UVA CS 4774: Machine Learning

S3: Lecture 18: Transformer Basic Architectures

Dr. Yanjun Qi

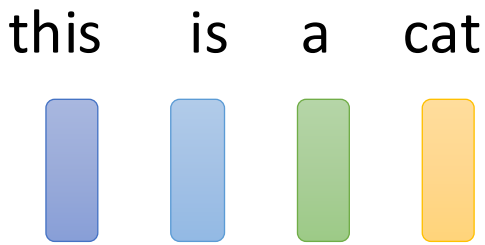
University of Virginia
Department of Computer Science

Module I

Roadmap : Today

- **Recap (word vectors, attention, bag-of-words)**
- Self-attention Basics (Q, K, V, scaling, attention map)
- Transformer Block Basics (self-attn + heads, positions, residual + norm)
- Masks & Model Families (encoder-only, encoder-decoder, decoder-only)
- Pretraining a GPT (tokens, next-token loss, Adam, GPT-1/2/3)
- Generating Text (greedy, sampling, beam search, KV cache)
- Post-training Preview (aligning a GPT: RLHF, DPO)

Recap: Word2Vec



One-hot Encoding

apple = [1 0 0 0 0 ……]

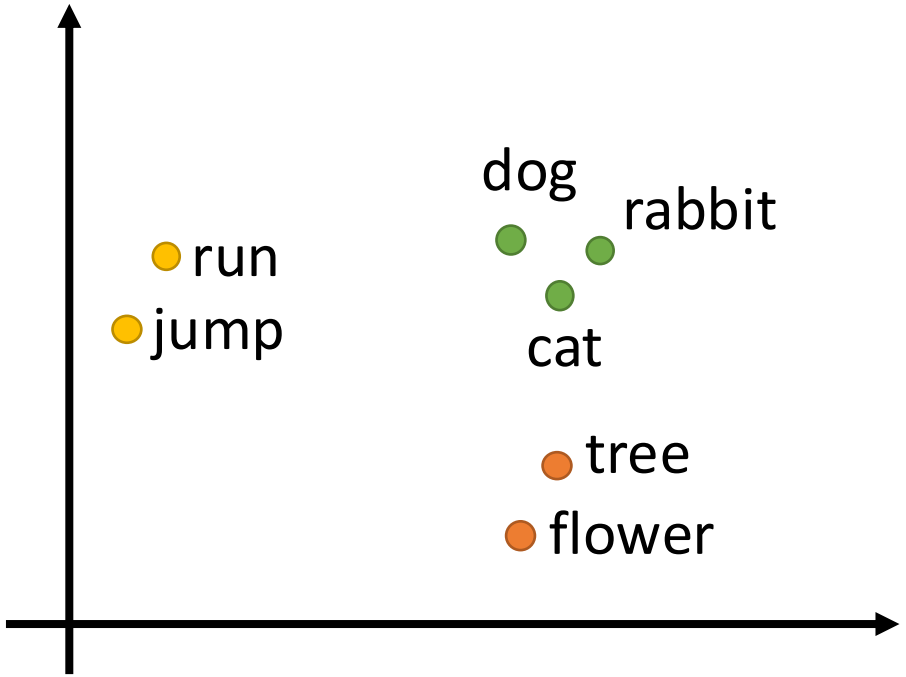
bag = [0 1 0 0 0 ……]

cat = [0 0 1 0 0 ……]

dog = [0 0 0 1 0 ……]

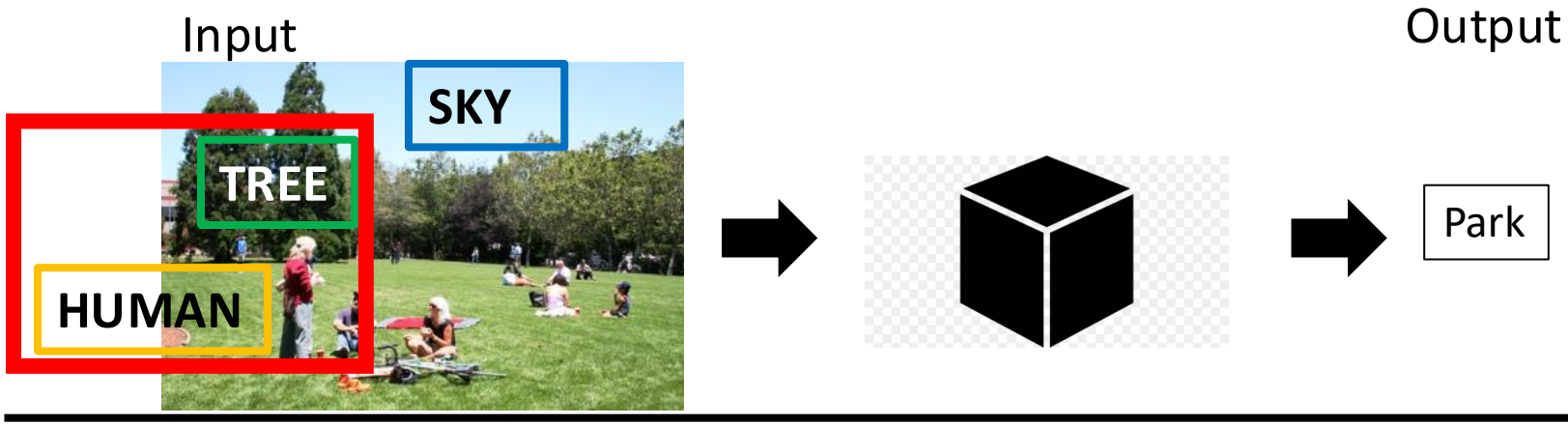
elephant = [0 0 0 0 1 ……]

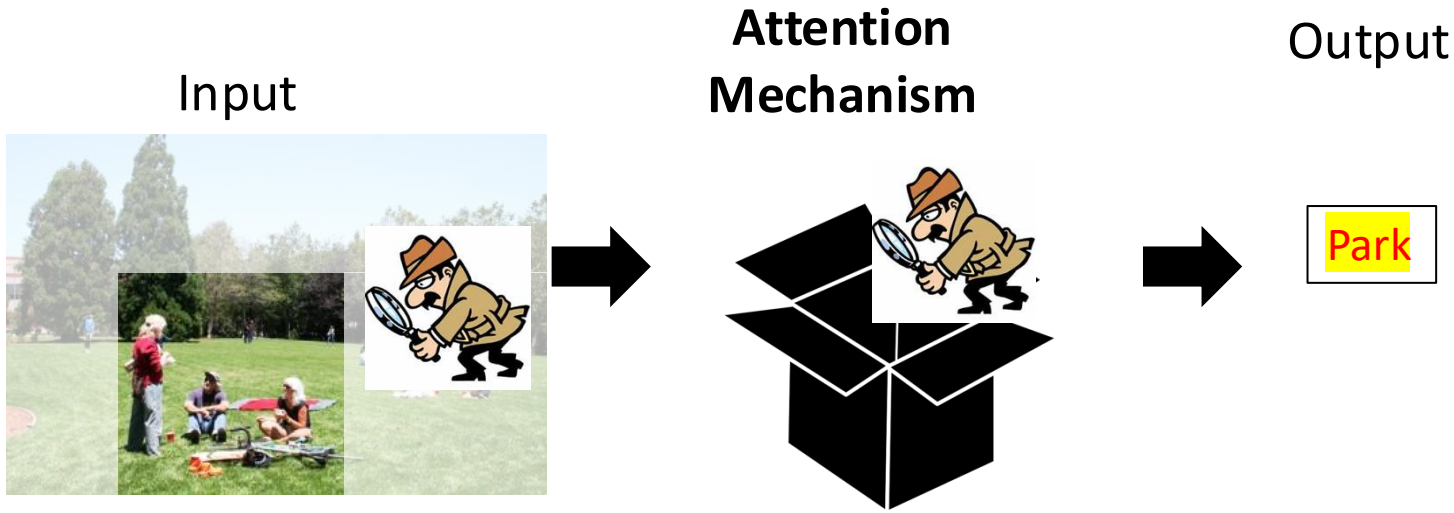
Word Embedding



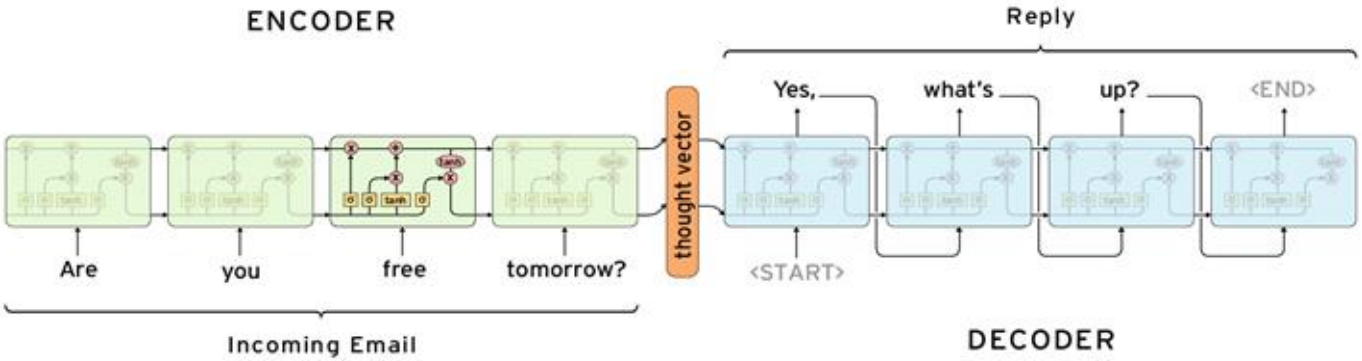
credit: Prof. Hung-yi Lee,

Recap: Attention Trick



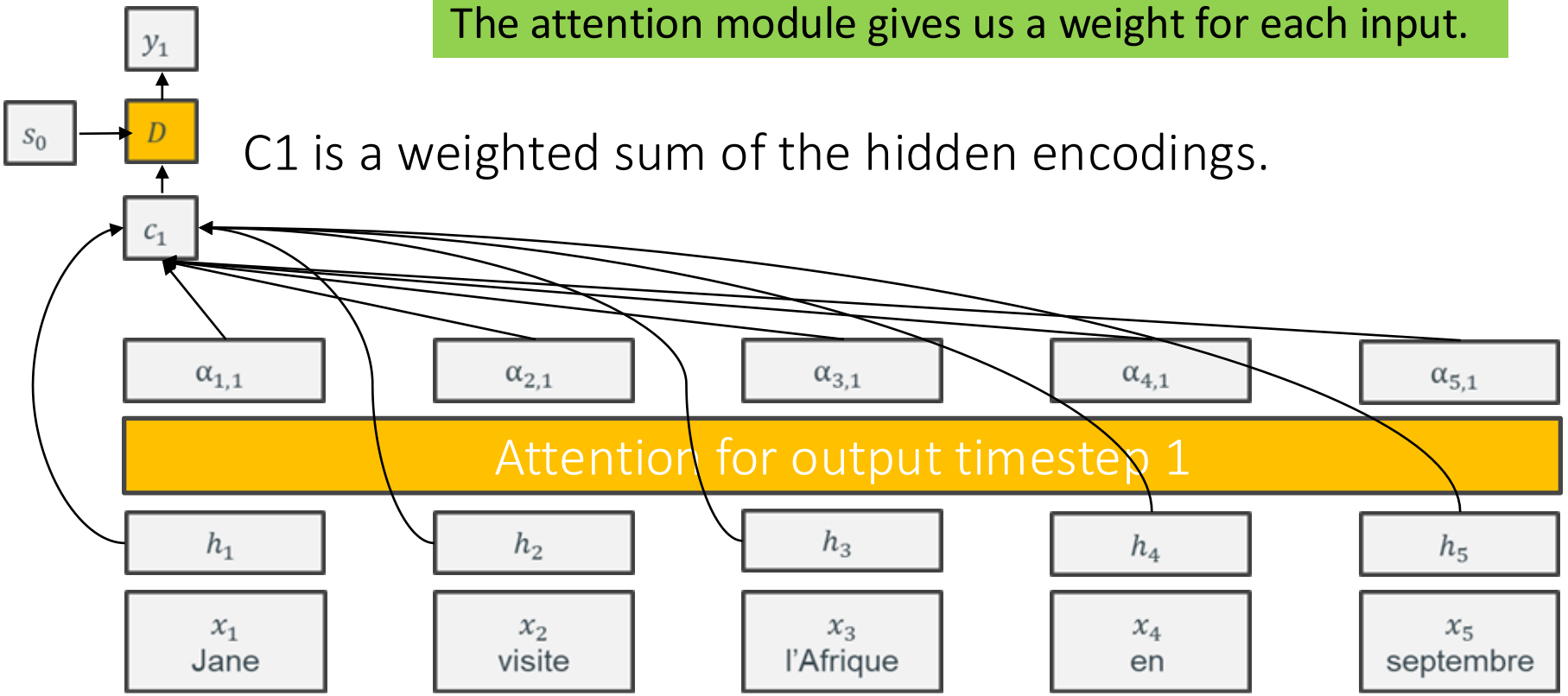


Recap:Seq2Seq



The attention module gives us a weight for each input.

C1 is a weighted sum of the hidden encodings.



Based: Dr. Yangqiu Song's slides

Recap: BOW + logistic regression

$$P(\text{label} \mid x) = \text{sigmoid}(\text{score(review)}) = \text{Sigmoid}\left(\sum w_{\text{word}} \cdot \text{count}_{\text{word}} + b\right)$$

Each word adds its own fixed amount, whatever the other words are.

Review	the	movie	was	not	good	score	label
"the movie was good"	0	0	0	–	+2.0	+2.0	positive ✓
"the movie was not good"	0	0	0	−0.5	+2.0	+1.5	positive ✗

"not" can only add its own weight in BoW. It cannot flip the meaning of "good".

Bigram features ("not good") helps, but the vocabulary explodes.

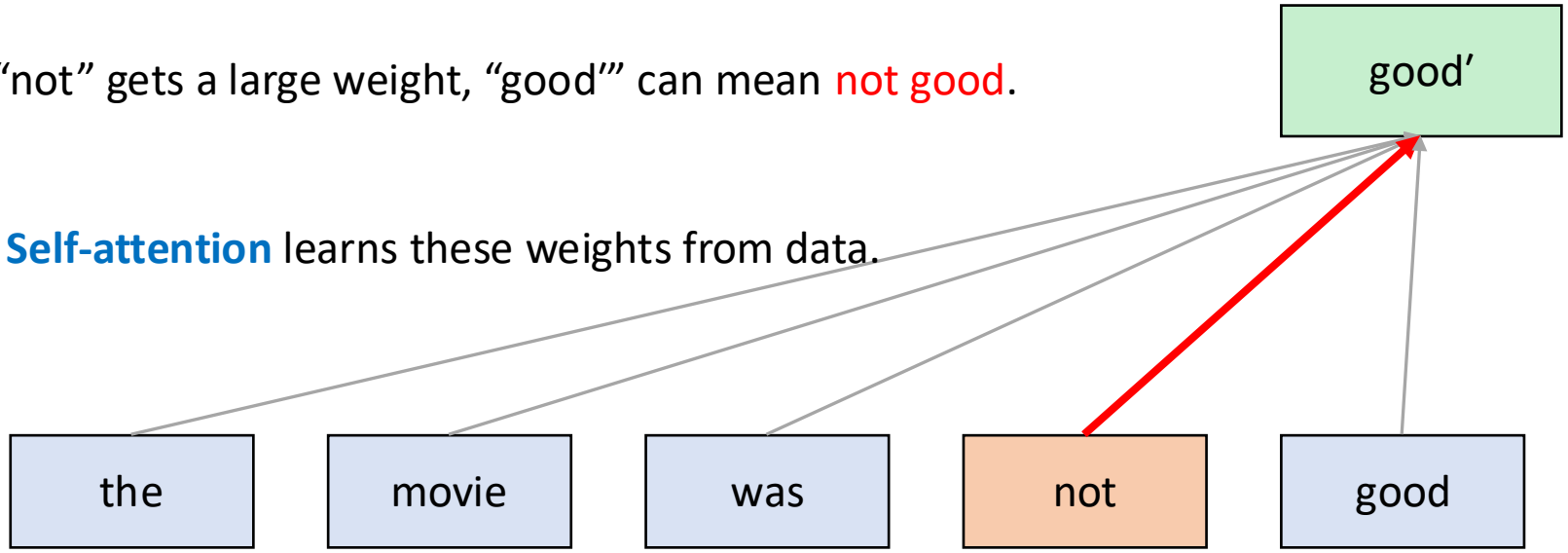
credit: Prof. Hung-yi Lee,

What we want: context-dependent word vectors

The new vector for describing **good** should be built from all the words in the sentence,
with a learned weight (attention weight) on each one.

If “not” gets a large weight, “good’” can mean not good.

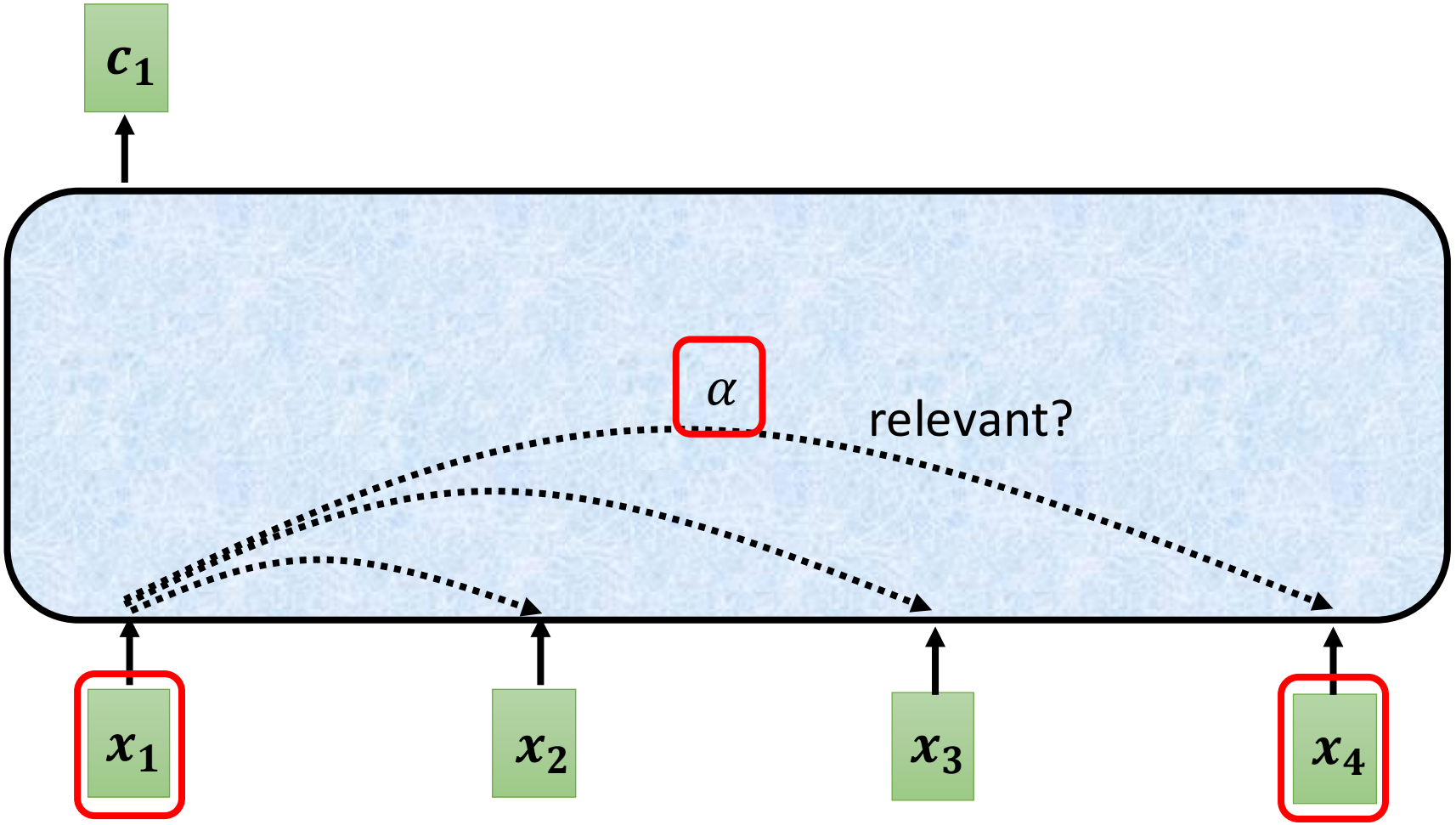
→ **Self-attention** learns these weights from data.



Roadmap : Today

- Recap (word vectors, attention, bag-of-words)
- **Self-attention Basics (Q, K, V, scaling, attention map)**
- Transformer Block Basics (self-attn + heads, positions, residual + norm)
- Masks & Model Families (encoder-only, encoder-decoder, decoder-only)
- Pretraining a GPT (tokens, next-token loss, Adam, GPT-1/2/3)
- Generating Text (greedy, sampling, beam search, KV cache)
- Post-training Preview (aligning a GPT: RLHF, DPO)

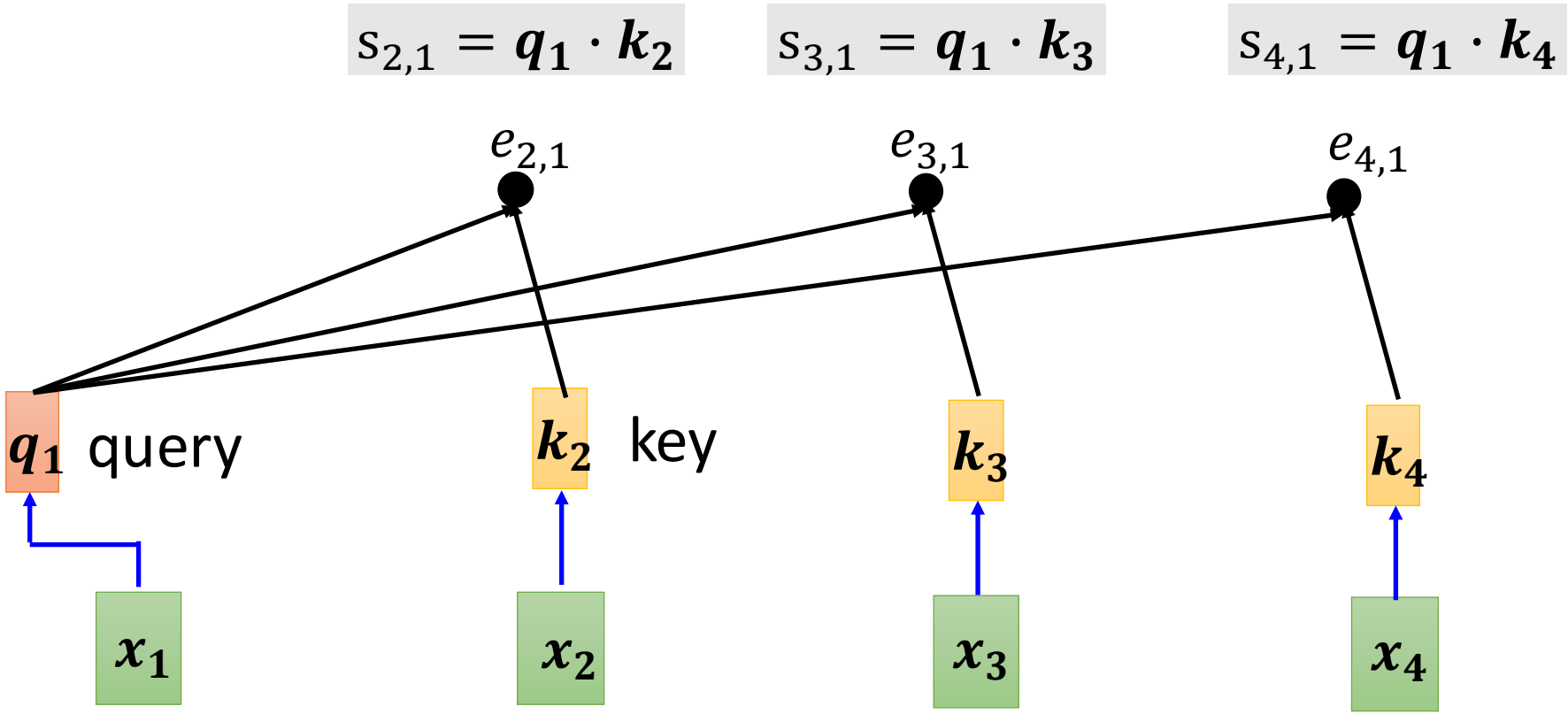
Self-attention



Find the relevant vectors in a sequence

credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

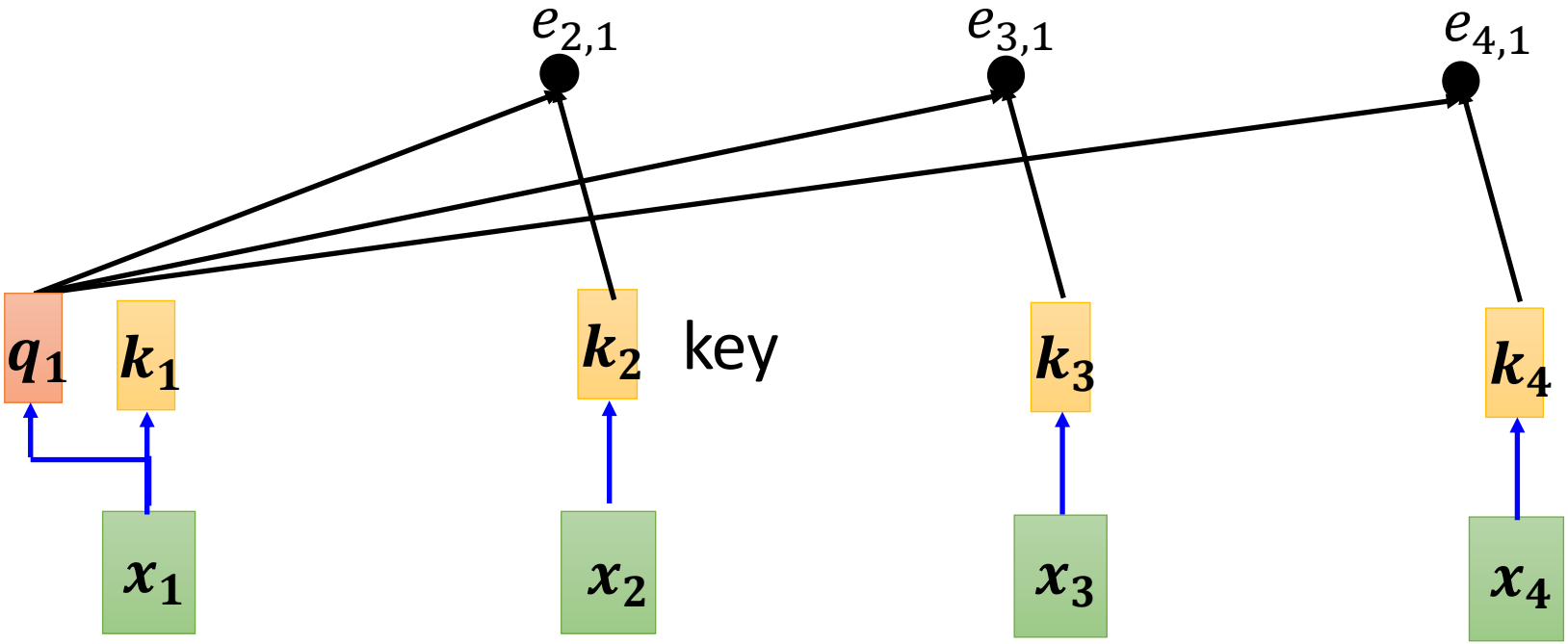
Self-attention



credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

Self-attention

$s_{1,1} = \boldsymbol{q_1} \cdot \boldsymbol{k_1}$ $s_{2,1} = \boldsymbol{q_1} \cdot \boldsymbol{k_2}$ $s_{3,1} = \boldsymbol{q_1} \cdot \boldsymbol{k_3}$ $s_{4,1} = \boldsymbol{q_1} \cdot \boldsymbol{k_4}$



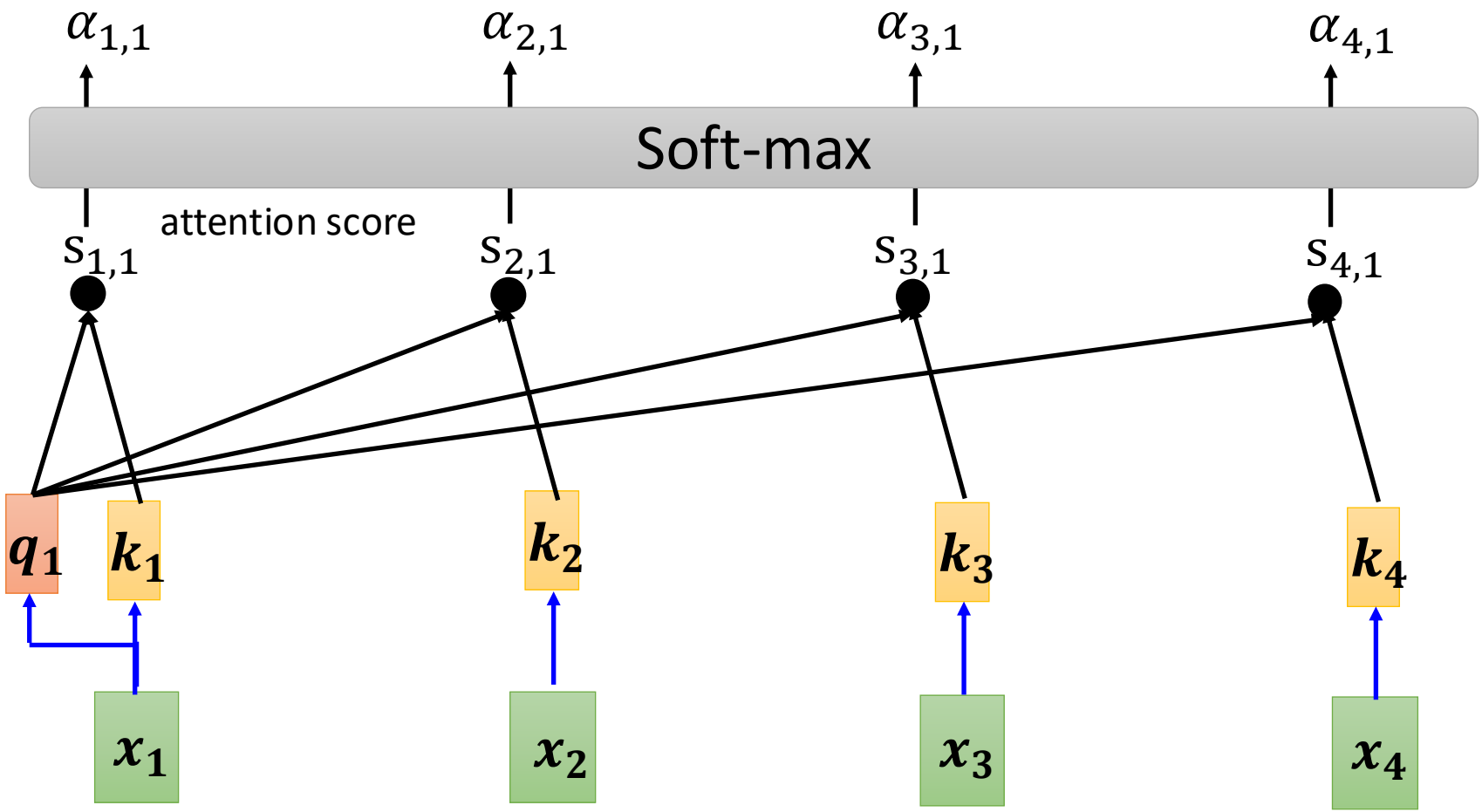
$\boldsymbol{q_1} = \boldsymbol{x_1} W^q$ $\boldsymbol{k_2} = \boldsymbol{x_2} W^k$ $\boldsymbol{k_3} = \boldsymbol{x_3} W^k$ $\boldsymbol{k_4} = \boldsymbol{x_4} W^k$

$\boldsymbol{k_1} = \boldsymbol{x_1} W^k$

credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

Self-attention

$$\alpha_{i,1} = exp(s_{i,1}) / \sum_j exp(s_{j,1})$$



$q_1 = x_1 W^q$ $\quad k_2 = x_2 W^k$ $\quad k_3 = x_3 W^k$ $\quad k_4 = x_4 W^k$

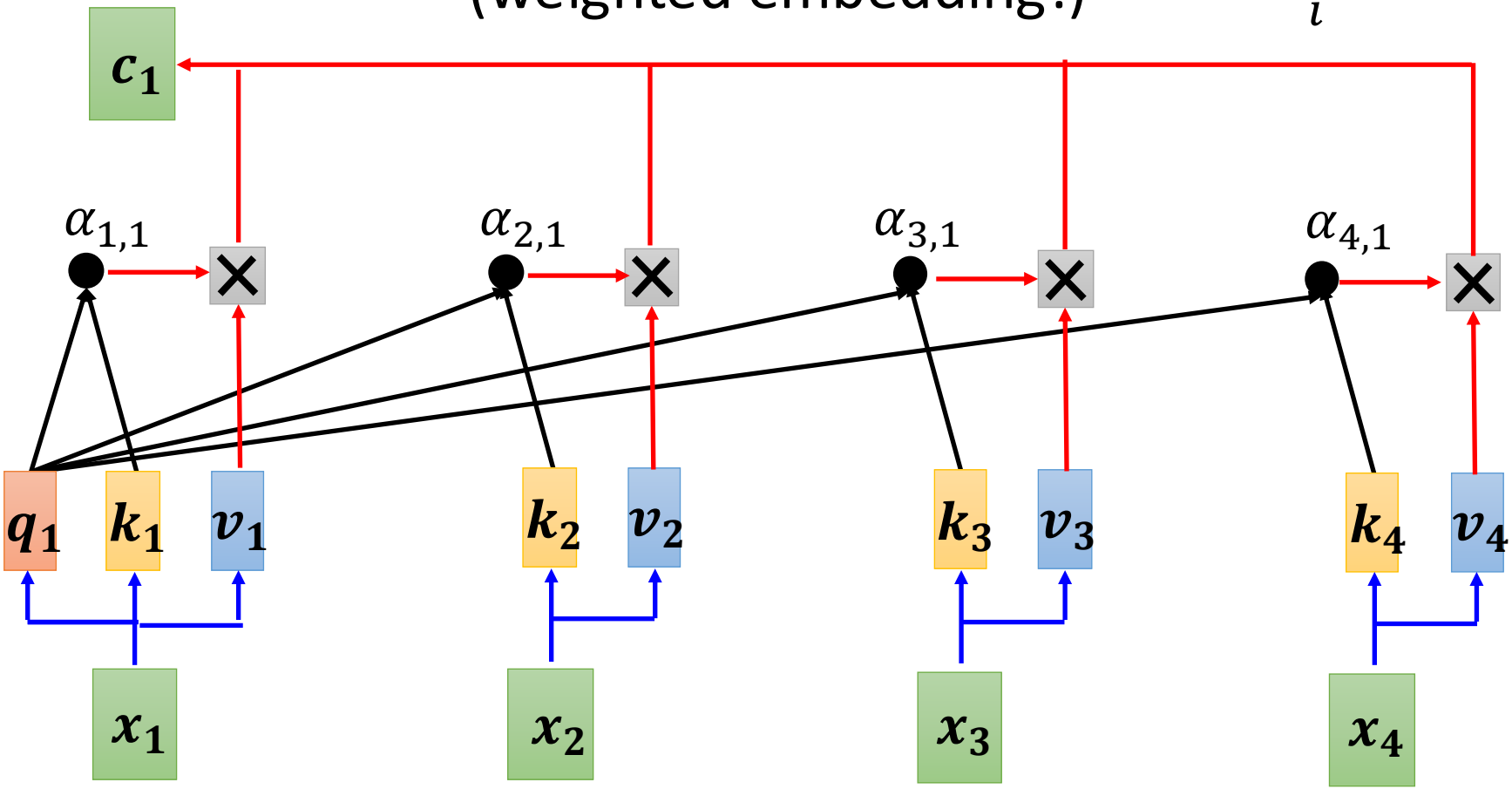
$k_1 = x_1 W^k$

credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

Self-attention:

Extract information based on attention scores (weighted embedding!)

$$\boldsymbol{c_1} = \sum_i \alpha_{i,1} \boldsymbol{v_i}$$



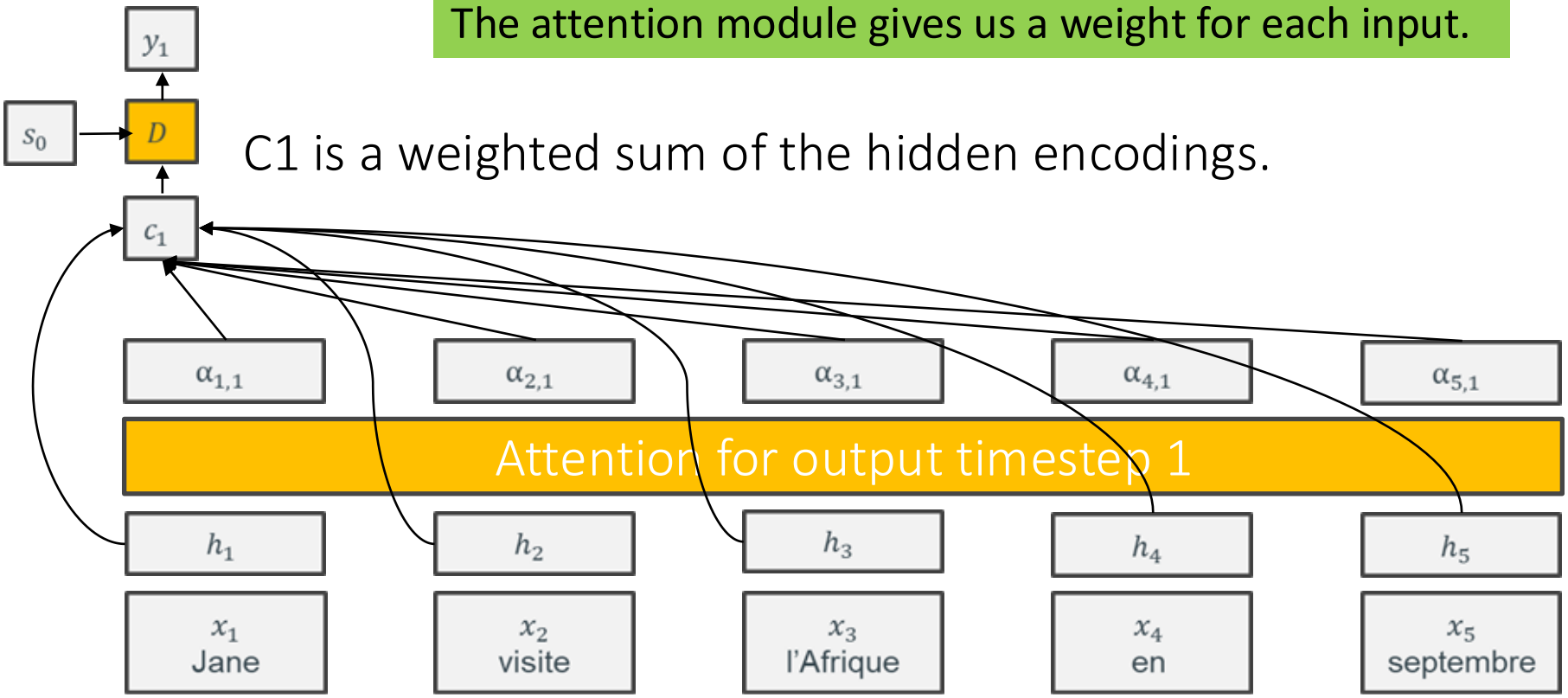
$$\boldsymbol{v_1} = \boldsymbol{x_1} W^v \qquad \boldsymbol{v_2} = \boldsymbol{x_2} W^v \qquad \boldsymbol{v_3} = \boldsymbol{x_3} W^v \qquad \boldsymbol{v_4} = \boldsymbol{x_4} W^v$$

credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

RECAP: seq2seq attention

The attention module gives us a weight for each input.

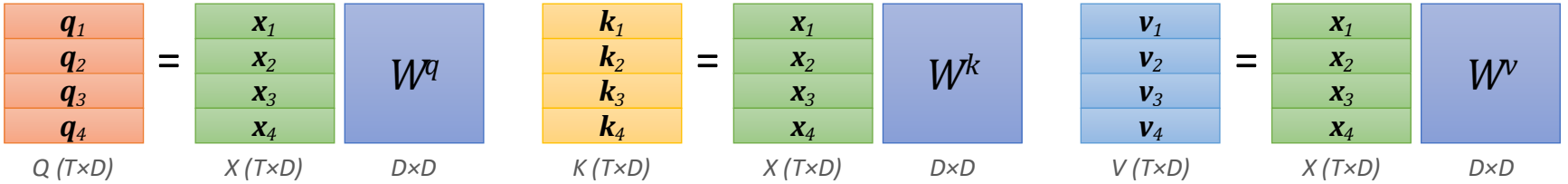
C1 is a weighted sum of the hidden encodings.



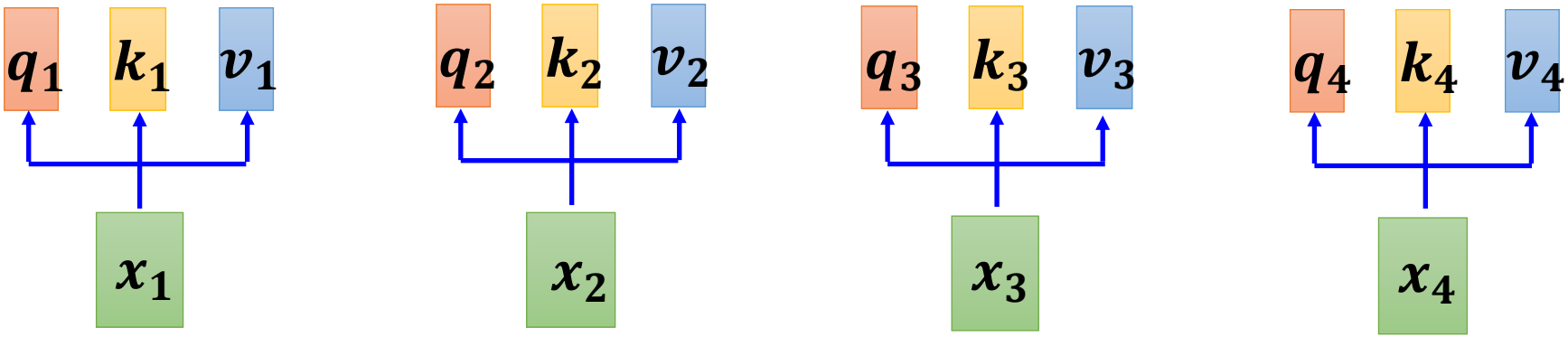
Based: Dr. Yangqiu Song's slides

Self-attention

$q_i = x_i W^q$ $\quad k_i = x_i W^k$ $\quad v_i = x_i W^v$

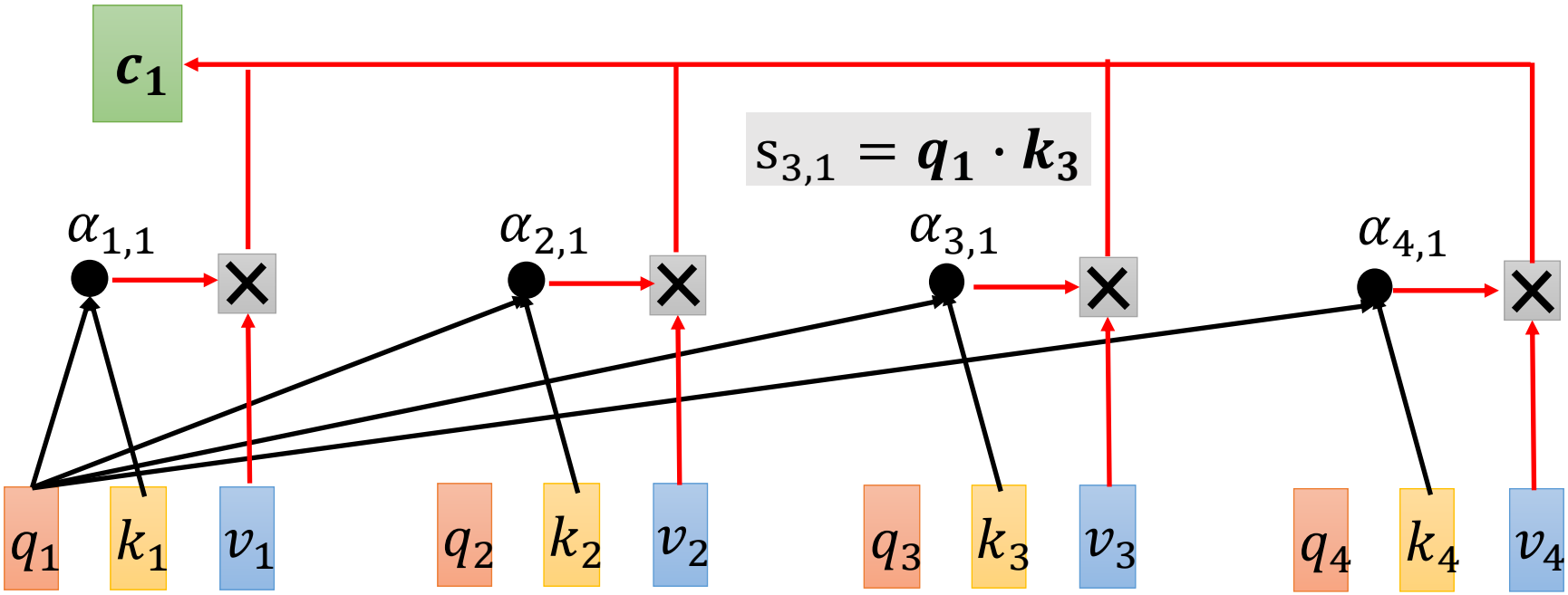


One token per row: row t of X is $\mathbf{x}_t$, row t of Q is $\mathbf{q}_t$.



credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

Self-attention



c_1		$\alpha_{1,1}$	$\alpha_{2,1}$	$\alpha_{3,1}$	$\alpha_{4,1}$		v_1
c_2	=	$\alpha_{1,2}$	$\alpha_{2,2}$	$\alpha_{3,2}$	$\alpha_{4,2}$		v_2
c_3		$\alpha_{1,3}$	$\alpha_{2,3}$	$\alpha_{3,3}$	$\alpha_{4,3}$		v_3
c_4		$\alpha_{1,4}$	$\alpha_{2,4}$	$\alpha_{3,4}$	$\alpha_{4,4}$		v_4
C (T×D)		*A (T×T)*					*V (T×D)*

Row t of A: how much output t takes from each position i

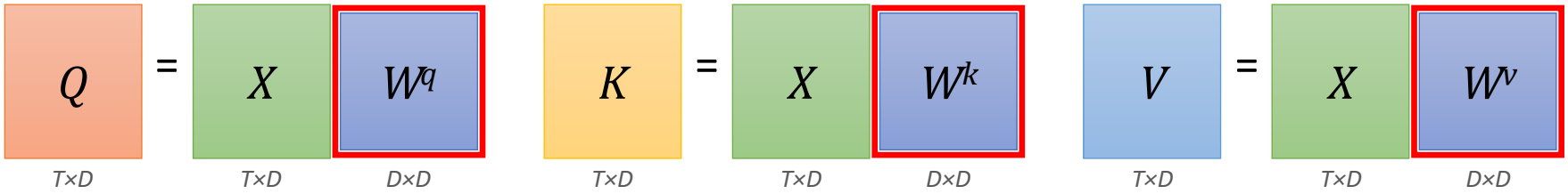
Each row of A sums to 1

Red row: $\mathbf{c}_1 = \Sigma_i \alpha_{i,1} \mathbf{v}_i$, as in the figure above

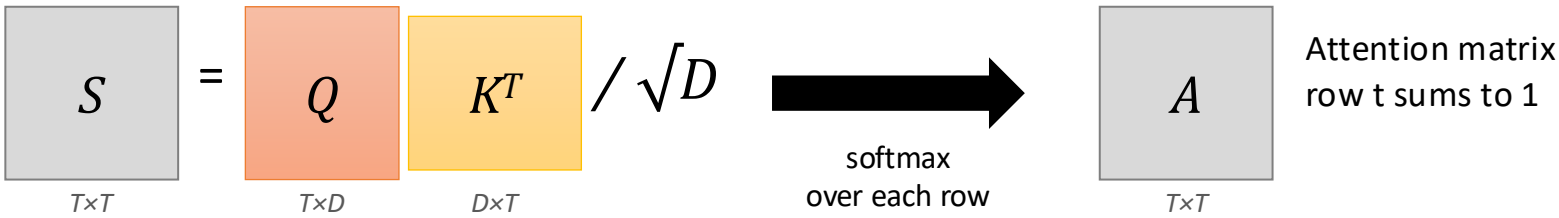
credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

Self-attention

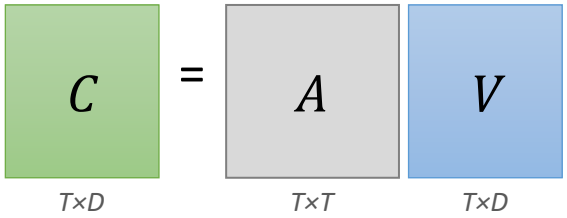
One token per row (as in PyTorch)



Red: the only parameters to be learned



Attention matrix
row t sums to 1



Row t of C: $\mathbf{c}_t = \Sigma_i \alpha_{i,t} \mathbf{v}_i$

$$\mathbf{C = softmax(Q K^T / \sqrt{D}) V}$$

Code: the same steps in PyTorch

	Math (one token per row, as in the figures)	PyTorch
Input	X: row t is $\mathbf{x}_t$ (T × D)	`X = tok_embed(idx) + pos_embed(pos)`
Query / Key / Value	$\mathbf{q}_t = \mathbf{x}_t W^q$, $Q = X W^q$ (same for K, V)	`Q = X @ W_q` (or `nn.Linear`)
Scores	$S = Q K^T / \sqrt{D}$ (T × T); $S[t, i] = s_{i,t} / \sqrt{D}$	`S = Q @ K.transpose(-2, -1) / math.sqrt(D)`
Normalize	softmax over each row; $A[t, i] = \alpha_{i,t}$, row t sums to 1	`A = torch.softmax(S, dim=-1)`
Output	$C = A V$; row t of C is $\mathbf{c}_t = \Sigma_i \alpha_{i,t} \mathbf{v}_i$	`C = A @ V` (T × D)

Same $\alpha_{i,t}$ as in seq2seq: $\mathbf{c}_t = \Sigma_i \alpha_{i,t} \mathbf{h}_i$. Self-attention swaps $\mathbf{h}_i$ for $\mathbf{v}_i = \mathbf{x}_i W^v$, and every position t asks with its own query.

With a batch, every tensor gets a leading B: Q is B × T × D, S is B × T × T.

Scaling: divide by √D?

$S = Q K^T / \sqrt{D} \quad (T \times T);$
$S[t, i] = s_{i,t} / \sqrt{D}$

- If q and k have D independent entries with mean 0 and variance 1,
  - q · k has variance D, so scores grow like √D
- Large scores push softmax to a one-hot vector
  - gradients through the softmax almost vanish; training stalls
- Dividing by √D brings the score variance back to 1

D = 64	**Attention raw scores**	**softmax**
raw q·k	[12, 4, −4]	[0.9997, 0.0003, 0.0000]
scaled q·k / 8	[1.5, 0.5, −0.5]	[0.665, 0.245, 0.090]

PyTorch's F.scaled_dot_product_attention uses the same 1/√D, so you can check your code against it.

Reading an attention heatmap

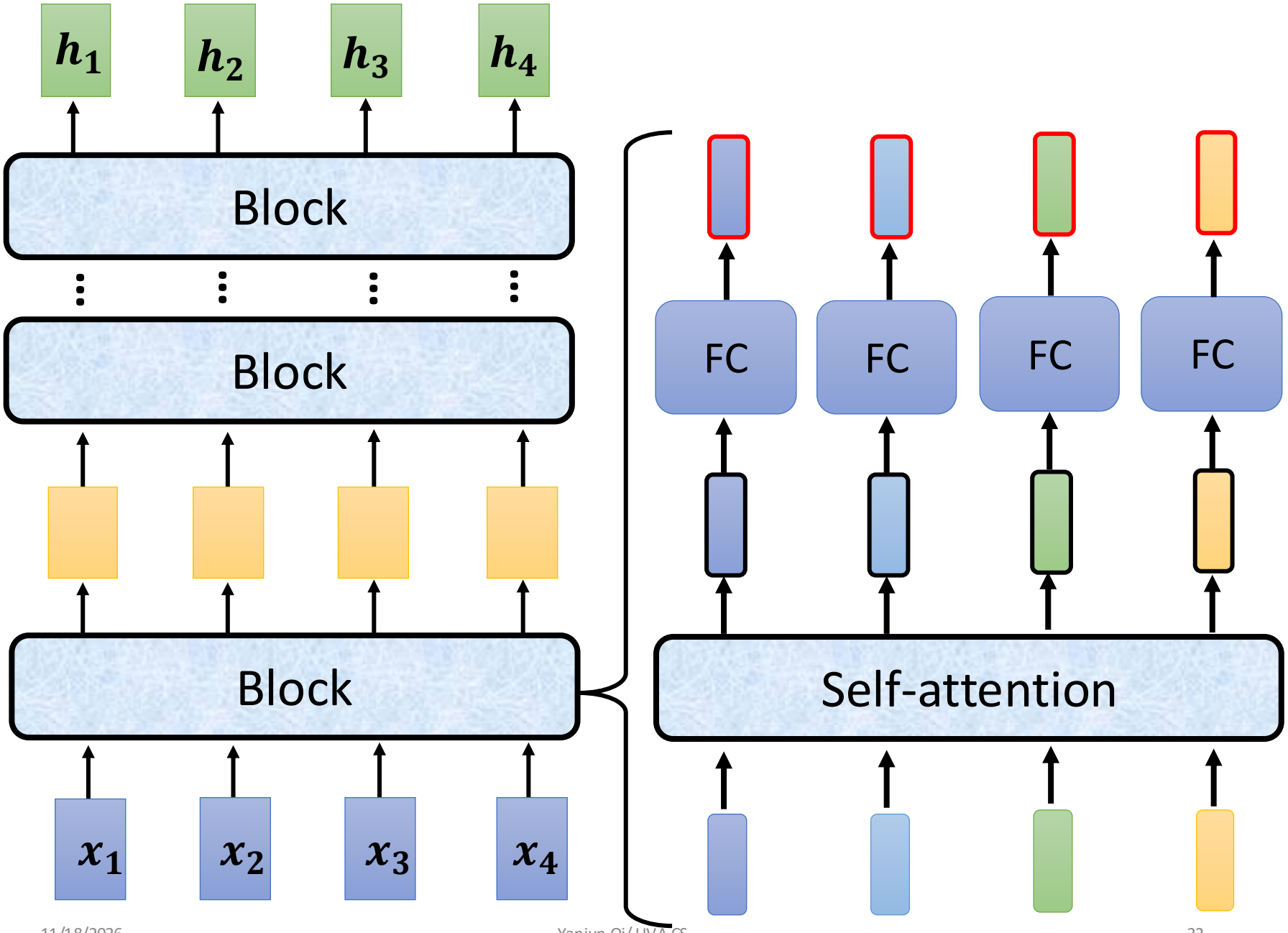


Illustrative weights, not from a trained model

- Row t = how query token t spreads its attention
- Each row sums to 1
- Here the row for **good** puts 0.50 on not
- Real models are often messier: attention can be spread out
- Try it: plot these in HW4!

Roadmap : Today

- Recap (word vectors, attention, bag-of-words)
- Self-attention Basics (Q, K, V, scaling, attention map)
- **Transformer Block Basics (self-attn + heads, positions, residual + norm)**
- Masks & Model Families (encoder-only, encoder-decoder, decoder-only)
- Pretraining a GPT (tokens, next-token loss, Adam, GPT-1/2/3)
- Generating Text (greedy, sampling, beam search, KV cache)
- Post-training Preview (aligning a GPT: RLHF, DPO)

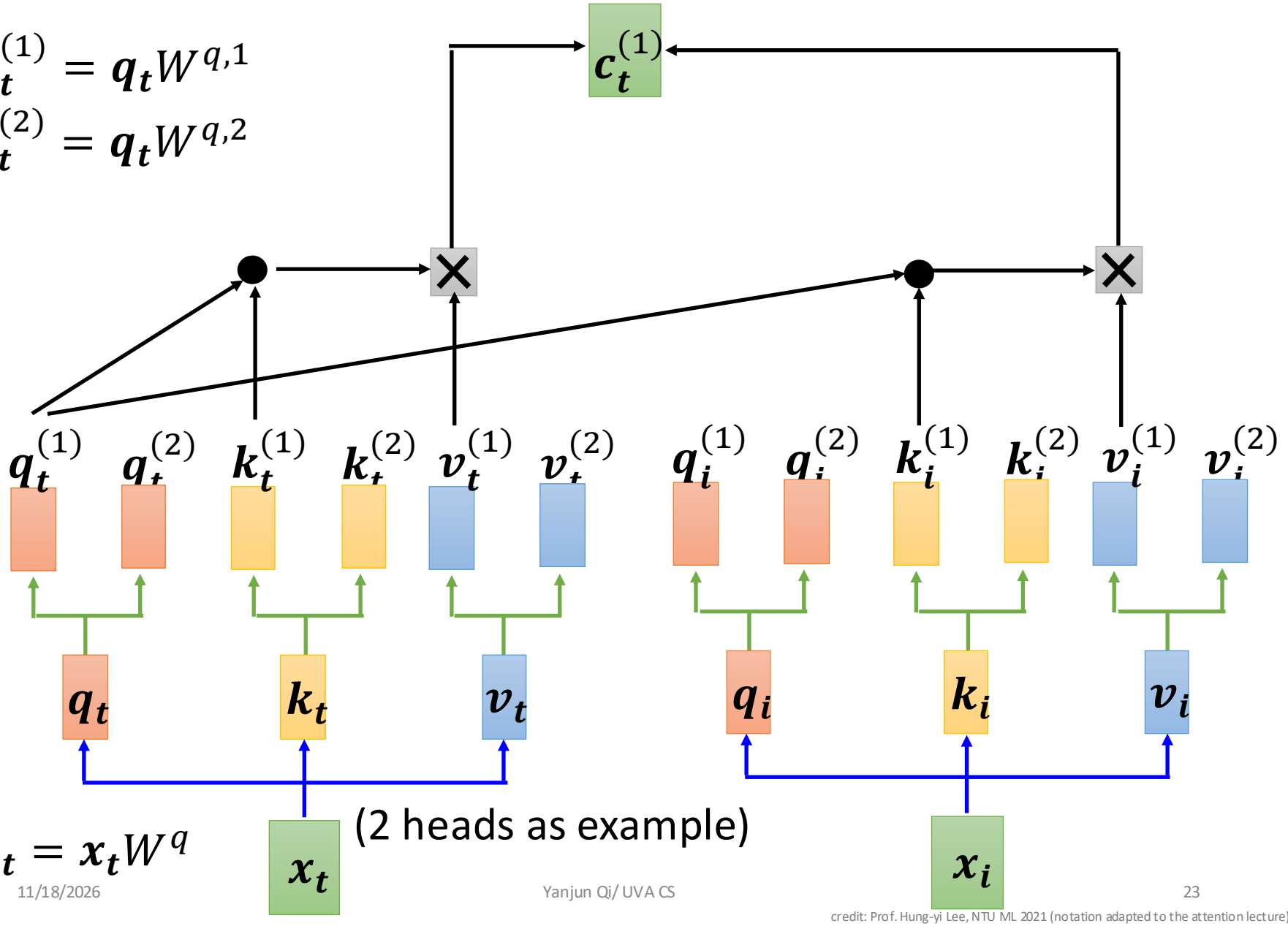


credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

Multi-head Self-attention

Different types of relevance

$$q_t^{(1)} = q_t W^{q,1}$$
$$q_t^{(2)} = q_t W^{q,2}$$

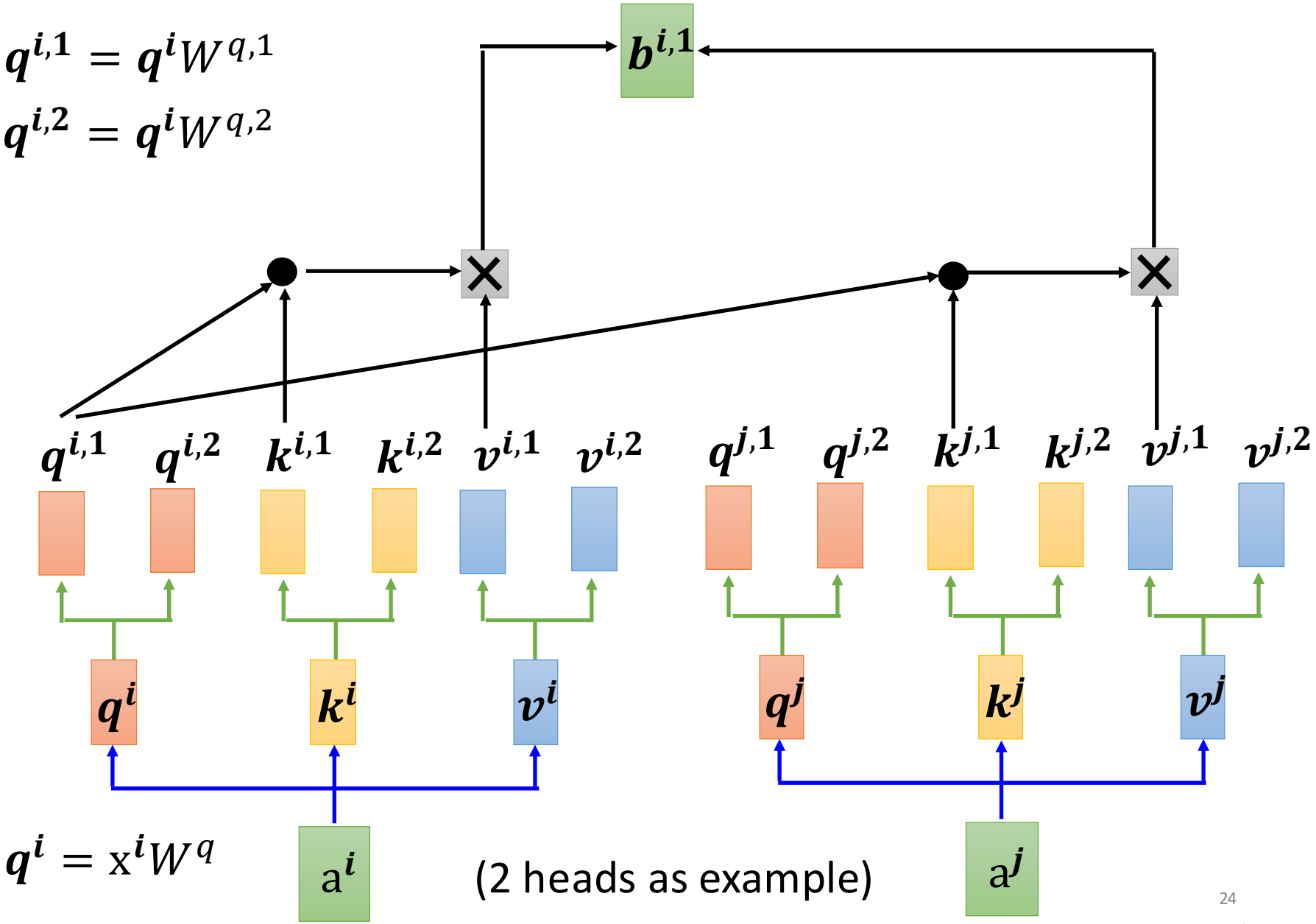


(2 heads as example)

$$q_t = x_t W^q$$

credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

Multi-head Self-attention Different types of relevance



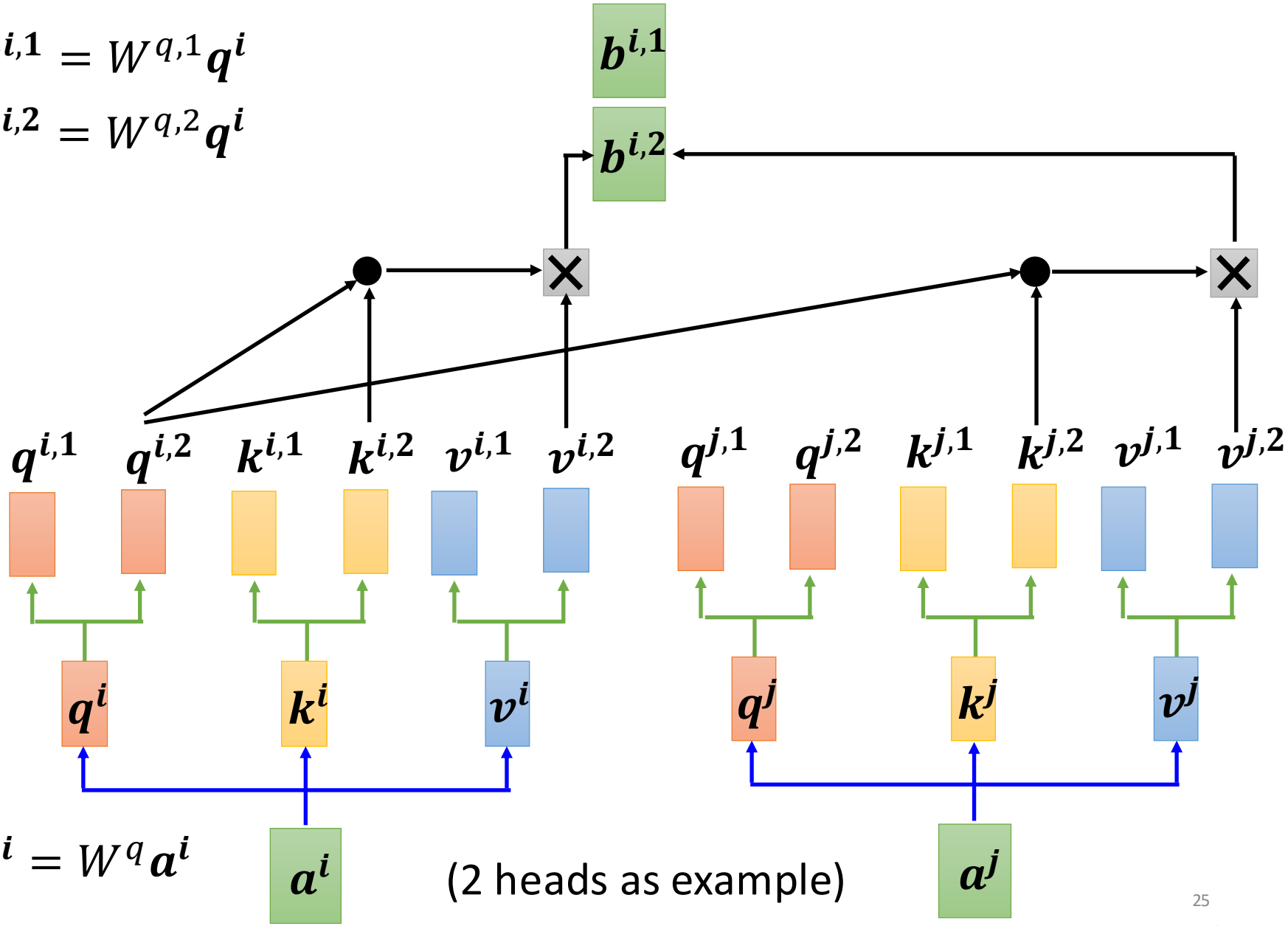
credit: Prof. Hung-yi Lee,

Multi-head Self-attention

Different types of relevance

$$q^{i,1} = W^{q,1} q^i$$

$$q^{i,2} = W^{q,2} q^i$$



$$q^i = W^q a^i$$

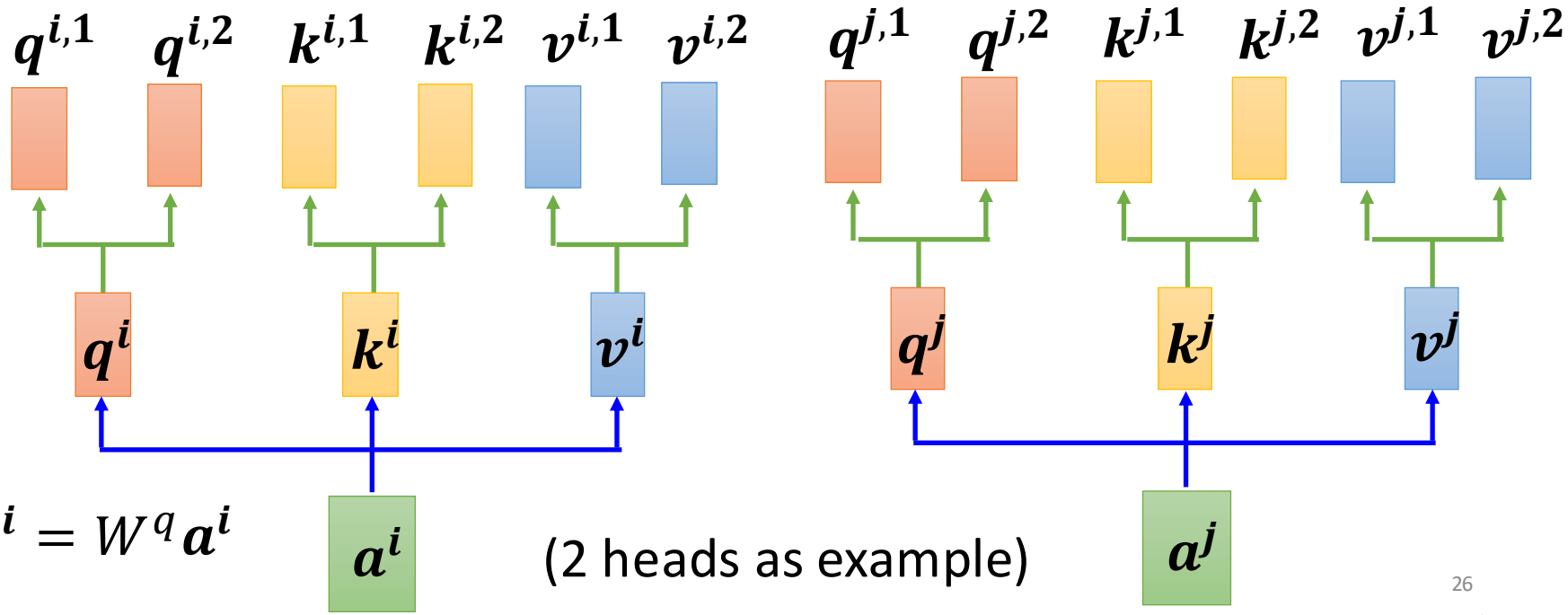
(2 heads as example)

credit: Prof. Hung-yi Lee,

Multi-head Self-attention

Different types of relevance

$$b^i = W^O \begin{bmatrix} b^{i,1} \\ b^{i,2} \end{bmatrix}$$



$q^i = W^q a^i$

(2 heads as example)

credit: Prof. Hung-yi Lee,

Question: does self-attention know word order?

"dog bites man"

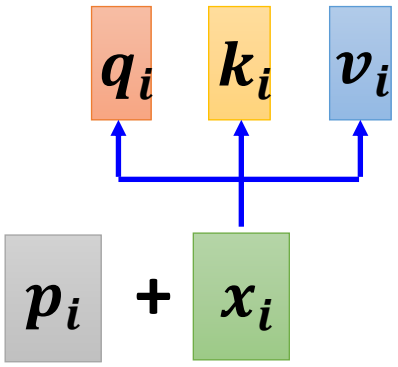
"man bites dog"

- Shuffle the rows of X with a permutation matrix P: $X \rightarrow PX$
  - What happens to $\text{softmax}(QK^T/\sqrt{D})V$?
  - What happens after a mean pool over tokens?
- Try to answer these with a small experiment in HW!

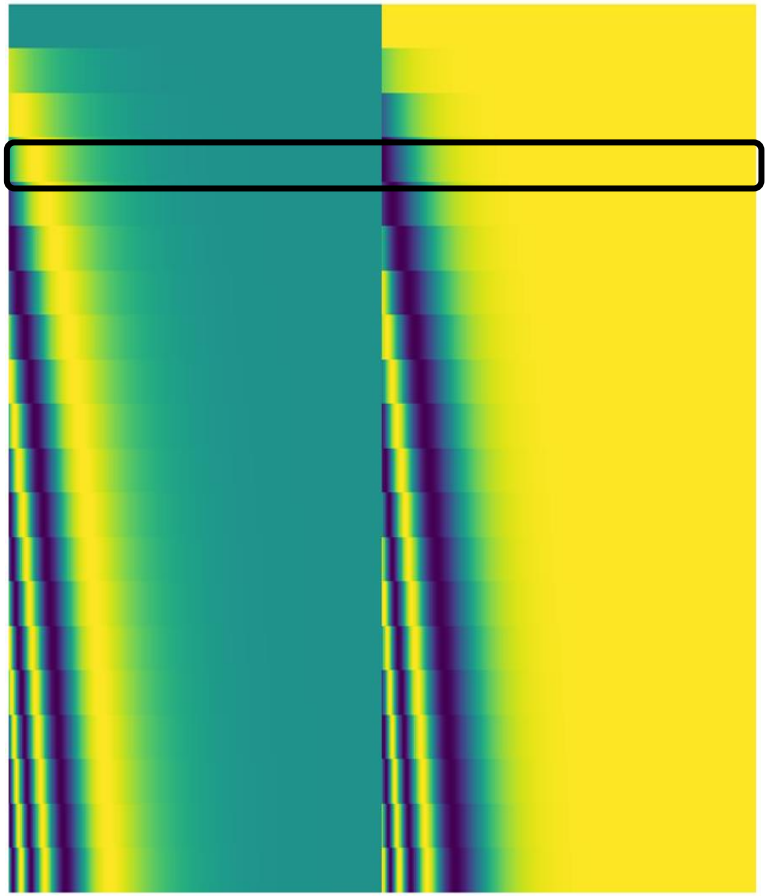
Many different types !!!!

Positional Encoding

- No position information in self-attention.
- Each position has a unique positional vector p_i
- **(1) hand-crafted**
- **(2) learned from data**



Each row represents a positional vector p_i

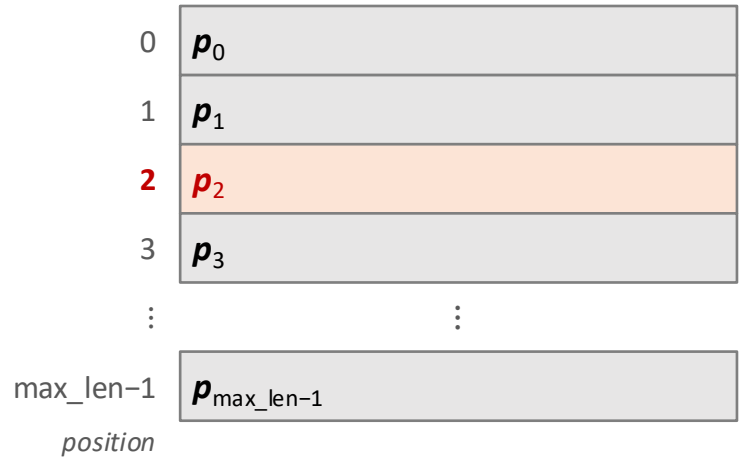




credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

e.g., Learned position embedding: a lookup table

P: max_len rows × D trainable numbers



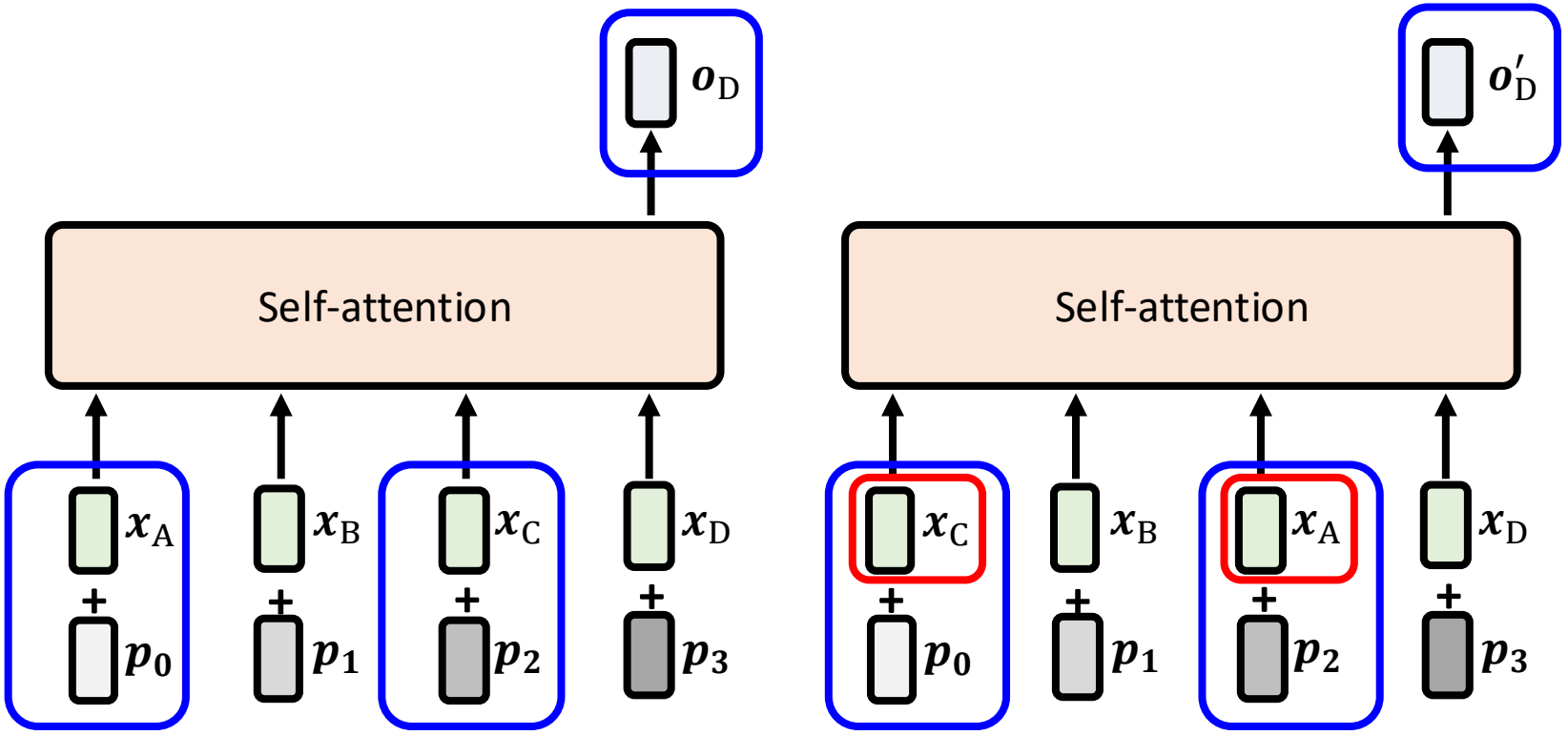
Position 2 input = $x_2 + p_2$

- Position i reads row i of P
- Rows start random; backprop trains them
- Same word, two positions: same x, different p_i
- BERT: 512 rows. GPT-2: 1024 rows.
- **Limit:** no row for positions ≥ max_len

```
pos_embed = nn.Embedding(max_len, D)    # rows indexed by position, not by token id
```

Sinusoidal: p_i = a fixed formula. **Learned:** p_i = fit to the data.

Absolute positional embedding: swap two words

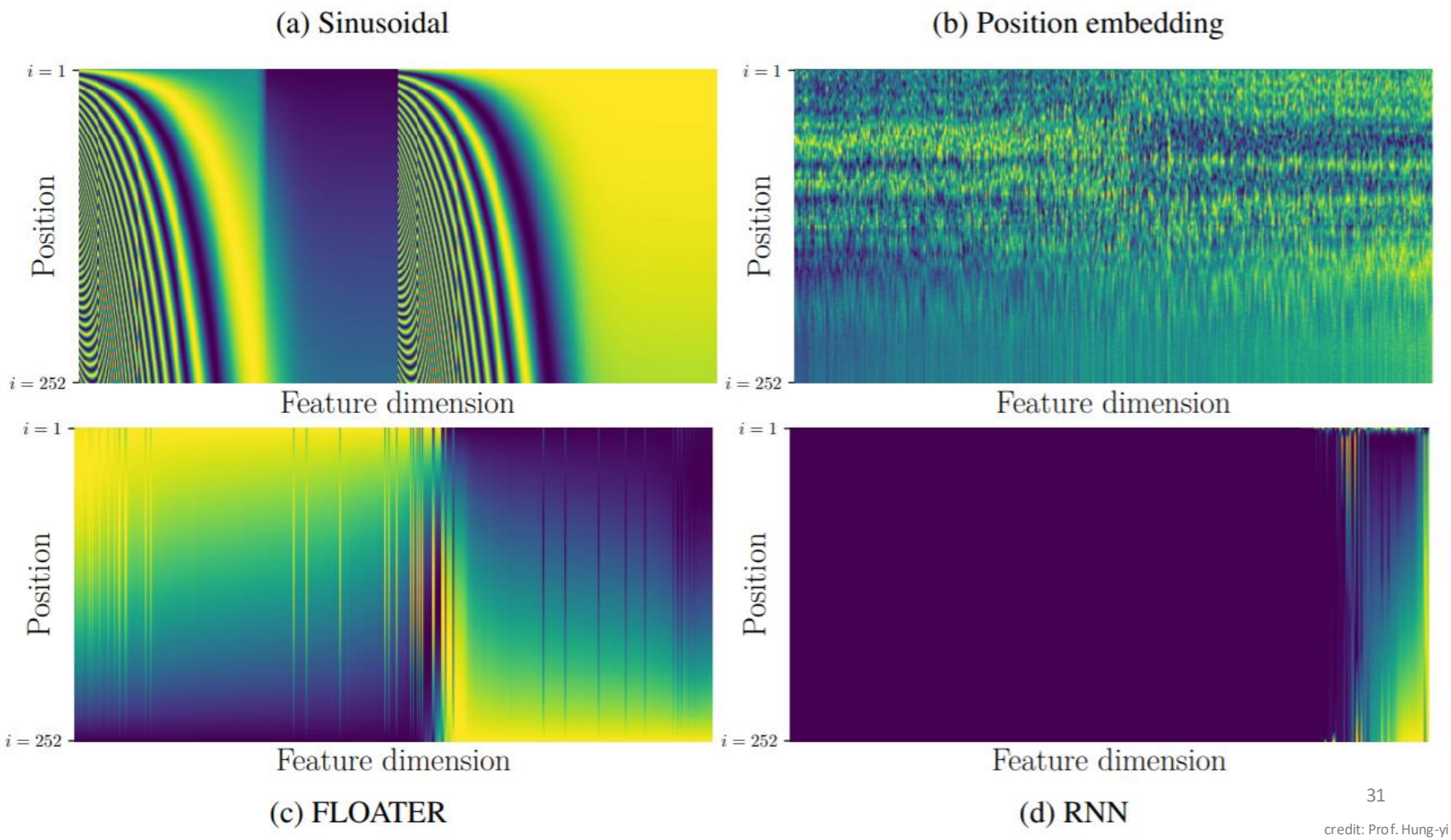


credit: Prof. Hung-yi Lee, ML 2026, positional embedding lecture

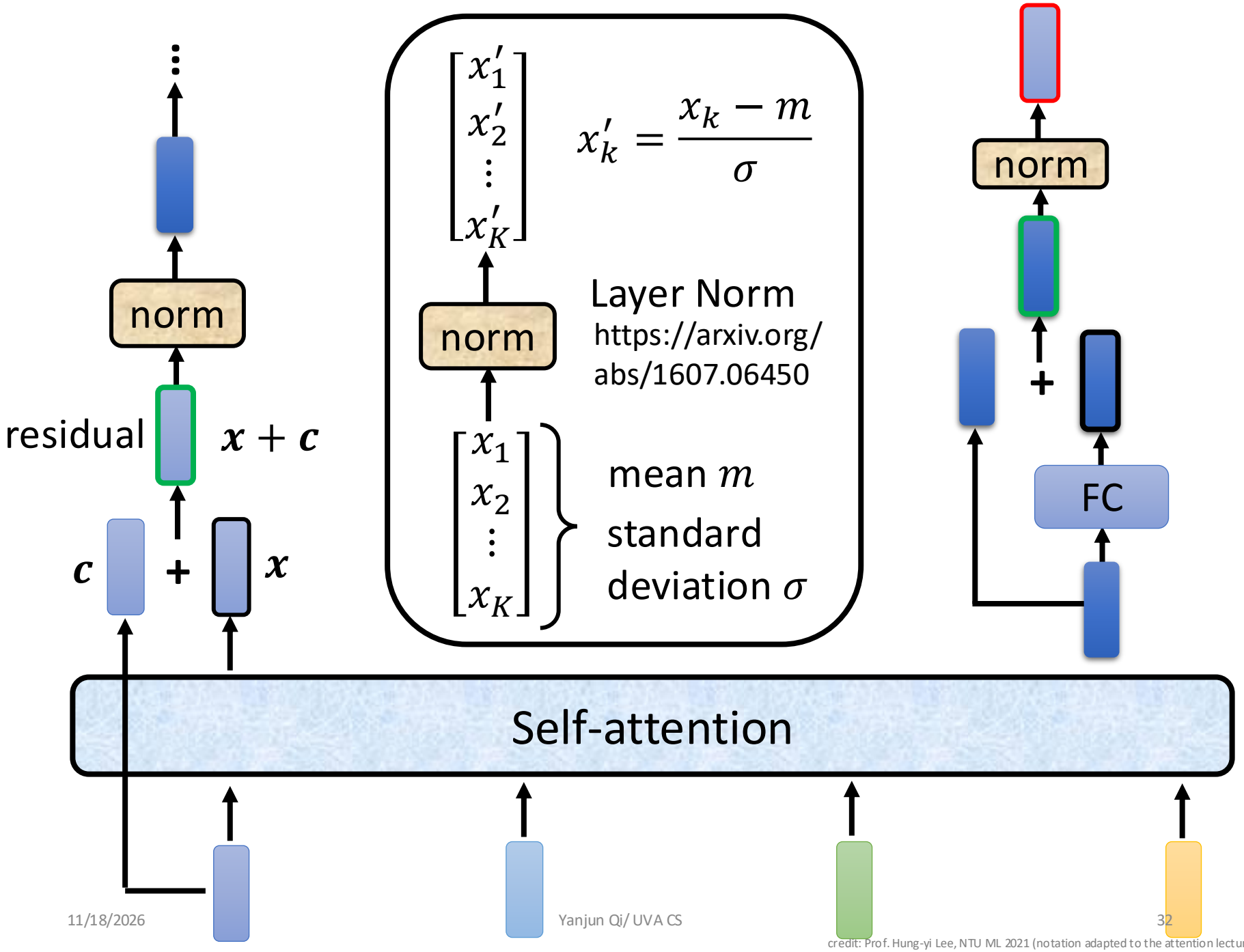
https://arxiv.org/abs/2003.09229

Table 1. Comparing position representation methods

Methods	Inductive	Data-Driven	Parameter Efficient
Sinusoidal (Vaswani et al., 2017)	✓	✗	✓
Embedding (Devlin et al., 2018)	✗	✓	✗
Relative (Shaw et al., 2018)	✗	✓	✓
This paper	✓	✓	✓



credit: Prof. Hung-yi Lee,



credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

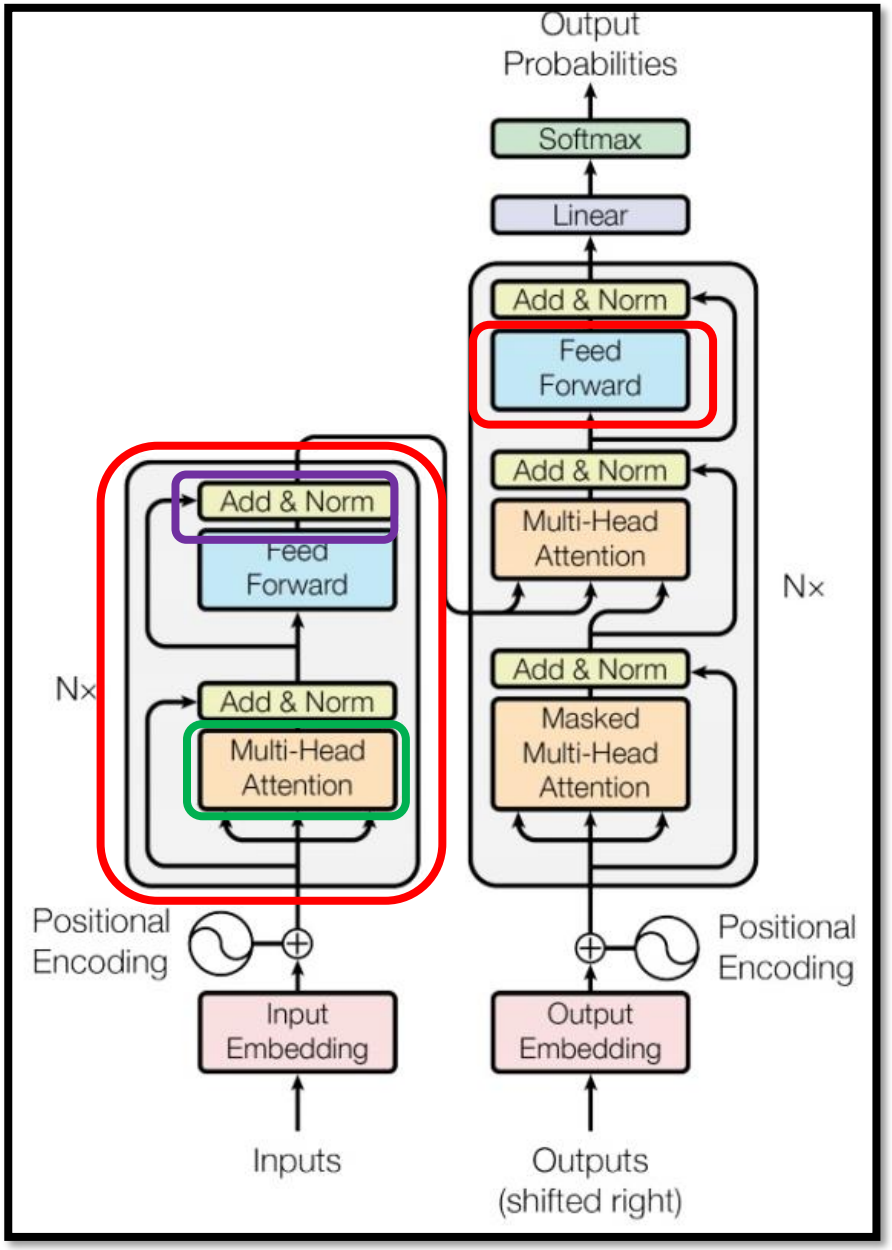
Counting parameters (one layer)

Component	**Shape**	**Count**	**D = 64**
W_q, W_k, W_v (no bias)	3 × (D × D)	$3D^2$	12,288
W_O: combine heads (no bias)	D × D	D^2	4,096
FC / Feed Forward: D → 4D → D (+ bias)	D×4D + 4D×D + 5D	$8D^2 + 5D$	33,088
2 LayerNorms (scale + shift)	2 × 2D	4D	256
Total (one layer)			**49,728**

- Each of h heads uses D/h dimensions, so W_q, W_k, W_v stay D × D for any number of heads
- Not counted here: the token and position embeddings (input) and the output layer that scores all V tokens
- Check yourself: `sum(p.numel() for p in block.parameters())`

Transformer

https://arxiv.org/abs/1706.03762



UVA CS 4774: Machine Learning

S3: Lecture 18:

Transformer Basic Architectures

Dr. Yanjun Qi

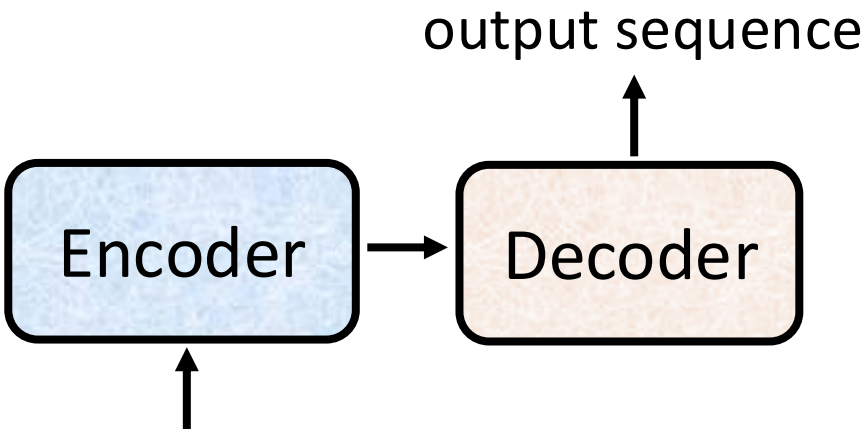
University of Virginia
Department of Computer Science

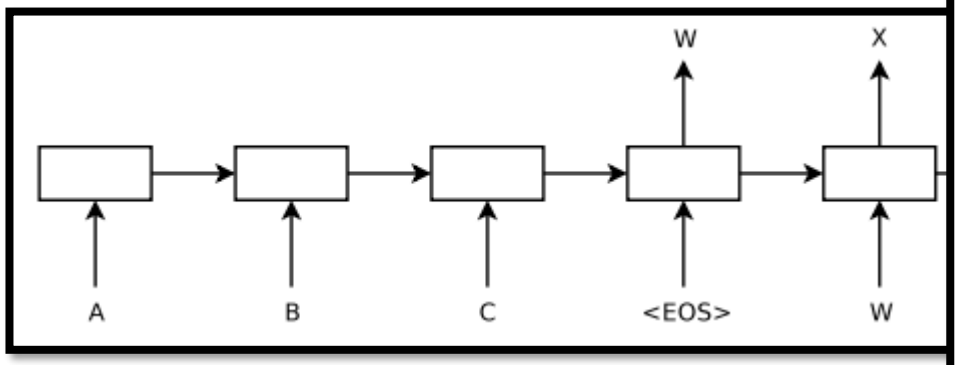
Module II

Roadmap : Today

- Recap (word vectors, attention, bag-of-words)
- Self-attention Basics (Q, K, V, scaling, attention map)
- Transformer Block Basics (self-attn + heads, positions, residual + norm)
- **Masks & Model Families (encoder-only, encoder-decoder, decoder-only)**
- Pretraining a GPT (tokens, next-token loss, Adam, GPT-1/2/3)
- Generating Text (greedy, sampling, beam search, KV cache)
- Post-training Preview (aligning a GPT: RLHF, DPO)

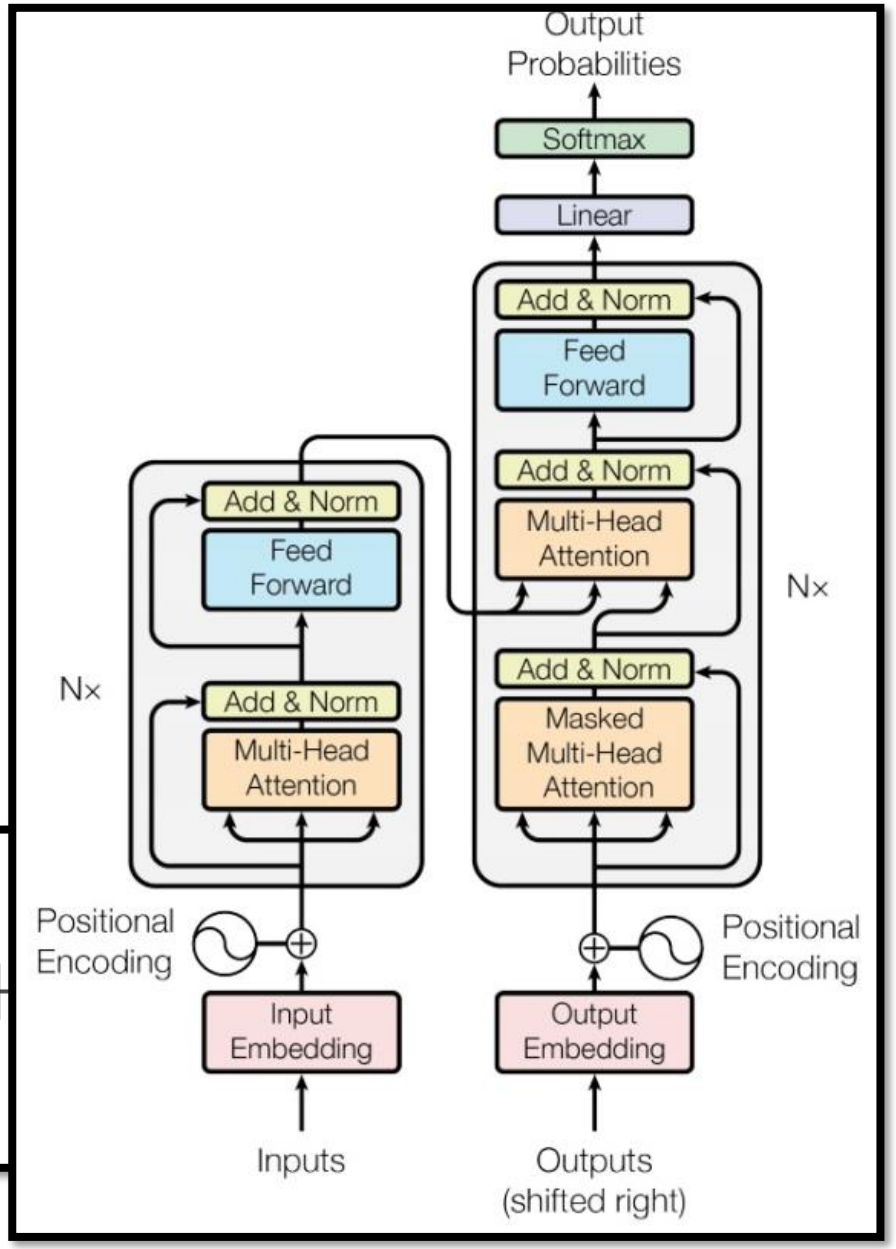
Recap: Seq2seq





Sequence to Sequence Learning with Neural Networks

https://arxiv.org/abs/1409.3215

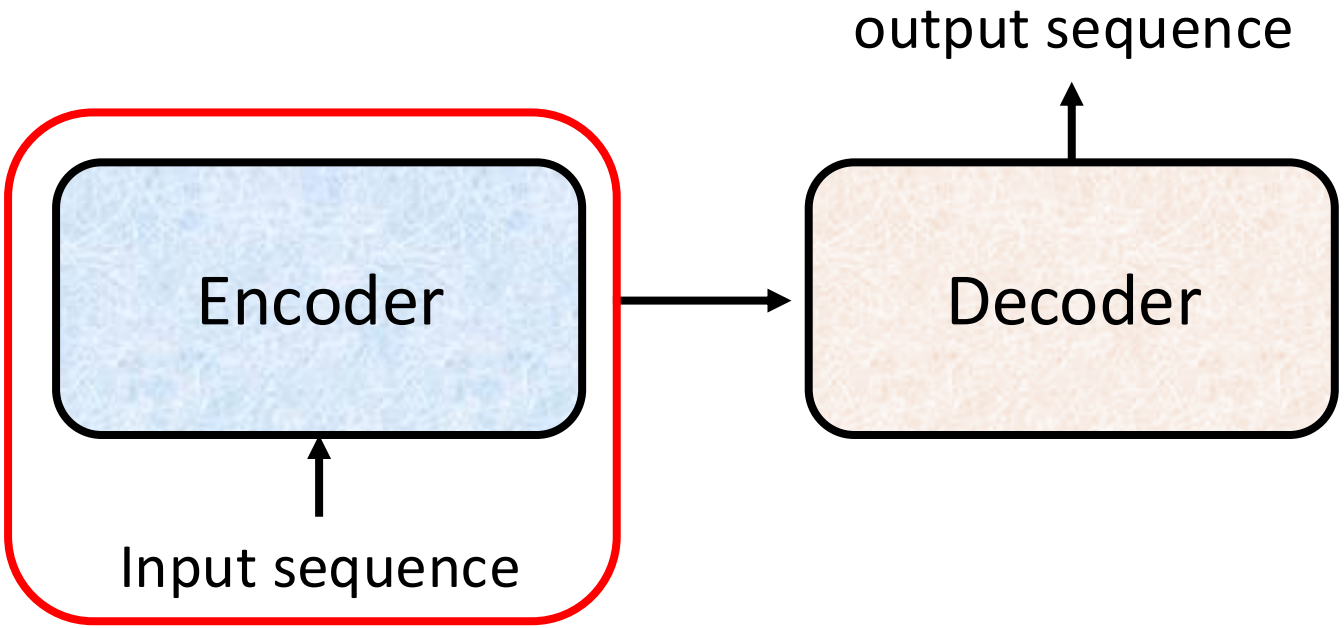


Transformer

https://arxiv.org/abs/1706.03762

credit: Prof. Hung-yi Lee,

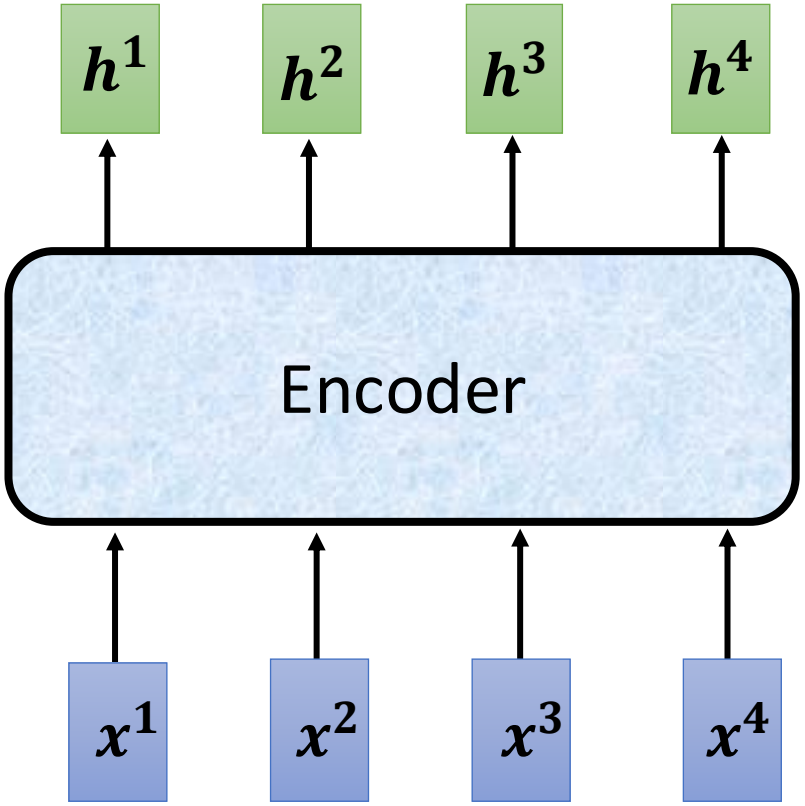
Encoder



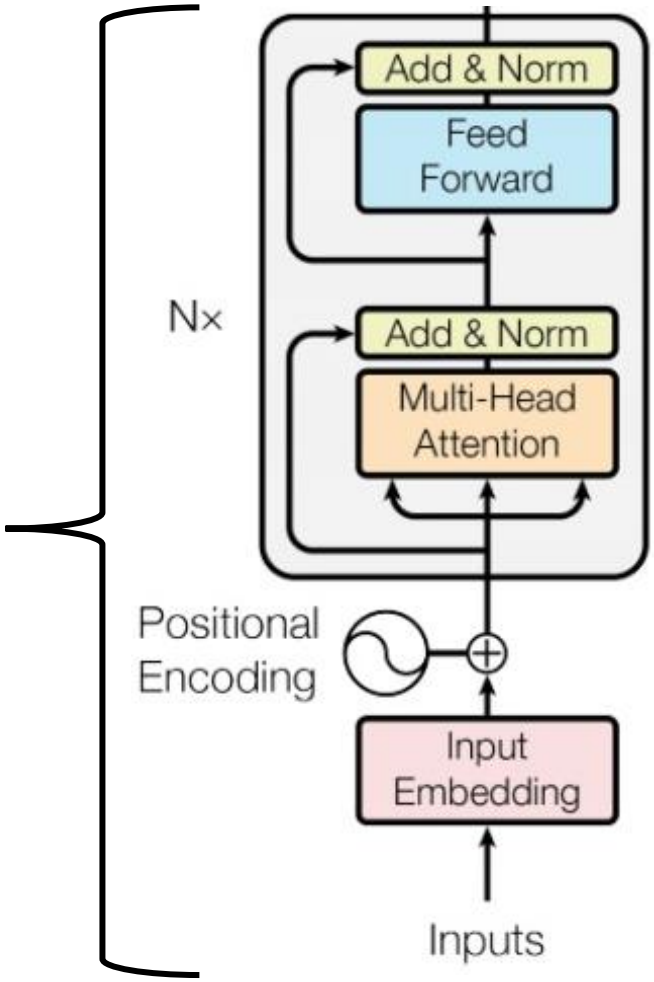
credit: Prof. Hung-yi Lee,

Encoder

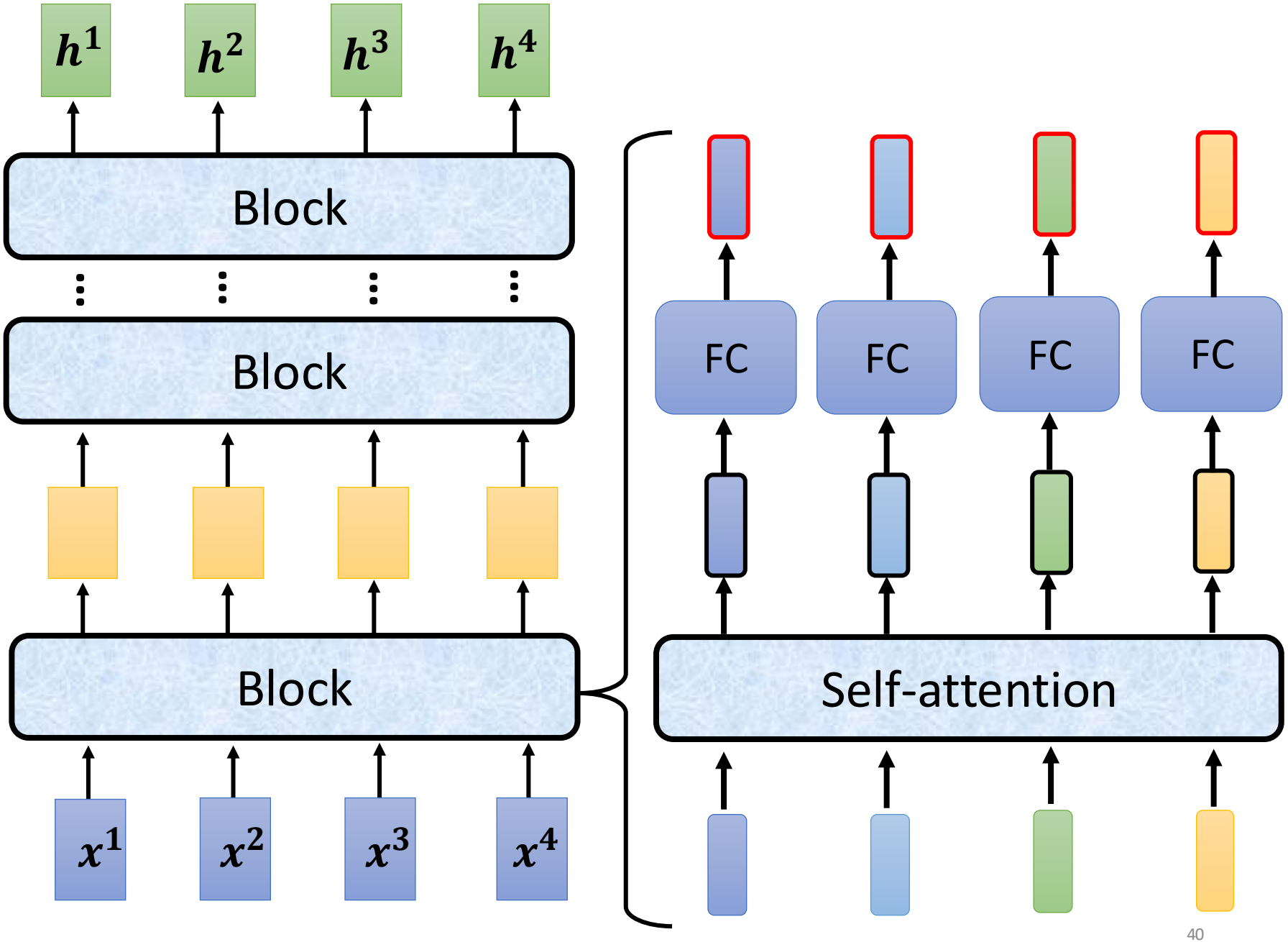
You can use **RNN** or **CNN**.



Transformer's Encoder



credit: Prof. Hung-yi Lee,

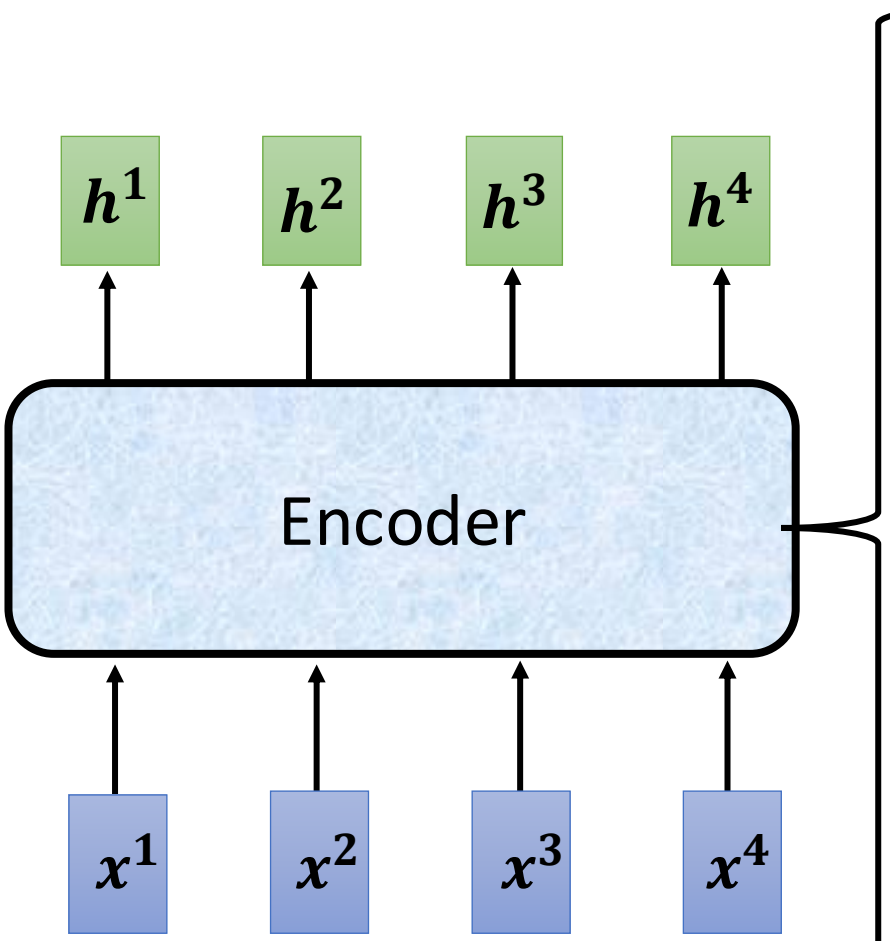


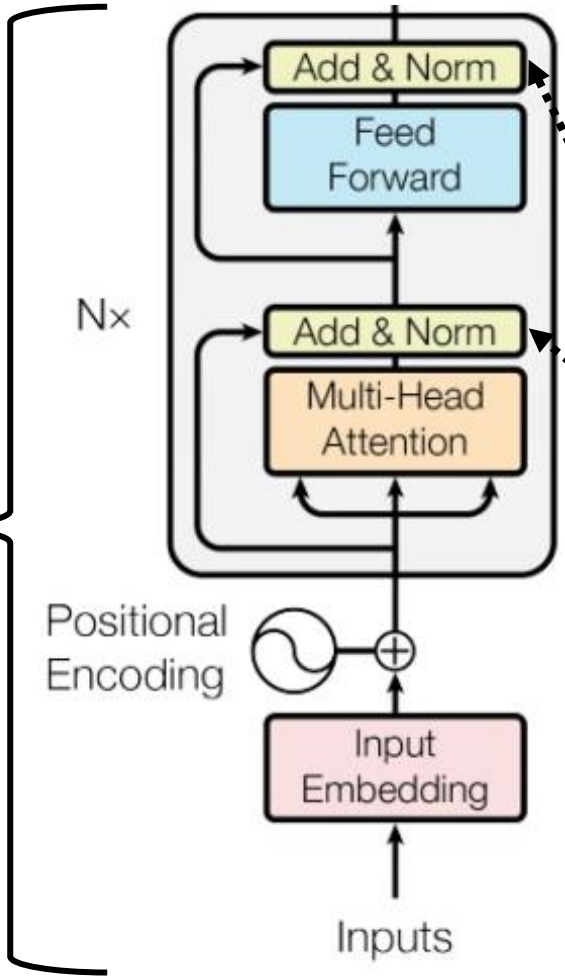
credit: Prof. Hung-yi Lee,

BERT

use the **same** network architecture as **transformer encoder**.







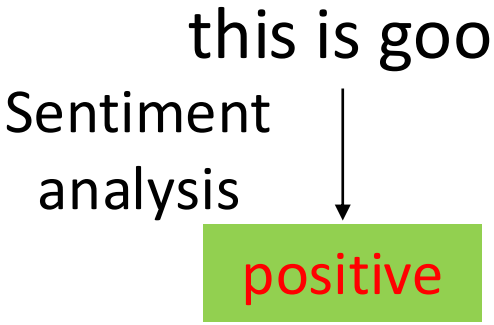
Residual + Layer norm

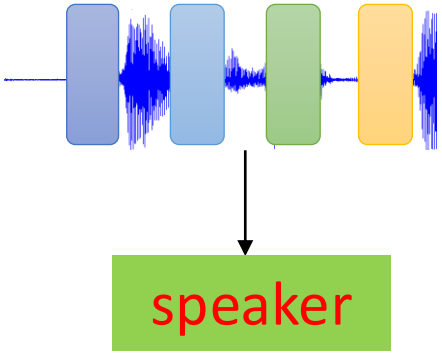
credit: Prof. Hung-yi Lee,

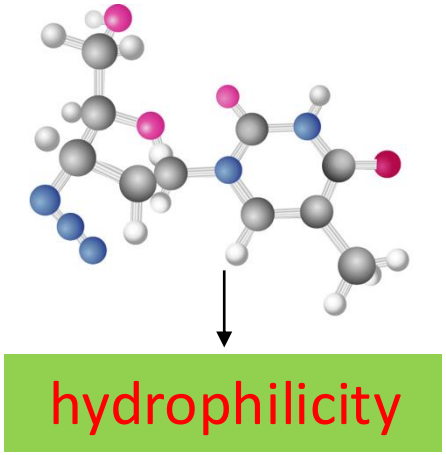
Example Applications

this is good

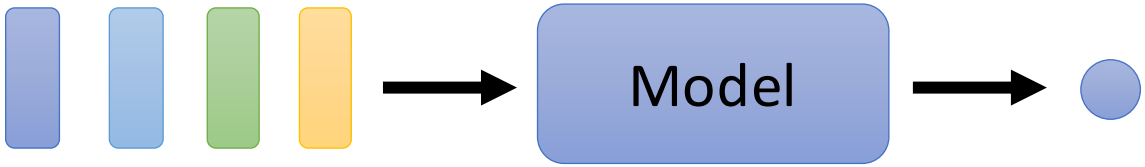
Sentiment analysis







- The whole sequence has a label.





We can just train them with Cross entropy loss for multi-class / binary classification

credit: Prof. Hung-yi Lee,

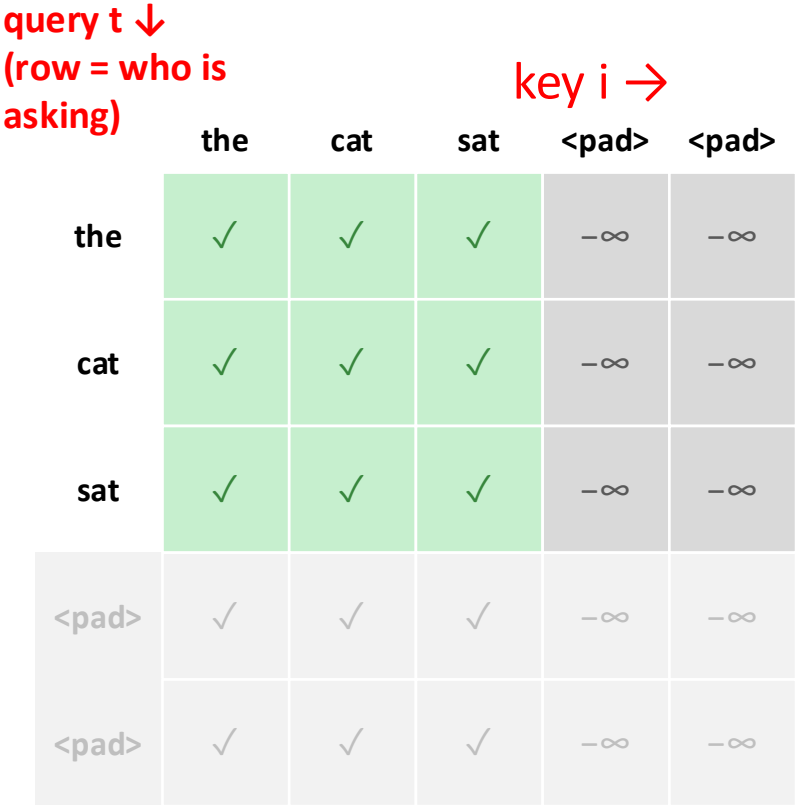
How BERT adapts to classification (many ways!)

A. First Way: [CLS] token.

1. BERT puts a special `[CLS]` token at the start of every input: `[CLS] the movie was good [SEP]; and`
2. **Use [CLS]'s output vector.** After the encoder layers, the final vector at the [CLS] position represents the whole sequence. With full, unmasked self-attention, that position can attend to every token.
3. **Add a small classifier head.** One linear layer (D → number of classes) and a softmax.
4. **Fine-tune everything.** Train the head and all the BERT weights together with cross-entropy on the labelled task, usually for only a few epochs.

B. Mean pooling instead of [CLS]: average the output vectors over the real tokens, skipping pads. Common for sentence embeddings (Sentence-BERT) and for simple classifiers trained from scratch, like the one in HW4. With padding, it must be a masked mean: `(h * mask).sum(1) / mask.sum(1)`, which uses the padding mask on the next page.

Masks (1): key-padding mask



Grey rows: pad queries are computed, then ignored by the loss; They are computed only because the matrix product computes every row, and the loss ignores them, so nothing is learned from them

- Sequences in a batch are right-padded to the same length
- No real token should attend to <pad>
- Set those score columns to −∞ before the softmax: exp(−∞) = 0
- Masking after the softmax would leave rows that no longer sum to 1

Code Example: key-padding mask in three lines

```
key_mask = mask.unsqueeze(1)                          # (B, 1, L): one 0/1 row per sentence
scores = scores.masked_fill(key_mask == 0, float("-inf"))    # pad columns -> -inf
attn = torch.softmax(scores, dim=-1)                  # after the mask: exp(-inf) = 0
```

One 0/1 row per sentence, broadcast down every query row

Example Toy batch: B = 2 sentences, L = 5

```
ids  = [[1, 2, 3, 0, 0],   # the cat sat <pad> <pad>
        [4, 5, 6, 7, 8]]   # a dog ran home fast
mask = [[1., 1., 1., 0., 0.],
        [1., 1., 1., 1., 1.]]  # shape (2, 5)
```

Line 1: key_mask = mask.unsqueeze(1), shape (2, 1, 5)

```
[[[1., 1., 1., 0., 0.]],
 [[1., 1., 1., 1., 1.]]]
```

key_mask == 0, broadcast:

```
  sentence 0          sentence 1
[0, 0, 0, 1, 1]     [0, 0, 0, 0, 0]
[0, 0, 0, 1, 1]     [0, 0, 0, 0, 0]
[0, 0, 0, 1, 1]     [0, 0, 0, 0, 0]
[0, 0, 0, 1, 1]     [0, 0, 0, 0, 0]
[0, 0, 0, 1, 1]     [0, 0, 0, 0, 0]
```

Line 2: scores.masked_fill(key_mask == 0, -inf)

sentence 0 (2 pads)

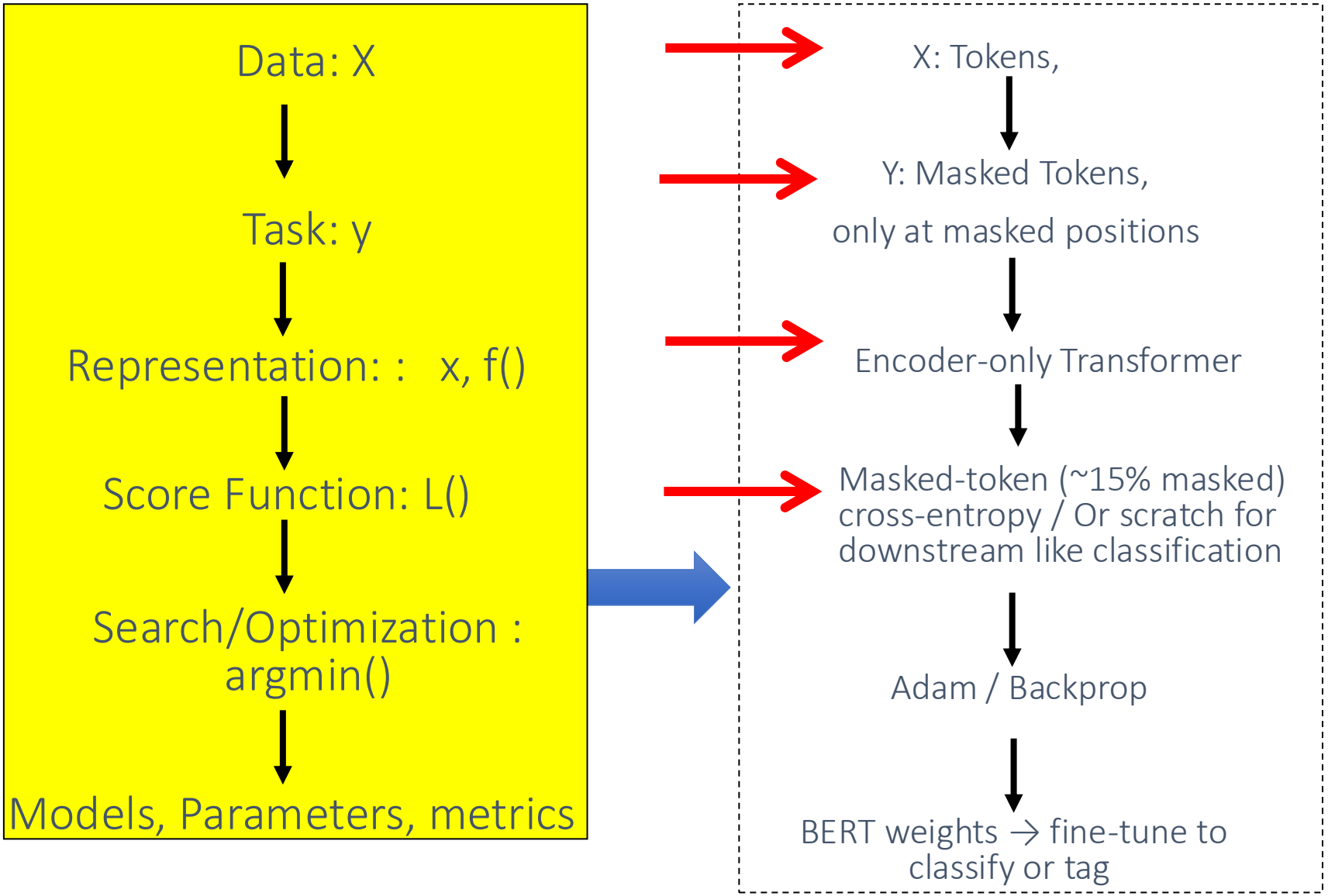
```
[ 0.26, -0.25,  0.25,  -inf,  -inf]
[-0.06, -0.17,  0.30,  -inf,  -inf]
[ 0.51, -0.13, -0.30,  -inf,  -inf]
[ 0.00,  0.00,  0.00,  -inf,  -inf]
[ 0.00,  0.00,  0.00,  -inf,  -inf]
```

sentence 1 (no pads)

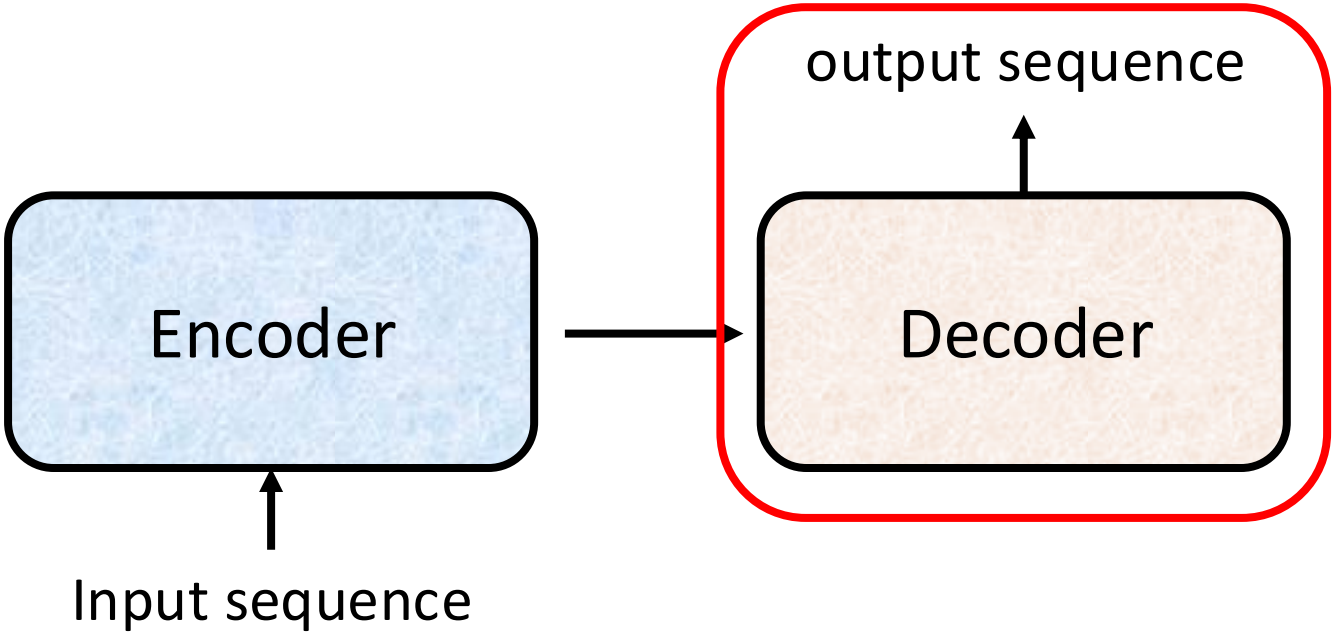
```
[ 0.21,  0.12,  0.07, -0.37, -0.13]
[-0.11, -0.13, -0.08,  0.33,  0.04]
[-0.08,  0.02, -0.05,  0.12, -0.01]
[-0.02, -0.02,  0.03, -0.07,  0.21]
[ 0.11,  0.10,  0.11, -0.25, -0.27]
```

Untrained random weights; same pattern as the previous page

BERT-like: Encoder-only Architecture and Training



Decoder – Autoregressive (AT)



credit: Prof. Hung-yi Lee,

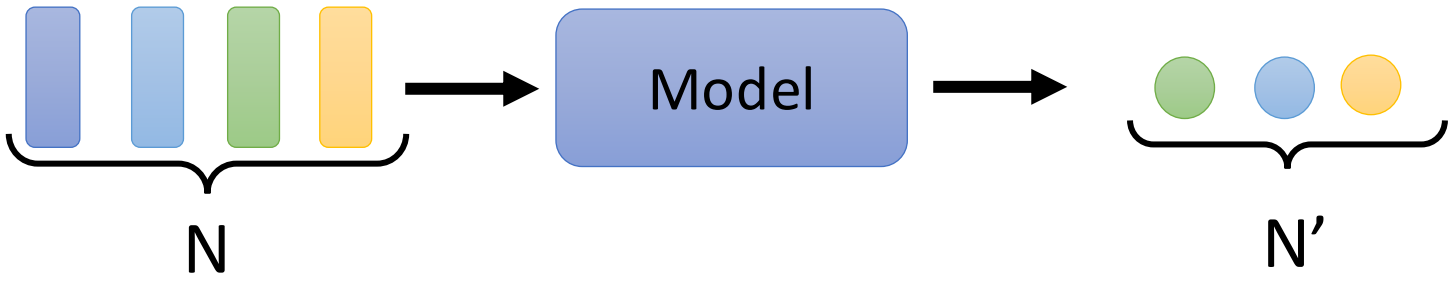
Example Applications :

Translation

Speech recognition

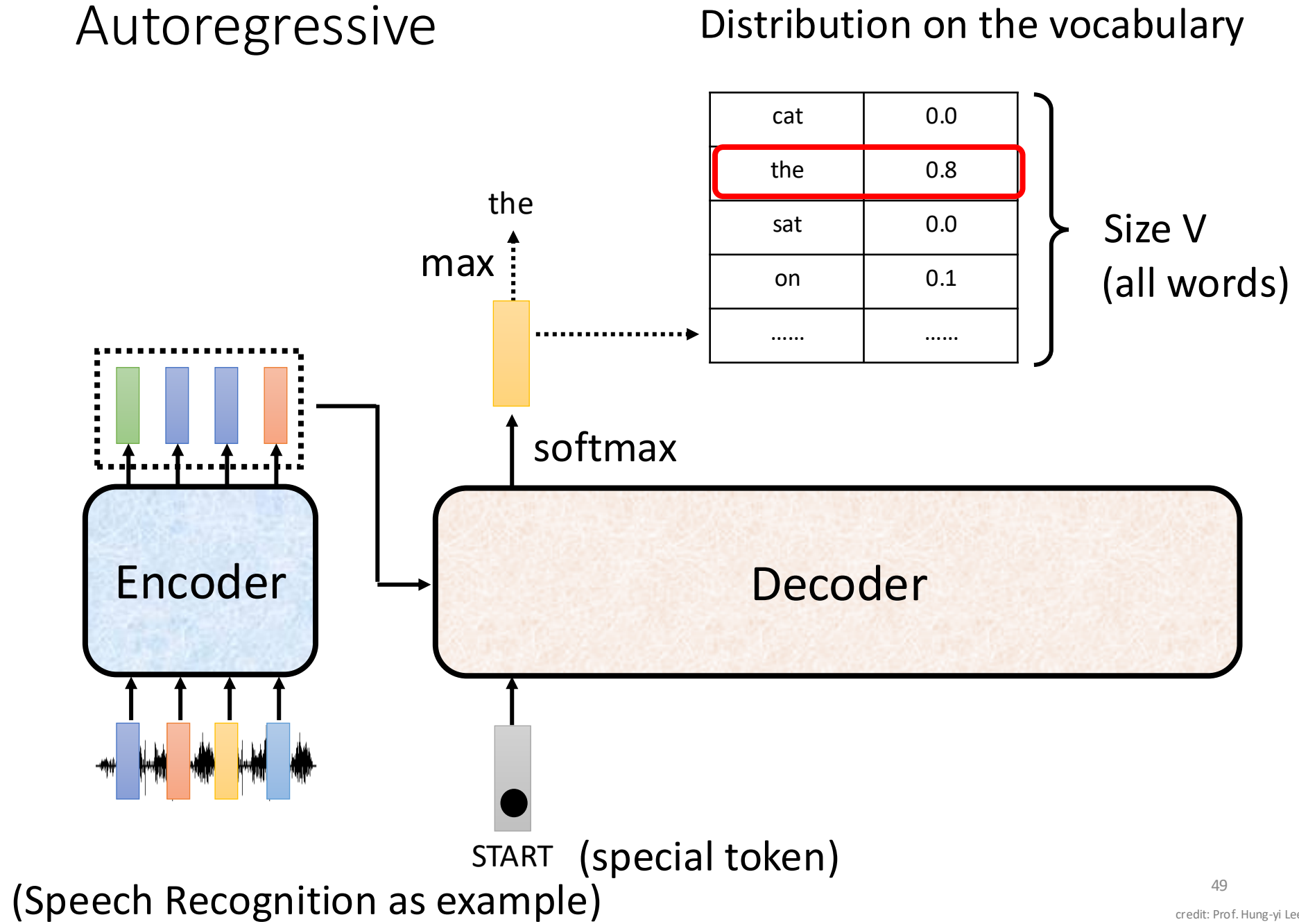
chatbot

- Model decides the number of outputs itself.



credit: Prof. Hung-yi Lee,

Autoregressive

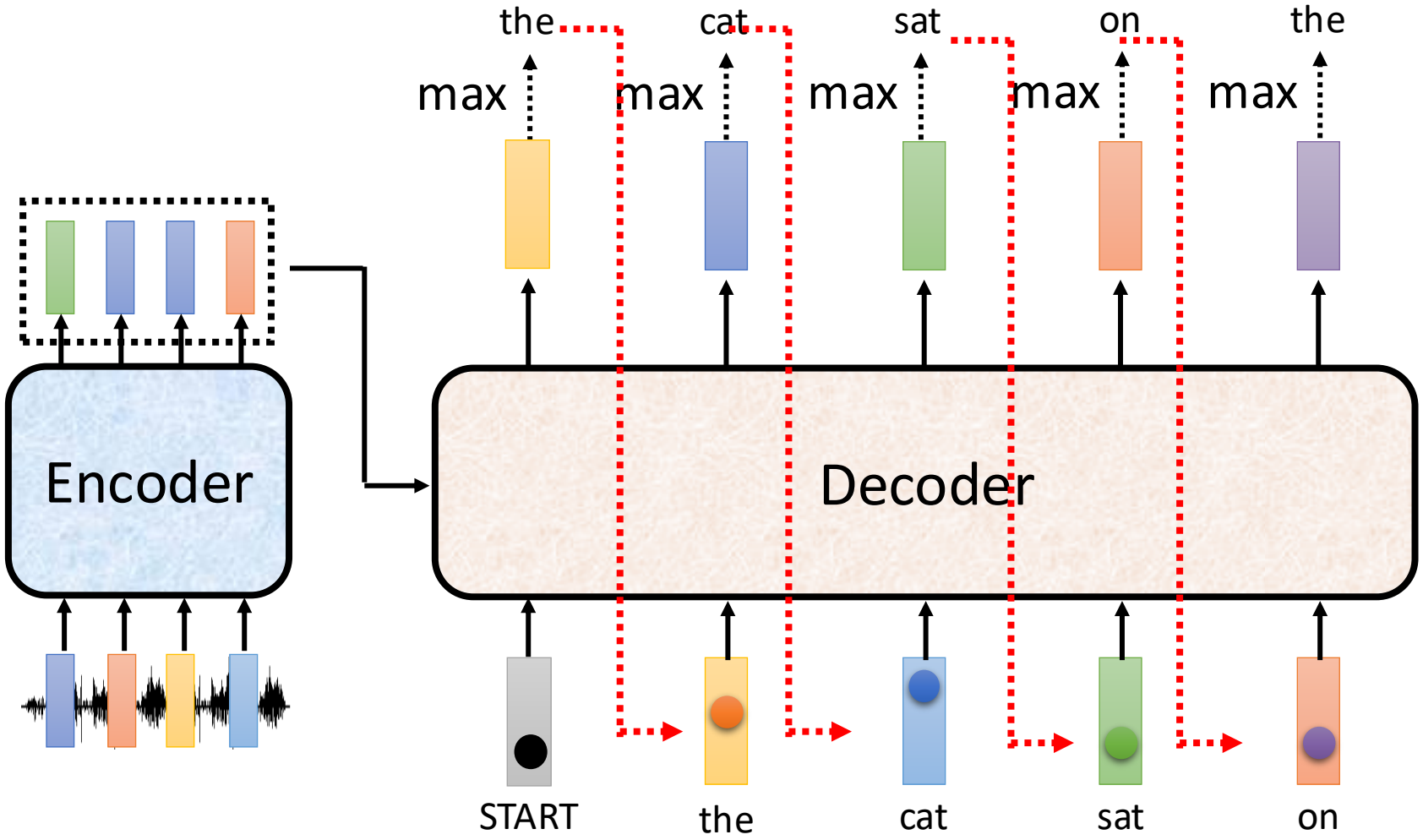


(Speech Recognition as example)

credit: Prof. Hung-yi Lee,

Autoregressive

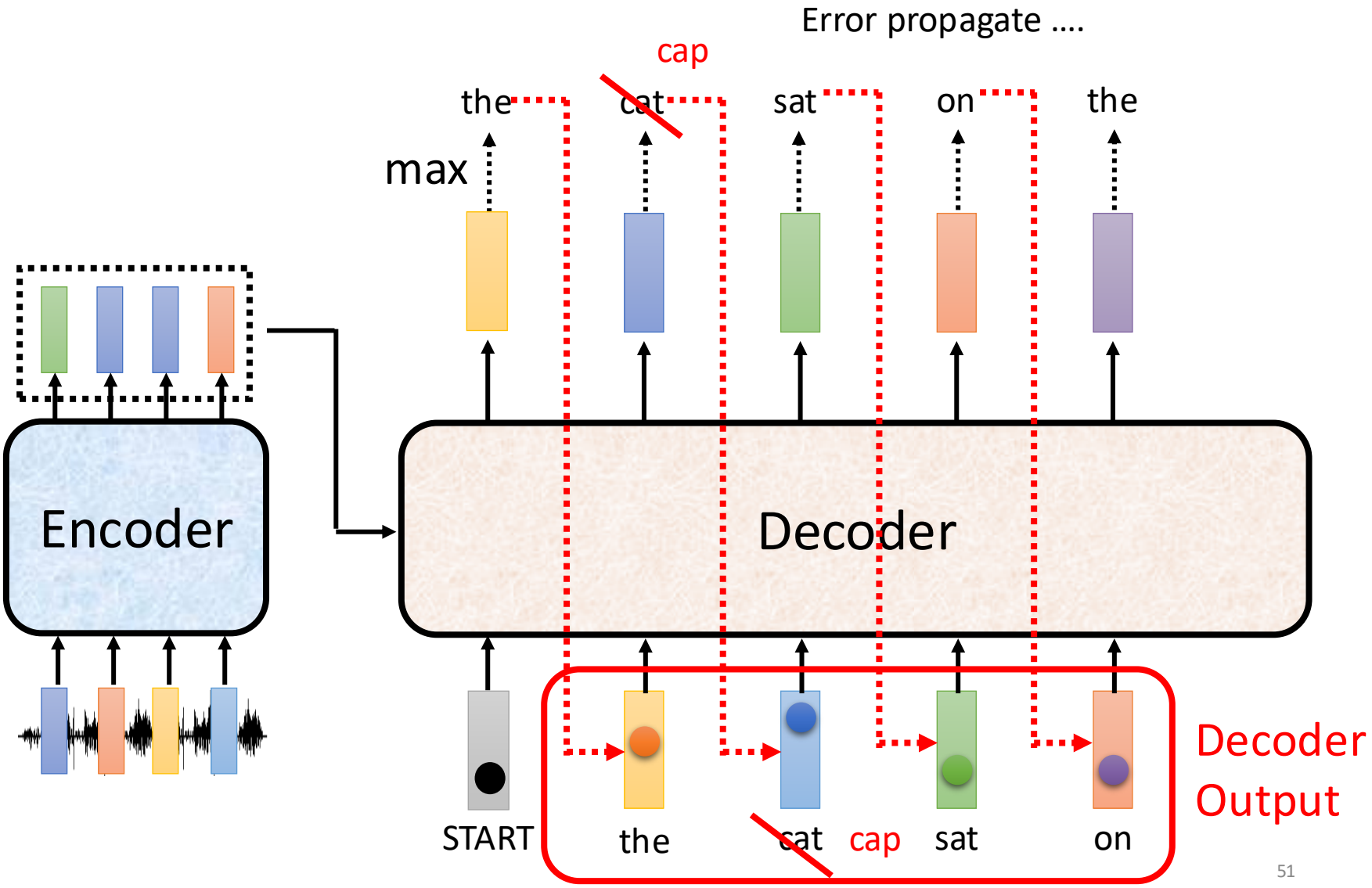
Each step takes everything produced so far and predicts one more token



credit: Prof. Hung-yi Lee,

Autoregressive

Each step takes everything produced so far and predicts one more token

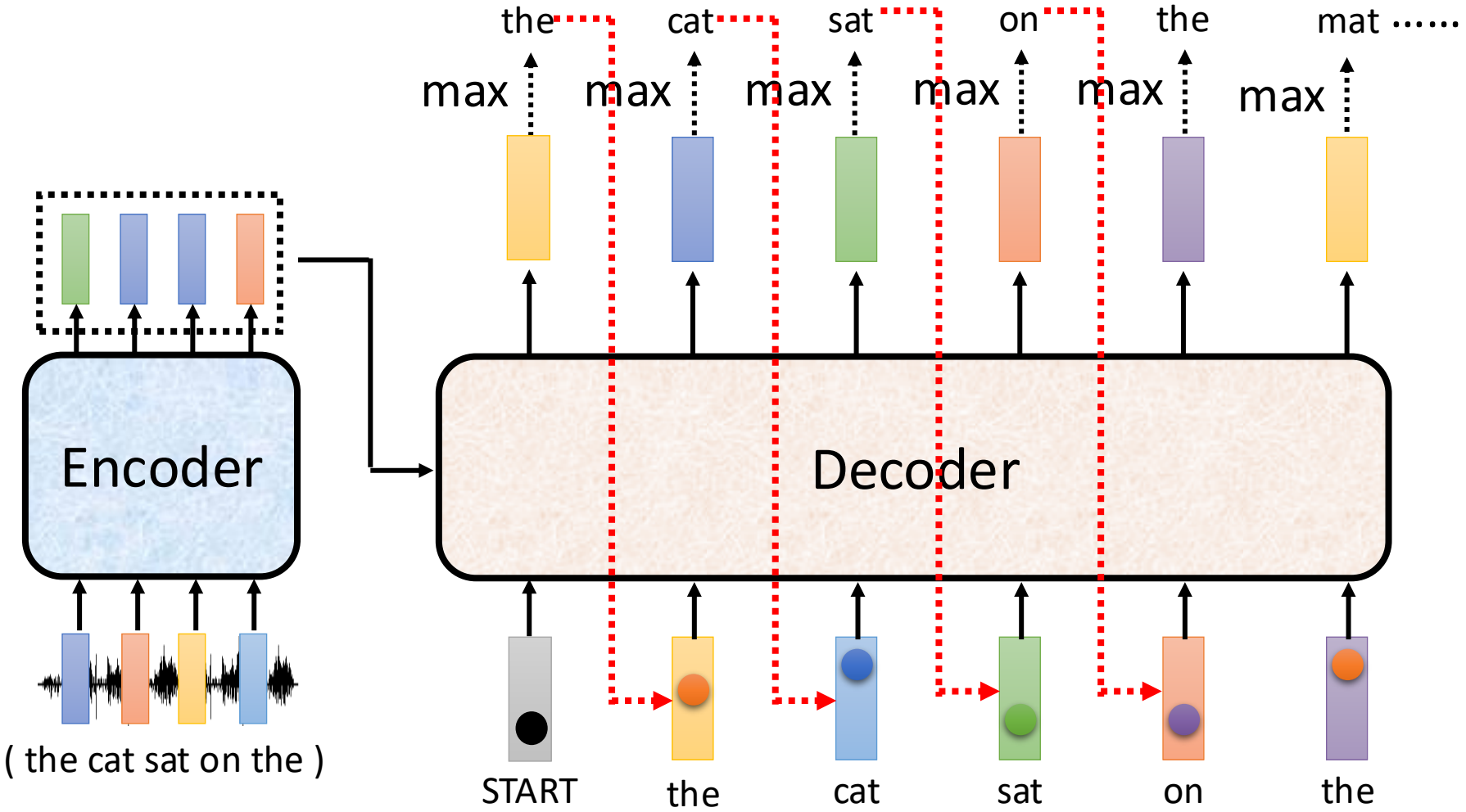


credit: Prof. Hung-yi Lee,

Autoregressive

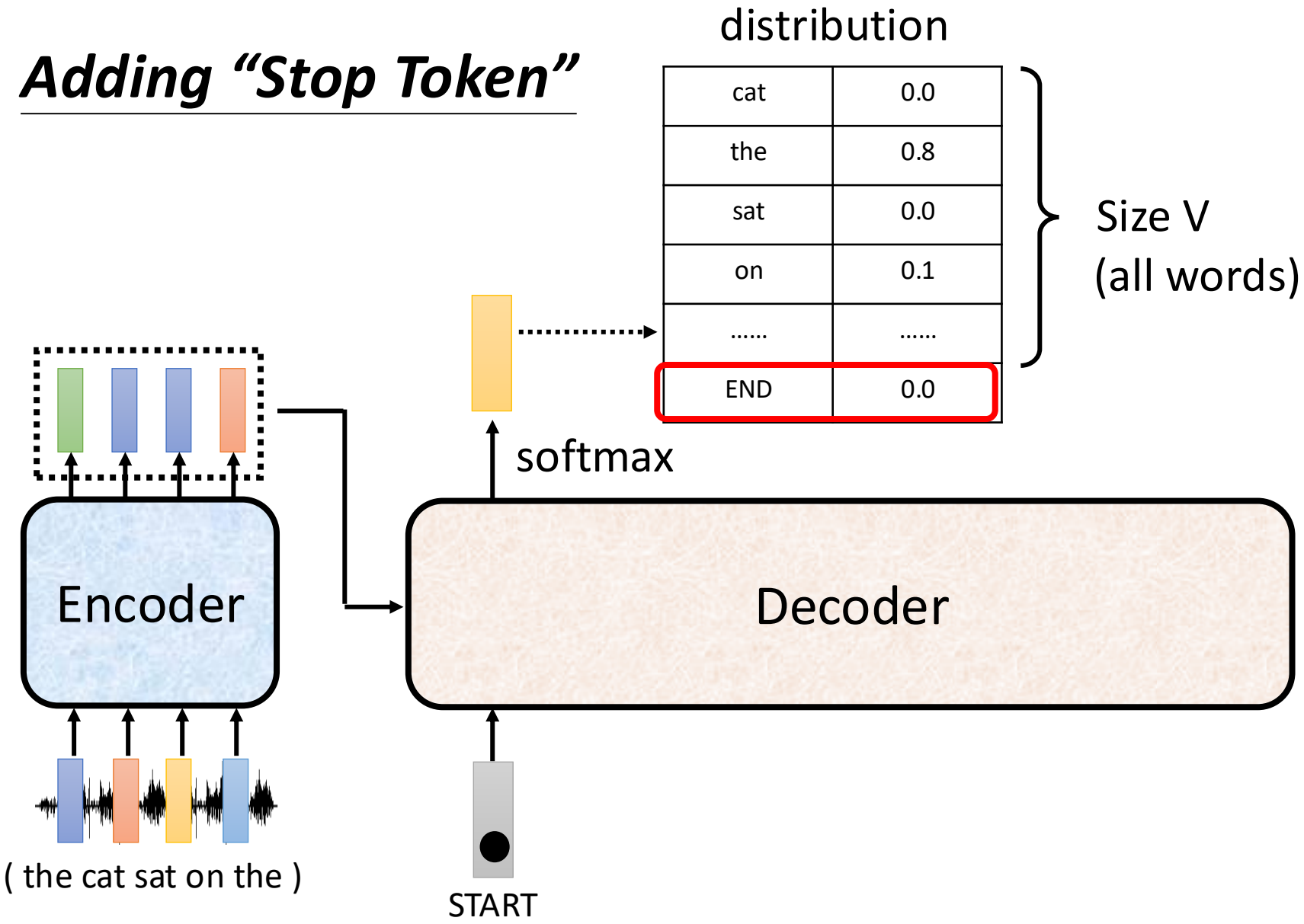
We do not know the correct output length.

Never stop!



credit: Prof. Hung-yi Lee,

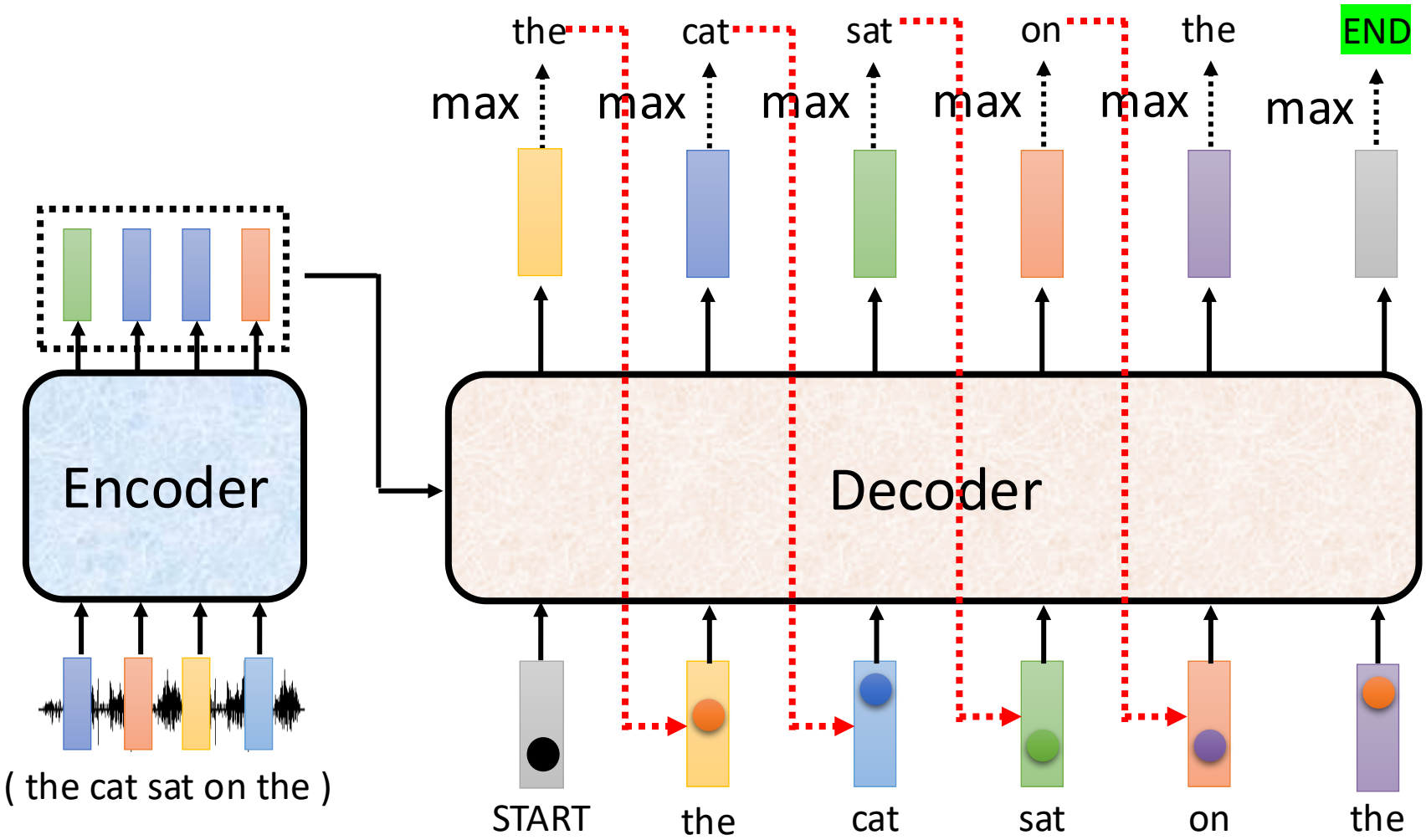
Adding "Stop Token"



credit: Prof. Hung-yi Lee,

Autoregressive

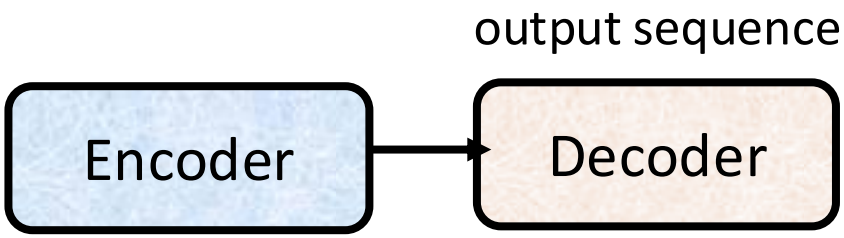
Stop at here!

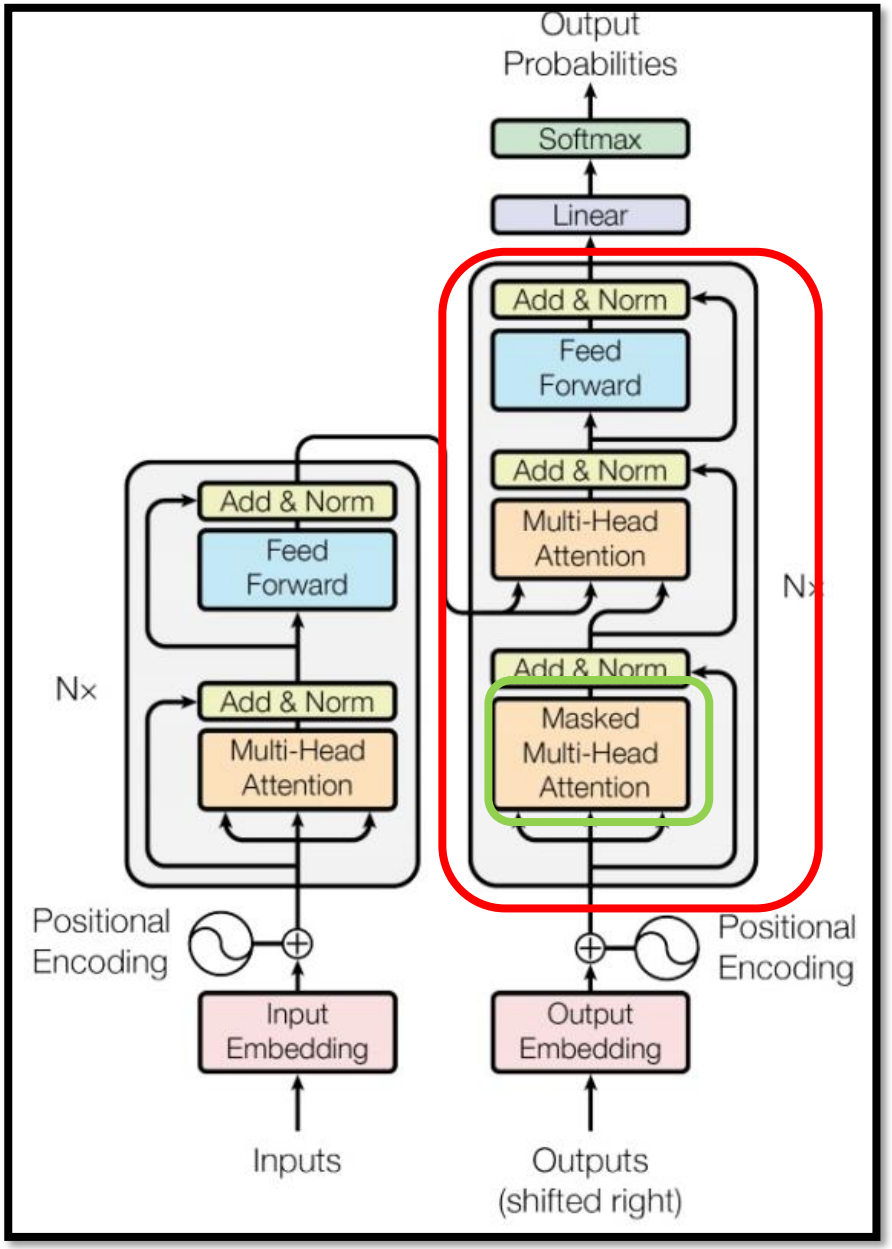


credit: Prof. Hung-yi Lee,

Transformer

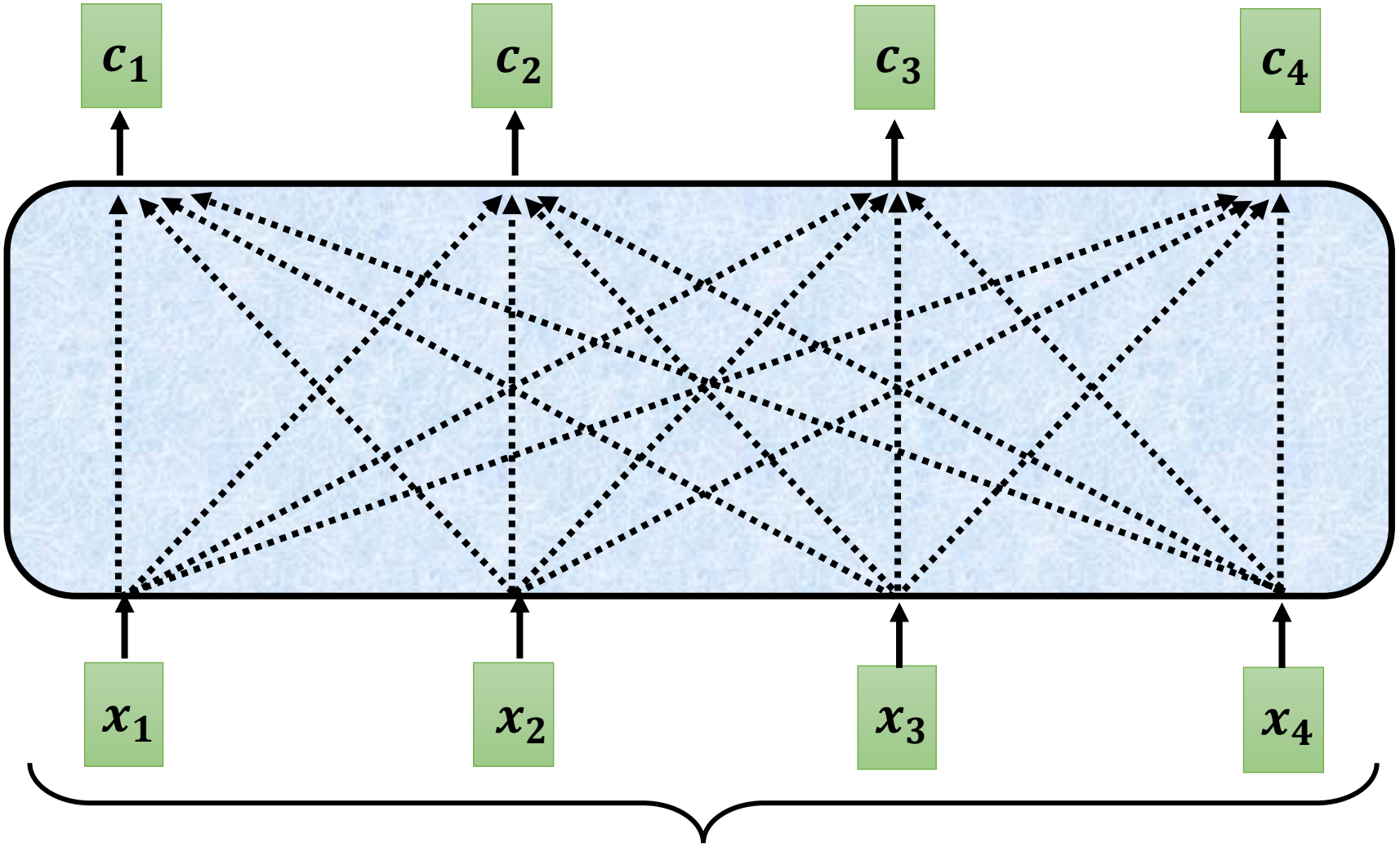
https://arxiv.org/abs/1706.03762





credit: Prof. Hung-yi Lee,

Self-attention

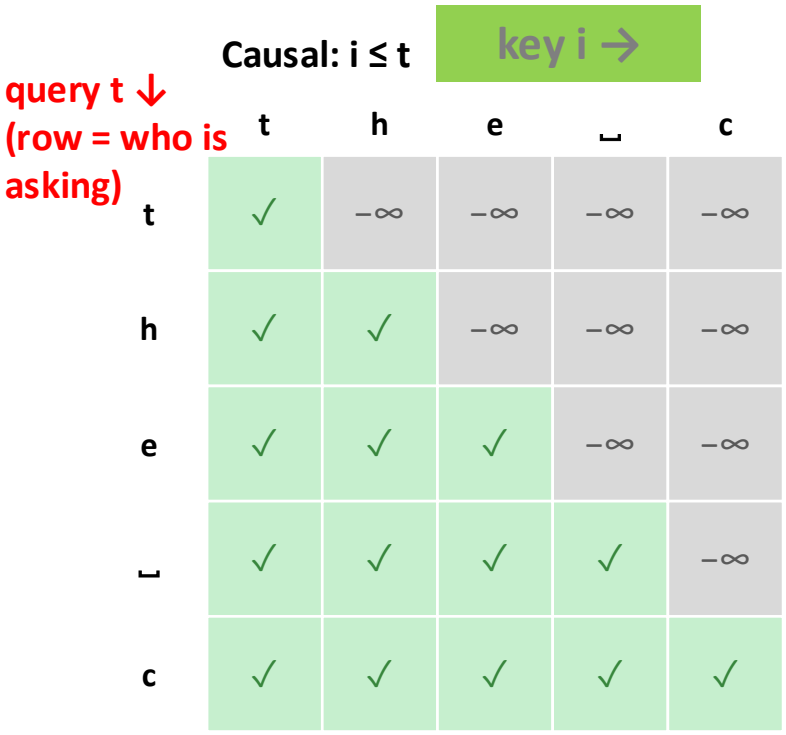


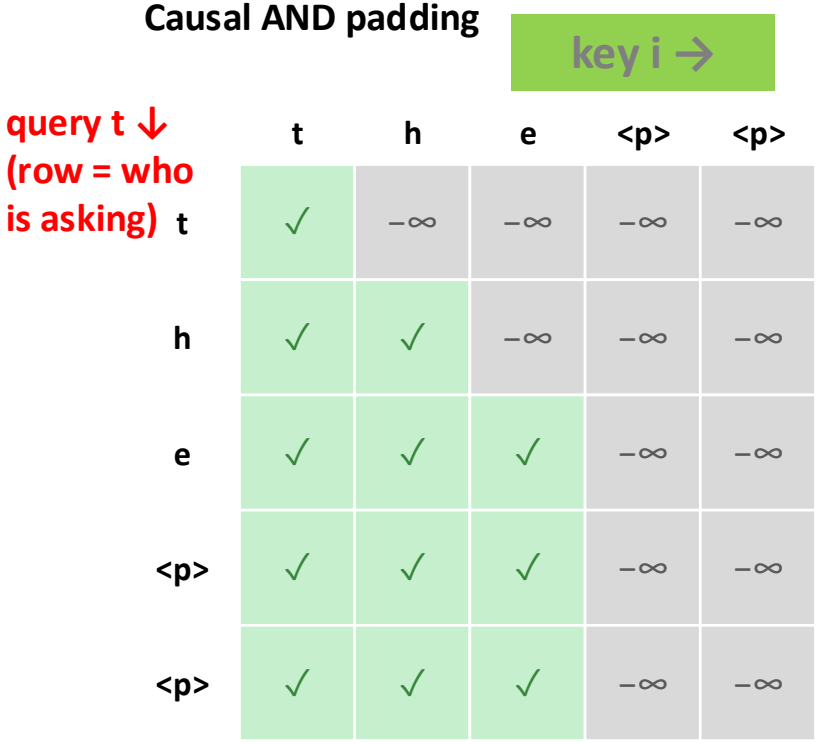
Can be either **input** or **a hidden layer**

credit: Prof. Hung-yi Lee, NTU ML 2021 (notation adapted to the attention lecture)

Self-attention → Masked Self-attention

Masks (2): causal mask,





Query t may only look at keys i ≤ t.
At generation time, position t + 1 does not exist yet.

Code: masked scaled dot-product attention

```
def masked_attention(Q, K, V, allowed):
    # Q, K, V: (B, T, D)      allowed: (B, T, T) bool
    D = Q.shape[-1]
    S = Q @ K.transpose(-2, -1) / math.sqrt(D)     # (B, T, T)
    S = S.masked_fill(~allowed, float('-inf'))    # mask BEFORE softmax
    A = torch.softmax(S, dim=-1)                  # each row sums to 1
    return A @ V, A                               # (B, T, D), weights

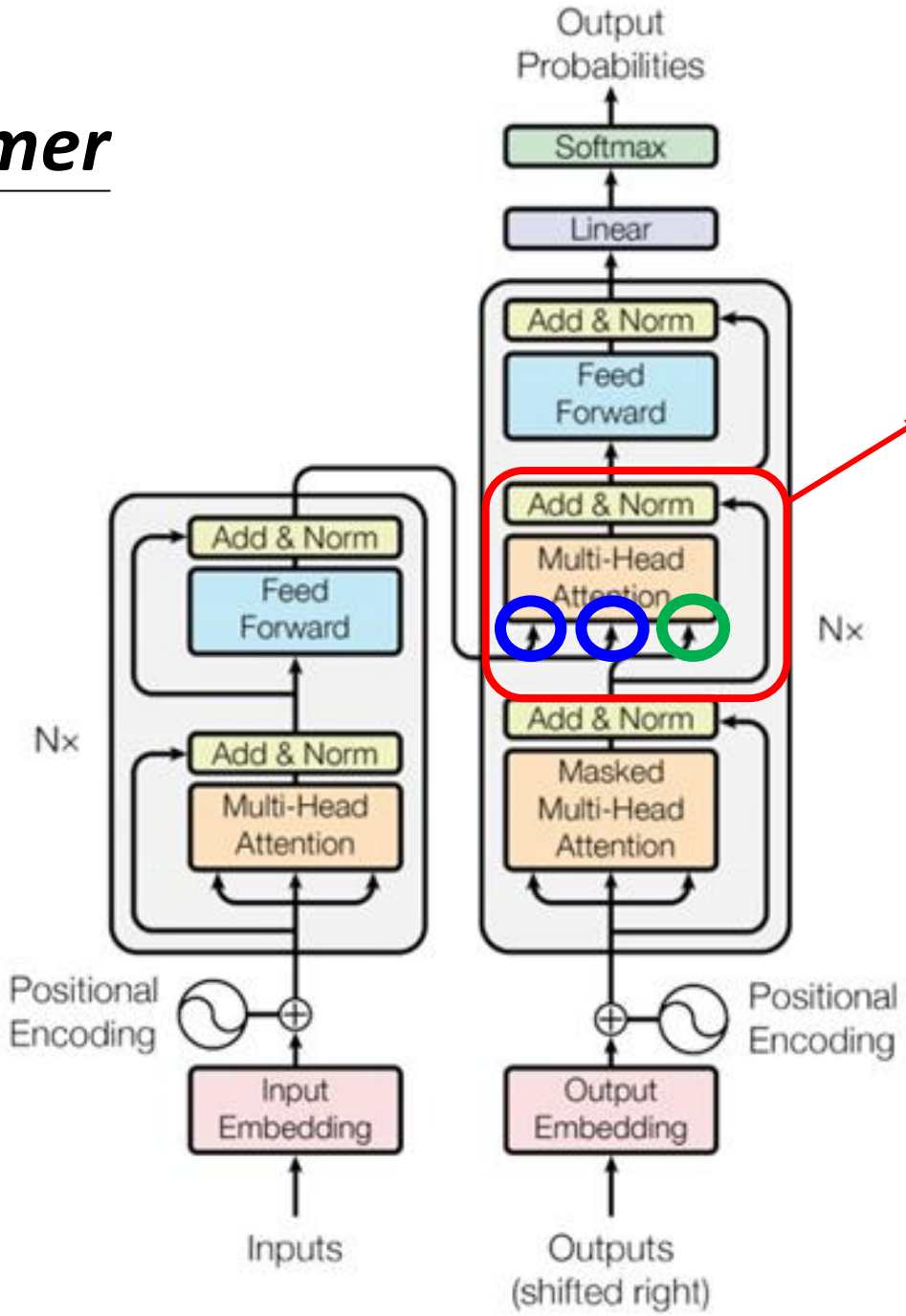
causal = torch.tril(torch.ones(T, T, dtype=torch.bool))
allowed = causal & pad_mask.bool()[:, None, :]    # (B, T, T)
```

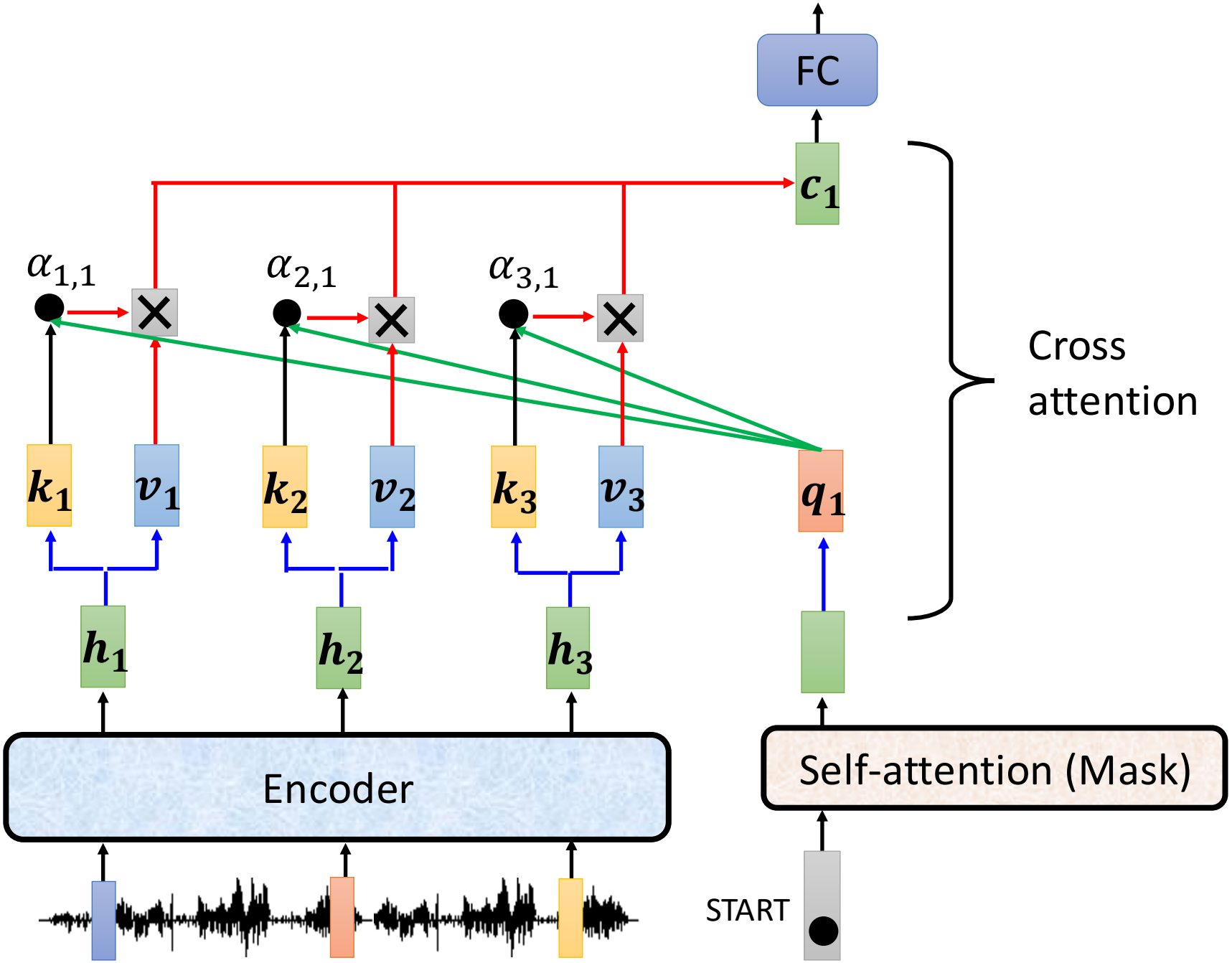
In a model this sits between the token / position embeddings, W_q, W_k, W_v and the output head. Writing it yourself, not nn.MultiheadAttention, shows every step.

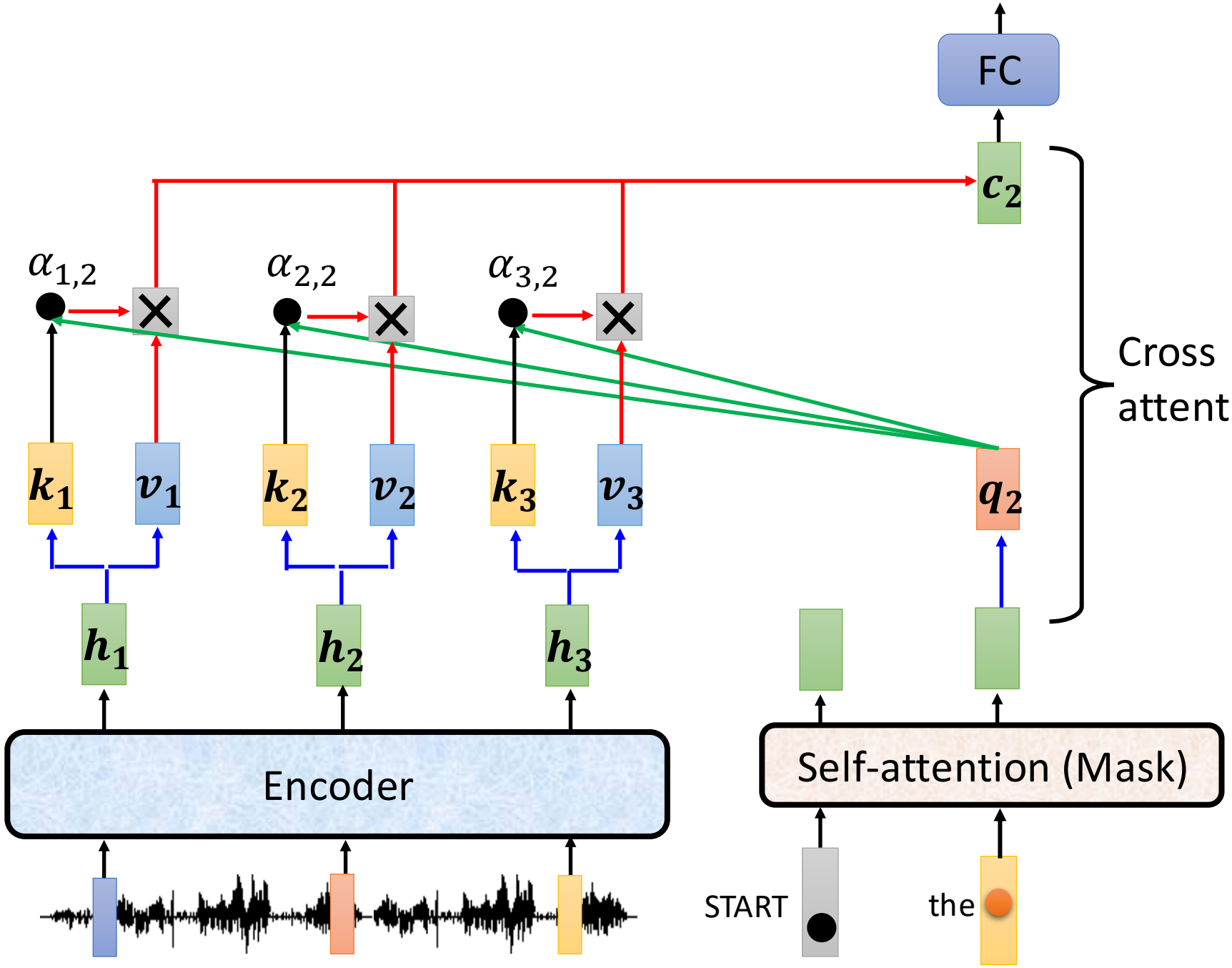
Quiz Examples:

- In a 6 × 6 causal mask, how many entries are $-\infty$?
- Why must the masks be applied before the softmax, not after?
- What are the shapes of Q, K^T, S and C for a batch of 32 sequences, T = 42, D = 64?

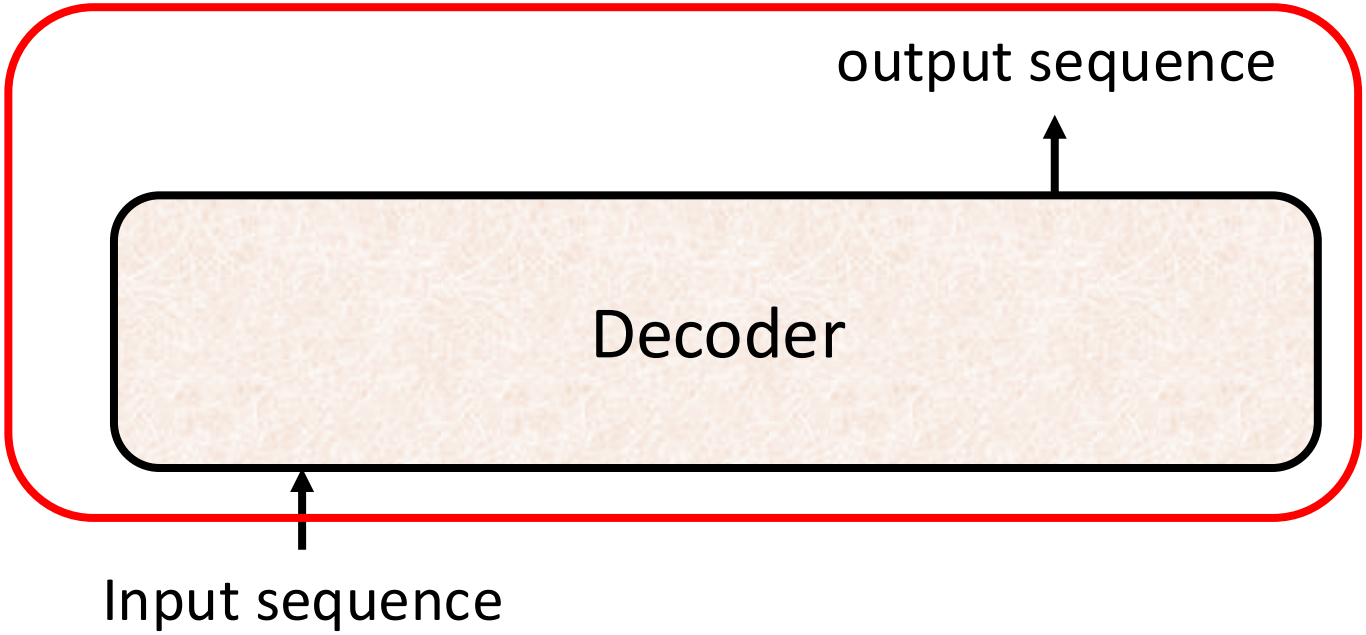
Transformer




FC
c_1
$\alpha_{1,1}$
$\alpha_{2,1}$
$\alpha_{3,1}$
Cross attention
k_1
v_1
k_2
v_2
k_3
v_3
q_1
h_1
h_2
h_3
Encoder
Self-attention (Mask)
START


FC
c_2
$\alpha_{1,2}$
$\alpha_{2,2}$
$\alpha_{3,2}$
Cross attention
k_1
v_1
k_2
v_2
k_3
v_3
q_2
h_1
h_2
h_3
Encoder
Self-attention (Mask)
START
the

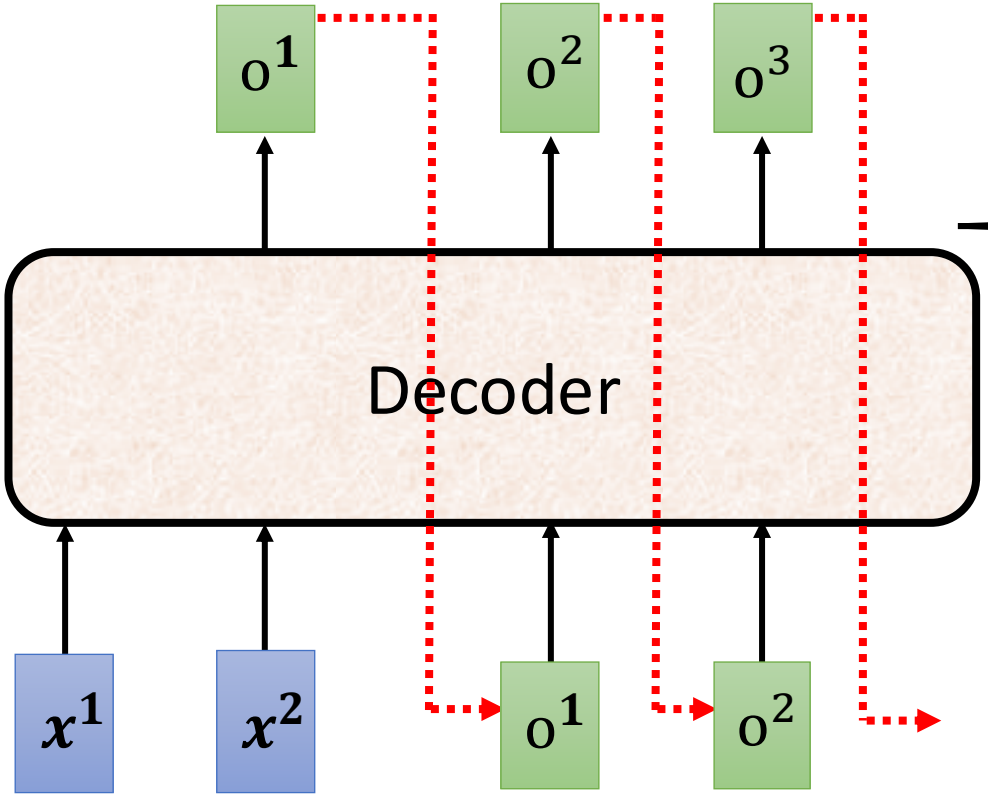
Decoder Only

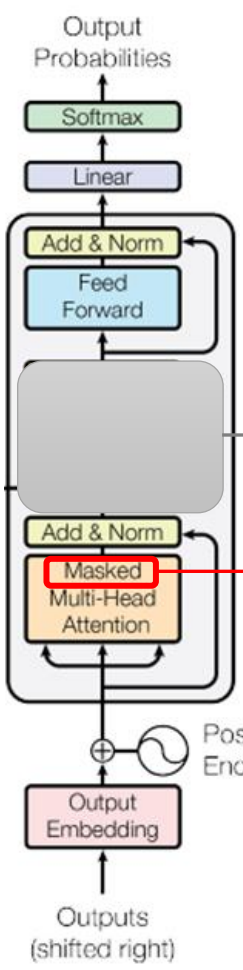


credit: Prof. Hung-yi Lee,

Transformer's Decoder

Decoder-only





no cross-attention (no encoder)

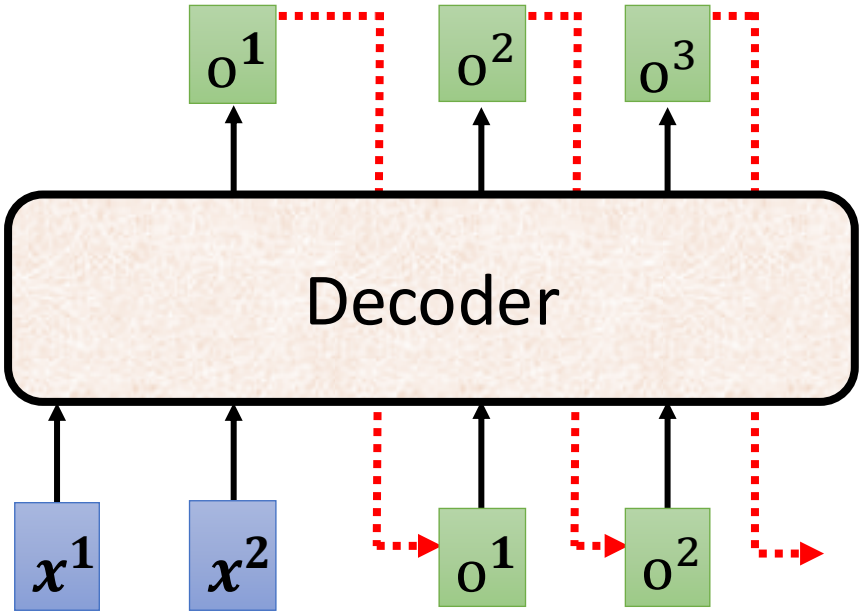
masked self-attention

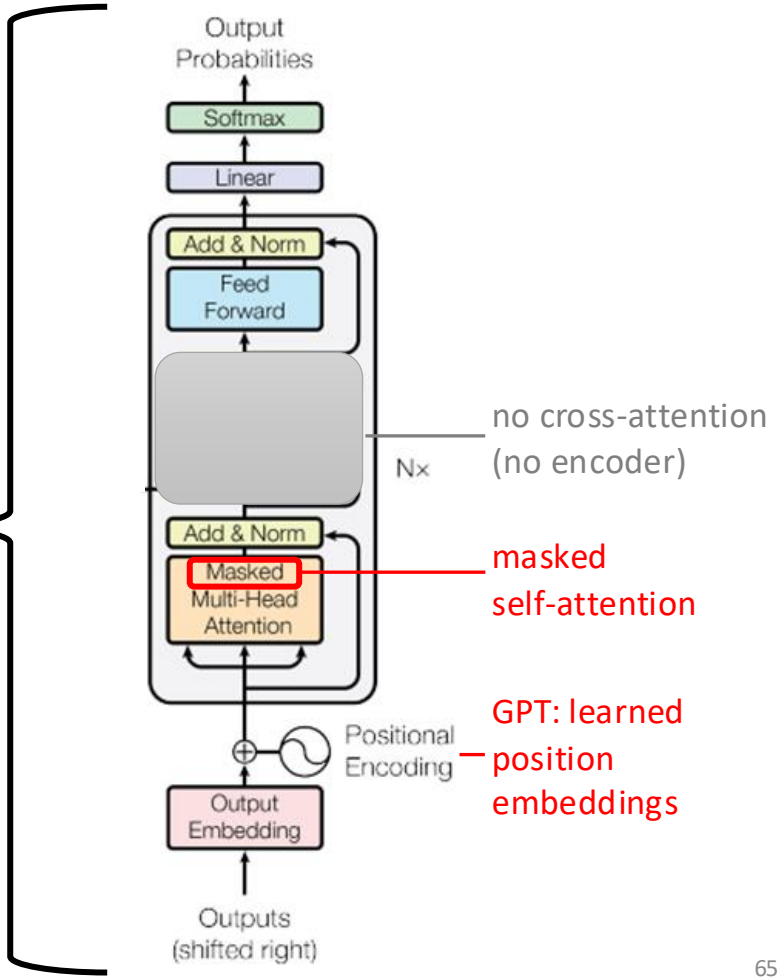
GPT: learned position embeddings

credit: Prof. Hung-yi Lee,

use the **same** network architecture as **transformer decoder**.

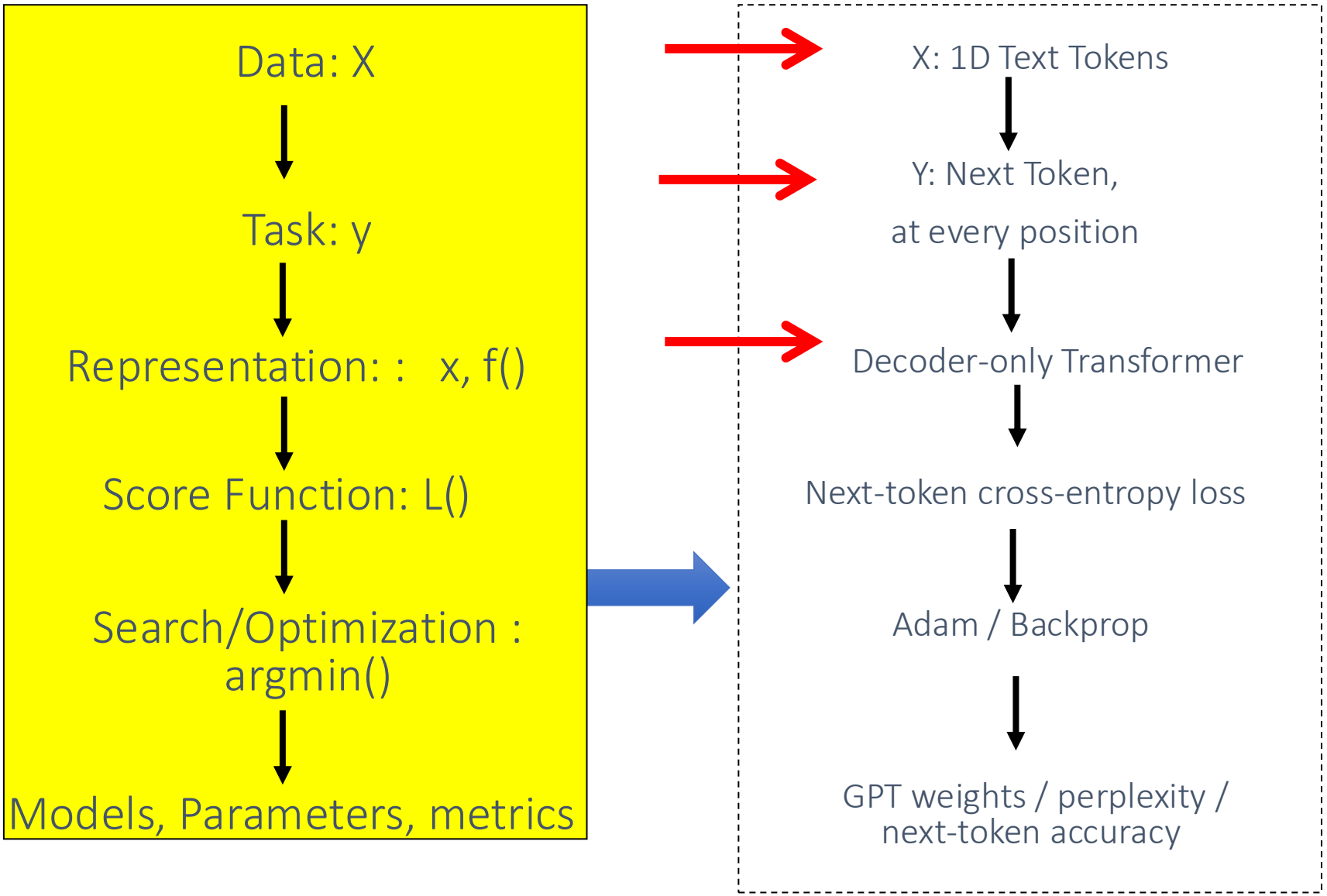
GPT





credit: Prof. Hung-yi Lee,

Today: Decoder-only LLM Architecture and Training



References

- Prof. Hung-yi Lee's ML 2021 slides: Self-attention and Transformer (NTU)
- Vaswani et al., "Attention Is All You Need", NeurIPS 2017
- Radford et al., GPT-1 (2018) and GPT-2 (2019); Brown et al., GPT-3, NeurIPS 2020
- Wei et al., "Emergent Abilities of Large Language Models", TMLR 2022
- Holtzman et al., "The Curious Case of Neural Text Degeneration", ICLR 2020
- Ouyang et al., "Training language models to follow instructions with human feedback", NeurIPS 2022
- S. Raschka, Build a Large Language Model (From Scratch), Manning 2024
- A. Karpathy, nanoGPT, github.com/karpathy/nanoGPT

Three kinds of attention in one Transformer

Attention	Encoder-only (BERT)	Encoder-decoder (Transformer, T5)	Decoder-only (GPT)
Encoder self-attention	padding mask only, no causal mask	padding mask only, no causal mask	—
Decoder self-attention	—	causal + padding mask	causal + padding mask, in every layer
Cross-attention (decoder queries → encoder keys)	—	not causal; uses the encoder's padding mask	—

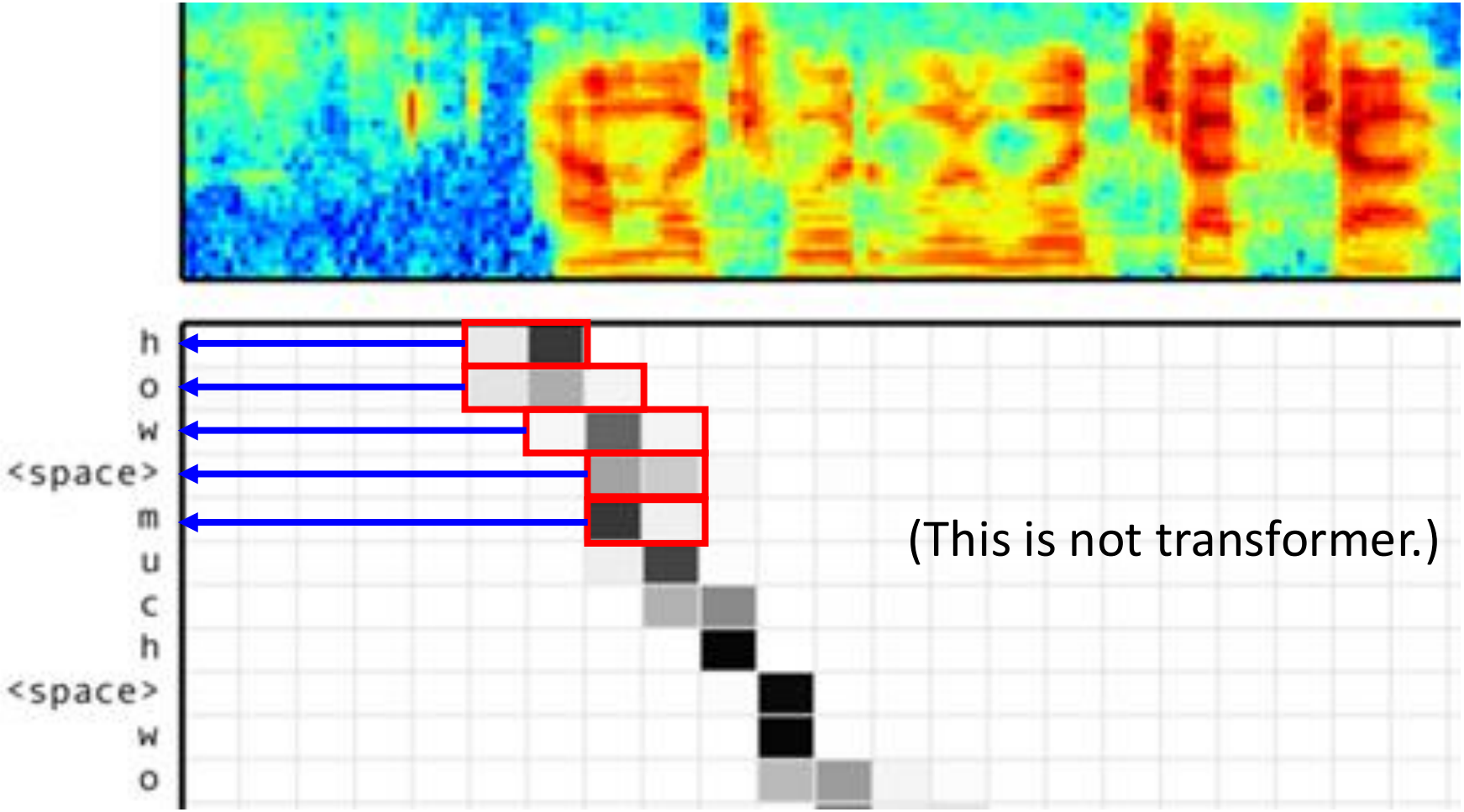
- Same formula each time: only the source of the vectors and the mask change
- Cross-attention is the attention trick from today, without an RNN
- BERT keeps only the first row; GPT keeps only the second, with the causal mask

credit: Vaswani et al. 2017

Cross Attention

Listen, attend and spell: A neural network for large vocabulary conversational speech recognition

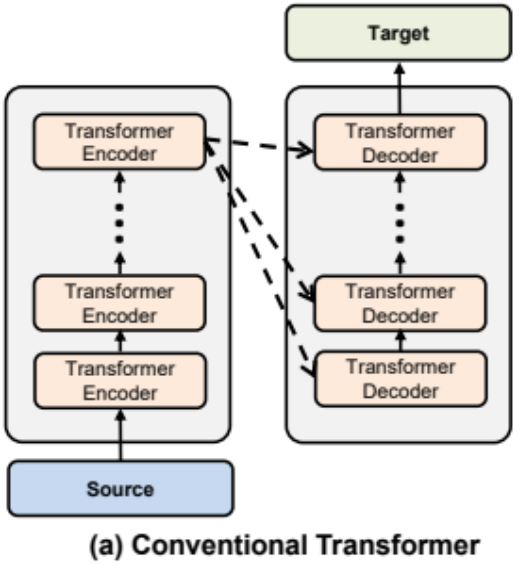
https://ieeexplore.ieee.org/document/7472621



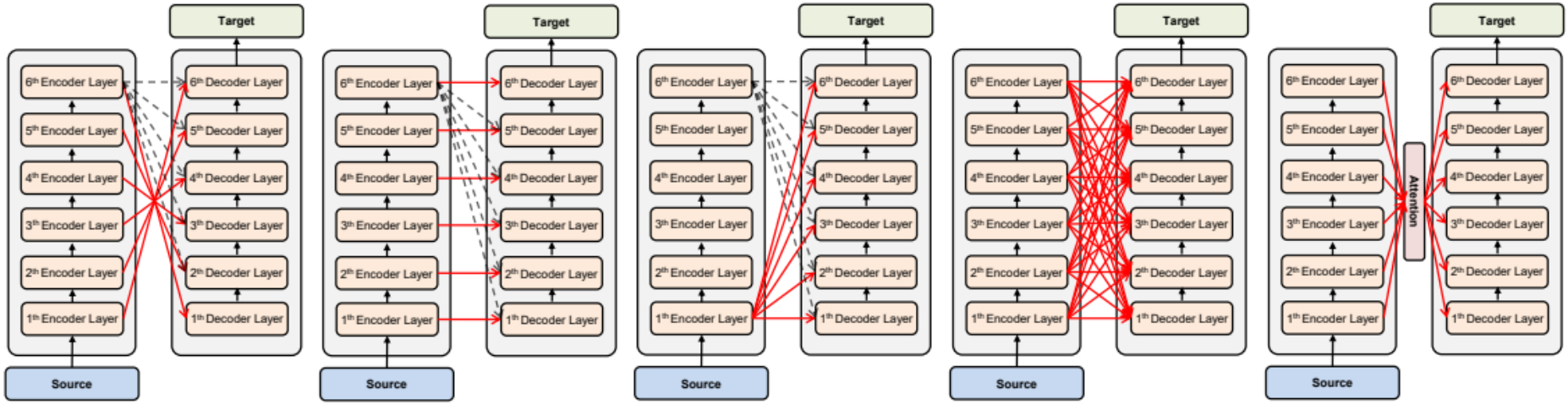
(This is not transformer.)

Cross Attention

Source of image:
https://arxiv.org/abs/2005.08081



(a) Conventional Transformer



(a) Granularity Consistent Attention (b) Granularity Parallel Attention (c) Fine-Grained Attention (d) Full Matching Attention (e) Adaptive Matching Attention