AI VIETNAM
All-in-One Course

Text Classification using Neural Network

Nguyen Quoc Thai

Year 2023

AI VIET NAM
@aivietnam.edu.vn

CONTENT

1 - Introduction

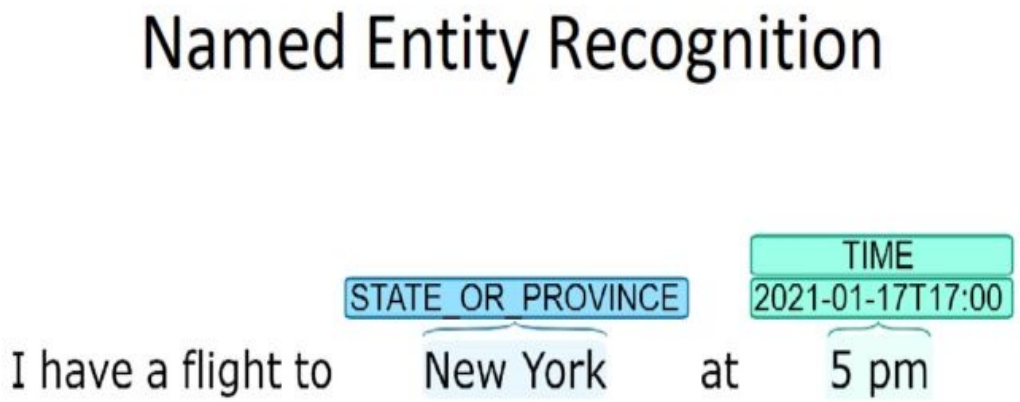
Text Classification

- **Input**
  - A fixed set of classes **C** = $\{c_1, c_2, \ldots, c_N\}$
  - A training set of **M** hand-labeled documents: $(d_1, c_1), \ldots, (d_M, c_N)$
  - A document **d**
- **Output**
  - A learned classifier **d** => **c (C)**

1 - Introduction

Text Classification

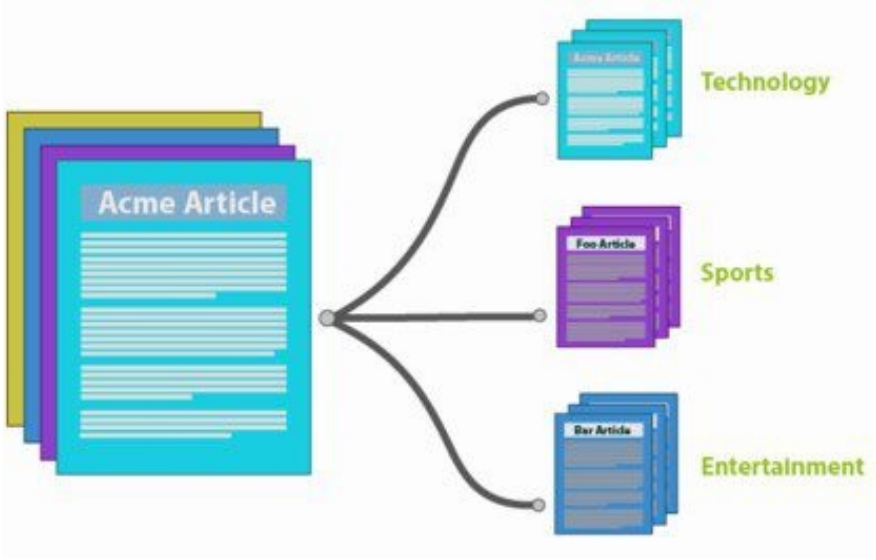
- **Token-level Classification**
  - Sequence labeling: Word Segmentation, POS Tagging, NER,…

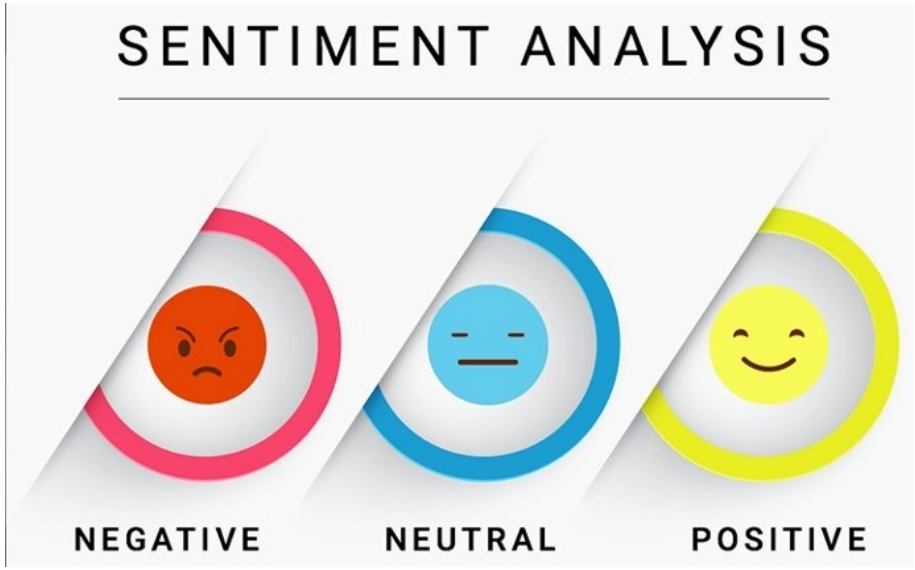


1 - Introduction

Text Classification

- **Token-level Classification**
  - Sequence labeling: Word Segmentation, POS Tagging, NER,…
- **Document-level Classification**
  - Sentiment Analysis

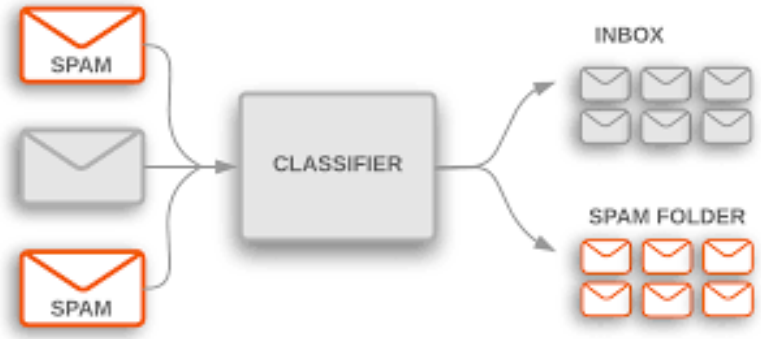




1 - Introduction

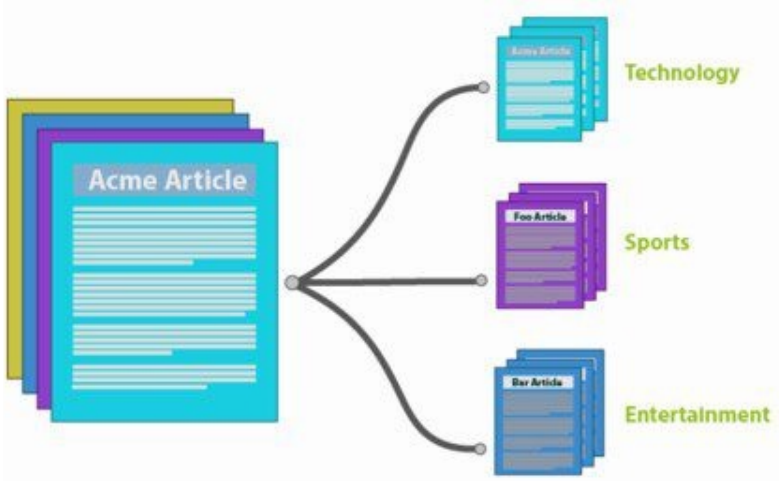
Text Classification

- **Binary Classification**
- **Multiclass Classification**
- **Multilabel Classification**

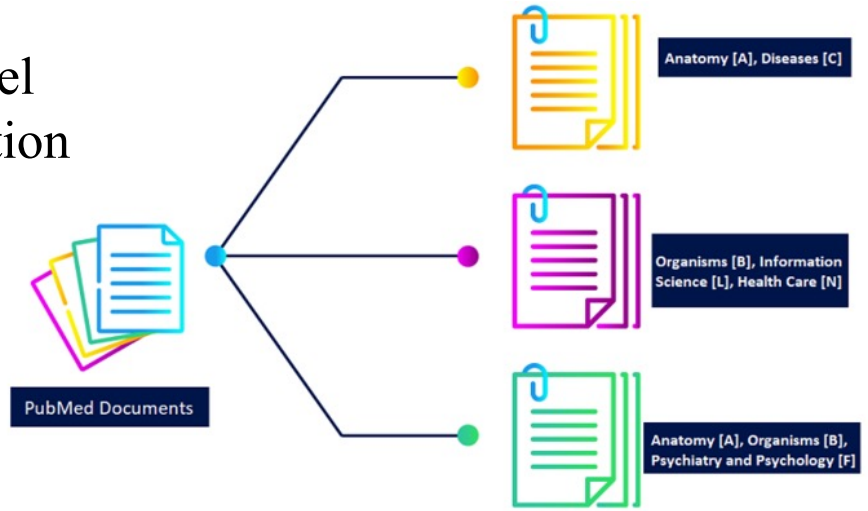


Binary Classification

Multiclass Classification



Multilabel Classification



1 - Introduction

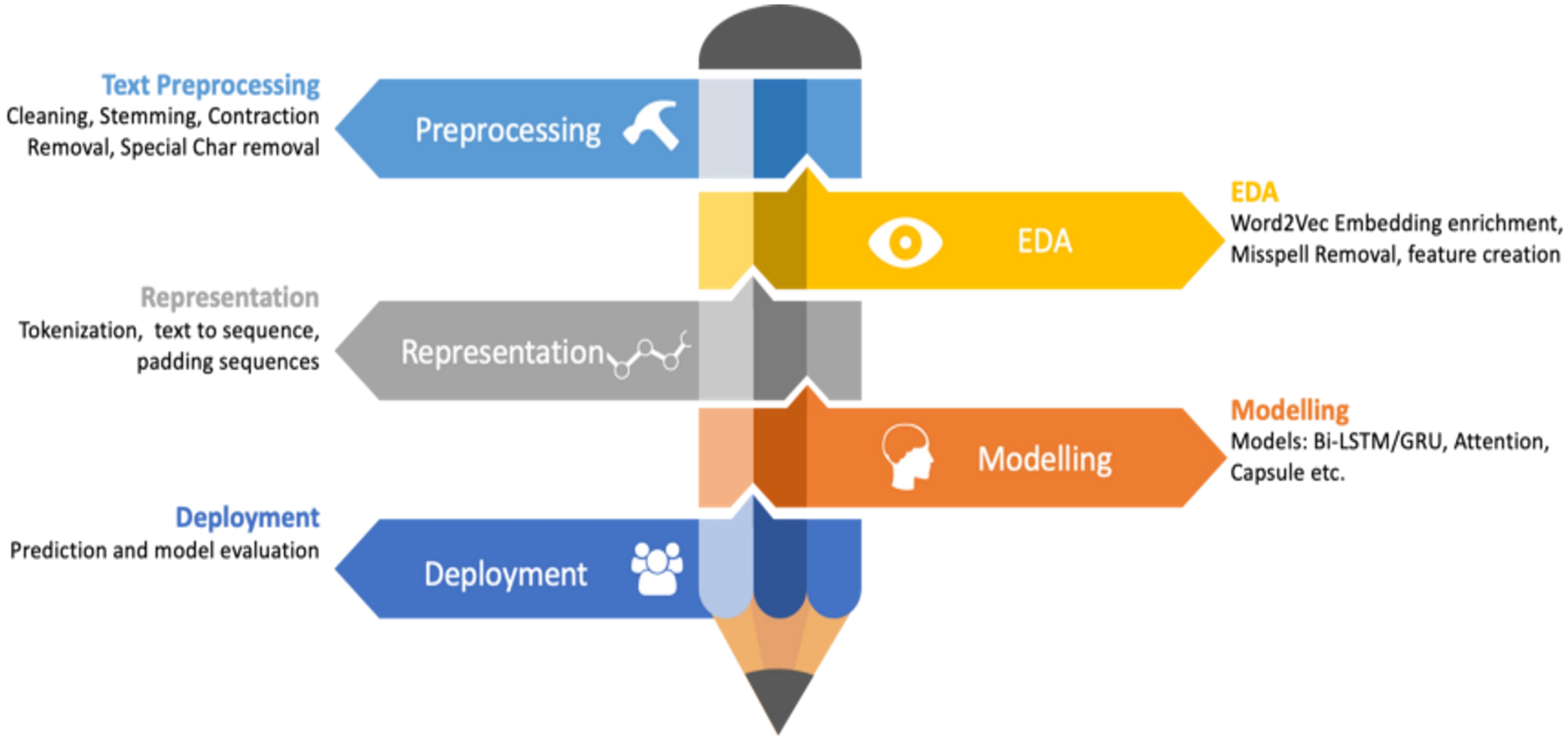
Text Classification (Sentiment Analysis)

- **NTC-SCV Dataset**
- Document-level Classification
- Binary Classification (2 Classes: Positive, Negative)

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

1 - Introduction

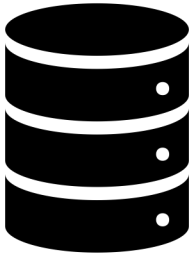
Pipeline



2 - Preprocessing

Text Preprocessing

❖ Language Detection



Language Detector

langid library

Vietnamese Language

Quán này thấy khá nhiều người bảo mình nên mình đã đi ăn thử , nhưng thực_sự ăn xong thấy không được ngon. 👍👍 </p>

Mình được 1 cô bạn giới_thiệu đến đây , tìm địa_chỉ khá dễ . Menu nước uống chất khỏi nói . https://foody.com

Other Language

Visiting_Da_Nang frequently but this is the first time I have found a coffee shop which has a creative design (korean style)

The room is cheap ! ! ! ! It ' s near the city center . The staff is so nice : - D 👍👍👍👍👍👍\n

2 - Preprocessing

Text Preprocessing

- **Language Detection**
- **Text Cleaning**

1 – Removal URLs, HTML Tags

2 – Removal punctuations, digits

3 – Removal emoticons, flags,…

4 – Normalize whitespace

5 – Lowercasing

Vietnamese Language

Quán này thấy khá nhiều người bảo mình nên mình đã đi ăn thử , nhưng thực_sự ăn xong thấy không được ngon. 👍👍 </p>

Mình được 1 cô bạn giới_thiệu đến đây , tìm địa_chỉ khá dễ . Menu nước uống chất khỏi nói . https://foody.com

3 - Representation

Numeric Representation

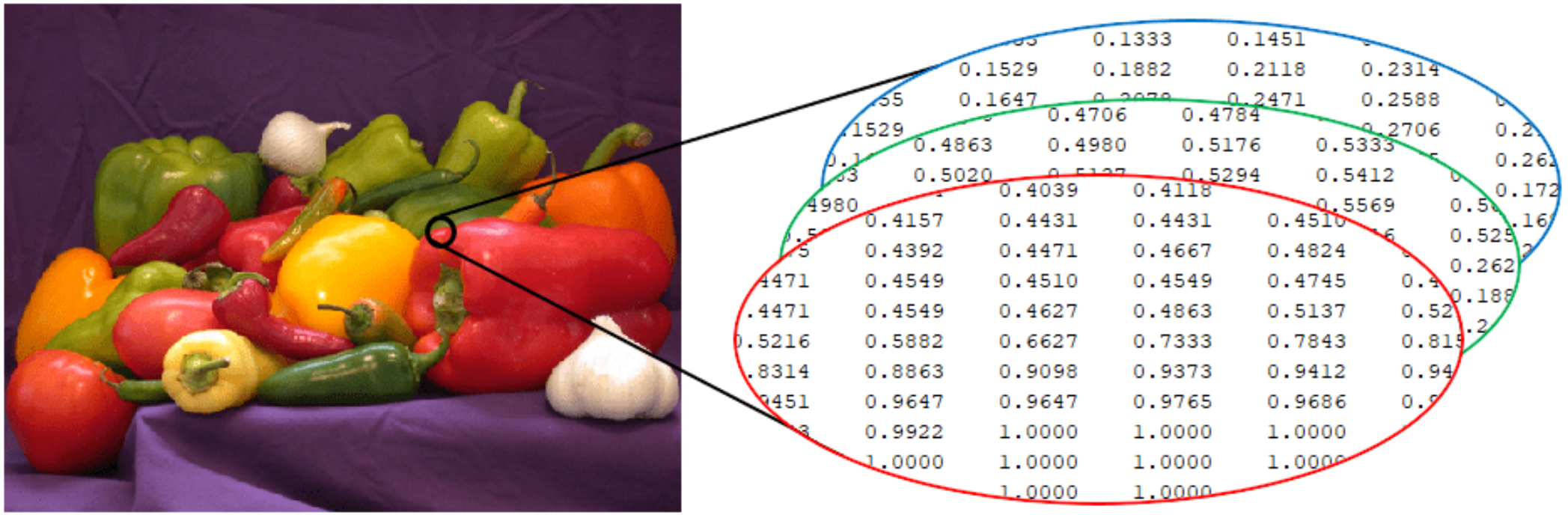
Petal_Length	Petal_Width	Label
1.4	0.2	0
1.5	0.2	0
3	1.1	1
4.1	1.3	1

Convert to Vector

$$x = \begin{bmatrix} 1 & 1.4 & 0.2 \\ 1 & 1.5 & 0.2 \\ 1 & 3.0 & 1.1 \\ 1 & 4.1 & 1.3 \end{bmatrix} \qquad y = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

3 - Representation

Numeric Representation

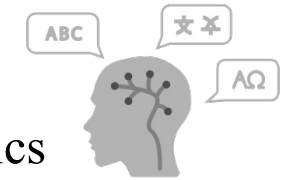


3 - Representation

Numeric Representation

Natural Language Understanding (NLU)

a computer's ability to understand language



- Syntax
- Semantics
- Phonology
- Pragmatics
- Morphology

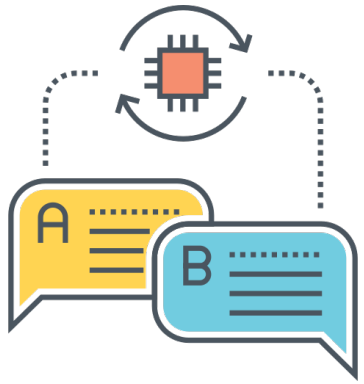
I go to school

Convert

[0 0 0 0 1 0 1 0 1]

Natural Language Generation (NLG)

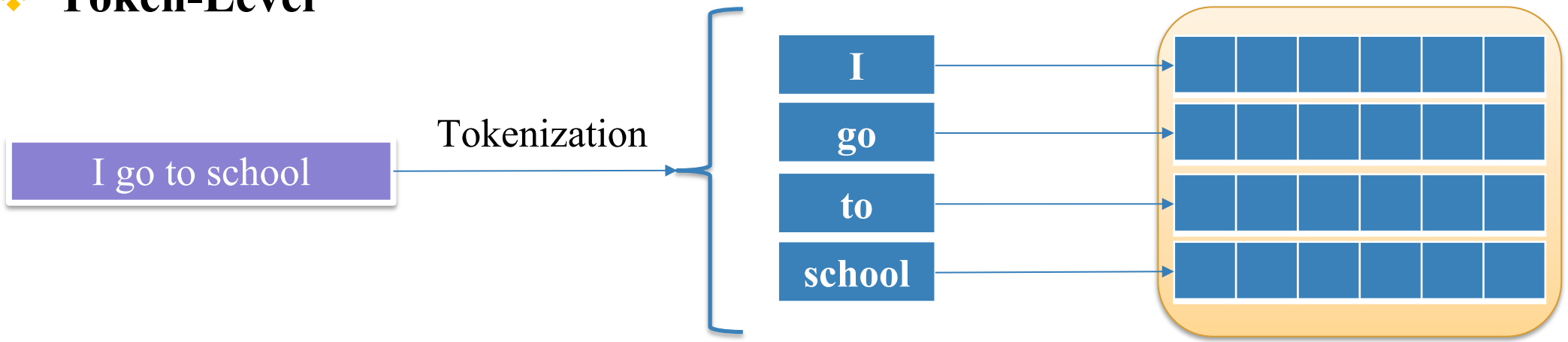
generate natural language by a computer



3 - Representation

Numeric Representation

Token-Level

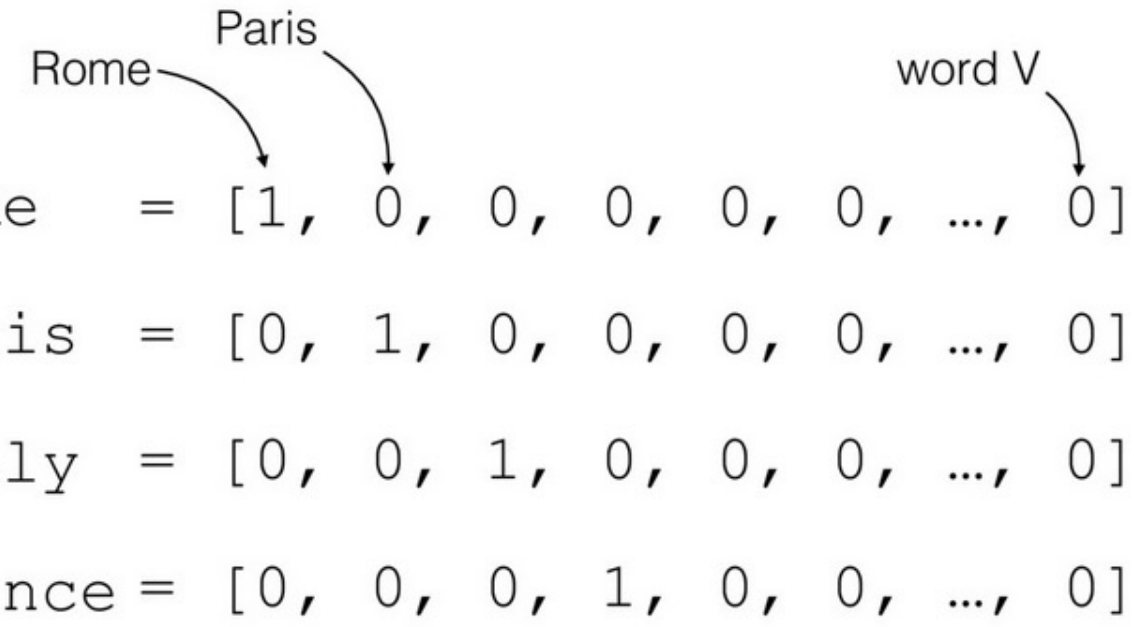


Document-Level

3 - Representation

Numeric Representation

- Basic Representation
  - One-hot Encoding
  - Bag of Words (BoW)
  - Bag of N-grams
  - TF-IDF



3 - Representation

Numeric Representation

Distributed Representation (Dense Vector)

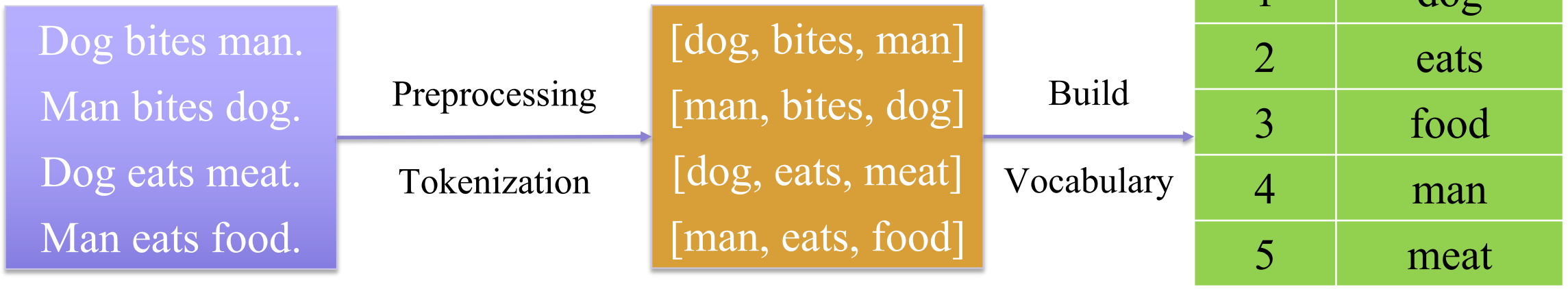
- Word2Vec
- Glove
- Fasttext
- ELMO

man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

3 - Representation

One-hot Encoding

- Token-Level
- Represented by a V-dimensional binary vector of 0s and 1s
  - All 0s barring the index, index = w_{id}
  - At this index, put 1



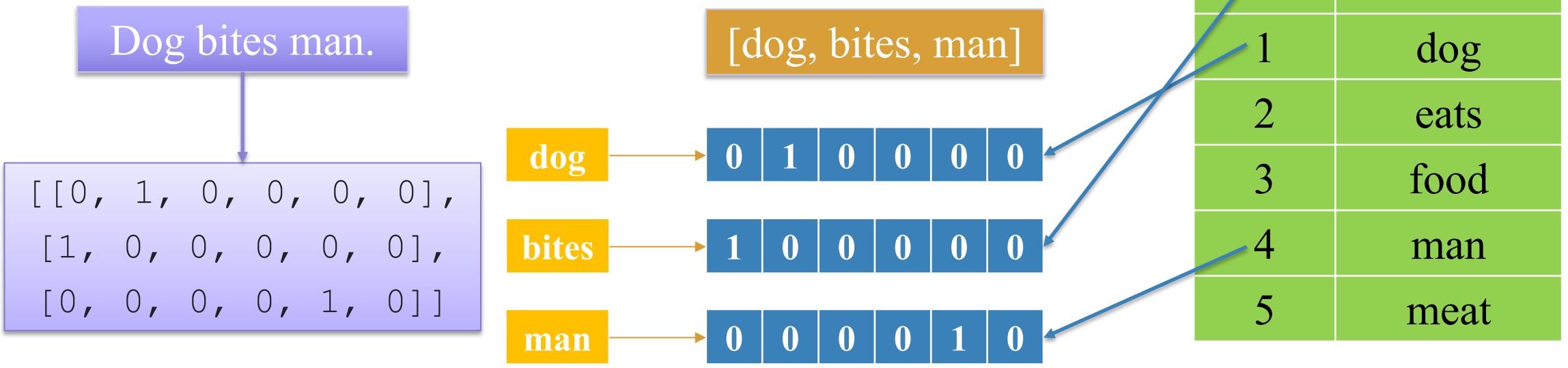
Vocabulary

IDX	Token
0	bites
1	dog
2	eats
3	food
4	man
5	meat

3 - Representation

One-hot Encoding

- Token-Level
- Represented by a V-dimensional binary vector of 0s and 1s
  - All 0s barring the index, index = w_{id}
  - At this index, put a 1

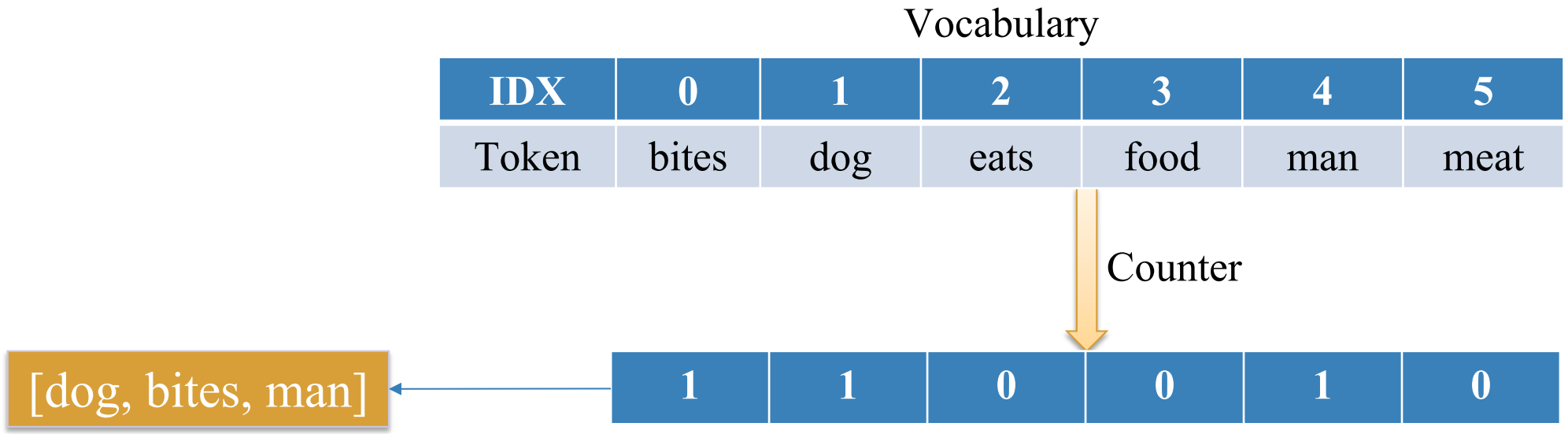


3 - Representation

Bag of Words (BoW)

- Document-Level: Consider text as a bag (collection) of words
- Represented by a V-dimensional

Use: the number of occurrences of the word in the document

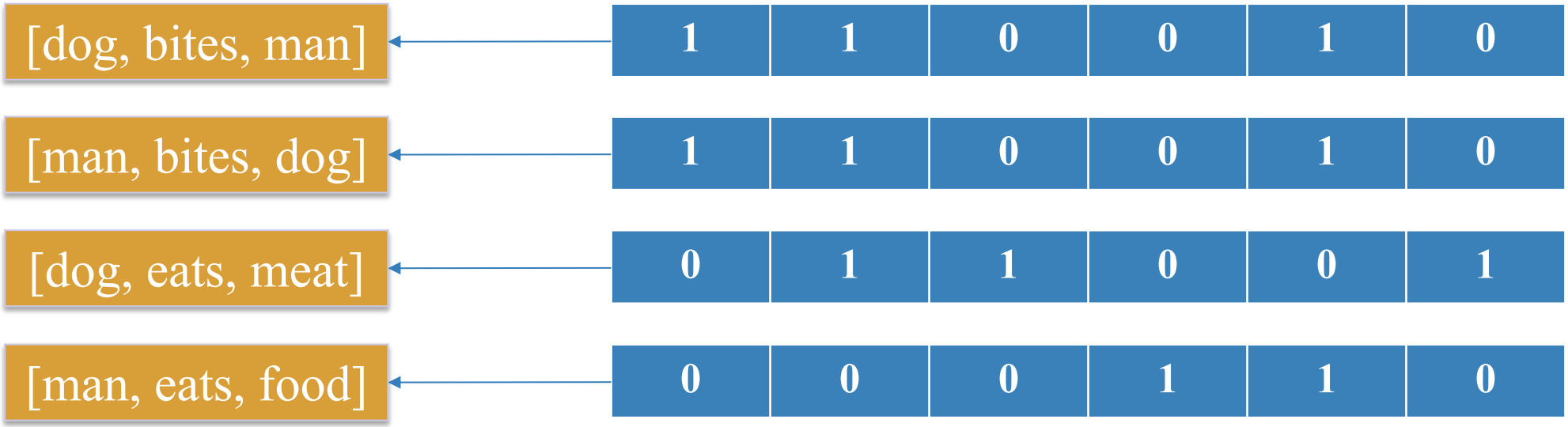


3 - Representation

Bag of Words (BoW)

- Document-Level: Consider text as a bag (collection) of words
- Represented by a V-dimensional

Use: the number of occurrences of the word in the document



3 - Representation

Index-based Representation

- Document-Level:
- Represented by a N-dimensional (length of sentence)

The ith component off the vector, $i = w_{id}$ is index of the word w occurs in vocabulary

- Attention to the order of words in the sentence

3 - Representation

Index-based Representation

- Use word-based tokenization
- Build a vocabulary

IDX	Token
0	<unk>
1	dog
2	man
3	bites
4	eats
5	food
6	meat

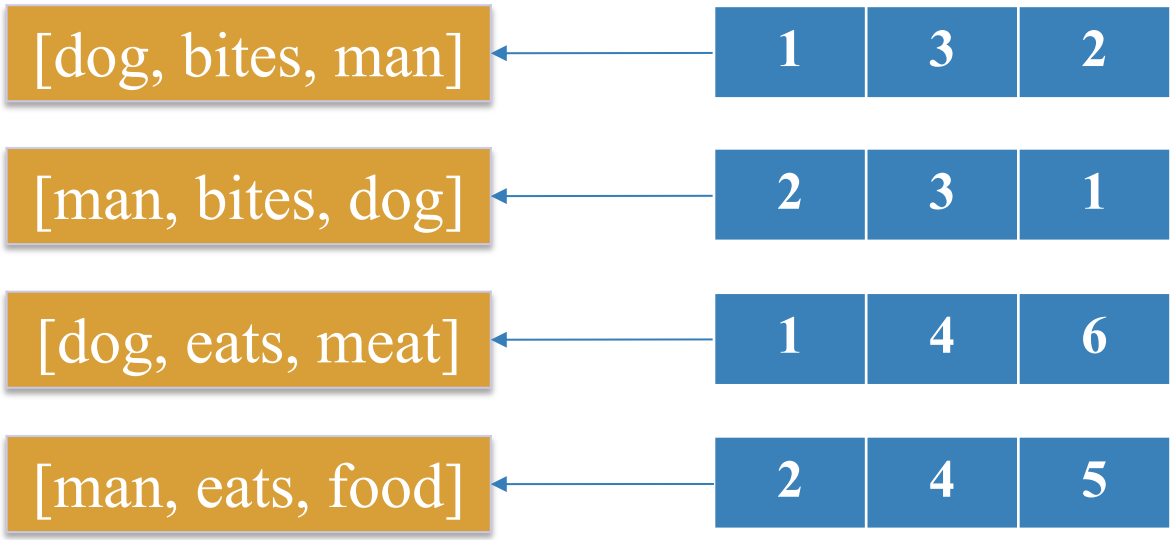
```python
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator

data = ["dog bites man", "man bites dog", "dog eats meat", "man eats food"]

# Define the max vocabulary size
vocab_size = 8

# Define tokenizer function
tokenizer = get_tokenizer('basic_english')

# Create a function to yield list of tokens
def yield_tokens(examples):
    for text in examples:
        yield tokenizer(text)

# Create vocabulary
vocab = build_vocab_from_iterator(
    yield_tokens(data),
    max_tokens=vocab_size,
    specials=["<unk>"]
)
vocab.set_default_index(vocab["<unk>"])

vocab.get_stoi()
```

```
{'food': 5, 'eats': 4, 'bites': 3, 'meat': 6, 'man': 2, 'dog': 1, '<unk>': 0}
```

3 - Representation

Index-based Representation

- Use word-based tokenization
- Build a vocabulary
- Convert text into features



Vocabulary

IDX	Token
0	<unk>
1	dog
2	man
3	bites
4	eats
5	food
6	meat

3 - Representation

Index-based Representation

- Use word-based tokenization
- Build a vocabulary
- Convert text into features


```
vocab(tokenizer("dog bites man"))
```

```
[1, 3, 2]
```

```
vocab(tokenizer("dog and man"))
```

```
[1, 0, 2]
```

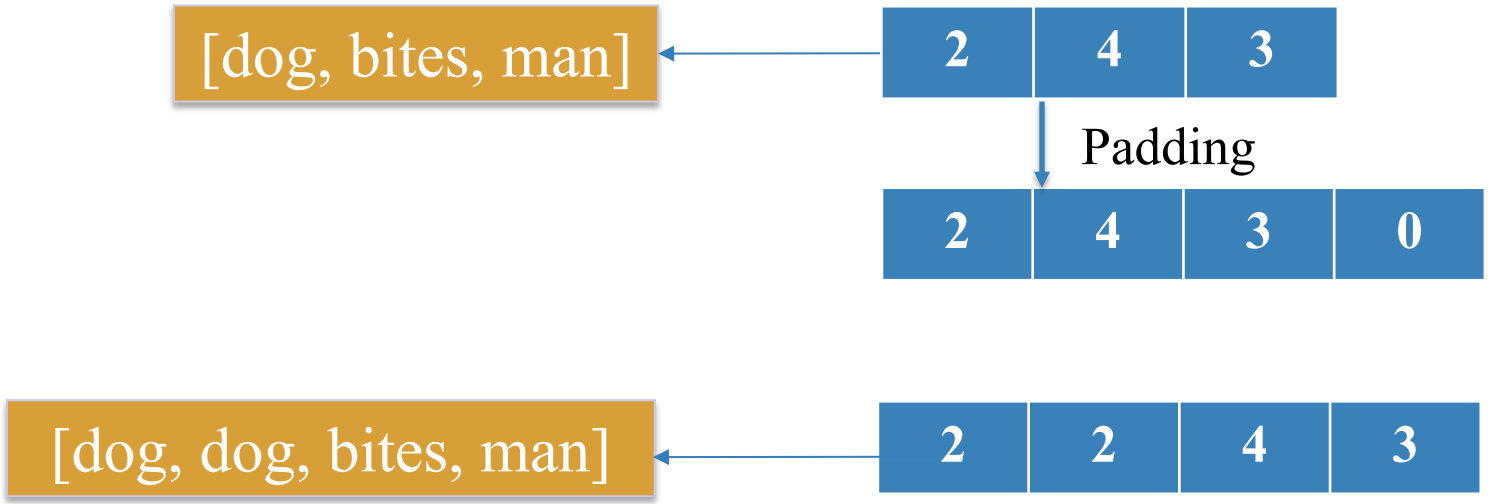
Vocabulary

IDX	Token
0	<unk>
1	dog
2	man
3	bites
4	eats
5	food
6	meat

3 - Representation

Index-based Representation

- Padding all sentences => the same length
- Append token "<pad>"



Vocabulary

IDX	Token
0	<pad>
1	<unk>
2	dog
3	man
4	bites
5	eats
6	food
7	meat

3 - Representation

Index-based Representation

- Padding all sentences => the same length
- Append token “<pad>”

```
from torchtext.transforms import PadTransform

# define padding
max_len = 4
pad_id = vocab.get_stoi()['<pad>']
pad_transform = PadTransform(max_len, pad_id)

input = torch.tensor([2, 4, 3])
padded_input = pad_transform(input)
padded_input
```

```
tensor([2, 4, 3, 0])
```

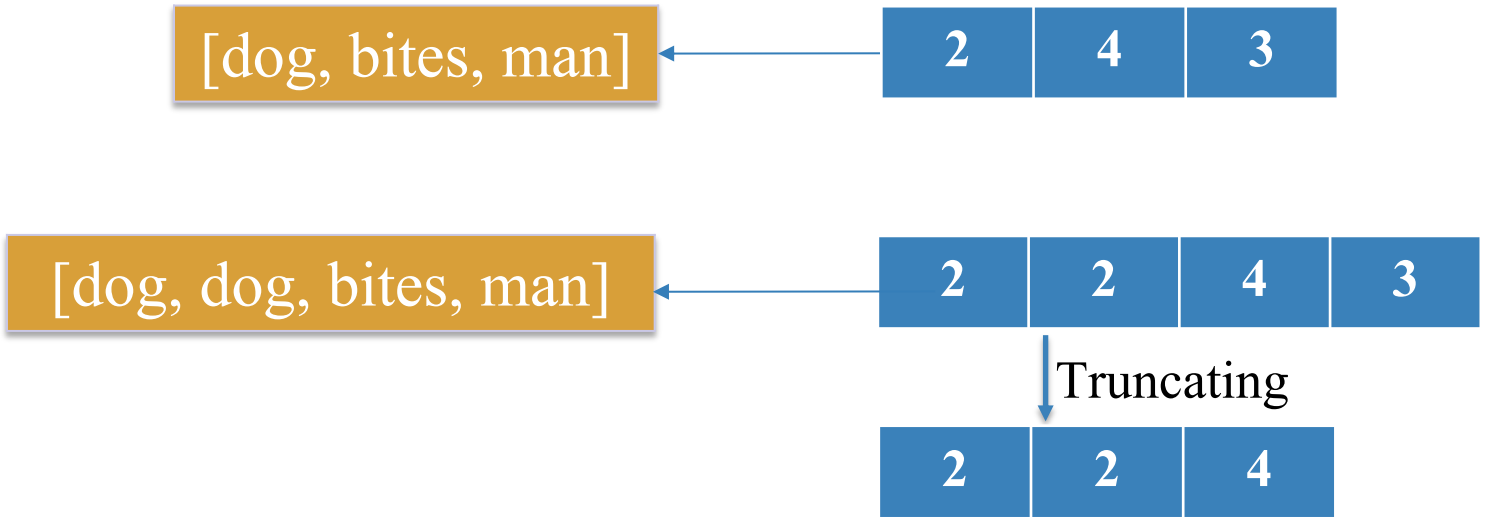
Vocabulary

IDX	Token
0	<pad>
1	<unk>
2	dog
3	man
4	bites
5	eats
6	food
7	meat

3 - Representation

Index-based Representation

- Padding all sentences => the same length
- Append token "<pad>"
- Truncating



Vocabulary

IDX	Token
0	<pad>
1	<unk>
2	dog
3	man
4	bites
5	eats
6	food
7	meat

3 - Representation

Index-based Representation

- Padding all sentences => the same length
- Append token “<pad>”
- Truncating

```
from torchtext.transforms import Truncate

# define padding
max_len = 3
pad_id = vocab.get_stoi()['<pad>']
truncater = Truncate(max_len)

input = [2, 2, 4, 3]
truncated_input = truncater(input)
truncated_input
```

```
[2, 2, 4]
```

Vocabulary

IDX	Token
0	<pad>
1	<unk>
2	dog
3	man
4	bites
5	eats
6	food
7	meat

3 - Representation

Dense Representation

- Token-level: dense vectors (low dimensional, hardly any zeros)
- Learn during training (weights)

[dog, bites, man]
[man, bites, dog]
[dog, eats, meat]
[man, eats, food]

Vocabulary

IDX	Token
0	<pad>
1	<unk>
2	dog
3	man
4	bites
5	eats
6	food
7	meat

Embedding Matrix
(Lookup Table)

0	0.1	3.1
1	0.5	2.5
2	1.3	0.6
3	0.4	0.1
4	0.7	1.4
5	2.3	1.7
6	2.5	2.5
7	0.3	1.2

Initial weights

3 - Representation

Dense Representation

- Index-based Representation
- Get vectors from embedding matrix

[dog, bites, man]
[man, bites, dog]

Input matrix
Index-based Representation

$$\begin{bmatrix} 2 & 4 & 3 \\ 3 & 4 & 2 \end{bmatrix}$$

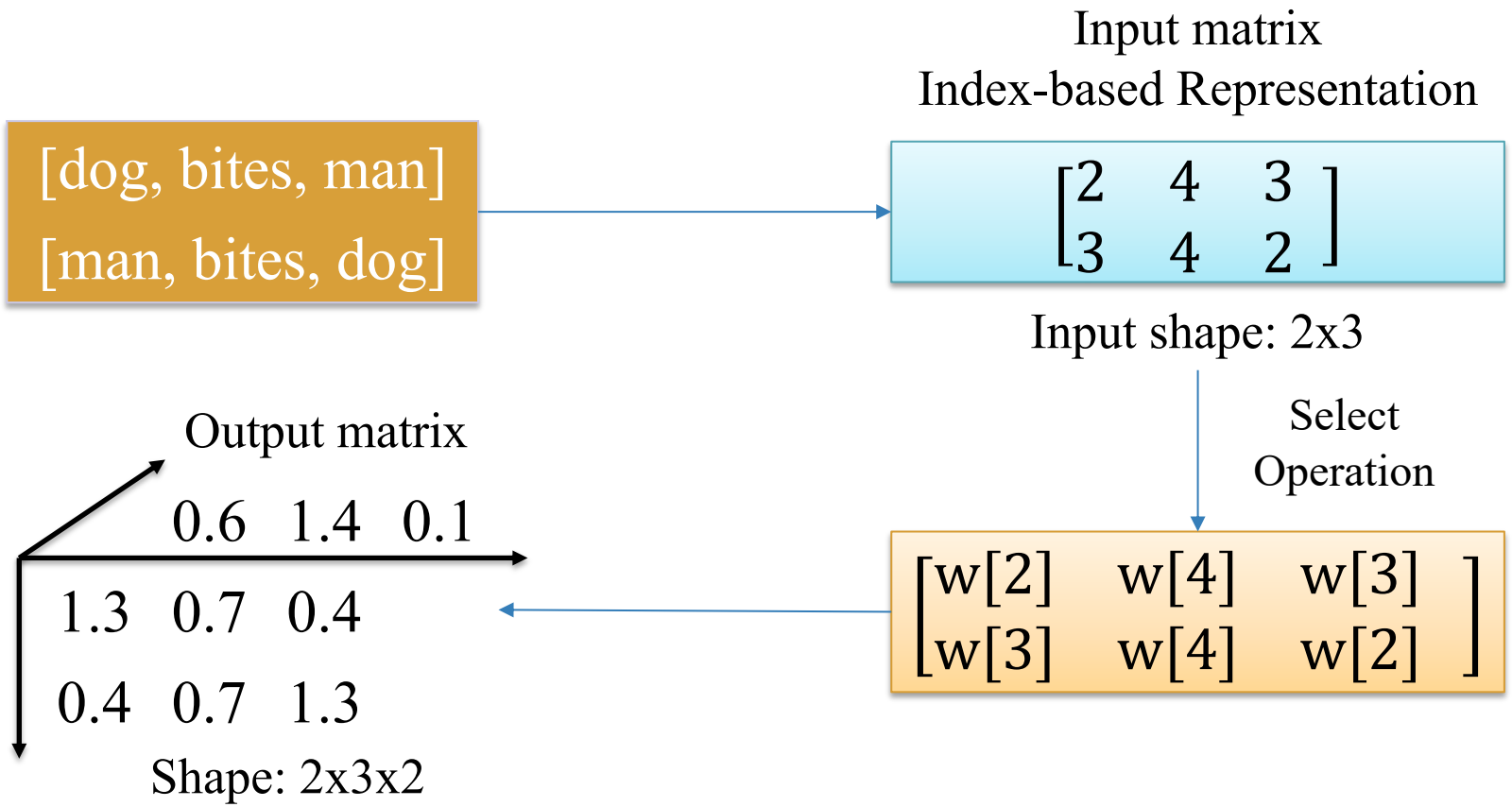
Input shape: 2x3

IDX	Token
0	<pad>
1	<unk>
2	dog
3	man
4	bites
5	eats
6	food
7	meat

3 - Representation

Dense Representation

- Index-based Representation



Embedding Matrix
(Lookup Table)

0	0.1	3.1
1	0.5	2.5
2	1.3	0.6
3	0.4	0.1
4	0.7	1.4
5	2.3	1.7
6	2.5	2.5
7	0.3	1.2

3 - Representation

Dense Representation

EMBEDDING

CLASS `torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=None, max_norm=None, norm_type=2.0, scale_grad_by_freq=False, sparse=False, _weight=None, device=None, dtype=None)` [SOURCE]

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Parameters

- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector
- **padding_idx** (*int, optional*) – If specified, the entries at `padding_idx` do not contribute to the gradient; therefore, the embedding vector at `padding_idx` is not updated during training, i.e. it remains as a fixed "pad". For a newly constructed Embedding, the embedding vector at `padding_idx` will default to all zeros, but can be updated to another value to be used as the padding vector.
- **max_norm** (*float, optional*) – If given, each embedding vector with norm larger than `max_norm` is renormalized to have norm `max_norm`.
- **norm_type** (*float, optional*) – The p of the p-norm to compute for the `max_norm` option. Default `2`.
- **scale_grad_by_freq** (*boolean, optional*) – If given, this will scale gradients by the inverse of frequency of the words in the mini-batch. Default `False`.
- **sparse** (*bool, optional*) – If `True`, gradient w.r.t. `weight` matrix will be a sparse tensor. See Notes for more details regarding sparse gradients.

3 - Representation

Dense Representation

```
import torch.nn as nn

vocab_size = 7
embedding_dim = 3
embedding = nn.Embedding(vocab_size, embedding_dim)
embedding.weight
```

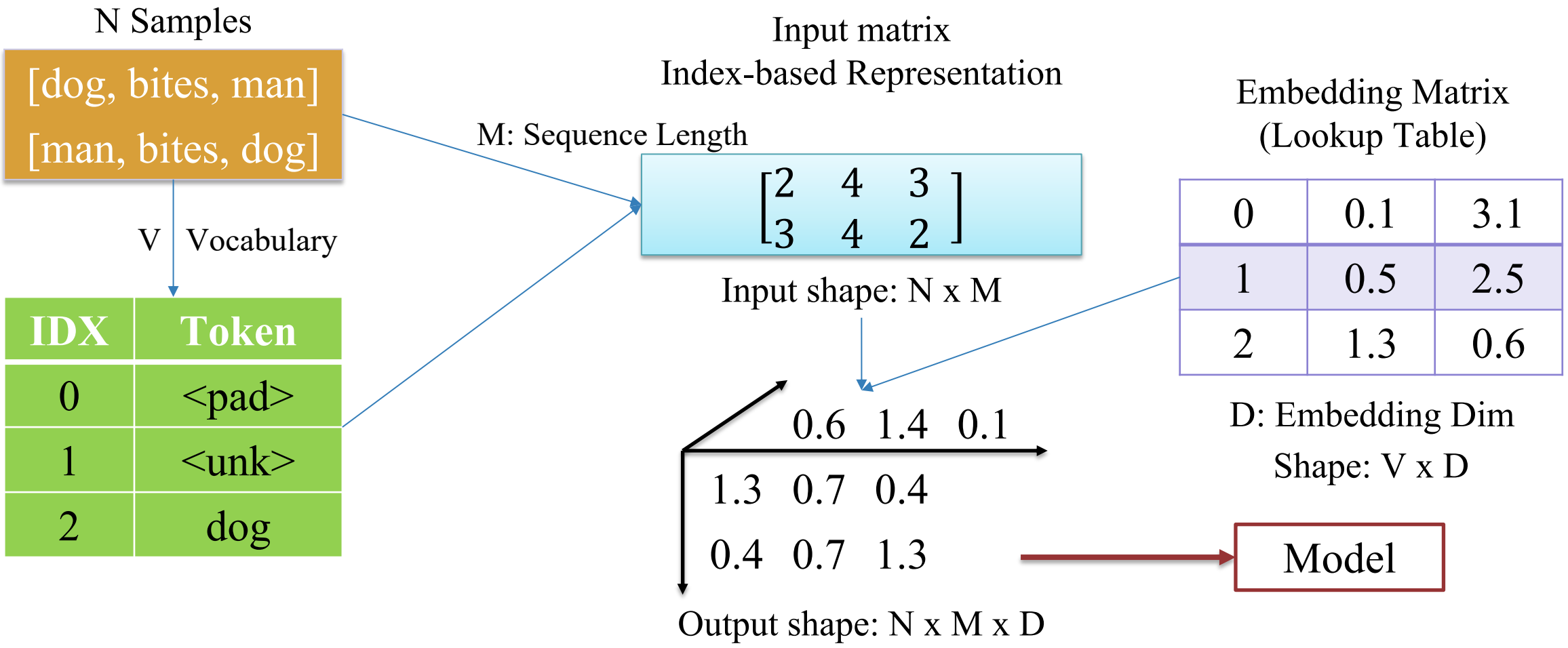
```
Parameter containing:
tensor([[-0.4956, -0.7590, -0.0154],
        [-0.2402,  0.1156, -0.2356],
        [ 0.0707,  0.3050,  0.1886],
        [-0.2861, -0.7316, -0.5887],
        [-1.1057, -0.5170, -0.4673],
        [ 0.7517,  0.7698, -0.6639],
        [ 0.8154, -1.1748,  0.9964]], requires_grad=True)
```

```
import torch

input = torch.LongTensor([[1, 2, 4], [4, 3, 2]])
embedding(input)
```

```
tensor([[[-0.2402,  0.1156, -0.2356],
         [ 0.0707,  0.3050,  0.1886],
         [-1.1057, -0.5170, -0.4673]],

        [[-1.1057, -0.5170, -0.4673],
         [-0.2861, -0.7316, -0.5887],
         [ 0.0707,  0.3050,  0.1886]]], grad_fn=<EmbeddingBackward0>)
```

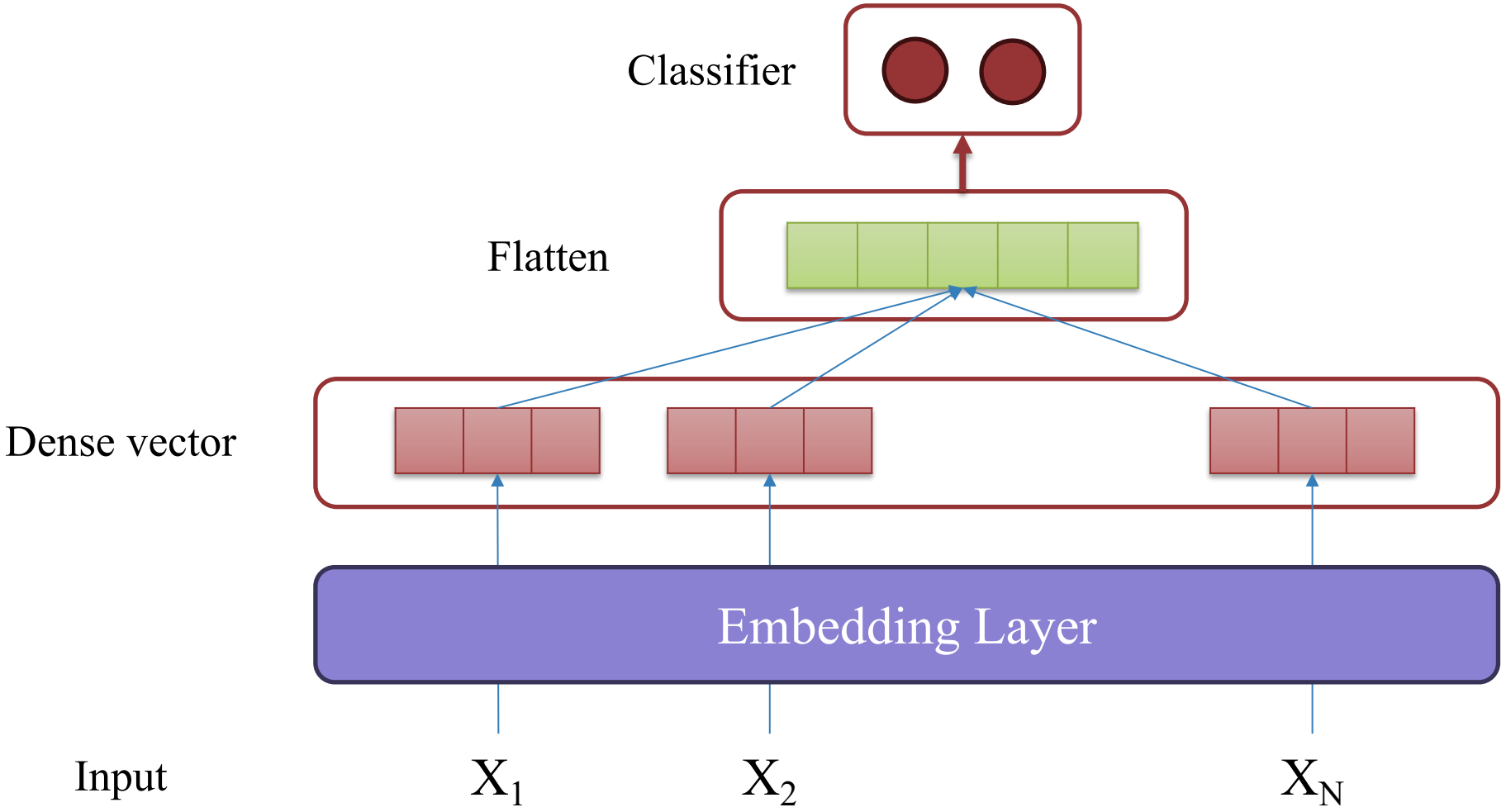
3 - Representation

Dense Representation



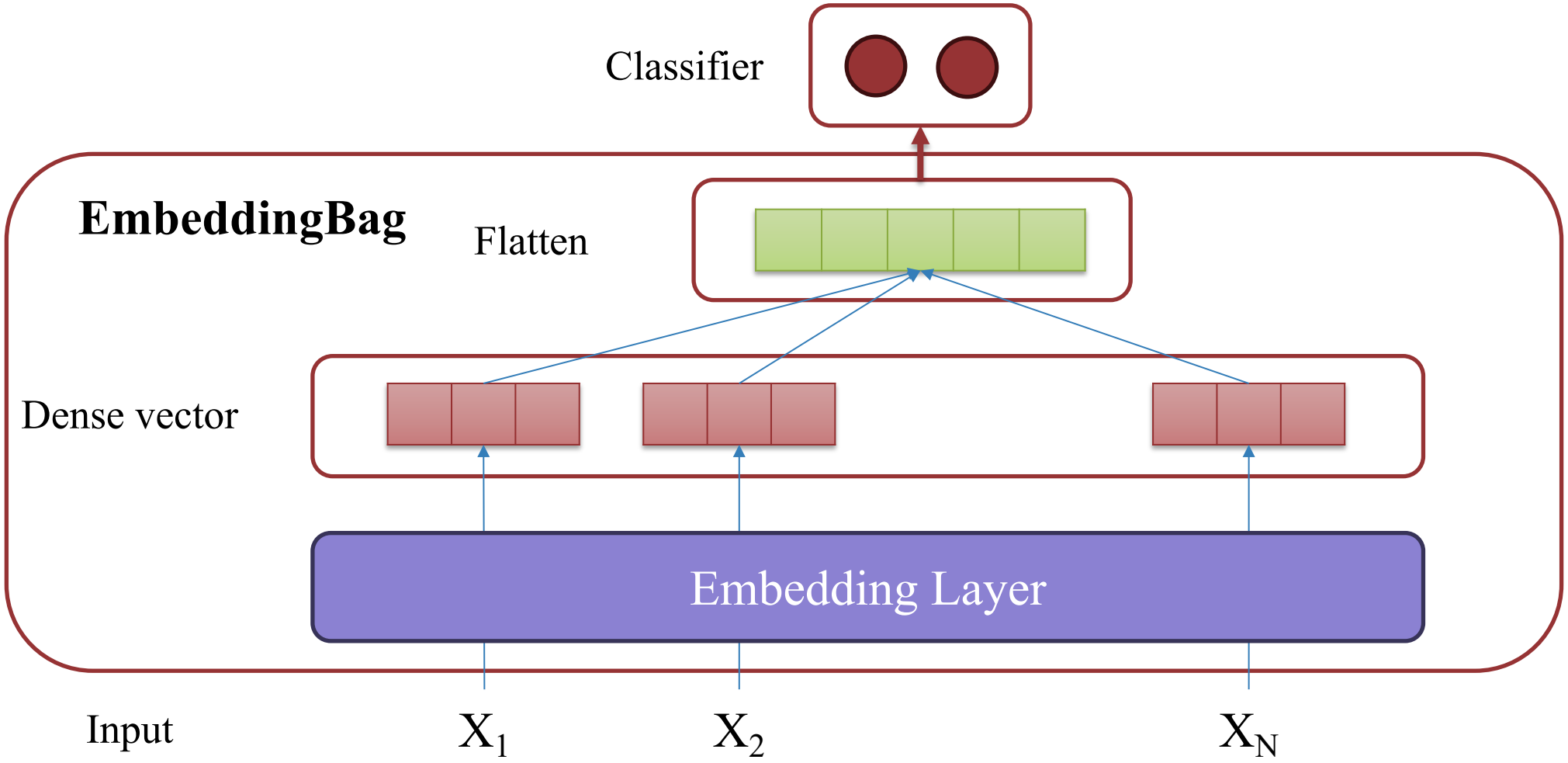
4 - Modeling

Classifier



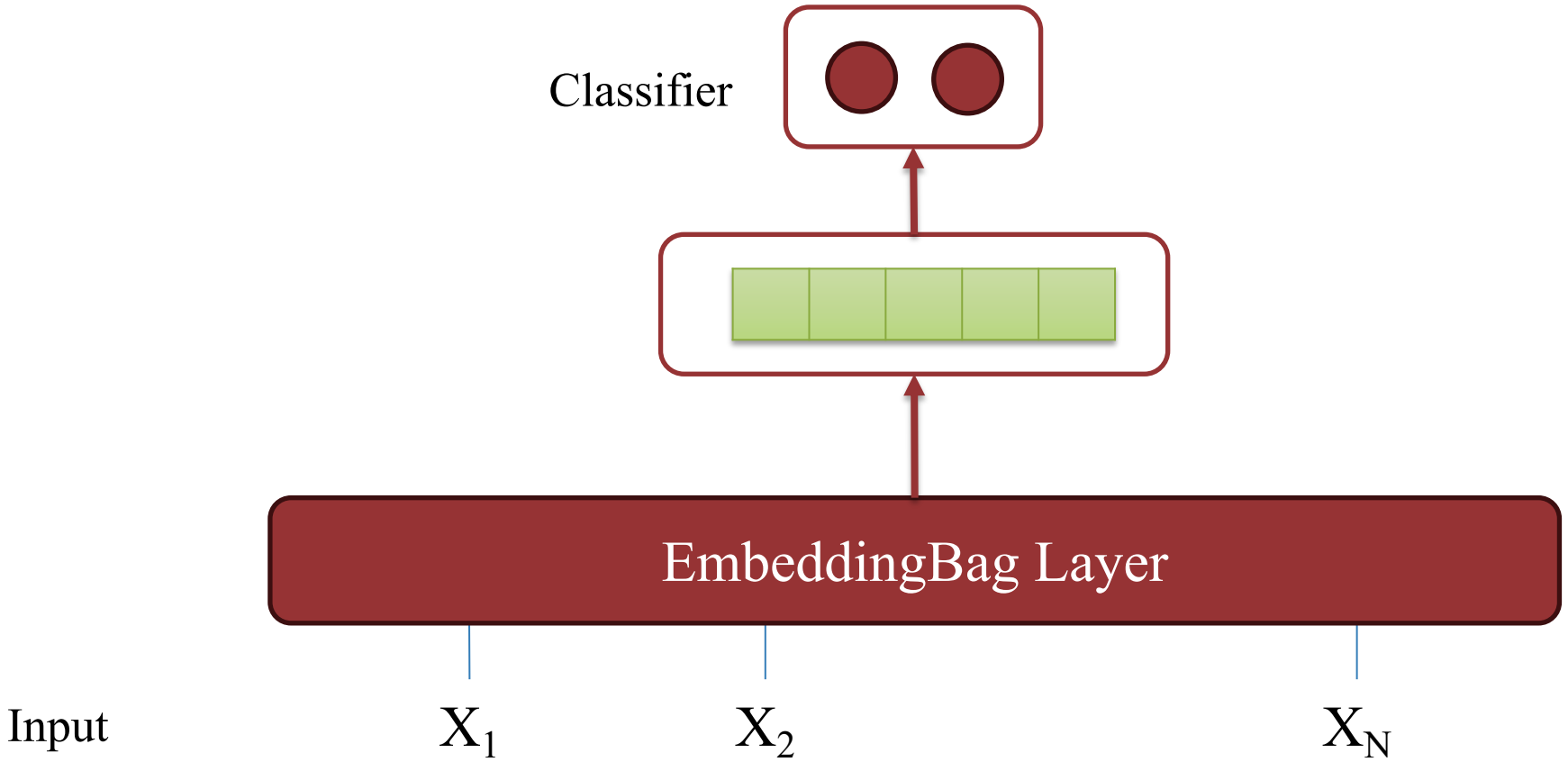
4 - Modeling

Classifier



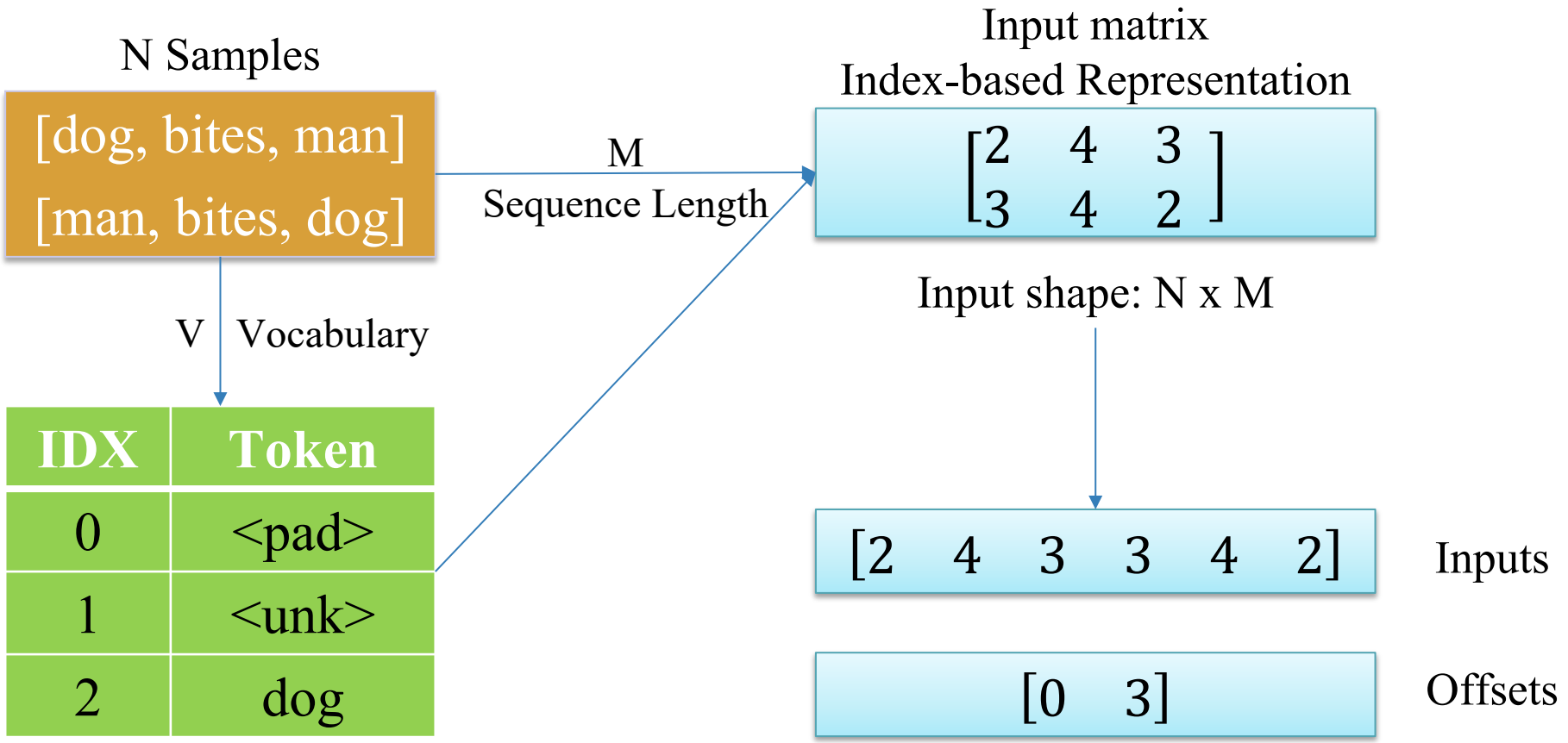
4 - Modeling

Classifier



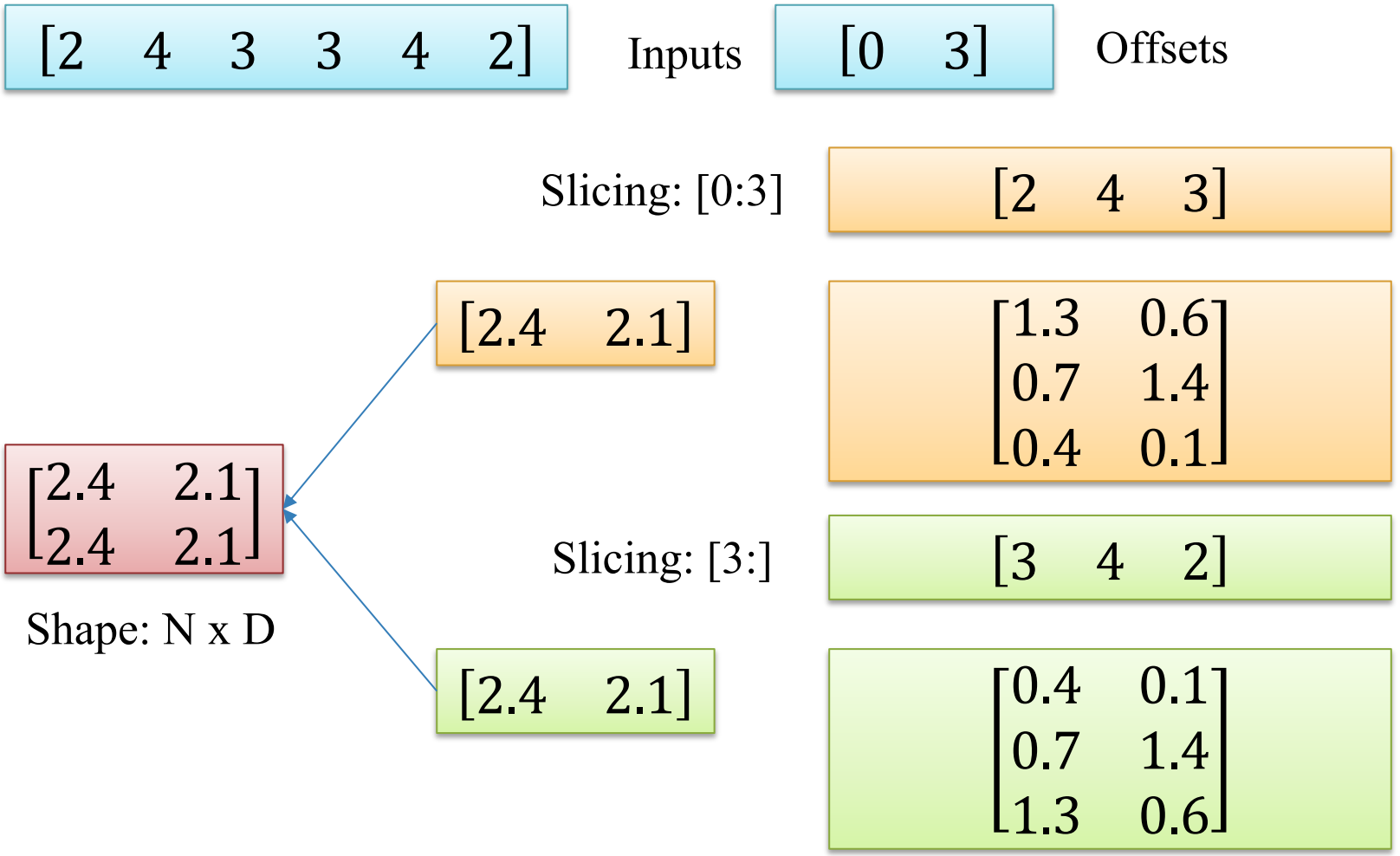
4 - Modeling

EmbeddingBag Layer



4 - Modeling

EmbeddingBag Layer



Embedding Matrix
(Lookup Table)

0	0.1	3.1
1	0.5	2.5
2	1.3	0.6
3	0.4	0.1
4	0.7	1.4
5	2.3	1.7
6	2.5	2.5
7	0.3	1.2

4 - Modeling

EmbeddingBag Layer

EMBEDDINGBAG

CLASS torch.nn.EmbeddingBag(*num_embeddings*, *embedding_dim*, *max_norm=None*, *norm_type=2.0*, *scale_grad_by_freq=False*, *mode='mean'*, *sparse=False*, *_weight=None*, *include_last_offset=False*, *padding_idx=None*, *device=None*, *dtype=None*) [SOURCE]

Computes sums or means of 'bags' of embeddings, without instantiating the intermediate embeddings.

For bags of constant length, no `per_sample_weights`, no indices equal to `padding_idx`, and with 2D inputs, this class

- with `mode="sum"` is equivalent to `Embedding` followed by `torch.sum(dim=1)`,
- with `mode="mean"` is equivalent to `Embedding` followed by `torch.mean(dim=1)`,
- with `mode="max"` is equivalent to `Embedding` followed by `torch.max(dim=1)`.

However, `EmbeddingBag` is much more time and memory efficient than using a chain of these operations.

EmbeddingBag also supports per-sample weights as an argument to the forward pass. This scales the output of the Embedding before performing a weighted reduction as specified by `mode`. If `per_sample_weights` is passed, the only supported `mode` is `"sum"`, which computes a weighted sum according to `per_sample_weights`.

Parameters

- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector
- **max_norm** (*float*, *optional*) – If given, each embedding vector with norm larger than `max_norm` is renormalized to have norm `max_norm`.
- **norm_type** (*float*, *optional*) – The p of the p-norm to compute for the `max_norm` option. Default `2`.
- **scale_grad_by_freq** (*bool*, *optional*) – if given, this will scale gradients by the inverse of frequency of the words in the mini-batch. Default `False`. Note: this option is not supported when `mode="max"`.
- **mode** (*str*, *optional*) – `"sum"`, `"mean"` or `"max"`. Specifies the way to reduce the bag. `"sum"` computes the weighted sum, taking `per_sample_weights` into consideration. `"mean"` computes the average of the values in the bag, `"max"` computes the max value over each bag. Default: `"mean"`
- **sparse** (*bool*, *optional*) – if `True`, gradient w.r.t. `weight` matrix will be a sparse tensor. See Notes for more details regarding sparse gradients. Note: this option is not supported when `mode="max"`.
- **include_last_offset** (*bool*, *optional*) – if `True`, `offsets` has one additional element, where the last element is equivalent to the size of *indices*. This matches the CSR format.
- **padding_idx** (*int*, *optional*) – If specified, the entries at `padding_idx` do not contribute to the gradient; therefore, the embedding vector at `padding_idx` is not updated during training, i.e. it remains as a fixed "pad". For a newly constructed EmbeddingBag, the embedding vector at `padding_idx` will default to all zeros, but can be updated to another value to be used as the padding vector. Note that the embedding vector at `padding_idx` is excluded from the reduction.

Variables

weight (*Tensor*) – the learnable weights of the module of shape (*num_embeddings*, *embedding_dim*) initialized from $\mathcal{N}(0, 1)$.

4 - Modeling

EmbeddingBag Layer

```
import torch.nn as nn

vocab_size = 7
embedding_dim = 4
embedding_sum = nn.EmbeddingBag(vocab_size, embedding_dim, mode='sum')
embedding_sum.weight
```

```
Parameter containing:
tensor([[-0.6818, -0.4208,  1.3662,  0.2827],
        [ 0.3430,  0.1221, -0.1738,  1.6392],
        [ 0.9458, -1.2662, -0.1538, -0.1881],
        [-0.7822, -1.4698, -1.3398,  0.3897],
        [ 0.5941,  0.0387, -0.5284,  0.9480],
        [-1.1119,  0.2135, -1.2069, -0.1746],
        [-0.7334,  0.2189,  1.2279,  0.5824]], requires_grad=True)
```

```
inputs = torch.tensor([1, 2, 4, 5, 4, 3], dtype=torch.long)
offsets = torch.tensor([0, 3], dtype=torch.long)
embedding_sum(inputs, offsets)
```

```
tensor([[ 1.8829, -1.1054, -0.8559,  2.3991],
        [-1.3000, -1.2176, -3.0751,  1.1632]], grad_fn=<EmbeddingBagBackward0>)
```

4 - Modeling

Model

```python
from torch import nn

class TextClassificationModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextClassificationModel, self).__init__()
        self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=False)
        self.fc = nn.Linear(embed_dim, num_class)
        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, inputs, offsets):
        embedded = self.embedding(inputs, offsets)
        return self.fc(embedded)
```

```python
num_class = len(set(train_df_vi['label']))
vocab_size = len(vocabulary)
embed_dim = 256
model = TextClassificationModel(vocab_size, embed_dim, num_class).to(device)
```

```python
model
```

```
TextClassificationModel(
  (embedding): EmbeddingBag(10000, 256, mode='mean')
  (fc): Linear(in_features=256, out_features=2, bias=True)
)
```

5 – Training, Prediction

Source Code

AI VIET NAM

@aivietnam.edu.vn

Thanks!

Any questions?