TRANSFORMERS

Lara J. Martin (she/they)
TA: Aydin Ayanzadeh (he)

12/12/2023

CMSC 671

By the end of class today, you will be able to:

1. Identify how a transformer differs from an RNN
2. Interpret how different prompts affect generation
3. Evaluate the ethical considerations of ML algorithms

Modified from slides by Dr. Daphne Ippolito

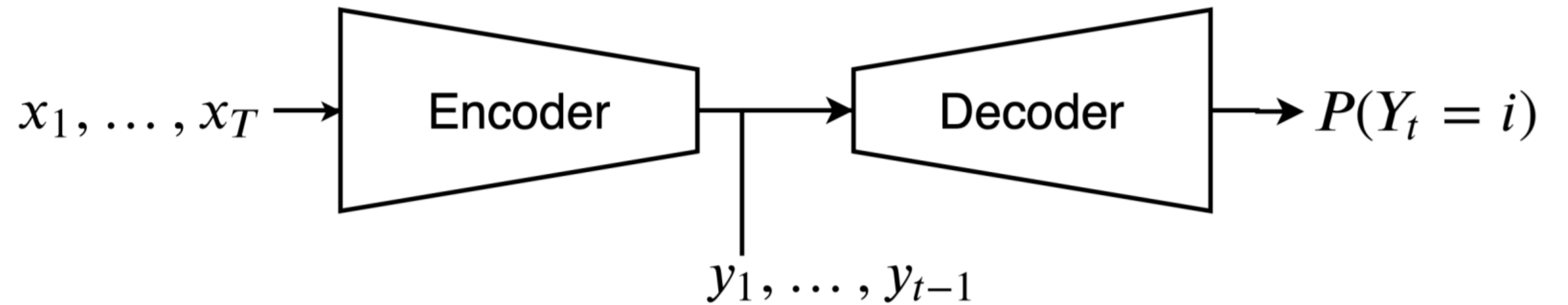
RECAP: MULTI-LAYER NETWORKS: GENERAL STRUCTURE

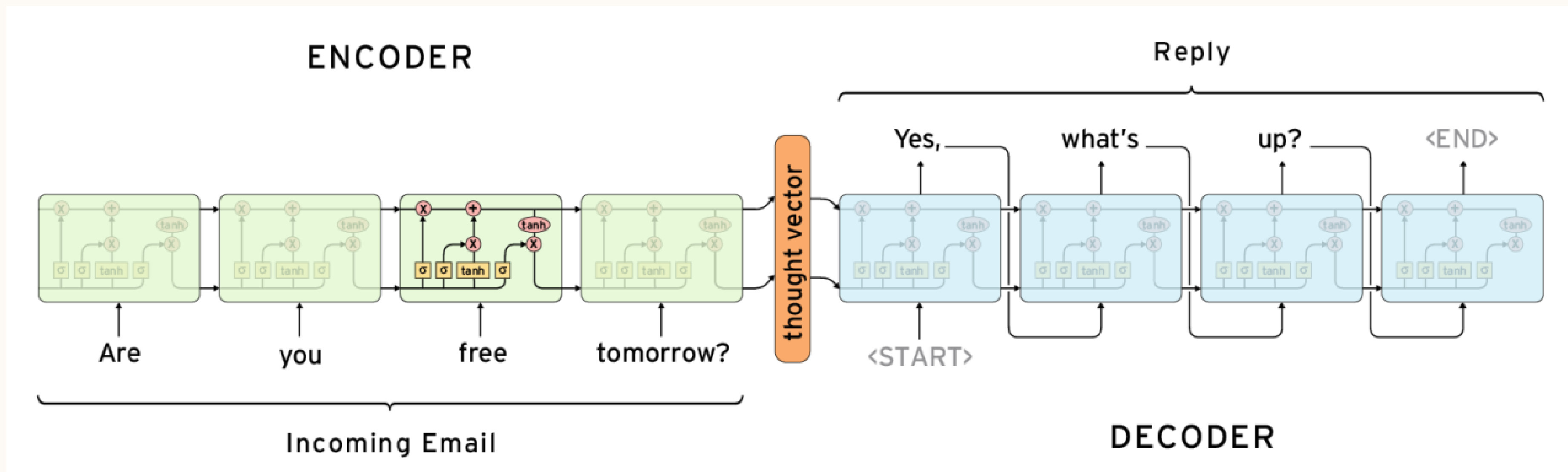
Mutli-layer perceptrons (aka neural networks) will have **inputs**, one or more **hidden layers**, and an **output layer:**

- Number of inputs, outputs, and number and size of hidden layers can vary
- Combination of **different weights** and **different structures** represent different **functions**
- We will treat each layer as **fully-connected**
  - Each unit in one layer connects to every unit in the next layer

ENCODER-DECODER

- Input sequence: $x_1,\ldots,x_T$
- Target sequence: $y_1,\ldots,y_{T'}$





https://www.linkedin.com/pulse/explanation-attention-based-encoder-decoder-deep-keshav-bhandari/

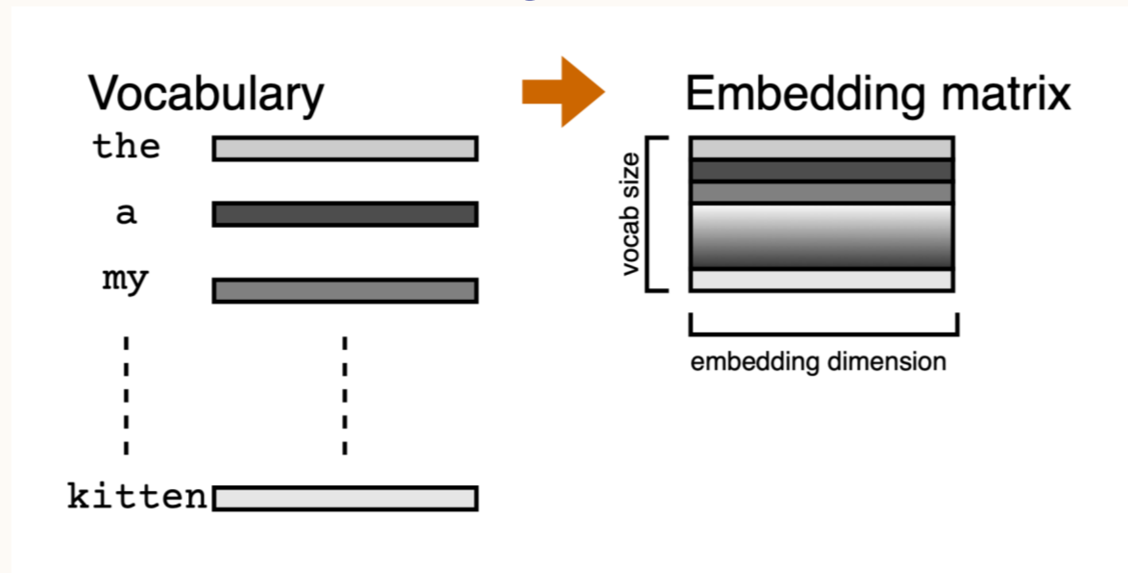
WHAT IS A TOKEN?

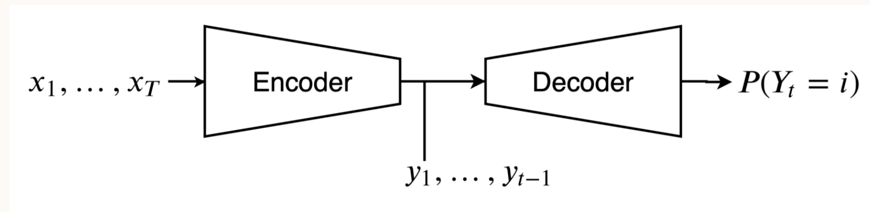
- The first step of building a neural language model is constructing a vocabulary of valid tokens.

Vocab Type	Example
character-level	['A', ' ', 'h', 'i', 'p', 'p', 'o', 'p', 'o', 't', 'a', 'm', 'u', 's',' ', 'a', 't', 'e', ' ', 'm', 'y', ' ', 'h', 'o', 'm', 'e', 'w', 'o', 'r', 'k', '.']
subword-level	['A', 'hip', '##pop', '##ota', '##mus', 'ate', 'my', 'homework', '.']
word-level	['A', 'hippopotamus', 'ate', 'my', 'homework', '.']

WHAT IS A TOKEN?

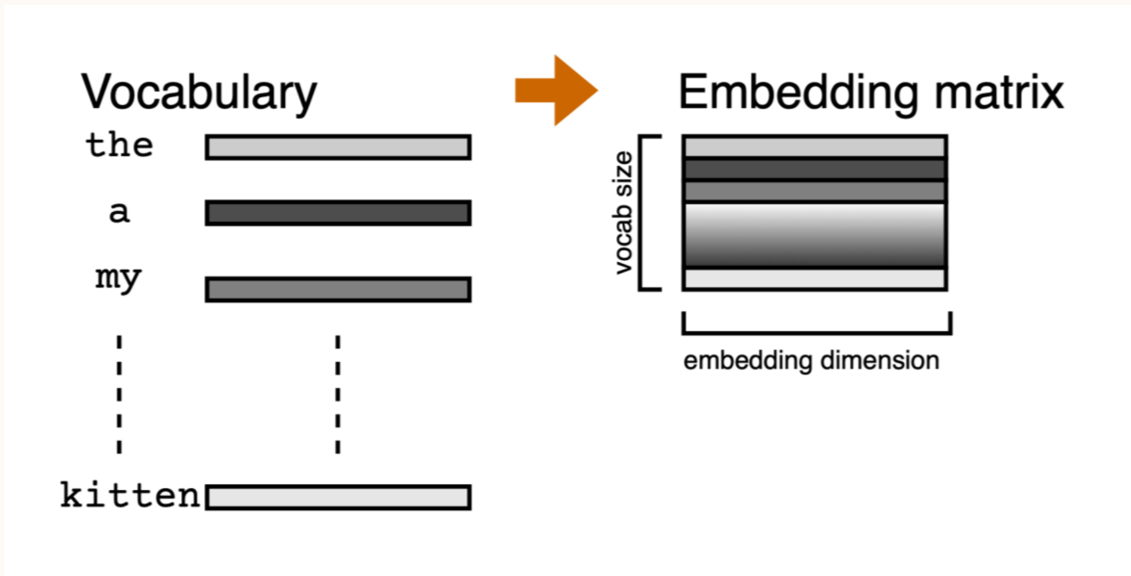
- The first step of building a neural language model is constructing a vocabulary of valid tokens.
- Each token in the vocabulary is associated with a vector embedding, and these are concatenated into an embedding matrix.

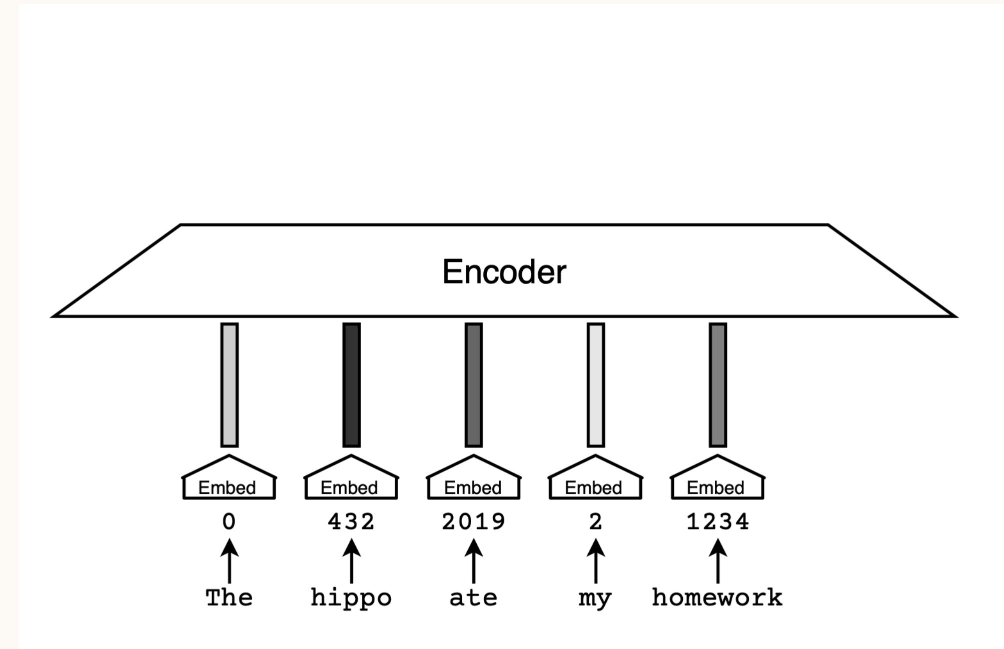


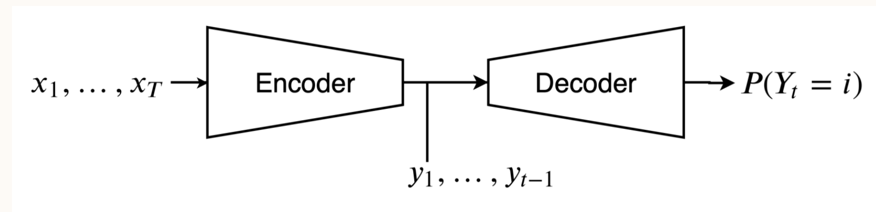


INPUTS TO THE ENCODER

- The encoder takes as input the embeddings corresponding to each token in the sequence.

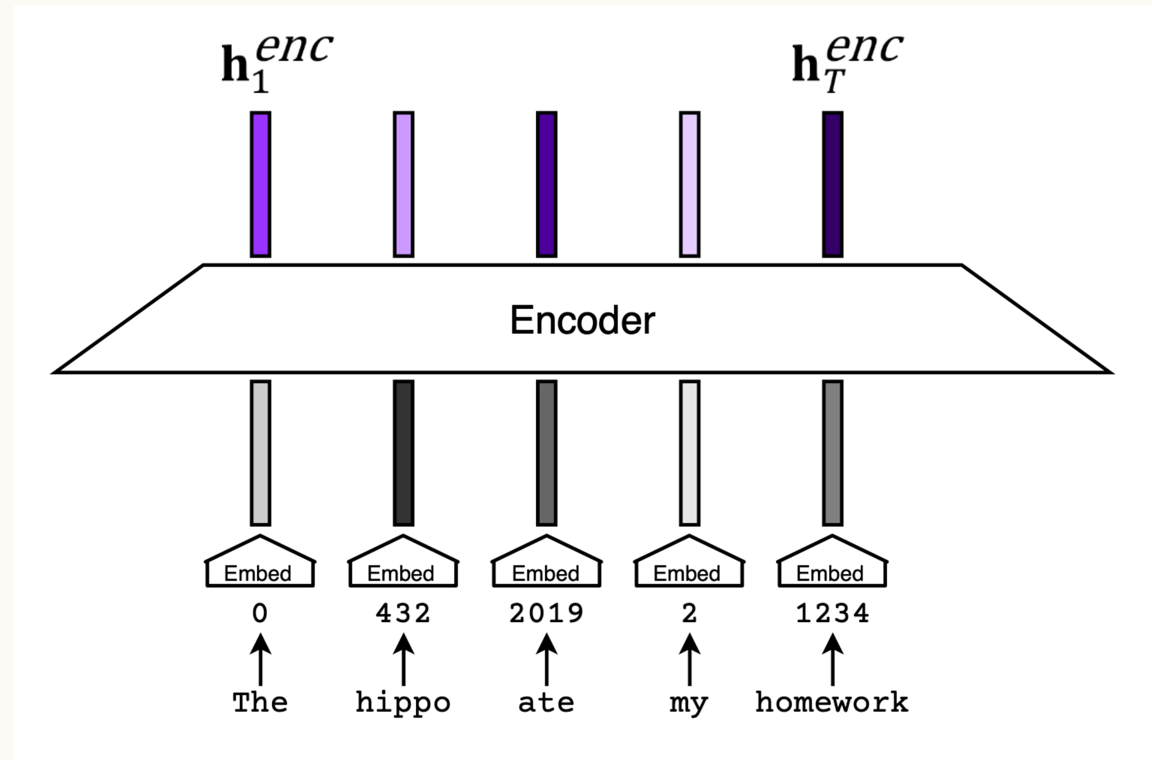


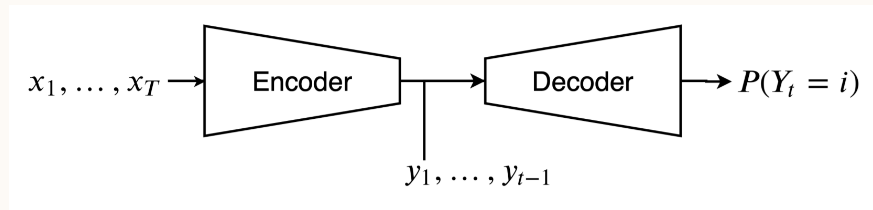




OUTPUTS FROM THE ENCODER

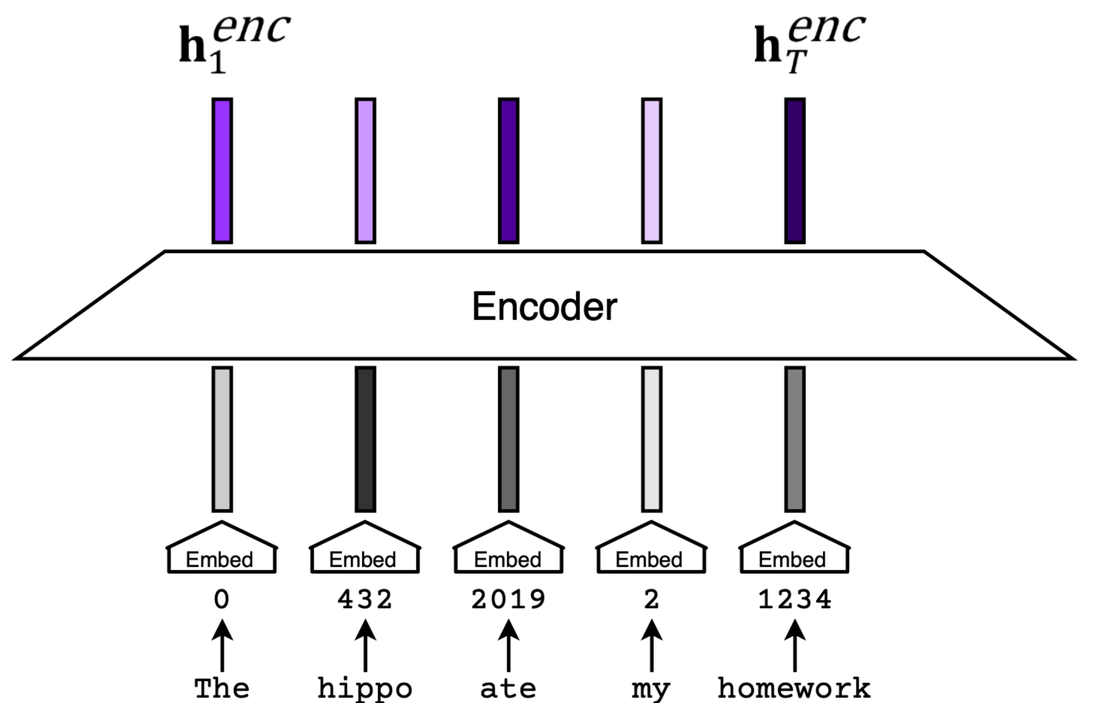
The encoder outputs a sequence of vectors. These are called the hidden state of the encoder.

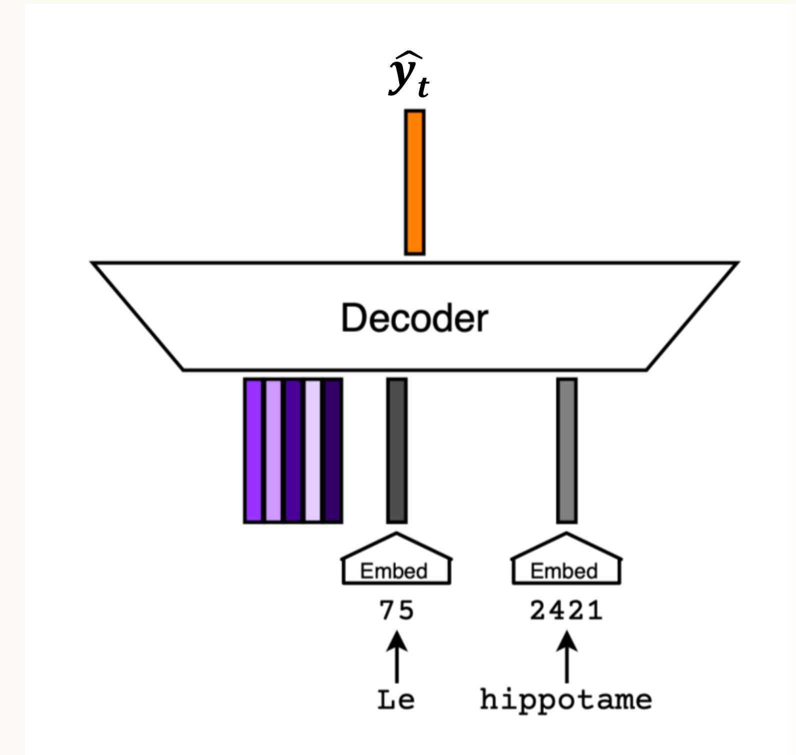


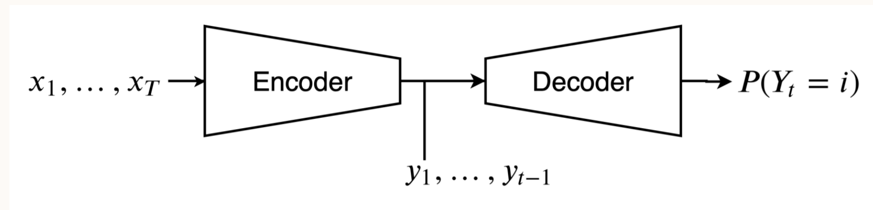


INPUTS TO THE DECODER

The decoder takes as input the hidden states from the encoder as well as the embeddings for the tokens seen so far in the target sequence.

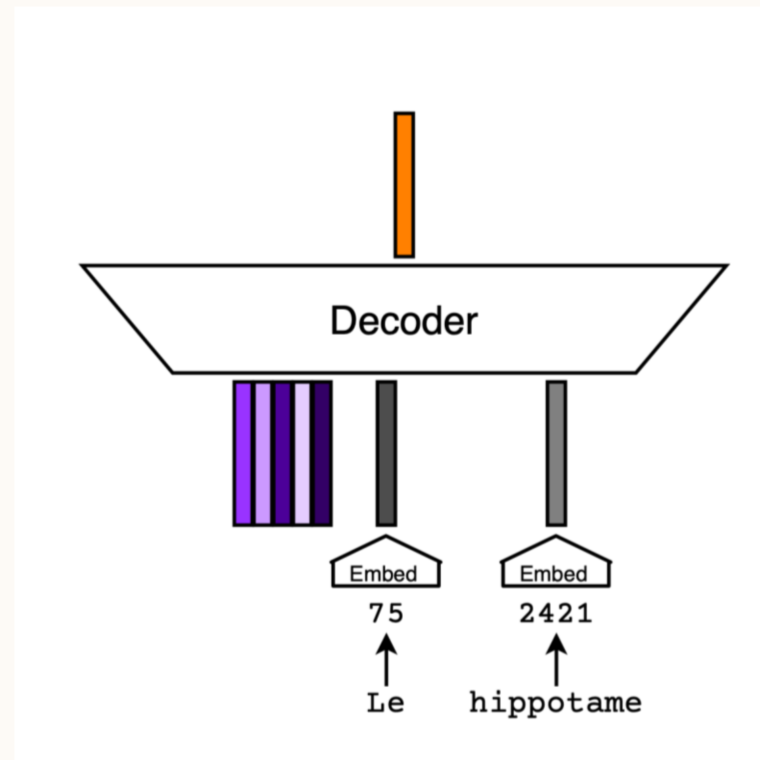






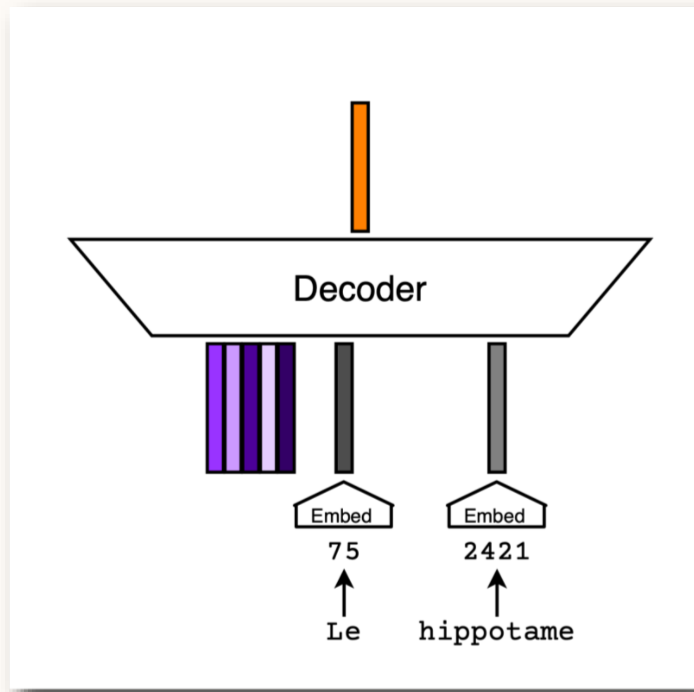
OUTPUTS FROM THE DECODER

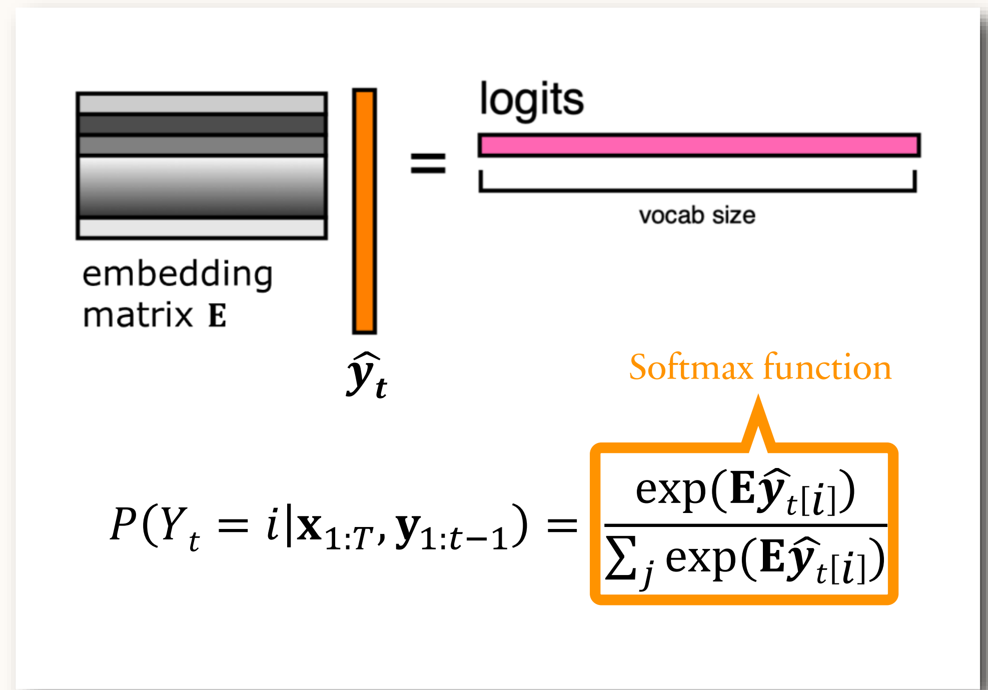
The decoder outputs an embedding $\hat{\boldsymbol{y}}_t$. The goal is for this embedding to be as close as possible to the embedding of the true next token.



TURNING $\hat{y}_t$ INTO A PROBABILITY DISTRIBUTION

- We can multiply the predicted embedding $\hat{\boldsymbol{y}}_t$ by our vocabulary embedding matric to get a score for each vocabulary word. These scores are referred to as logits.
- The softmax function then lets us turn the logits into probabilities.





LOSS FUNCTION

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

The index of the true tth word in the target sequence.

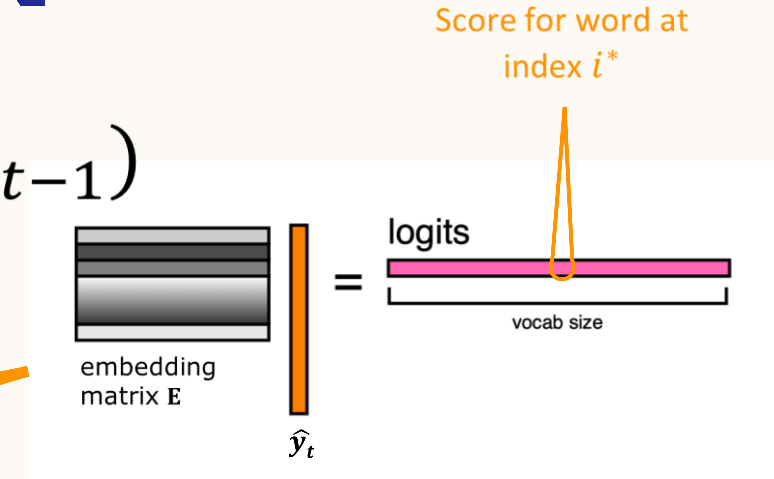
LOSS FUNCTION

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

The probability the language model assigns to the true tth word in the target sequence.

LOSS FUNCTION

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$



$$= -\sum_{t=1}^{T} \log \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i^*])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\boldsymbol{y}}_{t[i]})}{\sum_j \exp(\mathbf{E}\hat{\boldsymbol{y}}_{t[i]})}$$

LOSS FUNCTION

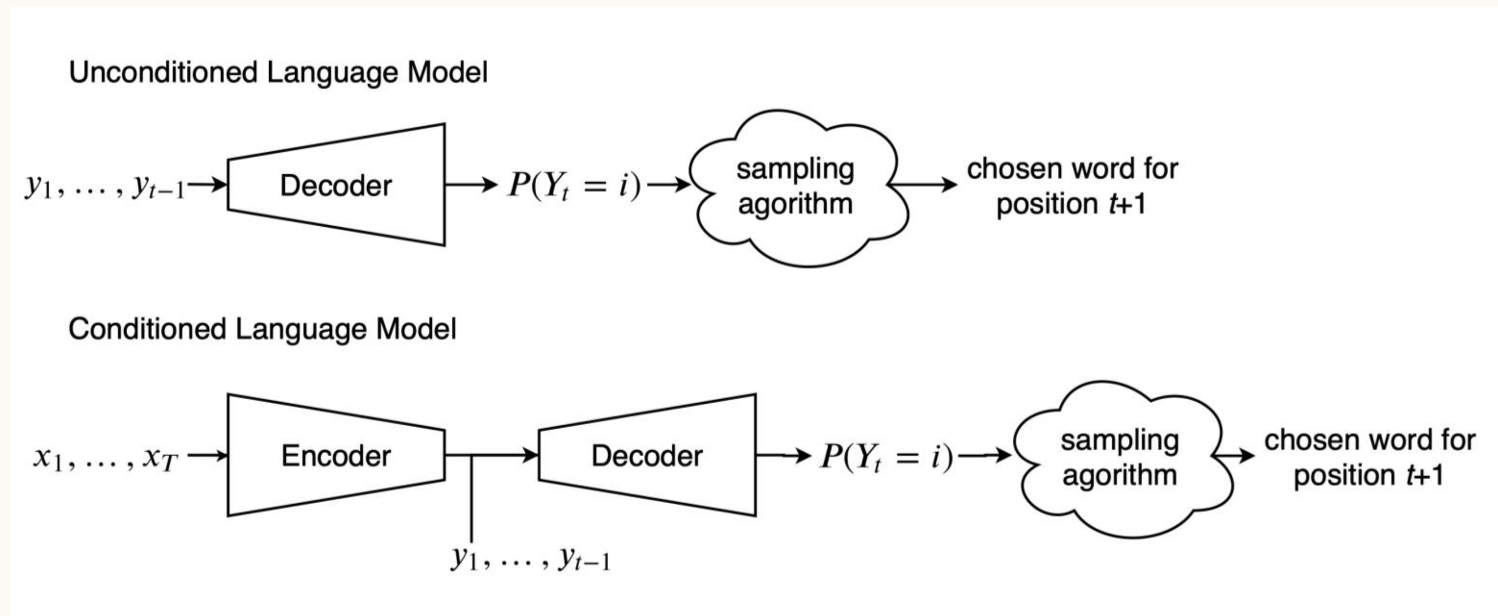
$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= -\sum_{t=1}^{T} \cancel{\log} \frac{\cancel{\exp}(\mathbf{E}\hat{\mathbf{y}}_t[i^*])}{\cancel{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}}$$

$$= -\sum_{t=1}^{T} \mathbf{E}\hat{\mathbf{y}}_t[i^*]$$

GENERATING TEXT AT INFERENCE TIME

- To generate text, we need an algorithm that selects tokens given the predicted probability distributions.



Examples:

- Argmax
- Random sampling
- Beam search

RECURRENT NEURAL NETWORKS

- Up until 2017 or so, neural language models were mostly built using recurrent neural networks.

v:1409.3215v3 [cs.CL] 14 Dec 2014

Sequence to Sequence Learning with Neural Networks

Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

Abstract

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT'14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous best result on this task. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source

Generating Sequences With Recurrent Neural Networks

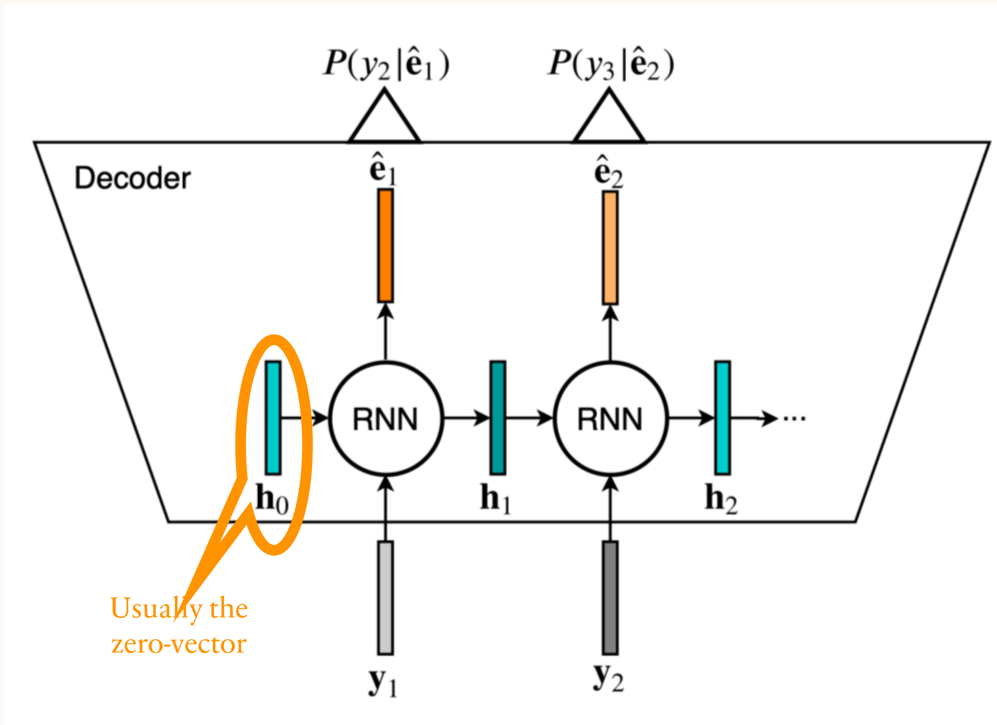
Alex Graves
Department of Computer Science
University of Toronto
graves@cs.toronto.edu

Abstract

This paper shows how Long Short-term Memory recurrent neural networks can be used to generate complex sequences with long-range structure, simply by predicting one data point at a time. The approach is demonstrated for text (where the data are discrete) and online handwriting (where the data are real-valued). It is then extended to handwriting synthesis by allowing the network to condition its predictions on a text sequence. The resulting system is able to generate highly realistic cursive

RECURRENT NEURAL NETWORKS

SINGLE LAYER DECODER ARCHITECTURE



- The current hidden state is computed as a function of the previous hidden state and the embedding of the current word in the target sequence.

$$\mathbf{h}_t = \text{RNN}(\mathbf{W}_{ih}\mathbf{y}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

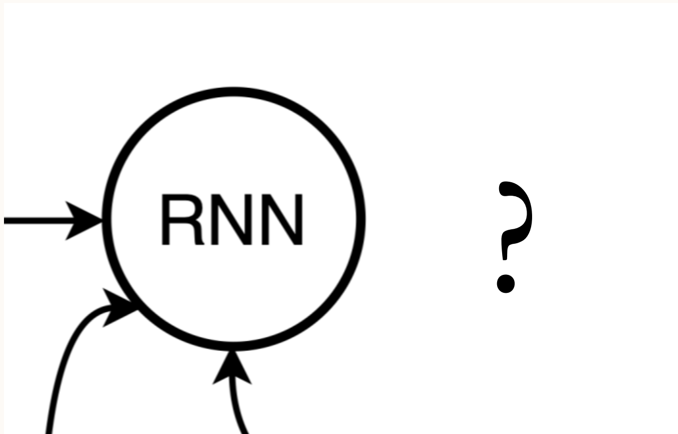
- The current hidden state is used to predict an embedding for the next word in the target sequence.

$$\hat{\mathbf{e}}_t = \mathbf{b}_e + \mathbf{W}_{he}\mathbf{h}_t$$

This predicted embedding is used in the loss function:

$\triangle$ = softmax (E $\hat{\mathbf{e}}_t$) = probabilities (vocab size)

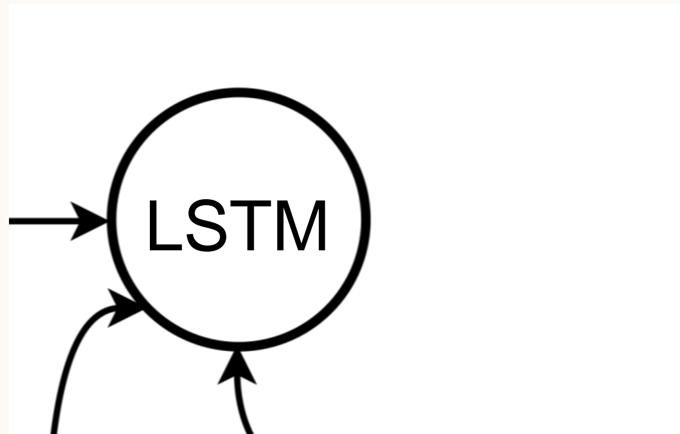
WHAT IS THE “RNN” UNIT?

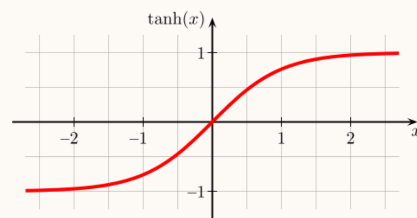


WHAT IS THE “RNN” UNIT?

- LSTM stands for long short-term memory.
- An LSTM uses a gating concept to control how much each position in the hidden state vector can be updated at each step.

LSTMs were originally designed as a mean to keep around information for longer in the hidden state as it gets repeatedly updated.

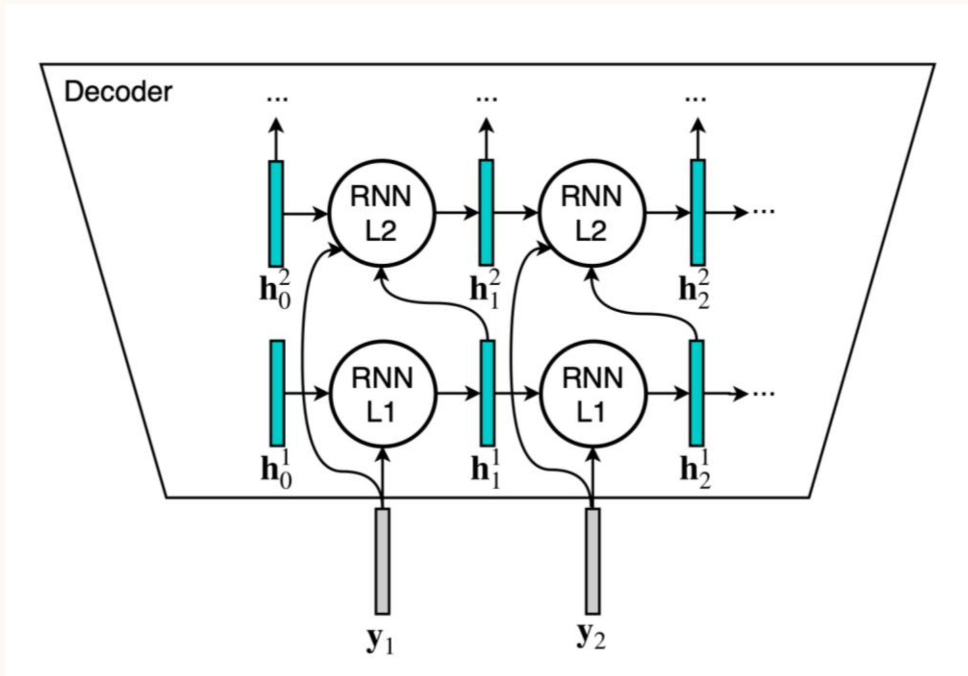




$$
\begin{aligned}
&\text{input gate} & \mathbf{i}_t &= \sigma\left(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\right) \\
&\text{forget gate} & \mathbf{f}_t &= \sigma\left(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f\right) \\
&\text{cell state} & \mathbf{c}_t &= \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh\left(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c\right) \\
&\text{output gate} & \mathbf{o}_t &= \sigma\left(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o\right) \\
&\text{hidden state} & \mathbf{h}_t &= \mathbf{o}_t \tanh(\mathbf{c}_t)
\end{aligned}
$$

https://en.m.wikipedia.org/wiki/File:Hyperbolic_Tangent.svg

RNN MULTI-LAYER DECODER ARCHITECTURE



- Computing the next hidden state:
- For the first layer:

$$\mathbf{h}_t^1 = \text{RNN}(\mathbf{W}_{ih^1}\mathbf{y}_t + \mathbf{W}_{h^1h^1}\mathbf{h}_{t-1}^1 + \mathbf{b}_h^1)$$

- For subsequent layers:

$$\mathbf{h}_t^l = \text{RNN}(\mathbf{W}_{ih^l}\mathbf{y}_t + \mathbf{W}_{h^{l-1}h^l}\mathbf{h}_t^{l-1} + \mathbf{W}_{h^lh^l}\mathbf{h}_{t-1}^l + \mathbf{b}_h^l)$$

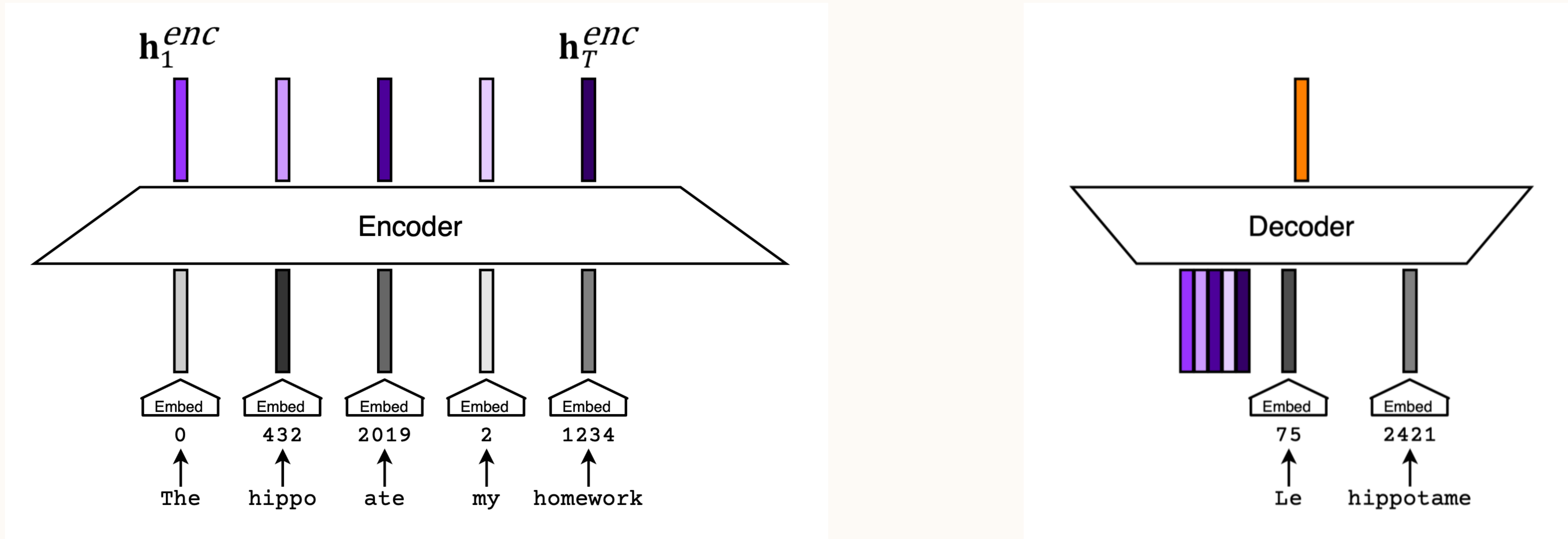
Predicting an embedding for the next token in the sequence:

$$\hat{\mathbf{e}}_t = \mathbf{b}_e + \sum_{l=1}^{L} \mathbf{W}_{h^le}\mathbf{h}_t^l$$

Each of the **b** and **W** are learned bias and weight matrices.

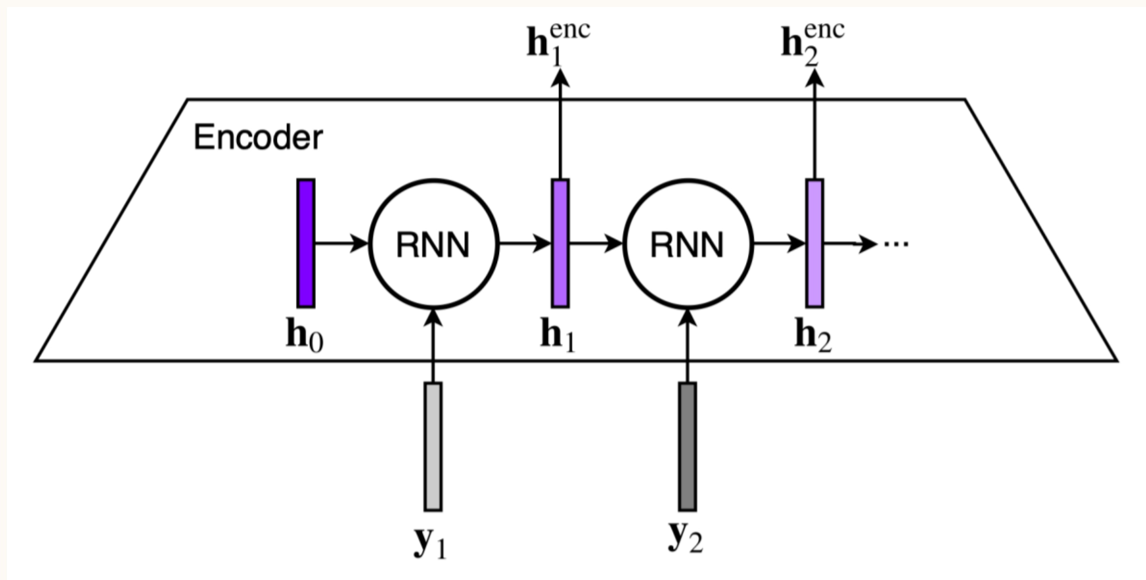
RNN ENCODER-DECODER ARCHITECTURES

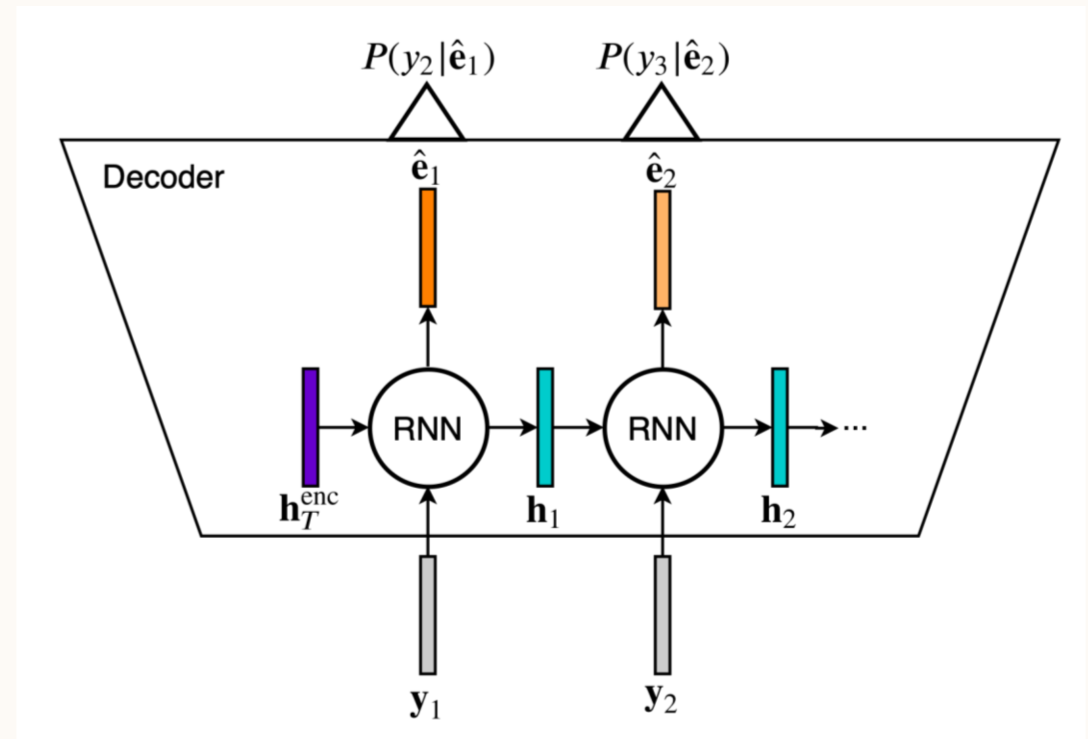
How do we implement an encoder-decoder model?



RNN ENCODER-DECODER ARCHITECTURES

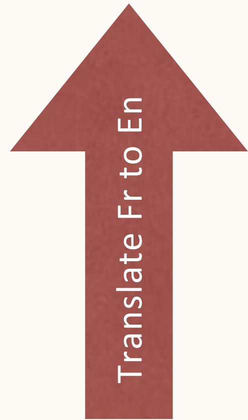
Simplest approach: Use the final hidden state from the encoder to initialize the first hidden state of the decoder.





RNN ENCODER-DECODER ARCHITECTURES

Better approach: an attention mechanism



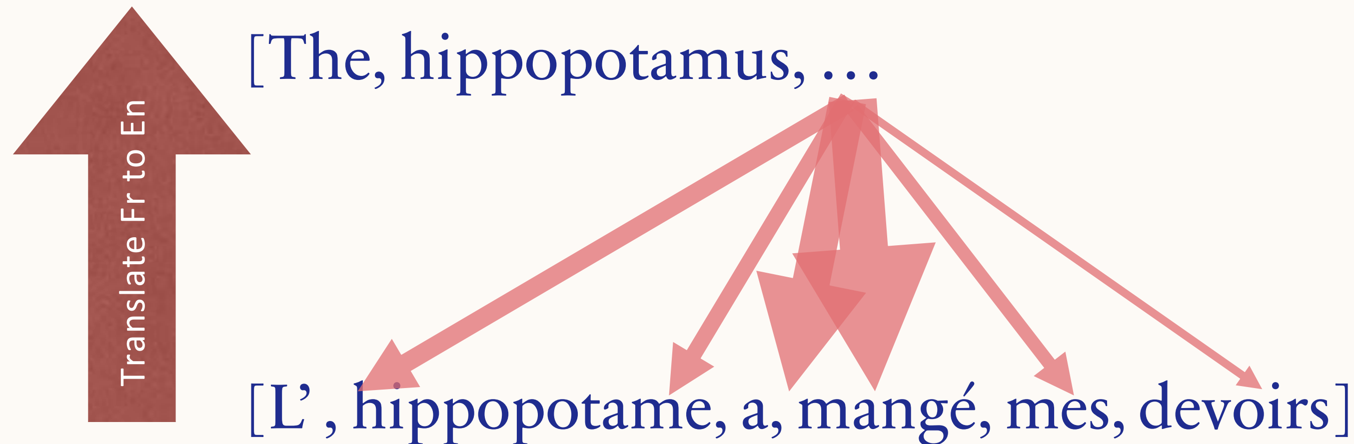
[The, hippopotamus, …

[L’ , hippopotame, a, mangé, mes, devoirs]

When predicting the next English word, how much weight should the model put on each French word in the source sequence?

RNN ENCODER-DECODER ARCHITECTURES

Better approach: an attention mechanism



Compute a linear combination of the encoder hidden states.

$$\mathbf{c}_t = \alpha_1 \; + \alpha_2 \; + \alpha_3 \; + \ldots + \alpha_T$$

Decoder's prediction at position t is based on both the context vector and the hidden state outputted by the RNN at that position.

$$\hat{\mathbf{e}}_t = f_\theta(\mathbf{h}_t^{\text{dec}} \; \mathbf{c}_t)$$

RNN ENCODER-DECODER ARCHITECTURES

- The tth context vector is computed as $\mathbf{c}_t = \mathbf{H}^{\text{enc}} a_t$
- $a_t[i] = \text{softmax}(\text{att_score}(\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}))$

- There are a few different options for the attention score:

$$\text{att_score}(\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}) = \begin{cases} \mathbf{h}_t^{\text{dec}} \cdot \mathbf{h}_i^{\text{enc}} & \text{dot product} \\ \mathbf{h}_t^{\text{dec}}\, \mathbf{W}_a\, \mathbf{h}_i^{\text{enc}} & \text{bilinear function} \\ w_{a_1}^{\top} \tanh(\mathbf{W}_{a2}[\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}]) & \text{MLP} \end{cases}$$

Compute a linear combination of the encoder hidden states.

$\mathbf{c}_t = \alpha_1 + \alpha_2 + \alpha_3 + \ldots + \alpha_T$

Decoder's prediction at position t is based on both the context vector and the hidden state outputted by the RNN at that position.

$\hat{\mathbf{e}}_t = f_\theta(\mathbf{h}_t^{\text{dec}}\ \mathbf{c}_t)$

$\mathbf{H}^{\text{enc}} =$

LIMITATIONS OF RECURRENT ARCHITECTURE

- Slow to train.
  - Can't be easily parallelized.
  - The computation at position t is dependent on first doing the computation at position t-1.
- Difficult to access information from many steps back.
  - If two tokens are K positions apart, there are K opportunities for knowledge of the first token to be erased from the hidden state before a prediction is made at the position of the second token.

TRANSFORMERS

Since 2018, the field has rapidly standardized on the Transformer architecture

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

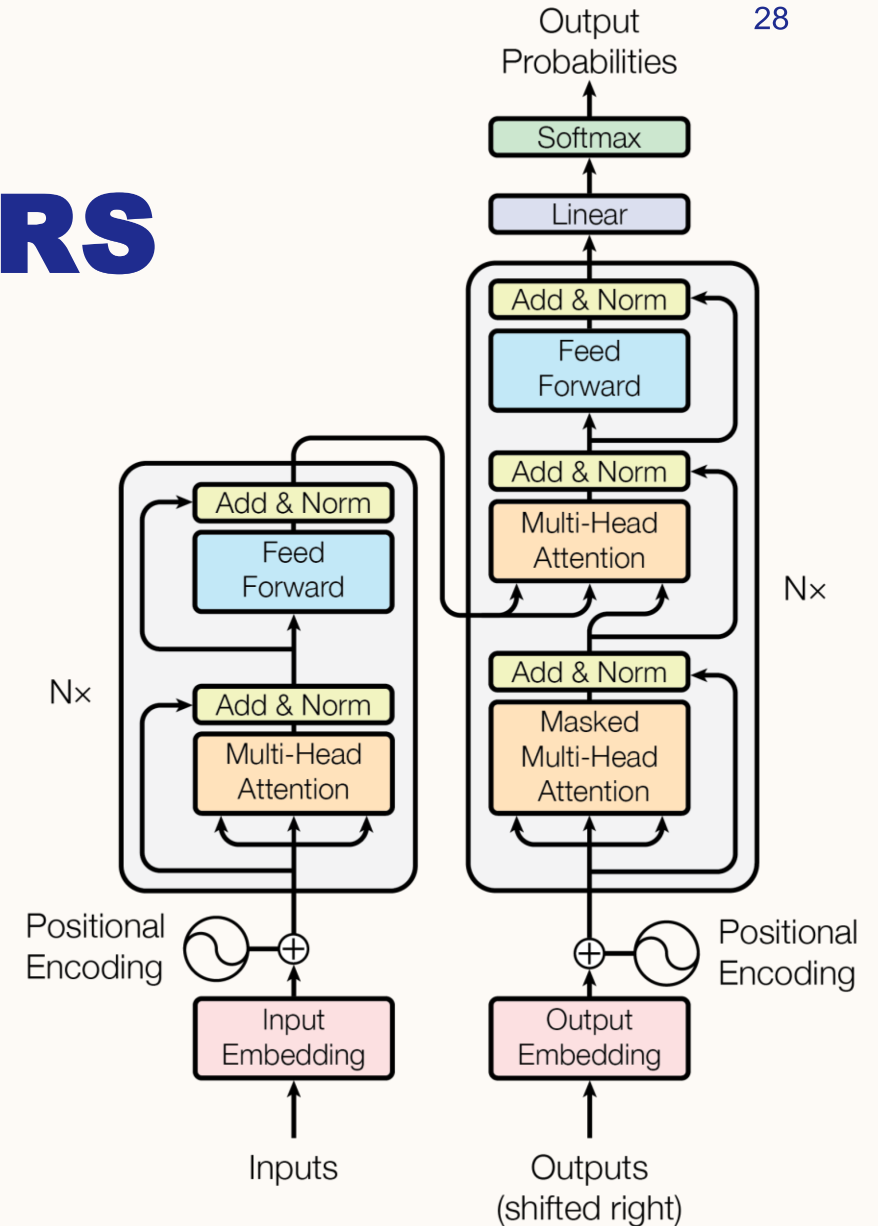
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

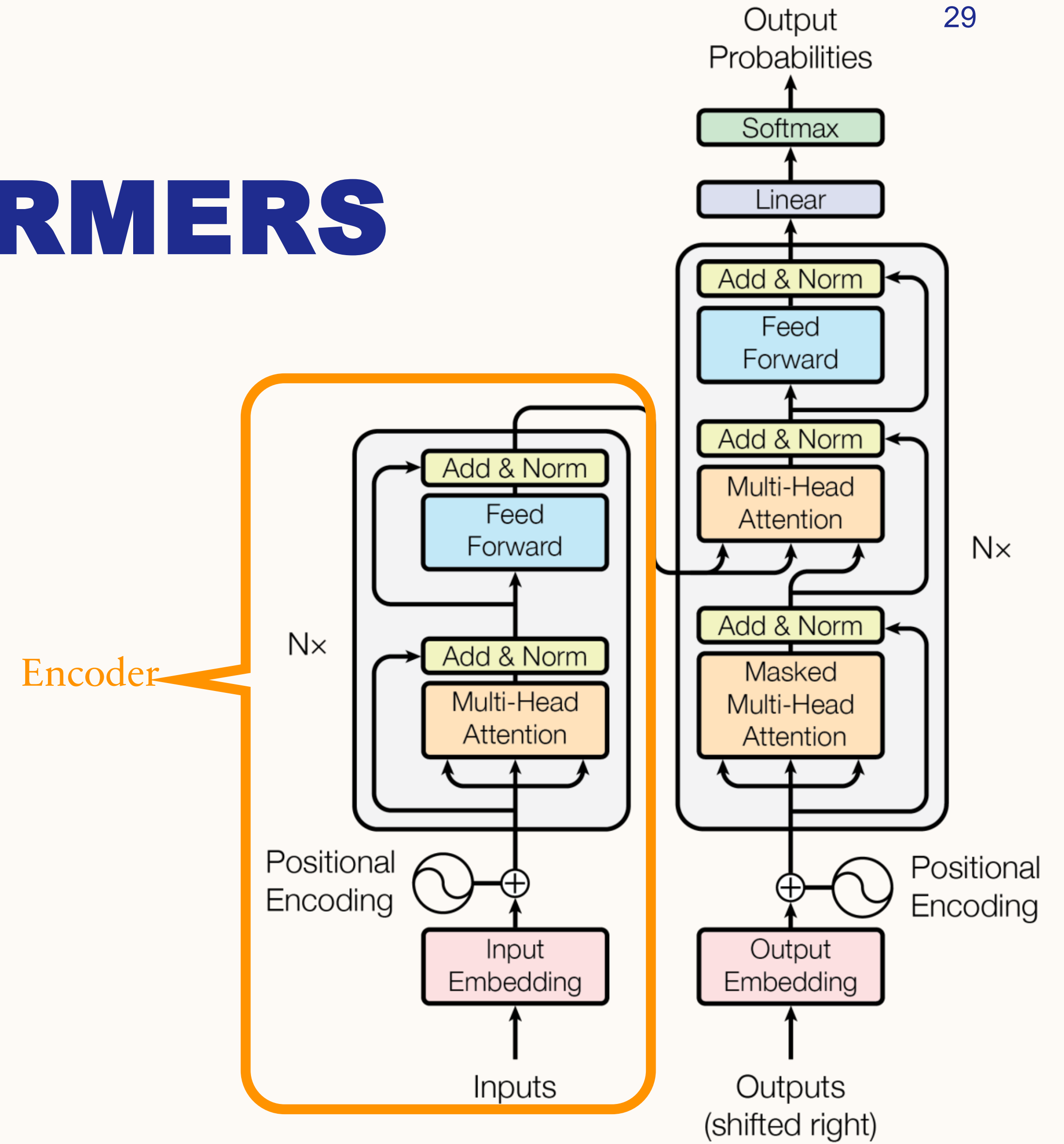
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, 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

TRANSFORMERS

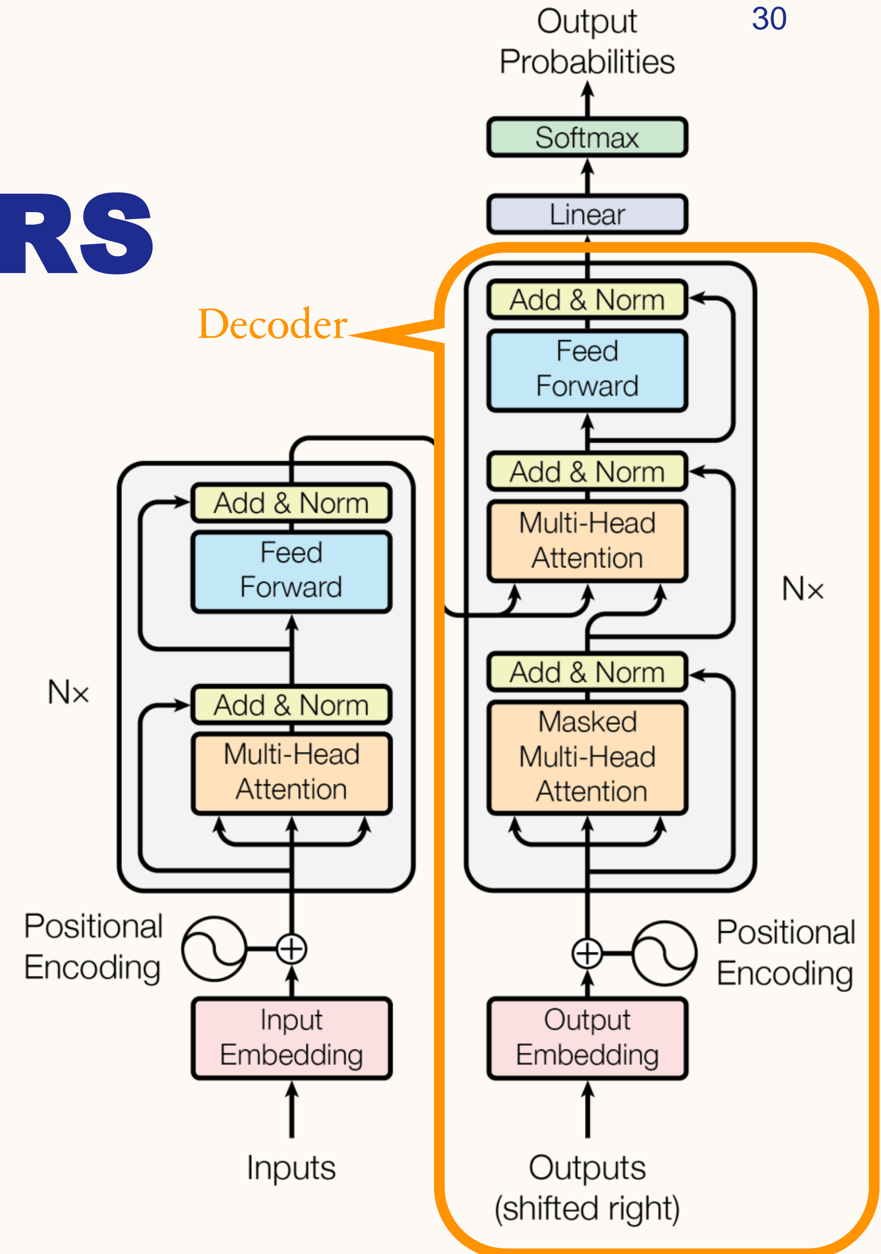
The Transformer is a non-recurrent non-convolutional (feed-forward) neural network designed for language understanding that introduces self-attention in addition to encoder-decoder attention.



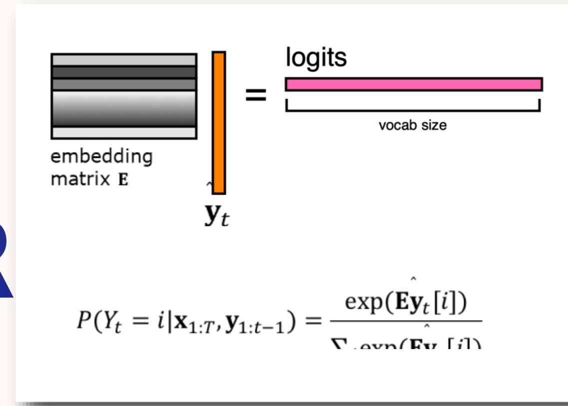
TRANSFORMERS

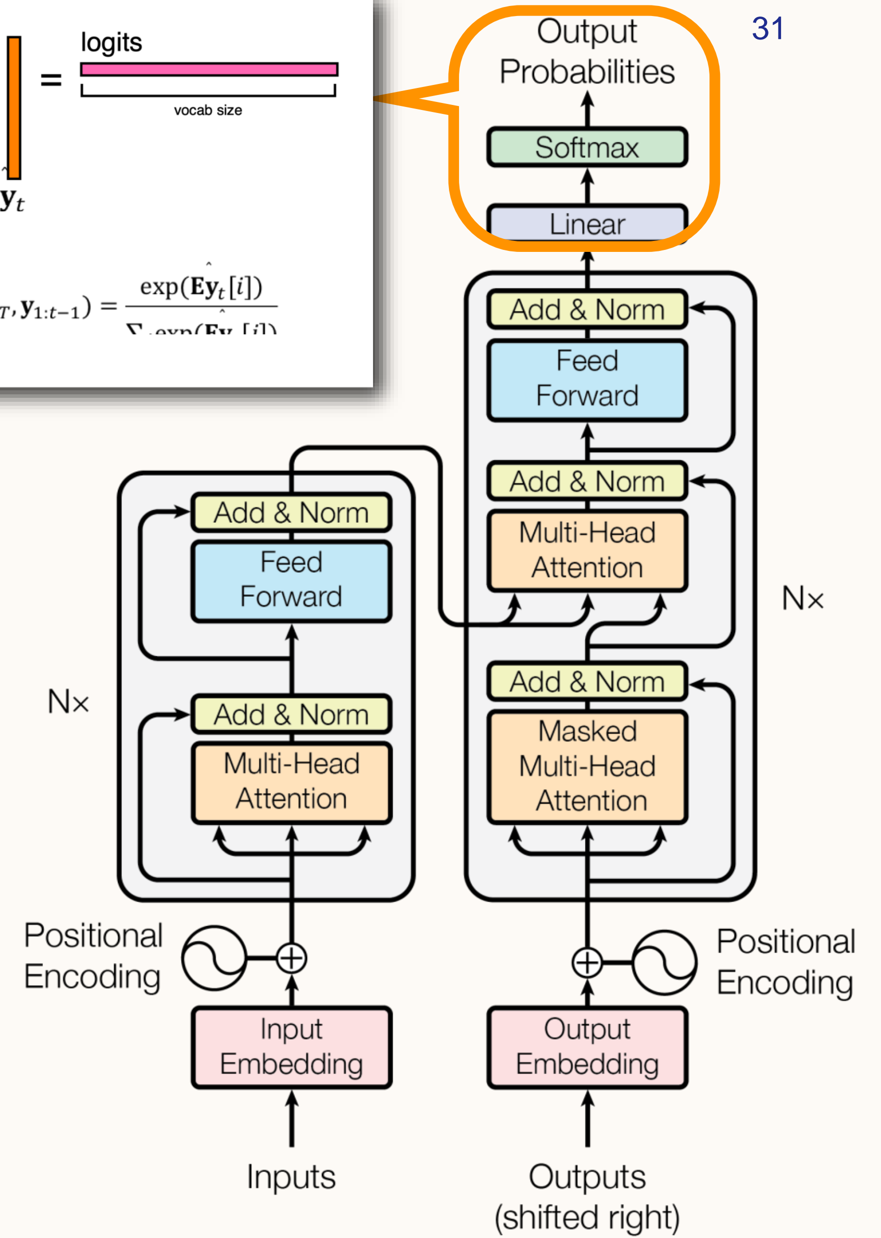


TRANSFORMERS

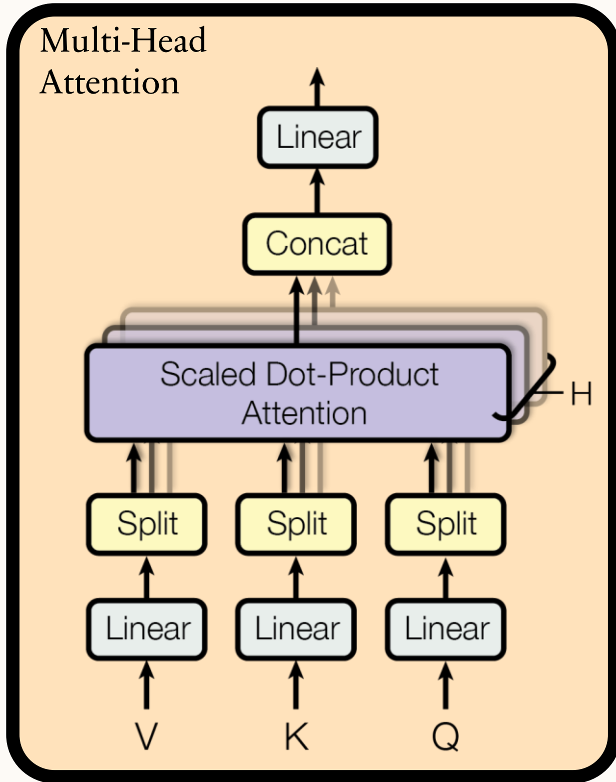


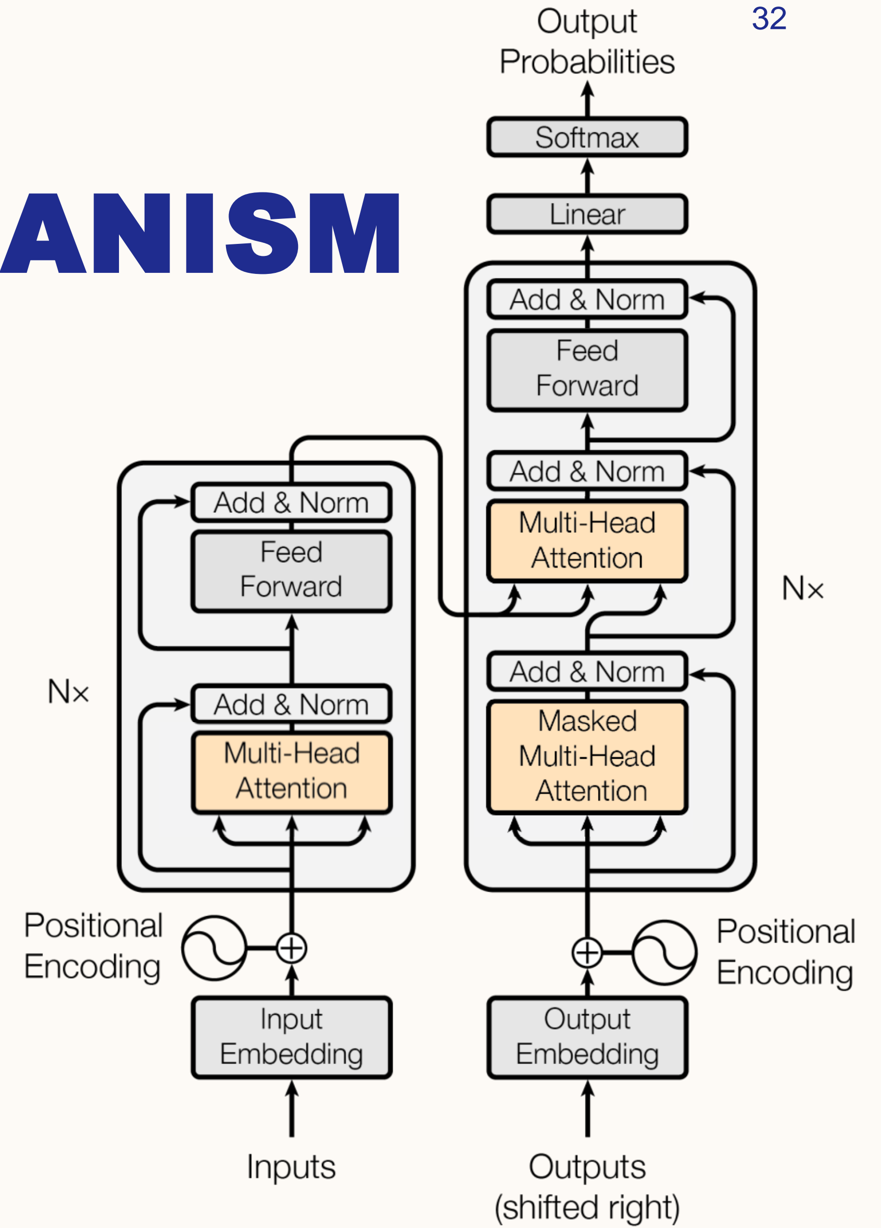
TRANSFOR



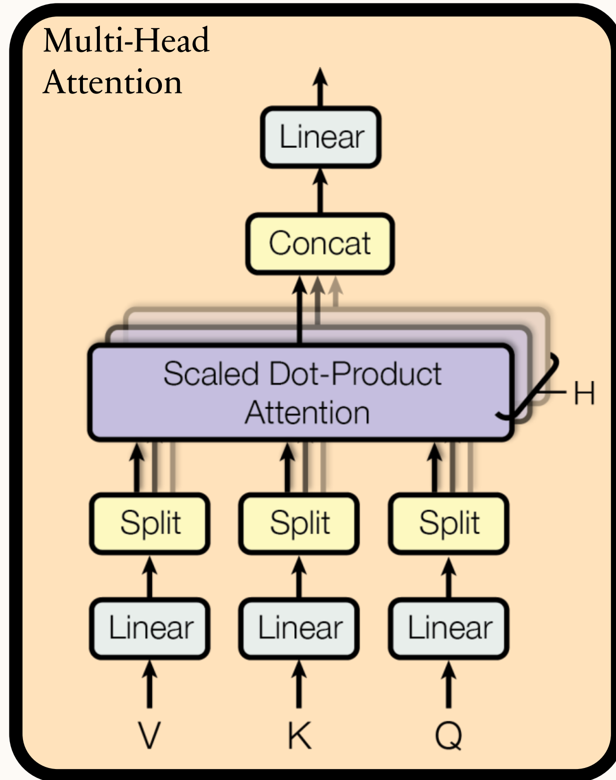


ATTENTION MECHANISM

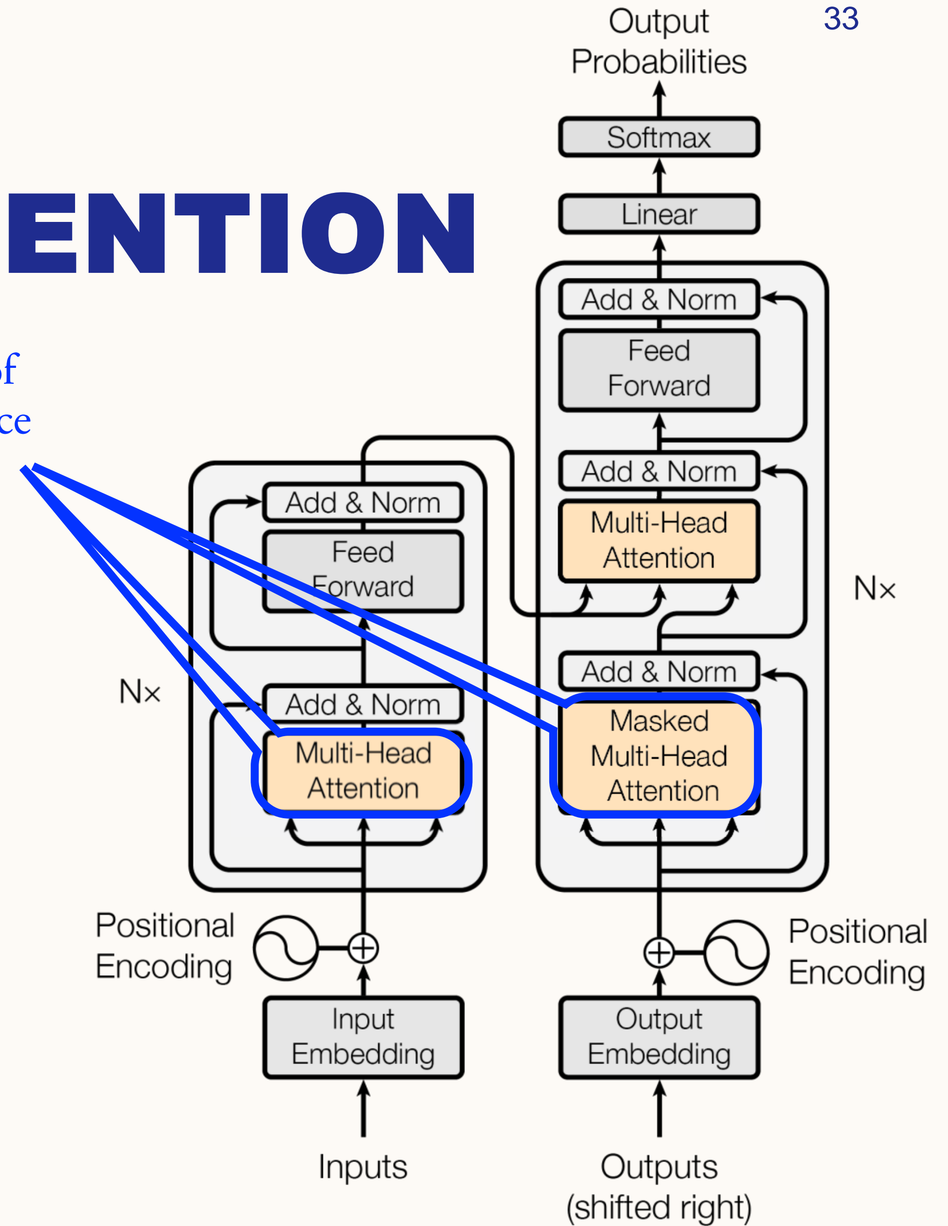




MULTI-HEAD ATTENTION

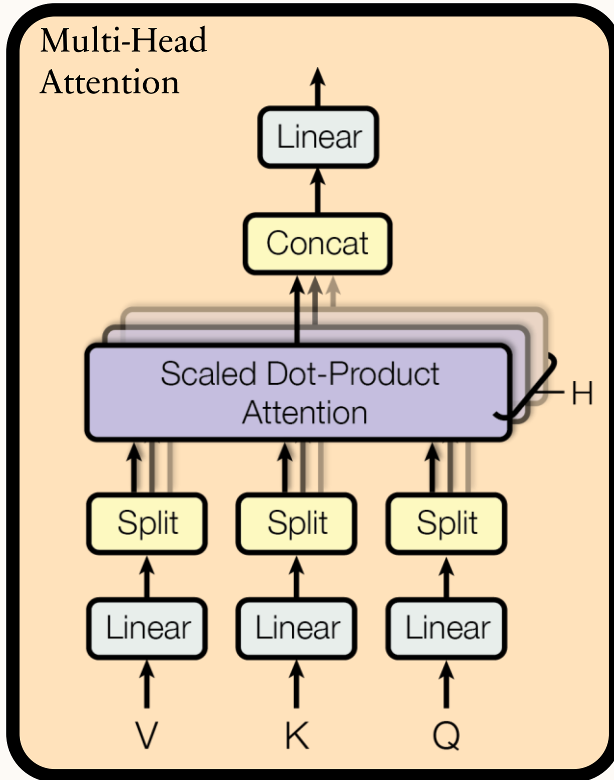


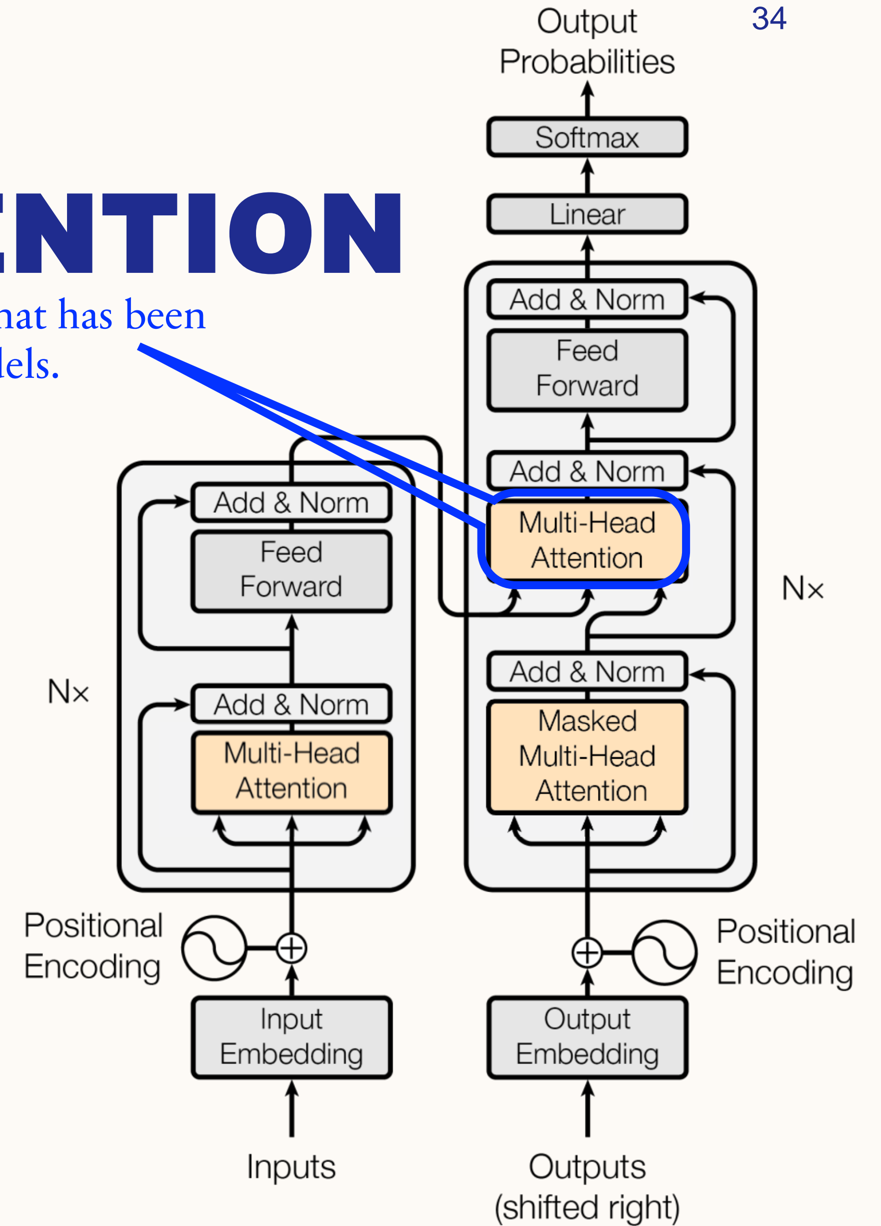
Self-attention between a sequence of hidden states and that same sequence of hidden states.



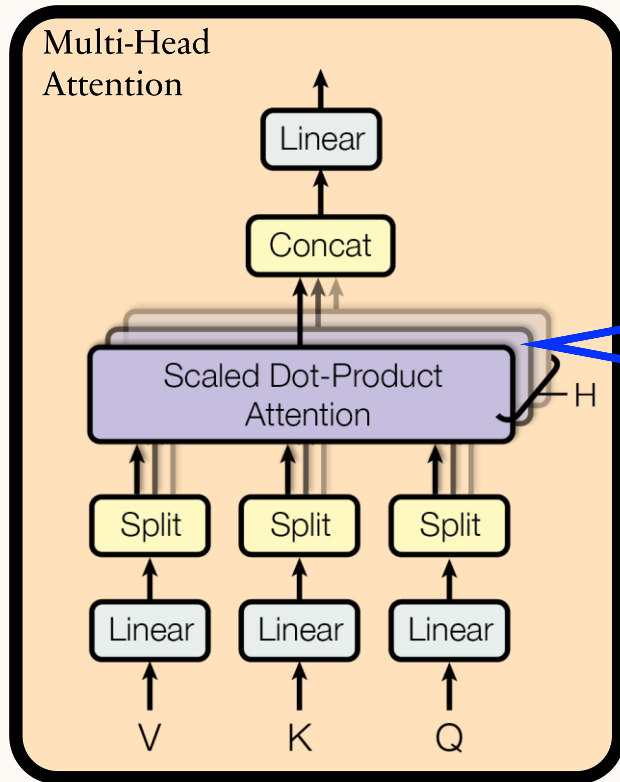
MULTI-HEAD ATTENTION

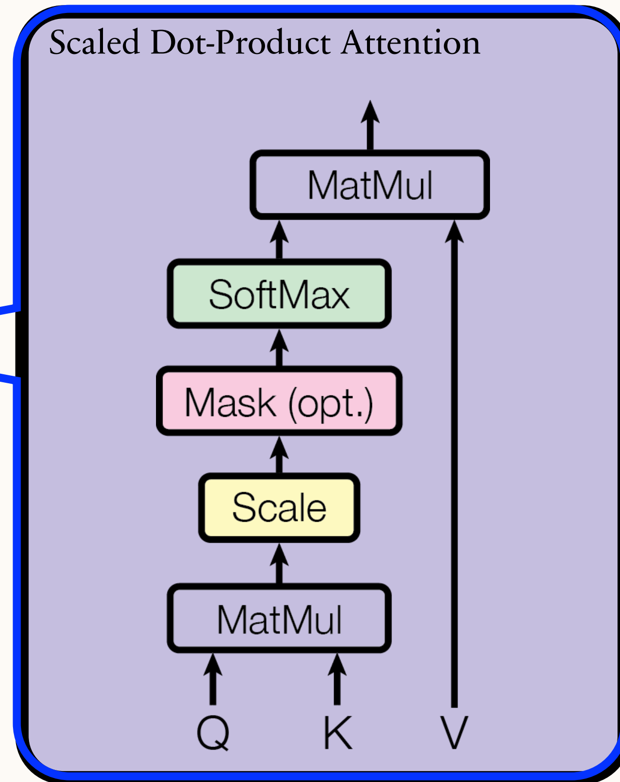
Encoder-decoder attention, like what has been standard in recurrent seq2seq models.

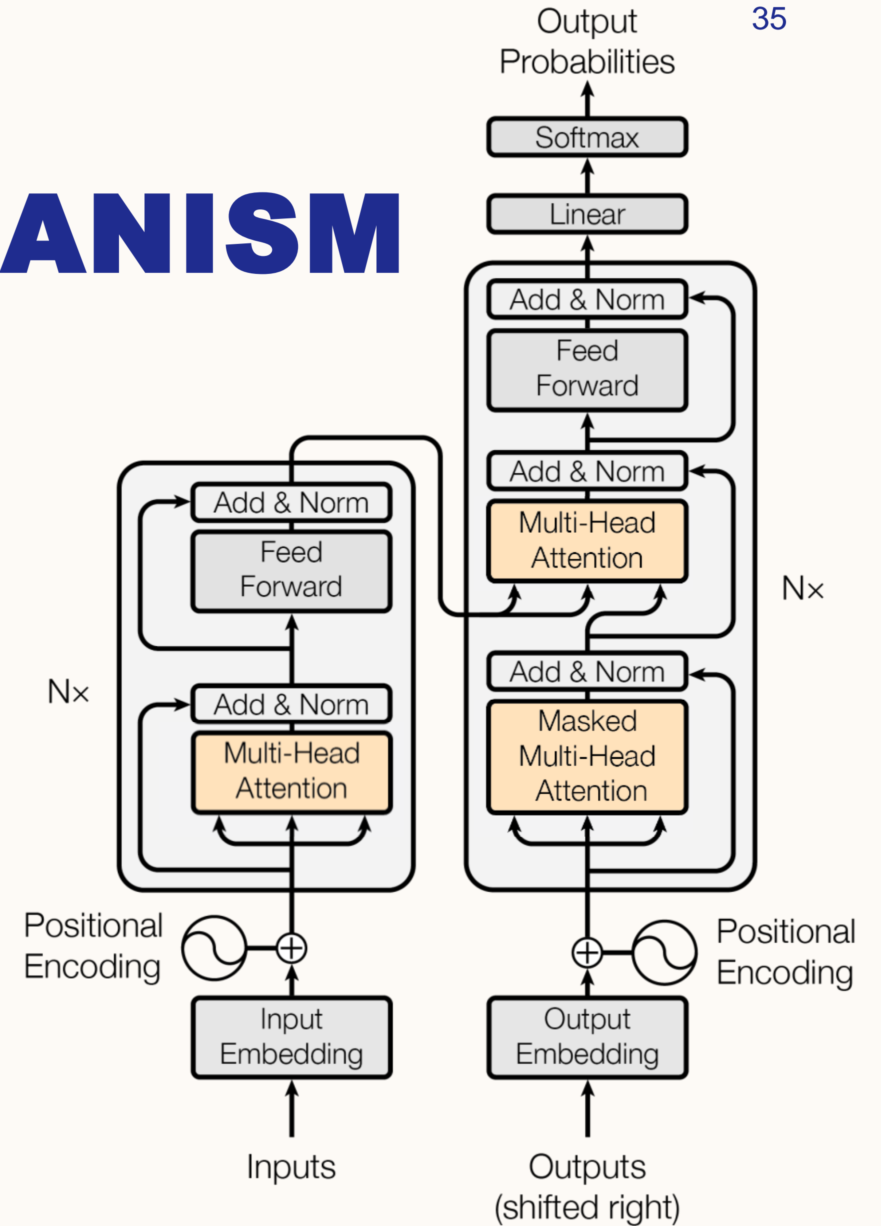




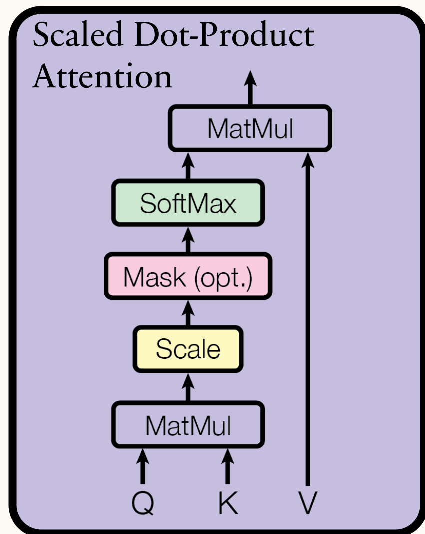
ATTENTION MECHANISM







SCALED DOT-PRODUCT ATTENTION

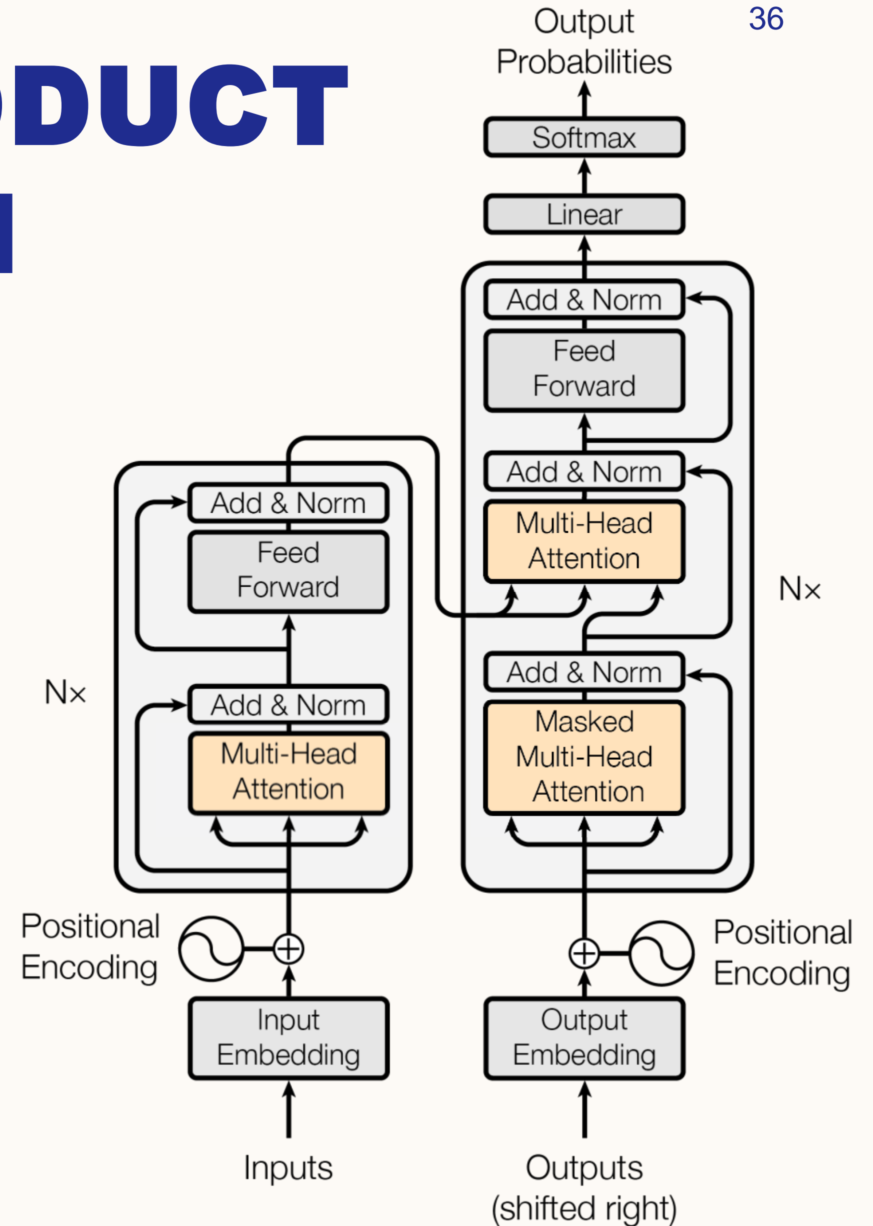


- The **scaled dot-product attention** mechanism is almost identical to the one we looked at, but let's turn it into matrix multiplications.

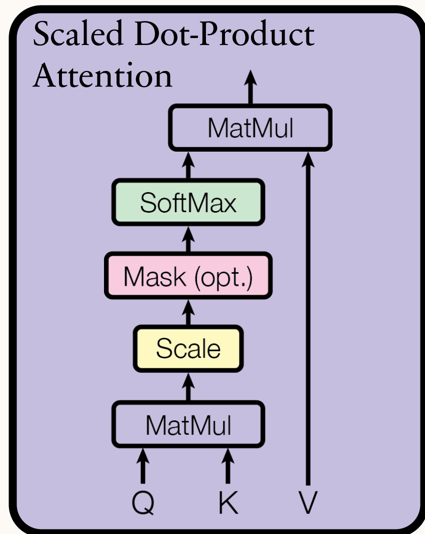
- The query: $\mathbf{Q} \in R^{Txd_k}$
- The key: $\mathbf{K} \in R^{T'xd_k}$
- The value: $\mathbf{V} \in R^{Txd_k}$

This is the α vector we learned about before.

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



SCALED DOT-PRODUCT ATTENTION

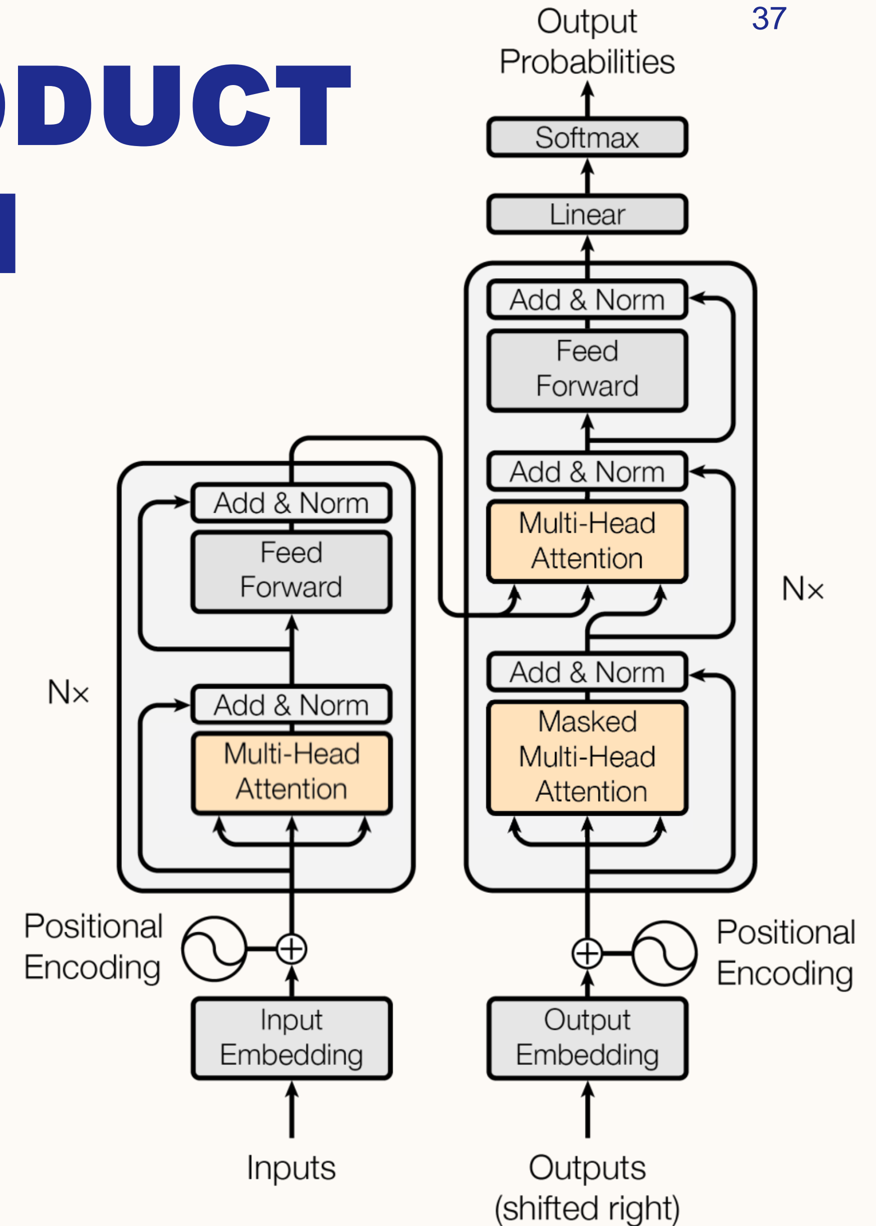


- The **scaled dot-product attention** mechanism is almost identical to the one we looked at, but let's turn it into matrix multiplications.

- The query: $\mathbf{Q} \in R^{Txd_k}$
- The key: $\mathbf{K} \in R^{T'xd_k}$
- The value: $\mathbf{V} \in R^{Txd_k}$

This is the dot-product scoring function from previous slides

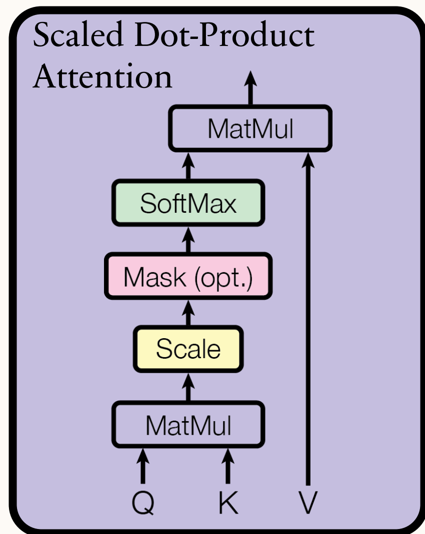
- $\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)\mathbf{V}$



The $\sqrt{d_k}$ in the denominator prevents the dot product from getting too big

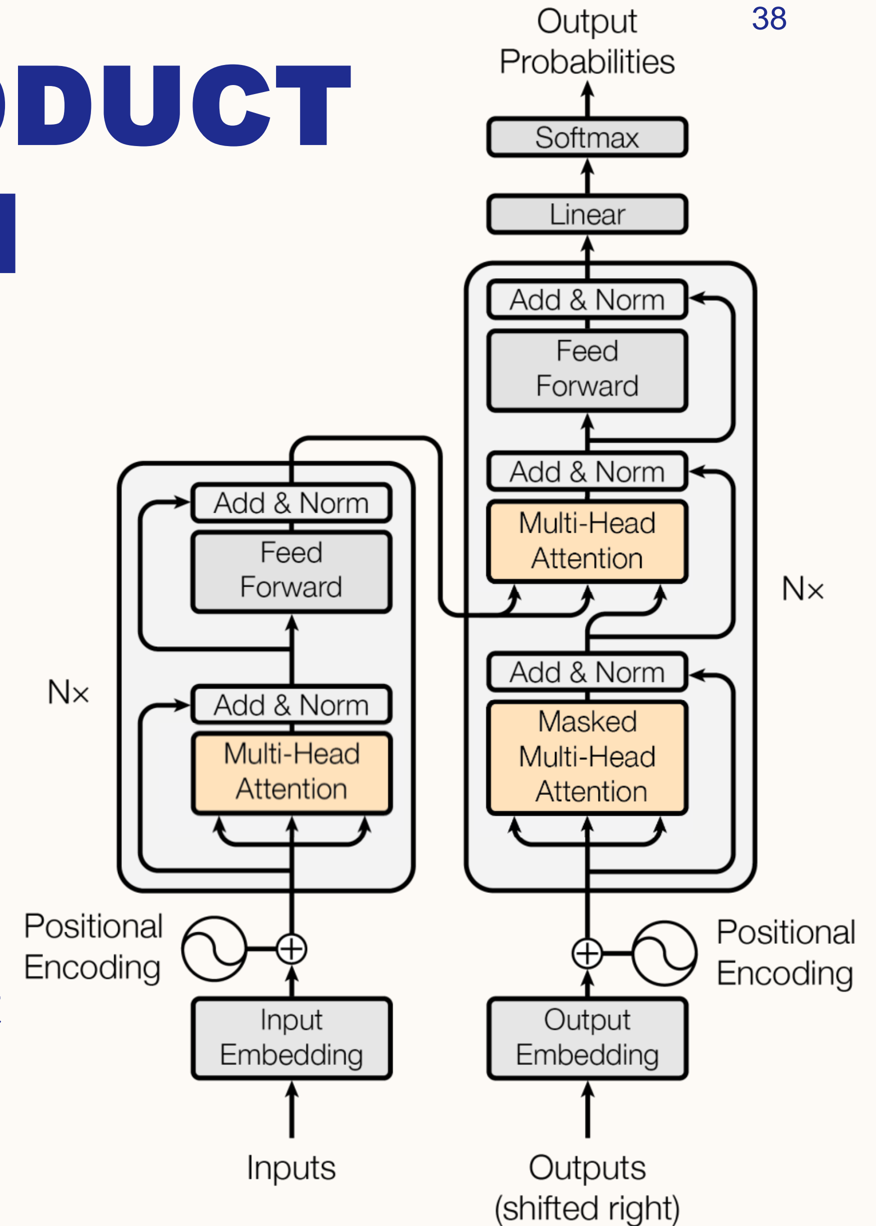
SCALED DOT-PRODUCT ATTENTION

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



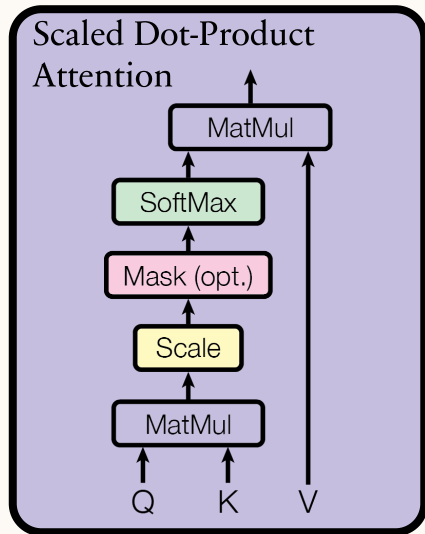
The rough algorithm:

- For each vector in Q (query matrix), take the linear sum of the vectors in V (value matrix)
- The amount to weigh each vector in V is dependent on how "similar" that vector is to the query vector
- "Similarity" is measured in terms of the dot product between the vectors



SCALED DOT-PRODUCT ATTENTION

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

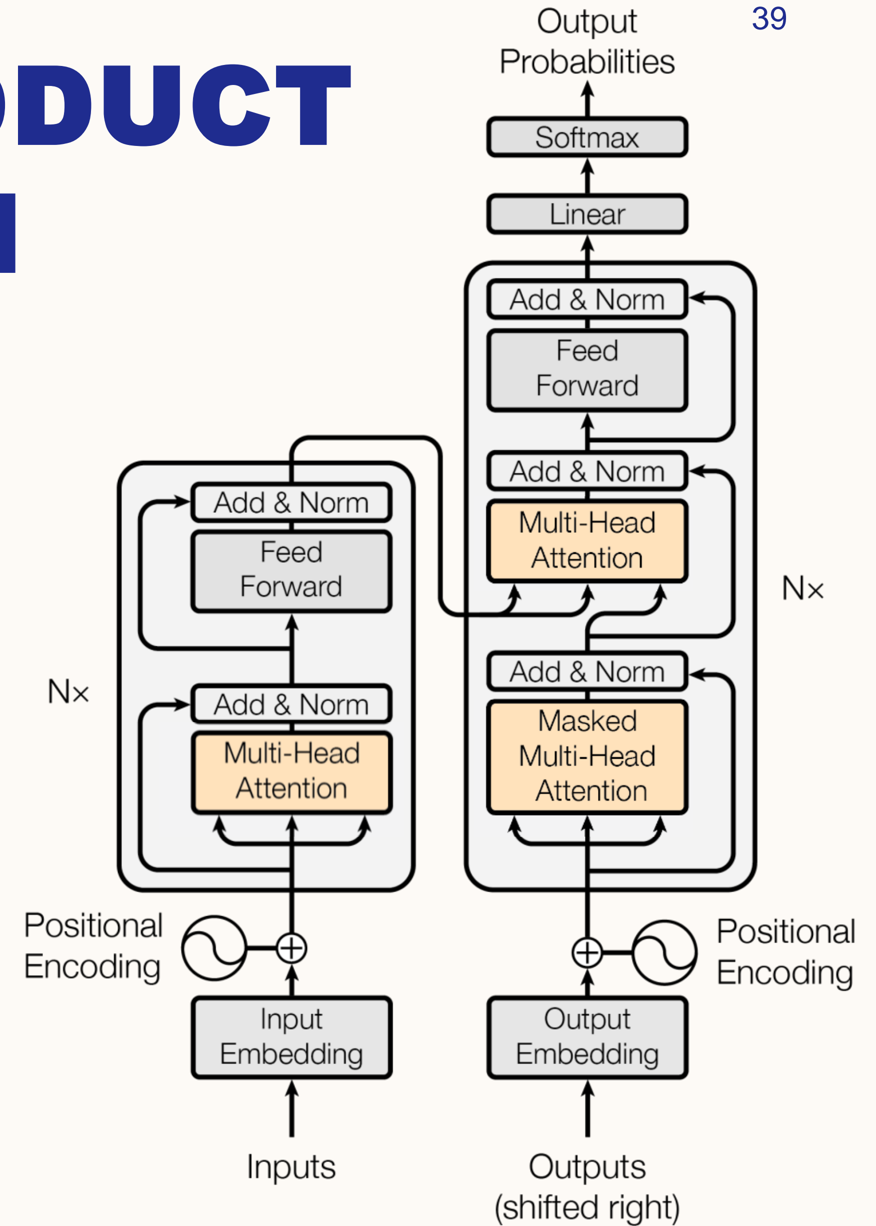


For self-attention:

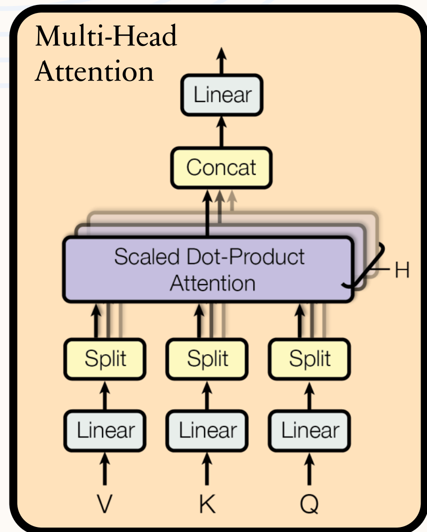
Keys, queries, and values all come from the outputs of the previous layer

For encoder-decoder attention:

Keys and values come from encoder's final output. Queries come from the previous decoder layer's outputs.



MULTI-HEAD ATTENTION



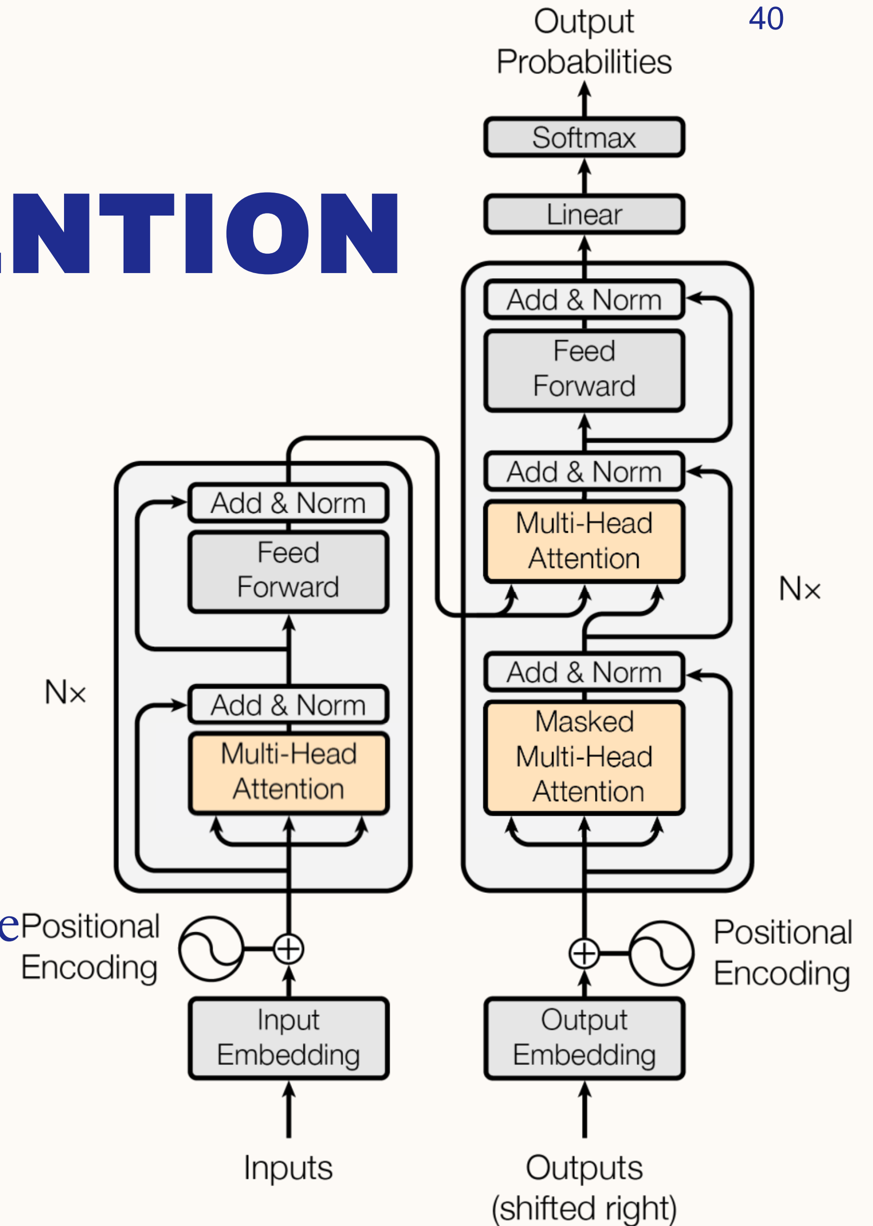
$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

$$\text{MultiHeadAtt}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Concat}(\text{head}_1, \ldots \text{head}_h)\mathbf{W}^O$$

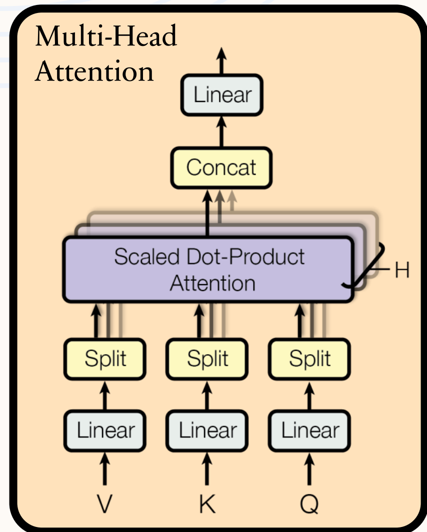
Instead of operating on **Q**, **K**, and **V** mechanism projects each input into a smaller dimension. This is done *h* times.

The attention operation is performed on each of these "heads," and the results are concatenated.

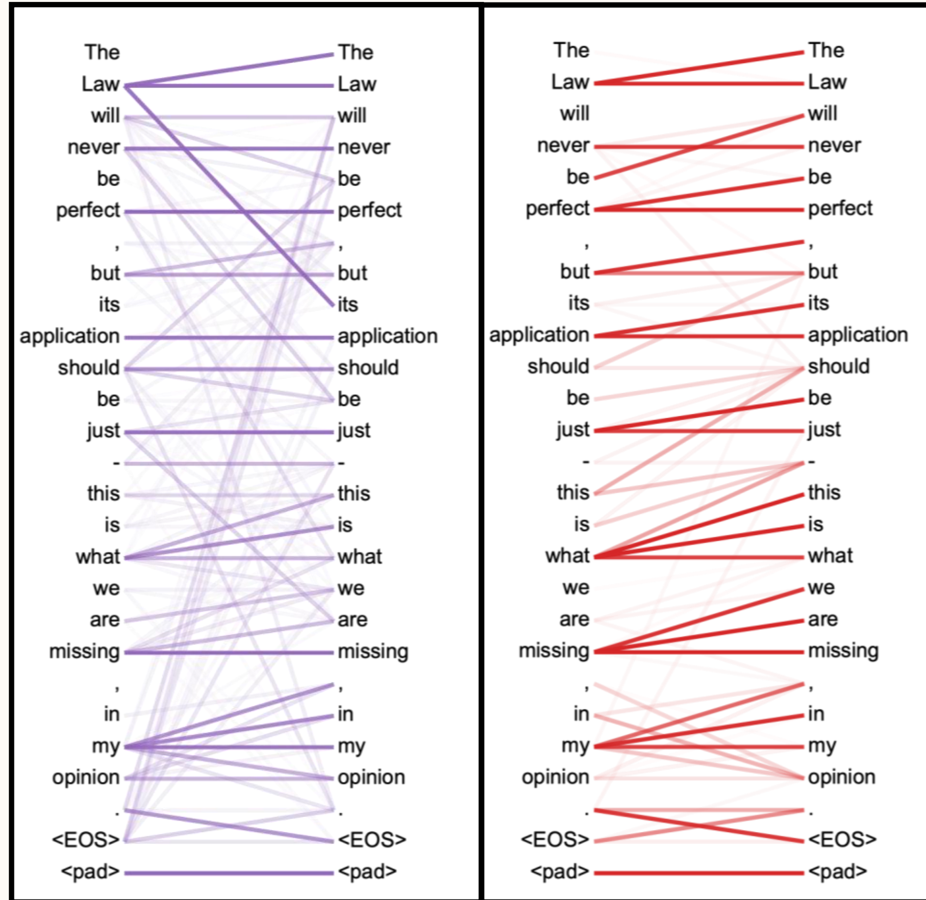
Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

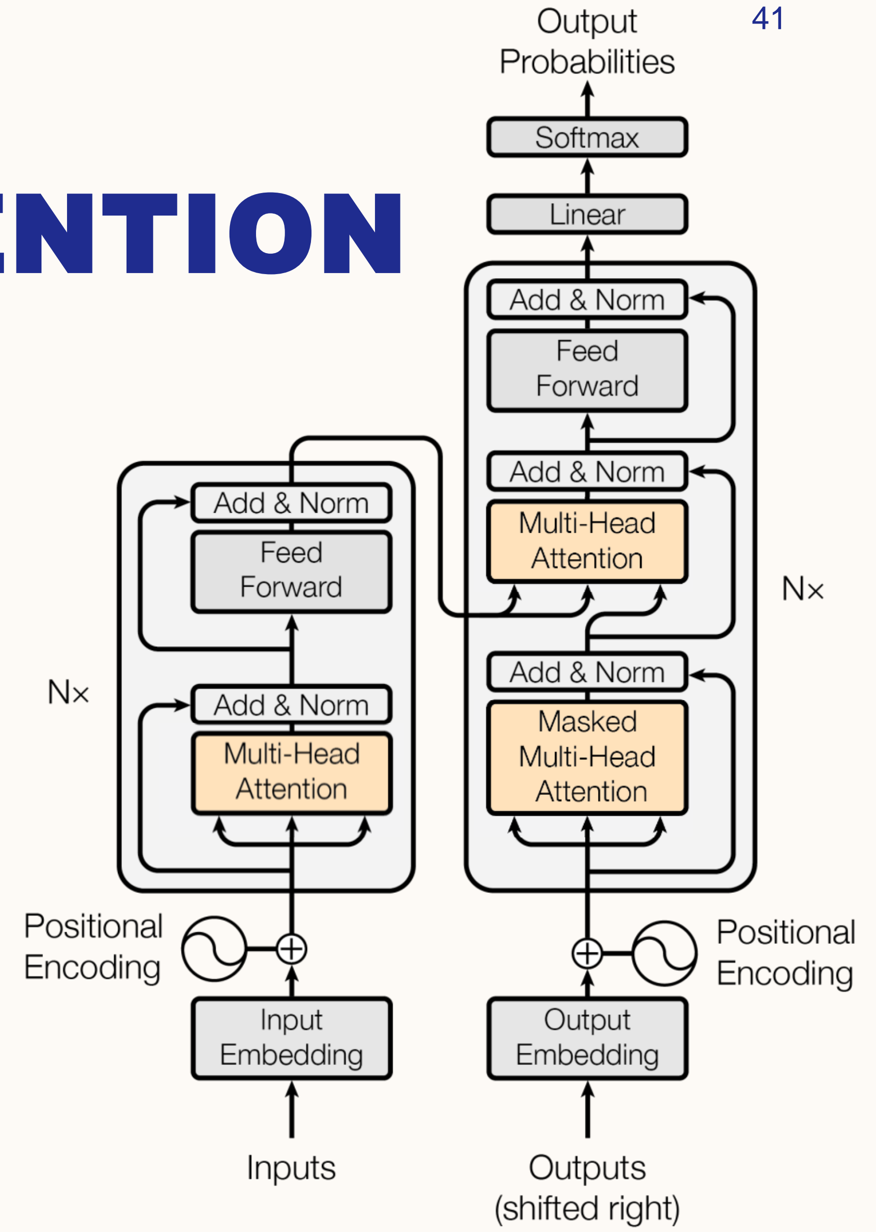


MULTI-HEAD ATTENTION



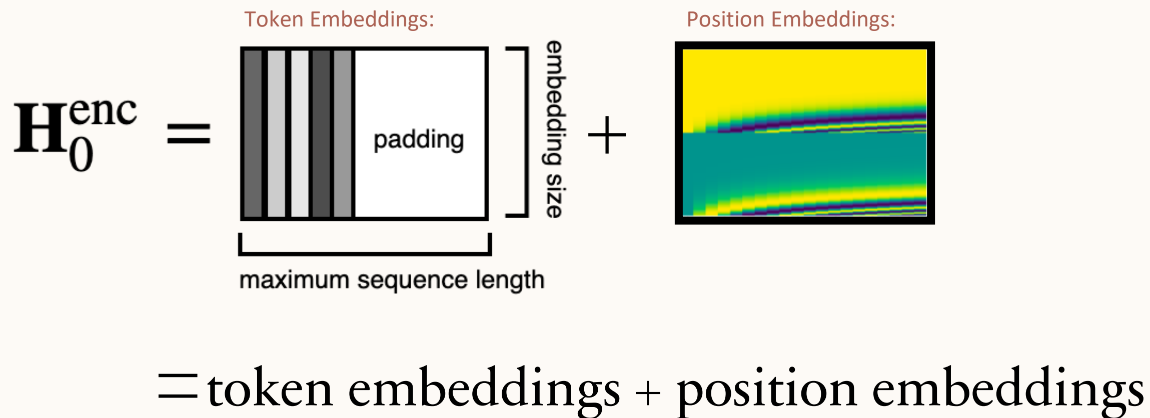
Two different self-attention heads:



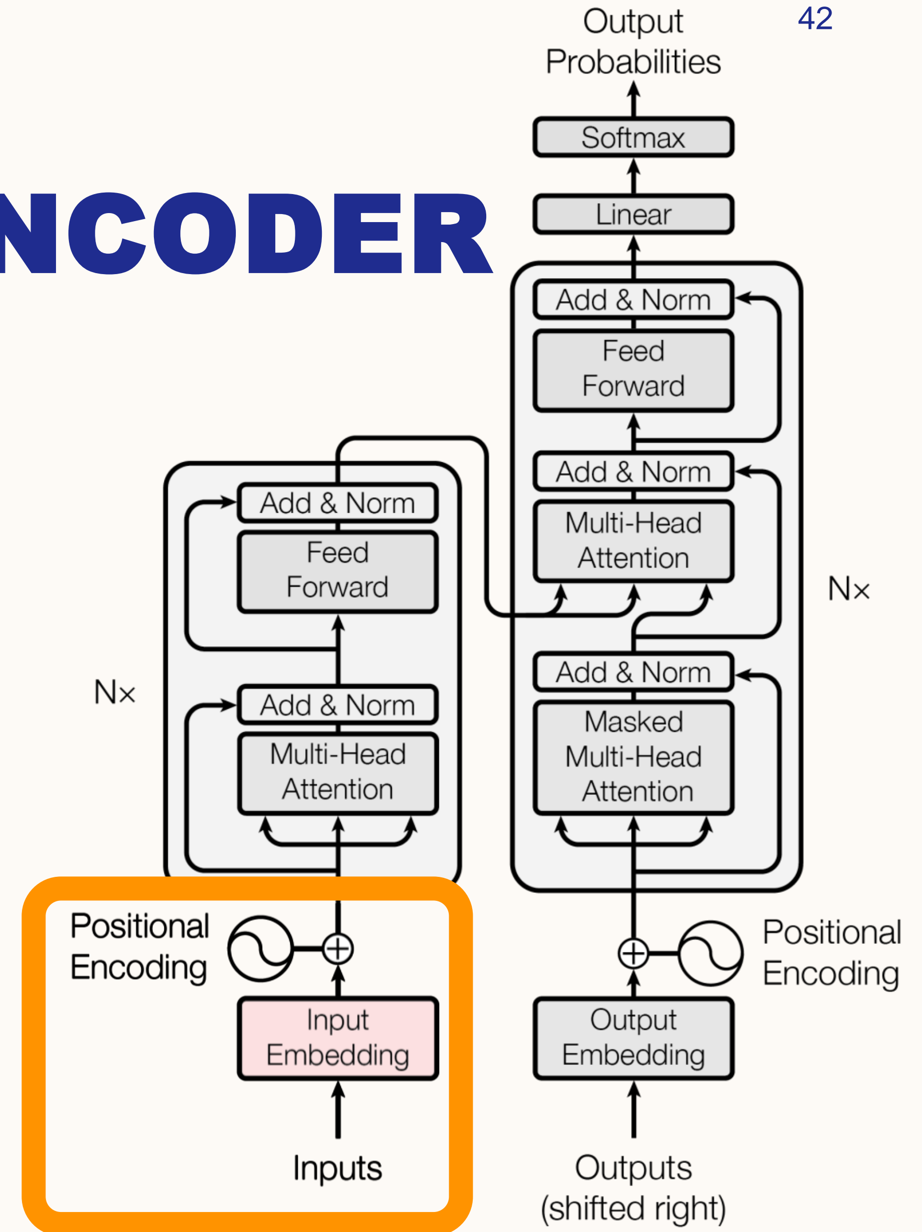


INPUTS TO THE ENCODER

- The input into the encoder looks like:

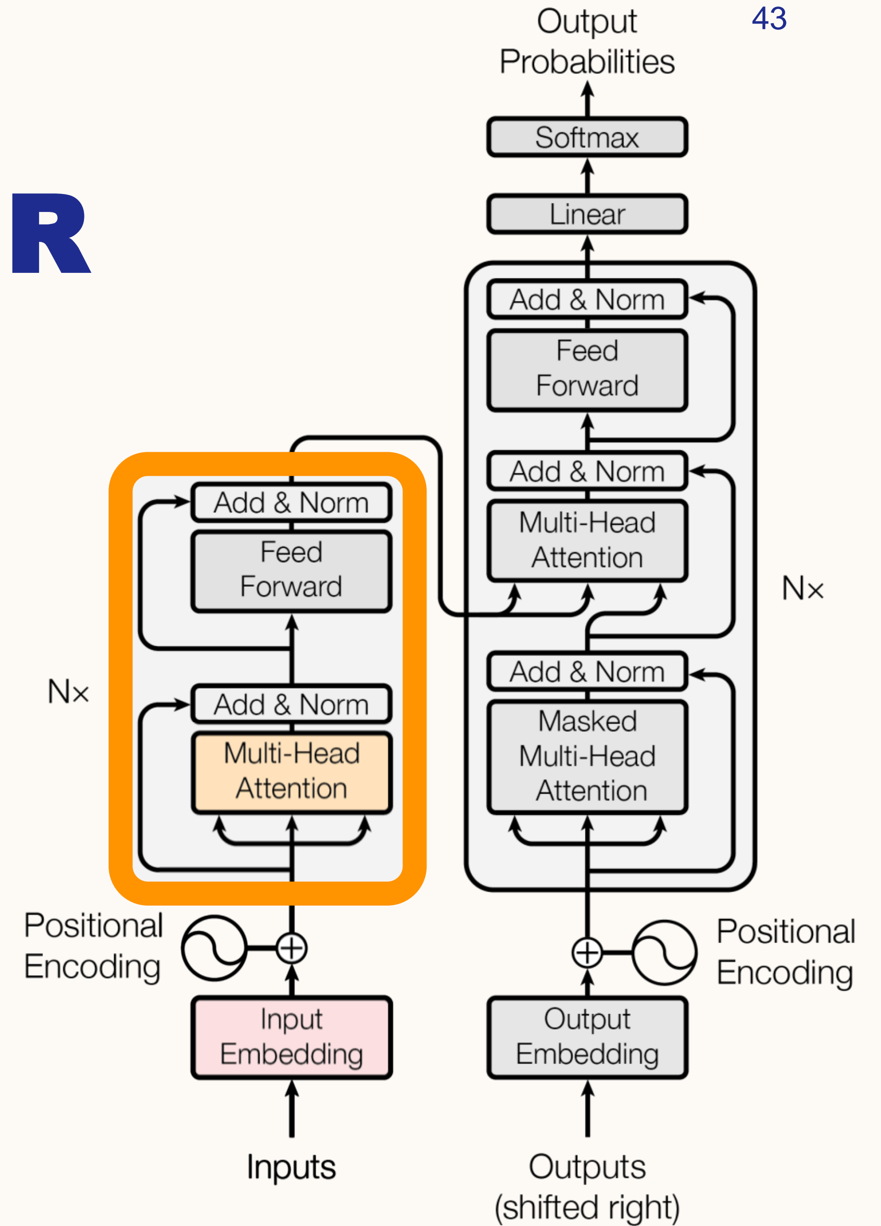


= token embeddings + position embeddings



THE ENCODER

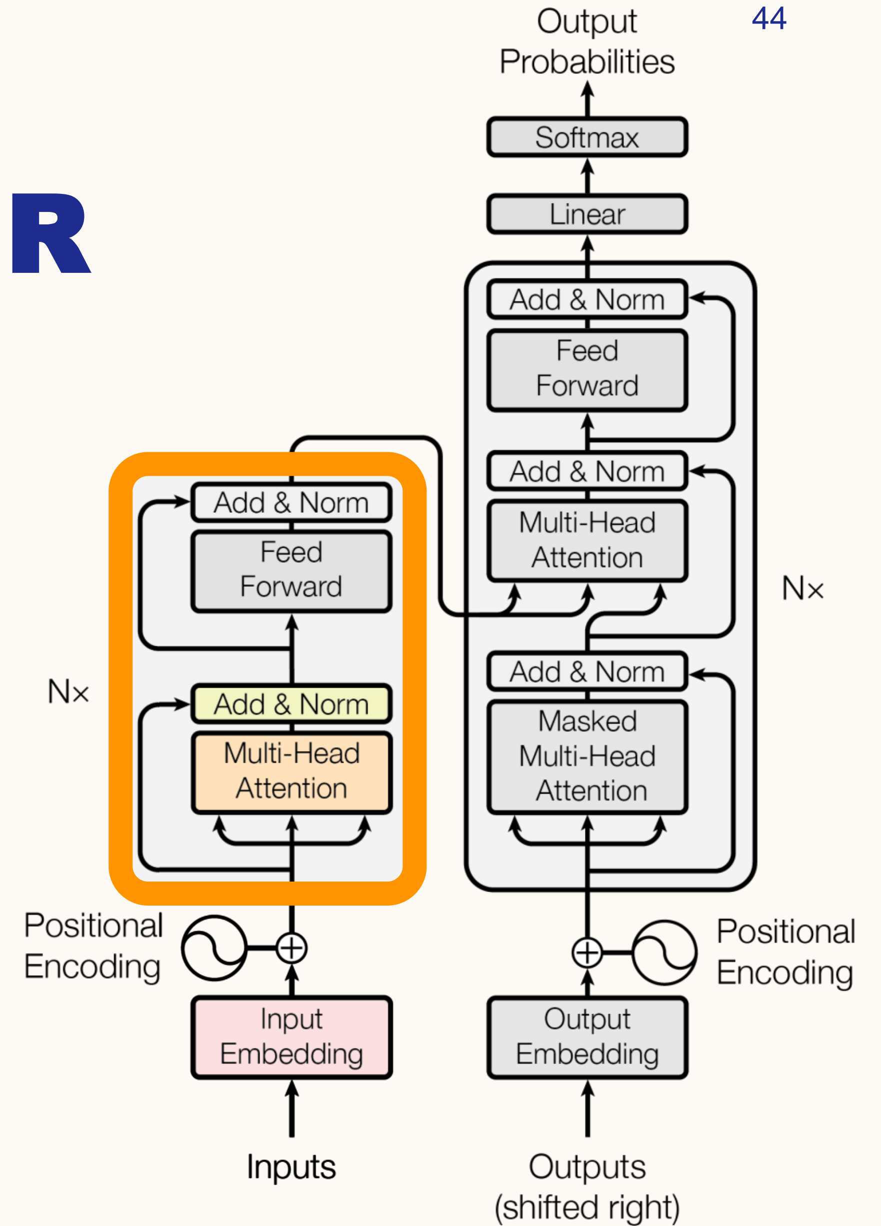
Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc})$



THE ENCODER

Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc})$

Add & Norm $= \text{LayerNorm}($ Multi-Head Attention $+ \mathbf{H}_i^{enc})$

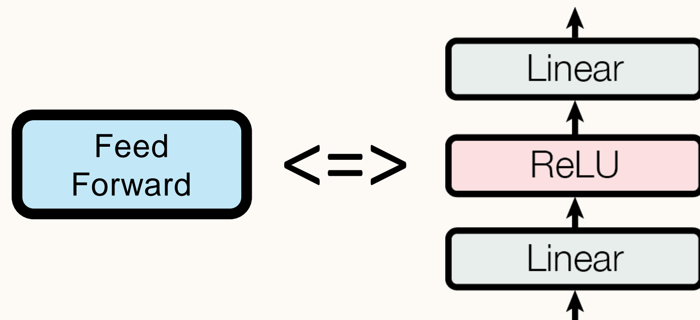


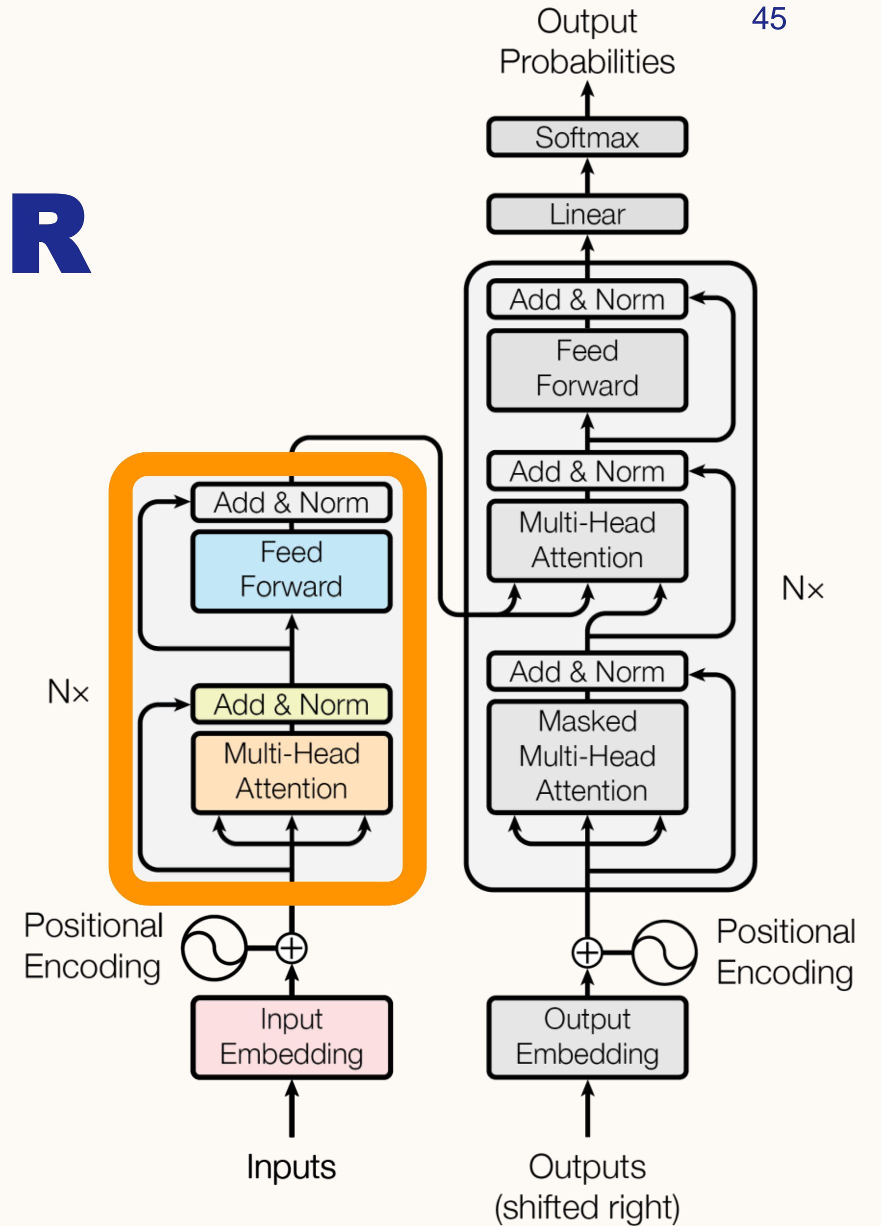
THE ENCODER

Multi-Head Attention = MultiHeadAtt($\mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}$)

Add & Norm = LayerNorm(Multi-Head Attention $+ \mathbf{H}_i^{enc}$)

Feed Forward = $max(0,$ Add & Norm $\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$





THE ENCODER

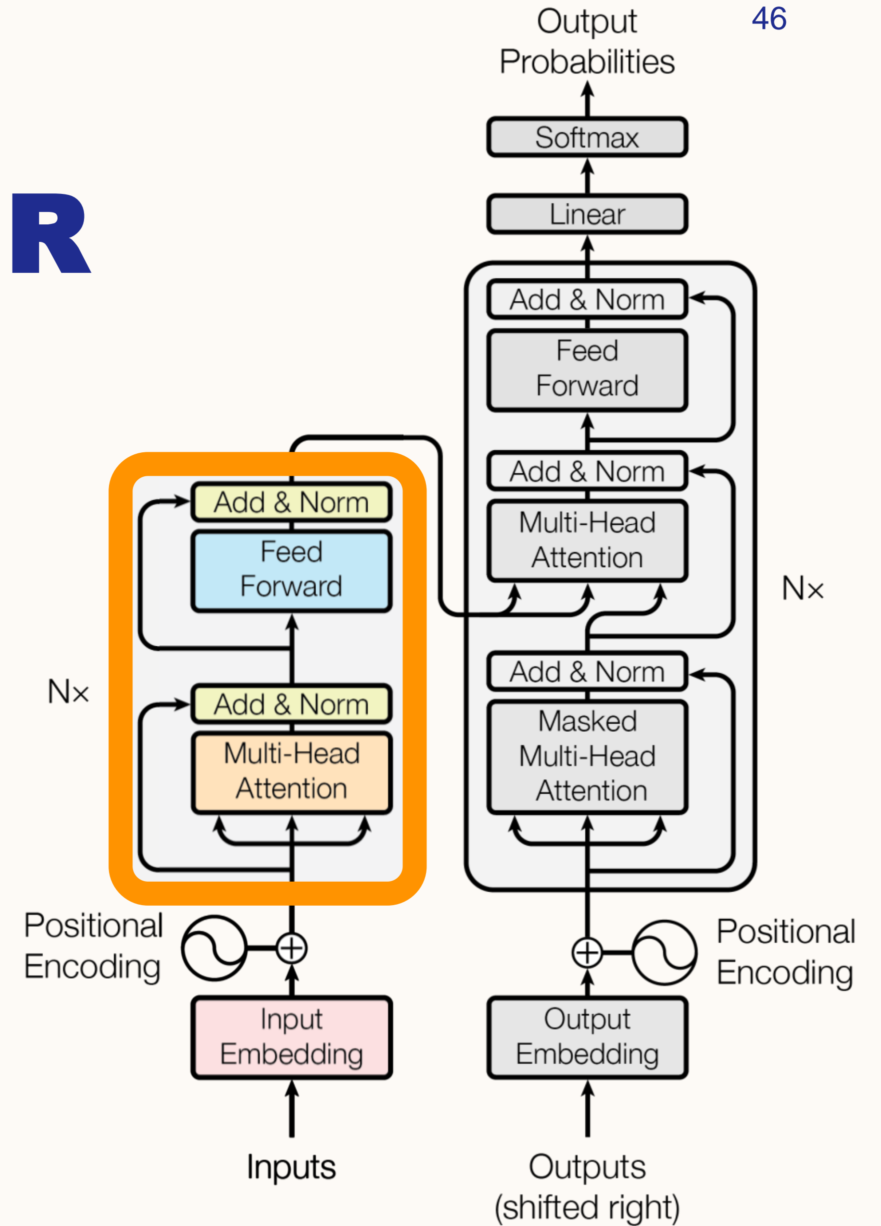
[Multi-Head Attention] = MultiHeadAtt($\mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}$)

[Add & Norm] = LayerNorm([Multi-Head Attention] + $\mathbf{H}_i^{enc}$)

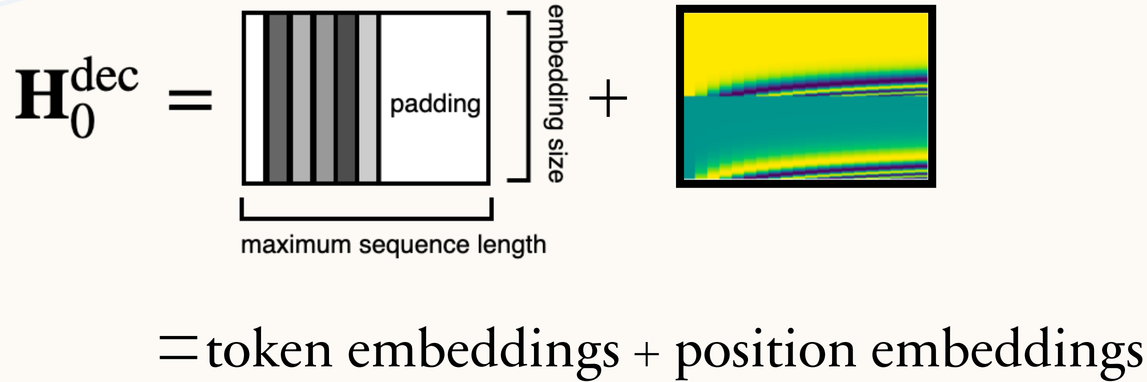
[Feed Forward] = $max(0,$ [Add & Norm] $\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$

[Add & Norm (2)] = LayerNorm([Feed Forward] + $\mathbf{H}_i^{enc}$)

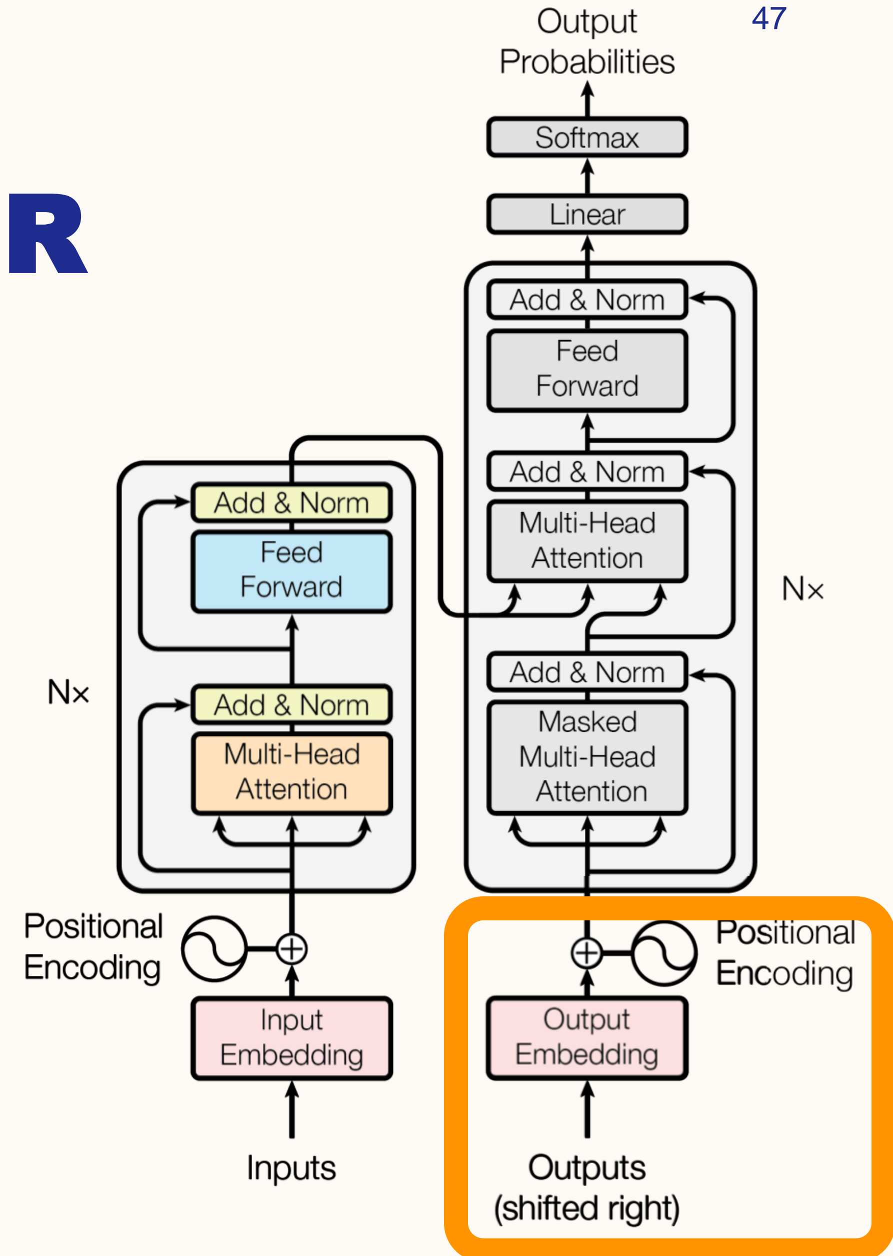
$\mathbf{H}_{i+1}^{enc}$ = [Add & Norm (2)]



THE DECODER

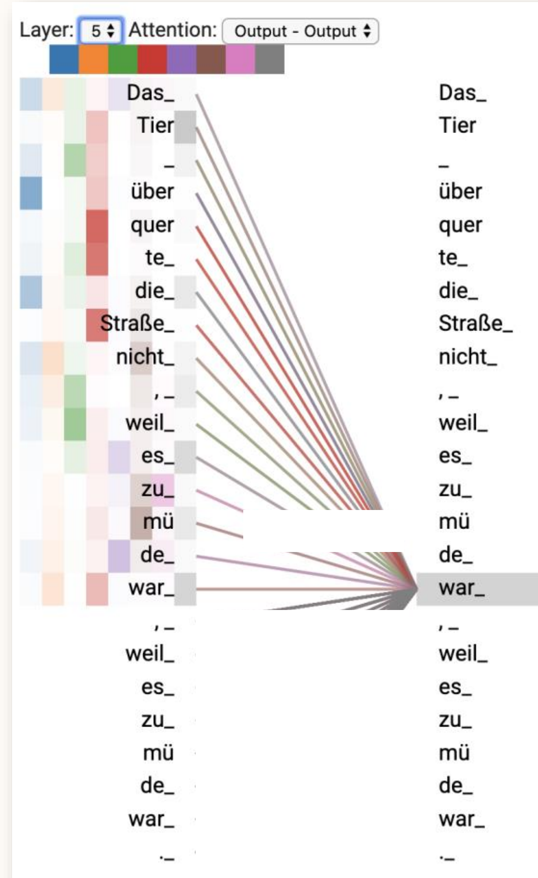


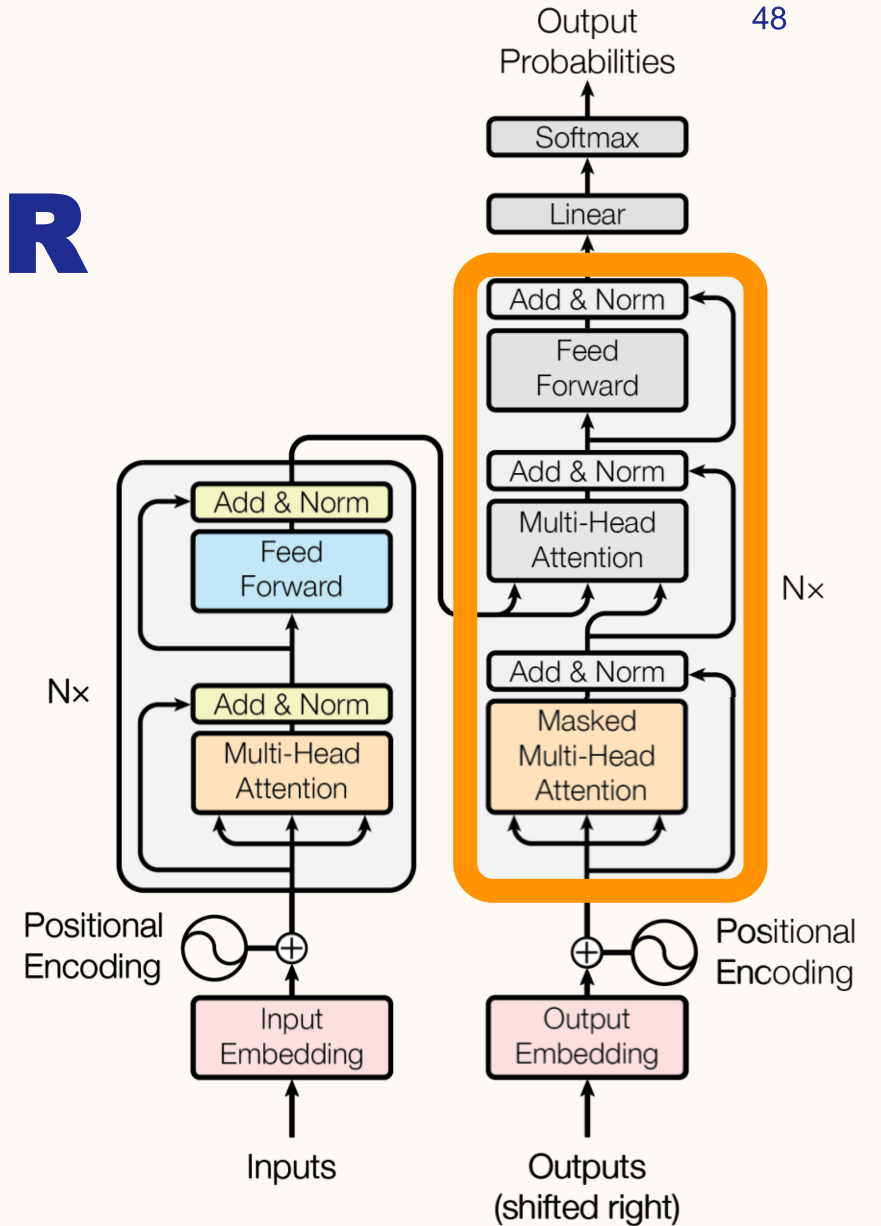
= token embeddings + position embeddings



THE DECODER

Masked Multi-Head Attention $= \text{MaskedMultiHeadAtt}(\mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}, \mathbf{H}_i^{dec})$

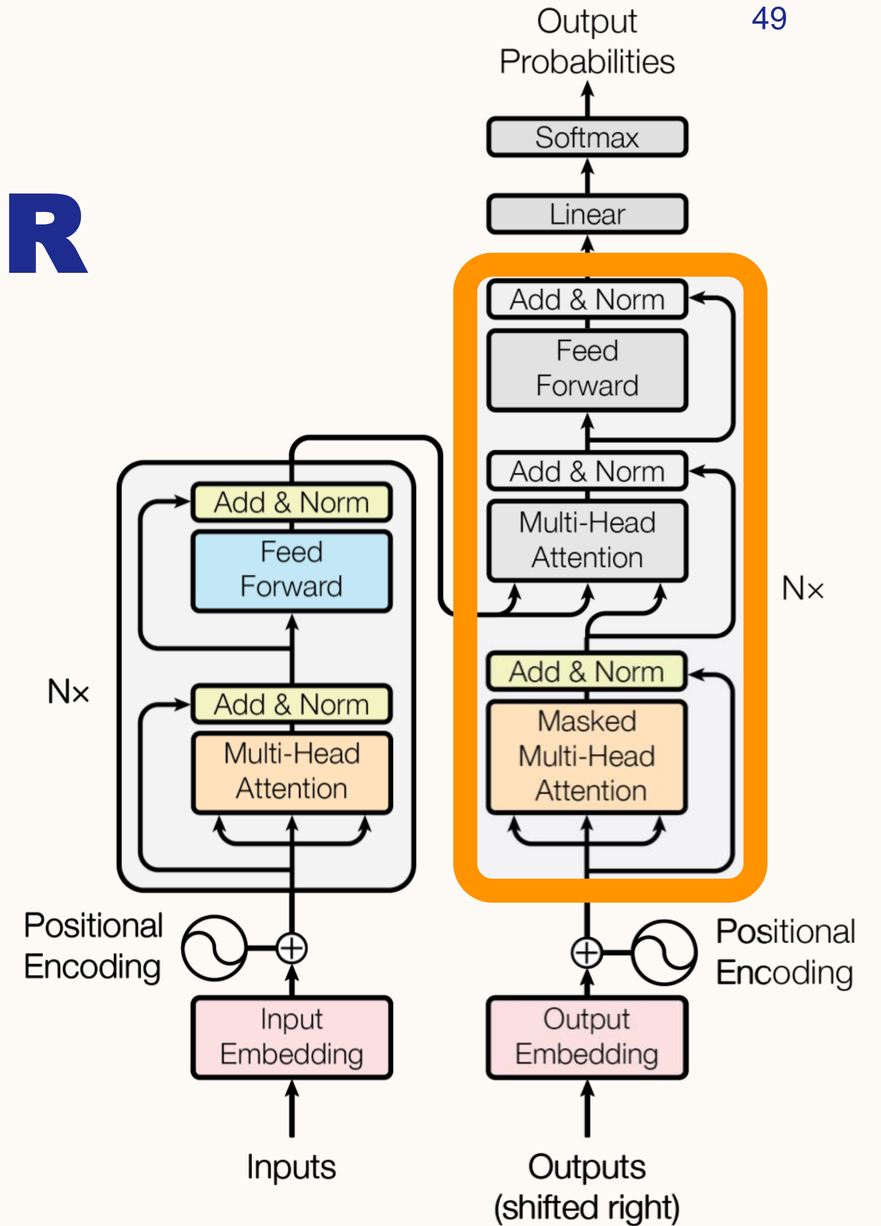




THE DECODER

Masked Multi-Head Attention $= \text{MaskedMultiHeadAtt}(\mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}, \mathbf{H}_i^{dec})$

Add & Norm $= \text{LayerNorm}($ Multi-Head Attention $+ \mathbf{H}_i^{dec})$

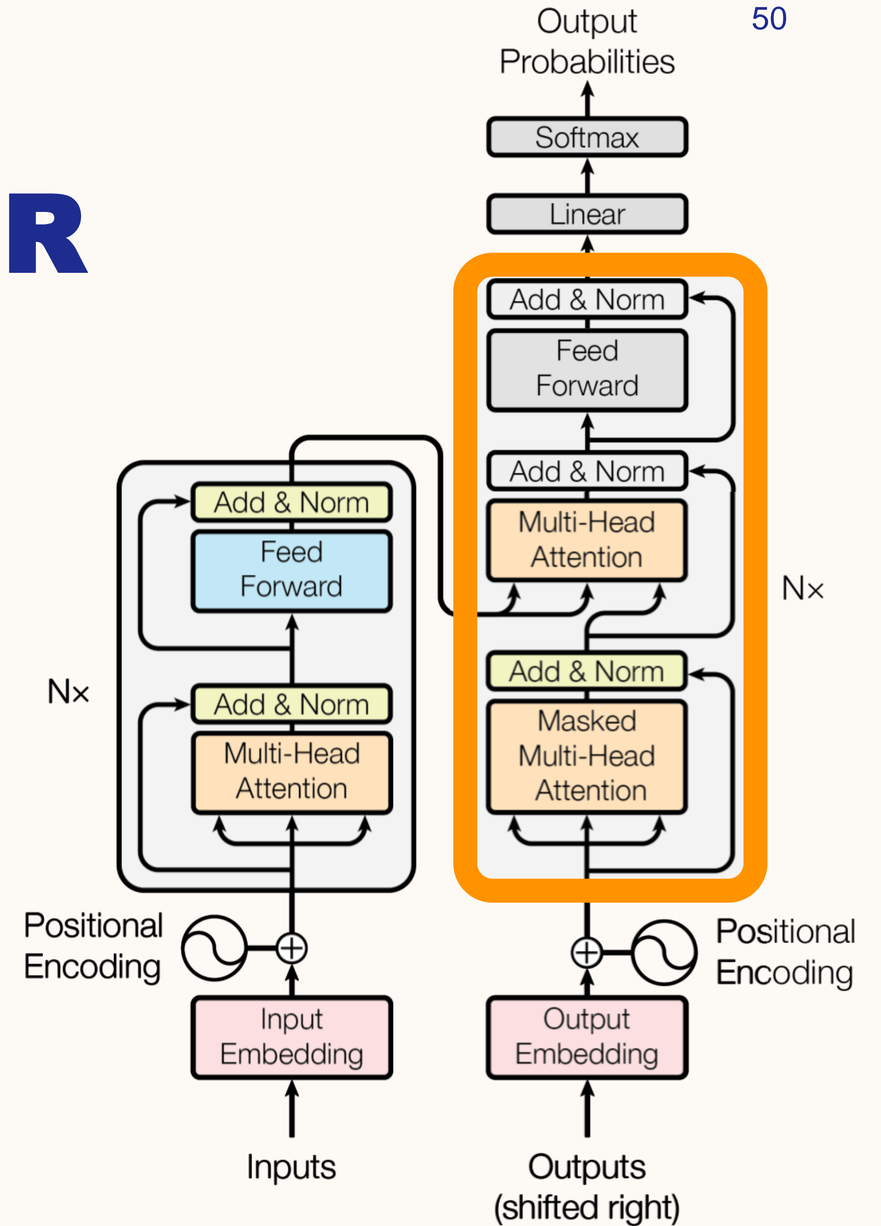


THE DECODER

Masked Multi-Head Attention $= \text{MaskedMultiHeadAtt}(\mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}, \mathbf{H}_i^{dec})$

Add & Norm $= \text{LayerNorm}($ Multi-Head Attention $+ \mathbf{H}_i^{dec})$

Enc-Dec Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{dec}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc})$



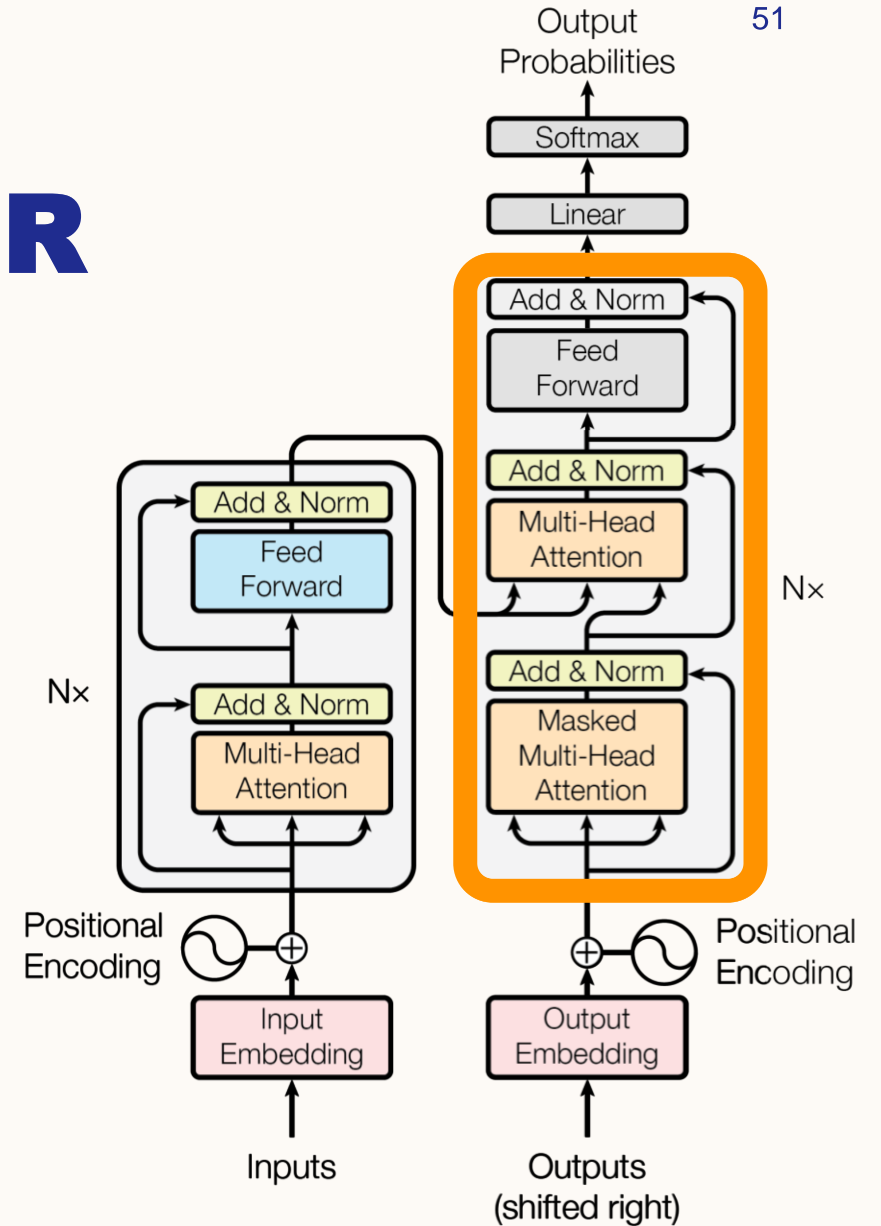
THE DECODER

Masked Multi-Head Attention = MaskedMultiHeadAtt($\mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}$)

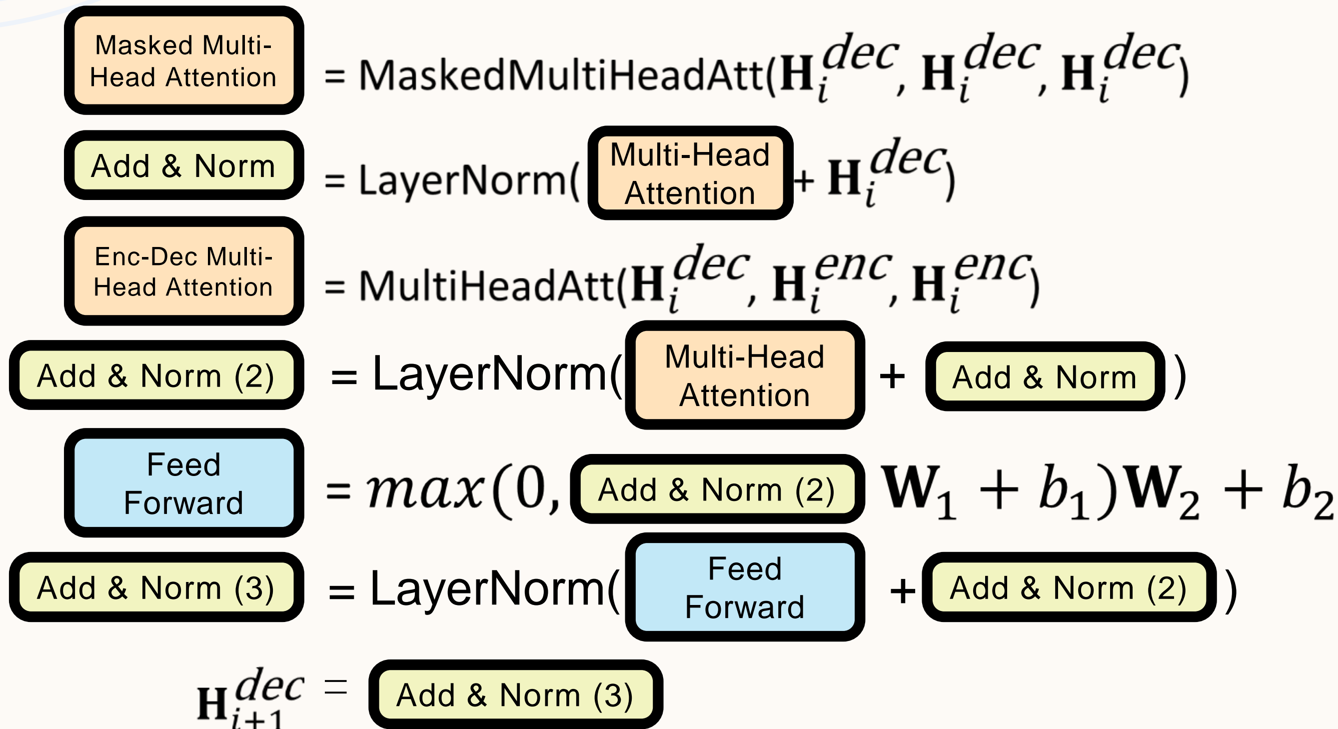
Add & Norm = LayerNorm(Multi-Head Attention + $\mathbf{H}_i^{dec}$)

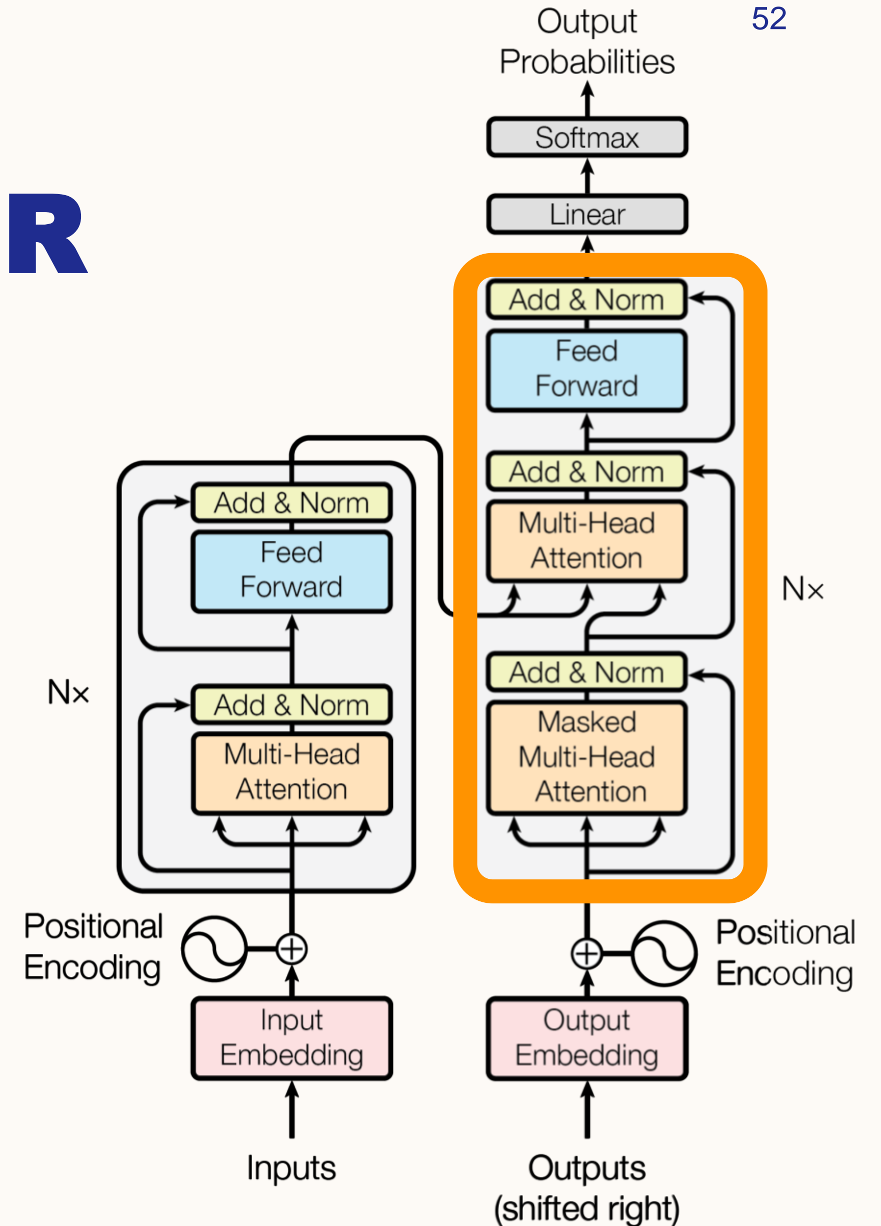
Enc-Dec Multi-Head Attention = MultiHeadAtt($\mathbf{H}_i^{dec}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}$)

Add & Norm (2) = LayerNorm(Multi-Head Attention + Add & Norm)



THE DECODER





STRENGTHS OF THE TRANSFORMER ARCHITECTURE

- Training is easily parallelizable
  - Larger models can be trained efficiently.
- Does not “forget” information from earlier in the sequence.
  - Any position can attend to any position.

ETHICS OF ML

EXPLAINABILITY AND INTERPRETABILITY

- How clear is our agent's decision making? Is it transparent or is it a black box?
- Can we make changes to the algorithm to make its decisions more explainable?
- Can we develop tools that make the algorithm's decisions easier to interpret?

INEQUALITY

- Who has access to this AI agent?
  - Could this create new inequality between groups that have access and do not have access?
- Is this system reinforcing existing structures that create inequality?
  - If yes, is there regulation for this technology that can prevent this?

JOB DISPLACEMENT

- Will this algorithm displace human workers?
    - If yes, is there a plan in place to help those displaced workers?
- Will this algorithm/agent create new jobs? Who will benefit?