- which are then merged together so as to produce a compact representation

$$z_i = w_i * v_i,$$

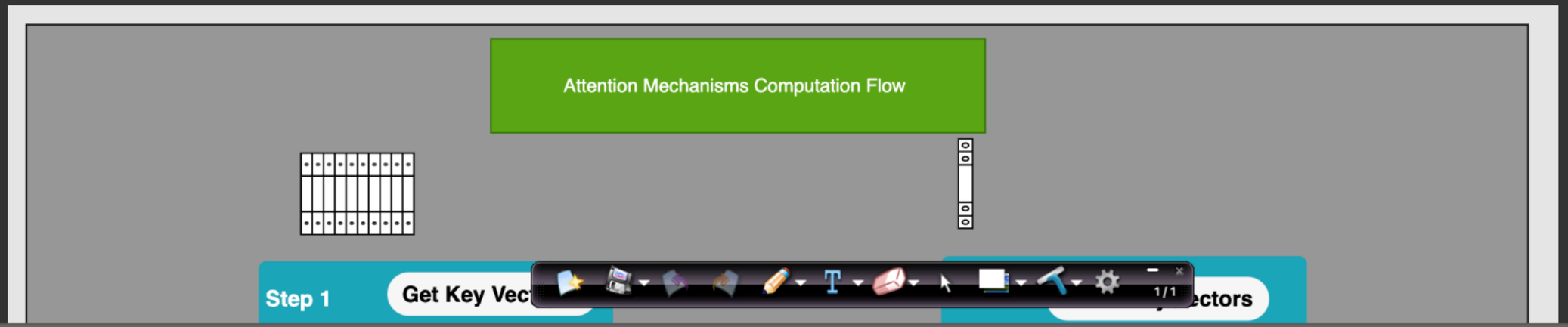
$$C = \sum_{i=1}^{d_k}(z_i),$$

Attention: $\text{softmax}\left(\frac{K^T Q}{\sqrt{d}}\right) \cdot V$

How to compute attention?

Ex - we have source sentence of 6 words and a target word , and we have to calculate attention score and context vector for target word.

- Assume source sentence `food is delicious but price is high`
- target words `tasty`
- word `3` is similar in context to `target words`.



Agenda:

ABSA

1. Attention - recap → $\text{Softmax}\left(\frac{K^T \cdot Q}{\sqrt{d_K}}\right) \cdot V$
2. Self attention ; Multi-head attention
3. Bi-dir LSTM + Attention ✓
   - architecture
   - code (ABSA) ✓

4. Language Translation → classical methods
5. Transformers → encoder, decoder
   - architecture

BERT

Attention_Function	Equation $f(Q, K)$
Location_Basedt	$f(Q)$
	$f(Q, K) = [e_0; e_1; ; ; ; ; e_{d_k}]$
Convolution_Based	$e_j = \frac{1}{l} \sum_{i=j-l}^{k} e_{i,j}$
	$activation(W \cdot [k_i; k_{i+1}; ; ; k_{i+l}] + b)$

$$Z = Softmax\left(\frac{K^T \cdot Q}{\sqrt{d_k}}\right) \cdot V$$

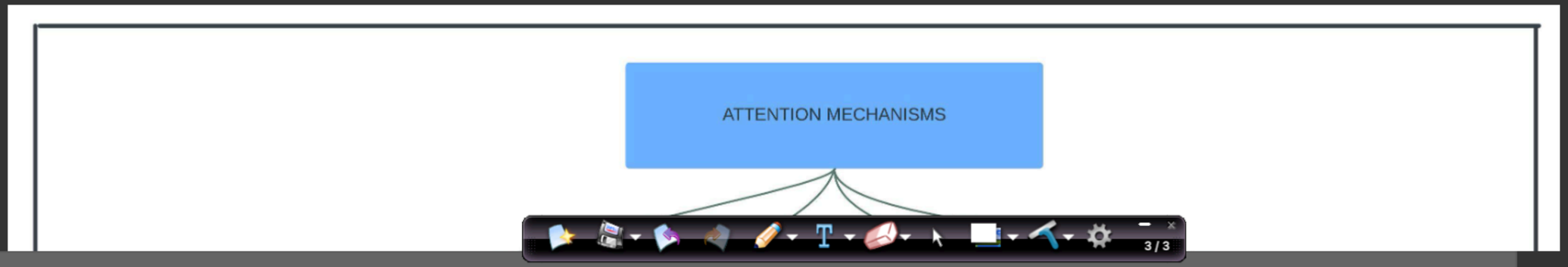
sent aspects =K

$d_k = 5$ dim of each word vector

What are different type of Attention Mechanisms?

Attention Mechanisms can be broadly categorized into four category:

1. Number of Sequences.
2. Number of Abstraction.
3. Number of Position.
4. Number of Representation.



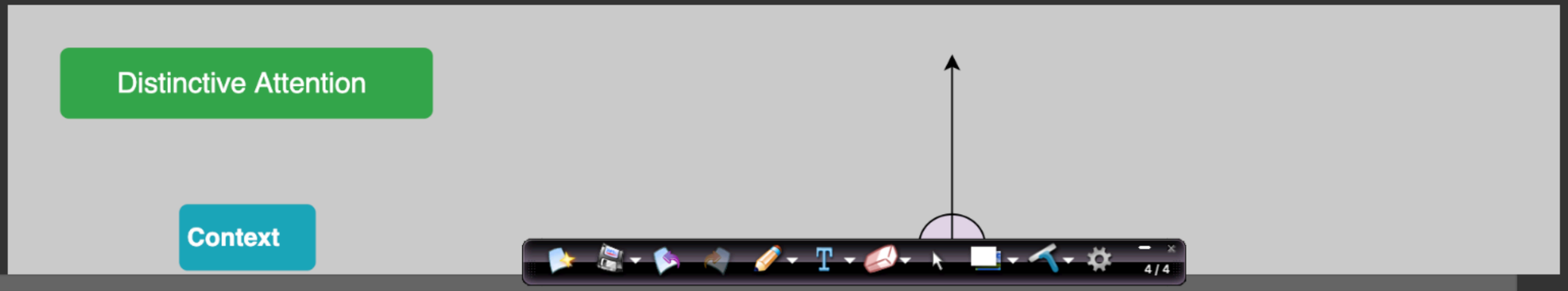
What are attention mechanisms based on Number of Sequences?

1. Distinctive Attention Mechanisms

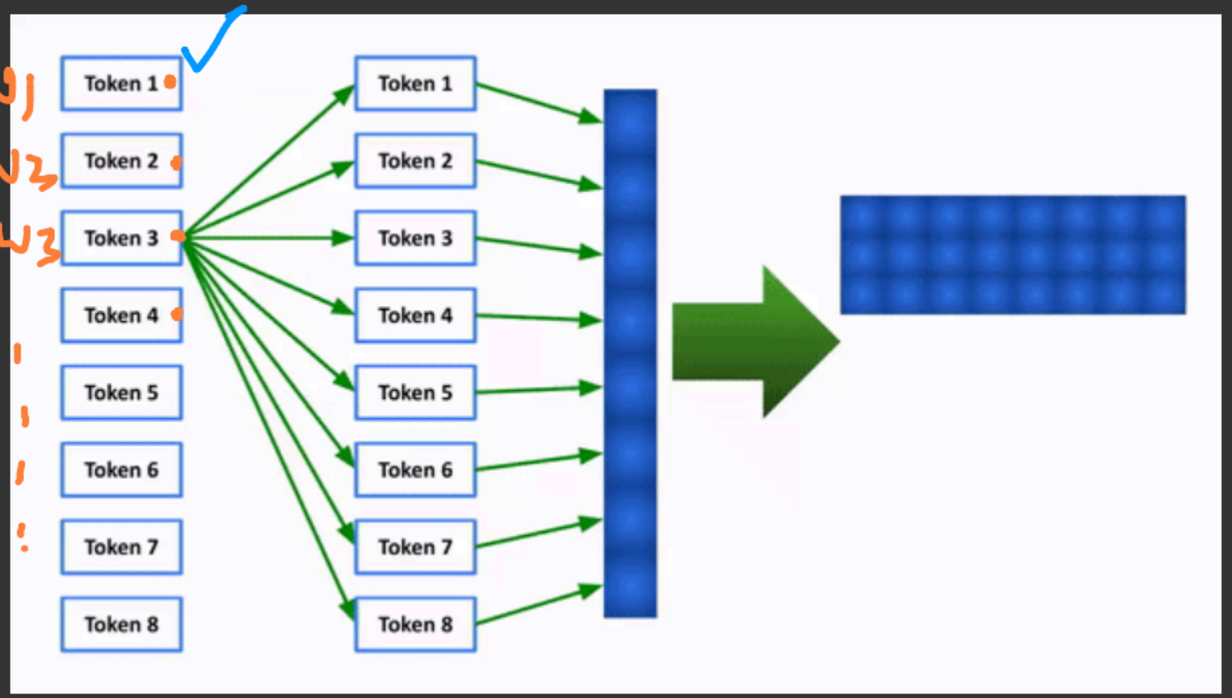
- In Distinctive Attention Mechanisms, Both key and Query text Sequence is different.
- KEY VECTOR == VALUE VECTOR!= QUERY VECTOR
- **How to Compute Distinctive AM?**
  - Compute KEY, QUERY and VALUE vectors
  - Compute Energy function
  - Compute Attention Score

(ABSA)

Fig Below - Distinctive Attention Mechanisms.



- Compute KEY, QUERY and VALUE Vectors
- Compute Energy function
- Compute Attention Score



$$Z = \text{Softmax}\left(\frac{K^T \cdot Q}{\sqrt{d_k}}\right) \cdot V$$

$K = Q = V \Leftrightarrow$ self attention

Self Attention

KEY = [h1, h2, h3...h]
QUERY=VALUE = KEY

What are attention mechanisms based on Number of Position?

1. Soft Attention Mechanisms

- In soft attention, we compute attention weight $w_1, w_2, w_3 \ldots w_n$, $where \sum W = 1.$
- And, It uses a weighted average of all hidden states of the input sequence to build the context vector.

$$z_i = w_i * v_i,$$

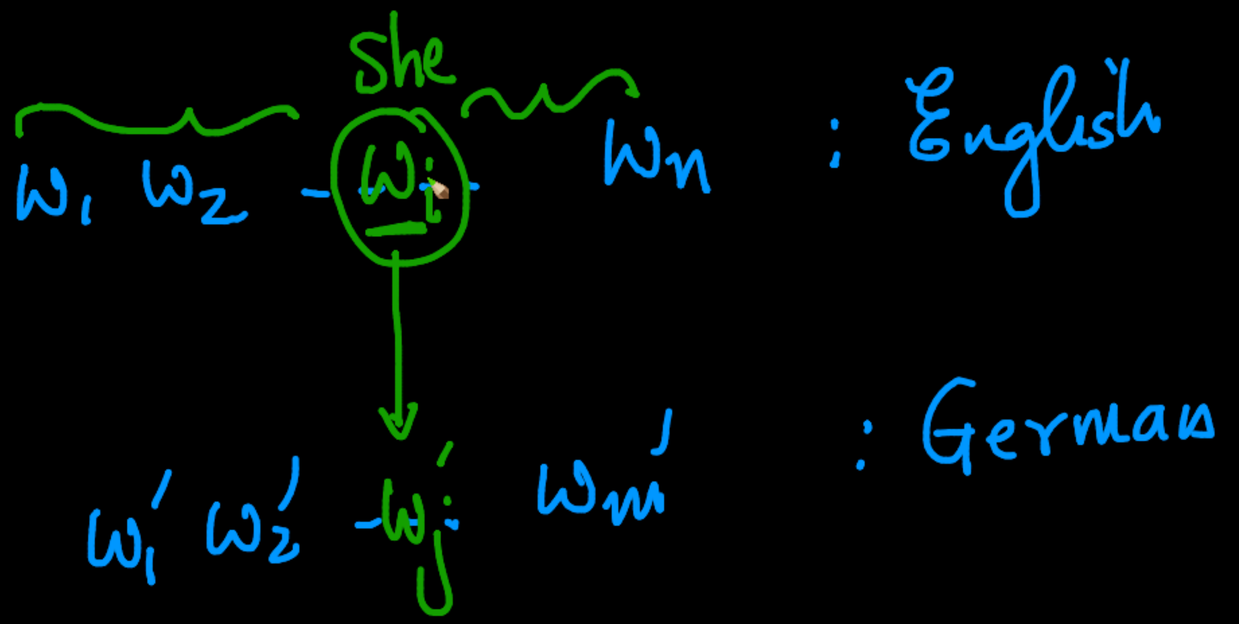
$$C = \sum_{i=1}^{d_k}(z_i),$$

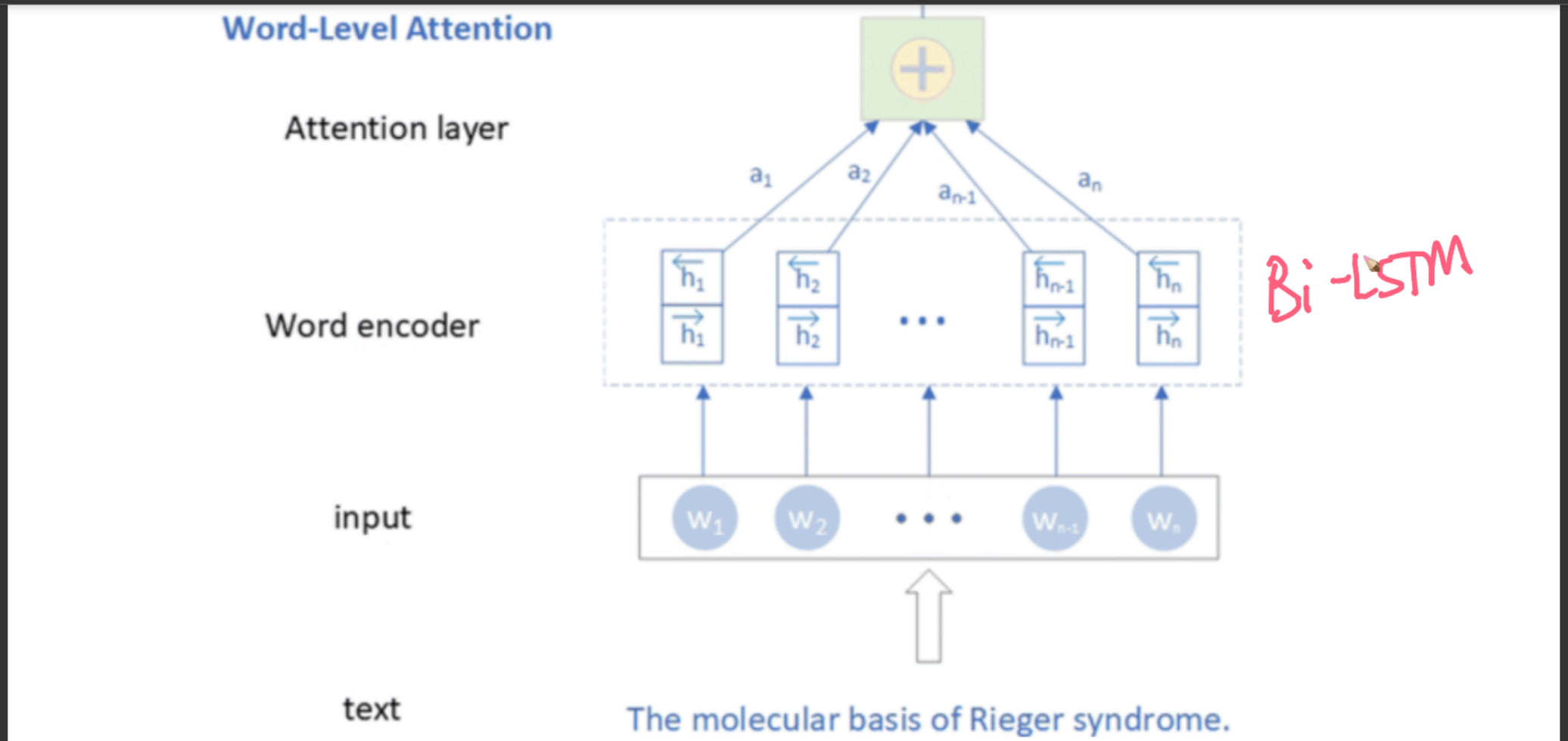
- The usage of the soft weighing method makes the neural network amenable to efficient learning through backpropagation,
- **Pro**: the model is smooth and differentiable.
- **Con**: expensive when the source input is large.

$$Z = \text{Softmax}\left(\frac{K^T \cdot Q}{\sqrt{d_K}}\right) \cdot V$$

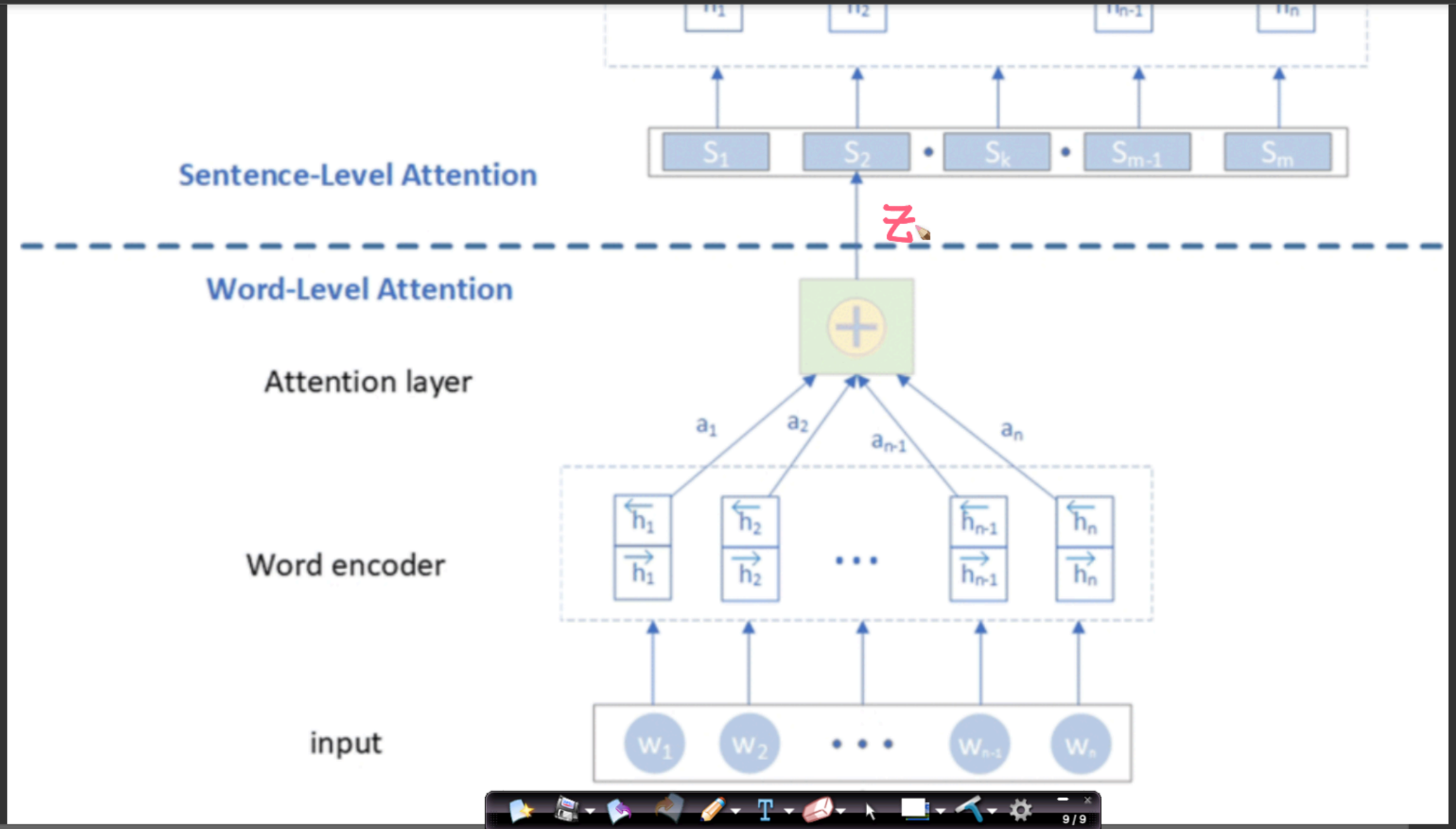
$w_1, w_2, w_3 \ldots w_n$

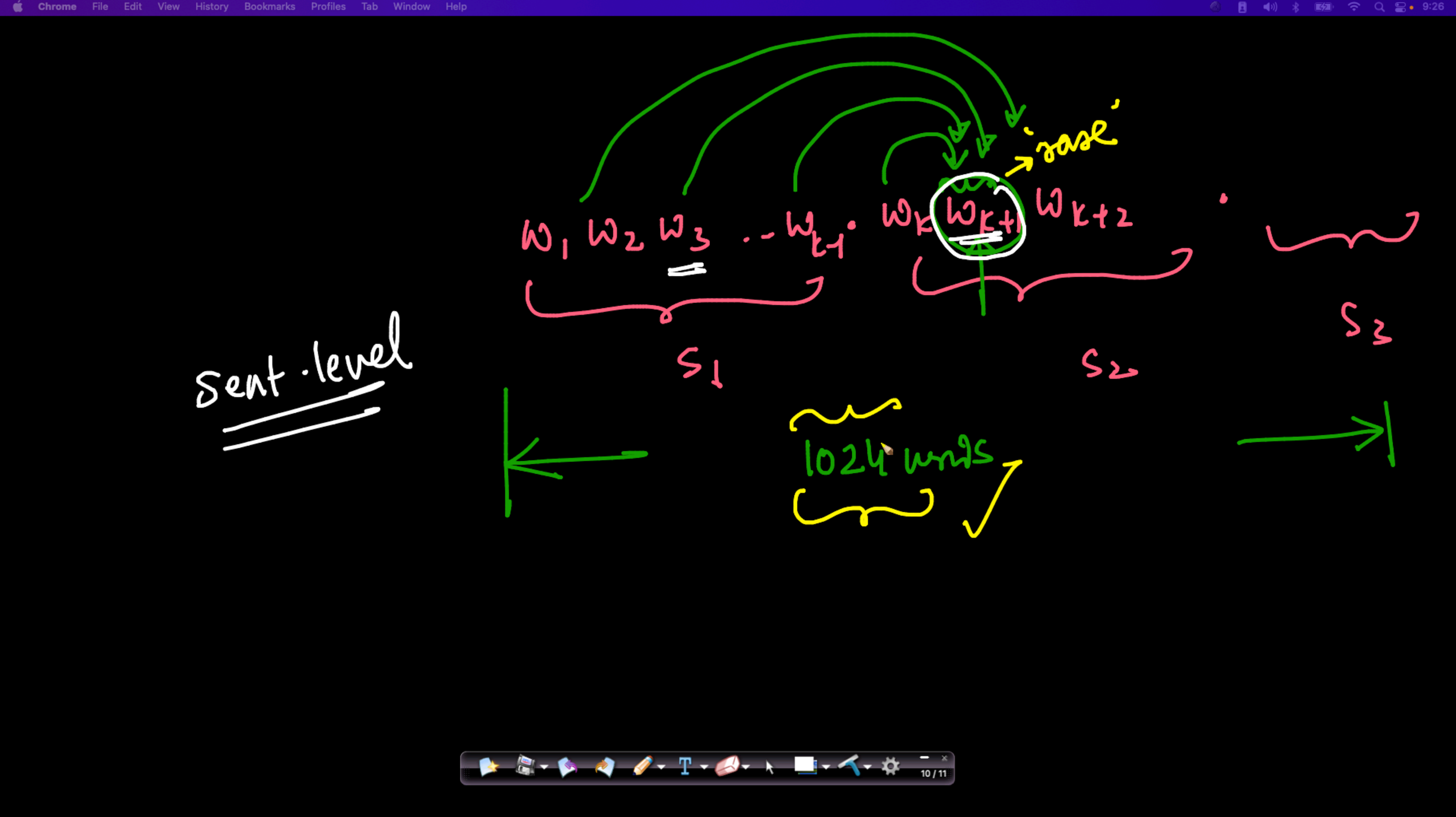
Self-attention :

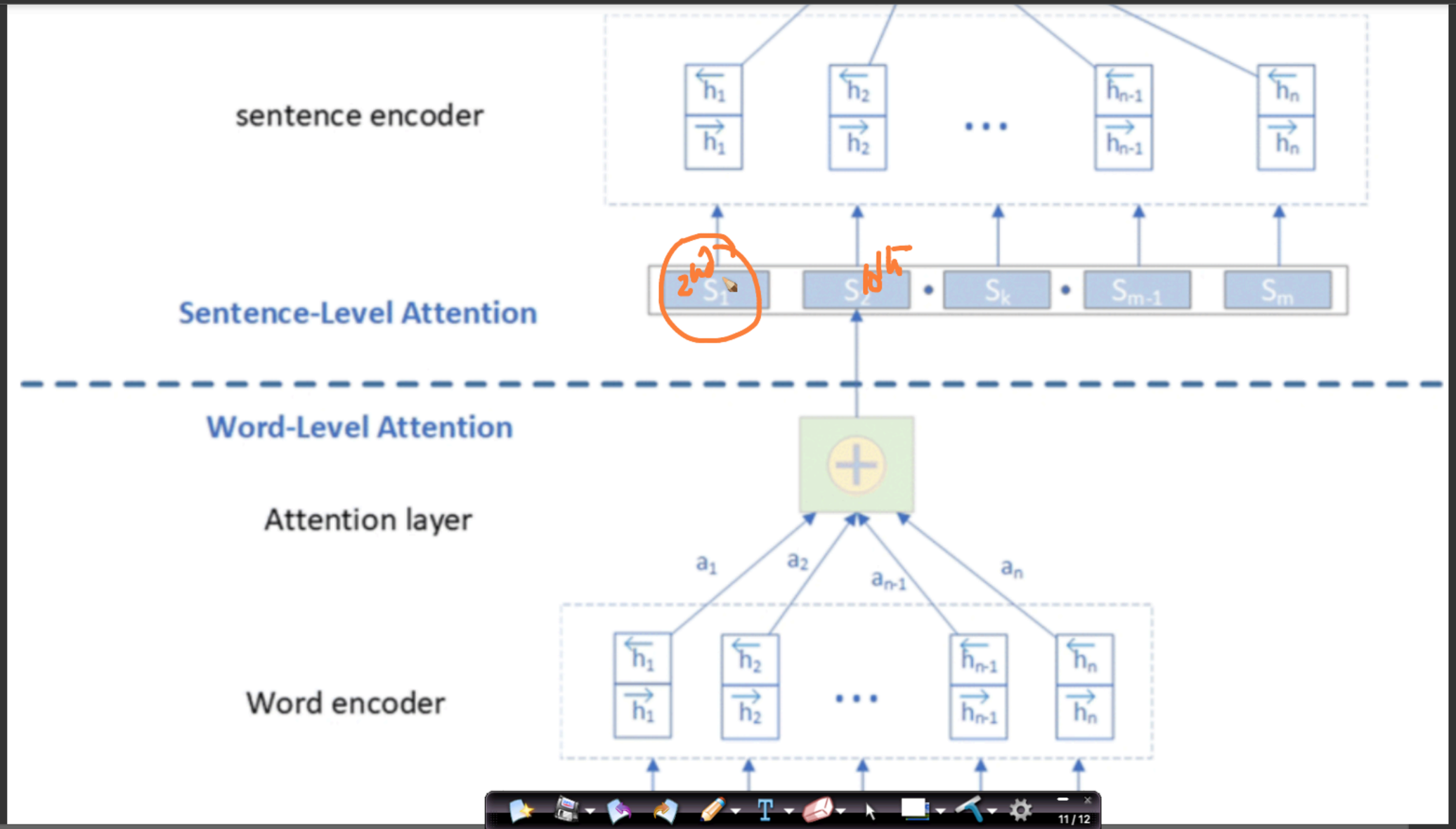




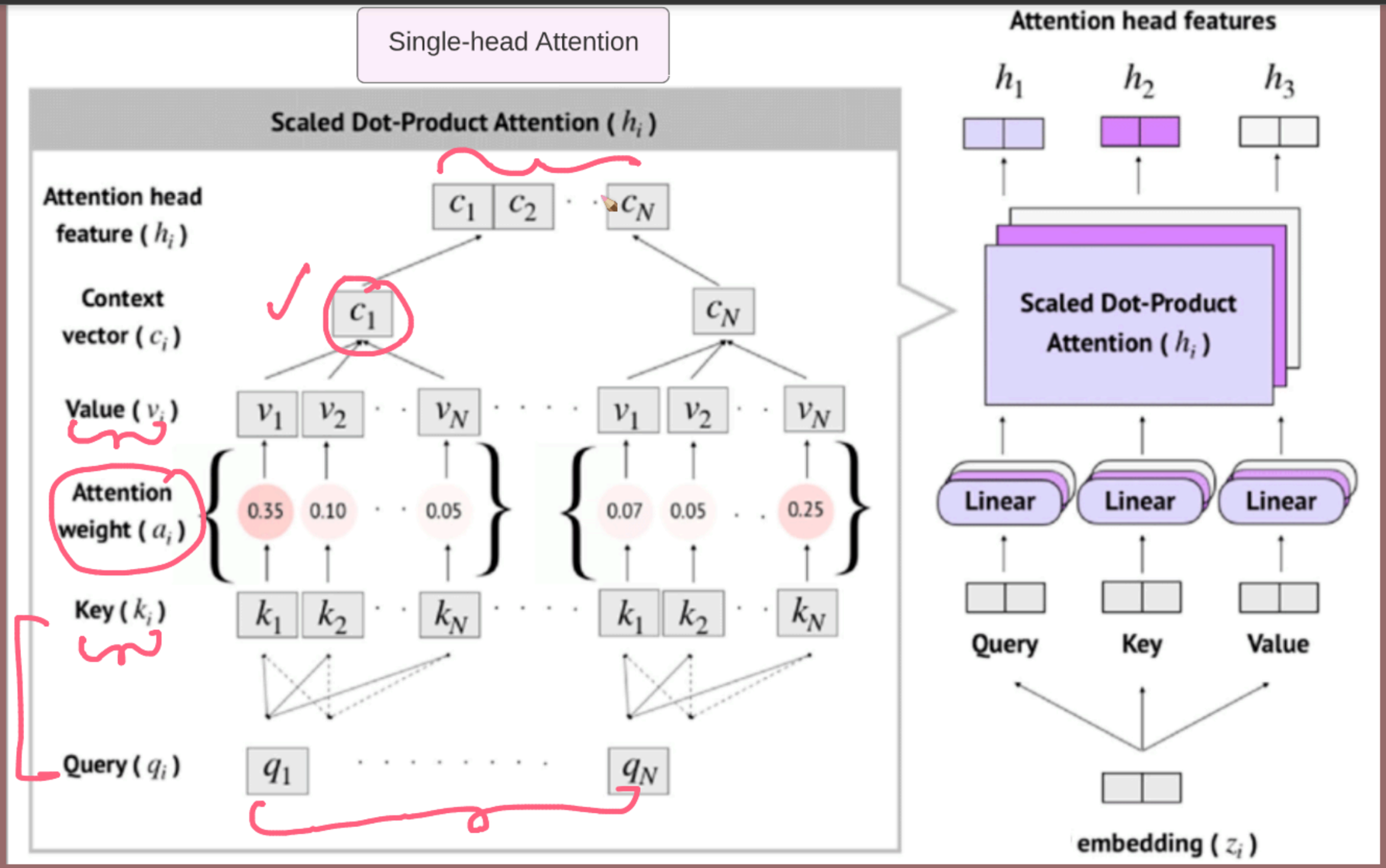
The hierarchical attention mechanism architecture. We connect the word-level and sentence-level features by attention mechanism to get a feature vector with more information and the Softmax function is used to normalize the result

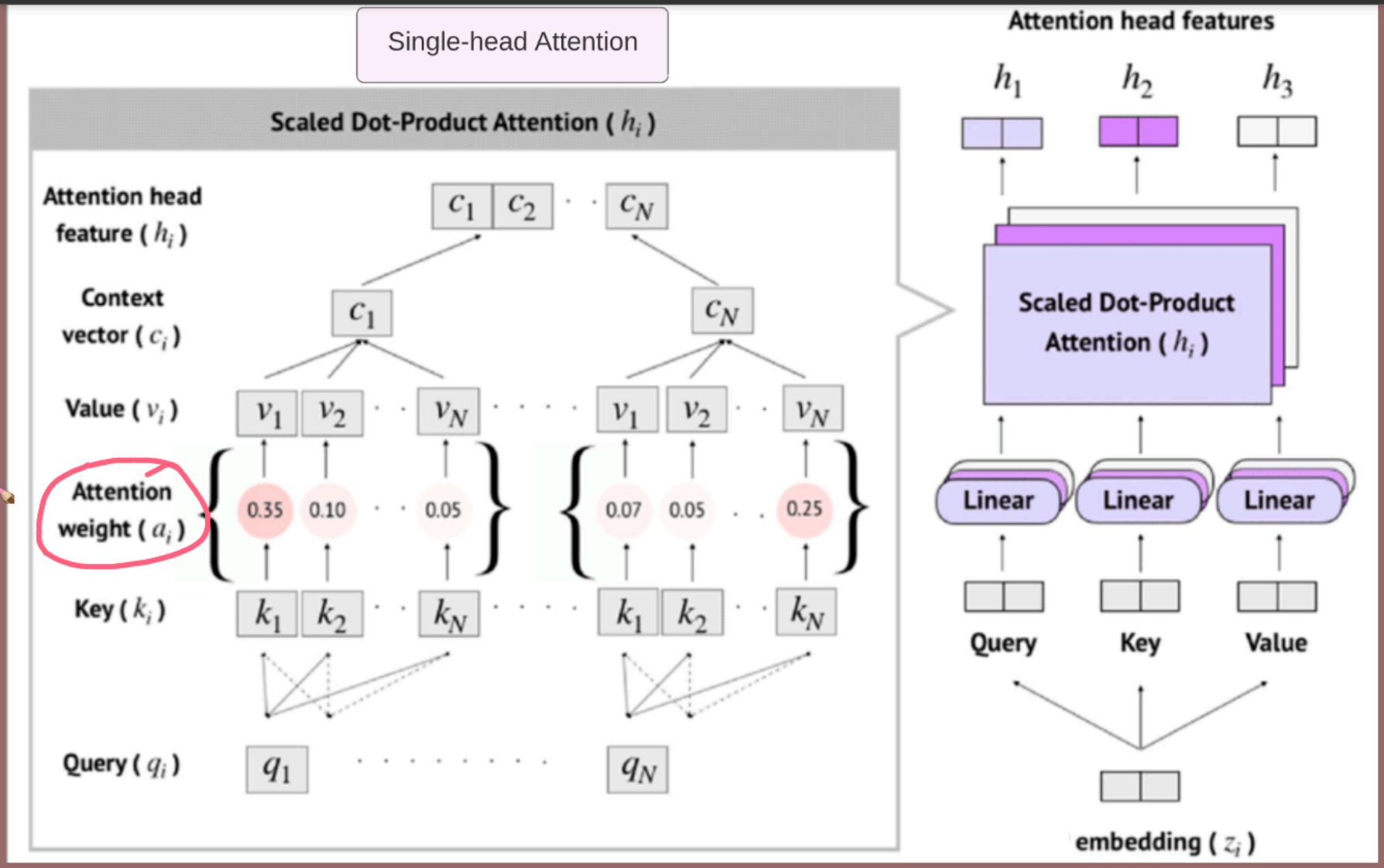

Sentence-Level Attention
Word-Level Attention
Attention layer
Word encoder
input
S_1 S_2 S_k S_{m-1} S_m
a_1 a_2 a_{n-1} a_n
h_1 h_2 h_{n-1} h_n
w_1 w_2 w_{n-1} w_n

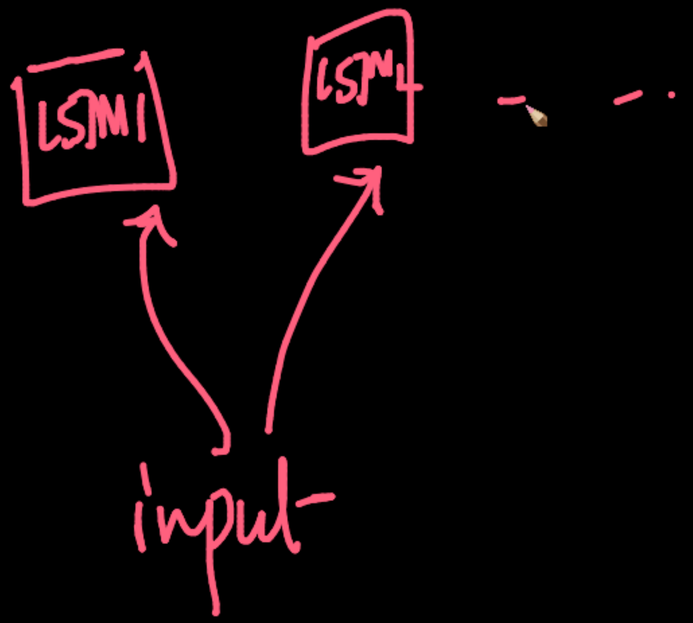

Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:26
'base'
w_1 w_2 w_3 .. w_{k-1} w_k w_{k+1} w_{k+2}
s_1
s_2
s_3
Sent. level
1024 units
10 / 11



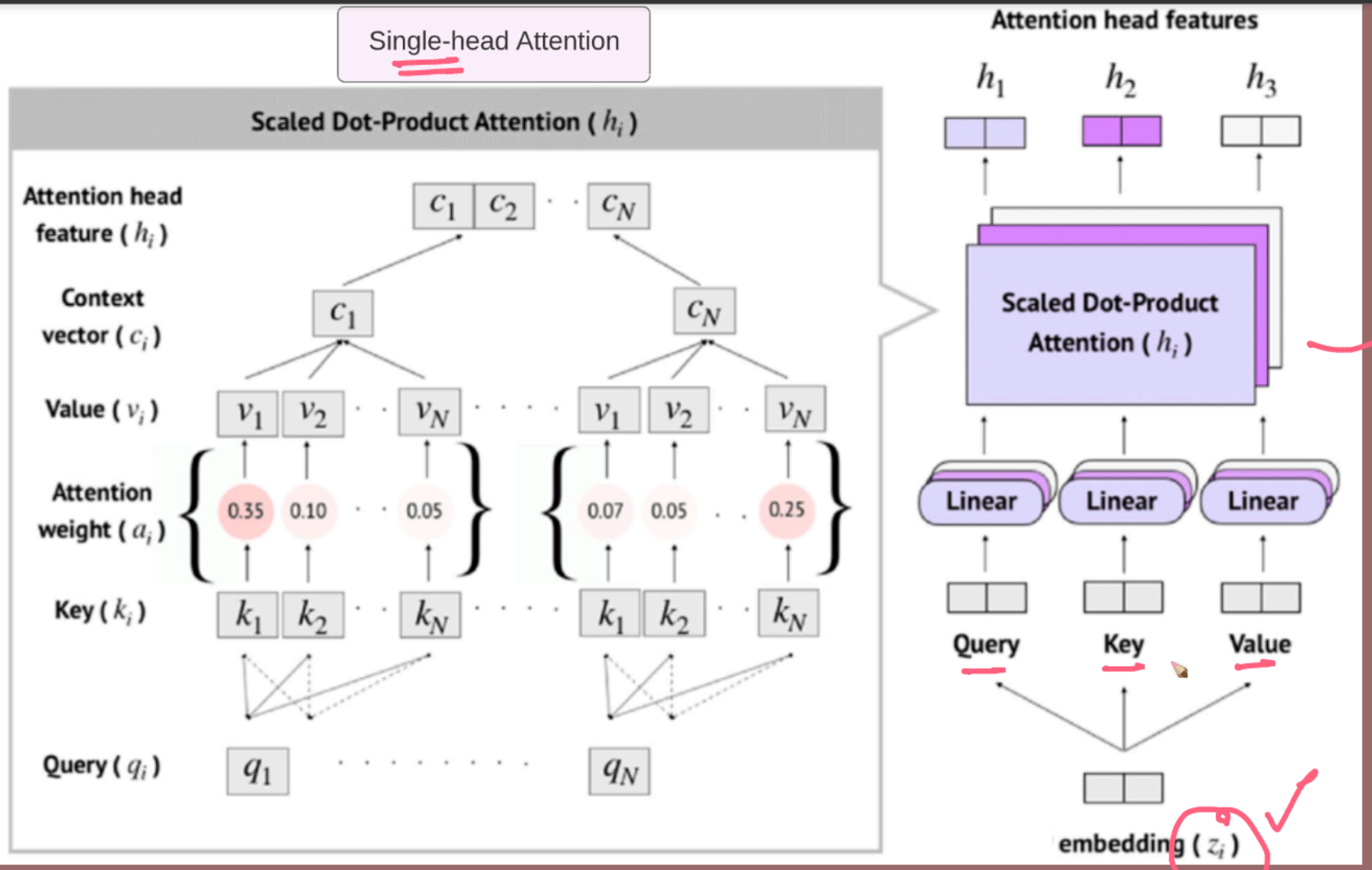
Single-head Attention





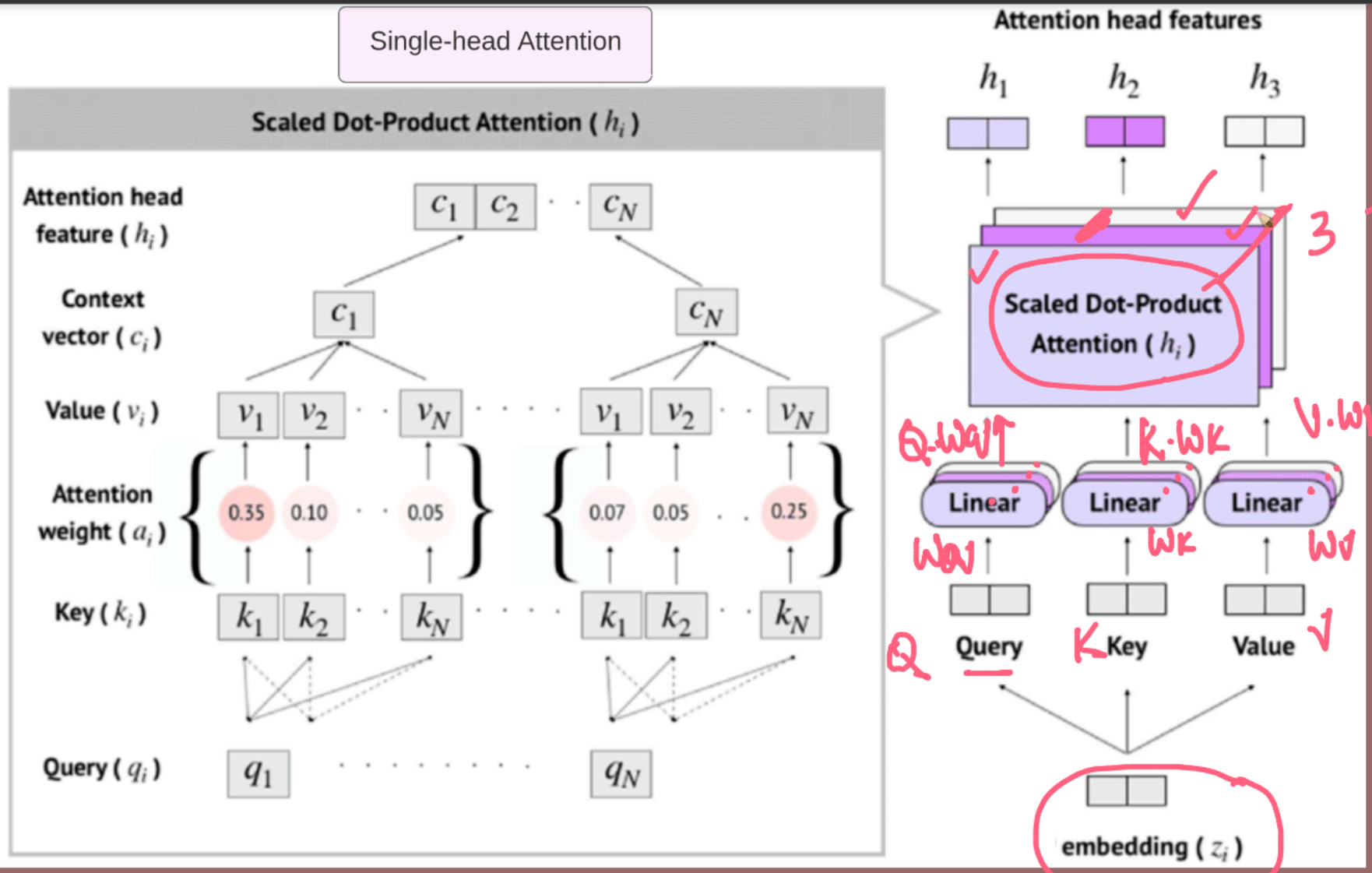


Single-head Attention

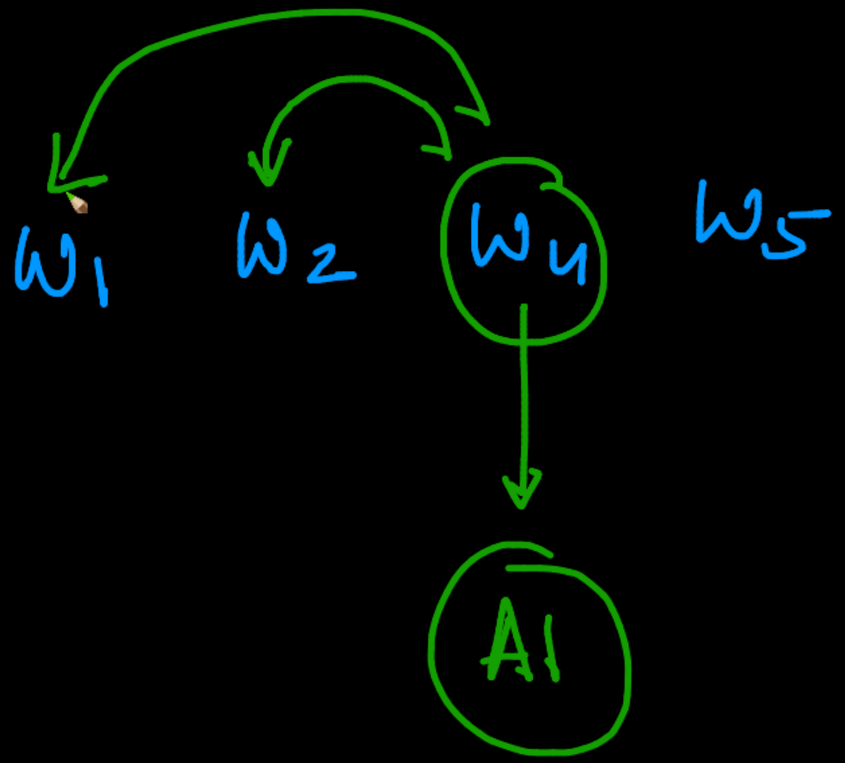


self attn

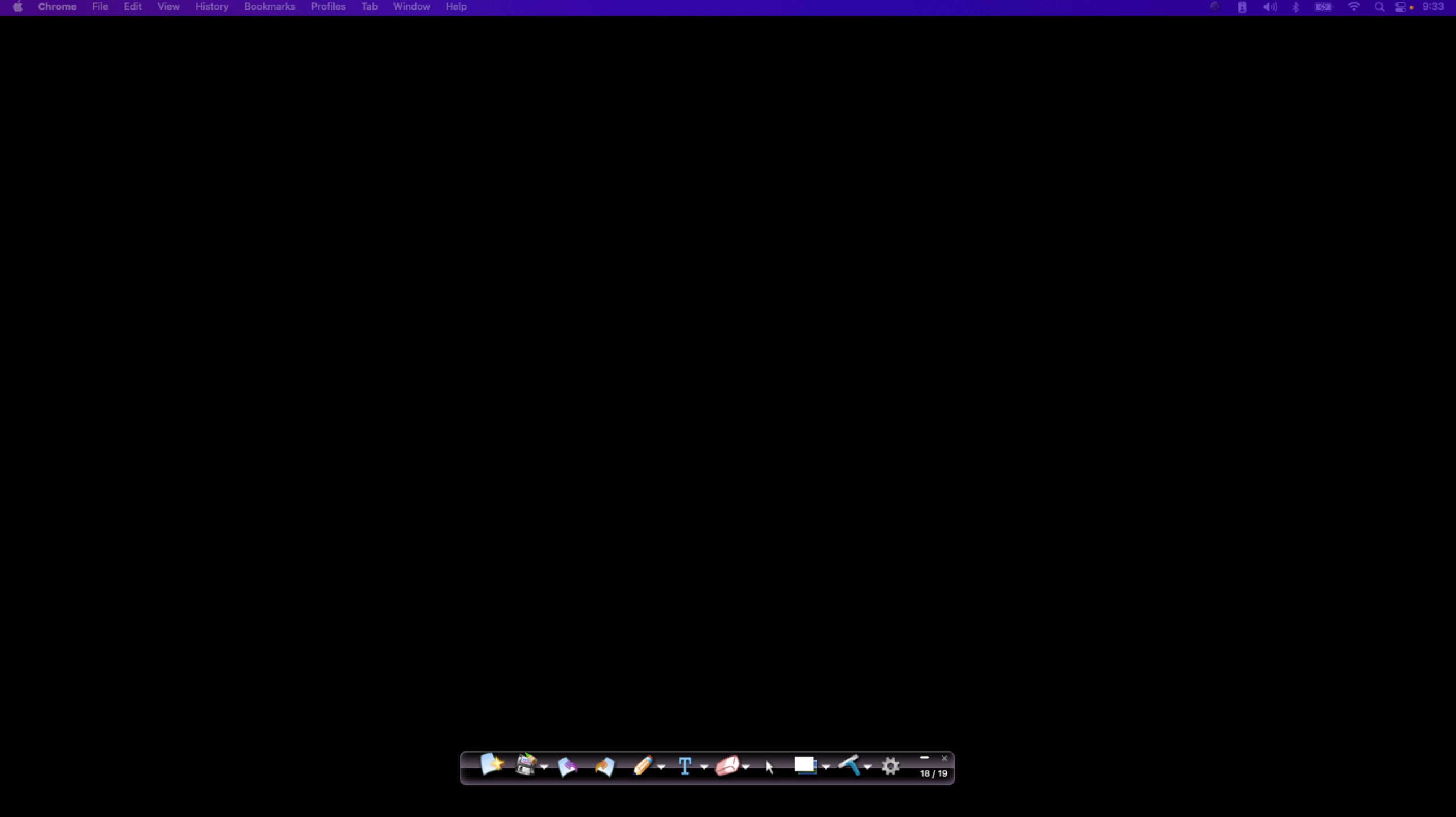
Single-head Attention

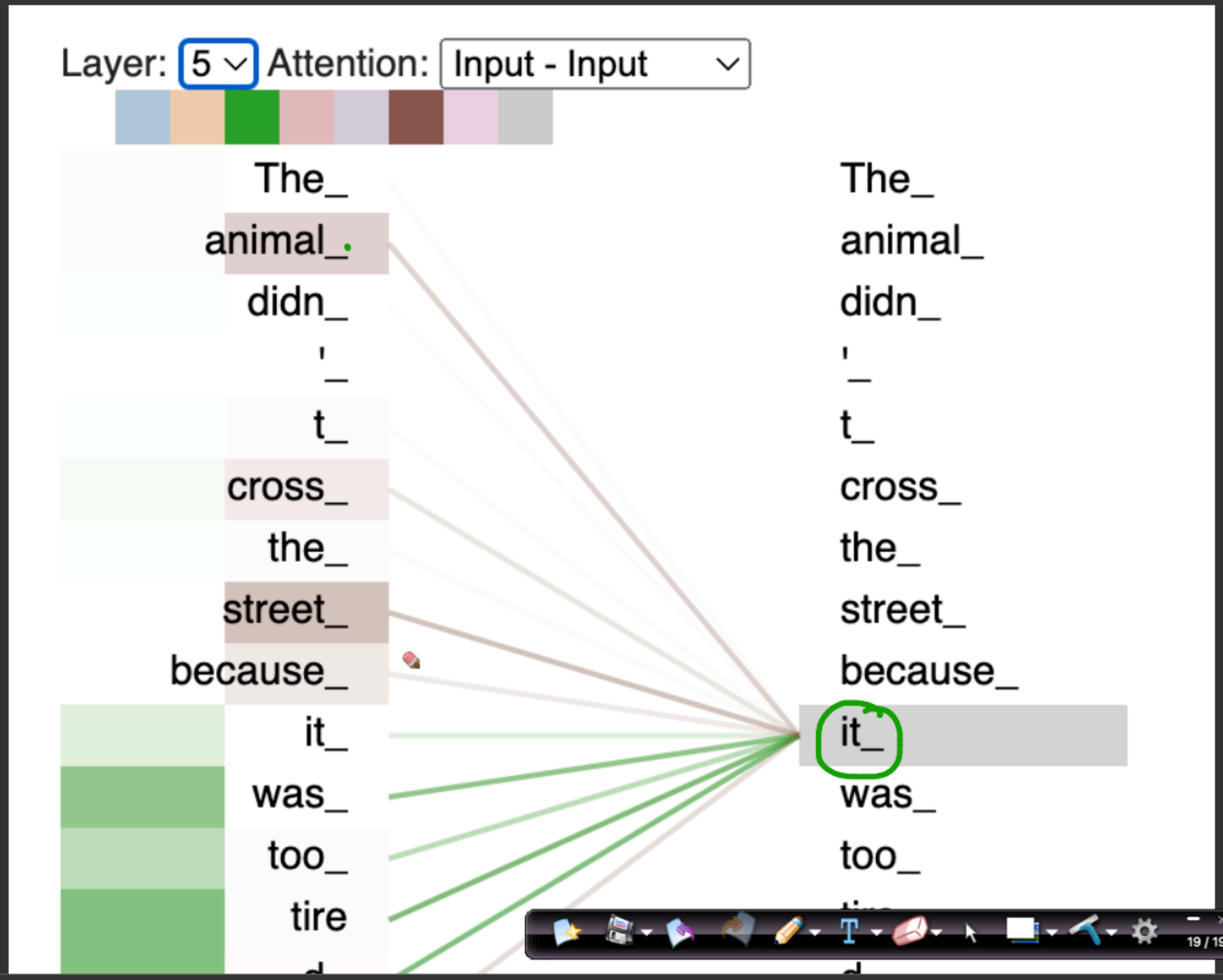


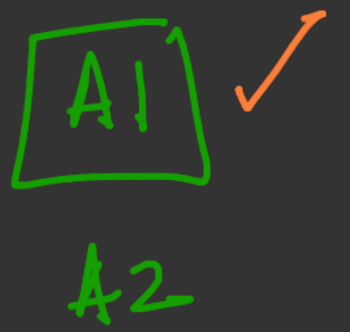
e.g.:

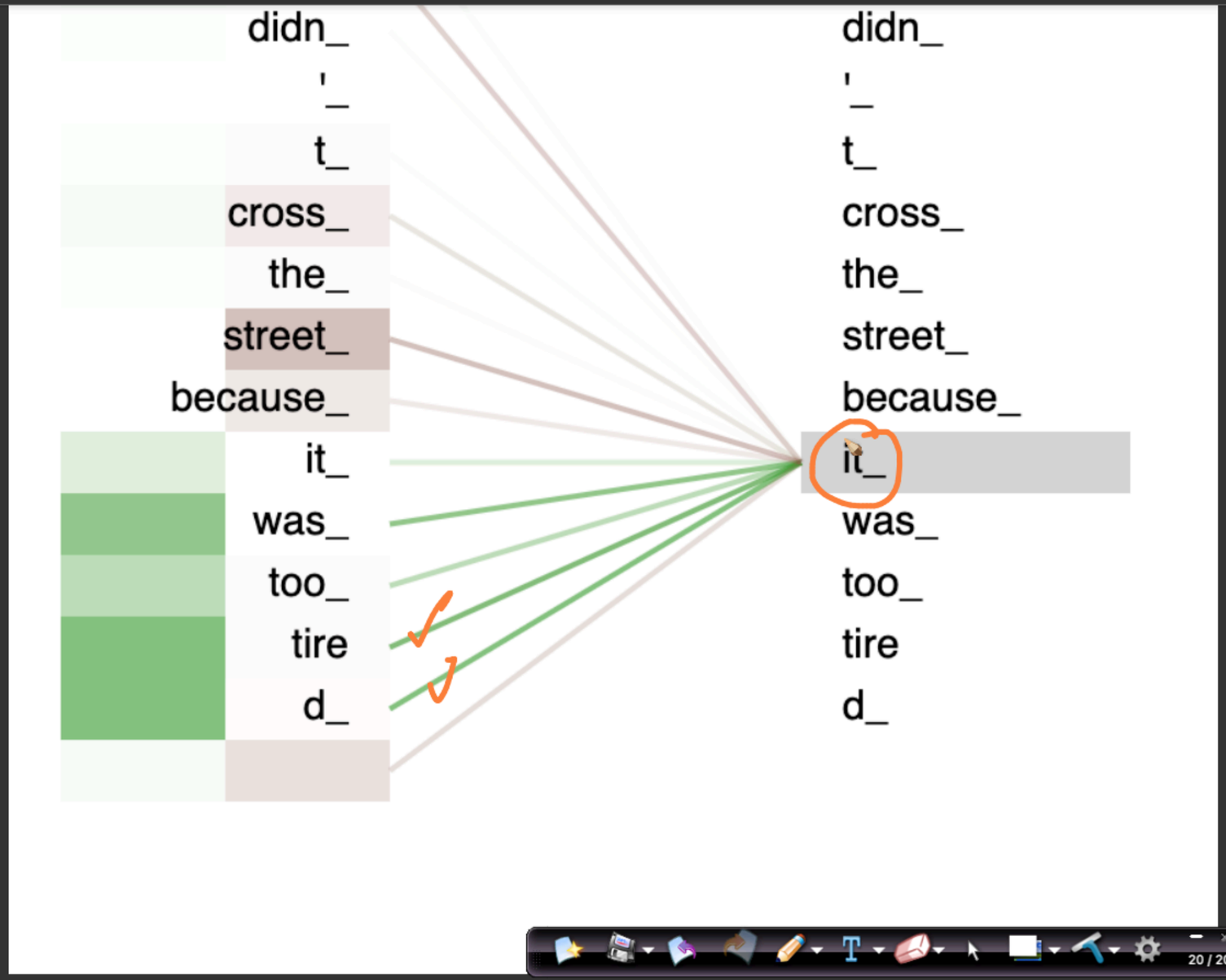


A2


Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:33
18 / 19






didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

```
Y = [glove_vectors[i].tolist() for i in target]
Y = np.stack([pca.transform(i) for i in Y]).reshape(11, 5)
```

Lets Compute one of the popular Multi-headed attention in NLP.

- `Multi-headed Self Attention`

1. Step-1 Randomly intialized wq, wk and wv MATRIX.

- Each word in X is represented using a vector of size 5.
- Randomly Initialize 3 matrices, `Wq, Wv, Wk = [5 × 9]` matrix

```
wk = np.ones((5, 9))
wv = np.ones((5, 9))
wq = np.ones( (5, 9))
display(HTML('<h3>WQ</h3>'))
np_display(wq)
```

WQ

	0	1	2	3	4	5	6	7	8
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Bi-dir LSTM ✓

BiDir LSTM + attention

Attention - Enc-Dec

↳ Transformers

Attention - Enc

↳ BERT

```
Y = [glove_vectors[i].tolist() for i in target]
Y = np.stack([pca.transform(i) for i in Y]).reshape(11, 5)
```

Lets Compute one of the popular Multi-headed attention in NLP.

- `Multi-headed Self Attention`

1. Step-1 Randomly intialized wq, wk and wv MATRIX.

- Each word in X is represented using a vector of size 5.
- Randomly Initialize 3 matrices, `Wq, Wv, Wk = [5 × 9]` matrix

```
wk = np.ones((5, 9))
wv = np.ones((5, 9))
wq = np.ones( (5, 9))
display(HTML('<h3>WQ</h3>'))
np_display(wq)
```

WQ

	0	1	2	3	4	5	6	7	8
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Wq, Wv, Wk

3 heads

2. Define number of Attention Head

- For our task assumes N=3

```
[ ]  N_head = 3
```

3. Compute Query, Value and Key Matrix

- Three new matrix can be generated by dot product on each weight matrices with seq_embedding,

$$Query\ Matrix = Q = X \bullet W_q$$
$$Key\ Matrix = K = X \bullet W_k$$
$$Value\ Matrix = V = X \bullet W_v$$

1) This is our input sentence*

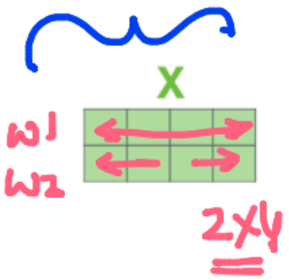
2) We embed each word*

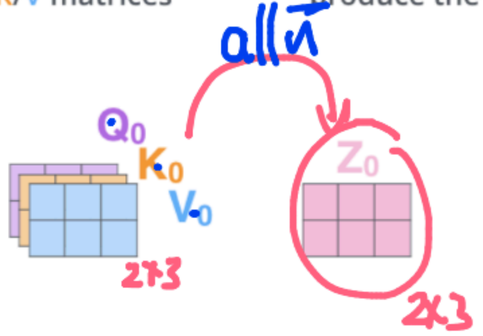
3) Split into 8 heads. We multiply X or R with weight matrices

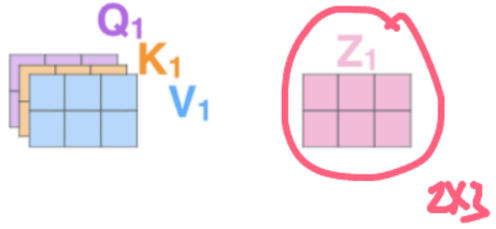
4) Calculate attention using the resulting Q/K/V matrices

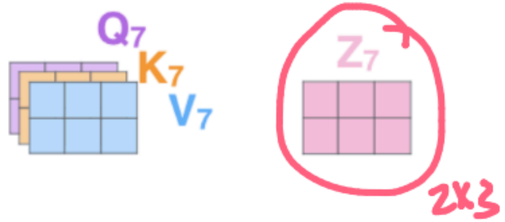
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

Thinking Machines

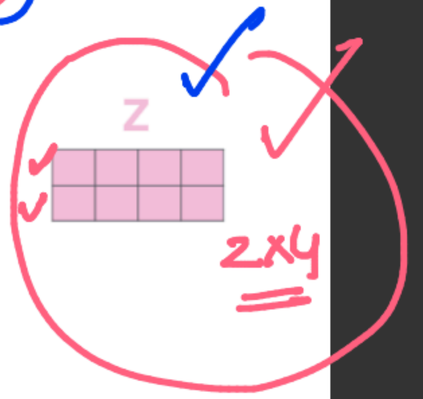




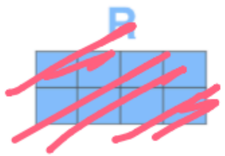




$$Z_0 = \text{softmax}\left(\frac{K_0^T \cdot Q_0}{\sqrt{d_k}}\right) \cdot V$$



* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



1) This is our input sentence*

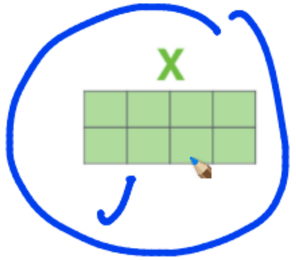
2) We embed each word*

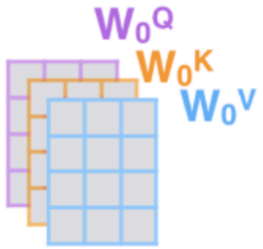
3) Split into 8 heads. We multiply X or R with weight matrices

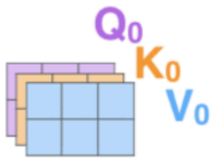
4) Calculate attention using the resulting Q/K/V matrices

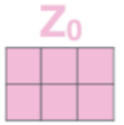
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

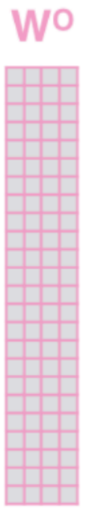
Thinking Machines

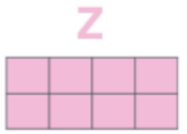


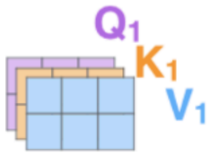


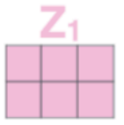




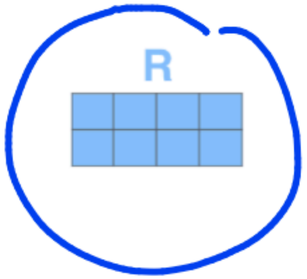




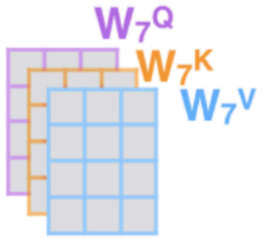


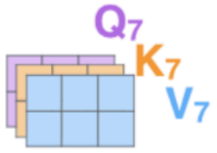


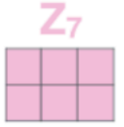
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

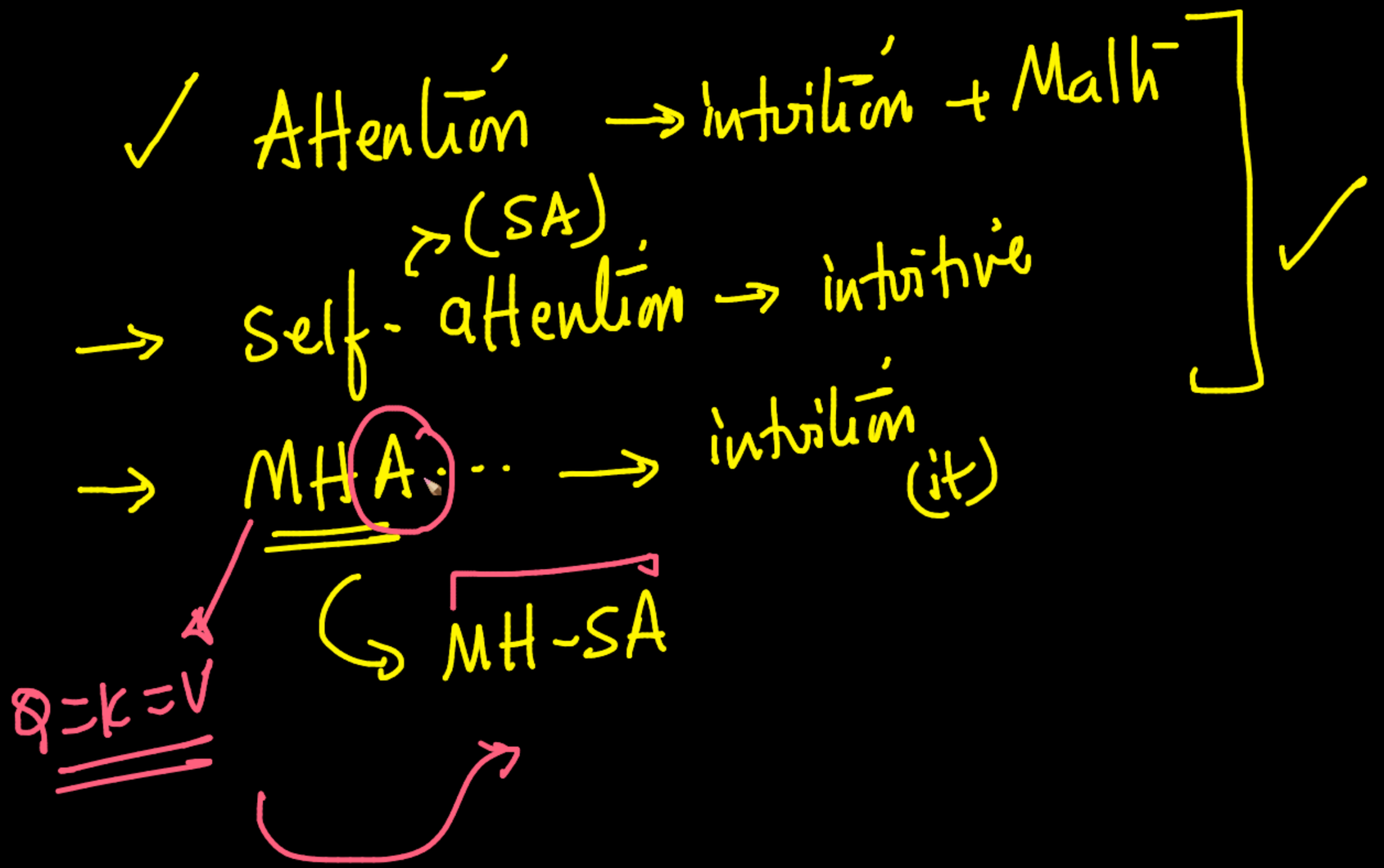


...









+ Code + Text Connect

```
print(f"{len(val_pairs)} validation pairs")
```

```
****************************** Train and Test Split ******************************
2202 total pairs
1872 training pairs
330 validation pairs
```

```
[ ] print('*'*30,'train', '*'*30)
    train_pairs.head()
```

```
****************************** train ******************************
```

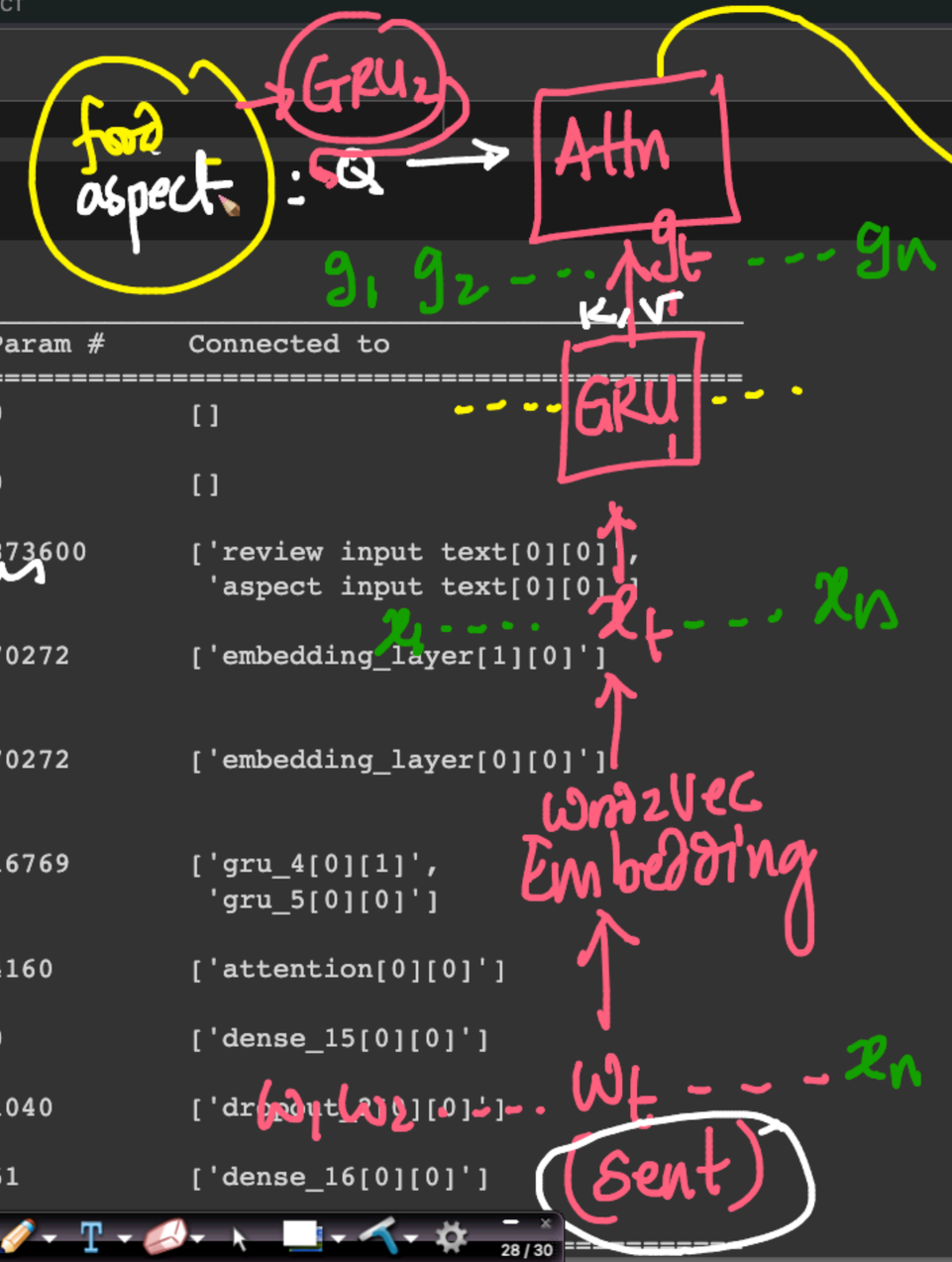
	sentence	term	polarity
432	your a sushi fan, you love expertly cut fish, ...	eel	2
291	the coffe is very good, too.	coffe	2
1197	during the course of the past 3 months, the ch...	staff	0
2390	the bread was stale, the salad was overpriced ...	salad	0
212	best restaurant in the world, great decor, gre...	decor	2

```
[ ] print('*'*30,'Train Sentiment label Distribution', '*'*30)
    train_pairs['polarity'].value_counts()
```

```
****************************** Train Sentiment label Distribution ******************************
2    1345
0     453
1      74
```

```
[ ] model.summary()
```

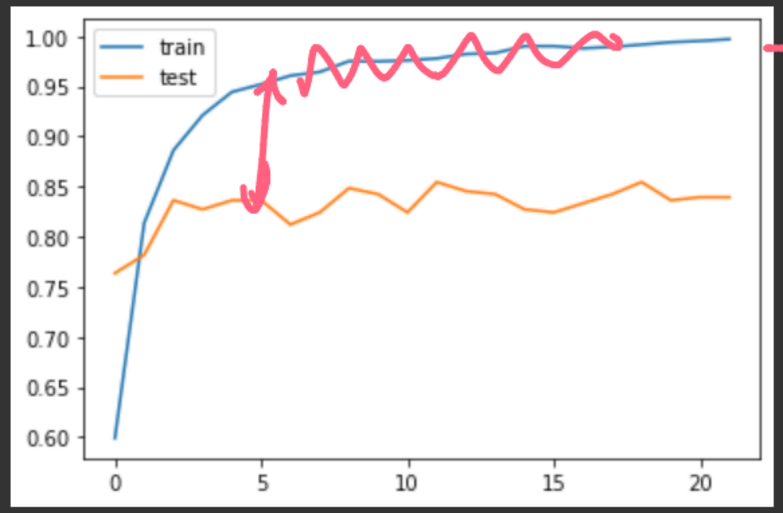
```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================
aspect input text (InputLayer)  [(None, 2)]          0         []

review input text (InputLayer)  [(None, 20)]         0         []

embedding_layer (Embedding)  multiple                873600    ['review input text[0][0]',
                                                                'aspect input text[0][0]']

gru_4 (GRU)                  [(None, 2, 64),         70272     ['embedding_layer[1][0]']
                              (None, 64)]

gru_5 (GRU)                  [(None, 20, 64),        70272     ['embedding_layer[0][0]']
                              (None, 64)]

attention (Attention)        ((None, 64),            16769     ['gru_4[0][1]',
                              (None, 20, 1))                   'gru_5[0][0]']

dense_15 (Dense)             (None, 64)              4160      ['attention[0][0]']

dropout_2 (Dropout)          (None, 64)              0         ['dense_15[0][0]']

dense_16 (Dense)             (None, 16)              1040      ['dropout_2[0][0]']

dense_17 (Dense)             (None, 3)               51        ['dense_16[0][0]']
```



```
aspect_input = tf.keras.Input(shape=(2, ), dtype="float64", name = "aspect_input_text")

review_embeddings = embedding_layer(review_input)
aspect_embeddings = embedding_layer(aspect_input)


_, aspect = tf.keras.layers.GRU(64,
  return_sequences=True, # return the full output sequence
  return_state=True, # return the last hidden state
)(aspect_embeddings)
lstm = tf.keras.layers.GRU(64,
  return_sequences=True,
  return_state=True,
)
whole_seq_output,_ = lstm(review_embeddings)



context_vector, attention_weights = Attention(128)(aspect, whole_seq_output)
dense_con = tf.keras.layers.Dense(64, activation="selu")(context_vector)
dropout = tf.keras.layers.Dropout(0.5)(dense_con) # Adding dropout layer to reduce overfitting
dense_output = tf.keras.layers.Dense(16, activation="selu")(dropout)
output = tf.keras.layers.Dense(3, activation="softmax")(dense_output)


model = tf.keras.Model(inputs=[review_input, aspect_input], outputs=output)
```

Train and Val set Preparartion

```
[ ]  from matplotlib import pyplot

     pyplot.plot(history.history['acc'], label='train')
     pyplot.plot(history.history['val_acc'], label='test')
     pyplot.legend()
     pyplot.show()
```



Inferencing

```
[ ]  # load the most accurate model, following val_acc curve
     model.load_weights('aspect_based_sa.h5')

     label_map = {
```

```
Model: "model_4"
_________________________________________________________________________________________________
 Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
 aspect input text (InputLayer)  [(None, 2)]          0           []

 review input text (InputLayer)  [(None, 20)]         0           []

 embedding_layer (Embedding)    multiple              873600      ['review input text[0][0]',
                                                                   'aspect input text[0][0]']

 gru_4 (GRU)                    [(None, 2, 64),       70272       ['embedding_layer[1][0]']
                                 (None, 64)]

 gru_5 (GRU)                    [(None, 20, 64),      70272       ['embedding_layer[0][0]']
                                 (None, 64)]

 attention (Attention)          ((None, 64),          16769       ['gru_4[0][1]',
                                 (None, 20, 1))                   'gru_5[0][0]']

 dense_15 (Dense)               (None, 64)            4160        ['attention[0][0]']

 dropout_2 (Dropout)            (None, 64)            0           ['dense_15[0][0]']

 dense_16 (Dense)               (None, 16)            1040        ['dropout_2[0][0]']

 dense_17 (Dense)               (None, 3)             51          ['dense_16[0][0]']

=================================================================================================
Total params: 1,036,164
Trainable params: 162,564
Non-trainable params: 873,600
_________________________________________________________________________________________________
```

recurrent drop

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Intro_to_Attention_Mechanism | L10_Transformers_re.ipynb - C | The Illustrated Transformer – J | Neural machine translation wit

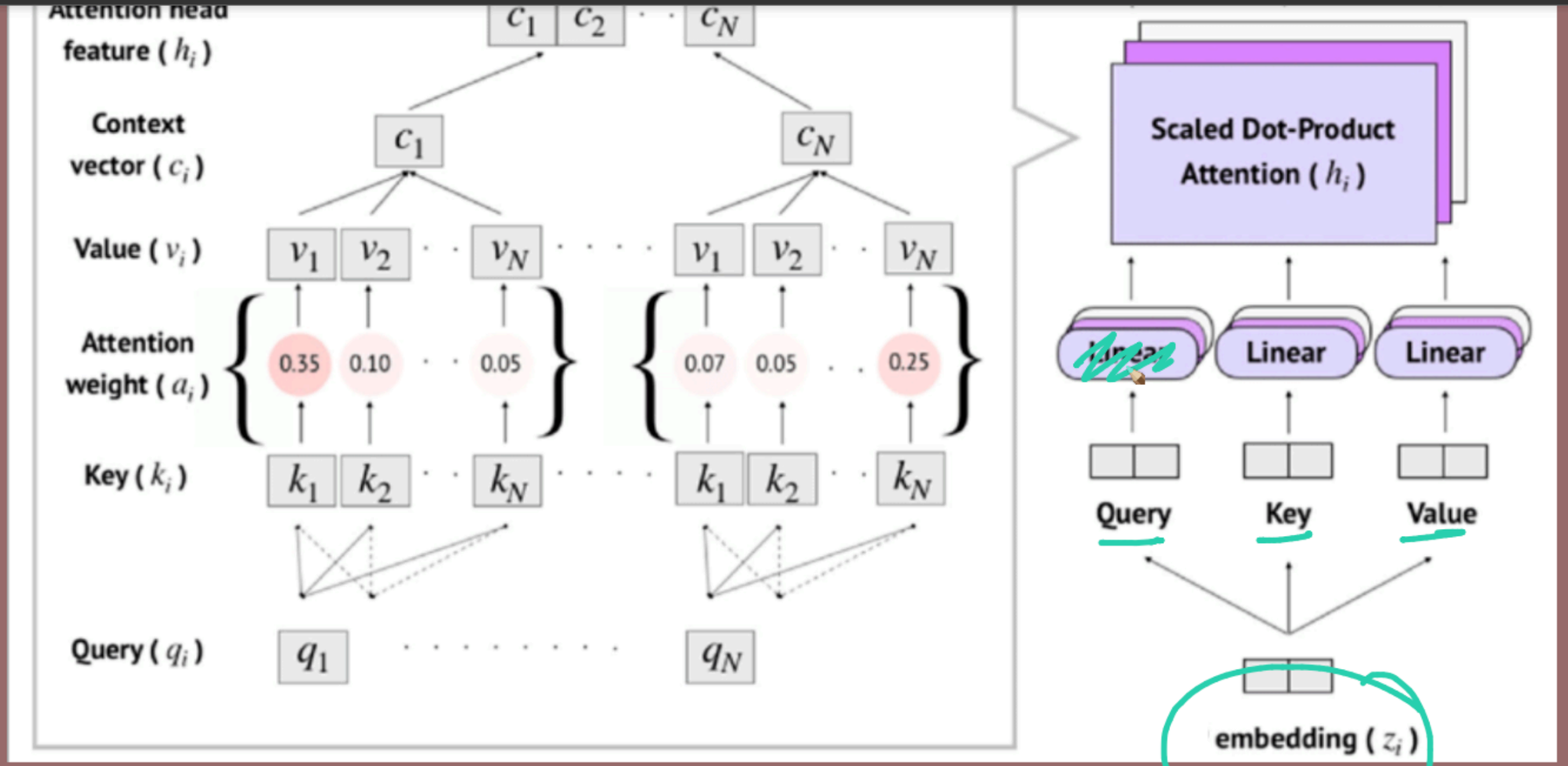
colab.research.google.com/drive/1ebHZKI6vDyLFR8BS2VPchQLCz5ywF0d4#scrollTo=ZnoWCOOyu0bZ

+ Code + Text Connect

```
Model: "model_4"
_________________________________________________________________________________________________
 Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
 aspect input text (InputLayer)  [(None, 2)]          0           []

 review input text (InputLayer)  [(None, 20)]         0           []

 embedding_layer (Embedding)    multiple              873600      ['review input text[0][0]',
                                                                   'aspect input text[0][0]']

 gru_4 (GRU)                    [(None, 2, 64),       70272       ['embedding_layer[1][0]']
                                 (None, 64)]

 gru_5 (GRU)                    [(None, 20, 64),      70272       ['embedding_layer[0][0]']
                                 (None, 64)]

 attention (Attention)          ((None, 64),          16769       ['gru_4[0][1]',
                                 (None, 20, 1))                   'gru_5[0][0]']

 dense_15 (Dense)               (None, 64)            4160        ['attention[0][0]']

 dropout_2 (Dropout)            (None, 64)            0           ['dense_15[0][0]']

 dense_16 (Dense)               (None, 16)            1040        ['dropout_2[0][0]']

 dense_17 (Dense)               (None, 3)             51          ['dense_16[0][0]']

=================================================================================================
Total params: 1,036,164
Trainable params: 162,564
Non-trainable params: 873,600
_________________________________________________________________________________________________
```

dense { W^a W^k W^v

32 / 34



3 Linear Layer in right is basically represents KEY, QUERY and VALUE Matrix

```
x1 = tokenizer.texts_to_sequences([['service']])
x1 = tf.keras.preprocessing.sequence.pad_sequences(x1, maxlen=2)
preds = model.predict([x,x1])
print('*'*30,'aspect-term : service', '*'*30)
print(label_map[np.argmax(preds)])
```

```
Input String :  the service was worst , but ambience was amazing
****************************** aspect-term : ambience ******************************
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7fcff424f440> trigg
1/1 [==============================] - 1s 1s/step
Positive
1/1 [==============================] - 0s 23ms/step
****************************** aspect-term : service ******************************
Negative
```

Attention Visualization

```
[ ] attention_model = tf.keras.Model(inputs=model.input, outputs=[model.output, model.get_layer('attention').output, model.input])
```

```
[ ] def plot_attention(target):

      # plot attention curve

      #define input output
      x1 = tokenizer.texts_to_sequences(
```

```
preds = model.predict([x,x1])
print('*'*30,'aspect-term : service', '*'*30)
print(label_map[np.argmax(preds)])
```

```
Input String :  the service was worst , but ambience was amazing
****************************** aspect-term : ambience ******************************
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7fcff424f440> trigg
1/1 [==============================] - 1s 1s/step
Positive
1/1 [==============================] - 0s 23ms/step
****************************** aspect-term : service ******************************
Negative
```

Attention Visualization

```
[ ] attention_model = tf.keras.Model(inputs=model.input, outputs=[model.output, model.get_layer('attention').output, model.input])
```

```
[ ] def plot_attention(target):

      # plot attention curve

      #define input output
      x1 = tokenizer.texts_to_sequences([[target]])
      x1 = tf.keras.preprocessing.sequence.pad_sequences(x1, maxlen=2)

      preds = model.predict([x,x1])
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Intro_to_Attention_Mechanism | L10_Transformers_re.ipynb - C | The Illustrated Transformer – J | Neural machine translation wit

colab.research.google.com/drive/1ebHZKI6vDyLFR8BS2VPchQLCz5ywF0d4#scrollTo=4KXyOP_Zyer5

Update

+ Code + Text

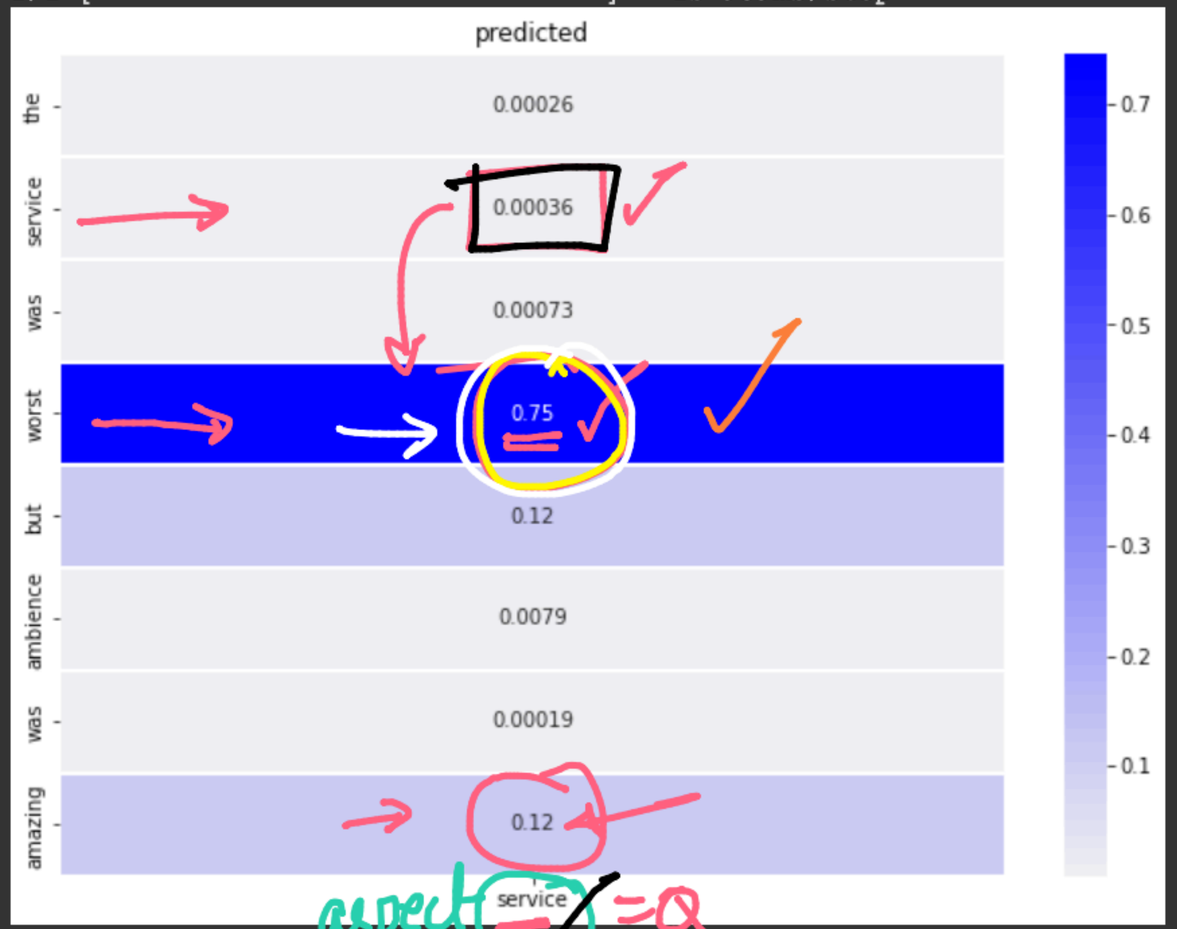
Connect

********** aspect-term : service **********

Negative

WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7fcff40560e0> trigg

1/1 [==============================] - 1s 959ms/step



K

aspect = Q

softmax

polarity

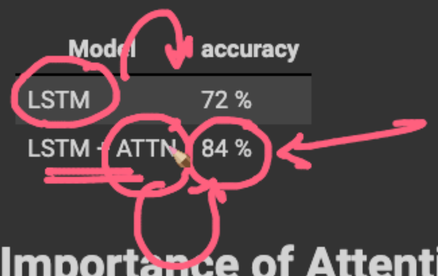
36 / 38

Explainability

Model	Explainability
LSTM	Not possible
LSTM + ATTN	possible to visualize

Performance

Model	accuracy
LSTM	72 %
LSTM + ATTN	84 %



Importance of Attention Mechanisms in training Popular NLP Model

- helps in learning good contexual representation of a text
- Adding Explainability
- helps in performance improvement
- Facilitates transfer learning

TRANSFORMERS

- Language translation Model
- based on `MULTI-HEAD SELF-ATTENTION MECHANISMS`
- Get rid of recurrent architecture to understand text

Explainability

Model	Explainability
LSTM	Not possible
LSTM + ATTN	possible to visualize

Performance

Model	accuracy
LSTM	72 %
LSTM + ATTN	84 %

Importance of Attention Mechanisms in training Popular NLP Model

- helps in learning good contexual representation of a text
- Adding Explainability
- helps in performance improvement
- Facilitates transfer learning

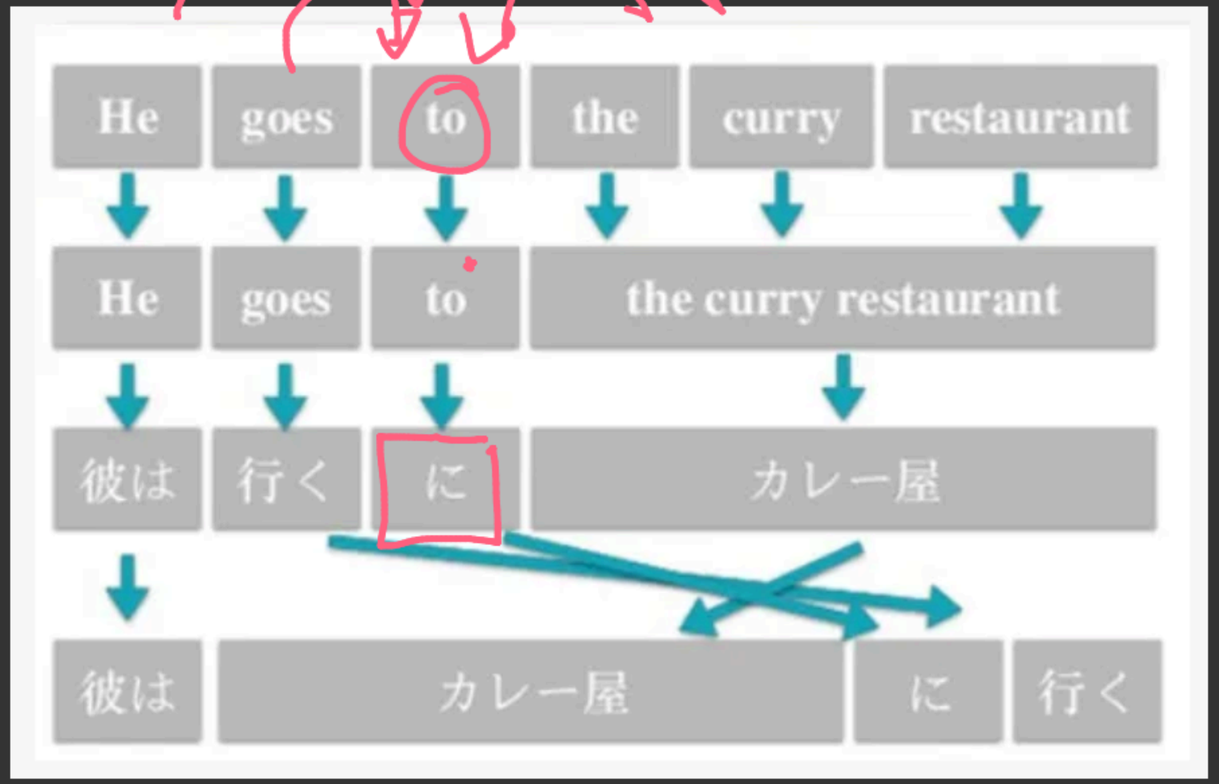
TRANSFORMERS

- Language translation Model
- based on `MULTI-HEAD SELF-ATTENTION MECHANISMS`
- Get rid of recurrent architecture to understand text

RERT

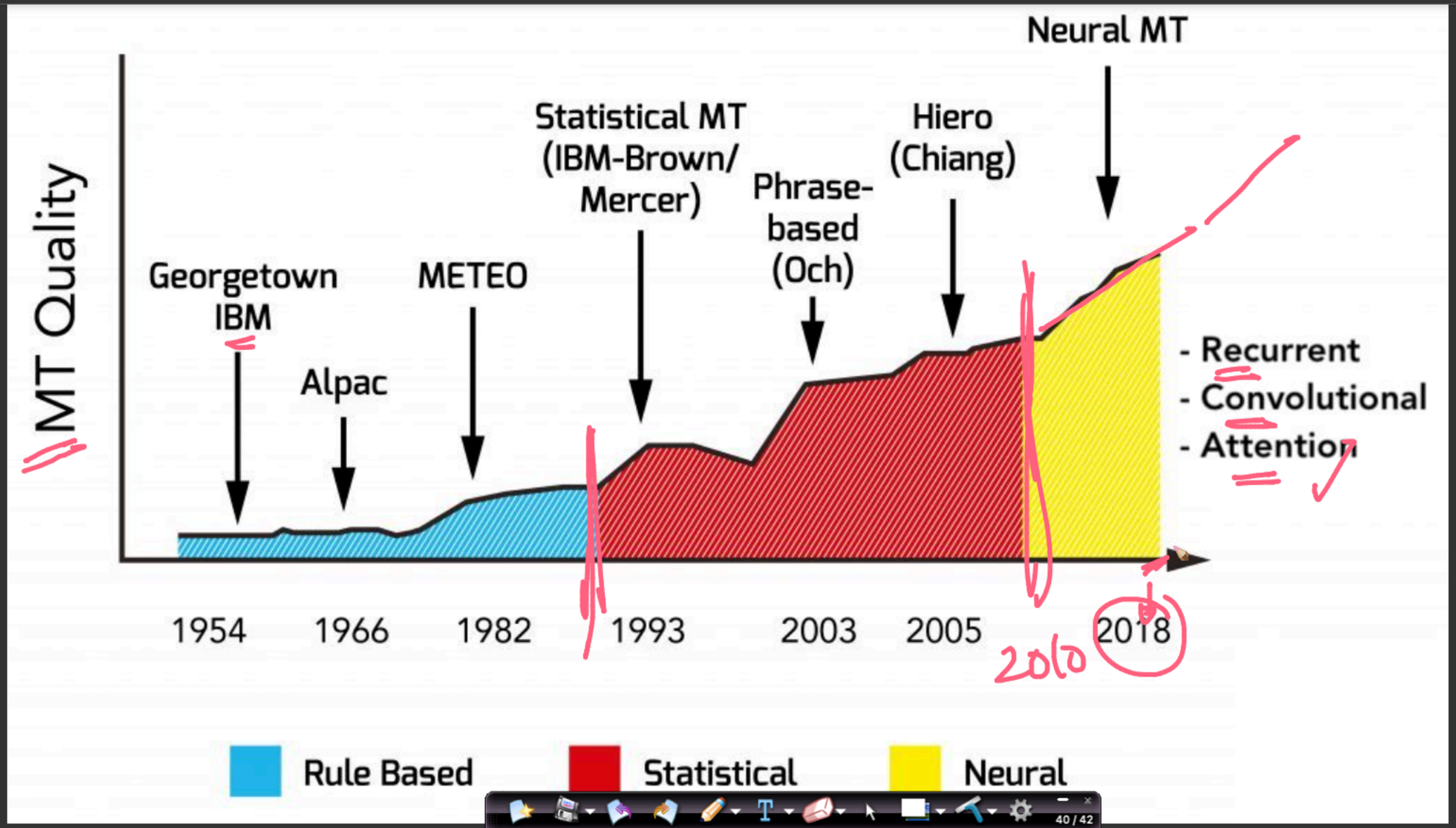
- Sentence is devided into small units of words and phrases.
- Statistical Models learn how a specific source word is translated into different word in target language.
- Models give more weightage to the source-target pair which comes maximum times in corpus.

How it works:



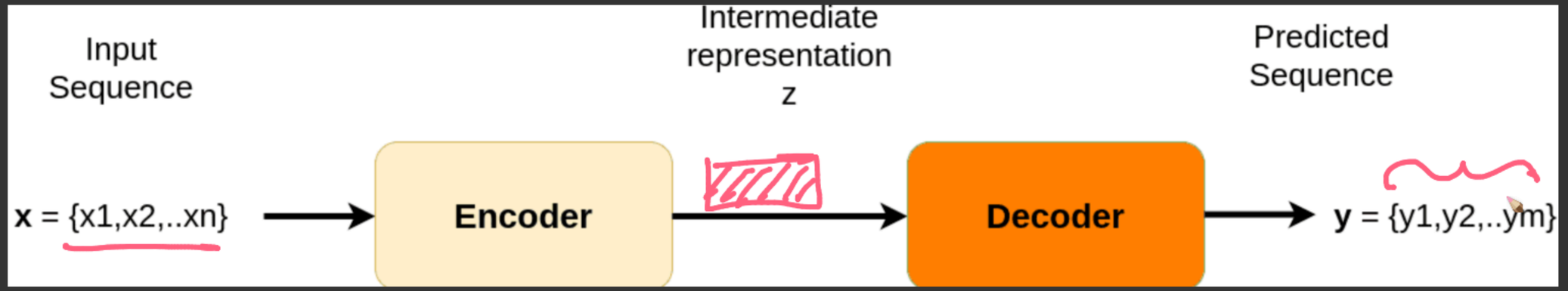
$$P(y_t \mid x_t x_{t+1} x_{t+2} \ldots)$$

- Corpus is created which contains very large data set of approved previous translations.
- Translation Model(TM) is build using the f



What is Encoder-Decoder Architecture?

Encoder-Decoder Architecture

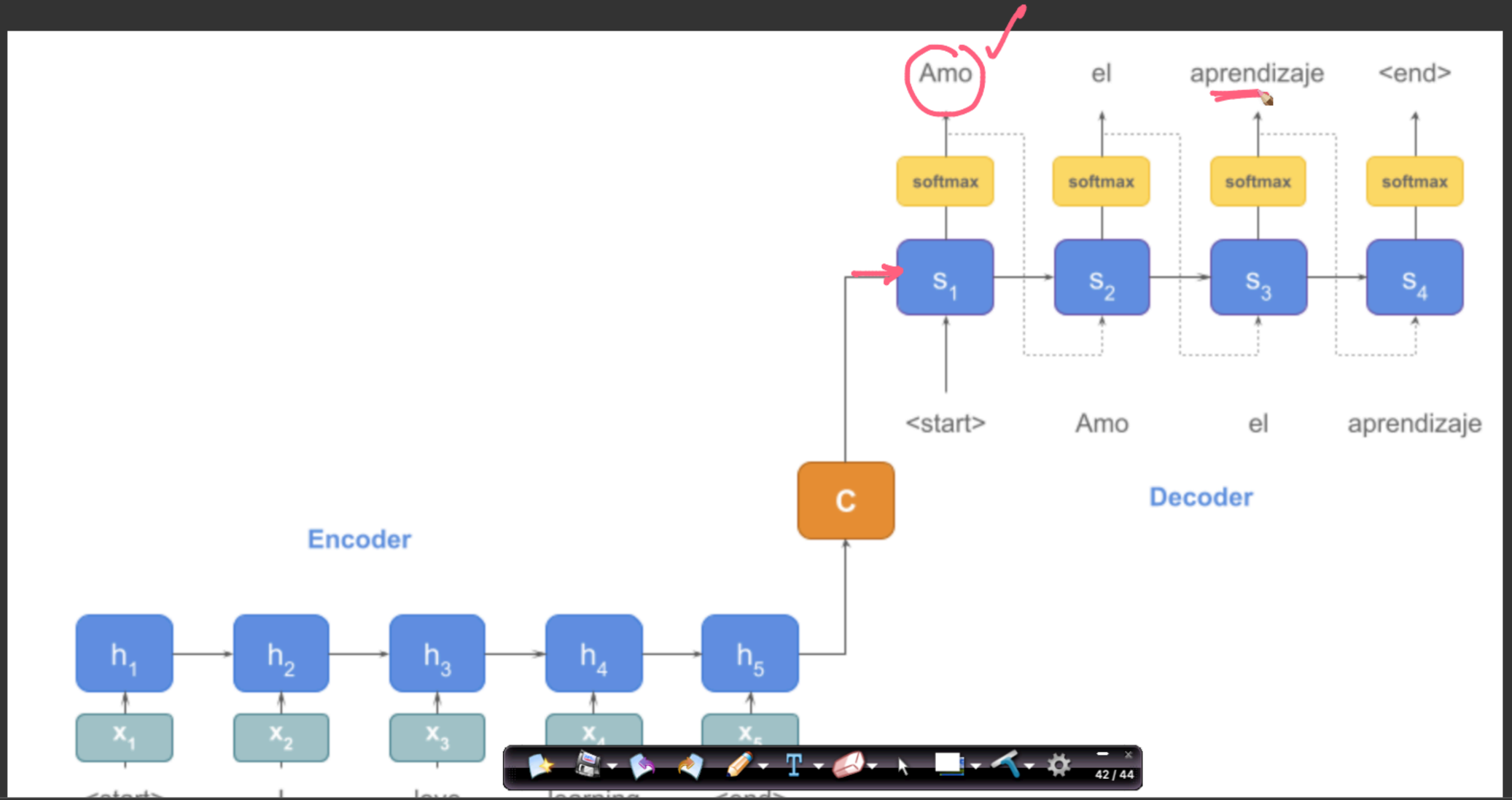


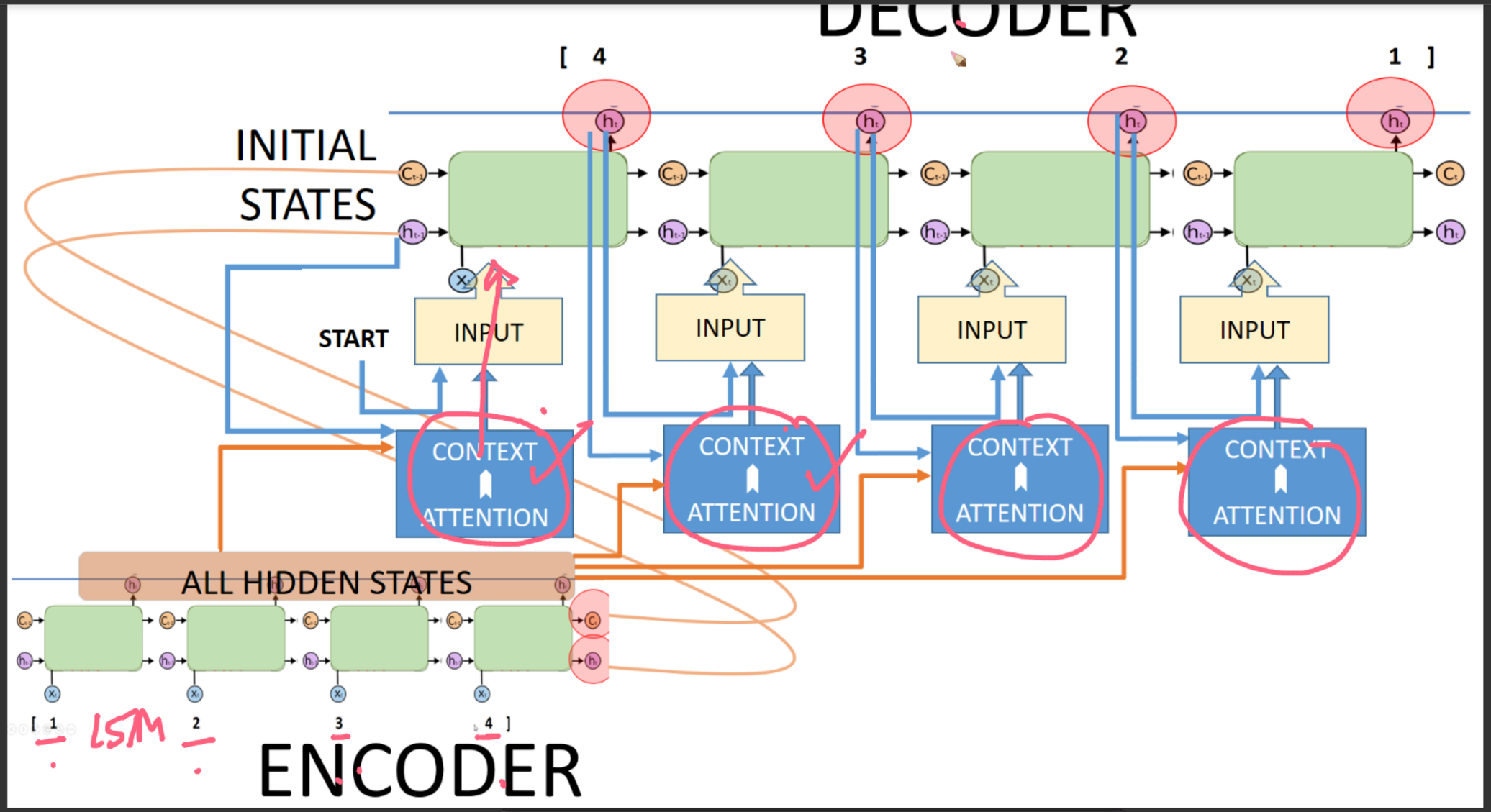
Consist of:

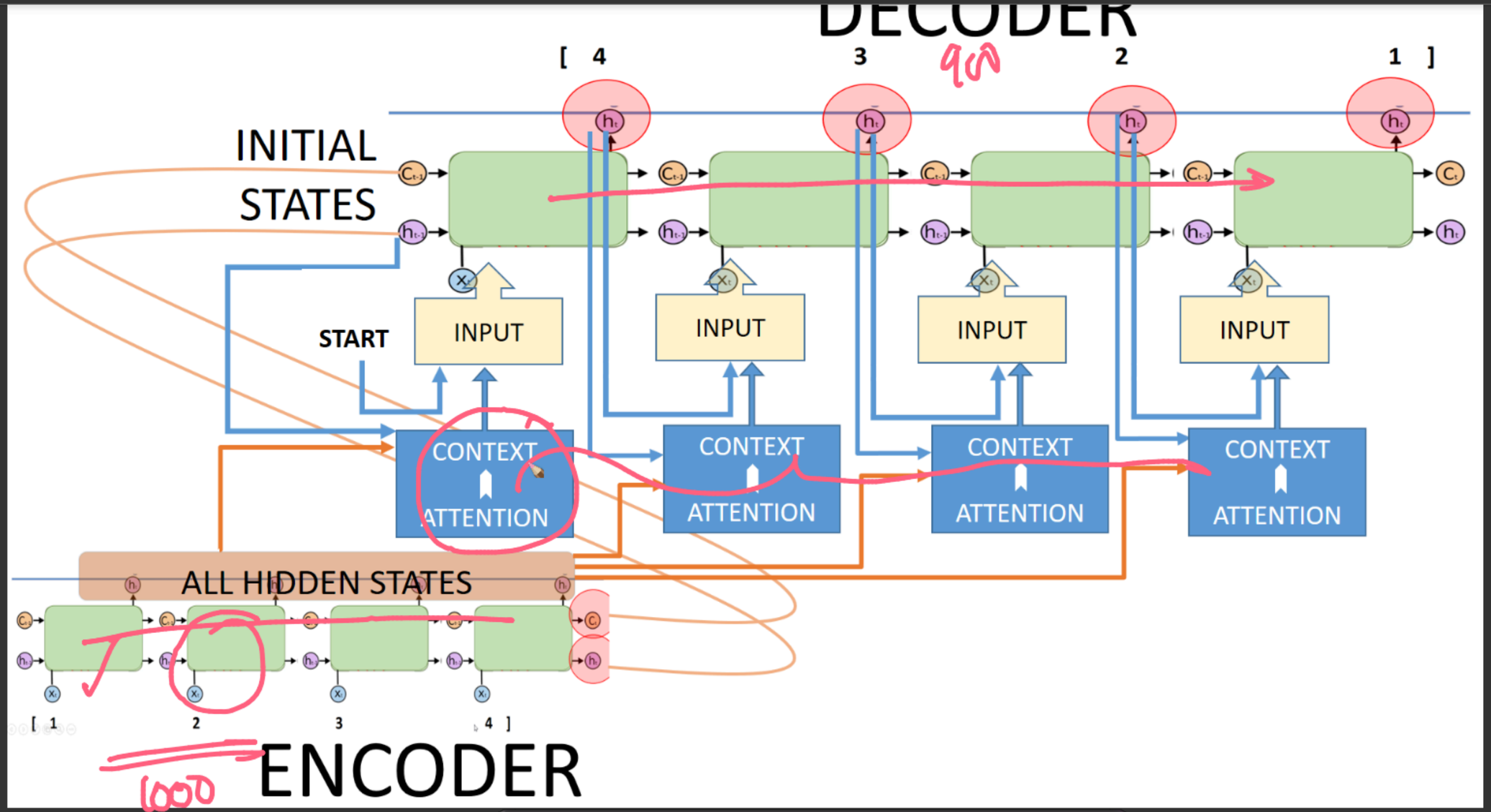
- Encoder Component
- Encoder Vector/Hidden State Vector/Context Vector
- Decoder Component

Encoder

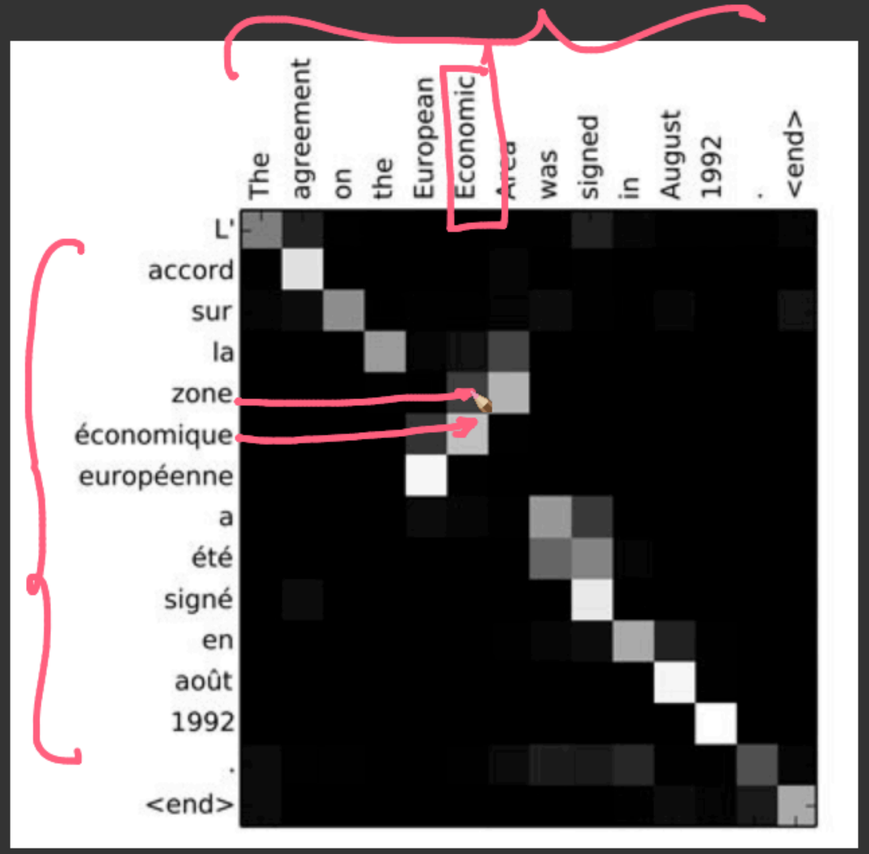
- The Encoder summarizes the input sequence into Encoded vectors
- Decoder uses Encoded vectors and Generates output sequences using softmax







```
The Attention Mechanism can be viewed as a method
for making the RNN work better by letting the network
 know where to look as it is performing its task.
```



Let's See our translation problems:

- Translating a sentence from English to Fre etwork is "paying attention" as it

Property	RNN/LSTM	RNN/LSTM+Attention	Transformers
Recurrence blocks	Yes	Yes	No
Attention	No	Yes	Yes
context vector	last timestamp state vector	calculated using attention mechanisms	using attention mechanisms
attention alighment function	NA	Dot product	Scaled Dot Product
attention type	NA	single head	Multi head
operation	sequential	sequential	parallel

What are Transformers Components?

Following are the components of Transformers:

1. Position Encodings
2. Transformers Encoders
3. Multi-headed Attention
4. Transformers Decoders

Property	RNN/LSTM	RNN/LSTM+Attention	Transformers
Recurrence blocks	Yes	Yes	No
Attention	No	Yes	Yes
context vector	last timestamp state vector	calculated using attention mechanisms	using attention mechanisms
attention alighment function	NA	Dot product	Scaled Dot Product
attention type	NA	single head	Multi head
operation	sequential	sequential	parallel

What are Transformers Components?

Following are the components of Transformers:

1. Position Encodings
2. Transformers Encoders
3. Multi-headed Attention
4. Transformers Decoders

attention alighment function	NA	Dot product	Scaled Dot Produ
attention type	NA	single head	Multi head
operation	sequential	sequential	parallel

What are Transformers Components?

Following are the components of Transformers:

1. Position Encodings
2. Transformers Encoders
3. Multi-headed Attention
4. Transformers Decoders
5. FFN

As Transformer does not use recurrense layer like RNN/LSTM, how it understand the relative position of sequence in the inputs?

attention alighment function	NA	Dot product	Scaled Dot Produ
attention type	NA	single head	Multi head
operation	sequential	sequential	parallel

What are Transformers Components?

Following are the components of Transformers:

1. Position Encodings
2. Transformers Encoders
3. Multi-headed Attention
4. Transformers Decoders
5. FFN

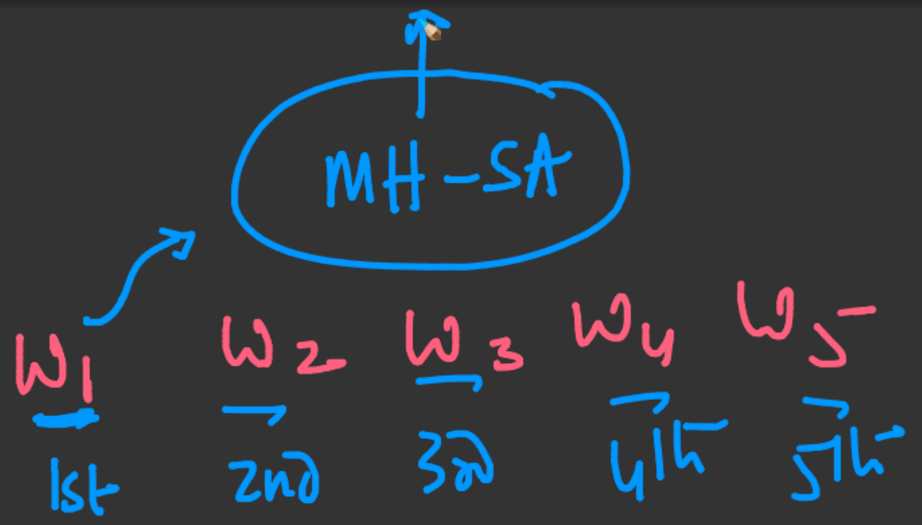
As Transformer does not use recurrense layer like RNN/LSTM, how it understand the relative position of sequence in the inputs?

attention alighment function	NA	Dot product	Scaled Dot Produ
attention type	NA	single head	Multi head
operation	sequential	sequential	parallel

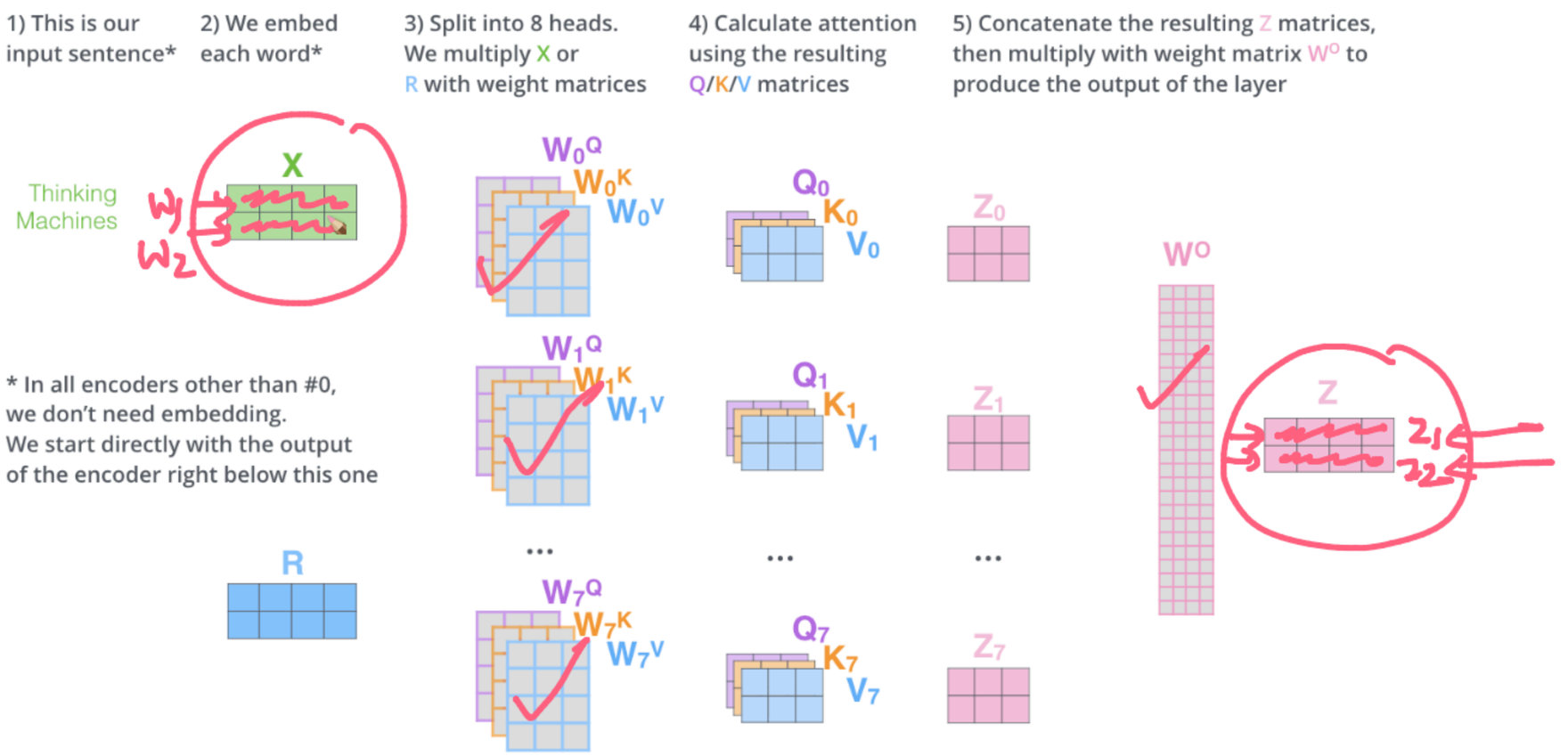
What are Transformers Components?

Following are the components of Transformers:

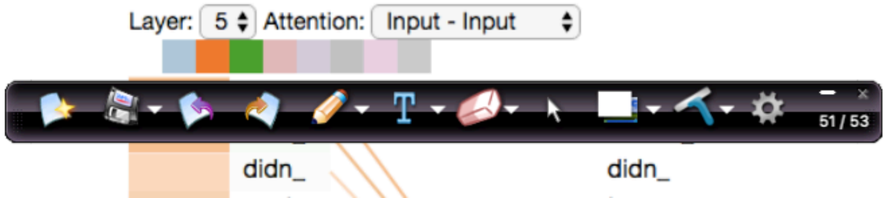
1. Position Encodings
2. Transformers Encoders
3. Multi-headed Attention
4. Transformers Decoders
5. FFN



As Transformer does not use recurrense layer like RNN/LSTM, how it understand the relative position of sequence in the inputs?



Now that we have touched upon attention heads, let's revisit our example from before to see where the different attention heads are focusing as we encode the word "it" in our example sentence:

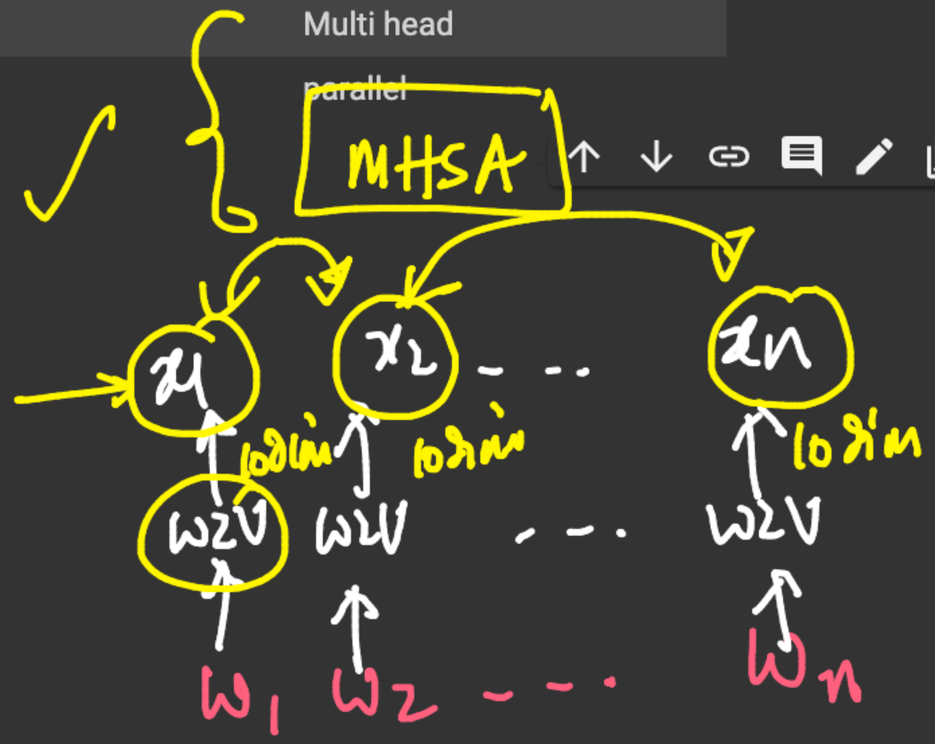


attention alighment function	NA	Dot product	Scaled Dot Product
attention type	NA	single head	Multi head
operation	sequential	sequential	parallel

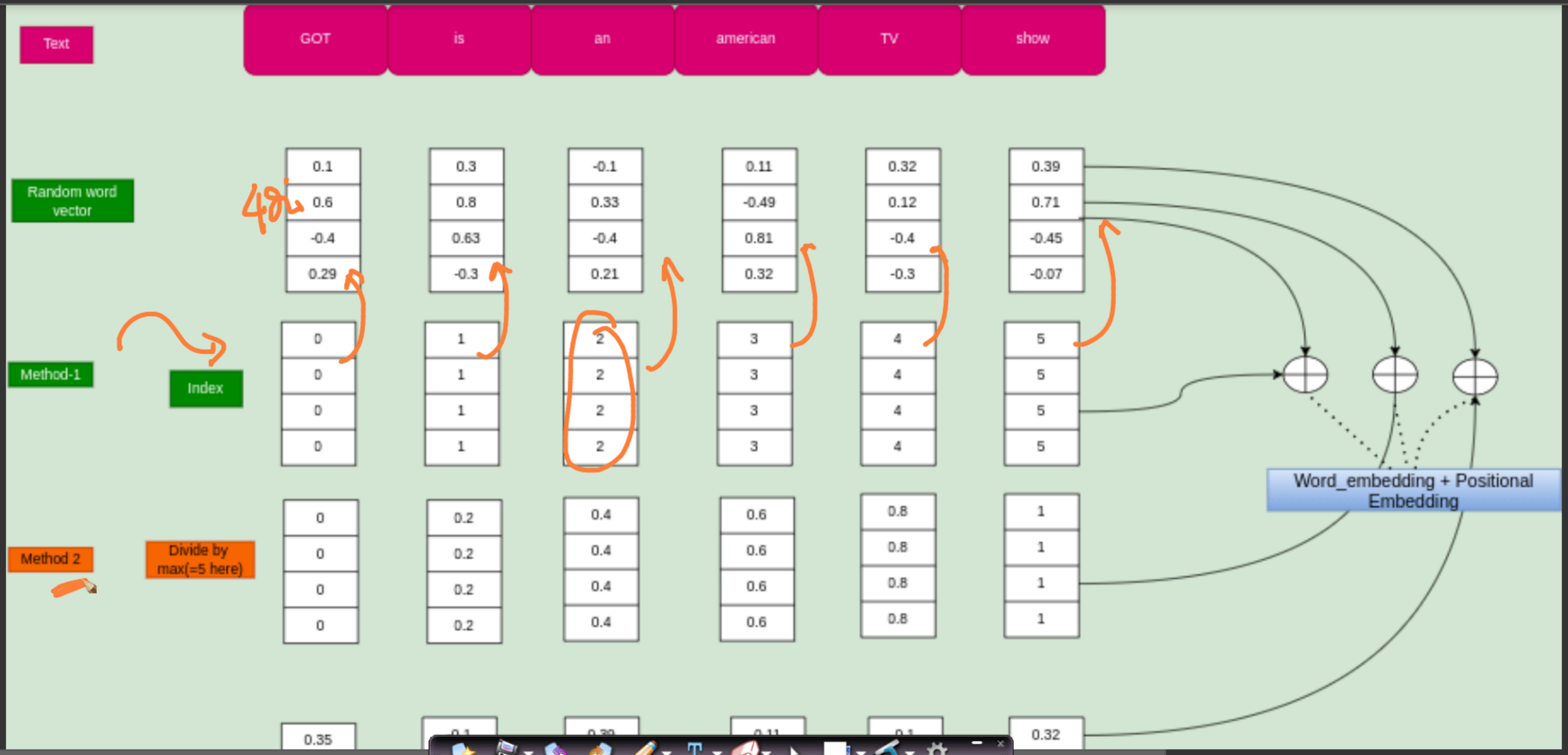
What are Transformers Components?

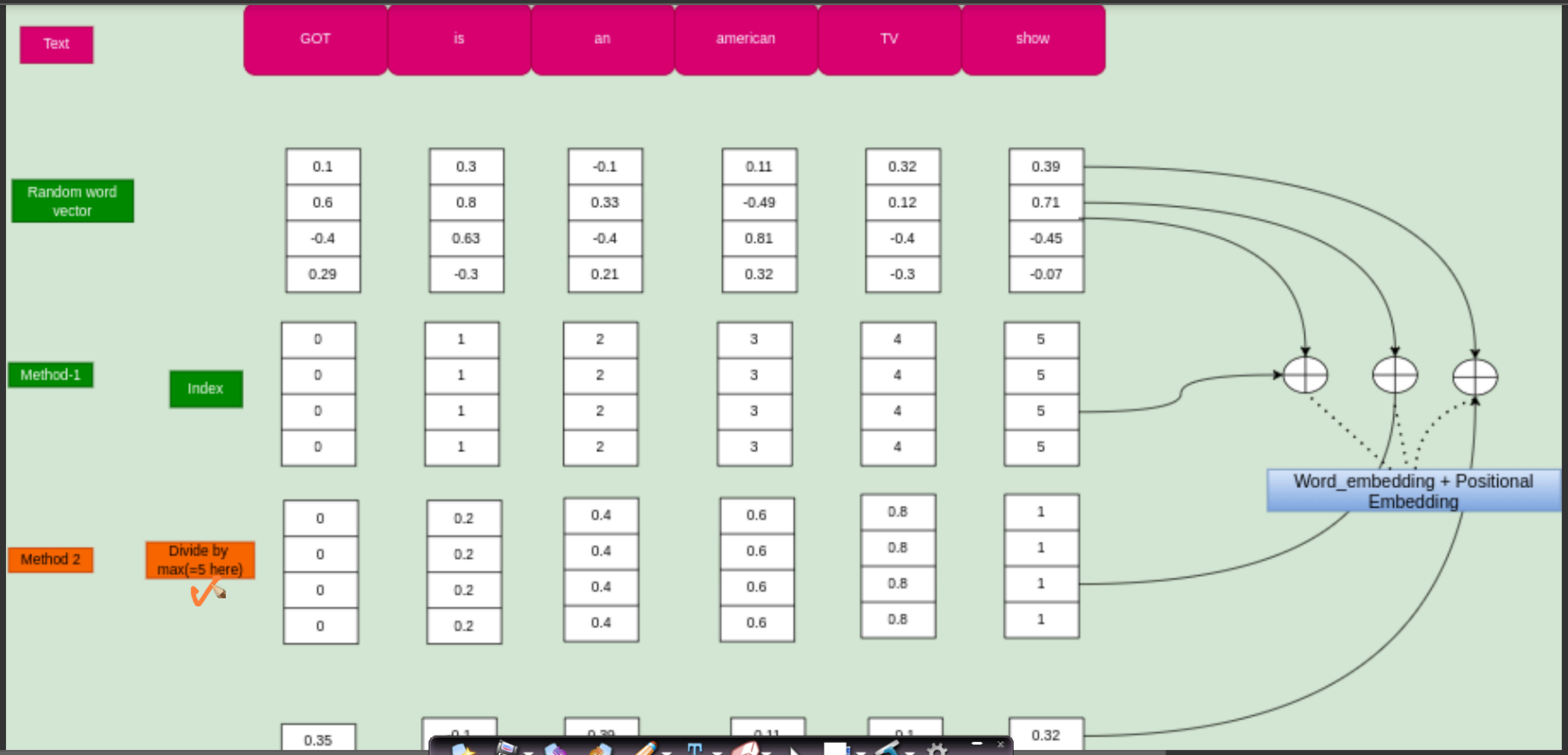
Following are the components of Transformers:

1. Position Encodings
2. Transformers Encoders
3. Multi-headed Attention
4. Transformers Decoders
5. FFN



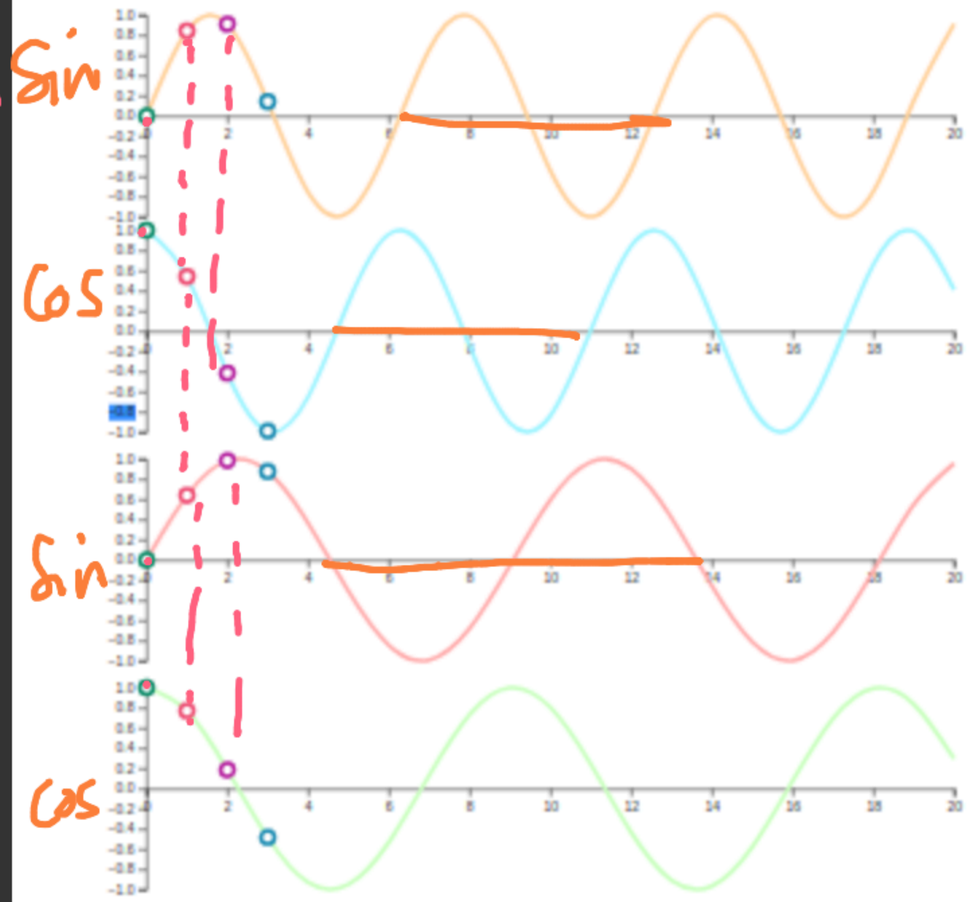
As Transformer does not use recurrense layer like RNN/LSTM, how it understand the relative position of sequence in the inputs?


Text
GOT
is
an
american
TV
show
Random word vector
0.1
0.6
-0.4
0.29
0.3
0.8
0.63
-0.3
-0.1
0.33
-0.4
0.21
0.11
-0.49
0.81
0.32
0.32
0.12
-0.4
-0.3
0.39
0.71
-0.45
-0.07
Method-1
Index
0
1
2
3
4
5
Method 2
Divide by max(=5 here)
0
0.2
0.4
0.6
0.8
1
Word_embedding + Positional Embedding
0.35
0.32


Text
GOT
is
an
american
TV
show
Random word vector
0.1
0.6
-0.4
0.29
0.3
0.8
0.63
-0.3
-0.1
0.33
-0.4
0.21
0.11
-0.49
0.81
0.32
0.32
0.12
-0.4
-0.3
0.39
0.71
-0.45
-0.07
Method-1
Index
0
0
0
0
1
1
1
1
2
2
2
2
3
3
3
3
4
4
4
4
5
5
5
5
Method 2
Divide by max(=5 here)
0
0
0
0
0.2
0.2
0.2
0.2
0.4
0.4
0.4
0.4
0.6
0.6
0.6
0.6
0.8
0.8
0.8
0.8
1
1
1
1
Word_embedding + Positional Embedding
0.35
0.32


Text
GOT
is
an
american
TV
show
Random word vector
0.1 0.6 -0.4 0.29
0.3 0.8 0.63 -0.3
-0.1 0.33 -0.4 0.21
0.11 -0.49 0.81 0.32
0.32 0.12 -0.4 -0.3
0.39 0.71 -0.45 -0.07
Method-1
Index
0 0 0 0
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
5 5 5 5
Method 2
Divide by max(=5 here)
0 0 0 0
0.2 0.2 0.2 0.2
0.4 0.4 0.4 0.4
0.6 0.6 0.6 0.6
0.8 0.8 0.8 0.8
1 1 1 1
Word_embedding + Positional Embedding
0.35
0.32

Positional encoding visualization



	p0	p1	p2	p3	
	0.000	0.841	0.909	0.141	i=0
	1.000	0.540	-0.416	-0.990	i=1
	0.000	0.638	0.983	0.875	i=2
	1.000	0.770	0.186	-0.484	i=3

Positional Encoding

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

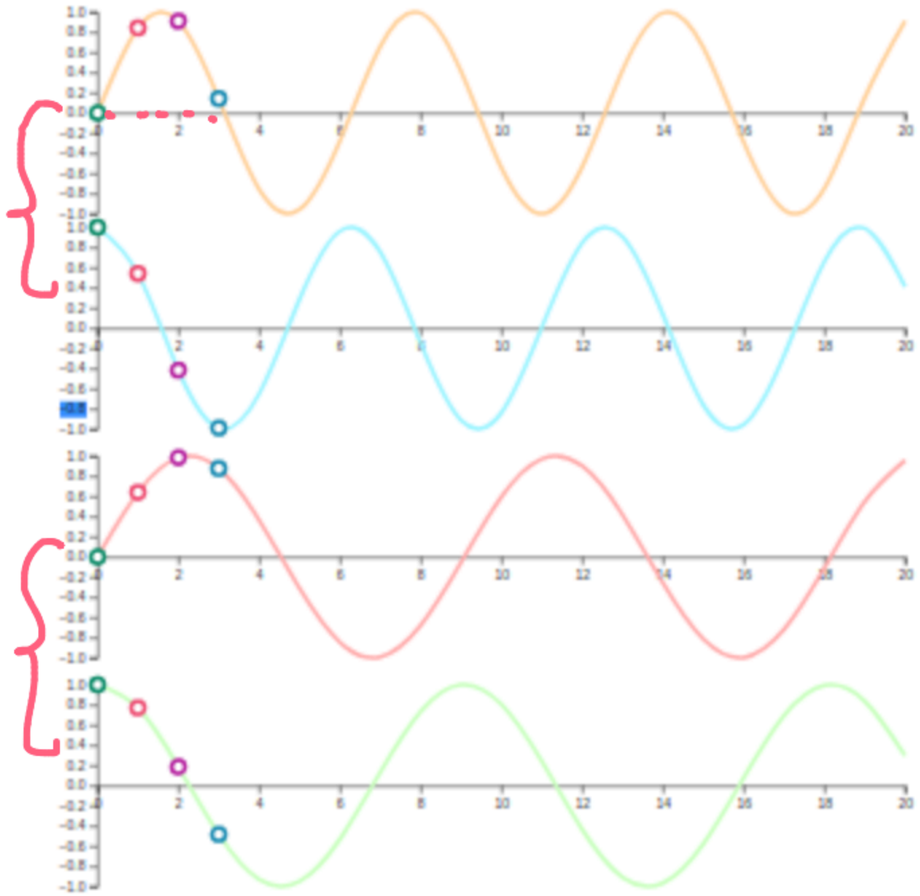
$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{model}}})$$

Settings: d = 50

The value of each positional encoding depends on the *position* (*pos*) and *dimension* (*d*). We calculate result for every *index* (*i*) to get the whole vector.

pos0 pos1 pos2 pos3

Positional encoding visualization



	p0	p1	p2	p3	
	0.000	0.841	0.909	0.141	i=0
	1.000	0.540	-0.416	-0.990	i=1
	0.000	0.638	0.983	0.875	i=2
	1.000	0.770	0.186	-0.484	i=3

Positional Encoding

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

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

Settings: d = 50

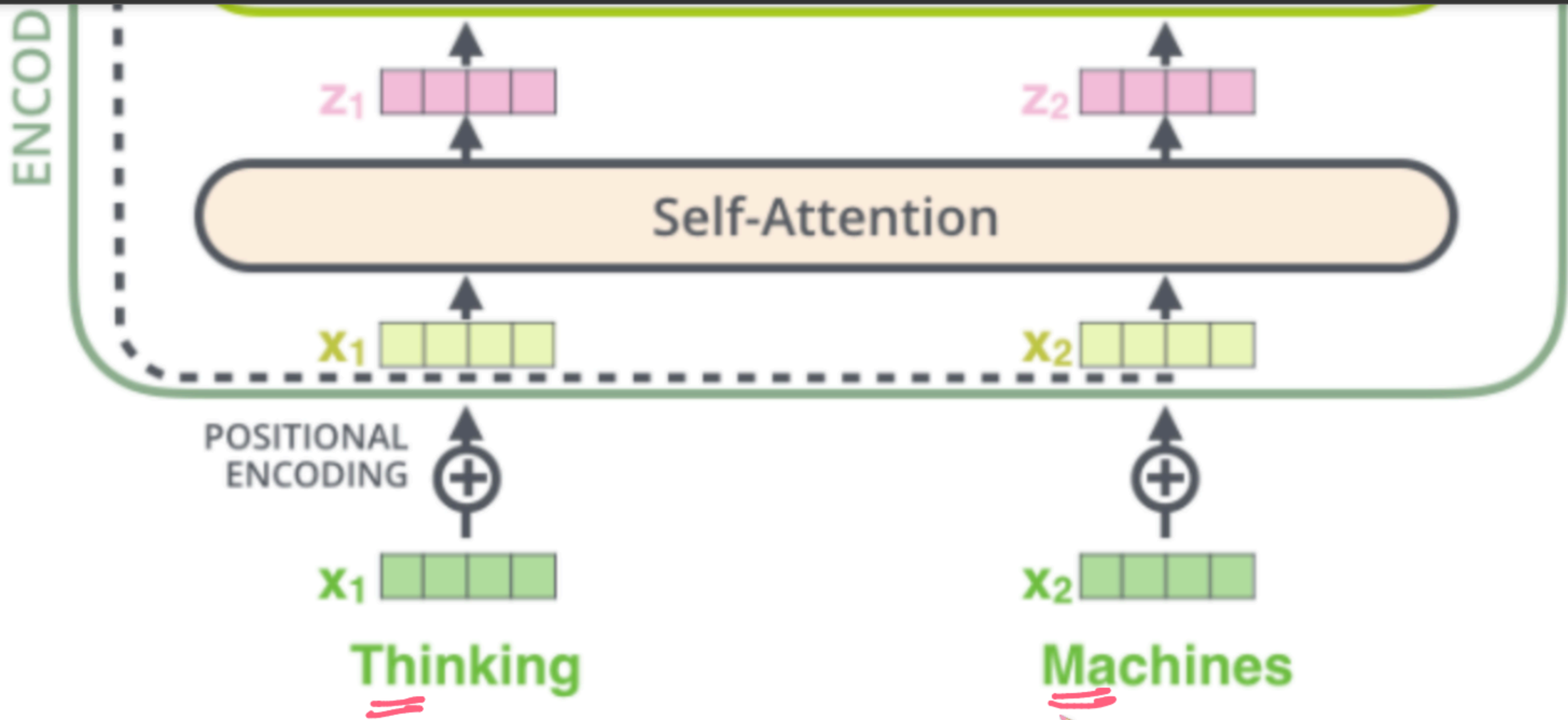
The value of each positional encoding depends on the *position* (*pos*) and *dimension* (*d*). We calculate result for every *index* (*i*) to get the whole vector.

pos0 pos1 pos2 pos3

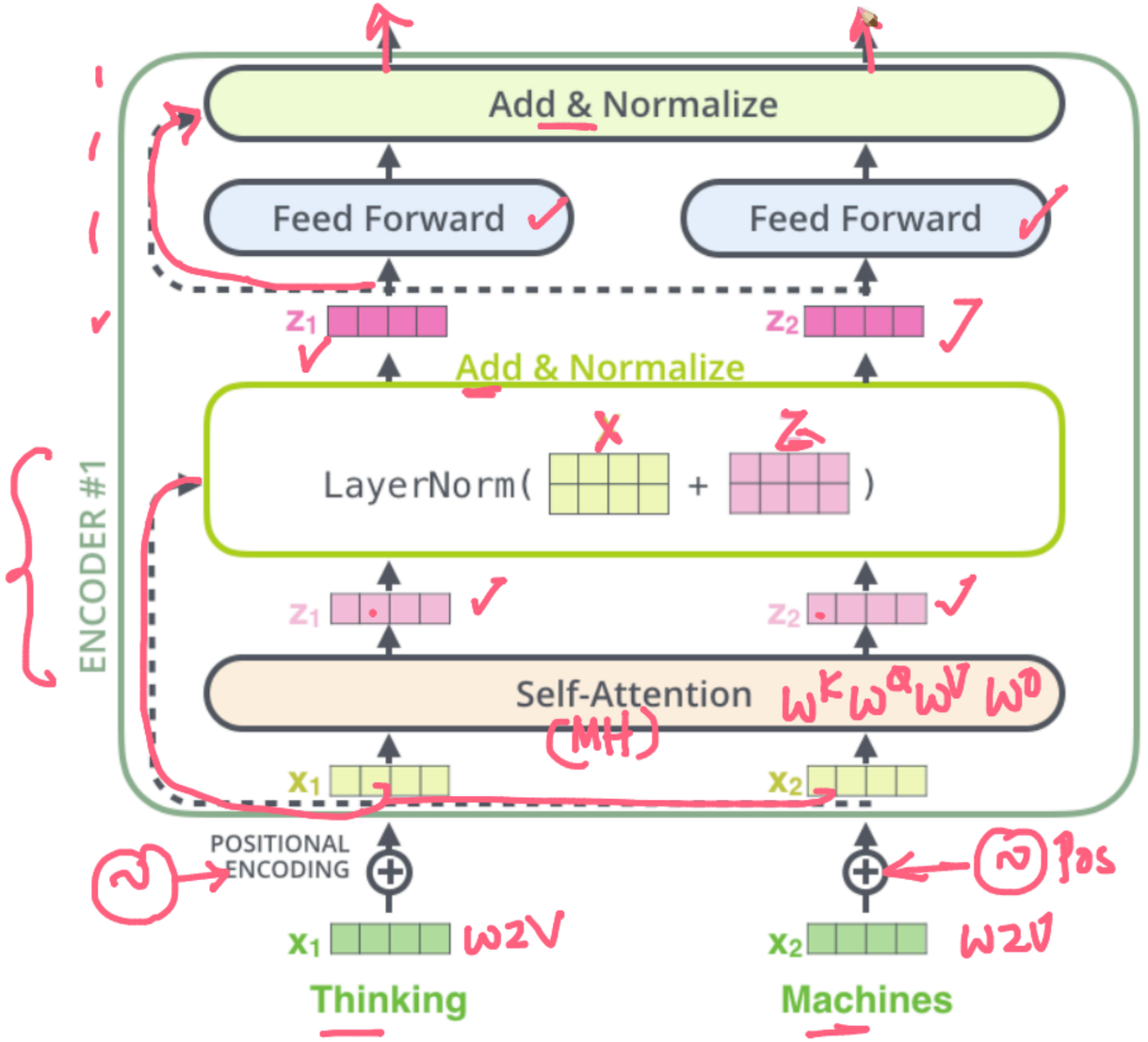
1. ±1
2. any dim-rep
3. any i/p length
4. rel. posn

pos → x_1 x_2 x_3 x_4

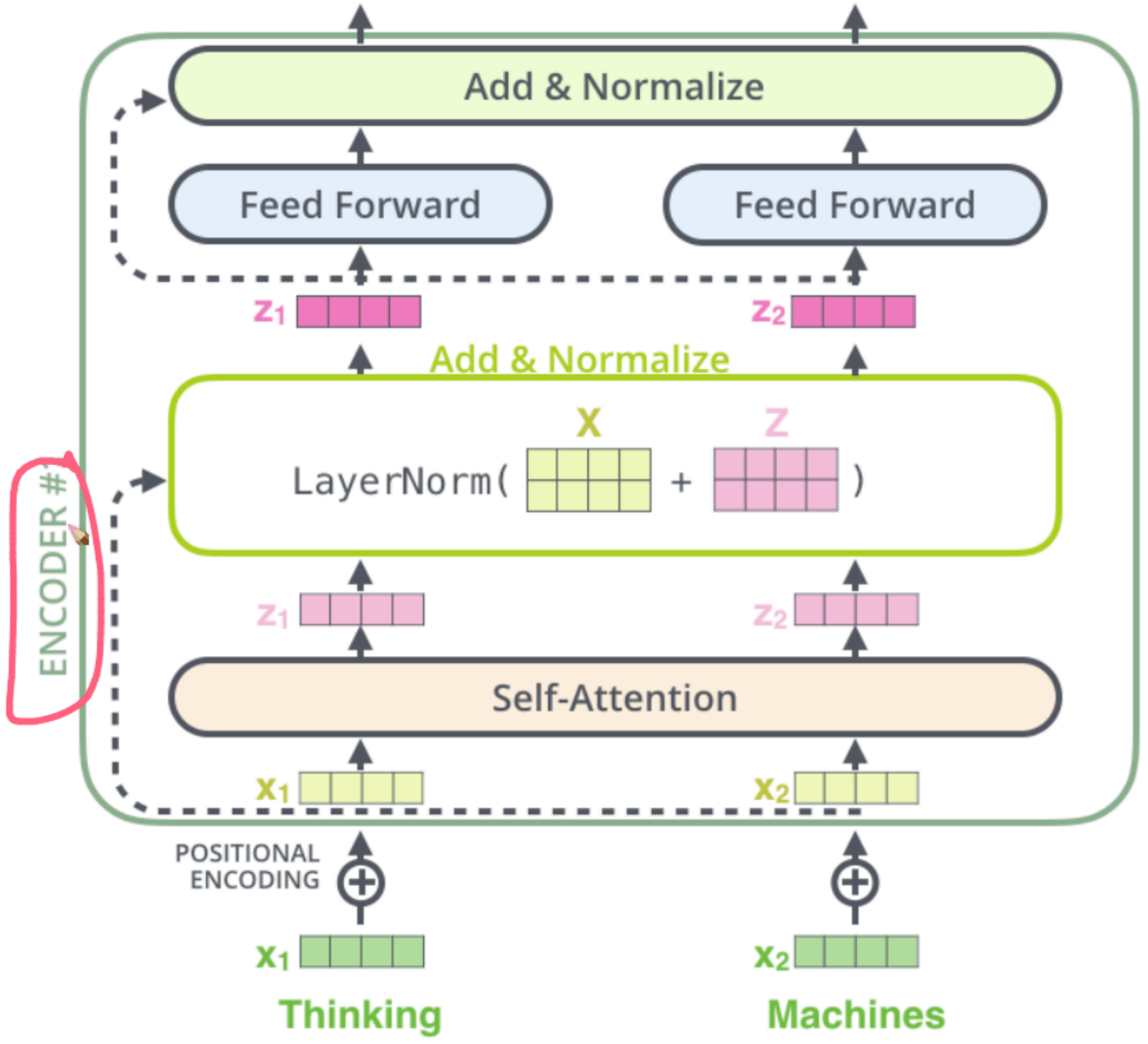
w_1 w_2 w_3 w_4



Transformer Decoder



This goes for the sub-layers of the decoder as well. If we're to think of a Transformer of 2 stacked encoders and decoders, it would look s



This goes for the sub-layers of the decoder as well. If we're to think of a Transformer of 2 stacked encoders and decoders, it would look s

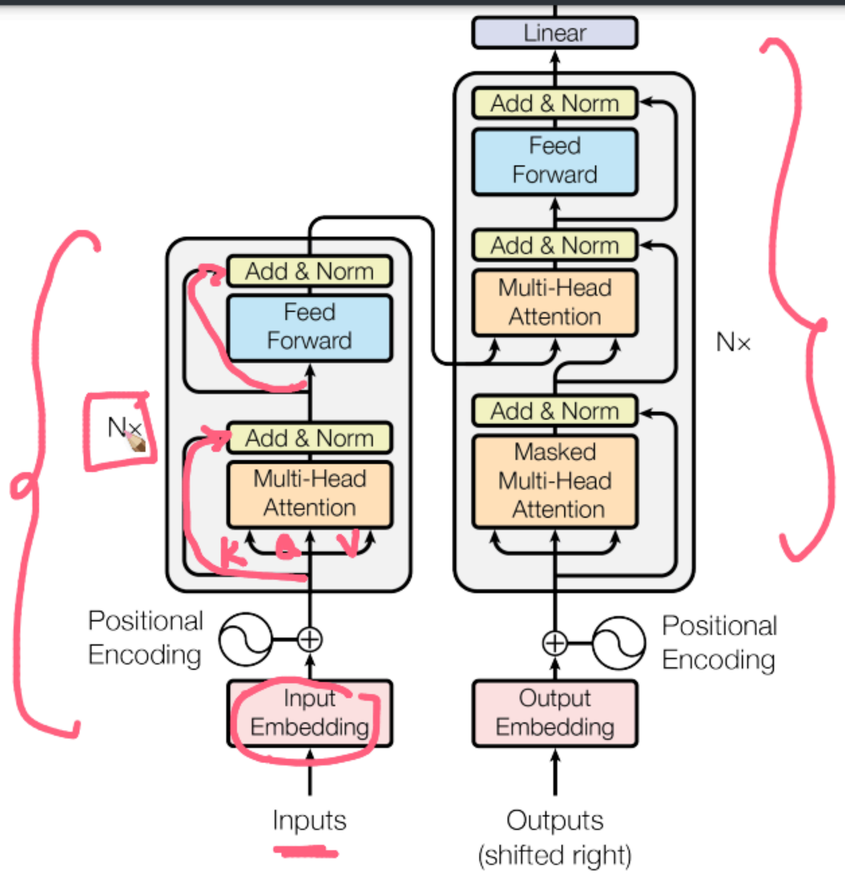
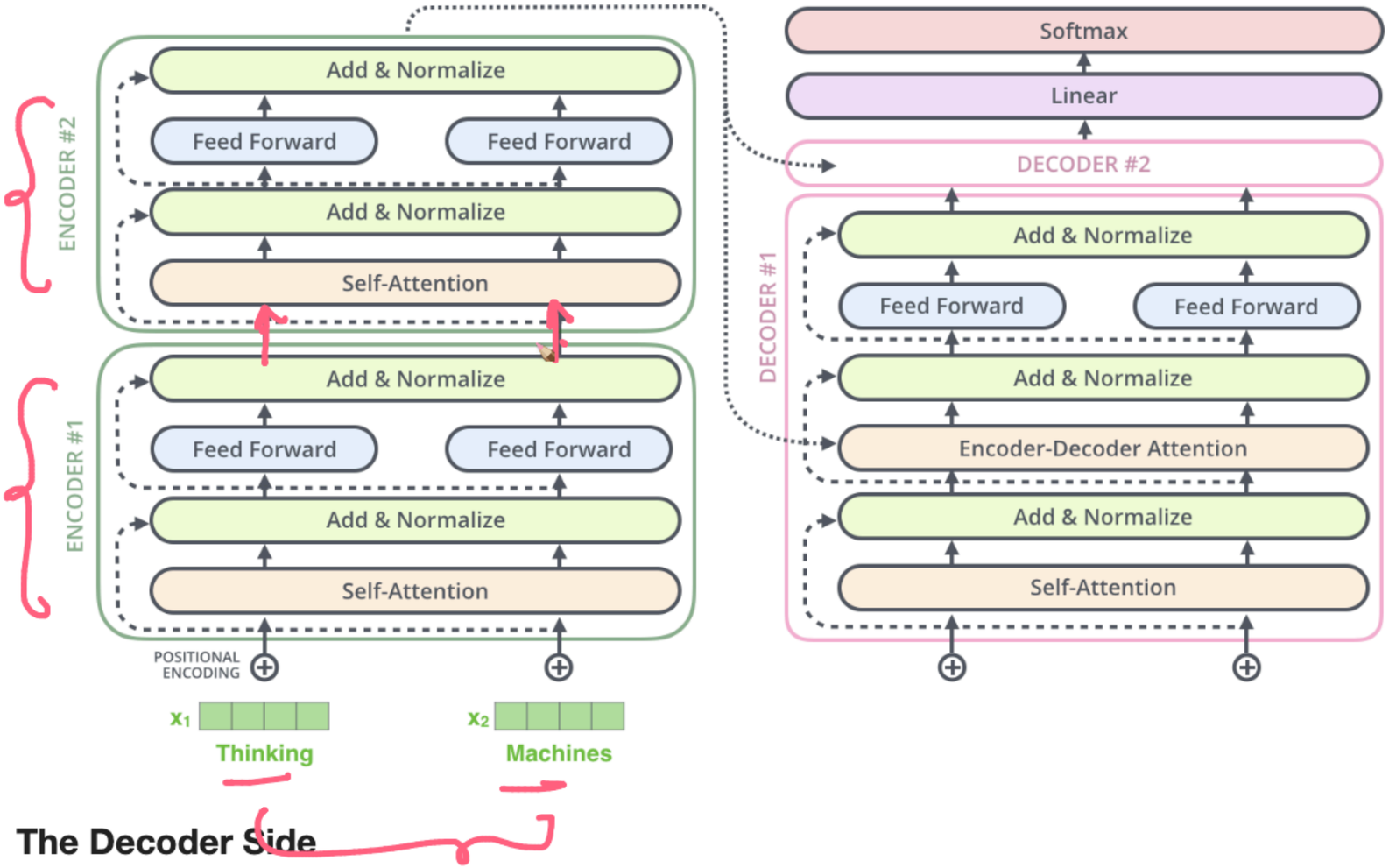


Figure 1: The Transformer - model architecture.

3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. T simple, position-wise fully co around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is

This goes for the sub-layers of the decoder as well. If we're to think of a Transformer of 2 stacked encoders and decoders, it would look something like this:

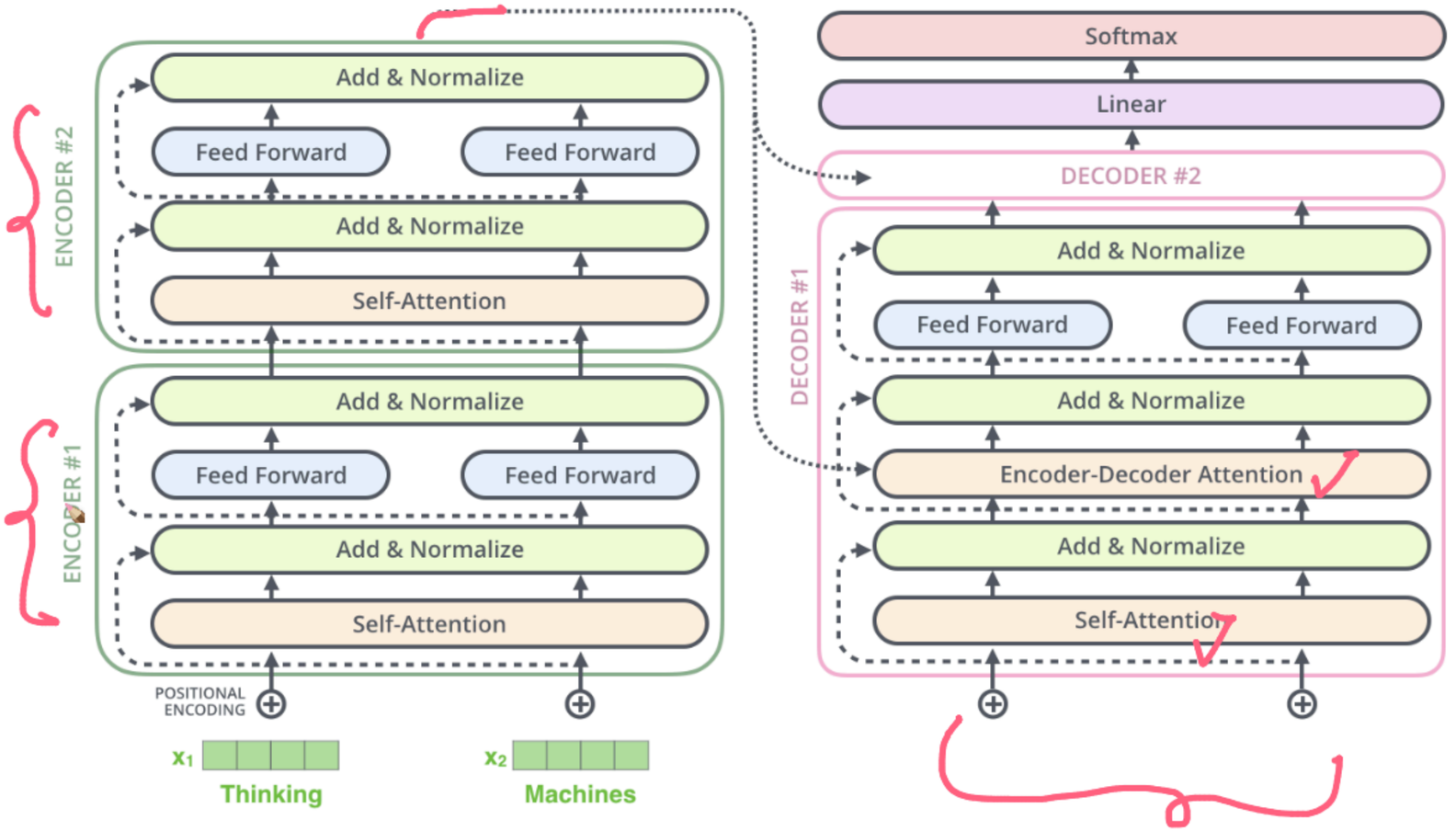


The Decoder Side

Now that we've covered most of the concepts on the encoder side, we basically know how the components of decoders work as well. B

The encoder start by processing the input sequence. The output of the top encoder is then transformed into a set of

This goes for the sub-layers of the decoder as well. If we're to think of a Transformer of 2 stacked encoders and decoders, it would look something like this:

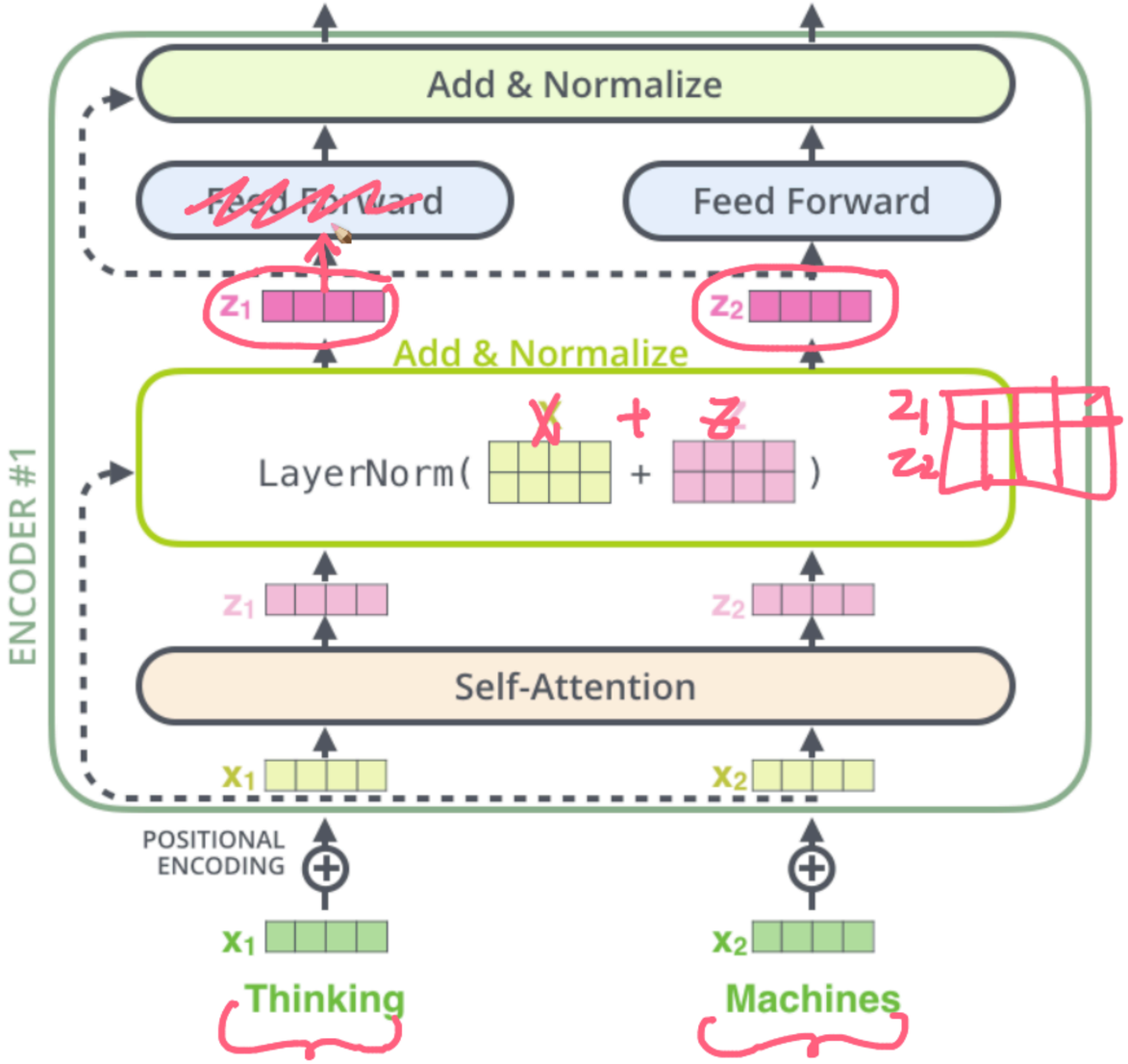


The Decoder Side

Now that we've covered most of the concepts on the encoder side, we basically know how the components of decoders work as well. B

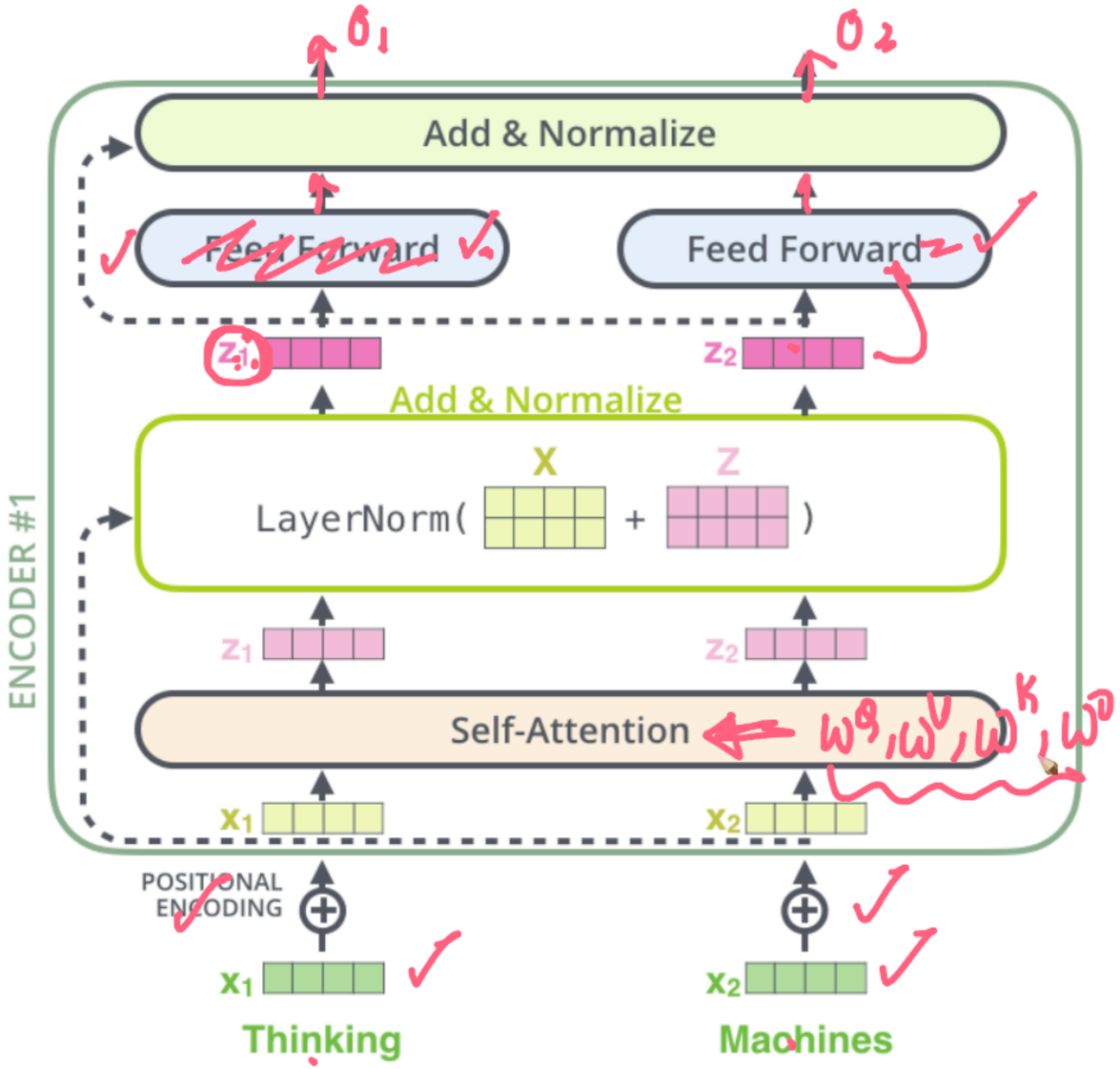
The encoder start by processing the input sequence. The output of the top encoder is then transformed into a set of

If we're to visualize the vectors and the layer-norm operation associated with self attention, it would look like this:



This goes for the sub-lay tacked encoders and decoders, it would look something like this:

If we're to visualize the vectors and the layer-norm operation associated with self attention, it would look like this:



This goes for the sub-lay tacked encoders and decoders, it would look something like this:

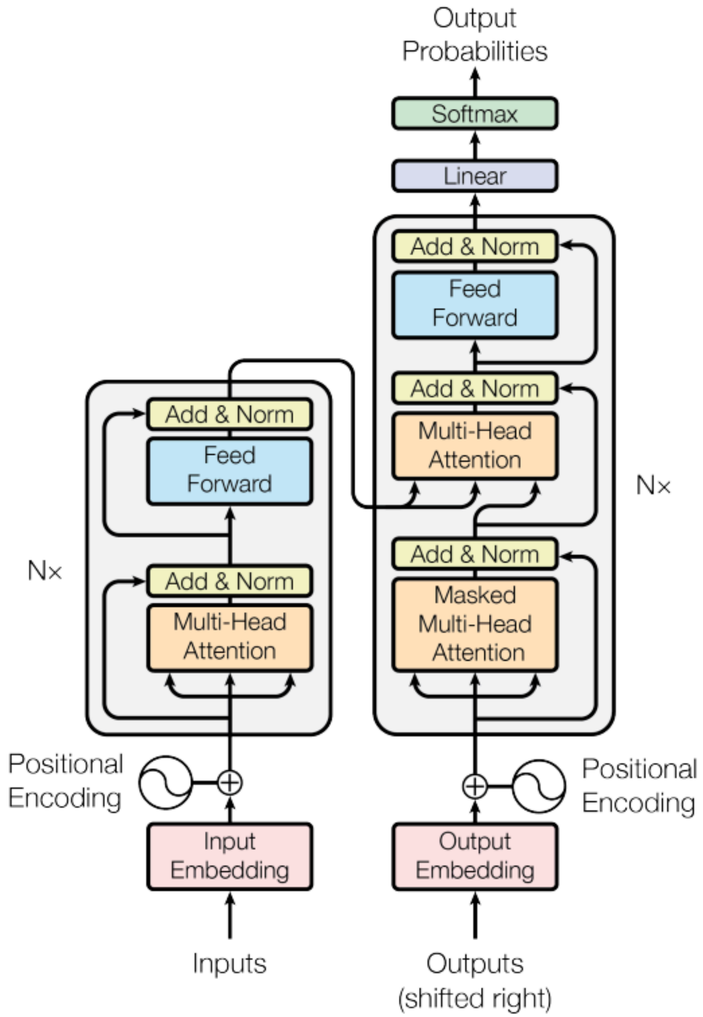


Figure 1: The Transformer - model architecture.

3.1 Encoder and Decoder Stacks