PURDUE UNIVERSITY NORTHWEST

Natural Language Processing
CS 52570

Lecture 15 ----- Self-Attention and Transformers

Prof. Yang Ni @ CS Department, PNW

Part of this course is derived using resources from Stanford CS 224N, originated by Dr. Manning and others.

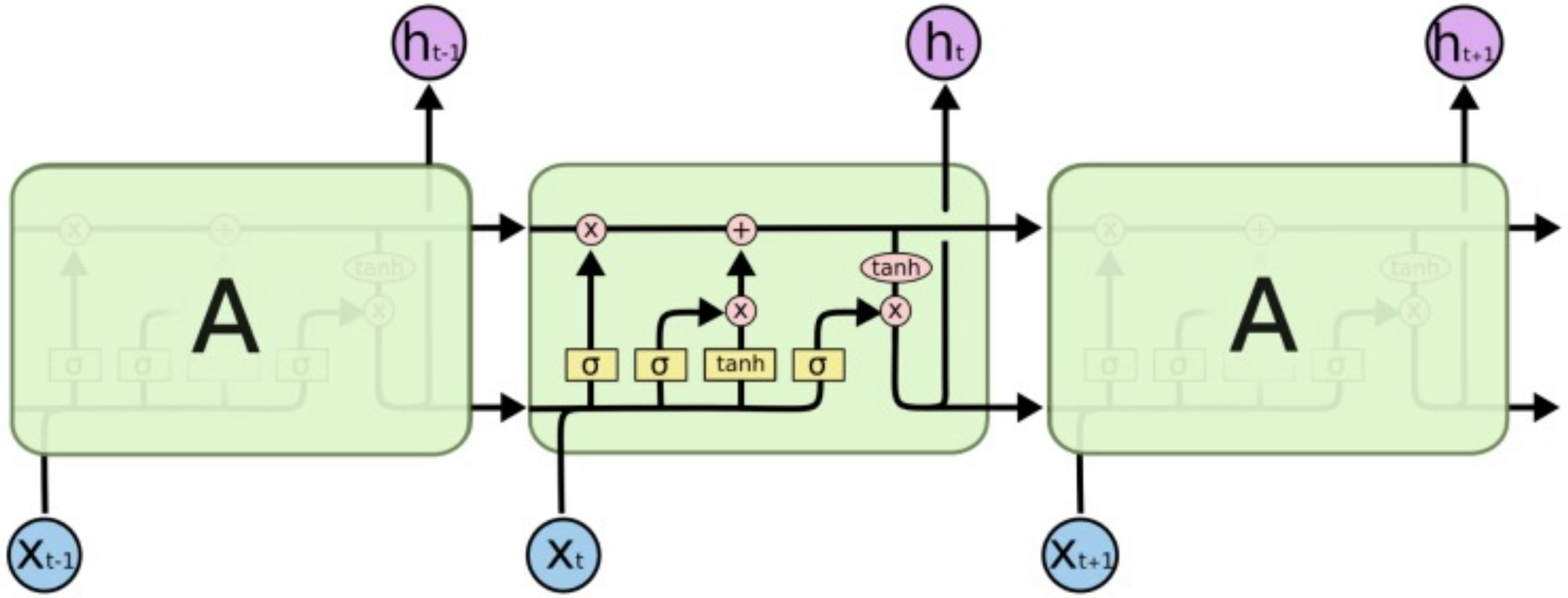
Recap Questions:

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

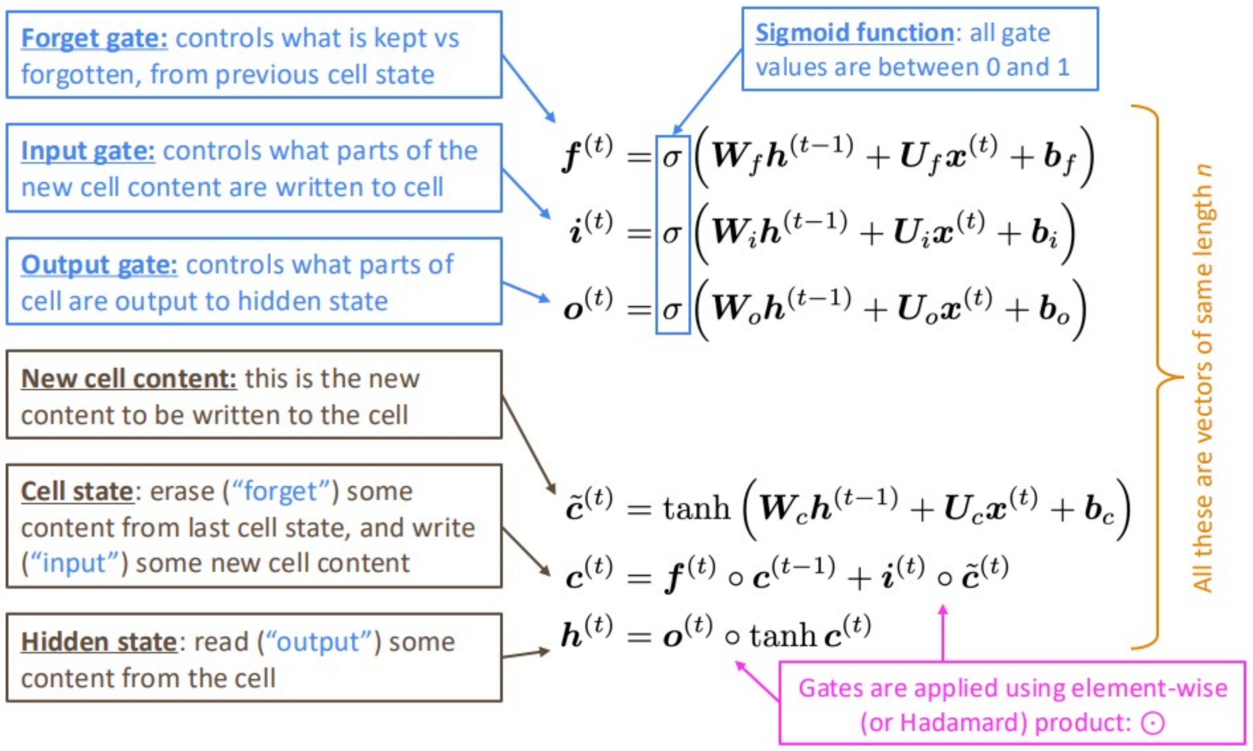
- This can cause bad updates: we take too large a step and reach a weird and bad parameter configuration (with large loss)
- In the worst case, this will result in **Inf** or **NaN** in your network (then you have to restart training from an earlier checkpoint)

What are the components in LSTM?



Long Short-Term Memory (LSTM)

We have a sequence of inputs $x\,(t)$, and we will compute a sequence of hidden states $h\,(t)$ and cell states $c\,(t)$. On timestep t:



Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

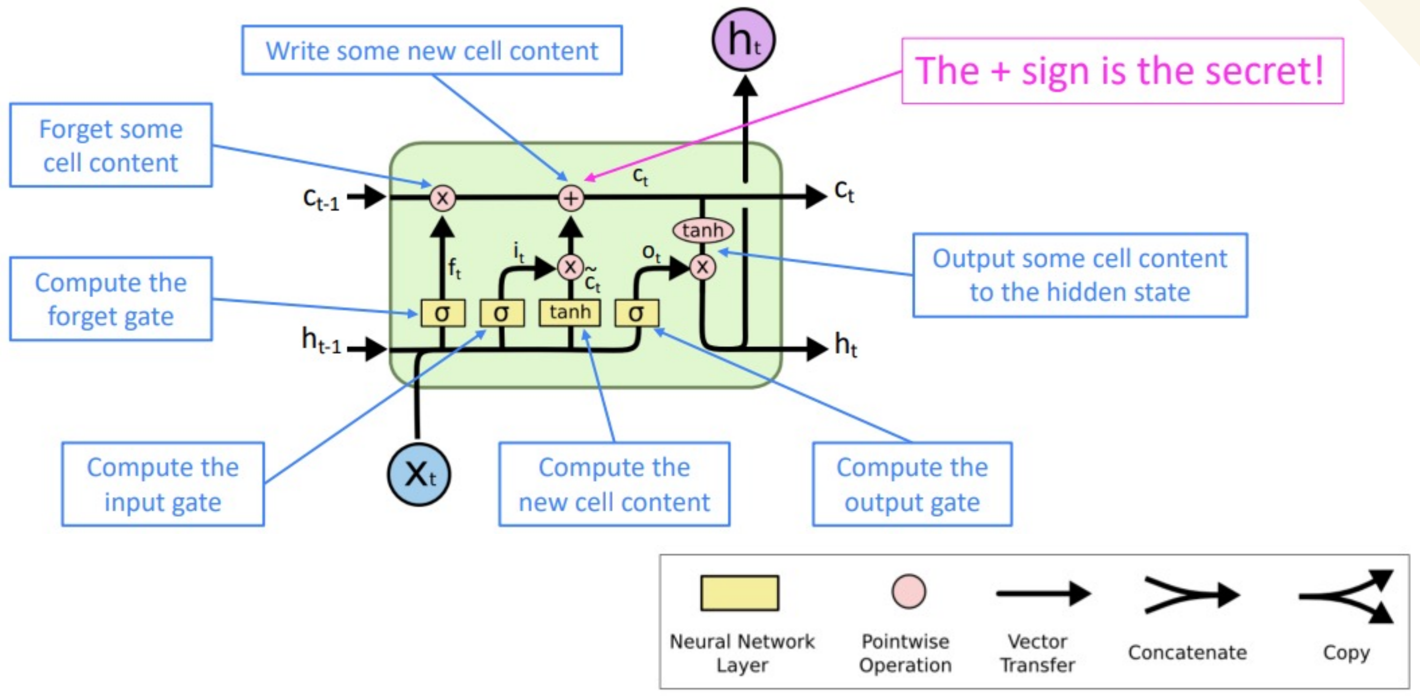
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length n

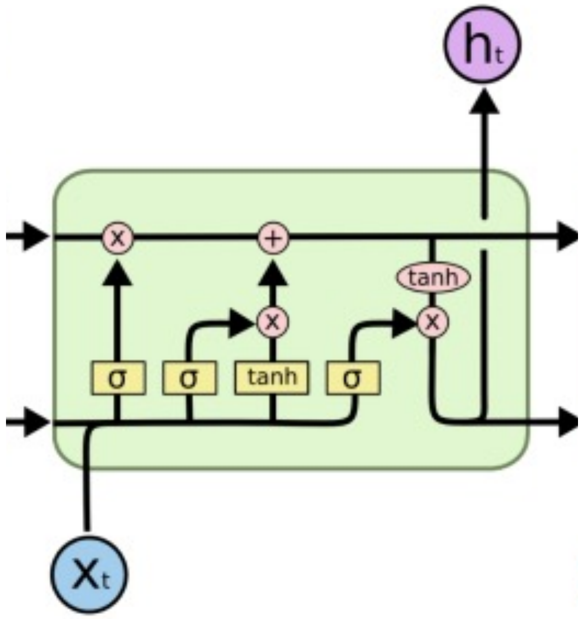
Gates are applied using element-wise (or Hadamard) product: $\odot$

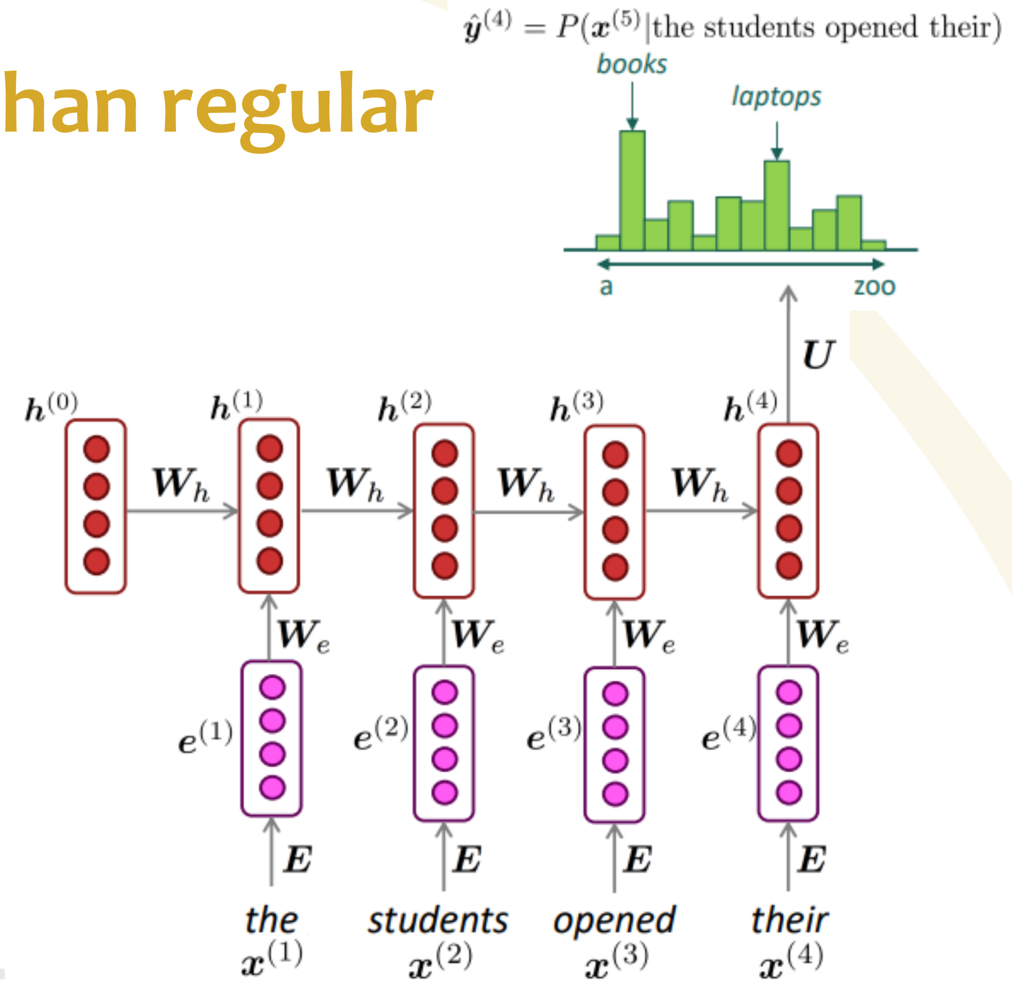
Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:

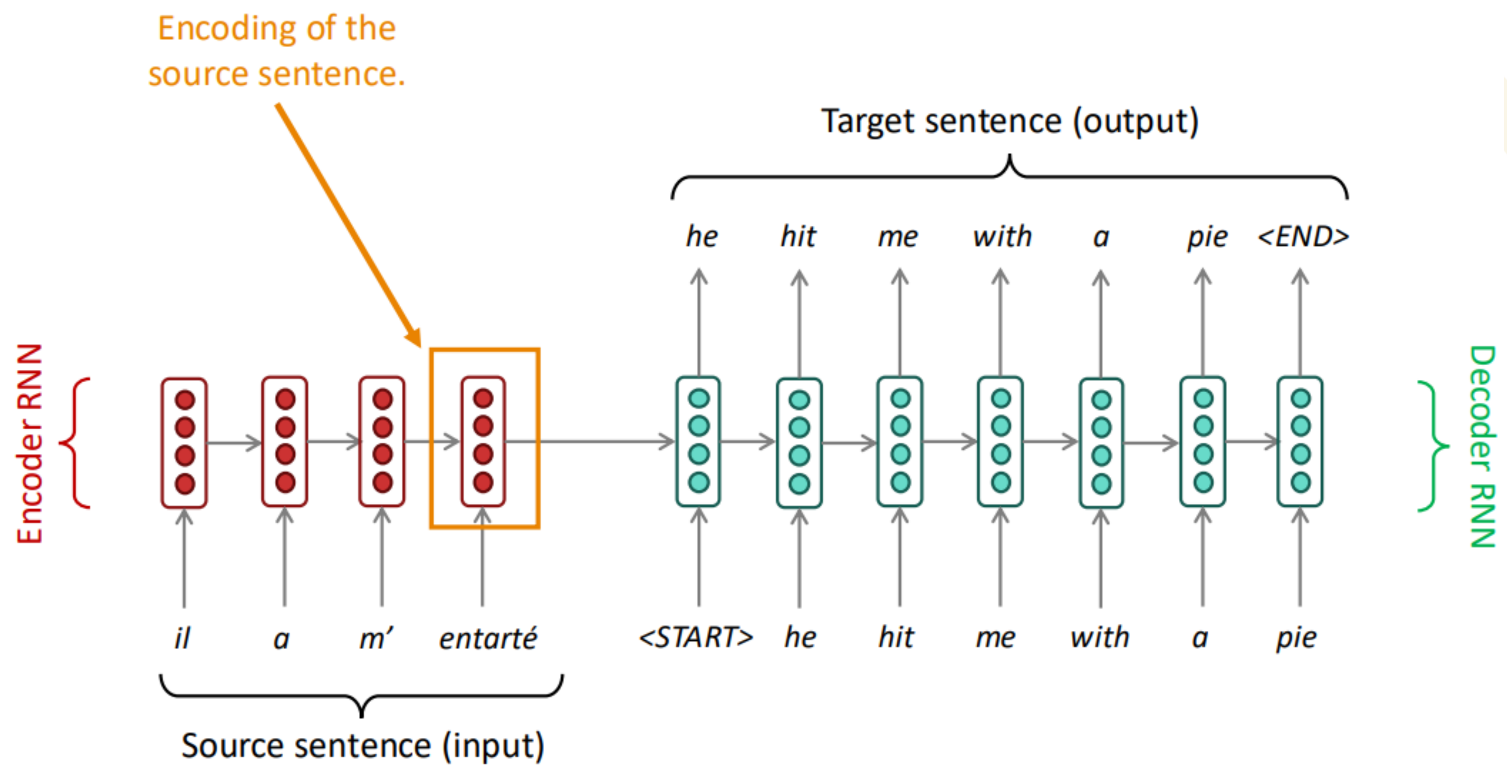


Why LSTM is better than regular

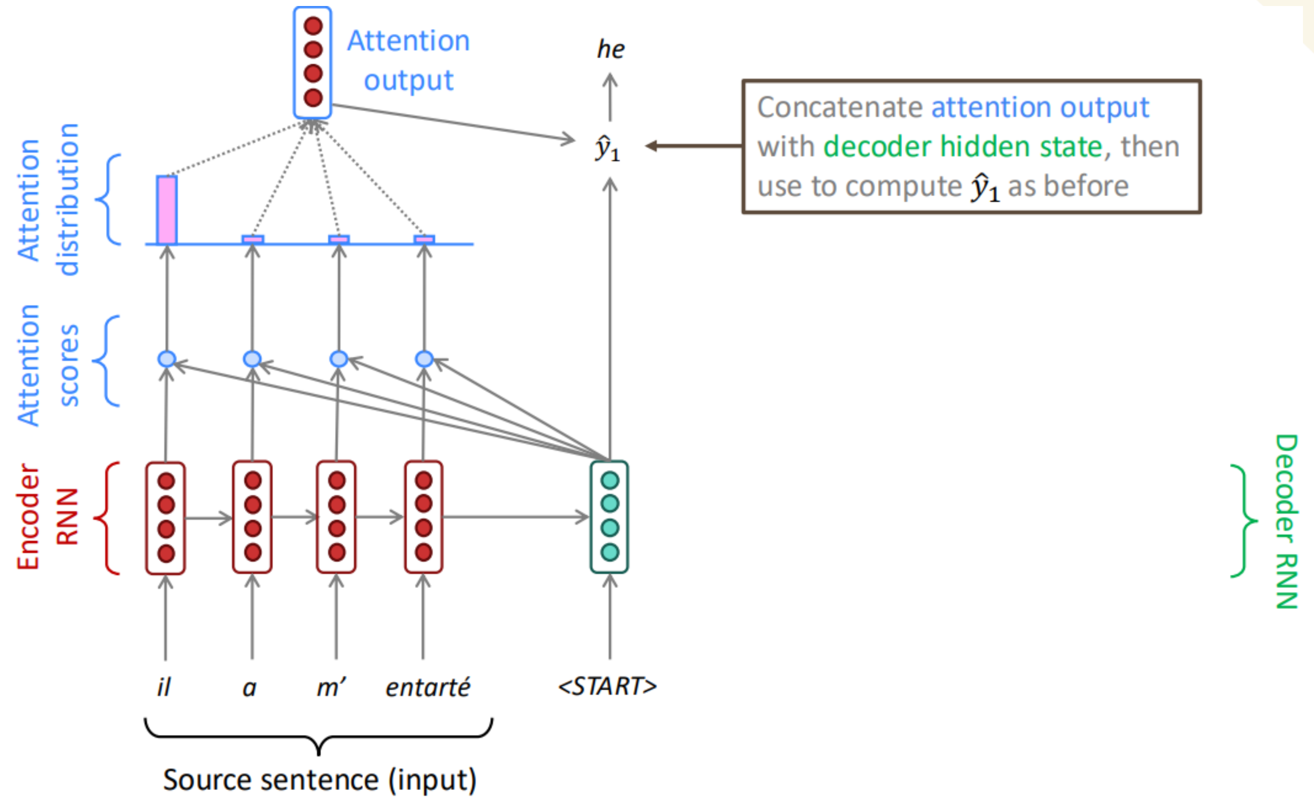




The bottleneck in RNN-based NMT



Sequence-to-sequence with attention



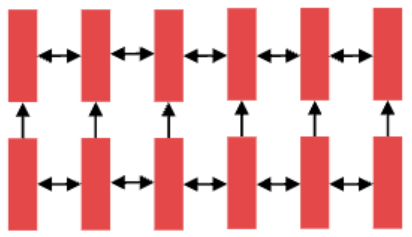
Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

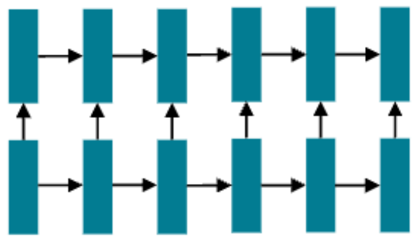
Lecture Plan

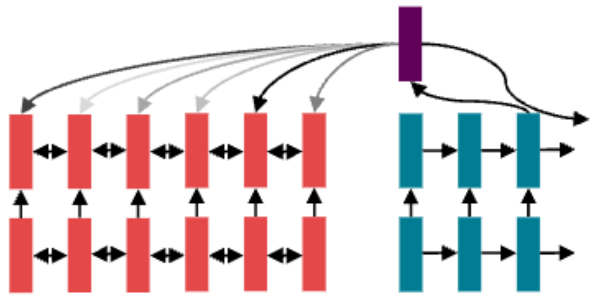
1. From recurrence (RNN) to attention-based NLP models
2. The Transformer model
3. Great results with Transformers
4. Drawbacks and variants of Transformers

In last few lectures: Recurrent models for (most) NLP!

- Around 2016, the de facto strategy in NLP is to encode sentences 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.

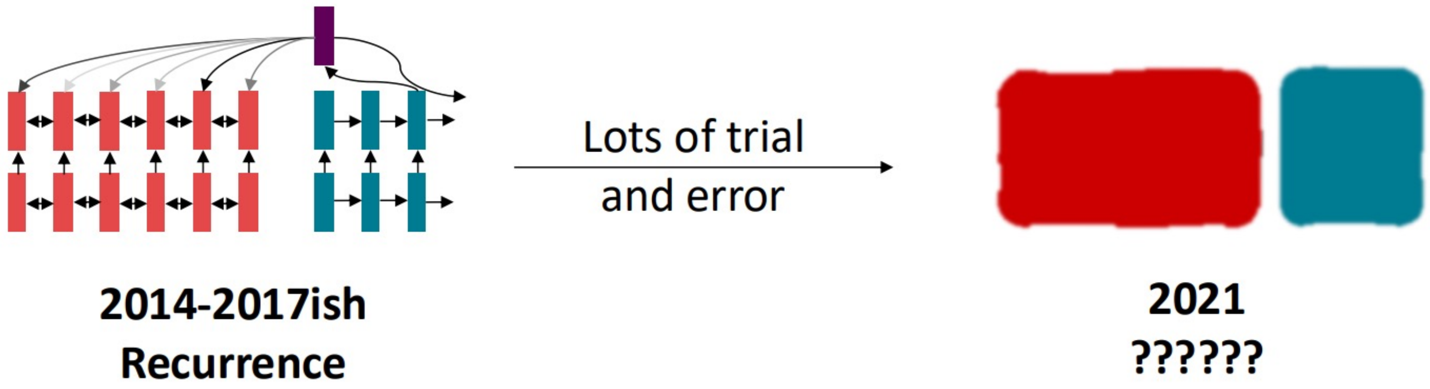






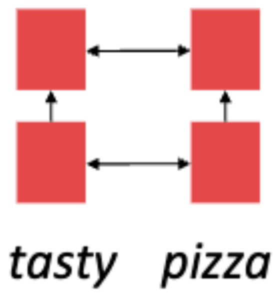
Do we even need recurrence at all?

- Abstractly: Attention is a way to pass information from a sequence (x) to a neural network input. (h_t)
- This is also exactly what RNNs are used for – to pass information!
- Can we just get rid of the RNN entirely? Maybe attention is just a better way to pass information!

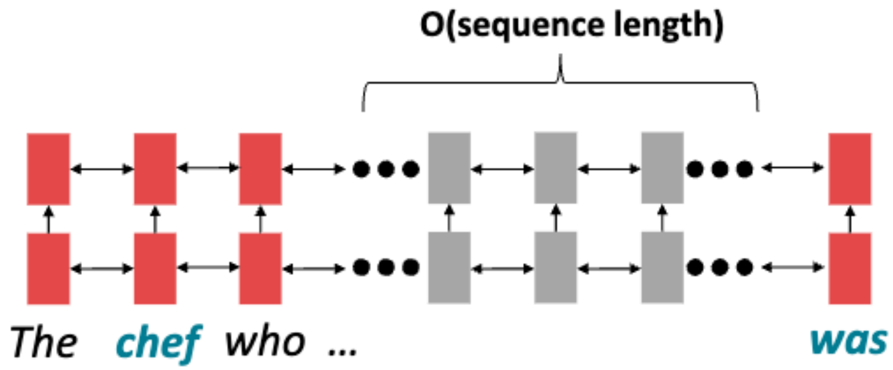


Issues with recurrent models: Linear interaction distance

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



- However, this also brings a problem:
- RNNs take O(sequence length) steps for distant word pairs to interact.



If not recurrence, how about word windows?

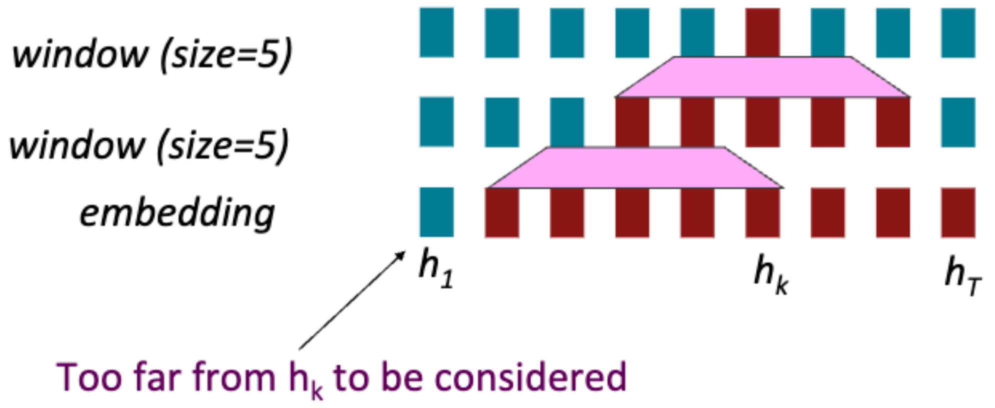
- **Word window models aggregate local contexts**
- Just like 1D convolution
- Number of unparallelizable operations does not increase sequence length

- Problem: What about long-distance dependencies?

If not recurrence, how about word windows?

- **Word window models aggregate local contexts**
- Just like 1D convolution
- Number of unparallelizable operations does not increase sequence length

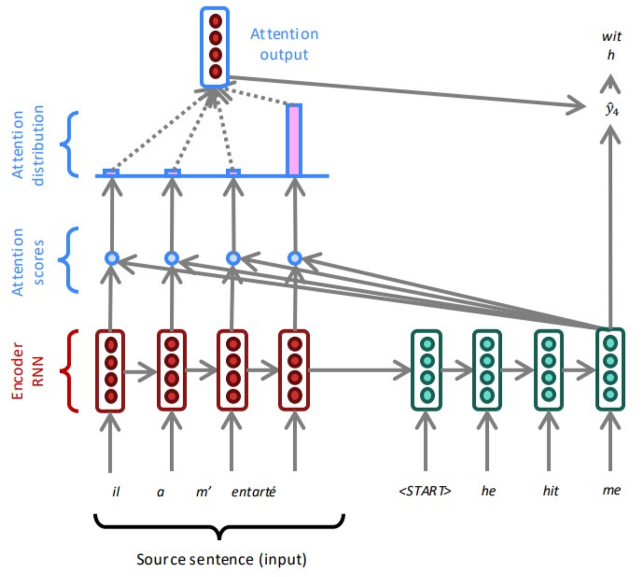
- Problem: What about long-distance dependencies?



Red states indicate those "visible" to h_k

The building block we need: self attention

- What we talked about
  – Cross attention: paying attention to the input x to generate y_t



- What we need
  – Self attention: to generate y_t , we need to pay attention to $y_{<t}$

Self-Attention: keys, queries, values from the same sequence

Let $\boldsymbol{w}_{1:n}$ be a sequence of words in vocabulary V. For each $\boldsymbol{w}_i$, let $\boldsymbol{x}_i = E\boldsymbol{w}_i$, where $E \in \mathbb{R}^{d\times|V|}$is an embedding matrix.

1. Transform each word embedding with weight matrices Q, K, V , each in $\mathbb{R}^{d\times d}$

$$\boldsymbol{q}_i = Q\boldsymbol{x}_i(\text{queries}) \qquad \boldsymbol{k}_i = K\boldsymbol{x}_i(\text{keys}) \qquad \boldsymbol{v}_i = V\boldsymbol{x}_i(\text{values})$$

2. Compute pairwise similarities between keys and queries; normalize with softmax

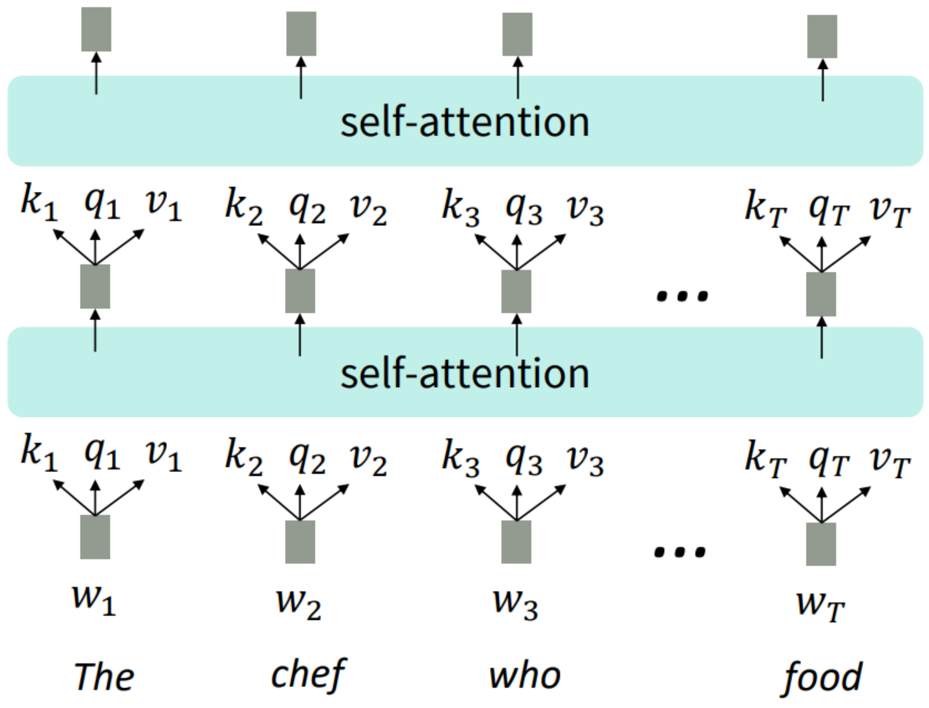
$$\boldsymbol{e}_{ij} = \boldsymbol{q}_i^T \boldsymbol{k}_j \qquad \boldsymbol{\alpha}_{ij} = \frac{exp(\boldsymbol{e}_{ij})}{\sum_{j'} exp(\boldsymbol{e}_{ij'})}$$

3. Compute output for each word as weighted sum of values

$$\boldsymbol{o}_i = \sum_j \boldsymbol{\alpha}_{ij}\boldsymbol{v}_i$$

Self-attention as an NLP building block

- In the diagram at the right, we have stacked self-attention blocks, like we might stack LSTM layers.
- Can self-attention be a drop-in replacement for recurrence?
  - ➢ No. It has a few issues, which we'll go through. First, self-attention is an operation on sets. It has no inherent notion of order.



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

Barriers and solutions for Self-Attention as a building block

Barriers		**Solutions**
1. Doesn't have an inherent notion of order!	$\longrightarrow$	**?**

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

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

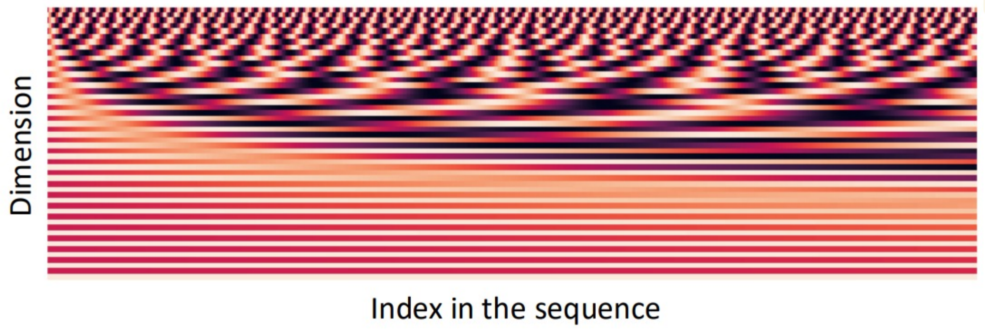
- Don't worry about what the $\boldsymbol{p}_i$ are made of yet!
- Easy to incorporate this info into our self-attention block: just add the $\boldsymbol{p}_i$ to our inputs!
- Recall that $\boldsymbol{x}_i$ is the embedding of the word at index i.

The positioned embedding: $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \boldsymbol{p}_i$

Position representation vectors through sinusoids

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

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



Pros:

- Periodicity indicates that maybe "absolute position" isn't as important
- Maybe can extrapolate to longer sequences as periods restart!

Cons:

- Not learnable
- The extrapolation doesn't really work!

Position representation vectors learned from scratch

- Learned absolute position representations: Let all $\boldsymbol{p}_i$ be learnable parameters! Learn a matrix $\boldsymbol{p} \in \mathbb{R}^{\mathrm{d} \times \mathrm{n}}$, 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, … , n.
- Most systems use this!
  - This often leads to strong empirical results, especially when the training and test sequence lengths are similar.
- Sometimes people try more flexible representations of position:
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

Common, modern relative position embeddings

- **High level thought process**: a relative position embedding should be some $f(x, i)$ subject to:

$$\langle f(x,i), f(y,j)\rangle = g(x,y,i-j)$$

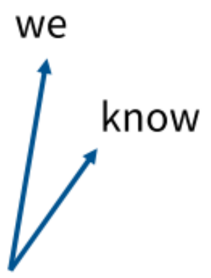
- That is, the attention score (dot product between query and key) only gets to depend on the relative position $(i - j)$.
- How do existing embeddings not fulfill this goal?
  1. **Sinusoidal**: the attention score has various cross-terms that are not just dependent on relative position (they depend on i or j, like sin(i)cos(j))
  2. **Absolute:** the positional embeddings are added to the input token embeddings, the resulting attention score will depend on individual position i or j, not the relative position.

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$

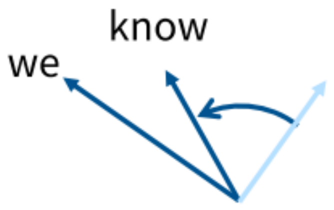
RoPE – Relative position embedding via rotation

How can we solve this problem?

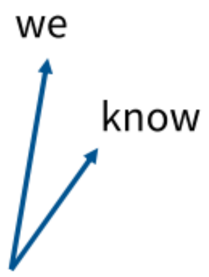
- We want our embeddings to be invariant to absolute position
- We know that inner products are invariant to arbitrary rotation



Position independent embedding



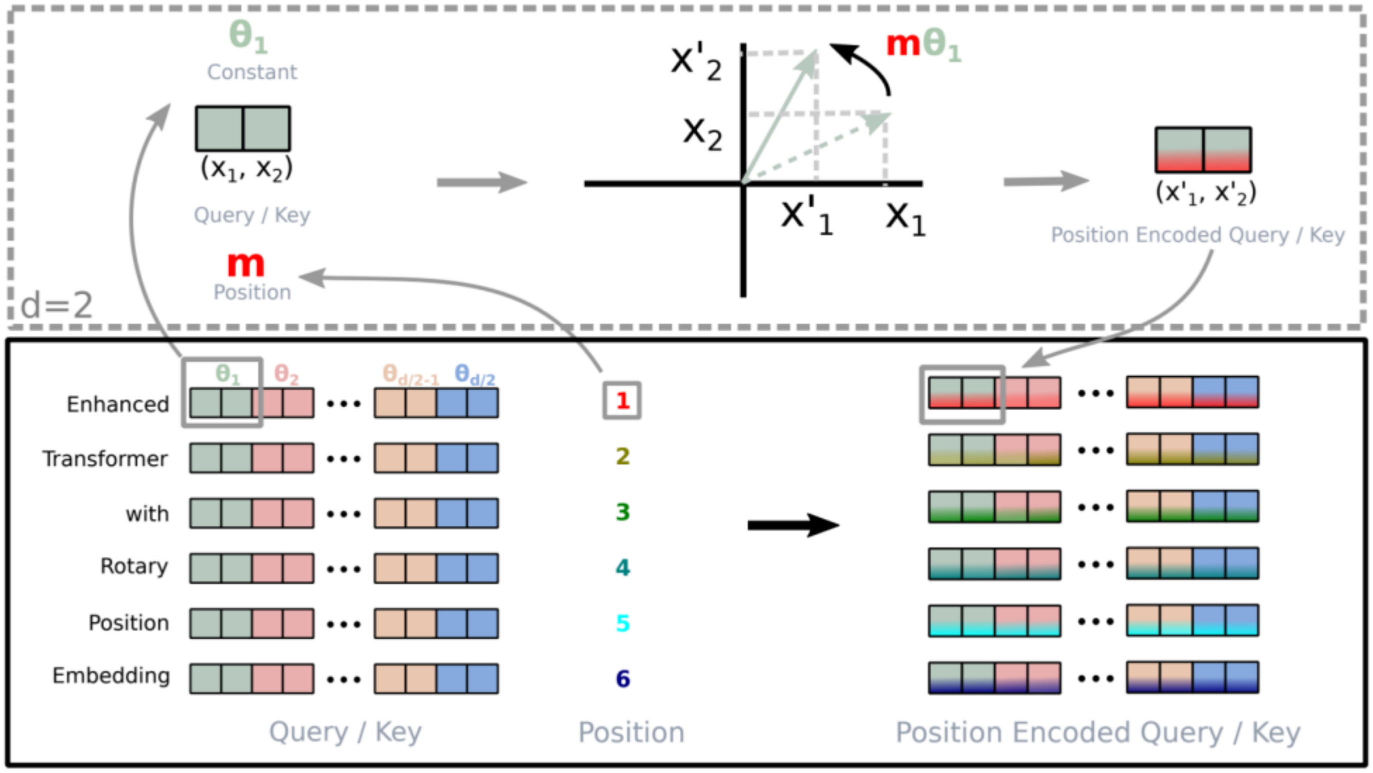
Embedding
“of course we know”
Rotate by ‘2 rotations’



Embedding
“we know that”
Rotate by ‘0 rotation’

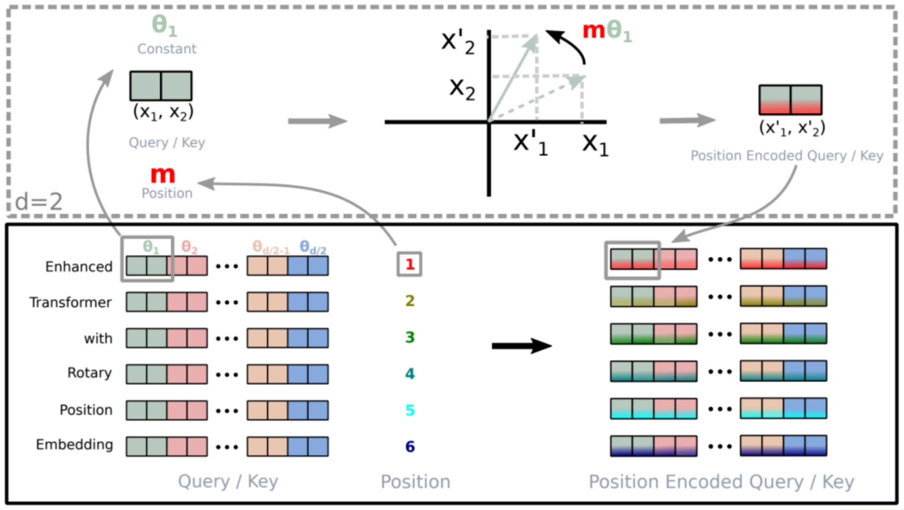
RoPE – From 2 to many dimensions

- Just pair up the coordinates and rotate them in 2d (motivation: complex numbers)



[Su et al 2021]

RoPE in equations



- Get the 2-D pairs
  - Each pair of dimensions (x_{2k}, x_{2k+1}) forms a 2-D vector (or complex number): $z_k = x_{2k} + jx_{2k+1}$
- Apply position-dependent rotation
  - For position p and input token embedding x: $\mathrm{RoPE}(x, p) = R(p)x$
  - For each pair, RoPE is defined as $R_k(p) = \begin{bmatrix} \cos(p\theta_k) & -\sin(p\theta_k) \\ \sin(p\theta_k) & \cos(p\theta_k) \end{bmatrix}, \theta_k = 10000^{-2k/d}$
  - For each pair, this matrix transformation equals to a rotation: $z'_k = z_k e^{jp\theta_k}$
- Finally, apply RoPE to queries and keys embeddings: $Q'_i = R(i)Q_i, K'_j = R(j)K_j$
  - Then the attention uses: $(Q'_i)^\top K'_j = Q_i^\top R(i-j)K_j$
  - Attention depends only on relative distance $(i-j)$

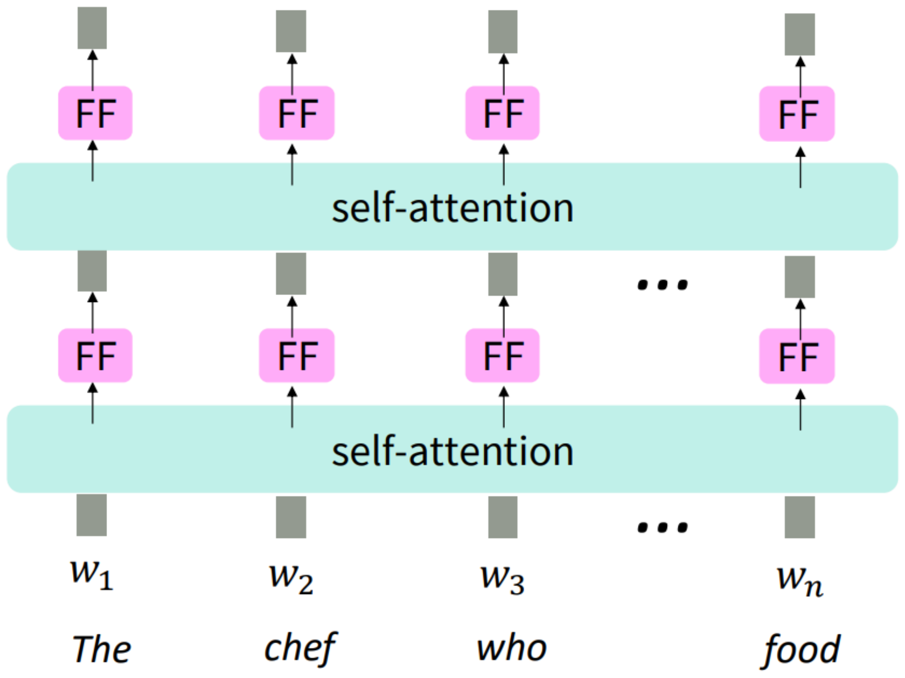
Barriers and solutions for Self-Attention as a building block

Barriers		**Solutions**
1. Doesn’t have an inherent notion of order!	$\longrightarrow$	1. Add position representations to the inputs.
2. No nonlinearities for deep learning magic! It’s all just weighted averages.	$\longrightarrow$	**?**

Adding nonlinearities in self-attention

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

$$\begin{aligned} m_i &= MLP(\text{output}_i) \\ &= W_2 * \text{ReLU}(W_1 \, \text{output}_i + b_1) + b_2 \end{aligned}$$



Intuition: the FF network processes the result of attention

Barriers and solutions for Self-Attention as a building block

Barriers

1. Doesn't have an inherent notion of order! →
2. No nonlinearities for deep learning magic! It's all just weighted averages →
3. Need to ensure we don't "look at the future" when predicting a sequence →
    - Like in machine translation
    - Or language modeling

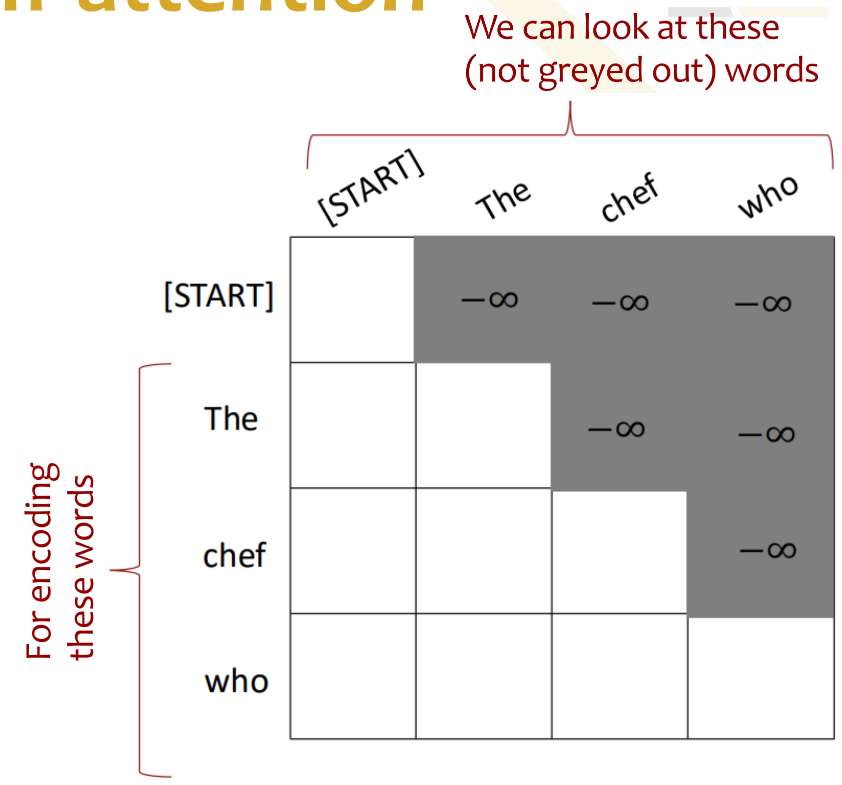
Solutions

1. Add position representations to the inputs
2. Easy fix: apply the same feedforward network to each self-attention output.
3. ?

Masking the future in self-attention

- To use self-attention in decoders, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of keys and queries to include only past words. (Inefficient!)
- To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$.

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



Barriers and solutions for Self-Attention as a building block

Barriers

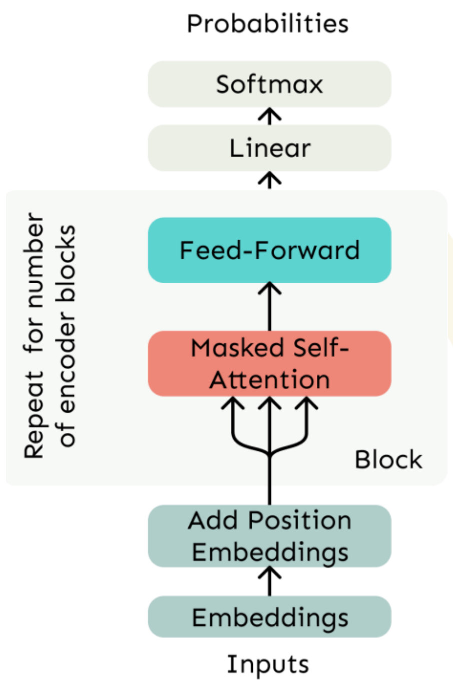
1. Doesn’t have an inherent notion of order!
2. No nonlinearities for deep learning magic! It’s all just weighted averages.
3. Need to ensure we don’t “look at the future” when predicting a sequence.
    - Like in machine translation
    - Or language modeling

Solutions

1. Add position representations to the inputs.
2. Easy fix: apply the same feedforward network to each self-attention output.
3. Mask out the future by artificially setting attention weights to zero.

Necessities for a self-attention building block

1. Self-attention:
   - The basis of the method.
2. Position representations:
   - Specify the sequence order, since self-attention is an unordered function of its inputs.
3. Nonlinearities:
   - At the output of the self-attention block.
   - Frequently implemented as a simple feedforward network.
4. Masking:
   - To parallelize operations while not looking at the future.
   - Keeps information about future from "leaking" to the past.


Probabilities
Softmax
Linear
Repeat for number of encoder blocks
Feed-Forward
Masked Self-Attention
Block
Add Position Embeddings
Embeddings
Inputs