Môn Cấu trúc dữ liệu và giải thuật nâng cao

# Computational Graph

## 1. NHẮC LẠI MỤC TIÊU VÀ YÊU CẦU

### *1.1. Mục tiêu*

- Nắm được khái niệm
- Nắm được bài toán thích hợp
- Hiểu cách tổ chức dữ liệu
- Hiểu cách xây dựng chương trình

### *1.2. Nội dung*

- Khái niệm
- Sự cần thiết
- Cách tổ chức và vận hành cấu trúc dữ liệu
- Ứng dụng (Các ứng dụng trong chương trình học và thực tế)
- Ví dụ minh họa
- Bài tập làm tại lớp
- Bài tập về nhà
- Chương trình demo một ứng dụng thực tế ; cần thông báo các thông số liên quan khi chương trình thực thi xong.

## 2. NỘI DUNG CHI TIẾT

### 2.1. Khái niệm

#### *2.1.1. Khái quát và phân loại*

Computational Graph là đồ thị có hướng, trong đó các nút tương ứng với các phép toán hoặc biến số.

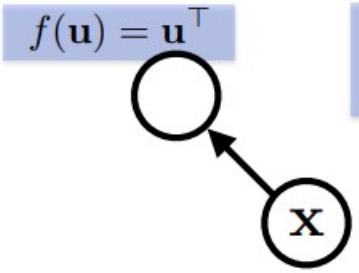
> Biến số có thể cung cấp các giá trị của chúng cho các phép toán và các phép toán có thể cung cấp đầu ra của chúng cho các phép toán khác. Như vậy, mỗi nút trong đồ thị xác định một hàm của các biến số.

Có hai loại computational graph chính là đồ thị tính toán tĩnh (Static Computational Graph) và đồ thị tính toán động (Dynamic Computational Graph).

### *2.1.2. Nút, cạnh và đồ thị*

Xét biểu thức : x
Đồ thị : (x)
Nhận xét :
Mỗi **nút** là một giá trị tensor, ma trận, vector hoặc vô hướng (scalar).

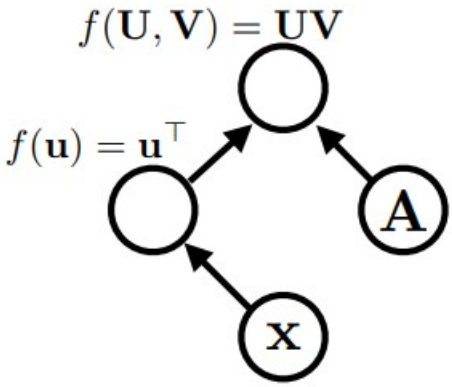
Xét biểu thức : x^T
Đồ thị :




$$\frac{\partial f(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathcal{F}}{\partial f(\mathbf{u})} = \left( \frac{\partial \mathcal{F}}{\partial f(\mathbf{u})} \right)^\top$$

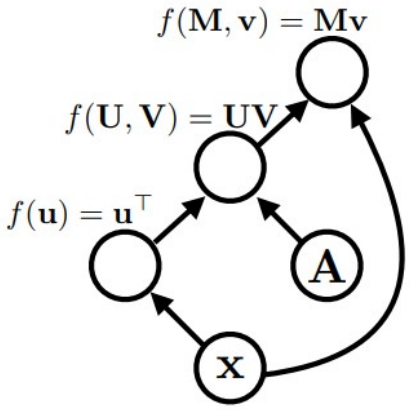
Nhận xét :
Mỗi cạnh đại diện cho đối số của hàm, trỏ vào nút.
Nếu nút có cạnh dẫn vào thì nút đó là hàm số của nút mà cạnh đó đi ra từ.
Mỗi nút biết cách tính giá trị của nó và đạo hàm ứng với từng đối số (cạnh) nhân với đạo hàm của một input bất kỳ dF/df(u)

Xét biểu thức : x^T * A
Đồ thị :



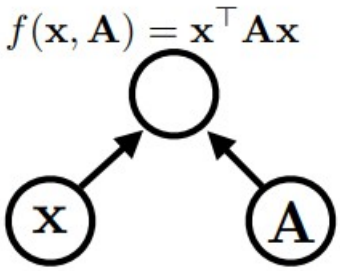

Nhận xét :
Hàm số có thể có 0, 1, 2… đối số. Thường thì chúng có một hay hai đối số.

Xét biểu thức : x^T A x
Đồ thị :




Đồ thị tính toán thì có hướng và không tuần hoàn (thường là vậy).

Đồ thị khác cho kết quả tương tự :

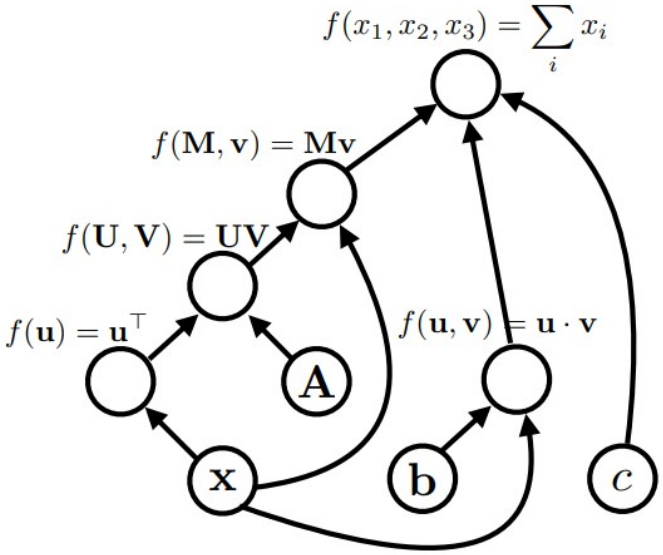



$$\frac{\partial f(\mathbf{x}, \mathbf{A})}{\partial \mathbf{x}} = (\mathbf{A}^\top + \mathbf{A})\mathbf{x}$$
$$\frac{\partial f(\mathbf{x}, \mathbf{A})}{\partial \mathbf{A}} = \mathbf{x}\mathbf{x}^\top$$

Biểu thức :

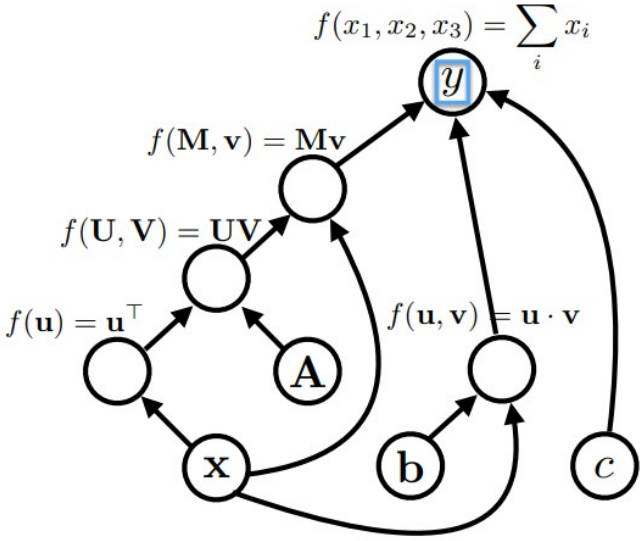
$$\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$$

Đồ thị :




Có thể biểu diễn lại như sau :

$$y = \mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$$




Ta thấy tên biến (y) chỉ là để dán nhãn cho các nút.

## 2.2. Sự cần thiết

Computational Graph có nhiều lợi ích khi sử dụng, bao gồm :

1. Lập lịch dựa trên sự phụ thuộc. Cụ thể là các phụ thuộc dữ liệu, xác định thứ tự thực hiện. Các phép toán không phụ thuộc vào nhau có thể được lập trình song song.
2. Tối ưu hóa đồ thị. Chẳng hạn như loại bỏ đồ thị con chung.
3. Vi phân tự động. Mô tả tính toán dưới dạng đồ thị cho phép chúng ta dễ dàng tính được đạo hàm.

## 2.3. Cách tổ chức và vận hành cấu trúc dữ liệu

Để làm việc với đồ thị tính toán, ta cần làm ba công việc :

1. Khởi tạo đồ thị.
2. Lan truyền xuôi (*Forward Propagation*).
    - Chạy qua các nút theo thứ tự tôpô. Trên mỗi nút, tính giá trị của chúng theo đầu vào đã cho.
    - Đưa ra dự đoán, hoặc tính lỗi theo đầu ra mục tiêu.
3. Lan truyền ngược (*Backward Propagation* hay *Backpropagation*).
    - Chạy qua các nút theo thứ tự tôpô ngược, bắt đầu từ nút đích cuối cùng. Trên mỗi nút, tính đạo hàm của nút đích cuối cùng ứng với nút đuôi của từng cạnh.
    - Output sẽ thay đổi ra sao nếu input có thay đổi nhỏ ?

### *2.3.1. Forward Propagation.*

Xem minh họa trên slide.

### *2.3.2. Backpropagation.*

### *2.3.3. Khởi tạo đồ thị.*

Có hai cách xây dựng đồ thị là

*Đồ thị tính toán tĩnh. (Static Computational Graph.)*

1. Xác định cấu trúc. Có thể bao gồm các điều khiển luồng thô sơ như vòng lặp và điều kiện rẽ nhánh.

2. Cho dữ liệu chạy qua để huấn luyện model và đưa ra dự đoán.

*Đồ thị tính toán động. (Dynamic Computational Graph.)* Đồ thị được định nghĩa *ngầm* (chẳng hạn bằng cách overload toán tử) khi thực hiện tính toán xuôi chiều.

#### *2.3.4. Cài đặt bằng PyTorch.*

Xem file Jupyter Notebook kèm theo (file example).

### 2.4. Ứng dụng

Tất cả các thuật toán sử dụng đạo hàm thì đồ thị tính toán đều có thể được áp dụng.

Trong Linear Regression, từ việc tìm đường thẳng để tìm w_0, w_1 để J nhỏ nhất, ta cần một thuật toán để tìm giá trị nhỏ nhất của J. Đó là Gradient Descent.

Backpropagation là thuật toán chính đào tạo các mô hình sâu có thể xử lý được bằng máy tính. Đối với các mạng neural hiện đại, nó có thể giúp quá trình huấn luyện giảm dần độ dốc nhanh hơn gấp 10 triệu lần so với cách triển khai ngây thơ.

### 2.5. Ví dụ minh họa

Xem file Jupyter Notebook kèm theo (file example).

### 2.6. Quiz

Có 10 câu hỏi từ dễ đến trung bình.

### 2.7. Bài tập về nhà

Bài tập về nhà được đem lên slide. Hạn chót nộp bài là trước buổi học sau. Bài tập sẽ được sửa trước lớp trong 15 phút.

### 2.8. Chương trình demo

Chương trình demo được viết bằng Jupyter Notebook (Python 3.10.11) và áp dụng PyTorch để cài đặt Computational Graph.

Xem file Jupyter Notebook kèm theo (file demo).

## 3. Tham khảo

1. 04-nn-compgraph.key (cornell.edu)

## 4. Xem thêm

1. Automatic differentiation package - torch.autograd — PyTorch 2.0 documentation
2. https://www.tutorialspoint.com/python_deep_learning/python_deep_learning_computational_graphs.htm
3. …