# Self-attention and transformers

Herman Kamper



Issues with RNNs

Attention recap

Self-attention

Positional encodings

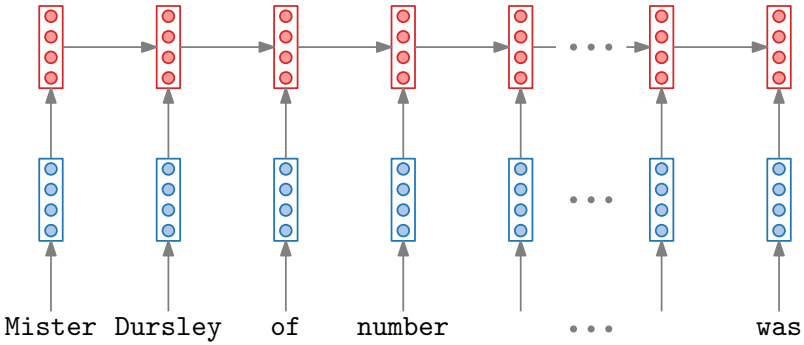
Multi-head attention

Masking the future in self-attention

Cross-attention

Transformer

# Issues with RNNs



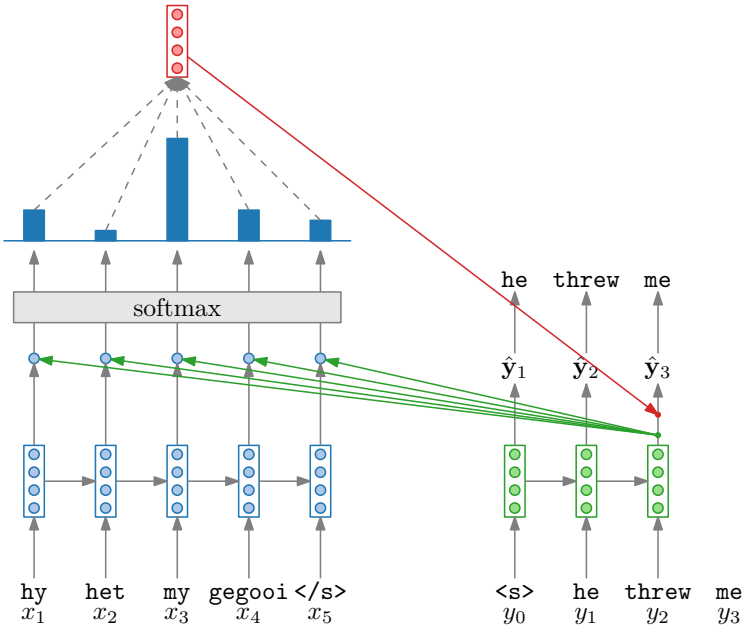

### Architectural

Even with changes to deal with long-range dependencies (e.g. LSTM), more recent observations inevitably have a bigger influence on the current hidden state than those that are far away.

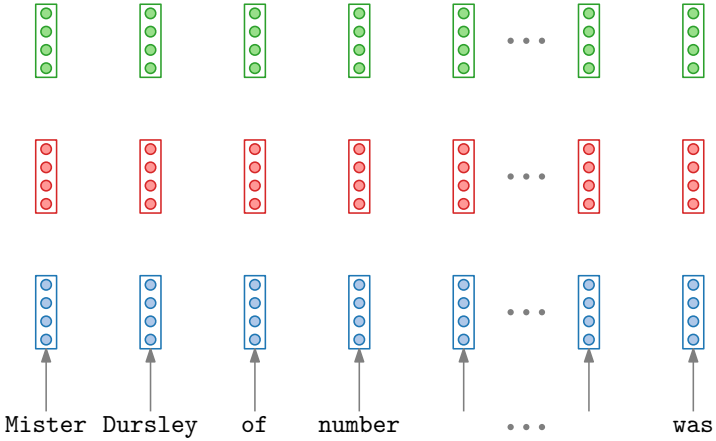
### Computational

- Future RNN states can't be computed before past hidden states have been computed.
- Computations over time steps are therefore not parallelizable.
- We just can't get away from the "for loop" over time in the forward pass over an RNN.
- So we can't take advantage of the full power of batching on GPUs, which wants several independent computations to be performed at once.

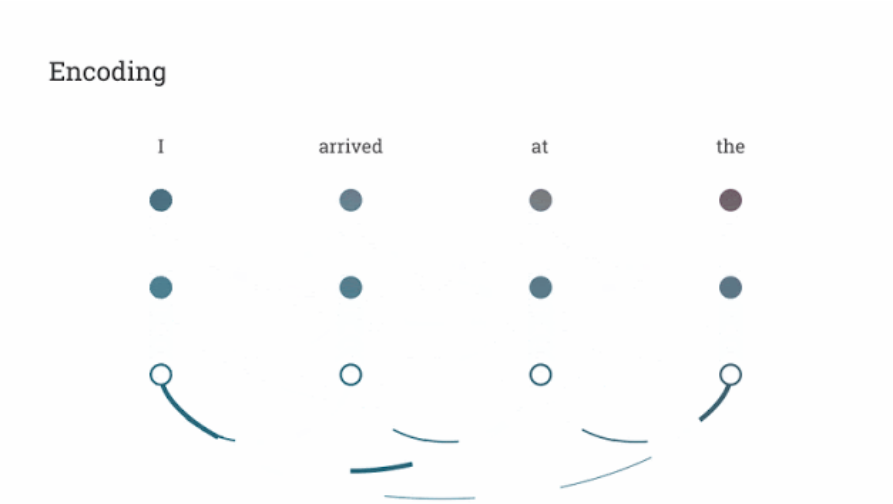
# Attention doesn't have these problems




**Idea:** Remove recurrence and rely solely on attention.

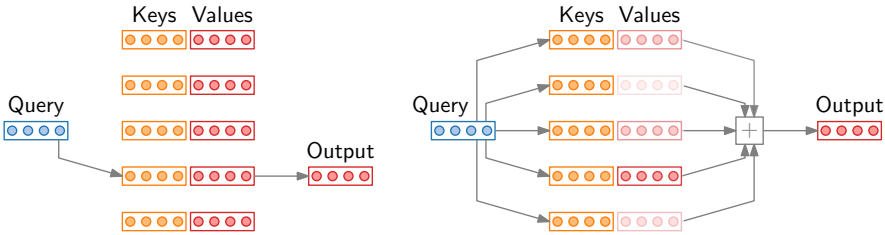


Intuition from the Google AI blog post:

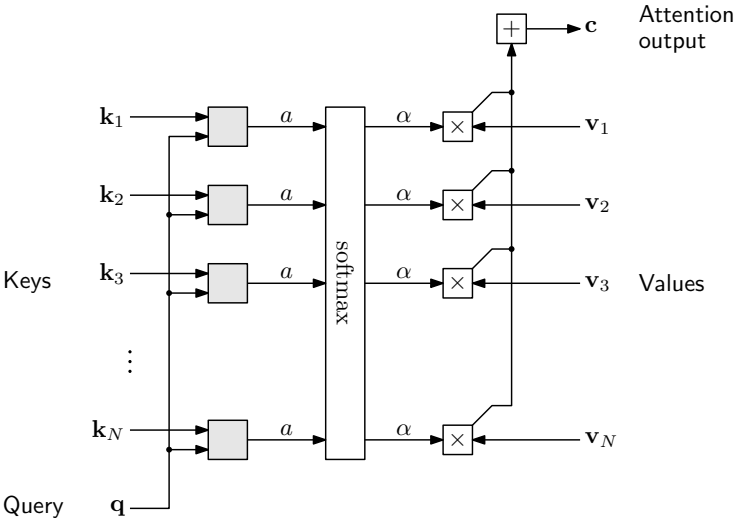


# Attention recap

One way to think of attention intuitively is as a soft lookup table:




Computational graph:



Mathematically:

- Output of attention: Context vector.

$$\mathbf{c} = \sum_{n=1}^{N} \alpha(\mathbf{q}, \mathbf{k}_n)\mathbf{v}_n \in \mathbb{R}^D$$

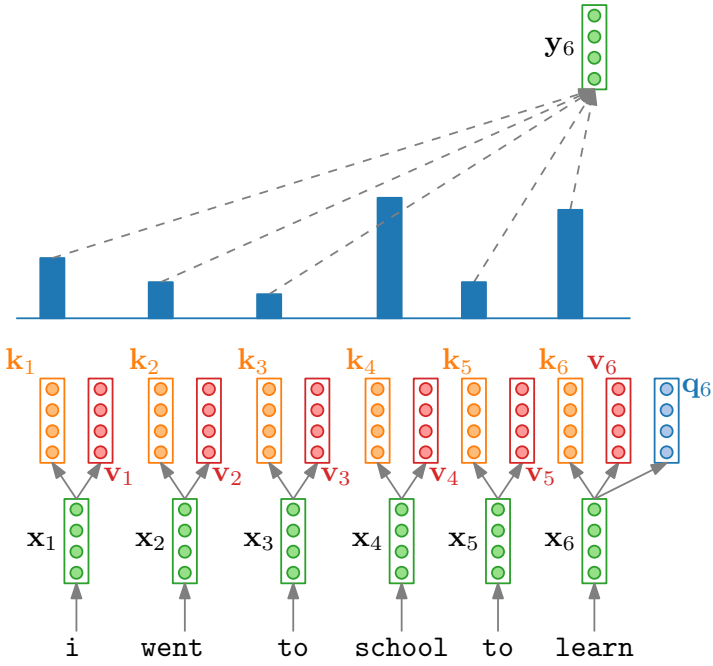
- Attention weight:

$$\begin{aligned}\alpha(\mathbf{q}, \mathbf{k}_n) &= \mathrm{softmax}_n\left(a(\mathbf{q}, \mathbf{k}_n)\right) \\ &= \frac{\exp\left\{a(\mathbf{q}, \mathbf{k}_n)\right\}}{\sum_{j=1}^{N} \exp\left\{a(\mathbf{q}, \mathbf{k}_j)\right\}} \in [0, 1]\end{aligned}$$

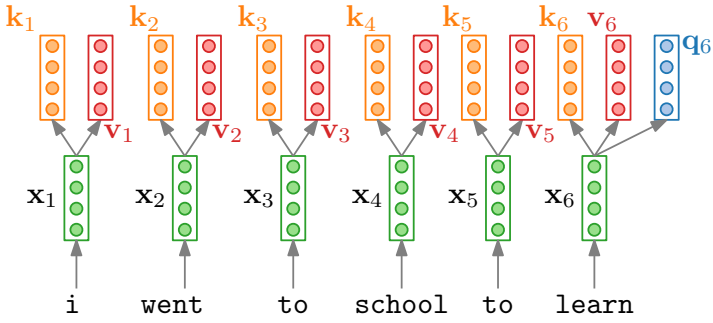
- Attention score:

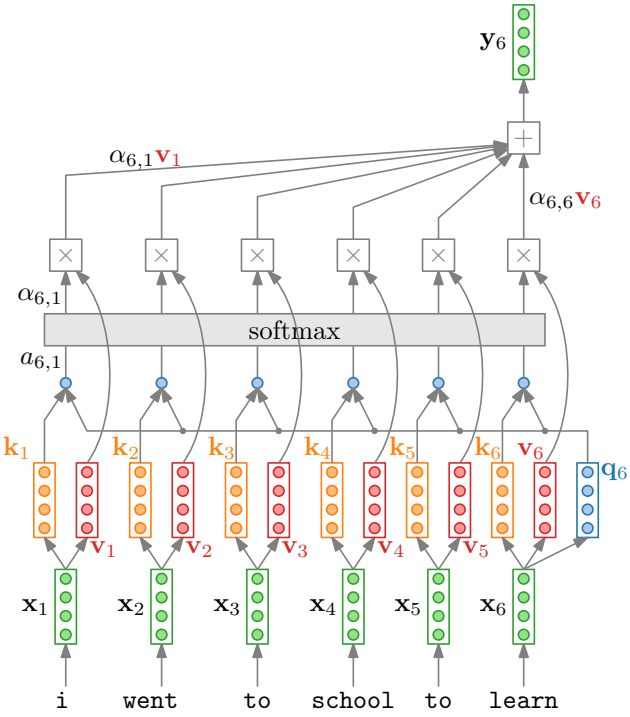
$$a(\mathbf{q}, \mathbf{k}_n) \in \mathbb{R}$$

# Self-attention



# Self-attention






$$\mathbf{y}_i = \sum_{t=1}^{T} \alpha_{i,t}\mathbf{v}_t$$

$$\alpha_{i,t} = \frac{e^{a_{i,t}}}{\sum_{j=1}^{T} e^{a_{i,j}}}$$

$$a_{i,t} = \frac{\mathbf{q}_i^\top \mathbf{k}_t}{\sqrt{D_k}}$$

$$\mathbf{q}_t = \mathbf{W}_q^\top \mathbf{x}_t$$
$$\mathbf{k}_t = \mathbf{W}_k^\top \mathbf{x}_t$$
$$\mathbf{v}_t = \mathbf{W}_v^\top \mathbf{x}_t$$

Layer input: $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_T$

Layer output: $\mathbf{y}_1, \mathbf{y}_2, \ldots \mathbf{y}_T$

## In matrix form

Each of the $T$ queries need to be compared to each of the $T$ keys. We can express this in a compact matrix form.

Let's stack all the queries, keys and values as rows in matrices:

$$\mathbf{Q} \in \mathbb{R}^{T \times D_k}$$
$$\mathbf{K} \in \mathbb{R}^{T \times D_k}$$
$$\mathbf{V} \in \mathbb{R}^{T \times D_v}$$

We can then write all the dot products and weighting in a short condensed form:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{D_k}}\right)\mathbf{V}$$

If we denote the output as $\mathbf{Y} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, then we end up with a result $\mathbf{Y} \in \mathbb{R}^{T \times D_v}$.

The above holds in general for attention. For self-attention specifically, we would have

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q$$
$$\mathbf{K} = \mathbf{X}\mathbf{W}_k$$
$$\mathbf{V} = \mathbf{X}\mathbf{W}_v$$

where the design matrix is $\mathbf{X} \in \mathbb{R}^{T \times D}$, with $D$ the dimensionality of the input.

You can figure out the shapes for the $\mathbf{W}$'s, e.g. $\mathbf{W}_k \in \mathbb{R}^{D \times D_k}$.

# Self-attention: A new computational block

A new block or layer, like an RNN or a CNN.

Can use this in both encoder and decoder modules, e.g. for machine translation:

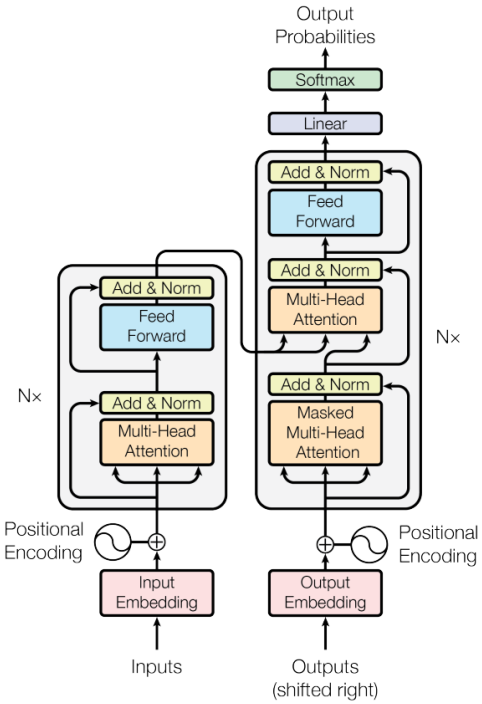



Figure from (Vaswani et al., 2017).

Sometimes "transformer" is used to refer to the self-attention layers themselves, but other times it is used to refer to this specific encoder-decoder model (which we will unpack in the rest of this note).

# Positional encodings

In contrast to RNNs, there isn't any order information in the inputs of self-attention.

We can add positional encodings to the inputs:

$$\mathbf{p}_t \in \mathbb{R}^D$$

There is a unique $\mathbf{p}_t$ for every input position. E.g. $\mathbf{p}_{10}$ will always be the same for all input sequences.

How do we incorporate them? The positional encodings can be concatenated to the inputs:

$$\tilde{\mathbf{x}}_t = \Big[\mathbf{x}_t; \mathbf{p}_t\Big]$$

But it is more common to just add them:[1]

$$\tilde{\mathbf{x}}_t = \mathbf{x}_t + \mathbf{p}_t$$

Where do the positional encodings $\mathbf{p}_t$ come from?

### Learned positional encodings

We can let the $\mathbf{p}_t$'s be learnable parameters. This means we are adding a learnable matrix $\mathbf{P} \in \mathbb{R}^{D \times T}$ to all input sequences.

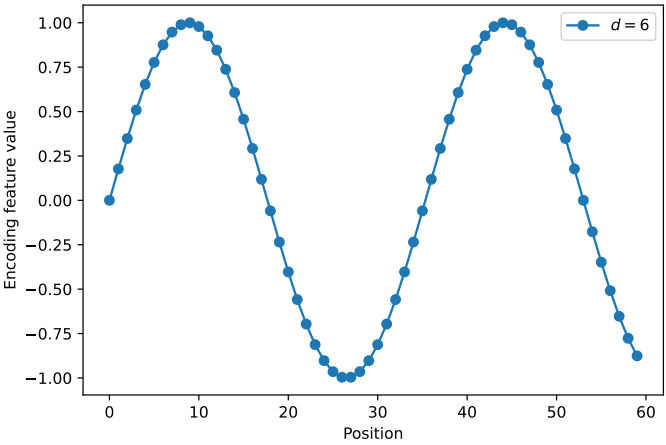
Problem: What if we have inputs that have longer lengths than $T$?

(But this is still often used in practice.)

[1] I like the idea of concatenation more than adding. But Benjamin van Niekerk pointed out to me that if you pass $\tilde{\mathbf{x}}_t$ through a single linear layer, then concatenation and addition are very similar: In both cases you end up with a new representation that is a weighted sum of the original input and the positional encoding (there are just additional weights specifically for the positional encoding when you concatenate).

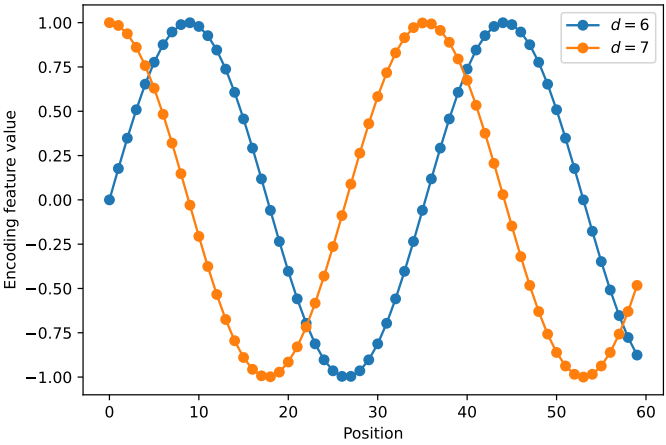
## Represent position using sinusoids

Let's use a single sinusoid as our $p_t$:



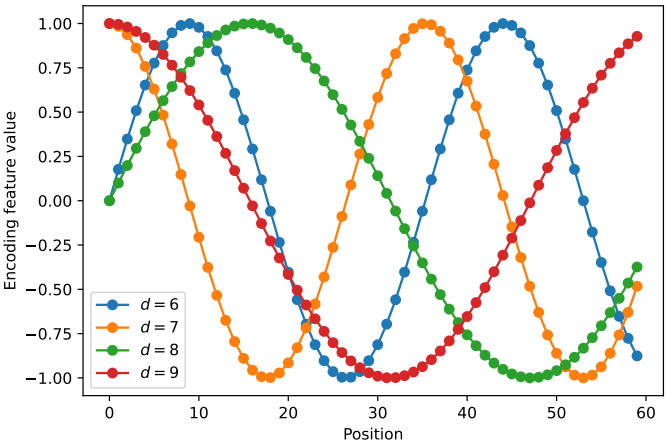

In this case, we would have unique positional feature value for inputs roughly with lengths up to $T = 36$, and then the feature value would repeat. This could be useful, if relative position at this scale is more important than absolute position.

Let's add a cosine to obtain $\mathbf{p}_t$:



Now we would have unique positional encodings for a longer range. But the model could also just decide that relative position matters more.

We used sinusoids at a single frequency, so you are limited in the types of relative relationships you can model. So let us add more sine and cosine functions at different frequencies:



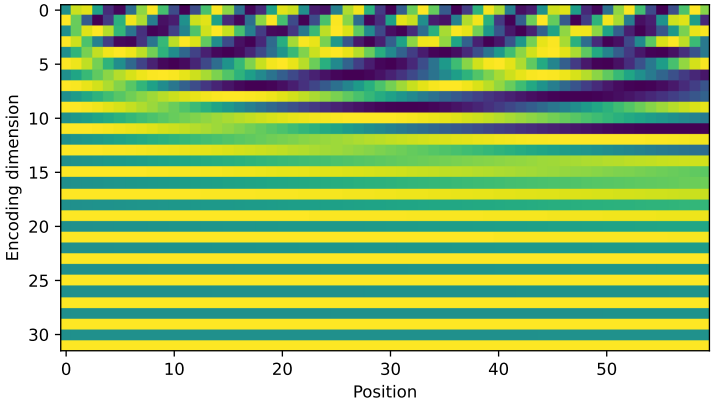

Formally (Vaswani et al., 2017):

$$\mathbf{p}_t = \begin{bmatrix} \sin\left(\frac{t}{f_1}\right) \\ \cos\left(\frac{t}{f_1}\right) \\ \sin\left(\frac{t}{f_2}\right) \\ \cos\left(\frac{t}{f_2}\right) \\ \vdots \\ \sin\left(\frac{t}{f_{D/2}}\right) \\ \cos\left(\frac{t}{f_{D/2}}\right) \end{bmatrix}$$

where

$$f_m = 10\,000^{2m/D}$$

If we stack all these into $\mathbf{P} \in \mathbb{R}^{D \times T}$:



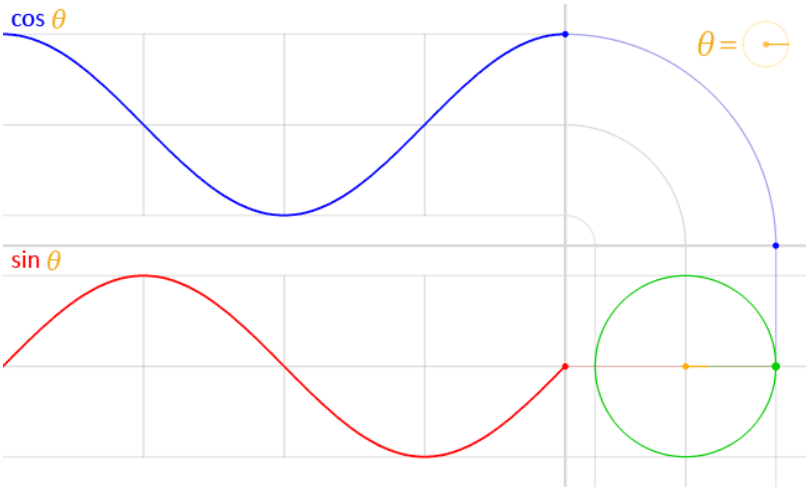

There are formal reasons that this encodes relative position (Denk, 2019).[2] But intuitively you should be able to see that periodicity indicates that absolute position isn't necessarily important.

In practice, however, this approach does not enable extrapolation to sequences that are way longer than those seen during training (Hewitt, 2023).

(But it is still often used in practice. The original transformer paper did this—look at the transformer diagram above.)

[2] For a fixed offset between two positional encodings, there is a linear transformation to take you from the one to the other. E.g. you can go from $\mathbf{p}_{10}$ to $\mathbf{p}_{15}$ using some linear transformation, and this will be the same transformation needed to go from $\mathbf{p}_{30}$ to $\mathbf{p}_{35}$.

## The clock analogy for positional encodings




Think of each pair of dimensions of $\mathbf{p}_t$ as a clock rotating at a different frequency. The position of the clock is uniquely determined by the sine and cosine functions for that frequency.[3]

We have $D/2$ clocks. For each position $t$, we will have a specific configuration of clocks. This tells us where in the input sequence we are. This works, even if we never saw a long input sequence length during training (the clocks just move on).

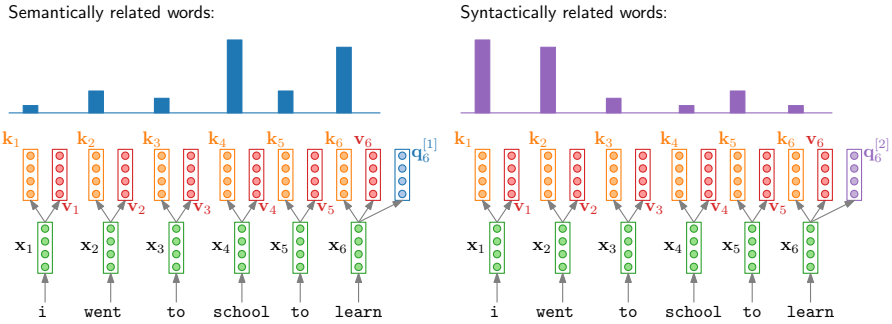
But there is also periodicity in how clocks change with different $t$. To move from the configuration $\mathbf{p}_{10}$ to $\mathbf{p}_{15}$, we need to change the clock faces in some way (this can be done through a linear transformation). But this way in which we change the clock faces, would be the same as the transformation from $\mathbf{p}_{30}$ to $\mathbf{p}_{35}$.

So in short, the sinusoidal positional encodings can tell us where we are in the input, even if that position was never seen during training. But it also allows for relative position information to be captured.

[3] Analogy from Benjamin van Niekerk.

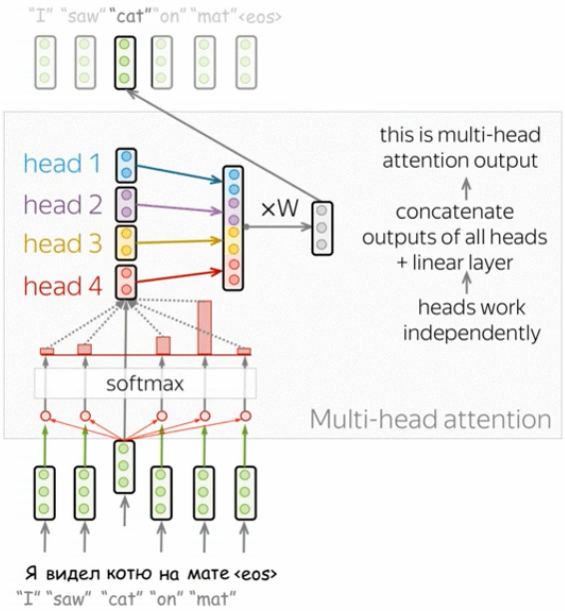
# Multi-head attention

Hypothetical example:




Analogy: Each head is similar to a different kernel in a CNN.

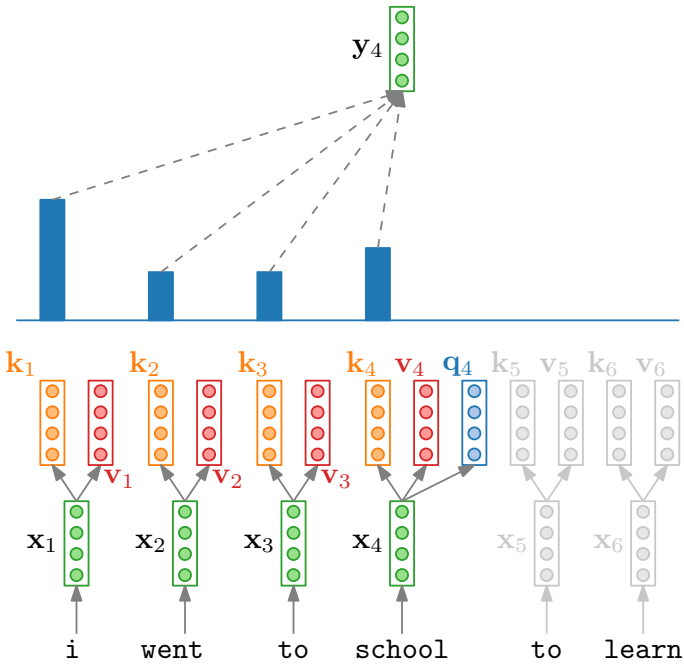
From Lena Voita's blog:



# Masking the future in self-attention

If we have a network or decoder that needs to be causal, then we should ensure that it can only attend to the past when making the current prediction.

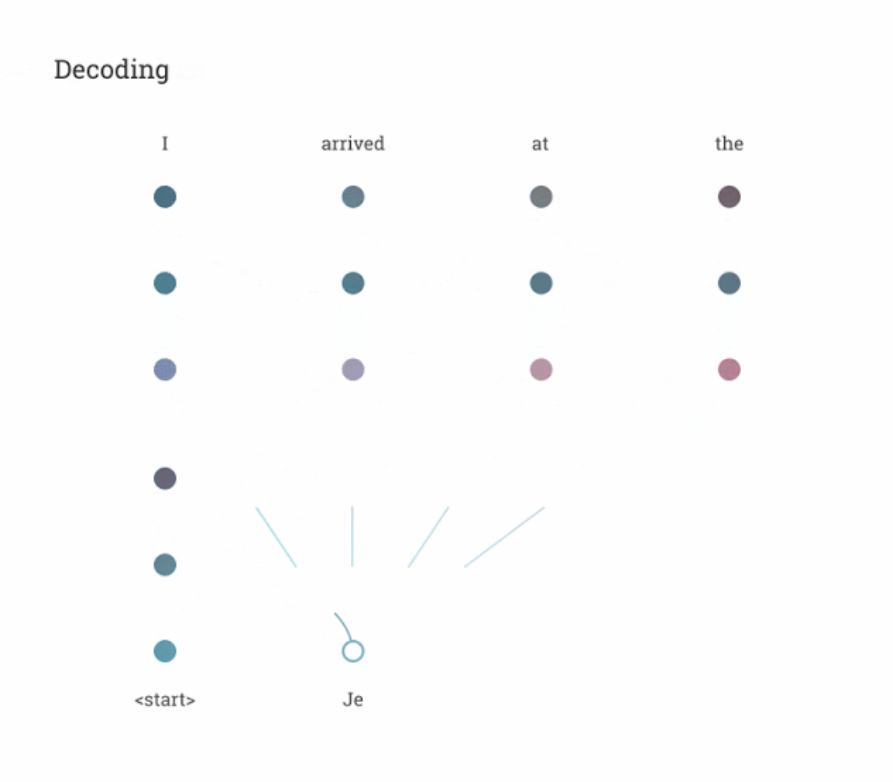
E.g. if we are doing language modelling:



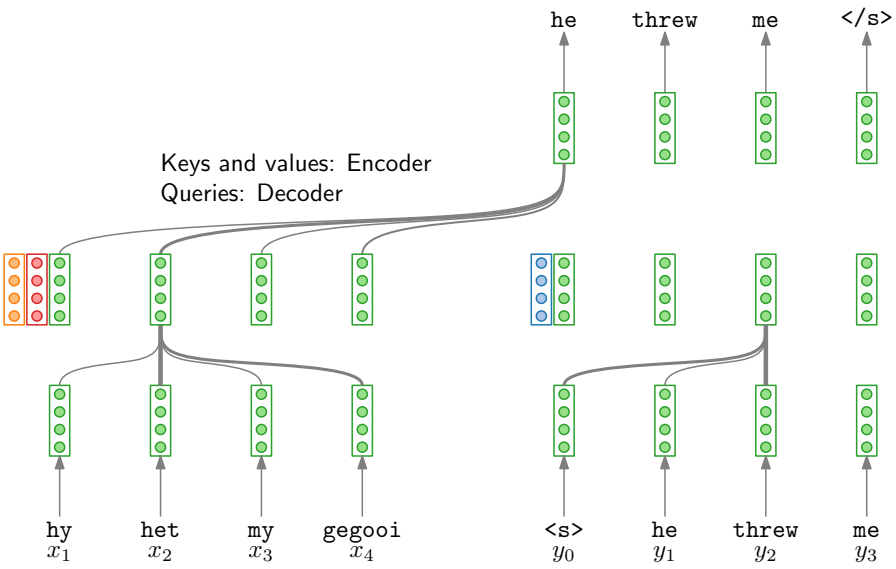

Mathematically:

$$a_{i,t} = \begin{cases} \frac{\mathbf{q}_i{}^\top \mathbf{k}_t}{\sqrt{D_k}} & \text{if } t \leq i \\ -\infty & \text{if } t > i \end{cases}$$

Have a careful look at what happens in the Google transformer diagram for machine translation:



# Cross-attention




Have a look at the Google transformer diagram again.

# Transformer

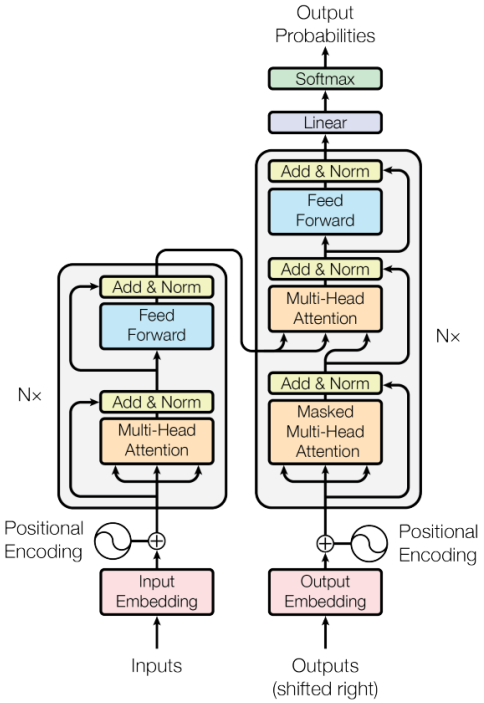



Figure from (Vaswani et al., 2017).

We haven't spoken about the add & norm block:

- Residual connections
- Layer normalization

## Acknowledgments

Christiaan Jacobs and Benjamin van Niekerk were instrumental in helping me to start to understand self-attention and transformers.

This note relied heavily on content from:

- The CS224N course of Chris Manning at Stanford University, particularly the transformer lecture taught by John Hewitt.
- The NLP Course for You blog by Lena Voita.

## Further reading

A. Goldie, "CS224N: Pretraining," *Stanford University*, 2022.

A. Huang, S. Subramanian, J. Sum, K. Almubarak, and S. Biderman, "The annotated transformer," *Harvard University*, 2022.

## References

T. Denk, "Linear relationships in the transformer's positional encoding," 2019.

J. Hewitt, "CS224N: Self-attention and transformers," *Stanford University*, 2023.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, 2017.

L. Voita, "Sequence to aequence (seq2seq) and attention," 2023.

A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, 2021.