#1. Neural Network from Scratch

Lý thuyết và code được dựa trên video sau:

Building a neural network FROM SCRATCH (no Tensorflow/Pytorch, just numpy & math)

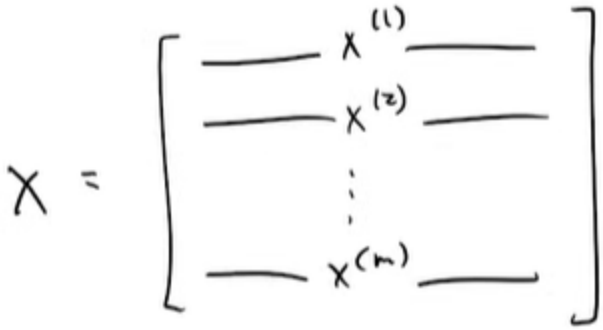
1. Problem

Lập trình một mạng neural “from scratch” để huấn luyện trên bộ dữ liệu MNIST. Không sử dụng các công cụ và thư viện có sẵn như Pytorch hay Keras mà sẽ chỉ sử dụng các thư viện như Numpy.

2. Math

Dữ liệu

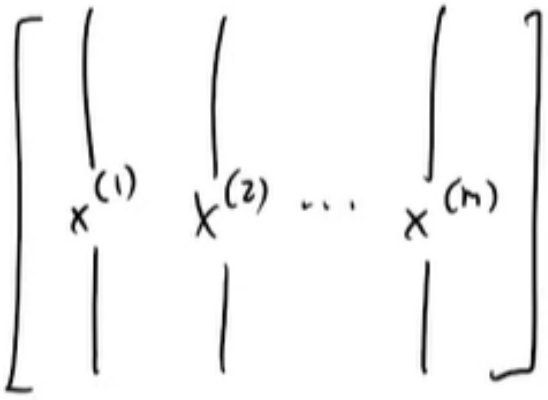
- Bộ dữ liệu MNIST bao gồm các bức ảnh có kích cỡ 28x28 tức là mỗi bức ảnh sẽ có tổng cộng 784 pixel. Mỗi pixel là một giá trị nằm trong khoảng từ [0 → 255].
- Các bức ảnh được sắp xếp thành 1 ma trận X kích cỡ $784 x m$ như sau:



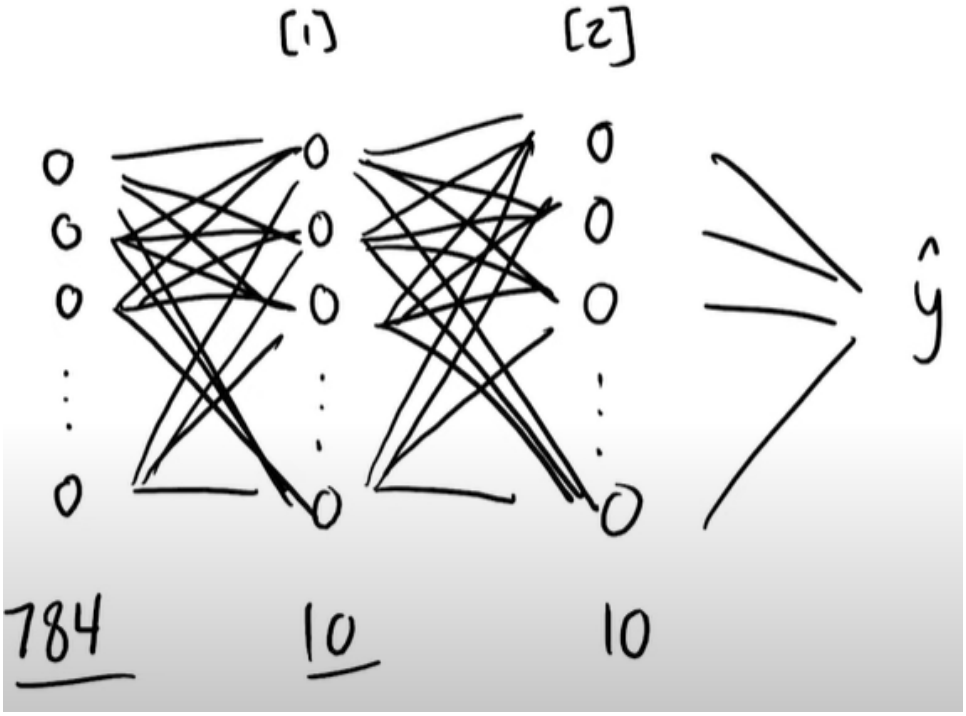
Với $x^1, x^2, \ldots, x^m$ tương ứng với các ảnh. Vì các ảnh nằm đè lên nhau nên để phù hợp cho việc đưa ảnh vào mô hình thì chúng ta phải thực hiện phép xoay ma trận (transpose).

💡 Phép transpose là một phép toán trong đại số tuyến tính, được sử dụng chủ yếu trong việc chuyển đổi vị trí của các hàng thành cột và ngược lại trong một ma trận. Nếu bạn có một ma trận *A* có kích thước *m* × *n*(m hàng và n cột), thì ma trận chuyển vị của *A*, ký hiệu là A^T, sẽ có kích thước *n* × *m*(n hàng và m cột).

Như vậy ma trận X của chúng ta sẽ có dạng như sau:



Mô hình Neural Network có cấu trúc đơn giản như sau:



- Input layer (1): gồm 784 node tương ứng với 784 pixel của input
- Hidden layer (2): gồm có 10 node
- Output layer(3): có 10 node tương ứng với 10 nhãn của bộ dữ liệu
- $\hat{y}$ là giá trị dự đoán đầu ra cho bài toán

Forward propagation

> 💡 Forward propagation (lan truyền tiến) là quá trình truyền dữ liệu qua mạng nơ-ron từ lớp đầu vào đến lớp đầu ra. Trong mô hình mạng nơ-ron, thông tin được truyền từ các đầu vào qua các lớp ẩn và cuối cùng đến lớp đầu ra. Trong quá trình này, các trọng số và độ lệch (bias) của mạng nơ-ron được sử dụng để tính toán giá trị đầu ra từ giá trị đầu vào.

Ta có các biến và quá trình xử lý sau:

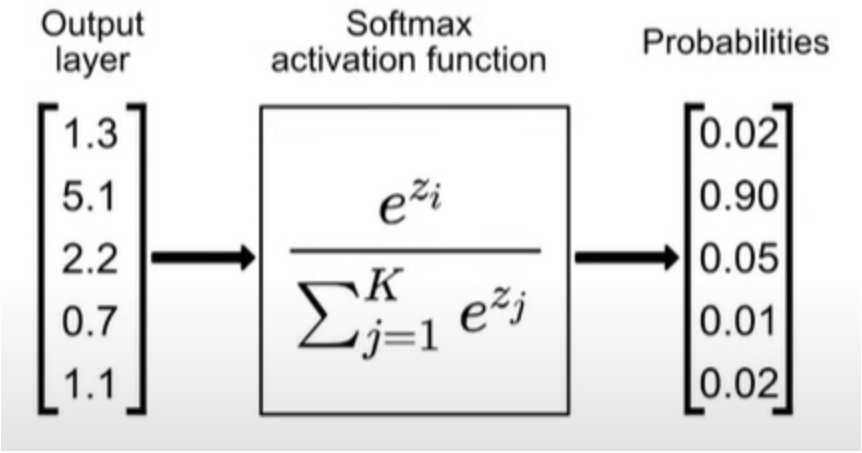
- $A^{[0]}$: tương ứng với input đầu vào, tức là 784 pixel ở input layer
- $z^{[1]}$: là giá trị lớp đầu tiên khi chưa kích hoạt, công thức như sau:

$$$$

- $A^{[1]}$: là giá trị sau khi đưa giá trị ở lớp đầu tiên đi qua một hàm kích hoạt $g(z^{[1]})$, ở đây chúng ta xử dụng hàm relu:

$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

- Quá trình tương tự với layer thứ 2 đến layer thứ 3 và chúng ta có các biến $A^{[2]}$ và $z^{[2]}$, tuy nhiên ở bước này chúng ta sẽ sử dụng một hàm kích hoạt khác là hàm softmax



💡 Hàm softmax là một hàm kích hoạt thường được sử dụng trong mô hình mạng nơ-ron để chuyển đổi đầu ra của một lớp thành một phân phối xác suất. Hàm này thường được áp dụng cho lớp đầu ra của mô hình trong các bài toán phân loại nhiều lớp. Các giá trị đầu ra sau khi áp dụng softmax sẽ nằm trong khoảng (0, 1) và có thể được coi là các xác suất dự đoán cho từng lớp.

- Cuối cùng, giá trị $\hat{y}$ được tính bằng cách lấy max của $A^{[2]}$

Backpropagation

💡 Backpropagation, viết tắt của "backward propagation of errors" (lan truyền ngược lỗi), là một phương pháp quan trọng trong quá trình huấn luyện mạng nơ-ron. Quá trình này giúp mô hình nơ-ron tự động điều chỉnh trọng số của các liên kết để làm giảm sai số (lỗi) giữa dự đoán và giá trị thực tế. Quá trình này giúp mô hình nơ-ron "học" từ dữ liệu huấn luyện và điều chỉnh các tham số để tối ưu hóa hiệu suất. Backpropagation dựa vào nguyên lý lan truyền ngược gradient để điều chỉnh trọng số của mạng nơ-ron dựa trên độ lỗi.

Ta có các biến sau:

- $dz^{[2]}$ đại diện cho số lỗi của lớp thứ 2, Y là nhãn thực tế của ví dụ

$$dz^{[2]} = A^{[2]} - Y$$

- $dw^{[2]}$ là đạo hàm của hàm loss ở lớp thứ 2

$$dw^{[2]} = \frac{1}{m} \times dz^{[2]} \times A^{[1]T}$$

- $db^{[2]}$ là số lỗi trung bình của mỗi ví dụ

$$db^{[2]} = \frac{1}{m} \times \sum dz^{[2]}$$

- Chúng ta có các biến tương tự ở lớp đầu tiên $dw^{[1]}$, $dz^{[1]}$ và $db^{[1]}$, tuy nhiên công thức có một số khác biệt:

$$dz^{[1]} = W^{[2]T} \times dz^{[2]} . \times g'(z^{[1]})$$

$$dw^{[2]} = \frac{1}{m} \times dz^{[1]} \times X^T$$

$$db^{[1]} = \frac{1}{m} \times \sum dz^{[1]}$$

Cập nhật weights

Ta có các công thức cập nhật trọng số như sau:

$$\begin{aligned} W_2 &:= W_2 - \alpha \cdot dW_2 \\ b_2 &:= b_2 - \alpha \cdot db_2 \\ W_1 &:= W_1 - \alpha \cdot dW_1 \\ b_1 &:= b_1 - \alpha \cdot db_1 \end{aligned}$$

Trong đó α là một giá trị learning rate được xác định trước