AI VIETNAM
All-in-One Course

Text Generation

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2024

Outline

- Motivation to Text Generation
- Simple Models
- Using RNNs
- Examples
- Using Masked Encoder (~Decoder)

Self-Supervision Using Image Data

Noisy images

Model

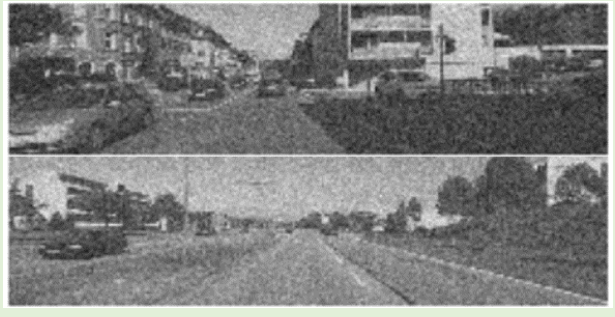
Clean images

Grayscale images

Model

Color images

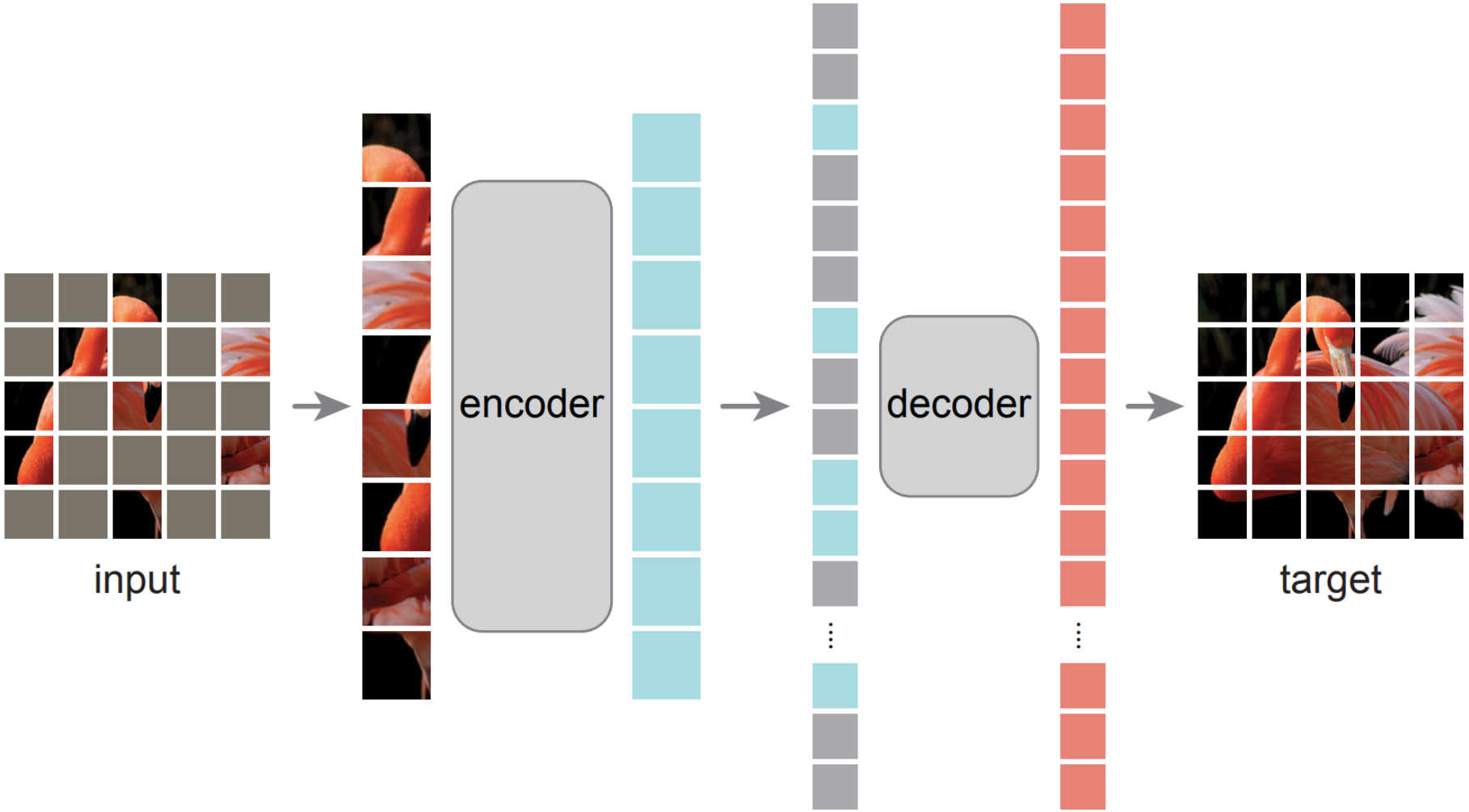
Noisy and grayscale images



Model

Clean and color images

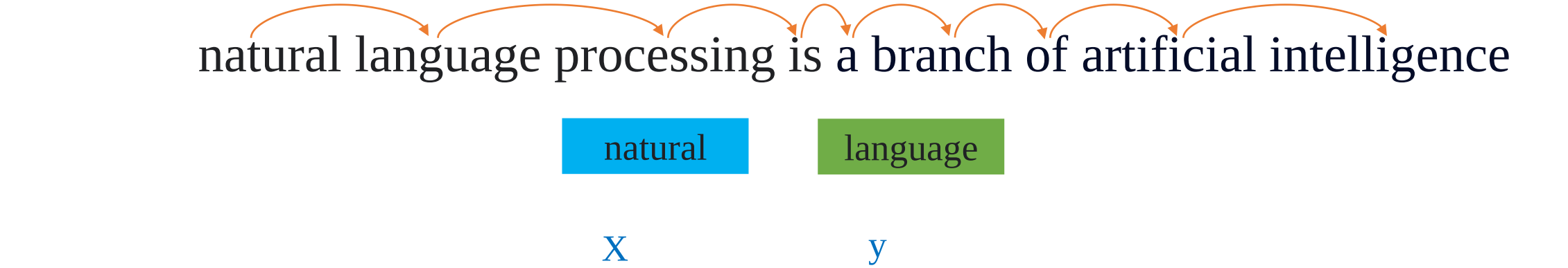
Self-Supervision Using Image Data

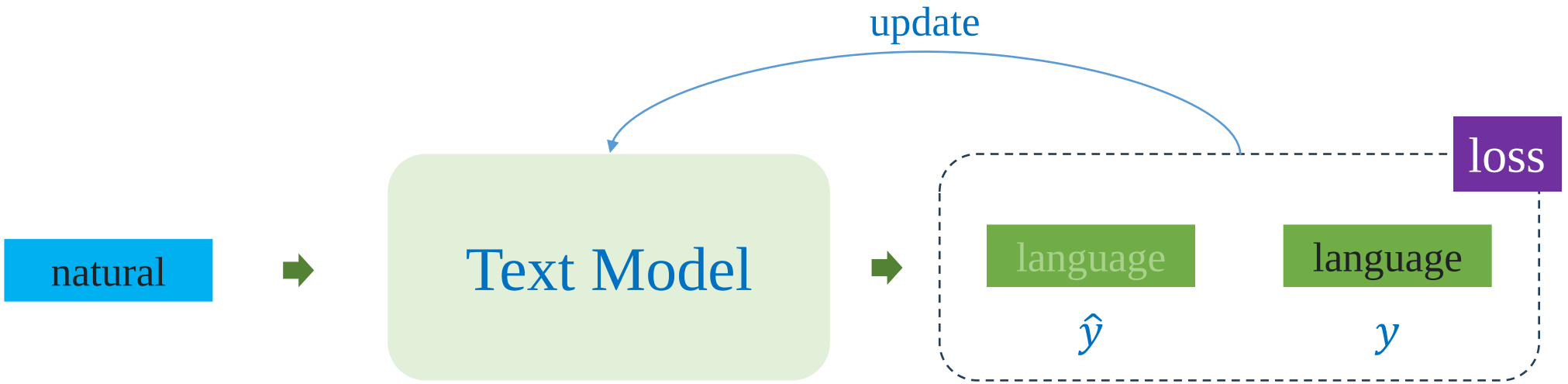


https://arxiv.org/pdf/2111.06377.pdf

Self-Supervision Using Text Data

❖ **How?**





Text Generation

Inference

❖ Applications

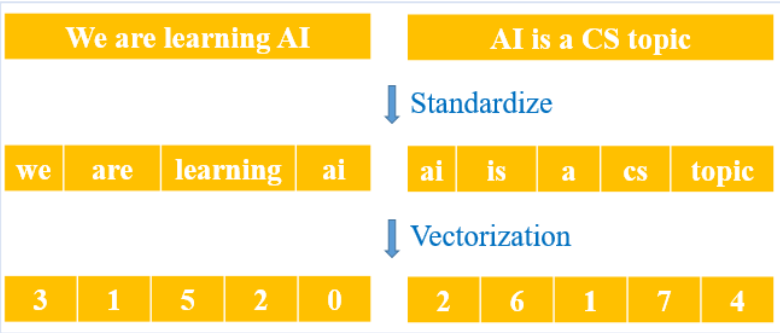
❖ From the viewpoint of users

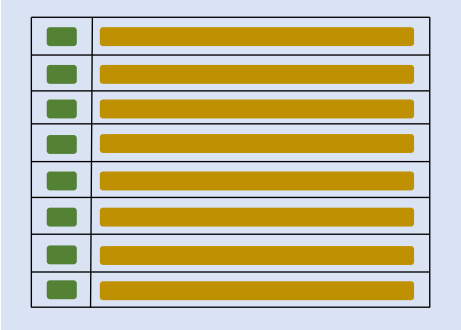
```
1 generate_text("thúy kiều", 4, generator_model)
```

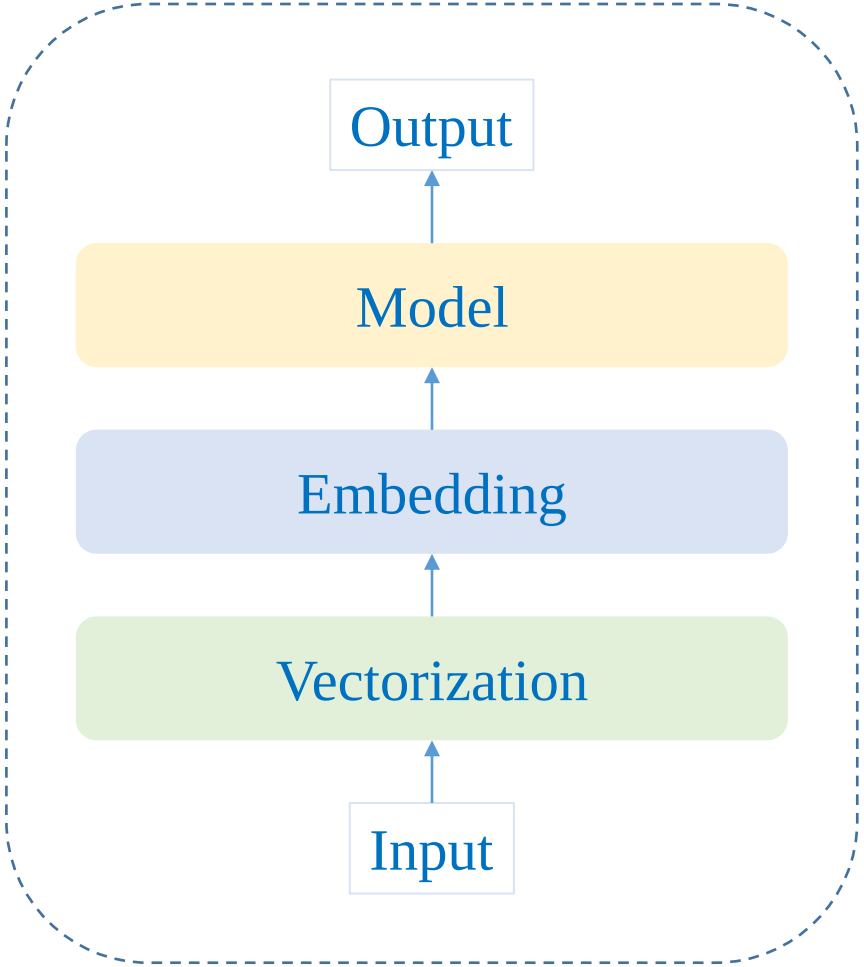
```
'thúy kiều sắc sảo khôn ngoan'
```

```
1 generate_text("kim trọng và thúy kiều", 6, generator_model)
```

```
'kim trọng và thúy kiều nhân gian sao tan tành ngay'
```

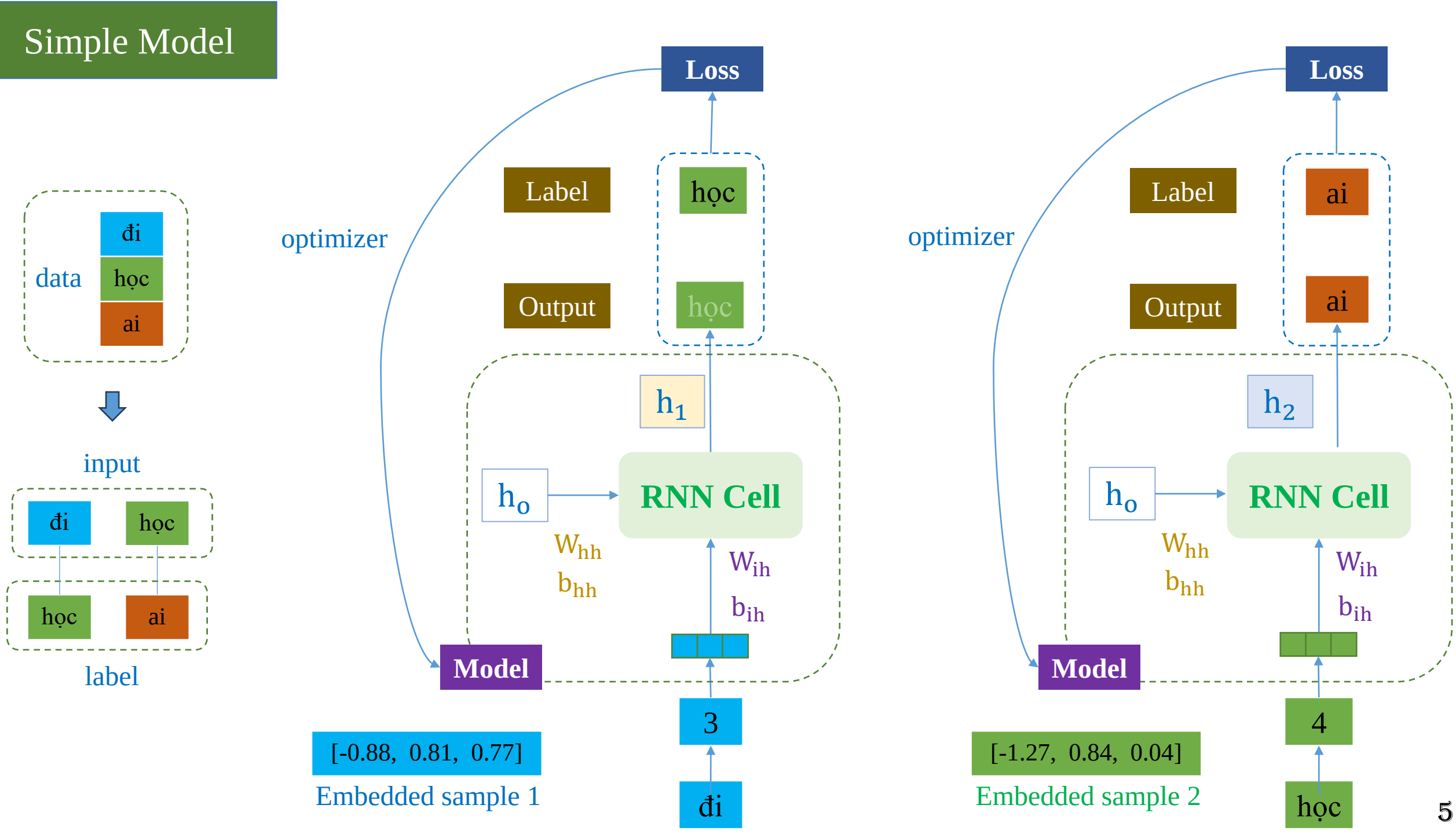




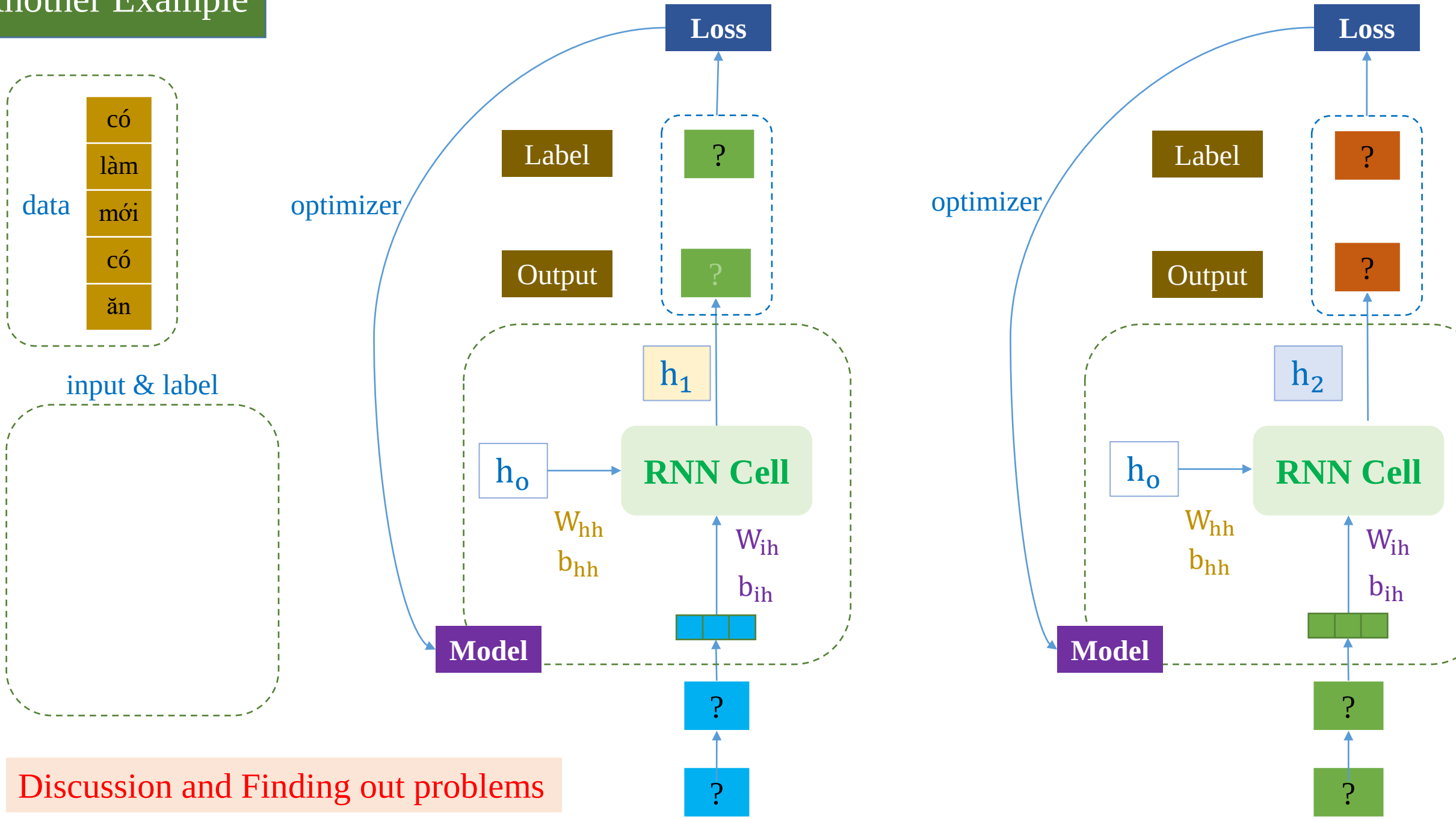


Simple Model

Simple Model



Another Example



Discussion and Finding out problems

Text Generation

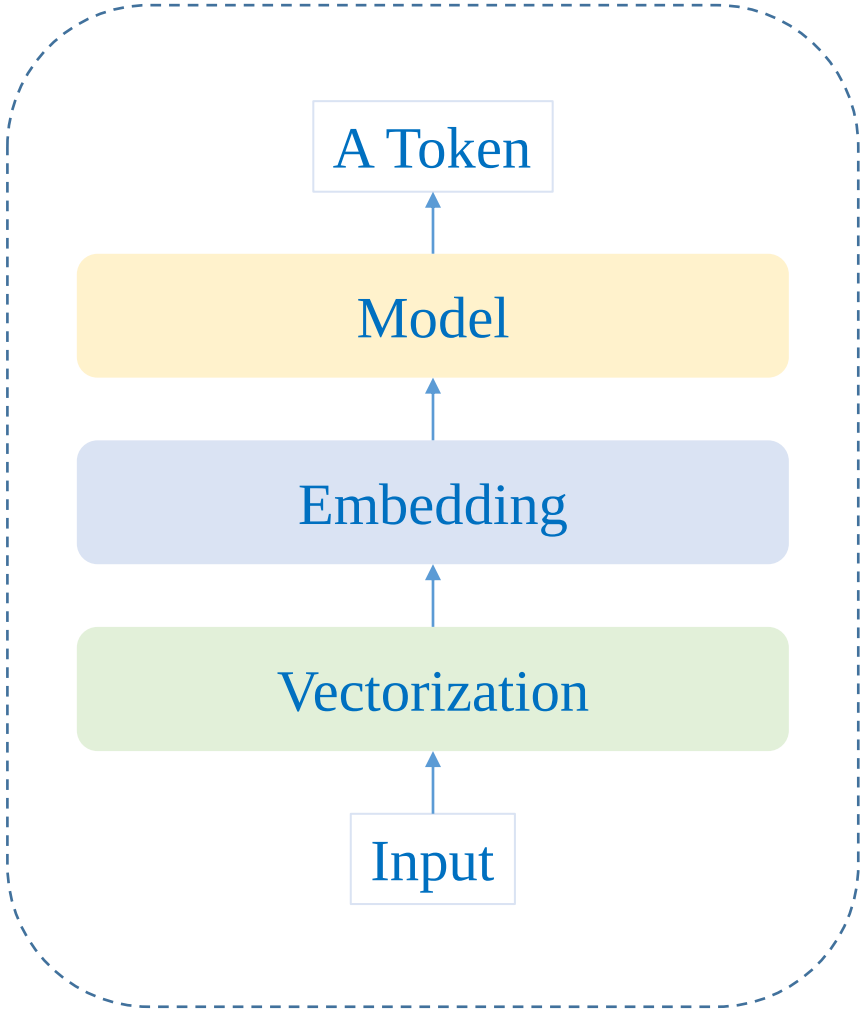
❖ Input is a set of tokens

```
1 generate_text("thúy kiều", 4, generator_model)
```

```
'thúy kiều sắc sảo khôn ngoan'
```

```
1 generate_text("kim trọng và thúy kiều", 6, generator_model)
```

```
'kim trọng và thúy kiều nhân gian sao tan tành ngay'
```



data = 'trăm năm trong cõi người ta'

```
Input data: ['trăm'] => output data: năm
Input data: ['trăm', 'năm'] => output data: trong
Input data: ['trăm', 'năm', 'trong'] => output data: cõi
Input data: ['trăm', 'năm', 'trong', 'cõi'] => output data: người
Input data: ['trăm', 'năm', 'trong', 'cõi', 'người'] => output data: ta
```

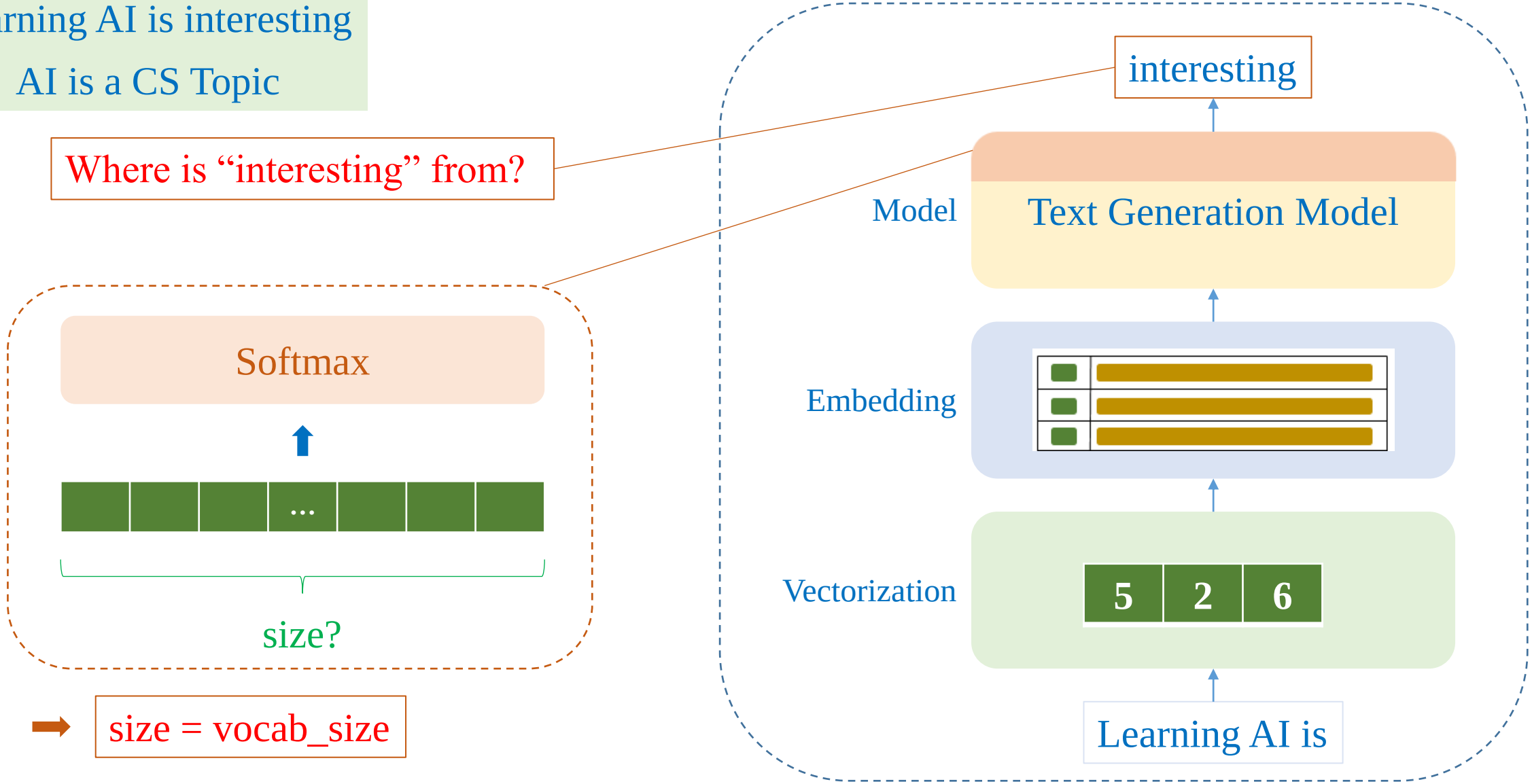
Text Generation

Another Example

Learning AI is interesting

AI is a CS Topic

Where is “interesting” from?

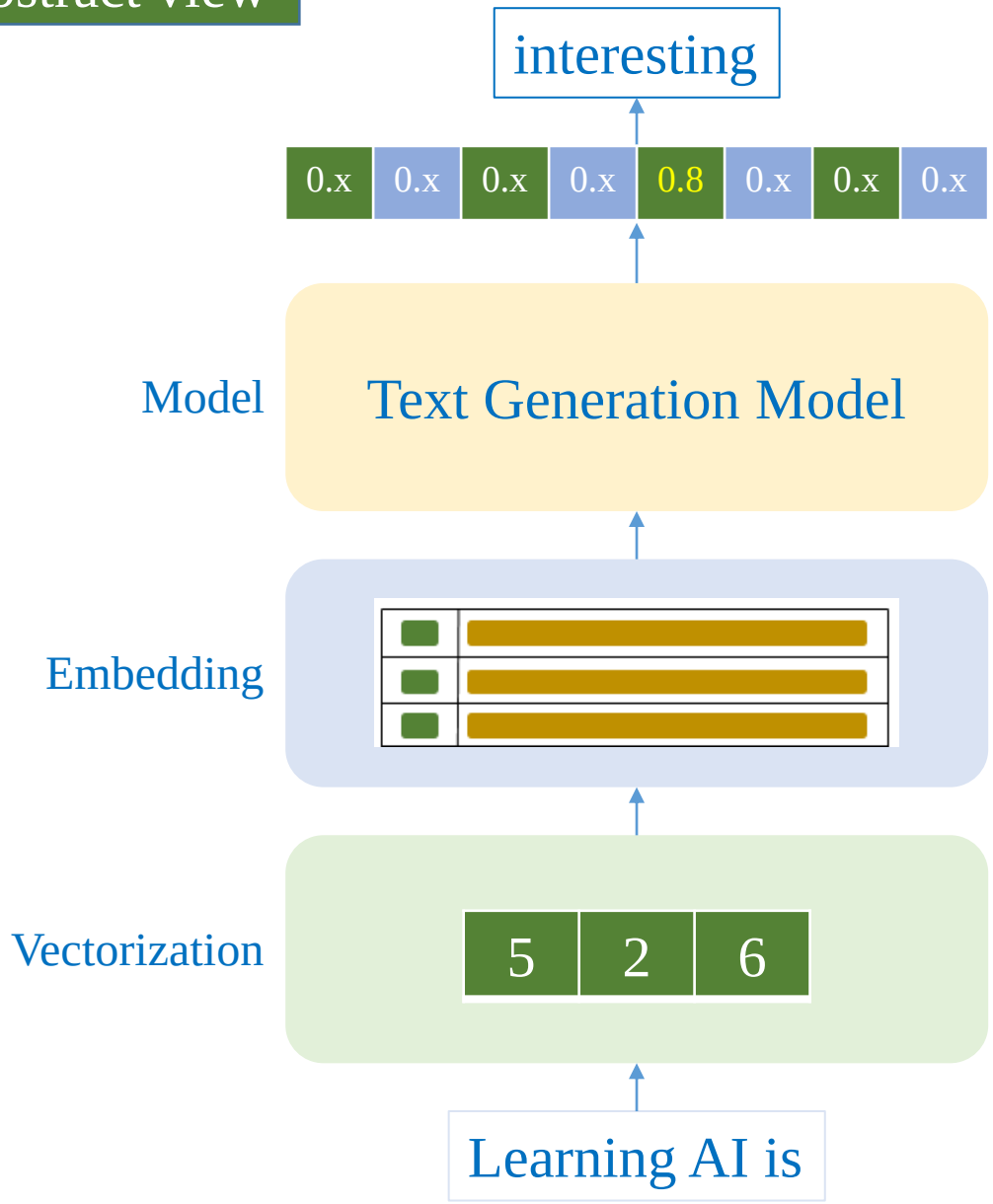


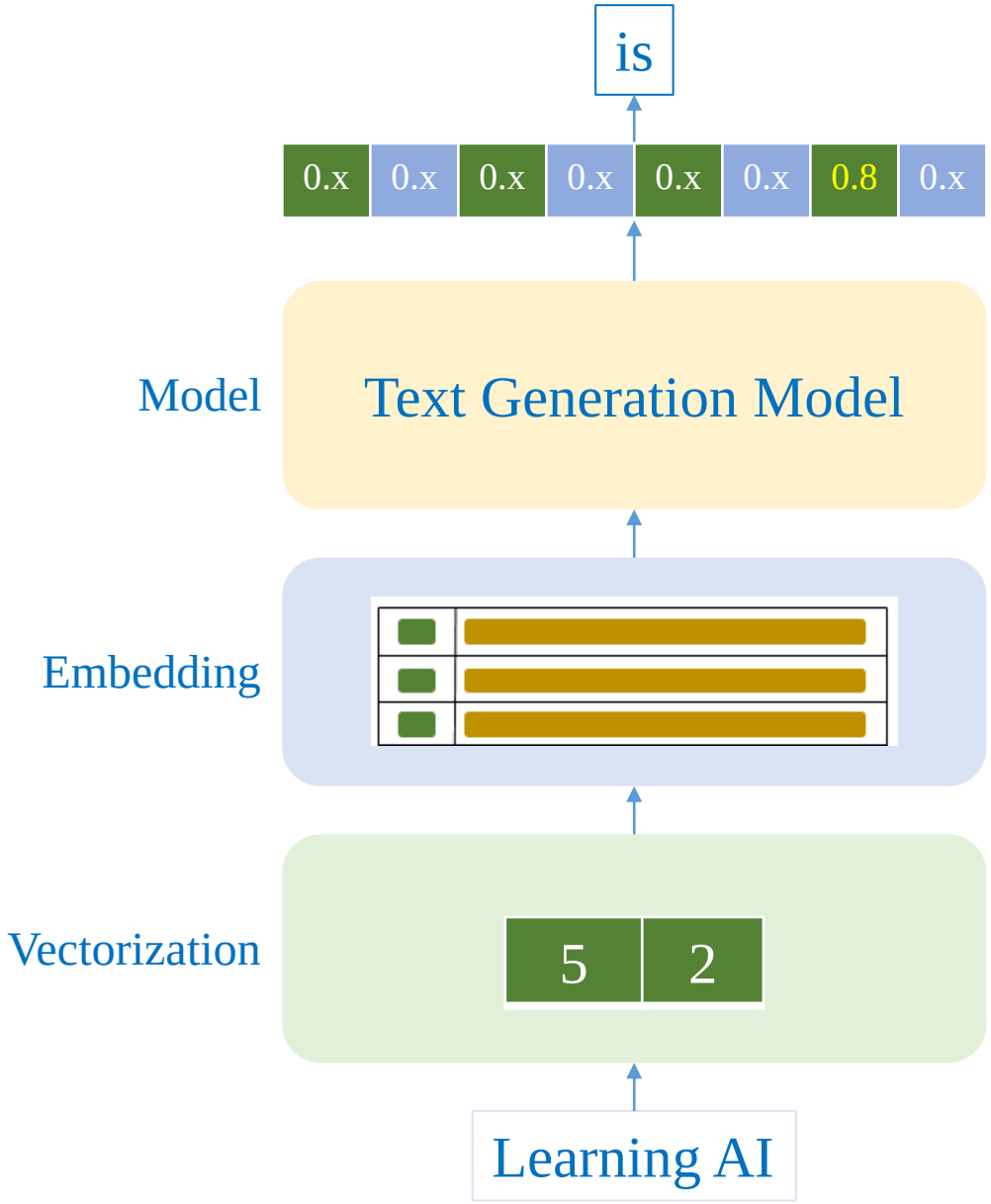
➡ size = vocab_size

Text Generation

Abstract view

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

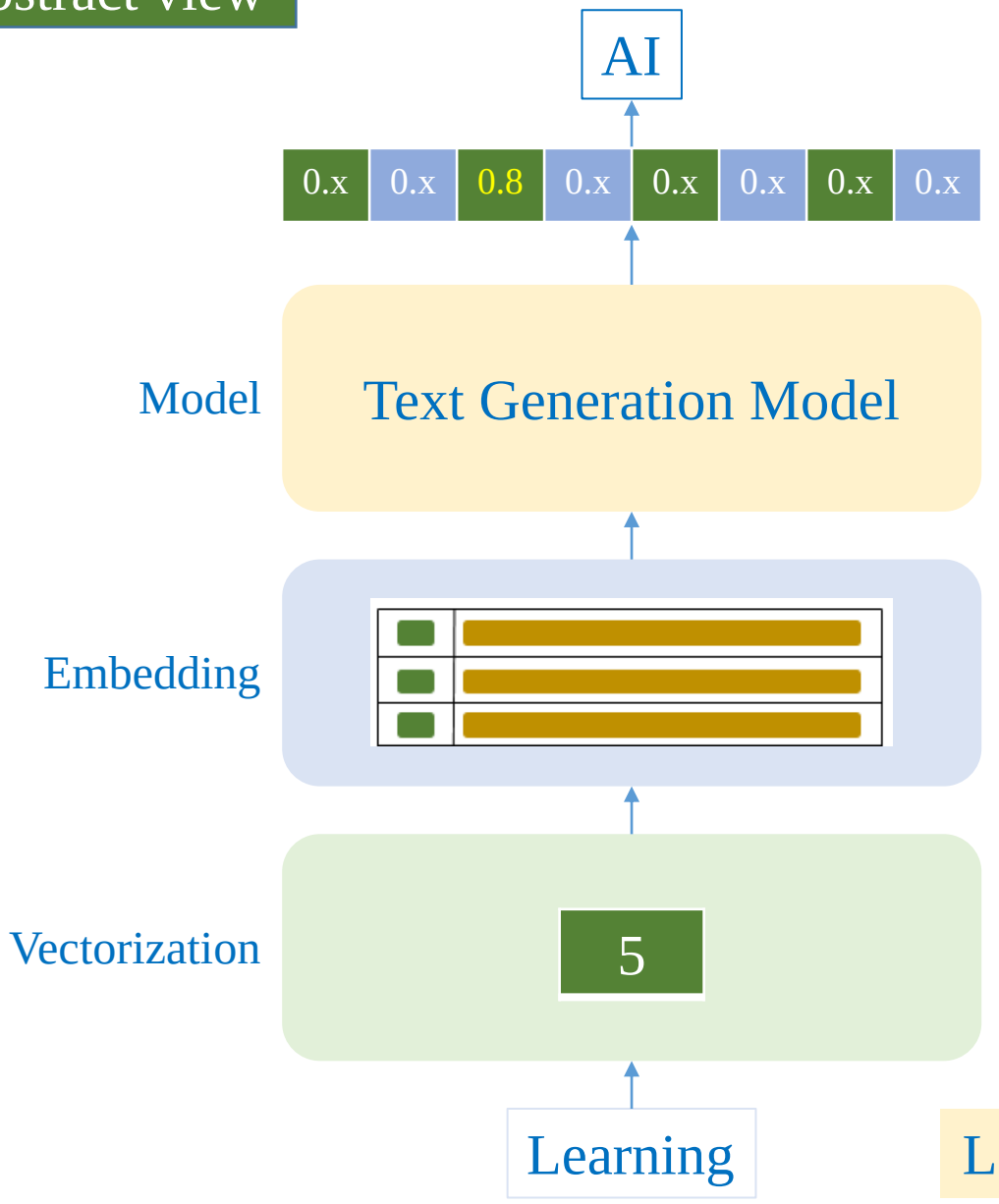


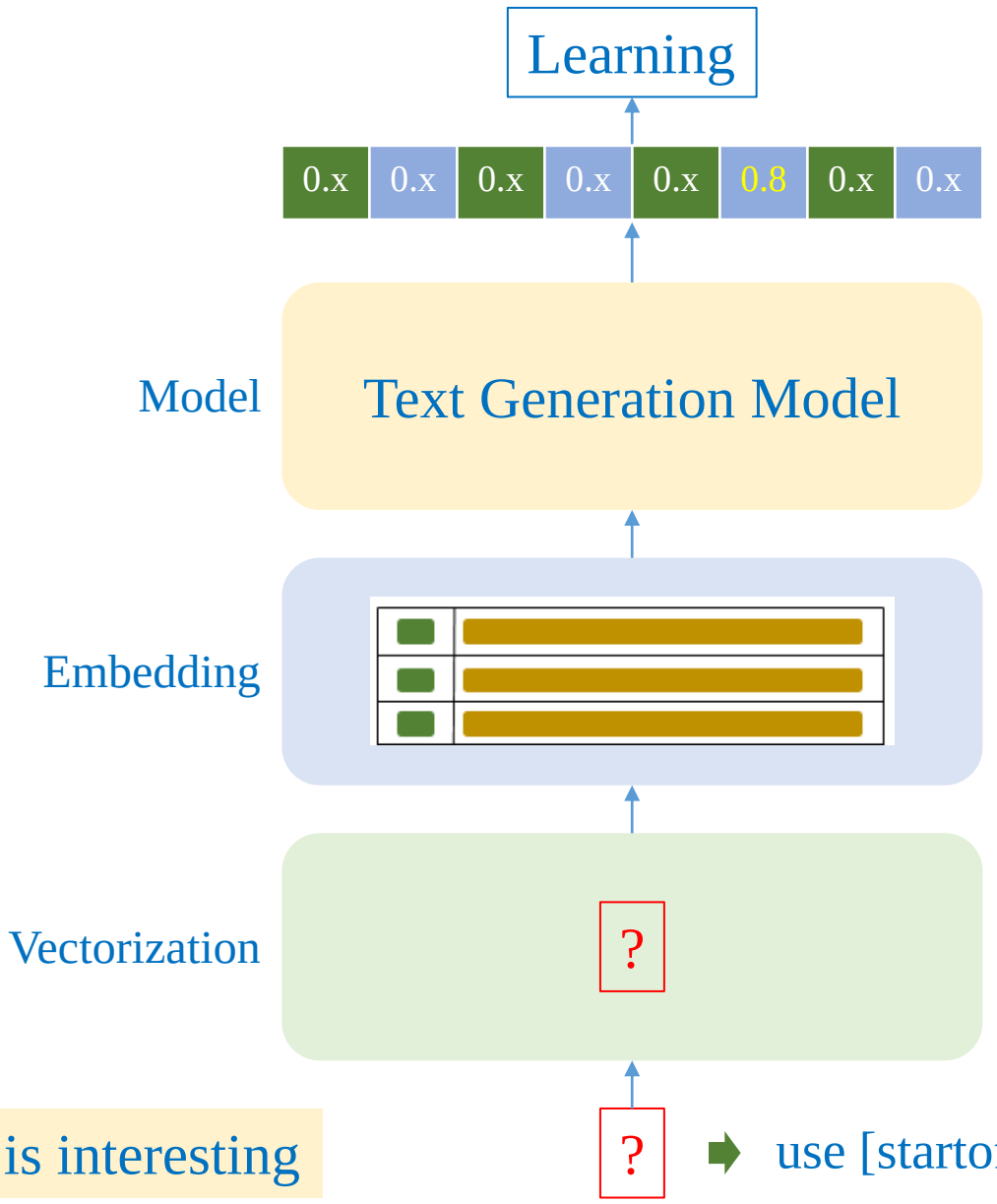


Text Generation

Abstract view

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



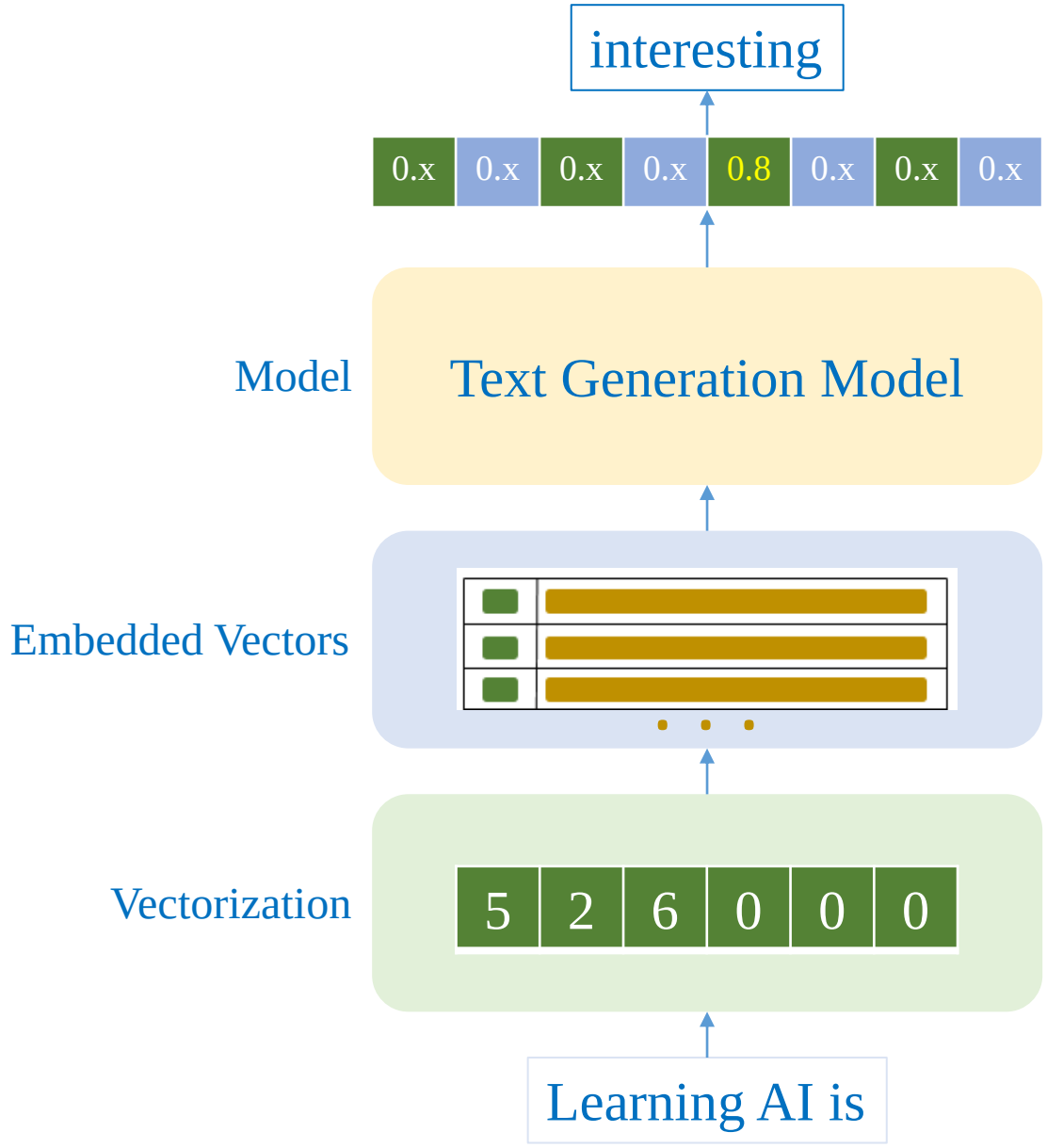


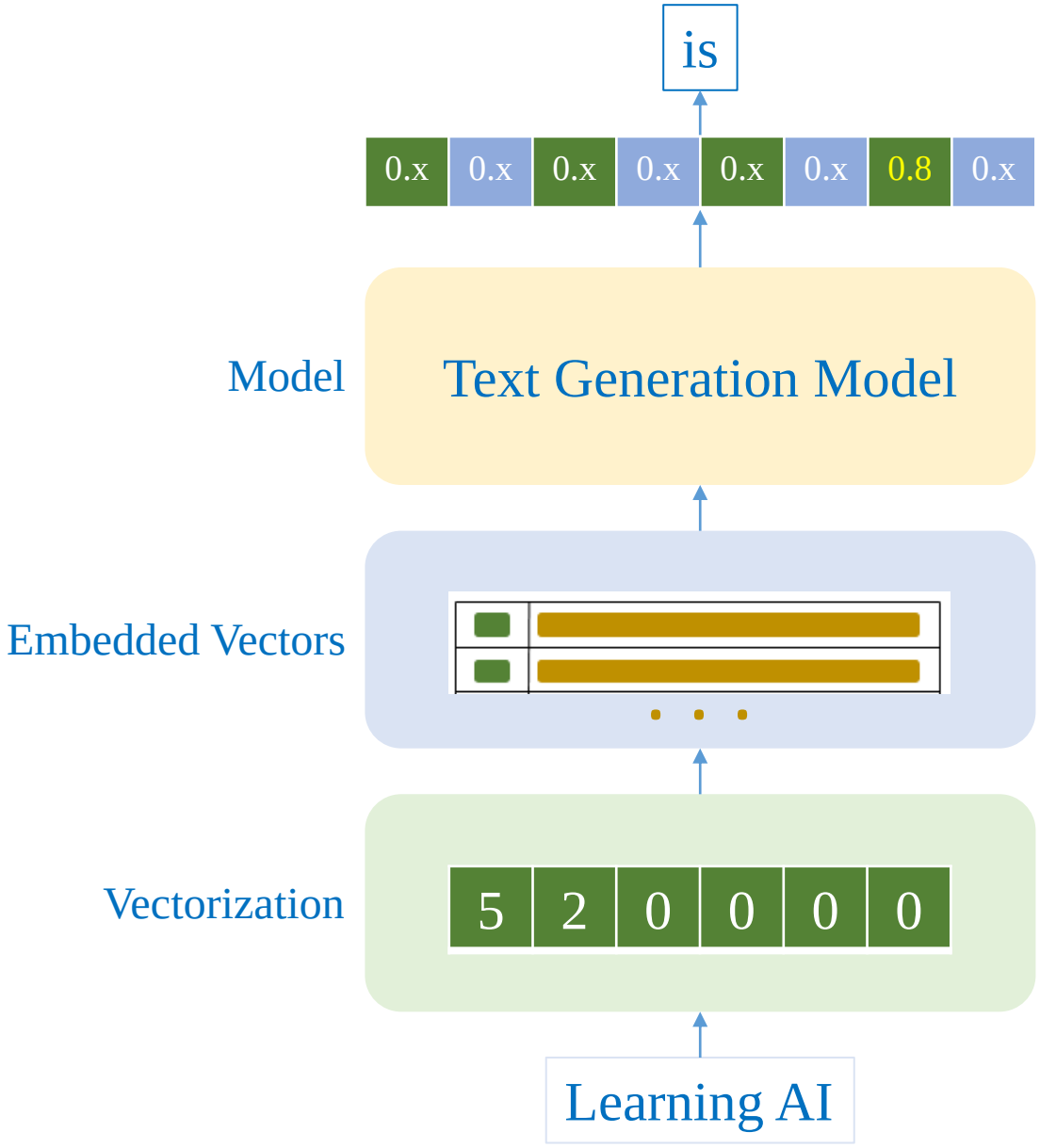
Learning AI is interesting

use [startoftext]

sequence_length = 6

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





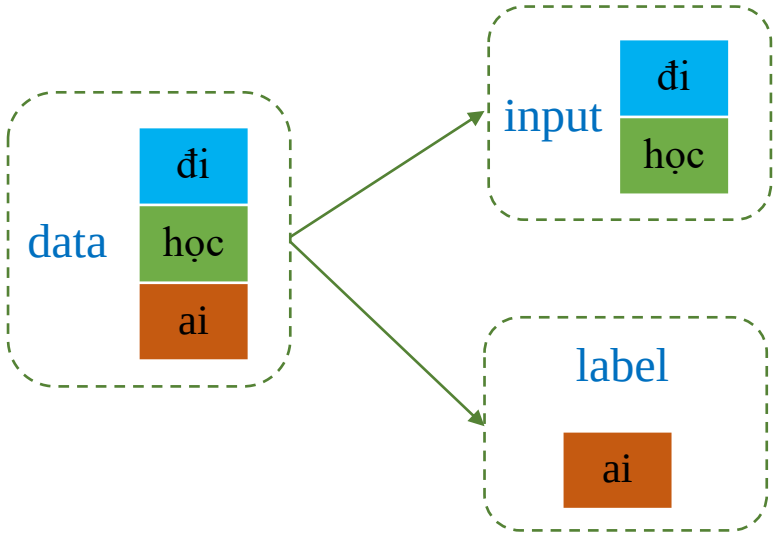
Outline

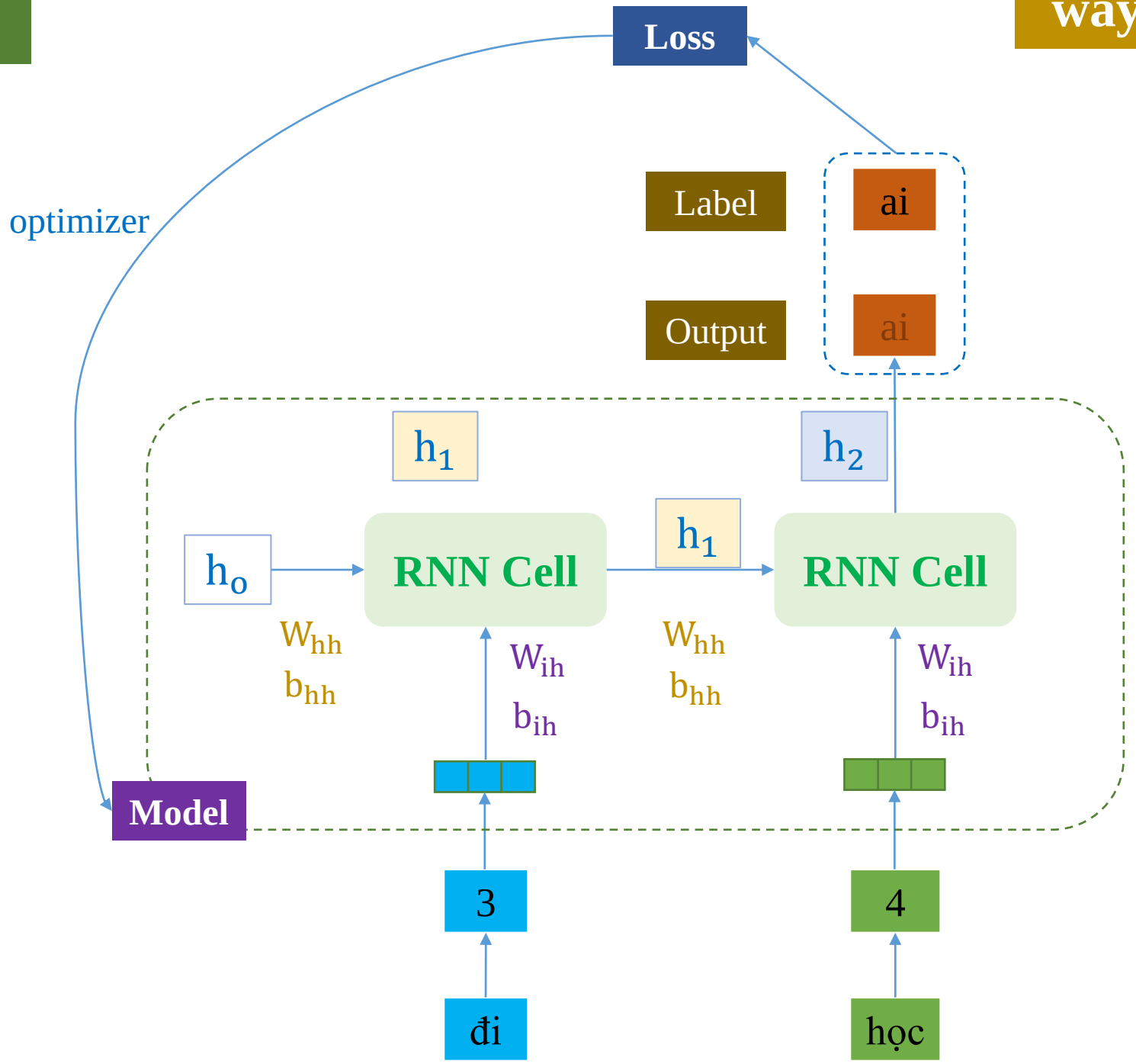
- Motivation to Text Generation
- Simple Models
- Using RNNs
- Examples
- Using Masked Encoder (~Decoder)

Implementation Using RNN

way 1

Extract context from input

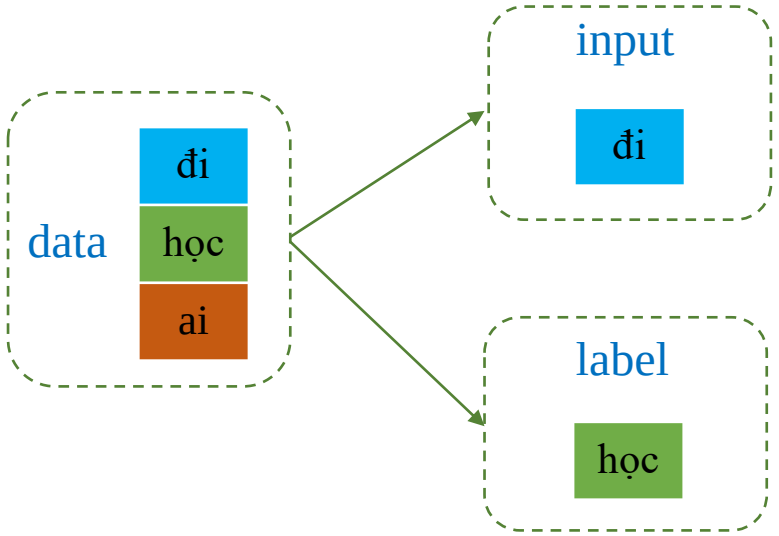


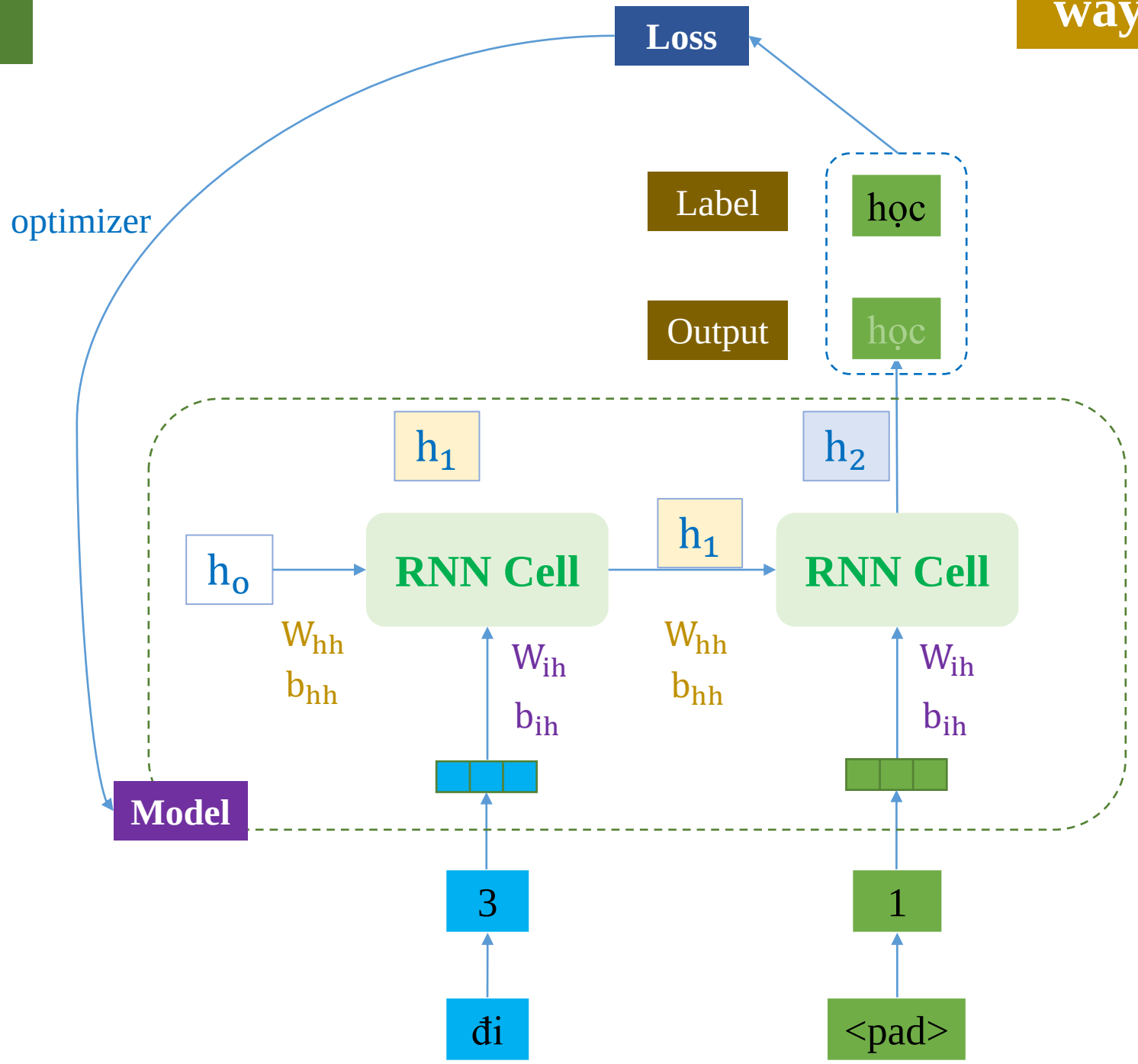


Implementation Using RNN

way 1

Extract context from input

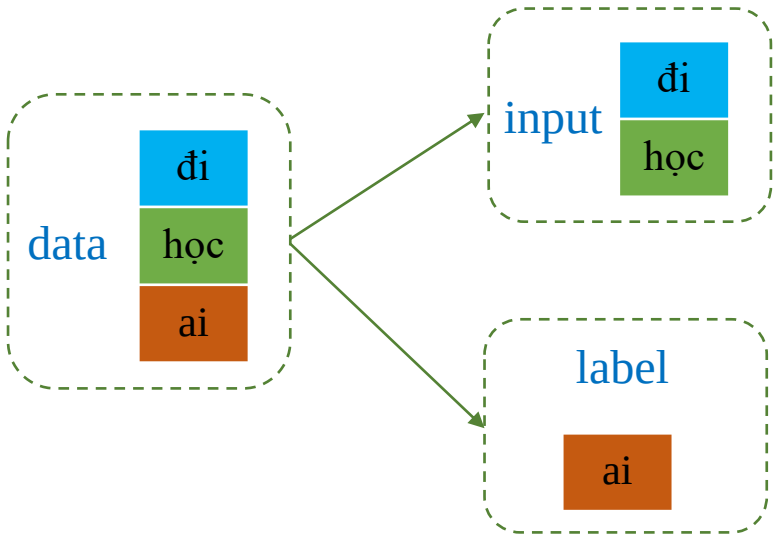


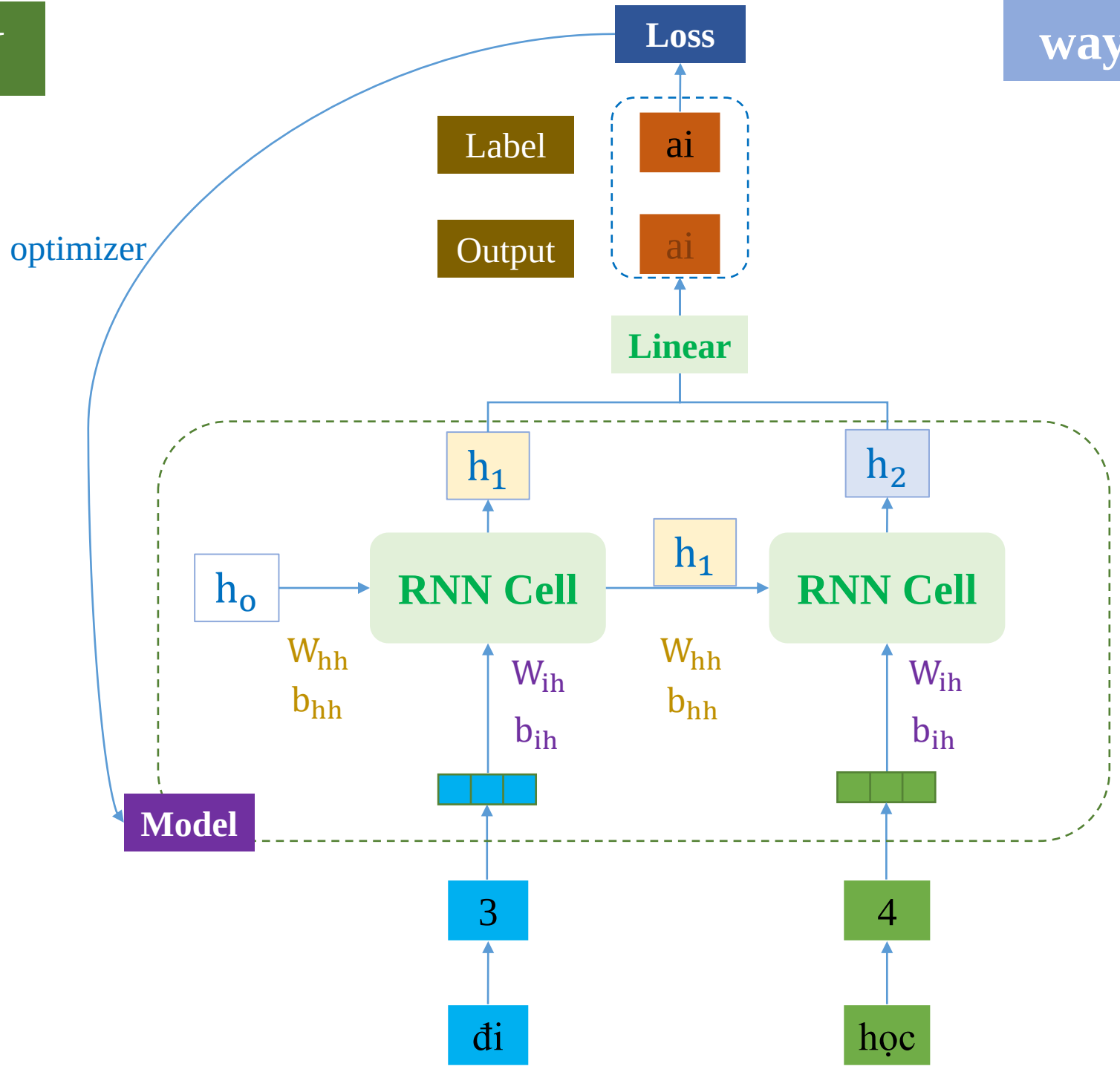


Implementation Using RNN

way 2

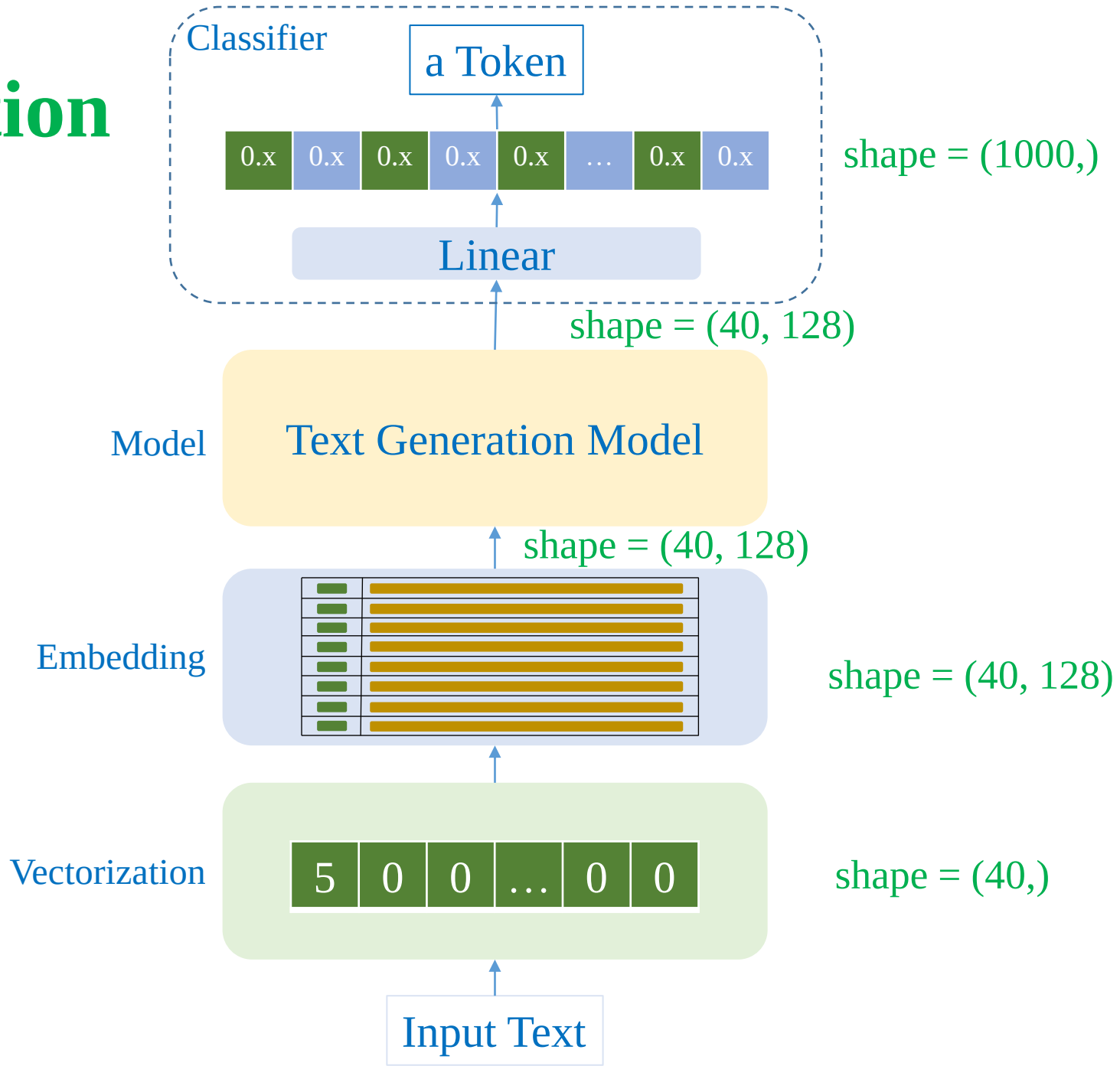
Using all the features





Text Generation

❖ **Practice**



sequence_length = 40

embed_dim = 128

vocab_size = 1000

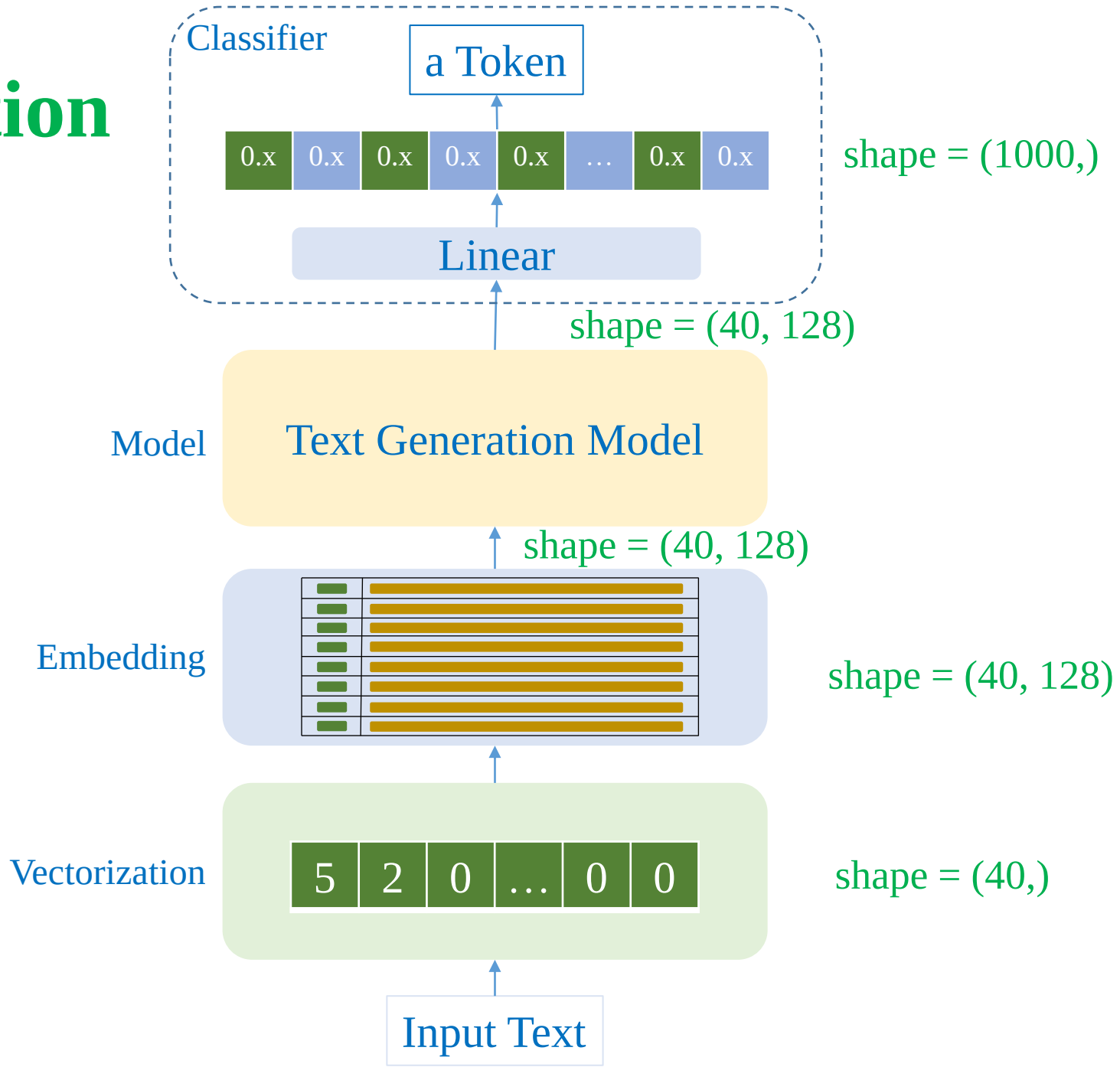
'Learning AI is interesting'

x = 'Learning'

y = 'AI'

Text Generation

❖ **Practice**



sequence_length = 40

embed_dim = 128

vocab_size = 1000

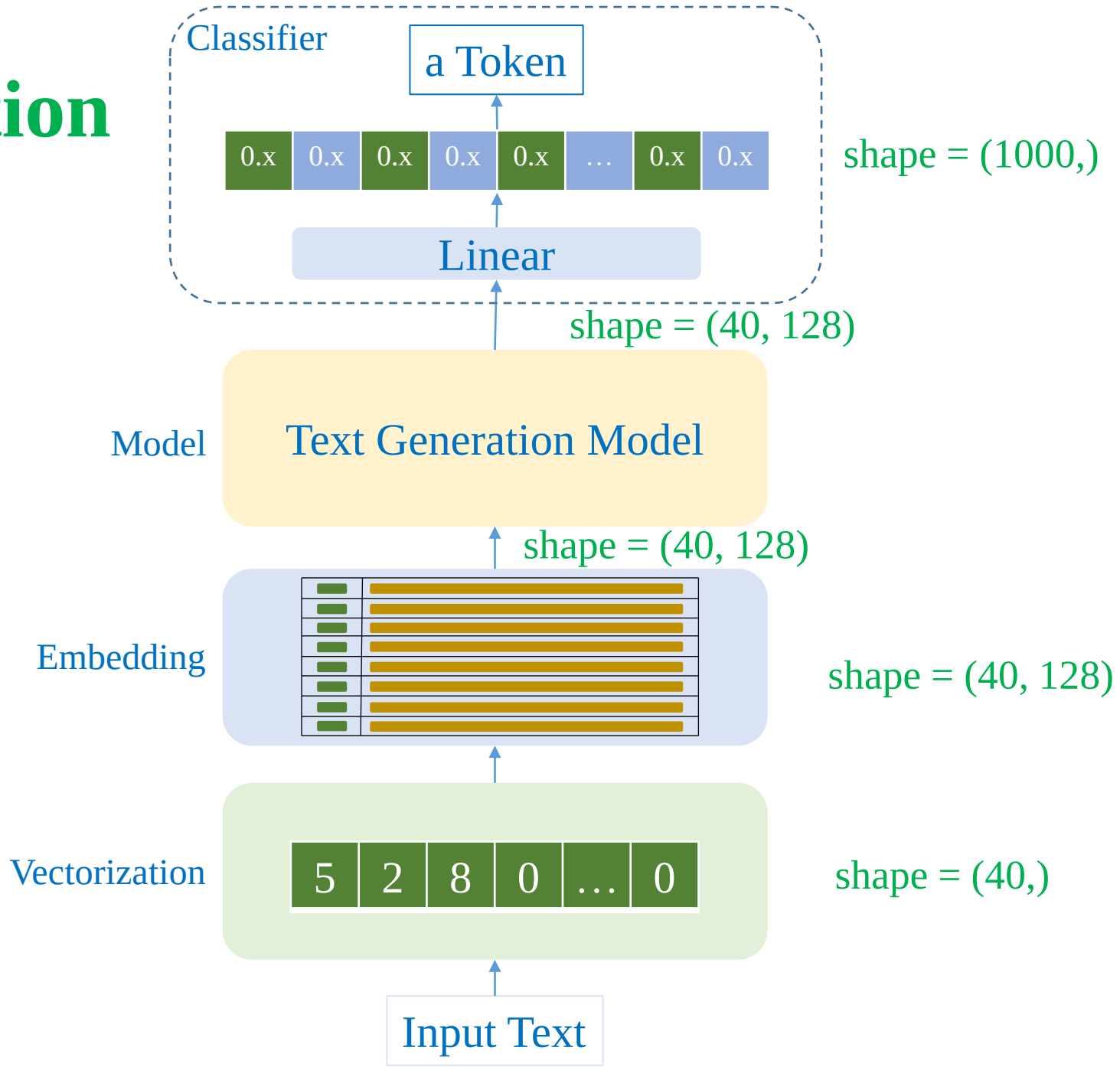
'Learning AI is interesting'

x = 'Learning AI'

y = 'is'

Text Generation

❖ **Practice**



sequence_length = 40

embed_dim = 128

vocab_size = 1000

'Learning AI is interesting'

x = 'Learning AI is'

y = 'interesting'

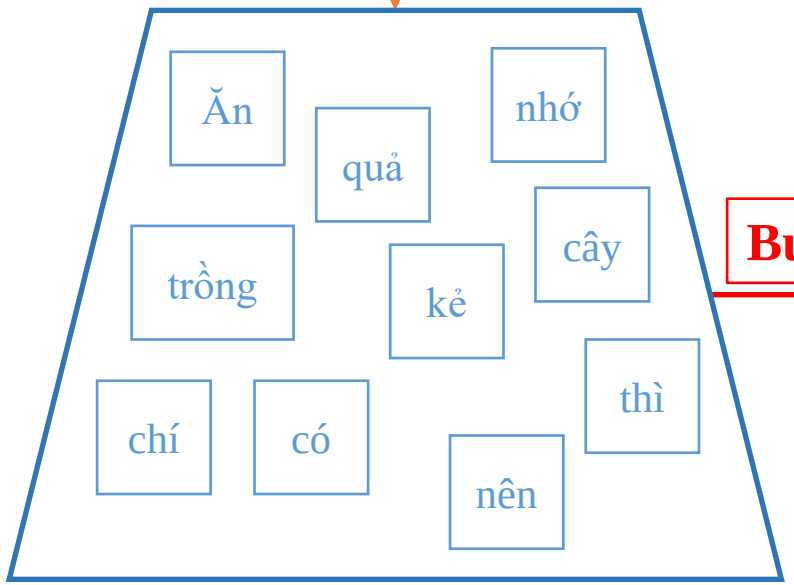
Loss function?

Example

Copus

Id	Text
0	Ăn quả nhớ kẻ trồng cây
1	Có chí thì nên

Tokenizer



Build vocab

Add

Special tokens

<unk> <pad> <sos>

Vocab

Token	Id
<unk>	0
<pad>	1
<sos>	2
chí	3
cây	4
có	5
kẻ	6
nhớ	7
nên	8
quả	9
thì	10
trồng	11
ăn	12

Example

Id	Text
0	Ăn quả nhớ kẻ trồng cây
1	Có chí thì nên

Input tokens	Target token
<sos>	Ăn
<sos> Ăn	quả
<sos> Ăn quả	nhớ
<sos> Ăn quả nhớ	kẻ
<sos> Ăn quả nhớ kẻ	trồng
<sos> Ăn quả nhớ kẻ trồng	cây
<sos>	Có
<sos> Có	chí
<sos> Có chí	thì
<sos> Có chí thì	nên

Next token prediction dataset

Token	Id	kẻ	6
<unk>	0	nhớ	7
<pad>	1	nên	8
<sos>	2	quả	9
chí	3	thì	10
cây	4	trồng	11
có	5	ăn	12

Vocab

padding

Input ids	Target ids
[2, 1, 1, 1, 1, 1]	12
[2, 12, 1, 1, 1, 1]	9
[2, 12, 9, 1, 1, 1]	7
[2, 12, 9, 7, 1, 1]	6
[2, 12, 9, 7, 6, 1]	11
[2, 12, 9, 7, 6, 11]	4
[2, 1, 1, 1, 1, 1]	5
[2, 5, 1, 1, 1, 1]	3
[2, 5, 3, 1, 1, 1]	10
[2, 5, 3, 10, 1, 1]	8

Training data

sequence_length = 6

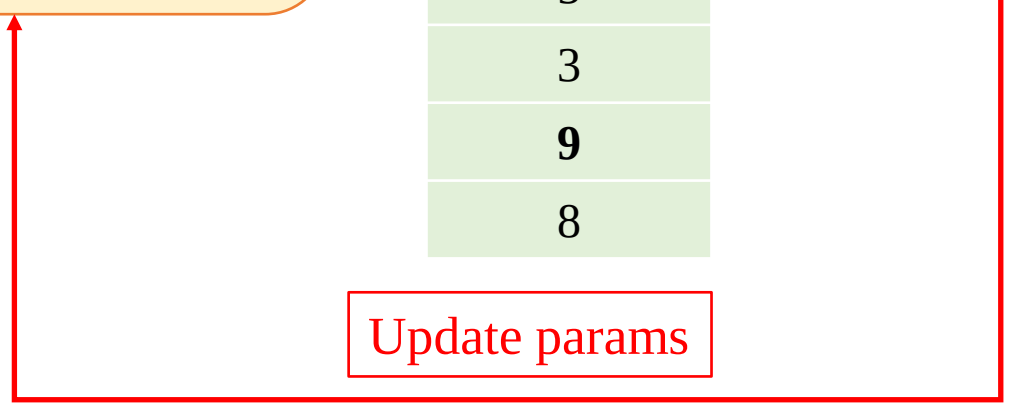
Example

Training

X ids
[2, 1, 1, 1, 1, 1]
[2, 12, 1, 1, 1, 1]
[2, 12, 9, 1, 1, 1]
[2, 12, 9, 7, 1, 1]
[2, 12, 9, 7, 6, 1]
[2, 12, 9, 7, 6, 11]
[2, 1, 1, 1, 1, 1]
[2, 5, 1, 1, 1, 1]
[2, 5, 3, 1, 1, 1]
[2, 5, 3, 10, 1, 1]

predict ids
12
9
7
6
12
4
5
3
9
8

target ids
12
9
7
6
11
4
5
3
10
8



sequence_length = 4

hidden_dim = 4

Input tokens	Target token
<sos> Có chí thì	nên

Input ids	Target ids
[2, 5, 3, 10]	[8]

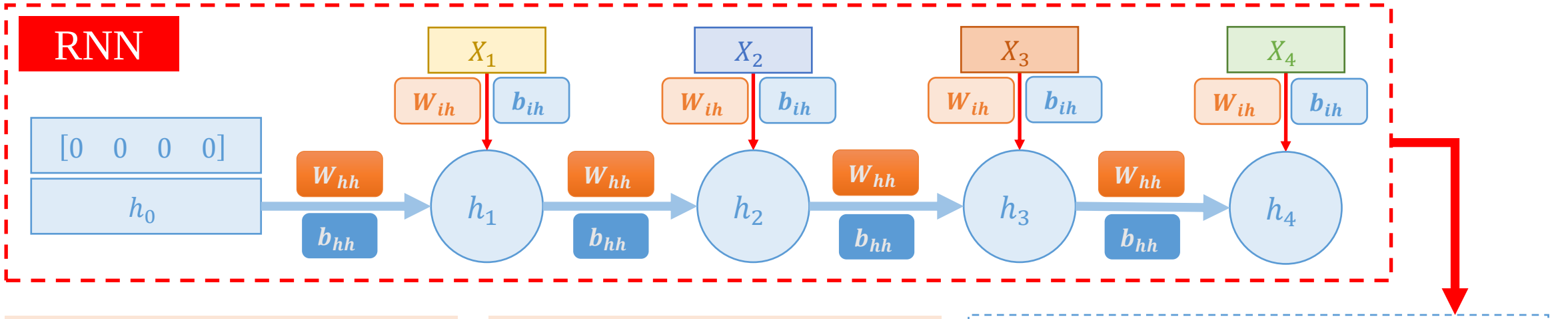
[2 5 3 10] → Embedding layer →

$X_1 = [-0.7521 \quad 1.6487 \quad -0.3925 \quad -1.4036]$

$X_2 = [-0.7581 \quad 1.0783 \quad 0.8008 \quad 1.6806]$

$X_3 = [-0.7279 \quad -0.5594 \quad -0.7688 \quad 0.7624]$

$X_4 = [-0.8371 \quad -0.9224 \quad 1.8113 \quad 0.1606]$



W_{hh}

$$\begin{bmatrix} 0.3035 & -0.1187 & 0.2860 & -0.3885 \\ -0.2523 & 0.1524 & 0.1057 & -0.1275 \\ 0.2980 & 0.3399 & -0.3626 & -0.2669 \\ 0.4578 & -0.1687 & -0.1773 & -0.4838 \end{bmatrix}$$

W_{ih}

$$\begin{bmatrix} -0.2982 & -0.2982 & 0.4497 & 0.1666 \\ 0.4811 & -0.4126 & -0.4959 & -0.3912 \\ -0.3363 & 0.2025 & 0.1790 & 0.4155 \\ -0.2582 & -0.3409 & 0.2653 & -0.2021 \end{bmatrix}$$

b_{hh}

$[0.0117 \quad -0.3415 \quad -0.4242 \quad -0.2753]$

b_{ih}

$[-0.2863 \quad 0.1249 \quad -0.0660 \quad -0.3629]$

$h_1 = [-0.7409 \quad -0.4739 \quad -0.5055 \quad -0.6786]$

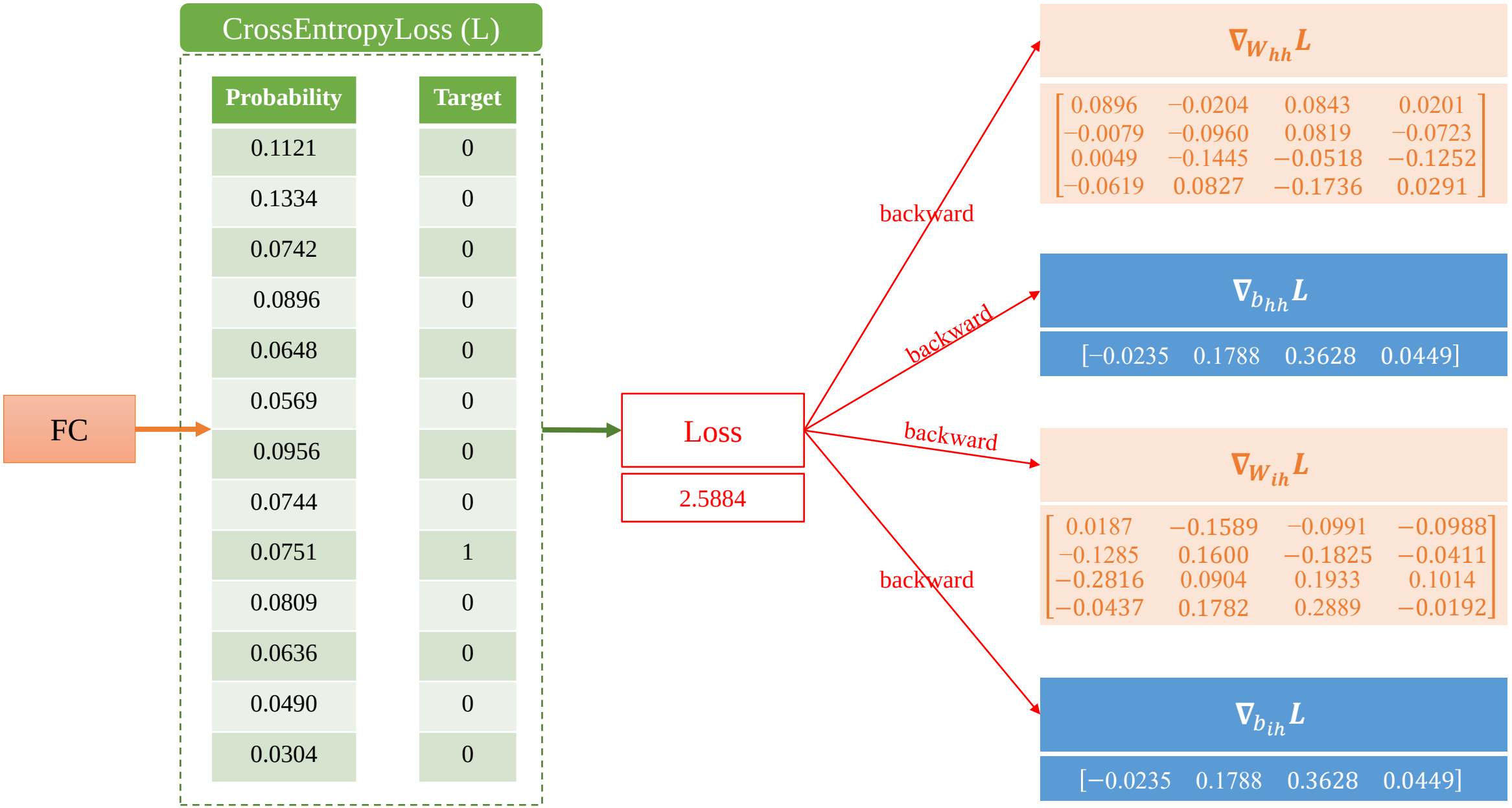
$h_2 = [0.2170 \quad -0.9590 \quad 0.6681 \quad -0.6519]$

$h_3 = [0.4735 \quad -0.2915 \quad -0.4692 \quad -0.1583]$

$h_4 = [0.8327 \quad -0.8839 \quad 0.2449 \quad 0.6446]$

Flatten → FC

Back-Propagation



2nd Feed Forward

Update Parameters
SGD($lr = 0.1$)

$$W_{hh} = W_{hh} - lr \times dW_{hh} \rightarrow \begin{bmatrix} 0.2945 & -0.1166 & 0.2776 & -0.3905 \\ -0.2515 & 0.1620 & 0.0975 & -0.1203 \\ 0.2975 & 0.3544 & -0.3574 & -0.2544 \\ 0.4640 & -0.1770 & -0.1599 & -0.4867 \end{bmatrix}$$

$$b_{hh} = b_{hh} - lr \times db_{hh} \rightarrow [0.0141 \quad -0.3594 \quad -0.4605 \quad -0.2798]$$

$$W_{ih} = W_{ih} - lr \times dW_{ih} \rightarrow \begin{bmatrix} -0.3001 & -0.2823 & 0.4596 & 0.1765 \\ 0.4940 & -0.4286 & -0.4777 & -0.3871 \\ -0.3082 & 0.1935 & 0.1597 & 0.4053 \\ -0.2538 & -0.3587 & 0.2364 & -0.2002 \end{bmatrix}$$

$$b_{ih} = b_{ih} - lr \times db_{ih} \rightarrow [-0.2840 \quad 0.1070 \quad -0.1023 \quad -0.3674]$$

Forward again

CrossEntropyLoss (L)

Probability	Target
0.1044	0
0.1163	0
0.0659	0
0.0865	0
0.0575	0
0.0521	0
0.0894	0
0.0673	0
0.1506	1
0.0754	0
0.0578	0
0.0488	0
0.0280	0

Loss

1.8929

Outline

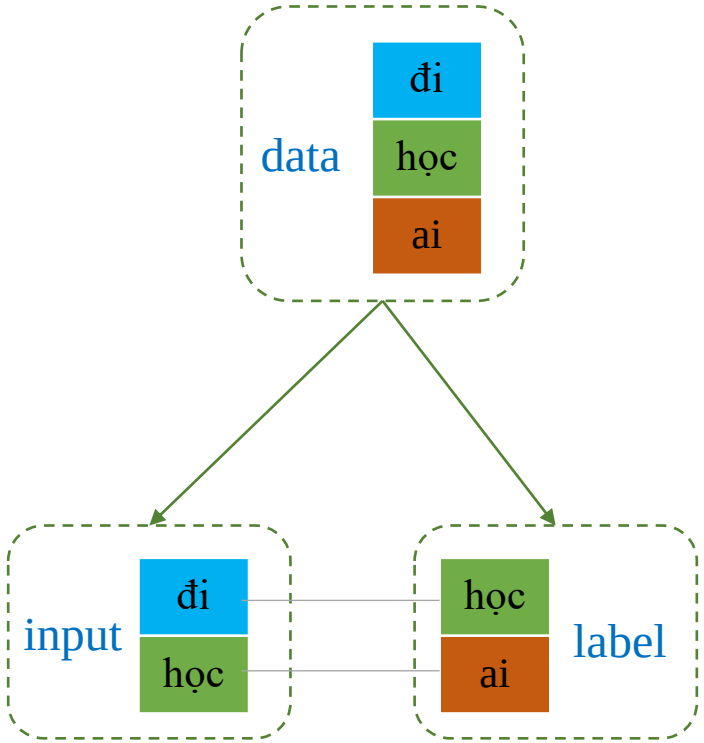
- Motivation to Text Generation
- Simple Models
- Using RNNs
- Examples
- Using Masked Encoder (~Decoder)

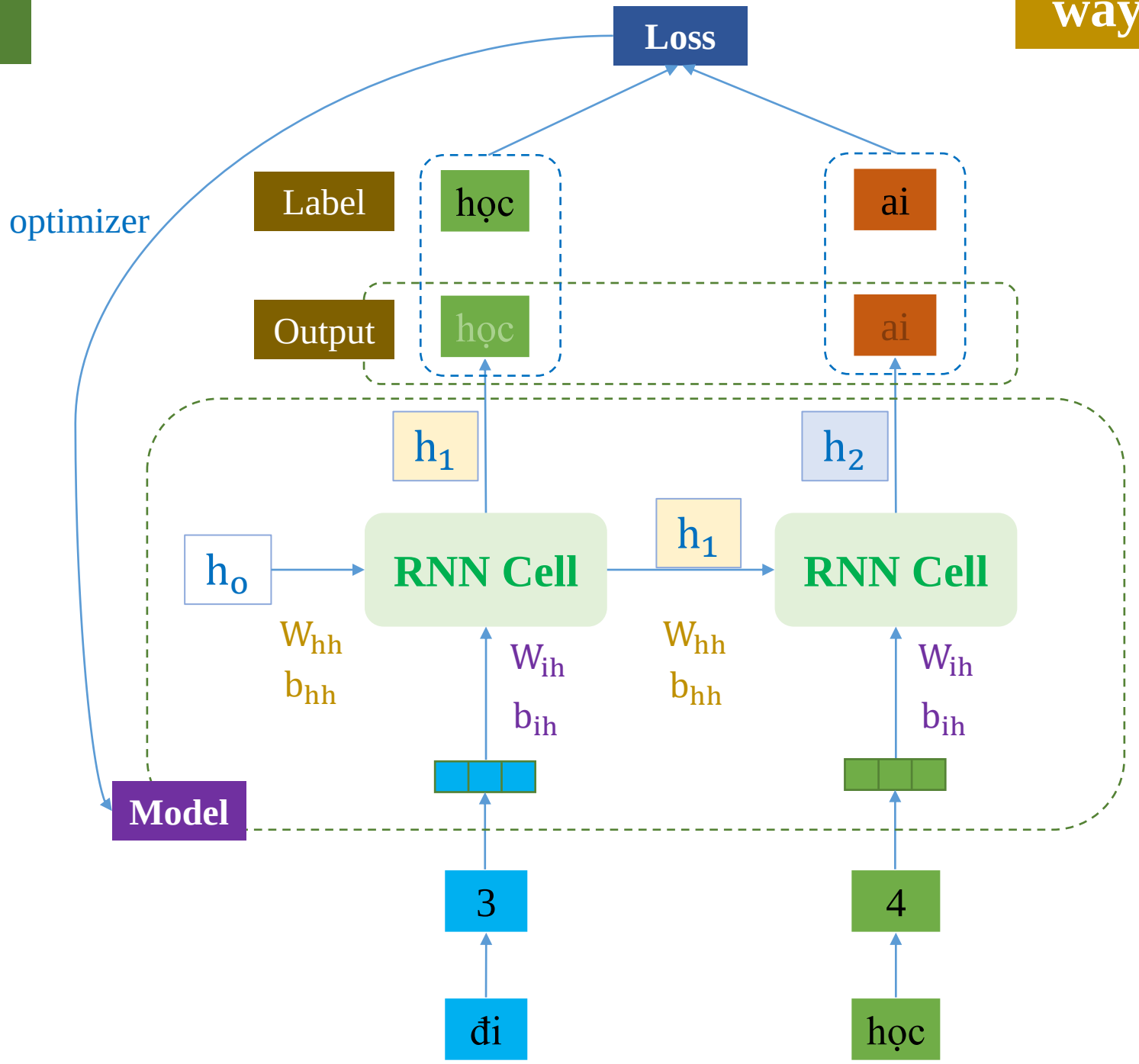
Implementation Using RNN

way 3

RNNs encode input sequentially

*Linear can be added (where?)

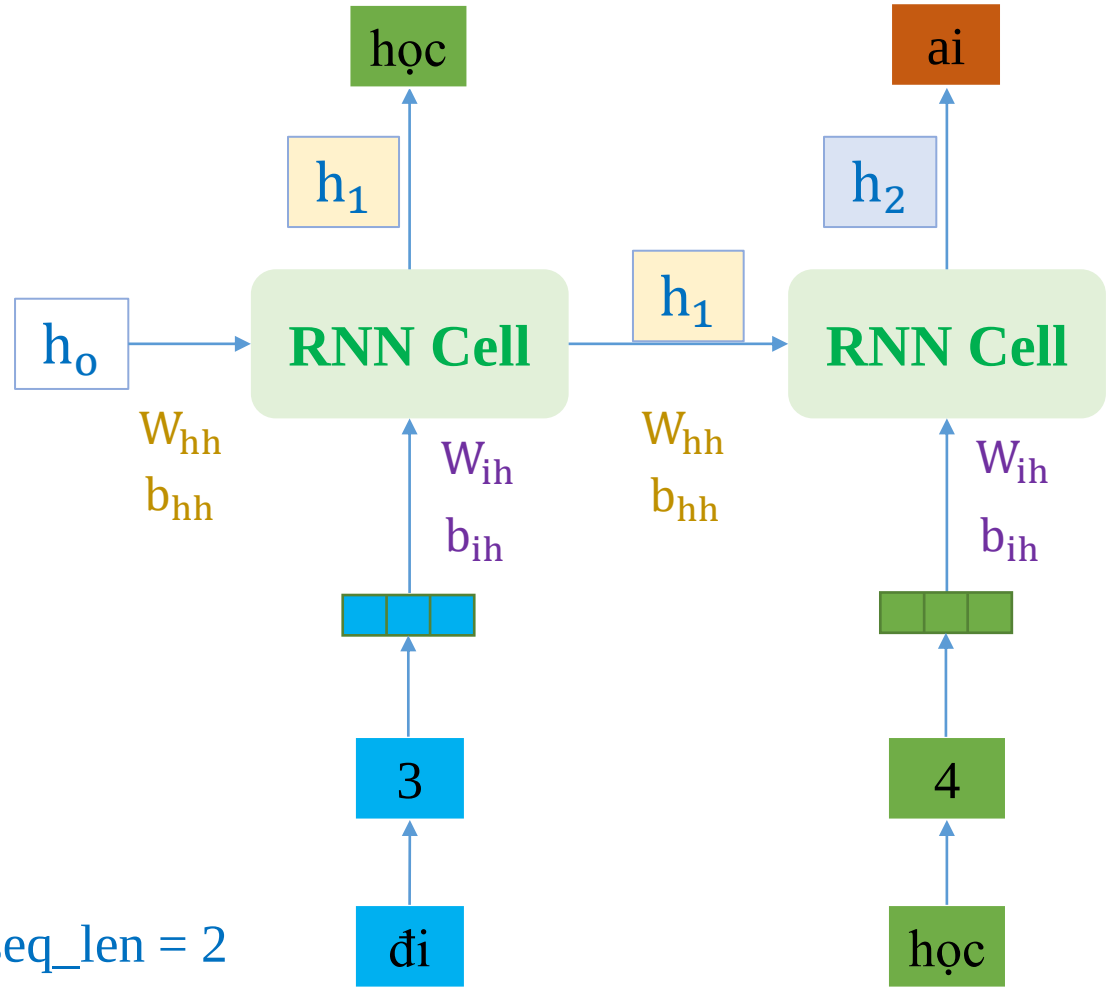




Implementation Using RNN

RNNs encode input sequentially

way 3



seq_len = 2

hidden_dim = vocab_size = 4

[-0.88, 0.81, 0.77]

[-1.27, 0.84, 0.04]

Embedded Input

W_{ih}

[0.25, 0.05, 0.17]

[-0.11, 0.35, 0.22]

[0.44, -0.22, -0.42]

[0.38, 0.36, -0.01]

b_{ih} [-0.47, 0.13, 0.46, -0.15]

W_{hh}

[0.38, -0.06, 0.37, -0.19]

[-0.16, 0.42, 0.08, -0.02]

[-0.36, -0.01, 0.17, 0.39]

[-0.43, 0.09, 0.33, 0.03]

b_{hh} [0.43, 0.16, 0.34, -0.39]

Output

[-0.08, 0.68, -0.08, -0.53]

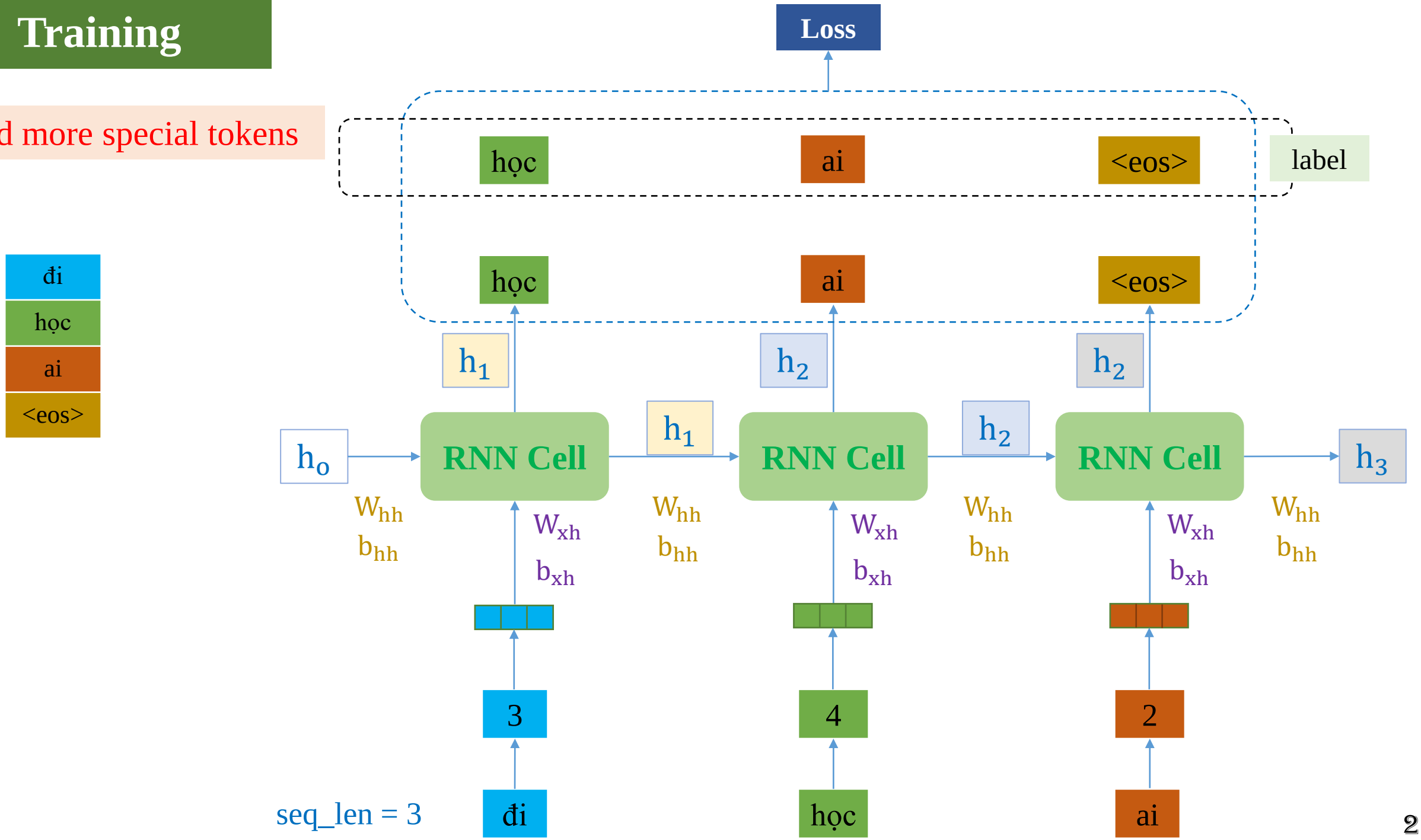
[-0.30, 0.77, -0.15, -0.58]

Hidden [-0.30, 0.77, -0.15, -0.58]

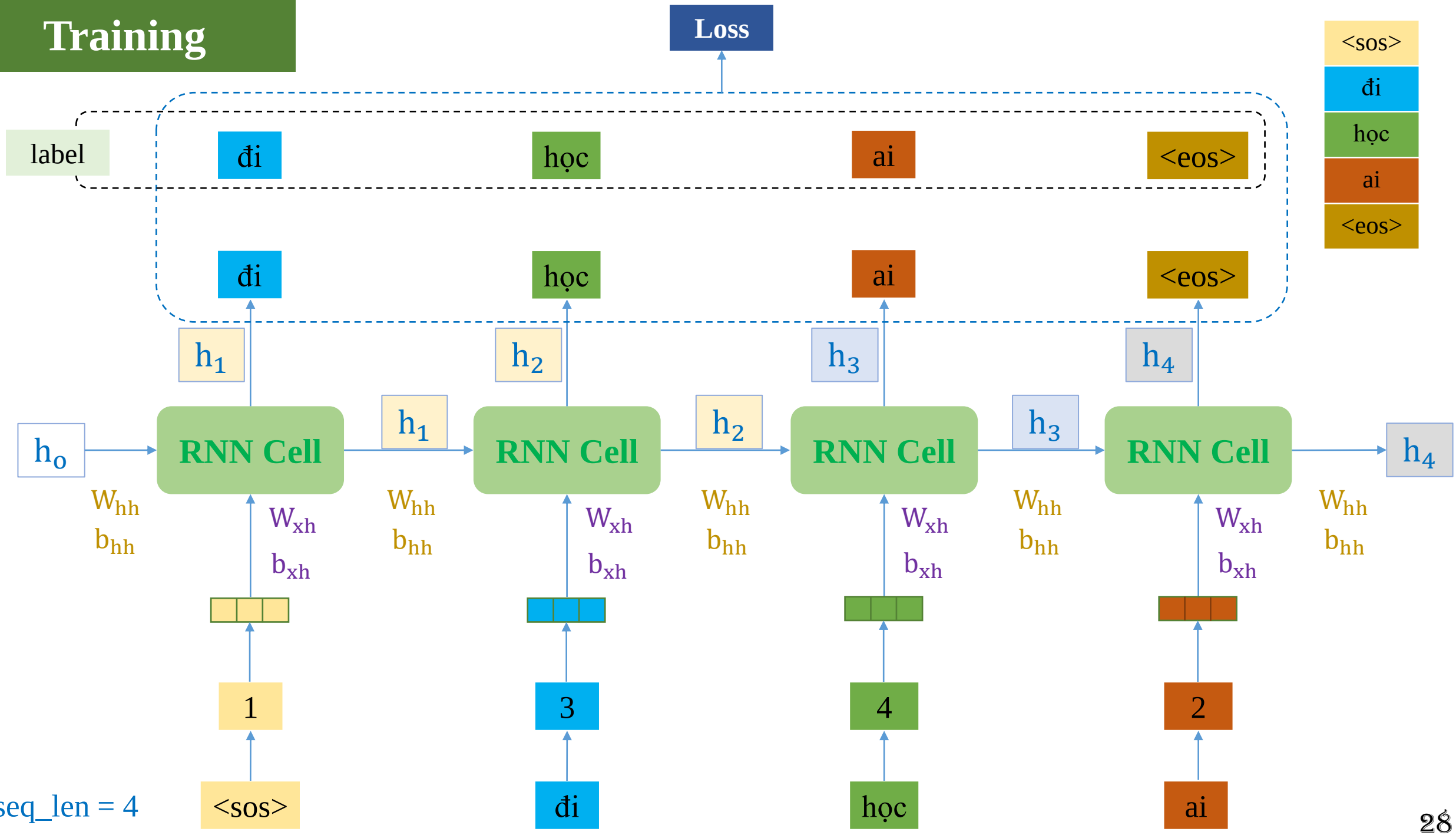
*Linear can be added

Training

Add more special tokens

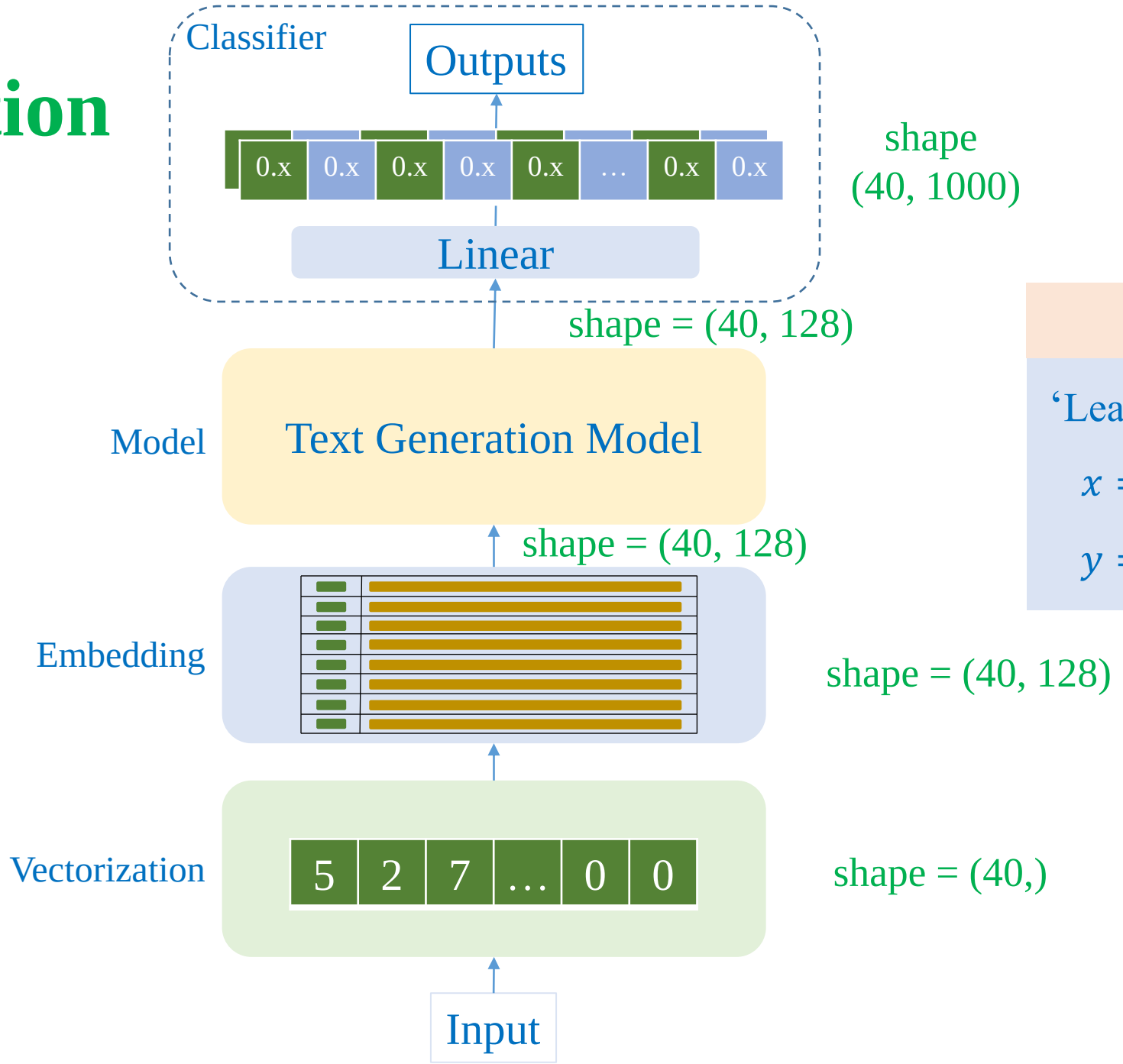


Training


Loss
label
đi
học
ai
<eos>
<sos>
đi
học
ai
<eos>
h1
h2
h3
h4
h0
RNN Cell
RNN Cell
RNN Cell
RNN Cell
h1
h2
h3
h4
Whh
bhh
Wxh
bxh
1
3
4
2
<sos>
đi
học
ai
seq_len = 4

Text Generation

❖ **Practice**



shape (40, 1000)

shape = (40, 128)

shape = (40,)

sequence_length = 40

embed_dim = 128

vocab_size = 1000

What about?

‘Learning AI is … very interesting’

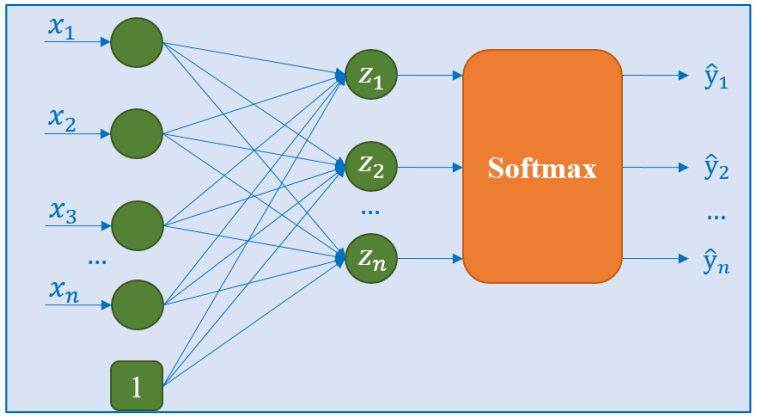
x = ‘Learning AI is … very’

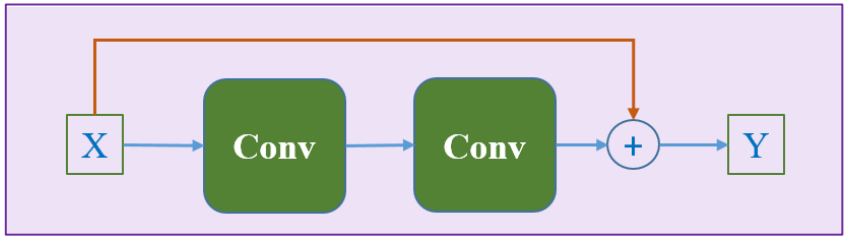
y = ‘AI is … very interesting’

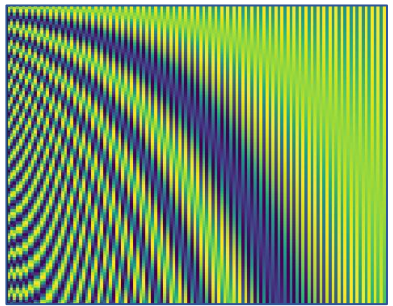
Outline

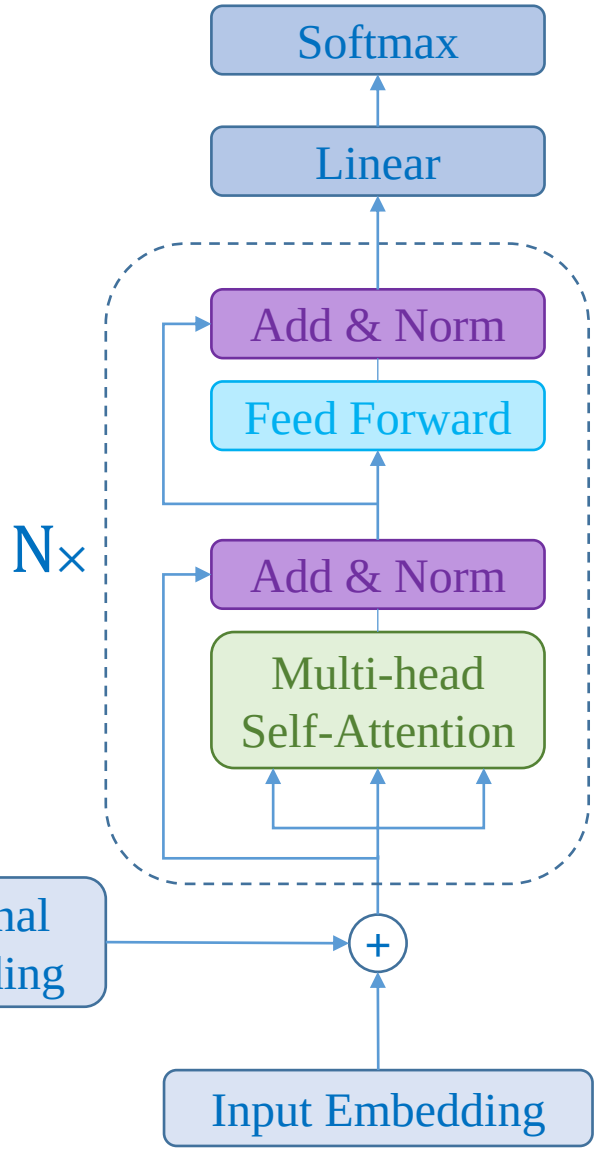
- Motivation to Text Generation
- Simple Models
- Using RNNs
- Examples
- Using Masked Encoder (~Decoder)

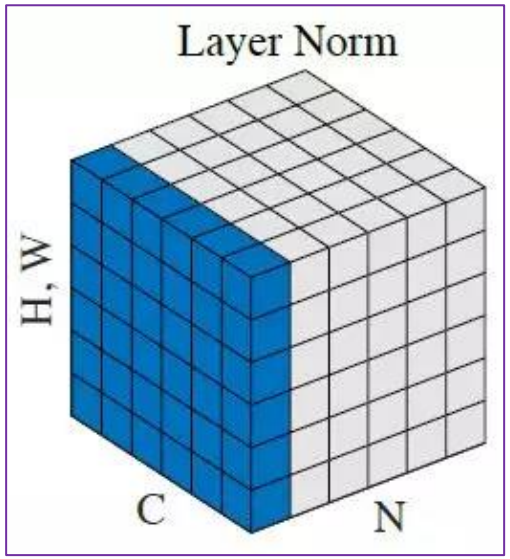
Transformer Encoder



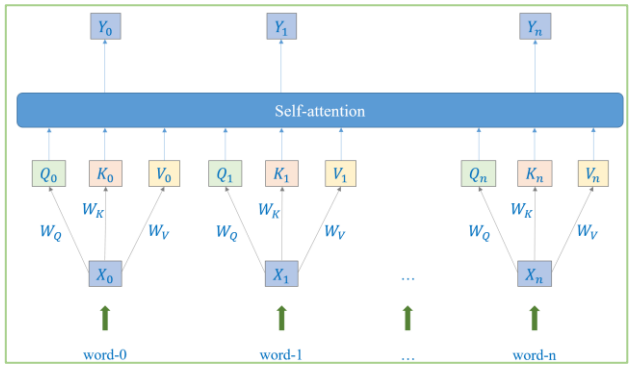


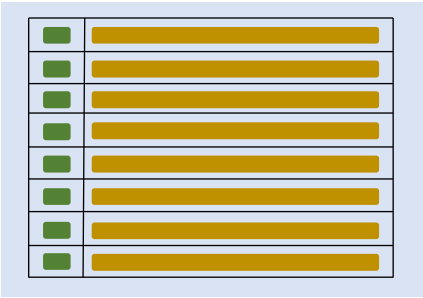


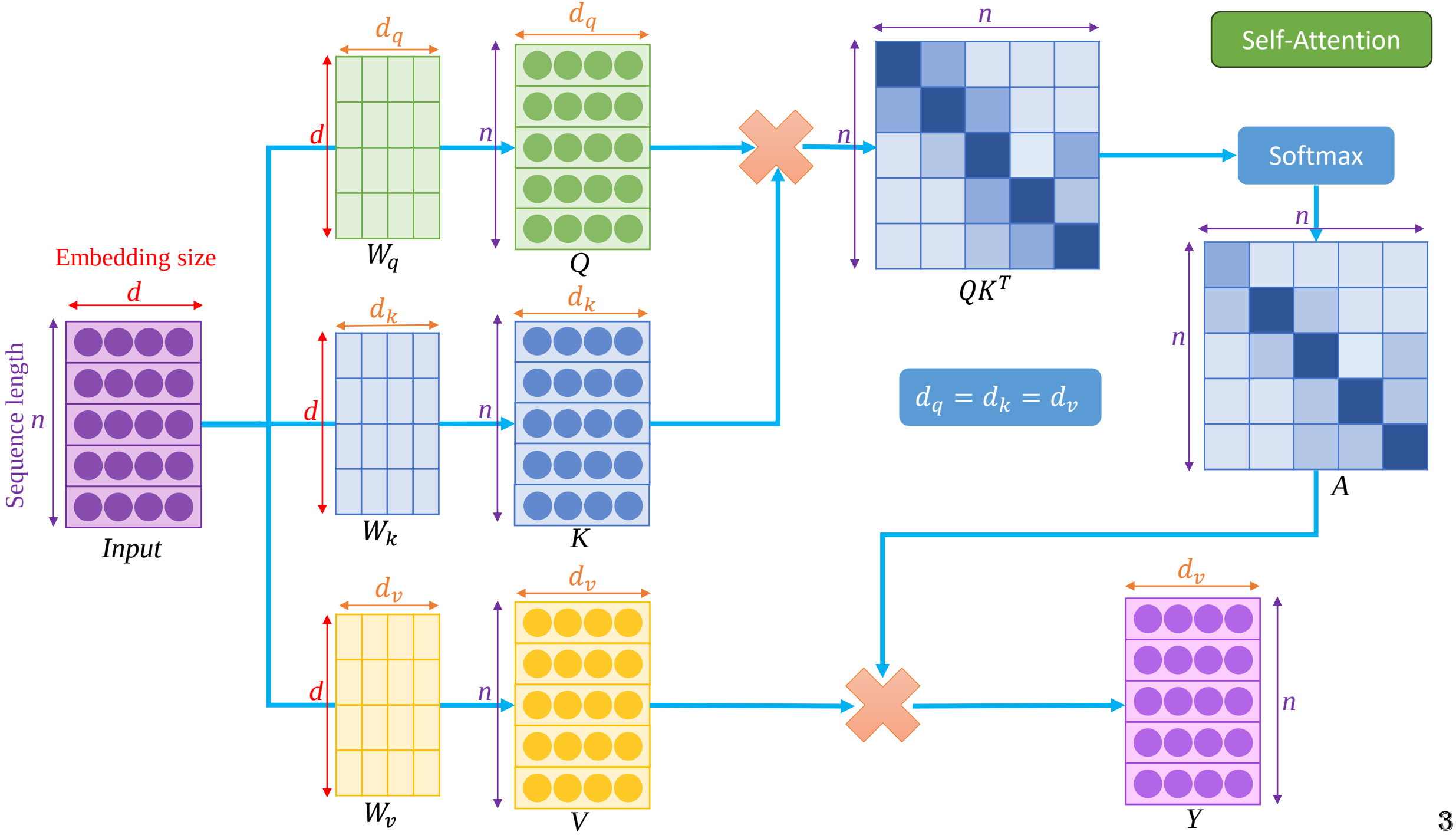




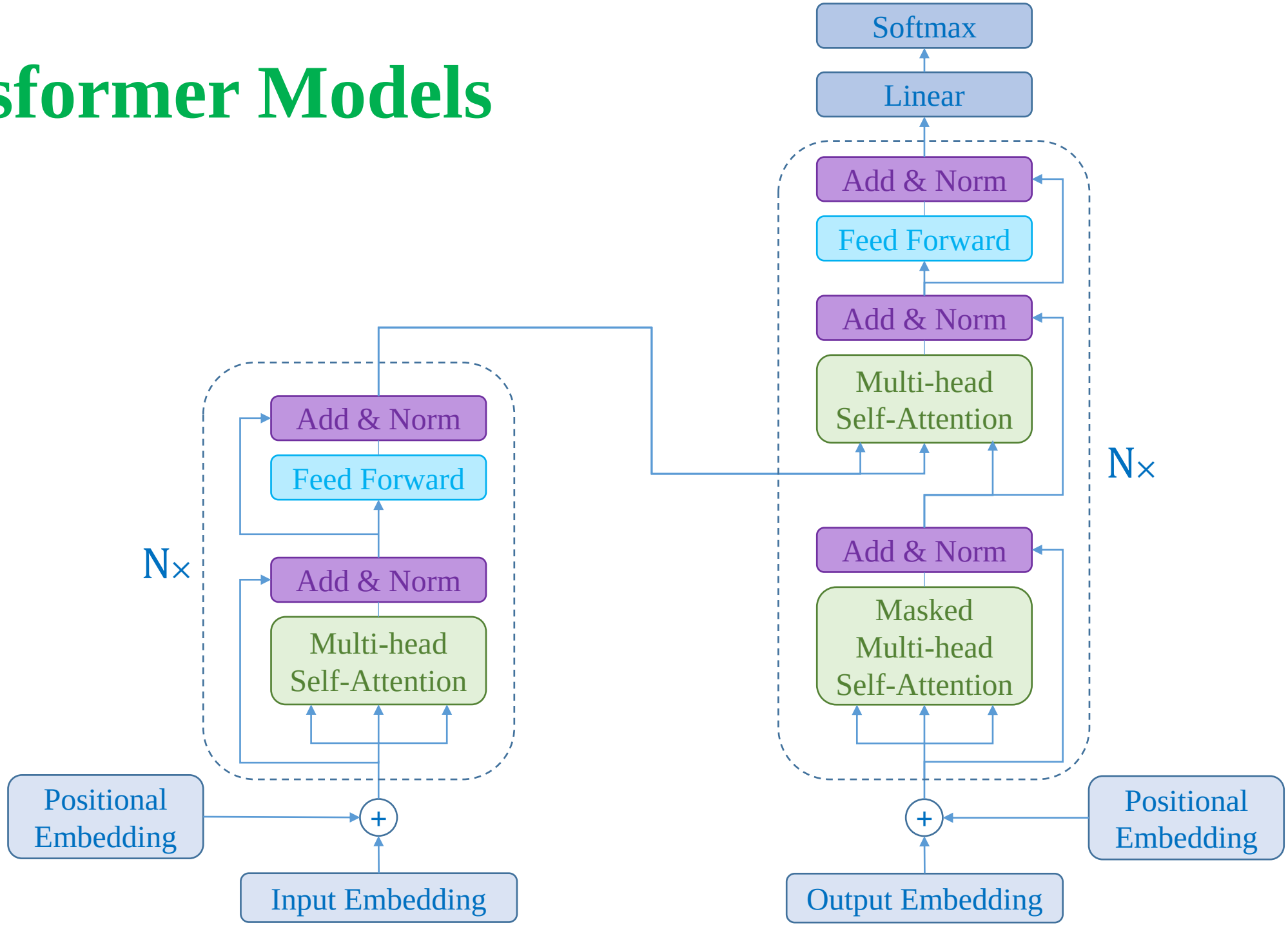
https://arxiv.org/pdf/1803.08494.pdf





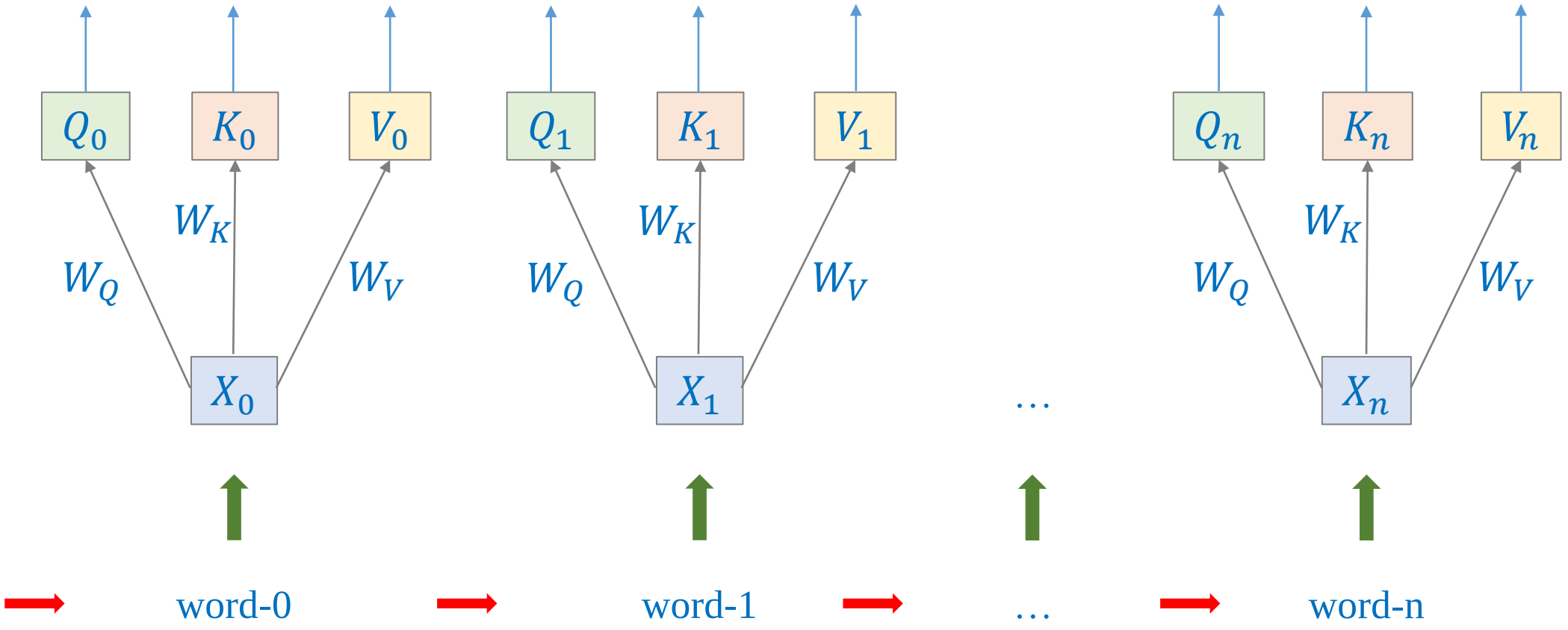

Self-Attention
Embedding size
Sequence length
Input
W_q
W_k
W_v
Q
K
V
QK^T
Softmax
A
Y
$d_q = d_k = d_v$

Transformer Models

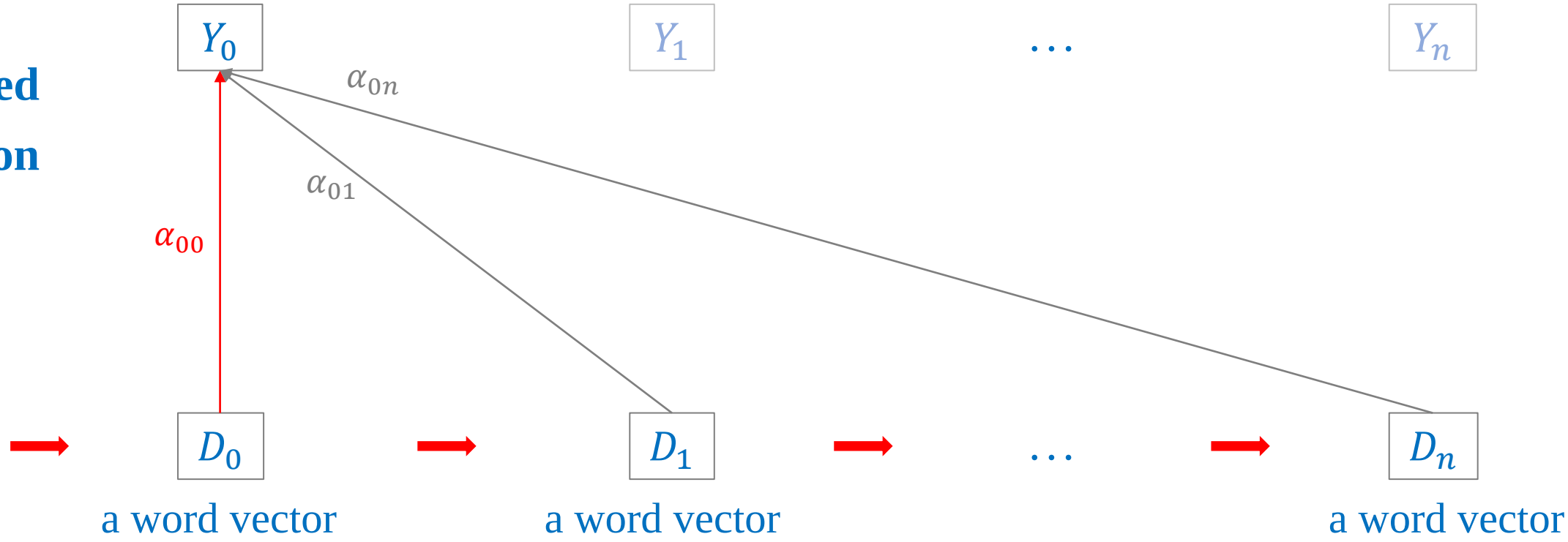


Transformer

❖ Masked self-attention



Masked self-attention



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

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

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

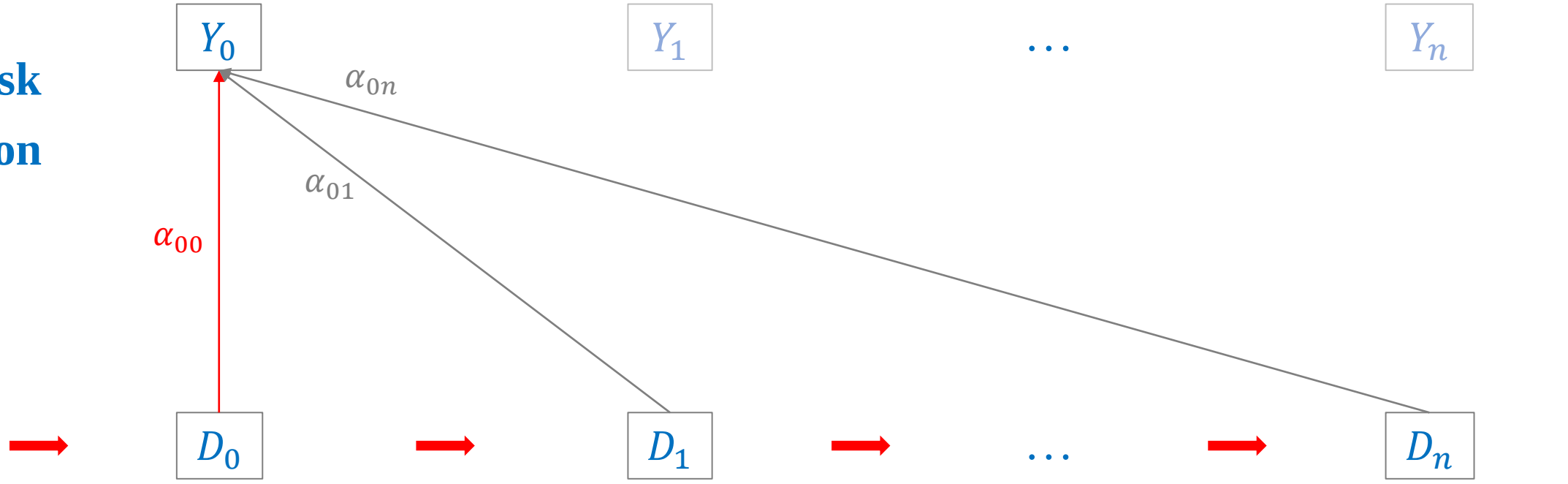
How to obtain kind of →

$$\begin{bmatrix} \alpha_{00} \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

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

?

Mask self-attention



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

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

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

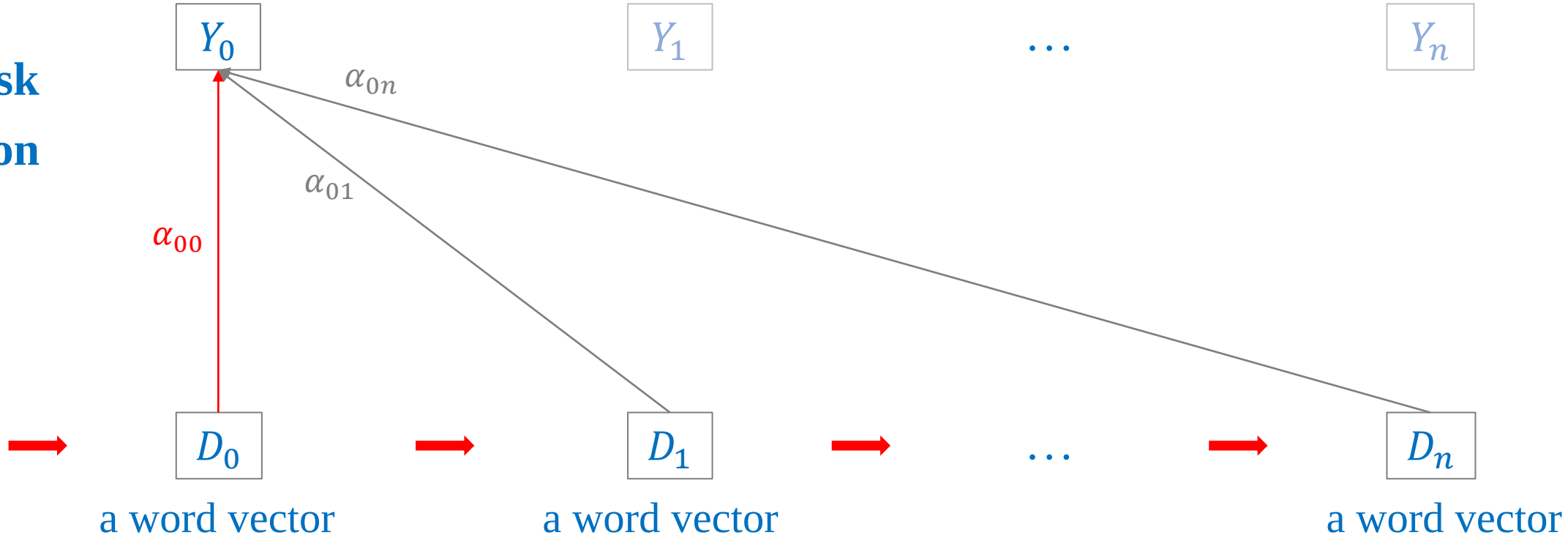
How to obtain kind of →

$$\begin{bmatrix} \alpha_{00} \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

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

?

Mask self-attention



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

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

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

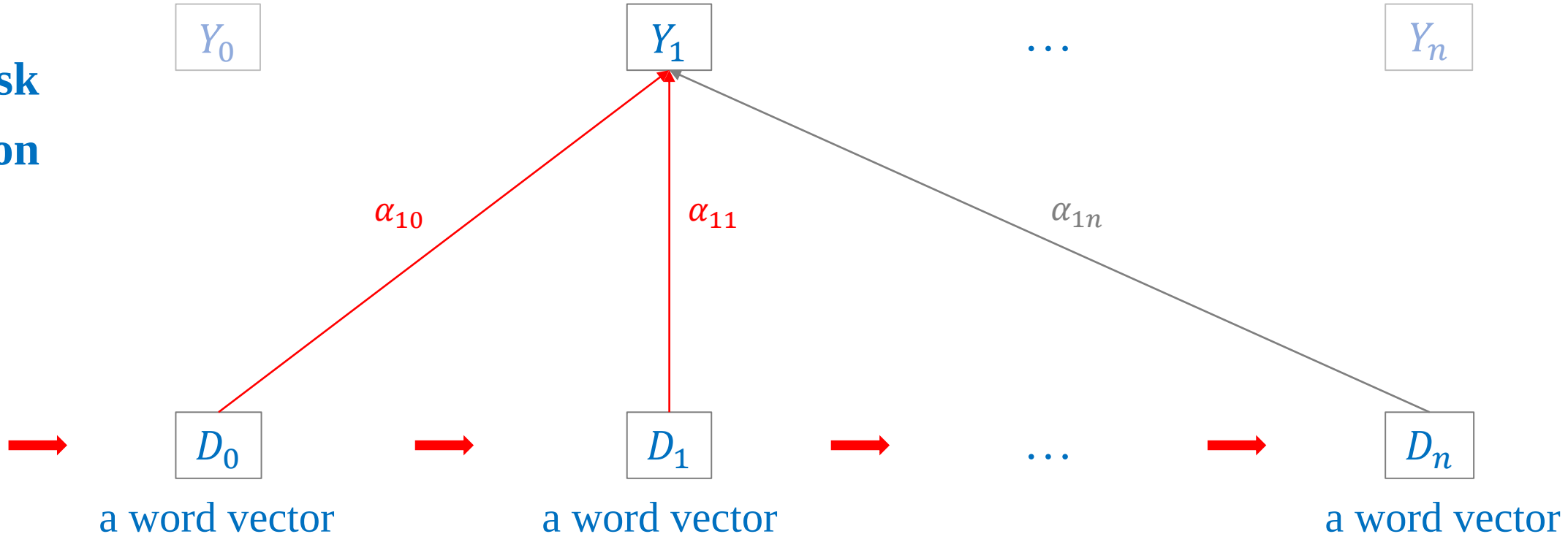
How to obtain kind of →

$$\begin{bmatrix} \alpha_{00} \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

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

?

Mask self-attention



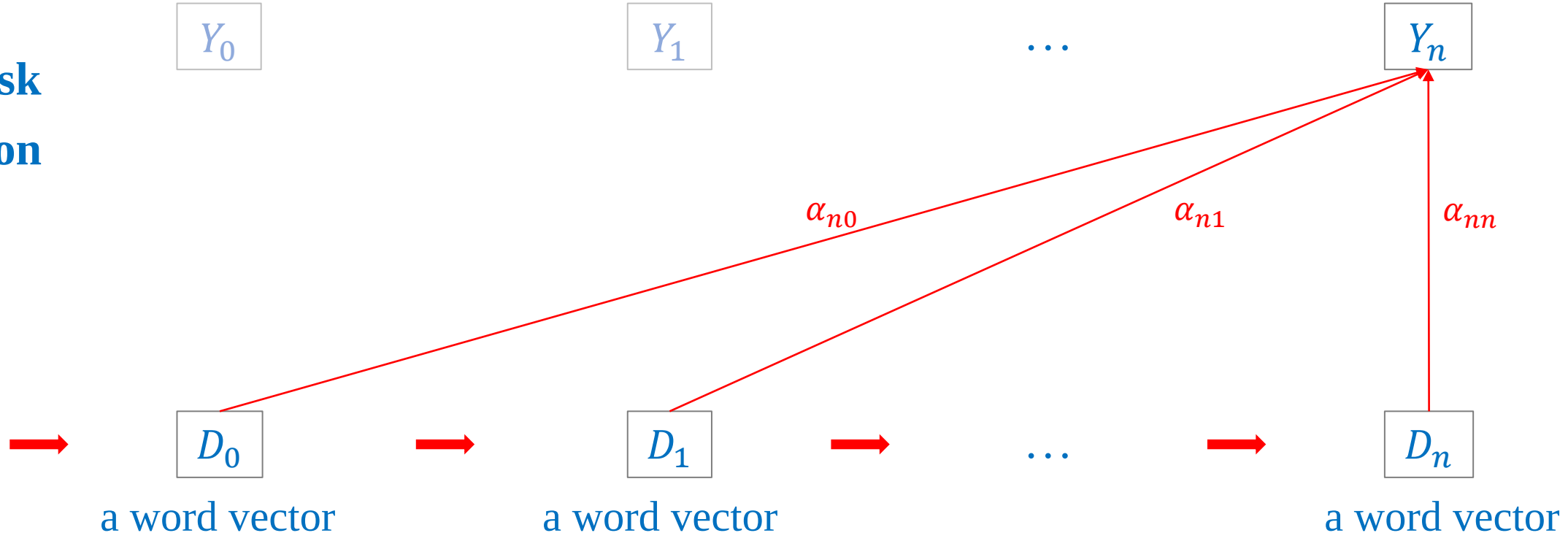
$$Y_1 = \alpha_{10}D_0 + \alpha_{11}D_1 + \cdots + \alpha_{1n}D_n$$

$$\Downarrow$$

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

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

Mask self-attention

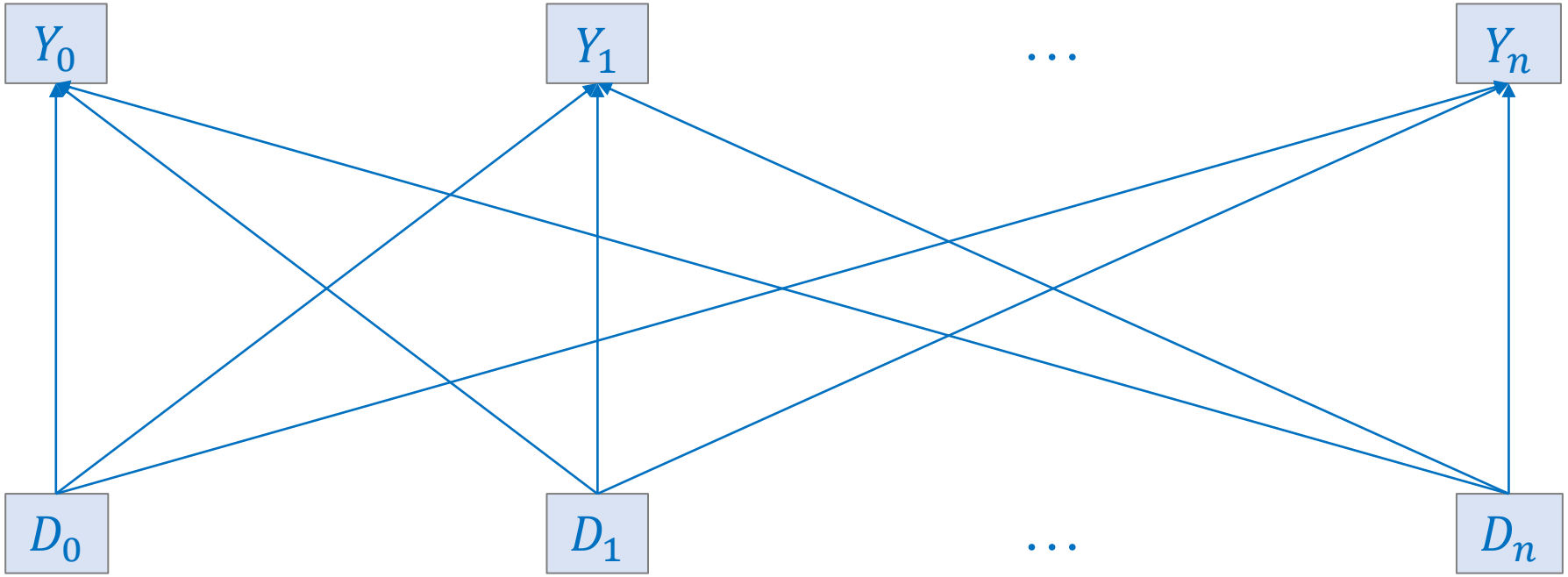


$$Y_n = \alpha_{n0}D_0 + \alpha_{n1}D_1 + \cdots + \alpha_{nn}D_n$$

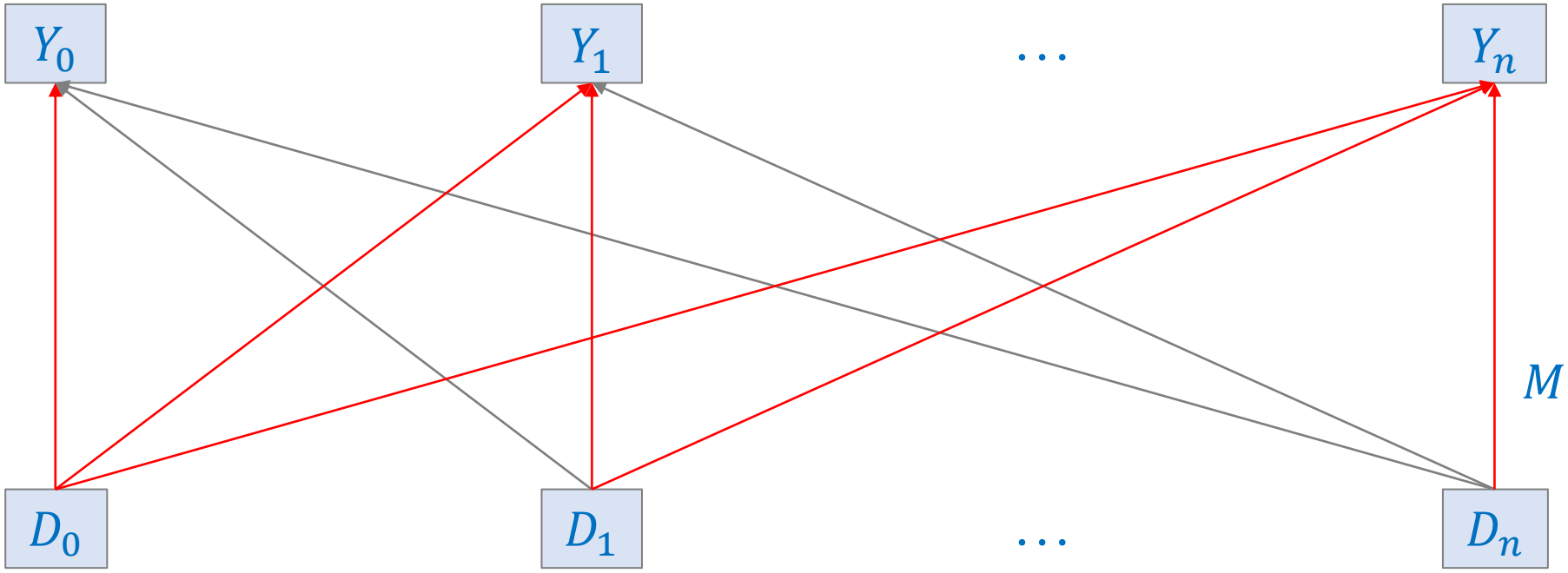
⬇

?

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



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

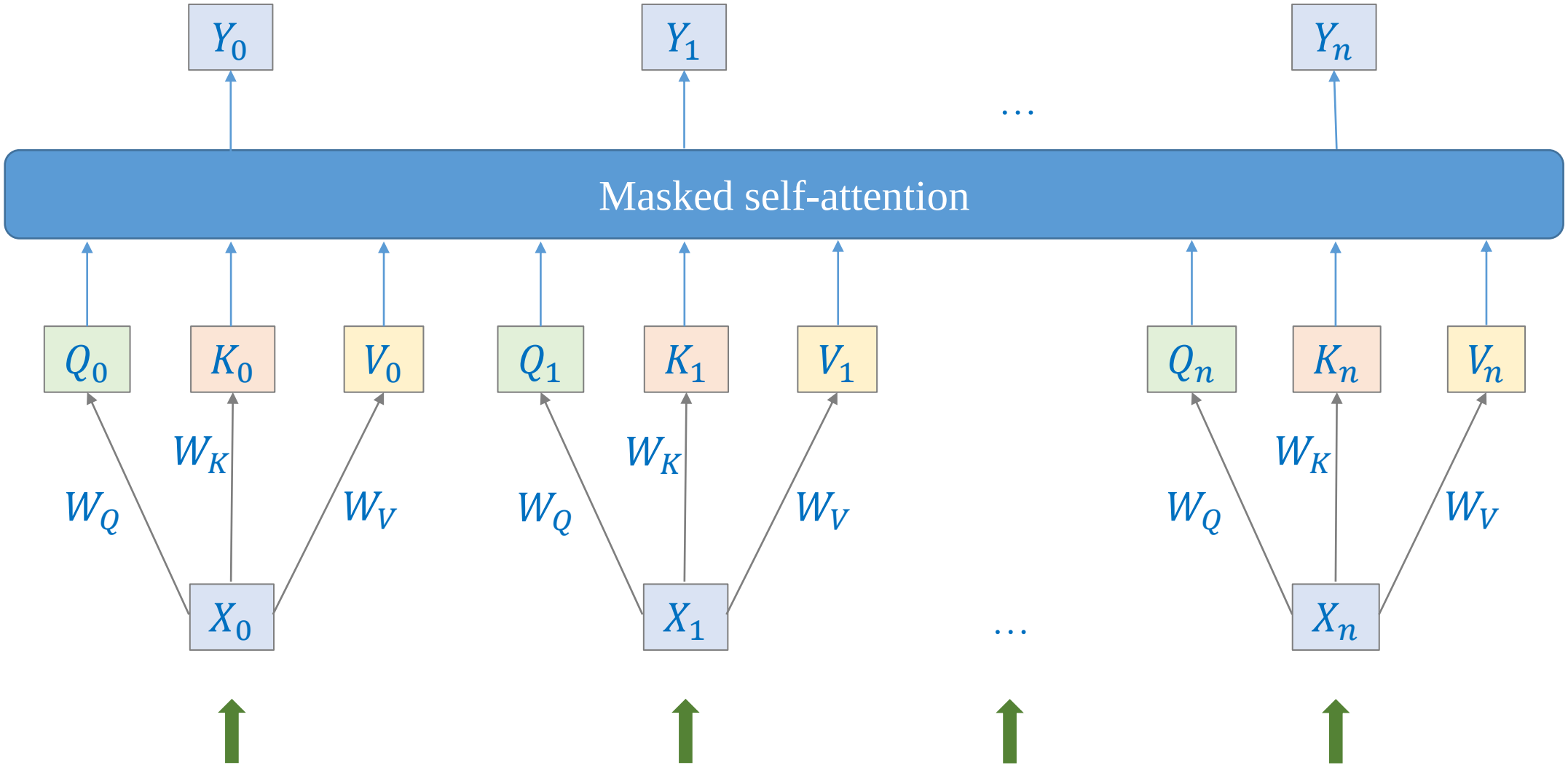


$$\alpha = sigmoid\left(\frac{DD^T}{\sqrt{d}} + M\right)$$

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

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

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



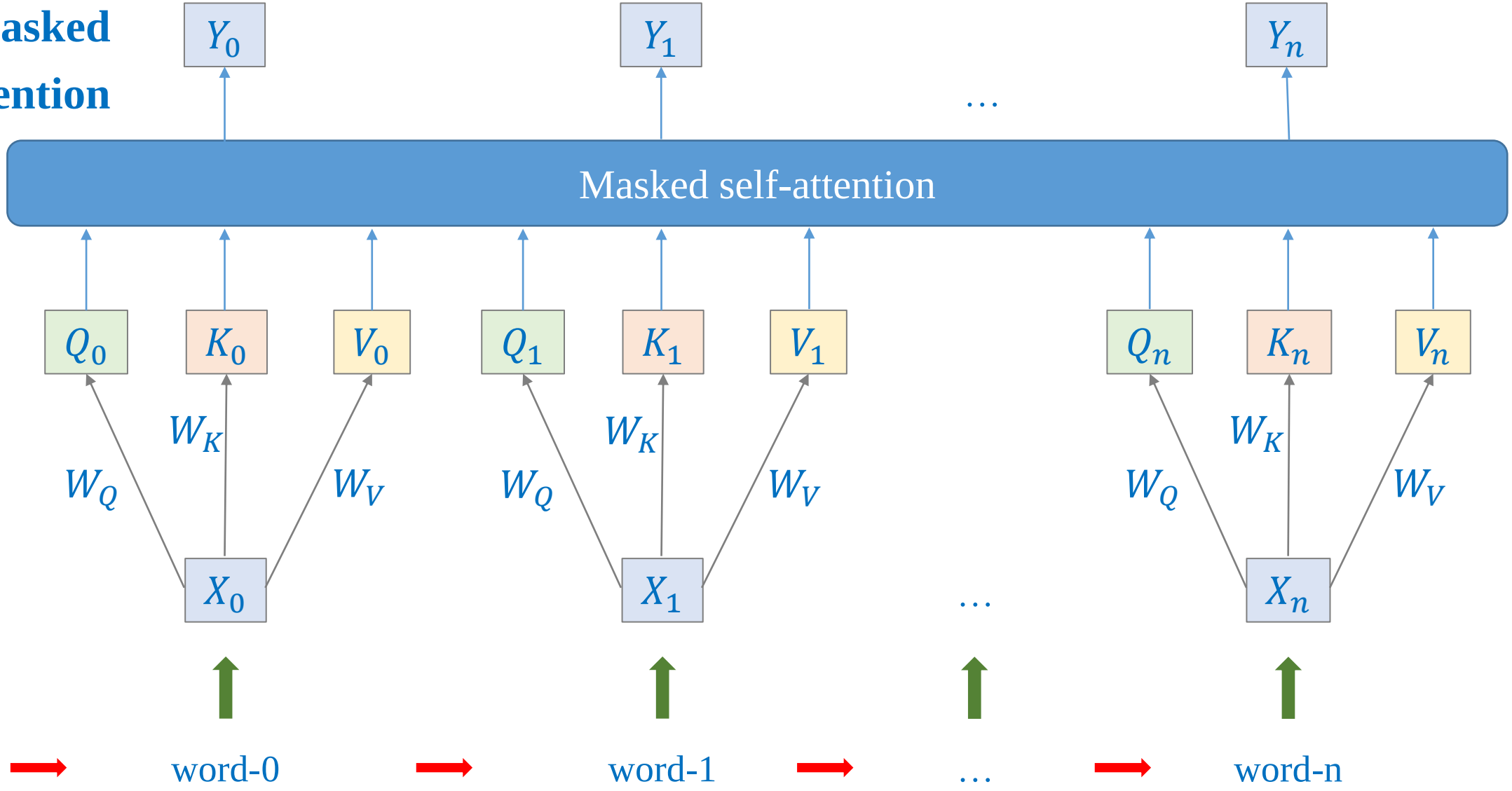
```python
x = torch.tensor([[[-0.1,  0.1,  0.3], [ 0.4, -1.1, -0.3]]])
layer = nn.MultiheadAttention(embed_dim=3, num_heads=1, batch_first=True)

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

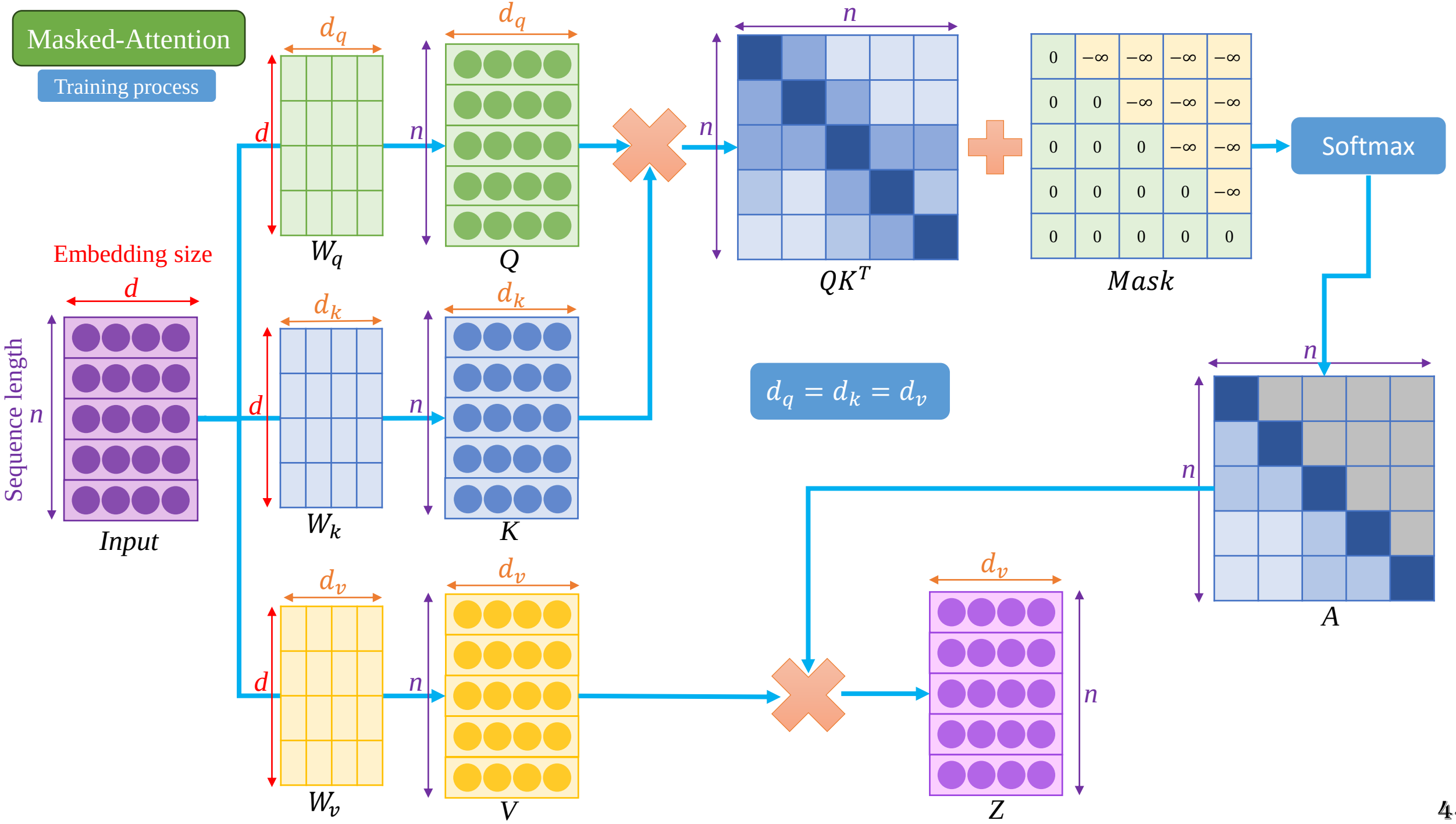
$$A = sigmoid\left(\frac{QK^T}{\sqrt{d}} + M\right)V$$

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

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

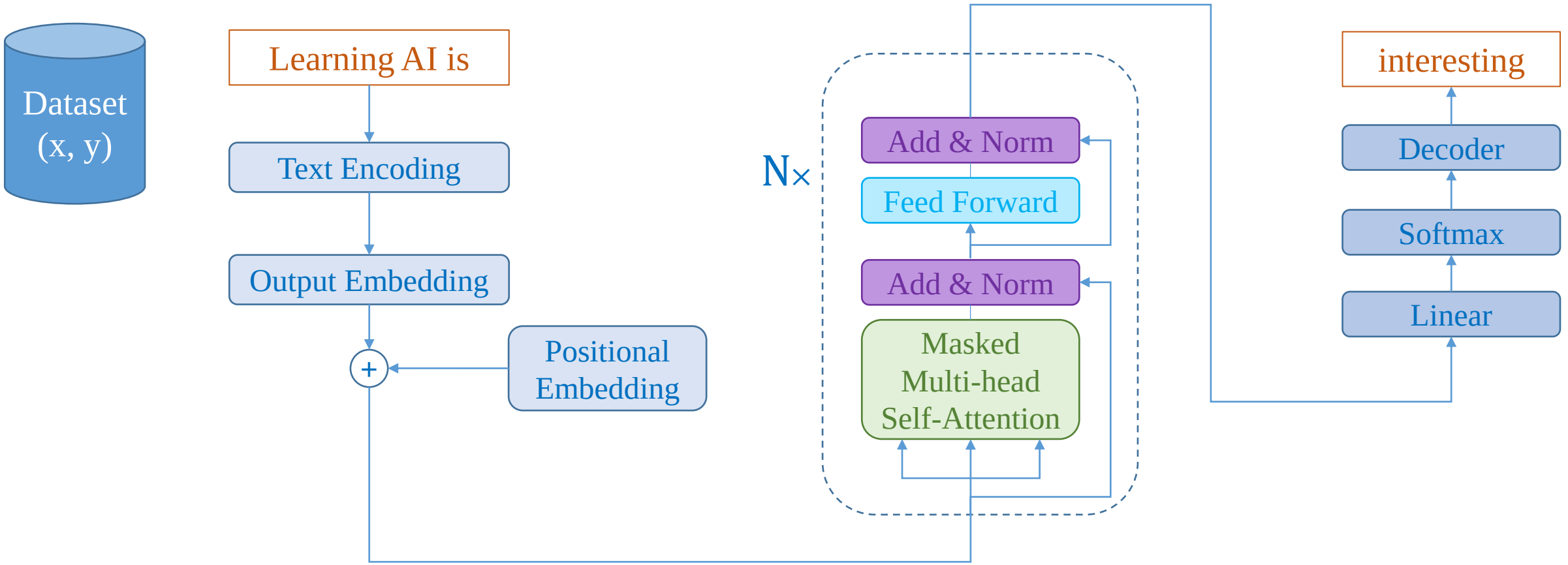
$$= \begin{bmatrix} 1.0 & 0.0 \\ 0.52 & 0.48 \end{bmatrix}\begin{bmatrix} -0.16 & -0.08 & -0.05 \\ -0.02 & -0.02 & 0.05 \end{bmatrix} = \begin{bmatrix} -0.16 & -0.08 & -0.05 \\ 0.12 & 0.08 & 0.06 \end{bmatrix}$$

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


Masked-Attention
Training process
Embedding size
Sequence length
Input
W_q
W_k
W_v
Q
K
V
QK^T
$Mask$
Softmax
A
Z
$d_q = d_k = d_v$

Text Generation

❖ Architecture

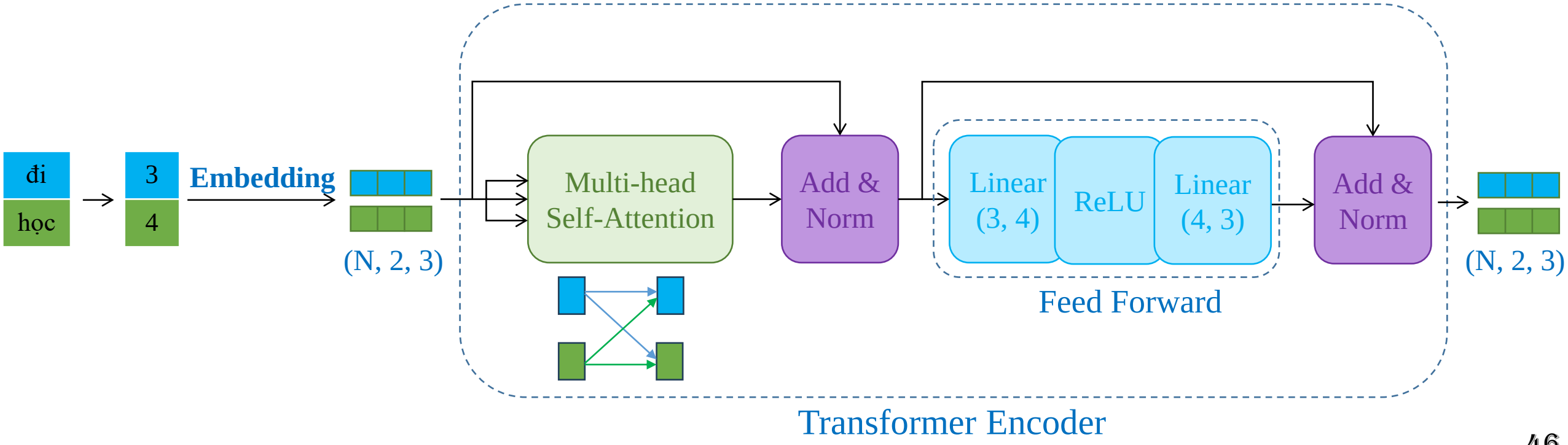


Encoder in PyTorch

Encoder in Pytorch

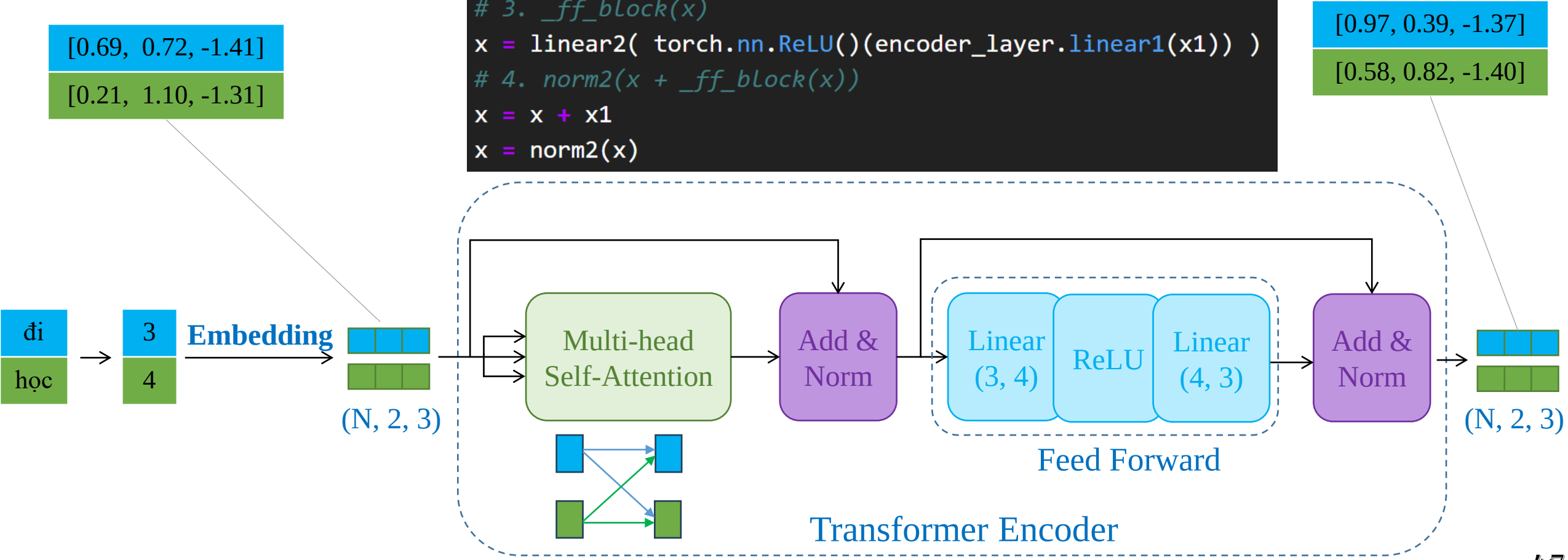
index	word
0	[UNK]
1	[pad]
2	ai
3	đi
4	học
…	…

index	Embedding
0	[-0.188, …, 0.7013]
1	[1.7840…, 1.3586]
2	[1.0281, …, 0.4211]
3	[-1.308, …, -0.3680]
4	[0.2293, …, 2.0501]
…	…



Encoder in Pytorch

```
# input after throught embedding
embed_input = torch.Tensor([[[ 0.69,  0.72, -1.41],
                             [ 0.21,  1.10, -1.31]]])
# 1. self_attn
x = self_attn(embed_input, embed_input, embed_input)[0]
# 2. norm1(x + self._sa_block(x))
x = src + x
x1 = norm1(x)
# 3. _ff_block(x)
x = linear2( torch.nn.ReLU()(encoder_layer.linear1(x1)) )
# 4. norm2(x + _ff_block(x))
x = x + x1
x = norm2(x)
```

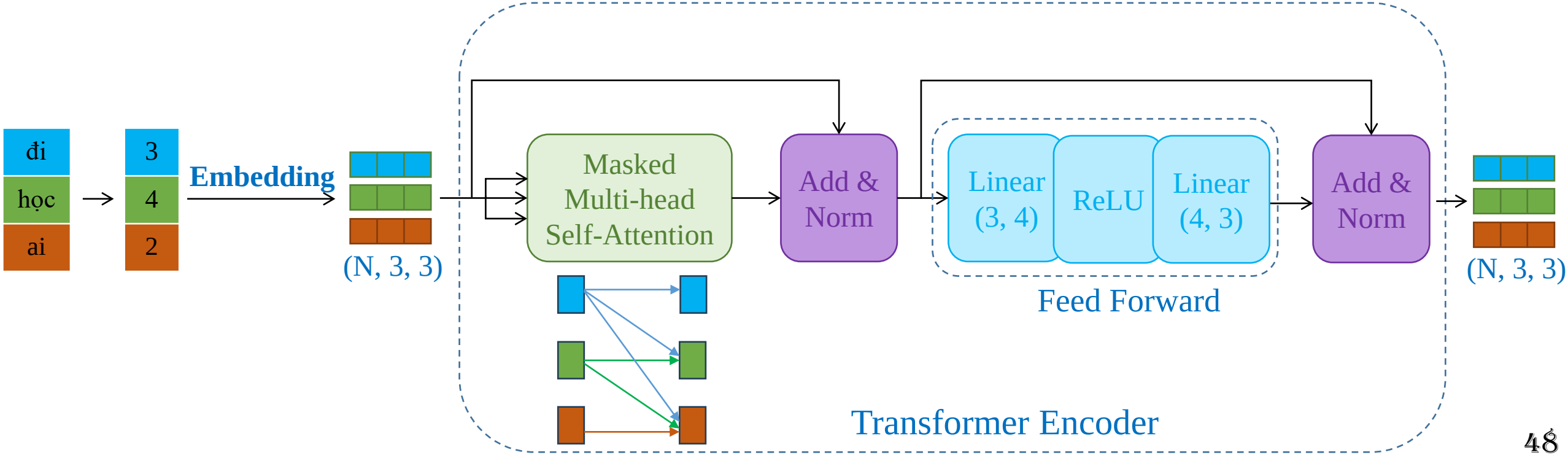


Masked Encoder in PyTorch

Masked Encoder in Pytorch

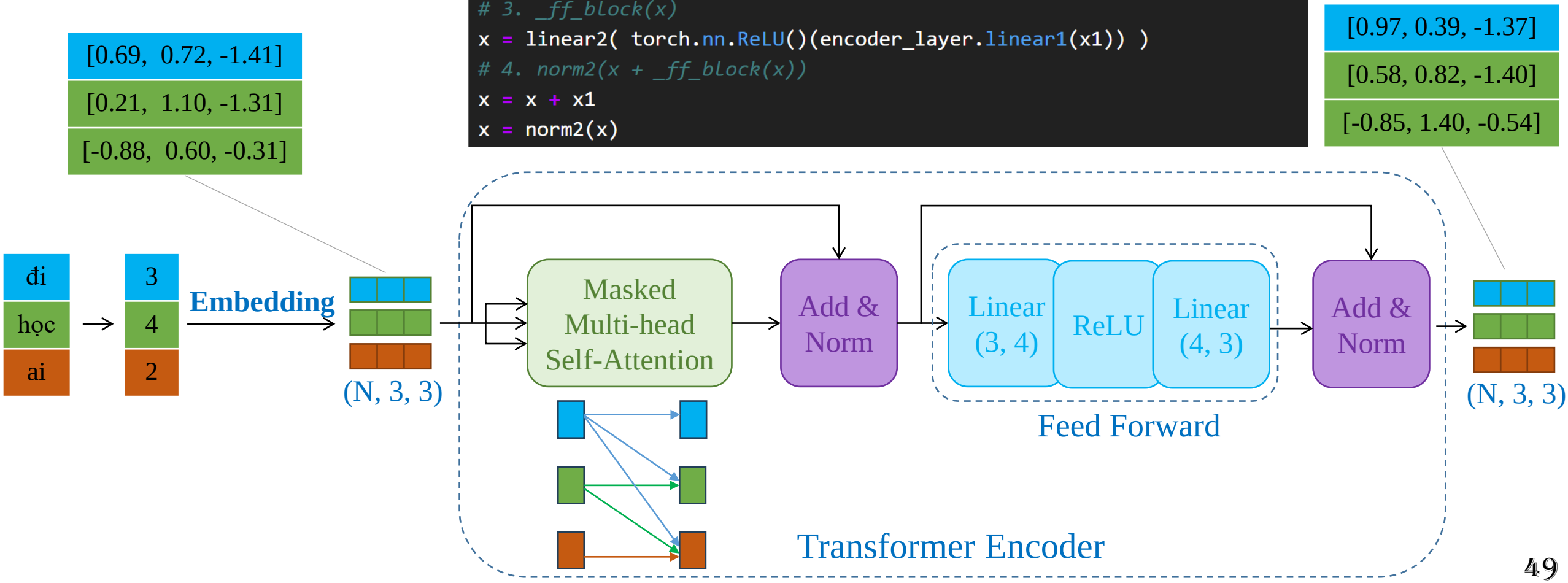
index	word
0	[UNK]
1	[pad]
2	ai
3	đi
4	học
…	…

index	Embedding
0	[-0.188, …, 0.7013]
1	[1.7840…, 1.3586]
2	[1.0281, …, 0.4211]
3	[-1.308, …, -0.3680]
4	[0.2293, …, 2.0501]
…	…



Masked Encoder in Pytorch

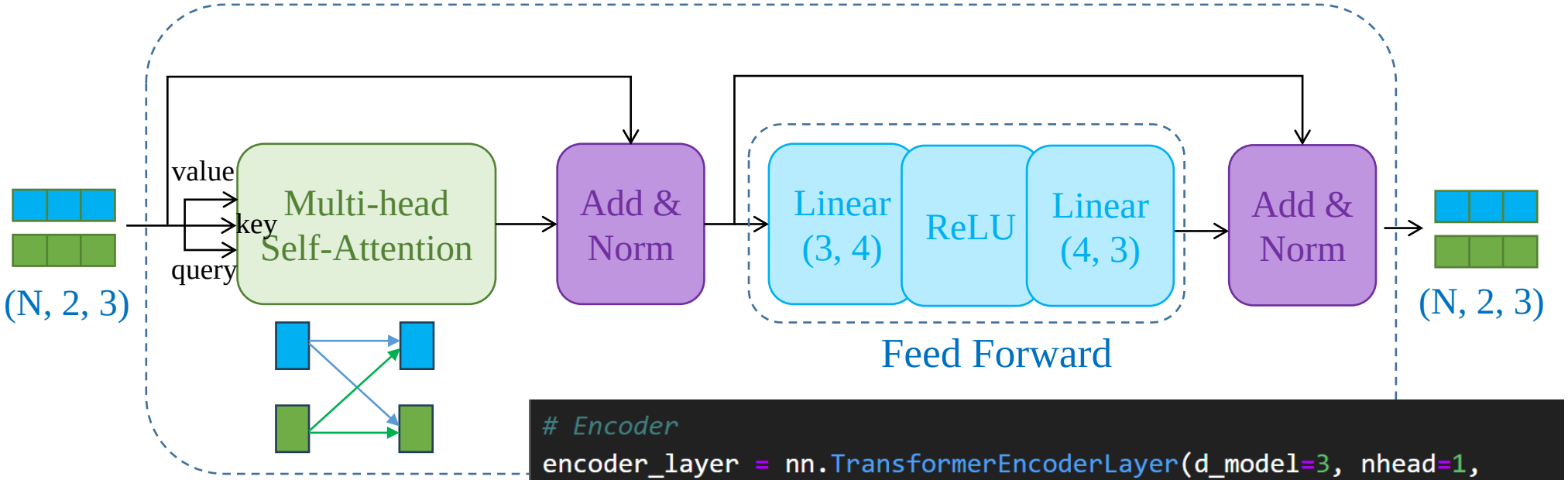
```python
# input after throught embedding
embed_input = torch.Tensor([[[ 0.69,  0.72, -1.41],
                             [ 0.21,  1.10, -1.31]]])
# 1. masked self_attn
mask = torch.triu(torch.ones(seq_len, seq_len), diagonal=1).bool()
x = self_attn(embed_input, embed_input, embed_input,attn_mask=mask)[0]
# 2. norm1(x + self._sa_block(x))
x = src + x
x1 = norm1(x)
# 3. _ff_block(x)
x = linear2( torch.nn.ReLU()(encoder_layer.linear1(x1)) )
# 4. norm2(x + _ff_block(x))
x = x + x1
x = norm2(x)
```



Transformer in PyTorch

Transformer in PyTorch

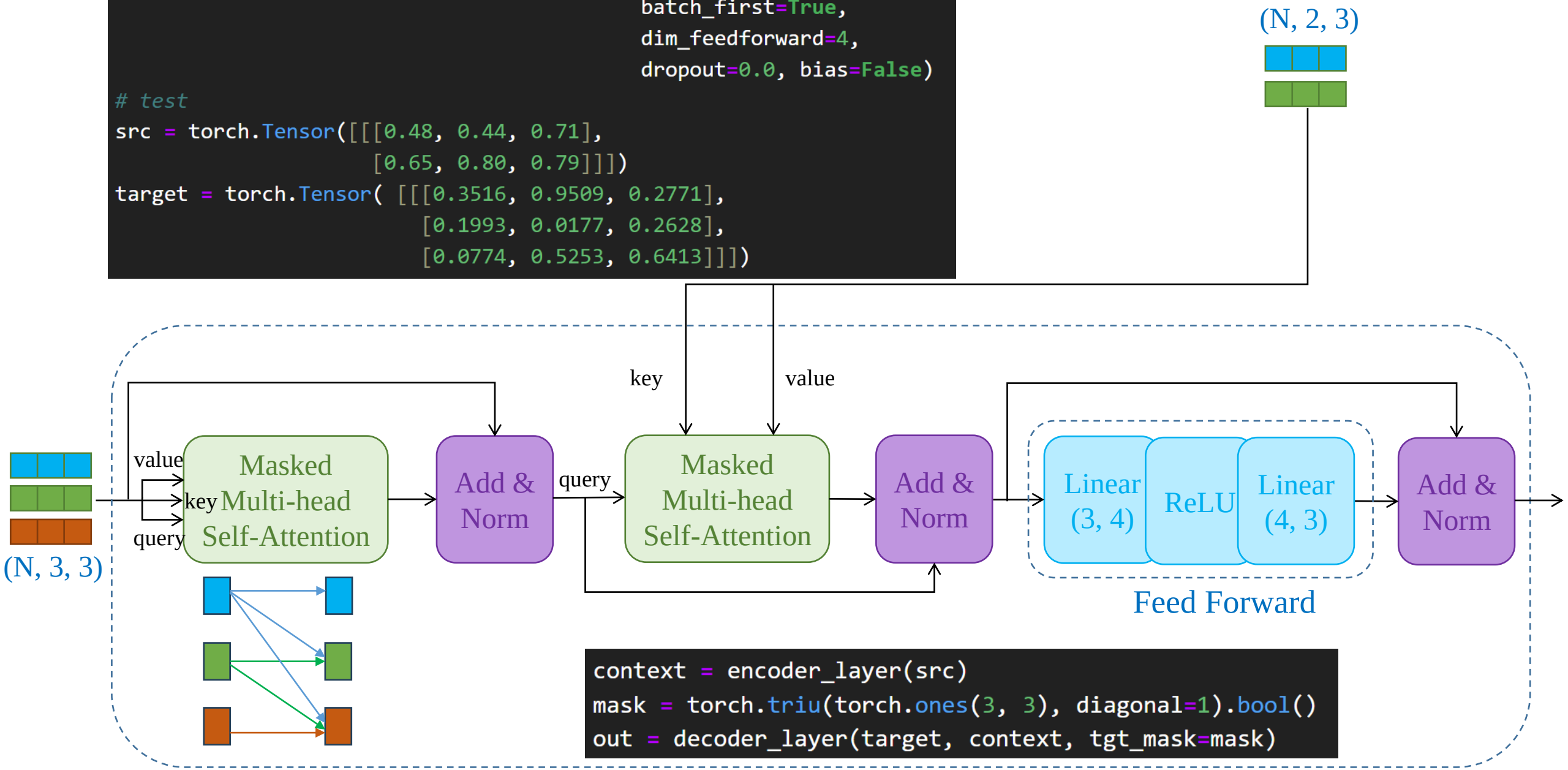
❖ Transformer Encoder



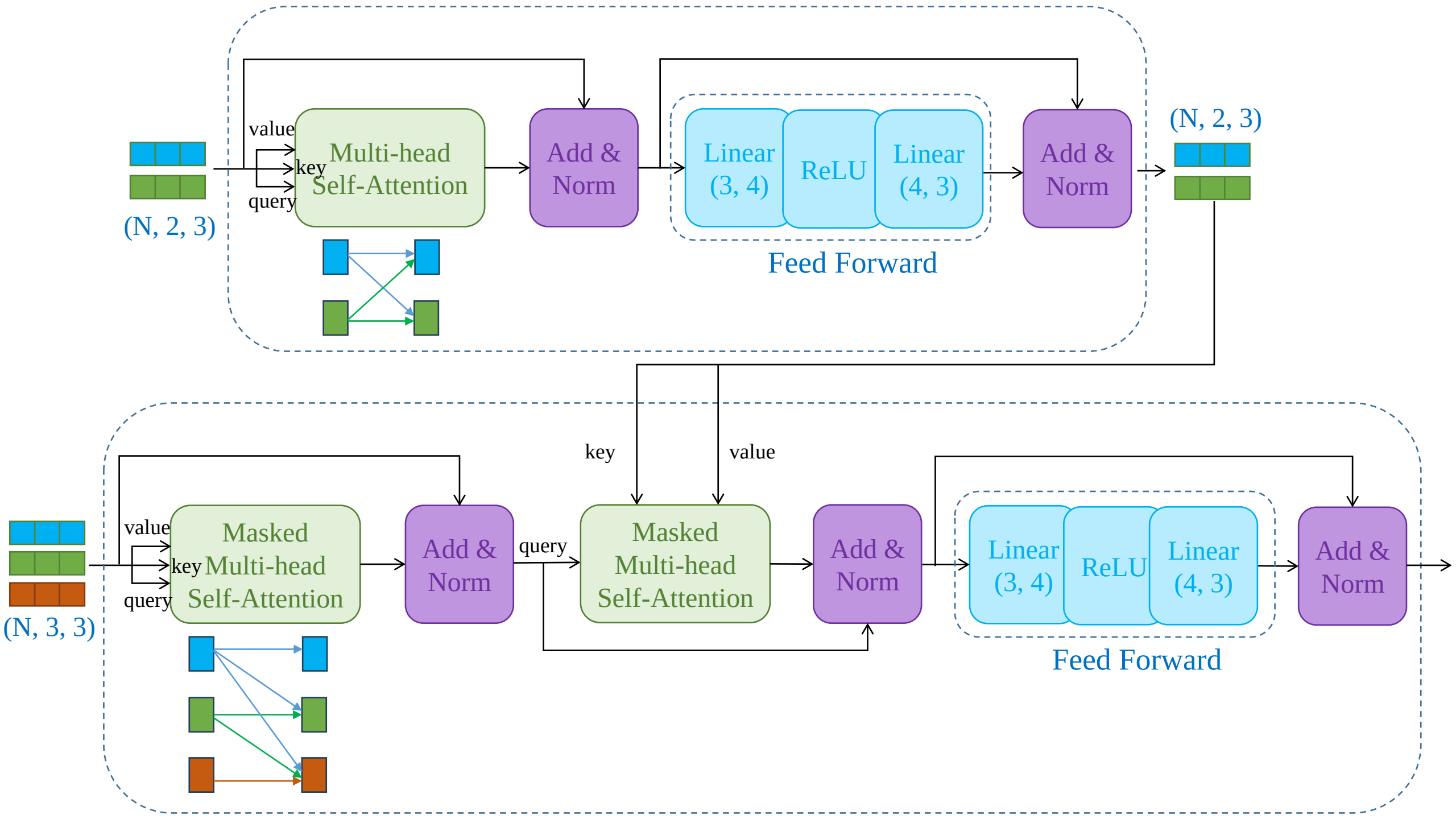

```python
# Encoder
encoder_layer = nn.TransformerEncoderLayer(d_model=3, nhead=1,
                                           batch_first=True,
                                           dim_feedforward=4,
                                           dropout=0.0, bias=False)

# test
src = torch.Tensor([[[0.48, 0.44, 0.71],
                     [0.65, 0.80, 0.79]]])
context = encoder_layer(src)
```

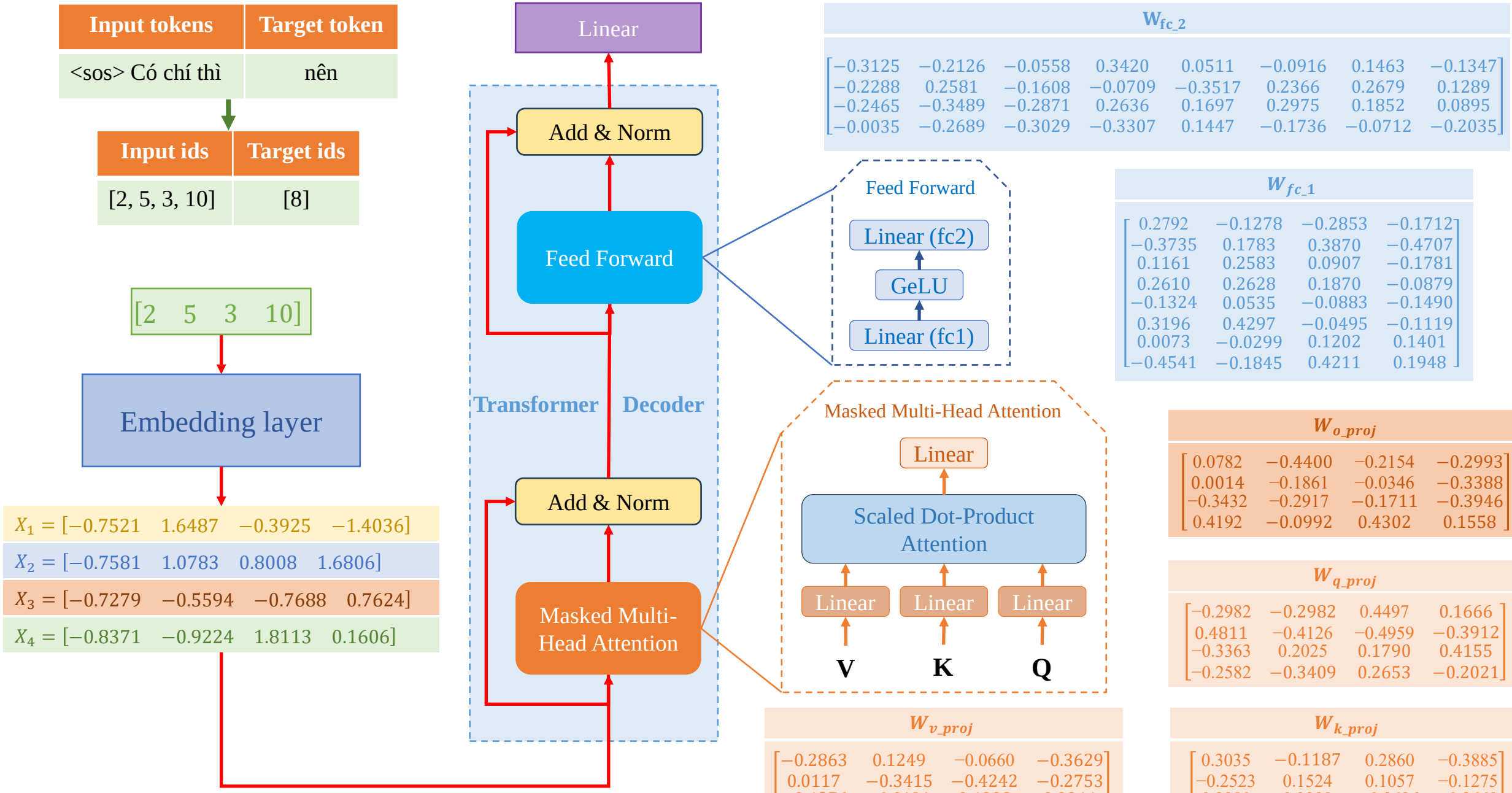
```python
# layers
encoder_layer = nn.TransformerEncoderLayer(...)
decoder_layer = nn.TransformerDecoderLayer(d_model=3, nhead=1,
                                           batch_first=True,
                                           dim_feedforward=4,
                                           dropout=0.0, bias=False)
# test
src = torch.Tensor([[[0.48, 0.44, 0.71],
                     [0.65, 0.80, 0.79]]])
target = torch.Tensor( [[[0.3516, 0.9509, 0.2771],
                         [0.1993, 0.0177, 0.2628],
                         [0.0774, 0.5253, 0.6413]]])
```

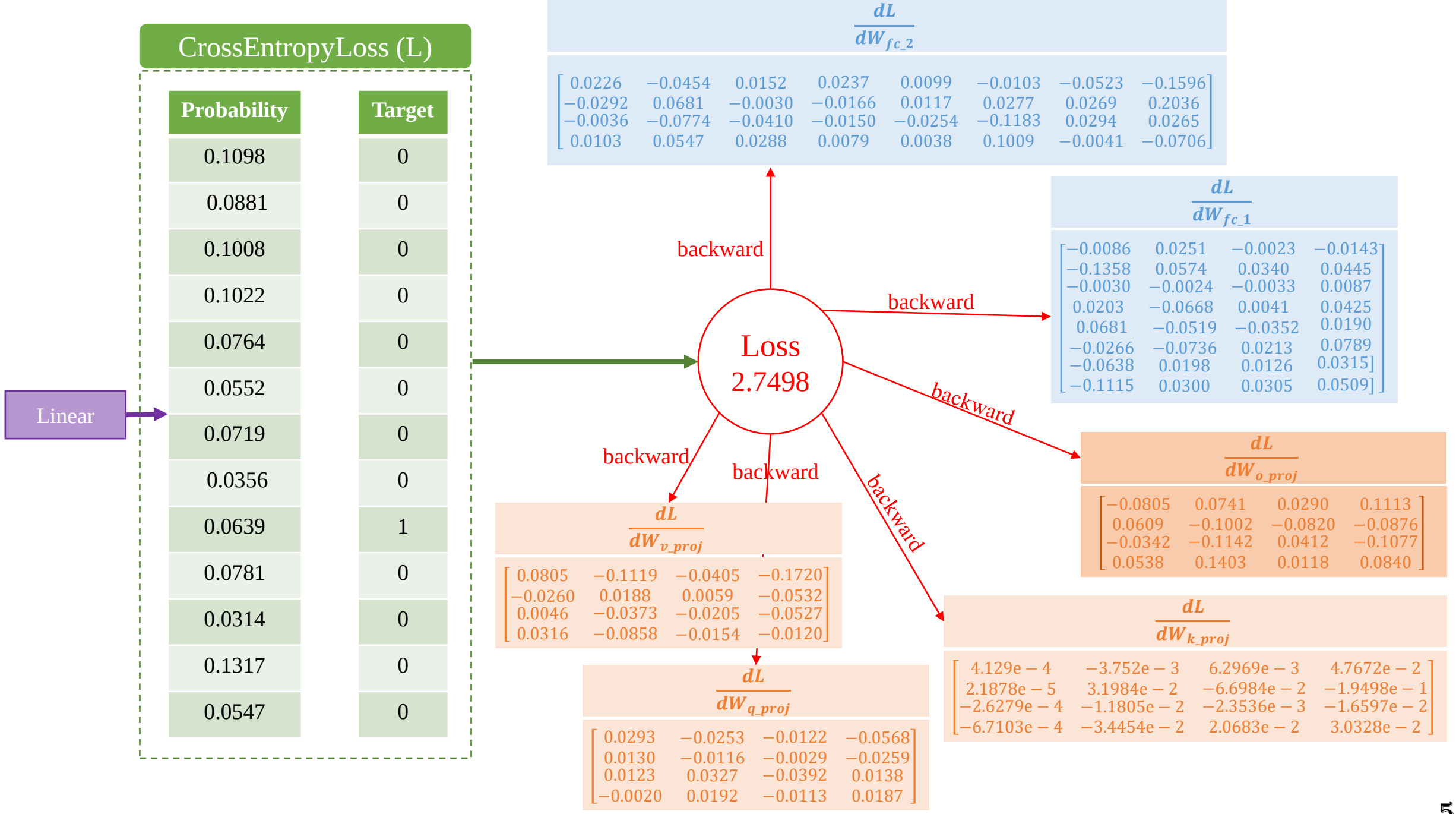



```python
context = encoder_layer(src)
mask = torch.triu(torch.ones(3, 3), diagonal=1).bool()
out = decoder_layer(target, context, tgt_mask=mask)
```


(N, 2, 3)
value
key
query
Multi-head Self-Attention
Add & Norm
Linear (3, 4)
ReLU
Linear (4, 3)
Feed Forward
Add & Norm
(N, 2, 3)
(N, 3, 3)
value
key
query
Masked Multi-head Self-Attention
Add & Norm
key
value
query
Masked Multi-head Self-Attention
Add & Norm
Linear (3, 4)
ReLU
Linear (4, 3)
Feed Forward
Add & Norm

Example


Input tokens
Target token
<sos> Có chí thì
nên
Input ids
Target ids
[2, 5, 3, 10]
[8]
[2 5 3 10]
Embedding layer
$X_1 = [-0.7521 \quad 1.6487 \quad -0.3925 \quad -1.4036]$
$X_2 = [-0.7581 \quad 1.0783 \quad 0.8008 \quad 1.6806]$
$X_3 = [-0.7279 \quad -0.5594 \quad -0.7688 \quad 0.7624]$
$X_4 = [-0.8371 \quad -0.9224 \quad 1.8113 \quad 0.1606]$
Linear
Add & Norm
Feed Forward
Add & Norm
Masked Multi-Head Attention
Transformer Decoder
Feed Forward
Linear (fc2)
GeLU
Linear (fc1)
Masked Multi-Head Attention
Linear
Scaled Dot-Product Attention
Linear
Linear
Linear
V
K
Q
W_{fc_2}
$\begin{bmatrix} -0.3125 & -0.2126 & -0.0558 & 0.3420 & 0.0511 & -0.0916 & 0.1463 & -0.1347 \\ -0.2288 & 0.2581 & -0.1608 & -0.0709 & -0.3517 & 0.2366 & 0.2679 & 0.1289 \\ -0.2465 & -0.3489 & -0.2871 & 0.2636 & 0.1697 & 0.2975 & 0.1852 & 0.0895 \\ -0.0035 & -0.2689 & -0.3029 & -0.3307 & 0.1447 & -0.1736 & -0.0712 & -0.2035 \end{bmatrix}$
W_{fc_1}
$\begin{bmatrix} 0.2792 & -0.1278 & -0.2853 & -0.1712 \\ -0.3735 & 0.1783 & 0.3870 & -0.4707 \\ 0.1161 & 0.2583 & 0.0907 & -0.1781 \\ 0.2610 & 0.2628 & 0.1870 & -0.0879 \\ -0.1324 & 0.0535 & -0.0883 & -0.1490 \\ 0.3196 & 0.4297 & -0.0495 & -0.1119 \\ 0.0073 & -0.0299 & 0.1202 & 0.1401 \\ -0.4541 & -0.1845 & 0.4211 & 0.1948 \end{bmatrix}$
W_{o_proj}
$\begin{bmatrix} 0.0782 & -0.4400 & -0.2154 & -0.2993 \\ 0.0014 & -0.1861 & -0.0346 & -0.3388 \\ -0.3432 & -0.2917 & -0.1711 & -0.3946 \\ 0.4192 & -0.0992 & 0.4302 & 0.1558 \end{bmatrix}$
W_{q_proj}
$\begin{bmatrix} -0.2982 & -0.2982 & 0.4497 & 0.1666 \\ 0.4811 & -0.4126 & -0.4959 & -0.3912 \\ -0.3363 & 0.2025 & 0.1790 & 0.4155 \\ -0.2582 & -0.3409 & 0.2653 & -0.2021 \end{bmatrix}$
W_{v_proj}
$\begin{bmatrix} -0.2863 & 0.1249 & -0.0660 & -0.3629 \\ 0.0117 & -0.3415 & -0.4242 & -0.2753 \\ -0.4376 & -0.3184 & 0.4998 & 0.0944 \\ 0.1541 & -0.4663 & -0.3284 & -0.1664 \end{bmatrix}$
W_{k_proj}
$\begin{bmatrix} 0.3035 & -0.1187 & 0.2860 & -0.3885 \\ -0.2523 & 0.1524 & 0.1057 & -0.1275 \\ 0.2980 & 0.3399 & -0.3626 & -0.2669 \\ 0.4578 & -0.1687 & -0.1773 & -0.4838 \end{bmatrix}$


CrossEntropyLoss (L)
Probability
Target
0.1098 0
0.0881 0
0.1008 0
0.1022 0
0.0764 0
0.0552 0
0.0719 0
0.0356 0
0.0639 1
0.0781 0
0.0314 0
0.1317 0
0.0547 0
Linear
Loss
2.7498
backward
backward
backward
backward
backward
backward
$\frac{dL}{dW_{fc\_2}}$
$\begin{bmatrix} 0.0226 & -0.0454 & 0.0152 & 0.0237 & 0.0099 & -0.0103 & -0.0523 & -0.1596 \\ -0.0292 & 0.0681 & -0.0030 & -0.0166 & 0.0117 & 0.0277 & 0.0269 & 0.2036 \\ -0.0036 & -0.0774 & -0.0410 & -0.0150 & -0.0254 & -0.1183 & 0.0294 & 0.0265 \\ 0.0103 & 0.0547 & 0.0288 & 0.0079 & 0.0038 & 0.1009 & -0.0041 & -0.0706 \end{bmatrix}$
$\frac{dL}{dW_{fc\_1}}$
$\begin{bmatrix} -0.0086 & 0.0251 & -0.0023 & -0.0143 \\ -0.1358 & 0.0574 & 0.0340 & 0.0445 \\ -0.0030 & -0.0024 & -0.0033 & 0.0087 \\ 0.0203 & -0.0668 & 0.0041 & 0.0425 \\ 0.0681 & -0.0519 & -0.0352 & 0.0190 \\ -0.0266 & -0.0736 & 0.0213 & 0.0789 \\ -0.0638 & 0.0198 & 0.0126 & 0.0315] \\ -0.1115 & 0.0300 & 0.0305 & 0.0509] \end{bmatrix}$
$\frac{dL}{dW_{o\_proj}}$
$\begin{bmatrix} -0.0805 & 0.0741 & 0.0290 & 0.1113 \\ 0.0609 & -0.1002 & -0.0820 & -0.0876 \\ -0.0342 & -0.1142 & 0.0412 & -0.1077 \\ 0.0538 & 0.1403 & 0.0118 & 0.0840 \end{bmatrix}$
$\frac{dL}{dW_{v\_proj}}$
$\begin{bmatrix} 0.0805 & -0.1119 & -0.0405 & -0.1720 \\ -0.0260 & 0.0188 & 0.0059 & -0.0532 \\ 0.0046 & -0.0373 & -0.0205 & -0.0527 \\ 0.0316 & -0.0858 & -0.0154 & -0.0120 \end{bmatrix}$
$\frac{dL}{dW_{k\_proj}}$
$\begin{bmatrix} 4.129e-4 & -3.752e-3 & 6.2969e-3 & 4.7672e-2 \\ 2.1878e-5 & 3.1984e-2 & -6.6984e-2 & -1.9498e-1 \\ -2.6279e-4 & -1.1805e-2 & -2.3536e-3 & -1.6597e-2 \\ -6.7103e-4 & -3.4454e-2 & 2.0683e-2 & 3.0328e-2 \end{bmatrix}$
$\frac{dL}{dW_{q\_proj}}$
$\begin{bmatrix} 0.0293 & -0.0253 & -0.0122 & -0.0568 \\ 0.0130 & -0.0116 & -0.0029 & -0.0259 \\ 0.0123 & 0.0327 & -0.0392 & 0.0138 \\ -0.0020 & 0.0192 & -0.0113 & 0.0187 \end{bmatrix}$

Update Parameters
SGD($lr = 0.1$)

$$W_{fc\_2} = W_{fc\_2} - lr \times dW_{fc\_2}$$

$$\begin{bmatrix} -0.3148 & -0.2081 & -0.0573 & 0.3396 & 0.0502 & -0.0905 & -0.1515 & -0.1187 \\ -0.2259 & 0.2512 & -0.1605 & -0.0692 & -0.3529 & 0.2339 & 0.2652 & 0.1085 \\ -0.2462 & -0.3412 & -0.2830 & 0.2651 & 0.1723 & 0.3093 & 0.1823 & 0.0868 \\ -0.0045 & -0.2743 & -0.3058 & -0.3315 & 0.1443 & -0.1837 & -0.0707 & -0.1964 \end{bmatrix}$$

$$W_{fc\_1} = W_{fc\_1} - lr \times dW_{fc\_1}$$

$$\begin{bmatrix} 0.2800 & -0.1303 & -0.2851 & -0.1698 \\ -0.3600 & 0.1726 & 0.3836 & -0.4752 \\ 0.1164 & 0.2585 & 0.0910 & -0.1789 \\ 0.2589 & 0.2694 & 0.1866 & -0.0921 \\ -0.1392 & 0.0587 & -0.0848 & -0.1509 \\ 0.3223 & 0.4371 & -0.0516 & -0.1198 \\ 0.0137 & -0.0318 & 0.1189 & 0.1370 \\ -0.4430 & -0.1875 & 0.4180 & 0.1897 \end{bmatrix}$$

$$W_{o\_proj} = W_{o\_proj} - lr \times dW_{o\_proj}$$

$$\begin{bmatrix} 0.0862 & -0.4474 & -0.2183 & -0.3105 \\ -0.0047 & -0.1760 & -0.0264 & -0.3301 \\ -0.3398 & -0.2803 & -0.1753 & -0.3839 \\ 0.4139 & -0.1133 & 0.4290 & 0.1474 \end{bmatrix}$$

$$W_{q\_proj} = W_{q\_proj} - lr \times dW_{q\_proj}$$

$$\begin{bmatrix} -0.3011 & -0.2957 & 0.4509 & 0.1723 \\ 0.4798 & -0.4115 & -0.4956 & -0.3886 \\ -0.3376 & 0.1992 & 0.1830 & 0.4141 \\ -0.2580 & -0.3428 & 0.2664 & -0.2040 \end{bmatrix}$$

$$W_{k\_proj} = W_{k\_proj} - lr \times dW_{k\_proj}$$

$$\begin{bmatrix} 0.3034 & -0.1183 & 0.2854 & -0.3933 \\ -0.2523 & 0.1492 & 0.1124 & -0.1080 \\ 0.2981 & 0.3411 & -0.3624 & -0.2653 \\ 0.4579 & -0.1653 & -0.1793 & -0.4868 \end{bmatrix}$$

$$W_{v\_proj} = W_{v\_proj} - lr \times dW_{v\_proj}$$

$$\begin{bmatrix} -0.2944 & 0.1361 & -0.0619 & -0.3457 \\ 0.0143 & -0.3434 & -0.4248 & -0.2700 \\ -0.4381 & -0.3146 & 0.5019 & 0.0997 \\ 0.1509 & -0.4578 & -0.3268 & -0.1652 \end{bmatrix}$$

Forward again

CrossEntropyLoss (L)

Probability	Target
0.0503	0
0.0227	0
0.0684	0
0.0322	0
0.0225	0
0.0170	0
0.0445	0
0.0075	0
0.6174	1
0.0274	0
0.0108	0
0.0522	0
0.0270	0

Loss

0.1699

Precision and Recall of Words

Predict/Candidate/Output: Tôi học NLP của AI VIET NAM

Reference: Tôi đang học lớp AI của AI VIET NAM

Precision 1-gram

$$\frac{\text{correct}}{\text{candidate\_length}} = \frac{6}{7}$$

Recall 1-gram

$$\frac{\text{correct}}{\text{reference\_length}} = \frac{6}{9}$$

F1-score 1-gram

$$\frac{\text{precision} \times \text{recall}}{(\text{precision} + \text{recall})/2} = 0.75$$

BLEU Score

❖ BLEU score

Precision 1-gram

$$\frac{\text{correct}}{\text{candidate\_length}} = \frac{6}{7}$$

Recall 1-gram

$$\frac{\text{correct}}{\text{reference\_length}} = \frac{6}{9}$$

N-gram overlap between machine translation candidate and reference translation

Compute precision for n-grams of size 1 to 4

With 4-gram and add brevity penalty

(for too short translations):

$$\text{BLEU} = \min\left(1, \frac{\text{candidate\_length}}{\text{reference\_length}}\right)\left(\prod_{i=1}^{4} \text{Precision}_i\right)^{1/4}$$

Precision and Recall of Words

Predict/Candidate/Output: Tôi học NLP của AI VIET NAM

Reference: Tôi đang học lớp CV và NLP của AI

Precision	1-gram	2-gram	3-gram	4-gram
	6/7	3/6	2/5	1/4

Multiple reference: N-grams may match in any of the reference and closest reference length used

Brevity penalty = 7/9

BLEU = 0.35

$$\text{BLEU} = \min\left(1, \frac{\text{candidate\_length}}{\text{reference\_length}}\right)\left(\prod_{i=1}^{4} \text{Precision}_i\right)^{1/4}$$