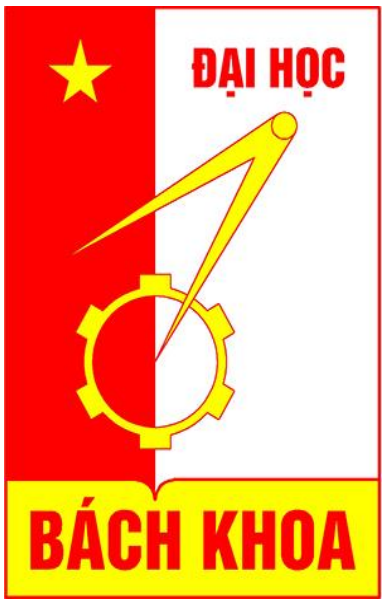
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

*



Báo cáo bài tập lớn

Học máy IT4866

Đề tài

Sử dụng mạng neuron trong nhận diện chữ số viết tay

*****

Sinh viên thực hiện:

Vũ Công Luật *20142745*

Nguyễn Văn Túc *20145070*

Hà Văn Quang *20143578*

Võ Anh Tuấn *20144963*

Giáo viên hướng dẫn: *TS. Thân Quang Khoát*

Hà Nội, ngày tháng năm 2018

Mục lục

Tổng quan

Ngày nay việc số hóa dữ liệu để lưu chữ trên máy tính đã trở nên phổ biến và gần như là bắt buộc của các tổ chức công ty. Tuy nhiên, một số lượng lớn các văn bản, thủ tục hành chính vẫn phải được viết bằng tay ví dụ như: đơn từ, ghi chú, biên bản hay bài thi. Ngoài ra các văn bản ghi chép cách đây nhiều năm cũng vẫn nằm dưới dạng viết tay và cần được số hóa để tiện truy cập, tìm kiếm, lưu trữ. Do đó việc có được một hệ thống nhận diện tự động chữ viết tay là rất hữu ích, giúp tiết kiệm công sức và tăng hiệu quả công việc.

Một hệ thống nhận diện chữ viết tay thường chia làm nhiều lớp nhận diện khác nhau như: nhận biết văn bản, tách các dòng, tách các từ, tác chữ cái, và nhận diện chữ cái. Trong phạm vi của đồ án môn học, hệ thống mà nhóm triển khai giới hạn trong lớp cuối cùng đó là nhận diện chữ cái, mà cụ thể hơn hệ thống sẽ chỉ có khả năng nhận diện chữ số viết tay (từ 0 tới 9).

Dữ liệu sử dụng là tập dữ liệu mẫu MNIST đã qua tiền xử lý.

Thuật toán sử dụng cho hệ thống là một mạng neuron. Trong phần sau của báo cáo sẽ đi sâu hơn về mạng neuron cũng như chi tiết thực hiện.

Ngôn ngữ sử dụng python, các thư viện sử dụng: numpy (phục vụ cho tính toán ma trận), scikit-learn (dùng để tính precision, recall, f1), matplotlib (dùng để vẽ đồ thị đánh giá).

Chương 1 Bài toán thực tế

1, Mục đích

Từ một văn bản chữ viết tay, ta cần chuyển văn bản đó thành văn bản trên máy tính (1 file dạng text) mà không cần can thiệp từ con người.

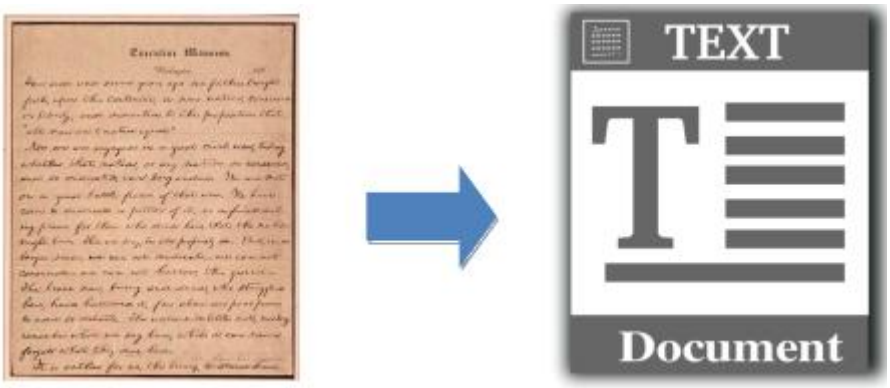


Figure 1: Hình ảnh minh họa

Quá trình thực hiện trên thực tế sẽ được chia thành các lớp nhận diện khác nhau. Bước đầu tiên nhận diện các dòng trong văn bản gốc, sau đó lớp thứ hai tách ra các từ trong mỗi dòng, lớp tiếp theo tách ra từng kí tự và cuối cùng ta nhận diện các kí tự tương ứng, đối với hệ thống mà nhóm xây dựng sẽ chỉ giới hạn trong việc nhận diện các kí tự là số (từ 0 tới 9).

2, Yêu cầu

Hệ thống sử dụng dữ liệu từ tập dữ liệu có sẵn đã qua tiền xử lý MNIST.

Sau khi huấn luyện, yêu cầu hệ thống phải nhận diện được các chữ số từ ảnh đầu vào và gán nhãn cho chữ số đó thuộc trong các nhãn từ 0 tới 9.

3, Kịch bản

Hệ thống hoạt động theo 3 giai đoạn:

Giai đoạn thu thập và biểu diễn dữ liệu: dữ liệu từ tập MNIST sẽ được chia làm 3 tập (training data, validation data, test data). Dữ liệu từ tập MNIST là tập các ảnh 28x28 pixel mỗi ảnh chứa một số có giá trị trong đoạn từ 0 tới 9. Để có thể biểu diễn dữ liệu đầu vào thuận lợi cho quá trình huấn luyện, với mỗi ảnh 28x28 pixel sẽ được vector hóa thành một vector 28x28 = 784 chiều. Kèm theo với mỗi ảnh tập MNIST đã bao gồm theo nhãn lớp cụ thể để phục vụ huấn luyện và kiểm thử hệ thống.

Giai đoạn huấn luyện: với dữ liệu từ tập training data ta tiến hành huấn luyện hệ thống, theo các tham số cụ thể nào đó, sẽ được trình bày kĩ hơn ở phần sau của báo cáo.

Giai đoạn phán đoán:

- Với mối ví dụ trong tập thử nghiệm khi được hệ thống xử lý sẽ được gán nhãn theo các nhãn số có giá trị trong đoạn [0,9] tương ứng.
- Để đánh giá hiệu năng hoạt động của hệ thống nhóm sử dụng các tiêu chí đánh giá phổ biến để đánh giá: classification accuracy, precision, recall, f1.

Chương 2 Thuật toán

1, Mạng Neuron

1.1 Tổng quan về mạng neuron.

Neuron là đơn vị cơ bản cấu tạo nên hệ thống thần kinh và là một phần quan trọng của bộ não. Ước tính có khoảng 100 tỷ neuron trong não người. Cấu tạo của mối neuron: gồm một thân chứa nhân, các sợi nhánh, sợi trục.

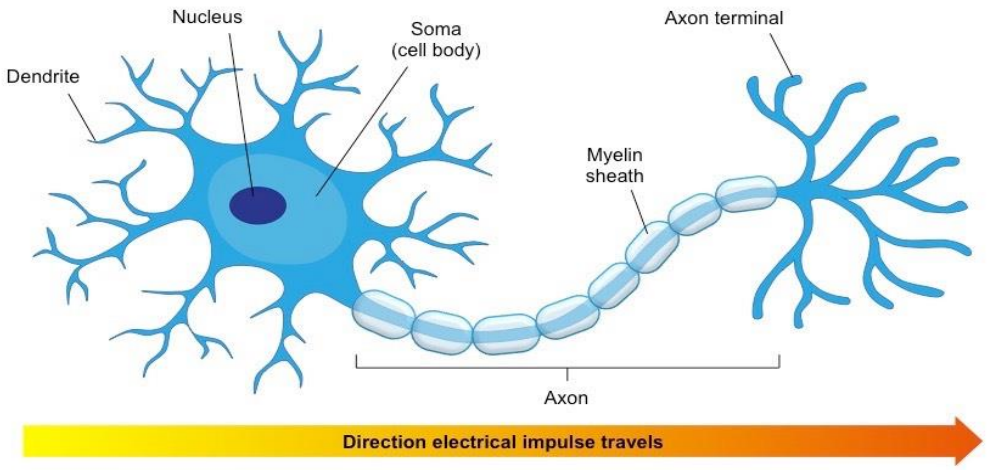


Figure 2: Hình ảnh của neuron thần kinh.

Bản thân quá trình học của hệ thần kinh là quá trình điều chỉnh kết nối giữa các neuron thần kinh thông qua các trải nghiêm, trong đó một số phản ứng (hành động) được tăng cường và một số khác bỉ giảm xuống.

Tương tự, một neuron trong mạng neuron nhân tạo cũng là một đơn vị cơ bản cẩu tạo lên mạng neuron. Mạng neuron nhân tạo là một mô hình thuật toán học máy mô phỏng lại quá trình học hỏi của hệ thống thần kinh trong tự nhiên. Quá trình học của mạng neuron nhân tạo là quá trình điều chỉnh các trọng số **w** (weight) và **b** (bias) sao cho mô hình có thể khái quát hóa tốt nhất tập dữ liệu học.

Mạng neuron nhân tạo đầu tiên được cho là phát minh của nhà khoa học Frank Rosenblatt trong khoảng từ (1950-1960) và gọi là perceptron sử dụng mô hình 3 lớp với đầu vào là dữ liệu biểu diễn dưới dạng nhị phân và đầu ra cũng là dạng nhị phân (0 và 1). Ngày nay perceptron đã không còn được sử dụng mà thay vào đó là các mô hình mạng neuron hiệu quả hơn. Với năng lực tính toán cao mạng neuron đang là thuật toán phổ biến dùng để giải quyết các bài toán nhận diện hình ảnh, nhận dạng giọng nói, xử lý ngôn ngữ tự nhiên… .

1.2 Kiến trúc của một mạng neuron.

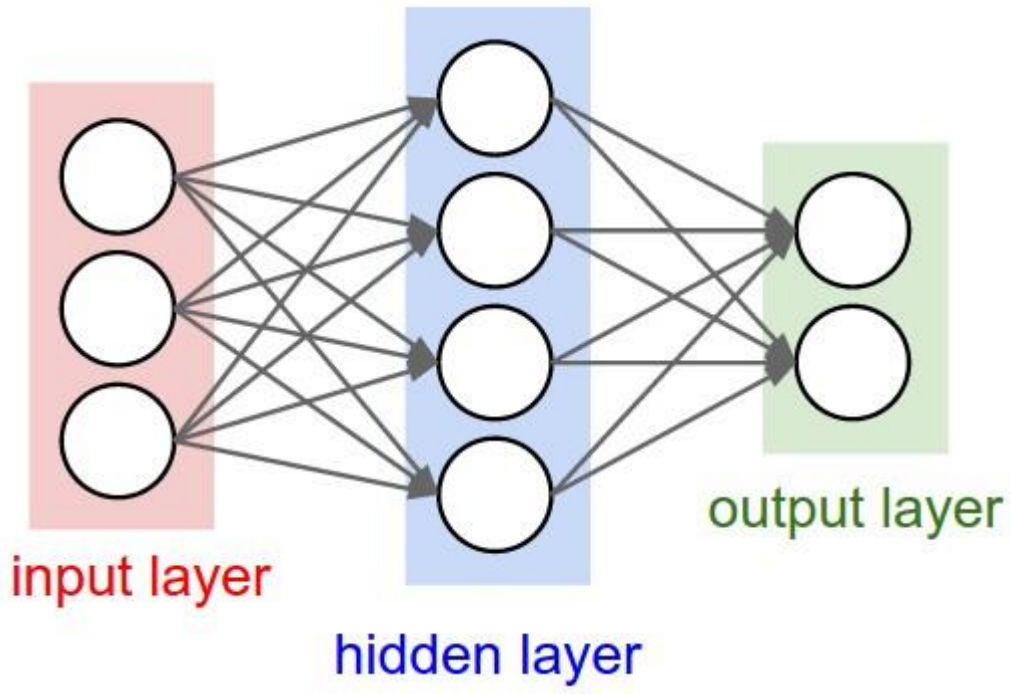


Figure 3: Kiến trúc đơn giản của một mạng neuron.

Mạng neuron thường được chia làm 3 loại lớp:

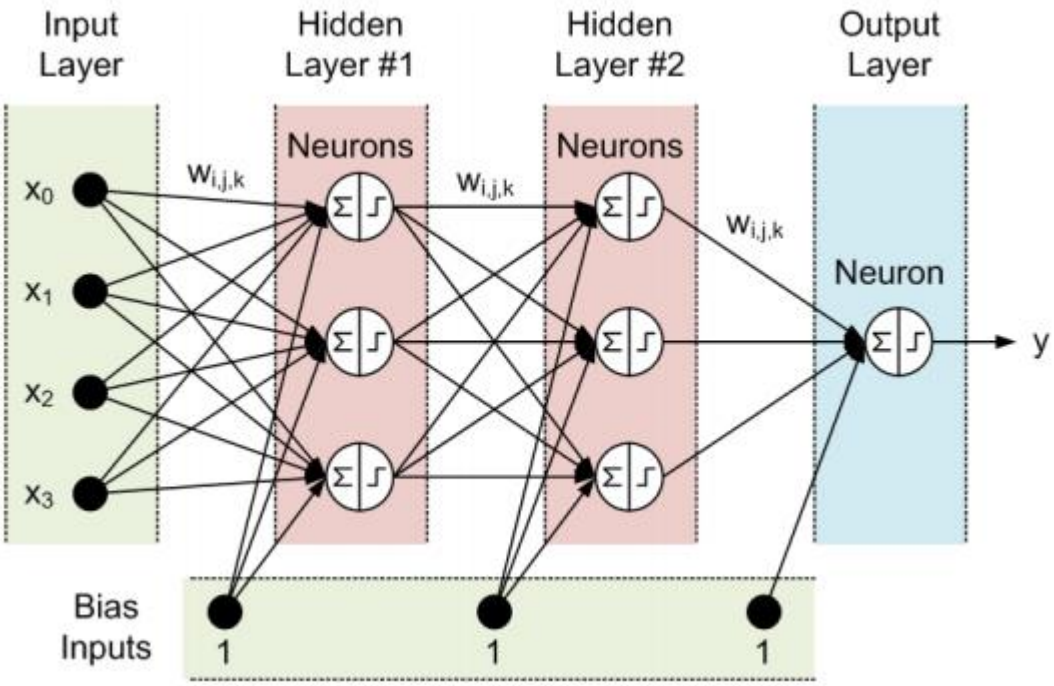
- Lớp đầu vào (input layer): là lớp đầu tiên của hệ thống, chứa các giá trị thông tin đầu vào. Giá trị của lớp này được lấy trực tiếp từ dữ liệu (dữ liệu đã qua tiền xử lý). Các neuron lớp này chỉ làm một nhiệm vụ là truyền các giá trị mà nó nhận được tới lớp thứ 2. Số lượng neuron bằng với số lượng các thông số của dữ liệu đưa vào.

  Ví dụ: Một hình ảnh đen trắng 28x28 pixel thì số neuron đầu vào sẽ là 28x28 = 784 neuron.
- Lớp ẩn (hidden layer): nằm giữa lớp đầu vào và lớp đầu ra. Số lượng neuron ở lớp này là tùy chọn, thông thường càng nhiều neuron trong lớp ẩn thì mô hình mà ta thu được sẽ có năng lực càng cao, tuy nhiên sẽ làm tăng lên khối lượng tính toán.
- Lớp đầu ra (output layer): dùng thông tin từ lớp ẩn để đưa ra kết luận cuối cùng. Có nhiều cách lựa chọn số lượng neuron ở lớp đầu ra, nhưng thông thường số neuron đầu ra sẽ được lựa chọn theo số nhãn lớp cần phân loại (đối với bài toán phân loại).

Quá trình tính toán dữ liệu đầu vào và đưa ra kết quả đầu ra của mỗi neuron thông qua một hàm kích hoạt, thường sử dụng các hàm phổ biến như là: sigmoid(), tanh().

2, Các giải thuật học.

2.1 Lan truyền xuôi (feed forward).



Đây là giải thuật đơn giản nhất của mạng neuron. Thông tin được lan truyền từ lớp đầu vào, đi qua lớp ẩn và lan truyền tới đầu ra, theo chỉ một chiều và không có chu trình hoặc vòng lặp trong mạng.

$$a_i = activation(\Sigma w_i x_i + 1)$$

Gắn liền với quá trình lan truyền xuôi là hàm kích hoạt (activation function), đối với mỗi neuron trên mỗi tầng cụ thể sẽ nhận dữ liệu đầu vào từ đầu ra của tất cả các neuron ở tầng ngay trước nó và thông qua hàm kích hoạt để tính toán đầu ra cho các neuron tầng tiếp theo, quá trình này cứ tiếp diễn cho tới khi lan truyền tới tầng đầu ra. Neuron tầng đầu ra cũng sử dụng hàm kích hoạt này để đưa ra kết quả cuối cùng.

Hàm kích hoạt mà nhóm sử dụng là hàm ***sigmoid()***, hàm ***sigmoid()*** có giá trị trong khoảng (0,1).

- Ưu điểm của hàm ***sigmoid()****:* là dễ dàng tính được đạo hàm.
  sigmoid(z)'* = *sigmoid(z)(1 – sigmoid(z)).
- Nhược điểm của hàm ***sigmoid()***: là nguyên nhân gây ra hiện tượng neuron bão hòa (sẽ được trình bày ở phần sau của báo cáo).

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

2.2 Gradient Descent

Hàm lỗi (Loss function): để học được các trọng số **w**(weight)**, b**(bias) giúp cho mạng neuron có thể phán đoán kết quả tương lai tốt nhất tức là sai số đầu ra phải nhỏ nhất cho mỗi quan sát. Tiêu chuẩn thường dùng là **hàm lỗi(loss function)**, hàm lỗi sẽ dùng để so sánh sai số giữa kết quả đầu ra và kết quả thực sự của mỗi ví dụ học (training example), hàm lỗi thường được sử dụng là hàm ***quadratic loss function*.**

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x) - a\|^2.$$

Mục tiêu của việc học là đi tối ưu hàm lỗi (hội tụ hàm lỗi, hay giảm giá trị của hàm lỗi), khi đó các trọng số **w** (weight), **b** (bias) sẽ được học và sau đó sẽ được sử dụng để mạng neuron phán đoán các kết quả trong tương lai.

Để tối ưu hàm lỗi có một phương pháp phổ biến là gradient descent phương pháp này sẽ làm cho hàm lỗi (loss function) hội tụ. ***Gradient descent*** của hàm lỗi C(w, b) là một vector có hướng.

$$\nabla \mathrm{C} = (\partial \mathrm{C}/\partial \mathrm{w},\ \partial \mathrm{C}/\partial \mathrm{b})$$

Sử dụng ***Gradient descent*** để hội tụ hàm lỗi (loss function) ta lặp đi lặp lại việc cập nhật giá trị các trọng số **w** và **b.**

$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

Qua đó sẽ làm hội tụ hàm lỗi, **η** là tỷ lệ học (learning rate) việc lựa chọn tỷ lệ học tùy thuộc vào bài toán cụ thể và kích thước của dữ liệu huấn luyện, **η** nhỏ thì hàm **C(w,b)** sẽ hội tụ chậm, **η** lớn thì hàm **C(w, b)** sẽ hội tụ nhanh, tuy nhiên nếu **η** quá lớn thì có thể gây ra việc **C(w, b)** sẽ không thể hội tụ tại điểm cực tiểu.

2.3 Lan truyền ngược (back propagation).

Mục đích của ***gradient descent*** là để tính các trọng số **w**(weight) và **b**(bias) qua đó làm hội tụ hàm lỗi (loss function). Làm sao để có thể tính được ***gradient descent***, nếu chỉ dựa vào giải tích để tính đạo hàm của hàm lỗi theo các trọng số thì khối lượng tính toán sẽ là rất lớn, chưa kể một số hàm lỗi còn không thể tính được ***gradient descent***.

Có một phương pháp hiệu quả để giải quyết vấn đề trên, đó là sử dụng thuật toán lan truyền ngược (back propagation).

Ý tưởng của thuật toán là từ một ví dụ học **x** có nhãn đúng **y** sau khi cho lan truyền tiến (feed forward) qua mạng ta tính được đầu ra suy diễn **h**. Để học được các trọng số **w**(weight) và **b**(bias) thì ta phải làm cho sai số giữa **y** và **h** là nhỏ nhất tức là độ lệch giữa chúng là nhỏ nhất, độ lệch này ta gọi là lỗi (error) và được kí hiệu là δ, lỗi được tính ở mỗi neuron lớp đầu ra bằng công thức.

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L).$$

Trong đó C là hàm lỗi, a là kết quả đầu ra suy diễn từ hàm kích hoạt.

Tại mỗi bước lan truyền ngược, lỗi tầng trước sẽ được tính dựa trên lỗi của tầng sau theo công thức.

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l),$$

Sau khi có được lỗi tại mỗi tầng, ta tiến hành tính **gradient descent** tại mỗi tầng theo công thức sau:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l. \qquad \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l.$$

2.4 Vấn đề Overfiting.

Một vấn đề phổ biến thường gặp phải đối với các giải thuật học máy là vấn đề overfiting. Các trọng số học được khớp rất tốt trên tập học nhưng lại khái quát rất kém trên dữ liệu mới (hay độ chính xác cao trên tập học nhưng lại rất thấp trên tập thử nghiệm).

Để giải quyết vấn đề này có rất nhiều cách:

- Tăng khối lượng tập học. Tuy nhiên trong một số trường hợp không phải lúc nào ta cũng có dữ liệu đủ lớn để huấn luyện.
- Giảm số tầng ẩn hoặc số lượng neuron ở tầng ẩn. Các này ít được sử dụng bởi giảm số lượng neuron, số tầng ẩn đồng nghĩa với việc mô hình của chúng ta sẽ bị giảm đi năng lực tính toán.
- Sử dụng một số phương pháp phổ biến khác (L1, L2 regularization,…).

Đối với hệ thống mà nhóm triển khai, nhóm sử dụng L2 regularization để giảm thiểu overfiting.

Nội dung của chuẩn L2 đó là thêm vào hàm lỗi một đại lượng R.

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2,$$

Mục đích khi cộng thêm R vào hàm lỗi là để khi thuật toán thực hiện tối ưu C(w,b) thì cũng sẽ làm giảm giá trị R tức là làm giảm **w**(weight).

Tại sao giảm w thì lại giúp giảm overfiting. Ta liên hệ với hồi quy tuyến tính.

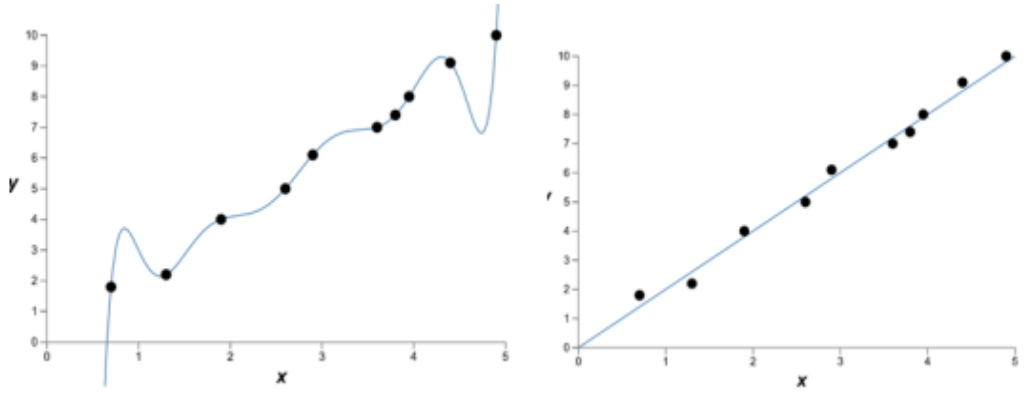


Figure 4: Đồ thị hàm số của hàm bậc 9 (bên trái) và hàm bậc nhất (bên phải).

Trên hình là hai đồ thị biểu diễn 10 điểm dữ liệu bằng 2 hàm số khác nhau. Hàm số ở đồ thị phía bên tay trái là hàm phi tuyến bậc 9:

$$y = a_0x^9 + a_1x^8 + \ldots + a_9$$

Hàm số ở phía bên tay phải là hàm số tuyến tính bậc nhất:

y = 2x

Hàm phi tuyến bậc 9 cho thấy nó khớp rất tốt với dữ liệu nhưng nó lại không thể hiện được xu hướng của dữ liệu (không khái quát hóa) chính vì vậy mà ta nói mô hình với hàm phi tuyến bậc 9 bị overfiting. Trong khi đó hàm tuyến tính một biến số lại biểu diễn đúng được xu hướng của dữ liệu, ta nói mô hình với hàm tuyến tính bậc nhất khái quát hóa tốt dữ liệu.

Chính vì vậy đôi khi hàm mà máy học quá phức tạp sẽ làm cho mô hình khớp rất tốt trên dữ liệu được cung cấp nhưng khả năng khái quát hóa sẽ rất kém so với hàm đơn giản.

Đối với mạng neuron các trọng số **w**(weight) có thể biến đổi từ âm vô cùng tới dương vô cùng làm tăng độ phức tạp của hàm học được. Khi sử dụng L2 để giảm giá trị của **w**(weight) về tiệm cận 0 sẽ làm cho giảm độ phức tạp của mô hình mà ta học được làm cho mạng neuron có thể khái quát dữ liệu tốt hơn.

2.5 Vấn đề neuron báo hòa

Neuron báo hòa là hiện tượng thường xảy ra khi mỗi neuron đầu ra cho kết quả gần với giá trị lớn nhất hoặc nhỏ nhất trong khi kết quả đúng là ngược lại, trên thực tế khi bắt đầu huấn luyện mạng neuron hiện tượng này có khả năng xảy ra rất cao. Đối với các hàm kích hoạt phổ biến hay được sử dụng như hàm sigmoid(), tanh() đồ thị hàm số của các hàm sẽ có độ dốc rất thấp khi đầu ra gần với 0 hoặc 1 đối với hàm sigmoid() và -1 hoặc 1 đối với hàm tanh() nên giá trị đạo hàm của hàm số theo các biến sẽ nhỏ. Dẫn đến tốc độ học **w**(weight) và **b**(bias) sẽ chậm khi sử dụng **gradient descent** (do phải tính đạo hàm).

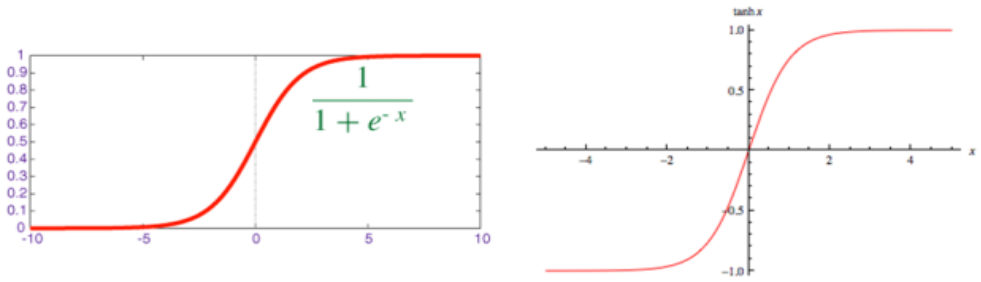


Figure 5: Đồ thị hàm số của hàm sigmoid (bên trái) và hàm tanh (bên phải)

Khi sử dụng hàm kích hoạt **sigmoid()**, và hàm lỗi Quadratic Loss Function.

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x)-a\|^2.$$

Để tính gradient descent ta gặp phải một vấn đề là :

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$
$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z),$$

Trong biểu thức này tồn tại σ’(z) liên hệ với đồ thị của hàm σ(z) khi σ(z) xấp xỉ 0 hoặc 1 thì đạo hàm của nó sẽ rất nhỏ dẫn tới mạng học các trọng số sẽ rất chậm. Để giải quyết vấn đề này có một phương pháp thay thế hàm lỗi ***Quadratic*** thành hàm ***Cross entropy.***

$$C = -\frac{1}{n}\sum_x [y\ln a + (1-y)\ln(1-a)],$$

Đạo hàm của ***Cross Entropy*** theo **w**(weight) và **b**(bias) loại bỏ được σ’(z) ra khỏi biểu thức, giúp ngăn ngừa hiện tượng neuron bão hòa ở lớp đầu ra.

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z)-y).$$

$$\frac{\partial C}{\partial b} = \frac{1}{n}\sum_x (\sigma(z)-y).$$

Tuy nhiên sử dụng Cross entropy chỉ giúp mô hình giải quyết được vấn đề ở neuron lớp đầu ra mà lại gần như không có tác dụng đối với neuron ở các lớp ẩn (hidden layer). Lý do là bởi vị trong quá trình học mạng neuron cần phải thực hiện bước lan truyền ngược lại lỗi để tính gradient descent ở các tầng trước. Tại mỗi lớp lỗi được tính từ lỗi của tầng sau liền kề với nó theo công thức:

$$\delta^l = ((w^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l),$$

Công thức tính lỗi ở mỗi lớp ẩn trong mạng neuron trên tồn tại $\sigma'(z)$ nên không thể tránh được neuron ở các tầng ẩn gặp phải hiện tượng bão hòa, khi hàm ***sigmoid()*** có giá trị gần với 0 hoặc 1. Tuy nhiên có một kĩ thuật có thể khắc phục trong trường hợp này. Khi ta nhìn vào hàm ***sigmoid()***:

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}. \qquad z = \sum_j w_j x_j + b$$

Ta sẽ thấy hàm nhận các giá trị là 0 hoặc 1 khi z tiến dần tới âm vô cùng hoặc dương vô cùng, nên để hàm sigmoid() không có giá trị gần vơi 0 hoặc 1 ta chỉ cần làm cho giá trị của z không quá lớn cũng như không quá bé bằng cách khởi tạo cho các trọng số **w**(weight) các giá trị nhỏ (tiệm cận 0) sẽ giúp giảm giá trị của z đồng thời sẽ ngăn cho hàm **sigmoid(z)** gần 0 hoặc 1. Khởi tạo các trọng số **w**(weight) của mỗi neuron trên một tầng cụ thể các giá trị nhỏ bằng cách, cho giá trị của **w** tuân theo phân phối xác suất chuẩn (Gaussian) sau đó mỗi giá trị thu được ta chia cho căn bậc 2 của số lượng neuron trên tầng đó.

Chương 3 Triển khai

1, Thu thập, biểu diễn dữ liệu.

Dữ liệu sử dụng trong bài tập lớn được nhóm thu thập từ tập dữ liệu mẫu MNIST dữ liệu đã qua tiền xử lý từ trước, bao gồm tập các ảnh 28x28 pixel, và được nhóm biểu diễn thành vector 784 chiều phục vụ cho quá trình huấn luyện mô hình. Từ tập MNIST nhóm chia tập ra làm 3 tập riêng biệt, tập dữ liệu huấn luyện (training data) bao gồm 50000 ví dụ, tập dữ liệu tối ưu (validation test) bao gồm 10000 ví dụ và tập dữ liệu kiểm thử (test data) bao gồm 10000 ví dụ.

2, Giai đoạn huấn luyện.

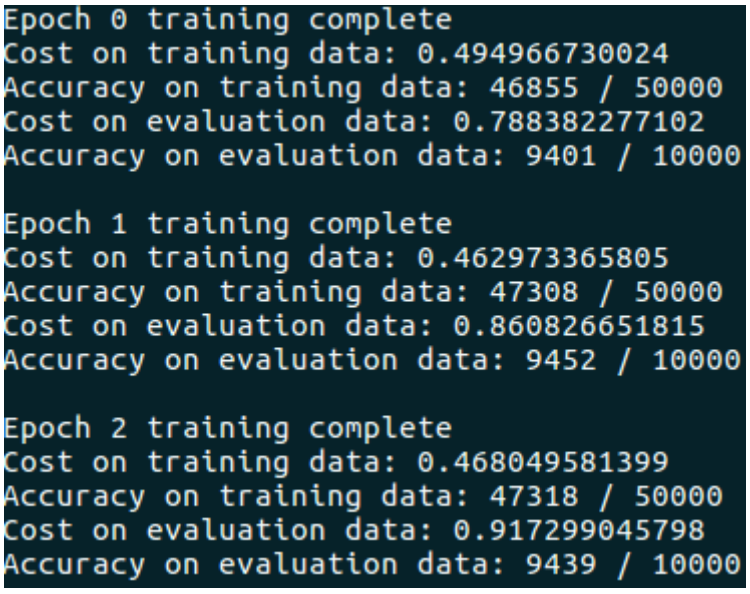


Figure 6: hình ảnh minh họa quá trình huấn luyện mạng neuron

Nhóm sử dụng mạng neuron 3 lớp: một lớp đầu vào (input layer) chứa 784 neuron, một lớp ẩn (hidden layer) có 30 neuron, một lớp đầu ra (output layer) có 10 neuron. Ban đầu mạng neuron mới khởi tạo, các trọng số w(weight) và b(bias) sẽ được khởi tạo mặc định theo phân bố xác suất Gaussian với trung bình là 0 và độ lệch chuẩn là 1, riêng đối với mỗi **w**(weight) trên mỗi neuron sau khi sử dụng Gaussian giá trí thu được sẽ được chia cho căn bậc hai của số lượng các **w**(weight) trên mỗi tầng (giúp cho giá trị của **w** lúc khởi tạo gần với 0).

Các giá trị của các tham số khác bao gồm:

- epoch: 30
- mini-batch size: 10
- learning rate: 0.5
- hệ số lamda: 5.0.

Thay vì huấn luyện mô hình trên toàn bộ tập học cùng một lúc, nhóm sử dụng kĩ thuật mini-batch với kích thước là 10 ví dụ học cho 1 quá trình huấn luyện, đối với một epoch tập dữ liệu học bao gồm 50000 ví dụ sẽ được đảo lộn ngẫu nhiên và chia làm nhiều mini-batch với kích thước 10 ví dụ một để cho vào huấn luyện, quá trình lặp lại từ epoch 1 tới epoch 30.

3, Gia đoạn phán đoán suy diễn.

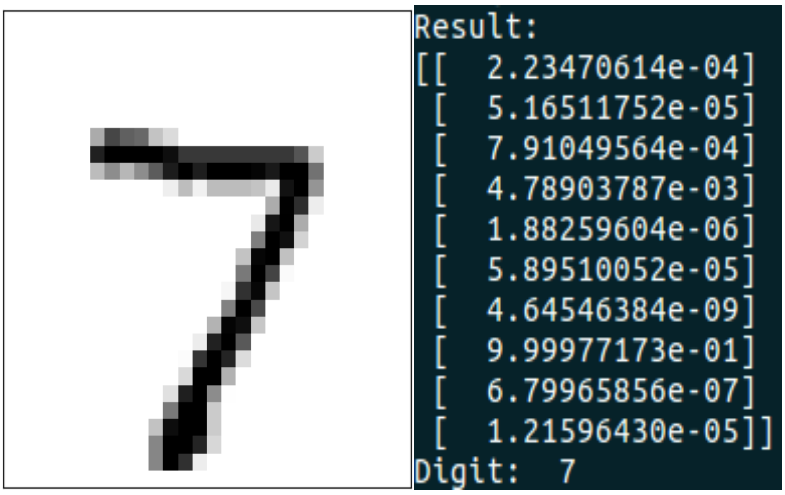


Figure 7: hình ảnh minh họa cho kết quả suy diễn từ mô hình

Sau giai đoạn huấn luyện nhóm thu được mô hình mang neuron có khả năng phán đoán, gán nhãn cho ảnh đầu vào từ tập thử nghiệm vào một trong các nhãn từ 0 tới 9. Đối với mối ví dụ từ tập kiểm thử sau khi đi qua mô hình sẽ cho ra kết quả là một vector 10 chiều, mỗi chiều sẽ có giá trị nằm trong đoạn [0,1] ví dụ được phân vào nhãn tương ứng với chiều của vector kết quả cho giá trị lớn nhất.

Chương 4 Kết quả

1, Độ chính xác mà hệ thống đạt được.

Độ chĩnh xác mà hệ thống đạt được trên các khối lượng tập học khác nhau. Đối với tập học có khối lượng 50000 độ chính xác của hệ thống lên tới trên 95%.

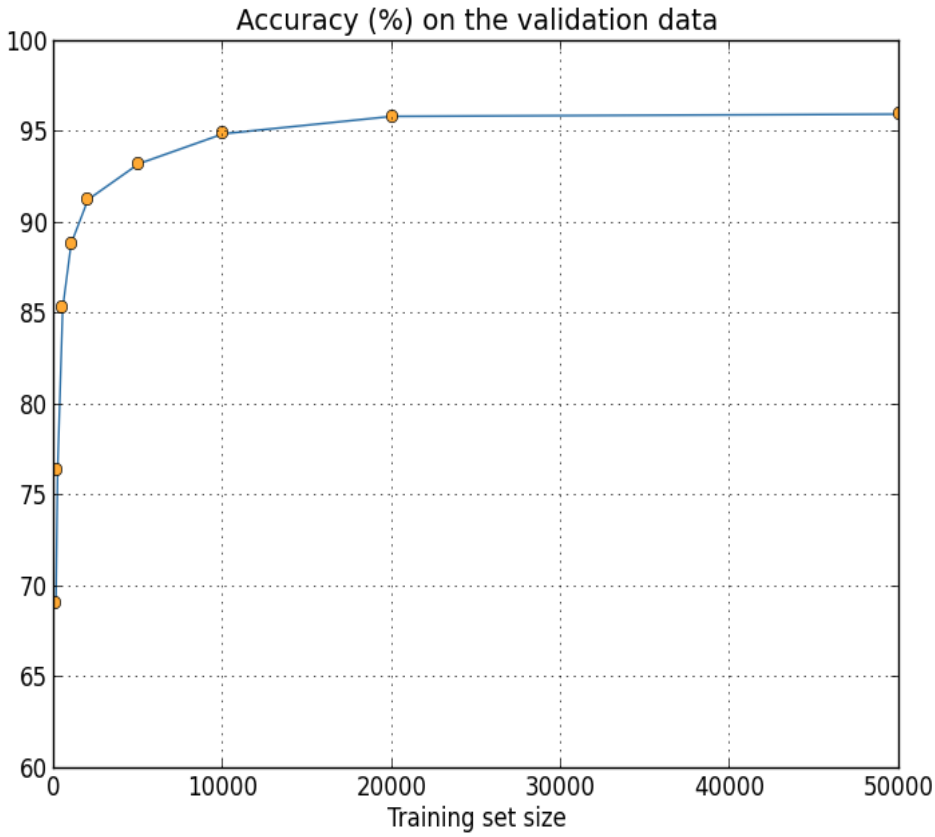


Figure 8: Độ chính xác của mô hình trên các khối lượng tập học khác nhau

2, Tốc độ học của mô hình trên các tỉ lệ học khác nhau.

Tỷ lệ học (learning rate) khác nhau ảnh hưởng tới tốc độ học của mô hình. Nếu tỷ lệ học quá nhỏ mô hình sẽ học chậm, tỷ lệ học cao mô hình học nhanh nhưng nếu quá cao sẽ dẫn đến trường hợp **gradient descent** của mô hình không thể chạm đến được điểm cực tiểu toàn cục, vì vậy giá trị hàm lỗi sẽ không giảm mà chỉ giao động quanh một giá trị nào đó. Hình 9 là kết quả thực nghiệm của hệ thống mà nhóm thử nghiệm trên 3 giá trị tỉ lệ học khác nhau. Từ đồ thị trong hình 9 có thể thấy khi $\eta = 2.5$ tỉ lệ học có giá trị trị lớn, hàm lỗi của mô hình không hội tụ được. Còn khi $\eta = 0.025$ giá trị này là quá nhỏ khiến cho mô hình học chậm. Giá trị $\eta = 0.25$ là giá trị thích hợp nhất.

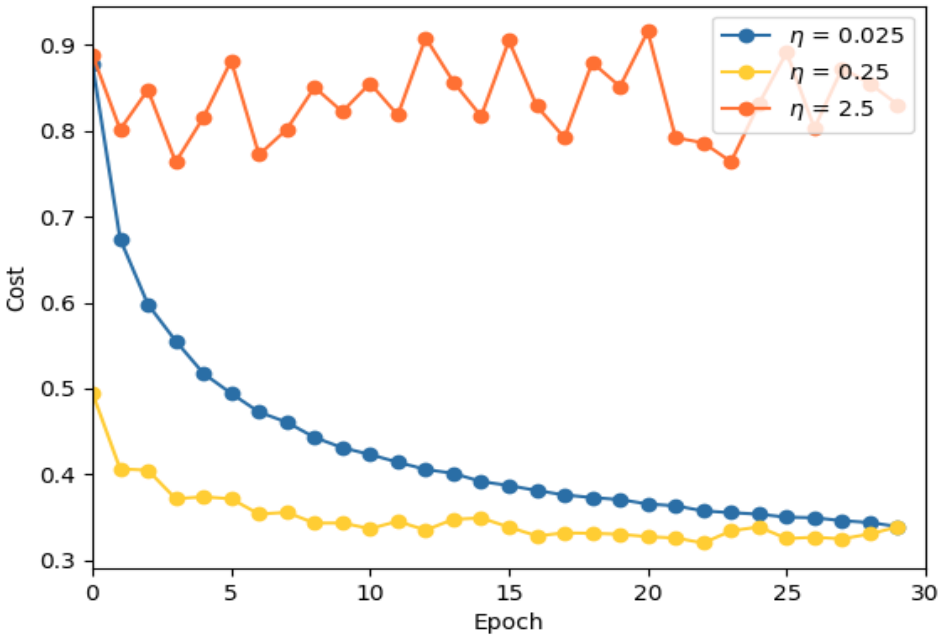


Figure 9: Tốc độ học của mô hình trên các giá tỉ lệ học khác nhau.

3, Tốc độ học của mô hình trên các hàm lỗi khác nhau.

Như đã nói ở các phần trước, hàm lỗi sử dụng để huấn luyện cũng ảnh hưởng một phần không nhỏ tới tốc độ học của mô hình. Khi sử dụng hàm Cross-Entropy sẽ cho tốc độ học nhanh hơn hàm Quadratic vì nó tránh được hiện tượng neuron ở lớp đầu ra bị bão hòa.

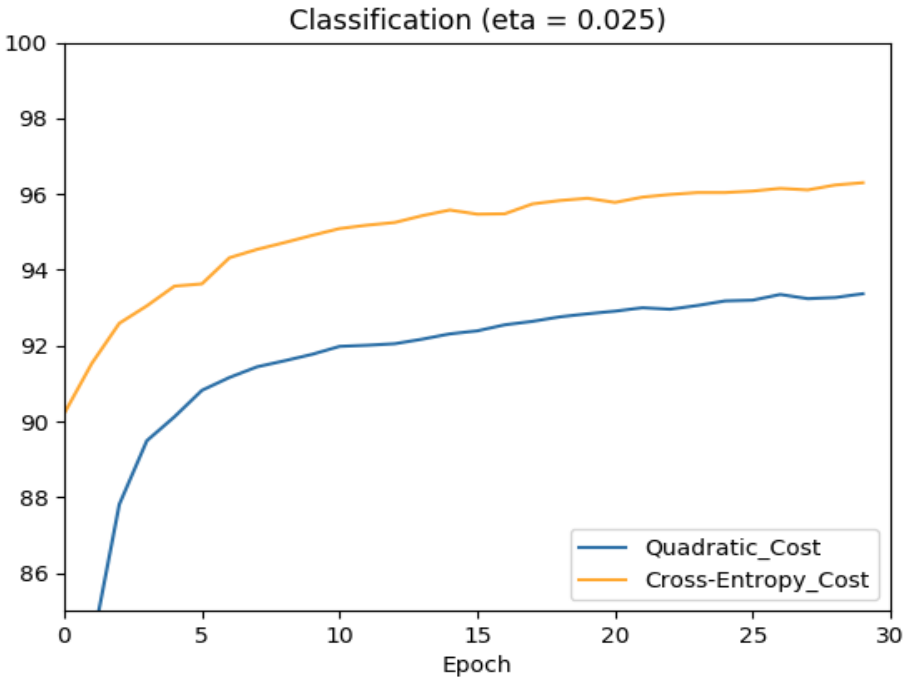


Figure 10: Tốc độ học của mô hình khi sử dụng hàm lỗi khác nhau.

4, Tốc độ học của mô hình khi khởi tạo giá trị **w**(weight) nhỏ so với khi khởi tạo **w**(weight) ngẫu nhiên theo phân phối Gaussian.

Như đã trình bày ở phần trước, khi ta khởi tạo giá trị **w**(weight) ban đầu cho mạng neuron giá trị nhỏ (gần với giá trị 0, âm hoặc dương) mô hình sẽ tránh được hiện tượng neuron bão hòa xảy ra ở các neuron ở các tầng ẩn. Dưới đây là kết quả đánh giá thực nghiệm của mô hình đạt được.

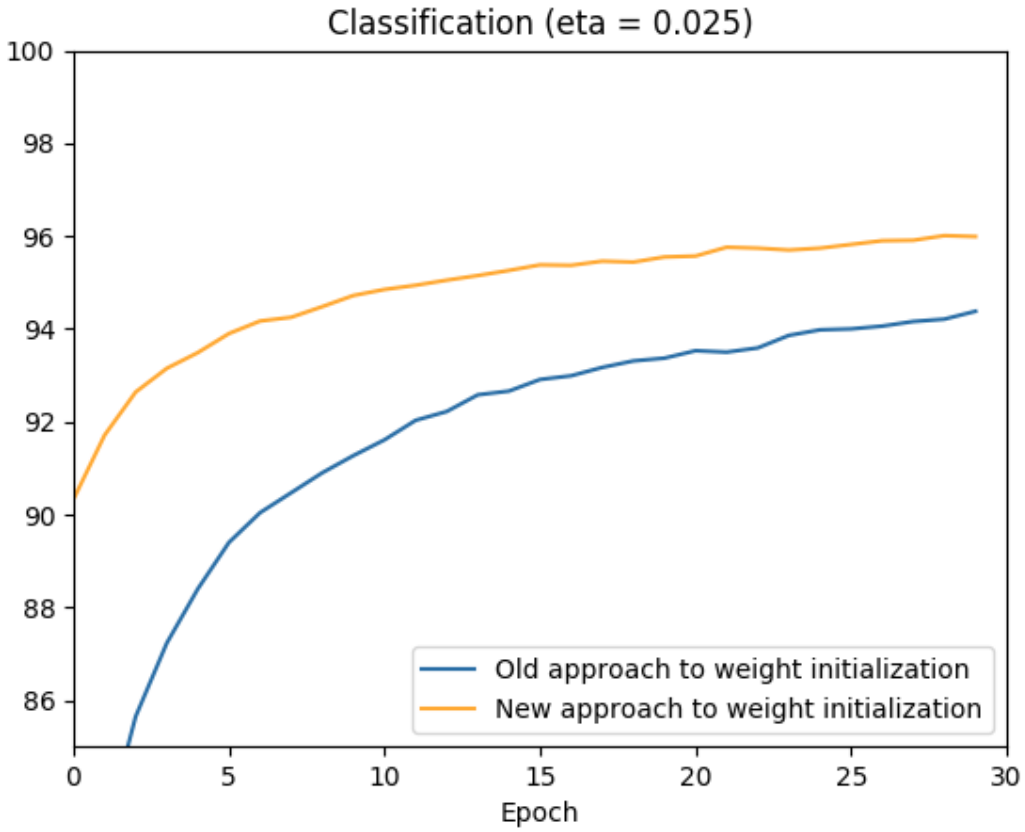


*Figure 11: Tốc độ học của mô hình khi sử dụng các kĩ thuật khởi tạo trọng số **w** khác nhau.*

Mô hình khi sử dụng kĩ thuật khởi tạo giá trị với trọng số **w**(weight) nhỏ (đường màu vàng) cho tốc độ học nhanh hơn so với việc chỉ sử dụng khởi tạo trọng số **w** theo phân phối chuẩn Gaussian.

5, Overfitting

Khi mô hình không sử dụng chuẩn hóa, với số lượng neuron trong tầng ẩn lên tới hàng trăm neuron mô hình bắt đầu gặp phải vấn đề overfitting.

Các tham số sử dụng để huấn luyện mô hình khi mô hình gặp phải vấn đề overfitting là:

- epoch: 30
- mini-batch size: 10
- learning rate: 0.5
- số tầng ẩn: 1
- số lượng neuron ở tầng ẩn: 150

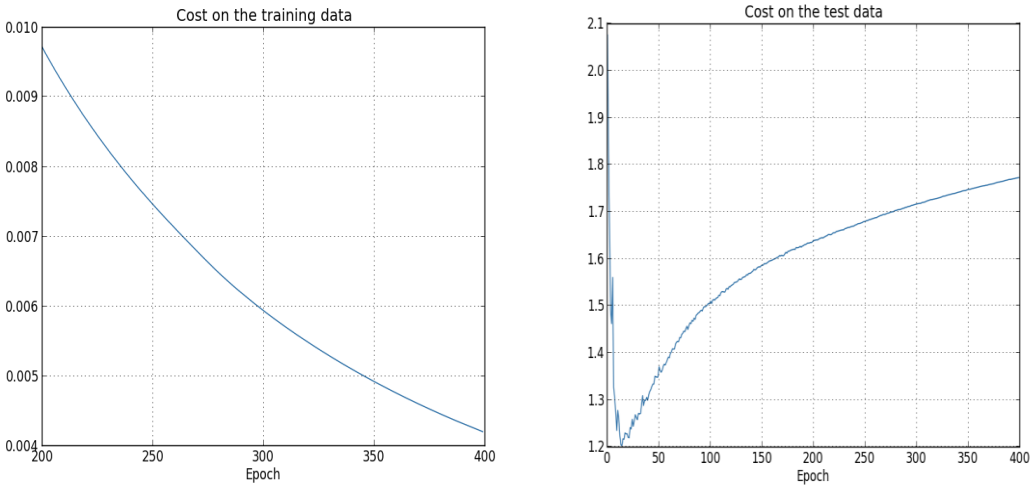


Figure 12: Đồ thị so sánh giá trị của hàm chi phí trên tập thử nghiệm và tập kiểm thử.

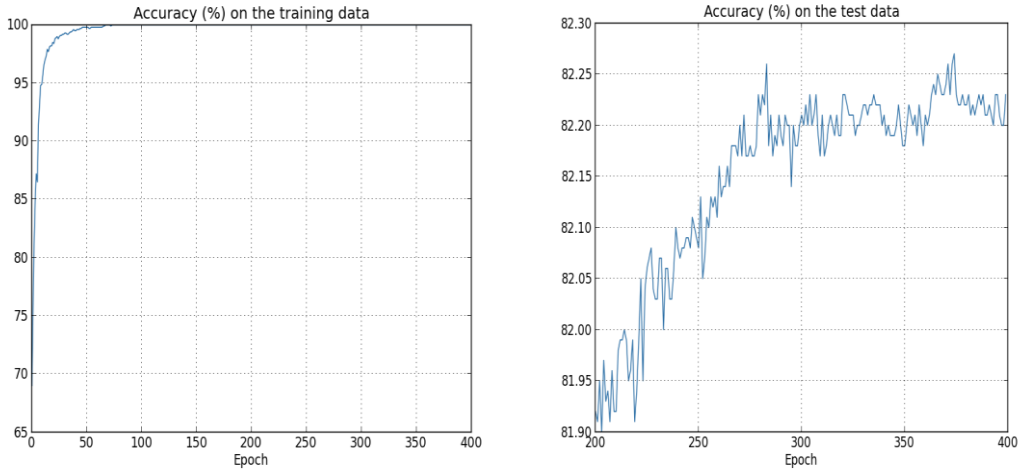


Figure 13: Đồ thị so sánh độ chính xác của mô hình trên tập thử nghiệm và tập kiểm thử.

Hình 11 cho thấy mô hình đang bị overfitting. Trong khi hàm lỗi (hay hàm chi phí) giảm dần khi số lượng epoch tăng lên khi đánh giá trên tập huấn luyện, thì đối với tập kiểm thử hàm lỗi chỉ giảm ở một vài epoch đầu tiên và lại có xu hướng tăng lên khi số lượng epoch tăng.

Hiện tượng đó cũng xảy ra tương tự khi tính trên độ chính xác của mô hình đạt được trên tập học và tập thử nghiệm, trong khi trên tập học độ chính xác đạt giá trị rất cao, 100% khi số lượng epoch là 50, với số lượng epoch tăng lên độ chính xác của mô hình vẫn giữ ở mức lý tưởng 100%. Còn khi đánh giá trên tập thử nghiệm độ chính xác của mô hình cũng tăng lên khi số lượng các epoch tăng, tuy nhiên sẽ không đạt được tối đa, và khi số lượng epoch vượt qua một ngưỡng nào đó ví du như trên hình là epoch bằng 300 thì độ chính xác của mô hình không tăng mà chỉ dao động quanh ngưỡng 82%.

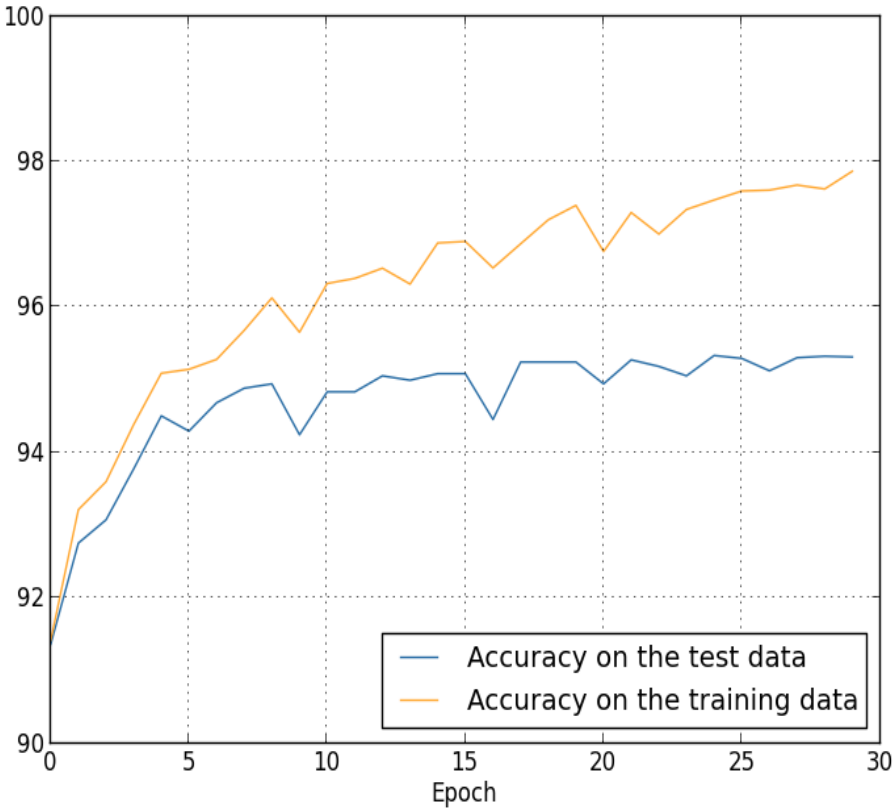


Figure 14: Hình ảnh cho thấy mô hình đang gặp phải vấn đề overfitting.

Hình 13 cho ta thấy mô hình đang bị overfitting khi đem đánh giá độ chính xác của mô hình đạt được trên tập huấn luyện và tập kiểm thử, ta thấy có độ chênh lệch lớn giữa hai tập.

6, Chuẩn hóa (Regularization)

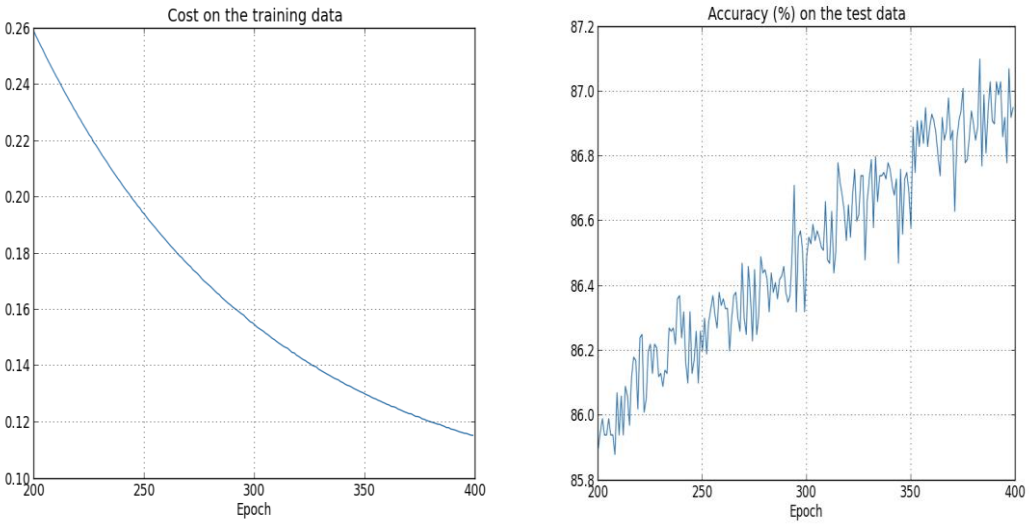


Figure 15:

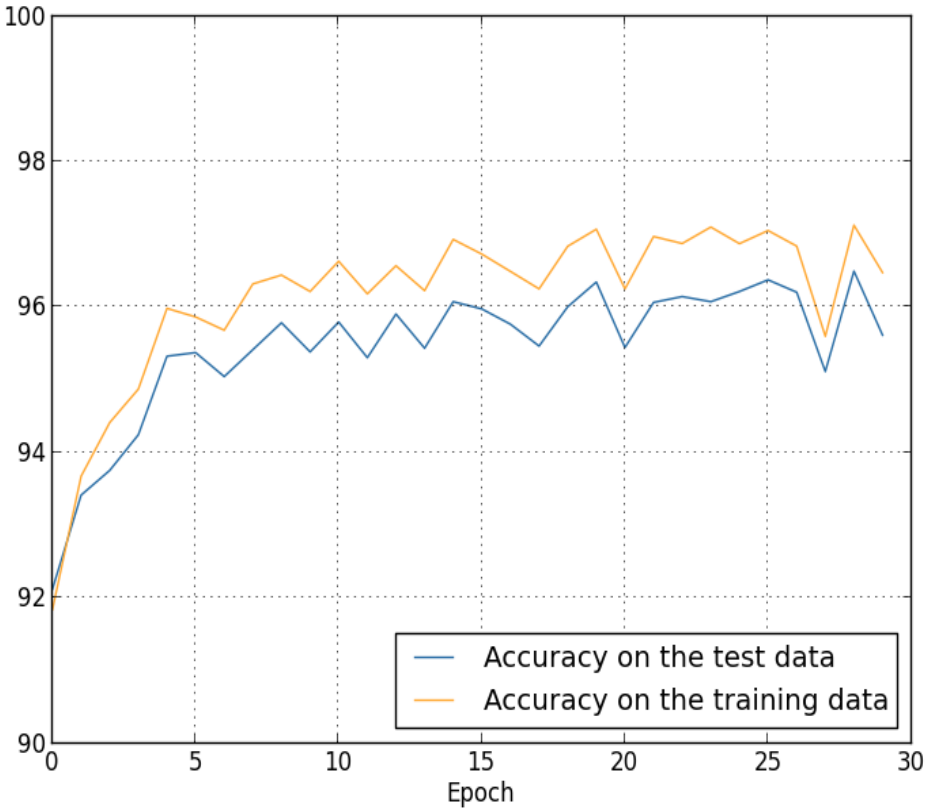


Figure 16

Khi sử dụng các kĩ thuật để chuẩn hóa mô hình, ta có thể ngăn ngừa được hiện tượng mô hình bị overfiting, hình 14 và hình 15 thu được từ kết quả thực nghiệm cho ta thấy mô hình đã giảm được đáng kể sự chệnh lệch xảy ra giữa tập dữ liệu huấn luyện và tập dữ liệu kiểm thử, và khi đó ta nói mô hình đã tránh được hiện tượng overfitting.

7, Đánh giá mô hình trên các độ đo khác.

Ngoài đánh giá mô hình dựa vào độ chính xác, còn có các độ đó khác để đánh giá chất lượng của mô hình được sử dụng khá phổ biến như độ đo: precision, recall, f1.

7.1 Precision là gì?

Độ đo precision là độ đo đánh giá chất lượng của một dự đoán đối với một nhãn cụ thể. Công thức tính precision cho từng nhãn cụ thể như sau:

$$\textbf{\textit{Precision}} = TP/(TP + FP)$$

TP: số lượng khẳng định **đúng.**

FP: số lượng khẳng định **sai.**

Ví dụ: Mô hình học máy gán nhãn ra được 100 ảnh là chữ số 1 trong đó có 80 ảnh được gán nhãn đúng còn 20 ảnh được gán nhã sai. Suy ra:

$$\textbf{\textit{Precision}} = 80/(80+20) = 0.8.$$

7.2 Recall là gì?

Độ đo Recall là độ đo đánh giá khả năng tìm thấy một nhãn trong dữ liệu được tính bởi công thức:

$$Recall = TP/(TP + FN)$$

TP: số lượng khẳng định **đúng.**

FN: số lượng phủ định **sai**.

Ví dụ: Mô hình học máy với dữ liệu đầu vào có 100 ảnh là chữ số 1, nhưng khi qua mô hình gán nhãn chỉ gán nhãn được 85 ảnh là chữ số 1, 15 ảnh còn lại mô hình gán nhãn không phải chữ số 1. Suy ra:

$$Recall = 85/(85+15) = 0.85$$

7.3 F1 là gì?

Một hệ thống không thể đồng thời đạt được cả hai giá trị cao trên hai độ đo precision và recall, khi precision cao thì recall sẽ thấp và ngược lại. Cho nên trong khi đánh giá mô hình người ta có đưa thêm vào đó một độ đo khác để đánh giá hệ thống, đó là độ đo F1, hay còn gọi là độ đo trung bình điều hòa. Độ đo F1 được tính bởi công thức:

$$F1 = 2*precision*recall / (precision + recall).$$

7.4 Bảng đánh giá.

Bảng đánh giá độ hiệu quả của hệ thống đạt được trên các độ đo precision, recall, f1:

	Precision	Recall	F1
Nhãn 0	0,96492986	0,982653061	0,973710819
Nhãn 1	0,967297762	0,99030837	0,978667828
Nhãn 2	0,94581749	0,964147287	0,954894434
Nhãn 3	0,950048972	0,96039604	0,955194485
Nhãn 4	0,937438905	0,976578411	0,956608479
Nhãn 5	0,978873239	0,934977578	0,956422018
Nhãn 6	0,955942623	0,973903967	0,96483971
Nhãn 7	0,958538993	0,944552529	0,951494366
Nhãn 8	0,952723535	0,95174538	0,952234206
Nhãn 9	0,978494624	0,901883053	0,938628159

8, Hạn chế, khó khăn và hướng phát triển

Khó khăn trong quá trình xây dựng hệ thống của nhóm gặp phải đó là các thành viên trong nhóm chưa có đủ kiến thức, kinh nghiệm xây dựng một hệ thống học máy. Yếu tố thời gian cũng là khó khăn ảnh hưởng tới việc hoàn thiện hệ thống của nhóm, với thời gian của một kì học chỉ vỏn vẹn mấy tháng phải dàn trải cho nhiều bài tập lớn của các môn học khác không cho phép nhóm có đủ thời gian để đầu tư tìm hiểu, tự tay xây dựng và hoàn thiện hết

tất cả các pha trong một hệ thống học máy đầy đủ. Chính vì vậy hệ thống mà nhóm xây dựng gặp phải nhiều hạn chế, như đã nói nhóm không có đủ thời gian để tìm hiểu, xây dựng hết các pha của một hệ thống học máy đầy đủ, cho nên hệ thống vẫn chưa tự thu thập và tiền xử lý được dữ liệu thực mà vẫn còn phải sử dụng dữ liệu mẫu có sẵn đã qua tiền xử lý từ trước.

Về hướng phát triển của nhóm trong tương lai:

Qua bài tập lớn, nhóm đã học được rất nhiều kiến thức, kinh nghiệm trong quá trình xây dựng một hệ thống học máy, và từ hệ thống mà nhóm xây dựng nhóm có thể từ đó phát triển hệ thống, khắc phục những hạn chế còn tồn đọng đặc biệt trong pha thu thập và tiền xử lý dữ liệu. Ngoài ra nhóm sẽ áp dụng các kĩ thuật của Deep learning (sử dụng mạng neuron ConvNet) để cái tiến hệ thống nâng cao độ chính xác.

Tài liệu tham khảo

[1] Giá trình học máy của TS. Thân Quang Khoát

[2] Neuron network and deep learning của Michael Nielsen

[3] Machine Learning course của Andrew Ng.