Transformers

Tripp Deep Learning F23

UNIVERSITY OF WATERLOO

TODAY'S GOAL

By the end of the class, you should be familiar with the main elements of transformer networks.

Summary

1. Transformers operate on sequences of vectors
2. Transformers use soft attention to weigh elements of a sequence
3. Queries, keys, and values undergo linear mappings
4. Attention is multi-headed
5. Encoder-decoder transformers can perform sequence-to-sequence mapping
6. Position encodings allow spatial attention
7. Transformers process elements of a sequence in parallel
8. Transformers can omit the encoder or decoder
9. Transformers perform self-supervised learning with large datasets
10. Transformers create contextualized embeddings

TRANSFORMERS OPERATE ON SEQUENCES OF VECTORS

Transformers were originally designed for text

- They were developed and first tested for language translation
- Today we will focus on this and other text-processing tasks, although they are now used in a much wider range of tasks such as speech, image, and video processing
- More generally, they deal with sequences, and things that can be serialized into sequences

Tokenization

- A sequence of text is broken down into elements from a predefined dictionary
- The dictionary could in principle consist of words, parts of words, or individual characters
  - Each item in the dictionary is mapped to an index
  - Any elements that are not part of the dictionary are mapped to a special index
- Text processing is more challenging with:
  - Longer sequences
  - Larger dictionaries
- Different approaches trade off sequence length and dictionary size

Word tokenization

- Example: “This sentence can be tokenized.” -> [“This”, “sentence”, “can”, “be”, “tokenized”]
- Limitations:
    - Related words such as “token” and “tokenize” or “cat” and “cats” are mapped to different indices, and the network has no prior knowledge that these indices are related
    - There are many words (about 170K in current use in English); this requires vocabularies that are large and usually incomplete (not including all possible words, such as rarely used words and acronyms)

Character tokenization

- Example: “This sentence can be tokenized.” -> [“T”, “h”, “i”, “s”, “ ”, “s”, “e”, “n”, “t”, “e”, “n”, “c”, “e”, “ ”, “c ”, “a”, “n”, “ ”, “b”, “e”, “ ”, “t”, “o”, “k”, “e”, “n”, “i”, “z”, “e”, “d”, “.”]
- Advantage: Smaller vocabulary (e.g., 256 ASCII characters)
- Limitations:
  - Individual sequence elements have little meaning; meaning emerges from combinations of multiple elements
  - Much longer sequences are required to encode the same text

Sub-word tokenization

- Example: “This sentence can be tokenized.” -> [“This”, “sentence”, “can”, “be”, “token”, “ized”]
- Infrequently used words are divided into parts, such as prefixed and suffixes, or even individual letters; very frequently used words are not divided
- Advantages:
  - Smaller vocabulary than word tokenization
  - Shorter sequences than character tokenization
- Common variations include WordPiece (Wu et al., 2016) and Byte-Pair Encoding (Sennrich et al., 2015) tokenizers, both of which start with a vocabulary of individual characters and combine them iteratively to reduce the token length of a training corpus

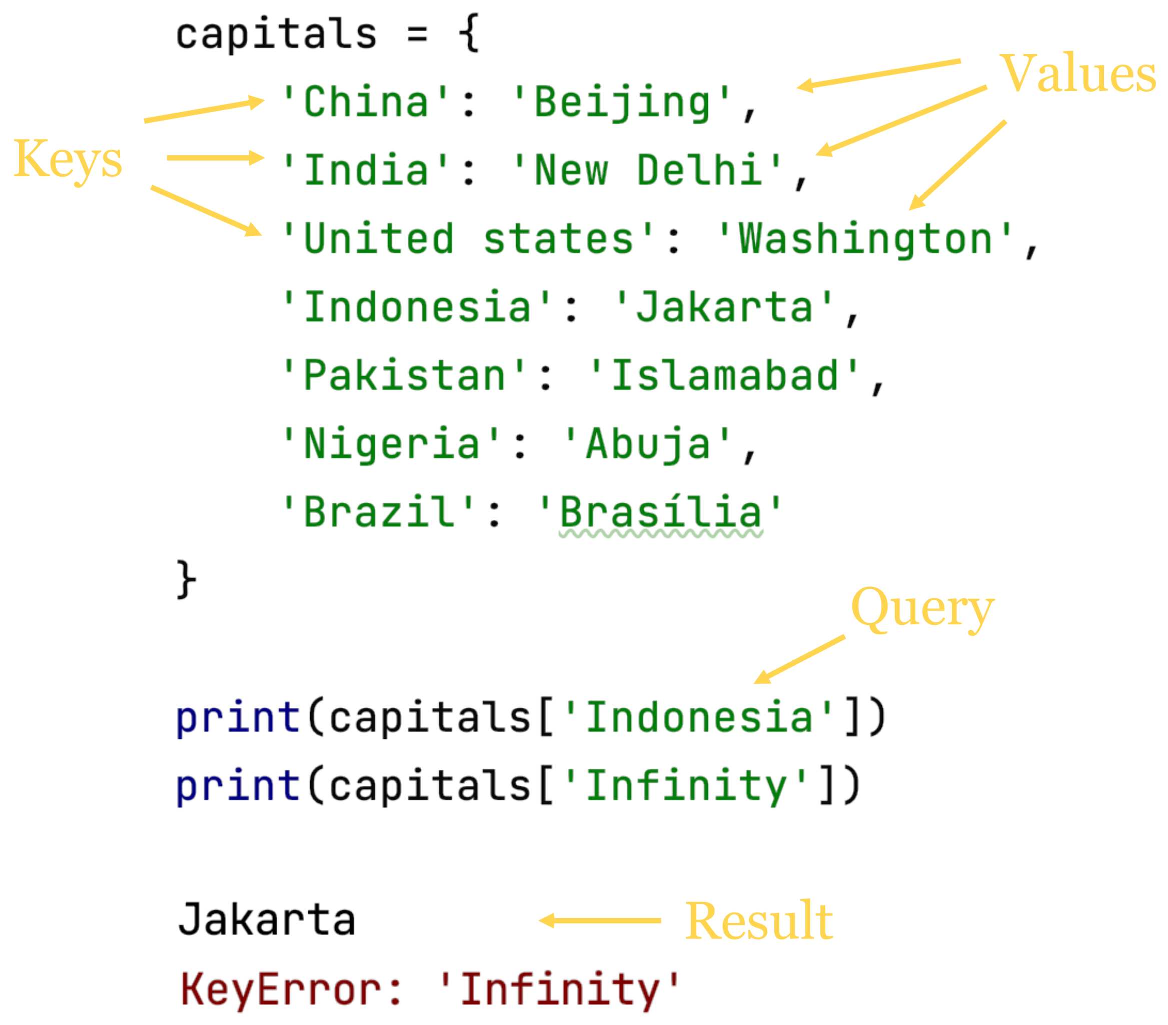
Embeddings

- Each index in the vocabulary is mapped to a high-dimensional embedding vector (e.g., 768D) as the first step in the transformer network
- The vectors are initialized randomly and optimized during training

TRANSFORMERS USE SOFT ATTENTION TO WEIGH ELEMENTS OF A SEQUENCE

Python dictionary analogy

- Python dictionaries contain key-value mappings
- If you provide a query that matches one of the keys, it will return the corresponding value
- Otherwise, it will raise an error



```
capitals = {
    'China': 'Beijing',
    'India': 'New Delhi',
    'United states': 'Washington',
    'Indonesia': 'Jakarta',
    'Pakistan': 'Islamabad',
    'Nigeria': 'Abuja',
    'Brazil': 'Brasília'
}

print(capitals['Indonesia'])
print(capitals['Infinity'])

Jakarta
KeyError: 'Infinity'
```

Python dictionary hypothetical generalization

- Suppose we wanted to support queries that partially matched multiple keys
- We could imagine a dictionary that returned multiple values with weights based on similarity of queries with different keys

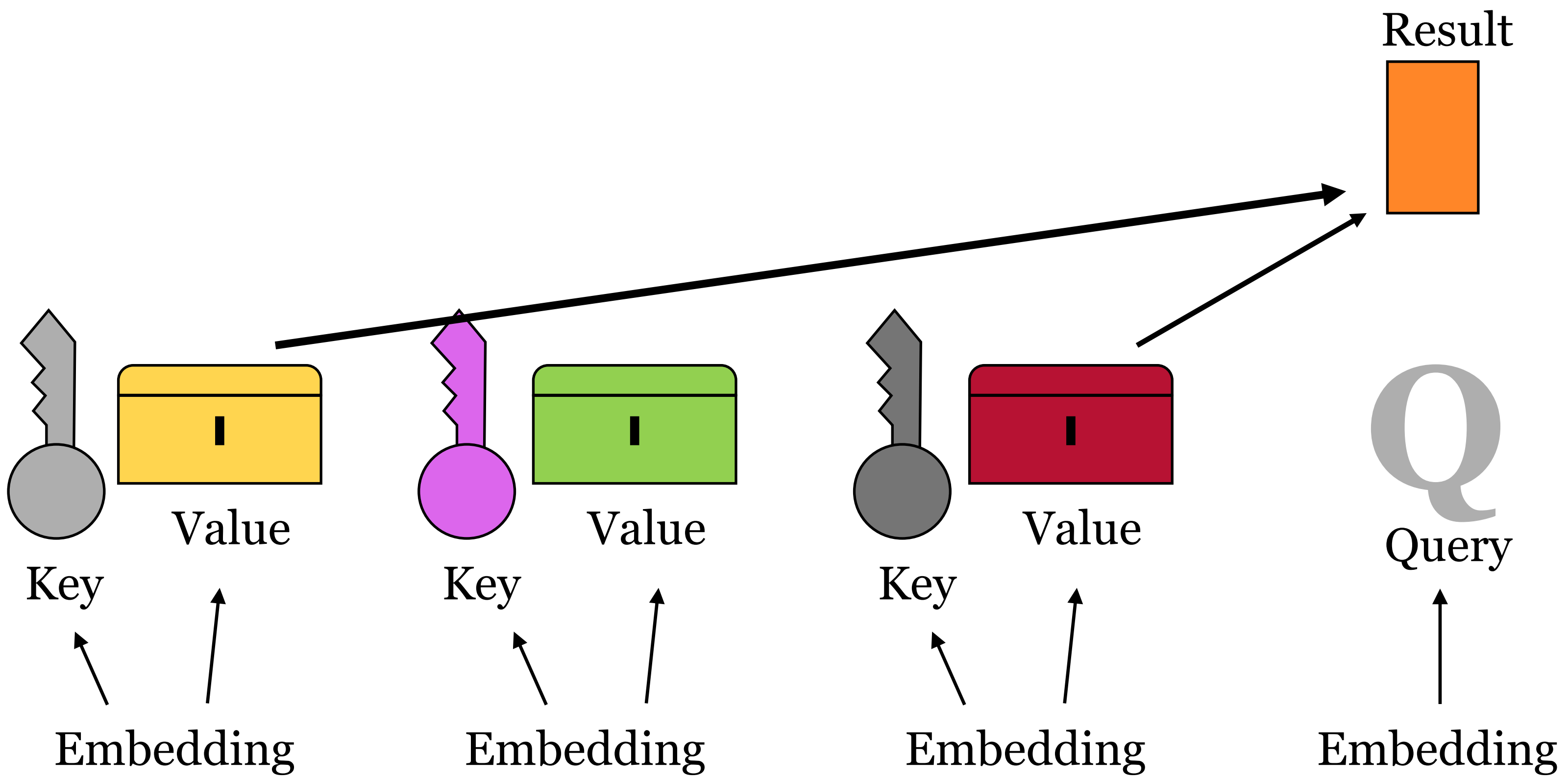
```
capitals = {
    'China': 'Beijing',
    'India': 'New Delhi',
    'United states': 'Washington',
    'Indonesia': 'Jakarta',
    'Pakistan': 'Islamabad',
    'Nigeria': 'Abuja',
    'Brazil': 'Brasília'
}

print(capitals['Indonesia'])
print(capitals['Infinity'])
```

```
('Jakarta', 1)
(('Jakarta', 0.5), ('New Delhi', 0.4), ('Islamabad', 0.1))
```

Weighted result (total weight = 1) ⟶

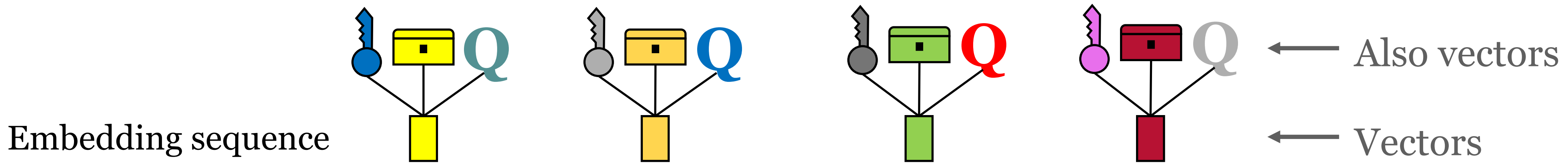
Transformer attention



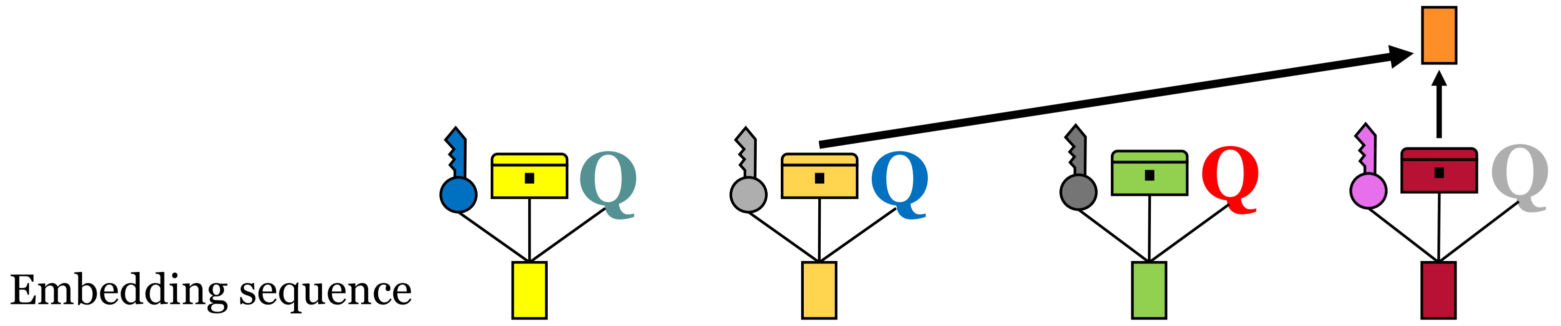
Transformer attention

Embedding sequence
Vectors

Transformer attention

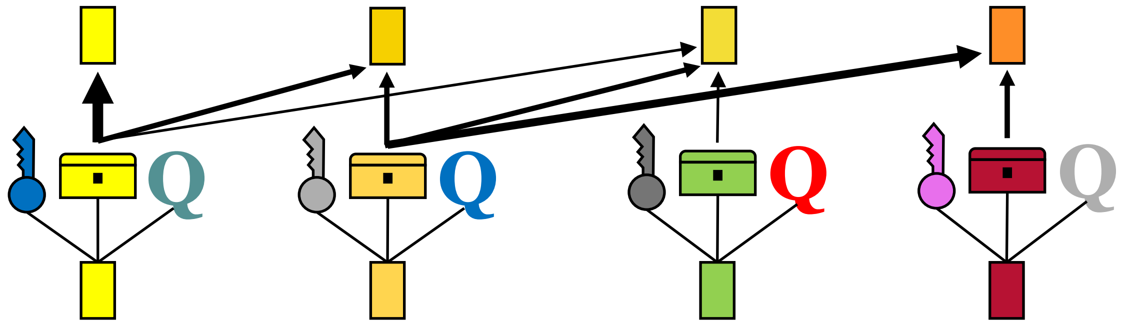


Transformer attention



Transformer attention

Embedding sequence



The transformer attention mechanism

- Queries, keys, and values are high-dimensional vectors (e.g., 768D)
- The similarity of query i and key j vectors is their dot product divided by the square root of the dimension, i.e., $s_{ij} = \boldsymbol{q}_i^T \boldsymbol{k}_j / \sqrt{d}$
  - The $\sqrt{d}$ factor controls the variance of s so it doesn't depend on d
- A softmax function produces value weightings for the i^{th} query so that they sum to one, i.e., $\boldsymbol{w}_i = [\frac{e^{s_{ij}}}{\sum_k e^{s_{ik}}}]$
- Multiple key and value vectors are processed together as columns of matrices K and V, so

$$\boldsymbol{w_i} = \text{softmax}\left(\frac{K^T \boldsymbol{q_i}}{\sqrt{d}}\right)$$

$$\text{Attention}(\boldsymbol{q_i}, K, V) = V\boldsymbol{w_i}$$

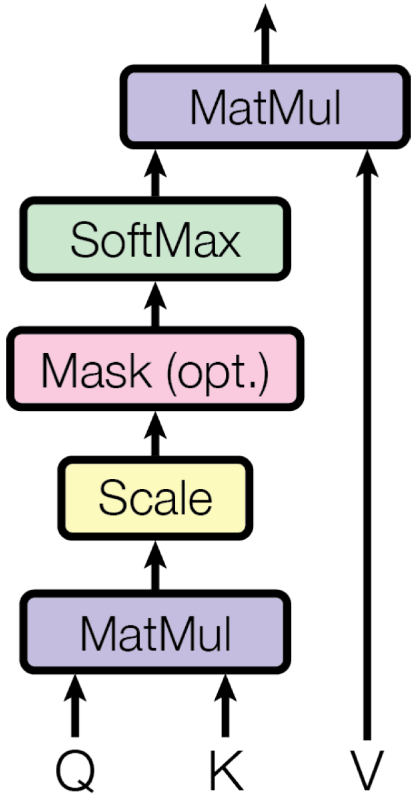
The transformer attention mechanism

- Furthermore, multiple queries are processed together as columns of matrix Q, so

$$\text{Attention}(Q, K, V) = VW$$

$$W = \text{softmax}\left(\frac{K^T Q}{\sqrt{d}}\right)$$

- Here the softmax is applied separately to each column and the result is a matrix in which each column contains a sum of weighted values for a different query
- Sometimes, some of the keys are masked by setting their similarities to $-\infty$



Vaswani et al., 2017

Example low-dimensional calculation

Here we have two keys,

$$K = \begin{bmatrix} -0.04 & 0.45 \\ 1.34 & 0.53 \end{bmatrix}$$

and one query,

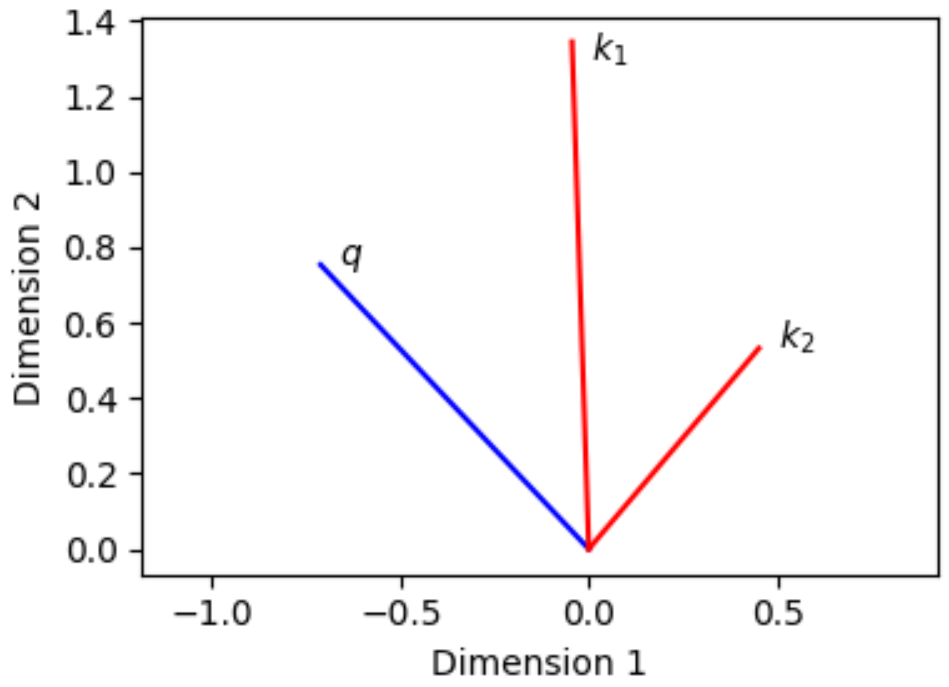
$$\boldsymbol{q} = \begin{bmatrix} -0.71 \\ 0.75 \end{bmatrix}$$

The similarities are 0.74 and 0.06. The softmax weights are 0.66 and 0.34. If the values were

$$V = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$$

then the result would be,

$$.66 \begin{bmatrix} 2 \\ 0 \end{bmatrix} + .34 \begin{bmatrix} 0 \\ -2 \end{bmatrix} = \begin{bmatrix} 1.32 \\ -0.68 \end{bmatrix}.$$



Soft vs. hard attention

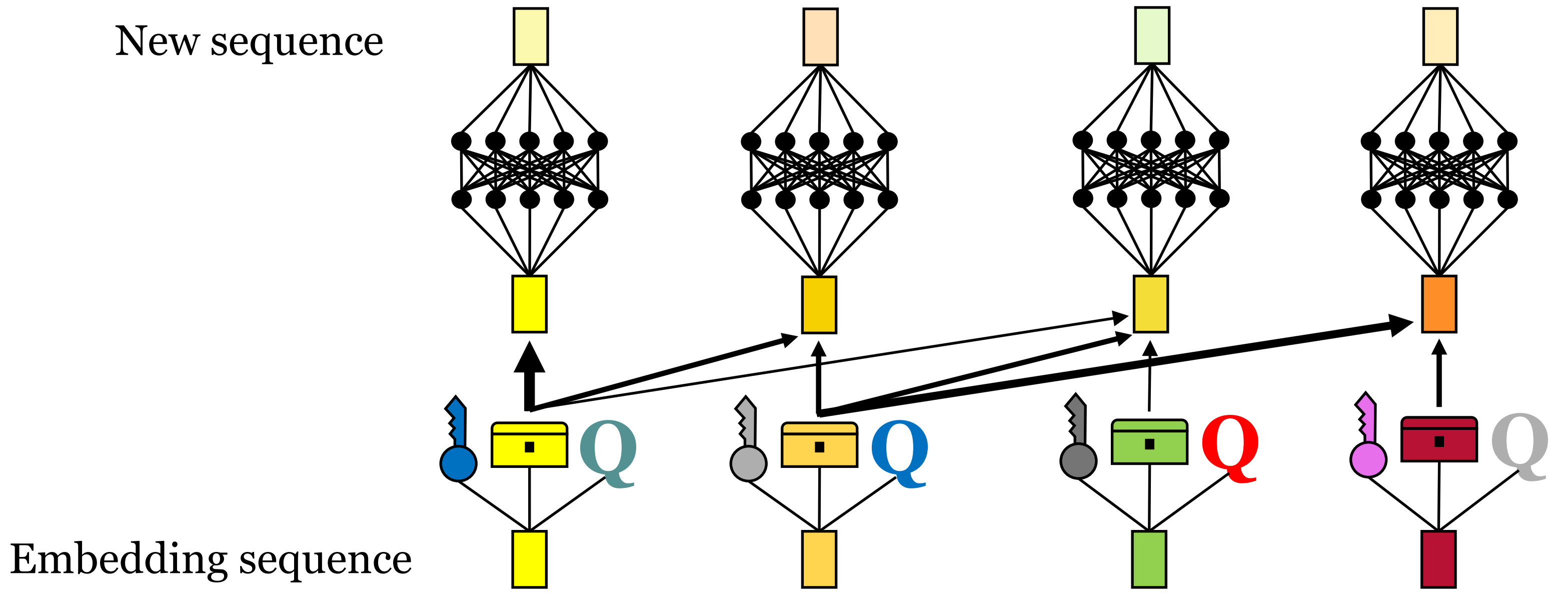
- This is a “soft” attention mechanism
- In contrast, some work before transformers had used “hard” attention
- Hard attention applies weights of zero and one, depending on whether each similarity is below or above a threshold

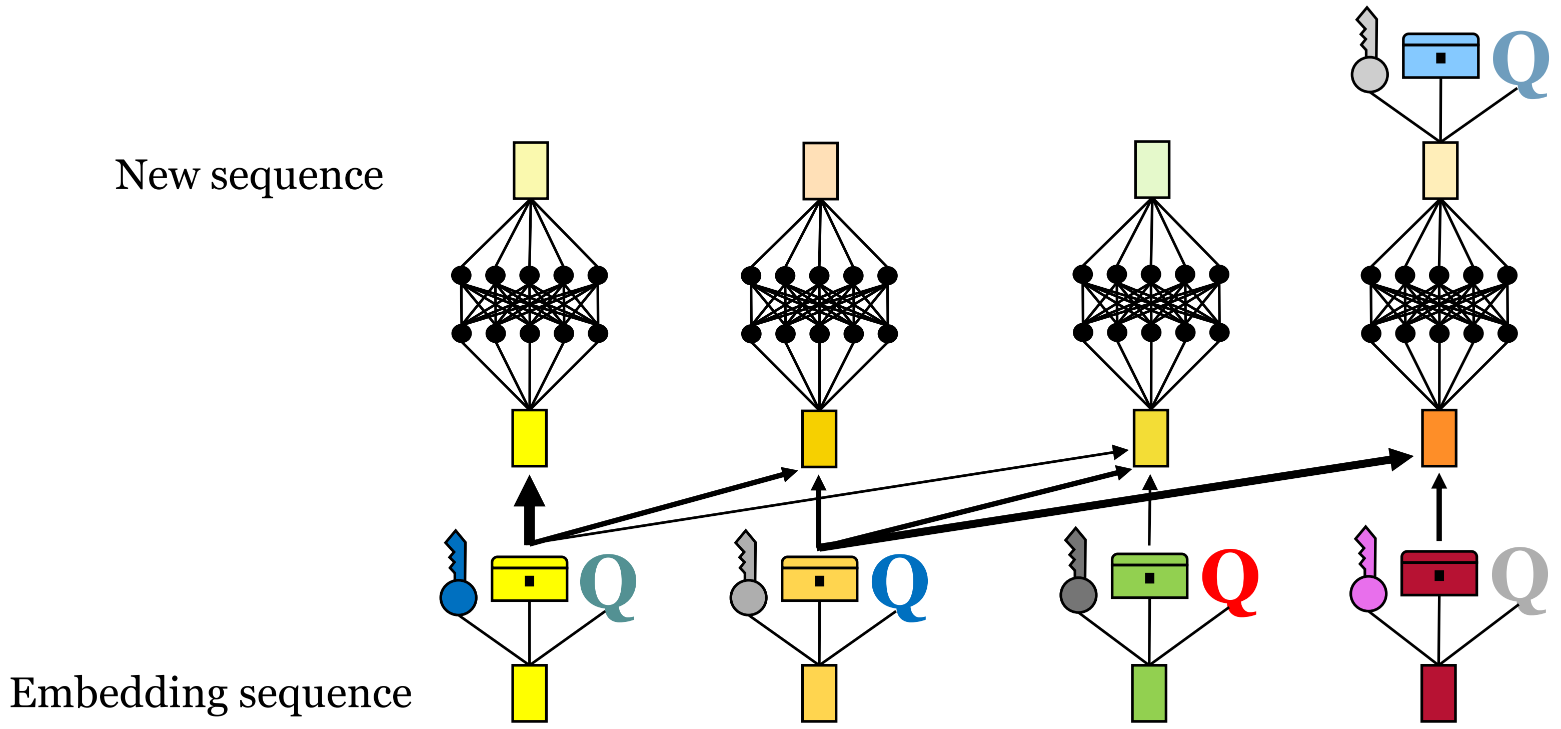
Contrast with psychology of attention

"Everyone knows what attention is. It is the taking possession by the mind, in clear and vivid form, of one out of what seem several simultaneously possible objects or trains of thought. Focalization, concentration, of consciousness are of its essence. It **implies withdrawal from some things in order to deal effectively with others** ..."

- William James

https://psychclassics.yorku.ca/James/Principles/prin11.htm


New sequence
Q
Q
Q
Q
Embedding sequence


New sequence
Q
Q
Q
Q
Q
Embedding sequence

QUERIES, KEYS, AND VALUES UNDERGO LINEAR MAPPINGS

Linear mappings

- Recall transformers process sequences of embeddings
- The embeddings do not enter the attention mechanism directly but are first multiplied by learned matrices
- This can be written as,

$$\text{Attention}(W^Q Q, W^K K, W^V V)$$

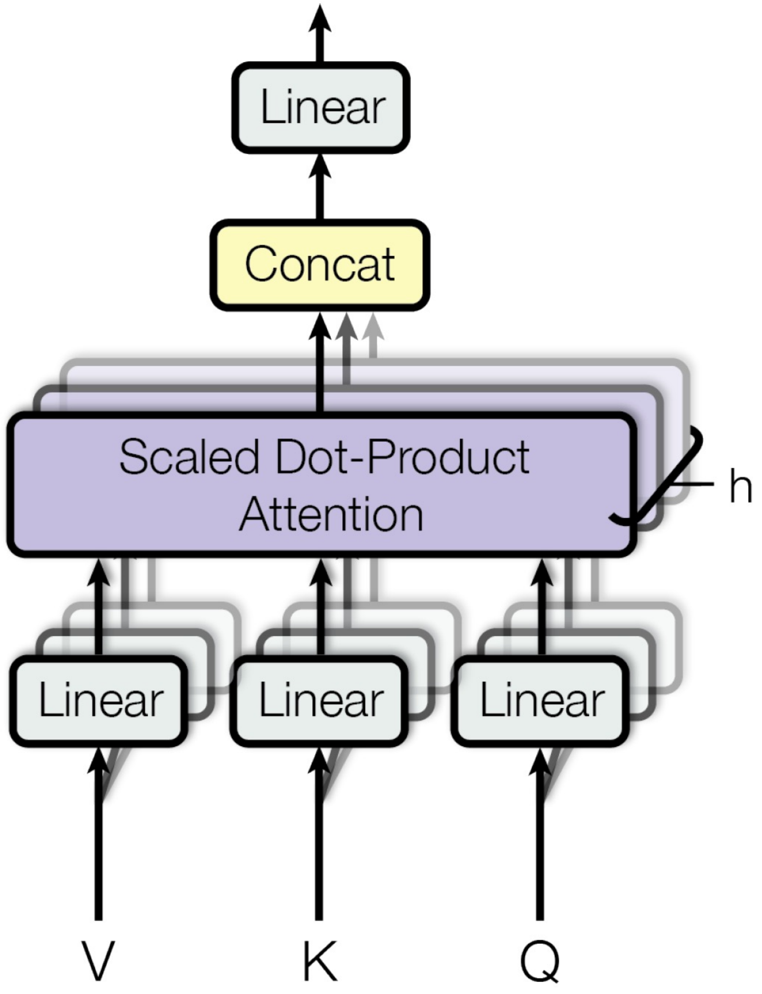
Consequences

- Mapping V allows the attention mechanism to change the incoming values (like a fully-connected layer)
- Mapping K and Q allows flexibility in query-key similarity
- Example: The network could learn to always ignore useless keys by making W^K rank-deficient
- Example: The network could learn that the query “x” should return values associated with key “y”, even though x and y are not similar

ATTENTION IS MULTI-HEADED

Multi-headed attention

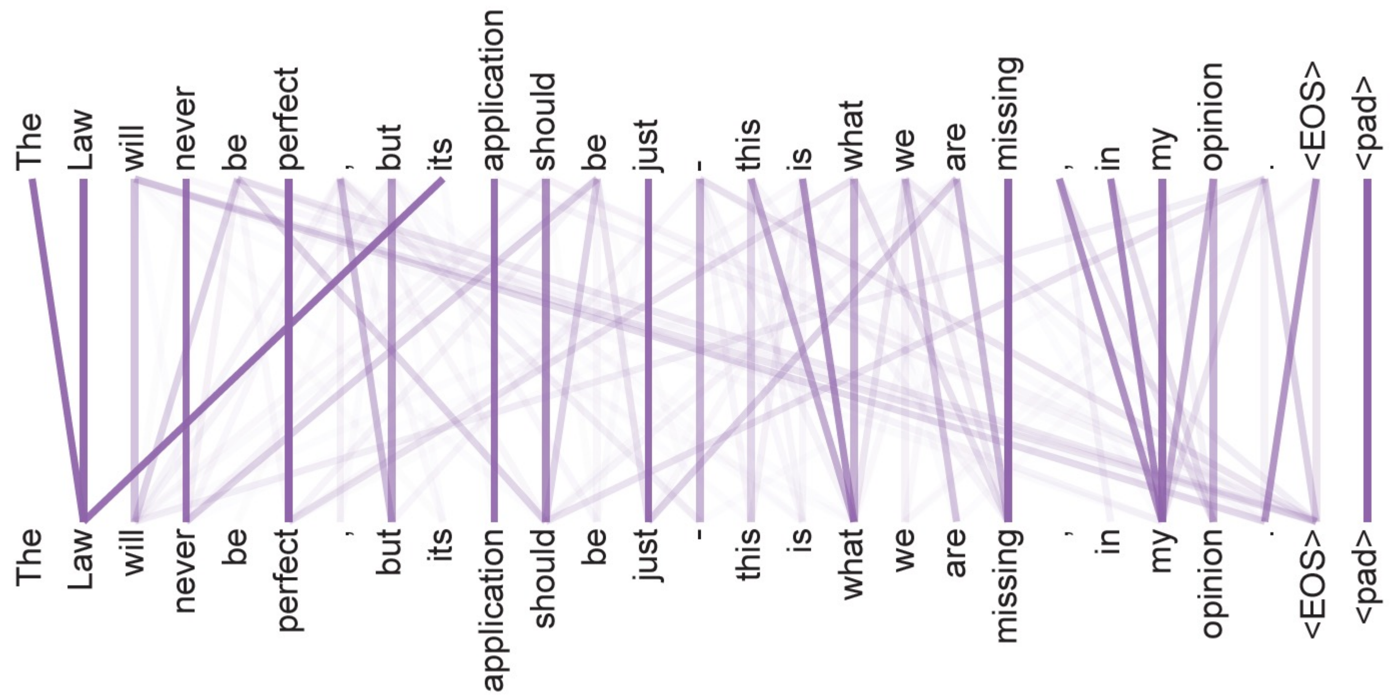
- Several attention mechanisms with different learned linear maps are concatenated and combined with an additional linear map
  - The final linear map mixes information from all heads
- Each "head" in such a group has the same V, K, and Q inputs
- If h is the number of heads and d is dimension of the embeddings, then each head uses dimension d/h
- Each head can attend to different things given the same query and can map values in different ways

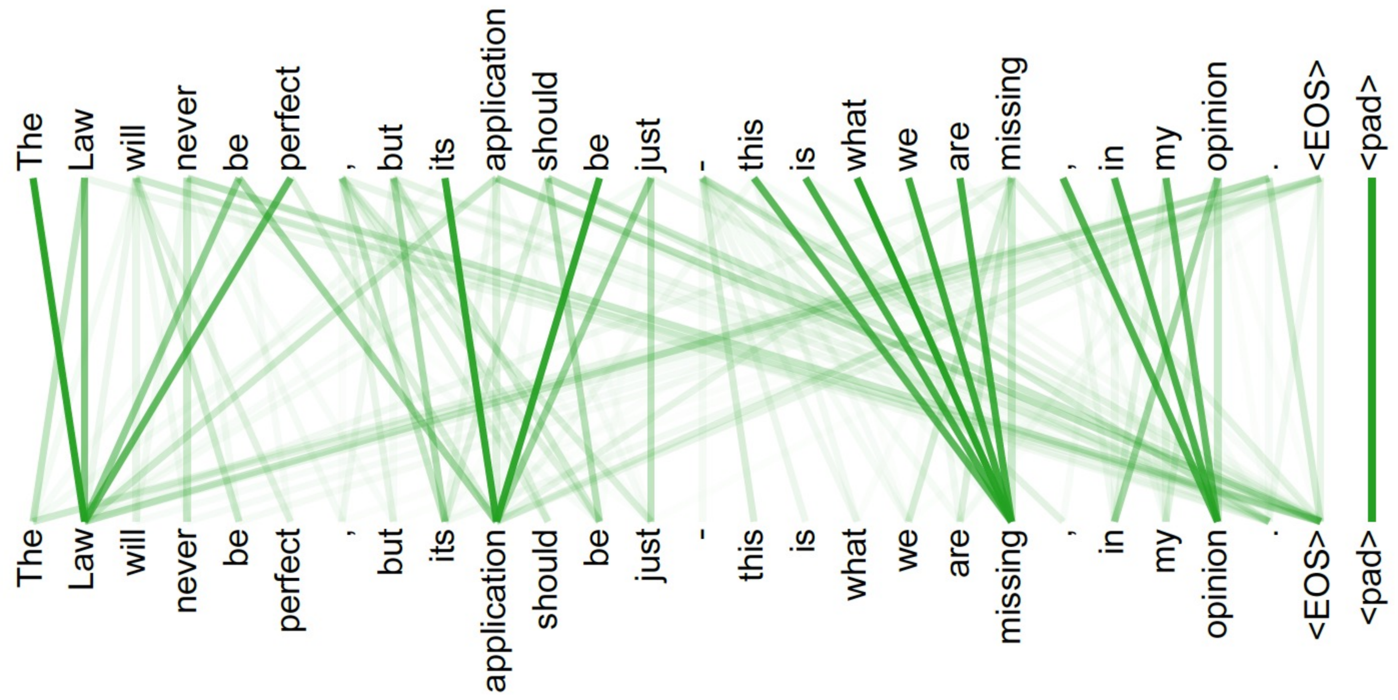


Vaswani et al., 2017

Example

- Attention weights of two different heads in a deeper layer (Vaswani et al., 2017)

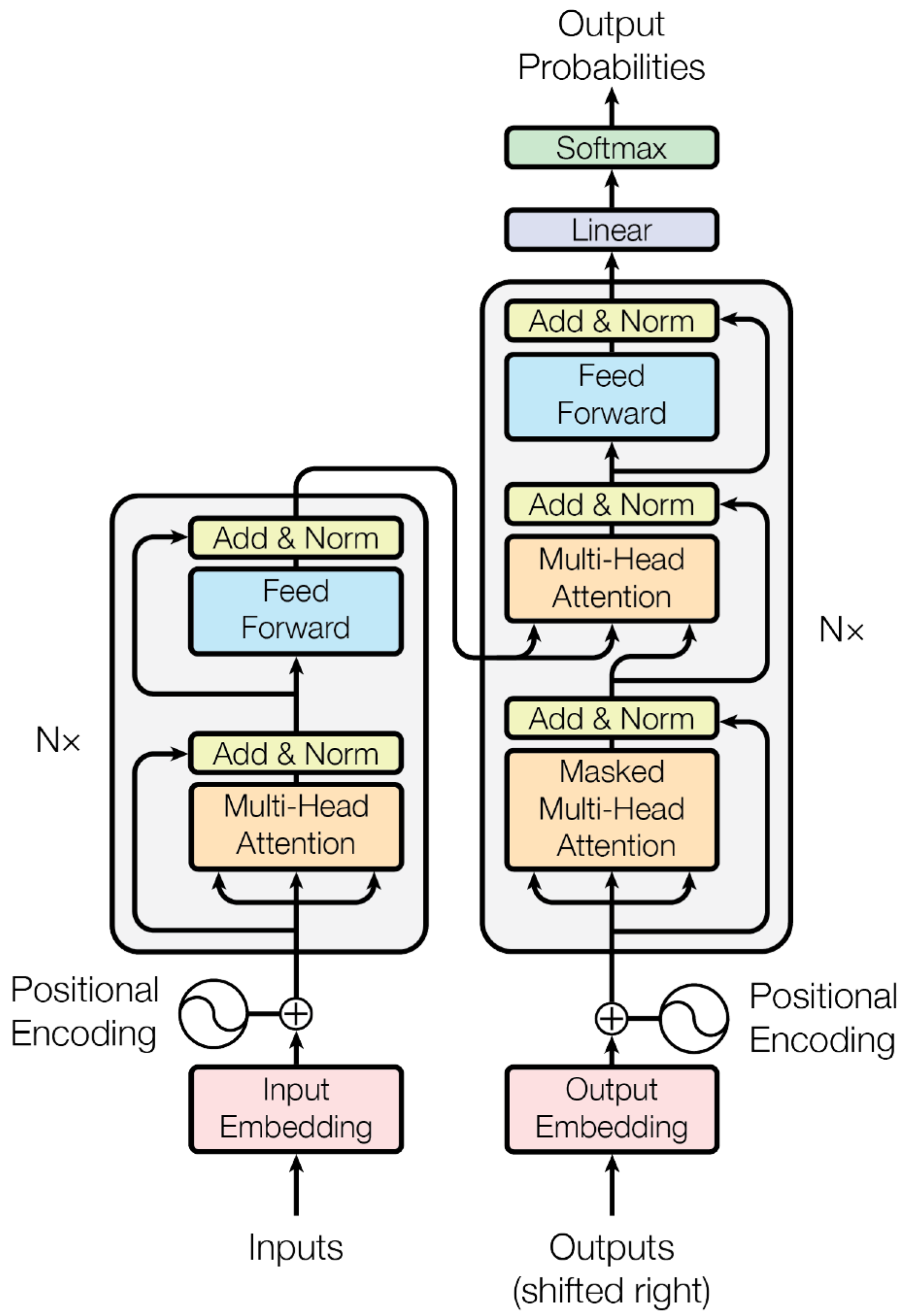




ENCODER-DECODER TRANSFORMERS CAN PERFORM SEQUENCE-TO-SEQUENCE MAPPING

Encoder-decoder architectures

- This is the original transformer model, which has an encoder-decoder architecture (Vaswani et al, 2017)
- Applied to English-German and English-French translation and established new state-of-art in each case
- At this time, translation had been done with recurrent networks, and adding attention to these networks had been found to improve performance
- This paper showed that a model with attention alone (no recurrence) was more effective and faster to train

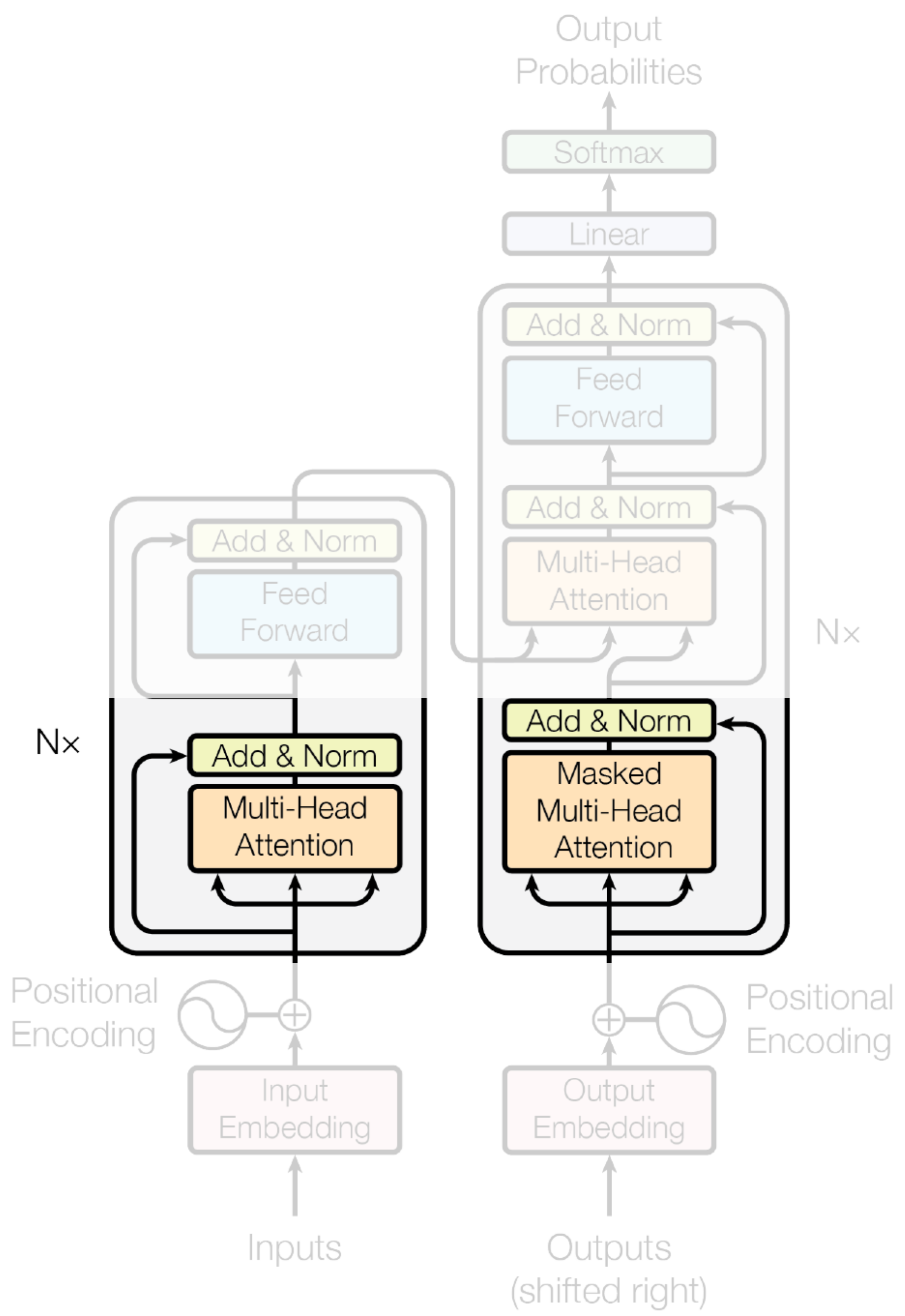


Encoder-decoder architectures

- More recent encoder-decoder transformers include T5, BART, and LaMDA
- Generally, these models are used for sequence-to-sequence mapping, such as translating between languages and summarizing text

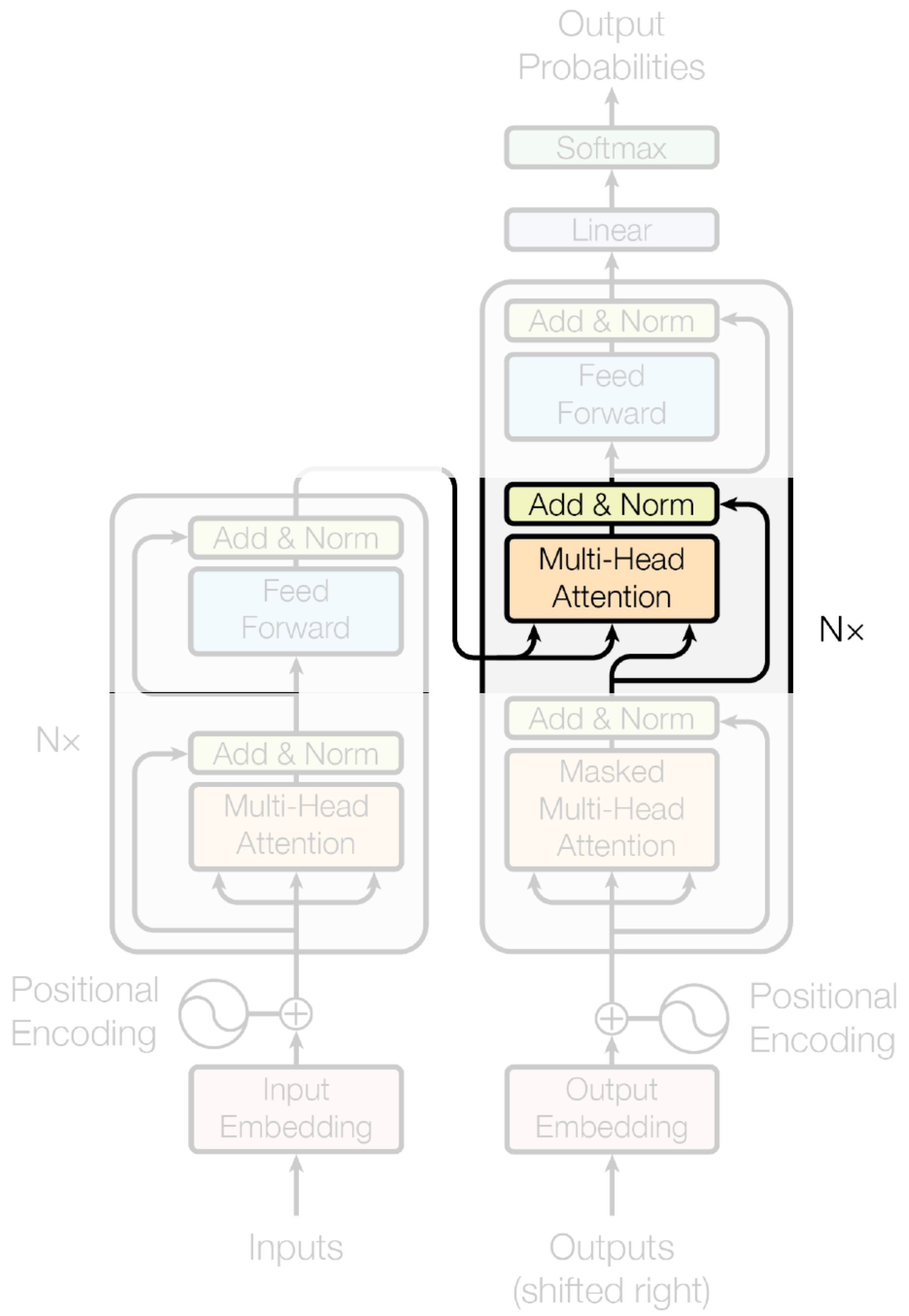
Self-attention

- In some parts of these networks, the queries, keys, and values all come from the same sequence
- Recall that queries and keys pass through different matrices, so a transformed query from a certain sequence element could be most like a transformed key from a different element
- Each element of the output can collect information from the whole sequence, contingent on the corresponding query



Cross-attention

- In other parts, queries come from the output sequence while keys and values come from the input sequence
- This allows the transformer to draw information from the whole input at each step in producing the output



The whole input may be important

I've really enjoyed **riding** recently because I have a new bike.

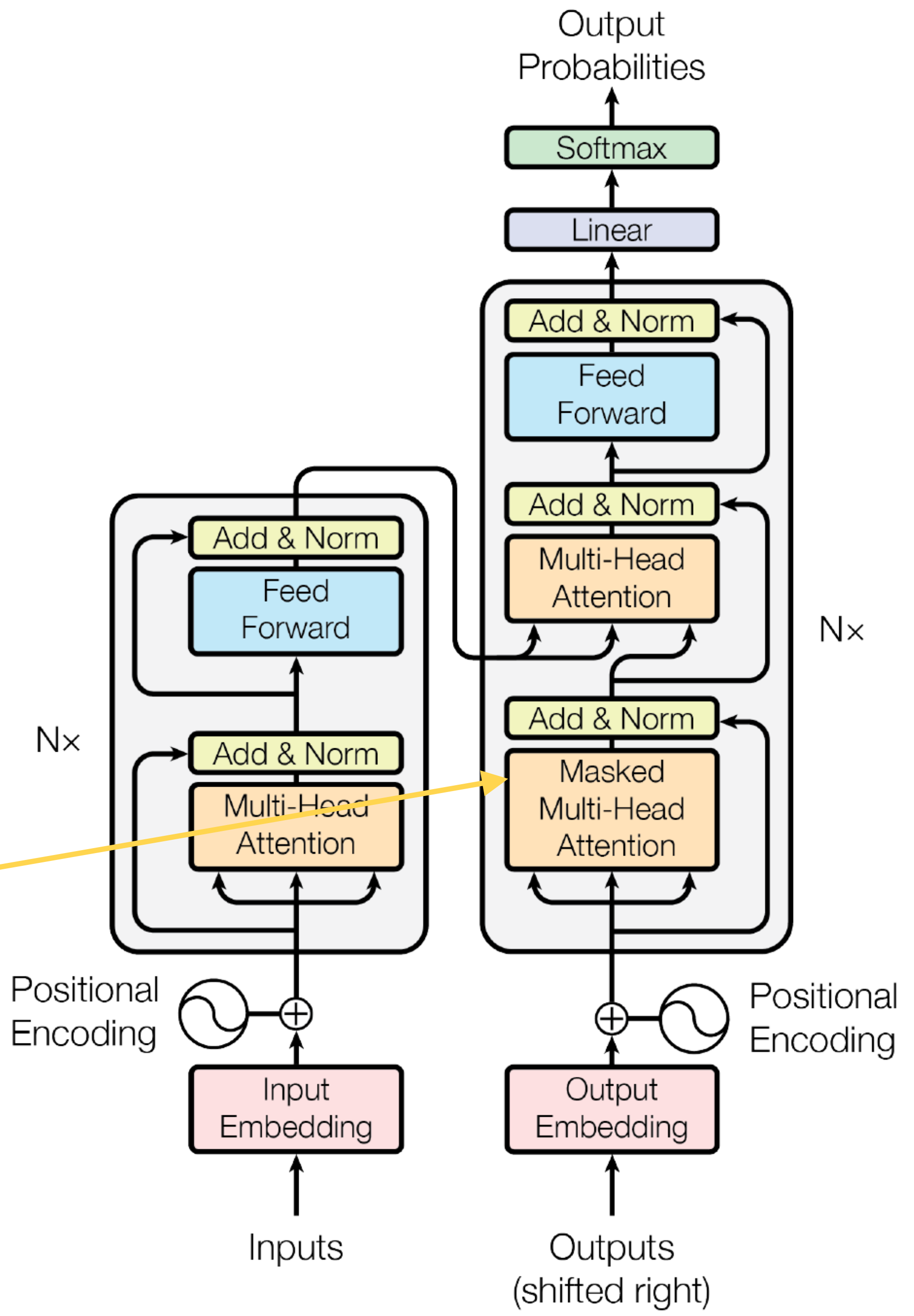
J'ai vraiment aimé **rouler** récemment parce que j'ai un nouveau vélo.

I've really enjoyed **riding** recently because I have a new horse.

J'ai vraiment aimé **monter à cheval** récemment parce que j'ai un nouveau cheval.

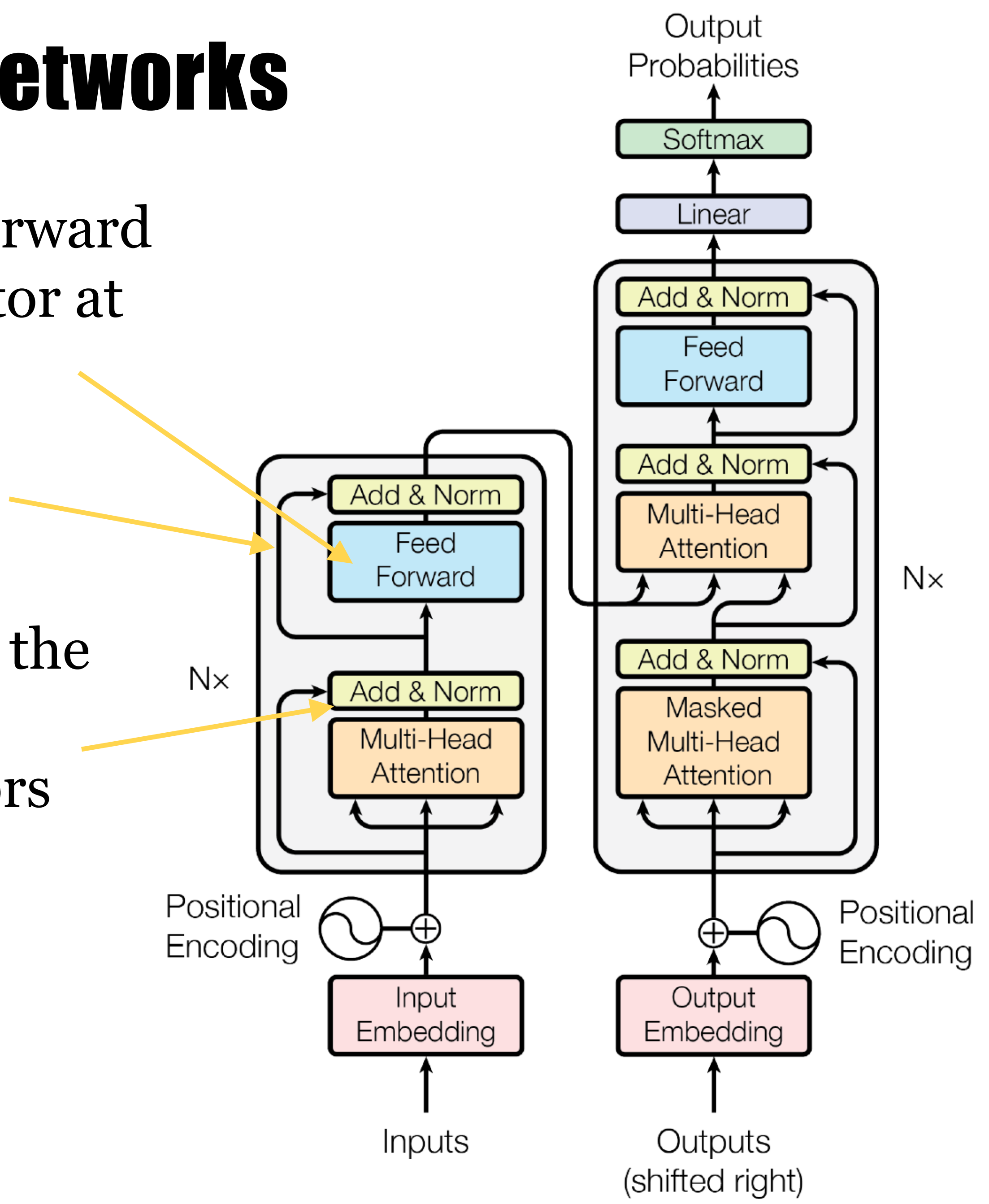
Masking

- During inference, the network gets a full input sequence, and then predicts each element of the output, one at a time
  - At each step, its previous outputs make up the output sequence
- During training, the decoder should only see the part of the output sequence up to but not including the element it is predicting
- The rest of the sequence is masked by setting the corresponding similarities to negative infinity



Other elements of transformer networks

- After the attention layers, a two-layer feedforward network is applied independently to the vector at each position
- There are residual connections around the attention layers and the feedforward layers
- Layer norm is used rather than batch norm; the difference is that the mean and standard deviation are calculated over the input vectors rather than over a batch



POSITION ENCODINGS ALLOW SPATIAL ATTENTION

Position encodings

- Nothing in the inner product of queries and keys is intrinsically sensitive to elements' positions in the sequence, but position can be important
- E.g., "People don't eat rotten fruit" has a different meaning than "Rotten people don't eat fruit"
- Transformers sum a position encoding with each token embedding, producing a sequence element that represents both the token and its position
- The position encoding allows an analogy of human spatial attention, in addition to feature attention

Sinusoidal position encoding

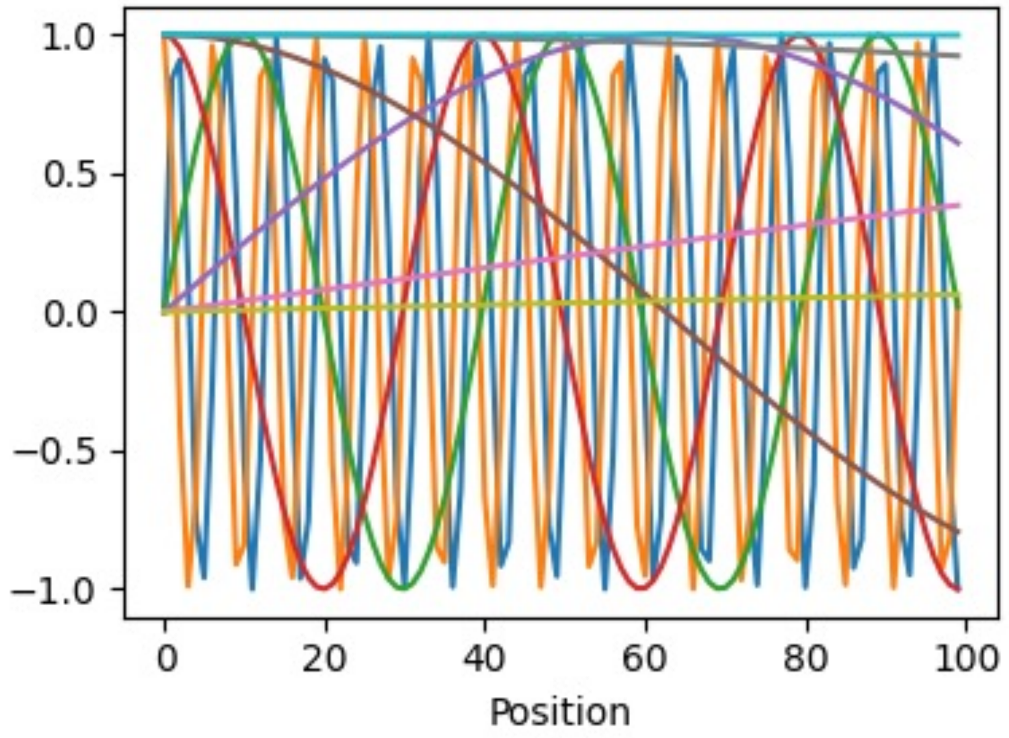
- Vaswani et al. used a sinusoidal position encoding
- For a given position k in the input sequence and encoding dimension d,

$$\boldsymbol{p}_{2i}(k) = \sin(k/10000^{2i/d})$$
$$\boldsymbol{p}_{2i+1}(k) = \cos(k/10000^{2i/d})$$

- A periodic encoding was used to allow the network to work on longer sequences than the training sequences
- For a fixed offset o, $\boldsymbol{p}(k+o)$ is a linear function of $\boldsymbol{p}(k)$, which may help the model attend to relative positions
  - Specifically, it is a bunch of rotations in different pairs of dimensions

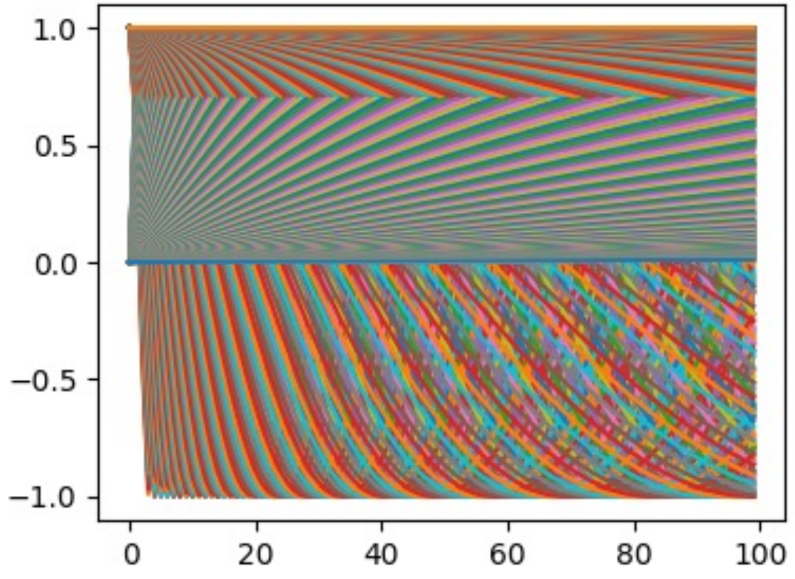
Sinusoidal position encoding

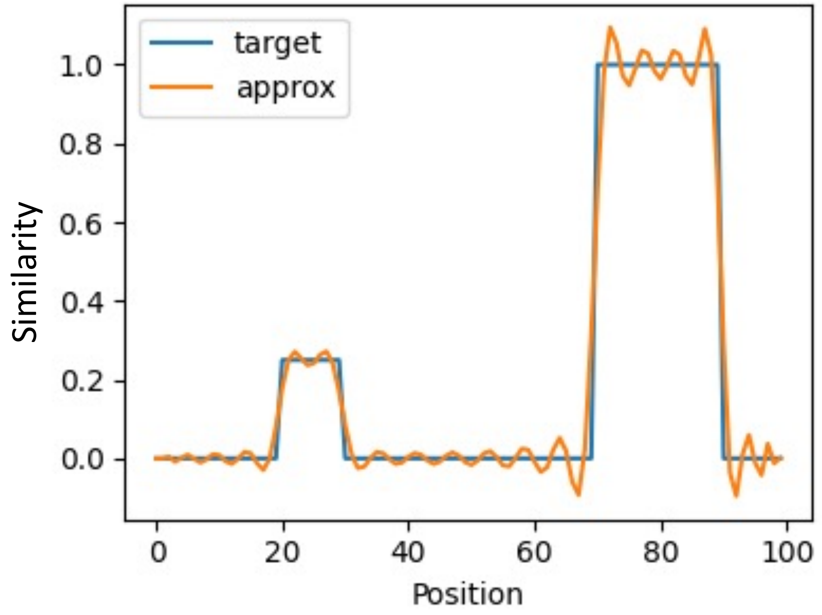
- In this plot the dimension is greatly reduced form typical values (e.g., 512 or greater) to 20 for visibility
- The longest period is 62,800, longer than sequences that are typically managed by transformer models



Sinusoidal position encoding

- With a dense encoding like this (512D), an attention head could potentially learn a family of flexible weighting functions related to a Fourier decomposition
- E.g., suppose we obtain the approximation on the right as $\boldsymbol{w}^T \boldsymbol{p}_i$, where $\boldsymbol{w}$ are weights in a linear regression of the target
- We would obtain the same position component of similarity with a certain token embedding $\boldsymbol{x}_j$ if $\boldsymbol{x}_j{}^T W_Q{}^T W_K = \boldsymbol{w}^T$





Learned position embeddings

- Some transformers learn position embeddings instead, beginning with randomly initialized encodings
- These have been found to perform similarly (e.g., Vaswani et al., 2017) or somewhat worse (e.g., Wang & Chen, 2020, *arXiv*) than sinusoidal encodings

Absolute vs. relative position embeddings

- Note that meaning is often affected by words at consistent *relative* positions, for example:
  - Adjectives often come right before nouns, as in “black cat and white dog”
  - Verbs are often between subjects and objects, as in “the provincial government is seeking an unlawful strike declaration”
- There may be advantages in explicitly considering the positions of various keys relative to the query, rather than their absolute positions
- Relative position encodings can be formulated in different ways, e.g., as in Shaw et al. (2018), Dai et al., (2019)
- Better results have been reported with relative than absolute position encodings in language translation (Shaw et al., 2018), but in general the difference is not definitive, and both methods continue to be used (Dufter et al., 2022)

Absolute vs. relative position encodings

- Consider self attention on a sequence $\mathrm{X} = \{\boldsymbol{x}_i, \dots, \boldsymbol{x}_n\}$
- Recall the standard attention mechanism involves the dot products of keys and values, both of which are linear mappings of the sequence elements,

$$\sqrt{d}s_{ij} = \boldsymbol{x}_i^T W_Q^T W_K \boldsymbol{x}_j$$

- Normally $\boldsymbol{x}_i$ consists of token and absolute position parts, $\boldsymbol{x}_i = \boldsymbol{t}_i + \boldsymbol{p}_i$
- We can decompose self attention with absolute position encoding as,

$$\sqrt{d}s_{ij}^{abs} = \boldsymbol{t}_i^T W_Q^T W_K \boldsymbol{t}_j + \boldsymbol{t}_i^T W_Q^T W_K \boldsymbol{p}_j + \boldsymbol{p}_i^T W_Q^T W_K \boldsymbol{t}_j + \boldsymbol{p}_i^T W_Q^T W_K \boldsymbol{p}_j$$

Relative position encodings

- From the last slide,

$$\sqrt{d}s_{ij}^{abs} = \boldsymbol{t}_i^T W_Q^T W_K \boldsymbol{t}_j + \boldsymbol{t}_i^T W_Q^T W_K \boldsymbol{p}_j + \boldsymbol{p}_i^T W_Q^T W_K \boldsymbol{t}_j + \boldsymbol{p}_i^T W_Q^T W_K \boldsymbol{p}_j$$

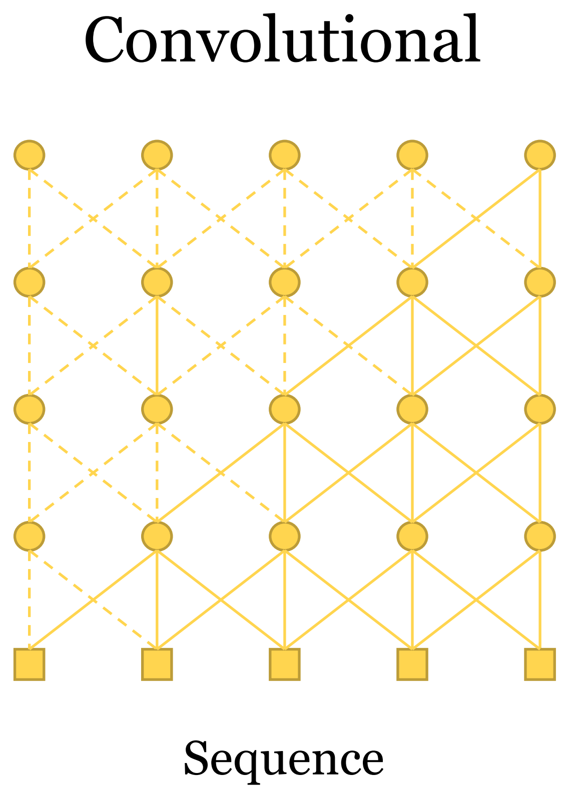
- To produce a relative position encoding, Dai et al. (2019, *arXiv*) replace the above with,

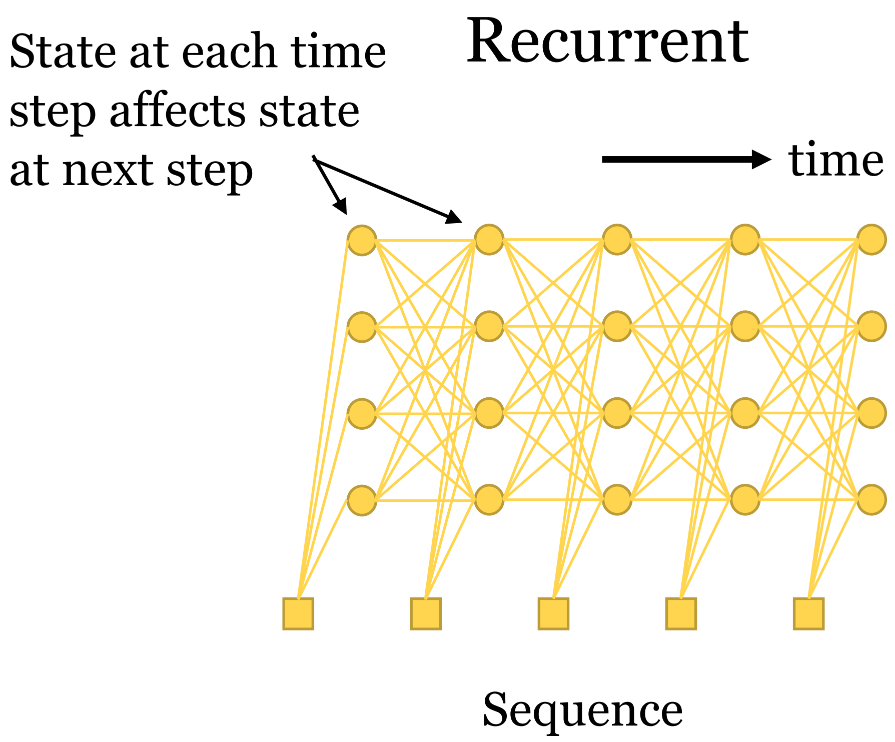
$$\sqrt{d}s_{ij}^{rel} = \boldsymbol{t}_i^T W_Q^T W_{K,t} \boldsymbol{t}_j + \boldsymbol{t}_i^T W_Q^T W_{K,r} \boldsymbol{r}_{i-j} + \boldsymbol{u}^T W_{K,t} \boldsymbol{t}_j + \boldsymbol{v}^T W_{K,r} \boldsymbol{r}_{i-j}$$

- Differences:
  - Elements of $\boldsymbol{r}_{i-j}$ are sinusoidal functions of $j - i$
  - $W_Q \boldsymbol{p}_i$ is replaced with new learned vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ in the 3rd and 4th terms
  - Separate key mapping matrices are used for tokens and positions
- The four new terms correspond to 1) content-based addressing; 2) content-based relative positional bias; 3) global content bias; 4) global relative position bias

TRANSFORMERS PROCESS ELEMENTS OF A SEQUENCE IN PARALLEL

Convolutional vs. recurrent networks for sequences





Convolutional neurons can run in parallel

- Neurons in the same layer of a convolutional network do not depend on each other, so they can run at the same time on parallel hardware
  - The number of sequential steps in a forward pass equals the network depth via the longest path
- In contrast, neurons in a recurrent layer depend on their own outputs in previous time steps
  - The number of sequential steps depends on the sequence length

Recurrent networks can access long-range context

- Recurrent networks usually use multiplication to determine when to use, retain, or discard older information
- This can allow information to be retained over long sequences until it is needed
- In contrast, older information has a declining influence in convolutional networks, and disappears altogether after a certain number of sequence elements (number of steps kernel goes back in time x depth)

Transformers have both advantages

- Transformers are not recurrent, so all responses in a layer can be computed in parallel during training
- Transformer neurons in each layer receive input from all elements of the sequence in the previous layer, so they can access long-range context
- The same is true of a fully connected network, but transformers are more parameter-efficient
- E.g., suppose an embedding dimension of 100 and a sequence length of 100
  - In a fully connected network, each layer would have $(100\times 100)^2 = 100{,}000{,}000$ weights
  - A transformer attention layer with one head would have $3\times 100^2 = 30{,}000$ weights

Teacher forcing and non-parallel inference

- When a transformer's decoder is used for sequence generation, it receives its own previous outputs as input, so the predictions must be sequential
- When training the decoder, each element can be predicted in parallel
  - Each element doesn't receive the network's previous predictions as input, but rather the correct previous elements as input, so that mistakes early in the sequence to not propagate through the whole sequence
  - For each query, later keys/values in the sequence are masked
- Feeding in labels in this way is called "teacher forcing" and this method has long been used with recurrent networks to make it easier to learn sequence generation

TRANSFORMERS CAN OMIT THE ENCODER OR DECODER

Types of transformer architecture

- Encoder/decoder
  - Good at sequence-to-sequence mapping, e.g., translation, summarization
- Encoder
  - Good at decisions about text, e.g., classification, finding spans of text that answer reading comprehension questions
- Decoder
  - Good at sequence generation, e.g., completing text

Decoder-only architectures

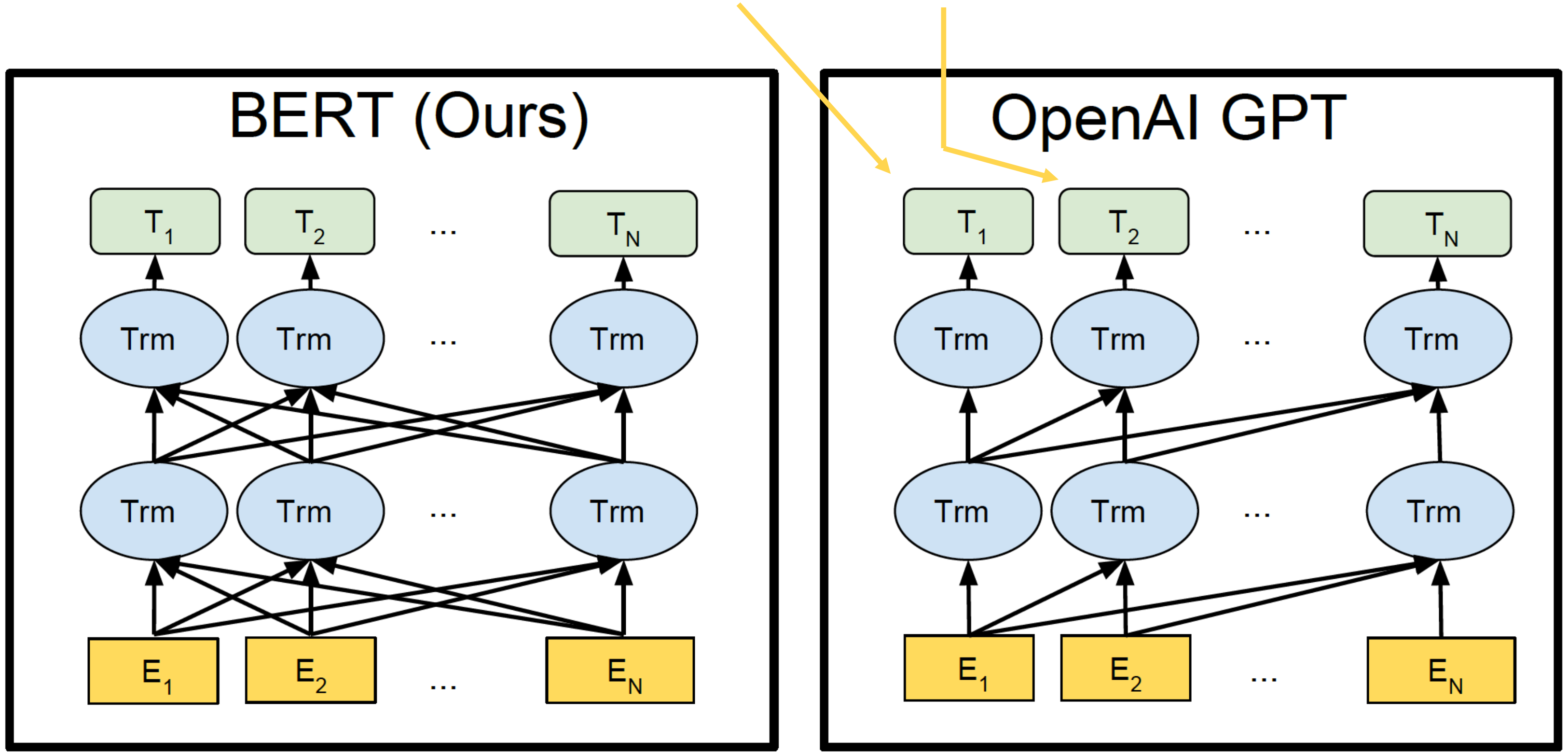
- Introduced by Liu et al., 2018, *ICLR* and used in more recent models such as GPT-3 (Brown et al., 2020) and PaLM (Chowdhery et al., 2022)
- To apply to a sequence-to-sequence task such as language translation, the mapping $(m^1, m^2, \dots, m^n) \longmapsto (y^1, y^2, \dots, y^\eta)$ is expressed as a single sequence, $(w^1, w^2, \dots, w^{n+\eta+1}) = (m^1, m^2, \dots, m^n, \delta, y^1, y^2, \dots, y^\eta)$ and the next token is predicted given previous tokens
- Advantages over encoder-decoder architecture:
  - Reduces the number of parameters by roughly half
  - Eliminates possibly redundant language learning in encoder and decoder

Encoder-only architectures

- Introduced by Devlin et al., 2019 in the BERT (Bidirectional Encoder Representations from Transformers) model
- Applied to various language tasks other than sequence generation (e.g., sentiment analysis, judging whether two sentences are semantically equivalent, and answering questions about passages of text)
- This approach has the same advantages over encoder-decoder transformers as the decoder-only approach, but it is less focused on text generation
- An advantage over decoder-only transformers in some contexts is that it allows use of context from later in the sequence
  - Not useful for sequence generation, but useful for other tasks such as named entity recognition
  - E.g., “Do you know **Kyle** very well?” “Yes, I grew up close to there.”

No contextual representation

Poor contextual representation



Devlin et al., 2019

TRANSFORMERS PERFORM SELF-SUPERVISED LEARNING WITH LARGE DATASETS

Causal language modelling

- This task consists of predicting the next word in a sequence
- This can be performed as a self-supervised task based on large collections of text
  - E.g., the first GPT network was trained on BookCorpus (Zhu et al., 2015, *ICCV*), a collection of 11K free books with about one billion words
- Given a sequence of tokens $\mathcal{U} = \{u_1, \dots, u_n\}$ the network is trained by minimizing,

$$-\sum_i \log P(u_i | u_{i-k}, \dots, u_{i-1}; \theta)$$

  where k is the context window size

- The input is $\{u_{i-k}, \dots, u_{i-1}\}$, the target is u_i, and the network outputs $P(u)$

Masked language modelling

- The encoder-only BERT model (Devlin et al., 2019) introduced masked language modelling to transformers
- Given a full sequence, 15% of tokens are masked at random using a new [MASK] token
- The network predicts these tokens at each masked location, using both forward and backward context
- A concern with this approach is that the [MASK] token doesn’t appear in other tasks the network is applied to (in fine tuning or inference)
- To reduce this gap, the [MASK] token is only used for 80% of masked/predicted elements, and random (10%) or correct (10%) tokens are used for the others

TRANSFORMERS CREATE CONTEXTUALIZED EMBEDDINGS

Word embeddings

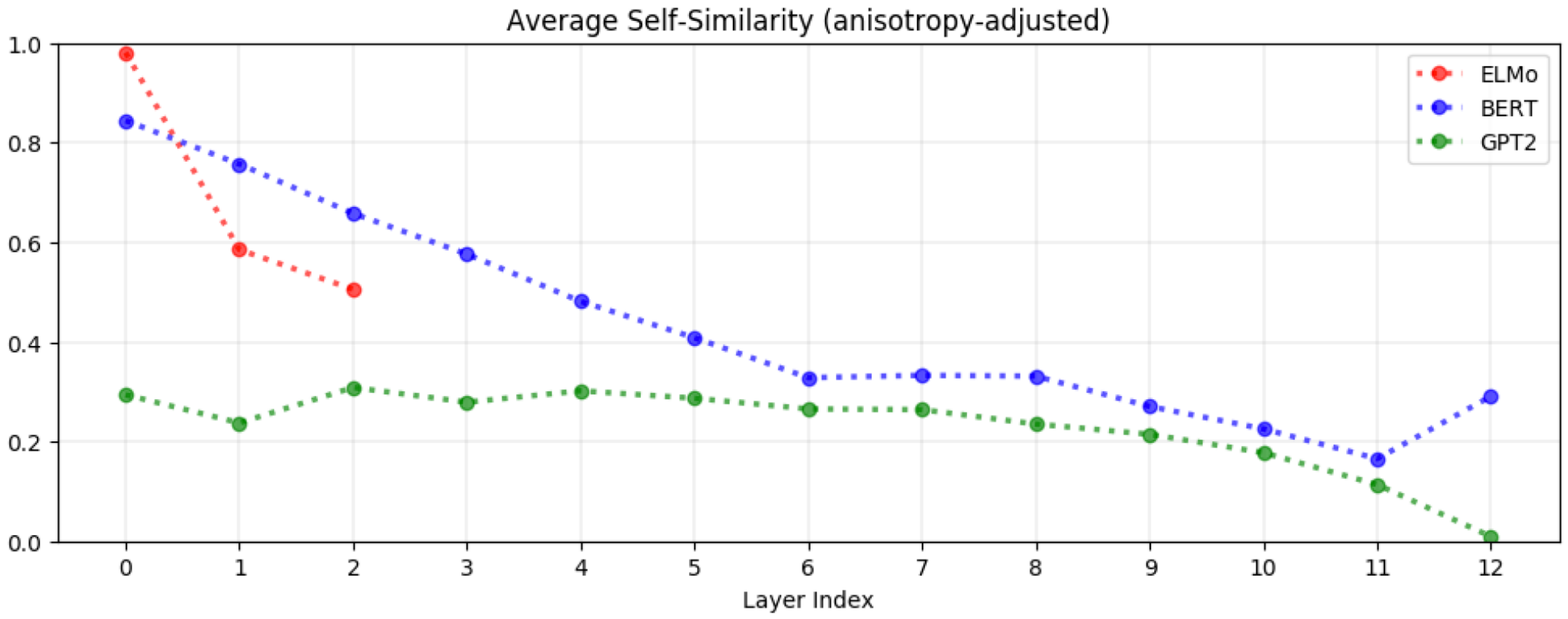
- Recall that transformers assign a learned vector to each token
- This approach builds on previous methods of assigning vectors to words, called “word embeddings”
- For example, one earlier method is called “Word2Vec”
  - This method is based on a two-layer neural network that is trained to predict segments of text
  - A one-hot code for a word is fed into the network, and the network is trained to predict the surrounding words
  - After training, the hidden-layer representation of each word is its embedding
- Such approaches lead to similar words having similar vectors
- This is important as it allows a machine learning system to apply what it learns about a word to other similar words

Contextual word embeddings

- A limitation of classical word embeddings is that they don’t handle words that have multiple meanings
  - These include homonyms, in which the same word is used for two distinct concepts, e.g., “bat” the animal vs. “bat” the striking action
  - These also include polysemes, which have different but related meanings depending on context, e.g., “bat” the striking action vs. “bat” the club that is typically used to bat a baseball
- To work around this limitation, some models have concatenated context embeddings with word embeddings
- Each layer of a transformer can be seen as producing a contextual embedding of each token that can incorporate context from the rest of the sequence

Contextual embeddings

- In transformer models including BERT and GPT-2, embeddings for a given token in different contexts become less similar in deeper layers



Ethayarajh (2019), EMNLP

Summary

1. Transformers operate on sequences of vectors
2. Transformers use soft attention to weigh elements of a sequence
3. Queries, keys, and values undergo linear mappings
4. Attention is multi-headed
5. Encoder-decoder transformers can perform sequence-to-sequence mapping
6. Position encodings allow spatial attention
7. Transformers process elements of a sequence in parallel
8. Transformers can omit the encoder or decoder
9. Transformers perform self-supervised learning with large datasets
10. Transformers create contextualized embeddings

References

- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.
- Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., ... & Fiedel, N. (2022). Palm: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311.
- Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. arXiv preprint arXiv:1901.02860.
- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
- Dufter, P., Schmitt, M., & Schütze, H. (2022). Position information in transformers: An overview. Computational Linguistics, 48(3), 733-763.
- Ethayarajh, K. (2019). How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. arXiv preprint arXiv:1909.00512.
- Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., & Shazeer, N. (2018). Generating wikipedia by summarizing long sequences. arXiv preprint arXiv:1801.10198.
- Sennrich, R., Haddow, B., & Birch, A. (2015). Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909.
- Shaw, P., Uszkoreit, J., & Vaswani, A. (2018). Self-attention with relative position representations. arXiv preprint arXiv:1803.02155.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
- Wang, Y. A., & Chen, Y. N. (2020). What do position embeddings learn? an empirical study of pre-trained language model positional encoding. arXiv preprint arXiv:2010.04903.
- Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... & Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144.
- Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In Proceedings of the IEEE international conference on computer vision (pp. 19-27).

UNIVERSITY OF WATERLOO