AI VIET NAM

@aivietnam.edu.vn

Sequence to sequence, Attention, Transformer

Nguyễn Quốc Thái

Applications

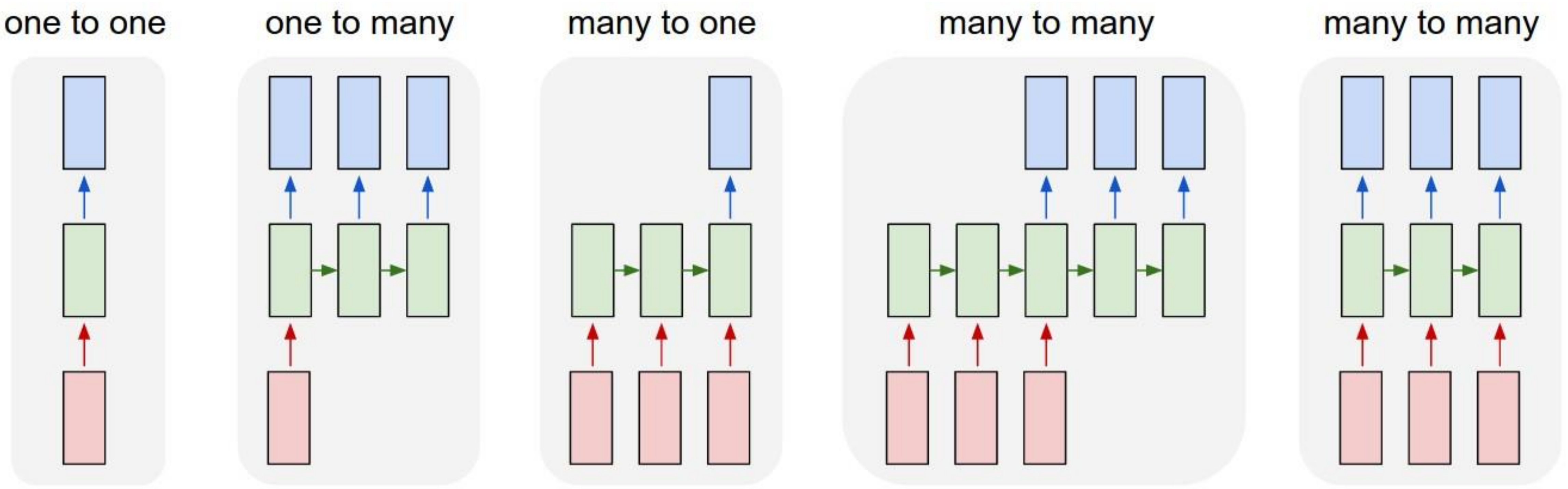


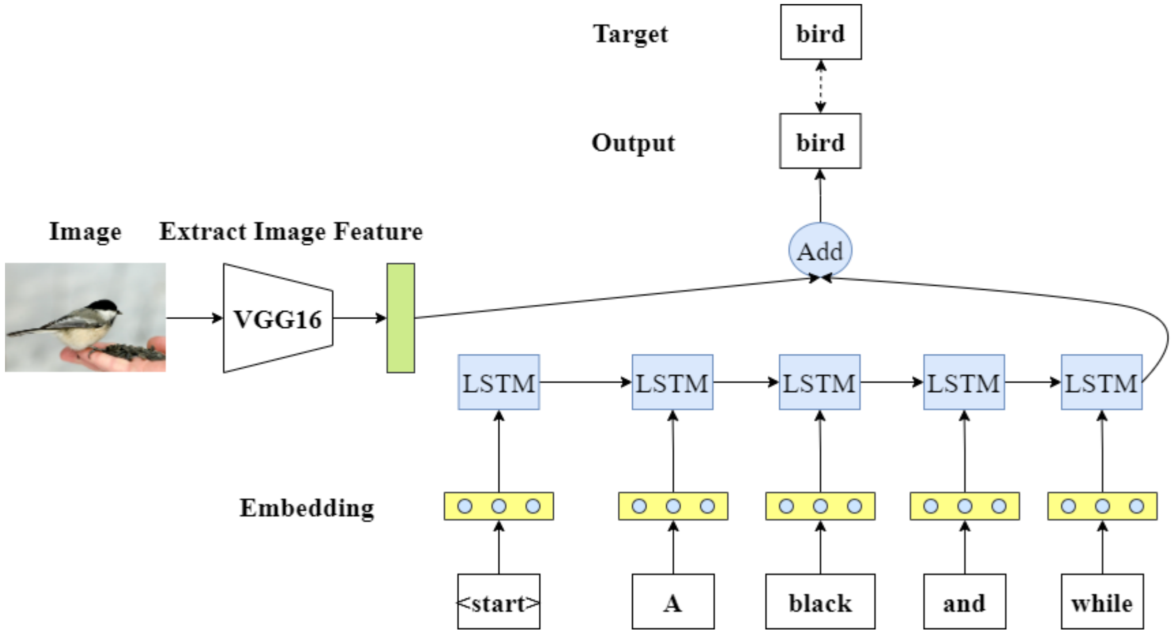
Image classification | Image Captioning | Text classification | Machine Translation Recognition | POS tagging, NER, Language Modeling

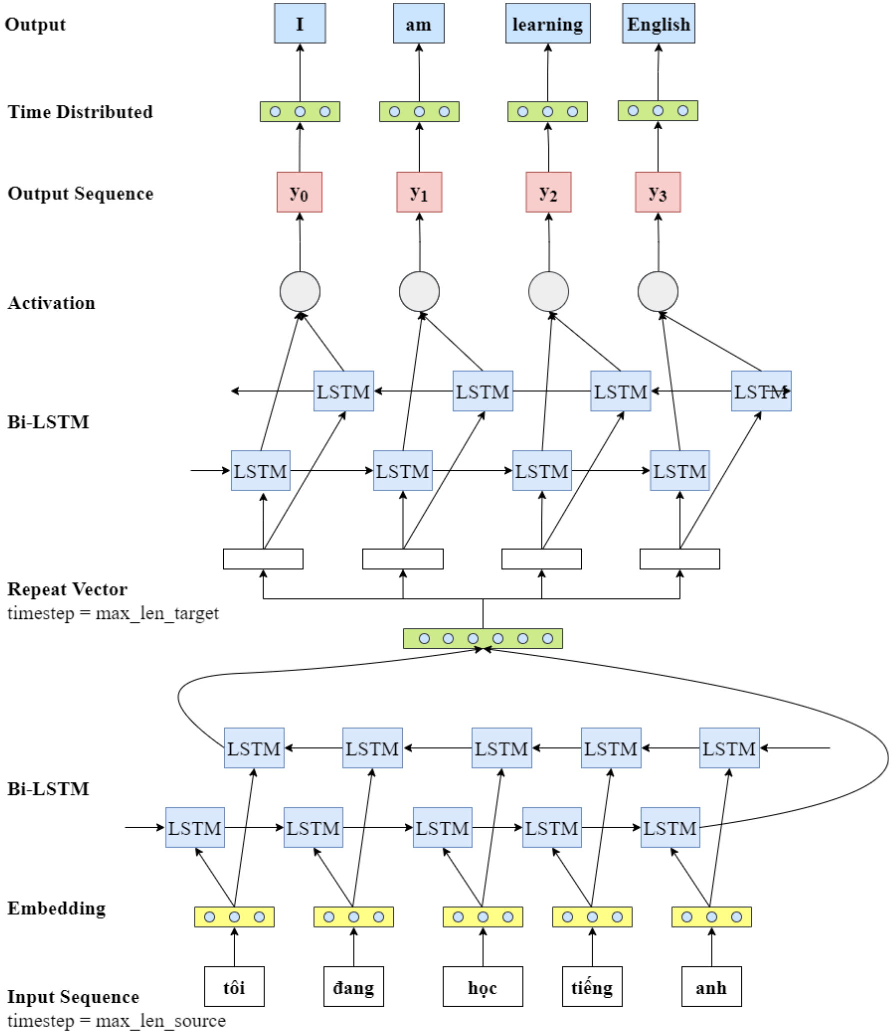
Source: [The Unreasonable Effectiveness of Recurrent Neural Networks]

Applications

Image Captioning
Machine Translation
Using RNNs

Problem?





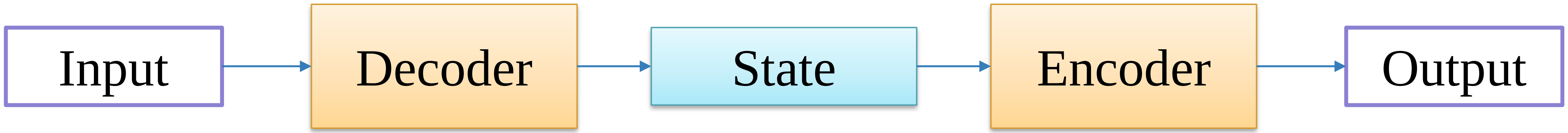
Machine Translation

Problem?

- Neural machine translation
- Some difficulties remain
  - Maintaining context over longer text
  - Out-of-vocabulary words
  - Low-resource language pair
  - Domain mismatch between train, validation and test data
  - …

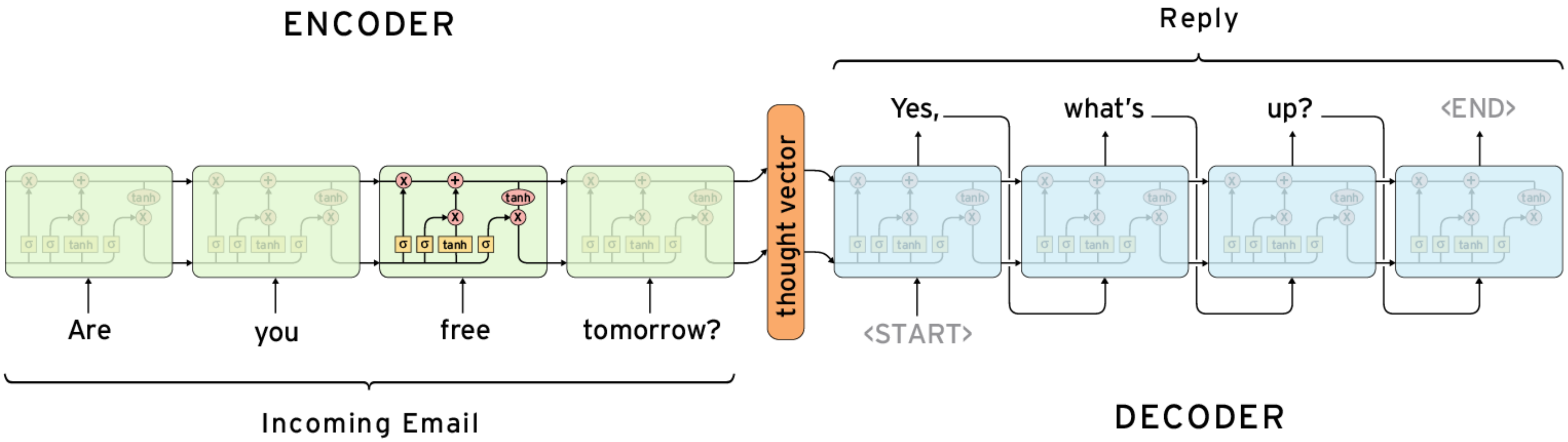
Encoder-Decoder model

- Take a sequence as input, predict a sequence as output
- Encoder: encoding the inputs into state
- Decoder: the state is passed into the decoder to generate the outputs



Encoder-Decoder model

- Input and output maybe the same or different length
- Applications: chatbot, text generation, question answering, machine translation,…

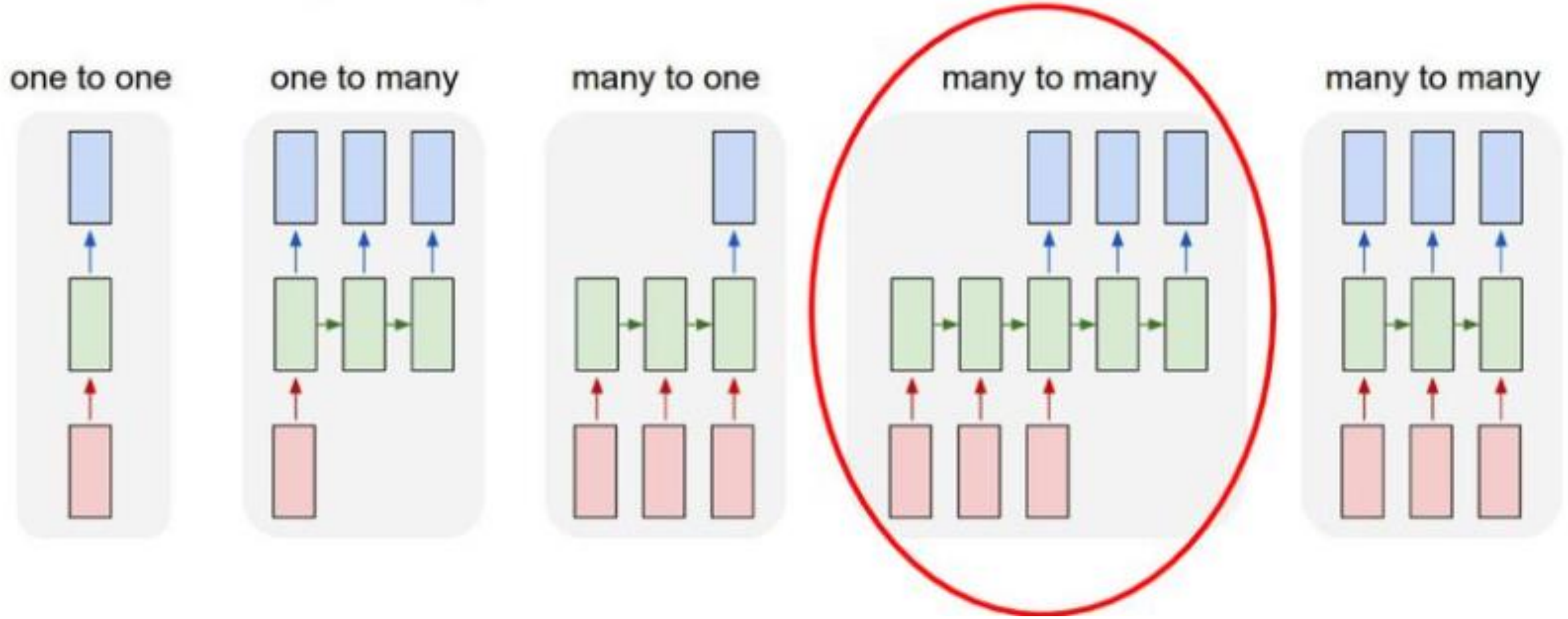


Ref

AI VIET NAM
@aivietnam.edu.vn

Sequence to sequence

- Based on encoder-decoder model
- Transform an input sequence (source) to a new sequence (target)
- Both sequences can be of arbitrary lengths



Ref

Machine Translation

- Prepare dataset

“tôi yêu bạn” => “i love you”

“tôi đang học tiếng anh” => “i am learning english”

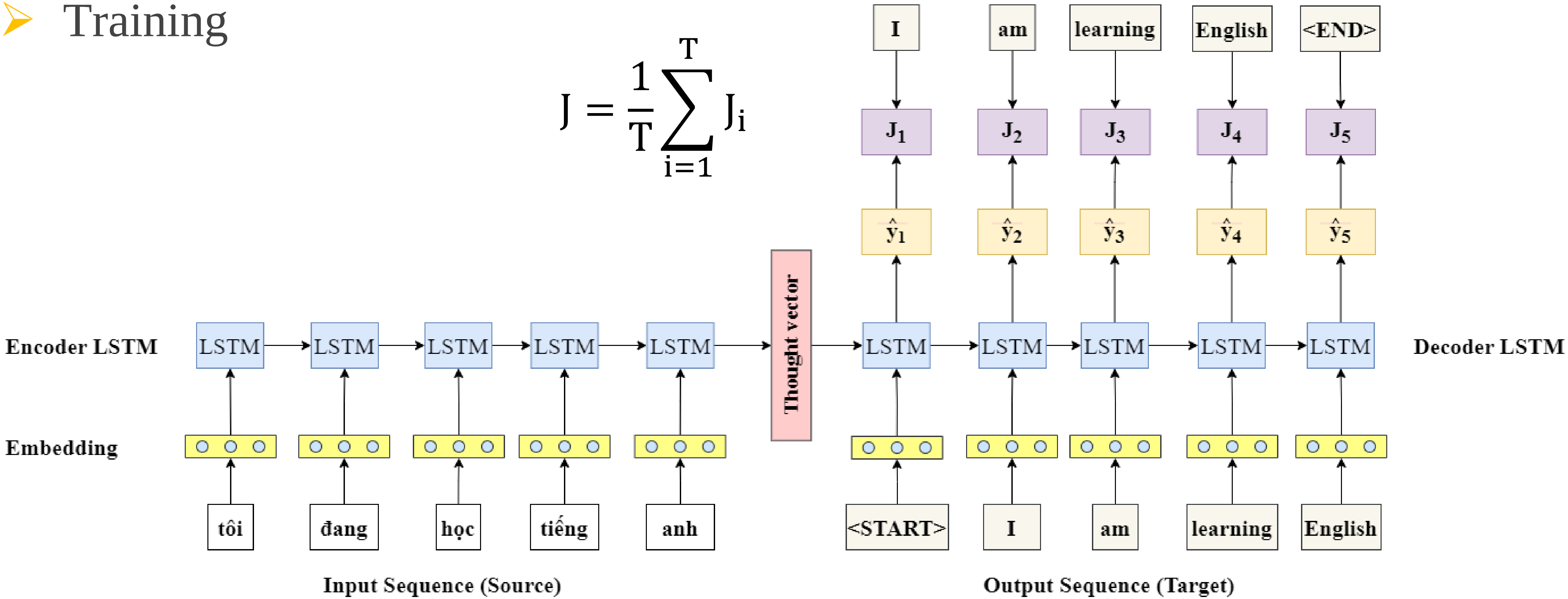
“buổi tối an lành” => “good evening”

Input encoder	Input decoder	Target decoder
tôi yêu bạn	<start> i love you	i love you <end>
tôi đang học tiếng anh	<start> i am learning english	i am learning english <end>
buổi tối an lành	<start> good evening	good evening <end>

Neural Machine Translation

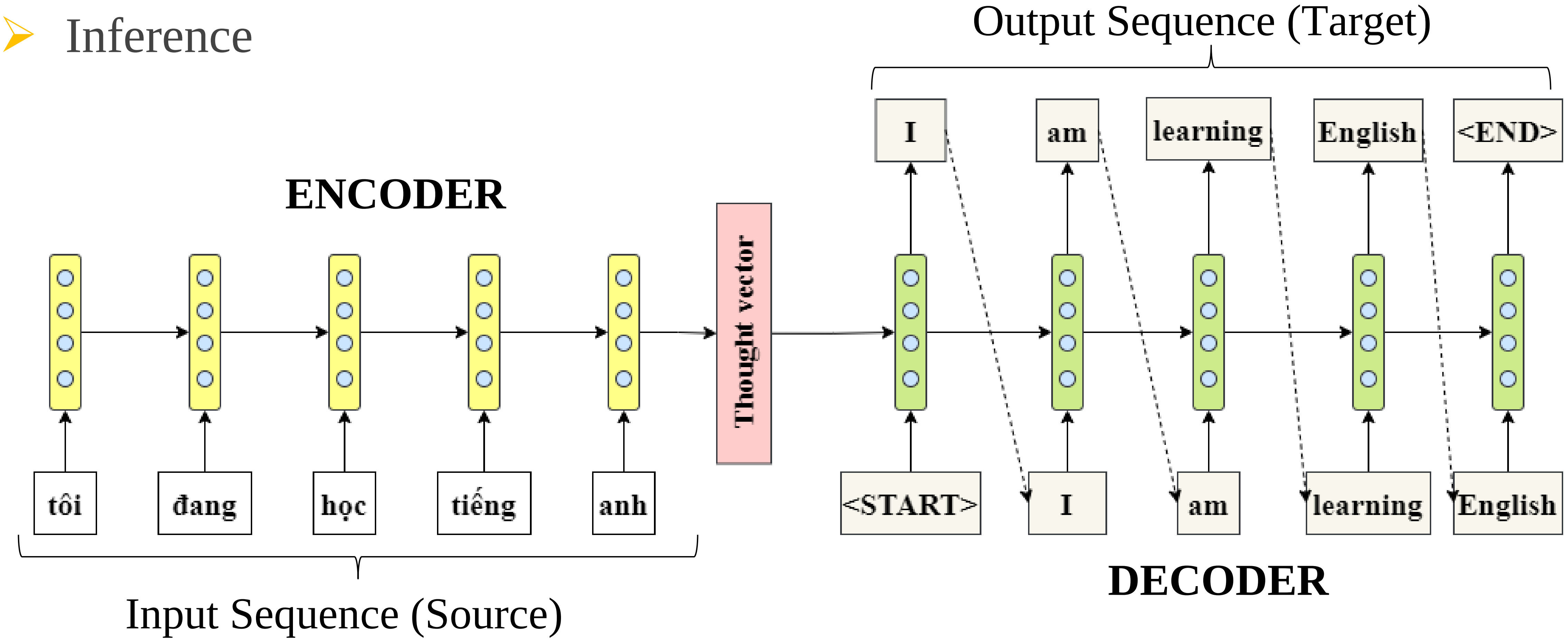
- Neural Machine Translation using Seq2Seq
- Training

$$J = \frac{1}{T}\sum_{i=1}^{T} J_i$$



Neural Machine Translation

- Neural Machine Translation using Seq2Seq
- Inference



Neural Machine Translation

➤ Neural Machine Translation using Seq2Seq

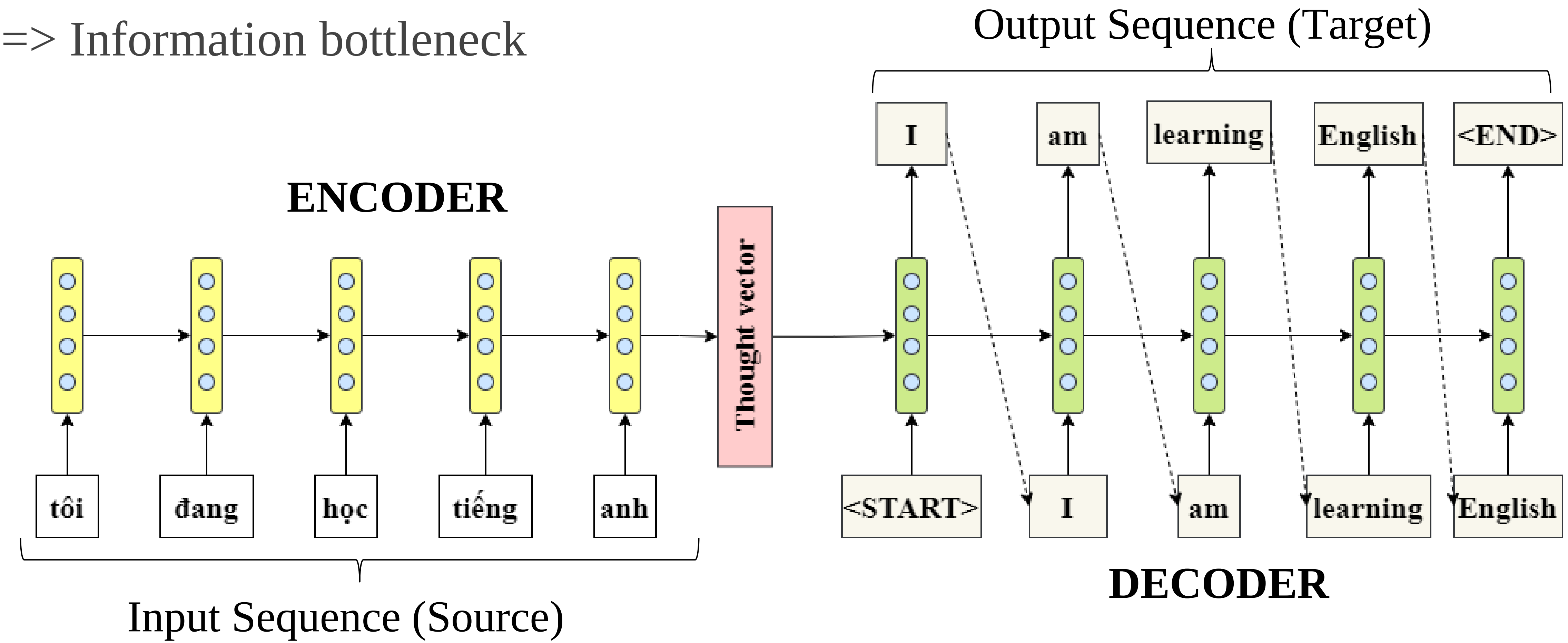
```
encoder_inputs = Input(shape=(vi_max_len,))
encoder_embedding = Embedding(vi_vocab_size,
                              embedding_dim,
                              mask_zero=True)(encoder_inputs)
encoder_outputs, state_h, state_c = LSTM(hidden_size,
                                         return_state=True)(encoder_embedding)
encoder_states = [state_h, state_c]

decoder_inputs = Input(shape=(en_max_len,))
decoder_embedding = Embedding(en_vocab_size,
                              embedding_dim,
                              mask_zero=True)(decoder_inputs)
decoder_lstm = LSTM(hidden_size,
                    return_state=True,
                    return_sequences=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding,
                                     initial_state=encoder_states)

decoder_dense = Dense(en_vocab_size, activation="softmax")
output = decoder_dense(decoder_outputs)

seq2seq_model = Model([encoder_inputs, decoder_inputs], output )

seq2seq_model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 100)]        0           []

 input_2 (InputLayer)           [(None, 80)]         0           []

 embedding (Embedding)          (None, 100, 200)     3211000     ['input_1[0][0]']

 embedding_1 (Embedding)        (None, 80, 200)      7251200     ['input_2[0][0]']

 lstm (LSTM)                    [(None, 256),        467968      ['embedding[0][0]']
                                 (None, 256),
                                 (None, 256)]

 lstm_1 (LSTM)                  [(None, 80, 256),    467968      ['embedding_1[0][0]',
                                 (None, 256),                     'lstm[0][1]',
                                 (None, 256)]                     'lstm[0][2]']

 dense (Dense)                  (None, 80, 36256)    9317792     ['lstm_1[0][0]']

==================================================================================================
Total params: 20,715,928
Trainable params: 20,715,928
Non-trainable params: 0
__________________________________________________________________________________________________
```

The bottleneck problem

Thought vector: capture all information of input sentence

=> Information bottleneck



The bottleneck problem

Thought vector: capture all information of input sentence

$\Rightarrow$ Information bottleneck

Solutions: each step of the decoder, directly connected to the components of the encoder

Focus on all timestep in the encoder

AI VIET NAM
@aivietnam.edu.vn

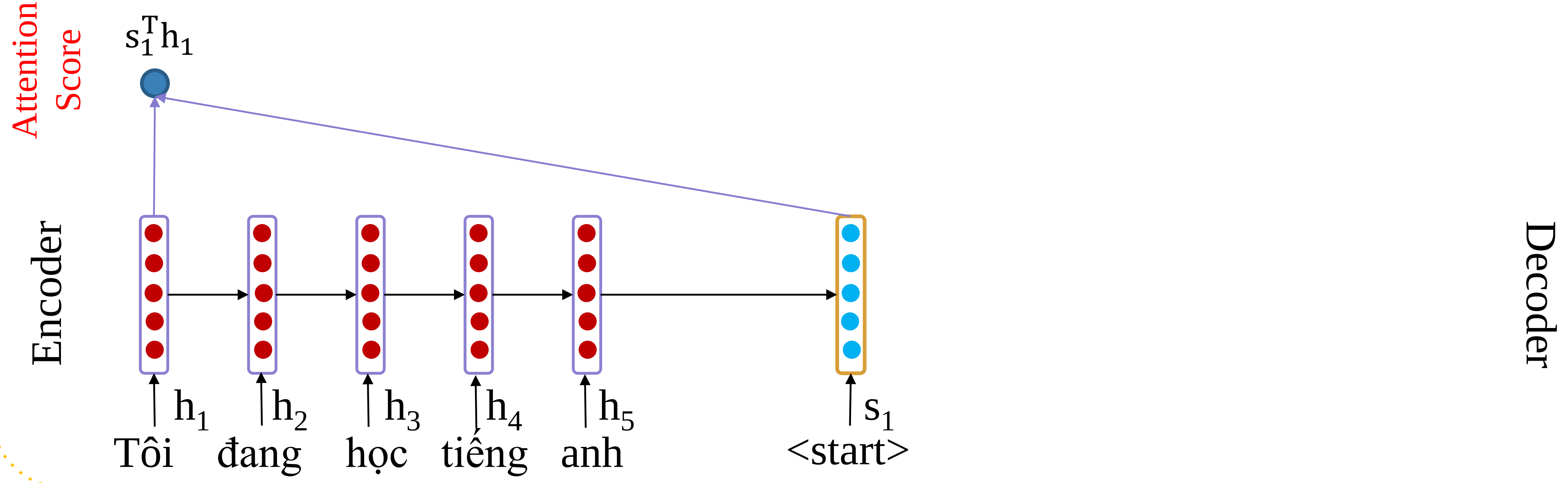
Attention

Nguyễn Quốc Thái

AI VIET NAM
@aivietnam.edu.vn

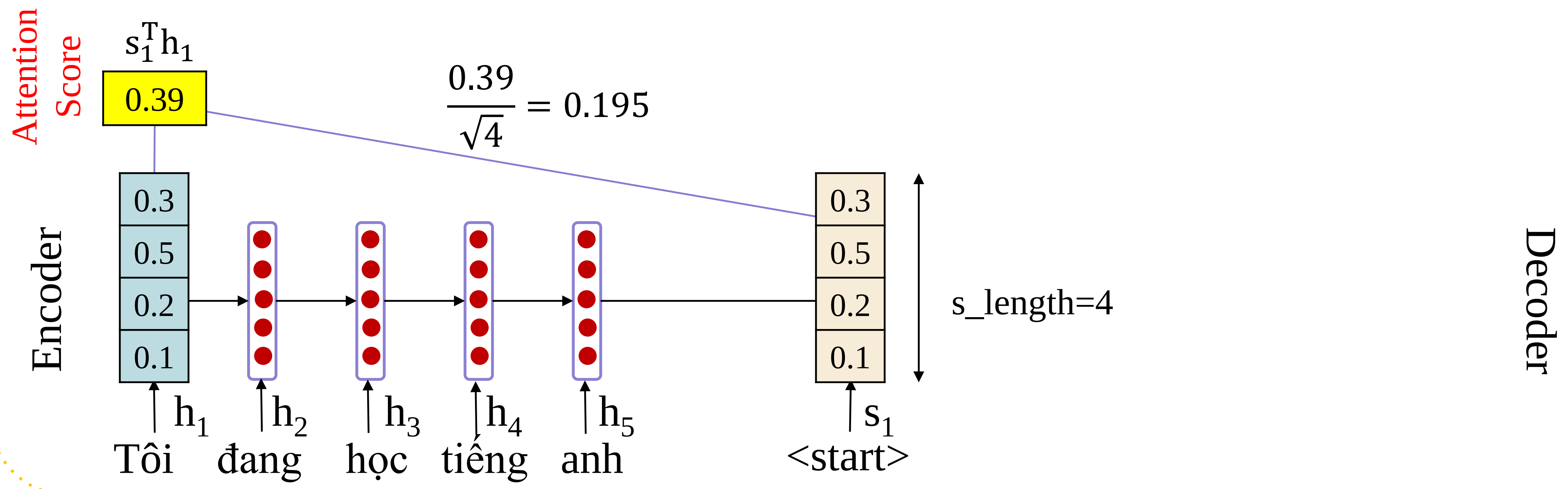
Attention

Note: With Dot-Product Attention, attention score is normalized to length of s (key), called Scaled dot product attention

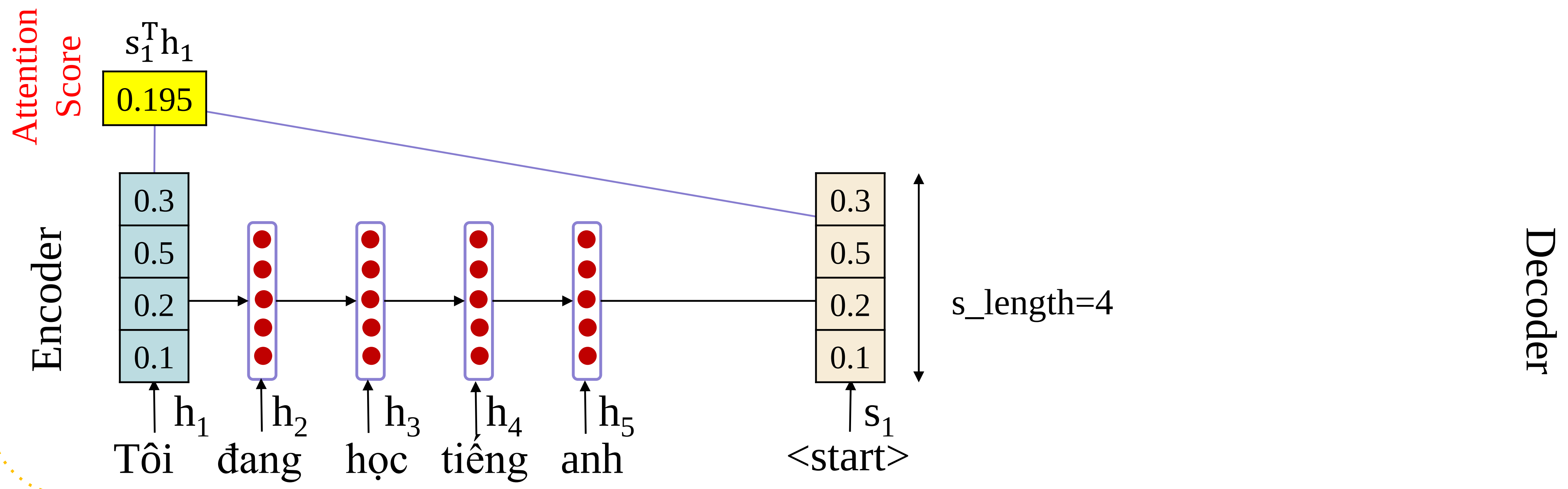


Attention

Note: With Dot-Product Attention, attention score is normalized to length of s (key), called Scaled dot product attention

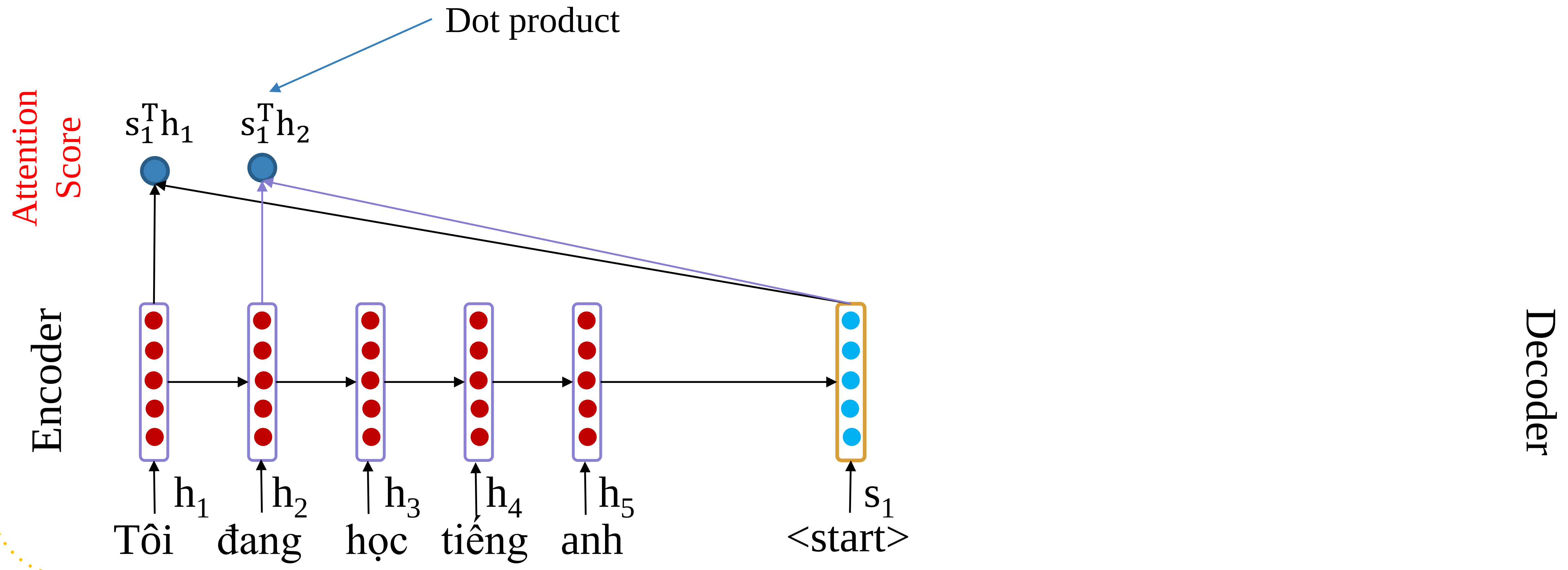


Attention

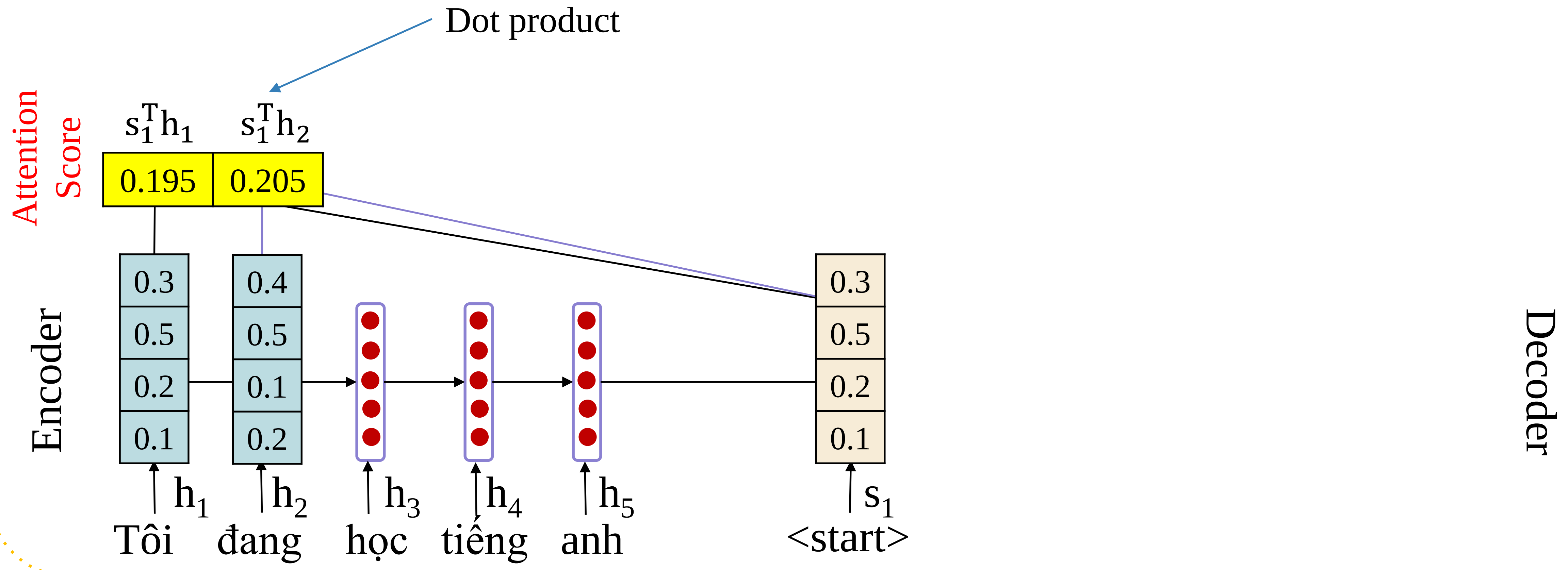


Attention Score
$s_1^T h_1$
0.195
Encoder
0.3
0.5
0.2
0.1
h_1
h_2
h_3
h_4
h_5
Tôi
đang
học
tiếng
anh
0.3
0.5
0.2
0.1
s_1
<start>
s_length=4
Decoder

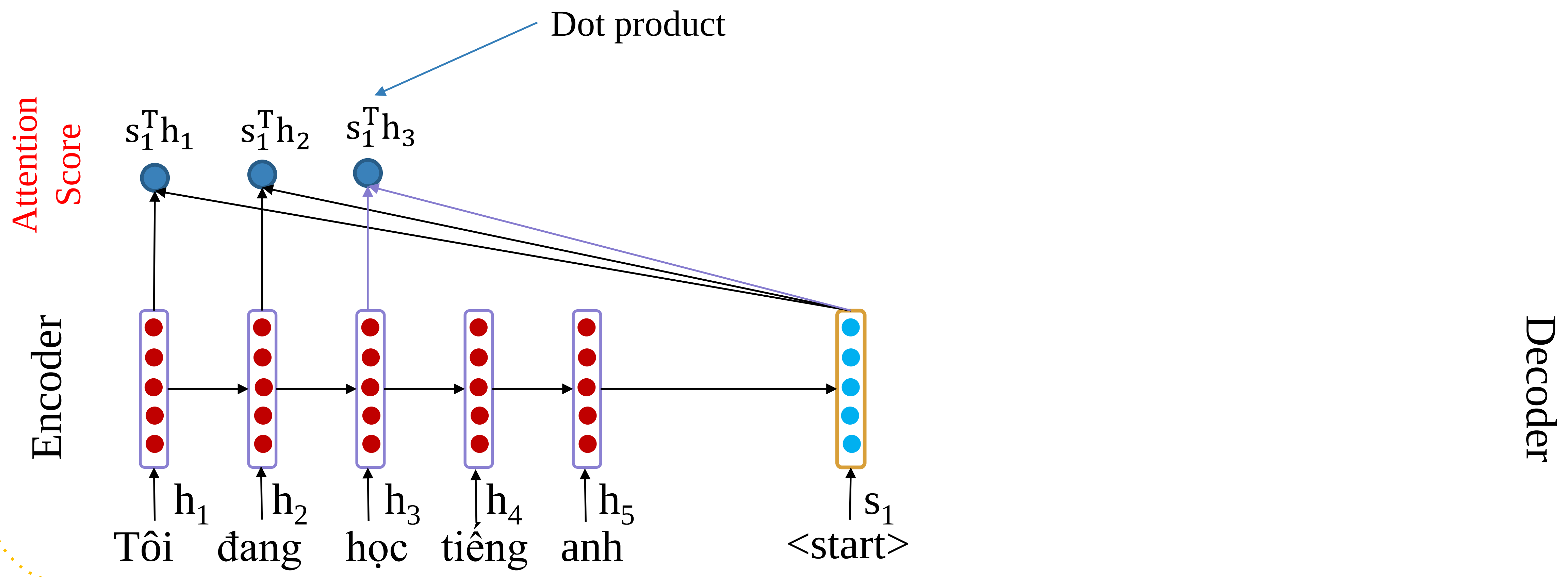
Attention


Dot product
Attention Score
$s_1^T h_1$
$s_1^T h_2$
Encoder
Decoder
h_1
h_2
h_3
h_4
h_5
s_1
Tôi
đang
học
tiếng
anh
<start>

Attention

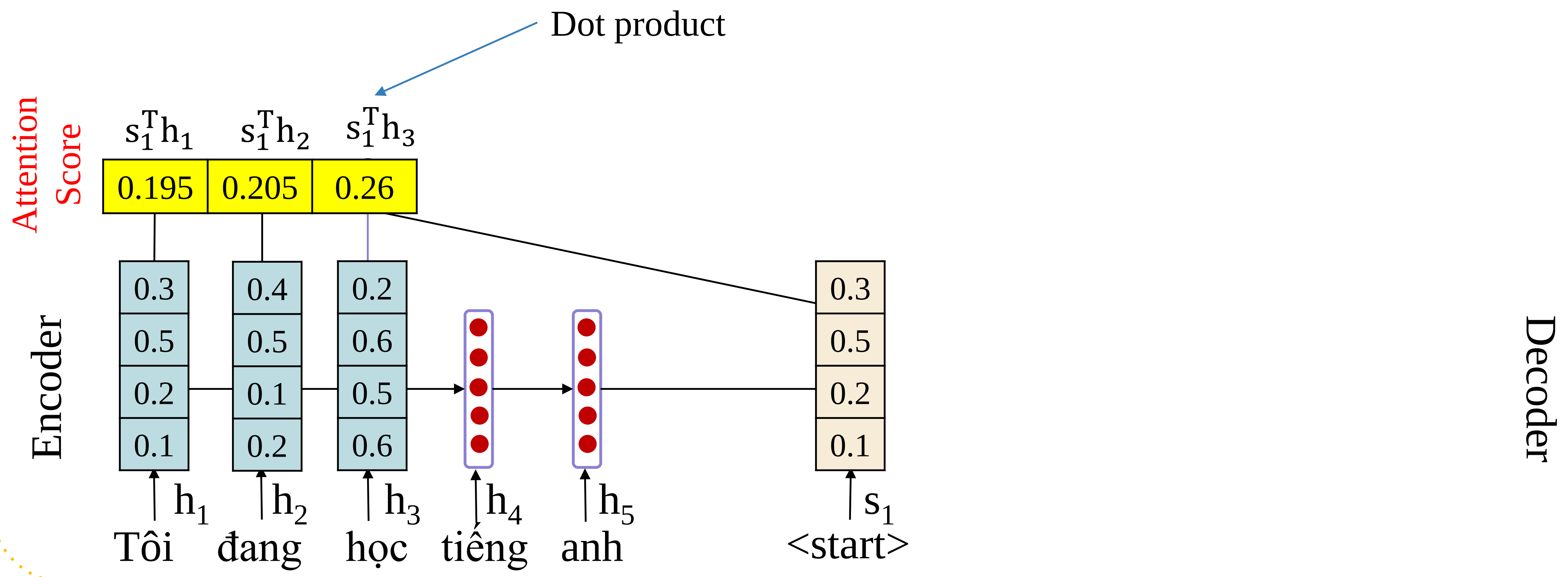

Dot product
Attention Score
$s_1^T h_1$
$s_1^T h_2$
0.195
0.205
Encoder
0.3
0.5
0.2
0.1
0.4
0.5
0.1
0.2
0.3
0.5
0.2
0.1
Decoder
h_1
h_2
h_3
h_4
h_5
s_1
Tôi
đang
học
tiếng
anh
<start>

Attention

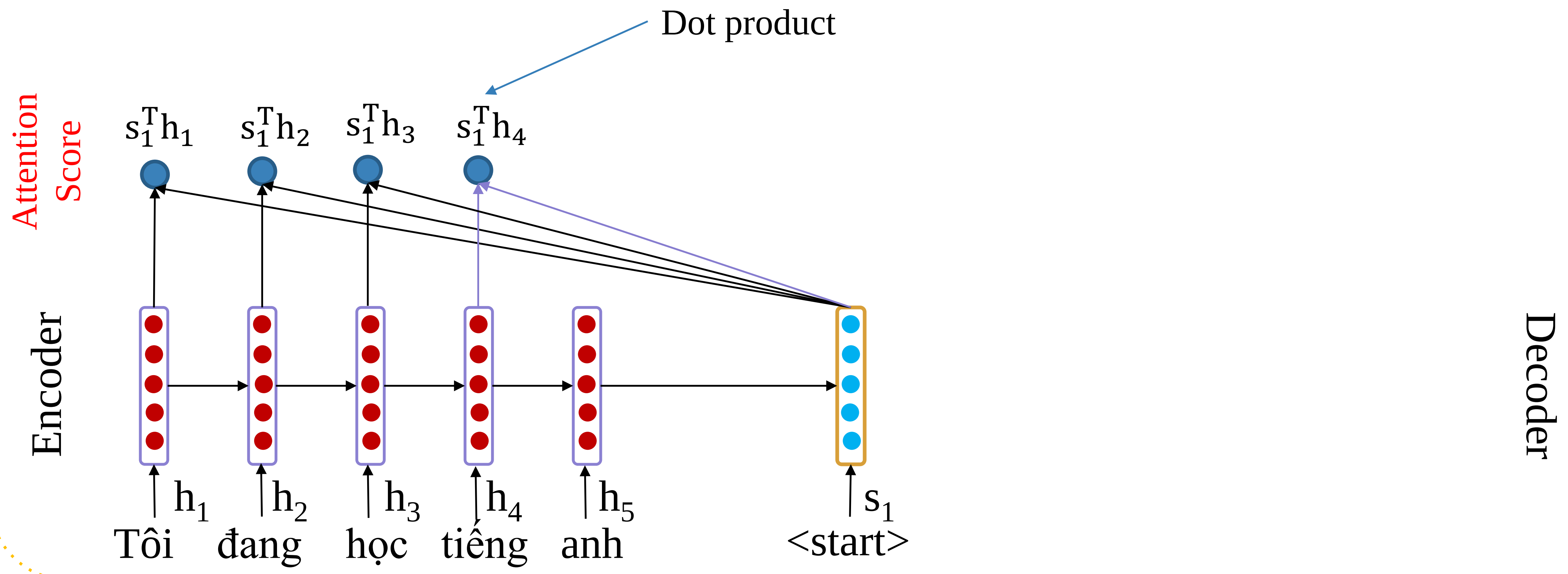

Dot product
$s_1^T h_1$
$s_1^T h_2$
$s_1^T h_3$
Attention Score
Encoder
Decoder
h_1
h_2
h_3
h_4
h_5
s_1
Tôi
đang
học
tiếng
anh
<start>

AI VIET NAM
@aivietnam.edu.vn

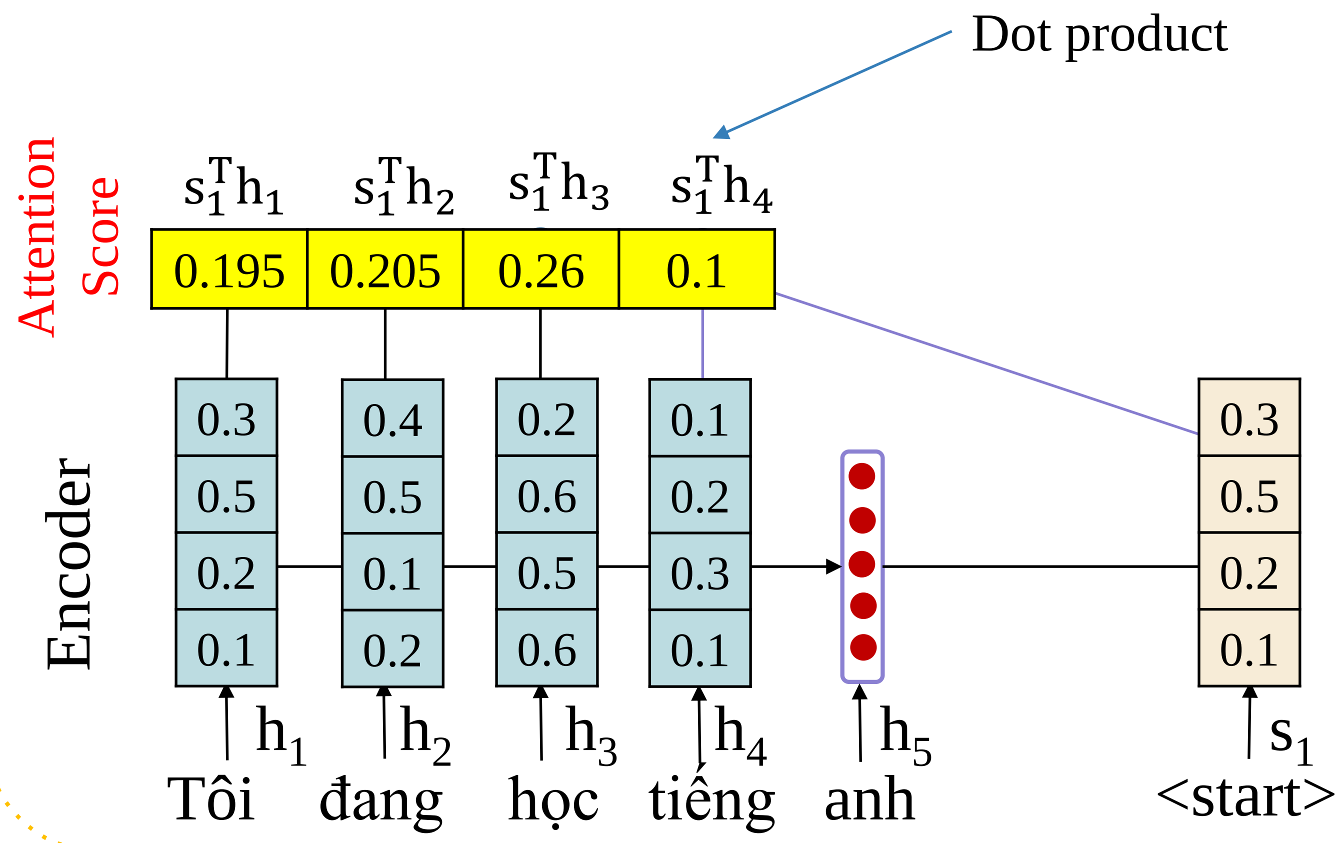
Attention



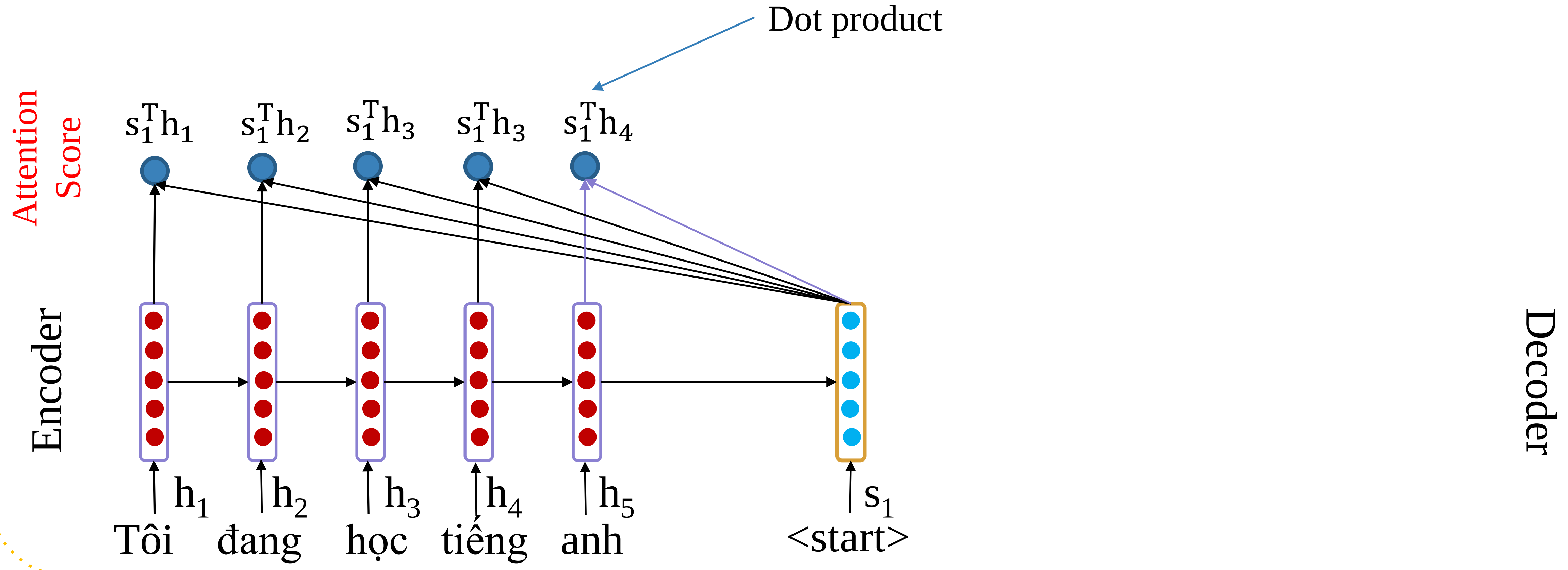
Attention


Dot product
Attention Score
$s_1^T h_1$
$s_1^T h_2$
$s_1^T h_3$
$s_1^T h_4$
Encoder
Decoder
h_1
h_2
h_3
h_4
h_5
s_1
Tôi
đang
học
tiếng
anh
<start>

Attention


Dot product
Attention Score
$s_1^T h_1$
$s_1^T h_2$
$s_1^T h_3$
$s_1^T h_4$
0.195
0.205
0.26
0.1
Encoder
0.3
0.5
0.2
0.1
0.4
0.5
0.1
0.2
0.2
0.6
0.5
0.6
0.1
0.2
0.3
0.1
0.3
0.5
0.2
0.1
Decoder
h_1
h_2
h_3
h_4
h_5
s_1
Tôi
đang
học
tiếng
anh
<start>

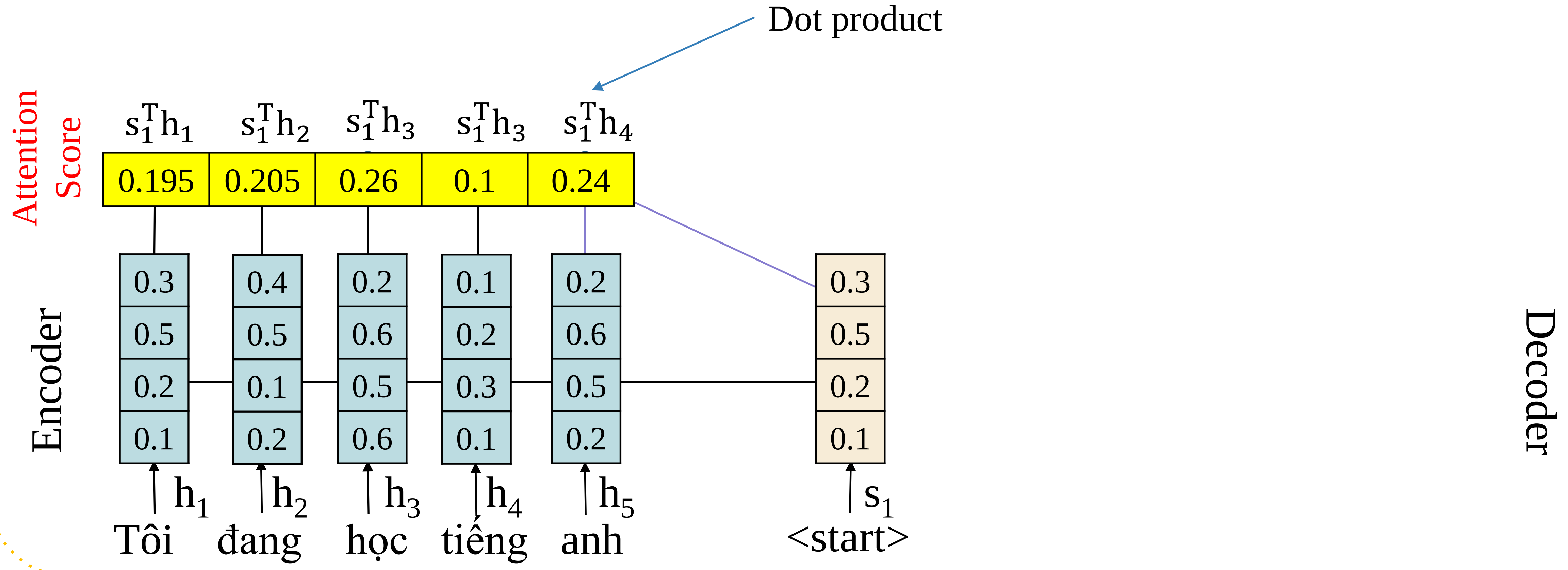
Attention



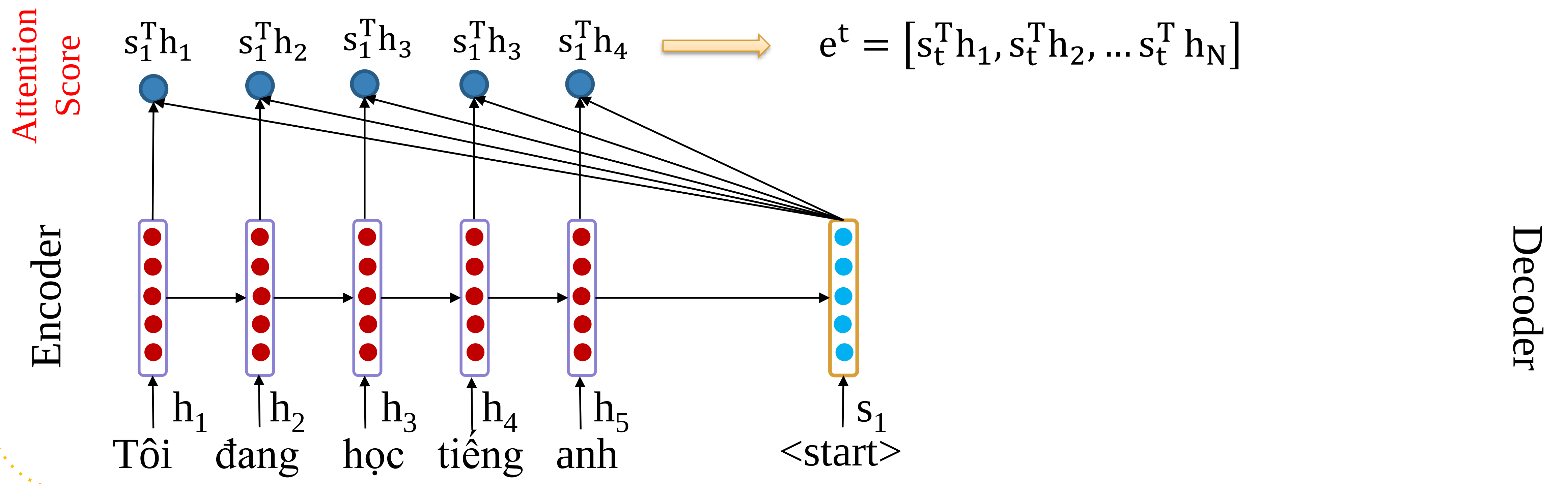
Dot product
Attention Score
$s_1^T h_1$
$s_1^T h_2$
$s_1^T h_3$
$s_1^T h_3$
$s_1^T h_4$
Encoder
Decoder
h_1
h_2
h_3
h_4
h_5
s_1
Tôi
đang
học
tiếng
anh
<start>

AI VIET NAM
@aivietnam.edu.vn

Attention

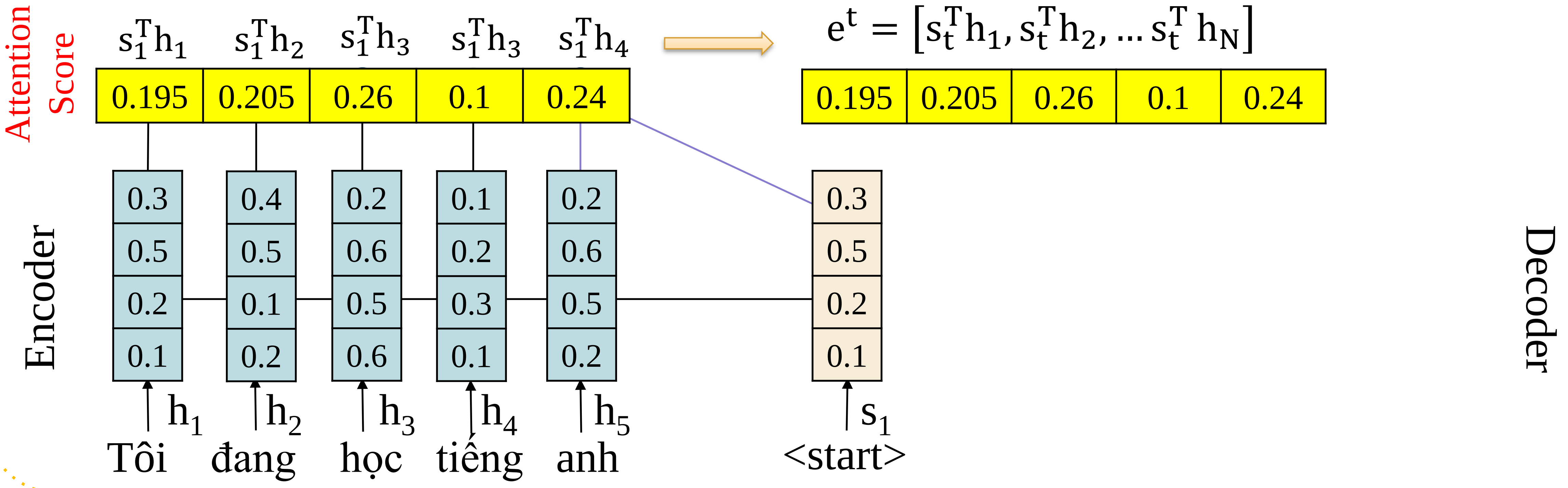


Attention



$$e^t = [s_t^T h_1, s_t^T h_2, \ldots s_t^T h_N]$$

Attention

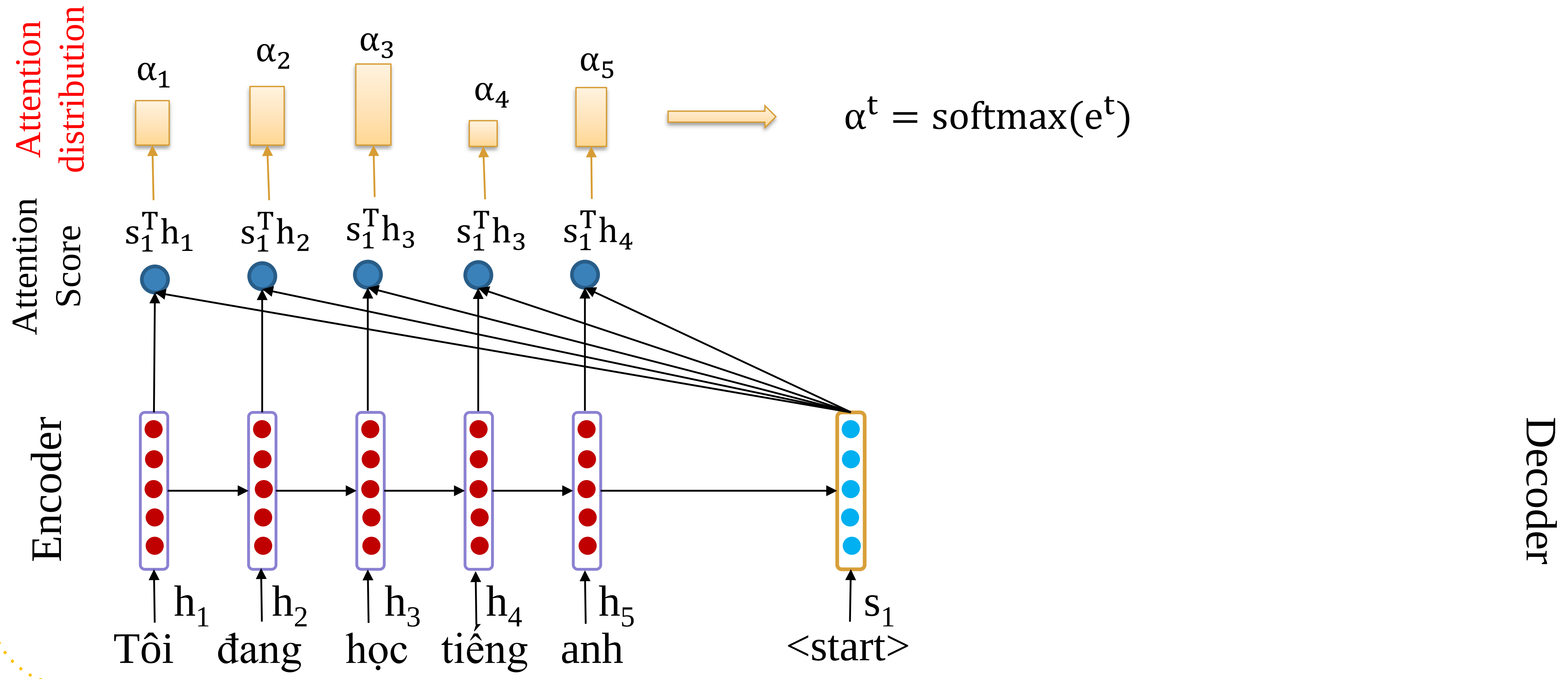


$$e^t = [s_t^T h_1, s_t^T h_2, \dots s_t^T h_N]$$

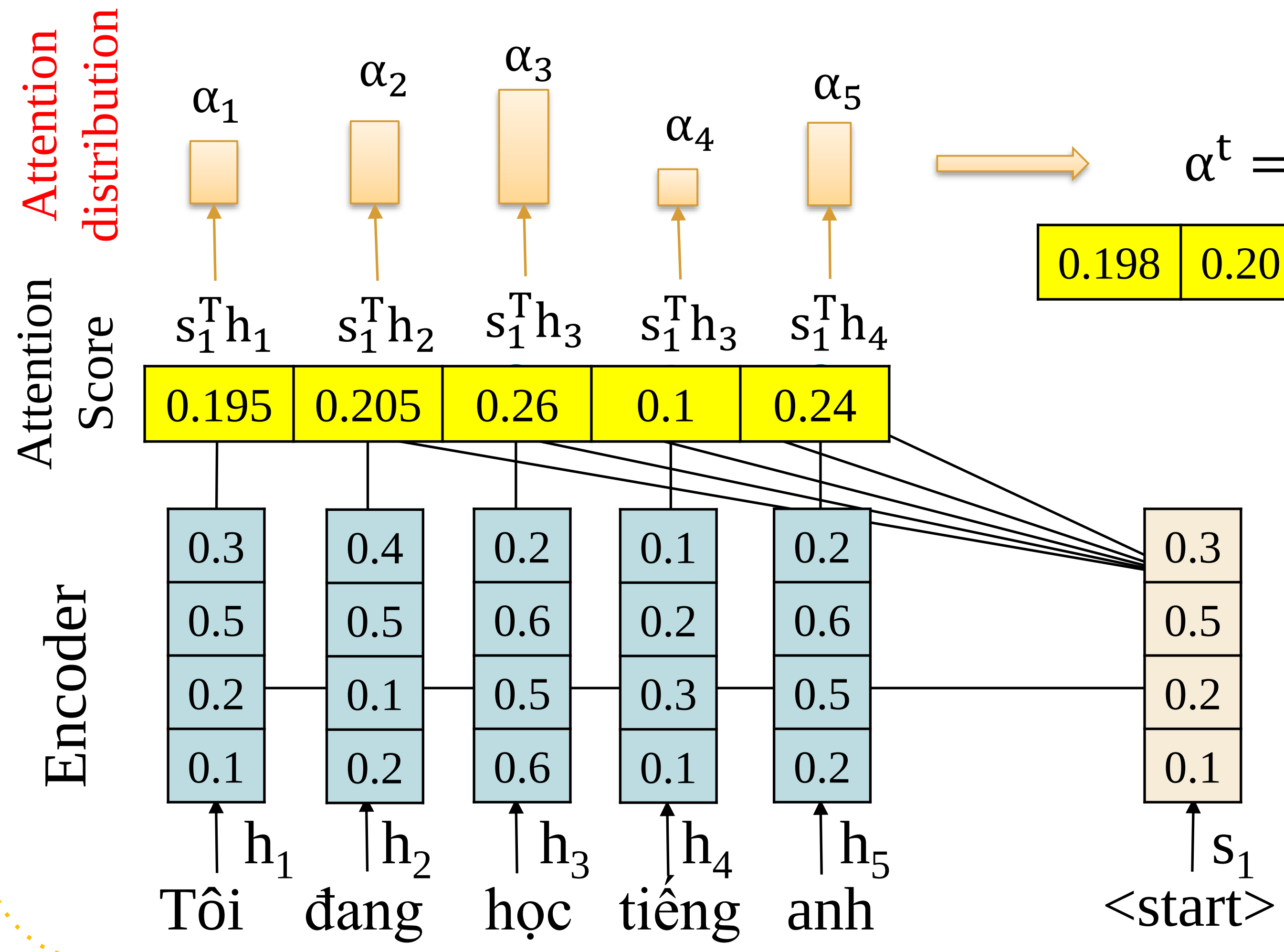
0.195	0.205	0.26	0.1	0.24

AI VIET NAM
@aivietnam.edu.vn

Attention



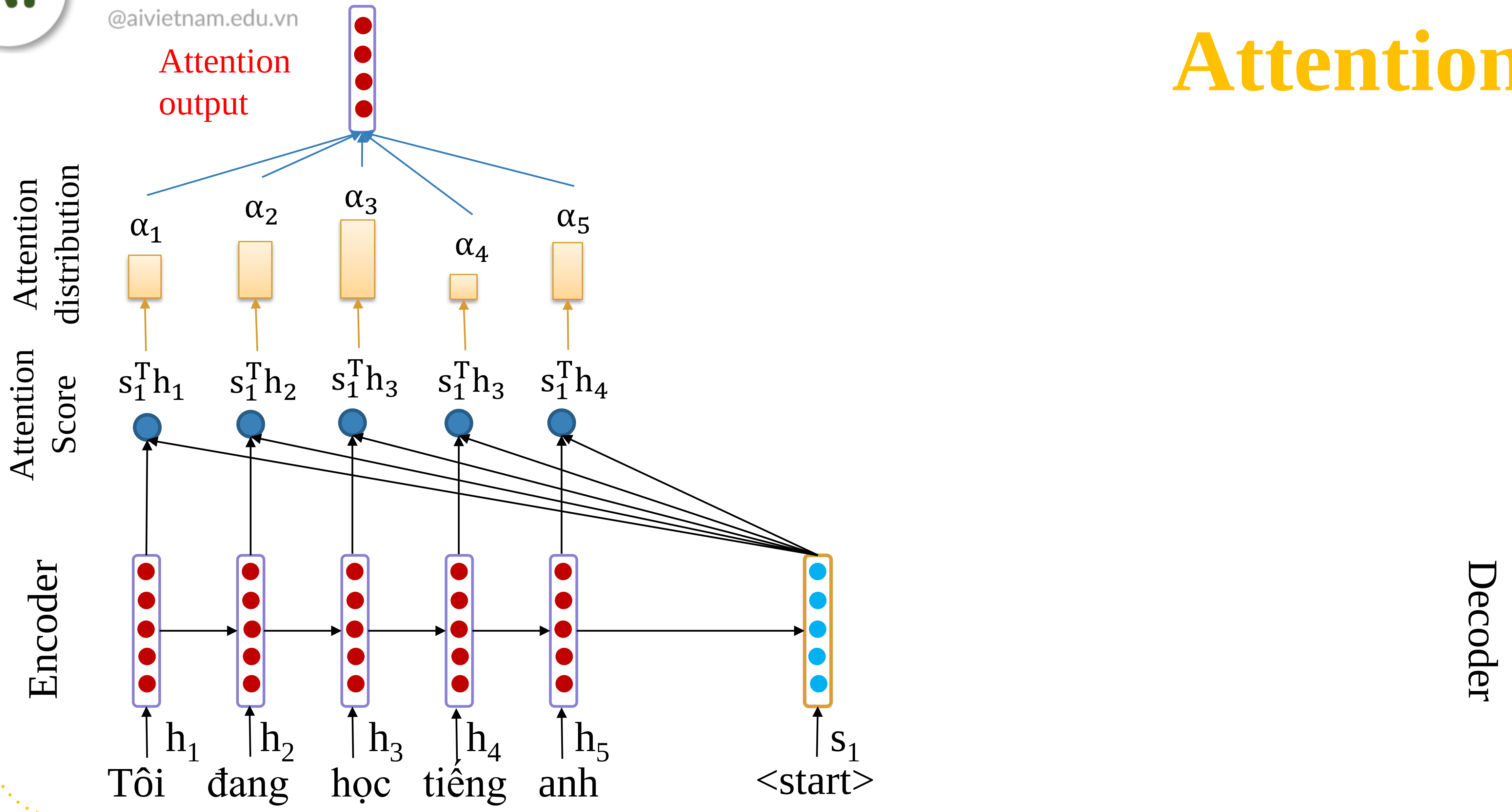
Attention



$$\alpha^t = \text{softmax}(e^t)$$

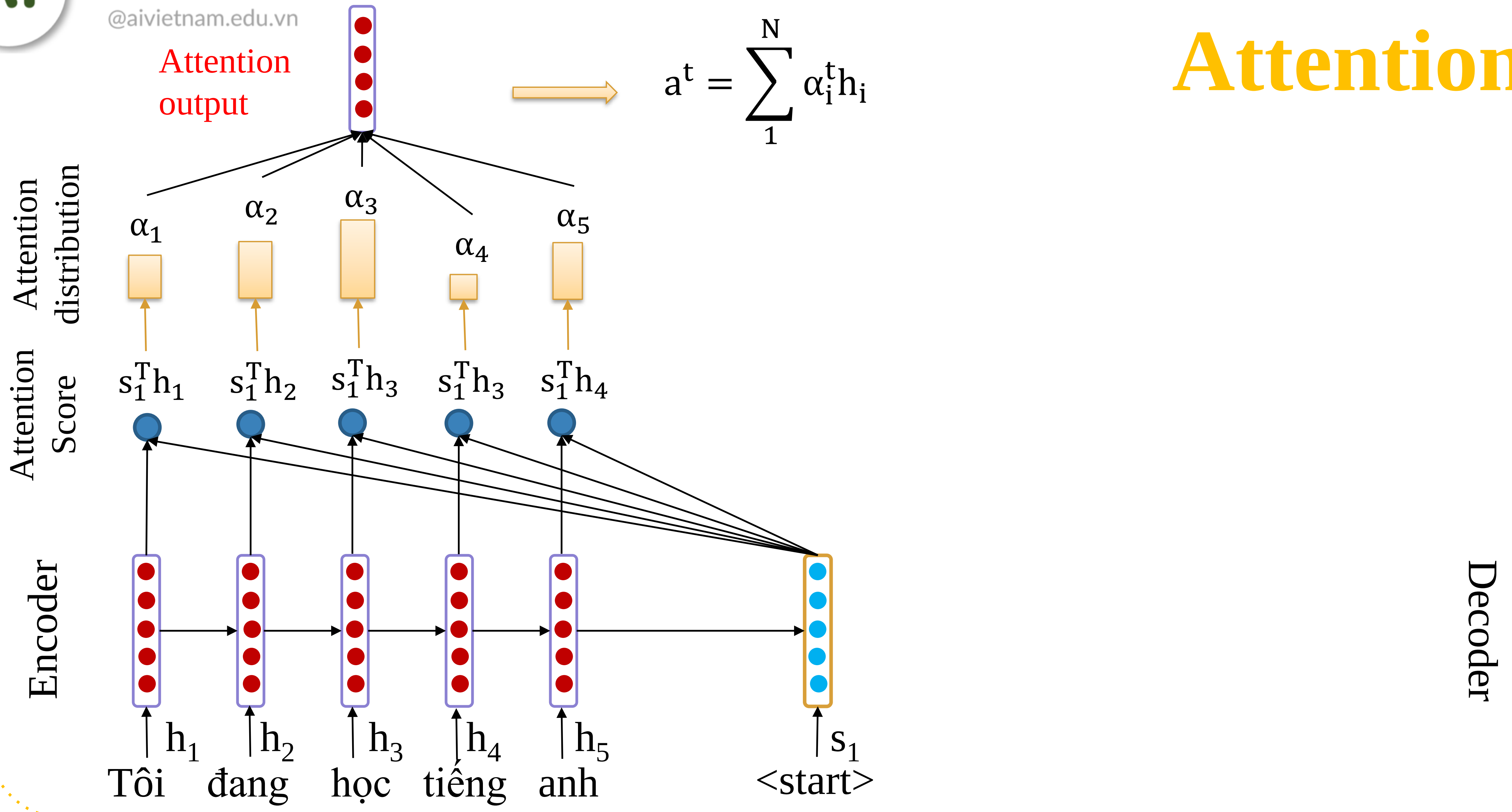
0.198	0.201	0.212	0.181	0.208

Attention

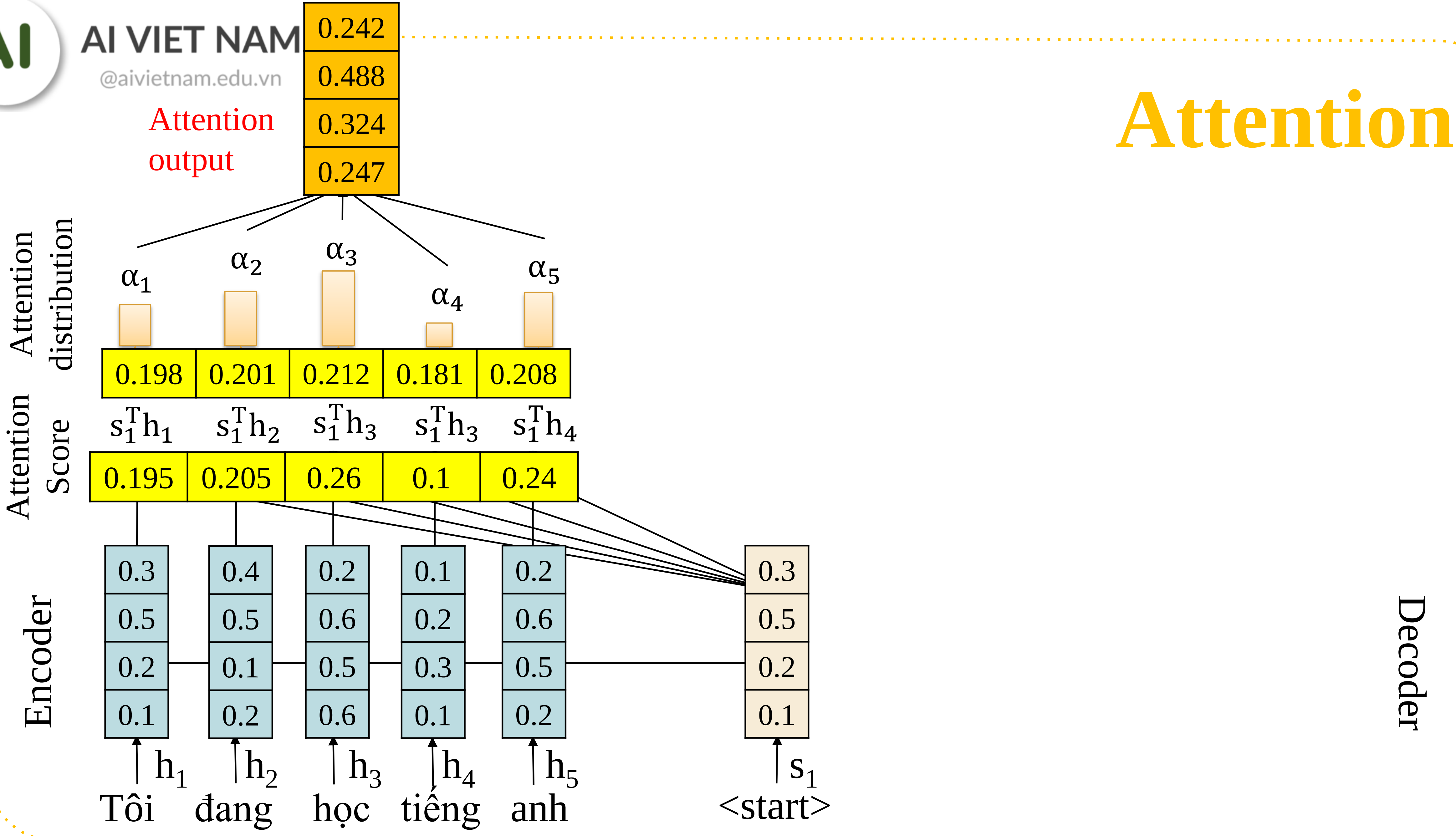

Attention
output
Attention
distribution
α_1
α_2
α_3
α_4
α_5
Attention
Score
$s_1^T h_1$
$s_1^T h_2$
$s_1^T h_3$
$s_1^T h_3$
$s_1^T h_4$
Encoder
Decoder
h_1
h_2
h_3
h_4
h_5
s_1
Tôi
đang
học
tiếng
anh
<start>

Attention

$$a^t = \sum_{1}^{N} \alpha_i^t h_i$$

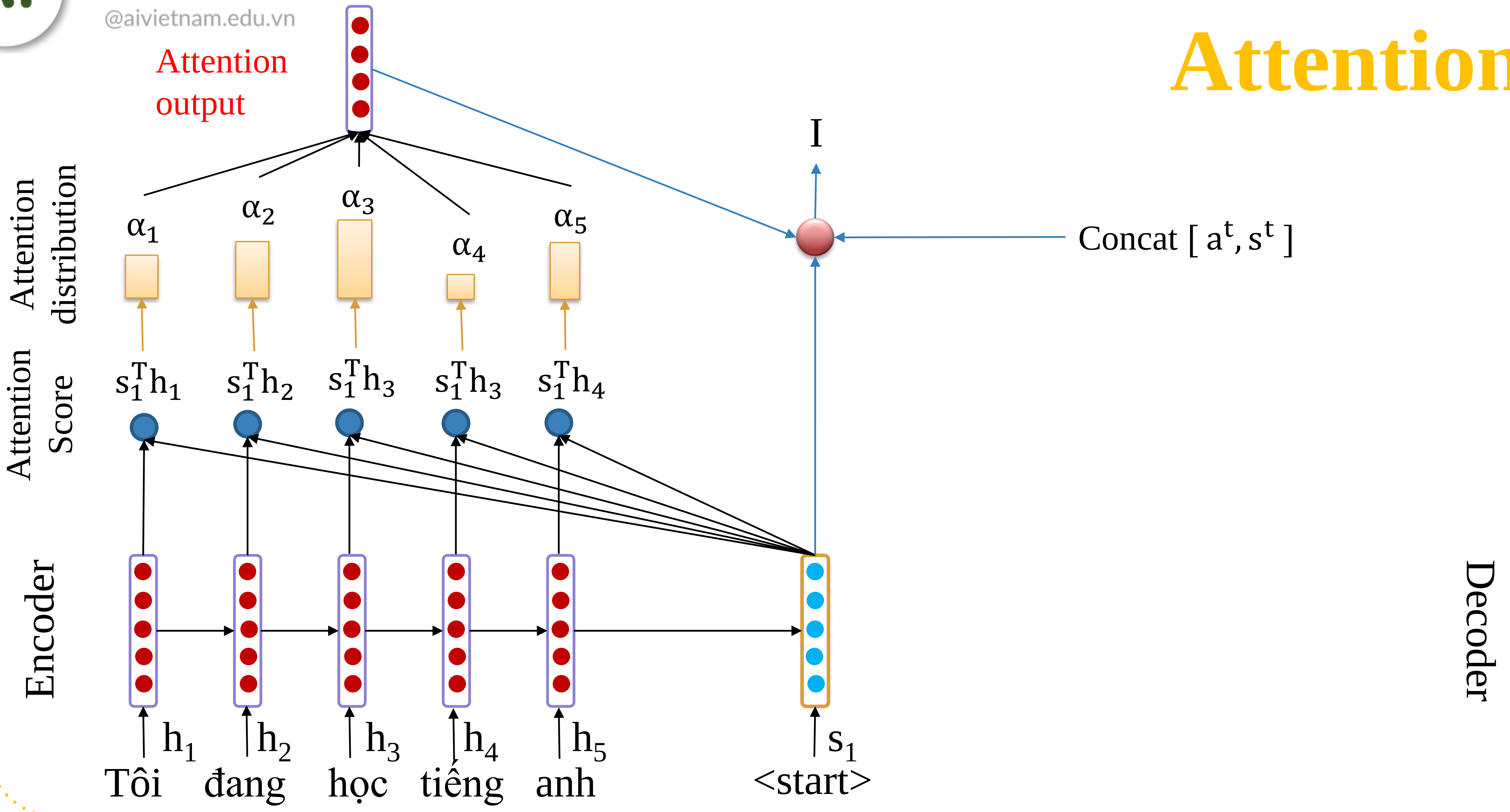


Attention

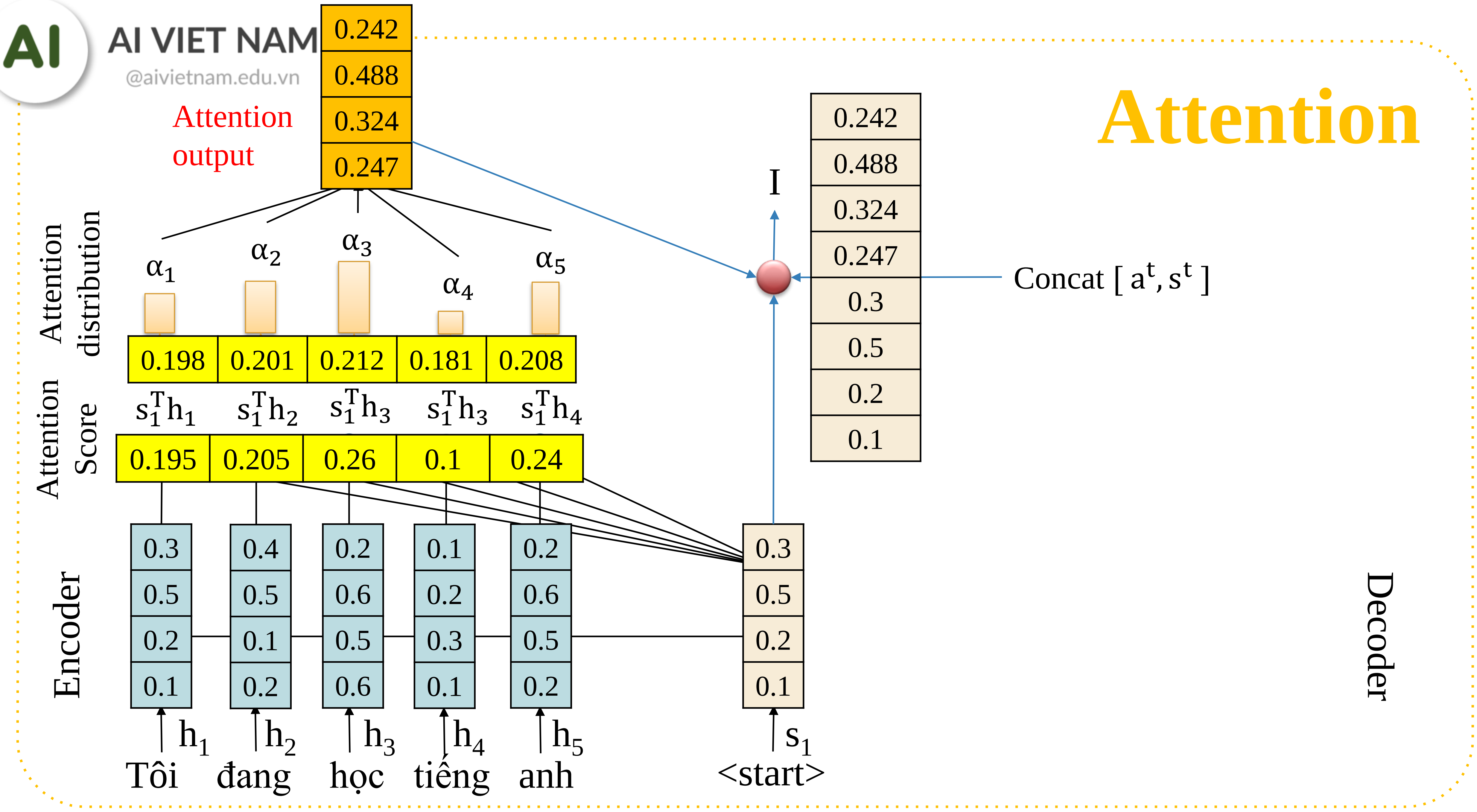


Attention output
0.242
0.488
0.324
0.247
Attention distribution
α_1
α_2
α_3
α_4
α_5
0.198
0.201
0.212
0.181
0.208
$s_1^T h_1$
$s_1^T h_2$
$s_1^T h_3$
$s_1^T h_3$
$s_1^T h_4$
Attention Score
0.195
0.205
0.26
0.1
0.24
Encoder
0.3 0.5 0.2 0.1
0.4 0.5 0.1 0.2
0.2 0.6 0.5 0.6
0.1 0.2 0.3 0.1
0.2 0.6 0.5 0.2
0.3 0.5 0.2 0.1
h_1
h_2
h_3
h_4
h_5
s_1
Tôi
đang
học
tiếng
anh
<start>
Decoder

Attention

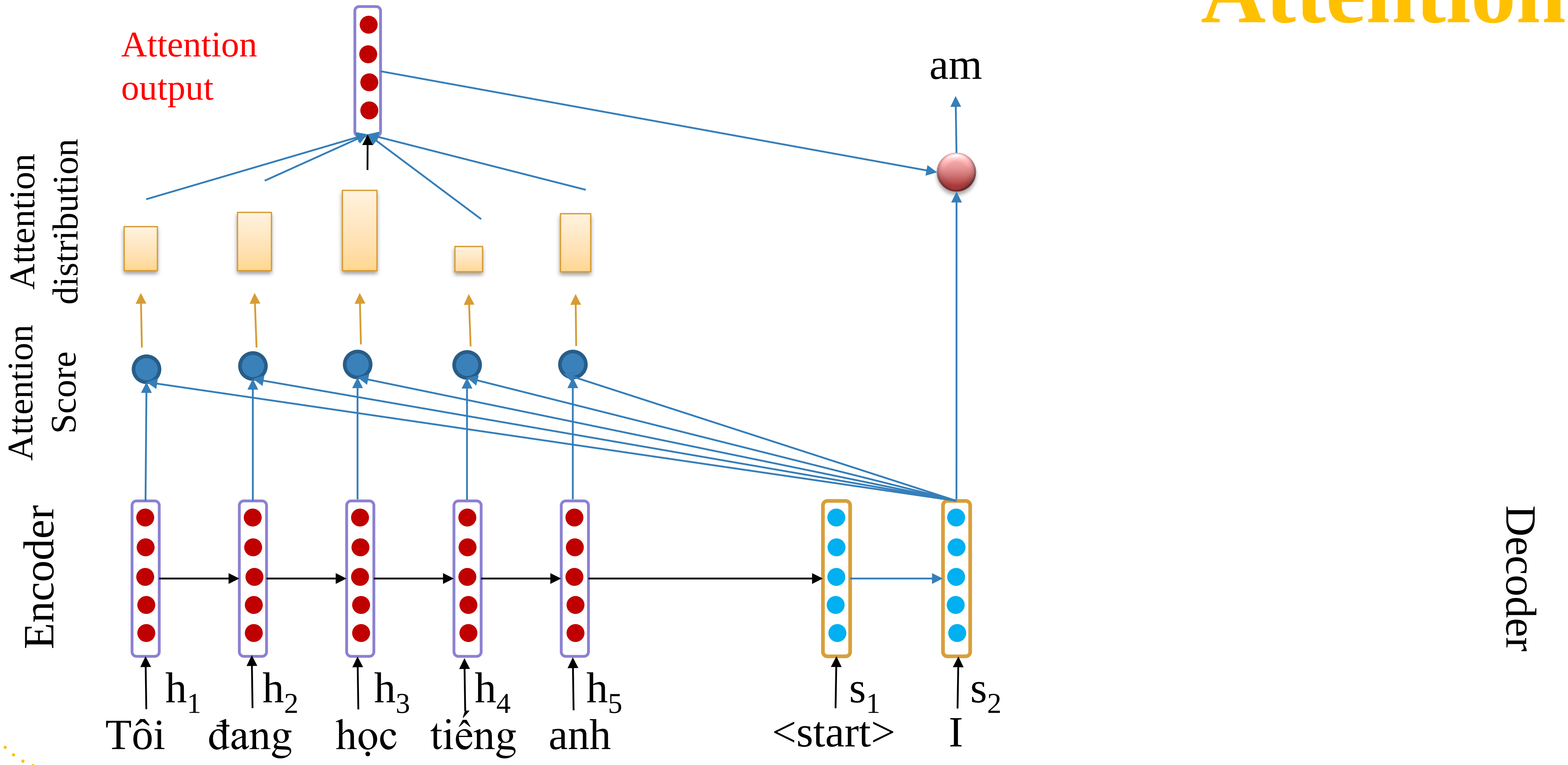


Attention output
I
Concat [a^t, s^t]
Attention distribution
α_1
α_2
α_3
α_4
α_5
Attention Score
$s_1^T h_1$
$s_1^T h_2$
$s_1^T h_3$
$s_1^T h_3$
$s_1^T h_4$
Encoder
Decoder
h_1
h_2
h_3
h_4
h_5
s_1
Tôi
đang
học
tiếng
anh
<start>

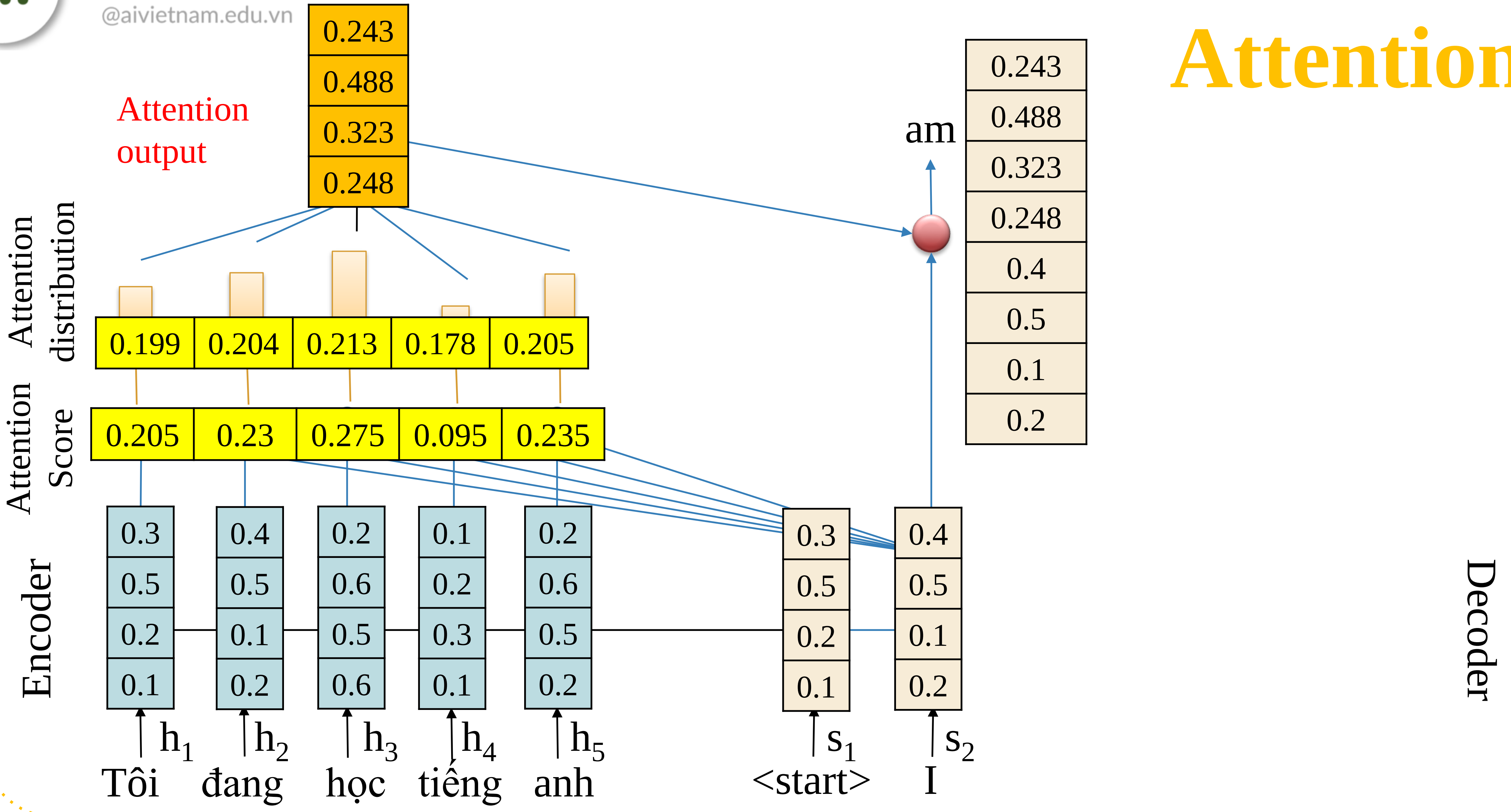


AI VIET NAM
@aivietnam.edu.vn
Attention
Attention output
0.242
0.488
0.324
0.247
I
Concat [a^t, s^t]
0.242
0.488
0.324
0.247
0.3
0.5
0.2
0.1
Attention distribution
α_1 α_2 α_3 α_4 α_5
0.198 0.201 0.212 0.181 0.208
Attention Score
$s_1^T h_1$ $s_1^T h_2$ $s_1^T h_3$ $s_1^T h_3$ $s_1^T h_4$
0.195 0.205 0.26 0.1 0.24
Encoder
0.3 0.5 0.2 0.1
0.4 0.5 0.1 0.2
0.2 0.6 0.5 0.6
0.1 0.2 0.3 0.1
0.2 0.6 0.5 0.2
0.3 0.5 0.2 0.1
h_1 h_2 h_3 h_4 h_5 s_1
Tôi đang học tiếng anh
<start>
Decoder

Attention

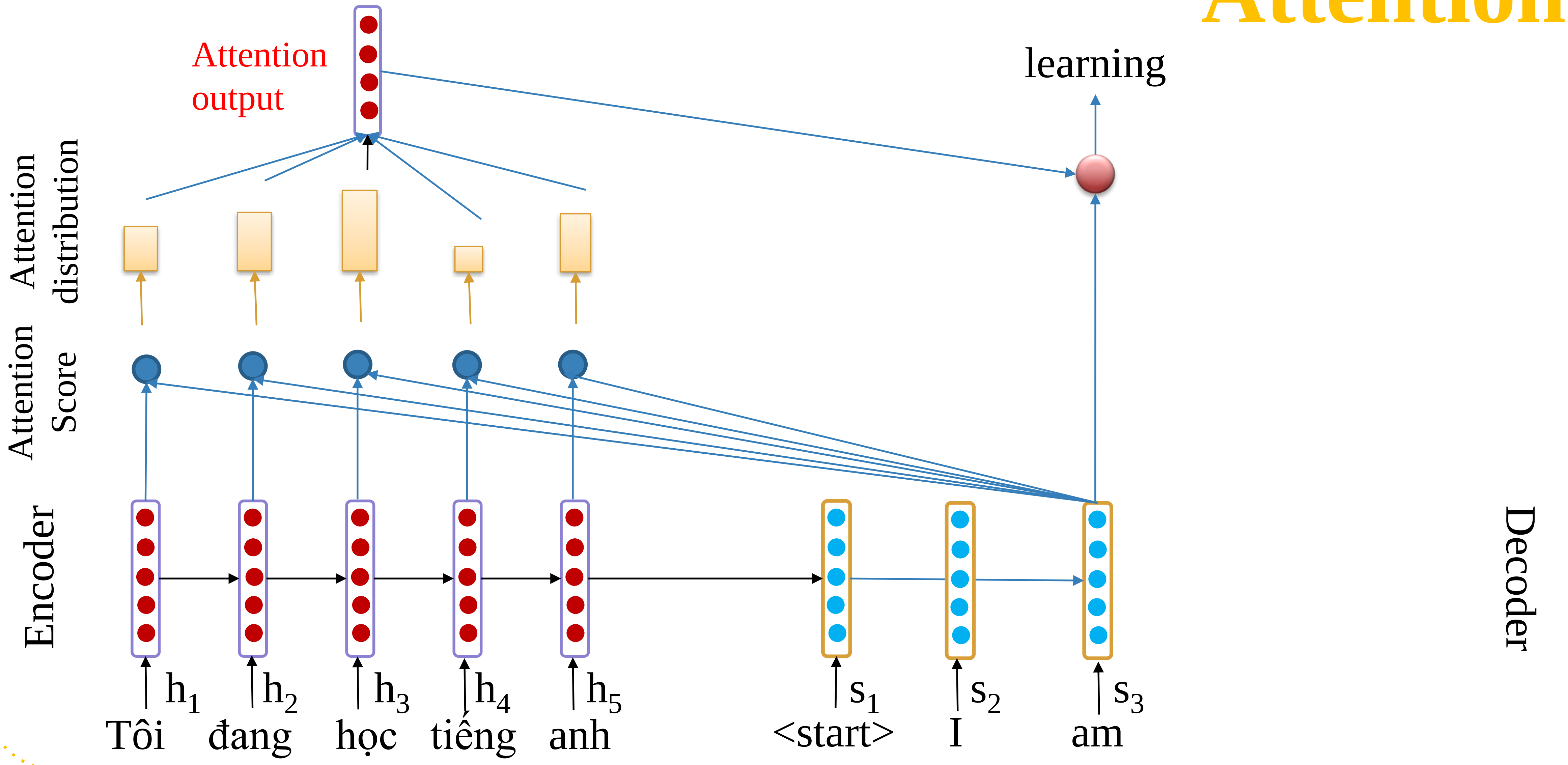

Attention output
am
Attention distribution
Attention Score
Encoder
Decoder
h1
h2
h3
h4
h5
s1
s2
Tôi
đang
học
tiếng
anh
<start>
I

Attention

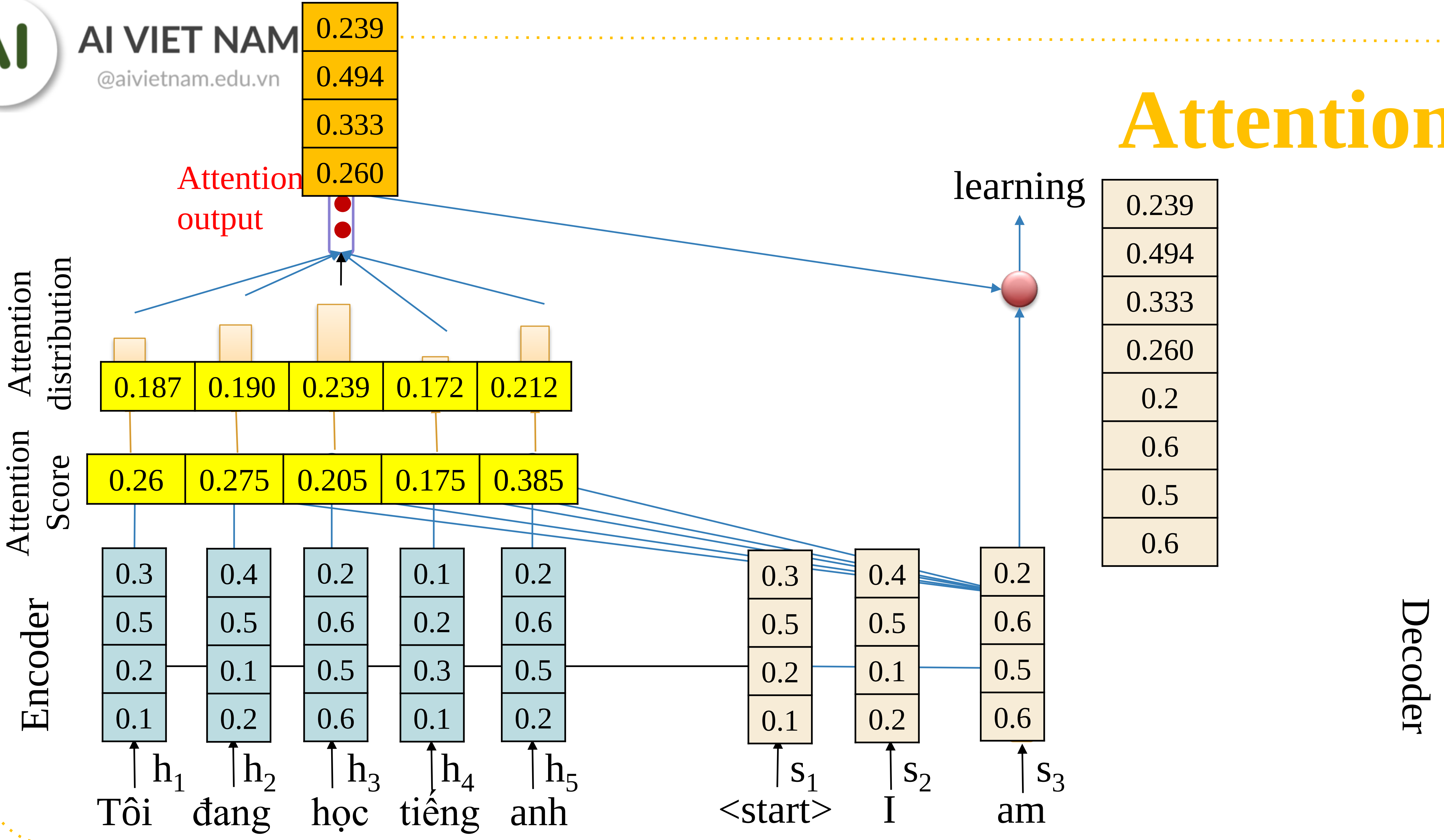

AI VIET NAM
@aivietnam.edu.vn
Attention output
0.243
0.488
0.323
0.248
Attention distribution
0.199
0.204
0.213
0.178
0.205
Attention Score
0.205
0.23
0.275
0.095
0.235
Encoder
0.3 0.5 0.2 0.1
0.4 0.5 0.1 0.2
0.2 0.6 0.5 0.6
0.1 0.2 0.3 0.1
0.2 0.6 0.5 0.2
h1 h2 h3 h4 h5
Tôi đang học tiếng anh
0.3 0.5 0.2 0.1
0.4 0.5 0.1 0.2
s1 s2
<start> I
am
0.243
0.488
0.323
0.248
0.4
0.5
0.1
0.2
Decoder

AI VIET NAM
@aivietnam.edu.vn

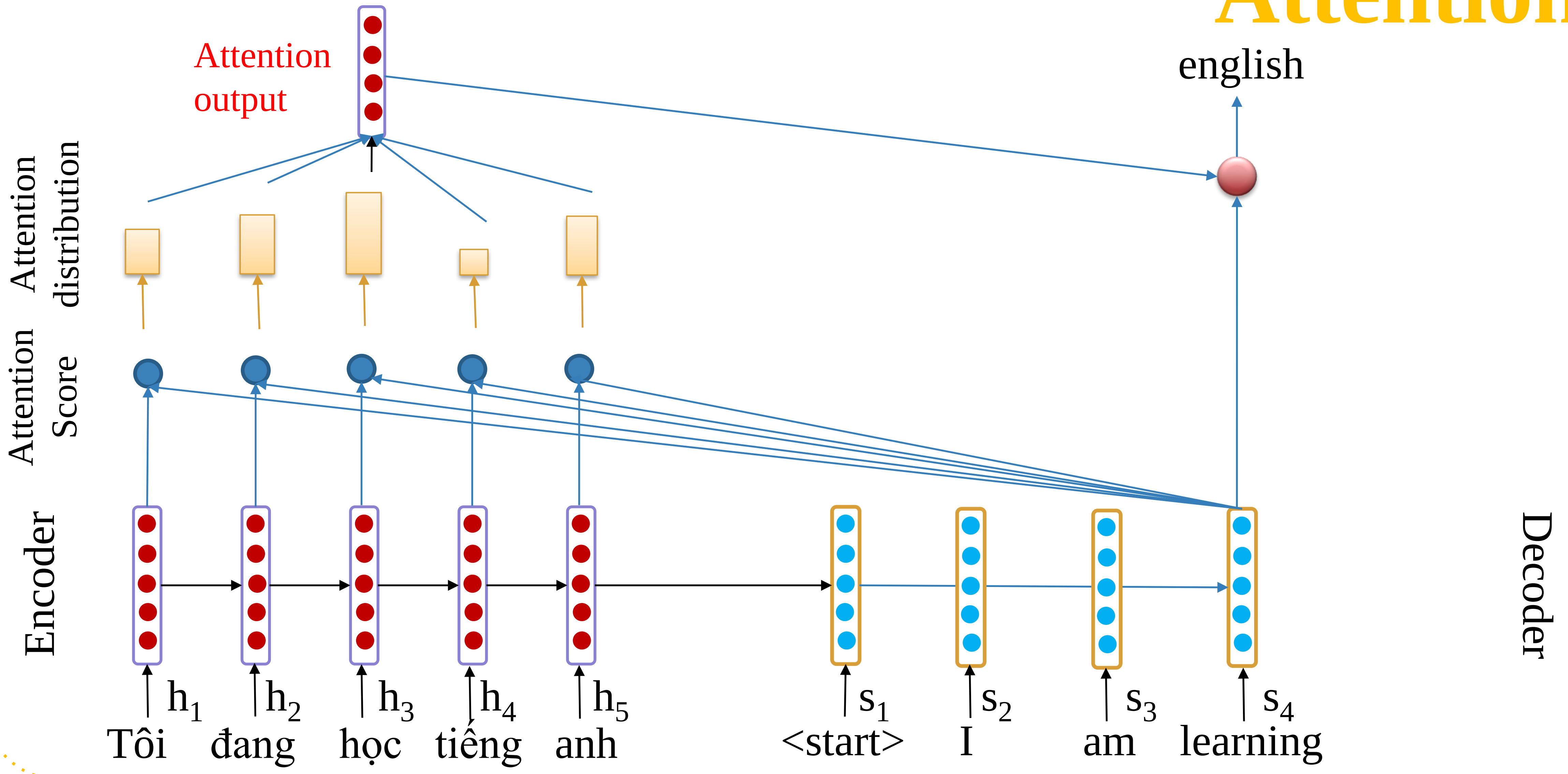
Attention



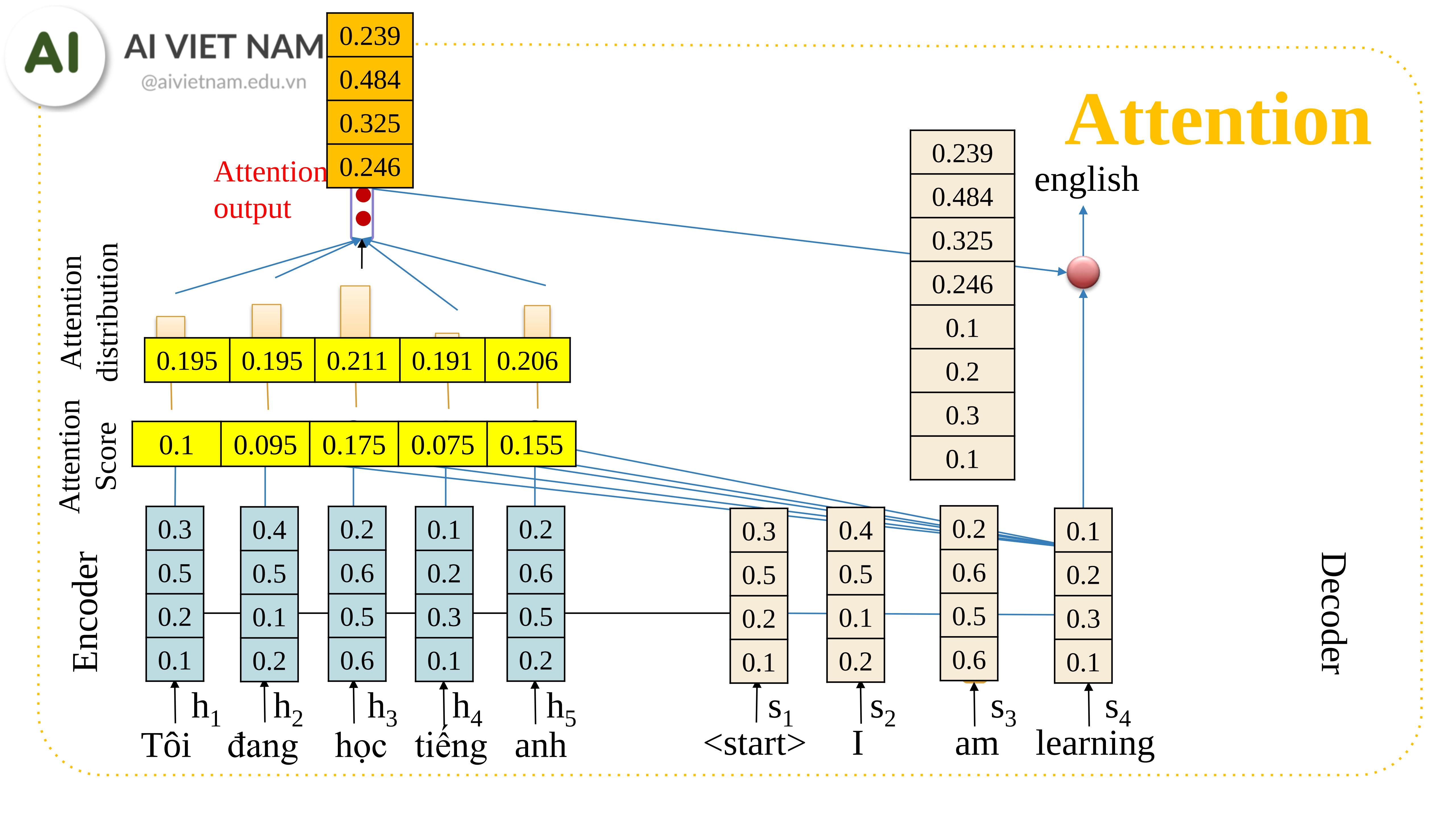
Attention


0.239
0.494
0.333
0.260
Attention output
learning
0.239
0.494
0.333
0.260
0.2
0.6
0.5
0.6
Attention distribution
0.187 0.190 0.239 0.172 0.212
Attention Score
0.26 0.275 0.205 0.175 0.385
Encoder
0.3 0.5 0.2 0.1
0.4 0.5 0.1 0.2
0.2 0.6 0.5 0.6
0.1 0.2 0.3 0.1
0.2 0.6 0.5 0.2
0.3 0.5 0.2 0.1
0.4 0.5 0.1 0.2
0.2 0.6 0.5 0.6
Decoder
h_1 h_2 h_3 h_4 h_5
Tôi đang học tiếng anh
s_1 s_2 s_3
<start> I am

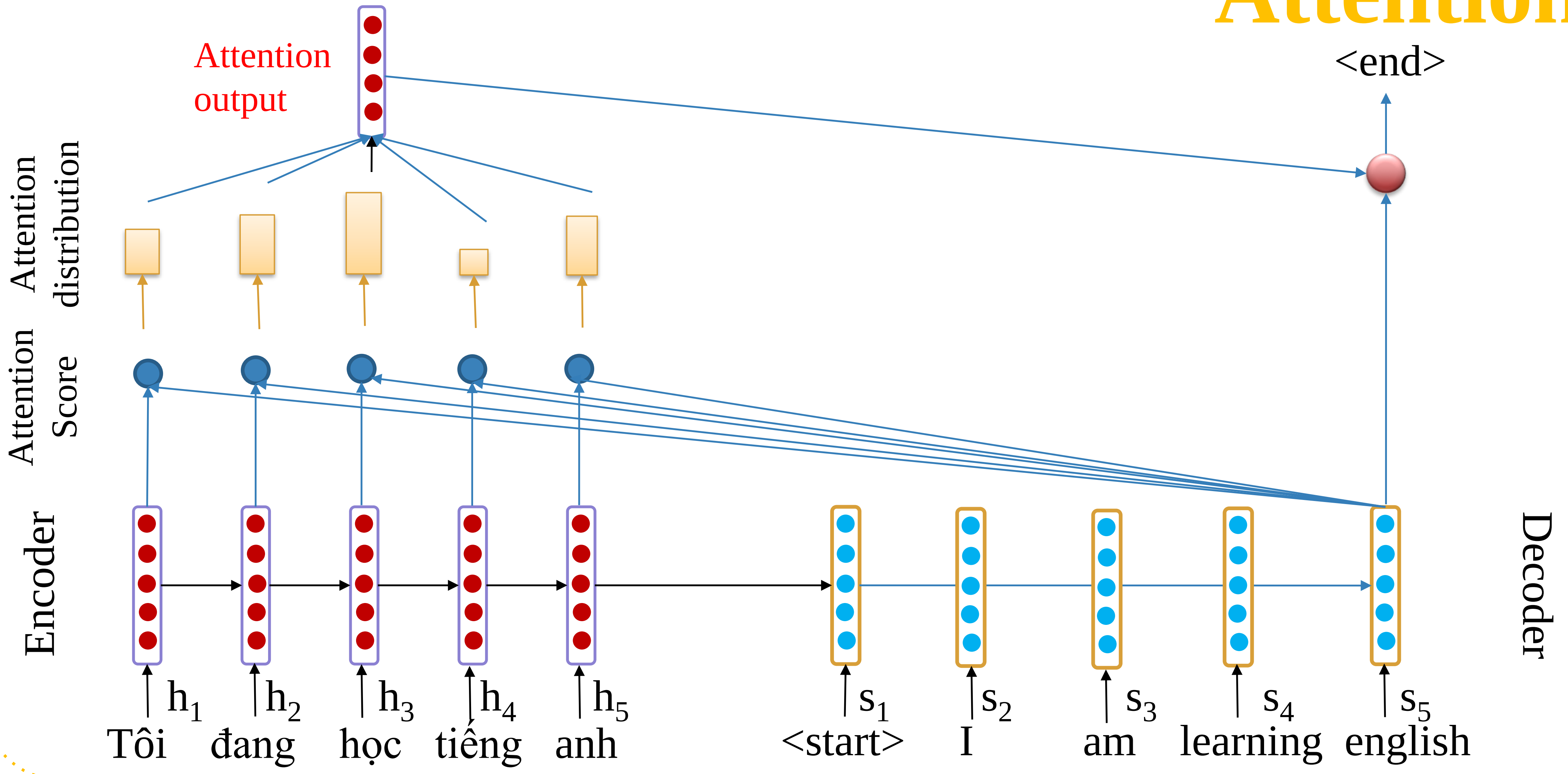
Attention



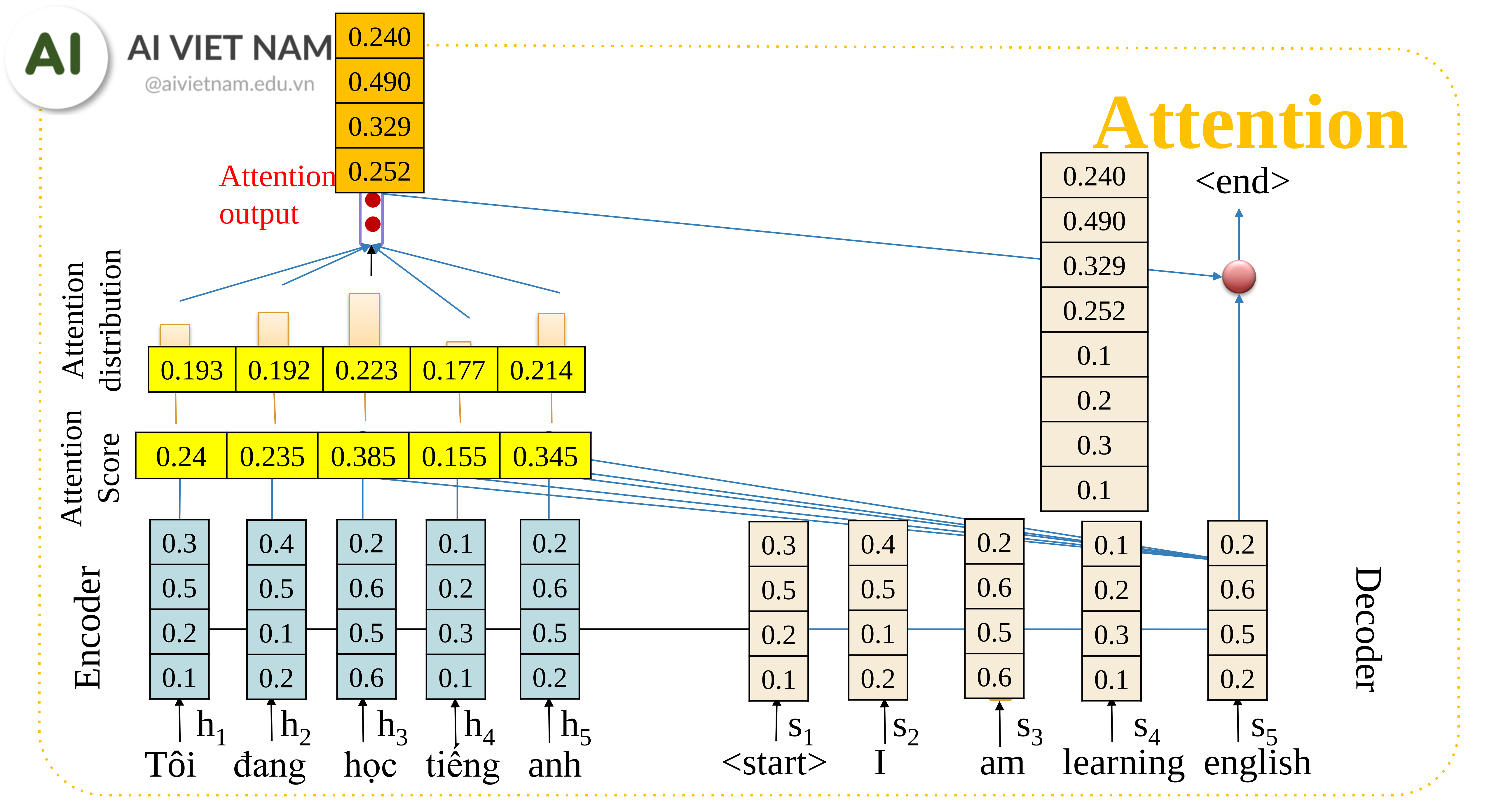
english
Attention
output
Attention
distribution
Attention
Score
Encoder
Decoder
h_1 h_2 h_3 h_4 h_5
Tôi đang học tiếng anh
s_1 s_2 s_3 s_4
<start> I am learning


AI VIET NAM
@aivietnam.edu.vn
Attention
Attention output
0.239
0.484
0.325
0.246
english
0.239
0.484
0.325
0.246
0.1
0.2
0.3
0.1
Attention distribution
0.195
0.195
0.211
0.191
0.206
Attention Score
0.1
0.095
0.175
0.075
0.155
Encoder
0.3 0.5 0.2 0.1
0.4 0.5 0.1 0.2
0.2 0.6 0.5 0.6
0.1 0.2 0.3 0.1
0.2 0.6 0.5 0.2
h1 h2 h3 h4 h5
Tôi đang học tiếng anh
0.3 0.5 0.2 0.1
0.4 0.5 0.1 0.2
0.2 0.6 0.5 0.6
0.1 0.2 0.3 0.1
s1 s2 s3 s4
<start> I am learning
Decoder

Attention



<end>
Attention output
Attention distribution
Attention Score
Encoder
Decoder
h_1 h_2 h_3 h_4 h_5
Tôi đang học tiếng anh
s_1 s_2 s_3 s_4 s_5
<start> I am learning english



AI VIET NAM
@aivietnam.edu.vn
Attention
0.240
0.490
0.329
0.252
Attention output
Attention distribution
0.193 0.192 0.223 0.177 0.214
Attention Score
0.24 0.235 0.385 0.155 0.345
Encoder
0.3 0.5 0.2 0.1
0.4 0.5 0.1 0.2
0.2 0.6 0.5 0.6
0.1 0.2 0.3 0.1
0.2 0.6 0.5 0.2
h1 h2 h3 h4 h5
Tôi đang học tiếng anh
0.3 0.5 0.2 0.1
0.4 0.5 0.1 0.2
0.2 0.6 0.5 0.6
0.1 0.2 0.3 0.1
0.2 0.6 0.5 0.2
s1 s2 s3 s4 s5
<start> I am learning english
0.240 0.490 0.329 0.252 0.1 0.2 0.3 0.1
<end>
Decoder

Attention

```
def scaled_dot_product_attention( q, k, v, mask=None):
        dk = tf.cast(tf.shape(k)[-1], tf.float32)
        attention_scores = tf.matmul(q, k, transpose_b=True) / tf.math.sqrt(dk)

        if mask is not None:
            attention_scores += (mask * -1e9)

        attention_weights = tf.nn.softmax(attention_scores, axis=-1)

        output = tf.matmul(attention_weights, v)

        return output, attention_weights, attention_scores
```

```
import tensorflow as tf
input = tf.constant([[0.3, 0.5, 0.2, 0.1],
                     [0.4, 0.5, 0.1, 0.2],
                     [0.2, 0.6, 0.5, 0.6],
                     [0.1, 0.2, 0.3, 0.1],
                     [0.2, 0.6, 0.5, 0.2]])
input
```

```
<tf.Tensor: shape=(5, 4), dtype=float32, numpy=
array([[0.3, 0.5, 0.2, 0.1],
       [0.4, 0.5, 0.1, 0.2],
       [0.2, 0.6, 0.5, 0.6],
       [0.1, 0.2, 0.3, 0.1],
       [0.2, 0.6, 0.5, 0.2]], dtype=float32)>
```

```
scaled_dot_product_attention(input, input, input)
```

```
(<tf.Tensor: shape=(5, 4), dtype=float32, numpy=
 array([[0.24194102, 0.48778105, 0.32396913, 0.24687952],
        [0.2428891 , 0.48836532, 0.32299504, 0.24765417],
        [0.23951267, 0.49354896, 0.33351758, 0.25972563],
        [0.23946747, 0.48449475, 0.32505167, 0.24576576],
        [0.23998849, 0.49056447, 0.32979524, 0.25222978]], dtype=float32)>,
 <tf.Tensor: shape=(5, 5), dtype=float32, numpy=
 array([[0.19870403, 0.20070107, 0.21204881, 0.18069609, 0.20784996],
        [0.19904016, 0.20407887, 0.21347219, 0.17830697, 0.20510183],
        [0.1871119 , 0.18993972, 0.23905815, 0.17186458, 0.21202557],
        [0.19589609, 0.19491905, 0.21115328, 0.19105938, 0.20697217],
        [0.19303995, 0.19207716, 0.22316182, 0.17730956, 0.21441151]],
       dtype=float32)>,
 <tf.Tensor: shape=(5, 5), dtype=float32, numpy=
 array([[0.195     , 0.20500001, 0.26      , 0.1       , 0.24000001],
        [0.20500001, 0.23      , 0.275     , 0.09500001, 0.23500001],
        [0.26      , 0.275     , 0.505     , 0.17500001, 0.385     ],
        [0.1       , 0.09500001, 0.17500001, 0.075     , 0.155     ],
        [0.24000001, 0.23500001, 0.385     , 0.155     , 0.345     ]],
       dtype=float32)>)
```

Attention Variants

Compute $e \in \mathbb{R}^N$ from $h_1, h_2, \ldots, h_N \in \mathbb{R}^{d_1}$ and $s \in \mathbb{R}^{d_2}$

- Dot-product attention $e_i = s^T h_i$

  $d_1 = d_2$

- Multiplicative attention $e_i = s^T W h_i$

  $W \in \mathbb{R}^{d_2 \times d_1}$

- Additive attention $e_i = v^T \tanh(W_1 h_i + W_2 s)$

  $W_1 \in \mathbb{R}^{d_3 \times d_1}, W_2 \in \mathbb{R}^{d_3 \times d_2}, v \in \mathbb{R}^{d_3}$

AI VIET NAM
@aivietnam.edu.vn

Transformer

Nguyễn Quốc Thái

Transformer

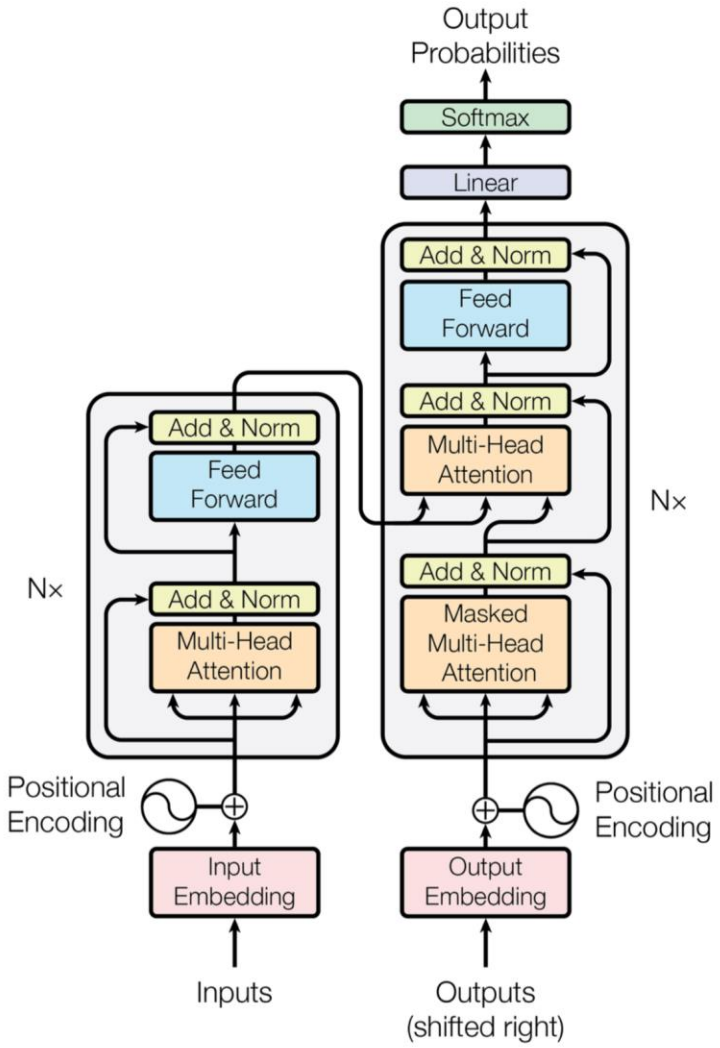
- Machine translation result

[Ref] - Attention is all you need

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**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Transformer

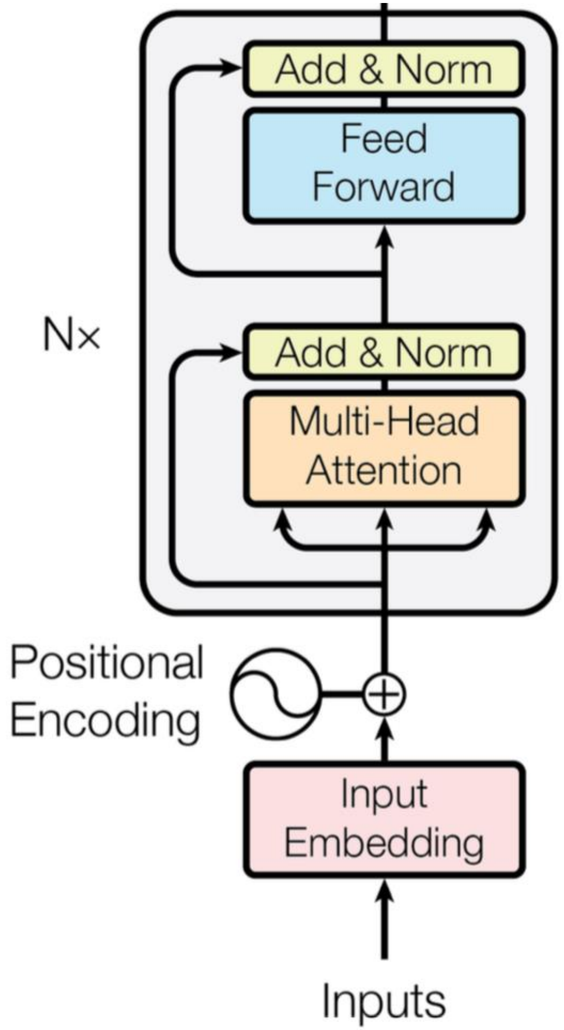
- Transformer Architecture
  - N Encoder layer
  - N Decoder layer
- Core technique: attention
- Loss function: cross-entropy



Ref

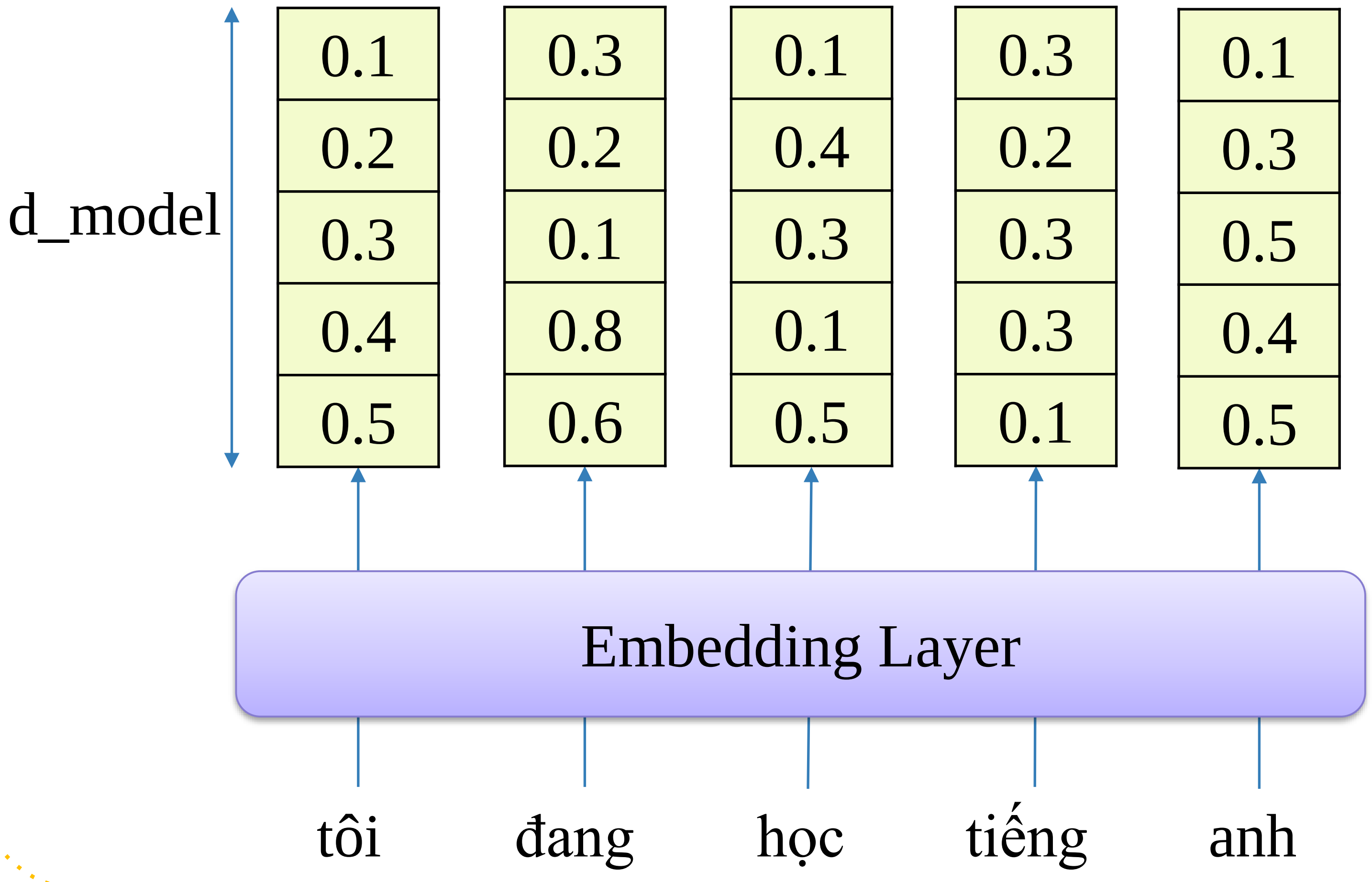
Transformer Encoder

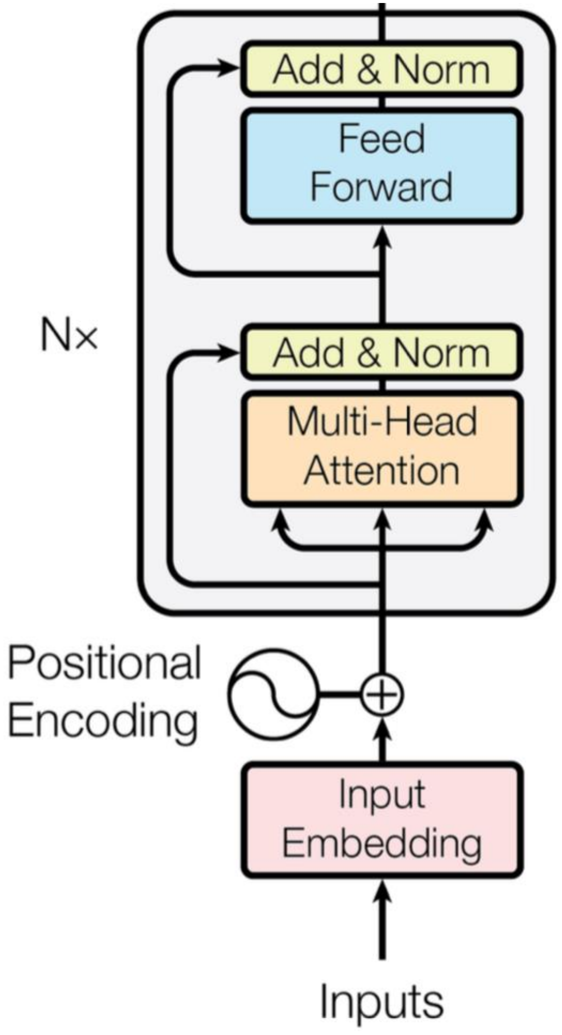
- To learn the relationship between word in the sentence
- Transformer encoder:
  - Input Embedding
  - Positional Encoding
  - N encoder layer:
    - Multi-Head Attention
    - Normalization Layer
    - Feed Forward Layer



Transformer Encoder

➢ Embedding layer

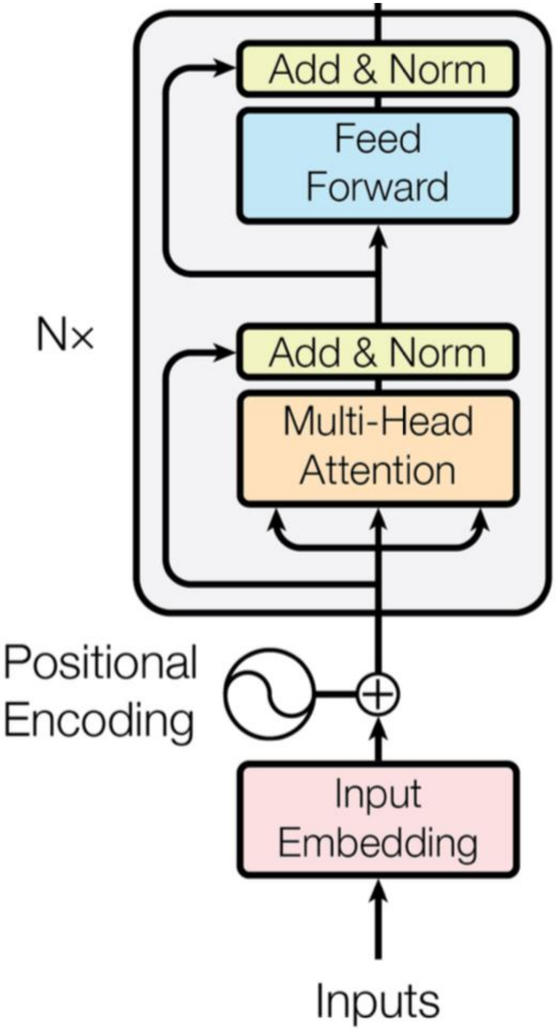




Transformer Encoder

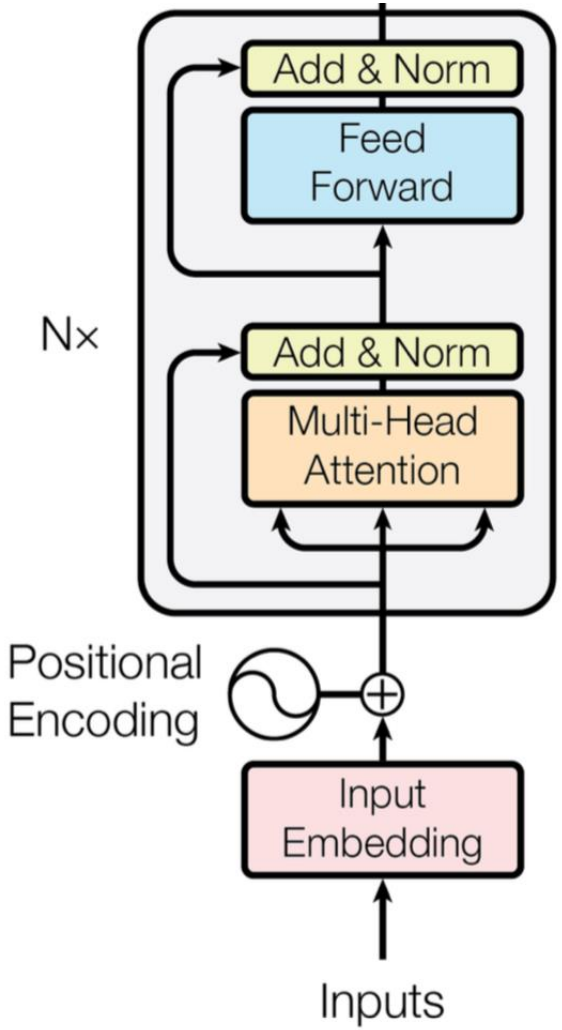
- Positional Encoding: describe the position of an entity in a sequence as unique representation – each position is mapped to a vector

Sequence	Index of token	Positional Encoding Matrix			
tôi	0	P_{00}	P_{01}	…	$P_{0(d-1)}$
đang	1	P_{10}	P_{12}	…	$P_{1(d-1)}$
học	2	P_{20}	P_{21}	…	$P_{2(d-1)}$
tiếng	3	P_{30}	P_{31}	…	$P_{3(d-1)}$
anh	4	P_{40}	P_{41}	…	$P_{4(d-1)}$



Transformer Encoder

- Positional Encoding: describe the position of an entity in a sequence as unique representation – each position is mapped to a vector
- Attention: position insensitive
- Some solutions:
  - Learned positional embedding (as learned token embedding)
  - Sinusoid



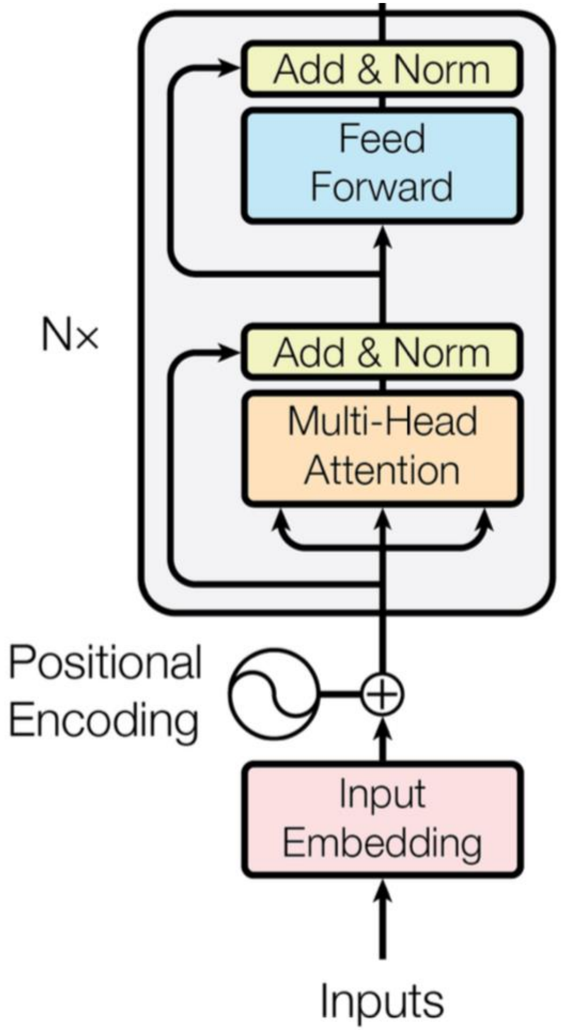
Transformer Encoder

- Sinusoid

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{n^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{n^{\frac{2i}{d_{model}}}}\right)$$

- pos: position of an entity in input sentence, 0 => L/2
- d_model: dimension of the output embedding space
- n = 10000 (recommend)
- d_model: dimension of the output embedding space



Transformer Encoder

- Sinusoid:

Example: d_model = 5, n=100

$$P_{00} = \sin\left(\frac{0}{100^{\frac{2*0}{5}}}\right)$$

Sequence | Index of token | Positional Encoding Matrix

i=0

Sequence	Index of token					
tôi	0	$P_{00} = 0$				
đang	1					
học	2					
tiếng	3					
anh	4					

Transformer Encoder

- Sinusoid:

Example: d_model = 5, n=100

$$P_{00} = \sin\left(\frac{0}{100^{\frac{2*0}{5}}}\right)$$

$$P_{01} = \cos\left(\frac{0}{100^{\frac{2*0}{5}}}\right)$$

Sequence	Index of token	Positional Encoding Matrix (i=0)	(i=0)			
tôi	0	P_{00}= 0	P_{01}= 1			
đang	1					
học	2					
tiếng	3					
anh	4					

AI VIET NAM
@aivietnam.edu.vn

Transformer Encoder

- Sinusoid:

Example: d_model = 5, n=100

$$P_{00} = \sin\left(\frac{0}{100^{\frac{2*0}{5}}}\right)$$

$$P_{01} = \cos\left(\frac{0}{100^{\frac{2*0}{5}}}\right)$$

Sequence	Index of token	i=0	i=0	i=1	i=1	i=2
tôi	0	P_{00}= 0	P_{01}= 1	P_{02}= 0	P_{03}= 1	P_{04}= 0
đang	1	P_{10}= 0.84	P_{11}=0.54	P_{12}=0.16	P_{13}=0.99	P_{14}=0.025
học	2	P_{20}= 0.91	P_{21}=-0.42	P_{22}=0.31	P_{23}=0.95	P_{24}=0.05
tiếng	3	P_{30}= 0.14	P_{31}=-0.99	P_{32}=0.46	P_{33}=0.89	P_{34}=0.141
anh	4	P_{40}= -0.76	P_{41}=-0.65	P_{42}=0.59	P_{43}=0.81	P_{4d}=0.1

Positional Encoding Matrix

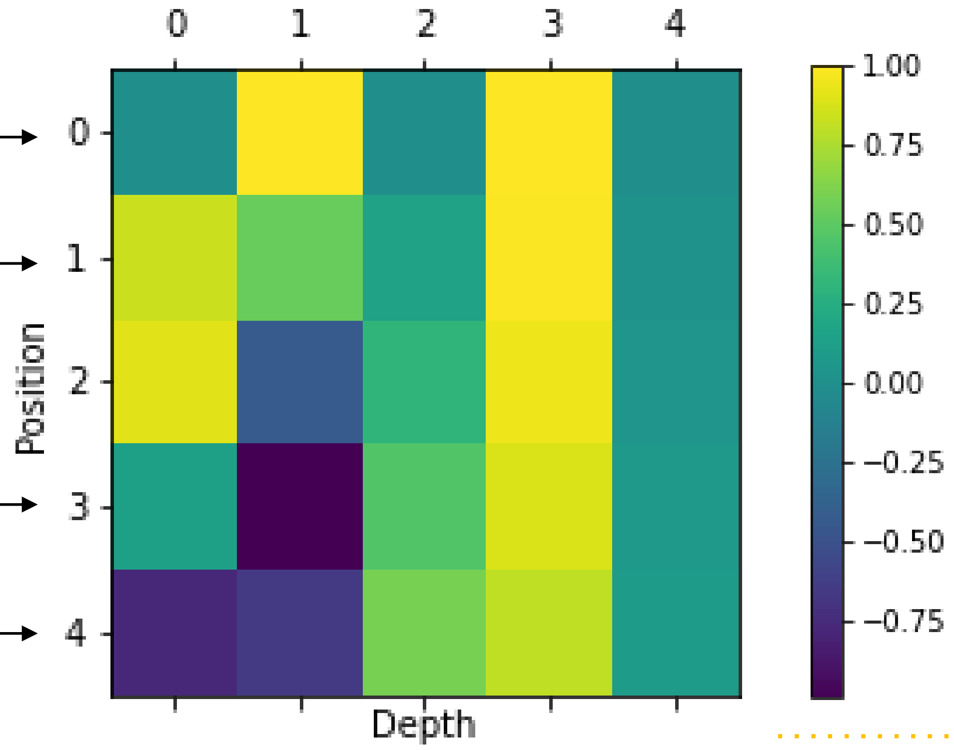
Transformer Encoder

➢ Sinusoid:

Example: d_model = 5, n=100

P_{00}= 0	P_{01}= 1	P_{02}= 0	P_{03}= 1	P_{04}= 0
P_{10}= 0.84	P_{11}=0.54	P_{12}=0.16	P_{13}=0.99	P_{14}=0.025
P_{20}= 0.91	P_{21}=-0.42	P_{22}=0.31	P_{23}=0.95	P_{24}=0.05
P_{30}= 0.14	P_{31}=-0.99	P_{32}=0.46	P_{33}=0.89	P_{34}=0.141
P_{40}= -0.76	P_{41}=-0.65	P_{42}=0.59	P_{43}=0.81	P_{4d}=0.1

Sequence	Index of token
tôi	0
đang	1
học	2
tiếng	3
anh	4



AI VIET NAM
@aivietnam.edu.vn

Transformer Encoder

➢ Positional encoding

```
[ ]  def get_angles(pos, i, d_model):
         angle_rates = 1 / np.power(100, (2 * (i//2)) / np.float32(d_model))
         return pos * angle_rates

     def positional_encoding(position, d_model):
         angle_rads = get_angles(np.arange(position)[:, np.newaxis],
                                 np.arange(d_model)[np.newaxis, :],
                                 d_model)

         # apply sin to even indices in the array; 2i
         angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])

         # apply cos to odd indices in the array; 2i+1
         angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])

         pos_encoding = angle_rads[np.newaxis, ...]

         return tf.cast(pos_encoding, dtype=tf.float32)
```

```
[ ]  pe = positional_encoding(5, 5)
     pe
```

```
<tf.Tensor: shape=(1, 5, 5), dtype=float32, numpy=
array([[[ 0.        ,  1.        ,  0.        ,  1.        ,
          0.        ],
        [ 0.84147096,  0.5403023 ,  0.15782665,  0.9874668 ,
          0.02511622],
        [ 0.9092974 , -0.41614684,  0.31169716,  0.9501815 ,
          0.0502166 ],
        [ 0.14112   , -0.9899925 ,  0.45775455,  0.8890786 ,
          0.07528529],
        [-0.7568025 , -0.6536436 ,  0.5923377 ,  0.80568975,
          0.10030649]]], dtype=float32)>
```

Transformer Encoder

➢ Token embedding with positional embedding $z_i = WE(i) + PE(i)$

tôi
đang
học
tiếng
anh

0.1	0.2	0.3	0.4	0.5
0.3	0.2	0.1	0.8	0.6
0.1	0.4	0.3	0.1	0.5
0.3	0.2	0.3	0.3	0.1
0.1	0.3	0.5	0.4	0.5

0	1	0	1	0
0.84	0.54	0.16	0.99	0.025
0.91	-0.416	0.311	0.95	0.05
0.141	-0.99	0.46	0.89	0.07
-0.76	-0.65	0.59	0.81	0.1

+

Final Embedding

0.1	1.2	0.3	1.4	0.5
1.14	0.74	0.26	1.79	0.625
1.01	-0.016	0.611	1.05	0.55
0.44	-0.79	0.76	1.12	0.176
-0.65	-0.35	1.09	1.21	0.6

Transformer Encoder

- Token embedding with positional embedding $z_i = WE(i) + PE(i)$

```
[ ] we = tf.constant([[0.1, 0.2, 0.3, 0.4, 0.5],
                      [0.3, 0.2, 0.1, 0.8, 0.6],
                      [0.1, 0.4, 0.3, 0.1, 0.5],
                      [0.3, 0.2, 0.3, 0.3, 0.1],
                      [0.1, 0.3, 0.5, 0.4, 0.5]])
    we
```

```
<tf.Tensor: shape=(5, 5), dtype=float32, numpy=
array([[0.1, 0.2, 0.3, 0.4, 0.5],
       [0.3, 0.2, 0.1, 0.8, 0.6],
       [0.1, 0.4, 0.3, 0.1, 0.5],
       [0.3, 0.2, 0.3, 0.3, 0.1],
       [0.1, 0.3, 0.5, 0.4, 0.5]], dtype=float32)>
```

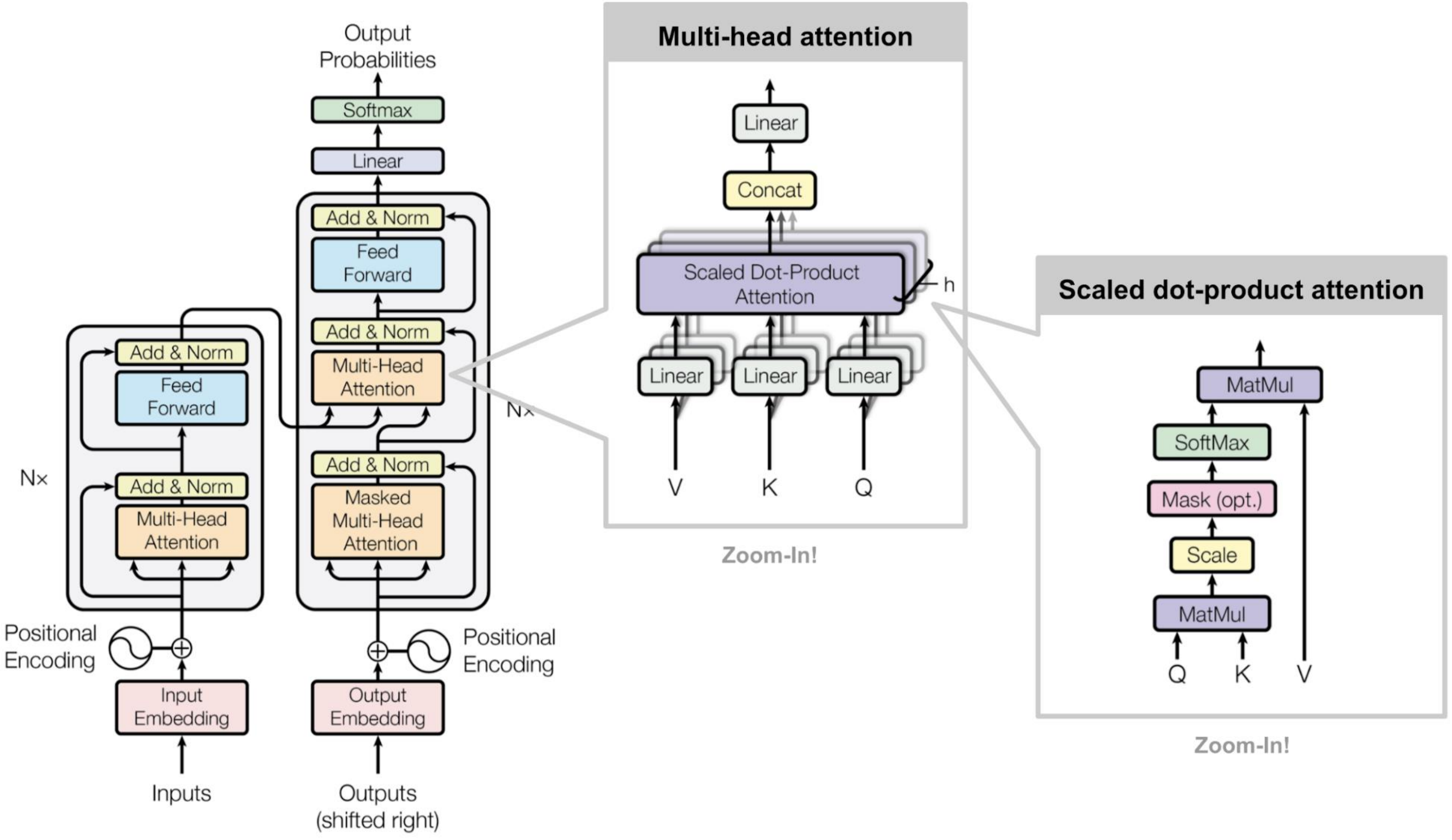
```
[ ] pe = positional_encoding(5, 5)
    pe
```

```
<tf.Tensor: shape=(1, 5, 5), dtype=float32, numpy=
array([[[ 0.        ,  1.        ,  0.        ,  1.        ,
          0.        ],
        [ 0.84147096,  0.5403023 ,  0.15782665,  0.9874668 ,
          0.02511622],
        [ 0.9092974 , -0.41614684,  0.31169716,  0.9501815 ,
          0.0502166 ],
        [ 0.14112   , -0.9899925 ,  0.45775455,  0.8890786 ,
          0.07528529],
        [-0.7568025 , -0.6536436 ,  0.5923377 ,  0.80568975,
          0.10030649]]], dtype=float32)>
```

```
e = pe + we
e
```

```
<tf.Tensor: shape=(1, 5, 5), dtype=float32, numpy=
array([[[ 0.1       ,  1.2       ,  0.3       ,  1.4       ,
          0.5       ],
        [ 1.1414709 ,  0.74030226,  0.25782666,  1.7874668 ,
          0.6251162 ],
        [ 1.0092974 , -0.01614684,  0.6116972 ,  1.0501815 ,
          0.5502166 ],
        [ 0.44112003, -0.7899925 ,  0.75775456,  1.1890786 ,
          0.1752853 ],
        [-0.6568025 , -0.3536436 ,  1.0923377 ,  1.2056898 ,
          0.6003065 ]]], dtype=float32)>
```

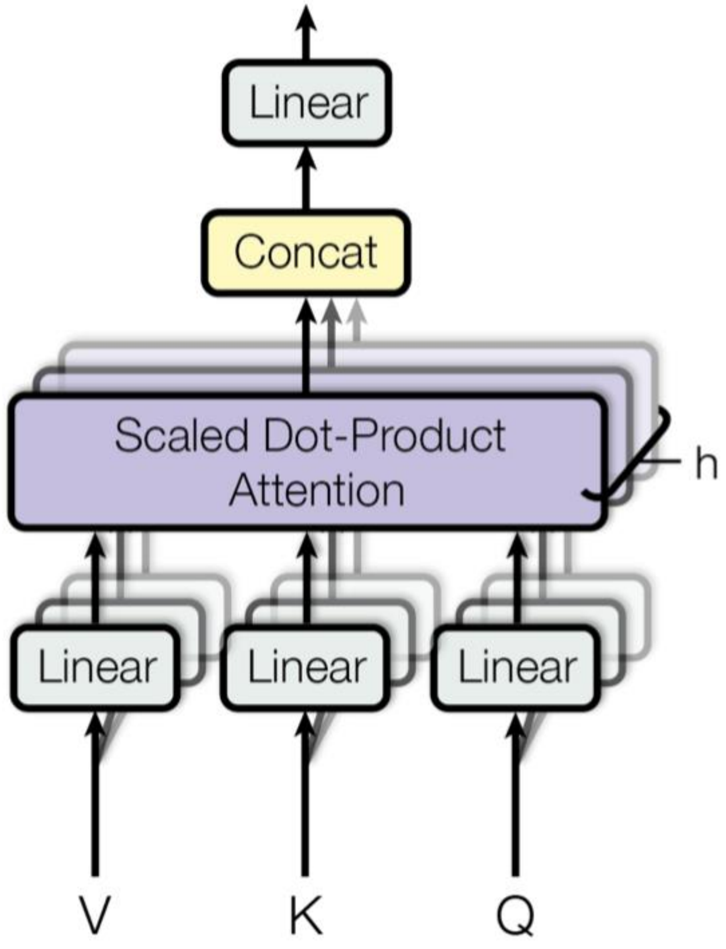
Multi-head attention



Multi-head attention

Multi-head Attention:

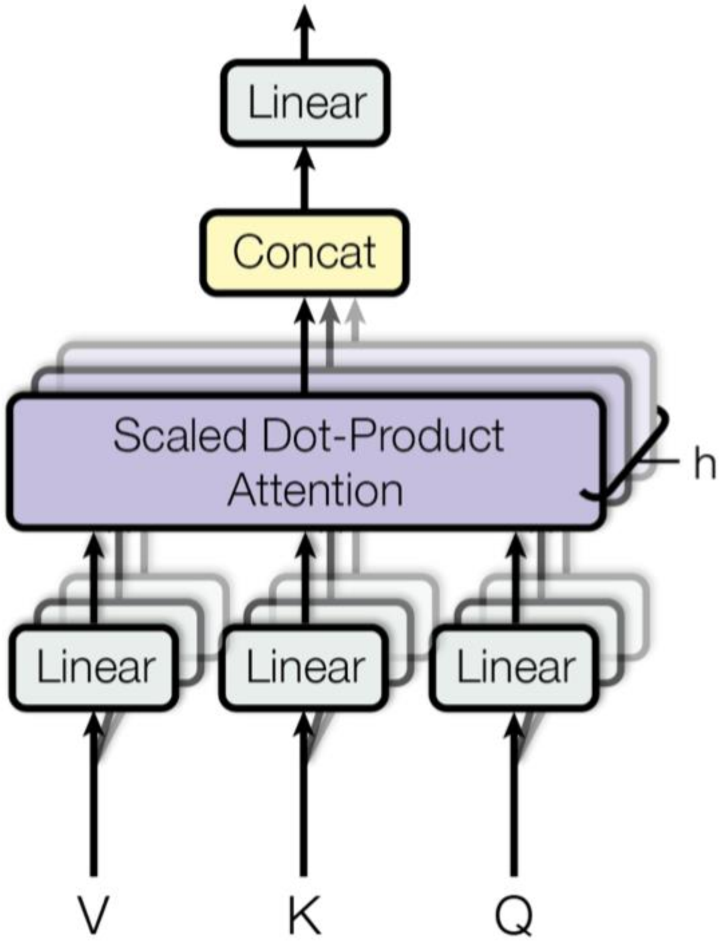
- Linear layers and split into heads
- Scaled dot-product attention
- Concat output
- Final linear layer



Multi-head attention

- Linear layers and split into heads:
- 3 linear layers for query, key and value
- Input passed through 3 linear layers to produce the Q, K, V matrices
- Split into the multiple attention heads

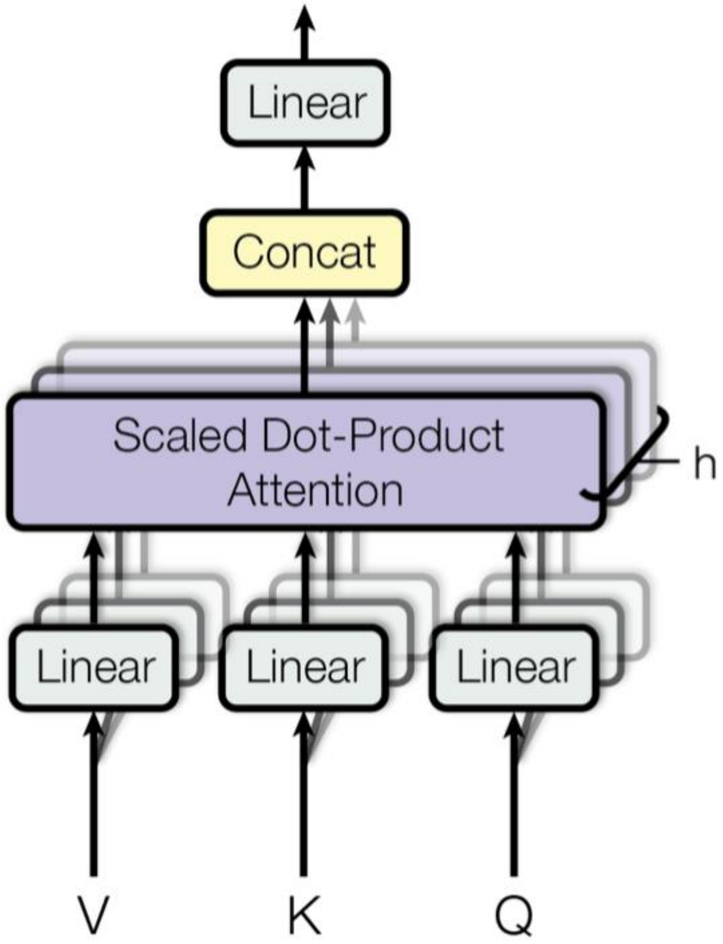
Each head process independently



Multi-head attention

- ➢ Linear layers and split into heads:
- 3 linear layers for query, key and value
- Input passed through 3 linear layers to produce the Q, K, V matrices
- Split into the multiple attention heads

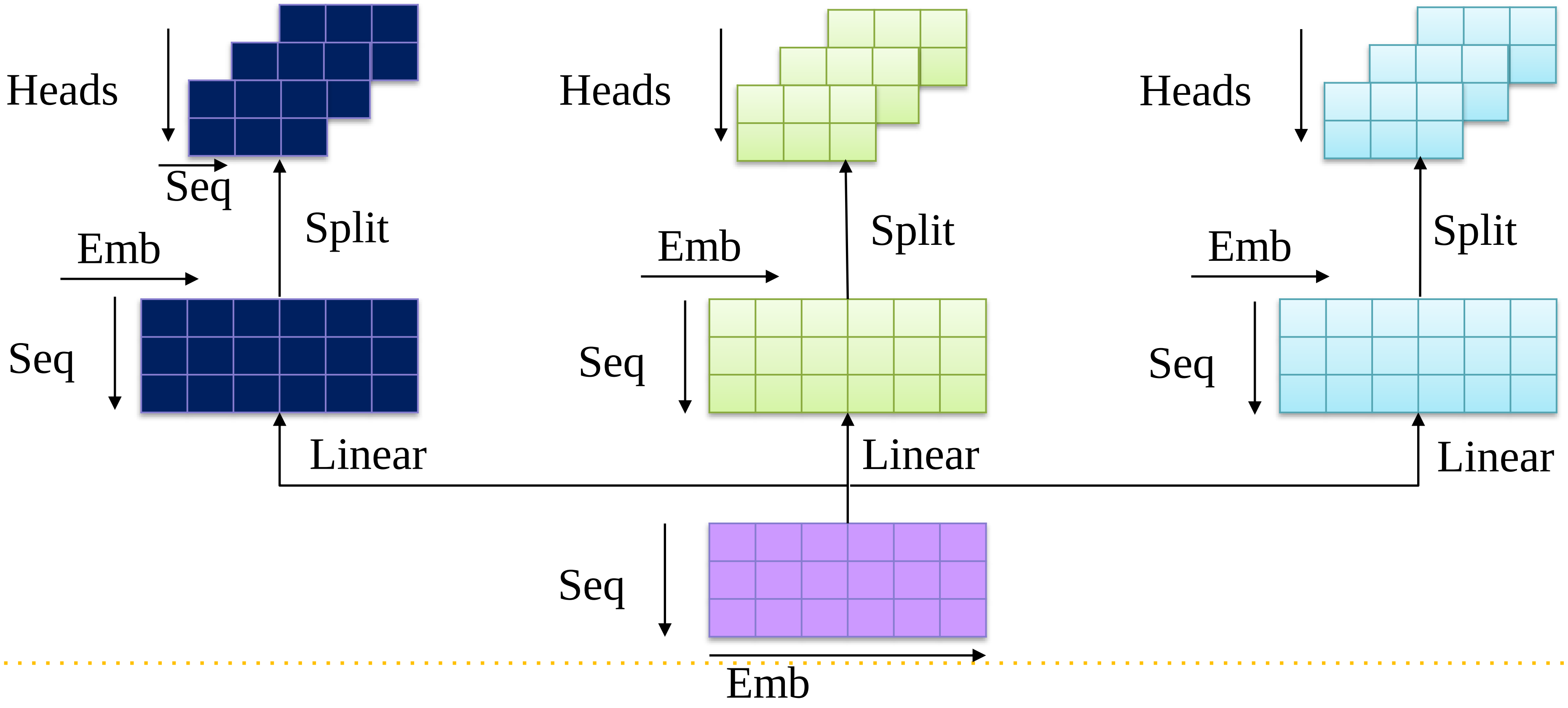
Each head process independently



AI VIET NAM
@aivietnam.edu.vn

Multi-head attention

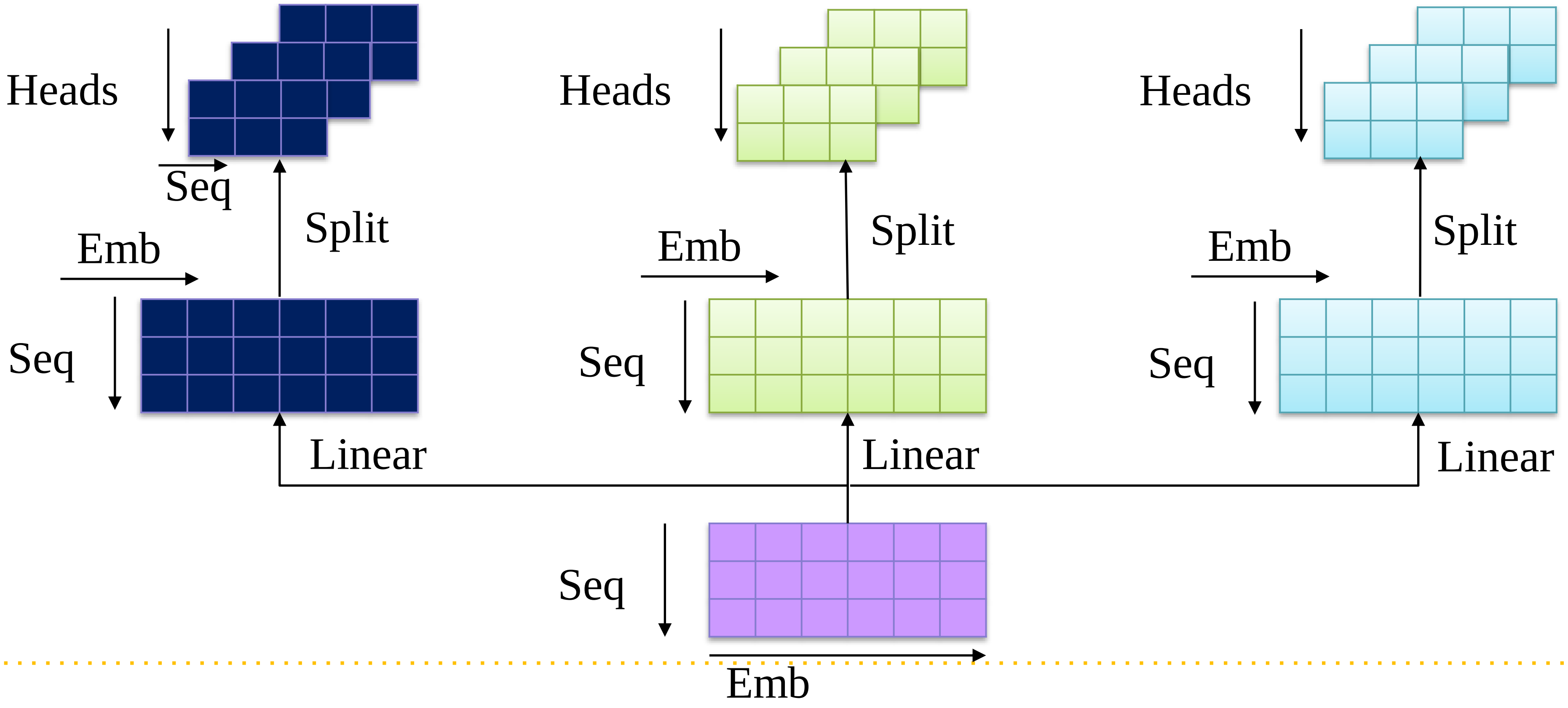
- Linear layers and split into heads



AI VIET NAM
@aivietnam.edu.vn

Multi-head attention

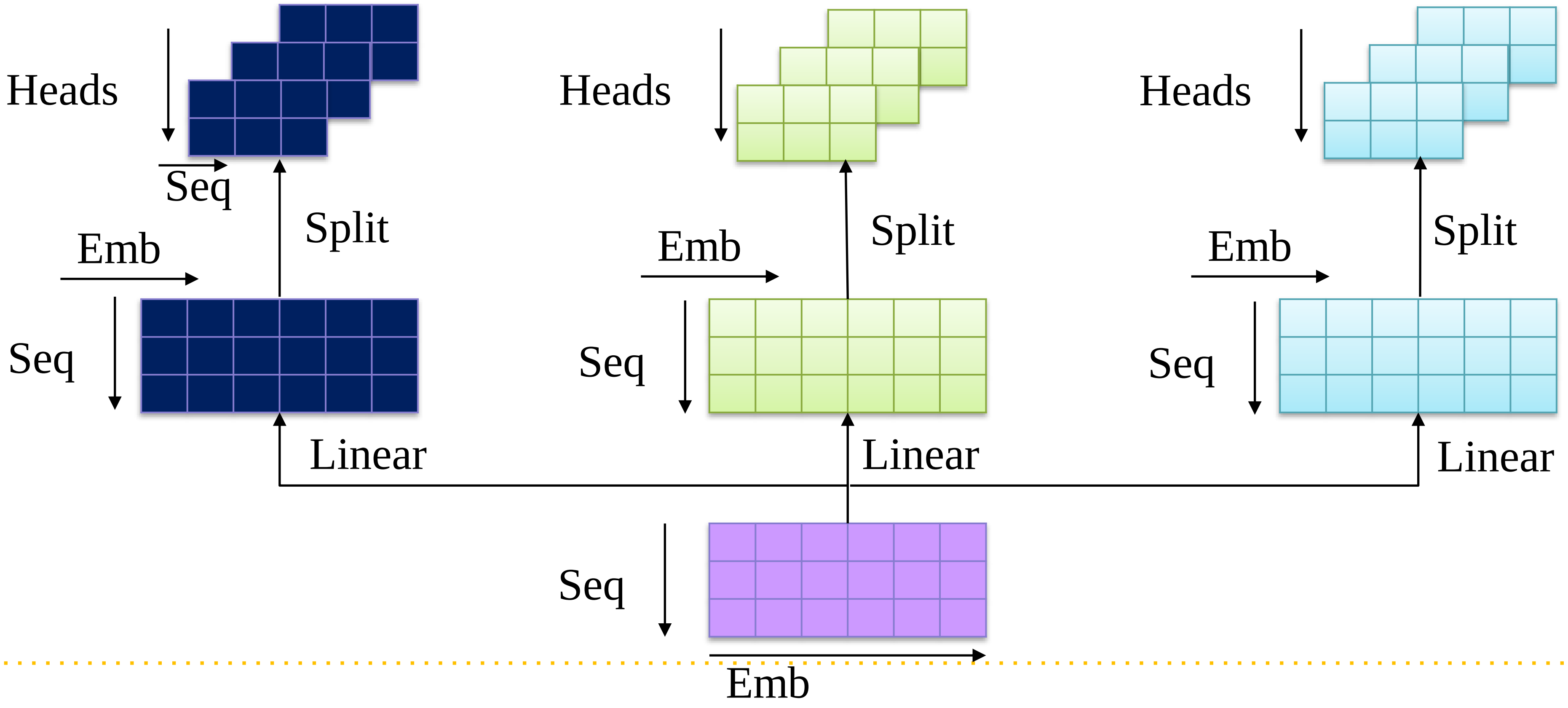
- Linear layers and split into heads



AI VIET NAM
@aivietnam.edu.vn

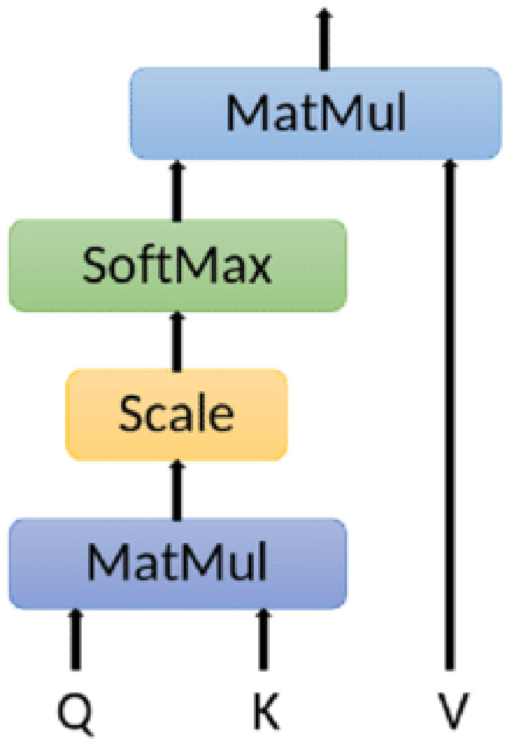
Multi-head attention

➢ Linear layers and split into heads



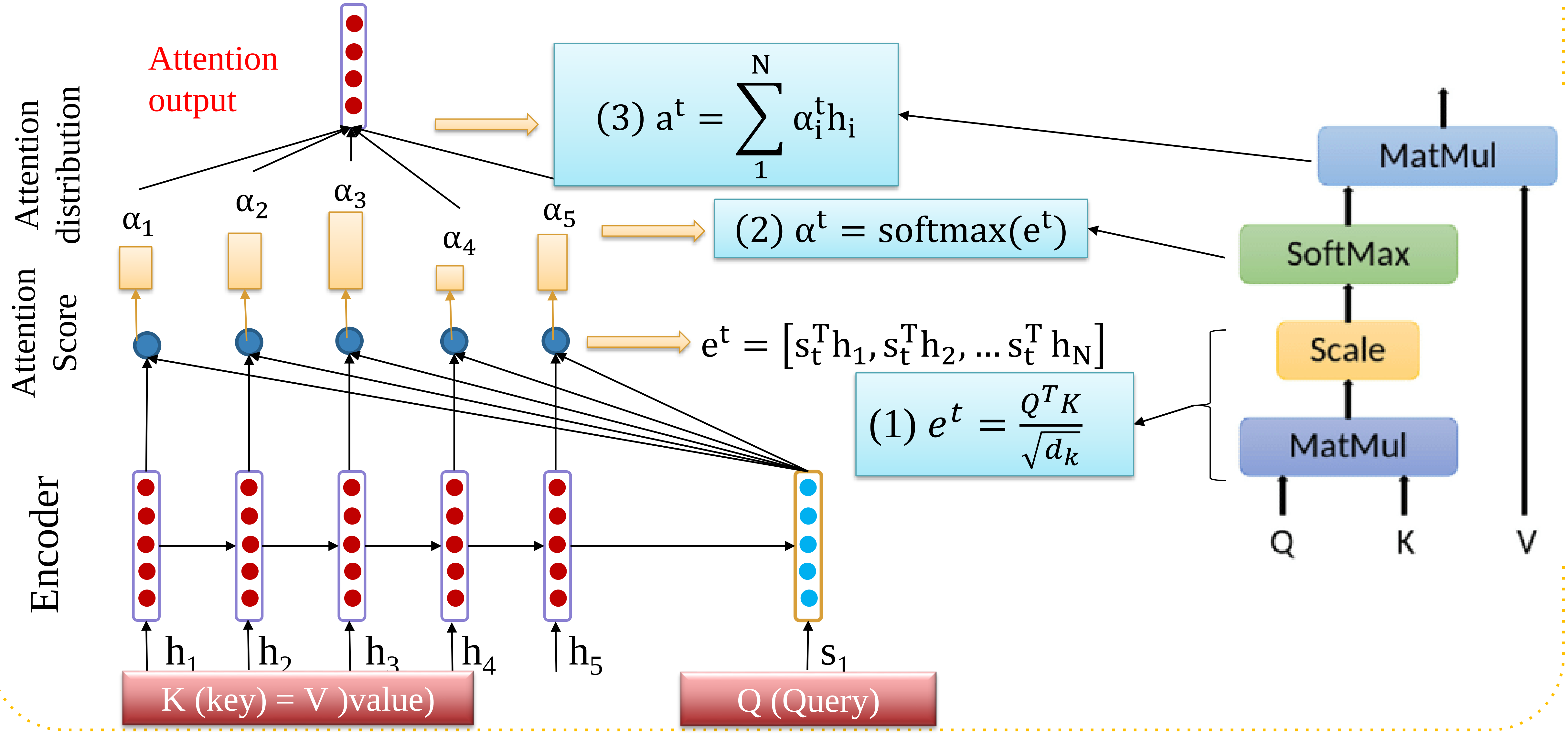
Multi-head attention

- Review scaled dot product attention
- Query (Q), Key (K), Value (V)



AI VIET NAM
@aivietnam.edu.vn

Multi-head attention



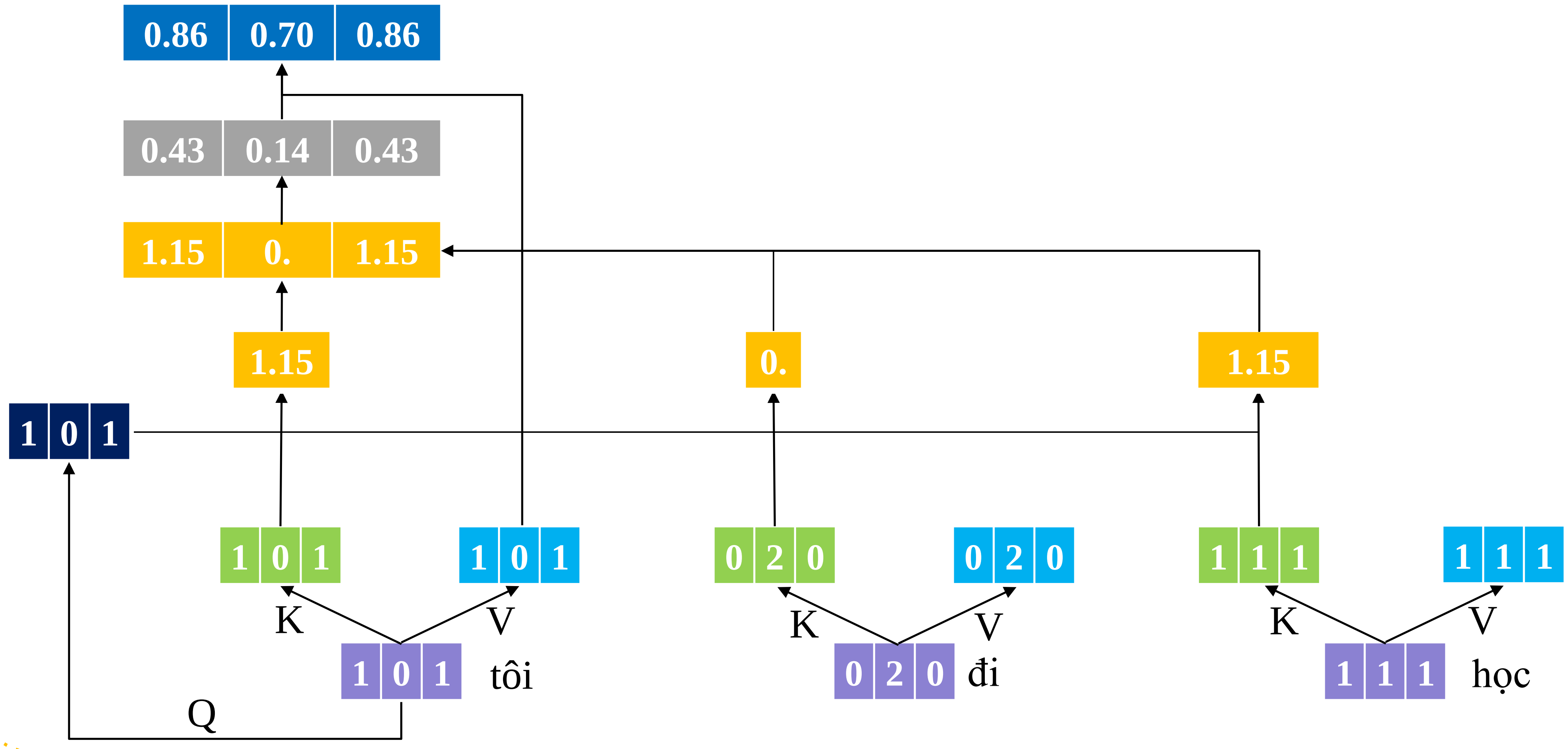
Multi-head attention

Self-attention

- Learn to represent a token in a sentence based on the surrounding tokens
- Embedding of a token as query, key and value

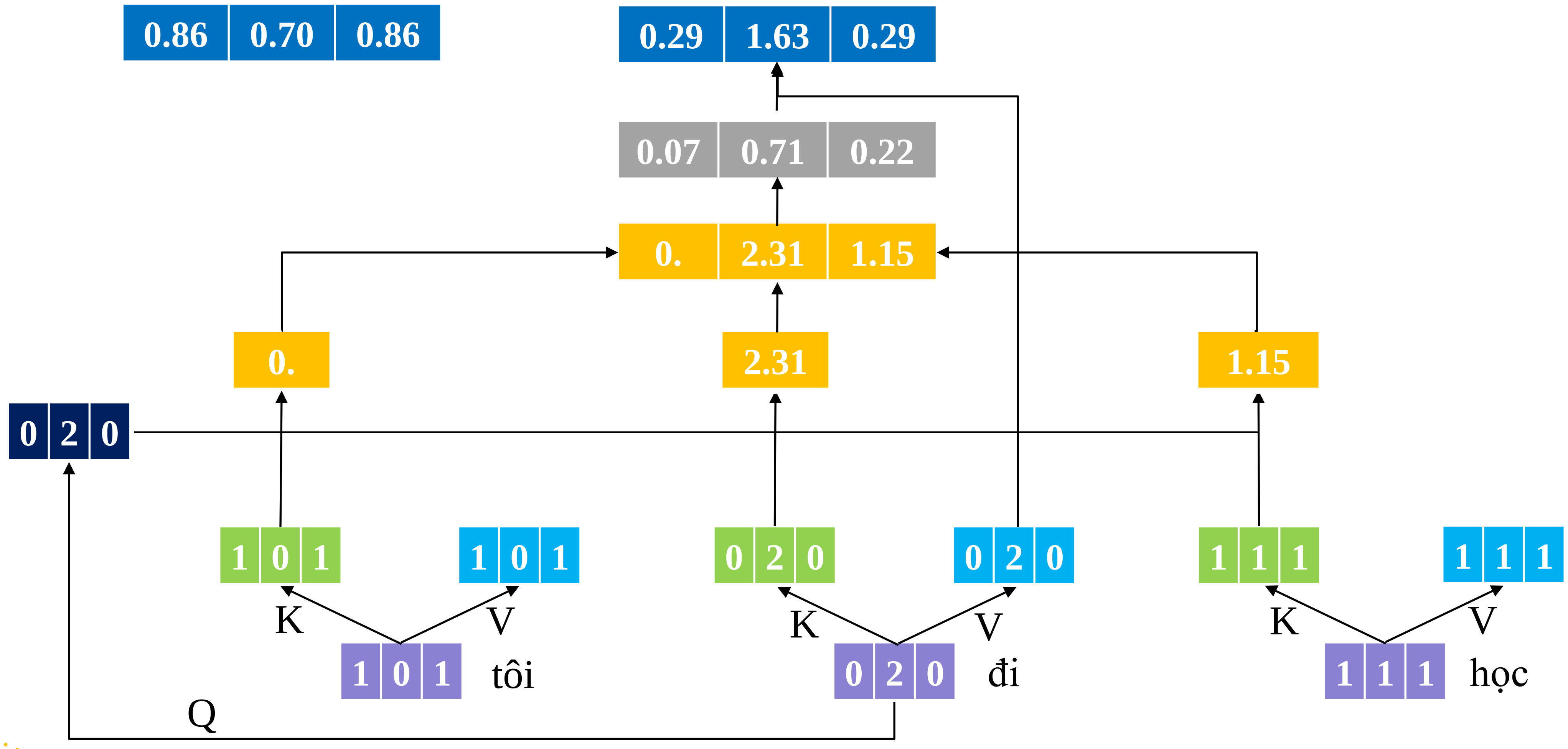
Multi-head attention

Self-attention



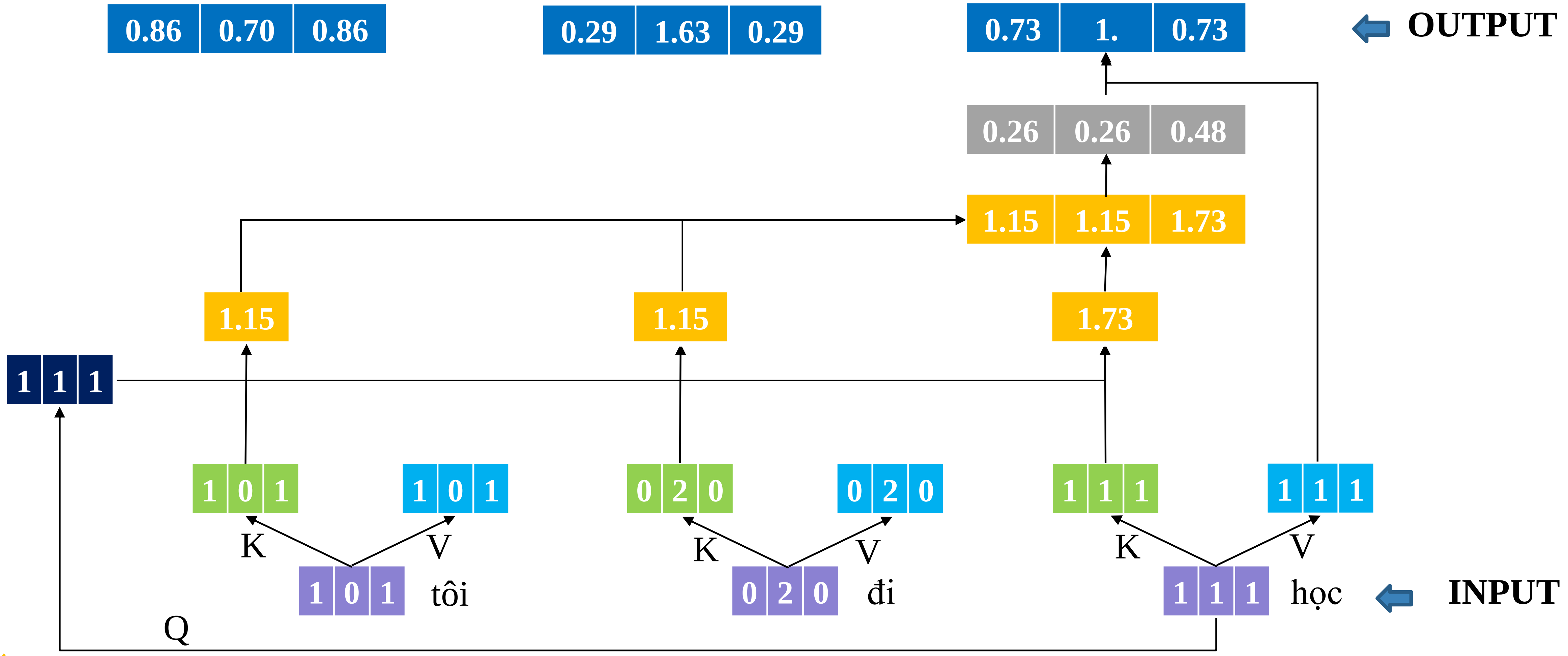
Multi-head attention

Self-attention



Multi-head attention

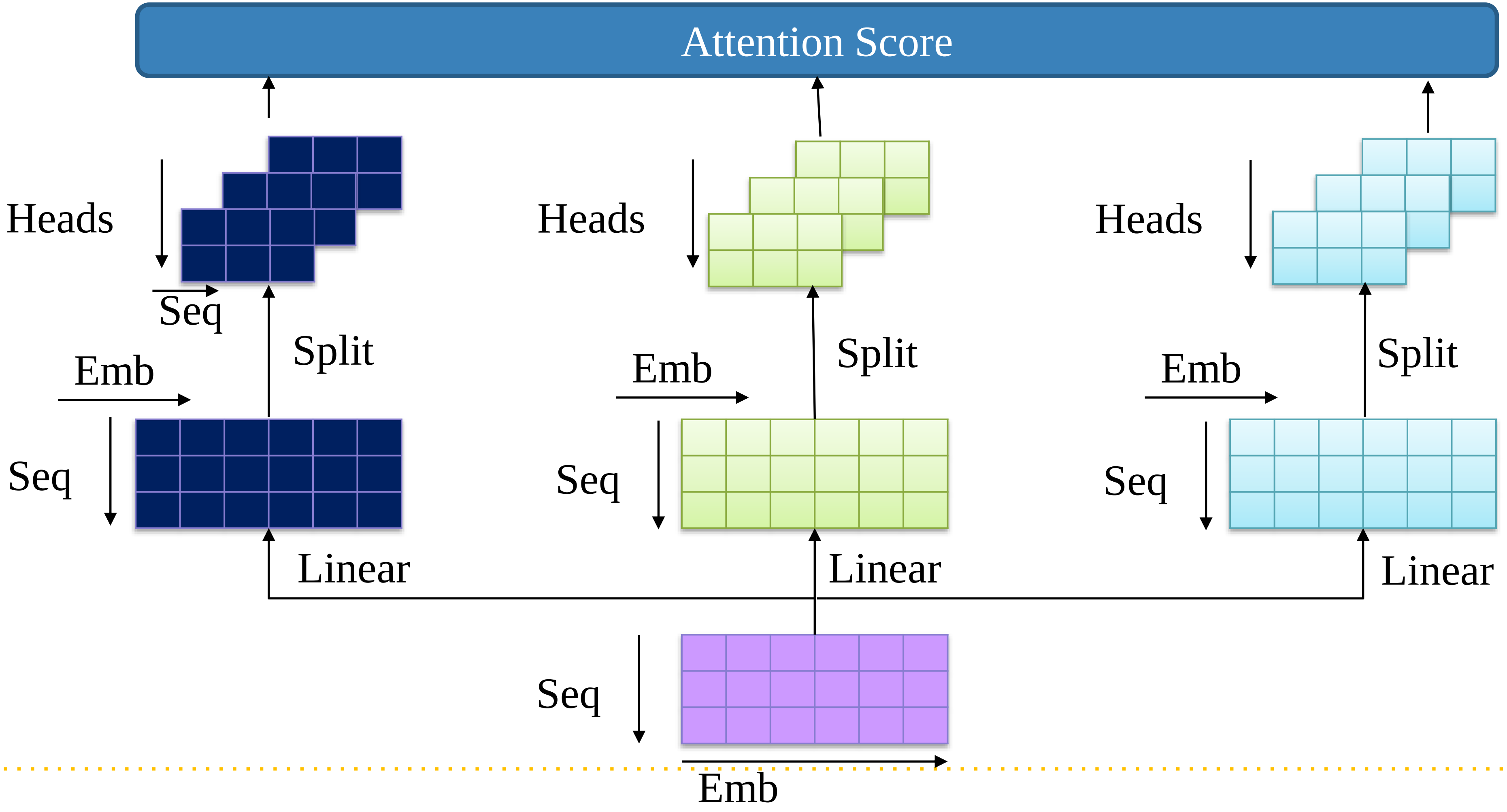
Self-attention



AI VIET NAM
@aivietnam.edu.vn

Multi-head attention

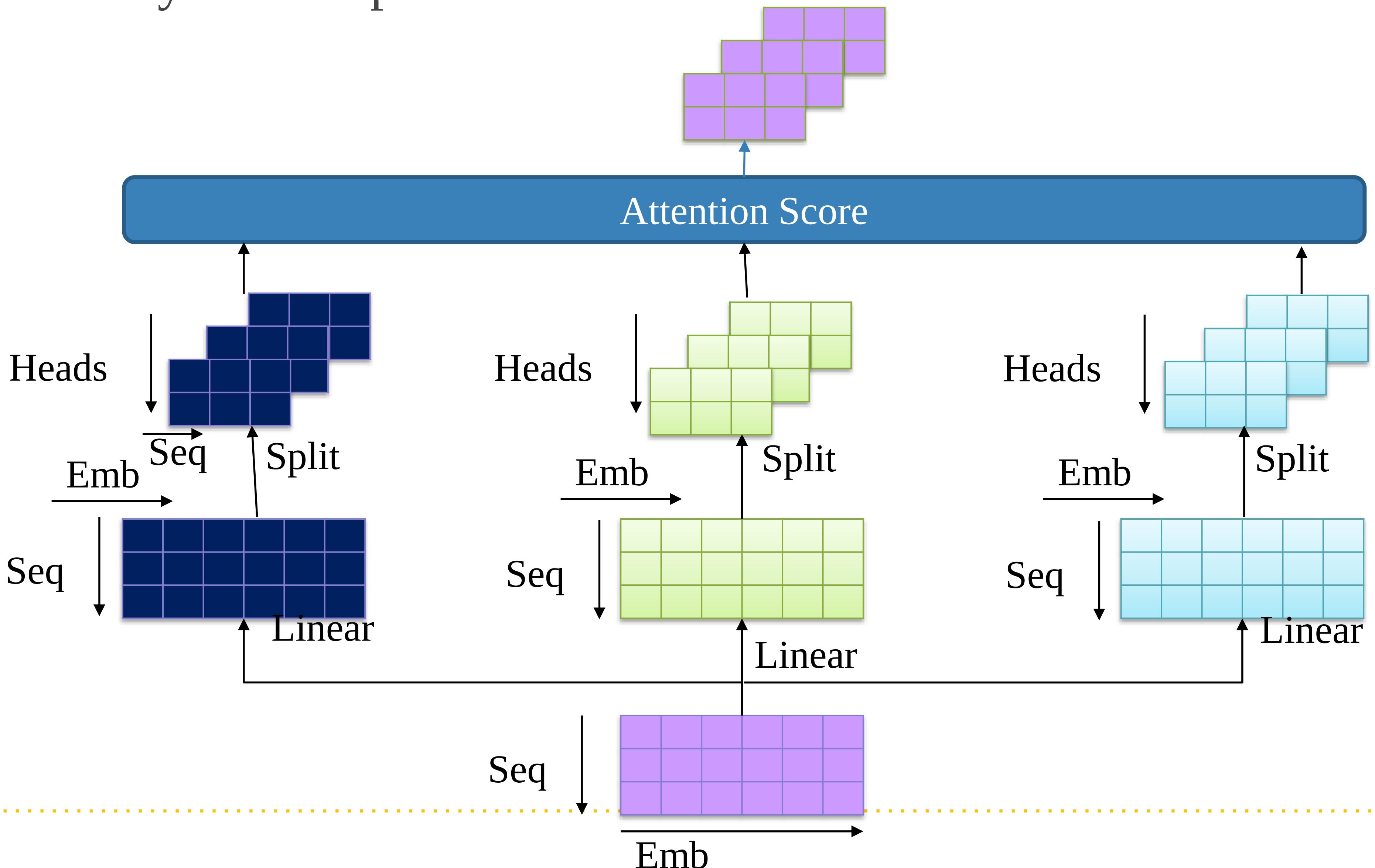
- Linear layers and split into heads => self-attention



AI VIET NAM
@aivietnam.edu.vn

Multi-head attention

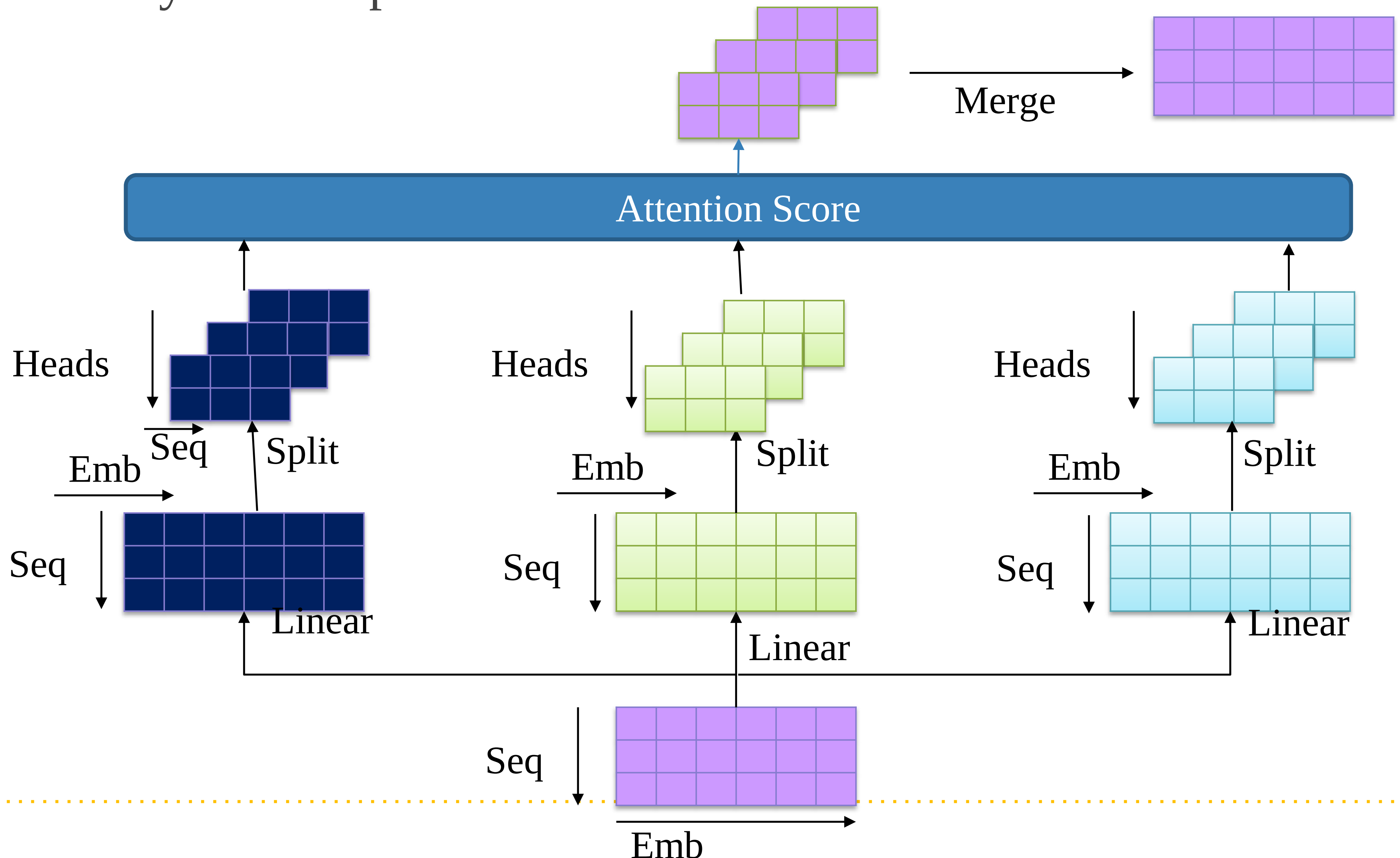
- Linear layers and split into heads => self-attention => concat



AI VIET NAM
@aivietnam.edu.vn

Multi-head attention

➢ Linear layers and split into heads => self-attention => concat



Multi-head attention

```python
class MultiHeadAttention(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads):
        super(MultiHeadAttention, self).__init__()
        self.d_model = d_model
        self.num_heads = num_heads

        assert d_model % num_heads == 0

        self.depth = d_model // self.num_heads

        self.wq = tf.keras.layers.Dense(d_model)
        self.wk = tf.keras.layers.Dense(d_model)
        self.wv = tf.keras.layers.Dense(d_model)
        self.dense = tf.keras.layers.Dense(d_model)

    def scaled_dot_product_attention(self, q, k, v, mask=None):
        dk = tf.cast(tf.shape(k)[-1], tf.float32)
        attention_scores = tf.matmul(q, k, transpose_b=True) / tf.math.sqrt(dk)

        if mask is not None:
            attention_scores += (mask * -1e9)

        attention_weights = tf.nn.softmax(attention_scores, axis=-1)

        output = tf.matmul(attention_weights, v)

        return output, attention_weights
```

Multi-head attention

```python
def split_heads(self, x, batch_size):
    x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth)) #depth = d_model // num_heads
    return tf.transpose(x, perm=[0, 2, 1, 3])

def call(self, q, k, v, mask=None):
    batch_size = tf.shape(q)[0]

    qw = self.wq(q)  # (batch_size, seq_len, d_model)
    kw = self.wk(k)  # (batch_size, seq_len, d_model)
    vw = self.wv(v)  # (batch_size, seq_len, d_model)

    heads_qw = self.split_heads(qw, batch_size)  # (batch_size, num_heads, seq_len_q, depth)
    heads_kw = self.split_heads(kw, batch_size)  # (batch_size, num_heads, seq_len_k, depth)
    heads_vw = self.split_heads(vw, batch_size)  # (batch_size, num_heads, seq_len_v, depth)

    # scaled_attention.shape == (batch_size, num_heads, seq_len_q, depth)
    # attention_weights.shape == (batch_size, num_heads, seq_len_q, seq_len_k)
    scaled_attention, attention_weights = self.scaled_dot_product_attention(heads_qw, heads_kw, heads_vw, mask)

    scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])  # (batch_size, seq_len_q, num_heads, depth)

    concat_attention = tf.reshape(scaled_attention, (batch_size, -1, self.d_model))  # (batch_size, seq_len_q, d_model)

    attention_output = self.dense(concat_attention)  # (batch_size, seq_len_q, d_model)

    return attention_output, attention_weights
```

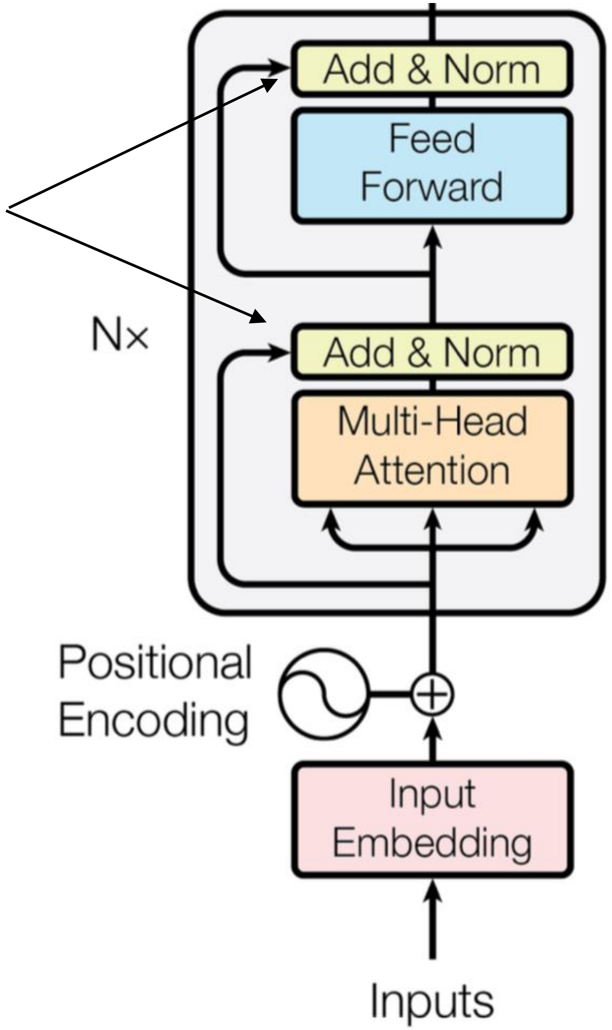
Transformer Encoder

- Layer Normalization

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_{ij}$$

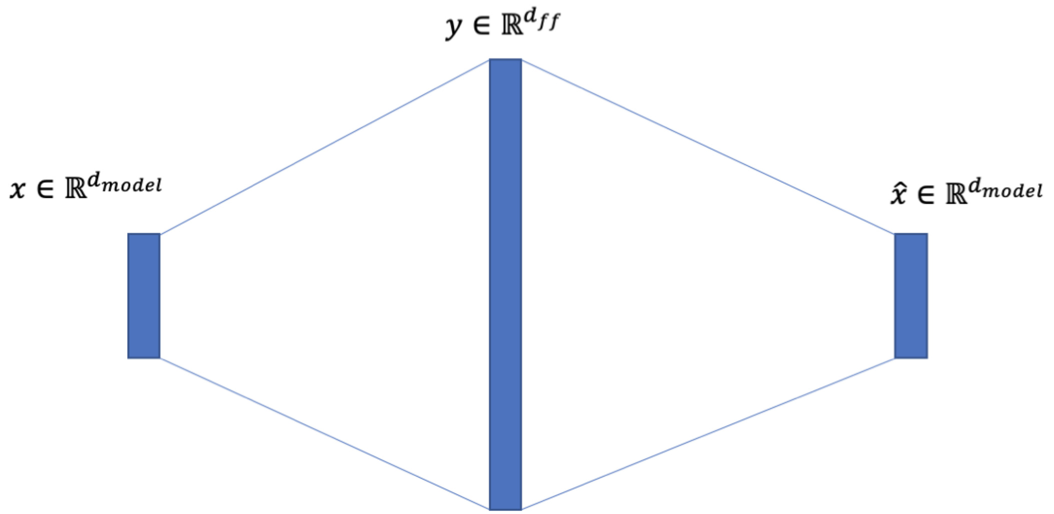
$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_{ij} - \mu_i)^2$$

$$\hat{x}_{ij} = \frac{(x_{ij} - \mu_i)}{\sqrt{\sigma_i^2 + \epsilon}}$$

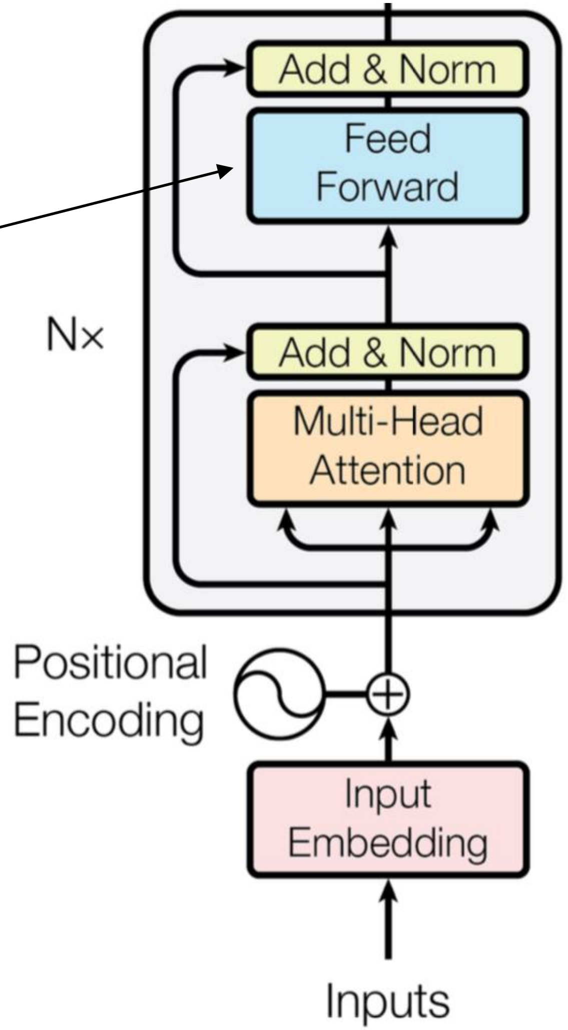


Transformer Encoder

- Point wise feed forward network



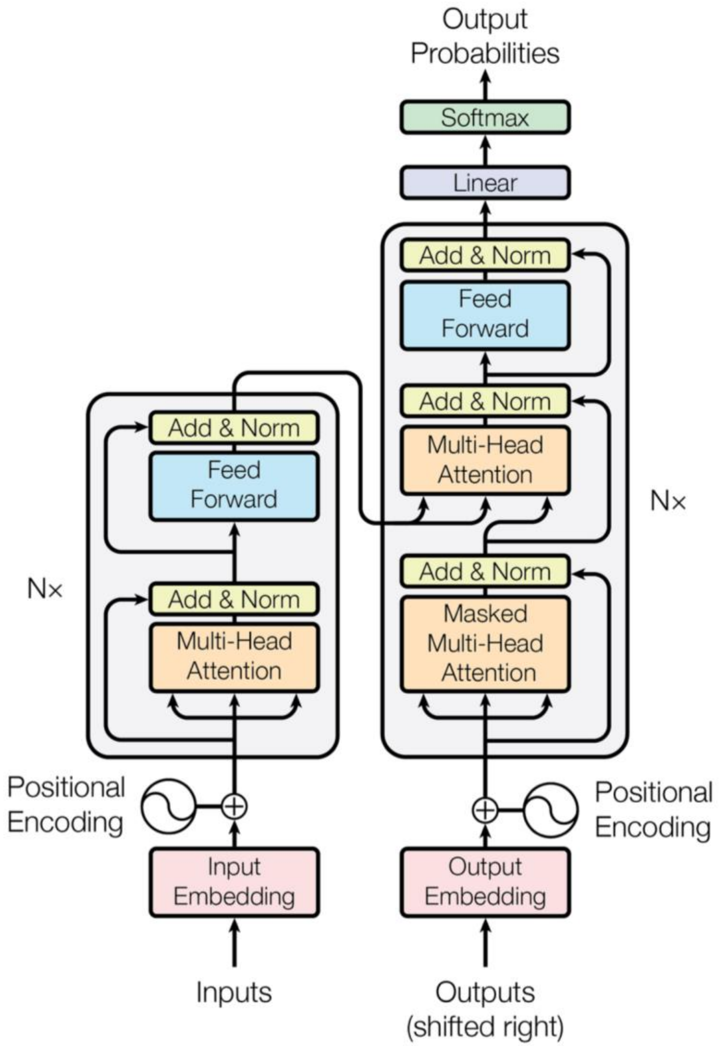

```python
class PositionWiseFeedForwardLayer(tf.keras.layers.Layer):
    def __init__(self, d_model, ffn_units):
        super(PositionWiseFeedForwardLayer, self).__init__()

        self.feed_forward = tf.keras.models.Sequential(
                [   tf.keras.layers.Dense(ffn_units, activation="relu"),
                    tf.keras.layers.Dense(d_model), ]
            )

    def call(self, input):
        return self.feed_forward(input)
```



AI VIET NAM
@aivietnam.edu.vn

Transformer Decoder

- As language model, learn to the token relationship with history token (Masked multi head attention)
- Predict next token based on the history token
- Transformer decoder:
  - Embedding
  - N decoder layer
  - Linear classifier

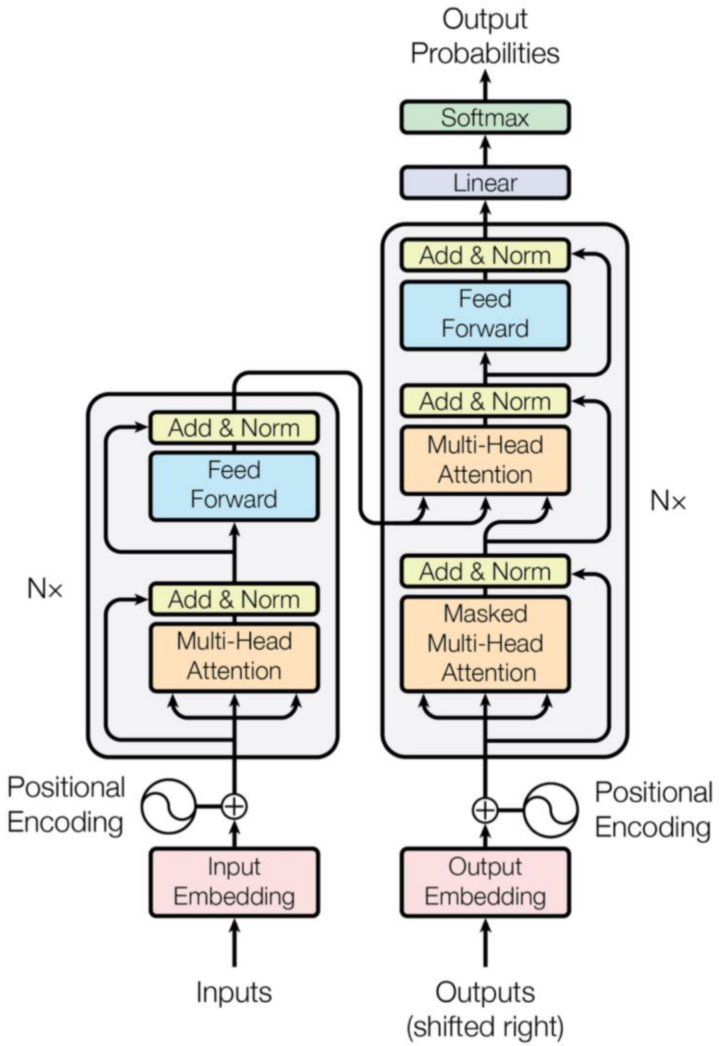


Transformer Decoder

Embedding

Token embedding

Positional encoding

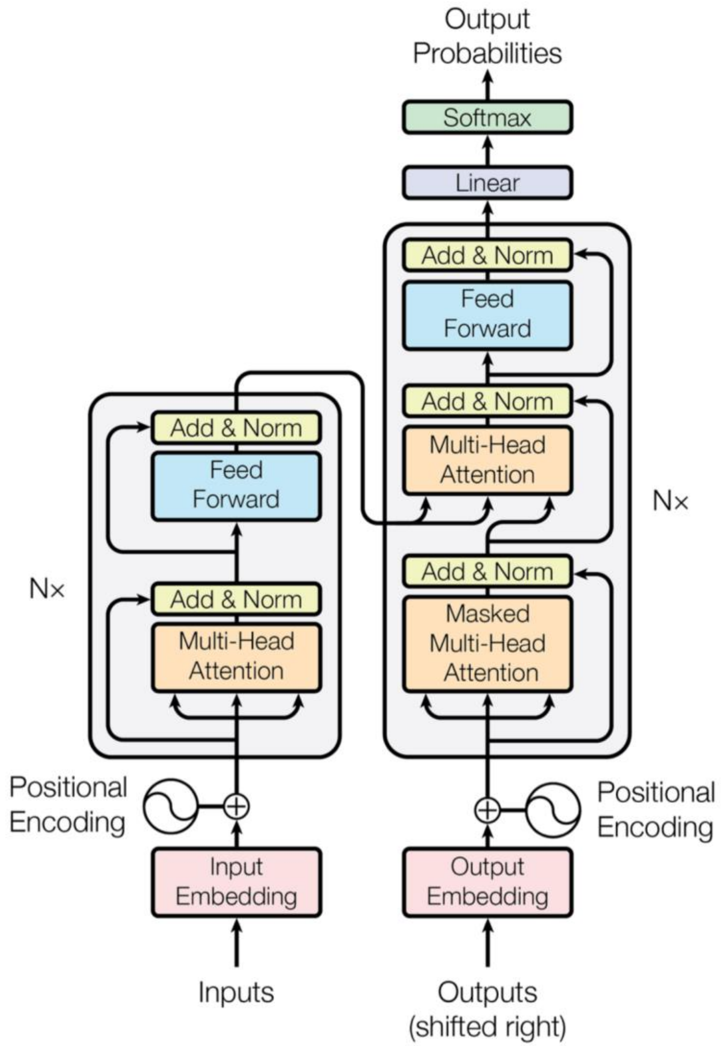


Transformer Decoder

Masked Multi-head Attention

➢ Masking the future in self-attention

	<start>	I	am	learning	English
<start>	1.2	-inf	-inf	-inf	-inf
I	2.3	1.5	-inf	-inf	-inf
am	0.5	1.4	1.6	-inf	-inf
learning	0.6	1.8	2.4	0.3	-inf
English	2.1	2.3	0.2	2.0	2.5



Transformer Decoder

Masked Multi-head Attention

➢ Masking the future in self-attention

1.2	-inf	-inf	-inf	-inf
2.3	1.5	-inf	-inf	-inf
0.5	1.4	1.6	-inf	-inf
0.6	1.8	2.4	0.3	-inf
2.1	2.3	0.2	2.0	2.5

Softmax →

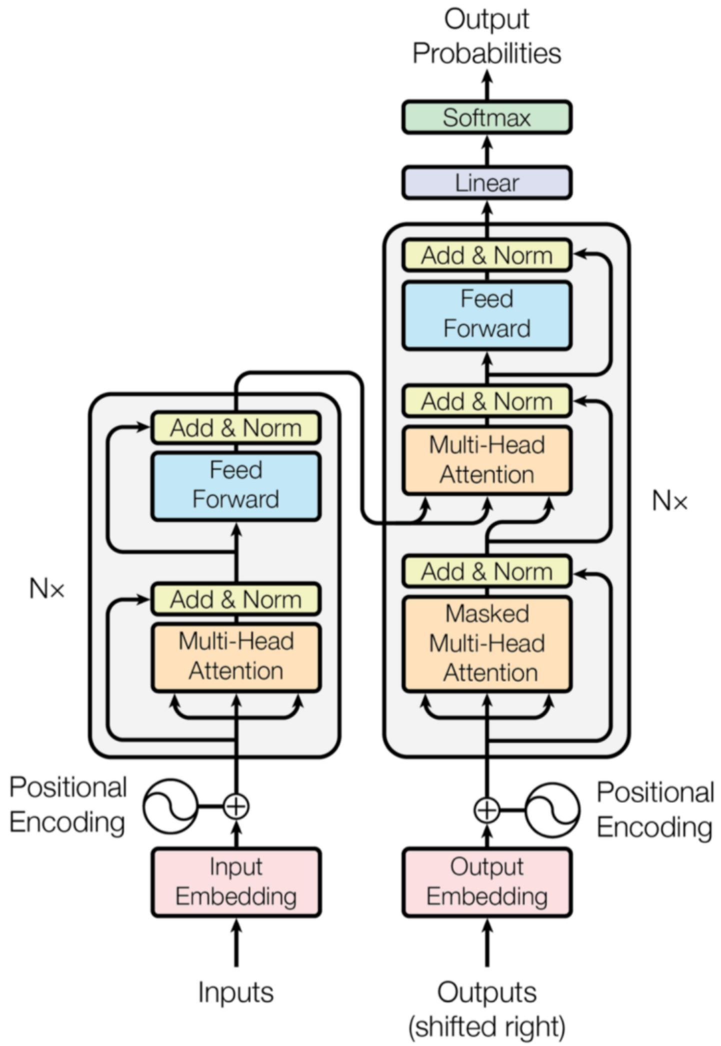
1.0	0.0	0.0	0.0	0.0
0.69	0.31	0.0	0.0	0.0
0.15	0.38	0.46	0.0	0.0
0.6	1.8	2.4	0.3	0.0
2.1	2.3	0.2	2.0	2.5

Transformer Decoder

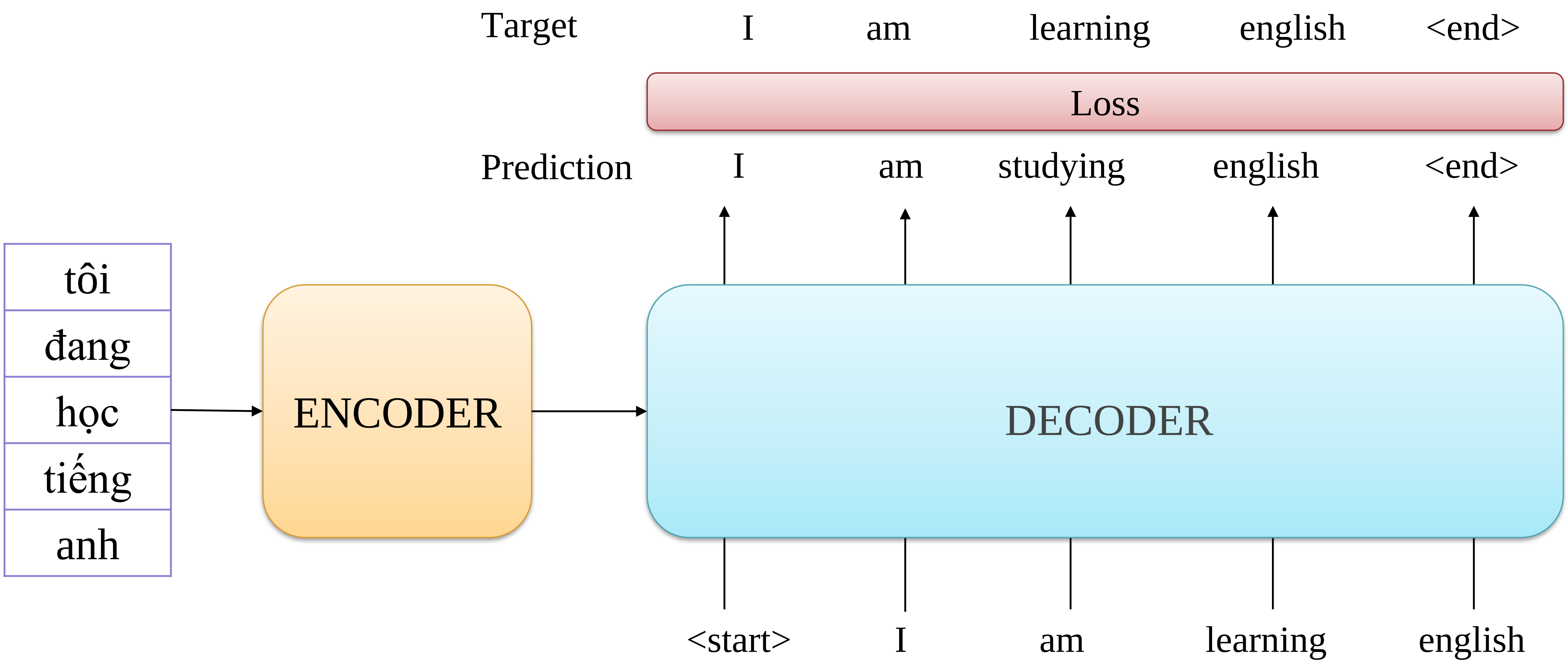
Masked Multi-head Attention

- Masking the future in self-attention

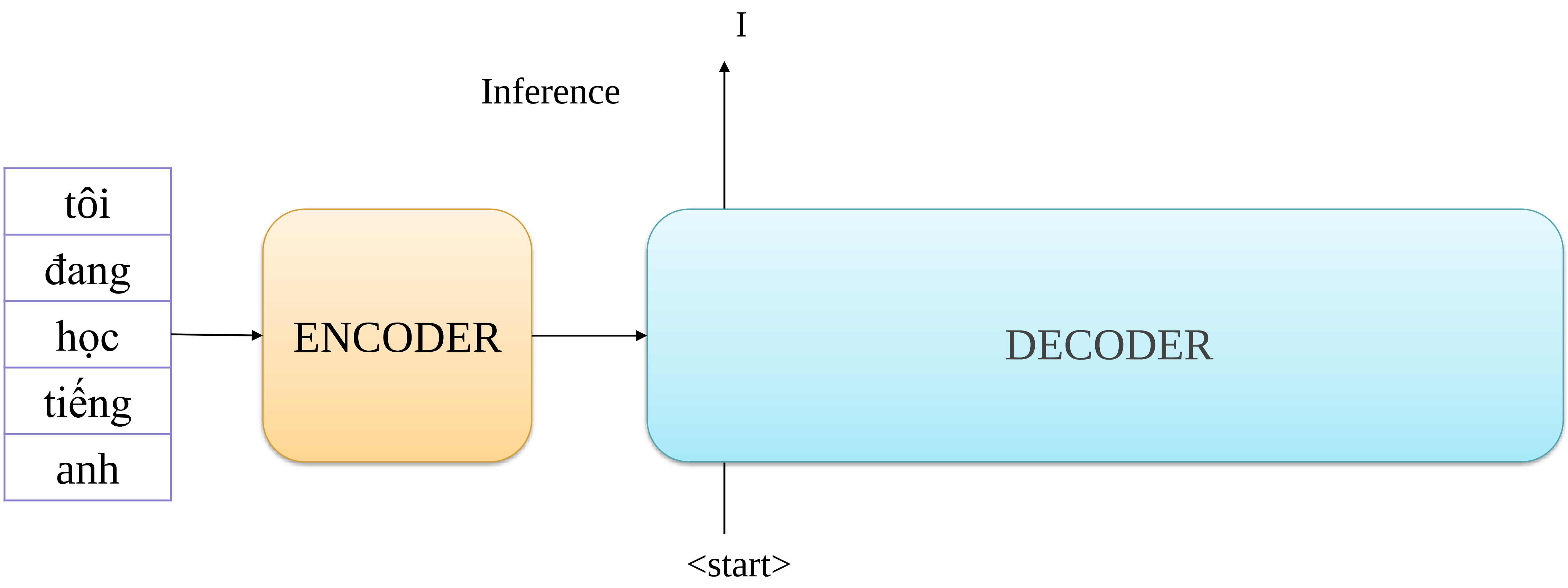
Multiple head attention, layer normalization, feed forward: the same transformer encoder



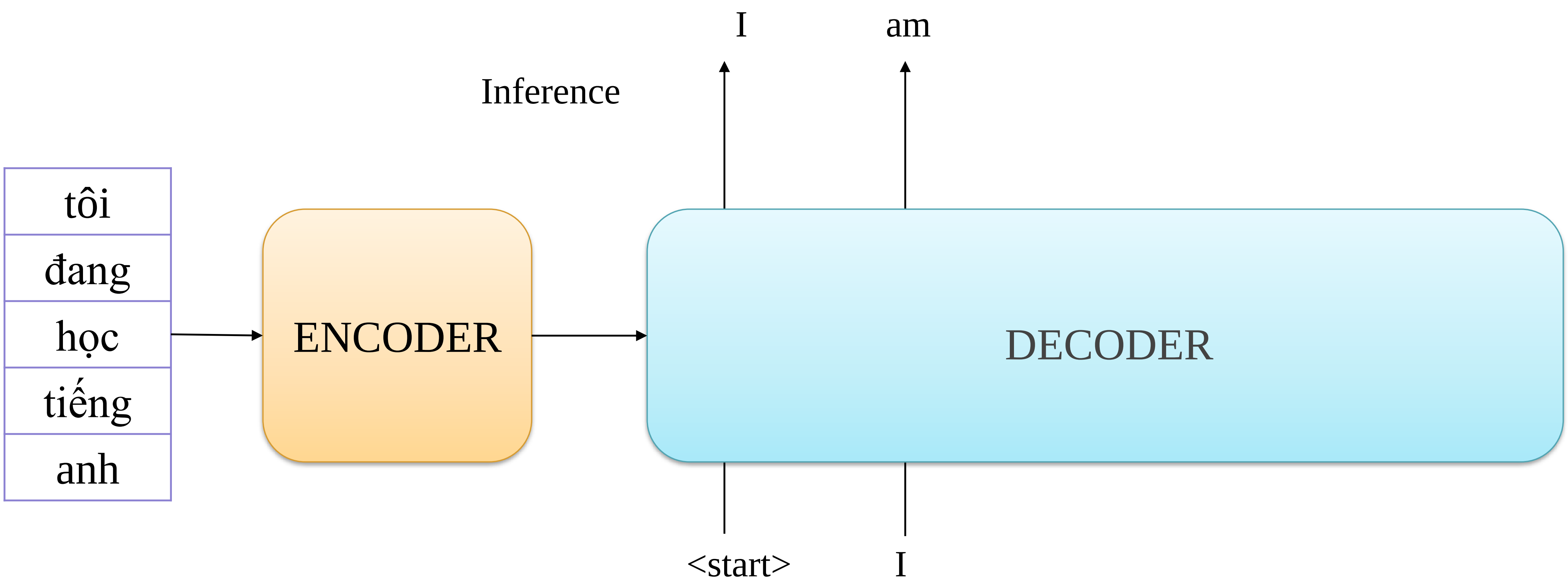
Transformer Decoder Training



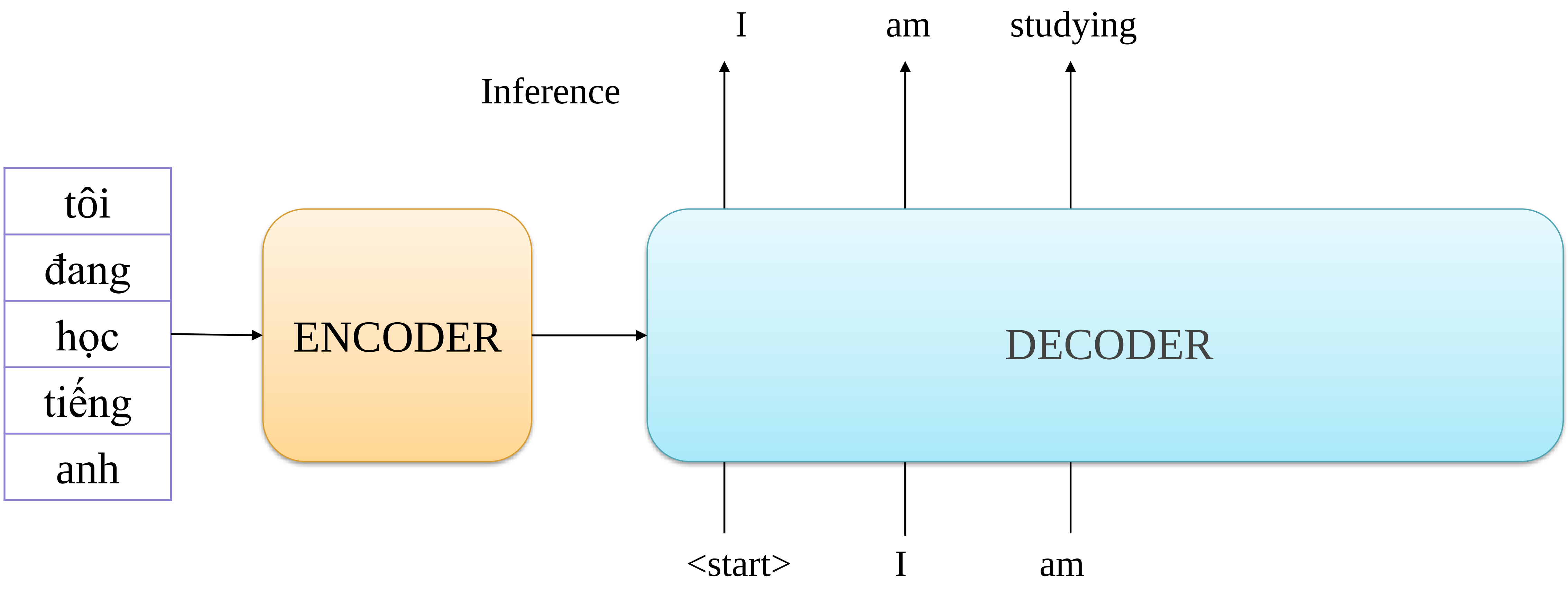
Transformer Decoder Inference



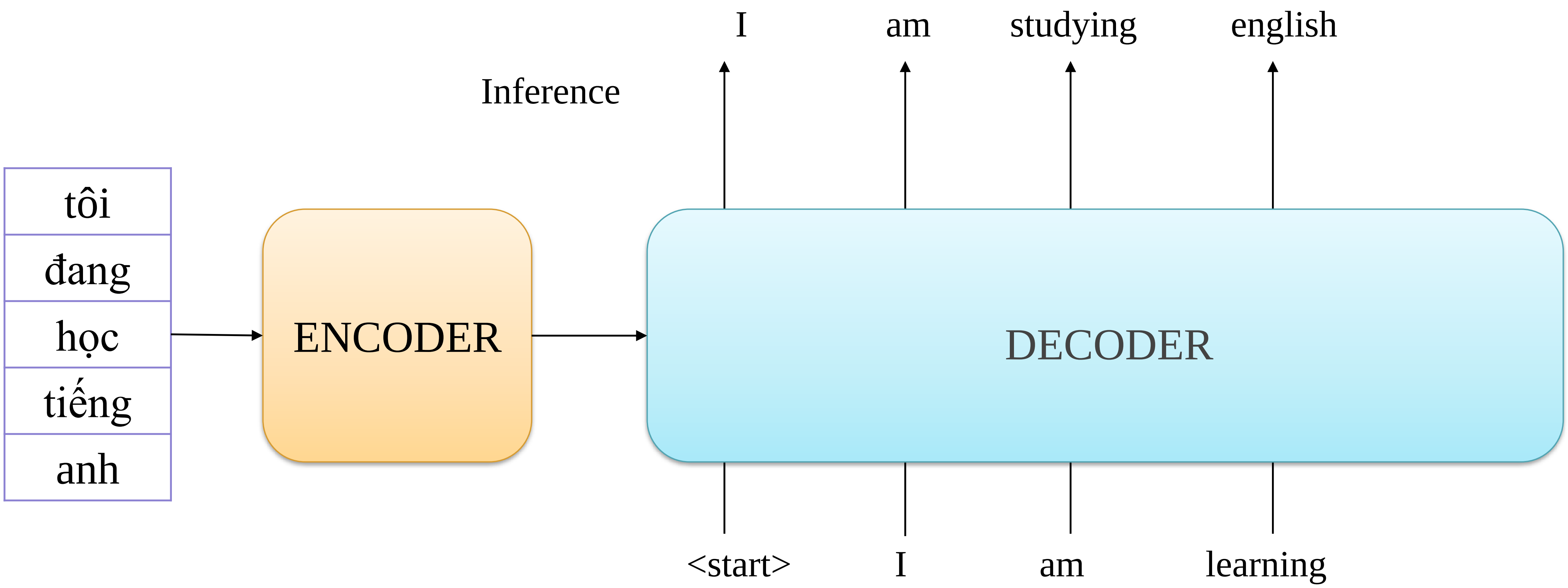
Transformer Decoder Inference



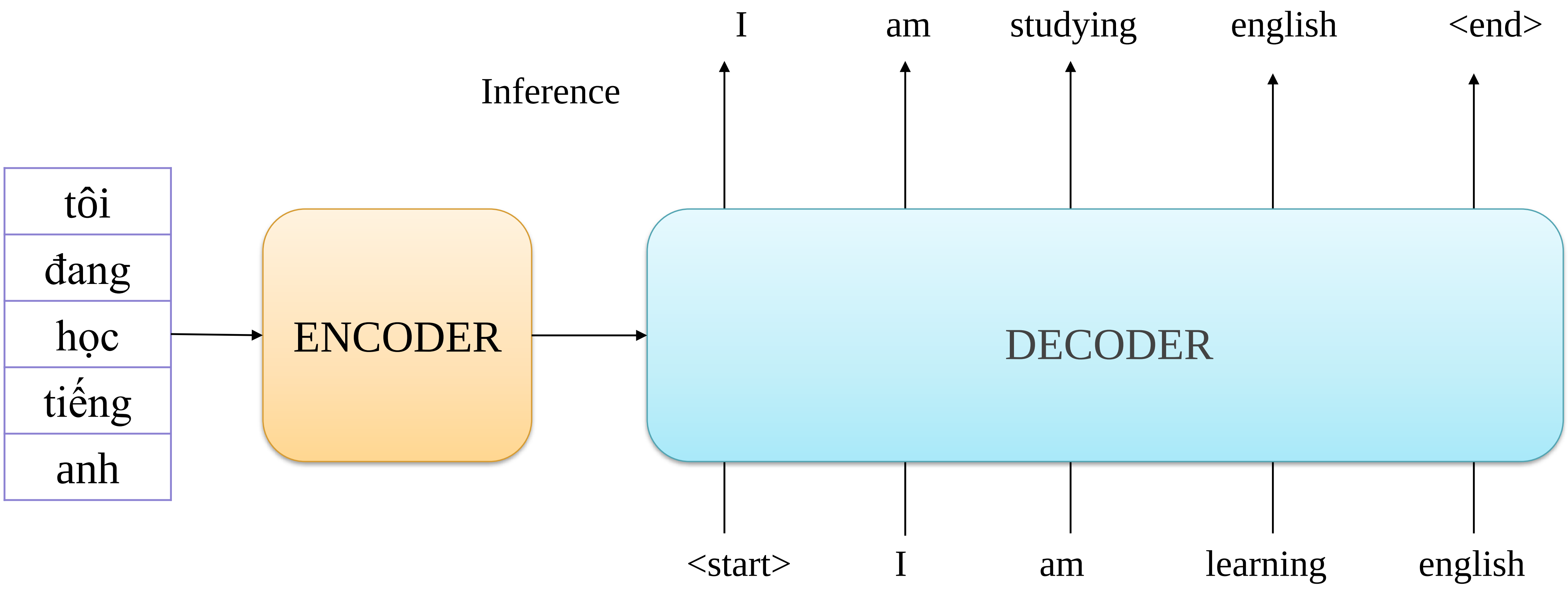
Transformer Decoder Inference



Transformer Decoder Inference



Transformer Decoder Inference



Thanks!

Any questions?