|  |
| --- |
| Nhóm 6: *(Lớp chiều thứ 4, tiết 9-12)*  - Bùi Thanh Tính 1513505  - Đỗ Văn Quốc 1512705  - Nguyễn Khắc Trung Tín 1513489  - Phạm Ngọc Tuấn 1533720 |

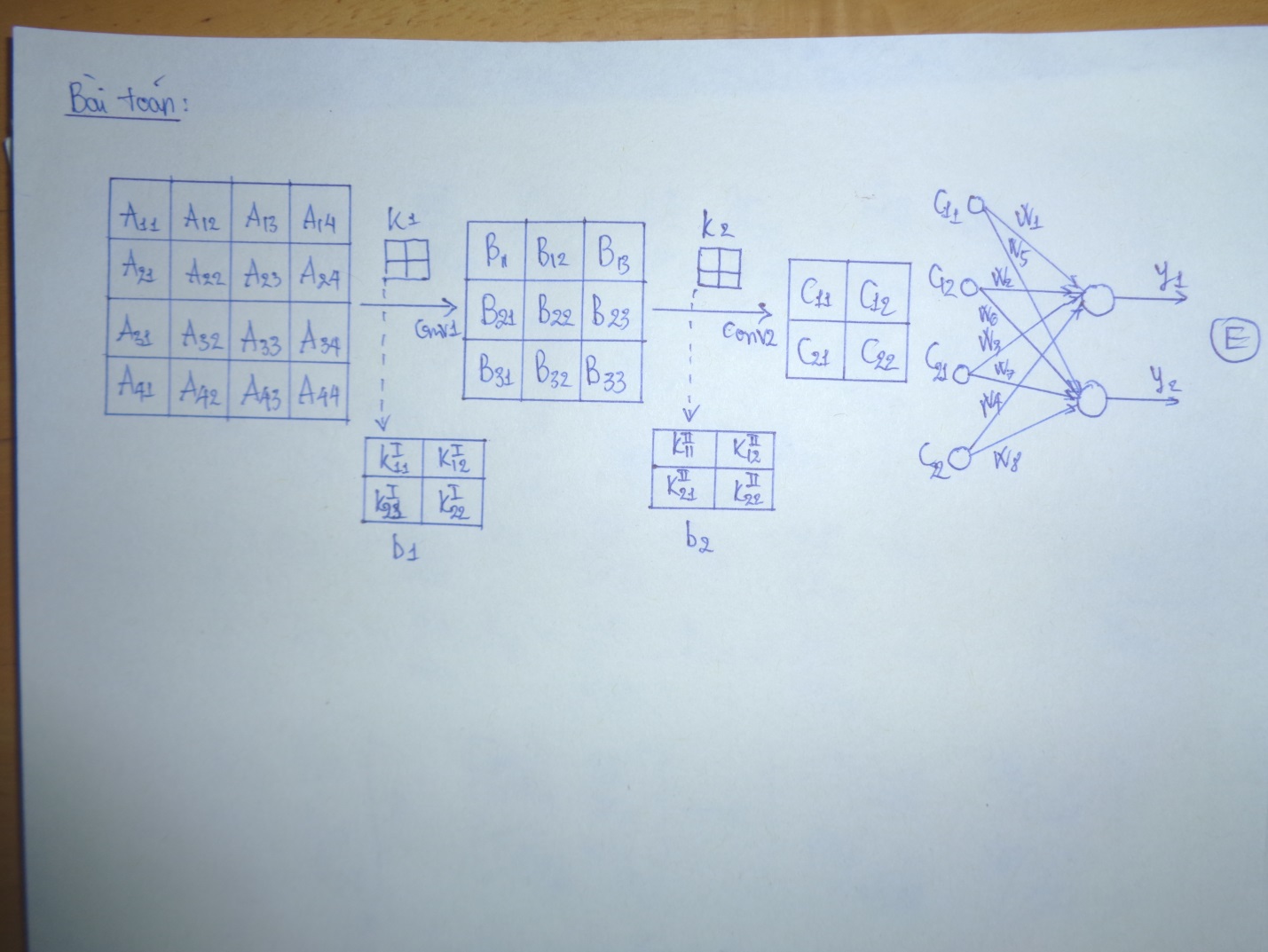
BÀI TẬP NHÓM SỐ 4

Giảng viên: Phạm Việt Cường

**Bài tập 1:**

Cho mạng CNN như **Hình 1**.Tính các đạo hàm sau và lập trình kiểm chứng kết quả:



**

**Hình 1.** Sơ đồ một mạng CNN.

Giải:

Để giảm khối lượng cồng kềnh mà không mất đi mục tiêu kiểm tra của bài toán, nhóm 6 chủ động lược bỏ 2 lớp phi tuyến và pooling của mạng. Do đó, trong các tính toán và công thức phía sau sẽ không nhắc đến 2 quá trình này.

**Quá trình Forward Propagation:**

1. Ta thực hiện tích chập ma trận đầu vào A có kích thức 4x4 với cửa sổ K1 thành ma trận B có kích thước 3x3. Các phần tử của ma trận B có giá trị như sau:



















2. Ta thực hiện tích chập ma trận B có kích thức 3x3 với cửa sổ K2 thành ma trận C có kích thước 2x2. Các phần tử của ma trận C có giá trị như sau:









3. Cuối cùng, ta thực hiện fully-connected để tạo ra 2 output, giá trị của chúng được cho bởi công thức sau:





4. Tính sai số so với mục tiêu: Trước khi tính toán sai số, ta đặt các giá trị mục tiêu là *T1* và *T2*. Giá trị sai số đầu ra như sau:





Và



**Quá trình Back Propagation:**

1. Lan truyền từ sai số về ngõ ra *y1* và *y2*.





2. Lan truyền từ sai số về ma trận *C*:









3. Lan truyền từ sai số về KII:



Tương tự, ta có:







Đối với thành phần bias *b2*:



4. Lan truyền từ sai số về ma trận *B*:



Tương tự, ta có:

















Nhận xét: từ kết quả thu được, ta nhận thấy *dB = full\_convolution(dC\*k).* Điều này sẽ được sử dụng trong phần demo code Python của nhóm.

5. Lan truyền từ sai số về ma trận *K1*:

Tương tự, ta có:



Đối với thành phần bias *b1*:



**Thử nghiệm mô phỏng:**

Code python nằm trong file HW4\_G6\_Code1.py và các hàm nằm trong file HW4\_G6\_Code2.py.

Trong file HW4\_G6\_Code2.py chứa 4 hàm bao gồm:

+ forwardconv\_func(): tính ma trận output sau khi cho ma trận đầu vào qua lớp tích chập.

+ forwardfully\_func(): tính *y1, y2* khi ma trận đầu vào qua lớp fully connected.

+ backfully\_func(): tính đạo hàm lan truyền từ sai số đến ma trận đầu vào của lớp fully connected.

+ backconv\_func(): tính đạo hàm lan truyền từ sai số đến ma trận đầu vào của lớp tích chập.

file HW4\_G6\_Code1.py gồm 2 phần:

+ Tính các tỉ số:



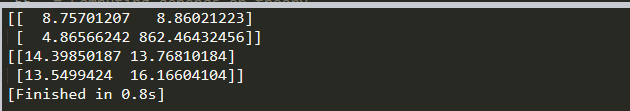
Bằng cách sử dụng thay đổi các giá trị *k* và *bias* từ -5 đến +5% giá trị của chúng, nhóm thực hiện tính hai lần các giá trị *y1*và *y2.* Sau đó, tính giá trị *E* và lập tỉ số sai lệch.

+ Tính các tỉ số:



Dựa vào công thức lý thuyết đã chứng minh ở trên, nhóm thực hiện sử dụng các hàm backconv\_func(), backfully\_func() và các giá trị trước khi thay đổi của k và bias lan truyền ngược tính các tỉ số.

So sánh các giá trị này để xem lý thuyết chứng minh có đúng hay không. Kết quả mô phỏng đối với hai giá trị  và  được mô tả trong Hình 2.



Hình 2. Kết quả thu được.

Kết quả thu được cho thấy 2 ma trận này khác nhau. Nguyên nhân nằm ở việc chọn không đúng hai giá trị *Target1* và *Target2* cho đầu ra *y1* và *y2*. Mục tiêu chọn các thông số sao cho sự thay đổi của *k* và *E* ngõ ra nhỏ để kết quả gần đúng với phép tính đạo hàm. Ở đây, việc chọn *k* thay đổi trong khoảng 5% giá trị đã được đáp ứng, tuy nhiên việc kiểm soát giá trị *E* trước và sau khi thay đổi giá trị k dựa vào biến *Target* là hoàn toàn khó khăn. Các giải pháp được đưa ra:

+ Chọn Target1 = 1.0005\*y1 và Target2 = 1.0005\*y2 để cho rằng mạng đã hội tụ, tuy nhiên dentaE thay đổi quá lớn so với giá trị E1.

+ Chọn Target1 và Target2 là các hằng số, giá trị  không kiểm soát được.

Cả 2 giải pháp thất bại. Trong tương lai, nhóm sẽ thực hiện một mạng CNN hoàn chỉnh có đầu đủ các lớp phi tuyến và pooling, training đến khi giá trị E hội tụ và so sánh với kết quả lý thuyết sẽ cho kết quả thuyết phục.