Song song hóa thuật toán Lan truyền ngược trong bài toán huấn luyện mạng nơ-ron để nhận diện chữ viết tay.

Nguyễn Thành Vinh, Hoàng Việt Anh, Nguyễn Quỳnh Ạnh.

1. Giới thiệu

Thuật toán lan truyền ngược (Backpropagation - BP) là nền tảng của việc huấn luyện các mạng nơ-ron sâu hiện đại. Nó hoạt động bằng cách tính toán gradient của hàm mất mát (loss function) đối với các trọng số của mạng, từ đó cập nhật trọng số để giảm thiểu sai số. Quá trình này được thực hiện qua nhiều lượt (epoch) để mô hình học được một cách trọn vẹn dữ liệu.

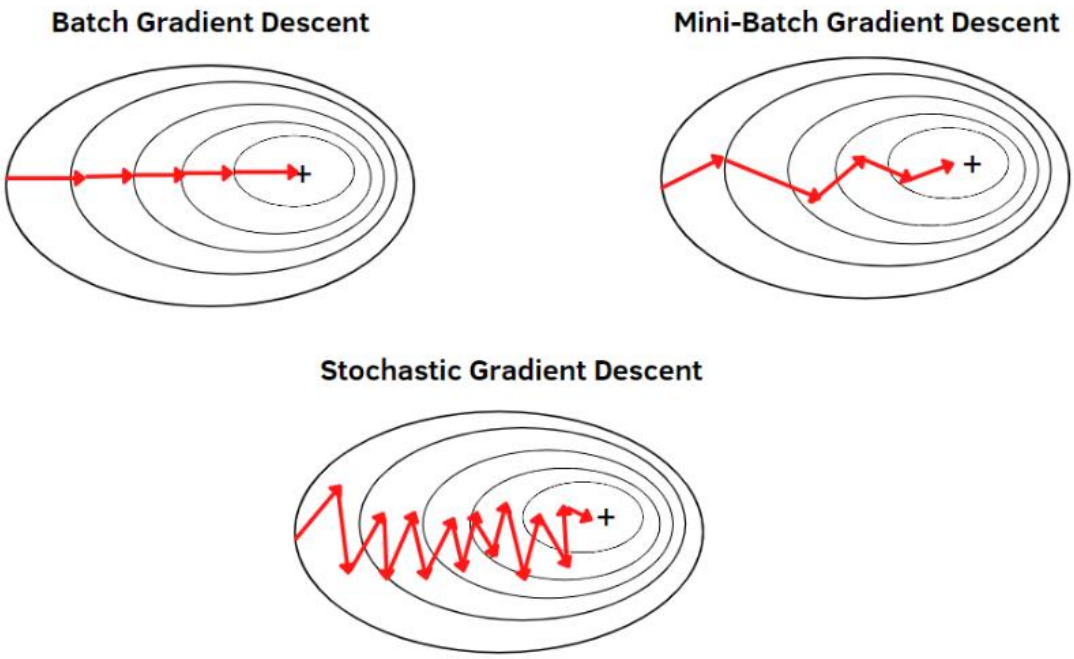
Thường thì sẽ có 3 cách để thực hiện thuật toán này. Các cách này đều liên quan đến việc chia dữ liệu ra sao để mô hình học được một cách tốt nhất: **Batch Gradient Descent, Stochastic Gradient Descent, Mini-Batch Gradient Descent**.

Batch Gradient Descent (BGD) tính toán gradient của hàm mất mát trên toàn bộ tập dữ liệu nên chỉ cập nhật trọng số 1 lần 1 epoch. Do được tính toán trên toàn bộ tập dữ liệu nên cách này tính ra gradient thực sự của toàn bộ tập dữ liệu, vì thế đường đi gradient thường sẽ ổn định và dễ dàng đạt đến diểm tối ưu. Tuy ổn định và dễ dàng đạt điểm tối ưu, BGD lại không thể lúc nào cũng đến được điểm tối ưu toàn cục. Ngoài ra, do việc sử dụng cả tập dữ liệu nên dẫn đến việc tốn tài nguyên và thời gian tính toán lâu cho một lần cập nhật trọng số.

Stochastic Gradient Descent (SGD) chỉ tính toán gradient trên một dữ liệu thay vì toàn bộ và một epoch sẽ cập nhật trọng số nhiều lần (bằng kích thước tập dữ liệu). SGD khắc phục trực tiếp những nhược điểm của BGD, do đó cách này thường nhanh hơn, yêu cầu bộ nhớ thấp hơn và có khả năng thoát khỏi điểm tối ưu địa phương. Nhưng do chỉ tính gradient của một dữ liệu mỗi lần nên nó không phản ánh đúng gradient của toàn bộ dữ liệu, dẫn đến đường đi gradient thường "zigzag" và lâu hội tụ hơn BGD.

Mini-Batch Gradient Descent là sự kết hợp giữa BGD và SGD, cho phép tính toán gradient trên một lô con (mini-batch) dữ liệu thay vì toàn bộ tập dữ liệu hoặc chỉ một dữ liệu. Mỗi epoch cũng sẽ cập nhật trọng số nhiều lần. Cách tính này cân bằng tốt 2

cách tính ban đầu, ngoài ra nó còn giúp tối ưu hóa phần cứng để đạt hiệu suất tốt nhất. Do đó nó luôn là lựa chọn hàng đầu trong các bài toán huấn luyện mô hình hiện nay.



Hình 1. So sánh đường đi gradient của 3 cách tính.

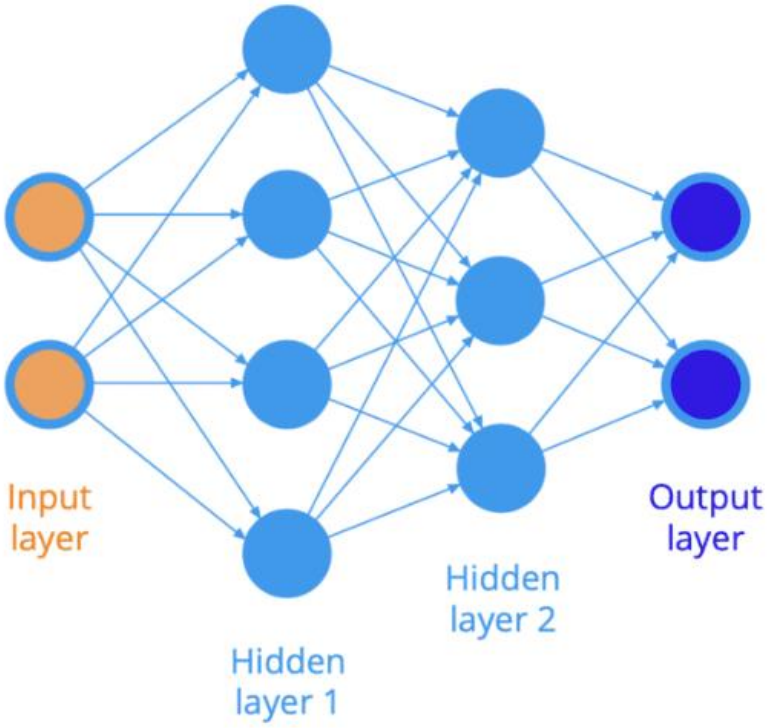
Ta có thể dễ dàng nhận ra được, thuật toán lan truyền ngược đang được thực hiện một cách tuần tự. Chỉ khi việc tính toán gradient và cập nhật trọng số của batch trước được thực hiện xong thì batch tiếp theo mới được thực hiện, vì thế nó chính là một trong những nguyên nhân dẫn đến thuật toán thực hiện chậm. Vì vậy, việc **song song hóa** quá trình lan truyền ngược là cần thiết nhằm **tăng tốc độ huấn luyện**, **tận dụng tài nguyên đa lõi CPU/GPU**, và **nâng cao hiệu suất hệ thống**.

Trong báo cáo lần này, nhóm chúng em sẽ giới thiệu hai cách để song song hóa thuật toán lan truyền ngược và sau đó áp dụng vào bài toán huấn luyện mạng nơ-ron để nhận diện chữ viết tay.

2. Tuần tự trong mạng lan truyền ngược

Trước hết, ta cần hiểu được cách thuật toán lan truyền ngược hoạt động như thế nào thông qua một mạng nơ-ron đơn giản (Hình 2).

Kiến trúc một mạng nơ-ron bao gồm lớp đầu vào (Input Layer), lớp ẩn (Hidden Layer) và lớp đầu ra (Output Layer). Đây cũng sẽ là kiến trúc được nhóm sử dụng để nhận diện chữ viết tay. Nhóm sẽ không sử dụng các kiến trúc CNN hiện đại trong báo cáo này vì mục tiêu chủ yếu của báo cáo là tập trung vào phần thuật toán, do đó lựa chọn mô hình nơ-ron đơn giản là hợp lí.



Hình 2. Mạng nơ-ron.

2.1. Quy trình huấn luyện tuần tự

2.1.1. Lan truyền xuôi

$$a_i = f(W_i * a_{i-1} + b_i)$$

Trong đó:

- a_i: Vectơ đầu ra của lớp thứ i sau hàm kích hoạt
- a_{i-1}: Vectơ đầu ra của lớp trước (hoặc lớp đầu vào nếu i=1)
- W_i: Ma trận trọng số giữa lớp i và i – 1
- b_i: Vectơ bias của lớp i
- $f(x)$: Hàm kích hoạt ,ví dụ:
  - Hàm sigmoid: $\frac{1}{1+e^{-x}}$
  - Hàm ReLU: $max(0, x)$
  - Hàm tanh: $\frac{e^x - e^{-x}}{e^x + e^{-x}}$

2.1.2. Tính toán sai số

$$L = \frac{1}{2M}\sum_{i=1}^{M}\left(a_O^i - y^i\right)^2$$

Trong đó:

- M: số lượng dữ liệu trong 1 lô con.

- L : Hàm mất mát – giá trị cần tối thiểu hóa
- y^i: Giá trị mục tiêu của mẫu thứ i
- a_O^i: Giá trị dự đoán của mạng cho mẫu thứ i

2.1.3. Lan truyền ngược

a. Truyền ngược tại lớp đầu ra:

$$\delta_{\mathrm{O}} = \frac{1}{\mathrm{M}}\sum_{\mathrm{i}=1}^{\mathrm{M}}\left(\mathrm{a}_{\mathrm{O}}^{\mathrm{i}} - \mathrm{y}^{\mathrm{i}}\right)\mathrm{f}'\left(\mathrm{z}_{\mathrm{O}}^{\mathrm{i}}\right)$$

Trong đó :

- $z_O = W_i * a_{i-1} + b_i$
- δ_{O}: Gradient tại lớp đầu ra (Đạo hàm hàm lỗi)
- $f'(z_{\mathrm{I}})$: Đạo hàm của hàm kích hoạt tại lớp cuối

b. Truyền ngược qua các lớp:

$$\delta_{\mathrm{i}} = (W_{i+1})^T \delta_{\mathrm{i+1}} \cdot f'(z_{\mathrm{i}})$$

Trong đó:

- δ_{i}: Gradient tại lớp ẩn thứ i
- $(W_{i+1})^T$: Ma trận trọng số của lớp sau
- $\delta_{\mathrm{i+1}}$: Gradient từ lớp sau truyền ngược
- $f'(z_{\mathrm{i}})$: Đạo hàm hàm kích hoạt tại lớp i

2.1.4. Cập nhật trọng số

$$W_{\mathrm{i}} = W_{\mathrm{i}} - \eta \cdot \delta_{\mathrm{i}}(a_{\mathrm{i-1}})^T$$

Trong đó:

- W_i: Trọng số lớp hiện tại
- η: Tốc độ học
- δ_{i}: Gradient của lớp i (Đạo hàm hàm lỗi)
- $(a_{\mathrm{i-1}})^T$: Đầu ra của lớp trước

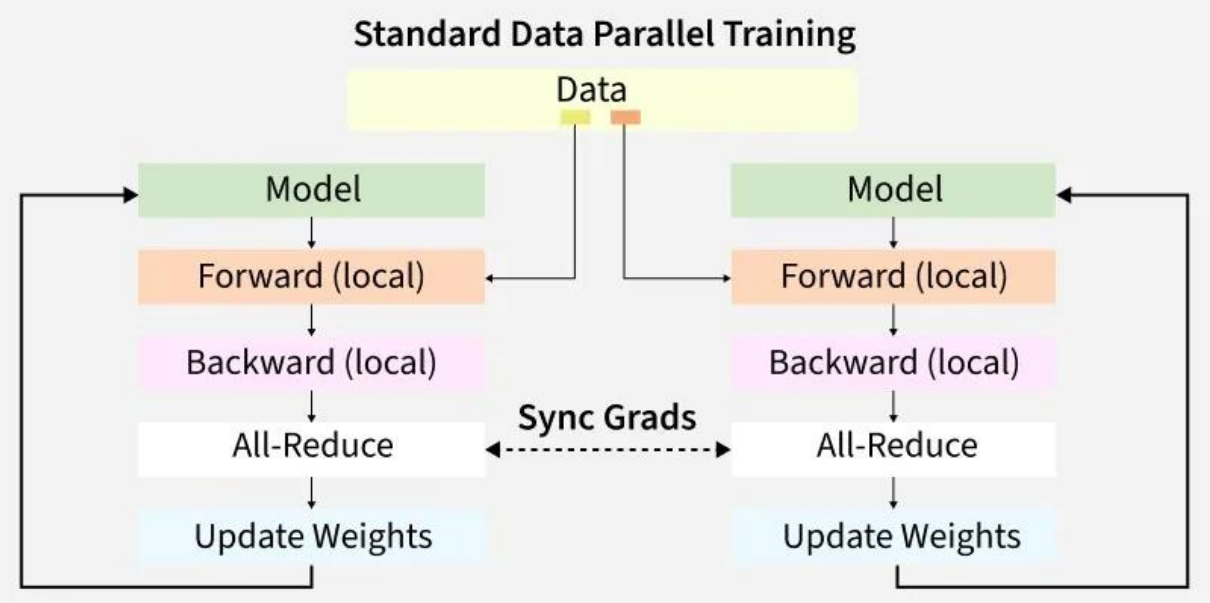
3. Song song hóa lan truyền ngược

3.1. Song song dữ liệu (Data Parallel)

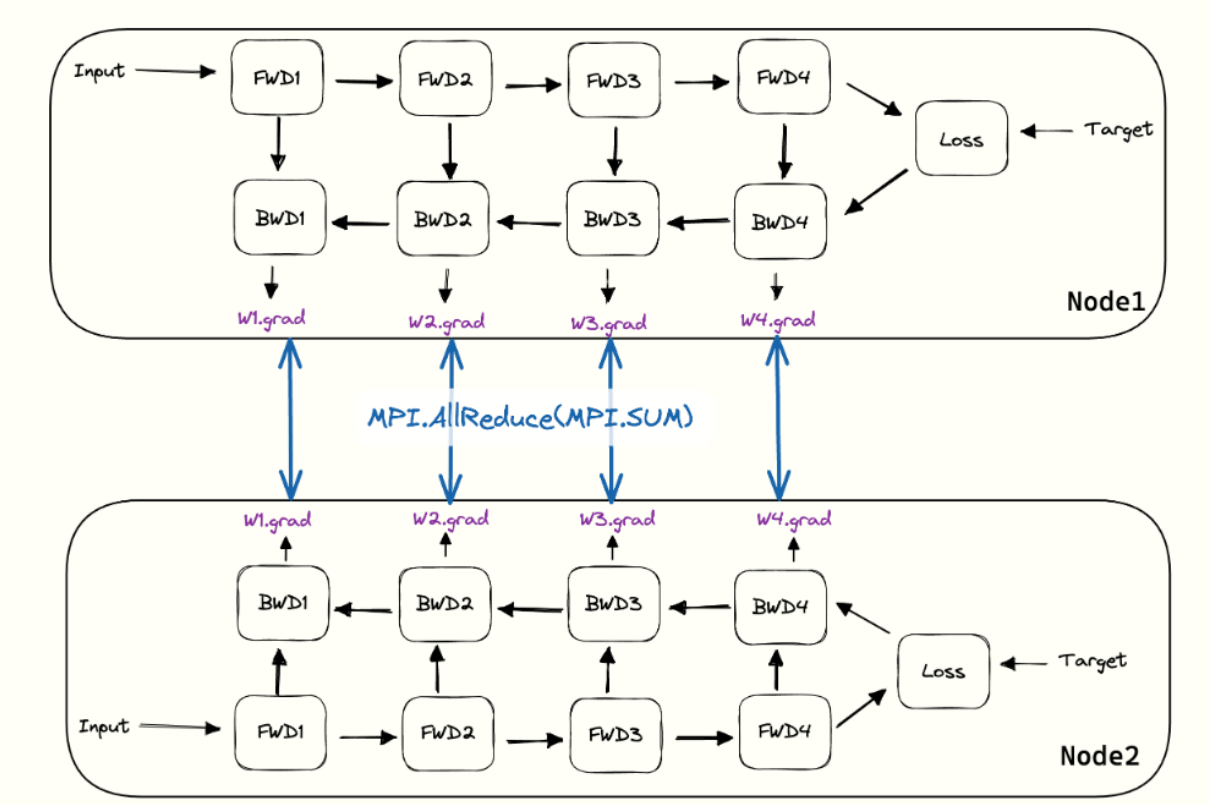
- Cơ chế: Mỗi tiến trình giữ một bản sao hoàn chỉnh của mô hình. Một lô dữ liệu lớn được chia nhỏ thành các lô con và gửi đến từng tiến trình.
- Quy trình:
  - Mỗi tiến trình thực hiện lan truyền xuôi và lan truyền ngược độc lập trên lô con dữ liệu của mình để tính gradient cục bộ.

- Bước đồng bộ hóa: Các gradient cục bộ từ tất cả tiến trình được tổng hợp lại (thường dùng thuật toán *All-Reduce*) để tính gradient trung bình.
- Tất cả các tiến trình cập nhật mô hình của mình với cùng một gradient trung bình, đảm bảo các bản sao mô hình luôn giống nhau.

- Thách thức: Chi phí truyền thông khi đồng bộ gradient có thể trở thành nút thắt cổ chai nếu mạng chậm hoặc số lượng tham số quá lớn



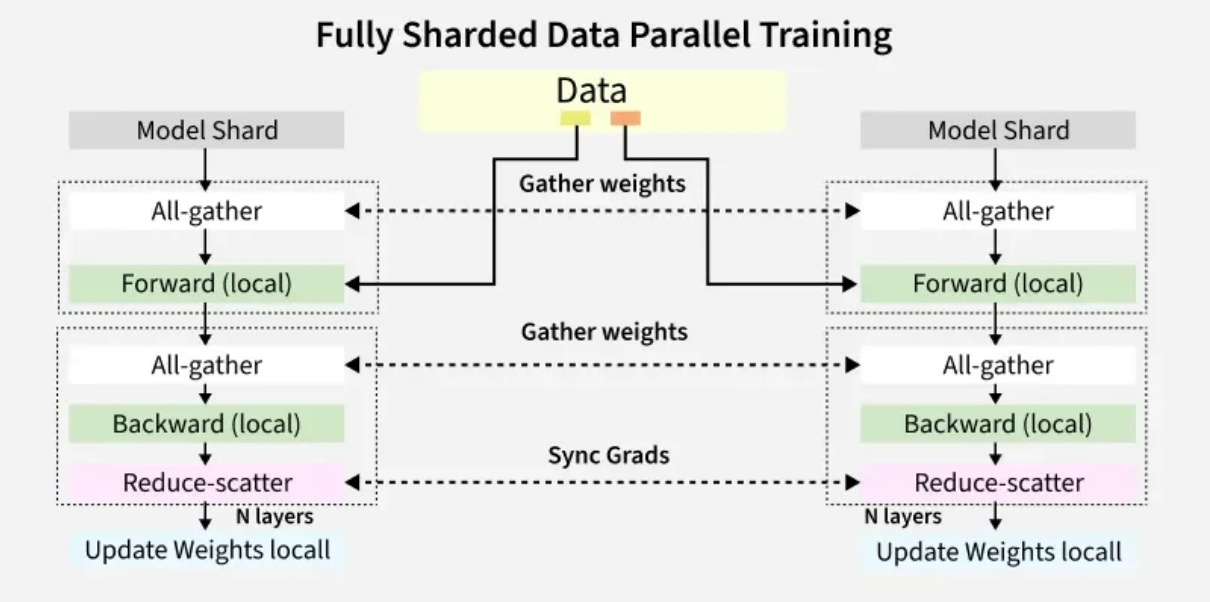
Hình 3. Cách hoạt động của Data Parallel.



Hình 4. Gradient của mỗi lớp được tính trung bình cộng qua các tiến trình.

3.2. Song song dữ liệu phân mảnh (Fully sharded data parallel)

- Cơ chế: Không giống như DP trong FSDP mỗi tiến trình chỉ giữ 1 phần trọng số của toàn mô hình và mỗi tiến trình sẽ tạm thời thu thập các trọng số còn thiếu của các tiến trình khác và giải phóng sau khi sử dụng.
- Quy trình**:**
    - Lan truyền xuôi (Forward Pass): Khi cần tính toán cho một lớp mạng cụ thể, mỗi tiến trình sẽ tạm thời thu thập các trọng số còn thiếu từ tiến trình đang nắm giữ trọng số của lớp đó (sử dụng thao tác *All-Gather*). Sau khi tính toán xong đầu ra của lớp, nó lập tức giải phóng các trọng số vừa thu thập để tiết kiệm bộ nhớ.
    - Lan truyền ngược (Backward Pass): Tương tự như lan truyền xuôi, tiến trình thu thập lại các trọng số đầy đủ của lớp đang được tính đạo hàm để tính gradient cục bộ.
    - Đồng bộ & Giảm gradient (Reduce-Scatter): Sau khi tính xong gradient cục bộ, thay vì tổng hợp toàn bộ gradient như DDP, FSDP sử dụng thao tác *Reduce-Scatter*. Mỗi tiến trình chỉ nhận về và lưu trữ phần gradient trung bình tương ứng với phần trọng số mà nó quản lý.
    - Cập nhật trọng số**:** Mỗi tiến trình cập nhật phần trọng số mà nó đang nắm giữ bằng cách sử dụng phần gradient và trạng thái tối ưu hóa cục bộ của mình.
- Việc chạy FSDP với nhiều tiến trình trên cùng một GPU vật lý là không được khuyến khích và đi ngược lại mục đích thiết kế của nó. Nó không giải quyết được vấn đề thiếu bộ nhớ mà còn làm giảm hiệu suất do tranh chấp tài nguyên. FSDP chỉ thực sự hiệu quả khi mỗi tiến trình kiểm soát một GPU vật lý riêng biệt



Hình 5. Cách hoạt đông của FSDP.

4. Thử nghiệm và đánh giá kết quả.

Hai phương pháp song song hóa được nêu ở **phần 3** đều có ưu và nhược điểm đối lập nhau. Trong khi DP dễ dàng để triển khai nhưng lại rất tốn tài nguyên, thì FSDP lại chỉ cần ít tài nguyên tuy nhiên lại đánh đổi bằng việc khó triển khai hơn và cần nhiều hơn một CPU/GPU để thực hiện (do lo ngại về việc chia sẻ trọng số giữa các tiến trình). Vì thế, nhóm đã quyết định lựa chọn phương pháp DP để thực hiện song song hóa trong báo cáo lần này.

Bộ dữ liệu mà nhóm sử dụng là bộ 70000 ảnh chữ viết tay **MNIST**. Bộ dữ liệu sẽ được chia ra thành như sau: 60% cho huấn luyện (training), 20% cho đánh giá (validation) và 20% cho thử nghiệm (testing). Ảnh đầu vào có kích thước 28x28 và sẽ được trải phẳng thành một vec-tơ 784 chiều làm đầu vào cho mạng nơ-ron.

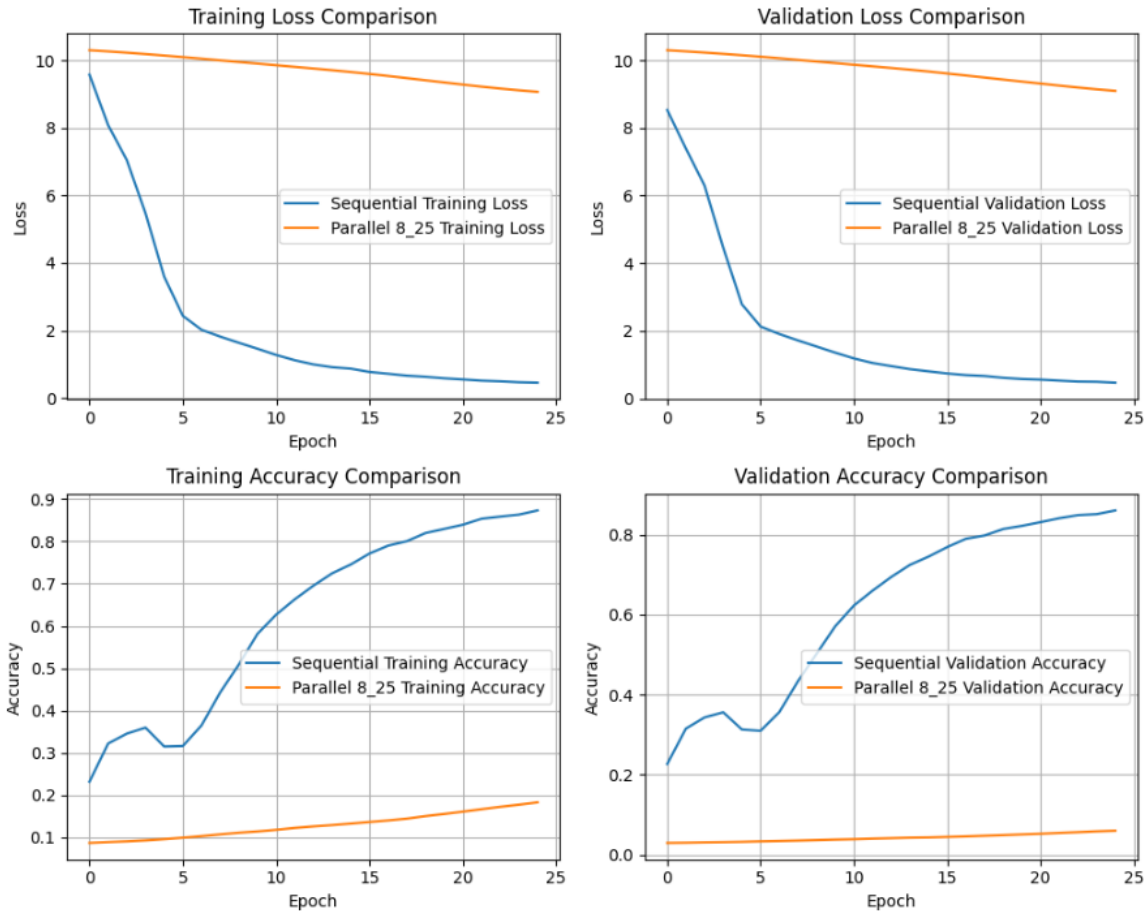
Kiến trúc mạng nơ-ron nhóm sử dụng như sau: lớp đầu vào có 784 nốt; tầng ẩn có 2 lớp, mỗi lớp có 16 nốt; lớp đầu ra có 10 nốt tương ứng với các con số từ 0-9. Mô hình sẽ được huấn luyện theo cả hai phương pháp song song và tuần tự nhằm so sánh về các tiêu chí như thời gian hội tụ, hiệu suất mô hình, …

Mô hình tuần tự sẽ sử dụng `batch_size = 1024` để huấn luyện. Trong khi đó, nếu cũng sử dụng `batch_size` như mô hình tuần tự thì mô hình song song sẽ cần đến xấp xỉ 42 tiến trình để chia dữ liệu. Điều này là không khả thi với máy tính cá nhân, cho nên nhóm đã quyết định chỉ sử dụng 8 tiến trình, xấp xỉ 5250 dữ liệu cho mỗi tiến trình.

Hình 6 cho thấy rằng cả hai mô hình có giá trị loss (training_loss) giảm và không bị overfitting (val_loss giảm cùng loss). Tuy nhiên, mô hình tuần tự đạt hiệu suất tốt hơn hẳn so với mô hình song song. Về mặt thời gian chạy thì mô hình song song nhanh hơn mô hình tuần tự (35.85s so với 53.53s).

Lí giải cho việc tại sao mô hình song song lại đạt hiệu suất yếu hơn, ta để ý vào số bước cập nhật trọng số của mô hình trong một epoch. Trong khi mô hình tuần tự với `batch_size=1024` có đến 41 bước cập nhật trọng số trong một epoch, thì đối với mô hình song song chỉ có một bước trong một epoch (đã được nói đến ở phần 3). Điều này đồng nghĩa với việc mô hình song song hoạt động không khác gì một mô hình sử dụng phương pháp BGD và nhược điểm chính của phương pháp này đó chính là thời gian hội tụ lâu. Để khắc phục vấn đề này, nhóm đã tăng epoch lên và đề xuất công thức sau để chỉnh sửa tốc độ học cho các mô hình song song:

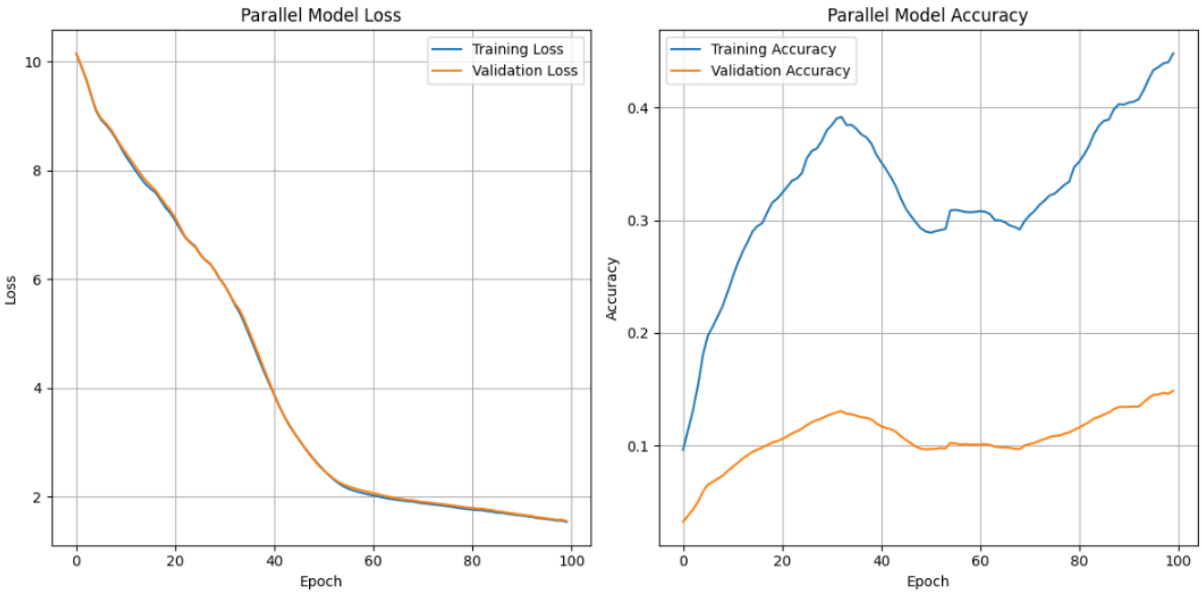
$$lr_{para} = lr_{seq} * \frac{B_{para}}{B_{seq}}$$



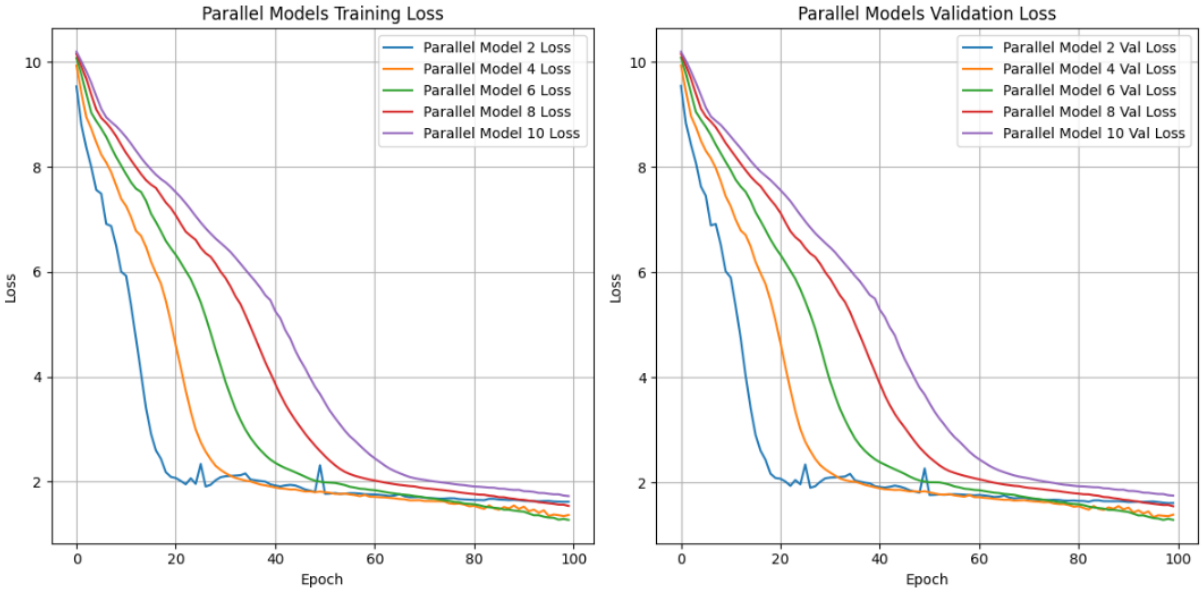
Hình 6. So sánh hai mô hình trên cùng một bố siêu tham số: `epoch=25`, `lr=0.01`.

Trong đó, lr_{seq} và lr_{para} lần lượt là tốc độ học của mô hình tuần tự và song song, B_{seq} và B_{para} lần lượt là batch_size của mô hình tuần tự và batch_size của một tiến trình trong mô hình song song. Công thức này đã khắc phục vấn đề hội tụ lâu của mô hình song song nhưng không phải hoàn toàn vì bản chất nó vẫn giống một mô hình sử dụng phương pháp BGD. Giải pháp tốt nhất cho vấn đề này vẫn là tăng số lượng tiến trình lên nhiều lần (điều mà máy tính cá nhân không làm được).

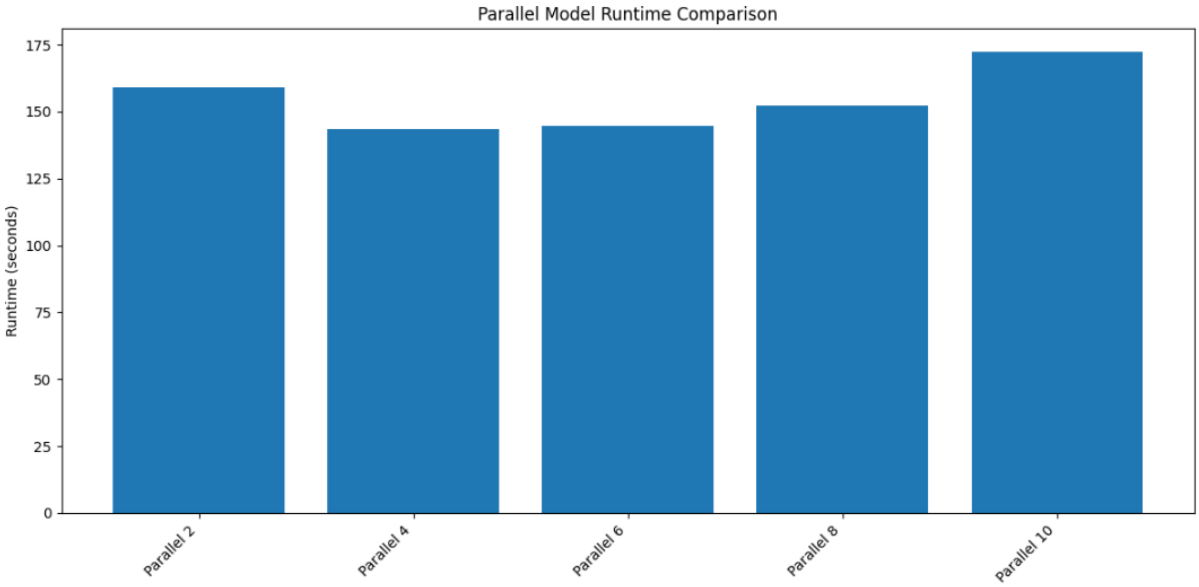
Nhóm cũng đã thực hiện thay đổi số lượng tiến trình nhằm tìm ra tham số tối ưu nhất và số lượng 6 tiến trình là phù hợp nhất cho máy tính cá nhân kể cả tiêu chí hiệu suất và thời gian chạy chương trình (Hình 8, Hình 9). Các mô hình được huấn luyện trong 100 epoch và tốc độ học được cập nhật theo công thức đã đề cập.



Hình 7. Hiệu suất mô hình song song sau khi đã tăng `epoch` và cập nhật `lr`.



Hình 8. So sánh các mô hình song song với số lượng tiến trình khác nhau.



Hình 9. Thời gian chạy của các mô hình song song.

Tài liệu tham khảo

[1]. Stanford CS231N. *Lecture 11: Large Scale Distributed Training.*

[2]. Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning representations by back-propagating errors.*