CS 839: Foundation Models
Transformers, Attention, Subquadratic Architectures

Fred Sala

University of Wisconsin-Madison

Sept. 17, 2024

Announcements

- **Announcements:** Recordings available on Canvas (under Kaltura tab)
- Class roadmap:

Tuesday Sept. 17	Architectures II: Subquadratic Architectures
Thursday Sept. 19	Language Models I
Tuesday Sept. 24	Language Models II
Thursday Sept. 26	Prompting I
Tuesday Oct. 1	Prompting II

Mostly Language Models

Outline

- **Conclude Attention Discussion**
  - Notions of attention, self-attention, basic attention layer, QKV setup and intuition, positional encodings
- **Transformers**
  - Architecture, encoder and decoder setups
- **Subquadratic Models**
  - Basic ideas. Examples: S4, Mamba.

Self-Attention: Retrieval Intuition

- How should we design the interactions?
  - Analogy: **search**

`“Which restaurants near me are open at 9:00 pm?”`

Query

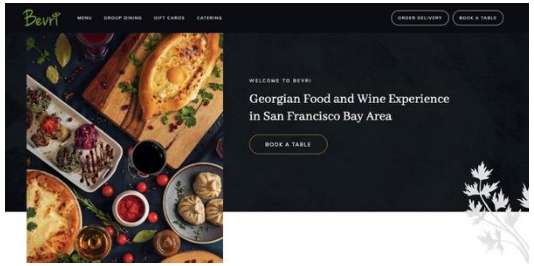
Key

Value

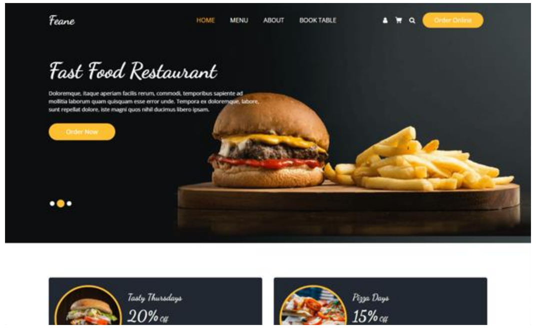
Objects:

Query
Key
Value

Score 0.3



Score 0.7



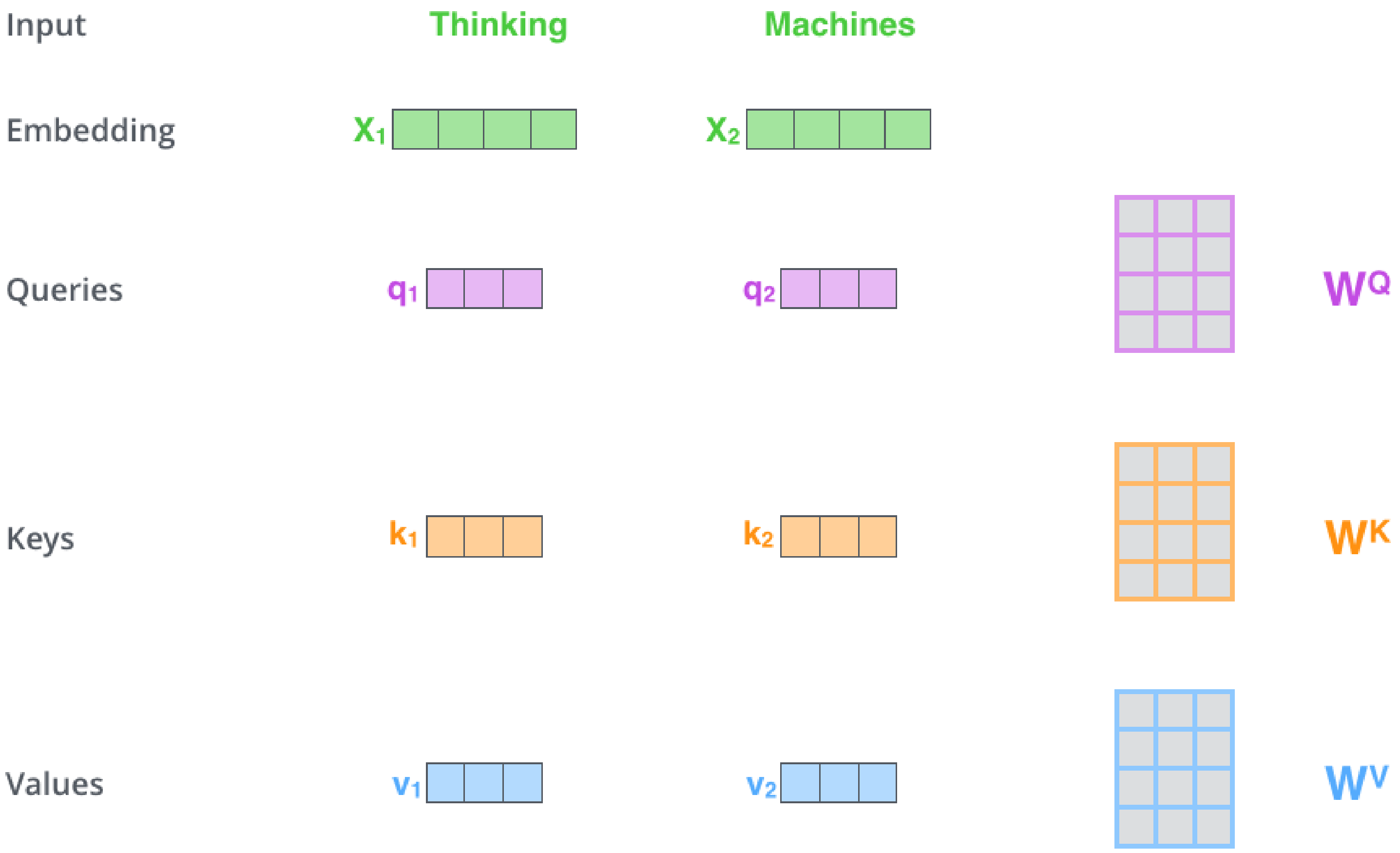
Self-Attention: Query, Key, Value Vectors

- *Transform* incoming word vectors,
- Enable *interactions* between words

- Get our **query, key, value** vectors via weight matrices: linear transformations!

Objects:

Query
Key
Value



Self-Attention: Score Functions

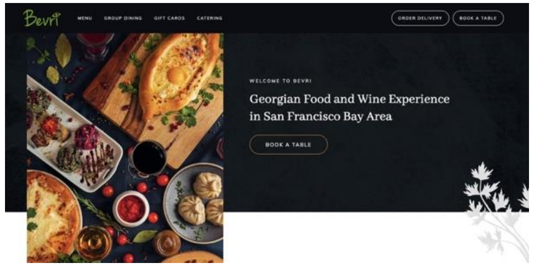
Have **query, key, value** vectors

- Next, get our **score**

q_1 k_2

Query Key

Score 0.3



- Lots of things we could do --- **simpler** is usually better!
- Dot product $q_1 \cdot k_2 = 96$

- Then we'll do **softmax**

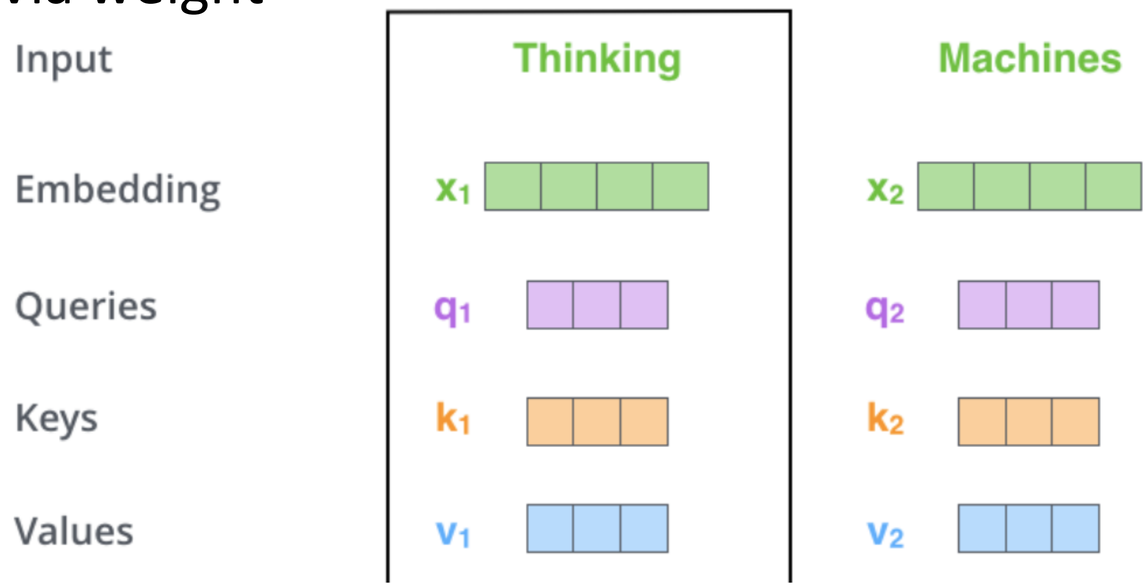
Self-Attention: Scoring and Scaling

- *Transform* incoming word vectors,
- Enable *interactions* between words

- Get our **query, key, value** vectors via weight matrices: linear transformations!
- Compute scores

Objects:

Query
Key
Value

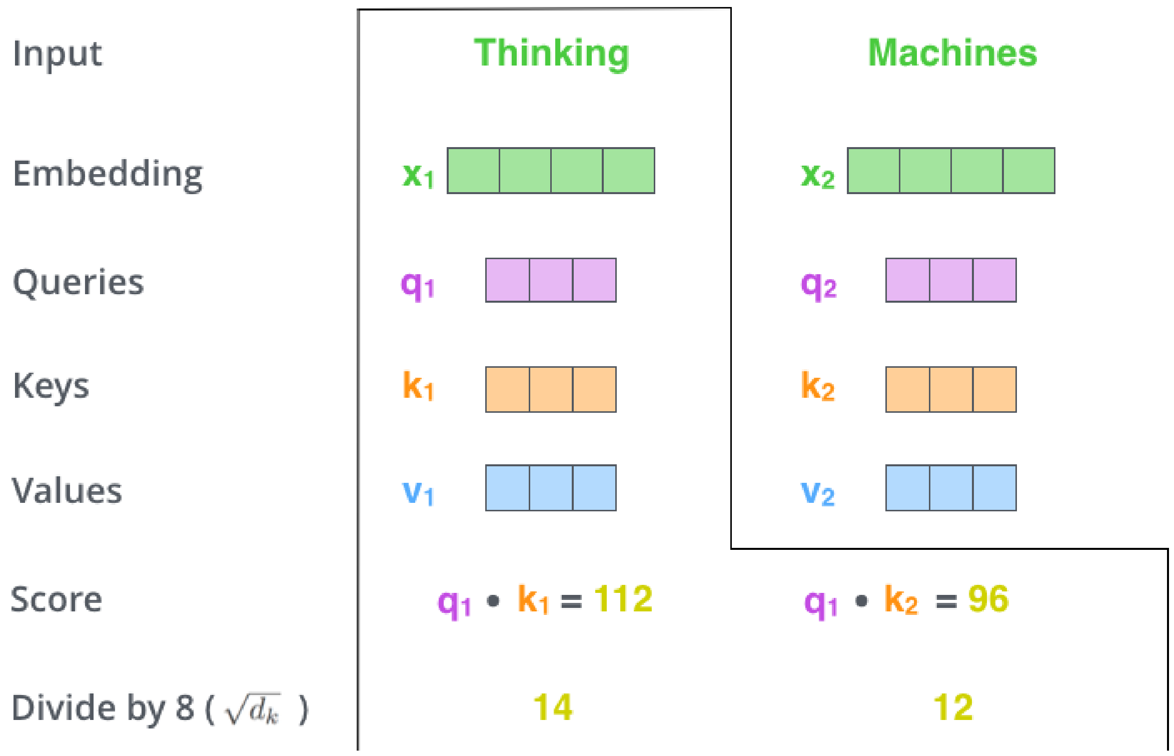


Self-Attention: Putting it Together

- Have **query, key, value** vectors via weight matrices: linear transformations!
- Have softmax score outputs (**focus**)
- Add up the values!

Objects:

Query
Key
Value



Self-Attention: Matrix Formulas

- Have **query, key, value** vectors via weight matrices: linear transformations!
- Have softmax score outputs (**focus**)
- Add up the values!

Objects:

Query
Key
Value

$$Q = XW_Q, K = XW_K, V = XW_V$$

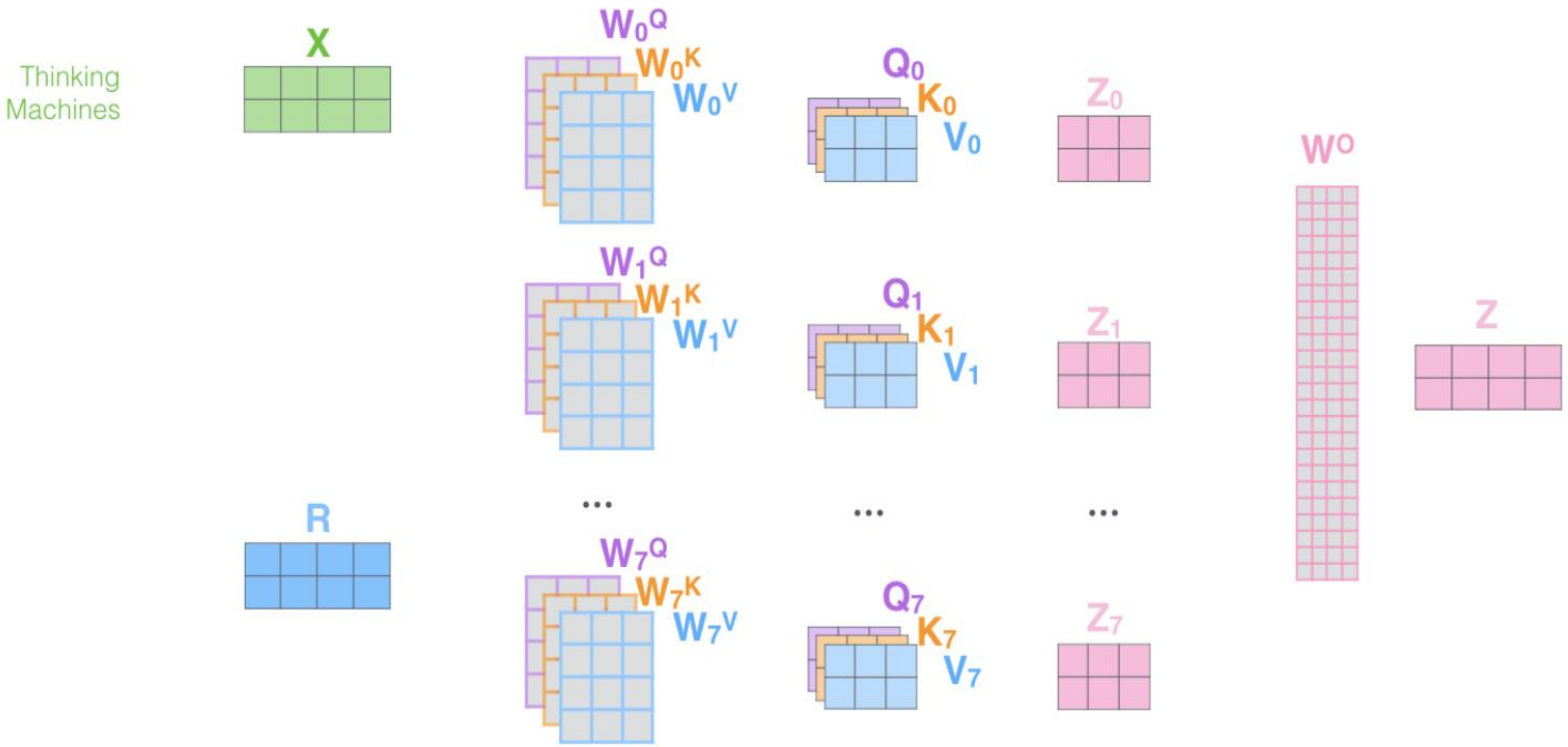
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(X\frac{W_Q {W_K}^T}{\sqrt{d_k}}X^T\right)V$$

Self-Attention: Multi-head

This is great but will we capture everything in one?

- Do we use just 1 kernel in CNNs? **No!**
- Do it many times in parallel: **multi-headed attention.** Concatenate outputs



Self-Attention: Position Encodings

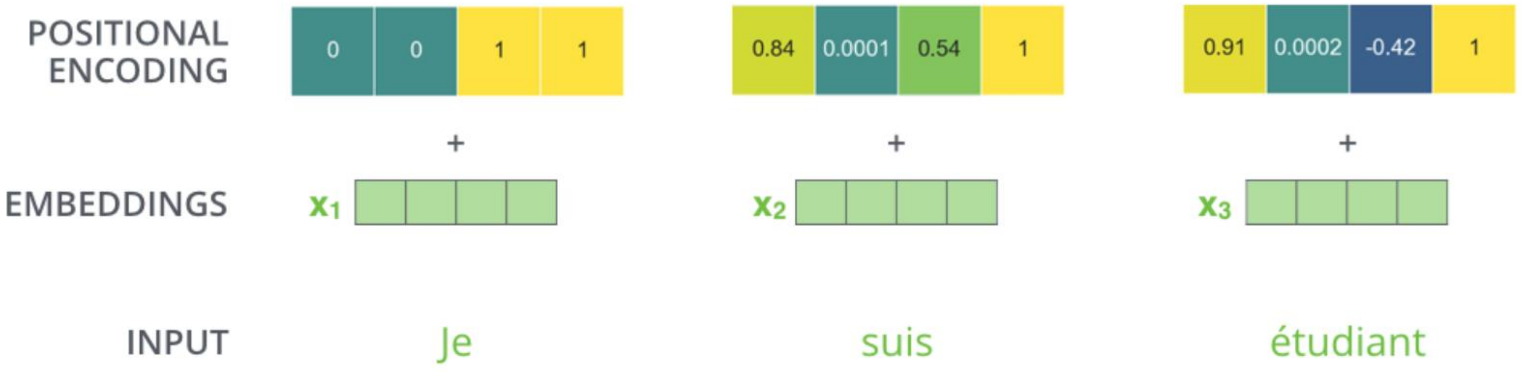
Almost have a full layer designed.

- One annoying issue: so far, order of words **(position) doesn't matter**!
- Solution: add positional encodings

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

↑ Component index

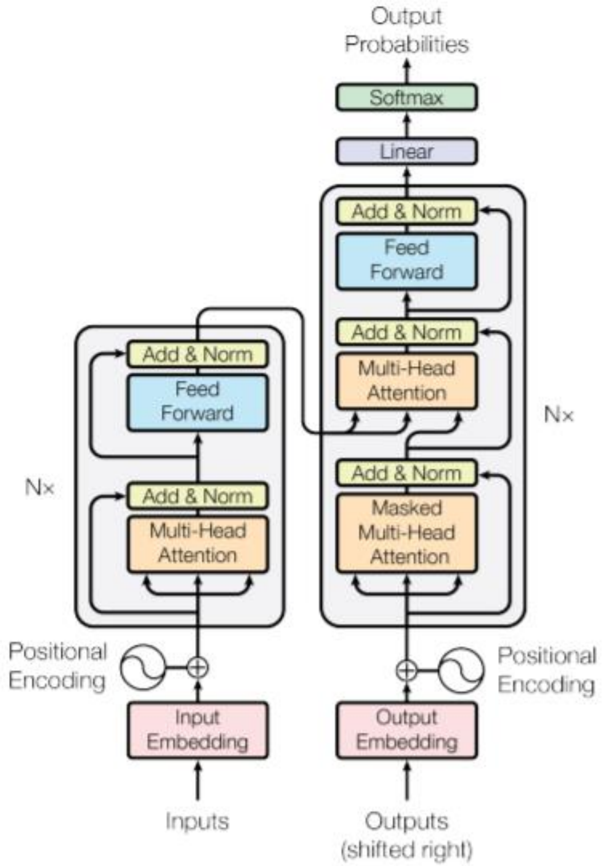


Break & Questions

Transformers: Model Architecture

- Initial goal for an architecture: **encoder-decoder**
  - Get **rid of recurrence**
  - Replace with **self-attention**

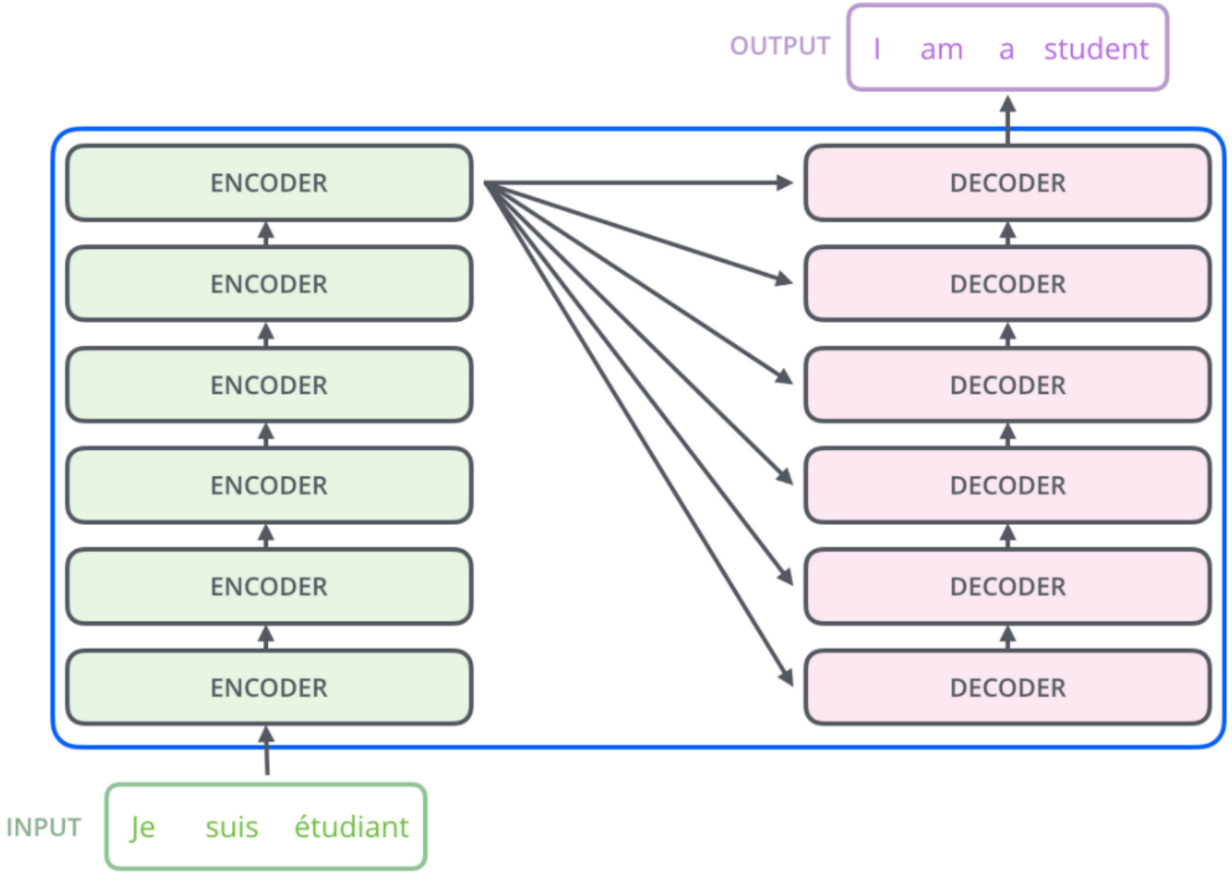
- Architecture
  - The famous picture you've seen
  - Centered on self-attention blocks



Vaswani et al. '17

Transformers: Architecture

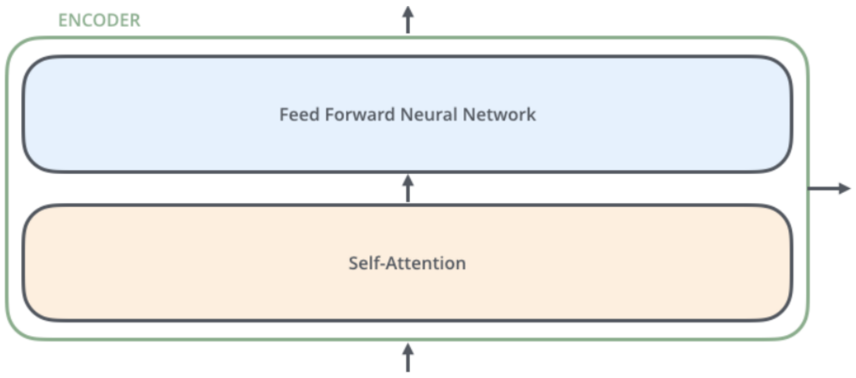
- **Sequence-sequence** model with **stacked** encoders/decoders:
  - For example, for French-English translation:

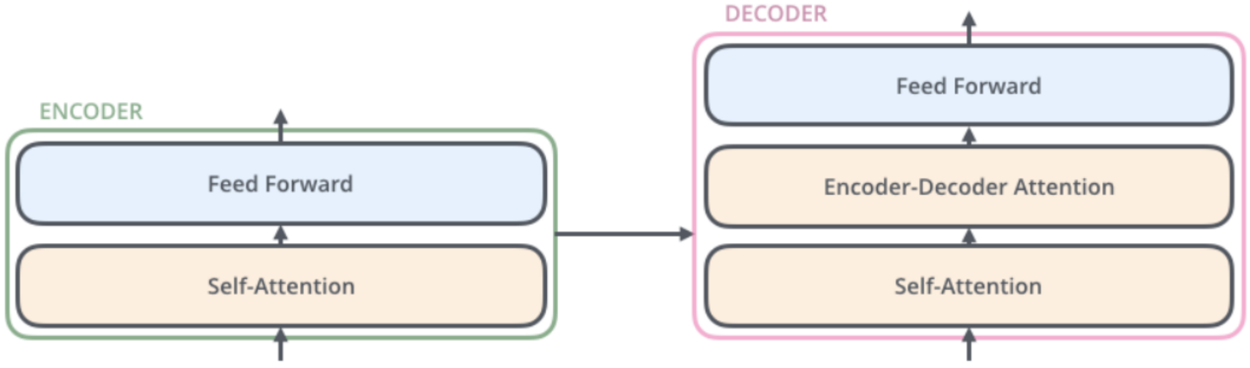


Transformers: Architecture

- Sequence-sequence model with **stacked** encoders/decoders:
  - What's inside each encoder/decoder unit?

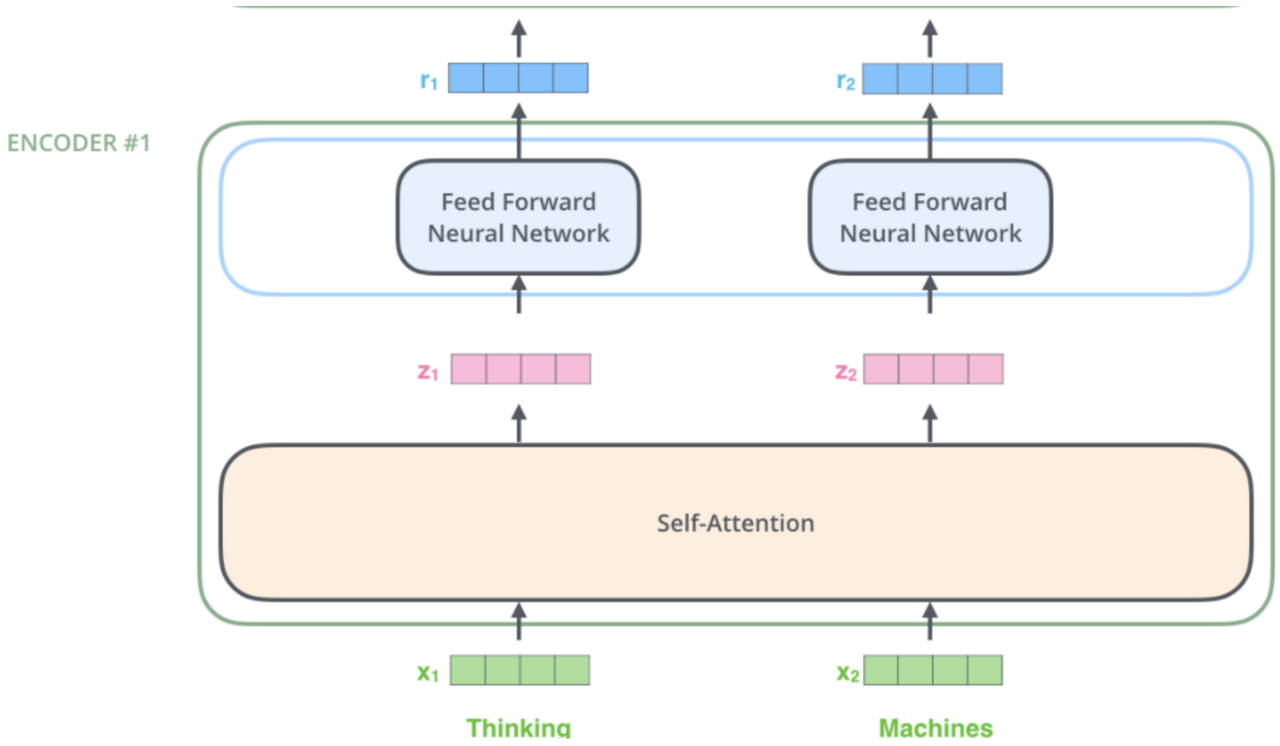
  - Focus encoder first: **pretty simple**! 2 components:
    - Self-attention block
    - Fully-connected layers (i.e., an MLP)





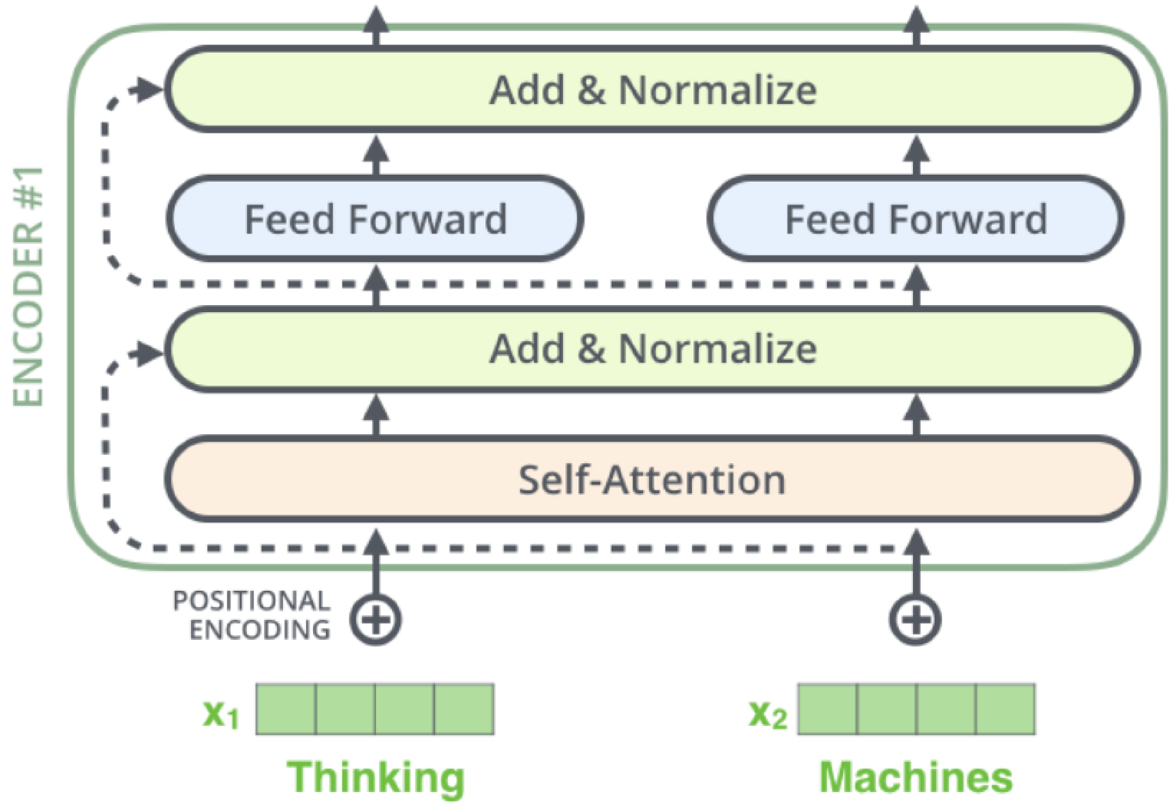
Transformers: Inside an Encoder

- Let’s take a look at the encoder. Two components:
  - 1. **Self-attention** layer (covered this)
  - 2. “Independent” **feedforward nets** for each head



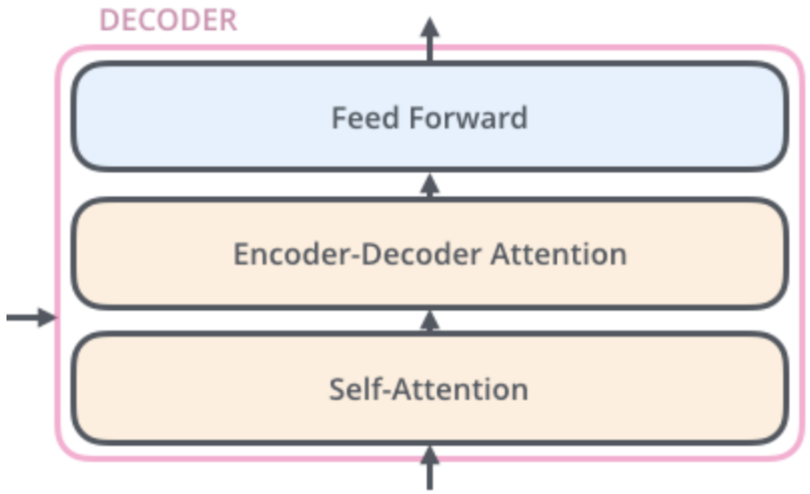
Transformers: More Tricks

- Recall a big innovation for ResNets: residual connections
  - And also layer normalizations
  - Apply to our encoder layers



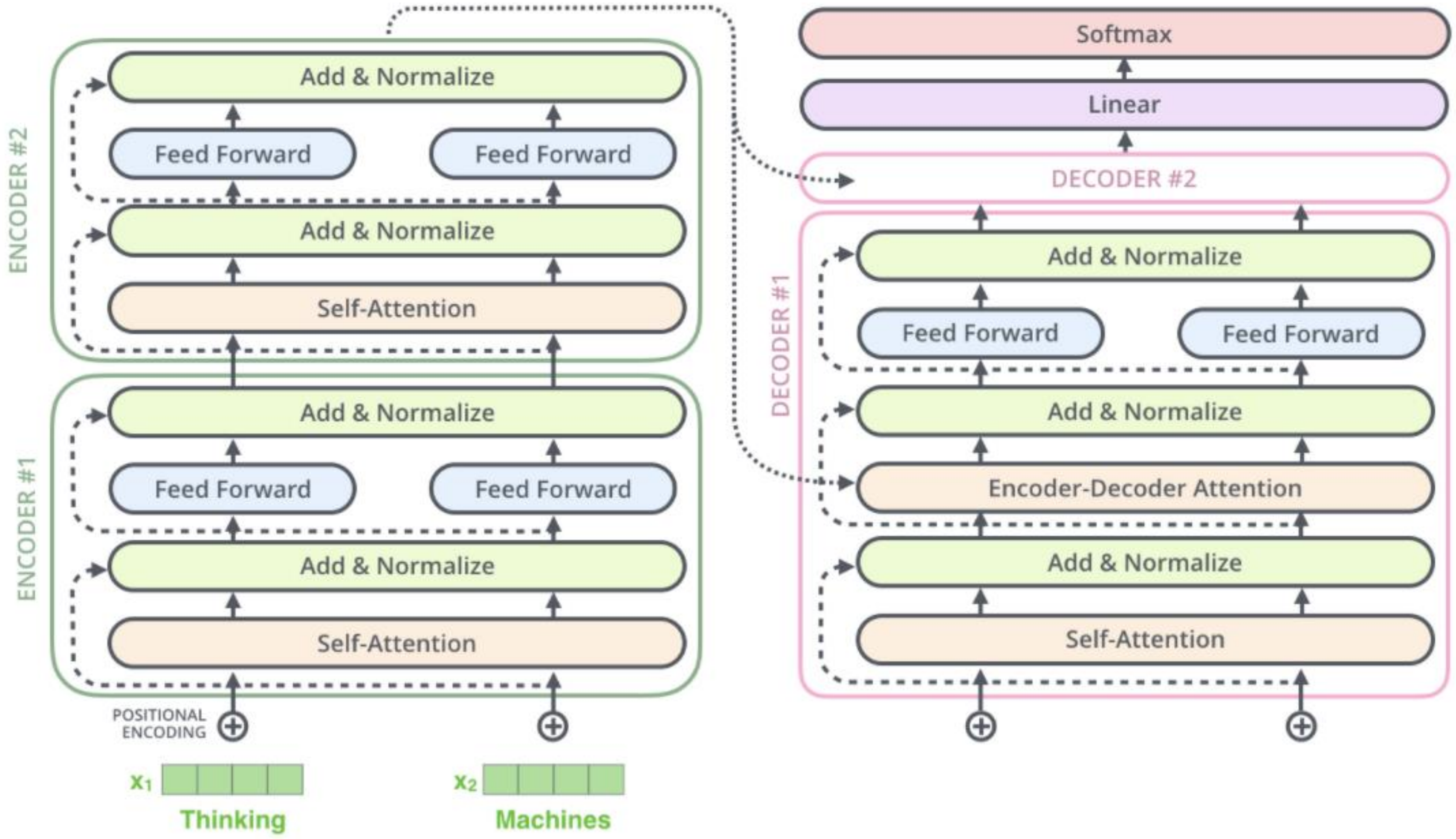
Transformers: Inside a Decoder

- Let’s take a look at the decoder. Three components:
  - 1. **Self-attention** layer (covered this)
  - 2. Encoder-decoder attention (same, but K, V come from encoder)
  - 3. “Independent” **feedforward nets** for each head



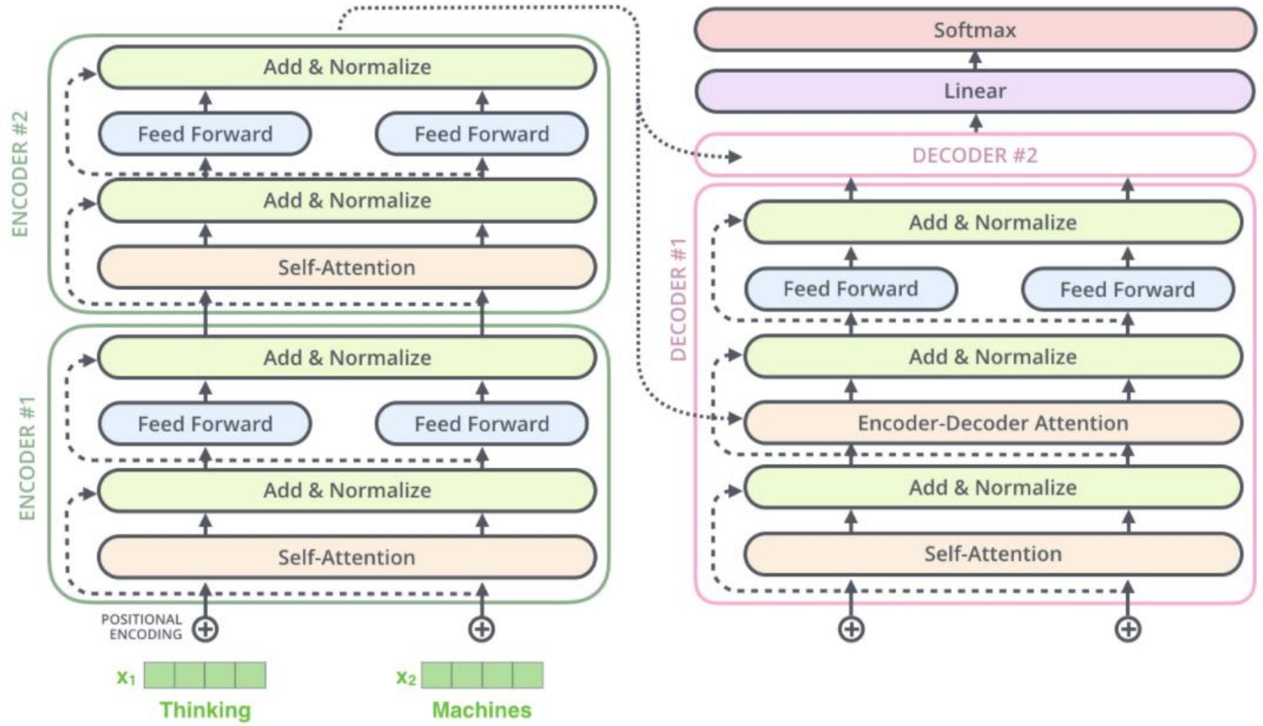
Transformers: Putting it All Together

- What does the full architecture look like?



Transformers: The Rest

- Next time: we'll talk about
  - How to **use** it (i.e., outputs)
  - How to **train** it
  - How to **rip** it apart and build other models with it.

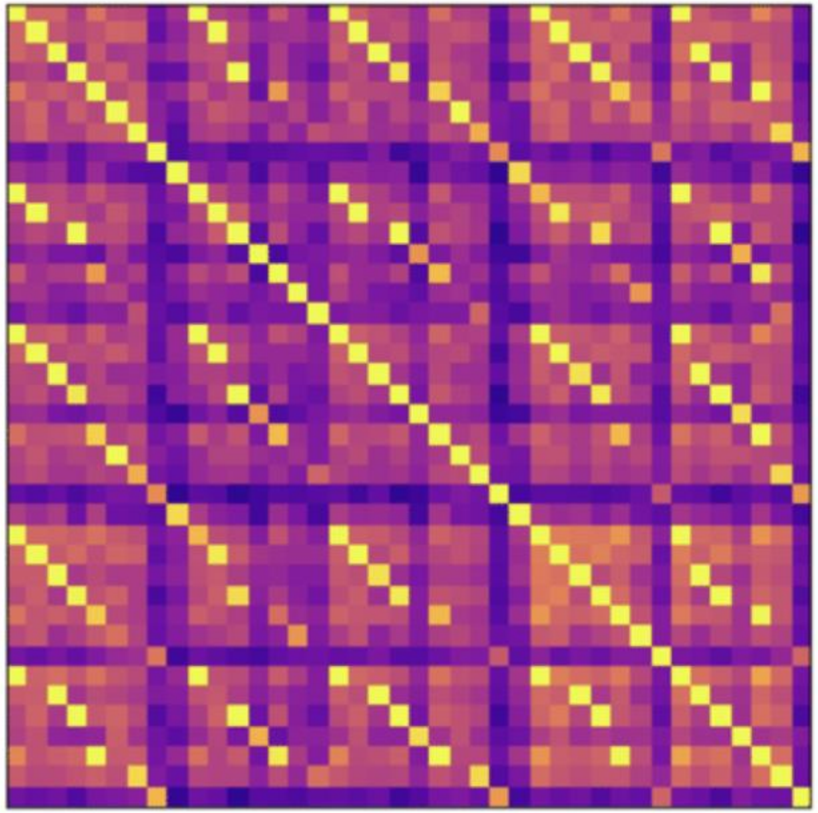


Break & Questions

Attention Alternatives?

- One annoying thing: if the sequence length is L, we're doing a $O(L^2)$ operation.
- This can be quite limiting for long

sequences...



I.e., 4000 tokens is fine, but

10^6 tokens is not.

Attention Alternatives?

Recently, lots of different approaches that attempt to get rid of this quadratic dependency

- Sometimes called **sub-quadratic** models.
- We'll briefly study a few.
- Step 1: let's get inspired by something RNN-like (well, fully linear for now). Borrow from continuous models:

$$x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t)$$
$$y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t)$$

State-Space Model

Step 1: let's get inspired by something RNN-like (well, fully linear for now). Borrow from continuous models:

State → $\boldsymbol{A}x(t)$, Input → $\boldsymbol{B}u(t)$

$$x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t)$$

Output →

$$y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t)$$

- Can ignore the "D" (think of this as a skip connection).
- Inputs, outputs are 1-D, state is higher dimensional.

State-Space Model: Discrete Form

Step 2: let’s make this a discrete function

State → $\overline{A}$; Input → $\overline{B}$

$$x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k$$

Output → $$y_k = \overline{\boldsymbol{C}} x_k$$

- Ignored D
- Can create approximations of A,B,C through discretizing.
- Looks a lot like an RNN! (or, a linear version of one)

State-Space Model: Convolutional Form

Step 3: let's unroll the recursion

$$x_0 = \overline{\boldsymbol{B}} u_0 \qquad x_1 = \overline{\boldsymbol{A}\boldsymbol{B}} u_0 + \overline{\boldsymbol{B}} u_1 \qquad x_2 = \overline{\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{A}\boldsymbol{B}} u_1 + \overline{\boldsymbol{B}} u_2$$

$$y_0 = \overline{\boldsymbol{C}\boldsymbol{B}} u_0 \qquad y_1 = \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{B}} u_1 \qquad y_2 = \overline{\boldsymbol{C}\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_1 + \overline{\boldsymbol{C}\boldsymbol{B}} u_2$$

- In general,

$$y_k = \overline{\boldsymbol{C}\boldsymbol{A}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}} u_k$$

$$y = \overline{\boldsymbol{K}} * u.$$

- This is a **convolution**!

State-Space Model: Convolutional Form

Step 3: let's unroll the recursion

- Convolution

$$y_k = \overline{C}\overline{A}^k\overline{B}u_0 + \overline{C}\overline{A}^{k-1}\overline{B}u_1 + \cdots + \overline{C}\overline{A}\overline{B}u_{k-1} + \overline{C}\overline{B}u_k$$
$$y = \overline{K} * u.$$

- But a weird one. It's a very **long** convolution.
  - Kernel as long as the input sequence (say, L).
  - Naively, is this better than attention?
  - Let's do **something else** instead.

Interlude: Time & Frequency Domains

Back to Signals and Systems class,

- Convolution in the time-domain is element-wise multiplication in the frequency domain
- So low-complexity.
- But, need to convert to frequency domain
- Solution: **FFT.** O(L log L) (and also for iFFT, to invert back).
- So, can compute fast and use during training!

$$
\begin{aligned}
y_k &= \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}}u_k \\
y &= \overline{\boldsymbol{K}} * u.
\end{aligned}
$$

Back to SSM Picture

Back to the formula

$$x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k$$
$$y_k = \overline{\boldsymbol{C}} x_k$$

- Just directly making all of these trainable parameters doesn't work so well.
  - Similar issues as in RNNs: stuff blowing up
  - Instead, various models propose approaches

S4 (Structured State Space Models) Gu et al' 22

- Build A with a special fixed transition matrix that is good at memorization
- Couple with a particular parametrization to get the discretization.

Using SSMs as Layers

Back to the formula

$$x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k$$
$$y_k = \overline{\boldsymbol{C}} x_k$$

S4 (Structured State Space Models) Gu et al' 22

- Special A state transition matrix
- Special parametrization/choice of trainable parameters

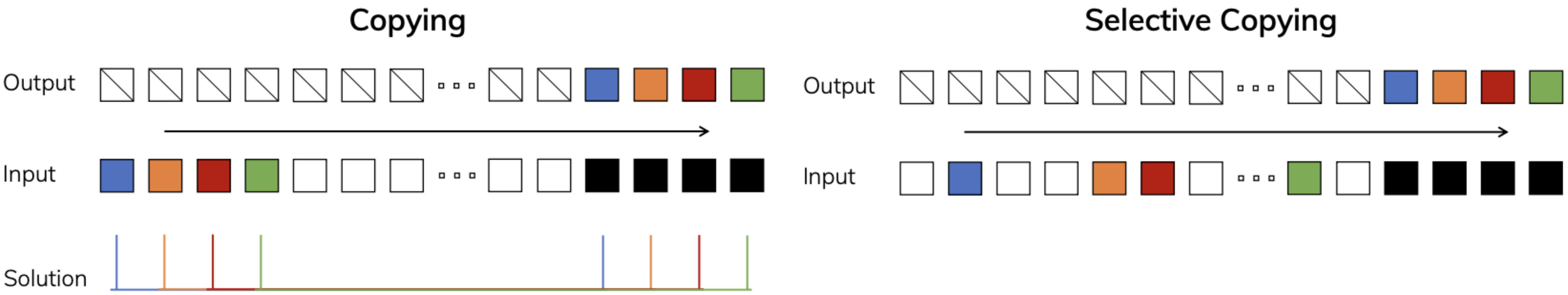
- How to actually use these? Need to define a layer,
  - Stack H of them together (similar to conv layers, multihead attn)
  - Mix with linear layer, place activation function at the end

S4 Results: The Good and the Bad

Models like S4 can address **very long sequences**

- "S4 solves the **Path-X task**, an extremely challenging task that involves reasoning about LRDs over sequences of length … 16384. All previous models have failed…"

- But, can struggle with "selective" tasks.



S4 Results: The Good and the Bad

Solution: need some type of context-aware approach

- **Mamba Model**
  - Gu and Dao ‘23, “Mamba: Linear-Time Sequence Modeling with Selective State Spaces”

Algorithm 1 SSM (S4)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{Parameter}$
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{Parameter}$
$\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{Parameter}$
$\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\mathsf{Parameter})$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \mathsf{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
return y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{Parameter}$
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
$\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D}) \leftarrow \tau_\Delta(\mathsf{Parameter} + s_\Delta(x))$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{B}, \mathsf{L}, \mathsf{D}, \mathsf{N}) \leftarrow \mathsf{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \mathsf{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

Lots of Related Approaches & Variations

- **Linear attention**. “Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention”. Katharopoulos et al, ‘20
- **RWKV**. “RWKV: Reinventing RNNs for the Transformer Era”, Peng et al ‘23

We’ll see more as we go!