CS 839: Foundation Models
Models I

Fred Sala

University of Wisconsin-Madison

Sept. 19, 2024

Announcements

- **Announcement:**
  - Homework 1 is released

- Class roadmap:

Thursday Sept. 19	Models I
Tuesday Sept. 24	Models II
Thursday Sept. 26	Prompting I
Tuesday Oct. 1	Prompting II
Thursday Oct. 3	Specialization

Mostly Language Models

Outline

- **From Last Time**
  - Finish up SSMs, a little bit more on decoders
- **Encoder-only Models**
  - Example: BERT, architecture, multitask training, fine-tuning
- **Decoder-only Models**
  - Example: GPT, architecture, basic functionality

Outline

- **From Last Time**
  - Finish up SSMs, a little bit more on decoders
- **Encoder-only Models**
  - Example: BERT, architecture, multitask training, fine-tuning
- **Decoder-only Models**
  - Example: GPT, architecture, basic functionality

State-Space Model: Discrete Form

Step 2: let's make this a discrete function

State Input

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

Output $\rightarrow$

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

- 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

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

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

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

- In general, $y = \overline{\boldsymbol{K}} * u.$

- This is a **convolution**!

State-Space Model: Convolutional Form

Step 3: let's unroll the recursion

- Convolution

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

- 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

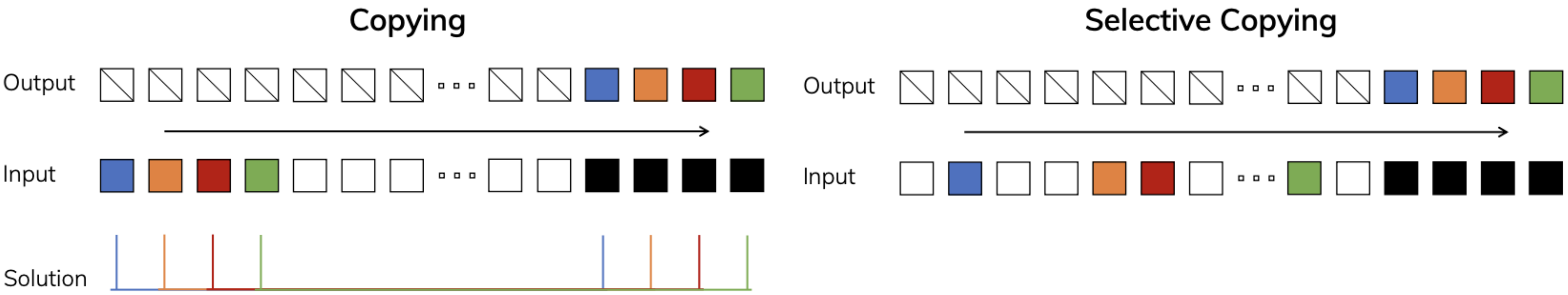
- How to actually use these? Need to define a layer,
  - Stack H of them together (similar to conv layers, multihead attn)
  - Mix with linear layer, place activation function at the end

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

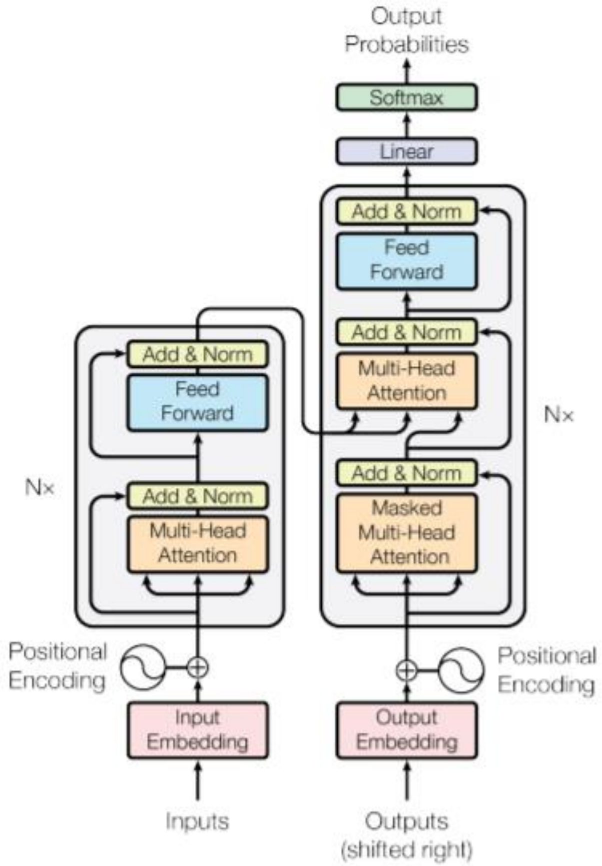
- **Linear attention**. “Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention”. Katharopoulos et al, ‘20
- **RWKV**. “RWKV: Reinventing RNNs for the Transformer Era”, Peng et al ‘23

We’ll see more as we go!

Back To Transformers: Model Architecture

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

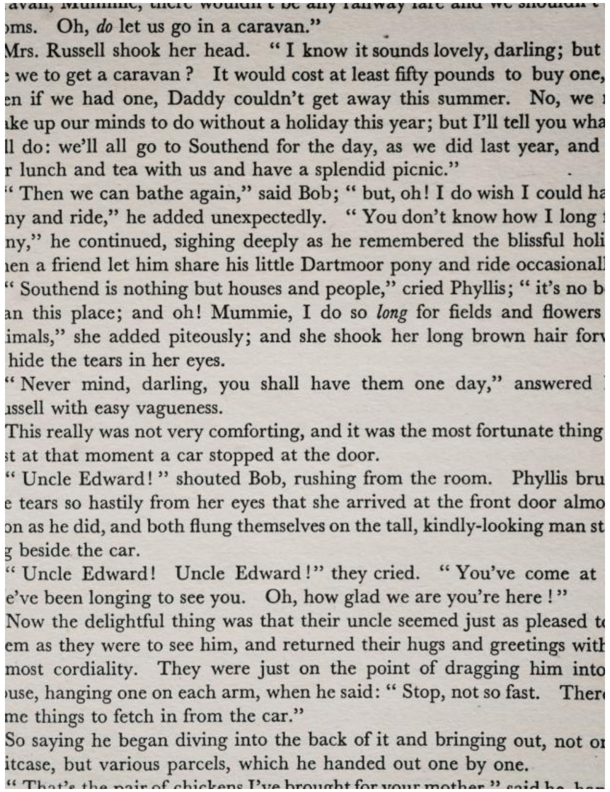
- Architecture
  - The famous picture you've seen
  - Centered on self-attention blocks

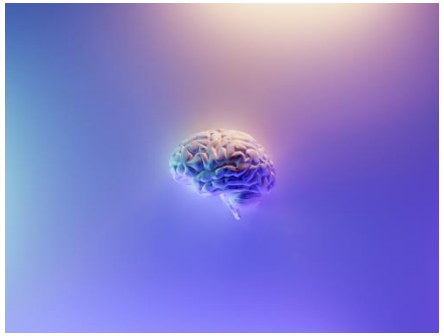


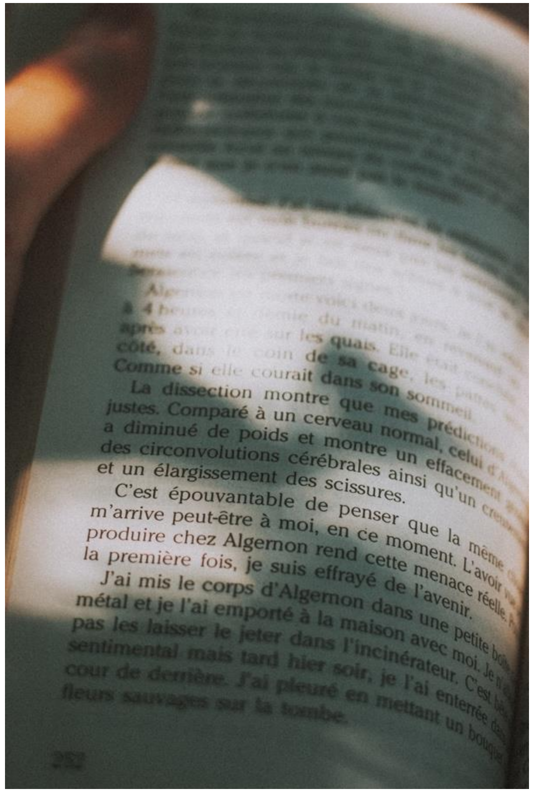
Vaswani et al. '17

Interlude: Encoder-Decoder Models

- Translation tasks: natural encoder-decoder architecture
- Intuition:

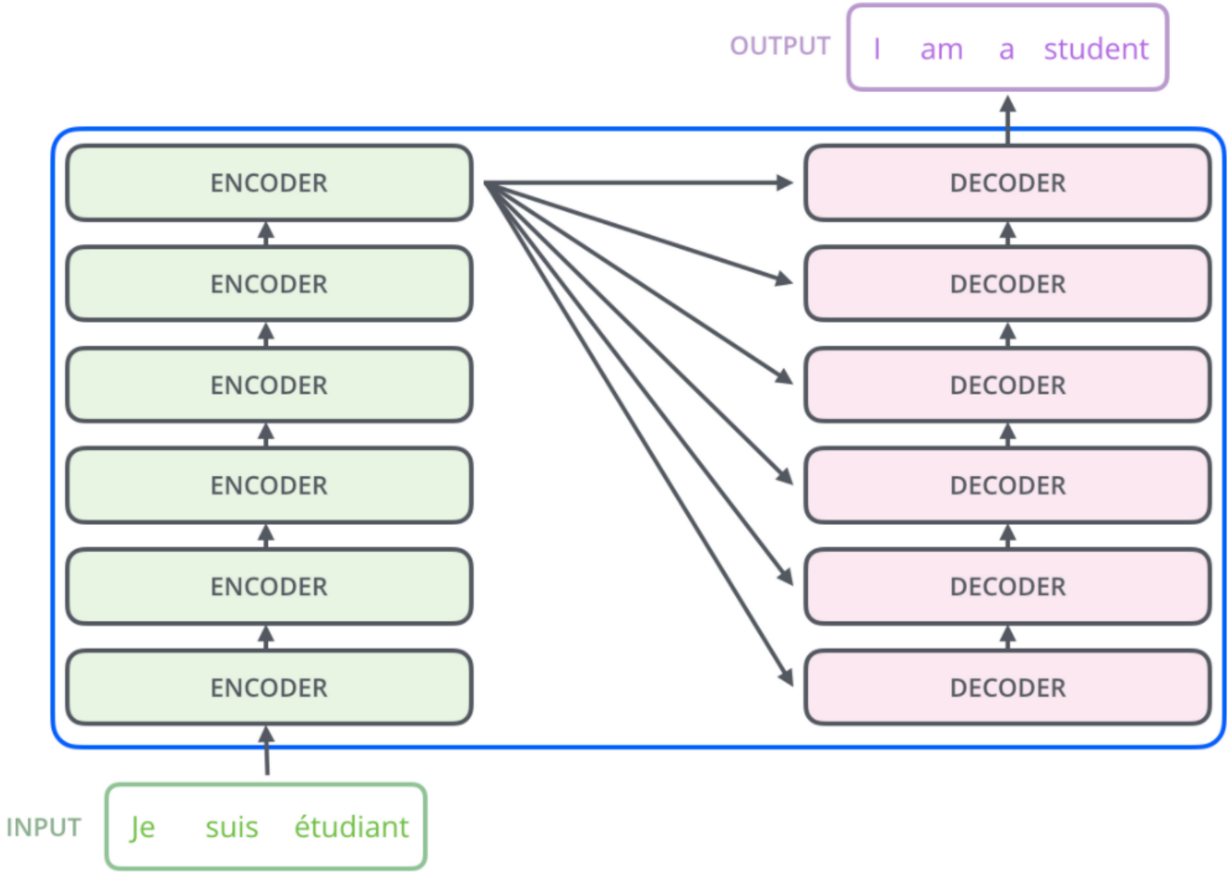






Transformers: Architecture

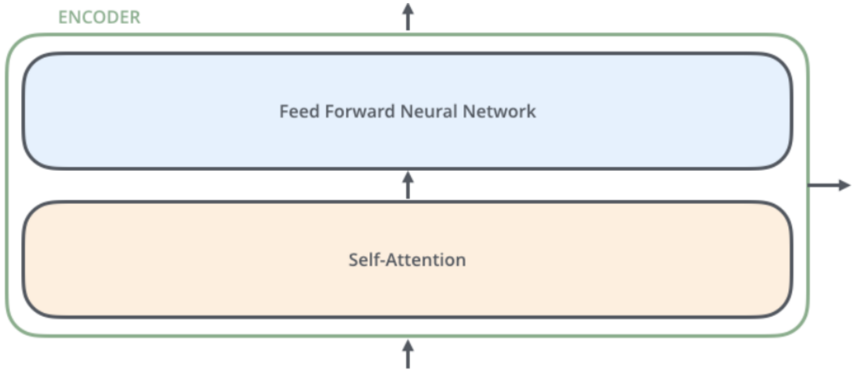
- **Sequence-sequence** model with **stacked** encoders/decoders:
  - For example, for French-English translation:

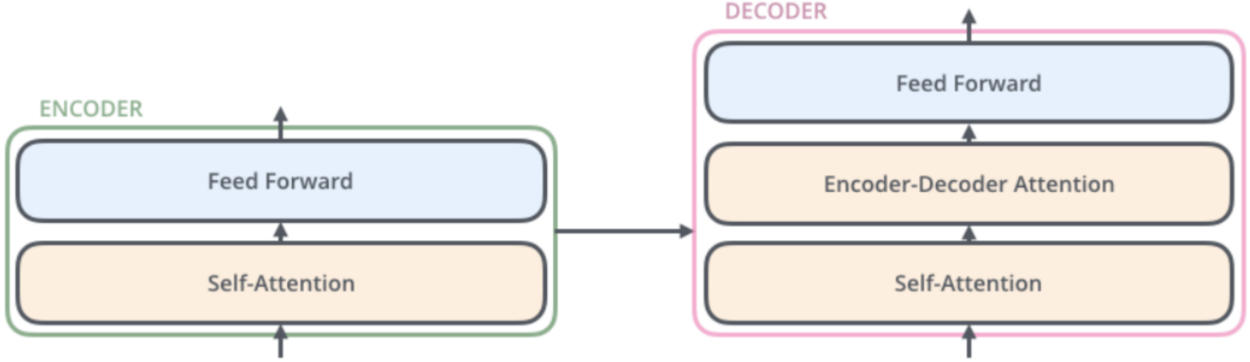


Transformers: Architecture

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

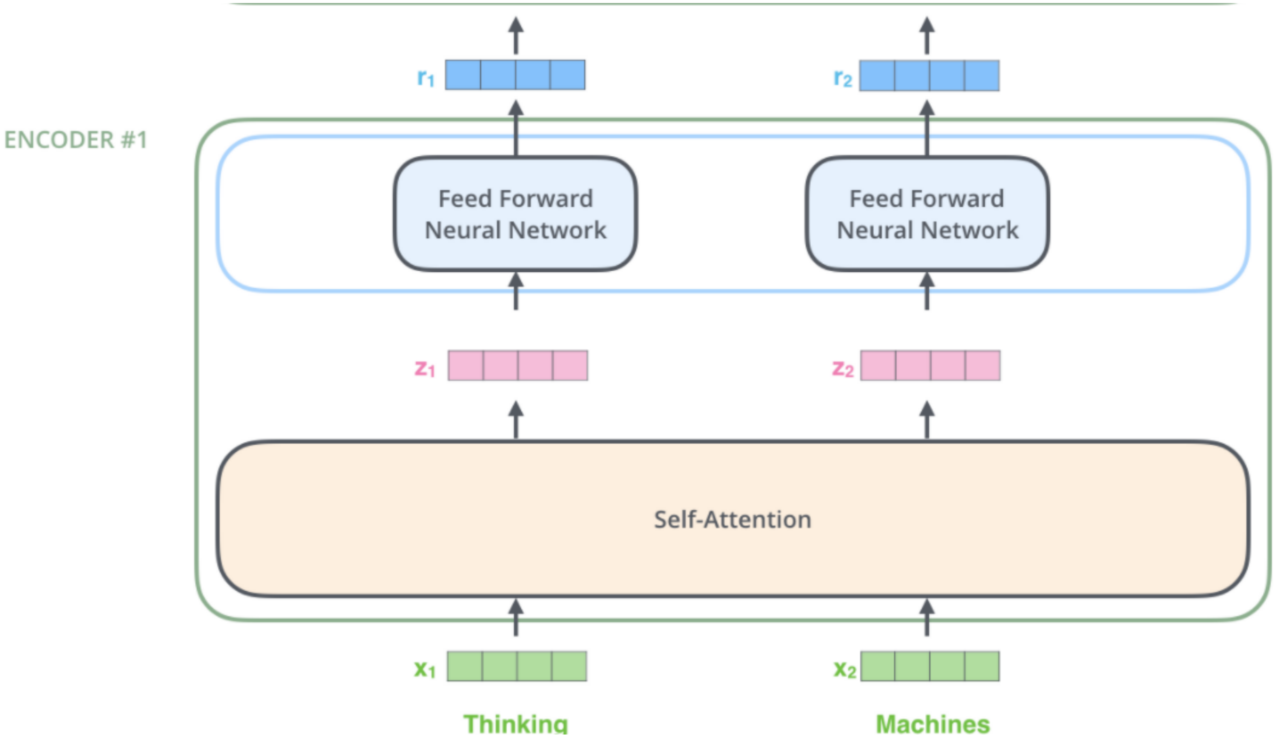
  - Focus on encoder first: **pretty simple**! 2 components:
    - Self-attention block
    - Fully-connected layers (i.e., an MLP)
    - Captures **1) interactions 2) processing** (separately!)





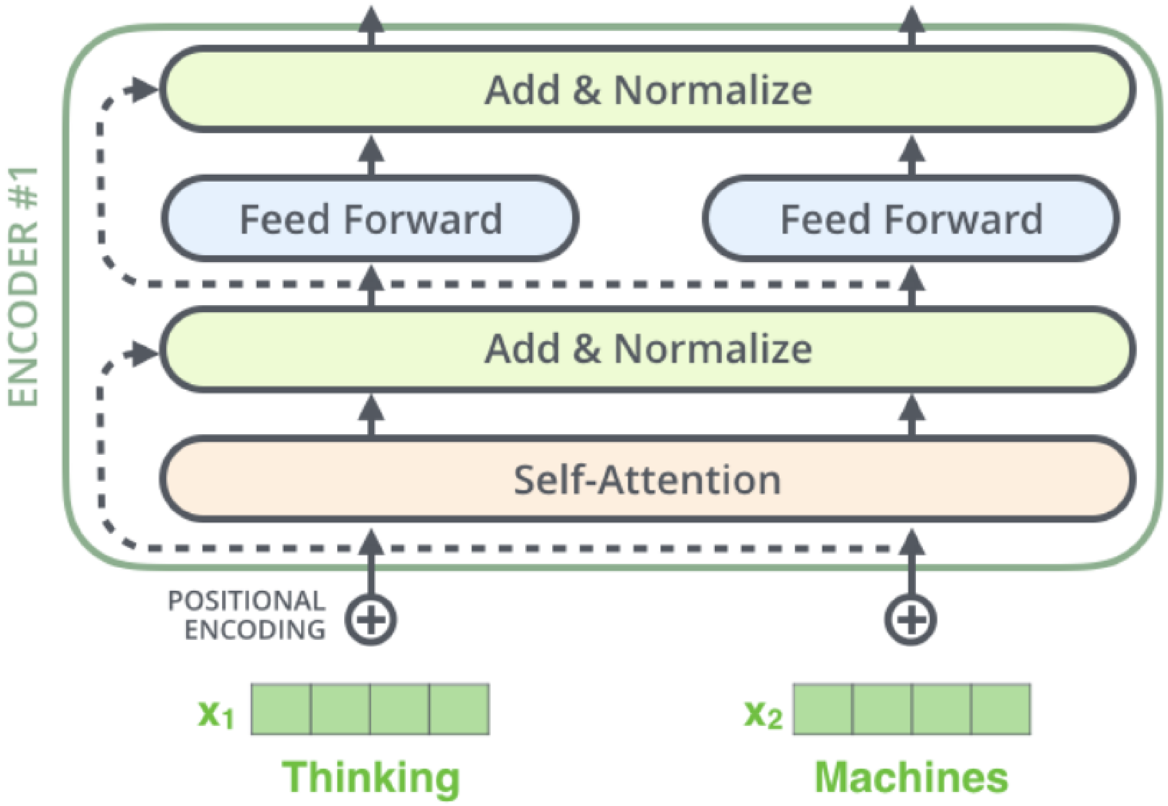
Transformers: Inside an Encoder

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



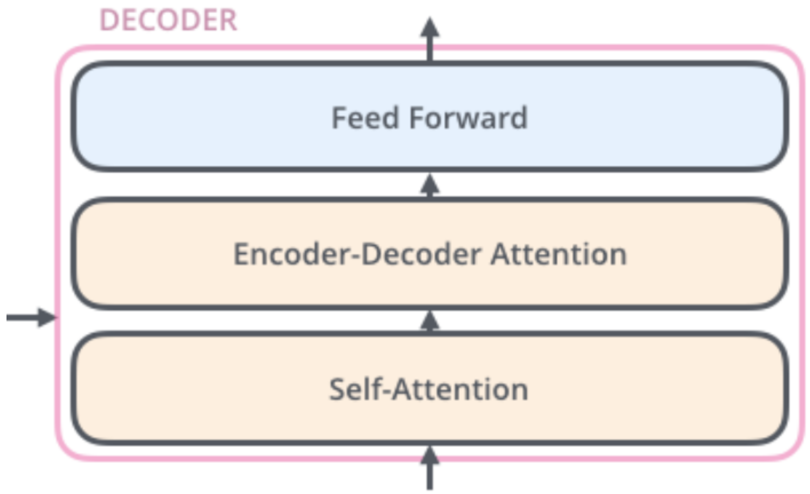
Transformers: More Tricks

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



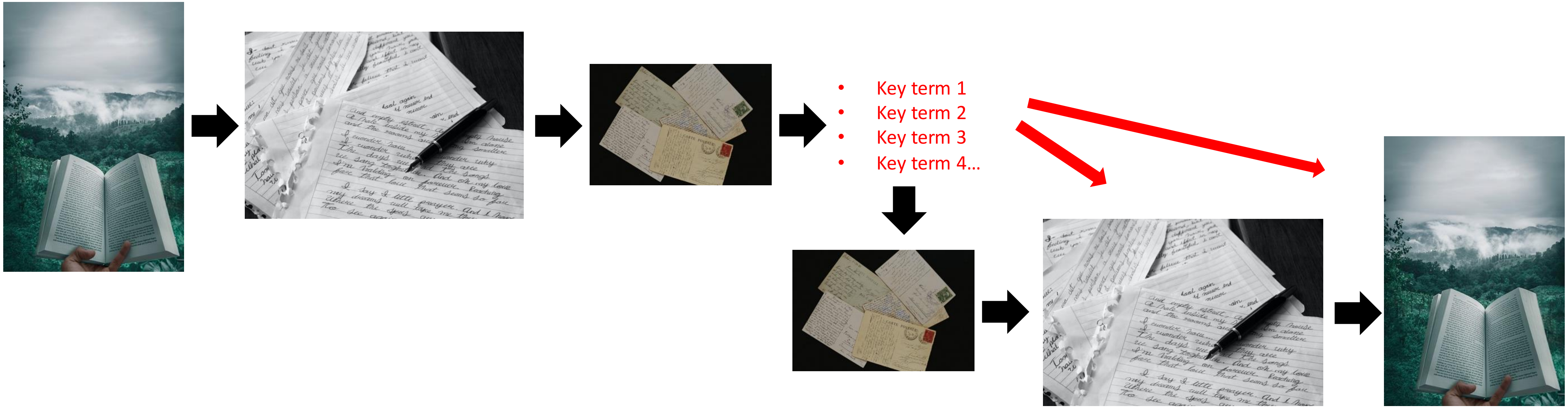
Transformers: Inside a Decoder

- 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



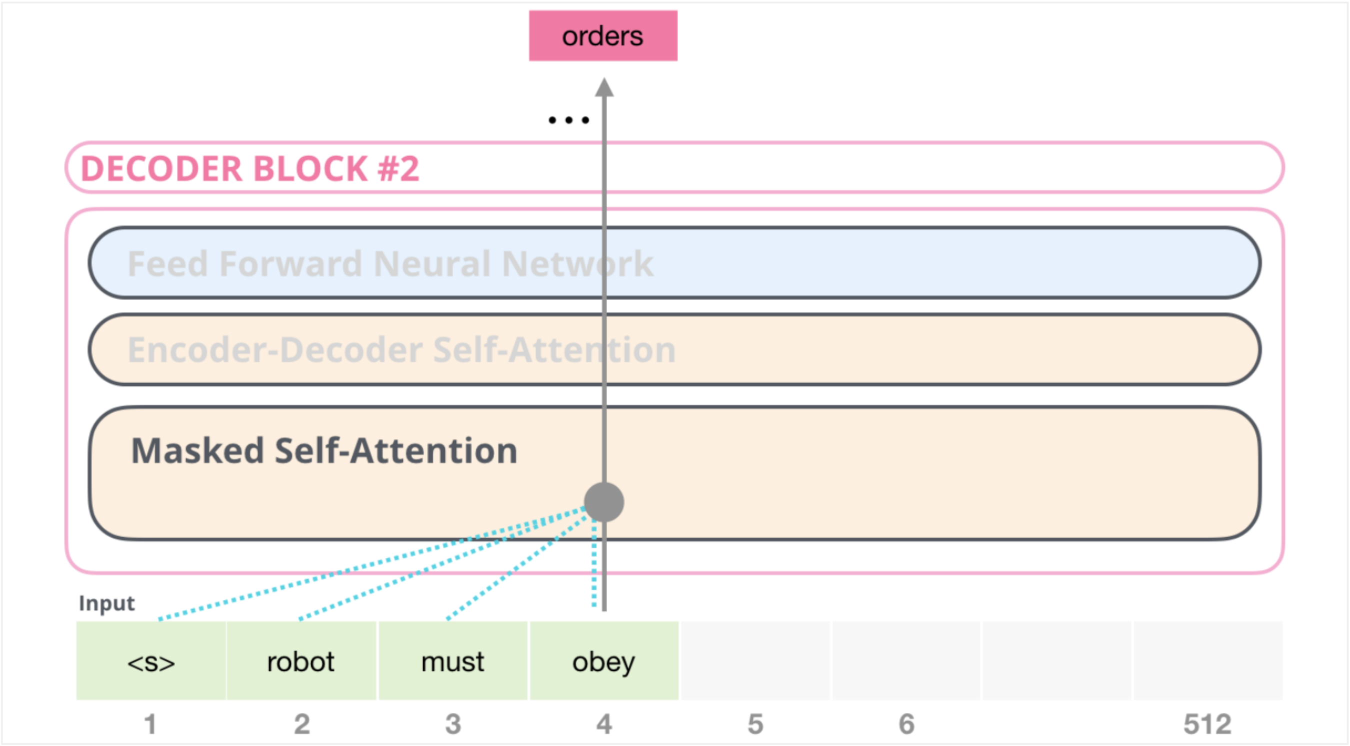
Transformers: Cross-Attention

- Why encoder-decoder attention ?
  - Recall: same as before, but K, V come from encoder
  - Actually more traditional, but… **intuition**:



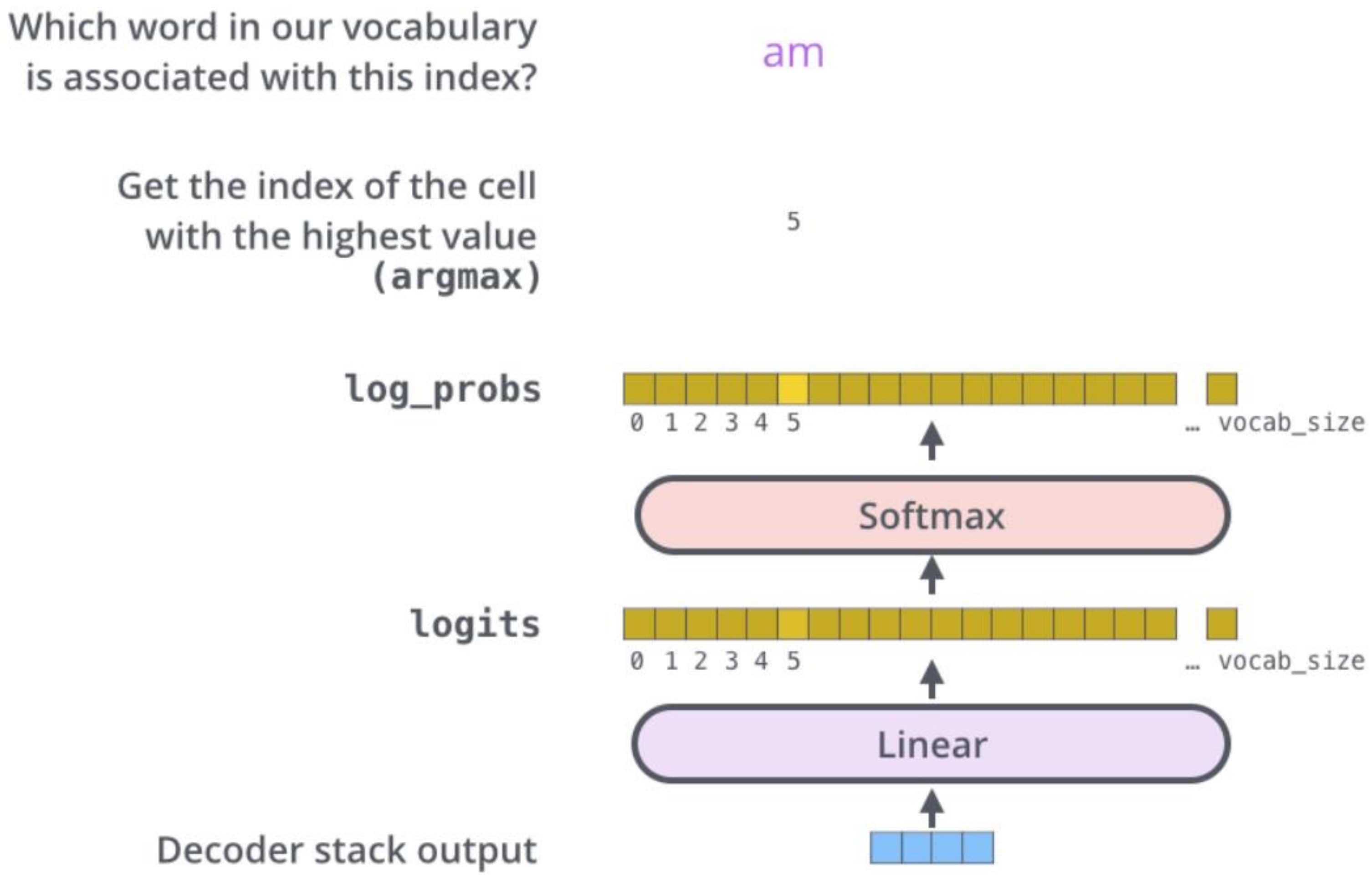
Transformers: Decoder Masking

- One more interesting bit!
  - At the decoder level, self-attention changes a bit:
  - Masked instead: block *future* words from being attended to



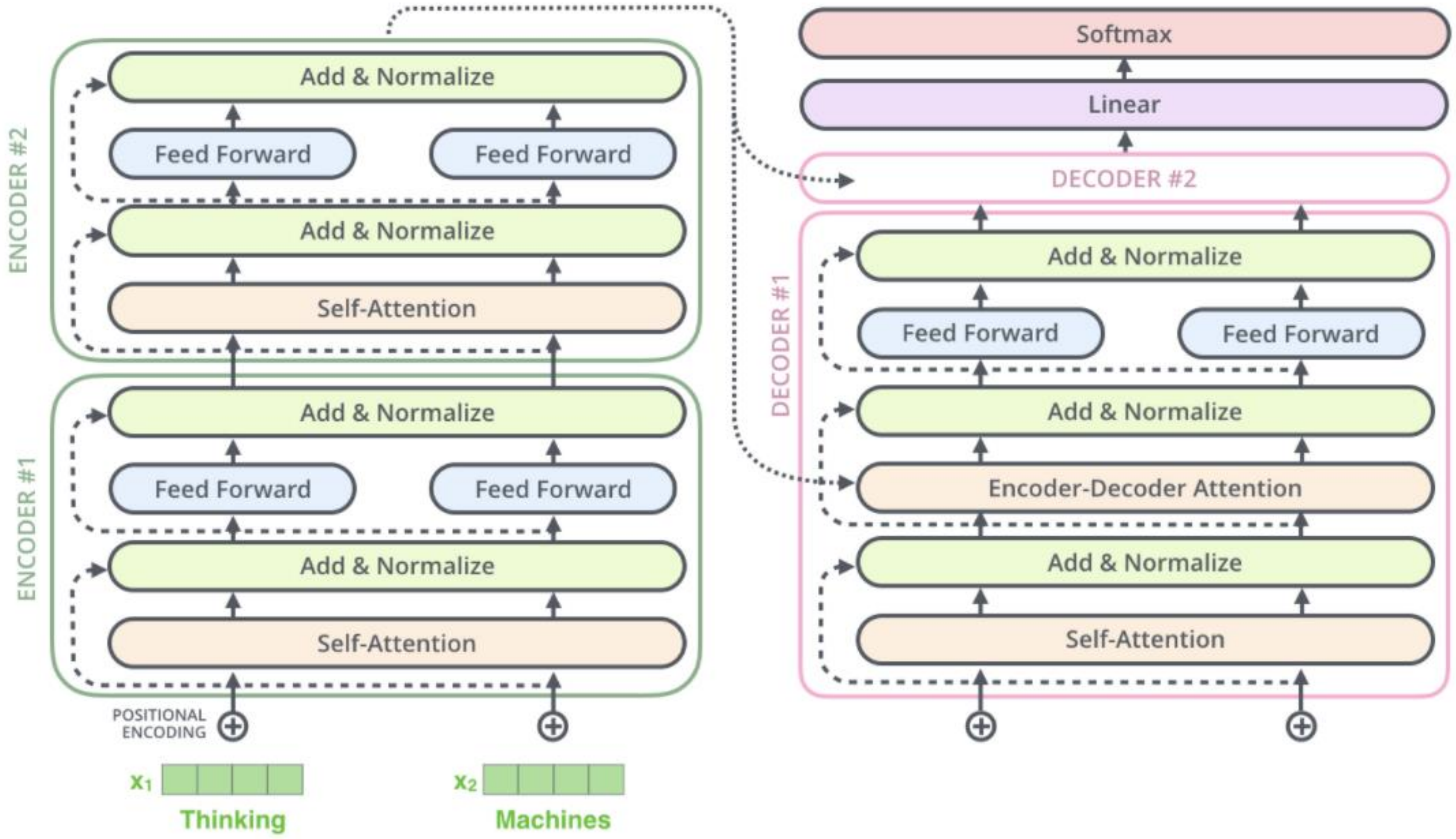
Transformers: Outputs

- Finally, let’s see the final layer and outputs



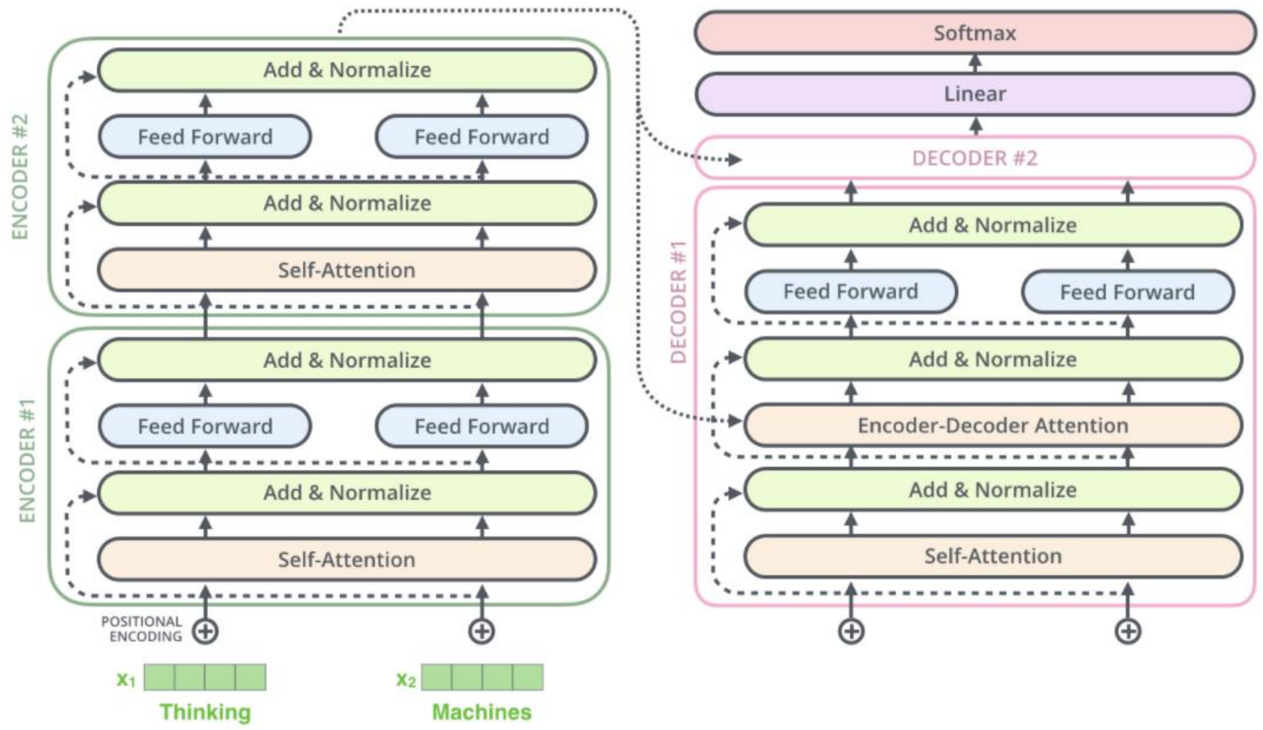
Transformers: Putting it All Together

- What does the full architecture look like?



Transformers: Training

- Data: standard datasets (WMT English-German)
  - ~5 million pairs for this dataset
  - Nothing very special: Adam optimizer



Break & Questions

Outline

- **From Last Time**
  - Finish up SSMs, a little bit more on decoders
- **Encoder-only Models**
  - Example: BERT, architecture, multitask training, fine-tuning
- **Decoder-only Models**
  - Example: GPT, architecture, basic functionality

Why Encoder-Decoder?

Wanted two things for translation:

- 1) **Outputs** in natural language
- 2) Tight alignment with **input**

What happens if we relax these?

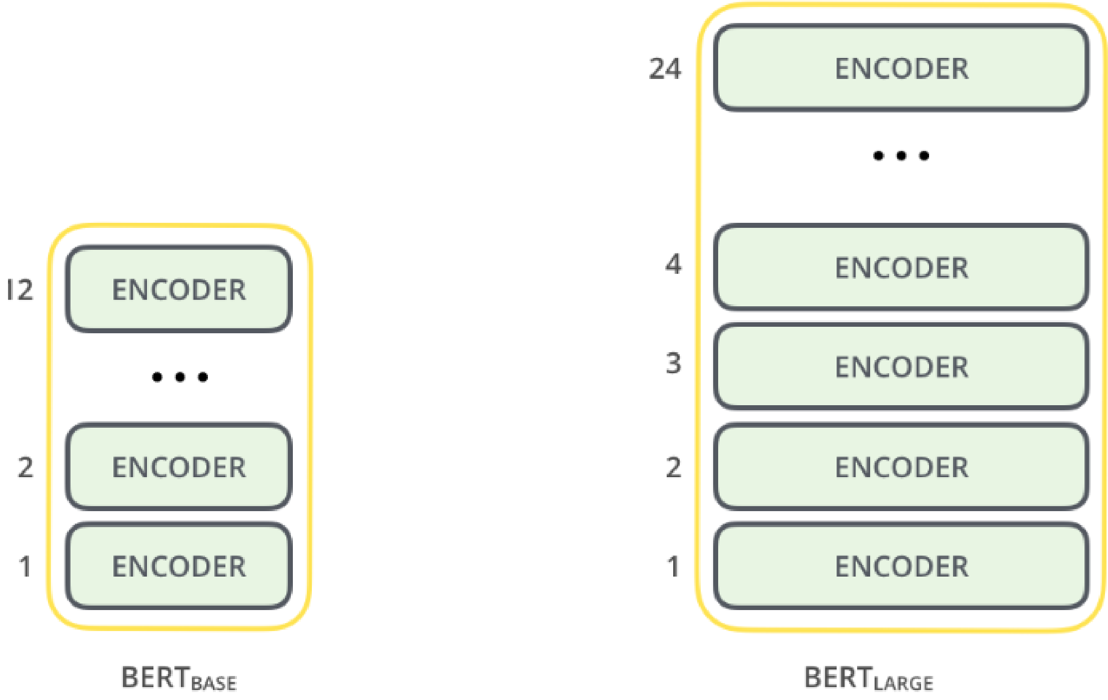
1. Encoder-only models
2. Decoder-only models

Encoder-Only Models: BERT

Let's get rid of the first part

- 1) **Outputs** in natural language
- 2) Tight alignment with **input**

- Rip away decoders
  - Just stack encoders

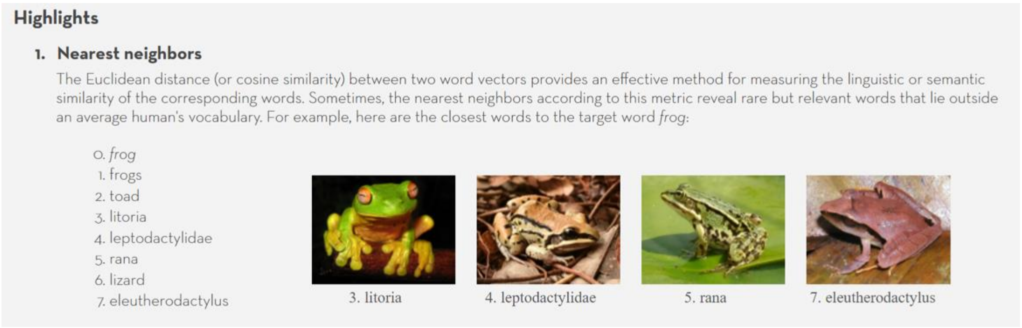


Interlude: Contextual Embeddings

Q: Why is it called "BERT"?

- **A**: In a sense, follows up ELMo

- Story:
  - **2013**: "Dense" word embeddings (**Word2Vec, Glove**)
  - Downside: fixed representations per word
    - "Bank": building or riverside?
  - Need: contextual representations
    - Using language model-like techniques
    - 2018: ELMo, BERT
    - ELMo: uses LSTMs, BERT uses transformers



Highlights

1. **Nearest neighbors**

The Euclidean distance (or cosine similarity) between two word vectors provides an effective method for measuring the linguistic or semantic similarity of the corresponding words. Sometimes, the nearest neighbors according to this metric reveal rare but relevant words that lie outside an average human's vocabulary. For example, here are the closest words to the target word *frog*:

0. *frog*
1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

3. litoria

4. leptodactylidae

5. rana

7. eleutherodactylus

https://nlp.stanford.edu/projects/glove/

Interlude: Contextual Embeddings

Q: Why is it called "BERT"?

- **A**: In a sense, follows up ELMo

BERT acronym:

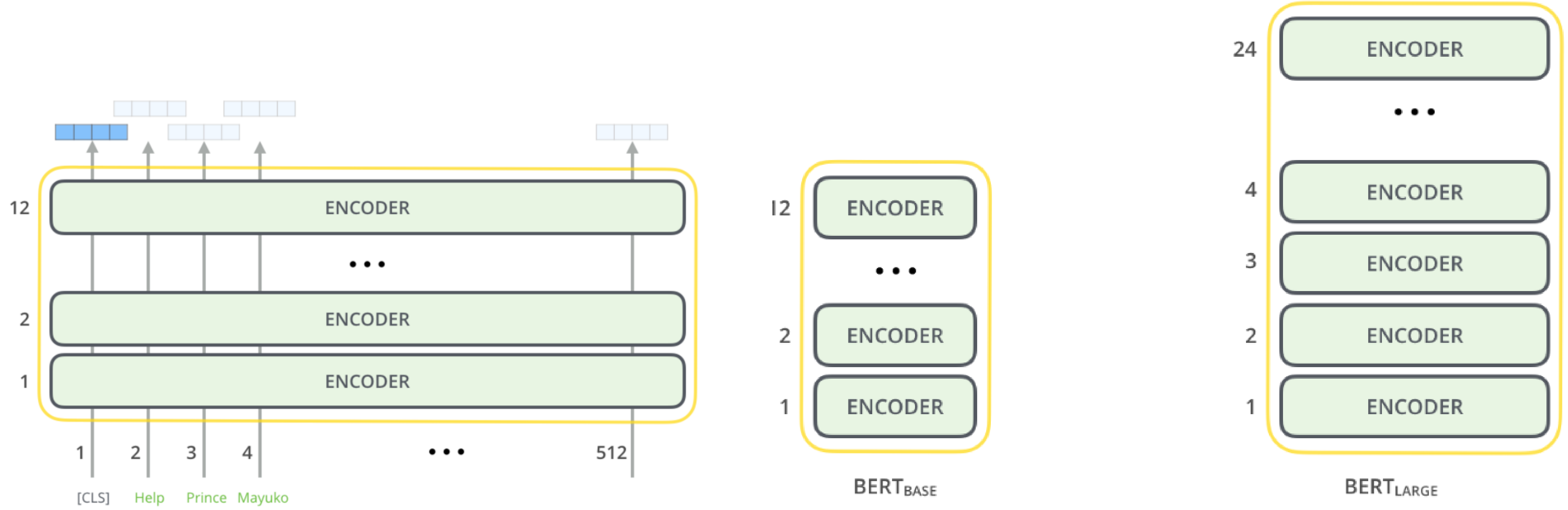
- **B**idirectional **E**ncoder **R**epresentations from **T**ransformers.
- ERT should make sense,

- Bidirectional: no causal masks, look at both sides of a word!
- Captured in self-attention block

BERT: Forward Pass

BERT architecture

- Rip away decoders
  - Just stack encoders

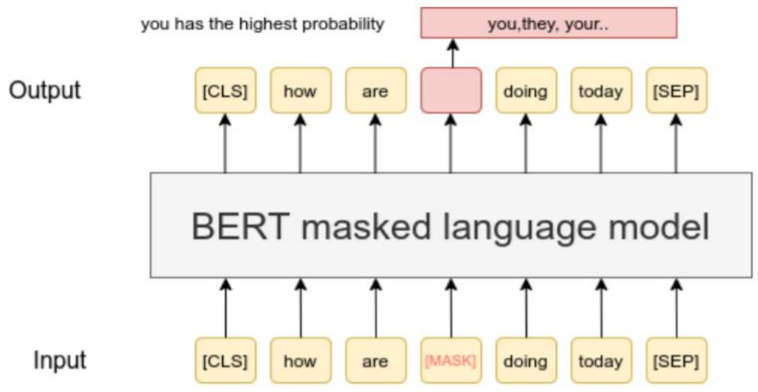


BERT: Training

Training is more interesting!

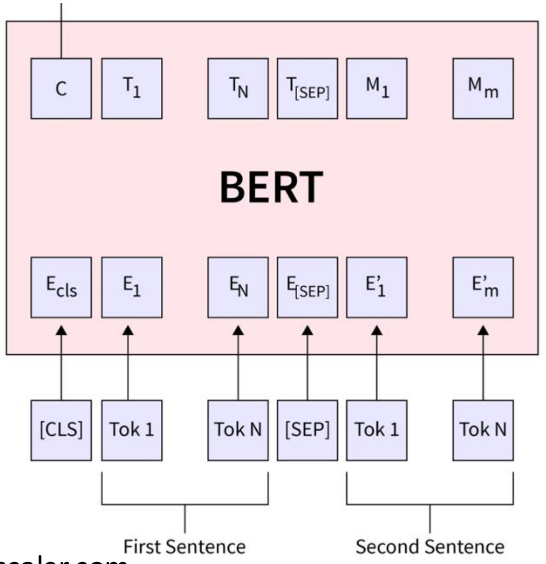
- Pretraining. Then fine-tuning on task of interest

- Back to **self-supervised learning!**
- Two tasks for **pretraining**.



sbert.net

1. Masked Language Modeling



scaler.com

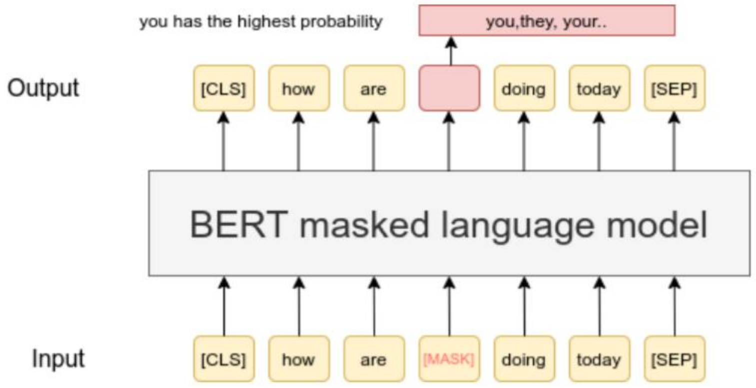
2. Next Sentence Prediction

BERT: Training Task 1

Masked Language Modeling Task

- Use [MASK] token for word to be predicted

- Which words to mask?
  - Original paper: 15% of words at random
  - But… of these
    - 10% of the time, no [MASK], flip word randomly
    - 10% of the time leave word unchanged

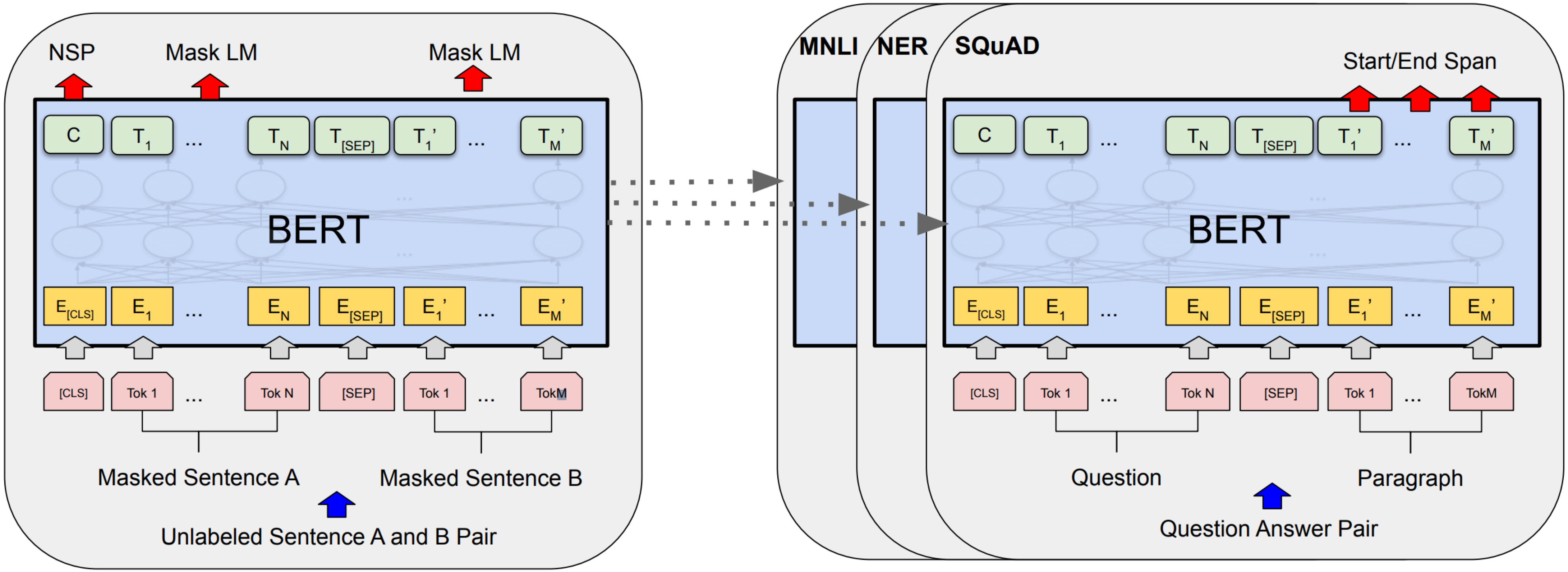


sbert.net

BERT: Training

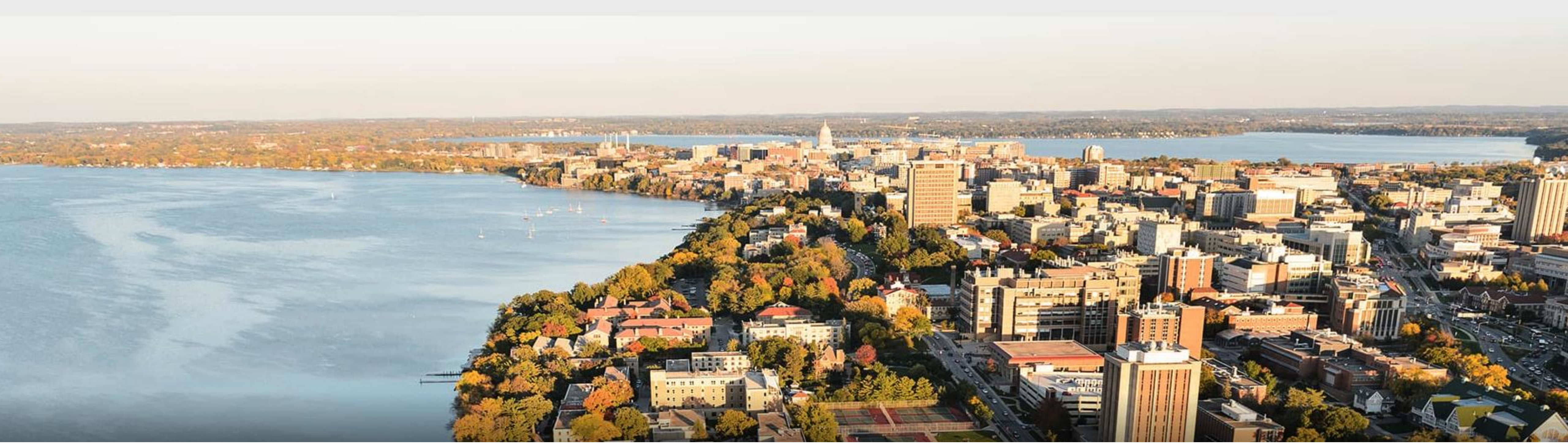
Training is more interesting,

- Pretraining. Then fine-tuning on task of interest



Pre-training Fine-Tuning

Devlin et al



Break & Questions

Outline

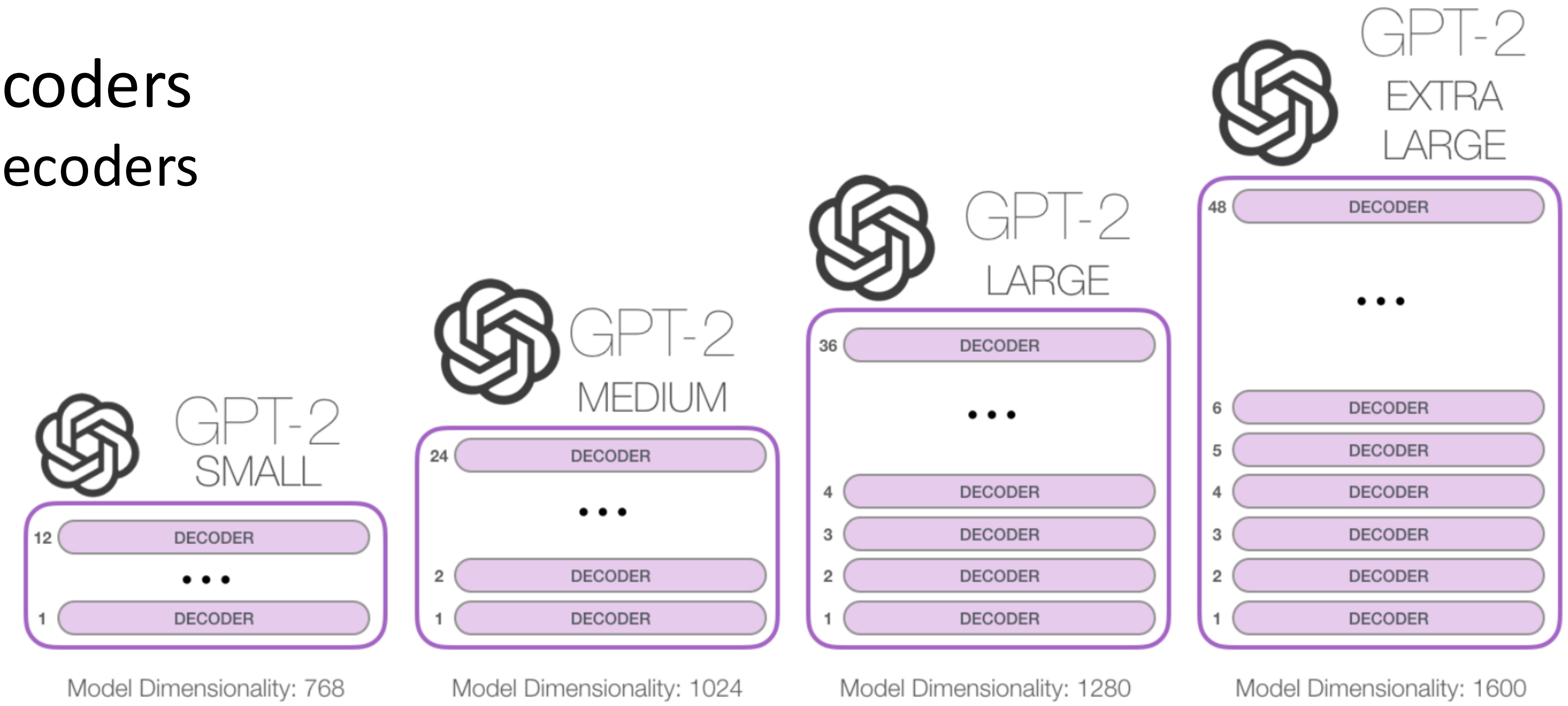
- **From Last Time**
  - Finish up SSMs, a little bit more on decoders
- **Encoder-only Models**
  - Example: BERT, architecture, multitask training, fine-tuning
- **Decoder-only Models**
  - Example: GPT, architecture, basic functionality

Decoder-Only Models: GPT

Let's get rid of the first part

- 1) **Outputs** in natural language
- 2) Tight alignment with **input**

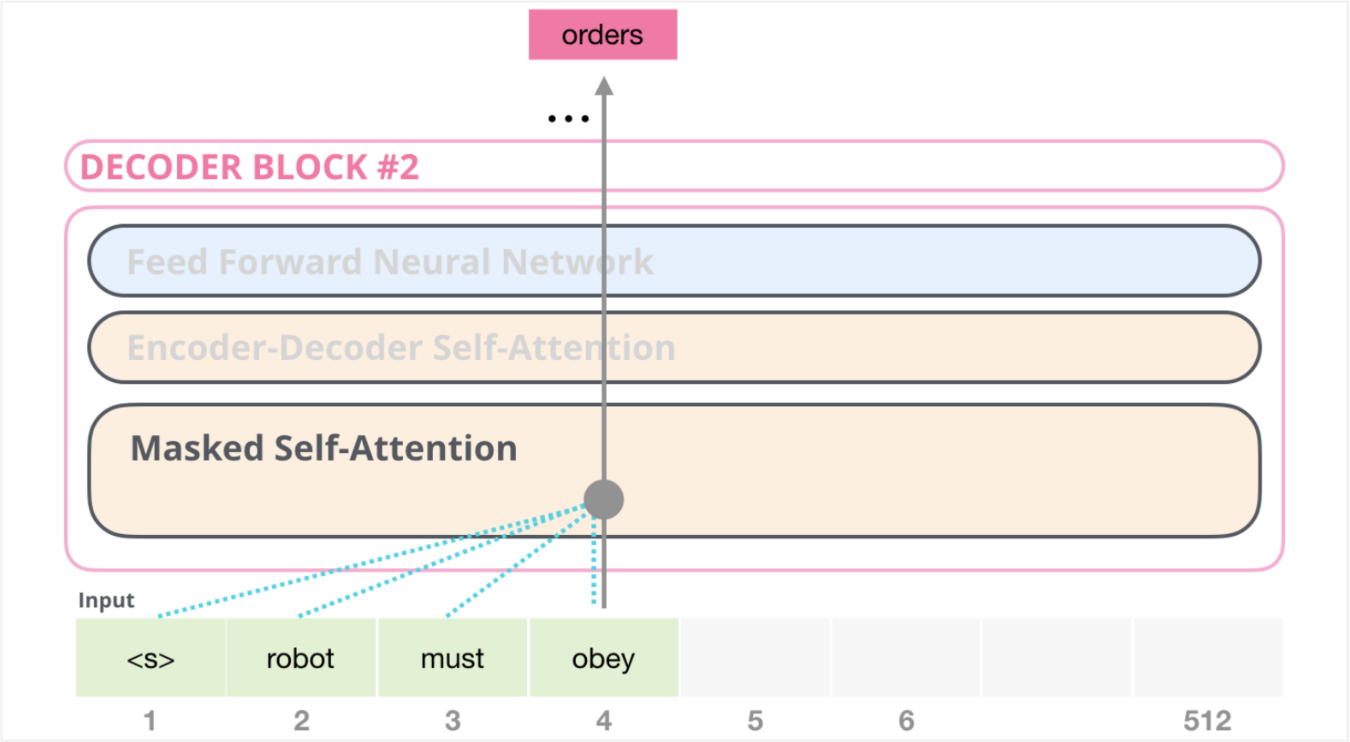
- Rip away encoders
  - Just stack decoders

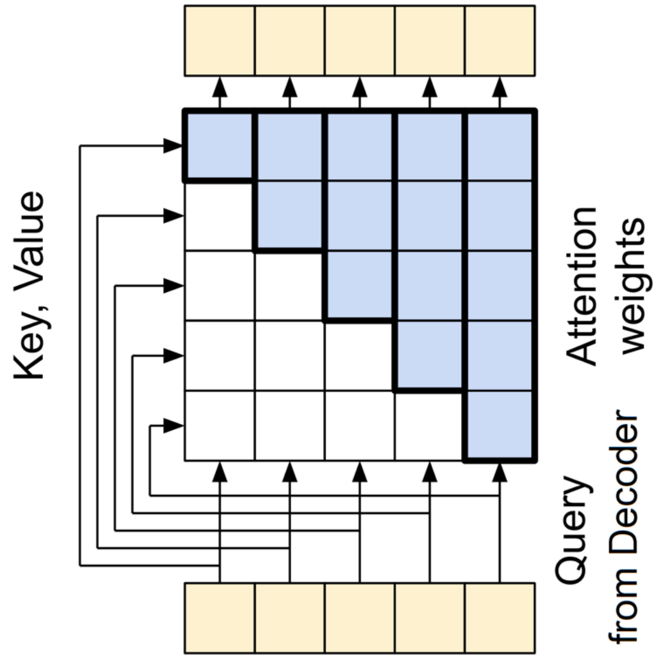


Decoder-Only Models: GPT

Rip away encoders

- Just stack decoders
- Use causal masking! NB: not a *mask token* like in BERT





PyLessons

Thank You!