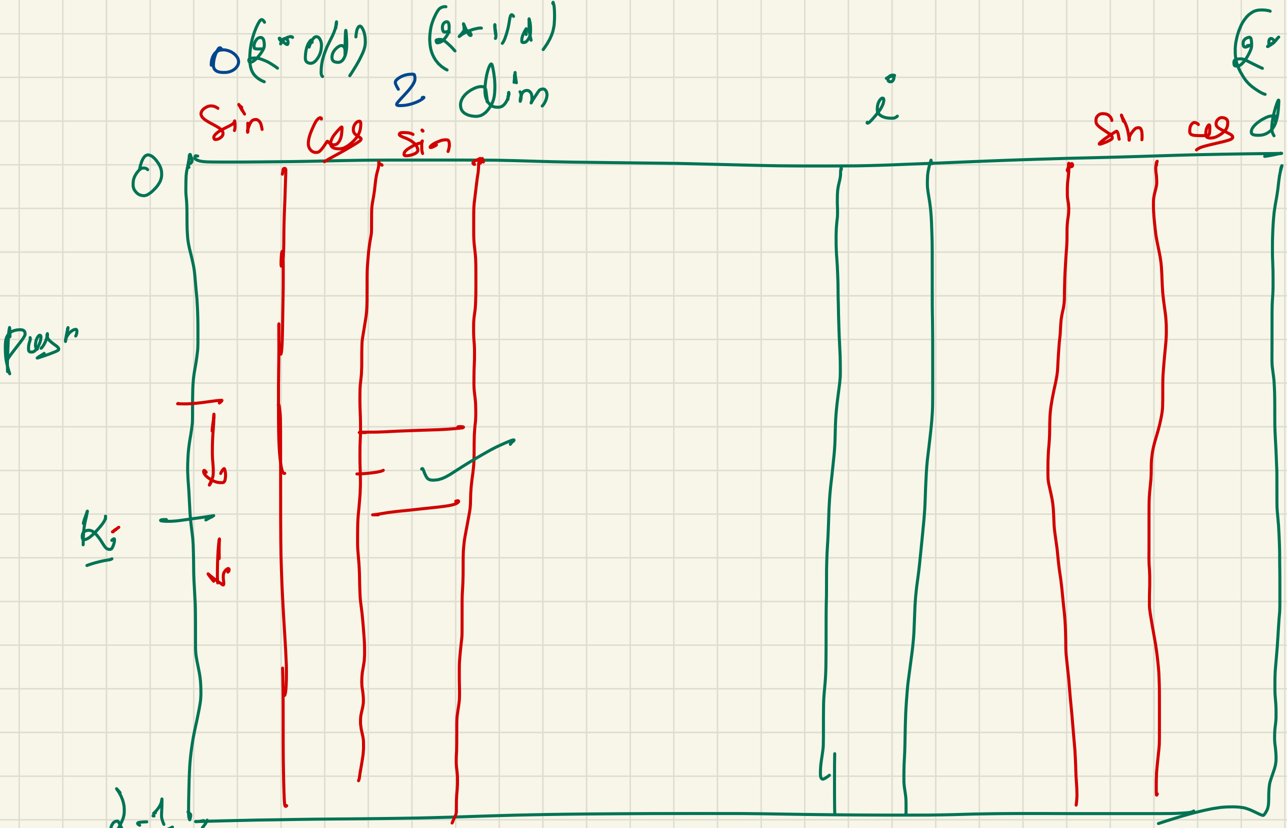


sinusoidal

= not learnable

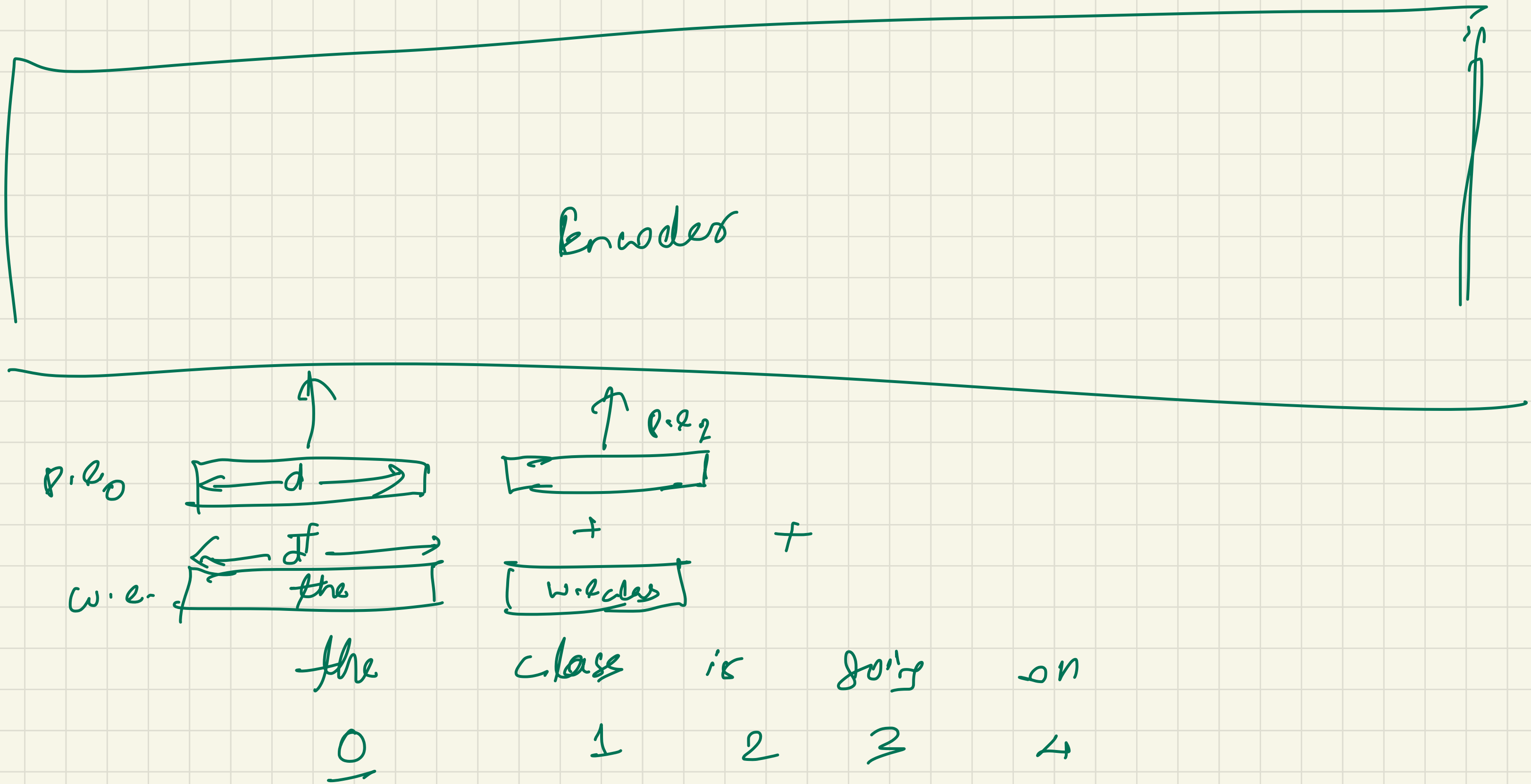
$L = 512$

$d = 512$

$L \times d$ values

Posn embedding

$$P(k, 2i) = \sin\left(\frac{k}{10000^{2*i/d}}\right)$$


Encoder
p.e 2
p.e 0
d
d
+
+
w.e.
the
w.e class
the
class
is
going
on
0
1
2
3
4

Multi-headed Self Attention Block

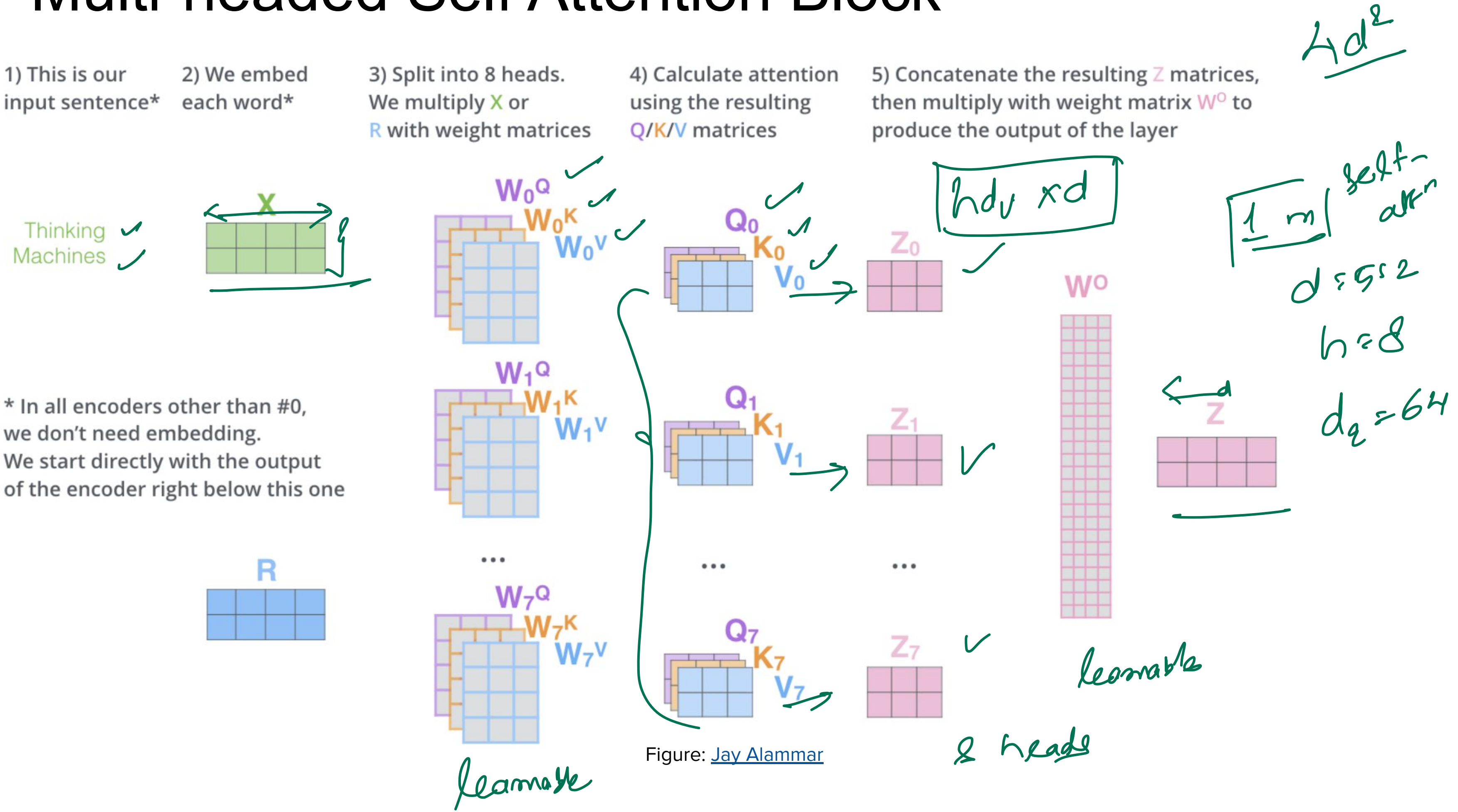


Figure: Jay Alammar

Encoder Block

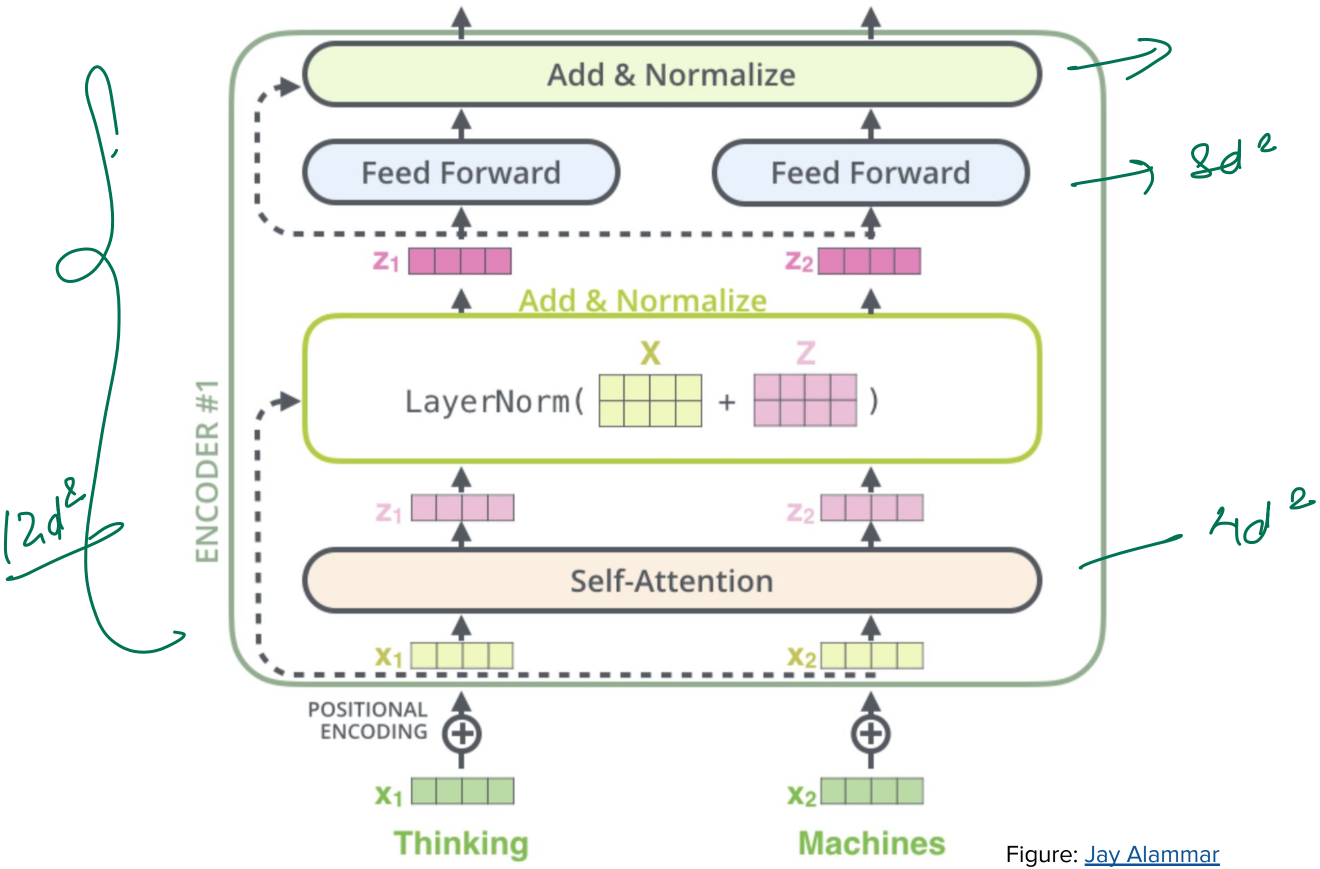


Figure: [Jay Alammar]

Transformer with encoders and decoders

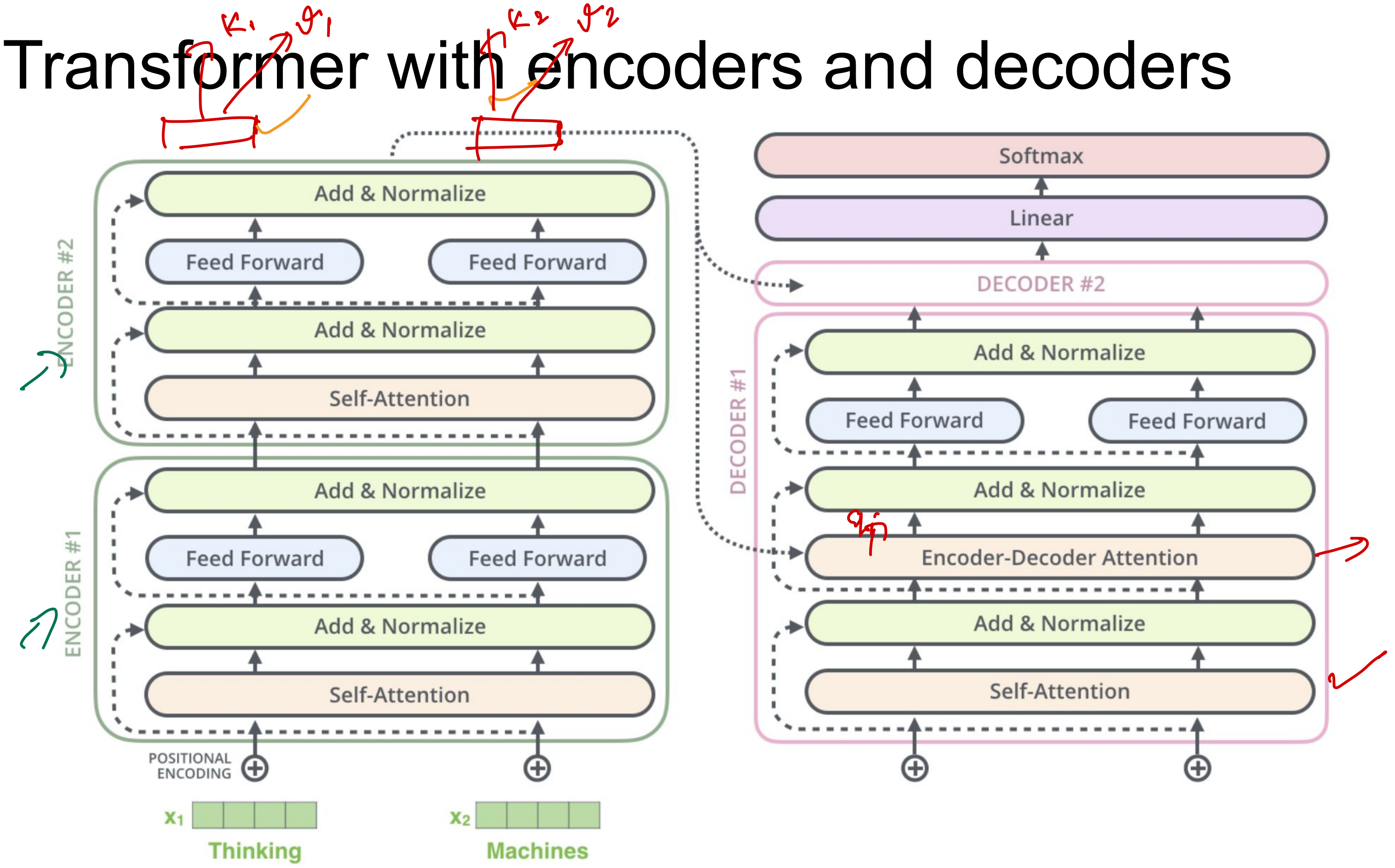


Figure: [Jay Alammar]

How is the decoder different?

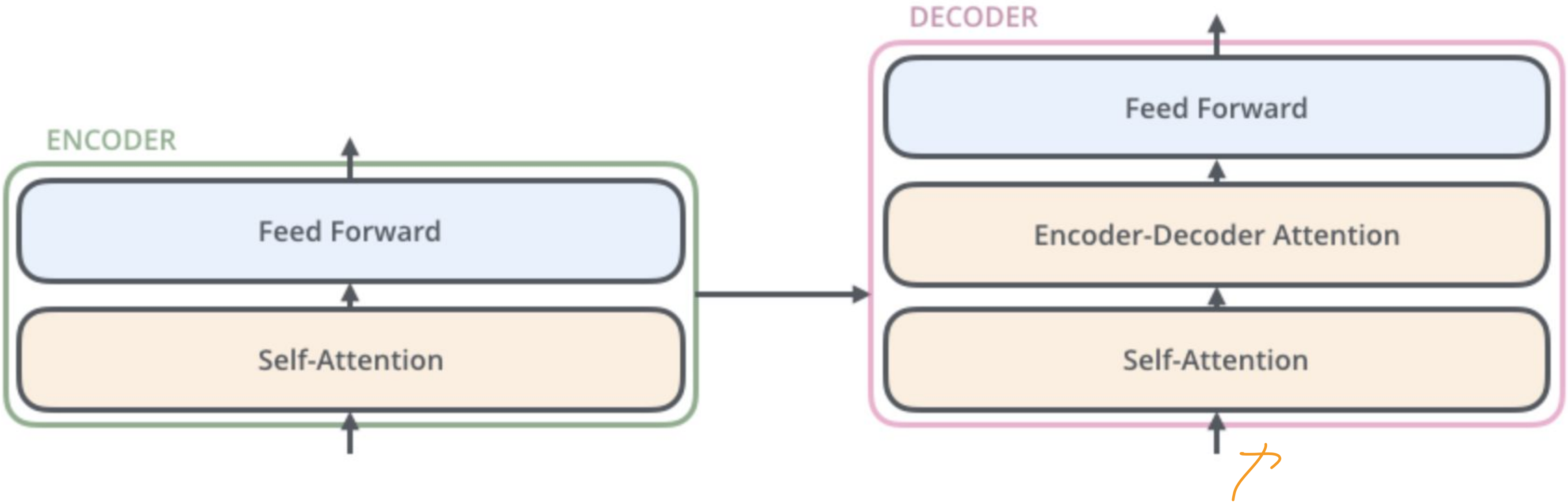


Figure: [Jay Alammar]

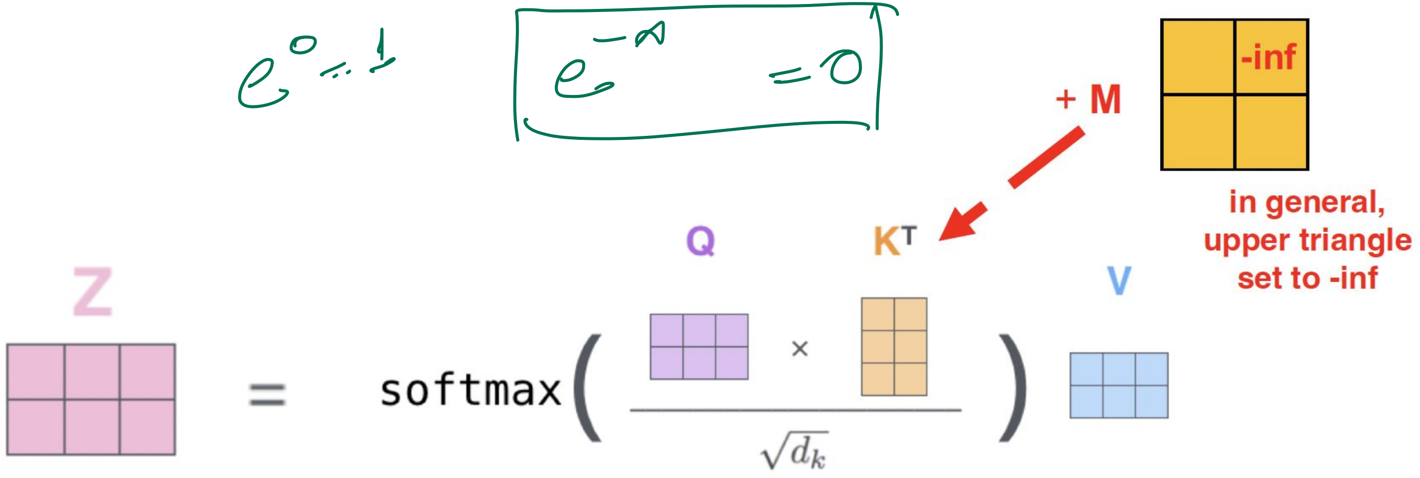
Masked Self-attention

In the **decoder**, the self-attention layer is only allowed to attend to **earlier positions** in the output sequence. Otherwise, we'd be cheating!

This is done by **masking future positions** **(setting the dot product score for future positions to -inf)** before the Softmax step in the self-attention calculation.

$e^0 = 1$

$\boxed{e^{-\infty} = 0}$



https://rycolab.io/classes/llm-s24/

Masked Self-attention for decoder (to avoid seeing the future tokens)

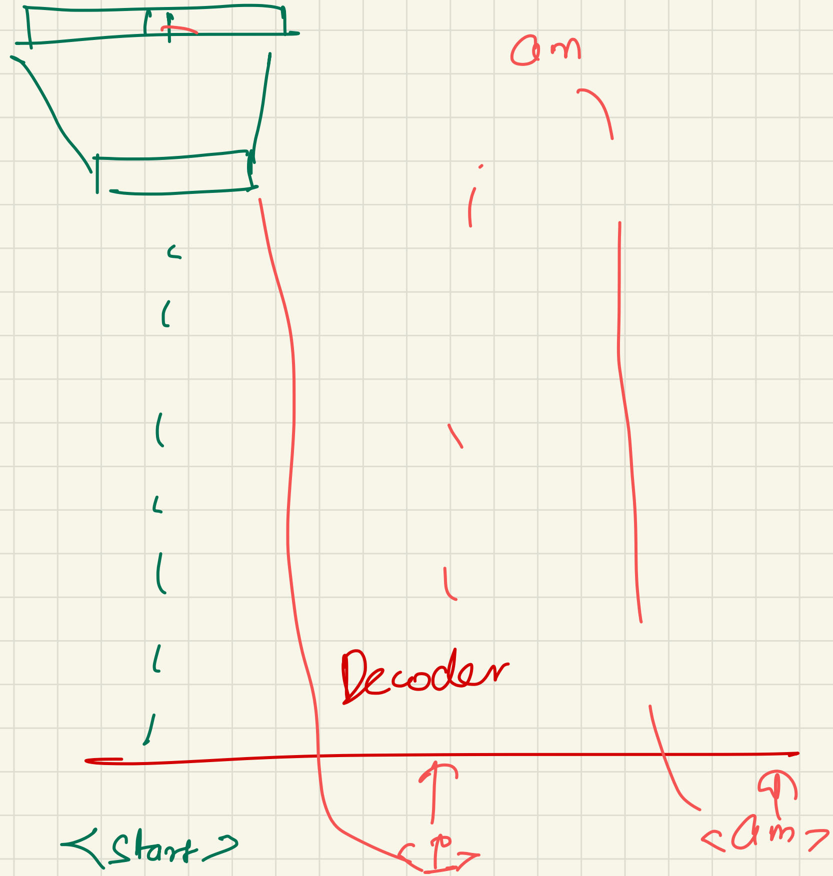
	1	2	3	4	5
1	q1•k1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	q2•k1	q2•k2	$-\infty$	$-\infty$	$-\infty$
$\underline{N}$	q3•k1	q3•k2	q3•k3	$-\infty$	$-\infty$
	q4•k1	q4•k2	q4•k3	q4•k4	$-\infty$
	q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

$\underline{N}$

Masked

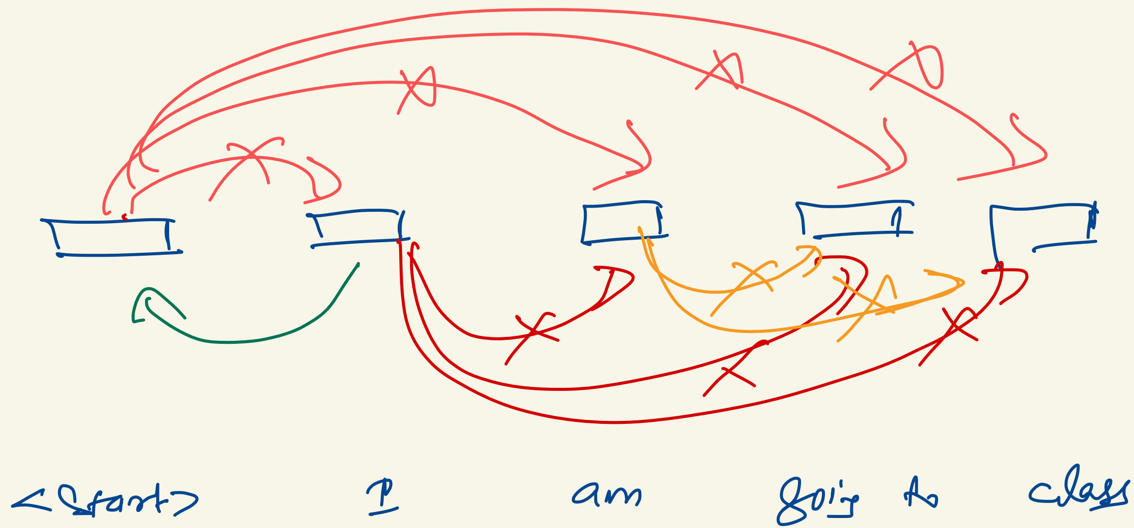
self-attn matrix

https://web.stanford.edu/~jurafsky/slp3/

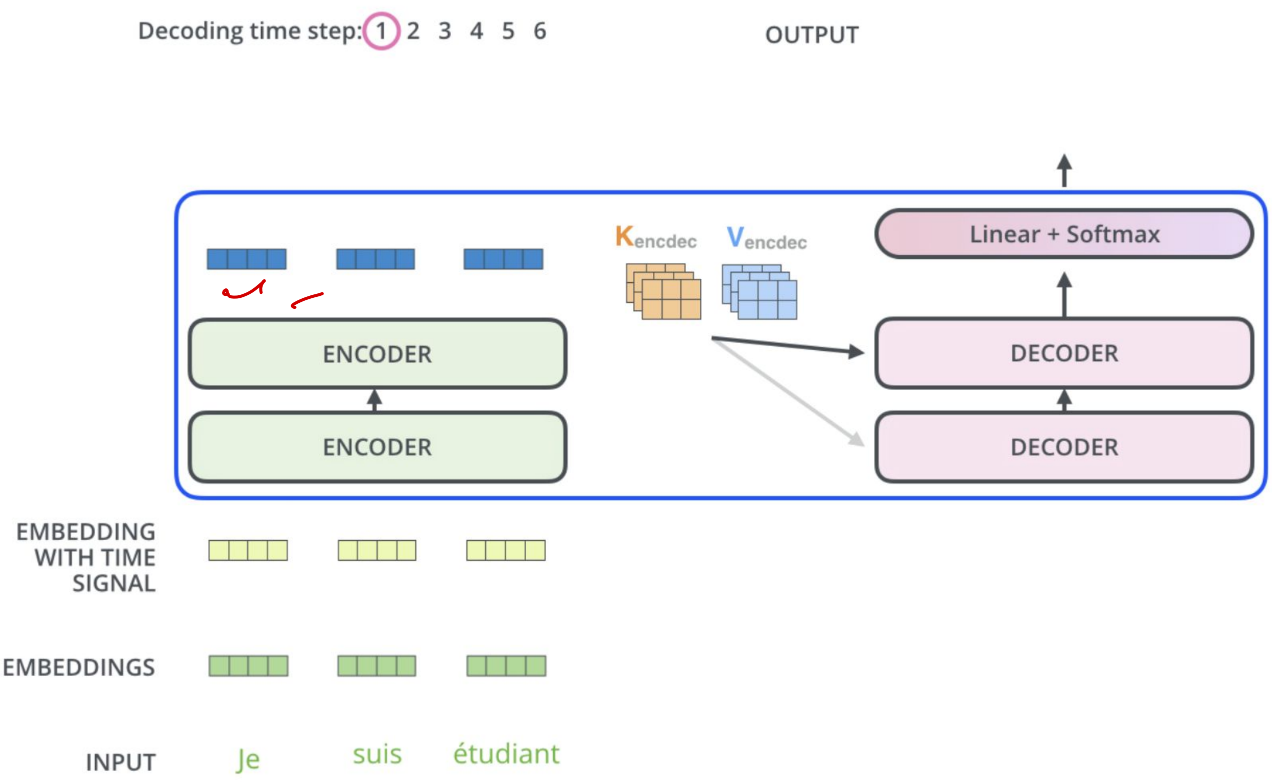


No masking
at inference
time

last
decoder



Encoder-Decoder Attention

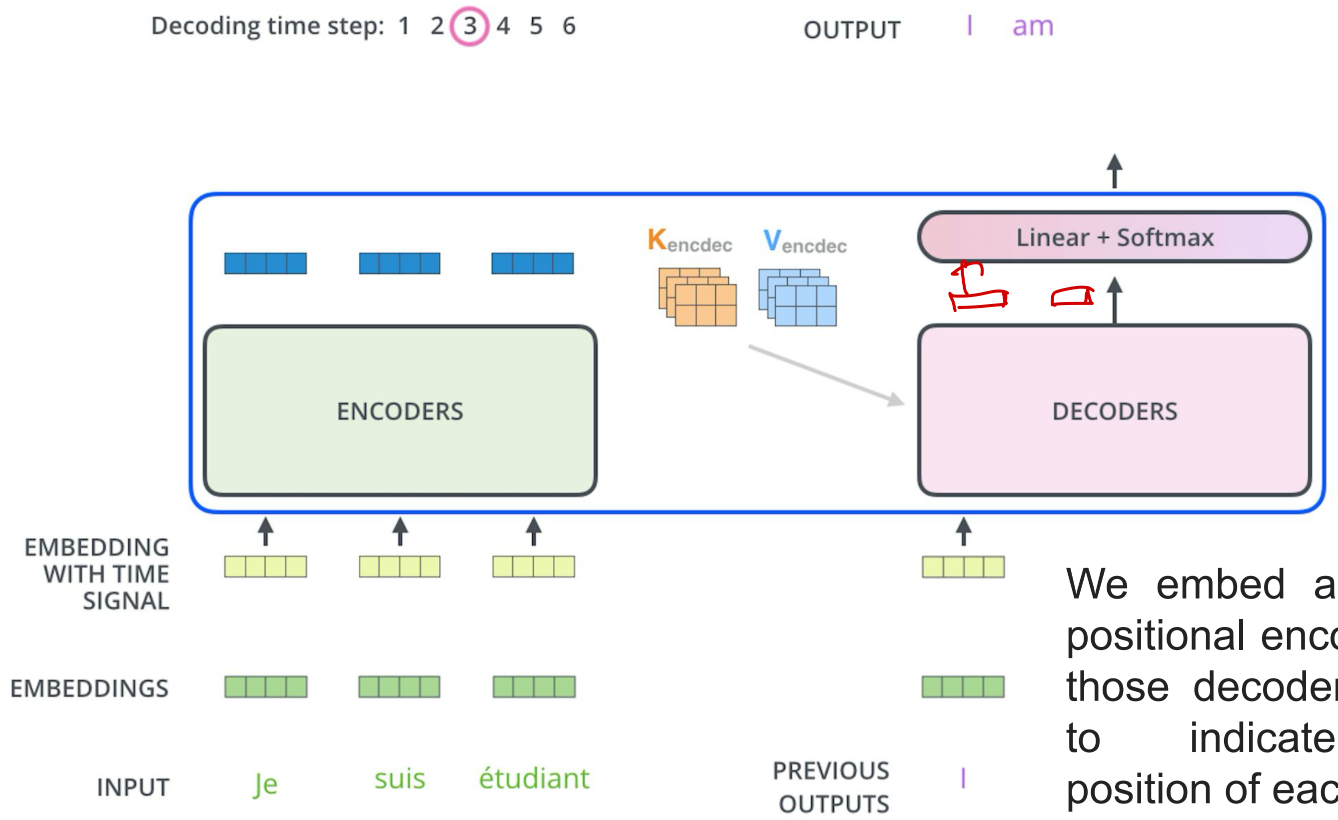


The output of the top encoder is transformed into a set of attention vectors K and V.

These are to be used by each decoder in its "***encoder-decoder attention***" layer which helps the decoder focus on appropriate places in the input sequence:

Figure: [Jay Alammar]()

Decoding



The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did.

We embed and add positional encoding to those decoder inputs to indicate the position of each word.

Figure: Jay Alammar

Converting decoder stack output to words

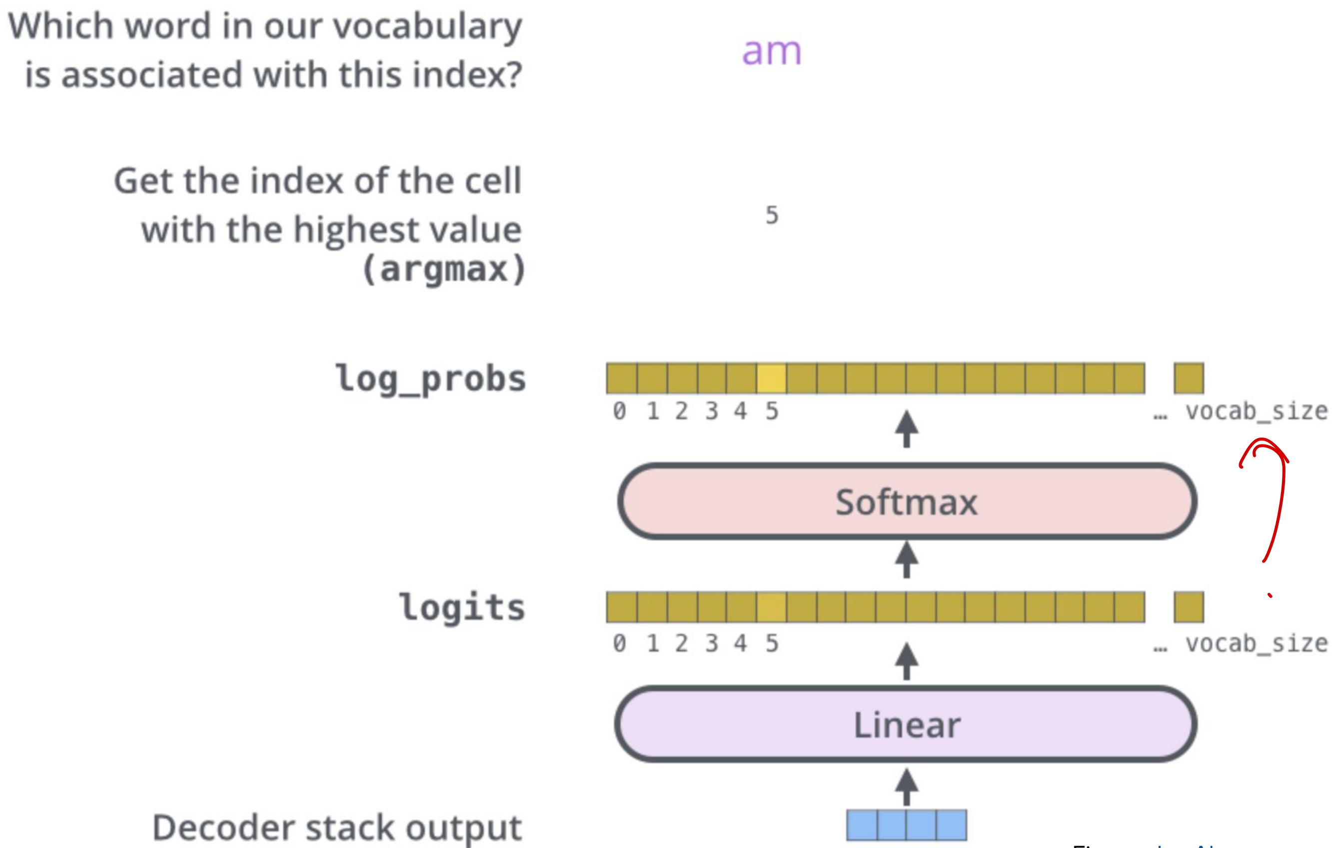


Figure: [Jay Alammar]

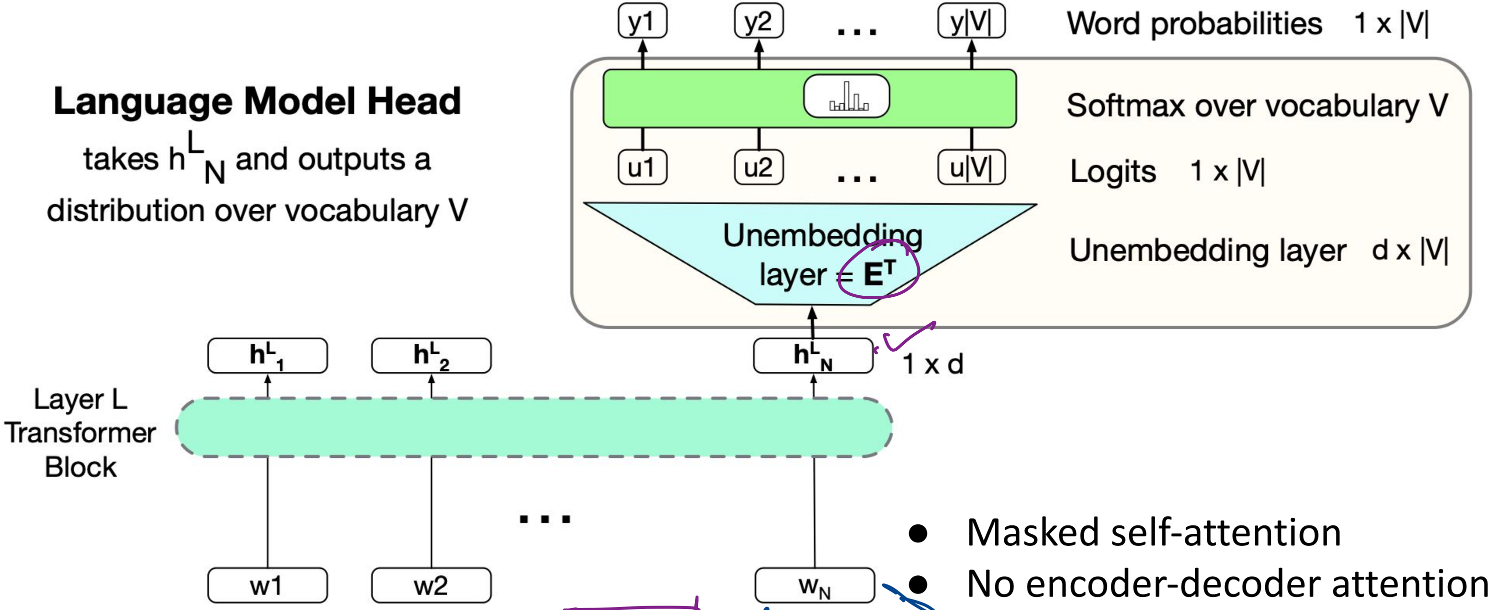
That's the job of the final ***Linear layer*** which is followed by a ***Softmax Layer***.

Encoders

Encoder-Decoder

Decoders?

Transformers as Language models (Decoder)



- Masked self-attention
- No encoder-decoder attention

$\boxed{V \times d}$

E — Embedding matrix

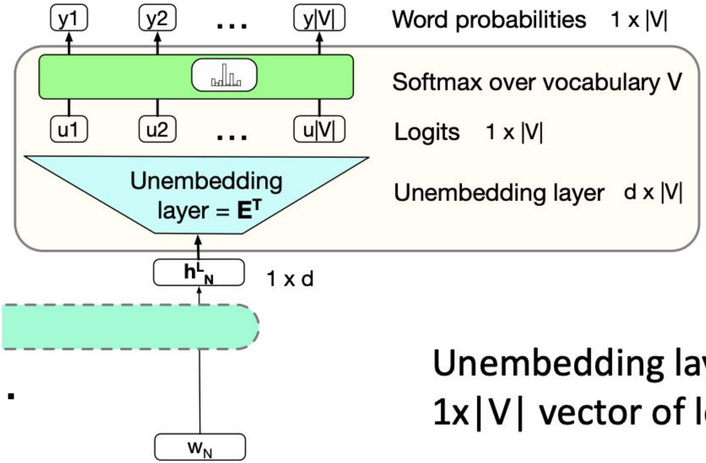
1-hot

(learnable with the model)

https://web.stanford.edu/~jurafsky/slp3/

Transformers as Language models

Unembedding layer: linear layer projects from h^L_N (shape $[1 \times d]$) to logit vector



Why "unembedding"? **Tied** to $\mathbf{E}^T$

Weight tying, we use the same weights for two different matrices

Unembedding layer maps from an embedding to a 1x|V| vector of logits

https://web.stanford.edu/~jurafsky/slp3/

Language modeling head

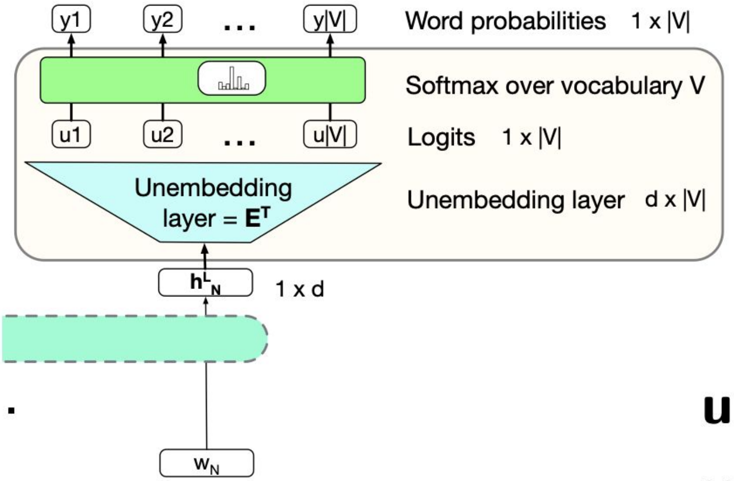
RNN LM

Transformer LM

Logits, the score vector u

One score for each of the $|V|$ possible words in the vocabulary V. Shape $1 \times |V|$.

Softmax turns the logits into probabilities over vocabulary. Shape $1 \times |V|$.

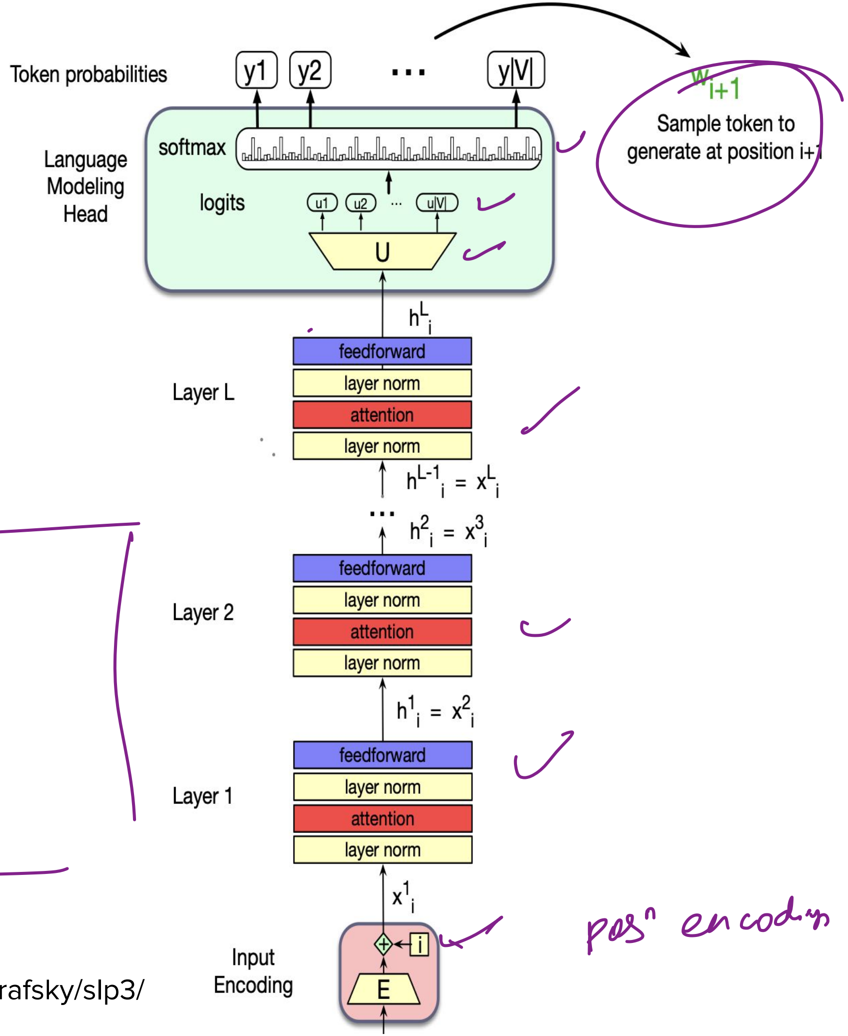


$$\mathbf{u} = \mathbf{h}^L_N \ \mathbf{E}^T$$

$$\mathbf{y} = \text{softmax}(\mathbf{u})$$

https://web.stanford.edu/~jurafsky/slp3/

Transformers LM



posⁿ encoding (Encoder/decoder)

https://web.stanford.edu/~jurafsky/slp3/

Number of parameters in encoder-decoder?

6 Layers – Encoders
Decoders

$d = 512$

$V = 40K$

$\#$ heads $= 8\ (h)$ $\quad d_k = d_q = 64$ $\quad FFN_{up} = 2048$

Embedding: $512 \times 40K$ (= Unembedding) $\approx \boxed{20m}$ (weight tying)

Encoder: self-attn $:\Rightarrow$ $\underbrace{h * \left(3 * d \times \frac{d}{h}\right)}_{W_q,\, W_k,\, W_v} + \underbrace{h * \frac{d}{h} * d}_{W_o} = 4d^2$

FFN $:\rightarrow$ $d \times 4d + 4d \times d = 8d^2$

$+$

$12d^2$

$12 \times 512 \times 512 \times 6$

$3m \times 6 = 18m$

$\boxed{12d^2 \times L} \rightarrow$ encoders

Number of parameters in encoder-decoder?

Decoder

self-attn : $4d^2$

feed-forward : $8d^2$

encoder-decoder : $4d^2$

$16d^2$

assumes a default conf

$h \times d_v = d$! — 1

Up-proj $= 4d$ — 2

$16d^2 \times L$

$4m \times 6 = 24m$

$18m + 24m + 20m$ emb $\approx 62m$

Number of parameters in the decoder only Transformer?

Same as encoder

(18m)

+

embedding / unembedding

GPT-3 (175 B)

$\boxed{12d^2 \times L}$

$d = \underbrace{12288}$ $L = \underline{96}$

Try this one

Decoding: Greedy and Beam Search are deterministic!

- Greedy decoding as well as Beam Search decoding will give a “deterministic” output
- Other common decoding algorithms involve “sampling”, and bring in some degree of “randomness”

$x \sim p(x) \rightarrow$ choose x by sampling from the distribution $p(x)$

Random Sampling

$i \leftarrow 1$
$w_i \sim p(w)$
while w_i != EOS
  $i \leftarrow i + 1$
  $w_i \sim p(w_i \mid w_{<i})$

Quality vs Diversity trade-off

Various sampling methods enable trading off two important factors in generation: *quality* and *diversity*.

Quality vs Diversity trade-off

- Methods that emphasize the most probable words tend to produce more coherent and accurate generations but also tend to be repetitive and boring
- Methods that give bit more weight to the middle probability words tend to be more creative and diverse, but likely to be incoherent and less factual

Random Sampling with Temperature

Intuition from thermodynamics

A system at a *high temperature* is flexible and can explore various states, while a system at a *low temperature* is likely to explore a subset of lower energy (better) states

How is this implemented

Divide the logits by a temperature parameter $\tau \in (0, 1]$ before passing it through softmax

Random sampling: $y = softmax(u)$

Random sampling with temperature: $y = softmax(u/\tau)$

$u = [3, 1, -1]$

$e^3 \quad e \quad e^{-1}$

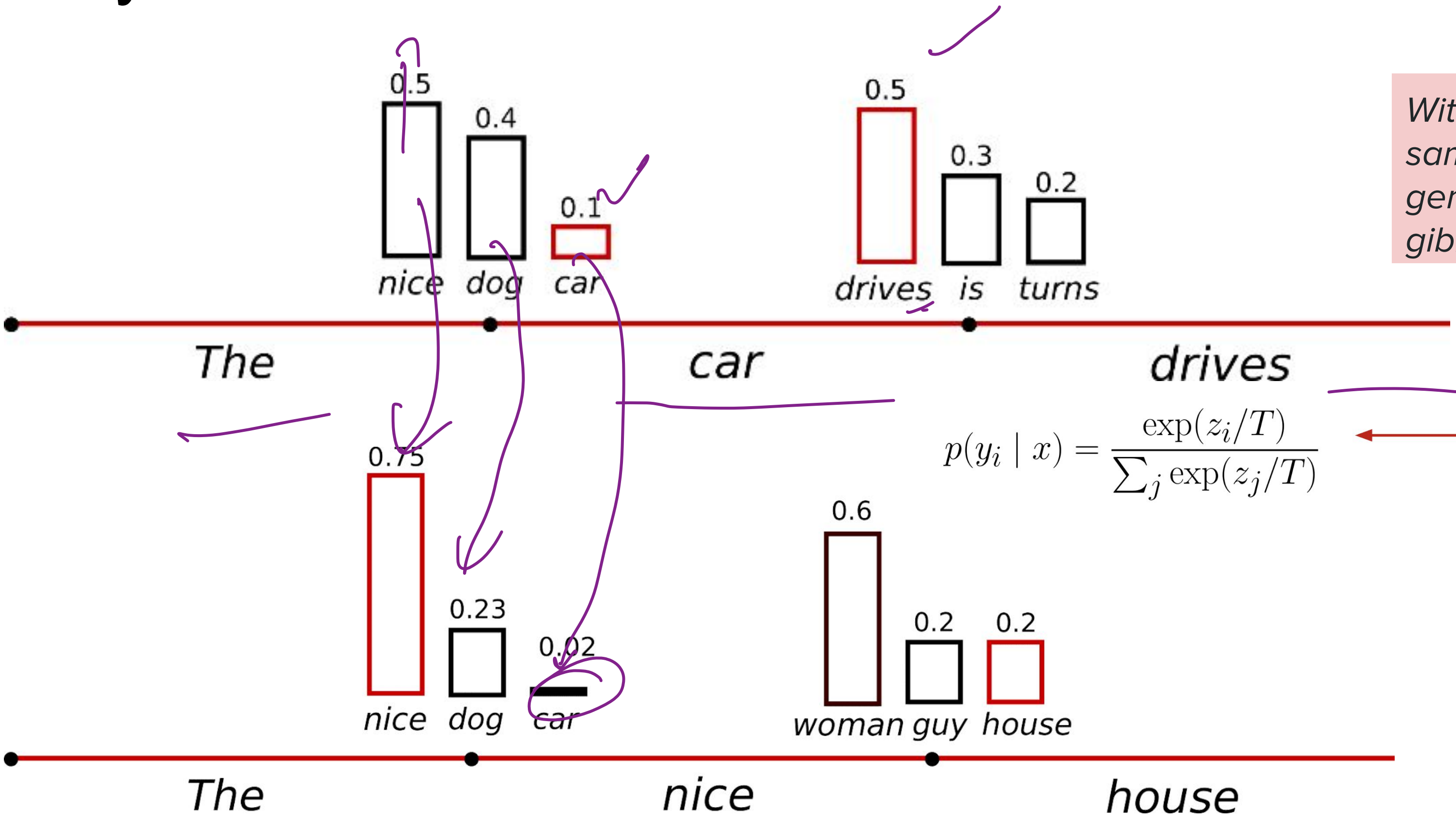
$u/\tau = [15, 5, -5]$

$e^{15} \quad e^5 \quad e^{-5}$

0.99

$\tau = 0.2$

Why does this work?



Without scaling, random sampling often generates incoherent gibberish

$$p(y_i \mid x) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

Higher T: softens probabilities.
Lower T: sharpens probabilities.

https://utah-cs6340-nlp.notion.site/Natural-Language-Processing-bd1a2ca290fc44f69556908ad8d25c70