04. Transformers

Transformers may not fix all your NLP problems.

.

.

.

But they are worth some attention.

Source: Ramsri Goutham

CS 2731
Introduction to Natural Language Processing

Session 12: Transformers, LLMs part 1

Michael Miller Yoder

October 7, 2024

University of Pittsburgh | School of Computing and Information

Course logistics: projects

- If you haven't yet, please schedule a group meeting with me this week through my Bookings link in person in IS 604B or on Zoom
  - If you have follow-up questions or comments, don't hesitate to email me or set up office hours
- Project proposal and literature **due next Thu, Oct 17**
  - Instructions are on the project webpage
  - Look for NLP papers in ACL Anthology, Semantic Scholar, and Google Scholar

Course logistics: homework assignments

- Homework 3 will be released by tomorrow, is **due Thu Oct 24**
- Homework 2 was due last Thursday
- Kaggle competition results (private leaderboard)

Logistic regression

Team	Members	Score
Jerry Chen		0.655
Anveshika Kamble		0.637
Geonyeong Choi		0.603

Neural network

Team	Members	Score
Maanya Shanker		0.827
Geonyeong Choi		0.681
Jerry Chen		0.646

Midterm course evaluation (OMETs)

- https://go.blueja.io/NsyfmLUs5Ee4GQiFKcqQ3g
- All types of feedback are welcome (critical and positive)
- **Completely anonymous, will not affect grades**
- Let me know what's working and what to improve on while the course is still running!
- Please be as specific as possible
- Available until **Mon Oct 7 at 11:59pm**

Lecture overview: Transformers

- Self-attention
- Multi-headed attention
- Residual connections and layer normalization
- Transformer blocks

From recurrence to self-attention

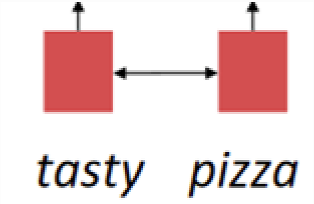
Transformers improved on RNNs and CNNs

- Google introduced Transformers in 2017 [Vaswani et al., "Attention is all you need"]
- At that time, most neural NLP models were based on
  - RNNs
  - CNNs
- These were good
- For many tasks, Transformers were better
- Has become the most successful NN architecture in NLP
- Adopted by famous pretrained LLMs (BERT, GPT)

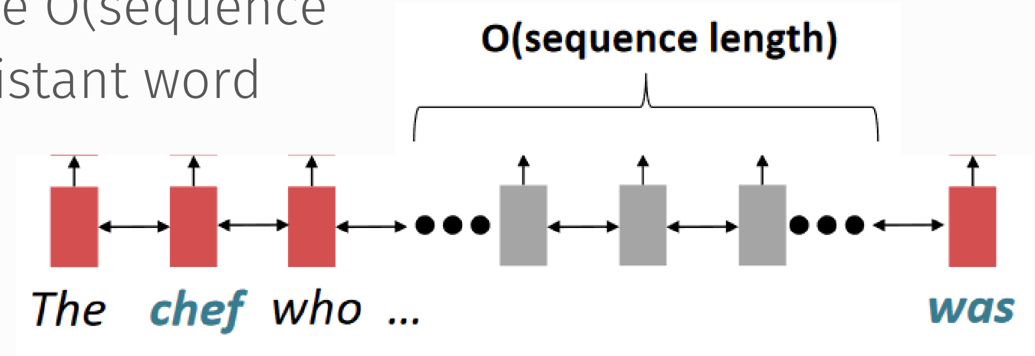
Slide adapted from David Mortensen

Issues with recurrent models: Linear interaction distance

- RNNs are unrolled “left-to-right”.
  - This encodes linear locality: a useful heuristic!
  - Nearby words often affect each other’s meanings



- **Problem**: RNNs take O(sequence length) steps for distant word pairs to interact

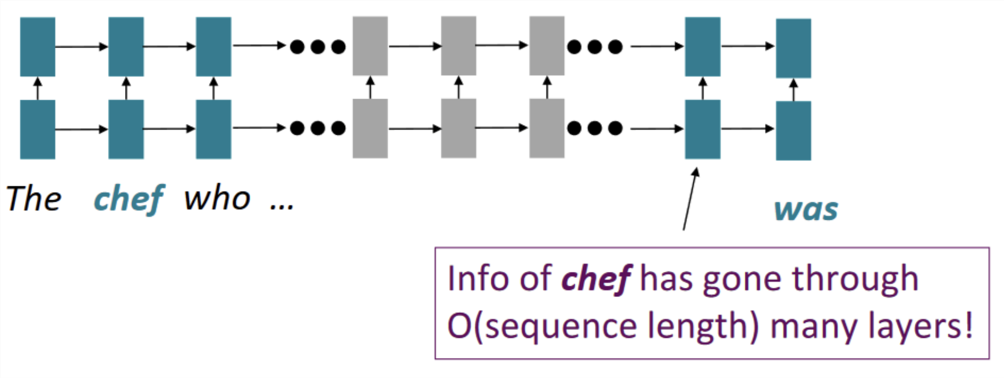


Slide adapted from John Hewitt

Issues with recurrent models: Linear interaction distance

O(sequence length) steps for distant word pairs to interact means:

- Hard to learn long-distance dependencies (because gradient problems!)
- Linear order of words is “baked in”; we already know linear order isn’t the right way to think about sentences…



Slide adapted from John Hewitt

Issues with recurrent models: Lack of parallelizability

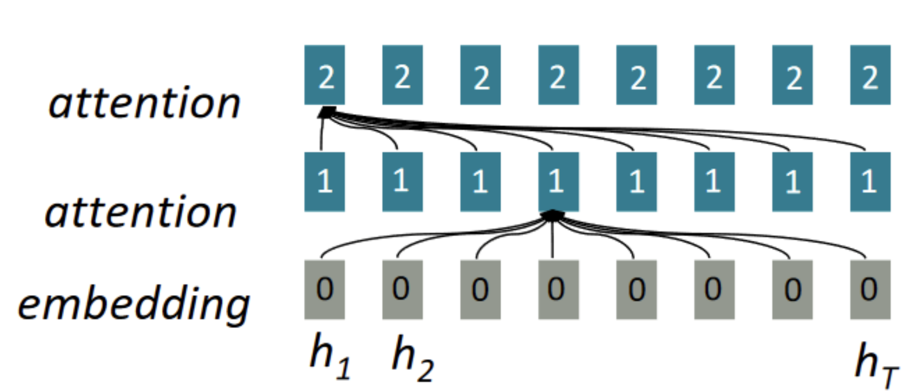
Forward and backward passes have O(sequence length) unparallelizable operations

- GPUs can perform a bunch of independent computations at once!
- But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
- Inhibits training on very large datasets!

Slide adapted from John Hewitt

If not recurrence, then what? How about attention?

- Attention treats each word's representation as a query to access and incorporate information from a set of values.
- We saw attention from the decoder to the encoder; today we'll think about attention within a single sentence (self-attention)
- Number of unparallelizable operations does not increase with sequence length.
- Maximum interaction distance: O(1), since all words interact at every layer!

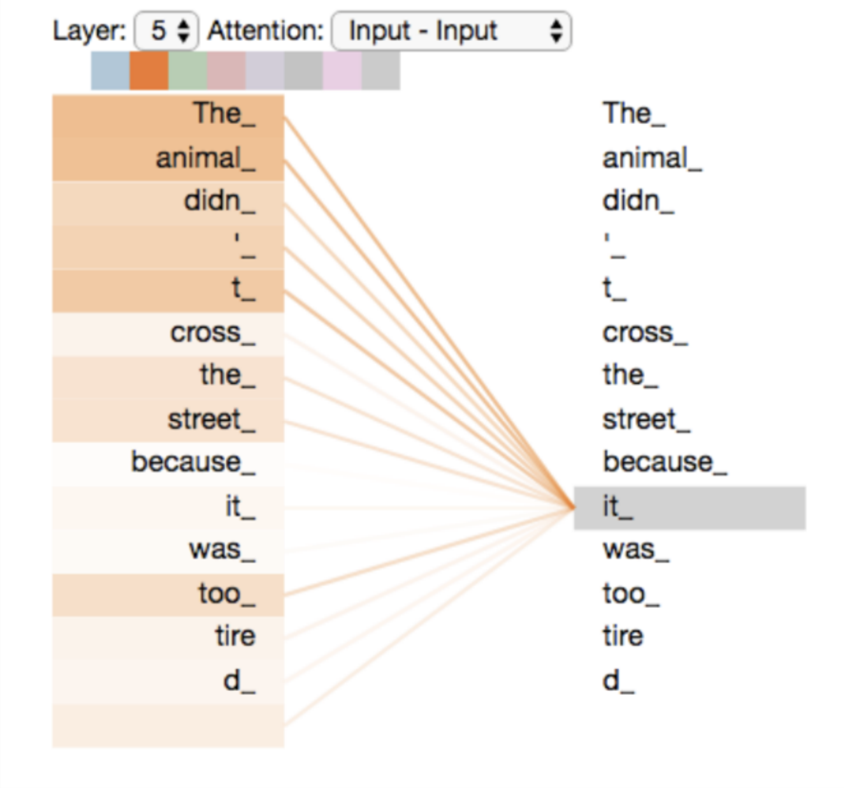


All words attend to all words in previous layer; most arrows here are omitted

Numbers indicate min # of steps before a state can be computed

Slide adapted from John Hewitt

Self-attention: all you need



Take the sentence: "The animal didn't cross the street because it was too tired". What is the antecedent of *it*?

Self-attention allows the model to "attend" to all of the other positions and to process each position (including *the* and *animal*) to help it better encode the pronoun *it*.

You can compare this to the hidden state in an RNN—it conveys information about other words in the sequence to the position one is currently processing.

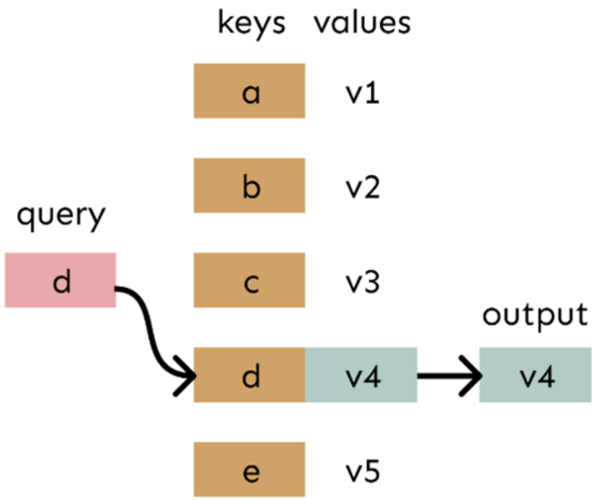
Transformers rely on self-attention.

Slide credit: David Mortensen

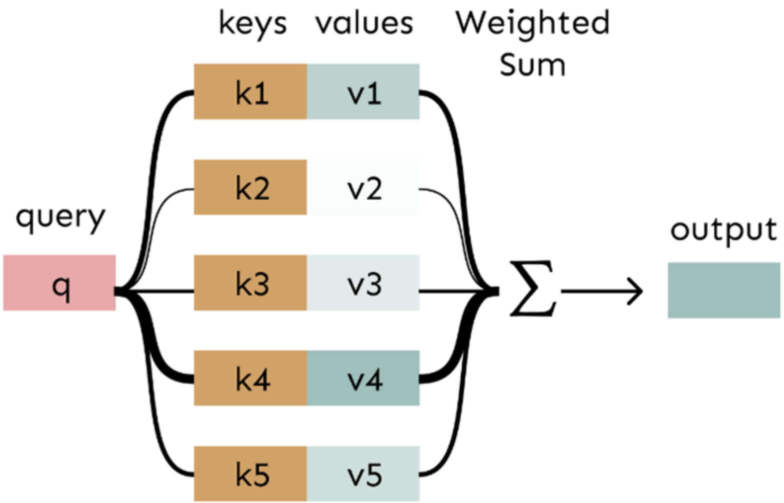
Attention as a soft, averaging lookup table

We can think of **attention** as performing fuzzy lookup in a key-value store.

In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



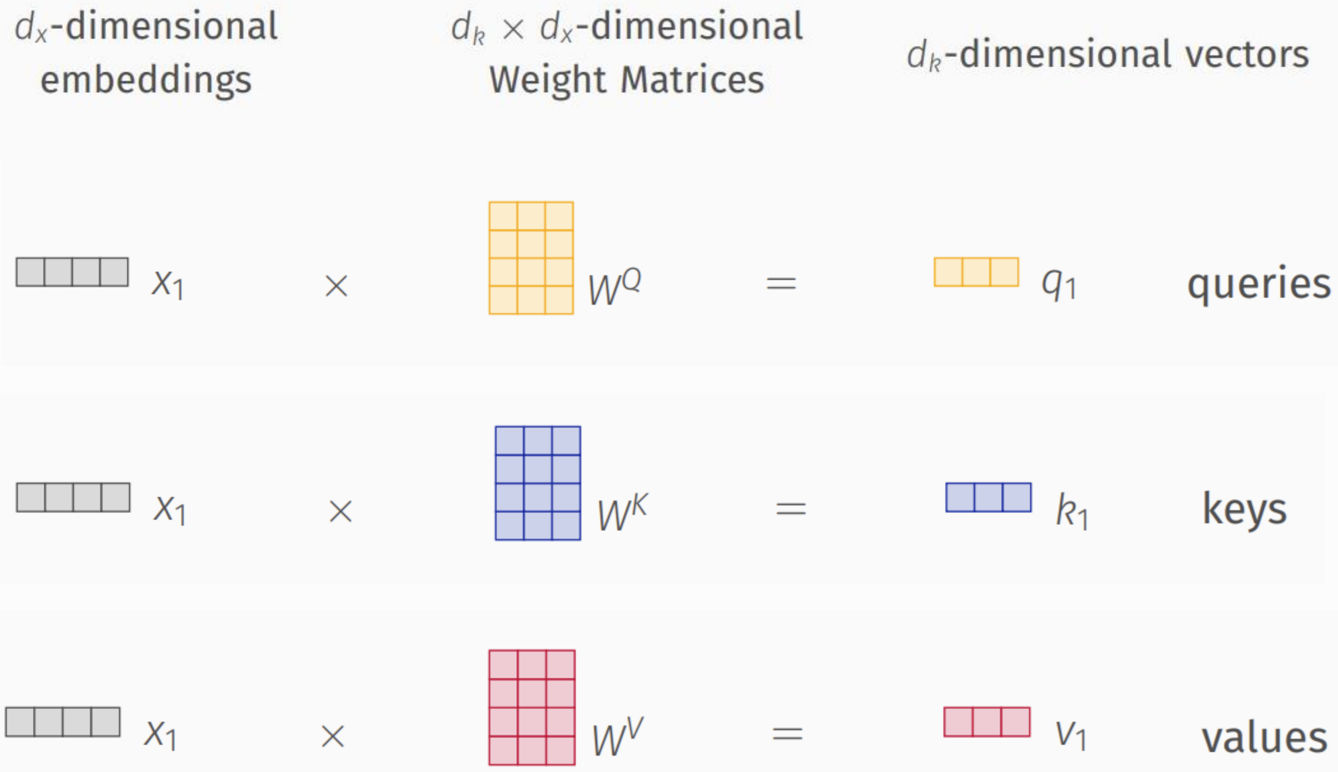
In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



9

Slide credit: John Hewitt

Computing Self-Attention, Step One: Compute Key, Query, and Value Vectors



Slide credit: David Mortensen

Computing Self-Attention, Step Two: Weighted Sum of Value Vectors

	We	wash	our	cats
	x_1	x_2	x_3	x_4
query-key dot product	$q_1 \cdot k_1 = 13$	$q_1 \cdot k_2 = 24$	$q_1 \cdot k_3 = 20$	$q_1 \cdot k_4 = 12$
divide by $\sqrt{d_k}$	$\frac{13}{\sqrt{64}} = 1.63$	$\frac{24}{\sqrt{64}} = 3.0$	$\frac{20}{\sqrt{64}} = 2.5$	$\frac{12}{\sqrt{64}} = 1.5$
softmax	0.12	0.48	0.29	0.10
$\times$ value	$0.12 \times v_1$	$0.48 \times v_2$	$0.29 \times v_3$	$0.10 \times v_4$
sum	z_1			

Barriers and solutions for self-attention as a building block

Barriers		Solutions
• Doesn't have an inherent notion of order!	→	

Fixing the first self-attention problem: **sequence order**

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each sequence index as a vector

$$\boldsymbol{p}_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\dots,n\} \text{ are position vectors}$$

- Don't worry about what the $\boldsymbol{p}_i$ are made of yet!
- Easy to incorporate this info into our self-attention block: just add the $\boldsymbol{p}_i$ to our inputs!
- Recall that $\boldsymbol{x}_i$ is the embedding of the word at index i. The positioned embedding is:

$$\widetilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \boldsymbol{p}_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add...

Slide credit: John Hewitt

Position embeddings learned from scratch

- Learned absolute position representations: Let all $\boldsymbol{p}_i$ be learnable parameters!
- Learn a matrix $\boldsymbol{p} \in \mathbb{R}^{d \times n}$, and let each $\boldsymbol{p}_i$ be a column of that matrix!
- Pros:
  - Flexibility: each position gets to be learned to fit the data
- Cons:
  - Definitely can't extrapolate to indices outside $1, \ldots, n$.
- Most systems use this!
- Sometimes people try more flexible representations of position:
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

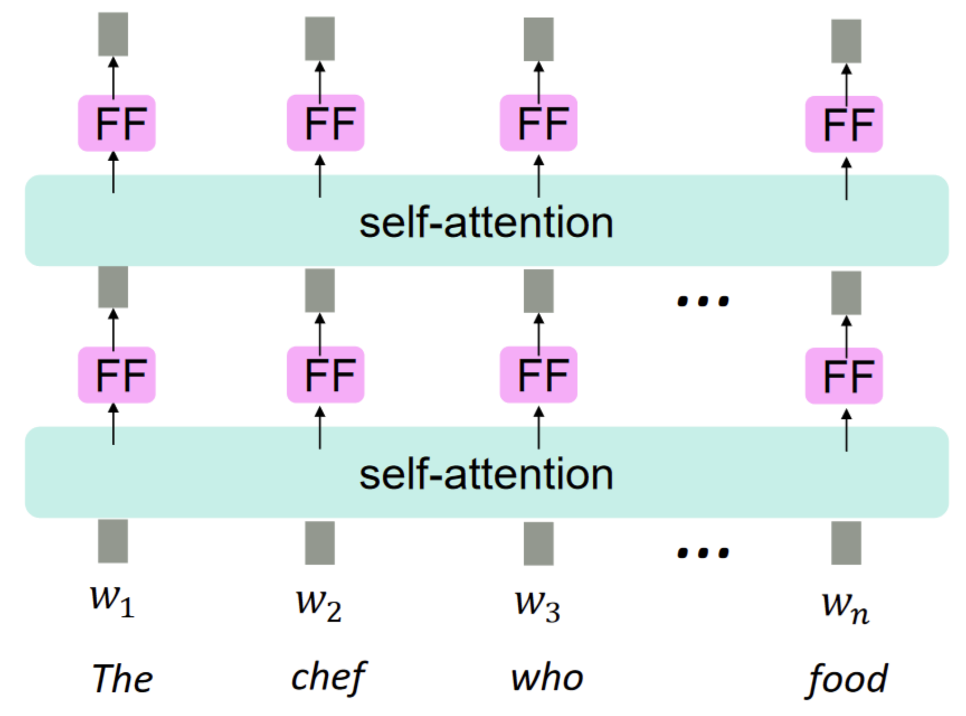
Slide adapted from John Hewitt

Barriers and solutions for self-attention as a building block

Barriers		**Solutions**
• Doesn't have an inherent notion of order!	→	• Add position representations to the inputs
• No nonlinearities for deep learning! It's all just weighted averages	→	

Slide credit: John Hewitt

Solution: add some feedforward NNs!



Intuition: the FF network processes the result of attention

Slide credit: John Hewitt

Barriers and solutions for self-attention as a building block

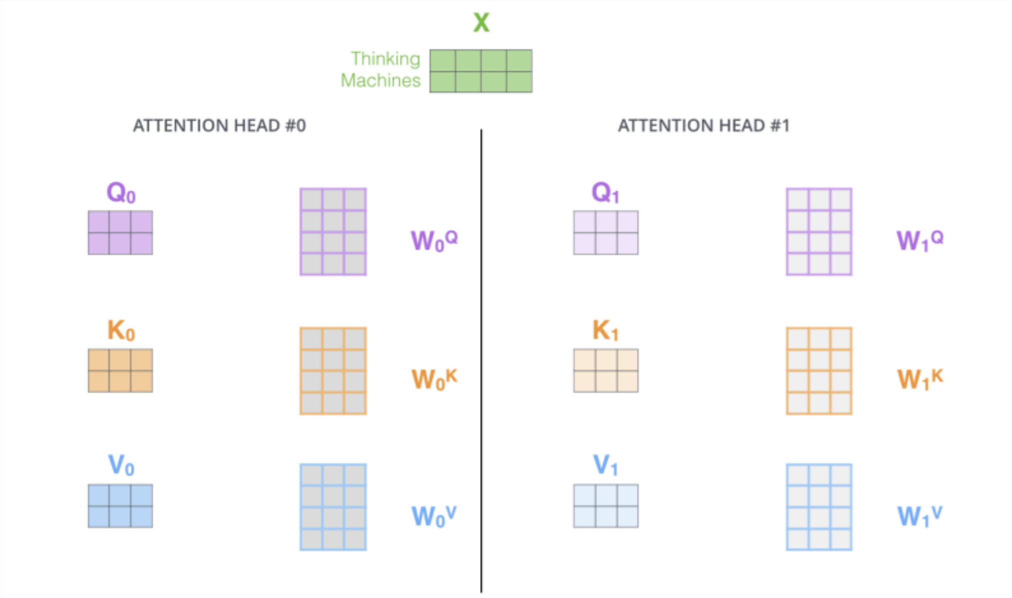
Barriers		Solutions
• Doesn't have an inherent notion of order!	→	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	→	• Easy fix: apply the same feedforward network to each self-attention output.

Slide credit: John Hewitt

Multi-headed attention

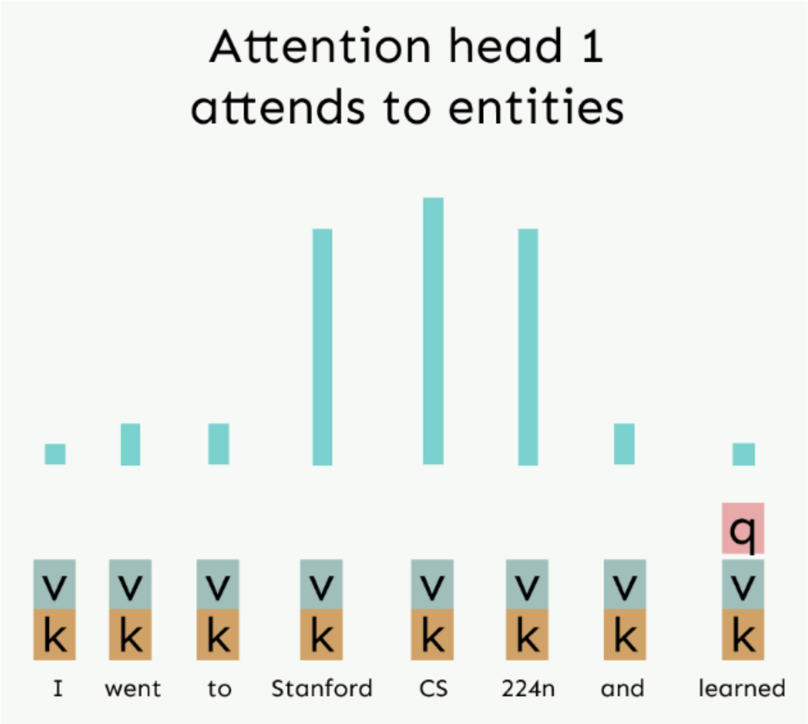
Multi-Headed Attention Expands Transformer Models' Ability to Focus on Different Positions

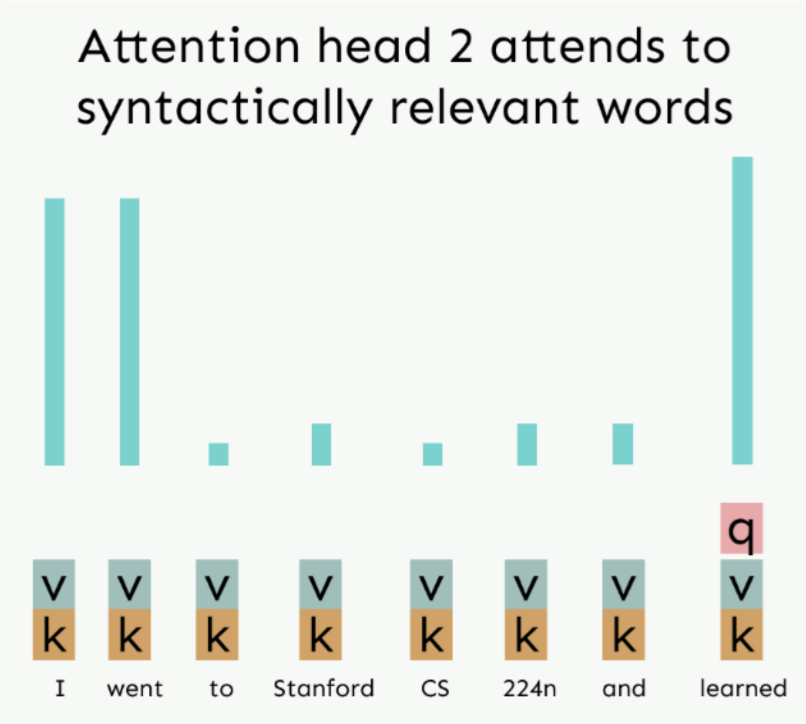
Maintain distinct weight matrices for each attention head—distinct representational subspaces:



Slide credit: David Mortensen

Hypothetical example of multi-headed attention





Slide credit: John Hewitt

Multi-headed attention

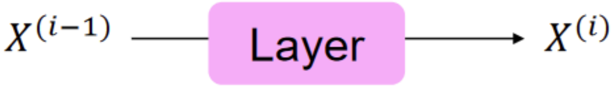
- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x_i^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h.
- Each attention head performs attention independently:
  - $\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = [\text{output}_1; \ldots; \text{output}_h]Y$, where $Y \in \mathbb{R}^{d \times d}$

- Each head gets to "look" at different things, and construct value vectors differently.

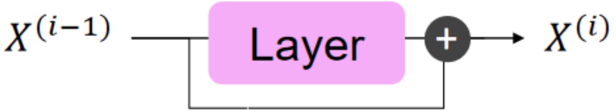
Optimization tricks: residual connections and layer normalization

Residual connections [He et al. 2016]

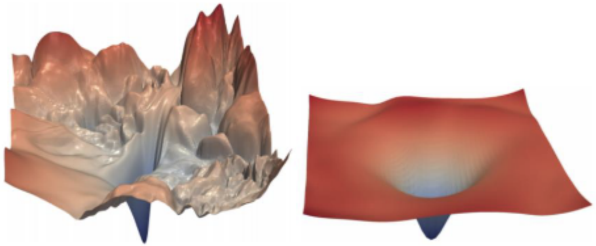
- **Residual connections** are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)



  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)



  - Gradient is **great** through the residual connection; it's 1!
  - Bias towards the identity function!



[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

Slide credit: John Hewitt

Layer normalization [Ba et al. 2016]

- **Layer normalization** is a trick to help models train faster.
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**.
  - LayerNorm's success may be due to its normalizing gradients [Xu et al., 2019]
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.
- Let $\mu = \sum_{j=1}^{d} x_j$; this is the mean; $\mu \in \mathbb{R}$.
- Let $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^{d} \left(x_j - \mu\right)^2}$; this is the standard deviation; $\sigma \in \mathbb{R}$.
- Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)
- Then layer normalization computes:

$$\text{output} = \frac{x - \mu}{\sqrt{\sigma} + \epsilon} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

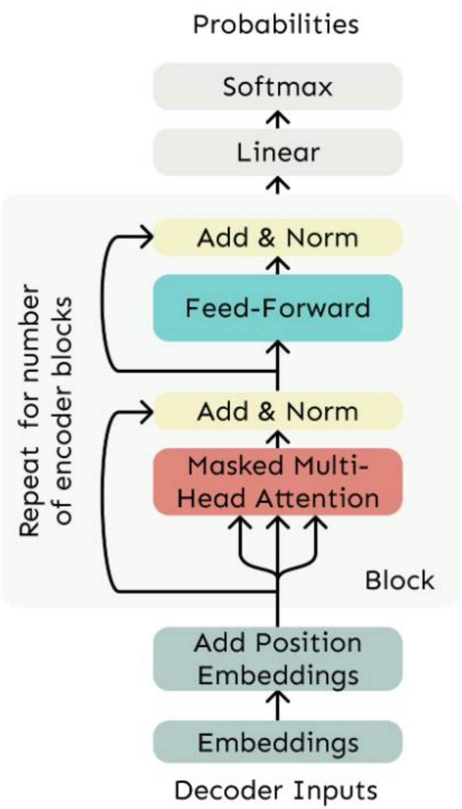
Slide credit: John Hewitt

Transformer blocks

The transformer decoder

The Transformer Decoder is a stack of Transformer Decoder Blocks.

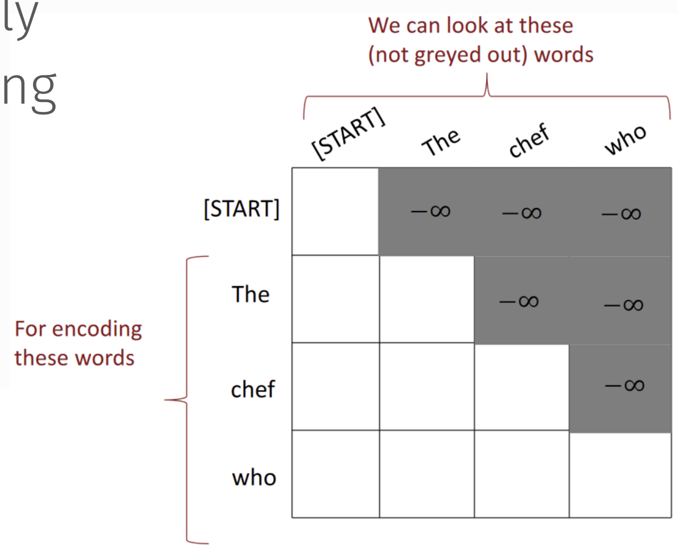
- Each Block consists of:
- Self-attention
- Add & Norm
- Feed-Forward
- Add & Norm
- But for decoding (language modeling), we can't look into the future!



Slide adapted from John Hewitt

Decoding: apply a "causal mask" for self-attention

- To do auto-regressive LM, we need to apply a "causal" mask to self-attention, forbidding it from getting future context.
- At timestep t, we set a_i = 0 for $i > t$



Slide adapted from Tianxing He, John Hewitt

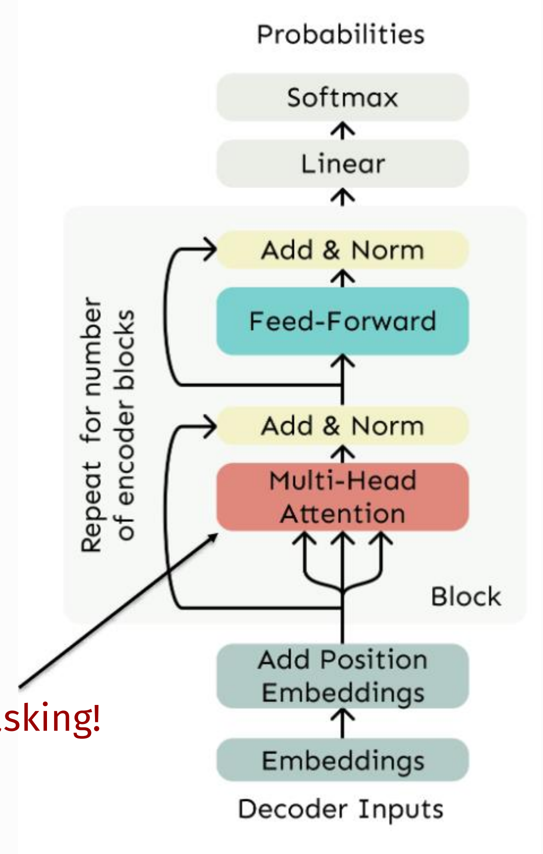
The transformer encoder

The Transformer Decoder constrains to unidirectional context, as for language models.

- What if we want bidirectional context, as for text classification?

- This is the Transformer Encoder. The only difference is that we remove the masking in the self-attention.

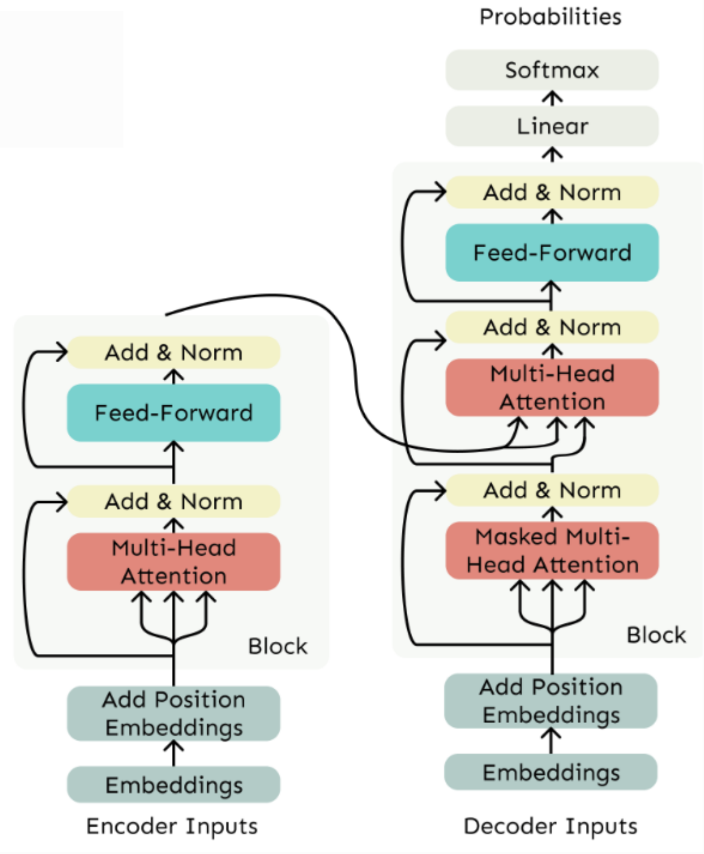
no masking!



Slide adapted from John Hewitt

The transformer encoder-decoder

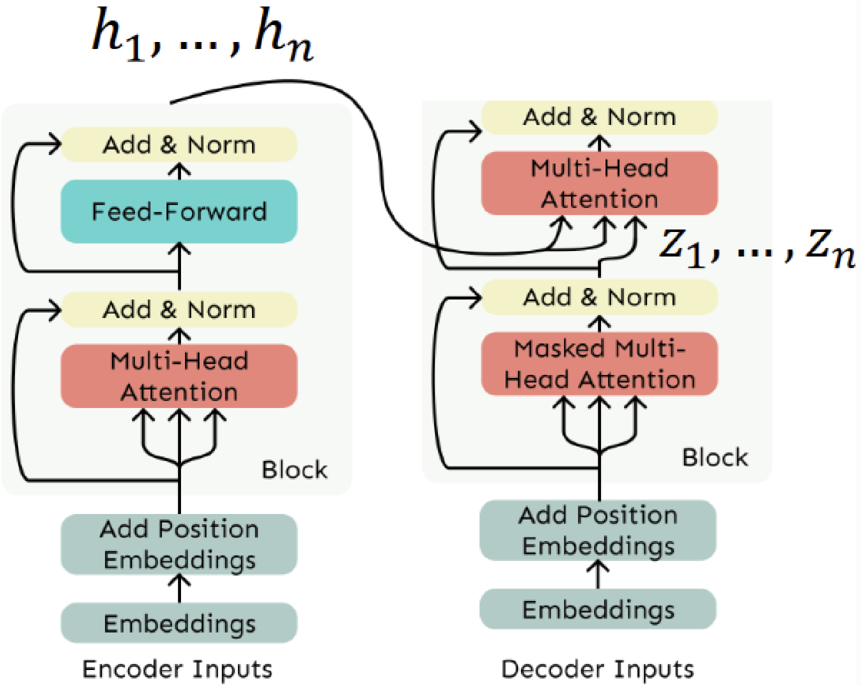
- Can use transformers for encoder-decoder (seq2seq) framework
- Transformer decoder modified to perform cross-attention to the output of the encoder



Slide adapted from John Hewitt

Cross-attention

- We saw that self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let $h_1, \dots, h_n$ be **output** vectors **from** the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \dots, z_n$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = Kh_i, v_i = Vh_i$.
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.



Slide adapted from John Hewitt

Drawbacks of transformers

- Quadratic compute in self-attention (today):
  - Computing all pairs of interactions means our computation grows quadratically with the sequence length!
  - For recurrent models, it only grew linearly!
- Can’t easily handle long sequences; usually set a bound of 512 tokens
- Position representations:
  - Are simple absolute indices the best we can do to represent position?
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

Slide adapted from John Hewitt

Wrapping up

- Transformers are a high-performing NLP architecture based on self-attention
- Transformers can be used for language modeling

Questions?