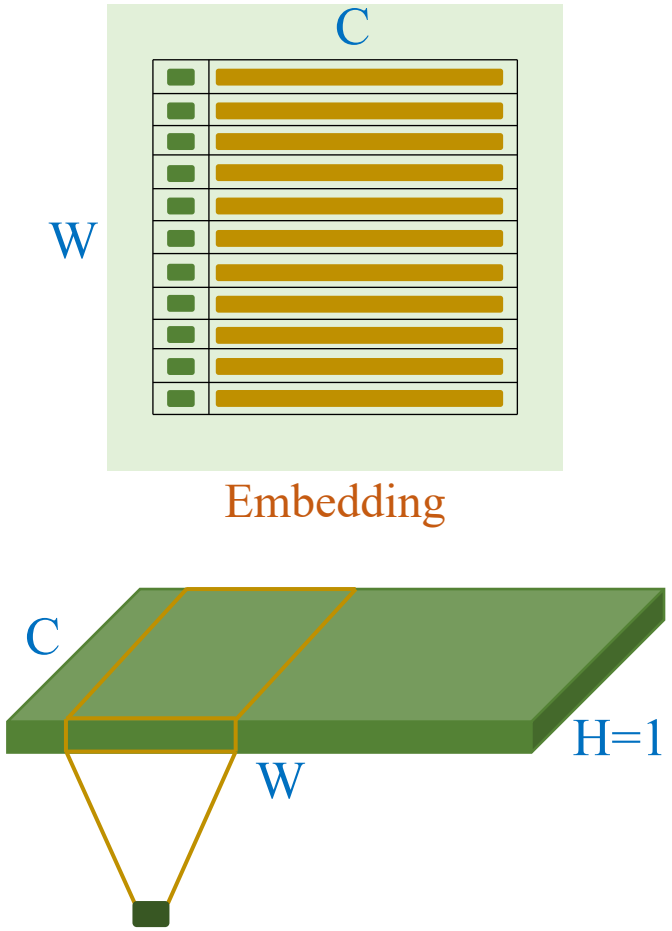
AI VIET NAM
@aivietnam.edu.vn

Transformer for Text Classification

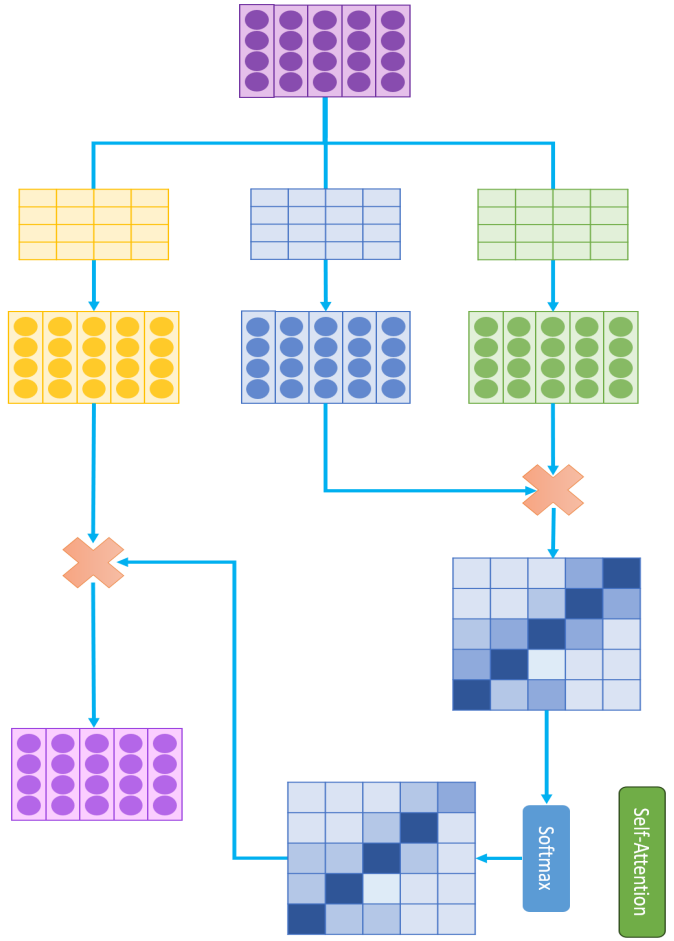
Quang-Vinh Dinh
Ph.D. in Computer Science

Objectives

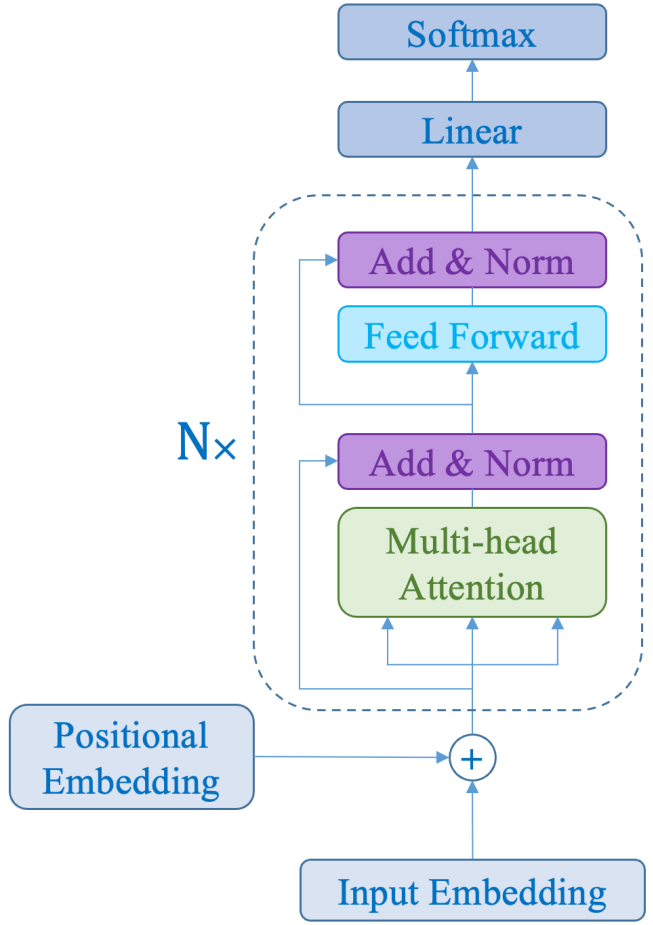
Text Classification



RNN → Transformer



Application



Outline

SECTION 1

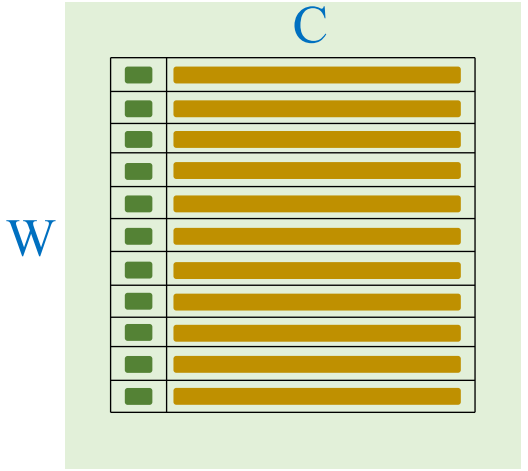
Text Classification

SECTION 2

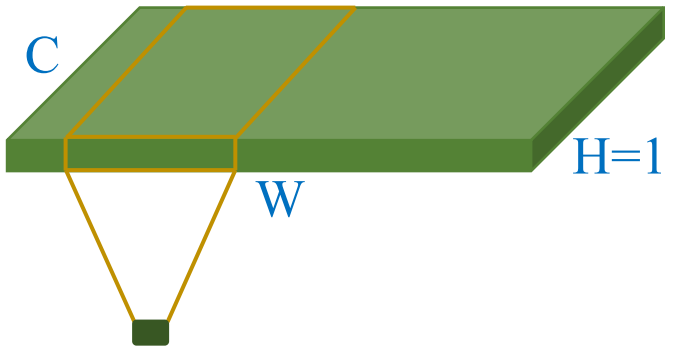
RNN → Transformer

SECTION 3

Application



Embedding



Text Classification

❖ Text classification model

Data

The first von Trier movie i've ever seen …

Convert words to indices

Text Vectorization

Map indices to vectors

Text Embedding

Understand context

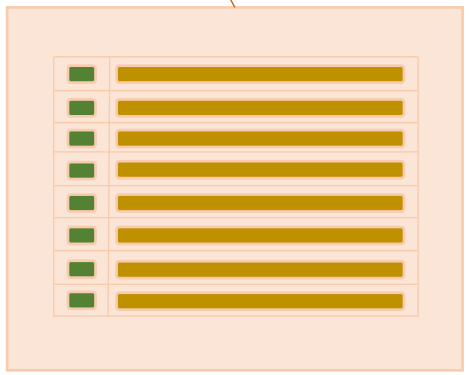
Text Encoding

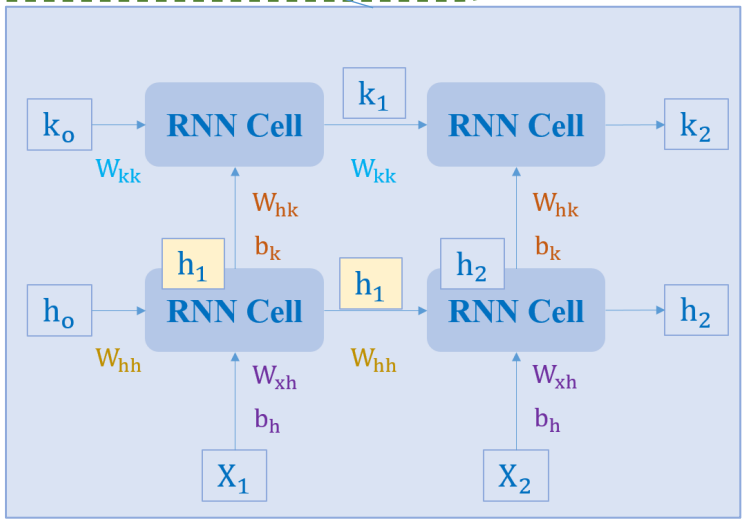
Classify

Linear & Softmax

Output

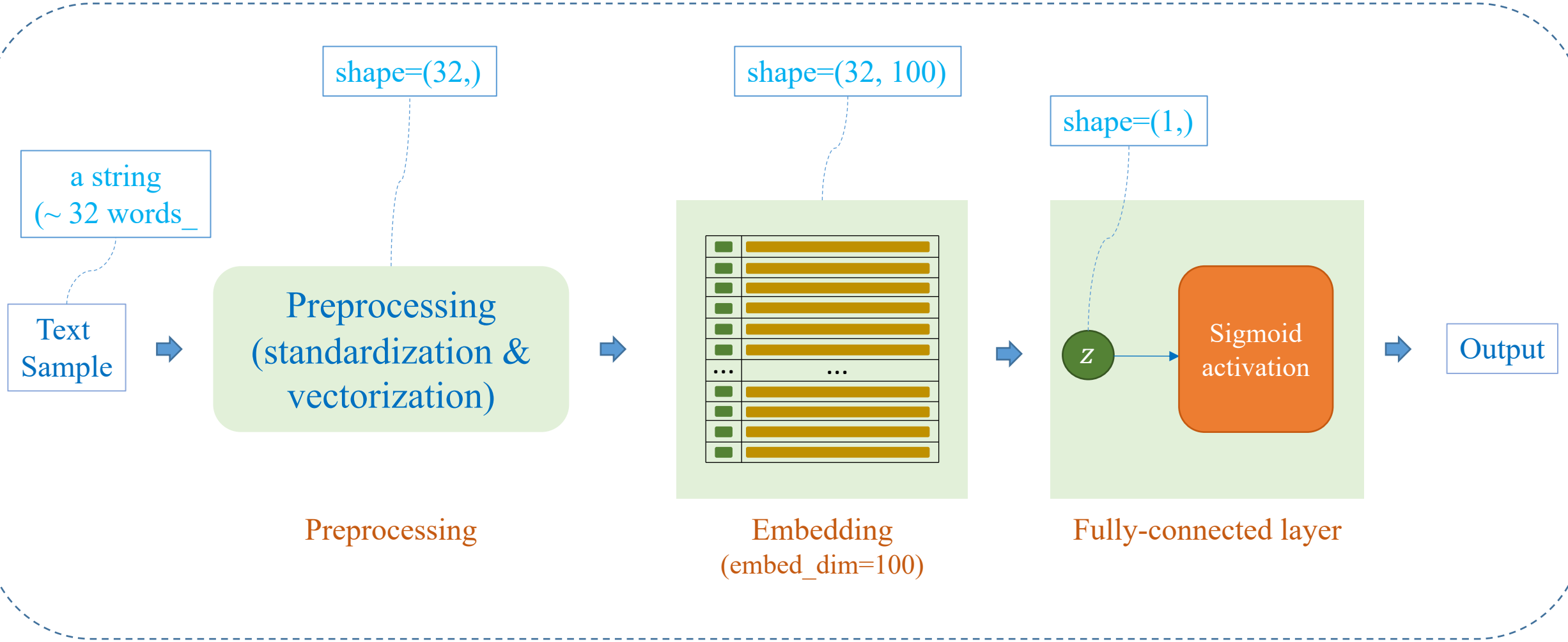
We are learning AI				AI is a CS topic					
				Standardize					
we	are	learning	ai	ai	is	a	cs	topic	
				Vectorization					
3	1	5	2	0	2	6	1	7	4





Text Classification

❖ Simple approach (binary classification)

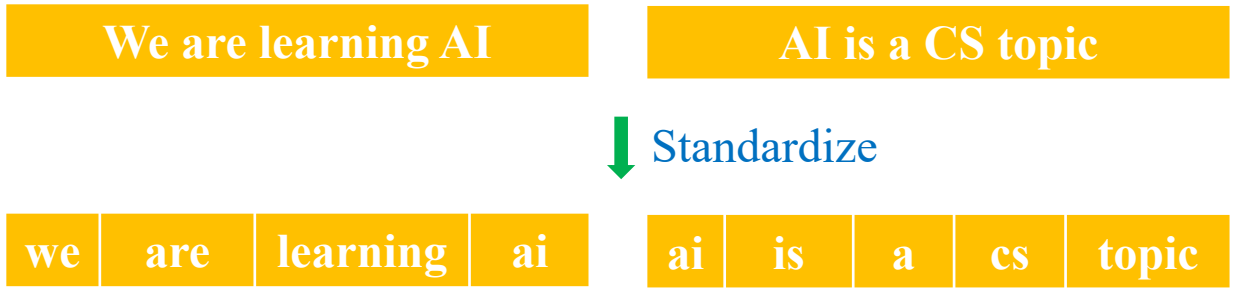


Preprocessing

index	0	1	2	3	4	5	6	7
word	[UNK]	pad	ai	a	are	cs	is	learning

- Example corpus
  - sample1: ‘We are learning AI’
  - sample2: ‘AI is a CS topic’

(1) Build vocabulary from corpus



Thành phần	Vai trò
Tokenizer	Lớp trung tâm, điều phối toàn bộ pipeline tokenize
WordLevel	Mô hình token hóa ở mức từ (word → id)
WordLevelTrainer	Cách học vocab (tần suất, giới hạn kích thước)
Whitespace	Pre-tokenizer tách từ theo khoảng trắng

```python
from tokenizers import Tokenizer
from tokenizers.models import WordLevel
from tokenizers.trainers import WordLevelTrainer
from tokenizers.pre_tokenizers import Whitespace
from tokenizers.normalizers import Lowercase

sample1 = 'We are learning AI'
sample2 = 'AI is a CS topic'
corpus = [sample1, sample2]

# Initialize the tokenizer and define a trainer
tokenizer = Tokenizer(WordLevel())
tokenizer.normalizer = Lowercase()
tokenizer.pre_tokenizer = Whitespace()
tokenizer.enable_padding(pad_id=1,
                         pad_token="<pad>",
                         length=5)
tokenizer.enable_truncation(max_length=5)

# Train the tokenizer on your corpus
trainer = WordLevelTrainer(vocab_size=8,
                           special_tokens=["<unk>",
                                           "<pad>"])
tokenizer.train_from_iterator(corpus, trainer)

vocab = tokenizer.get_vocab()
print(vocab)
```

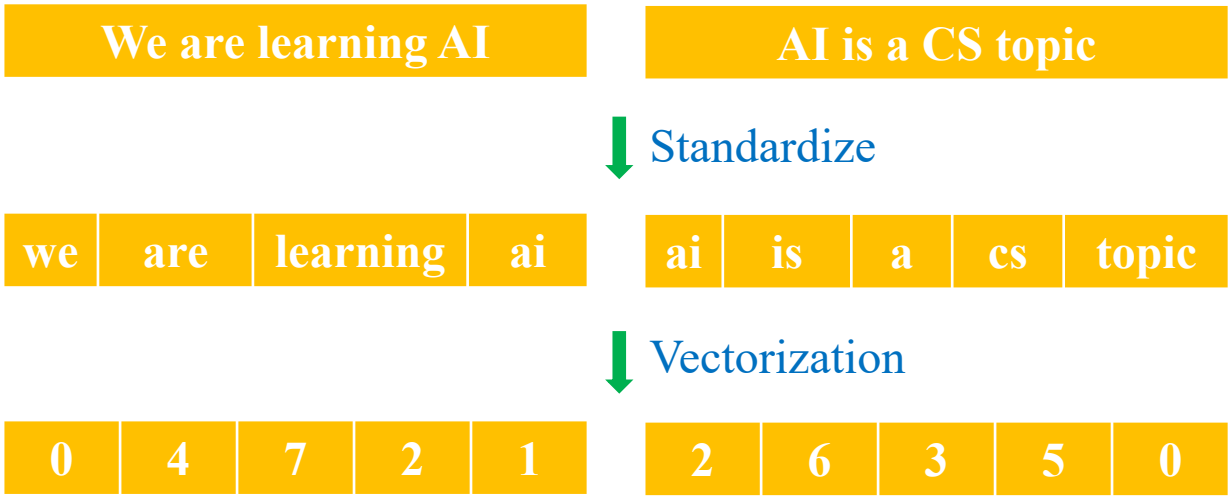
Preprocessing

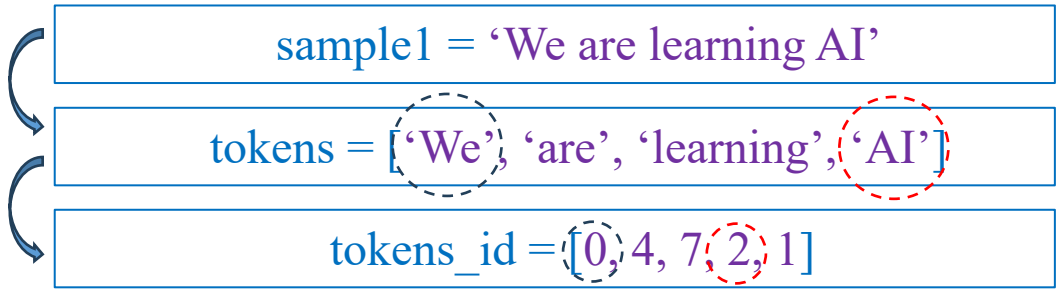
index	0	1	2	3	4	5	6	7
word	[UNK]	pad	ai	a	are	cs	is	learning

- Example corpus
  - sample1: ‘We are learning AI’
  - sample2: ‘AI is a CS topic’

(1) Build vocabulary from corpus

(2) Transform text into features






```
    output = tokenizer.encode(sample1)
    print(output.tokens)
    print(output.ids)
```

```
['<unk>', 'are', 'learning', 'ai', '<pad>']
[0, 4, 7, 2, 1]
```

```
    output = tokenizer.encode(sample2)
    print(output.tokens)
    print(output.ids)
```

```
['ai', 'is', 'a', 'cs', '<unk>']
[2, 6, 3, 5, 0]
```

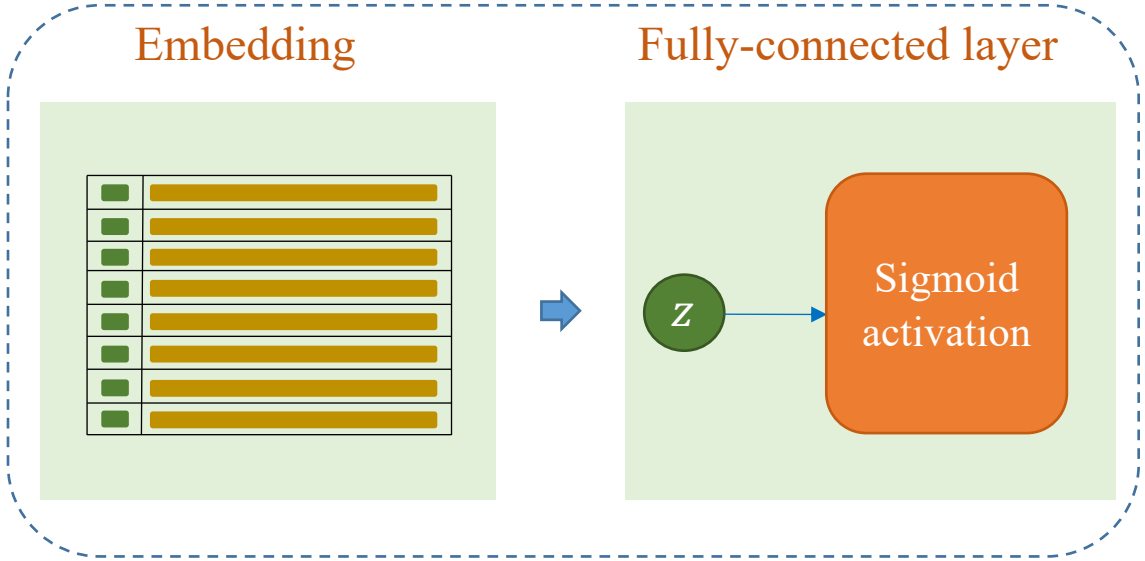
Embedding Layer

```python
import torch.nn as nn

vocab_size = 8
embed_dim = 4
embedding = nn.Embedding(vocab_size,
                         embed_dim)
```

```
Parameter containing:
tensor([[-0.1882,  0.5530,  1.6267,  0.7013],
        [ 1.7840, -0.8278, -0.2701,  1.3586],
        [ 1.0281, -1.9094,  0.3182,  0.4211],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2293,  1.3255,  0.1318,  2.0501],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4309, -1.3067, -0.8823,  1.5977]],
```

index	word
0	[UNK]
1	[pad]
2	ai
3	a
4	are
5	cs
6	is
7	learning

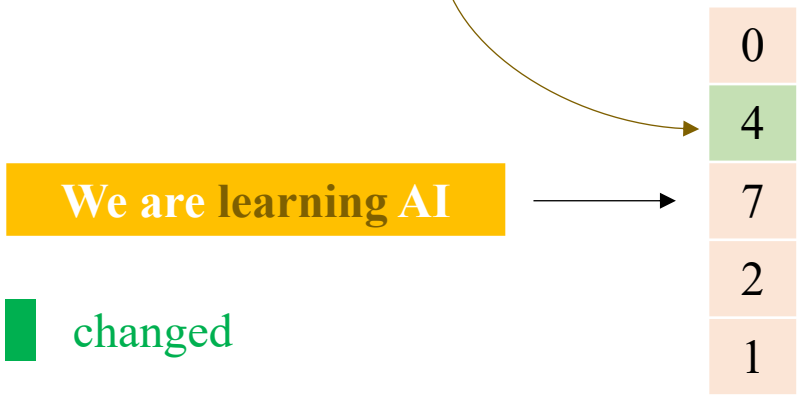
We are learning AI



Embedding Layer

(3) Embedding layer

index	word
0	[UNK]
1	[pad]
2	ai
3	a
4	are
5	cs
6	is
7	learning



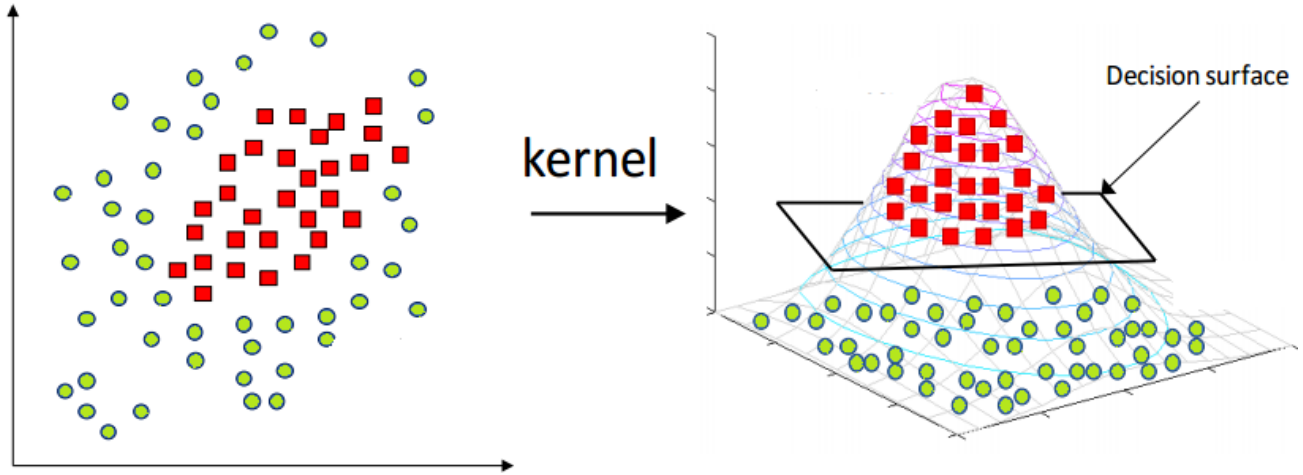

```
Parameter containing:
tensor([[-0.1882,  0.5530,  1.6267,  0.7013],
        [ 1.7840, -0.8278, -0.2701,  1.3586],
        [ 1.0281, -1.9094,  0.3182,  0.4211],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2293,  1.3255,  0.1318,  2.0501],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4309, -1.3067, -0.8823,  1.5977]],
```

```
Parameter containing:
tensor([[-0.1872,  0.5540,  1.6277,  0.7023],
        [ 1.7830, -0.8268, -0.2711,  1.3576],
        [ 1.0291, -1.9084,  0.3192,  0.4201],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2303,  1.3245,  0.1308,  2.0511],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4299, -1.3077, -0.8833,  1.5967]],
```

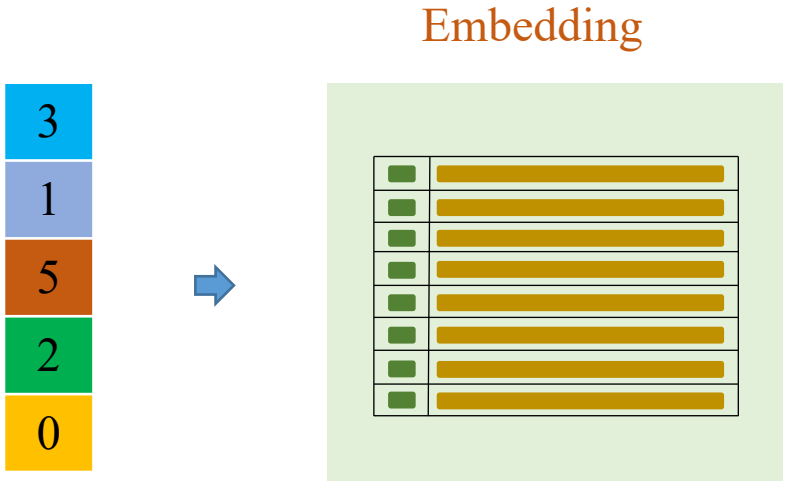
Word Embedding

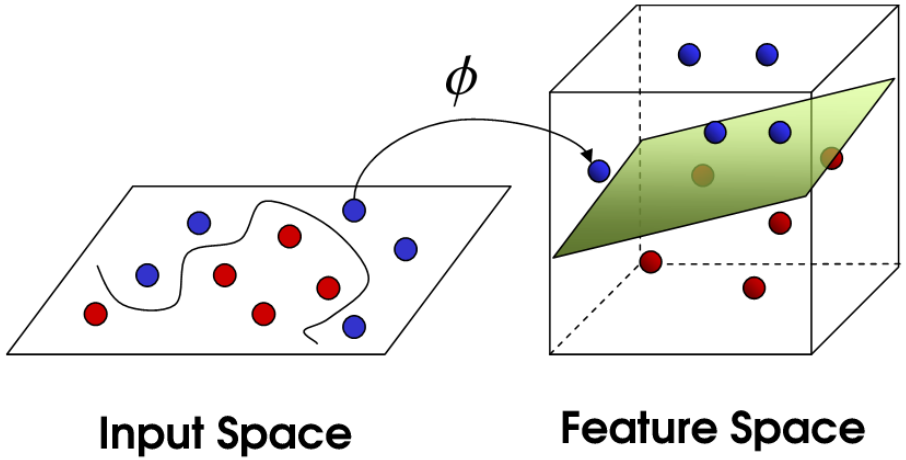
❖ **Why?**

1) Continuous values
2) Higher dimensions



https://codatalicious.medium.com/kernels-ee967067aa9





https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

Word Embedding

Convert from text to numbers

index	word
0	[UNK]
1	[pad]
2	ai
3	a
4	are
5	cs
6	is
7	learning

A sample X

We are learning AI

→

0
4
7
2
1

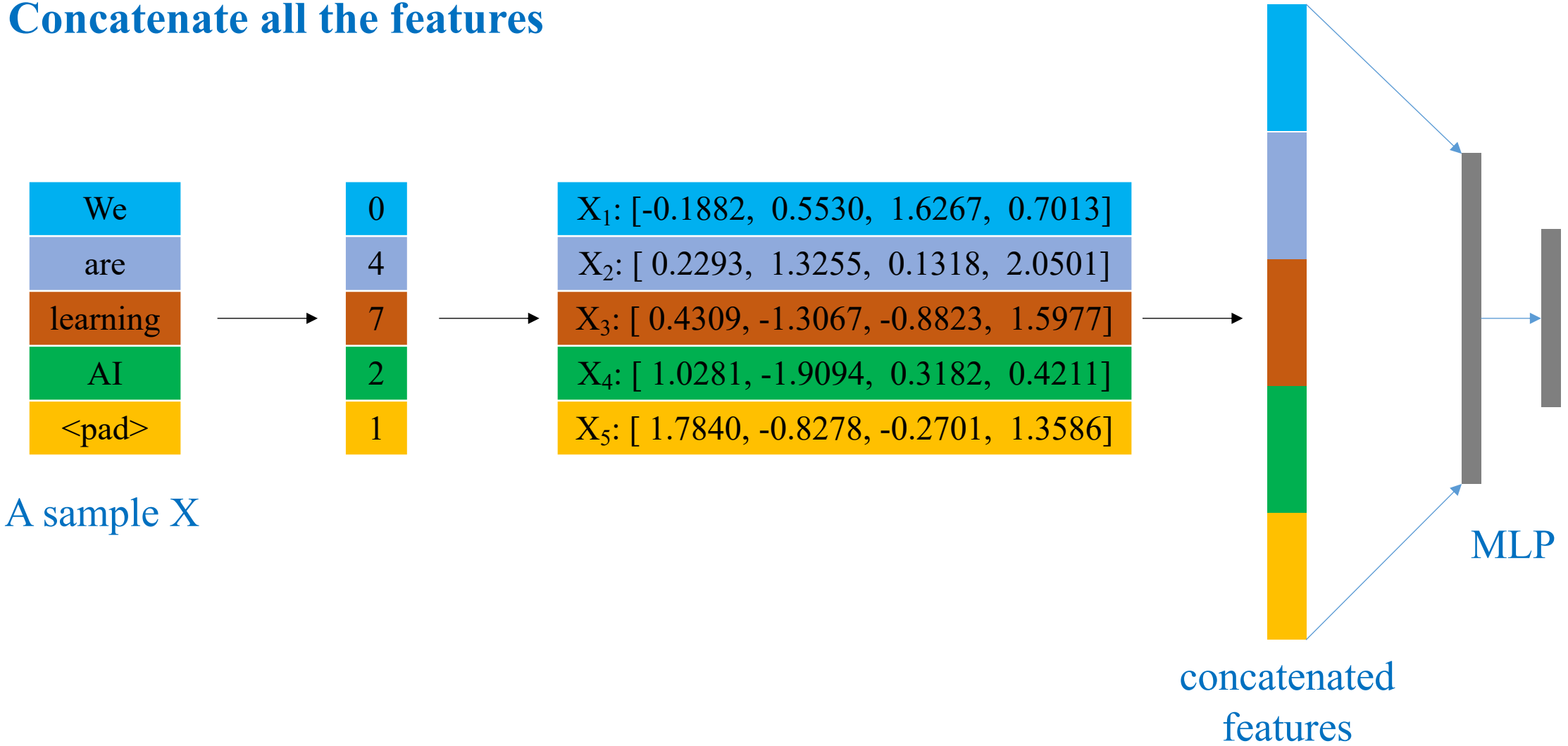
```
Parameter containing:
tensor([[-0.1882,  0.5530,  1.6267,  0.7013],
        [ 1.7840, -0.8278, -0.2701,  1.3586],
        [ 1.0281, -1.9094,  0.3182,  0.4211],
        [-1.3083, -0.0987,  0.7647, -0.3680],
        [ 0.2293,  1.3255,  0.1318,  2.0501],
        [ 0.4058, -0.6624, -0.8745,  0.7203],
        [ 0.5582,  0.0786, -0.6817,  0.6902],
        [ 0.4309, -1.3067, -0.8823,  1.5977]],
```

A sample X

We	0	X_1: [-0.1882, 0.5530, 1.6267, 0.7013]
are	4	X_2: [0.2293, 1.3255, 0.1318, 2.0501]
learning	7	X_3: [0.4309, -1.3067, -0.8823, 1.5977]
AI	2	X_4: [1.0281, -1.9094, 0.3182, 0.4211]
<pad>	1	X_5: [1.7840, -0.8278, -0.2701, 1.3586]

How to deal with this input?

- **Simplest idea: Based on MLP**
- **Concatenate all the features**



Text Classification: Example 1

Step-by-Step Example: Text Classification

Doc	Label
gậy ông đập lưng ông	0
có làm mới có ăn	1

Training data
- negative (0)
- positive (1)

building dictionary

index	word
0	[UNK]
1	[pad]
2	có
3	ông
4	gập
5	làm
6	lưng
7	mới

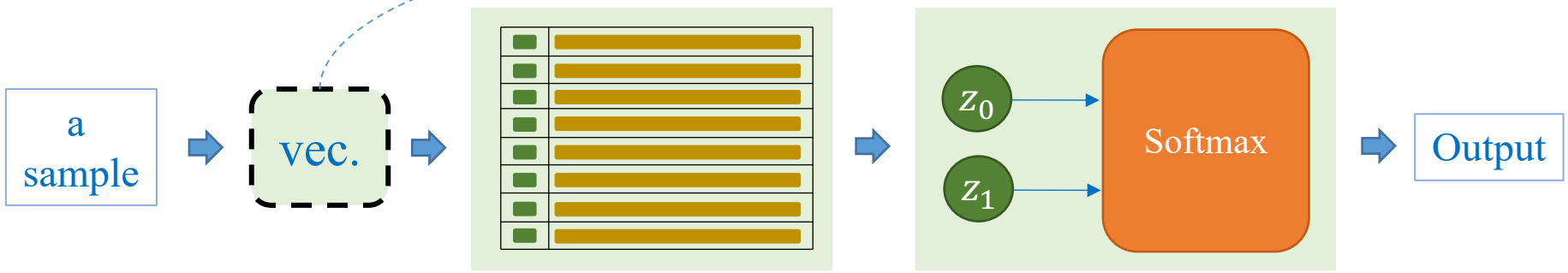
vocab size = 8
sequence length = 5

gậy	4
ông	3
đập	0
lưng	6
ông	3

sample 1

có	2
làm	5
mới	7
có	2
ăn	0

sample 2



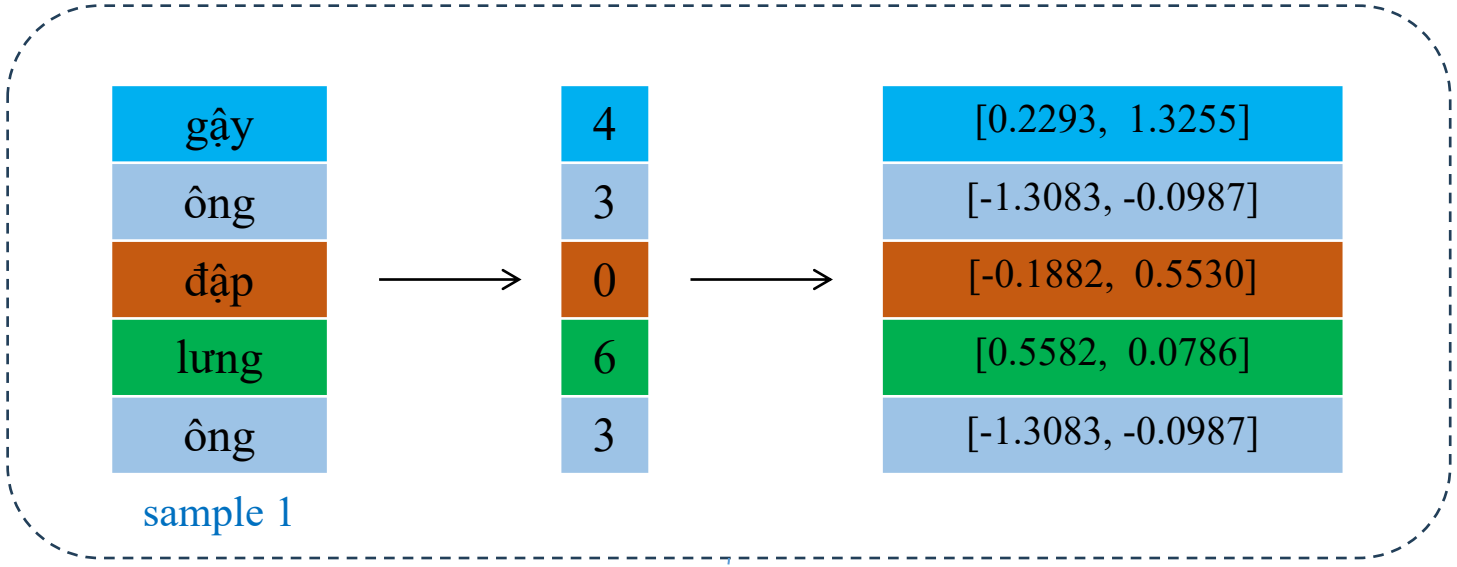
Step-by-Step Example: Text Classification

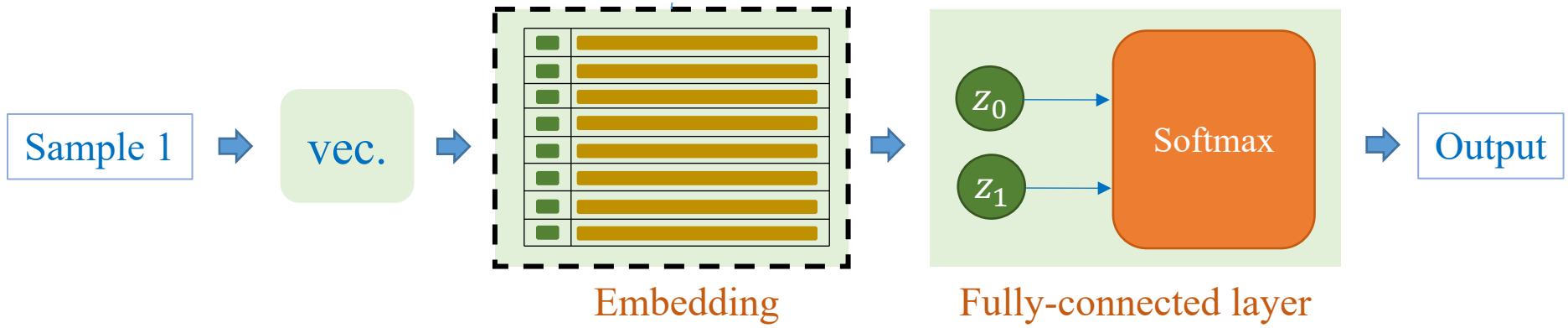
Doc	Label
gậy ông đập lưng ông	0
có làm mới có ăn	1

0	[-0.1882, 0.5530]
1	[1.7840, -0.8278]
2	[1.0281, -1.9094]
3	[-1.3083, -0.0987]
4	[0.2293, 1.3255]
5	[0.4058, -0.6624]
6	[0.5582, 0.0786]
7	[0.4309, -1.3067]

Embedding 8x2
(Random initialization)

parameter



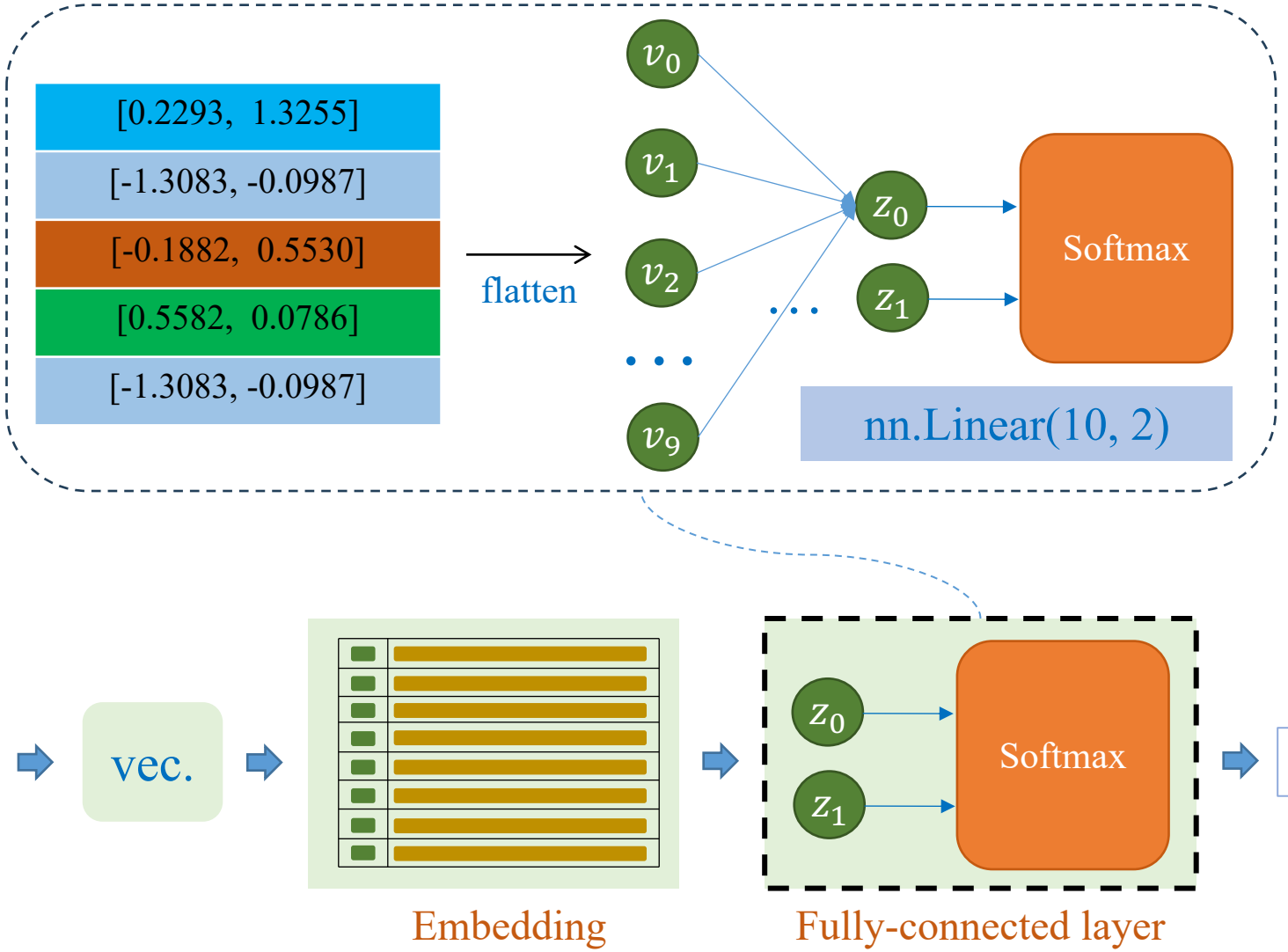


Step-by-Step Example: Text Classification

Doc	Label
gậy ông đập lưng ông	0
có làm mới có ăn	1

0	[-0.1882, 0.5530]
1	[1.7840, -0.8278]
2	[1.0281, -1.9094]
3	[-1.3083, -0.0987]
4	[0.2293, 1.3255]
5	[0.4058, -0.6624]
6	[0.5582, 0.0786]
7	[0.4309, -1.3067]

Embedding



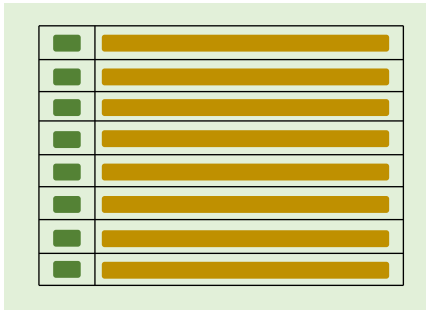
Example
Text Classification

Doc	Label
gậy ông đập lưng ông	0
có làm mới có ăn	1

0	[-0.1882, 0.5530]
1	[1.7840, -0.8278]
2	[1.0281, -1.9094]
3	[-1.3083, -0.0987]
4	[0.2293, 1.3255]
5	[0.4058, -0.6624]
6	[0.5582, 0.0786]
7	[0.4309, -1.3067]

Embedding

Sample 1 → vec. →

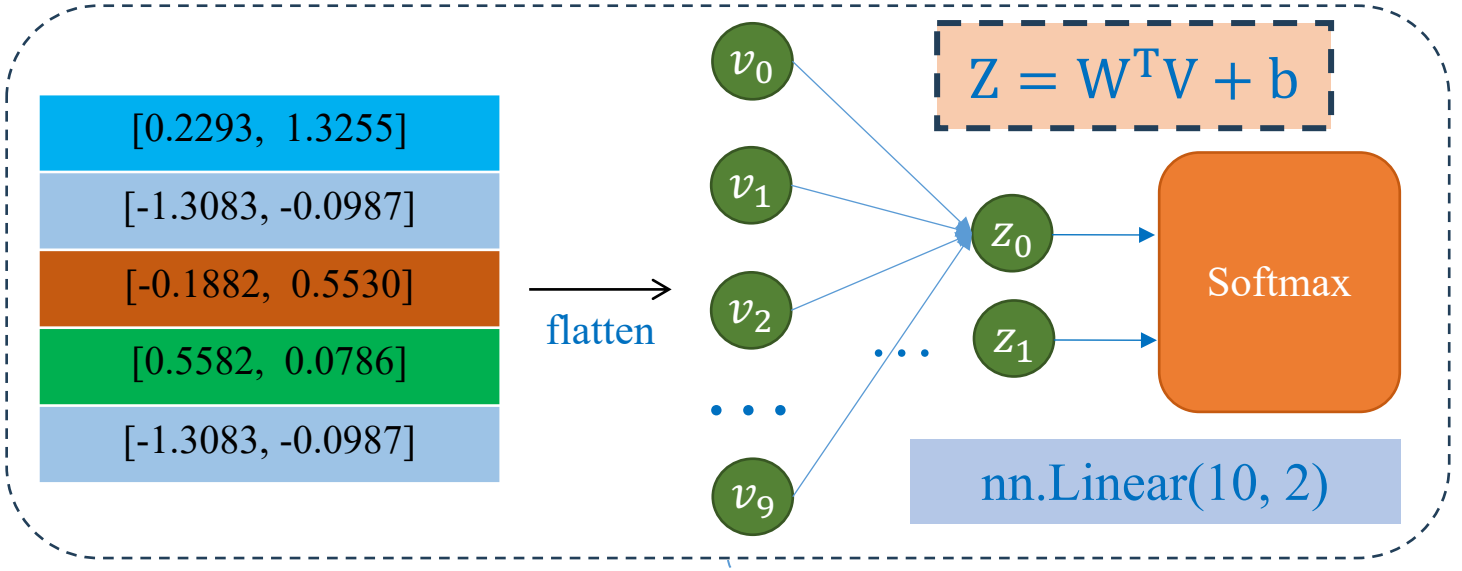


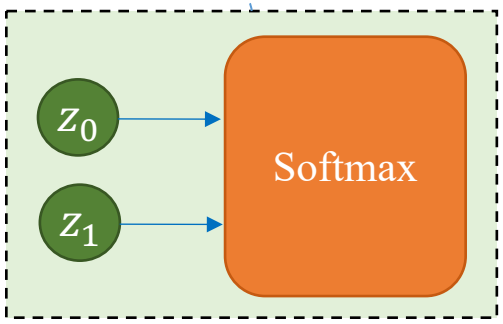
W
[0.2108, -0.0074, 0.2760, 0.2325, -0.0518, -0.1876, 0.0194, 0.0378, 0.0210, 0.2982]
[0.0284, 0.2968, -0.0260, 0.1251, -0.0282, 0.0175, -0.1817, 0.2483, 0.2338, 0.2985]

b [-0.3049, 0.1028]

z [-0.7875, 0.1221]

$\hat{y}$ [0.28, 0.71]





Output → loss=1.25

y = 0

Example
Text Classification

Doc	Label
gậy ông đập lưng ông	0
có làm mới có ăn	1

$$\theta_t = \theta_{t-1} - \eta \nabla_\theta L$$

update

0	[-0.1899, 0.5384]
1	[1.7840, -0.8278]
2	[1.0281, -1.9094]
3	[-1.3019, -0.0911]
4	[0.2423, 1.3038]
5	[0.4058, -0.6624]
6	[0.5725, 0.0636]
7	[0.4309, -1.3067]

Embedding

W

[0.3108, 0.0926, 0.1760, 0.1325, -0.1518, -0.0876, 0.1194, 0.1378, -0.0790, 0.1982]

[-0.0716, 0.1968, 0.0740, 0.2251, 0.0718, -0.0825, -0.2817, 0.1483, 0.3338, 0.3985]

```
nn.CrossEntropyLoss()
torch.optim.SGD(model.parameters(),
                lr=0.1)
```

b

[-0.2049, 0.0028]

[0.2423, 1.3038]

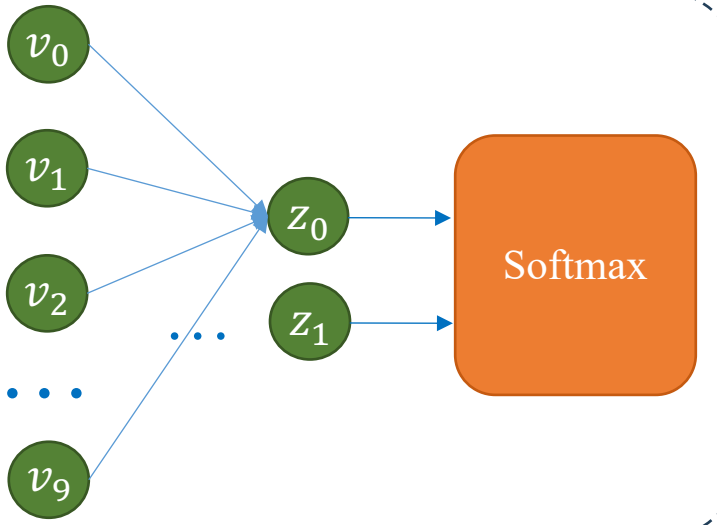
[-1.3019, -0.0911]

[-0.1899, 0.5384]

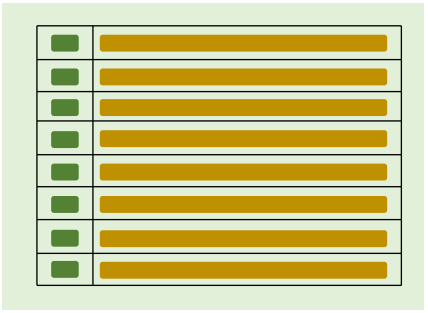
[0.5725, 0.0636]

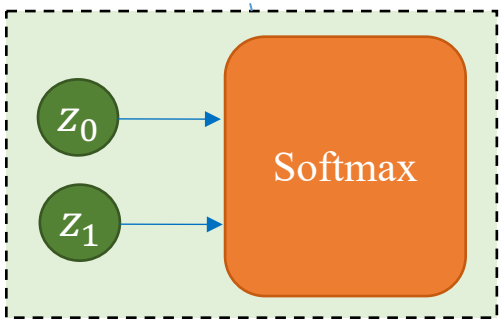
[-1.3019, -0.0911]

flatten



Sample 1 → vec. →





→ Output

Example
Text Classification

Doc	Label
gậy ông đập lưng ông	0
có làm mới có ăn	1

W

[0.3108, 0.0926, 0.1760, 0.1325, -0.1518, -0.0876, 0.1194, 0.1378, -0.0790, 0.1982]

[-0.0716, 0.1968, 0.0740, 0.2251, 0.0718, -0.0825, -0.2817, 0.1483, 0.3338, 0.3985]

b

[-0.2049, 0.0028]

z

[-0.0261, -0.5182]

$\hat{y}$

↑ [0.63, 0.37] ↓

- Loss reduces
- $\hat{y}_0$ increases
- $\hat{y}_1$ reduces

0	[-0.1899, 0.5384]
1	[1.7840, -0.8278]
2	[1.0281, -1.9094]
3	[-1.3019, -0.0911]
4	[0.2423, 1.3038]
5	[0.4058, -0.6624]
6	[0.5725, 0.0636]
7	[0.4309, -1.3067]

Embedding

[0.2423, 1.3038]

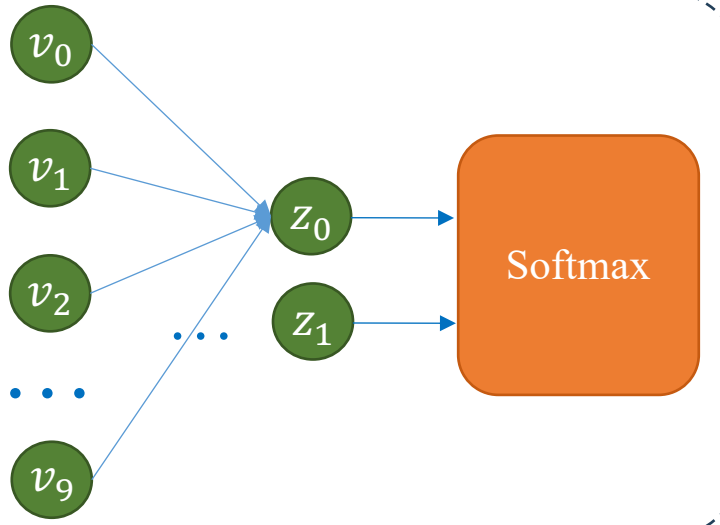
[-1.3019, -0.0911]

[-0.1899, 0.5384]

[0.5725, 0.0636]

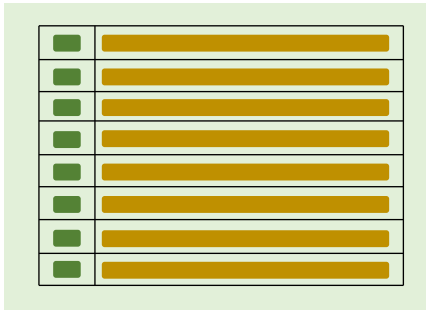
[-1.3019, -0.0911]

flatten

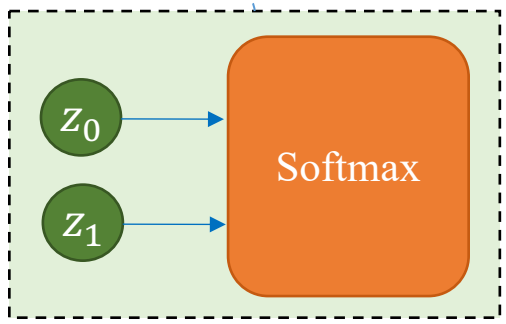


Feed sample 1 for the second time

Sample 1 → vec. →



→



→ Output → loss=0.65

y = 0

Model is learning

Linear Layer

2.2.text_classification_MLP2.ipynb

W
[0.3847, -0.4621, 0.1749]
[-0.3024, -0.1529, 0.4329]

b
[0.3548, -0.2819]

V_1 [-0.40, 0.66, -0.87]
V_2 [0.43, -1.30, 0.88]
V_3 [-0.18, 0.55, 1.62]
V_4 [-1.30, 0.09, 0.76]

Vectorized and Embedded Input
shape = (N=1, L=4, **E_dim=3**)

Linear(3, 2)

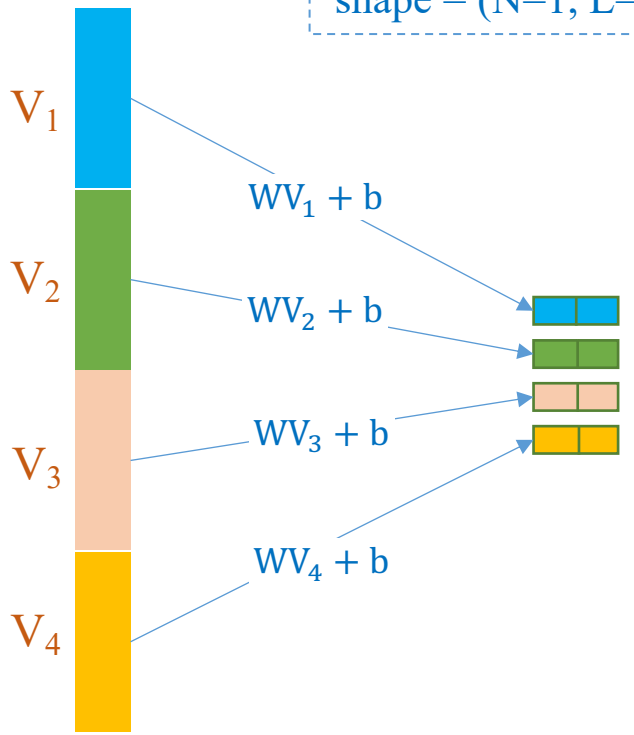
[0.58, -0.45]
[-0.09, -0.23]
[0.82, -1.67]
[-1.83, 0.47]

shape = (N=1, L=4, **C=2**)

```
import torch
import torch.nn as nn

linear = nn.Linear(3, 2)
input = torch.randn(1, 4, 3)
output = linear(input)
print(output.shape)
```

✓ 0.0s

```
torch.Size([1, 4, 2])
```



Outline

SECTION 1

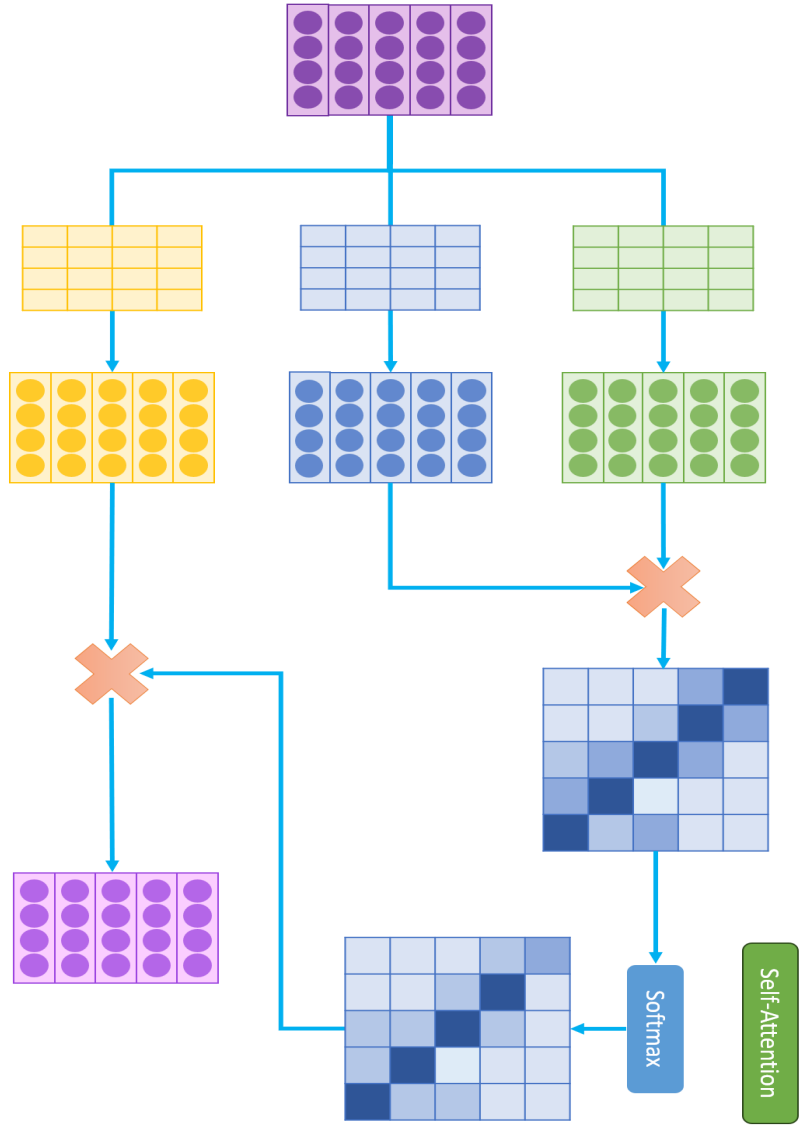
Text Classification

SECTION 2

RNN → Transformer

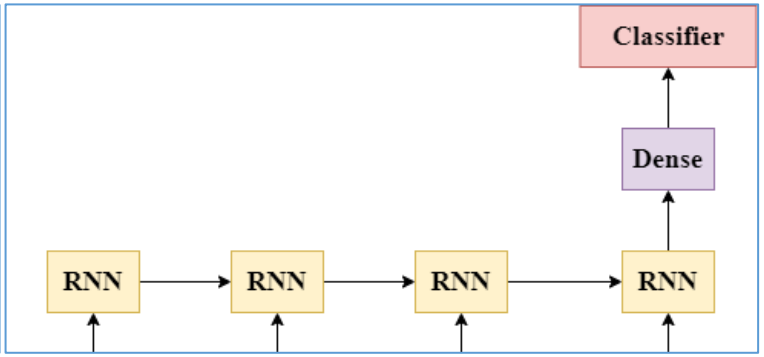
SECTION 3

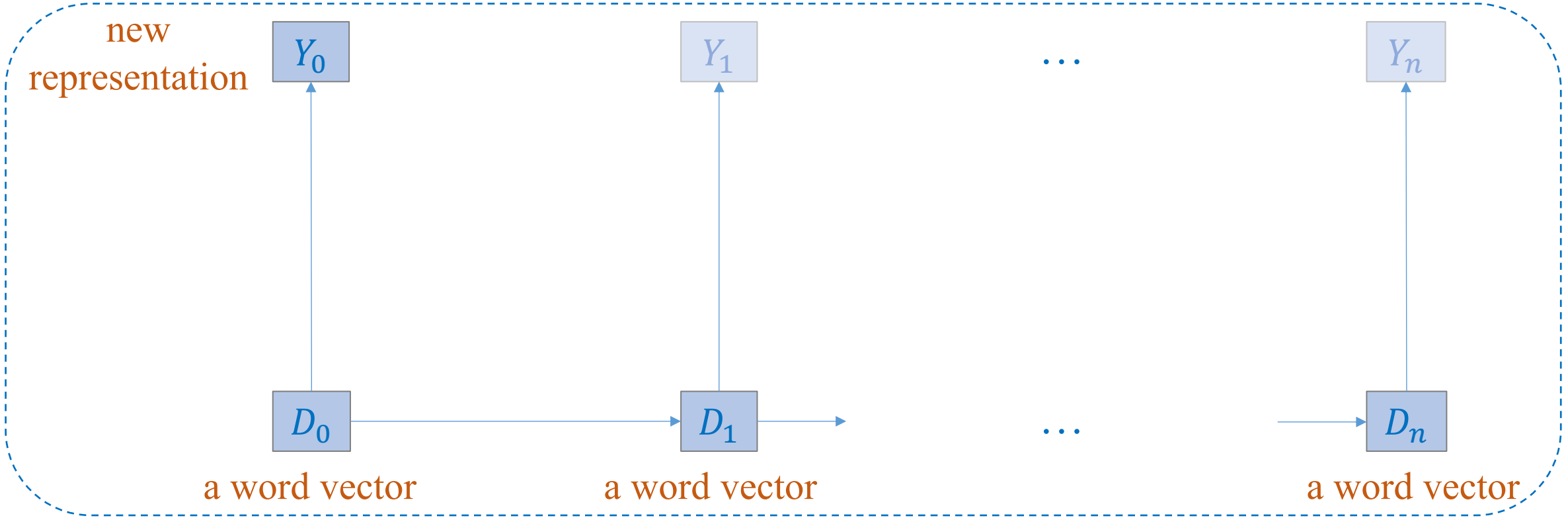
Application



From RNNs to Transformers

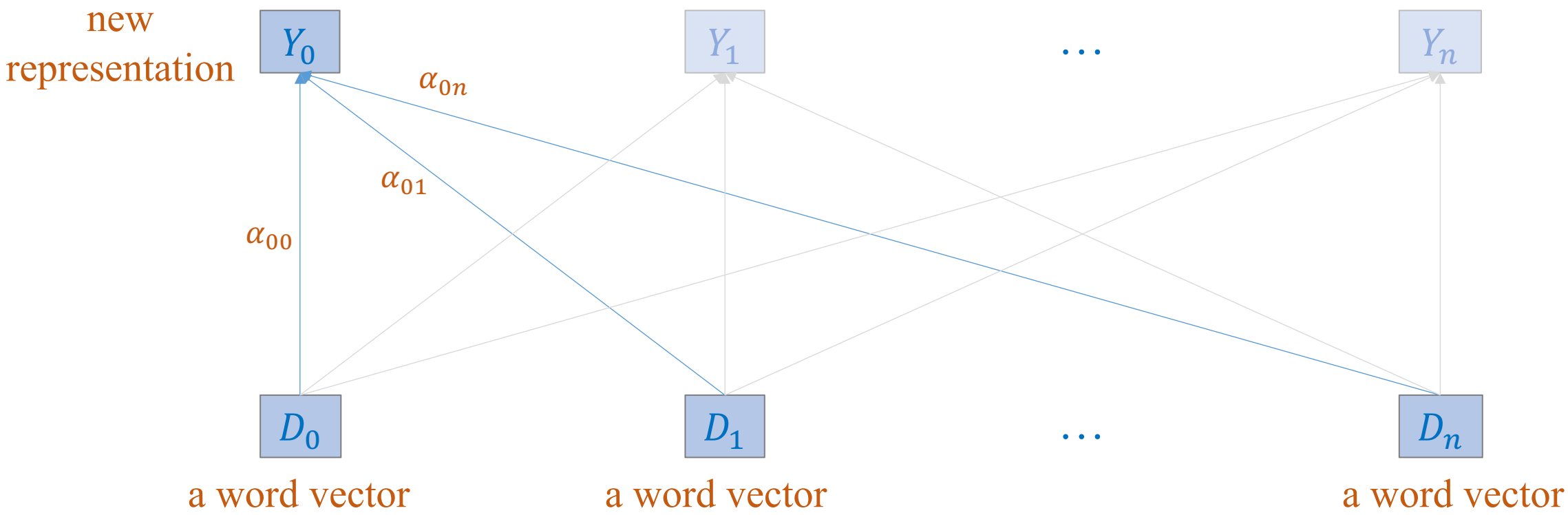
❖ RNN/LSTM/GRU Limitations





From RNNs to Transformers

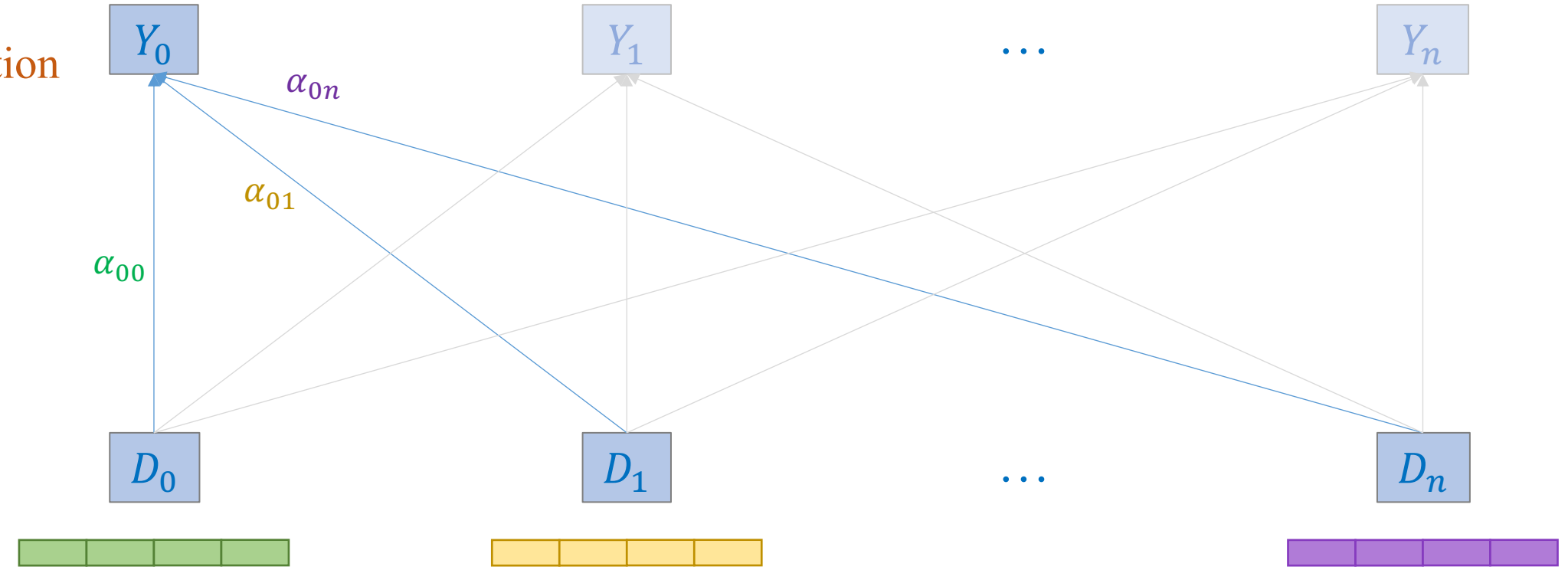
❖ Desire properties



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

Desire Property

new representation

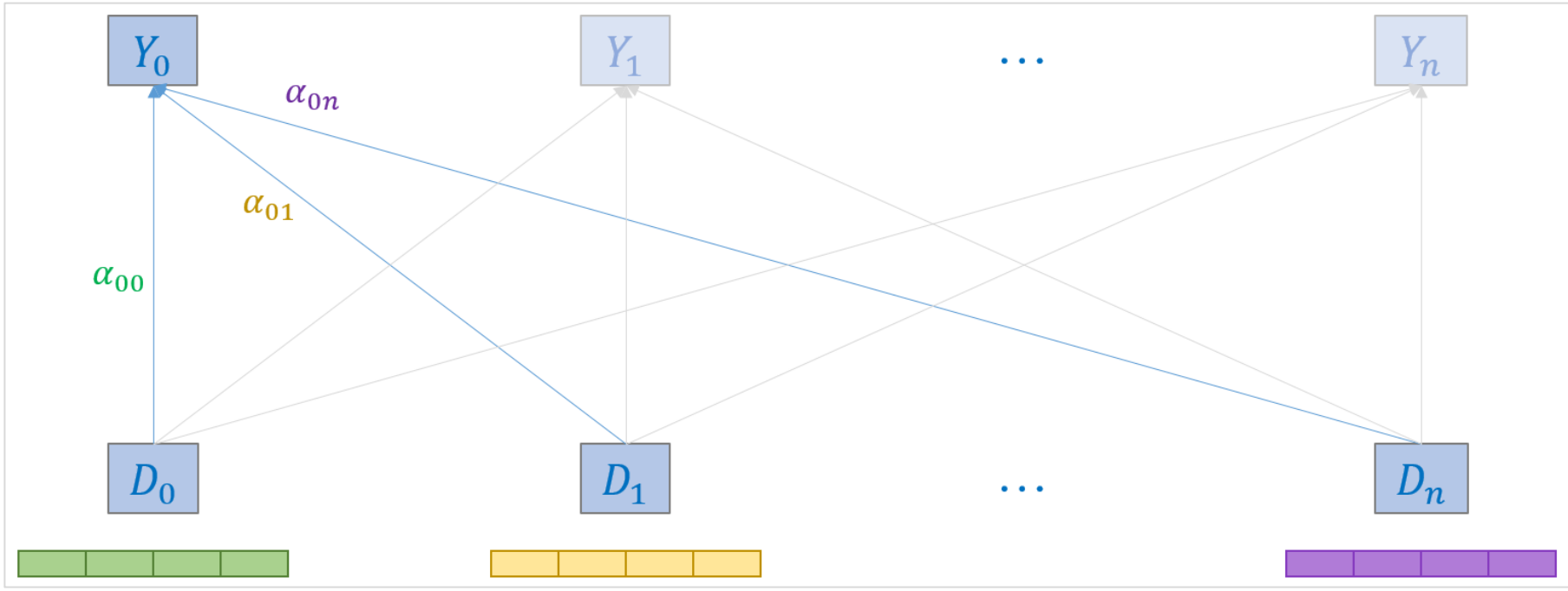


$$Y_0 = \alpha_{00} + \alpha_{01} + \cdots + \alpha_{0n}$$

$$\alpha_{0i} = \cdot$$

Desire Property

$$Y_0 = \alpha_{00} + \alpha_{01} + \alpha_{0n}$$

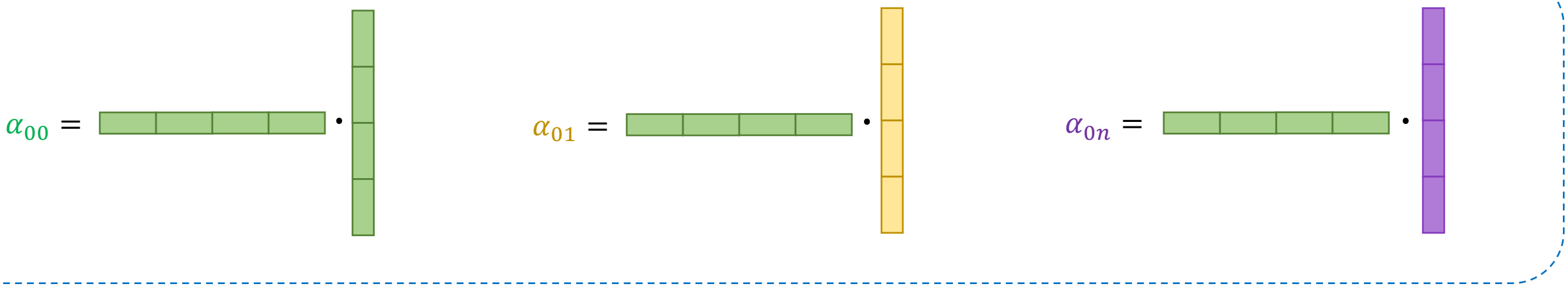


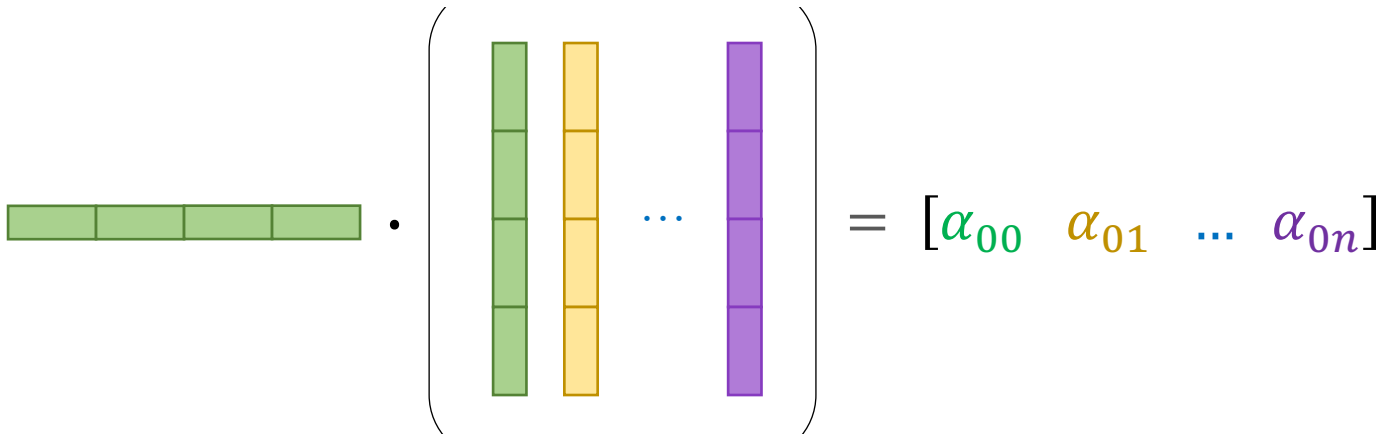
$$\alpha_{00} = \cdot$$

$$\alpha_{01} = \cdot$$

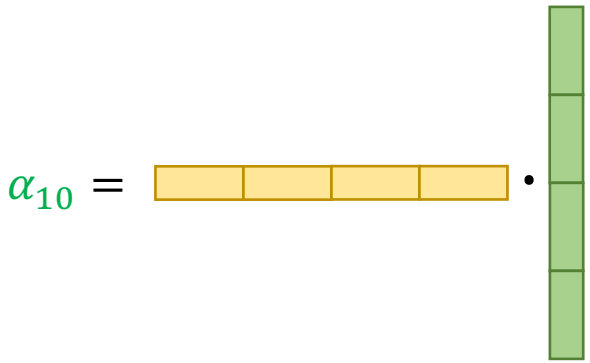
$$\alpha_{0n} = \cdot$$

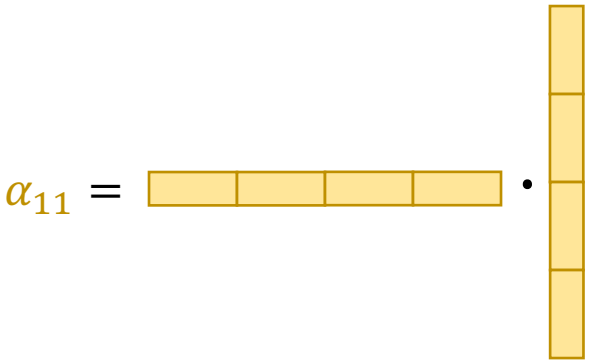
Context Awareness

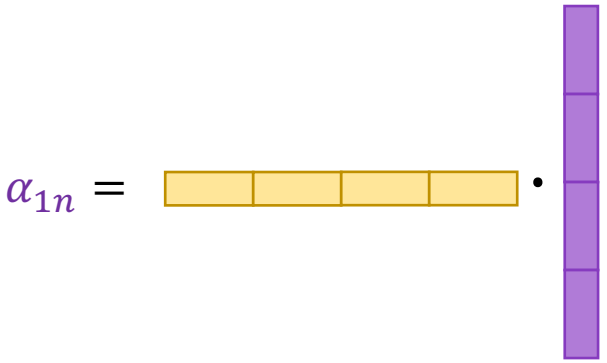


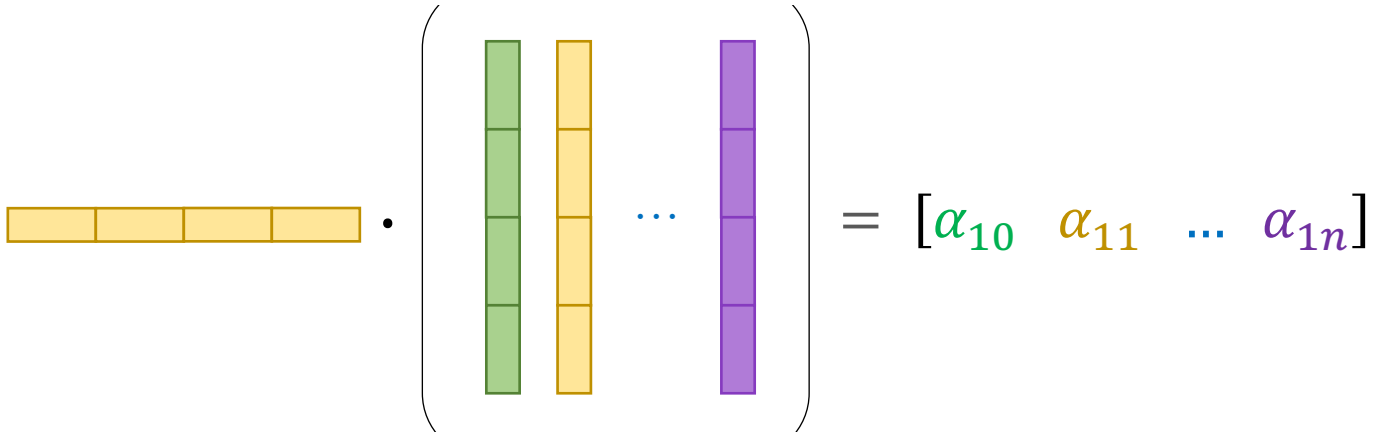


Context Awareness

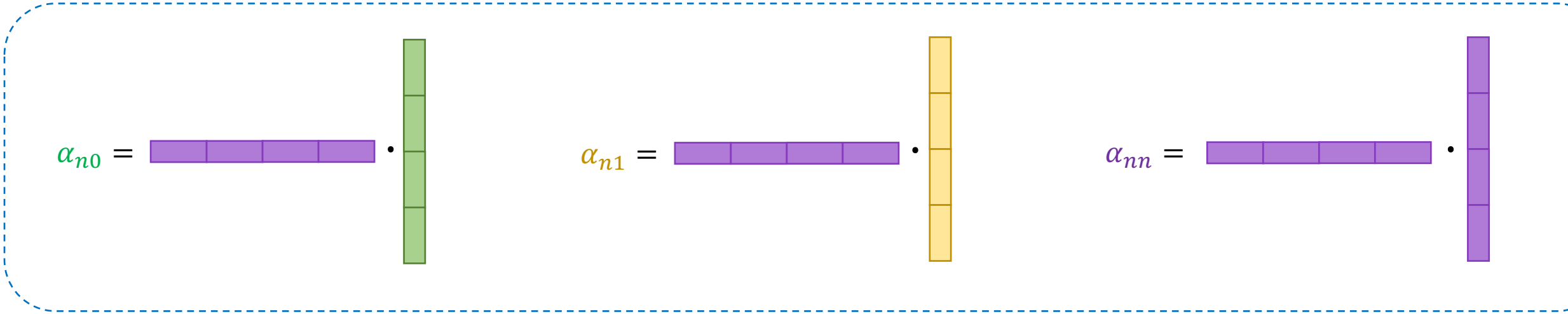


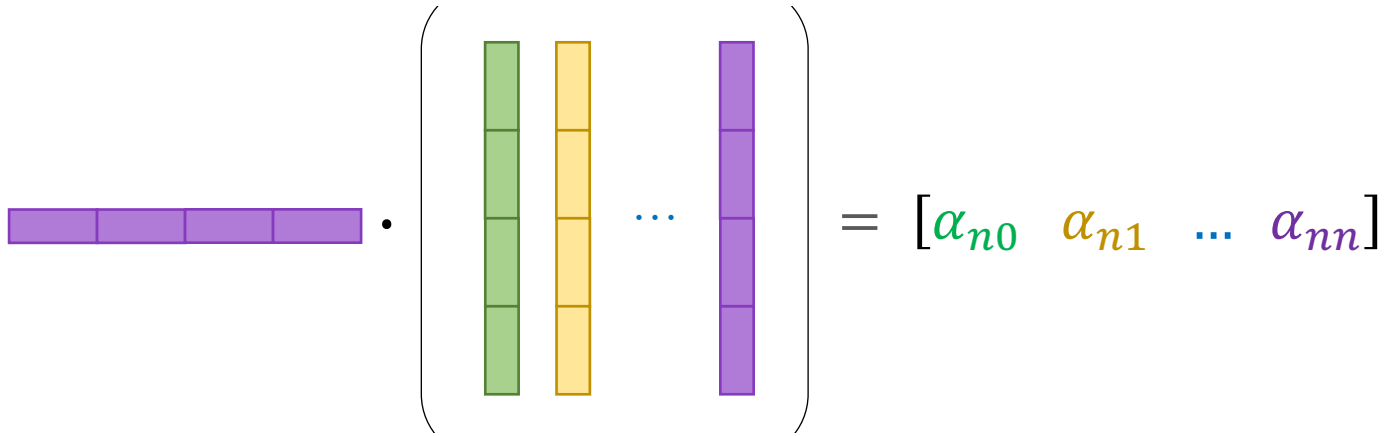


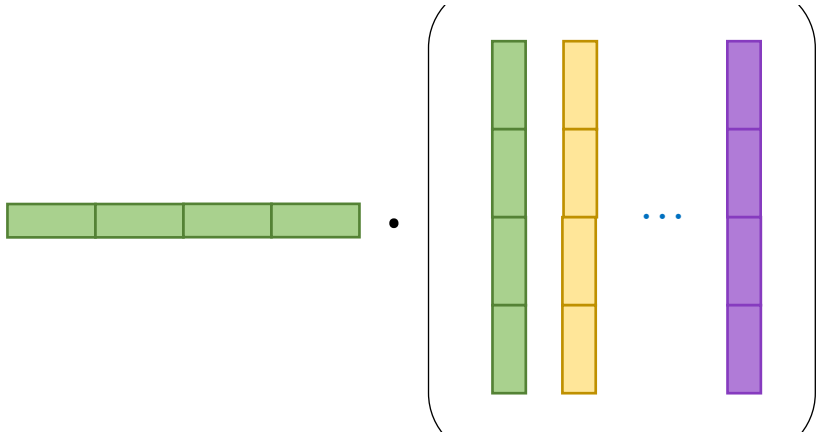




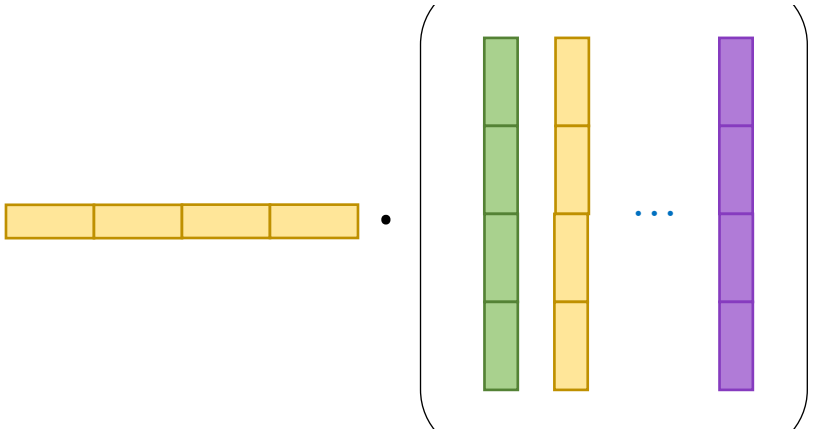
Context Awareness





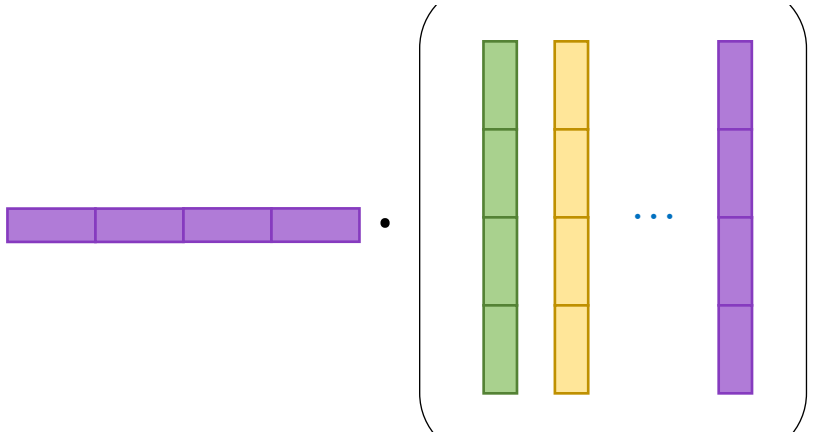


$$= [\alpha_{00} \quad \alpha_{01} \quad \dots \quad \alpha_{0n}]$$

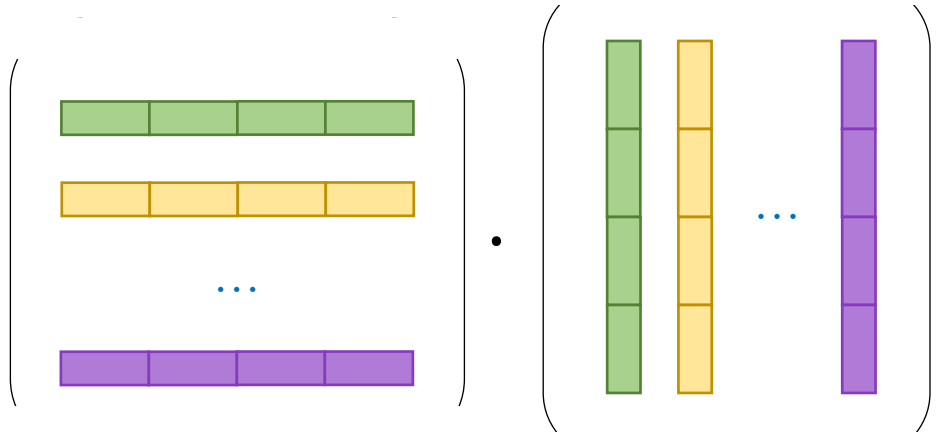


$$= [\alpha_{10} \quad \alpha_{11} \quad \dots \quad \alpha_{1n}]$$

…



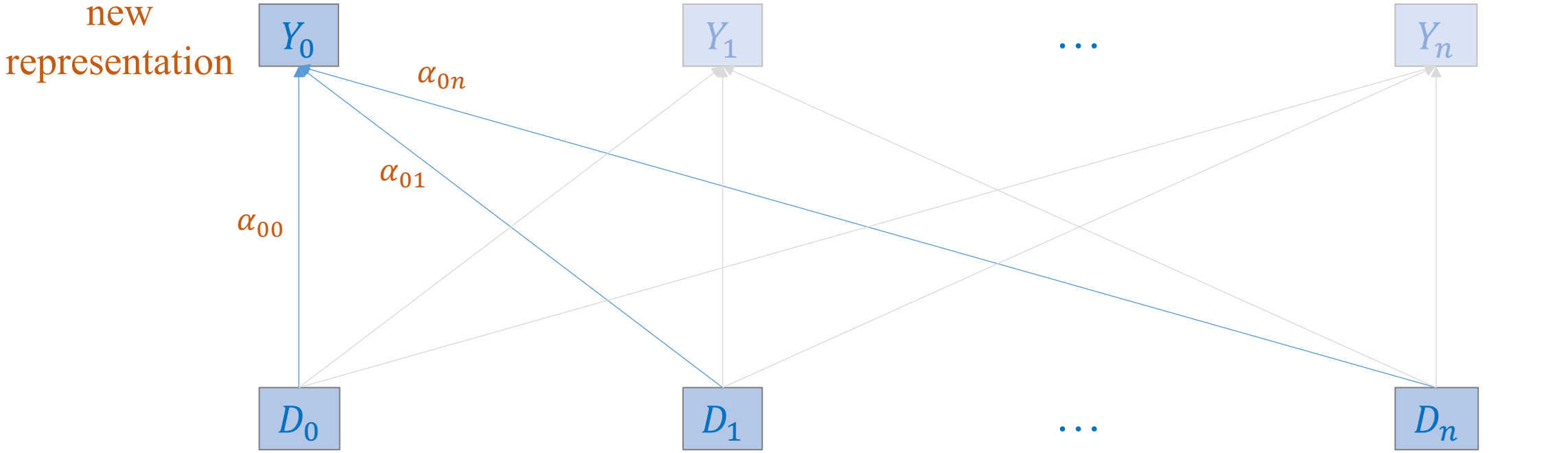
$$= [\alpha_{n0} \quad \alpha_{n1} \quad \dots \quad \alpha_{nn}]$$



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

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

$$\alpha = DD^T$$



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

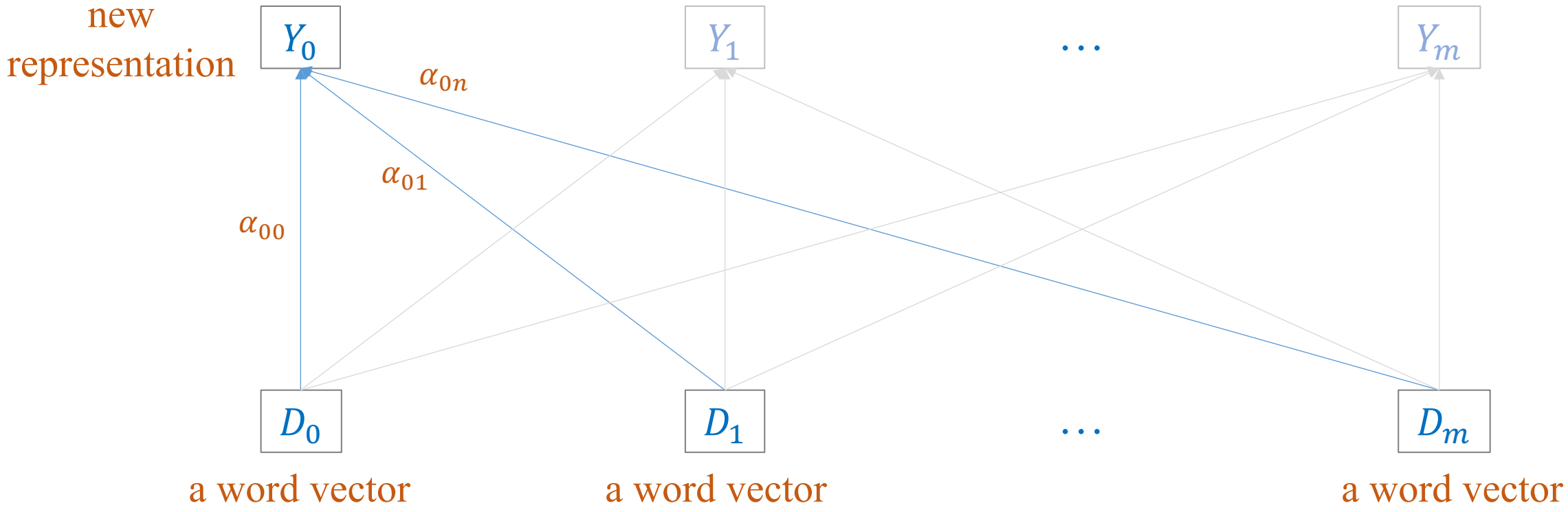
weight: determine how much X_0 contributes to Y_0

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

normalization

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

$$\alpha = \text{softmax}(DD^T)$$



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

weight: determine how much X_0 contributes to Y_0

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

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

weight vector

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

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

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

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

weight vector

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

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

Contextualized representation

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

❖ Implementation

❖ Implementation

```python
class MyMultiheadAttention(nn.Module):
    def __init__(self, embed_dim):
        super(MyMultiheadAttention, self).__init__()
        self.embed_dim = embed_dim

    def forward(self, query, key, value):
        scores = torch.matmul(query, key.transpose(-2, -1)) / (self.embed_dim ** 0.5)
        attention_weights = F.softmax(scores, dim=-1)
        output = torch.matmul(attention_weights, value)

        return output

class TransformerTextCls(nn.Module):
    def __init__(self, vocab_size, max_length, embed_dim, dropout, device):
        super().__init__()
        self.word_emb = nn.Embedding(num_embeddings=vocab_size, embedding_dim=embed_dim)
        self.attn = MyMultiheadAttention(embed_dim=embed_dim)
        self.fc = nn.Linear(in_features=200*32, out_features=2)

    def forward(self, x):
        output = self.word_emb(x)
        query, key, value = output, output, output
        attn_output = self.attn(query, key, value)

        output = nn.Flatten()(output)
        output = self.fc(output)
        return output
```

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

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

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

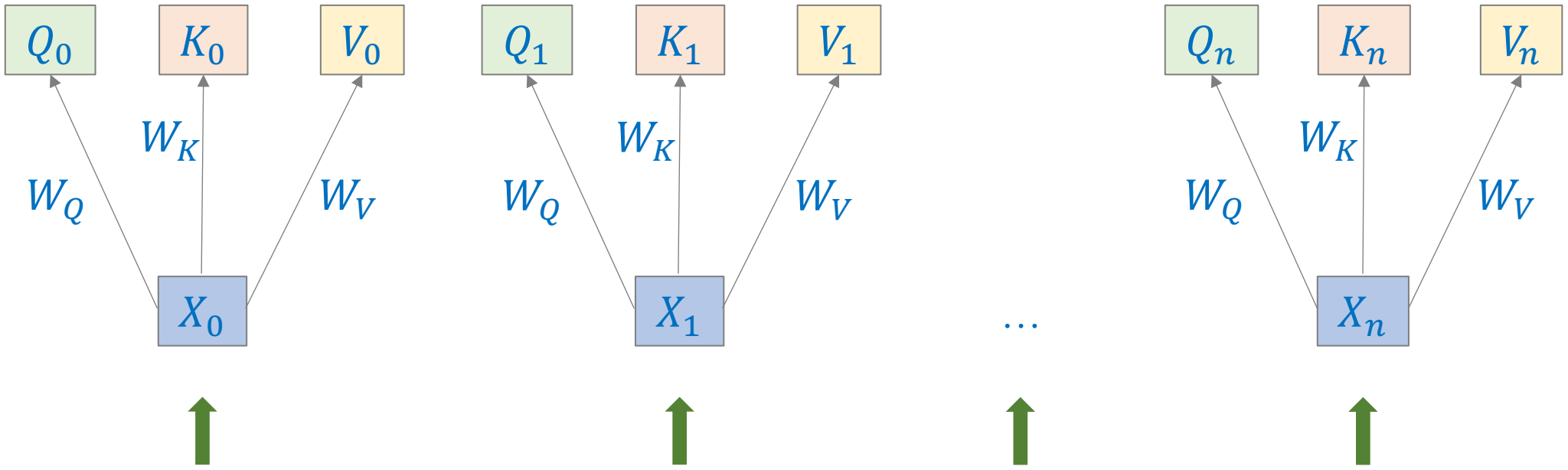
$$W_V = [\theta_{v0} \quad \theta_{v1} \quad \dots \quad \theta_{vm}] \in \mathcal{R}^{d \times m}$$

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

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

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

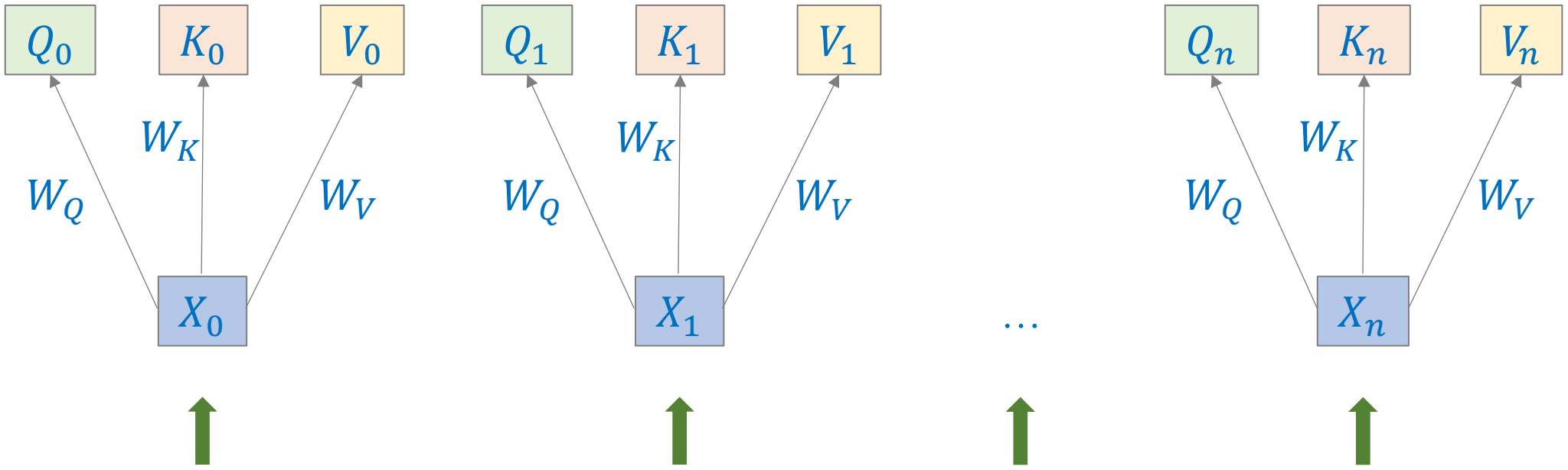
$$\text{Value } V = XW_V \in \mathcal{R}^{n \times m}$$



Contextualized representation

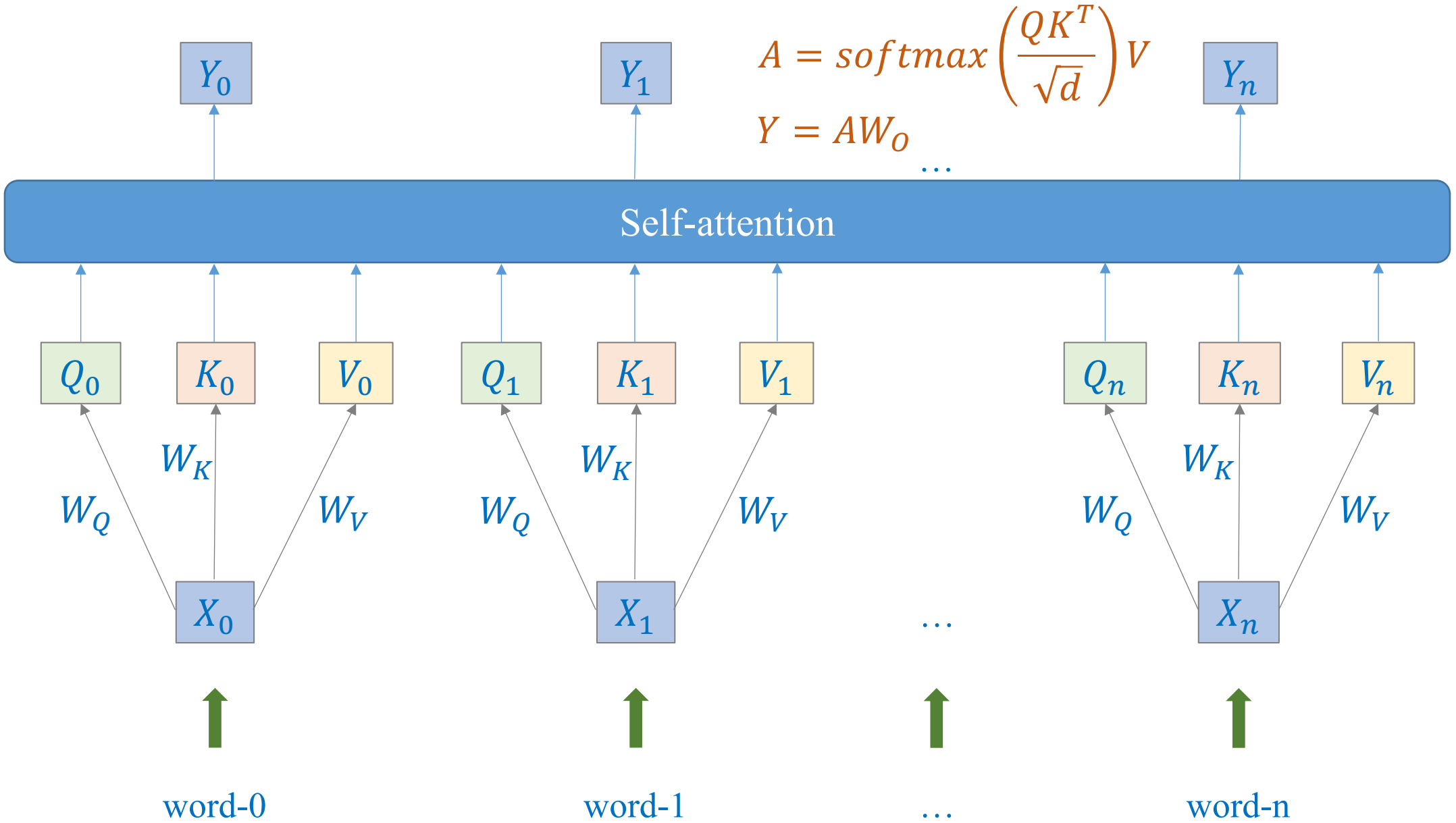
$$Y = \alpha D = \text{softmax}(\frac{DD^T}{\sqrt{d}})D$$

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



Self-Attention

```
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)
```



Self-Attention

❖ **Example 1**

$$X = [-0.1 \;\; 0.1 \;\; 0.3]$$

$$W_Q = \begin{bmatrix} -0.35 & 0.51 & 0.50 \\ 0.36 & -0.47 & -0.29 \\ -0.51 & -0.14 & -0.56 \end{bmatrix} \quad W_K = \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix} \quad W_V = \begin{bmatrix} -0.41 & 0.39 & -0.65 \\ -0.40 & -0.07 & -0.34 \\ -0.55 & -0.13 & -0.29 \end{bmatrix} \quad W_O = \begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix}$$

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

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

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

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

head = 1

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

❖ Example 2

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

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

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

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

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

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

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

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

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

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

$$= \begin{bmatrix} -0.16 & -0.08 & -0.05 \\ 0.45 & 0.27 & 0.20 \end{bmatrix}$$

Self-Attention

❖ Example 2

approximately

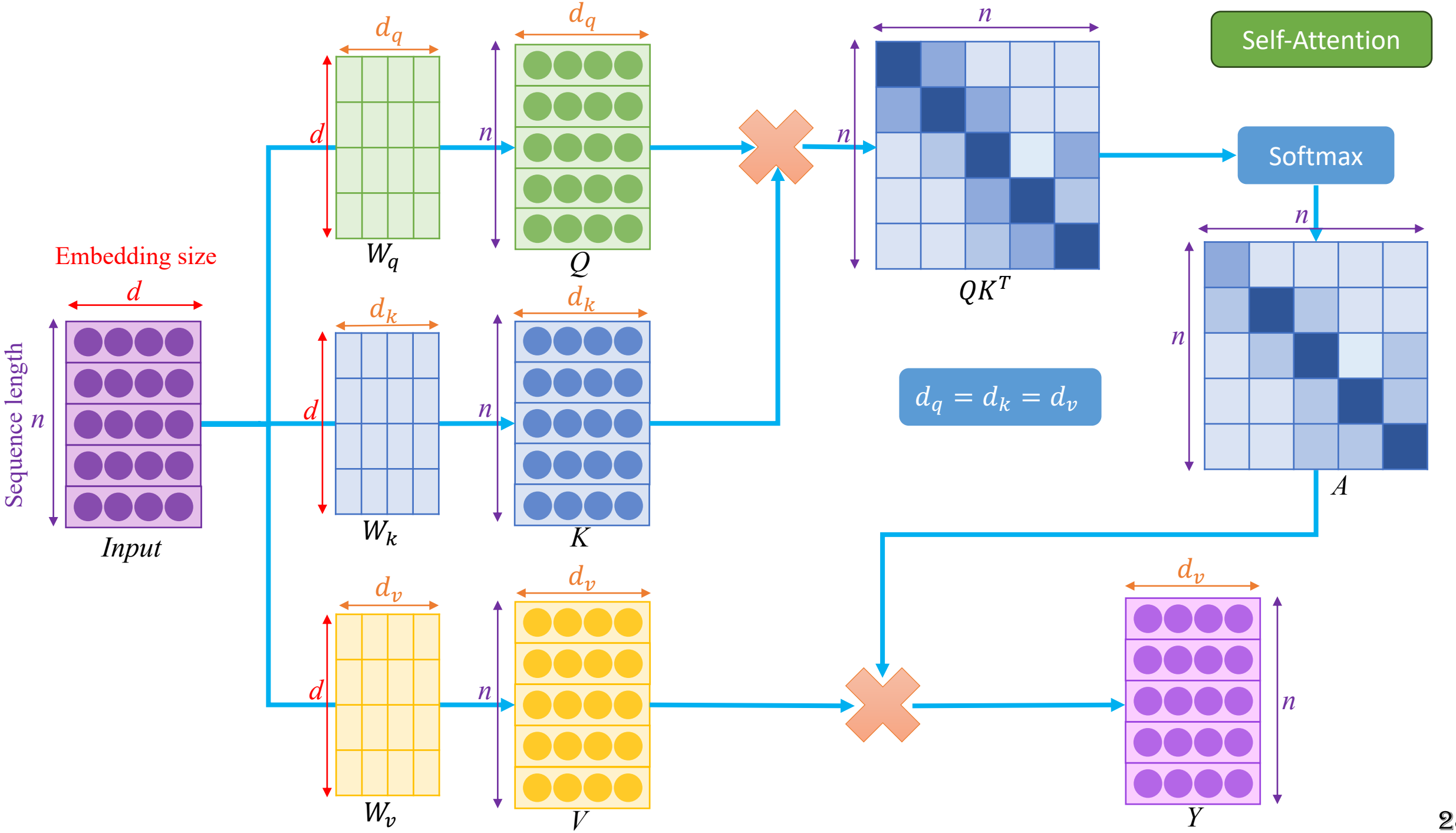
$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$= \text{softmax}\left(\begin{bmatrix}-0.08 & -0.14 & -0.24\\ -0.39 & 0.77 & 0.69\end{bmatrix}\begin{bmatrix}0.02 & 0.27\\ -0.01 & 0.27\\ 0.13 & -0.26\end{bmatrix}\frac{1}{\sqrt{d}}\right)\begin{bmatrix}-0.16 & -0.08 & -0.05\\ 0.45 & 0.27 & 0.20\end{bmatrix}$$

$$= \text{softmax}\left(\begin{bmatrix}-0.019 & 0.002\\ 0.043 & -0.046\end{bmatrix}\right)\begin{bmatrix}-0.16 & -0.08 & -0.05\\ 0.45 & 0.27 & 0.20\end{bmatrix}$$

$$= \begin{bmatrix}0.49 & 0.51\\ 0.52 & 0.48\end{bmatrix}\begin{bmatrix}-0.16 & -0.08 & -0.05\\ 0.45 & 0.27 & 0.20\end{bmatrix} = \begin{bmatrix}0.14 & 0.09 & 0.07\\ 0.12 & 0.08 & 0.06\end{bmatrix}$$

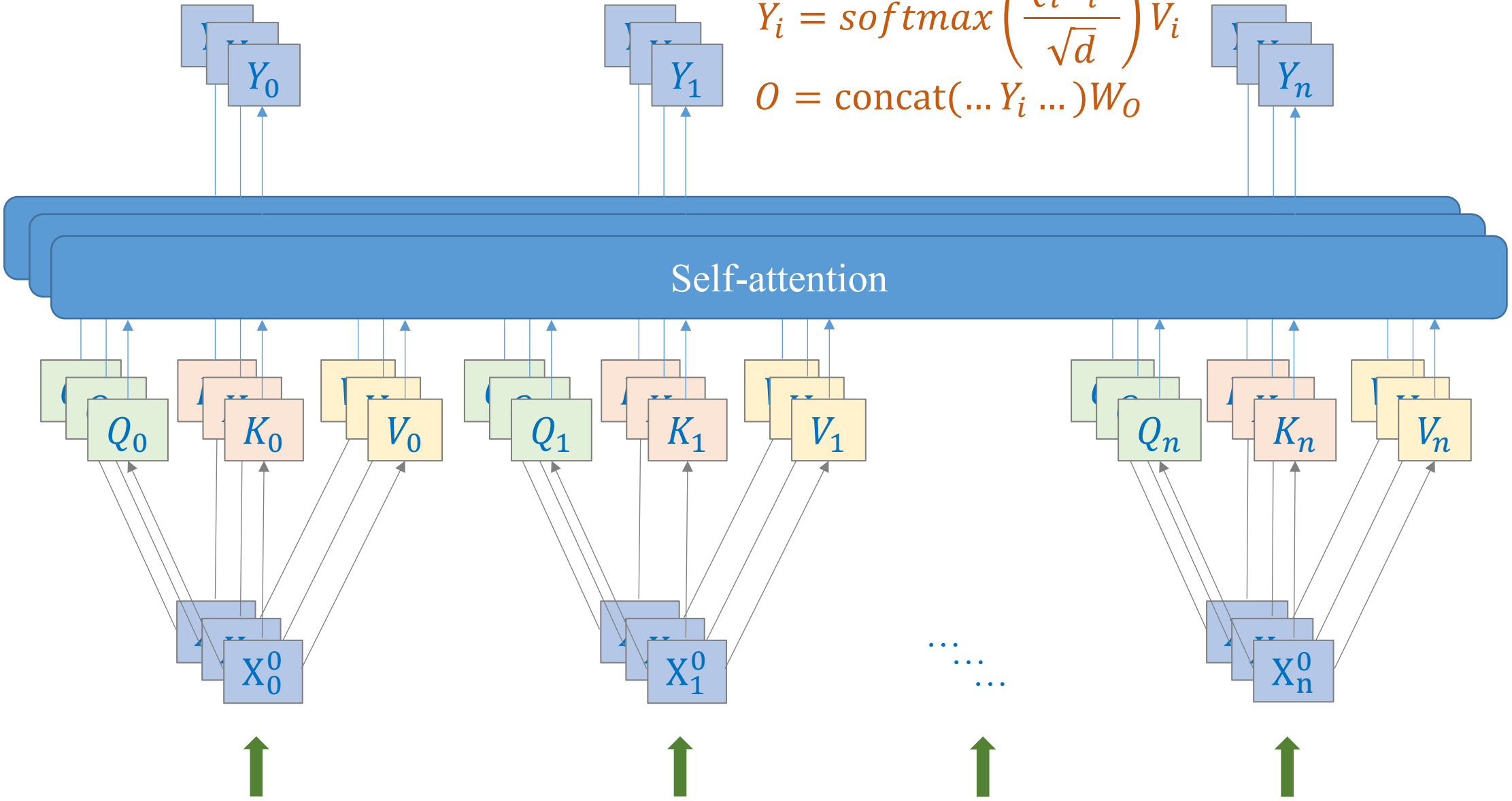
$$Y = AW_O = \begin{bmatrix}0.14 & 0.09 & 0.07\\ 0.12 & 0.08 & 0.06\end{bmatrix}\begin{bmatrix}-0.36 & -0.08 & 0.32\\ 0.27 & 0.05 & 0.15\\ -0.05 & -0.28 & 0.05\end{bmatrix} = \begin{bmatrix}-0.029 & -0.028 & 0.065\\ -0.025 & -0.025 & 0.058\end{bmatrix}$$


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

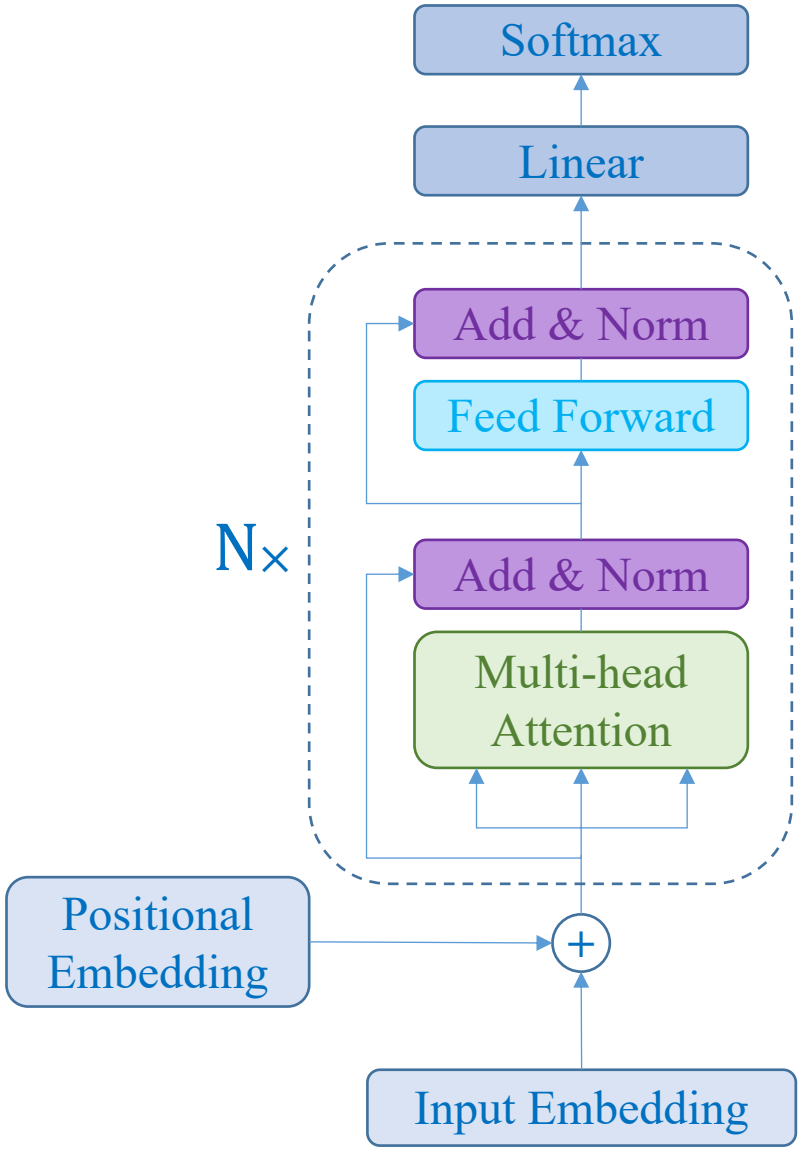
$W_O = [\theta_0 \quad \theta_1 \quad \dots \quad \theta_o] \in \mathcal{R}^{h*m\times o}$

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

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

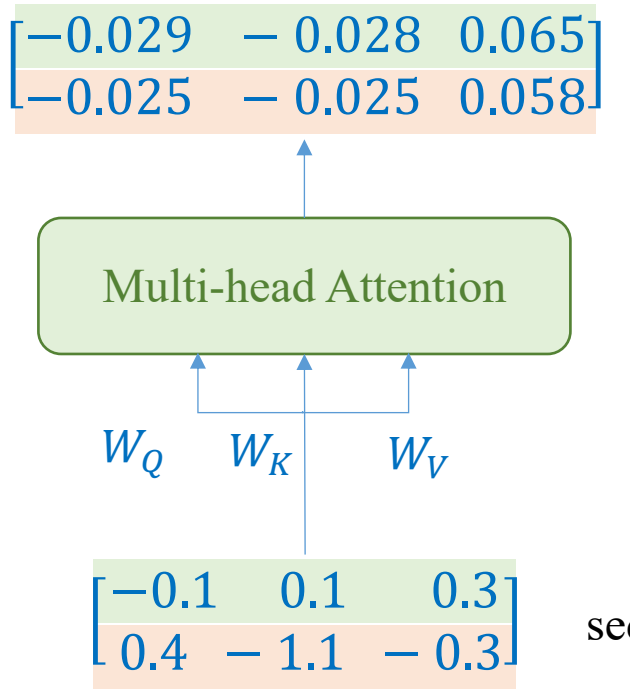


Transformer Model




```python
class TransformerTextCls(nn.Module):
    def __init__(self, vocab_size,
                 max_length, embed_dim,
                 num_heads, ff_dim):
        super().__init__()
        self.embd_layer = TokenAndPositionEmbedding(vocab_size,
                                                    embed_dim,
                                                    max_length)
        self.transformer_layer = TransformerBlock(embed_dim,
                                                  num_heads,
                                                  ff_dim)
        self.pooling = nn.AvgPool1d(kernel_size=max_length)
        self.fc = nn.Linear(in_features=embed_dim,
                            out_features=2)
        self.relu = nn.ReLU()

    def forward(self, x):
        output = self.embd_layer(x)
        output = self.transformer_layer(output, output, output)
        output = self.pooling(output.permute(0,2,1)).squeeze()
        output = self.fc(output)
        return output
```

Transformer Block



bs = 1
sequence_length = 2
embed_dim = 3

bs = 1
sequence_length = 2
embed_dim = 3

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

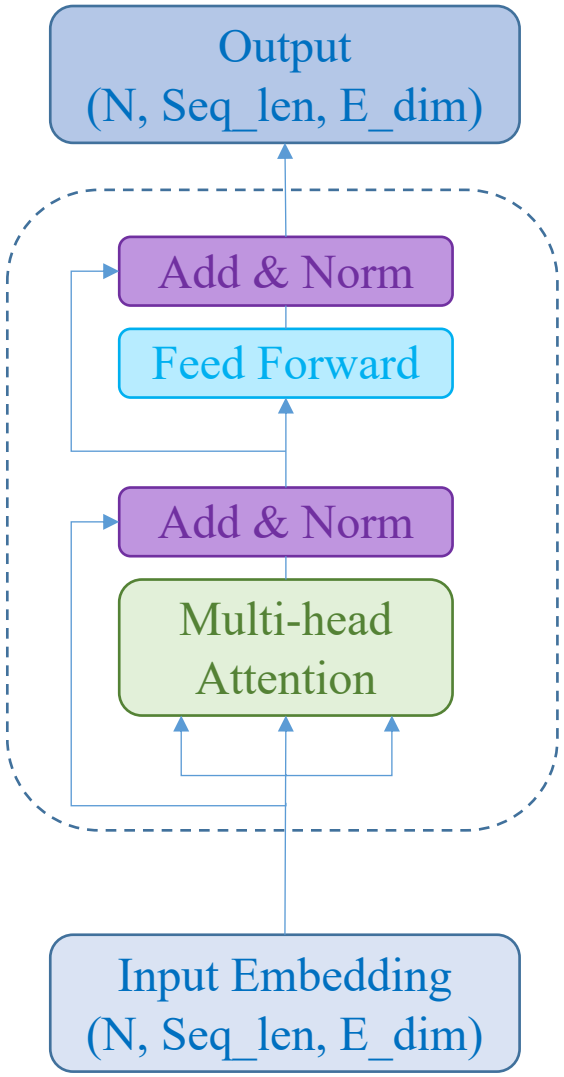
$$W_K = \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix}$$

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

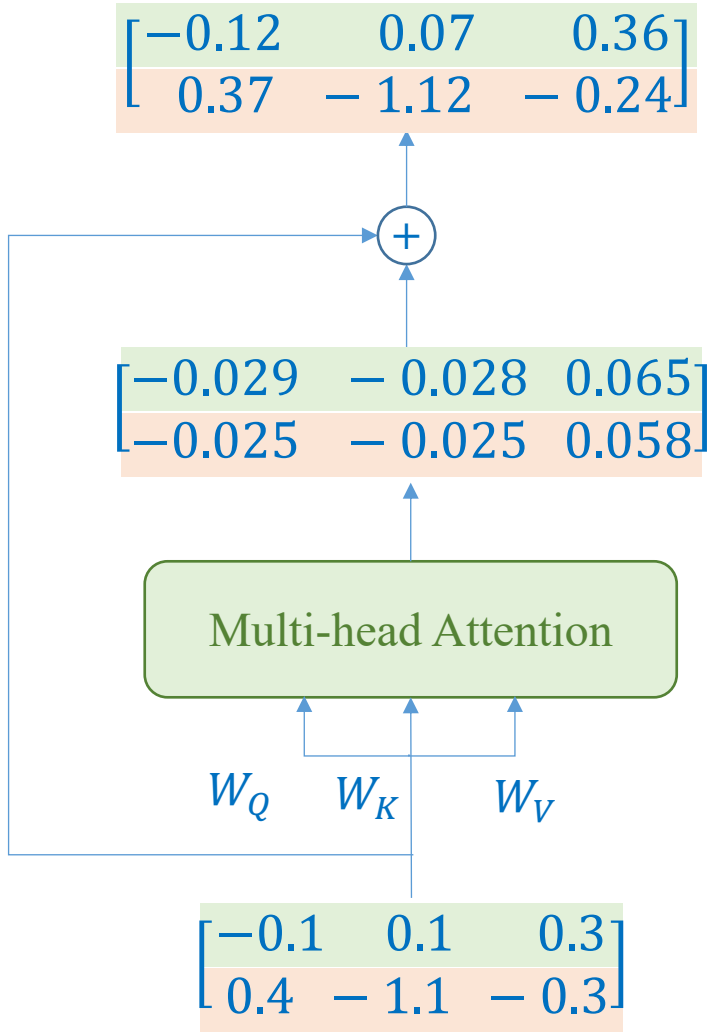
$$W_O = \begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix}$$

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

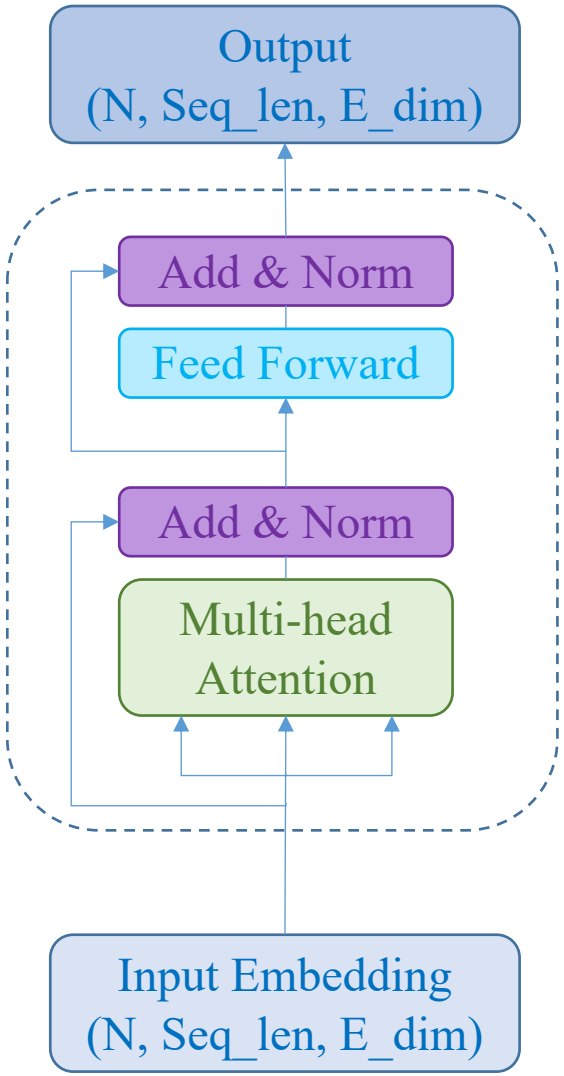
Transformer Block

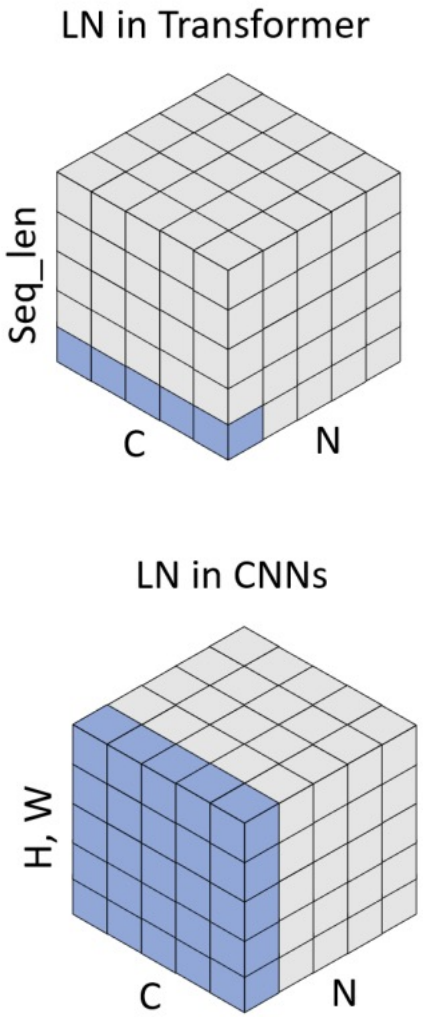


$$Y = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

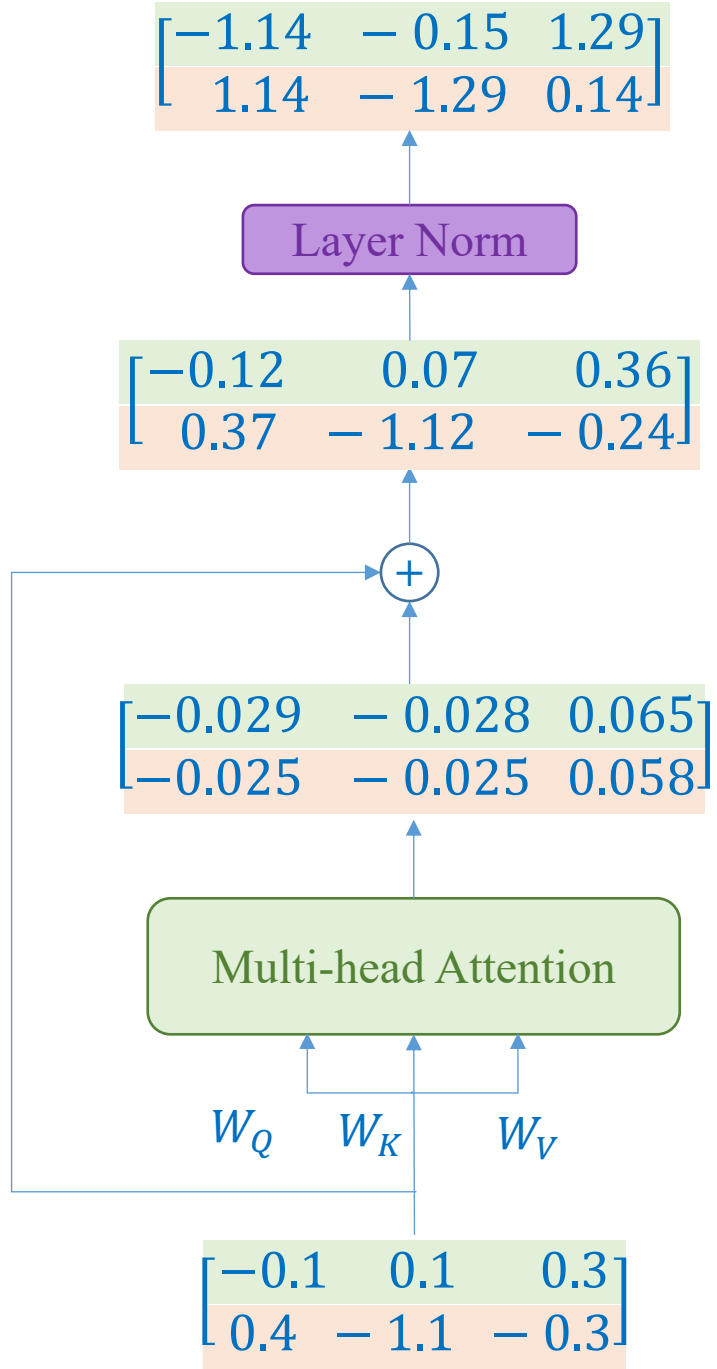


Transformer Block

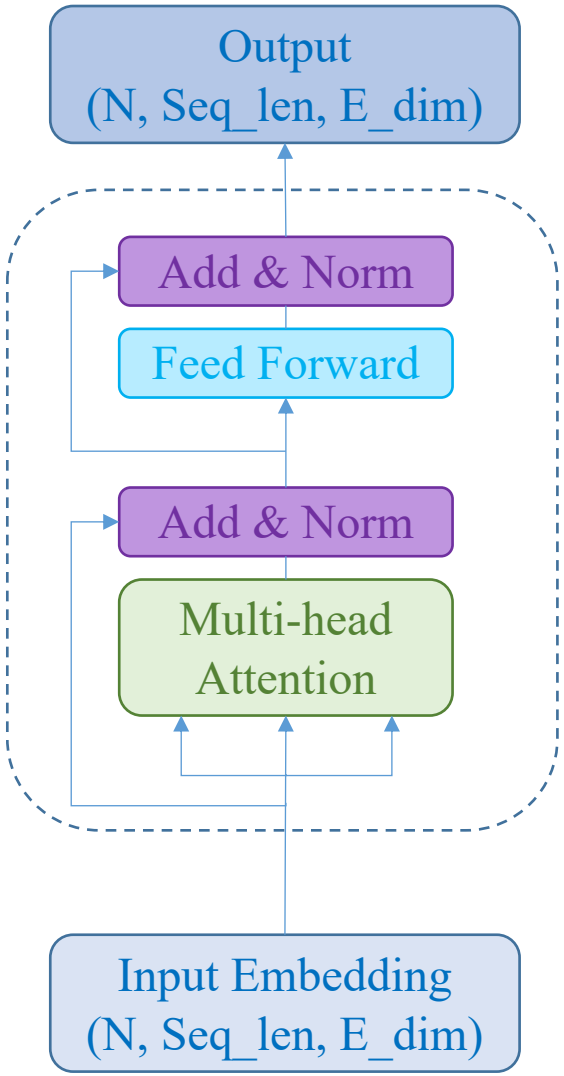




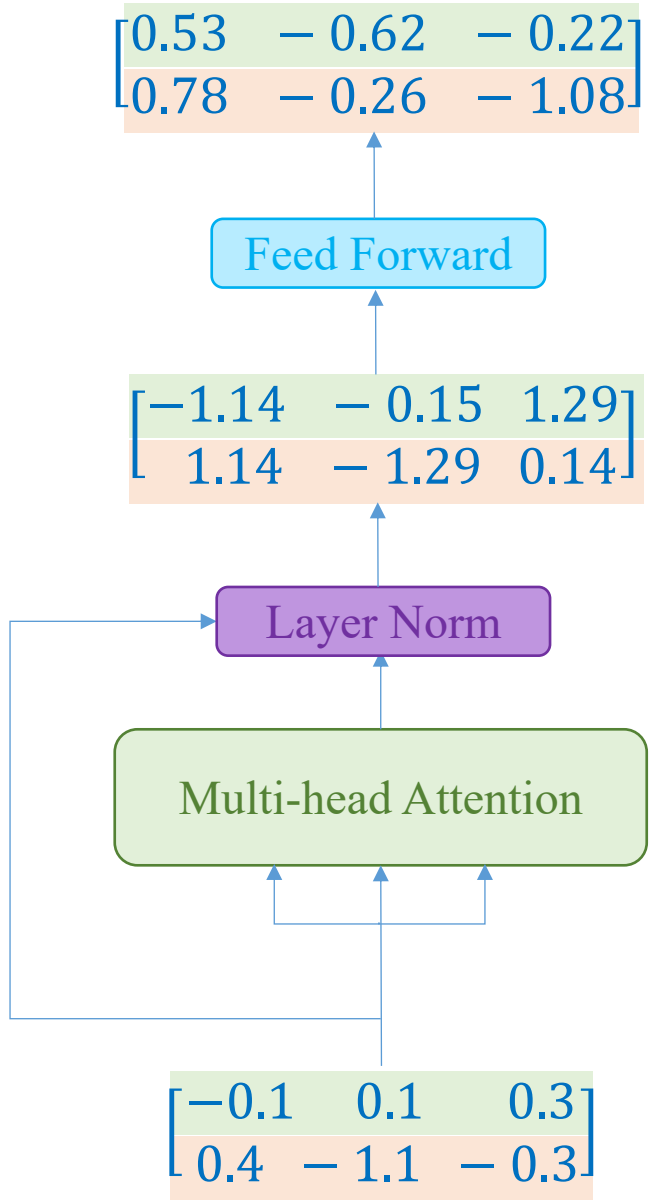
https://openaccess.thecvf.com/content/ICCV2021W/NeurArch/papers/Yao_Leveraging_Batch_Normalization_for_Vision_Transformers_ICCVW_2021_paper.pdf



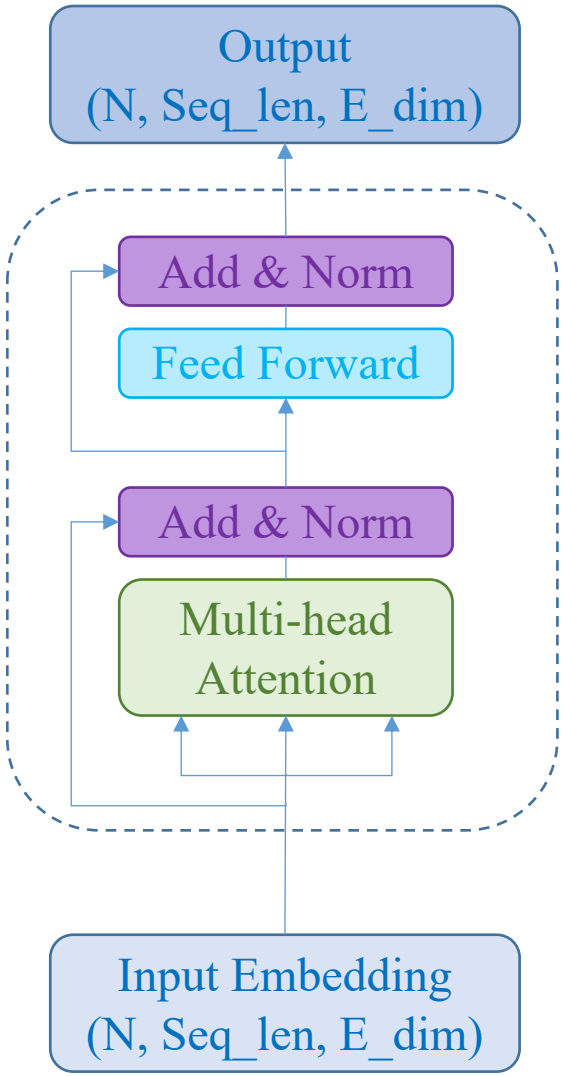
Transformer Block



$$W = \begin{bmatrix} 0.15 & 0.39 & -0.34 \\ -0.41 & 0.54 & 0.49 \\ 0.50 & -0.07 & -0.41 \end{bmatrix}$$



Transformer Block



bs = 1
sequence_length = 2
embed_dim = 3

$$\begin{bmatrix} -0.59 & -0.81 & 1.40 \\ 1.39 & -0.90 & -0.49 \end{bmatrix}$$

Layer Norm

Feed Forward

$$\begin{bmatrix} -1.14 & -0.15 & 1.29 \\ 1.14 & -1.29 & 0.14 \end{bmatrix}$$

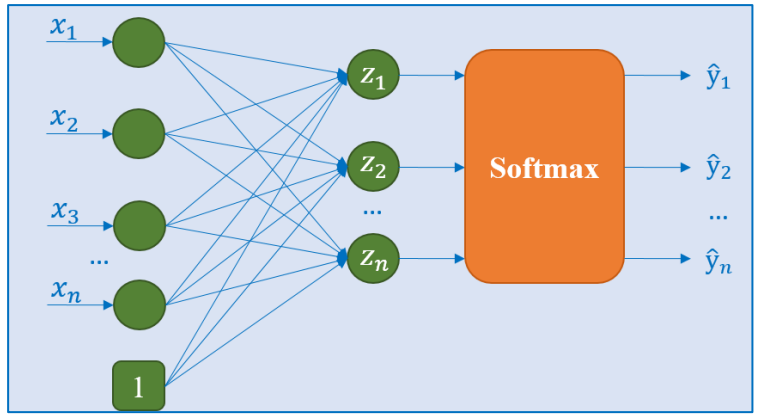
Layer Norm

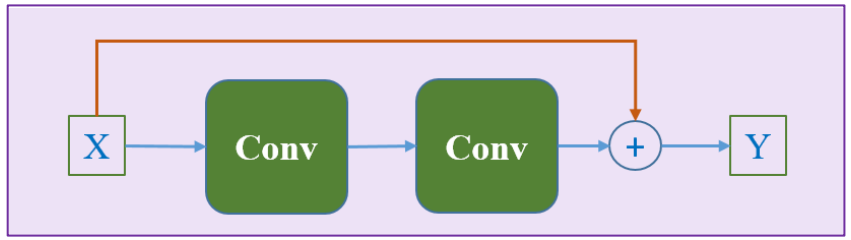
Multi-head Attention

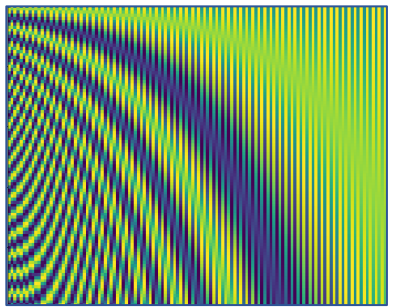
bs = 1
sequence_length = 2
embed_dim = 3

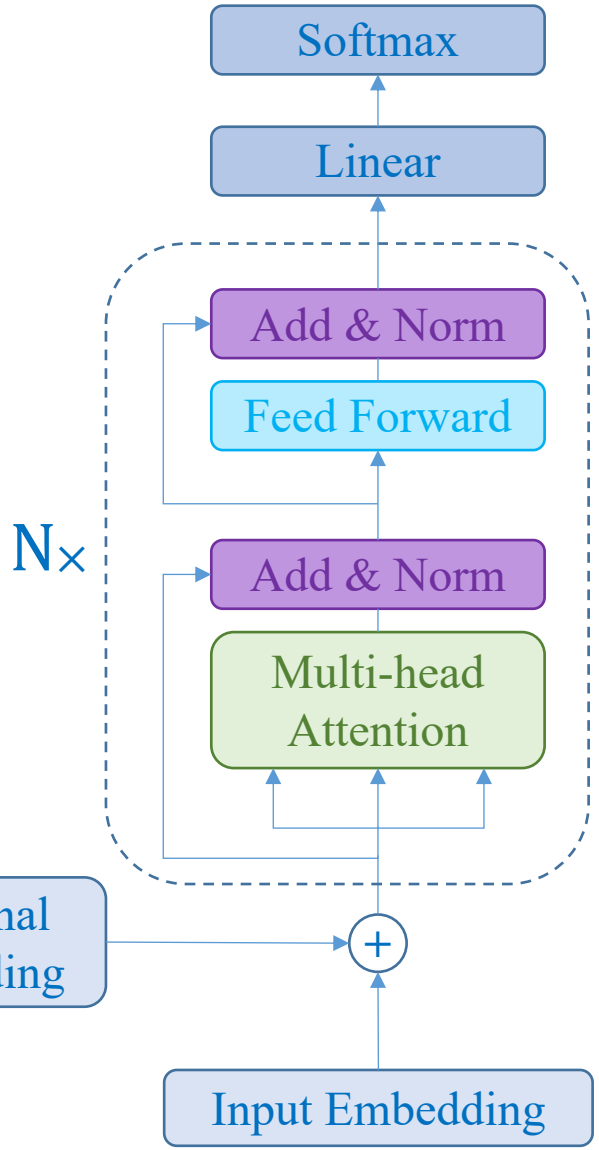
$$\begin{bmatrix} -0.1 & 0.1 & 0.3 \\ 0.4 & -1.1 & -0.3 \end{bmatrix}$$

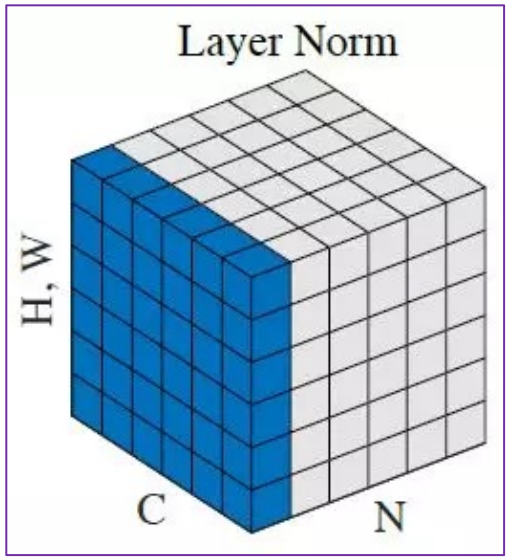
Transformer Models for Text Classification



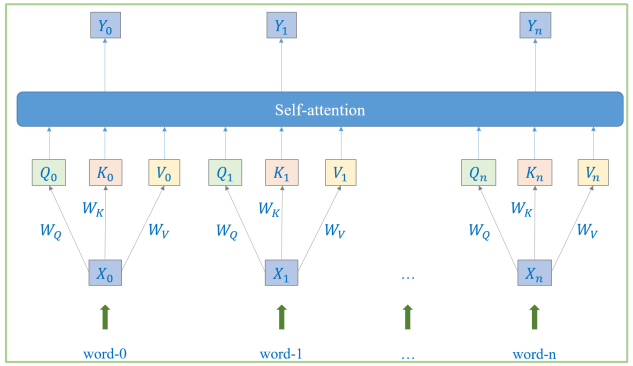


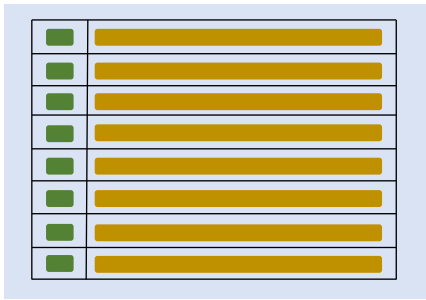






https://arxiv.org/pdf/1803.08494.pdf





Outline

SECTION 1

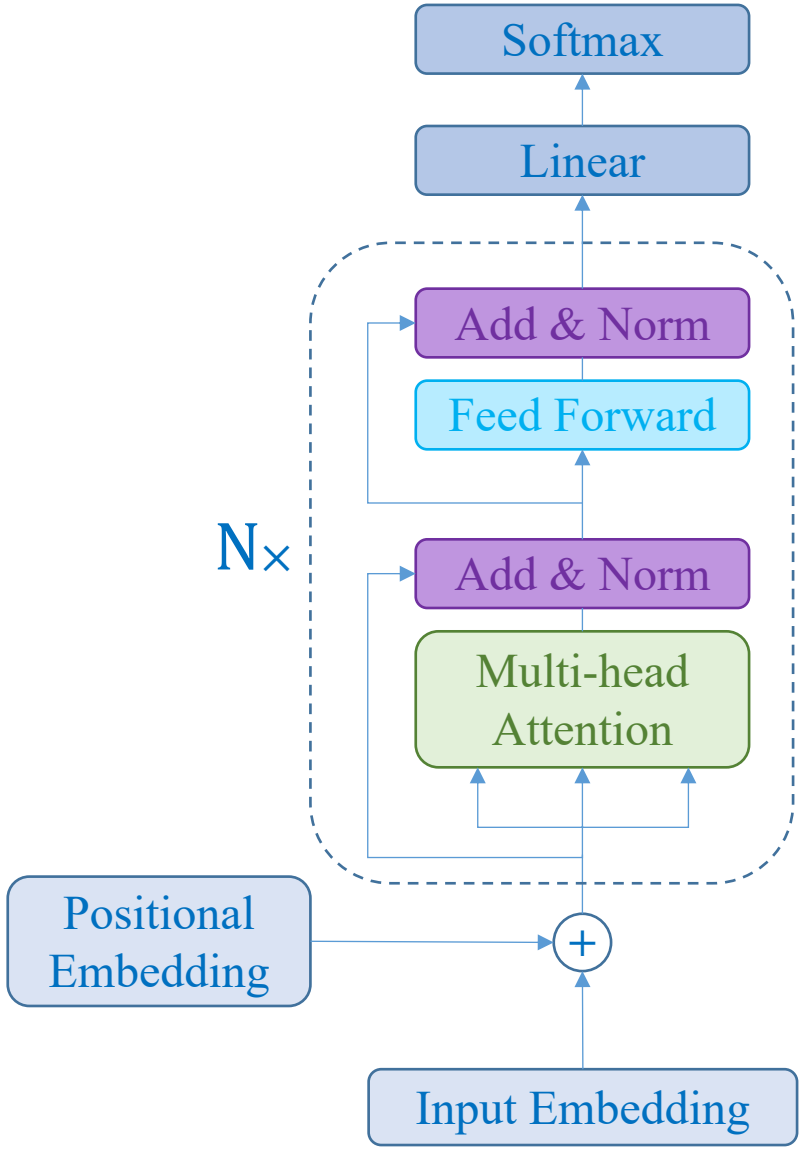
Text Classification

SECTION 2

RNN → Transformer

SECTION 3

Application



Text Classification

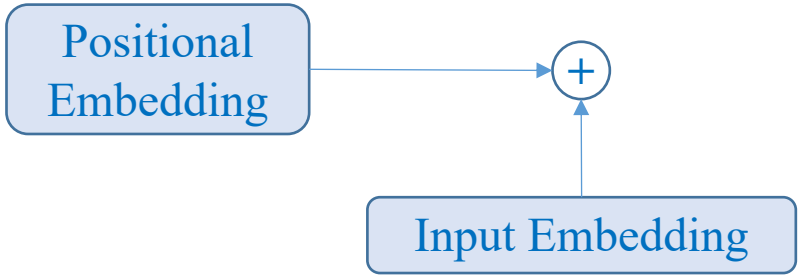
❖ IMDB dataset

- 50,000 movie review for sentiment analysis (data)
- Consist of: + 25,000 movie review for training
  + 25,000 movie review for testing
- Label: positive – negative = 1 – 1

"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

Transformer Models for Text Classification

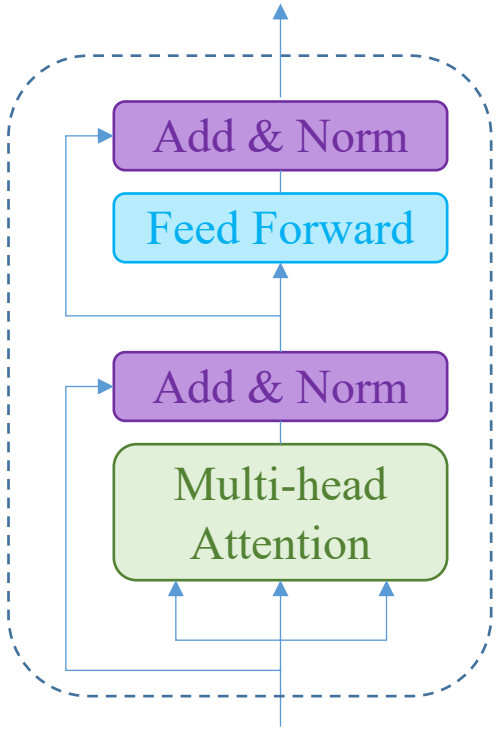
❖ Embedding




```
class TokenAndPositionEmbedding(nn.Module):
    def __init__(self, vocab_size, embed_dim, max_length):
        super().__init__()
        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)
        output1 = self.word_emb(x)
        output2 = self.pos_emb(positions)
        output = output1 + output2
        return output
```

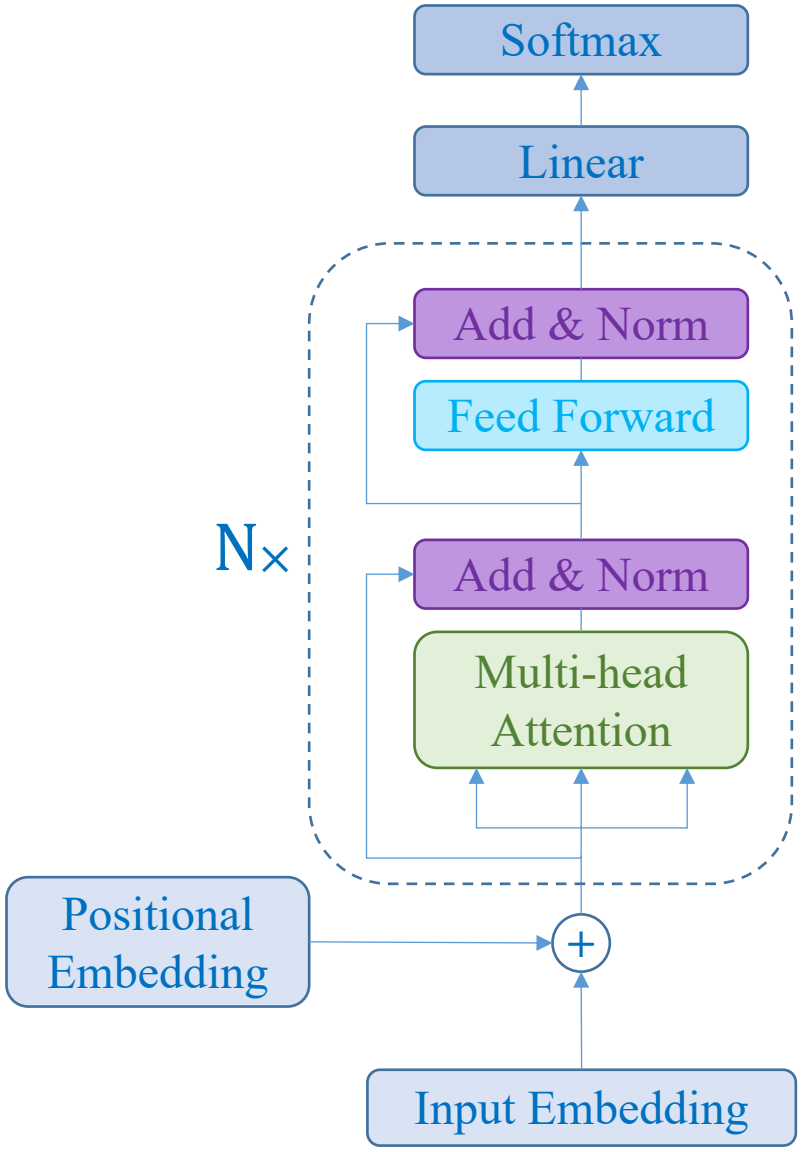
Transformer Models for Text Classification

❖ Transformer block




```python
class TransformerBlock(nn.Module):
    def __init__(self, embed_dim, num_heads, ff_dim, dropout):
        super().__init__()
        self.attn = nn.MultiheadAttention(embed_dim=embed_dim,
                                          num_heads=num_heads)
        self.ffn = nn.Linear(in_features=embed_dim,
                             out_features=ff_dim)
        self.layernorm_1 = nn.LayerNorm(normalized_shape=embed_dim)
        self.layernorm_2 = nn.LayerNorm(normalized_shape=embed_dim)

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

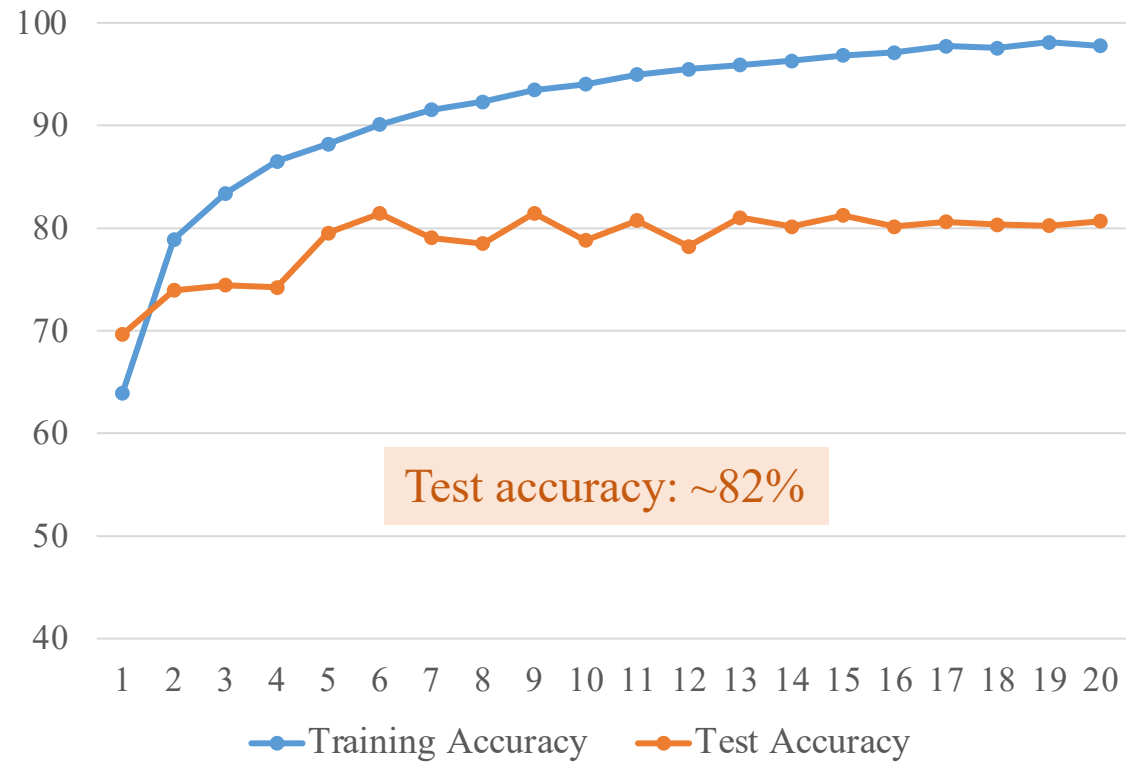
Transformer Models for Text Classification




```python
class TransformerTextCls(nn.Module):
    def __init__(self, vocab_size,
                 max_length, embed_dim,
                 num_heads, ff_dim,
                 dropout, device):
        super().__init__()
        self.embd_layer = TokenAndPositionEmbedding(vocab_size,
                                                    embed_dim,
                                                    max_length)
        self.transformer_layer = TransformerBlock(embed_dim,
                                                  num_heads,
                                                  ff_dim)
        self.pooling = nn.AvgPool1d(kernel_size=max_length)
        self.fc = nn.Linear(in_features=embed_dim,
                            out_features=2)
        self.relu = nn.ReLU()

    def forward(self, x):
        output = self.embd_layer(x)
        output = self.transformer_layer(output, output, output)
        output = self.pooling(output.permute(0,2,1)).squeeze()
        output = self.fc(output)

        return output
```

Transformer Models for Text Classification

❖ Results



Train Transformer from Scratch

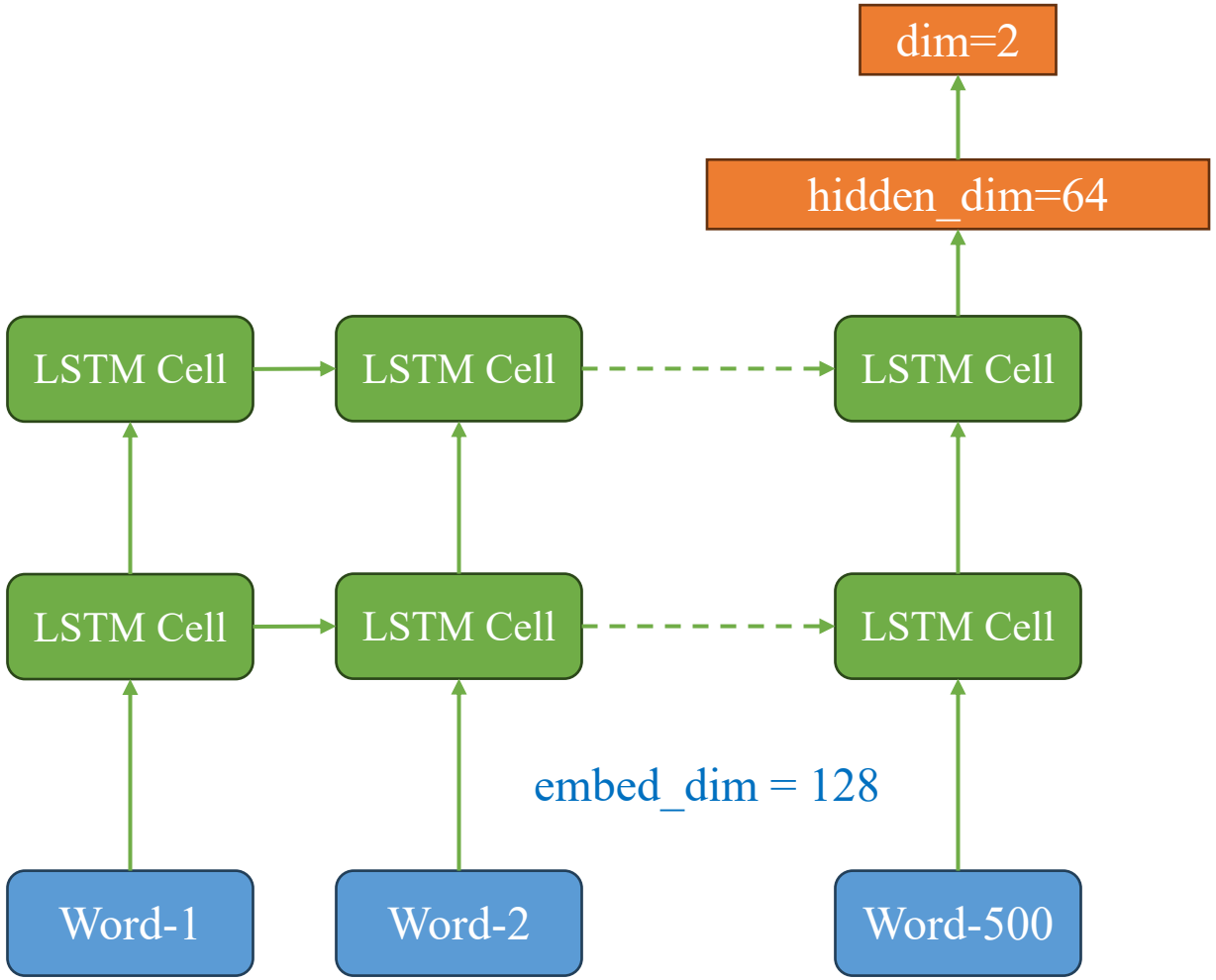
Epoch	Training Loss	Validation Loss	Accuracy
1	0.282000	0.216251	0.915120
2	0.149400	0.200550	0.929160
3	0.102600	0.250929	0.924840
4	0.055700	0.321075	0.930280
5	0.038100	0.344045	0.928200
6	0.021000	0.375231	0.928480
7	0.020200	0.384939	0.928680
8	0.010400	0.413715	0.930320
9	0.009400	0.428601	0.930840
	0.005800	0.441084	0.930840

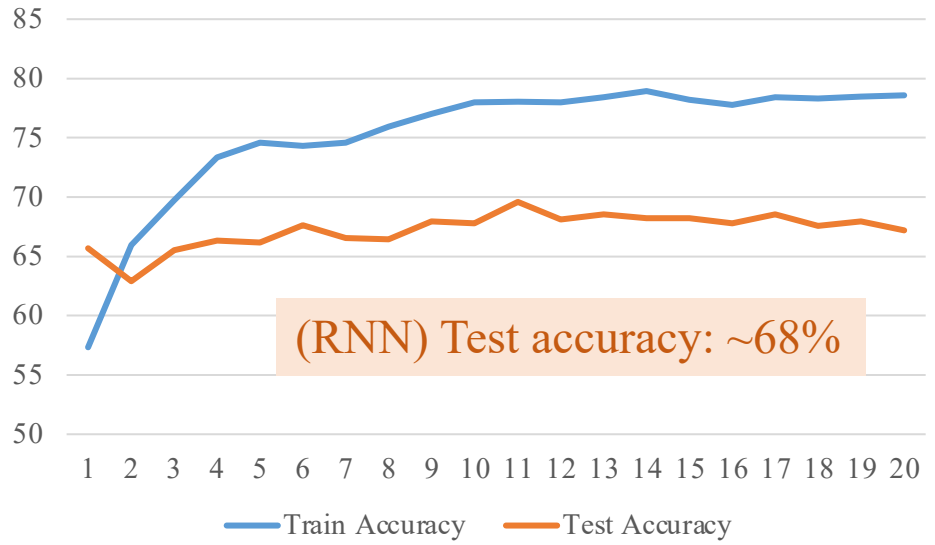
transfer learning

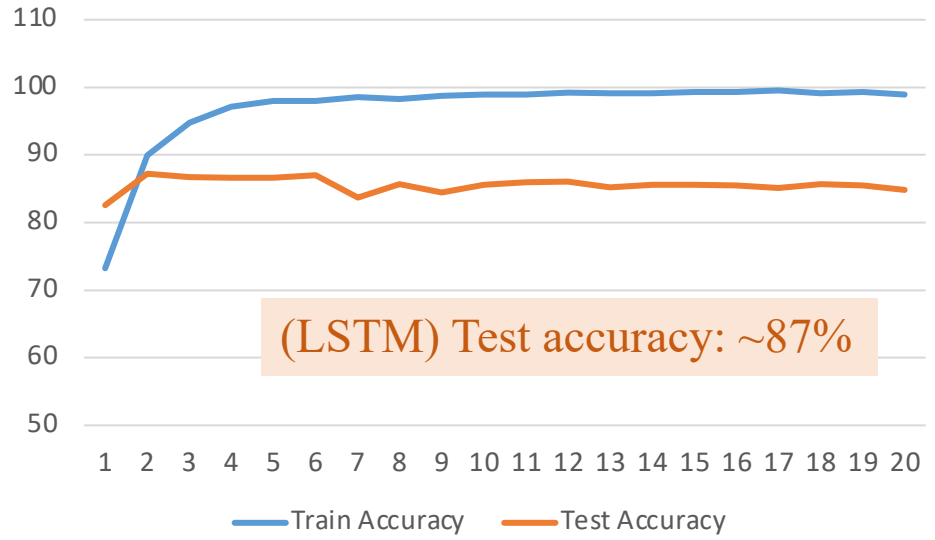
```
{'eval_loss': 0.20054994523525238,
 'eval_accuracy': 0.92916,
 'eval_runtime': 253.348,
 'eval_samples_per_second': 98.678,
 'eval_steps_per_second': 3.087,
 'epoch': 10.0}
```

Text Deep Models

Long short-term memory







Text Classification

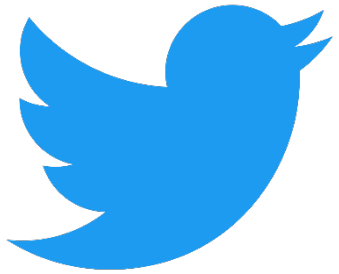
❖ Tweets dataset

- Training samples: 7613
- Total Number of disaster tweets: 4342
- Total Number of non-disaster tweets: 3271

id	keyword	location	text	target
48	ablaze	Birmingham	@bbcmtd Wholesale Markets ablaze http://t.co/lH	1
49	ablaze	Est. September 2012 - Bri	We always try to bring the heavy. #metal #RT htt	0
144	accident	UK	.@NorwayMFA #Bahrain police had previously c	1
145	accident	Nairobi, Kenya	I still have not heard Church Leaders of Kenya co	0
146	aftershock	Instagram - @heyimginog	@afterShock_DeLo scuf ps live and the game... c	0
1669	bombing	London	Japan marks 70th anniversary of Hiroshima atomi	1
1670	bombing	United States	Japan marks 70th anniversary of Hiroshima atomi	1
6536	injury	beijing .China	Rory McIlroy to Test Ankle Injury in Weekend P	0
6537	injury	MISSING	CLEARED:incident with injury:I-495 inner loop	1

Text Classification

❖ Tweets dataset



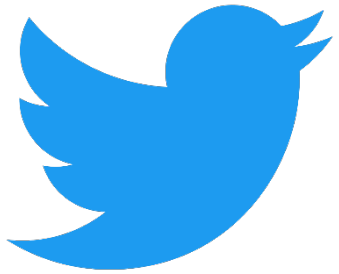
id	keyword	location	text	target
48	ablaze	Birmingham	@bbcmtd Wholesale Markets ablaze http://t.co/lH	1
49	ablaze	Est. September 2012 - Bri	We always try to bring the heavy. #metal #RT htt	0
144	accident	UK	.@NorwayMFA #Bahrain police had previously	1
145	accident	Nairobi, Kenya	I still have not heard Church Leaders of Kenya co	0
146	aftershock	Instagram - @heyimginog	@afterShock_DeLo scuf ps live and the game... c	0
1669	bombing	London	Japan marks 70th anniversary of Hiroshima atomi	1
1670	bombing	United States	Japan marks 70th anniversary of Hiroshima atomi	1
6536	injury	beijing .China	Rory McIlroy to Test Ankle Injury in Weekend P	0
6537	injury	MISSING	CLEARED:incident with injury:I-495 inner loop	1

```python
import pandas as pd
import numpy as np

df = pd.read_csv('train.csv')
df_shuffled = df.sample(frac=1,
                        random_state=42)

df_shuffled.drop(["id", "keyword", "location"],
                 axis=1, inplace=True)
df_shuffled.reset_index(inplace=True,
                        drop=True)
```

text	target
@bbcmtd Wholesale Markets ablaze http://t.co/lH	1
We always try to bring the heavy. #metal #RT htt	0
.@NorwayMFA #Bahrain police had previously	1
I still have not heard Church Leaders of Kenya co	0
@afterShock_DeLo scuf ps live and the game... c	0
Japan marks 70th anniversary of Hiroshima atomi	1
Japan marks 70th anniversary of Hiroshima atomi	1
Rory McIlroy to Test Ankle Injury in Weekend P	0
CLEARED:incident with injury:I-495 inner loop	1

Text Classification

❖ Tweets dataset

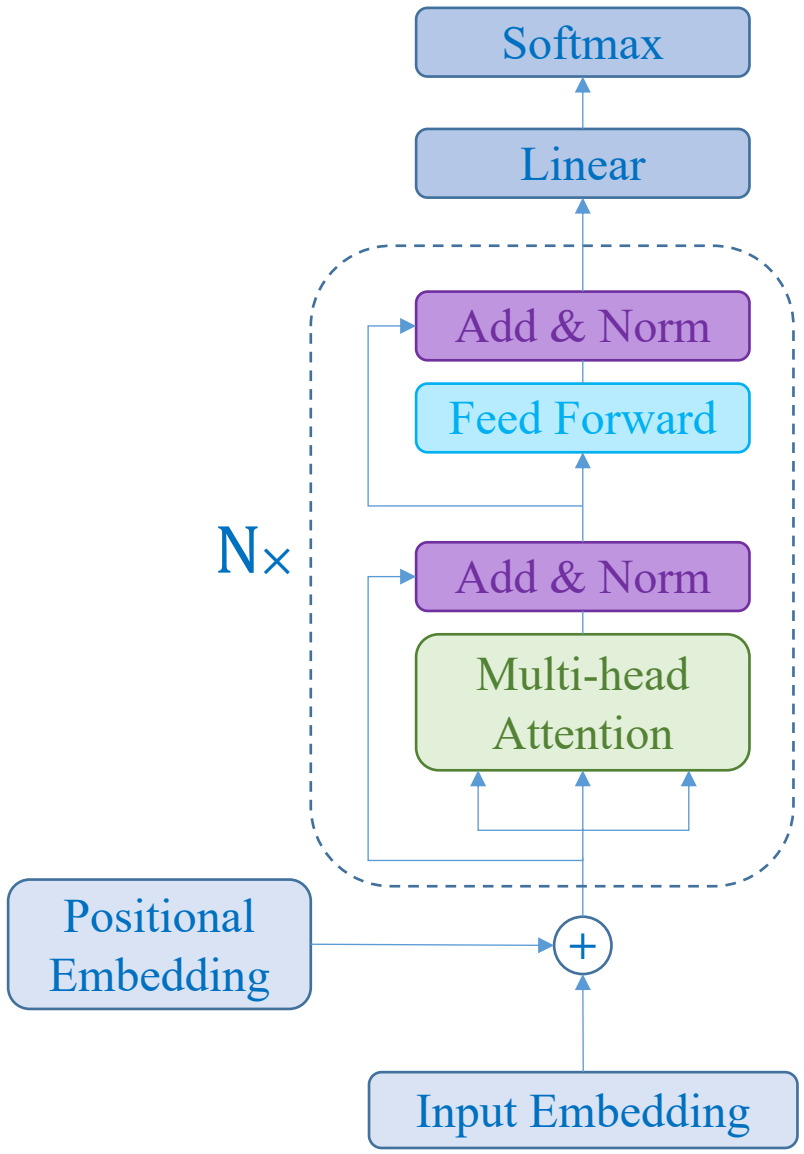


id	keyword	location	text	target
48	ablaze	Birmingham	@bbcmtd Wholesale Markets ablaze http://t.co/lH	1
49	ablaze	Est. September 2012 - Bri	We always try to bring the heavy. #metal #RT htt	0
144	accident	UK	.@NorwayMFA #Bahrain police had previously	1
145	accident	Nairobi, Kenya	I still have not heard Church Leaders of Kenya co	0
146	aftershock	Instagram - @heyimginog	@afterShock_DeLo scuf ps live and the game... c	0
1669	bombing	London	Japan marks 70th anniversary of Hiroshima atomi	1
1670	bombing	United States	Japan marks 70th anniversary of Hiroshima atomi	1
6536	injury	beijing .China	Rory McIlroy to Test Ankle Injury in Weekend P	0
6537	injury	MISSING	CLEARED:incident with injury:I-495 inner loop	1

```
test_df = df_shuffled.sample(frac=0.1, random_state=42)
train_df = df_shuffled.drop(test_df.index)

train_df = train_df.to_numpy()
test_df  = test_df.to_numpy()

#...

train_loader = DataLoader(train_data,
                          batch_size=32, shuffle=True)
test_loader = DataLoader(test_data,
                         batch_size=32, shuffle=False)
```

text	target
@bbcmtd Wholesale Markets ablaze http://t.co/lH	1
We always try to bring the heavy. #metal #RT htt	0
.@NorwayMFA #Bahrain police had previously	1
I still have not heard Church Leaders of Kenya co	0
@afterShock_DeLo scuf ps live and the game... c	0
Japan marks 70th anniversary of Hiroshima atomi	1
Japan marks 70th anniversary of Hiroshima atomi	1
Rory McIlroy to Test Ankle Injury in Weekend P	0
CLEARED:incident with injury:I-495 inner loop	1

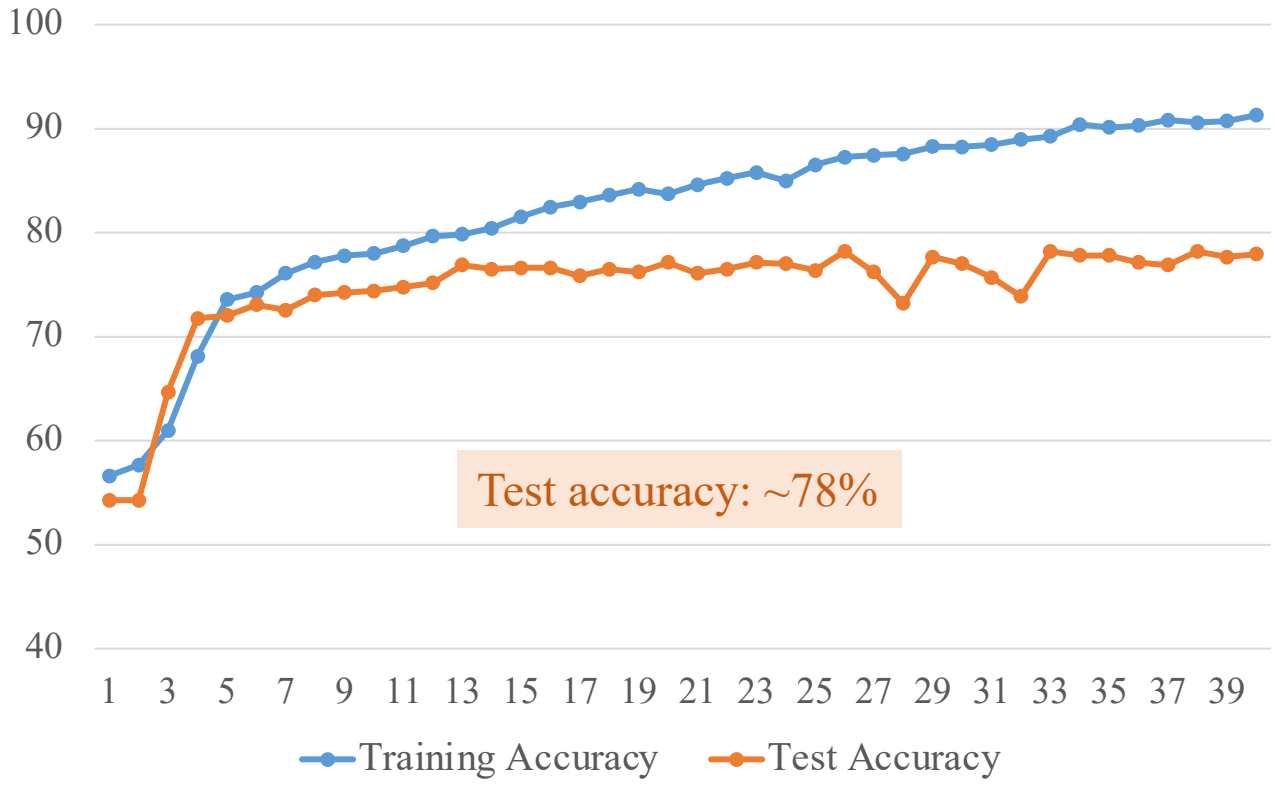
Transformer Models for Text Classification




```python
class TransformerBlock(nn.Module):
    def __init__(self, embed_dim, num_heads, ff_dim):
        # ...

    def forward(self, query, key, value):
        # ...

class TokenAndPositionEmbedding(nn.Module):
    def __init__(self, vocab_size, embed_dim, max_length):
        # ...

    def forward(self, x):
        # ...

class TransformerTextCls(nn.Module):
    def __init__(self, vocab_size, max_length,
                 embed_dim, num_heads, ff_dim):
        # ...

    def forward(self, x):
        output = self.embd_layer(x)
        output = self.transformer_layer(output, output, output)
        output = self.pooling(output.permute(0,2,1)).squeeze()
        output = self.fc(output)
        return output
```

Text Classification

❖ Results



Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.410340	0.819974
2	0.448100	0.419809	0.830486
3	0.320300	0.452259	0.822602
4	0.234100	0.545349	0.818660
5	0.176700	0.664652	0.809461
6	0.138900	0.859782	0.814717
14	0.035800	1.288030	0.808147
15	0.035800	1.281955	0.825230
16	0.029600	1.302865	0.819974
17	0.025800	1.324648	0.816032
18	0.022500	1.339616	0.819974
19	0.023800	1.363005	0.818660
20	0.014200	1.395317	0.817346

```
{'eval_loss': 0.41033995151519775,
 'eval_accuracy': 0.8199737187910644,
 'eval_runtime': 2.696,
 'eval_samples_per_second': 282.267,
 'eval_steps_per_second': 17.804,
 'epoch': 20.0}
```

Train Transformer from Scratch