Natural Language Processing (NLP) and Large Language Models (LLMs)
Lecture 7-1: Attention

Chendi Wang (王晨笛)
chendi.wang@xmu.edu.cn

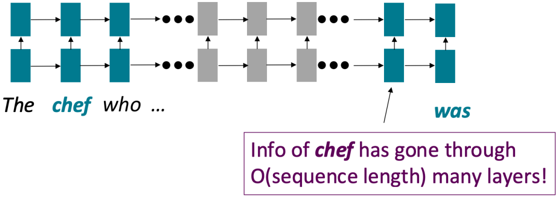
WISE @ XMU

2025 年 4 月 15 日

Adapted in part from CS 224n and Dive into deep learning

Recap: issues with RNNs

- Linear interaction distance.



- Lack of parallelizability!

An example from psychology: distracted by “red” coffee

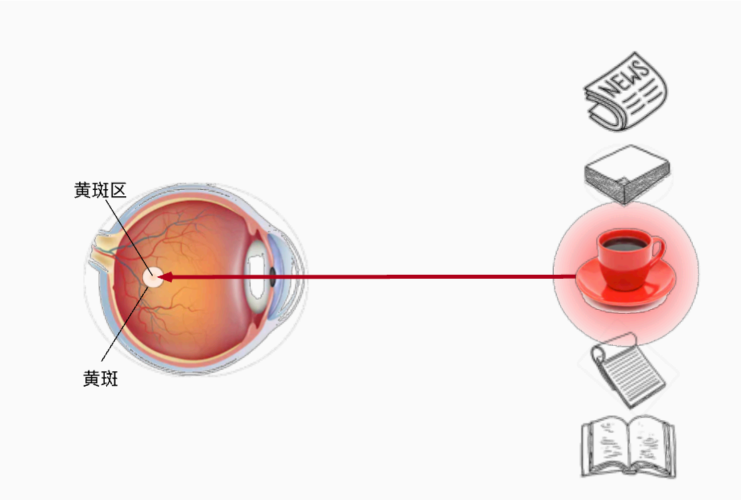


Figure is from `https://zh.d2l.ai/chapter_attention-mechanisms/attention-cues.html`

Attention please!

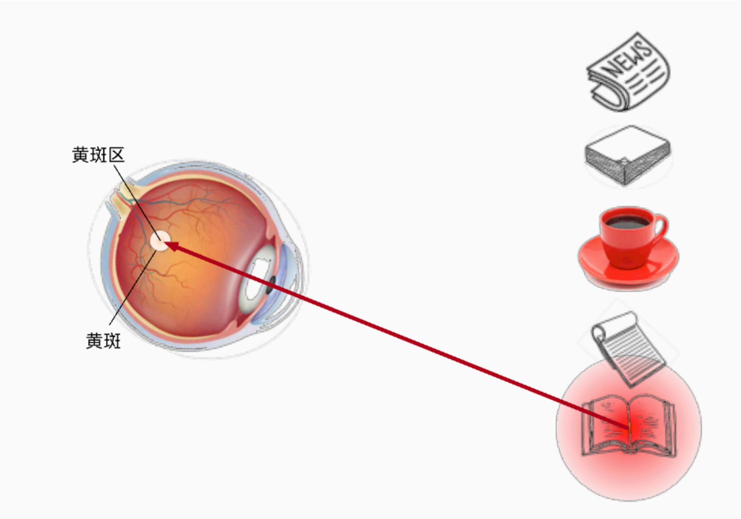
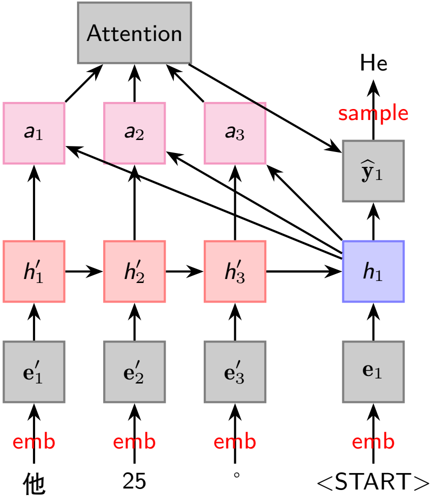


Figure is from `https://zh.d2l.ai/chapter_attention-mechanisms/attention-cues.html`

Attention please!



Reading materials

- Neural machine translation by jointly learning to align and translate, Bahdanau et al., 2014.
- Attention Is All You Need, Vaswani et al., 2017
- Note: The phrase "Is All You Need" or "Is not All You Need" has become overused—avoid it in your titles...

1. Section 1: Queries, keys, and values

2. Section 2: Attention for language models

3. Section 3: Implementation in PyTorch

Key-value pairs

- Consider a dataset {(Chendi,Teacher), (Jingtao, TA),(Xiaoming, Student),..., (Xiaomei, Student)}
- Key: name (such as Chendi); value: position (Teacher; TA, Student)
- You use it frequently in python dictionaries.
- Each pair of data point can be rewritten as a tuple $(\mathbf{k}_i, \mathbf{v}_i)$, where $\mathbf{k}_i$ is the key and $\mathbf{v}_i$ is the value.

Attention Mechanism, I

- We define a database $\mathcal{D} = \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}$ consisting of m key-value pairs.
- For each value $\mathbf{v}_i$, we compute a weight α_i based on its corresponding key $\mathbf{k}_i$.
- The attention mechanism emphasizes values with higher weights:

$$\text{Attention}(\mathcal{D}) = \sum_{i=1}^{m} \alpha_i \mathbf{v}_i.$$

- These weights are determined through an operation called query.
- Intuition: Different queries should focus attention on different values in the database.
- α_i is termed attention weights.

Query

- Query is a broad concept, a query operates on each key-value pair $(\mathbf{k}_i, \mathbf{v}_i)$.
- Denote by $\mathbf{q}$ a query.
- Exact query: The exact query for "Xiaoming" returns its value "Student".
- The would be no valid answer if (Xiaoming, Student) is not in the database.
- Approximate matches: it may return (Xiaomei, Student) instead as (Xiaomei, Student) is similar to (Xiaoming, Student).

Attention Mechanism, II

- Recall the database $\mathcal{D} = \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}$ consisting of m key-value pairs.
- The attention mechanism emphasizes values with higher weights:

$$\text{Attention}(\mathcal{D}) = \sum_{i=1}^{m} \alpha_i \mathbf{v}_i = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i.$$

- $\alpha(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}$ are scalar attention weights.
- $\alpha(\mathbf{q}, \mathbf{k}_i) \geq 0$ and $\sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i) = 1$.

Examples of attention weights

- One-hot: exact one of the weights $\alpha_i = 1$ and the rest $m - 1$ weights are 0.
- Averaging: $\alpha_i \equiv \frac{1}{m}$, aka, average pooling in deep learning.
- For any function $a(\mathbf{q}, \mathbf{k}_i)$ we can apply the SoftMax operation and define

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))}.$$

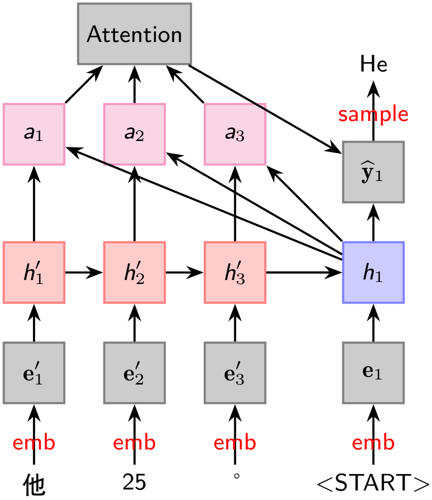
- A straightforward example is $a(\mathbf{q}, \mathbf{k}_i) = \mathbf{q}^T \mathbf{k}_i$.

(Scaled) Dot product Attention

- $a(\mathbf{q}, \mathbf{k}_i) = -\frac{1}{2}\|\mathbf{q} - \mathbf{k}_i\|_2^2$ (pay more attention to the keys that are closer to the query).
- $a(\mathbf{q}, \mathbf{k}_i) = \mathbf{q}^T\mathbf{k}_i - \frac{1}{2}\|\mathbf{q}\|^2 - \frac{1}{2}\|\mathbf{k}_i\|^2$
- Applying the SoftMax function eliminates the term $-\frac{1}{2}\|\mathbf{q}\|^2$
- Layer normalization normalizes the term $-\frac{1}{2}\|\mathbf{k}_i\|^2$.
- Rescale it (layer normalization) and we obtain the scaled dot product attention

$$a(\mathbf{q}, \mathbf{k}_i) = \mathbf{q}^T\mathbf{k}_i/\sqrt{d}$$

Recap


Attention
He
sample
a_1
a_2
a_3
$\widehat{\mathbf{y}}_1$
h'_1
h'_2
h'_3
h_1
$\mathbf{e}'_1$
$\mathbf{e}'_2$
$\mathbf{e}'_3$
$\mathbf{e}_1$
emb
emb
emb
emb
他
25
。
<START>

Seq2seq meets attention

- Source sequence: $\mathbf{x}' = (x_1, \cdots, x_{L_0})$ with embeddings $\mathbf{e}' = (\mathbf{e}'_1, \cdots, \mathbf{e}'_{L_0})$.
- Target sequence: $\mathbf{x} = (x_1, \cdots, x_L)$ with embeddings $\mathbf{e} = (\mathbf{e}_1, \cdots, \mathbf{e}_L)$.
- Modeling $p(\cdot|x_1, \cdots, x_t, \mathbf{x}') \approx \widehat{\mathbf{y}}_t = f(h_t, \mathbf{e}_t, c_t)$
- $c_t = c_t(\mathbf{x}', h_t)$ is from an attention mechanism.

Seq2seq meets attention

- Query: h_t
- Key & value: h'_t
- $$\alpha_{ts} = \frac{\exp(a(h_t, h'_s))}{\sum_{l=1}^{L_0} \exp(a(h_t, h'_l))}$$
- $$c_t = \sum_s \alpha_{ts} h'_s$$
- $$a(h_t, h'_s) = \mathbf{v}_a^T \left(\mathbf{W}_{ah} h_t + \mathbf{W}'_{ah} h'_s\right) ,$$

  here $\mathbf{v}_a, \mathbf{W}_{ah}, \mathbf{W}'_{ah}$ are all trainable parameters.

Encoder-decoder

- Encoder: h'_t is the hidden state of an encoding RNN.
- Decoder: h_t is the hidden state of a RNN but with the bias term $\mathbf{b} = \mathbf{W}_c c_t$ (to be specified next page).
- Final output (Bahdanau et al., 2014):

$$\widehat{\mathbf{y}}_t = \mathrm{SoftMax}\left(\mathbf{W}_{yh} h_t + \mathbf{W}_{ye} \mathbf{e}_t + \mathbf{W}_{yc} c_t\right)$$

Hidden state of the decoder

- Let's consider GRU which is used by Bahdanau et al. (2014).
- Using attention term c_i, the reset gate is

$$\mathbf{R}_t = \sigma_t(\mathbf{W}_{rh} h_{t-1} + \mathbf{W}_{re}\mathbf{e}_t + \mathbf{W}_{rc} c_t)$$

- Then, the candidate hidden state $\widetilde{h}_t$ is given by

$$\widetilde{h}_t = \sigma_h\left(\mathbf{W}_{hh}(\mathbf{R}_t \odot h_{t-1}) + \mathbf{W}_{he}\mathbf{e}_t + \mathbf{W}_{hc} c_t\right).$$

- Similarly, the update gate is

$$\mathbf{Z}_t = \sigma_z(\mathbf{W}_{zh} h_{t-1} + \mathbf{W}_{ze}\mathbf{e}_t + \mathbf{W}_{zc} c_t)$$

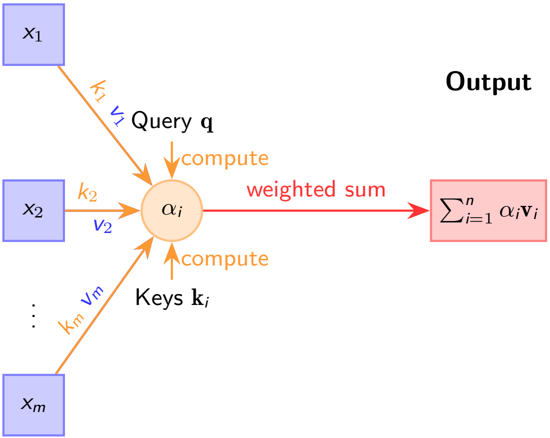
and the final hidden state is

$$h_t = \mathbf{Z}_t \odot h_{t-1} + (1 - \mathbf{Z}_t) \odot \widetilde{h}_t.$$

Self-Attention vs. Cross-Attention

- **Cross-Attention:**
  - Query ($\mathbf{q}$) comes from one sequence (e.g., the target sentence)
  - Key ($\mathbf{k}$) and Value ($\mathbf{v}$) come from another different sequence (e.g., the source sentence)
- **Self-Attention:** (More widely used)
  - $\mathbf{q}$, $\mathbf{k}$, $\mathbf{v}$ all derive from the same sequence
  - Example: When predicting token x_t, all vectors $\mathbf{q}$, $\mathbf{k}$, $\mathbf{v}$ come from the first $t-1$ tokens of that sentence

Self-attention: graph



Self-attention: query, key, value

- Recall: a sequence $\mathbf{x}$ and the corresponding embeddings $\mathbf{e} = (\mathbf{e}_1, \cdots, \mathbf{e}_L)$.
- Suppose that $\mathbf{e}_t \in \mathbb{R}^d$ for $i = 1, \cdots, L$.
- To generate the queries, keys, and values from the embeddings, we introduce three matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{d \times d}$.
- Query: $\mathbf{q}_t = \mathbf{Q}\mathbf{e}_t$; Key: $\mathbf{k}_t = \mathbf{K}\mathbf{e}_t$; Value: $\mathbf{v}_t = \mathbf{V}\mathbf{e}_t$
- $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{d \times d}$ are also weights (trainable parameters).
- We denote the output of a self-attention layer as $h_t = \mathrm{Attention}(\mathbf{e}_t, \mathbf{Q}, \mathbf{K}, \mathbf{V})$.

Self-attention: output

- $a_{ts} = a(\mathbf{q}_t, \mathbf{k}_s) = \mathbf{q}_t^T \mathbf{k}_s / \sqrt{d}$
- $\alpha_{ts} = \frac{\exp(a_{ts})}{\sum_l \exp(a_{tl})}$
- Output of a self-attention layer: $h_t = \sum_s \alpha_{ts} \mathbf{v}_s =: \mathrm{Attention}(\mathbf{e}_t; \mathbf{Q}, \mathbf{K}, \mathbf{V})$.

The position information has not been considered so far!

- Self-attention mechanisms lack inherent position awareness.
- We encode the position information use another vector $\mathbf{p}_t$.
- $\mathbf{p}_t$ should be incorporated in the query, the key, and the value at position t.
- Solution: Simply add position vectors to embeddings (Simple but effective solutions are elegant).
- The positioned embeddings:

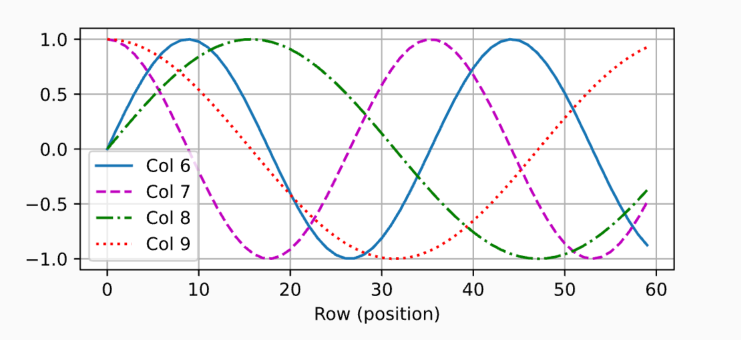
$$\widetilde{\mathbf{e}}_t = \mathbf{e}_t + \mathbf{p}_t.$$

Position embeddings through trigonometric functions

- Let $\mathbf{p}_t = (p_{t,1}, \cdots, p_{t,d})$.
- For the odd term $p_{t,2j+1}$, we let $p_{t,2j+1} = \sin(t/10000^{2j/d})$.
- For the even term $p_{t,2j}$, we let $p_{t,2j} = \cos(t/10000^{2j/d})$

Visualize the position embeddings

- Let $\mathbf{P} \in \mathbb{R}^{L \times d}$ be a matrix with the i-th row being $\mathbf{p}_i$.
- One can visualize it as follows.
- The frequency of the 6-th and 7-th column is lower than the 8-th and the 9-th column



Absolute position

- In binary representations, a higher bit has a lower frequency than a lower bit.

```
0 in binary is 000
1 in binary is 001
2 in binary is 010
3 in binary is 011
4 in binary is 100
5 in binary is 101
6 in binary is 110
7 in binary is 111
```

Relative position

- For any $\delta > 0$, there is a rotation matrix

$$R_\delta = \begin{bmatrix} \cos(\delta\omega_1), & \sin(\delta\omega_1), & \cos(\delta\omega_2), & \sin(\delta\omega_2), & \cdots \\ -\sin(\delta\omega_1), & \cos(\delta\omega_1), & -\sin(\delta\omega_2), & \cos(\delta\omega_2) & \cdots \end{bmatrix},$$

with $\omega_j = 1/10000^{2j/d}$ such that

$$R_\delta \mathbf{p}_t = \mathbf{p}_{t+\delta}.$$

More advanced position encoder

- Self-Attention with Relative Position Representations. Shaw et al., 2018.
- Self-Attention with Structural Position Representations. Wang et al., 2019.

Multiple self-attention layers

- Recall a single self-attention layer: $\mathrm{Attention}(\mathbf{e}_t, \mathbf{Q}^{[0]}, \mathbf{K}^{[0]}, \mathbf{V}^{[0]})$.
- Let the output of the first layer be $h_t^{[1]} = \mathrm{Attention}(\mathbf{e}_t)$.
- What if we consider stacking another layer:

$$\mathrm{Attention}(h_t^{[1]}, \mathbf{Q}^{[1]}, \mathbf{K}^{[1]}, \mathbf{V}^{[1]})?$$

Elementwise nonlinearity

- Let $h_t^{\text{Self-attention}} = \text{Attention}(\mathbf{e}_t; \mathbf{Q}, \mathbf{K}, \mathbf{V}) \in \mathbb{R}^d$.
- Apply a nonlinear layer immediately after the self-attention layer:

$$h_t = \mathbf{W}_2 \sigma \left(\mathbf{W}_1 h_t^{\text{Self-attention}} + \mathbf{b}_1 \right) + \mathbf{b}_2$$

- Here, $\mathbf{W}_1 \in \mathbb{R}^{d_1 \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d \times d_1}$; σ is typically ReLU.
- The intermediate dimension d_1 is often much larger than d, as matrix multiplication is highly parallelizable.

We Can't Look at the Future When Predicting It!

We Can't Look at the Future When Predicting It!

- Recall that our goal is to predict the next word x_t based on the preceding words $x_1, \cdots, x_{t-1}$.
- To avoid access to future information, a uni-directional RNN processes the sequence in a left-to-right manner, attending only to past inputs.
- How can we ensure that a self-attention layer does not attend to future tokens?

Future masking

- We ensure that a self-attention layer does not look at the future by applying a mask to the attention weights α_{ts}.
- Precisely, we can define

$$\alpha_{ts}^{mask} = \begin{cases} \alpha_{ts} & s \leq t, \\ 0, & \text{otherwise.} \end{cases}$$

An example

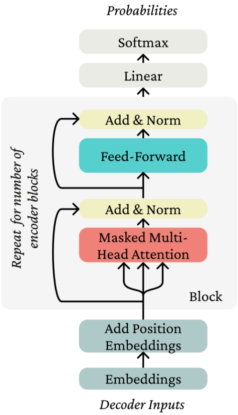
	Zuko	made	his	uncle	tea
Zuko		$-\infty$	$-\infty$	$-\infty$	$-\infty$
made			$-\infty$	$-\infty$	$-\infty$
his				$-\infty$	$-\infty$
uncle					$-\infty$
tea					

From `https://web.stanford.edu/class/cs224n/readings/cs224n-self-attention-transformers-2023_draft.pdf`

Summary

- *Attention is all you need*: self-attention is a core mechanism in modern sequence modeling.
- Positional representations are essential to encode the order of tokens.
- Nonlinearities: apply a feedforward neural network with non-linear activation after each self-attention layer.
- Future masking: prevent information leakage from future tokens—avoid “spoilers” .

Summary



Transformer Decoder

Figure is from Stanford cs224n

Dot Product Attention

- Recall the dot product attention: $a_{ts} = \mathbf{q}_t^\top \mathbf{k}_s$, and

$$h_t = \sum_s \alpha_{ts} \mathbf{v}_s,$$

  where α_{ts} are attention weights.
- Here, $\mathbf{q}_t$ and $\mathbf{k}_s$ are vectors of the same dimension d, and $\{(\mathbf{k}_s, \mathbf{v}_s)\}_{s=1}^m$ is a set of m key-value pairs.
- The computation can be expressed in matrix form as $\mathbf{Q} \in \mathbb{R}^{n\times d}$, $\mathbf{K} \in \mathbb{R}^{m\times d}$, and $\mathbf{V} \in \mathbb{R}^{m\times v}$. The corresponding attention output is:

$$\mathrm{SoftMax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V} \in \mathbb{R}^{n\times v}.$$

- Implementation in `PyTorch`: dot_product_attention.ipynb

What if $\mathbf{q}$ and $\mathbf{k}$ are of different dimensions

- Introduce a matrix $\mathbf{M}$ to solve the mis-match problem and define $a_{ts} = \mathbf{q}_t^T \mathbf{M} \mathbf{k}_s$.
- Additive Attention: $a_{ts} = \mathbf{w}_v^T \left[\tanh\left(\mathbf{W}_q \mathbf{q}_t + \mathbf{W}_k \mathbf{k}_s\right)\right] \in \mathbb{R}$
- Implementation in `PyTorch`: add_attention.ipynb