Transformers

DL4DS – Spring 2024

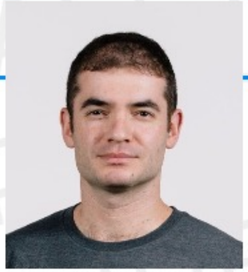
A Brief History of Transformers

2000

Yoshua Bengio*

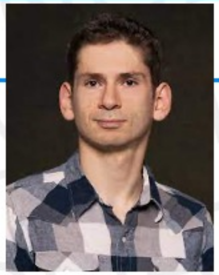
2014

Ilya Sutskever*



2014

Dzmitry Bahdanau*



2017

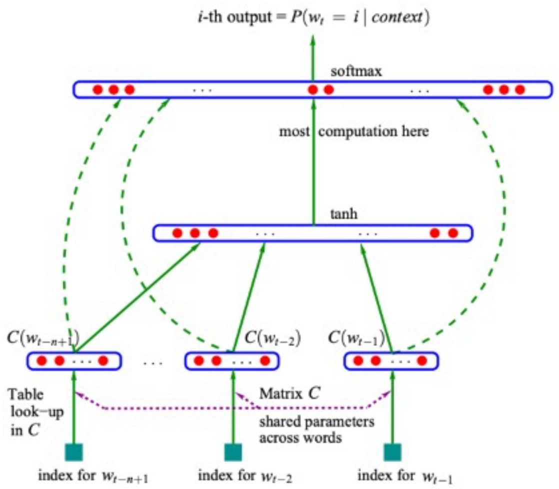
A Team at Google

Use LSTMs

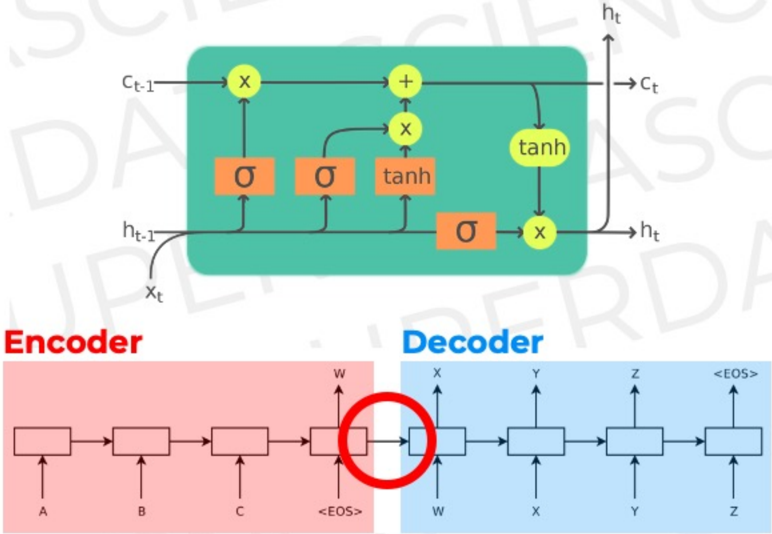
Add Attention

Remove LSTMs

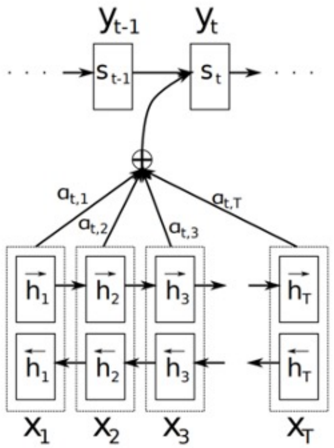
A Neural Probabilistic Language Model



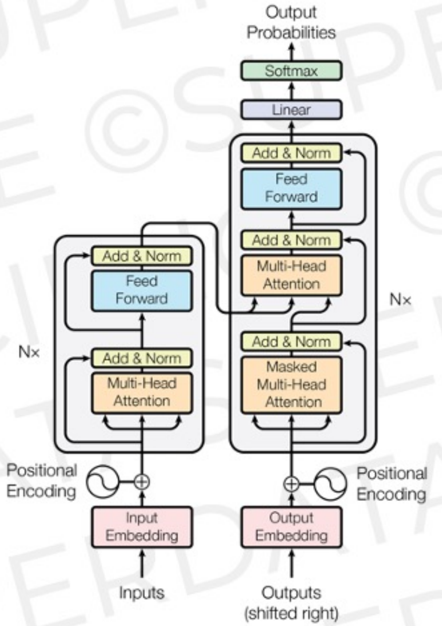
Seq-to-Seq Learning with Neural Networks



Neural Machine Translation by Jointly Learning to Align and Translate



Attention is all you need

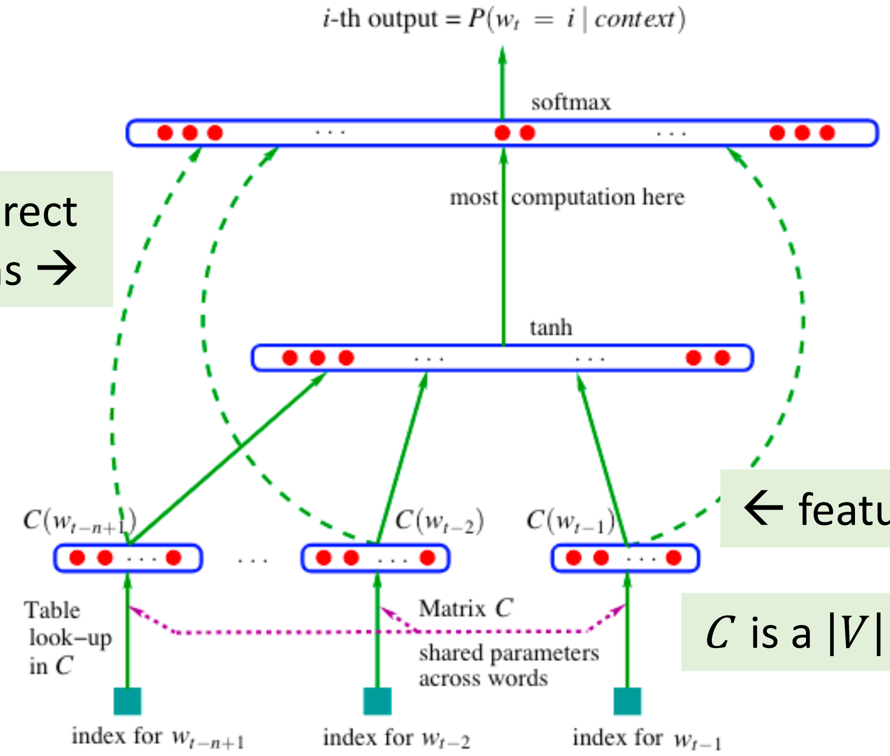


*And others; Chronological analysis inspired by Andrej Karpathy's lecture, youtube.com/watch?v=XfpMkf4rD6E

A Neural Probabilistic Language Model

Bengio et al, 2000 and 2003

Optional direct connections →



← feature vectors, $C(w_t)$

C is a $|V| \times m$ matrix

Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

$w_t \in V$ words in the vocabulary

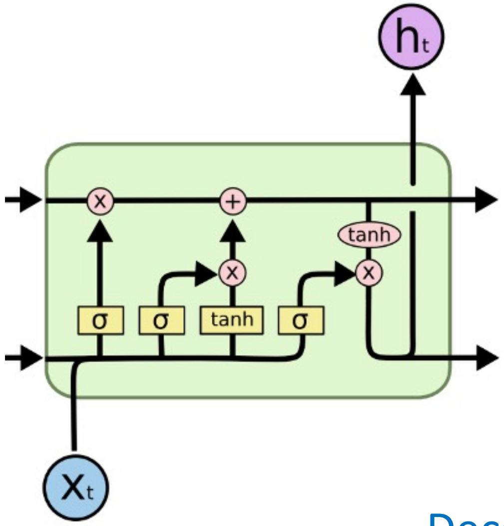
- Build a probabilistic language model from NNs
- Feed forward network with shared parameters, C, that create embeddings
- Predicts the probability of a word at time t, based on the context of the last n words
- Can use shallow feed forward or recurrent neural networks

Limited to context length of n

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, “A Neural Probabilistic Language Model,” *Journal of Machine Learning Research*, vol. 3, pp. 1137-- 1155, Feb. 2003.

Sequence to Sequence Learning with Neural Networks

Sutskever et al (2014)

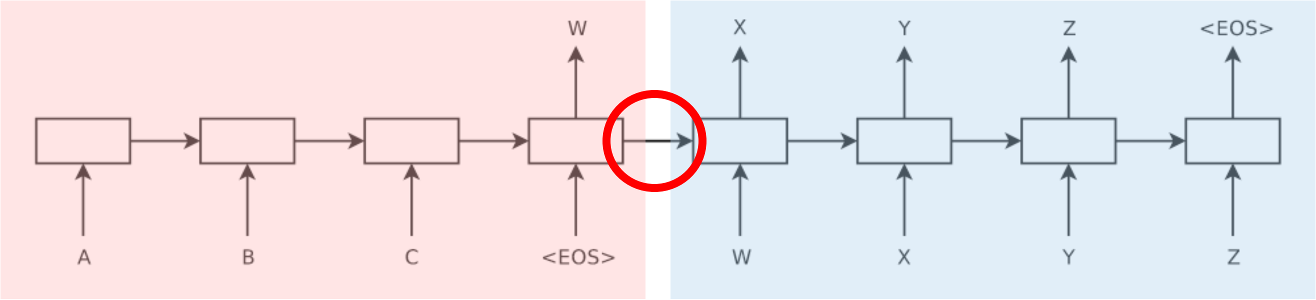


Use LSTMs

- Used LSTMs in an Encoder/Decoder structure
- Estimate the probability of $p(y_1, \ldots, y_{T'} \,|x_1, \ldots, x_T)$ where $T' \neq T$
- Encoder mapped sequence to a fixed size token (hidden state)
- The hidden state may not encode all the information needed by the decoder

Encoder

Decoder



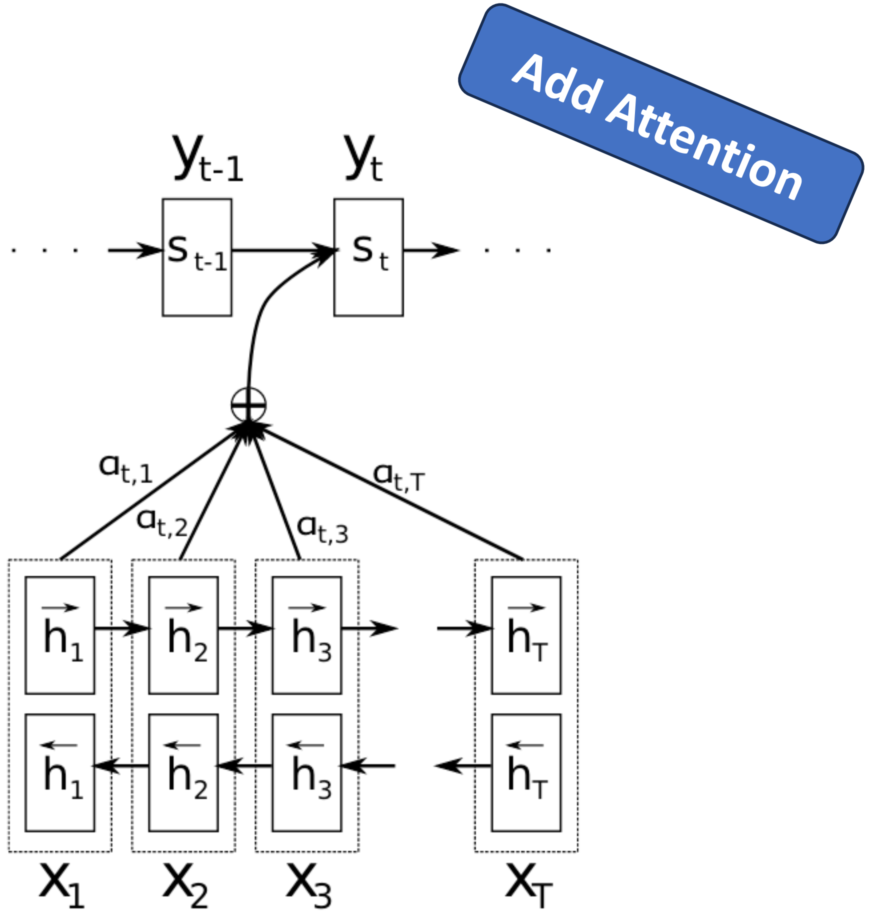
Bottleneck

Bottleneck between Encoder and Decoder!

I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2014. Link

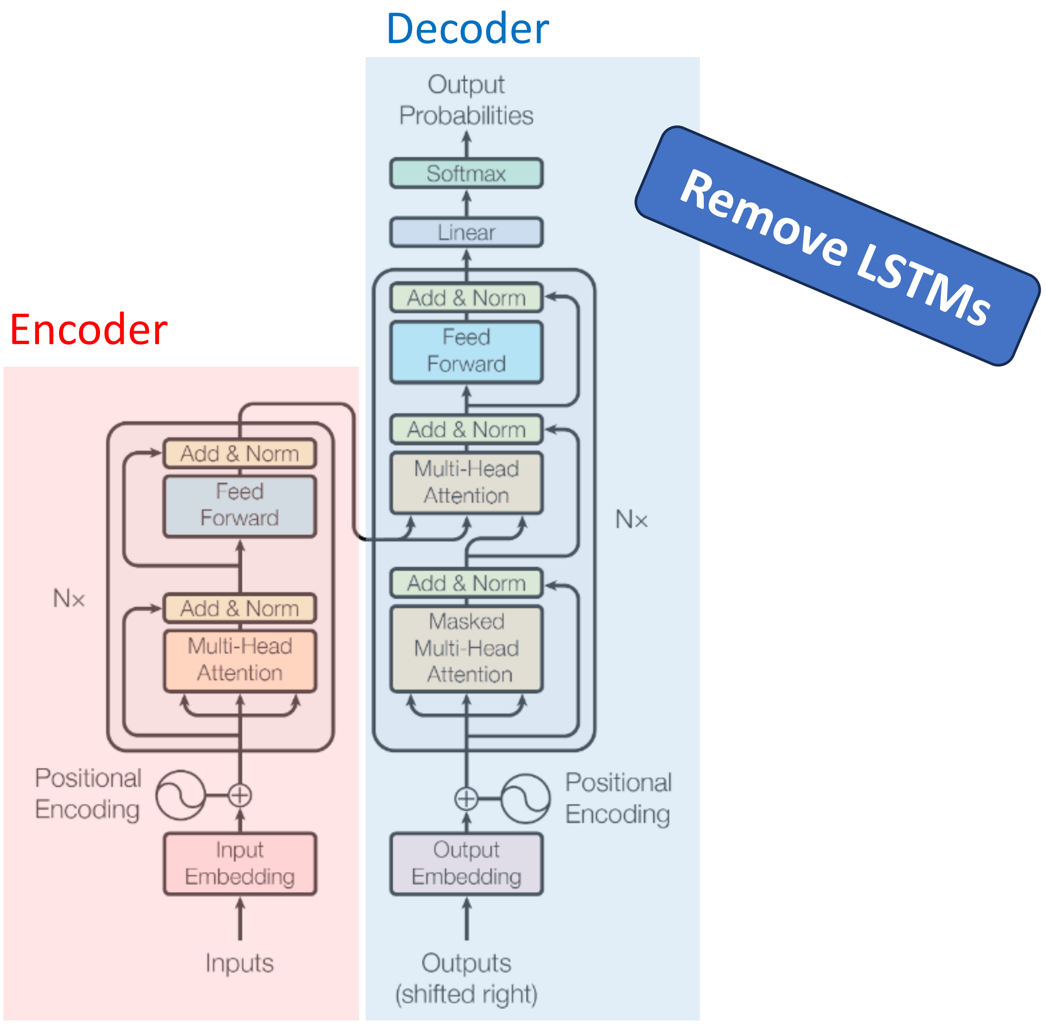
Neural Machine Translation by Jointly Learning to Align and Translate

Bahdanau, Cho & Bengio (2014-15)



- Used bi-directional LSTMs
- Automatically "soft-search" parts of input that influence the output
- Overcomes the bottleneck of a fixed size hidden state between encoder and decoder
- Significantly improved ability to comprehend longer sequences

Attention is All You Need
Vaswani et al (2017)



- Removed LSTMs and didn't use convolutions
- Only attention mechanisms and MLPs
- Parallelizable by removing sequential hidden state computation
- Outperformed all previous models

Transformers applied to many NLP applications

- Translation
- Question answering
- Summarizing
- Generating new text
- Correcting spelling and grammar
- Finding entities
- Classifying bodies of text
- Changing style etc.

Transformers

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models

Transformers

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models

Motivation

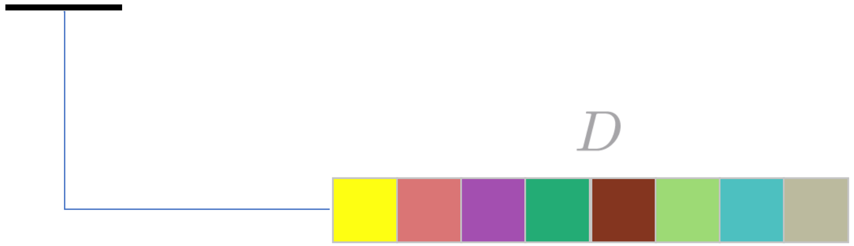
Design neural network to encode and process text:

The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.

Motivation

Design neural network to encode and process text:

The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.



Encode word (or word parts) in some kind of D-dimensional embedding vector.

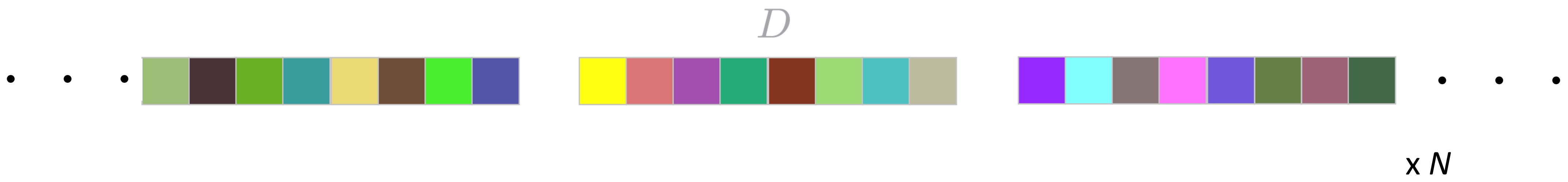
We'll look at tokenization and embedding encoding later.

For now assume a word is a token.

Motivation

Design neural network to encode and process text:

The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.

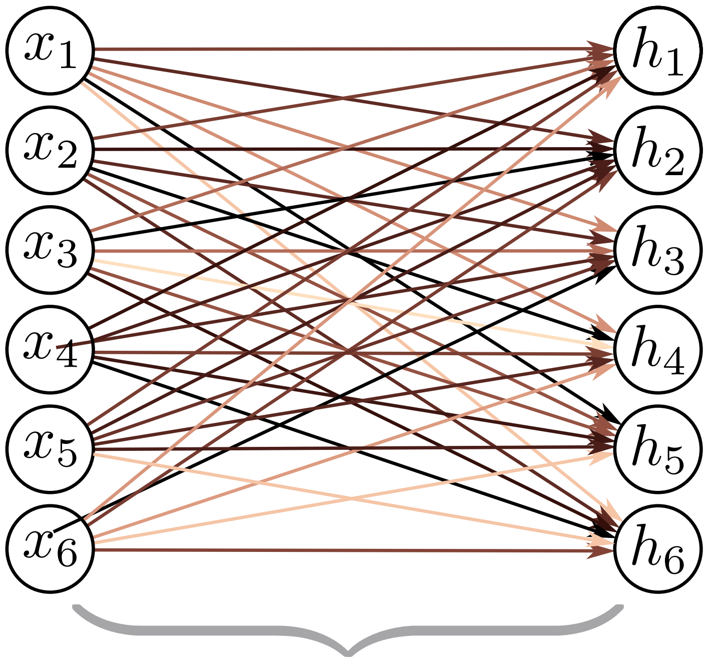


In this example, we have a D-dimensional input vector for each of the 37 words above.

Normally we would represent punctuation, capitalization, spaces, etc. as well.

Standard fully-connected layer

$$\mathbf{h} = \mathbf{a}[\boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{x}]$$



$\boldsymbol{\Phi}$ contains
D^2 connections

Assuming D inputs and
D hidden units.

Standard fully-connected layer

$$\mathbf{h} = \mathbf{a}[\boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{x}]$$

Problem:

- token (word) vectors may be 512 or 1024 dimensional
- need to process large segment of text
- Hence, would require a very large number of parameters
- Can’t cope with text of different lengths

Conclusion:

- We need a model where parameters don’t increase with input length

Motivation

Design neural network to encode and process text:

The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.

The word their must "attend to" the word restaurant.

Motivation

Design neural network to encode and process text:

The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.

The word their must "attend to" the word restaurant.

Conclusions:

- There must be connections between the words.
- The strength of these connections will depend on the words themselves.

Motivation

- Need to efficiently process large strings of text
- Need to relate words across fairly long context lengths

Self-Attention addresses these problems

Transformers

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models

Dot-Product Self-Attention

1. Shares parameters to cope with long input passages of different lengths

2. Contains connections between word representations that depend on the words themselves

Dot-product self attention

- Takes N inputs of size Dx1 and returns N inputs of size Dx1
- Computes N values (no ReLU)

$$\mathbf{v}_n = \boldsymbol{\beta}_v + \boldsymbol{\Omega}_v \mathbf{x}_n$$

Dot-product self attention

- Takes N inputs of size Dx1 and returns N inputs of size Dx1
- Computes N values (no ReLU)

$$\mathbf{v}_n = \boldsymbol{\beta}_v + \boldsymbol{\Omega}_v \mathbf{x}_n$$

- N outputs are weighted sums of these values

$$\mathbf{sa}[\mathbf{x}_n] = \sum_{m=1}^{N} a[\mathbf{x}_n, \mathbf{x}_m]\mathbf{v}_m$$

Dot-product self attention

- Takes N inputs of size Dx1 and returns N inputs of size Dx1
- Computes N values (no ReLU)

$$\mathbf{v}_n = \boldsymbol{\beta}_v + \boldsymbol{\Omega}_v \mathbf{x}_n$$

- N outputs are weighted sums of these values

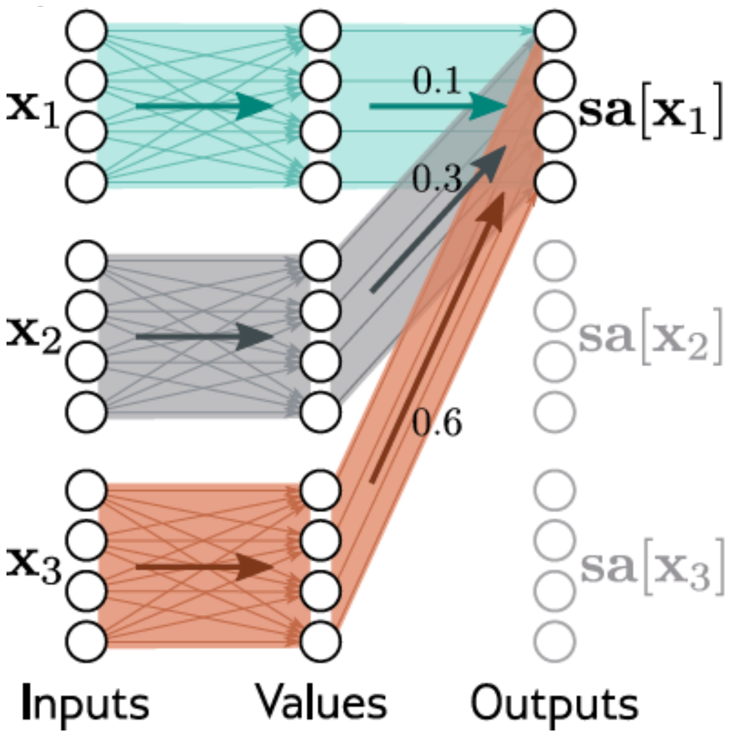
Scalar self-attention weights that represent how much attention the n^{th} token should pay to the m^{th} token

$$\mathbf{sa}_n[\mathbf{x}_1, \ldots, \mathbf{x}_N] = \sum_{m=1}^{N} \overbrace{a[\mathbf{x}_m, \mathbf{x}_n]}\mathbf{v}_m.$$

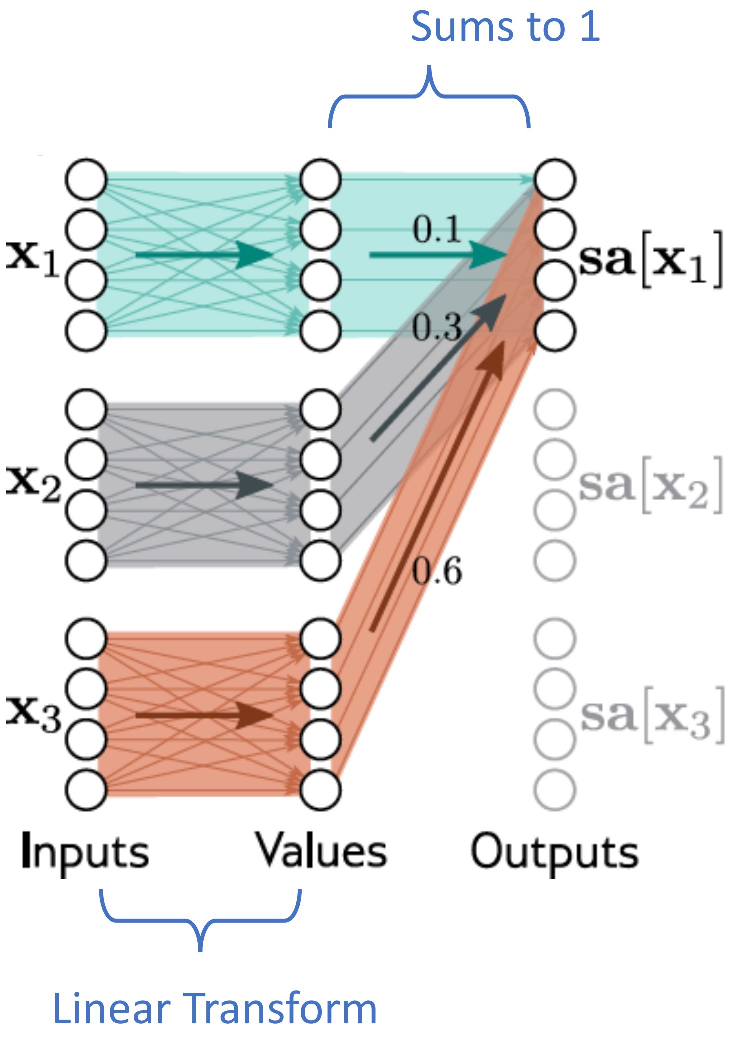
- Weights depend on the inputs themselves

$a[\cdot, \mathbf{x}_n]$ are non-negative and sum to one

Attention as routing



Attention as routing



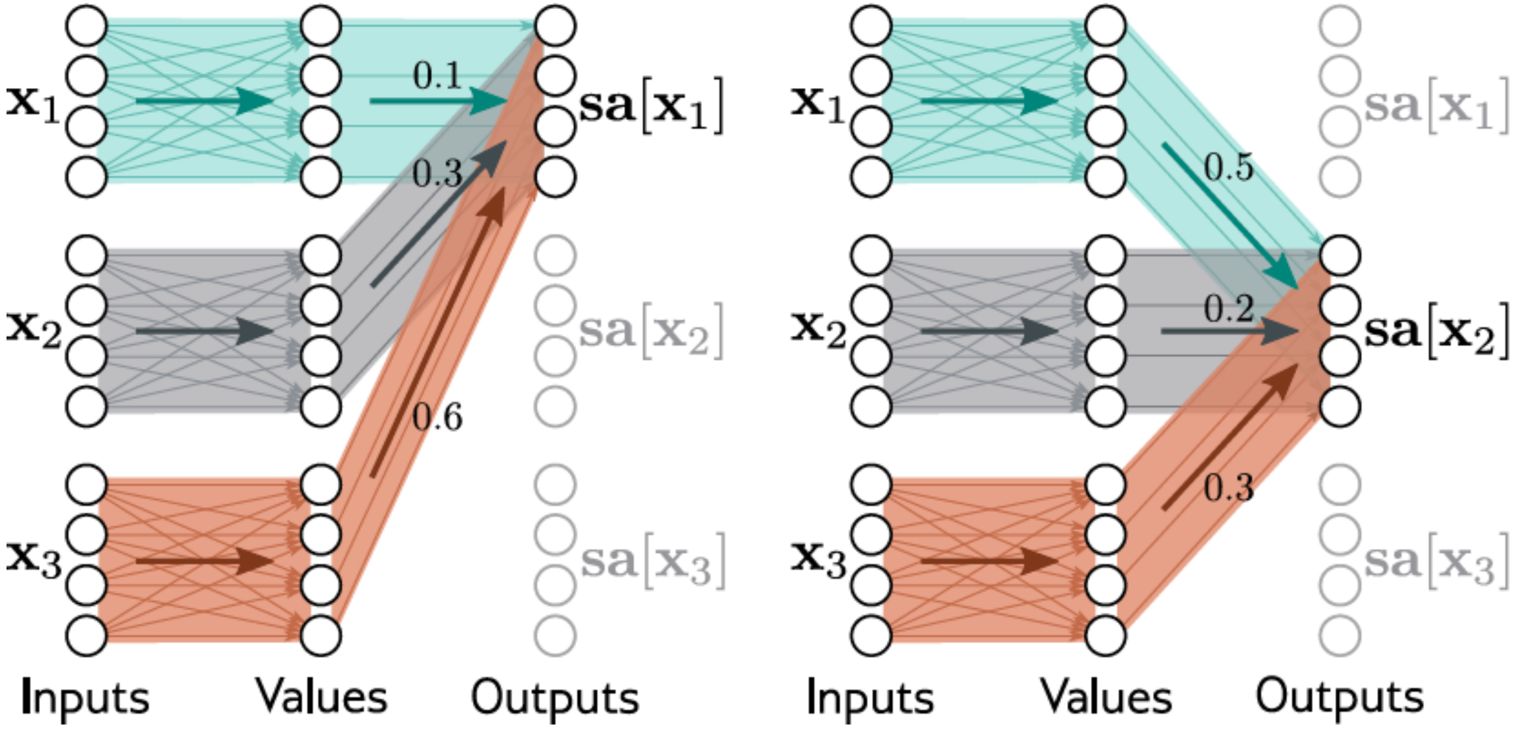
Here:

# of inputs, N = 3

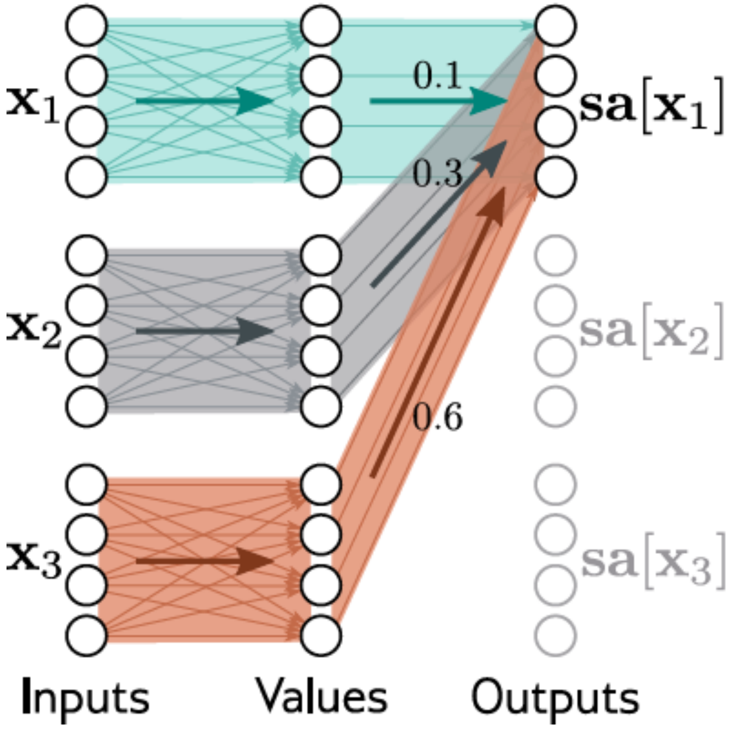
Dimension of each input, D = 4

We'll show how to calculate the self-attention weights shortly.

Attention as routing



Attention as routing



Attention weights

- Compute N "queries" and N "keys" from input

$$\mathbf{q}_n = \boldsymbol{\beta}_q + \boldsymbol{\Omega}_q \mathbf{x}_n$$
$$\mathbf{k}_n = \boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{x}_n,$$

- Calculate similarity and pass through softmax:

$$\begin{aligned} a[\mathbf{x}_n, \mathbf{x}_m] &= \text{softmax}_m \left[\text{sim}[\mathbf{k}_m \mathbf{q}_n]\right] \\ &= \frac{\exp\left[\text{sim}[\mathbf{k}_m \mathbf{q}_n]\right]}{\sum_{m'=1}^{N} \exp\left[\text{sim}[\mathbf{k}'_m \mathbf{q}_n]\right]}, \end{aligned}$$

Attention weights

- Compute N "queries" and N "keys" from input

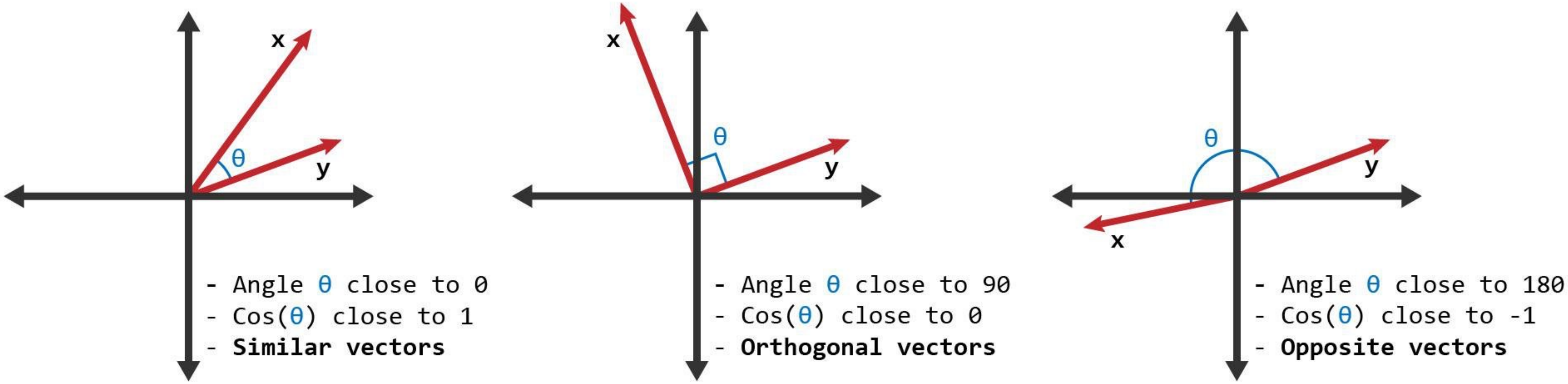
$$\mathbf{q}_n = \boldsymbol{\beta}_q + \boldsymbol{\Omega}_q \mathbf{x}_n$$
$$\mathbf{k}_n = \boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{x}_n,$$

- Take dot products and pass through softmax:

$$\begin{aligned} a[\mathbf{x}_n, \mathbf{x}_m] &= \mathrm{softmax}_m \left[\mathbf{k}_m^T \mathbf{q}_n\right] \\ &= \frac{\exp\left[\mathbf{k}_m^T \mathbf{q}_n\right]}{\sum_{m'=1}^{N} \exp\left[\mathbf{k}_{m'}^T \mathbf{q}_n\right]} \end{aligned}$$

Dot product = measure of similarity

$$\mathbf{x}^T\mathbf{y} = |\mathbf{x}||\mathbf{y}|\cos(\theta)$$



A drawback of the dot product as similarity measure is the magnitude of each vector influences the value. More rigorous to divide by magnitudes.

Cosine Similarity: $\frac{\mathbf{x}^T\mathbf{y}}{|\mathbf{x}||\mathbf{y}|} = \cos(\theta)$

Motivation

Design neural network to encode and process text:

The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.

Conclusions:

- ✓ We need a model where parameters don't increase with input length, e.g.

$$\phi = \{\boldsymbol{\beta}_v, \boldsymbol{\Omega}_v, \boldsymbol{\beta}_q, \boldsymbol{\Omega}_q, \boldsymbol{\beta}_k, \boldsymbol{\Omega}_k\}$$

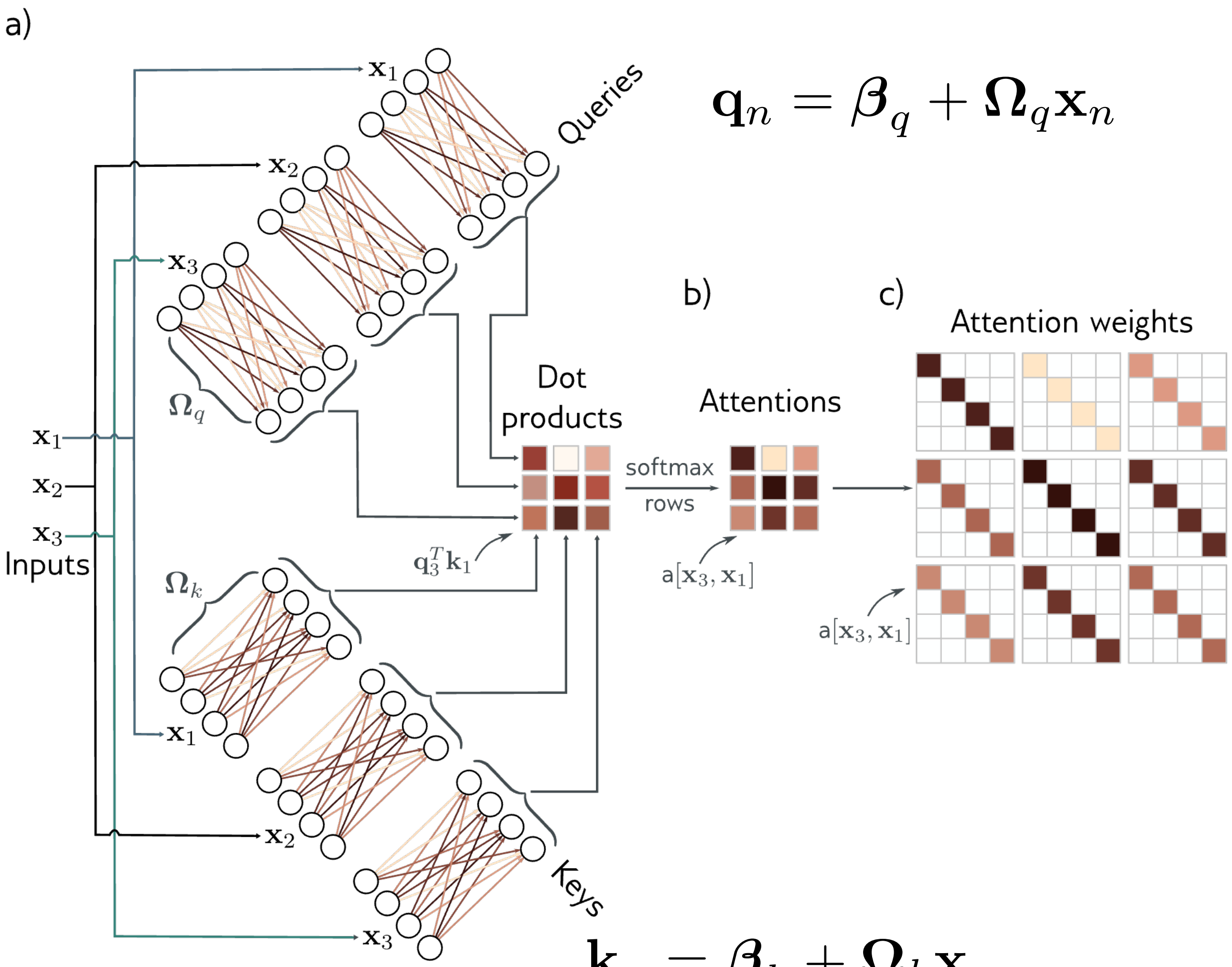
- ✓ There must be connections between the words.
- ✓ The strength of these connections will depend on the words themselves.

Ok, we defined *queries*, *keys* and *values*, but how are they used?

Transformers

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models

Computing Attention Weights

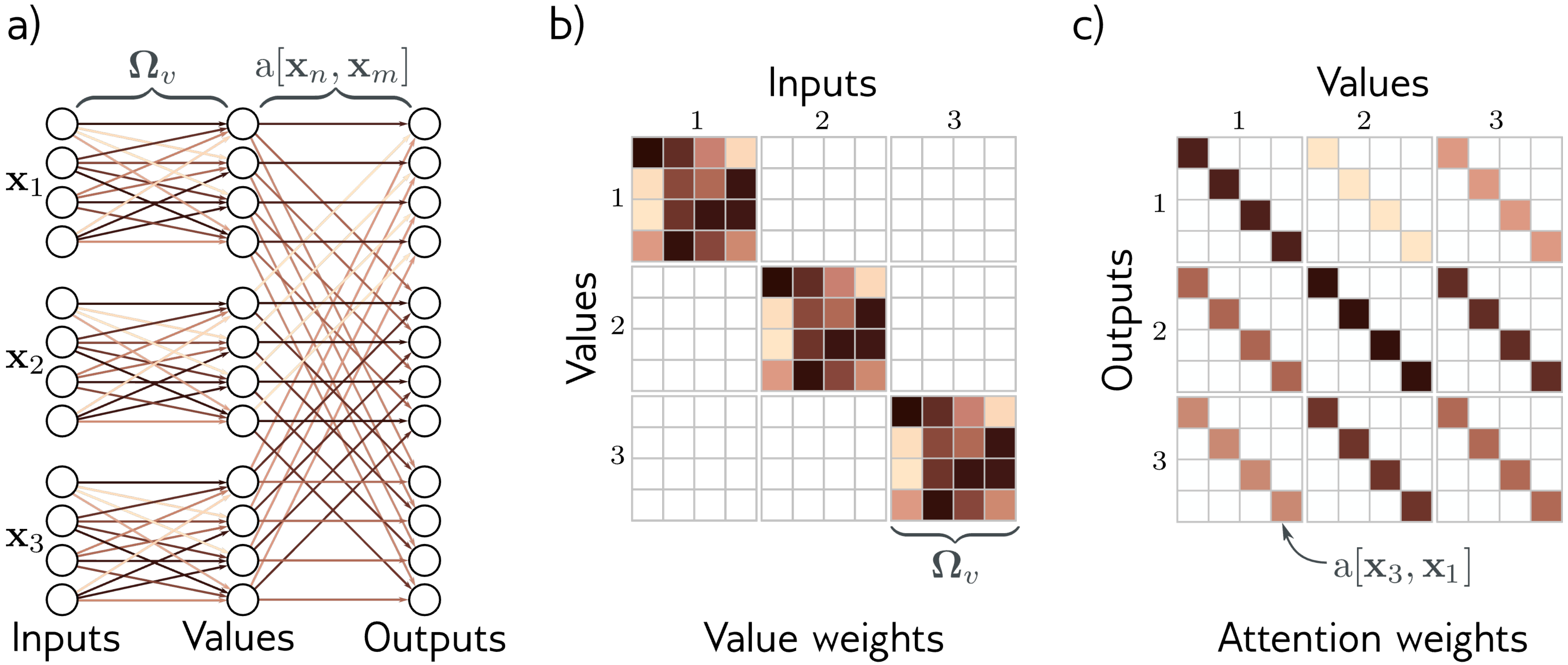


$$\mathbf{q}_n = \boldsymbol{\beta}_q + \mathbf{\Omega}_q \mathbf{x}_n$$

$$a[\mathbf{x}_n, \mathbf{x}_m] = \text{softmax}_m \left[\mathbf{k}_m^T \mathbf{q}_n\right]$$

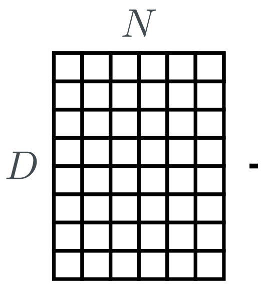
$$\mathbf{k}_n = \boldsymbol{\beta}_k + \mathbf{\Omega}_k \mathbf{x}_n,$$

Computing Values and Self-Attention Outputs as Sparse Matrix Ops



From Input Vector to Input Matrix

- Store N input vectors in matrix X



Input, $\mathbf{X}$

- Compute values, queries and keys:

$$\mathbf{V}[\mathbf{X}] = \boldsymbol{\beta}_v \mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_{\mathbf{v}} \mathbf{X}$$
$$\mathbf{Q}[\mathbf{X}] = \boldsymbol{\beta}_q \mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_{\mathbf{q}} \mathbf{X}$$
$$\mathbf{K}[\mathbf{X}] = \boldsymbol{\beta}_k \mathbf{1}^{\mathbf{T}} + \boldsymbol{\Omega}_{\mathbf{k}} \mathbf{X},$$

- Combine self-attentions

$$\mathbf{Sa}[\mathbf{X}] = \mathbf{V}[\mathbf{X}] \cdot \mathbf{Softmax}\Big[\mathbf{K}[\mathbf{X}]^T \mathbf{Q}[\mathbf{X}]\Big] = \mathbf{V} \cdot \mathbf{Softmax}[\mathbf{K}^T \mathbf{Q}]$$

Scaled Dot Product Self-Attention

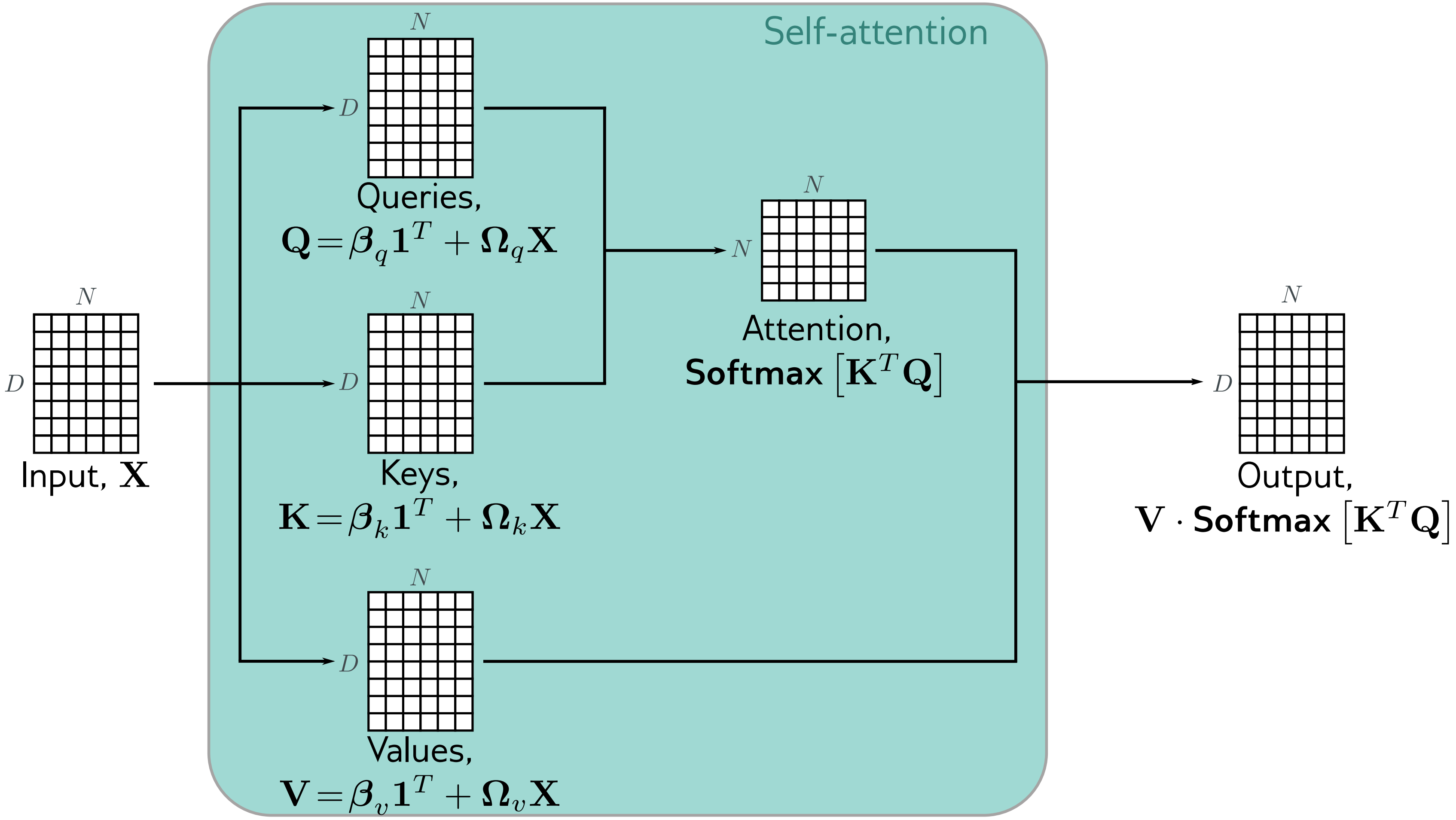
- To avoid the case where a large value dominates the softmax in

$$\mathbf{Sa}[\mathbf{X}] = \mathbf{V} \cdot \mathbf{Softmax}[\mathbf{K}^T\mathbf{Q}]$$

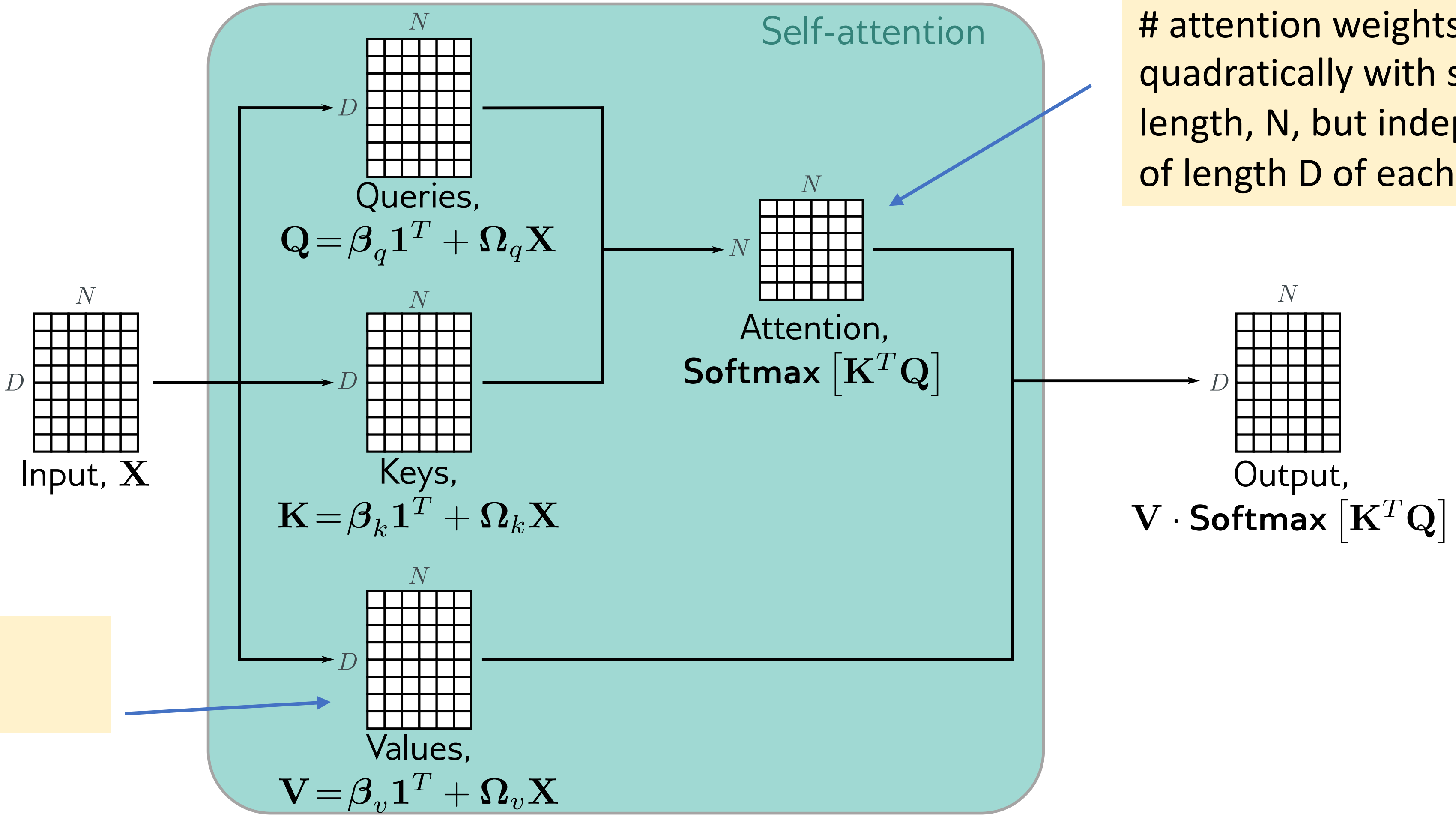
- you can scale the dot product by the square root of the dimension of the query

$$\mathbf{Sa}[\mathbf{X}] = \mathbf{V} \cdot \mathbf{Softmax}\left[\frac{\mathbf{K}^T\mathbf{Q}}{\sqrt{D_q}}\right]$$

Put it all together in matrix form



Put it all together in matrix form

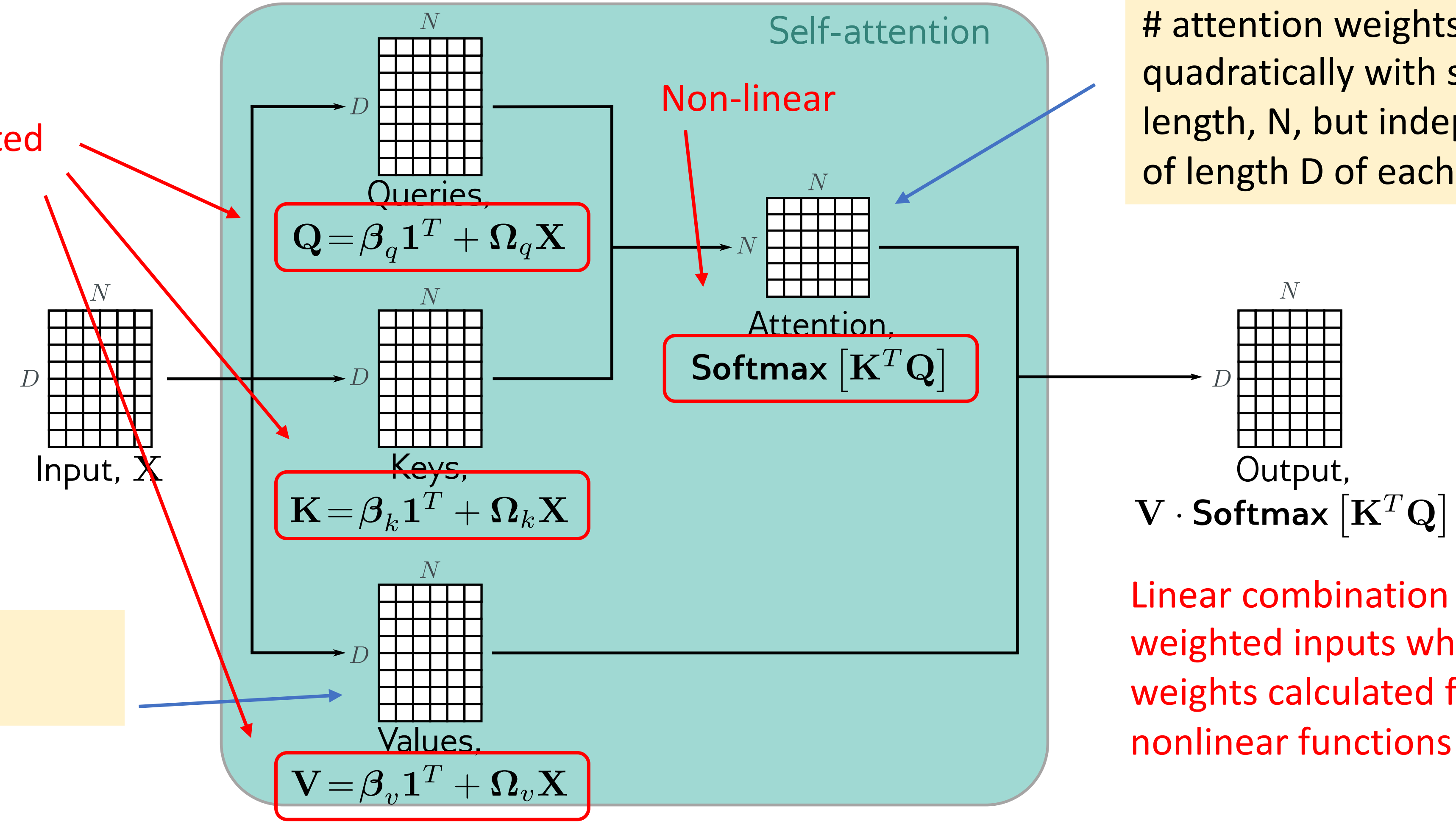


\# attention weights scales quadratically with sequence length, N, but independent of length D of each input

Scales linearly with sequence length, N

Put it all together in matrix form

Linear
&
Can be calculated in parallel



attention weights scales quadratically with sequence length, N, but independent of length D of each input

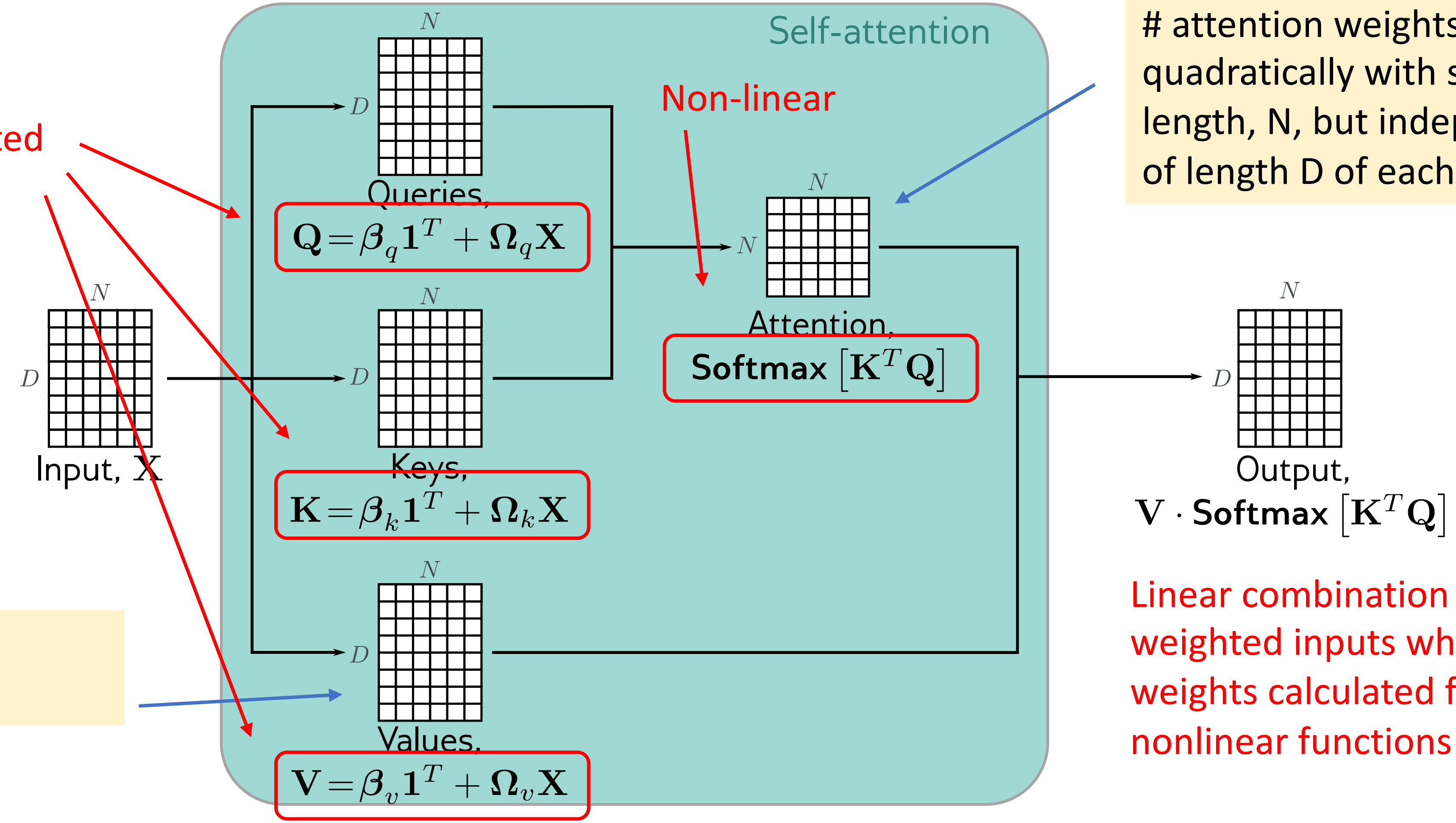
Scales linearly with sequence length, N

Linear combination of weighted inputs where weights calculated from nonlinear functions

Hypernetwork – 1 branch calculates weights of other branch

Linear
&
Can be calculated in parallel

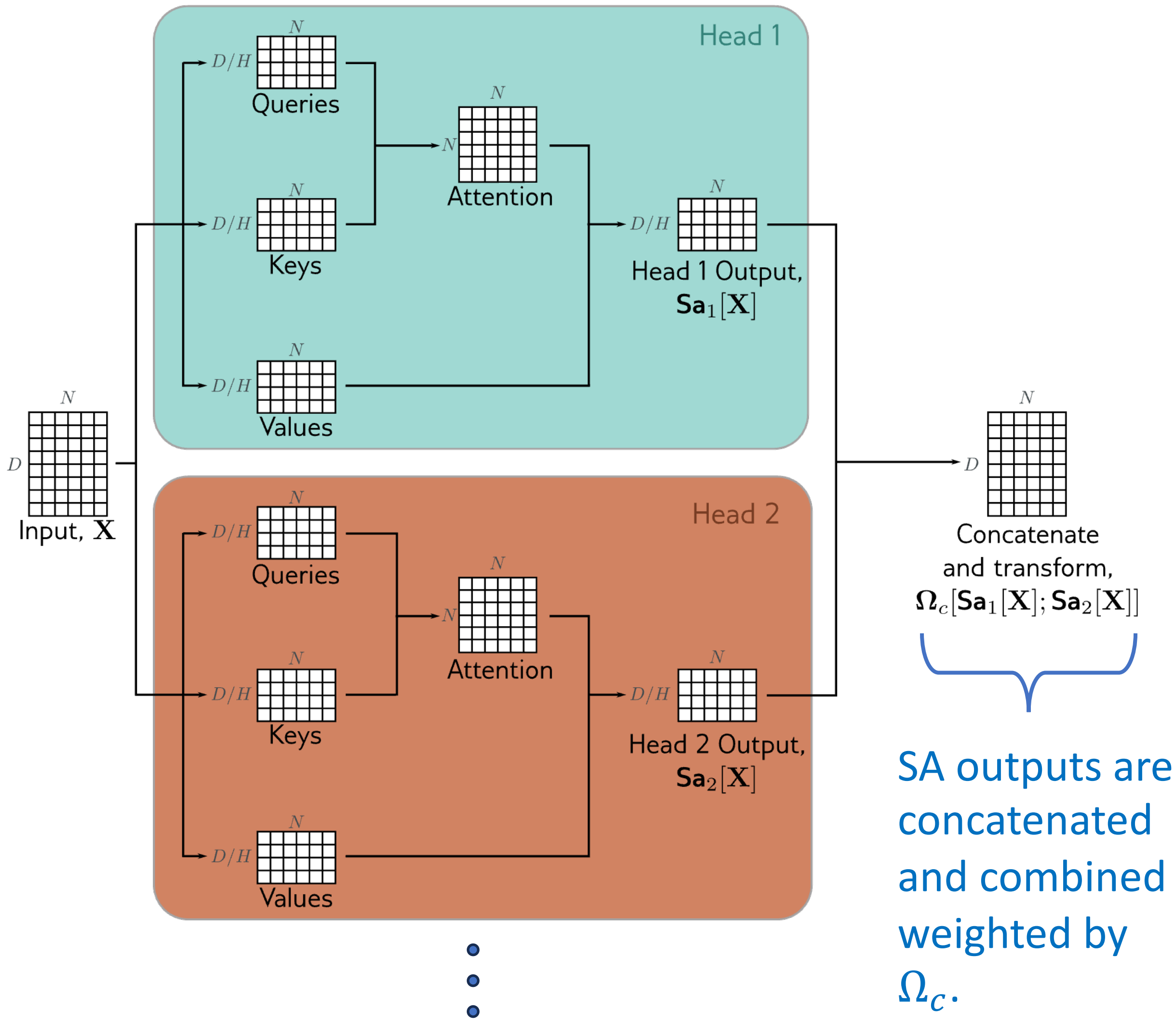
attention weights scales quadratically with sequence length, N, but independent of length D of each input



Scales linearly with sequence length, N

Linear combination of weighted inputs where weights calculated from nonlinear functions

Multi-Head Self Attention



SA outputs are concatenated and combined weighted by Ω_c.

- Multiple self-attention heads are usually applied in parallel
- "allows model to jointly attend to info from different representation subspaces at different positions"
- Original paper used 8 heads
- All can be executed in parallel

Equivariance to Word Order

A function f[x] is equivariant to a transformation t[] if: $\mathbf{f}[\mathbf{t}[\mathbf{x}]] = \mathbf{t}[\mathbf{f}[\mathbf{x}]]$

Self-attention is *equivariant* to permuting word order. Just a bag of words.

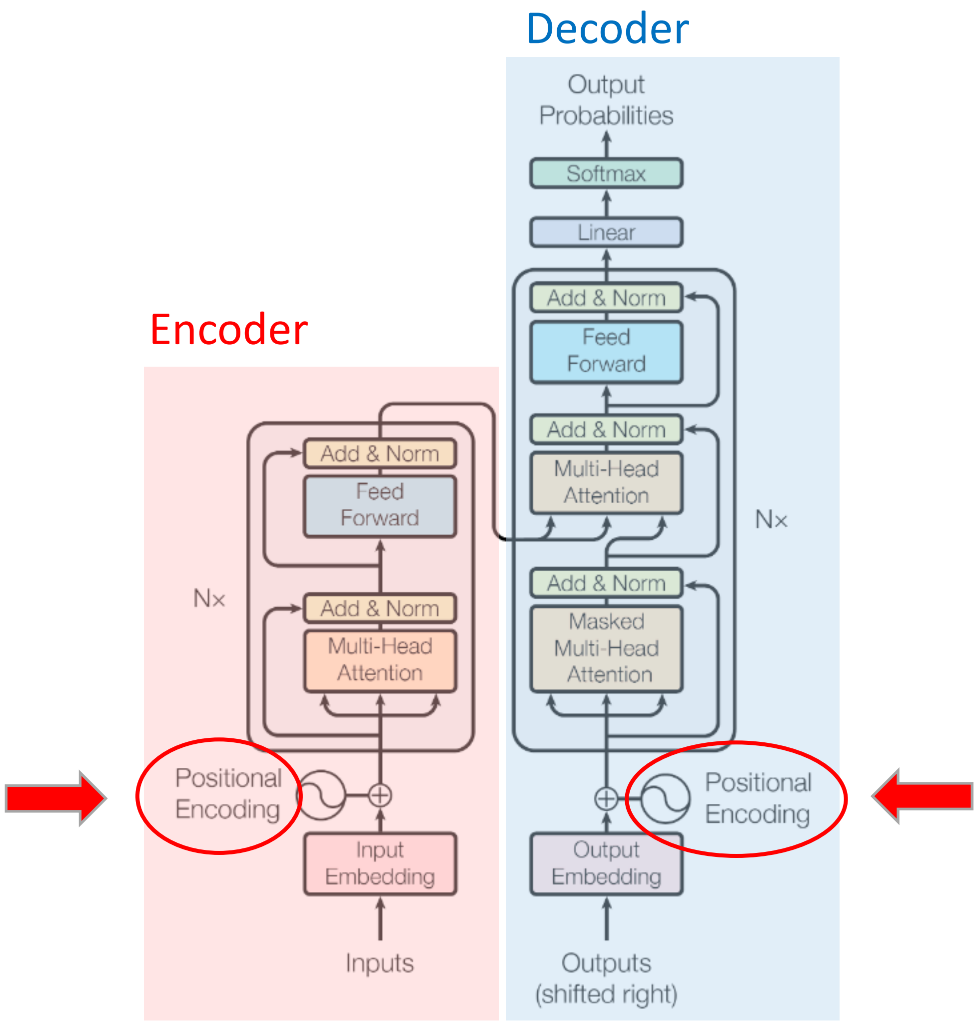
But word order is important in language:

The man ate the fish

vs.

The fish ate the man

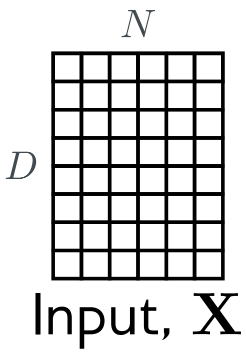
Solution: Position Encoding



Idea is to somehow encode *absolute* or *relative* position in the inputs

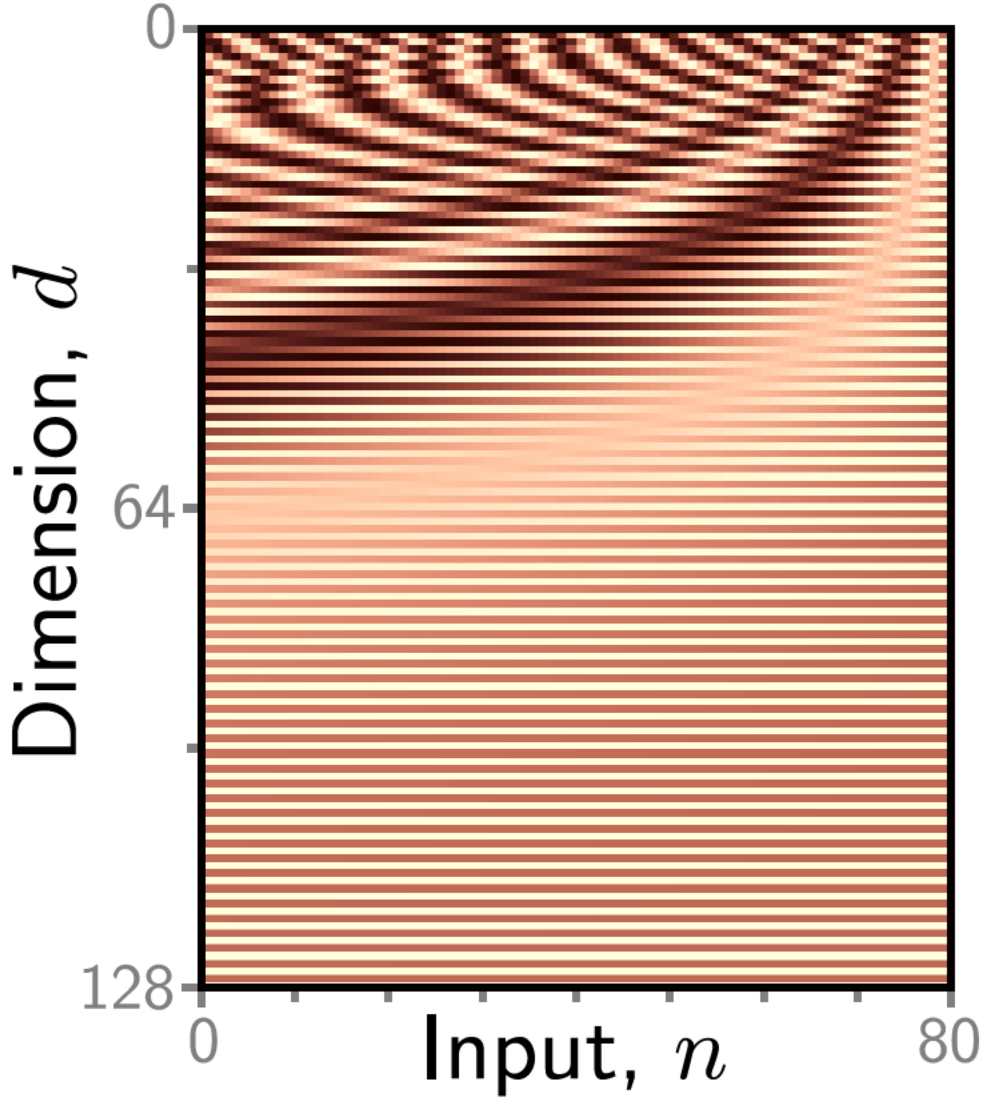
Absolute Position encoding

Add some matrix, Π, to the $D \times N$ input matrix:



$+ \Pi$

$\Pi =$



Π can be pre-defined or learned

Absolute Position encoding

Alternatively, could be added to each layer

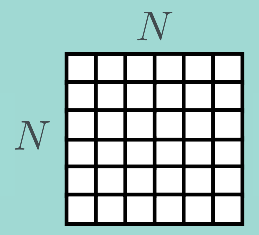
$$\mathbf{Sa}[\mathbf{X}] = \mathbf{V} \cdot \mathbf{Softmax}[\mathbf{K}^T \mathbf{Q}]$$

$$\mathbf{Sa}[\mathbf{X}] = (\mathbf{V} + \mathbf{\Pi}) \cdot \mathbf{Softmax}[(\mathbf{K} + \mathbf{\Pi})^T (\mathbf{Q} + \mathbf{\Pi})]$$

Relative Position Encoding

Absolute position of a word is less important than relative position between inputs

	The	panda	eats	shoots	and	leaves
Abs Pos:	0	1	2	3	4	5
Rel Pos:	-2	-1	0	1	2	3



Attention

Softmax $\left[\mathbf{K}^T\mathbf{Q}\right]$

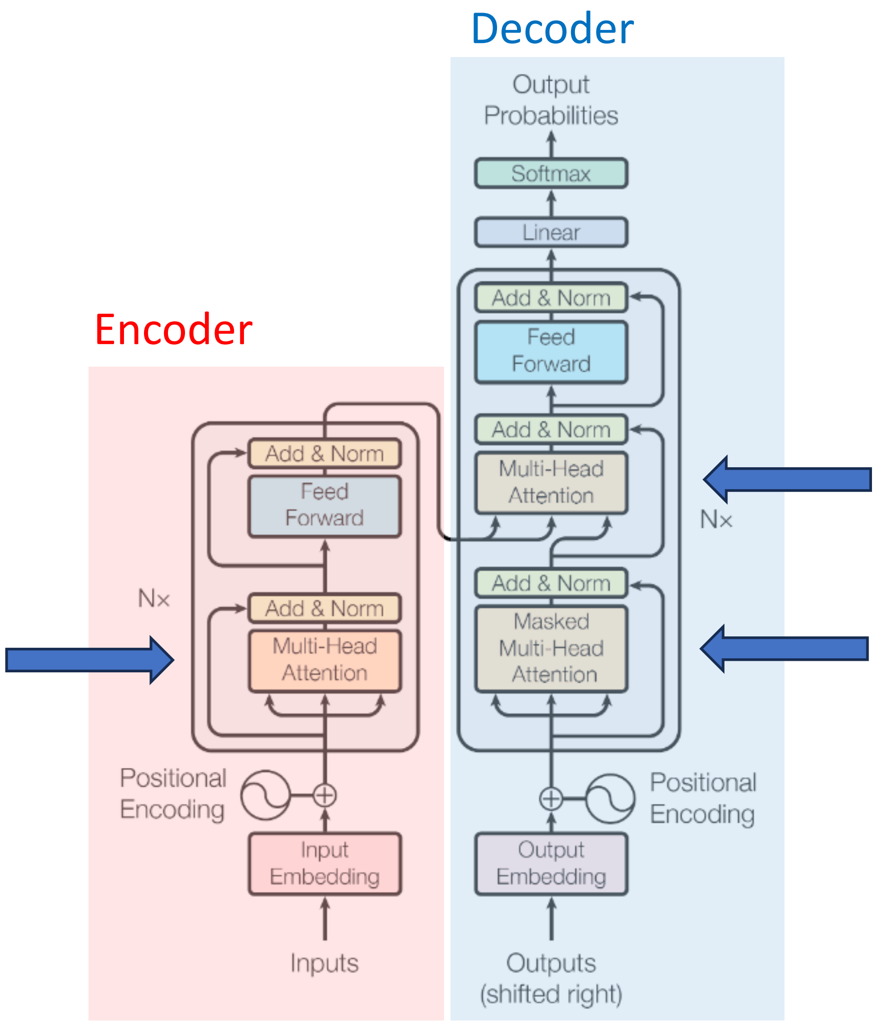
Each element of the attention matrix corresponds to an offset between query position a and key position b

Learn a parameter $\pi_{a,b}$ for each offset and modify Attention[a,b] in some way.

Transformers

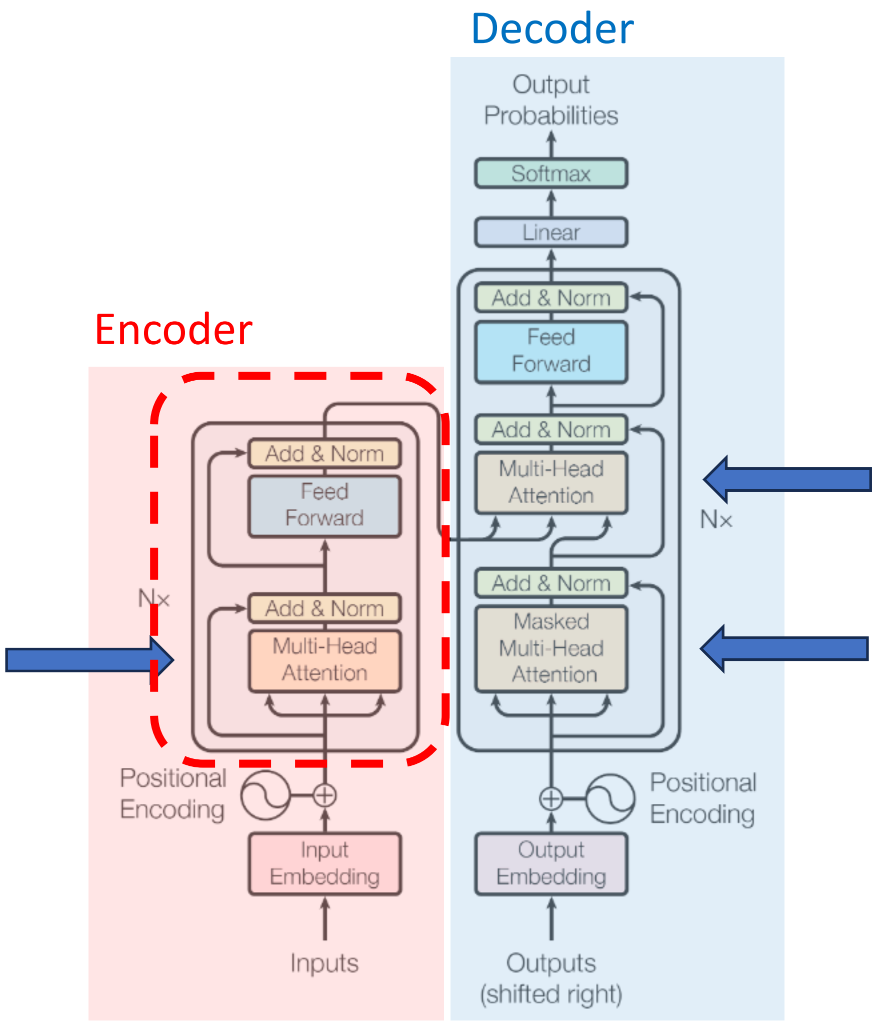
- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models

Transformers



- *Multi-headed Self Attention* is just one component of the transformer architecture

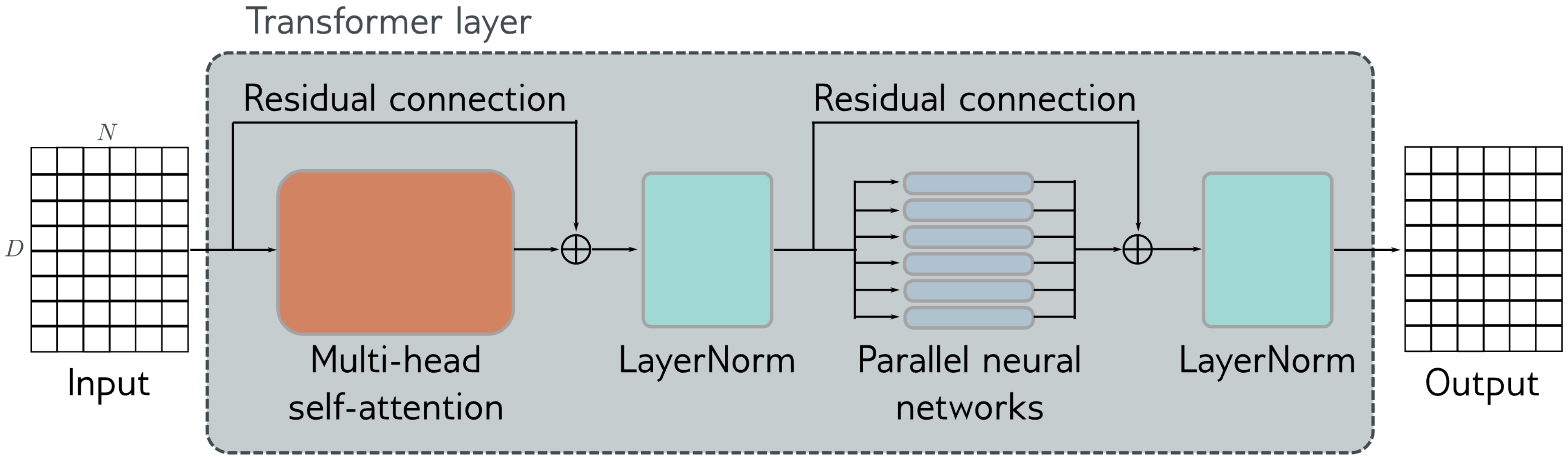
Transformers



- *Multi-headed Self Attention* is just one component of the transformer architecture

- Let's look at a transformer *block* (or *layer*) from the encoder

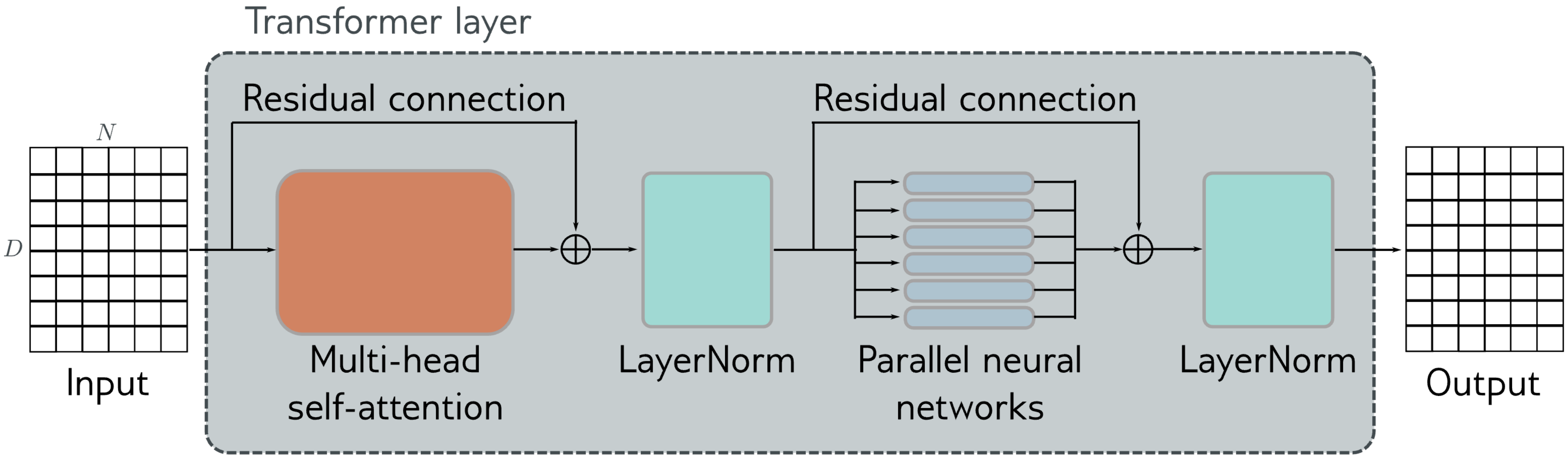
Transformer Layer -- Complete



- Adds a 2-layer MLP
- Adds residual connections around multi-head self-attentions and the parallels MLPs
- Adds LayerNorm, which normalizes across all the N input samples

Transform Layer
$\mathbf{X} \leftarrow \mathbf{X} + \mathbf{MhSa}[\mathbf{X}]$
$\mathbf{X} \leftarrow \mathbf{LayerNorm}[\mathbf{X}]$
$\mathbf{x}_n \leftarrow \mathbf{x}_n + \mathbf{mlp}[\mathbf{x}_n]$
$\mathbf{X} \leftarrow \mathbf{LayerNorm}[\mathbf{X}],$

Transformer Layer -- MLP

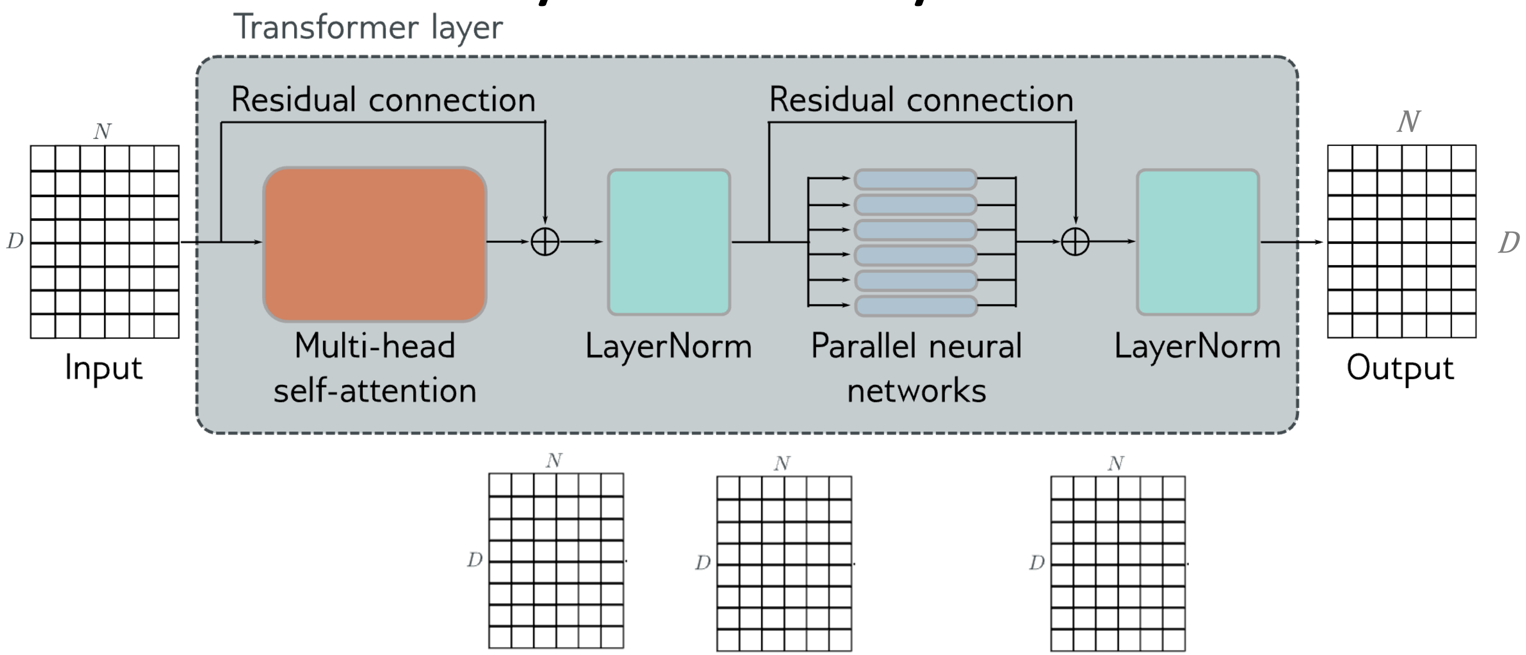


- Ads 2-layer MLP

- Same network (same weights) operates independently on each word
- Learn more complex representations and expand model capacity

$\text{Linear}_{Dx4D} \rightarrow \text{ReLU}(.) \rightarrow \text{Linear}_{4DxD}$

Transformer Layer -- LayerNorm



- Normalize across same layer
- Learned gain and offset

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Calculated column-wise

```
# NLP Example
batch, sentence_length, embedding_dim = 20, 5, 10
embedding = torch.randn(batch, sentence_length, embedding_dim)
layer_norm = nn.LayerNorm(embedding_dim)

# Activate module
layer_norm(embedding)
```

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

Transformers

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models

Transformers

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models
    - Encoder
    - Decoder
    - Encoder-Decoder

Transformers

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models
  - Encoder
  - Decoder
  - Encoder-Decoder

3 Types of Transformer Models

1. *Encoder* – transforms text embeddings into representations that support variety of tasks (e.g. sentiment analysis, classification)
   - ❖ Model Example: BERT

2. *Decoder* – predicts the next token to continue the input text (e.g. ChatGPT, AI assistants)
   - ❖ Model Example: GPT4, GPT4

3. *Encoder-Decoder* – used in sequence-to-sequence tasks, where one text string is converted to another (e.g. machine translation)

Encoder Model Example: BERT (2019)

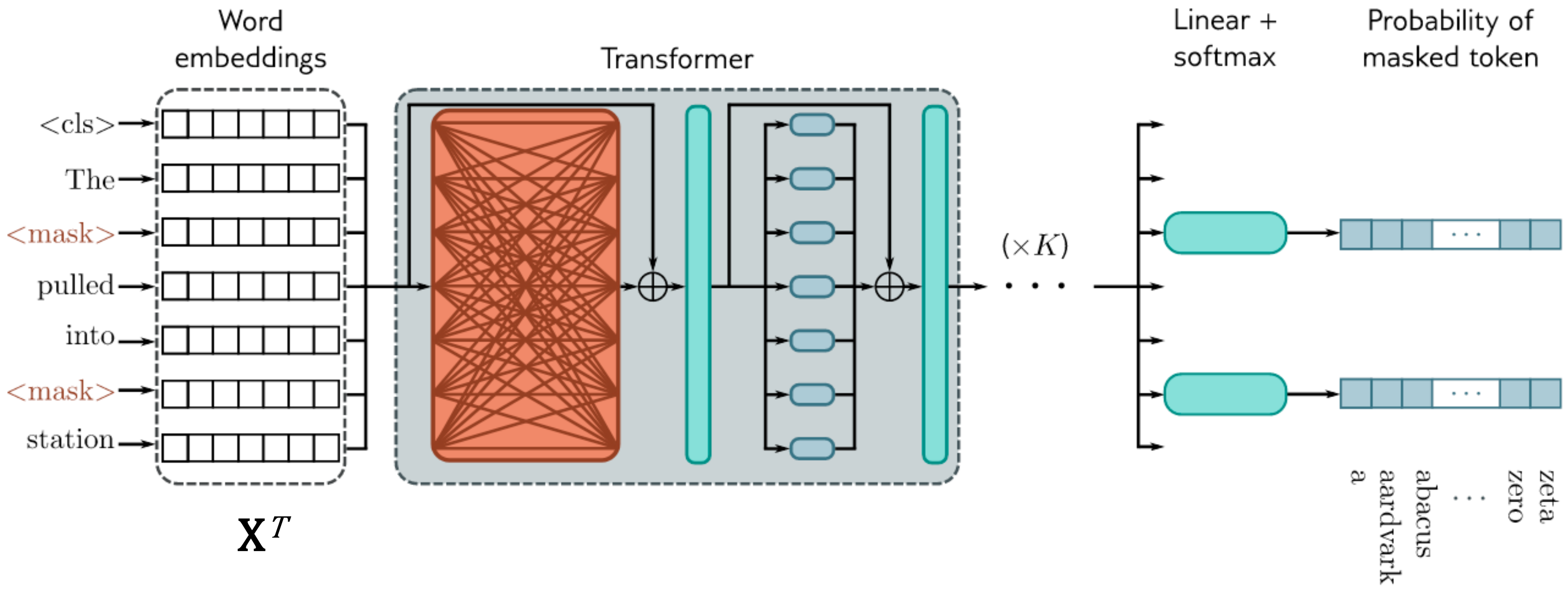
<u>B</u>idirectional <u>E</u>ncoder <u>R</u>epresentations from <u>T</u>ransformers

- Hyperparameters
  - 30,000 token vocabulary
  - 1024-dimensional word embeddings
  - 24x transformer layers
  - 16 heads in self-attention mechanism
  - 4096 hidden units in middle of MLP
- ~340 million parameters
- *Pre-trained* in a *self-supervised* manner,
- then can be adapted to task with one additional layer and *fine-tuned*

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” arXiv, May 24, 2019. doi: 10.48550/arXiv.1810.04805.

Encoder Pre-Training

Special <cls> token used for aggregate sequence representation for classification

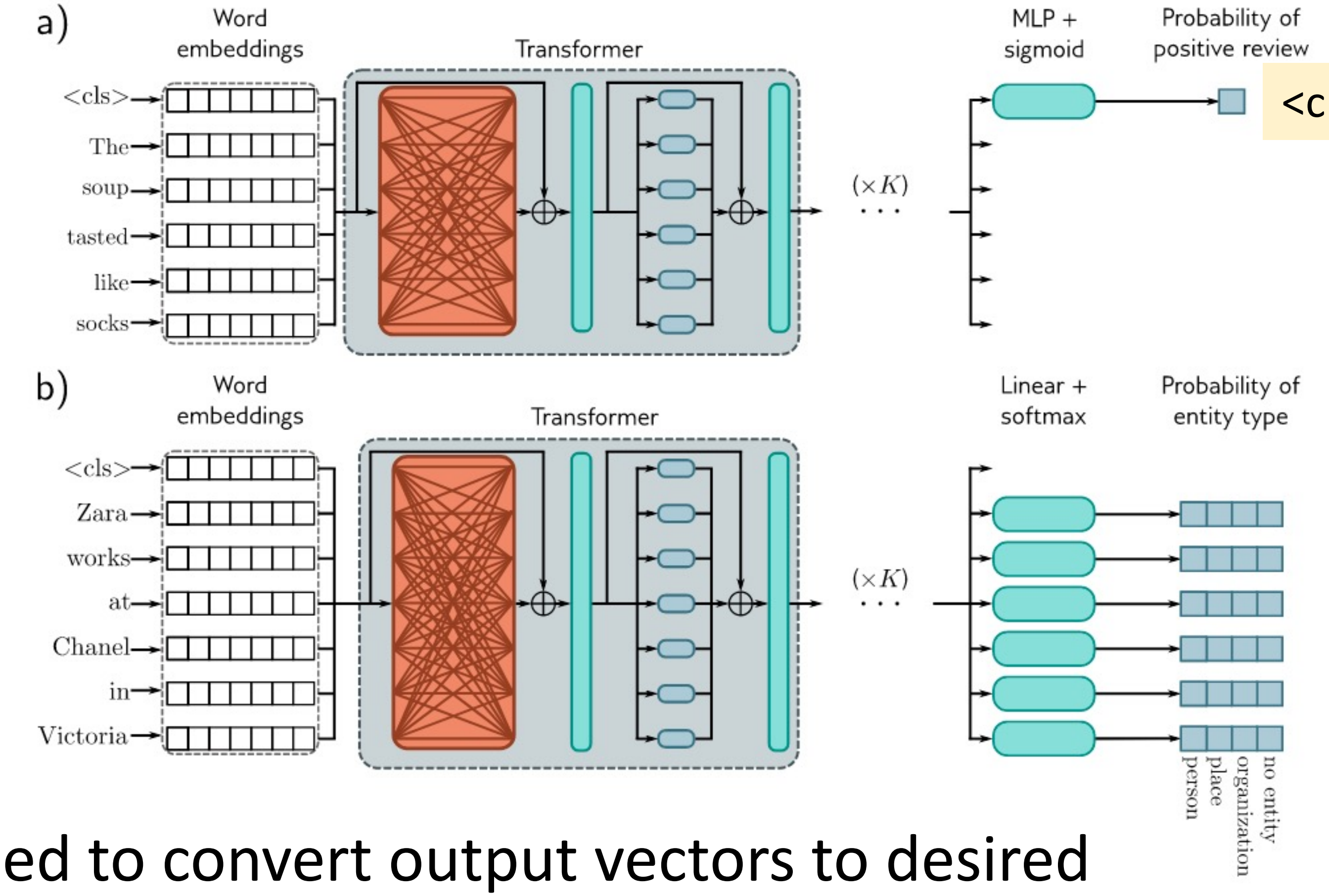


- A small percentage of input embedding replaced with a generic <mask> token
- Predict missing token from output embeddings
- Added linear layer and softmax to generate probabilities over vocabulary
- Trained on BooksCorpus (800M words) and English Wikipedia (2.5B words)

Encoder Fine-Tuning

Sentiment Analysis

Named Entity Recognition (NER)



<cls> token position

- Extra layer(s) appended to convert output vectors to desired output format
- 3rd Example: Text span prediction -- predict start and end location of answer to a question in passage of Wikipedia, see https://rajpurkar.github.io/SQuAD-explorer/

Transformers

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models
    - Encoder
    - Decoder
    - Encoder-Decoder

Decoder Model Example: GPT3 (2020)

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

- One purpose: *generate the next token in a sequence*
- By constructing an autoregressive model

T. B. Brown *et al.*, “Language Models are Few-Shot Learners.” arXiv, Jul. 22, 2020. doi: [10.48550/arXiv.2005.14165](https://doi.org/10.48550/arXiv.2005.14165).

Decoder Model Example: GPT3 (2020)

Generative Pre-trained Transformer

- One purpose: *generate the next token in a sequence*
- By constructing an autoregressive model
- Factors the probability of the sentence:

$$\Pr(Learning\ deep\ learning\ is\ fun) =$$

$$\Pr(Learning) \times \Pr(deep \mid learning) \quad \times$$
$$\Pr(learning \mid Learning\ deep) \times$$
$$\Pr(is \mid Learning\ deep\ learning) \times$$
$$\Pr(fun \mid Learning\ deep\ learning\ is)$$

T. B. Brown *et al.*, “Language Models are Few-Shot Learners.” arXiv, Jul. 22, 2020. doi: 10.48550/arXiv.2005.14165.

Decoder Model Example: GPT3 (2020)

Generative Pre-trained Transformer

- One purpose: *generate the next token in a sequence*
- By constructing an autoregressive model
- Factors the probability of the sentence:

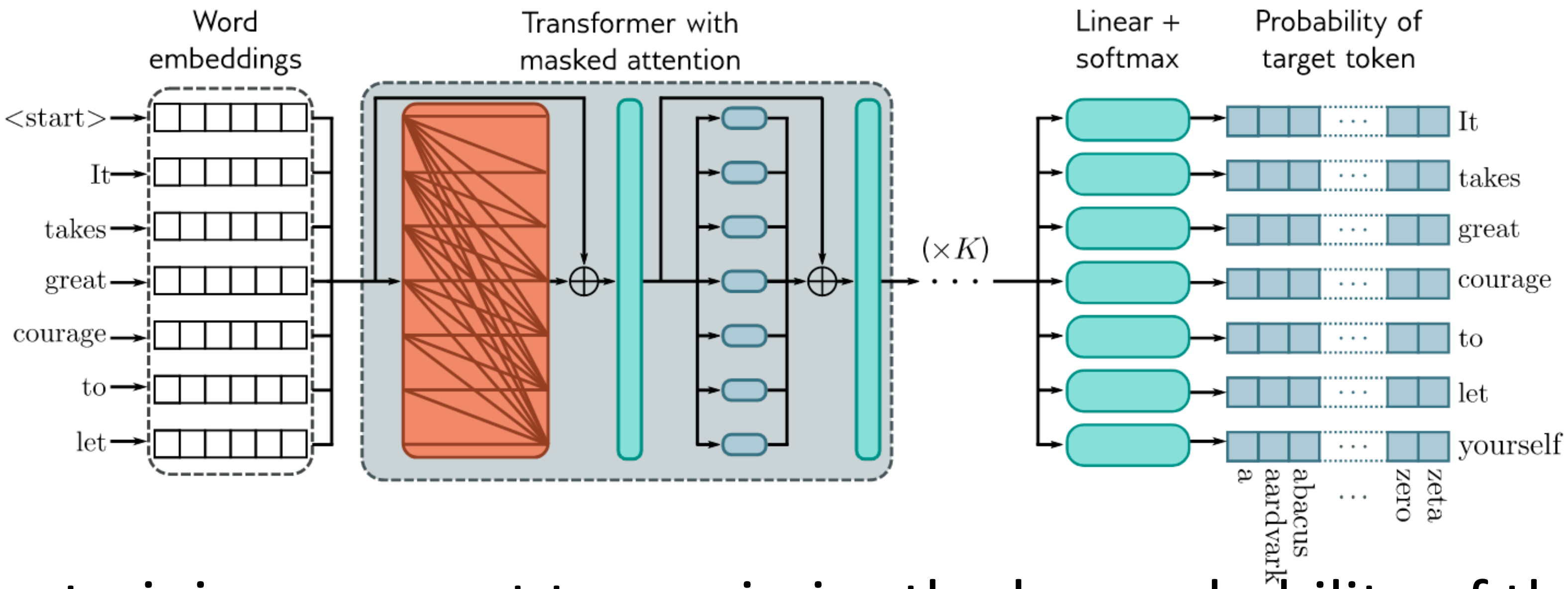
$$
\begin{aligned}
\Pr(\mathit{Learning\ deep\ learning\ is\ fun}) = \\
&\Pr(\mathit{Learning}) \times \Pr(\mathit{deep} \mid \mathit{learning}) \times \\
&\Pr(\mathit{learning} \mid \mathit{Learning\ deep}) \times \\
&\Pr(\mathit{is} \mid \mathit{Learning\ deep\ learning}) \times \\
&\Pr(\mathit{fun} \mid \mathit{Learning\ deep\ learning\ is})
\end{aligned}
$$

- More formally: Autoregressive model

$$\Pr(t_1, t_2, \dots, t_N) = \Pr(t_1) \prod_{n=2}^{N} \Pr(t_n \mid t_1, t_2, \dots, t_{n-1})$$

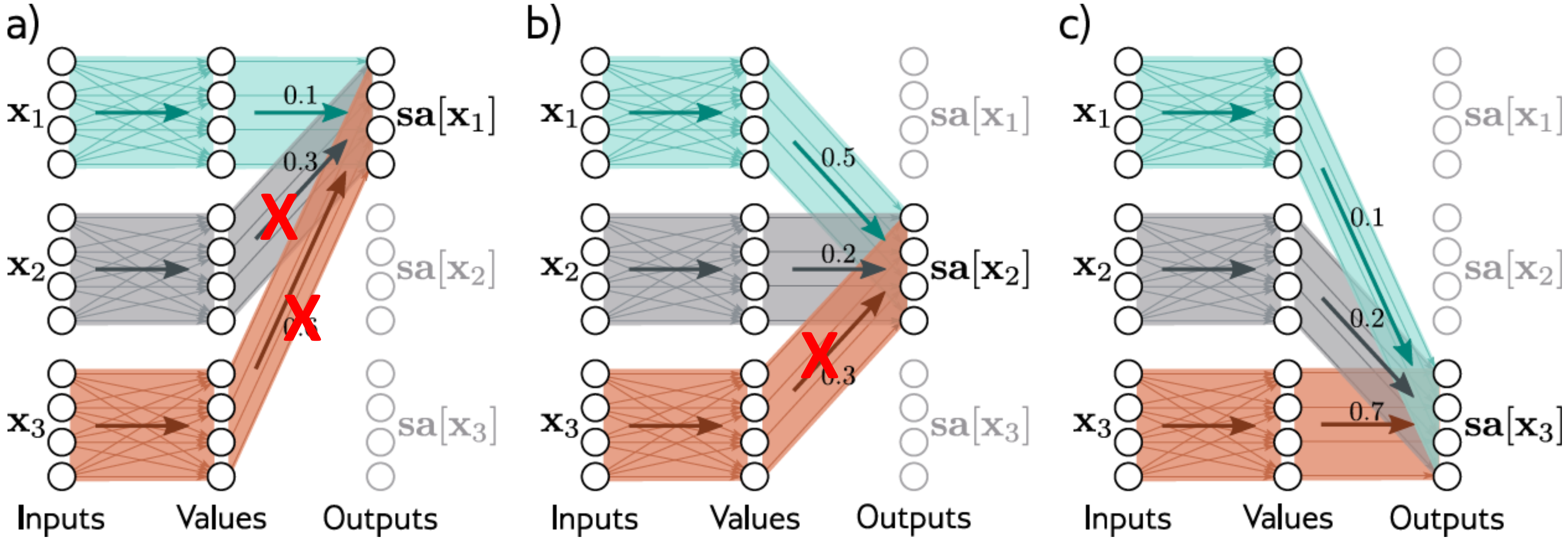
T. B. Brown *et al.*, "Language Models are Few-Shot Learners." arXiv, Jul. 22, 2020. doi: 10.48550/arXiv.2005.14165.

Decoder: *Masked* Self-Attention



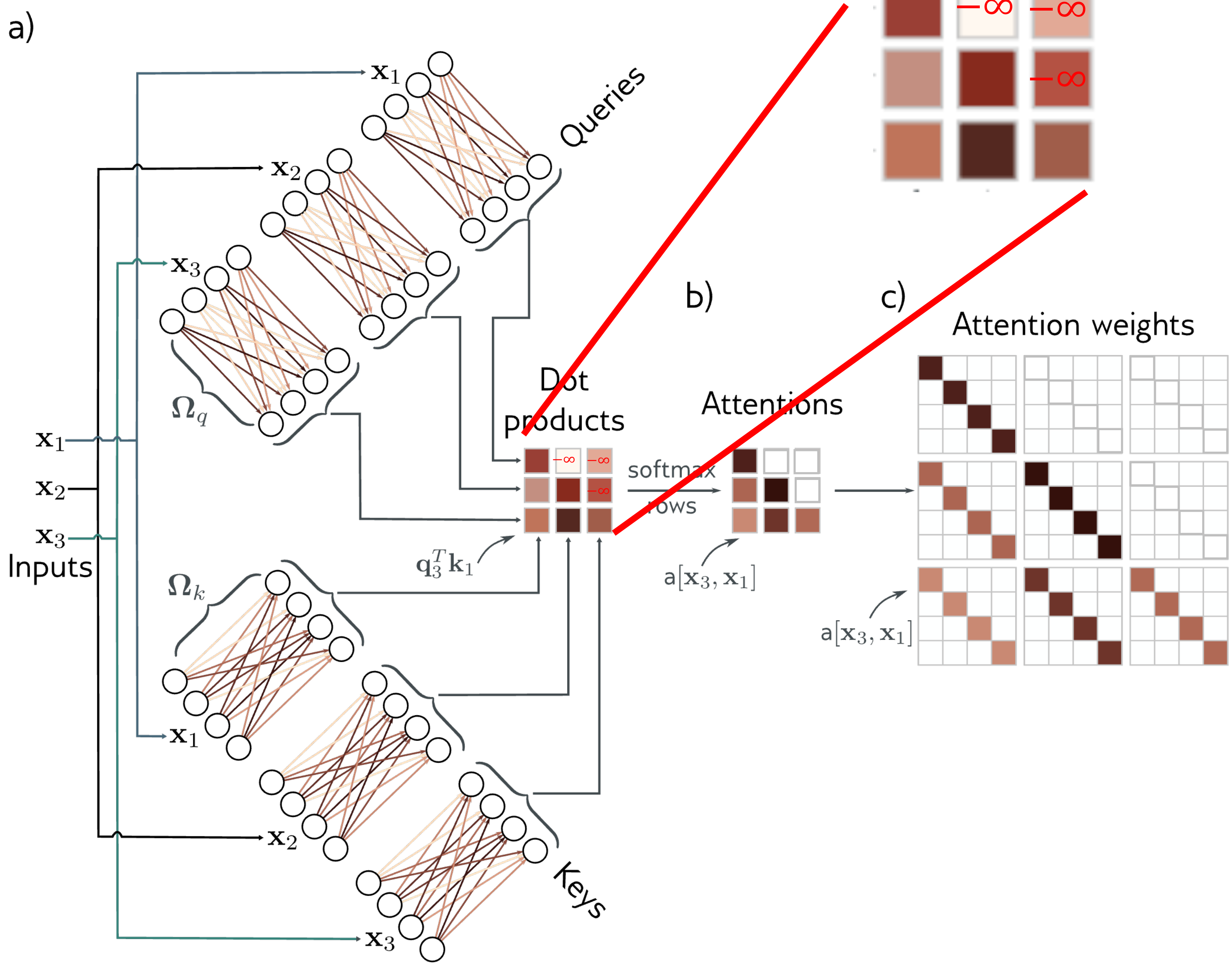
- During training we want to maximize the log probability of the input text under the autoregressive model
- We want to make sure the model doesn't "cheat" during training by looking ahead at the next token
- Hence we mask the self attention weights corresponding to current and right context to *negative infinity*

Masked Self-Attention

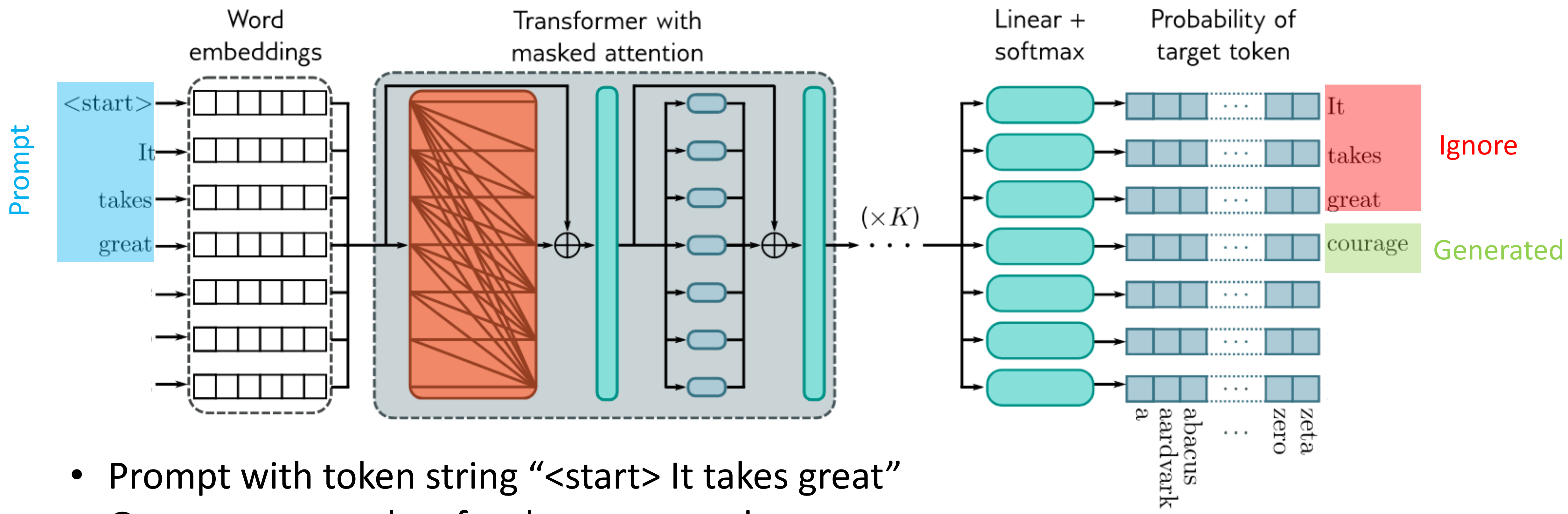


Mask right context self-attention weights to zero

Masked Self-Attention

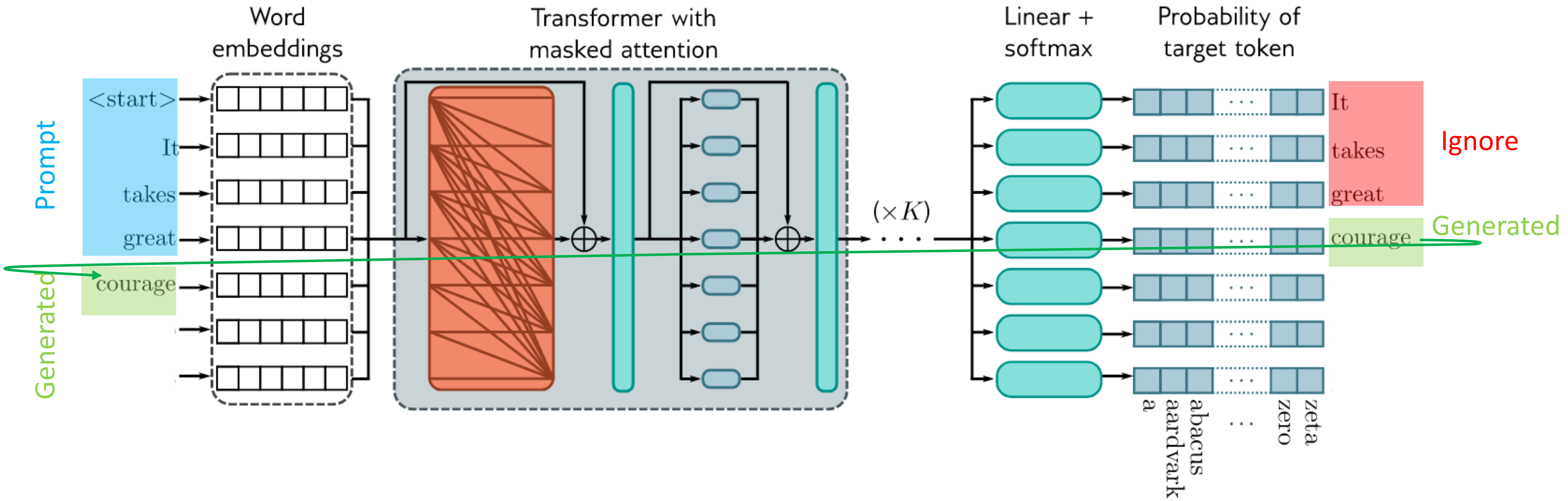


Decoder: Text Generation (Generative AI)



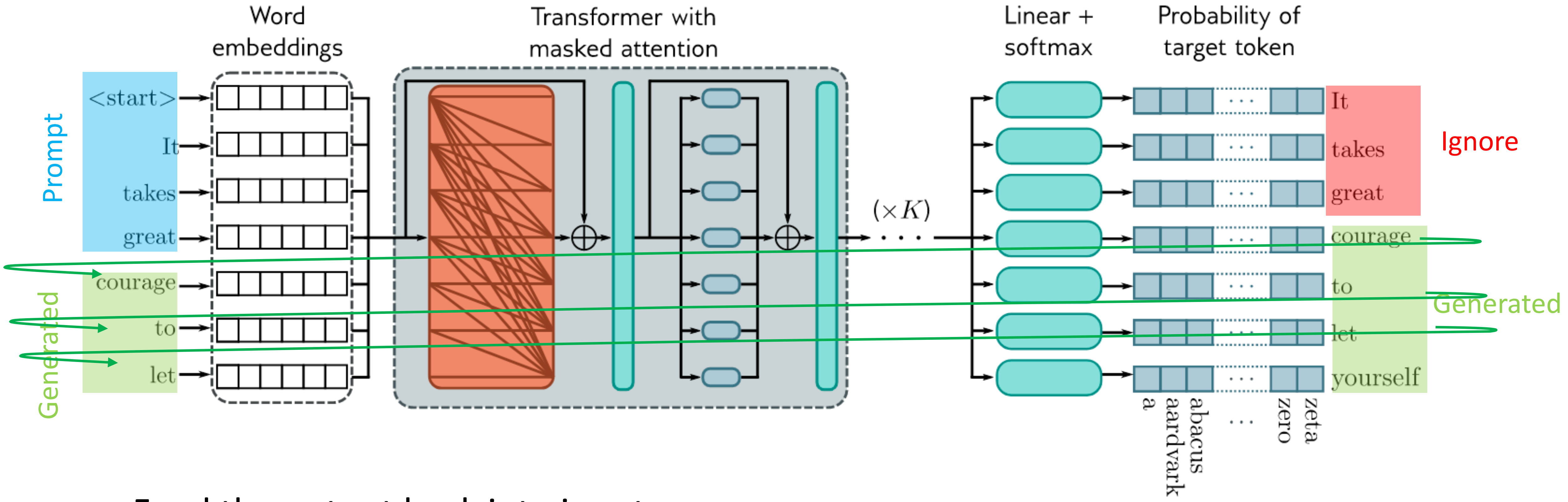
- Prompt with token string “<start> It takes great”
- Generate next token for the sequence by
  - picking most likely token
  - sample from the probability distribution
    - alternative *top-k* sampling to avoid picking from the long tail
  - beam search – select the most likely sentence rather than greedily pick

Decoder: Text Generation (Generative AI)



- Feed the output back into input

Decoder: Text Generation (Generative AI)



- Feed the output back into input

Technical Details

	BERT	GPT3
Model Architecture	Encoder	Decoder
Embedding Size	1024	12,288
Vocabulary	30K tokens	
Sequence Length		2048
# Heads	16	96
# Layers	24	96
Q,K,V dimensions	64	128
Training set size	3.3B tokens	300B+ tokens
# Parameters	340M	175B

Transformers

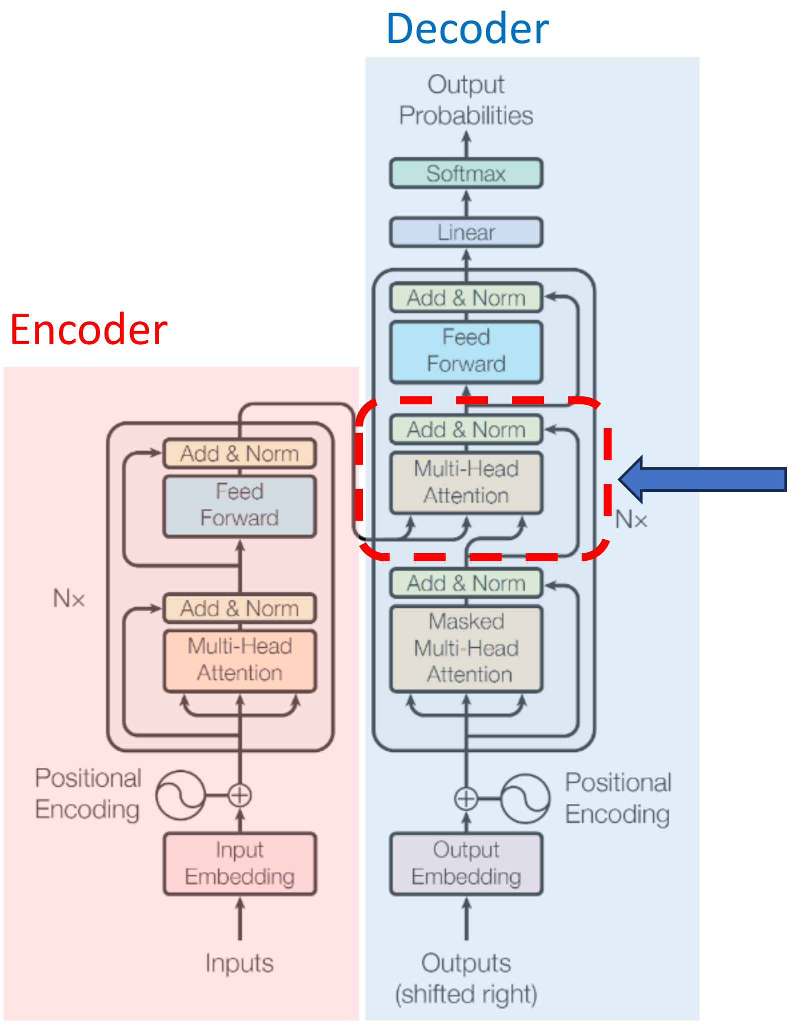
- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models
    - Encoder
    - Decoder
    - Encoder-Decoder

Encoder-Decoder Model

- Used for *machine translation*, which is a *sequence-to-sequence* task

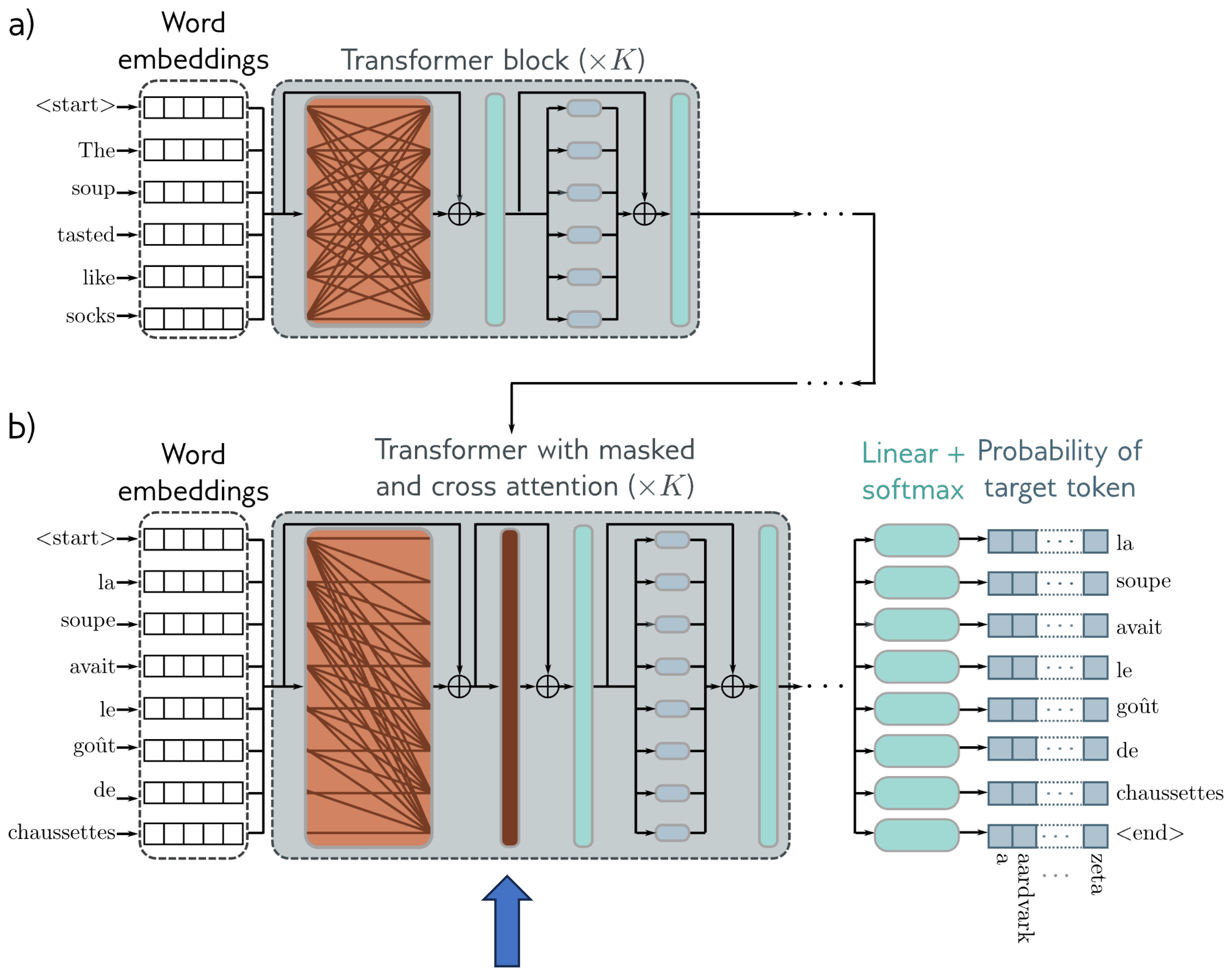
https://jalammar.github.io/illustrated-transformer/

Encoder Decoder Model



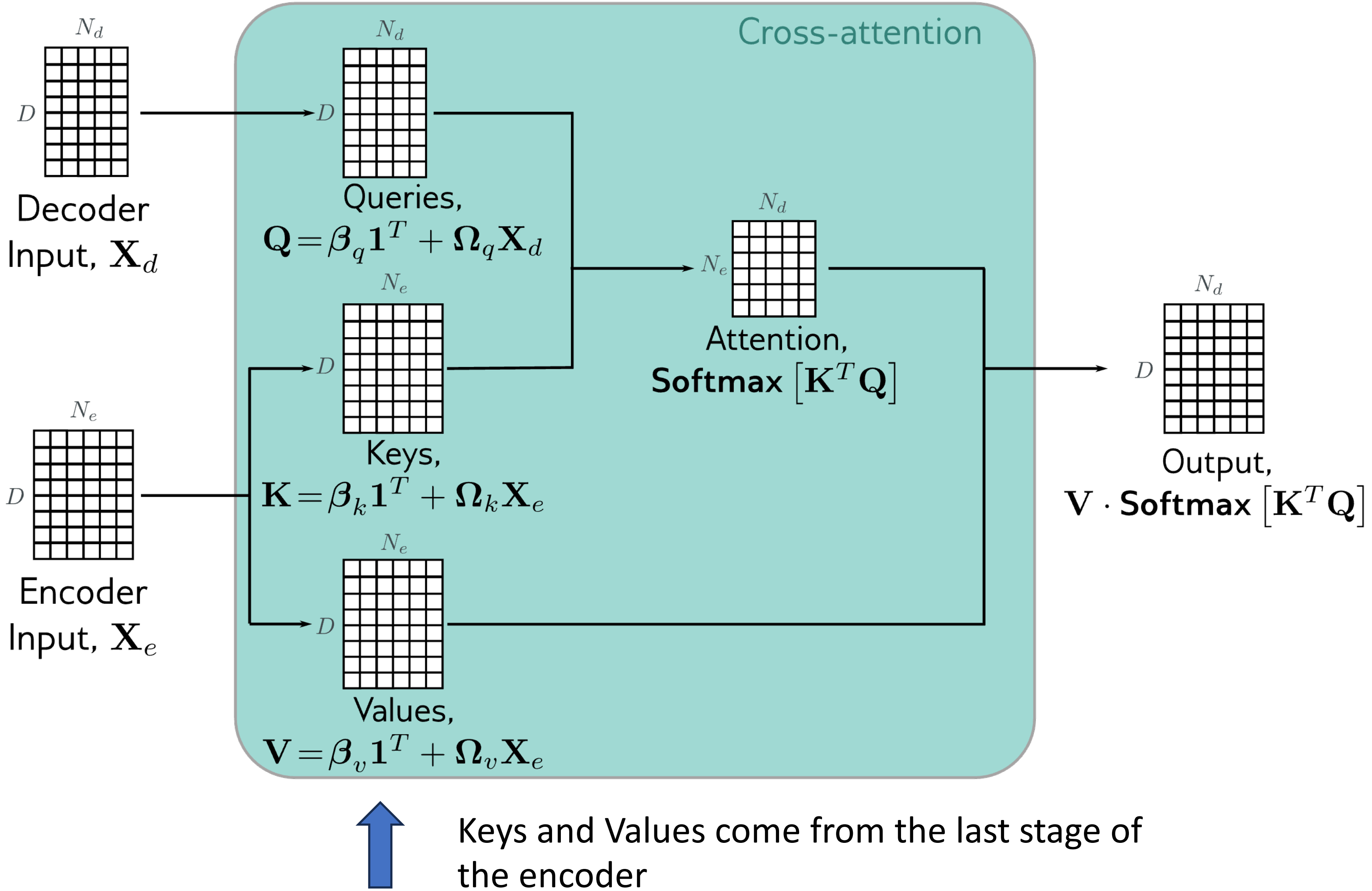
- The transformer layer in the decoder of the encoder-decoder model has an extra stage
- Attends to the input of the encoder with *cross attention* using Keys and Values from the output of the encoder
- Shown here on original diagram from "Attention is all you need" paper

Encoder Decoder Model



- Same view per UDL book

Cross-Attention



Keys and Values come from the last stage of the encoder

Next Time

- Tokenization and Learned Embeddings
- Training and Fine-Tuning Transformers
- Image Transformers
- Multimodal Transformers
- …

Feedback

Link