S&DS 365 / 665

Intermediate Machine Learning

Attention and Transformers

November 29

Yale

Reminders

- Quiz 5 (last!) today at 10:30am
  - 48 hours/20 mins
  - RL, HMMs, RNNs, GRUs, Seq2seq
- Assn 5 out; due next Wednesday
- Notes posted on mixtures, HMMs and Kalman filters

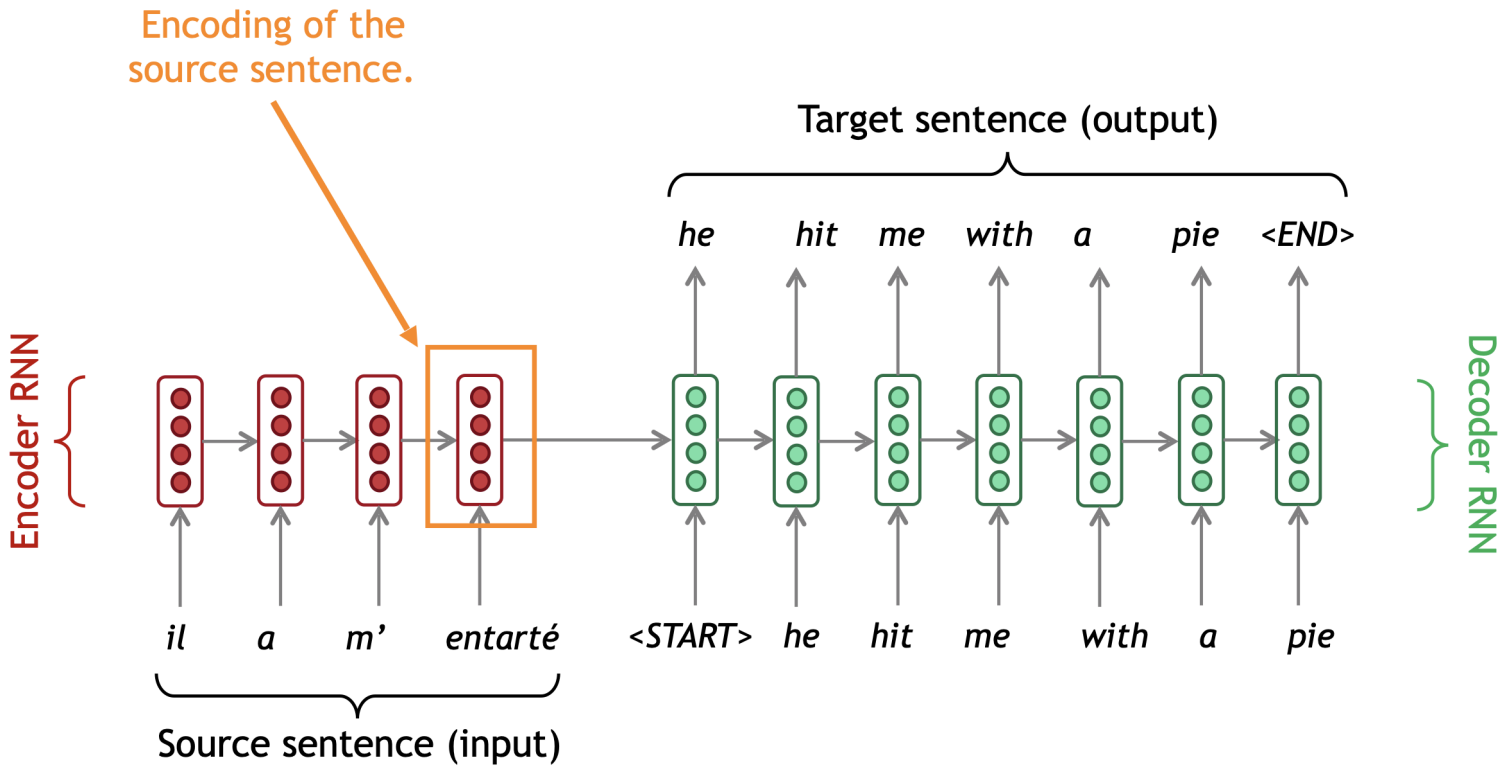
For Today

Sequence models

- Attention mechanisms
- Transformers

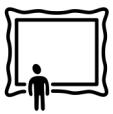
Recall: Sequence-to-sequence (seq2seq) models

Encoding of source is used together with decoder state. Results in a "bottleneck" problem

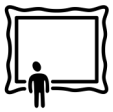


Important modification: Attention

Attention/Alignment: Seq2seq

 On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

Attention/Alignment: Seq2seq



On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

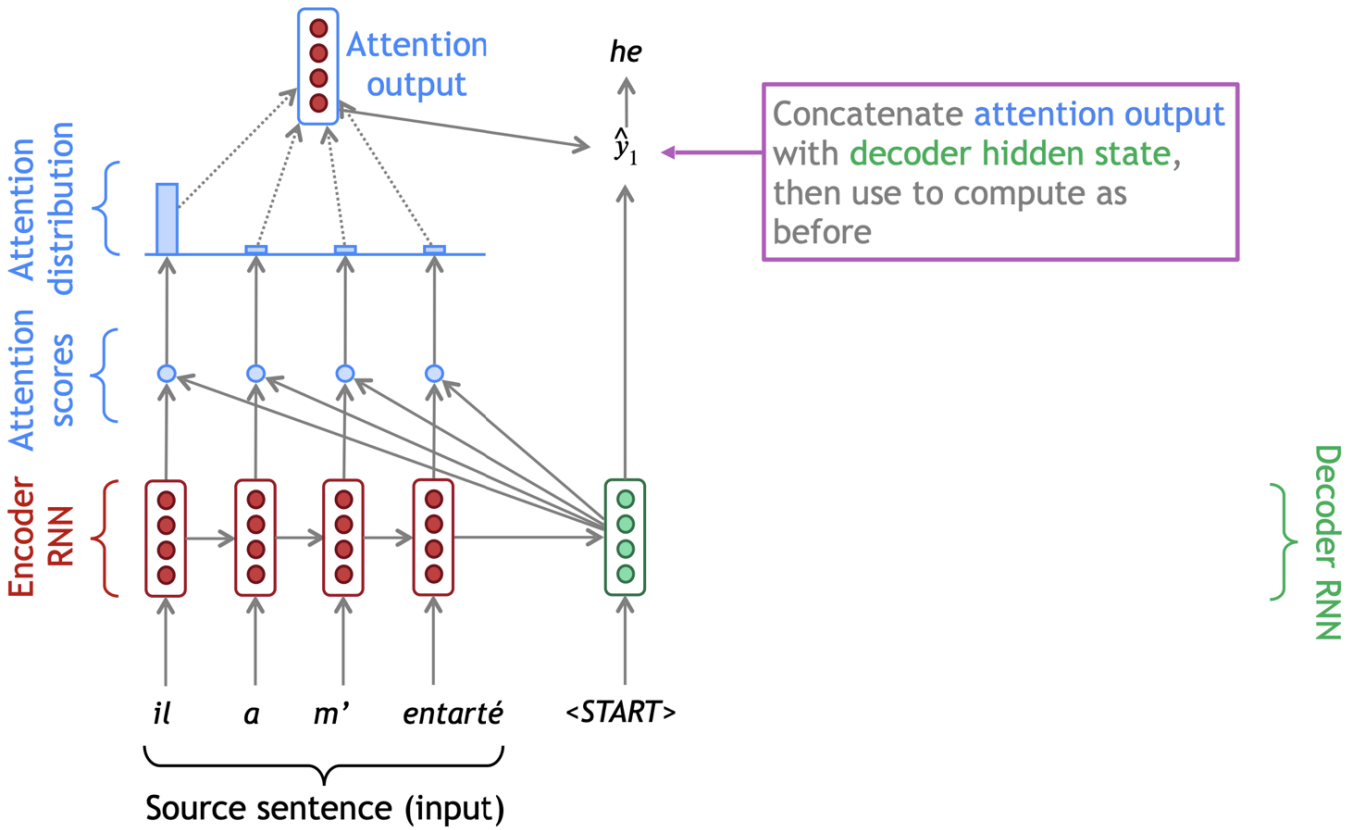
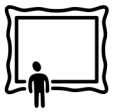


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Attention/Alignment: Seq2seq



On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

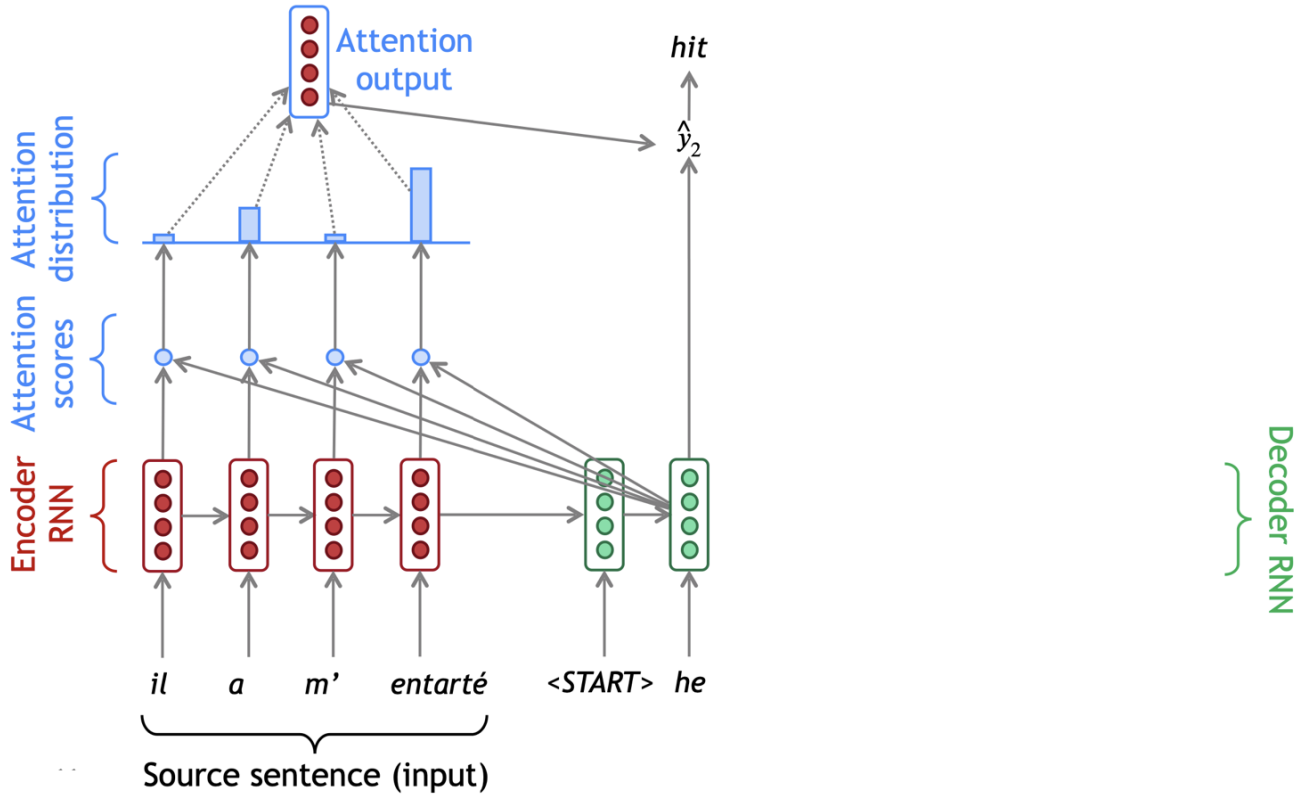
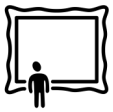


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Attention/Alignment: Seq2seq



On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

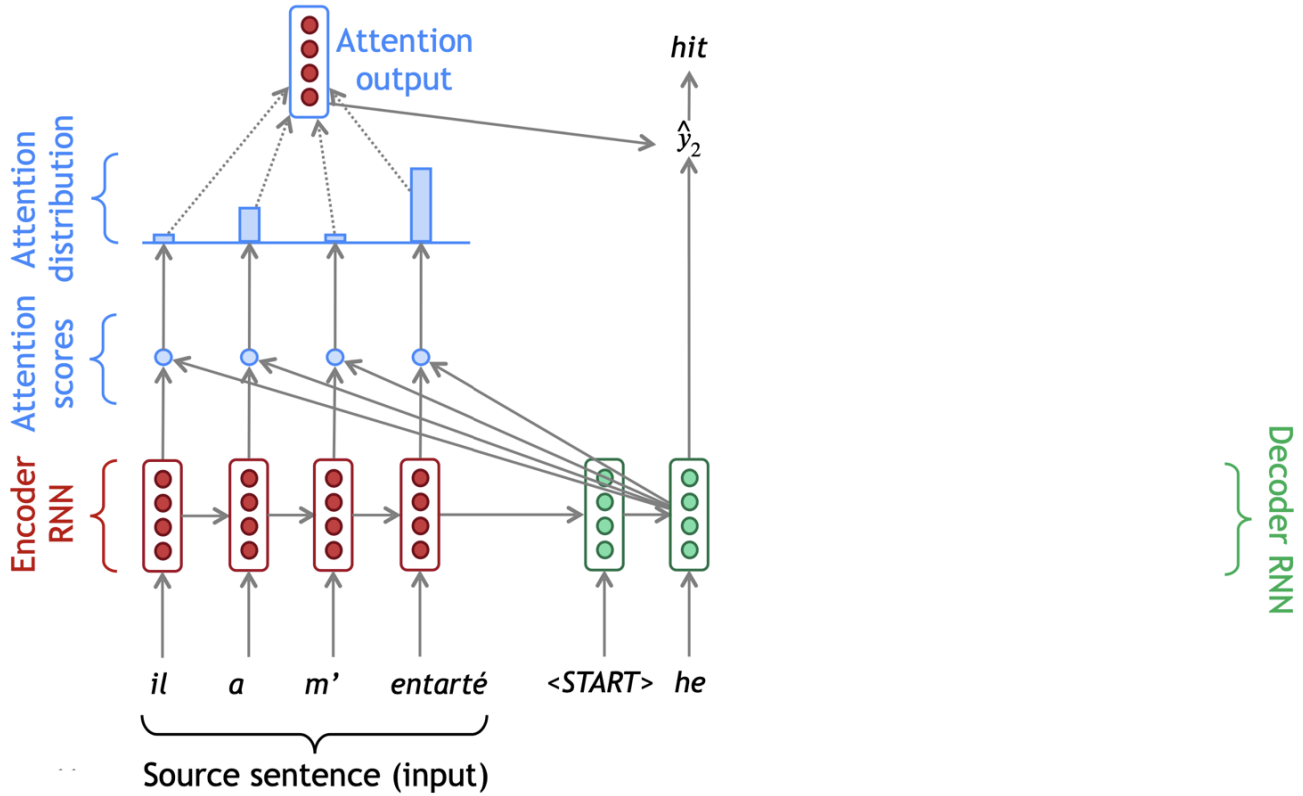
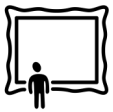


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Attention/Alignment: Seq2seq

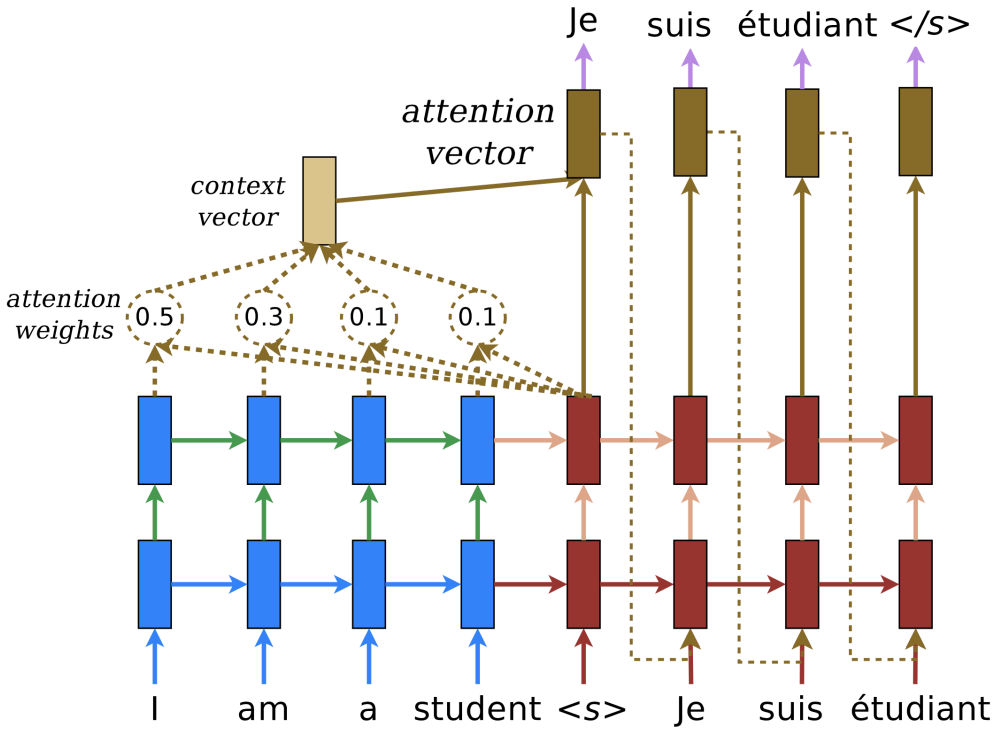
 On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

Decoder state h_t then evolves as follows:

$$h_{t+1} = \tanh\left(W_{hh}h_t + W_{xh}x_t + W_{ah}a_t + b_h\right)$$

where a_t is the attention output

Attention/Alignment: Seq2seq



another illustration, from PML

Attention/Alignment: Seq2seq

Attention is also important for other types of data

The two elephants played with the orange ball

The two **elephants** played with the orange ball

The two **elephants** played with the orange ball

The two elephants played with the **orange** ball

The two elephants played with the orange **ball**

Terminology

Different ways of referring to this:

- `played` is “aligned with” `ball`
- `ball` “attends to” `played`

Term “attention” is anthropomorphic (or cognopomorphic)

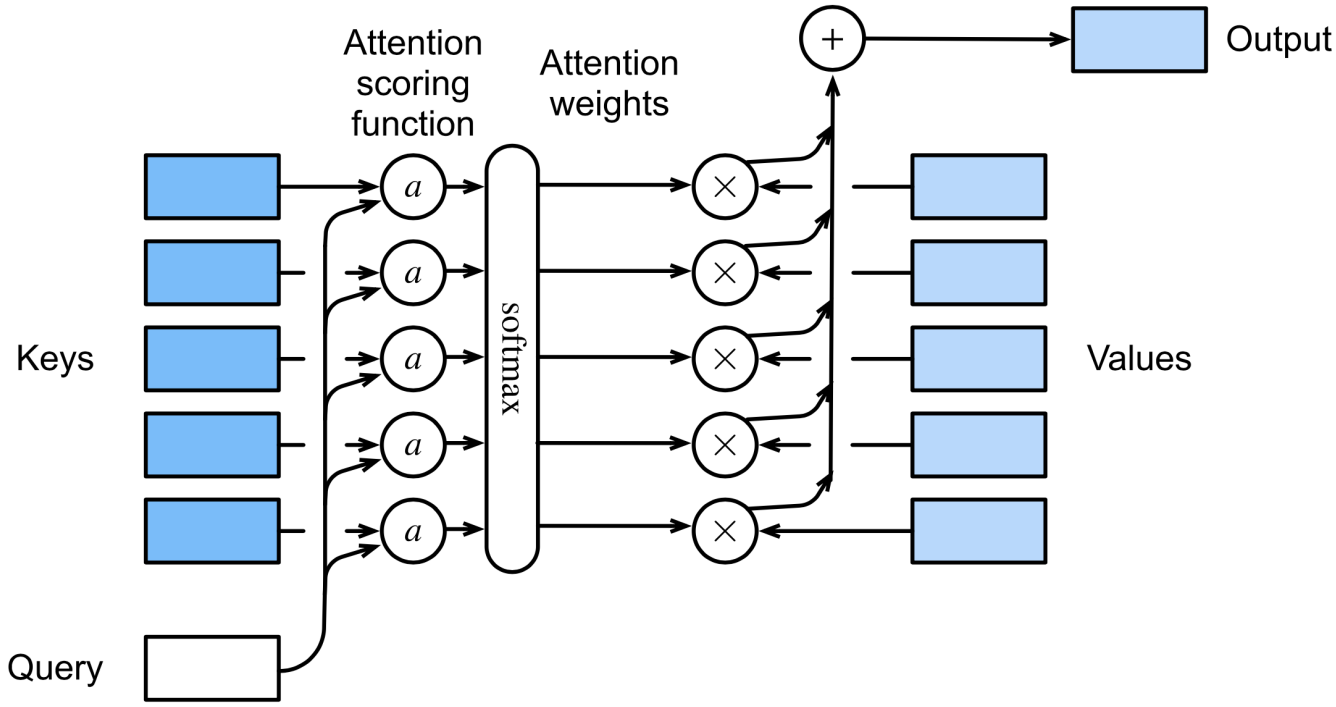
It’s a crude abstraction of notions of attention in cognition, just as feedforward nets and activation functions are crude abstractions of biological mechanisms in the brain.

Attention in terms of query/key/values

- Attention mechanisms give a flexible module for building new features:
  - We have a set of m feature vectors or values $V \in \mathbb{R}^{m \times v}$
  - The model dynamically chooses which to use
  - based on how similar a query vector $q \in \mathbb{R}^{q}$ is to a set of m keys $K \in \mathbb{R}^{m \times k}$.
  - If q is most similar to key i, then we use value (feature) v_i.

Exercise: Try to think of convolutional neural networks in this way (see PML, §15.5)

Attention in terms of query/key/values



Exercise: Try to think of convolutional neural networks in this way (see PML, §15.5)

Attention in terms of query/key/values

In formulas:

$$\begin{aligned}\text{Attn}(q, \{(k_1, v_1), \ldots (k_m, v_m)\}) &= \text{Attn}(q, (k_{1:m}, v_{1:m})) \\ &= \sum_{i=1}^{m} w_i(q, k_{1:m}) v_i\end{aligned}$$

where weights w_i are softmax of attention scores:

$$w_i(q, k_{1:m}) = \frac{\exp(a(q, k_i))}{\sum_{j=1}^{m} \exp(a(q, k_j))}$$

Exercise: Try to think of convolutional neural networks in this way (see PML, §15.5)

Dot product attention

Attention scores are computed as

$$a(q, k_i) = (W_Q q)^T (W_K k_i)$$

and then the attention weights are the softmax:

$$\text{Attn}(q, k_{1:m}, v_{1:m}) = \text{Softmax}\left(a(q, k)\right) v$$

The matrices W_Q and W_K allow comparison of query and keys from different "languages"—-French and English or images and words

Moreover, there are multiple attention "heads," each one with different learned transformations W_Q and W_K

Kernel regression as attention

Recall the kernel regression estimator:

$$\widehat{m}(x) = \sum_{i=1}^{n} w(x, x_{1:n}) y_i$$

Here the attention scores (for Gaussian kernel) are

$$a(x, x_i) = -\frac{1}{2h^2} \|x - x_i\|^2$$

Kernel regression as attention

Recall the kernel regression estimator:

$$\widehat{m}(x) = \sum_{i=1}^{n} w(x, x_{1:n}) y_i$$

query:	test point x
keys:	data $x_1, \ldots, x_n$
values:	responses $y_1, \ldots, y_n$

Nonparametric – number of keys grows with size of data

Dot product attention

Note that if x_i and x have a fixed norm then the attention scores are just scaled dot products:

$$a(x, x_i) = \frac{1}{h^2} x^T x_i$$

Dot product attention is

$$a(q, k) \equiv q^T k = (W_Q q)^T (W_K k) \in \mathbb{R}$$

In matrix notation

$$\text{Attn}(Q, K, V) = \underbrace{\text{Softmax}\left(QK^T\right)}_{\text{weights}} V$$

Often scaled by $\sqrt{d}$ keeps variance constant

Dot product attention

Note that if x_i and x have a fixed norm then the attention scores are just scaled dot products:

$$a(x, x_i) = \frac{1}{h^2} x^T x_i$$

Dot product attention is

$$a(q, k) \equiv q^T k = (W_Q q)^T (W_K k) \in \mathbb{R}$$

In matrix notation

$$\text{Attn}(Q, K, V) = \underbrace{\text{Softmax}\left(QK^T\right)}_{\text{weights}} V$$

Attention is tantamount to learning the kernel!

Often scaled by $\sqrt{d}$ keeps variance constant

Attention/Alignment: Seq2seq

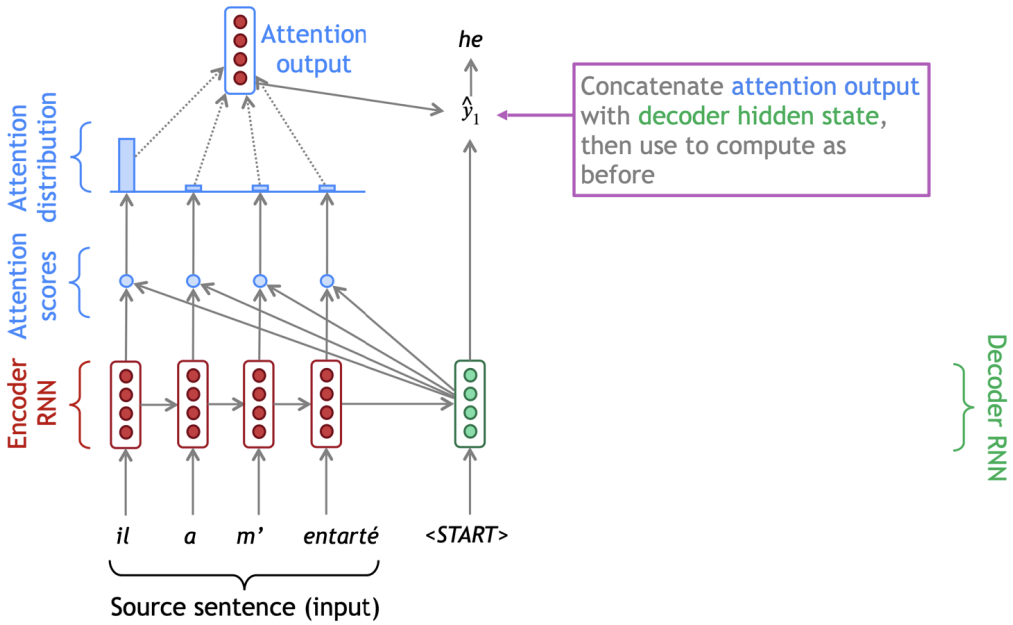


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

query: green state

keys: red states

values: red states

Attention/Alignment: Seq2seq

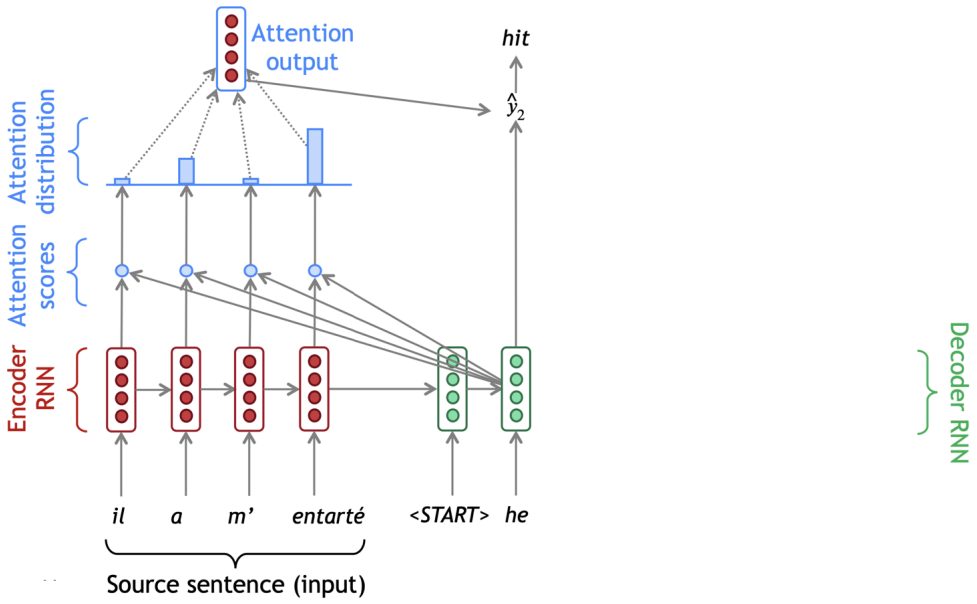


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

query: green state

keys: red states

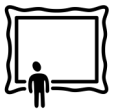
values: red states

Transformers

The current state-of-the-art for sequence modeling is based on *transformers*

- Attention is the key ingredient
- Rather than processing sequences word-by-word, transformers handle larger chunks of text at once
- The separation between words matters less

Latent variables (and lack thereof)



Before diving into the details of transformers, let's revisit latent variable models versus deterministic transformations

HMMs are mixture models

The HMM state evolves stochastically, resulting in a mixture model:

$$p(x_1, x_2 \ldots, x_T) = \sum_{s_1, s_2, \ldots, s_T} p(s_1, \ldots s_T) \prod_t p(x_t \mid s_t)$$

$$= \sum_s p(s) \prod_t p(x_t \mid s_t)$$

RNNs are not mixture models

The RNN state evolves *deterministically*:

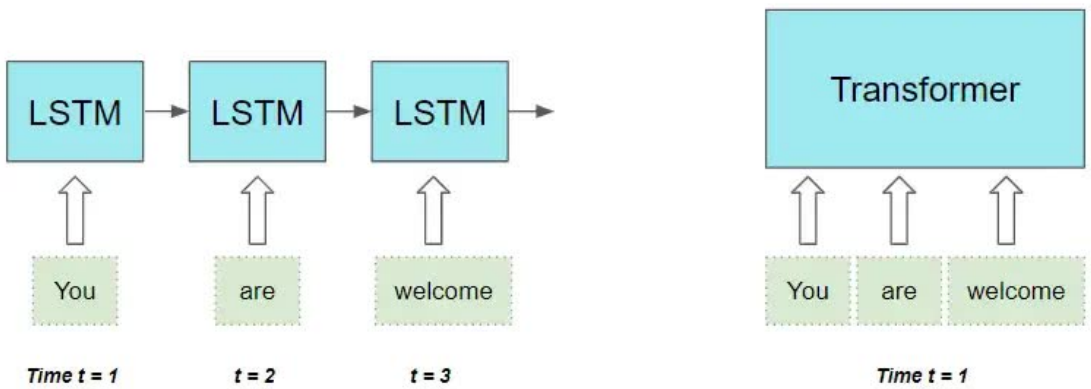
$$p(x_1, x_2 \ldots, x_T) = \prod_t p(x_t \mid s_t), \quad s_t = f(s_{t-1}, x_{t-1})$$

RNNs are not mixture models

The RNN state evolves *deterministically*:

$$p(x_1, x_2 \ldots, x_T) = \prod_t p(x_t \mid s_t), \quad s_t = f(s_{t-1}, x_{t-1})$$

Transformers, are similar — but scrap Markov structure



Combining attention vectors

- Different attention vectors are stacked up on top of each other
- Like channels and feature maps in a CNN
- Words “hungry” and “sweet” are predicted using all of them

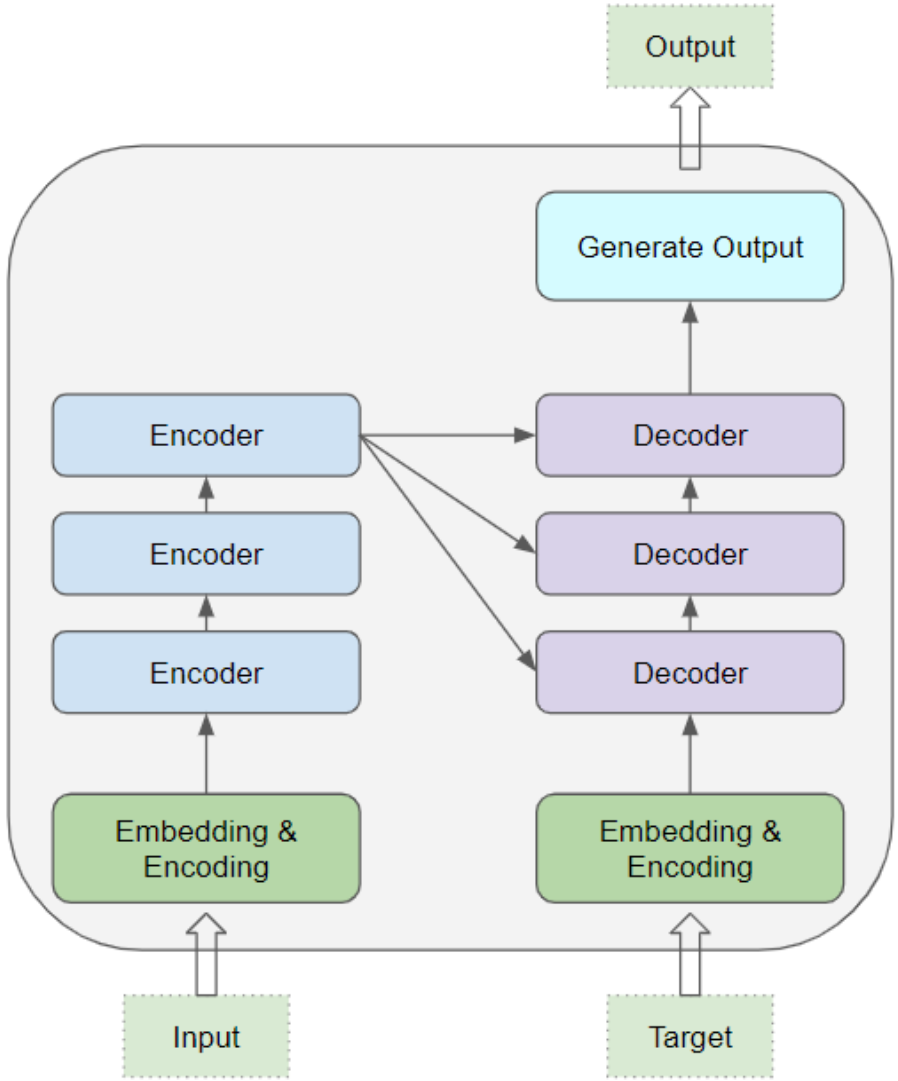
Overview of transformers

Following illustrations are from “Transformers explained visually” by Ketan Doshi, and from Probabilistic Machine Learning (PML) by Kevin Murphy

---

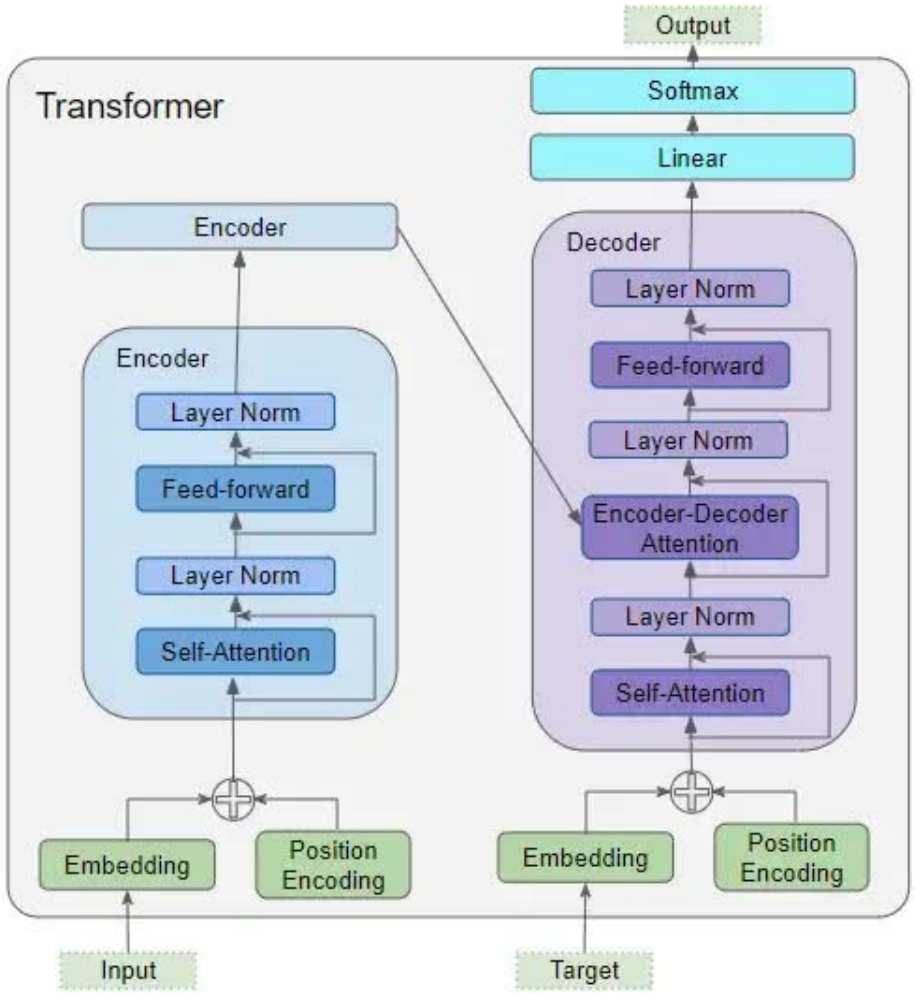
https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Transformer architecture



Transformer architecture

With one encoder and decoder module it would look like this:



Transformer architecture

A transformer is a seq2seq model that uses self-attention for the encoder and decoder rather than an RNN. The encoder and decoder use a series of transformations, each of which uses multi-headed attention

Input embeddings

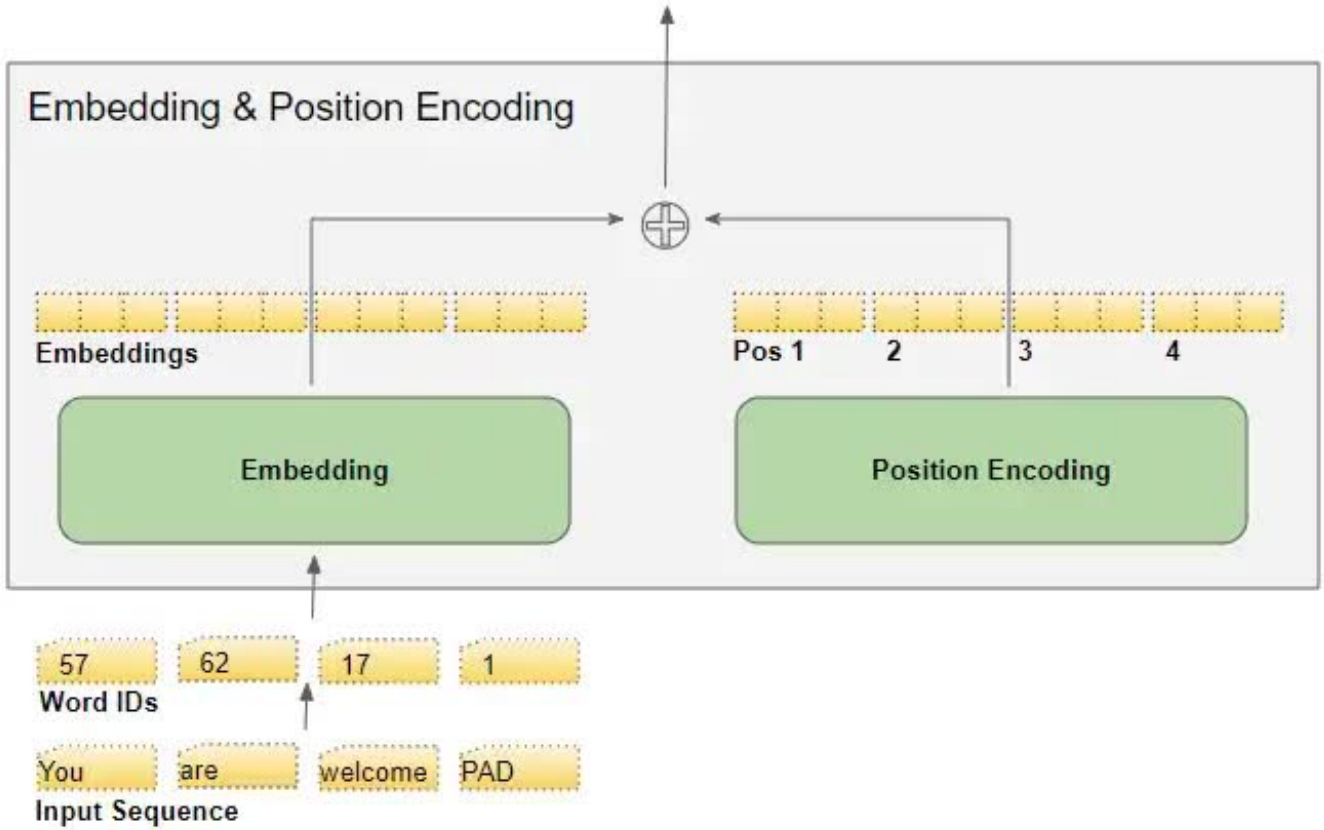
Source/target sequences are first passed through an embedding layer and positional encoding layer

Input embeddings

Source/target sequences are first passed through an embedding layer and positional encoding layer

Input embeddings

Source/target sequences are first passed through an embedding layer and positional encoding layer



Positional encoding

Positional encoding is an interesting way of getting position information into the model

Positional encoding

- Positions are integers, but neural nets can't handle integers
- Could use binary encoding; fixed precision
- Transformers use a sine/cosine basis

Positional encoding

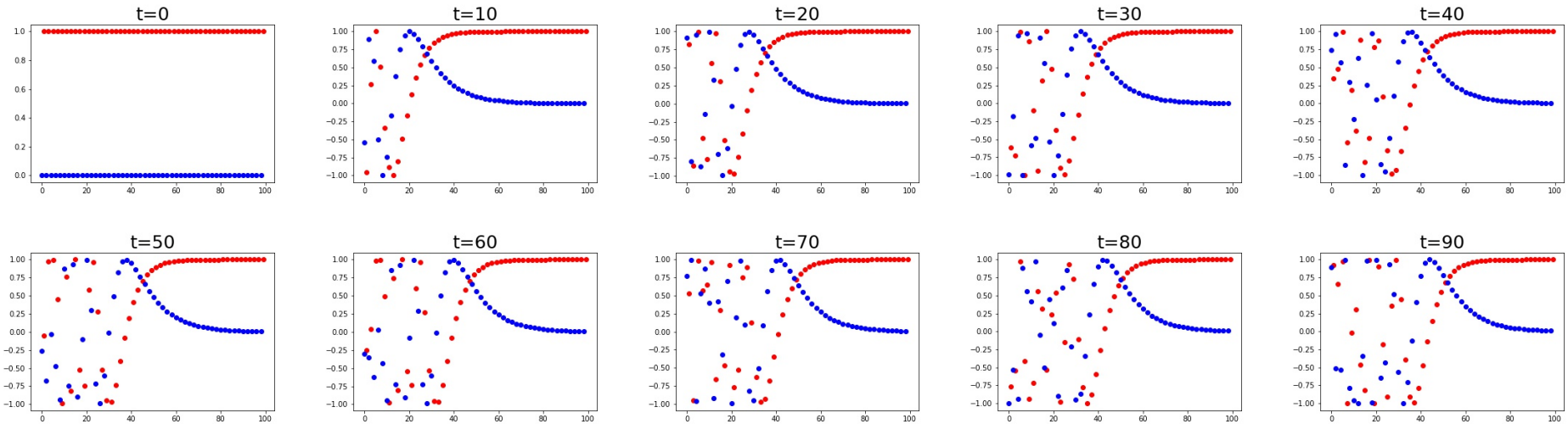
For the tth word in the sequence, components are

$$\mathrm{PE}_{(t,2i)} = \sin\left(\frac{t}{C^{2i/d}}\right)$$

$$\mathrm{PE}_{(t,2i+1)} = \cos\left(\frac{t}{C^{2i/d}}\right)$$

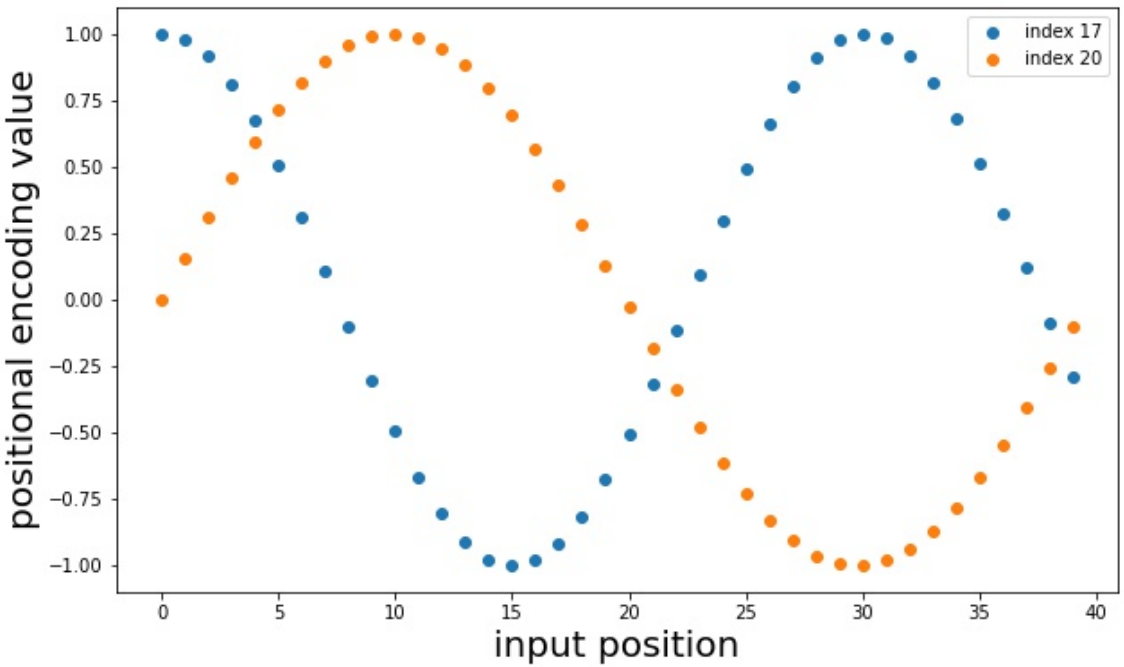
where d is the embedding dimension and $C = 10{,}000$ is a maximum sequence length

Positional encoding



100-dimensional embedding vectors for different positions

Positional encoding



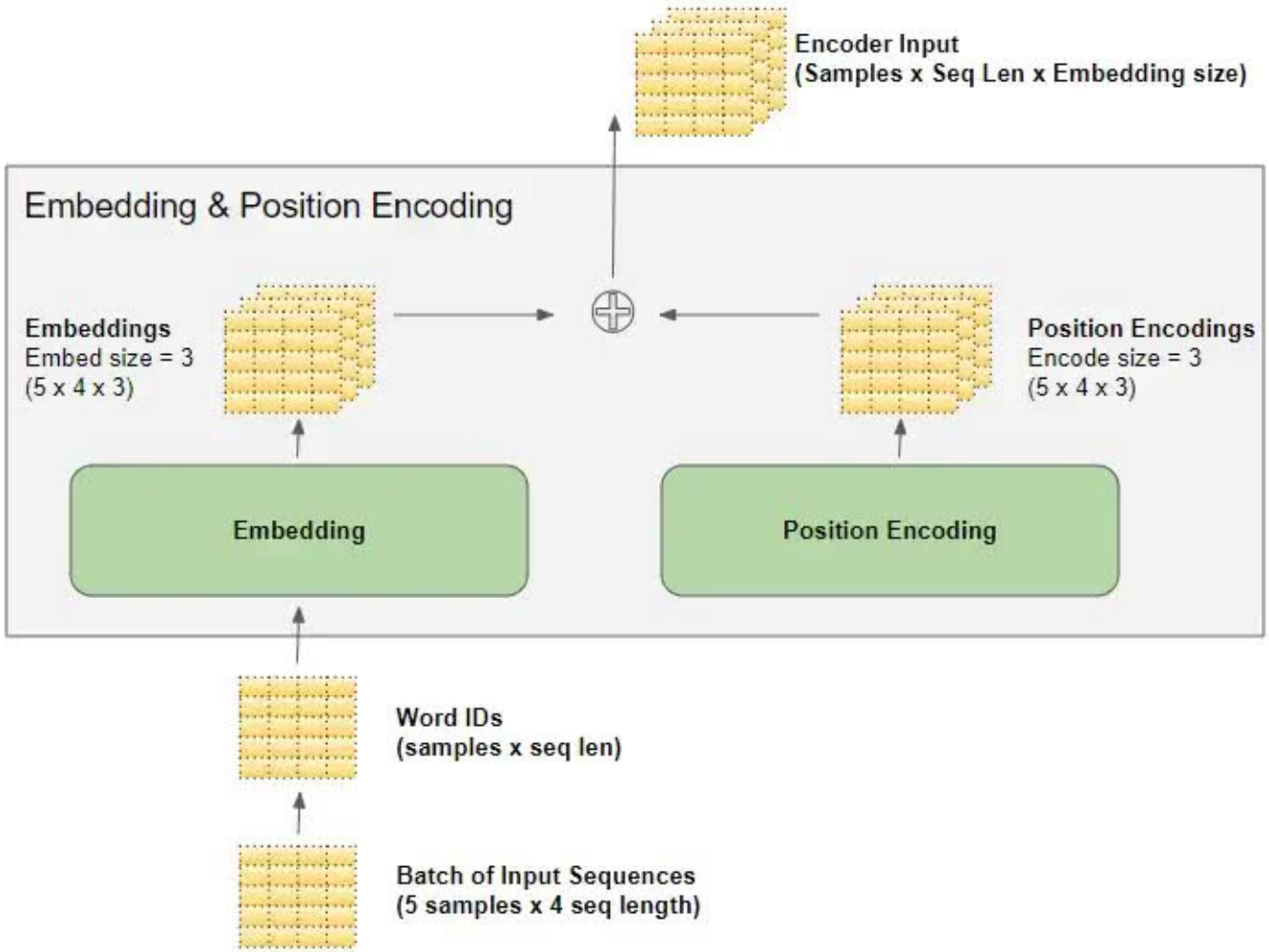
Two components for a length 40 input

Positional encoding

- Encoding of one position is linearly predictable from another, given their offset distance (phase shift)
- If the positional encoding is removed, the resulting model is permutation invariant

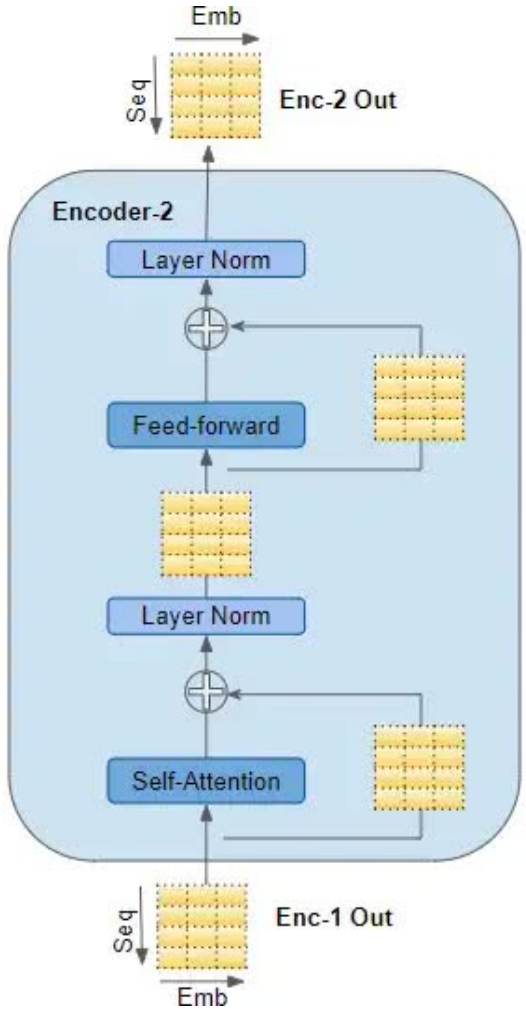
First stage

Everything is processed in batches; embedding stage looks like this:



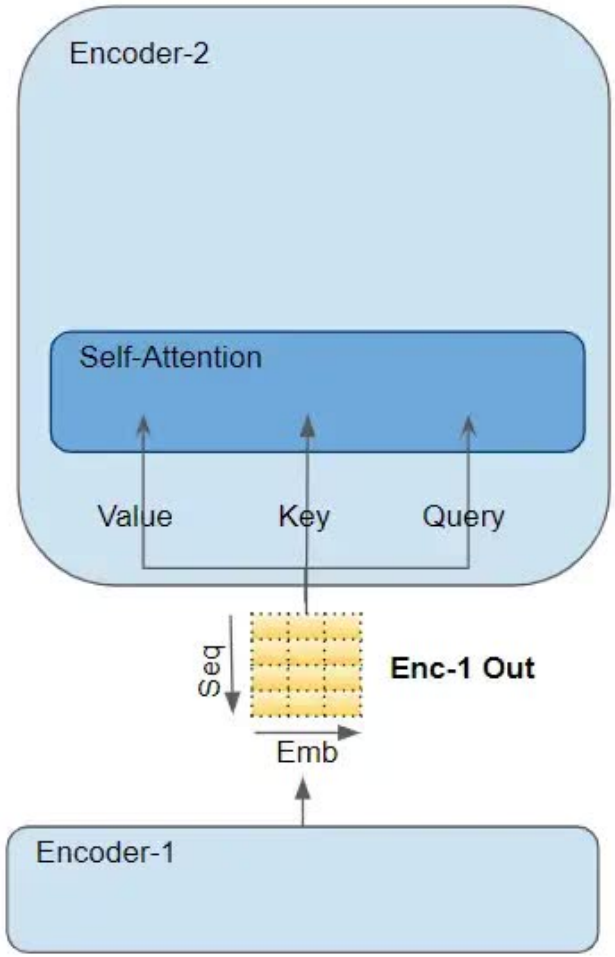
Multihead Attention

Next is an encoder layer



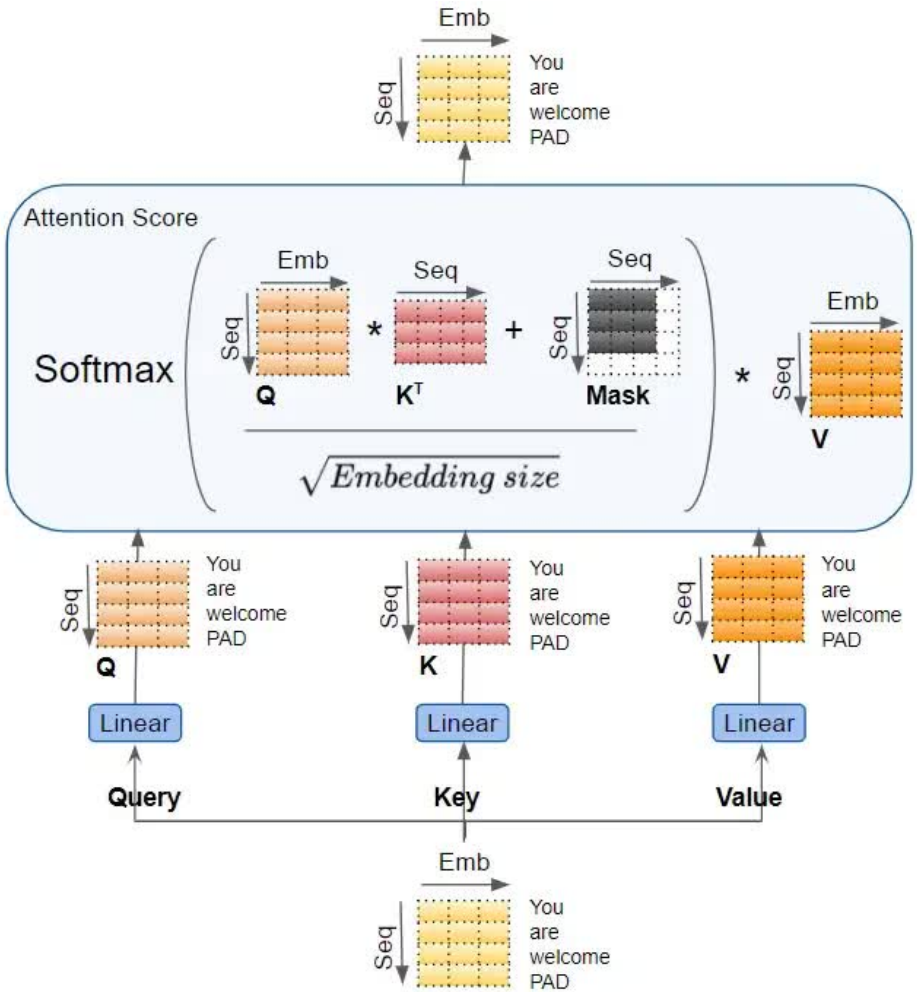
Multihead Attention

Detail of self-attention component

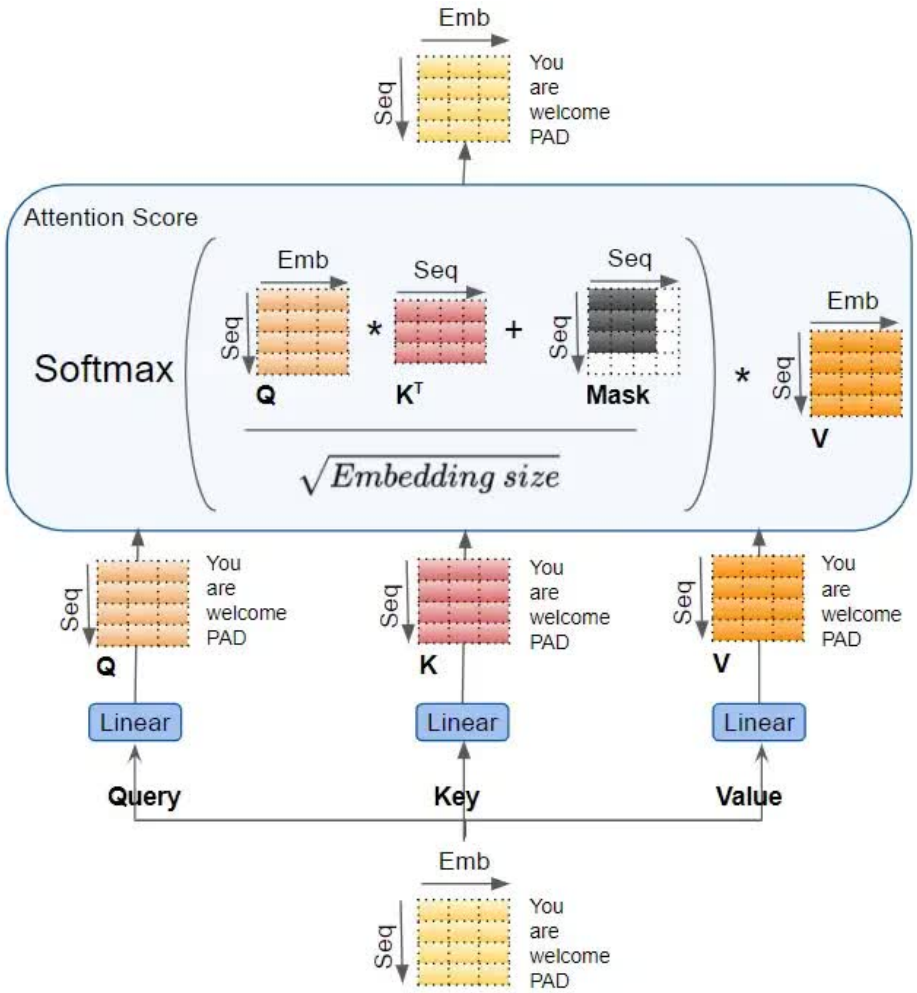


Multihead Attention

Detail of self-attention component (Blue is trainable parameters):

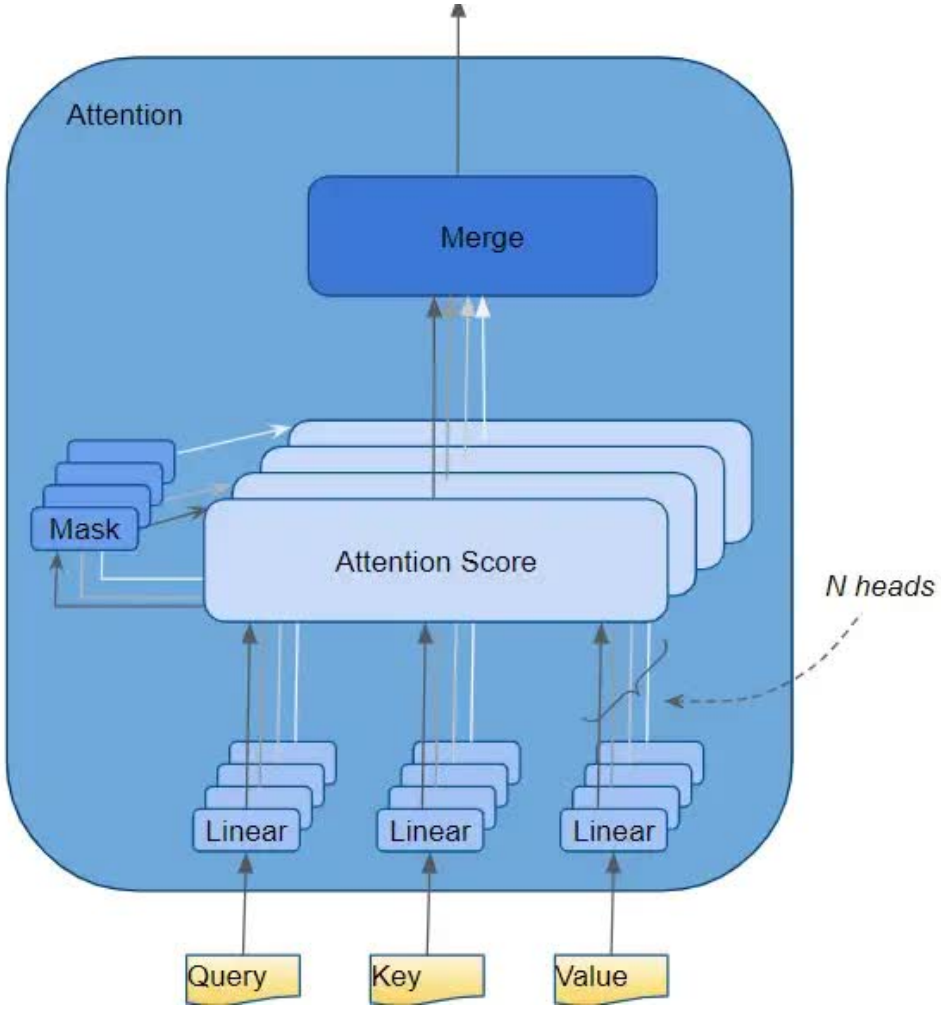


Multihead Attention



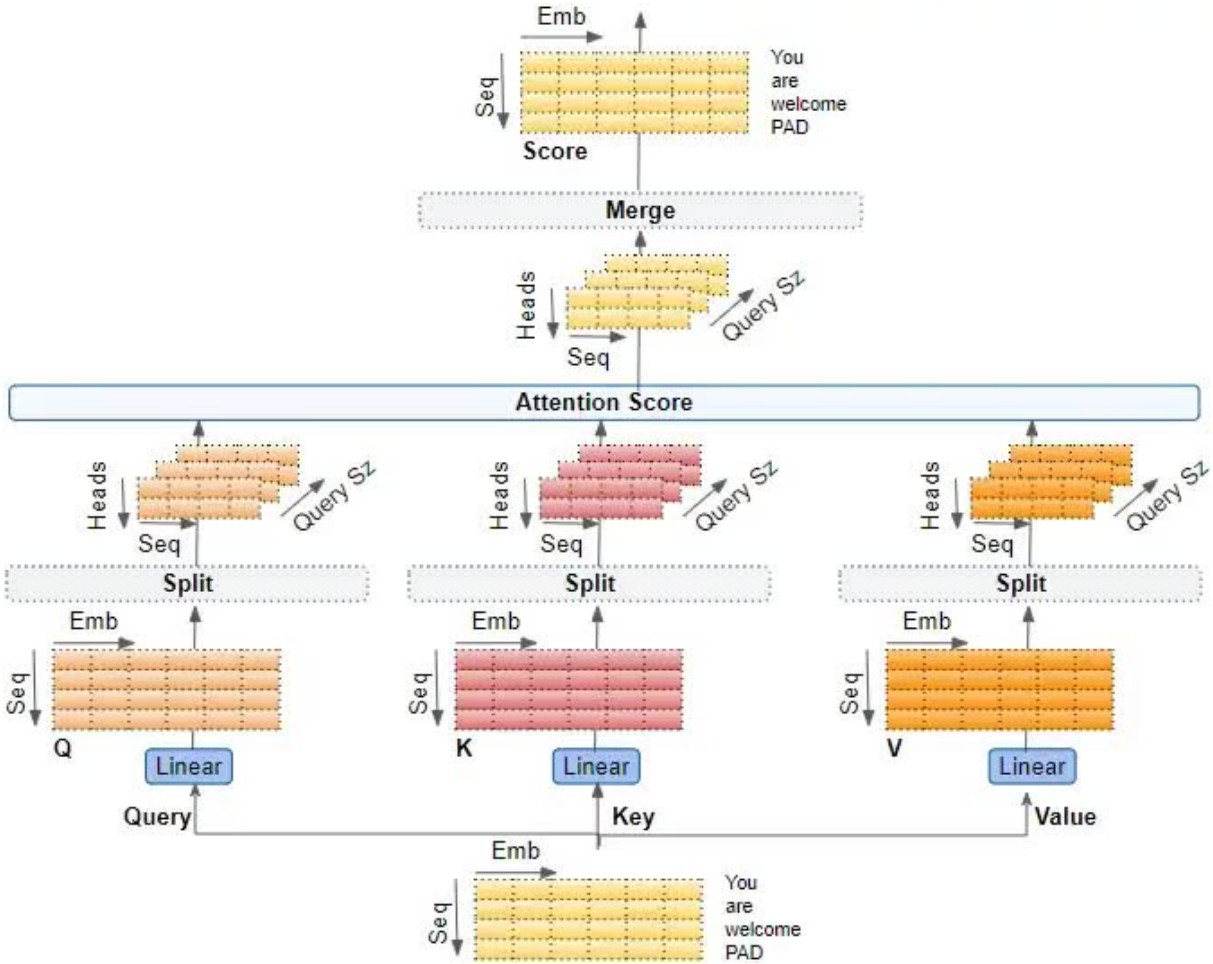
Mask is used in decoder to prevent use of future words

Multihead Attention



Multiple attention vectors are computed, then merged (concatenated)

Multihead Attention

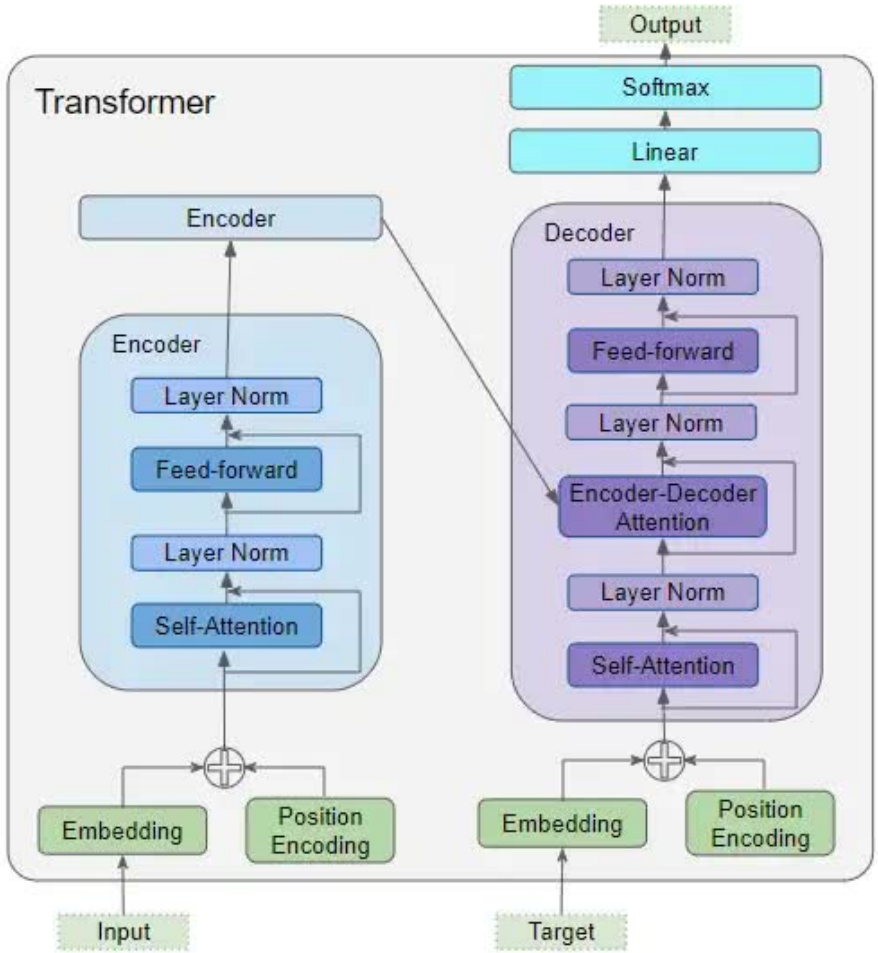


Multiple attention embeddings (alignments) are computed, then merged

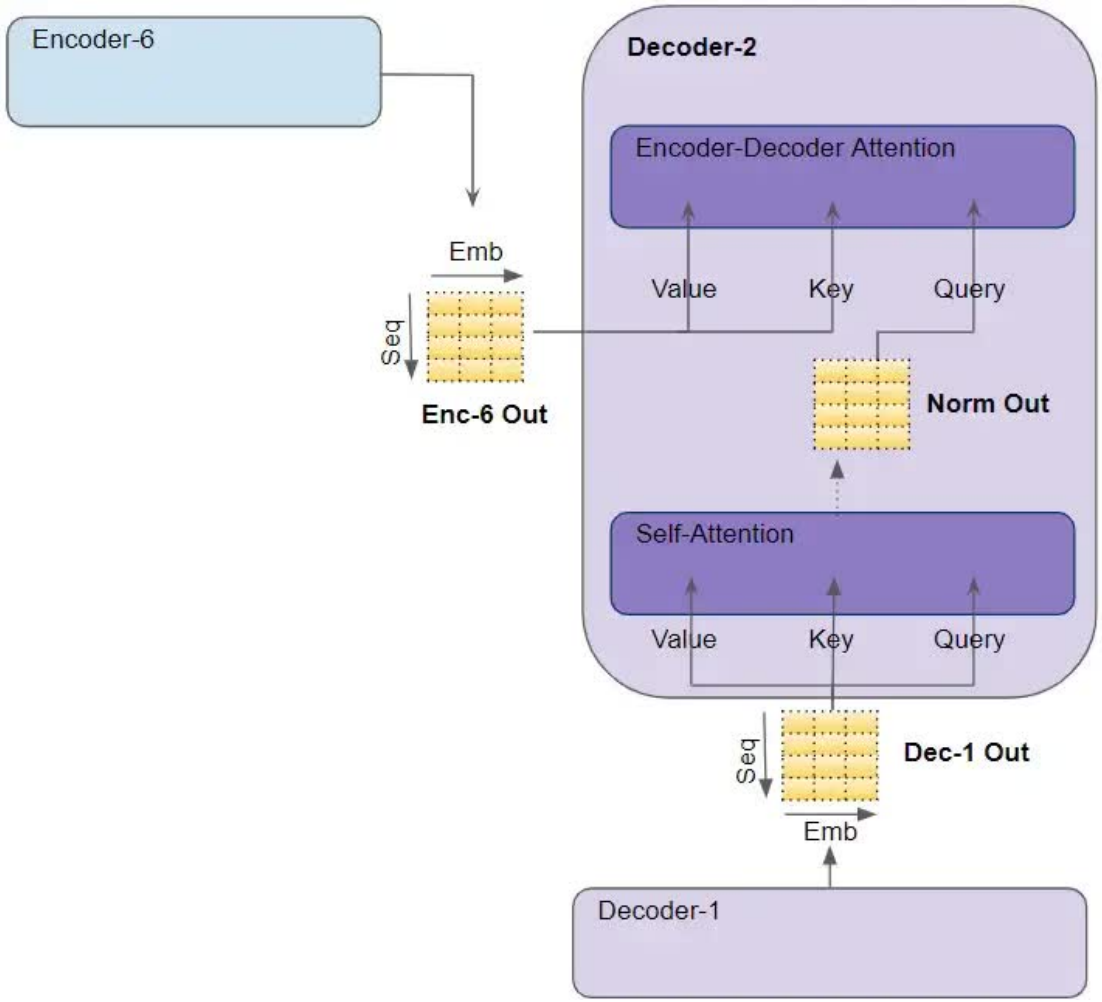
Back to intuition

- Different attention vectors are concatenated
- Capture different meanings of the sequence
- Words “hungry” and “sweet” are predicted using all of them

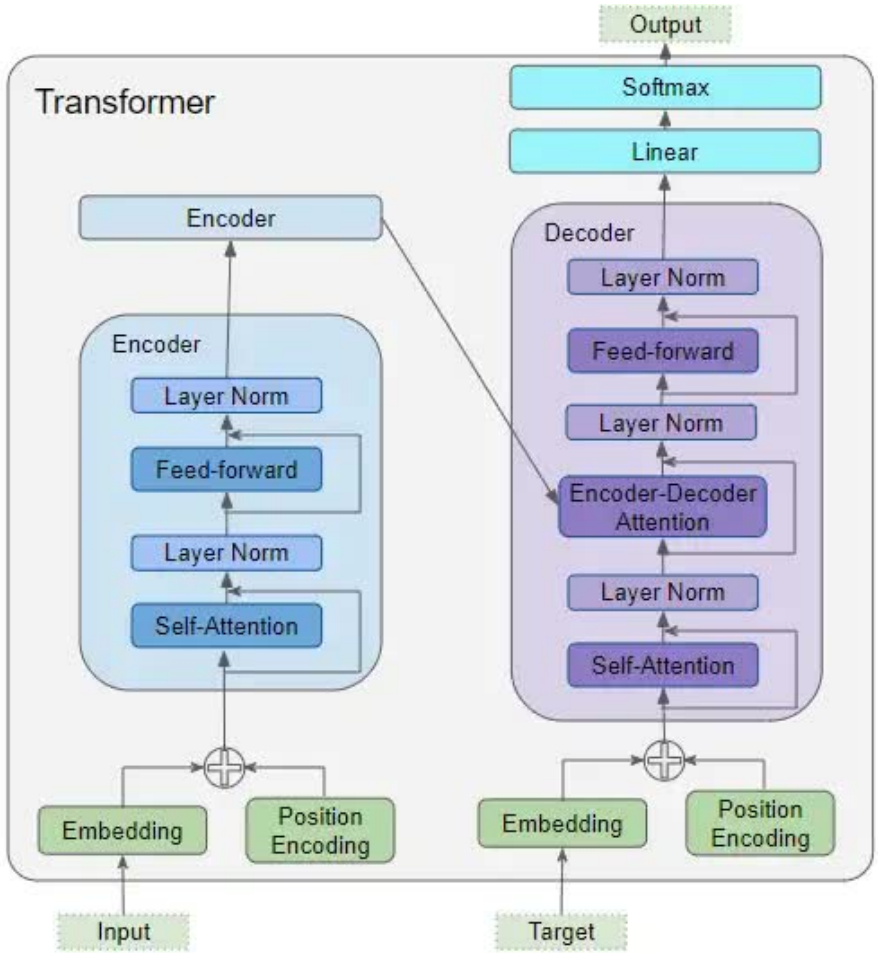
Encoder-Decoder attention



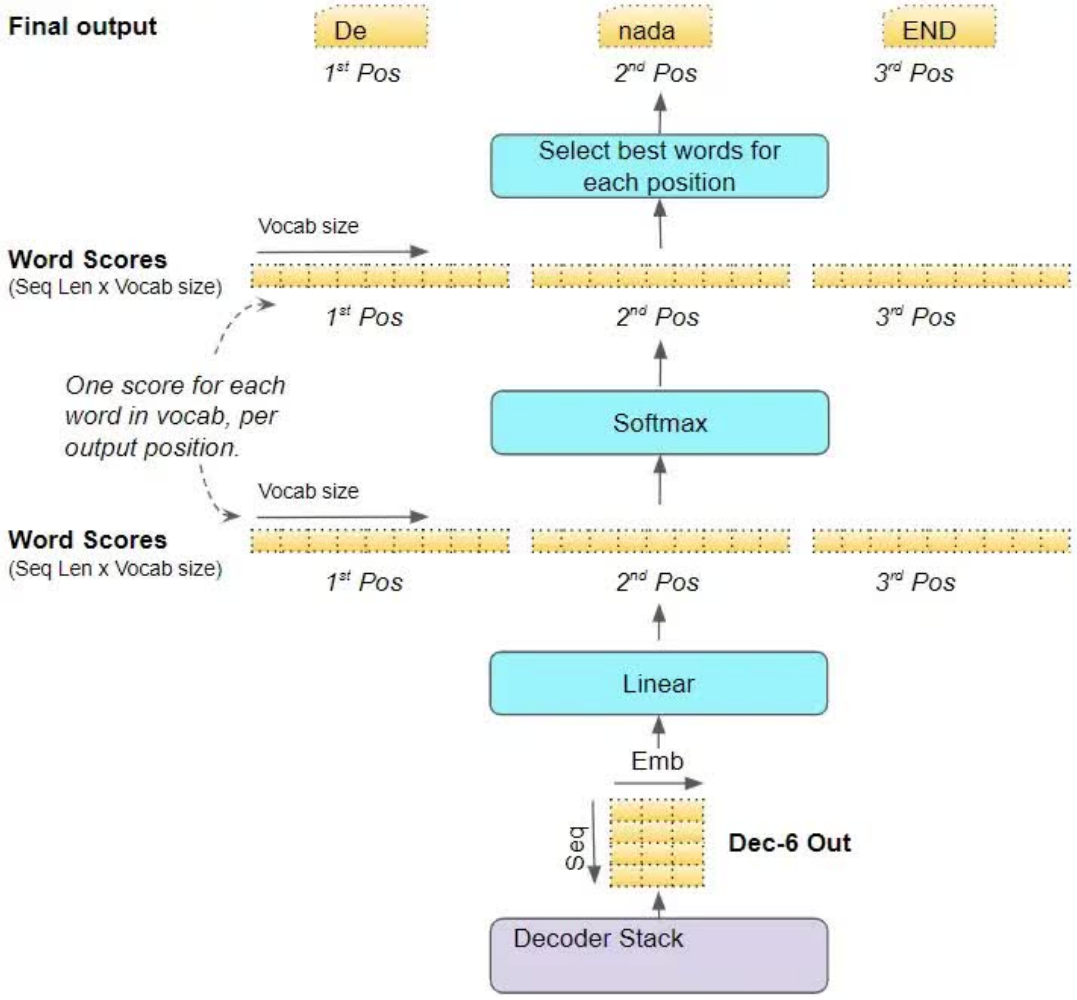
Encoder-Decoder attention



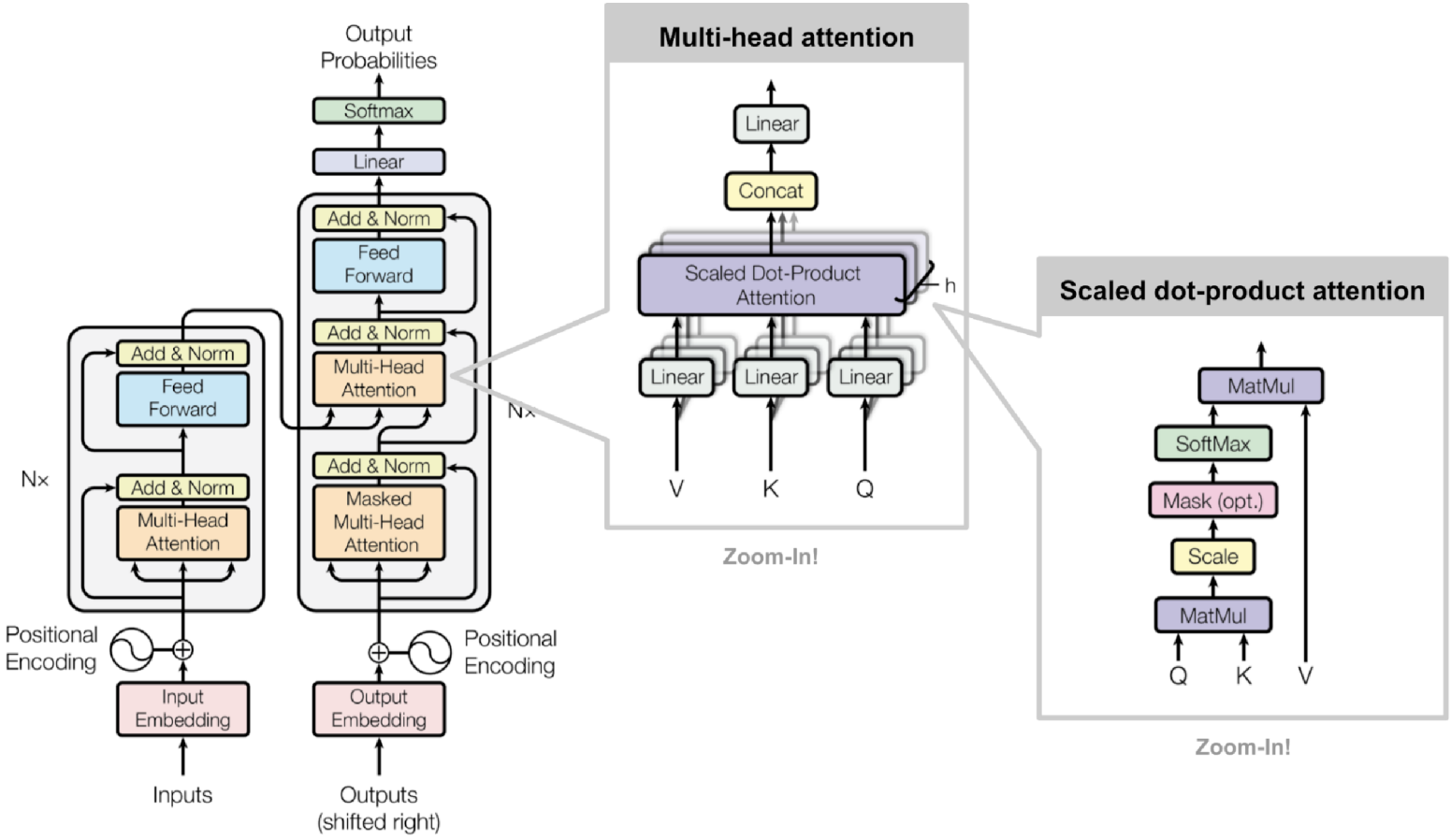
Final output generation



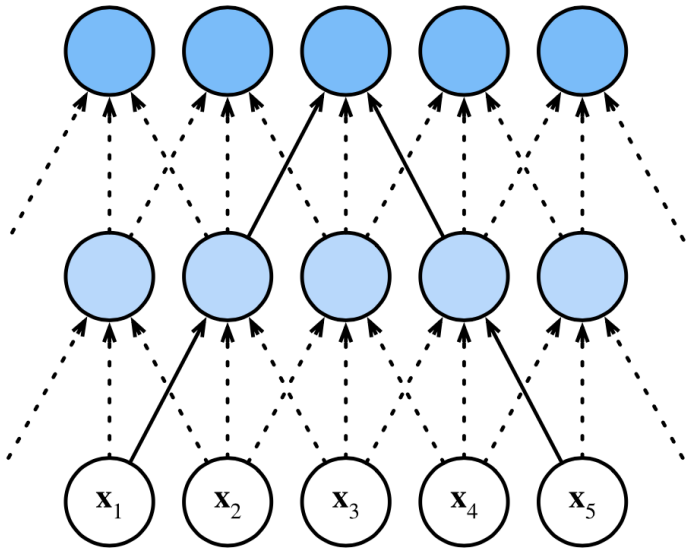
Final output generation

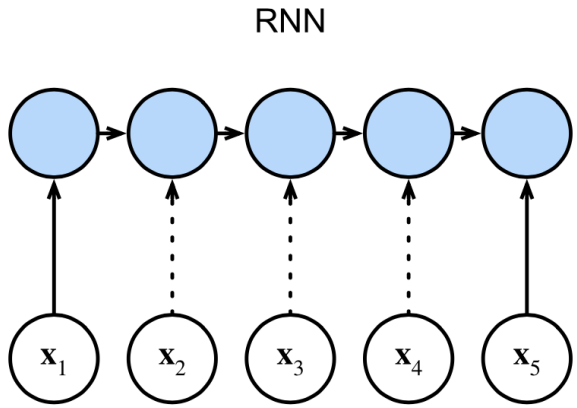


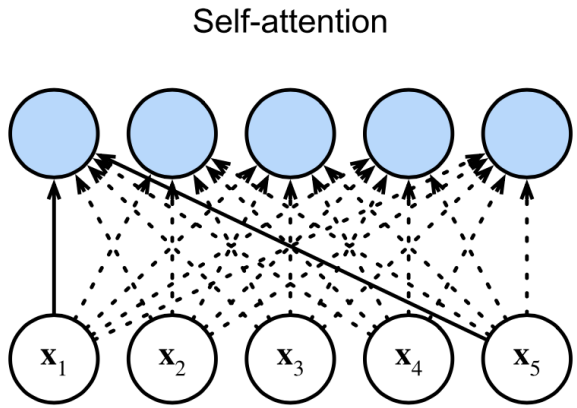
Overall architecture



Abstract comparison

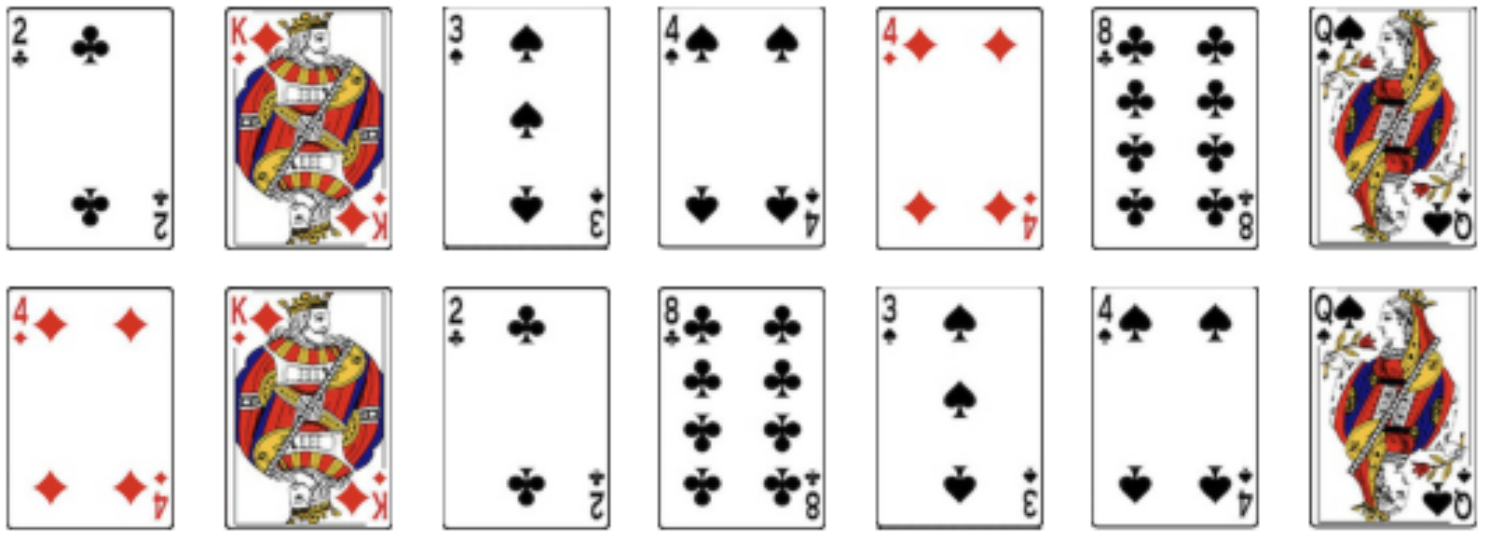






Demo

```
for i in range(5):
    j = np.random.choice(range(len(output)))
    deck.show_hand(source_test[j,1:(hand_size+1)])
    deck.show_hand(output[j,1:(hand_size+1)])
    print('\n')
```



Summary

Transformers:

- Process all words together, using layers of encoders and decoders, each with a multi-head attention component
- Replace sequential state structure with a series of transformations of the entire sequence. (“All you need is attention”)
- Combine multiple “alignments” deterministically — not as mixture
- Give powerful prediction engines for sequential data and sequence-to-sequence problems