AI VIET NAM
@aivietnam.edu.vn

Deep Architectures for POS Tagging and NER

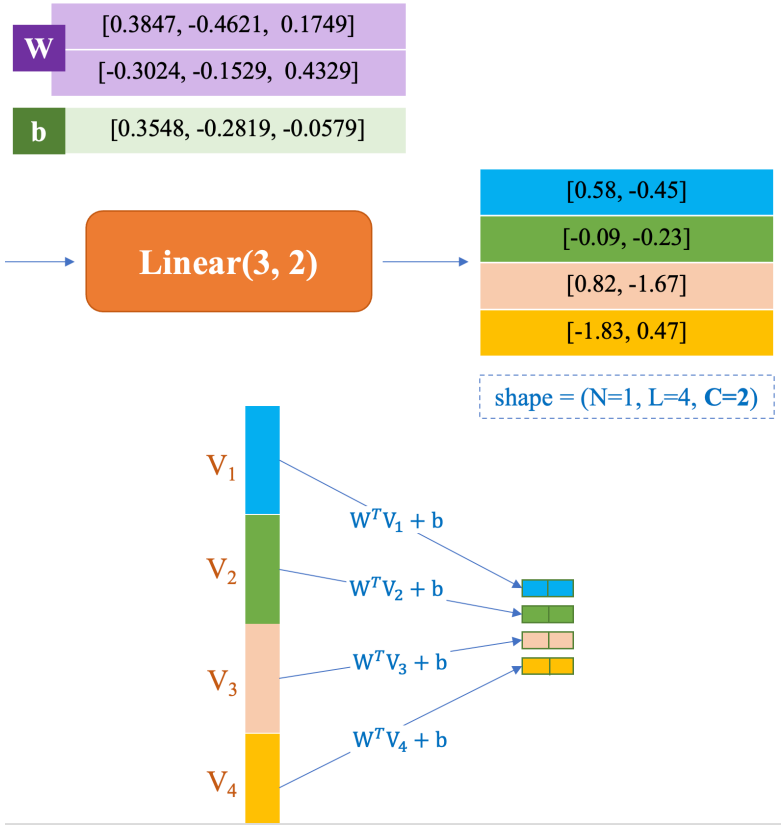
Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2025

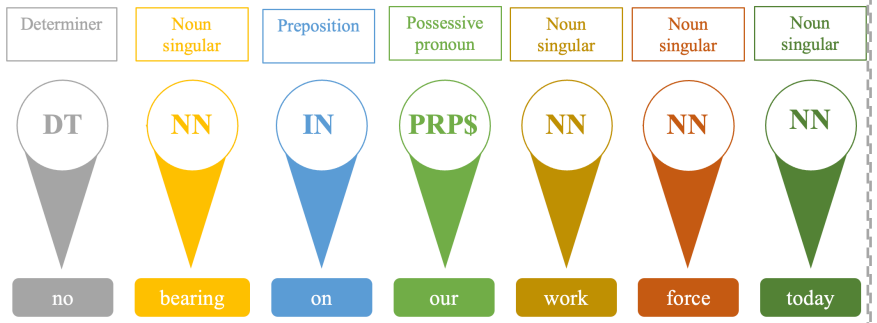
code&data

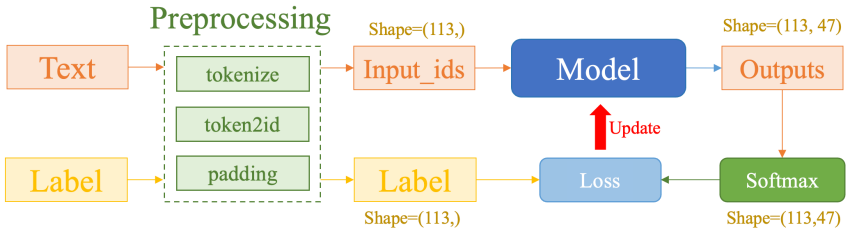
Objectives

Review

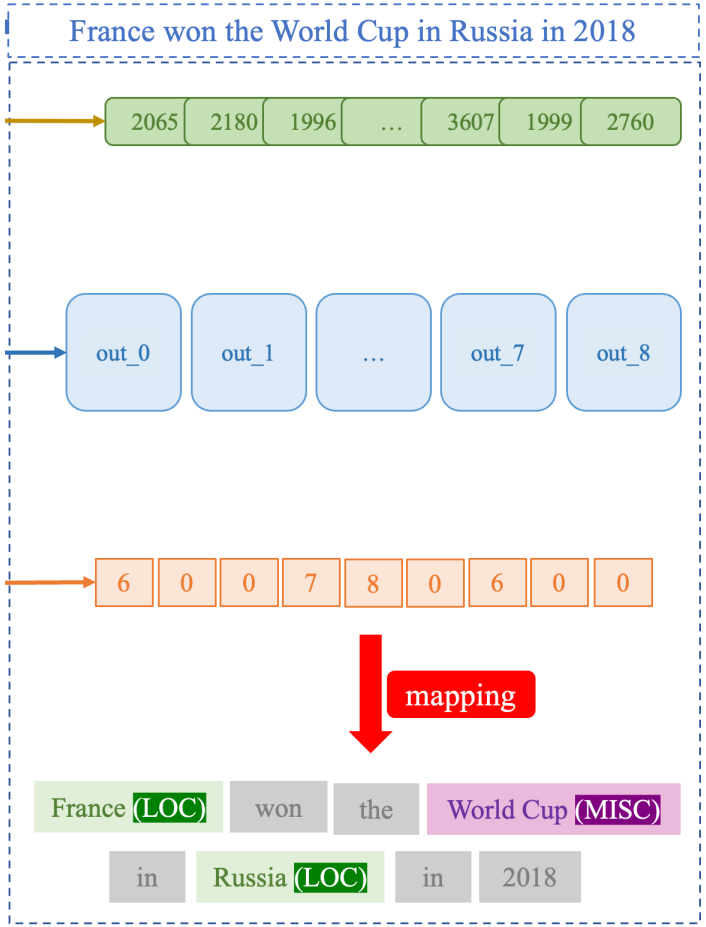


POS Tagging





Named Entity Recognition



Outline

SECTION 1

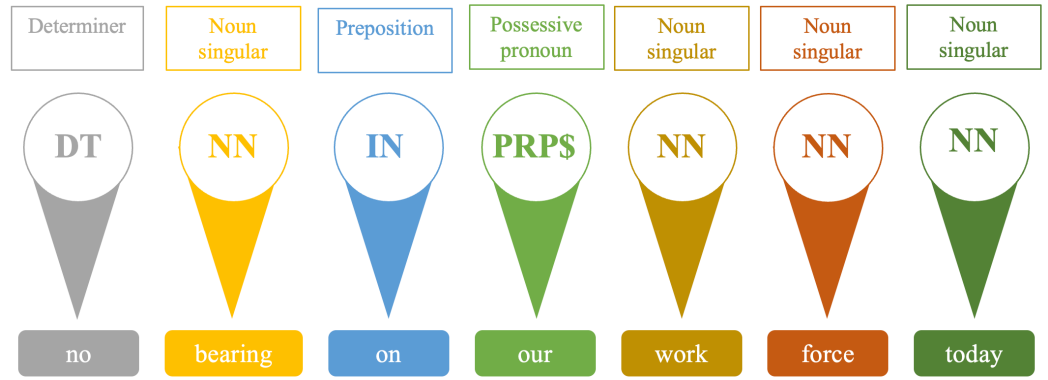
Review

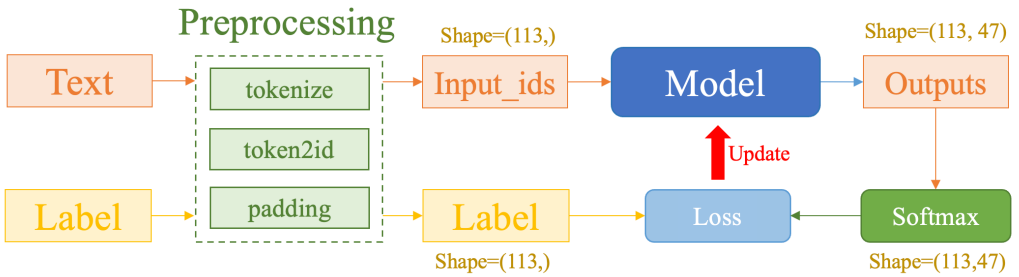
SECTION 2

POS Tagging

SECTION 3

Named Entity Recognition





Cross Entropy Loss

N_classes = 3

shape = (N, C, L)

Label shape = (N, L)

$$L = -\sum_i y_i log(\hat{y}_i)$$

	Logits	Softmax	Probabilities	One-hot label	Loss
Word 0	0.2 0.1 1.2		0.2163 0.1957 0.588	0 0 1	$-log(0.588) = 0.531$
Word 1	3.6 -2.5 -0.2		0.9776 0.0022 0.0218	1 0 0	$-log(0.9776) = 0.2429$

$$\mathcal{L}oss = \frac{0.531 + 0.2429}{2}$$

$$= 0.2777$$

Linear Layer

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)

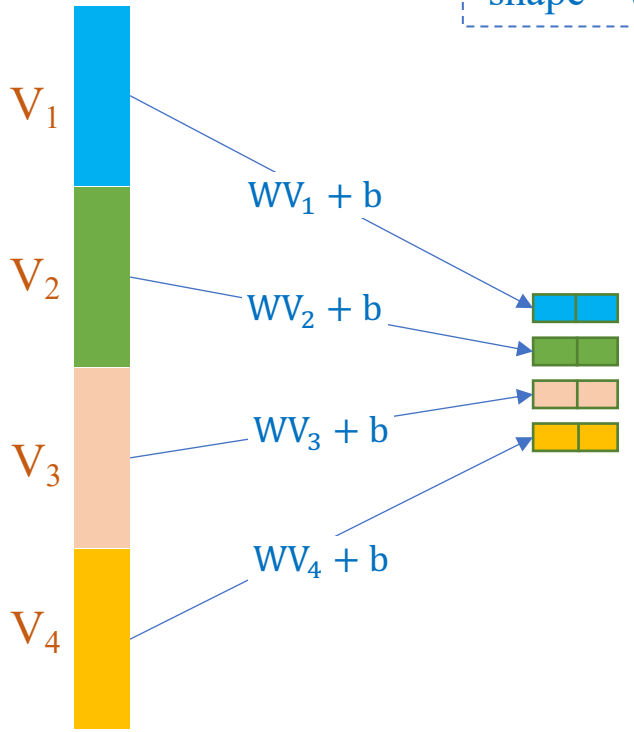
[0.58, -0.45]
[-0.09, -0.23]
[0.82, -1.67]
[-1.83, 0.47]

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

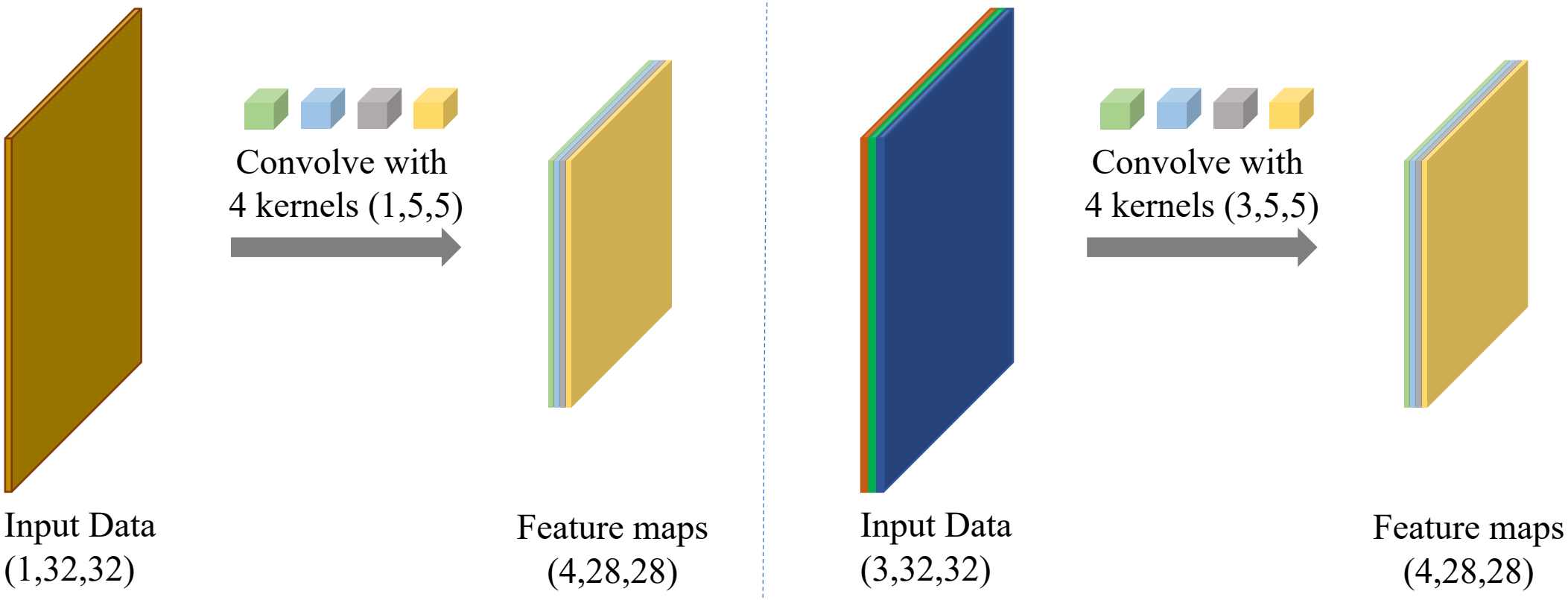
```
torch.Size([1, 4, 2])
```



Convolutional Neural Network

❖ Convolutional layer in PyTorch

```
nn.Conv2d(in_channels, out_channels, kernel_size)
```



RNN Models

❖ Implementation

A sample X

We are learning AI

(L, H_{out})

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix}$$

Batch: (N, L, H_{out})

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix} = \begin{bmatrix} [\ 0.2973 & 0.2388 & -0.8593] \\ [-0.3616 & 0.6365 & -0.8125] \\ [-0.3297 & 0.9502 & 0.6365] \\ [-0.1884 & 0.9009 & 0.4258] \\ [-0.6319 & 0.9454 & 0.2323] \end{bmatrix}$$

```
test_data = data_embedding.reshape(1, 5, 4)
print(test_data.shape)
```

```
torch.Size([1, 5, 4])
```

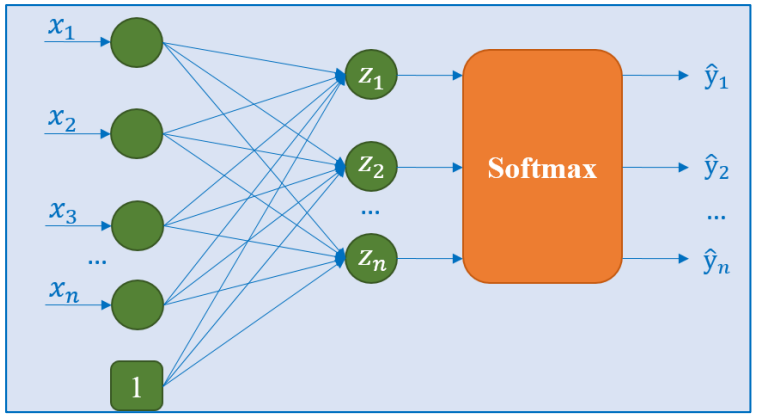
```
rnn_output, rnn_hidden = rnn(test_data)

# (N, L, H_out)
print(rnn_output.shape)
print(rnn_output)
print(rnn_output[:, -1, :])
```

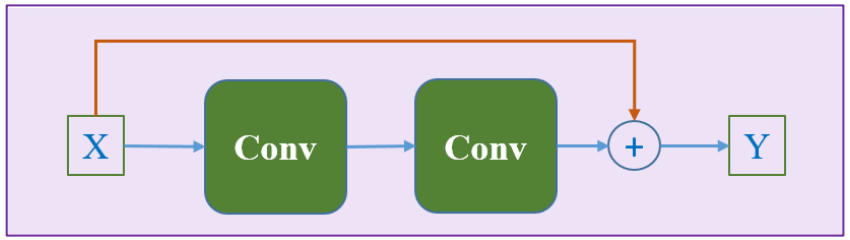
```
torch.Size([1, 5, 3])
tensor([[[ 0.2973,  0.2388, -0.8593],
         [-0.3616,  0.6365, -0.8125],
         [-0.3297,  0.9502,  0.6365],
         [-0.1884,  0.9009,  0.4258],
         [-0.6319,  0.9454,  0.2323]]], grad_fn=<T
tensor([[-0.6319,  0.9454,  0.2323]], grad_fn=<Sli
```

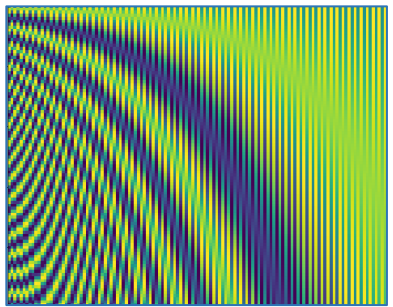
```
# (num_layers, N, H_out)
print(rnn_hidden.shape)
print(rnn_hidden)
print(rnn_hidden[-1, :, :])
```

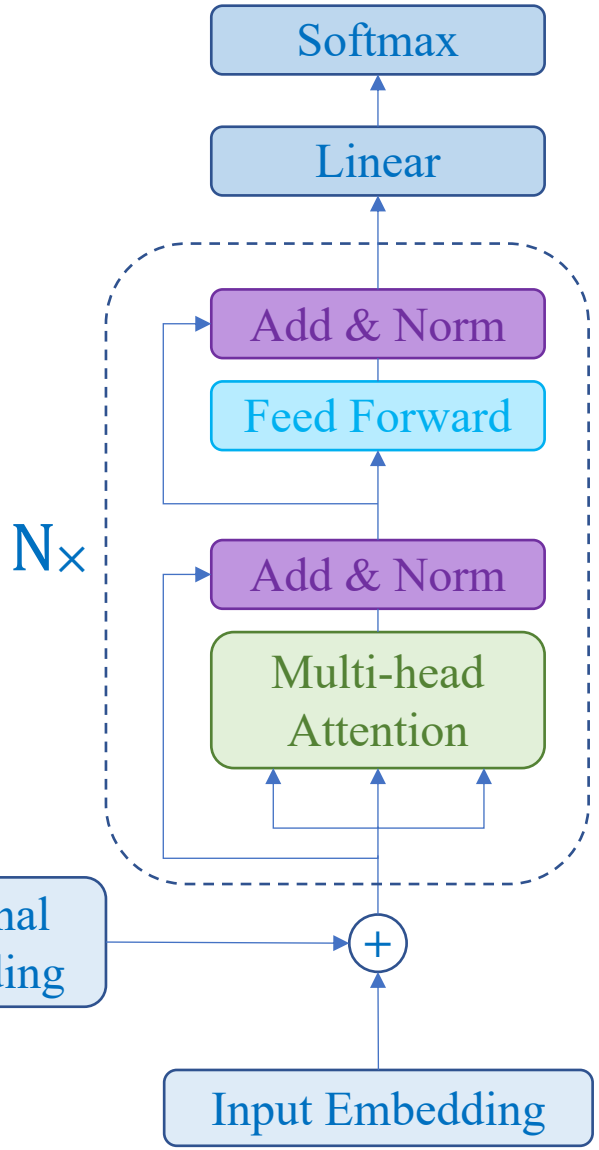
```
torch.Size([1, 1, 3])
tensor([[[-0.6319,  0.9454,  0.2323]]], grad_fn=<S
tensor([[-0.6319,  0.9454,  0.2323]], grad_fn=<Sli
```

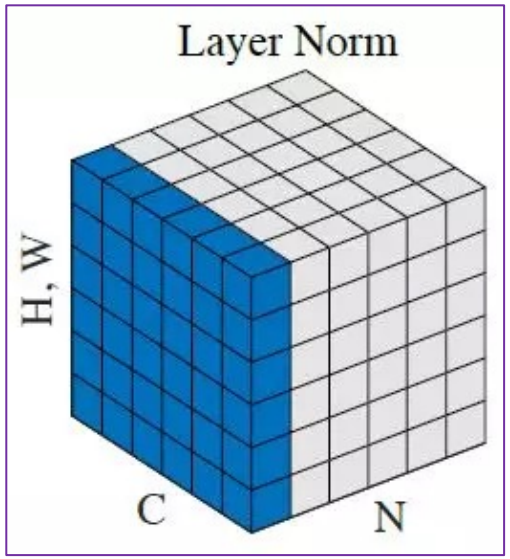
Transformer Models for Text Classification



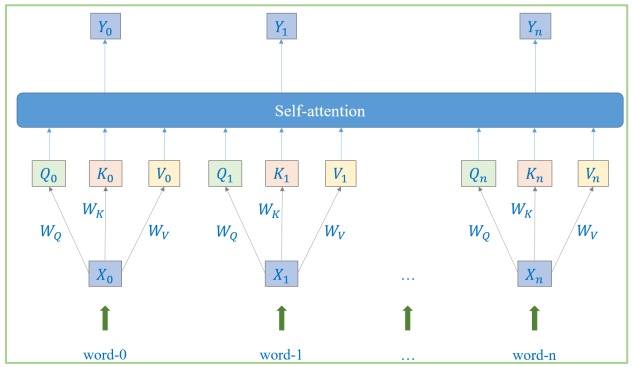


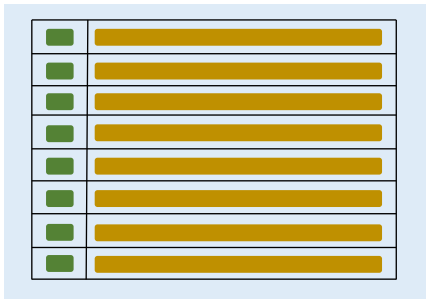






https://arxiv.org/pdf/1803.08494.pdf





Quiz 1

❖ Which one has more weights?

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(5, 4)
        self.fc2 = nn.Linear(4, 3)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = MyModel()
```

```python
from torchinfo import  summary

input_x = torch.randn((32, 5))
model = MyModel()

summary(model, input_data=input_x)
```

```
==========================================================
Layer (type:depth-idx)                   Output Shape
==========================================================
MyModel                                  [32, 3]
├─Linear: 1-1                            [32, 4]
├─Linear: 1-2                            [32, 3]
==========================================================
Total params: 39
```

```python
from torchinfo import  summary

input_x = torch.randn((32, 8, 5))
model = MyModel()

summary(model, input_data=input_x)
```

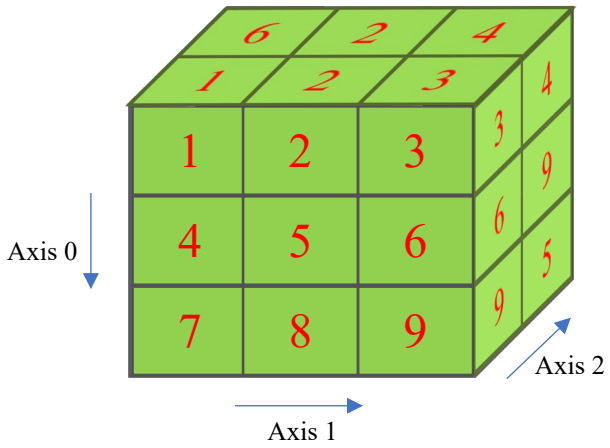
```
==========================================================
Layer (type:depth-idx)                   Output Shape
==========================================================
MyModel                                  [32, 8, 3]
├─Linear: 1-1                            [32, 8, 4]
├─Linear: 1-2                            [32, 8, 3]
==========================================================
Total params: 39
```

Quiz 2

❖ Loss function

PyTorch

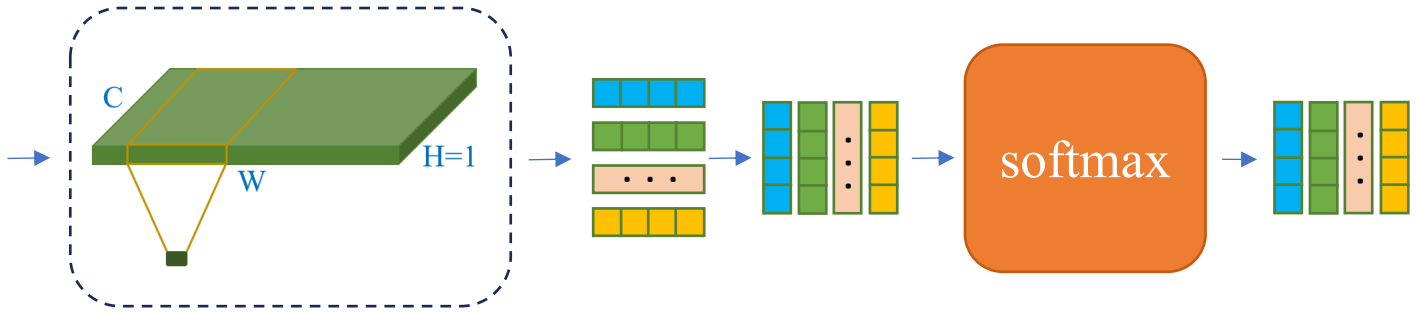
```
criterion = nn.CrossEntropyLoss()
loss = criterion(Z, y)
```



Three dimensions includes
- batch_size N
- sequence_length L
- num_classes C

if the y shape is (N,)

the Z shape is (?)



if the Z shape is (N, C, L)

the y shape is (?)

Conll2003 Dataset for Part-of-Speed Tagging

Num_classes = 47

Train	Val	Test
14041	3250	3453

0	“	Quotation mask
1		space
2	#	Hash
3	$	Dolla
4	(	Opening parenthesis
5	)	Closing parenthesis
6	,	Comma
7	.	Dot
8	:	Colon
9	``	Apostrophe

10	CC	Coordinating conjunction
11	CD	Cardinal number
12	DT	Determiner
13	EX	Existential *there*
14	FW	Foreign word
15	IN	Preposition or subordinating conjunction
16	JJ	Adjective
17	JJR	Adjective, comparative
18	JJS	Adjective, superlative
19	LS	List item marker

20	MD	Modal
21	NN	Noun, singular or mass
22	NNP	Proper noun, singular
23	NNPS	Proper noun, plural
24	NNS	Noun, plural
25	NN|SYM	Noun or Symbol
26	PDT	Predeterminer
27	POS	Possessive ending
28	PRP	Personal pronoun
29	PRP$	Possessive pronoun

Conll2003 Dataset for Part-of-Speed Tagging

Num_classes = 47

Example

Input tokens

["Cup", "qualifying", "round", ",", "second", "leg", "soccer", "matches", "on", "Thursday"]

Label

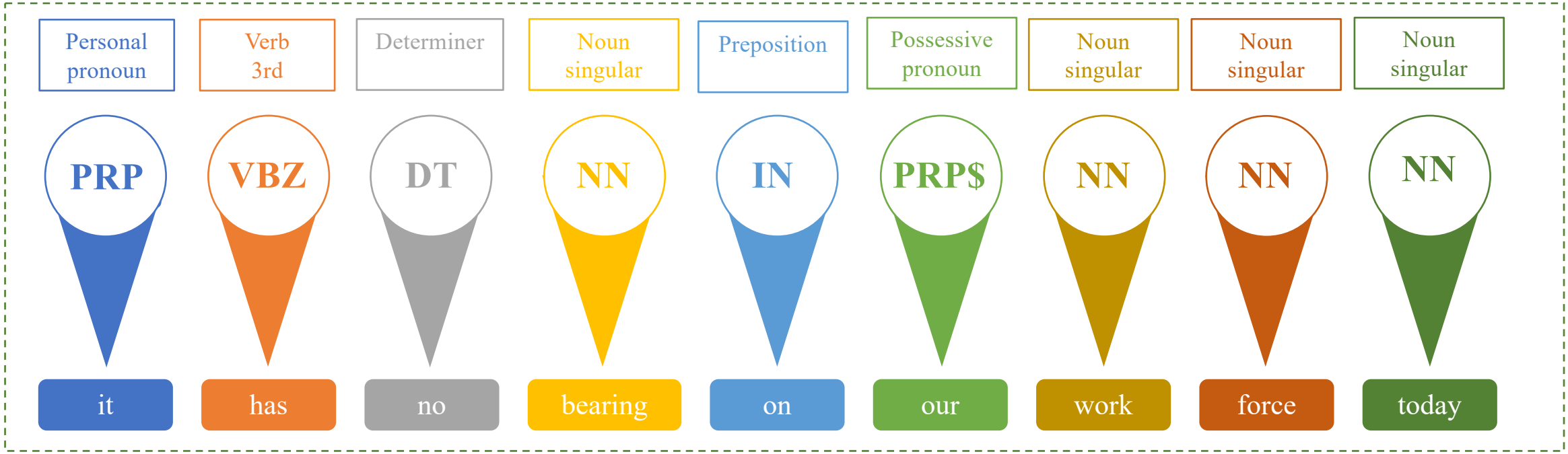
[”NNP", ”VBG", ”RB", ",", ”JJ", ”NN", ”NN", ”NNS", ”IN", ”NNP"]

Label-encoded

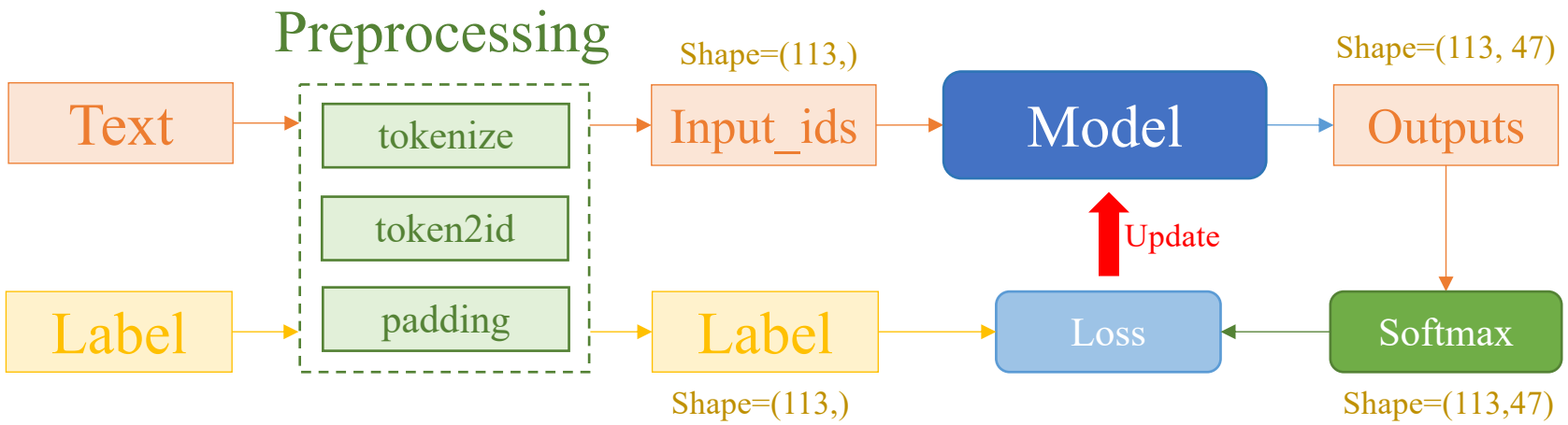
[22, 39, 30, 6, 16, 21, 21, 24, 15, 22]

30	RB	Adverb
31	RBR	Adverb, comparative
32	RBS	Adverb, superlative
33	RP	Particle
34	SYM	Symbol
35	TO	to
36	UH	Interjection
37	VB	Verb, base form
38	VBD	Verb, past tense
39	VBG	Verb, gerund or present participle
40	VBN	Verb, past participle
41	VBP	Verb, non-3rd person singular present
42	VBZ	Verb, 3rd person singular present
43	WDT	Wh-determiner
44	WP	Wh-pronoun
45	WP$	Possessive wh-pronoun
46	WRB	Wh-adverb

Part-of-speed Tagging



Index	Label
0	<unk>
1	NN
2	IN
3	NNP
…	…
43	LS
44	FW
45	UH
46	SYM



Designing a Model for POS Tagging

Doc	Label
i want a dog	[0, 1, 2, 0]
books are expensive	[0, 1, 2]

Label	Meaning
0	Noun/Pronoun
1	Verb
2	Others

building dictionary

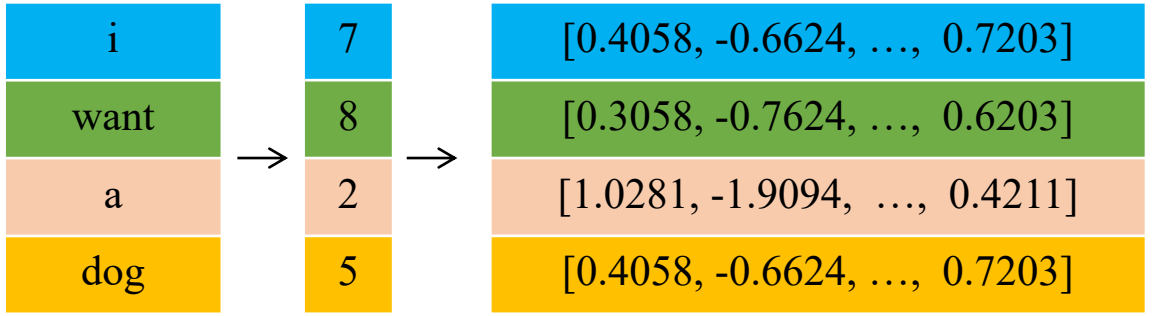
vocab size = 9
sequence length = 4

index	word
0	[UNK]
1	[pad]
2	a
3	are
4	books
5	dog
6	expensive
7	i
8	want

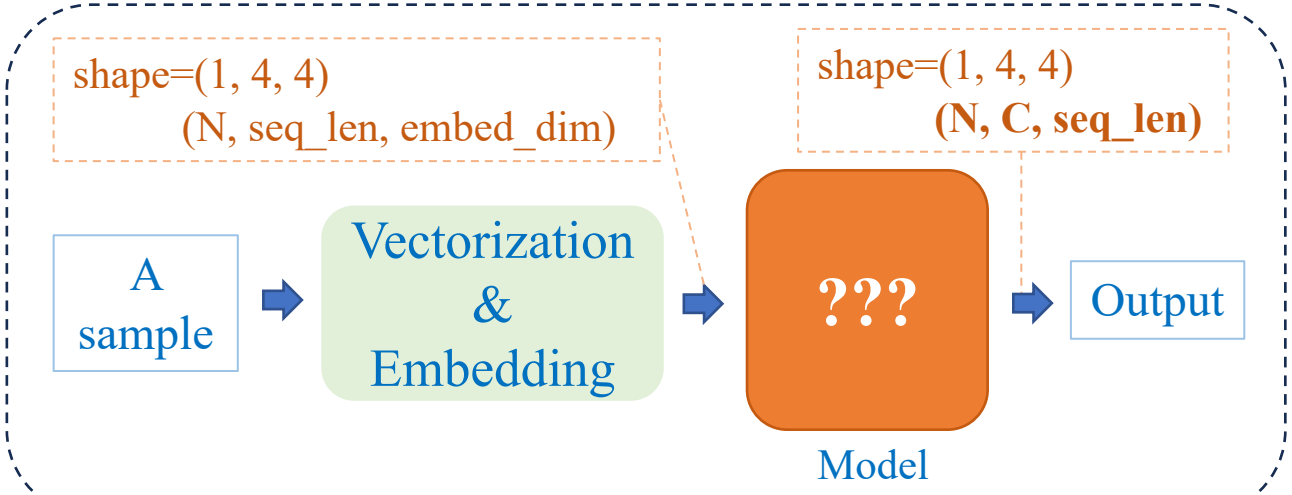
Dictionary

0	[-0.1882, 0.5530, …, 0.7013]
1	[1.7840, -0.8278, …, 1.3586]
2	[1.0281, -1.9094, …, 0.4211]
3	[-1.3083, -0.0987, …, -0.3680]
4	[0.2293, 1.3255, …, 2.0501]
5	[0.4058, -0.6624, …, 0.7203]
6	[0.5582, 0.0786, …, 0.6902]
7	[0.4309, -1.3067, …, 1.5977]
8	[0.3058, -0.7624, …, 0.6203]

Embedding

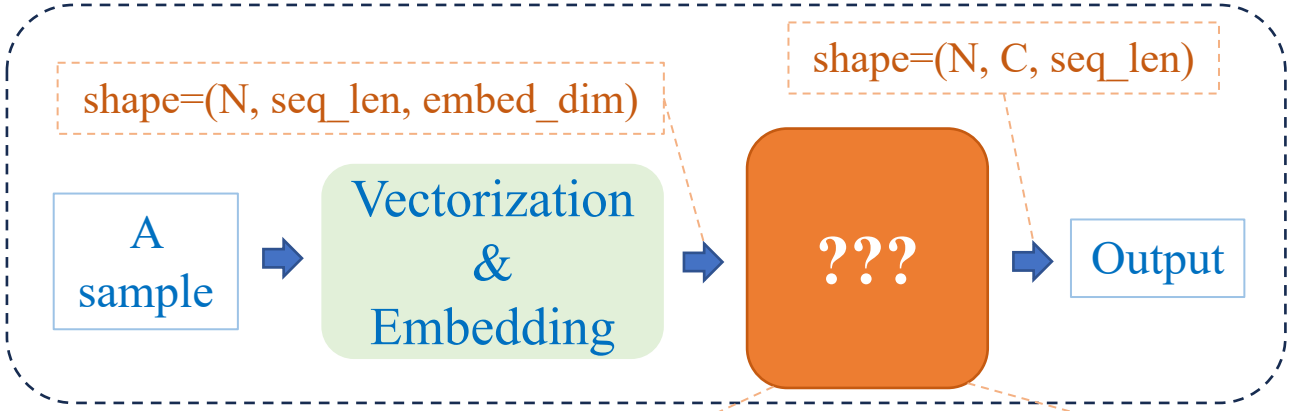


Vectorization and Embedding

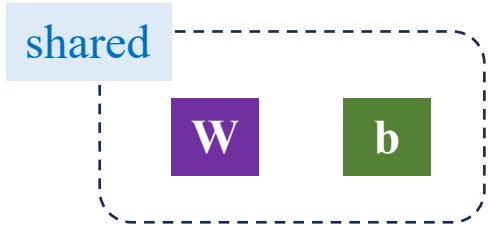


Model Pipeline

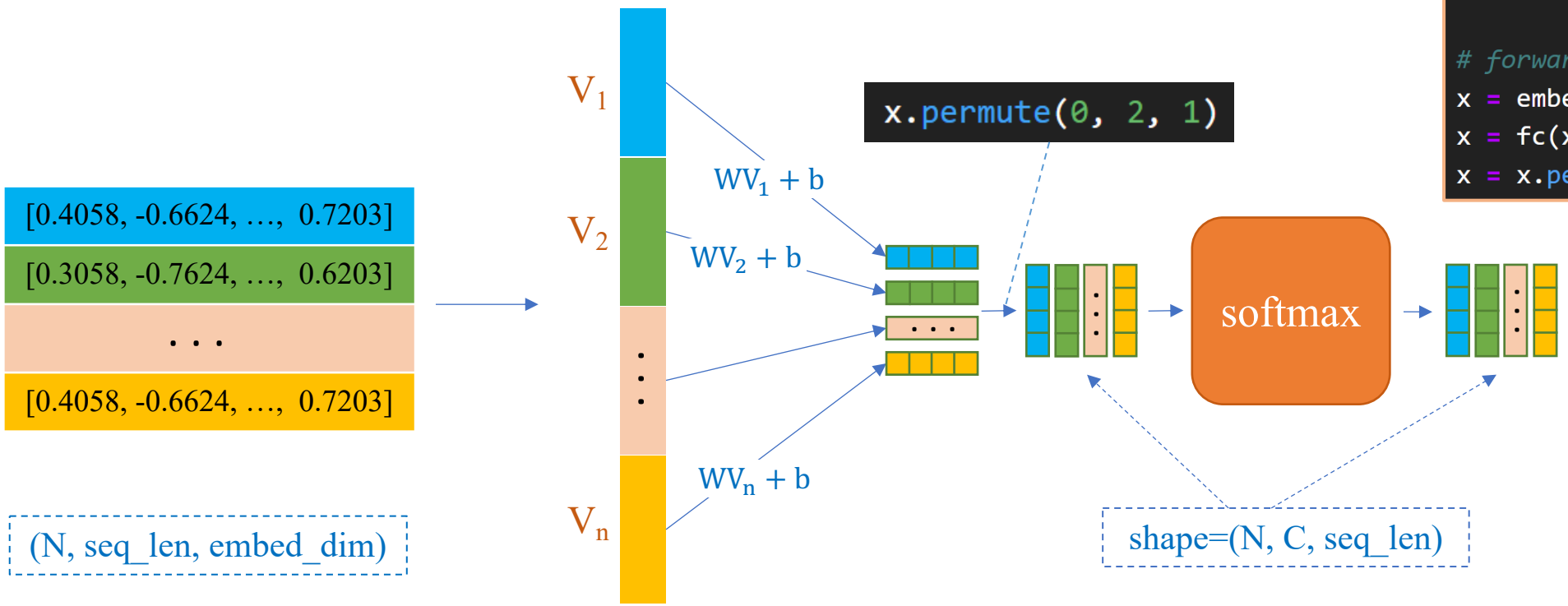
Designing a Model for POS Tagging



Using MLP




```
embedding = nn.Embedding(vocab_size,
                         embed_dim)
fc = nn.Linear(embed_dim,
               num_classes)

# forward
x = embedding(x)
x = fc(x)
x = x.permute(0, 2, 1)
```



POS Tagging: Using MLP

Doc	Label
i want a dog	[0, 1, 2, 0]
books are expensive	[0, 1, 2]

building dictionary

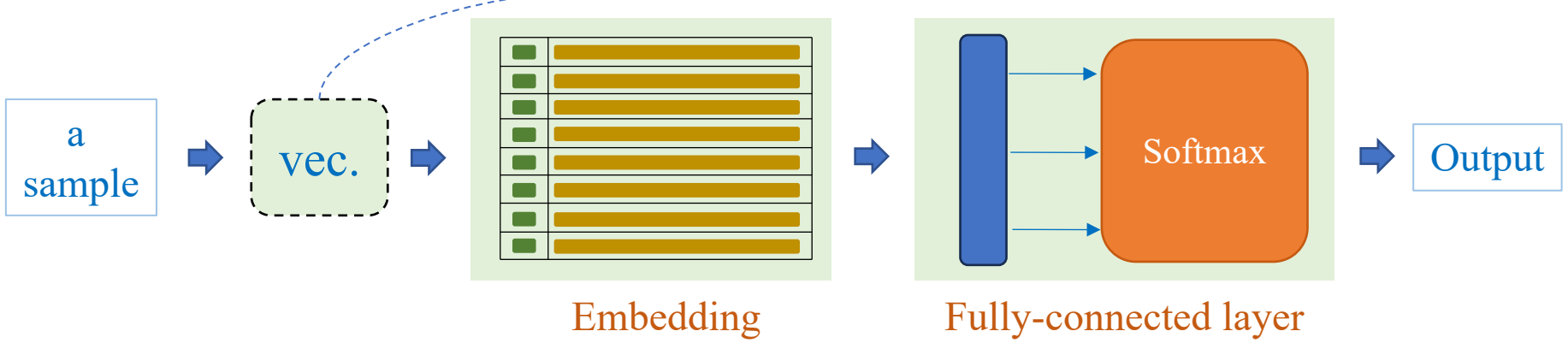
index	word
0	[UNK]
1	[pad]
2	a
3	are
4	books
5	dog
6	expensive
7	i
8	want

vocab size = 9
sequence length = 4

Label
0: (pro)noun
1: verb
2: others

i	→	7	0
want		8	1
a		2	2
dog		5	0
sample 1		vec 1	label 1

books	→	4	0
are		3	1
expensive		6	2
<pad>		1	?
sample 2		vec 2	label 2



POS Tagging: Using MLP

Doc	Label
i want a dog	[0, 1, 2, 0]
books are expensive	[0, 1, 2]

building dictionary

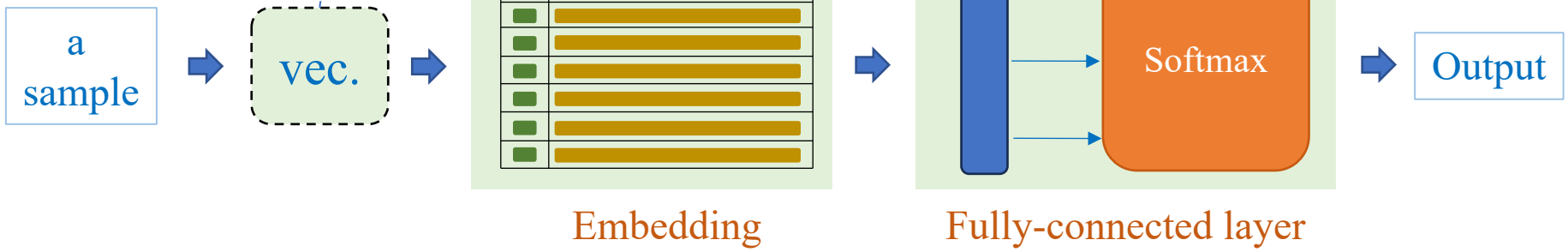
index	word
0	[UNK]
1	[pad]
2	a
3	are
4	books
5	dog
6	expensive
7	i
8	want

vocab size = 9
sequence length = 4

Label
0: (pro)noun
1: verb
2: others
3: <pad>

i	→	7	0
want		8	1
a		2	2
dog		5	0
sample 1		vec 1	label 1

books	→	4	0
are		3	1
expensive		6	2
<pad>		1	3
sample 2		vec 2	label 2

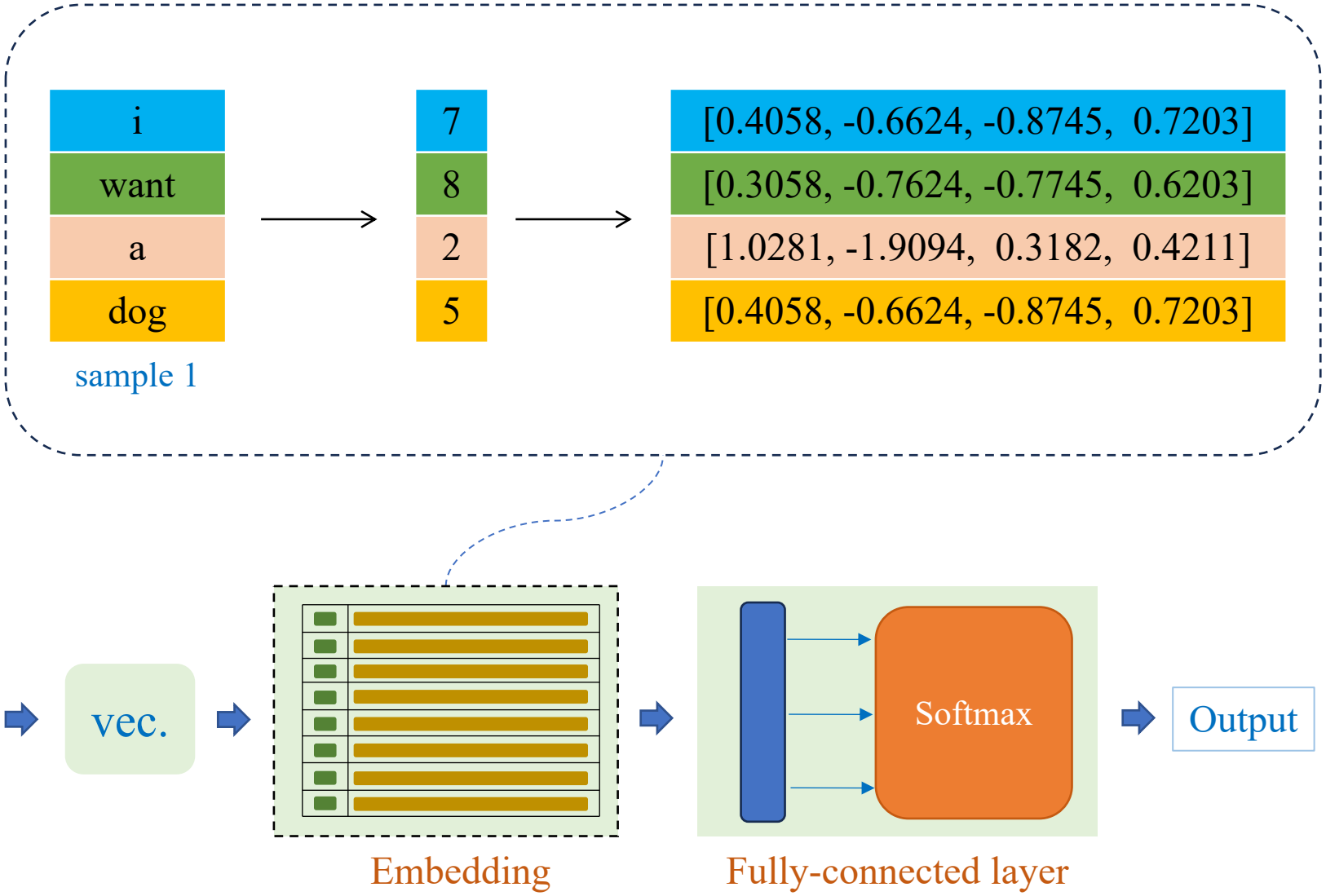


POS Tagging: Using MLP

Doc	Label
i want a dog	[0, 1, 2, 0]
books are expensive	[0, 1, 2]

0	[-0.1882, 0.5530, 1.6267, 0.7013]
1	[1.7840, -0.8278, -0.2701, 1.3586]
2	[1.0281, -1.9094, 0.3182, 0.4211]
3	[-1.3083, -0.0987, 0.7647, -0.3680]
4	[0.2293, 1.3255, 0.1318, 2.0501]
5	[0.4058, -0.6624, -0.8745, 0.7203]
6	[0.5582, 0.0786, -0.6817, 0.6902]
7	[0.4309, -1.3067, -0.8823, 1.5977]
8	[0.3058, -0.7624, -0.7745, 0.6203]

vocab size = 9
sequence length = 4



W

[-0.3875, -0.3519, -0.1275, -0.1719]
[0.4391, 0.0455, -0.1566, -0.2897]
[0.1777, -0.1178, -0.3101, -0.2451]
[0.3730, 0.0996, -0.3004, 0.2219]

b

[0.3548, -0.2819, -0.0579, 0.5113]

$\hat{y}$

[0.26, 0.09, 0.16, 0.49]
[0.27, 0.13, 0.19, 0.41]
[0.29, 0.15, 0.21, 0.35]
[0.24, 0.13, 0.19, 0.44]

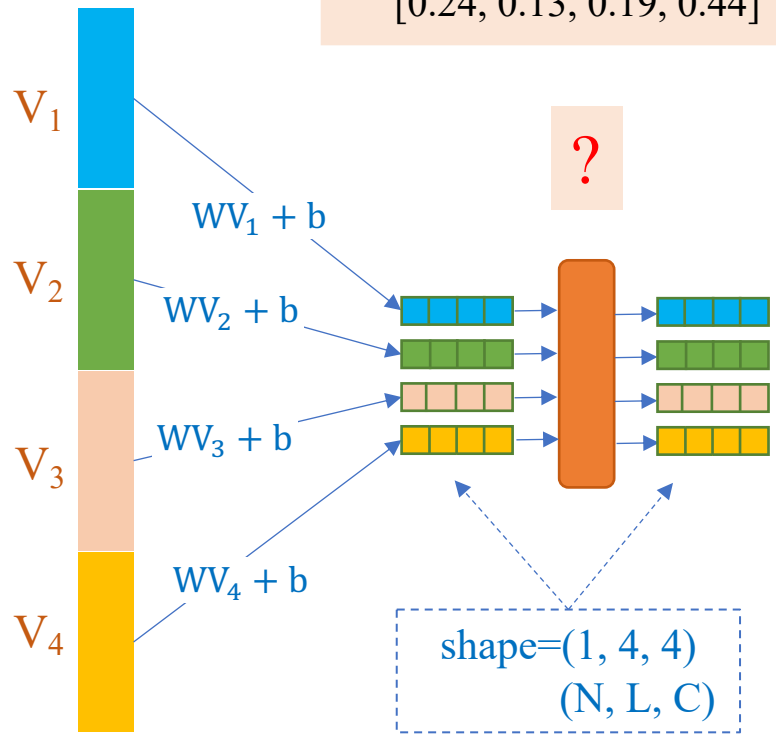
Doc	Label
i want a dog	[0, 1, 2, 0]
books are expensive	[0, 1, 2]

Label	
0: (pro)noun	2: others
1: verb	3: <pad>

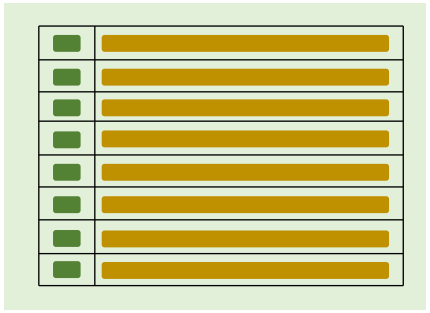
0	[-0.1882, 0.5530, 1.6267, 0.7013]
1	[1.7840, -0.8278, -0.2701, 1.3586]
2	[1.0281, -1.9094, 0.3182, 0.4211]
3	[-1.3083, -0.0987, 0.7647, -0.3680]
4	[0.2293, 1.3255, 0.1318, 2.0501]
5	[0.4058, -0.6624, -0.8745, 0.7203]
6	[0.5582, 0.0786, -0.6817, 0.6902]
7	[0.4309, -1.3067, -0.8823, 1.5977]
8	[0.3058, -0.7624, -0.7745, 0.6203]

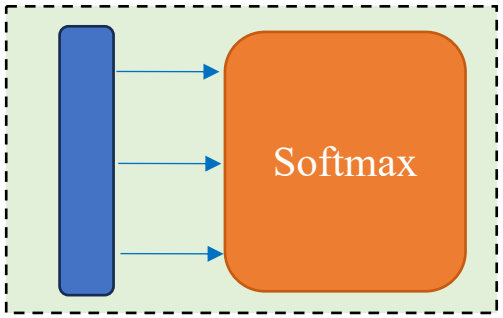
vocab size = 9
sequence length = 4

[0.4058, -0.6624, -0.8745, 0.7203]
[0.3058, -0.7624, -0.7745, 0.6203]
[1.0281, -1.9094, 0.3182, 0.4211]
[0.4058, -0.6624, -0.8745, 0.7203]



vec.





Output

y = [0,1, 2, 0] ; shape=(1,4)

W

[-0.3875, -0.3519, -0.1275, -0.1719]
[0.4391, 0.0455, -0.1566, -0.2897]
[0.1777, -0.1178, -0.3101, -0.2451]
[0.3730, 0.0996, -0.3004, 0.2219]

b

[0.3548, -0.2819, -0.0579, 0.5113]

Shape of logits = (N, C, L)
Shape of target = (N, L)

pytorch requirement

Doc	Label
i want a dog	[0, 1, 2, 0]
books are expensive	[0, 1, 2]

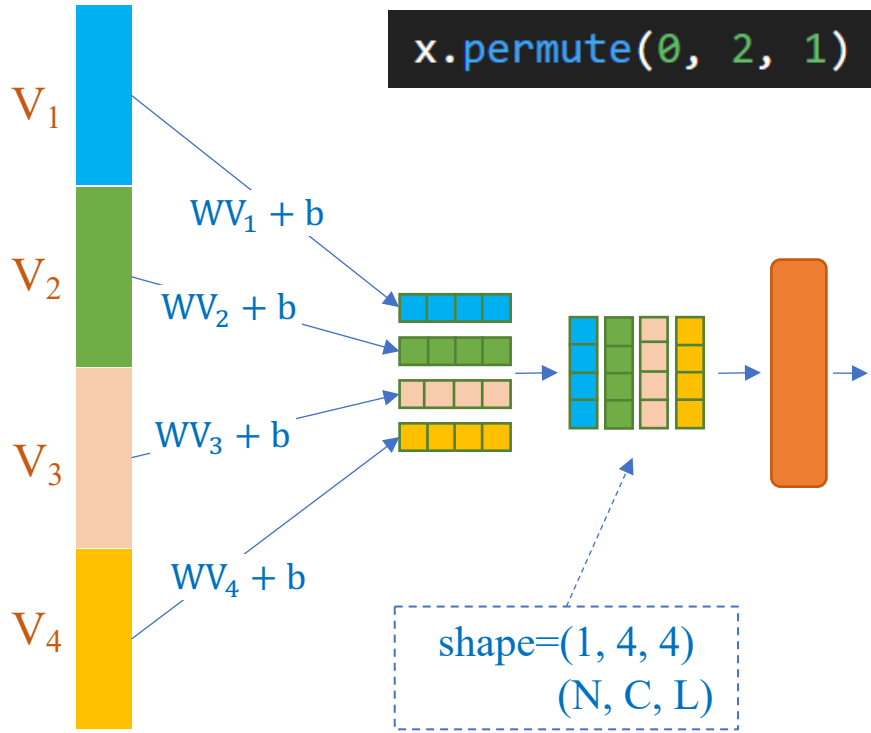
Label	
0: (pro)noun	2: others
1: verb	3: <pad>

```
embedding = nn.Embedding(9, 4)
fc = nn.Linear(4, 4)

# forward
x = embedding(x)
x = fc(x)
x = x.permute(0, 2, 1)
```

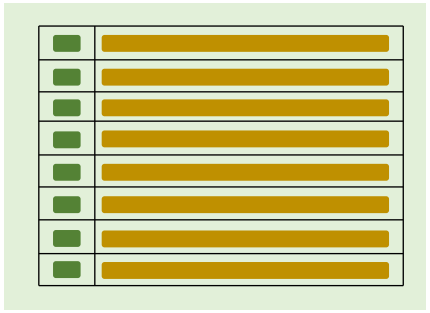
```
nn.CrossEntropyLoss(ignore_index=3)
torch.optim.Adam(model.parameters(),
                 lr=0.1)
```

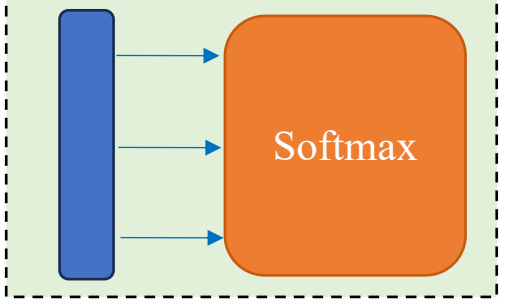
vocab size = 9
sequence length = 4

[0.4058, -0.6624, -0.8745, 0.7203]
[0.3058, -0.7624, -0.7745, 0.6203]
[1.0281, -1.9094, 0.3182, 0.4211]
[0.4058, -0.6624, -0.8745, 0.7203]



vec.





Output

y = [0, 1, 2, 0] ; shape=(1,4)

POS Tagging: Using MLP

❖ Implementation

```
class POS_Model(nn.Module):
    def __init__(self, vocab_size, num_classes):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, 4)
        self.fc = nn.Linear(4, num_classes)

    def forward(self, x):
        x = self.embedding(x)
        x = self.fc(x)
        return x.permute(0, 2, 1)

model = POS_Model(vocab_size, 4)

# test
input = torch.tensor([[7, 8, 2, 5]], dtype=torch.long)
output = model(input)
print(output.shape)
```

✓ 0.0s

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

```
for _ in range(30):
    optimizer.zero_grad()
    outputs = model(input_data)
    loss = criterion(outputs,
                     label_data)
    print(loss.item())
    loss.backward()
    optimizer.step()
```

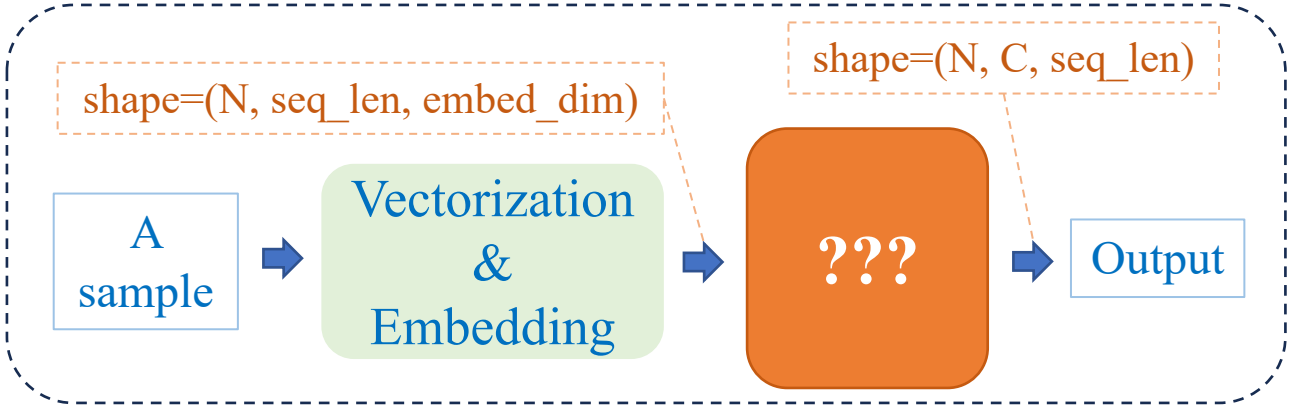
```
# [[0, 1, 2, 0],
#  [0, 1, 2, *]]

outputs = model(input_data)
o_softmax = torch.softmax(outputs,
                          axis=1)
o_softmax.argmax(axis=1)
```

✓ 0.0s

```
tensor([[0, 1, 2, 0],
        [0, 1, 0, 3]])
```

Using Other Deep Architectures

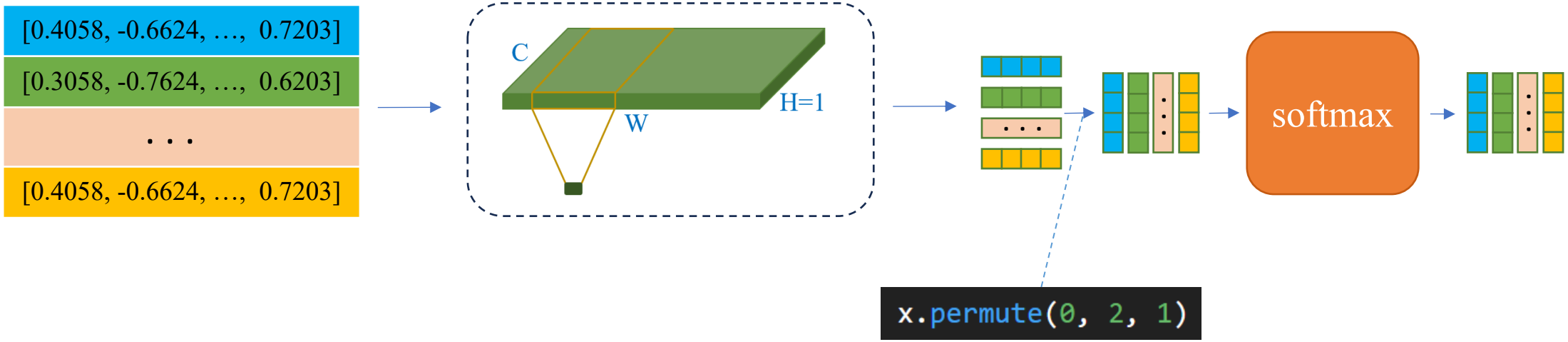
Designing a Model for POS Tagging



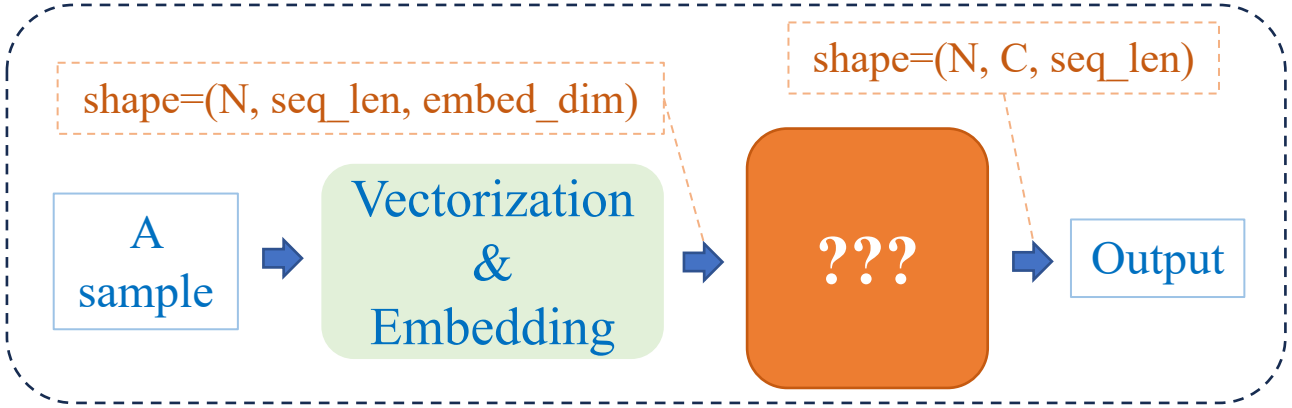
Using CNN: Quiz 3

This pipeline is wrong.
Let's find out!

```
embedding = nn.Embedding(vocab_size, 3)
conv1d = nn.Conv1d(3, num_classes,
                   kernel_size=3, padding='same')
# forward
x = self.embedding(x)
x = self.conv1d(x)
x = x.permute(0, 2, 1)
```

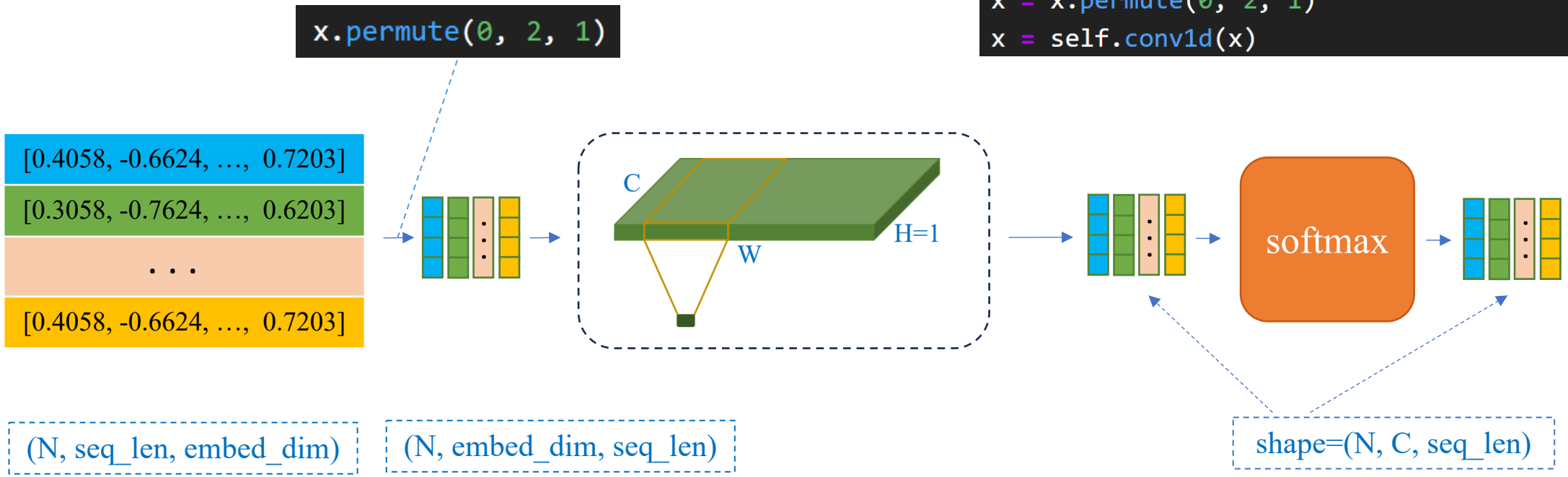


Designing a Model for POS Tagging



Using CNN

```
embedding = nn.Embedding(vocab_size, 3)
conv1d = nn.Conv1d(3, num_classes,
                   kernel_size=3, padding='same')
# forward
x = self.embedding(x)
x = x.permute(0, 2, 1)
x = self.conv1d(x)
```



POS Tagging: Using CNN

Implementation

```
class POS_Model(nn.Module):
    def __init__(self, vocab_size, num_classes):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, 3)
        self.conv1d = nn.Conv1d(3, num_classes,
                                kernel_size=3,
                                padding='same')

    def forward(self, x):
        x = self.embedding(x)
        x = x.permute(0, 2, 1)
        x = self.conv1d(x)
        return x

num_classes = 4
model = POS_Model(vocab_size, num_classes)

# test
data = torch.tensor([[3, 2, 3, 2, 1]]).long()
output = model(data)
print(output.shape)
```

✓ 0.0s

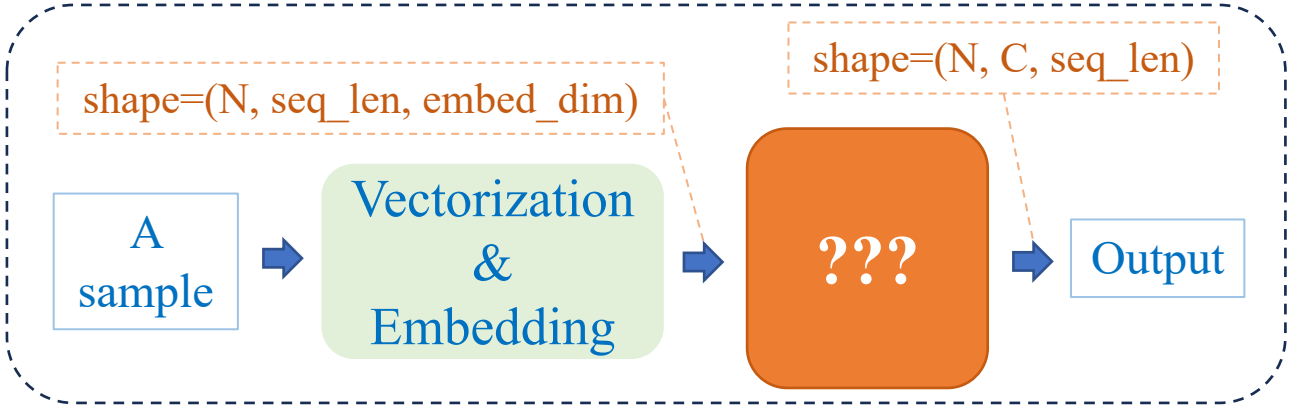
```
torch.Size([1, 4, 5])
```

```
for _ in range(30):
    optimizer.zero_grad()
    outputs = model(input_data)
    loss = criterion(outputs,
                     label_data)
    print(loss.item())
    loss.backward()
    optimizer.step()
```

```
# [[0, 1, 0, 2, 0],
#  [0, 1, 2, 3, 3]]

outputs = model(input_data)
# outputs: (N, C, L)

o_softmax = torch.softmax(outputs,
                          axis=1)
o_softmax.argmax(axis=1)
```

```
tensor([[0, 1, 0, 2, 0],
        [0, 1, 2, 2, 2]])
```

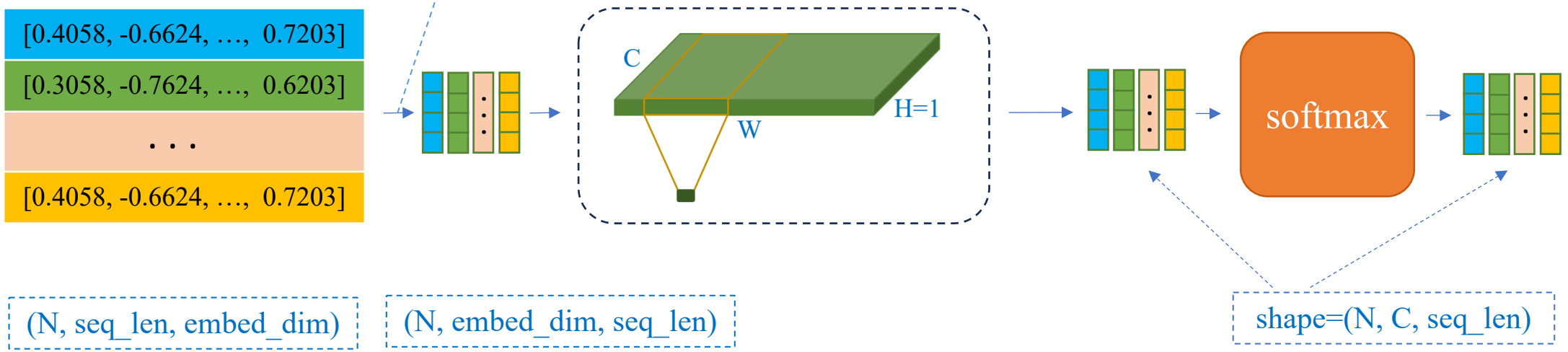
Designing a Model for POS Tagging



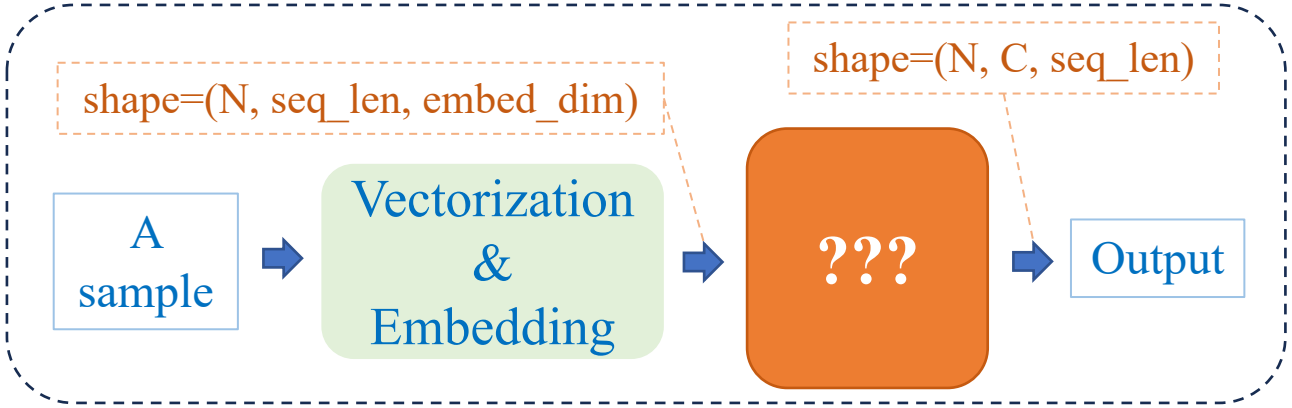
Using CNN: Quiz 4

```
embedding = nn.Embedding(vocab_size, 3)
conv1d = nn.Conv1d(3, num_classes,
                   kernel_size=3)
# forward
x = self.embedding(x)
x = x.permute(0, 2, 1)
x = self.conv1d(x)
```

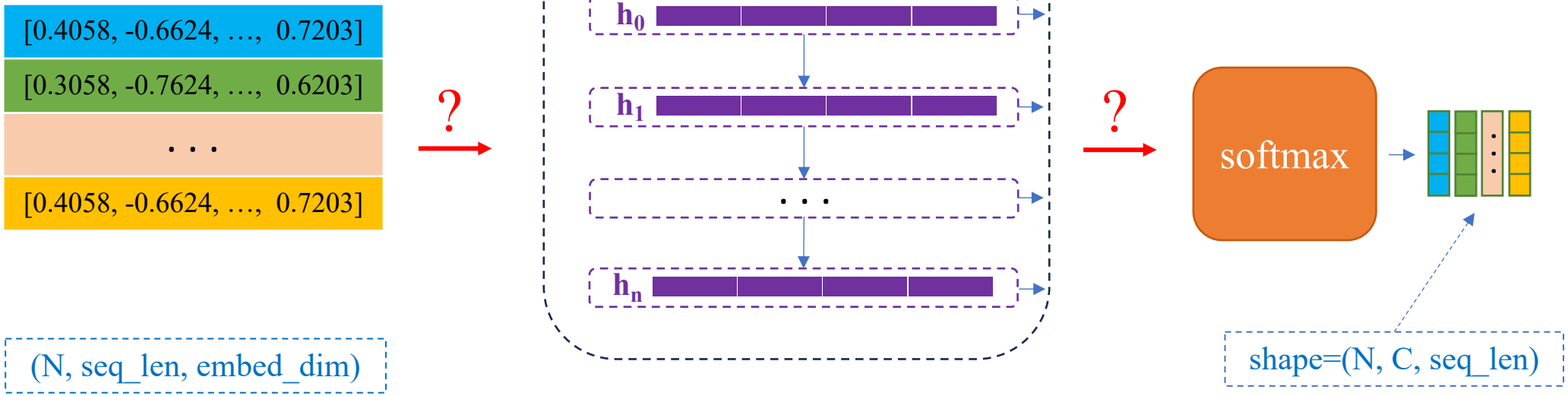
```
x.permute(0, 2, 1)
```



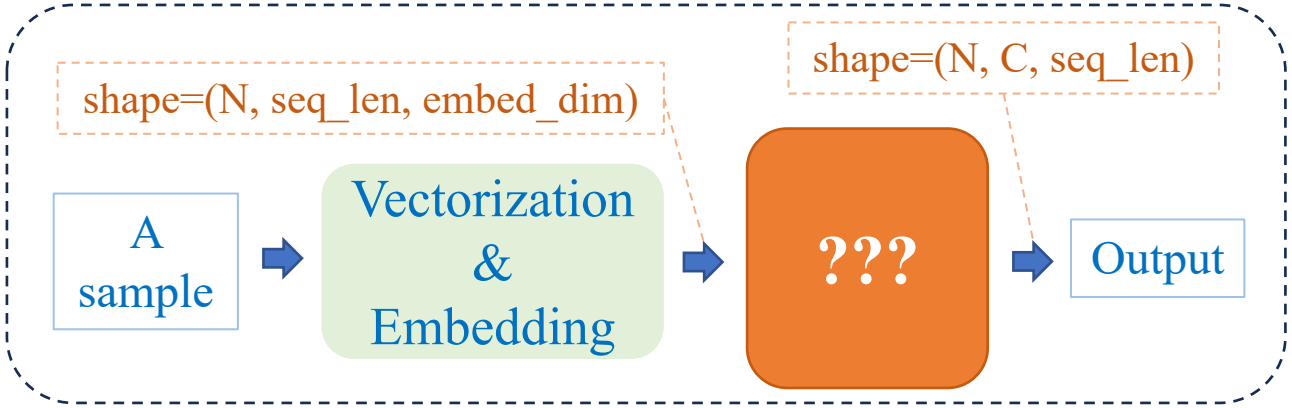
Designing a Model for POS Tagging



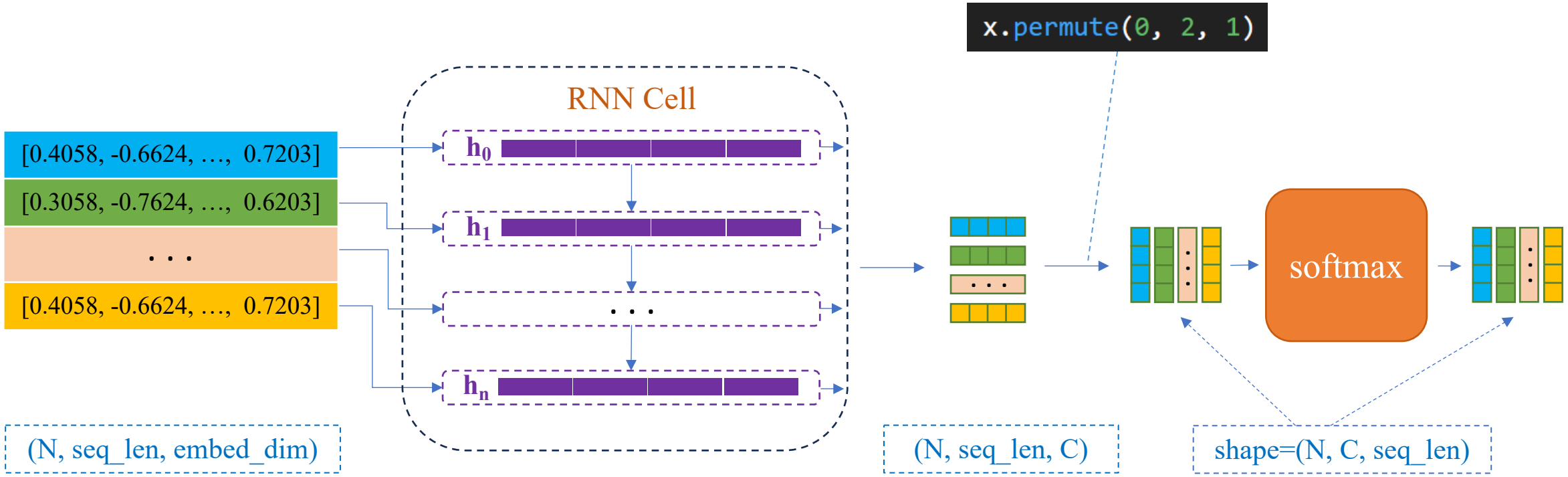
Using RNN: Quiz 5



Designing a Model for POS Tagging



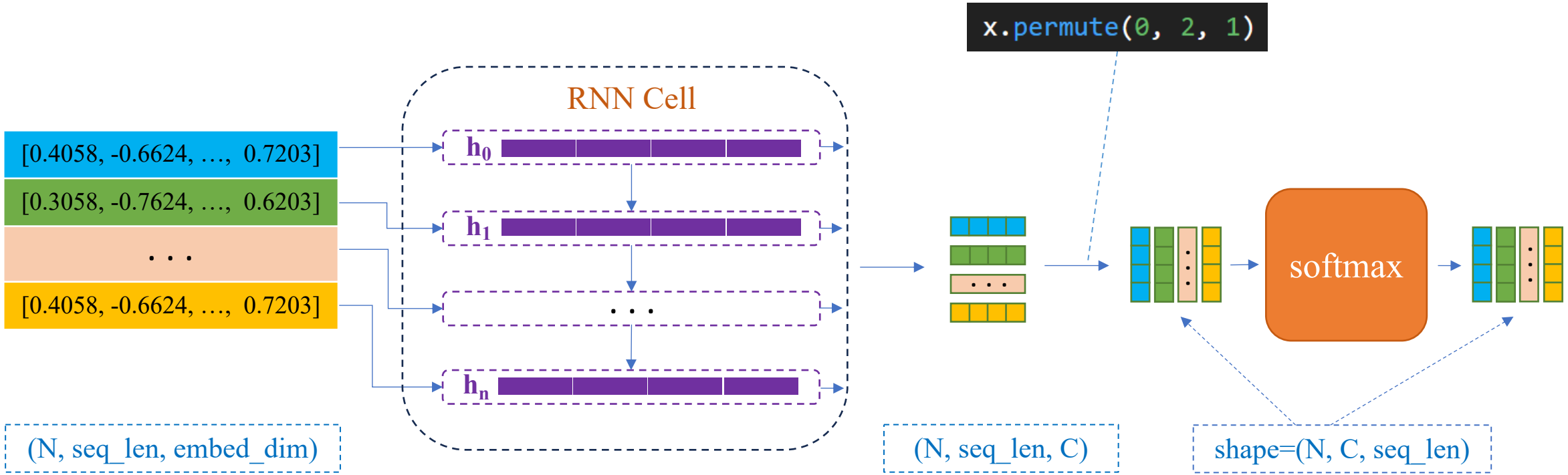
Using RNN



Designing a Model for POS Tagging

Using RNN: Implementation

```
embedding = nn.Embedding(vocab_size, emb_dim)
recurrent = nn.RNN(emb_dim, num_classes, batch_first=True)

# forward
x = embedding(x)
output, _ = recurrent(x)
x = output.permute(0, 2, 1)
```



POS Tagging: Using RNN

Implementation

```python
class POS_Model(nn.Module):
    def __init__(self, vocab_size, num_classes):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, 3)
        self.recurrent = nn.RNN(3, num_classes,
                                batch_first=True)

    def forward(self, x):
        x = self.embedding(x)
        x,_ = self.recurrent(x)
        return x.permute(0, 2, 1)

num_classes = 4
model = POS_Model(vocab_size, num_classes)

# test
data = torch.tensor([[3, 2, 3, 2, 1]]).long()
output = model(data)
print(output.shape)
```

✓ 0.0s

```
torch.Size([1, 4, 5])
```

```python
for _ in range(30):
    optimizer.zero_grad()
    outputs = model(input_data)
    loss = criterion(outputs,
                     label_data)
    print(loss.item())
    loss.backward()
    optimizer.step()
```

```python
# [[0, 1, 0, 2, 0],
#  [0, 1, 2, *, *]]

outputs = model(input_data)
o_softmax = torch.softmax(outputs,
                          axis=1)
o_softmax.argmax(axis=1)
```

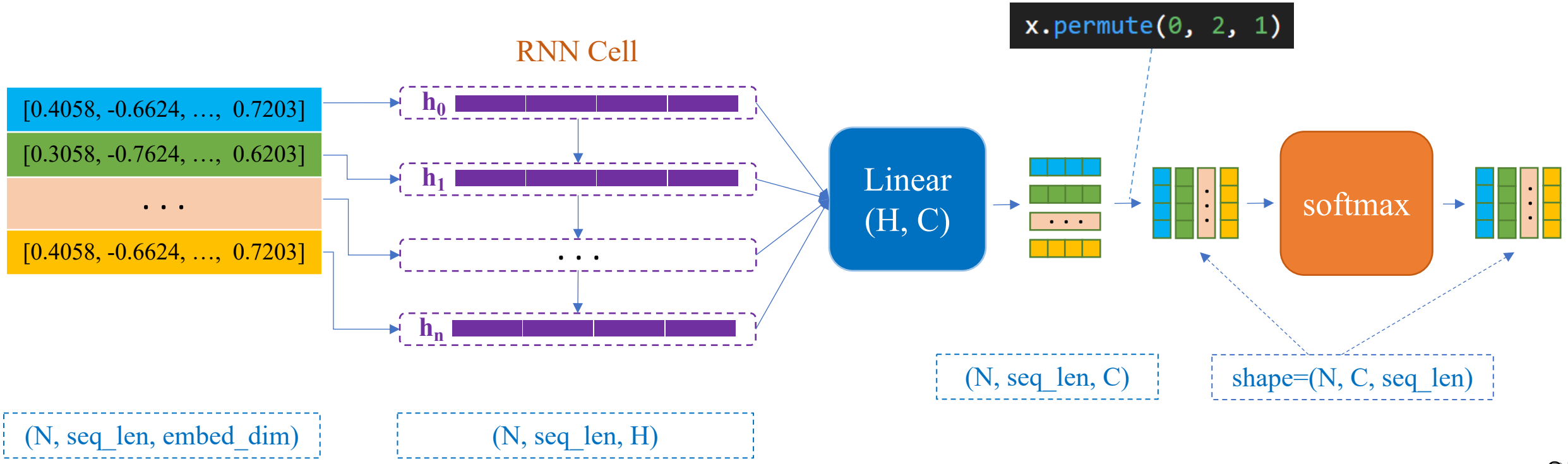
```
tensor([[0, 1, 0, 2, 0],
        [0, 1, 0, 1, 0]])
```

Designing a Model for POS Tagging

Using RNN + Linear

Similar to LSTM/GRU

```
embedding = nn.Embedding(vocab_size, emb_dim)
recurrent = nn.RNN(emb_dim, hidden_size, batch_first=True)
fc = nn.Linear(hidden_size, num_classes)

# forward
x = embedding(x)
output, _ = recurrent(x)
x = fc(output)
x = x.permute(0, 2, 1)
```



POS Tagging: Using RNN+Linear

Implementation

```
class POS_Model(nn.Module):
    def __init__(self, vocab_size, num_classes):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, 4)
        self.recurrent = nn.RNN(4, 3,
                                batch_first=True)
        self.fc = nn.Linear(3, num_classes)

    def forward(self, x):
        x = self.embedding(x)
        x,_ = self.recurrent(x)
        x = self.fc(x)
        return x.permute(0, 2, 1)

num_classes = 4
model = POS_Model(vocab_size, num_classes)

# test
data = torch.tensor([[3, 2, 3, 2, 1]]).long()
output = model(data)
print(output.shape)
```

✓ 0.0s

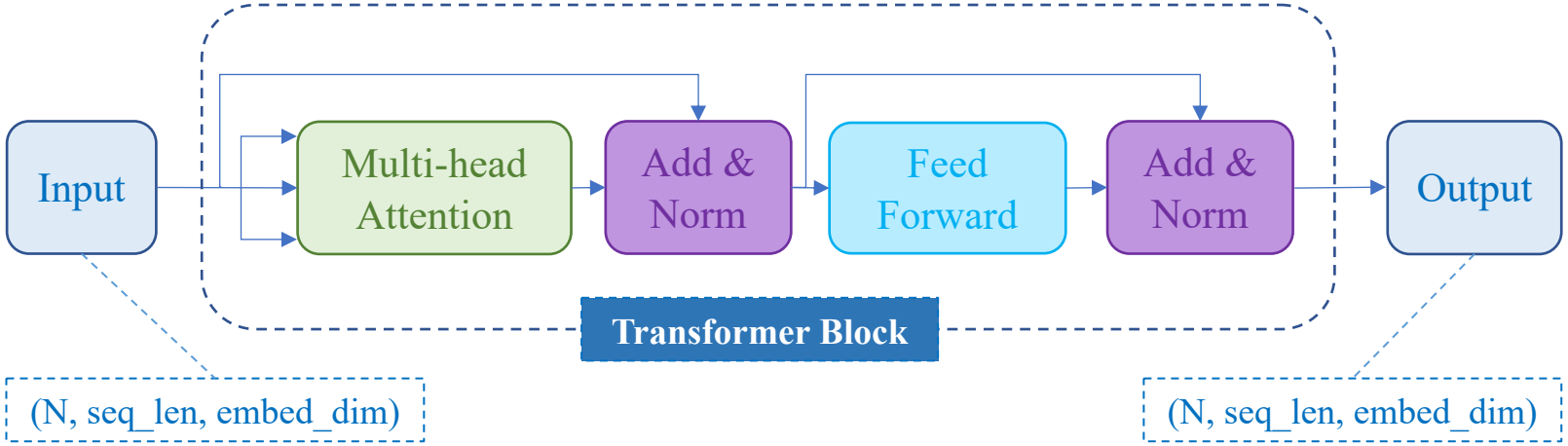
```
torch.Size([1, 4, 5])
```

```
for _ in range(30):
    optimizer.zero_grad()
    outputs = model(input_data)
    loss = criterion(outputs,
                     label_data)
    print(loss.item())
    loss.backward()
    optimizer.step()
```

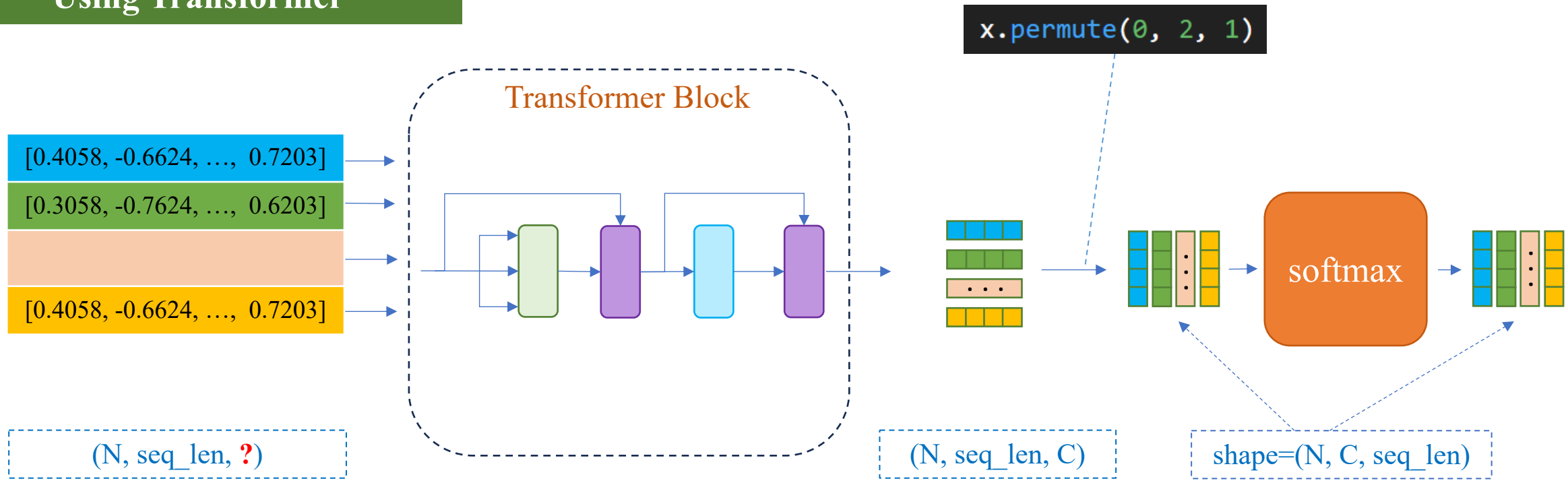
```
# [[0, 1, 0, 2, 0],
#  [0, 1, 2, *, *]]

outputs = model(input_data)
o_softmax = torch.softmax(outputs,
                          axis=1)
o_softmax.argmax(axis=1)
```

```
tensor([[0, 1, 0, 2, 0],
        [0, 1, 0, 1, 0]])
```

Designing a Model for POS Tagging



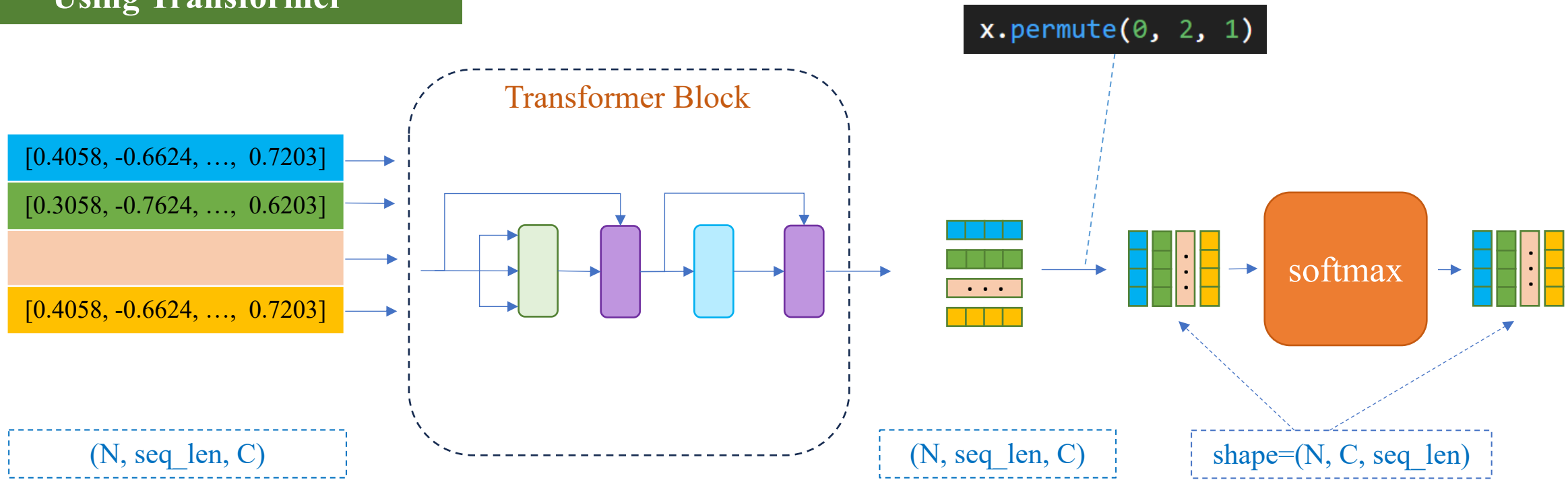
Using Transformer



Designing a Model for POS Tagging

```
embedding = nn.Embedding(vocab_size, 4)
transformer = TransformerBlock(4, 1, 4)
# embed_dim, num_heads, ff_dim

# forward
x = self.embedding(x)
x = self.transformer(x, x, x)
x =  x.permute(0, 2, 1)
```

Using Transformer



POS Tagging: Using Transformer

Implementation

```
class POS_Model(nn.Module):
    def __init__(self, vocab_size, num_classes):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, 4)
        self.transformer = TransformerBlock(4, 1, 4)

    def forward(self, x):
        x = self.embedding(x)
        x = self.transformer(x, x, x)
        return x.permute(0, 2, 1)

num_classes = 4
model = POS_Model(vocab_size, num_classes)

# test
data = torch.tensor([[3, 2, 3, 2, 1]]).long()
output = model(data)
print(output.shape)
```

✓ 0.0s

```
torch.Size([1, 4, 5])
```

```
for _ in range(30):
    optimizer.zero_grad()
    outputs = model(input_data)
    loss = criterion(outputs,
                     label_data)
    print(loss.item())
    loss.backward()
    optimizer.step()
```

```
# [[0, 1, 0, 2, 0],
#  [0, 1, 2, *, *]]

outputs = model(input_data)
o_softmax = torch.softmax(outputs,
                          axis=1)
o_softmax.argmax(axis=1)
```

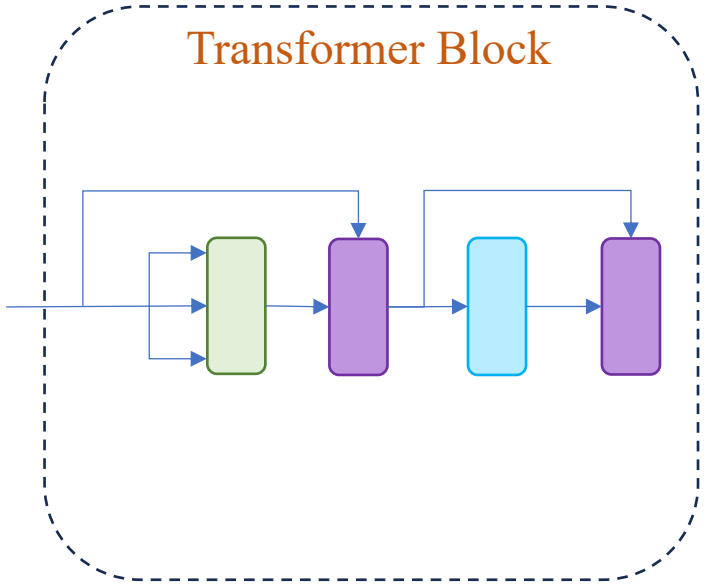
```
tensor([[0, 1, 0, 2, 0],
        [0, 1, 0, 1, 0]])
```

POS Tagging: Using Transformer

Implementation

```
class TransformerBlock(nn.Module):
    def __init__(self, embed_dim, num_heads, ff_dim):
        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)
        x = self.layernorm_2(out_1 + ffn_output)

        return x
```

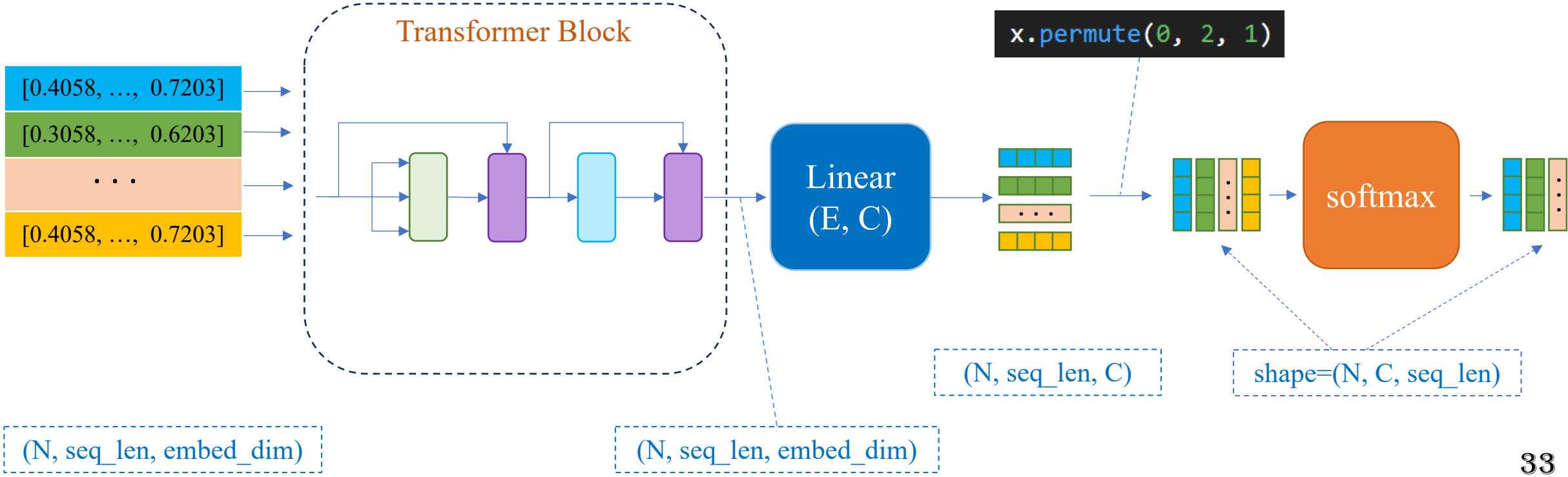
✓ 0.0s



Designing a Model for POS Tagging

```
embedding = nn.Embedding(vocab_size, embed_dim)
transformer = TransformerBlock(embed_dim, 1, embed_dim)
fc = nn.Linear(embed_dim, num_classes)

# forward
x = self.embedding(x)
x = self.transformer(x, x, x)
x = self.fc(x)
x = x.permute(0, 2, 1)
```

Using Transformer + Linear



POS Tagging: Using Transformer+Linear

Implementation

```
class POS_Model(nn.Module):
    def __init__(self, vocab_size, num_classes):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, 5)
        self.transformer = TransformerBlock(5, 1, 5)
        self.fc = nn.Linear(5, num_classes)

    def forward(self, x):
        x = self.embedding(x)
        x = self.transformer(x, x, x)
        x = self.fc(x)
        return x.permute(0, 2, 1)

num_classes = 4
model = POS_Model(vocab_size, num_classes)

# test
data = torch.tensor([[3, 2, 3, 2, 1]]).long()
output = model(data)
print(output.shape)
```

✓ 0.0s

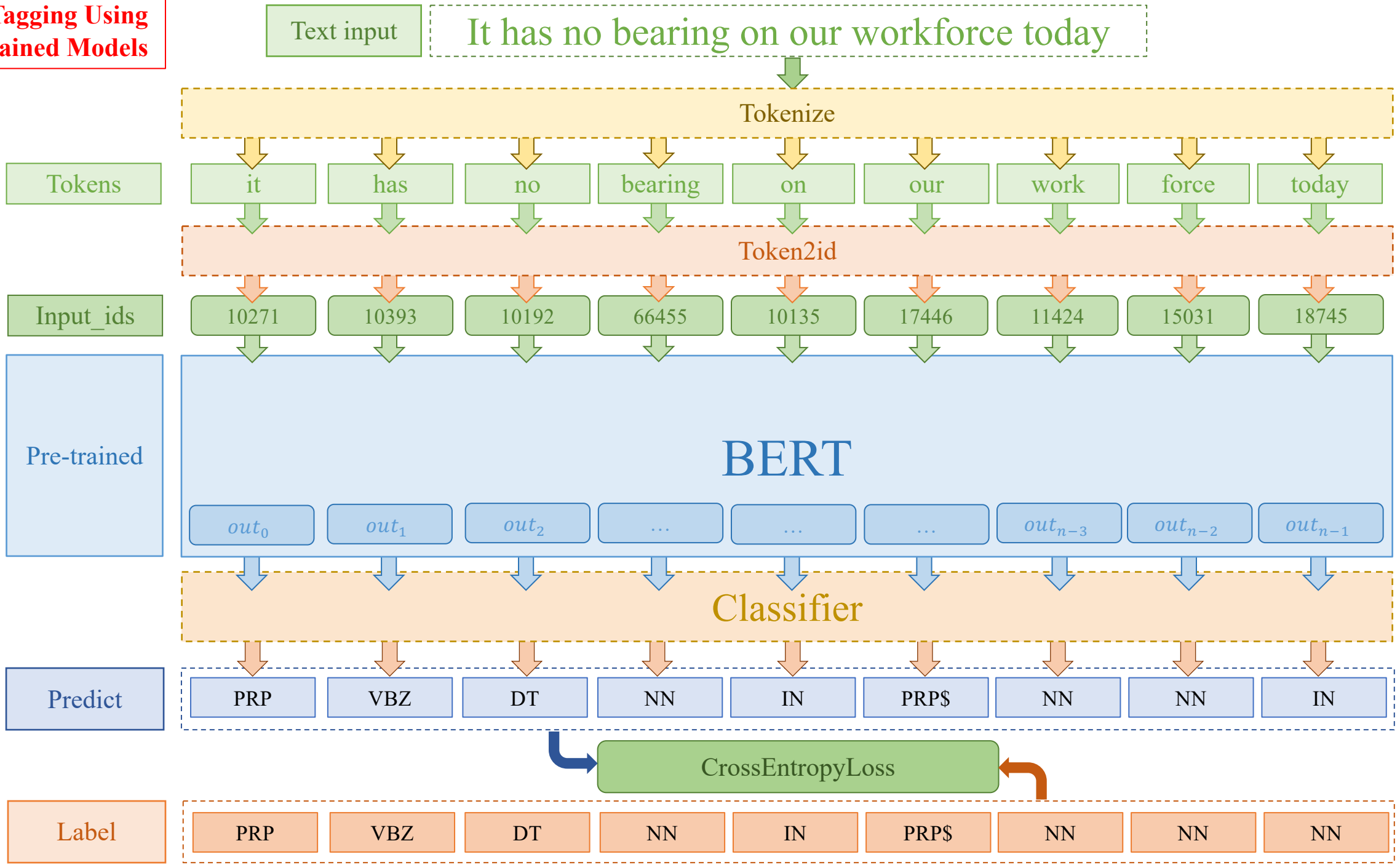
```
torch.Size([1, 4, 5])
```

```
for _ in range(30):
    optimizer.zero_grad()
    outputs = model(input_data)
    loss = criterion(outputs,
                     label_data)
    print(loss.item())
    loss.backward()
    optimizer.step()
```

```
# [[0, 1, 0, 2, 0],
#  [0, 1, 2, *, *]]

outputs = model(input_data)
o_softmax = torch.softmax(outputs,
                          axis=1)
o_softmax.argmax(axis=1)
```

```
tensor([[0, 1, 0, 2, 0],
        [0, 1, 0, 1, 0]])
```

POS Tagging Using Pre-trained Models



Outline

SECTION 1

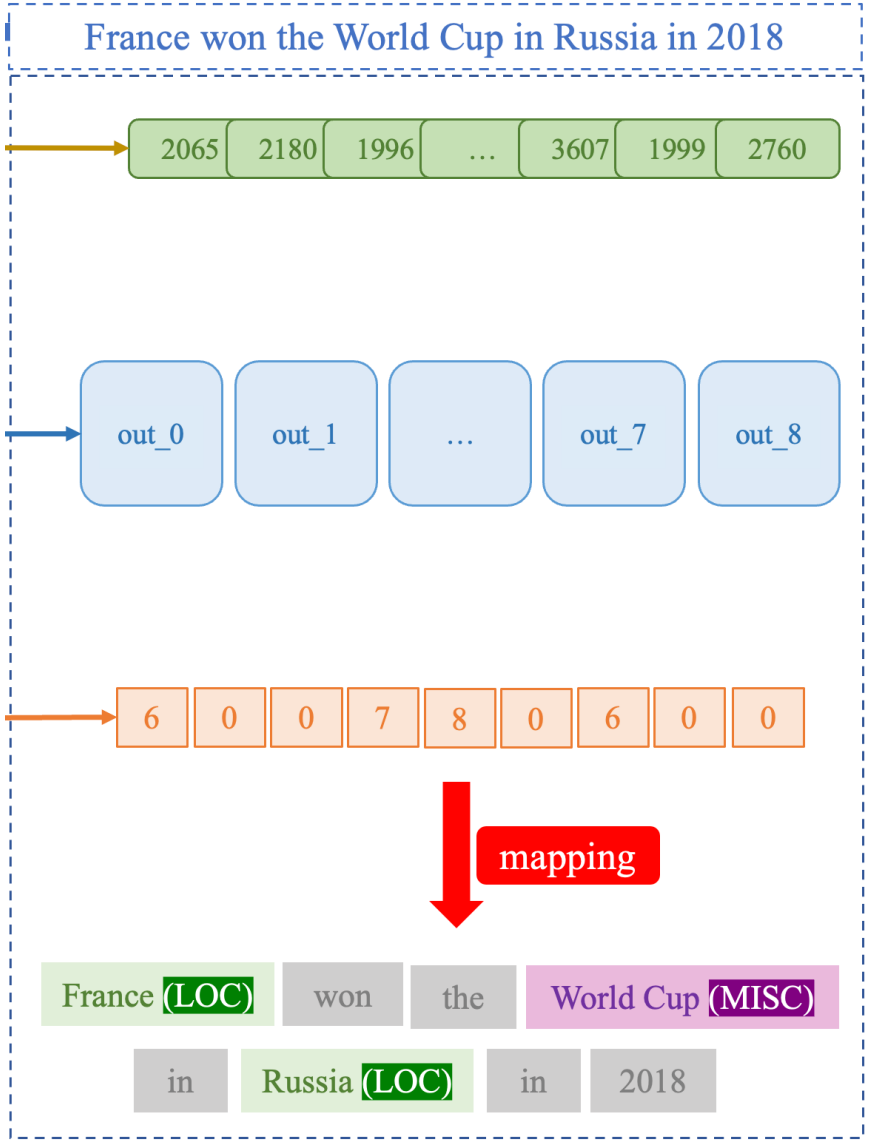
Review

SECTION 2

POS Tagging

SECTION 3

Named Entity Recognition



Named Entity Recognition

❖ Introduction

Conll2003 dataset for Named-Entity Recognition

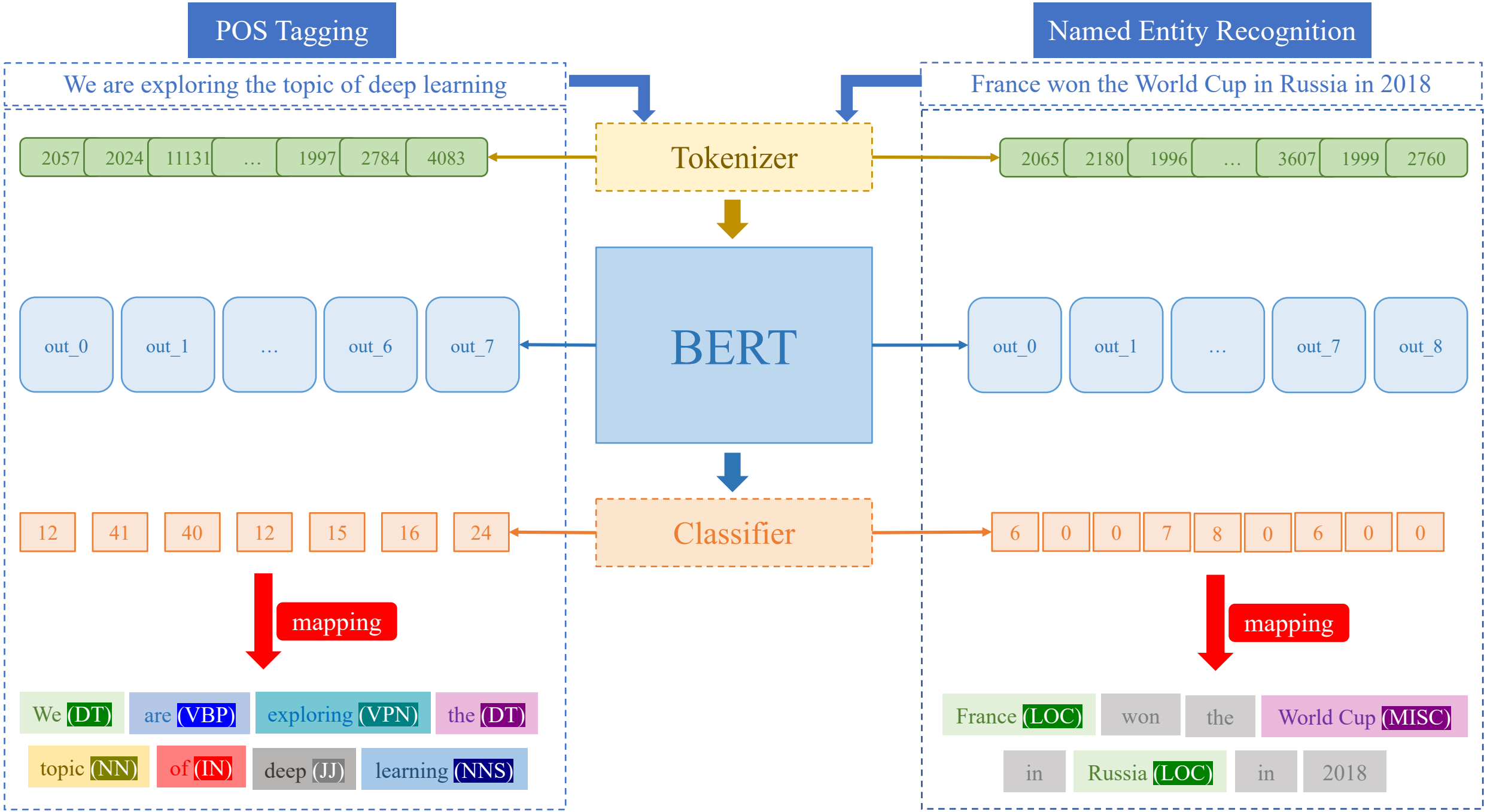
Num_classes = 9

Train	Val	Test
14041	3250	3453

0	O	Out-of-class
1	B-PER	Begin-Person
2	I-PER	In-Person
3	B-ORG	Begin-Organization
4	I-ORG	In-Organization
5	B-LOC	Begin-Location
6	I-LOC	In-Location
7	B-MISC	Begin-Miscellaneous
8	I-MISC	In-Miscellaneous

Example

Input tokens	["BCH", "in", "the", "hive", "of", "Chilean", "pensions"]
Label	[”B-ORG", ”O", ”O", ”O", ”O", ”B-MISC", ”O"]
Label-encoded	[3, 0, 0, 0, 0, 7, 0]


POS Tagging
Named Entity Recognition
We are exploring the topic of deep learning
France won the World Cup in Russia in 2018
Tokenizer
2057 2024 11131 … 1997 2784 4083
2065 2180 1996 … 3607 1999 2760
BERT
out_0 out_1 … out_6 out_7
out_0 out_1 … out_7 out_8
Classifier
12 41 40 12 15 16 24
6 0 0 7 8 0 6 0 0
mapping
mapping
We (DT) are (VBP) exploring (VPN) the (DT)
topic (NN) of (IN) deep (JJ) learning (NNS)
France (LOC) won the World Cup (MISC)
in Russia (LOC) in 2018

Step-by-step Examples

Named Entity Recognition

Doc	Label
karpathy is working in openai	[0, 4, 4, 4, 2]
geoffrey hinton is from canada	[0, 1, 4, 4, 2]

building dictionary

index	word
0	[UNK]
1	[pad]
2	is
3	canada
4	from
5	geoffrey
6	hinton
7	in
8	karpathy
9	openai
10	working

Dictionary

karpathy	8	[0.7109, -1.2178, -1.5470, -1.2587]
is	2	[0.5303, 0.7931, -1.1894, 0.1906]
working	10	[-0.2059, 1.3111, -1.2398, -1.0455]
in	7	[-0.1117, 1.2757, -0.3398, 0.5976]
openai	9	[-0.4392, 0.5843, -0.7790, 0.2032]

Sample 1 → Sample 1 _ Embedding

0	[-1.5755, 0.0146, 0.2361, 0.3852]
1	[0.2267, -1.1683, 0.0791, -1.3988]
2	[0.5303, 0.7931, -1.1894, 0.1906]
3	[0.0649, -0.0649, 2.3004, 0.3508]
4	[0.4401, -0.1977, 1.1706, -0.4241]
5	[-0.9880, 1.1651, -0.7740, -0.5781]
6	[-0.1220, 0.3313, 0.6327, -0.3742]
7	[-0.1117, 1.2757, -0.3398, 0.5976]
8	[0.7109, -1.2178, -1.5470, -1.2587]
9	[-0.4392, 0.5843, -0.7790, 0.2032]
10	[-0.2059, 1.3111, -1.2398, -1.0455]

Embedding

vocab size = 11
sequence length = 5
num of classes = 5+1

ID	Meaning
0	B-Person
1	I-Person
2	B-Org./Location
3	I-Org./Location
4	Others
5	<padding>

Label Codes

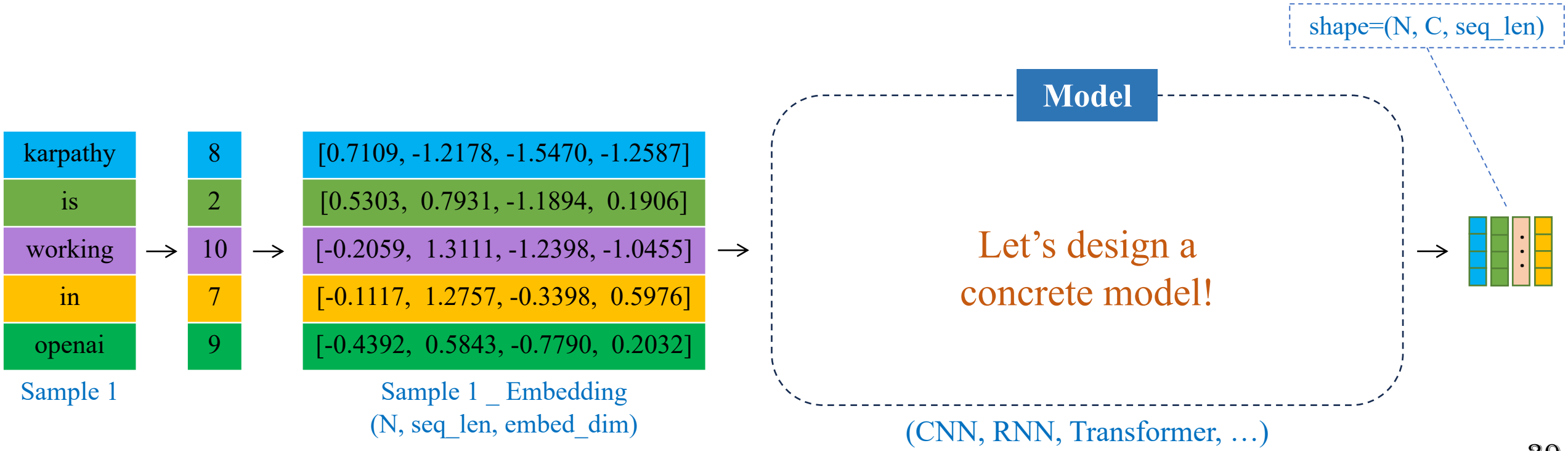
Named Entity Recognition

Doc	Label
karpathy is working in openai	[0, 4, 4, 4, 2]
geoffrey hinton is from canada	[0, 1, 4, 4, 2]

vocab size = 11
sequence length = 5
num of classes = 5+1

ID	Meaning
0	B-Person
1	I-Person
2	B-Org./Location
3	I-Org./Location
4	Others
5	<padding>

Label Codes



[0.7109, -1.2178, -1.5470, -1.2587]

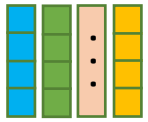
[0.5303, 0.7931, -1.1894, 0.1906]

[-0.2059, 1.3111, -1.2398, -1.0455]

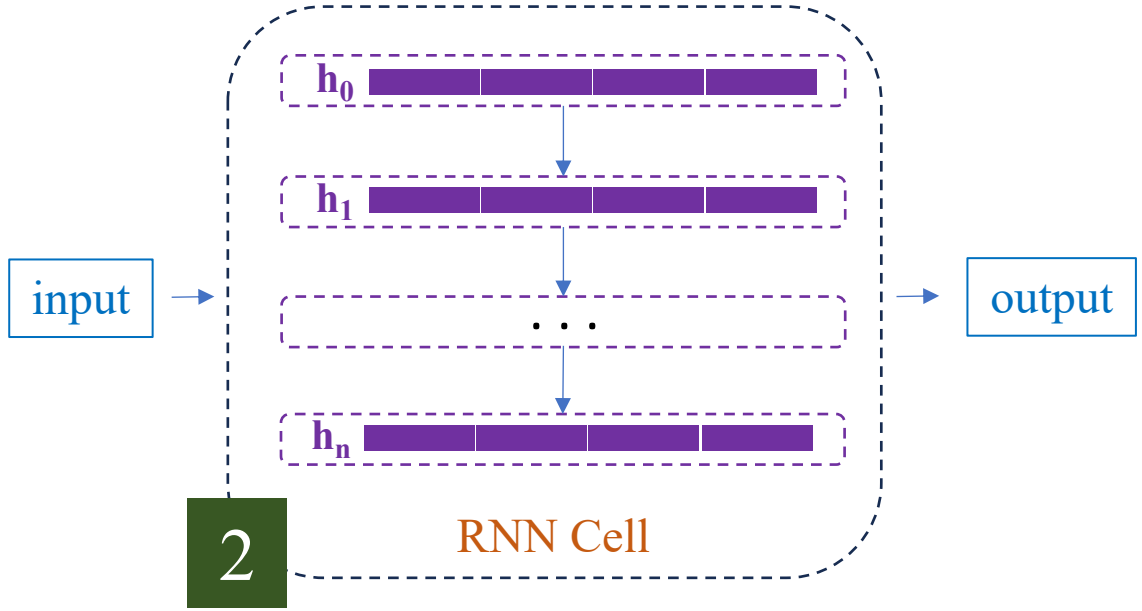
[-0.1117, 1.2757, -0.3398, 0.5976]

[-0.4392, 0.5843, -0.7790, 0.2032]

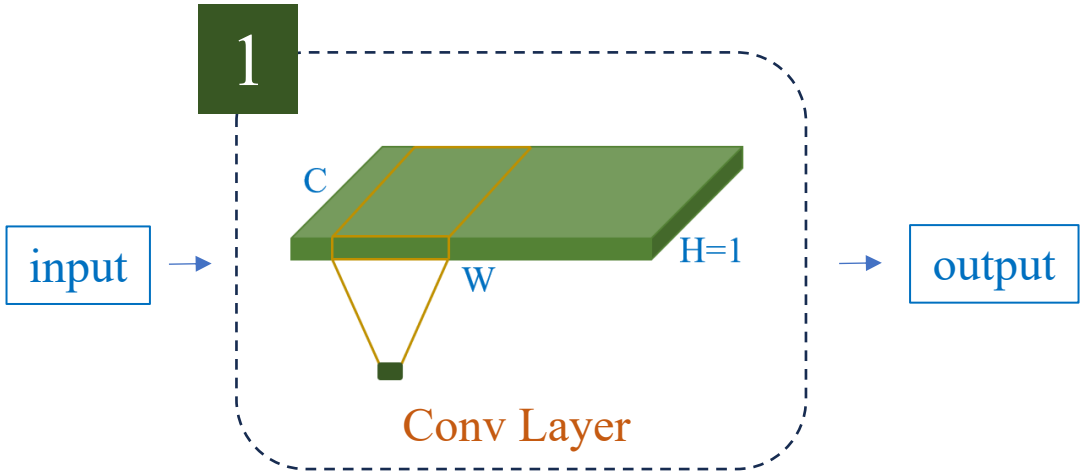
Input: Sample 1 _ Embedding
(N, seq_len, embed_dim)

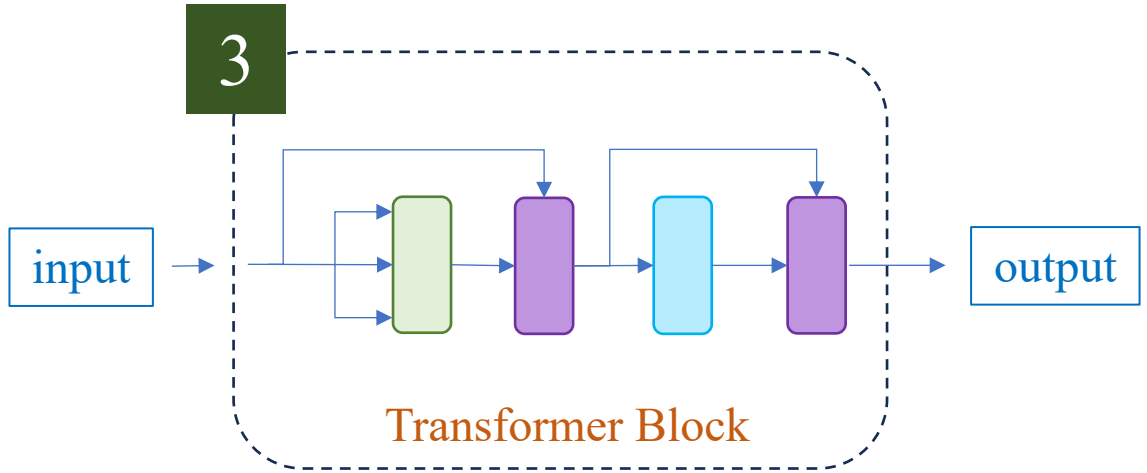


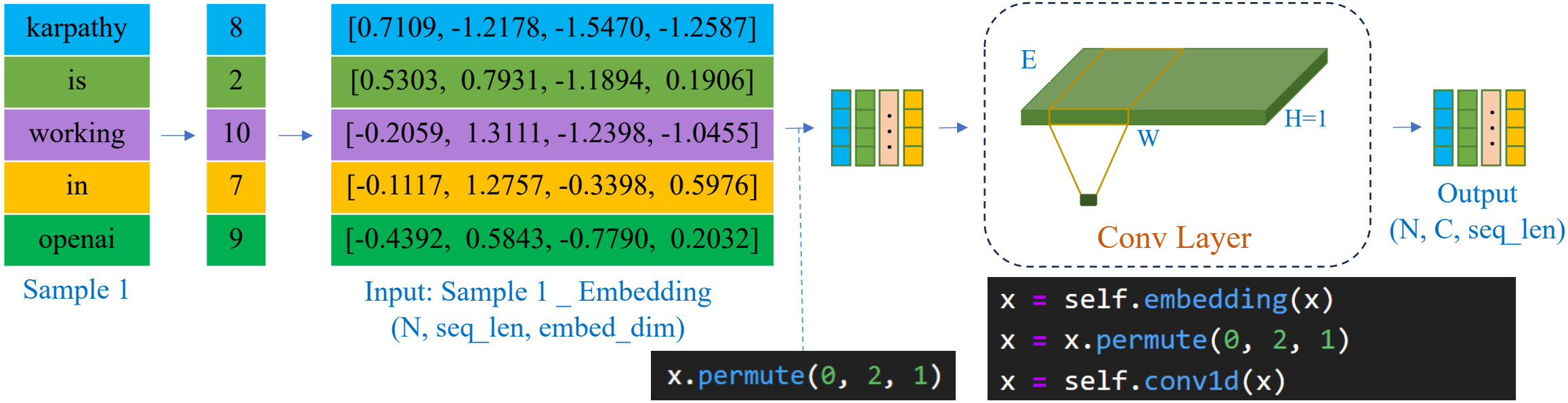
Output
shape=(N, C, seq_len)

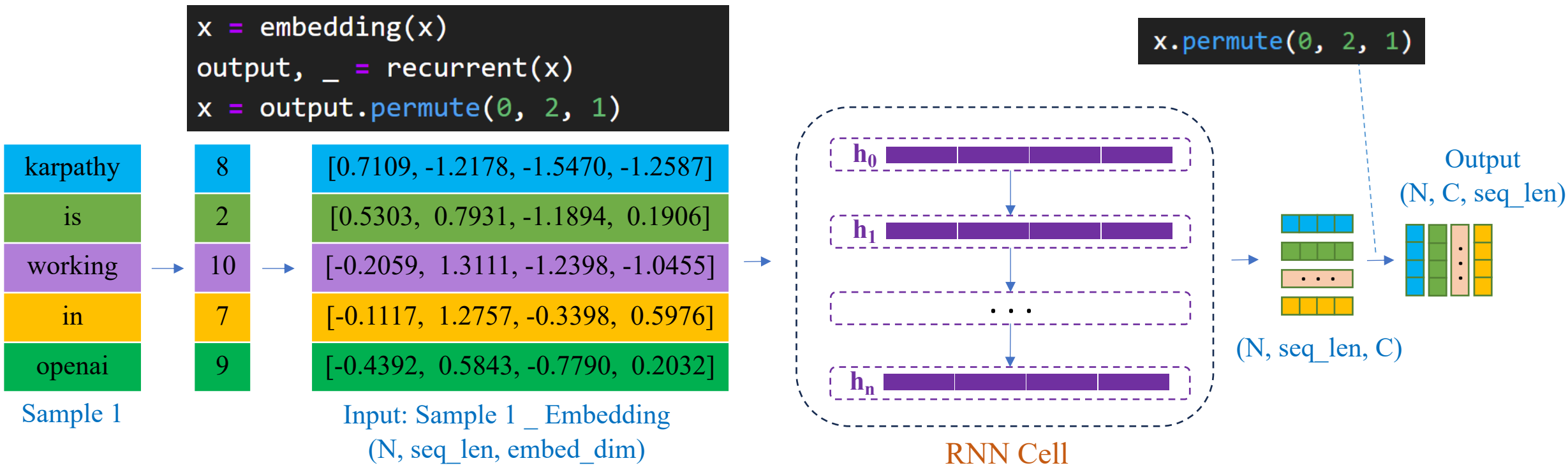


Which ones are feasible?





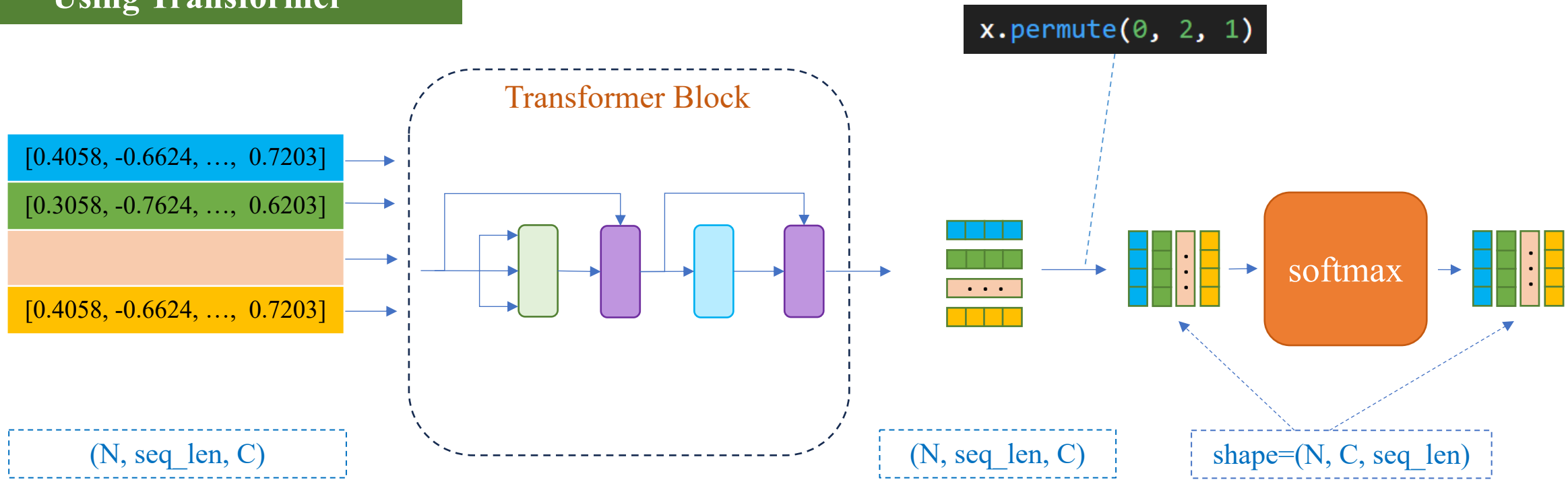

karpathy
is
working
in
openai
8
2
10
7
9
[0.7109, -1.2178, -1.5470, -1.2587]
[0.5303, 0.7931, -1.1894, 0.1906]
[-0.2059, 1.3111, -1.2398, -1.0455]
[-0.1117, 1.2757, -0.3398, 0.5976]
[-0.4392, 0.5843, -0.7790, 0.2032]
Sample 1
Input: Sample 1 _ Embedding
(N, seq_len, embed_dim)
E
H=1
W
Conv Layer
Output
(N, C, seq_len)
x.permute(0, 2, 1)
x = self.embedding(x)
x = x.permute(0, 2, 1)
x = self.conv1d(x)


x = embedding(x)
output, _ = recurrent(x)
x = output.permute(0, 2, 1)
x.permute(0, 2, 1)
karpathy
is
working
in
openai
8
2
10
7
9
[0.7109, -1.2178, -1.5470, -1.2587]
[0.5303, 0.7931, -1.1894, 0.1906]
[-0.2059, 1.3111, -1.2398, -1.0455]
[-0.1117, 1.2757, -0.3398, 0.5976]
[-0.4392, 0.5843, -0.7790, 0.2032]
Sample 1
Input: Sample 1 _ Embedding
(N, seq_len, embed_dim)
h0
h1
hn
RNN Cell
(N, seq_len, C)
Output
(N, C, seq_len)

Designing a Model for POS Tagging

```
embedding = nn.Embedding(vocab_size, 4)
transformer = TransformerBlock(4, 1, 4)
# embed_dim, num_heads, ff_dim

# forward
x = self.embedding(x)
x = self.transformer(x, x, x)
x =  x.permute(0, 2, 1)
```

Using Transformer



Custom Dataset in Pytorch

Create a Custom Dataset

__init__(self, …) function:
Khởi tạo các thuộc tính/biến

__len__(self) function:
Trả về độ dài của dataset

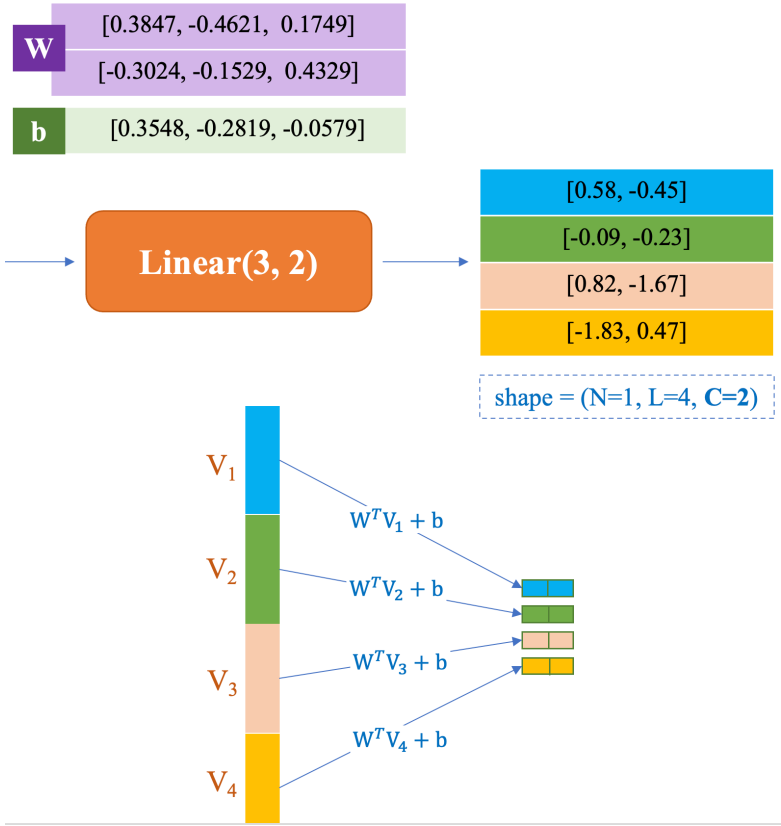
__getitem__(selfm idx) function:
Xử lý một sample và trả về x và y

```python
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
sequence_length = 5

sample1 = 'We are learning AI'
sample2 = 'AI is a CS topic'
sentences = [sample1, sample2]
labels = [0, 1]
```

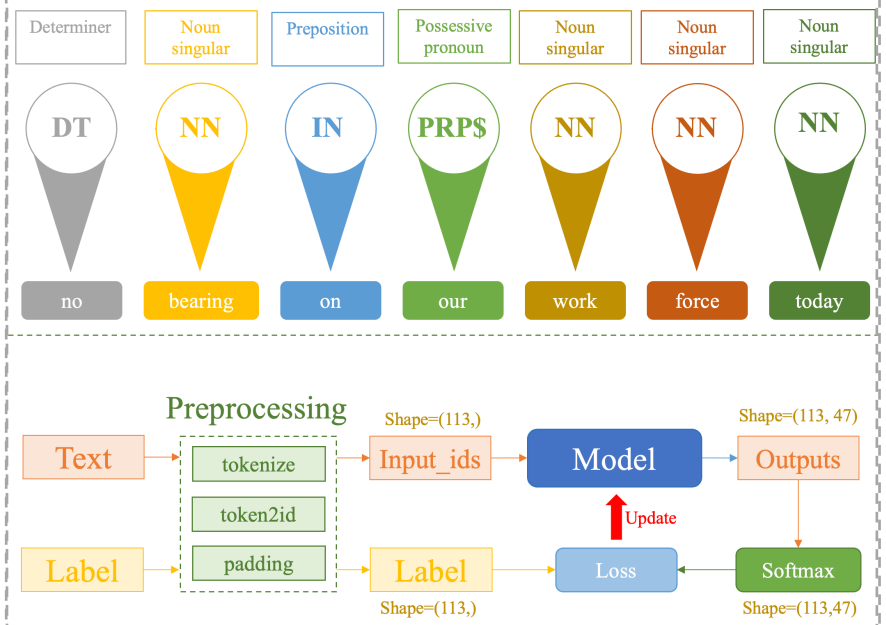
```python
from torch.utils.data import Dataset

class MyDataset(Dataset):
    def __init__(self, sentences, labels, tokenizer, max_len):
        super().__init__()
        #...

    def __len__(self):
        return len(self.sentences)

    def __getitem__(self, idx):
        #...

        return tokens, sentence_label
```

Summary

Review



POS Tagging



Named Entity Recognition