

04. Transformers

Transformers may not fix all your NLP problems.

-
-
-

But they are worth some attention.



CS 2731 Introduction to Natural Language Processing

Session 13: Transformers part 1, beam search

Michael Miller Yoder

October 11, 2023

Course logistics

- Proposal and literature review is **due tomorrow, Thu 10-12, 11:59pm**
 - Instructions are on the [project webpage](#)
 - Submit on Canvas
 - One submission per group
- Organize at least 4 papers into themes of approaches, datasets, findings
- Discussion post instead of reading quiz for Monday (TBD)

Lecture overview: Transformers part 1, beam search

- Self-attention
- Multi-headed attention
- Residual connections and layer normalization
- Transformer blocks
- Beam search
- GPT preview
- Class time to fill out midterm OMETs



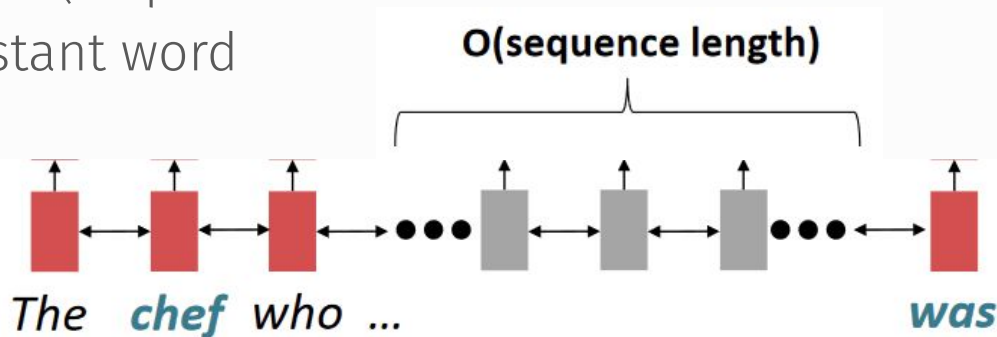
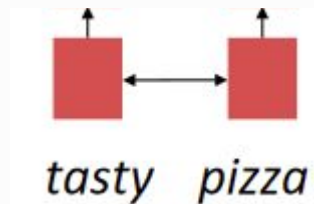
From recurrence to self-attention

Transformers improved on RNNs and CNNs

- Google introduced Transformers in 2017 [Vaswani et al. 2017, “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)

Issues with recurrent models: Linear interaction distance

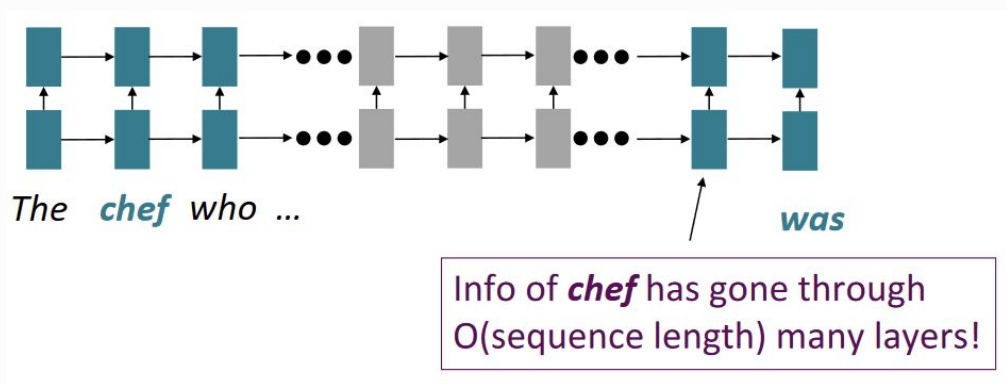
- RNNs are unrolled “left-to-right”.
 - This encodes linear locality: a useful heuristic!
 - Nearby words often affect each other’s meanings
- **Problem:** RNNs take $O(\text{sequence length})$ steps for distant word pairs to interact



Issues with recurrent models: Linear interaction distance

$O(\text{sequence length})$ 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...



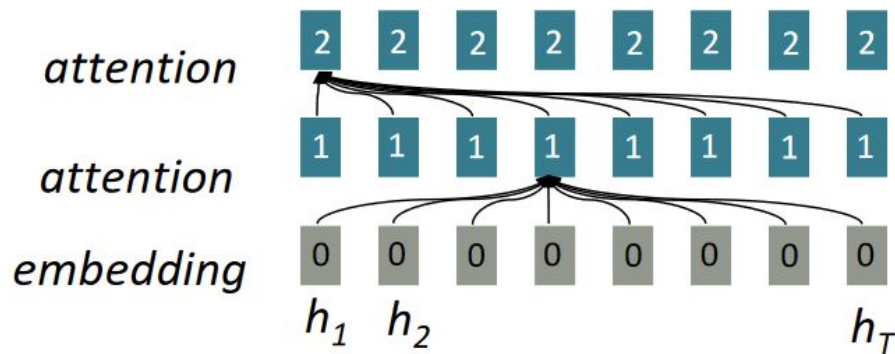
Issues with recurrent models: Lack of parallelizability

Forward and backward passes have $O(\text{sequence length})$ 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!

If not recurrence, then what? How about attention?

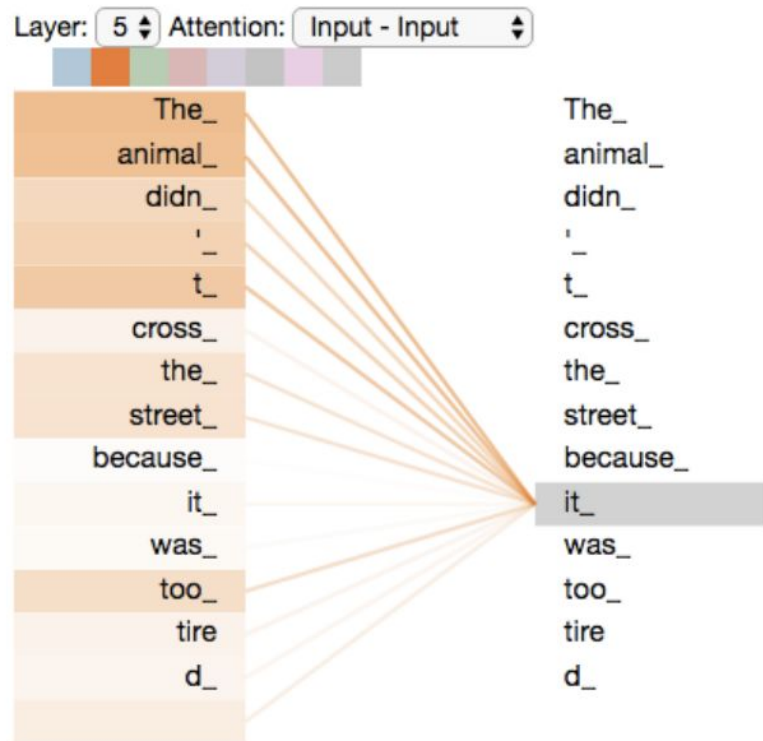
- Attention treats each word's representation as a query to access and incorporate information from a set of values.
- We saw attention from the decoder to the encoder; today we'll think about attention within a single sentence (self-attention)
- Number of unparallelizable operations does not increase 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

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*.

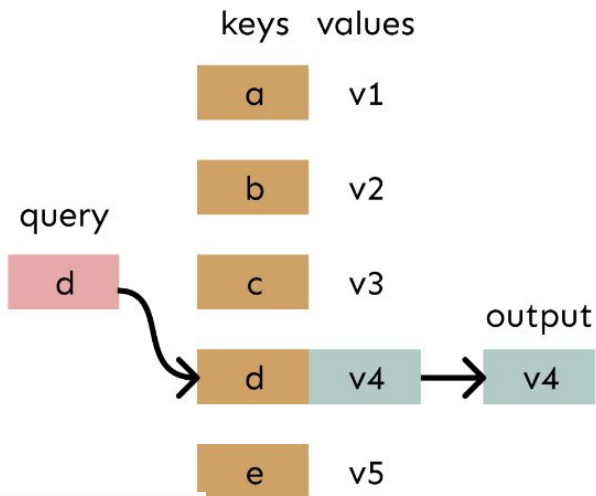
You can compare this to the hidden state in an RNN—it conveys information about other words in the sequence to the position one is currently processing.

Transformers rely on self-attention.

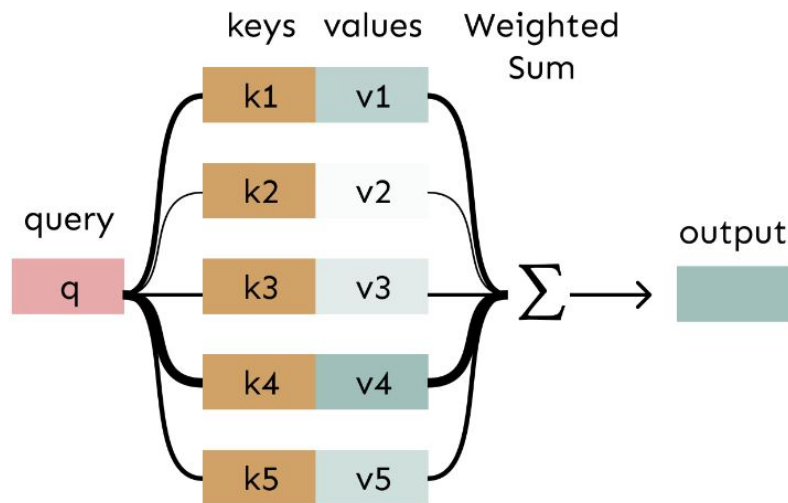
Attention as a soft, averaging lookup table

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.



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


d_x -dimensional
embeddings

$d_k \times d_x$ -dimensional
Weight Matrices

d_k -dimensional vectors


$$\begin{bmatrix} \square & \square & \square & \square \end{bmatrix} x_1 \times \begin{bmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix} W^K = \begin{bmatrix} \square & \square & \square \end{bmatrix} k_1 \quad \text{keys}$$


$$\begin{bmatrix} \square & \square & \square & \square \end{bmatrix} x_1 \times \begin{bmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix} W^Q = \begin{bmatrix} \square & \square & \square \end{bmatrix} q_1 \quad \text{queries}$$


$$\begin{bmatrix} \square & \square & \square & \square \end{bmatrix} x_1 \times \begin{bmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix} W^V = \begin{bmatrix} \square & \square & \square \end{bmatrix} v_1 \quad \text{values}$$

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



Barriers and solutions for self-attention as a building block

Barriers

- Doesn't have an inherent notion of order!



Solutions

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

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

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

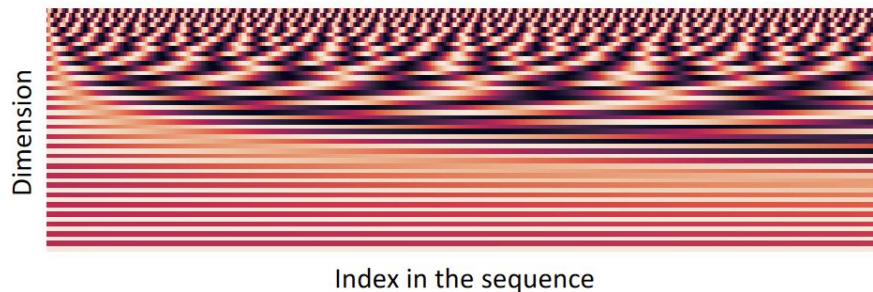
$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{p}_i$$

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

Position embeddings through sinusoids

- Sinusoidal position representations: concatenate sinusoidal functions of varying periods:

$$\mathbf{p}_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*d/2/d}) \\ \cos(i/10000^{2*d/2/d}) \end{pmatrix}$$



- Pros:
 - Periodicity indicates that maybe “absolute position” isn’t as important
 - Maybe can extrapolate to longer sequences as periods restart!
- Cons:
 - Not learnable; also the extrapolation doesn’t really work!

Position embeddings learned from scratch

- Learned absolute position representations: Let all p_i be learnable parameters!
- Learn a matrix $\mathbf{p} \in \mathbb{R}^{d \times n}$, and let each \mathbf{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 $1, \dots, n$.
- 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]

Barriers and solutions for self-attention as a building block

Barriers

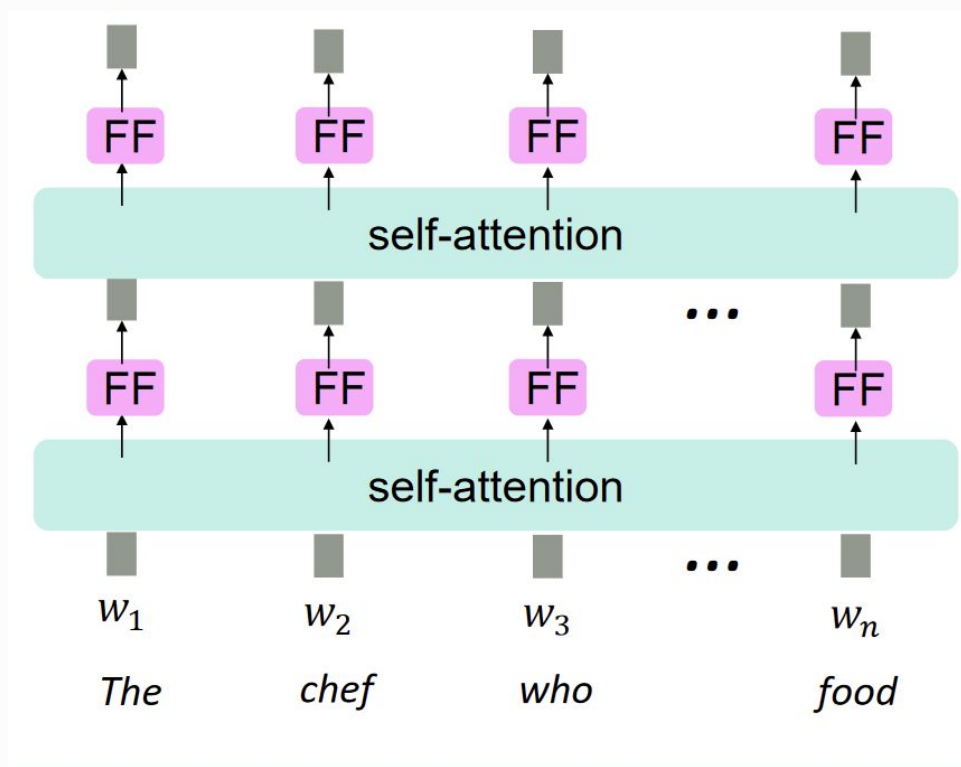
- Doesn't have an inherent notion of order!
- No nonlinearities for deep learning! It's all just weighted averages



Solutions

- Add position representations to the inputs

Solution: add some feedforward NNs!

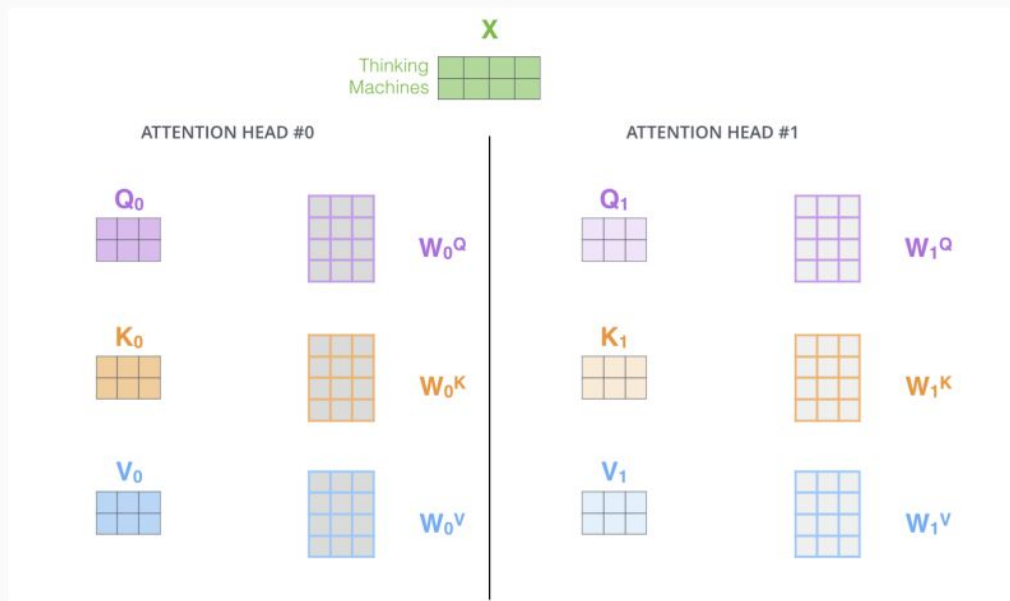


Intuition: the FF network processes the result of attention

Multi-headed attention

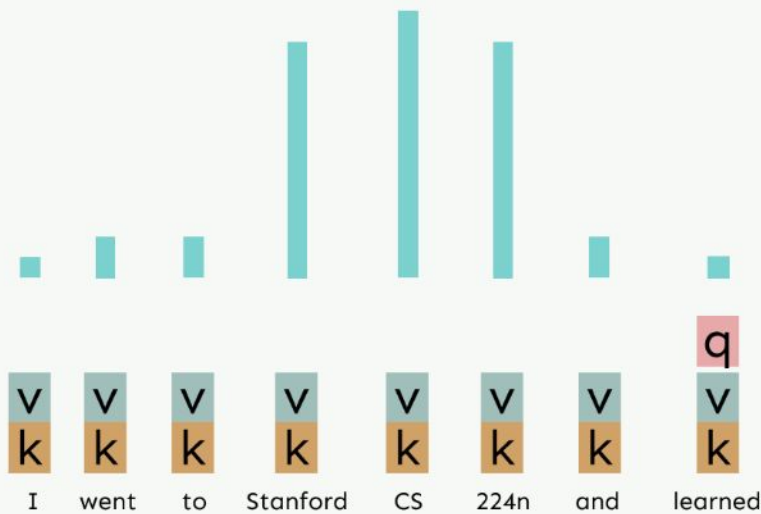
Multi-Headed Attention Expands Transformer Models' Ability to Focus on Different Positions

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

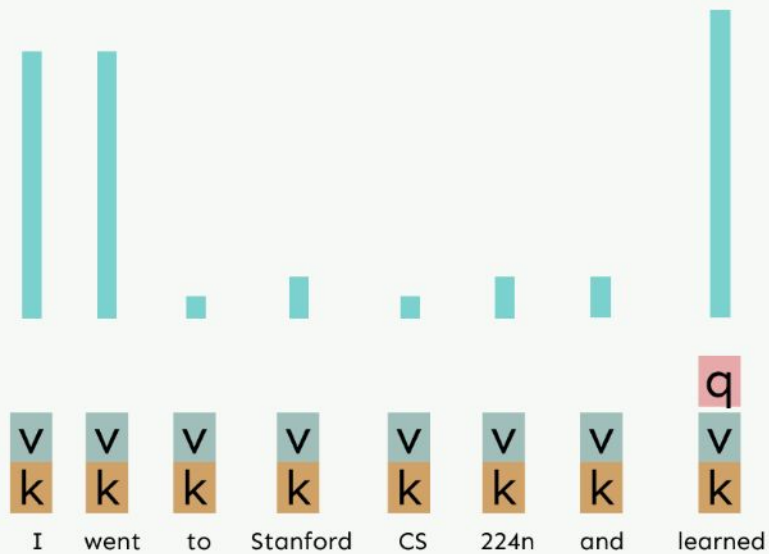


Hypothetical example of multi-headed attention

Attention head 1
attends to entities



Attention head 2 attends to
syntactically relevant words



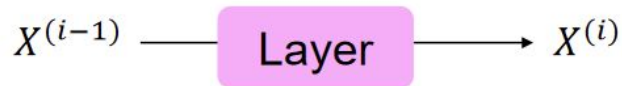
Multi-headed attention

- 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:
 - $\text{output}_\ell = \text{softmax}(X Q_\ell K_\ell^\top X^\top) * X V_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
 - $\text{output} = [\text{output}_1; \dots; \text{output}_h] Y$, 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]

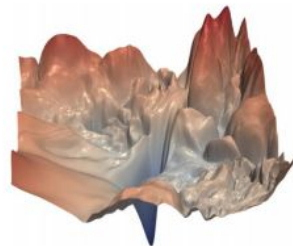
- **Residual connections** are a trick to help models train better.
 - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)



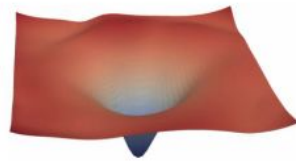
- We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn “the residual” from the previous layer)



- 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]

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 $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^d (x_j - \mu)^2}$; 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

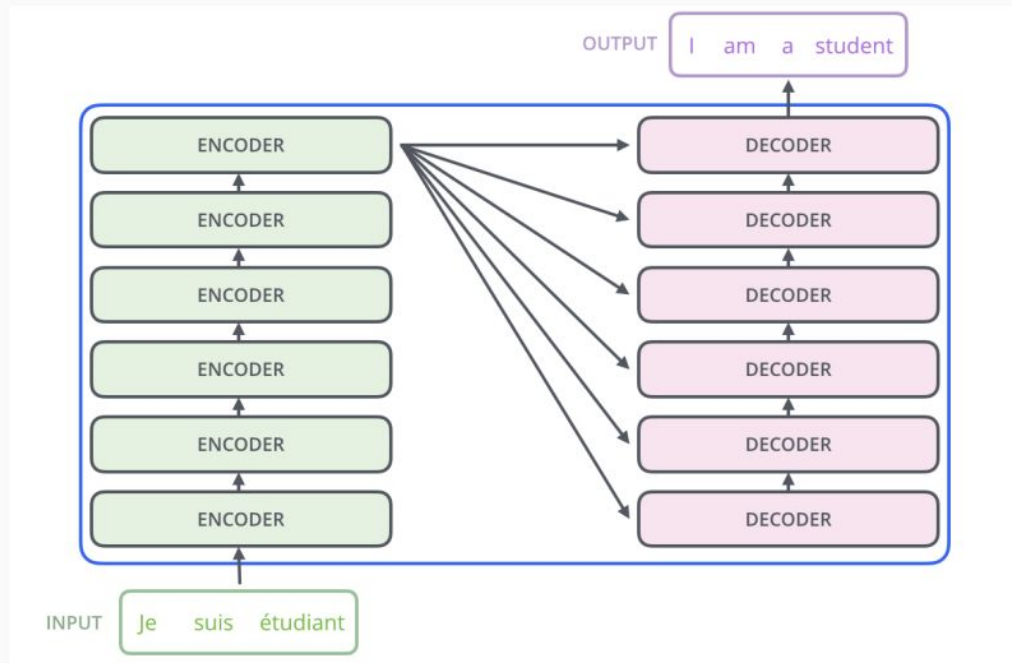
Transformer blocks

Transformers: Stacks of Encoders and Decoders

A transformer is a stack of ~ 6 encoders and decoders. The encoders are identical in structure but do not share weights.

Encoders encode entire input sentences, so can look at future words

Decoders generate output text a step at a time, so can **not** look at future words (language modeling)



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 $i > t$



For encoding
these words

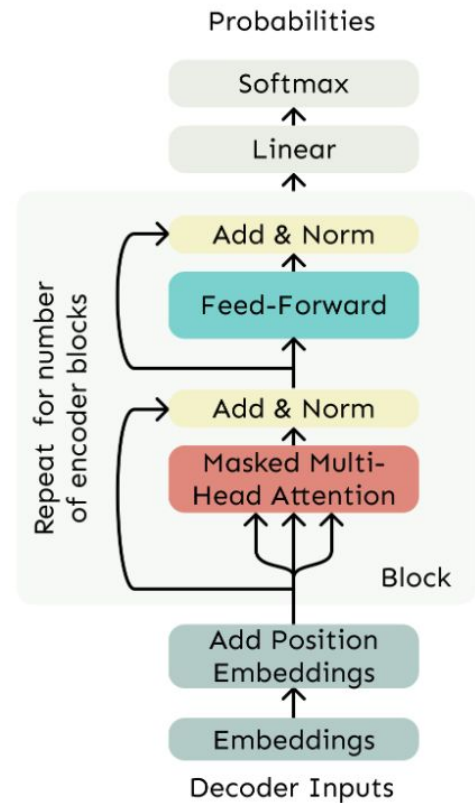
We can look at these
(not greyed out) words

| | [START] | The | chef | who |
|---------|---------|-----------|-----------|-----------|
| [START] | | $-\infty$ | $-\infty$ | $-\infty$ |
| The | | | $-\infty$ | $-\infty$ |
| chef | | | | $-\infty$ |
| who | | | | |

The transformer decoder

The Transformer Decoder is a stack of Transformer Decoder Blocks.

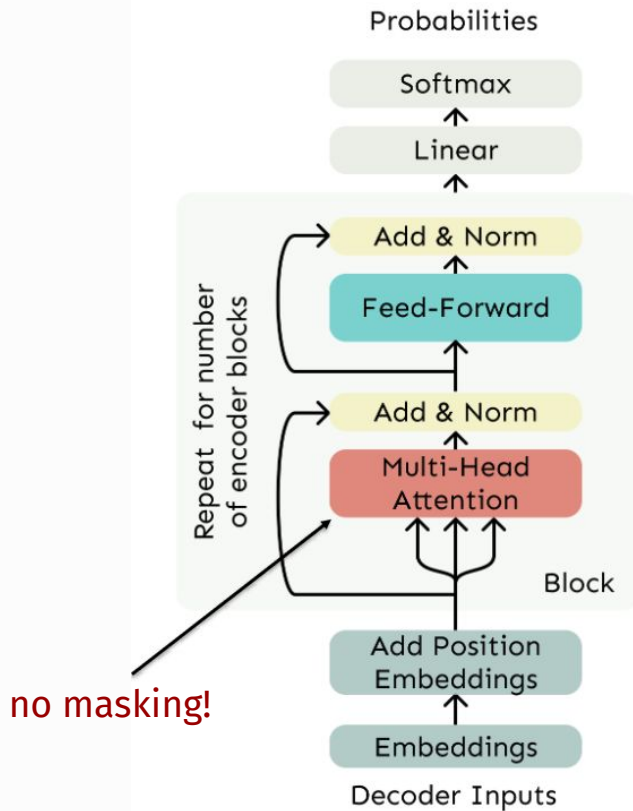
- Each Block consists of:
- Self-attention
- Add & Norm
- Feed-Forward
- Add & Norm
- That's it! We've gone through the Transformer Decoder.



The transformer encoder

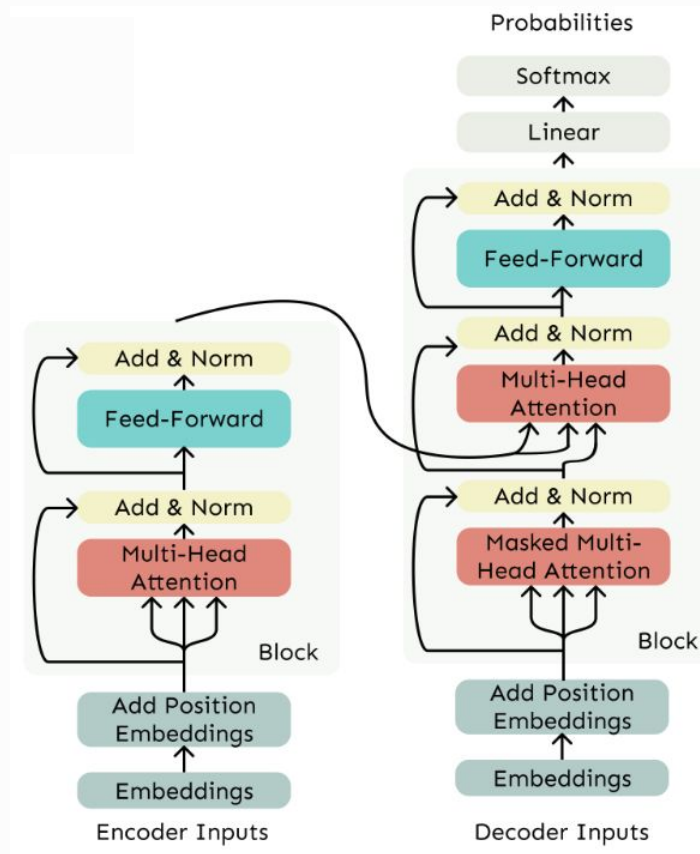
The Transformer Decoder constrains to unidirectional context, as for language models.

- 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.



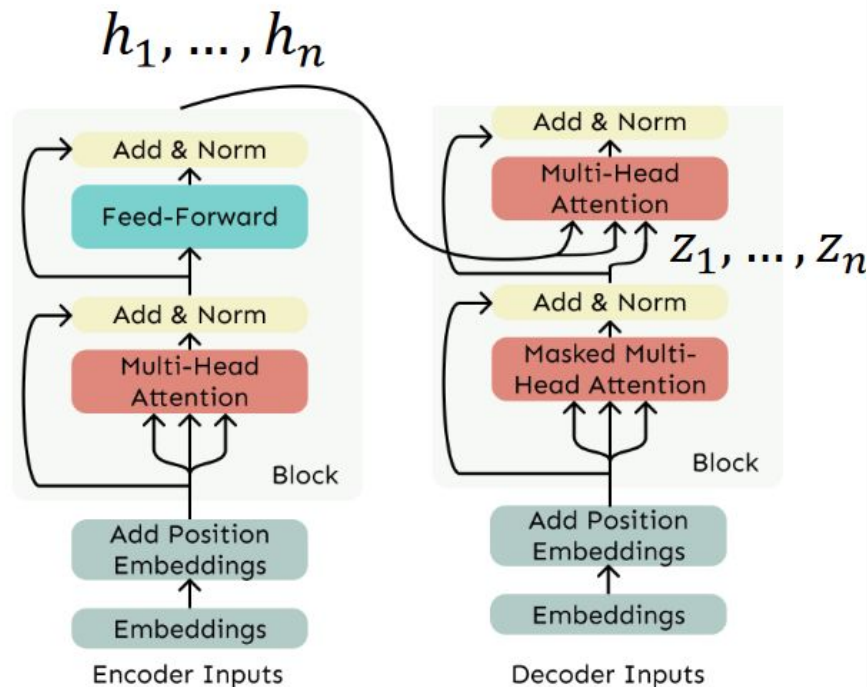
The transformer encoder-decoder

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



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**, $q_i = Qz_i$.



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]

Beam search

Beam search improves on greedy decoding

- Traditional encoder-decoder framework involves generating highest probability word (argmax) at each timestep in the decoding
- But this greedy approach suffers from issues if choosing early high-probability tokens leads to low-probability sequences!
- **Solution:** Don't commit to just the 1 highest probability word, but keep multiple options in a “beam”
- Prune to k highest-probability sequences after each timestep

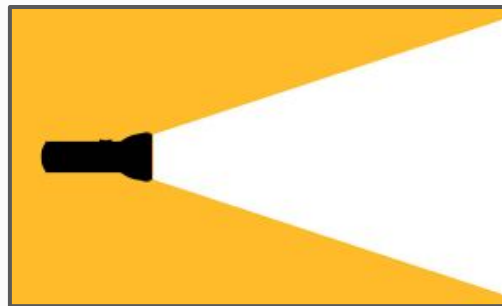
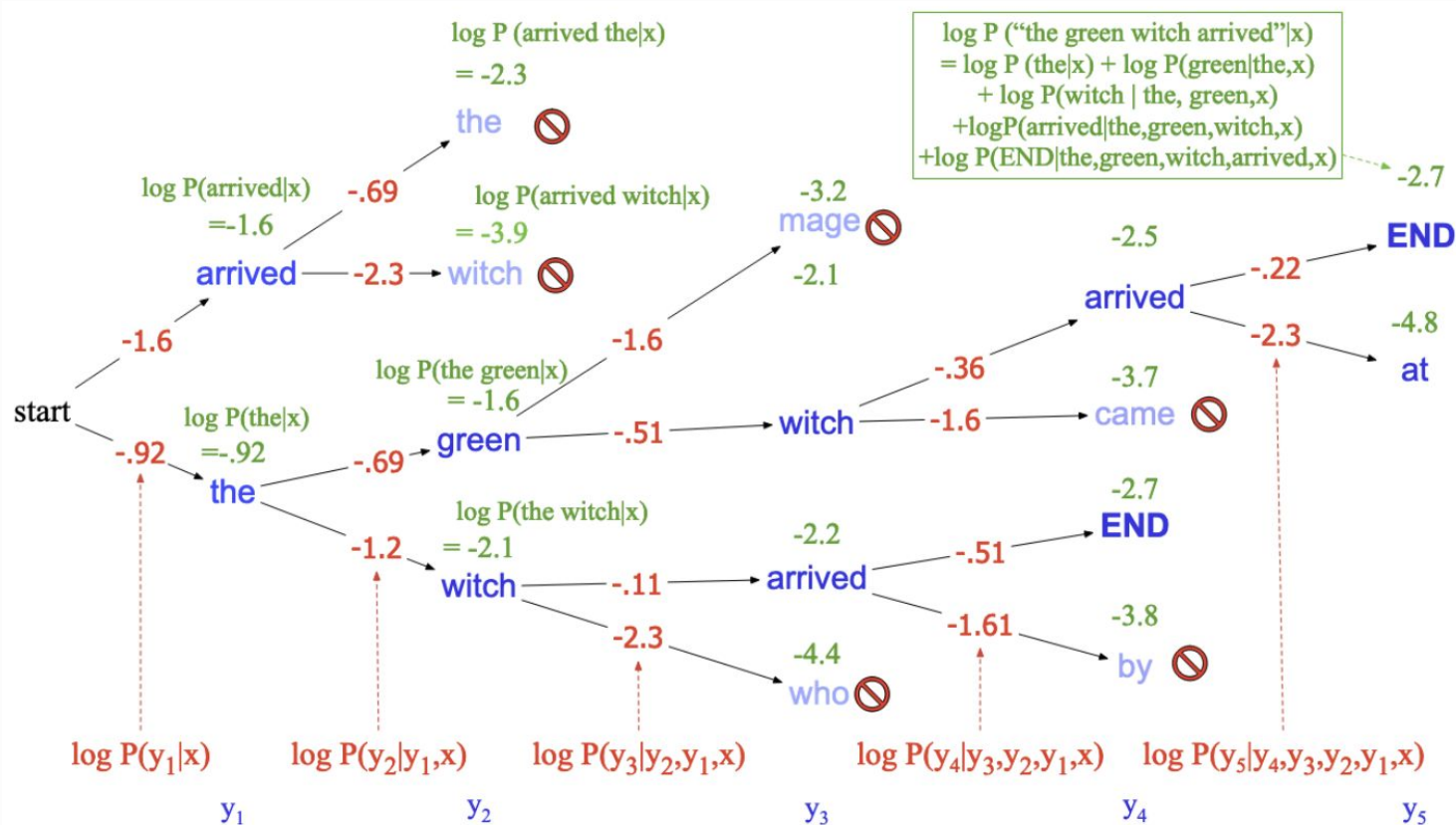


Image: iStock

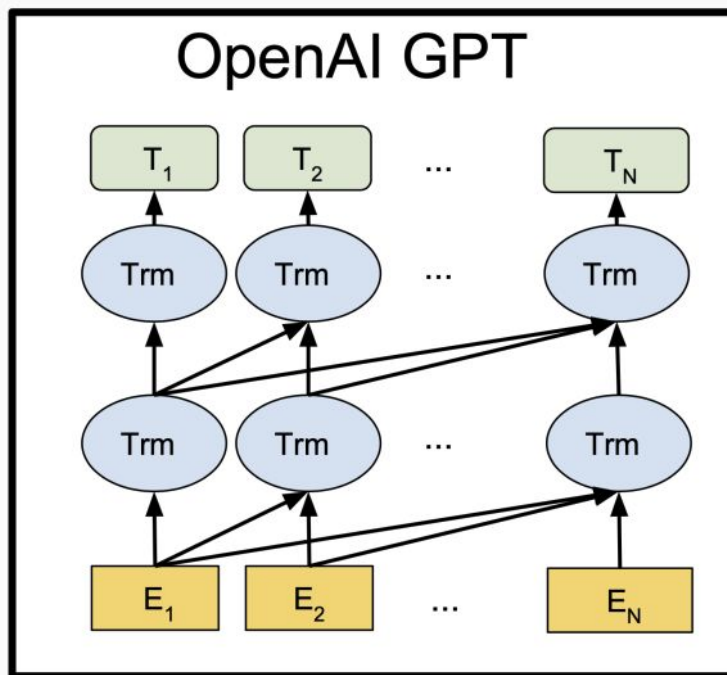
Beam search example



GPT (Generative Pretrained Transformer) preview

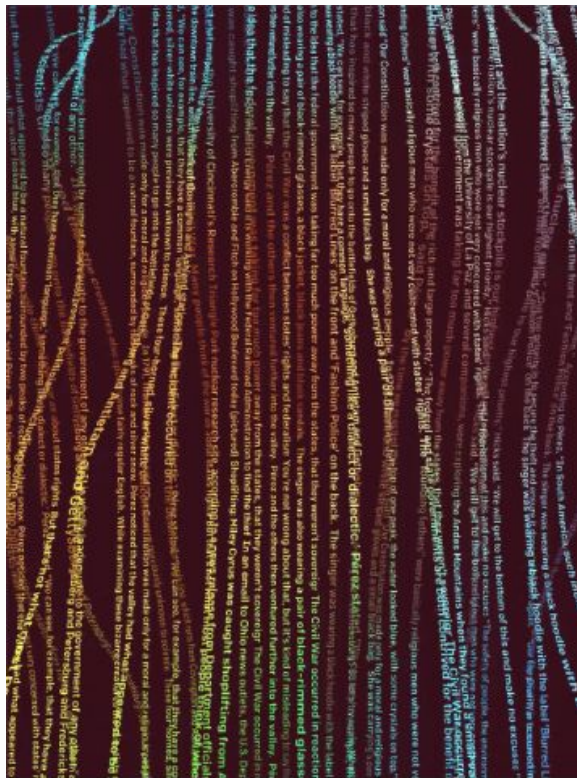
GPT (Generative Pre-Trained Transformer)

The original GPT, released in 2018, combined transformers and unsupervised pre-training (ingredients that would also be central to BERT, GPT-2, and GPT-3). A broad view of its architecture is given below:



The GPT models from OpenAI

- They are basically larger and larger autoregressive transformer LM trained on larger and larger amounts of data
- “Decoder-only” transformers
- They have shown amazing language generation capability when you give it a prompt (aka. prefix, the beginning of a paragraph)



Generation example from the GPT-2 model

SYSTEM PROMPT
(HUMAN-WRITTEN)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

MODEL COMPLETION
(MACHINE-WRITTEN,
10 TRIES)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

A sample from GPT2 (with top-k sampling)

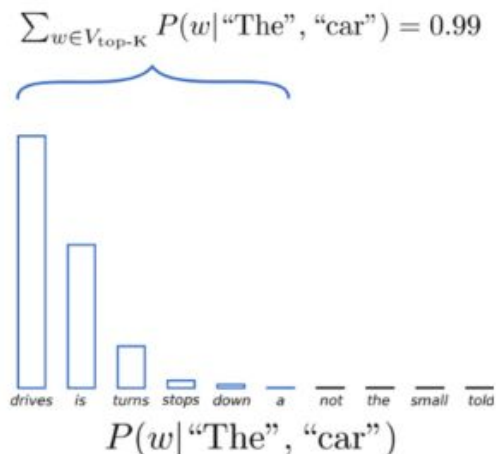
The top-k sampling algorithm

We will represent $P(\cdot | W_{1..i})$ by $p = (p_1, p_2, \dots, p_{|V|})$, where the elements is sorted that $p_1 \geq p_2 \geq p_3 \dots \geq p_{|V|}$.

Top-K sampling transforms p to \hat{p} by:

$$\hat{p}_i = \frac{p_i \cdot 1\{i \leq K\}}{Z}$$

And we sample W_{i+1} from \hat{p} .



Lower k provides higher quality, but less diversity

Wrapping up

- Transformers are a high-performing NLP architecture based on self-attention
- Transformers can be used for language modeling
- Beam search is used to find higher probability sequences than greedy approaches find in decoding

Midterm course evaluation (OMETs)

- Please fill out the midterm course evaluation:
<https://go.blueja.io/XdNK-fTi6kqeUBlVLV4jcQ>
- I welcome all types of feedback (positive and critical)
- **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/as you're comfortable with
- Closes Fri Oct 13, 11:59pm



Questions?