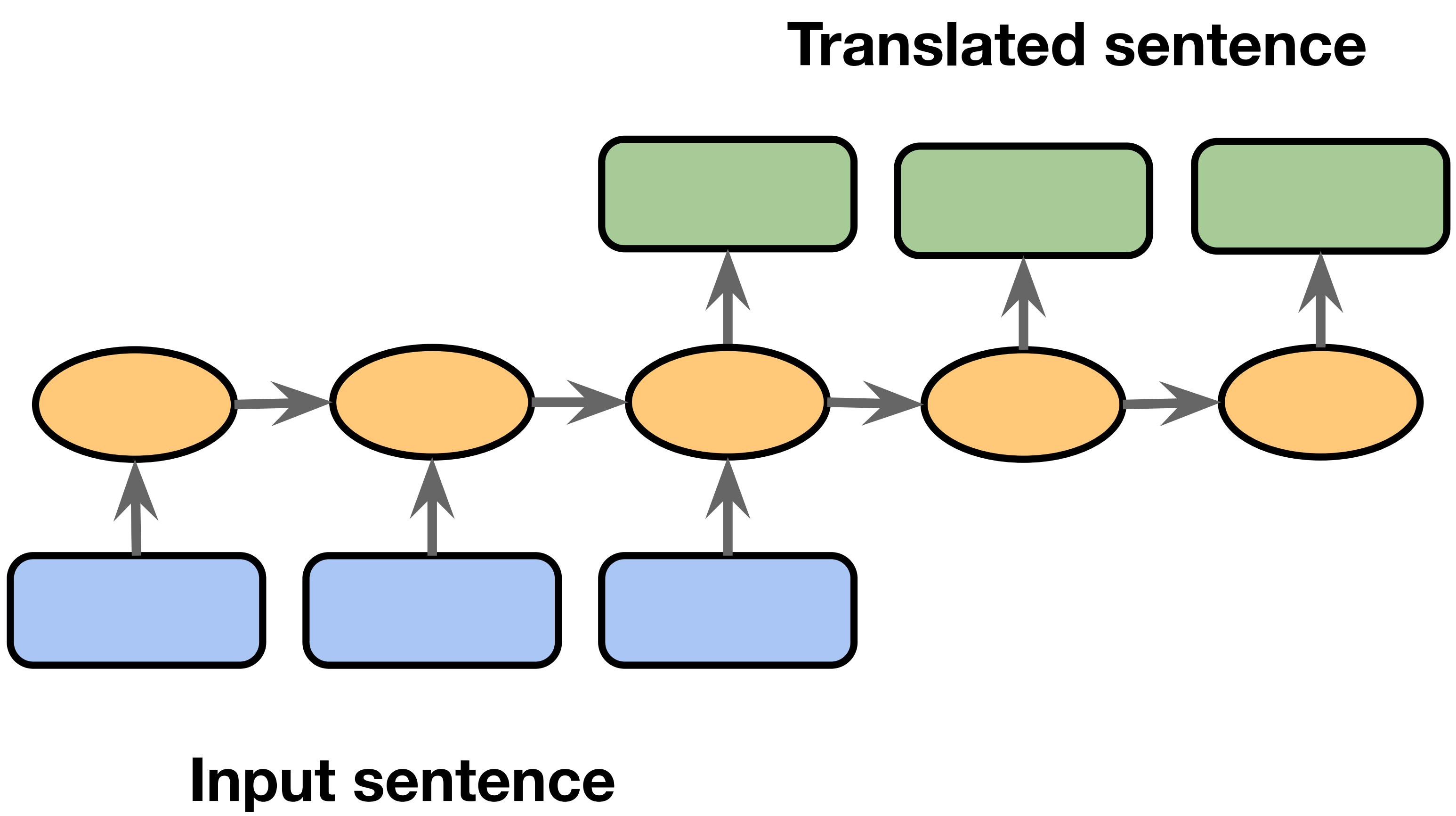
Translation with a Sequence to Sequence Network and Attention (English to French)

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Dealing Better with Long Sequences by Outfitting RNNs with an Attention Mechanism

1. Sequence Generation with RNNs
2. Character RNN in PyTorch
3. **RNNs with Attention**
4. Attention is All We Need
5. Transformer Models
6. Transformer in PyTorch

Many-to-Many Architecture for Language Translation



Translation with a Sequence to Sequence Network and Attention (English to French)

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Input: Today is a great day

Translation: Heute ist ein großartiger Tag

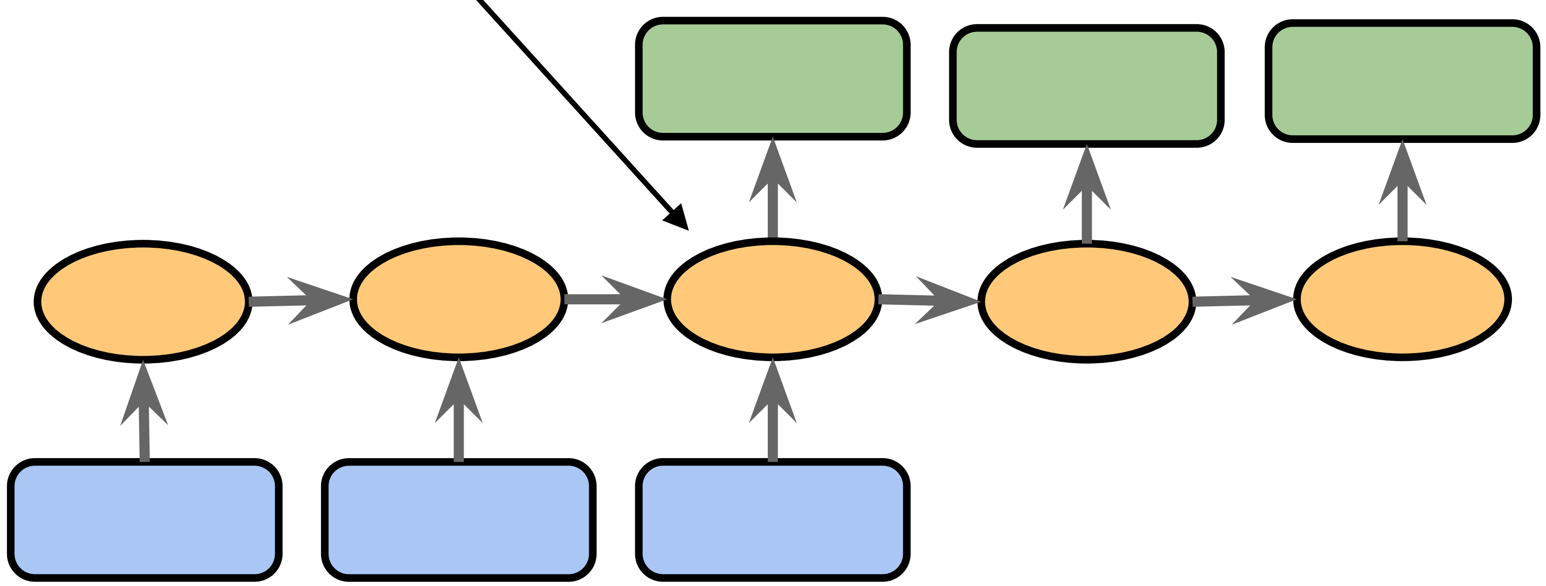
Input:

If you’ve ever studied a foreign language, you’ve probably encountered a “false friend” at some point.

Translation:

Wenn Sie jemals eine Fremdsprache gelernt haben, sind Sie wahrscheinlich irgendwann auf einen „falschen Freund“ gestoßen.

Challenge in language translation: memorize whole input sentence in one hidden state



many-to-many

Attention Mechanism

- Originally developed for language translation:
  Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. https://arxiv.org/abs/1409.0473

"... allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word ..."

"traditional" encoder+decoder RNN

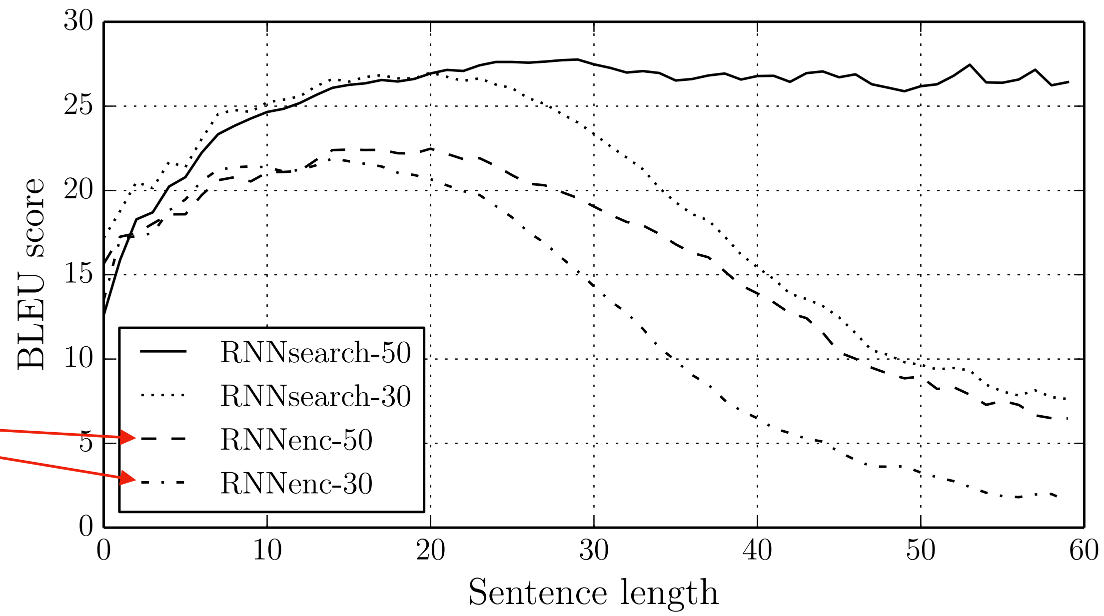


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

Attention Mechanism

Assign attention weight to each word, to know how much "attention" the model should pay to each word (i.e., for each word, the network learns a "context")

Attention Mechanism

- Originally developed for language translation:
  Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. https://arxiv.org/abs/1409.0473

Hidden state in a regular RNN (RNN #2)

1st translated word

Attention weight

Bidirectional RNN (RNN #1)

1st input word

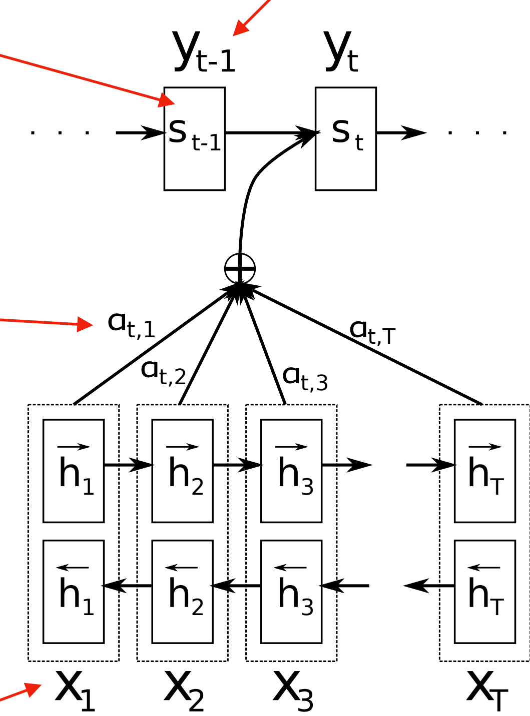
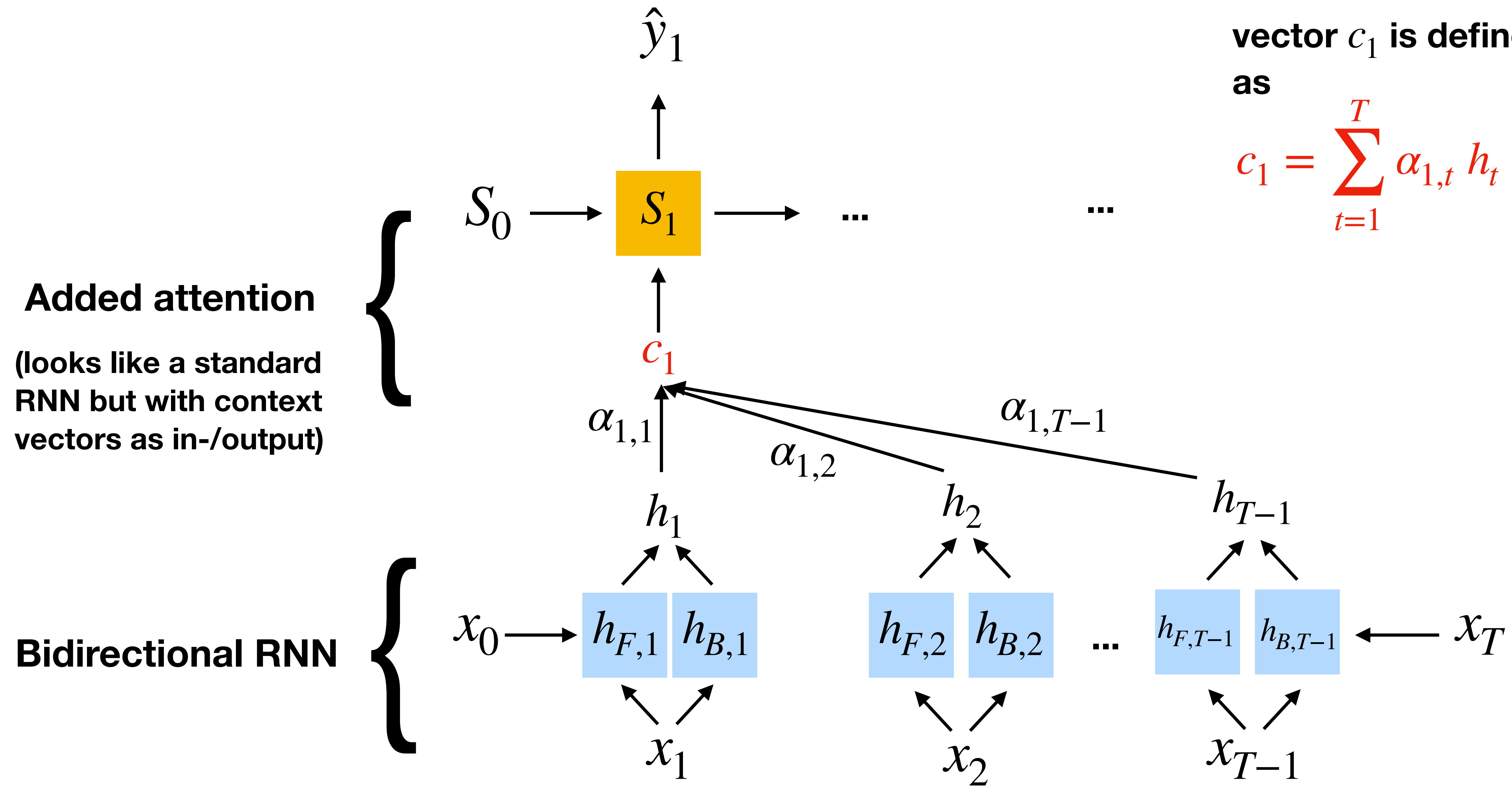


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

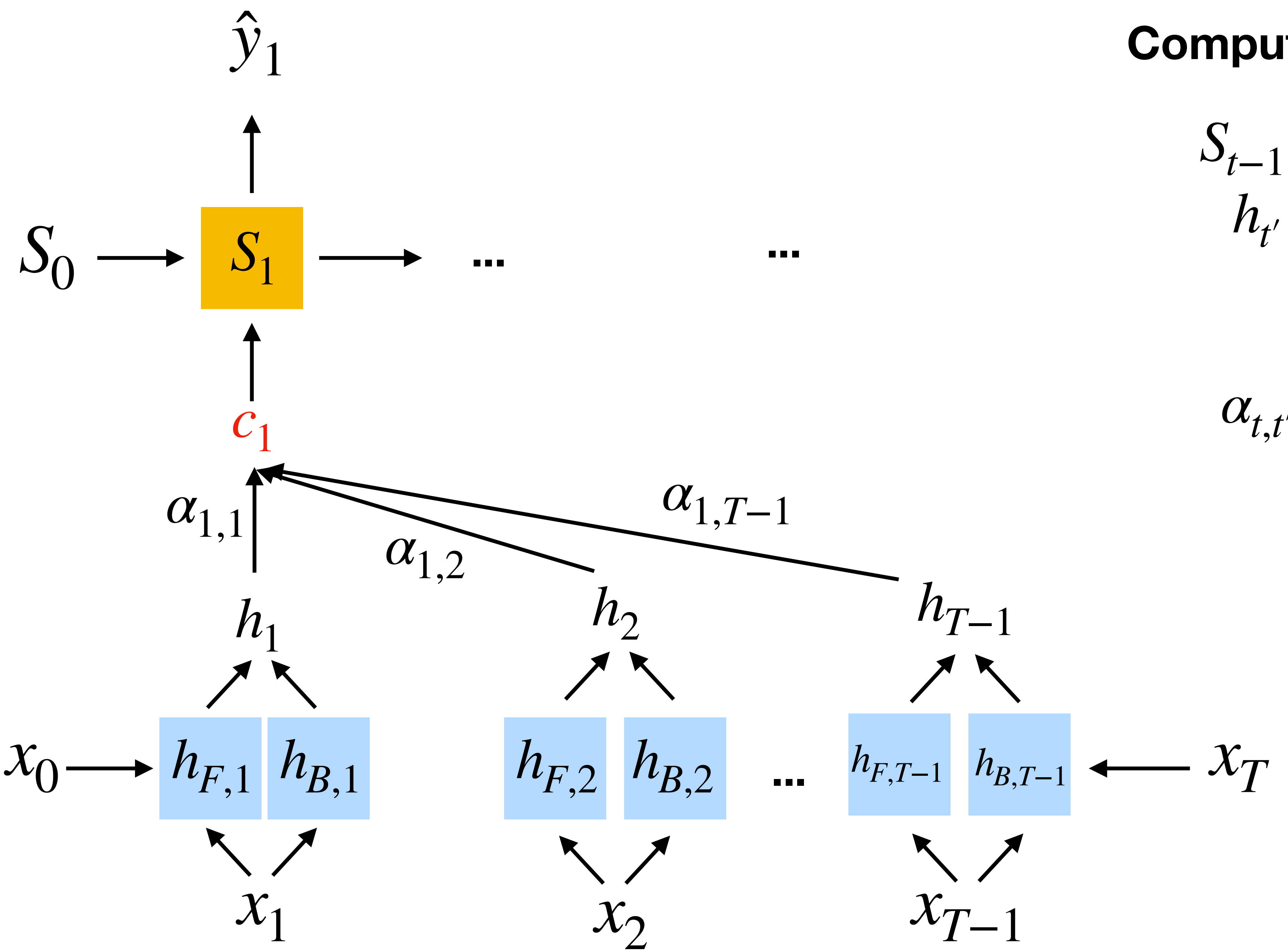
RNN Attention Mechanism



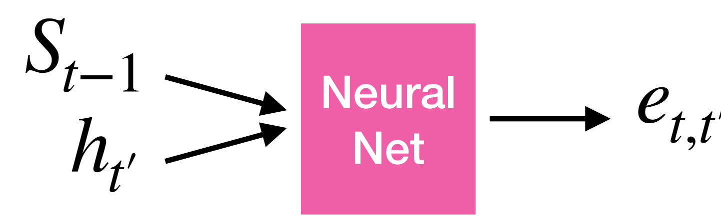
where the context vector c_1 is defined as

$$c_1 = \sum_{t=1}^{T} \alpha_{1,t}\, h_t$$

Attention Mechanism



Computing attention weights



$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{t'=1}^{T} \exp(e_{t,t'})}$$

Attention Mechanism

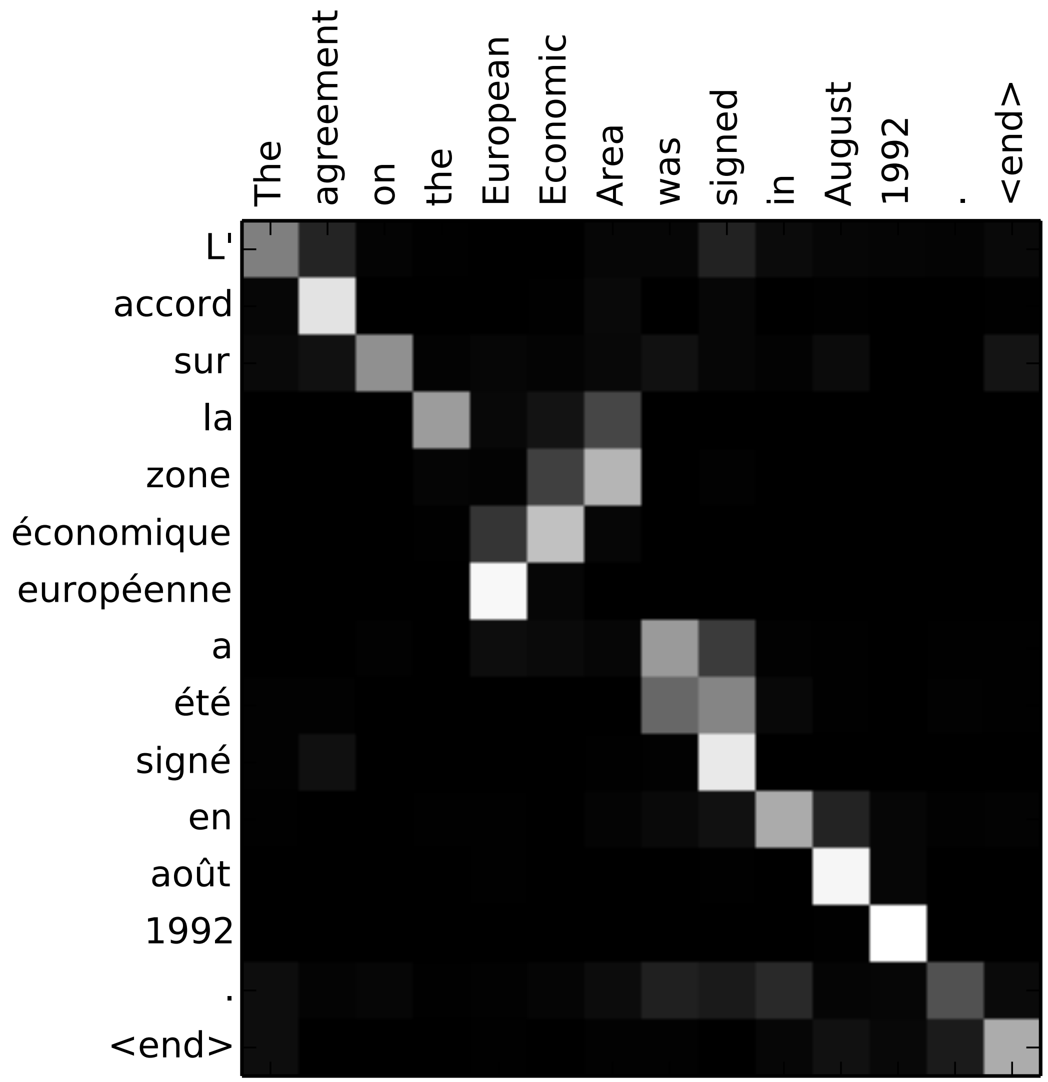
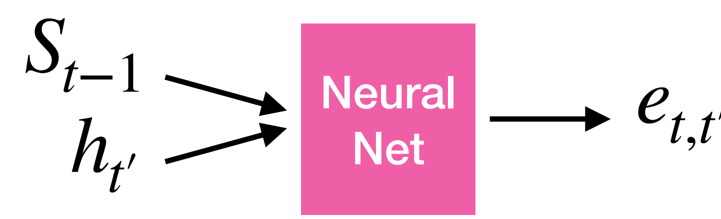


Figure: Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. https://arxiv.org/abs/1409.0473

Computing attention weights



$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{t'=1}^{T} \exp(e_{t,t'})}$$

Using Attention Without the RNN
-- Self-Attention Mechanism & Transformers

1. Sequence Generation with RNNs
2. Character RNN in PyTorch
3. RNNs with Attention
4. **Attention is All We Need**

   4.1. Basic Form of Self-Attention

   4.2. Self-Attention & Scaled Dot-Product Attention

   4.3. Multi-Head Attention
5. Transformer Models

   5.1. The Transformer Architecture

   5.2. Some Popular Transformer Models: BERT, GPT, and BART
6. Transformer in PyTorch

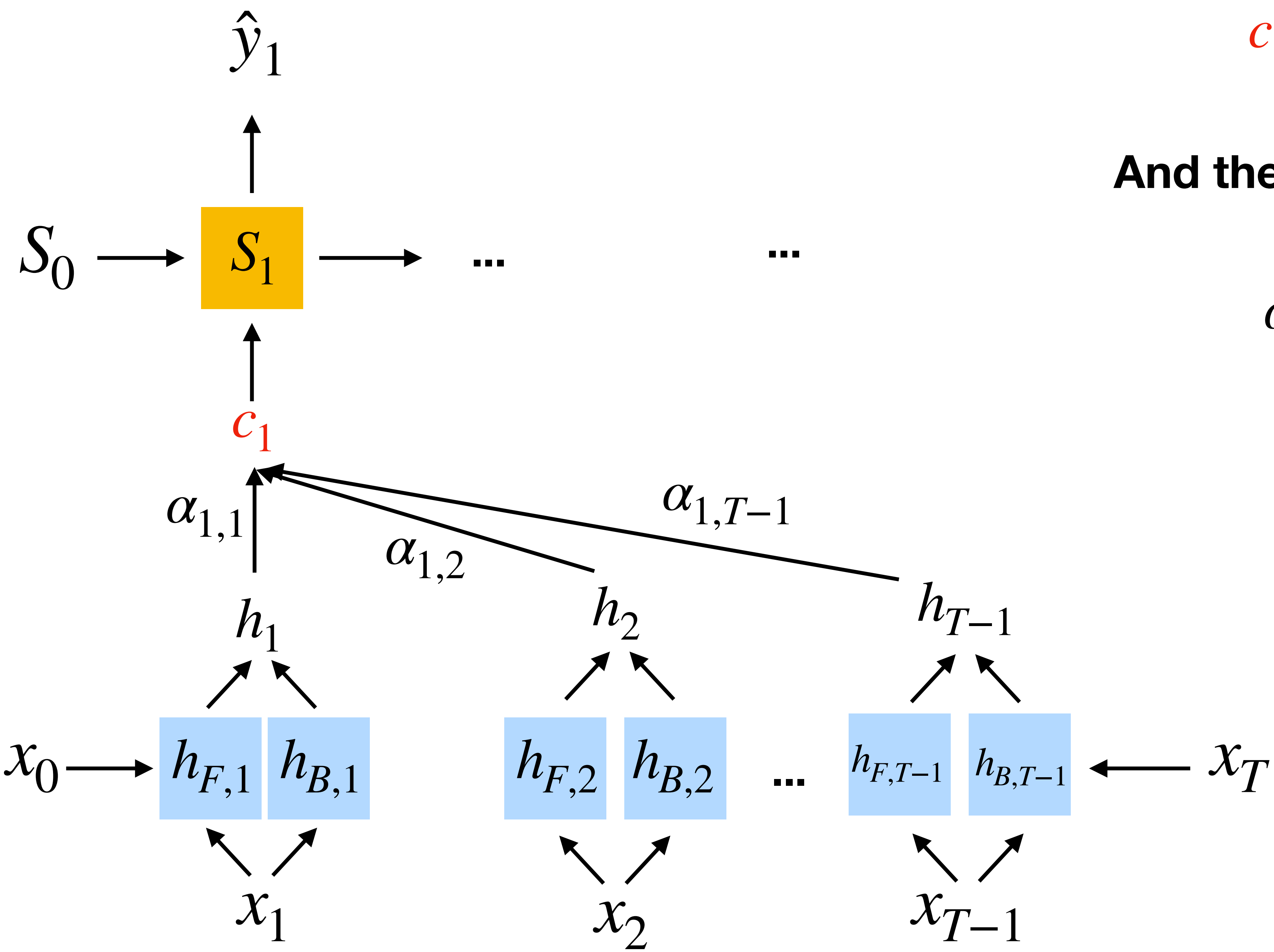
"Original" (RNN) Attention Mechanism

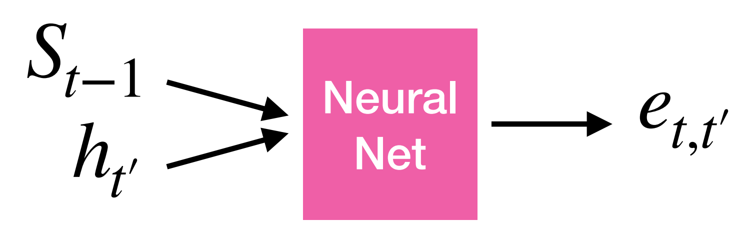
Where the context vector c_1 is defined as

$$c_1 = \sum_{t=1}^{T} \alpha_{1,t} \, h_t$$

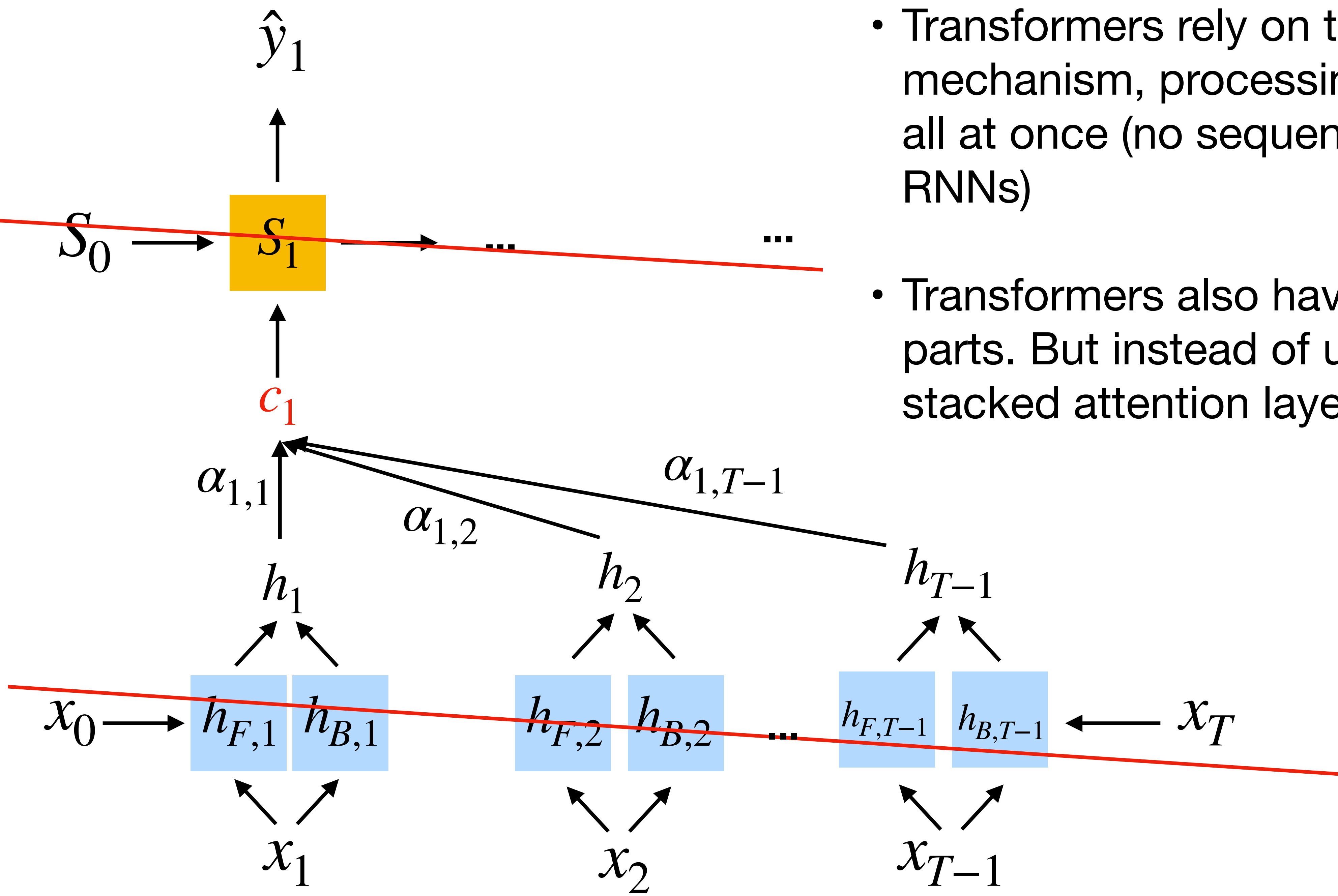
And the attention weights are

$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{t'=1}^{T} \exp(e_{t,t'})}$$





Getting rid of the sequential parts ...



- No recurrence, no convolution
- Transformers rely on the self-attention mechanism, processing the whole sequence all at once (no sequential processing like in RNNs)
- Transformers also have encoder & decoder parts. But instead of using LSTMs, they use stacked attention layers

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

https://arxiv.org/abs/1706.03762

Since ~2018, Transformers have been growing in popularity ... and size

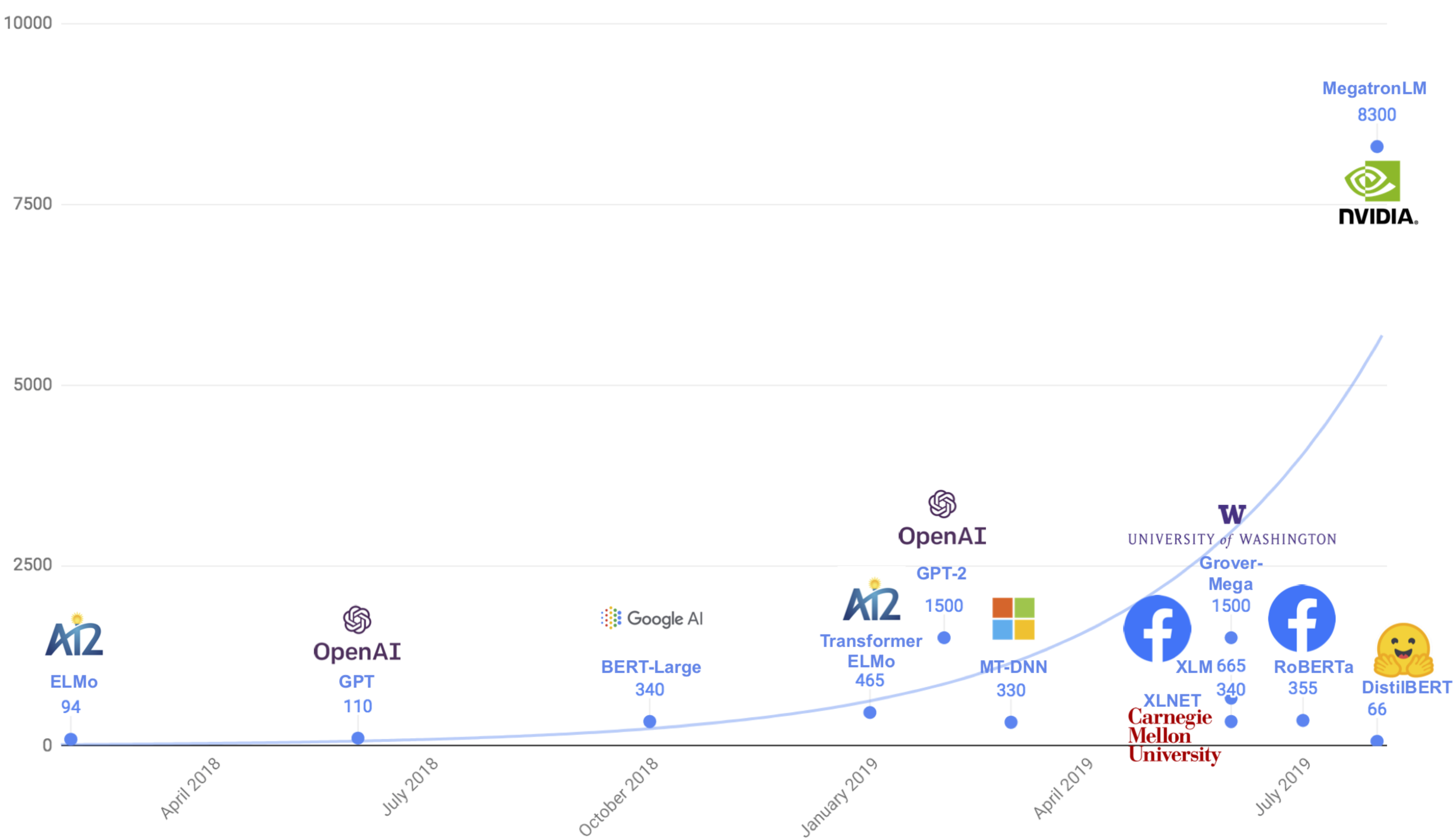


Image Source: https://medium.com/huggingface/distilbert-8cf3380435b5

Self-Attention Mechanism

-- Very Basic Form

Main procedure:

1) Derive attention <u>weights</u>: similarity between current input and all other inputs (next slide)
2) Normalize weights via softmax (next slide)
3) Compute attention value from normalized weights and corresponding inputs (below)

Self-attention as weighted sum:

$$\boldsymbol{A}_i = \sum_{j=1}^{T} a_{ij} \boldsymbol{x}_j$$

output corresponding to the i-th input

weight based on similarity between current input $\boldsymbol{x}_i$ and all other inputs

Self-Attention Mechanism

-- Very Basic Form

Self-attention as weighted sum:

$$A_i = \sum_{j=0}^{T} a_{ij} \boldsymbol{x}_j$$

output corresponding to the i-th input

weight based on similarity between current input $\boldsymbol{x}_i$ and all other inputs

How to compute the attention weights?

here as simple dot product:

$$e_{ij} = \boldsymbol{x}_i^\top \boldsymbol{x}_j$$

repeat this for all inputs $j \in \{1...T\}$, then normalize

$$a_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{j=1}^{T} \exp\left(e_{ij}\right)} = \operatorname{softmax}\left(\left[e_{ij}\right]_{j=1,...,T}\right)$$

Self-Attention Mechanism
-- Very Basic Form

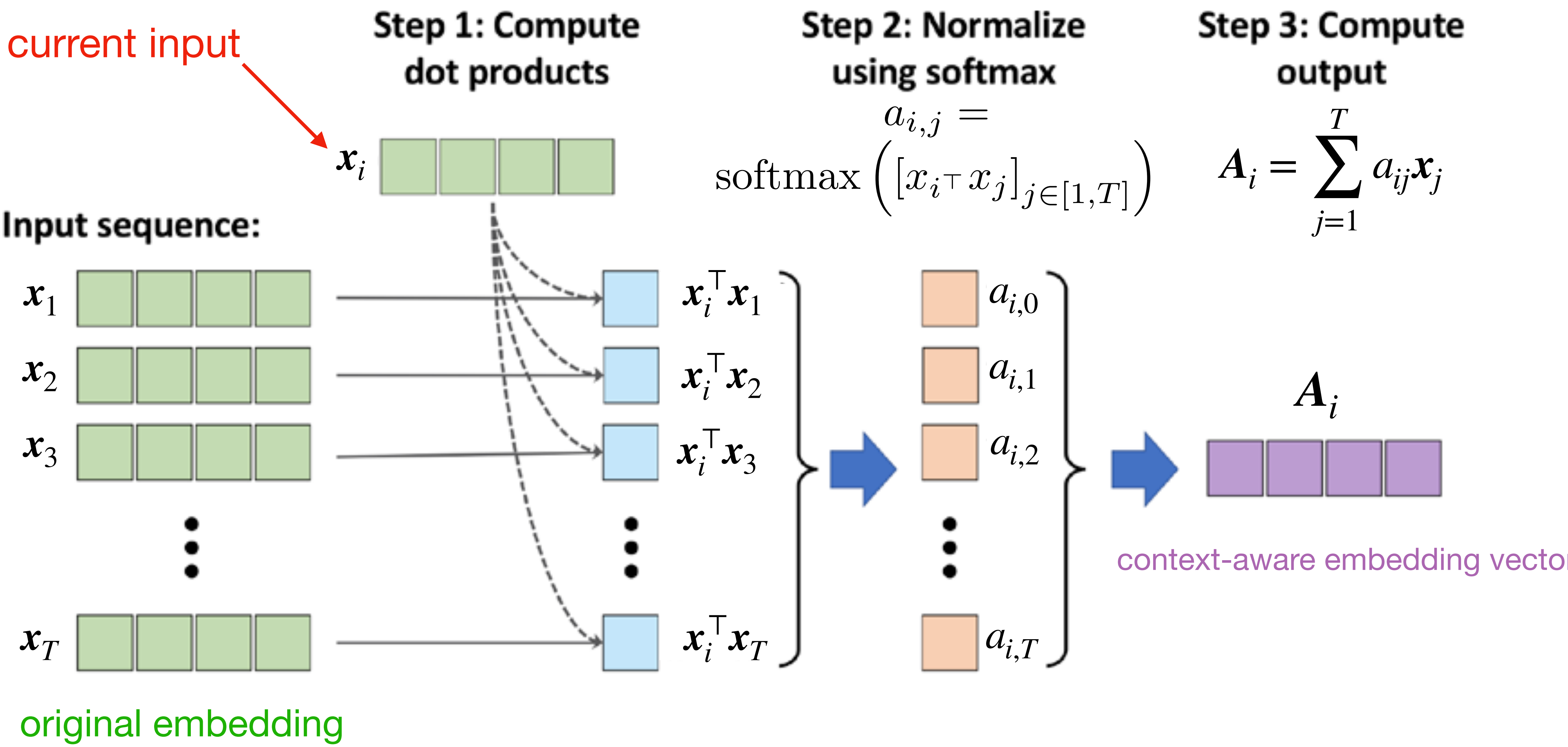


Image source: Raschka & Mirjalili 2019. Python Machine Learning, 3rd edition

Using Attention Without the RNN
-- Self-Attention Mechanism & Transformers

Self-Attention Mechanism
-- Very Basic Form

self-attention: relating different positions within a single sequence
(vs. between in- and output sequences)

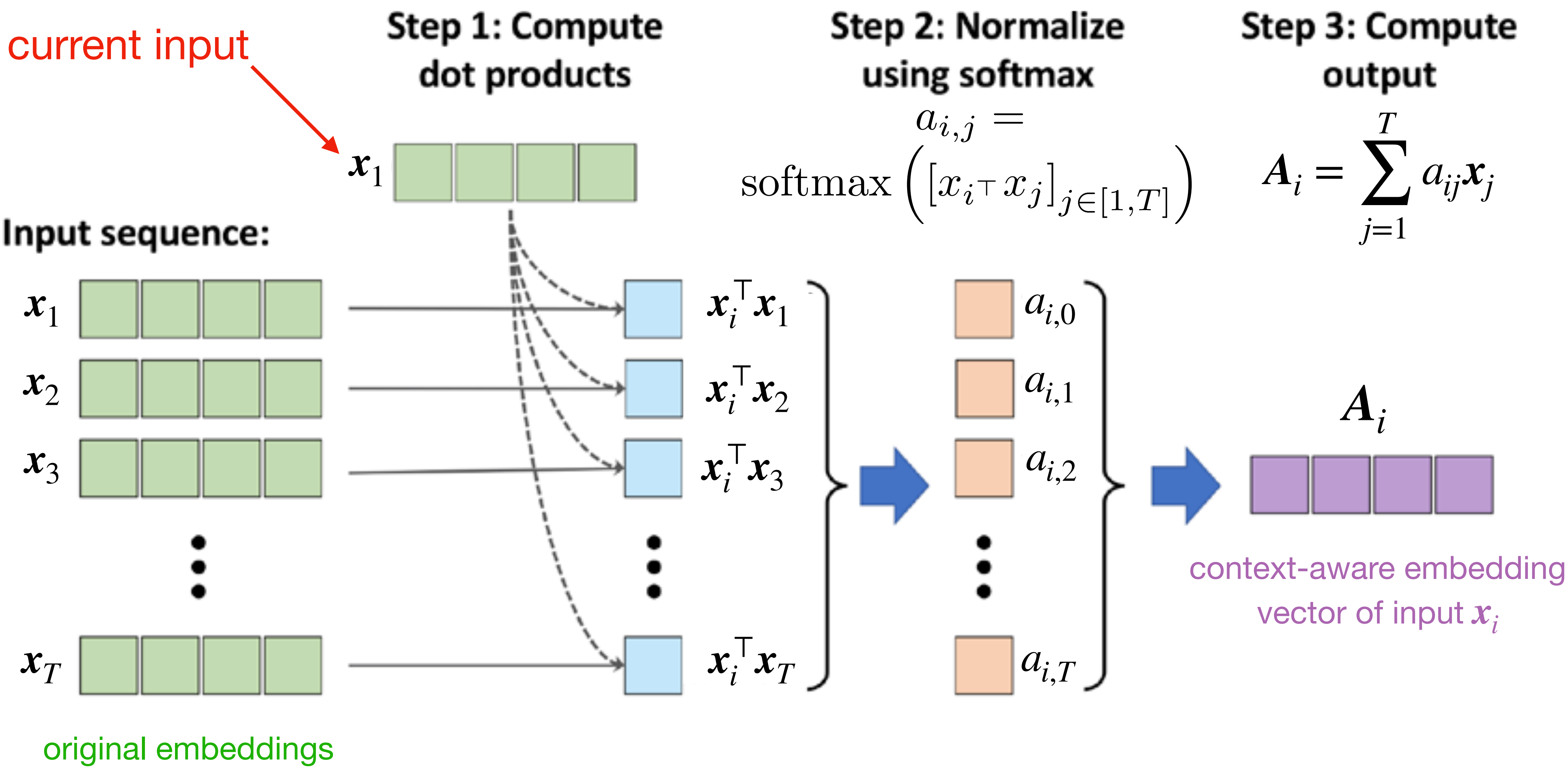


Image source: Raschka & Mirjalili 2019. Python Machine Learning, 3rd edition

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

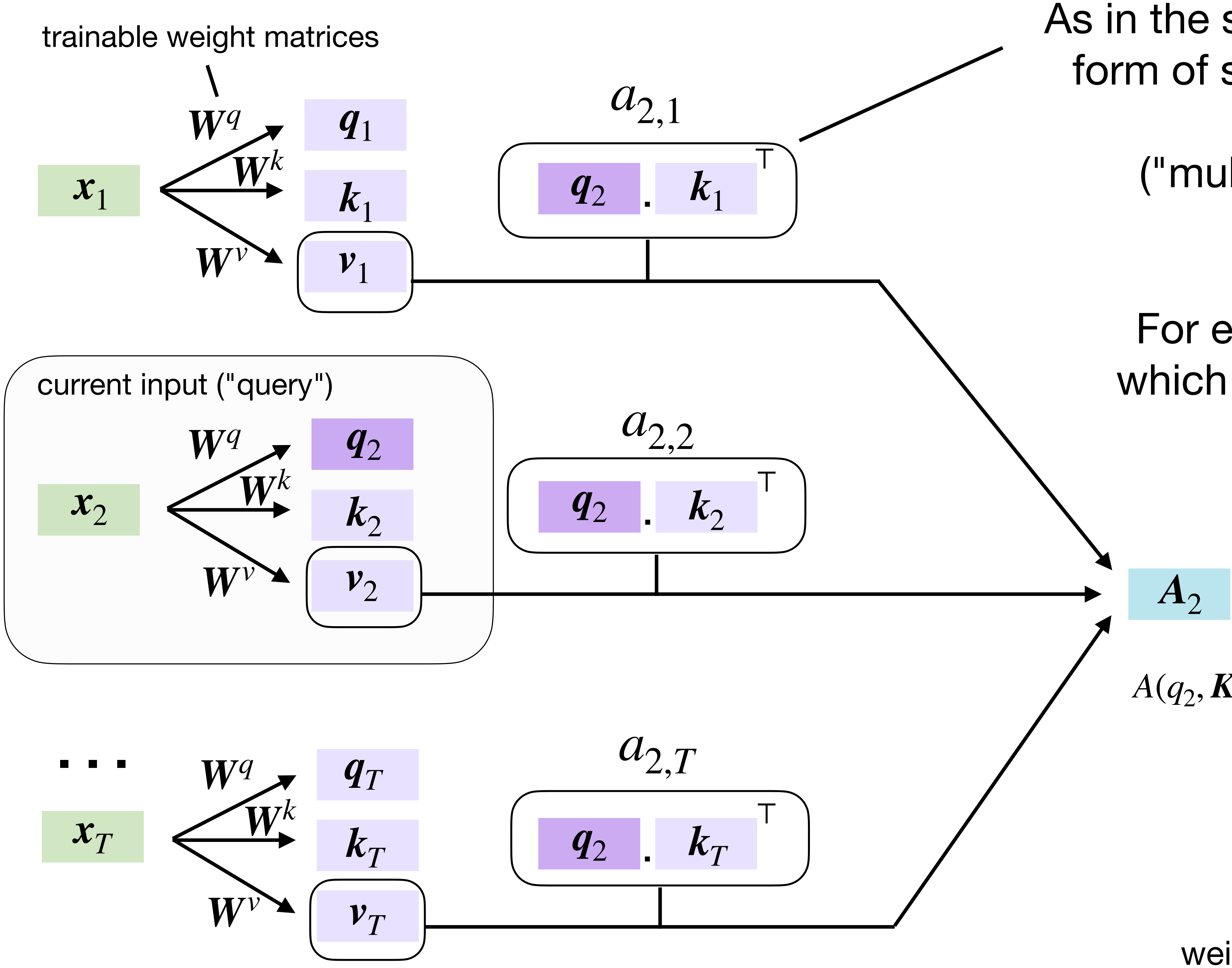
https://arxiv.org/abs/1706.03762

Self-Attention Mechanism

- Previous basic version did not involve any learnable parameters, so not very useful for learning a language model
- We are now adding 3 trainable weight matrices that are multiplied with the input sequence embeddings ($\boldsymbol{x_i}$'s)

$$\text{query} = \boldsymbol{W}^q \boldsymbol{x}_i$$
$$\text{key} = \boldsymbol{W}^k \boldsymbol{x}_i$$
$$\text{value} = \boldsymbol{W}^v \boldsymbol{x}_i$$

Self-Attention Mechanism



As in the simplified version, this is a form of similarity or compatibility measure ("multiplicative attention")

For each query, model learns which **k**ey-**v**alue input it should attend to

$$A(q_2, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{T} \Big[\underbrace{\frac{\exp\left(q_2 \cdot k_i^\top\right)}{\sum_j \exp\left(q_2 \cdot k_j^\top\right)}}_{\text{softmax}} \times \boldsymbol{v}_i \Big]$$

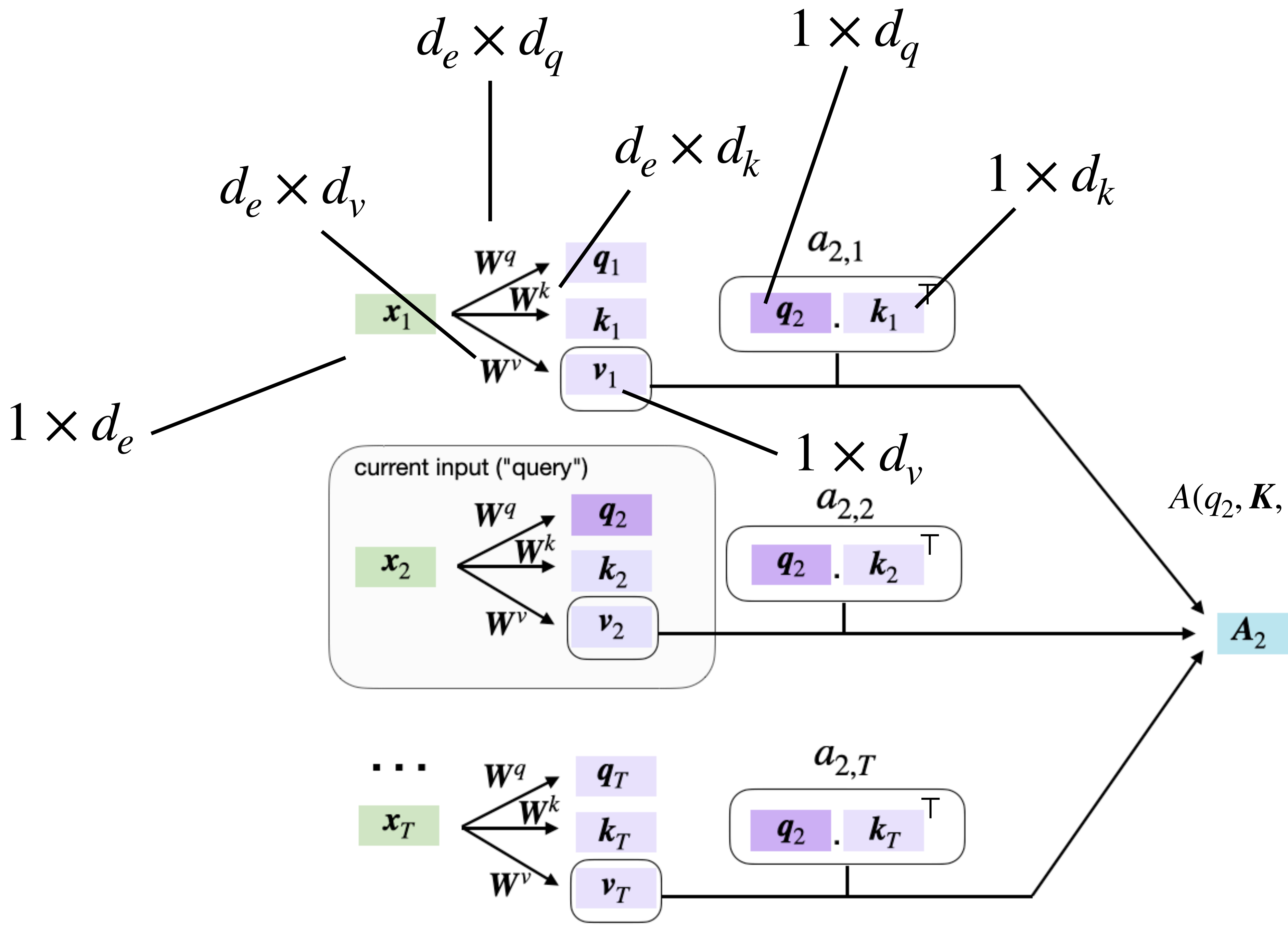
weighted sum: values weighted by attention weight (softmax score)

Self-Attention Mechanism

d_e = embedding size (original transformer = 512)

where $d_q = d_k$

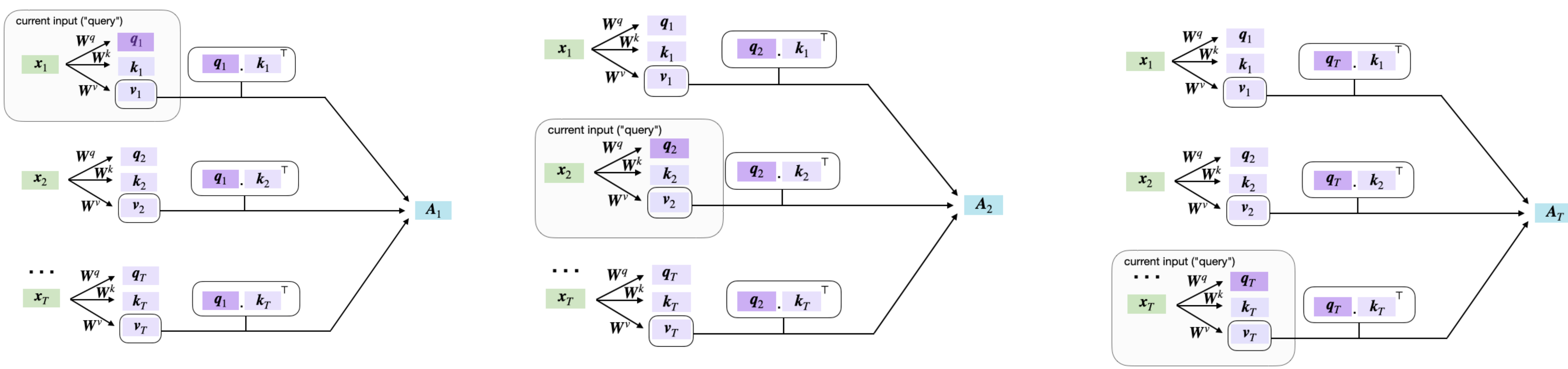
In original transformer, $d_q = d_v$ as well



$$A(q_2, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{T} \Big[\underbrace{\frac{\exp(q_2 \cdot k_i^\top)}{\sum_j \exp(q_2 \cdot k_j^\top)}}_{\substack{\text{softmax} \\ 1 \times 1}} \times \boldsymbol{v}_i \Big]$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{1 \times d_v}$$

Self-Attention Mechanism



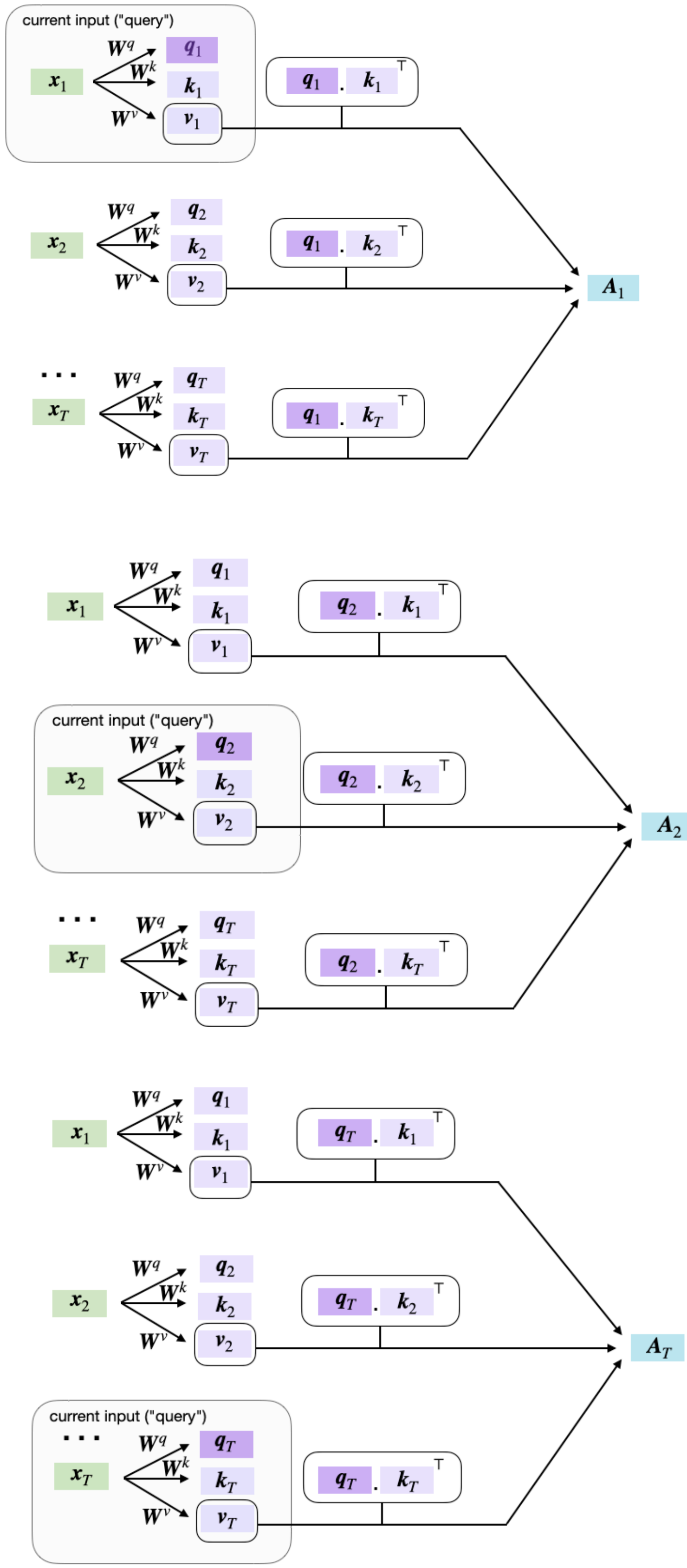
Attention score matrix: $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \\ \boldsymbol{A}_3 \end{bmatrix}$

Self-Attention Mechanism (Scaled Dot Product Attention)

$d_e =$ embedding size

$T =$ input sequence size

$x \in \mathbb{R}^{T \times d_e}$



$Q \in \mathbb{R}^{T \times d_q}$

$K \in \mathbb{R}^{T \times d_k}$

$V \in \mathbb{R}^{T \times d_v}$

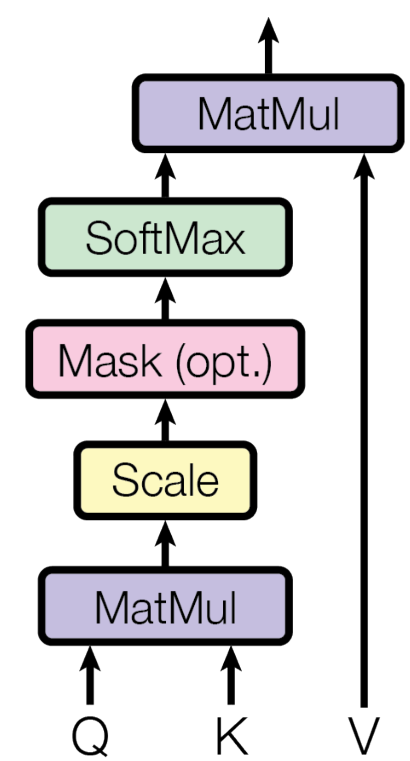
$$A(Q, K, V) = \underbrace{\overbrace{\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)}^{\substack{\text{"attention matrix"} \\ T \times T}} V}_{\substack{T \times d_v \\ \text{"attention-based embedding"}}}$$

Scaled Dot-Product Attention

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V$$

To ensure that the dot-products between query and and key don't grow too large (and softmax gradient become too small) for large d_k

Scaled Dot-Product Attention



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Using Attention Without the RNN
-- Self-Attention Mechanism & Transformers

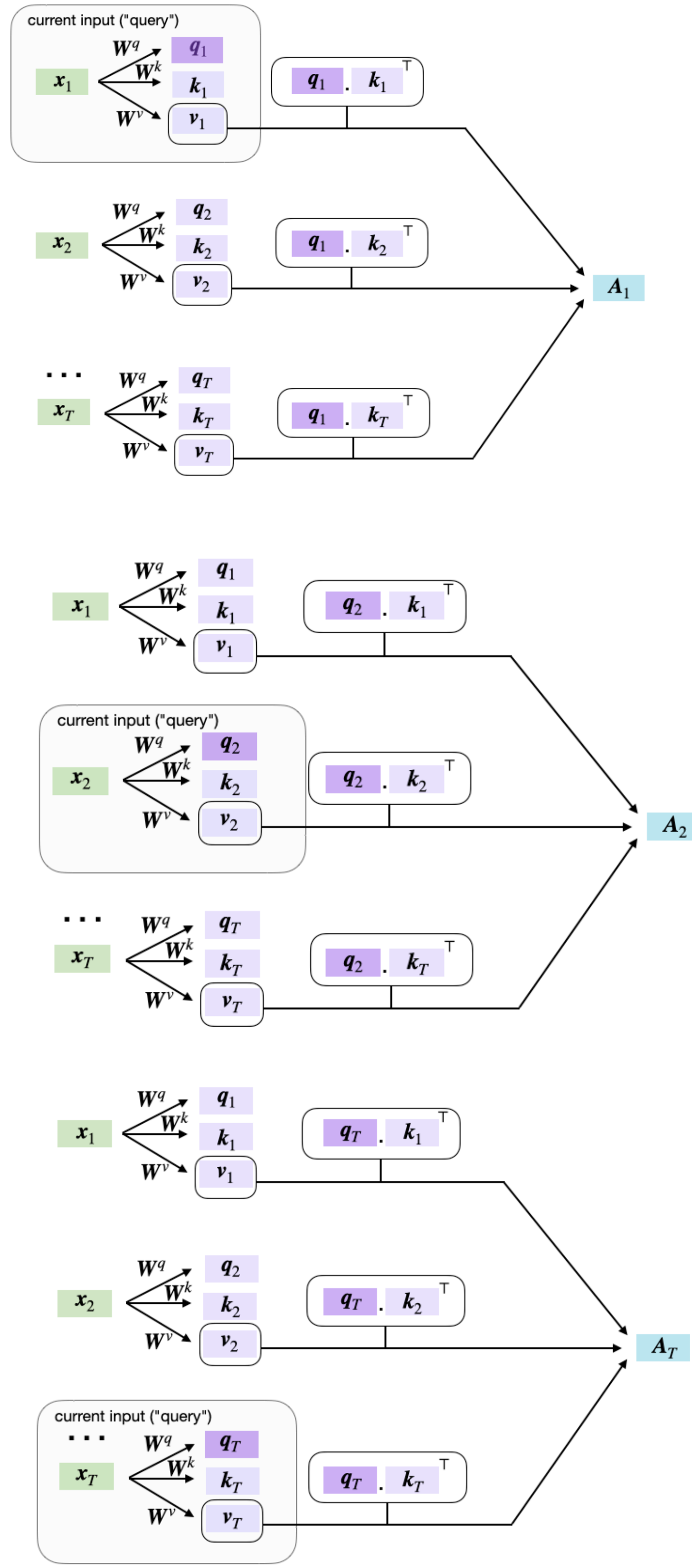
1. Sequence Generation with RNNs
2. Character RNN in PyTorch
3. RNNs with Attention
4. **Attention is All We Need**
   4.1. Basic Form of Self-Attention
   4.2. Self-Attention & Scaled Dot-Product Attention
   4.3. Multi-Head Attention
5. Transformer Models
   5.1. The Transformer Architecture
   5.2. Some Popular Transformer Models: BERT, GPT, and BART
6. Transformer in PyTorch

Self-Attention Mechanism (Scaled Dot Product Attention)

d_e = embedding size

T = input sequence size

$\boldsymbol{x} \in \mathbb{R}^{T \times d_e}$



$\boldsymbol{Q} \in \mathbb{R}^{T \times d_q}$

$\boldsymbol{K} \in \mathbb{R}^{T \times d_k}$

$\boldsymbol{V} \in \mathbb{R}^{T \times d_v}$

"attention matrix"

$$A(Q,K,V) = \underbrace{\overbrace{\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)}^{T \times T} V}_{T \times d_v}$$

"attention-based embedding"

Multi-Head Attention

- Apply self-attention multiple times in parallel (similar to multiple kernels for channels in CNNs)
- For each head (self-attention layer), use different $\boldsymbol{W}^q, \boldsymbol{W}^k, \boldsymbol{W}^v$, then concatenate the results, $\boldsymbol{A}_{(i)}$
- 8 attention heads in the original transformer, i.e., $\boldsymbol{W}^q_{(1)}, \boldsymbol{W}^k_{(1)}, \boldsymbol{W}^v_{(1)} \dots \boldsymbol{W}^q_{(8)}, \boldsymbol{W}^k_{(8)}, \boldsymbol{W}^v_{(8)}$
- Allows attending to different parts in the sequence differently

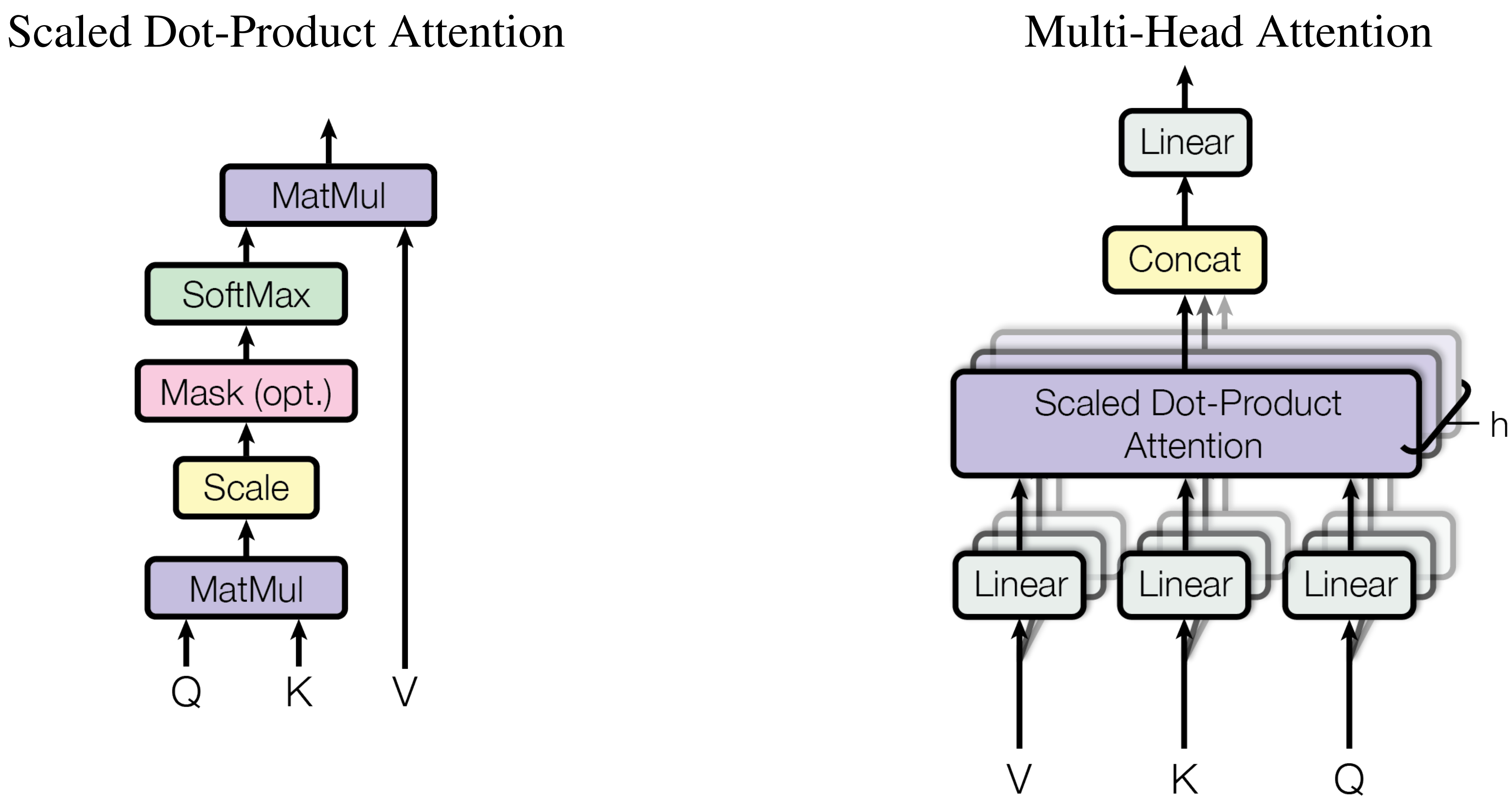
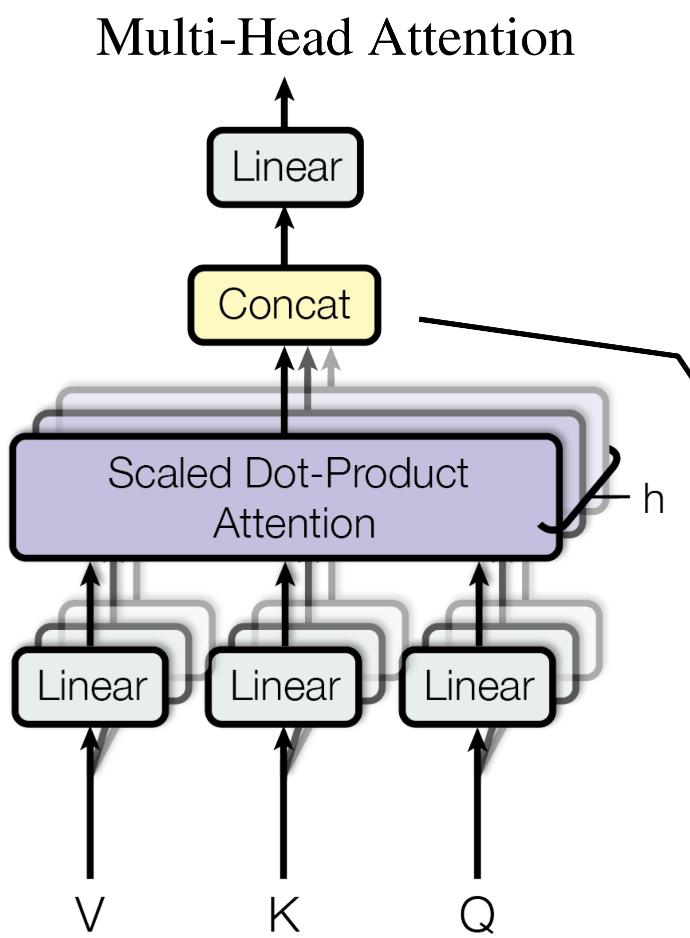
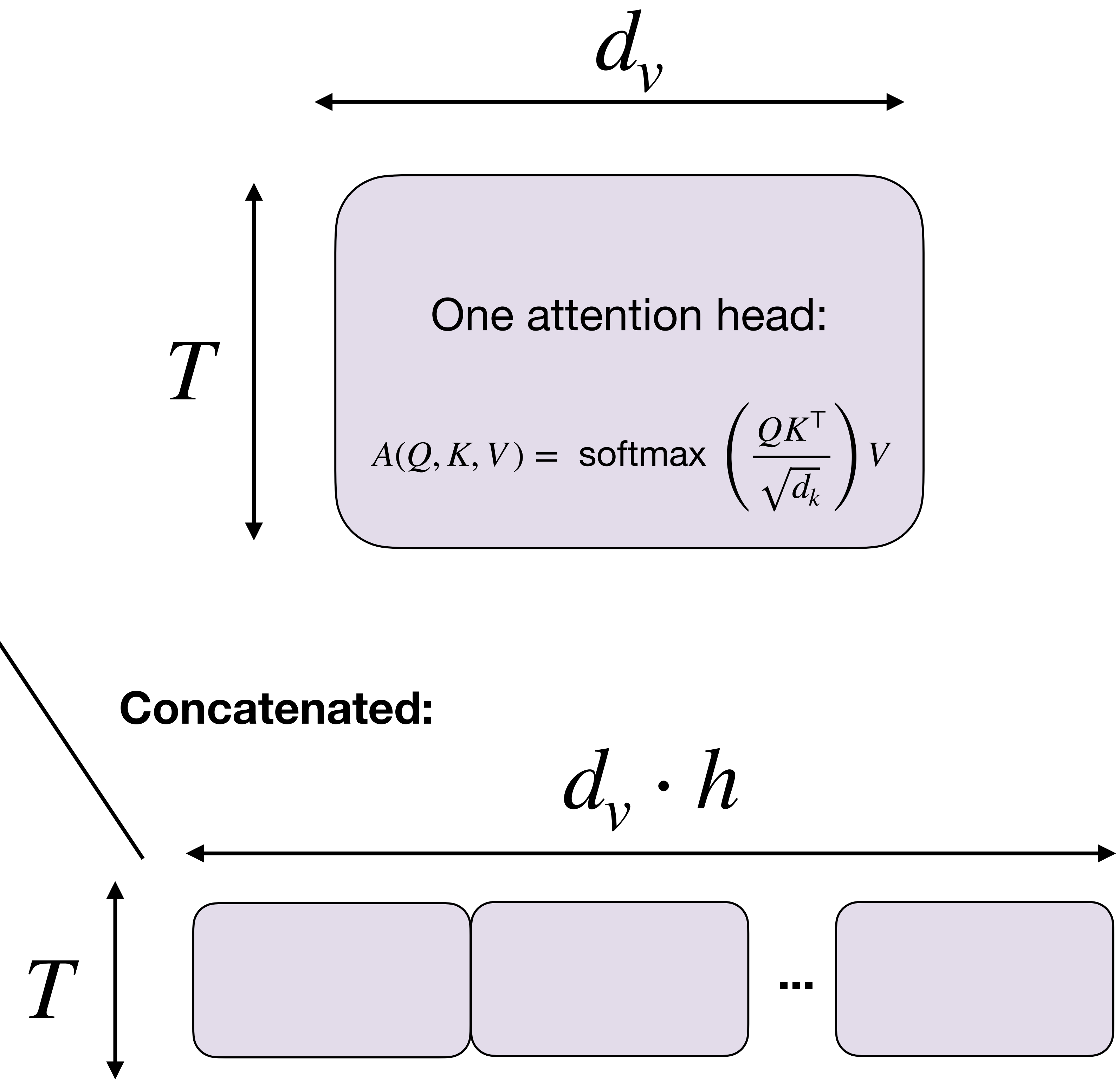


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

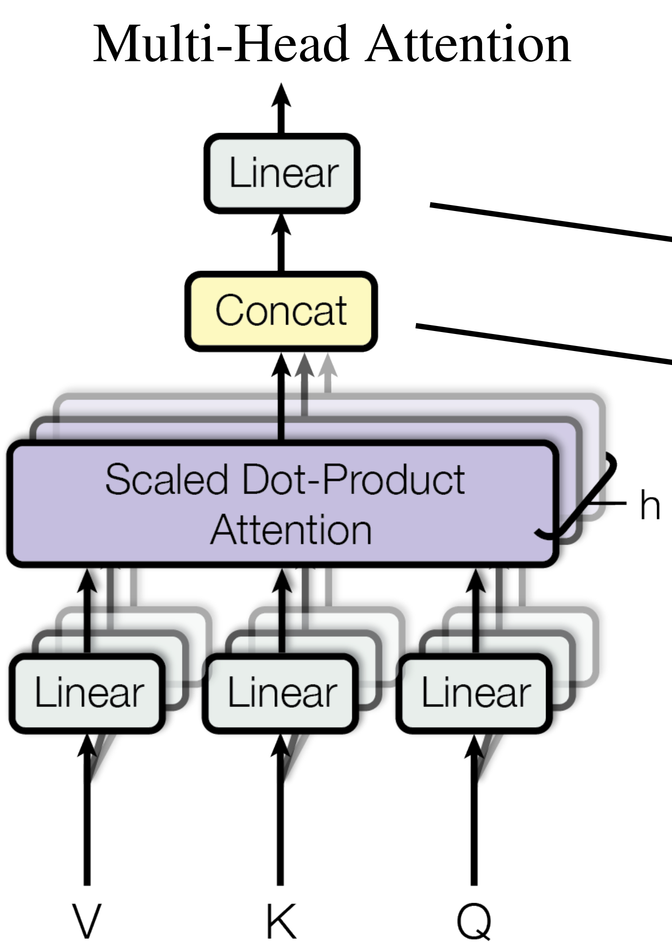


Input sequence dim.
in original transformer:

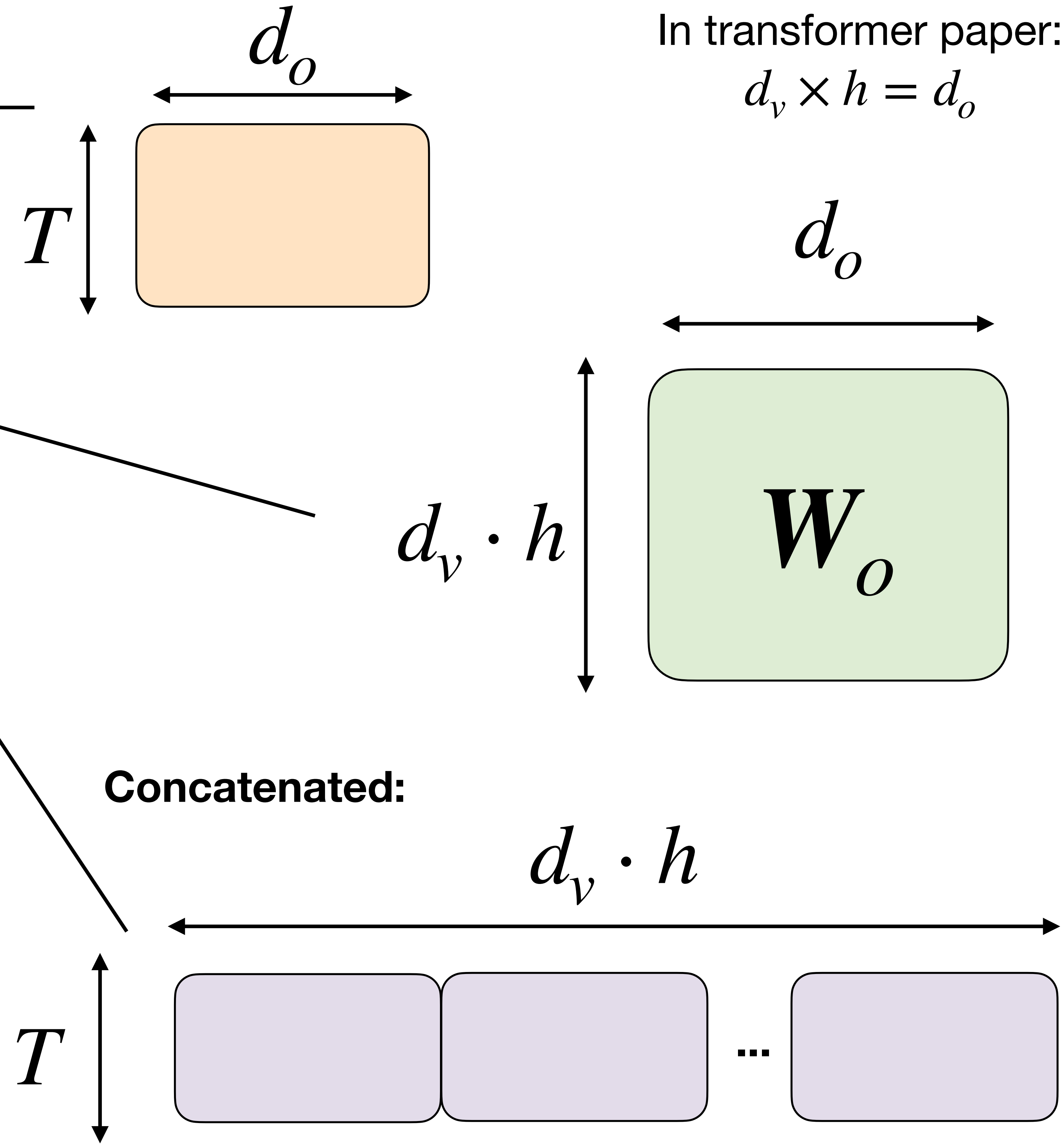
$$T \times d_e = T \times 512$$

and

$$d_v = 512/h = 64$$



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.



Using Attention Without the RNN
-- Self-Attention Mechanism & Transformers

1. Sequence Generation with RNNs
2. Character RNN in PyTorch
3. RNNs with Attention
4. Attention is All We Need
   4.1. Basic Form of Self-Attention
   4.2. Self-Attention & Scaled Dot-Product Attention
   4.3. Multi-Head Attention
5. **Transformer Models**
   5.1. The Transformer Architecture
   5.2. Some Popular Transformer Models: BERT, GPT, and BART
6. Transformer in PyTorch

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

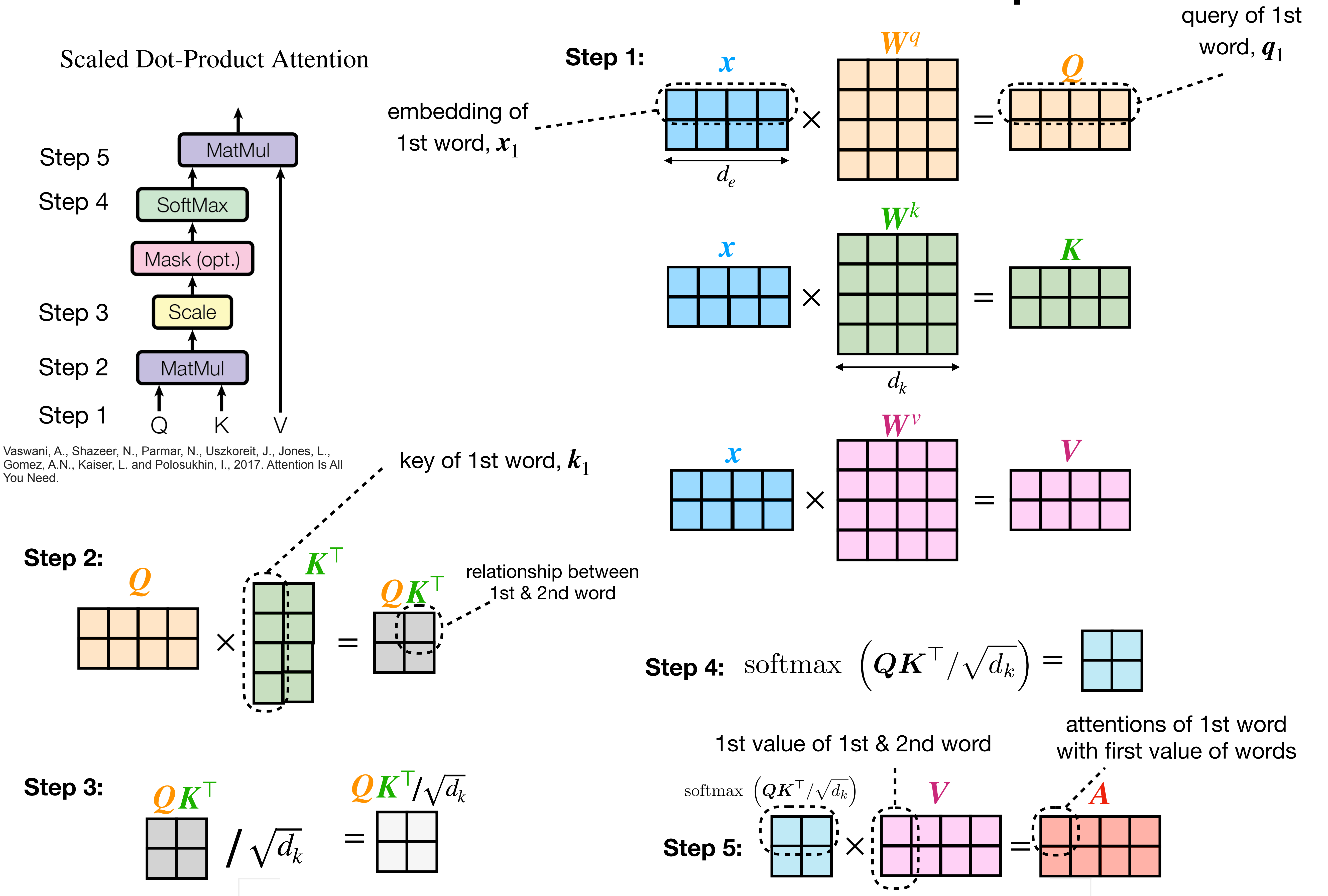
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

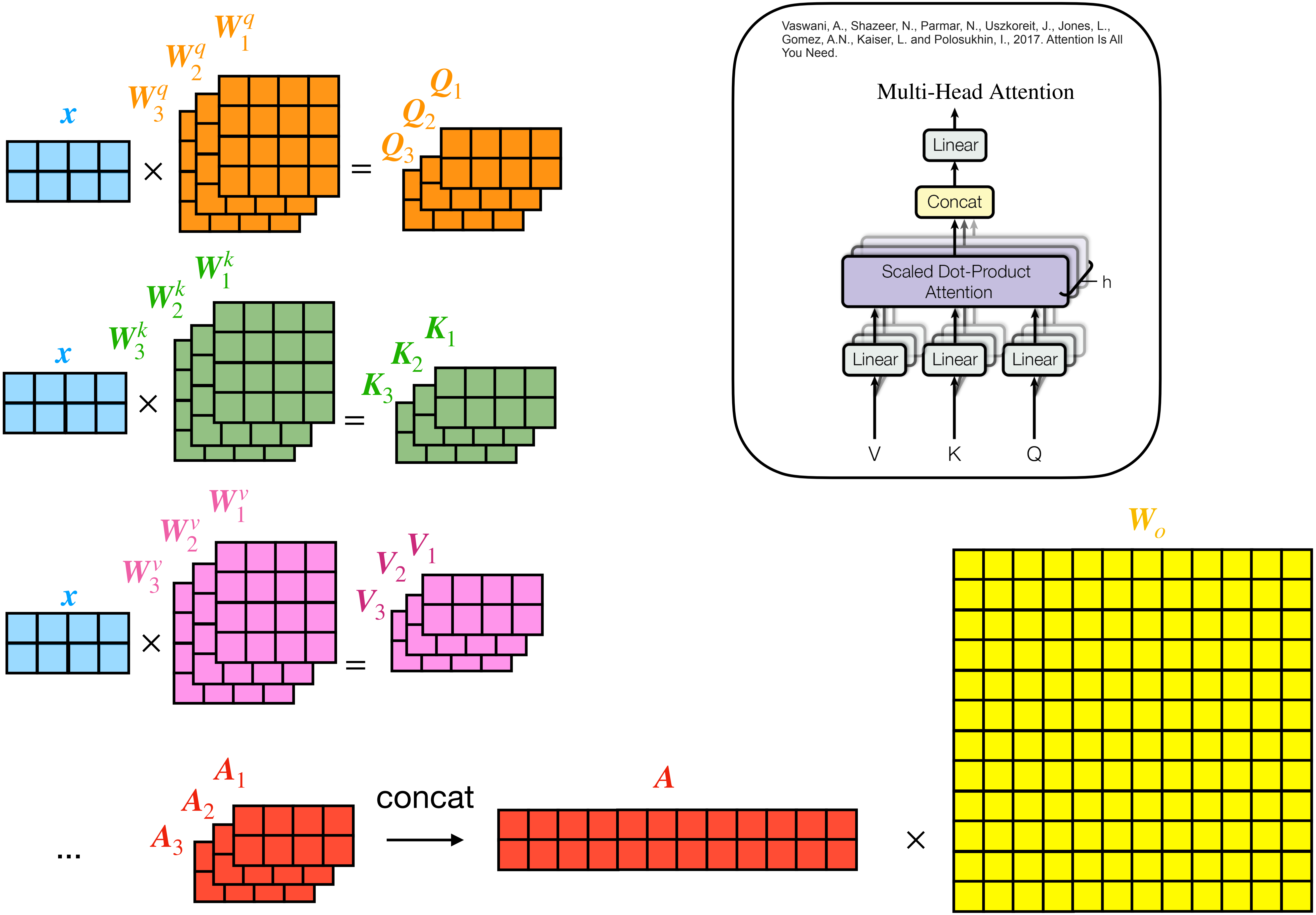
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

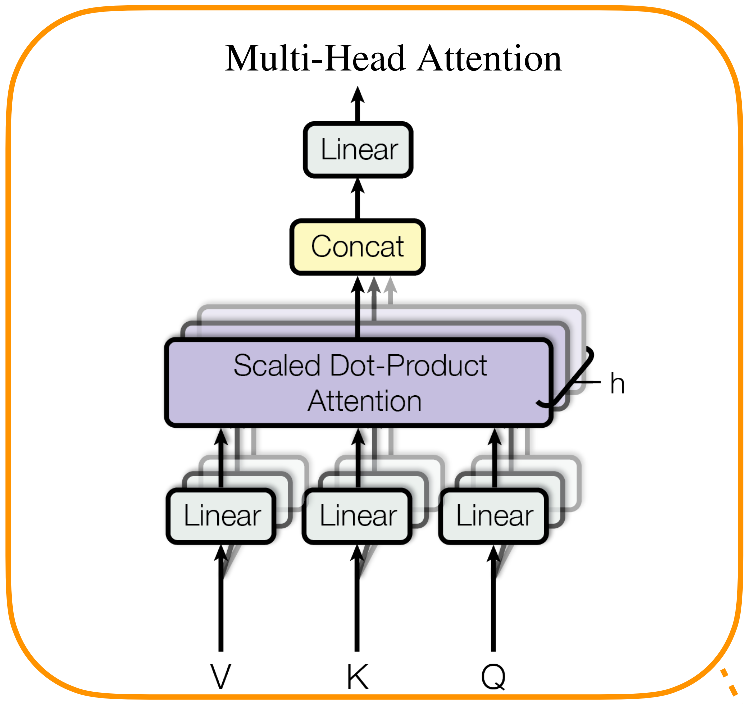
https://arxiv.org/abs/1706.03762

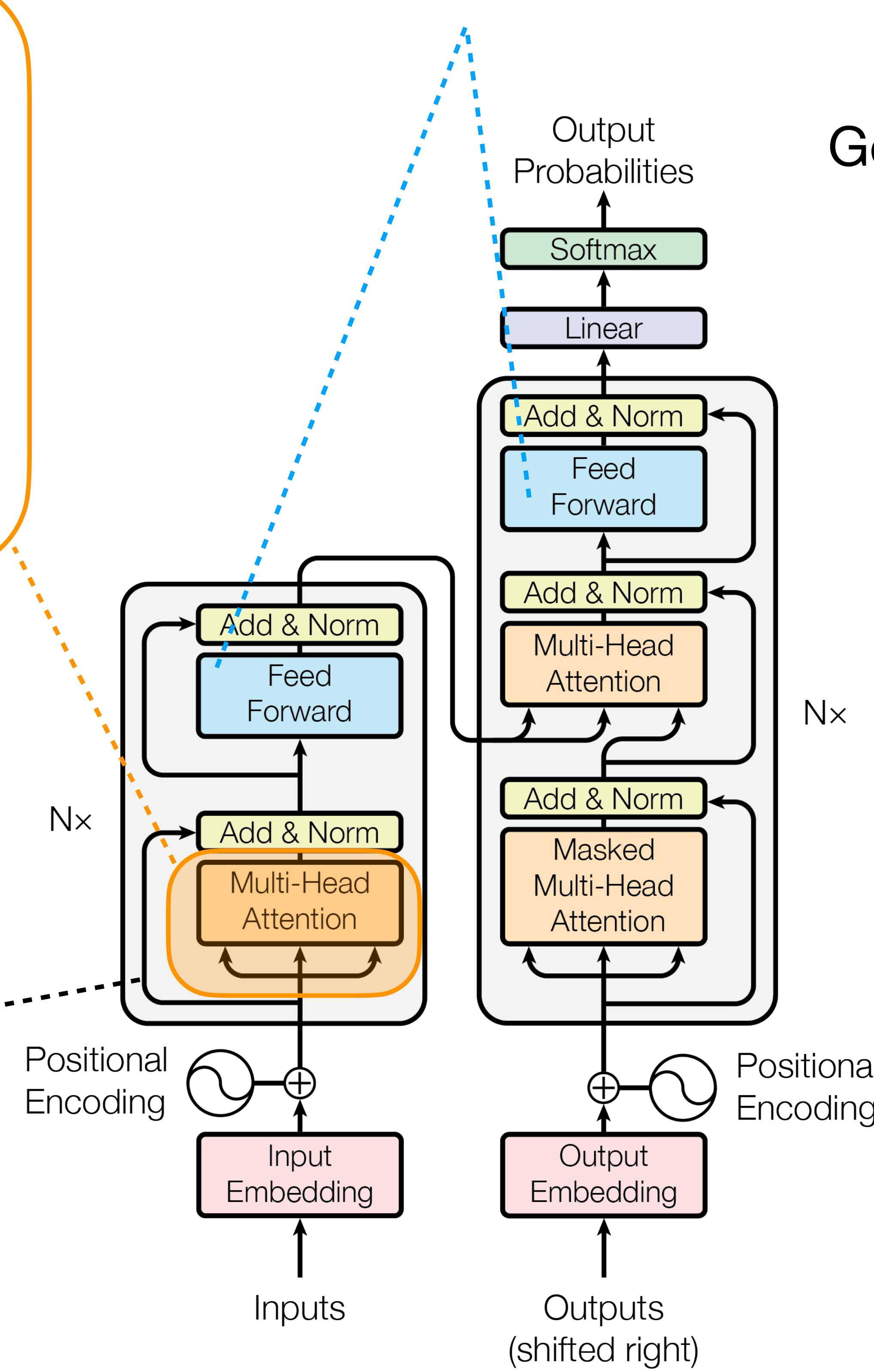
Scaled Dot-Product Attention Recap




Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.
Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q
W_1^q W_2^q W_3^q
x
Q_1 Q_2 Q_3
W_1^k W_2^k W_3^k
K_1 K_2 K_3
W_1^v W_2^v W_3^v
V_1 V_2 V_3
A_1 A_2 A_3
...
concat
A
W_o

Multilayer Perceptrons





Generates output words one at a time

Skip connection (like in ResNet)

$$x + \text{layer}(x)$$

Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

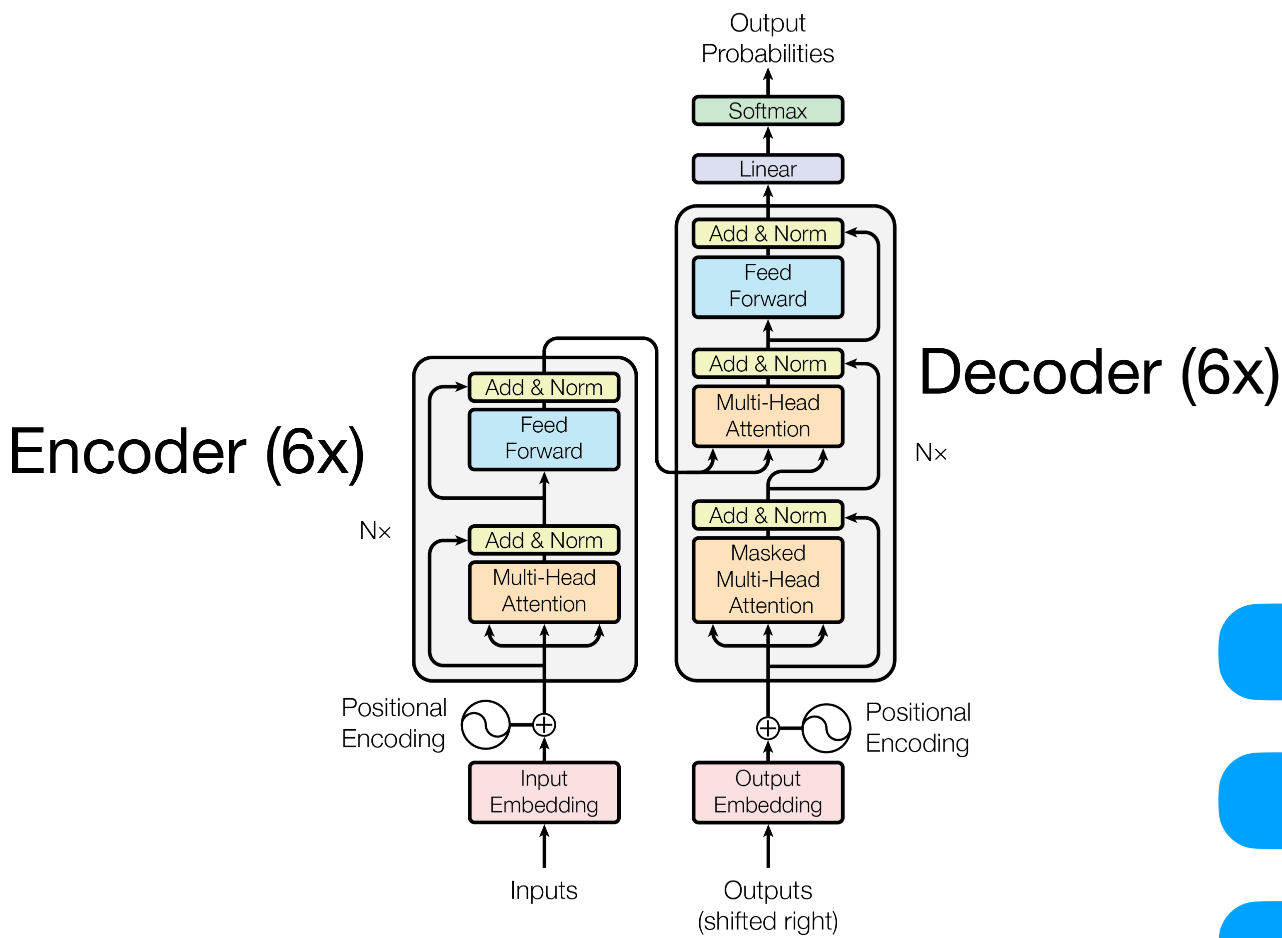
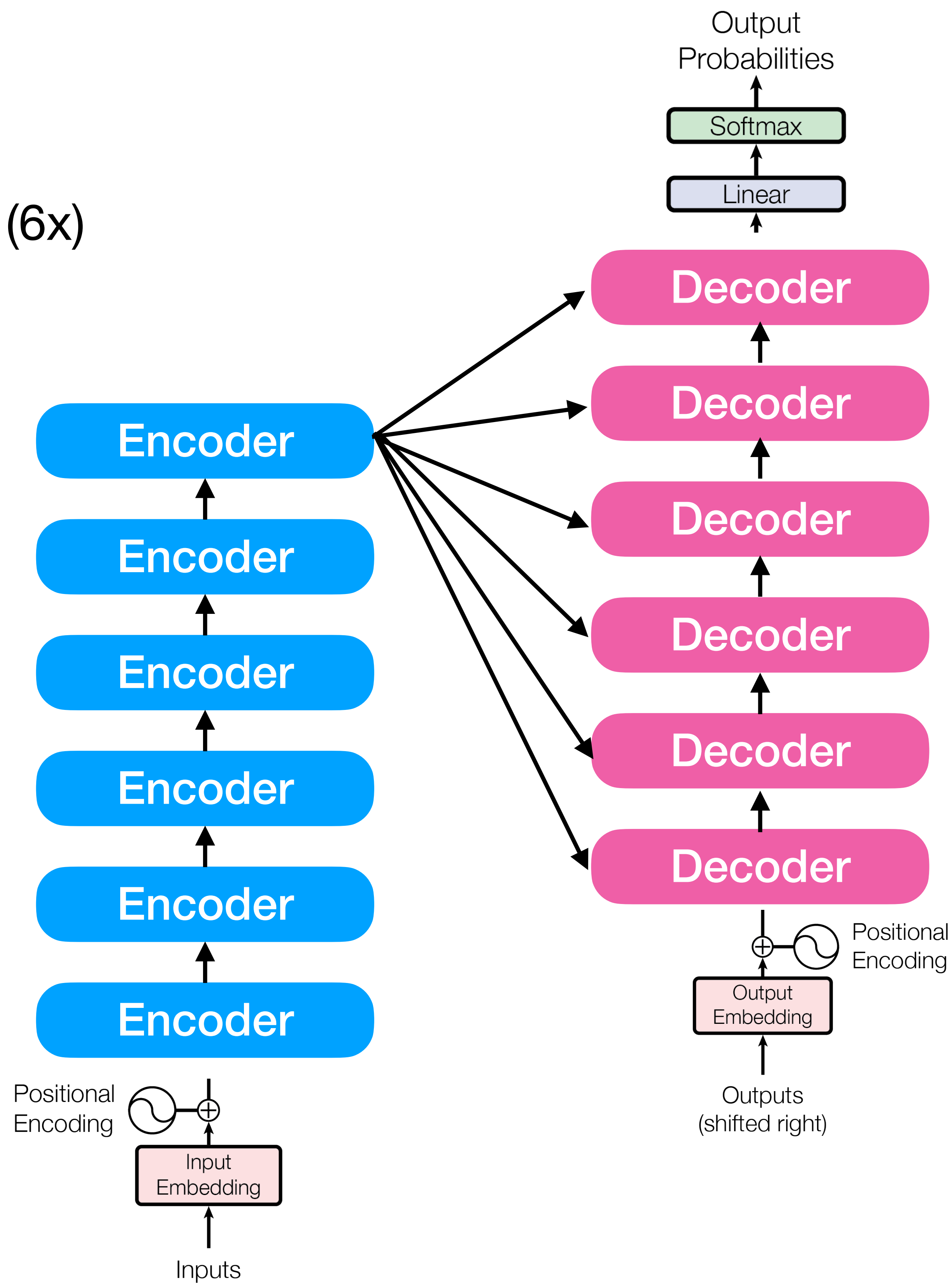
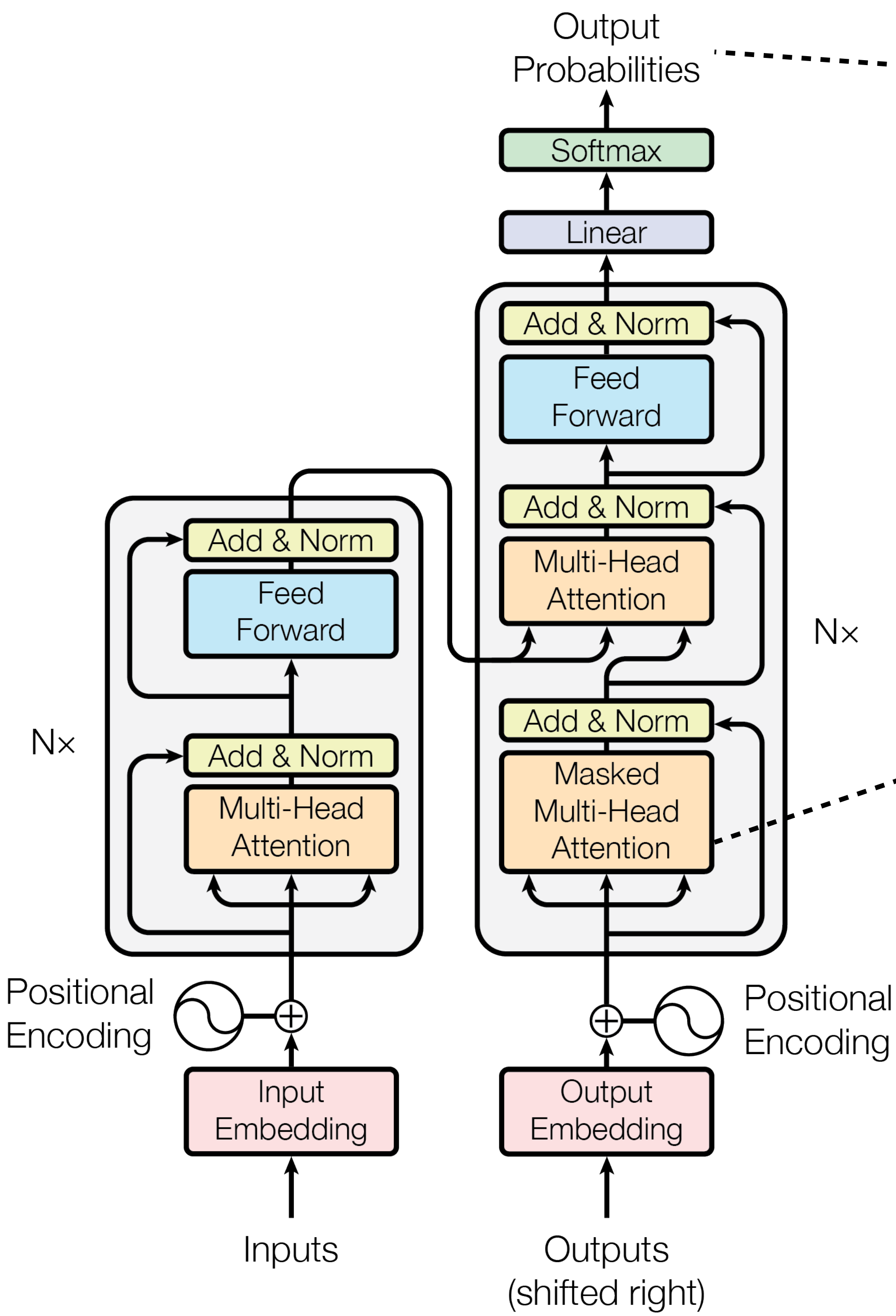


Figure 1: The Transformer - model architecture.

Same structure and dimension in/out for each encoder & decoder



Masked Multi-head attention

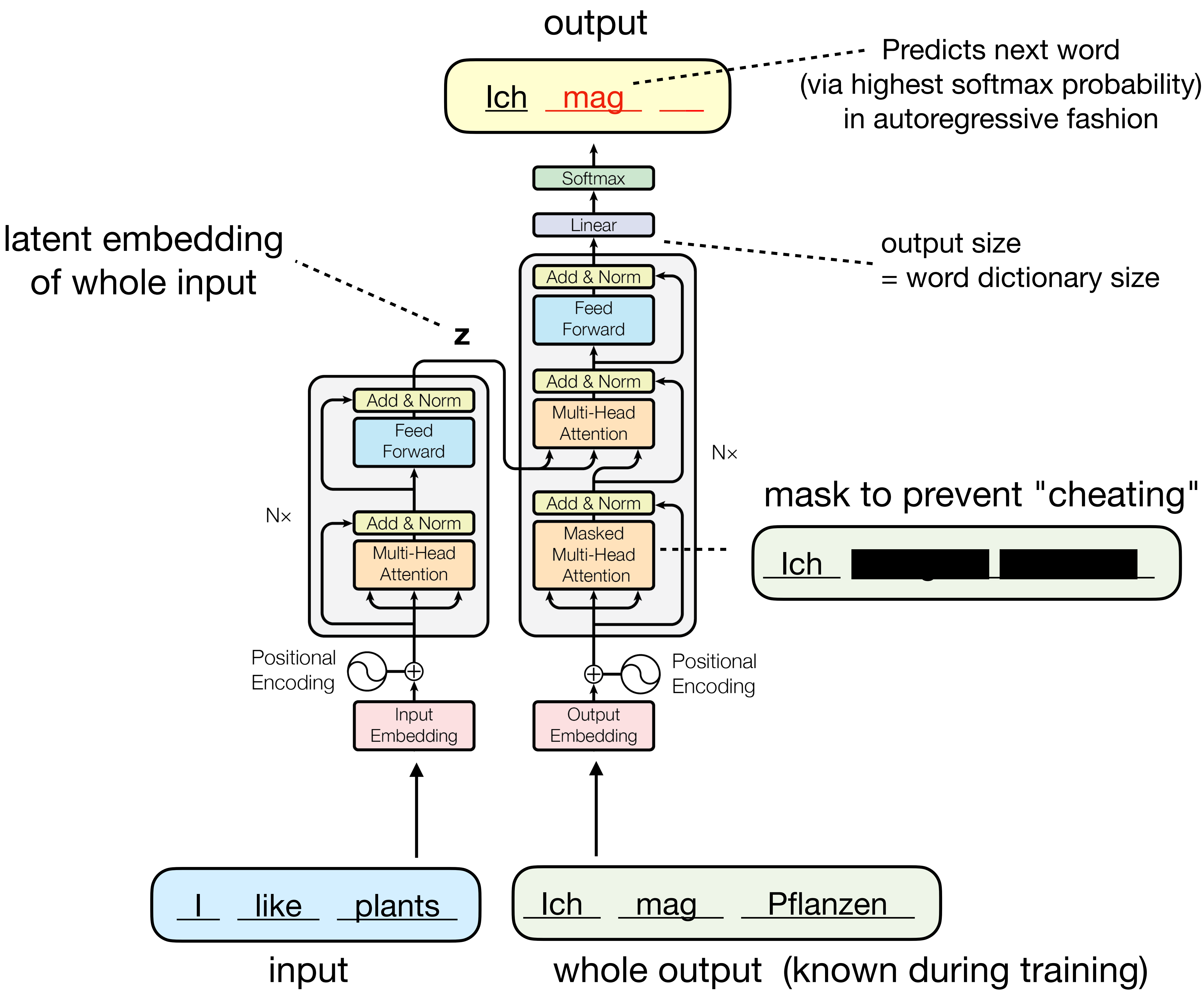


Word probabilities

Mask subsequent sequence elements. I.e., only allow to attend to positions up to and including the current position. This is achieved by setting softmax values for those to $-\infty$

Figure 1: The Transformer - model architecture.

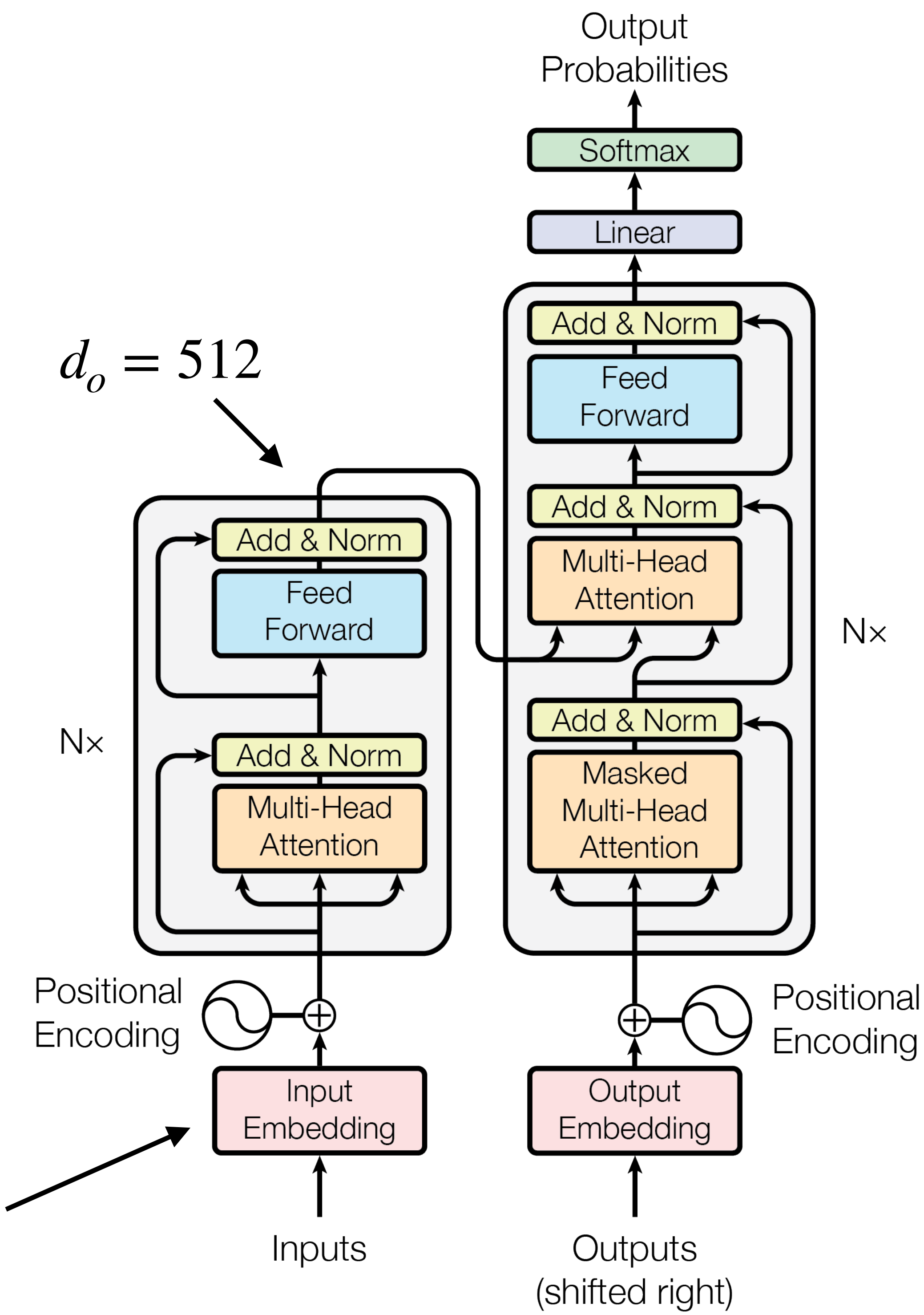
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.


output
Ich mag
Predicts next word
(via highest softmax probability)
in autoregressive fashion
Softmax
Linear
output size
= word dictionary size
latent embedding
of whole input
z
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
mask to prevent "cheating"
Ich [redacted] [redacted]
Add & Norm
Feed
Forward
Nx
Add & Norm
Multi-Head
Attention
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
I like plants
Ich mag Pflanzen
input
whole output (known during training)

Add Positional Encoding Matrix to Word Embedding Matrix

- Scaled dot-product and fully-connected layer are permutation invariant
- Sinusoidal positional encoding is a vector of small values (constants) added to the embeddings
- As a result, same word will have slightly different embeddings depending on where they occur in the sentence

$d_o = 512$



$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d}\text{model}}\right)$$

$$PE_{pos,2i+1} = \cos\left(\frac{\text{pos}}{10000^{(2i+1)/d}\text{model}}\right)$$

$d_e = 512$

Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Layer Normalization

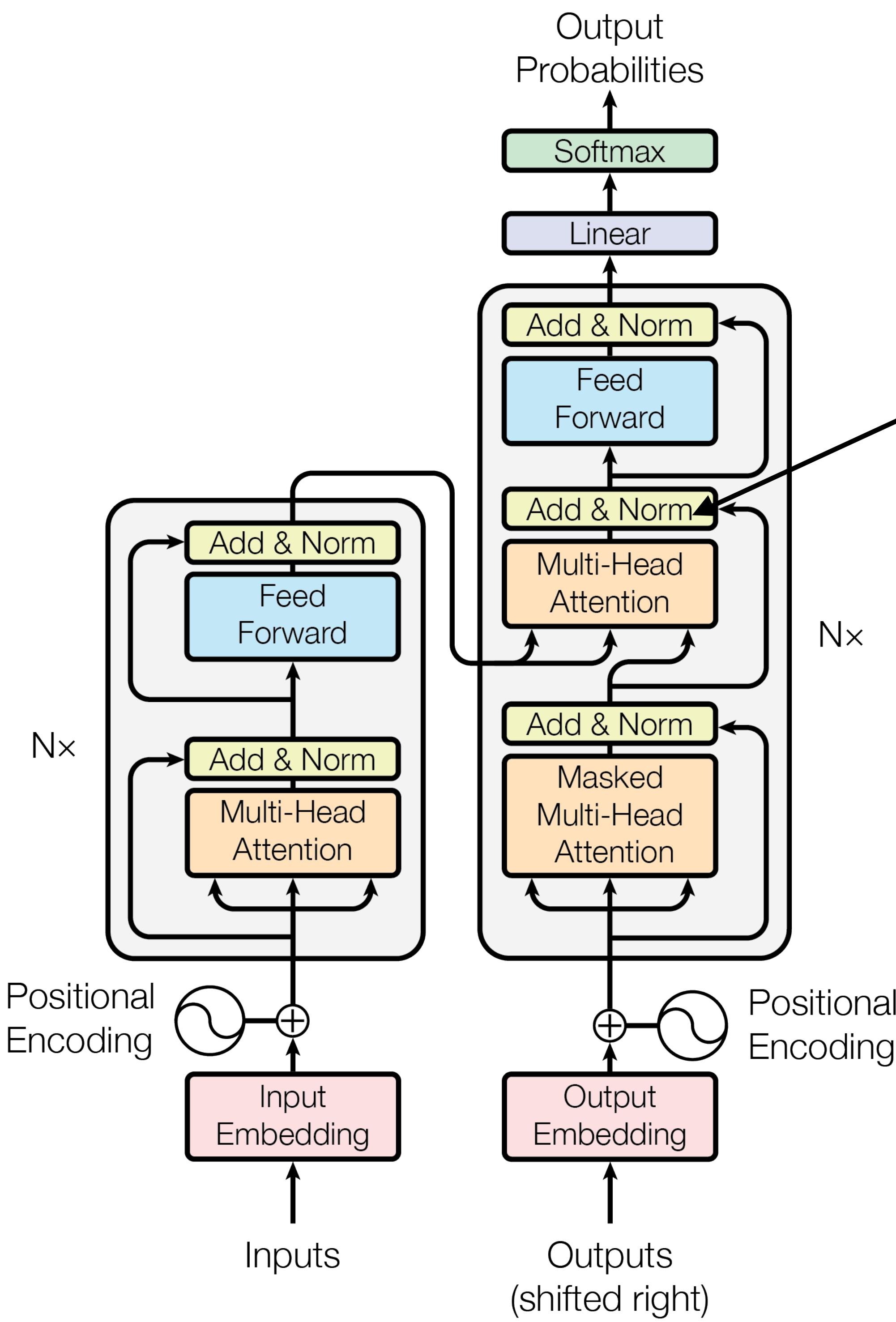


Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

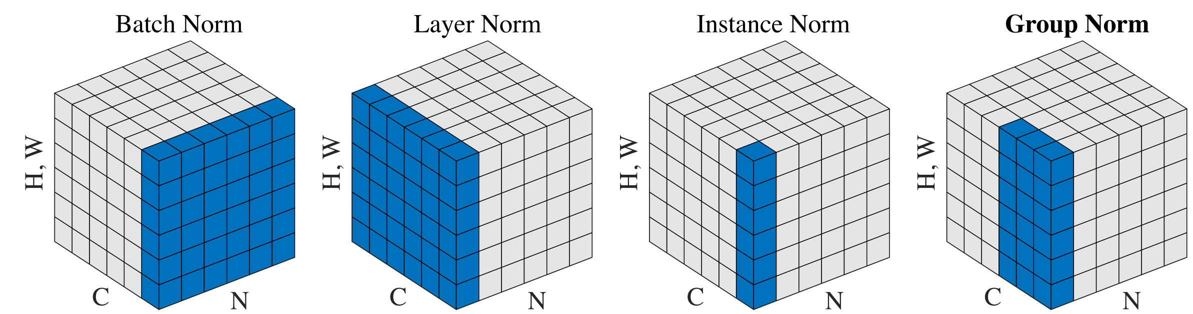


Image source: Wu, Y., & He, K. (2018). Group normalization. ECCV

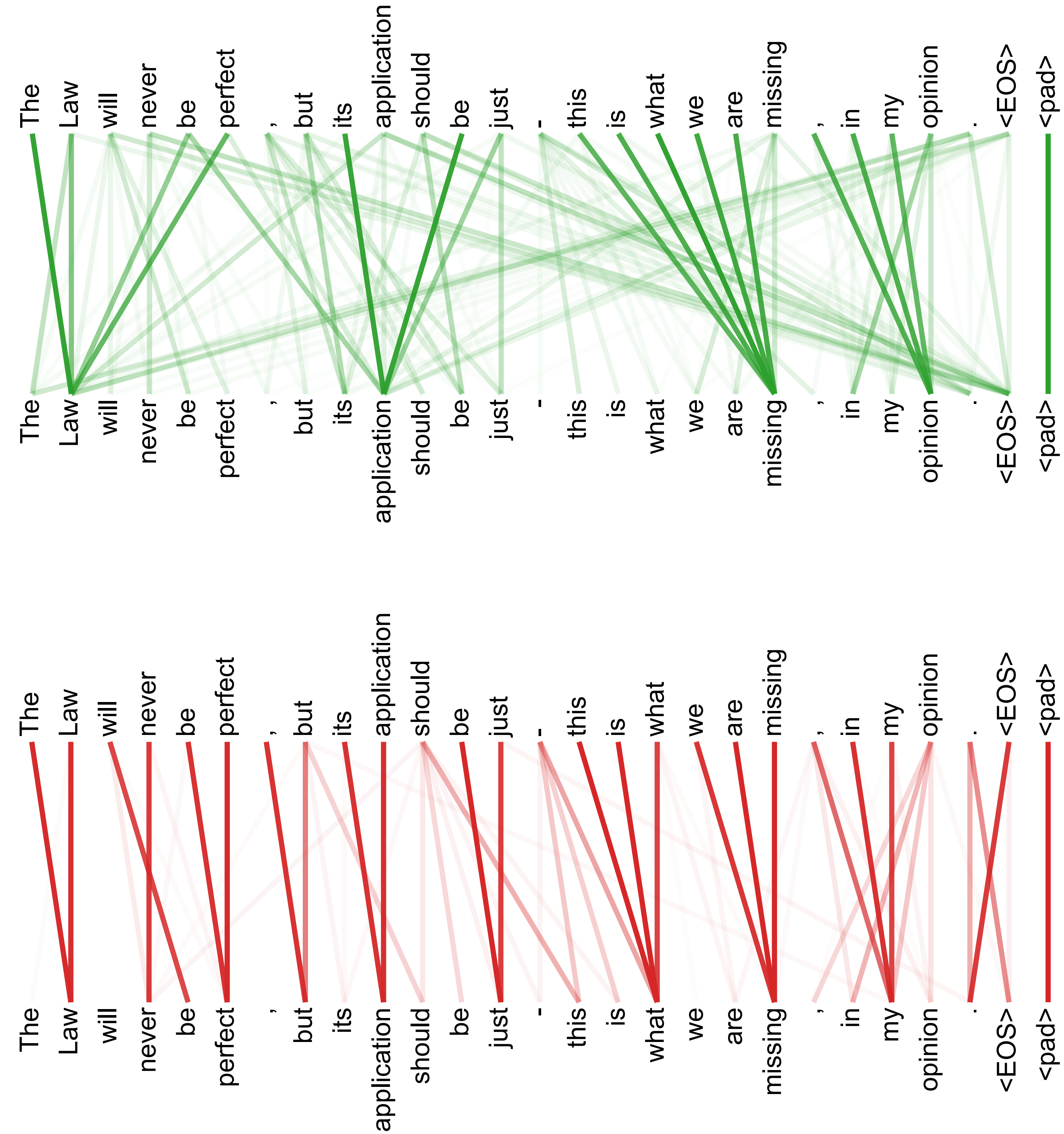


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Using Attention Without the RNN
-- Self-Attention Mechanism & Transformers

Recap

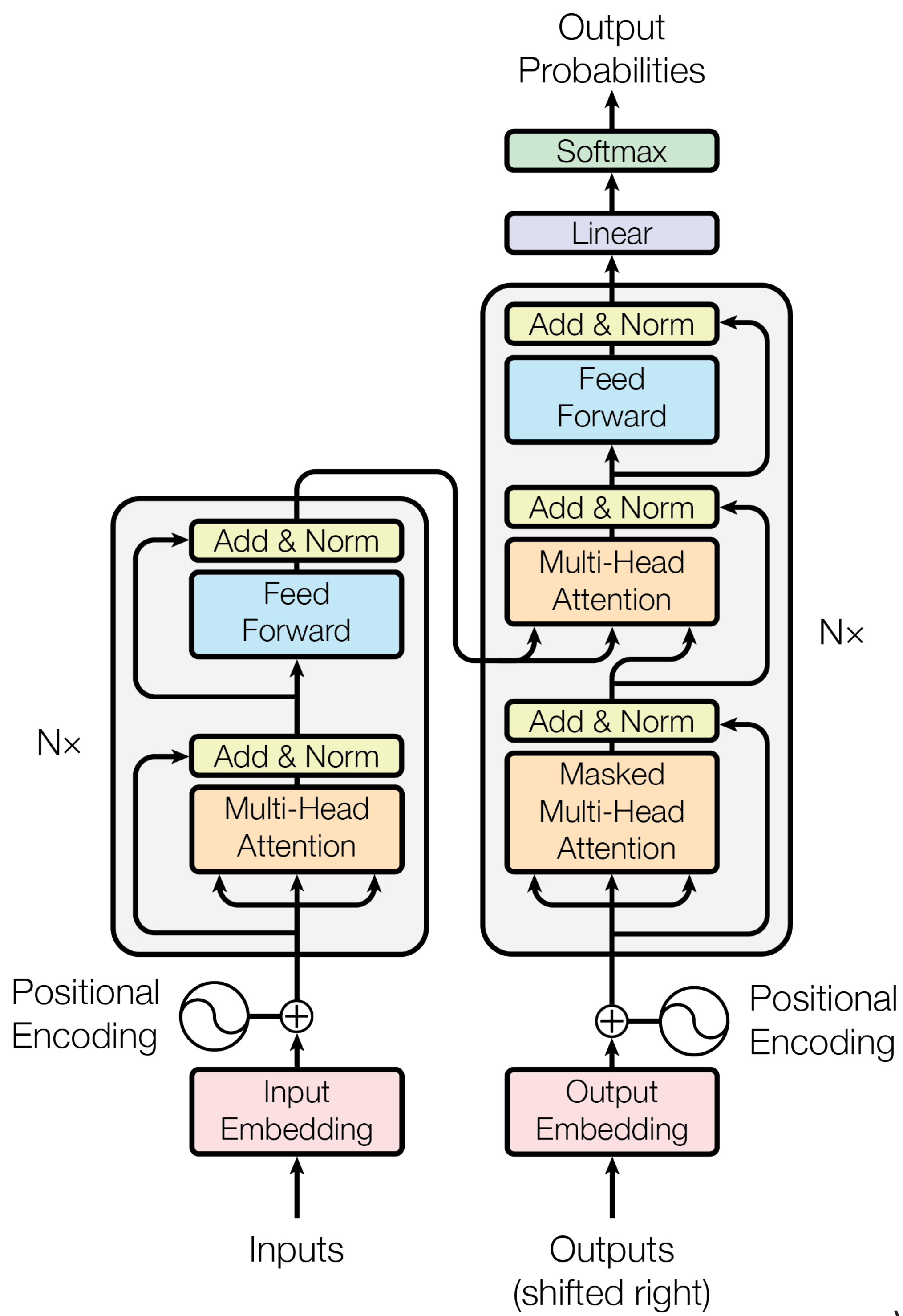


Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

The Two Keys to Success Behind Transformers

1. Self-attention for encoding long-range dependencies

2. Self-supervision for leveraging large unlabeled datasets

Transformer Training Approach

1. Pre-training on large unlabeled datasets
   (self-supervised learning)

2. Training for downstream-tasks on labeled data
   (supervised learning)

   a) fine-tuning approach
   b) feature-based approach

5.2 Some Popular Transformer Models: BERT, GPT, and BART

- 5.2.2 GPT-v1: Generative Pre-Trained Transformer
- 5.2.3 BERT: Bidirectional Encoder Representations from Transformers
- 5.2.4 GPT-v2: Language Models are Unsupervised Multitask Learners
- 5.2.5 GPT-v3: Language Models are Few-Shot Learners
- 5.2.6 BART: Combining Bidirectional and Auto-Regressive Transformers
- 5.2.7: Closing Words -- The Recent Growth of Language Transformers

5.2.2 GPT-v1:

Generative Pre-Trained Transformer

GPT (<u>G</u>enerative <u>P</u>re-trained <u>T</u>ransformer)

- Developed by OpenAI
- Unidirectional: trained to predict next word in a sentence

GPT (110 million parameters)
Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT-2 1.5 billion parameters)
Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.
https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

GPT-3 (175 billion parameters)
Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165. https://arxiv.org/abs/2005.14165

GPT-v1 Key Concepts

- Bottleneck: Lack of labeled data
- 2-step training process ("semi-supervised")
  1. Generative pre-training (on unlabeled data); unsupervised/"self-supervised" learning
  2. Discriminative fine-tuning (on labeled data), supervised learning
- Pre-training on large BookCorpus dataset (7000 books)
- Based on decoder architecture from original Transformer ("Attention Is All You Need")

GPT-v1 Architecture and Downstream Tasks

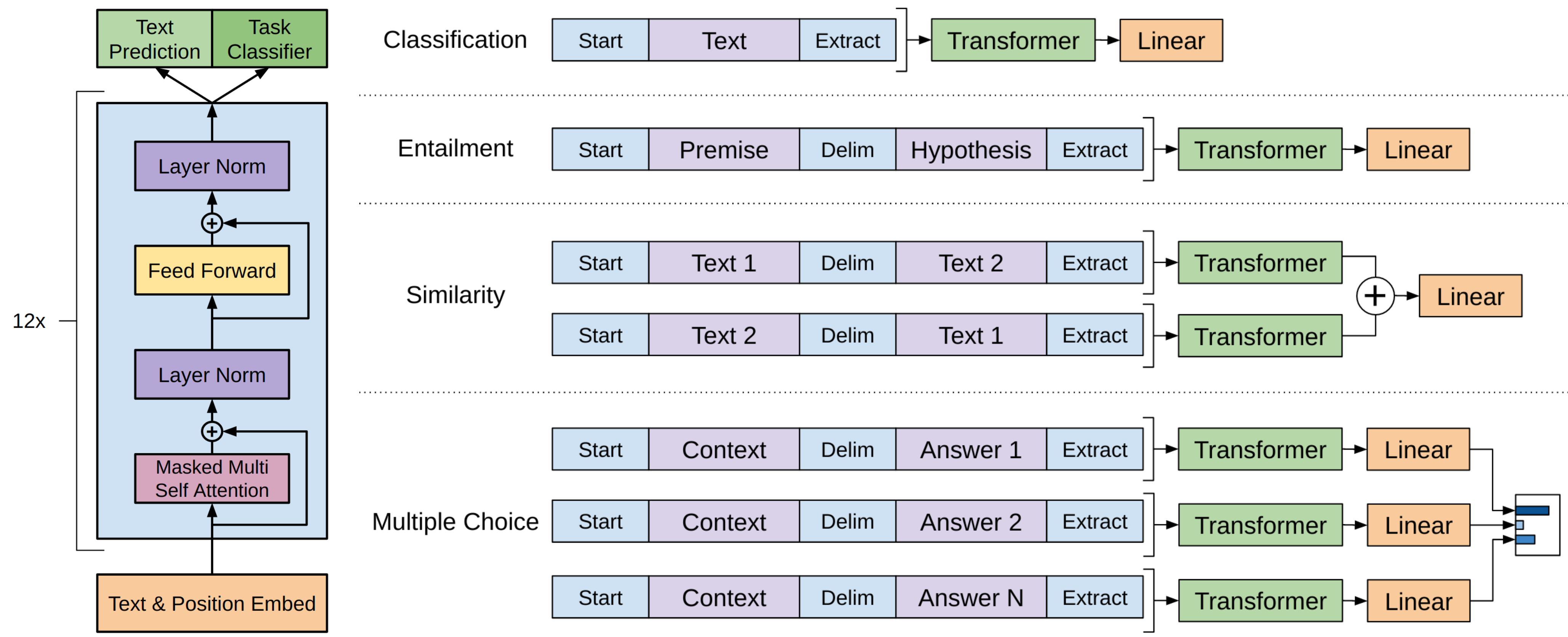


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT-v1 Ablation Study

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

5.2.3 BERT: Bidirectional Encoder Representations from Transformers

BERT 

(Bidirectional Encoder Representations from Transformers)

Paper: Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding.
https://arxiv.org/abs/1810.04805
(Google Research, 2018)

- multi-layer bidirectional transformer encoder
- architecture almost identical to original transformer & GPT, except
    - bidirectional masking (known as "Cloze" task, Taylor 1953*)
    - next sentence prediction as additional pre-training task

* Wilson L Taylor. 1953. Cloze procedure: A new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433.

BERT Inputs

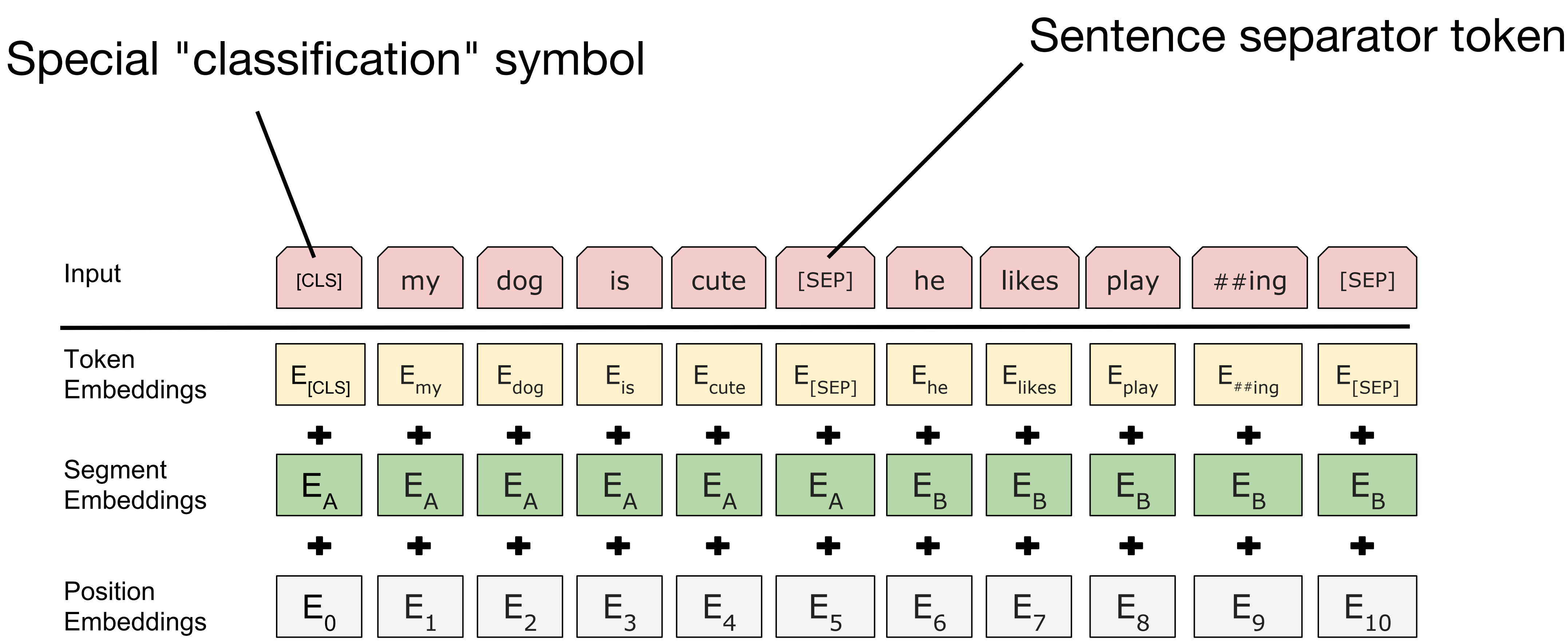


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Token embeddings are WordPiece embeddings* with vocabulary size of 30,000

* Wu Y, Schuster M, Chen Z, Le QV, Norouzi M, Macherey W, Krikun M, Cao Y, Gao Q, Macherey K, Klingner J. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144. 2016 https://arxiv.org/abs/1609.08144

BERT Pre-Training Tasks

Pre-training datasets

- BookCorpus (800 million words)
- Wikipedia (2500 million words)

Pre-training tasks

- Masked language model ("Cloze")
- Next sentence prediction

BERT Pre-Training Task #1

Masked Language Model

Input sentence: A quick brown fox jumps over the lazy dog

"Mark" 15% of the words

80%: replace with [MASK]

10%: replace with random word (coffee)

10%: leave as is (fox)
to mimick fine-tuning scenario

BERT Pre-Training Task #1

Masked Language Model

Input sentence: A quick brown fox jumps over the lazy dog

Randomly masked: A quick brown [MASK] jumps over the lazy dog

BERT

Possible classes (all words)

0.2%	ant
...	...
11%	fox
...	...
0.01%	zoo

BERT Pre-Training Task #2

Next Sentence Prediction

Balanced binary classification task (50% IsNext, 50% NotNext)

Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

BERT Pre-Training & Downstream Tasks

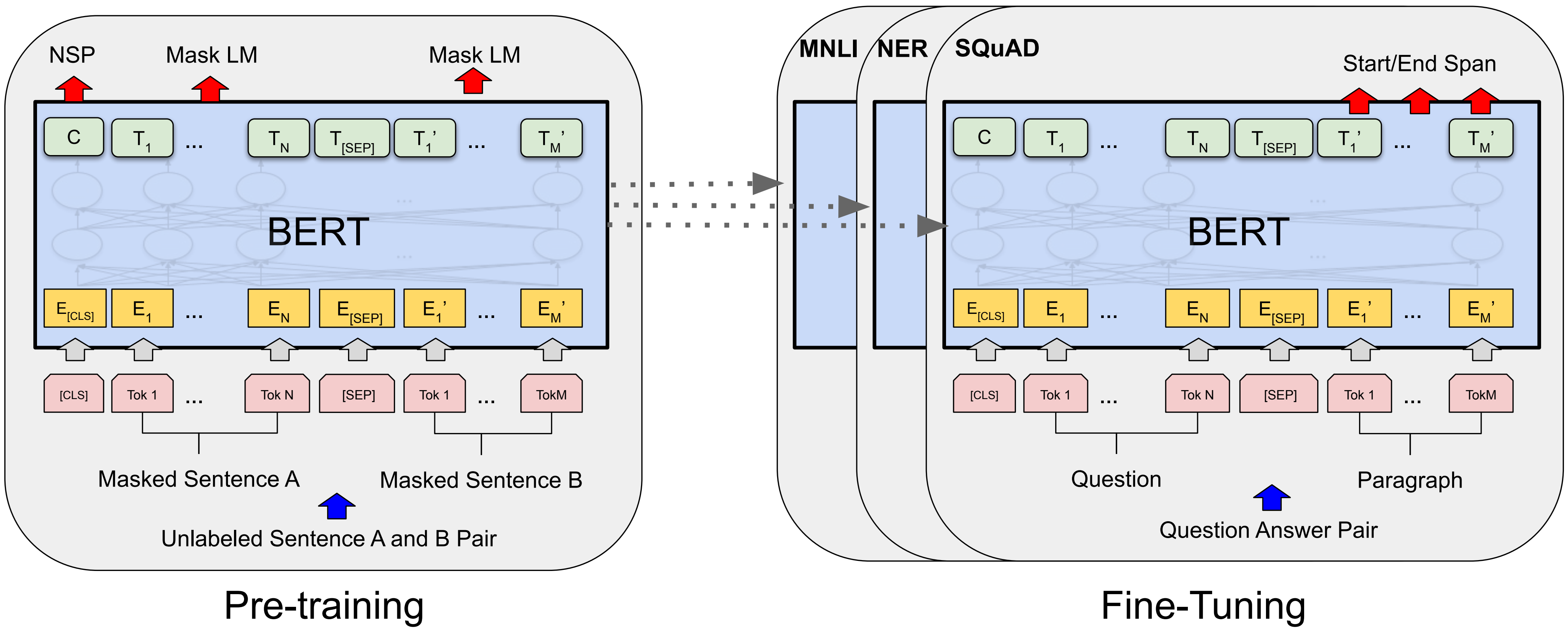


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

Transformer Training Approach

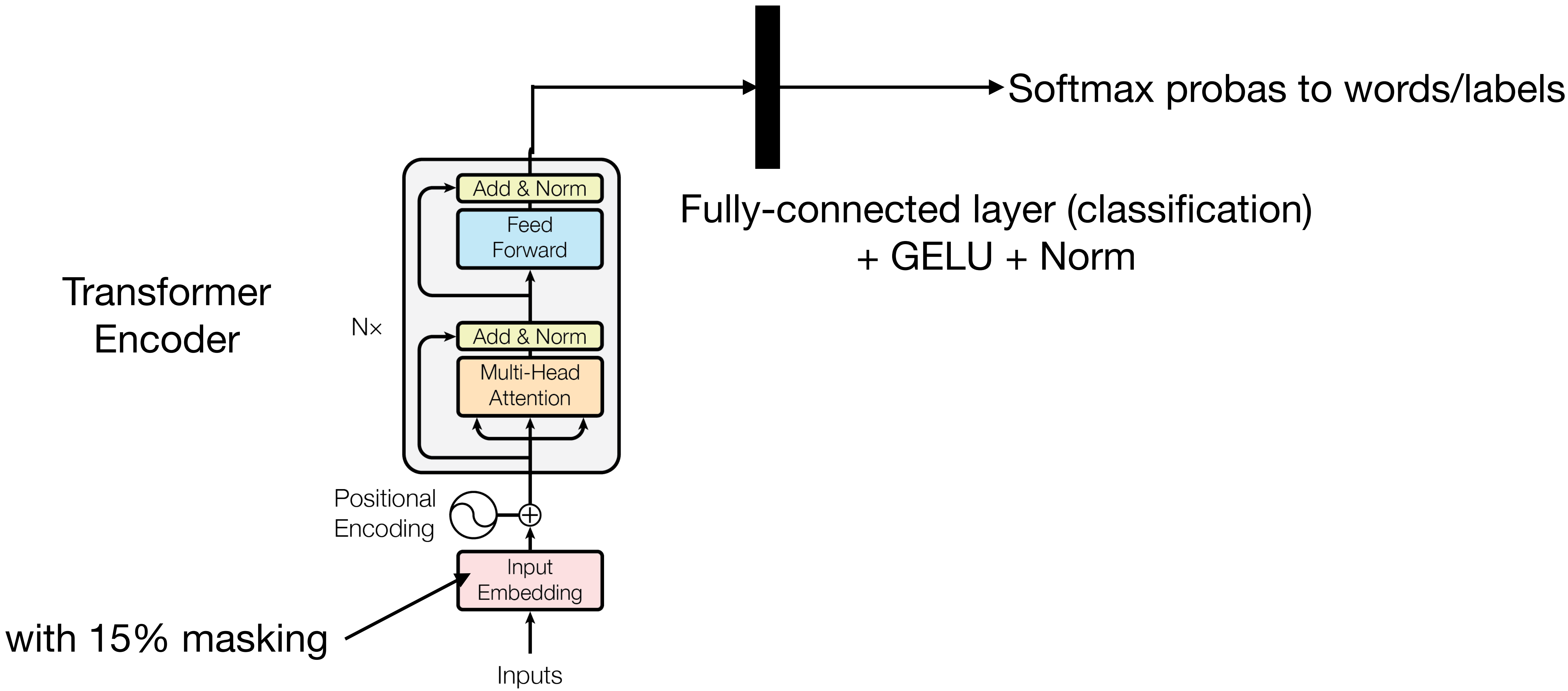
1. **Pre-training** on large unlabeled datasets
   (self-supervised learning)

2. **Training for downstream-tasks** on labeled data
   (supervised learning)

   a) fine-tuning approach
   b) feature-based approach
   (nowadays also called "fine-tuning")

BERT Pre-Training & Fine-Tuning Approach

- Add classification layer
- Train end-to-end on labeled dataset for downstream task (update ALL parameters)



BERT vs GPT-v1 Performance

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	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

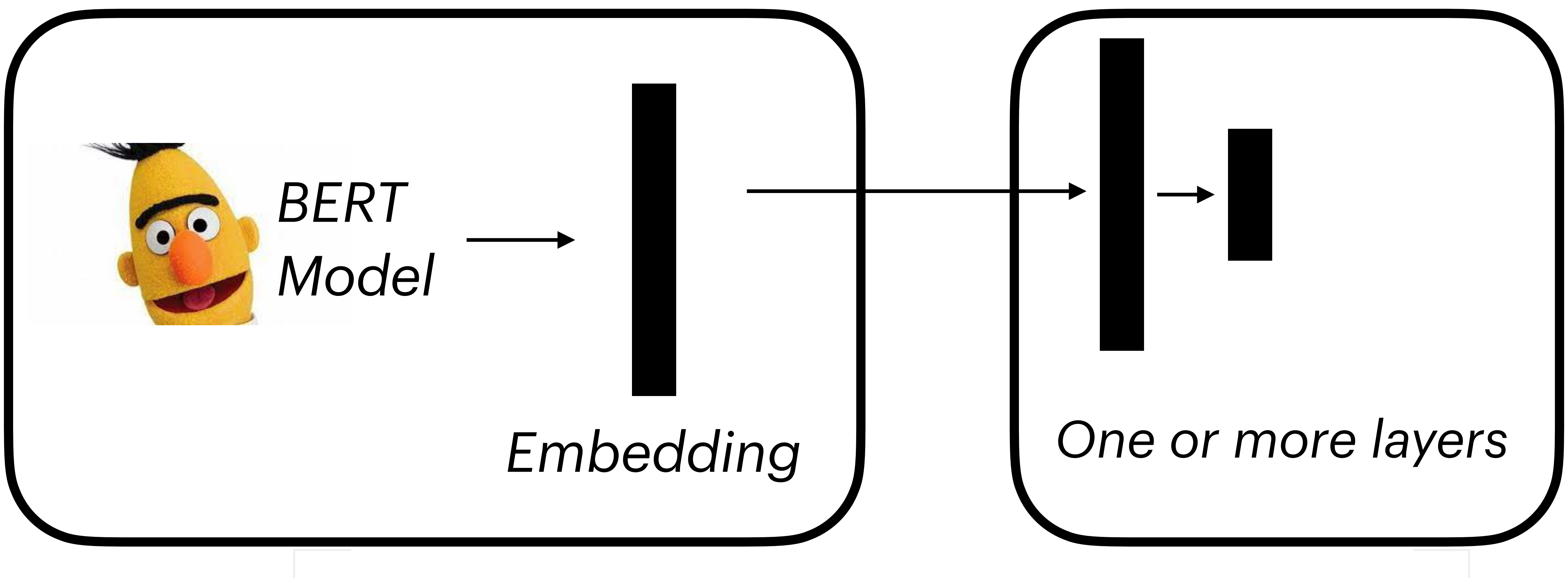
Table 1: GLUE Test results, scored by the evaluation server (`https://gluebenchmark.com/leaderboard`). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

BERT Pre-Training & Feature-based Training

- Keep BERT frozen after pre-training
- Create BERT embeddings for labeled dataset for downstream task
  and train new model on these embeddings
  (in original paper,
  2-layer biLSTM on embeddings from concatenated last 4 layers performed best)

1) Download BERT
pre-trained on large corpus
(in self-supervised fashion)

2) Feature-based training ("fine-tuning")
on target task
(supervised learning)



System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	**93.1**
Fine-tuning approach		
$\text{BERT}_{\text{LARGE}}$	96.6	92.8
$\text{BERT}_{\text{BASE}}$	96.4	92.4
Feature-based approach ($\text{BERT}_{\text{BASE}}$)		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

Table 7: CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

5.2.4 GPT-v2: Language Models are Unsupervised Multitask Learners

GPT (Generative Pre-trained Transformer)

- Developed by OpenAI
- Unidirectional: trained to predict next word in a sentence

GPT (110 million parameters)

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT-2 (1.5 billion parameters)

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.
https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

GPT-3 (175 billion parameters)

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165. https://arxiv.org/abs/2005.14165

GPT-v1 Architecture and Downstream Tasks

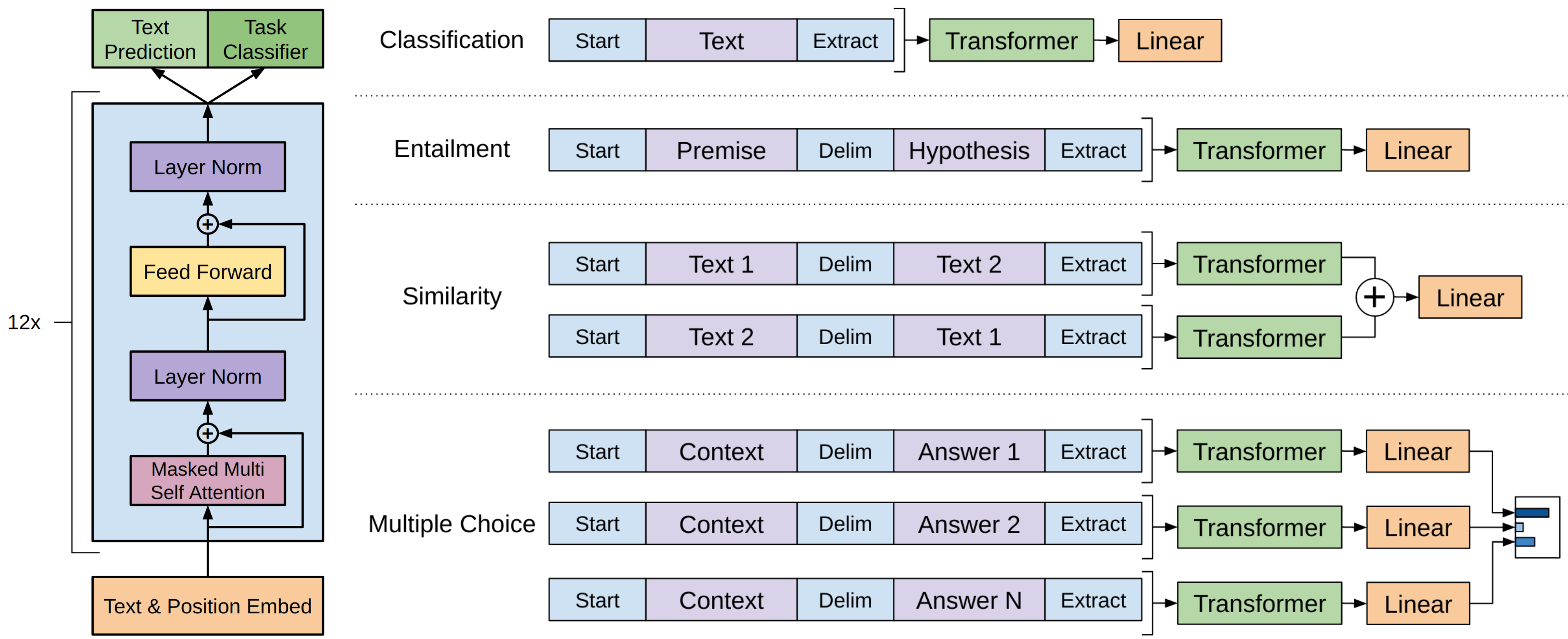


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.
https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT-v2 Key Concepts

- Unidirectional like GPT-v1
- Compared to GPT-v1
  - Larger model (the larger the better)
  - Larger unlabeled dataset (the larger the better)
  - No fine-tuning (use zero-shot transfer instead)

GPT-v2 Architecture

- Overall, similar to GPT-v1 (which is based on original Transformer decoder)
- Some small rearranging of layer norm and residual layers
- Increase vocabulary size from 30,000 -> 50,257
- Increase context size from 512 -> 1024 tokens
- Overall, 1.5 billion instead of 110 million parameters

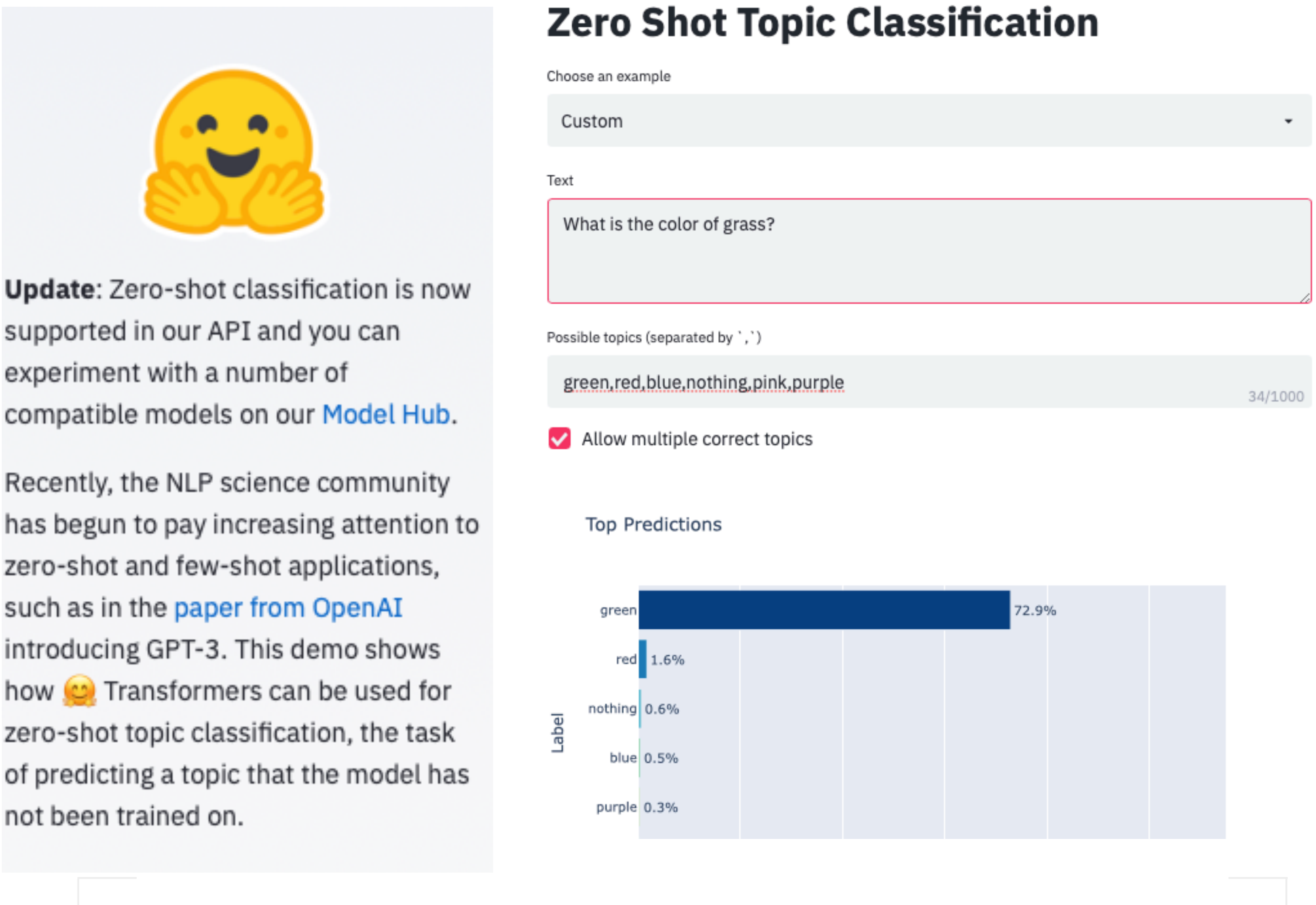
GPT-v2 Training Dataset

- WebText (millions of webpages)
- Emphasized dataset quality
- Based on Reddit posts with more than 3 karma
  - Get 45 million links to websites
    - After preprocessing and cleaning: 8 million documents
    - 40 Gb of text

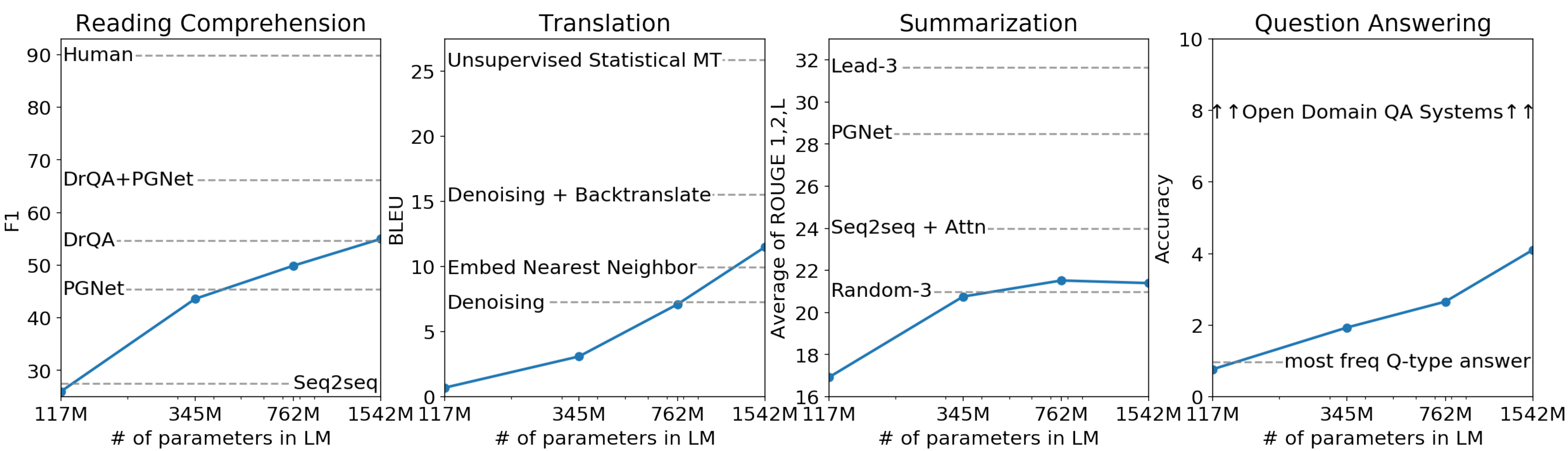
Zero-Shot Task Transfer

In contrast to GPT-v1, no specific instruction / rearranging for specific tasks

https://huggingface.co/models?filter=zero-shot-classification



Language Models are Unsupervised Multitask Learners



Language Models are Unsupervised Multitask Learners

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW* (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	**21.8**
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**	65.85	1.16	1.17	37.50	75.20
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.72
762M	**10.87**	**60.12**	**93.45**	**88.0**	**19.93**	**40.31**	**0.97**	**1.02**	22.05	44.575
1542M	**8.63**	**63.24**	**93.30**	**89.05**	**18.34**	**35.76**	**0.93**	**0.98**	**17.48**	42.16

* bad 1BW performance probably due to sentence-level reshuffling in that dataset, so larger, long-range contexts are lost

5.2.5 GPT-v3: Language Models are Few-Shot Learners

GPT (<u>G</u>enerative <u>P</u>re-trained <u>T</u>ransformer)

- Developed by OpenAI
- Unidirectional: trained to predict next word in a sentence

GPT (110 million parameters)

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT-2 1.5 billion parameters)

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.
https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

GPT-3 (175 billion parameters)

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165. https://arxiv.org/abs/2005.14165

GPT-v3 Architecture

- Overall, similar to GPT-v2
- 175 billion instead 1.5 billion parameters (more layers etc.)
- Double the context size (2048 instead of 1024)
- Larger word embeddings (12.8k instead of 1.6k)
- Attention pattern from Sparse Transformer*

*Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences With Sparse Transformers, 2019.

GPT-v3 Training Datasets

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

Implicit Task Learning
(... While Learning to Predict the Next Word)

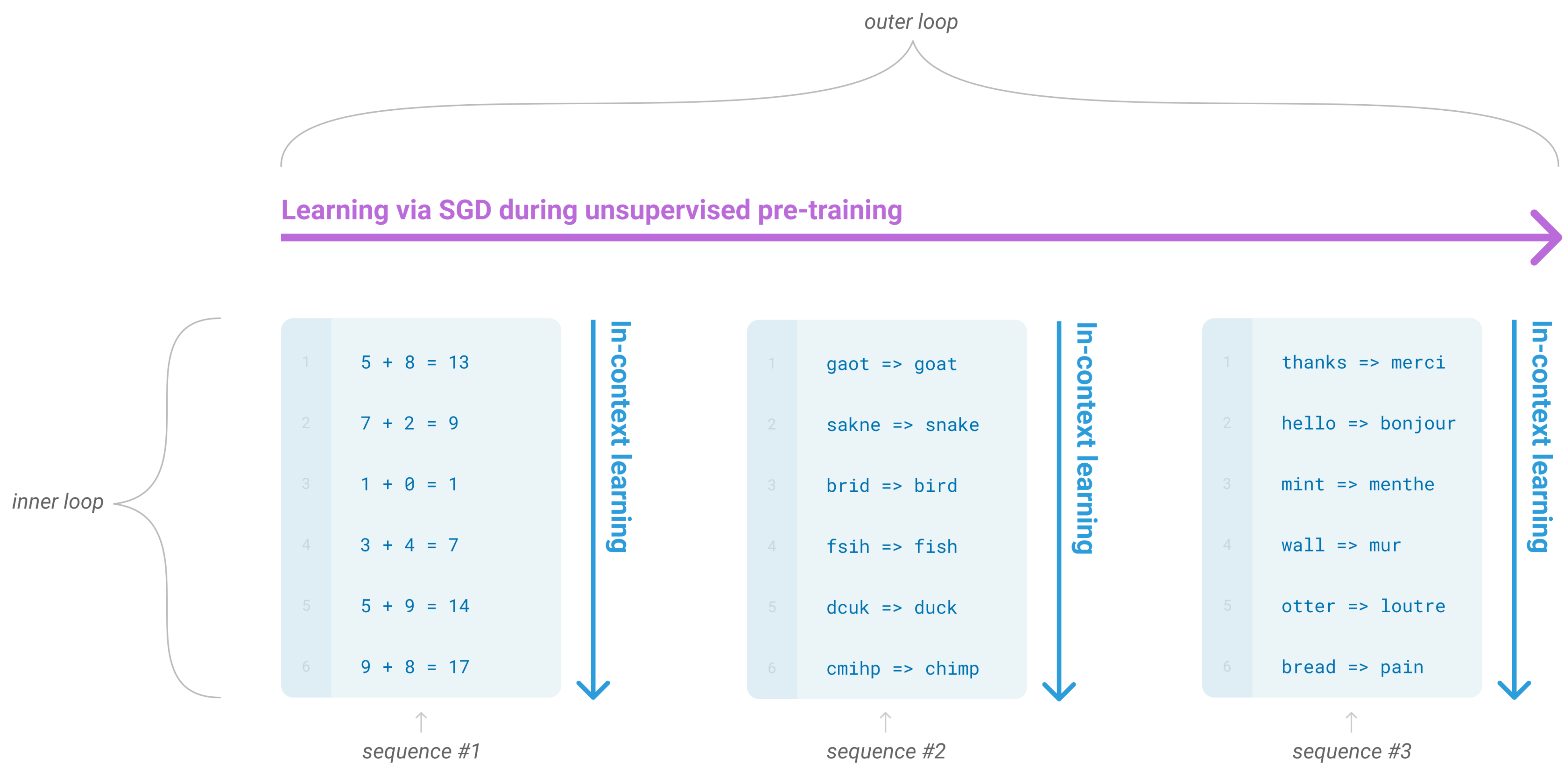


Figure 1.1: Language model meta-learning. During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task. We use the term "in-context learning" to describe the inner loop of this process, which occurs within the forward-pass upon each sequence. The sequences in this diagram are not intended to be representative of the data a model would see during pre-training, but are intended to show that there are sometimes repeated sub-tasks embedded within a single sequence.

Showing Examples vs Fine-Tuning

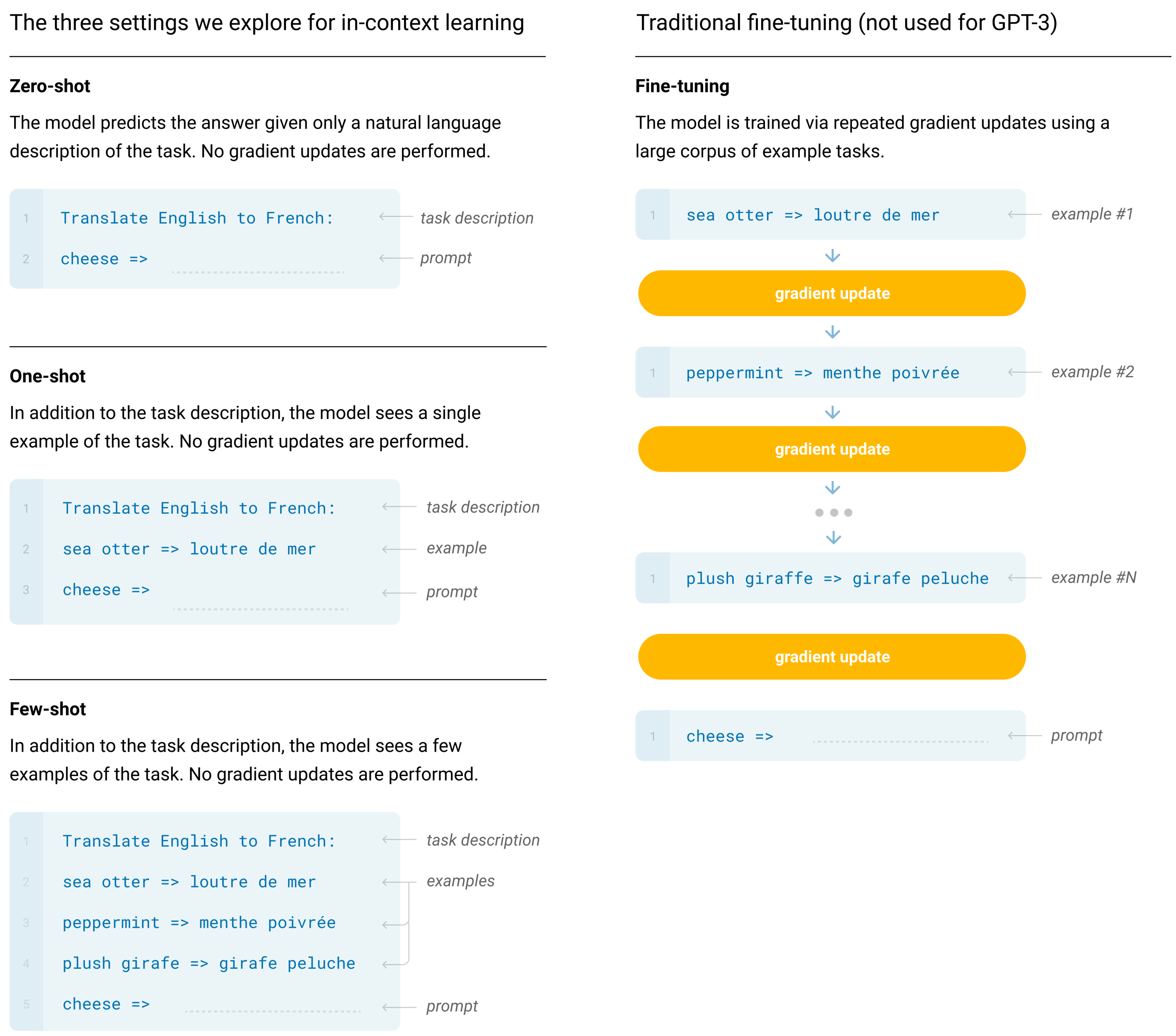


Figure 2.1: Zero-shot, one-shot and few-shot, contrasted with traditional fine-tuning. The panels above show four methods for performing a task with a language model – fine-tuning is the traditional method, whereas zero-, one-, and few-shot, which we study in this work, require the model to perform the task with only forward passes at test time. We typically present the model with a few dozen examples in the few shot setting. Exact phrasings for all task descriptions, examples and prompts can be found in Appendix G.

Some of the Many Results ...

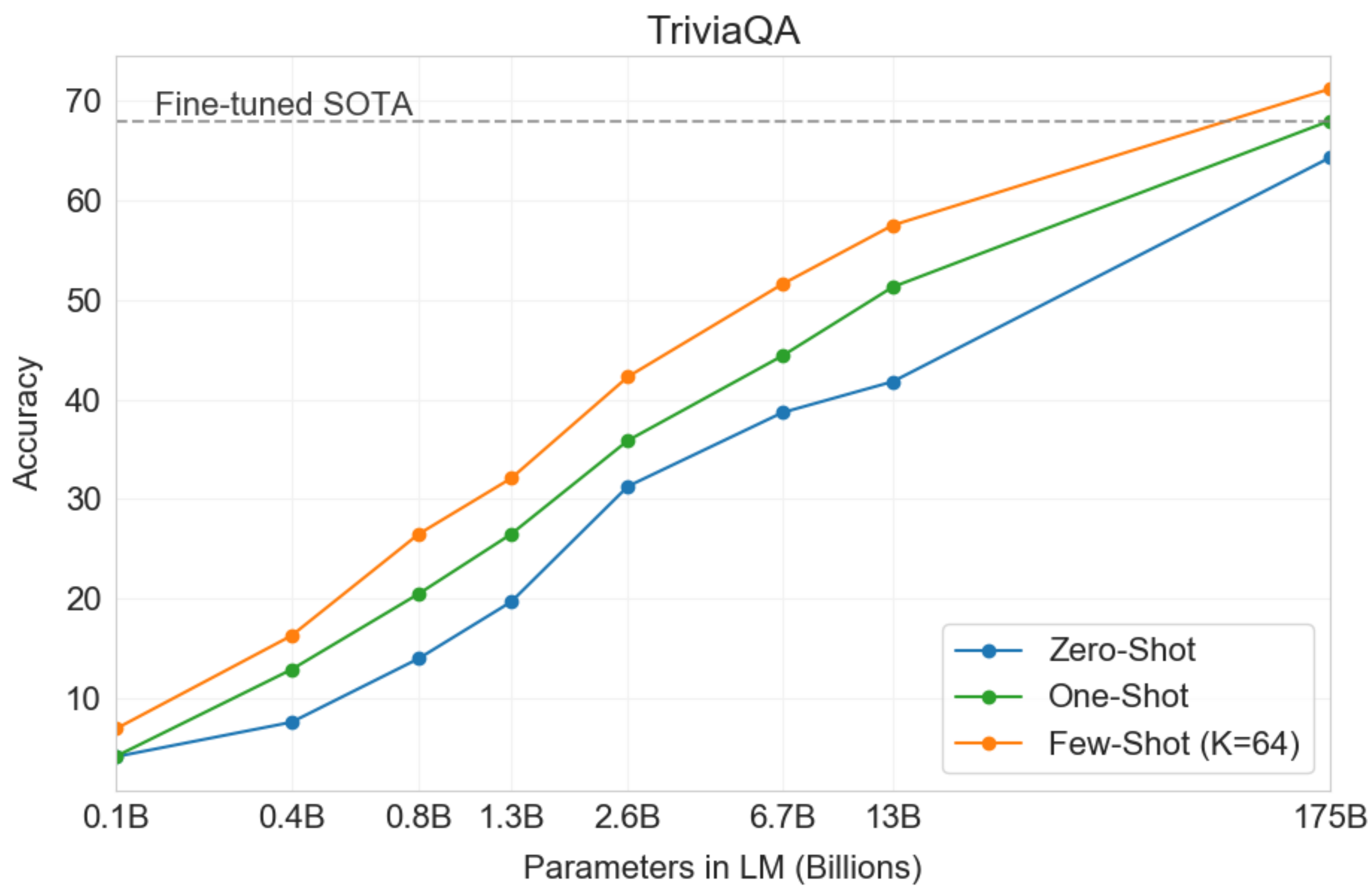


Figure 3.3: On TriviaQA GPT3's performance grows smoothly with model size, suggesting that language models continue to absorb knowledge as their capacity increases. One-shot and few-shot performance make significant gains over zero-shot behavior, matching and exceeding the performance of the SOTA fine-tuned open-domain model, RAG [LPP+20]

5.2.6 BART:
Combining Bidirectional and Auto-Regressive Transformers

BART: Combining Bidirectional and Auto-Regressive Transformers

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. http://arxiv.org/abs/1910.13461

Facebook AI's BART combines Google's BERT and OpenAI's GPT

BERT's bidirectional, **autoencoder** nature is ...

+ good for downstream tasks (e.g., classification) that require info about the whole sequence

- not so good for generation tasks where generated word should only depend on previously generated words

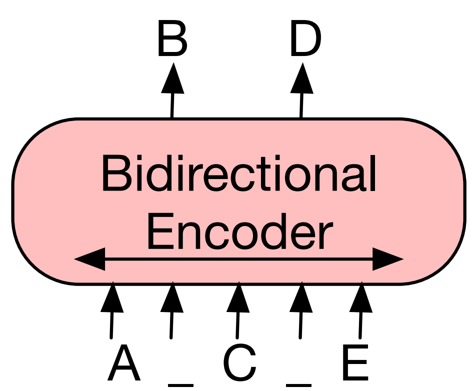
GPT's unidirectional, **autoregressive** approach is ...

+ good for text generation

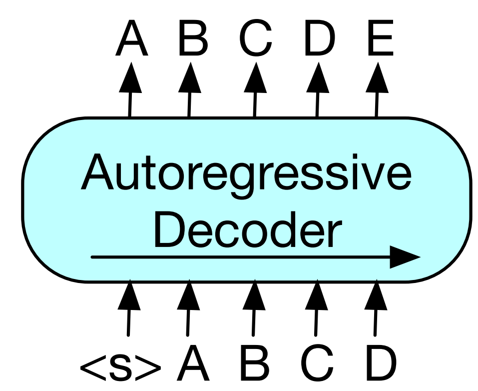
- not so good for tasks that require info of whole sequence, e.g., classification

BART is the best of both worlds

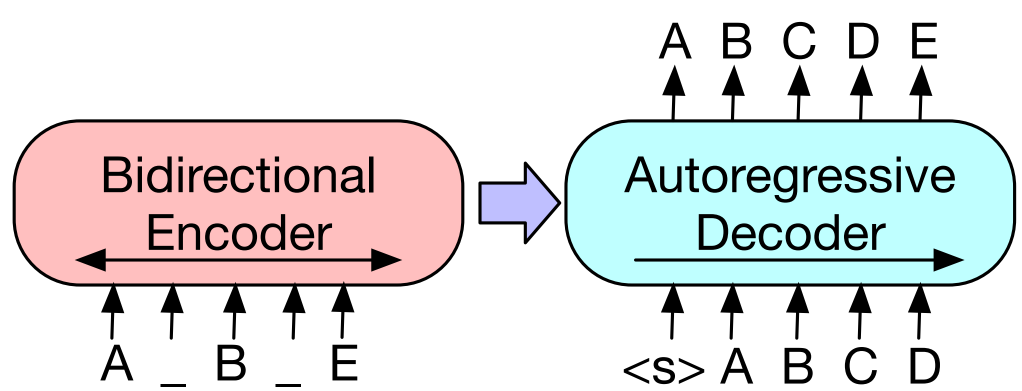
BART: BERT Encoder + GPT Decoder + Noise Transformations



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

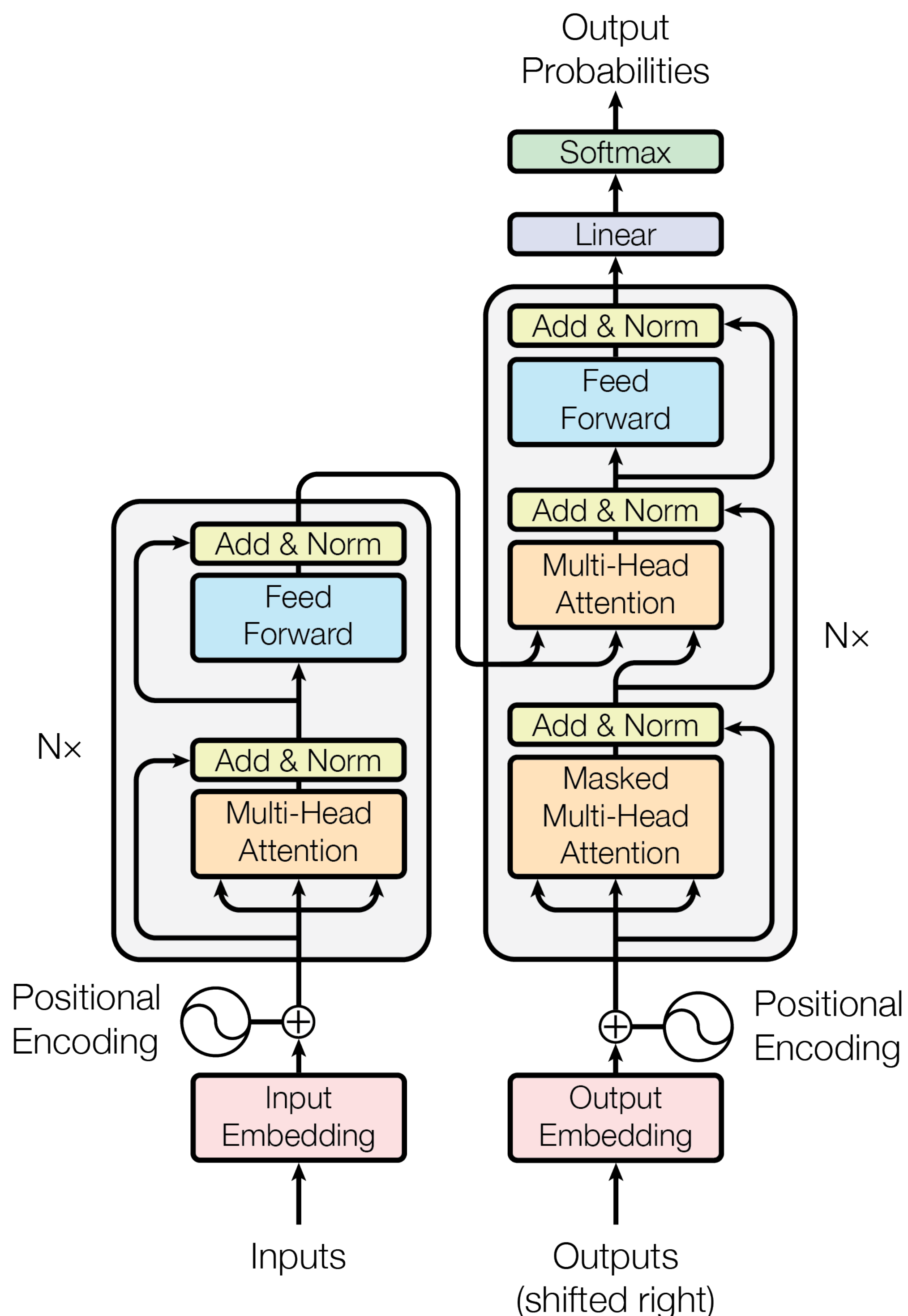


Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Noise Transformations in BART for Pre-Training on Unlabeled Data

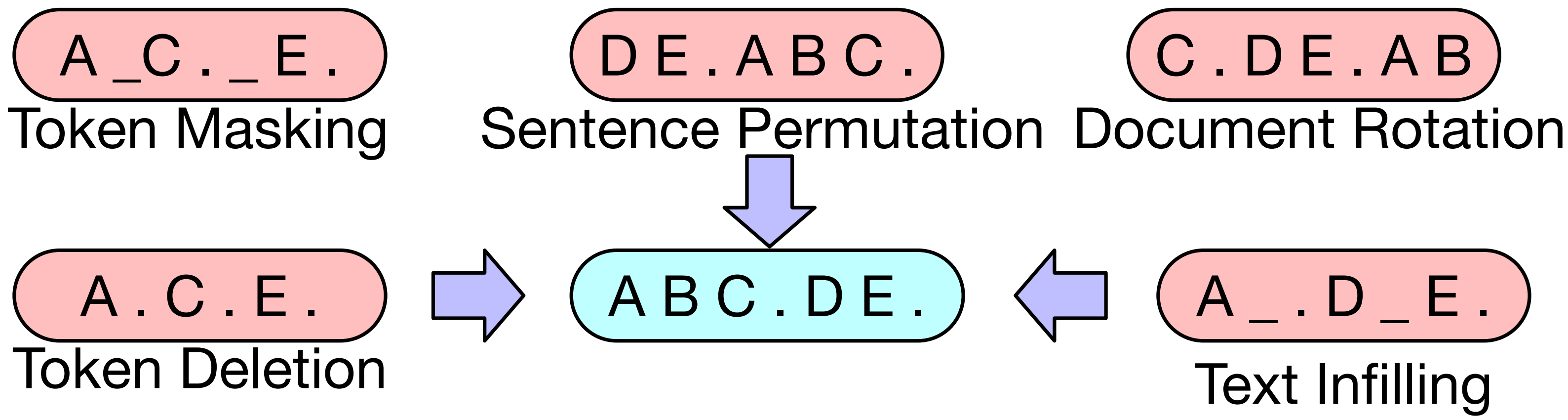


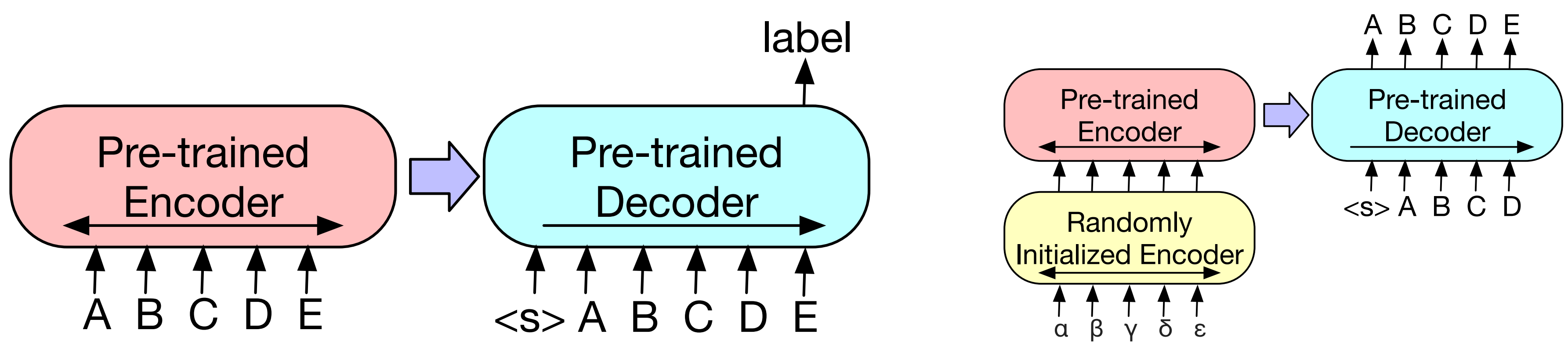
Figure 2: Transformations for noising the input that we experiment with. These transformations can be composed.

Like a denoising autoencoder, it optimizes reconstruction loss

BART Performance Under Different Noise Transformations

Model	SQuAD 1.1 F1	MNLI Acc	ELI5 PPL	XSum PPL	ConvAI2 PPL	CNN/DM PPL
BERT Base (Devlin et al., 2019)	88.5	**84.3**	-	-	-	-
Masked Language Model	90.0	83.5	24.77	7.87	12.59	7.06
Masked Seq2seq	87.0	82.1	23.40	6.80	11.43	6.19
Language Model	76.7	80.1	**21.40**	7.00	11.51	6.56
Permuted Language Model	89.1	83.7	24.03	7.69	12.23	6.96
Multitask Masked Language Model	89.2	82.4	23.73	7.50	12.39	6.74
BART Base						
w/ Token Masking	90.4	84.1	25.05	7.08	11.73	6.10
w/ Token Deletion	90.4	84.1	24.61	6.90	11.46	5.87
w/ Text Infilling	**90.8**	84.0	24.26	**6.61**	**11.05**	5.83
w/ Document Rotation	77.2	75.3	53.69	17.14	19.87	10.59
w/ Sentence Shuffling	85.4	81.5	41.87	10.93	16.67	7.89
w/ Text Infilling + Sentence Shuffling	**90.8**	83.8	24.17	6.62	11.12	**5.41**

Fine-Tuning on Labeled Data



(a) To use BART for classification problems, the same input is fed into the encoder and decoder, and the representation from the final output is used.

(b) For machine translation, we learn a small additional encoder that replaces the word embeddings in BART. The new encoder can use a disjoint vocabulary.

Figure 3: Fine tuning BART for classification and translation.

BART Performance for Discriminative Tasks

	SQuAD 1.1 EM/F1	**SQuAD 2.0** EM/F1	**MNLI** m/mm	**SST** Acc	**QQP** Acc	**QNLI** Acc	**STS-B** Acc	**RTE** Acc	**MRPC** Acc	**CoLA** Mcc
BERT	84.1/90.9	79.0/81.8	86.6/-	93.2	91.3	92.3	90.0	70.4	88.0	60.6
UniLM	-/-	80.5/83.4	87.0/85.9	94.5	-	92.7	-	70.9	-	61.1
XLNet	**89.0**/94.5	86.1/88.8	89.8/-	95.6	91.8	93.9	91.8	83.8	89.2	63.6
RoBERTa	88.9/**94.6**	**86.5/89.4**	**90.2/90.2**	96.4	92.2	94.7	**92.4**	86.6	**90.9**	**68.0**
BART	88.8/**94.6**	86.1/89.2	89.9/90.1	**96.6**	**92.5**	**94.9**	91.2	**87.0**	90.4	62.8

Table 2: Results for large models on SQuAD and GLUE tasks. BART performs comparably to RoBERTa and XLNet, suggesting that BART's uni-directional decoder layers do not reduce performance on discriminative tasks.

BART Performance for Generative Tasks

	CNN/DailyMail			XSum		
	R1	R2	RL	R1	R2	RL
Lead-3	40.42	17.62	36.67	16.30	1.60	11.95
PTGEN (See et al., 2017)	36.44	15.66	33.42	29.70	9.21	23.24
PTGEN+COV (See et al., 2017)	39.53	17.28	36.38	28.10	8.02	21.72
UniLM	43.33	20.21	40.51	-	-	-
BERTSUMABS (Liu & Lapata, 2019)	41.72	19.39	38.76	38.76	16.33	31.15
BERTSUMEXTABS (Liu & Lapata, 2019)	42.13	19.60	39.18	38.81	16.50	31.27
BART	**44.16**	**21.28**	**40.90**	**45.14**	**22.27**	**37.25**

Table 3: Results on two standard summarization datasets. BART outperforms previous work on summarization on two tasks and all metrics, with gains of roughly 6 points on the more abstractive dataset.

	ConvAI2	
	Valid F1	Valid PPL
Seq2Seq + Attention	16.02	35.07
Best System	19.09	17.51
BART	**20.72**	**11.85**

Table 4: BART outperforms previous work on conversational response generation. Perplexities are renormalized based on official tokenizer for ConvAI2.

	ELI5		
	R1	R2	RL
Best Extractive	23.5	3.1	17.5
Language Model	27.8	4.7	23.1
Seq2Seq	28.3	5.1	22.8
Seq2Seq Multitask	28.9	5.4	23.1
BART	**30.6**	**6.2**	**24.3**

Table 5: BART achieves state-of-the-art results on the challenging ELI5 abstractive question answering dataset. Comparison models are from Fan et al. (2019).

5.2.7: Closing Words -- The Recent Growth of Language Transformers

Transformers for Longer Sequences

Transformer-XL:

- encoder-free, decoder-only model
- trained to predict next word in sentence
- uses hidden states to remember previous (512-token) text segment
- "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context", Dai et al. 2019. https://arxiv.org/abs/1901.02860

Longformer:

- instead of attention mechanism that scales quadratically, uses attention mechanism that scales linearly with sequence length
- uses extremely long text segments (thousands of tokens); similar to RoBERTa
- "Longformer: The Long-Document Transformer", Beltagy et al. 2020. https://arxiv.org/abs/2004.05150

GPT-3
(175 billion)

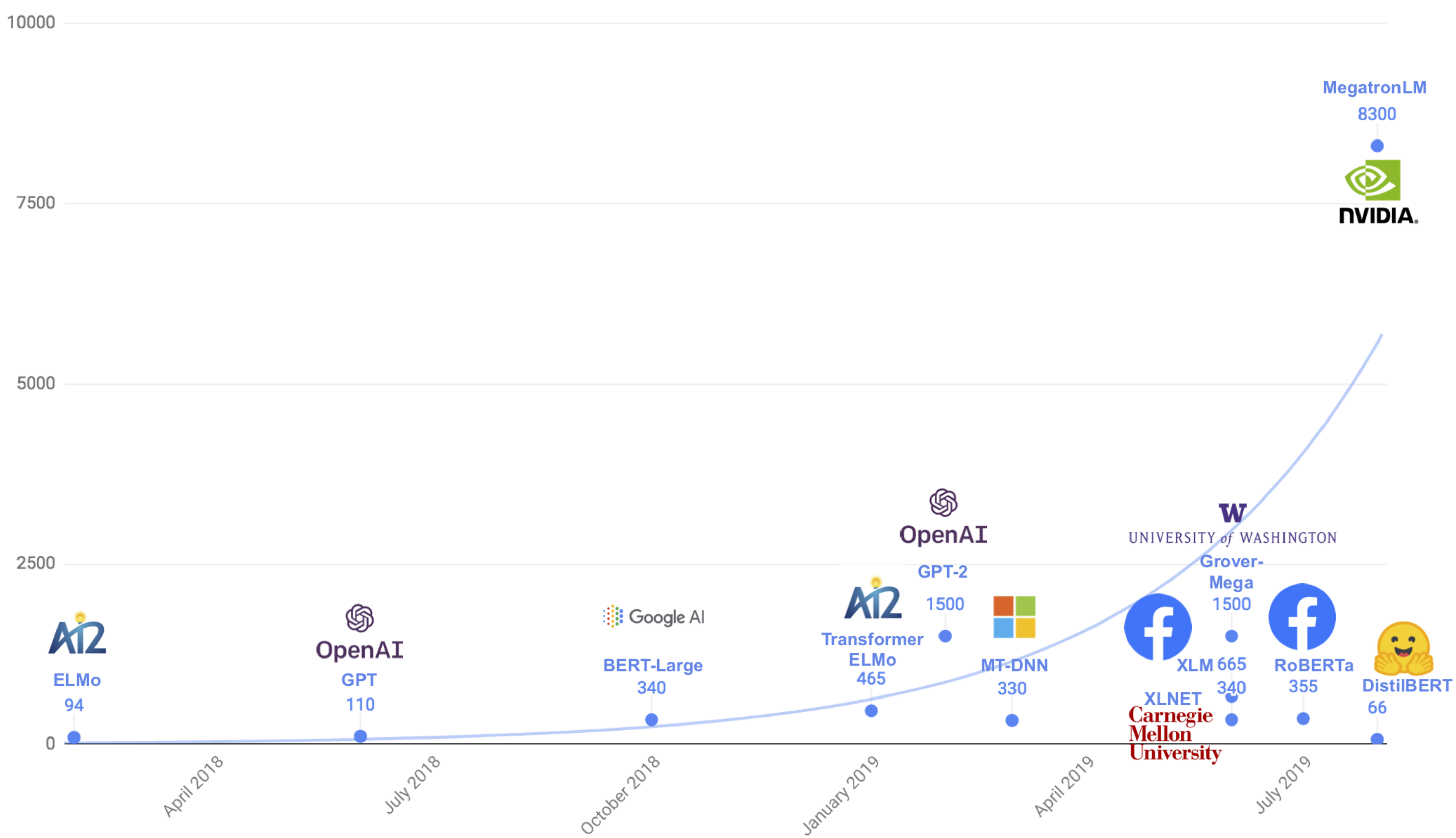


Image Source: https://medium.com/huggingface/distilbert-8cf3380435b5

TECH ARTIFICIAL INTELLIGENCE

OpenAI's text-generating system GPT-3 is now spewing out 4.5 billion words a day

Robot-generated writing looks set to be the next big thing

By James Vincent | Mar 29, 2021, 8:24am EDT

https://www.theverge.com/2021/3/29/22356180/openai-gpt-3-text-generation-words-day

THE COST OF TRAINING NLP MODELS
A CONCISE OVERVIEW

Or Sharir
AI21 Labs
ors@ai21.com

Barak Peleg
AI21 Labs
barakp@ai21.com

Yoav Shoham
AI21 Labs
yoavs@ai21.com

April 2020

http://arxiv.org/abs/2004.08900

Costs: Not for the faint hearted

- $2.5k - $50k (110 million parameter model)
- $10k - $200k (340 million parameter model)
- $80k - $1.6m (1.5 billion parameter model)

Transformers for Better Efficiency

Reformer:

- dot-product attention is replaced with locality-sensitive hashing (LSH) attention
- this achieves attention with $O(nlog(n))$ instead of $O(n^2)$ memory cost
  Kitaev, N., Kaiser, Ł. and Levskaya, A., 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*. https://arxiv.org/abs/2001.04451

ALBERT:

- 5x smaller size as BERT at same performance, due to compression via pruning
- Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P. and Soricut, R., 2019. Albert: A lite BERT for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942. https://arxiv.org/abs/1909.11942

1. Sequence Generation with RNNs
2. Character RNN in PyTorch
3. RNNs with Attention
4. Attention is All We Need
   4.1. Basic Form of Self-Attention
   4.2. Self-Attention & Scaled Dot-Product Attention
   4.3. Multi-Head Attention
5. Transformer Models
   5.1. The Transformer Architecture
   5.2. Some Popular Transformer Models: BERT, GPT, and BART
6. **Transformer in PyTorch**