AI VIETNAM
All-in-One Course

Transformer Insight

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

Outline

- Revisiting RNNs, MLPs, and CNNs
- From RNNs to Transformers
- Self-Attention
- Masked Self-Attention
- Cross-Attention

Text Classification

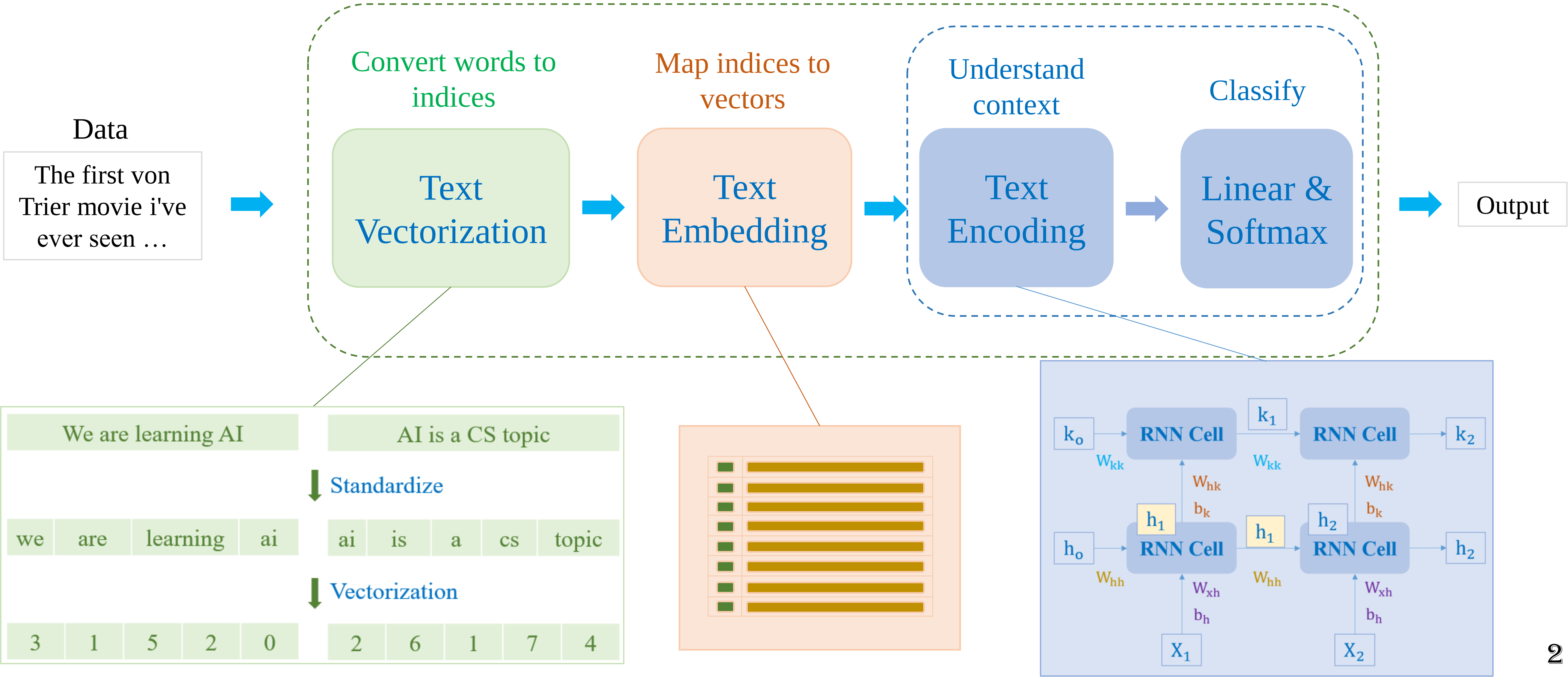
❖ IMDB dataset

- 50,000 movie review for sentiment analysis
- Consist of: + 25,000 movie review for training
  + 25,000 movie review for testing
- Label: positive – negative

"A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece….."	positive
"This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 8 years were brilliant, but things dropped off after that. By 1990, the show was not really funny anymore, and it's continued its decline further to the complete waste of time it is today…."	negative
"I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue is witty and the characters are likable (even the well bread suspected serial killer)…."	positive
"BTW Carver gets a very annoying sidekick who makes you wanna shoot him the first three minutes he's on screen."	negative

Review

❖ Text classification model



Text Classification

- 50,000 movie review for sentiment analysis
- Consist of: + 25,000 movie review for training
  + 25,000 movie review for testing
- Label: positive – negative

❖ **IMDB dataset**

```
from datasets import load_dataset

imdb = load_dataset("imdb")
train_data, test_data = imdb['train'], imdb['test']
```

```
print(train_data.shape)
print(test_data.shape)
```

```
(25000, 2)
(25000, 2)
```

```
tokenizer = get_tokenizer("basic_english")
vocab_size = 20000

def yield_tokens(data_iter):
    for data in data_iter:
        yield tokenizer(data["text"])

vocab = build_vocab_from_iterator(yield_tokens(train_data),
                                  min_freq = 3,
                                  max_tokens=vocab_size,
                                  specials=["<pad>", "<s>", "<unk>"])
vocab.set_default_index(vocab["<unk>"])
```

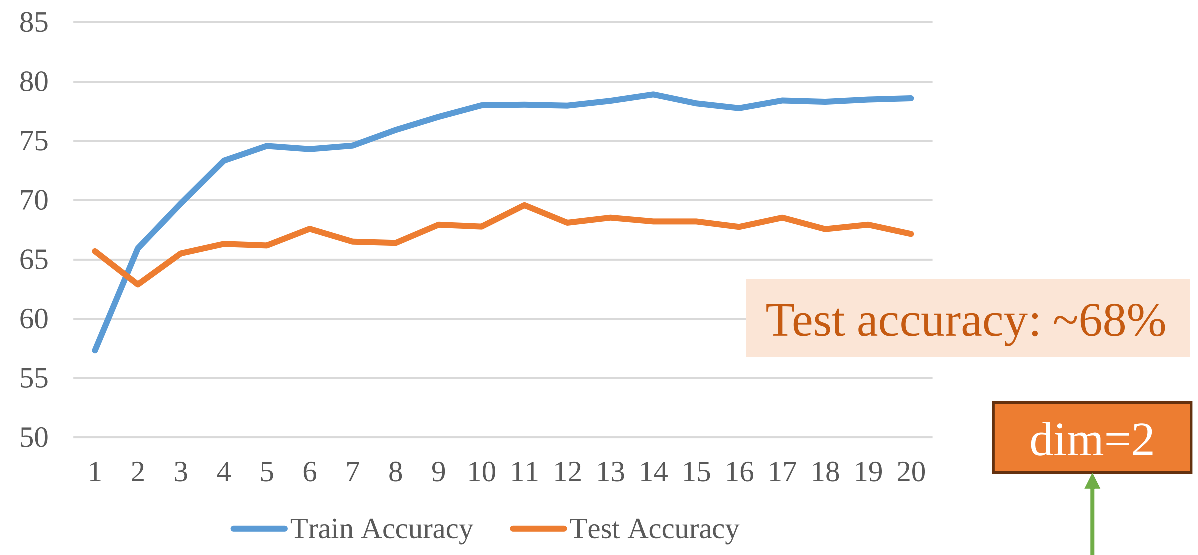
```
print(train_data[0]['text'])
```

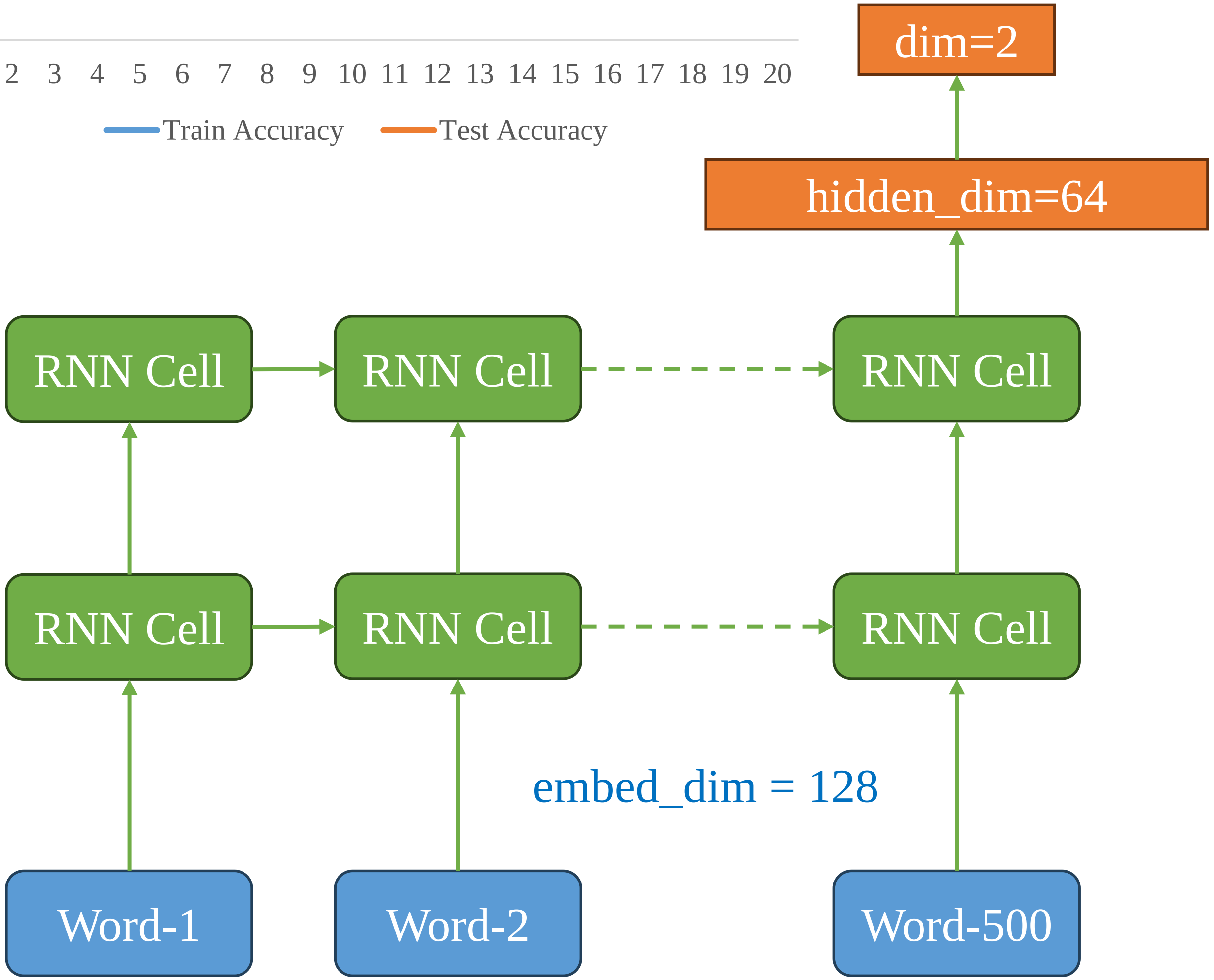
```
I rented I AM CURIOUS-YELLOW from my video st
heard that at first it was seized by U.S. cus
sial" I really had to see this for myself.<br
everything she can about life. In particular
ought about certain political issues such as
denizens of Stockholm about their opinions or
me about I AM CURIOUS-YELLOW is that 40 years
n, even then it's not shot like some cheaply
le in Swedish cinema. Even Ingmar Bergman, ar
nd the filmmakers for the fact that any sex s
o be shown in pornographic theaters in Americ
ntended) of Swedish cinema. But really, this
```

```
print(train_data[0]['label'])
```

```
0
```

Using RNN






```python
class TextClsModel(nn.Module):
    def __init__(self, vocab_size, emb_dim,
                 hidden_dim, num_layers):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, emb_dim)
        self.rnn = nn.RNN(emb_dim, hidden_dim,
                          num_layers = num_layers,
                          batch_first = True)
        self.fc = nn.Linear(hidden_dim, 2)

    def forward(self, x):
        x = self.embedding(x)
        _, hidden = self.rnn(x)
        last_hidden = hidden[-1,:,:]
        x = self.fc(last_hidden)
        return x
```

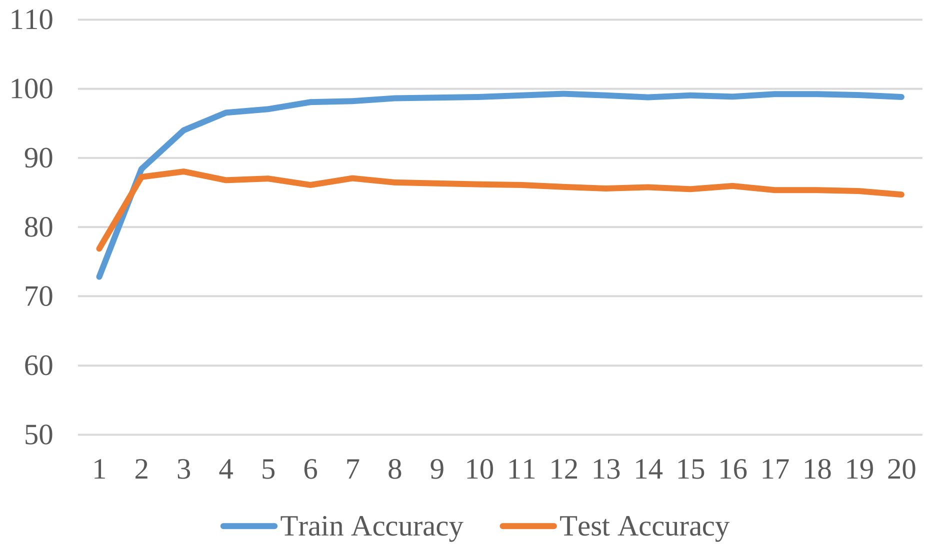
```
==========================================================================
Layer (type:depth-idx)                    Output Shape
==========================================================================
├─Embedding: 1-1                          [-1, 500, 128]
├─RNN: 1-2                                [-1, 500, 64]
├─Linear: 1-3                             [-1, 2]
==========================================================================
```

Text Deep Models

❖ Bidirectional RNN/LSTM

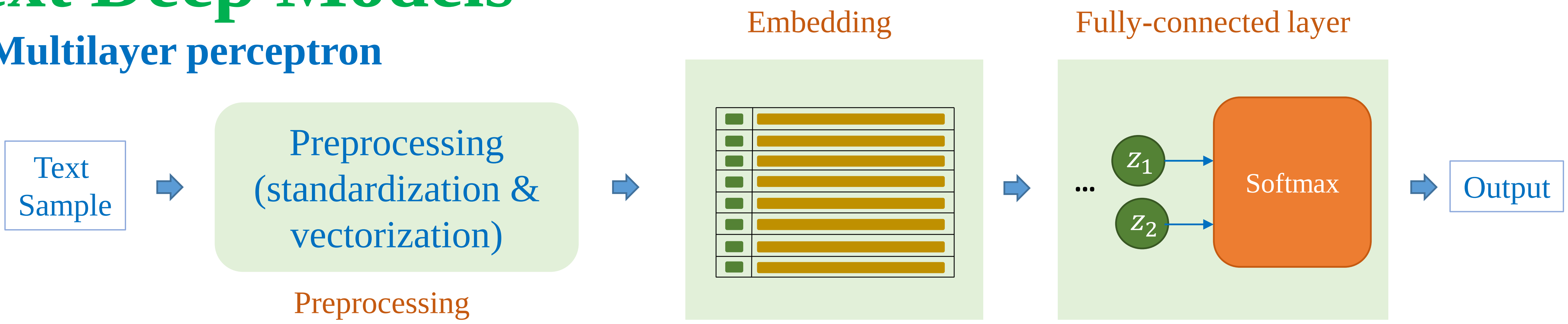
```
class TextClsModel(nn.Module):
    def __init__(self, vocab_size, emb_dim,
                 hidden_dim, num_layers):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size,
                                      emb_dim)
        self.lstm = nn.LSTM(emb_dim, hidden_dim,
                            num_layers = 2,
                            bidirectional = True,
                            batch_first = True)
        self.fc = nn.Linear(hidden_dim, 2)
    def forward(self, x):
        x = self.embedding(x)
        _, (hidden, _) = self.lstm(x)
        last_hidden = hidden[-1,:,:]
        x = self.fc(last_hidden)
        return x
```

(LSTM) Test accuracy: ~88%



Text Deep Models

❖ Multilayer perceptron




```python
class TextClsModel(nn.Module):
    def __init__(self, vocab_size, seq_len, emb_dim, hiden_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, emb_dim)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(seq_len*emb_dim, hiden_dim)
        self.fc2 = nn.Linear(hiden_dim, 128)
        self.fc3 = nn.Linear(128, 2)

    def forward(self, x):
        x = self.embedding(x)
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
==========================================================
Layer (type:depth-idx)                   Output Shape
==========================================================
├─Embedding: 1-1                         [-1, 500, 128]
├─Flatten: 1-2                           [-1, 64000]
├─Linear: 1-3                            [-1, 1024]
├─Linear: 1-4                            [-1, 128]
├─Linear: 1-5                            [-1, 2]
==========================================================
Total params: 68,228,482
Trainable params: 68,228,482
Non-trainable params: 0
Total mult-adds (M): 68.23
```

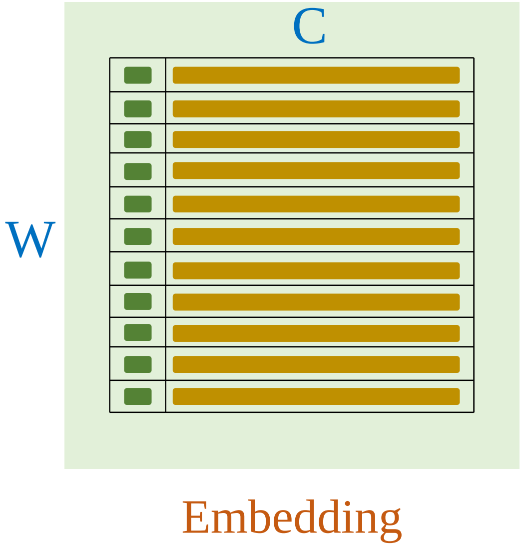
Text Deep Models

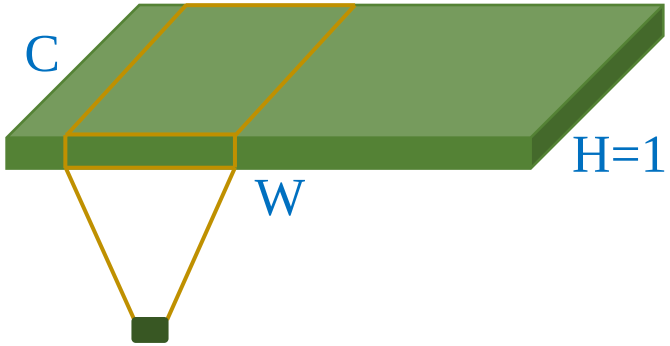
❖ Using Conv1D

```
class TextClsModel(nn.Module):
    def __init__(self, vocab_size, emb_dim, hiden_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, emb_dim)
        self.conv1 = nn.Conv1d(emb_dim, 128,
                               kernel_size=7, stride=3)
        self.conv2 = nn.Conv1d(128, 256,
                               kernel_size=7, stride=3)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(256*53, hiden_dim)
        self.fc2 = nn.Linear(hiden_dim, 2)

    def forward(self, x):
        x = self.embedding(x)
        x = x.permute(0, 2, 1)
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

input size (N, C_{in}, L) and output $(N, C_{\text{out}}, L_{\text{out}})$






```
==========================================================================
Layer (type:depth-idx)                   Output Shape
==========================================================================
├─Embedding: 1-1                         [-1, 500, 128]
├─Conv1d: 1-2                            [-1, 128, 165]
├─Conv1d: 1-3                            [-1, 256, 53]
├─Flatten: 1-4                           [-1, 13568]
├─Linear: 1-5                            [-1, 128]
├─Linear: 1-6                            [-1, 2]
==========================================================================
Total params: 4,641,538
Trainable params: 4,641,538
Non-trainable params: 0
Total mult-adds (M): 35.38
```

Global Pooling

Max pooling

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

2x2 max pooling (

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

) =

m_1	m_2
m_3	m_4

$$m_1 = \max(v_1, v_2, v_5, v_6)$$

$$m_2 = \max(v_3, v_4, v_7, v_8)$$

$$m_3 = \max(v_9, v_{10}, v_{13}, v_{14})$$

$$m_4 = \max(v_{11}, v_{12}, v_{15}, v_{16})$$

Global max pooling

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

global max pooling (

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

) =

m

$$m = \max(v_1, v_2, \ldots, v_{16})$$

max_pool1d(input_size)

Text Deep Models

❖ Conv1D + GlobalMaxPooling1D

```python
class TextClsModel(nn.Module):
    def __init__(self, vocab_size, emb_dim, hiden_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, emb_dim)
        self.conv1 = nn.Conv1d(emb_dim, 128,
                               kernel_size=7, stride=3)
        self.conv2 = nn.Conv1d(128, 128,
                               kernel_size=7, stride=3)
        self.fc1 = nn.Linear(128, hiden_dim)
        self.fc2 = nn.Linear(hiden_dim, 2)

    def forward(self, x):
        x = self.embedding(x)
        x = F.relu(self.conv1(x.permute(0, 2, 1)))
        x = F.relu(self.conv2(x))
        x = F.max_pool1d(x, kernel_size=x.size(-1)).squeeze(-1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

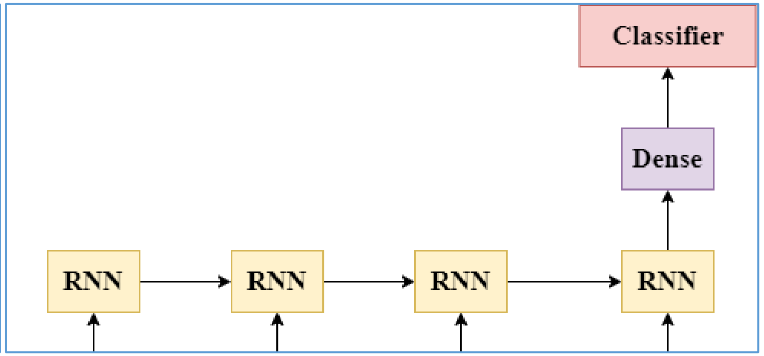
```
==========================================================================
Layer (type:depth-idx)                          Output Shape
==========================================================================
├─Embedding: 1-1                                [-1, 500, 128]
├─Conv1d: 1-2                                   [-1, 128, 165]
├─Conv1d: 1-3                                   [-1, 128, 53]
├─Linear: 1-4                                   [-1, 128]
├─Linear: 1-5                                   [-1, 2]
==========================================================================
Total params: 2,806,402
Trainable params: 2,806,402
Non-trainable params: 0
Total mult-adds (M): 27.58
==========================================================================
Input size (MB): 0.12
Forward/backward pass size (MB): 0.70
Params size (MB): 10.71
Estimated Total Size (MB): 11.53
==========================================================================
```

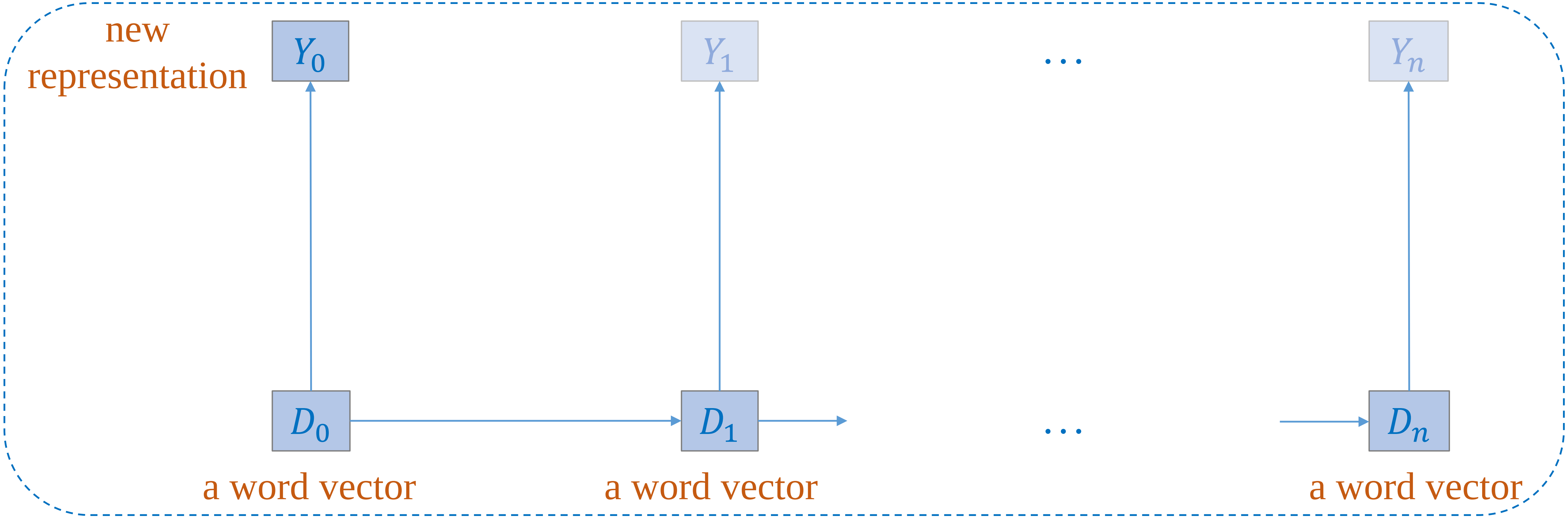
Outline

- Revisiting RNNs, MLPs, and CNNs
- From RNNs to Transformers
- Self-Attention
- Masked Self-Attention
- Cross-Attention

From RNNs to Transformers

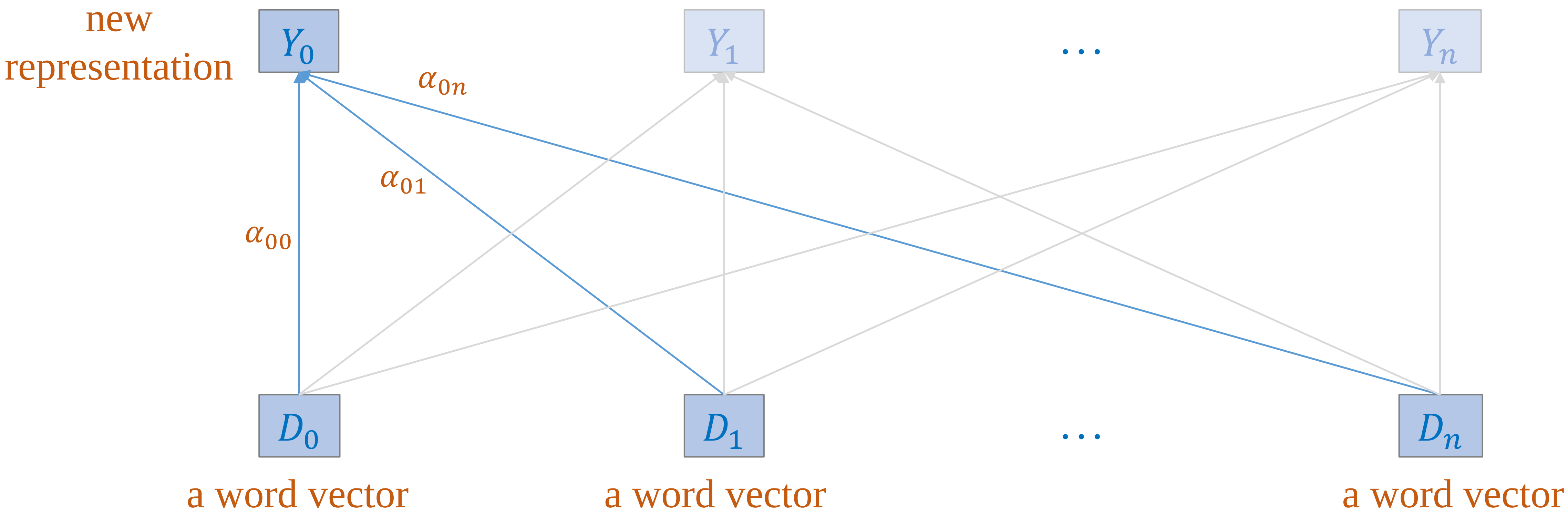
❖ RNN/LSTM/GRU Limitations





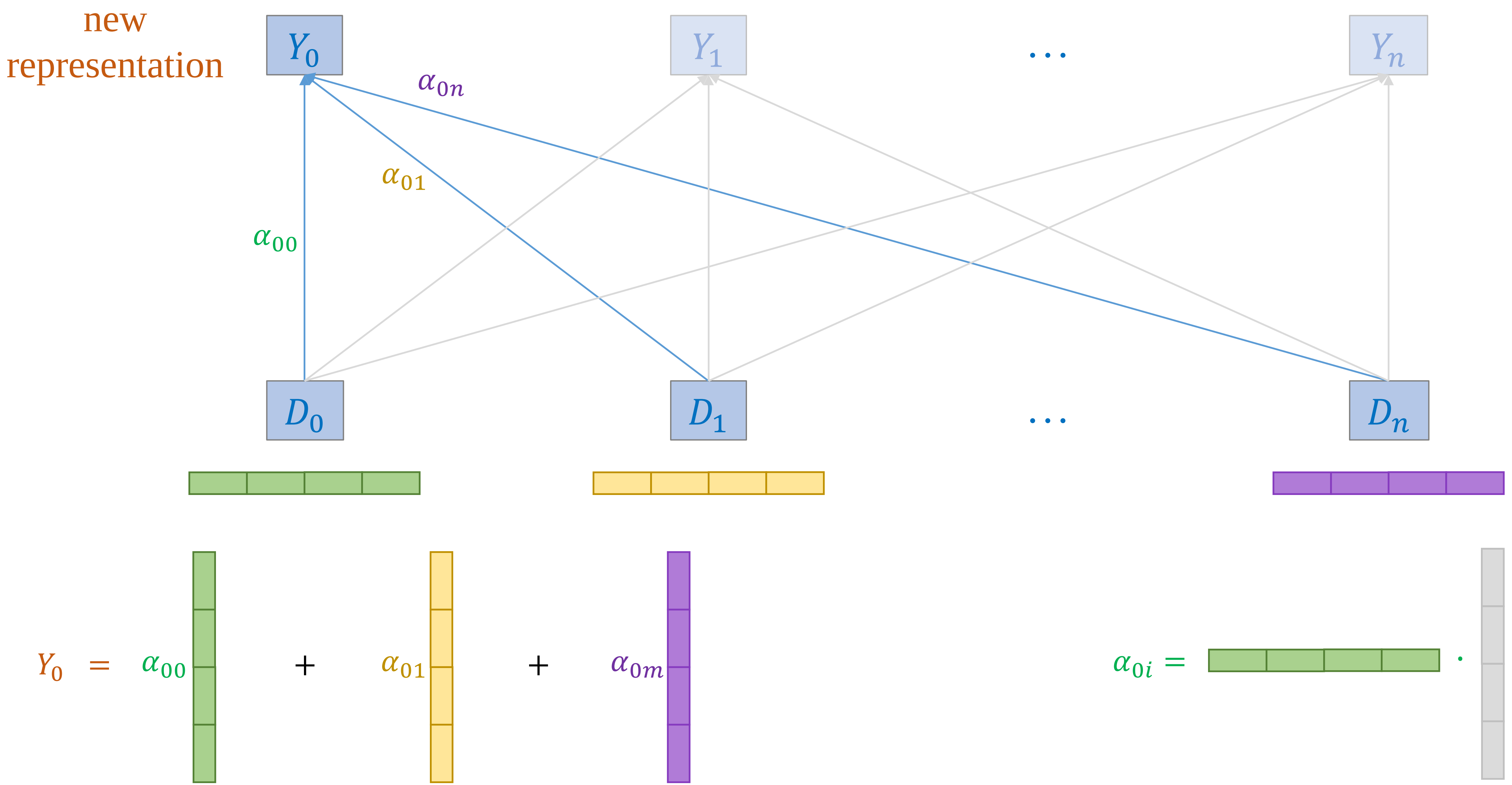
From RNNs to Transformers

❖ Desire properties

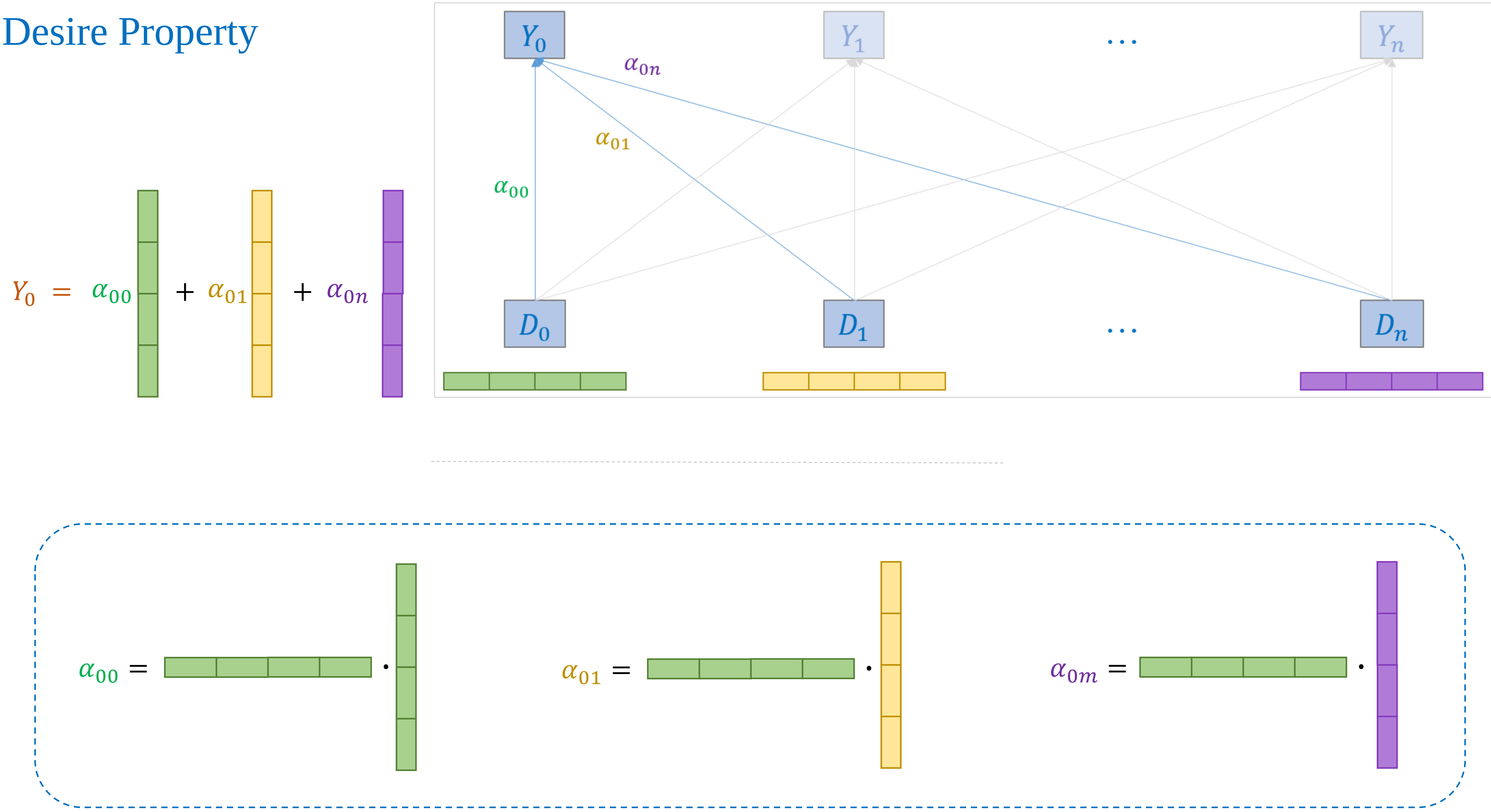


$$Y_0 = \alpha_{00}D_0 + \alpha_{01}D_1 + \cdots + \alpha_{0n}D_n$$

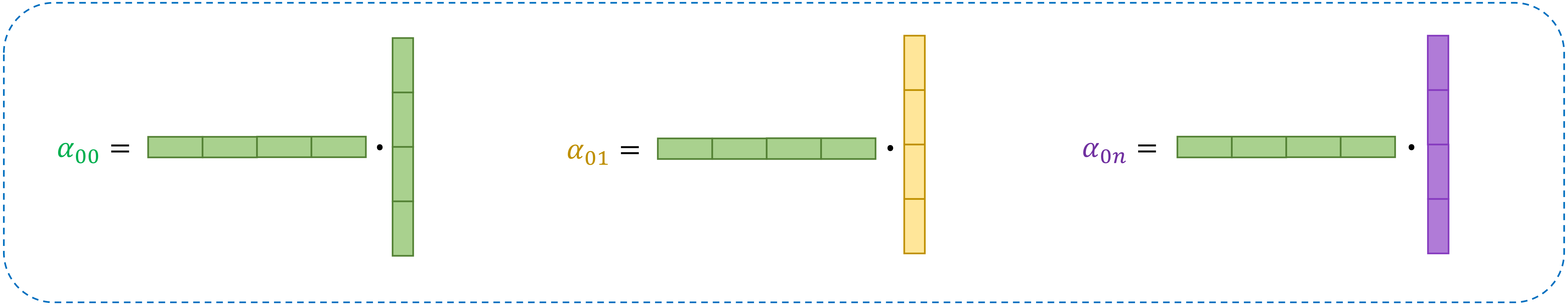
Desire Property

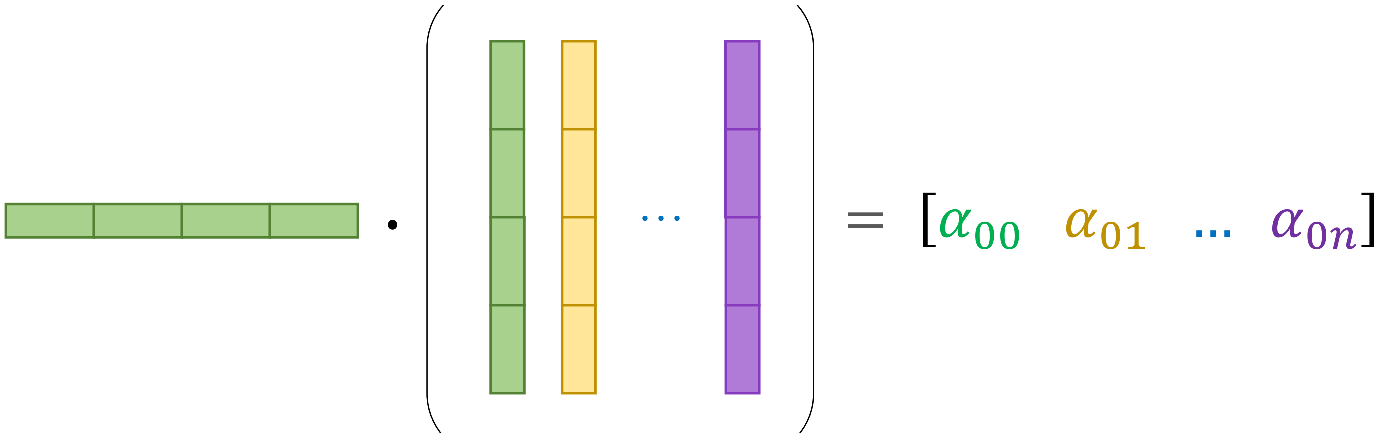


Desire Property

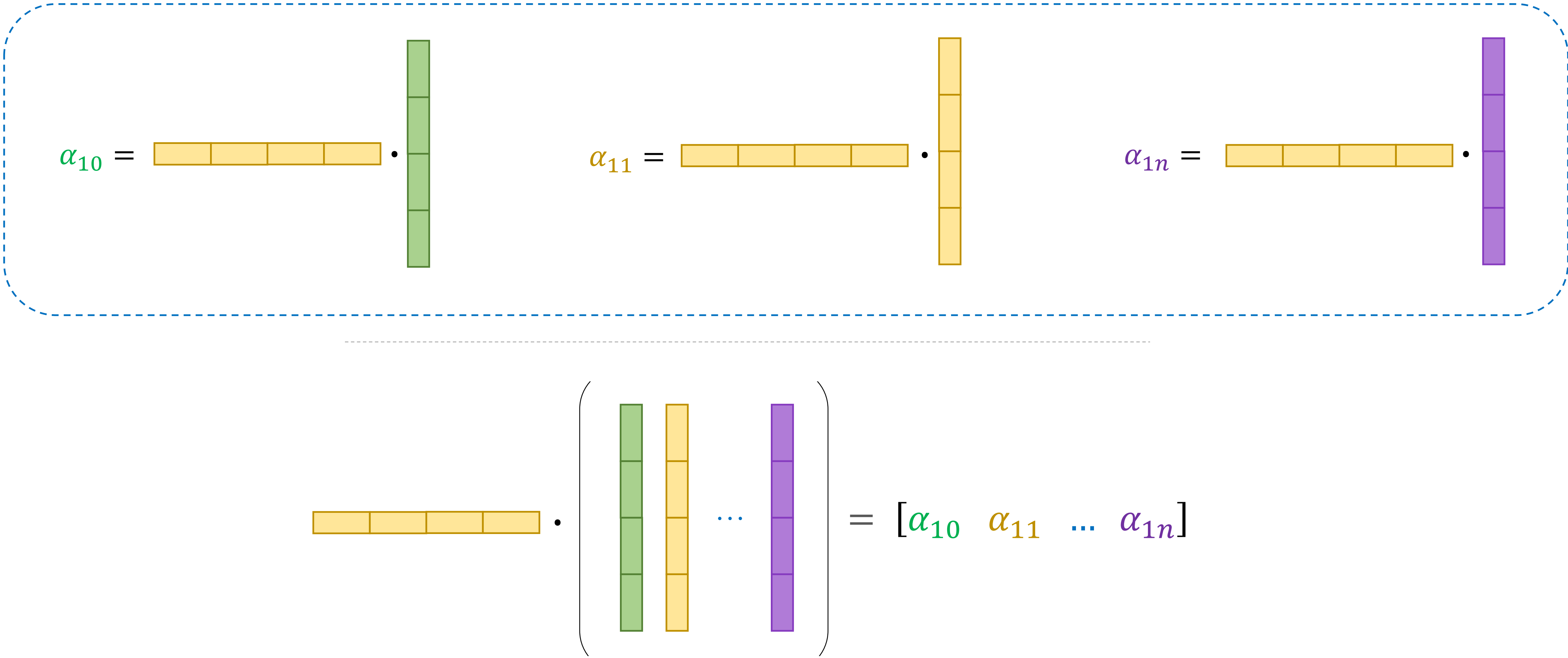


Context Awareness

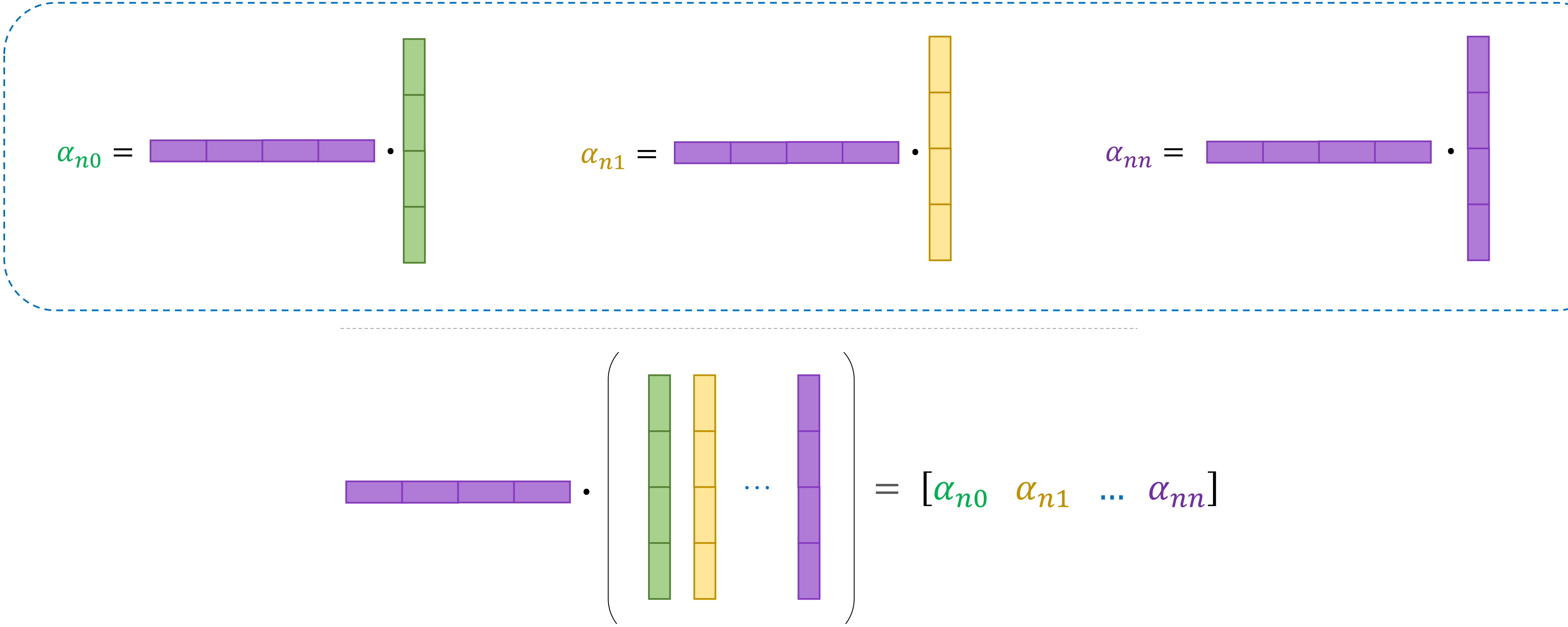


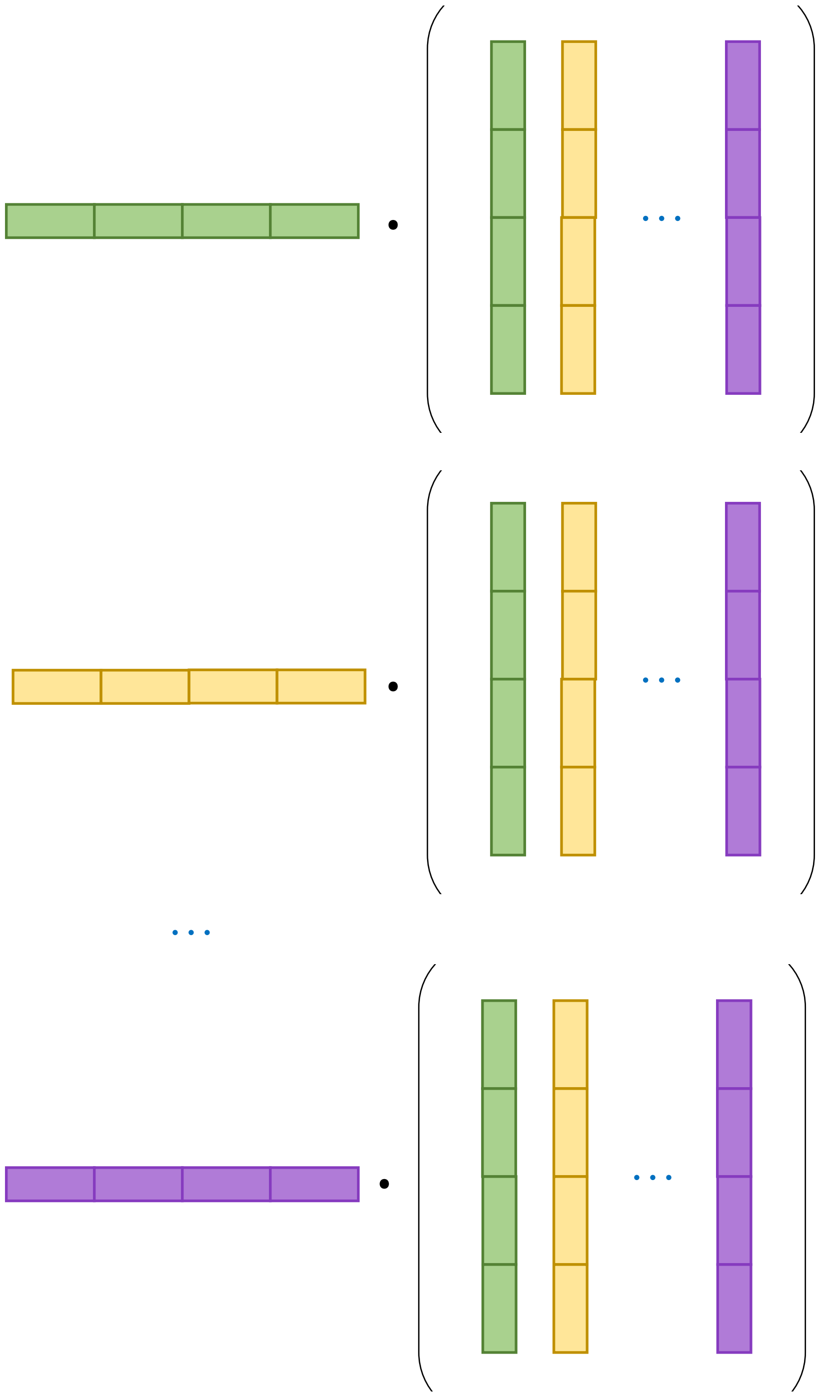


Context Awareness



Context Awareness



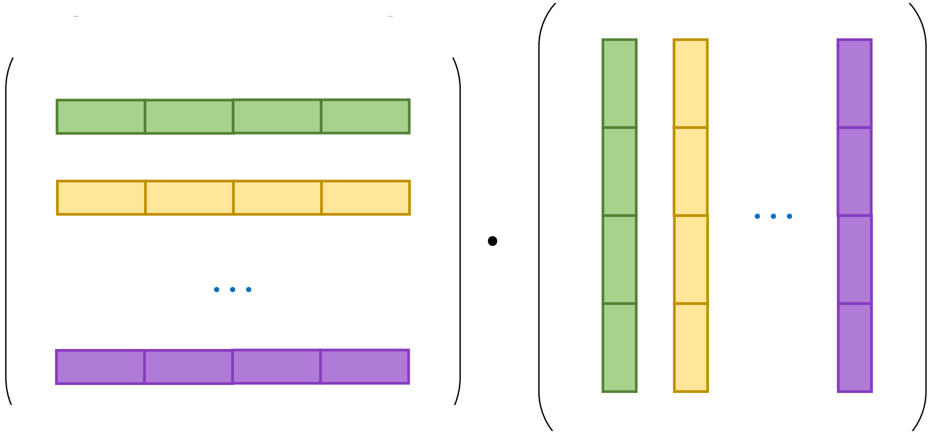


$= [\alpha_{00} \quad \alpha_{01} \quad \dots \quad \alpha_{0n}]$

$= [\alpha_{10} \quad \alpha_{11} \quad \dots \quad \alpha_{1n}]$

$\dots$

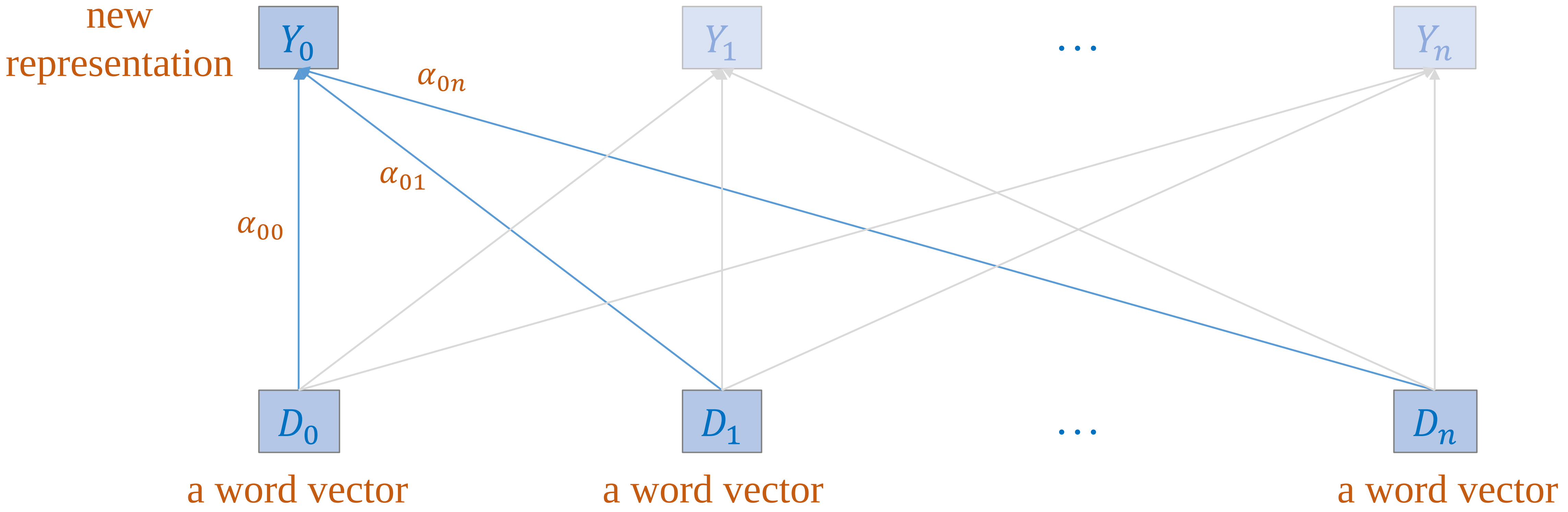
$= [\alpha_{n0} \quad \alpha_{n1} \quad \dots \quad \alpha_{nn}]$



$$= \begin{bmatrix} \alpha_{00} & \alpha_{01} & \dots & \alpha_{0n} \\ \alpha_{10} & \alpha_{11} & \dots & \alpha_{1n} \\ & & \dots & \\ \alpha_{n0} & \alpha_{n1} & \dots & \alpha_{nn} \end{bmatrix}$$

$$= \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \dots \\ \alpha_n \end{bmatrix}$$

$$\alpha = DD^T$$



$$Y_0 = \alpha_{00} D_0 + \alpha_{01} D_1 + \cdots + \alpha_{0n} D_n$$

weight: determine how much X_0 contributes to Y_0

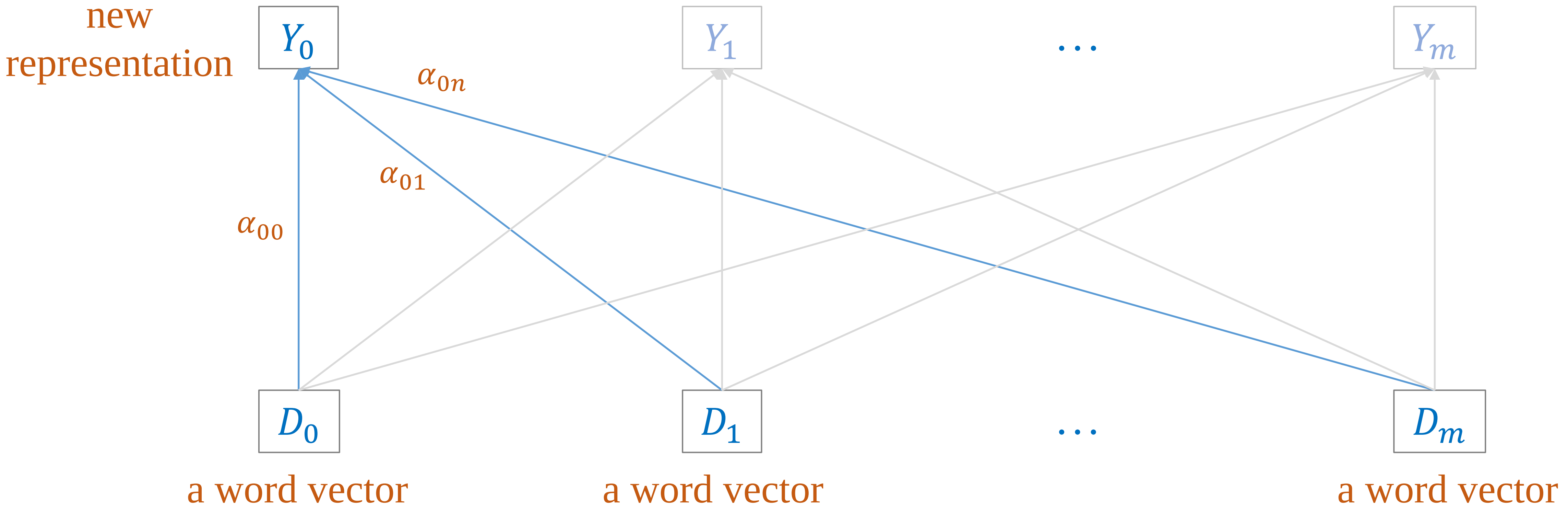
$$\begin{pmatrix} (\alpha_{00}, \alpha_{01}, \dots, \alpha_{0n}) \\ (\alpha_{10}, \alpha_{11}, \dots, \alpha_{1n}) \\ \dots \\ (\alpha_{n0}, \alpha_{n1}, \dots, \alpha_{nn}) \end{pmatrix}$$

$$\downarrow$$

$$\begin{pmatrix} \mathrm{softmax}(\alpha_{00}, \alpha_{01}, \dots, \alpha_{0n}) \\ \mathrm{softmax}(\alpha_{10}, \alpha_{11}, \dots, \alpha_{1n}) \\ \dots \\ \mathrm{softmax}(\alpha_{n0}, \alpha_{n1}, \dots, \alpha_{nn}) \end{pmatrix} \Rightarrow \alpha = \mathrm{softmax}(DD^T)$$

normalization

1



$$Y_0 = \alpha_{00} D_0 + \alpha_{01} D_1 + \cdots + \alpha_{0n} D_n$$

weight: determine how much X_0 contributes to Y_0

$$\begin{pmatrix} (\alpha_{00}, \alpha_{01}, \ldots, \alpha_{0n}) \\ (\alpha_{10}, \alpha_{11}, \ldots, \alpha_{1n}) \\ \ldots \\ (\alpha_{m0}, \alpha_{m1}, \ldots, \alpha_{nn}) \end{pmatrix}$$

$$\begin{pmatrix} \mathrm{softmax}(\alpha_{00}, \alpha_{01}, \ldots, \alpha_{0n}) \\ \mathrm{softmax}(\alpha_{10}, \alpha_{11}, \ldots, \alpha_{1n}) \\ \ldots \\ \mathrm{softmax}(\alpha_{n0}, \alpha_{n1}, \ldots, \alpha_{nn}) \end{pmatrix}$$

weight vector

2

$$\alpha = \mathrm{softmax}\left(\frac{DD^T}{\sqrt{d}}\right)$$

Context Awareness

❖ Contextualized representation

$$Y_0 = \alpha_{00}D_0 + \alpha_{01}D_1 + \cdots + \alpha_{0n}D_n$$

weight: determine how much X_0 contributes to Y_0

$$\begin{pmatrix} (\alpha_{00}, \alpha_{01}, \dots, \alpha_{0n}) \\ (\alpha_{10}, \alpha_{11}, \dots, \alpha_{1n}) \\ \dots \\ (\alpha_{n0}, \alpha_{n1}, \dots, \alpha_{nn}) \end{pmatrix}$$

$$\begin{pmatrix} \text{softmax}(\alpha_{00}, \alpha_{01}, \dots, \alpha_{0n}) \\ \text{softmax}(\alpha_{10}, \alpha_{11}, \dots, \alpha_{1n}) \\ \dots \\ \text{softmax}(\alpha_{n0}, \alpha_{n1}, \dots, \alpha_{nn}) \end{pmatrix} \Rightarrow \alpha = \text{softmax}\left(\frac{DD^T}{\sqrt{d}}\right)$$

weight vector

$$Y = \begin{bmatrix} \alpha_{0i} & \alpha_{01} & \dots & \alpha_{0n} \\ \alpha_{1i} & \alpha_{11} & \dots & \alpha_{1n} \\ & \dots & & \\ \alpha_{ni} & \alpha_{n1} & \dots & \alpha_{nn} \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \\ \dots \\ D_n \end{bmatrix} = \begin{bmatrix} Y_0 \\ Y_1 \\ \dots \\ Y_n \end{bmatrix}$$

Contextualized representation

$$Y = \alpha D = \text{softmax}\left(\frac{DD^T}{\sqrt{d}}\right)D$$

$$X = \begin{bmatrix} X_0 \\ X_1 \\ \dots \\ X_n \end{bmatrix} \in \mathcal{R}^{n \times d}$$

$$W_Q = [\theta_{q0} \quad \theta_{q1} \quad \dots \quad \theta_{qm}] \in \mathcal{R}^{d \times m}$$

$$W_K = [\theta_{k0} \quad \theta_{k1} \quad \dots \quad \theta_{km}] \in \mathcal{R}^{d \times m}$$

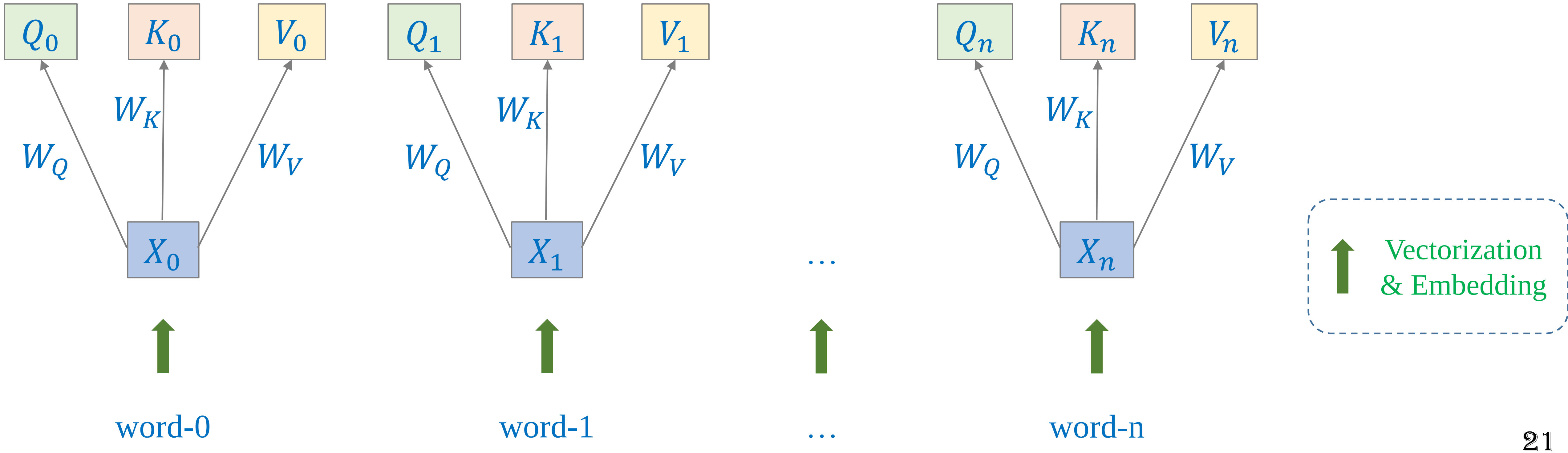
$$W_V = [\theta_{v0} \quad \theta_{v1} \quad \dots \quad \theta_{vm}] \in \mathcal{R}^{d \times m}$$

$$W_O = [\theta_0 \quad \theta_1 \quad \dots \quad \theta_o] \in \mathcal{R}^{m \times o}$$

Query $Q = XW_Q \in \mathcal{R}^{n \times m}$

Key $K = XW_K \in \mathcal{R}^{n \times m}$

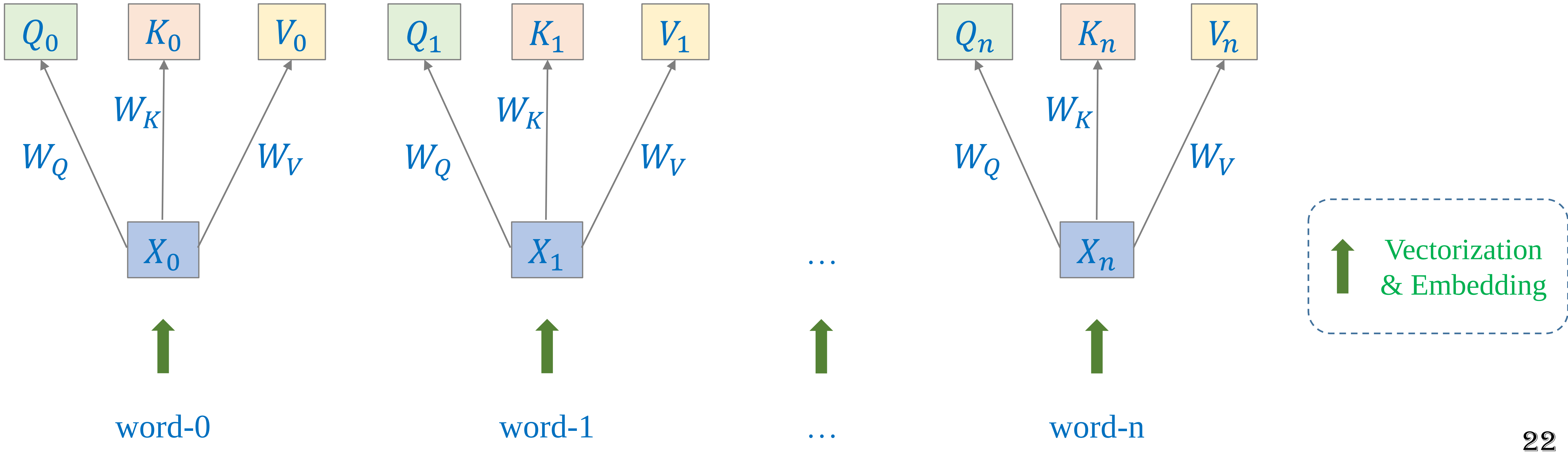
Value $V = XW_V \in \mathcal{R}^{n \times m}$



Contextualized representation

$$Y = \alpha D = \text{softmax}\left(\frac{DD^T}{\sqrt{d}}\right)D$$

$$Y = sigmoid\left(\frac{QK^T}{\sqrt{d}}\right)V$$

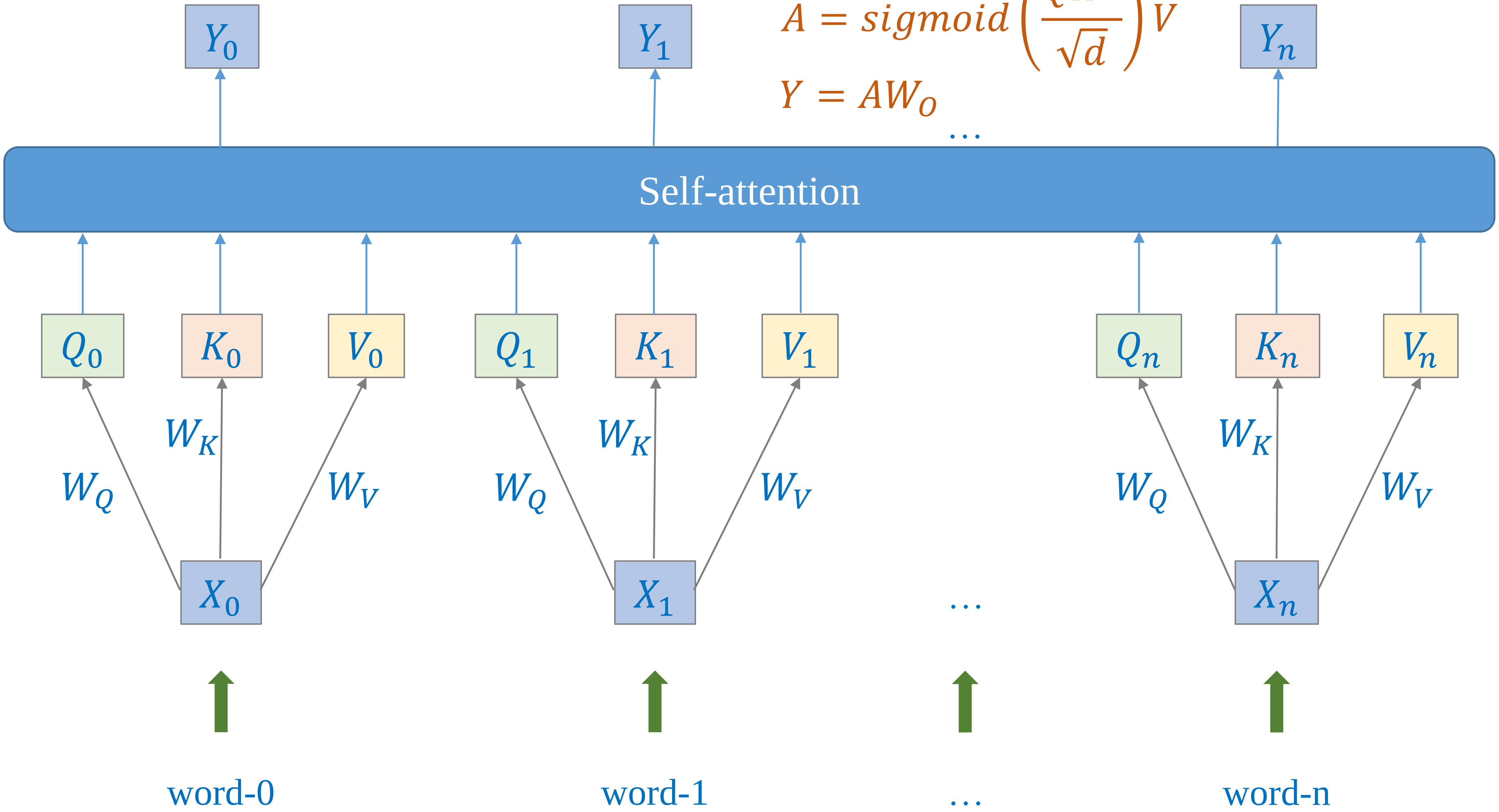


Self-Attention

```python
x = torch.tensor([[[-0.1,  0.1,  0.3], [ 0.4, -1.1, -0.3]]])
layer = nn.MultiheadAttention(embed_dim=3, num_heads=1, batch_first=True)
output_tensor, attn_output_weights = layer(query=x, key=x, value=x)
```

$$A = sigmoid\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$Y = AW_O$$



Self-Attention

head = 1

❖ **Example 1**

$$W_Q = \begin{bmatrix} -0.35 & 0.51 & 0.50 \\ 0.36 & -0.47 & -0.29 \\ -0.51 & -0.14 & -0.56 \end{bmatrix} \qquad W_V = \begin{bmatrix} -0.41 & 0.39 & -0.65 \\ -0.40 & -0.07 & -0.34 \\ -0.55 & -0.13 & -0.29 \end{bmatrix}$$

$$X = [-0.1 \;\; 0.1 \;\; 0.3] \qquad W_K = \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix} \qquad W_O = \begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix}$$

$$Q = XW_Q = [-0.1 \;\; 0.1 \;\; 0.3] \begin{bmatrix} -0.35 & 0.51 & 0.50 \\ 0.36 & -0.47 & -0.29 \\ -0.51 & -0.14 & -0.56 \end{bmatrix} = [-0.08 \;\; -0.14 \;\; -0.24]$$

$$K = XW_K = [-0.1 \;\; 0.1 \;\; 0.3] \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix} = [0.02 \;\; -0.01 \;\; 0.13]$$

$$V = XW_V = [-0.1 \;\; 0.1 \;\; 0.3] \begin{bmatrix} -0.41 & 0.39 & -0.65 \\ -0.40 & -0.07 & -0.34 \\ -0.55 & -0.13 & -0.29 \end{bmatrix} = [-0.16 \;\; -0.08 \;\; -0.05]$$

Self-Attention

❖ Example 1

approximately

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$= \text{softmax}([-0.08 \quad -0.14 \quad -0.24])\begin{bmatrix} 0.02 \\ -0.01 \\ 0.13 \end{bmatrix}\frac{1}{\sqrt{d}})[-0.16 \quad -0.08 \quad -0.05]$$

$$= \text{softmax}([-0.0198])[-0.16 \quad -0.08 \quad -0.05]$$

$$= [-0.16 \quad -0.08 \quad -0.05]$$

$$Y = AW_O = [-0.16 \quad -0.08 \quad -0.05]\begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix} = [-0.16 \quad -0.08 \quad -0.05]$$

Self-Attention

❖ **Example 2**

$$A = sigmoid\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$W_Q = \begin{bmatrix} -0.35 & 0.51 & 0.50 \\ 0.36 & -0.47 & -0.29 \\ -0.51 & -0.14 & -0.56 \end{bmatrix}$$

$$W_K = \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix}$$

$$W_V = \begin{bmatrix} -0.41 & 0.39 & -0.65 \\ -0.40 & -0.07 & -0.34 \\ -0.55 & -0.13 & -0.29 \end{bmatrix}$$

$$W_O = \begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix}$$

$$X = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix}$$

$$Q = XW_Q = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix} \begin{bmatrix} -0.35 & 0.51 & 0.50 \\ 0.36 & -0.47 & -0.29 \\ -0.51 & -0.14 & -0.56 \end{bmatrix}$$

$$= \begin{bmatrix} -0.08 & -0.14 & -0.24 \\ -0.39 & 0.77 & 0.69 \end{bmatrix}$$

$$K = XW_K = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix} \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix}$$

$$= \begin{bmatrix} 0.02 & -0.01 & 0.13 \\ 0.27 & 0.27 & -0.26 \end{bmatrix}$$

$$V = XW_V = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix} \begin{bmatrix} -0.41 & 0.39 & -0.65 \\ -0.40 & -0.07 & -0.34 \\ -0.55 & -0.13 & -0.29 \end{bmatrix}$$

$$= \begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix}$$

Self-Attention

❖ Example 2

approximately

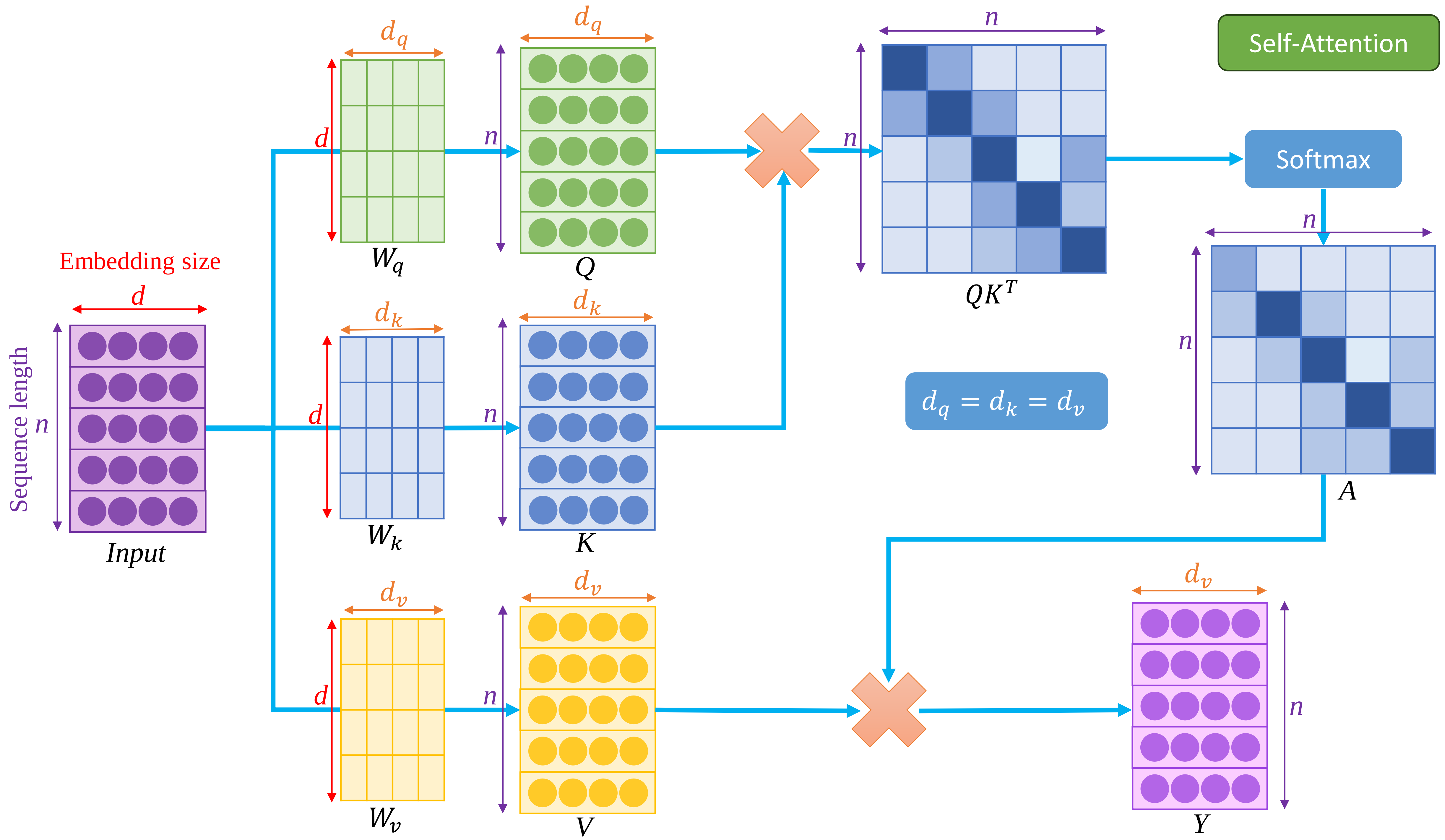
$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$= \text{softmax}\left(\begin{bmatrix} -0.08 & -0.14 & -0.24 \\ -0.39 & 0.77 & 0.69 \end{bmatrix} \begin{bmatrix} 0.02 & 0.27 \\ -0.01 & 0.27 \\ 0.13 & -0.26 \end{bmatrix} \frac{1}{\sqrt{d}}\right) \begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix}$$

$$= \text{softmax}\left(\begin{bmatrix} -0.019 & 0.002 \\ 0.043 & -0.046 \end{bmatrix}\right) \begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix}$$

$$= \begin{bmatrix} 0.49 & 0.51 \\ 0.52 & 0.48 \end{bmatrix} \begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix} = \begin{bmatrix} 0.14 & 0.09 & 0.07 \\ 0.12 & 0.08 & 0.06 \end{bmatrix}$$

$$Y = AW_O = \begin{bmatrix} 0.14 & 0.09 & 0.07 \\ 0.12 & 0.08 & 0.06 \end{bmatrix} \begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix} = \begin{bmatrix} -0.029 & -0.028 & 0.065 \\ -0.025 & -0.025 & 0.058 \end{bmatrix}$$

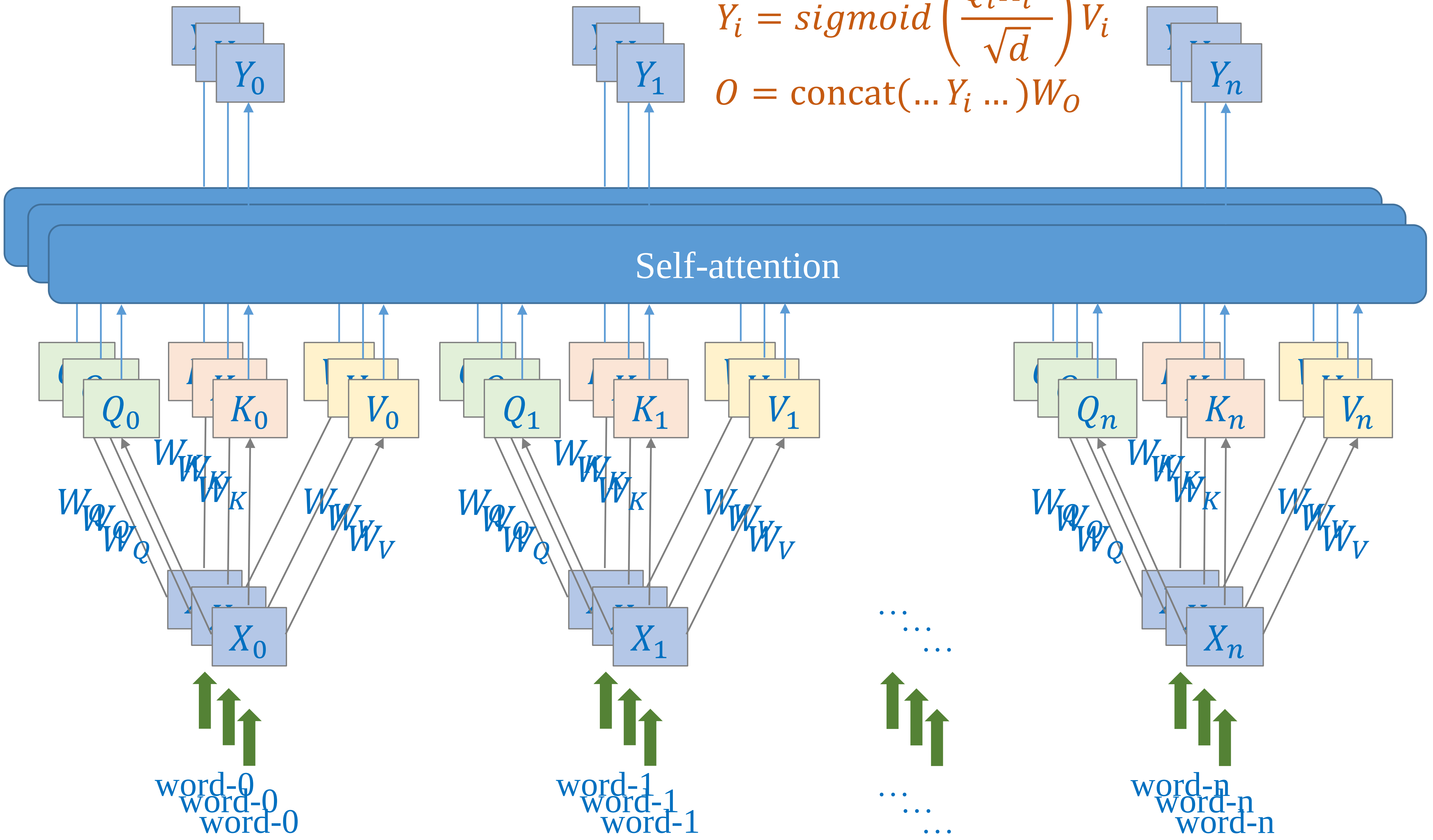


Self-Attention
Embedding size
d
Sequence length n
Input
d_q
d
W_q
n
Q
d_k
d
W_k
n
K
d_v
d
W_v
n
V
n
n
QK^T
Softmax
n
n
A
$d_q = d_k = d_v$
d_v
n
Y

$W_O = [\theta_0 \quad \theta_1 \quad \dots \quad \theta_o] \in \mathcal{R}^{h*m\times o}$

$$Y_i = sigmoid\left(\frac{Q_i {K_i}^T}{\sqrt{d}}\right) V_i$$

$$O = \mathrm{concat}(\dots Y_i \dots) W_O$$

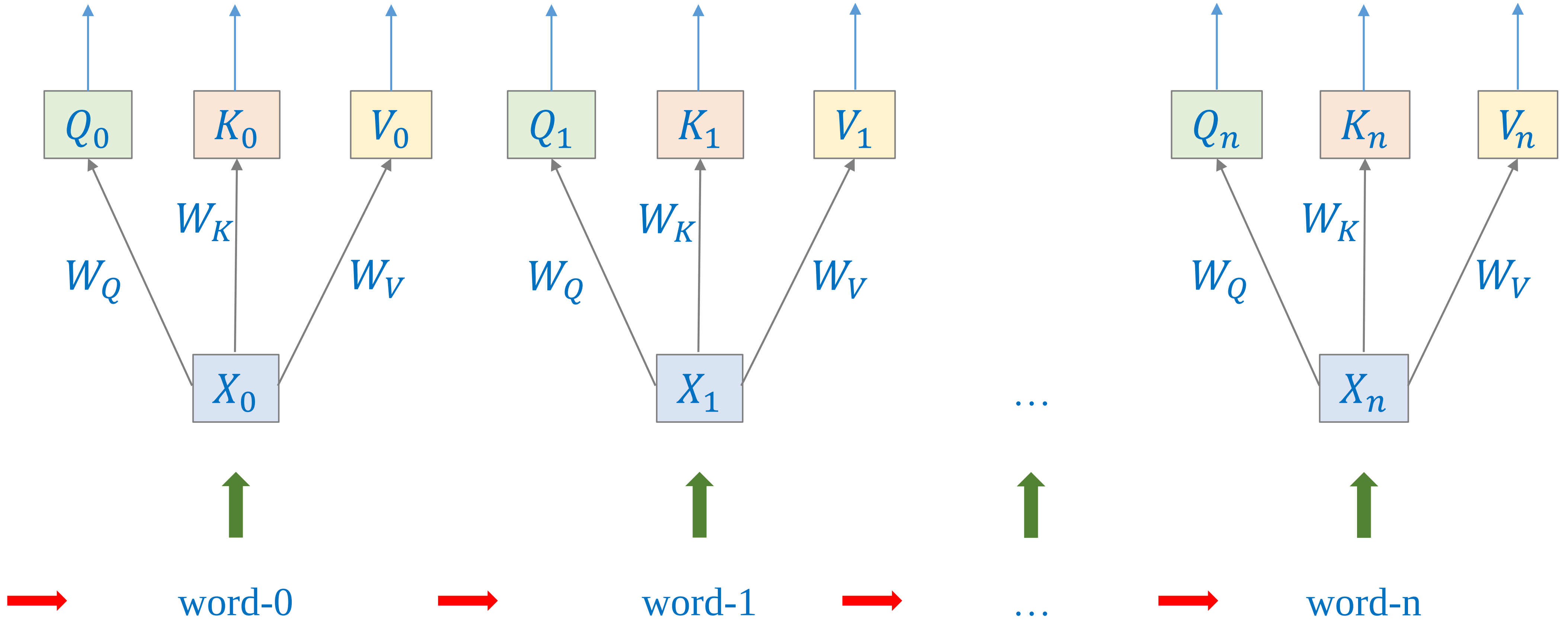


Outline

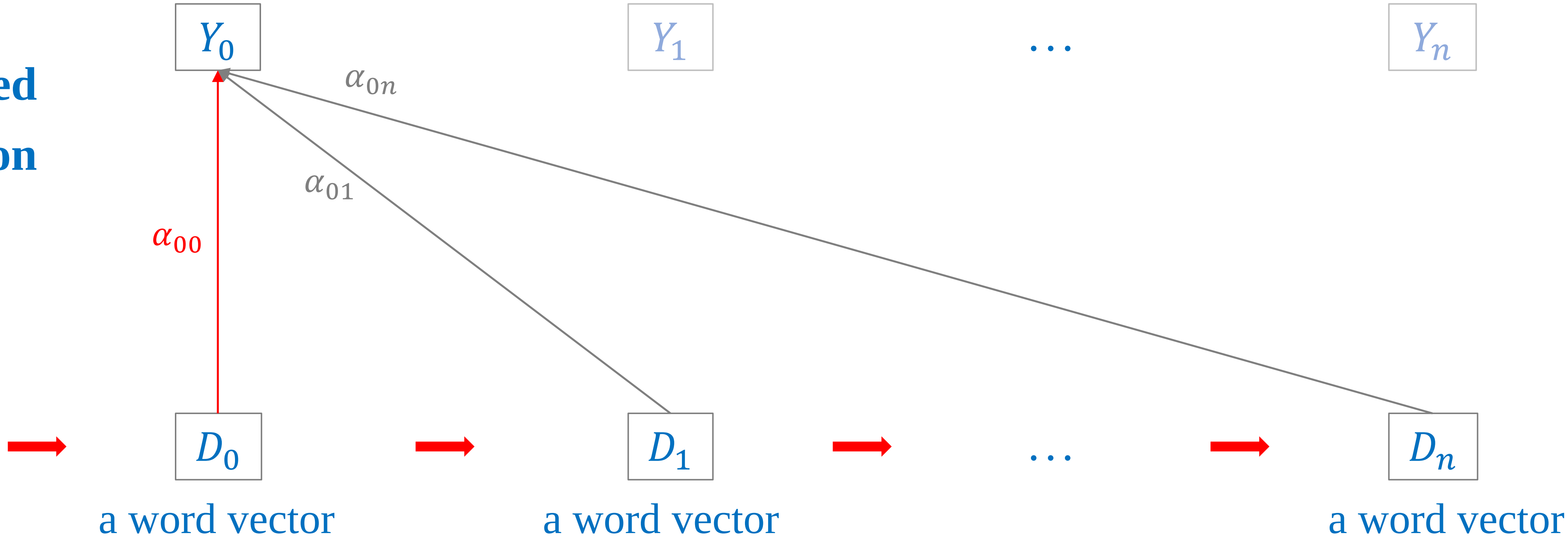
- Revisiting RNNs, MLPs, and CNNs
- From RNNs to Transformers
- Self-Attention
- Masked Self-Attention
- Cross-Attention

Transformer

❖ Masked self-attention



Masked self-attention



$$Y_0 = \alpha_{00} D_0 + \alpha_{01} D_1 + \cdots + \alpha_{0n} D_n$$

⬇

$$Y_0 = \alpha_{00} D_0 + 0 \times D_1 + \cdots + 0 \times D_n$$

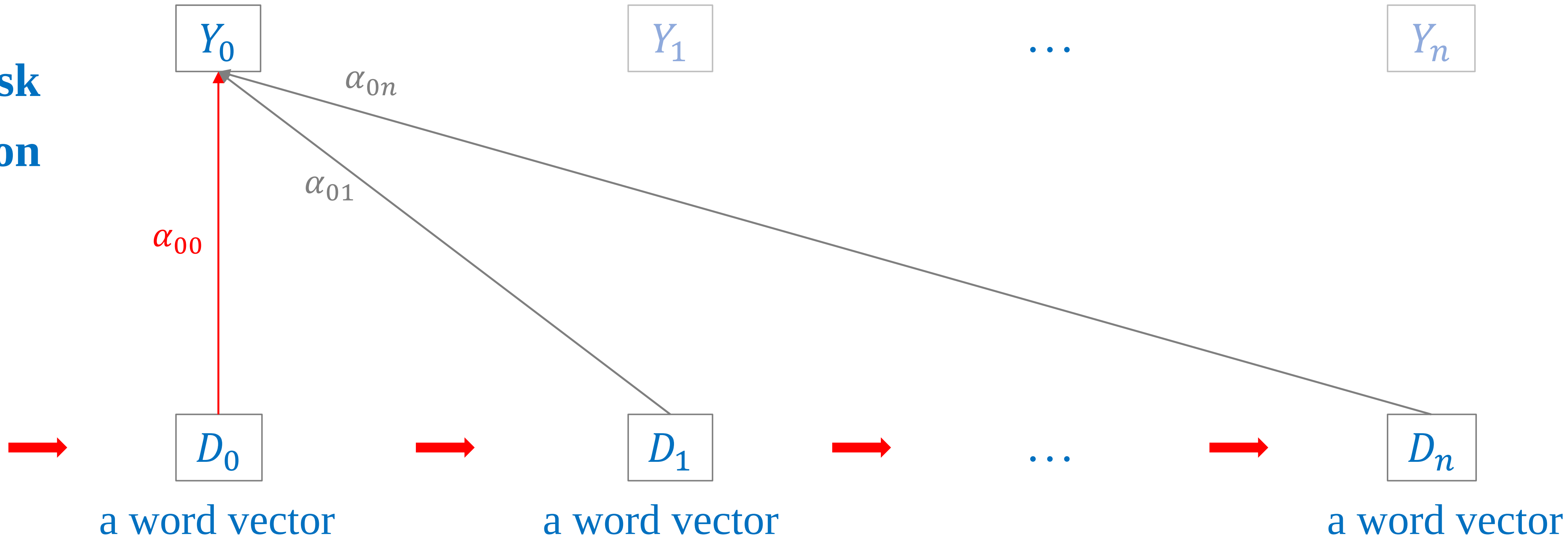
$$\alpha_0 = \text{softmax}\left(\frac{D_0 D^T}{\sqrt{d}}\right) = \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \dots \\ \alpha_{0n} \end{bmatrix}$$

How to obtain kind of → $\begin{bmatrix} \alpha_{00} \\ 0 \\ \dots \\ 0 \end{bmatrix}$

$$\alpha_0 = \text{softmax}\left(\frac{D_0 D^T}{\sqrt{d}}\right) * \begin{bmatrix} 1 \\ 0 \\ \dots \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha_{00} \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

?

Mask self-attention



$$\alpha_0 = \text{softmax}\left(\frac{D_0 D^T}{\sqrt{d}}\right) = \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \dots \\ \alpha_{0n} \end{bmatrix}$$

How to obtain kind of → $\begin{bmatrix} \alpha_{00} \\ 0 \\ \dots \\ 0 \end{bmatrix}$

$$Y_0 = \alpha_{00} D_0 + \alpha_{01} D_1 + \cdots + \alpha_{0n} D_n$$

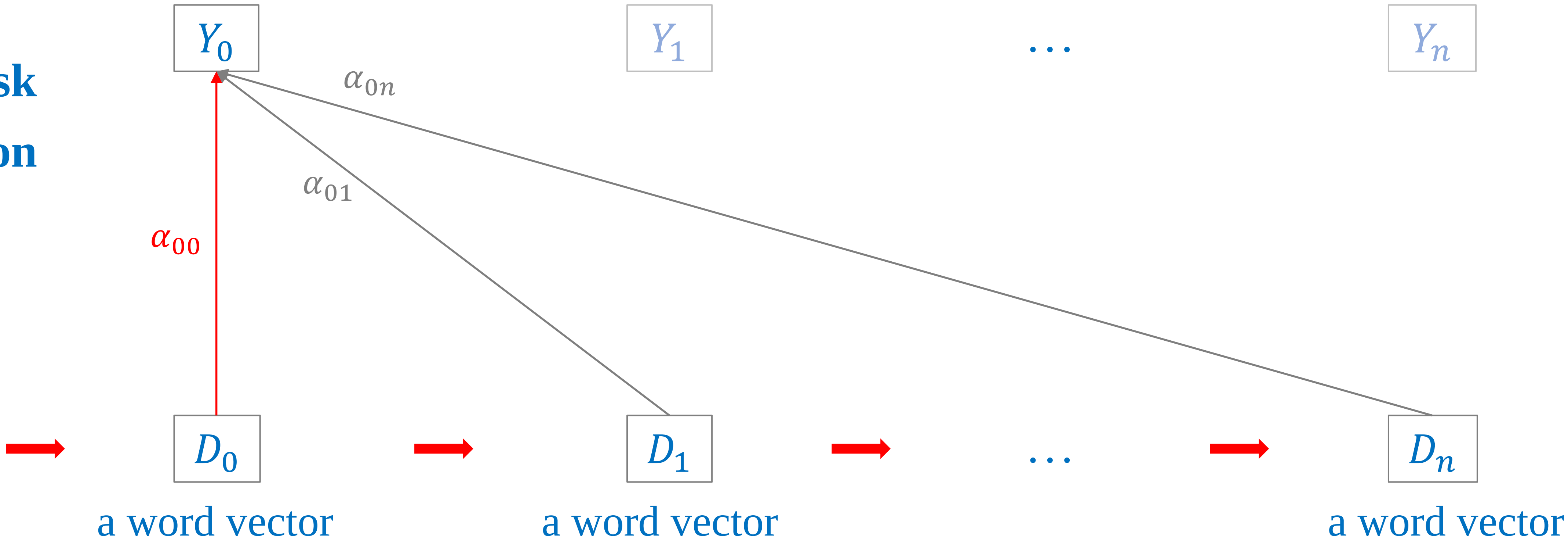
⬇

$$Y_0 = \alpha_{00} D_0 + 0 \times D_1 + \cdots + 0 \times D_n$$

$$\alpha_0 = \text{softmax}\left(\frac{D_0 D^T}{\sqrt{d}} * \begin{bmatrix} 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}\right) = \begin{bmatrix} \alpha_{00} \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

?

Mask self-attention



$$\alpha_0 = \mathrm{softmax}\left(\frac{D_0 D^T}{\sqrt{m}}\right) = \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \dots \\ \alpha_{0n} \end{bmatrix}$$

How to obtain kind of → $\begin{bmatrix} \alpha_{00} \\ 0 \\ \dots \\ 0 \end{bmatrix}$

$$Y_0 = \alpha_{00} D_0 + \alpha_{01} D_1 + \cdots + \alpha_{0n} D_n$$

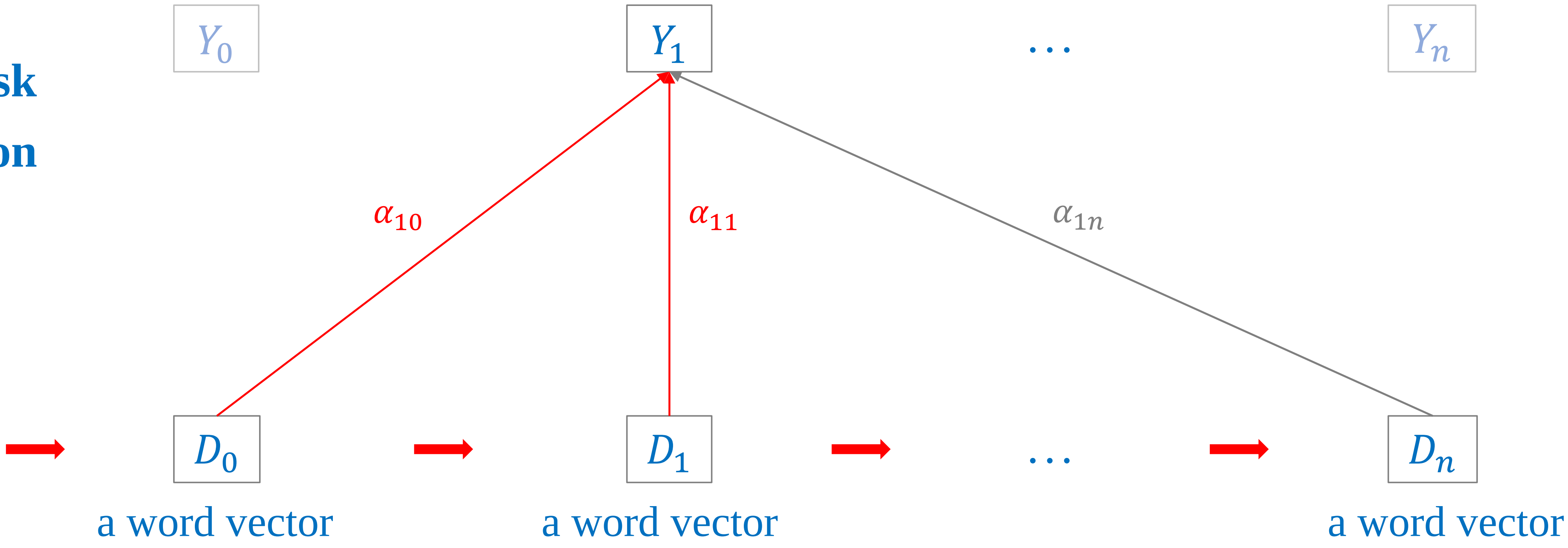
⬇

$$Y_0 = \alpha_{00} D_0 + 0 \times D_1 + \cdots + 0 \times D_n$$

$$\alpha_0 = \mathrm{softmax}\left(\frac{D_0 D^T}{\sqrt{d}} + \begin{bmatrix} 0 \\ -\infty \\ \dots \\ -\infty \end{bmatrix}\right) = \begin{bmatrix} \alpha_{00} \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

?

Mask self-attention



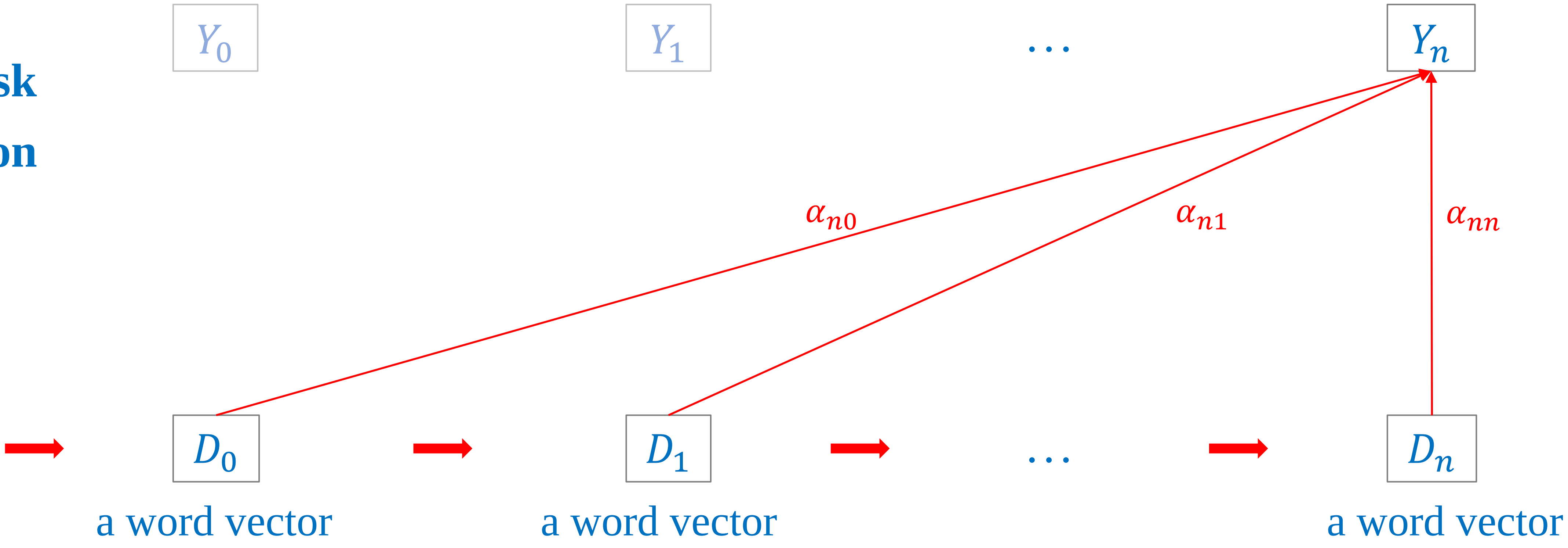
$$Y_1 = \alpha_{10} D_0 + \alpha_{11} D_1 + \cdots + \alpha_{1n} D_n$$

$$\Downarrow$$

$$Y_1 = \alpha_{10} D_0 + \alpha_{11} D_1 + \cdots + 0 \times D_n$$

$$\alpha_1 = \mathrm{softmax}\left(\frac{D_1 D^T}{\sqrt{d}} + \begin{bmatrix} 0 \\ 0 \\ -\infty \\ \dots \\ -\infty \end{bmatrix} \right) = \begin{bmatrix} \alpha_{10} \\ \alpha_{11} \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

Mask self-attention

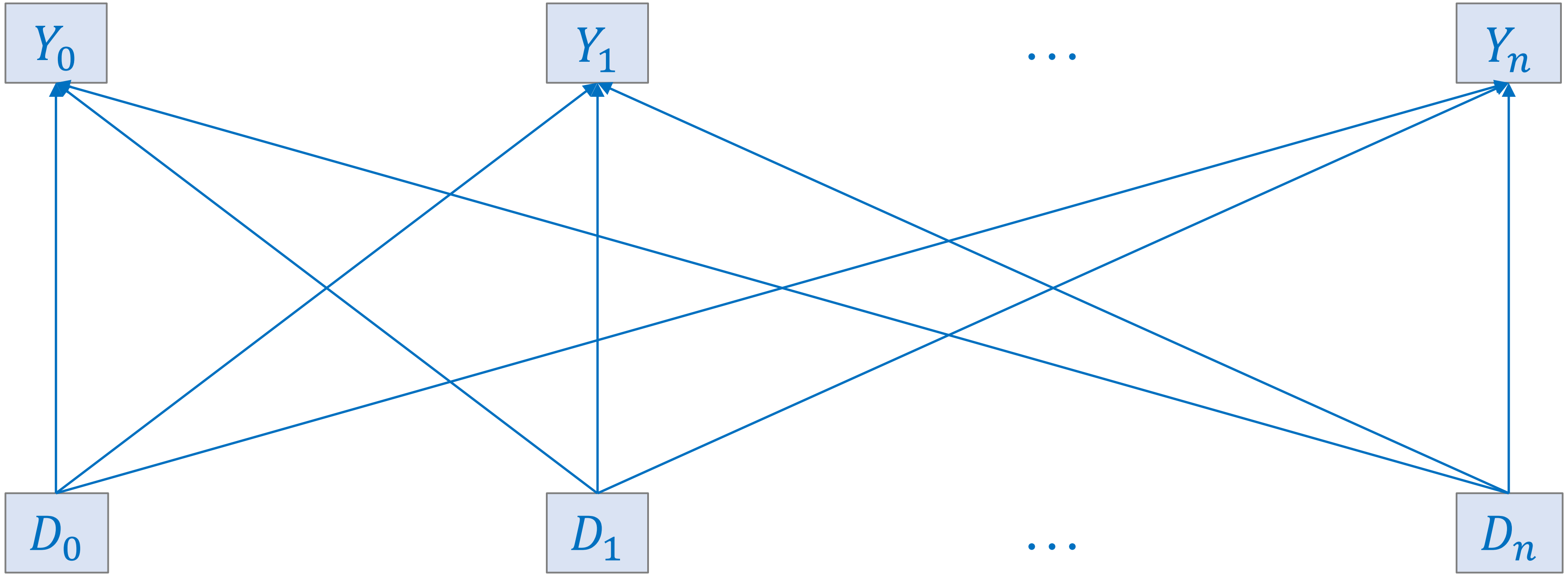


$$Y_n = \alpha_{n0} D_0 + \alpha_{n1} D_1 + \cdots + \alpha_{nn} D_n$$

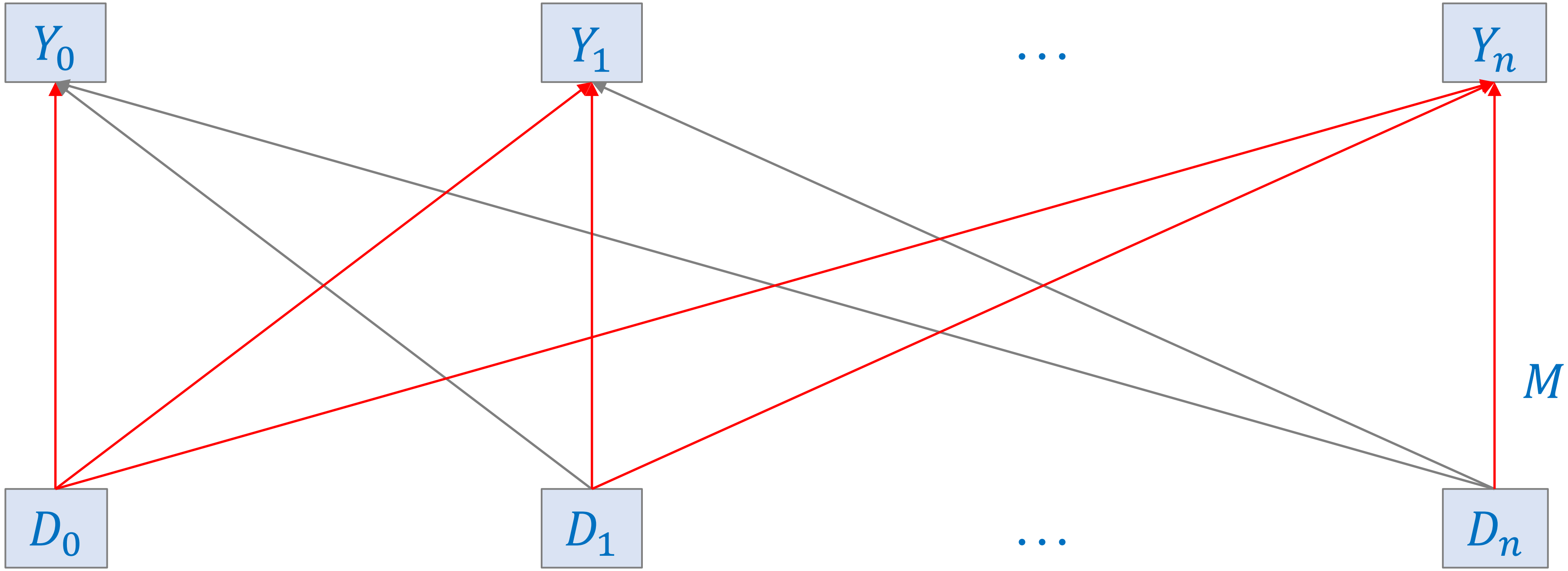
⬇

?

$$\alpha_n = \mathrm{softmax}\left(\frac{D_n D^T}{\sqrt{d}} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} \right) = \begin{bmatrix} \alpha_{n0} \\ \alpha_{n1} \\ \alpha_{n2} \\ \dots \\ \alpha_{nn} \end{bmatrix}$$



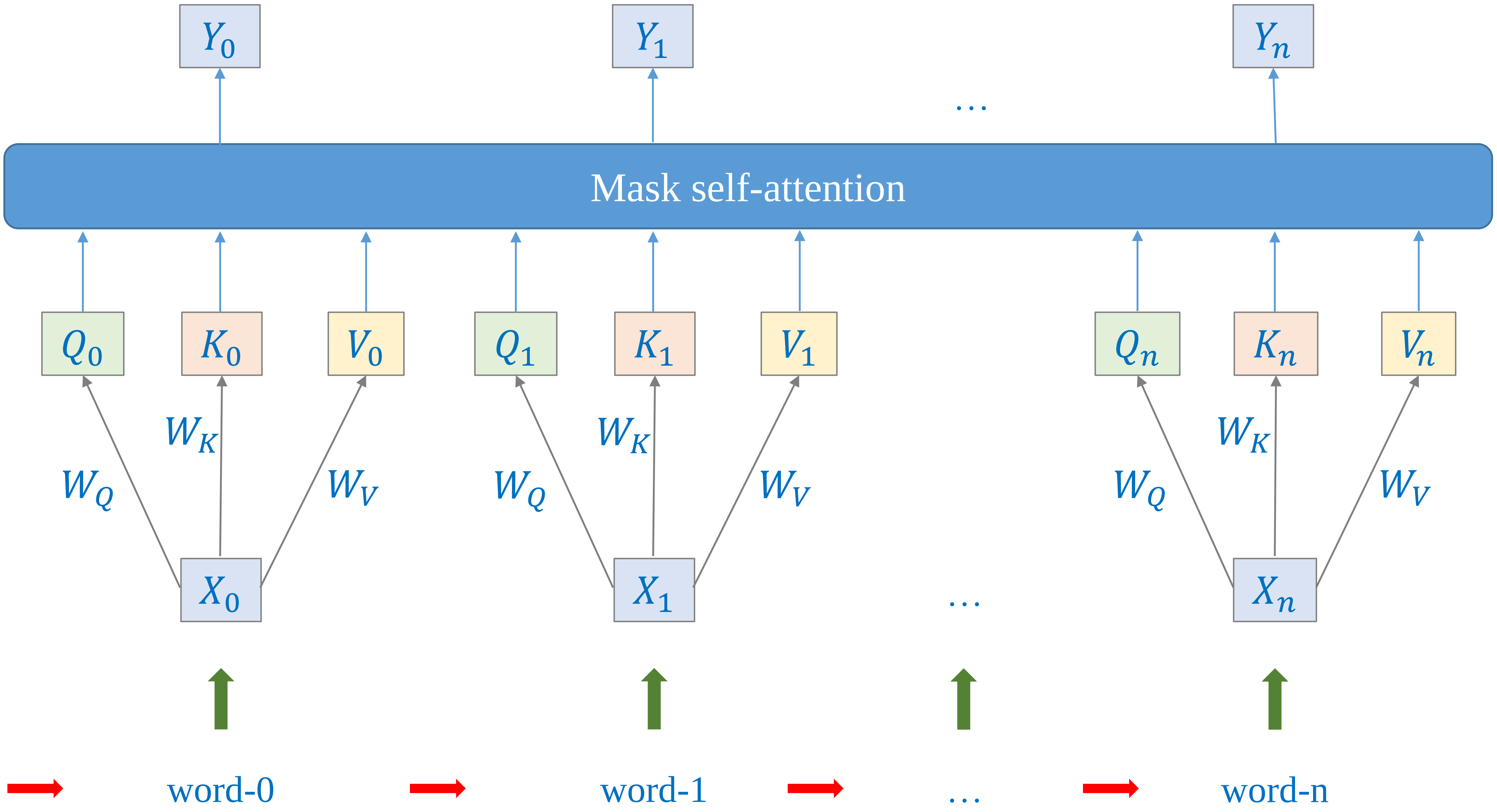
$$\alpha = sigmoid\left(\frac{DD^T}{\sqrt{d}}\right)$$



$$\alpha = sigmoid\left(\frac{DD^T}{\sqrt{d}} + M\right)$$

$$M = \begin{bmatrix} [0 & -\infty & -\infty \dots & -\infty] \\ [0 & 0 & -\infty \dots & -\infty] \\ & & \dots & \\ [0 & 0 & 0 \quad \dots & 0] \end{bmatrix}$$

$$Y = sigmoid\left(\frac{QK^T}{\sqrt{d}} + M\right)V \qquad M = \begin{bmatrix} [0 & -\infty & -\infty \ldots & -\infty] \\ [0 & 0 & -\infty \ldots & -\infty] \\ & & \ldots & \\ [0 & 0 & 0 \quad \ldots & 0] \end{bmatrix}$$



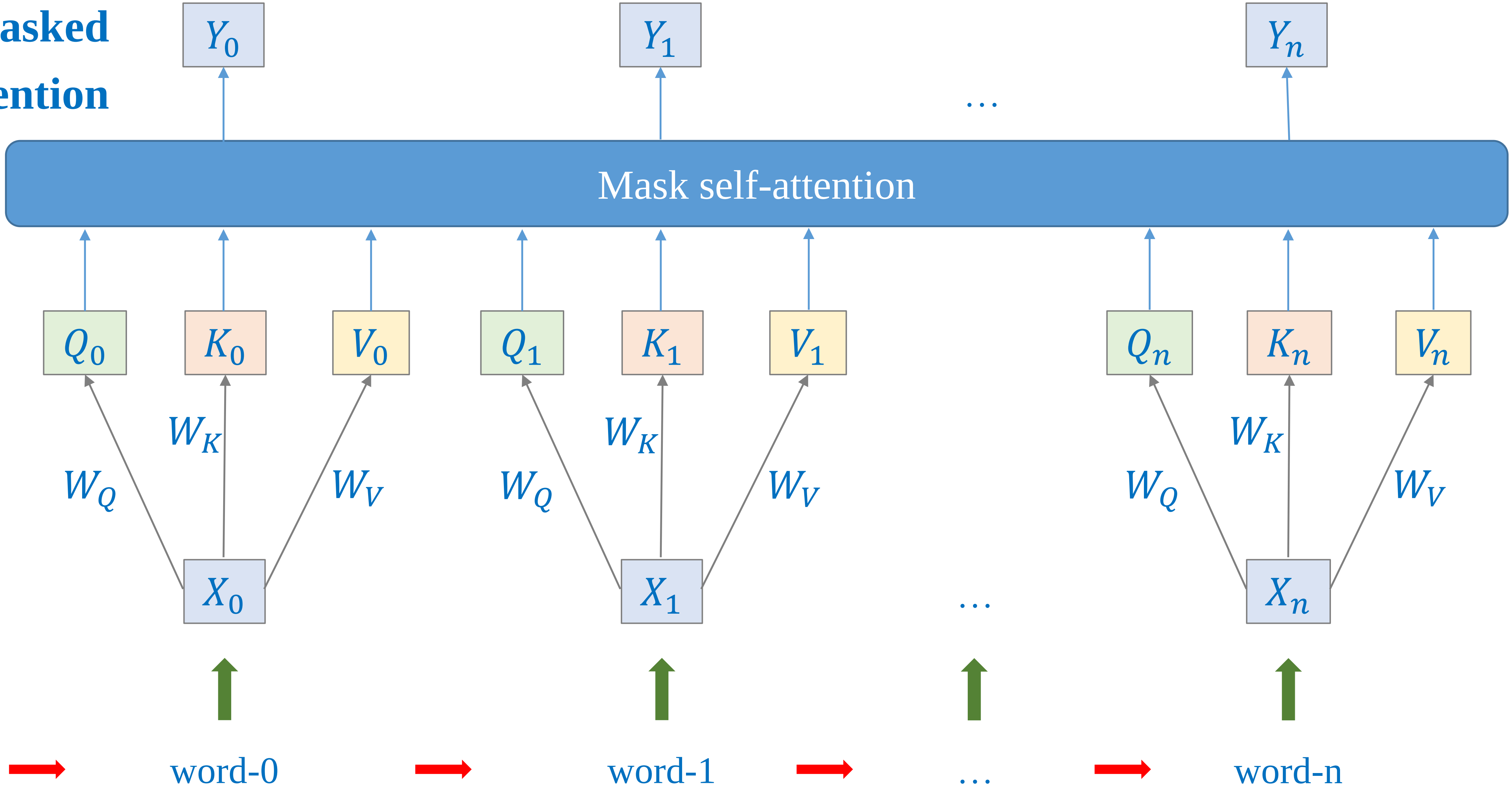
```python
x = torch.tensor([[[-0.1,  0.1,  0.3], [ 0.4, -1.1, -0.3]]])
layer = nn.MultiheadAttention(embed_dim=3, num_heads=1, batch_first=True)

mask = torch.triu(torch.ones(1, 2, 2), diagonal=1).bool()
output_tensor, attn_output_weights = layer(query=x, key=x, value=x, attn_mask=mask)
```

$$A = sigmoid\left(\frac{QK^T}{\sqrt{d}} + M\right)V$$

$$Y = AW_O$$

Masked Self-Attention



Masked Multi-head Attention

head = 1

$$W_Q = \begin{bmatrix} -0.35 & 0.51 & 0.50 \\ 0.36 & -0.47 & -0.29 \\ -0.51 & -0.14 & -0.56 \end{bmatrix}$$

$$W_K = \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix}$$

$$W_V = \begin{bmatrix} -0.41 & 0.39 & -0.65 \\ -0.40 & -0.07 & -0.34 \\ -0.55 & -0.13 & -0.29 \end{bmatrix}$$

$$W_O = \begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix}$$

$$X = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix}$$

$$Q = XW_Q = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix} \begin{bmatrix} -0.35 & 0.51 & 0.50 \\ 0.36 & -0.47 & -0.29 \\ -0.51 & -0.14 & -0.56 \end{bmatrix}$$

$$= \begin{bmatrix} -0.08 & -0.14 & -0.24 \\ -0.39 & 0.77 & 0.69 \end{bmatrix}$$

$$K = XW_K = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix} \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix}$$

$$= \begin{bmatrix} 0.02 & -0.01 & 0.13 \\ 0.27 & 0.27 & -0.26 \end{bmatrix}$$

$$V = XW_V = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix} \begin{bmatrix} -0.41 & 0.39 & -0.65 \\ -0.40 & -0.07 & -0.34 \\ -0.55 & -0.13 & -0.29 \end{bmatrix}$$

$$= \begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix}$$

$$M = \begin{bmatrix} 0 & -\infty \\ 0 & 0 \end{bmatrix}$$

Masked Multi-head Attention

❖ Example

approximately

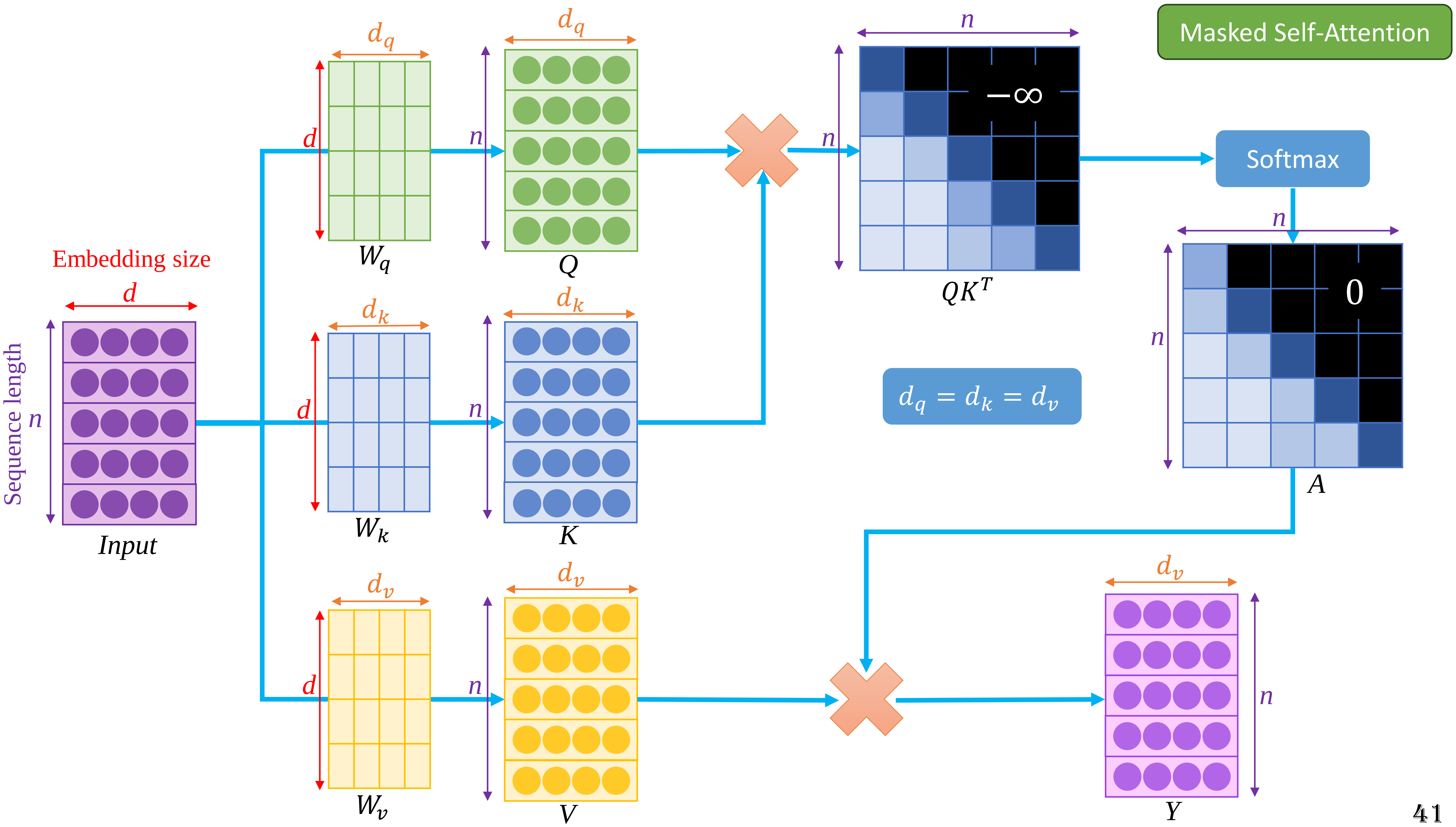
$$A = sigmoid\left(\frac{QK^T}{\sqrt{d}} + M\right)V$$

$$= sigmoid\left(\begin{bmatrix} -0.08 & -0.14 & -0.24 \\ -0.39 & 0.77 & 0.69 \end{bmatrix}\begin{bmatrix} 0.02 & 0.27 \\ -0.01 & 0.27 \\ 0.13 & -0.26 \end{bmatrix}\frac{1}{\sqrt{d}} + \begin{bmatrix} 0 & -\infty \\ 0 & 0 \end{bmatrix}\right)\begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix}$$

$$= sigmoid\left(\begin{bmatrix} -0.019 & 0.002 \\ 0.043 & -0.046 \end{bmatrix} + \begin{bmatrix} 0 & -\infty \\ 0 & 0 \end{bmatrix}\right)\begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix}$$

$$= \begin{bmatrix} 1.0 & 0.0 \\ 0.52 & 0.48 \end{bmatrix}\begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix} = \begin{bmatrix} -0.16 & -0.08 & -0.05 \\ 0.12 & 0.08 & 0.06 \end{bmatrix}$$

$$Y = AW_O = \begin{bmatrix} -0.16 & -0.08 & -0.05 \\ 0.12 & 0.08 & 0.06 \end{bmatrix}\begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix} = \begin{bmatrix} 0.03 & 0.02 & -0.06 \\ -0.02 & -0.02 & 0.05 \end{bmatrix}$$


Masked Self-Attention
Embedding size
d
Sequence length n
Input
d_q
d
W_q
n
Q
d_k
W_k
K
d_v
W_v
V
QK^T
$-\infty$
Softmax
0
A
$d_q = d_k = d_v$
Y

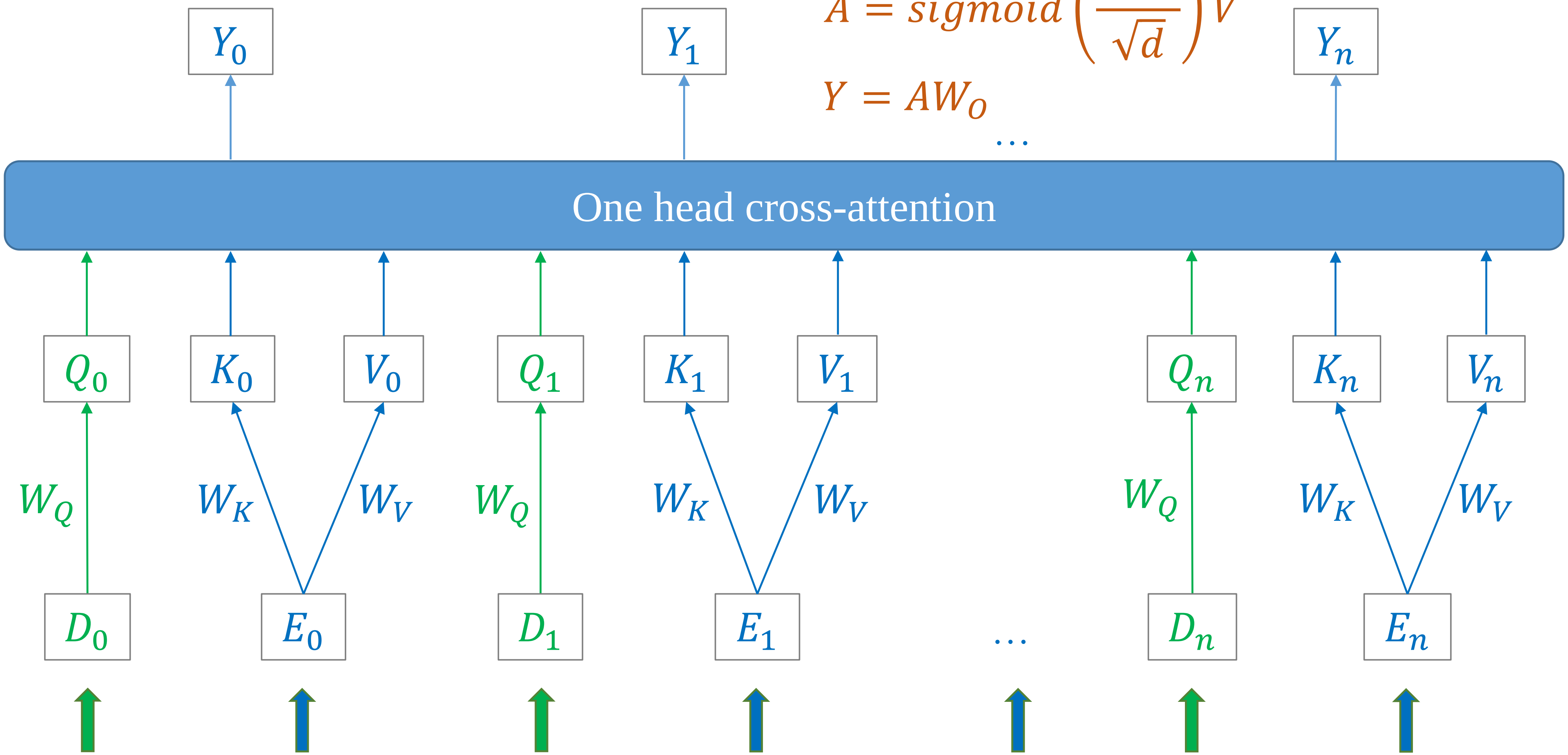
Cross-Attention

```python
x = torch.tensor([[[-0.1,  0.1,  0.3], [ 0.4, -1.1, -0.3]]])
c = torch.tensor([[[-0.6, 0.3, -0.4], [ 0.5, 0.9, -0.5]]])
layer = nn.MultiheadAttention(embed_dim=3, num_heads=1, batch_first=True)
output_tensor, attn_output_weights = layer(query=c, key=x, value=x)
```

❖ **Example**

$$A = sigmoid\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$Y = AW_O$$



head = 1

Cross-Attention

❖ Example

$$W_Q = \begin{bmatrix} -0.35 & 0.51 & 0.50 \\ 0.36 & -0.47 & -0.29 \\ -0.51 & -0.14 & -0.56 \end{bmatrix}$$

$$W_K = \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix}$$

$$W_V = \begin{bmatrix} -0.41 & 0.39 & -0.65 \\ -0.40 & -0.07 & -0.34 \\ -0.55 & -0.13 & -0.29 \end{bmatrix}$$

$$W_O = \begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix}$$

$$X = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix}$$

$$C = \begin{bmatrix} -0.6 & 0.3 & -0.4 \\ 0.5 & 0.9 & -0.5 \end{bmatrix}$$

$$Q = CW_Q = \begin{bmatrix} -0.6 & 0.3 & -0.4 \\ 0.5 & 0.9 & -0.5 \end{bmatrix} \begin{bmatrix} -0.35 & 0.51 & 0.50 \\ 0.36 & -0.47 & -0.29 \\ -0.51 & -0.14 & -0.56 \end{bmatrix}$$

$$= \begin{bmatrix} 0.52 & -0.39 & -0.16 \\ 0.40 & -0.09 & 0.27 \end{bmatrix}$$

$$K = XW_K = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix} \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix}$$

$$= \begin{bmatrix} 0.02 & -0.01 & 0.13 \\ 0.27 & 0.27 & -0.26 \end{bmatrix}$$

$$V = XW_V = \begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix} \begin{bmatrix} -0.41 & 0.39 & -0.65 \\ -0.40 & -0.07 & -0.34 \\ -0.55 & -0.13 & -0.29 \end{bmatrix}$$

$$= \begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix}$$

Cross-Attention

❖ Example

approximately

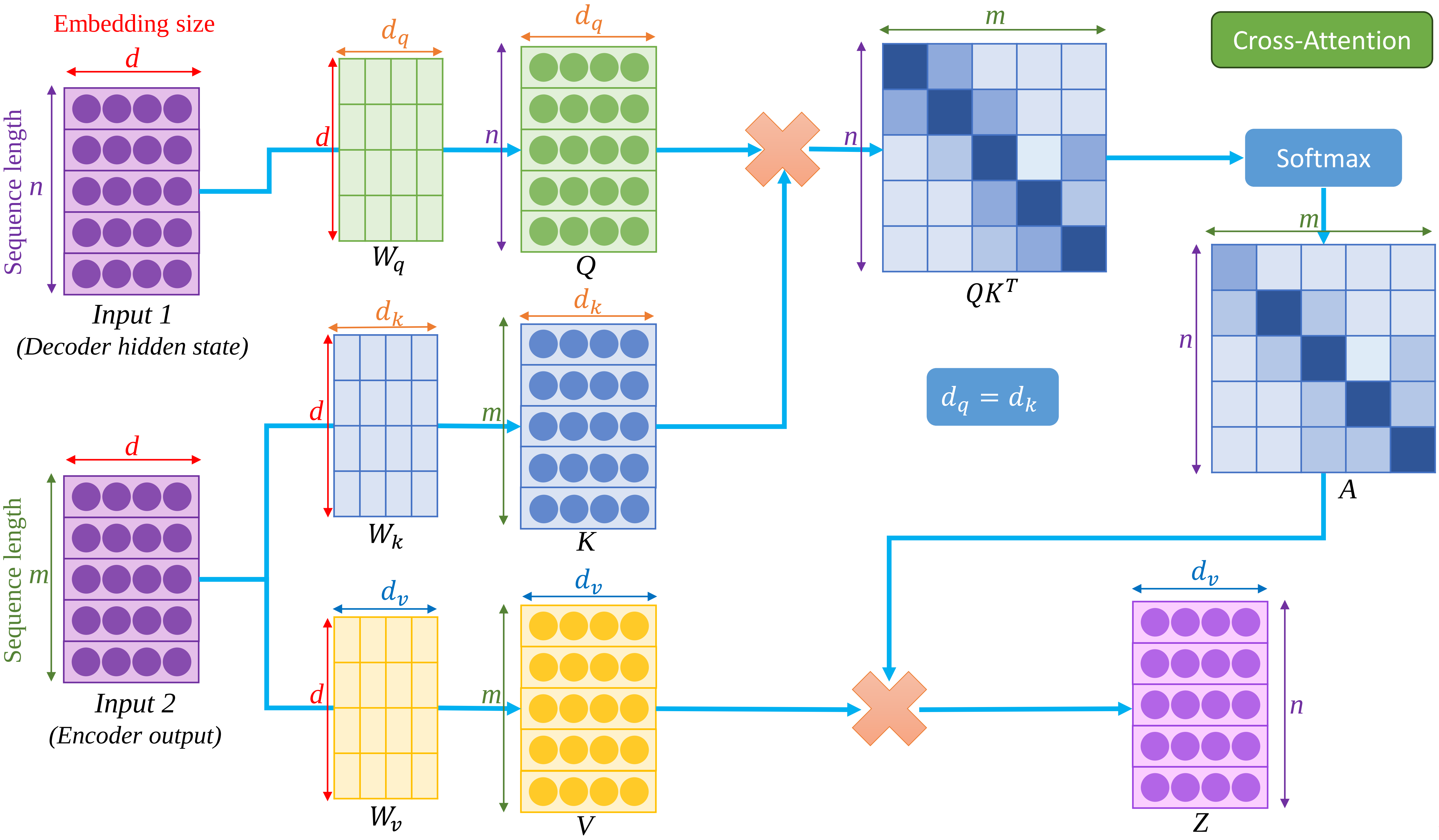
$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$= \text{softmax}\left(\begin{bmatrix} 0.52 & -0.39 & -0.16 \\ 0.40 & -0.09 & 0.27 \end{bmatrix}\begin{bmatrix} 0.02 & 0.27 \\ -0.01 & 0.27 \\ 0.13 & -0.26 \end{bmatrix}\frac{1}{\sqrt{d}}\right)\begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix}$$

$$= \text{softmax}\left(\begin{bmatrix} -0.0028 & 0.0470 \\ 0.0278 & 0.0075 \end{bmatrix}\right)\begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix}$$

$$= \begin{bmatrix} 0.49 & 0.51 \\ 0.51 & 0.49 \end{bmatrix}\begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix} = \begin{bmatrix} 0.15 & 0.10 & 0.07 \\ 0.13 & 0.09 & 0.07 \end{bmatrix}$$

$$Y = AW_O = \begin{bmatrix} 0.15 & 0.10 & 0.07 \\ 0.13 & 0.09 & 0.07 \end{bmatrix}\begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix} = \begin{bmatrix} -0.0305 & -0.0296 & 0.0677 \\ -0.0281 & -0.0277 & 0.0630 \end{bmatrix}$$


Cross-Attention
Embedding size
d
Sequence length n
Input 1
(Decoder hidden state)
d_q
d
W_q
n
Q
m
QK^T
Softmax
A
d_k
W_k
m
K
$d_q = d_k$
Sequence length m
Input 2
(Encoder output)
d_v
W_v
V
Z