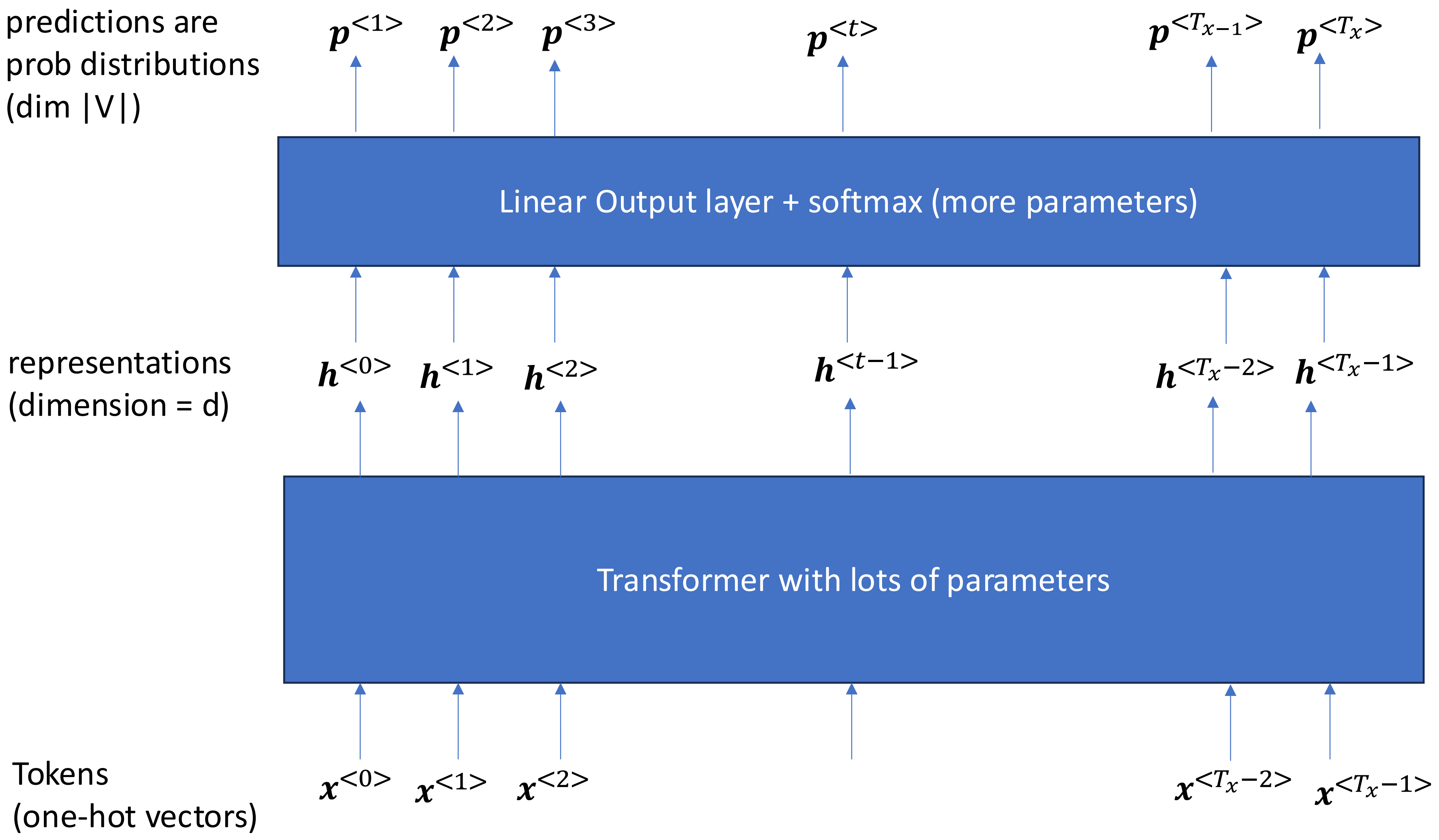
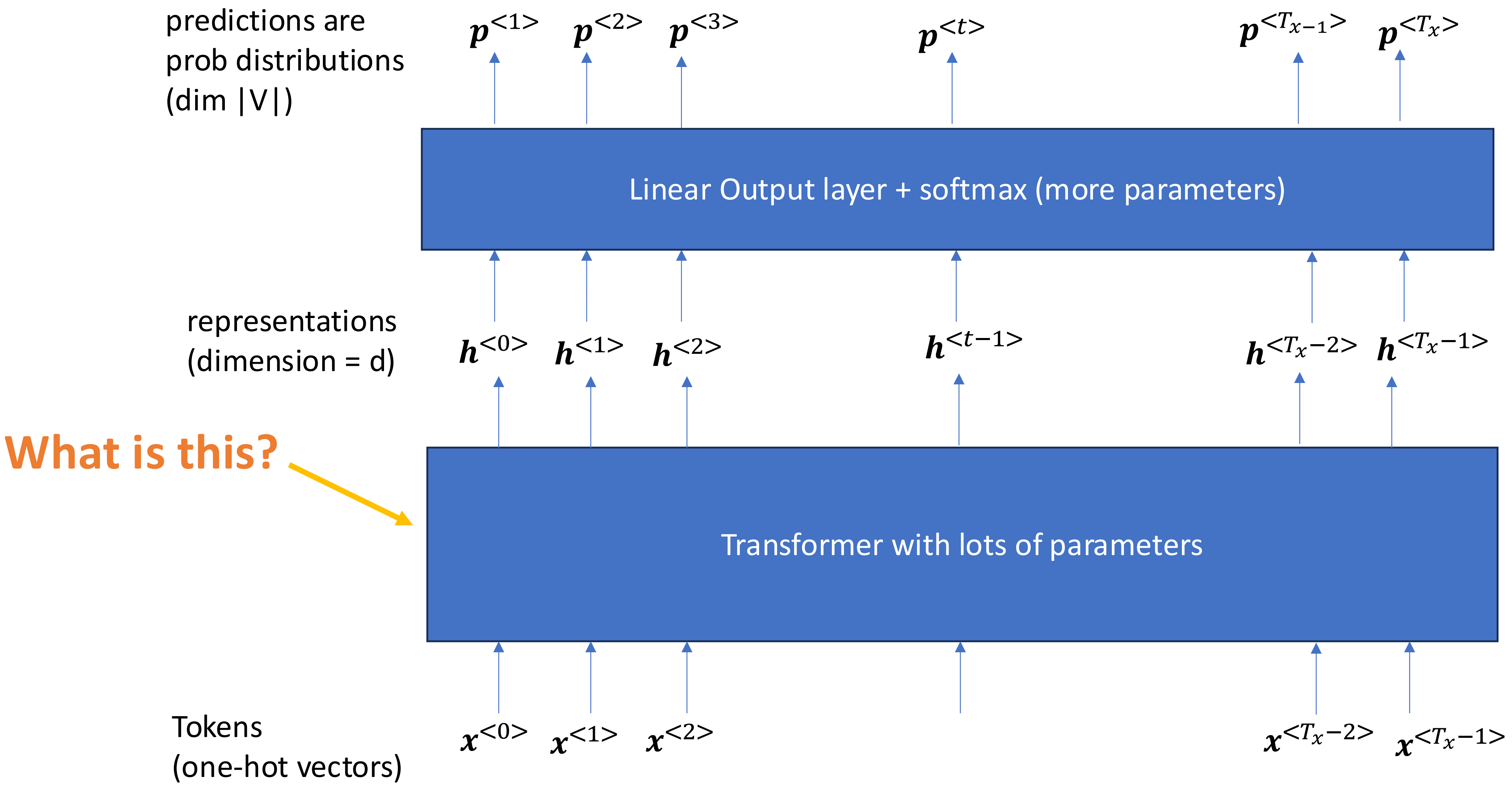
Recall LLM structure:

L(θ) = -[log p_θ("I")+ log p_θ("love" | "I")+ log p_θ("music" | "I love") + log p_θ("<EOS>" | "I love music")]

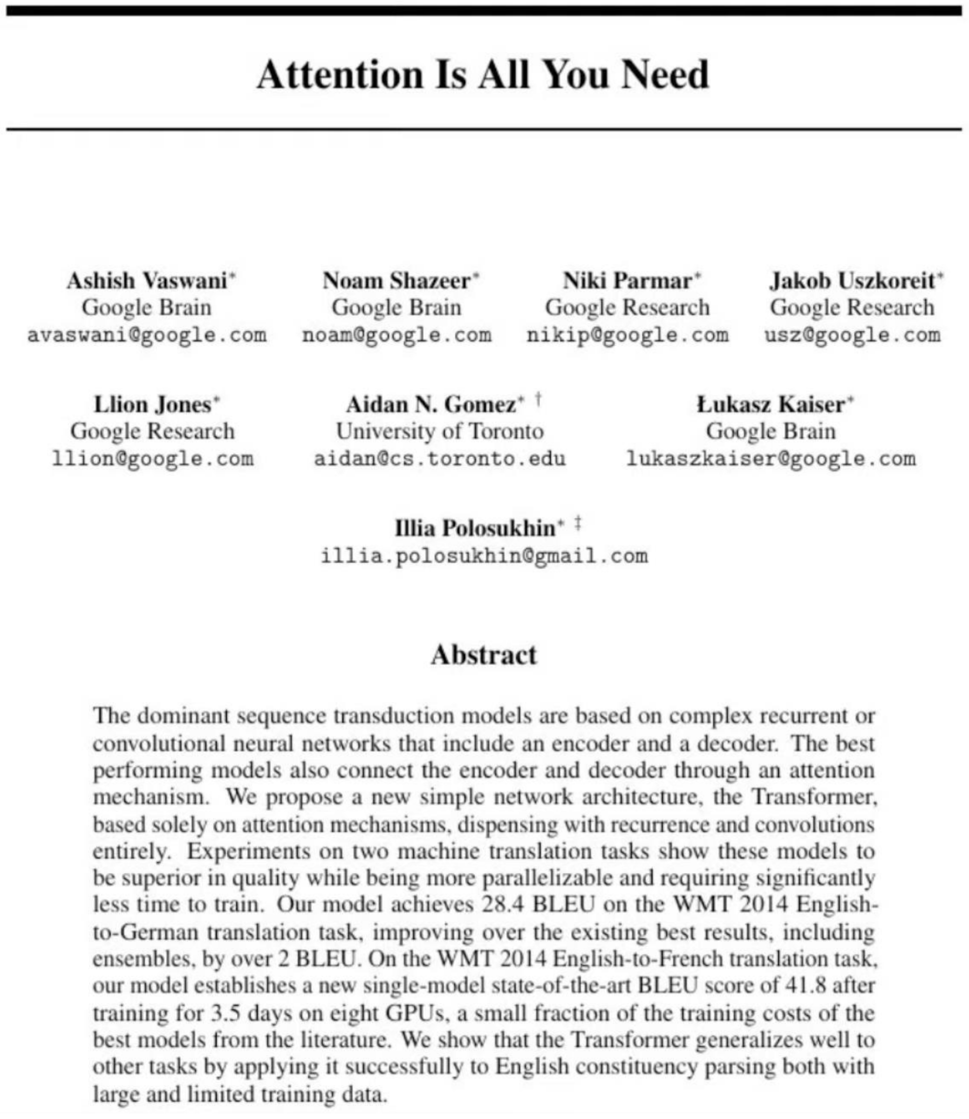


Recall LLM structure:

$L(\theta) = -[\log p_\theta(\text{"I"}) + \log p_\theta(\text{"love"} \mid \text{"I"}) + \log p_\theta(\text{"music"} \mid \text{"I love"}) + \log p_\theta(\text{"<EOS>"} \mid \text{"I love music"})]$


predictions are prob distributions (dim |V|)
$p^{<1>}$ $p^{<2>}$ $p^{<3>}$ $p^{<t>}$ $p^{<T_{x-1}>}$ $p^{<T_x>}$
Linear Output layer + softmax (more parameters)
representations (dimension = d)
$h^{<0>}$ $h^{<1>}$ $h^{<2>}$ $h^{<t-1>}$ $h^{<T_x-2>}$ $h^{<T_x-1>}$
What is this?
Transformer with lots of parameters
Tokens (one-hot vectors)
$x^{<0>}$ $x^{<1>}$ $x^{<2>}$ $x^{<T_x-2>}$ $x^{<T_x-1>}$

Famous 2017 Transformer paper (~150K citations)



Attention Is All You Need

Ashish Vaswani[*] Google Brain avaswani@google.com

Noam Shazeer[*] Google Brain noam@google.com

Niki Parmar[*] Google Research nikip@google.com

Jakob Uszkoreit[*] Google Research usz@google.com

Llion Jones[*] Google Research llion@google.com

Aidan N. Gomez[*] [†] University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser[*] Google Brain lukaszkaiser@google.com

Illia Polosukhin[*] [‡] illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

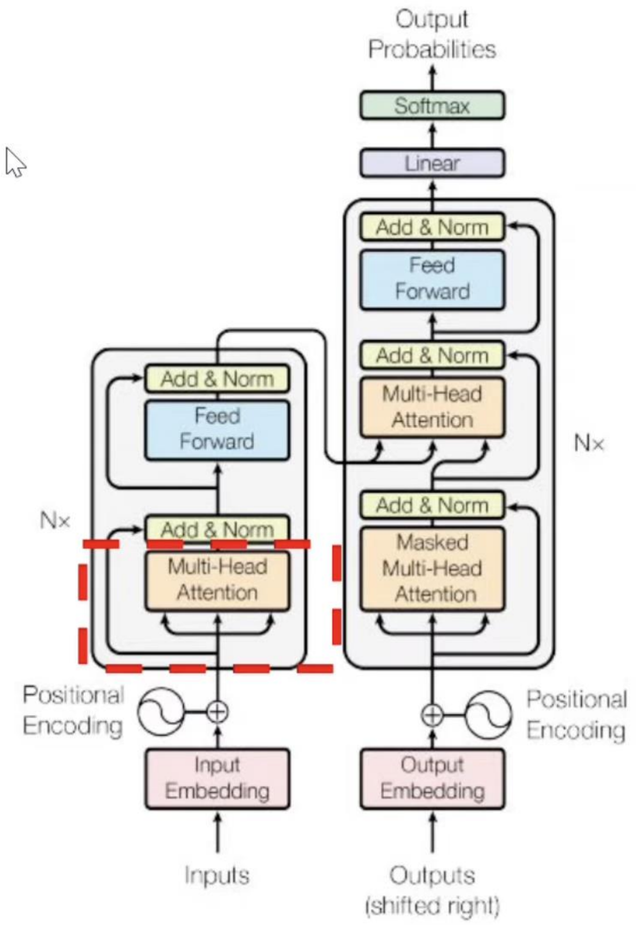
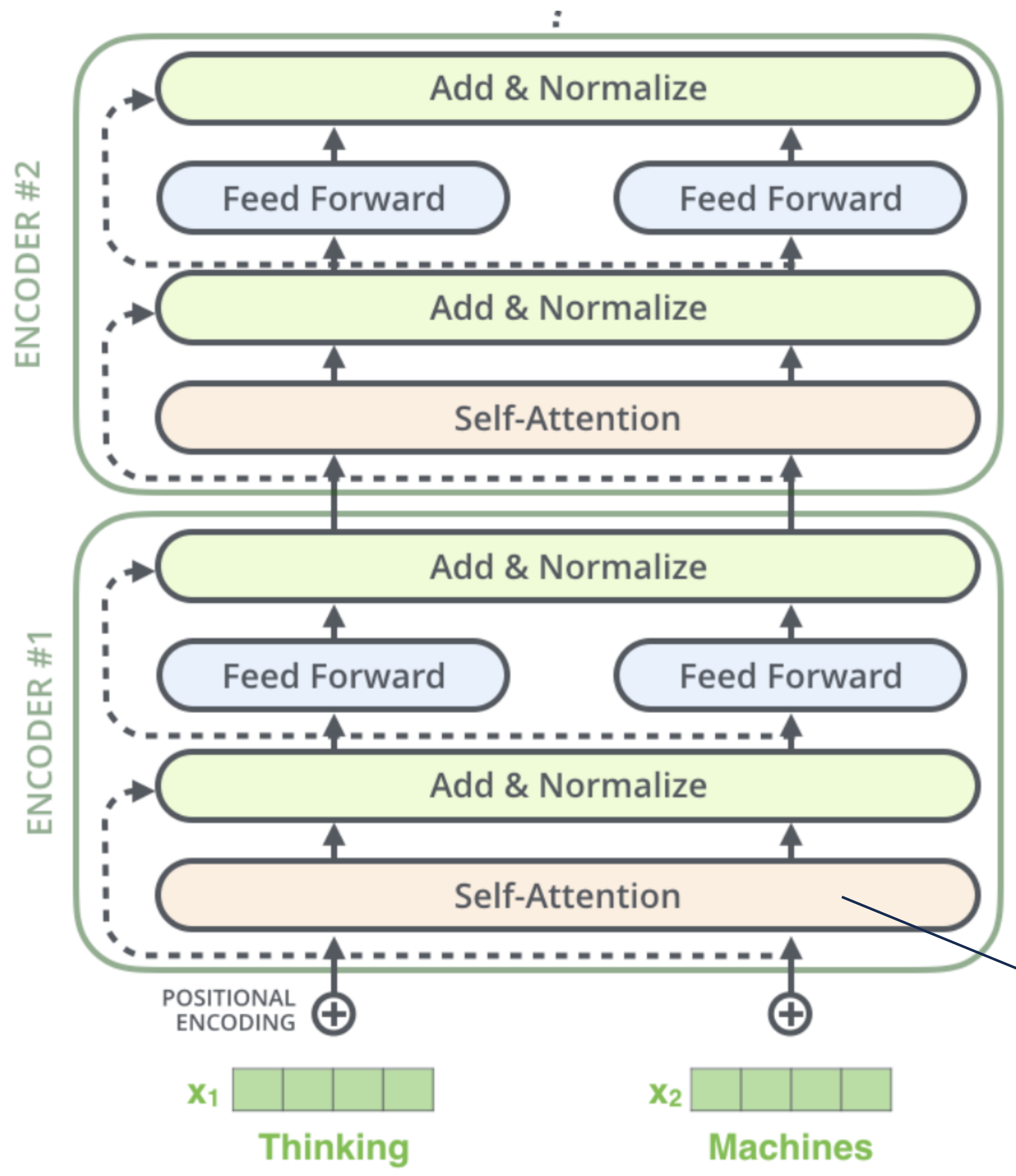


Figure 1: The Transformer - model architecture.

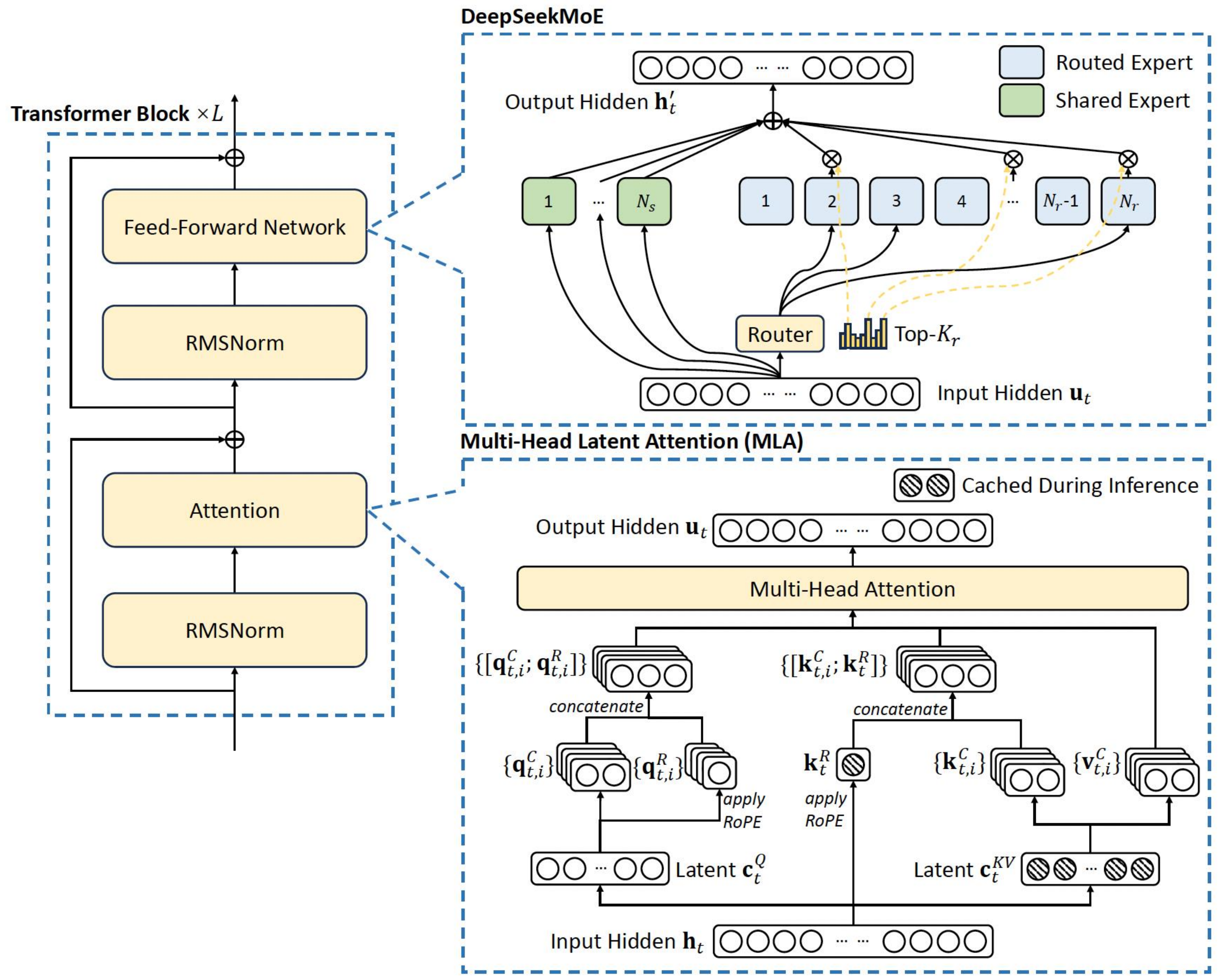
Attention is All You Need [Vaswani et al.]

Transformer encoder



First discuss this

Deepseek Transformer



Preamble: Word Embeddings

- One-hot vectors are sparse and have no semantic meaning.
- Word embeddings are dense and provide semantic meaning.
- One-hot vectors might have, say, dimension $|V| = 20{,}000$ and word embeddings might have, say, dimension 512.

Word Embeddings: An Illustrative Algorithm

- Created from a large corpus.
- Example algorithm:
  - Let x_{ij} be the number of sentences in corpus for which words i and j both appear
  - For word i, create preliminary representation $r_i = (x_{i1}, x_{i2},\dots,x_{i|V|})$.
    - Suppose i is book, j is car, k is library. In the corpus there 303 sentences with book and car; there are 1027 with book and library. What is x_{ij} and x_{ik}?
  - Then use PCA to reduce the dimension of all the representations to 512.
- The above approach just gives the rough idea. In practice there are better ways.

Word Embeddings: Summary

- Instead of one-hot vector, represent word with a dense vector, say, of dimension d =512.
  - small models d=512 (BERT); very large models d $\geq$ 4096
- The fixed length vector is called a **representation** or a **word embedding**
- Word embedding has contextual information: similar words tend to be close to each other in the embedding space.
- Popular embedding algorithm: Word2Vec
- (Transformer doesn’t typically use word embeddings. But for now let’s suppose it does.)

Preamble: Homographs

```
I swam across the river to get to the other bank
```

homograph

Bank == financial institution?
Bank == sloping raised land?

You shall know a word by the company it keeps!
J.R.Firth(1957)

Problem with word embeddings

Same embedding for both contexts. So does not fully capture meaning.

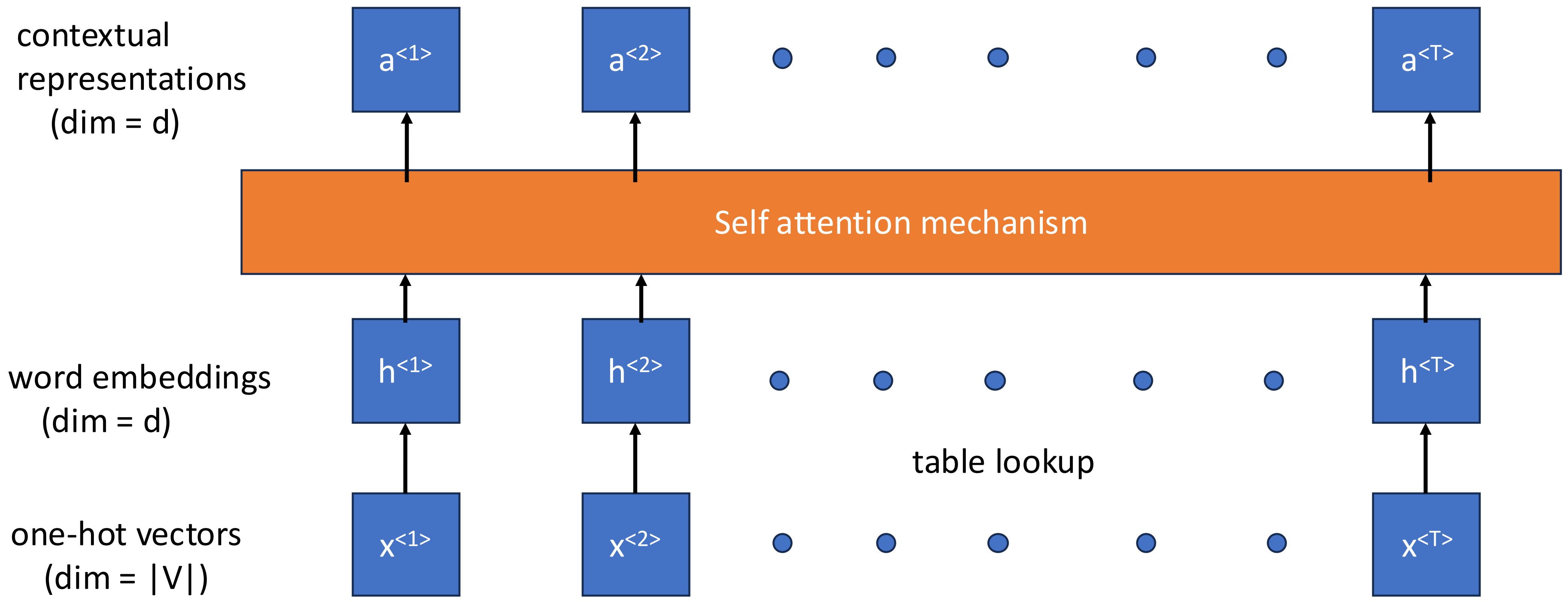
I swam across the river to get to the other bank

Self Attention

- We will transform the embeddings to get representations that take the context of the current sentence into account.
- An embedding for a word will no longer be fixed; instead it depends on the sentence it appears in.

- “Self” because words in the same sentence attend to each other.

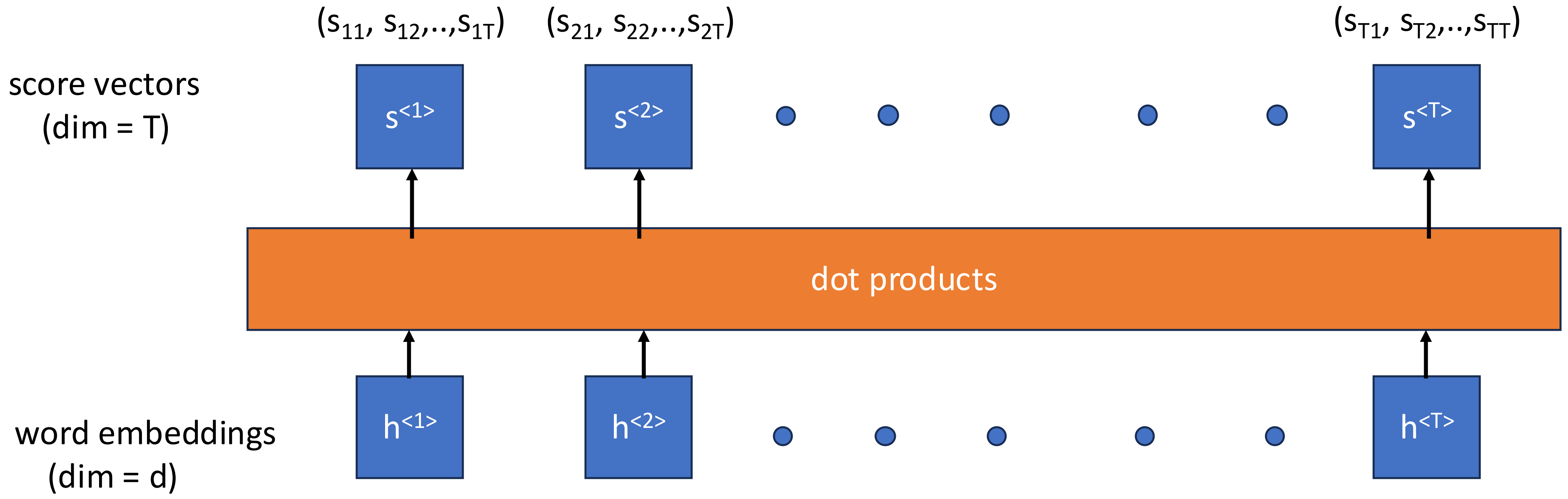
Self attention: Big Picture



From word embeddings to contextual representations

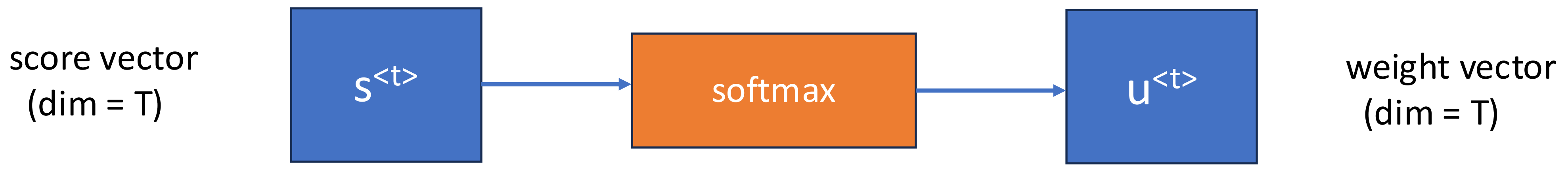
- Assume word embeddings are normalized, all with length one.
- For now assume no parameters.
- Calculate all the dot products (cosine distance) among the T word embeddings to get “scores”: s_{ij}, $1 \leq i,j \leq T$
- Weigh embeddings by scores to get contextual representation.

(1) get score vectors from dot products


$(s_{11}, s_{12},..,s_{1T})$
$(s_{21}, s_{22},..,s_{2T})$
$(s_{T1}, s_{T2},..,s_{TT})$
score vectors (dim = T)
$s^{<1>}$
$s^{<2>}$
$s^{<T>}$
dot products
word embeddings (dim = d)
$h^{<1>}$
$h^{<2>}$
$h^{<T>}$

- Q: Can a score s_{ij} be negative? How many scores are there?
- Q: What is the dimension of the score vector $s^{<t>} = (s_{t1}, s_{t2},..,s_{tT})$
- Q: Which scores will be the larges

(2) Apply softmax to score vectors to get attention weight vectors



- Q: What is the dimension of $u^{<t>}$?
- Q: Can a weight u_{ij} be negative?
- Q: $u_{i1} + u_{i2} + \ldots + u_{iT} = ??$
- Q: Among u_{i1} , u_{i2} , … , u_{iT} , which will be the largest?

- The value of u_{ij} indicates how much word $x^{<i>}$ relates to $x^{<j>}$.
- It indicates how much ”attention” word $x^{<i>}$ should give to word $x^{<j>}$

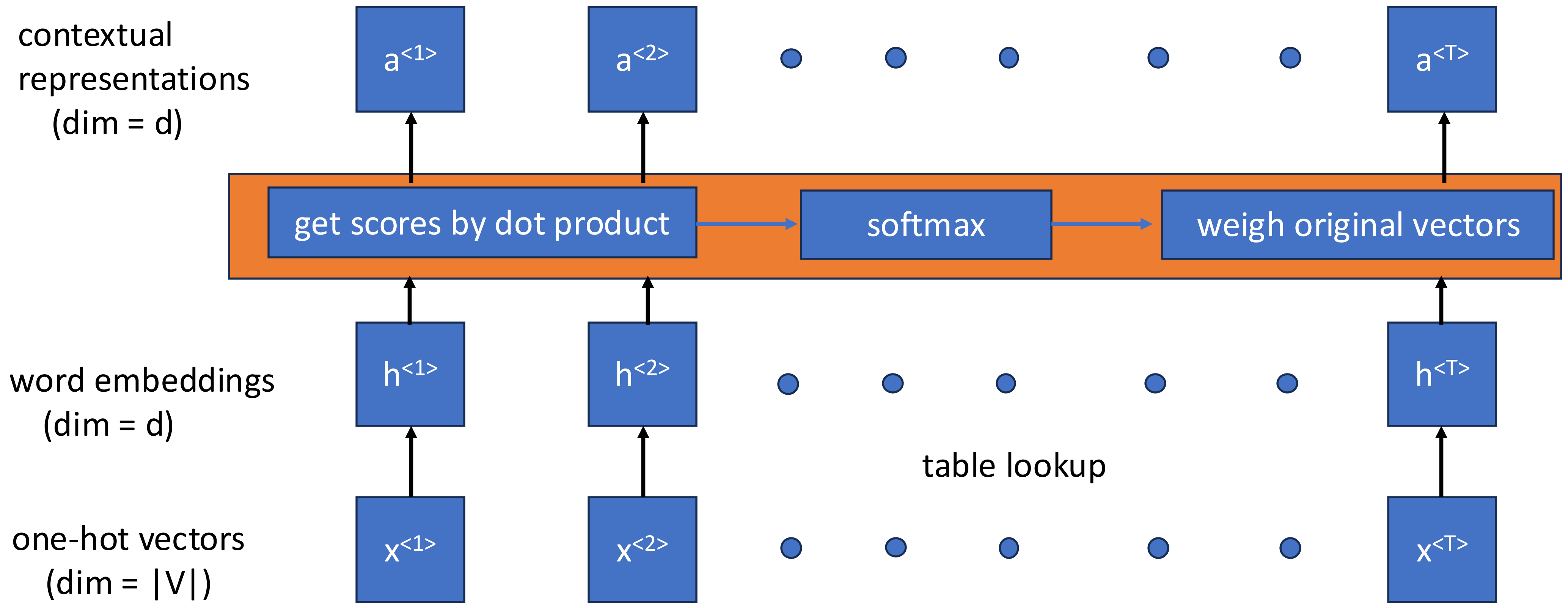
(3) Calculate the representation vectors

- Set the representation $a^{<t>}$ to be a weighted combination of the T embedding vectors $h^{<1>}$, $h^{<2>}$,…,$h^{<T>}$:

.
$$a^{<t>} = u_{t1}\, h^{<1>} + u_{t2}\, h^{<2>} + \ldots + u_{tT}\, h^{<T>}$$

- Recall that the weights $u_{t1}, u_{t2}, .., u_{tT}$ are positive and sum to 1.
- So the representation vector $a^{<t>}$ is a weighted combination of the T embeddings.
- Q: What is the dimension of $a^{<t>}$?

Summary: Self-attention without parameters



Q: Is there any non-linear operation when going from the h's to the a's?

But can we do even better?

- The weights so far are not learned; they are instead determined by a fixed formula.
- How can we incorporate learnable parameters?
- Useful to introduce query, key, value terminology (database terminology)
- In standard databases, each key has a value. Given a query, you match it to a key, and return the associated value. Here things are a little different…

Learnable Matrices

- Introduce three learnable dxd matrices: M_Q, M_K, M_V
- Calculate $Q^{<i>} = M_Q h^{<i>}$, $K^{<i>} = M_K h^{<i>}$, $V^{<i>} = M_V h^{<i>}$,
- When calculating inner products, instead of $s_{ij} = (h^{<i>} \cdot h^{<j>})$, use $s_{ij} = (Q^{<i>} \cdot K^{<j>})$
- Then apply softmax as before to s_{i2} , … , s_{iT} to get attention weights for word i: u_{i1} , u_{i2} , … , u_{iT}
- Calculate the representation as before, but now replace $v^{<j>}$ with $V^{<j>}$:

$$a^{<t>} = u_{t1} V^{<1>} + u_{t2} V^{<2>} + \ldots + u_{tT} V^{<T>}$$

- Note that M_Q, M_K, M_V are the same for all words and all sentences.
- $a^{<t>}$ is still a nonlinear function of the embeddings $h^{<1>},\ldots,h^{<T>}$, but now also depends on parameters in M_Q, M_K, M_V.

Additional trick: Scaling

Ignore this slide for exam

- Recall that d denotes the dimension of the embedding (eg, 512)
- Before the softmax, scale (i.e., multiply) each score by $^1/_{\sqrt{d}}$
- Has the effect of keeping magnitude of scores in roughly same range as original embedding vectors. Leads to better gradient signals.
- Do not worry about the “why” behind this step

How to increase the number of parameters?

- Similar to the channel idea (multiple filters) for ConvNets, we can have multiple $M_Q^{(j)}$, $M_K^{(j)}$, $M_V^{(j)}$, j=1,…,J matrices: so-called "attention heads, " where J is the number of heads (typically $4 \leq J \leq 16$).
- Creates J contextual representations for the *ith* word in sentence:

$$a_1^{<i>}, a_2^{<i>}, \ldots, a_J^{<i>}$$

- Could combine the multiple representations (for a single token) by averaging them. In practice, we instead pass the multiple representations through a linear layer with learnable parameters $M_0^{(j)}$, j=1,…,J, to get final representation vectors $y_1, \ldots, y_n$:

$$a^{<i>} = \sum_{j=1}^{J} M_0^{(j)} a_j^{<i>}, \quad \mathrm{i} = 1, \ldots, \mathrm{T}$$

- Note that $M_0^{(j)}$ is also d x d for each j = 1,..,J

Quiz

- Question: Suppose an attention layer has J heads, the representation has dimension d, and there are n input tokens. How many parameters are there in the attention layer?

Answer

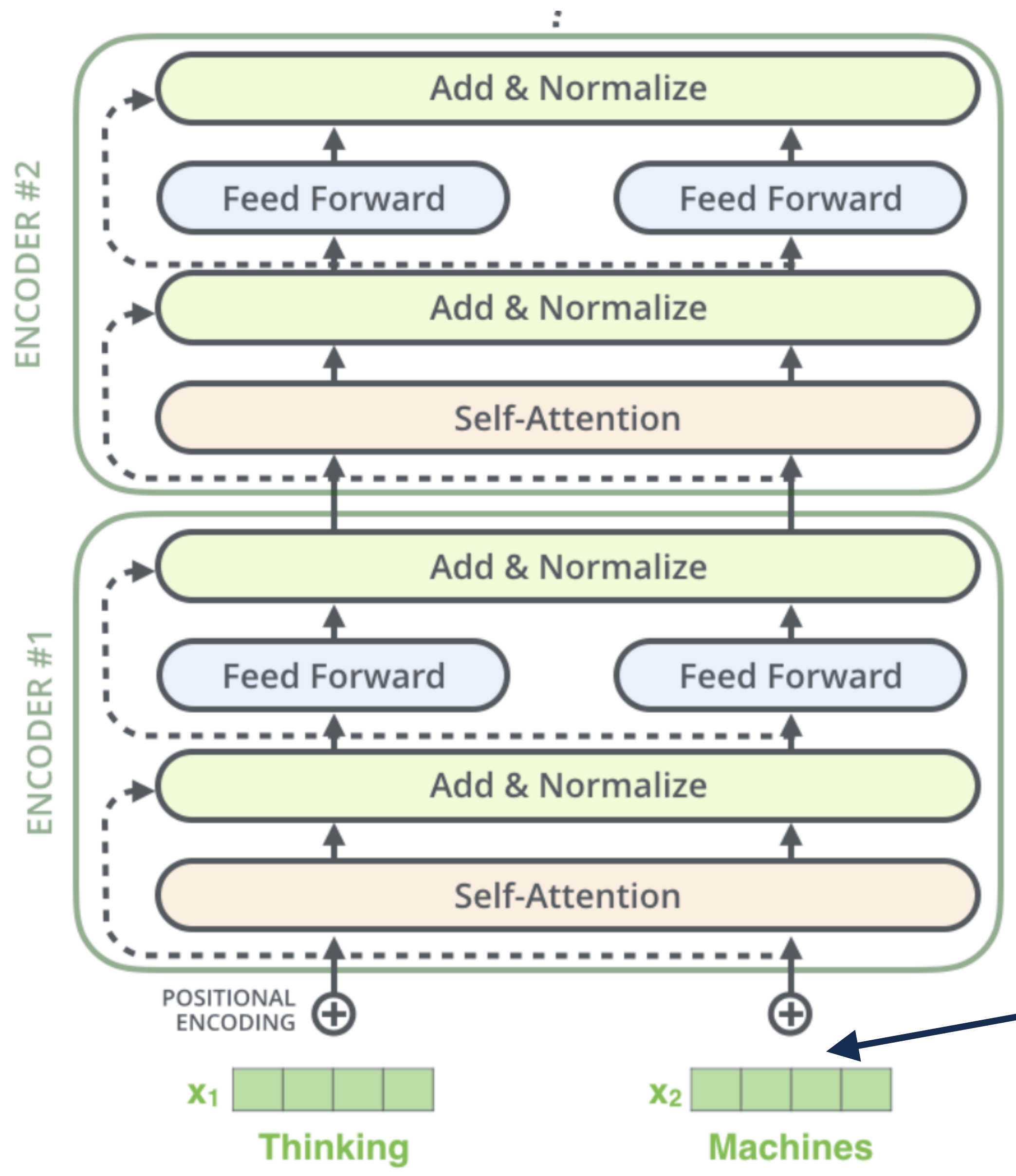
- Question: Suppose an attention layer has J heads, and the representation has dimension d. How many parameters are there in the attention layer?
- Answer: $J3d^2 + Jd^2 = 4Jd^2$. Does not depend on n.
- Example J = 12, d = 4096; # of parameters = 805,306,368

But the Transformer is more than just attention

- Projection layer
- Feed-forward network (MLP)
- Positional encoding
- Residual connections
- Layer normalization
- Multiple encoding layers

- *Some of the subsequent diagrams from the "Illustrated Transformer"*

Transformer encoder



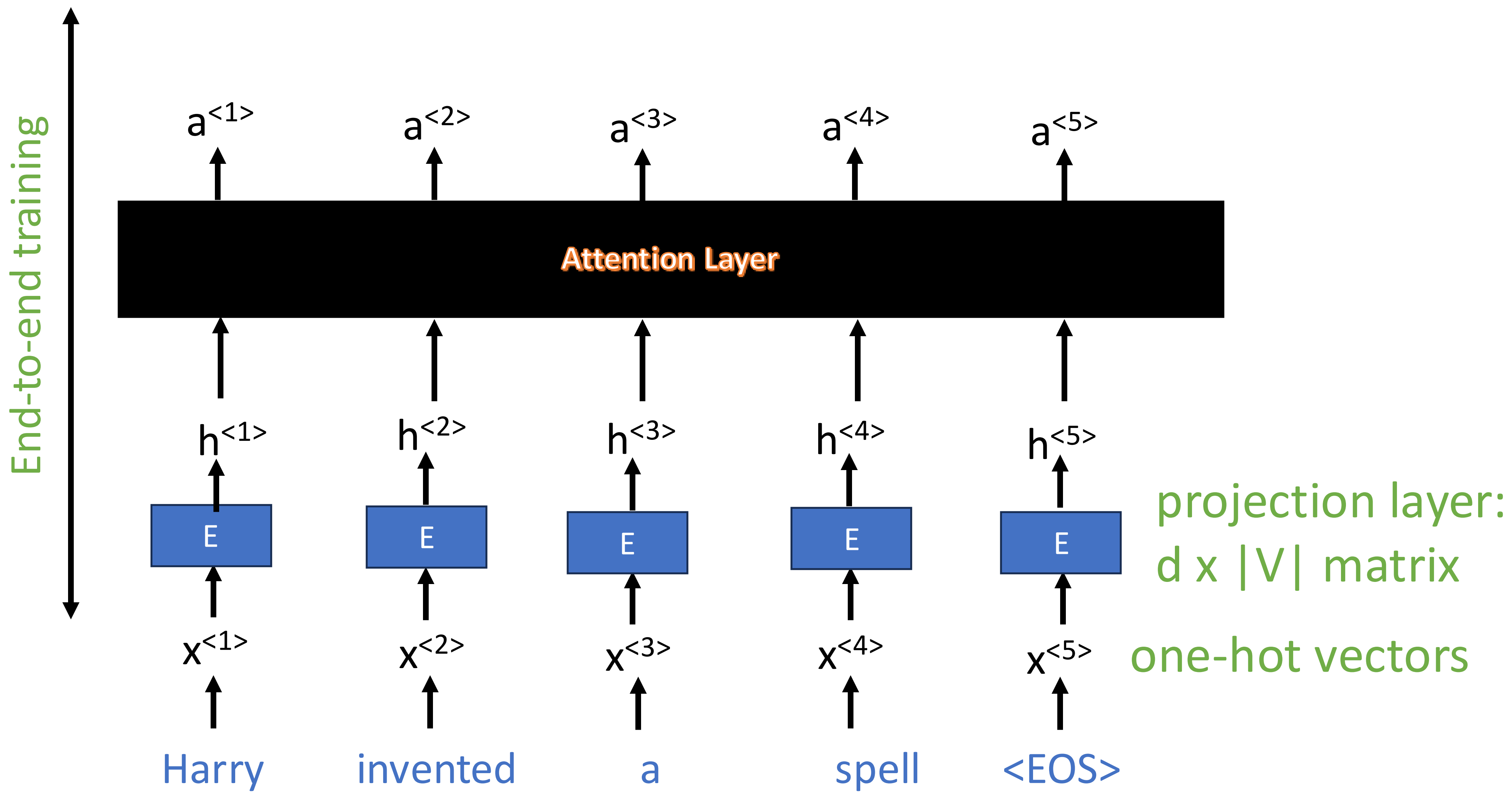
projection layer first

Transformer does not use word embeddings

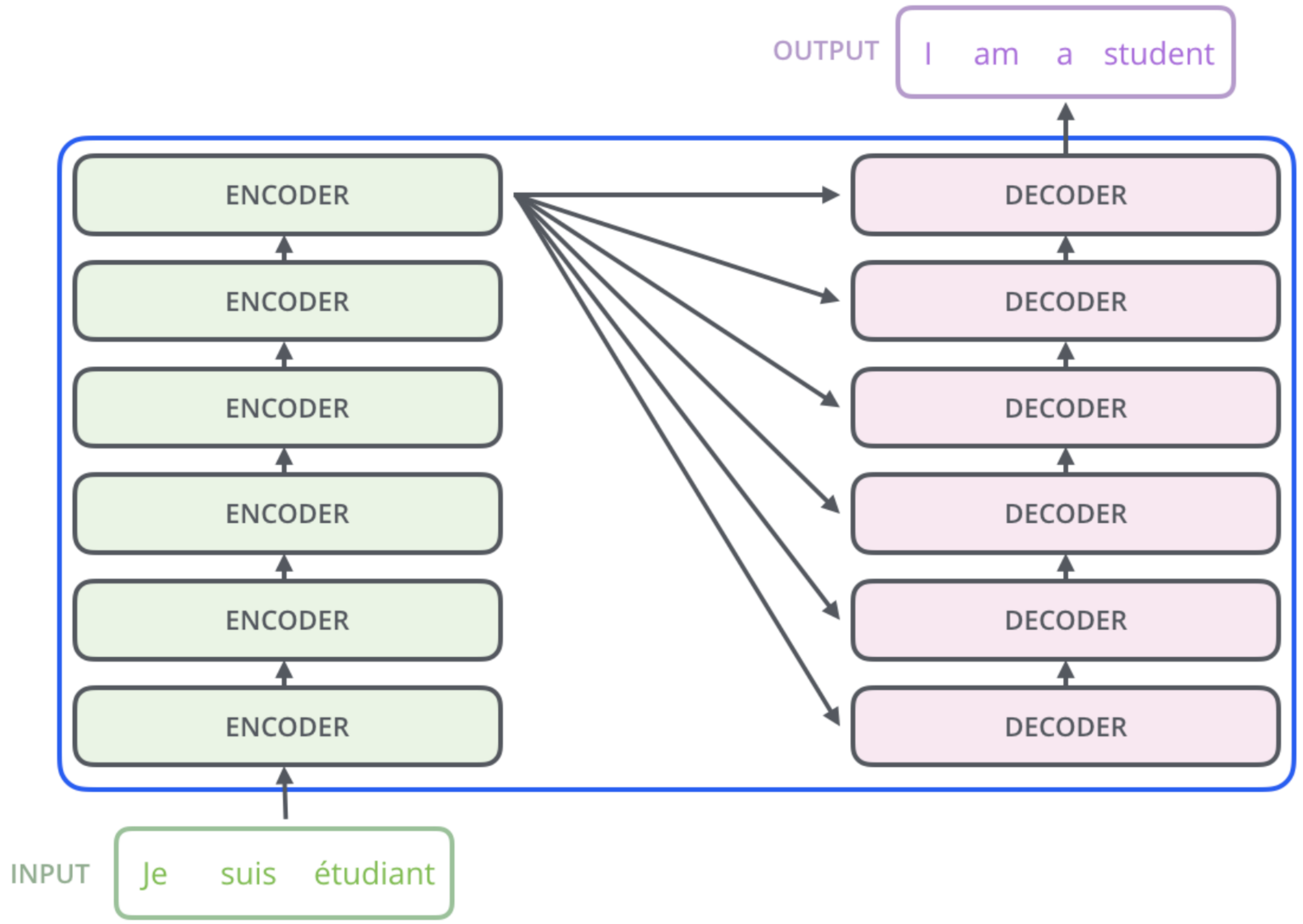
- Input 1-hot vectors (each of dim |V|)
- Have a learnable projection matrix E which is d x |V|
- $h^{<t>} = E\, x^{<t>}$ dimension = d
- Output of E provides a d dimensional dense representation.
- In this matter, the learning is done end-to-end, not in two separate stages (ie, Word2Vec learning in stage 1, and Transformer learning in stage 2).

- We only introduced Word embeddings into the self-attention discussion for pedagogic purposes.

Projection Layer



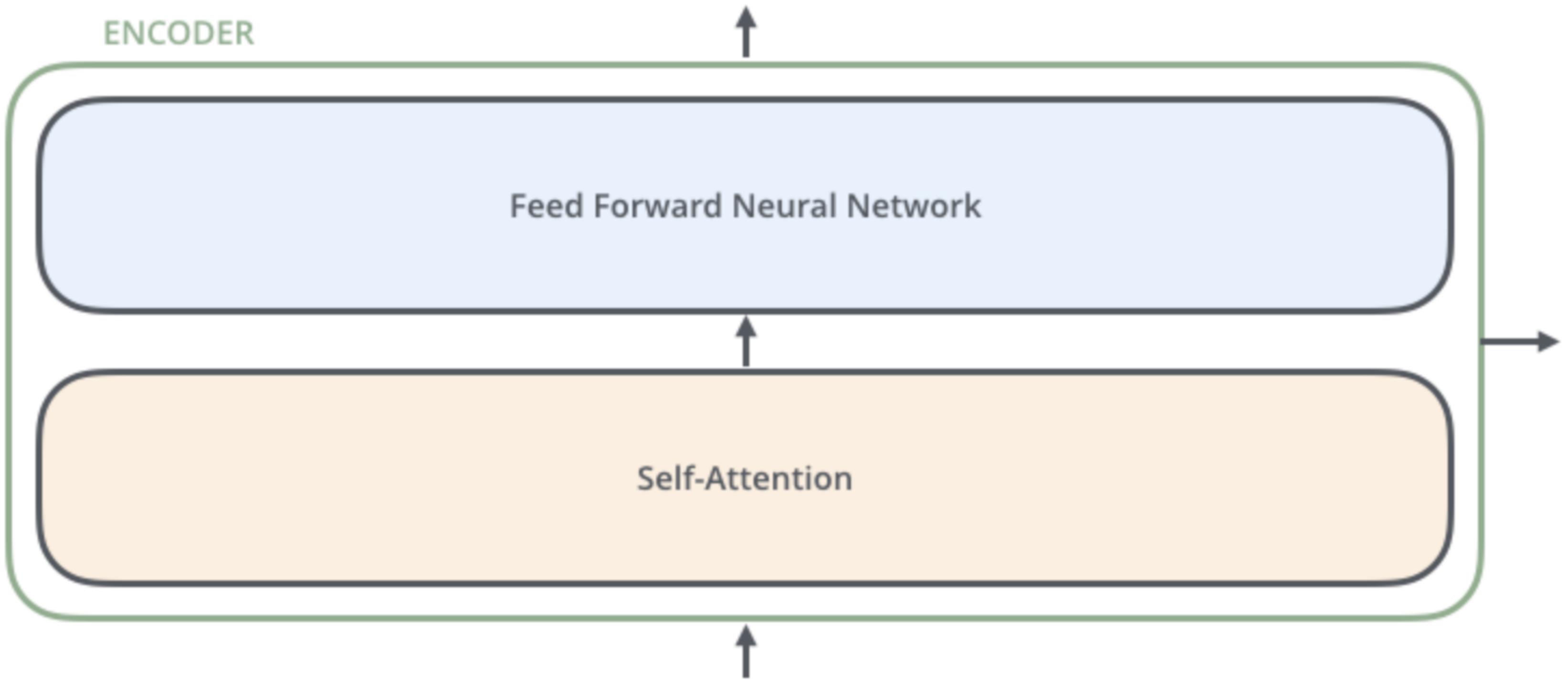
Transformer: Big Picture



Encoder layers are identical in structure but they do not share weights.

Encoder layer: big picture

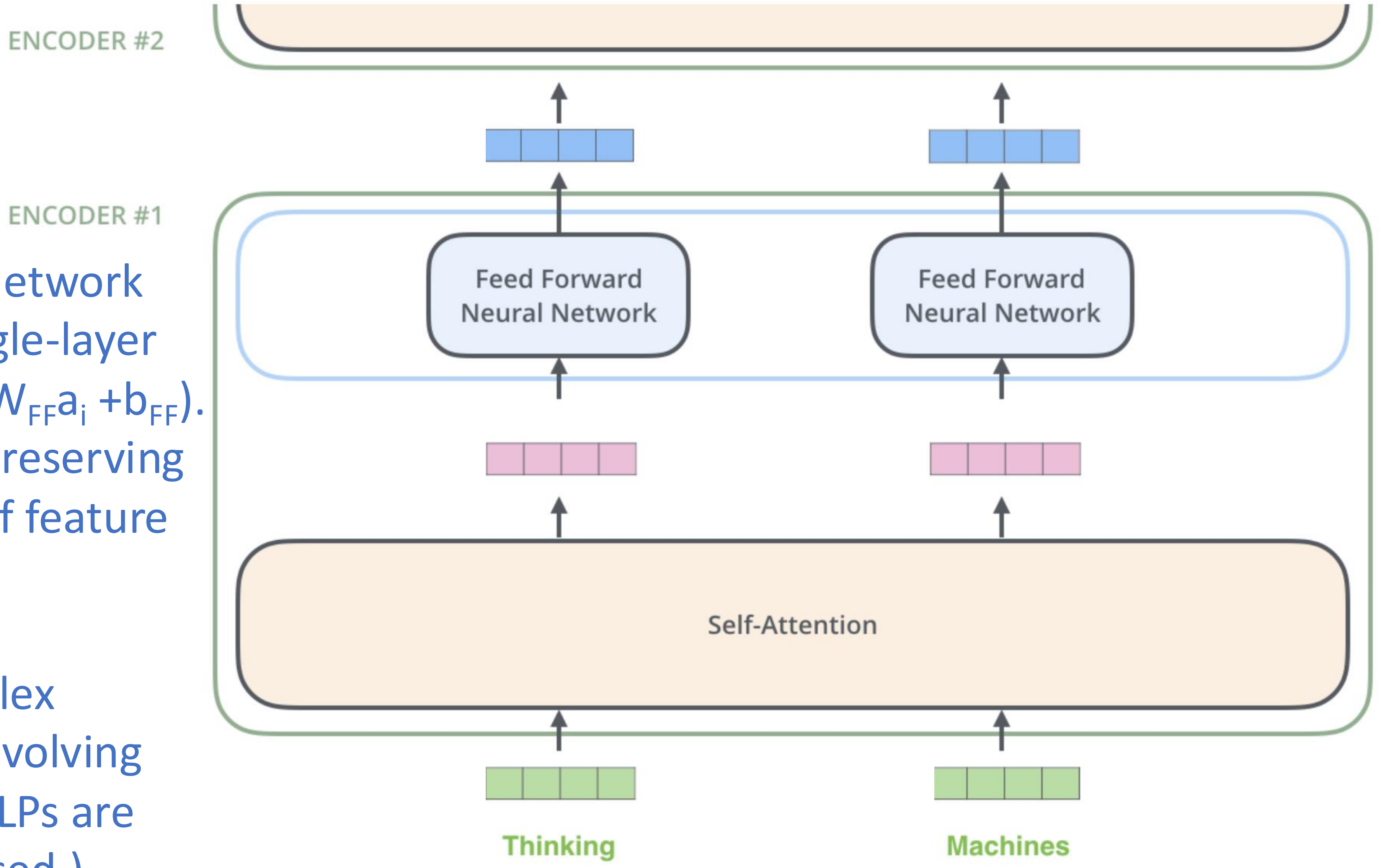
Feed-forward network == MLP



Why Feed-Forward Network?

- Additional non-linear transformation (beyond softmax)
- Additional parameters, increasing capacity of model
- Per-token transformation, complementing the between-token mechanism in self-attention.

Inside Encoder Layer



=new h_1, h_2

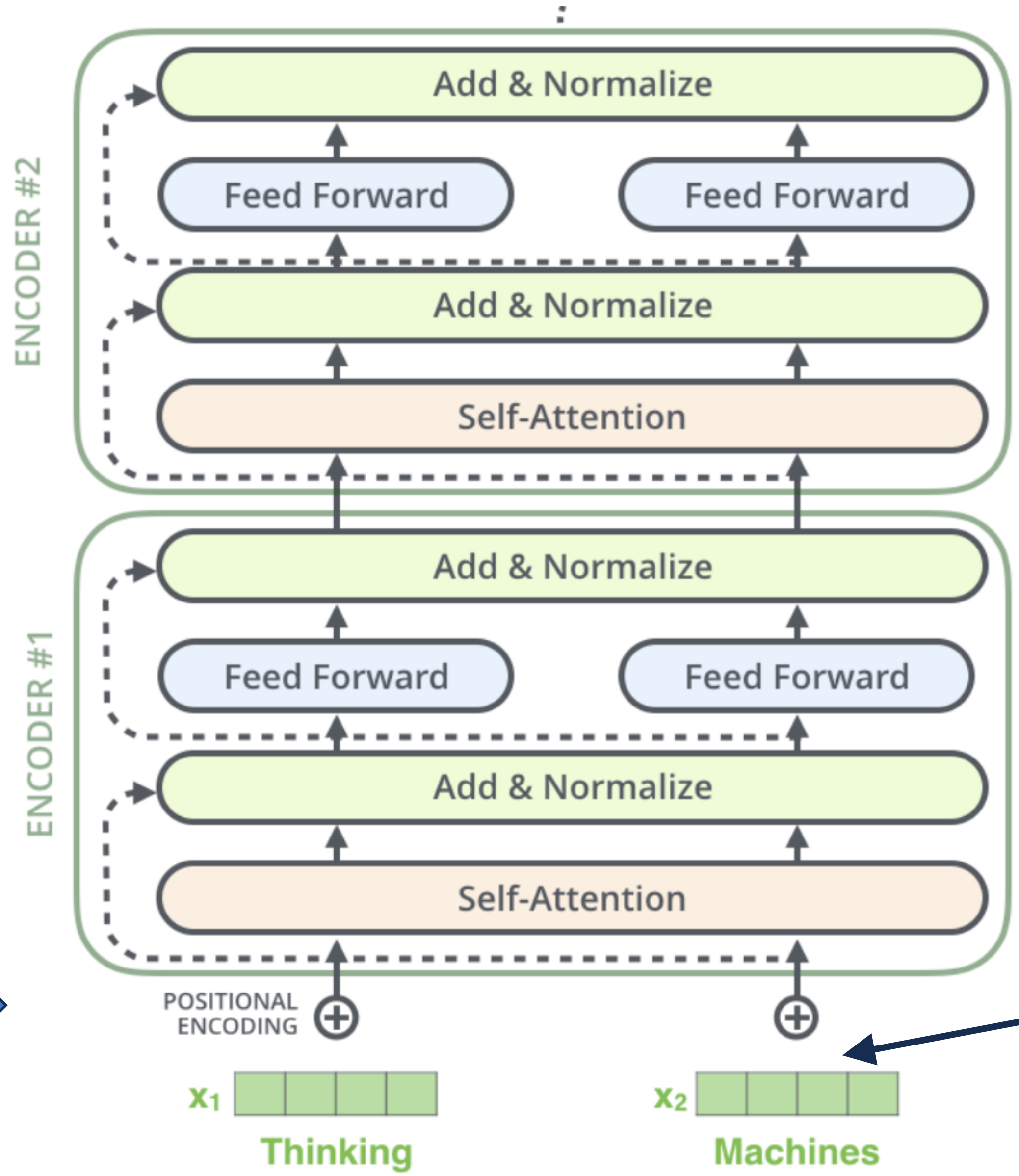
= a_1, a_2

= h_1, h_2

FF forward network can be a single-layer MLP: $h_i = g(W_{FF}a_i + b_{FF})$. W_{FF} is dxd, preserving dimension of feature vector

(More complex structures involving two-layer MLPs are also often used.)

Positional encoding Ignore this slide

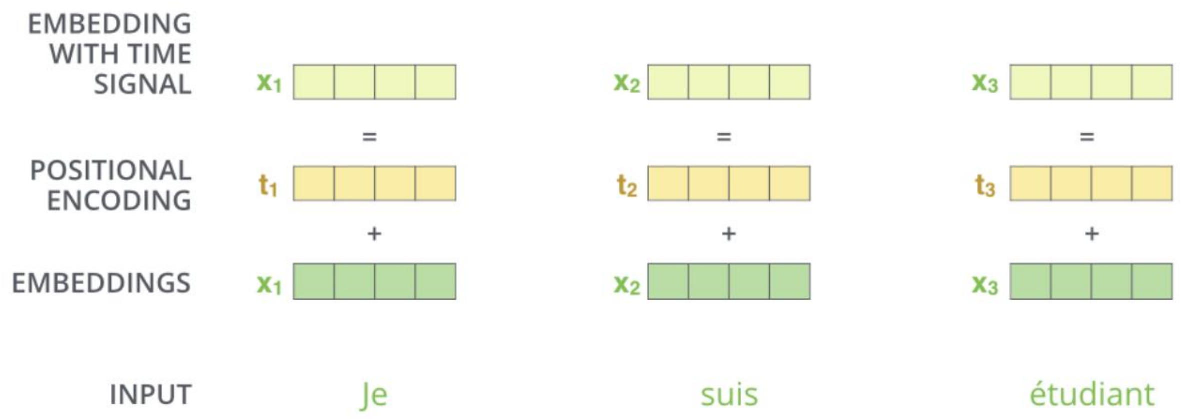


projection layer first

Positional encodings   Ignore this slide

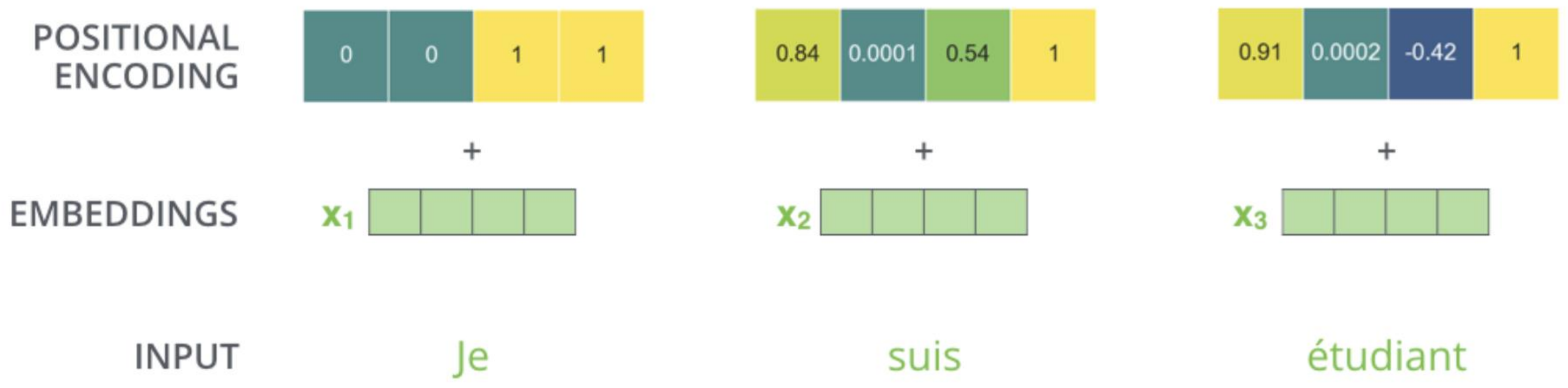
- Attention provides contextual representations
- But it alone does not capture token positional information
- Ideally representation has both semantics + position information
- So the input to first layer is modified with positional encoding

Positional encodings Ignore this slide



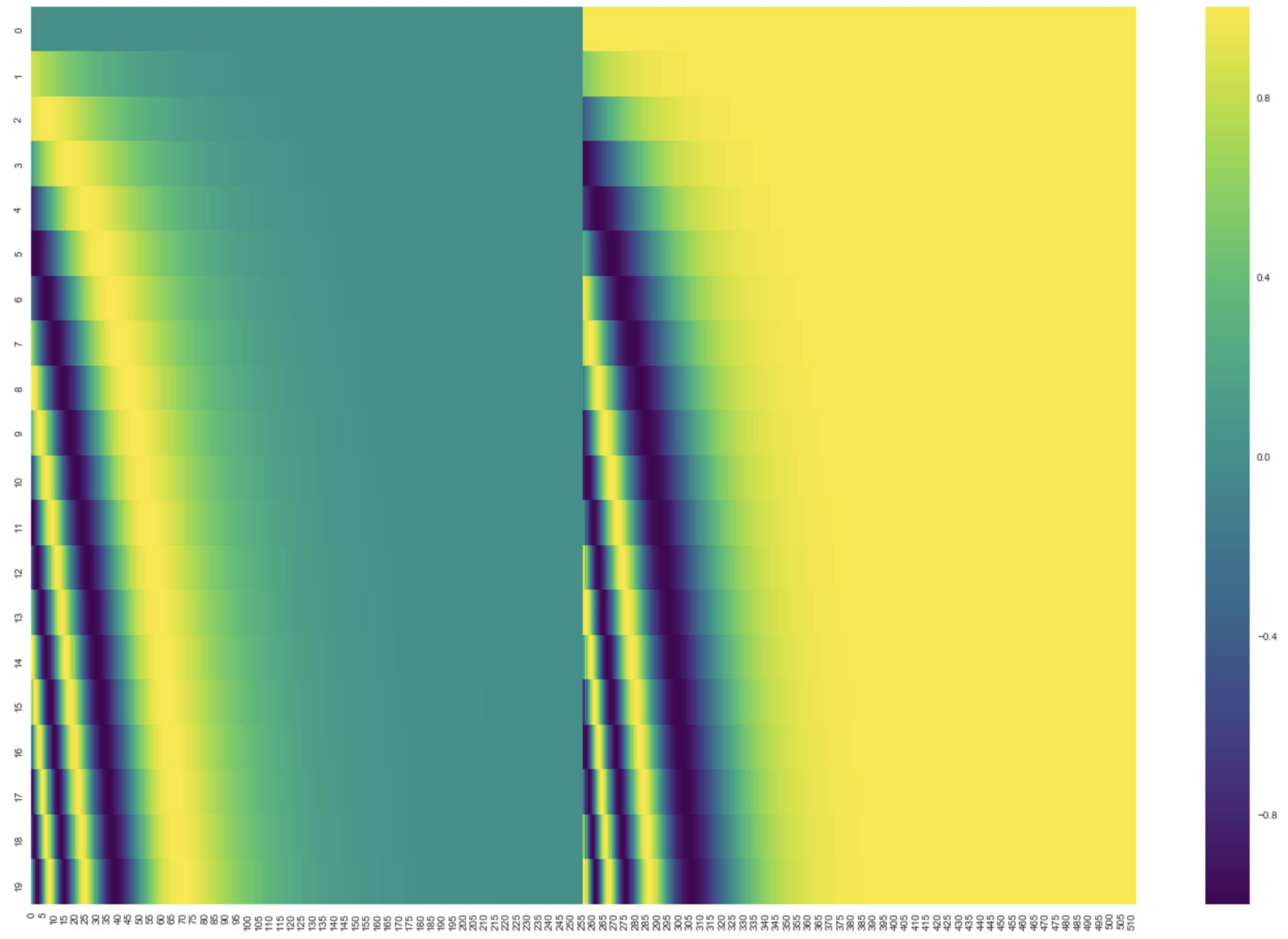
To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.

If we assumed the embedding has a dimensionality of 4, the actual positional encodings would look like this:



Positional encodings for vectors of dim 512

Encodings for 20 words

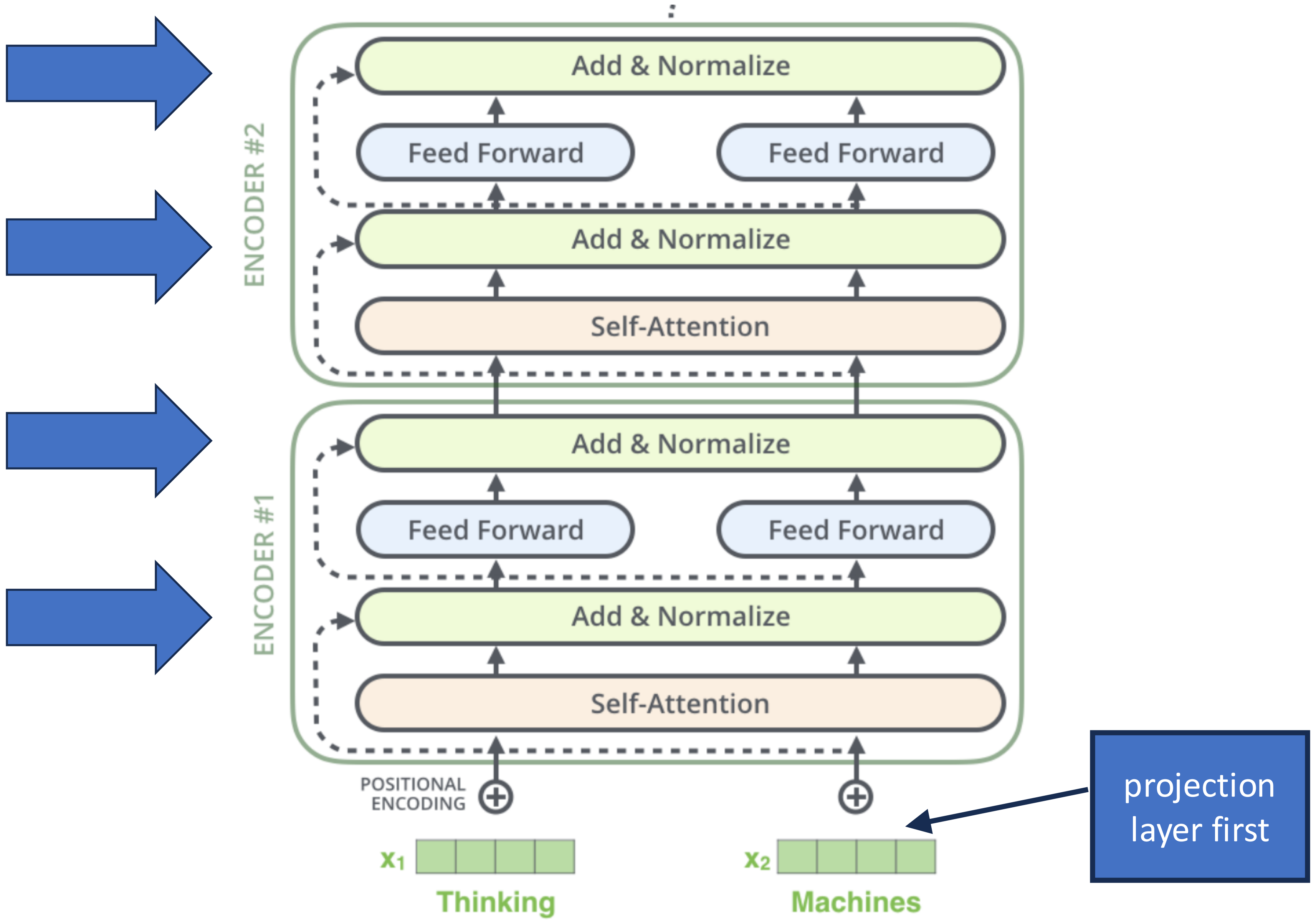


Positional encoding mechanics during inference and training: Ignore this slide

- Determine the number of input tokens *n*
- Use mathematical formula to calculate the *n* positional encoding vectors, each of dimension *d*
- Add the positional encodings to the input embeddings
- Input the resulting *n* vectors into the encoder layer 1

- Note: only add positional encodings for input into the first encoding layer.

Normalize: Layer Encoding



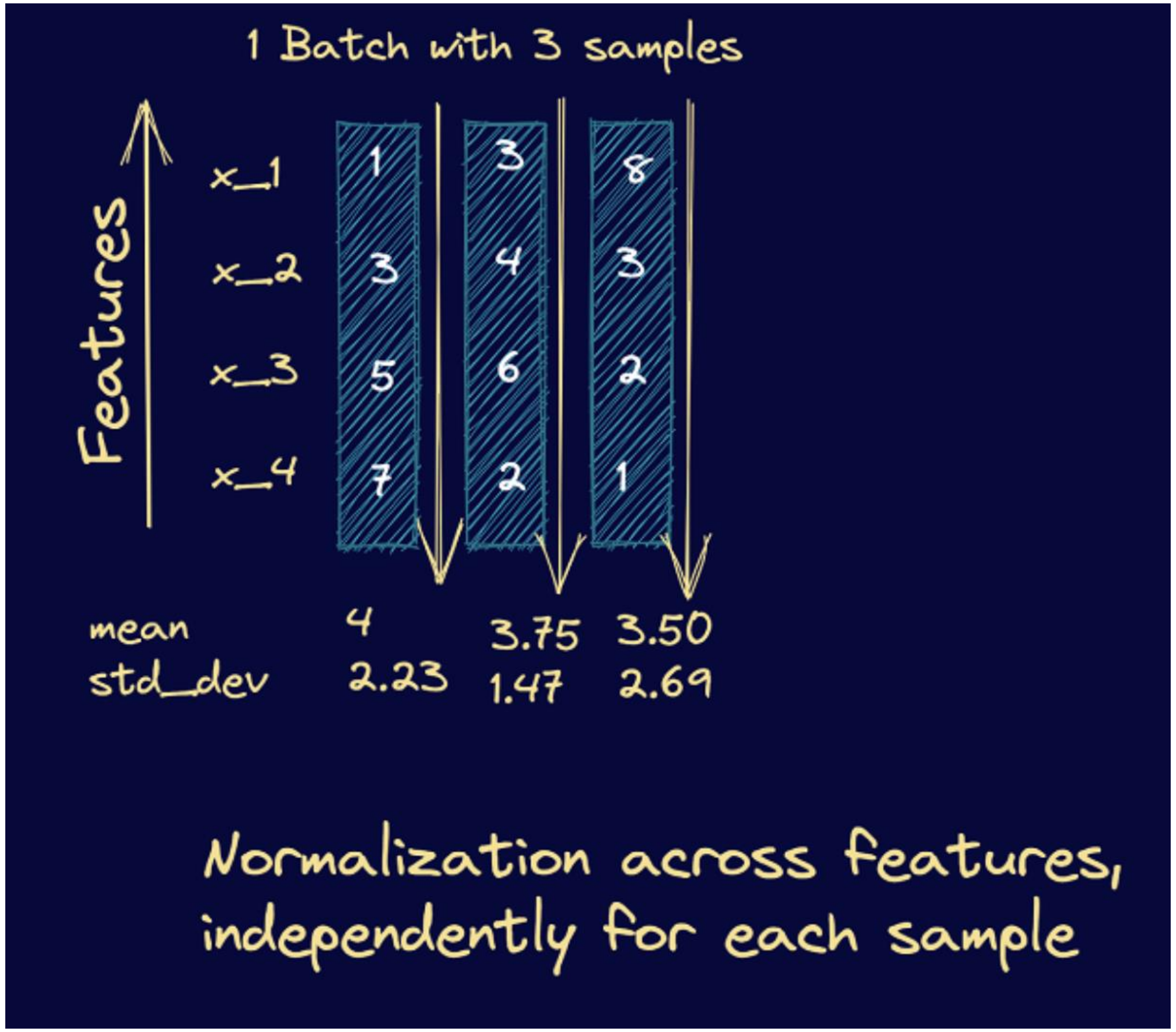
Network internal normalization

- We have already discussed normalizing inputs $\mathbf{x}^{(i)}$
- Neural networks often ***normalize within*** the neural network, that is, normalize after every layer.
- Two broad types: layer normalization and batch normalization.
- Internal normalization addresses vanishing and exploding gradient problems. Leads to more stable training and better convergence.
- Batch normalization often used in ConvNets
- Layer normalization used in Transformer

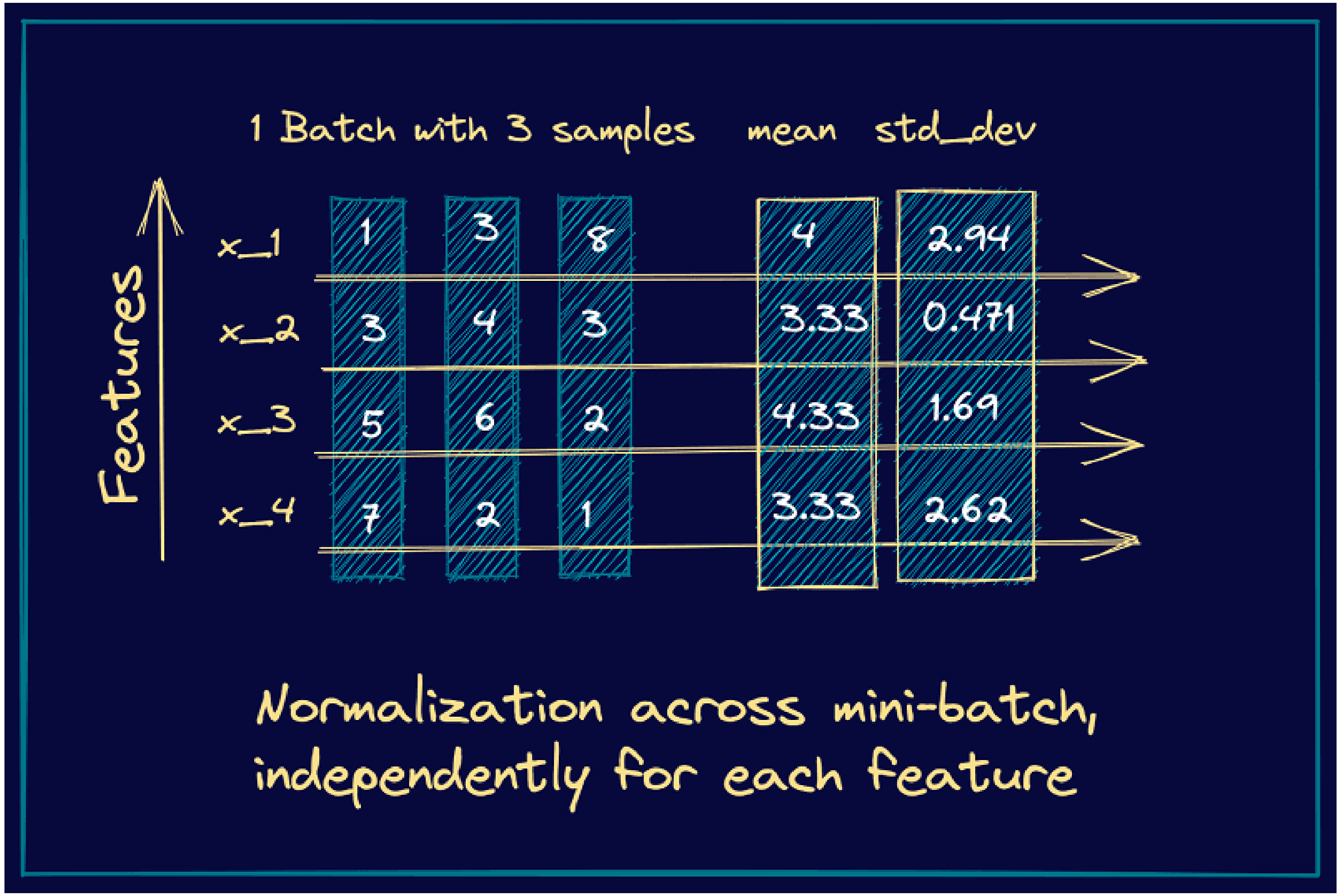
Layer and batch normalization Ignore

- Both can be done at the output of any layer
- Layer normalization: For a particular input $\mathbf{x}^{(i)}$, let $\mathbf{a} = (a_1,\ldots a_d)$ be the activations at layer ℓ. Calculate mean μ and variance σ^2 of $a_1,\ldots a_d$. Replace a_j with $(a_j - \mu)/\sigma$. Similar to normalizing an input vector but now normalizing the feature vectors at the outputs of all the layers. It is done separately for each input $\mathbf{x}^{(i)}$.
- Batch normalization: Normalize over mini-batch instead of features. For a given activation unit in layer ℓ, consider minibatch of activations $a^{(1)},a^{(2)},\ldots,a^{(m)}$. Calculate mean μ and variance σ^2 of $a^{(1)},a^{(2)},\ldots,a^{(m)}$. Replace $a^{(i)}$ with $(a^{(i)} - \mu)/\sigma$. Normalization is done across the mini-batch but separately for each feature.

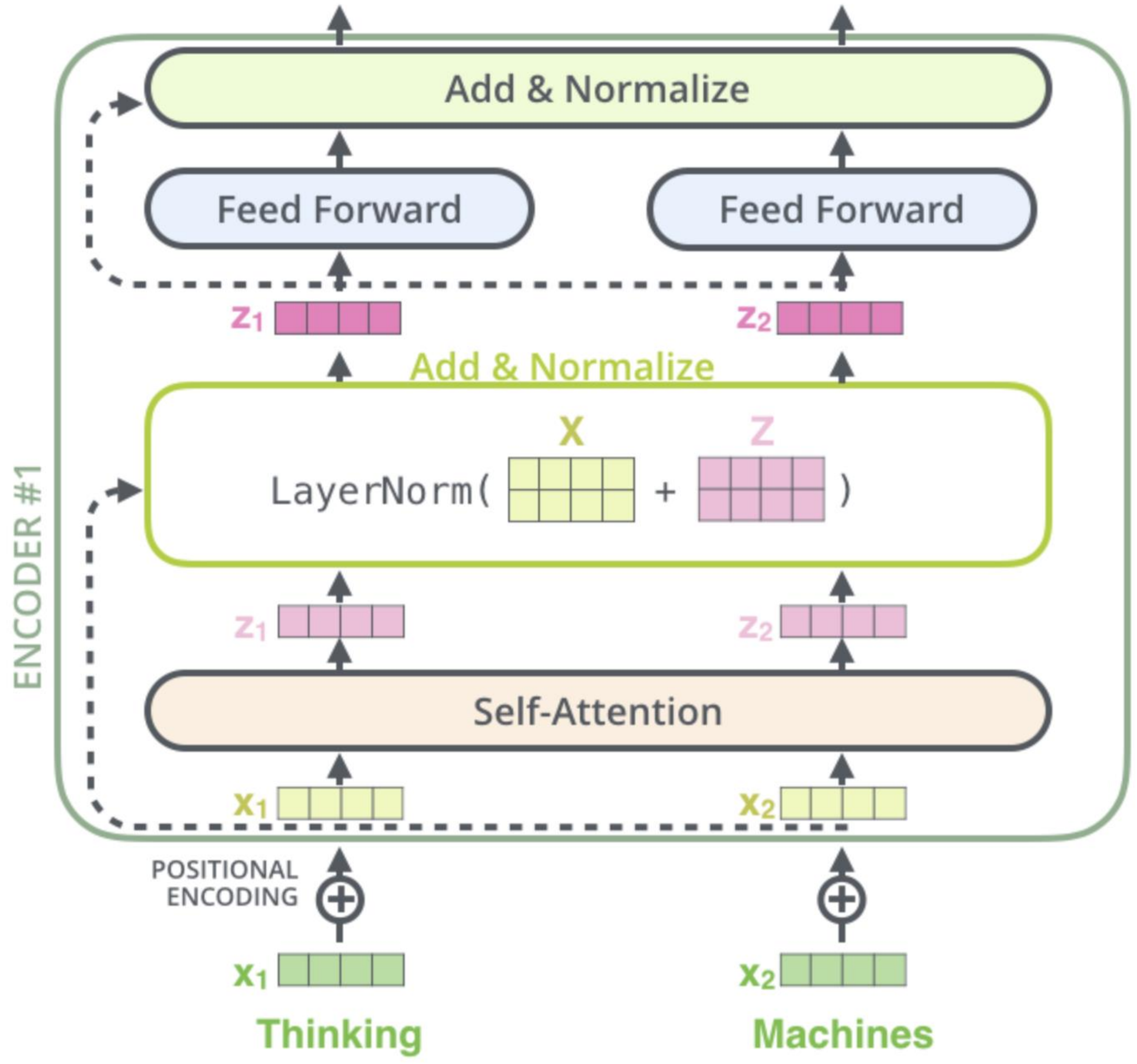
Layer Normalization Ignore



Batch Normalization Ignore



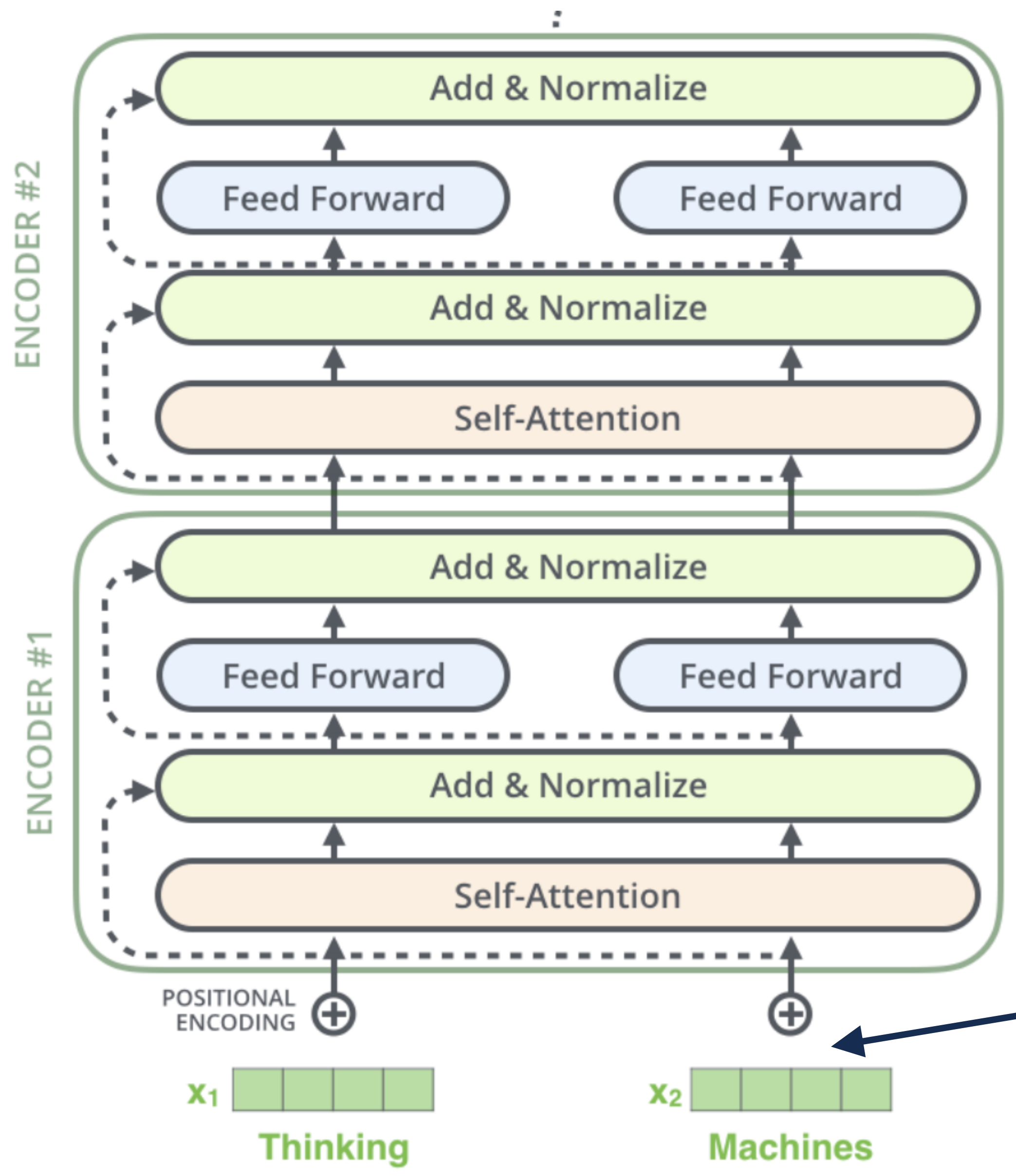
Residual and Layer Normalization Ignore



Residual: replace each z_i (aka a_i in this slide deck) with z_i+x_i. That is, at the output of each layer, replace the output with output plus the input. Same idea as Resnet: prevents vanishing gradients.

Layer norm is for each token separately: Layer norm for x_1+z_1 and for x_2+z_2 separately

Summary of Transformer



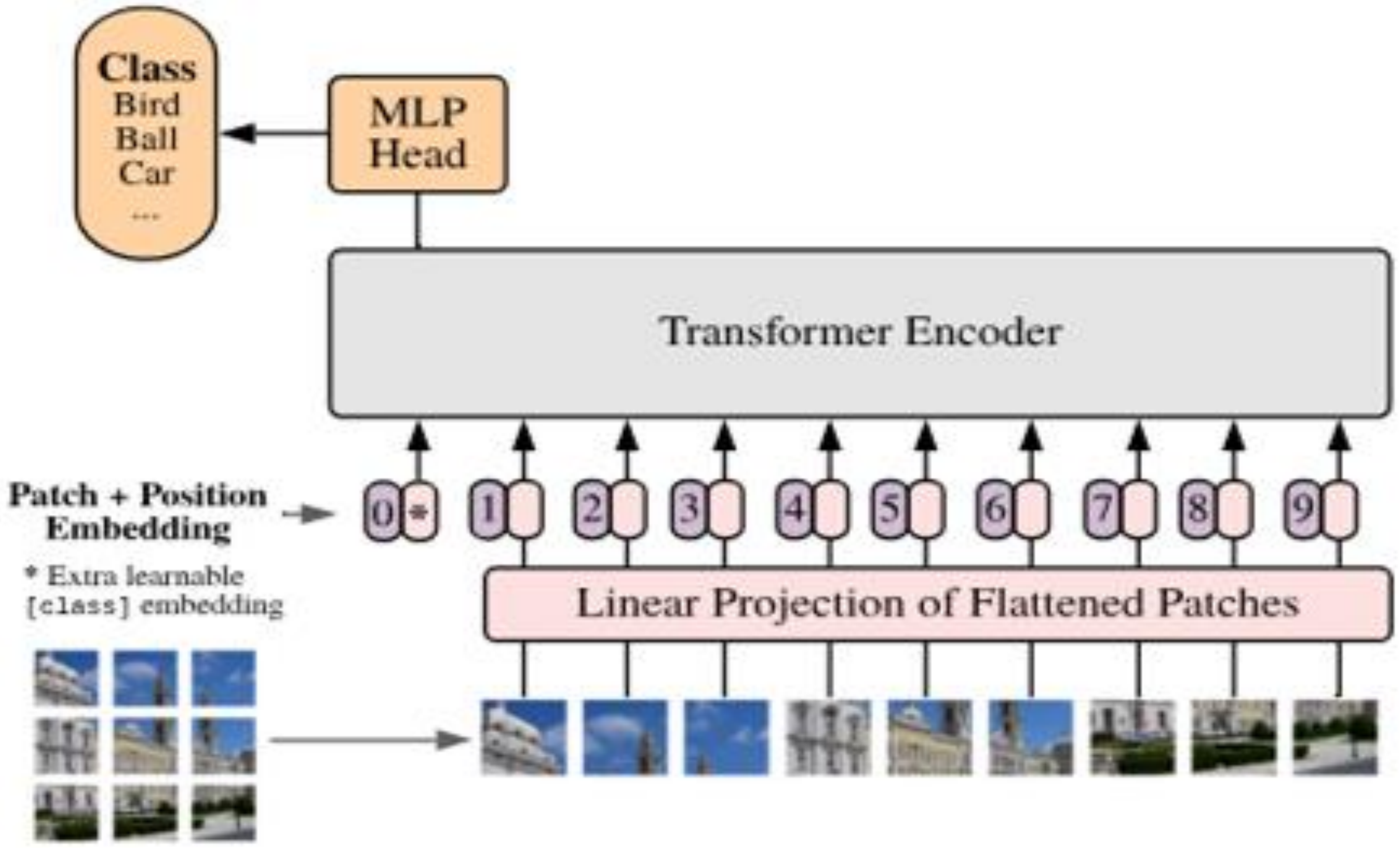
Each layer has its own weight matrices.

projection layer first

Transformer summary

- The Transformer takes in one-hot vectors $x^{<1>}, x^{<2>}, \ldots, x^{<T>}$ and produces representations $h^{<1>}, h^{<2>}, \ldots, h^{<T>}$
- The Transformer consists of many layers, each with its own parameters. Each layer takes in one $h^{<1>}, h^{<2>}, \ldots, h^{<T>}$ and outputs a new $h^{<1>}, h^{<2>}, \ldots, h^{<T>}$
- Each layer has attention sublayer, FF sublayer, residual connections, and layer normalization.
- Before first layer, need to add positional encoding.
- In LLMs, use representation $h^{<t>}$ to generate (inference) or predict (training) the next word $x^{<t+1>}$.

Vision Transformer (ViT)


Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 *
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches

Pros and Cons of ViT

- Global context understanding: Unlike Convnets, which rely on local receptive filters, ViTs use self-attention to capture global dependencies across the image.
- Scalability: Transformers scale efficiently with data and computational resources, often outperforming CNNs when trained on large datasets.
- Flexibility: ViTs can be adapted to various tasks beyond classification, such as object detection and segmentation.
- Data Requirements: ViTs require large datasets for effective training. They often underperform CNNs on smaller datasets.
- Computational Cost: The quadratic complexity of self-attention can make ViTs computationally expensive for high-resolution images.