BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Encoder - Decoder

Dr YVK Ravi Kumar

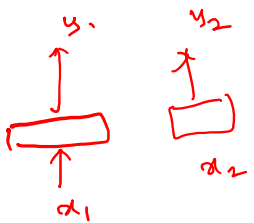
Sequence to Sequence Learning with Neural Networks

Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

In Machine translation, the input sequence and output sequence may not be of same length



RNN / LSTM / GRU

Sec 2 seq

→ fixed length

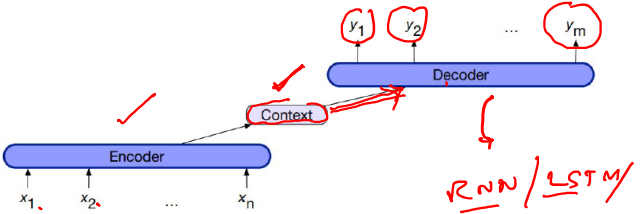
→ Vanishing gradient

Encoder - Decoder

- **Goal:** Develop an architecture capable of generating *contextually appropriate*, *arbitrary length*, output sequences

- **Applications**:
  - Machine translation
  - Summarization
  - Question answering
  - Dialogue modeling.

RNN - ?
LSTM - ?



Encoder - Decoder

Encoder:

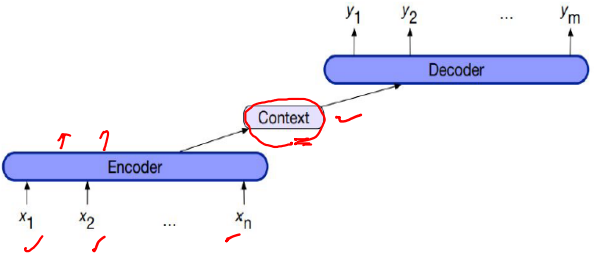
- Input sequence ($x_{1...n}$) → Sequence of contextualized representations, ($h_{1...n}$)
- Ex: LSTM, CNN, Transformers etc.

Context:

- c, a function of ($h_{1...n}$)

Decoder:

- c → arbitrary length sequence of hidden states ($h_{1...m}$) → sequence of output states ($y_{1...m}$)



Encoder - Decoder

Encoder:

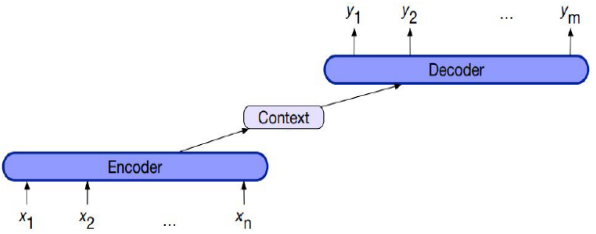
- Input sequence ($x_{1...n}$) → Sequence of contextualized representations, ($h_{1...n}$)
- Ex: LSTM, CNN, Transformers etc.

Context:

- c, a function of ($h_{1...n}$)

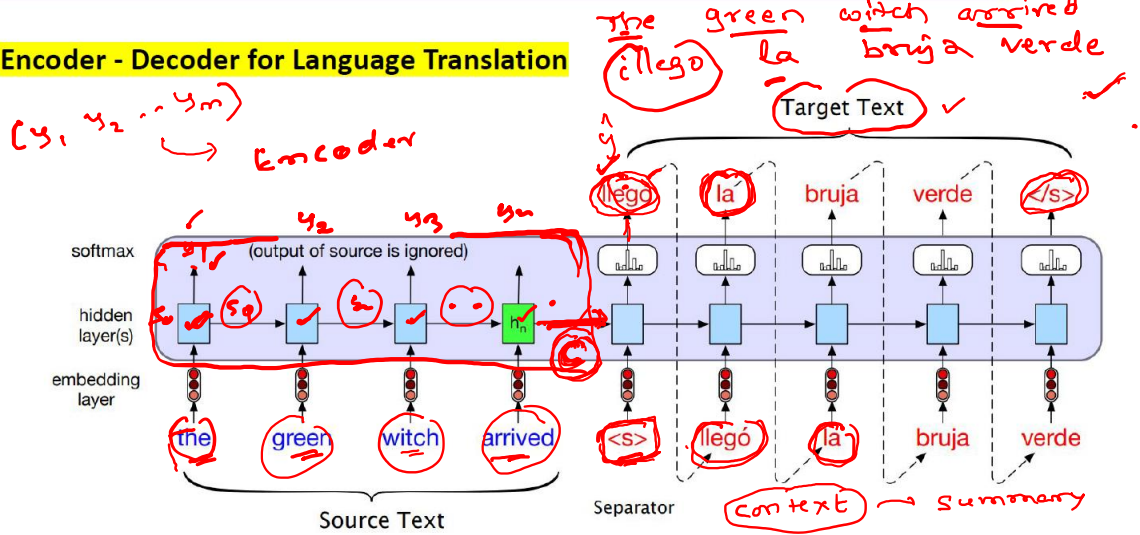
Decoder:

- c → arbitrary length sequence of hidden states ($h_{1...m}$) → sequence of output states ($y_{1...m}$)



Encoder - Decoder

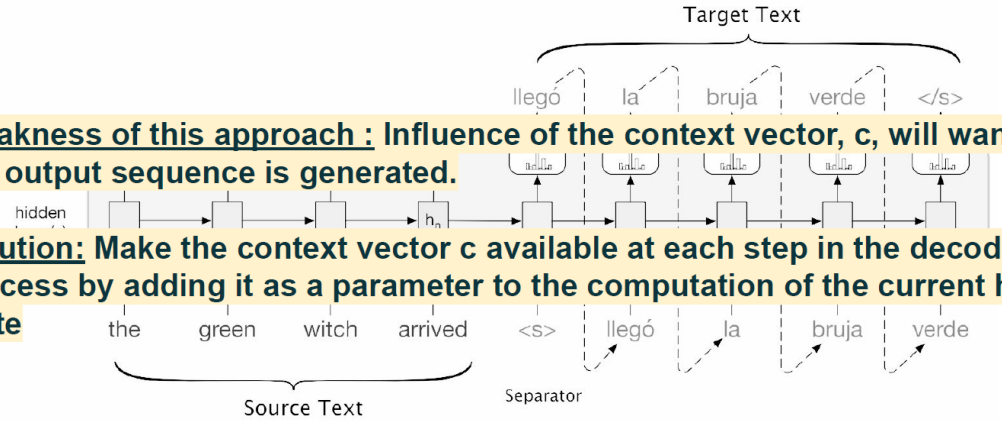
Encoder - Decoder for Language Translation



Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

Encoder - Decoder

Encoder - Decoder for Language Translation

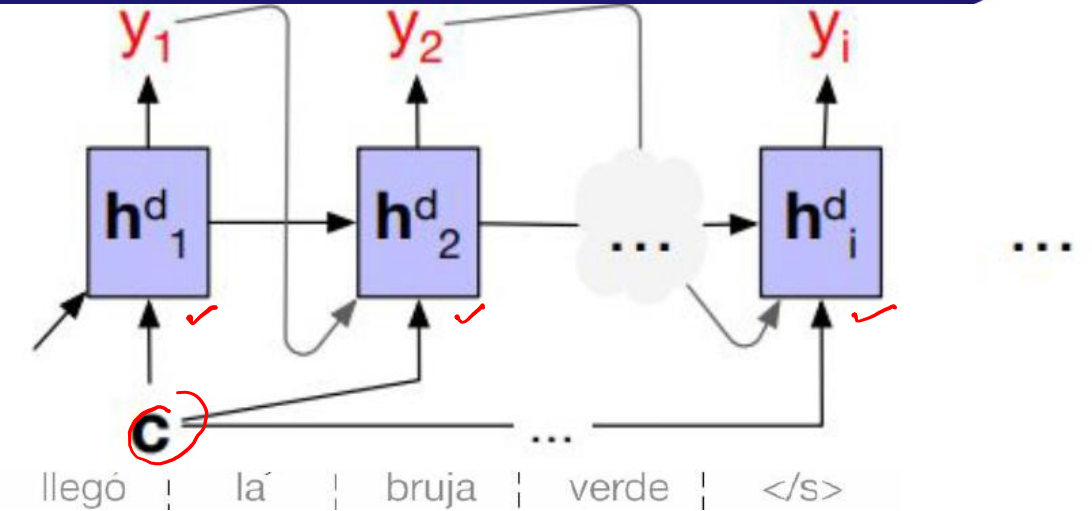


Weakness of this approach : Influence of the context vector, c, will wane as the output sequence is generated.

Solution: Make the context vector c available at each step in the decoding process by adding it as a parameter to the computation of the current hidden state

Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

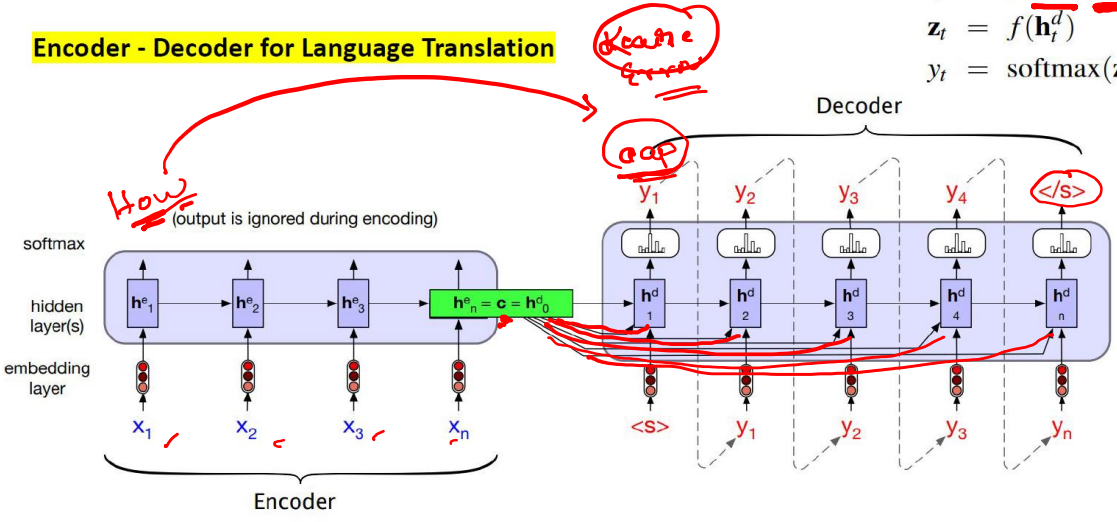
Encoder - Decoder



Encoder - Decoder

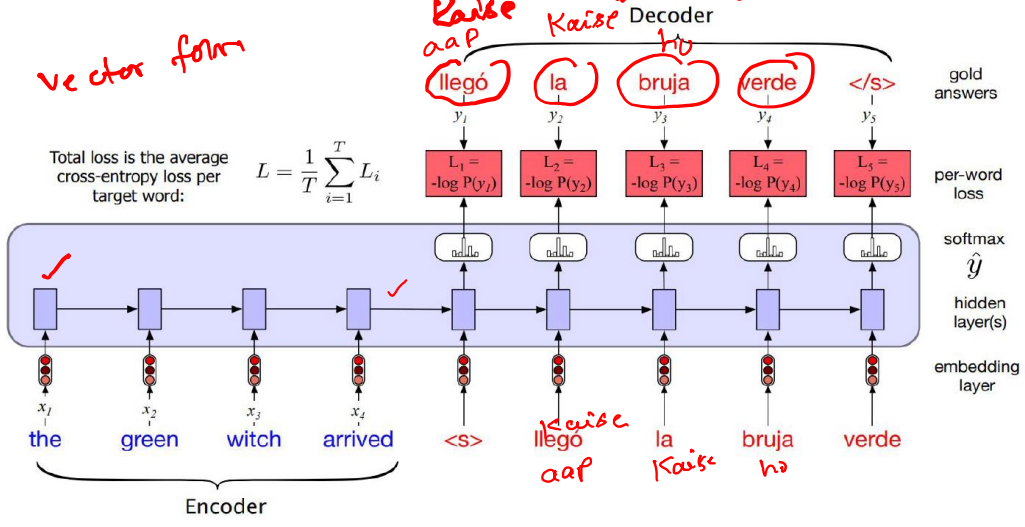
$$
\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t)
\end{aligned}
$$

Encoder - Decoder for Language Translation



Encoder - Decoder

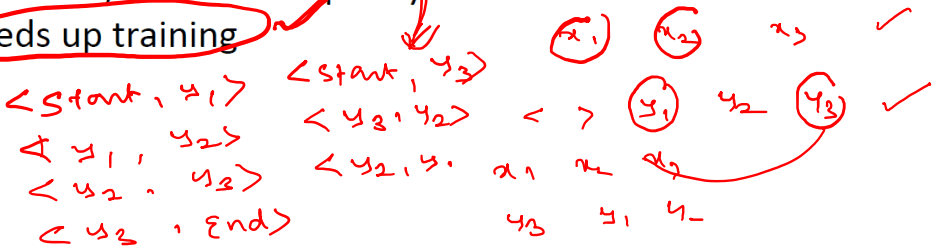
Training



Encoder - Decoder

Teaching Forcing

- Force the system to use the gold target token from training as the next input x_{t+1}, rather than allowing it to rely on the (possibly erroneous) decoder output ˆyt .
- Speeds up training



Encoder - Decoder

- The encoder – decoder architecture improved the performance over seq2seq model
- Its performance is good over short sized sentences , but not so in case of larger sized sentences
- It assumes that the entire input sequence is equally important for all the words in the output sequence

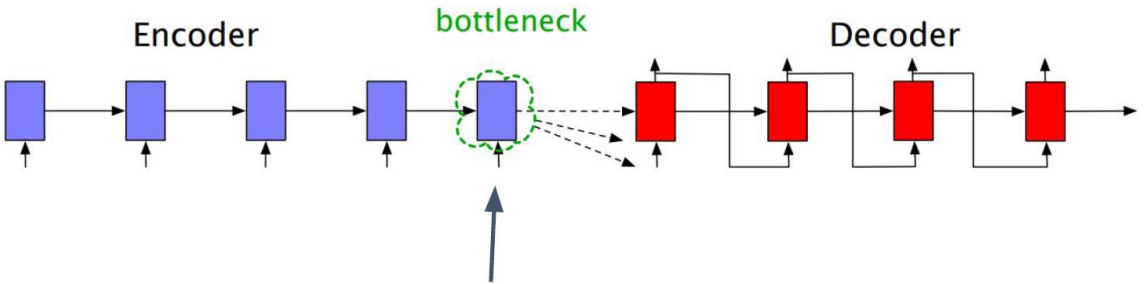
Attention Mechanism

- In Encoder length of a sentence is a challenge
- In decoder static representation is a problem

Attention Mechanism

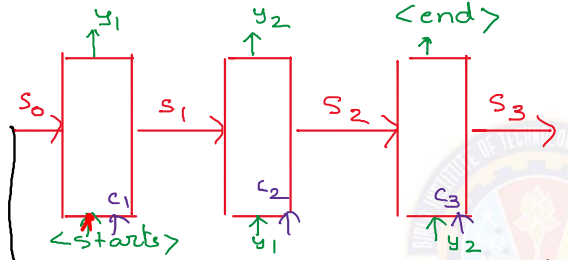
- The attention mechanism is at the core of the transformer architecture
- In a transformer, the attention mechanism enables the model to assign different *attention weights/scores* to different tokens in a sequence.
- This way, the model can give more importance to relevant information, ignore irrelevant information, and effectively capture long-range dependencies in the data.

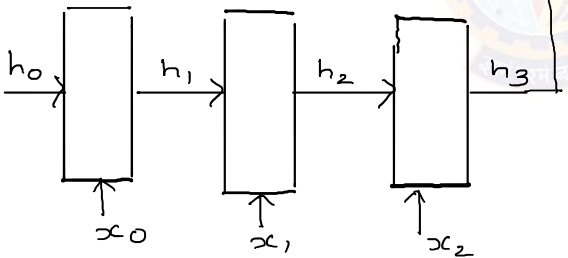
Attention !



In an encoder-decoder arch, the final hidden state acts as a bottleneck:

- It must represent absolutely everything about the meaning of the source text
- The only thing the decoder knows about the source text is what's in this context vector





Vanila Encoder-decoder

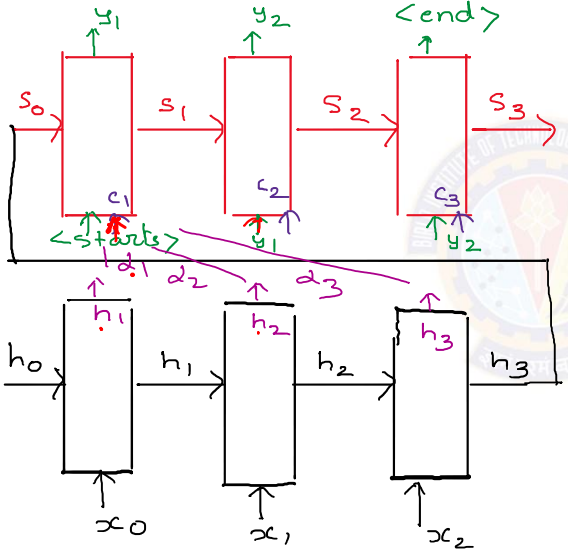
at i, input

$[y_{i-1}, s_{i-1}]$ ✓

Attention:

$[y_{i-1}, s_{i-1}, c_i]$

c_i = attention ✓



$$c_1 = \alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3$$

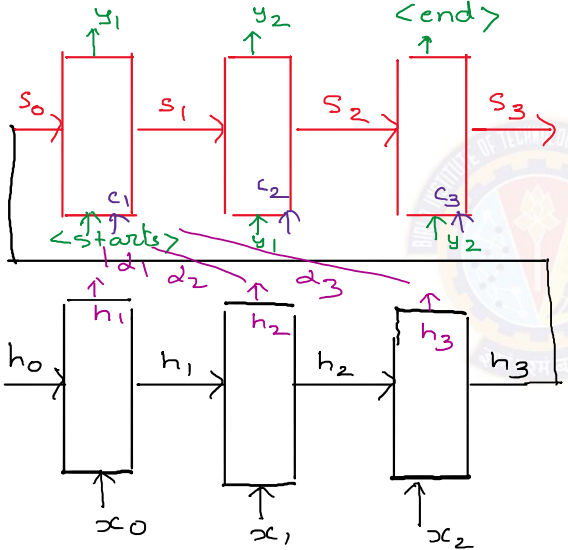
$$c_1 = \alpha_{11} h_1 + \alpha_{12} h_2 + \alpha_{13} h_3$$

$$c_2 = \alpha_{21} h_1 + \alpha_{22} h_2 + \alpha_{23} h_3$$

$$c_i = \sum \alpha_{ij} h_j$$

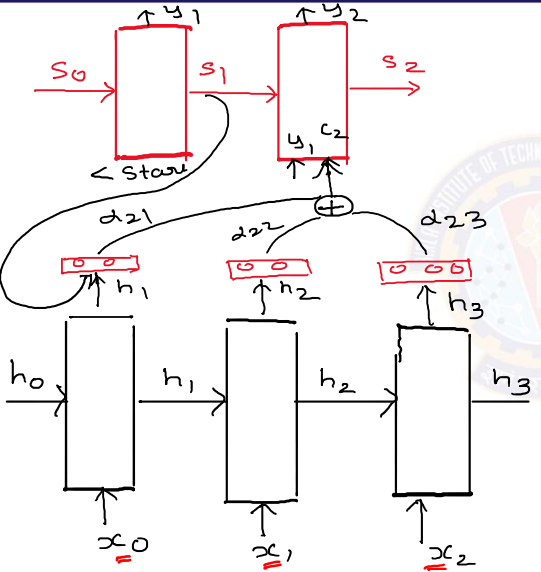
Encoder time step j

decoder time step i



$$\alpha_{ij} = f(h_j, s_{i-1})$$

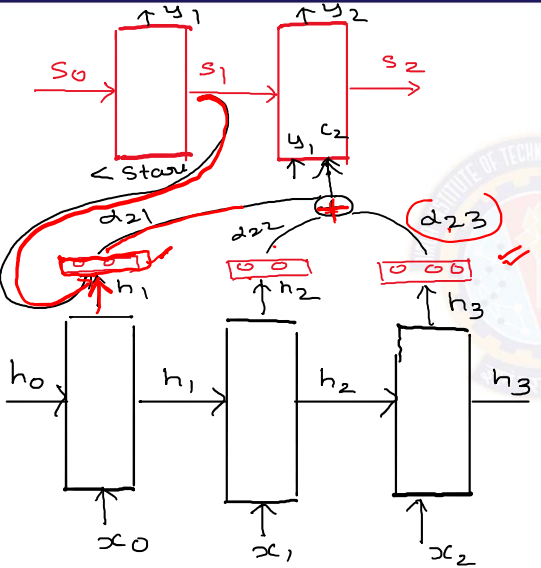
$$\alpha_{21} = f(h_1, s_1)$$



$$c_2 = \alpha_{21} h_1 + \alpha_{22} h_2 + \alpha_{23} h_3$$

Finding α'

- Bahdanau attention
- Luong attention



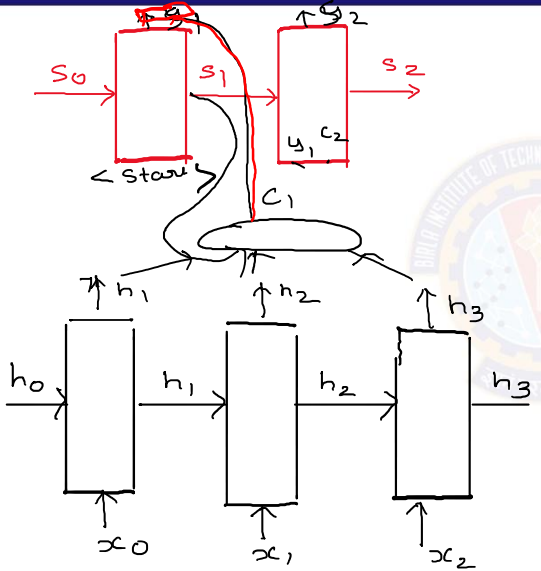
Bahadanau attention

$$c_i = \sum \alpha_{ij} h_j$$

$$\alpha_{ij} = f(s_{i-1}, h_j)$$

function f is ANN

Training → models

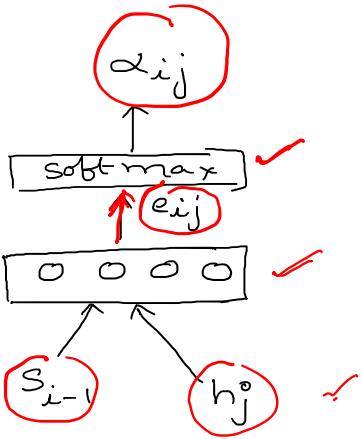


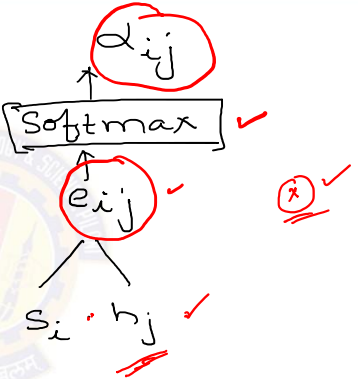
Luong attention

$$\alpha_{ij} = f(s_i, h_j)$$

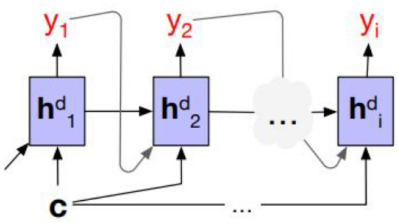
where f is the vector dot product of s_i^T and h_j

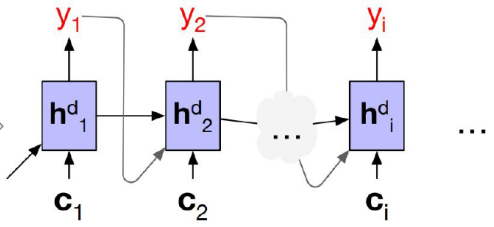
Luong →


α_{ij}
softmax
e_{ij}
s_{i-1}
h_j


α_{ij}
Softmax
e_{ij}
$s_i \cdot h_j$

Attention !





Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

With attention, decoder to sees a different, dynamic, context, which is a function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

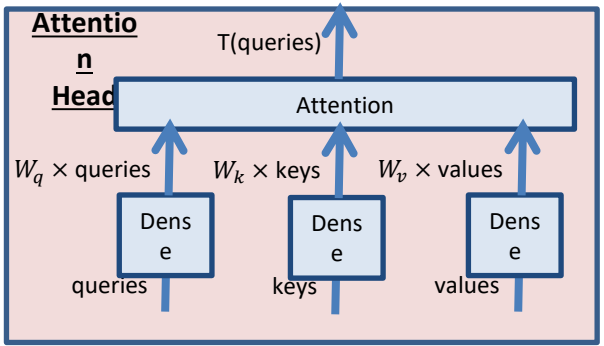
With attention, decoder gets information from all the hidden states of the encoder, not just the last hidden state of the encoder

Each context vector is obtained by taking a weighted sum of all the encoder hidden states.

The weights focus on ('attend to') a particular part of the source text that is relevant for the token the decoder is currently producing

Review: Attention Head

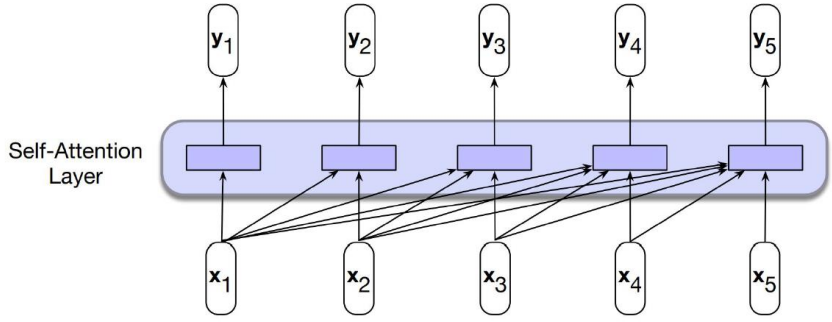
- Input: queries, keys, and values.
  - For self-attention, queries = keys = values.
  - For English-to-Spanish translation, sometimes queries != keys.
- Output: the “transformed“ version of the queries.



Self Attention

- Attention ⇒ Ability to compare an item of interest to a collection of other items in a way that reveals their relevance in the current context.
- Self-attention ⇒
  - > Set of ***comparisons*** are to other elements ***within a given sequence***
  - > Use these comparisons to compute an output for the current input

Self-Attention | Transformers



In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one.

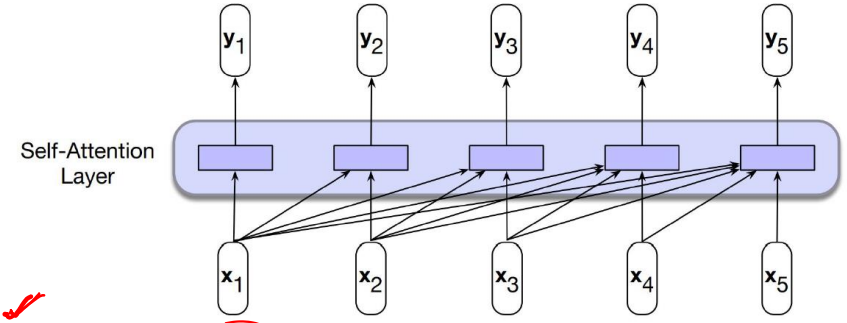
Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Self Attention

- During self-attention, the tokens *within the input sequence* are compared to each other, computing *attention weights.*
- A high attention weight for a pair of two tokens indicates that they are syntactically or semantically related. Thus high weights indicate important tokens that the model should "attend" to.
- In the example sentence "*The cat watches the dog run fast*", "run" and "fast" could be two tokens where we would expect high attention weights since they are semantically related. In contrast, "the" and "fast" should lead to low attention weights.

Attention → self attention

Self-Attention | Transformers



$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \le i$$

$$= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \quad \forall j \le i$$

$$\mathbf{y}_i = \sum_{j \le i} \alpha_{ij} \mathbf{x}_j$$

Self Attention

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

Query, Q

As the current focus of attention when being compared to all of the other preceding inputs.

Key, K

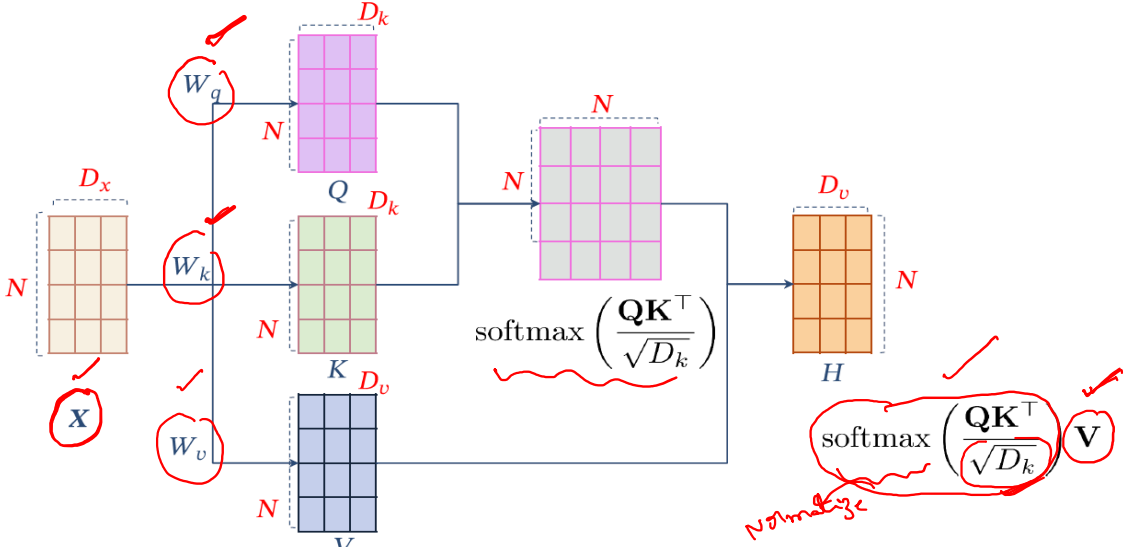
In its role as a preceding input being compared to the current focus of attention.

Value, V

As a value used to compute the output for the current focus of attention

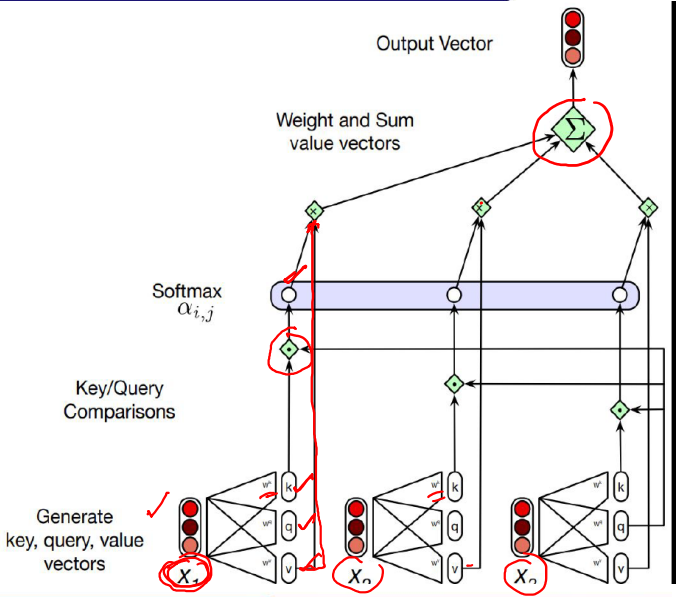
Three different roles each x_i (input embedding) , in the computation of self attention

Scaled Dot product attention



Self Attention

- Each output, y_i , is computed independently
- Entire process can be parallelized



Calculating the value of y3, the third element of a sequence using causal (left-to-right) self-attention

Self Attention

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

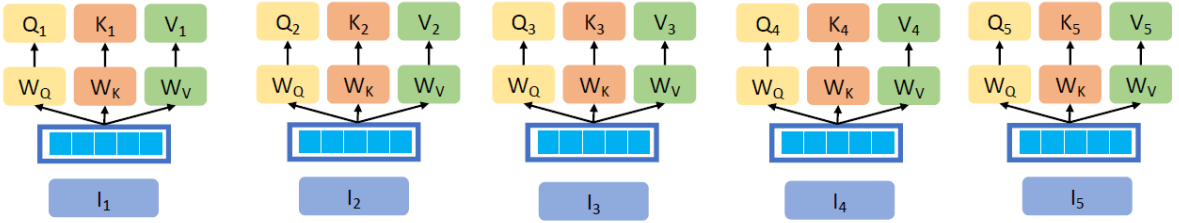
$\mathbf{QK}^\top$ Matrix

N

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

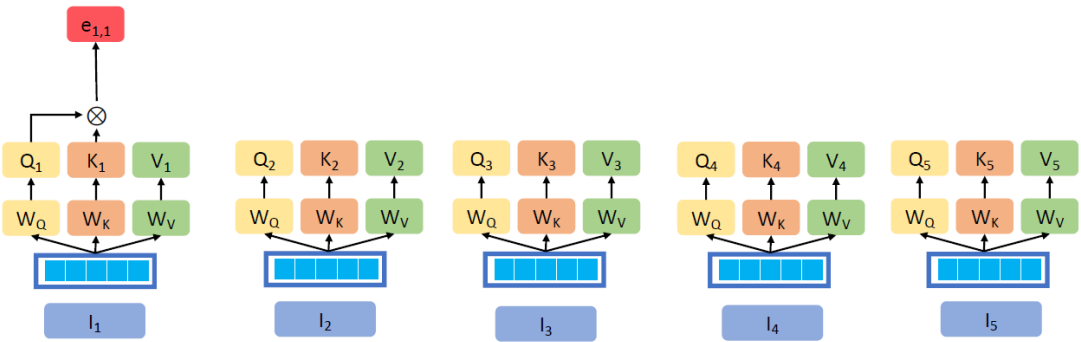
N

Note: Upper-triangle portion of the comparisons matrix zeroed out (set to −∞, which the softmax will turn to zero)


Q1 K1 V1
WQ WK WV
I1
I
Q2 K2 V2
WQ WK WV
I2
ate
Q3 K3 V3
WQ WK WV
I3
an
Q4 K4 V4
WQ WK WV
I4
apple
Q5 K5 V5
WQ WK WV
I5
<eos>

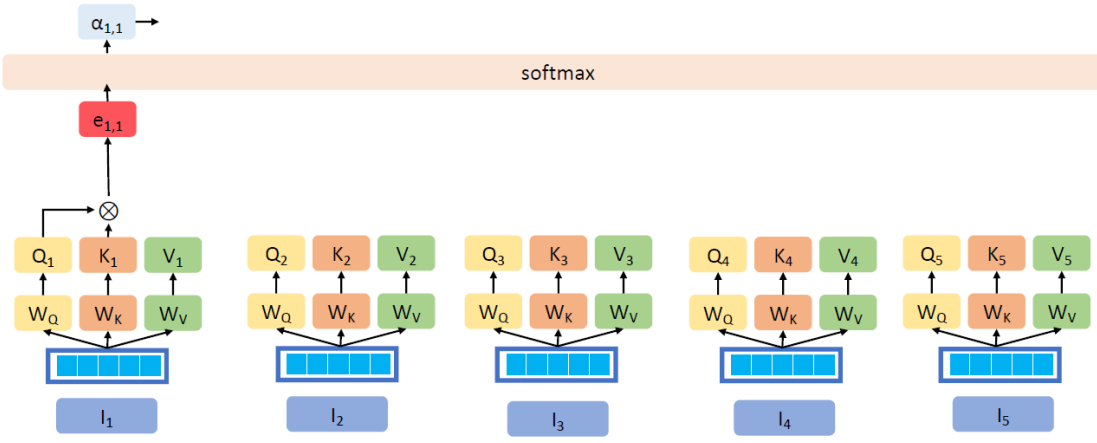
Dimensions across QKV have been dropped for brevity

Attention

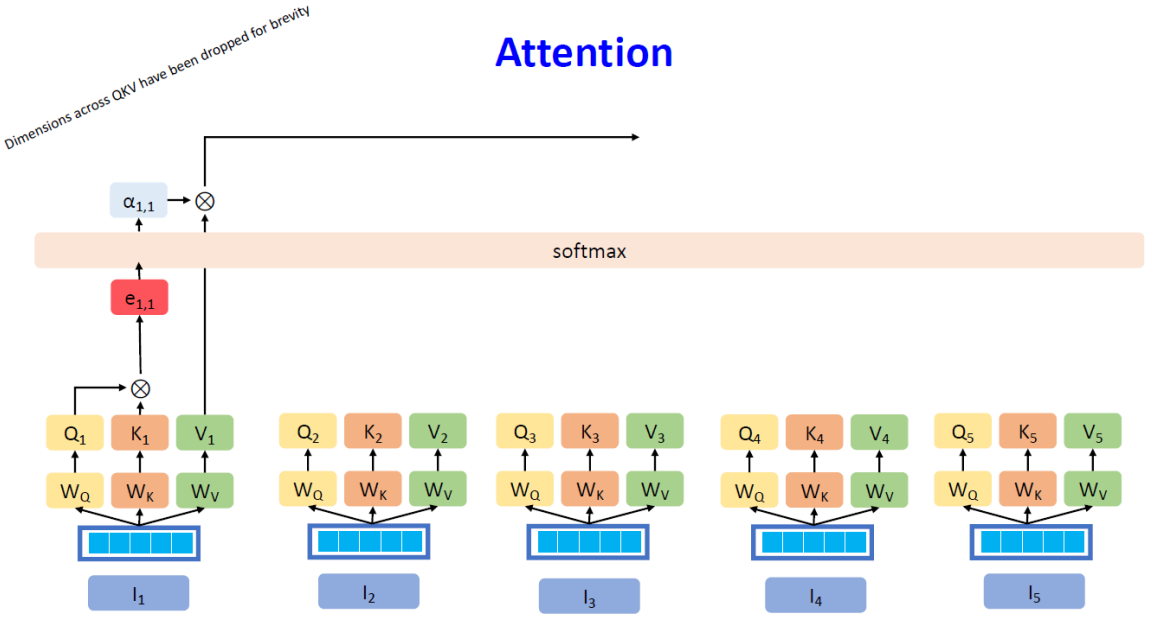


Attention

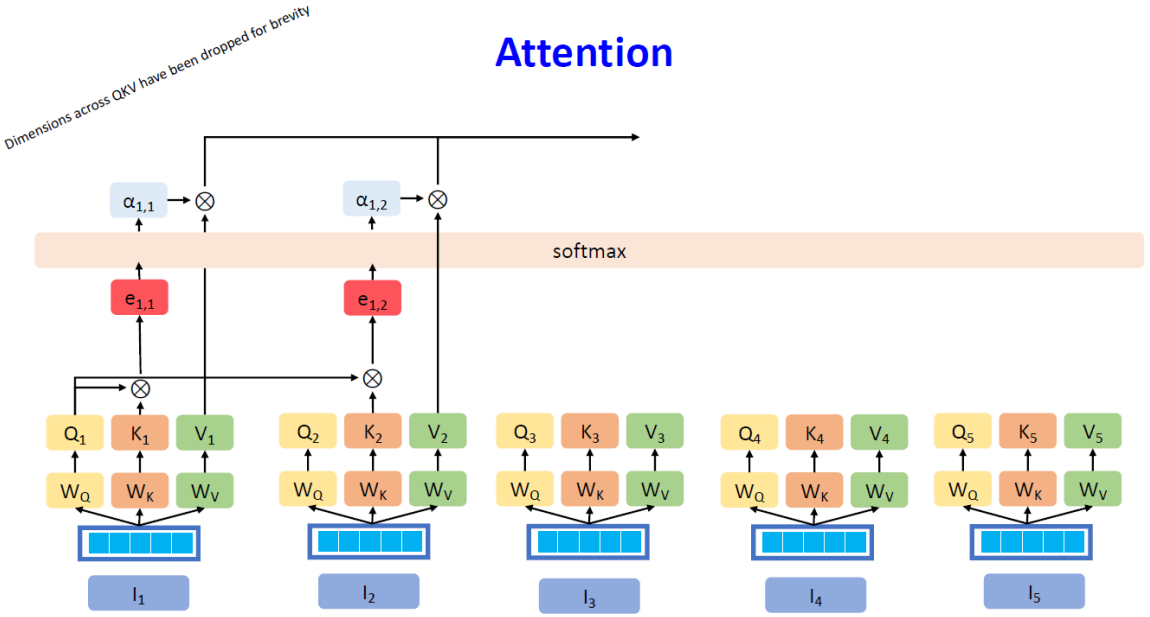
Dimensions across QKV have been dropped for brevity



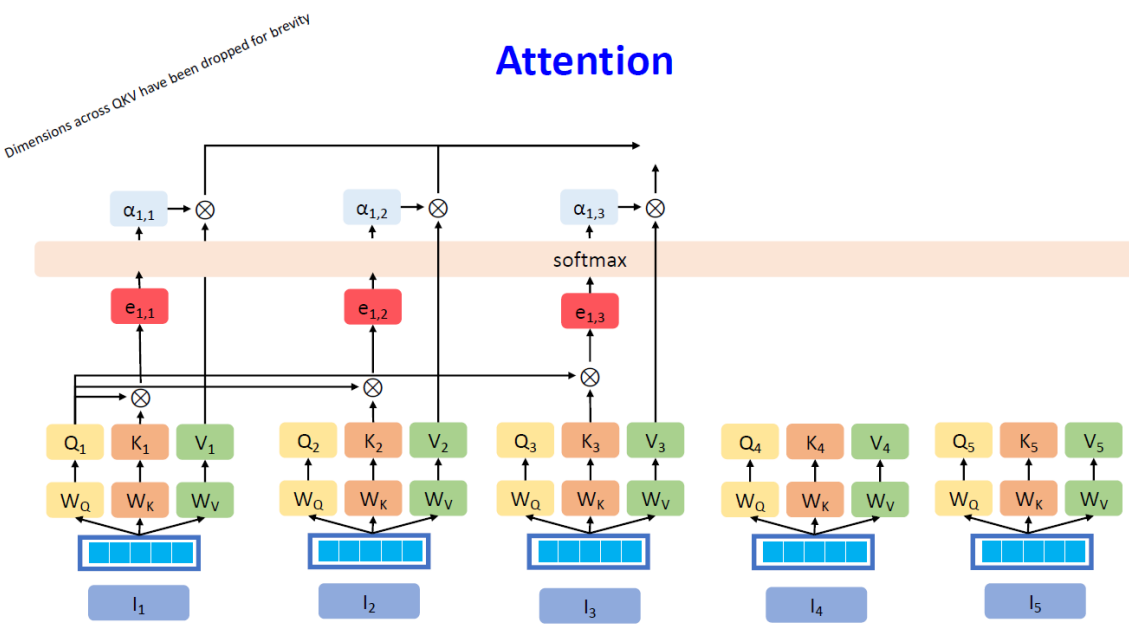
Attention


Dimensions across QKV have been dropped for brevity
$\alpha_{1,1}$
softmax
$e_{1,1}$
Q_1 K_1 V_1
Q_2 K_2 V_2
Q_3 K_3 V_3
Q_4 K_4 V_4
Q_5 K_5 V_5
W_Q W_K W_V
I_1 I_2 I_3 I_4 I_5
I
ate
an
apple
<eos>

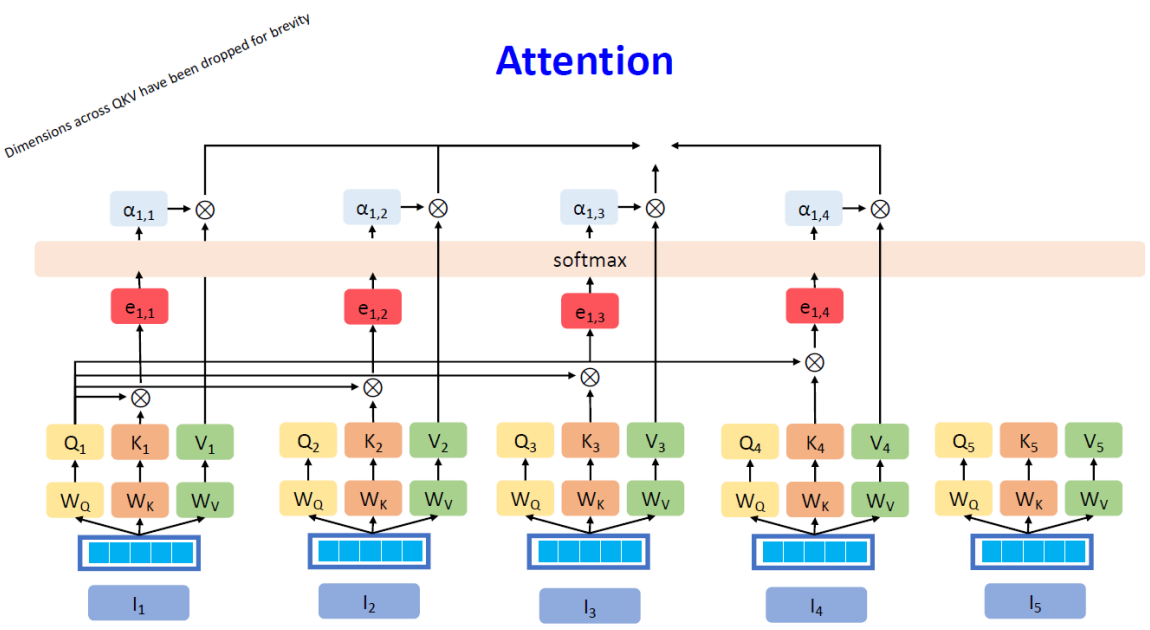
Attention


Dimensions across QKV have been dropped for brevity
α1,1
α1,2
softmax
e1,1
e1,2
Q1
K1
V1
WQ
WK
WV
Q2
K2
V2
Q3
K3
V3
Q4
K4
V4
Q5
K5
V5
I1
I2
I3
I4
I5
I
ate
an
apple
<eos>

Attention

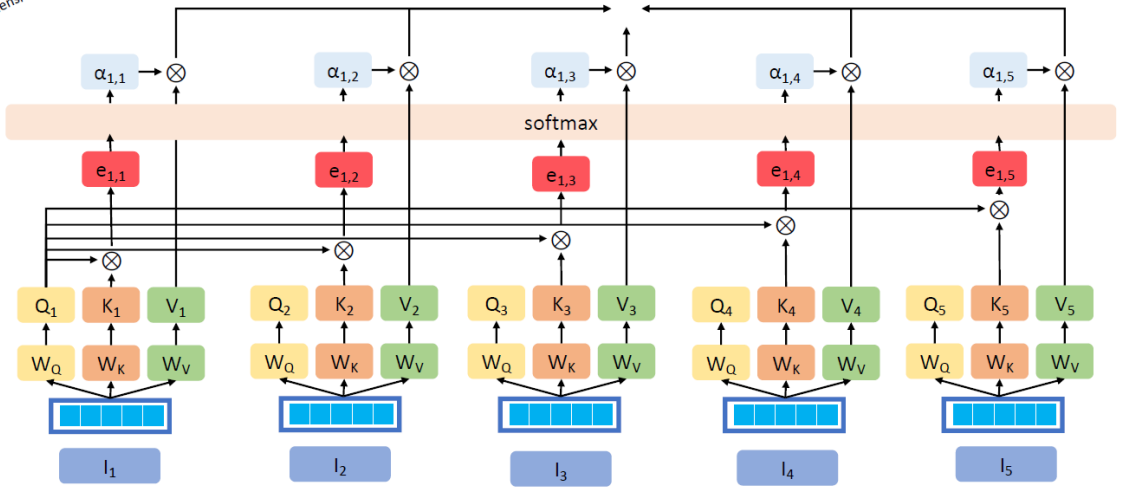

Dimensions across QKV have been dropped for brevity
α1,1
α1,2
α1,3
softmax
e1,1
e1,2
e1,3
Q1
K1
V1
Q2
K2
V2
Q3
K3
V3
Q4
K4
V4
Q5
K5
V5
WQ
WK
WV
I1
I2
I3
I4
I5
I
ate
an
apple
<eos>

Attention

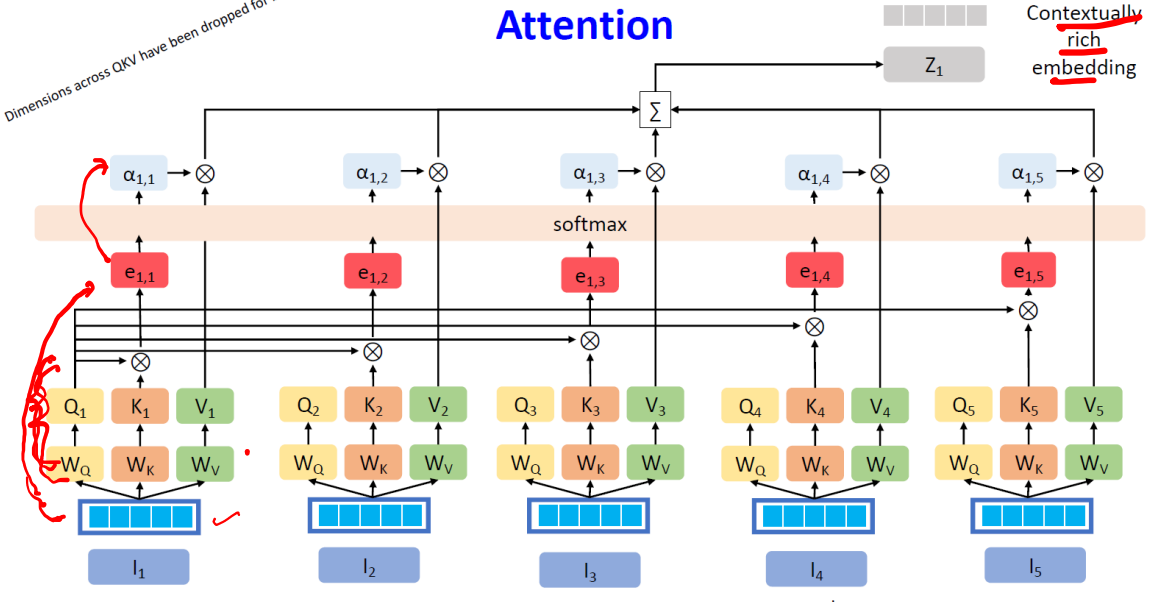

Dimensions across QKV have been dropped for brevity
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
softmax
$e_{1,1}$
$e_{1,2}$
$e_{1,3}$
$e_{1,4}$
Q_1 K_1 V_1
Q_2 K_2 V_2
Q_3 K_3 V_3
Q_4 K_4 V_4
Q_5 K_5 V_5
W_Q W_K W_V
I_1 I_2 I_3 I_4 I_5
I
ate
an
apple
<eos>

Attention

Dimensions across QKV have been dropped for brevity



Attention


Dimensions across QKV have been dropped for brevity
Contextually rich embedding
Z_1
$\sum$
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
$\alpha_{1,5}$
softmax
$e_{1,1}$
$e_{1,2}$
$e_{1,3}$
$e_{1,4}$
$e_{1,5}$
Q_1 K_1 V_1
Q_2 K_2 V_2
Q_3 K_3 V_3
Q_4 K_4 V_4
Q_5 K_5 V_5
W_Q W_K W_V
I_1 I_2 I_3 I_4 I_5
I
ate
an
apple
<eos>

Attention Weights calculation

- "*The cat watches the dog run fast.*"
- Let's assume "cat" is embedded as vector (0.6, 0.3, 0.8)
  "dog" as vector (0.5, 0.1, 0.9),

i.e., we have *d=3* dimensional embedding vectors.

- The dot product of the two vectors here is 0.6*0.5 + 0.3*0.1 + 0.8*0.9 = 1.05. (Note that the product is larger for more similar vectors, which are closer in the embedding space)
- The result of the multiplication is scaled by dividing by $\sqrt{d}$.
  that is 1.05 / √3 = 0.606.
- If *d* is large, the dot product may grow very large. In this case, scaling helps reducing the magnitude and ensuring suitable inputs to the softmax function.

Attention Weights calculation

- Next, the scaled product is fed into a softmax function, which outputs the attention weights. The returned attention weights of all pairs of tokens are between 0 and 1 and sum up to 1.

In our example, assume we just have two other token pairs with scaled products of 0.2 and -0.8 in addition to the 0.606.
In this case, softmax([0.606, 0.2, -0.8]) = [0.523, 0.348, 0.128]),
i.e., we have the highest attention weights for our two tokens “cat” and “dog” and smaller weights for the other two pairs.

We can also mask (=ignore) certain positions by setting them to negative infinity, where softmax(-∞)=0. This is used in the decoder during training.

Attention Weights calculation

The computed attention weights are then multiplied with the values V.

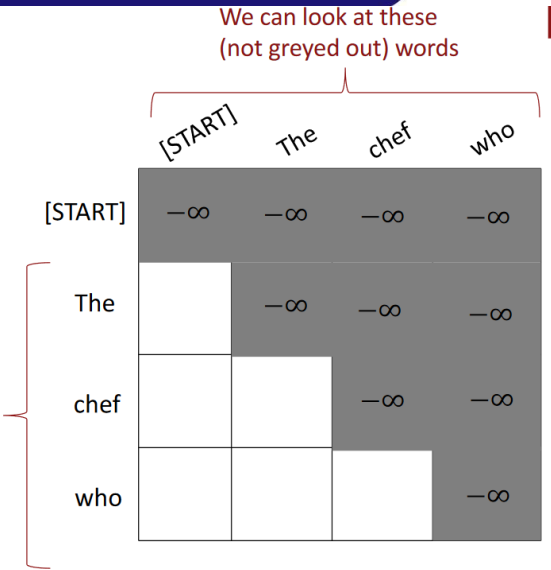
In our example, “dog” is used as value and its embedding vector (0.5, 0.1, 0.9) is multiplied with attention weight 0.523, resulting in (0.262, 0.052, 0.471).

Other token pairs where Q and K are less compatible may have lower attention weights closer to zero.

Masked Self attention

- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$



Cross Attention

- *Self- and cross-attention only differ in which tokens are given as inputs Q, K, and V.*
- In **self-attention**, the same tokens are used for all three inputs, i.e., each token is used for Q, K, and V.
- In **cross-attention**, the same tokens are given to K and V, but different tokens are used for Q.

Multi head attention

- Different words in a sentence can relate to each other in many different ways simultaneously

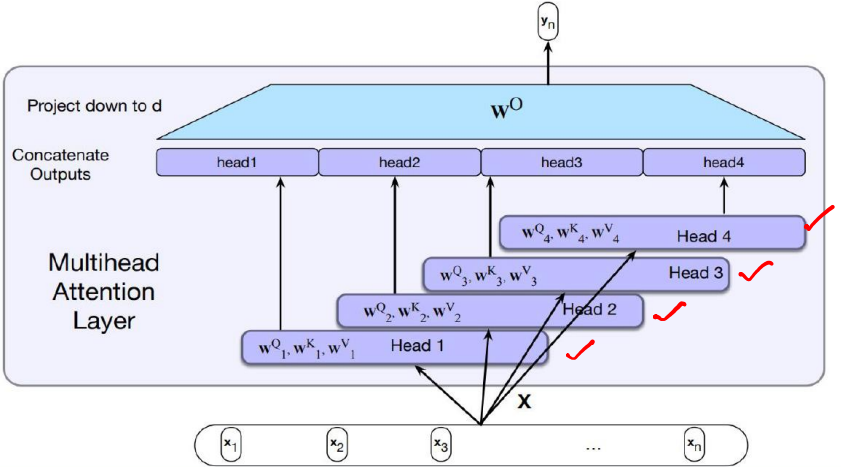
  >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.

- Multihead self-attention layers

  >> Heads ⇒ sets of self-attention layers, that reside in parallel layers at the same depth in a model, each with its own set of parameters.

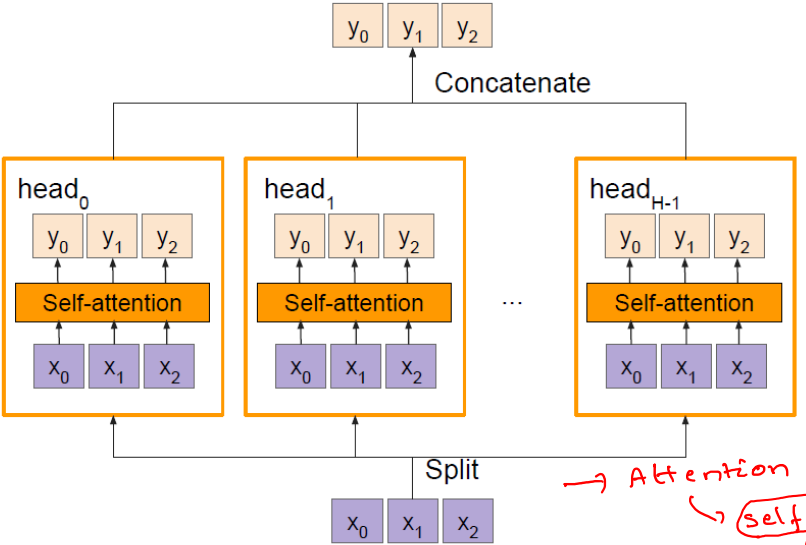
  >> Each head learn different aspects of the relationships that exist among inputs at the same level of abstraction

Multi head attention



Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices. The outputs from each of the layers are concatenated and then projected down to d, thus producing an output of the same size as the input so layers can be stacked.

Multiple self-attention heads in parallel

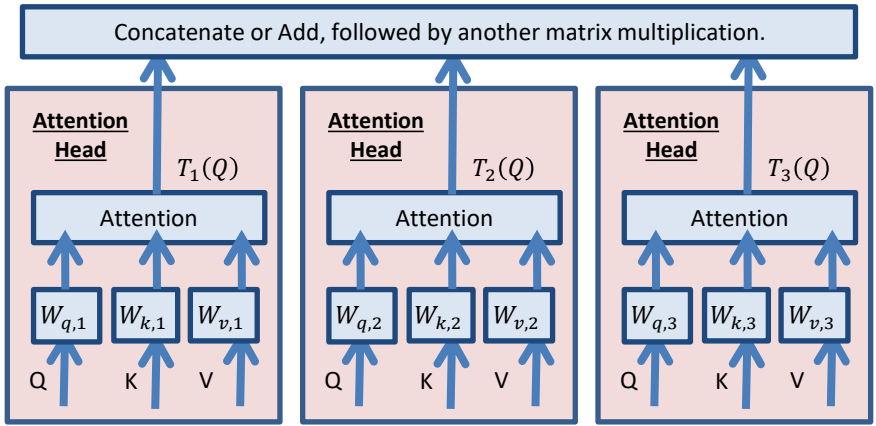


→ Attention

↳ self attention

Review: Multi-Head Attention

- Multiple attention heads can be used instead of a single one.

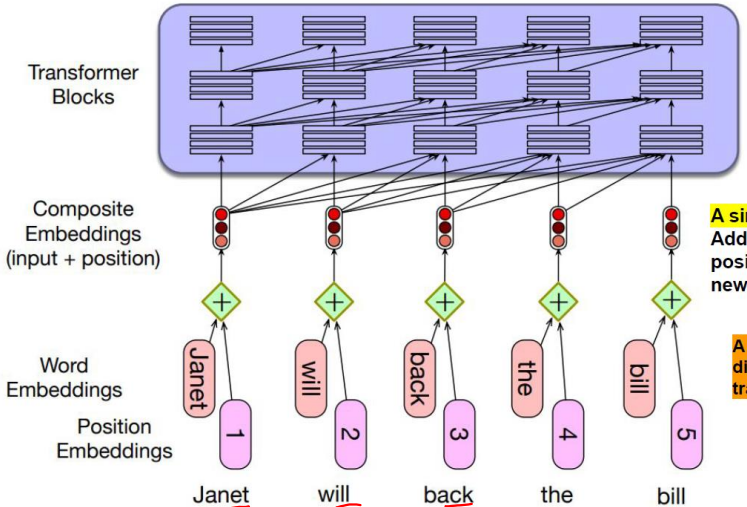


Positional Embeddings

512

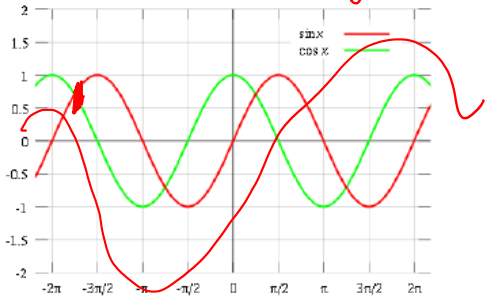
- **positional encoding** indicates the absolute and relative position of the tokens in the sequence.
- For example, in "*The cat watches the dog run fast.*", if the position of "cat" and "dog" were interchanged, it would alter subject and object and thus the meaning of the sentence.

Positional Embeddings



A simple way to model position:
Add an embedding representation of the absolute position to the input word embedding to produce a new embedding of the same dimensionality.

A combination of sine and cosine functions with differing frequencies was used in the original transformer work.



sin / cos

Transformer positional encoding

$$PE_{(pos,2i)} = \sin(\frac{pos}{10000^{2i/d_{model}}})$$

$$PE_{(pos,2i+1)} = \cos(\frac{pos}{10000^{2i/d_{model}}})$$

Positional encoding is a 512d vector
i = a particular dimension of this vector
pos = dimension of the word
d_model = 512

[sin( ), cos( )]

[si]

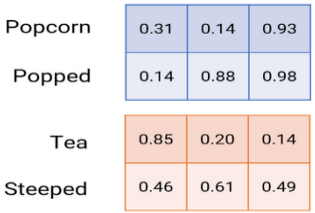
Transformer positional encoding

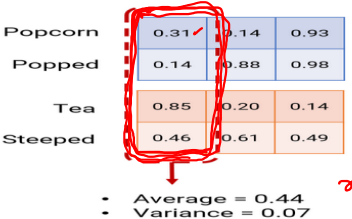
$$PE_{(pos,2i)} = \sin(\frac{pos}{10000^{2i/d_{model}}})$$

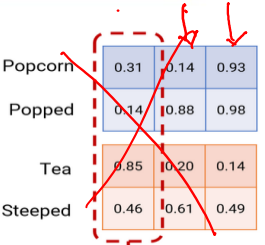
$$PE_{(pos,2i+1)} = \cos(\frac{pos}{10000^{2i/d_{model}}})$$

Positional encoding is a 512d vector
i = a particular dimension of this vector
pos = dimension of the word
d_model = 512

Normalization(Batch and Layer)



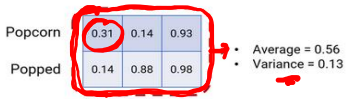


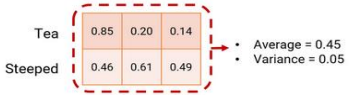


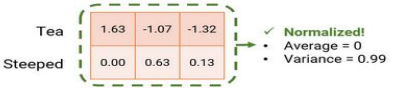
Batch

Mean: 0
Var: 1

$\frac{x-\mu}{\sigma}$







$\frac{x-\mu}{\sigma}$

Layer normalized

Transformer

→ Self attention
→ Multi-head attention
→ positional embeds
→ Normalization

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

Transformer

- Used for modeling long dependencies between input sequence elements
- Supports parallel processing of sequence as compared to RNN (e.g. LSTM)
- Allows processing multiple modalities

  (e.g., images, videos, text and speech) using similar processing blocks
- Typically, pre-trained using pretext tasks on largescale (unlabeled) datasets
- Demonstrates excellent scalability to very large networks and huge datasets.

Transformers

- Recognition tasks

  (e.g., image classification, object detection, action recognition, and segmentation),

- Generative modeling, multi-modal tasks

  (e.g., visual-question answering, visual reasoning, and visual grounding),

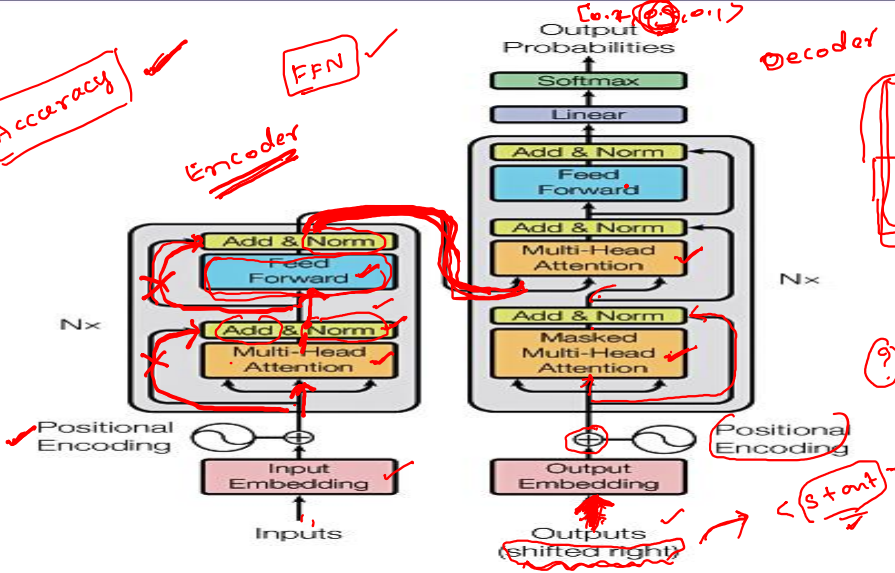
- Video processing

  (e.g., activity recognition, video forecasting),

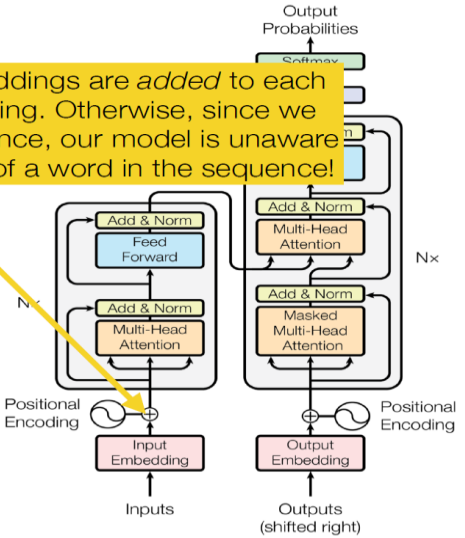
- Low-level vision

  (e.g., image super-resolution, image enhancement , and colorization)

Transformer

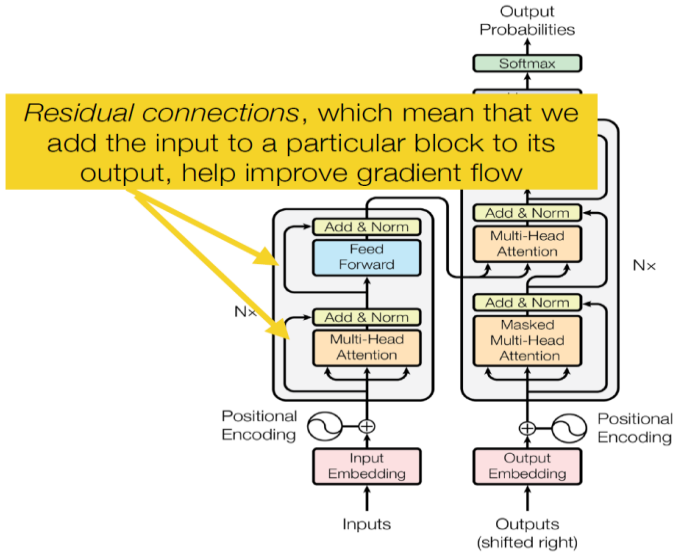


Position embeddings are *added* to each word embedding. Otherwise, since we have no recurrence, our model is unaware of the position of a word in the sequence!

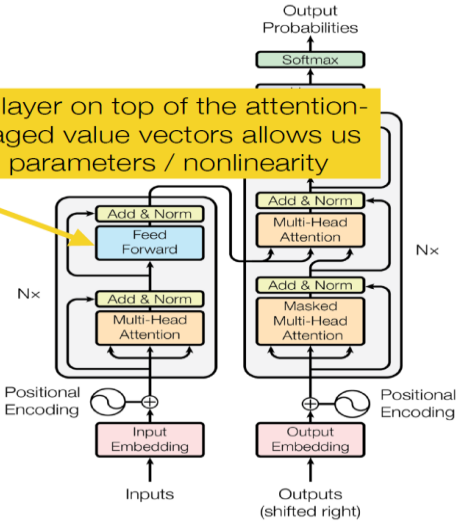


3.40 3.5

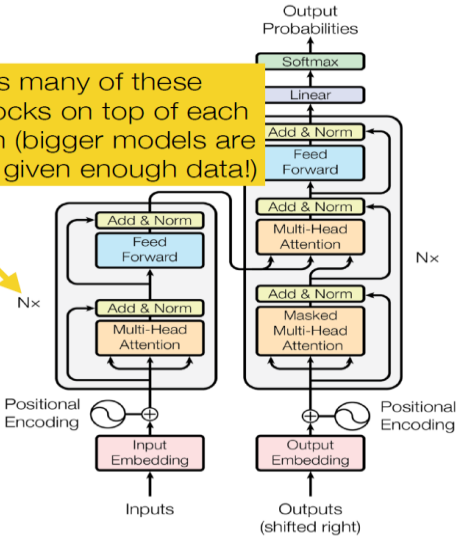
Friday @ 7 PM


Residual connections, which mean that we add the input to a particular block to its output, help improve gradient flow
Output Probabilities
Softmax
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

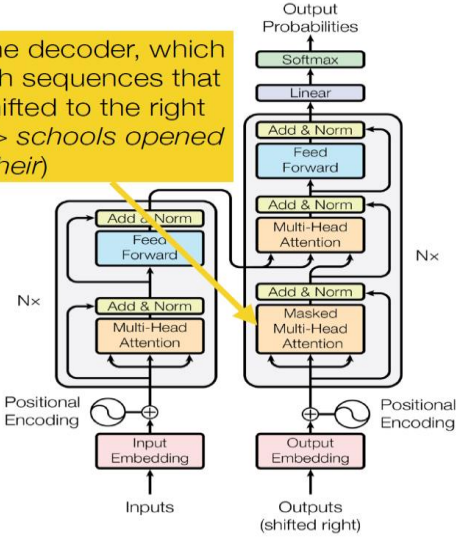
A feed-forward layer on top of the attention-weighted averaged value vectors allows us to add more parameters / nonlinearity



We stack as many of these *Transformer* blocks on top of each other as we can (bigger models are generally better given enough data!)

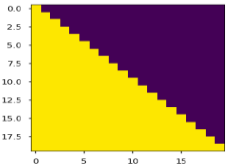


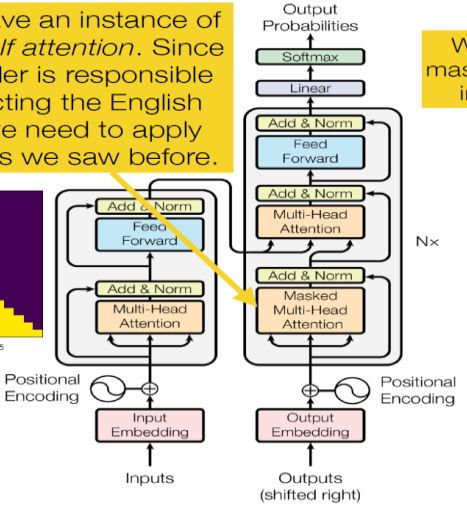
Moving onto the decoder, which takes in English sequences that have been shifted to the right (e.g., *<START> schools opened their*)



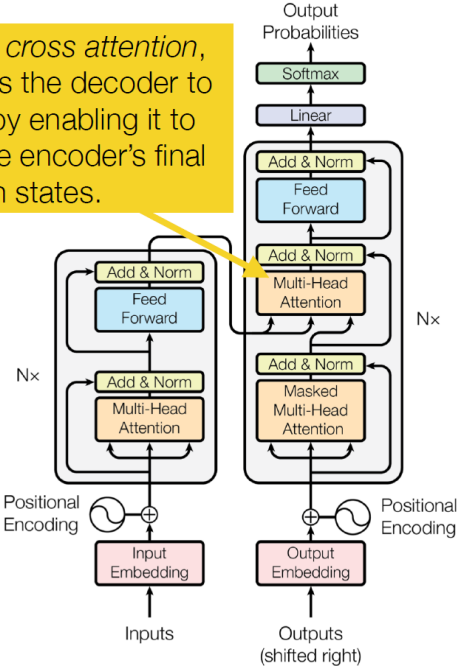
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

Why don't we do masked self-attention in the encoder?

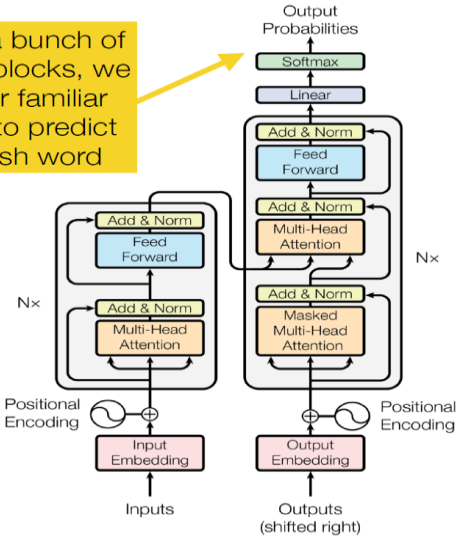




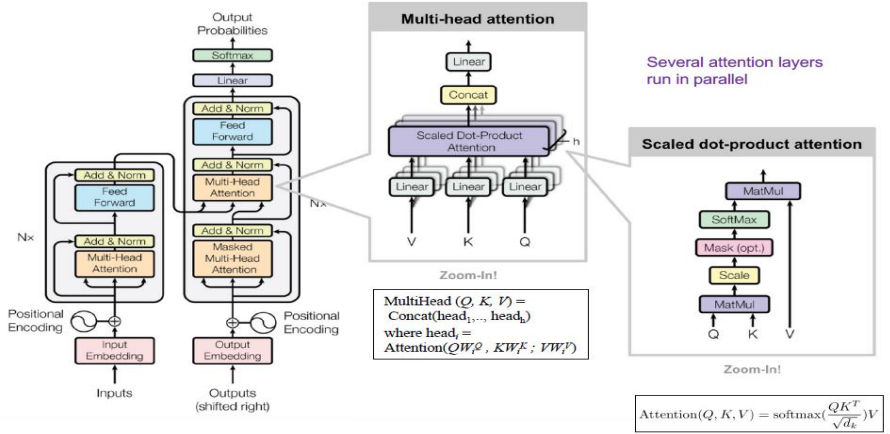
Now, we have *cross attention*, which connects the decoder to the encoder by enabling it to attend over the encoder's final hidden states.



After stacking a bunch of these decoder blocks, we finally have our familiar Softmax layer to predict the next English word



Full Transformer (with Attention)



Query, Key and Value in Attention

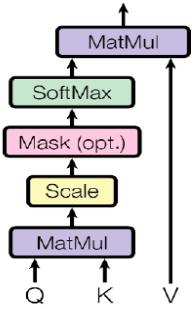
- To compute next word in translation, attention mechanism creates a vector using the source sentence and what has been generated so far

Input sentence	elle	alla	à	la	plage
Key	subject	verb	filler	filler	location
Value	*she*	*to go, past tense*	-	-	*beach*

Output sentence	she	went	to	the	?????
Query	subject	verb	filler	filler	location

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

Detail of Scaled dot product attention

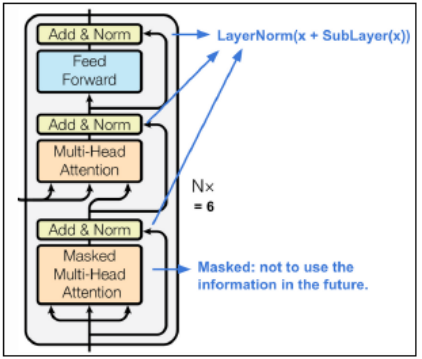


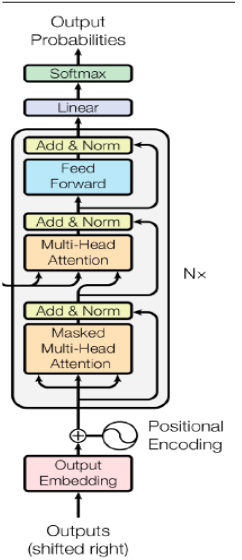
The input consists of queries and keys of dimension d_k, and values of dimension d_k. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

We compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as:

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

In contrast to the encoder, the decoder contains a **cross-attention** block, which compares to the encoded tokens of the input sequence using cross-attention (explained in more detail below). This cross-attention is at the heart of the decoder and helps producing the next output token, taking both the full input and the output produced so far into account.


Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
LayerNorm(x + SubLayer(x))
Nx
= 6
Masked: not to use the information in the future.


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
N×
Positional Encoding
Output Embedding
Outputs (shifted right)