Ứng dụng Trí tuệ nhân tạo trong Nuôi trồng thủy sản

NGUYỄN HẢI TRIỀU[1]

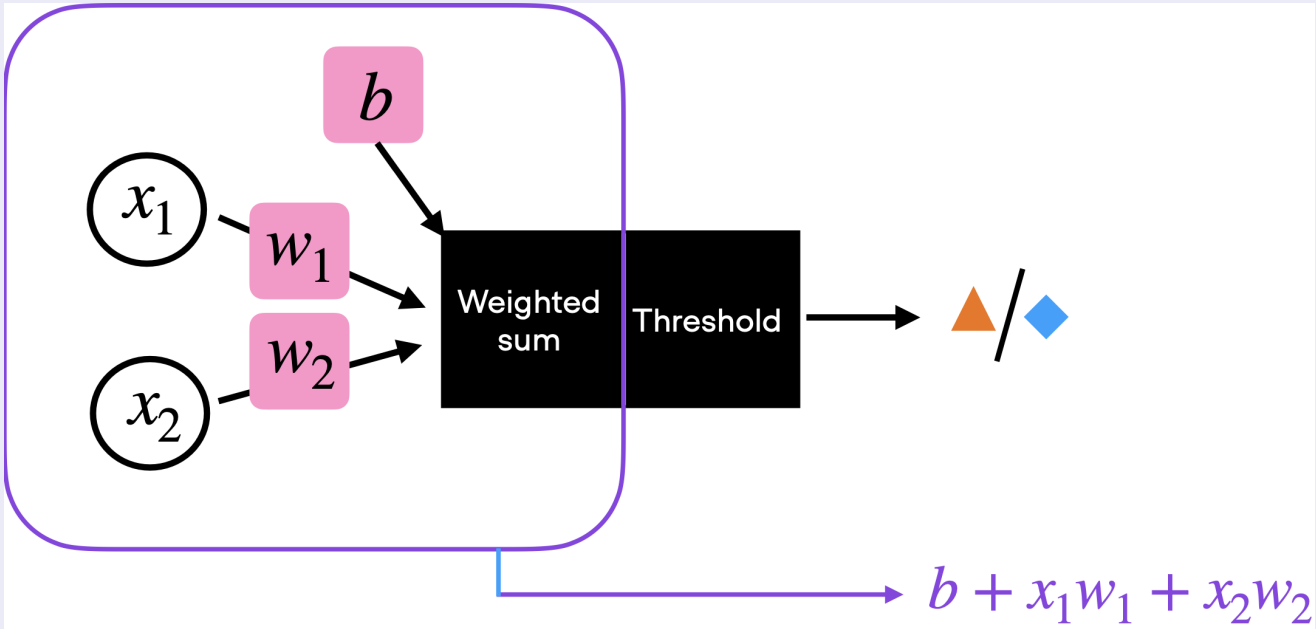
[1] Bộ môn Kỹ thuật phần mềm,

Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, September 2024

From For-Loops to Dot Products

Nhắc lại thuật toán Perceptron: ***1* training example with *2* features values** x_1, x_2; Computation block: $z = b + x_1 w_1 + x_2 w_2$



Đưa về dạng tổng quát cho trường hợp có m đặc trưng:

$$z = b + \sum_{j=1}^{m} x_j w_j.$$

Code Python sử dụng vòng lặp cho ví dụ Perceptron có 1 training example và 2 đặc trưng.

```
b = 0.0
x = [1.2, 2.2]
w = [3.3, 4.3]
output = b
for xj, wj in zip(x,w):
  output += xj*wj
print(output)
# -> 13.42
```

Dot Product Between 2 Vectors

Bên cạnh việc sử dụng vòng lặp cho bài toán trên, chúng ta có thể sử dụng tính chất của đại số tuyến tính: phép tính Dot Product của 2 vector $\mathbf{x}$, $\mathbf{w}$

$$z = b + \mathbf{x}^T \mathbf{w},$$

Trong đó,

$$\mathbf{x}^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \ \mathbf{w} = [w_1 \ w_2 \ \dots \ w_m]$$

Dot Product Between 2 Vectors

Chúng ta có thể dùng *.dot()* trong PyTorch như sau:

```
1  import torch
2  b = torch.tensor([0.])
3  x = torch.tensor([1.2, 2.2])
4  w = torch.tensor([3.3, 4.3])
5  output = b + x.dot(w)
6  print(output)
7  # -> tensor([13.4200])
```

Rõ ràng sử dụng Dot Product theo như đại số tuyến tính sẽ làm code đơn giản, dễ hiểu hơn viết vòng lặp

Sử dụng toán tử .dot()

```
1 import torch
2 b = torch.tensor([0.])
3 x = torch.tensor([1.2, 2.2])
4 w = torch.tensor([3.3, 4.3])
5 output = b + x.dot(w)
6 print(output)
```

✓ 0.0s Python

```
tensor([13.4200])
```

for-loop for 1 training example 2 feature

```
1 b = 0.0
2 x = [1.2, 2.2]
3 w = [3.3, 4.3]
4 output = b
5 for xj, wj in zip(x,w):
6     output += xj*wj
7 print(output)
```

[1] ✓ 0.0s Pyth

```
13.42
```

Câu hỏi đặt ra: sử dụng Dot Product có giúp chương trình hiệu quả, chạy nhanh hơn không? *Hãy tiến hành Benchmark để xem kết quả.*

Benchmark

Chúng ta sẽ tiến hành so sánh thời gian thực thi code Perceptron với 1 training example và 10.000 features bằng vòng lặp và Dot Product

Benchmarking the Plain Python Approach

```python
import random
random.seed(123) # đảm bảo kết quả ngẫu nhiên trùng giữa các lần chạy trên máy local
def plain_python(x, w, b):
    output = b
    for xj, wj in zip(x,w):
        output += xj*wj
    return output

n_features = 10000
b = 0.0
x = [random.random() for _ in range(n_features)]
w = [random.random() for _ in range(n_features)]
%timeit plain_python(x, w, b)
```

✓ 2.2s Python

```
279 µs ± 10.2 µs per loop (mean ± std. dev. of 7 runs, 1,0
```

Benchmarking the Dot Product

```python
import torch
import random
def pytorch_dot(x, w, b):
    return b + x.dot(w)
random.seed(123)
b = torch.tensor([0.0])
x = torch.tensor([random.random() for _ in range(n_features)])
w = torch.tensor([random.random() for _ in range(n_features)])
%timeit pytorch_dot(x, w, b)
```

[6] ✓ 2.5s Python

```
3.08 µs ± 23.8 ns per loop (mean ± std. dev. of 7 runs, 10
```

Dựa vào kết quả benchmark: sử dụng Dot Product giúp chương trình hiệu quả, chạy **nhanh hơn vòng lặp khoảng *90* lần** cho 1 training example có 10.000 đặc trưng.

Dealing with Multiple Training Examples via Matrix Multiplication

Trong trường hợp có n training example, chúng ta sử dụng chỉ số i đại diện cho training example thứ i và sử dụng *cùng 1 vector trọng số cho tất cả các training example*:

$$z^i = b + \sum_{j=1}^{m} x_j^i w_j.$$

$$z^1 = b + \sum_{j=1}^{m} x_j^1 w_j$$

$$z^2 = b + \sum_{j=1}^{m} x_j^2 w_j$$

$$\cdots\cdots\cdots\cdots\cdots\cdots$$

$$z^n = b + \sum_{j=1}^{m} x_j^n w_j$$

Viết lại dưới dạng Dot product

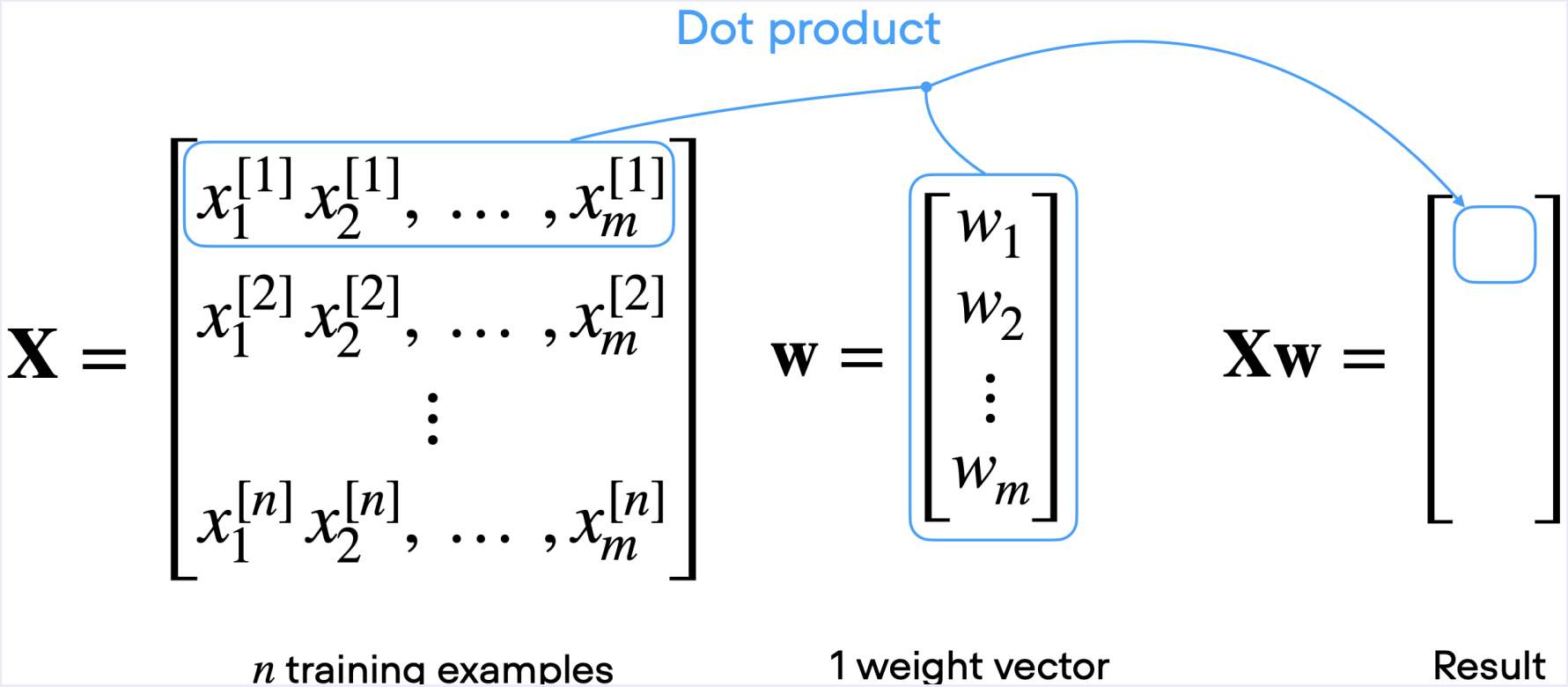
$$z^1 = b + \mathbf{x}^1\mathbf{w}$$
$$z^2 = b + \mathbf{x}^2\mathbf{w}$$
$$\ldots\ldots\ldots\ldots$$
$$z^n = b + \mathbf{x}^n\mathbf{w}$$

Tổng quát: $\mathbf{z} = b + \mathbf{X}\mathbf{w}$, trong đó,

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \ldots \\ \mathbf{x}^n \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \ldots & x_m^1 \\ x_1^2 & x_2^2 & \ldots & x_m^2 \\ \ldots & \ldots & \ldots & \ldots \\ x_1^n & x_2^n & \ldots & x_m^n \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \ldots \\ w_m \end{bmatrix}.$$

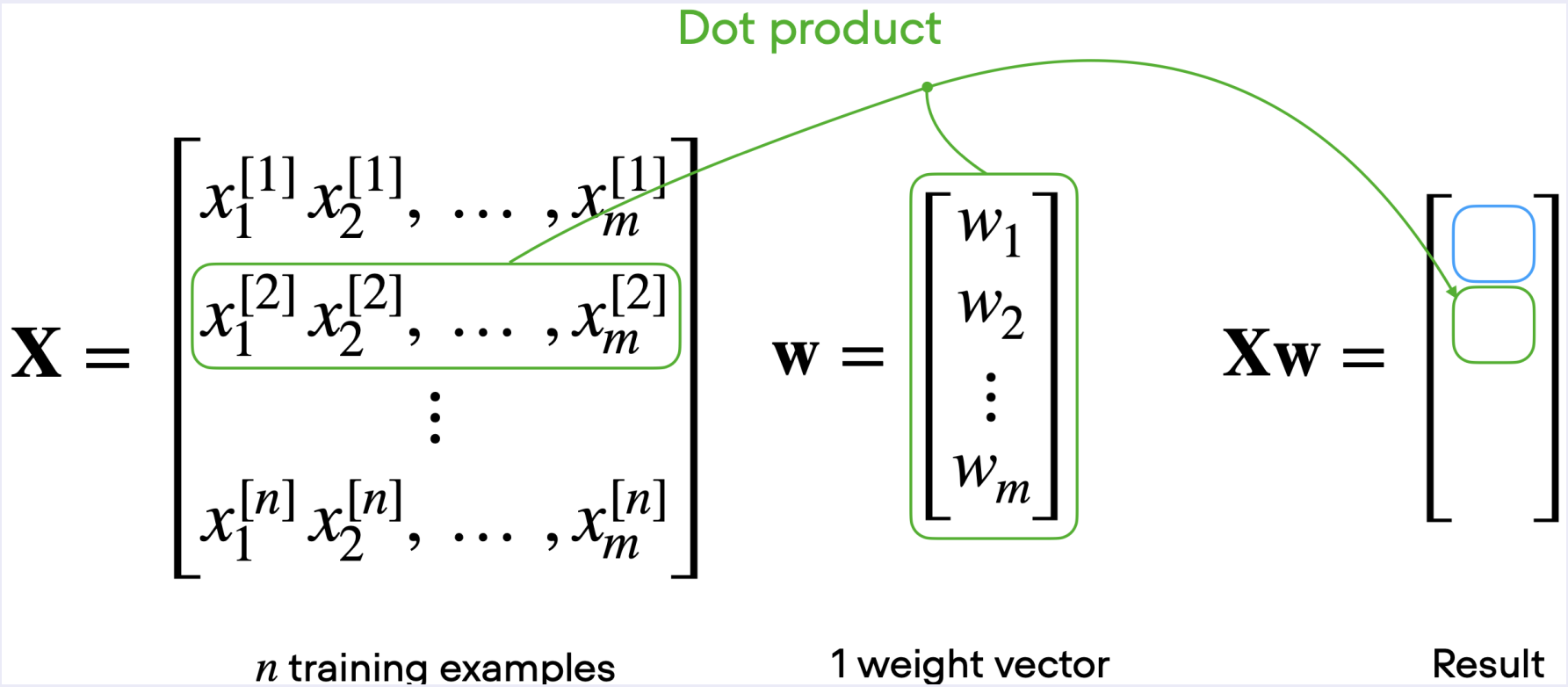
Matrix Multiplication

Từ công thức tổng quát, chúng ta thực hiện nhân ma trận



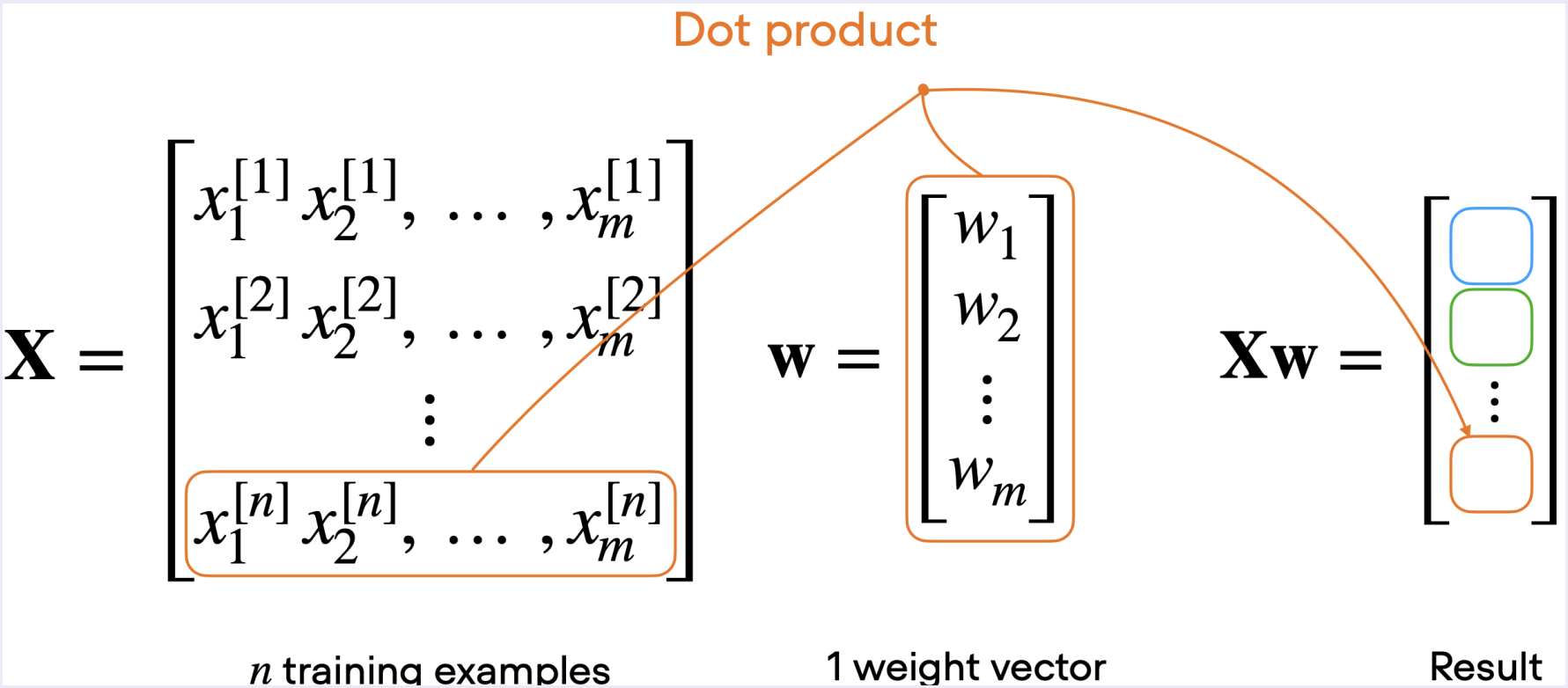
Matrix Multiplication

Từ công thức tổng quát, chúng ta thực hiện nhân ma trận



Matrix Multiplication

Từ công thức tổng quát, chúng ta thực hiện nhân ma trận



So sánh giữa code Python thuần tuý và sử dụng nhân ma trận của PyTorch:

Multiplying a matrix and a vector in plain Python

```python
b = 0.0
X = [[1.2, 2.2],
     [4.4, 5.5]]
w = [3.3, 4.3]
output = []
for x in X:
    sum = b
    for xj, wj in zip(x,w):
        sum += xj*
    output.append(sum)
print(output)
```

(variable) sum: float

0.0s Python

```
[13.42, 38.17]
```

Multiplying a matrix and a vector in PyTorch

```python
import torch
b = torch.tensor([0.])
X = torch.tensor([[1.2, 2.2],
                  [4.4, 5.5]])
w = torch.tensor([3.3, 4.3])
output = b + torch.matmul(X,w)
print(output)
```

[2] 1.5s Python

```
tensor([13.4200, 38.1700])
```

Benchmarking the Plain Python implementation

```python
import random
random.seed(123)

def plain_python(X, w, b):
    output = []
    for x in X:
        sum = b
        for xj, wj in zip(x,w):
            sum += xj*wj
        output.append(sum)
    return output

n_features = 10000
n_examples = 10000
X = [[random.random() for _ in range(n_features)]
for i in range(n_examples)]
b = 0.0
w = [random.random() for _ in range(n_features)]
%timeit plain_python(X, w, b)
```

✓ 41.2s Python

```
4.35 s ± 64.1 ms per loop (mean ± std. dev. of 7 runs, 1 l
```

Benchmarking the PyTorch implementation

```python
import torch
import random

def pytorch_implementation(X, w, b):
    return b + torch.matmul(X,w)

random.seed(123)
n_features = 10000
n_examples = 10000
b = torch.tensor([0.0])
X = torch.tensor([[random.random() for _ in
range(n_features)] for i in range(n_examples)])
w = torch.tensor([random.random() for _ in
range(n_features)])
%timeit pytorch_implementation(X, w, b)
```

[2] ✓ 25.1s Python

```
12.2 ms ± 18.4 µs per loop (mean ± std. dev. of 7 runs, 1
```

Dựa vào thực nghiệm ta thấy, sử dụng nhân ma trận của PyTorch nhanh hơn code Python thuần tuý khoảng 357 lần.

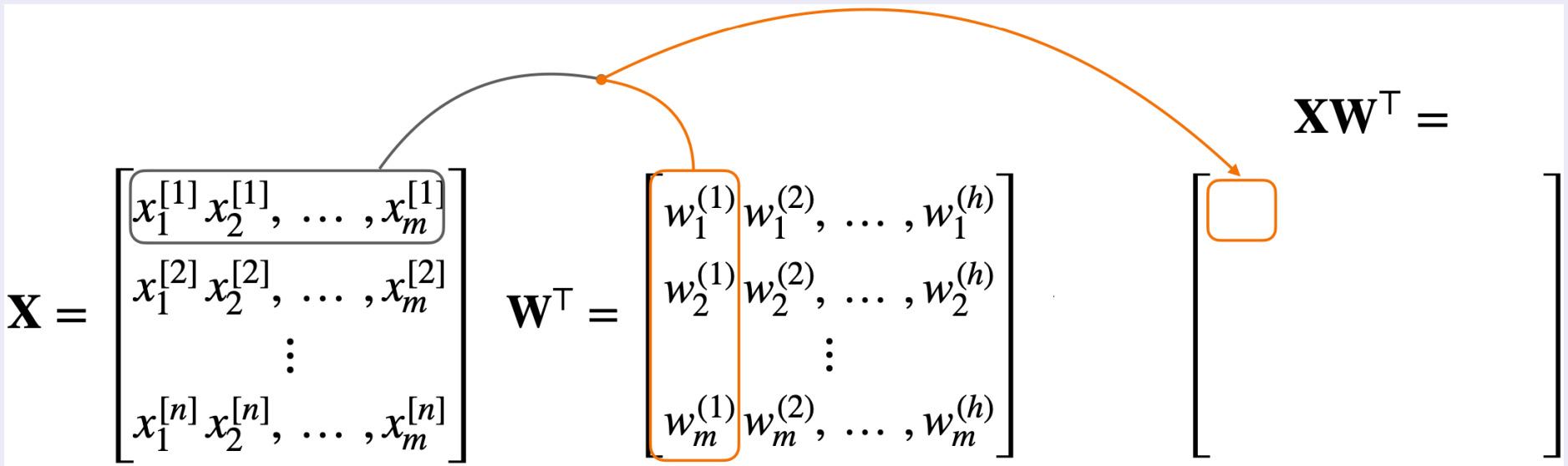
Trong trường hợp trọng số là một ma trận có h vector (feature weights), mỗi vector tương ứng với một training example, ta có biểu diễn dưới dạng ma trận như sau

$$\mathbf{X} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_m^1 \\ x_1^2 & x_2^2 & \dots & x_m^2 \\ \dots & \dots & \dots & \dots \\ x_1^n & x_2^n & \dots & x_m^n \end{bmatrix}, \mathbf{W} = \begin{bmatrix} w_1^1 & w_2^1 & \dots & w_m^1 \\ w_1^2 & w_2^2 & \dots & w_m^2 \\ \dots & \dots & \dots & \dots \\ w_1^h & w_2^h & \dots & w_m^h \end{bmatrix},$$

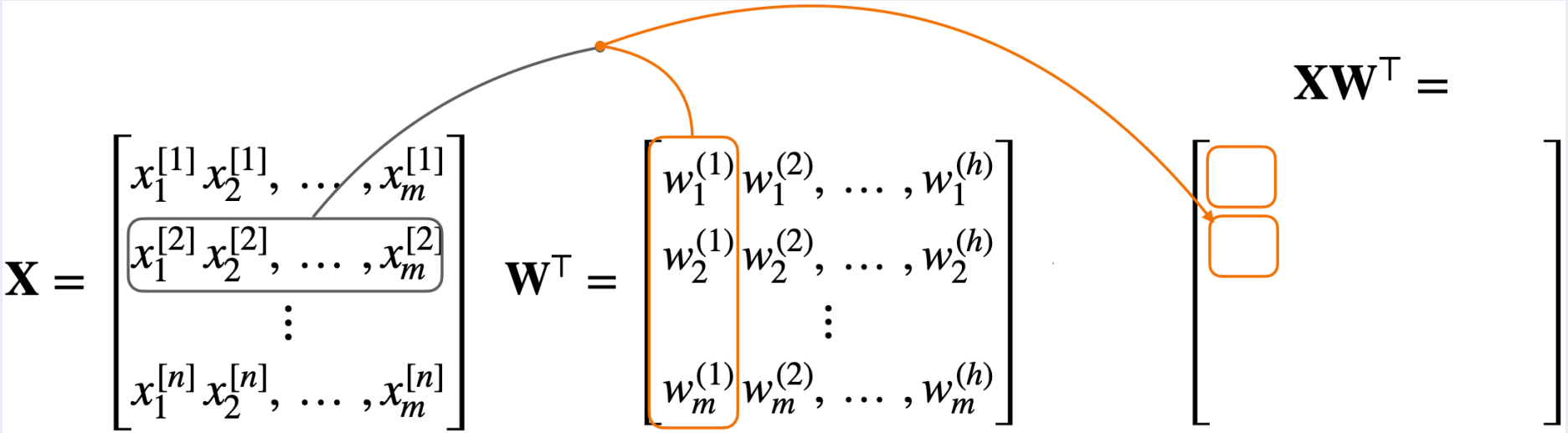
Requirement: Number of columns of $\boldsymbol{X}$ *and rows of* $\boldsymbol{W}$ *have to match.* Vì vậy, chúng ta cần phải chuyển vị ma trận trọng số để khớp số chiều.

Matrix Multiplication with Weight Matrix

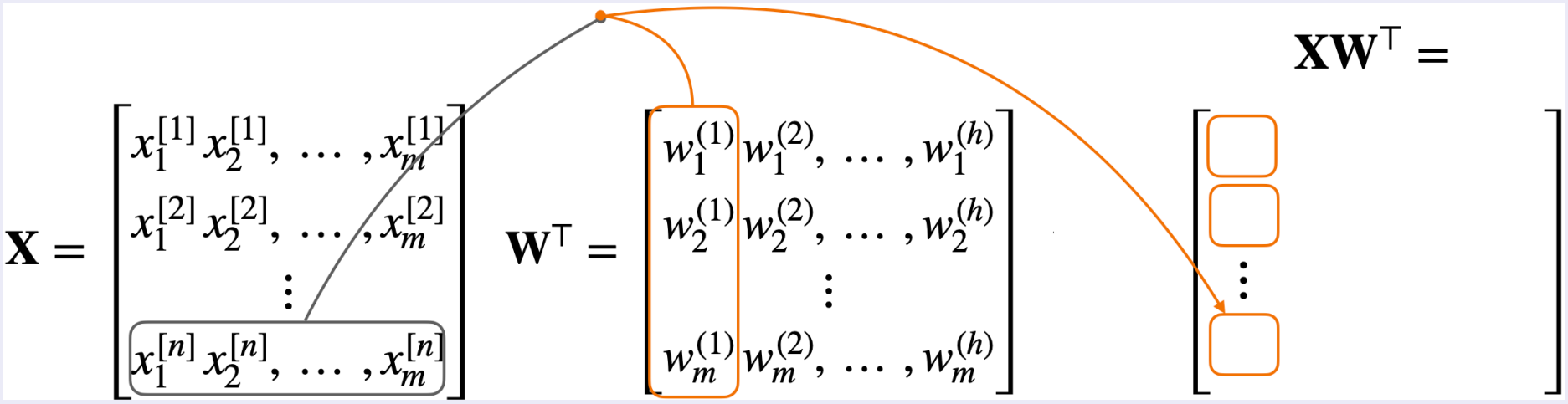
$$\mathbf{X} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_m^1 \\ x_1^2 & x_2^2 & \dots & x_m^2 \\ \dots & \dots & \dots & \dots \\ x_1^n & x_2^n & \dots & x_m^n \end{bmatrix}, \mathbf{W}^T = \begin{bmatrix} w_1^1 & w_1^2 & \dots & w_1^h \\ w_2^1 & w_2^2 & \dots & w_2^h \\ \dots & \dots & \dots & \dots \\ w_m^1 & w_m^2 & \dots & w_m^h \end{bmatrix},$$



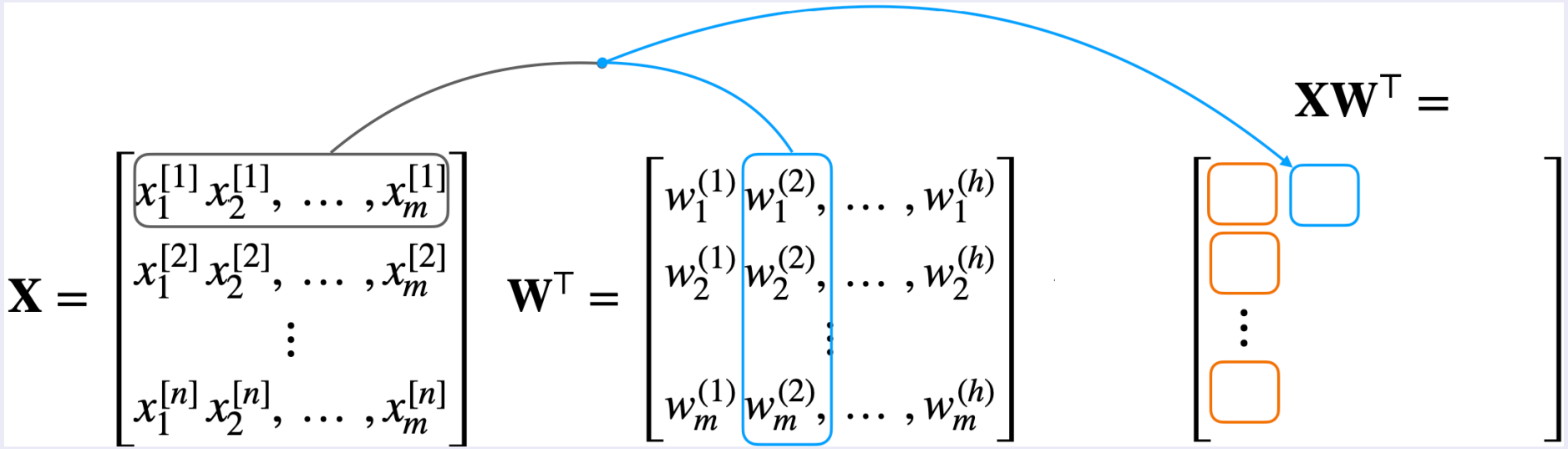
Matrix Multiplication with Weight Matrix



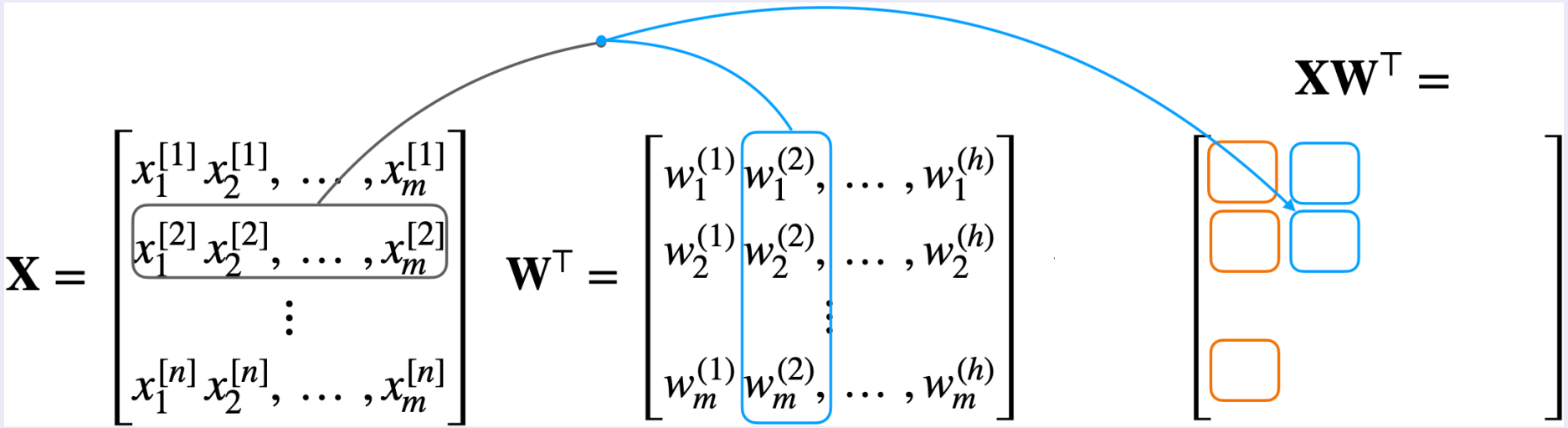
Matrix Multiplication with Weight Matrix



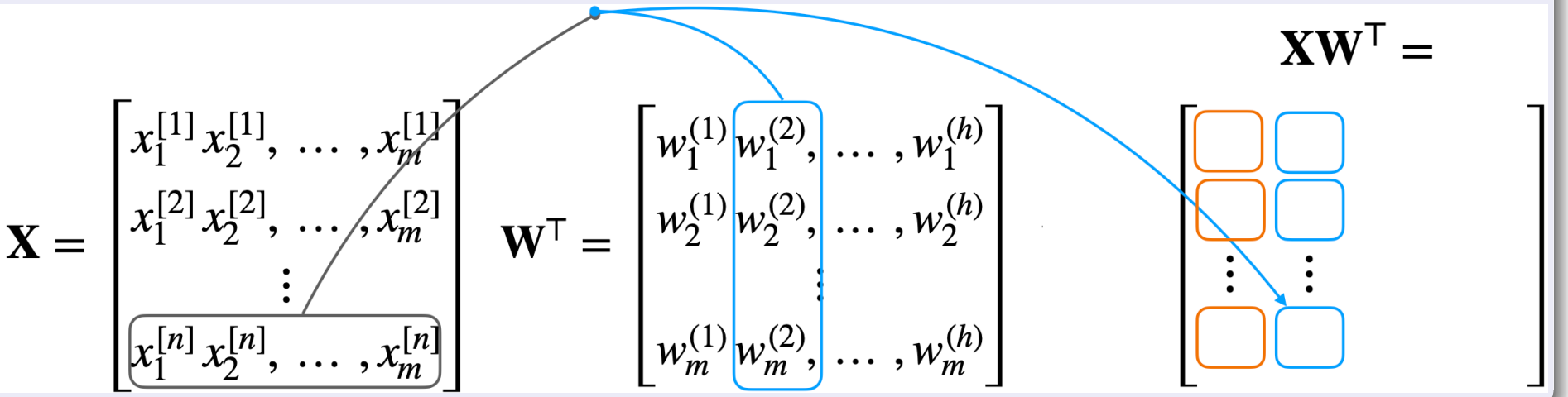
Matrix Multiplication with Weight Matrix



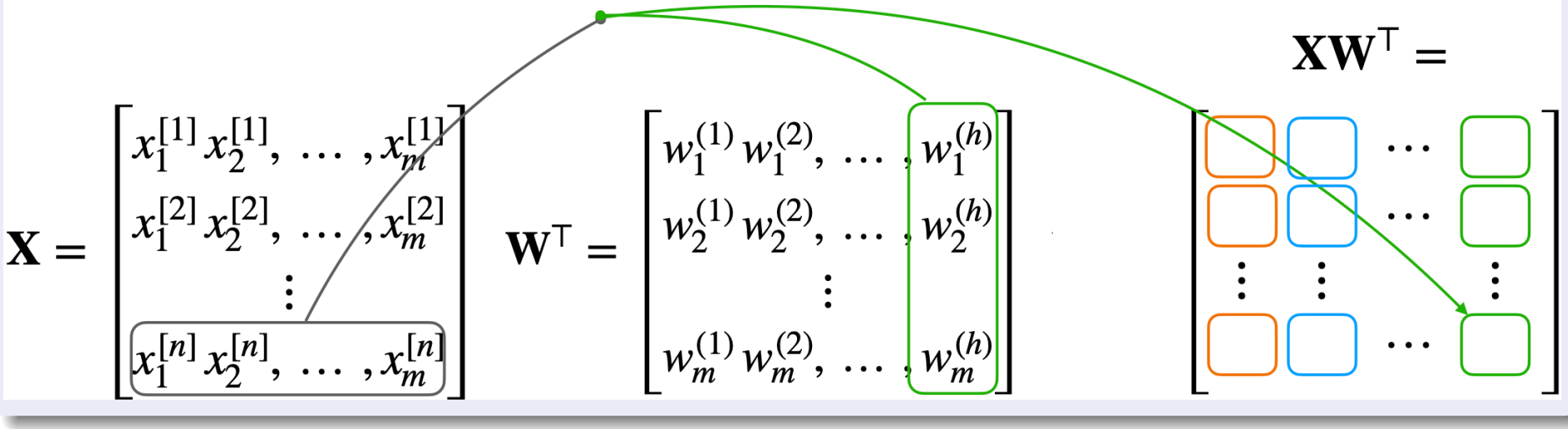
Matrix Multiplication with Weight Matrix



Matrix Multiplication with Weight Matrix



Matrix Multiplication with Weight Matrix

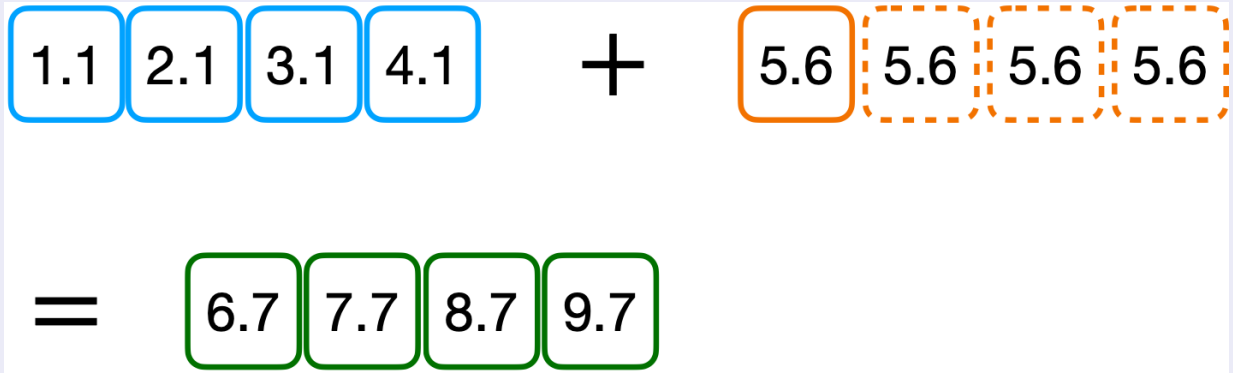


Computations with Unequal Tensor Shapes

Trong PyTorch, ta có thể sử dụng Broadcasting trên các Tensor không cùng kích thước. Ví dụ một vector cộng với một số vô hướng (trong toán học không thể thực hiện được)

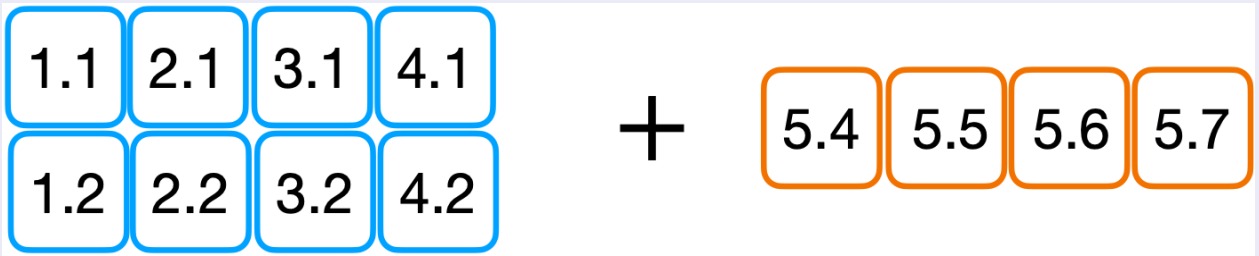
Nhưng PyTorch cho phép thực hiện bằng cách *ngầm bổ sung cho vô hướng thành một vector có cùng kích thước*

Kết quả thu được

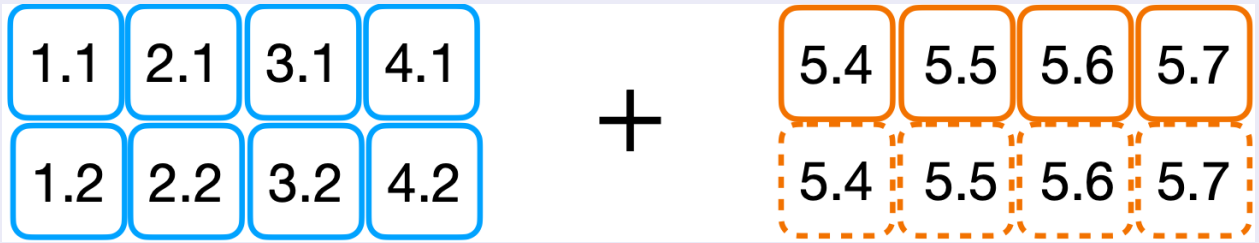


Computations with Unequal Tensor Shapes

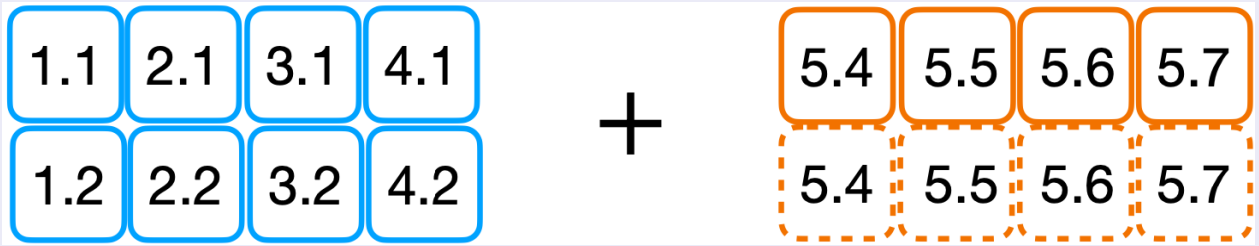
Ví dụ:



PyTorch ngầm bổ sung chiều:



Kết quả thu được



Computations with Unequal Tensor Shapes

Ví dụ code cho Broadcasting

```
1  a = torch.tensor([1.,2.,3.,4.])
2  b = torch.tensor(1.)
3  a+b
4  # -> tensor([2., 3., 4., 5.])
```

Tài liệu tham khảo


Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili
Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python (2022). Published by Packt Publishing Ltd, ISBN 978-1-80181-931-2.


Sebastian Raschka
MACHINE LEARNING Q AND AI: 30 Essential Questions and Answers on Machine Learning and AI (2024). ISBN-13: 978-1-7185-0377-9 (ebook).


LightningAI
LightningAI: PyTorch Lightning (2024) .