Nhóm 13:

* 52100911 - Võ Luyện
* 52100349 - Nguyễn Thanh Tú
* 52100674 - Trần Thị Vẹn

…………………………………………………………………………………………

1. Các bước xây dựng mô hình trên Pytorch:

* Bước 1: Chuẩn bị dữ liệu

Chuẩn bị tập train và tập test.

Chuyển đổi dữ liệu thành các tensor.

* Bước 2: Xác định kiến trúc mạng neural

Xác định số lượng lớp và các kích thước đầu vào/đầu ra của mạng neural.

Xác định kiểu lớp mạng neural cho từng lớp (ví dụ: fully connected layer, convolutional layer, recurrent layer, etc.).

Xác định các tham số của mỗi lớp mạng neural (ví dụ: số lượng neuron, kích thước kernel, số lượng tầng ẩn, hệ số học, hàm kích hoạt, etc.).

* Bước 3: Khởi tạo mô hình mạng neural

Kế thừa class nn.Module của PyTorch để tạo một lớp mô hình mạng neural.

Trong hàm khởi tạo (\_\_init\_\_) của lớp mô hình, khởi tạo các lớp mạng neural.

* Bước 4: Định nghĩa quá trình lan truyền thuận (forward pass)

Định nghĩa phương thức forward trong lớp mô hình, các hàm kích hoạt.

Trong phương thức forward, xác định quá trình lan truyền thuận của mạng neural bằng cách sử dụng các lớp mạng neural đã khởi tạo và các phép toán tính toán.

* Bước 5: Định nghĩa hàm mất mát (loss function)

Xác định hàm mất mát phù hợp.

PyTorch cung cấp nhiều hàm mất mát được tích hợp sẵn (ví dụ: nn.MSELoss(), nn.CrossEntropyLoss(), ...).

* Bước 6: Xác định thuật toán tối ưu hóa

Xác định thuật toán tối ưu hóa để cập nhật trọng số mô hình trong quá trình huấn luyện.

PyTorch cung cấp nhiều thuật toán tối ưu hóa như Stochastic Gradient Descent (SGD), Adam, RMSProp, …

Có thể tùy chỉnh các siêu tham số của thuật toán tối ưu hóa như learning rate, momentum, etc.

* Bước 7: Chuẩn bị vòng lặp huấn luyện

Xác định số lượng epoch và kích thước batch.

Chuẩn bị các biến để lưu trữ thông tin về quá trình huấn luyện như giá trị mất mát, độ chính xác, etc.

* Bước 8: Huấn luyện mạng neural

Lặp qua các epoch.

Trong mỗi epoch, lặp qua từng batch dữ liệu và thực hiện các bước sau:

* Xóa gradient trước mỗi lần cập nhật trọng số.
* Lan Truyền thuận (forward pass) dữ liệu qua mạng neural để tính toán đầu ra.
* Tính toán giá trị mất mát bằng cách so sánh đầu ra dự đoán với đầu ra thực tế sử dụng hàm mất mát.
* Thực hiện lan truyền ngược (backward pass) để tính gradient của hàm mất mát theo từng tham số của mạng neural.
* Cập nhật trọng số của mô hình.
* Cập nhật thông tin về giá trị mất mát và độ chính xác của mạng neural.

Bước 9: Đánh giá mô hình

Sau khi hoàn thành quá trình huấn luyện, đánh giá mạng neural trên dữ liệu kiểm tra để đánh giá hiệu suất của mô hình.

1. Khác nhau giữa 2 code Keras và Pytorch:

# **Processing data và Custom Dataset**

* Pytorch:
* Chuyển đổi dữ liệu thành các tensor PyTorch.
* Tạo một bộ dữ liệu tùy chỉnh bằng cách sử dụng lớp Dataset của PyTorch, cho phép tùy chỉnh dễ dàng của việc tải và xử lý dữ liệu.
* Sử dụng DataLoader của PyTorch để tải dữ liệu theo các batch và xáo trộn dữ liệu để huấn luyện hiệu quả.
* Keras:
* Sử dụng mảng NumPy để xử lý dữ liệu.
* Sử dụng StandardScaler từ scikit-learn để chuẩn hóa dữ liệu.
* Chia dữ liệu thành các tập train và test.

1. **Building a Neural Network**

* Pytorch:
* Định nghĩa lớp mô hình Model, trong đó có các lớp tuyến tính (Linear) và các hàm kích hoạt như Relu, Sigmoid và Tanh.
* Tạo một đối tượng mạng nơ-ron với số lượng đặc trưng đầu vào và đầu ra cụ thể.
* Định nghĩa hàm checkpoint để lưu trạng thái của mạng nơ-ron.
* Định nghĩa hàm mất mát (loss function) là BCELoss (Cross-Entropy Loss cho bài toán nhị phân).
* Định nghĩa các tối ưu hóa (optimizers) như SGD, Adam, RMSprop và AdamW.
* Keras:
* Định nghĩa một mô hình mạng nơron sâu bằng cách sử dụng Sequential và các lớp Dense tương ứng với các lớp tuyến tính trong PyTorch.
* Định nghĩa các tối ưu hóa tương tự như trong PyTorch, nhưng sử dụng các tối ưu hóa từ thư viện TensorFlow/Keras.

1. **Training**

Cú pháp định nghĩa mô hình:

* TensorFlow/Keras: Sử dụng Sequential hoặc lớp Model để định nghĩa mô hình, sau đó thêm các lớp bằng cách sử dụng phương thức .add()
* PyTorch: Định nghĩa một lớp mô hình bằng cách kế thừa từ lớp nn.Module và xác định quá trình lan truyền thuận trong phương thức forward.

Quá trình huấn luyện:

* TensorFlow/Keras: Sử dụng phương thức fit để huấn luyện mô hình. Cần định nghĩa hàm mất mát, tối ưu hóa và các metric để đánh giá mô hình trong quá trình huấn luyện.
* PyTorch: Tự viết vòng lặp huấn luyện, trong đó phải thực hiện các bước lan truyền thuận, tính toán mất mát, lan truyền ngược và cập nhật trọng số.

Tính toán mất mát và cập nhật trọng số:

* TensorFlow/Keras: Tính toán mất mát và cập nhật trọng số được thực hiện tự động trong phương thức fit.
* PyTorch: Tự viết code để tính toán giá trị mất mát và cập nhật trọng số mô hình sau mỗi lần lan truyền thuận và lan truyền ngược.

Lưu trữ checkpoint của mô hình:

* TensorFlow/Keras: Sử dụng phương thức model.save để lưu trữ checkpoint của mô hình trong quá trình huấn luyện.
* PyTorch: Tự viết hàm để lưu trữ checkpoint của mô hình, thông thường sử dụng torch.save.

Trong TensorFlow/Keras, sử dụng phương thức fit để đào tạo mô hình, làm cho quy trình đào tạo trở nên ngắn gọn hơn. Chỉ cần cung cấp dữ liệu đào tạo và kiểm tra cũng như các tham số như số lượng epoch và batch.

Trong TensorFlow/Keras, mất mát và cập nhật trọng số được thực hiện tự động bên trong phương thức fit. Không cần phải lo lắng về việc tính toán mất mát hoặc làm thủ công bất kỳ phép cập nhật trọng số nào.

Trong PyTorch, phải tự viết vòng lặp huấn luyện. Điều này đòi hỏi phải chủ động thực hiện lan truyền thuận, tính toán mất mát, lan truyền ngược và cập nhật trọng số. Mặc dù điều này có thể phức tạp hơn, nhưng nó cung cấp sự linh hoạt cao hơn trong việc tùy chỉnh và kiểm soát quá trình đào tạo.

Trong PyTorch, phải tự viết code để tính toán mất mát và cập nhật trọng số sau mỗi lần lan truyền thuận và lan truyền ngược. Điều này cung cấp sự linh hoạt cao hơn, nhưng cũng đòi hỏi một lượng lớn mã code hơn.