

Lời nói đầu

Trong khuôn khổ của đề tài này nhóm đã nghiên cứu chi tiết và có hệ thống về các thuật toán về Machine Learning nói chung, và ứng dụng neuromorphic computing trong bài toán nhận dạng nói riêng. Chúng tôi đã phát triển một phần mềm bằng C++ trên máy tính cho thuật toán nhận dạng dùng neuromorphic computing và sử dụng cơ sở dữ liệu của MNIST để kiểm định kết quả cho bài toán nhận diện kí tự số viết tay. Thiết kế sau đó được mô tả phần cứng VHDL và hiện thực hoá trên FPGA. Kết quả kiểm tra chạy thực tế trên mạch bước đầu cho thấy sự vượt trội về tốc độ xử lý (2000 lần nhanh hơn) sơ với việc thực thi trên máy tính với một mô hình hoàn toàn tương đương. Kết quả của đề tài cho thấy tính thực tiễn và khả năng ứng dụng của FPGA bao gồm khả năng xử lý thời gian thực và khả năng tích hợp vào hệ thống cho các ứng dụng tính toán thông minh nói chung và nhận diện hình ảnh nói riêng.

Mục lục

Lời nói đầu.....	1
Mục lục	2
I. Giới thiệu chung.....	3
1. Giới thiệu về Machine Learning.....	3
a. Unsupervised Learning	3
b. Supervised Learning.....	3
2. Neuron Network	4
II. Cơ sở toán học của thuật toán Softmax Regression.....	6
1. Softmax Regression	6
a. Xây dựng thuật toán	6
b. Tối ưu loss function.....	10
2. Gradient Descent	12
a. Stochastic Gradient Descent.....	14
b. Mini-batch Gradient Descent.....	14
c. Điểm dừng của thuật toán.....	15
III. Sơ đồ nhận dạng kí tự số viết tay sử dụng FPGA	17
1. Cơ sở dữ liệu	17
2. Cấu trúc chương trình C++.....	19
3. Cấu trúc các khối trên FPGA	20
IV. Tối ưu hoá thiết kế FPGA.....	23
1. Nén đầu vào.....	23
2. Hàm softmax	23
3. Thuật toán cập nhật ma trận W	23
4. Tài nguyên sử dụng	23
5. Tính toán tốc độ xử lí của FPGA	24
V. Kết quả	25
VI. Tài liệu tham khảo	32
VII. Phụ lục	33
1. Cấu trúc file database của MNIST	33
2. Cấu trúc ảnh BMP	34
3. Phần mềm hỗ trợ giao tiếp với mạch FPGA	34
4. Sơ đồ khối chi tiết của mô hình trên FPGA.....	36

I. Giới thiệu chung

1. Giới thiệu về Machine Learning

Machine Learning là một lĩnh vực của khoa học máy tính, giúp cho máy tính có khả năng "học hỏi" cách thực hiện một công việc nào đó mà không cần lập trình trước. Hiện nay tốc độ tính toán của máy tính đã đạt đến mức có thể tính toán hàng tỷ phép tính trong 1 giây, cùng với một lượng dữ liệu khổng lồ được tạo ra và thu thập thông qua mạng Internet, đây chính là mảnh đất màu mỡ cho các công ty công nghệ lớn đã tạo ra những bộ máy có thể làm được những việc mà trước giờ chỉ có con người làm được như phân loại cả ngàn vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn hay âm nhạc... với tốc độ và số lượng không một con người nào theo kịp. Một vài ví dụ về sản phẩm của các công ty lớn sử dụng trí tuệ nhân tạo với nền tảng là Machine Learning đã có là xe tự hành của Google và Tesla, hệ thống tự tag khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm của Amazon, hệ thống gợi ý phim của Netflix, máy chơi cờ vây AlphaGo của Google DeepMind...

Có nhiều cách phân loại Machine Learning, nhưng ta có thể phân loại dựa trên phương thức học như sau: Unsupervised Learning (học không giám sát) và Supervised Learning (học có giám sát)

a. Unsupervised Learning

Thuật toán *unsupervised learning* sẽ dựa vào cấu trúc của dữ liệu đầu vào để đoán ra được một hình mẫu hay một qui luật nào đó mà ta không biết được đầu ra chính xác. Vì thuật toán này hoạt động mà không biết chính xác đầu ra cho nên thuật toán này có cái tên "không giám sát".

Các bài toán ứng dụng thuật toán *unsupervised learning* gồm hai loại chính:

- Phân loại các dữ liệu đầu vào.
Ví dụ: Phân nhóm khách hàng dựa theo sở thích mua hàng của họ.
- Tìm ra qui luật của dữ liệu.
Ví dụ: Dự đoán kiểu mặt hàng mà khách hàng có thể đang quan tâm dựa trên lịch sử mua hàng của họ.

b. Supervised Learning

Supervised learning là thuật toán dự đoán đầu ra (outcome) của một dữ liệu mới (new input) dựa trên các cặp (input, outcome) đã biết từ trước. Cặp dữ liệu này còn được gọi là (data, label). Nhóm thuật toán này là nhóm thuật toán phổ biến nhất trong các thuật toán của Machine Learning.

Về mặt toán học, ta có thể coi tập hợp đầu vào là X , tập hợp đầu ra là Y . Khi đó:

$$X = \{x_1, x_2, x_3, \dots, x_n\}$$

$$Y = \{y_1, y_2, y_3, \dots, y_n\}$$

Tập hợp các cặp (x_i, y_i) biết trước được gọi là tập *training data*. Từ tập hợp này ta cần tìm được một phép ánh xạ các phần tử x_i sang y_i tương ứng:

$$y \approx f(x_i) \text{ với } \forall i = 1, 2, 3, \dots, n$$

Mục đích của thuật toán này là để khi có một phần tử x mới thì ta sẽ tìm được đầu ra y tương ứng:

$$y = f(x)$$

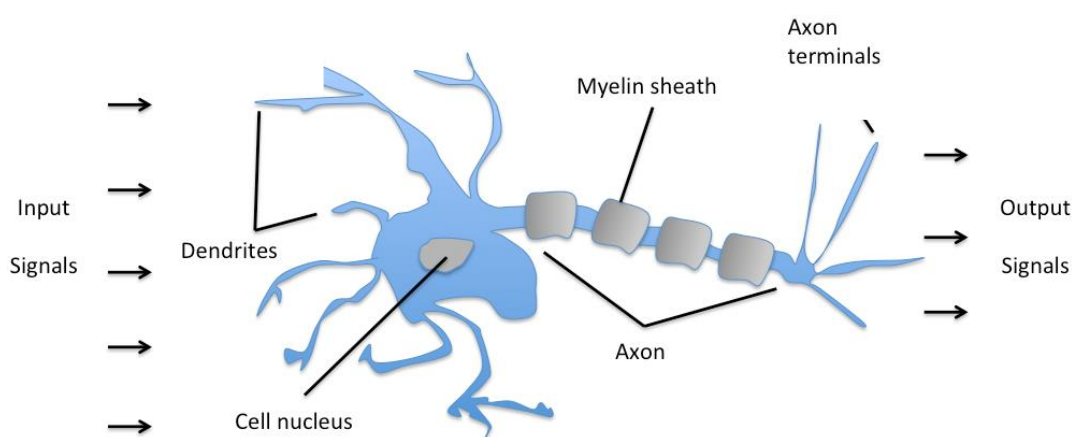
Thuật toán này rất hay được ứng dụng trong các bài toán như nhận dạng các sự vật, hoặc tìm ra phép xấp xỉ có rất nhiều đầu vào trong các bài toán thống kê. Neuron Network là một loại thuật toán có giám sát.

Ví dụ: Nhận dạng khuôn mặt, nhận dạng chữ viết tay, tìm ra qui luật của giá cả một mặt hàng,...

Thực tế, việc có được tập hợp label của một tập hợp data nhiều khi rất phức tạp và cần có chuyên môn cao để phân loại, xong lượng data rất lớn, giả sử như tập hợp các bệnh nhân với các triệu chứng khác nhau), thì ta có thể sử dụng Unsupervised learning để xếp nhóm các triệu chứng, sau đó các bác sĩ có thể gán nhãn cho từng nhóm bệnh đã được phân loại, từ đó ta có thể xây dựng được hệ thống chẩn đoán bệnh sử dụng máy tính.

2. Neuron Network

Neuron Network là một mô hình toán học mô phỏng hệ thần kinh bậc cao, với phần tử cơ bản là các neuron thần kinh. Một neuron thần kinh khi nhận được sự kích thích từ các neuron trước đó sẽ tạo ra một tín hiệu điện sinh học đến neuron kế sau. Với mạng lưới hàng tỷ neuron thì bộ não đã tạo được những nhận thức rất phức tạp để phản ứng lại các kích thích từ môi trường.



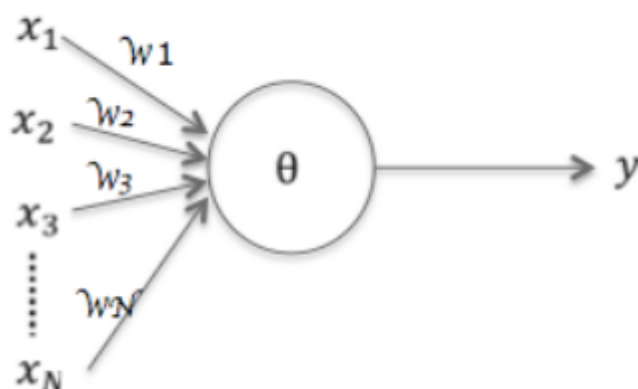
Hình I.2.1. Mô hình sinh học của một neuron

(Nguồn: http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html)

Ta có thể mô hình hoá thành toán học như sau:

$$y = \theta(w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_Nx_N)$$

với x_i là các đầu vào, w_i là các trọng số nhân với x_i thể hiện độ quan trọng của đầu vào đó.

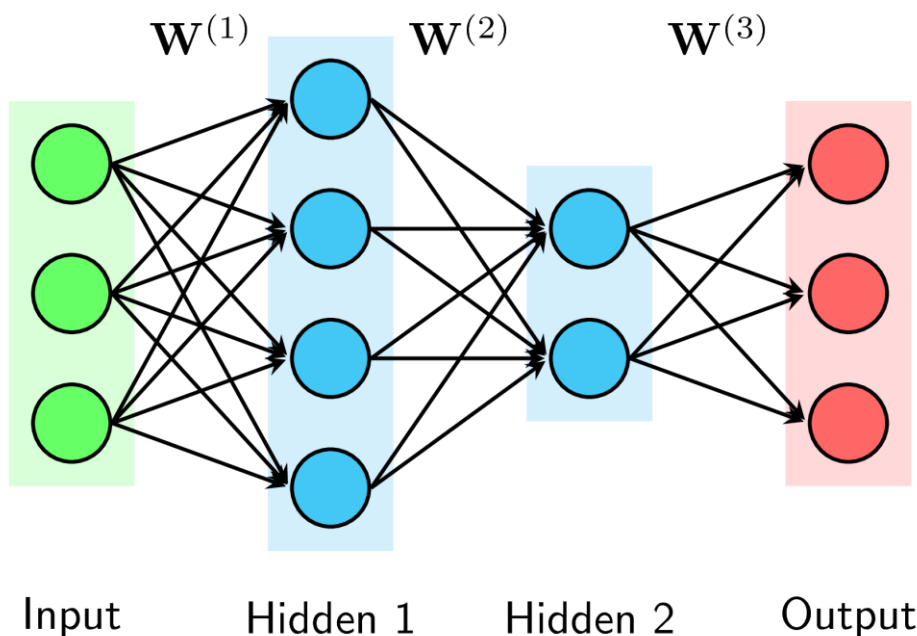


Hình I.2.2. Mô hình toán học của một neuron

(Nguồn: <https://codetudau.com/neural-network-va-deep-learning-la-gi/index.html>)

Đầu vào nào có ảnh hưởng càng lớn đến đầu ra thì sẽ có trọng số càng lớn. Hàm θ biểu thị việc xử lý các đầu vào bên trong neuron.

Đối với mô hình toán học, hàm số $\theta(w_1x_1 + w_2x_2 + \dots + w_nx_n)$ là một hàm dùng để giới hạn miền giá trị hoặc là biến đổi đầu ra y về một khoảng xác định tùy theo yêu cầu bài toán, được gọi là **activation function**. Kết nối các mô hình neuron trên vào một mạng lưới với đầu ra của neuron này là đầu vào của neuron sau, ta nhận được một Neuron Network.



Hình I.2.3. Mô hình mạng neuron với 3 đầu vào, 3 đầu ra và 2 lớp neuron trung gian

(Nguồn: <https://machinelearningcoban.com/2017/02/24/mlp/>)

Bài nghiên cứu này sẽ đi vào thiết kế mô hình 1-layer Neuron Network sử dụng thuật toán Softmax Regression áp dụng Stochastic Gradient Descent trên FPGA để nhận diện các ký tự số viết tay.

II. Cơ sở toán học của thuật toán Softmax Regression

1. Softmax Regression

a. Xây dựng thuật toán

Xét mạng neuron có d đầu vào, 1 đầu ra, không có lớp trung gian:

$$a = f(x) = f(w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d)$$

Đặt:

$\mathbf{x} = \{1, x_1, x_2, x_3, \dots, x_d\}^T$ là vector cột chứa các đầu vào,

y là số vô hướng biểu diễn đầu ra theo thống kê,

a là số vô hướng biểu diễn đầu ra theo tính toán,

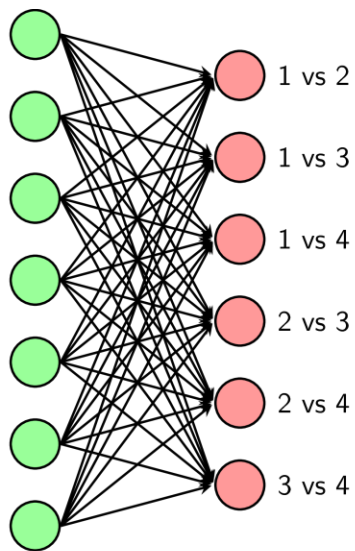
$\mathbf{w} = \{w_0, w_1, w_2, w_3, \dots, w_d\}^T$ là vector cột chứa các trọng số.

Ta có:

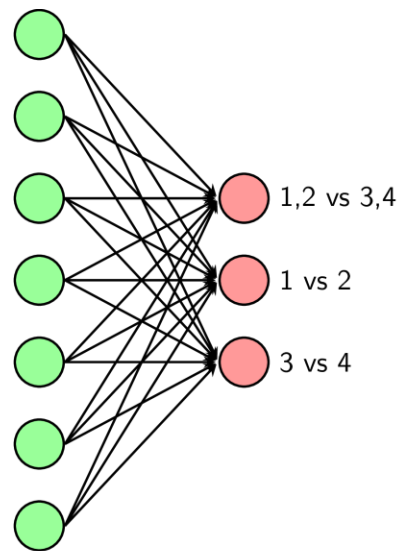
$$a = f(\mathbf{w}^T \mathbf{x})$$

Nếu a chỉ có 2 giá trị mỗi giá trị tương ứng với 1 class, ta gọi đây là mô hình binary classifier.

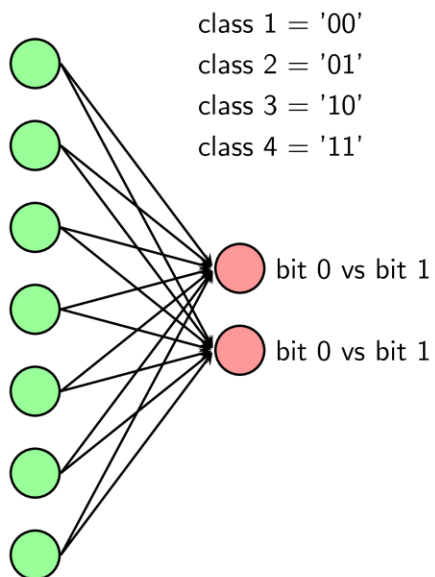
Softmax Regression là một thuật toán sử dụng kết hợp các binary classifier (phân loại 2 class) để tạo ra một bộ phân loại nhiều class với nhau.



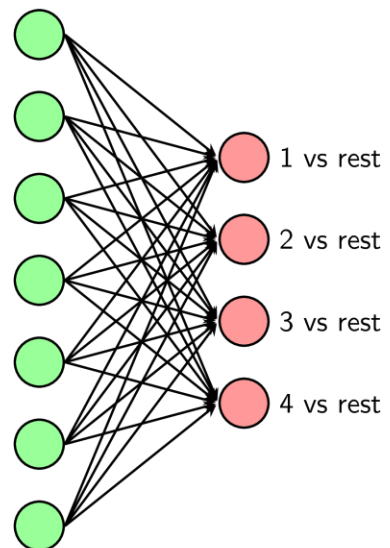
one-vs-one



Hierarchical



Binary coding

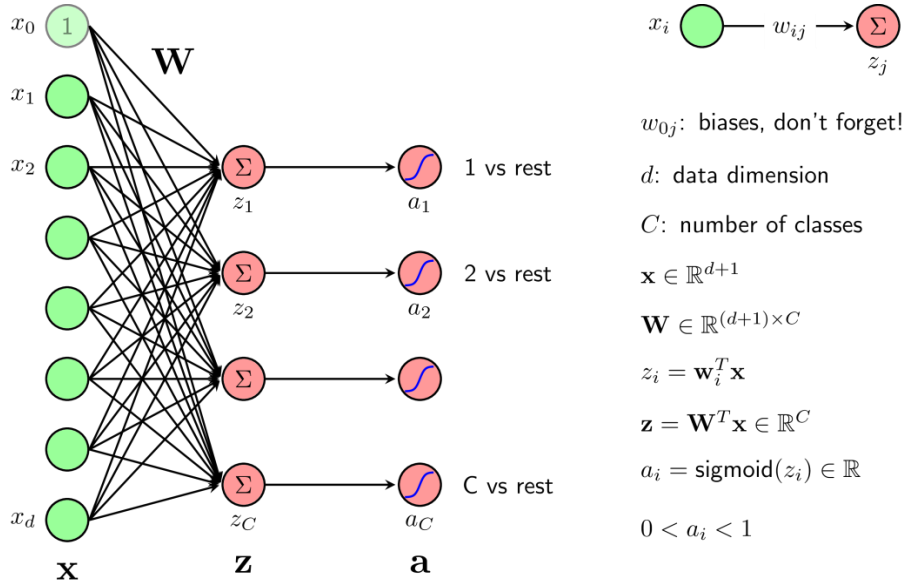


one-vs-rest

Hình II.5. Các phương pháp xây dựng bộ phân loại dựa trên binary classifier

(Nguồn: <https://machinelearningcoban.com/2017/02/11/binaryclassifiers>)

Ta sẽ sử dụng phương pháp one-vs-rest để xây dựng mô hình Softmax Regression. Như vậy với C class, ta cần xây dựng C bộ phân loại, tương ứng với C vector trọng số.



Hình II.6.a.1. Mô hình Neuron Network one-vs-rest sử dụng hàm sigmoid
(Nguồn: <https://machinelearningcoban.com/2017/02/17/softmax>)

Đặt $\mathbf{a} = \{a_1, a_2, a_3, \dots, a_C\}^T$ là vector cột đầu ra.

Đặt $W = \{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \dots, \mathbf{w}_C\}$ là ma trận trọng số, với \mathbf{w}_i là vector cột chứa các trọng số của một bộ binary classifier.

Công thức để tính một đầu ra là:

$$a_i = f(\mathbf{w}_i^T \mathbf{x}) = f(z_i)$$

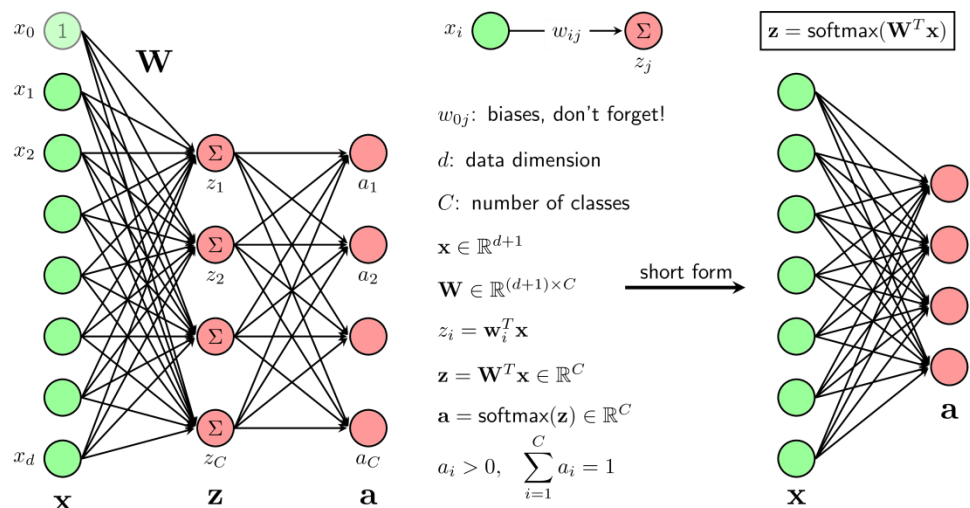
Công thức để tính vector đầu ra của Neuron Network như sau:

$$\mathbf{a} = f(W^T \mathbf{x})$$

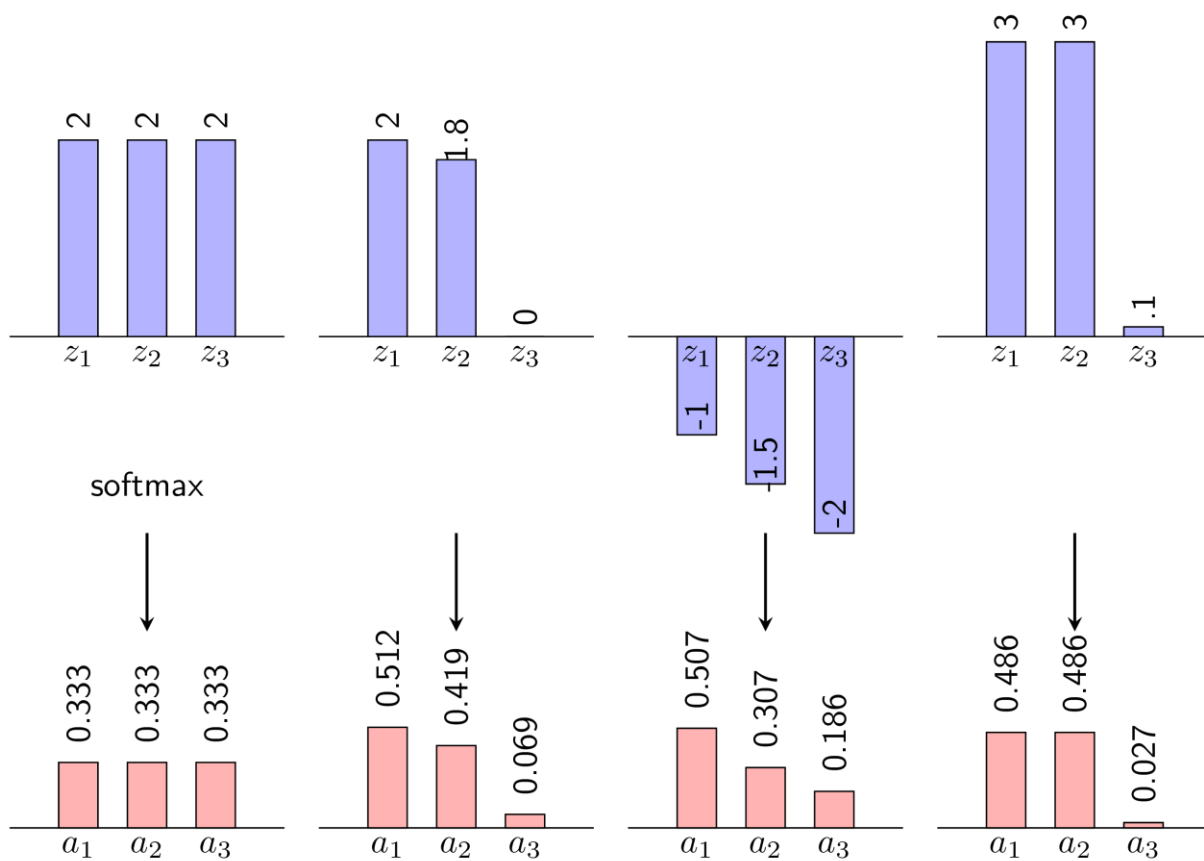
Ta cần có một mô hình xác suất sao cho với mỗi điểm dữ liệu \mathbf{x} thì mỗi a_i sẽ thể hiện xác suất rơi vào class i . Vậy nên ta có điều kiện $0 < a_i < 1$ và $\sum_i a_i = 1$. Cũng rất tự nhiên khi xác suất rơi vào class i của một điểm \mathbf{x} phải tỷ lệ với z_i , nghĩa là z_i càng lớn thì a_i cũng phải lớn theo. Để thỏa mãn các điều kiện đó, ta có thể sử dụng hàm $\exp(x)$ làm activation function theo cách sau:

$$a_i = \frac{\exp(\mathbf{w}_i^T \mathbf{x})}{\sum_{j=1}^C \exp(\mathbf{w}_j^T \mathbf{x})}, \quad \forall i = 1, 2, \dots, C$$

Hàm số này có đạo hàm tại mọi \mathbf{x} và thỏa mãn các điều kiện nêu trên.



Hình II.6.a.2. Mô hình Neuron Network của Softmax Regression
(Nguồn: <https://machinelearningcoban.com/2017/02/17/softmax>)



Hình II.6.a.3. Một số ví dụ về đầu ra của hàm Softmax
(Nguồn: <https://machinelearningcoban.com/2017/02/17/softmax>)

Khi z_i quá lớn, việc tính toán $\exp(z_i)$ có thể gây tràn số trong máy tính, ảnh hưởng đến kết quả tính toán. Ta có thể giới hạn độ lớn của số mũ bằng cách sử dụng hàm softmax sau:

$$a_i = \frac{\exp(z_i - m)}{\sum_{j=1}^C \exp(z_j - m)} \quad , \quad \forall i = 1, 2, 3, \dots, C$$

với m là một hằng số bất kì.

Trong thực nghiệm, người ta thường chọn $m = \max_i z_i$.

b. Tối ưu loss function

Loss function là hàm số thể hiện sự sai lệch giữa đầu ra tính toán được với đầu ra thực tế.

Với thuật toán softmax, mỗi output sẽ là một vector có đúng một phần tử bằng 1 và các phần tử khác bằng 0, phần tử 1 thể hiện xác suất của input rơi vào class đó chính bằng 1. Điều này là chắc chắn vì mỗi phần tử chỉ có một class, nên không có chuyện nó có xác suất thuộc class khác.

Với mỗi **vector cột** đầu vào \mathbf{x} và **ma trận** trọng số W , ta sẽ có **vector cột** đầu ra \mathbf{a} tương ứng có dạng one-hot coding theo công thức:

$$\mathbf{a} = \text{softmax}(W^T \mathbf{x})$$

Với **vector cột** đầu ra thực tế là \mathbf{y} , ta có thể sử dụng loss function sau để đánh giá độ sai lệch:

$$L(W) = \sum_{i=1}^N \|\mathbf{a}_i - \mathbf{y}_i\|^2$$

Nhưng các đại lượng cần đánh giá là các phân bố xác suất, nên ta có một lựa chọn tốt hơn, đó là sử dụng đại lượng *Cross Entropy*.

Với 2 phân bố p và q thì cross entropy được định nghĩa như sau:

$$H(p, q) = E_p[-\ln q]$$

Nếu p và q là 2 phân bố rời rạc thì ta có:

$$H(p, q) = - \sum_{i=1}^C p_i \ln q_i$$

Dựa vào đại lượng này ta xây dựng nên loss function của Softmax Regression đối với một điểm dữ liệu là:

$$L(W; \mathbf{x}_i, \mathbf{y}_i) = - \sum_{j=1}^C y_{ji} \ln a_{ji} \quad , \quad \forall i = \overline{1, N}$$

với N là tổng số điểm dùng để training, W là ma trận trọng số, a_{ji} và y_{ji} lần lượt là đầu ra tính toán và đầu ra thực tế thứ j của điểm thứ i .

Đặt $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ là ma trận các điểm đầu vào.

Đặt $Y = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N\}$ là ma trận các đầu ra thực tế tương ứng.

Kết hợp tất cả các cặp dữ liệu ta có:

$$L(W; X, Y) = - \sum_{i=1}^N \sum_{j=1}^C y_{ji} \ln a_{ji}$$

Tối ưu loss function đối với từng điểm dữ liệu:

$$\begin{aligned} L(W; \mathbf{x}_i, \mathbf{y}_i) &= - \sum_{j=1}^C y_{ji} \ln a_{ji} \\ &= - \sum_{j=1}^C y_{ji} \ln \frac{\exp(\mathbf{w}_j^T \mathbf{x}_i)}{\sum_{k=1}^C \exp(\mathbf{w}_k^T \mathbf{x}_i)} \\ &= - \sum_{j=1}^C \left(y_{ji} \mathbf{w}_j^T \mathbf{x}_i - y_{ji} \ln \left(\sum_{k=1}^C \exp(\mathbf{w}_k^T \mathbf{x}_i) \right) \right) \\ &= - \sum_{j=1}^C y_{ji} \mathbf{w}_j^T \mathbf{x}_i + \ln \left(\sum_{k=1}^C \exp(\mathbf{w}_k^T \mathbf{x}_i) \right) \\ &\quad (\sum_{j=1}^C y_{ji} = 1 \text{ do là tổng các xác suất}) \end{aligned}$$

Ta tính đạo hàm của loss function:

$$\frac{\partial L_i(W)}{\partial W} = \left\{ \frac{\partial L_i(W)}{\partial \mathbf{w}_1}, \frac{\partial L_i(W)}{\partial \mathbf{w}_2}, \frac{\partial L_i(W)}{\partial \mathbf{w}_3}, \dots, \frac{\partial L_i(W)}{\partial \mathbf{w}_C} \right\}$$

với đạo hàm của từng cột là:

$$\begin{aligned} \frac{\partial L_i(W)}{\partial \mathbf{w}_j} &= -y_{ji} \mathbf{x}_i + \frac{\exp(\mathbf{w}_j^T \mathbf{x}_i)}{\sum_{k=1}^C \exp(\mathbf{w}_k^T \mathbf{x}_i)} \mathbf{x}_i \\ &= -y_{ji} \mathbf{x}_i + a_{ji} \mathbf{x}_i = \mathbf{x}_i (a_{ji} - y_{ji}) \end{aligned}$$

Đặt $e_{ji} = a_{ji} - y_{ji}$ là sai số đối với từng đầu ra, $\mathbf{e}_i = \{e_{1i}, e_{2i}, \dots, e_{Ci}\}^T$ là ma trận sai số đối với điểm đầu vào \mathbf{x}_i , $E = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_N\}$ là ma trận sai số đối với toàn bộ điểm dữ liệu.

Suy ra:

$$\frac{\partial L_i(W)}{\partial W} = \mathbf{x}_i \times \{e_{1i}, e_{2i}, \dots, e_{Ci}\} = \mathbf{x}_i \mathbf{e}_i^T$$

Từ đó suy ra đối với toàn bộ tập dữ liệu ta có:

$$\frac{\partial L(W)}{\partial W} = XE^T$$

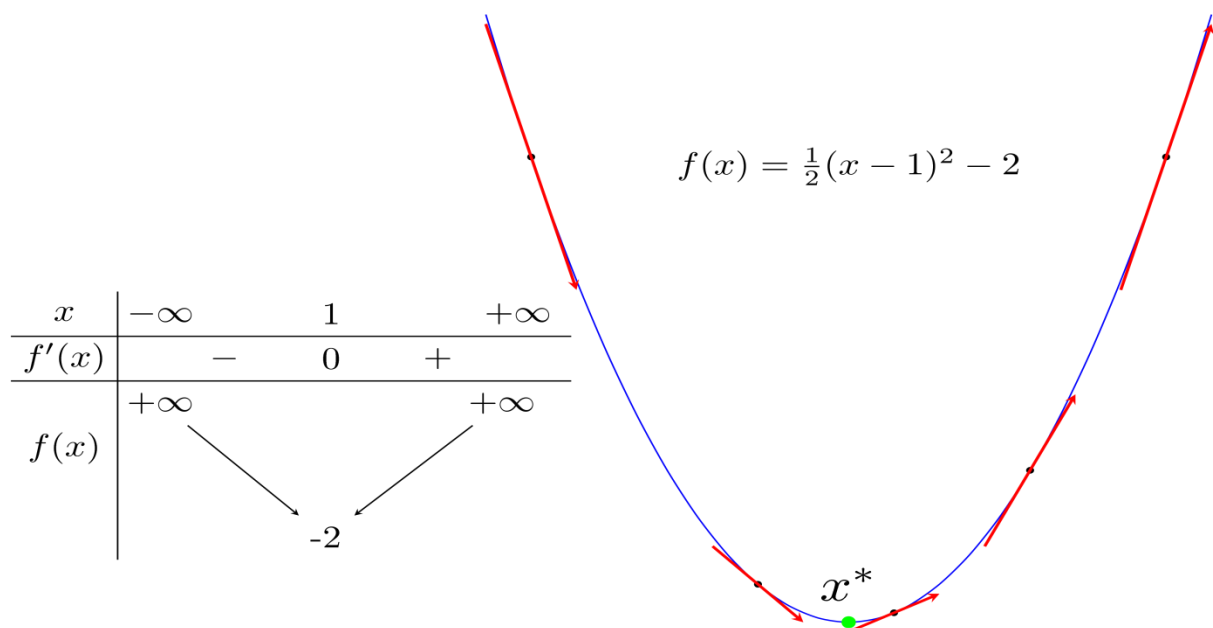
Ta cần tìm điểm cực tiểu W^* của đạo hàm này, đó chính là ma trận trọng số của mô hình.

2. Gradient Descent

Trong Machine Learning nói riêng và Toán Tối Ưu nói chung, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc đôi khi là lớn nhất) của một hàm số nào đó, với mô hình Neuron Network trên ta phải tìm cực tiểu của hàm sai số giữa đầu ra tính được và đầu ra thực tế. Cách thường được dùng để tìm cực trị một hàm số $f(x)$ là thông qua tìm nghiệm của đạo hàm. Việc tính đạo hàm $f'(x)$ của một hàm số là có thể làm được, nhưng tìm nghiệm của nó thông qua giải phương trình $f'(x) = 0$ thì thông thường khó hơn rất nhiều. Vì vậy ta cần tìm một phương pháp dễ dàng hơn để tìm điểm cực tiểu.

Giả sử ta có hàm số có đạo hàm ở mọi điểm đang xét $f(x)$, thì cực tiểu của hàm số chính là một nghiệm của phương trình $f'(x) = 0$.

Ví dụ đối với việc tìm cực tiểu của hàm số một ẩn:



Đạo hàm $f'(x)$ chính là hệ số góc của tiếp tuyến tại x , biểu hiện độ dốc của đồ thị, $f'(x)$ càng lớn thì độ dốc của đồ thị càng lớn, nghĩa là x^* càng xa so với điểm đang xét. Ngoài việc ta có $f'(x^*) = 0$ thì ta còn thấy trong lân cận của x^* , đạo hàm của các điểm bên trái x^* là không dương, các điểm bên phải x^* là không âm.

Giả sử nếu ta chỉ biết cách tính $f'(x)$ mà không giải được phương trình $f'(x) = 0$ thì dựa vào nhận xét trên ta có:

Với một điểm x_0 bất kì có đạo hàm $f'(x_0)$:

- Nếu $f'(x_0) > 0 \Rightarrow x^* < x_0$
- Nếu $f'(x_0) < 0 \Rightarrow x^* > x_0$

Sau một số lần nhất định xác định vị trí của x^* so với các điểm x_0 khác nhau, ta có thể thu hẹp khoảng giá trị của x^* và tìm được giá trị gần đúng của điểm cực tiểu, điểm gần đúng đó sẽ thoả mãn:

$$f'(\hat{x}^*) \approx 0$$

Việc tính xấp xỉ với số lượng đầu vào rất lớn nhưng hữu hạn số phép tính thì các máy tính hiện đại có thể tính toán được và trong khoảng thời gian rất nhanh, giúp ta có một phương pháp đủ đơn giản và chính xác để tìm được điểm cực tiểu. Đây chính là cơ sở trong việc tìm được các tham số của Neuron Network.

Giả sử x_t là điểm ta tìm được sau t vòng lặp, ta cần điểm tiếp theo x_{t+1} gần với x^* hơn. Khi đó ta cần di chuyển x_t theo hướng ngược dấu của đạo hàm. Đạo hàm càng lớn thì ta dịch chuyển càng nhiều, nghĩa là dịch chuyển một lượng tỉ lệ thuận với đạo hàm. Thuật toán đơn giản nhất để thực hiện việc đó như sau:

$$x_{t+1} = x_t - \eta f'(x_t)$$

Với η là một số dương được gọi là *learning rate*, thể hiện tốc độ của việc dịch chuyển, dấu trừ thể hiện việc ta đi ngược dấu với đạo hàm đến "đáy" của đồ thị (descent – dốc xuống). Nếu η quá nhỏ thì việc hội tụ sẽ chậm, nếu η quá lớn thì có khả năng ta sẽ luôn đi qua điểm cực tiểu.

Quay lại bài toán ở phần II.1, ta có thể tìm điểm cực tiểu như sau:

Loss function:

$$L(W; X, Y) = - \sum_{i=1}^N \sum_{j=1}^C y_{ji} \ln a_{ji}$$

Đạo hàm của loss function:

$$\nabla_W L(W; X, Y) = XE^T$$

Áp dụng cách tìm cực tiểu ở trên, ta có cách cập nhật ma trận trọng số:

$$W_{t+1} = W_t - \eta \nabla_W L(W; X, Y)$$

với W_0 là ma trận trọng số khởi tạo ban đầu, thường thì sẽ chọn các giá trị ngẫu nhiên gần 0.

Đối với các bài toán có số lượng đầu vào và số lượng (input, label) rất lớn, việc tính ma trận như trên sẽ rất phức tạp và gây ra khó khăn cho việc cập nhật ma trận trọng số khi có một tập dữ liệu mới, cho nên một vài biến thể của Gradient Descent sau thường được sử dụng:

a. Stochastic Gradient Descent

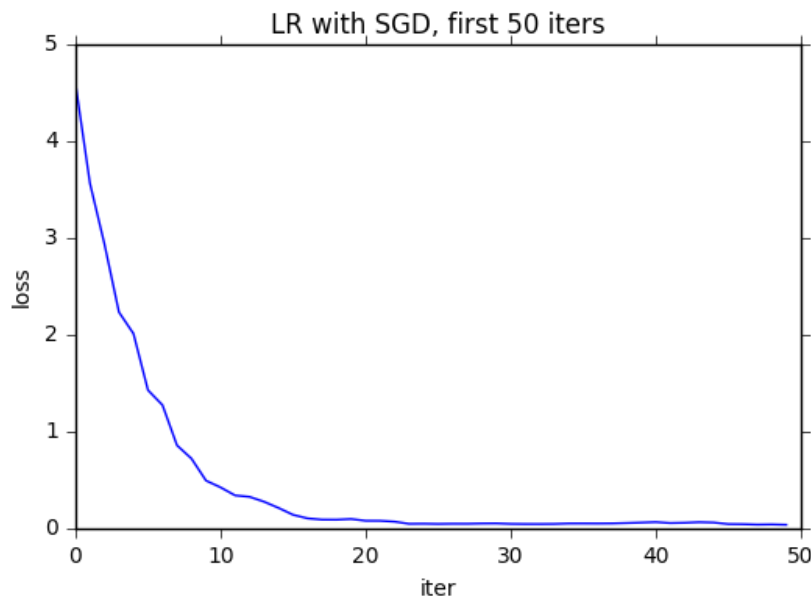
Trong thuật toán này, tại 1 thời điểm, ta chỉ tính đạo hàm của hàm mất mát dựa trên chỉ một điểm dữ liệu x_i rồi cập nhật W dựa trên đạo hàm này. Việc này được thực hiện với từng điểm trên toàn bộ dữ liệu, sau đó lặp lại quá trình trên. Thuật toán rất đơn giản này trên thực tế lại làm việc rất hiệu quả.

Mỗi lần duyệt một lượt qua tất cả các điểm trên toàn bộ dữ liệu được gọi là một epoch. Với GD thông thường thì mỗi epoch ứng với 1 lần cập nhật θ , với SGD thì mỗi epoch ứng với N lần cập nhật W với N là số điểm dữ liệu. Nhìn vào một mặt, việc cập nhật từng điểm một như thế này có thể làm giảm đi tốc độ thực hiện 1 epoch. Nhưng nhìn vào một mặt khác, SGD chỉ yêu cầu một lượng epoch rất nhỏ (thường là 10 cho lần đầu tiên, sau đó khi có dữ liệu mới thì chỉ cần chạy dưới một epoch là đã có nghiệm tốt). Vì vậy SGD phù hợp với các bài toán có lượng cơ sở dữ liệu lớn (chủ yếu là Deep Learning mà chúng ta sẽ thấy trong phần sau của blog) và các bài toán yêu cầu mô hình thay đổi liên tục, tức online learning.

Quy tắc cập nhật bằng SGD được biểu diễn toán học như sau:

$$W_{t+1} = W_t - \eta \nabla_W L(W; \mathbf{x}_i, \mathbf{y}_i)$$

với $\nabla_W L(W; \mathbf{x}_i, \mathbf{y}_i)$ là đạo hàm tính tại một cặp (input, label).



Hình II.2.a. Giá trị của loss function tại 50 vòng lặp đầu tiên
(Nguồn: <https://machinelearningcoban.com/2017/01/16/gradientdescent2>)

b. Mini-batch Gradient Descent

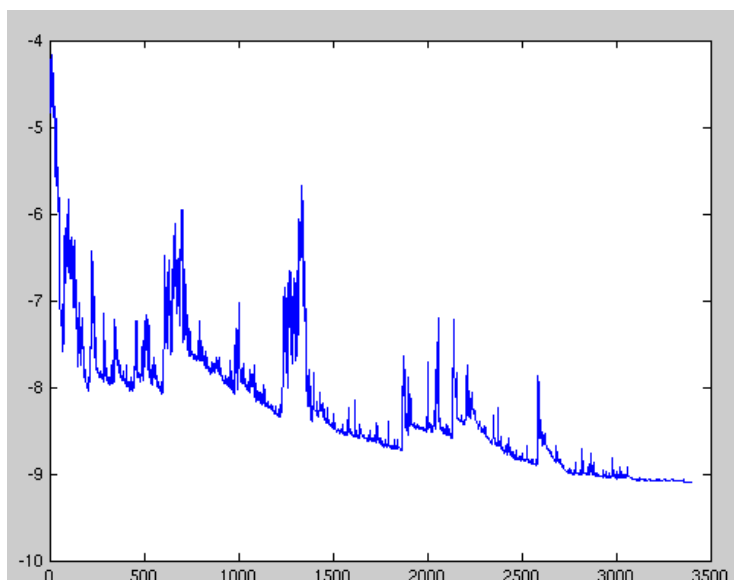
Khác với SGD, mini-batch sử dụng một số lượng n lớn hơn 1 (nhưng vẫn nhỏ hơn tổng số dữ liệu N rất nhiều). Giống với SGD, Mini-batch Gradient Descent bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia toàn bộ dữ liệu thành các mini-batch, mỗi mini-batch có n điểm dữ liệu (trừ mini-batch cuối có thể có ít hơn nếu N không chia hết cho n).

Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật. Công thức có thể viết dưới dạng:

$$W_{t+1} = W_t - \eta \nabla_W L(W; \mathbf{x}_{i:i+n-1}, \mathbf{y}_{i:i+n-1})$$

với $\mathbf{x}_{i:i+n-1}$ được hiểu là dữ liệu từ thứ i tới thứ $i+n-1$, $\mathbf{y}_{i:i+n-1}$ là label tương ứng.

Dữ liệu này sau mỗi epoch là khác nhau vì chúng cần được xáo trộn.



Hình II.2.b. Giá trị Loss function của một bài toán phức tạp biến động liên tục nhưng vẫn có xu hướng giảm dần và hội tụ

(Nguồn: https://en.wikipedia.org/wiki/Stochastic_gradient_descent)

c. Điểm dừng của thuật toán

Có một điểm cũng quan trọng đối với việc lập trình trên máy tính: khi nào thì chúng ta biết thuật toán đã hội tụ và dừng lại?

Trong thực nghiệm, có một vài phương pháp như dưới đây:

1. Giới hạn số vòng lặp: đây là phương pháp phổ biến nhất và cũng để đảm bảo rằng chương trình chạy không quá lâu. Tuy nhiên, một nhược điểm của cách làm này là có thể thuật toán dừng lại trước khi đủ gần với nghiệm.
2. So sánh gradient của nghiệm tại hai lần cập nhật liên tiếp, khi nào giá trị này đủ nhỏ thì dừng lại. Phương pháp này cũng có một nhược điểm lớn là việc tính đạo hàm đôi khi trở nên quá phức tạp (ví dụ như khi có quá nhiều dữ liệu), nếu áp dụng phương pháp này thì coi như ta không được lợi khi sử dụng SGD và mini-batch GD.
3. So sánh giá trị của hàm mất mát của nghiệm tại hai lần cập nhật liên tiếp, khi nào giá trị này đủ nhỏ thì dừng lại. Nhược điểm của phương pháp này là nếu tại một thời điểm, đồ thị hàm số có dạng bằng phẳng tại một khu vực nhưng khu vực đó không chứa điểm local minimum (khu vực này thường được gọi là saddle points), thuật toán cũng dừng lại trước khi đạt giá trị mong muốn.

4. Trong SGD và mini-batch GD, cách thường dùng là so sánh nghiệm sau một vài lần cập nhật. Nếu 2 nghiệm liên tiếp không thay đổi nhiều thì ta có thể dừng lại.

Phần III của báo cáo này sẽ giới thiệu cách áp dụng một mô hình 1-layer Neuron Network sử dụng thuật toán Softmax Regression vào cấu trúc FPGA để nhận diện chữ số viết tay.

III. Sơ đồ nhận dạng kí tự số viết tay sử dụng FPGA

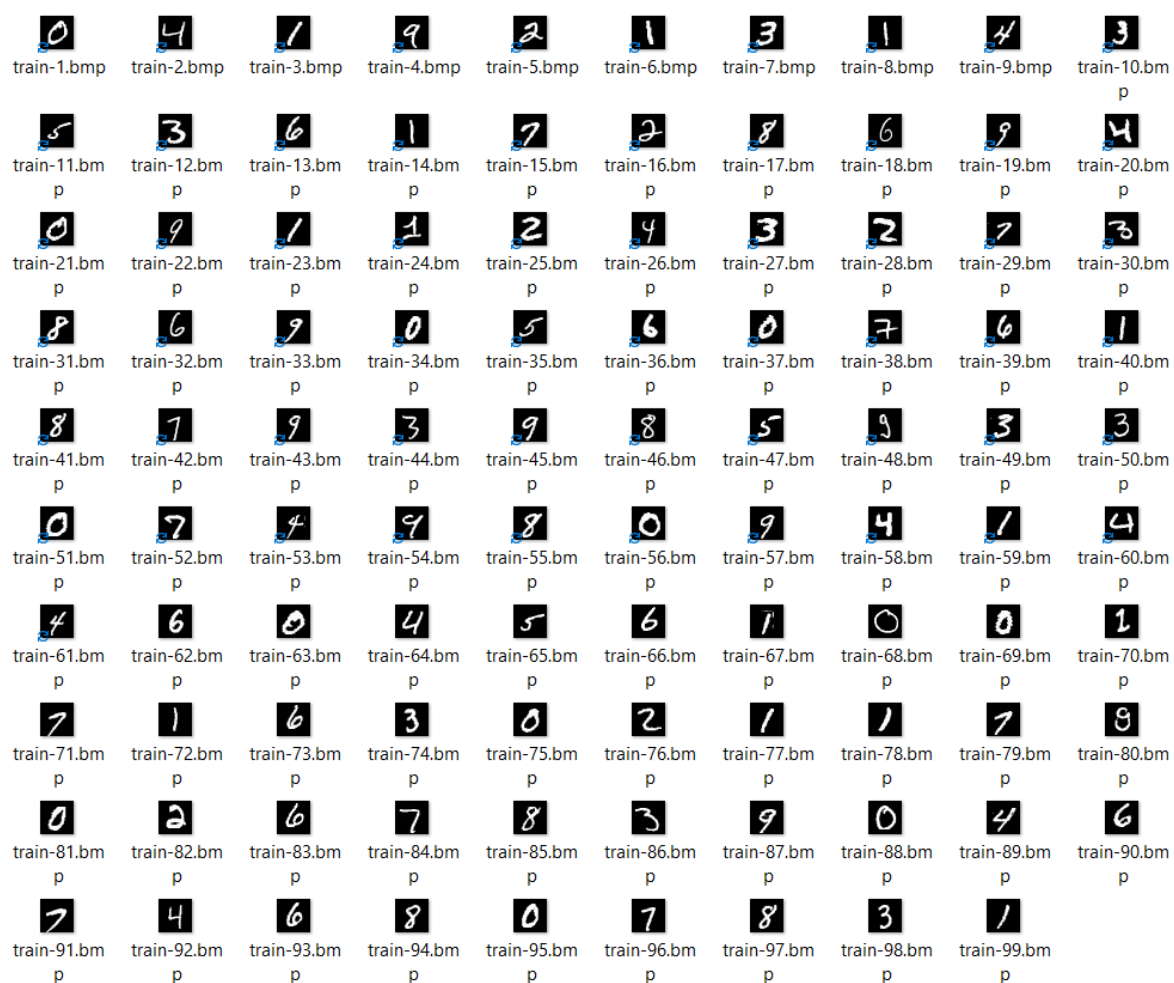
Trước hết ta thấy, FPGA là mạch logic lập trình được, nó có khả năng mô phỏng các mạch logic và thực hiện nhiều tiến trình logic song song, mang lại tốc độ tính toán rất lớn trong một chu kỳ CLK, nhưng số lượng phép toán hỗ trợ khá hạn chế nếu không có kiến thức đủ tốt về việc thiết kế các phép toán phức tạp dựa trên các phép toán logic. Vì vậy để có thể áp dụng toàn bộ thuật toán training đã nêu ở các phần trước không phải là điều đơn giản. Cũng đi từ thực tiễn, một hệ thống nhận diện không phải lúc nào cũng ở trạng thái training, việc training chỉ tốn thời gian và sức mạnh tính toán vào lúc đầu để có thể tính ra ma trận trọng số W thích hợp. Cho nên ta có thể sử dụng một công cụ tính toán khác rất phù hợp cho việc này, đó là các CPU trong các máy vi tính hiện đại. Các vi xử lý hiện nay có tốc độ rất cao và độ chính xác đáng kinh ngạc, cho nên ta sẽ sử dụng nó để thực hiện việc khó khăn nhất là tìm ra các tham số phù hợp, sau đó ta sẽ triển khai các tham số đó lên FPGA để có được tốc độ nhận diện nhanh nhất.

Báo cáo này sẽ đề cập đến việc áp dụng mô hình 1-layer Neuron Network với thuật toán Softmax Regression, sử dụng C++ để lập trình cho vi xử lý, từ đó tính ra các trọng số của Neuron Network rồi nạp vào FPGA.

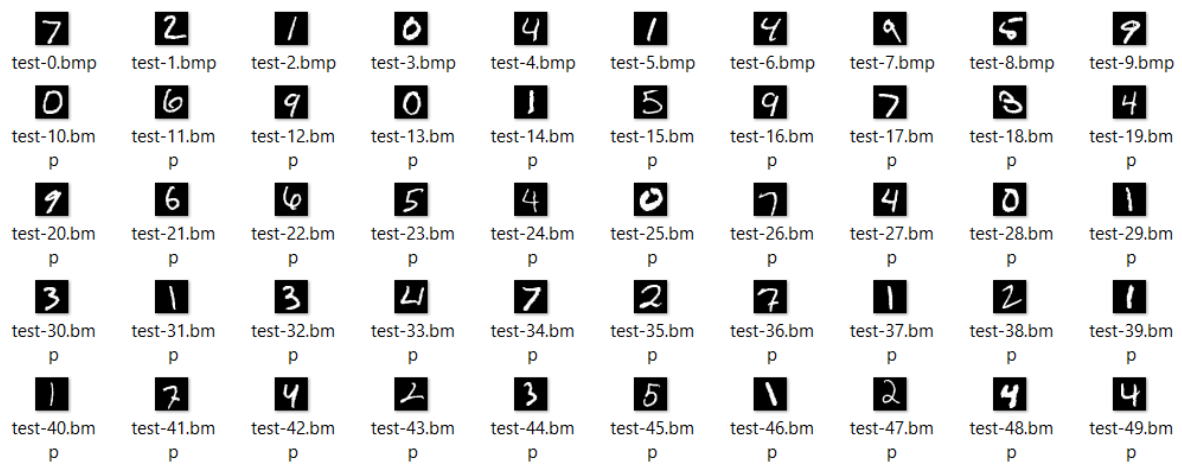
1. Cơ sở dữ liệu

Phần nghiên cứu này sử dụng *MNIST DATABASE of handwritten digits* làm cơ sở dữ liệu cho kí tự chữ số viết tay, gồm 60,000 bức ảnh 28x28 pixels 8bit-depth giúp cho việc training và 10,000 bức ảnh 28x28 pixels 8bit-depth giúp cho việc test cùng với label của từng bức ảnh để giúp đánh giá độ chính xác của hệ thống.

Nguồn: <http://yann.lecun.com/exdb/mnist/>



Hình III.1.1. 100 mẫu ảnh đầu tiên trong bộ train-images



Hình III.1.2. 50 mẫu ảnh đầu tiên trong bộ t10k-test

2. Cấu trúc chương trình C++

- Định nghĩa class Matrix và các thuật toán ma trận:

```
+ class Matrix { ... };

+ Matrix operator+(const Matrix &matrix_1, const Matrix &matrix_2) { ... }

+ Matrix operator-(const Matrix &matrix_1, const Matrix &matrix_2) { ... }

+ Matrix operator*(const double &multiplier, const Matrix &source_matrix) { ... }

+ Matrix operator*(const Matrix &source_matrix, const double &multiplier) { ... }

+ Matrix operator*(const Matrix &matrix_1, const Matrix &matrix_2) { ... }

+ Matrix Matrix::T() { ... }
```

- Định nghĩa class Softmax và các function của thuật toán Softmax Regression:

```
+ class Softmax { ... };

+ void Softmax::init(unsigned numbers_of_inputs, unsigned max_output, double eta) { ... }

+ void Softmax::set(Matrix &x, unsigned &y) { ... }

+ Matrix Softmax::softmax(Matrix z) { ... }

+ Matrix Softmax::grad(Matrix w)
{
    return x * (softmax(w.T()*x) - y).T();
}

+ void Softmax::update_w()
{
    Matrix old_v;
    Matrix old_w;
    old_w = w;
    v = eta * grad(w);
    w = old_w - v;
}

+ double Softmax::get_output(Matrix &x) { ... }

+ Matrix Softmax::get_output_matrix(Matrix &x) { ... }
```

- Các function đọc ghi file hỗ trợ việc lấy data từ database, thực hiện train và test, ghi lại các trọng số, làm cơ sở để nạp vào FPGA.

```

+int train() { ... }

+int test() { ... }

+int getoutput() { ... }

+void conv_b() { ... }

```

Sau khi chạy chương trình với 10,000 ảnh đầu tiên của bộ train-image, ta nhận được ma trận trọng số có 785 dòng và 10 cột.

```

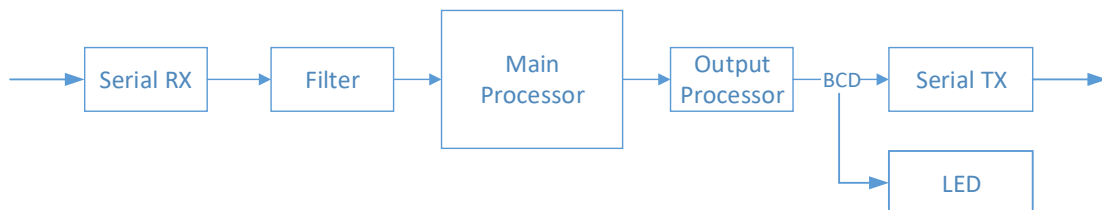
-3.43683 0.790635 1.25784 -1.619 2.62596 20.5969 -1.3589 8.30415 -20.2226 -6.83807
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01

```

Hình III.2.1. 10 dòng đầu tiên của ma trận trọng số

3. Cấu trúc các khối trên FPGA

Ta sẽ xây dựng một sơ đồ tổng quát như sau:



Hình III.3.1. Sơ đồ khối tổng quát của các khối trên FPGA

Tác dụng của từng khối:

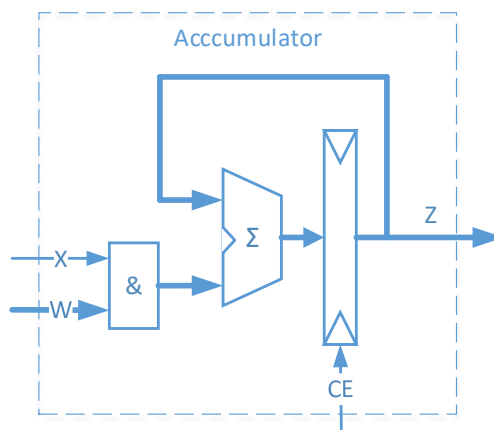
- Serial RX, TX: khối giao tiếp với các thiết bị bên ngoài, cụ thể ở nghiên cứu này sử dụng đường truyền RS232 vì sự đơn giản của nó
- Filter: Bộ lọc đầu vào giúp nén các pixel lại ở mức đủ để xử lý, giúp tăng tốc độ xử lý
- Main Processor: Nơi thực hiện các phép toán feedforward và backpropagation ở mức đơn giản hoá
- Output Processor: Chuyển đổi ma trận đầu ra thành giá trị nhị phân của kết quả để hiển thị ra LED và gửi kết quả ra bên ngoài

Căn bản của việc xây dựng mô hình 1-layer Neuron Network trên FPGA là ta phải thiết kế để FPGA thực hiện được phép nhân ma trận, cụ thể là $\mathbf{z}_{10 \times 1} = (\mathbf{W}_{785 \times 10})^T \mathbf{x}_{785 \times 1}$.

Ta có:

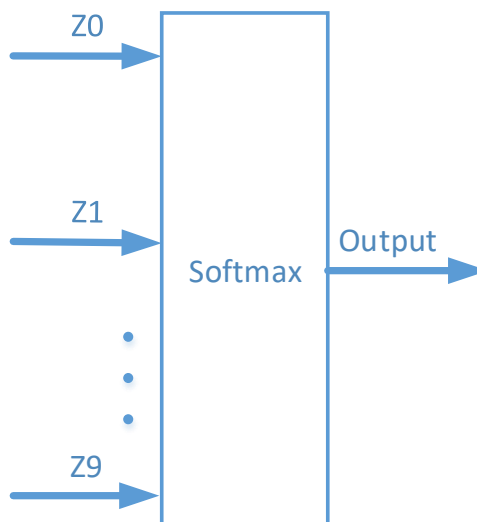
$$z_j = \sum_{i=1}^{785} w_{ij}x_i \quad , \quad \forall j = \overline{1,10}$$

Có nhiều cách để thực hiện phép tính tổng này. Ta có thể thiết kế một bộ cộng để cộng 785 phần tử một cách song song, đồng thời cũng cần 785 bộ nhân đi kèm cho mỗi đầu ra. Vậy tổng cộng đối với 10 đầu ra ta sẽ cần 10 bộ cộng 785 đầu vào và 7850 bộ nhân. Trong nghiên cứu này, để có được sự đơn giản hoá nhưng vẫn đảm bảo tốc độ, bộ cộng tích lũy sẽ được sử dụng. Như vậy đối với mỗi đầu ra ta sẽ sử dụng 1 bộ cộng tích lũy kèm với 1 bộ nhân.



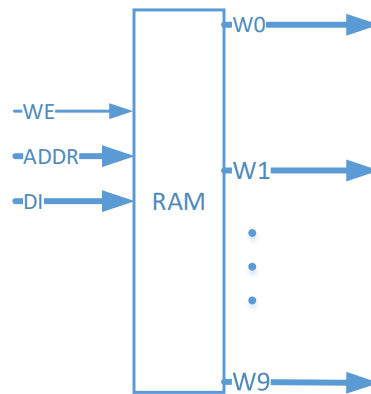
Hình III.3.1. Bộ cộng tích lũy

Để có được đầu ra ta cần thiết kế một bộ tính hàm softmax và chuyển đổi dạng one-hot coding và biến đổi one-hot coding thành BCD.



Hình III.3.2. Hàm Softmax và chuyển đổi đầu ra

Để có thể lưu trữ ma trận trọng số và phục vụ việc tính toán song song cho 10 đầu ra, ta sẽ sử dụng 10 BRAMs, mỗi BRAM sẽ lưu một cột của ma trận trọng số. Ma trận trọng số sẽ được nạp trực tiếp vào mạch thông qua lập trình bằng VHDL.



Hình III.3.3. Cấu trúc của phần bộ nhớ

Ngoài ra ta cũng cần khối giao tiếp để gửi nhận dữ liệu với bên ngoài và các khối điều khiển cho toàn bộ hệ thống. Sơ đồ khối chi tiết sẽ được nêu ra ở phần IV sau khi thực hiện tối ưu thiết kế. Trong thiết kế này, để đơn giản hoá việc giao tiếp thì giao thức được sử dụng UART thông qua cổng RS232.

IV. Tối ưu hoá thiết kế FPGA

Để FPGA có thực hiện các chức năng trên, các phép toán cũng cần được đơn giản hoá để có thể dễ dàng hơn trong việc thiết kế.

1. Nén đầu vào

Do đặc thù của nhận dạng ký tự không phải là lấy ra màu sắc hay độ sáng tối của nét chữ, tham số ta cần lấy được từ một điểm ảnh chính là nó có phải dữ liệu hay không và vị trí của nó, cho nên ta có thể nén một pixel từ 8 bit xuống 1 bit, tùy theo độ rõ nét của pixel mà quyết định nó là thông nền hay là dữ liệu.

2. Hàm softmax

Hàm softmax gồm rất nhiều hàm $\exp(x)$, việc hiện thực hoá hàm này không hề đơn giản. Nhưng do ta đã thực hiện việc learning bằng vi xử lý của máy vi tính nên vị trí phần tử cực đại của vector $\mathbf{z} = \mathbf{w}^T \mathbf{x}$ chính là vị trí của phần tử $y_i = 1$. Vì vậy ta có thể dùng một bộ so sánh các phần tử của \mathbf{z} và đưa ra vector \mathbf{a} theo dạng one-hot coding tương ứng.

3. Thuật toán cập nhật ma trận W

Công thức cập nhật ma trận trọng số:

$$W_{t+1} = W_t - \eta \mathbf{x}_i (\mathbf{a}_i - \mathbf{y}_i)^T = W_t + \eta \mathbf{x}_i (\mathbf{y}_i - \mathbf{a}_i)^T = W_t + \eta \mathbf{x}_i \mathbf{e}_i^T$$

với \mathbf{y}_i , \mathbf{a}_i là 2 vector được mã hoá theo dạng one-hot coding.

Với định dạng như thế thì nếu có sai khác giữa \mathbf{y}_i , \mathbf{a}_i thì hiệu của chúng sẽ là một vector có một phần tử là 1, một phần tử là -1, còn lại là 0. Ta cũng có thể nhân 2 vế với hằng số $c > 1$ để $c\eta = 1$, đồng nghĩa với việc W được tăng lên c lần. Khi đó:

$$a_i = \frac{\exp(cw_i^T x)}{\sum_j \exp(cw_j^T x)} = \frac{(\exp(w_i^T x))^c}{\sum_j (\exp(w_j^T x))^c}$$

Dễ thấy với $w_i^T x$ lớn nhất thì $(\exp(w_i^T x))^c$ cũng là lớn nhất và a_i cũng là lớn nhất. Cho nên nếu dùng thuật toán so sánh để tìm ra đầu ra thì kết quả thuật toán không thay đổi.

Kết luận: Sau khi đã nạp vào ma trận W, nếu ta nhận được kết quả nhận diện sai, ta có thể cập nhật lại ma trận W theo một thuật toán đơn giản hơn. Thuật toán cập nhật có thể đơn giản hoá thành như sau:

- Nếu $e_j = 1$ thì $\mathbf{w}_j = \mathbf{w}_j + \mathbf{x}$
 - Nếu $e_j = -1$ thì $\mathbf{w}_j = \mathbf{w}_j - \mathbf{x}$
- với \mathbf{w}_j là cột thứ j của ma trận W.

Sau khi tối ưu hoá thiết kế, ta sẽ xây dựng được sơ đồ khối chi tiết của mô hình trên FPGA. Sơ đồ khối chi tiết này được đưa ra ở phần VIII.4 của tài liệu này.

4. Tài nguyên sử dụng

Thiết kế được thử nghiệm trên bo mạch Spartan3E XC3S500E-PQ208-5.

Kết quả sau tổng hợp:

Device utilization summary:

Selected Device : 3s500epq208-5

Number of Slices:	899	out of	4656	19%
Number of Slice Flip Flops:	1313	out of	9312	14%
Number of 4 input LUTs:	1532	out of	9312	16%
Number of IOs:	9			
Number of bonded IOBs:	9	out of	158	5%
Number of BRAMs:	10	out of	20	50%
Number of GCLKs:	1	out of	24	4%

5. Tính toán tốc độ xử lý của FPGA

Thiết kế trên thực hiện 785 phép nhân nối tiếp và 10 phép cộng song song để hoàn thành phép nhân $(W^T)_{10 \times 785} \times \mathbf{x}_{1 \times 785}$ và mất thêm 10 chu kì để so sánh 10 phần tử đầu ra với nhau, tổng cộng mất 795 chu kì CLK để hoàn thành việc nhận dạng.

Các tham số thời gian của thiết kế:

Timing Summary:

Speed Grade: -5

Minimum period: 9.274ns (Maximum Frequency: 107.834MHz)
Minimum input arrival time before clock: 4.877ns
Maximum output required time after clock: 8.931ns
Maximum combinational path delay: No path found

Với tần số CLK cực đại là 100MHz, thời gian thực để nhận diện 1 bức ảnh là:

$$T = 795 \times 10ns = 7950ns \approx 8\mu s$$

V. Kết quả

Với khả năng tính toán song song của FPGA và khả năng làm việc với từng bit dữ liệu, ta có cơ sở để nói rằng với một thuật toán được tối ưu tốt thì tốc độ xử lý của FPGA hoàn toàn có thể nhanh hơn tốc độ xử lý của vi xử lý hiện đại, với đơn vị tính là số chu kỳ CLK.

Đề tài này đã có sản phẩm thử nghiệm và hoạt động tốt với thuật toán Softmax Regression.

Để hỗ trợ việc nhận diện và thống kê, nghiên cứu này sử dụng một phần mềm được viết bằng C# (xem thêm về phần mềm này ở phần VIII.3 của tài liệu này).

Điều kiện thử nghiệm:

- Ma trận trọng số được train bằng 10,000 ảnh đầu tiên của 60,000 ảnh trong bộ train-image

Sơ đồ kiểm tra:



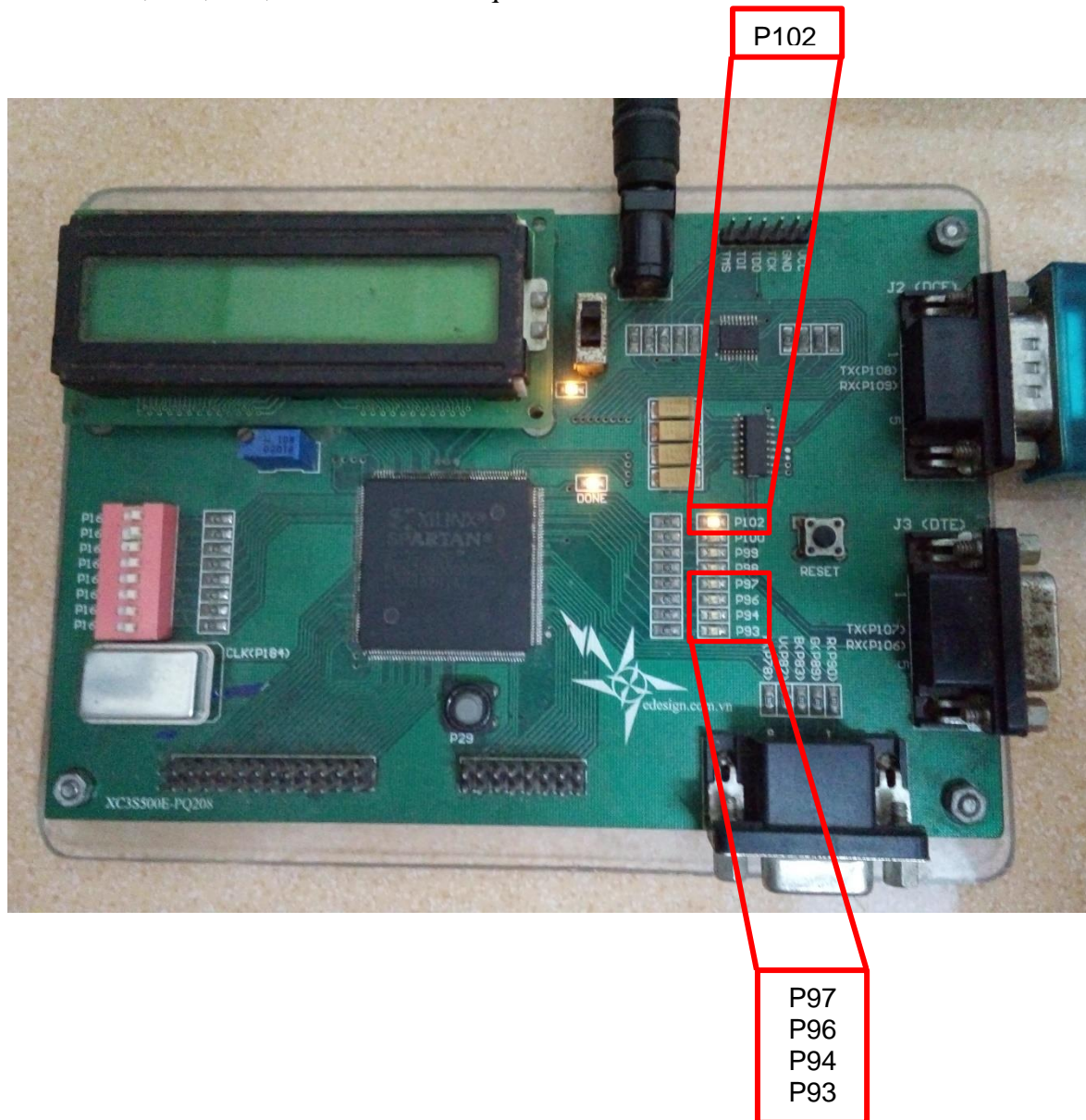
Các thành phần có trong sơ đồ:

- Mạch FPGA Spartan3E
- CPU Intel® Core™ i7-5600U 2.6GHz

- Cáp chuyển đổi USB-RS232

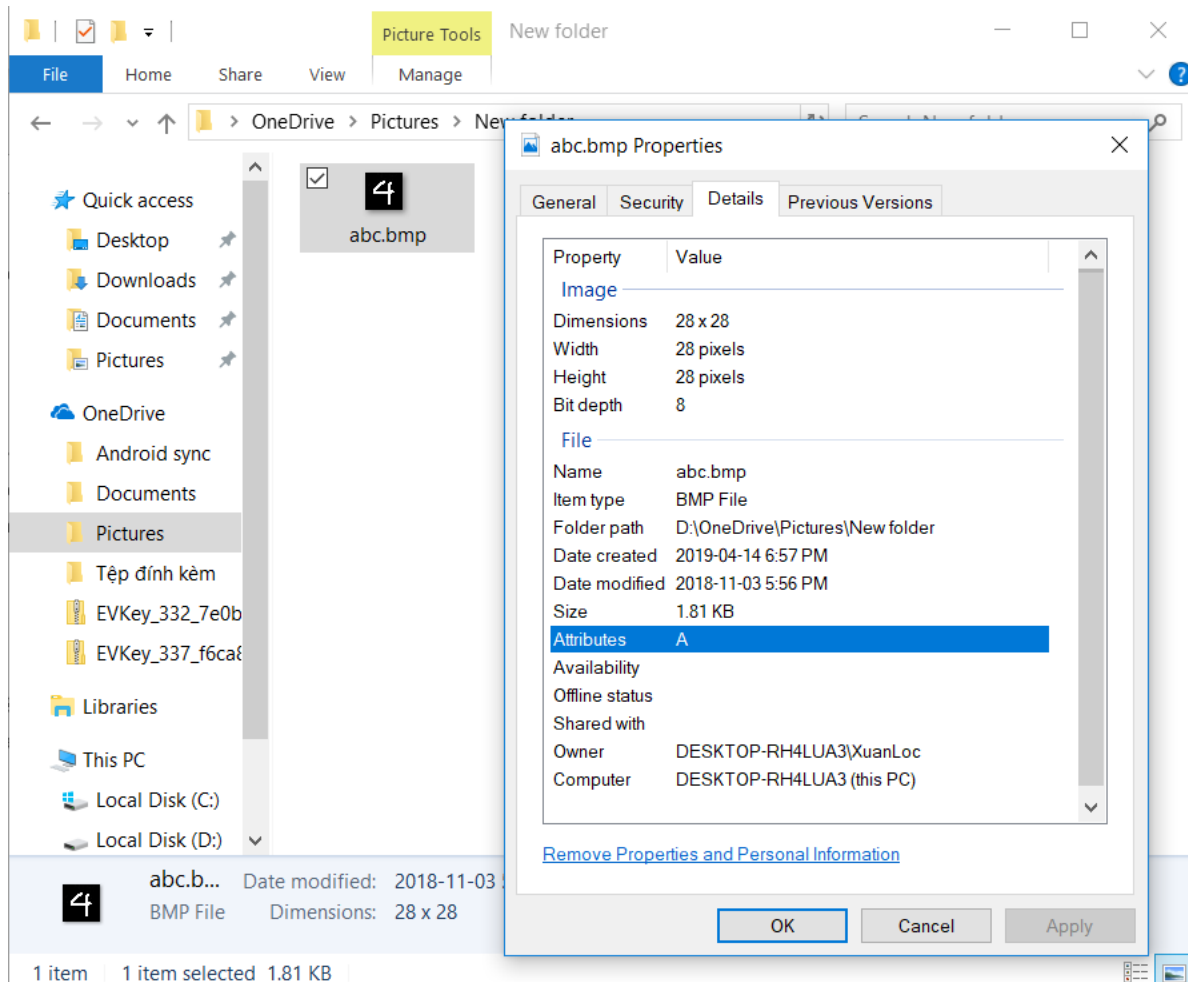
Các đèn báo hiệu trên mạch:

- P102: đèn báo mạch ở chế độ chờ
- P93, P94, P96, P97: đèn báo kết quả theo mã BCD

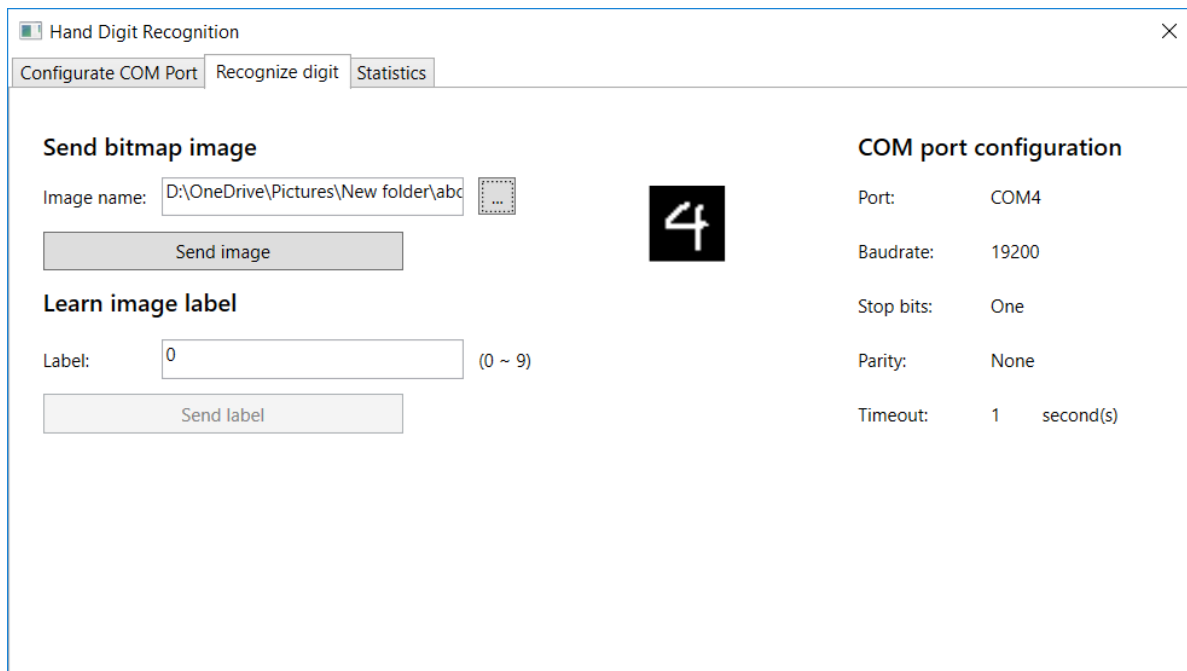


Kiểm tra chức năng nhận diện:

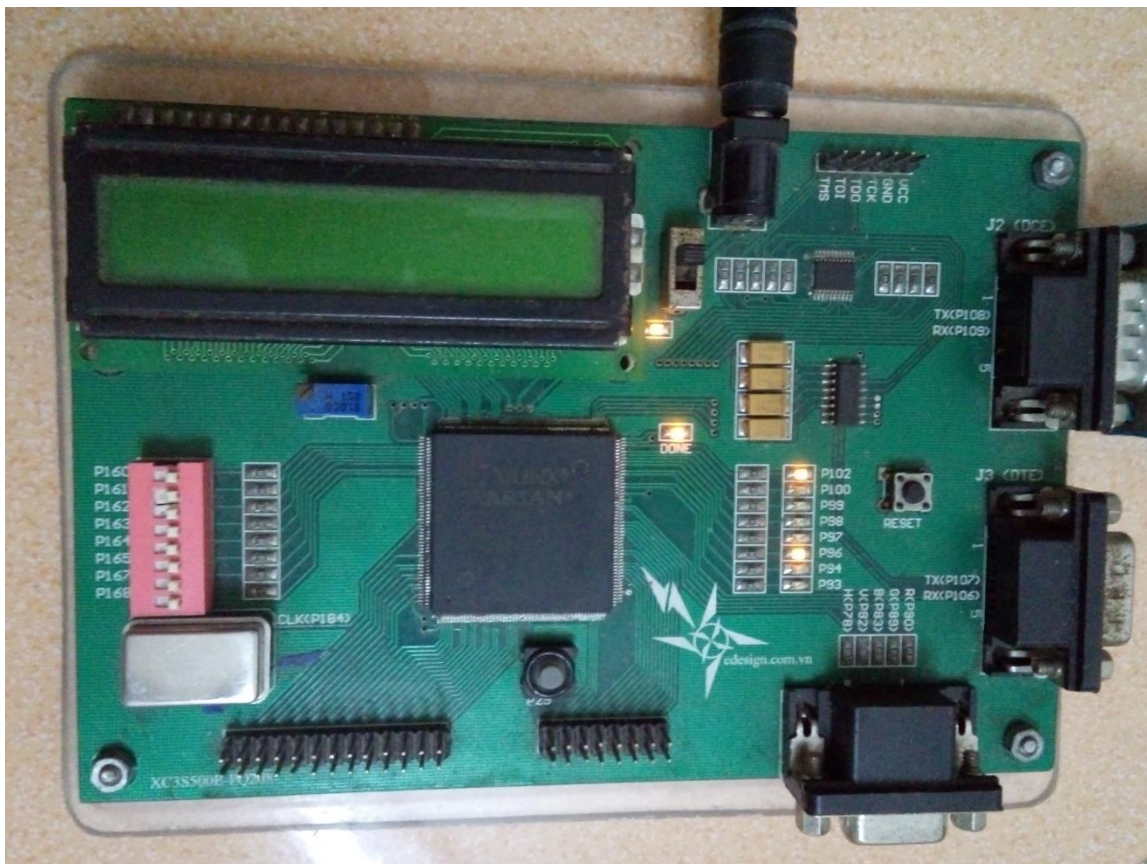
- Sử dụng phần mềm Paint trên Window để tạo ra một ảnh có định dạng .bmp kích cỡ 28x28, bitdepth 8bit, nền màu đen, phân kí tự màu trắng:



- Sử dụng phần mềm riêng để gửi ảnh xuống mạch thông qua cổng COM:
(xem thêm phần VIII.3 của tài liệu này)



- Kết quả nhận được trên mạch:



Đèn báo là 0100b = 4, kết quả đúng.

Thử nghiệm khả năng Online-learning:

- Ảnh gửi đi:

Hand Digit Recognition

Configure COM PortRecognize digitStatistics

Send bitmap image

Image name: D:\OneDrive\Pictures\New folder\traiv

Send image

Learn image label

Label: 0 (0 ~ 9)

Send label

COM port configuration


Port: COM4

Baudrate: 19200

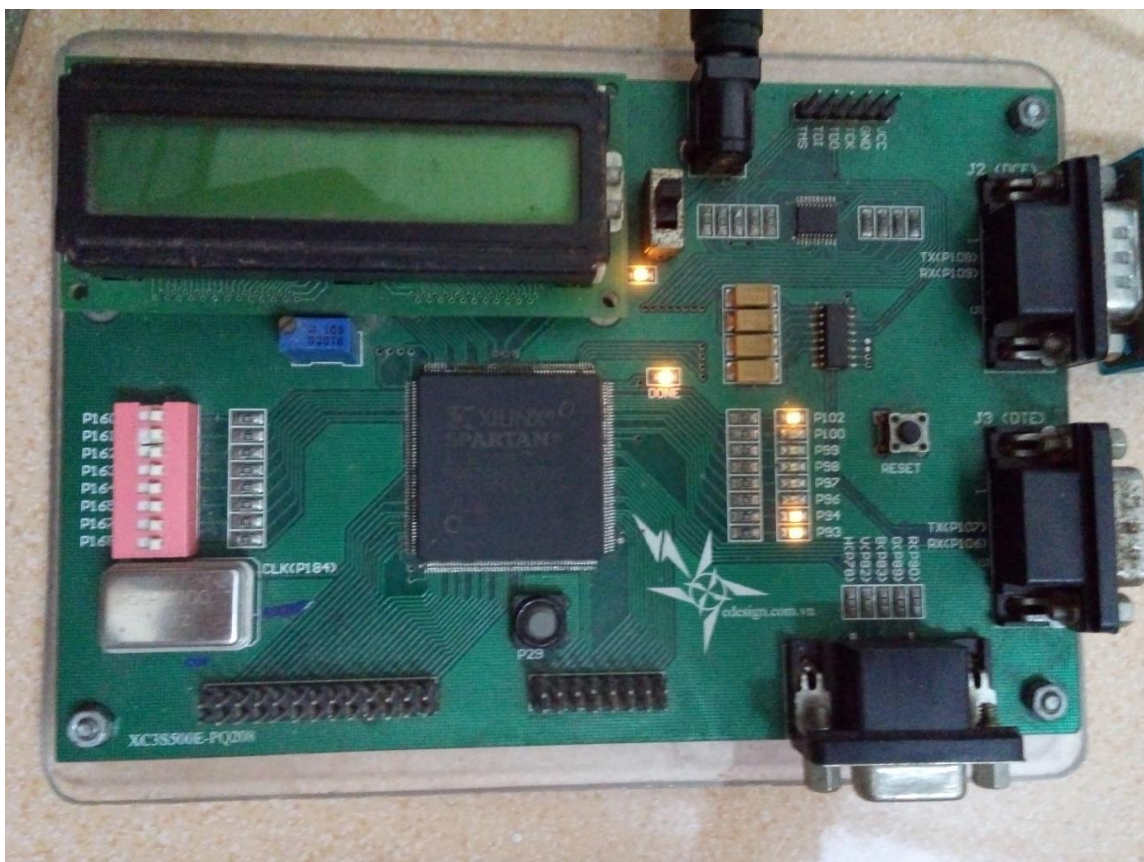
Stop bits: One

Parity: None

Timeout: 1 second(s)

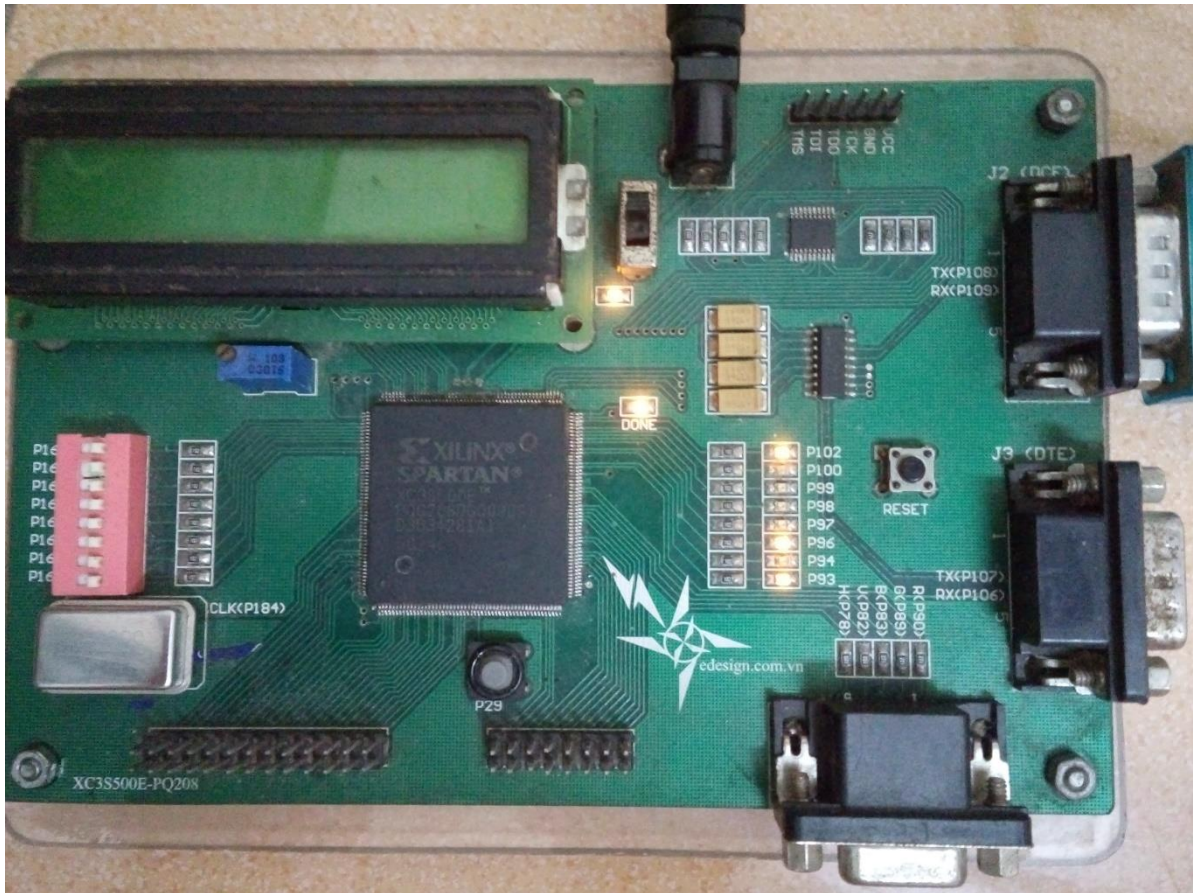


- Kết quả trên mạch:



Đèn báo là 0011b = 3, kết quả sai.

- Sau online-learning:



Đèn báo là 0101b = 5, kết quả đúng.

Kiểm thử với FPGA:

- Test với 1000 ảnh đầu tiên của bộ t10k-test
- Kết quả: Số ảnh nhận diện sai là 218/1000 (21.8%)

Kiểm thử với máy vi tính:

- Test với 1000 ảnh đầu tiên của bộ t10k-test
- Kết quả: Tỷ lệ lỗi là 19.2%
- Thời gian chạy toàn bộ chương trình: 18s, trung bình 18ms/ảnh

Kết quả do MNIST công bố:

- Mô hình Linear classifier (1-layer NN) – None preprocessing – Error rate 12.0%

Kết luận:

- Thời gian nhận diện trên FPGA nhỏ hơn rất nhiều so với thời gian nhận diện trên máy tính ($8\mu s$ trên FPGA với 18ms trên CPU), FPGA nhanh hơn khoảng 2000 lần.
- Sai lệch về tỷ lệ lỗi giữa máy tính và mạch có thể là do thuật toán trên mạch đã được tối giản và làm tròn, nên gây ra sai lệch giữa hai bên.

- Kết quả công bố của MNIST tốt hơn với kết quả nhận được có thể do nghiên cứu đó sử dụng thuật toán khác tối ưu hơn.

Như vậy độ chính xác của mô hình 1-layer NN tuyến tính là không quá cao. Muốn nâng cao chất lượng của mô hình thì cần những phương pháp khác như chuẩn hoá đầu vào, hoặc là sử dụng mô hình Neuron Network tốt hơn như là Multi-layer NN.

VI. Tài liệu tham khảo

1. Blog Machine Learning Cơ Bản (<https://machinelearningcoban.com>)
2. Bishop, Christopher M. “Pattern recognition and Machine Learning.”, Springer (2006)
(<http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf>)
3. Andre Ng. CS229 Lecture notes
([https://datajobs.com/data-science-repo/Generalized-Linear-Models-\[Andrew-Ng\].pdf](https://datajobs.com/data-science-repo/Generalized-Linear-Models-[Andrew-Ng].pdf))
4. A Tour of Machine Learning Algorithms
(<https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms>)
5. Linear regression (https://en.wikipedia.org/wiki/Linear_regression)
6. An overview of gradient descent optimization algorithms
(<http://runder.io/optimizing-gradient-descent>)
7. Stochastic gradient descent
(https://en.wikipedia.org/wiki/Stochastic_gradient_descent)
8. Multiclass classification (https://en.wikipedia.org/wiki/Multiclass_classification)
9. Softmax function (https://en.wikipedia.org/wiki/Softmax_function)
10. Softmax Regression
(<http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression>)
11. Improving the way neural networks learn
(<http://neuralnetworksanddeeplearning.com/chap3.html>)
12. CS231n Convolutional Neural Networks for Visual Recognition
(<https://cs231n.github.io/neural-networks-case-study>)
13. How the backpropagation algorithm works
(<http://neuralnetworksanddeeplearning.com/chap2.html>)
14. Simple Image Classification using Convolutional Neural Network—Deep Learning in python
(<https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8>)
15. SimpleNeuralNetwork (<https://github.com/huangzehao/SimpleNeuralNetwork>)
16. MNIST database of handwritten digits (<http://yann.lecun.com/exdb/mnist>)

VII. Phụ lục

1. Cấu trúc file database của MNIST

Database của MNIST gồm có 4 file:

```
train-images-idx3-ubyte: training set images
train-labels-idx1-ubyte: training set labels
t10k-images-idx3-ubyte:  test set images
t10k-labels-idx1-ubyte:  test set labels
```

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803 (2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

Các pixels được sắp xếp từ trái qua phải, từ trên xuống dưới, giá trị mỗi pixel là từ 0 đến 255. 0 nghĩa là background, 255 nghĩa là foreground.

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801 (2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

Giá trị các label là từ 0 đến 9.

Tương tự với **TEST SET LABEL FILE** và **TEST SET IMAGE FILE**.

2. Cấu trúc ảnh BMP

Basic BMP File Format				
Name		Size	Description	
Header		14 bytes	Windows Structure: BITMAPFILEHEADER	
	Signature	2 bytes	'BM'	
	FileSize	4 bytes	File size in bytes	
	reserved	4 bytes	unused (=0)	
	DataOffset	4 bytes	File offset to Raster Data	
InfoHeader		40 bytes	Windows Structure: BITMAPINFOHEADER	
	Size	4 bytes	Size of InfoHeader =40	
	Width	4 bytes	Bitmap Width	
	Height	4 bytes	Bitmap Height	
	Planes	2 bytes	Number of Planes (=1)	
	BitCount	2 bytes	Bits per Pixel 1 = monochrome palette. NumColors = 1 4 = 4bit palletized. NumColors = 16 8 = 8bit palletized. NumColors = 256 16 = 16bit RGB. NumColors = 65536 (?) 24 = 24bit RGB. NumColors = 16M	
	Compression	4 bytes	Type of Compression 0 = BI_RGB no compression 1 = BI_RLE8 8bit RLE encoding 2 = BI_RLE4 4bit RLE encoding	
	ImageSize	4 bytes	(compressed) Size of Image It is valid to set this =0 if Compression = 0	
	XpixelsPerM	4 bytes	horizontal resolution: Pixels/meter	
	YpixelsPerM	4 bytes	vertical resolution: Pixels/meter	
	ColorsUsed	4 bytes	Number of actually used colors	
ColorsImportant	4 bytes	Number of important colors 0 = all		
ColorTable		4 * NumColors bytes	present only if Info.BitsPerPixel <= 8 colors should be ordered by importance	
		Red	1 byte	Red intensity
		Green	1 byte	Green intensity
		Blue	1 byte	Blue intensity
		reserved	1 byte	unused (=0)
	repeated NumColors times			
Raster Data		Info.ImageSize bytes	The pixel data	

Sau ColorTable là Data, bao gồm giá trị của các pixel.

Thứ tự pixel trong file BMP là từ trái qua phải, từ dưới lên trên.

3. Phần mềm hỗ trợ giao tiếp với mạch FPGA

Đây là phần mềm tự phát triển để hỗ trợ máy tính giao tiếp với mạch. Phần mềm gồm 3 tab:

- Configure COM Port: cấu hình cho cổng COM
- Recognize digit: Gửi ảnh để mạch nhận diện và gửi label đúng giúp mạch có thể thực hiện online-learning
- Statistic: Thống kê số ảnh nhận diện sai dựa trên MNIST database file

Tab 1:

Hand Digit Recognition

Configure COM Port Recognize digit Statistics

COM port configuration

Port: COM1

Baudrate: 9600

Stop bits: One

Parity: None

Timeout: 0 second(s)

Check connection Connect Disconnect

Hình VII.3.1. Tab Configure COM Port

Tab này giúp cấu hình và kết nối cổng COM.

Tab 2:

Hand Digit Recognition

Configure COM Port Recognize digit Statistics

Send bitmap image

Image name: ...

Send image

Learn image label

Label: 0 (0 ~ 9)

Send label

COM port configuration

Port: COM1

Baudrate: 9600

Stop bits: One

Parity: None

Timeout: 0 second(s)

Hình VII.3.2. Tab Recognize digit

Khi “Send image”, phần mềm sẽ gửi ký tự “I” (0x49 ASCII) để thông báo sẽ gửi ảnh, sau đó sẽ gửi 784 bytes tương ứng 784 pixels.

Khi “Send label”, phần mềm sẽ gửi kí tự “R” (0x52 ASCII) để thông báo sẽ gửi label, sau đó sẽ gửi 1 byte là giá trị label.

Hiện tại phần mềm chỉ hỗ trợ gửi ảnh định dạng BMP 28x28 pixels, 8 bitdepth. Trước khi gửi, phần mềm sẽ sắp xếp lại thứ tự các điểm ảnh trong phần data của file BMP về dạng giống như image-set của MNIST, giúp đồng nhất việc Learn bằng MNIST database và Test bằng ảnh BMP.

Line 28	pixel 757	pixel 758	...	pixel 784		Line 1	pixel 1	pixel 2	...	pixel 28
Line 27	pixel 729	pixel 730	...	pixel 756		Line 2	pixel 29	pixel 30	...	pixel 56
.						.				
.	→
.						.				
Line 2	pixel 29	pixel 30	...	pixel 56		Line 27	pixel 729	pixel 730	...	pixel 756
Line 1	pixel 1	pixel 2	...	pixel 28		Line 28	pixel 757	pixel 758	...	pixel 784

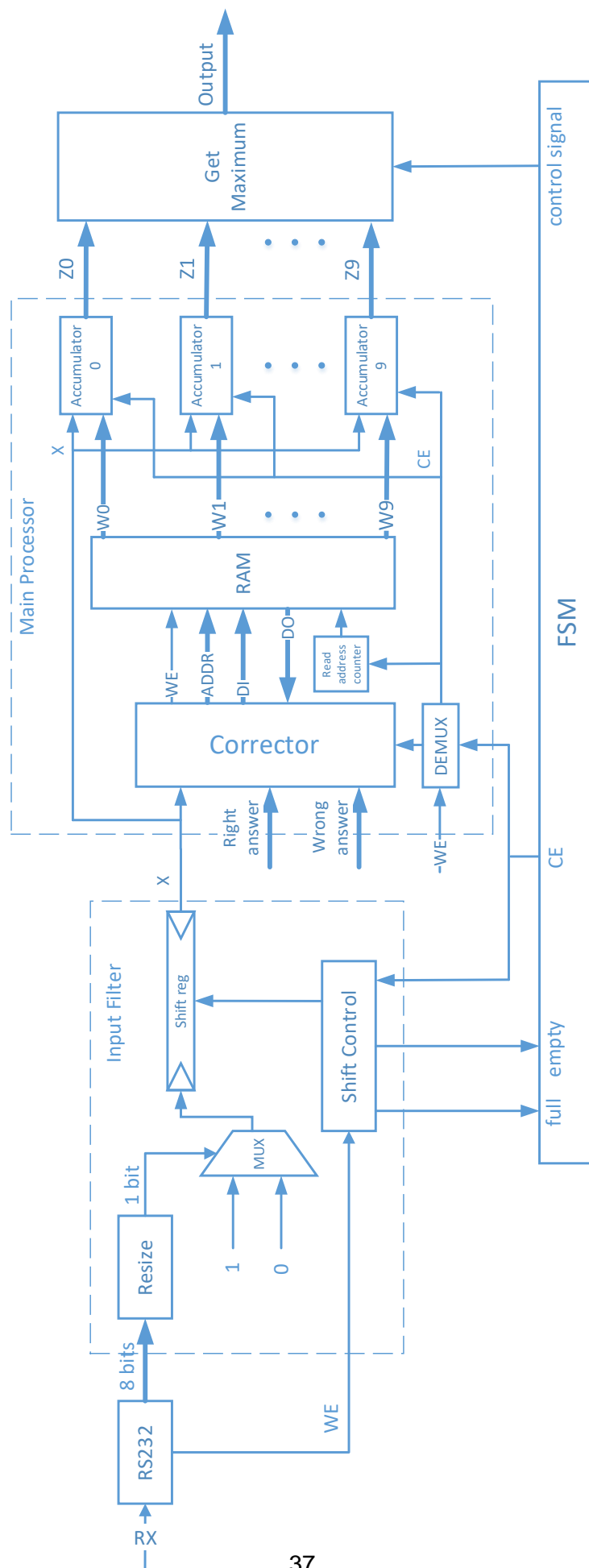
Hình VII.3.4. Vị trí các pixel trong file BMP được sắp xếp lại trước khi gửi xuống mạch

Tab 3:

Hình VII.3.3. Tab Statistic

Sau khi chọn image-set và label-set tương ứng của MNIST, khi ấn start, phần mềm sẽ gửi các ảnh trong image-set, đợi mạch gửi trả kết quả về để so sánh kết quả với label rồi đưa ra thống kê về số ảnh nhận diện sai.

4. Sơ đồ khối chi tiết của mô hình trên FPGA



Hình IV.1.1. Sơ đồ khối chi tiết của thiết kế trên FPGA