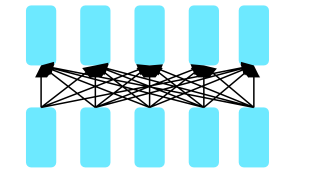
DSA4213

Lecture 6 - 20240304

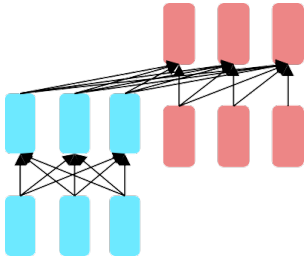
Dr Vishal Sharma,
Data Science Lead, H2O.AI

Today – Transformers

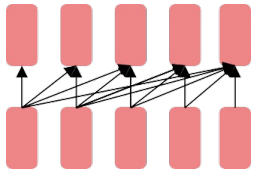
- Attention
- Transformer for MT
- BERT
- Other architectures



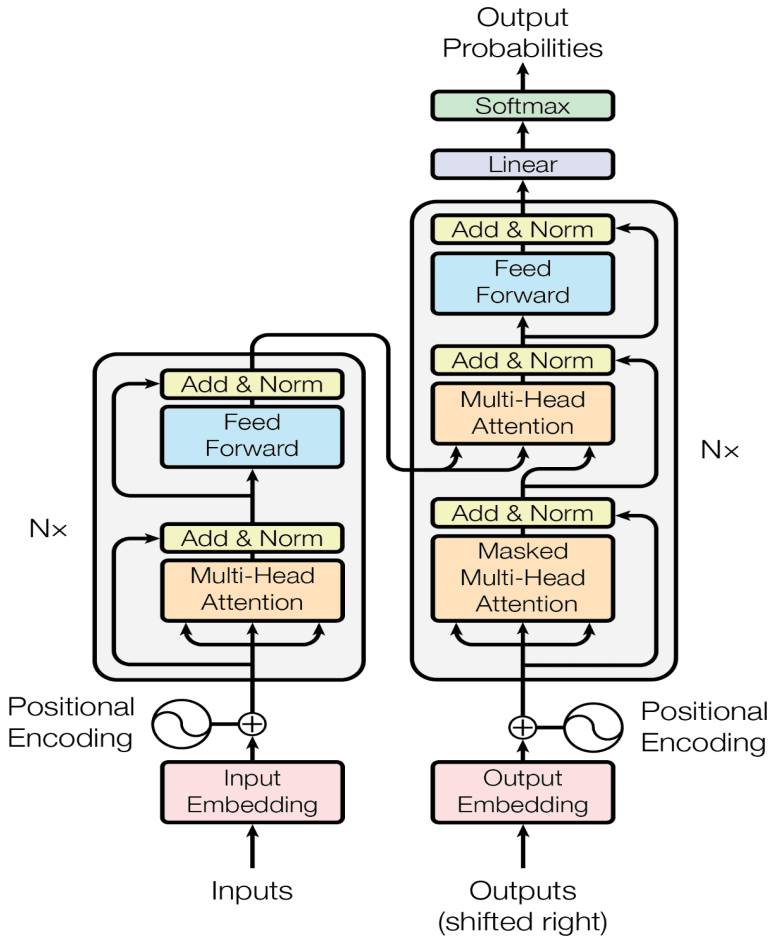
Encoders



Encoder-Decoders

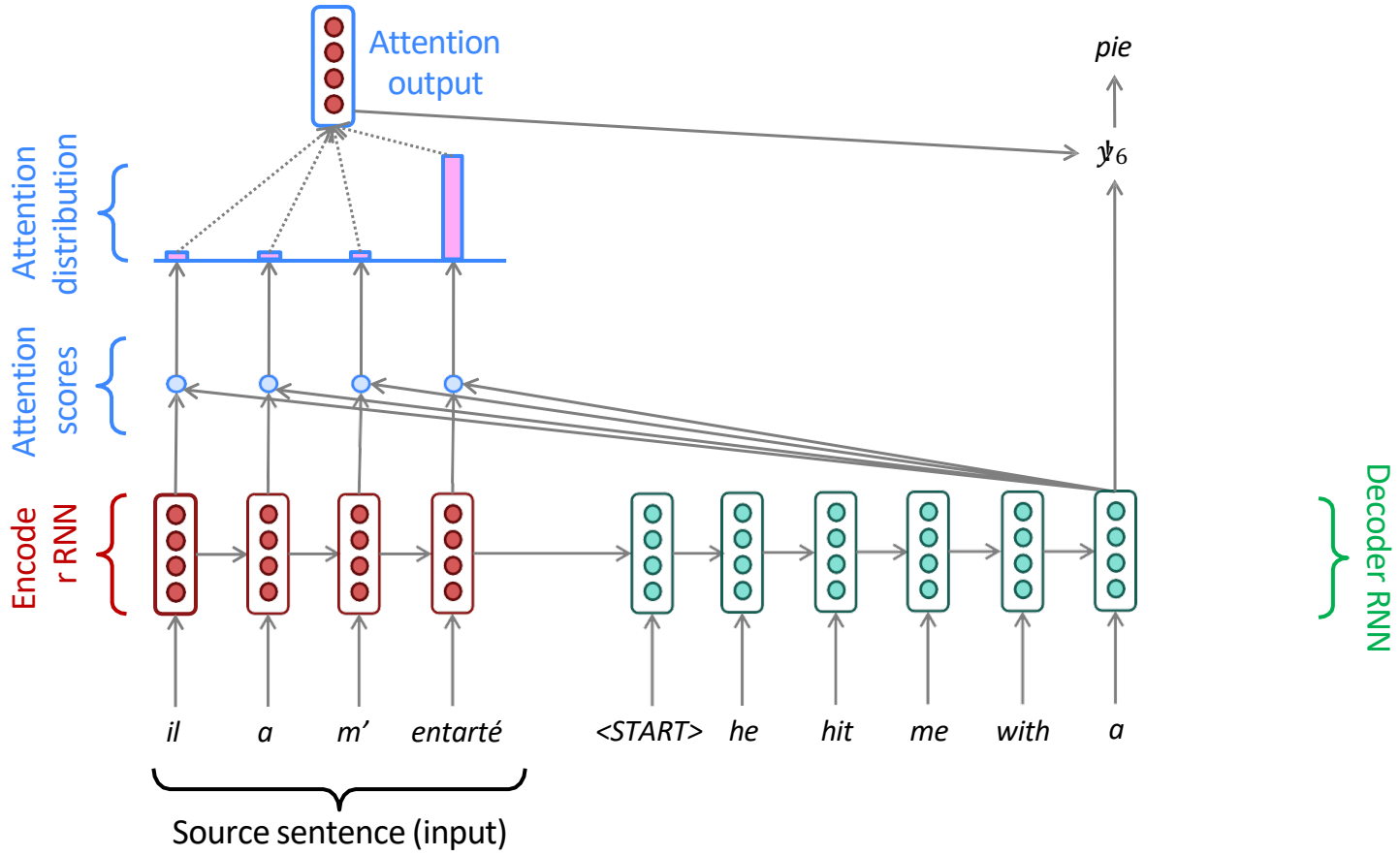


Decoders



Attention Recap

At every decoding step, use *attention mechanism to pay attention to a particular part* of the source sequence



Attention Terminologies

The general attention mechanism makes use of three main components, namely the *queries*, **Q**, the *keys*, **K**, and the *values*, **V**.

- **Q -** Think of this as a 'question' posed by each token,
- **K -** This acts like a 'label' for each token, against which the 'question' is matched..
- **V -** Contains the 'content' or actual information of the token

Attention in terms of Q, K, V

At every decoding step, use *attention mechanism to pay attention to a particular part* of the source sequence

The general attention mechanism makes use of three main components, namely the *queries*, **Q**, the *keys*, **K**, and the *values*, **V**.

- query would be analogous to the previous decoder output, y_{t-1},
- values would be analogous to the encoded inputs, h_i.
- Keys are encoded inputs which are weighted by attention score vector

- **MT Definition** → Given a set of encoded hidden states for input sentence, and a current word of translation target sentence, attention weight vector is estimated and the attention weighted encoder hidden states are passed as context to decoder at current word

- **General Definition** → Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

- **Query** determines the weight (attention vector) that should be given to **Values**. And that weights are obtained by comparing **Query** with **Keys**
- The weighted sum is a selective summary of the information contained in the **Values**, where the **Query** determines which **Values** to focus on.

Attention Equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $\boldsymbol{e}^t$ for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

Keys, Values - $\mathbb{R}^{d \times d}$

Query - $\mathbb{R}^{d \times d}$

Attention score

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

Softmaxed Attention score

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

Weighted Values

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

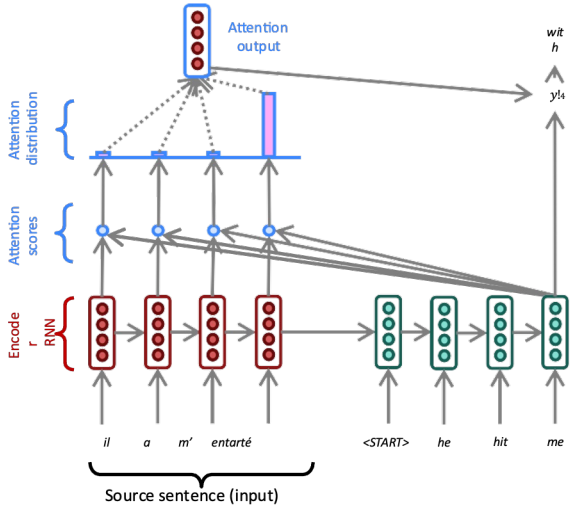
Input to Decoder RNN

Attention Benefits

Early NMT models used RNNs to develop representations of words.

h_{t+1} encodes information about w_t and w_{t+1}
h_{t+2} encodes information about w_t, w_{t+1} and w_{t+2}

Similarly $a_t s_t$, which includes “attended” information from $h_{1:N}$ and s_t decodes y_t



However words which appear later in the sentence may help in better encoding the current word

E.g.

Do you have apple charger, I need the charge my phone

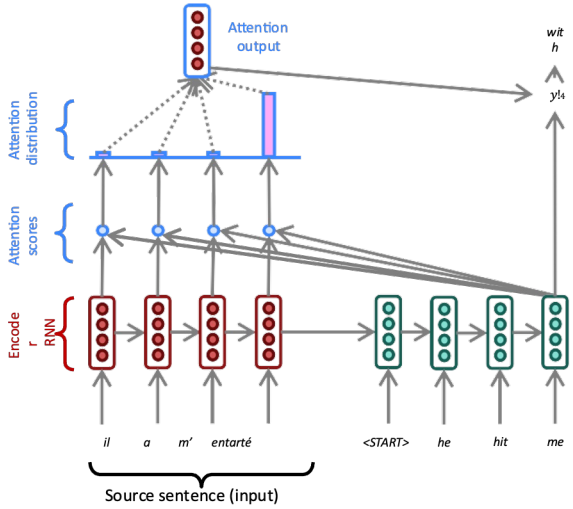
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Attention Benefits

Early NMT models used RNNs to develop representations of words.

h_{t+1} encodes information about w_t and w_{t+1}
h_{t+2} encodes information about w_t, w_{t+1} and w_{t+2}

Similarly $a_t s_t$, which includes “attended” information from $h_{1:N}$ and s_t decodes y_t



However words which appear later in the sentence may help in better encoding the current word

E.g.

Do you have apple charger, I need the charge my phone

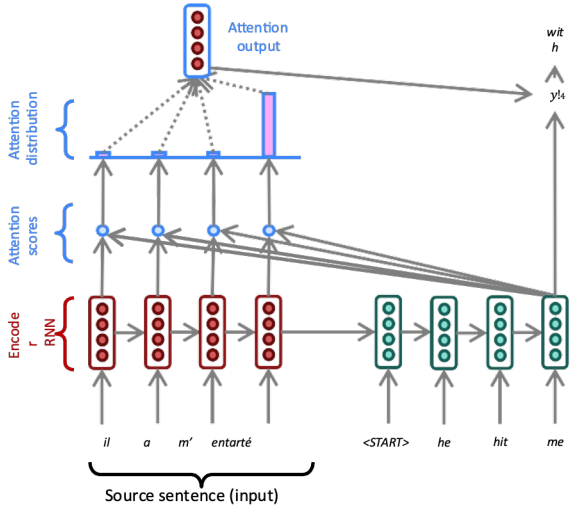
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Attention Benefits

Early NMT models used RNNs to develop representations of words.

h_{t+1} encodes information about w_t and w_{t+1}
h_{t+2} encodes information about w_t, w_{t+1} and w_{t+2}

Similarly $a_t s_t$, which includes “attended” information from $h_{1:N}$ and s_t decodes y_t



However words which appear later in the sentence may help in better encoding the current word

E.g.

Do you have apple charger, I need the charge my phone

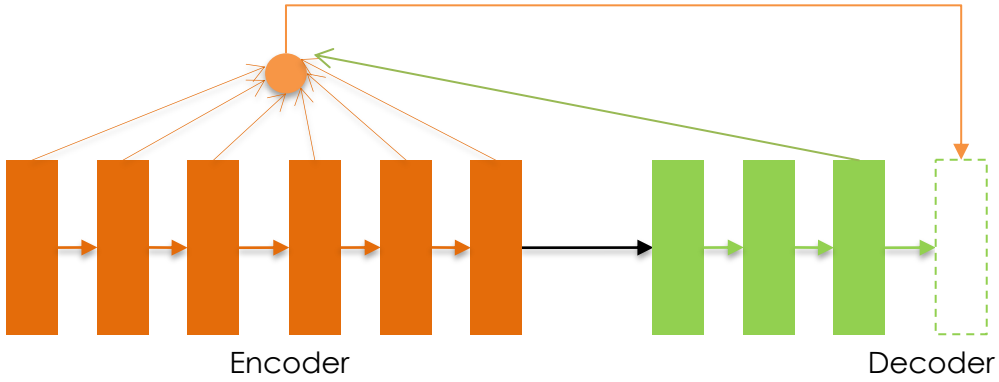
Since attention is effective in helping the decoder predict the next sentence, can we use attention mechanism to encode representation of word on same sentence

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Attention Benefits

Since attention is effective in helping the decoder predict the translation target sentence

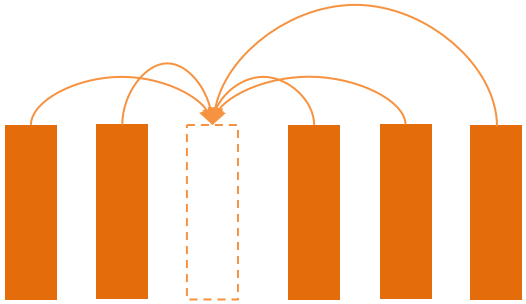
- query - previous decoder output, y_{t-1},
- values - encoded inputs, h_i.
- Keys - encoded inputs which are weighted by attention score vector



cross attention

can we use attention mechanism to encode representation of word on input source sentence as well

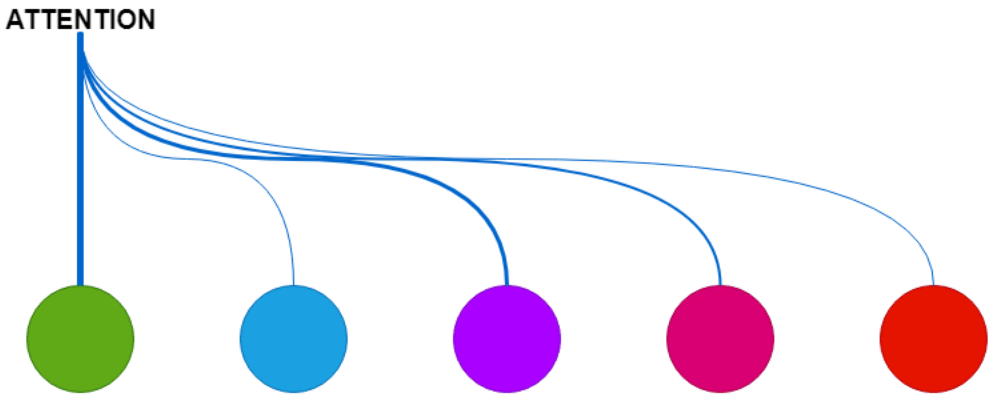
- query – current token
- values – all tokens
- Keys - encoded inputs which are weighted by attention score vector



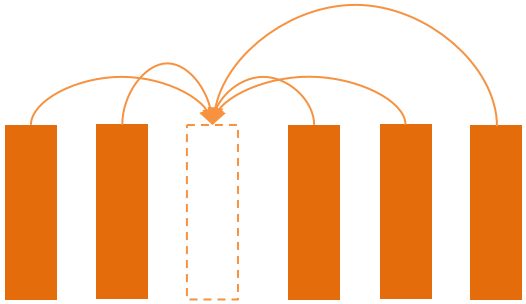
self attention

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Self Attention



can we use attention mechanism to encode representation of word on input source sentence as well



self attention

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Self Attention

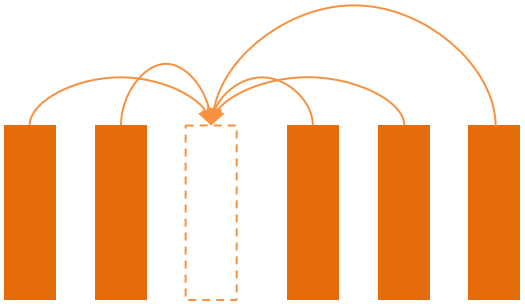
- The following sentence is an input sentence we want to translate: **"The animal didn't cross the street because it was too tired"**

- What does **"it"** in this sentence refer to?
- Is it referring to the **street** or to **the animal**? It's a simple question to a human but not as simple to an algorithm.

- When the model is processing the word "it", self-attention allows it to associate **"it"** with **"animal".**

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

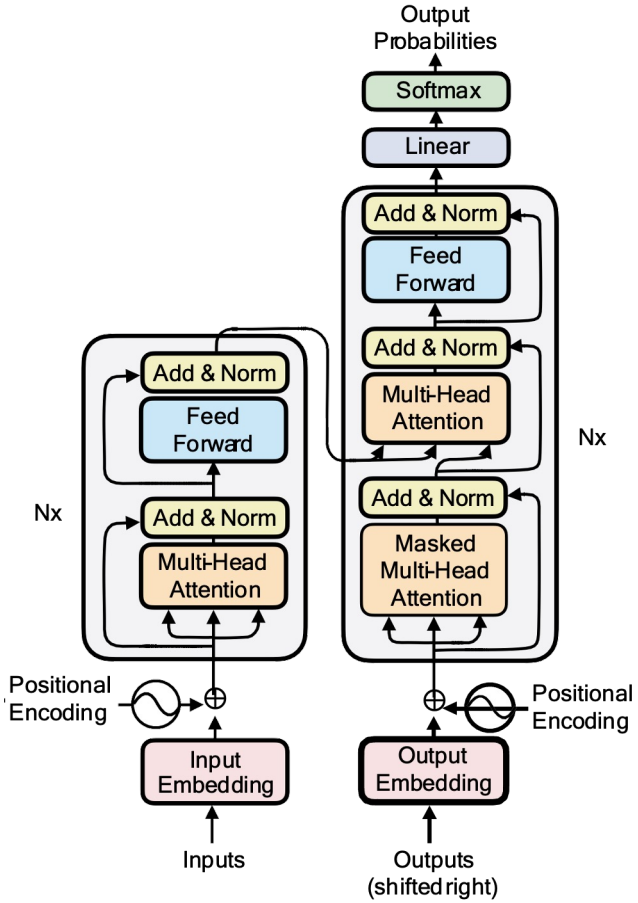
“Attention is All You Need” – Vaswani et al. 2017



self attention

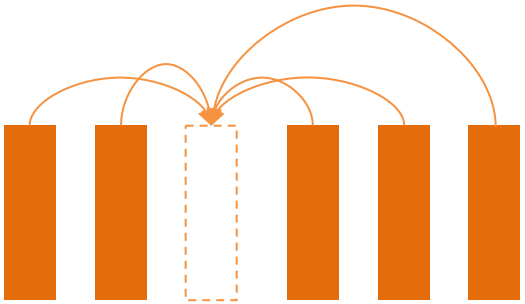
Transformer

- A sequence-to-sequence model based entirely on attention
- Strong results on machine translation
- Fast: Parallel Computation



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

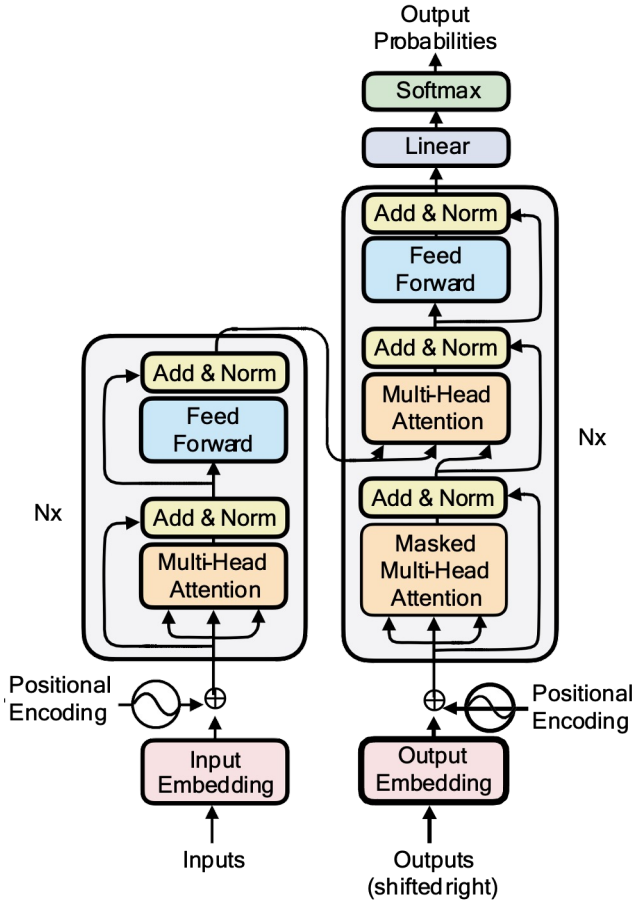
“Attention is All You Need” – Vaswani et al. 2017



self attention

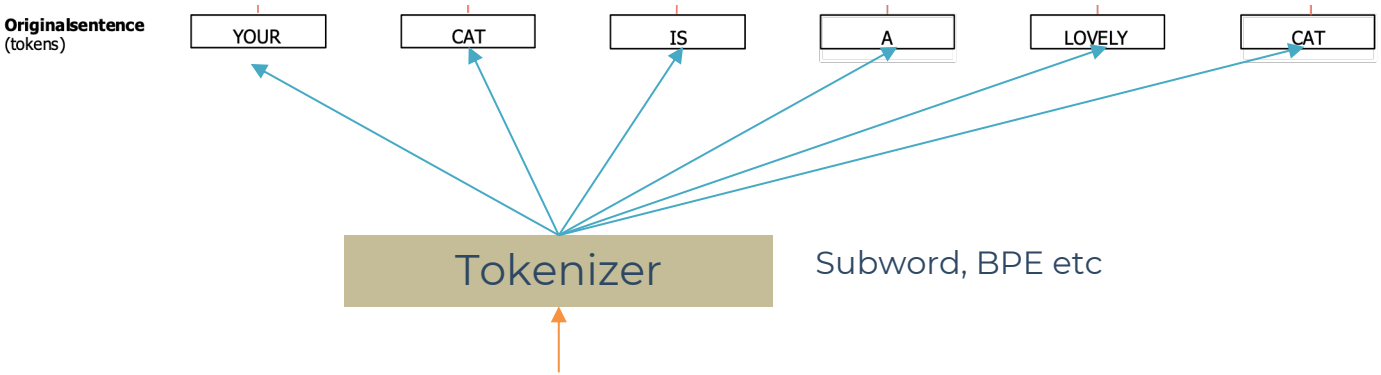
Some New Concepts

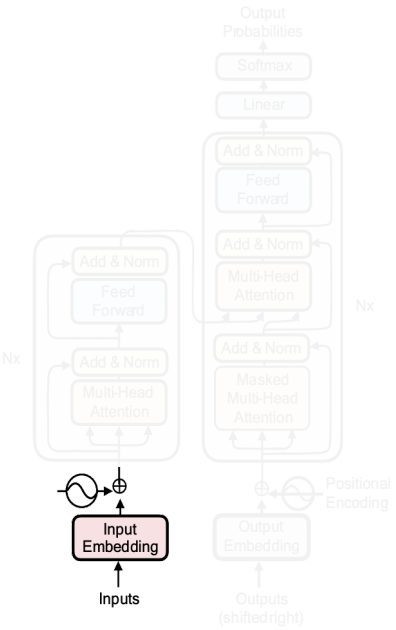
- Positional encodings
- Self-attention
- Multi-headed attention
- Masked attention
- Residual + layer normalization



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Before the input - Tokenization





"Your cat is a lovely cat"

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

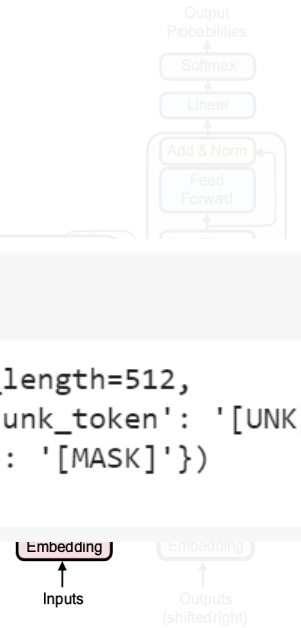
Before the input - Tokenization

"Huggingface Transformer" library allows you to use the same tokenizer as the model

```
# Import AutoTokenizer and create tokenizer object
from transformers import AutoTokenizer
checkpoint = 'bert-base-cased'
tokernizer = AutoTokenizer.from_pretrained(checkpoint)
```

```
]  tokernizer
```

```
BertTokenizerFast(name_or_path='bert-base-cased', vocab_size=28996, model_max_length=512,
is_fast=True, padding_side='right', truncation_side='right', special_tokens={'unk_token': '[UNK]',
'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]', 'mask_token': '[MASK]'})
```



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

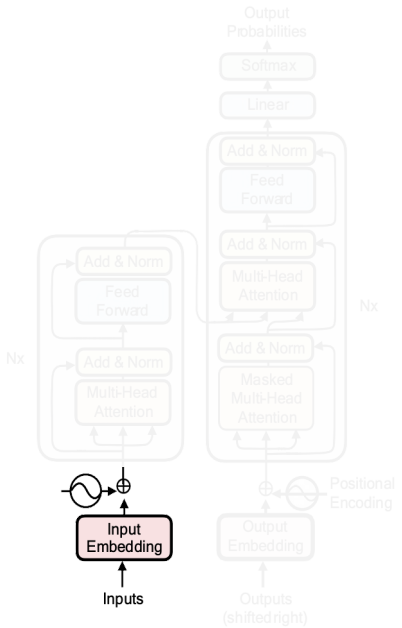
Before the input – Padding, Truncation

preprocessing techniques used in transformers to ensure that all input sequences have the same length.

maximum length for the sequences is set based on the specific task and the available computational resources.

Padding refers to the process of adding extra tokens (usually a special token such as [PAD]) to the end of short sequences

Truncation, on the other hand, refers to the process of cutting off the end of longer sequences so that they are all the same length.



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Before the input – Padding, Truncation

```
Sequence 1: "The cat sat on the mat [PAD] [PAD] [PAD] [PAD]"
Sequence 2: "The dog chased the cat [PAD] [PAD] [PAD] [PAD] [PAD]"
Sequence 3: "The mouse ran away from the cat and the dog"
```

Map tokens to integers

```
unpadded = [
  [1, 2, 3],
  [4, 5],
  [6, 7, 8, 9, 10],
]
```

```
padded = [
  [1, 2, 3, 0, 0],
  [4, 5, 0, 0, 0],
  [6, 7, 8, 9, 10],
]
```

Padding

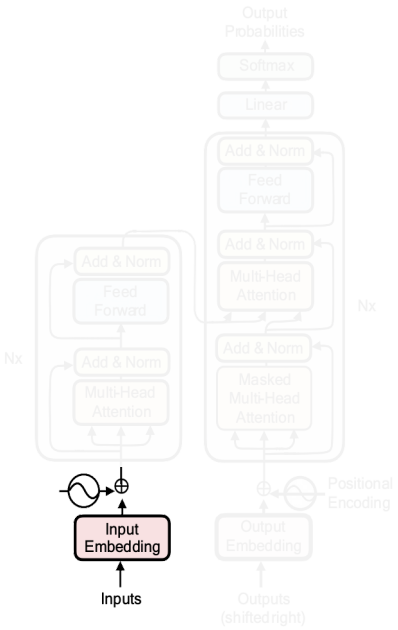
```
Sequence 1: "The cat sat on the"
Sequence 2: "The dog chased the cat"
Sequence 3: "The mouse ran away from"
```

Map tokens to integers

```
toolong = [
  [1, 2, 3, …, lim],
  [4, 5],
  [6, 7, …, lim, …, 8],
]
```

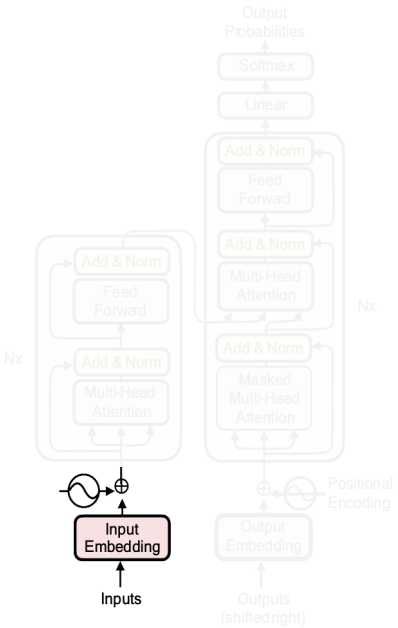
```
justright = [
  [1, 2, 3, …, lim],
  [4, 5, 0, …, lim],
  [6, 7, …,    lim],
]
```

Truncation



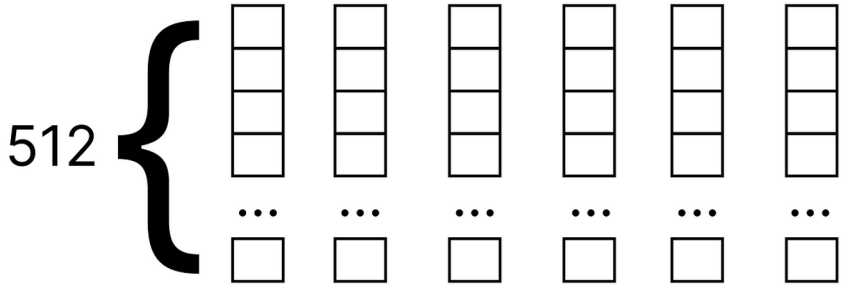
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

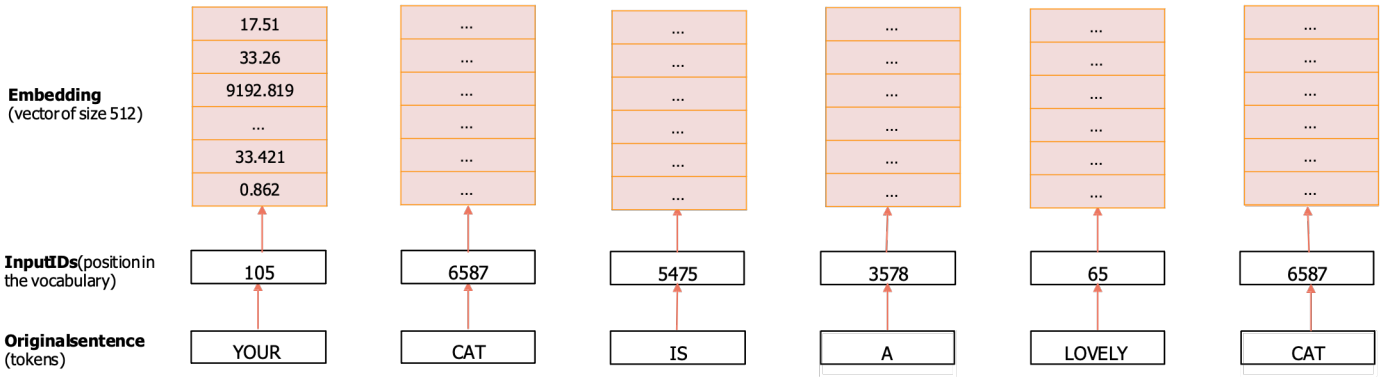
Input to Model – TokenId – Assigned Id in the vocabulary

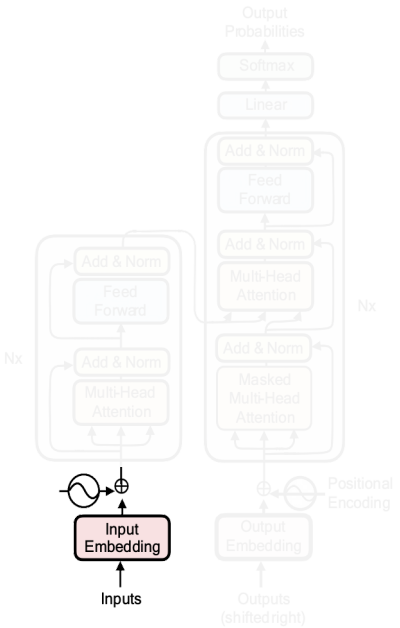


InputIDs (position in the vocabulary)	105	6587	5475	3578	65	6587
Original sentence (tokens)	YOUR	CAT	IS	A	LOVELY	CAT

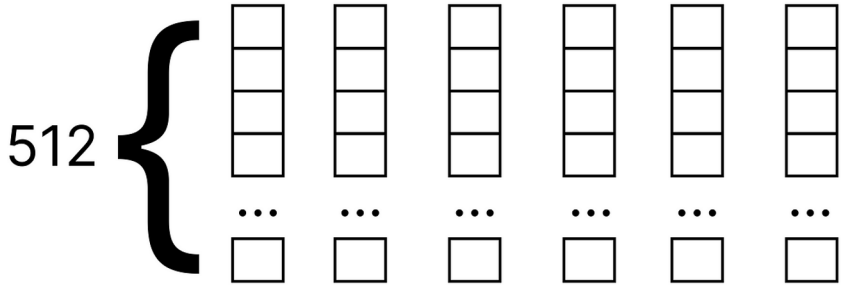
Input to Model

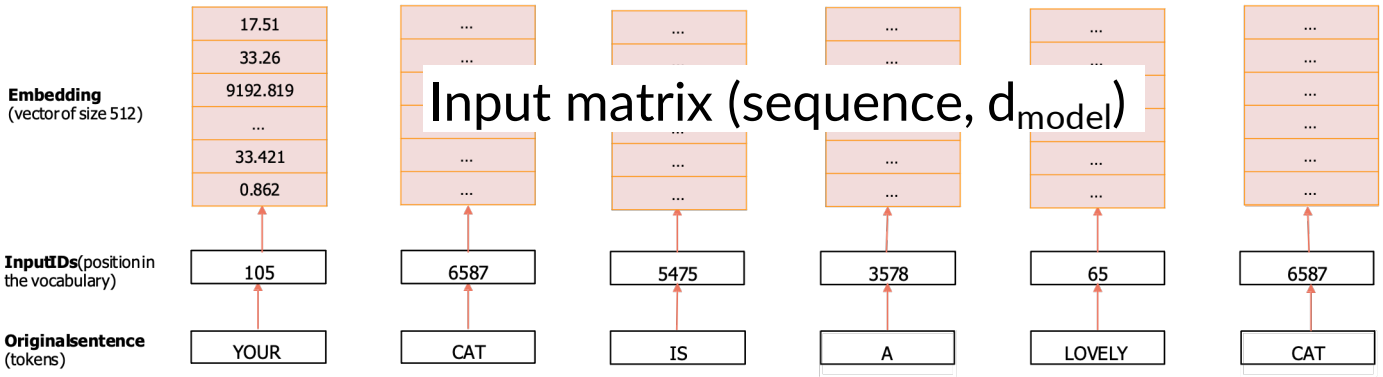


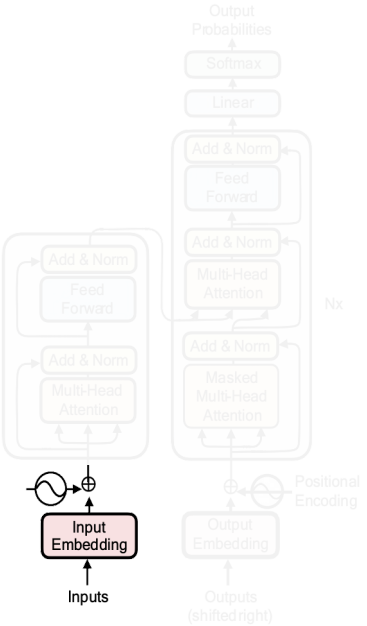




Input to Model



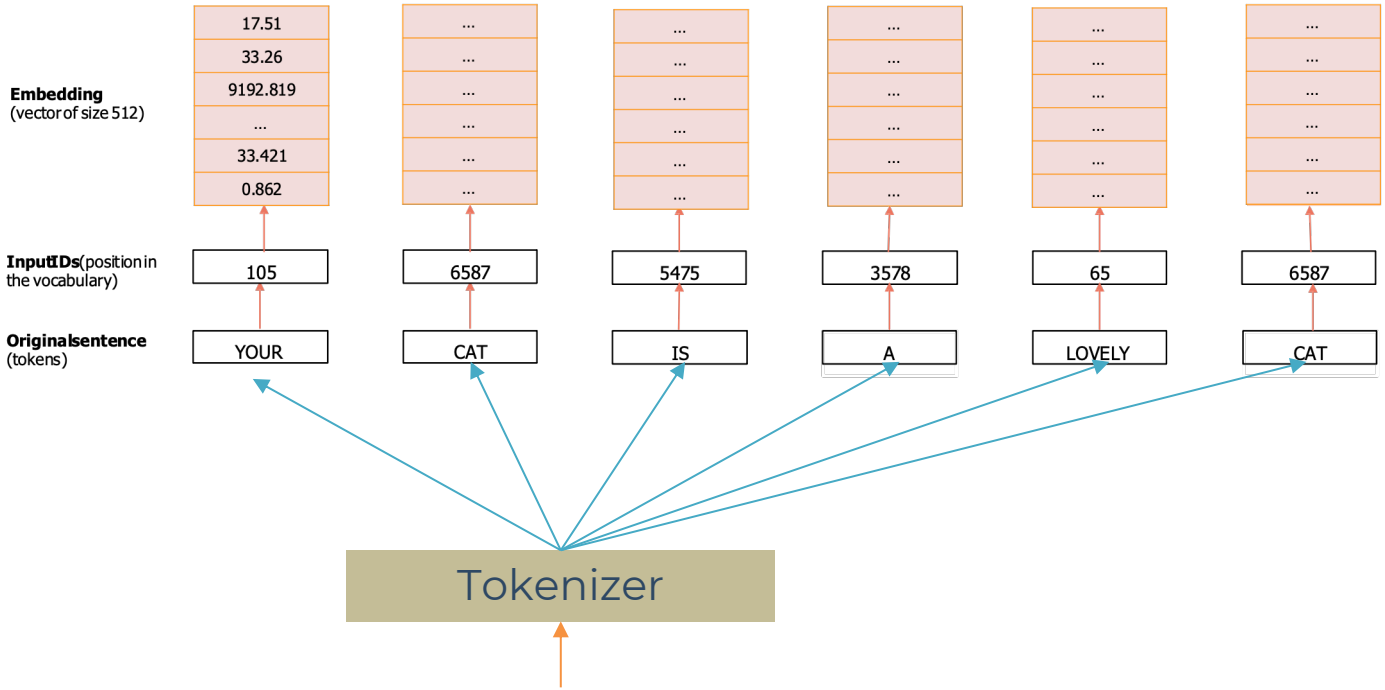


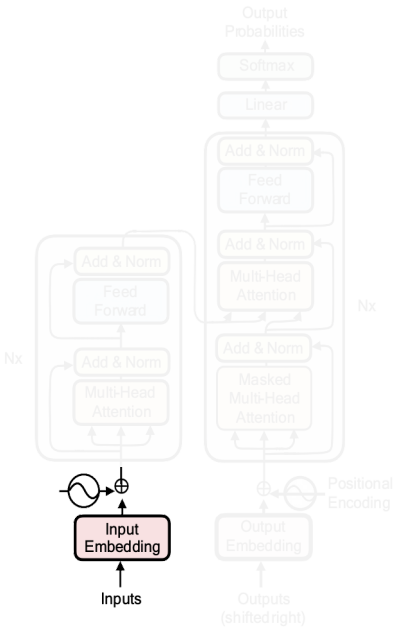


Embedding Vector can be initialised using word2vec, but generally randomly initialized and learned

Overall Input Mechanism –

Tokenization ->padding/truncation/ -> TokenId -> Embedding

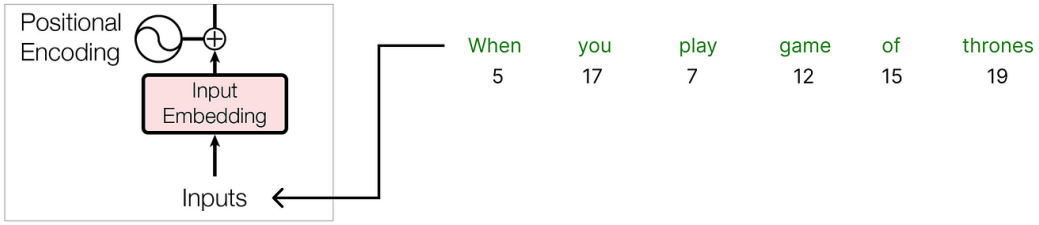


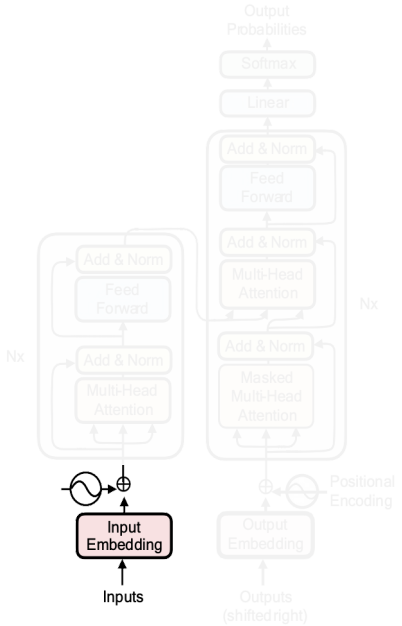


"Your cat is a lovely cat"

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

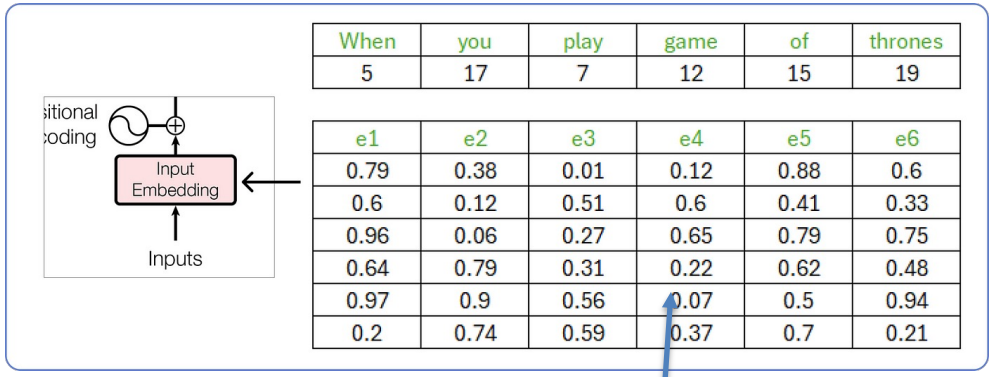
Input to Model – another example





Input to Model - Embeddings

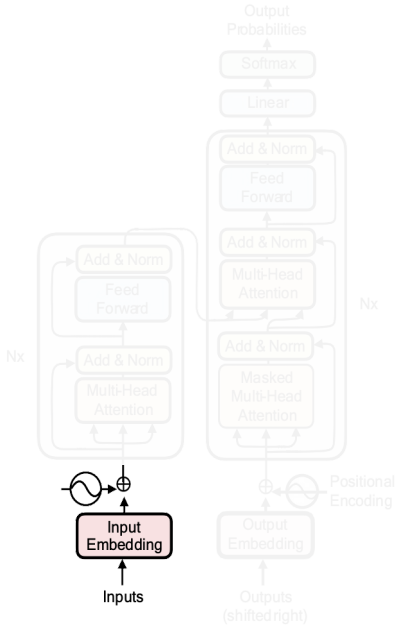
These embeddings do not have sequence information at all



When	you	play	game	of	thrones
5	17	7	12	15	19

e1	e2	e3	e4	e5	e6
0.79	0.38	0.01	0.12	0.88	0.6
0.6	0.12	0.51	0.6	0.41	0.33
0.96	0.06	0.27	0.65	0.79	0.75
0.64	0.79	0.31	0.22	0.62	0.48
0.97	0.9	0.56	0.07	0.5	0.94
0.2	0.74	0.59	0.37	0.7	0.21

random values



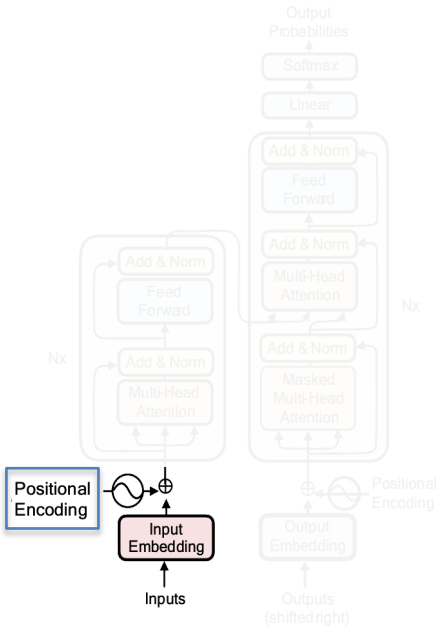
Input to Model – Position Encodings

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$$\boldsymbol{p}_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\ldots,n\} \text{ are position vectors}$$

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the $\boldsymbol{p}_i$ to our inputs!
- Recall that e_i is the embedding of the word at index i. Overall embeddings is

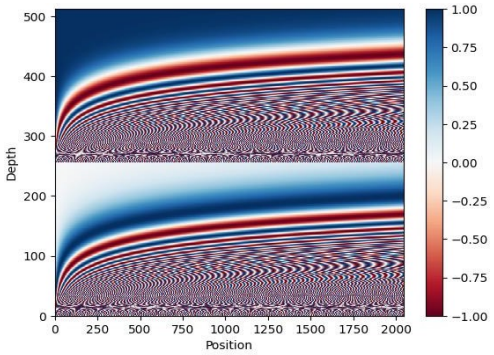
$$\boldsymbol{ep_i = e_i + p_i}$$



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

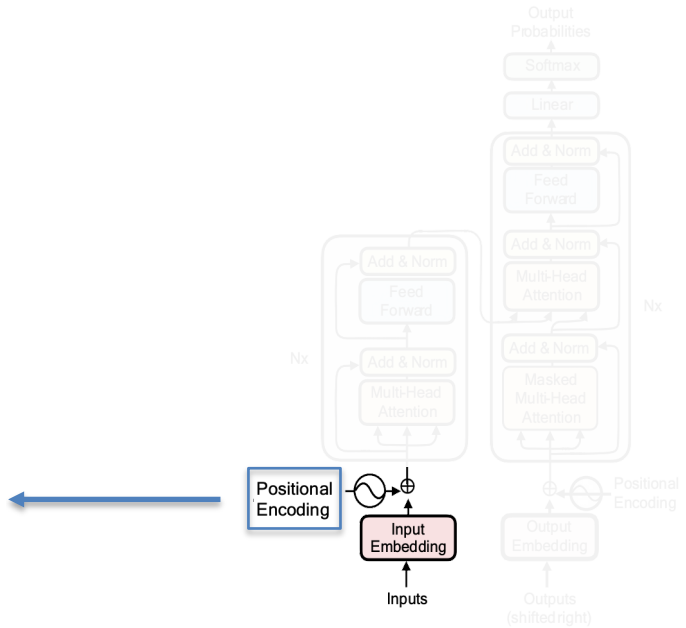
Input to Model

Trigonometric functions like **cos** and **sin** naturally represent a pattern that the model can recognize as continuous, so relative positions are easier to see for the model. By watching the plot of these functions, we can also see a regular pattern, so we can hypothesize that the model will see it too.



When
5

i	e1	Position	Formula	p1
0	0.79	Even	$sin(0/10000^{(2*0/6)})$	0
1	0.6	Odd	$cos(0/10000^{(2*1/6)})$	1
2	0.96	Even	$sin(0/10000^{(2*2/6)})$	0
3	0.64	Odd	$cos(0/10000^{(2*3/6)})$	1
4	0.97	Even	$sin(0/10000^{(2*4/6)})$	0
5	0.2	Odd	$cos(0/10000^{(2*5/6)})$	1
d (dim)	6			
POS	0			



random values

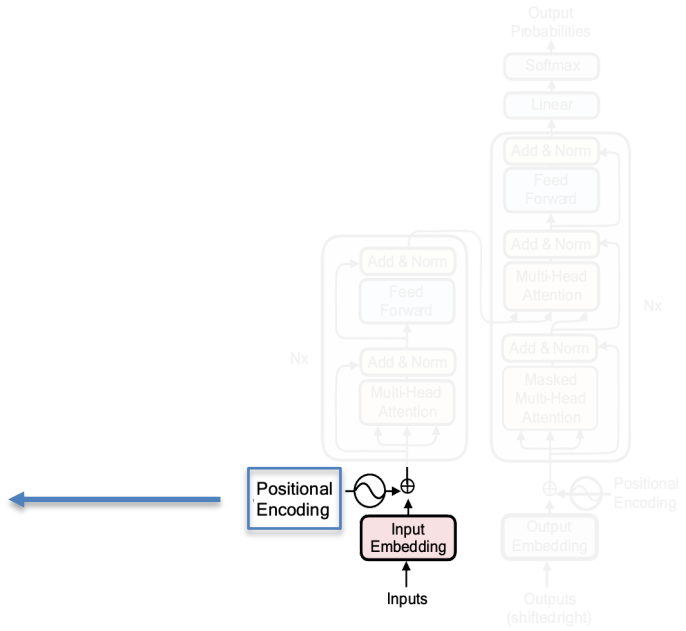
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Input to Model

We only need to compute the positional encodings once and then reuse them for every sentence, no matter if it is training or inference.

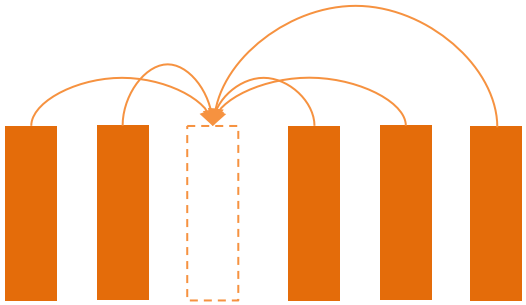
When	you	play	game	of	thrones
5	17	7	12	15	19

i	p1	p2	p3	p4	p5	p6
0	0	0.8415	0.9093	0.1411	-0.7568	-0.9589
1	1	0.0464	0.9957	0.1388	0.1846	0.9732
2	0	0.0022	0.0043	0.0065	0.0086	0.0108
3	1	0.0001	1	0.0003	0.0004	1
4	0	0	0	0	0	0
5	1	0	1	0	0	1
d (dim)	6	6	6	6	6	6
POS	0	1	2	3	4	5

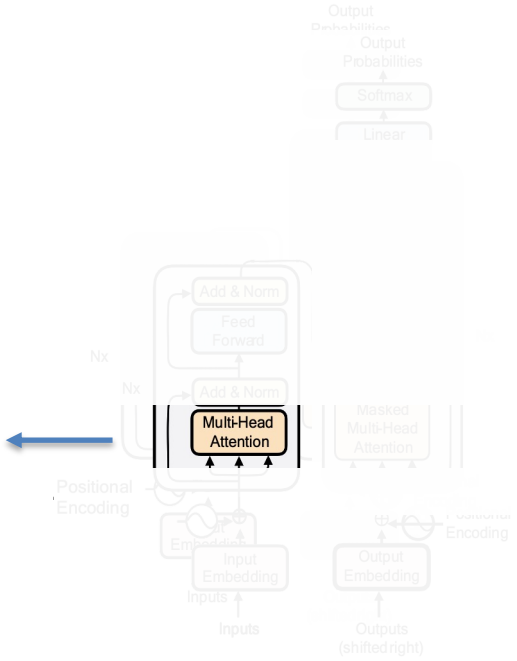


Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Attention in Transformers

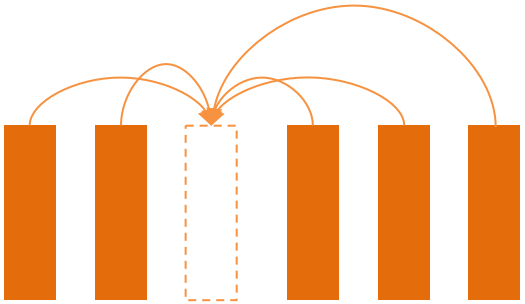


Incorporates Self-attention Mechanism



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

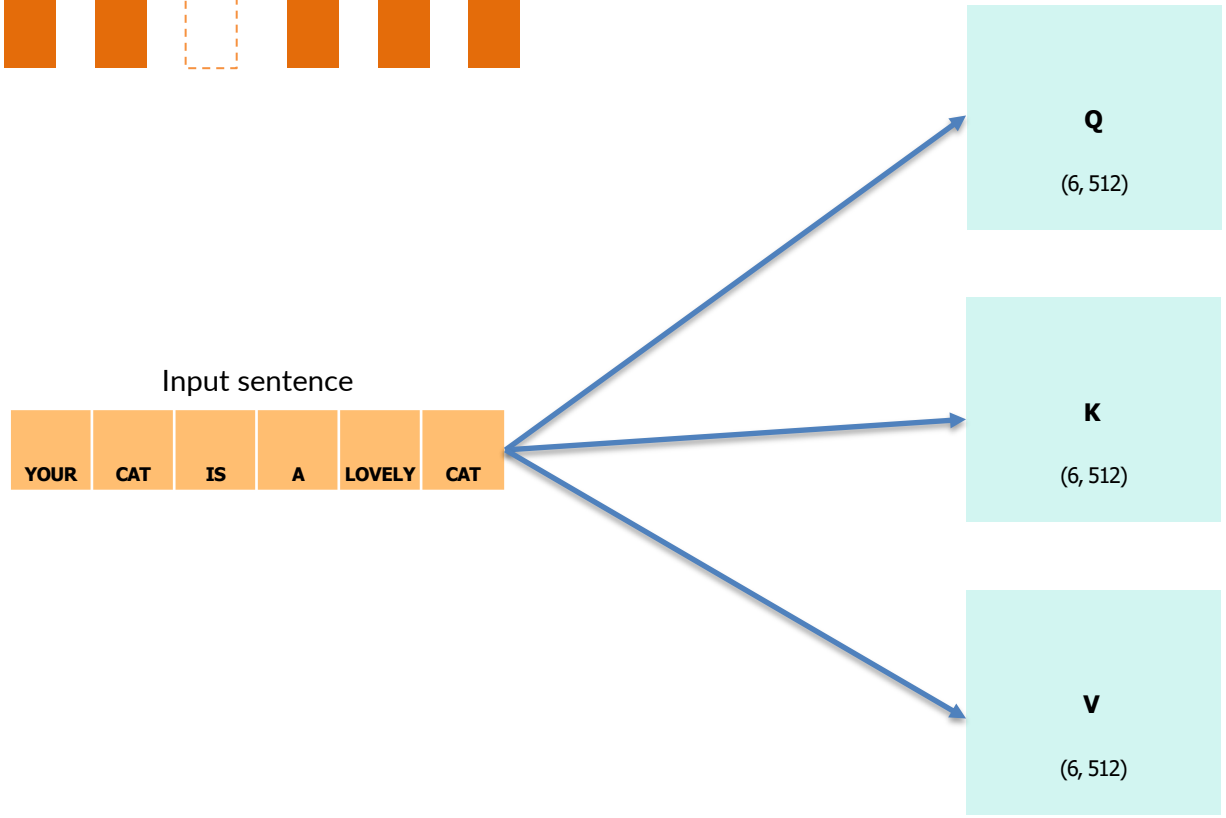
Attention in Transformers



Self-Attention allows the model to relate words to each other.

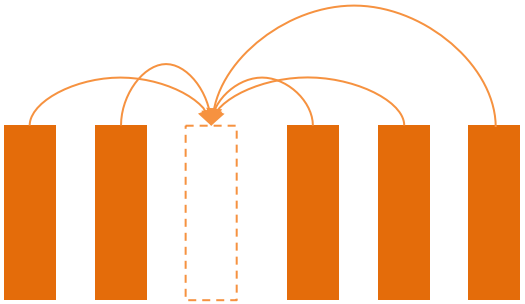
Let say the the sequence length **seq** = 6

Embedding size of the words in Transformer models $\mathbf{d_{model}} = \mathbf{d_k} = 512$.



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Attention in Transformers



Self-Attention allows the model to relate words to each other.

Let say the the sequence length **seq** = 6

Embedding size of the words in Transformer models $\mathbf{d_{model}} = \mathbf{d_k} = 512$.

Input sentence

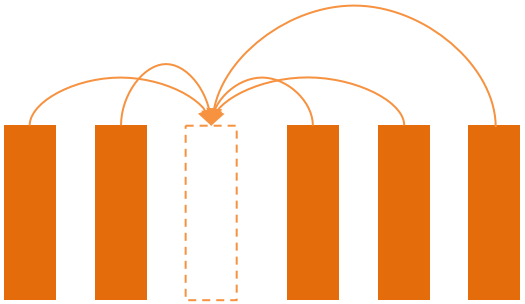
YOUR	CAT	IS	A	LOVELY	CAT

Self Attention in Transformer model

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

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Attention in Transformers



Self-Attention allows the model to relate words to each other.

Let say the the sequence length **seq** = 6

Embedding size of the words in Transformer models $\mathbf{d_{model}} = \mathbf{d_k} = 512$.

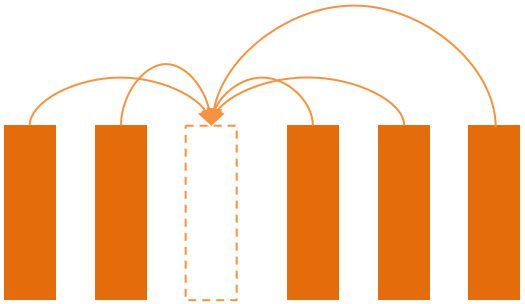
$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{softmax}\left(\frac{Q \text{ (6, 512)} \times K^T \text{ (512, 6)}}{\sqrt{512}}\right) =$$

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	---	---	---	---	---	---
CAT	---	---	---	---	---	---
IS	---	---	---	---	---	---
A	---	---	---	---	---	---
LOVELY	---	---	---	---	---	---
CAT	---	---	---	---	---	---

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Attention in Transformers



Self-Attention allows the model to relate words to each other.

Let say the the sequence length **seq** = 6

Embedding size of the words in Transformer models $\mathbf{d_{model}} = \mathbf{d_k} = 512$.

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	---	---	---	---	---	---
CAT	---	---	---	---	---	---
IS	---	---	---	---	---	---
A	---	---	---	---	---	---
LOVELY	---	---	---	---	---	---
CAT	---	---	---	---	---	---

X

V

(6, 512)

=

Attention

(6, 512)

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

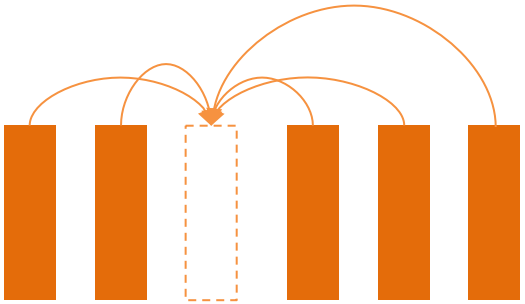
"Scaled Dot Product" attention aids in training. When dimensionality d becomes large, dot products between vectors tend to become large.Because of this, inputs to the softmax function can be large, making the gradients small.

Each row in this matrix captures

- the meaning (given by the embedding)
- the position of the word in the sentence (represented by the positional encodings)
- each word's interaction with other words.

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Attention in Transformers



Self-Attention allows the model to relate words to each other.

Let say the the sequence length **seq** = 6

Embedding size of the words in Transformer models $\mathbf{d_{model}} = \mathbf{d_k} = 512$.

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

=

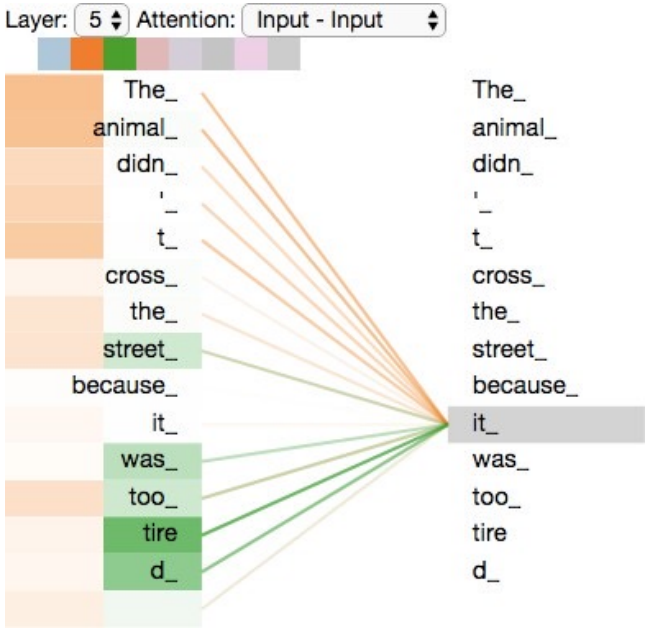
	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

random values

- Self-Attention is permutation invariant.
- Self-Attention requires no parameters. Up to now the interaction between words has been driven by their embedding and the positional encodings. This will change later.
- We expect values along the diagonal to be the highest.
- If we don't want some positions to interact, we can always set their values to **−∞** before applying the *softmax* in this matrix and the model will not learn those interactions. We will use this in the decoder.

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Self Attention based Representation of Tokens



As we encode the word "**it**", one attention head is focusing most on "**the animal**", while another is focusing on "**tired**" , in a sense, the model's representation of the word "**it**" bakes in some of the representation of both "**animal**" and "**tired**".

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

The attention mechanism from scratch

```python
from numpy import array
from numpy import random
from numpy import dot
from scipy.special import softmax

# Encoder representations of four different words
word_1 = array([1, 0, 0])
word_2 = array([0, 1, 0])
word_3 = array([1, 1, 0])
word_4 = array([0, 0, 1])

# Generating the weight matrices
W_Q = random.randint(3, size=(3, 5))
W_K = random.randint(3, size=(3, 5))
W_V = random.randint(3, size=(3, 5))

print("W_Q : \n",W_Q,"\n W_K \n",W_K,"\n W_V \n",W_V)
```

```
W_Q :
 [[2 0 2 1 2]
 [0 0 1 2 2]
 [1 2 2 0 2]]
 W_K
 [[2 1 1 0 2]
 [2 2 0 0 1]
 [0 2 2 0 2]]
 W_V
 [[2 0 0 2 2]
 [2 1 1 1 0]
 [1 0 0 1 1]]
```

```python
# Generating the queries, keys and values
query_1 = word_1 @ W_Q
key_1   = word_1 @ W_K
value_1 = word_1 @ W_V
print("query_1 : \n",query_1)
print("key_1   : \n",key_1)
print("value_1 : \n",value_1)

query_2 = word_2 @ W_Q
key_2   = word_2 @ W_K
value_2 = word_2 @ W_V

query_3 = word_3 @ W_Q
key_3   = word_3 @ W_K
value_3 = word_3 @ W_V

query_4 = word_4 @ W_Q
key_4   = word_4 @ W_K
value_4 = word_4 @ W_V
```

```
query_1 :
 [2 0 2 1 2]
key_1   :
 [2 1 1 0 2]
value_1 :
 [2 0 0 2 2]
```

```python
# Scoring the first query vector against all key vectors
score_1 = array([dot(query_1, key_1), dot(query_1, key_2),
                 dot(query_1, key_3), dot(query_1, key_4)])
print("score_1    : ",score_1)
score_2 = array([dot(query_2, key_1), dot(query_2, key_2),
                 dot(query_2, key_3), dot(query_2, key_4)])
score_3 = array([dot(query_3, key_1), dot(query_3, key_2),
                 dot(query_3, key_3), dot(query_3, key_4)])
score_4 = array([dot(query_4, key_1), dot(query_4, key_2),
                 dot(query_4, key_3), dot(query_4, key_4)])

# Computing the weights by a softmax operation
softmax_1 = softmax(score_1 / key_1.shape[0] ** 0.5)
print("softmax_1  :",softmax_1)
softmax_2 = softmax(score_2 / key_1.shape[0] ** 0.5)
softmax_3 = softmax(score_3 / key_1.shape[0] ** 0.5)
softmax_4 = softmax(score_4 / key_1.shape[0] ** 0.5)

# Computing the attention by a weighted sum of the value vectors
attention_1 = (softmax_1[0] * value_1) + (softmax_1[1] * value_2) + (softmax_1[2] * value_3) + (softmax_1[3] * value_4)
print("attention_1 : ",attention_1)
attention_2 = (softmax_2[0] * value_1) + (softmax_2[1] * value_2) + (softmax_2[2] * value_3) + (softmax_2[3] * value_4)
attention_3 = (softmax_3[0] * value_1) + (softmax_3[1] * value_2) + (softmax_3[2] * value_3) + (softmax_3[3] * value_4)
attention_4 = (softmax_4[0] * value_1) + (softmax_4[1] * value_2) + (softmax_4[2] * value_3) + (softmax_4[3] * value_4)
```

```
score_1    : [10  6 16  8]
softmax_1  : [0.06169402 0.01031225 0.90277064 0.02522309]
attention_1 : [3.7803182  0.9130829  0.9130829  2.86723531 1.95415241]
```

Matrix Calculation of Self-Attention

```python
from numpy import array
from numpy import random
from numpy import dot
from scipy.special import softmax
# Encoder representations of four different words
word_1 = array([1, 0, 0])
word_2 = array([0, 1, 0])
word_3 = array([1, 1, 0])
word_4 = array([0, 0, 1])
# Stacking the word embeddings into a single array
words = array([word_1, word_2, word_3, word_4])
# Generating the weight matrices
W_Q = random.randint(3, size=(3, 5))
W_K = random.randint(3, size=(3, 5))
W_V = random.randint(3, size=(3, 5))
# Generating the queries, keys and values
Q = words @ W_Q
K = words @ W_K
V = words @ W_V
# Scoring the query vectors against all key vectors
scores = Q @ K.transpose()
# Computing the weights by a softmax operation
softmax_weights = softmax(scores / K.shape[1] ** 0.5, axis=1)
# Computing the attention by a weighted sum of the value vectors
attention = softmax_weights @ V

print(attention)
```

```
[[1.86757486 1.81117752 0.97776709 0.99354807 1.97131516]
 [1.31344188 1.56645073 0.74699115 0.96376578 1.71075693]
 [1.94407694 1.91688789 0.99257915 0.99952531 1.99210446]
 [1.31344188 1.56645073 0.74699115 0.96376578 1.71075693]]
```

Multi-Headed Attention

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

- Transformer model actually uses Multi-headed attention where different

- What if we want to look in multiple places in the sentence at once?

- For word i, self-attention "looks" where $x^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices

  Let, $Q_P, K_P, V_P \in \mathbb{R}^h$, where h is the number of attention heads, and P ranges from 1 to h.

- Each attention head performs attention independently:

  output_P = softmax ($X Q_P K^\top X^\top$)x XV_P, where $\text{output}_P \in \mathbb{R}^{d/h}$

- Then the outputs of all the heads are combined!
  - output = $\text{output}_1; \ldots; \text{output}_h$ Y, where $Y \in \mathbb{R}^{d \times d}$

$$MultiHead(Q, K, V) = Concat(head_1 \ \ldots head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

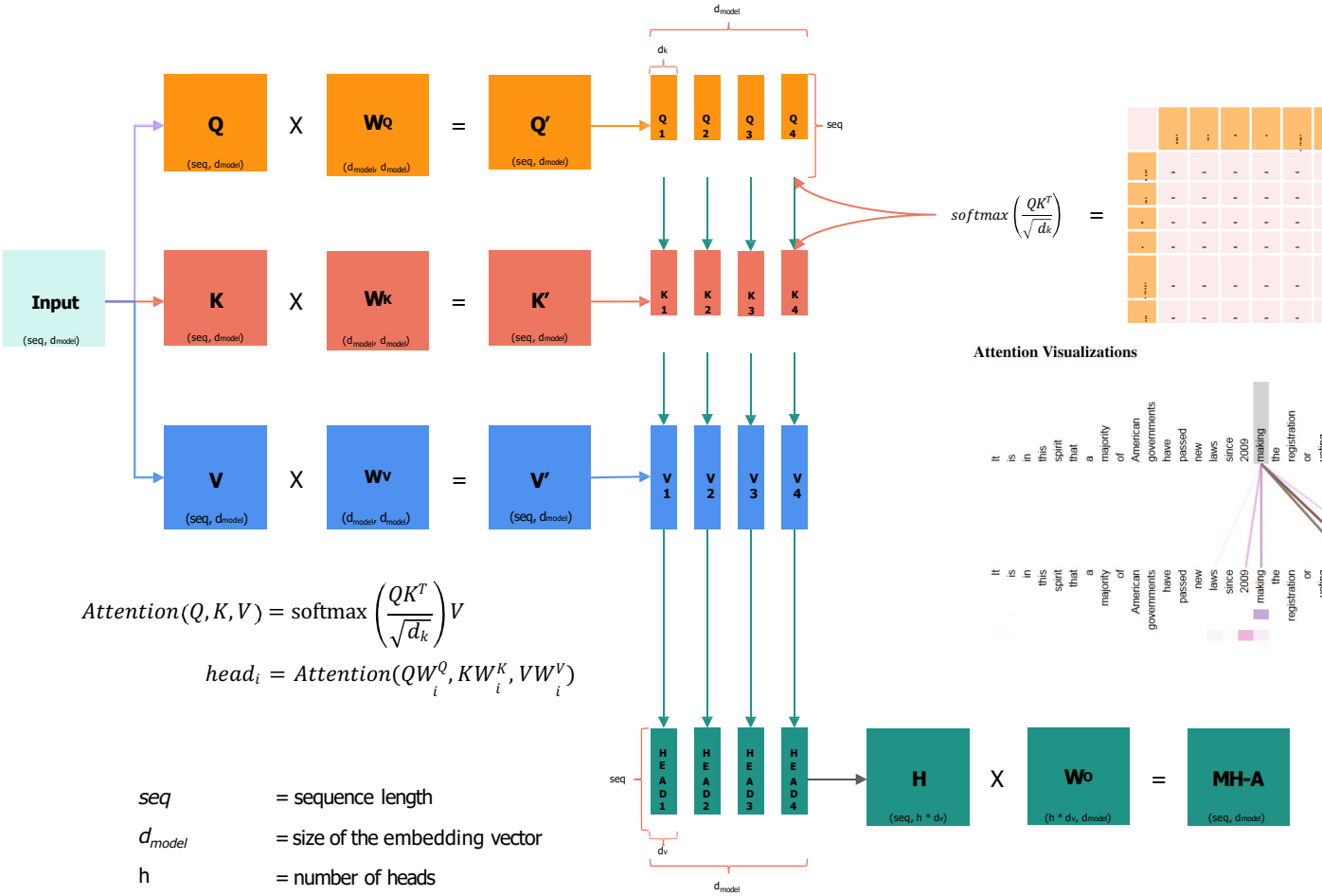
Multi-Headed Attention

Multi-Headed Attention improves the performance of the attention layer in two ways:

- It expands the model's ability to focus on **different positions**. Yes, in the example above, z1 contains a little bit of every other encoding, but it could be dominated by the actual word itself.

- It gives the attention layer **multiple representation subspaces**.

Multi-Headed Attention

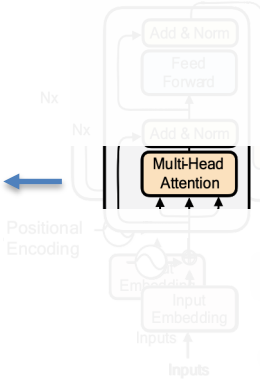


$$softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) =$$

Attention Visualizations

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

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

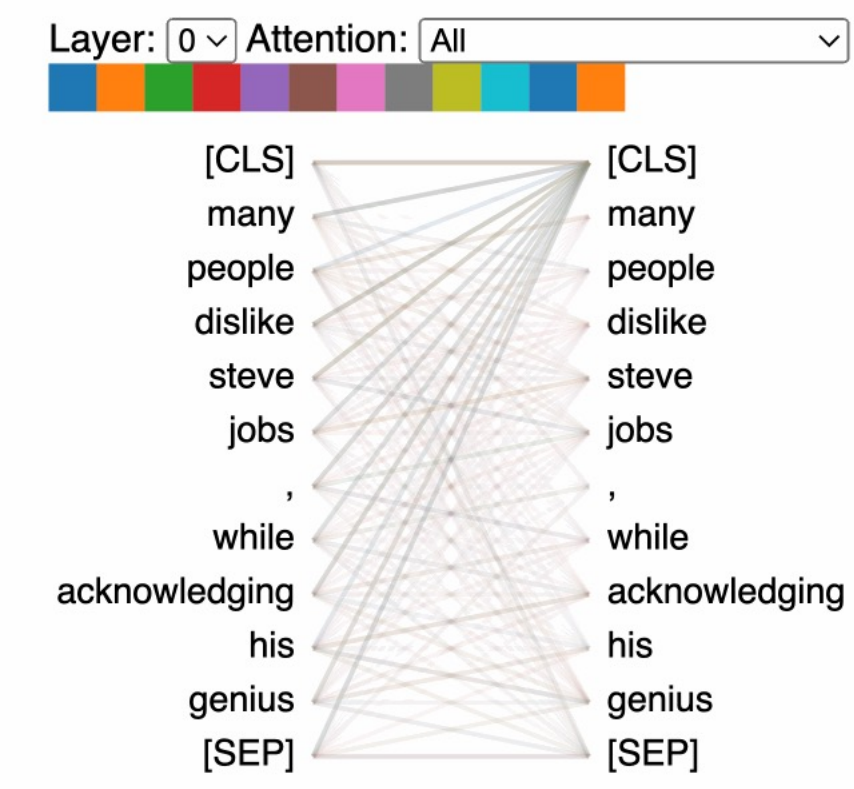


seq	= sequence length
d_{model}	= size of the embedding vector
h	= number of heads
$d_k = d_v$	= d_{model} / h

$$MultiHead(Q, K, V) = Concat(head_1 \ldots head_h)W^O$$

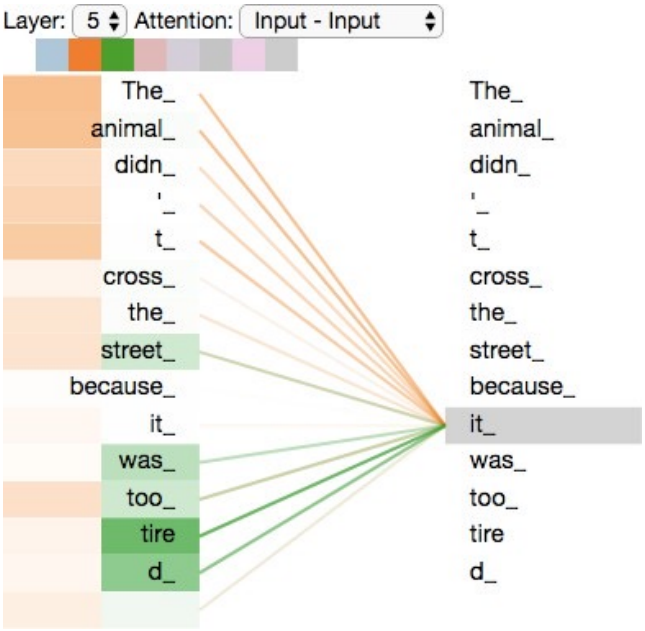
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Multi-Headed Attention

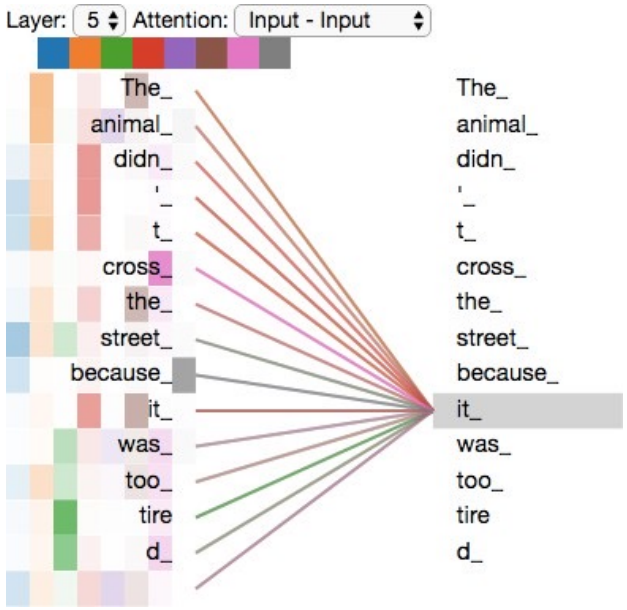


Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Multi-Headed Attention



As we encode the word "**it**", one attention head is focusing most on "**the animal**", while another is focusing on "**tired**" , in a sense, the model's representation of the word "**it**" bakes in some of the representation of both "**animal**" and "**tired**".



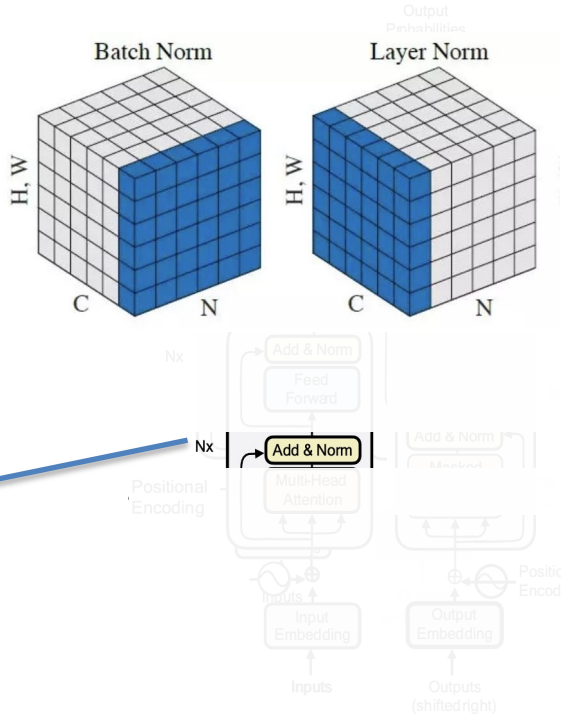
If we add all the attention heads to the picture, however, things can be harder to interpret.

Layer Normalization

- **Layer normalization** is a trick to help models train faster.
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**.
  - LayerNorm's success may be due to its normalizing gradients
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.

  $\mu = \sum^d x_j$; mean $\mu \in \mathbb{R}$.

  $\sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d} (x_j - \mu)^2}$; $\sigma \in \mathbb{R}$.
- Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters



Then layer normalization is computed as :

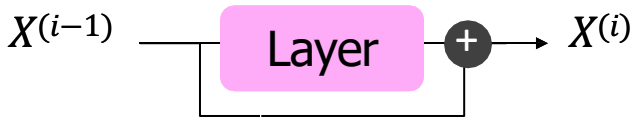
$$\frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

Modulate by learned elementwise gain and bias

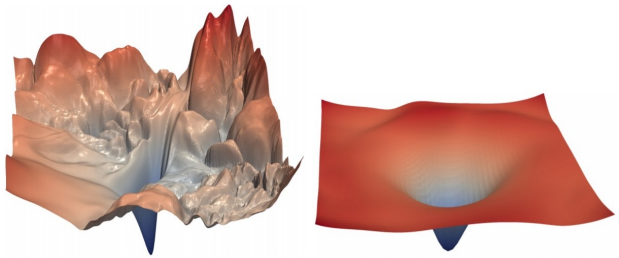
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Add + Normalization

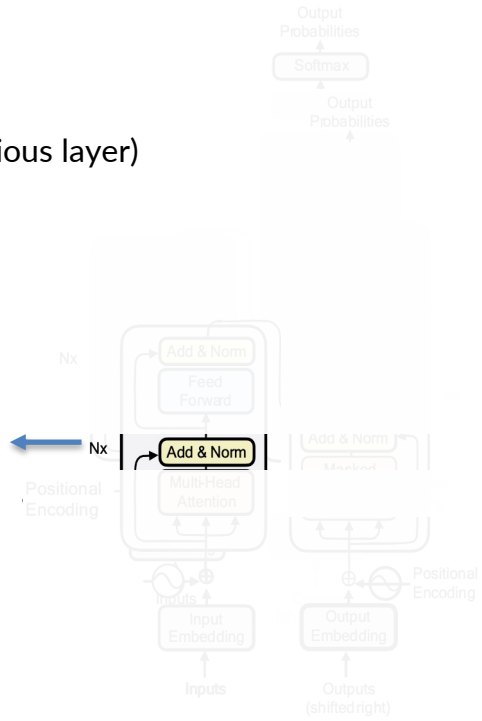
- **Residual connections** are a trick to help models train better.



- We let $X(i) = X(i-1)$ +Layer $X(i-1)$ (so we only have to learn "the residual" from the previous layer)



- [no residuals] [residuals]
- [Loss landscape visualization, Li et al., 2018, on a ResNet]



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

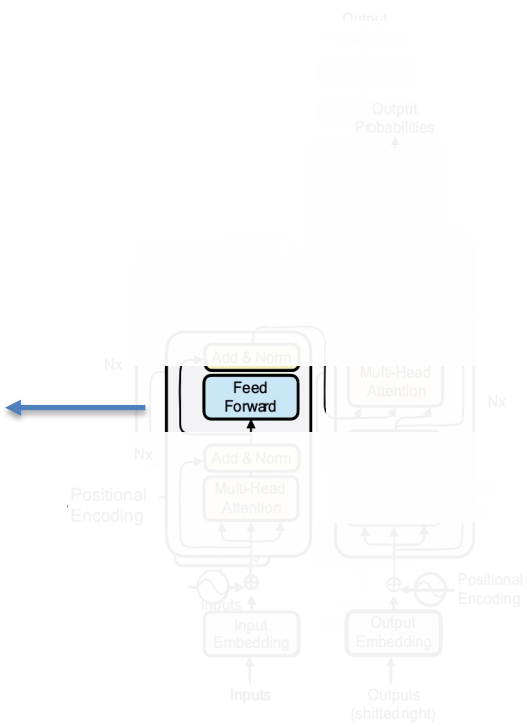
Feed Forward Layer – to add non-linearity

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors (Why? Look at the notes!)

- Easy fix: add a **feed-forward network** to post-process each output vector.

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \text{ x output }_i) + b_1 + b_2$$

$$\text{ReLU}(x) = \max(0, x)$$



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

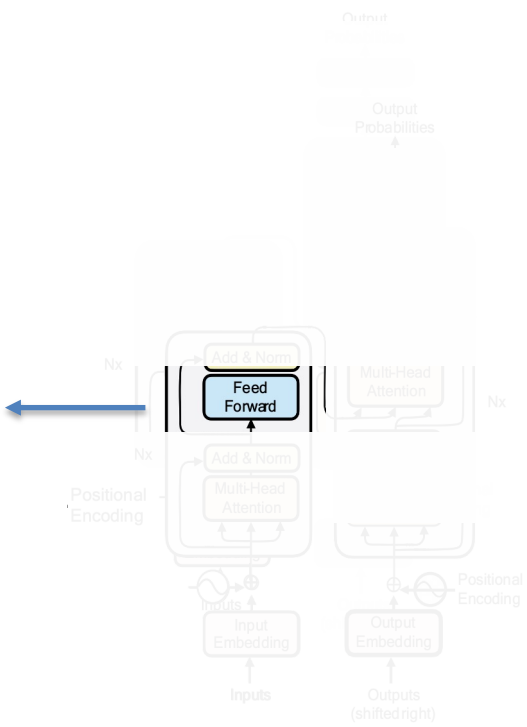
Feed Forward Layer – to add non-linearity

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors (Why? Look at the notes!)

- Easy fix: add a **feed-forward network** to post-process each output vector.

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \text{ x output }_i) + b_1 + b_2$$

$$\text{ReLU}(x) = \max(0, x)$$



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

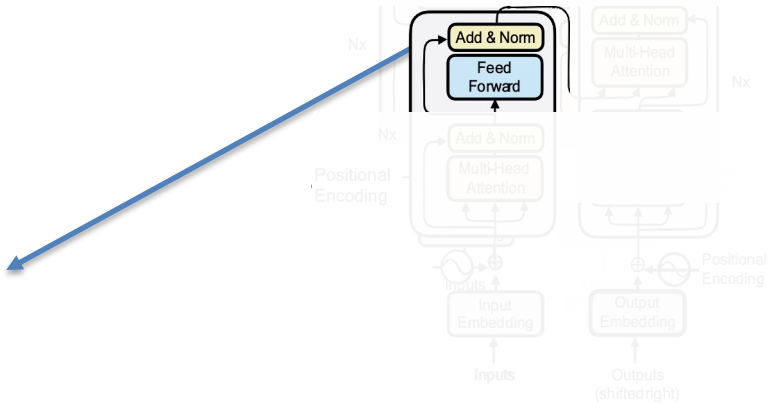
Feed Forward Layer – to add non-linearity

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors

- Easy fix: add a **feed-forward network** to post-process each output vector.

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \text{ x output }_i) + b_1 + b_2$$

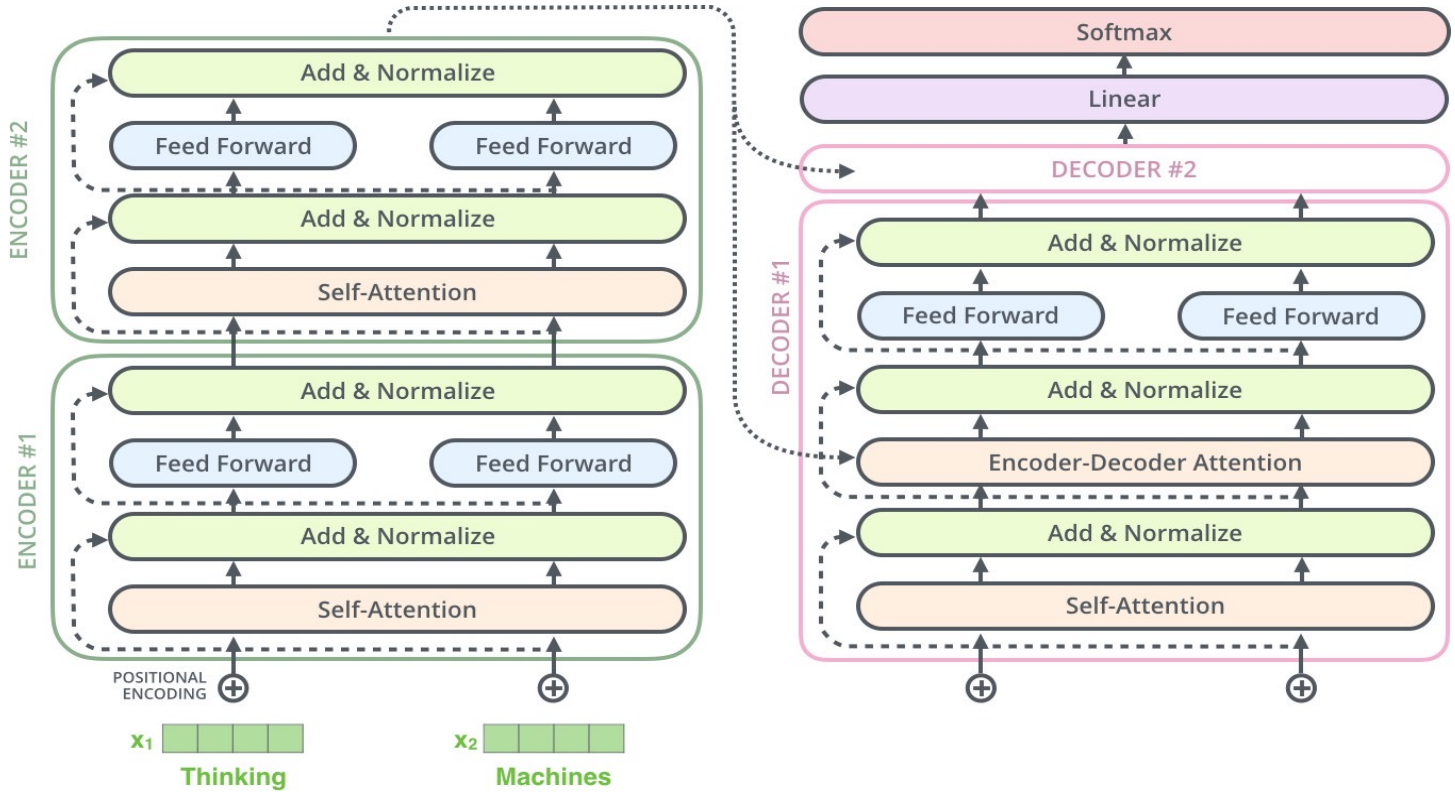
$$\text{ReLU}(x) = \max(0, x)$$



Adding and Normalizing again

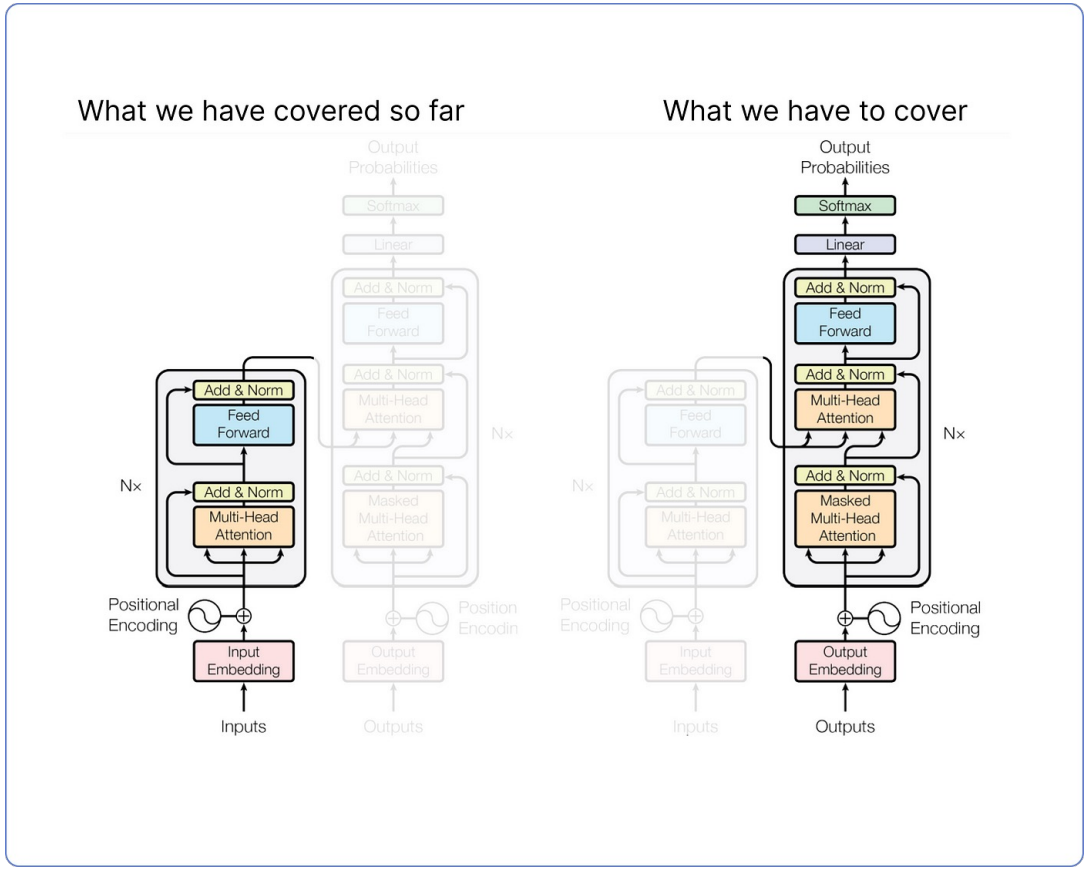
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Transformer Encoder-Decoder



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Decoder of Transformer

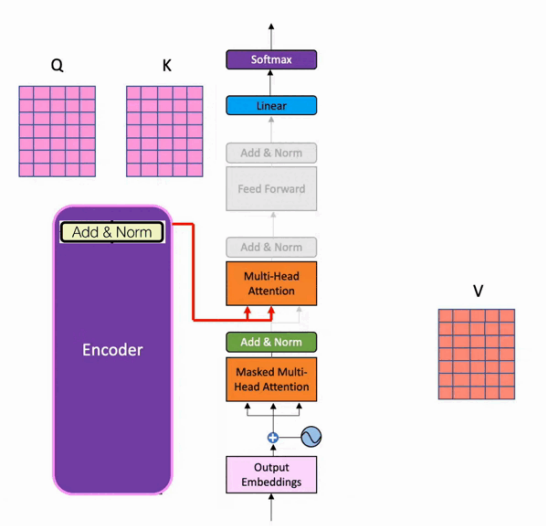


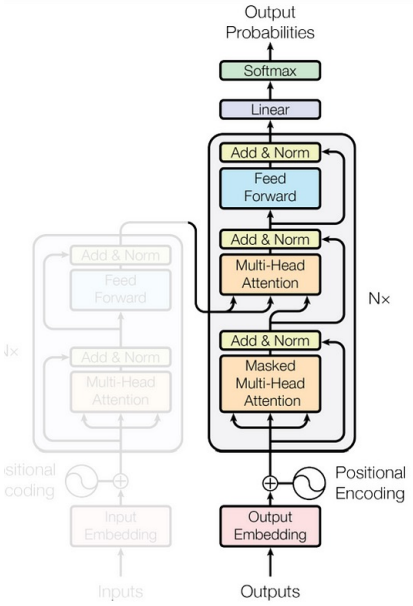
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Decoder of Transformer

When training, there are two inputs to the decoder.

- One is from the encoder, where the output matrix of the last add and norm layer serves as the **query** and **key** for the second multi-head attention layer in the decoder part.
- The value matrix comes from the decoder after the first add and norm step.





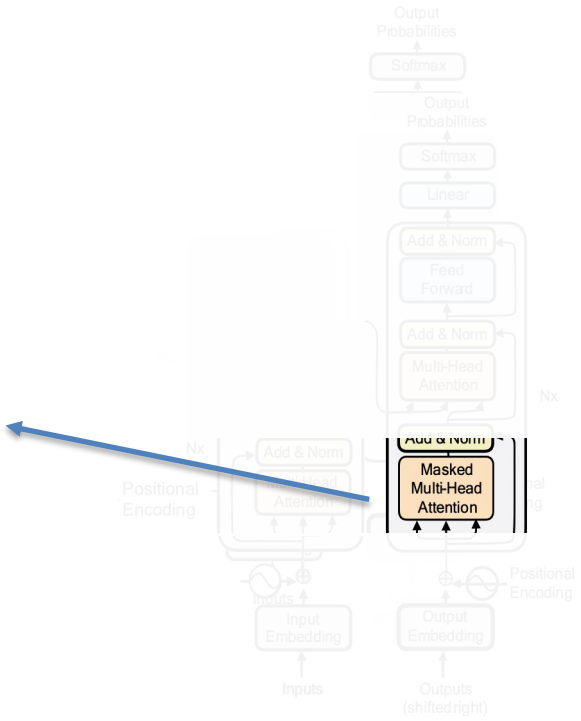
This is similar to cross attention mechanism we discussed in LSTM enc-dec

The difference is, here the value is coming from self (masked) attention instead of LSTM

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Decoder of Transformer – Masked attention

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	~~0.119~~	~~0.134~~	~~0.148~~	~~0.179~~	~~0.152~~
CAT	0.124	0.278	~~0.201~~	~~0.128~~	~~0.154~~	~~0.115~~
IS	0.147	0.132	0.262	~~0.097~~	~~0.218~~	~~0.145~~
A	0.210	0.128	0.206	0.212	~~0.119~~	~~0.125~~
LOVELY	0.146	0.158	0.152	0.143	0.227	~~0.174~~
CAT	0.195	0.114	0.203	0.103	0.157	0.229



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Decoder of Transformer – Masked attention

Our goal is to make the model causal (during training stage as the same situation will be encountered during Inference):

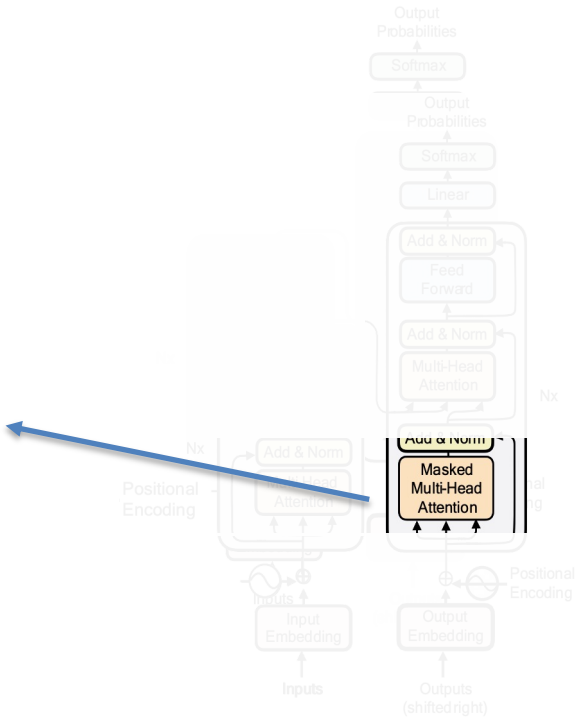
it means the output at a certain position can only depend on the words on the previous positions. The model **must not** be able to see future words.

To use self-attention in decoders, 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 -∞

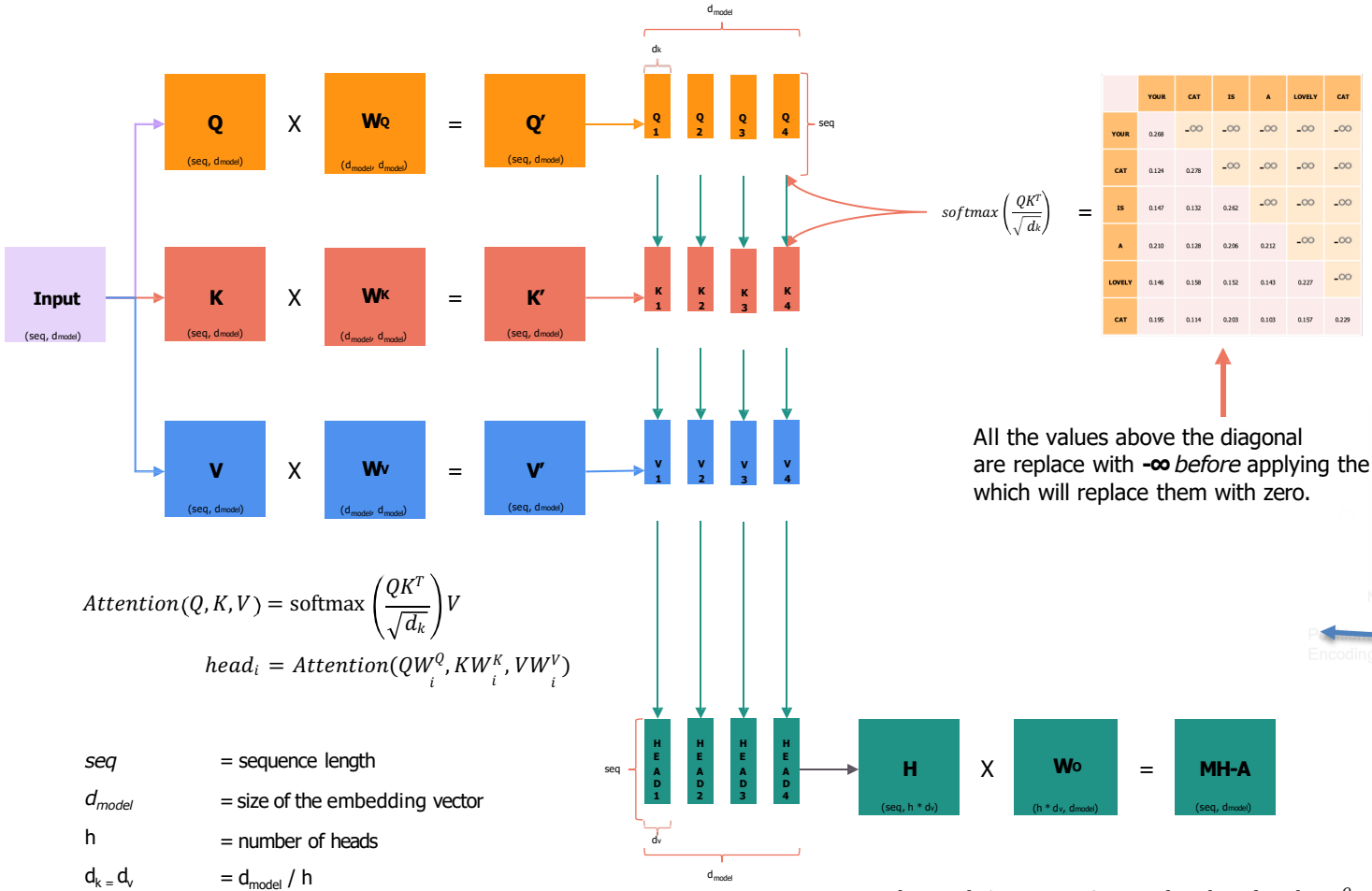
softmax will turn **-∞** *to* zero

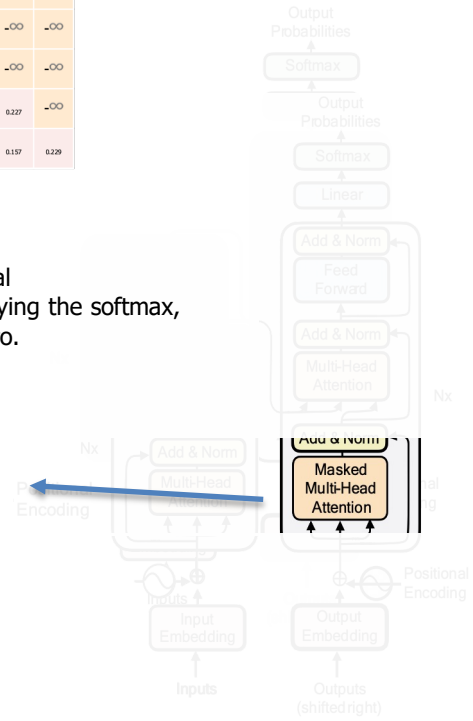
	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	-∞	-∞	-∞	-∞	-∞
CAT	0.124	0.278	-∞	-∞	-∞	-∞
IS	0.147	0.132	0.262	-∞	-∞	-∞
A	0.210	0.128	0.206	0.212	-∞	-∞
LOVELY	0.146	0.158	0.152	0.143	0.227	-∞
CAT	0.195	0.114	0.203	0.103	0.157	0.229



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Decoder of Transformer – Masked Multi-headed Attention





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

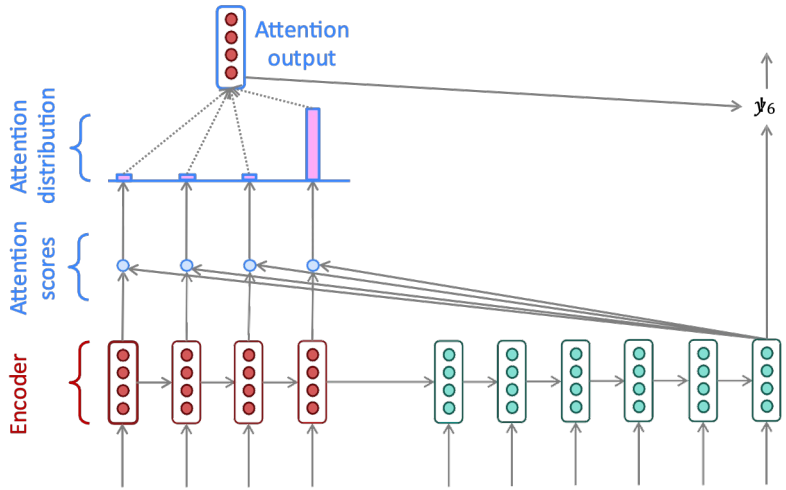
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

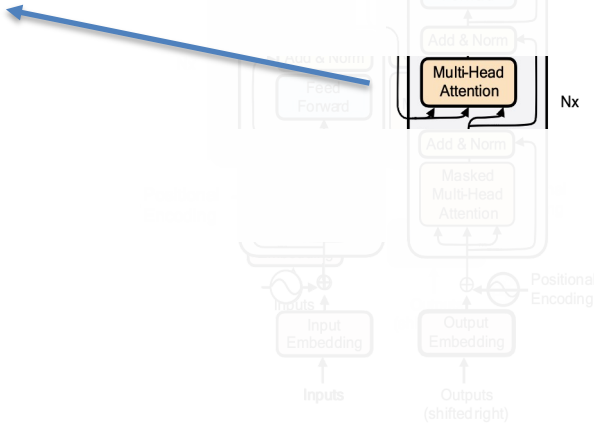
$$MultiHead(Q, K, V) = Concat(head_1 \ \ldots head_h)W^O$$

seq	= sequence length
d_{model}	= size of the embedding vector
h	= number of heads
$d_k = d_v$	= d_{model} / h

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Decoder of Transformer – Multi-headed attention Similar to RNN based NMT



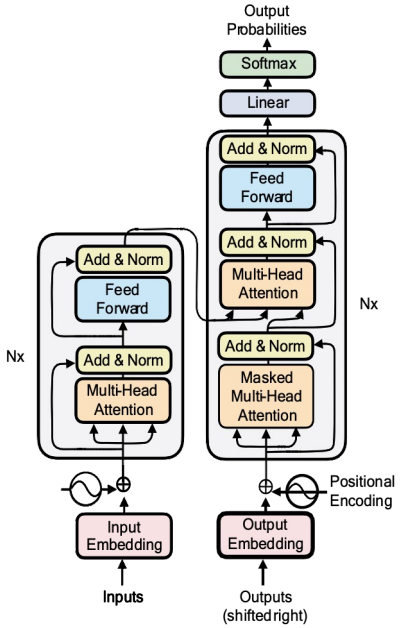


Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Transformer Training

I love you very much

Ti amo molto



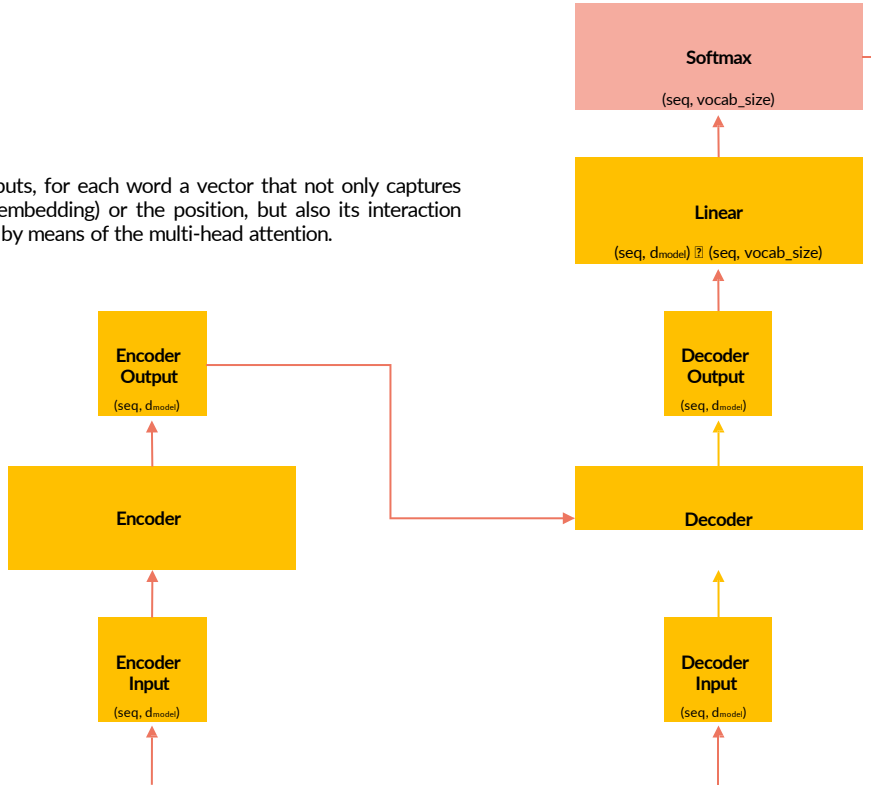
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

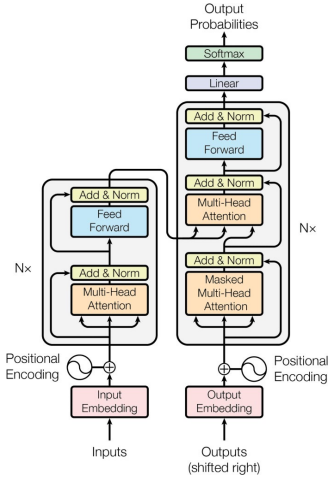
Transformer Training

Ti amo molto <EOS>

Cross Entropy Loss at each word level

The encoder outputs, for each word a vector that not only captures its meaning (the embedding) or the position, but also its interaction with other words by means of the multi-head attention.





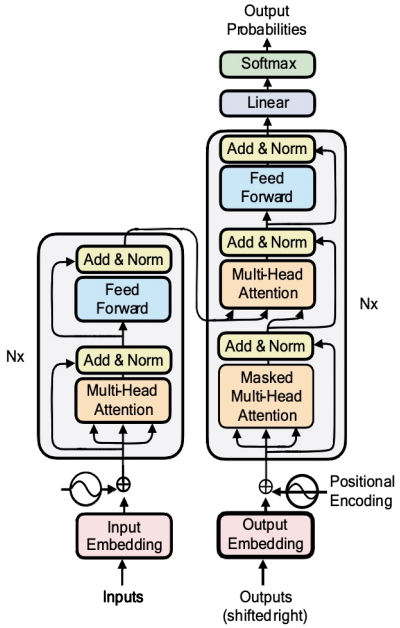
We prepend the <SOS> token at the beginning.

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Transformer Inference

I love you very much

Ti amo molto

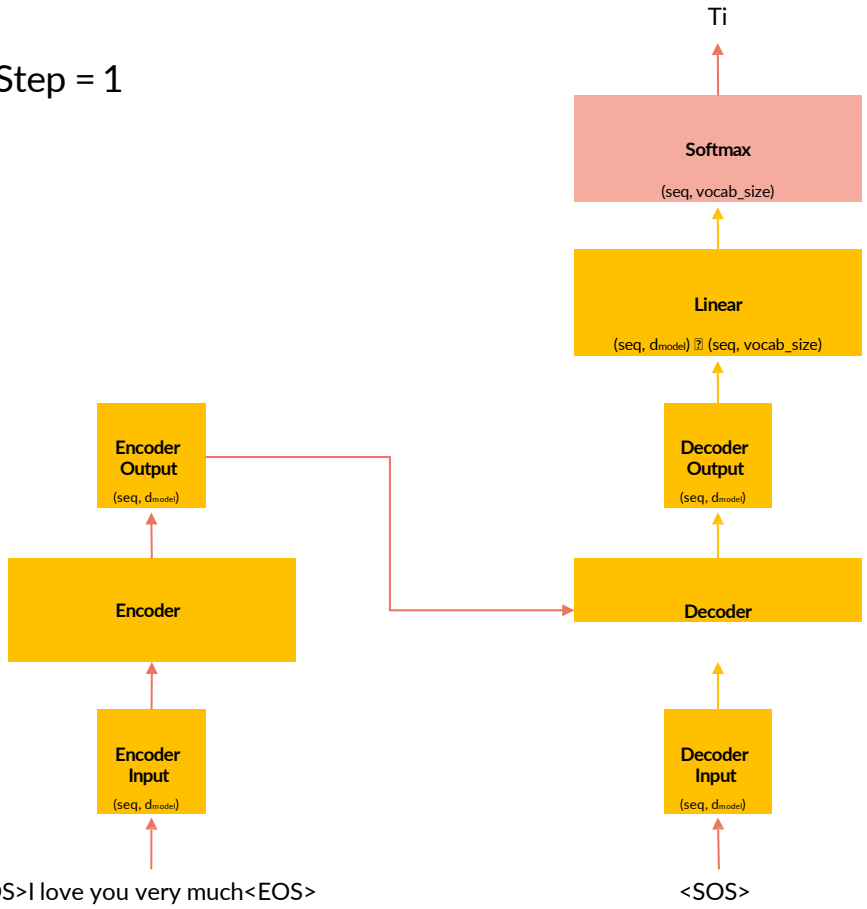


- We selected, at every step, the word with the maximum softmax value. This strategy is called **greedy** and usually does not perform very well.
- A better strategy is to select at each step the top *B* words and evaluate all the possible next words for each of them and at each step, keeping the top *B* most probable sequences. This is the **Beam Search** strategy and generally performs better.

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

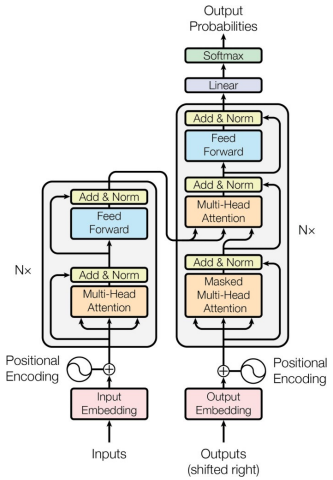
Word by Word Inference

Time Step = 1



We select a token from the vocabulary corresponding to the position of the token with the maximum logit value.

The output of the last layer is commonly known as **logits**

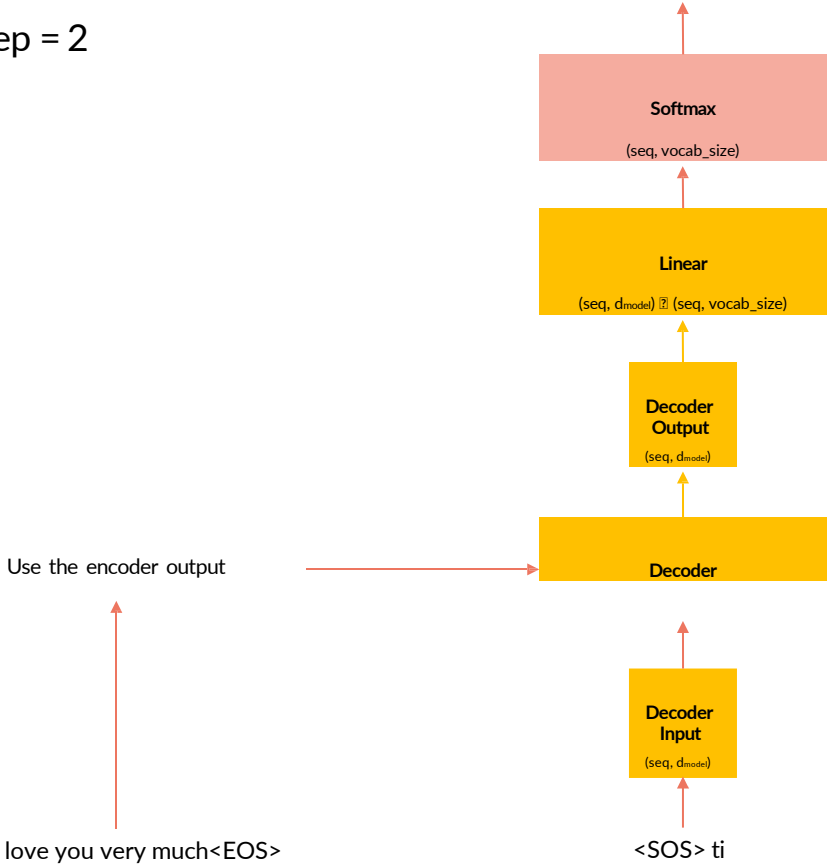


* Both sequences will have same length thanks to padding

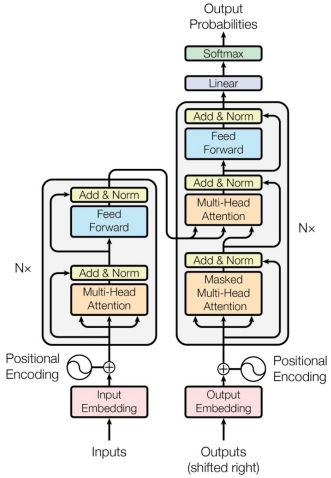
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Word by Word Inference

Time Step = 2



Since decoder input now contains **two** tokens, we select the softmax corresponding to the second token.

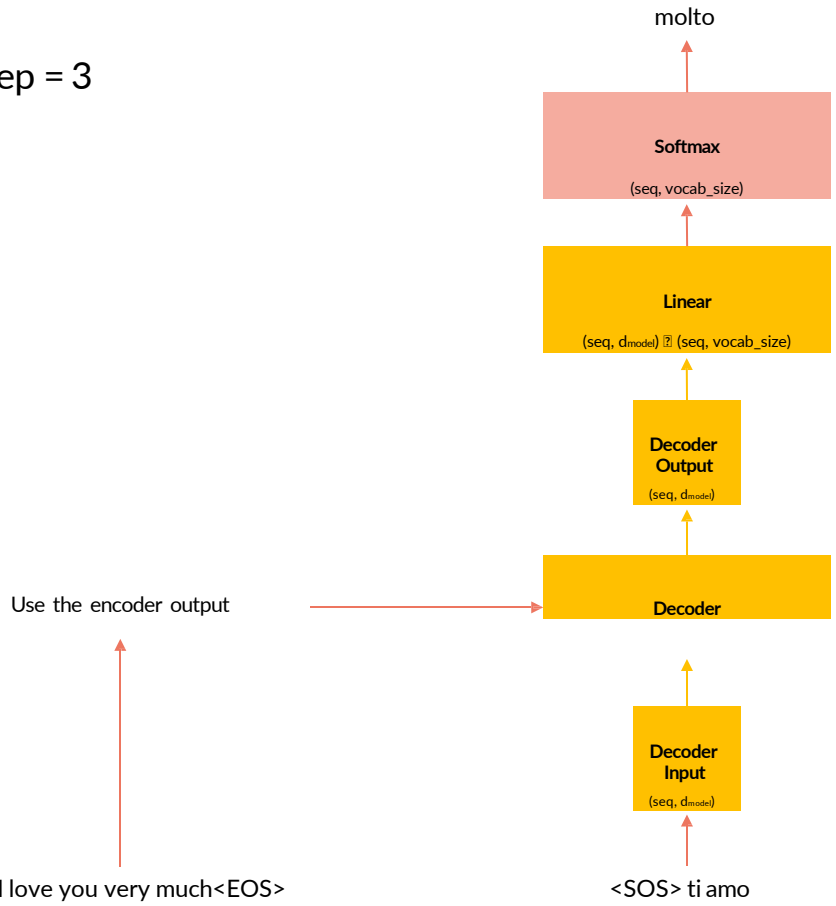


Append the previously output word to the decoder input

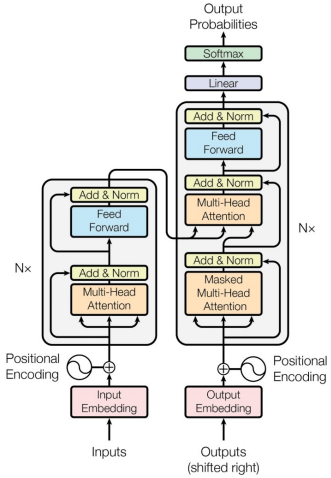
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Word by Word Inference

Time Step = 3



Since decoder input now contains **three** tokens, we select the softmax corresponding to the third token.

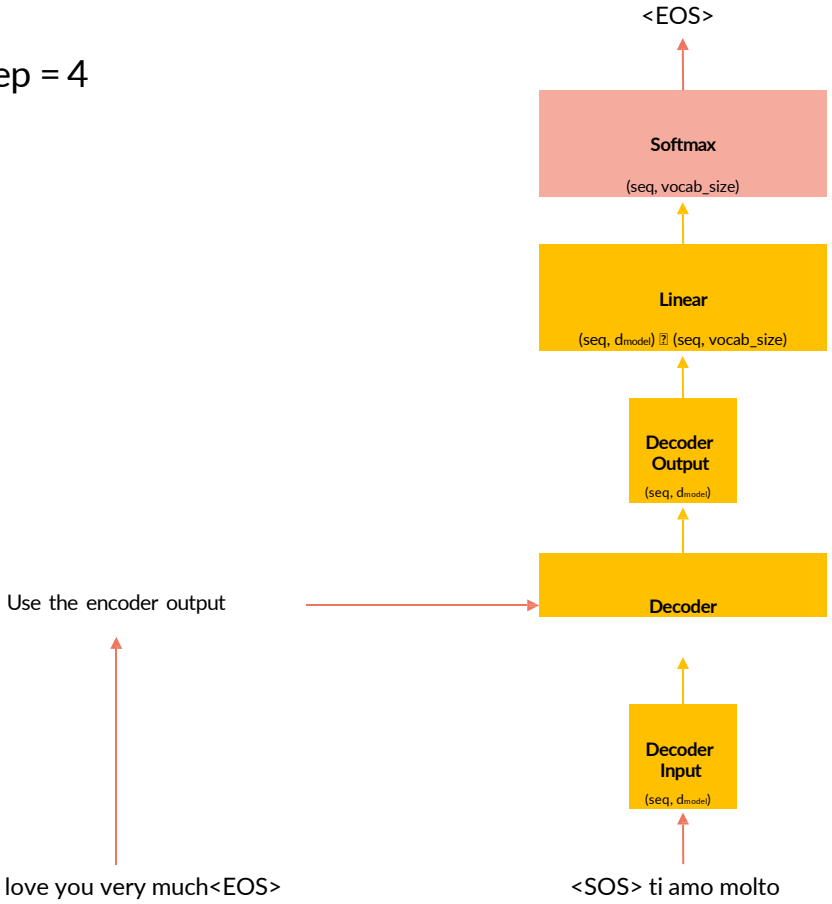


Append the previously output word to the decoder input

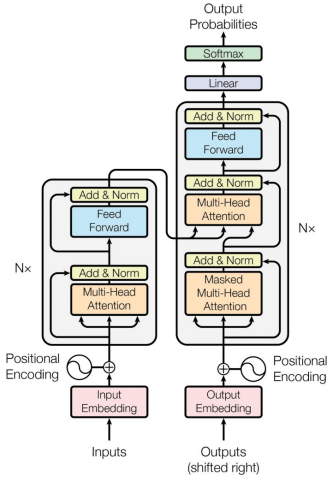
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Word by Word Inference

Time Step = 4



Since decoder input now contains **four** tokens, we select the softmax corresponding to the fourth token.

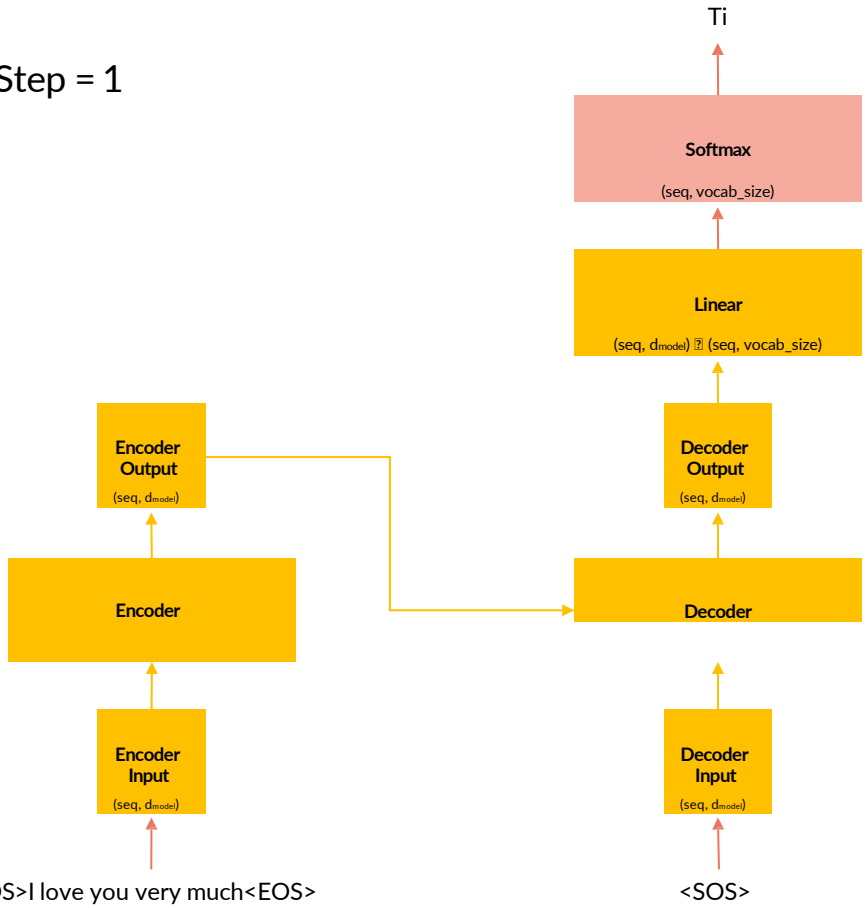


Append the previously output word to the decoder input

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Word by Word Inference

Time Step = 1

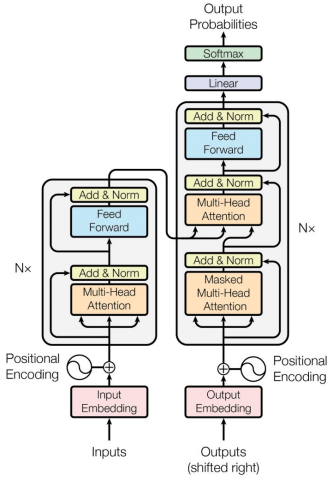


We select a token from the vocabulary corresponding to the position of the token with the maximum logit value. – GREEDY SEARCH

$$wt = argmax\ w\ P(w|w1{:}t-1)$$

major issue where words with high probabilities can be masked by words in front of them with low probabilities, so the model is unable to explore more diverse combinations of words.

The output of the last layer is commonly known as **logits**

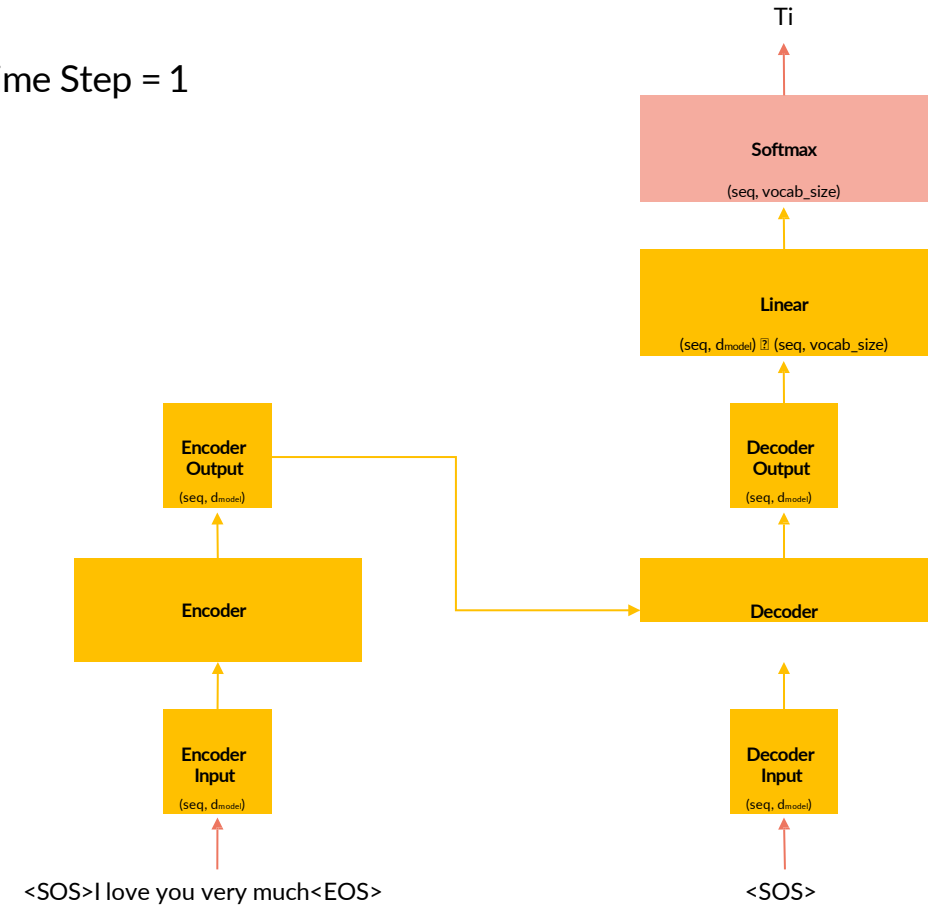


* Both sequences will have same length thanks to padding

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Word by Word Inference

Time Step = 1



We select a token from the vocabulary corresponding to the position of the token with the maximum logit value. – GREEDY SEARCH

$$wt = argmax\ w\ P(w|w1{:}t-1)$$

major issue where words with high probabilities can be masked by words in front of them with low probabilities, so the model is unable to explore more diverse combinations of words.

POSSIBLE SOLUTION – EXHAUSTIVE SEARCH

If the goal is to obtain the most likely sequence, we may consider using exhaustive search: enumerate all the possible output sequences with their conditional probabilities, and then output the one that scores the highest predicted probability.

$$wt \sim P(w|w1{:}t-1)$$

Unbearable Latency

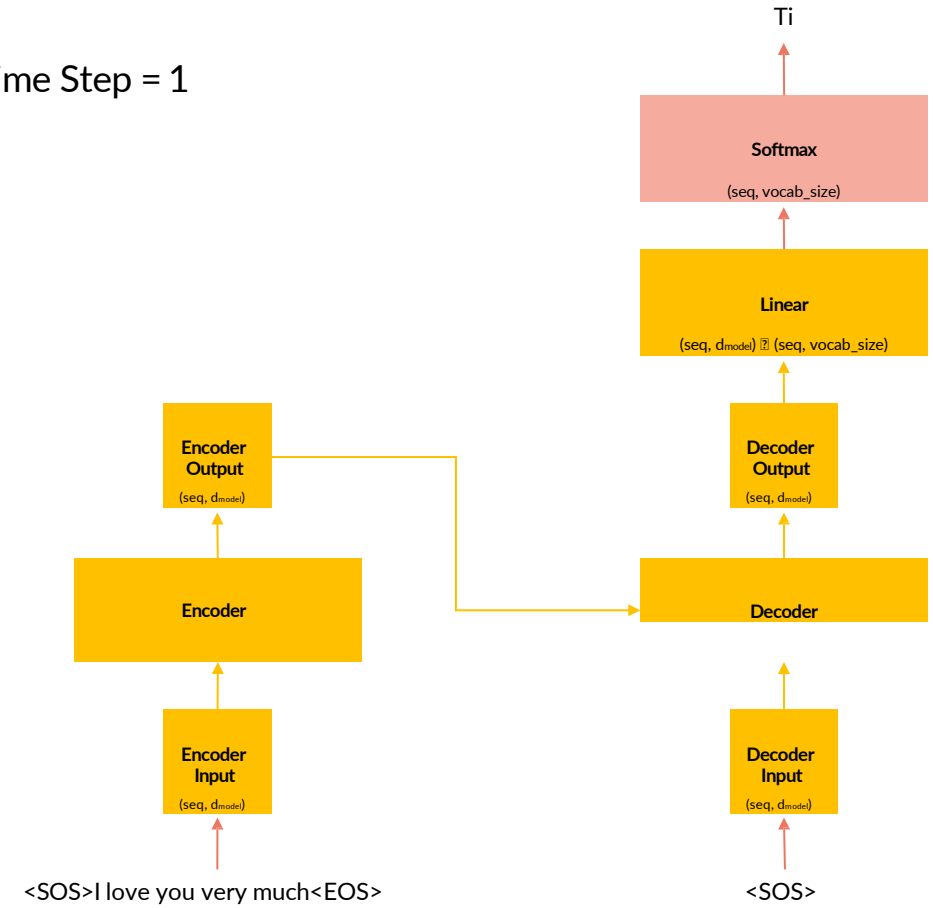
COMPROMISE– BEAM SEARCH

characterized by a single hyperparameter, the *beam size*, *K*. Let's explain this terminology. At time step 1, we select the *K* tokens with the highest predicted probabilities. Each of them will be the first token of *K* candidate output sequences, respectively. At each subsequent time step, based on the *K* candidate output sequences at the previous time step, we continue to select *K* candidate output sequences with the highest predicted probabilities different possible choices.

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Word by Word Inference

Time Step = 1



COMPROMISE– BEAM SEARCH

characterized by a single hyperparameter, the *beam size*, *K*. Let's explain this terminology. At time step 1, we select the *K* tokens with the highest predicted probabilities. Each of them will be the first token of *K* candidate output sequences, respectively. At each subsequent time step, based on the *K* candidate output sequences at the previous time step, we continue to select *K* candidate output sequences with the highest predicted probabilities different possible choices.

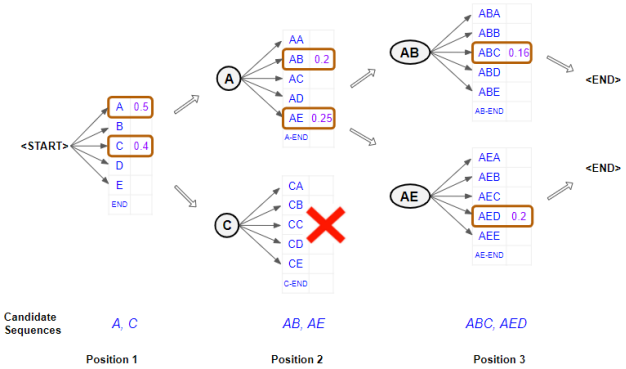
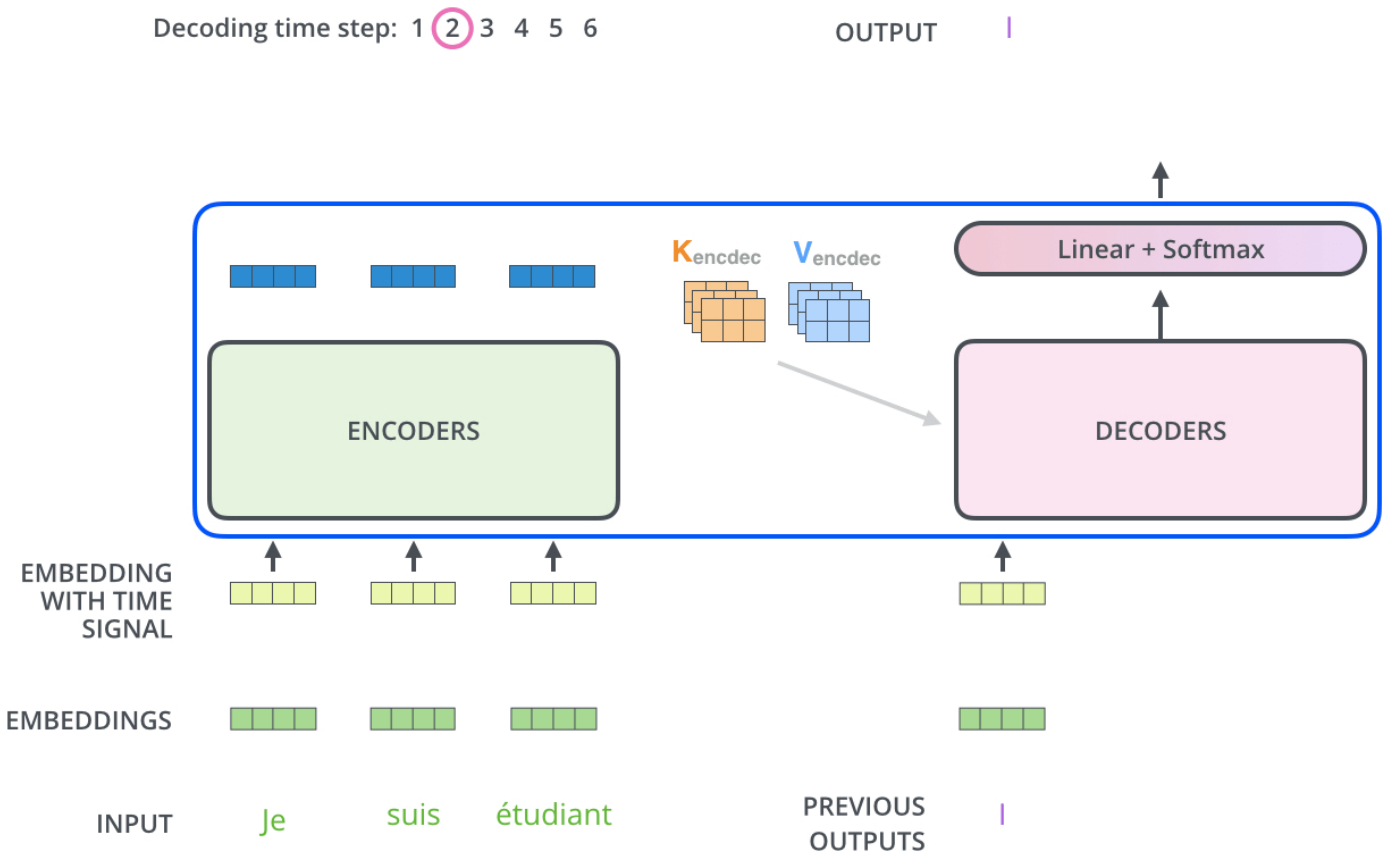


Fig. Beam Search example, with width = 2

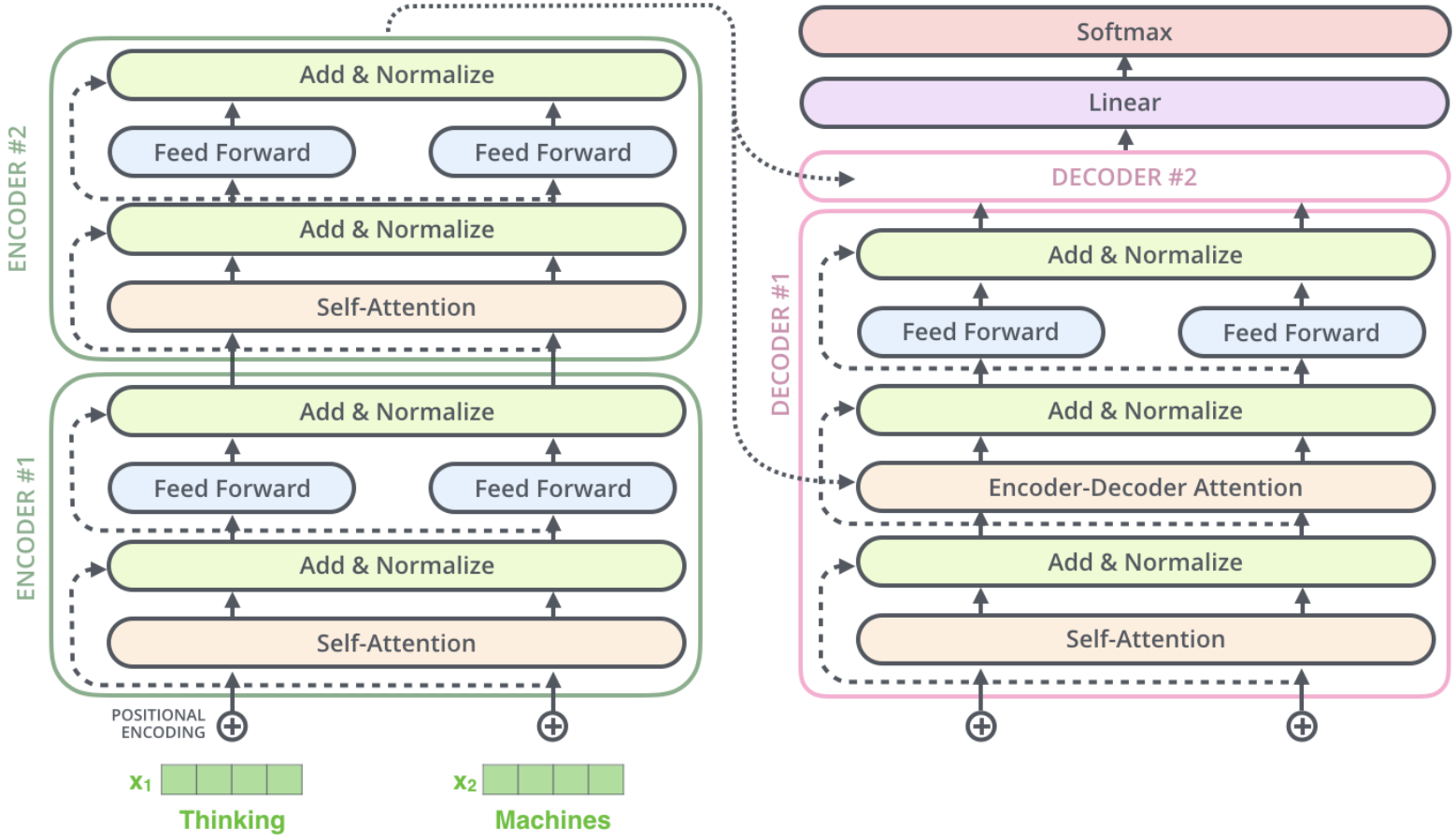
Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Transformers Inference/Decoding



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

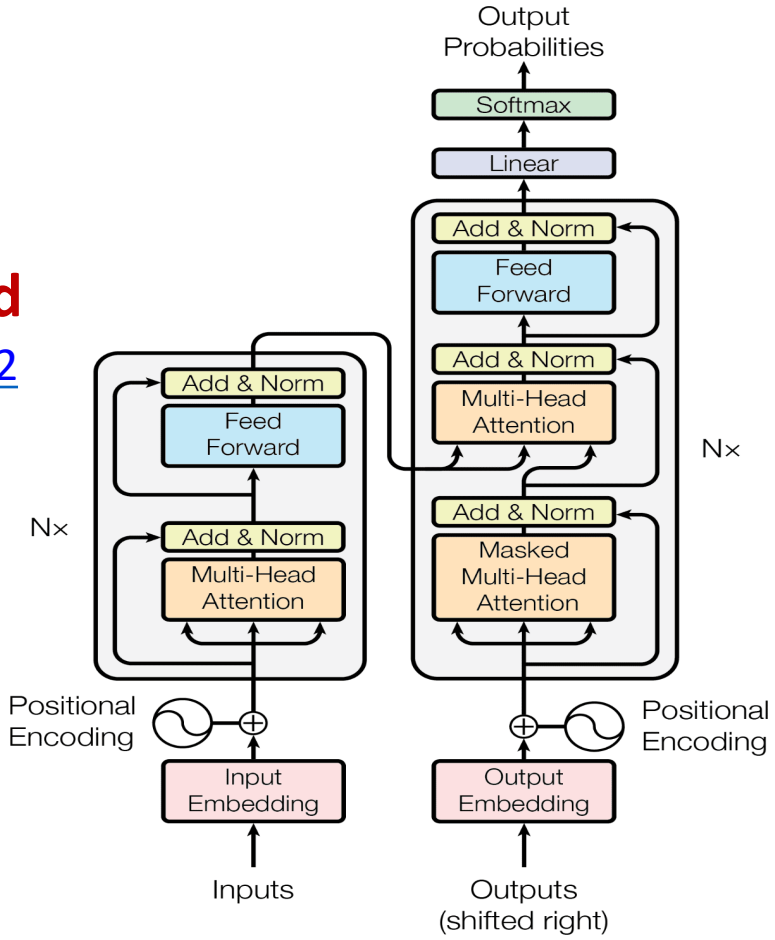
Transformers Overall



Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Attention Is All You Need

https://arxiv.org/abs/1706.03762



How do you Evaluate Quality of Generated Text

Can we use **Precision?**

Target Sentence: He eats an apple
Predicted Sentence: He ate an apple

Precision = Number of correct predicted words / Number of total predicted words
Precision = 3 / 4

Issue: Repetition
Predicted Sentence: He He He
Precision = 3 / 3 = 1

Another Issue: There can be multiple Target sentences, this metric will change for different possible targets

Alternative – **clipped Precision?**

- limit the count for each correct word to the maximum number of times that that word occurs in the Target Sentence.
- compare each word from the predicted sentence with all of the target sentences. If the word matches any target sentence, it is considered to be correct.

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

How do you Evaluate Quality of Generated Text

Alternative – clipped Precision?

limit the count for each correct word to the maximum number of times that that word occurs in the Target Sentence.
compare each word from the predicted sentence with all of the target sentences. If the word matches any target sentence, it is correct.

Target Sentence1: He eats an apple
Target Sentence2: He is eating a tasty apple
Predicted Sentence: He He He eats tasty fruit

Word	*Matching Sentence*	*Matched Predicted Count*	*Clipped Count*
He	*Both*	*3*	*1*
eats	*Target 1*	*1*	*1*
tasty	*Target 2*	*1*	*1*
fruit	*None*	*0*	*0*
Total		***5***	***3***

Clipped Precision = Clipped number of correct predicted words / Number of total predicted words

Clipped Precision = 3 / 6

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

How do you Evaluate Quality of Generated Text

Alternative – clipped Precision?

limit the count for each correct word to the maximum number of times that that word occurs in the Target Sentence.
compare each word from the predicted sentence with all of the target sentences. If the word matches any target sentence, it is correct.

Target Sentence1: He eats an apple
Target Sentence2: He is eating a tasty apple
Predicted Sentence: He He He eats tasty fruit

Word	*Matching Sentence*	*Matched Predicted Count*	*Clipped Count*
He	*Both*	*3*	*1*
eats	*Target 1*	*1*	*1*
tasty	*Target 2*	*1*	*1*
fruit	*None*	*0*	*0*
Total		***5***	***3***

Clipped Precision = Clipped number of correct predicted words / Number of total predicted words

Clipped Precision = 3 / 6

Bleu Score – Clipped Precision for n-grams

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

How do you Evaluate Quality of Generated Text – Bleu Score

Bleu Score – Clipped Precision for n-grams

- **Target Sentence**: The guard arrived late because it was raining
- **Predicted Sentence**: The guard arrived late because of the rain

Clipped Precision 1-gram

Clipped Precision 1-gram = Number of correct predicted 1-grams / Number of total predicted 1-grams

Target Sentence: The guard arrived late because ~~it was raining~~

Predicted Sentence: The guard arrived late because of the rain

So, Clipped Precision 1-gram (p_1) = 5 / 8

Similarly, Clipped Precision 2-gram

Clipped Precision 2-gram = Number of correct predicted 2-grams / Number of total predicted 2-grams

Target Sentence: The guard arrived late because it was raining

Predicted Sentence: The guard arrived late because of the rain

So, Clipped Precision 2-gram (p_1) = 4 / 7

Sources: A
applied

How do you Evaluate Quality of Generated Text – Bleu Score

Similarly, calculate Clipped Precision n-gram, n=1,2,3,..N
Typically, *N = 4*

Geometric Average Precision Scores

$$\begin{aligned} Geometric\ Average\ Precision\ (N) &= exp(\sum_{n=1}^{N} w_n\ log\ p_n) \\ &= \prod_{n=1}^{N} {p_n}^{w_n} \\ &= (p_1)^{\frac{1}{4}} \cdot (p_2)^{\frac{1}{4}} \cdot (p_3)^{\frac{1}{4}} \cdot (p_4)^{\frac{1}{4}} \end{aligned}$$

Brevity Penalty
Penalty on predicted sentences which are too short

$$Brevity\ Penalty = \begin{cases} 1, & \text{if } c > r \\ e^{(1-r/c)}, & \text{if } c <= r \end{cases}$$

- c is *predicted length = number of words in the predicted sentence* and
- r is *target length = number of words in the target sentence*

Bleu Score

$$Bleu\ (N) = Brevity\ Penalty \cdot Geometric\ Average\ Precision\ Scores\ (N)$$

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Great Results with Transformers

Machine Translation from the original Transformers paper!

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$

[Test sets: WMT 2014 English-German and English-French]

[Vaswani et al., 2017]

Great Results with Transformers

document generation!

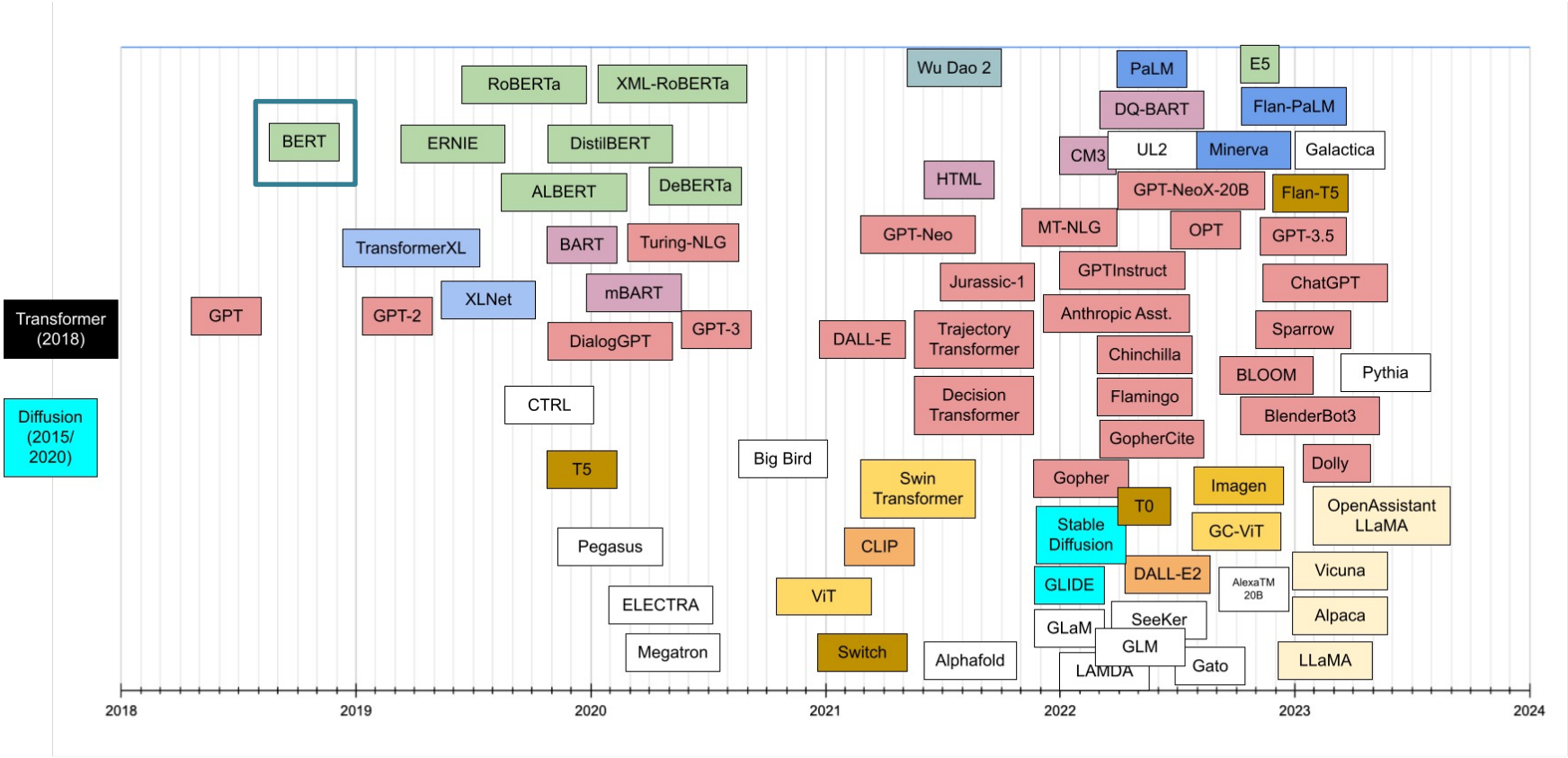
Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

The old standard

Transformers all the way down.

[Liu et al., 2018]; WikiSum dataset

Transformer evolution



BERT

Sources: Attribution and credit to various internet sourced and copyrighted content with edits applied

Language model Recap

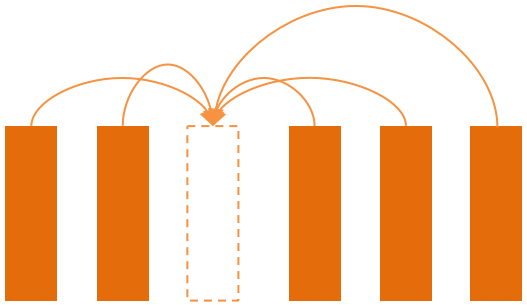
A language model is a probabilistic model that assign probabilities to sequence of words.
In practice, a language model allows us to compute the following:

P [“China” | “Shanghai is a city in”]

Next Token **Prompt**

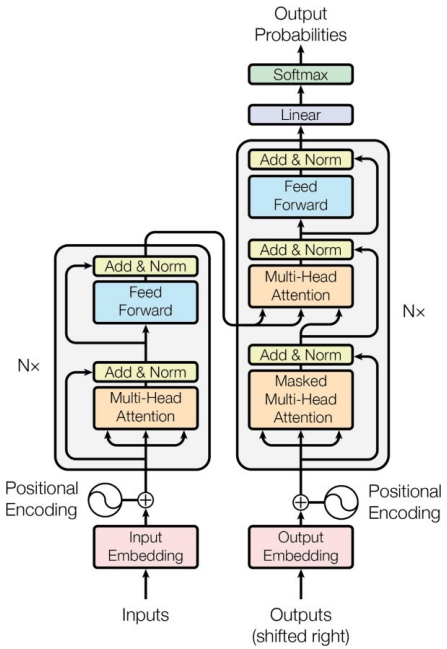
We usually train a neural network to predict these probabilities. A neural network trained on a large corpora of text is known as a Large Language Model (LLM).

But next token prediction doesn’t utilise full context of the sentence as self-attention



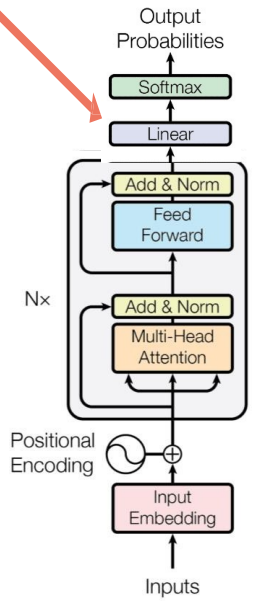
self attention

Transformer Encoder For LM



Transformer

Output layer depending on the specific task



Transformer Encoder

BERT

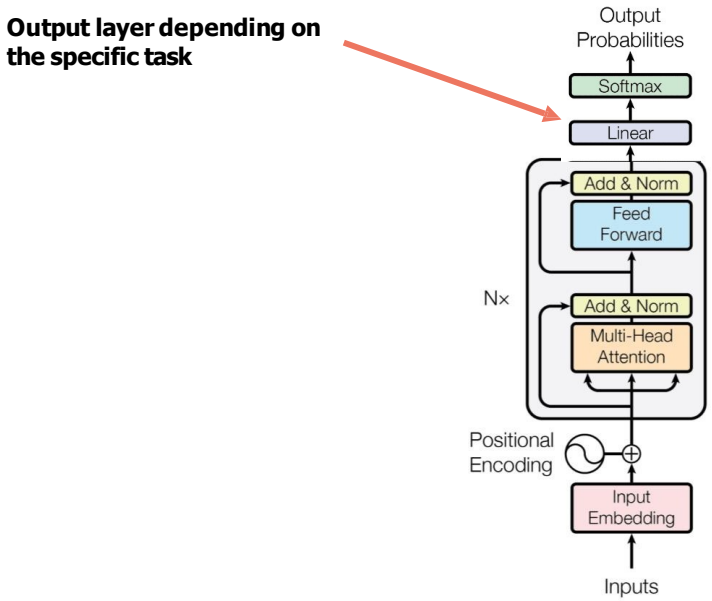
Introducing BERT

BERT's architecture is made up of layers of encoders of the Transformer model:

Differences between Bert and Transformer Encoder:

- The embedding vector is 768 and 1024 for the two models
- Positional embeddings are absolute and learnt during training and limited to 512 positions
- The linear layer head changes according to the application

Uses the **WordPiece** tokenizer, which also allows sub-word tokens. The vocabulary size is ~ 30,000 tokens.



Introducing BERT

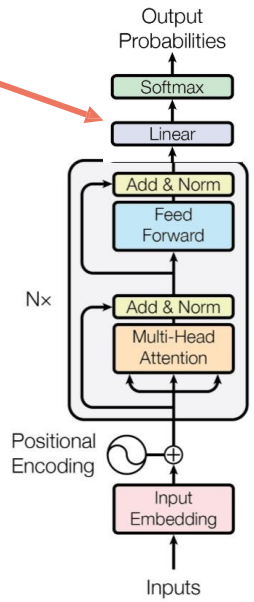
$BERT_{BASE}$

- 12 encoder layers
- The size of the hidden size of the feedforward layer is 3072
- 12 attention heads.

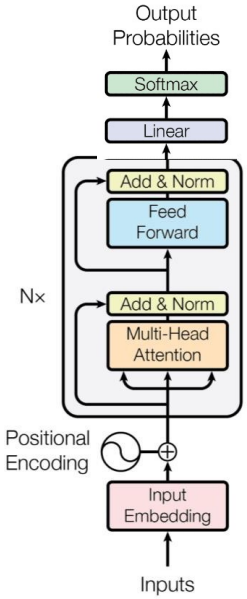
$BERT_{LARGE}$

- 24 encoder layers
- The size of the hidden size of the feedforward layer is 4096
- 16 attention heads.

Output layer depending on the specific task



BERT Pre-Training

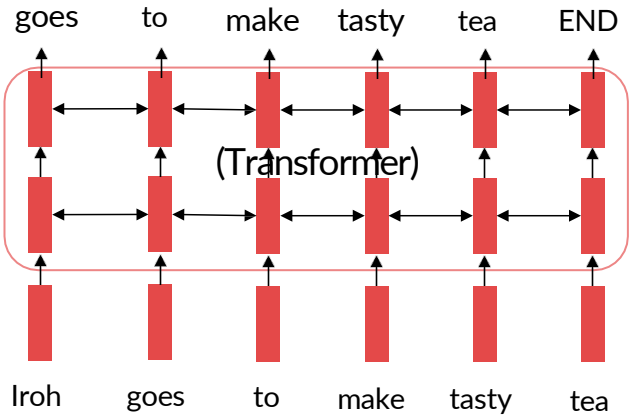


Pre-Training Objective

Pretraining can improve NLP applications by serving as parameter initialization.

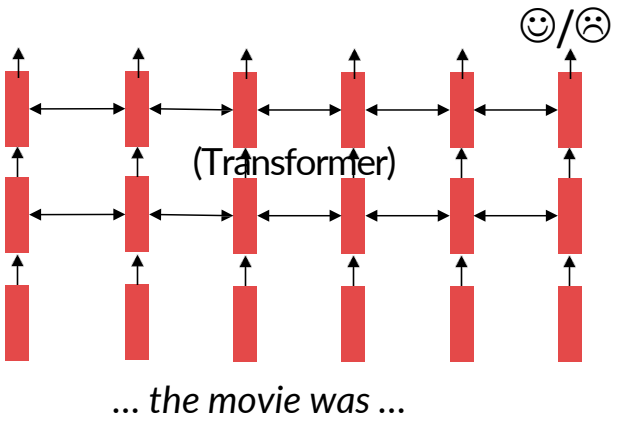
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!



<u>Pretraining through language modeling:</u>

- Train a neural network to reconstruct Input sentences
- Perform this on a large amount of text.
- Save the network parameters.

What can we learn from reconstructing the input?

Stanford University is located in _____, California.

I put _____ fork down on the table.

The woman walked across the street, checking for traffic over ___ shoulder.

I went to the ocean to see the fish, turtles, seals, and _____.

Masked Language Model (MLM)

Also known as the Cloze task. It means that randomly selected words in a sentence are masked, and the model must **predict the right word given the left and right context**.

Rome is the capital of Italy, which is why it hosts many government buildings.

Randomly select one or more tokens and replace them with the special token **[MASK]**

Rome is the [MASK] of Italy, which is why it hosts many government buildings.

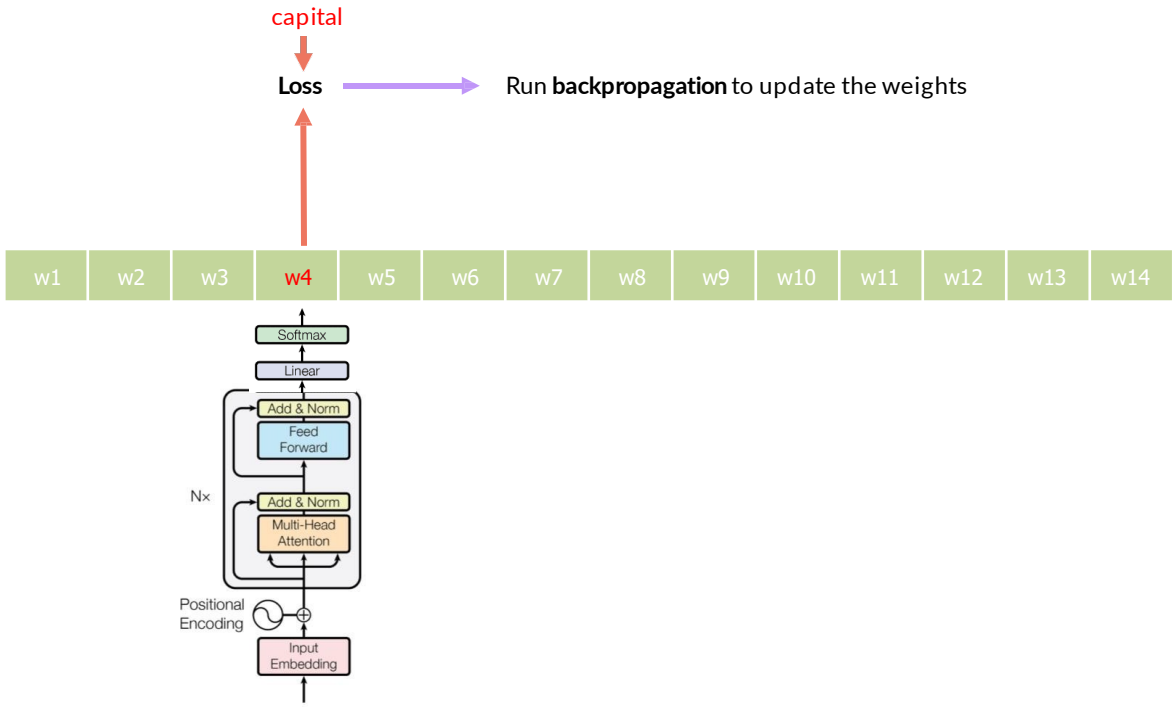
capital

BERT Training using MLM

Devlin et al., 2018 proposed the “Masked LM” objective in BERT paper and **released the weights**

Masked Language Model (MLM): training

Target (1 token):



Input (14 tokens): Rome is the [mask] of Italy, which is why it hosts many government buildings.

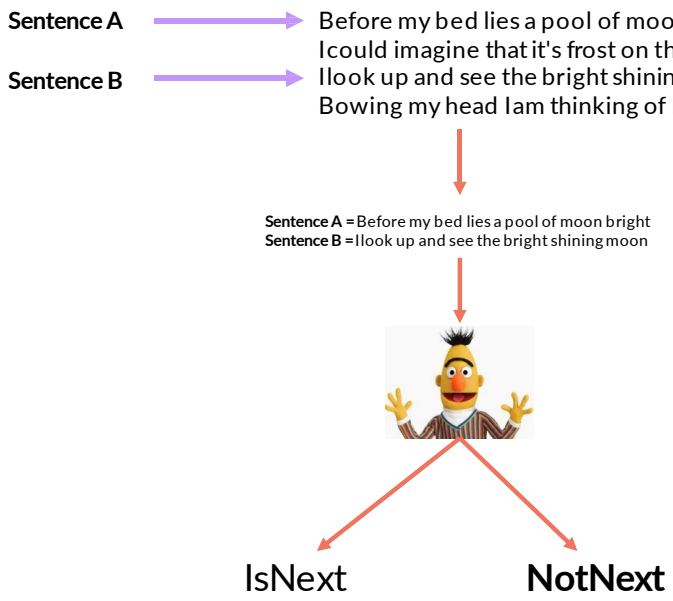
Masked Language Model (MLM): training

Some more details about Masked LM for BERT:

- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)

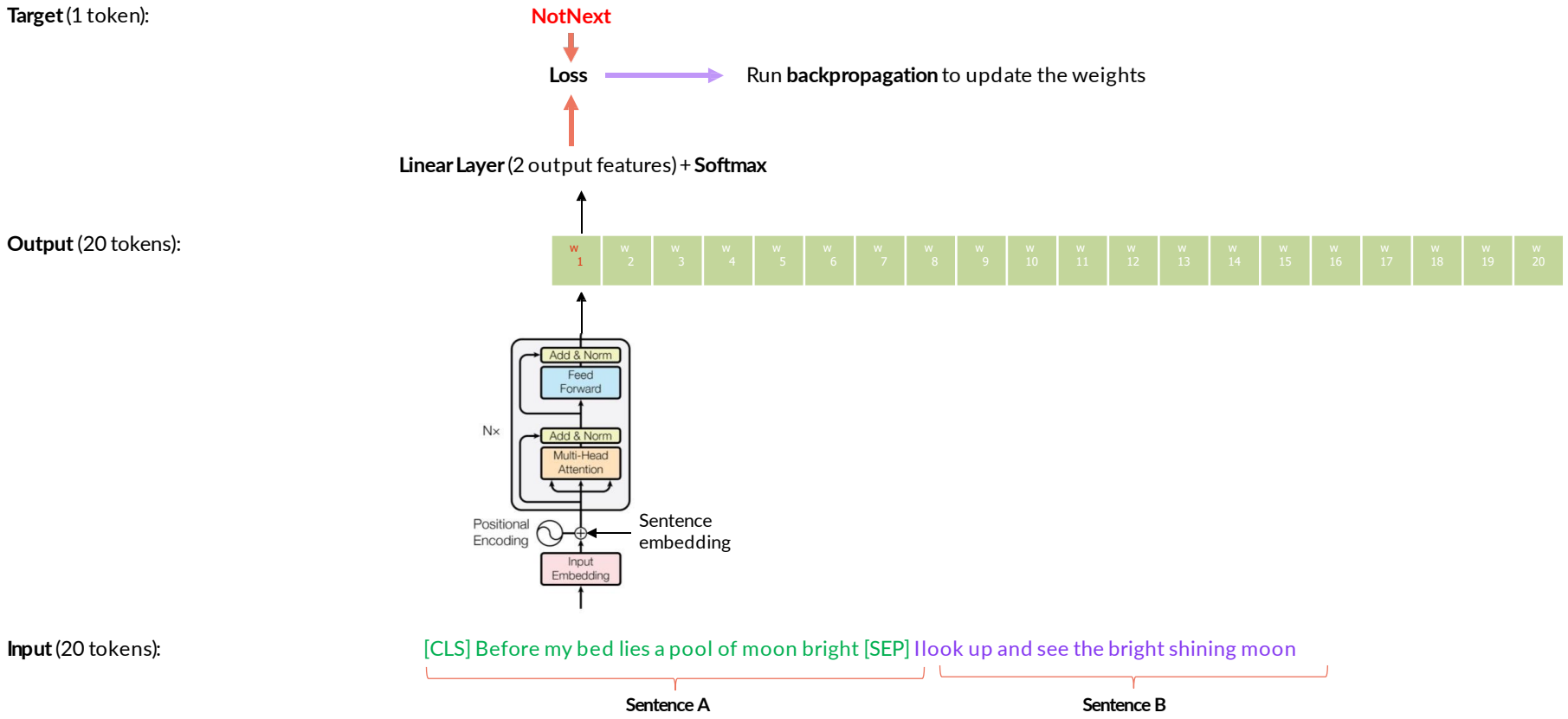
Next Sentence Prediction (NSP) for Training

Many downstream applications (for example choosing the right answer given 4 choices) require learning relationships between sentences rather than single tokens, that's why BERT has been pre-trained on the Next Sentence Prediction task.



- 50% of the time, we select the actual next sentence.
- 50% of the time we select a random sentence from the text.

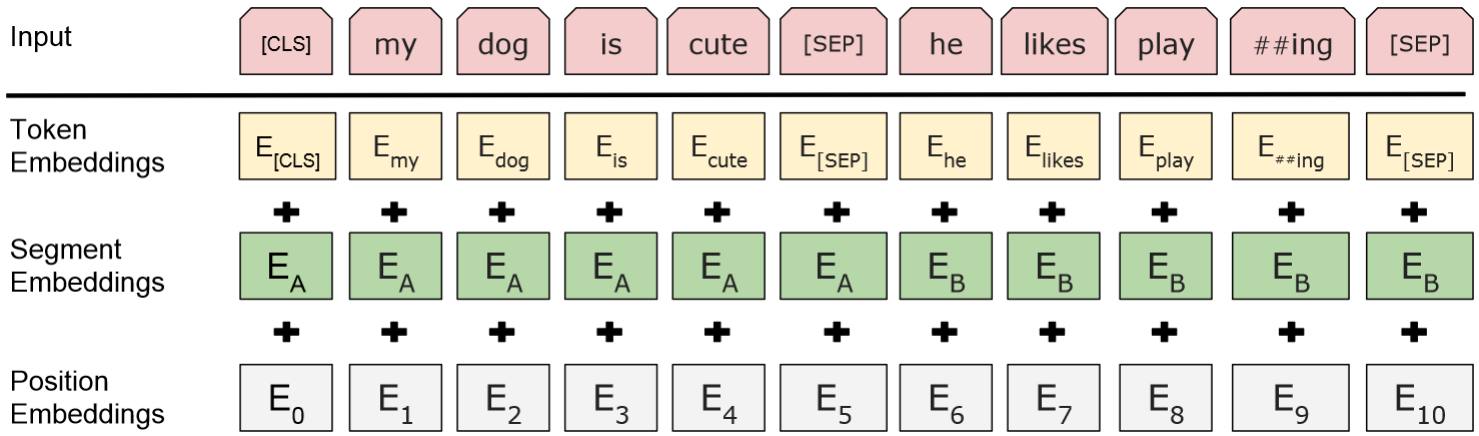
Next Sentence Prediction (NSP) for Training



Changes for NSP

Given the sentence A and the sentence B, how can BERT understand which tokens belongs to the sentence A and which to the sentence B? We introduce the segmentation embeddings!

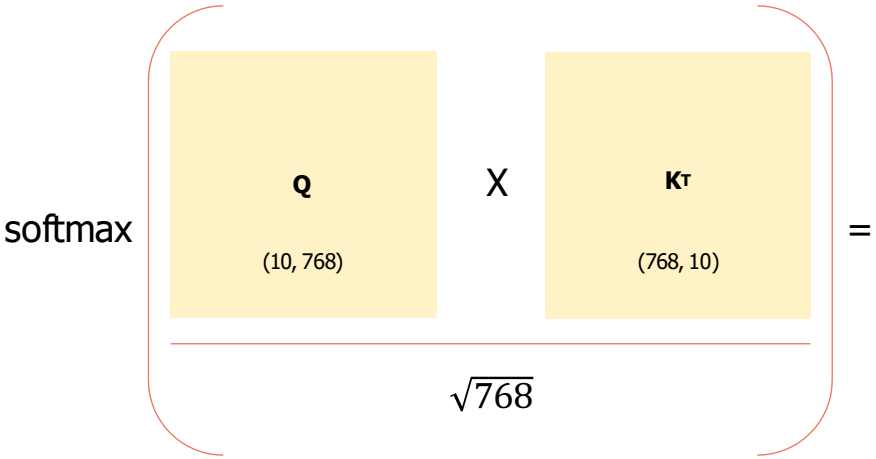
We also introduce two special tokens: **[CLS]** and **[SEP]**



[CLS] token in BERT

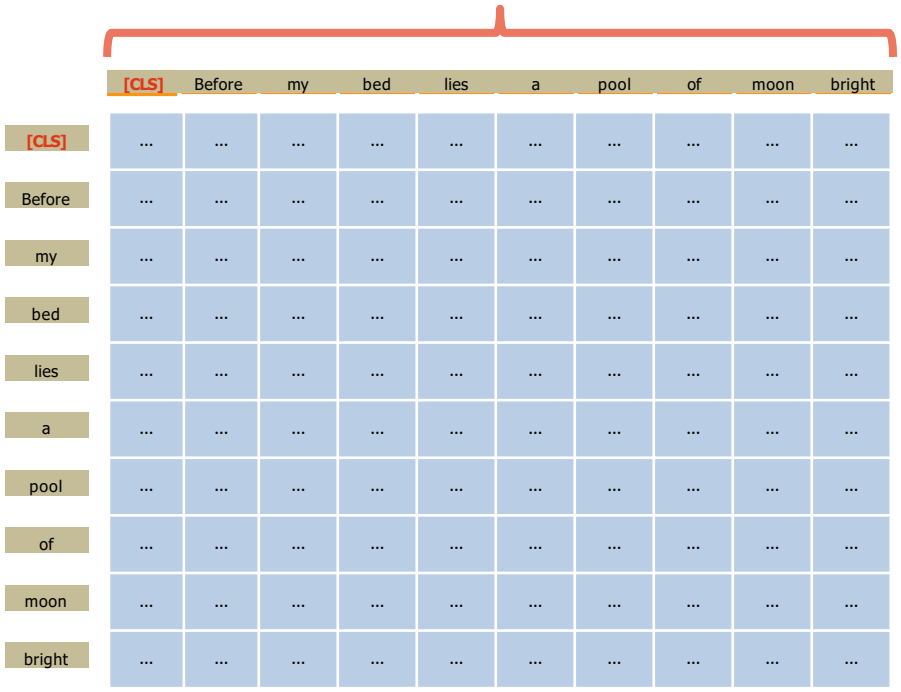
The **[CLS]** token always interacts with all the other tokens, as we do not use any mask.
So, we can consider the **[CLS]** token as a token that "captures" the information from all the other tokens.

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



	[CLS]	Before	my	bed	lies	a	pool	of	moon	bright
[CLS]	...	...	...	...	...	...	...	...	...	...
Before	...	...	...	...	...	...	...	...	...	...
my	...	...	...	...	...	...	...	...	...	...
bed	...	...	...	...	...	...	...	...	...	...
lies	...	...	...	...	...	...	...	...	...	...
a	...	...	...	...	...	...	...	...	...	...
pool	...	...	...	...	...	...	...	...	...	...
of	...	...	...	...	...	...	...	...	...	...
moon	...	...	...	...	...	...	...	...	...	...
bright	...	...	...	...	...	...	...	...	...	...

[CLS] token: output sequence



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

X

V

(10, 768)

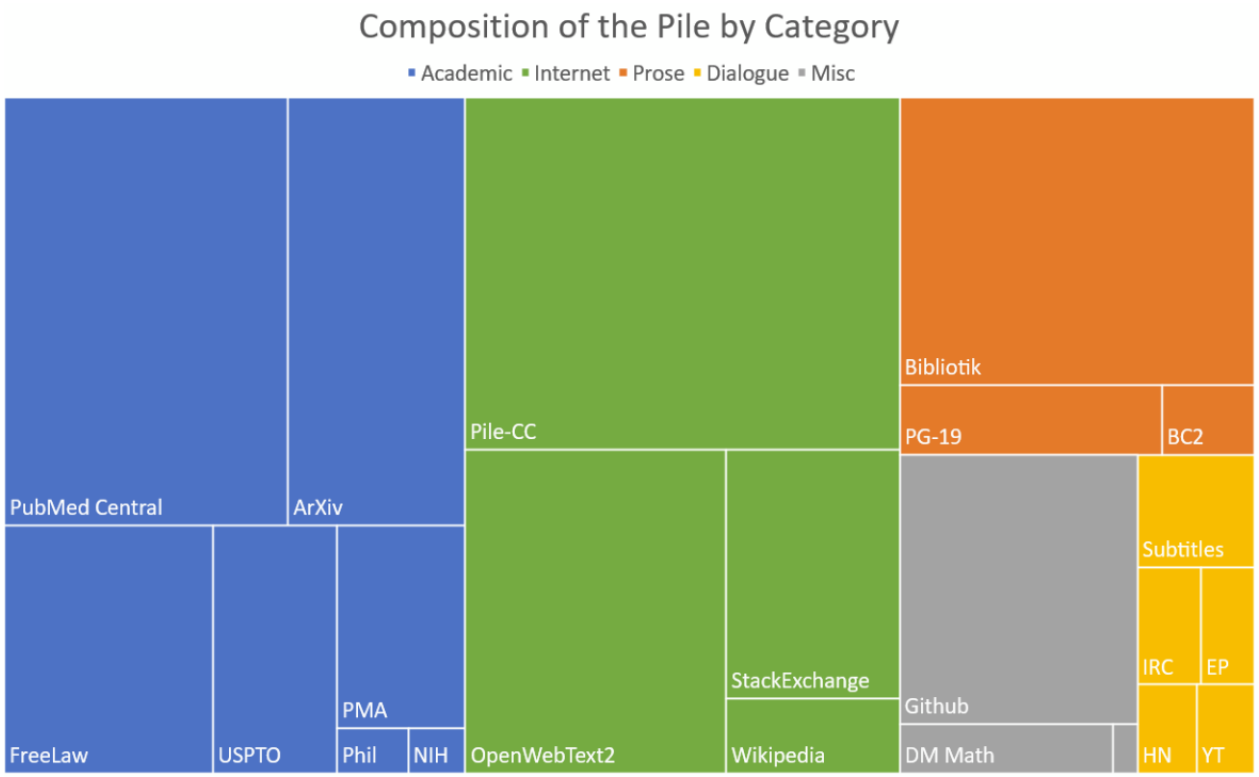
=

Attention Output

(10, 768)

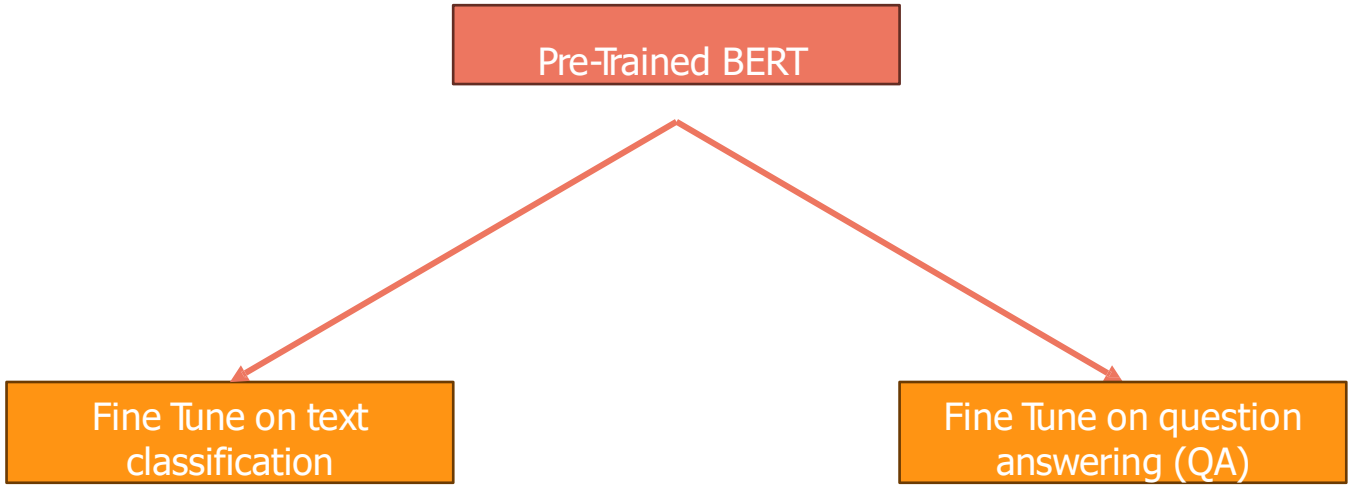
Each row of the "Attention Output" matrix represents the embedding of the output sequence: it captures not only the meaning of each token, not only its position, but also the interaction of each token with all the other tokens, but only the interactions for which the softmax score is not zero. All the 512 dimensions of each vector only depend on the attention scores that are non-zero.

Training Data

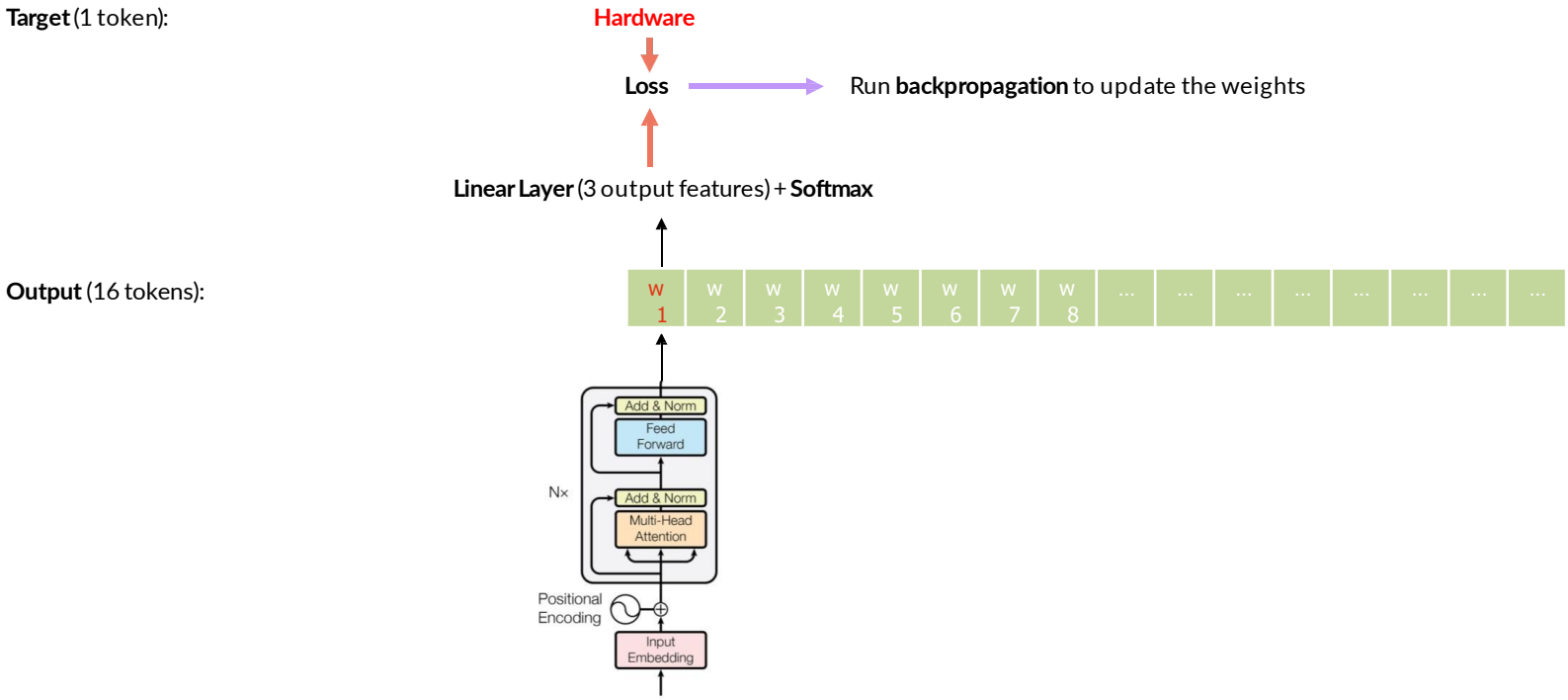


Model	Training Data
BERT	BookCorpus, English Wikipedia
GPT-1	BookCorpus
GPT-3	CommonCrawl, WebText, English Wikipedia, and 2 book databases (“Books 1” and “Books 2”)
GPT-3.5 +	Undisclosed

Fine-Tuning BERT for Downstream Tasks



Text Classification



BERT: Bidirectional Encoder Representations from Transformers

- Two models were released:
  - BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
  - BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.

- Trained on:
  - BooksCorpus (800 million words)
  - English Wikipedia (2,500 million words)

- Pretraining is expensive and impractical on a single GPU.
  - BERT was pretrained with 64 TPU chips for a total of 4 days.
  - (TPUs are special tensor operation acceleration hardware)

- Finetuning is practical and common on a single GPU
  - “Pretrain once, finetune many times.”

BERT: Bidirectional Encoder Representations from Transformers

finetuning BERT led to new state-of- the-art results on a broad range of tasks.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

QQP: Quora Question Pairs (detect paraphrase questions)

QNLI: natural language inference over question answering data

SST-2: sentiment analysis

CoLA: corpus of linguistic acceptability (detect whether sentences are grammatical.)

STS-B: semantic textual similarity

MRPC: microsoft paraphrase corpus

RTE: a small natural language inference corpus

Popular BERT Extensions

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

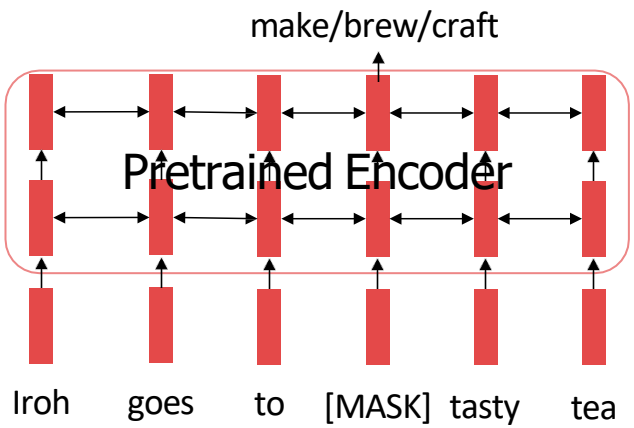
Model	SST-2	QQP	MNLI-m/mm	QNLI	Avg.
$\text{BERT}_{\text{BASE}}$	93.7	71.5	84.8 / 83.8	91.3	85.0
DistilBERT†	92.5	70.1	82.6 / 81.3	88.9	83.1
+ Vanilla KD	92.8	70.4	83.1 / 81.8	90.0	83.6
+ BERT-PKD	92.9	**71.3**	83.6 / **82.8**	90.1	84.1
+ GKD-CLS	**93.7**	**71.3**	**83.7 / 82.8**	**90.2**	**84.3**

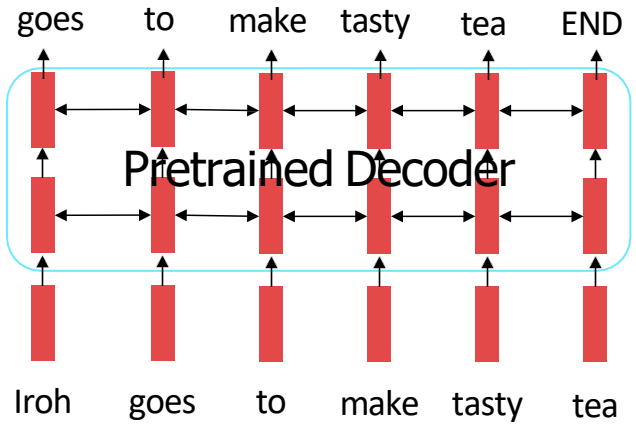
Limitations of pretrained encoders

Pretrained Encoders are great at understanding the language. However they are not capable of ”generating” content

For generating abilities, we need decoder style language modeling with next word prediction

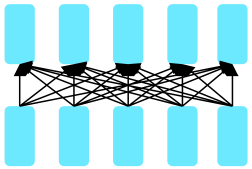
Does generation forces creativity, reasoning?





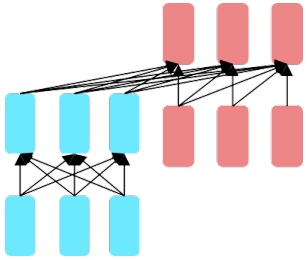
Pretraining can be done for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



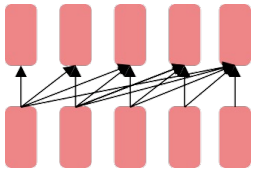
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

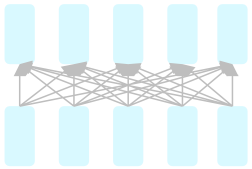


Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

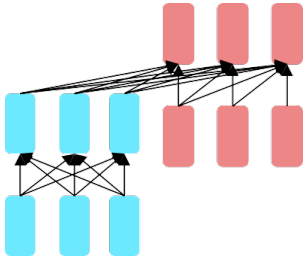
Pretraining can be done for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



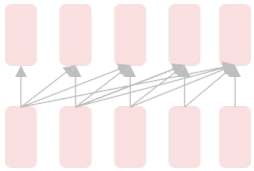
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



Decoders

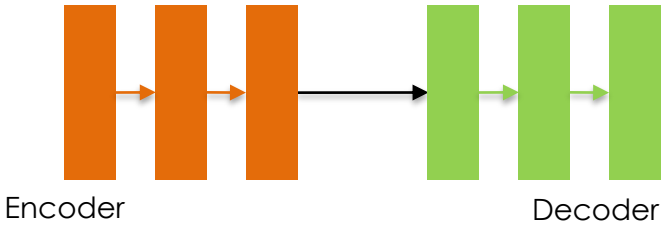
- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Pretraining of Encoder-Decoder Architectures

For **encoder-decoders**, we could do something like **language modeling**, but where a prefix of every input is provided to the encoder and is not predicted.

$h_1, \ldots, h_T = \text{Encoder}\ (w_1, \ldots, w_T)$

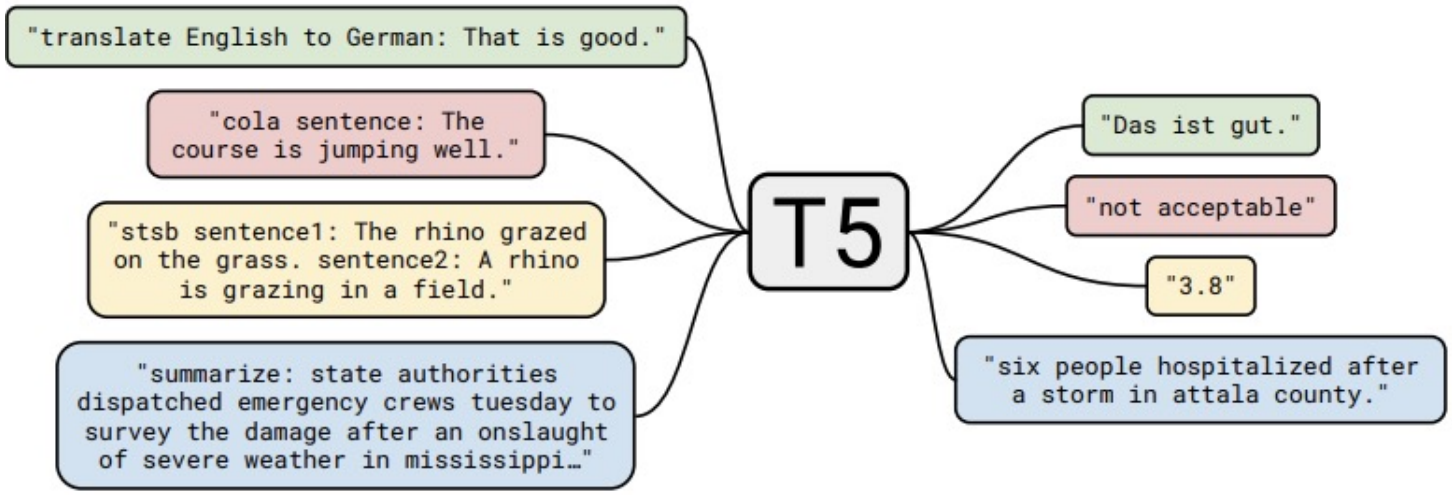
$h_{T+1}, \ldots, h_2 = Decoder\ (w_1, \ldots, w_T, h_1, \ldots, h_T)$



The **encoder** portion benefits from bidirectional context;

the **decoder** portion is used to train the whole model through language modeling.

Encoder-Decoder Model – T5

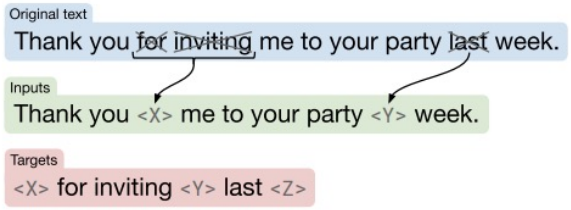


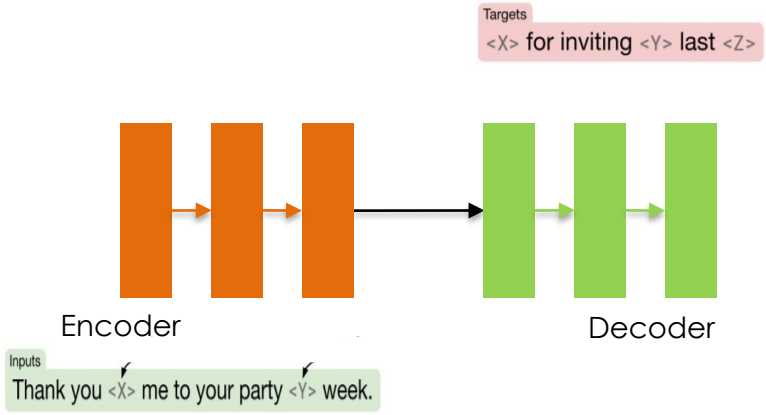
can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.

Training T5

span corruption

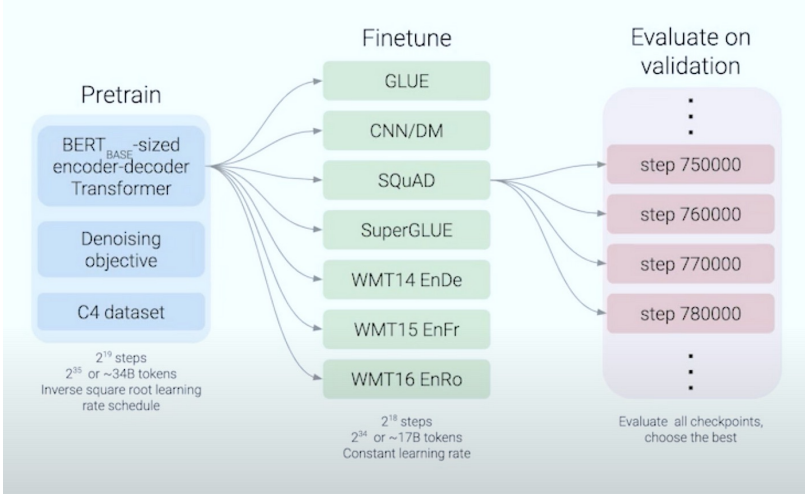
Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!
Words are dropped out independently uniformly at random. The model is trained to predict sentinel tokens to delineate the dropped out text.





Encoder-Decoder Model – T5

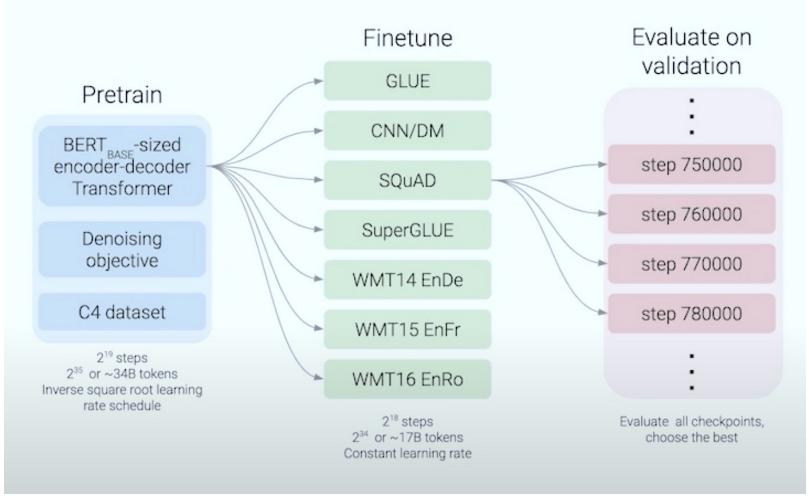
can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.



Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Encoder-Decoder Model – T5

can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.



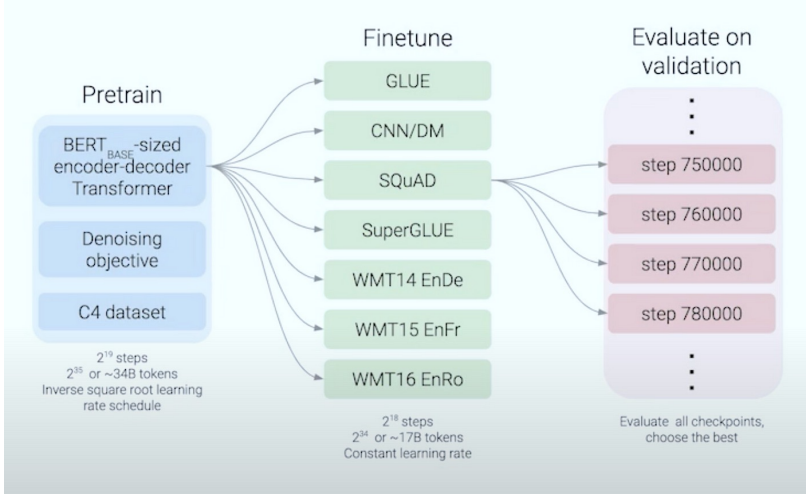
	NQ	WQ	TQA	
			dev	test
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–
T5.1.1-Base	25.7	28.2	24.2	30.6
T5.1.1-Large	27.3	29.5	28.5	37.2
T5.1.1-XL	29.5	32.4	36.0	45.1
T5.1.1-XXL	32.8	35.6	42.9	52.5
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**

220 million params
770 million params
3 billion params
11 billion params

NQ: Natural Questions
WQ: WebQuestions
TQA: Trivia QA

Encoder-Decoder Model – T5

can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.



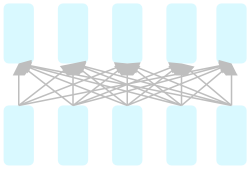
	NQ	WQ	TQA	
			dev	test
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–
T5.1.1-Base	25.7	28.2	24.2	30.6
T5.1.1-Large	27.3	29.5	28.5	37.2
T5.1.1-XL	29.5	32.4	36.0	45.1
T5.1.1-XXL	32.8	35.6	42.9	52.5
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**

220 million params
770 million params
3 billion params
11 billion params

NQ: Natural Questions
WQ: WebQuestions
TQA: Trivia QA

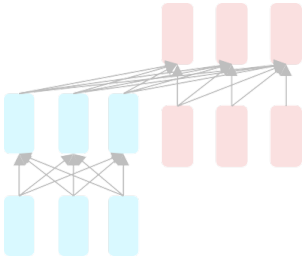
Pretraining can be done for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



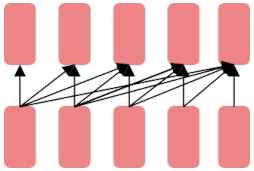
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

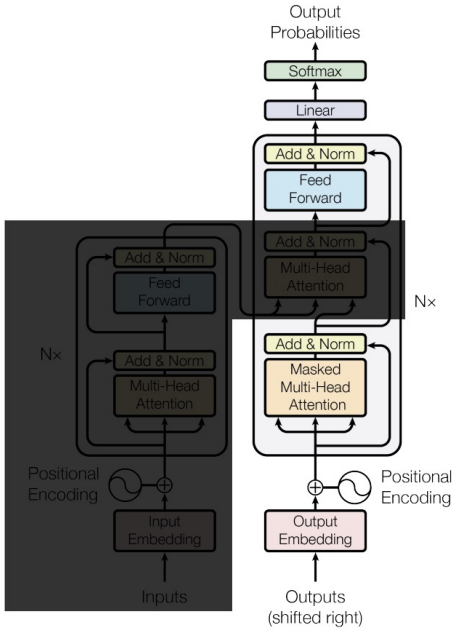
- Good parts of decoders and encoders?
- What's the best way to pretrain them?

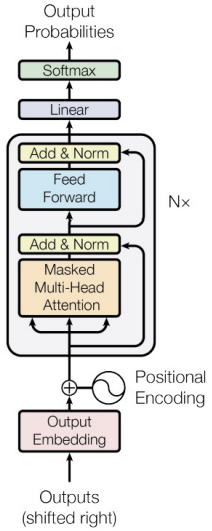


Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- All the biggest pretrained models are Decoders.

Pretraining Decoder

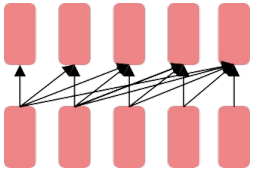




Decoders part of transformer

trained to model $p(w_t | w_{1:t-1})$

Finetuned on different tasks

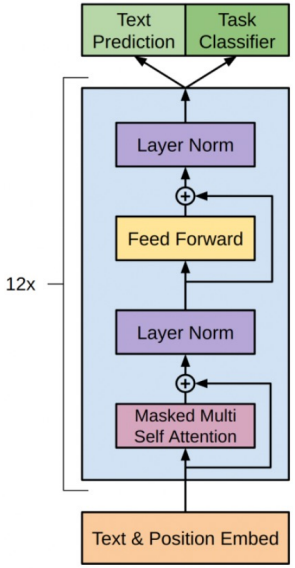


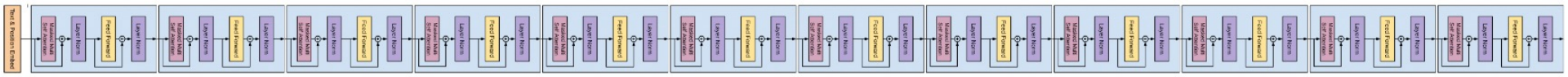
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- All the biggest pretrained models are Decoders.

Pretraining Decoder

Improving Language Understanding by Generative Pre-Training, Radford et. al. 2018 (openai team)





Pretraining Decoder

Unsupervised Pre-training

corpus of tokens w_t where t = 1:n

use a standard language modeling objective to maximize the following likelihood:

$L_1(W) = \sum_i \log P(w_i \mid w_{i-k}, ..., w_{i-1}; \theta)$

Generative Pretrained Transformer (GPT 1) Architecture [Radford et al., 2018]

2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers, 117M parameters.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.
- The acronym "GPT" never showed up in the original paper; it could stand for "Generative PreTraining" or "Generative Pretrained Transformer"

Pretraining Decoder

Supervised Fine Tuning

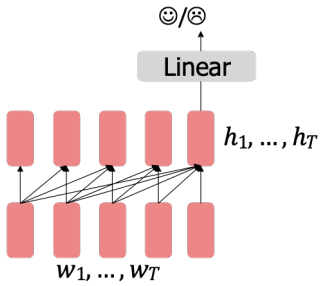
labeled dataset C, where each instance consists of a sequence of input tokens, $x_1, \ldots, x_m$, along with a label y.

The inputs are passed through pre-trained model to obtain the final transformer block's activation h_m^l, which is then fed into an added linear output layer with parameters W_y to predict y

$$P(y|x^1, \ldots, x^m) = \texttt{softmax}(h_l^m W_y).$$

objective to maximize:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m)$$



GPT for downstream Tasks

Supervised Fine Tuning

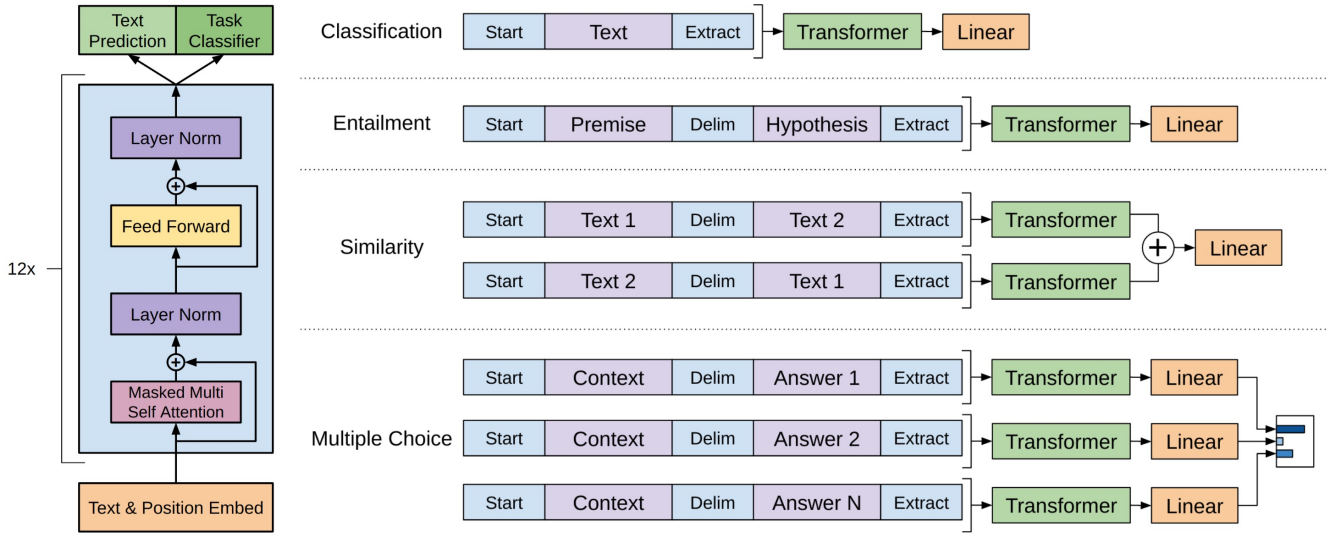


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

GPT for downstream Tasks

How to format inputs to our decoder for **finetuning tasks?**

Natural Language Inference: Label pairs of sentences as *entailing/contradictory/neutral*

Premise: *The man is in the doorway*

Hypothesis: *The person is near the door*

} **entailment**

Radford et al., 2018 evaluate on natural language inference.
Here's roughly how the input was formatted, as a sequence of tokens for the decoder.

[START] *The man is in the doorway* [DELIM] *The person is near the door* [EXTRACT] The linear

classifier is applied to the representation of the [EXTRACT] token.

GPT results on various NLI datasets.

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

Better text generation with GPT2

GPT-2, a larger version (1.5B) of GPT trained on more data, was shown to produce relatively convincing samples of natural language.

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

GPT Models in the last 6 years

Thank You

Next Week – GPT LLMs