AI VIETNAM
@aivietnam.edu.vn

Module 07 – Extra Class

Introduction to Transformer

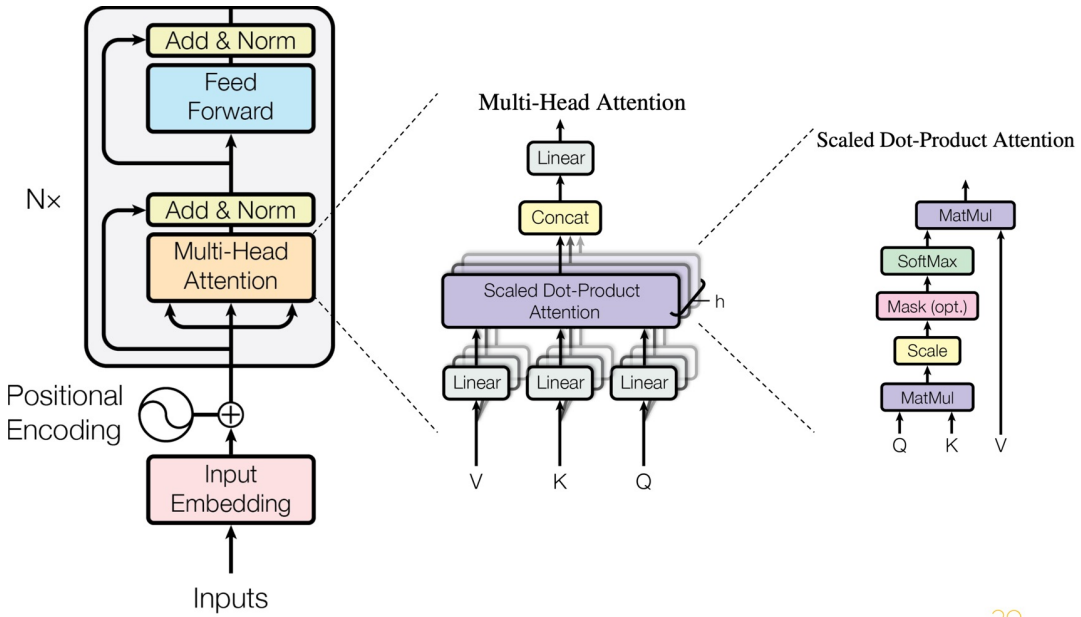
Code - Data Quiz-Feedback

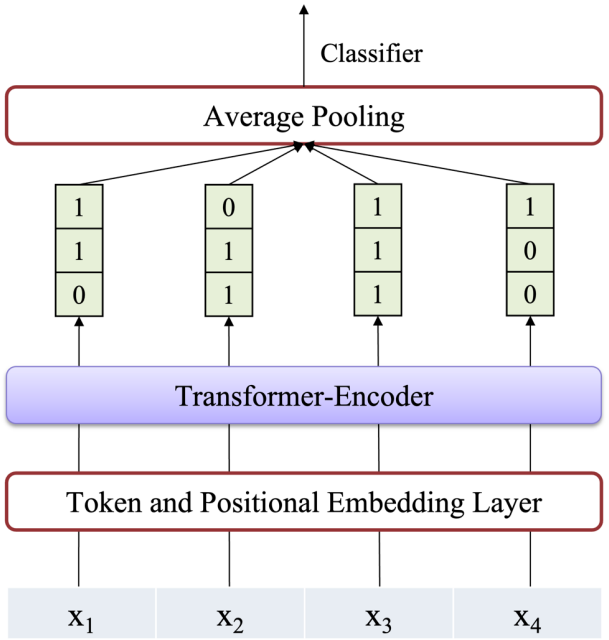
MSc. Nguyen Quoc Thai

Objectives

Transformer-Encoder

- Introduction
- Attention Mechanism
- Transformer Encoder





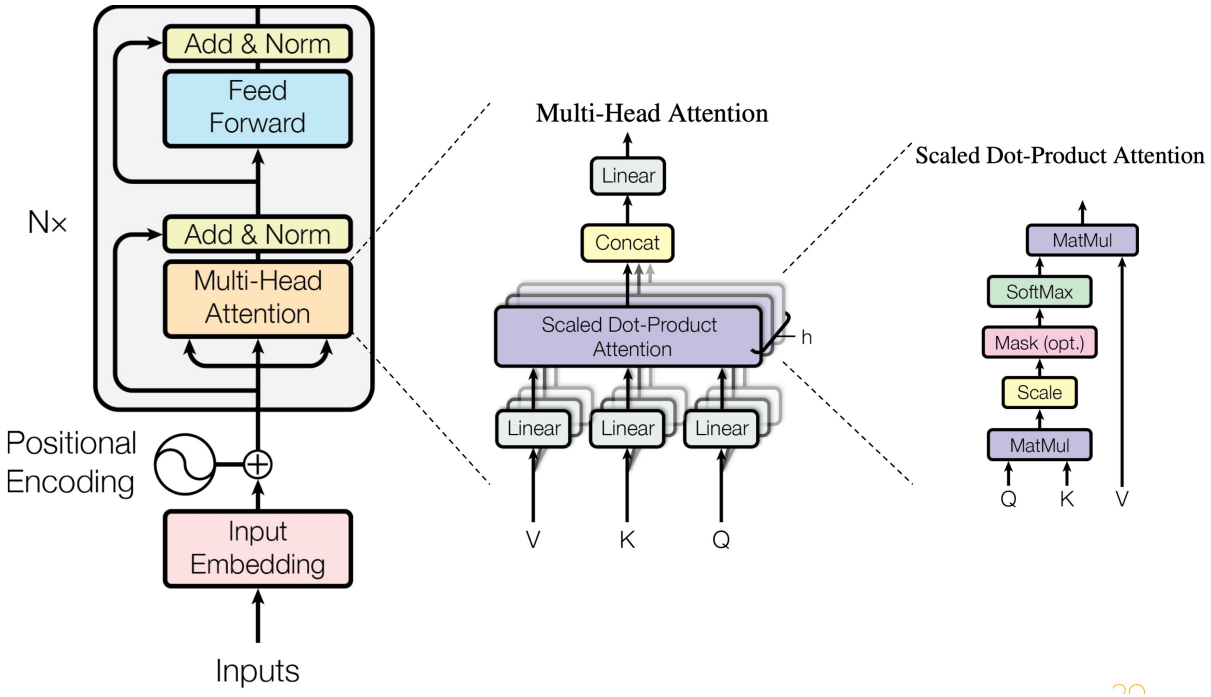
Text Classification

- Text Classification Problem
- Transformer Encoder for Text Classification Application
- Question

Outline

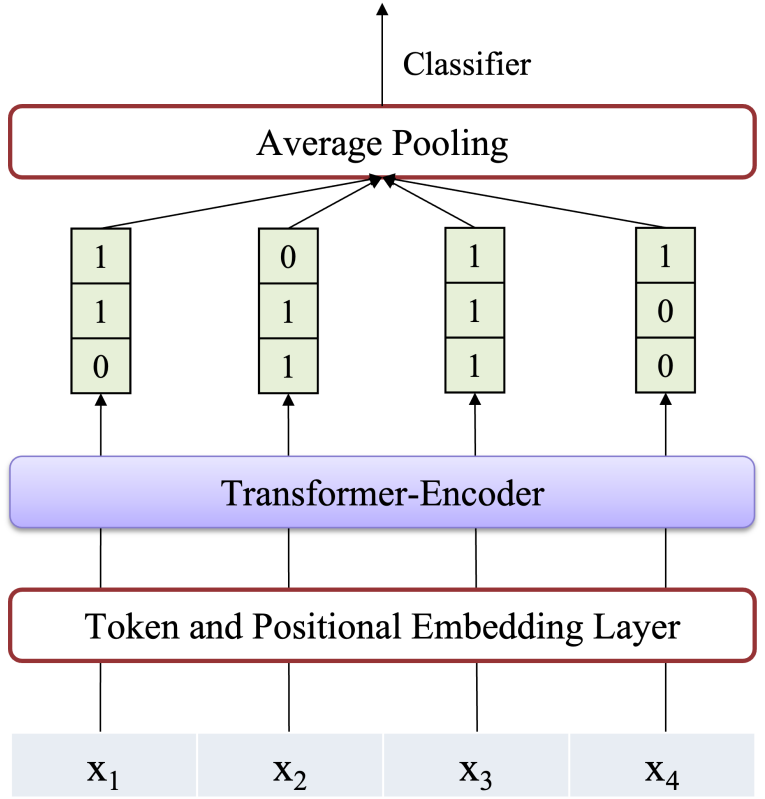
SECTION 1

Transformer-Encoder



SECTION 2

Text Classification



Attention Mechanism

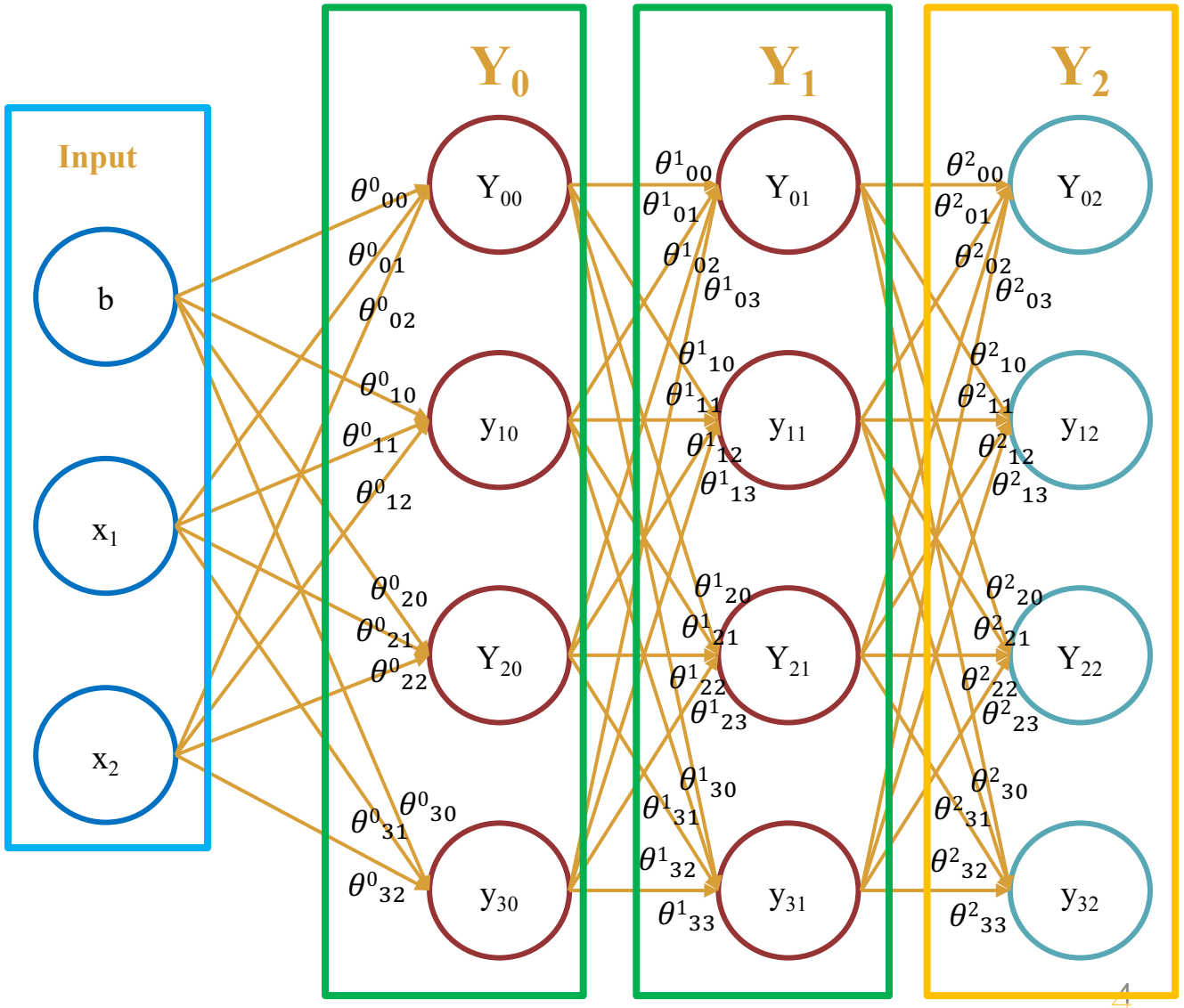
❖ Multilayer Perceptron

Input layer

Hidden layer

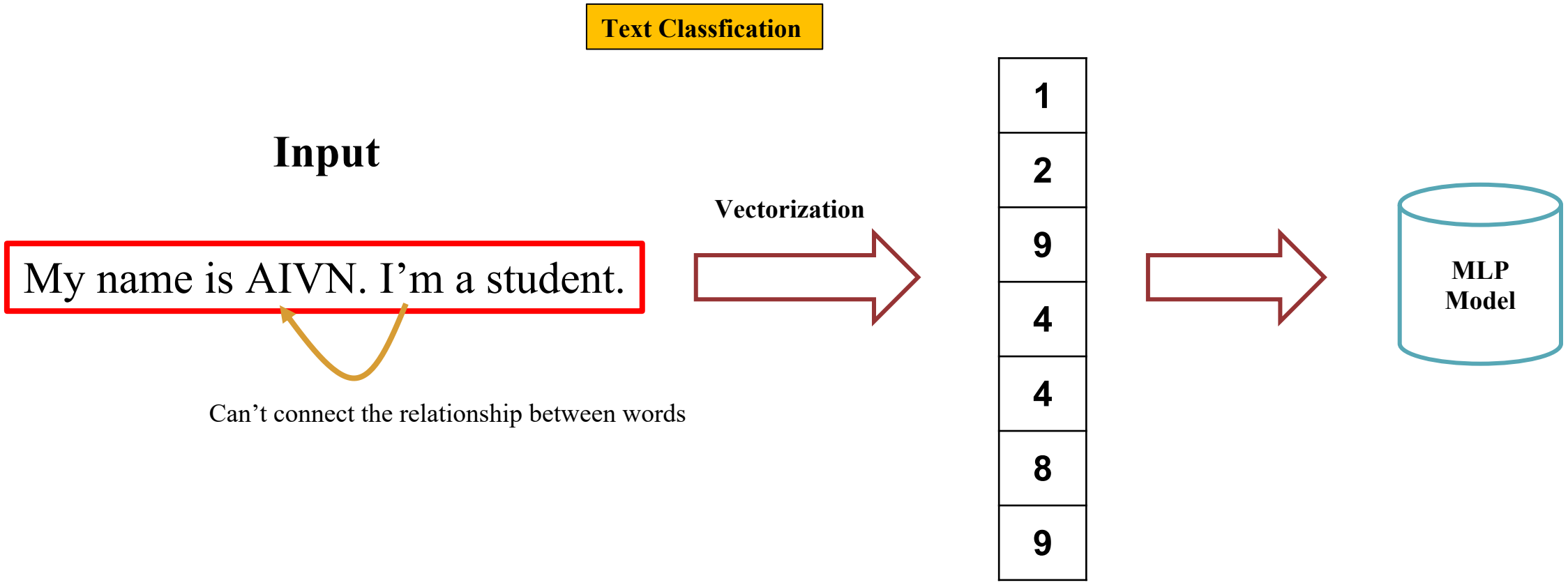
Output layer

Multilayer Perceptron



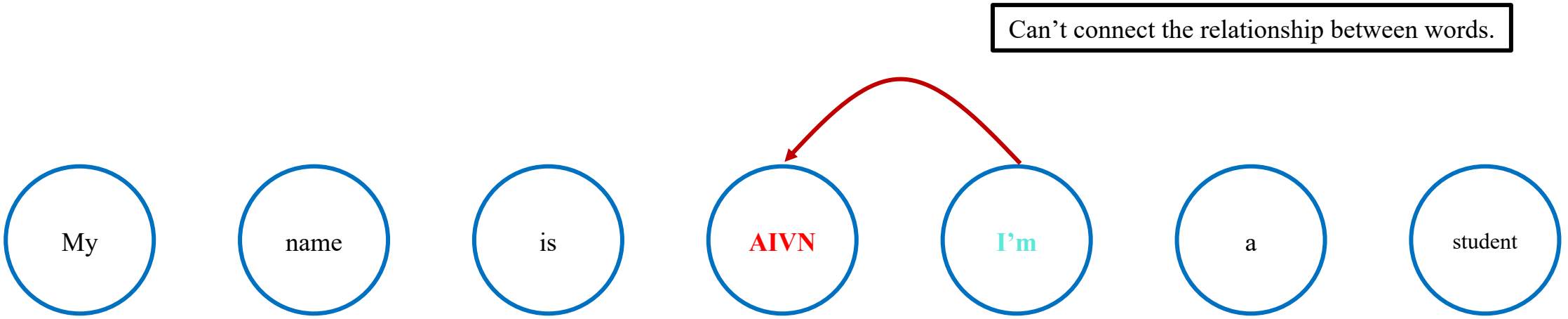
Attention Mechanism

❖ Text Classification Problem



Attention Mechanism

❖ Basic Approach Problem

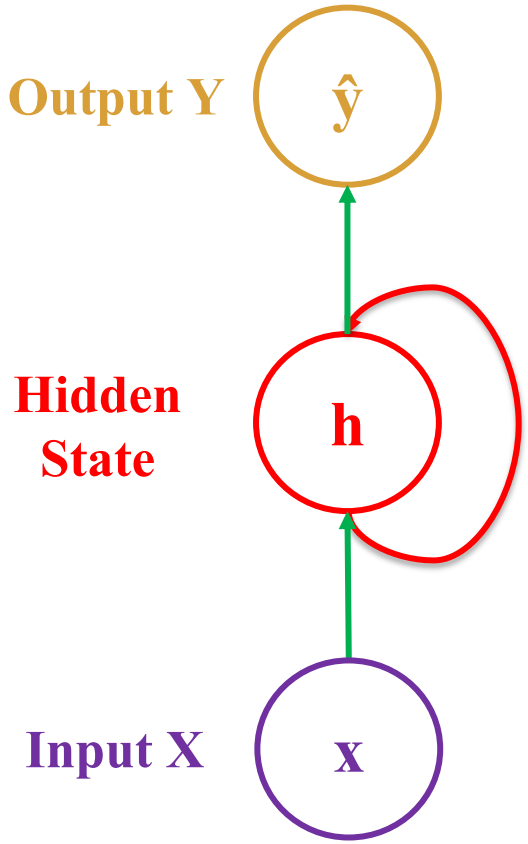


We want a vector that could represent all properties of a word.

Recurrent Neural Network.

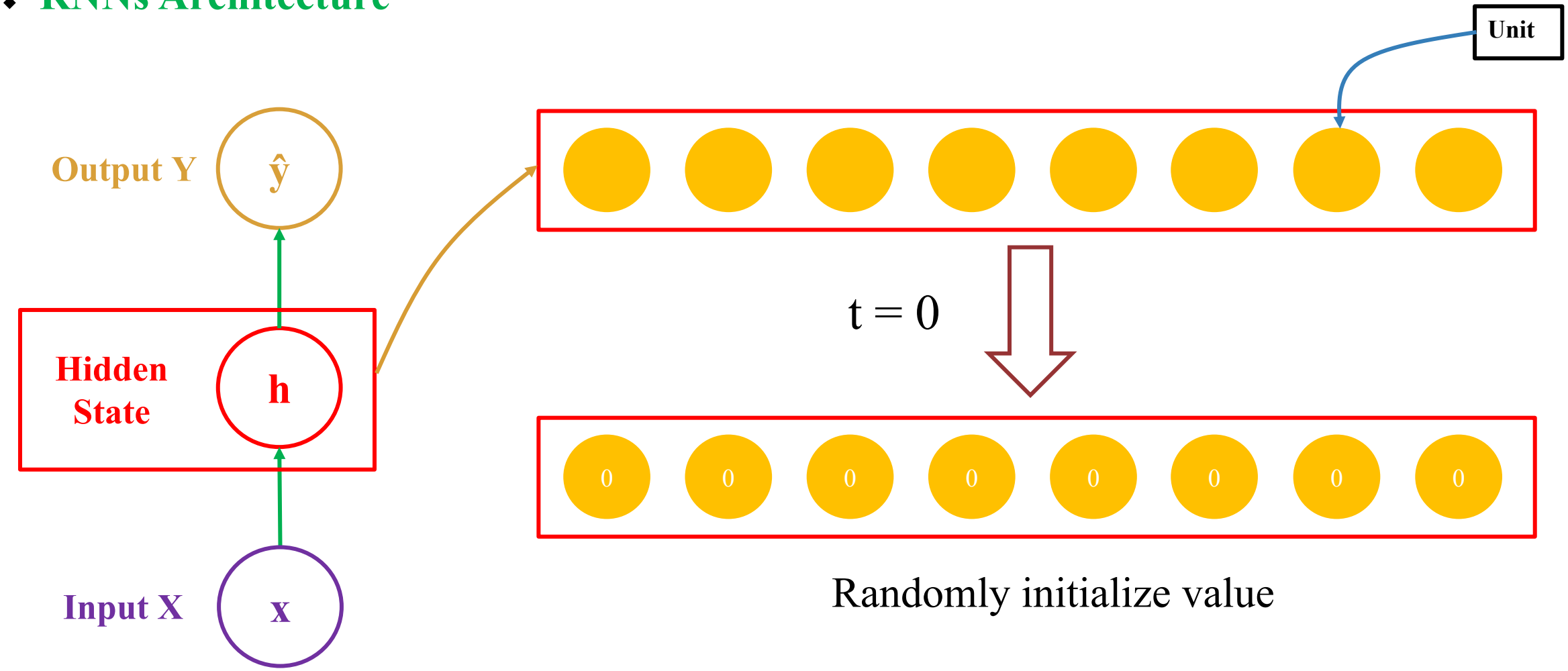
Attention Mechanism

❖ Recurrent Neural Networks (RNNs)



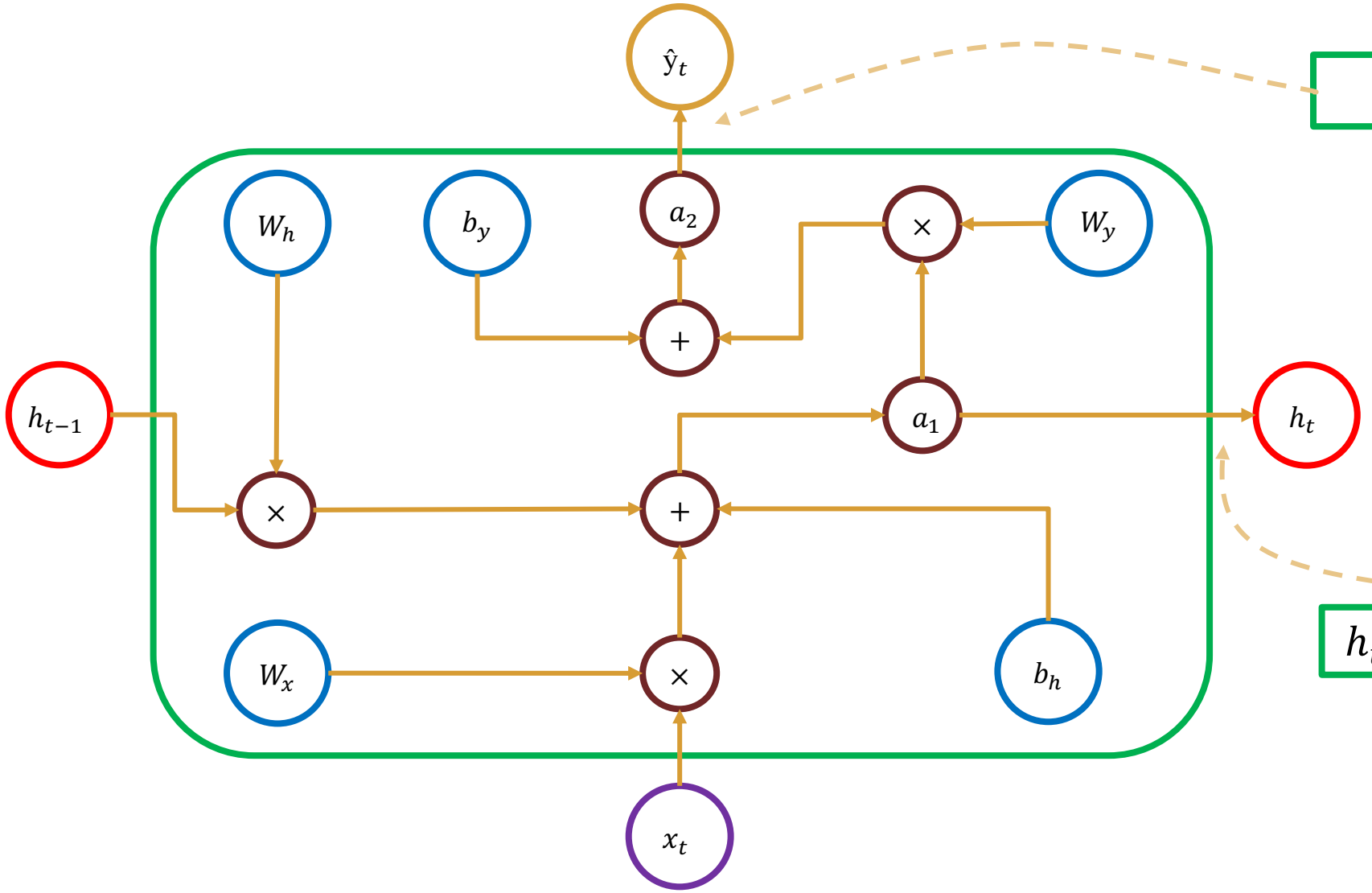
Attention Mechanism

❖ RNNs Architecture



Attention Mechanism

❖ RNNs Architecture



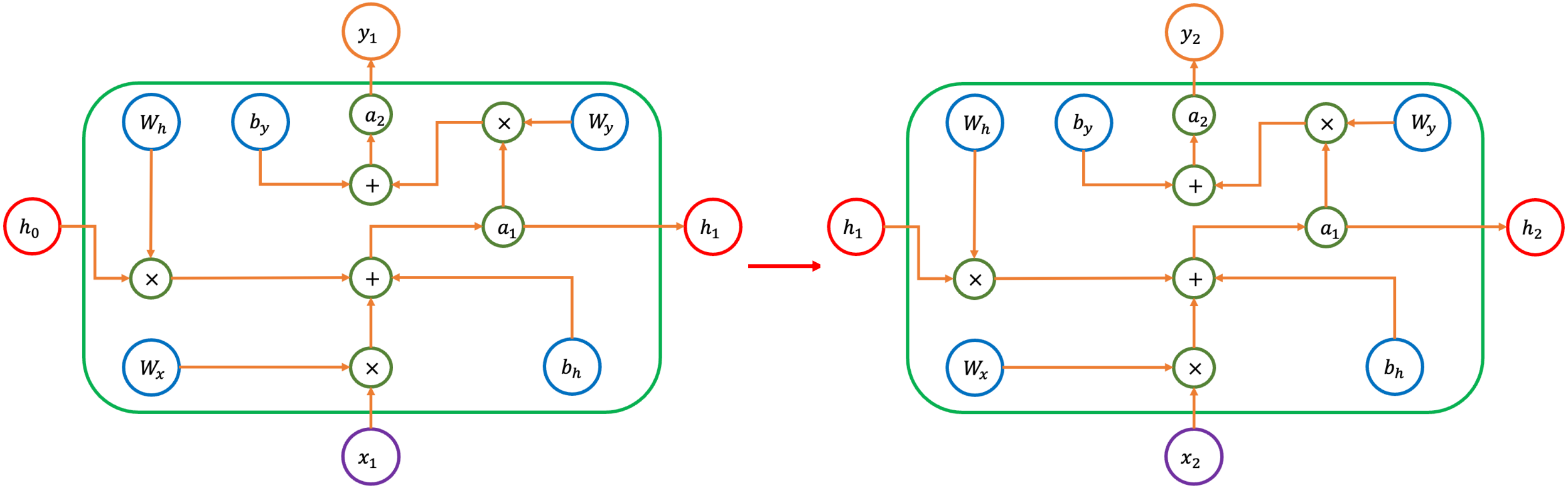
$$\hat{y}_t = a_2(h_t W_y + b_y)$$

Inside hidden state

$$h_t = a_1(h_{t-1} W_h + x_t W_x + b_h)$$

Attention Mechanism

❖ RNNs Architecture (from t=1 to t=2)

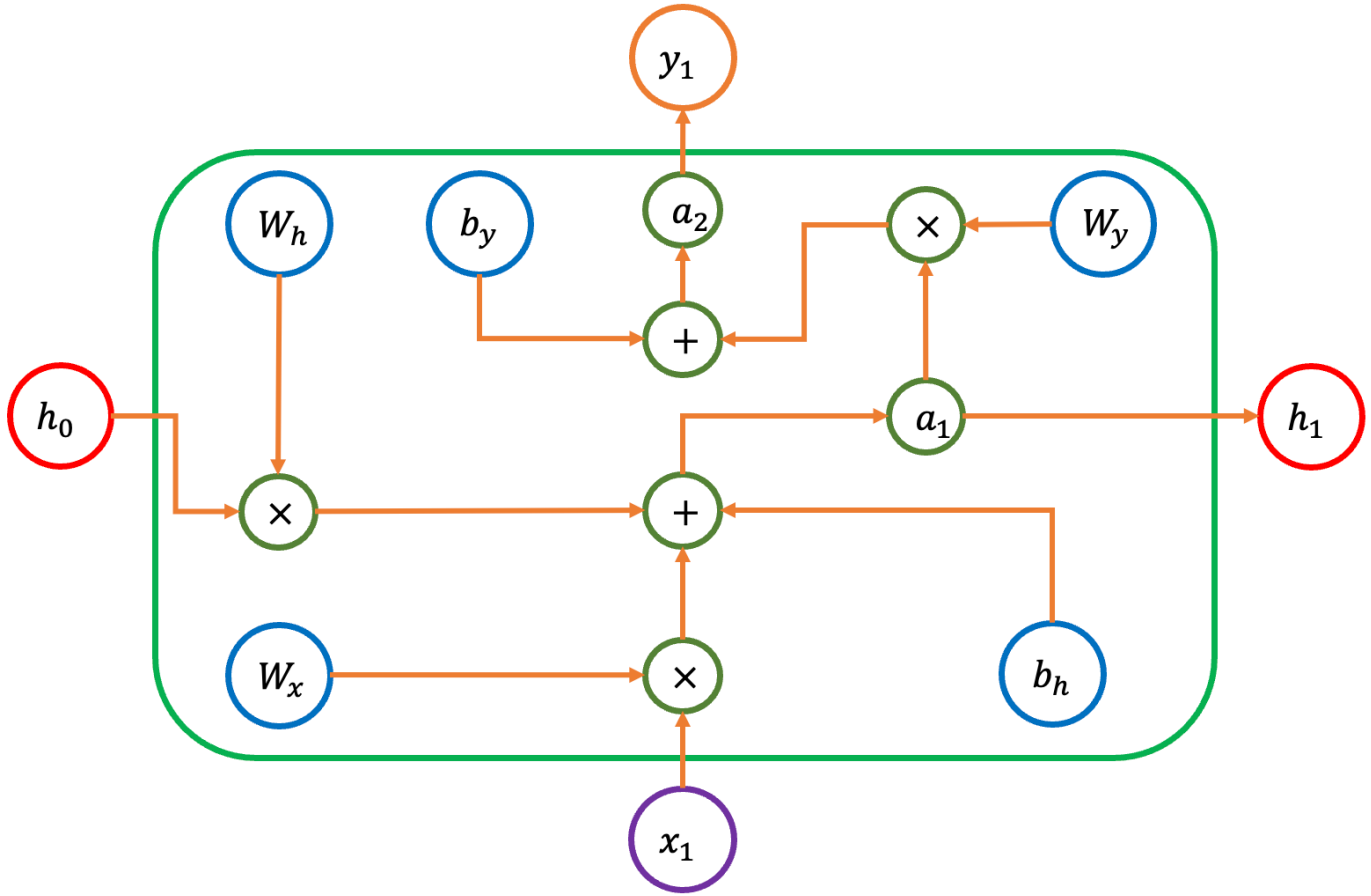


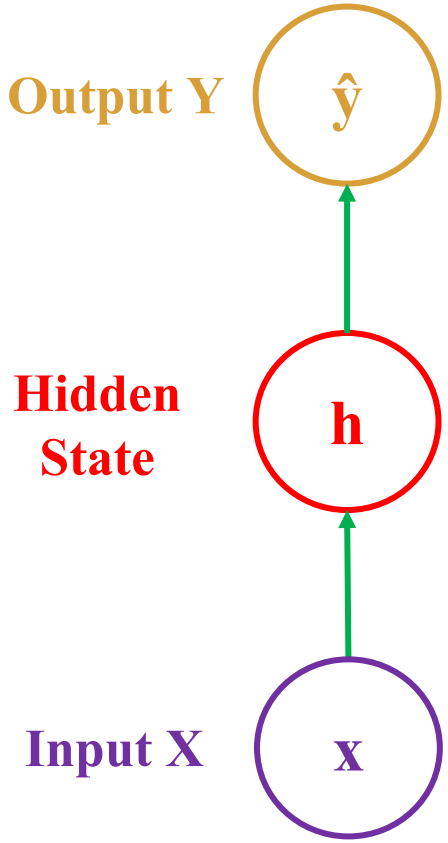
t = 1

t = 2

Attention Mechanism

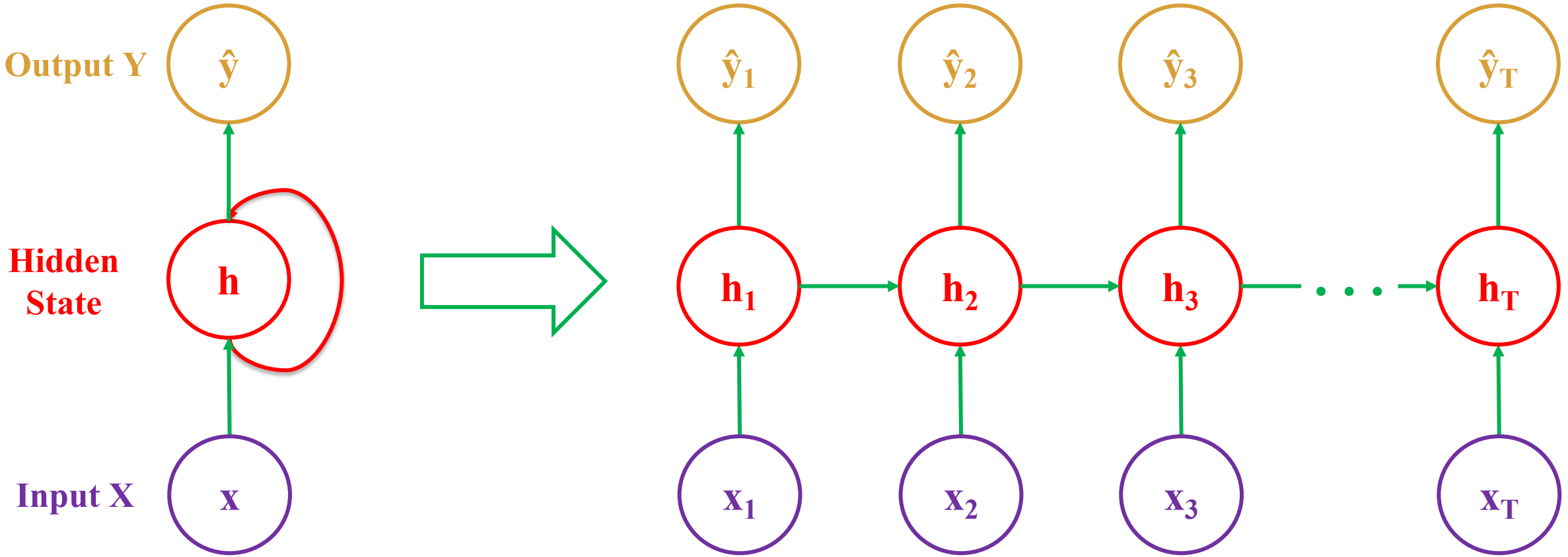
❖ RNNs Architecture (One-to-One)





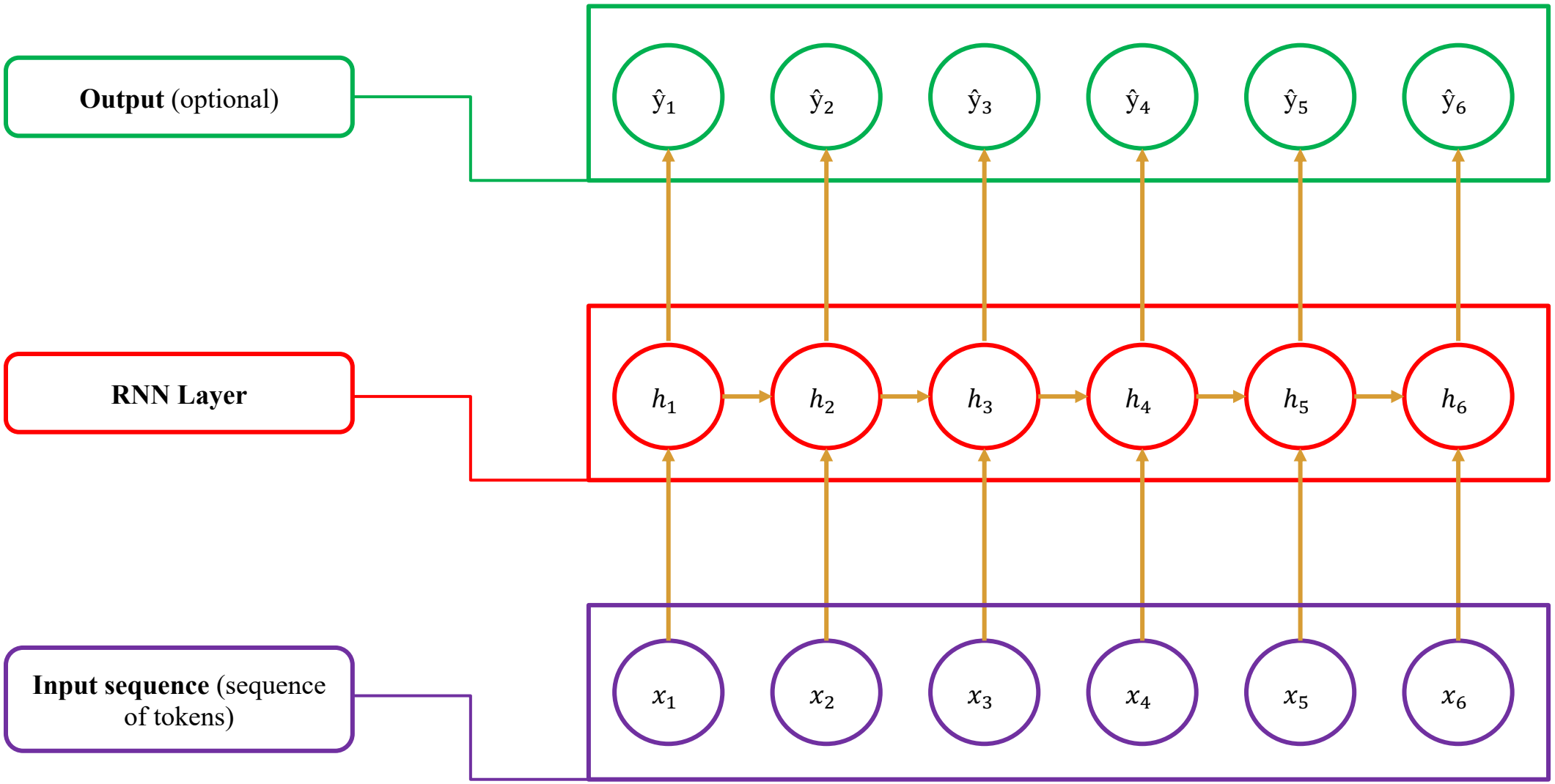
Attention Mechanism

❖ RNNs Architecture (Many-to-Many)



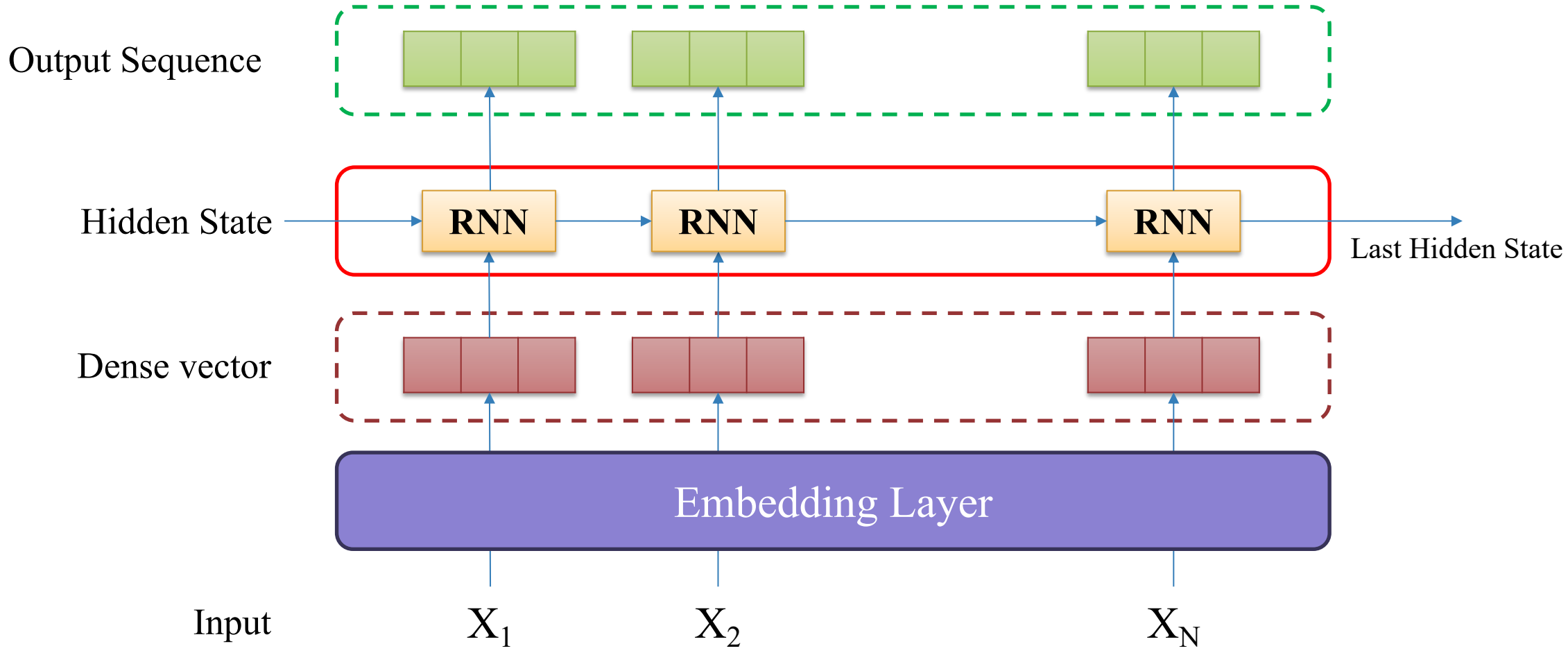
Attention Mechanism

❖ RNNs Architecture



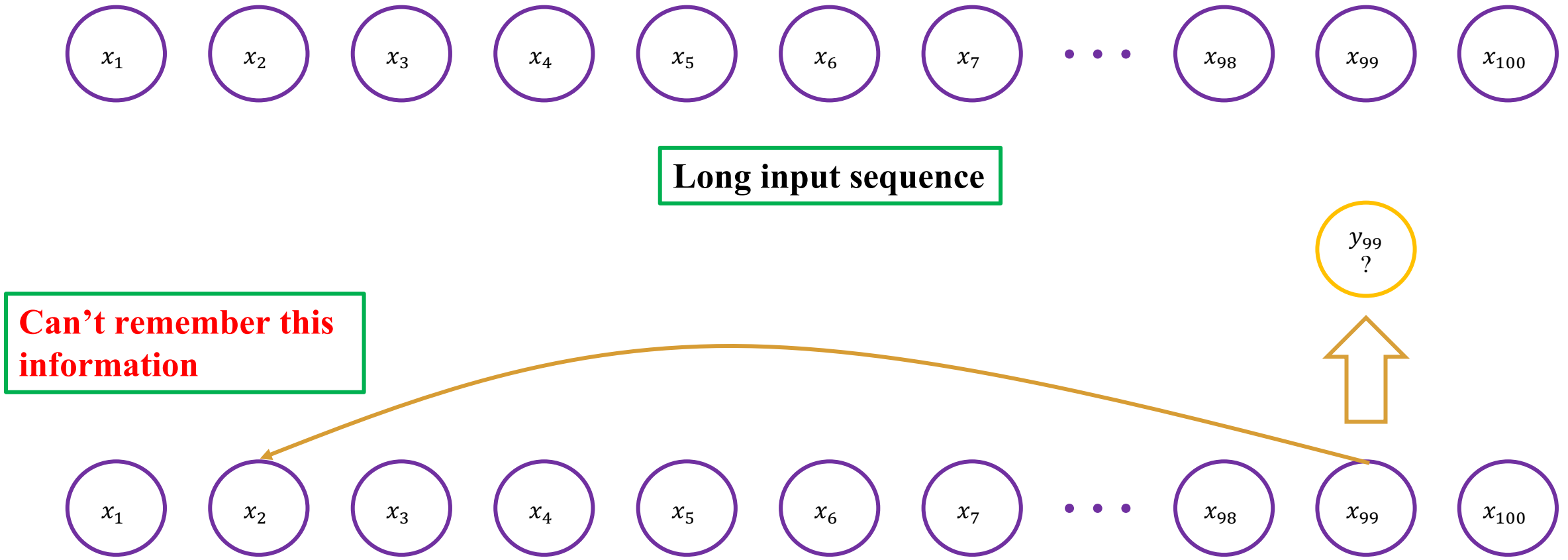
Attention Mechanism

❖ RNN Architecture



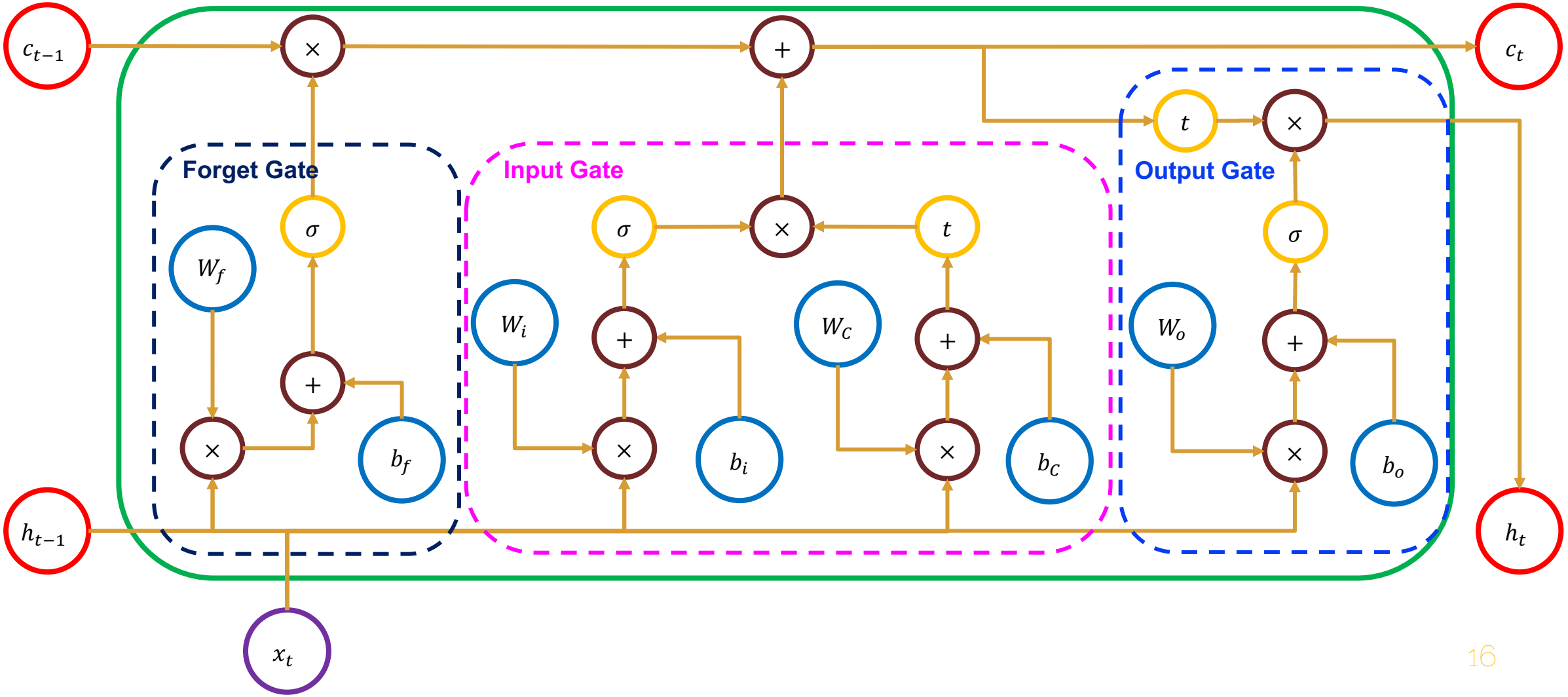
Attention Mechanism

❖ Long Short-Term Memory (LSTM)



Attention Mechanism

❖ LSTM Architecture



Attention Mechanism

❖ LSTM Architecture

Forget Gate:

- $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$

Input Gate:

- $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$
- $C_t^i = \tanh(W_C[h_{t-1}, x_t] + b_c)$

Cell State:

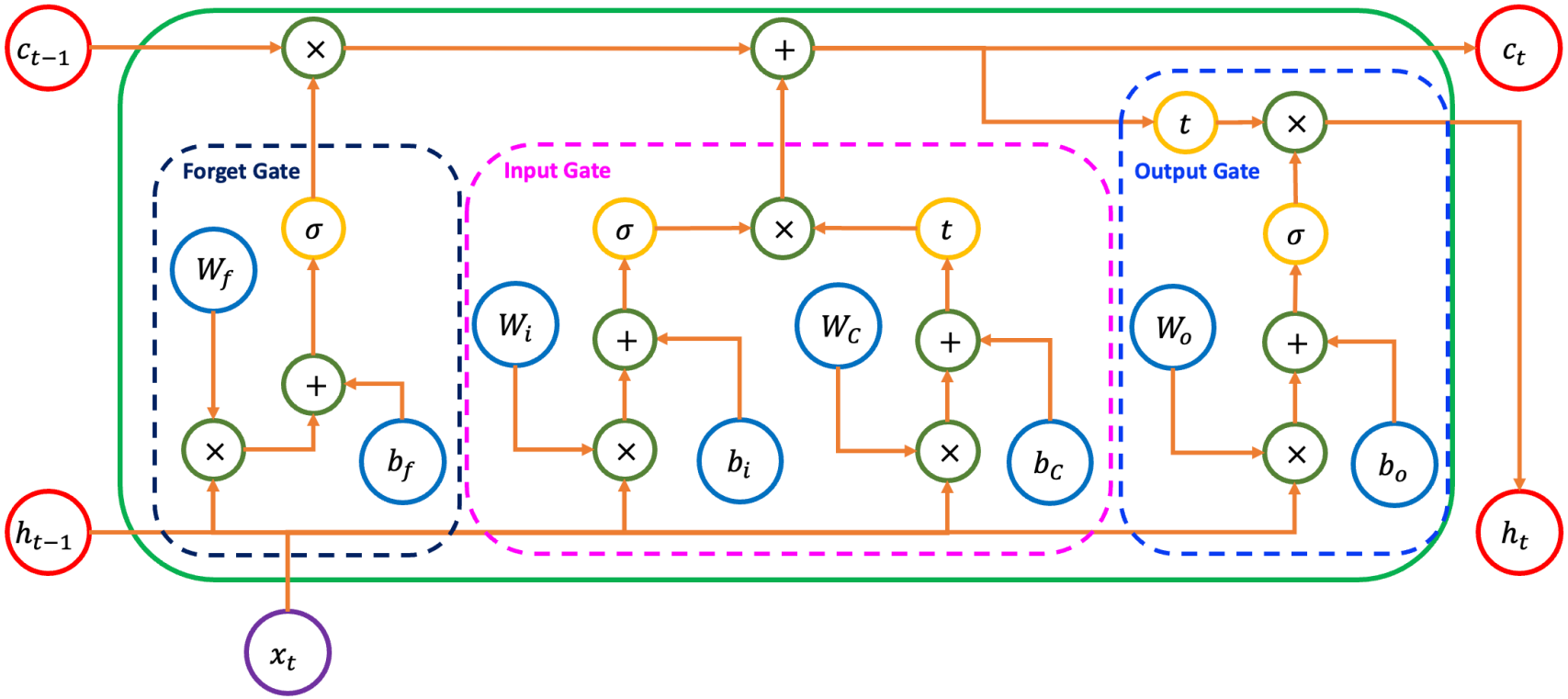
- $C_t = f_t C_{t-1} + i_t C_t^i$

Output Gate:

- $o_t = \sigma(W_o[h_{t-1}. x_t] + b_o)$

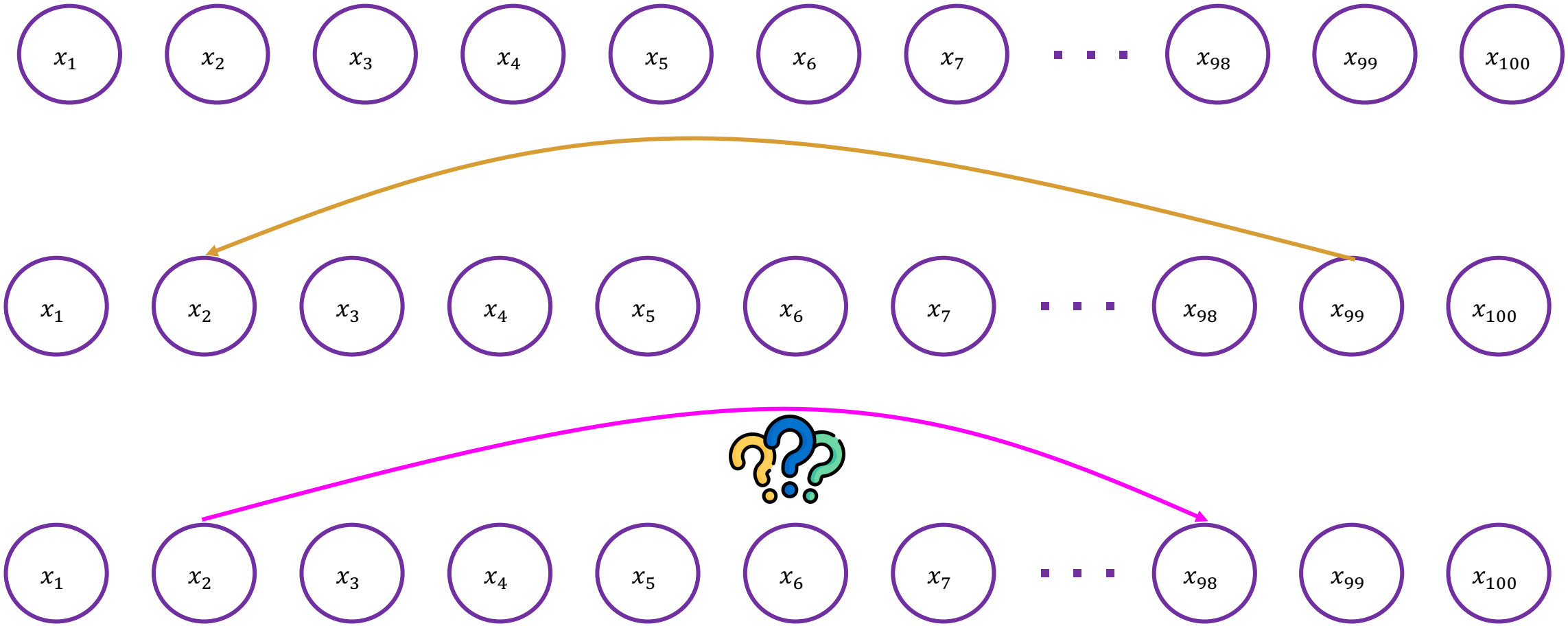
Hidden State:

- $h_t = \tanh(C_t) o_t$



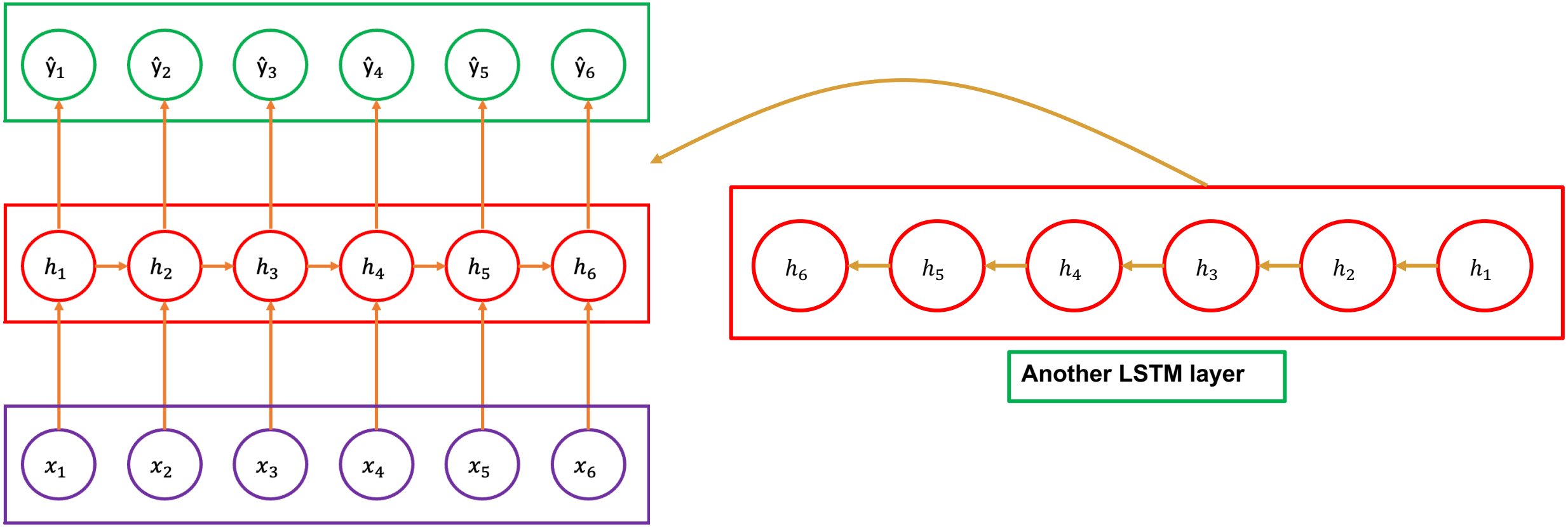
Attention Mechanism

❖ Bidirectional LSTM



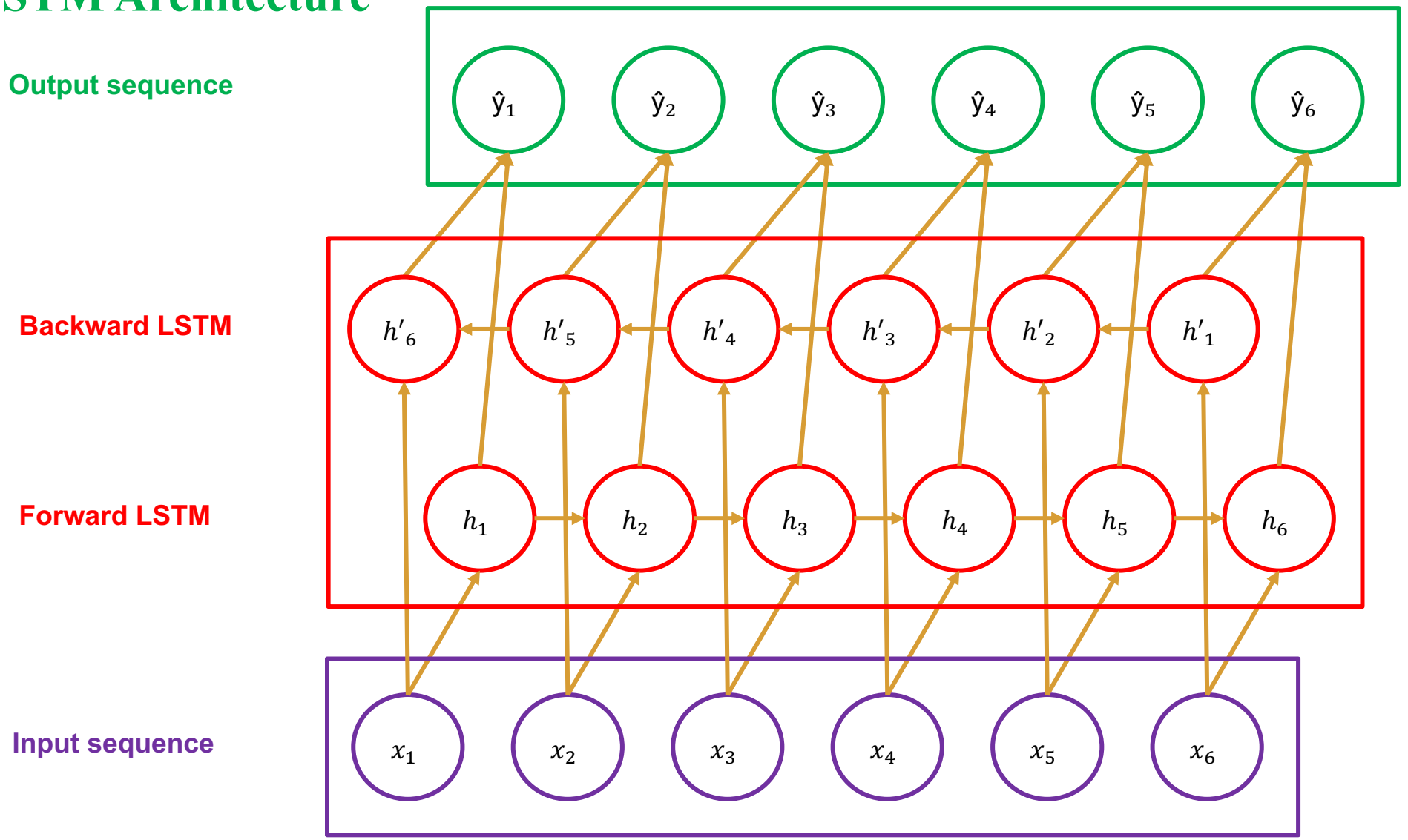
Attention Mechanism

❖ Bi-LSTM Architecture



Attention Mechanism

❖ Bi-LSTM Architecture

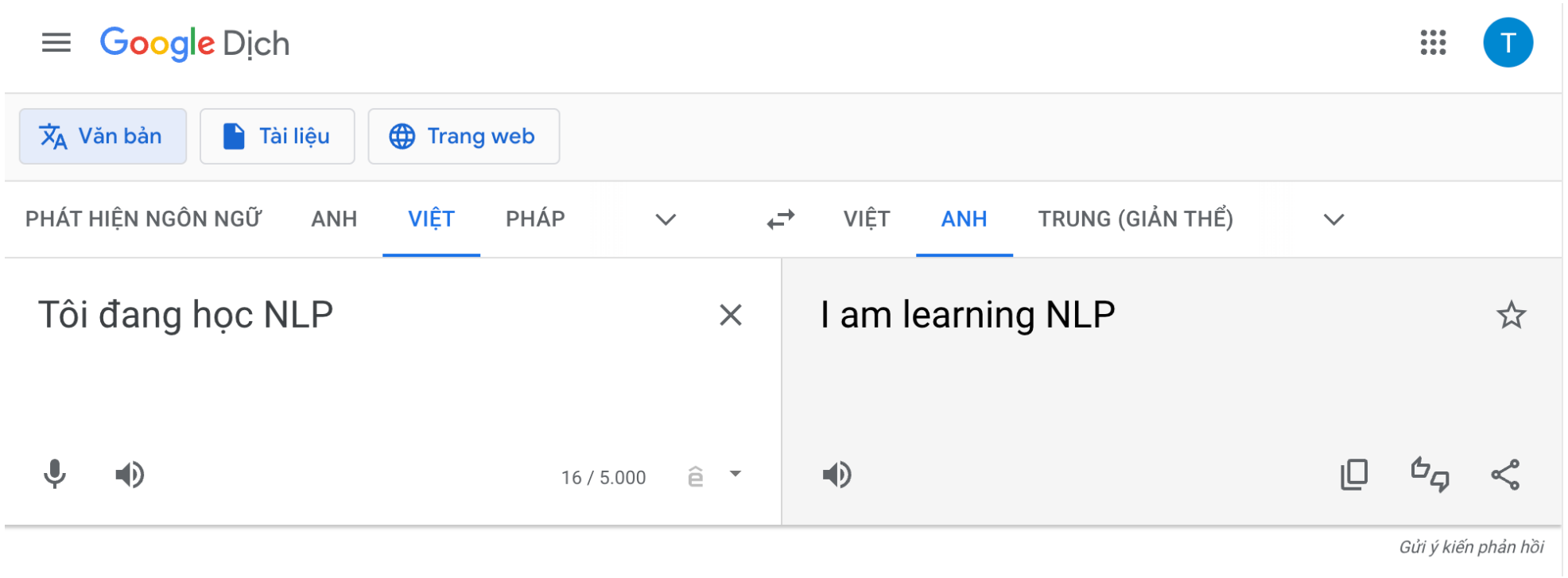


AI VIET NAM
@aivietnam.edu.vn

Attention Mechanism

❖ RNN/LSTM in Machine Translation

❖ Goal: Translate a sentence $w^{(s)}$ in a **source language (input)** to a sentence $w^{(t)}$ in the **target language (output)**



AI VIET NAM
@aivietnam.edu.vn

Attention Mechanism

❖ RNN/LSTM in Machine Translation

❖ Goal: Translate a sentence $w^{(s)}$ in a **source language (input)** to a sentence $w^{(t)}$ in the **target language (output)**

Automatic Speech Recognition (ASR)

translation of spoken language into text

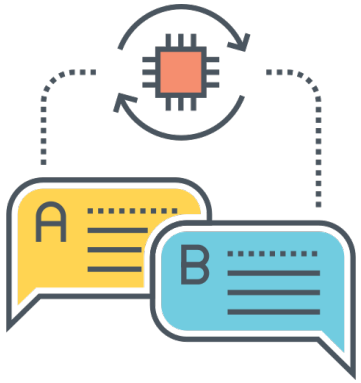
Natural Language Understanding (NLU)

a computer's ability to understand language

- ❑ Syntax
- ❑ Semantics
- ❑ Phonology
- ❑ Pragmatics
- ❑ Morphology

Natural Language Generation (NLG)

generate natural language by a computer



Attention Mechanism

❖ Sequence to Sequence (Seq2Seq)

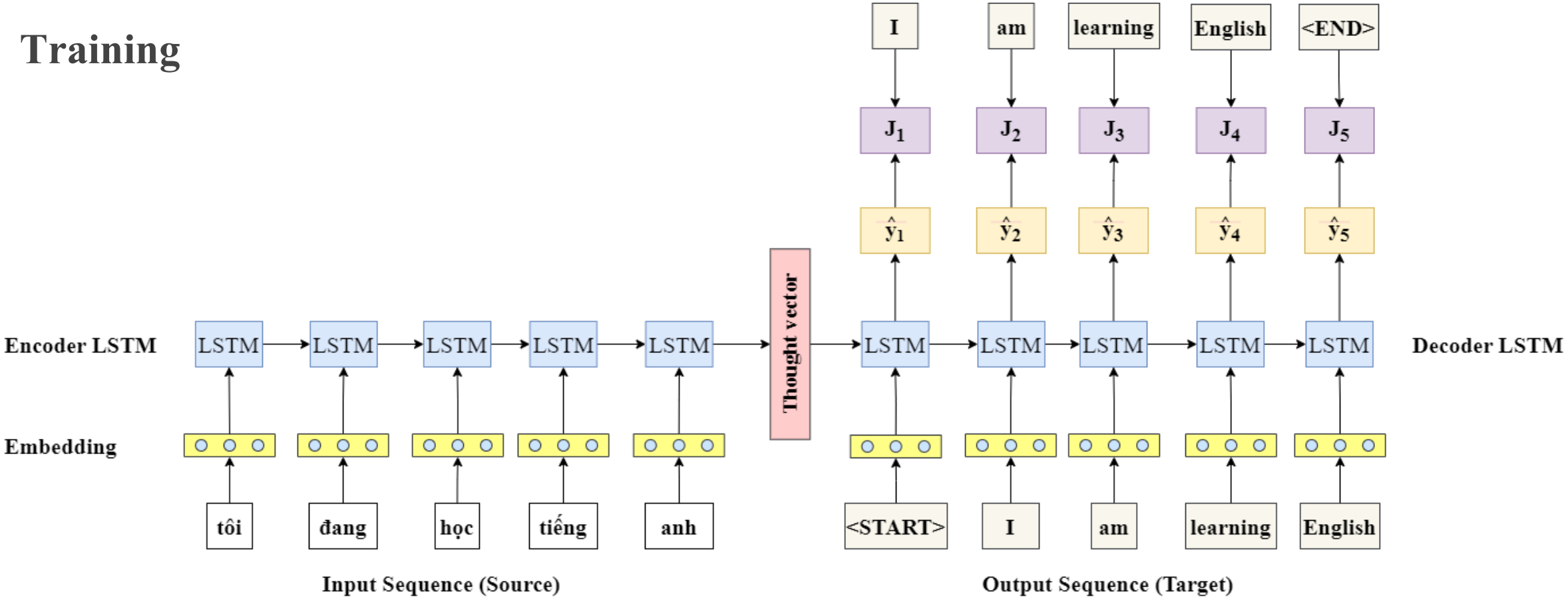
- A single neural network is used to translate from source to target.
- Architecture: Encoder-Decoder.
- Encoder: Convert source sentence (input) into a vector/matrix (State).
- Decoder: Convert encoding into a sentence in target language (output).

Thought Vector
Capture all information of input sentence

Attention Mechanism

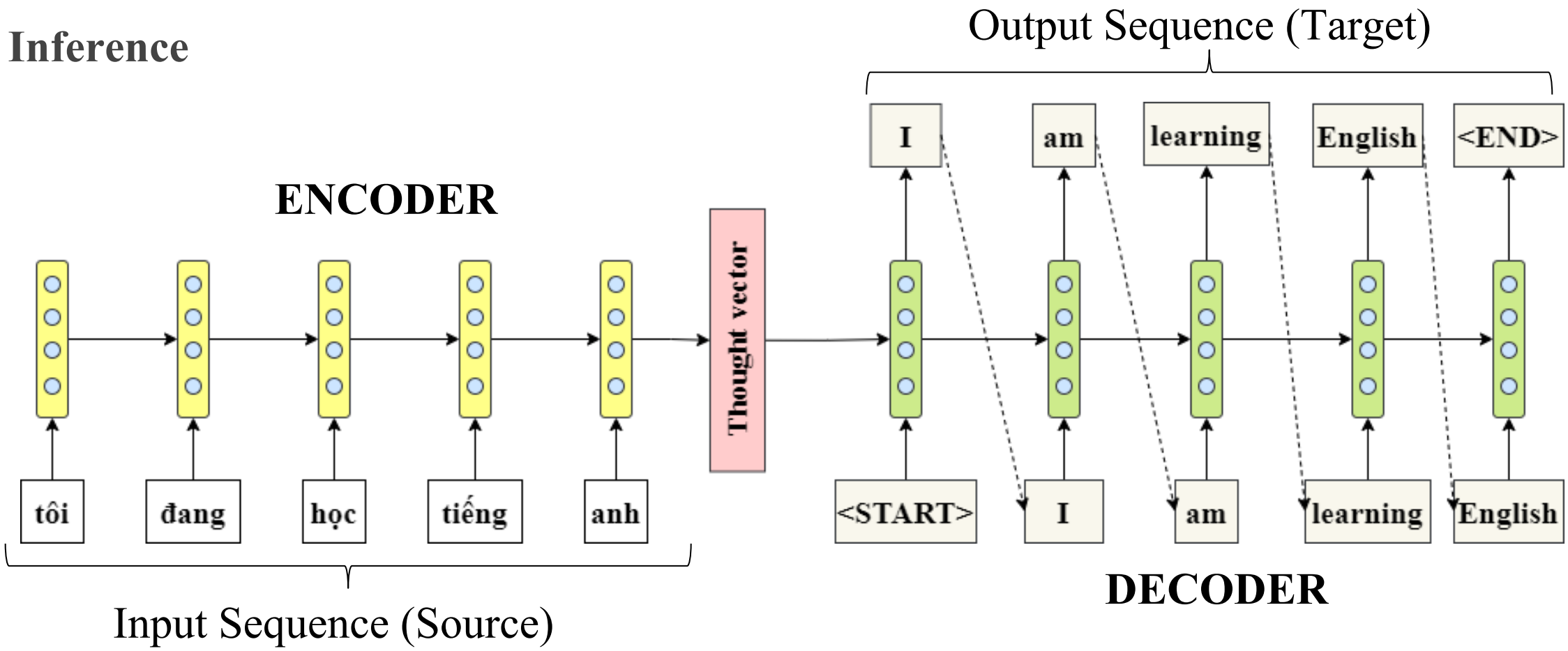
❖ Sequence to Sequence (Seq2Seq)

Training



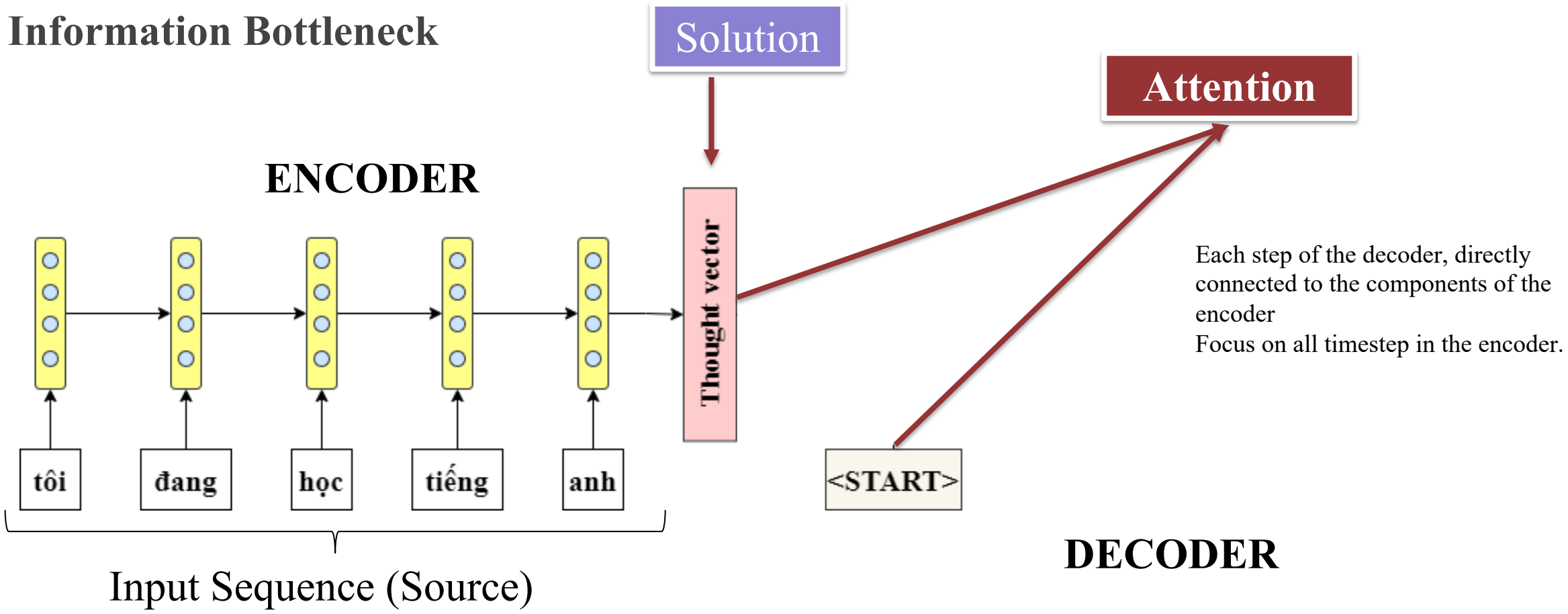
Attention Mechanism

❖ Sequence to Sequence (Seq2Seq)



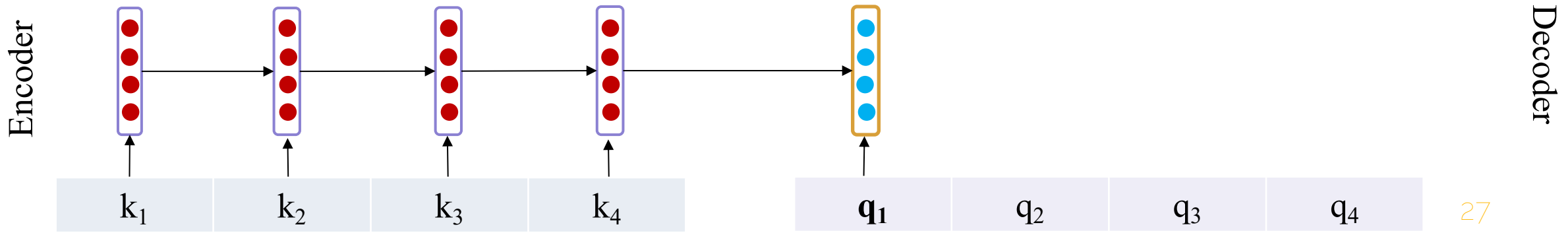
Attention Mechanism

❖ Sequence to Sequence (Seq2Seq)



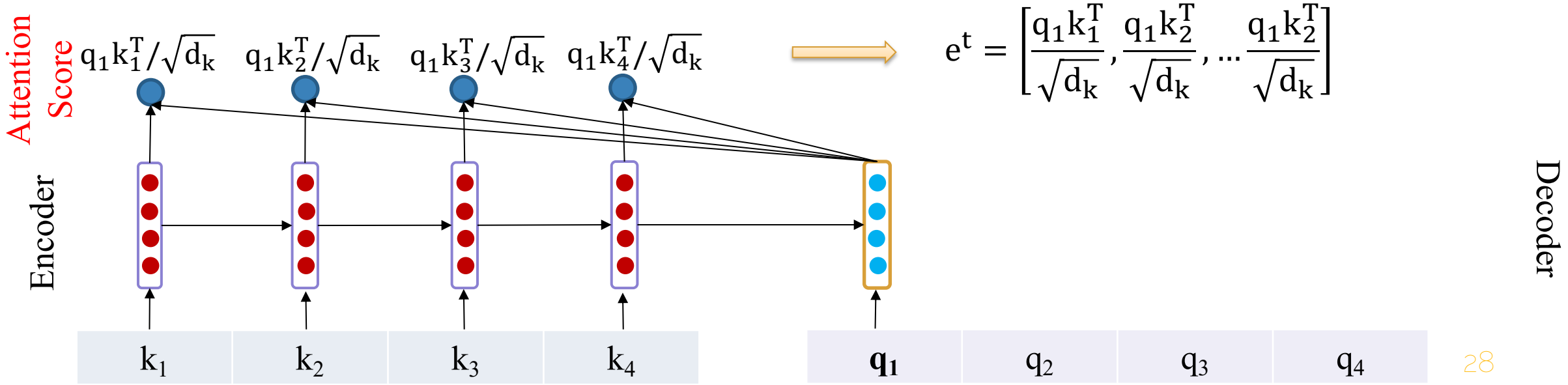
Attention Mechanism

❖ Scaled Dot-Product Attention



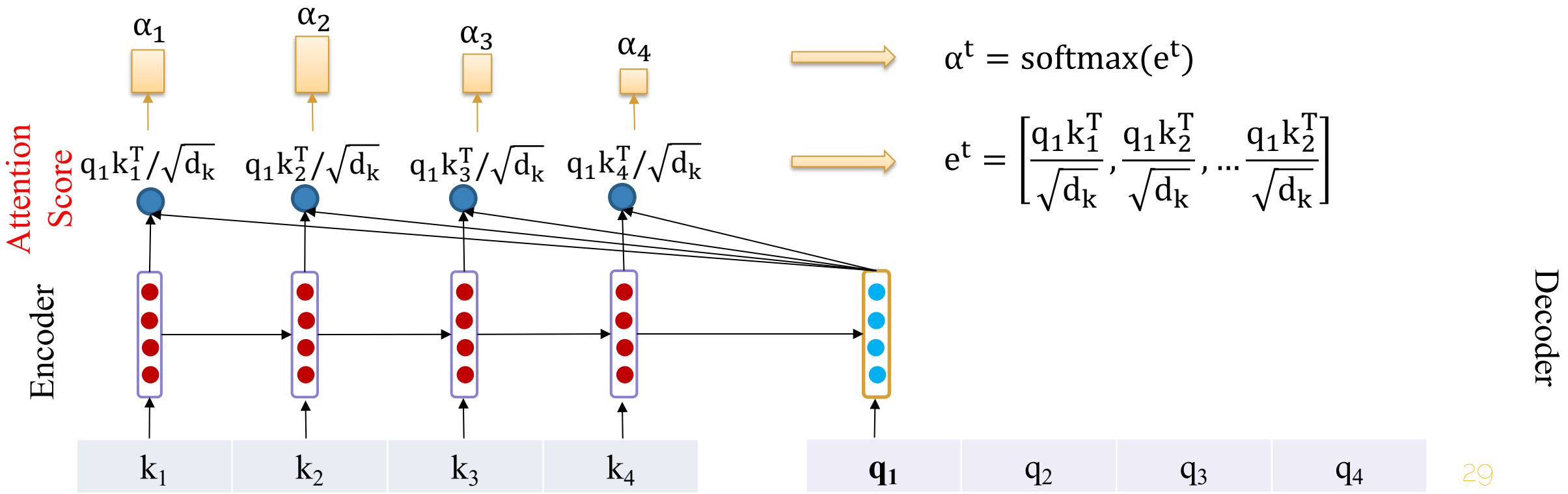
Attention Mechanism

❖ Scaled Dot-Product Attention



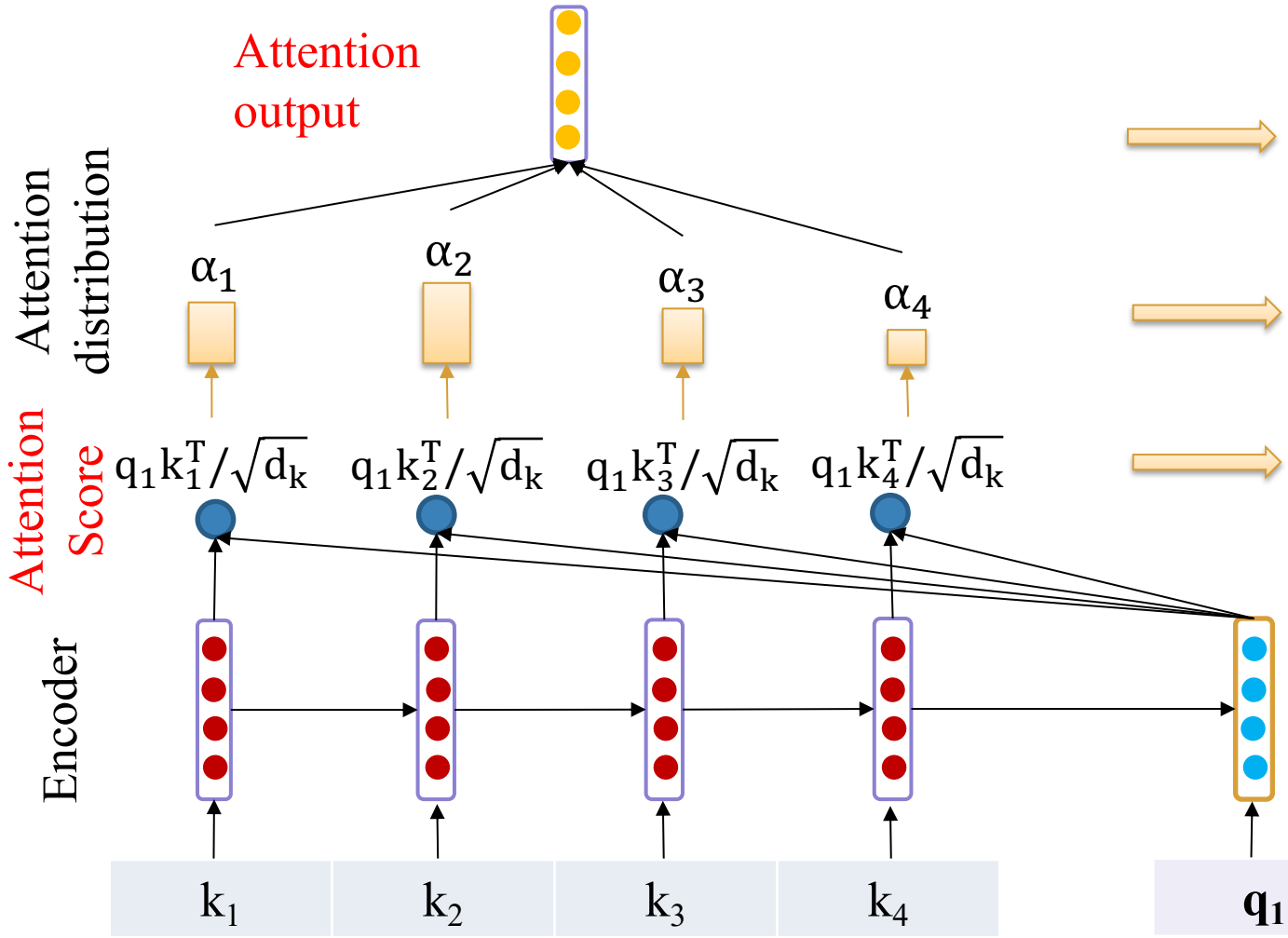
Attention Mechanism

❖ Scaled Dot-Product Attention



Attention Mechanism

❖ Scaled Dot-Product Attention



$$a^t = \sum_1^N \alpha_i^t v_i \; ; \; v_i = k_i$$

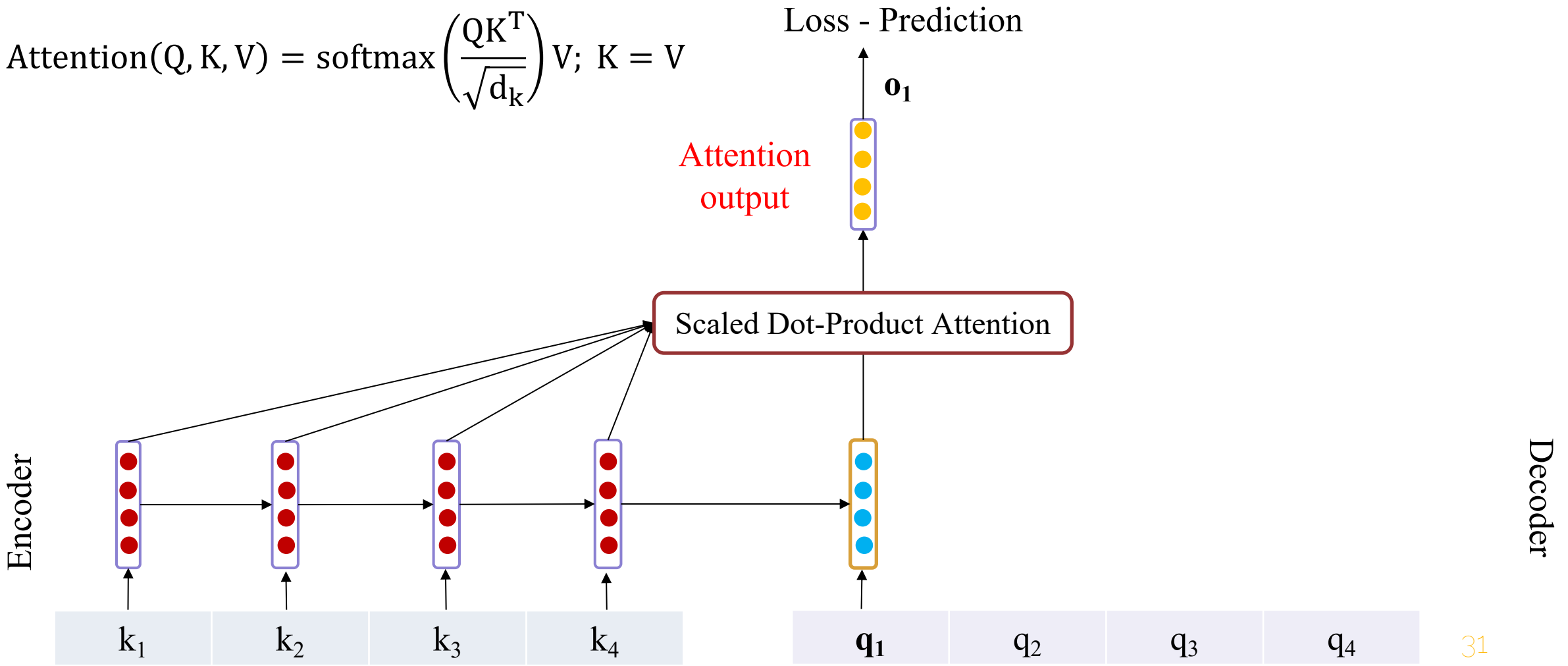
$$\alpha^t = \text{softmax}(e^t)$$

$$e^t = \left[\frac{q_1k_1^T}{\sqrt{d_k}}, \frac{q_1k_2^T}{\sqrt{d_k}}, \ldots \frac{q_1k_2^T}{\sqrt{d_k}}\right]$$

Attention Mechanism

❖ Scaled Dot-Product Attention

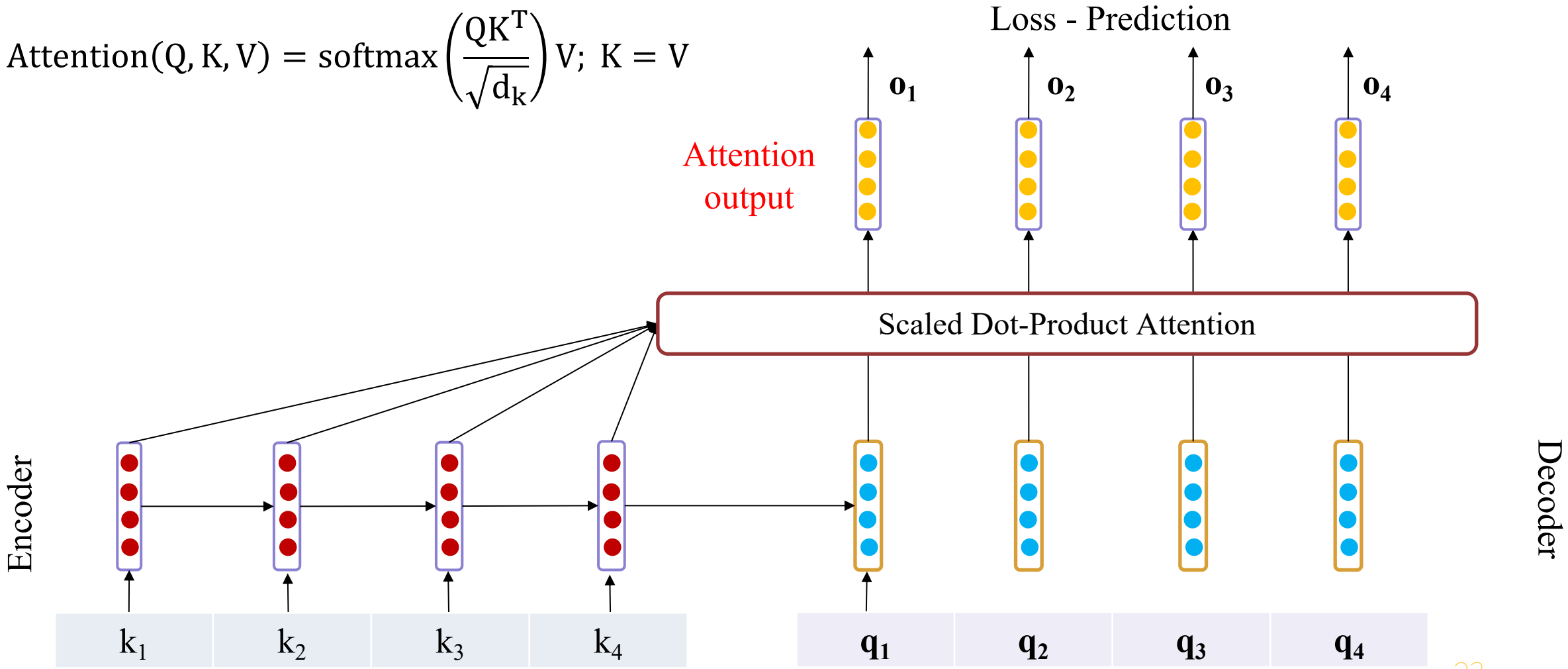
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V;\ K = V$$



Attention Mechanism

❖ Scaled Dot-Product Attention

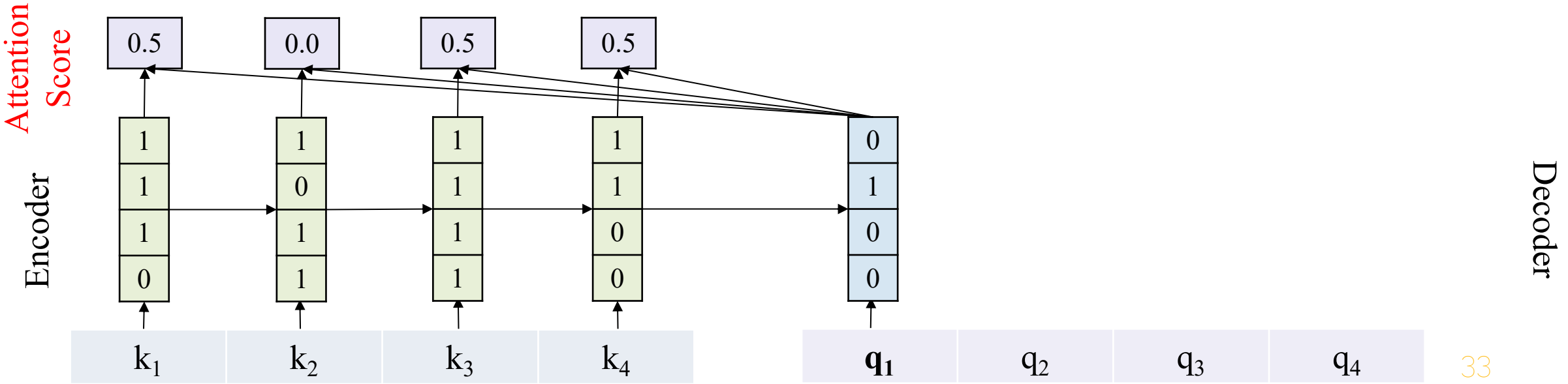
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V;\ K = V$$



Attention Mechanism

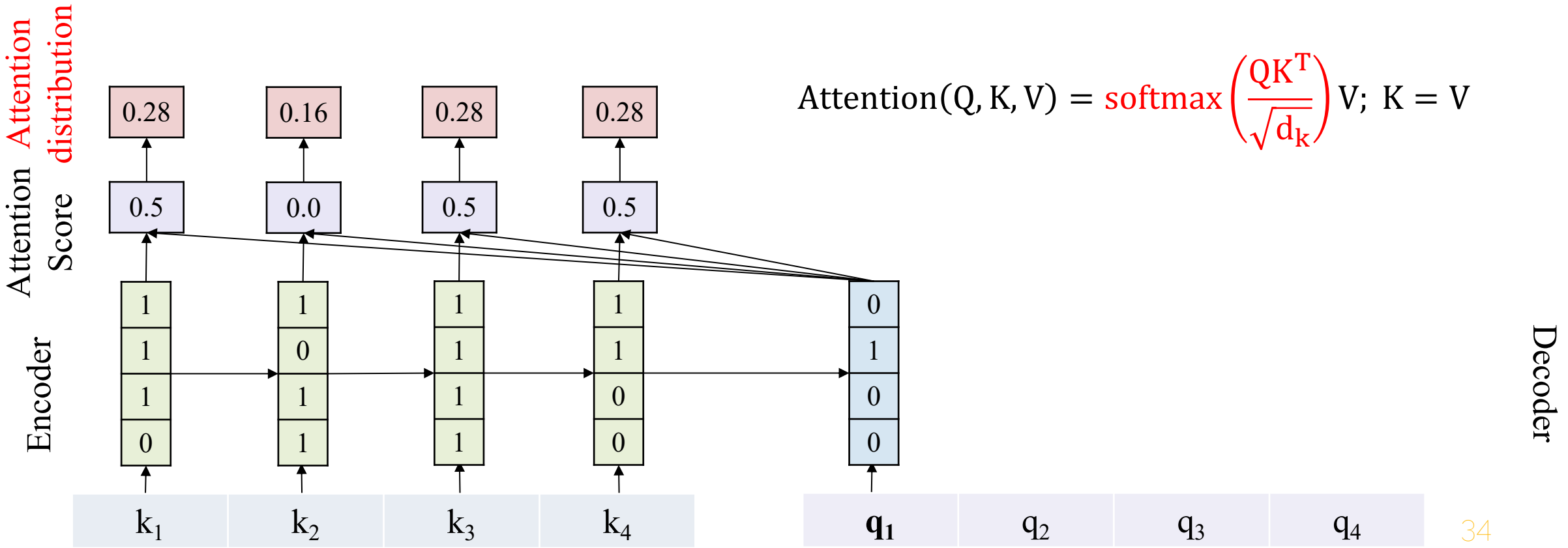
❖ Scaled Dot-Product Attention

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



Attention Mechanism

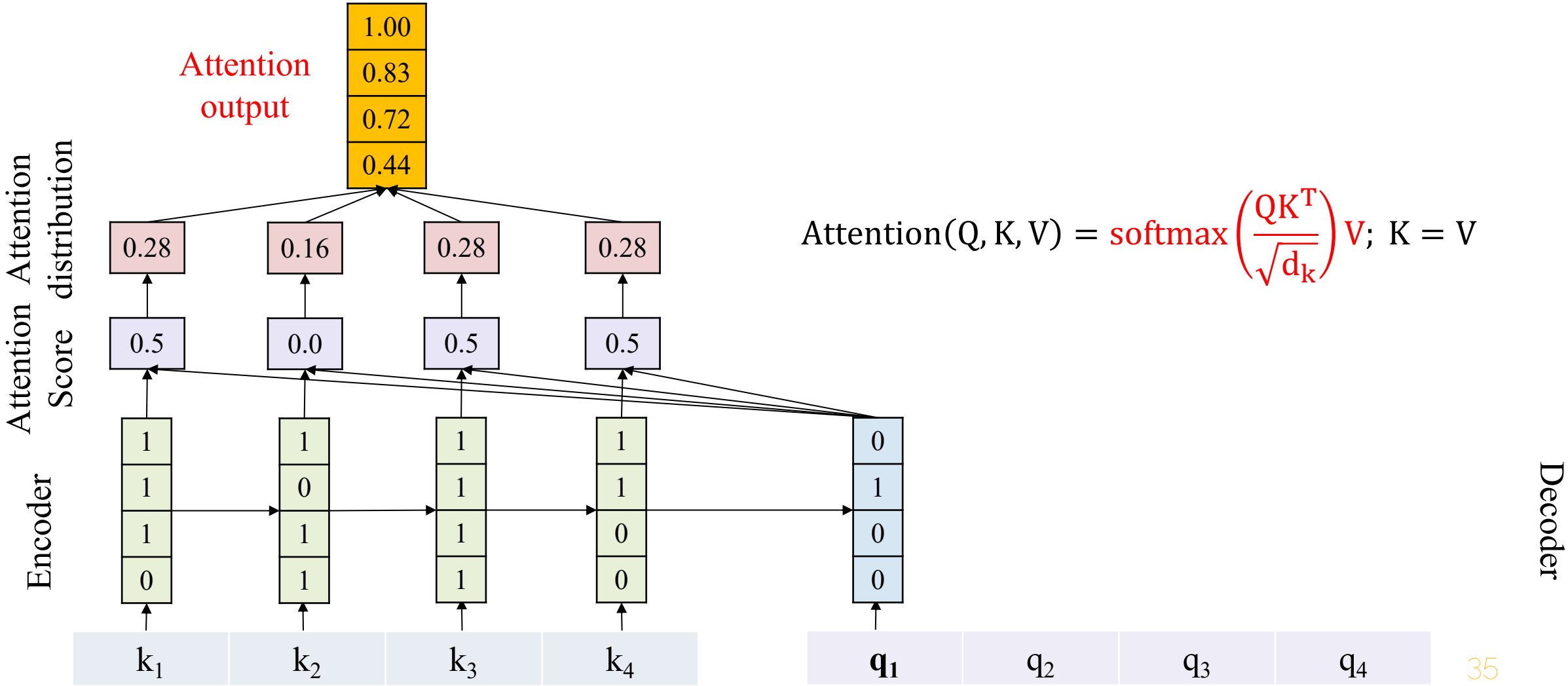
❖ Scaled Dot-Product Attention



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

Attention Mechanism

❖ Scaled Dot-Product Attention

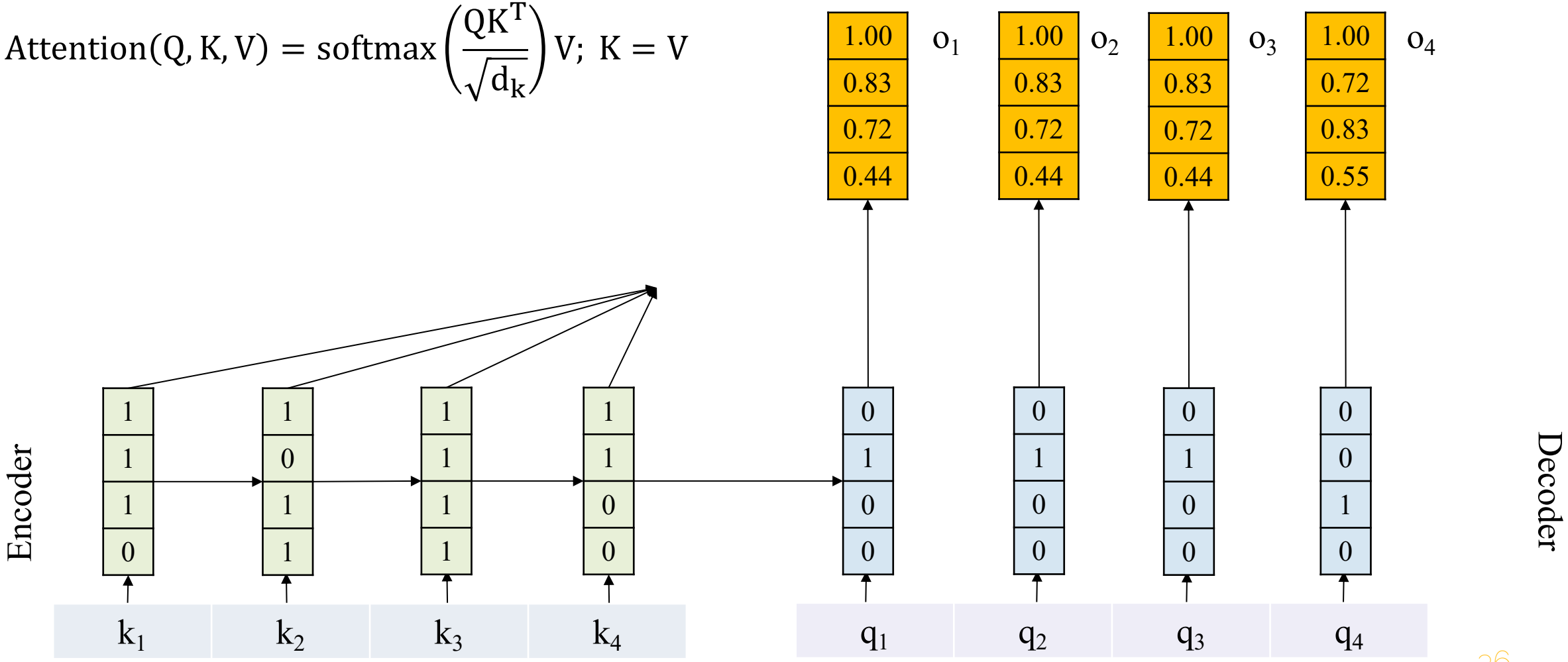


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V;\ K = V$$

Attention Mechanism

❖ Scaled Dot-Product Attention

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



Attention Mechanism

❖ Scaled Dot-Product Attention

```
query = torch.randint(
    high=2,
    size=(1, 4, 4), # batch_size x seq_len x embedding_dim
    dtype=torch.float32
)
query
```

```
tensor([[[0., 1., 0., 0.],
         [0., 1., 0., 0.],
         [0., 1., 0., 0.],
         [0., 0., 1., 0.]]])
```

```
key = torch.randint(
    high=2,
    size=(1, 4, 4),
    dtype=torch.float32
)
key
```

```
tensor([[[1., 1., 1., 0.],
         [1., 0., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 0., 0.]]])
```

```
value = key
value
```

```
tensor([[[1., 1., 1., 0.],
         [1., 0., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 0., 0.]]])
```

```
attentionn_weight = F.scaled_dot_product_attention(
    query=query,
    key=key,
    value=value
)
attentionn_weight
```

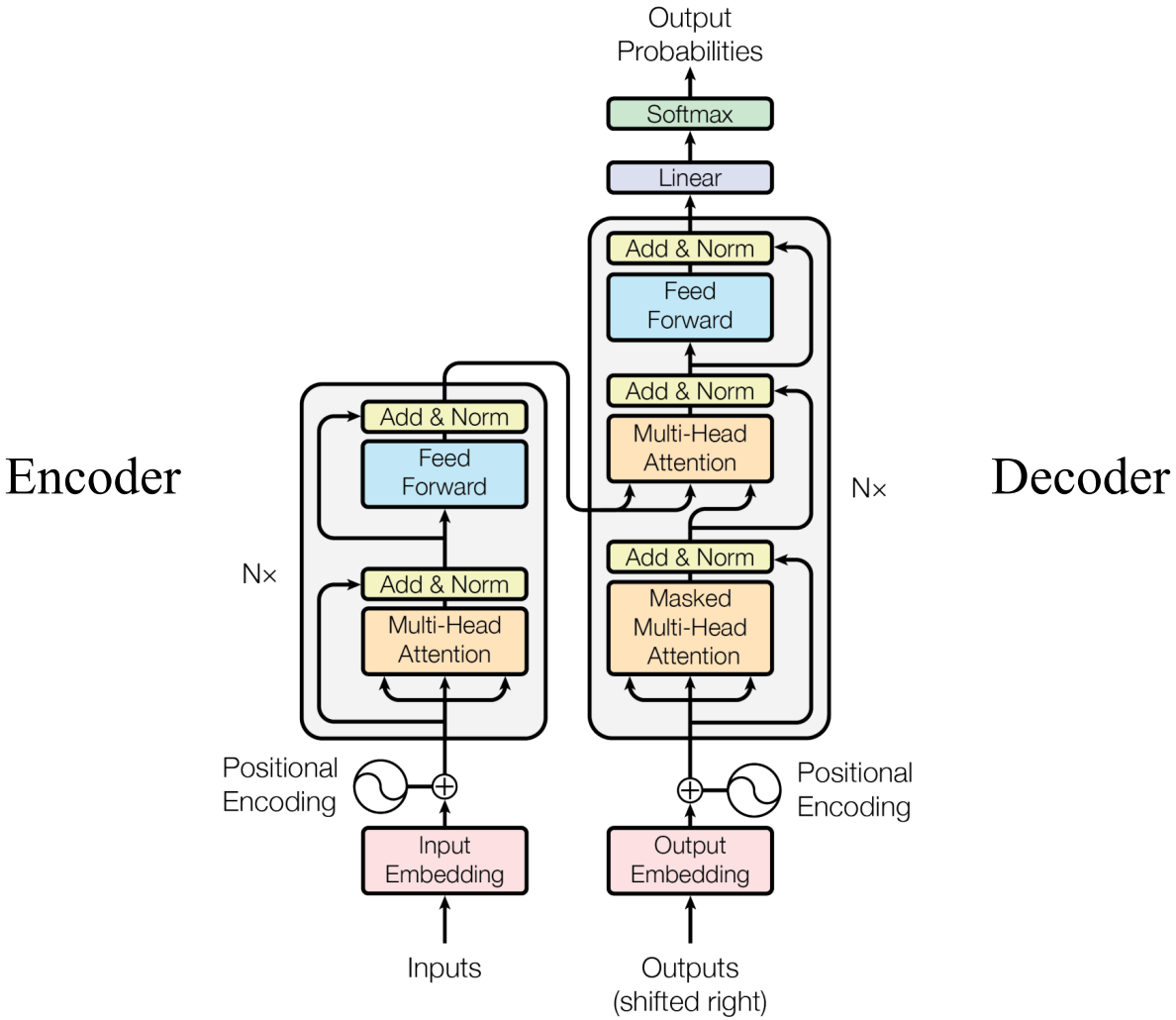
```
tensor([[[1.0000, 0.8318, 0.7227, 0.4455],
         [1.0000, 0.8318, 0.7227, 0.4455],
         [1.0000, 0.8318, 0.7227, 0.4455],
         [1.0000, 0.7227, 0.8318, 0.5545]]])
```

AI VIET NAM
@aivietnam.edu.vn

Transformer Encoder

Introduction

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



Encoder

Decoder

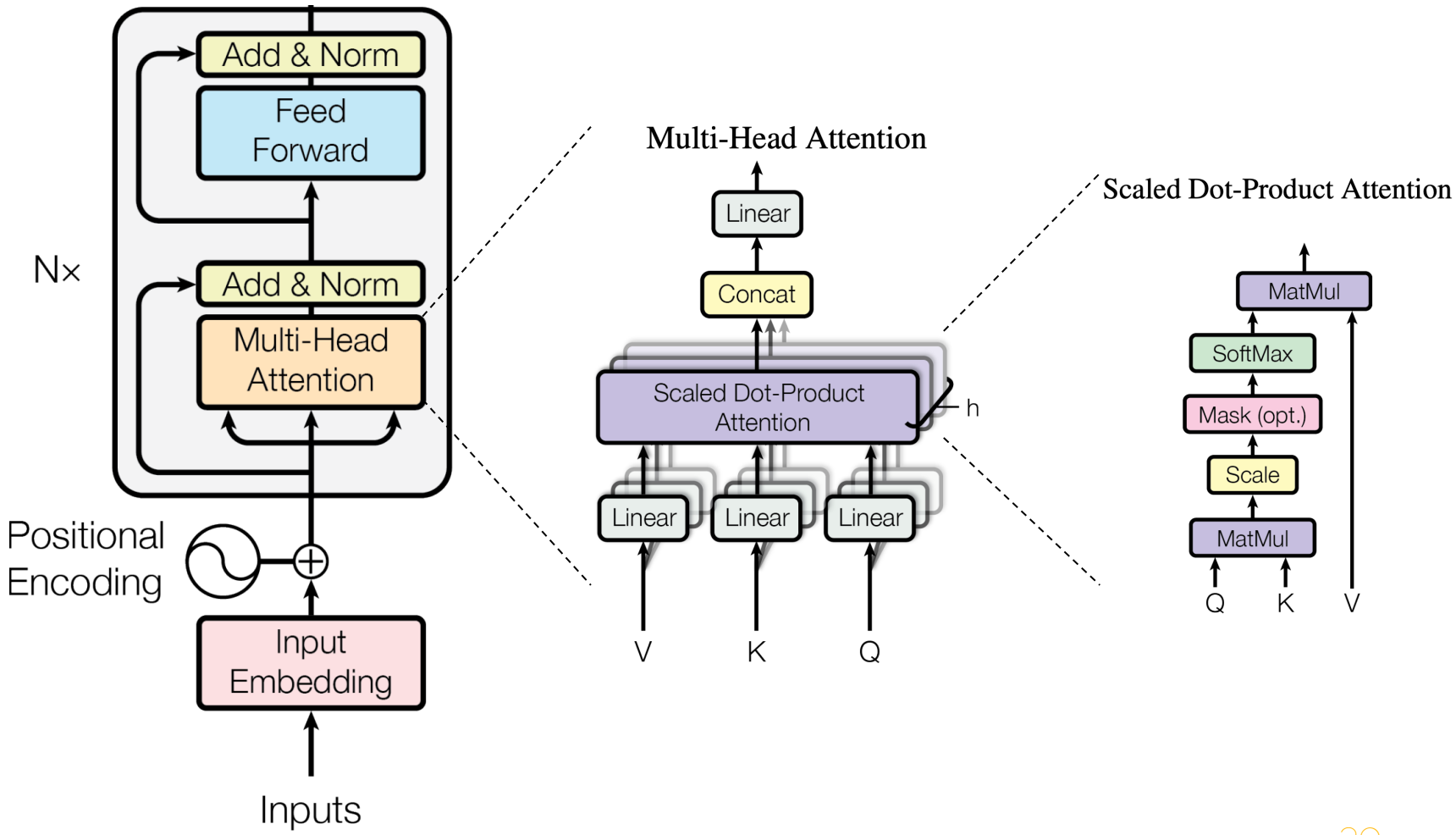
Figure: Architecture of Transformer.

https://arxiv.org/pdf/1706.03762

Transformer Encoder

❖ Transformer Encoder

- Input Embedding
- Positional Encoding
- Multi-Head Attention
- Feed Forward
- Add & Norm

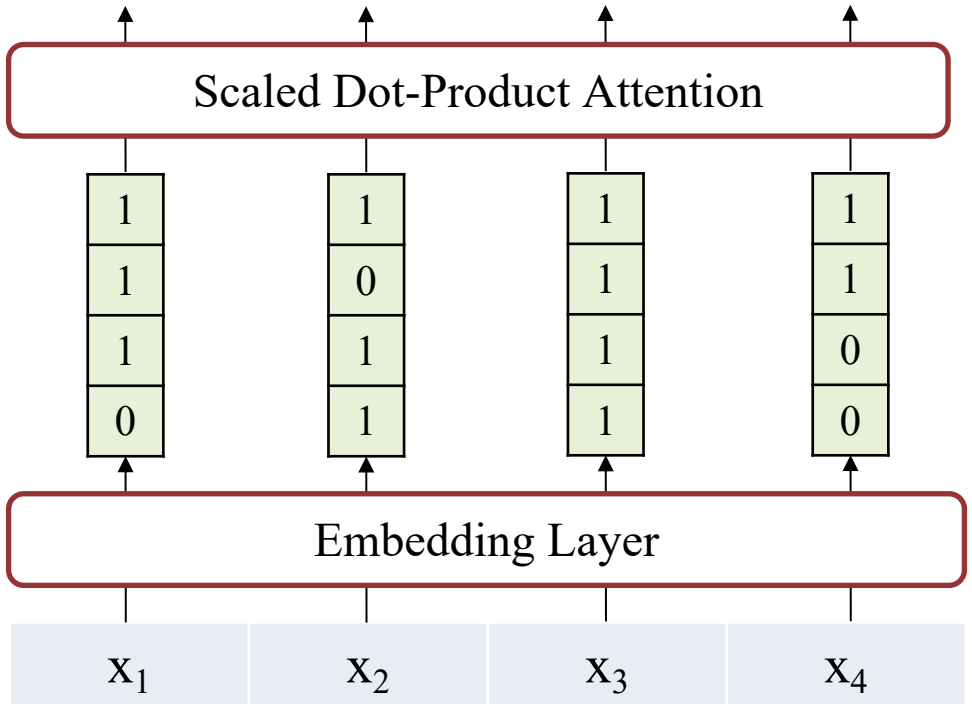


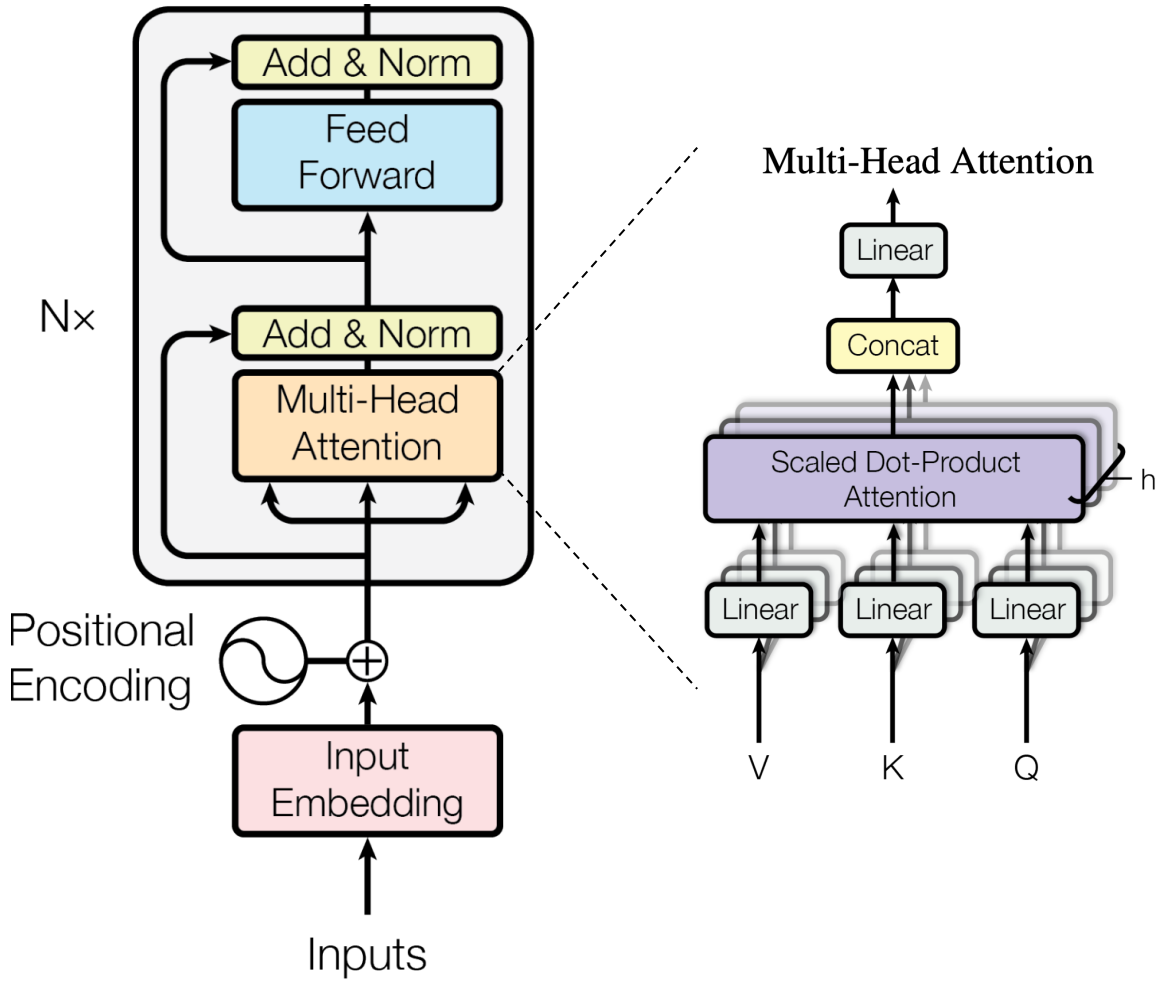
Transformer Encoder

❖ Input Embedding

❖ Input Embedding: Embedding Layer

QUERY – KEY – VALUE ?

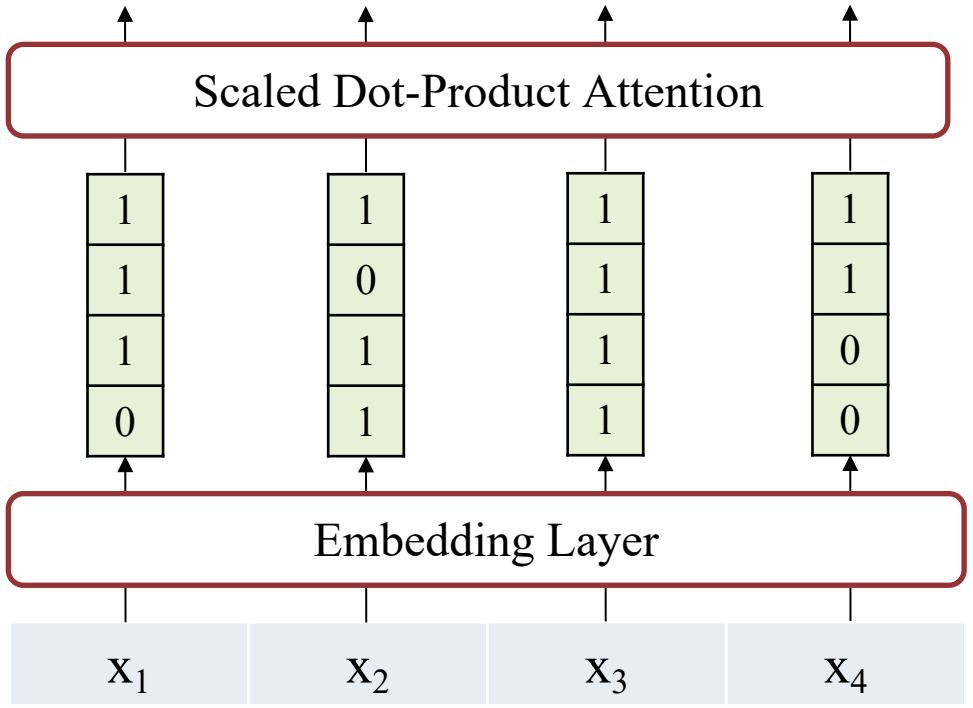


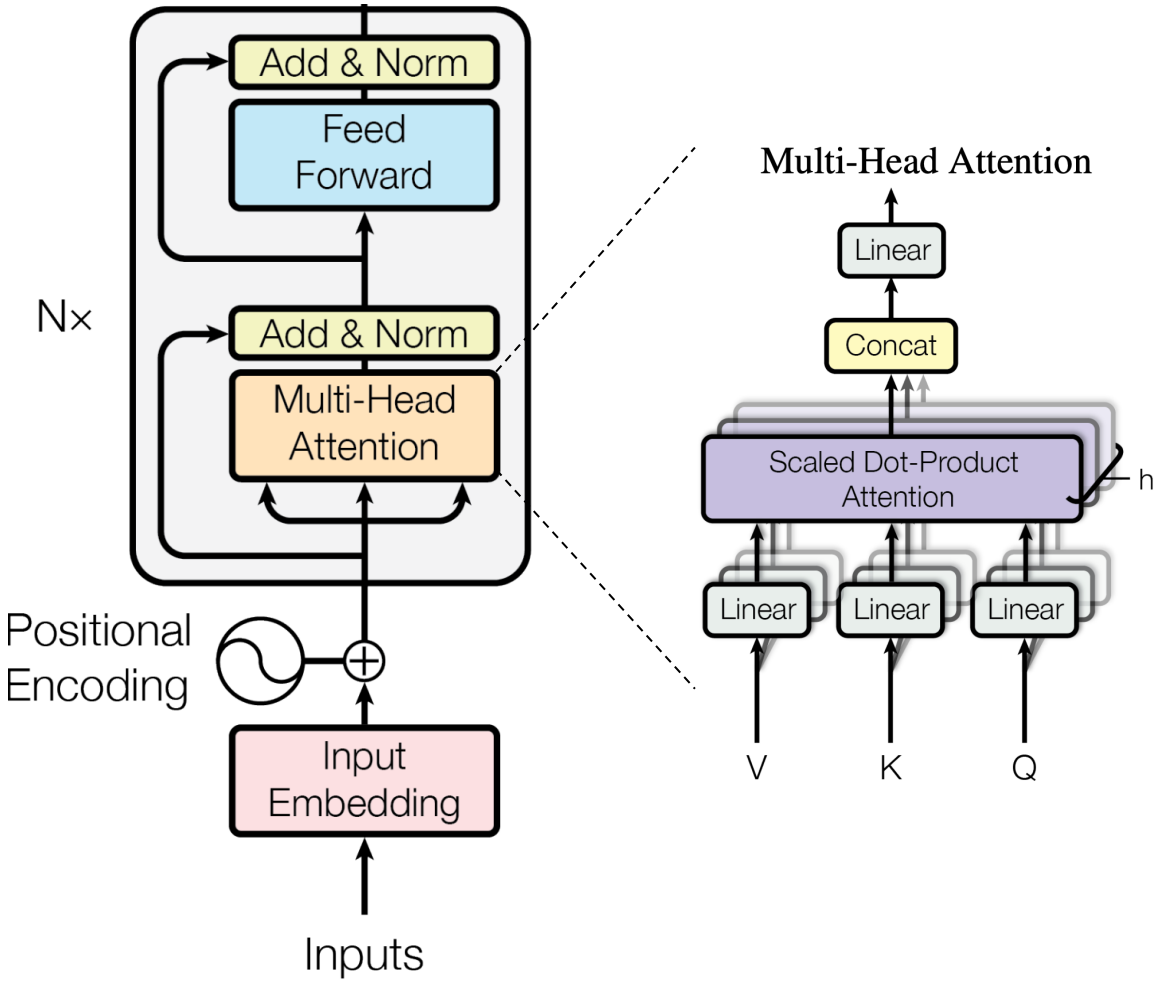


Transformer Encoder

❖ Self-attention

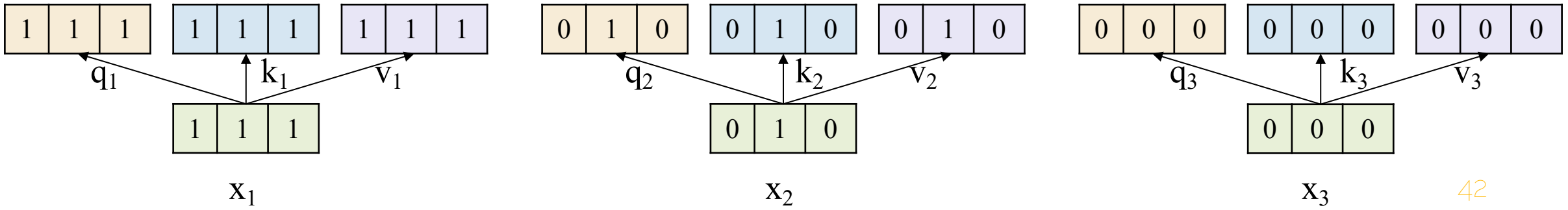
QUERY = KEY = VALUE = EMBEDDED





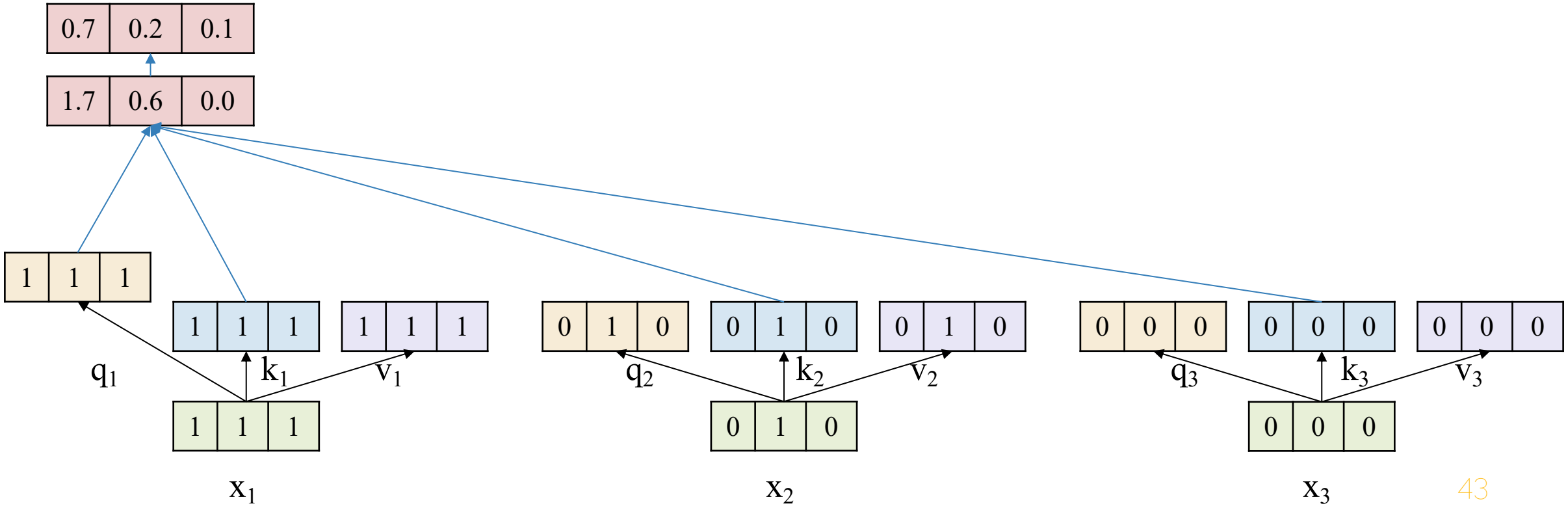
Transformer Encoder

❖ Self-attention



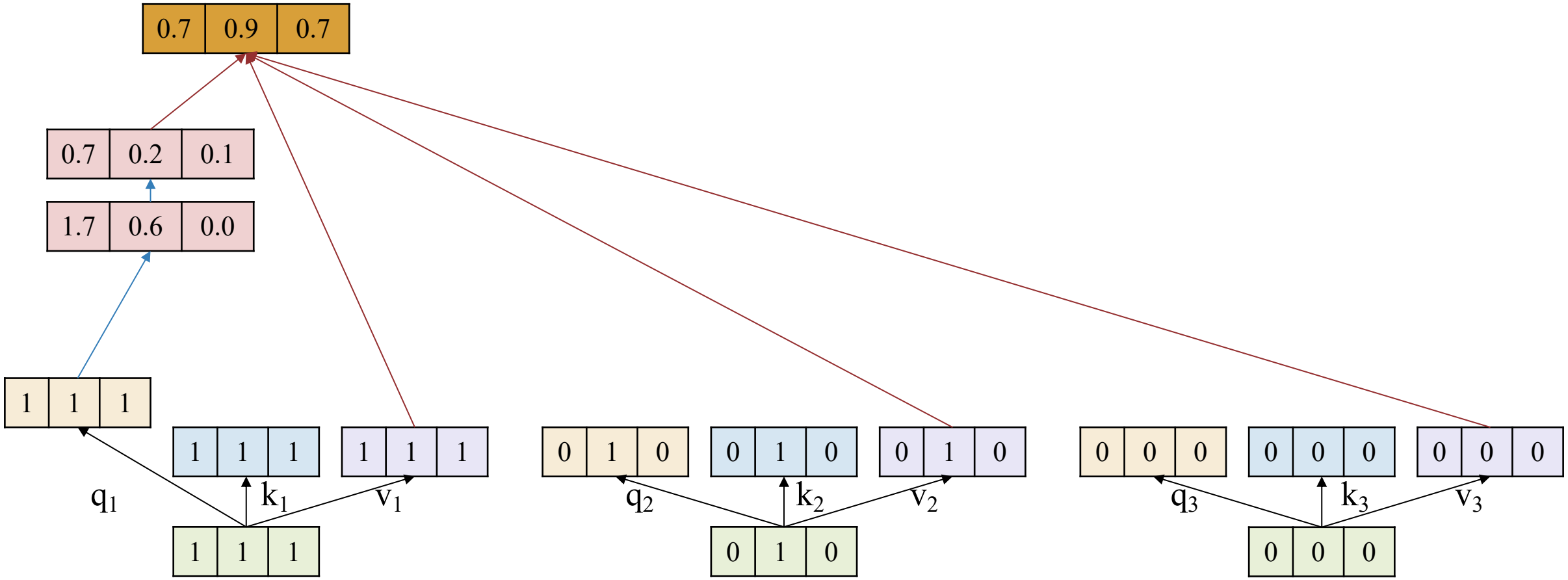
Transformer Encoder

❖ Self-attention



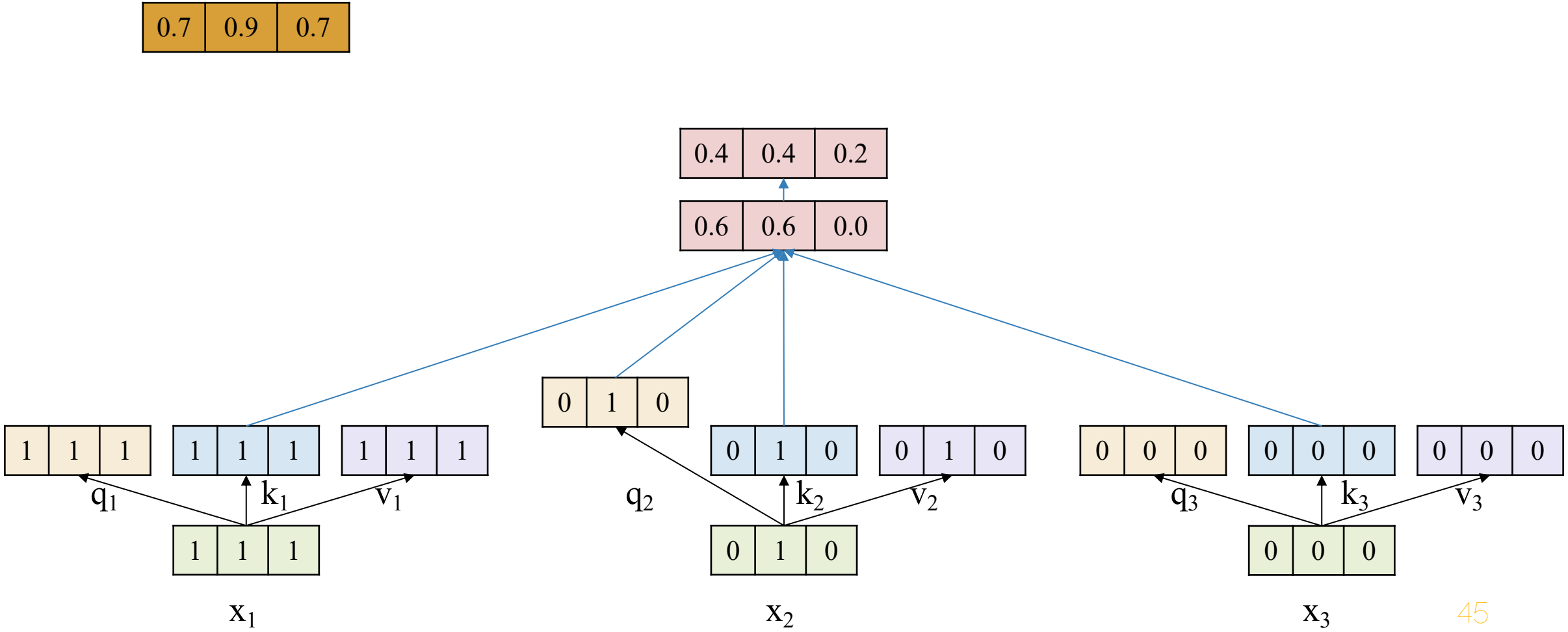
Transformer Encoder

❖ Self-attention



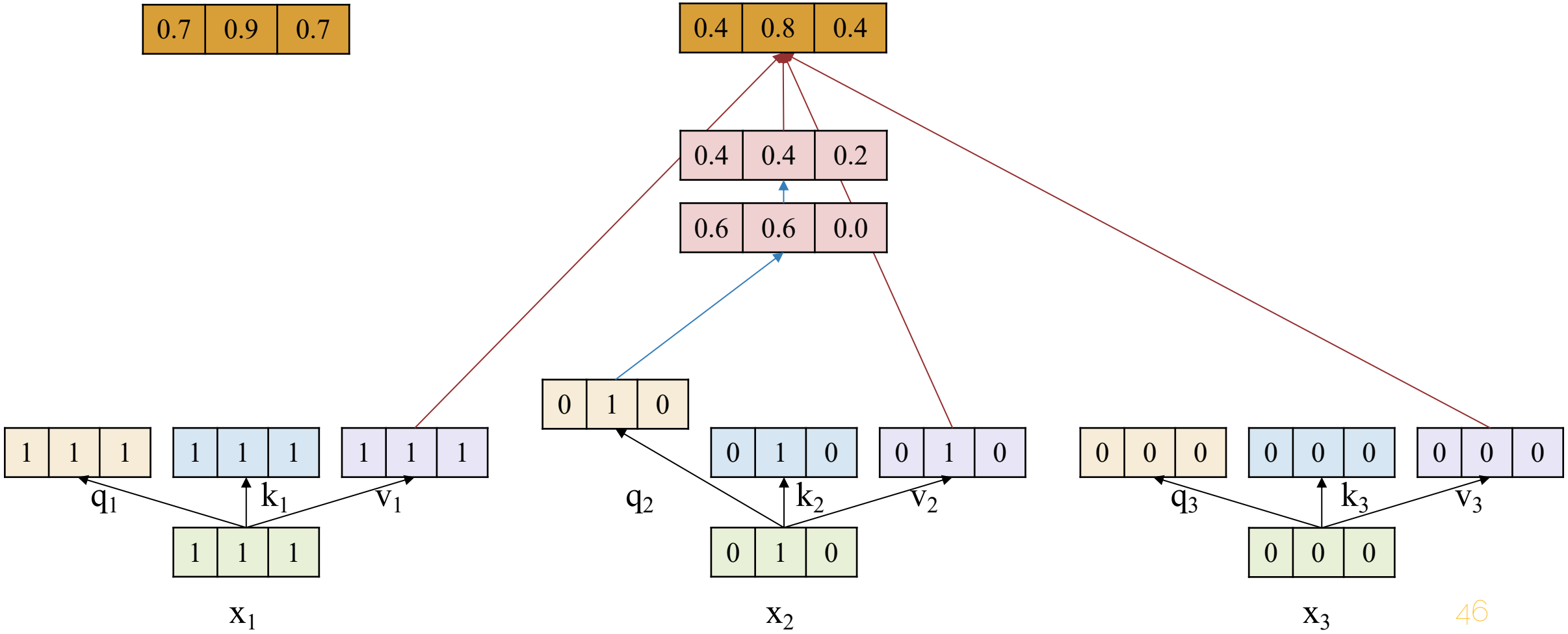
Transformer Encoder

❖ Self-attention



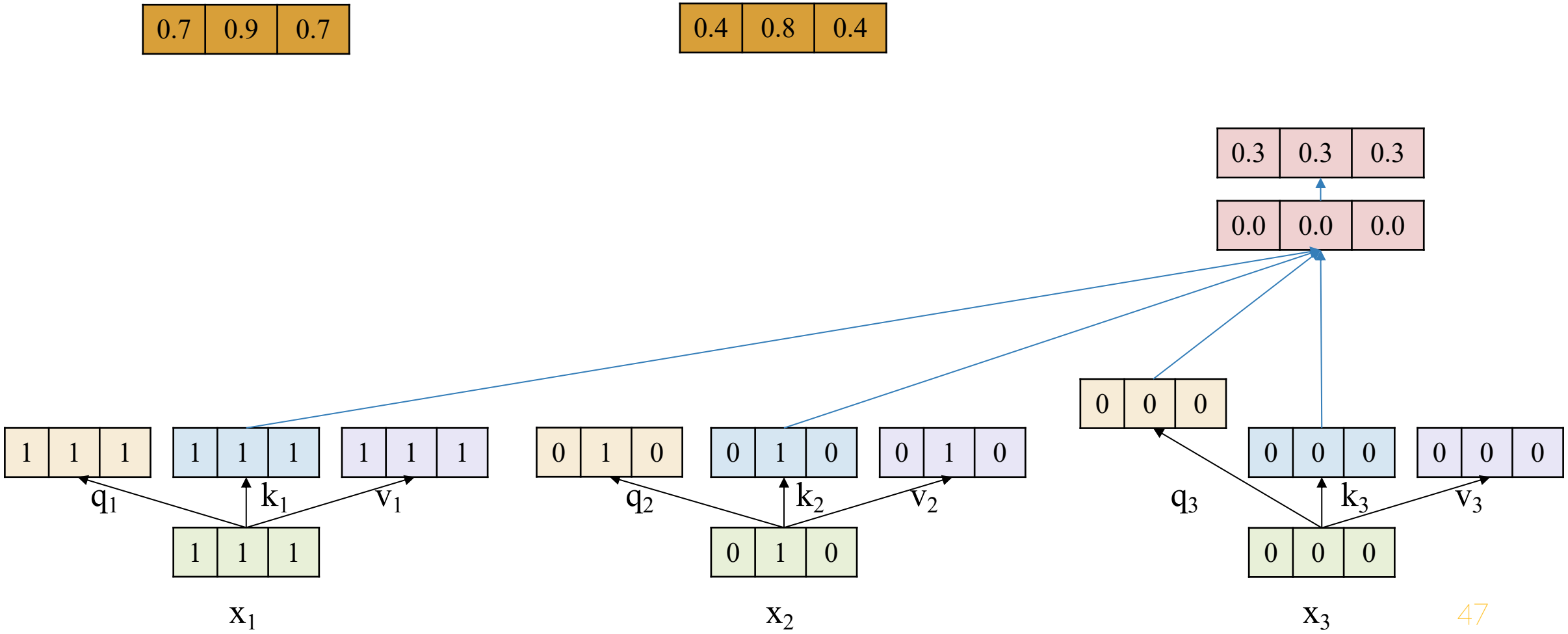
Transformer Encoder

❖ Self-attention



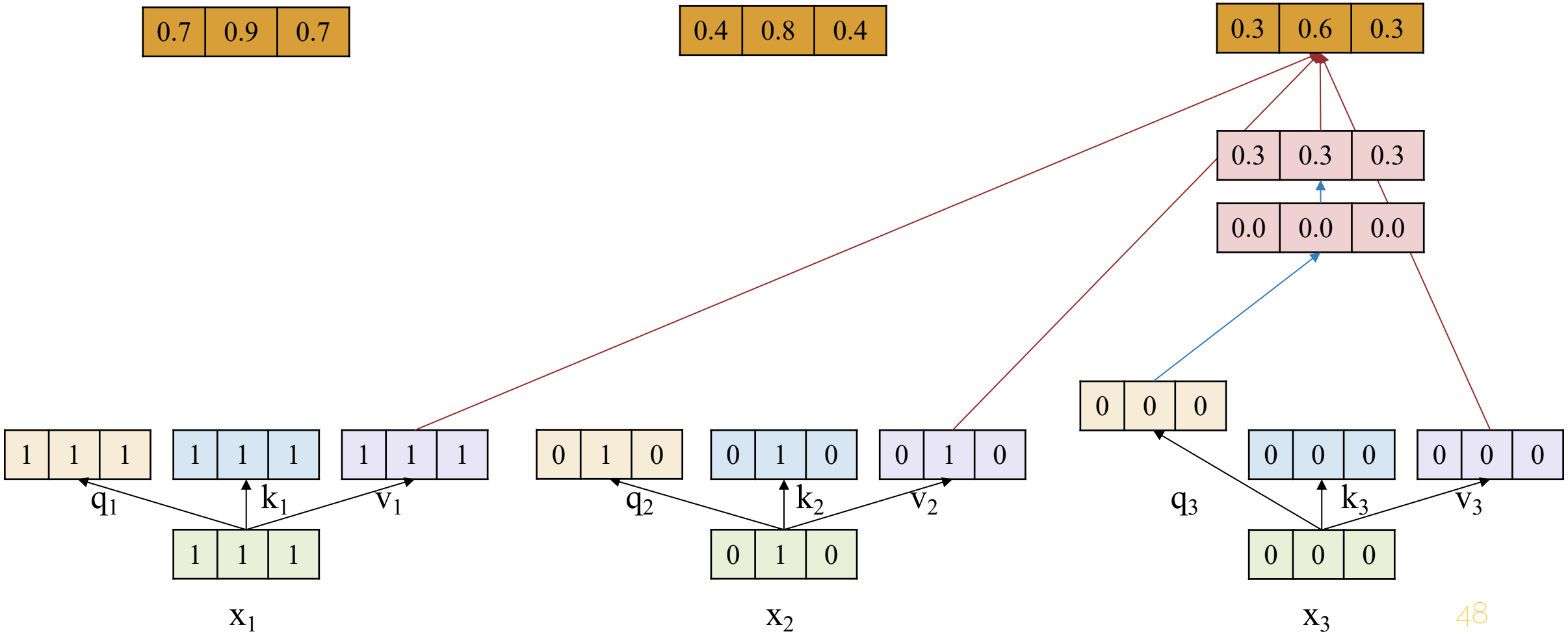
Transformer Encoder

❖ Self-attention



Transformer Encoder

❖ Self-attention

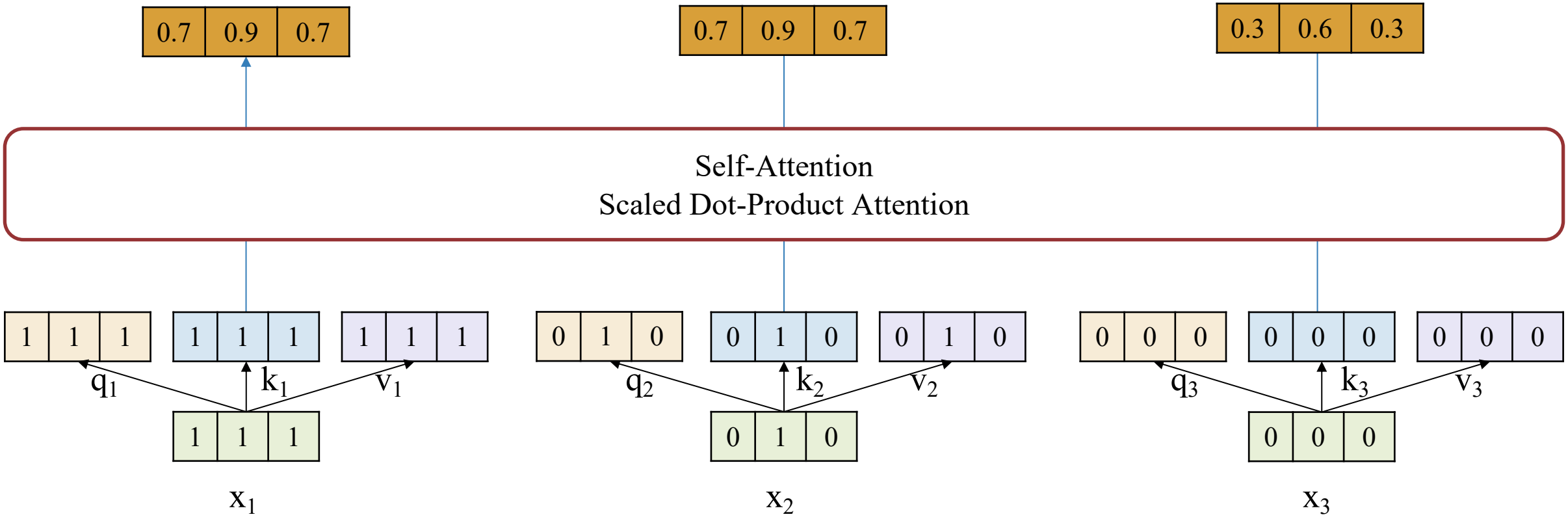


Transformer Encoder

❖ Self-attention

❖ To learn the relationship between word in the sentence

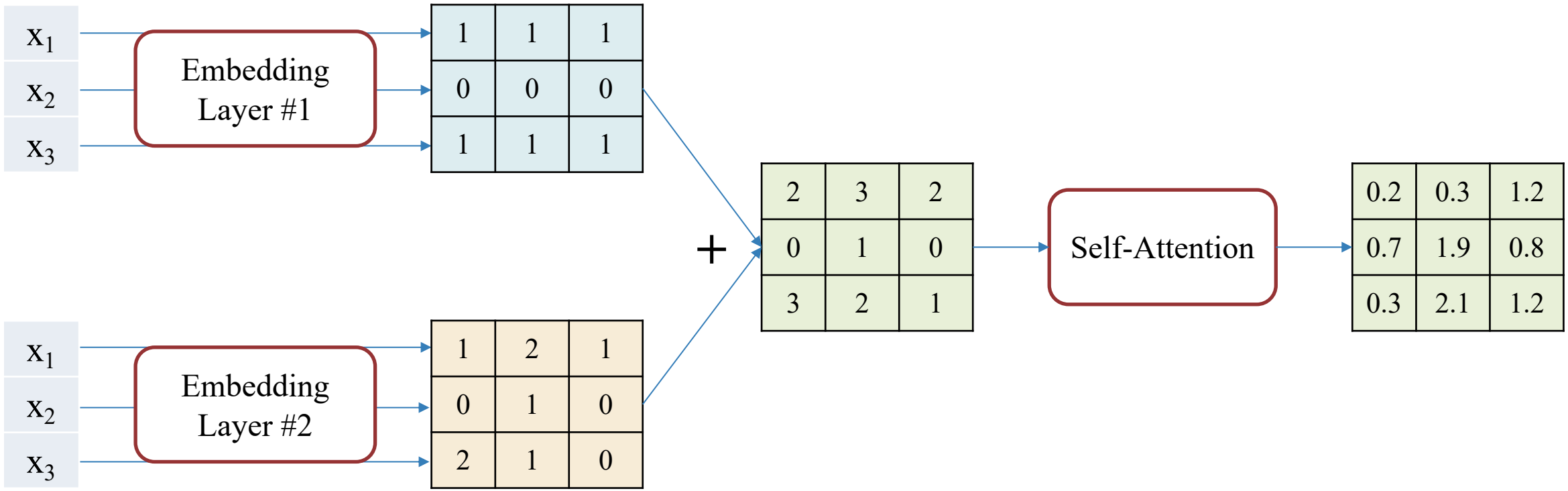
Ignore the order of words in the sentence ?



Transformer Encoder

❖ Positional Encoding

- The position of a token in a sentence as unique representation – each position is mapped to a vector
- Methods: Sinusoid; Learned positional embedding (as learned input embedding)



Transformer Encoder

❖ Positional Encoding: Demo

```python
class TokenAndPositionEmbedding(nn.Module):
    def __init__(self, vocab_size, embed_dim, max_length, device='cpu'):
        super().__init__()
        self.device = device
        self.word_emb = nn.Embedding(
            num_embeddings=vocab_size,
            embedding_dim=embed_dim
        )
        self.pos_emb = nn.Embedding(
            num_embeddings=max_length,
            embedding_dim=embed_dim
        )

    def forward(self, x):
        N, seq_len = x.size()
        positions = torch.arange(0, seq_len).expand(N, seq_len).to(self.device)
        output1 = self.word_emb(x)
        output2 = self.pos_emb(positions)
        output = output1 + output2
        return output
```

```python
vocab_size = 10000
embed_dim = 200
max_length = 50
embedding = TokenAndPositionEmbedding(
    vocab_size,
    embed_dim,
    max_length
)
```

```python
batch_size = 32

input = torch.randint(
    high=2,
    size=(batch_size, max_length),
    dtype=torch.int64
)
```

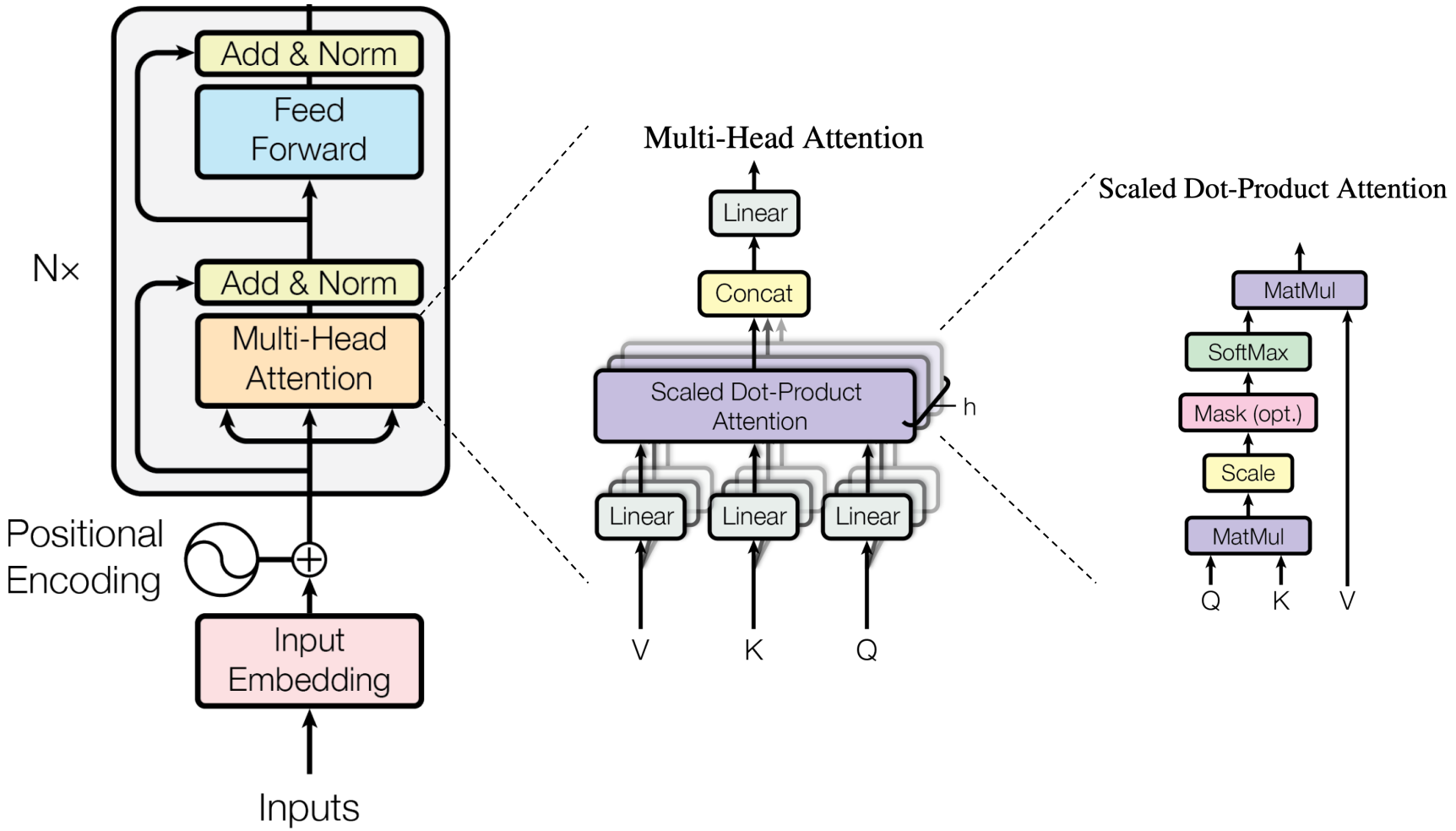
```python
embedded = embedding(input)
```

```python
embedded.shape
```

```
torch.Size([32, 50, 200])
```

Transformer Encoder

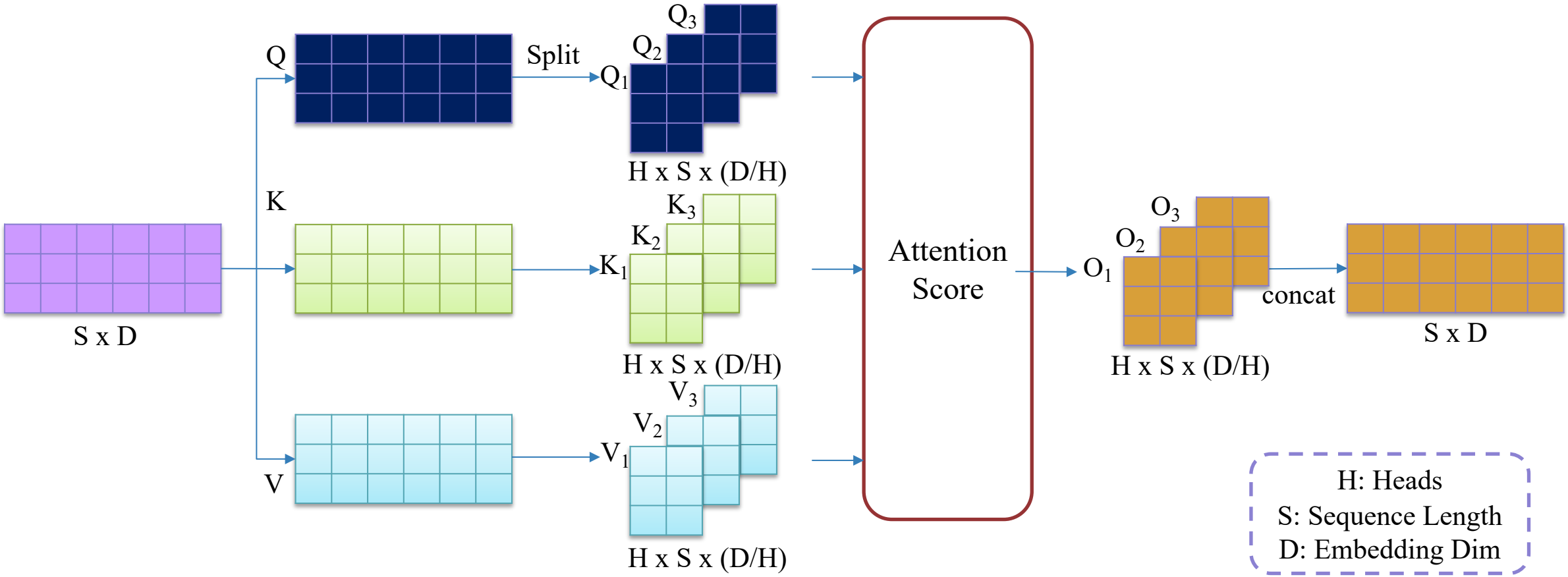
❖ Transformer Encoder



Transformer Encoder

❖ Multi-head Attention

❖ Split into the multiple attention heads (process independently) => self-attention => concat



Transformer Encoder

❖ Multi-head Attention: Demo

```
batch_size = 1
seq_len = 50
embedding_dim = 200

input = torch.randint(
    high=2,
    size=(batch_size, seq_len, embedding_dim),
    dtype=torch.float32
)
input
```

```
tensor([[[0., 1., 1.,  ..., 0., 1., 1.],
         [0., 1., 0.,  ..., 0., 0., 0.],
         [1., 0., 1.,  ..., 1., 1., 1.],
         ...,
         [0., 0., 0.,  ..., 1., 0., 0.],
         [1., 0., 1.,  ..., 0., 1., 1.],
         [0., 1., 1.,  ..., 1., 1., 1.]]])
```

```
embedding_dim = 200
num_heads = 5

att_layer = nn.MultiheadAttention(
    embed_dim=embedding_dim,
    num_heads=num_heads,
    batch_first=True
)
```

```
attn_output, attn_output_weights = att_layer(
    query=input,
    key=input,
    value=input
)
```

```
attn_output.shape
```

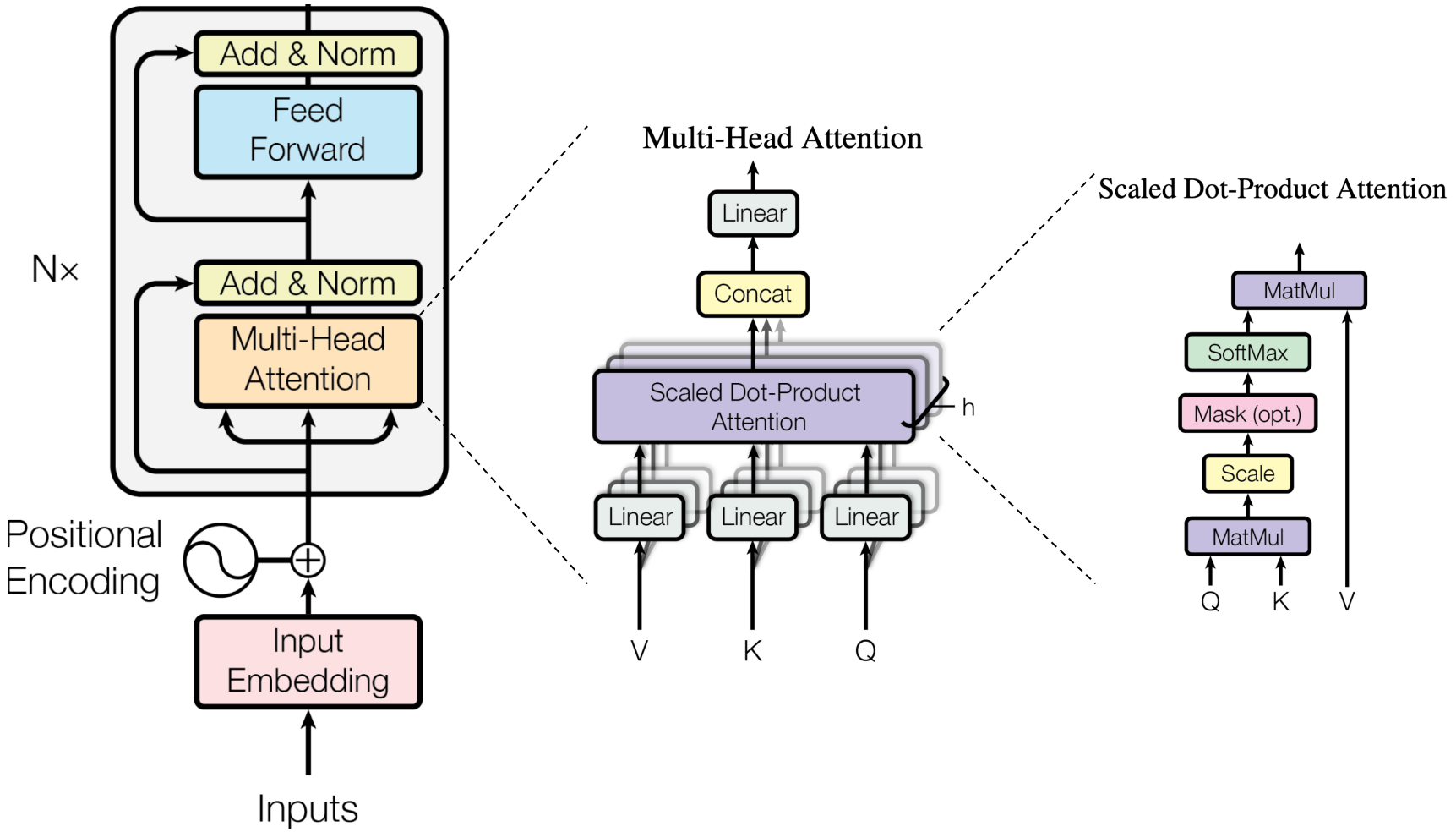
```
torch.Size([1, 50, 200])
```

```
attn_output_weights.shape
```

```
torch.Size([1, 50, 50])
```

Transformer Encoder

❖ Transformer Encoder



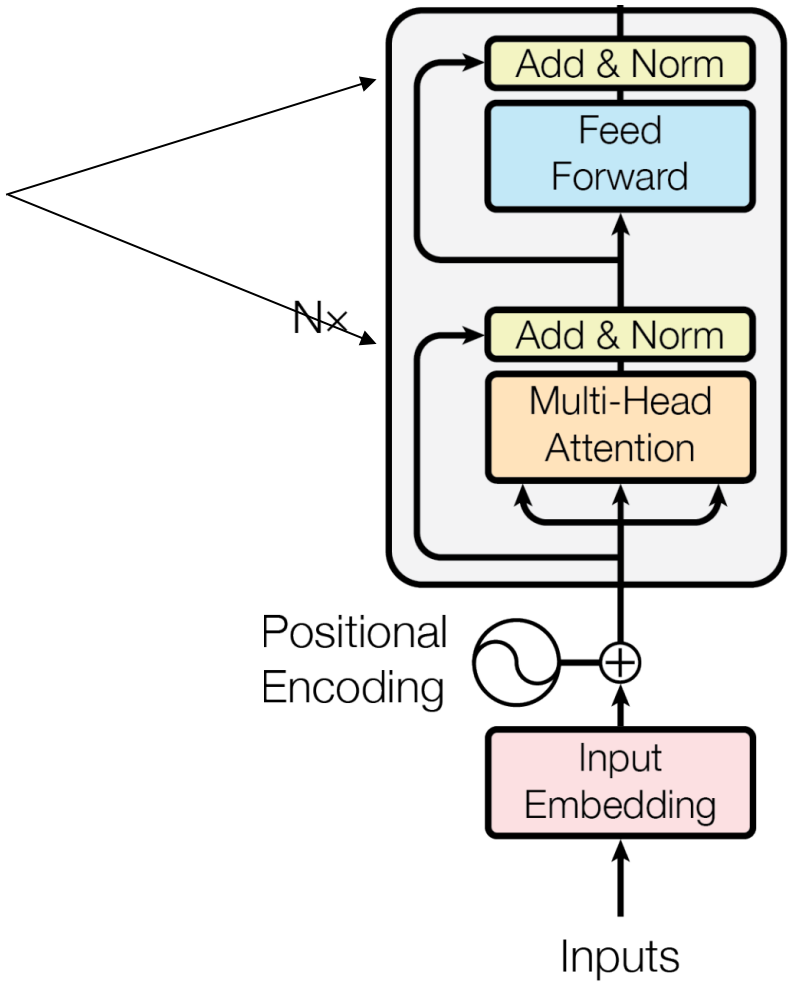
Transformer Encoder

❖ Layer Normalization

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

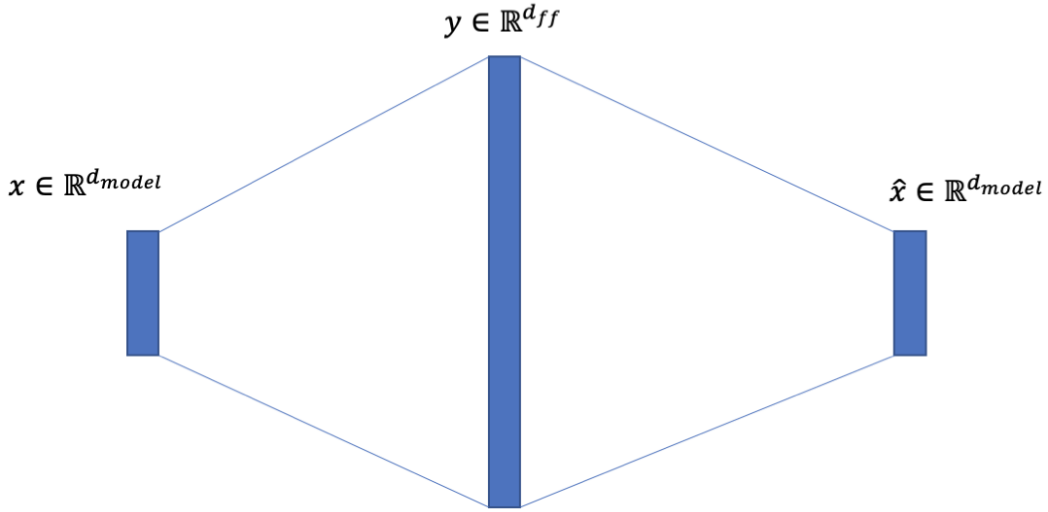
$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_{ij} - \mu_i)^2$$

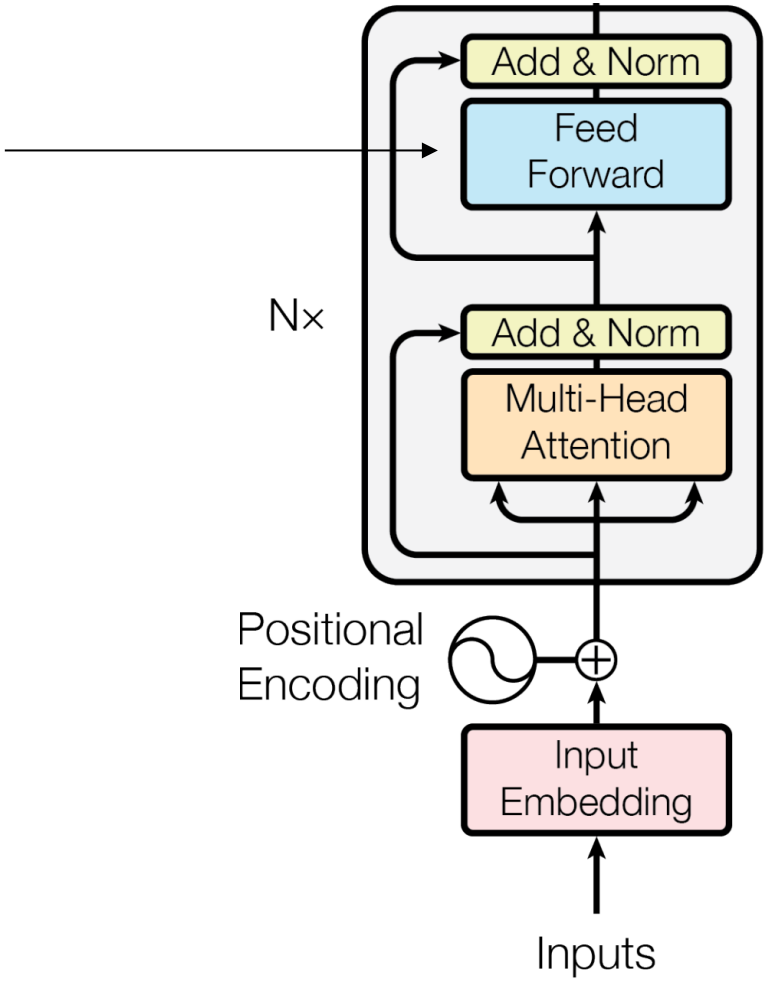
$$\hat{x}_{ij} = \frac{(x_{ij} - \mu_i)}{\sqrt{\sigma_i^2 + \epsilon}}$$



Transformer Encoder

❖ Fully-connected Layers





Transformer Encoder

❖ Transformer Encoder: Demo

```python
class TransformerEncoder(nn.Module):
    def __init__(self, embed_dim, num_heads, ff_dim, dropout=0.1):
        super().__init__()
        self.attn = nn.MultiheadAttention(
            embed_dim=embed_dim,
            num_heads=num_heads,
            batch_first=True
        )
        self.ffn = nn.Sequential(
            nn.Linear(in_features=embed_dim, out_features=ff_dim, bias=True),
            nn.ReLU(),
            nn.Linear(in_features=ff_dim, out_features=embed_dim, bias=True)
        )
        self.layernorm_1 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
        self.layernorm_2 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
        self.dropout_1 = nn.Dropout(p=dropout)
        self.dropout_2 = nn.Dropout(p=dropout)

    def forward(self, query, key, value):
        attn_output, _ = self.attn(query, key, value)
        attn_output = self.dropout_1(attn_output)
        out_1 = self.layernorm_1(query + attn_output)
        ffn_output = self.ffn(out_1)
        ffn_output = self.dropout_2(ffn_output)
        out_2 = self.layernorm_2(out_1 + ffn_output)
        return out_2
```

```python
encoder_layer = TransformerEncoder(
    embed_dim=200,
    num_heads=5,
    ff_dim=1024
)
```

```python
embedded.shape
```

```
torch.Size([32, 50, 200])
```

```python
encoded = encoder_layer(embedded, embedded, embedded)
```

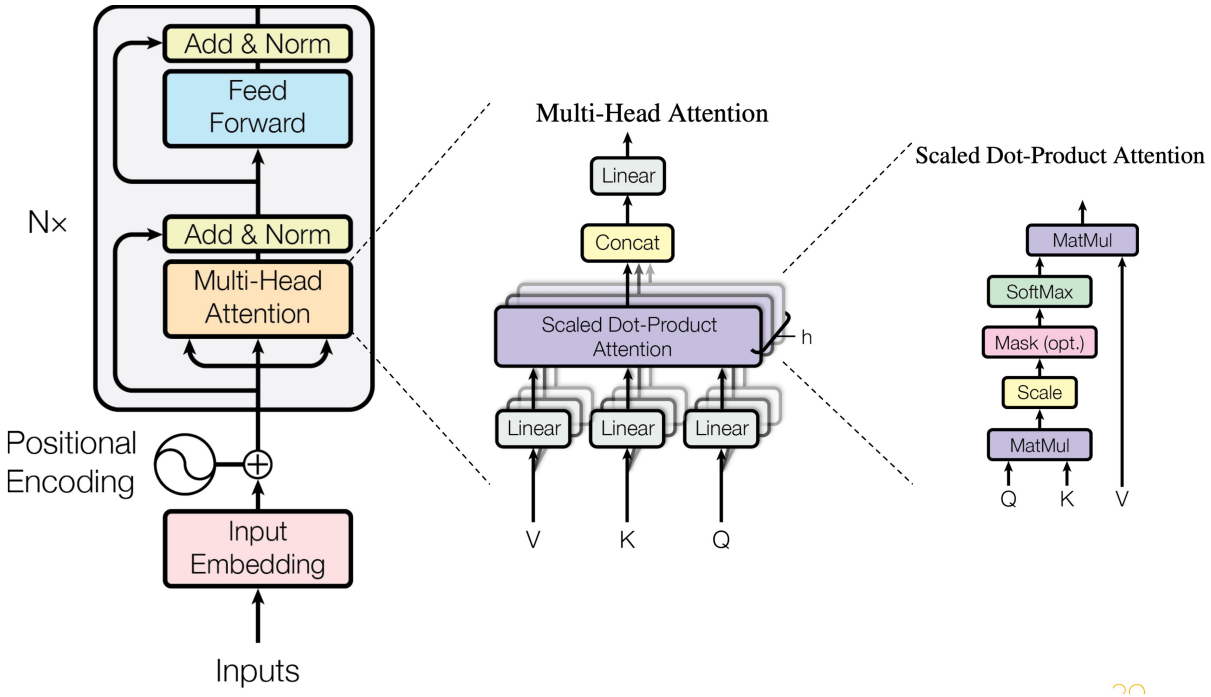
```python
encoded.shape
```

```
torch.Size([32, 50, 200])
```

Outline

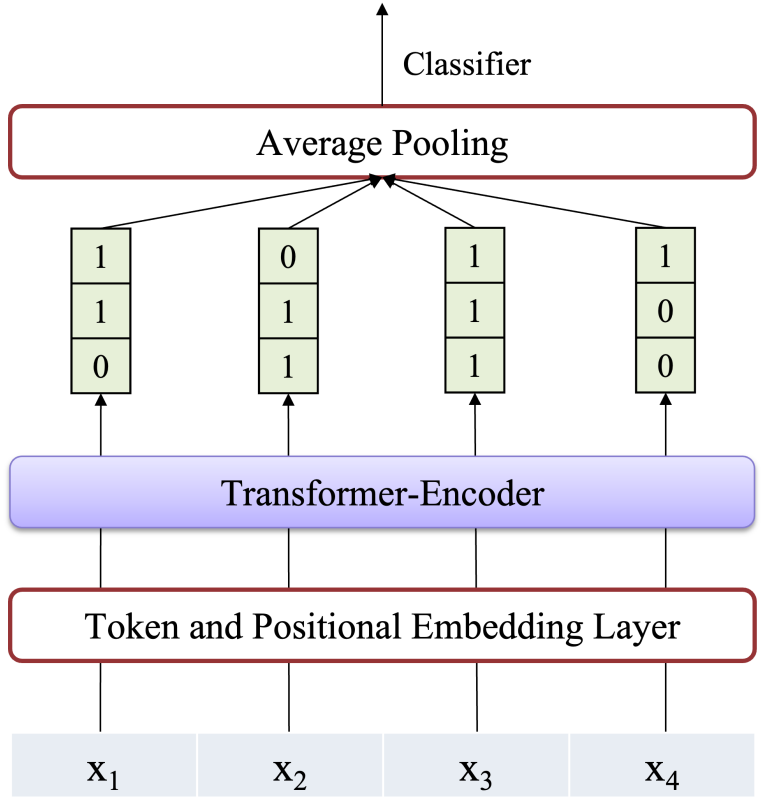
SECTION 1

Transformer-Encoder



SECTION 2

Text Classification



AI VIET NAM
@aivietnam.edu.vn

Text Classification

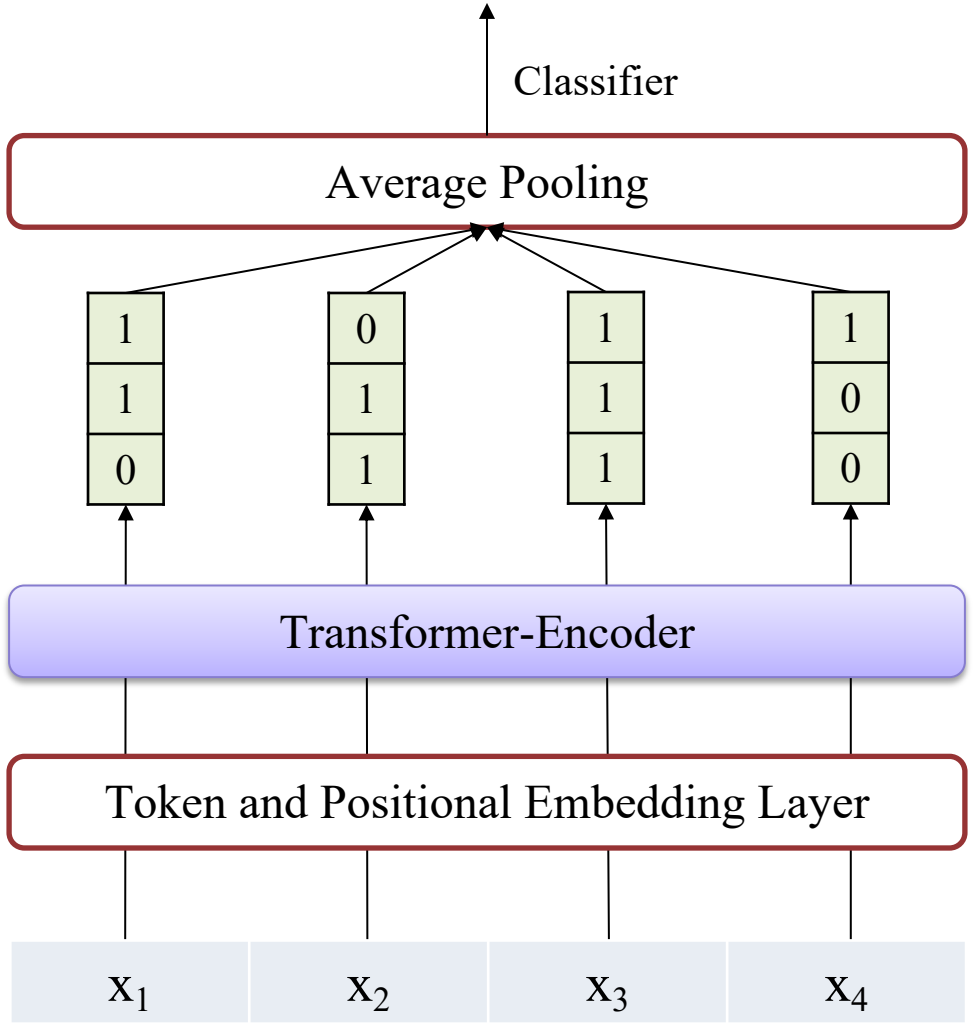
❖ Introduction

Description: Given NTC-SCV dataset, build a sentiment analysis model based on transformer-encoder architecture.

Positive Example	Negative Example
Mình được 1 cô bạn giới_thiệu đến đây , tìm địa_chỉ khá dễ . Menu nước uống chất khỏi nói . Mình muốn cũng đc 8 loại nước ở đây , món nào cũng ngon và bổ_dưỡng cả .	Quán chế_biến đồ_ăn lâu , Cá_Sapa nướng ướp rất dở , sò Lông ko tươi , nước_chấm ko ngon\n Tóm_lại sẽ ko bao_giờ ghé nữa , ăn_dở mà uổng tiền
Mỗi lần thèm trà sữa là làm 1 ly . Quán dễ kiếm , không_gian lại rộng_rãi . Nhân_viên thì dễ_thương gần_gũi . Nói_chung thèm trà sữa là mình ghé Quán ở đây vì gần nhà .	Quán này thấy khá nhiều người bảo mình nên mình đã đi ăn thử , nhưng thực_sự ăn xong thấy không được như mong_đợi lắm .

Text Classification

❖ Model Architecture



Text Classification

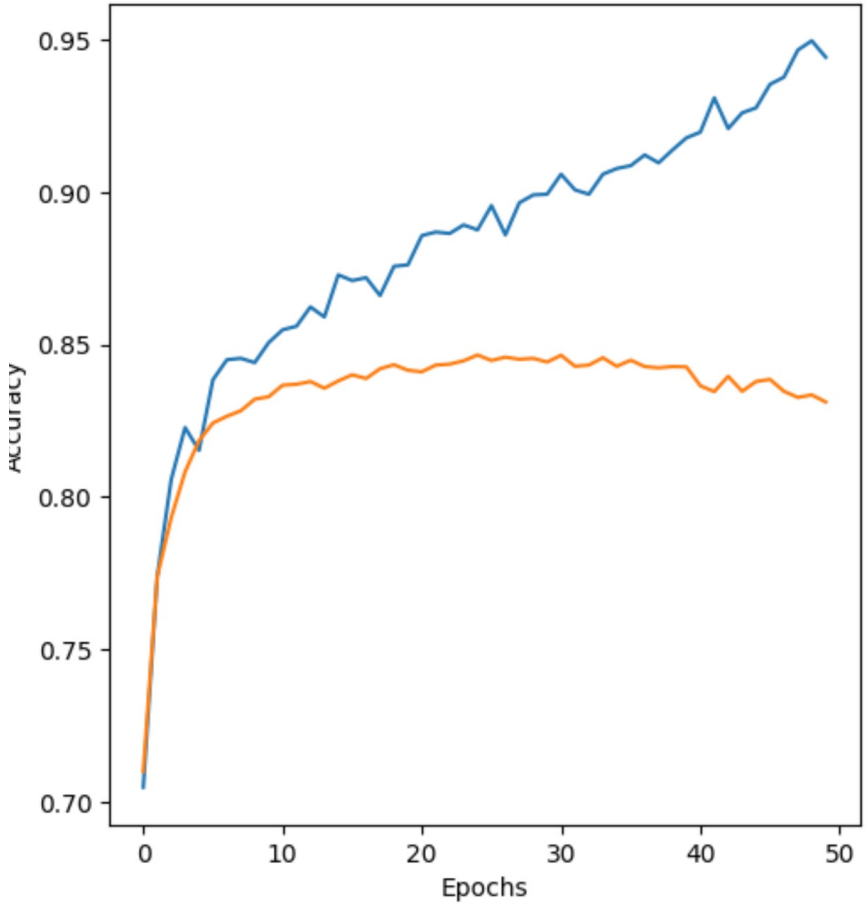
❖ Coding

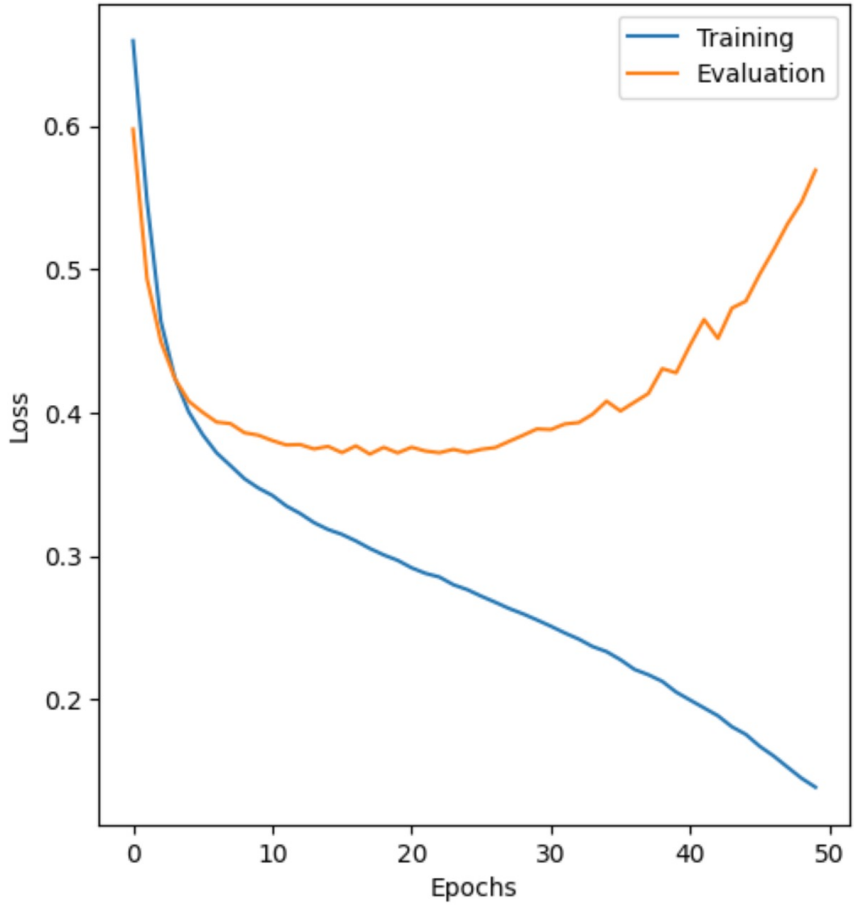
```python
class TransformerEncoder(nn.Module):
    def __init__(self, embed_dim, num_heads, ff_dim, dropout=0.1):
        super().__init__()
        self.attn = nn.MultiheadAttention(
            embed_dim=embed_dim,
            num_heads=num_heads,
            batch_first=True
        )
        self.ffn = nn.Sequential(
            nn.Linear(in_features=embed_dim, out_features=ff_dim, bias=True),
            nn.ReLU(),
            nn.Linear(in_features=ff_dim, out_features=embed_dim, bias=True)
        )
        self.layernorm_1 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
        self.layernorm_2 = nn.LayerNorm(normalized_shape=embed_dim, eps=1e-6)
        self.dropout_1 = nn.Dropout(p=dropout)
        self.dropout_2 = nn.Dropout(p=dropout)

    def forward(self, query, key, value):
        attn_output, _ = self.attn(query, key, value)
        attn_output = self.dropout_1(attn_output)
        out_1 = self.layernorm_1(query + attn_output)
        ffn_output = self.ffn(out_1)
        ffn_output = self.dropout_2(ffn_output)
        out_2 = self.layernorm_2(out_1 + ffn_output)
        return out_2
```

Text Classification

❖ Coding: Training



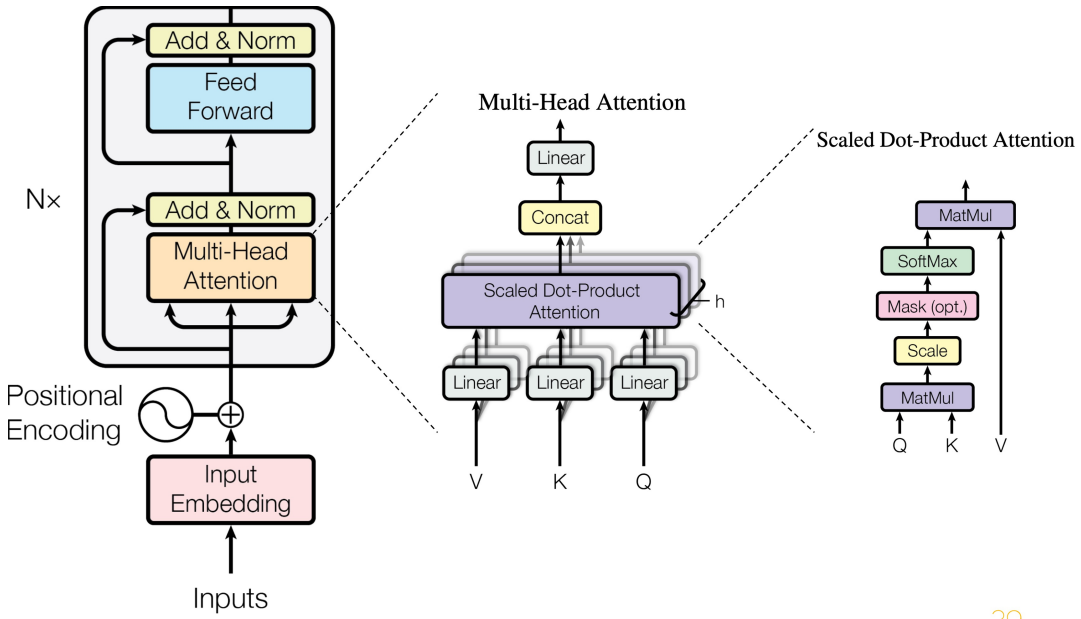


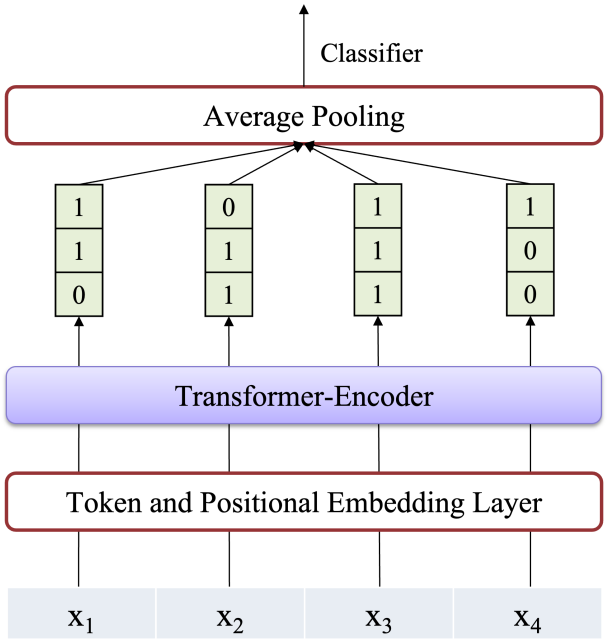

QUIZ TIME

Objectives

Transformer-Encoder

- Introduction
- Attention Mechanism
- Transformer Encoder





Text Classification

- Text Classification Problem
- Transformer Encoder for Text Classification Application
- Question

Thanks!

Any questions?