Agenda:

→ Intro to sequence modeling

→ Brief overview of RNNs

→ Overview of Transformers and self Attention

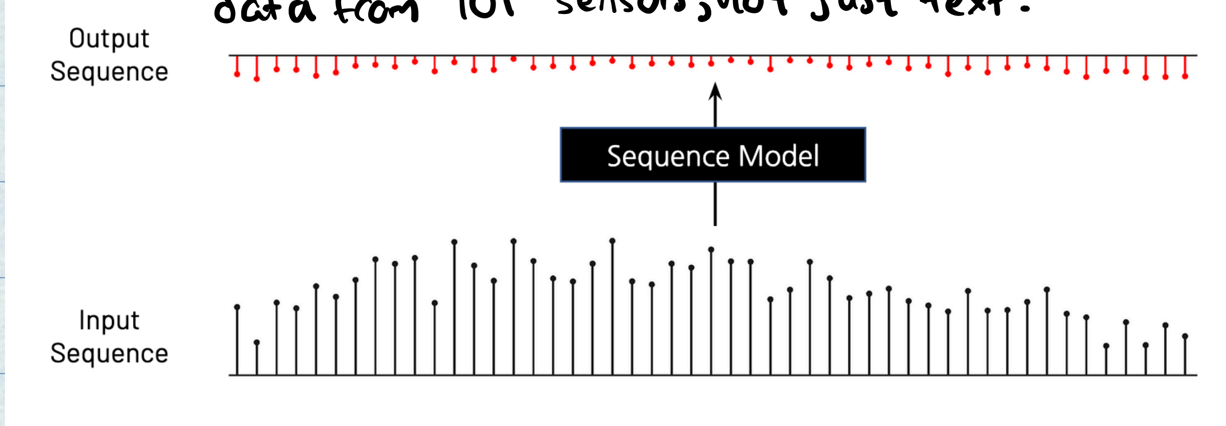
→ Multihead Attention

→ revisiting Tensor model parallelism &
megatron-LM.

*Intro to Sequence Modeling

- our new powerful LMs can.
  1) summarize large volumes of text
  2) Answer questions about world
  3) carry out long conversations
- in general we can think of these tasks as "sequence modeling" - mapping input sequences to output sequences:

- can also be useful for: audio, video, images, eks/mri signals, data from IOT sensors; not just text!



* Today: sequence models for language

INPUT seq	LM	Output seq
"The cat in the "		"Hat is a book by Dr. Seuss"
"turn off the "		"lights"

→ model needs:

- world knowledge
- common sense
- linguistic knowledge (english)

* A LM is a compact probability distribution over sequences of tokens

→ model has access to vocab w/ tokens $x_1 \dots x_L$

→ model assigns sequence of tokens some probability in 0..1

→ suppose vocab is: [lights, off, turn, the]:

$P(\text{"turn off the lights"}) \rightarrow .03$

$P(\text{"off turn the lights"}) \rightarrow .002$

... and so on

* we could model a JOINT DISTRIBUTION:

- choose some length L

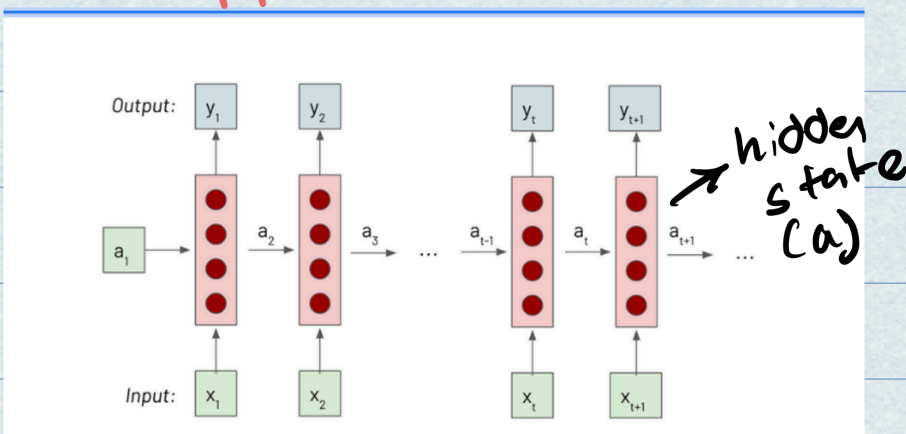
- have LM learn joint distribution of tokens over variables $x_1, x_2 \dots x_i$:

→ p("I like to eat cake")
→ p("The movie was very funny")
} for L = 5

*** BUT this has 3 cons:**

1) Ineffective - we typically want to sample sequences of different lengths
2) Too many options - as length increases, # of possible sequences grows (need to represent space)
3) Imprecise: probabilities will represent sequences "on average" → not best overall sequence

*** Another Approach: RNNs** - sota in 2016

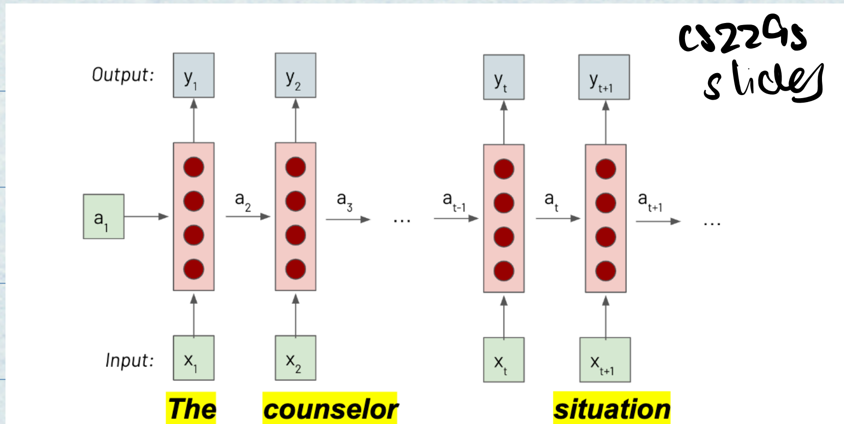


CS229S slides

- idea: capture "history" of all previously seen tokens to predict next token
- update HIDDEN STATE ($a_1, a_2 \dots a_{t+1}$):
  - use H.S. to generate output tokens

*** challenge 1: modeling long-range dependencies**

"The counselor helped frame the situation"



cs224s slides

"The" has gone through many iterations by getting to situation

*** challenge 2: Difficult to train!**

- recall: backprop needs to go through many layers
- easy to get into a situation where model becomes UNSTABLE or STOPS learning

  UNSTABLE: gradients too large

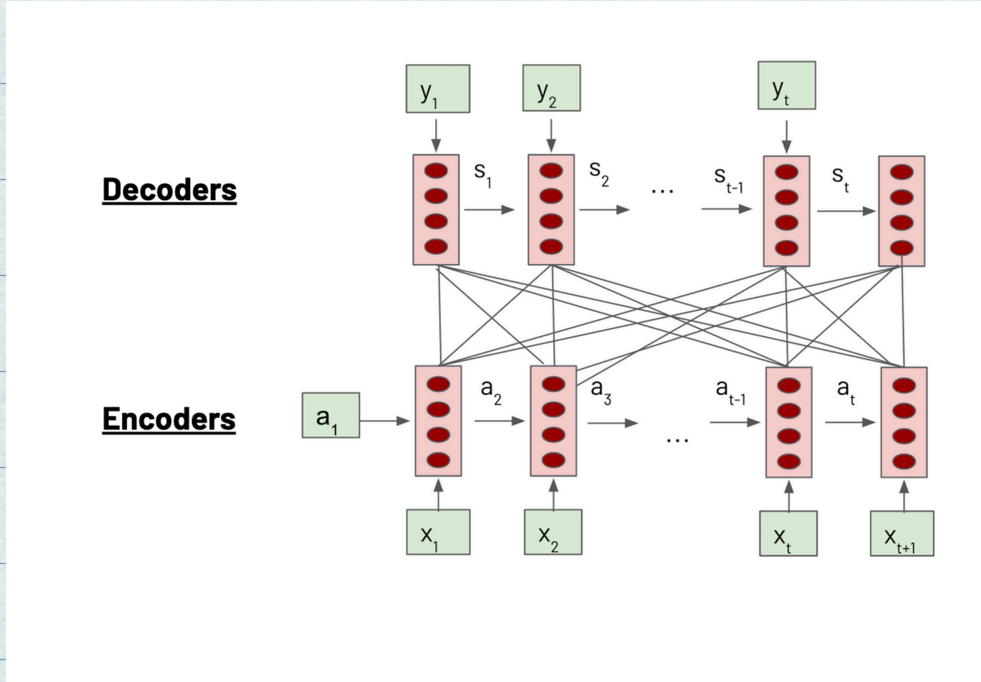
  STOPS learning: gradients too small

*** challenge 3: FW / BW passes CAN'T be parallelized:**

- each timestep needs to be processed before we move onto next step

* RNNs + Attention made some progress

• idea: all tokens should interact w/ representation of other tokens (helps w/ challenge 1)



* current mainstream approach: TRANSFORMERS

↓

1) uses attention idea
2) parallelizable
3) in last 7 years- we've gotten better at training

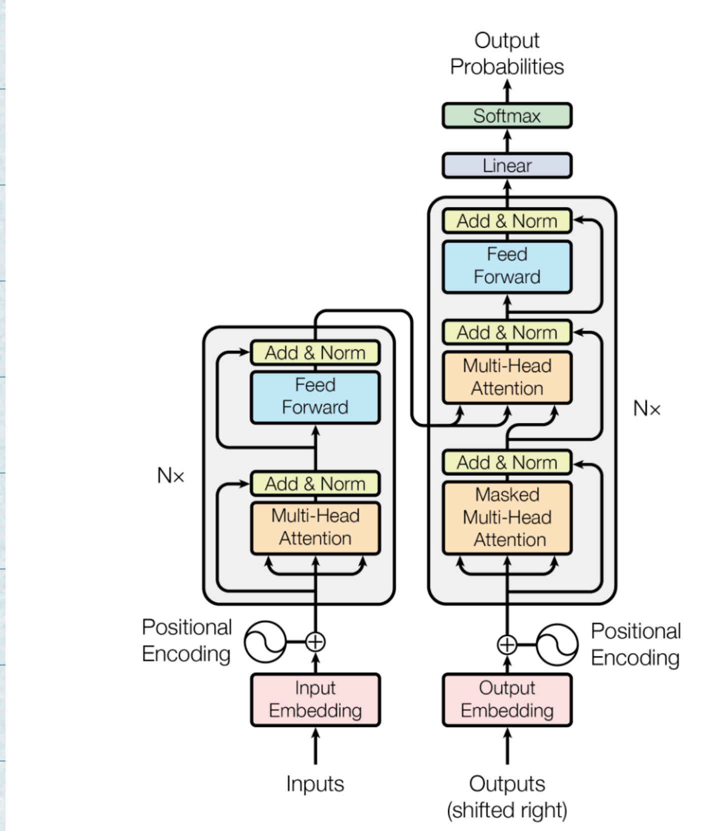
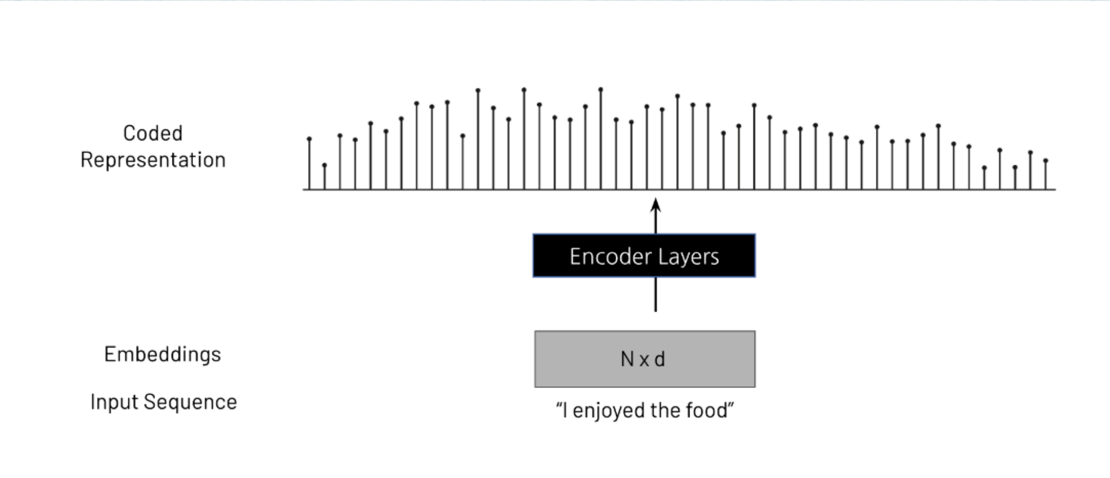


Figure 1: The Transformer - model architecture.

* from original google paper on transformers - google et. al.

*Before we get into specifics, we can use transformers in a few dif. ways:

• Encoder only



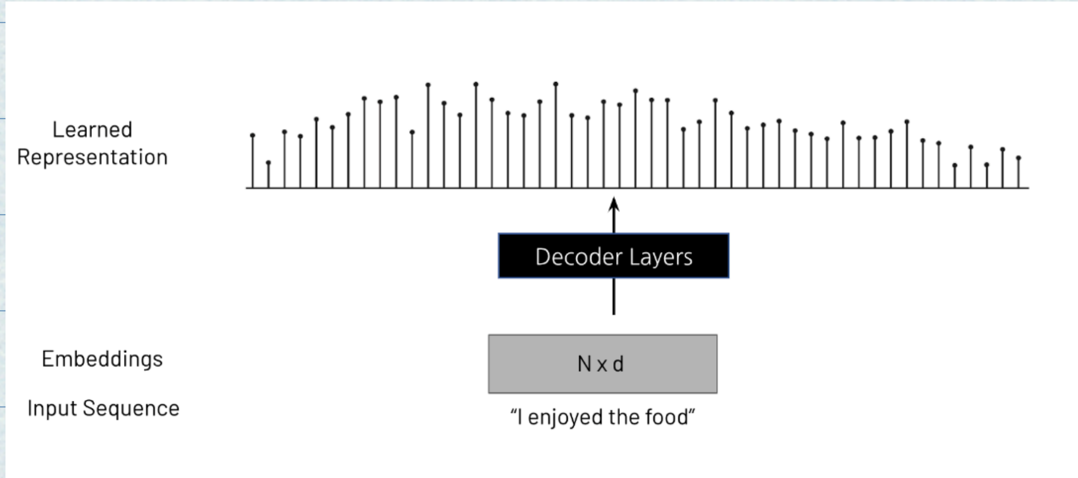
example:

BERT

- purpose: use final representation to perform classification (e.g., sentiment analysis) or get good "representations" of sequence → create embeddings of a document
- processes text bidirectionally
- training objective: span prediction - mask some words so model can predict

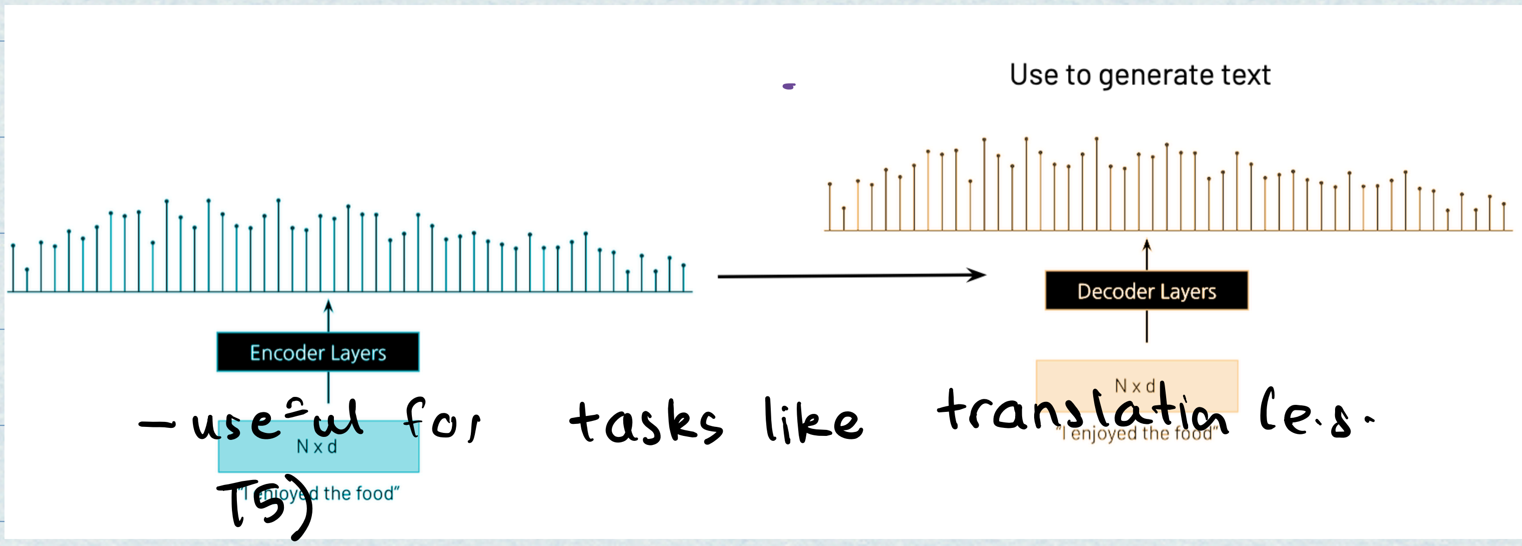
• Decoder only
  - purpose: use final representation to generate text
  - training objective: next token prediction
  - processes text in one direction (from left to

right):



example: GPT

• or both (encoder-decoder):

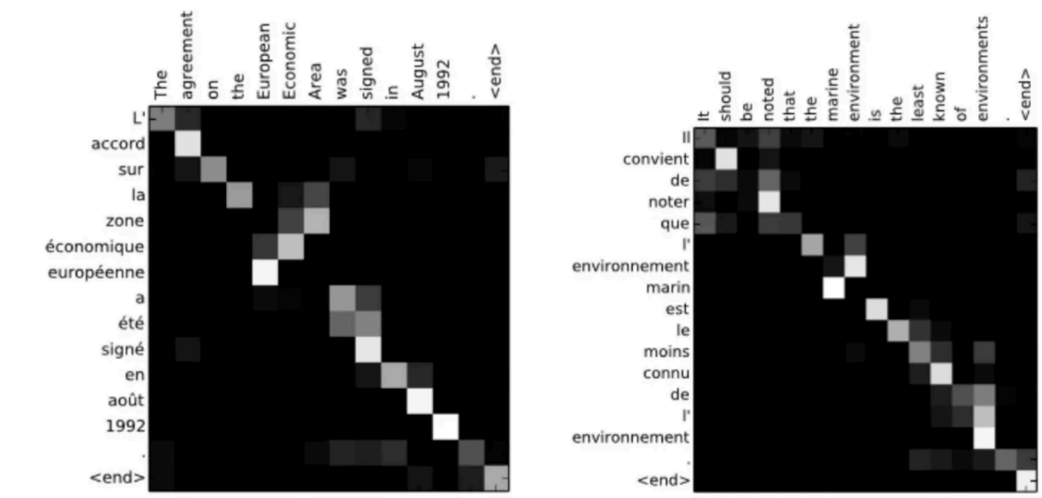


– useful for tasks like translation (e.g. T5)

* Building Intuition for Attention:

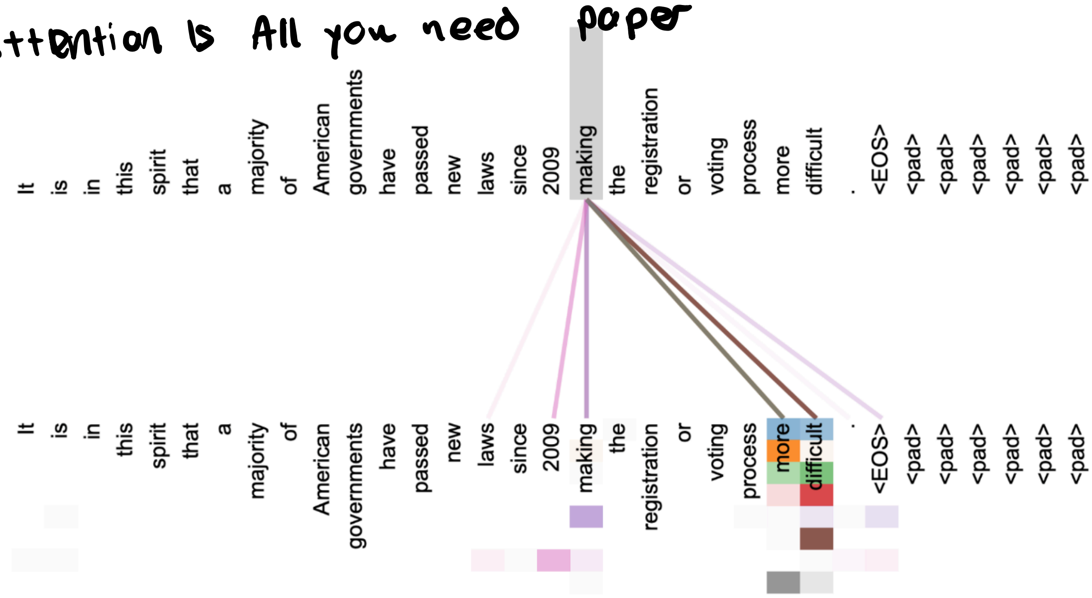
– The meaning of "frame" depends on context:
- "The counselor helped frame the situation"
- "I hung up the photo frame"

– Attention op computes how much each token is influenced by other tokens:



Bandanau et. al (Neural machine translation...)

Attention is All you need paper



Vaswani et al., 2017

*Let's try to break down self attention

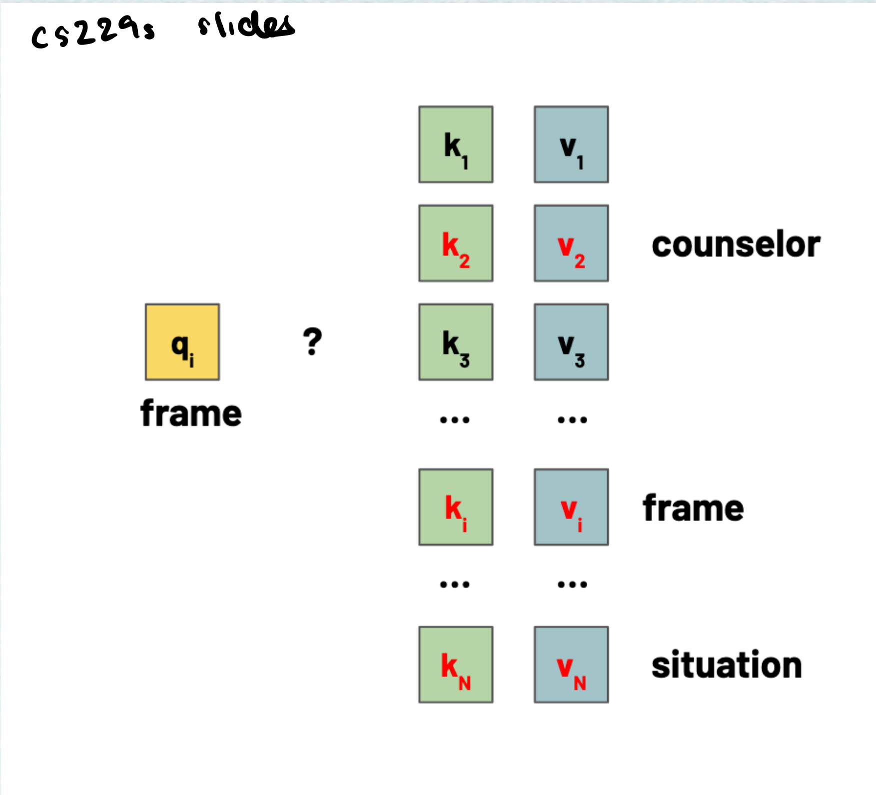
→ given a sequence $(x_1, x_2 \dots x_n)$
"the counselor helped frame the situation"
→ for each item x_i, compute how much it should "pay attention" to each value in the sequence

→ To do this, for each token x_i:

→ treat it as a "query token"

→ map entire sequence to "keys" w/ corresponding "values"

→ for query token, find all keys to pay attention to → and take those keys values



$x_i^{NEW} = 0.25\, v_2 + 0.45 v_i + 0.3\, v_N$

↓

create new version of x_i by combining values at some probability

- probabilities calculated w/ query, and all keys

✱ Full Attention Algorithm:

1) Transform each token x_i to set:
(multiply by specific pre-trained matrix)

q_i — query vector
k_i — key vector
v_i — value vector

2) Compute dot product between q_i and all k_i's in sequence:

$O = [q_i \cdot k_1, q_i \cdot k_2, q_i \cdot k_3, \dots q_i \cdot k_N]$

3) Pass O through softmax to set probabilities

$$S = \text{Softmax}(O)$$

4) Take weighted sum of values given softmax output:

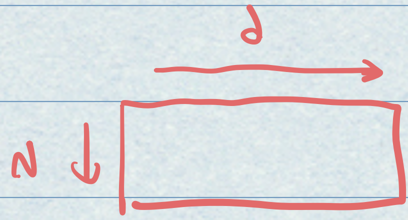
$$x_i^{new} = \sum_{i=1}^{N} s_i \cdot v_i$$

* This is great! We can compute in a batch:

N = sequence length / # of tokens in model input

d = model dimension

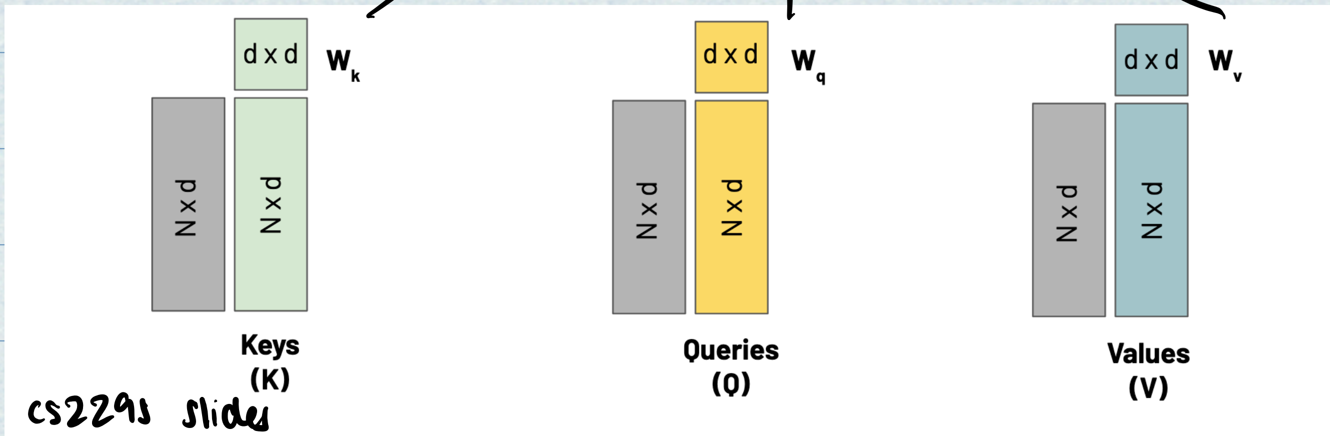
→ given sequence $(x_1 \dots x_n)$, model will get it as an $N \times d$ matrix of embeddings



* Attention step 1: compute all queries, keys, values

→ use matrix multiply to get query (Q), key (K) and value (V) matrices

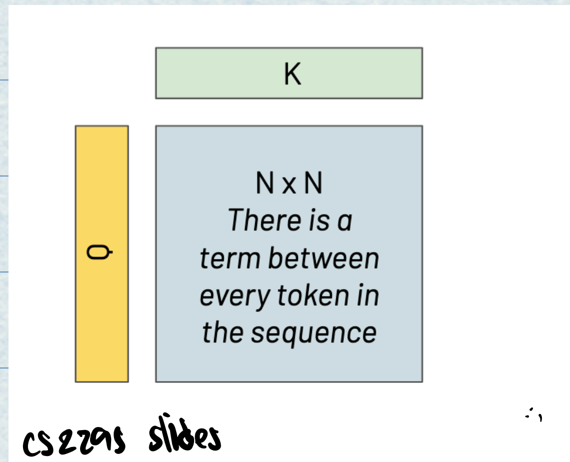
learnable params



2) Now how do we perform "lookup" and match queries to keys?

$Q = QK^T$

matrix multiply:

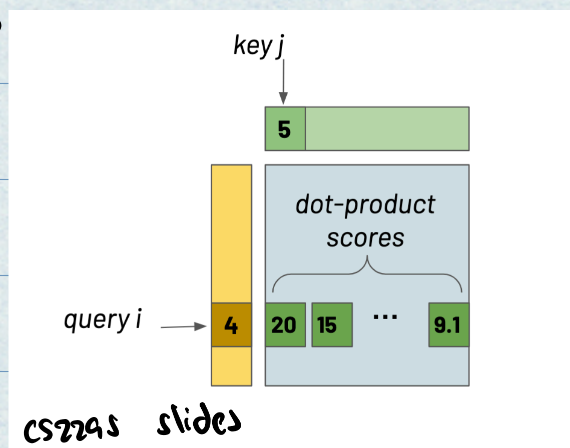


Note:

- Distance btwn any two tokens is now $O(1)$ vs. $O(N)$ in iterating through RNN
- Parallelizable: can compute pairwise interactions in parallel

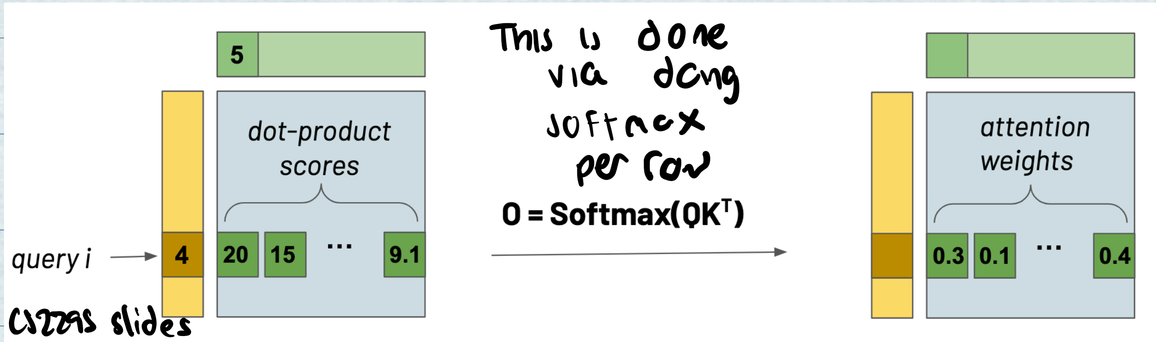
3) How do we pick the "best" key matches for our query?

→ Dot product results in "score" btwn each token pair $\langle i, j \rangle$



→ now for each query token i, we NORMALIZE dot product scores so they sum to 1

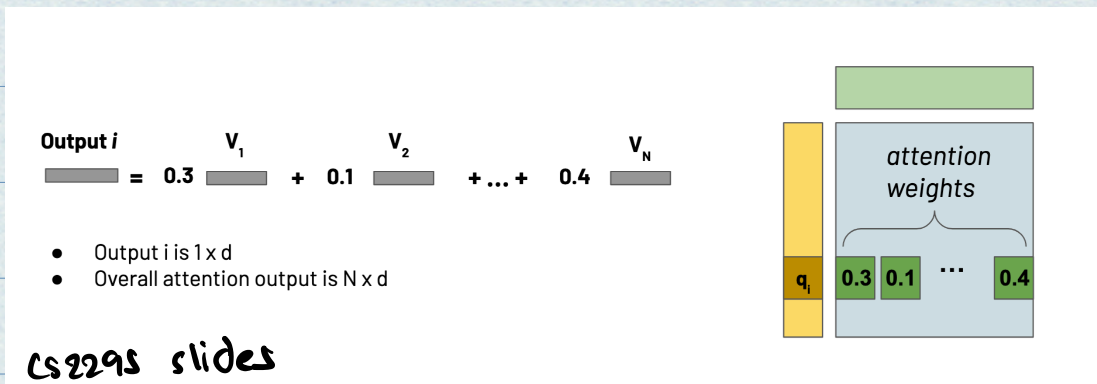
→ scores ≈ weights that reflect how much query "i" matches w/ key "j"



This is done via doing softmax per row

CS229S slides

→ now : take WEIGHTED sum of relevant tokens, where weights are these attention weights:

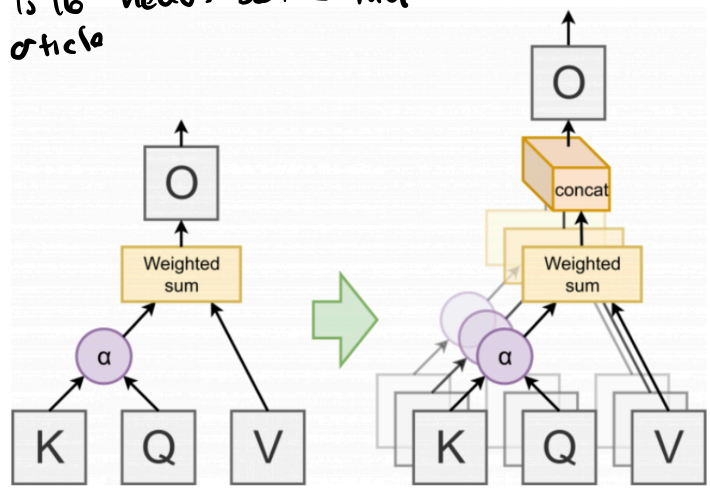


CS229S slides

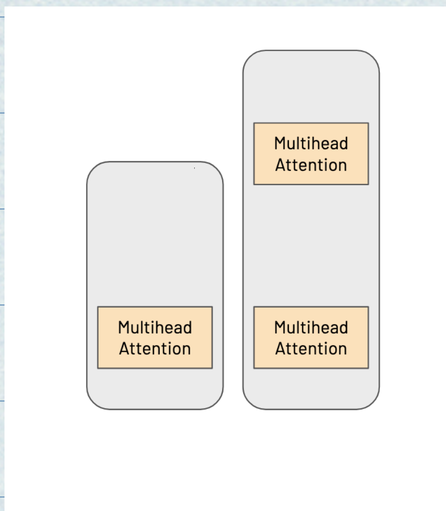
★ What is MULTIHEAD ATTENTION?

- each head operates on smaller # of dimensions
- if input is d-dimensional, and we have h heads, each attention op. is over d/h inputs

Is 16 heads better than one
article



* where are we?



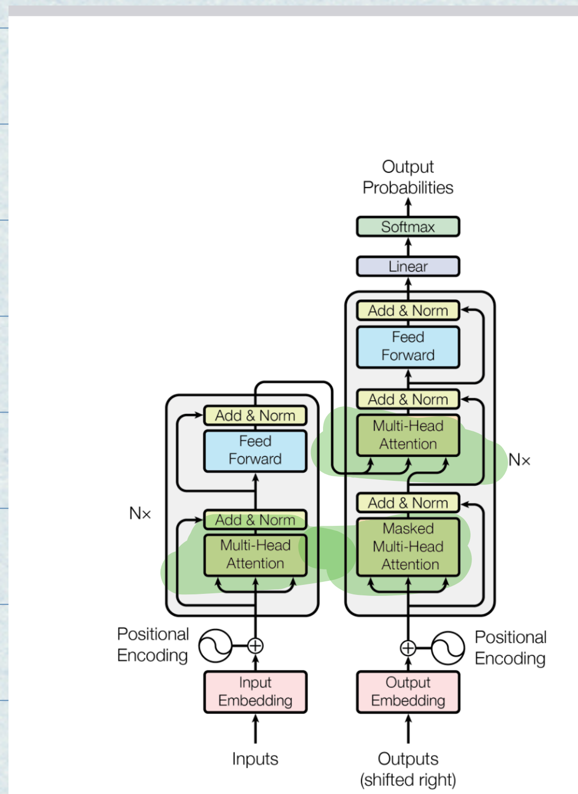


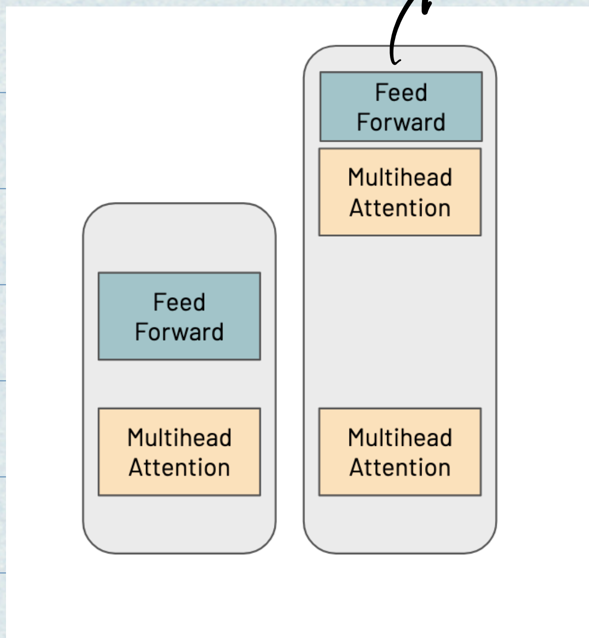
Figure 1: The Transformer - model architecture.

* ADD Feed-forward networks

→ so far: we haven't applied non-linearities;

just taken weighted averages

non-linearities improve expressivity



* use standard techniques to make architectures stable:

1) Residual connections:

Layer output = Attention output + Att. input

2) Layer normalization: normalize output to have zero mean / std 1 to keep scale manageable

3) scale dot product: divide attention weights by sqrt of model dim as values are too big otherwise:

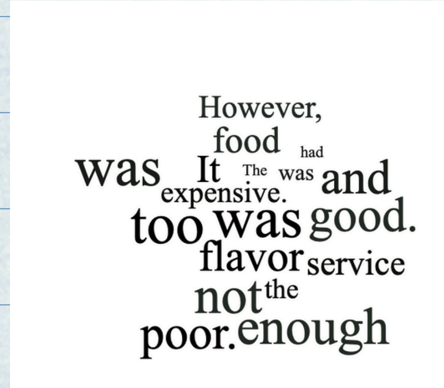
$$\text{Att. weights} = \frac{\text{softmax}(QK^T)}{\sqrt{d}}$$

Handling Position

→ problem: we don't know which token is in each position in our encoding!

The food was ***good****. It had enough flavor and was not too expensive. However, the service was* ***poor****.*

Good *refers to food and* ***poor*** *refers to service.*

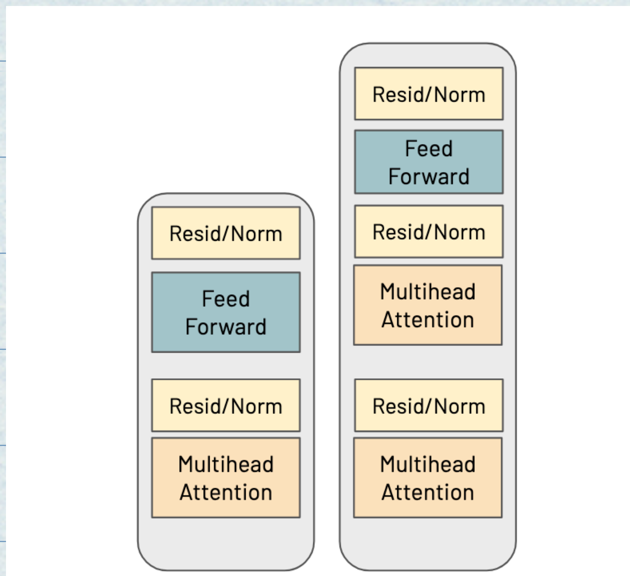


* solution: Add Position Encodings:

(before passing input through).

$$x_i = t_i + p_i$$

↳ could also concatenate
in practice they are added



⊕ Positional Encoding

⊕ Positional encoding

* output layer: (decoder)

- predict "next token" from full vocab size √

using output repr

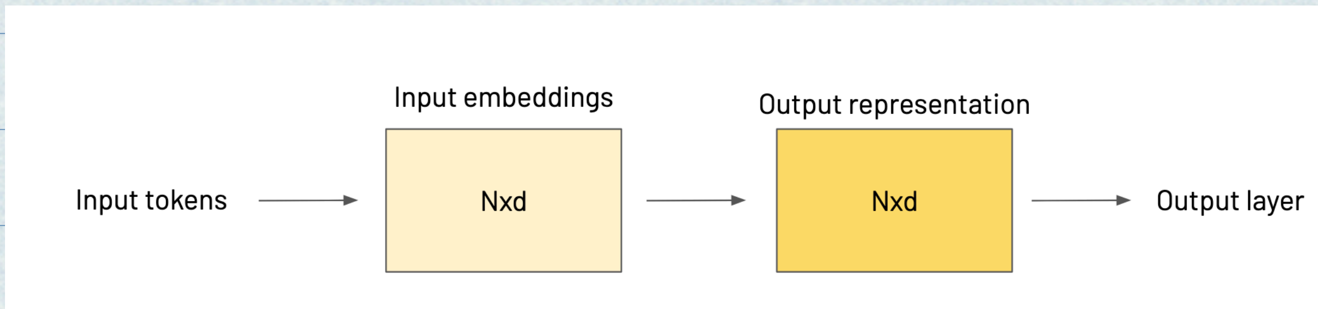
1) output layer w learned map from
$d \to V$ dimension.

↳ values associated w/ each vocab

↳ "logit"

2) Add softmax to get scores summing to 1

3) output token w/ HIGHEST softmax score



* revisiting Tensor model parallelism in context of transformers (megatron!)

→ feed forward part might include:

- two-layer multi-layer perceptron (MLP)
- MLP = 2 matrix multiplies (GEMM) + GeLU non-linearity

$$Y = \text{GeLU}(XA)$$

$$Z = \text{Dropout}(Y)$$

- Discuss: should we split A along columns or rows?

General lecture flow:
- Lecture 2 of CS229s 2023, given by Simran Arora: https://docs.google.com/presentation/d/1Pqu-TYLSnL9XNXF6BHlWahEtgNGNbm1lwQzpgxduJJc/edit#slide=id.g2863f73f1c2_0_0
- Transformers and Attention Lecture of CMU 15-442

Images:
- Sequence Modeling: Simran's slides, # 7
- RNN breakdowns and challenges: Simran's slides, 34-37
- Attention Figure: Attention is All You need Paper (Vaswani et. Al): https://arxiv.org/pdf/1706.03762
- Encoder, Decoder, Encoder-Decoder diagrams: slides 27-29 of simran's lecture
- Heatmap of attention: Neural Machine Translation by Jointly Learning to Align and Translate: https://arxiv.org/abs/1409.0473
- Self attention matrix diagrams: 44-55 of Simran's lecture
- Multi head attention diagram: Are 16 heads better than 1: https://blog.ml.cmu.edu/2020/03/20/are-sixteen-heads-really-better-than-one/
- Transformers Architecture Build Up: Slide 66 - end of Simran's lecture

Other references:
Megatron Paper: https://arxiv.org/pdf/2104.04473