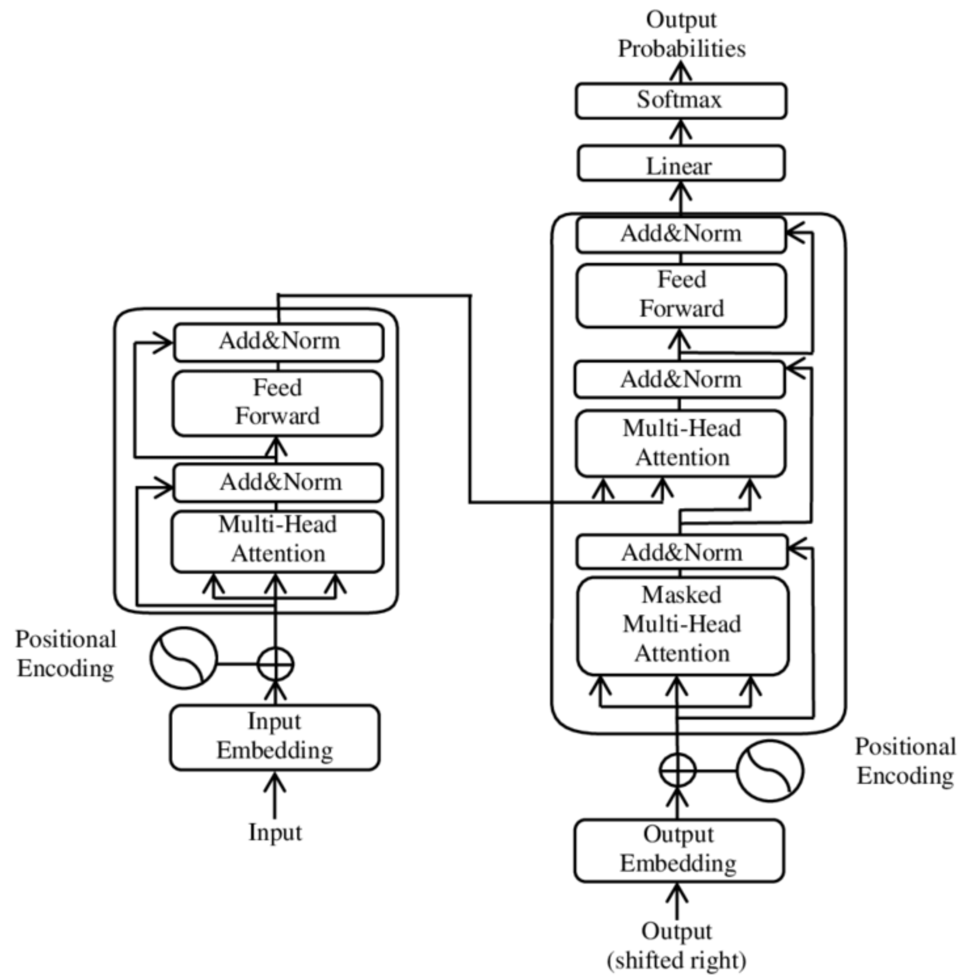
Lecture 29: Transformers

Mark Hasegawa-Johnson

4/2024



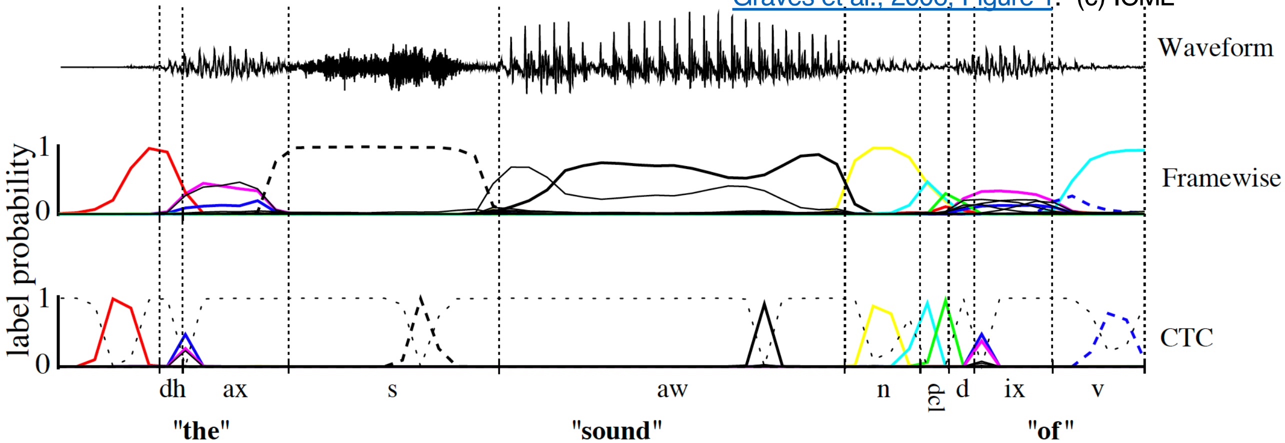
By Yuening Jia - DOI:10.1088/1742-6596/1314/1/012186, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=121340680

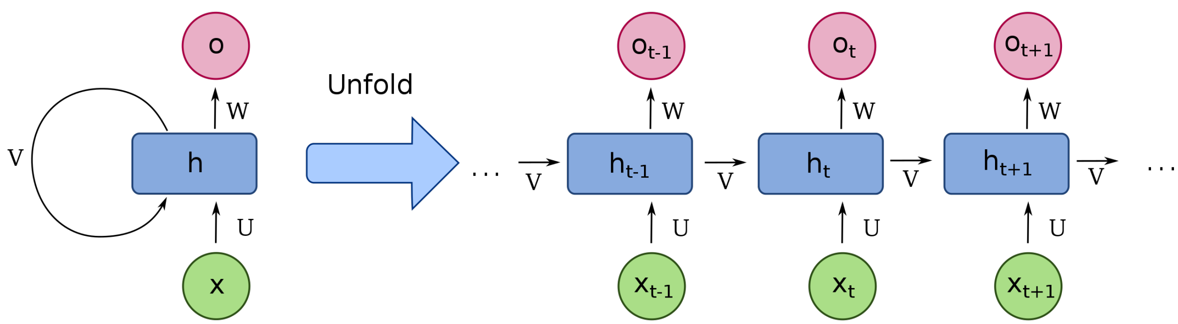
Outline

- Recurrent neural networks
- Attention
- Self-attention, Multi-headed attention, Cross-attention, and Masked attention
- Positional embedding

Recurrent neural network

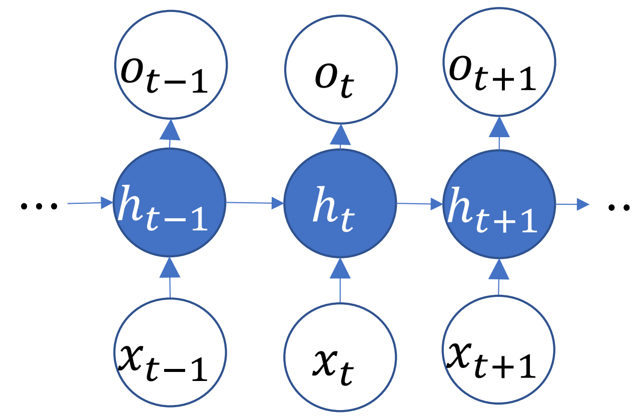
Graves et al., 2006, Figure 1. (c) ICML





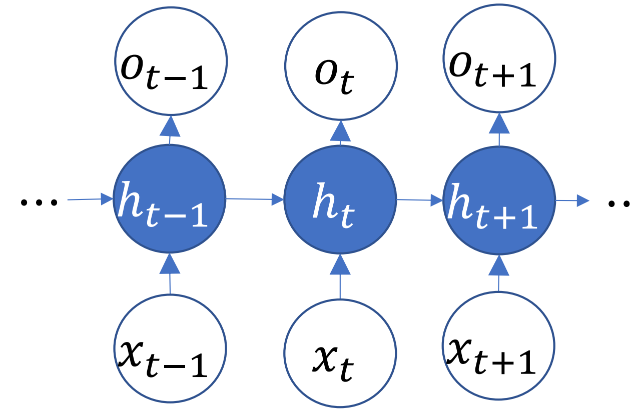
- In a recurrent neural network (RNN), the hidden node activation vector, h_t, depends on the value of the same vector at time $t-1$.
- From 2014-2017, the best speech recognition and machine translation used RNNs.
- The input is x_t=speech or input-language text
- The output is o_t=text in the target language

Example: Part of speech tagging



- $\boldsymbol{x}_t$ =vector representation of the tth word, e.g., trained using CBOW
- $\boldsymbol{h}_t$ =hidden state vector = $\tanh(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{V}\boldsymbol{h}_{t-1})$
- $\boldsymbol{o}_t = \text{softmax}(\boldsymbol{W}\boldsymbol{h}_t) = [P(Y_t = \text{Noun}|X_1, \dots, X_t), P(Y_t = \text{Verb}|X_1, \dots, X_t), \dots]$

Training an RNN



An RNN is trained using gradient descent, just like any other neural network!

$$u_{j,i} \leftarrow u_{j,i} - \eta \frac{\partial \mathfrak{L}}{\partial u_{j,i}}$$

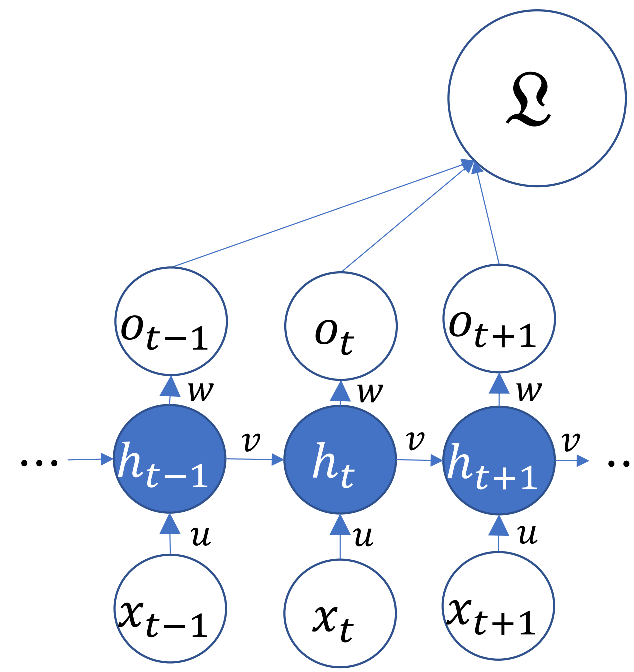
$$w_{j,k} \leftarrow w_{j,k} - \eta \frac{\partial \mathfrak{L}}{\partial w_{j,k}}$$

…where $\mathfrak{L}$ is the loss function, and η is a step size.

Training an RNN: Infinite recursion?

The big difference is that now the loss function depends on $\boldsymbol{U}, V$ and $\boldsymbol{W}$ in many different ways:

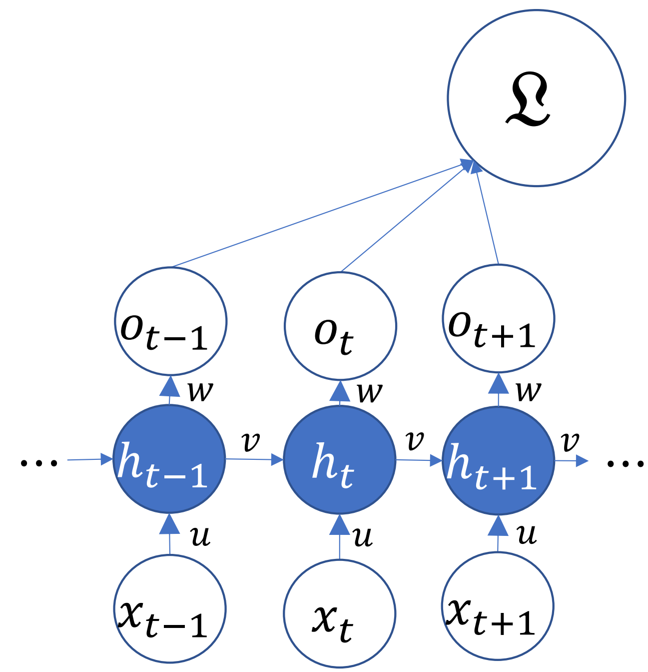
- The loss function depends on each of the state vectors $\boldsymbol{h}_t$, which depends directly on $\boldsymbol{U}$ and $\boldsymbol{V}$.
- But $\boldsymbol{h}_t$ also depends on $\boldsymbol{h}_{t-1}$, which, in turn, depends on $\boldsymbol{U}$ and $\boldsymbol{V}$.
- … and so on.



Back-propagation through time

The solution is something called back-propagation through time:

$$\frac{d\mathfrak{L}}{dh_{i,t}} = \frac{\partial\mathfrak{L}}{\partial h_{i,t}} + \sum_j \frac{d\mathfrak{L}}{dh_{j,t+1}} \frac{\partial h_{j,t+1}}{\partial h_{i,t}}$$



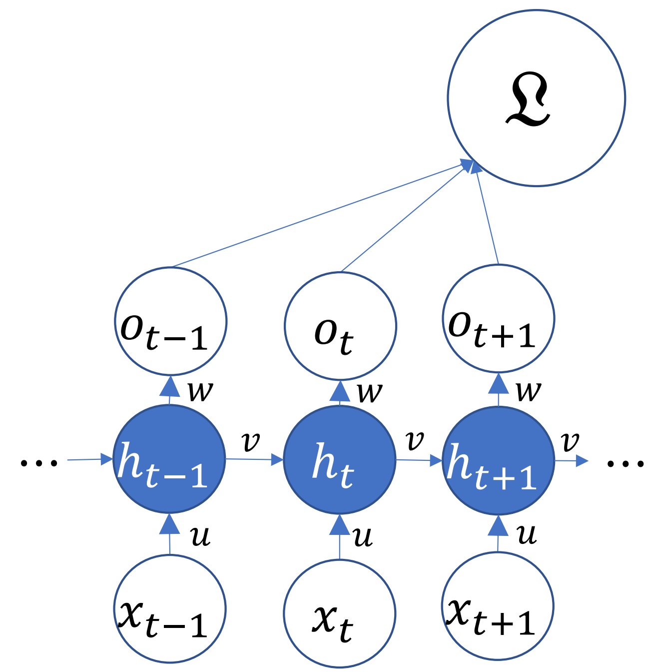
- The first term measures losses caused directly by $h_{i,t}$, for example, if $o_{i,t}$ is wrong.
- The second term measures losses caused indirectly, for example, because $h_{i,t}$ caused $h_{j,t+1}$ to be wrong.

Back-propagation through time

Notice that this is just like training a very deep network!

- Back-propagation through time: back-propagate from time step $t + 1$ to time step t
- Back-propagation in a very deep network: back-propagate from layer $l + 1$ to layer l

Toolkits like PyTorch may use the same code in both cases.



Outline

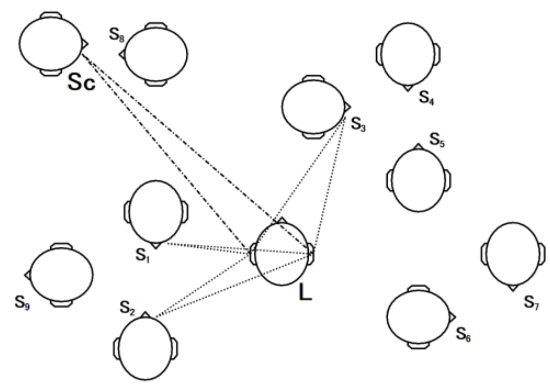
- Recurrent neural networks
- Attention
- Self-attention, Multi-headed attention, Cross-attention, and Masked attention
- Positional embedding

The Cocktail-Party Effect

- If you are focusing on one person's voice, but hear your name spoken by another person, your attention immediately shifts to the second voice.
- This "cocktail-party effect" suggests a model of hearing in which all sounds are processed preconsciously. Trigger sounds in an unattended source will cause attention to re-orient to that source.

https://commons.wikimedia.org/wiki/File:Cocktail_party_attendees_at_Fuller_Lodge,_1946.jpg

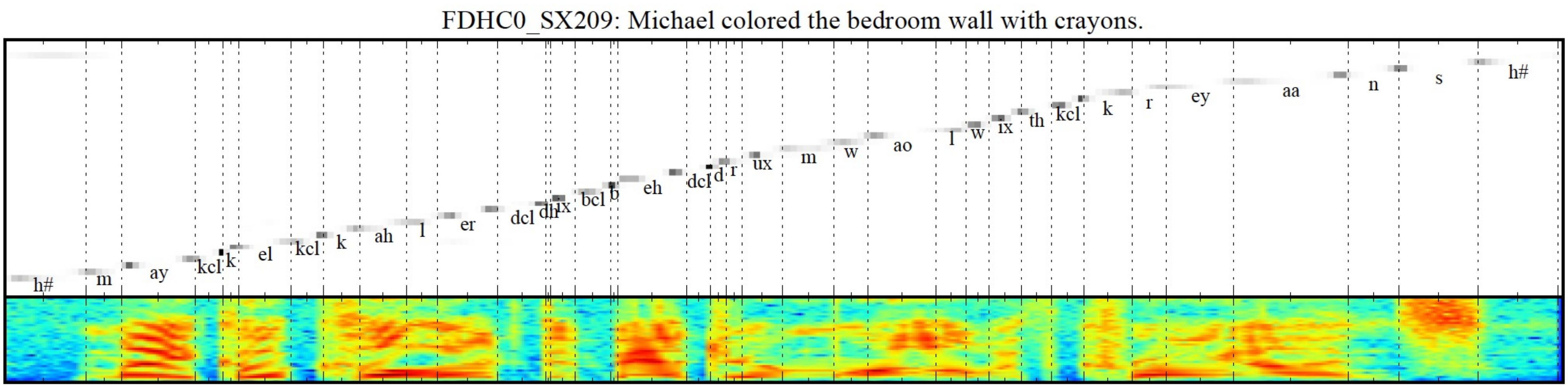
https://commons.wikimedia.org/wiki/File:Cocktail_Party_At_The_Imperial_Hotel_March_13,_1961_(Tokyo,_Japan)_(496610682).jpg



https://commons.wikimedia.org/wiki/File:Cocktail-party_effect.svg

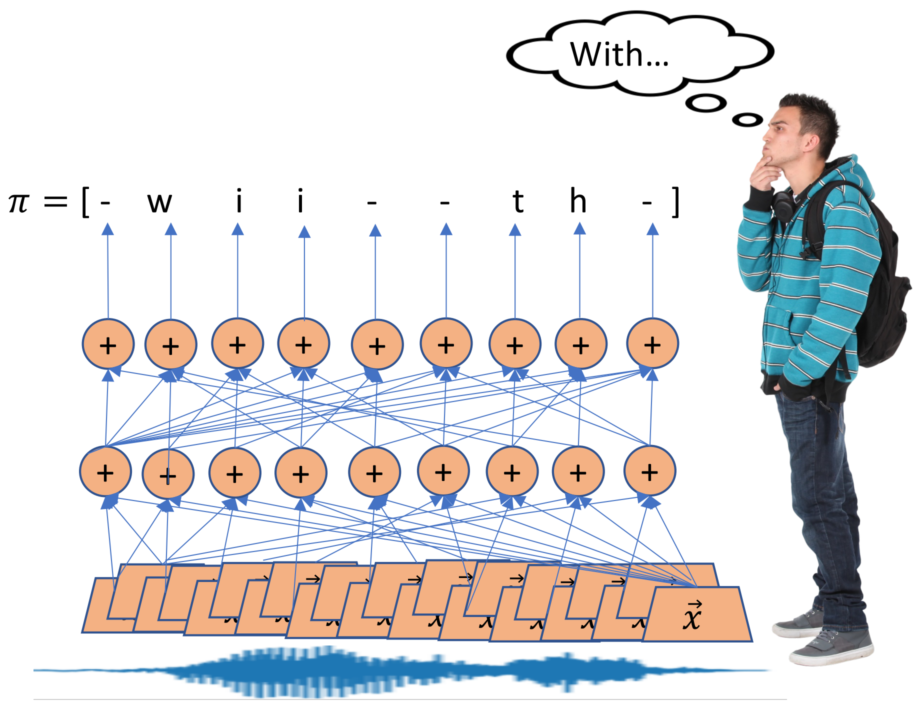
Bottom-up attention as a strategy for machine listening

- In 2014, researchers proposed that the past 200ms of RNN state vectors should be stored in a "short-term memory buffer"
- A speech recognizer can attend to several centiseconds, all at one time, to decide what words it thinks it is hearing



Chorowski, Bahdanau, Serdyk, Cho & Bengio, [Attention-Based Models for Speech Recognition](), Fig. 1

The Transformer: "Attention is all you need"



- In 2017, researchers proposed that the short-term memory buffer should contain raw signals, not processed signals.
- All processing is done using a model of bottom-up attention.

Attention: Key concepts

- The neural net needs to make a series of decision vectors, $\boldsymbol{o}_i$
- Each decision needs to be based on some context vector, $\boldsymbol{c}_i$
- Each context vector is a weighted sum of input values, $\boldsymbol{c}_i = \sum_t \alpha_{i,t} \boldsymbol{v}_t$
- $\alpha_{i,t}$ is the amount of attention that the output decision $\boldsymbol{o}_i$ is paying to the input value $\boldsymbol{v}_t$. It is based on the similarity between a key vector, $\boldsymbol{k}_t$, that describes the type of information available in $\boldsymbol{v}_t$, and a query vector, $\boldsymbol{q}_i$, that describes the type of information necessary in order to make the output decision

Inputs to an attention network

- Neural net inputs: a sequence of row vectors, $\boldsymbol{x}_t$
- Neural net outputs: a sequence of row vectors, $\boldsymbol{o}_i$
- Value: What type of information should $\boldsymbol{x}_t$ provide to the output? This may be just a linear transform of $\boldsymbol{x}_t$, e.g.: $\boldsymbol{v}_t = \boldsymbol{W}_V \boldsymbol{x}_t$
- Query: What type of information does o_i need? This may be just a linear transform of $\boldsymbol{o}_{i-1}$, e.g.: $\boldsymbol{q}_i = \boldsymbol{W}_Q \boldsymbol{o}_{i-1}$
- Key: The dot product $\boldsymbol{q}_i^T \boldsymbol{k}_t$ should be positive if $\boldsymbol{v}_t$ is useful, and negative if $\boldsymbol{v}_t$ is useless. This may be $\boldsymbol{k}_t = \boldsymbol{W}_K \boldsymbol{x}_t$

Attention = a probability mass over time

- Attention is like probability: You only have a fixed amount of attention, so you need to decide how to distribute it.
- $\alpha_{i,t} = P(\boldsymbol{v}_t | q_i)$ = the probability that $\boldsymbol{v}_t$ is the context that you need in order to make a decision related to the query vector $\boldsymbol{q}_i$.

$$\sum_t \alpha_{i,t} = 1$$

- Each output context vector ($\boldsymbol{c}_i$) is based on some input value vectors ($\boldsymbol{v}_t$). But which ones? Answer: decide which inputs to pay attention to, then pay attention.

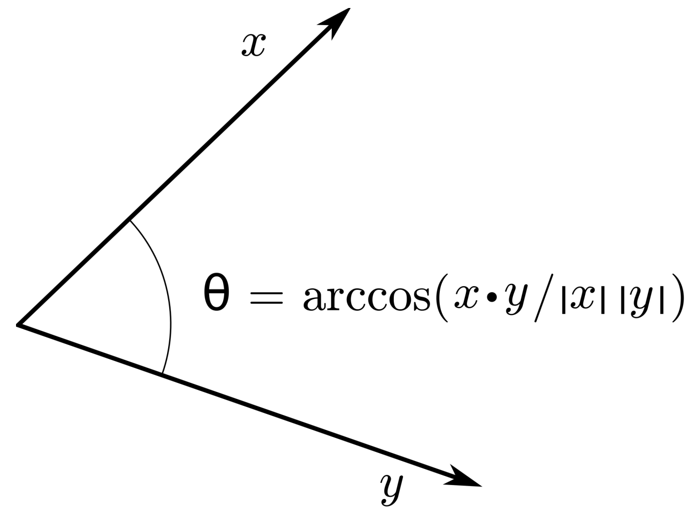
$$\boldsymbol{c}_i = \sum_t \alpha_{i,t} \boldsymbol{v}_t$$

Dot-product attention

How can you decide which value vectors, v_t are most relevant to a particular query? Answer:

1. Create a key vector, $\boldsymbol{k}_t$, such that $\boldsymbol{q}_i^T \boldsymbol{k}_t > 0$ if v_t is relevant to $\boldsymbol{q}_i$, otherwise $\boldsymbol{q}_i^T \boldsymbol{k}_t < 0$.
2. Convert the similarity measures into a probability distribution using softmax:

$$\alpha_{i,t} = \frac{\exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_t\right)}{\sum_\tau \exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_\tau\right)}$$



By BenFrantzDale at the English Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=49972362

Putting it all together

- Stack up $\boldsymbol{v}_t$, $\boldsymbol{k}_t$, and $\boldsymbol{q}_i$ into matrices:

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_n^T \end{bmatrix}, \boldsymbol{K} = \begin{bmatrix} \boldsymbol{k}_1^T \\ \vdots \\ \boldsymbol{k}_n^T \end{bmatrix}, \boldsymbol{Q} = \begin{bmatrix} \boldsymbol{q}_1^T \\ \vdots \\ \boldsymbol{q}_m^T \end{bmatrix}$$

- $\alpha_{i,t}$ is the t[th] output of a softmax whose input vector is $\boldsymbol{K}\boldsymbol{q}_i$:

$$\alpha_{i,t} = \mathrm{softmax}_t(\boldsymbol{K}\boldsymbol{q}_i) = \frac{\exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_t\right)}{\sum_\tau \exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_\tau\right)}$$

- c_i is the product of the vector $\mathrm{softmax}(\boldsymbol{K}\boldsymbol{q}_i)$ times the $\boldsymbol{V}^{\boldsymbol{T}}$ matrix:

$$\boldsymbol{c}_i = \boldsymbol{V}^{\boldsymbol{T}} \mathrm{softmax}(\boldsymbol{K}\boldsymbol{q}_i) = \sum_t \alpha_{i,t}\, \boldsymbol{v}_t$$

Quiz!

Try the quiz!

https://us.prairielearn.com/pl/course_instance/147925/assessment/2412318

Outline

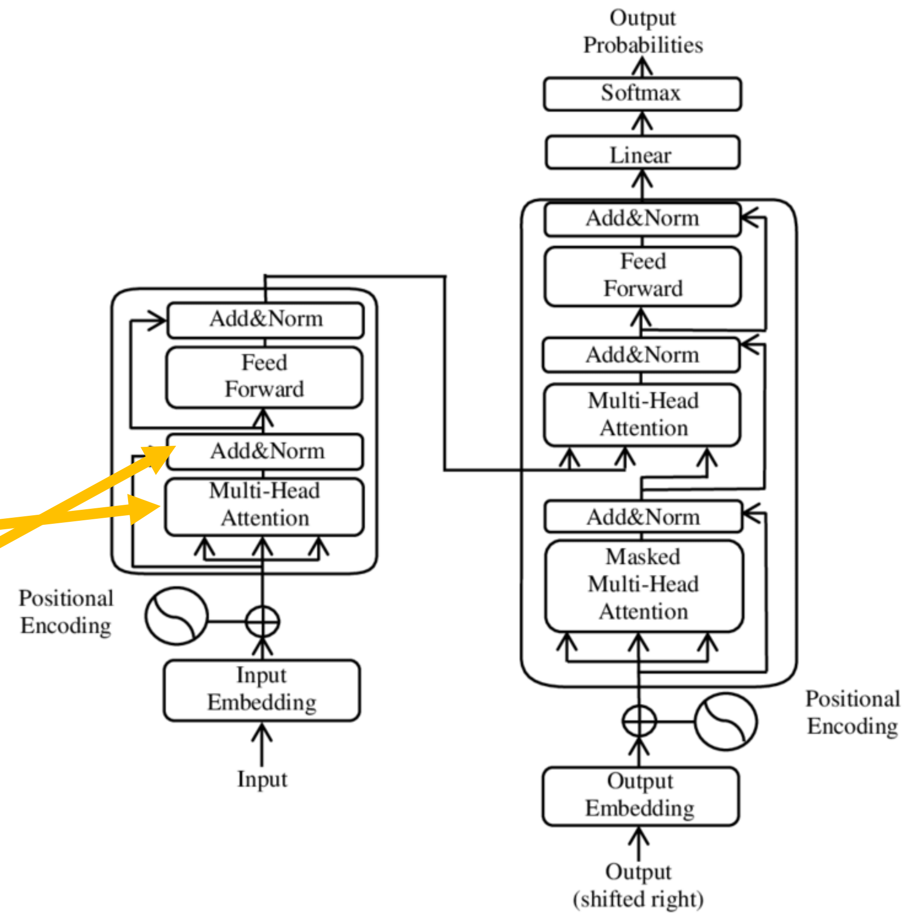
- Recurrent neural networks
- Attention
- Self-attention, Multi-headed attention, Cross-attention, and Masked attention
- Positional embedding

Self-attention

Self-attention (literally!) adds context to each input vector:

$$\boldsymbol{q}_i = \boldsymbol{W}_Q \boldsymbol{x}_i$$
$$\boldsymbol{k}_t = \boldsymbol{W}_K \boldsymbol{x}_t$$
$$\boldsymbol{v}_t = \boldsymbol{W}_V \boldsymbol{x}_t$$
$$\boldsymbol{c}_i = \boldsymbol{V}^{\boldsymbol{T}} \mathrm{softmax}(\boldsymbol{K}\boldsymbol{q}_i)$$

$$y_i = \frac{\boldsymbol{x}_i + \boldsymbol{c}_i - E[\boldsymbol{x}_i + \boldsymbol{c}_i]}{\sqrt{Var(\boldsymbol{x}_i + \boldsymbol{c}_i)}}$$



Multi-headed-attention

Multi-headed-attention uses 8 different w_Q, w_K, and w_V matrices, in order to get 8 different views of the input data:

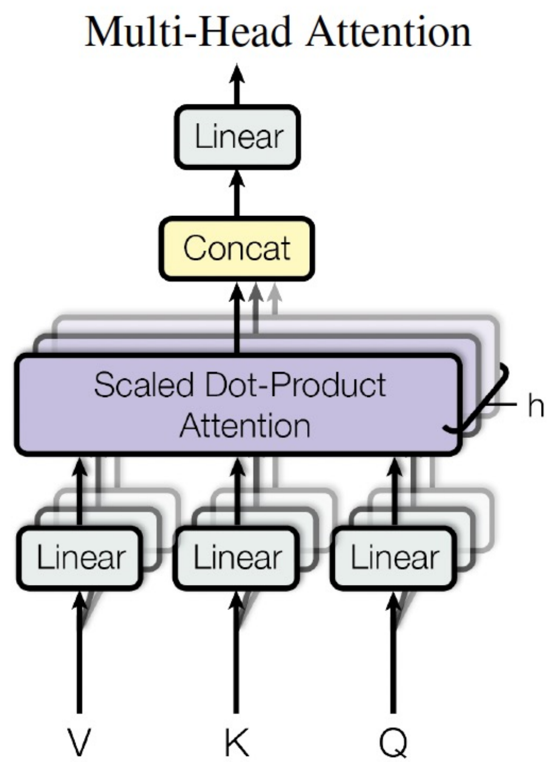
$$\boldsymbol{q}_{j,i} = \boldsymbol{W}_{j,Q}\boldsymbol{x}_{j,i}, \qquad 1 \le j \le 8$$

$$\boldsymbol{k}_{j,t} = \boldsymbol{W}_{j,K}\boldsymbol{x}_{j,t}, \qquad 1 \le j \le 8$$

$$\boldsymbol{v}_{j,t} = \boldsymbol{W}_{j,V}\boldsymbol{x}_{j,t}, \qquad 1 \le j \le 8$$

$$\boldsymbol{h}_{j,i} = \boldsymbol{V}_j^T \mathrm{softmax}\left(\boldsymbol{K}_j \boldsymbol{q}_{j,i}\right)$$

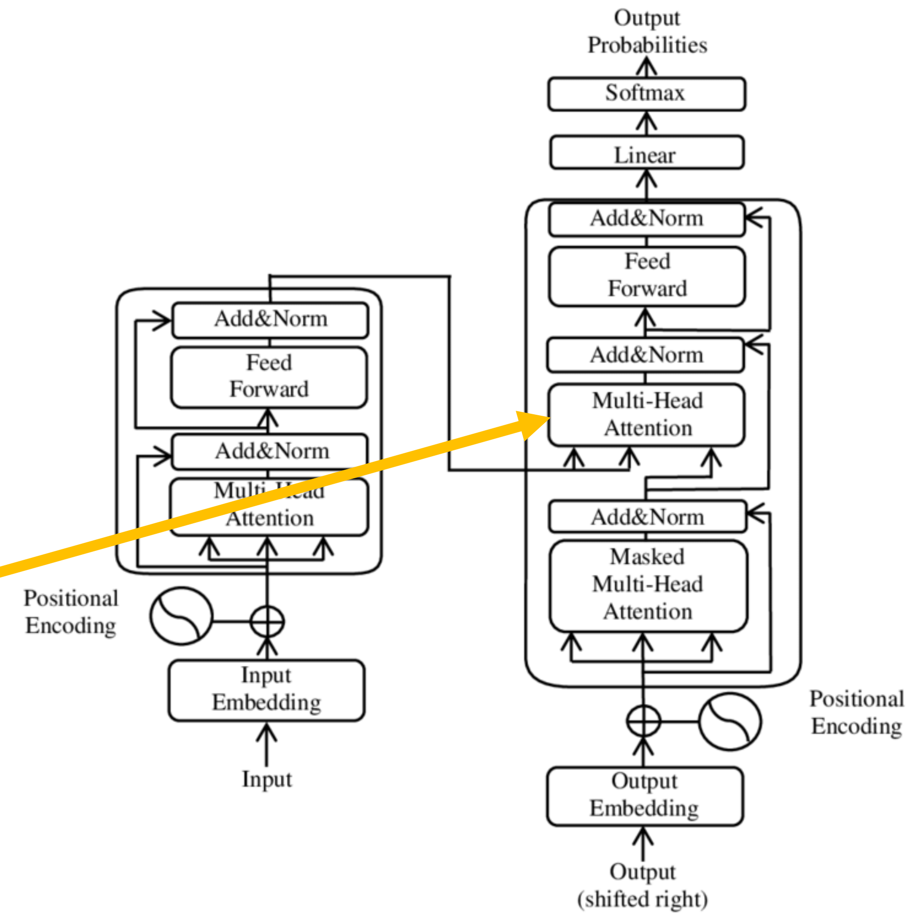
$$\boldsymbol{c}_i = \boldsymbol{W}_{j,O}\begin{bmatrix}\boldsymbol{h}_{1,i}\\ \vdots \\ \boldsymbol{h}_{8,i}\end{bmatrix}$$



Cross-attention

Cross-attention: query depends on preceding output, key and value depend on input:

$$\boldsymbol{q}_{j,i} = \boldsymbol{W}_{j,Q}\boldsymbol{o}_{j,i-1}$$
$$\boldsymbol{k}_{j,t} = \boldsymbol{W}_{j,K}\boldsymbol{x}_{j,t}$$
$$\boldsymbol{v}_{j,t} = \boldsymbol{W}_{j,V}\boldsymbol{x}_{j,t}$$
$$\boldsymbol{h}_{j,i} = \boldsymbol{V}_j^T \operatorname{softmax}\left(\boldsymbol{K}_j \boldsymbol{q}_{j,i}\right)$$



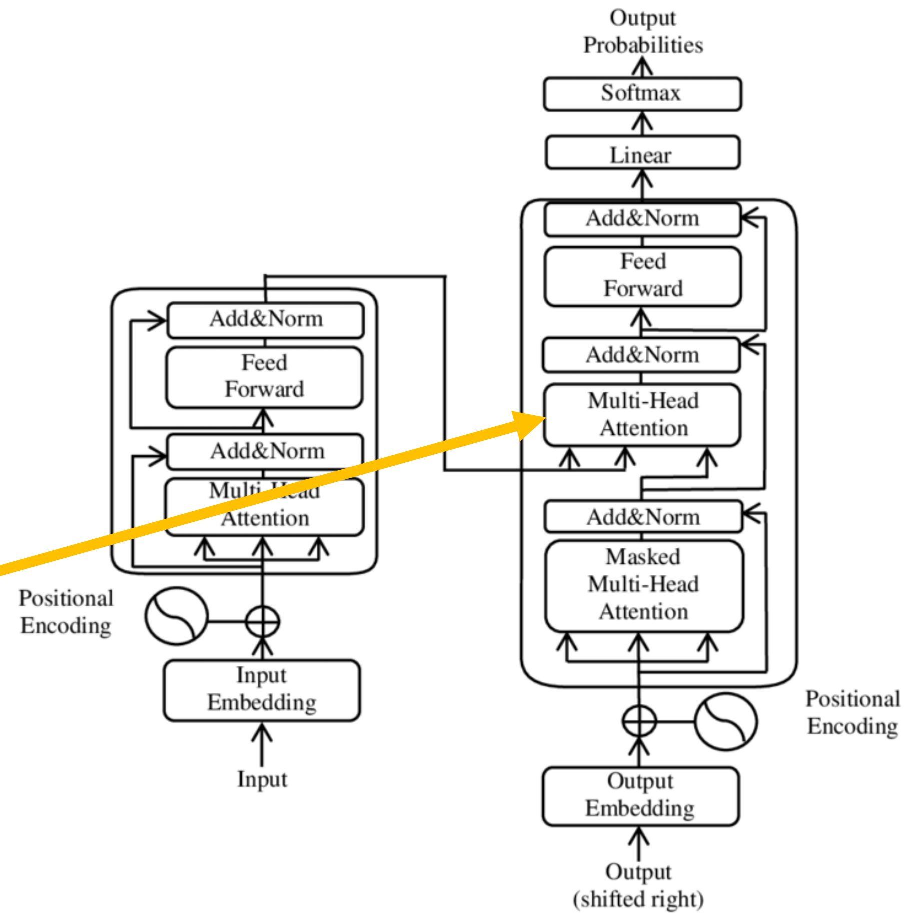
Masked attention

Masked attention forces c_i to pay attention to value vectors v_t only if $t < i$:

$$s(\boldsymbol{q}_i, \boldsymbol{k}_t) = \begin{cases} \boldsymbol{q}_i^T \boldsymbol{k}_t & t < i \\ -\infty & t \geq i \end{cases}$$

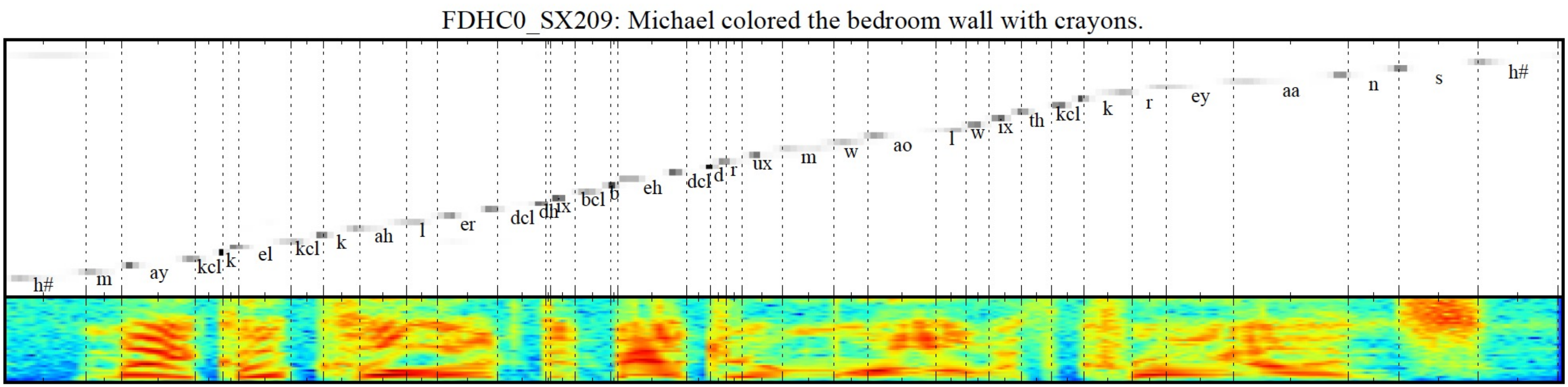
$$\alpha_{i,t} = \frac{\exp\big(s(\boldsymbol{q}_i, \boldsymbol{k}_t)\big)}{\sum_\tau \exp\big(s(\boldsymbol{q}_i, \boldsymbol{k}_\tau)\big)}$$

$$= \begin{cases} \text{softmax}\big(\boldsymbol{q}_i^T \boldsymbol{k}_t\big) & t < i \\ 0 & t \geq i \end{cases}$$



Cross-attention visualization

This plot shows $\alpha_{i,t}$ where i = output character, and t = input spectrum



Chorowski, Bahdanau, Serdyk, Cho & Bengio, [Attention-Based Models for Speech Recognition](), Fig. 1

Word Error Rates using Transformers

By 9/2020, transformers had error rates of:

- 2%: English, quiet recording conditions
- 4%: Chinese or Japanese, quiet recording conditions
- 5-7%: if the reference transcript has errors
- 14%: 2-talker mixtures, synthetic reverberation
- 38%: actual in-home recordings in noisy households

Table 1. CER/WER results on various open source ASR corpora. Both Transformer and Conformer models are implemented based on ESPnet toolkit. ∗ marks ESPnet2 results. † and ‡ indicate only w/ speed or only w/ SpecAugment, respectively. § denotes w/o any data augmentation.

Dataset	Vocab	Metric	Evaluation Sets	Transformer	Conformer
AIDATATANG	Char	CER	dev / test	(†) 5.9 / 6.7	**4.3 / 5.0**
AISHELL-1	Char	CER	dev / test	(†) 6.0 / 6.7	(*) **4.4 / 4.7**
AISHELL-2	Char	CER	android / ios / mic	(†) 8.9 / 7.5 / 8.6	**7.6 / 6.8 / 7.4**
AURORA4	Char	WER	dev_0330 (A / B / C / D)	**3.3 / 6.0 / 4.5** / 10.6	4.3 / 6.0 / 5.4 / **9.3**
CSJ	Char	CER	eval{1, 2, 3}	(∗) 4.7 / 3.7 / 3.9	(∗) **4.5 / 3.3 / 3.6**
CHiME4	Char	WER	{dt05, et05}_{simu, real}	(†) 9.6 / 8.2 / 15.7 / 14.5	**9.1 / 7.9 / 14.2 / 13.4**
Fisher-CallHome	BPE	WER	dev / dev2 / test / devtest / evltest	22.1 / 21.5 / 19.9 / 38.1 / 38.2	**21.5 / 21.1 / 19.4 / 37.4 / 37.5**
HKUST	Char	CER	dev	(†) 23.5	(†) **22.2**
JSUT	Char	CER	our split	(†) 18.7	**14.5**
LibriSpeech	BPE	WER	{dev, test}_{clean, other}	2.1 / 5.3 / 2.5 / 5.5	**1.9 / 4.9 / 2.1 / 4.9**
REVERB	Char	WER	et_{near, far}	(†) 13.1 / 15.4	(†) **10.5 / 13.9**
Switchboard	BPE	WER	eval2000 (callhm / swbd)	17.2 / 8.2	**14.0 / 6.8**
TEDLIUM2	BPE	WER	dev / test	9.3 / 8.1	**8.6 / 7.2**
TEDLIUM3	BPE	WER	dev / test	10.8 / 8.4	**9.6 / 7.6**
VoxForge	Char	CER	our split	(§) 9.4 / 9.1	(§) **8.7 / 8.2**
WSJ	BPE	WER	dev93/ eval92	(‡) **7.4 / 4.9**	(‡) 7.7 / 5.3
WSJ-2mix	Char	WER	tt	(§) 12.6	(§) **11.7**

Guo, Boyer, Chang, Hayashi, Higuchi et al., ICASSP 2021, © IEEE

Outline

- Recurrent neural networks
- Attention
- Self-attention, Multi-headed attention, Cross-attention, and Masked attention
- Positional Encoding

What we have lost…

- With the recurrent neural net, each state vector paid attention to the one that preceded it:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{U}\boldsymbol{v}_t + \boldsymbol{V}\boldsymbol{h}_{t-1})$$

- With a transformer, each state vector pays attention to the input that is most similar, regardless of what time it happened:

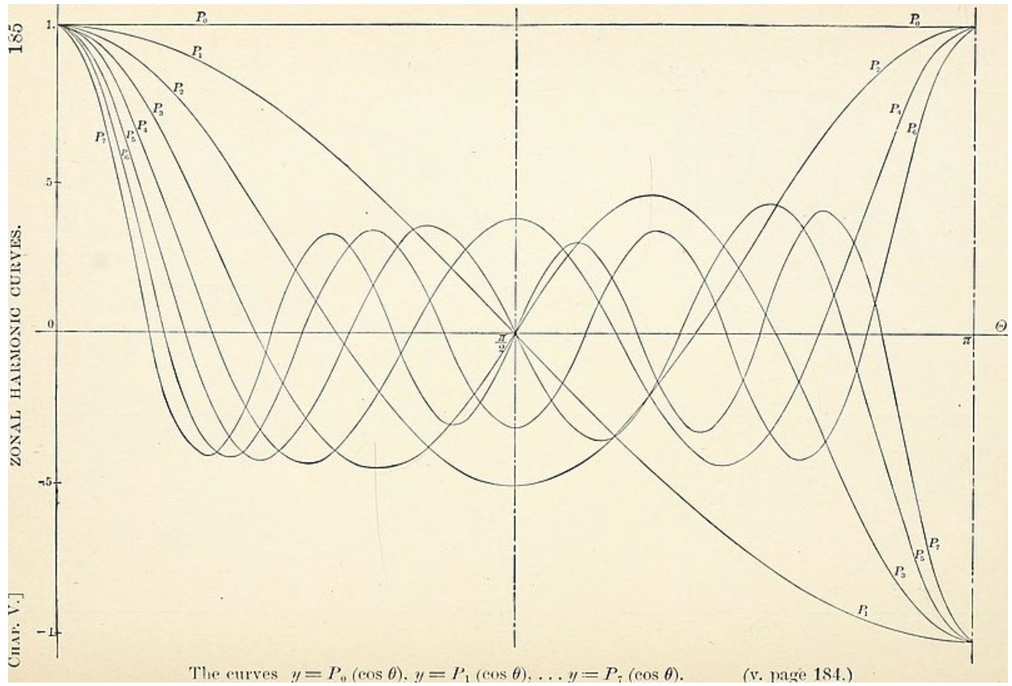
$$\boldsymbol{h}_i = \sum_t \alpha_{i,t}\,\boldsymbol{v}_t, \qquad \alpha_{i,t} = \frac{\exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_t\right)}{\sum_\tau \exp\left(\boldsymbol{q}_i^T \boldsymbol{k}_\tau\right)}$$

- What if we always want $\boldsymbol{h}_t$ to pay special attention to $\boldsymbol{v}_{t-1}$? Is that possible?

Position encoding by Fourier basis

The solution is to encode the relative position of each input, $\boldsymbol{v}_t$, using a Fourier basis $\boldsymbol{e}_t$:

$$\boldsymbol{e}_t = \begin{bmatrix} \cos\left(\frac{\pi t}{T}\right) \\ \sin\left(\frac{\pi t}{T}\right) \\ \vdots \\ \cos\left(\frac{\pi D t}{2T}\right) \\ \sin\left(\frac{\pi D t}{2T}\right) \end{bmatrix}$$



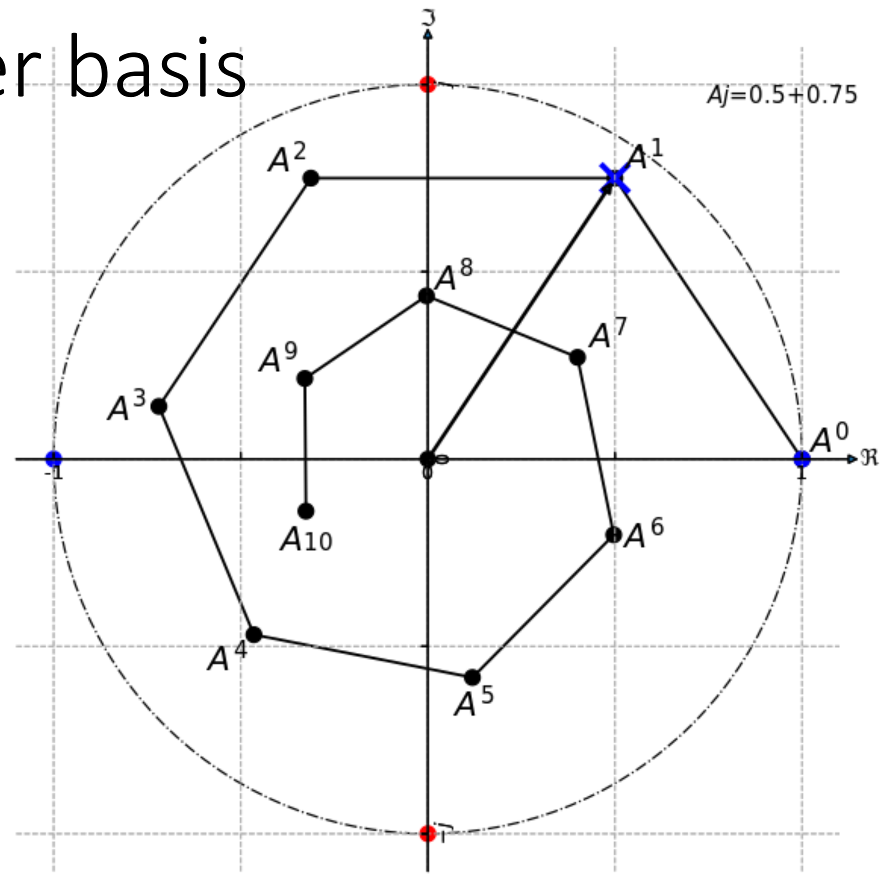
Public domain image, https://commons.wikimedia.org/wiki/File:An_elementary_treatise_on_Fourier%27s_series_and_spherical,_cylindrical,_and_ellipsoidal_harmonics,_with_applications_to_problems_in_mathematical_physics_(1893)_(14780364665).jpg

Position encoding by Fourier basis

The Fourier basis is useful because shifting by a fixed time offset, to $t-d$, can be accomplished by a matrix multiplication:

$$\boldsymbol{e}_{t-d} = \begin{bmatrix} \cos\left(\frac{\pi d}{T}\right) & \sin\left(\frac{\pi d}{T}\right) & \cdots \\ -\sin\left(\frac{\pi d}{T}\right) & \cos\left(\frac{\pi d}{T}\right) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \boldsymbol{e}_t$$

...so if we want a particular query to pay attention to vectors with a time delay of d, we just set $\boldsymbol{W}_{j,Q}$ to the matrix shown above.



Public domain image, https://commons.wikimedia.org/wiki/File:Exponentials_of_complex_number_within_unit_circle-2.svg

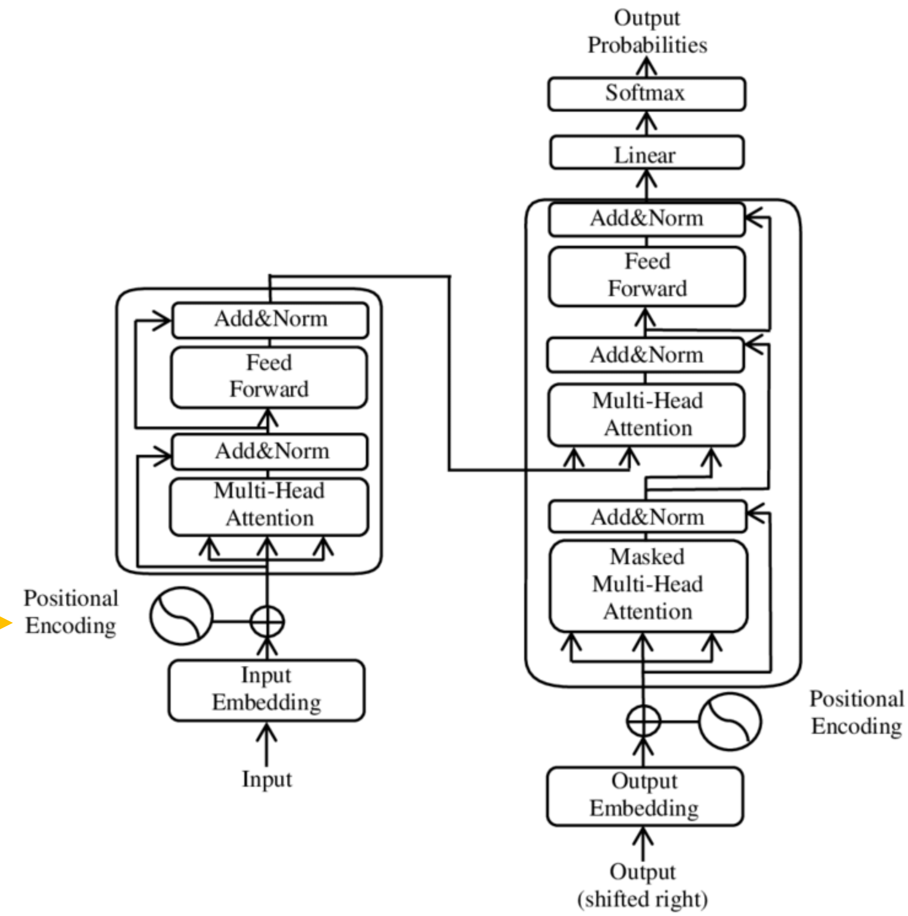
Where do we put the positional encoding?

- Possibility #1: Concatenate it, i.e., $\boldsymbol{x}_t^T = [\boldsymbol{x}_t^T, \boldsymbol{e}_t^T]$
  - Advantage: $\boldsymbol{W}_{j,Q}$ can learn to operate separately on the content $\boldsymbol{x}_t^T$ and the positional encoding $\boldsymbol{e}_t^T$
  - Disadvantage: every vector is twice as large, and every matrix is four times as large
- Possibility #2: Add it, i.e., $\boldsymbol{x}_t = \boldsymbol{x}_t + \boldsymbol{e}_t$
  - Advantage: fewer parameters to learn
  - Disadvantage: $\boldsymbol{W}_{j,Q}$ can only operate directly on $\boldsymbol{e}_t$ if $\boldsymbol{x}_t$ is mostly zero
  - Surprise: this works well in practice. Apparently, the positional encoding can learn to ignore local fluctuations in $\boldsymbol{x}_t$, and pretend that it's mostly 0 on average

Positional encoding

In the standard transformer, position of the input is encoded using

$$\boldsymbol{x}_t = \boldsymbol{x}_t + \begin{bmatrix} \cos\left(\dfrac{\pi t}{T}\right) \\ \sin\left(\dfrac{\pi t}{T}\right) \\ \vdots \\ \cos\left(\dfrac{\pi D t}{2T}\right) \\ \sin\left(\dfrac{\pi D t}{2T}\right) \end{bmatrix}$$



Summary

- Recurrent neural networks

$$\boldsymbol{h}_t = \tanh(\boldsymbol{U}\boldsymbol{v}_t + \boldsymbol{V}\boldsymbol{h}_{t-1})$$

- Attention

$$\boldsymbol{c}_i = \boldsymbol{V}^{\boldsymbol{T}}\mathrm{softmax}(\boldsymbol{K}\boldsymbol{q}_i) = \sum_t \frac{\exp\left(\boldsymbol{q}_i^T\boldsymbol{k}_t\right)}{\sum_\tau \exp\left(\boldsymbol{q}_i^T\boldsymbol{k}_\tau\right)}\boldsymbol{v}_t$$

- Self-attention, Multi-headed attention, Cross-attention, and Masked attention
- Positional encoding

$$\boldsymbol{x}_t \mathrel{+}= \begin{bmatrix} \cos\left(\dfrac{\pi t}{T}\right) \\ \sin\left(\dfrac{\pi t}{T}\right) \\ \vdots \end{bmatrix}$$