

## 04. Transformers

Transformers may not fix all your NLP problems.

- 
- 
- 

But they are worth some attention.



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/NsyfmLU5Ee4GQiFKcqQ3g>
- 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, LLMs part 1

- Self-attention
- Multi-headed attention
- Residual connections and layer normalization
- Transformer blocks
- Intro to LLMs
- Sampling techniques for decoder-only LLMs



# From recurrence to self-attention

---

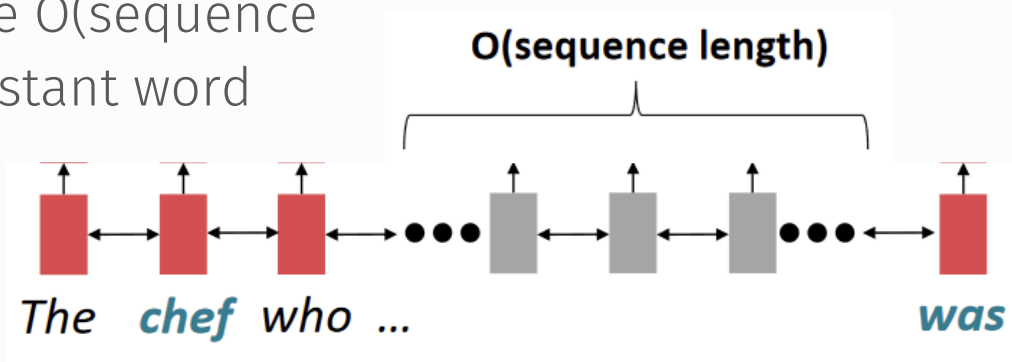
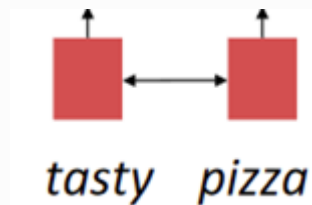
# Transformers improved on RNNs and CNNs

- 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)



# Issues with recurrent models: Linear interaction distance

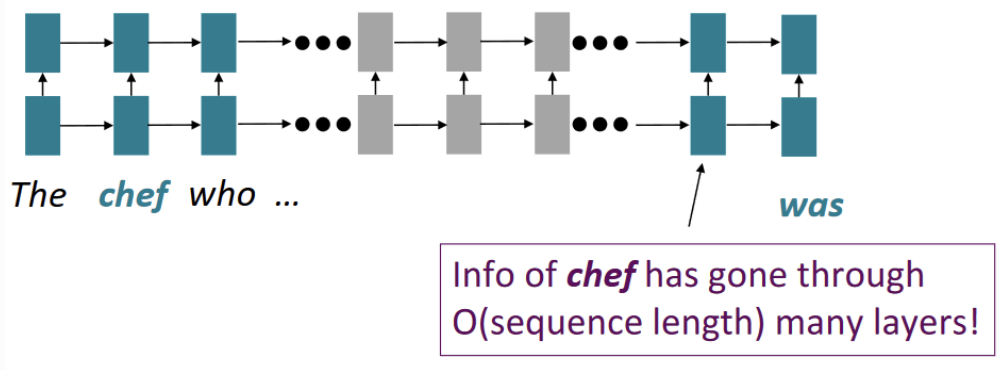
- RNNs are unrolled “left-to-right”.
  - This encodes linear locality: a useful heuristic!
  - Nearby words often affect each other’s meanings
- **Problem:** RNNs take  $O(\text{sequence length})$  steps for distant word pairs to interact



# 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...



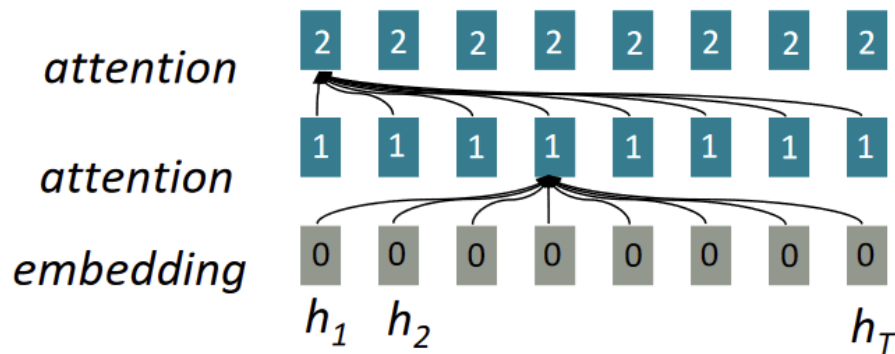
# Issues with recurrent models: Lack of parallelizability

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?

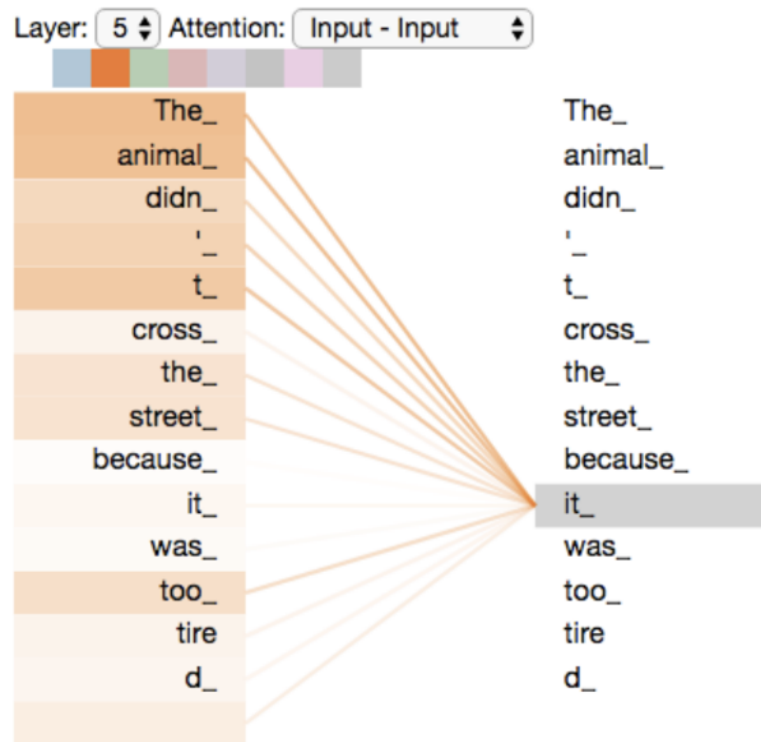
- Attention treats each word's representation as a query to access and incorporate information from a set of values.
- We saw attention from the decoder to the encoder; today we'll think about attention within a single sentence (self-attention)
- Number of unparallelizable operations does not increase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

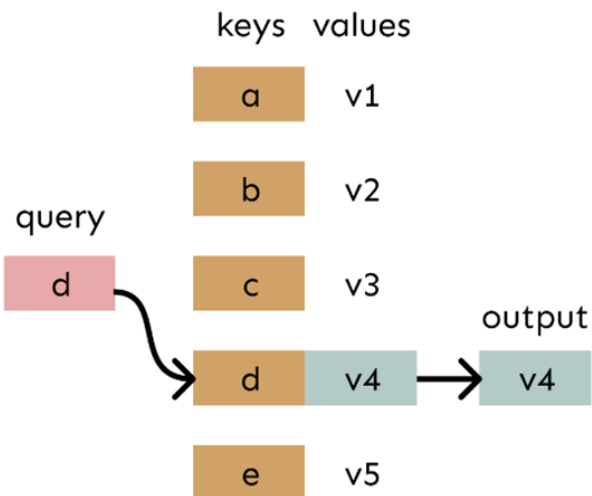
You can compare this to the hidden state in an RNN—it conveys information about other words in the sequence to the position one is currently processing.

Transformers rely on self-attention.

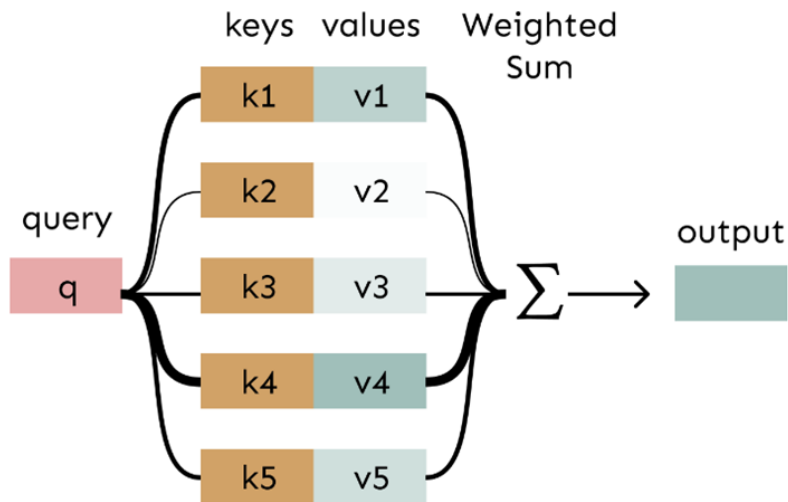
# Attention as a soft, averaging lookup table

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

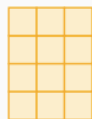
$d_k \times d_x$ -dimensional  
Weight Matrices

$d_k$ -dimensional vectors



$x_1$

$\times$



$W^Q$

$=$



$q_1$

queries



$x_1$

$\times$



$W^K$

$=$



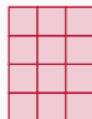
$k_1$

keys



$x_1$

$\times$



$W^V$

$=$



$v_1$

values

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

	We $x_1$	wash $x_2$	our $x_3$	cats $x_4$
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 $\sqrt{d_k}$	$\frac{13}{\sqrt{64}} = 1.63$	$\frac{24}{\sqrt{64}} = 3.0$	$\frac{20}{\sqrt{64}} = 2.5$	$\frac{12}{\sqrt{64}} = 1.5$
------------------------	-------------------------------	------------------------------	------------------------------	------------------------------

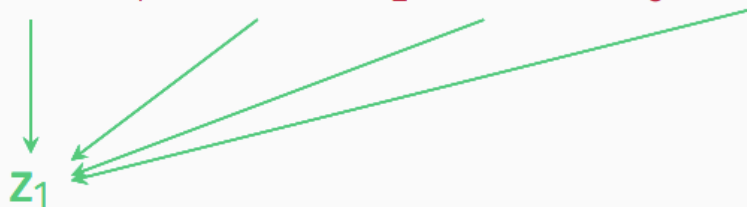
softmax

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 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

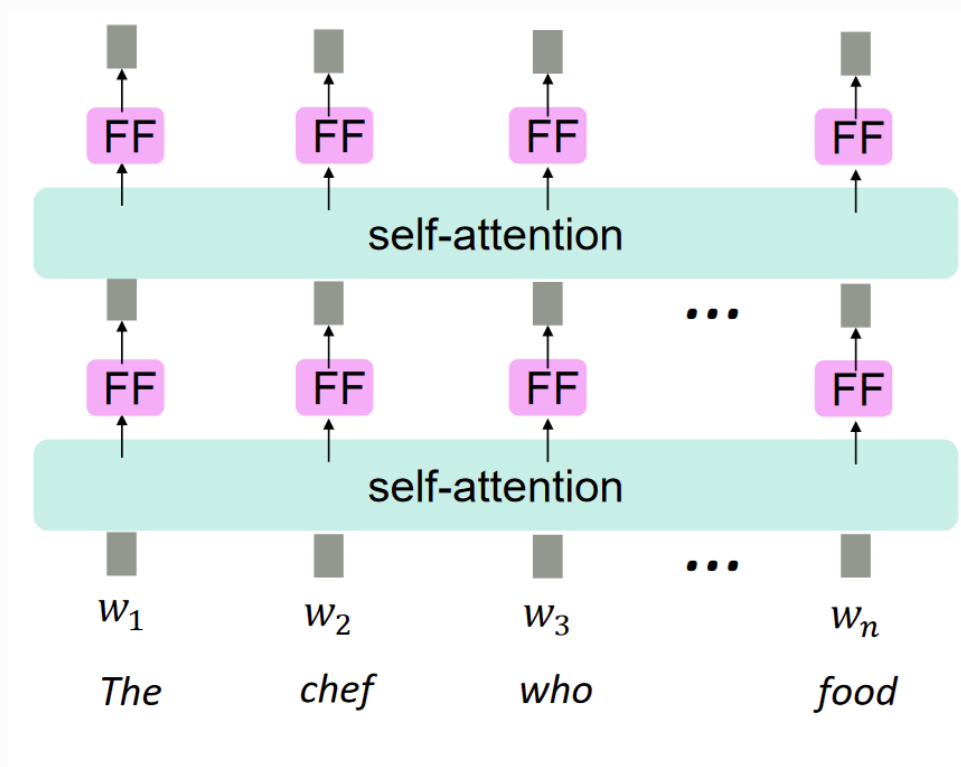
- Doesn't have an inherent notion of order!
- No nonlinearities for deep learning! It's all just weighted averages



## Solutions

- Add position representations to the inputs

# Solution: add some feedforward NNs!



Intuition: the FF network processes the result of attention

# Barriers and solutions for self-attention as a building block

## Barriers

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



## Solutions

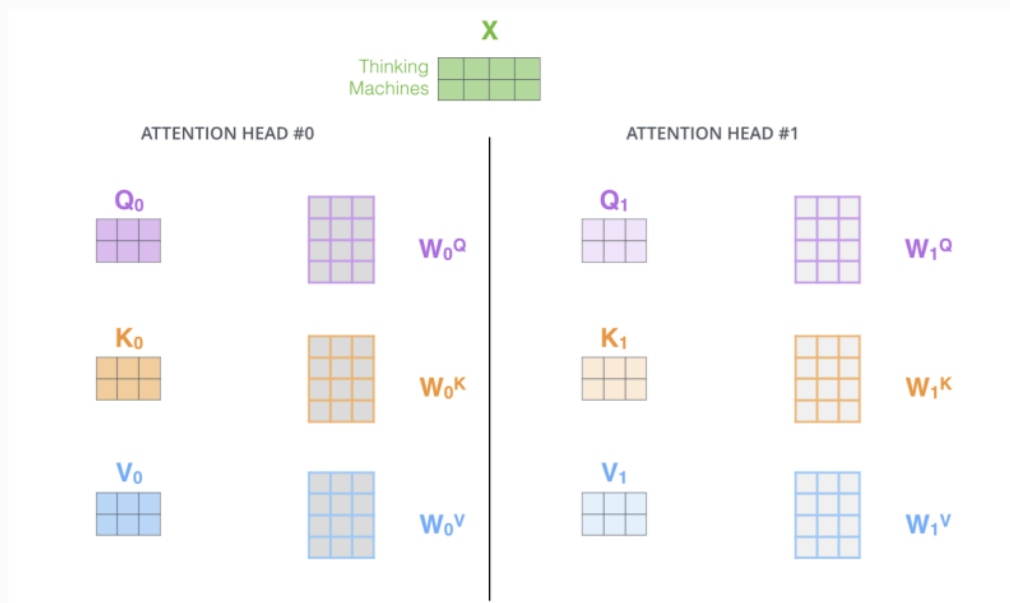
- Add position representations to the inputs
- Easy fix: apply the same feedforward network to each self-attention output.

# Multi-headed attention

---

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

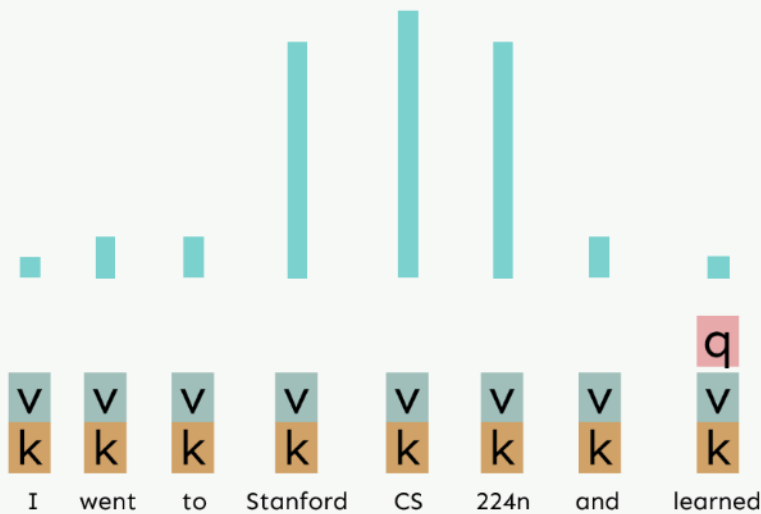
Maintain distinct weight matrices for each attention head—distinct representational subspaces:



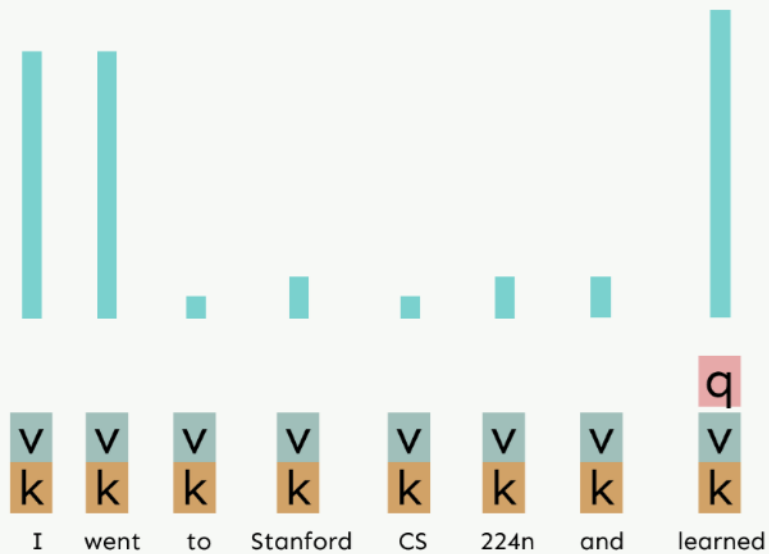


# Hypothetical example of multi-headed attention

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  $\ell$  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.

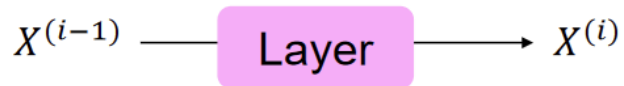
# Optimization tricks: residual connections and layer normalization

---

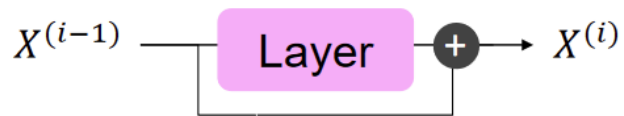
# Residual connections [He et al. 2016]

- **Residual connections** are a trick to help models train better.

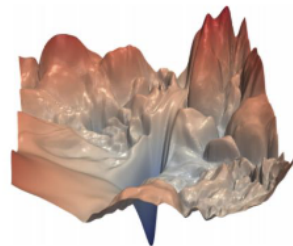
- Instead of  $X^{(i)} = \text{Layer}(X^{(i-1)})$  (where  $i$  represents the layer)



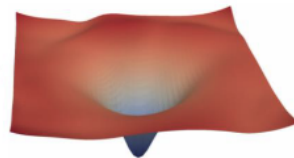
- We let  $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$  (so we only have to learn “the residual” from the previous layer)



- 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

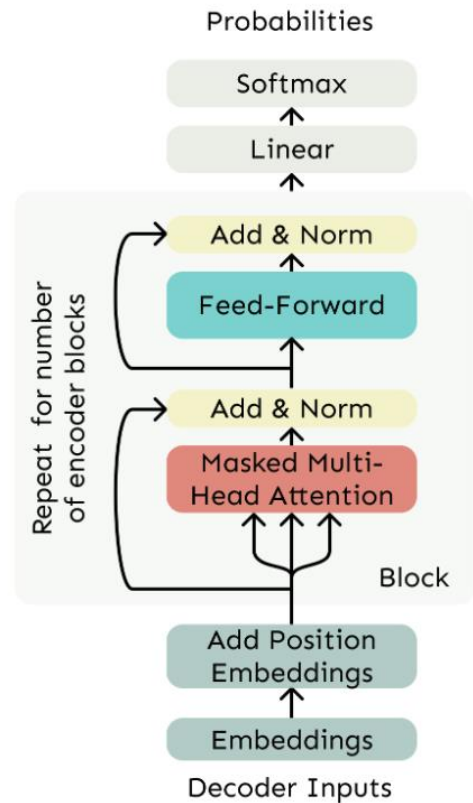
# Transformer blocks

---

# 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!



# 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

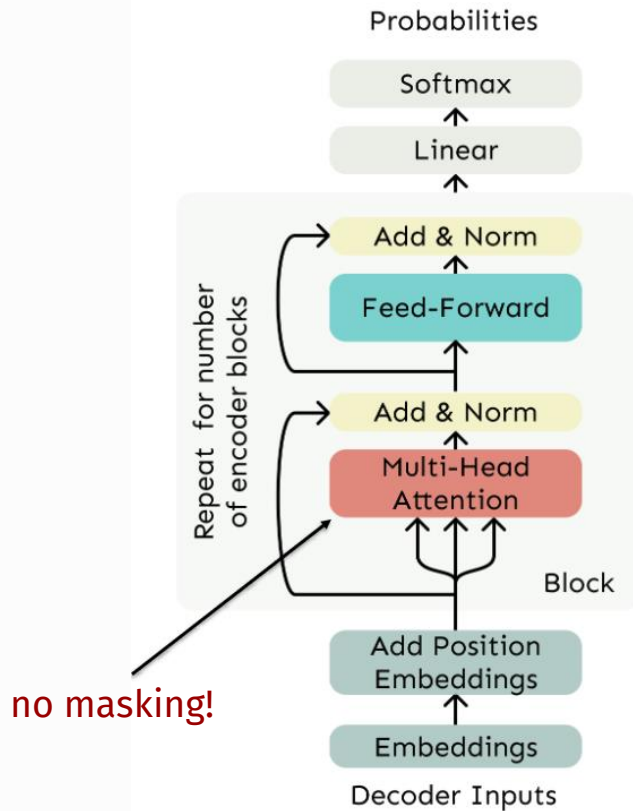
	[START]	The	chef	who
[START]		$-\infty$	$-\infty$	$-\infty$
The			$-\infty$	$-\infty$
chef				$-\infty$
who				



# The transformer encoder

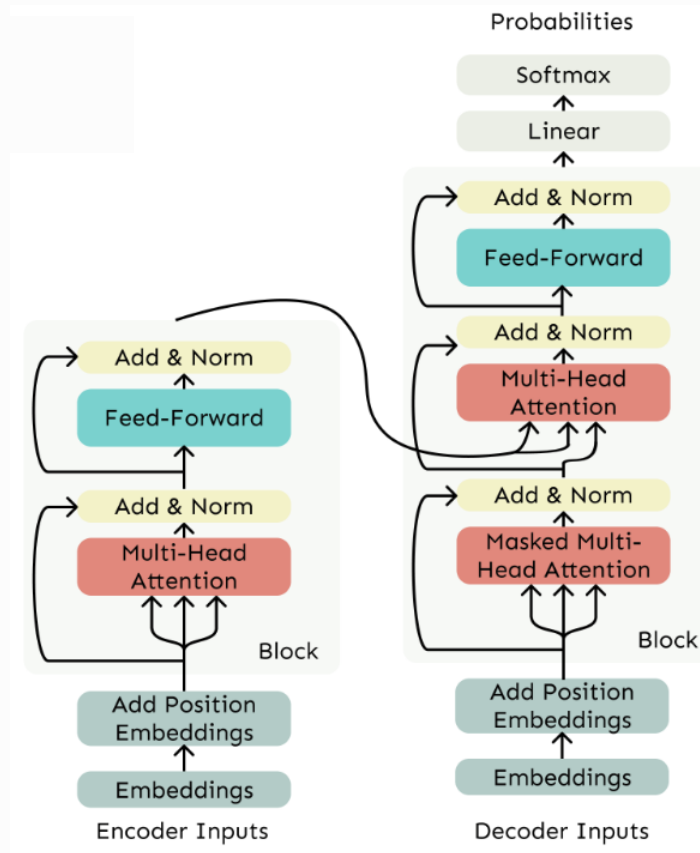
The Transformer Decoder constrains to unidirectional context, as for language models.

- 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.



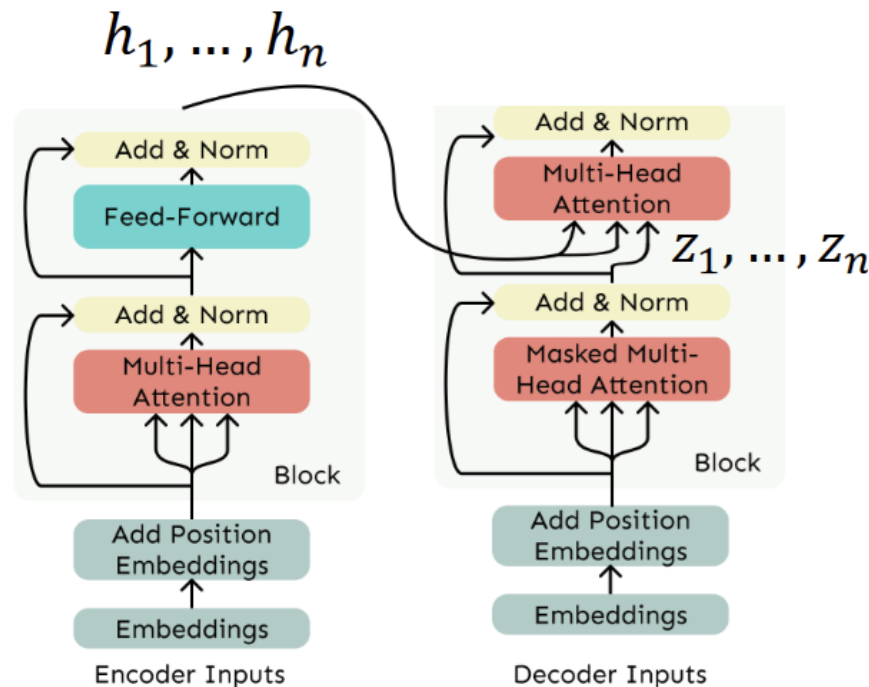
# The transformer encoder-decoder

- 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]

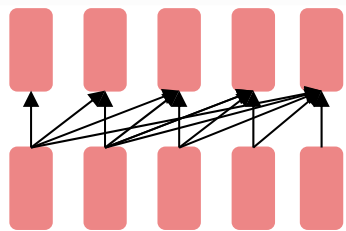
# Intro to large language models (LLMs)

---

# Large language models

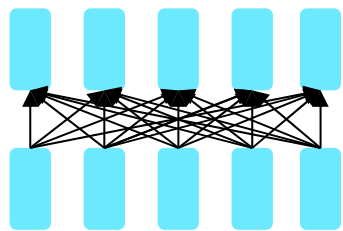
- Even though pretrained only to predict words
- Learn a lot of useful language knowledge
- Since training on a **lot** of text

# Three architectures for large language models



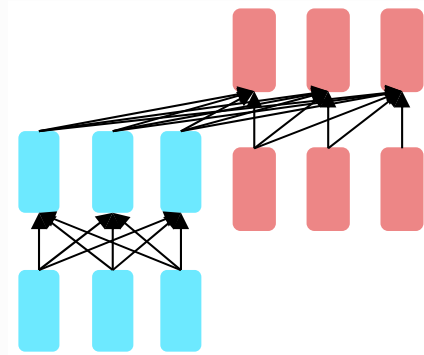
## Decoders

GPT, Claude,  
Llama, Mixtral



## Encoders

BERT family,  
HuBERT



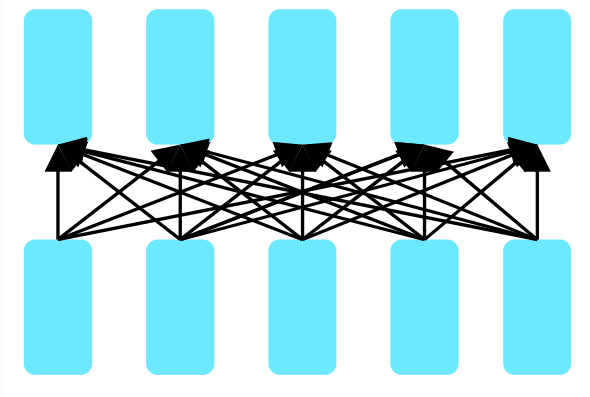
## Encoder-decoders

Flan-T5, Whisper

# Encoders

Many varieties!

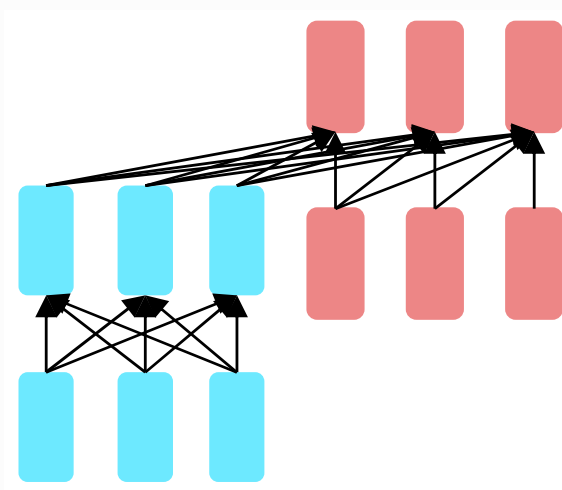
- Popular: Masked Language Models (MLMs)
- BERT family
- Trained by predicting words from surrounding words on both sides
- Are usually **finetuned** (trained on supervised data) for classification tasks.





# Encoder-Decoders

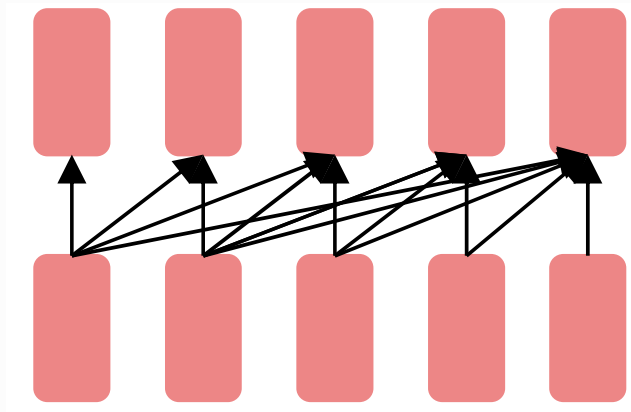
- Trained to map from one sequence to another
- Very popular for:
  - machine translation (map from one language to another)
  - speech recognition (map from acoustics to words)



# Decoder-only models

Also called:

- Causal LLMs
  - Autoregressive LLMs
  - Left-to-right LLMs
- 
- Predict words left to right

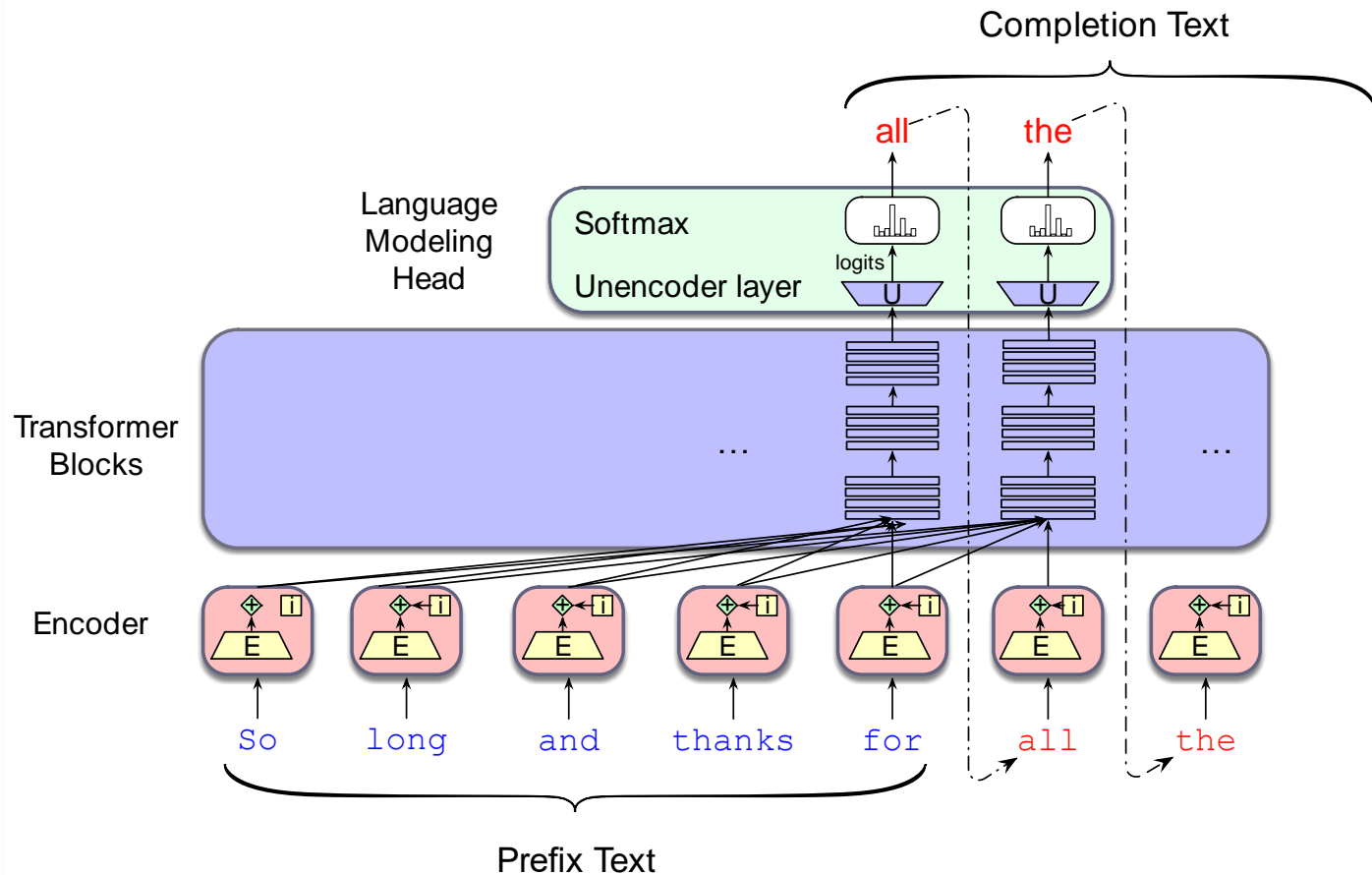


# Decoder-only models can handle many tasks

- Many tasks can be turned into tasks of predicting words!

# Conditional generation

Generating text  
conditioned  
on previous  
text!



# Many practical NLP tasks can be cast as word prediction!

Sentiment analysis: “I like Jackie Chan”

1. We give the language model this string:  
The sentiment of the sentence "I like Jackie Chan" is:
2. And see what word it thinks comes next:  
 $P(\text{positive} | \text{The sentiment of the sentence ``I like Jackie Chan" is:})$   
 $P(\text{negative} | \text{The sentiment of the sentence ``I like Jackie Chan" is:})$

# Framing lots of tasks as conditional generation

QA: “Who wrote The Origin of Species”

1. We give the language model this string:

Q: Who wrote the book ``The Origin of Species"? A:

2. And see what word it thinks comes next:

$P(w|Q: \text{Who wrote the book ``The Origin of Species"? A:})$

3. And iterate:

$P(w|Q: \text{Who wrote the book ``The Origin of Species"? A: Charles})$

# Summarization

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For \$89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

Original But not if you live in New England or surrounding states. “We will not ship snow to any states in the northeast!” says Waring’s website, ShipSnowYo.com. “We’re in the business of expunging snow!”

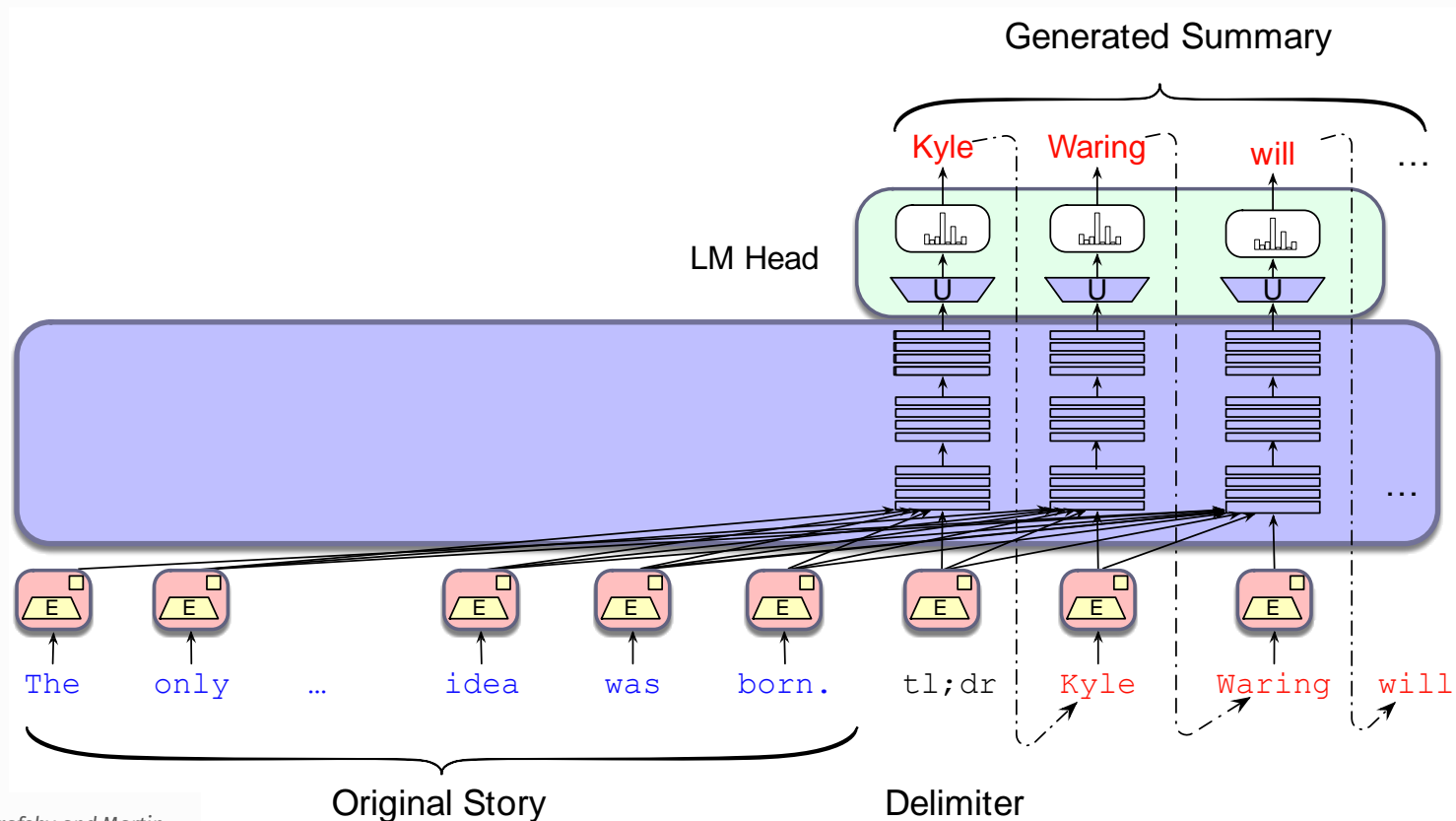
His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. [...]

## Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

# LLMs for summarization (using tl;dr)





# Sampling for LLM generation

---

# Decoding and Sampling

- This task of choosing a word to generate based on the model's probabilities is called **decoding**.
- The most common method for decoding in LLMs: **sampling**.
- Sampling from a model's distribution over words:
  - choose random words according to their probability assigned by the model.
- After each token we'll sample words to generate according to their probability *conditioned on our previous choices*,
  - A transformer language model will give the probability

# Random sampling

```
i ← 1  
wi ∼ p(w)  
while wi ≠ EOS  
    i ← i + 1  
    wi ∼ p(wi | w<i)
```

# Random sampling doesn't work very well

- Even though random sampling mostly generate sensible, high-probable words,
- There are many odd, low- probability words in the tail of the distribution
- Each one is low- probability but added up they constitute a large portion of the distribution
- So they get picked enough to generate weird sentences

# Factors in word sampling: **quality** and **diversity**

Emphasize **high-probability** words

- + **quality**: more accurate, coherent, and factual,
- **diversity**: boring, repetitive.

Emphasize **middle-probability** words

- + **diversity**: more creative, diverse,
- **quality**: less factual, incoherent

# Top-k sampling:

1. Choose # of words  $k$
2. For each word in the vocabulary  $V$ , use the language model to compute the likelihood of this word given the context  $p(w_t | w_{<t})$
3. Sort the words by likelihood, keep only the top  $k$  most probable words.
4. Renormalize the scores of the  $k$  words to be a legitimate probability distribution.
5. Randomly sample a word from within these remaining  $k$  most-probable words according to its probability.

# Top-p sampling (= nucleus sampling)

Holtzman et al., 2020

Problem with top- $k$ :  $k$  is fixed so may cover very different amounts of probability mass in different situations

Idea: Instead, keep the top  $p$  percent of the probability mass

Given a distribution  $P(w_t | \mathbf{w}_{<t})$ , the top- $p$  vocabulary  $V(p)$  is the smallest set of words such that

$$\sum_{w \in V(p)} P(w | \mathbf{w}_{<t}) \geq p$$

# Temperature sampling

Reshape the distribution instead of truncating it

Intuition from thermodynamics,

- a system at high temperature is flexible and can explore many possible states,
- a system at lower temperature is likely to explore a subset of lower energy (better) states.

In **low-temperature sampling**, ( $\tau \leq 1$ ) we smoothly

- increase the probability of the most probable words
- decrease the probability of the rare words.



# Temperature sampling

Divide the logit by a temperature parameter  $\tau$  before passing it through the softmax.

Instead of  ~~$\mathbf{y} = \text{softmax}(u)$~~

We do  $\mathbf{y} = \text{softmax}(u/\tau)$

# Temperature sampling

$$\mathbf{y} = \text{softmax}(\mathbf{u}/\tau) \quad 0 \leq \tau \leq 1$$

Why does this work?

- When  $\tau$  is close to 1 the distribution doesn't change much.
- The lower of a fraction  $\tau$  is, the larger the scores being passed to the softmax
- Softmax pushes high values toward 1 and low values toward 0.
- Large inputs (lower temperature) pushes high-probability words higher and low probability word lower, making the distribution more greedy.
- As  $\tau$  approaches 0, the probability of most likely word approaches 1

# Wrapping up

- Transformers are a high-performing NLP architecture based on self-attention
- Transformers can be used for language modeling
- Transformer-based language models pretrained on lots of text are called **large language models (LLMs)**
- LLMs can have decoder-only, encoder-only, or encoder-decoder architectures
- Decoder-only LLMs can cast many different NLP tasks as word prediction
- There are many different sampling approaches that balance diversity and quality in text generation from LLMs

*Questions?*