Transformers

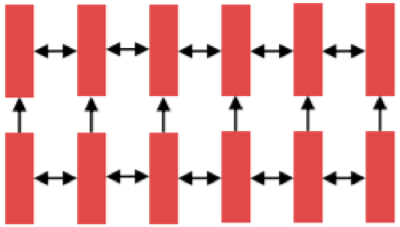
ECE324, Winter 2023

Michael Guerzhoy

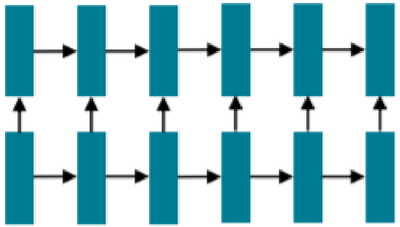
Slides from John Hewitt
http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture09-transformers.pdf

Language models in the last 5 years

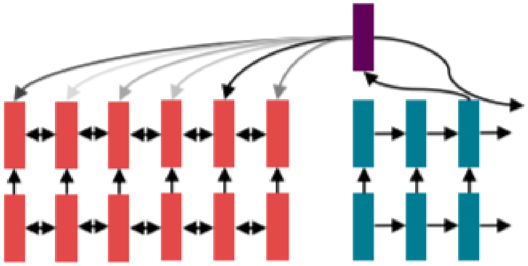
- Circa 2016, the de facto strategy in NLP is to encode texts with a bidirectional LSTM: (for example, the source sentence in a translation)



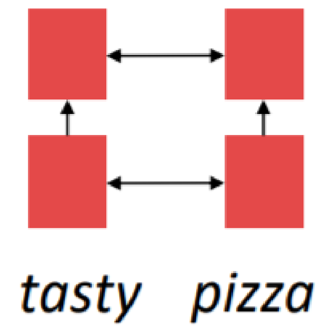
- Define your output (parse, sentence, summary) as a sequence, and use an LSTM to generate it.



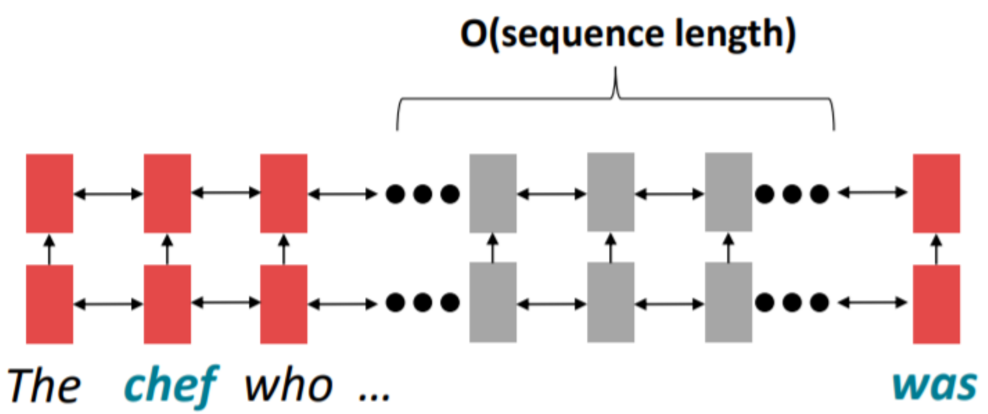
- Use attention to allow flexible access to memory



Recurrent models: linear interaction distance

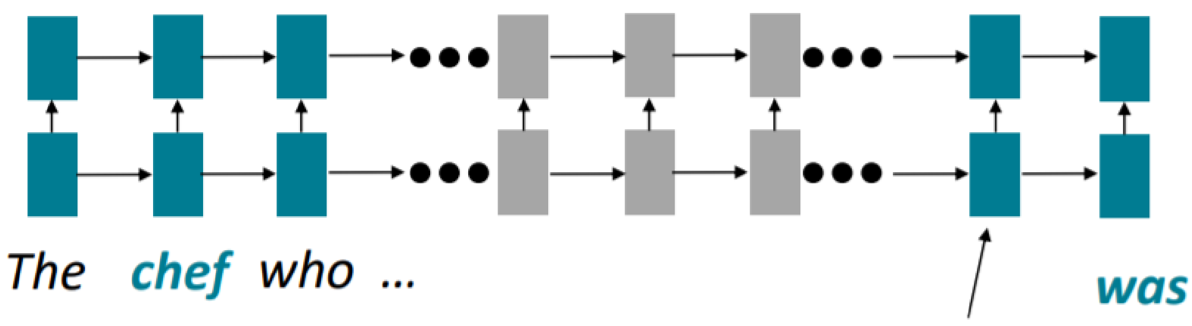


- RNNs are unrolled "left-to-right"
- This encodes linear locality: a useful heuristic
  - Nearby words often affect each other's meaning
- Problem: RNNs take O(sequence length) steps for distant word pairs to interact



Recurrent models: linear interaction distance

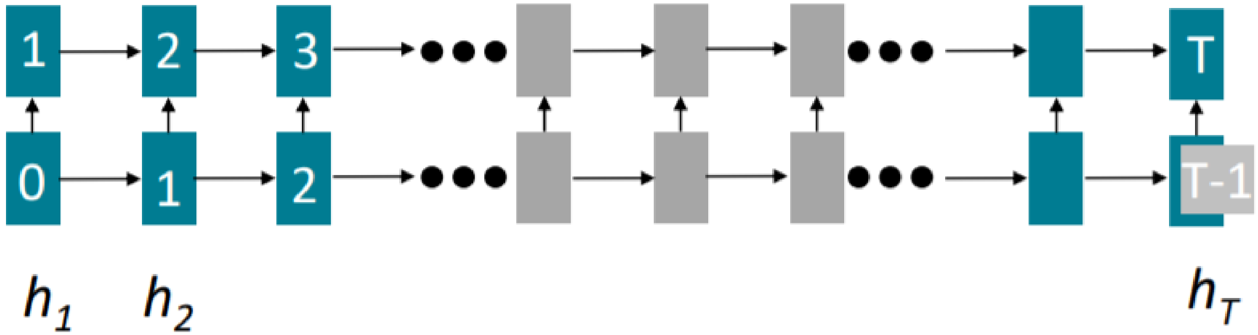
- O(sequence length) steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (vanishing gradient problems)
  - Linear order of words is "baked in"



Info of ***chef*** has gone through O(sequence length) many layers!

Recurrent models: lack of parallelizability

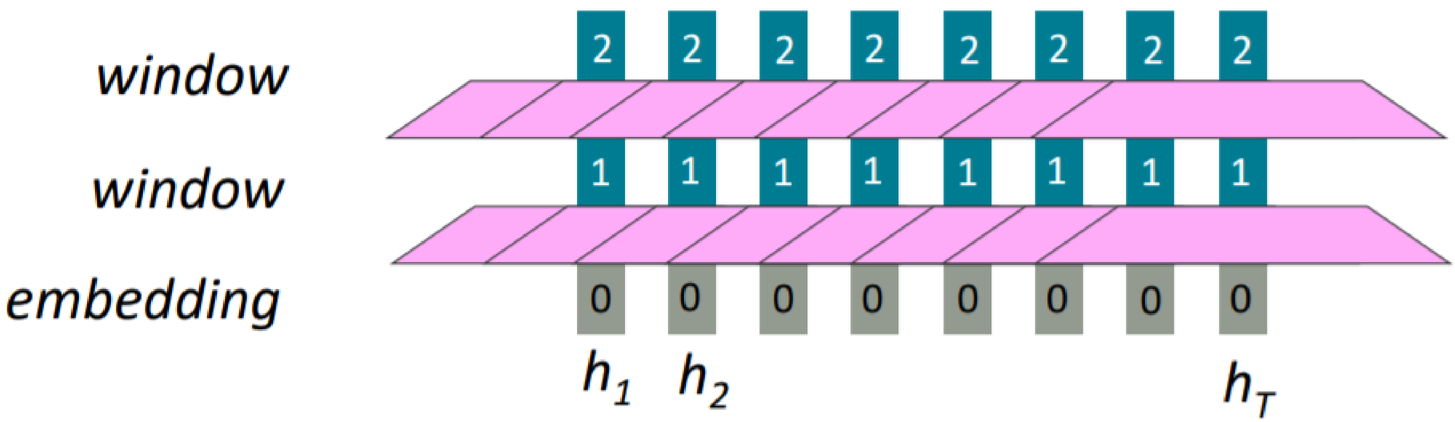
- Forward and backward passes have **O(sequence length)** unparallelalizable operations
  - Can’t compute the state of an RNN with one matrix multiplication: need to compute states sequentially
    - Can’t use GPU parallelization
    - (But can process multiple sequence at a time)



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

Alternative: word windows

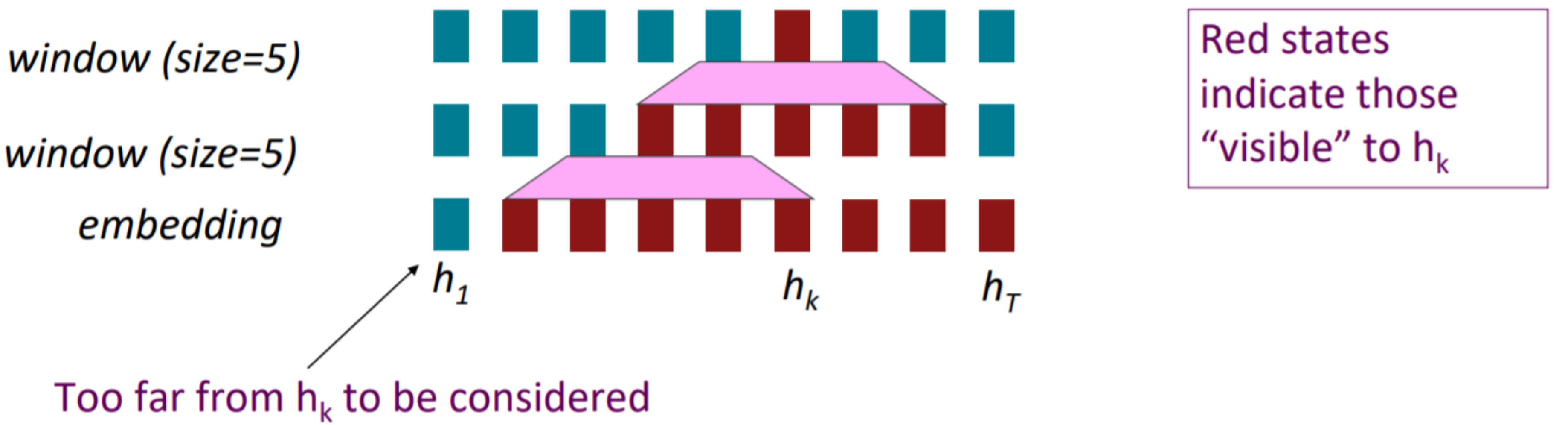
- Word window models aggregate local context
  - AKA 1D convolution
  - Number of unparallelizable operations does not increase with sequence length



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

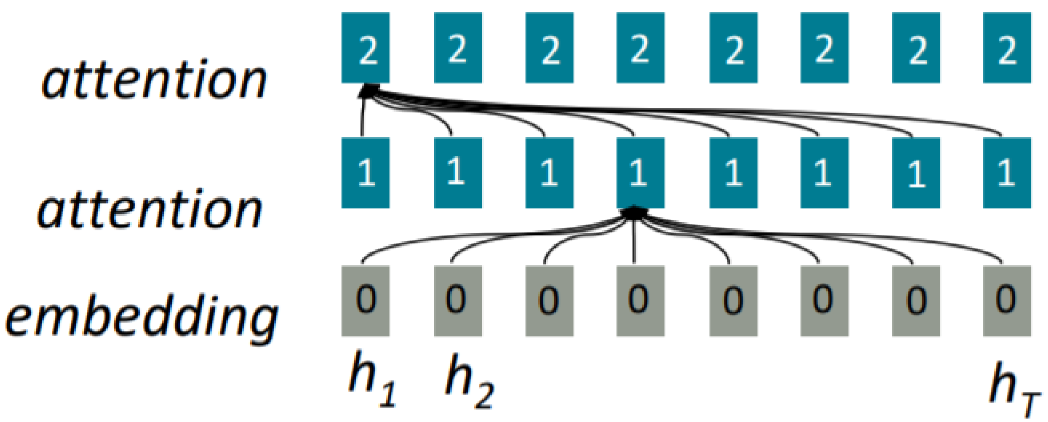
Alternative: word windows

- Stacking word-window layers allows interactions between words that are farther apart
- Maximum interaction distances = seq length/window size
  - More long-distance context would be ignored



Alternative: attention

- Attention treats each word's representation as a query to access and incorporate information from a set of values
- Number of unparallelizable operations does not increase 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

(Vanilla) Self-attention

- Attention operates on queries, keys, and values.
  - Queries $q_1, q_2, \ldots$
  - Keys $k_1, k_2, \ldots$
  - Values $v_1, v_2 \ldots$

$$e_{ij} = q_i^{\top} k_j$$

Compute **key-query** affinities

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

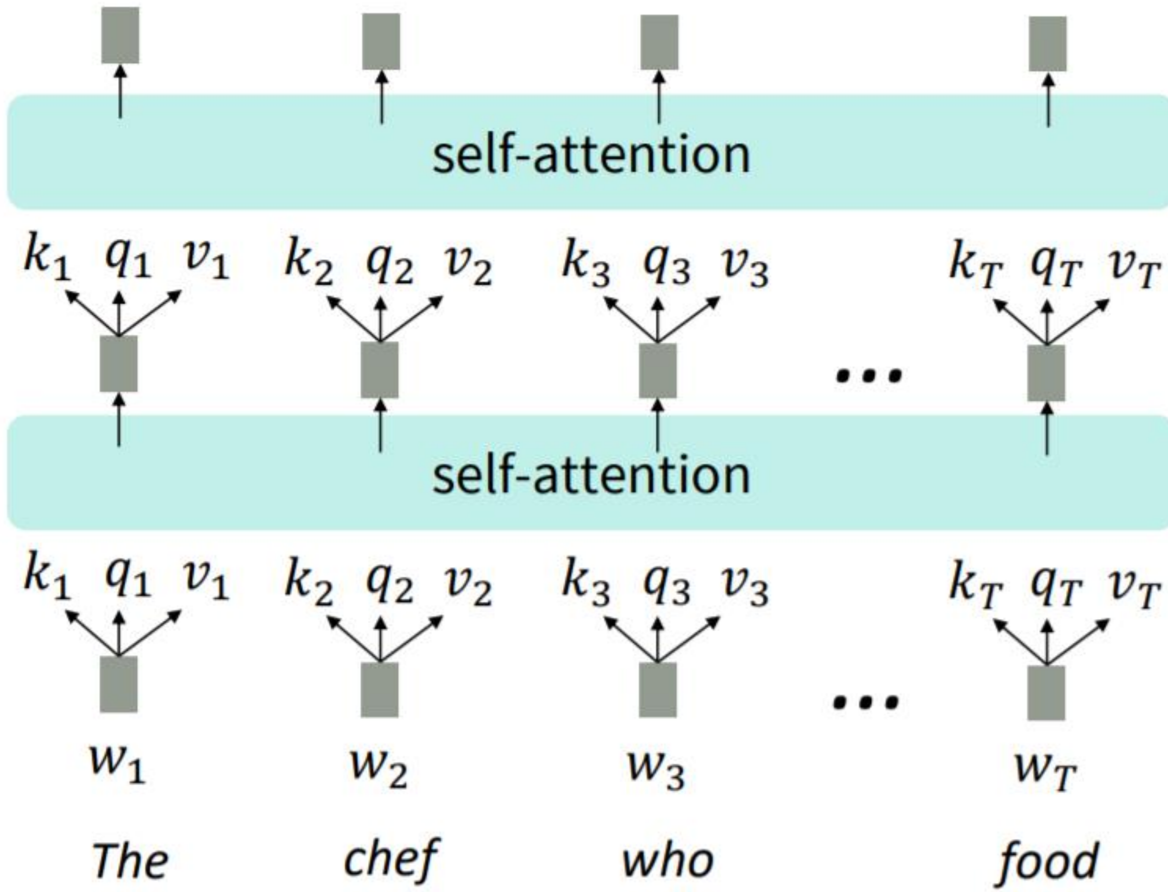
Compute attention weights from affinities (softmax)

$$\text{output}_i = \sum_j \alpha_{ij} v_j$$

Compute outputs as weighted sum of **values**

- Self-attention: use $\mathrm{q}_i = v_i = k_i = x_i$ for layer x

Self attention blocks



Self-attention doesn't know the order of its inputs.

Self-attention: sequence order

- Self-attention doesn't know about the order of the inputs
- Make position vectors $p_i \in R^d$ (same dimensionality is *x*) for $i \in \{1, 2, \ldots, T\}$
- Make $\tilde{x}_i = x_i + p_i$

$$\tilde{v}_i = v_i + p_i$$
$$\tilde{q}_i = q_i + p_i$$
$$\tilde{k}_i = k_i + p_i$$

Position vectors through sinusoids

- Sinusoidal position representation: concatenate sinusoidal functions of varying periods

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

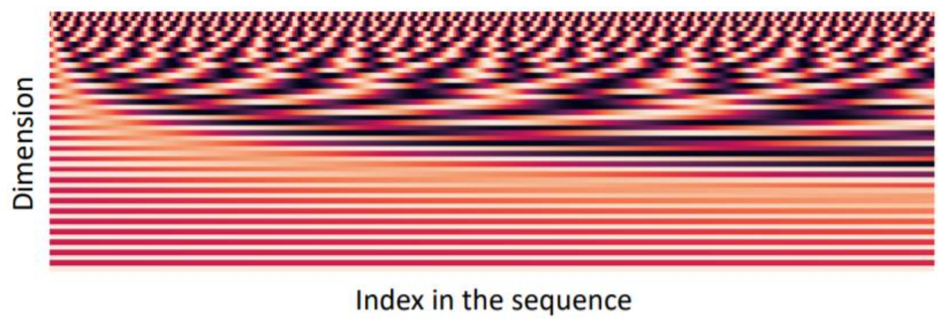


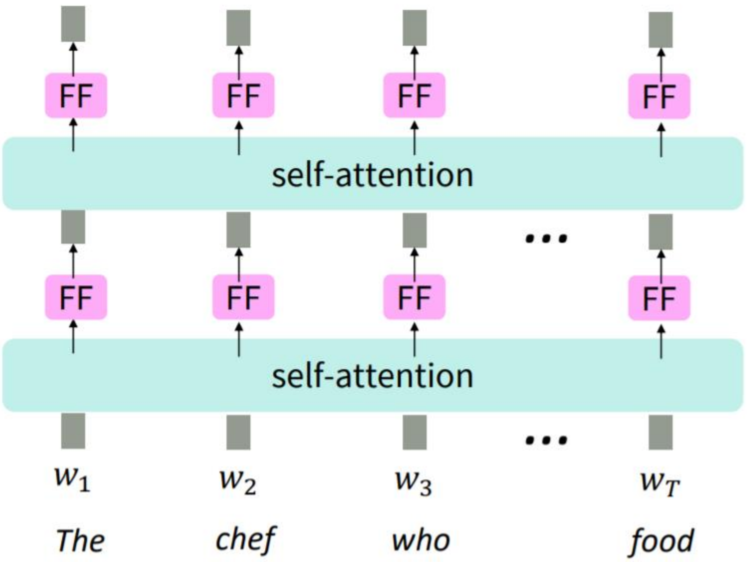
Image: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

Position representation vectors learned from scratch

- Learned absolute position representations: Let all p_i be learnable parameters
  - Learn a matrix $p \in R^{d \times T}$, and let each p_i be a column of that matrix
- Pro: each position gets to be learned to fit the data
- Con: cannot extrapolate outside of 1, …, T
- Most systems use this

Adding nonlinearities in self-attention

- Add feed-forward network to post-process each output vector
- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors

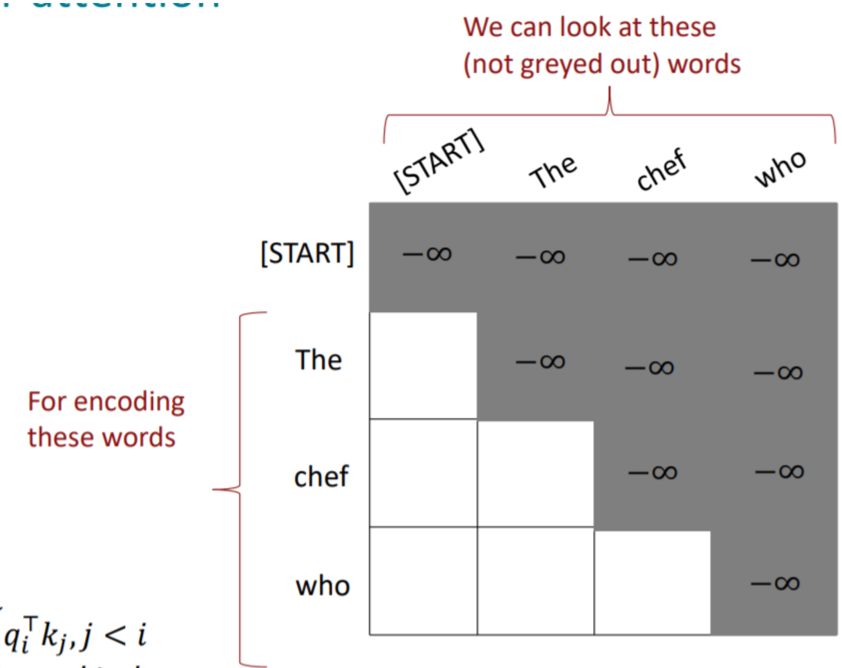


Intuition: the FF network processes the result of attention

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$

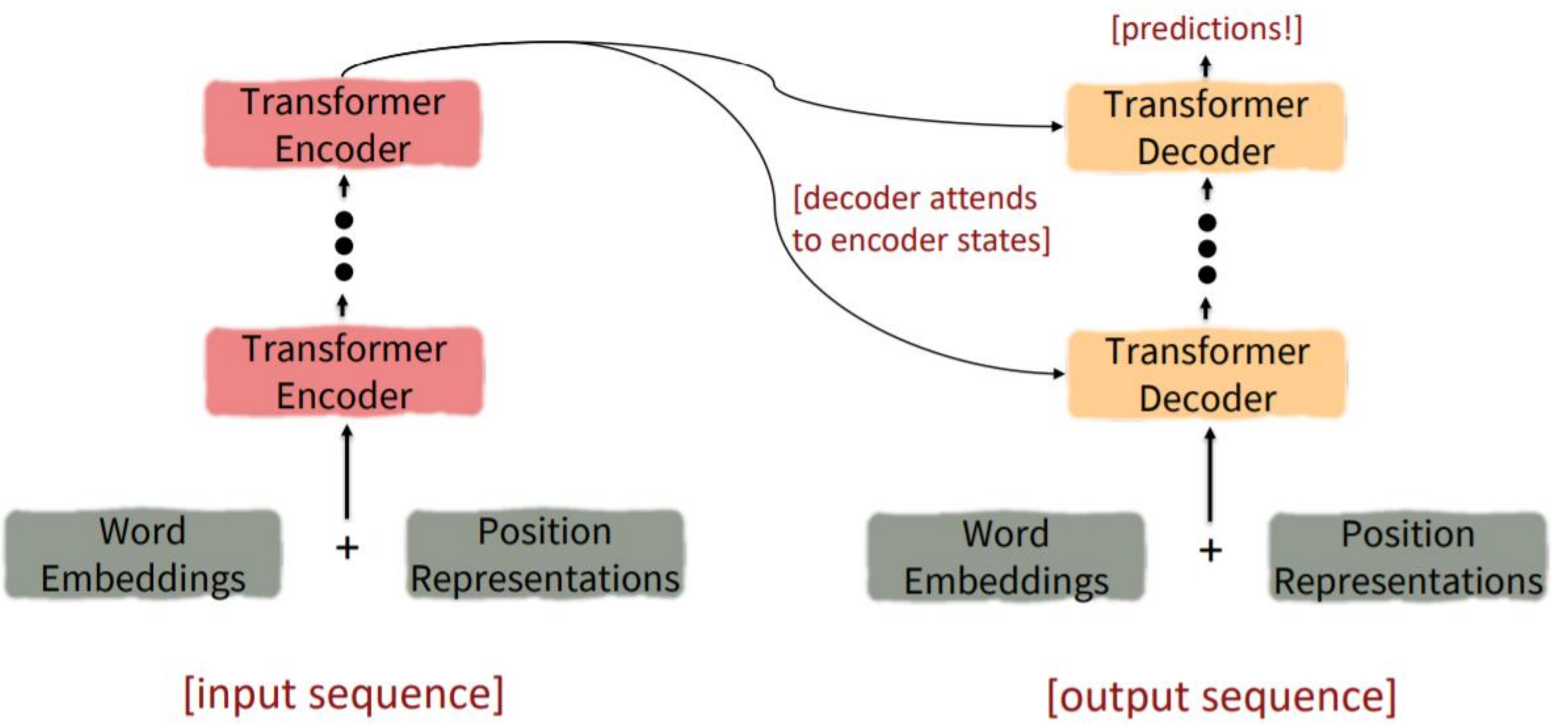
Masking the future in self-attention

- To use self-attention in decoders, we need to ensure we can't peek in the future
- We mask out attention to future words by setting attention scores to $-\infty$



$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$

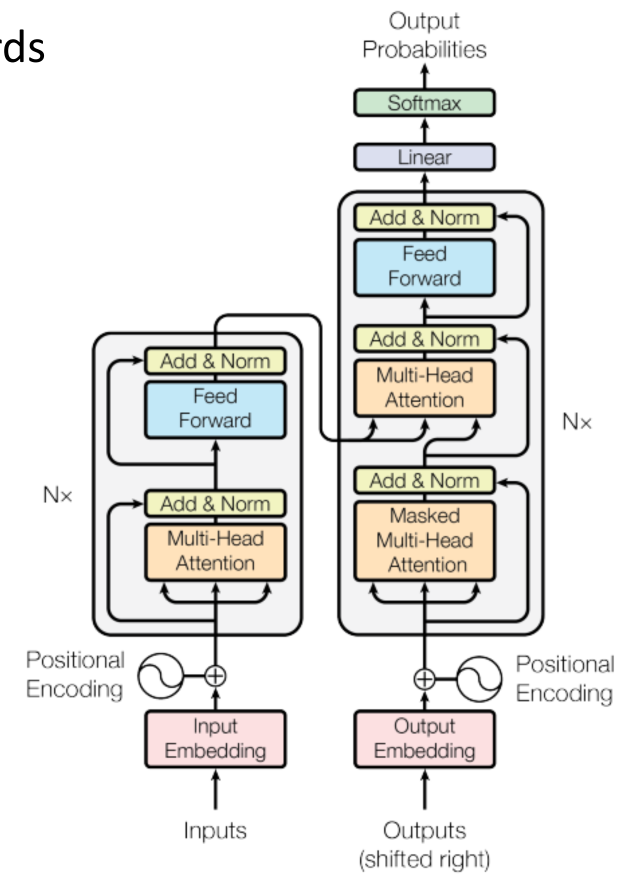
The Transformer Encoder-Decoder



Training the network

Maximize the probabilities of the observed words
In the training set

$$\max_{\theta} \sum_{t} \log P(w_t | w_{1\ldots t-1})$$



The Transformer Encoder: Key-Query-Value Attention

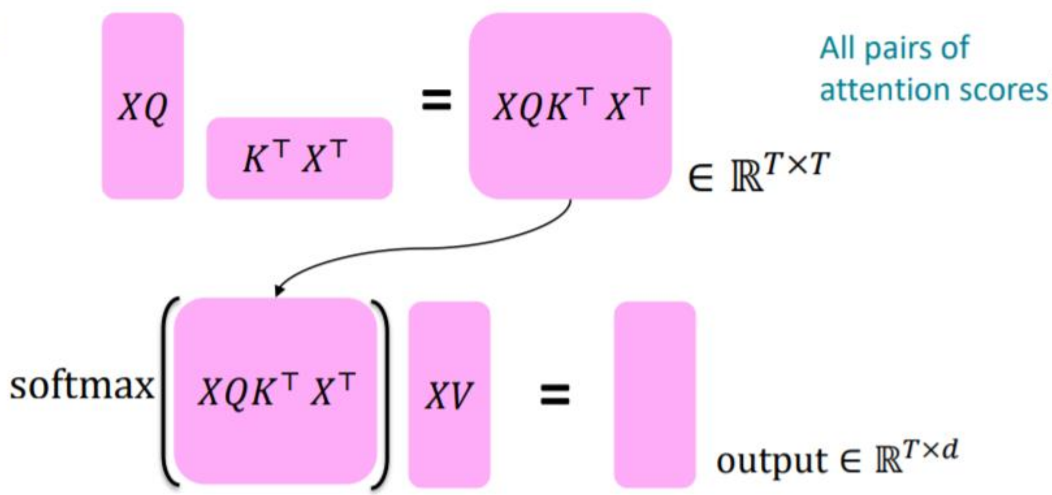
- Let $x_1, x_2, \ldots, x_T$ be the input vectors, $x_i \in R^d$
- Then the keys, queries, and values are
  - $k_i = x_i' K, K \in R^{d \times d}$
  - $q_i = x_i' Q, Q \in R^{d \times d}$
  - $v_i = x_i' V, V \in R^{d \times d}$
- Allow for different aspects of the *x* vectors to be used/emphasized in each of the three different roles

The Transformer Encoder: Key-Query-Value Attention

- Let $\mathrm{X} = [x_1;\ \ldots; x_T] \in R^{T\times d}$
- Then $XK \in R^{T\times d}$, $XQ \in R^{T\times d}$, $XV \in R^{T\times d}$
- The output is $softmax(XQ(XK)^T)XV$

First, take the query-key dot products in one matrix multiplication: $XQ(XK)^{\top}$

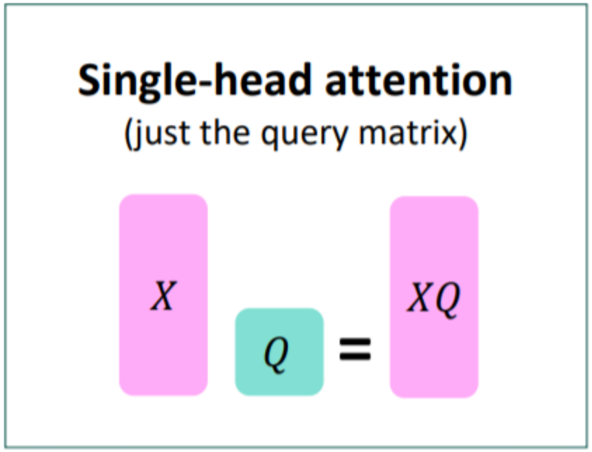
Next, softmax, and compute the weighted average with another matrix multiplication.

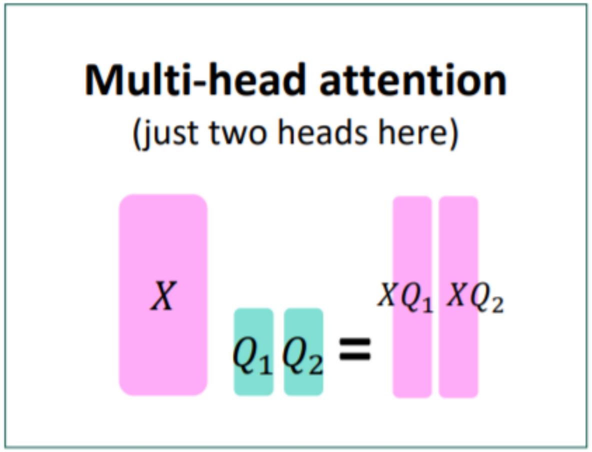


Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x_i^T Q^T K x_j$ 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 entries
- Let $Q_l, K_l, V_l \in R^{d \times \frac{d}{h}}$, where *h* is the number of attention heads, and *l* ranges from 1 to h
- Each attention head performs attention independently:
  $output_l = softmax\left(XQ_l K_l^T X^T\right)XV$, where $output_l \in R^{d/h}$
- The outputs of all the heads are combined:
  - output = $Y[output_1; \ldots; output_h]$, where $Y \in R^{d \times d}$
- Each head gets to "look" at different things, and construct value vectors differently.

Multi-headed attention



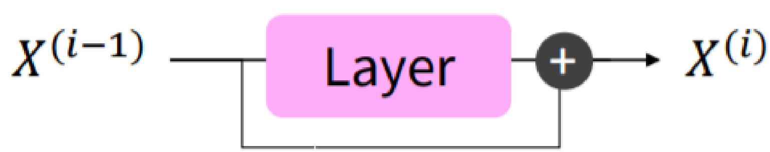


Same amount of computation as single-head self-attention!

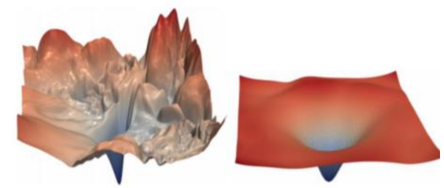
Residual connections

- Residual connections are a trick to help models train better
  - Instead of $X^{(i)} = Layer\left(X^{(i-1)}\right)$

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



  - Residual connections are thought to make the loss landscape considerably smoother



[no residuals] [residuals]

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

Layer normalization

- 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
- Let $x \in R^d$ be an individual (word) vector in the model
- Let $\mu = \frac{1}{d}\sum_j x_j$ and $\sigma = \sqrt{\frac{1}{d}\sum_j \left(x_j - \mu\right)^2}$
- output = $\gamma \frac{x-\mu}{\sqrt{\sigma}+\epsilon} + \beta$

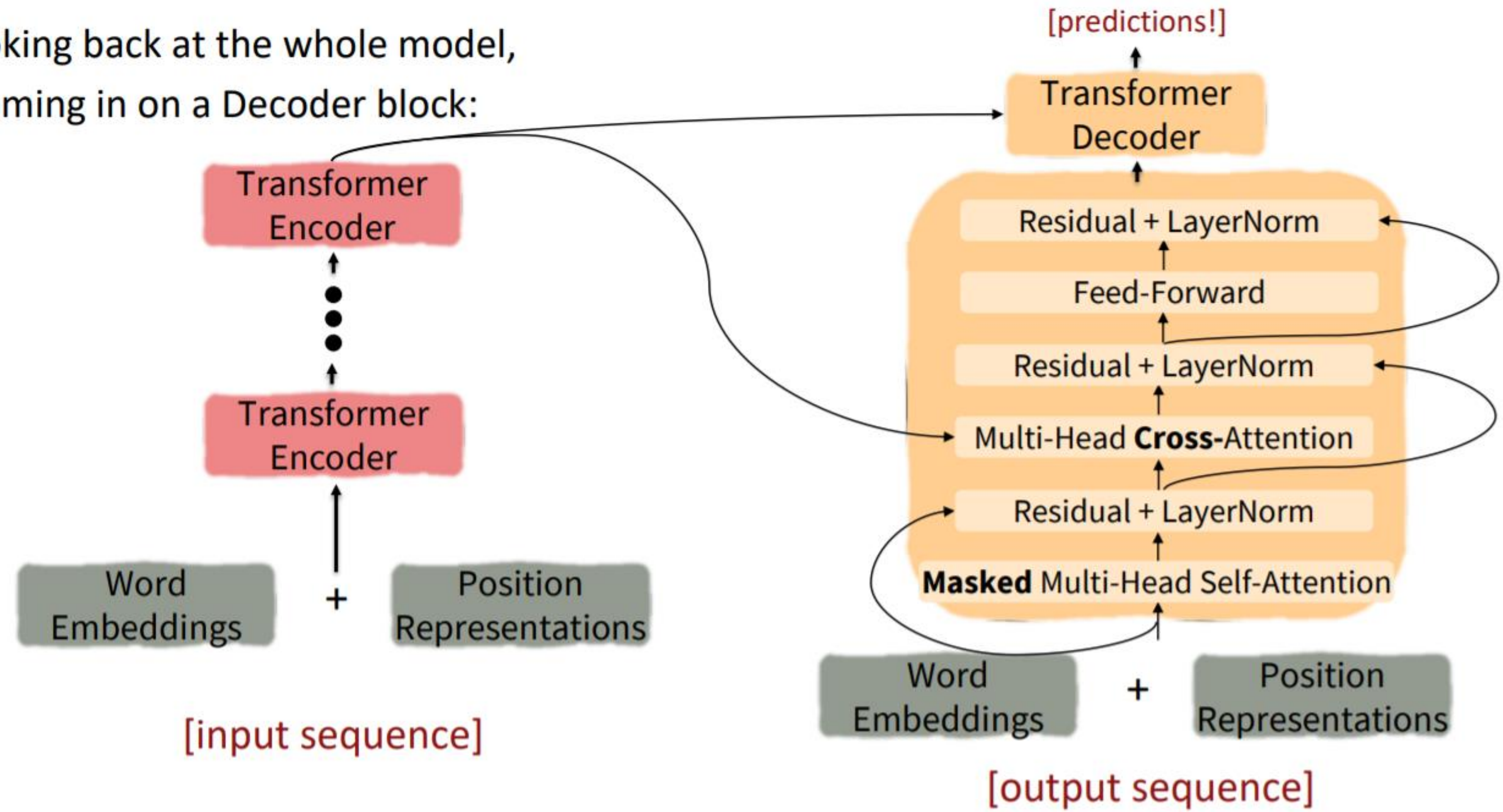
Scaled dot product

- When dimensionality d becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.
- Instead of the self-attention function we've seen

$$output_l = softmax\left(XQ_lK_l^TX^T\right)XV_l,$$

- We divide the attention scored by $\sqrt{d/h}$ to stop the scores from becoming large just as a function of d/h (the dimensionality divided by the number of heads)
  - $output_l = softmax\left(\frac{XQ_lK_l^TX^T}{\sqrt{\frac{d}{h}}}\right)XV_l$

Looking back at the whole model, zooming in on a Decoder block:



Cross-attention

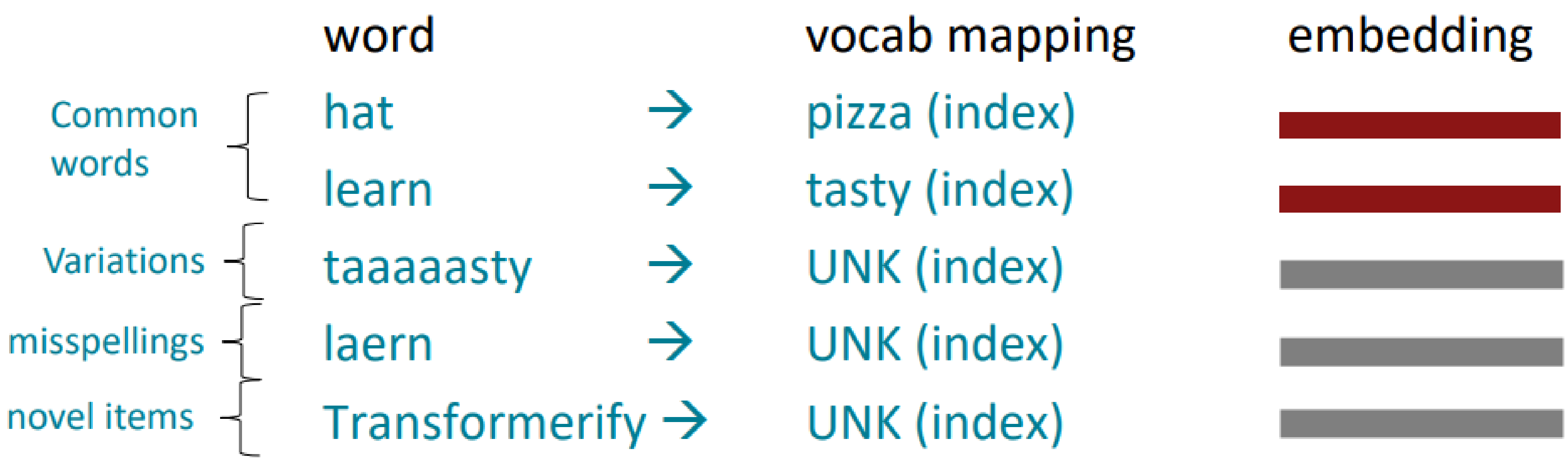
- Let $h_1, \ldots, h_T$ be the output vectors from the Transformer encoder with $x_i \in R^d$
- Let $z_1, \ldots, z_T$ be the input vectors from the Transformer decoder, $z_i \in R^d$
- Then keys and values are drawn from the encoder
  - $k_i = Kh_i, v_i = Vh_i$
- Queries are drawn from the decoder, $q_i = Qz_i$

Transformer complexity

- Quadratic compute in self-attention in the original model
  - Computing all pairs of interactions means our computation grows quadratically with the sequence length
  - For recurrent models, it only grew linearly

Word structure and subword models

- Baseline: a fixed vocabulary from the training set. Unknown words are all mapped to UNK

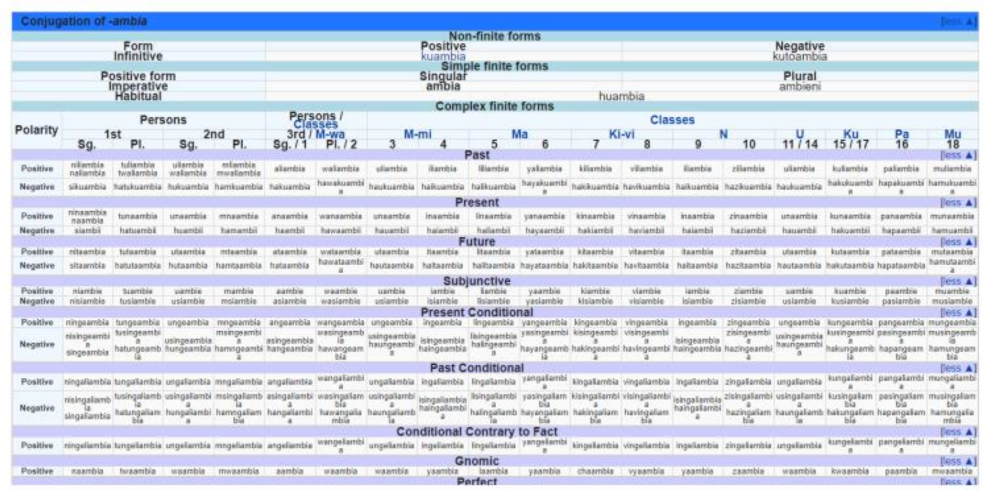


Word structure and subword models

- In many languages, finite vocabulary assumptions make less sense than in English
- Example: Swahili verbs can have hundreds of conjugations, each encoding a wide variety of information. (Tense, mood, definiteness, negation, information about the object, ++)

- Conjugation of *ambia*

(to tell) from Wiktionary

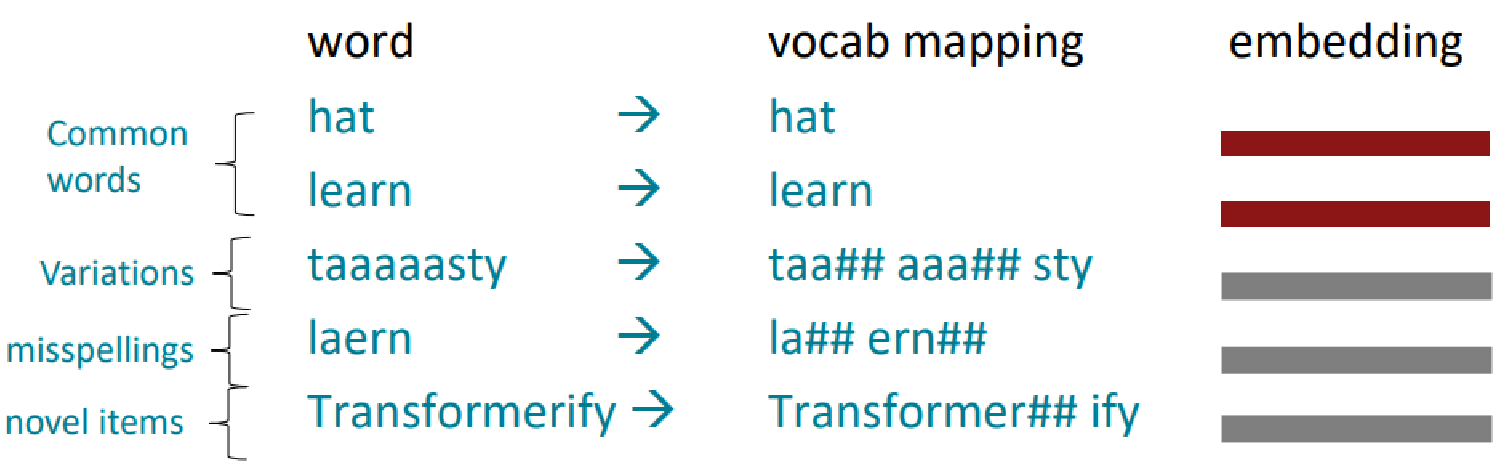


Byte-pair encoding algorithm

- Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)
  - The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens).
  - At training and testing time, each word is split into a sequence of known subwords
- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary.
  1. Start with a vocabulary containing only characters and an "end-of-word" symbol
  2. Using a corpus of text, find the most common adjacent characters "a,b"; add "ab" as a subword
  3. Replace instances of the character pair with the new subword; repeat until desired vocab size

Word structure and subword models

- Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components
- In the worst case, words are split into as many subwords as they have characters.



Pretraining Transformers

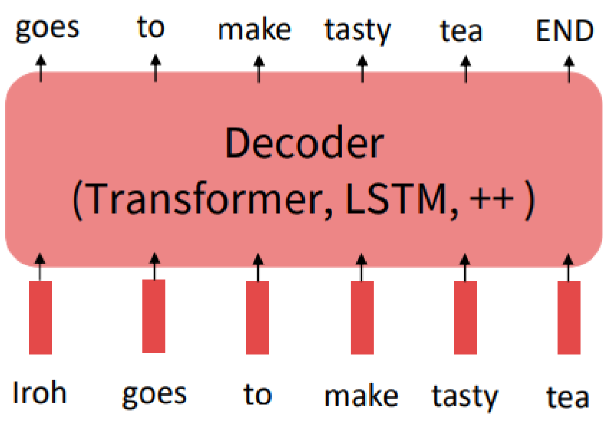
- Idea: pre-train the network on a large corpus of text
- Fine-tune on your dataset

Pretraining through language modeling

- Language modeling task:
  - Model $p_\theta(w_t|w_{1:t-1})$, the probability distribution for the next word after $w_{1:t-1}$
- Lots of data to train on

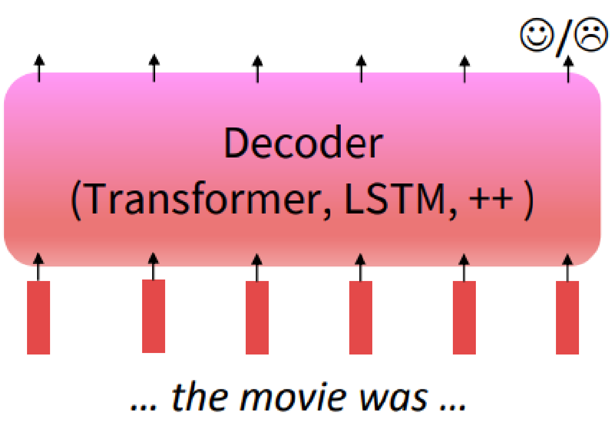
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!



What can language modelling tell us?

I put ___ fork down on the table.

The woman walked across the street,
checking for traffic over ___ shoulder.

I went to the ocean to see the fish, turtles, seals, and _____.

Overall, the value I got from the two hours watching
it was the sum total of the popcorn and the drink.
The movie was ___.

Iroh went into the kitchen to make some tea.
Standing next to Iroh, Zuko pondered his destiny.
Zuko left the ______.

I was thinking about the sequence that goes
1, 1, 2, 3, 5, 8, 13, 21, ____

Using pre-trained decoders

- Ignore that they're trained to model $p(w_t|w_{1\ldots t-1})$
- Fine-tune by training a classifier on the the hidden state

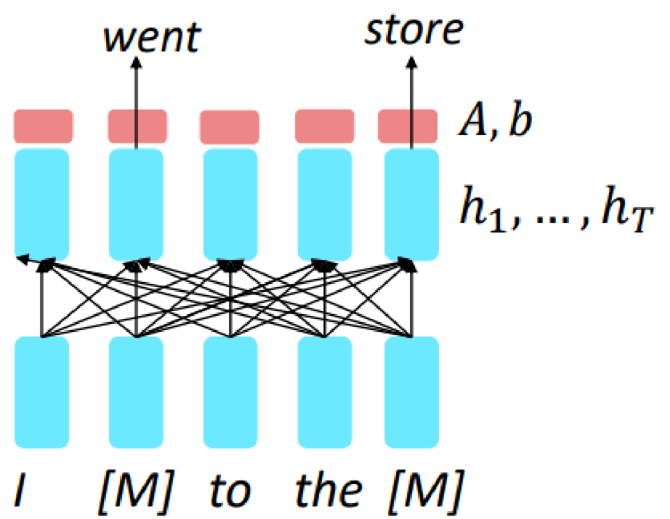
$$h = Decoder(w_1, \ldots w_t)$$
$$y \sim Ah_T + b$$

Pre-training encoders

- Idea: replace some fraction of words in the input with a special [MASK] token; predict those words

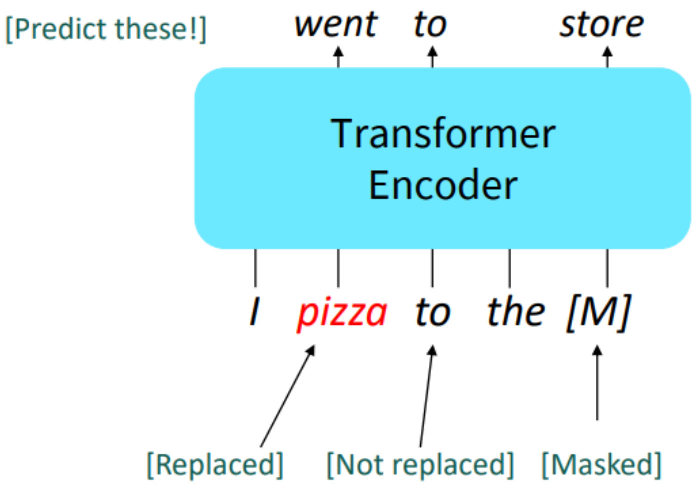
$$h_1, \dots, h_T = Encoder(w_1, \dots, w_T)$$
$$y_i \sim Ah_i + b$$

- Only add loss terms from words that are “masked out”



BERT: Bidirectional Encoder Representations from Transformers

- Devlin et al., 2018 proposed the "Masked LM" objective and released the weights of a pretrained Transformer, a model they labeled BERT
- Some more details about Masked LM for BERT:
    - Predict a random 15% of (sub)word tokens
    - Replace input word with [MASK] 80% of the time
    - Replace input word with a random token 10% of the time
    - Leave input word unchanged 10% of the time (but still predict it!)
        - Why? Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



- BERT was massively popular and hugely versatile; finetuning BERT led to new state-ofthe-art results on a broad range of tasks.

- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI**: natural language inference over question answering data
- **SST-2**: sentiment analysis
- **CoLA**: corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B**: semantic textual similarity
- **MRPC**: microsoft paraphrase corpus
- **RTE**: a small natural language inference corpus