ỦY BAN NHÂN DÂN TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

ĐỀ TÀI: TÌM HIỂU NEURAL NETWORK THÔNG QUA ỨNG DỤNG NHẬN DIỆN SỐ QUA HÌNH ẢNH

MÔN: KHAI PHÁ DỮ LIỆU

GIẢNG VIÊN HƯỚNG DẪN: THS. TRỊNH TẤN ĐẠT

Sinh viên thực hiện:

Nguyễn Tuấn Anh	:3118410013
Tăng Chí Chung	:3118410043
Nguyễn Ngọc Tiến Em	:3118410094

Thành phố Hồ Chí Minh tháng 5 năm 2022

Mục lục

Danh sách các hình

Lời mở đầu

Sự ra đời của chiếc máy vi tính là một đột phá trong lịch sử loài người, một bước tiến lớn trong khoa học. Máy vi tính đã giúp đỡ con người hoàn thành từ những công việc nhỏ nhặt như tính toán vài con số đến những nhiệm vụ to lớn như đưa con người lên mặt trăng một cách thuận lợi. Công dụng của máy vi tính vô cùng đa dạng. Sự đa dạng này là do con người đã tận dụng khả năng xử lý, tính toán và lưu trữ của máy để linh hoạt sáng tạo ra các ứng dụng khác nhau phục vụ cho từng mục tiêu riêng biệt.

Hiện nay, với hệ thống dữ liệu hình ảnh khổng lồ trên toàn thế giới, việc phân tích, xử lý để khai thác sử dụng thông tin trong ảnh là một nhu cầu thiết yếu. Tuy nhiên, với số lượng dữ liệu khổng lồ, việc phân tích, xử lý thủ công sẽ mất rất nhiều thời gian và nguồn nhân lực. Thực tiễn đã chứng minh, việc ứng dụng trí tuệ nhân tạo nói chung hay Deep Learning nói riêng vào các nhiệm vụ trên đã giúp tiết kiệm được nhiều thời gian và công sức. Từ giữa năm 2011, Google đã giới thiệu dự án Deep Learning sử dụng mạng neuron nhân tạo dùng cho nhận dạng giọng nói và sau đó mở rộng lên các lĩnh vực khác như Gmail, Google dịch, Google ảnh. Đối với bài toán nhận dạng chữ số viết tay, mạng nơron nhiều lớp sẽ được huấn luyện dựa trên các pixel, đơn vị nhỏ nhất của hình ảnh. Vì vậy mạng neuron nhân tạo là công cụ vô cùng thích hợp cho việc xử lý, phân tích hình ảnh và mang lại kết quả rất khả quan.

Nhận dạng chữ số người là một trong những ứng dụng tuyệt vời của thị giác máy tính nói riêng cũng như khoa học máy tính nói chung.Sau đây chúng ta sẽ “Tìm hiểu Neural Network thông qua ứng dụng nhận diện số qua hình ảnh”.

Dù đã hết sức cố gắng nhưng trong bài viết khó tránh khỏi những sai sót.

Chương I : Tổng quan

1. Nhu cầu thực tế:

Để giải quyết các bài toán có tính phi tuyến, phức tạp và đặc biệt trong các trường hợp mà mối quan hệ giữa các quá trình không dễ thiết lập một cách rõ ràng. Tiêu biểu như biến động chất lượng nước và sự thay đổi các yếu tố ảnh hưởng (tải lượng chất ô nhiễm, vị trí xả thải, lưu lượng nguồn nước,…).

Một ưu điểm vượt trội của mô hình mạng nơron nhân tạo là khả năng tự học và điều chỉnh các trọng số để kết quả tính toán phù hợp với thực tế mà không phụ thuộc vào ý kiến chủ quan. Vậy nên mạng nơron nhân tạo là giải pháp tốt cho các vấn đề nhận biết thực trạng và xu hướng hiện nay.

2. Khái quát Neural Network

Mạng nơ-ron nhân tạo (Neural Network) là một chuỗi các thuật toán được đưa ra để nỗ lực tìm kiếm các mối quan hệ cơ bản trong một tập hợp dữ liệu, thông qua quá trình bắt chước cách thức hoạt động của bộ não con người.

Mạng nơ-ron nhân tạo có thể thích ứng với các thay đổi trong đầu vào, do đó, nó đưa ra các kết quả tốt nhất có thể mà không cần phải thiết kế lại các tiêu chí đầu ra. Khái niệm mạng nơ-ron nhân tạo có nguồn gốc từ trí tuệ nhân tạo, đang nhanh chóng trở nên phổ biến trong sự phát triển của các hệ thống giao dịch điện tử.

Được sử dụng rộng rãi vào những thập niên 80, 90. Sau thập niên 90 thì bắt đầu phổ biến. Khoa học mạng nơ-ron nhân tạo đã bước ra thế giới vào giữa thế kỷ 20 đang phát triển nhanh chóng. Ngày nay, chúng ta đã xem xét những ưu điểm của mạng nơ-ron nhân tạo và những vấn đề gặp phải trong quá trình sử dụng chúng. Không nên quên rằng những nhược điểm của mạng ANN, vốn là một ngành khoa học đang phát triển, bị loại bỏ từng cái một và những ưu điểm của chúng đang tăng lên từng ngày. Điều này có nghĩa là mạng nơ-ron nhân tạo sẽ trở thành một phần không thể thiếu trong cuộc sống của chúng ta ngày càng quan trọng.

3. Đặc điểm Mạng nơ-ron nhân tạo

Mạng nơ-ron nhân tạo là sự bắt chước đơn giản của bộ não bao gồm mạng lưới dày đặc của các cấu trúc đơn giản như mạng nơ-ron của con người. Một "nơ-ron thần kinh" trong mạng nơ-ron nhân tạo là một hàm toán học có chức năng thu thập và phân loại thông tin theo một cấu trúc cụ thể.

Mạng nơ-ron nhân tạo có sự tương đồng chuẩn mạnh với các phương pháp thống kê như các đồ thị đường cong và phân tích hồi quy.

Mạng nơ-ron nhân tạo chứa các lớp bao hàm các nút (node) được liên kết với nhau. Mỗi nút là một tri giác (hay một nơron nhân tạo), cấu tạo tương tự như một hàm hồi quy đa tuyến tính.

Tri giác sẽ cung cấp tín hiệu được tạo bởi hàm hồi quy đa tuyến tính, tạo thành một hàm kích hoạt (có thể là phi tuyến).

Trong một tri giác đa lớp (MLP), các tri giác sẽ được sắp xếp theo các lớp liên kết với nhau. Lớp đầu vào thu thập các mẫu đầu vào, và lớp đầu ra nhận các phân loại hoặc tín hiệu đầu ra mà các mẫu đầu vào có thể phản ánh.

Ví dụ, mô hình có thể có đầu vào là một danh sách các đại lượng chỉ báo kỹ thuật về một chứng khoán nhất định, kết quả đầu ra tiềm năng có thể là các đề xuất "Mua", "Giữ" hoặc "Bán".

4. Ưu điểm và nhược điểm

Ưu điểm:

- Lưu trữ thông tin trên toàn bộ mạng: Thông tin như trong lập trình truyền thống được lưu trữ trên toàn bộ mạng, không phải trên cơ sở dữ liệu. Sự biến mất của một vài thông tin ở một nơi không ngăn cản mạng hoạt động.
- Khả năng làm việc với kiến thức chưa đầy đủ: Sau khi đào tạo NN, dữ liệu có thể tạo ra đầu ra ngay cả với thông tin không đầy đủ. Sự mất hiệu suất ở đây phụ thuộc vào mức độ quan trọng của thông tin bị thiếu.
- Có khả năng chịu lỗi: Việc một hoặc nhiều ô của NN bị hỏng không ngăn nó tạo ra đầu ra. Tính năng này làm cho các mạng có khả năng chịu lỗi.
- Tham nhũng dần dần: Mạng chậm dần theo thời gian và trải qua quá trình suy thoái tương đối. Sự cố mạng không ăn mòn ngay lập tức.
- Có bộ nhớ phân tán: Để NN có thể học được, cần phải xác định các ví dụ và dạy mạng theo đầu ra mong muốn bằng cách hiển thị các ví dụ này lên mạng. Sự thành công của mạng tỷ lệ thuận với các trường hợp đã chọn và nếu sự kiện không thể được hiển thị cho mạng ở tất cả các khía cạnh của nó, mạng có thể tạo ra kết quả sai
- Khả năng thực hiện học máy: Các mạng thần kinh nhân tạo tìm hiểu các sự kiện và đưa ra quyết định bằng cách nhận xét về các sự kiện tương tự.
- Khả năng xử lý song song: Mạng nơ-ron nhân tạo có sức mạnh số có thể thực hiện nhiều hơn một công việc cùng một lúc.

Nhược điểm:

- Sự phụ thuộc vào phần cứng: Mạng nơ-ron nhân tạo yêu cầu bộ xử lý có sức mạnh xử lý song song, phù hợp với cấu trúc của chúng. Vì lý do này, việc thực hiện các thiết bị là phụ thuộc.
- Hành vi không giải thích được của mạng: Đây là vấn đề quan trọng nhất của NN. Khi NN tạo ra một giải pháp thăm dò, nó không đưa ra manh mối về lý do tại sao và như thế nào. Điều này làm giảm sự tin tưởng vào mạng.
- Xác định cấu trúc mạng thích hợp: Không có quy tắc cụ thể nào để xác định cấu trúc của mạng noron nhân tạo. Cấu trúc mạng phù hợp đạt được thông qua kinh nghiệm và thử nghiệm và sai sót.
- Khó khăn khi hiển thị sự cố với mạng: NN có thể hoạt động với thông tin số. Các vấn đề phải được dịch thành các giá trị số trước khi được đưa vào NN. Cơ chế hiển thị được xác định ở đây sẽ ảnh hưởng trực tiếp đến hiệu suất của mạng. Điều này phụ thuộc vào khả năng của người dùng.
- Thời hạn của mạng không xác định: Mạng được giảm đến một giá trị lỗi nào đó trên mẫu có nghĩa là quá trình đào tạo đã được hoàn thành. Giá trị này không cho chúng ta kết quả tối ưu.

5. Ứng dụng của Mạng nơ-ron nhân tạo

Mạng nơ-ron nhân tạo được sử dụng rộng rãi ở nhiều lĩnh vực, có thể ứng dụng cho tài chính, lập kế hoạch doanh nghiệp, giao dịch, phân tích kinh doanh và bảo trì sản phẩm. Các ứng dụng phổ biến:

Tạo sử dụng trong các hoạt động kinh doanh như dự báo và tìm kiếm giải pháp nghiên cứu tiếp thị, phát hiện gian lận và đánh giá rủi ro.

Trong tài chính, hỗ trợ phát triển các quy trình như dự báo chuỗi thời gian, các giao dịch thuật toán, phân loại chứng khoán, các mô hình rủi ro tín dụng và xây dựng các chỉ báo độc quyền và các công cụ phái sinh giá cả.

Có thể đánh giá dữ liệu giá và khai quật các cơ hội giao dịch dựa trên phân tích dữ liệu lịch sử.

Có thể phân biệt sự phụ thuộc phi tuyến lẫn nhau của đầu vào mà các mô hình phân tích kỹ thuật khác không thể làm được.

Tuy nhiên, tính chính xác trong việc sử dụng mạng nơ-ron nhân tạo để đưa ra dự đoán giá cho cổ phiếu là khác nhau.

Một số mô hình dự đoán giá cổ phiếu chính xác 50 đến 60% thời gian, trong khi những mô hình khác có tính chính xác lên đến 70% trong tất cả các trường hợp… .

Chương II : Cơ sở lý thuyết

1. Ý tưởng thuật toán

Ý tưởng khởi nguồn của neural network là bắt bước cách thức hoạt động từ não bộ con người. Nói cách khác, mạng nơ ron nhân tạo được xem là hệ thống của các tế bào thần kinh nhân tạo. Đây thường có thể là hữu cơ hoặc nhân tạo về bản chất.

Các bài toán cơ bản được giải quyết bởi neural network:

- Bài toán phân lớp: Loại bài toán này đòi hỏi giải quyết vấn đề phân loại các đối tượng quan sát được thành các nhóm dựa trên các đặc điểm của các nhóm đối tượng đó. Đây là dạng bài toán cơ sở của rất nhiều bài toán trong thực tế: nhận dạng chữ viết, tiếng nói, phân loại gen, phân loại chất lượng sản phẩm, …
- Bài toán dự báo: Mạng nơron nhân tạo đã được ứng dụng thành công trong việc xây dựng các mô hình dự báo sử dụng tập dữ liệu trong quá khứ để dự đoán số liệu trong tương lai. Đây là nhóm bài toán khó và rất quan trọng trong nhiều ngành khoa học.
- Bài toán điều khiển và tối ưu hoá: Nhờ khả năng học và xấp xỉ hàm mà mạng nơron nhân tạo đã được sử dụng trong nhiều hệ thống điều khiển tự động cũng như góp phần giải quyết những bài toán tối ưu trong thực tế.

2. Trọng số (Weight - w)

Weight hay trọng số (w) là con số biểu thị mức độ quan trọng của ngõ vào so với ngõ ra. Giá trị ngõ ra của Perceptron phụ thuộc vào tổng giữa trọng số và ngõ vào:

$$\text{ngõ ra} = \begin{cases} 0 \text{ nếu } \sum_j x_j w_j \leq \text{ngưỡng} \\ 1 \text{ nếu } \sum_j x_j w_j > \text{ngưỡng} \end{cases}$$

Trong đó

- x_j : ngõ vào thứ j của Perceptron.
- w_j : trọng số của ngõ vào x_j .
- Ngưỡng : mức ngưỡng quyết định giá trị ngõ ra. Hàm xác định ngõ ra ở trên còn được gọi là hàm step… .

3. Bias - b và Variance

Bias: nghĩa là độ lệch, biểu thị sự chênh lệch giữa giá trị trung bình mà mô hình dự đoán và giá trị thực tế của dữ liệu. Để đơn giản cho perceptron quyết định giá trị đầu ra 0

hay 1, có thể thay thế giá trị ngưỡng (threshold) bằng nghịch đảo số Bias hay ngưỡng = -b.

$$\text{ngõ ra} = \begin{cases} 0 \text{ nếu } \sum_j x_j w_j + b \leq 0 \\ 1 \text{ nếu } \sum_j x_j w_j + b > 0 \end{cases}$$

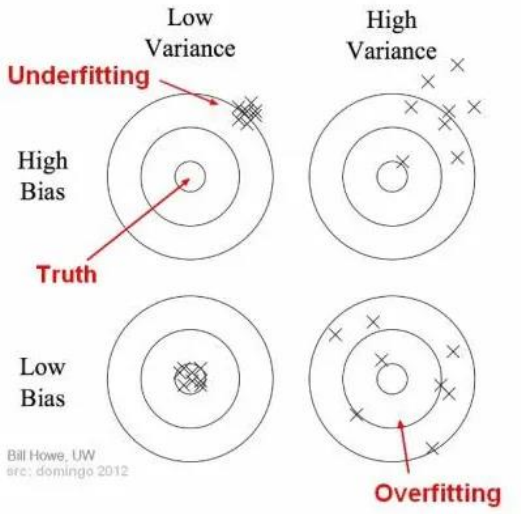
Bias có những đặc tính sau:

- Bias chỉ giống như một số chặn được thêm vào trong một phương trình tuyến tính.
- Đây là một tham số bổ sung trong mạng thần kinh được sử dụng để điều chỉnh đầu ra cùng với tổng trọng số của các đầu vào cho nơ-ron.
- Độ chệch hoạt động giống như một hằng số giúp mô hình phù hợp với dữ liệu đã cho.

Variance: nghĩa là phương sai, biểu thị độ phân tán của các giá trị mà mô hình dự đoán so với giá trị thực tế.

Dựa vào hình 1. , giá trị thật dữ liệu (ground truth) ở giữa tâm các đường tròn. Các dấu X là các giá trị dự đoán. Ta thấy nếu high bias thì giá trị dự đoán rất xa giá trị thực, việc này tương đối dễ xử lý khi chỉ cần điều chỉnh lại.

Tuy nhiên nếu high variance thì các giá trị dự đoán phân tán rộng dẫn đến việc ra giá trị thực tế sai lệch.



Hình 1. So sánh việc lựa chọn giá trị của độ lệch và phương sai

4. Loss Function

Khái quát : loss function hay còn gọi là hàm mất mát, thể hiện một mối quan hệ giữa y* (là kết quả dự đoán của model) và y (là giá trị thực tế).

Người ta đưa vào hàm loss function này mục đích là để tối ưu model của mình sao cho tốt nhất, hay cũng dùng để đánh giá độ tốt của model , y* (là kết quả dự đoán của

model) càng gần y (là giá trị thực tế) thì càng tốt. Tức là dựa vào loss function, khi đó chúng ta có thể tính ra gradient descent để tối ưu loss function càng về gần 0 càng tốt.

5. Gradient Descent

Gradient descent là thuật toán tìm giá trị nhỏ nhất của hàm số f(x) dựa trên đạo hàm.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. Nhìn chung, việc tìm global minimum của các hàm mất mát trong Machine Learning là rất phức tạp, thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm local minimum, và ở một mức độ nào đó, coi đó là nghiệm cần tìm của bài toán.

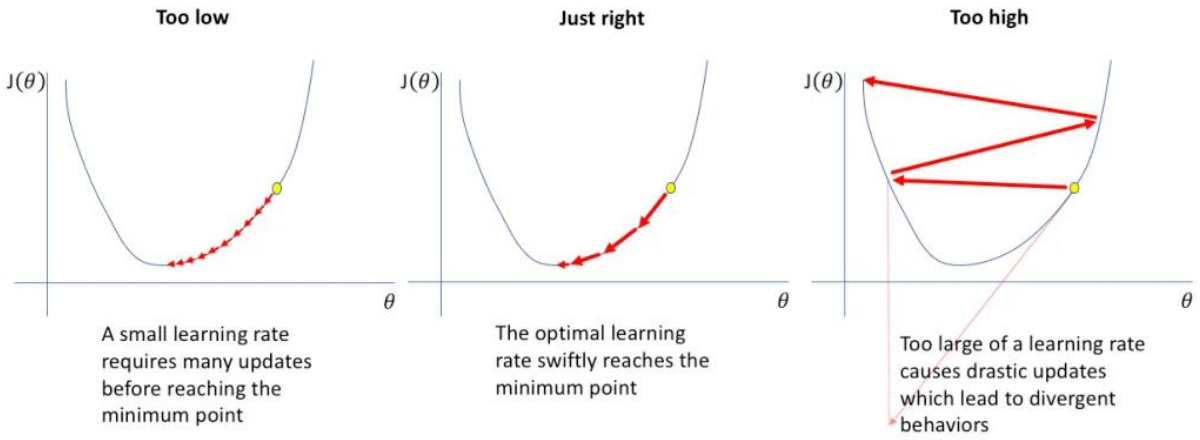
Các điểm local minimum là nghiệm của phương trình đạo hàm bằng 0. Nếu bằng một cách nào đó có thể tìm được toàn bộ (hữu hạn) các điểm cực tiểu, ta chỉ cần thay từng điểm local minimum đó vào hàm số rồi tìm điểm làm cho hàm có giá trị nhỏ nhất (*đoạn này nghe rất quen thuộc, đúng không?*). Tuy nhiên, trong hầu hết các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân có thể đến từ sự phức tạp của dạng của đạo hàm, từ việc các điểm dữ liệu có số chiều lớn, hoặc từ việc có quá nhiều điểm dữ liệu.

Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là *gần* với nghiệm của bài toán, sau đó dùng một phép toán lặp để *tiến dần* đến điểm cần tìm, tức đến khi đạo hàm gần với 0. Gradient Descent (viết gọn là GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

Việc chọn hệ số learning_rate cực kì quan trọng, có 3 trường hợp:

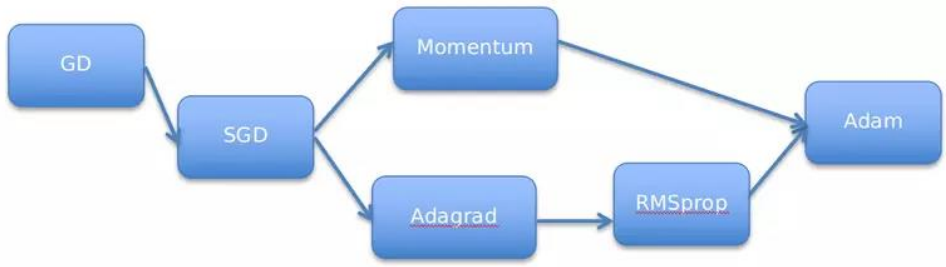
- Nếu learning_rate nhỏ: mỗi lần hàm số giảm rất ít nên cần rất nhiều lần thực hiện bước 2 để hàm số đạt giá trị nhỏ nhất làm tăng thời gian quá trình huấn luyện
- Nếu learning_rate hợp lý: sau một số lần lặp bước 2 vừa phải thì hàm sẽ đạt giá trị đủ nhỏ.
- Nếu learning_rate quá lớn: sẽ gây hiện tượng overshoot và không bao giờ đạt được giá trị nhỏ nhất của hàm.

Ngoài ra để thời gian hội tụ nhanh hơn thì người ta còn cài vào các cơ chế gọi là momentum nó làm các giá trị trên biểu đồ không thể đột ngột tăng và giảm giá trị gây ra overshoot khi nó vượt qua điểm có đạo hàm bằng 0.

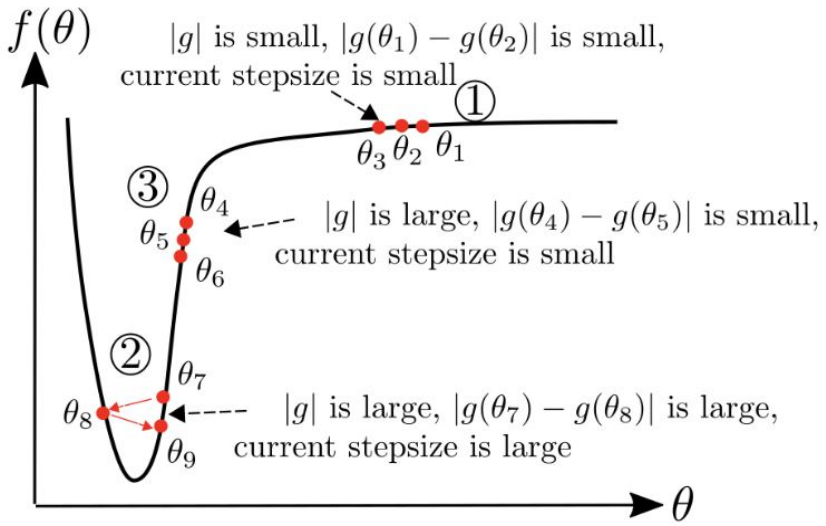


Hình 2. Learning rate state

6. Thuật toán tối ưu hóa Adabelief Optimizer



Hình 3. Cây phả hệ của các thuật toán tối ưu



Hình 4. Minh họa thuật toán Adam

Adam optimizer là một thuật toán kết hợp kỹ thuật của RMS prop và momentum. Thuật toán sử dụng hai internal states momentum (m) và squared momentum (v) của gradient cho các tham số. Sau mỗi batch huấn luyện, giá trị của m và v được cập nhật lại sử dụng exponential weighted averaging.

Mã giải của việc cập nhật m và v

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) g_t .$$

$$v_t = \beta_2 * m_{t-1} + (1 - \beta_2) g_t^2 .$$

trong đó, beta được xem như là một siêu tham số. Công thức cập nhật theta như sau:

$$\theta_t = \theta_{t-1} - a \frac{m_t}{\sqrt{v_t} + \epsilon} .$$

trong đó, alpha là learning rate, epsion là giá trị được thêm vào để ngăn việc chia cho 0.

Để việc descent được thực hiện nhanh hơn, thuật toán đã sử dụng hai kỹ thuật:

- Tính exponential moving average của giá trị đạo hàm lưu vào biến m và sử dụng nó là tử số của việc cập nhật hướng. Với ý nghĩa là nếu m có giá trị lớn, thì việc descent đang đi đúng hướng và chúng ta cần bước nhảy lớn hơn để đi nhanh hơn. Tương tự, nếu giá trị m nhỏ, phần descent có thể không đi về hướng tối tiểu và chúng ta nên đi 1 bước nhỏ để thăm dò. Đây là phần momentum của thuật toán.
- Tính exponential moving average của bình phương giá trị đạo hàm lưu vào biến v và sử dụng nó là phần mẫu số của việc cập nhật hướng. Với ý nghĩa như sau: Giả sử gradient mang các giá trị dương, âm lẫn lộn, thì khi cộng các giá trị lại theo công thức tính m ta sẽ được giá trị m gần số 0. Do âm dương lẫn lộn nên nó bị triệt tiêu lẫn nhau. Nhưng trong trường hợp này thì v sẽ mang giá trị lớn. Do đó, trong trường hợp này, chúng ta sẽ không hướng tới cực tiểu, chúng ta sẽ không muốn đi theo hướng đạo hàm trong trường hợp này. Chúng ta để v ở phần mẫu vì khi chia cho một giá trị cao, giá trị của các phần cập nhật sẽ nhỏ, và khi v có giá trị thấp, phần cập nhật sẽ lớn. Đây chính là phần tối ưu RMSProp của thuật toán.

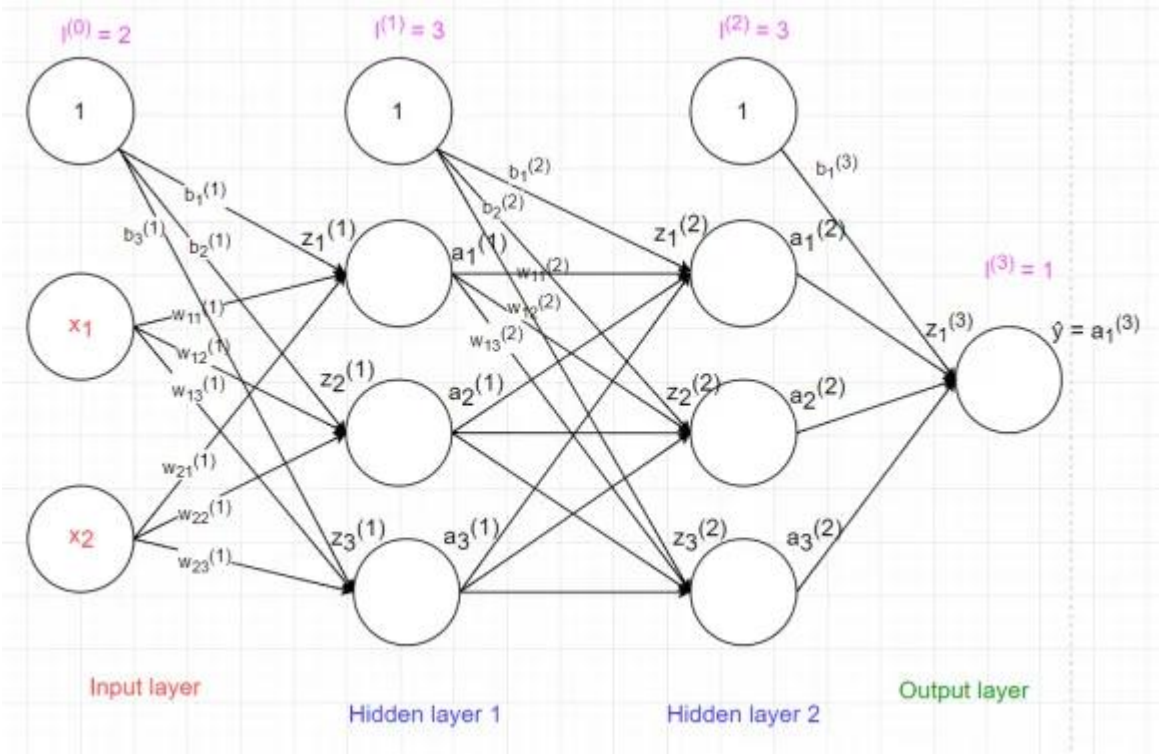
Ở đây, m được xem như là momentum thứ nhất, v xem như là momentum thứ hai, nên thuật toán có tên là "Adaptive moment estimation".

Để lý giải vì sao Adam lại hội tụ nhanh hơn so với SGD, chúng ta có thể giải thích như sau: Exponential weighted averaging cho chúng ta giá trị xấp xỉ gradient mượt hơn

qua mỗi lần lặp, dẫn tới tăng tính dừng. Sau đó, việc chia cho căn bậc 2 của giá trị v làm số ước của chúng ta giảm mạnh khi phương sai của giá trị gradient tăng lên. Điều này , như giải thích ở trên, có nghĩa là, khi hướng đi của mô hình chỉ ra không rõ ràng, thuật toán Adam thực hiện các bước đi nhỏ coi như là thăm dò thôi. Và sẽ thực hiện các bước đi lớn, nhanh khi hướng đi rõ ràng.

7. FeedForward

Feedforward Neural Networks hay tên gọi khác là Multi-layer Perceptrons. Một cấu trúc mà ở đó tất cả các các nốt trước dùng để tính các nốt sau với các trọng số và bias khác nhau.



Hình 5. Neural Network 2 Hidden Layer

Tại node thứ 2 ở layer 1, ta có:

- $z_2^{(1)} = x_1 * w_{12}^{(1)} + x_2 * w_{22}^{(1)} + b_2^{(1)}$.
- $a_2^{(1)} = \sigma\ (z_2^{(1)})$.

Hay ở node thứ 3 layer 2, ta có:

- $z_3^{(2)} = a_1^{(1)} * w_{13}^{(2)} + a_2^{(1)} * w_{23}^{(2)} + a_3^{(1)} * w_{33}^{(2)} + b_3^{(2)}$.
- $a_3^{(2)} = \sigma(z_3^{(2)})$.

Viết lại dưới dạng ma trận trọng số:

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right)$$

Dạng phương trình tổng quát cách tính giá trị neuron:

$$a^{(1)} = \sigma\left(W\, a^{(0)} + b\right).$$

8. Hàm kích hoạt

Hàm kích hoạt (activation function) mô phỏng tỷ lệ truyền xung qua axon của một neuron thần kinh. Trong một mạng nơ-ron nhân tạo, hàm kích hoạt đóng vai trò là thành phần phi tuyến tại output của các nơron, một hàm kích hoạt là một hàm ánh xạ các đầu vào của một nút với đầu ra tương ứng của nó.

Nếu không có các hàm kích hoạt phi tuyến, thì mạng nơ-ron của chúng ta dù có nhiều lớp vẫn sẽ có hiệu quả như một lớp tuyến tính mà thôi. Để hiểu rõ hơn và trực quan hơn, ta sẽ xem xét một mạng nơ-ron nhỏ gồm 2 lớp (có các hàm kích hoạt tuyến tính tương ứng). Mạng này sẽ nhận vào input là X và trả ra output : F(x)→y.

Trong lớp thứ nhất, ta có trọng số , hệ số bias B^{[1]}. Output Z^{[1]} được tính như sau:

$$Z^{(1)} = W^{(1)} * X + B^{(1)}.$$

Sau đó, output được đẩy qua hàm kích hoạt tuyến tính $g(\ \ )$ thu được kết quả là đầu ra của lớp thứ nhất:

$$a^{(1)} = g\left(Z^{(1)}\right).$$

Tương tự như vậy, output lại trở thành đầu vào của lớp thứ 2:

$$Z^{(2)} = W^{(2)} * a^{(1)} + B^{(2)}.$$

Vì ta đang giả sử hàm kích hoạt g() là tuyến tính nên $a^{(1)} = c * Z^{(1)}$ (c là một số thực). Giả sử c = 1. Sử dụng các công thức bên trên, ta có:

$$Z^{(2)} = W^{(2)} * Z^{(1)} + B^{(2)}.$$
$$\Rightarrow Z^{(2)} = W^{(2)} * \left(W^{(1)} * X + B^{(1)}\right) + B^{(2)}.$$

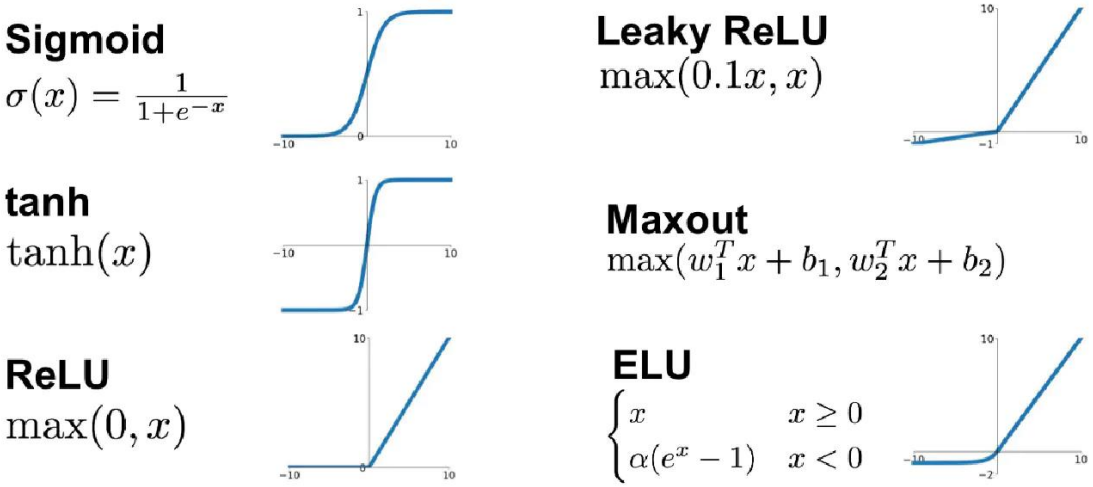
Công thức có thể viết dưới dạng:

$Z^{(2)} = W'X + B'. \Rightarrow a^{(2)} = g\left(Z^{(2)}\right) = c'W'X + c'B' = W''X + B''.$

—> Dễ dàng nhận thấy $a^{(2)}$ chỉ là một tuyến tính của X ban đầu thế nên việc xếp chồng neuron chỉ là vô nghĩa.

Để cho đơn giản, ta giả sử c = 1. Sử dụng các công thức bên trên, ta có:

Các hàm kích hoạt phổ biến:



Hình 6. Các hàm kích hoạt phổ biến

Chúng tôi lựa chọn hàm kích hoạt ReLu (Rectified Linear Unit) bởi vì chúng tôi nhận thấy:

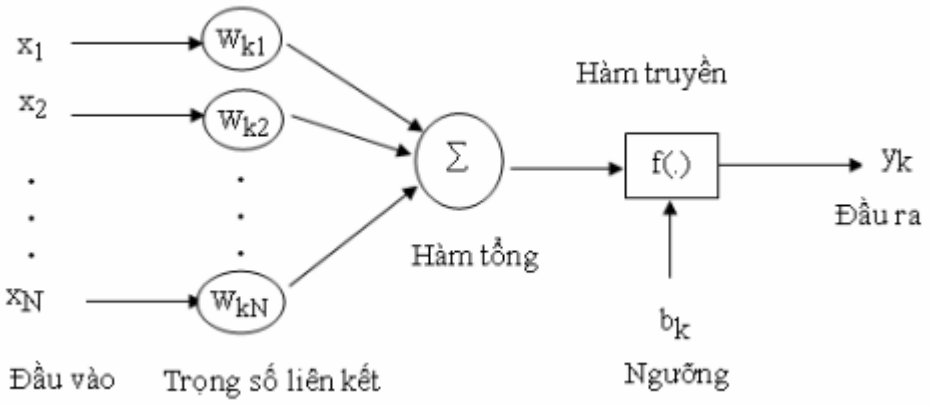
- Hàm ReLU đang được sử dụng khá nhiều trong những năm gần đây khi huấn luyện các mạng neuron. ReLU đơn giản lọc các giá trị < 0. Nhìn vào công thức chúng ta dễ dàng hiểu được cách hoạt động của nó. Một số ưu điểm khá vượt trội của nó so với Sigmoid và Tanh.
- Tốc độ hội tụ nhanh hơn hẳn. ReLU có tốc độ hội tụ nhanh gấp 6 lần Tanh (Krizhevsky et al.). Điều này có thể do ReLU không bị bão hoà ở 2 đầu như Sigmoid và Tanh.
- Tính toán nhanh hơn. Tanh và Sigmoid sử dụng hàm exp và công thức phức tạp hơn ReLU rất nhiều do vậy sẽ tốn nhiều chi phí hơn để tính toán dẫn tới ReLU có thời gian huấn luyện thấp hơn.

Phương trình:

- $f(x) = 0 \text{ nếu } x < 0$
- $f(x) = x \text{ nếu } x \geq 0$

9. Cấu trúc nơron nhân tạo

Cấu trúc của một nơron nhân tạo tiêu biểu có các thành phần cơ bản như mô tả ở hình 7. .



Hình 7. Cấu trúc của nơ ron nhân tạo

Đầu vào cung cấp các tín hiệu vào (input signals) của nơron, các tín hiệu này thường được đưa vào dưới dạng một vector N chiều. Các liên kết: Mỗi liên kết được thể hiện bởi một trọng số (gọi là trọng số liên kết – Synaptic weight). Trọng số liên kết giữa tín hiệu vào thứ j với nơ ron k thường được kí hiệu là $\mathbf{w}_{k(j)}$.

Thông thường, các trọng số này được khởi tạo một cách ngẫu nhiên ở thời điểm khởi tạo mạng và được cập nhật liên tục trong quá trình huấn luyện mạng.

Hàm tổng (Summing function): Thường dùng để tính tổng của tích các đầu vào với trọng số liên kết của nó.

$$\sum_{i=1}^{m}(w_i x_i) + bias$$

Ngưỡng (còn gọi là một độ lệch - bias): Ngưỡng này thường được đưa vào như một thành phần của hàm truyền.

Hàm truyền (Transfer function): Hàm này được dùng để giới hạn phạm vi đầu ra của mỗi nơ ron. Nó nhận đầu vào là kết quả của hàm tổng và ngưỡng đã cho. Thông thường, phạm vi đầu ra của mỗi nơron được giới hạn trong đoạn [0,1] hoặc [-1, 1]. Các hàm truyền rất đa dạng, có thể là các hàm tuyến tính hoặc phi tuyến 3 . Một số hàm truyền thường sử dụng trong các mô hình mạng nơ ron gồm: Symmetrical Hard Limit (hardlims), Linear (purelin), Saturating Linear (satlin) và Log-Sigmoid (logsig).

$$f(x) = \begin{cases} 1 \text{ if } \Sigma wx + b \geq 0 \\ 0 \text{ if } \Sigma wx + b < 0 \end{cases}$$

Đầu ra (output) Là tín hiệu đầu ra của một nơron, với mỗi nơron sẽ có tối đa là một đầu ra.

Như vậy tương tự như nơron sinh học, nơ ron nhân tạo cũng nhận các tín hiệu đầu vào, xử lý (nhân các tín hiệu này với trọng số liên kết, tính tổng các tích thu được rồi gửi kết quả tới hàm truyền), và tái tạo tín hiệu đầu ra (là kết quả của hàm truyền).

Cấu trúc mạng nơ ron nhân tạo Một mạng nơ ron có thể gồm 1 hoặc nhiều nơron 4 . Mỗi nơ ron là một đơn vị xử lý thông tin, sự liên kết giữa các nơron tạo thành cấu trúc mạng. Mặc dù mỗi nơron đơn lẻ có thể thực hiện những chức năng xử lý thông tin nhất định, sức mạnh của tính toán nơron chủ yếu có được nhờ sự kết hợp các nơ ron trong một kiến trúc thống nhất. Một mạng nơ ron là một mô hình tính toán được xác định qua các tham số: kiểu nơ ron (như là các nút nếu ta coi cả mạng nơ ron là một đồ thị), kiến trúc kết nối (sự tổ chức kết nối giữa các nơron) và thuật toán học (thuật toán dùng để học cho mạng). Các nơron kết nối với nhau bằng ma trận trọng số. Cách thức kết nối các nơron trong mạng xác định cấu trúc (topology) của mạng, vì vậy, có nhiều cấu trúc mạng khác nhau.

Một mạng nơ-ron gồm những cấu phần sau:

- Dữ liệu đầu vào – **input layer**, $\boldsymbol{x}$
- Lớp ẩn – **hidden layers**
- Dữ liệu đầu ra – **output layer**, $\hat{\boldsymbol{y}}$
- Các tham số trọng lượng tương ứng $\boldsymbol{W}$ và ngưỡng quyết định $\boldsymbol{b}$
- Hàm ánh xạ cho lớp ẩn $\boldsymbol{\sigma}$.

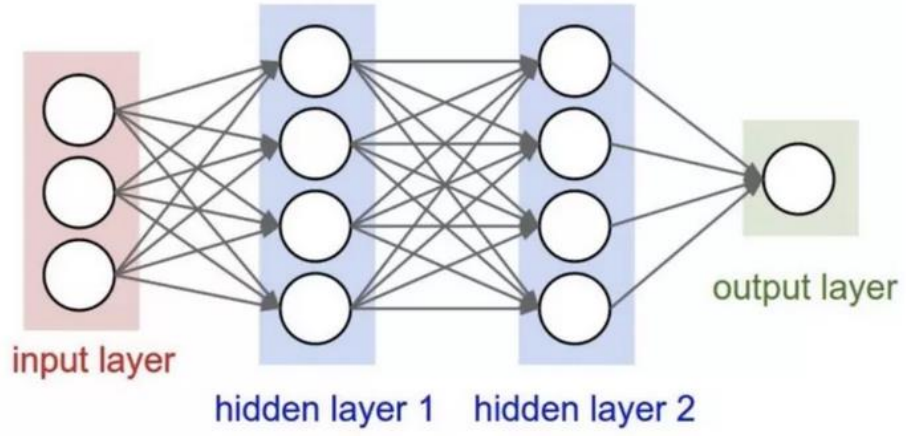
Lớp đầu vào gồm một hay nhiều biến số đầu vào, cung cấp thông tin cho mạng nhân tạo. Đối với dự báo lạm phát và tăng trưởng kinh tế, các biến này có thể là: Tăng trưởng lượng cung tiền thực tế M2, Tỷ giá, tăng trưởng sản xuất công nghiệp... và chính số liệu lịch sử của lạm phát và tăng trưởng kinh tế.

Lớp đầu ra có thể gồm một hay nhiều biến số đầu ra. Trong nghiên cứu này lớp đầu ra có thể là một biến riêng lẻ, hoặc Lạm phát hoặc tăng trưởng GDP hoặc là một tổ hợp biến. Sự liên kết giữa các yếu tố đầu vào và đầu ra được tính toán qua lớp ẩn trung gian với một hệ thống các hàm truyền và ngưỡng.

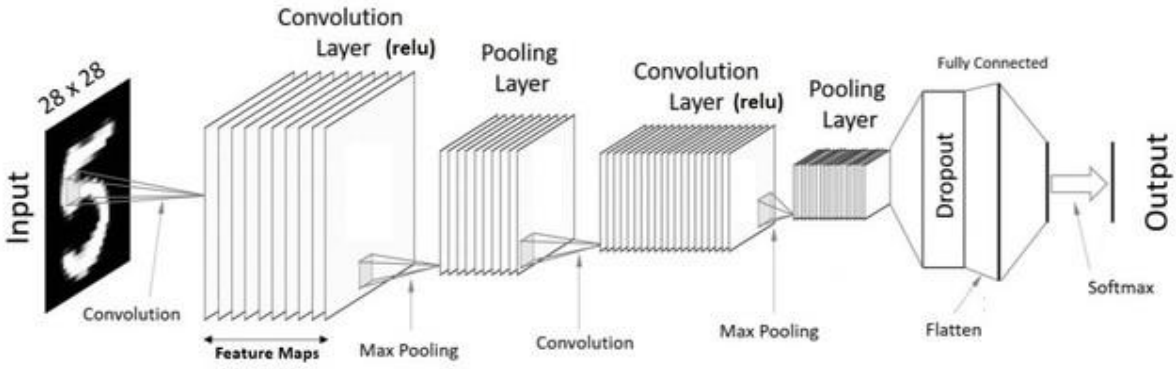
10. Phân loại Neural Network

Dựa trên tính chất kết nối giữa các nơron đầu ra tới các nơron đầu vào, mạng được chia thành hai kiến trúc chính:

- Artificial Neural Network (ANN)
- Convolutional Neural Network (CNN)



Hình 8. Mạng nơron nhân tạo ANN



Hình 9. Mạng nơron tích chập CNN

Chương III: Bài toán nhận diện số bằng Artificial Neural Network

1. Tổng quan

Đây là loại bài toán phân lớp: Loại bài toán này đòi hỏi giải quyết vấn đề phân loại các đối tượng quan sát được thành các nhóm dựa trên các đặc điểm của các nhóm đối tượng đó. Đây là dạng bài toán cơ sở của rất nhiều bài toán trong thực tế: nhận dạng chữ viết, tiếng nói, phân loại gen, phân loại chất lượng sản phẩm, … .

2. Cấu trúc

Thiết lập hai lớp hidden trong đó lớp thứ nhất làm các nơron nhận biết bộ phận của các nét và lớp thứ hai nhận biết các nét hoàn chỉnh.

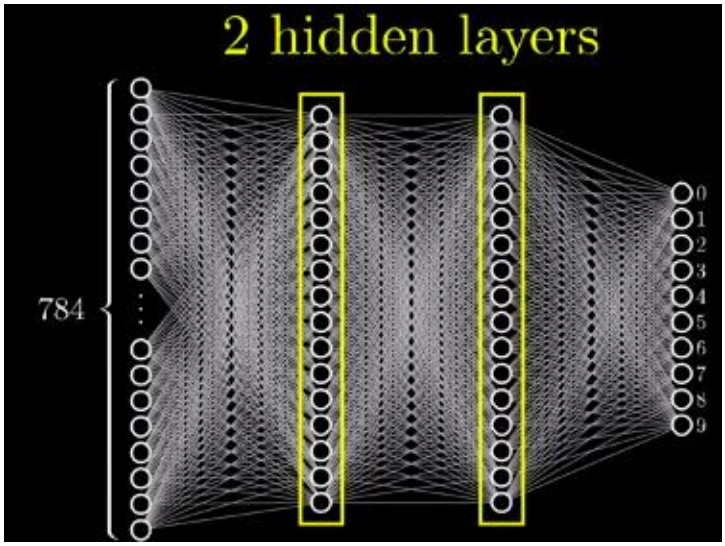
Input nhập vào một hình ảnh, nó được chuyển đổi thành dạng ma trận trọng số sau đó qua quá trình thụ cảm và lan truyền dữ liệu của hai lớp hidden mà xác định được hình ảnh chứa các nét nào và nó thuộc về số nào.

Output là một tập các kết quả từ 0 đến 9 với các neuron có giá trị khác nhau thể hiện các số được vẽ ở input thuộc về số nào ở output. Các neuron có trọng số càng lớn thì khả năng số đó là số cần nhận biết càng cao.

Với input 784 pixel, 16 bộ phận nét, 16 nét hoàn chỉnh và 10 output (0,9)

Số lượng Weight sẽ cần là 784 x 16 + 16x16 + 16x10 = 12960.

Số lượng Bias cần có: 16+16+10 = 42.



Hình 10. Mô hình ANN đối với dữ liệu số

3. Thực hiện

a. Ma trận hóa các trọng số

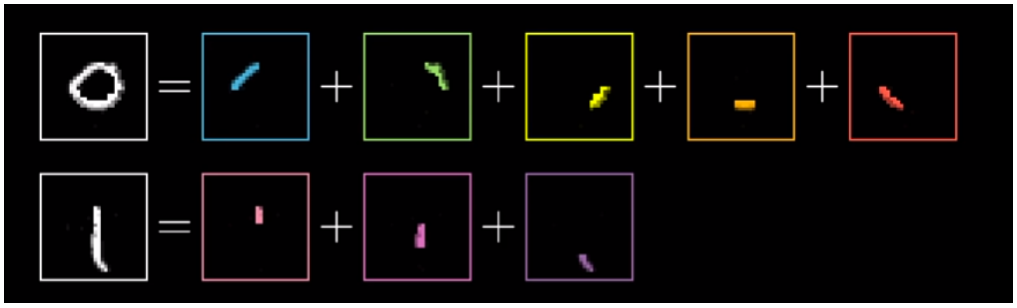
Chuyển đổi hình ảnh thành dãy trọng số có giá trị từ 0 đến 1 tương ứng với giá trị độ sáng của nó. Dãy có chiều dài bằng tích chiều dài và rộng của hình ảnh. Với ảnh kích

thước 28*28 pixel, các số được vẽ trên ma trận số được chuyển thành các mảng input dài 784 phần tử.

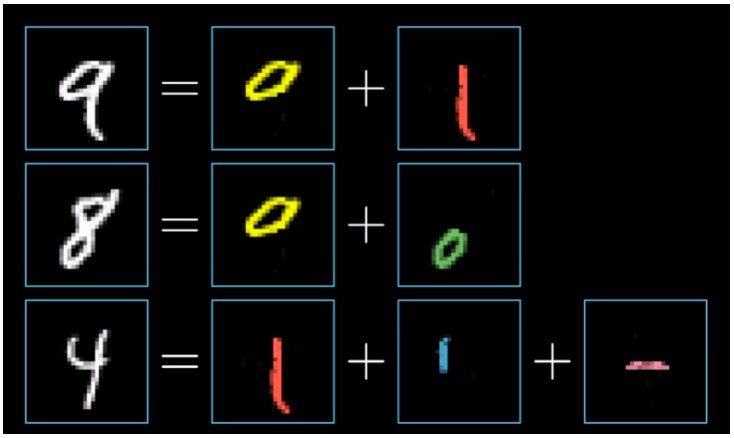
b. Quá trình tiếp nhận và lan truyền tiến của neuron

Giá trị neuron nằm trong 0 tới 1. Đại diện cho khoảng giá trị giữa có và không. Giá trị càng cao thì tiên đoán sẽ càng thiên về thông tin của neuron đó. Những trọng số đi qua các lớp nơron được lập trình để phát hiện ra các yếu tố đặc trưng. Mỗi neuron giữ một điều kiện kích hoạt đặc trưng, mỗi neuron từ lớp trước đẩy giá trị nó thu được sang lớp tiếp theo, qua đó các đặc trưng số dần được nhận diện và giúp hệ thống đoán được chữ số mà ma trận trọng số đang thể hiện.

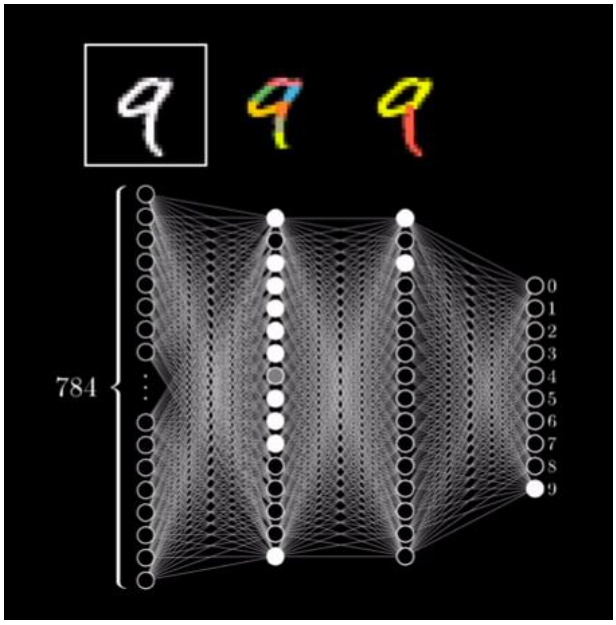
Với ví dụ 2 lớp hidden network, ở lớp đầu tiên (hình 11.) chứa các mảnh của 1 nét, giúp nhận biết các input đang thể hiện nét nào ở lớp thứ 2 . Lớp thứ 2 (hình 12.) chứa các nét hoàn chỉnh sẽ chỉ ra các nét đó thuộc chữ số nào ở output như hình 13. . Có rất nhiều các neuron sẽ được kích hoạt ở hidden network nhưng mỗi neutron đều nhận các giá trị kích hoạt khác nhau nên các neuron cũng có mức ảnh hưởng khác nhau, cuối cùng chỉ có những nốt ở output có giá trị cao nhất mới được chọn để thể hiện được kết quả cuối cùng.



Hình 11. Thể hiện cấu tạo nét của từng số ở lớp 1



Hình 12. Thể hiện cấu tạo nét của từng số ở lớp 2



Hình 13. Kết quả phán đoán thông qua 2 lớp mạng

c. Các công thức toán học:

Phương trình tổng quát cho 1 neuron ở lớp hidden 1:

$$a_0^{(1)} = \sigma\left(w_{0,0} * a_0^{(0)} + w_{0,1} * a_1^{(0)} + \ldots + w_{0,n} * a_n^{(0)} + Bias\right) .$$

Dãy hidden 1**:** Công thức tính giá trị cho các neuron:

$$\sigma(w_1 * a_1 + w_2 * a_2 + w_3 * a_3 + \ldots + w_n * a_n).$$

Sau đó, chúng ta thêm Bias vào để điều chỉnh độ lệch.

$$\sigma(w_1 * a_1 + w_2 * a_2 + w_3 * a_3 + \ldots + w_n * a_n - 10).$$

Ý nghĩa của hàm trên:

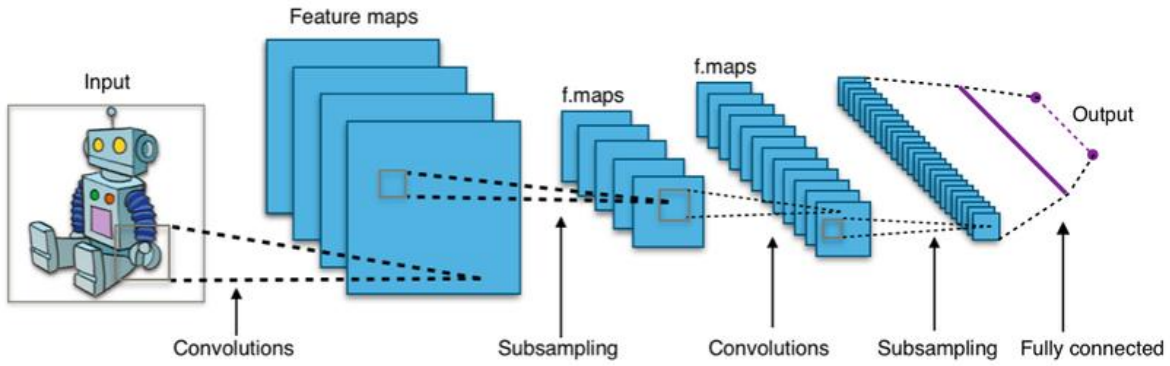
- Bias là cố định, trọng số weight luôn thay đổi.
- Bias giúp lọc các nhiễu và làm vô hiệu hóa những neuron quá yếu

Dãy hidden 2: Tương tự, sử dụng công thức ở hidden 1, để tính giá trị neuron ở hidden 2 với đầu vào là giá trị các neuron ở hidden 1.

Chương IV: Bài toán nhận diện số bằng Convolutional Neural Network

1. Tổng quan

Convolutional Neural Network (CNNs – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay. CNN được sử dụng nhiều trong các bài toán nhận dạng các object trong ảnh.



Hình 14. Mô hình CNN

Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo. Các layer liên kết được với nhau thông qua cơ chế convolution. Layer tiếp theo là kết quả convolution từ layer trước đó, nhờ vậy mà chúng ta có được các kết nối cục bộ. Như vậy mỗi neuron ở lớp kế tiếp sinh ra từ kết quả của filter áp đặt lên một vùng ảnh cục bộ của neuron trước đó.

Mỗi một lớp được sử dụng các filter khác nhau thông thường có hàng trăm hàng nghìn filter như vậy và kết hợp kết quả của chúng lại. Ngoài ra có một số layer khác như pooling/subsampling layer dùng để chắt lọc lại các thông tin hữu ích hơn (loại bỏ các thông tin nhiễu).

Trong mô hình CNN có 2 khía cạnh cần quan tâm là tính bất biến (Location Invariance) và tính kết hợp (Compositionality). Với cùng một đối tượng, nếu đối tượng này được chiếu theo các góc độ khác nhau (translation, rotation, scaling) thì độ chính xác của thuật toán sẽ bị ảnh hưởng đáng kể. Pooling layer sẽ cho bạn tính bất biến đối với phép dịch chuyển (translation), phép quay (rotation) và phép co giãn (scaling). Tính kết hợp cục bộ cho ta các cấp độ biểu diễn thông tin từ mức độ thấp đến mức độ cao và trừu tượng hơn thông qua convolution từ các filter.

2. Tại sao dùng CNN

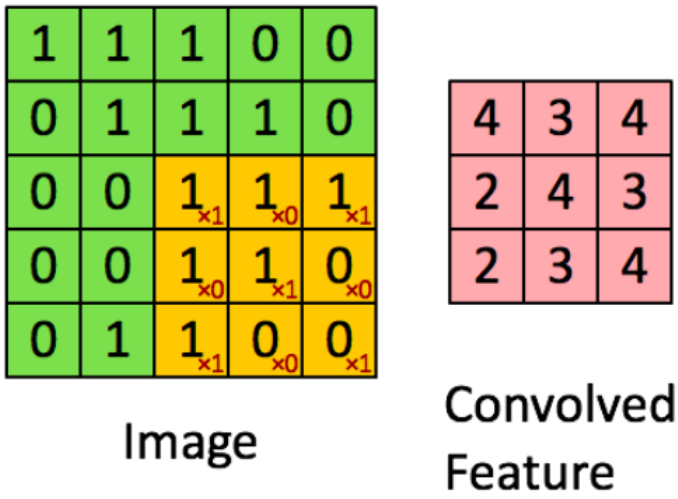
Thông thường chúng ta xử lý các dữ liệu dạng đơn giản như ANN có hình ảnh 28x28 pixels thì số lượng Weight là 12960 , đối với xử lý các ảnh lớn hơn như 200x200 pixel thì số lượng Weight cần có là 640416, gấp xấp xỉ 50 lần sẽ làm cho khối lượng tính toán rất là lớn. Vì vậy chúng ta cần CNN cho những hình ảnh có kích thước lớn.

3. Cấu trúc

Convolutional Neural Network có hai phần chính:

- Lớp trích lọc đặc trưng của ảnh (Convolution, ReLU và Pool).
- Lớp phân loại (Fully Connected và softmax).

a. Convolution Layer



Hình 15. Nhân ma trận

Convolution là một cửa sổ trượt (Sliding Windows) trên một ma trận như mô tả hình 15. Ví dụ ta có ảnh đầu vào là 5x5 pixels, cái khung màu vàng (Sliding Windows) sẽ trượt từ trên xuống trái sang phải, từ trên xuống dưới đề nhân ma trận giữa các phần tử để tạo ra đặc trưng của ảnh Convolved Feature. Với khung màu vàng ở hình 15. ta tính như sau:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = 1 \cdot 1 + 1 \cdot 0 + \ldots + 0 \cdot 1 = 4.$$

Với ma trận đầu tiên là của hình ảnh đầu vào, ma trận kế tiếp được gọi kernel, filter hoặc feature detect là một ma trận có kích thước nhỏ như trong ví dụ trên là 3×3. là do được học khi train mô hình CNN làm sao để cho cái.

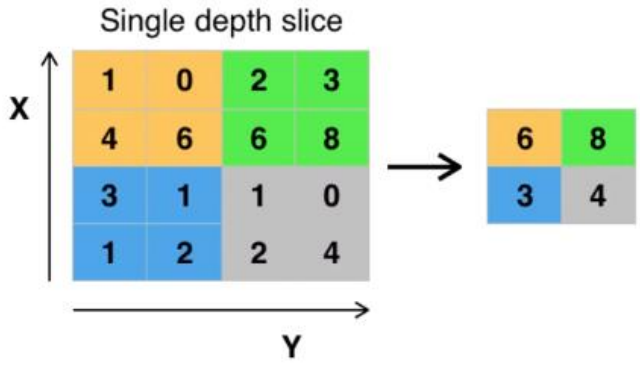
b. ReLU Layer

ReLU áp dụng các kích hoạt lên đầu ra của Convolution Layer (phần màu hồng của hình 15.), có tác dụng đưa các giá trị âm về 0 tránh ảnh hưởng việc tính toán cho các Layer sau đó.

Layer này đảm bảo các giá trị nonlinear (không tuyến tính) qua từng Layer.

c. Pooling Layer

Pooling Layer có tác dụng giảm chiều không gian của đầu vào và giảm độ phức tạp tính toán. Giúp cho các hình rất to nó sẽ nhỏ đi mà vẫn giữ lại được nhưng chi tiết quan trọng. Có hai loại pooling hay dùng là max pooling và average pooling.



Hình 16. Pooling layer

d. Fully Connected Layer và Softmax

- Là các kết nối đầy đủ như các mạng neural network bình thường.

- Input: để đưa các ảnh từ các Layer trước vào mạng này, buộc phải dàn phẳng bức ảnh ra thành 1 vector thay vì mảng nhiều chiều như trước. Tại Layer cuối cùng sẽ sử dụng 1 hàm softmax để phân loại đối tượng dựa vào vector đặc trưng đã được tính toán của các lớp trước đó.

e. Cách chọn tham số

- Số các convolution layer: càng nhiều các convolution layer thì performance càng được cải thiện. Sau khoảng 3 hoặc 4 layer, các tác động được giảm một cách đáng kể

- Filter size: thường filter theo size 5×5 hoặc 3×3

- Pooling size: thường là 2×2 hoặc 4×4 cho ảnh đầu vào lớn

- Cách cuối cùng là thực hiện nhiều lần việc train test để chọn ra được param tốt nhất.

Chương V: Xây dựng thuật toán với code python

1. Chuẩn bị

Ngôn ngữ lập trình	Python 3.9
Platform, library và framework	Tensorflow 2.0, Keras, Pygame, Matplotlib, Numpy, Tkinter
IDE	Spyder
Tập huấn luyện	Mnist
Hàm kích hoạt	ReLu, Softmax
Loss function	Categorical_crossentropy, Sparse_categorical_crossentropy
Optimizer	Adam

2. Cài đặt

Sử dụng tập huấn luyện **MNIST** là bộ cơ sở dữ liệu về chữ số viết tay, bao gồm 2 tập con: training set gồm 60.000 ảnh các chữ số viết tay và test set gồm 10.000 ảnh các chữ số.

- Train Model cho Artificial Neural Networks (ANN)

```
# 1. Thêm các thư viện cần thiết
import keras
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
mnist = tf.keras.datasets.mnist

# 2. Chuẩn hóa dữ liệu từ MNIST
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)

# 3. Chuyển đổi hình ảnh thành dãy trọng số có giá trị từ 0 đến 1 tương ứng với giá trị độ sáng của nó
for train in range(len(x_train)):
    for row in range(28):
```

```
        for x in range(28):
            if x_train[train][row][x] != 0:
                x_train[train][row][x] = 1

# 4. Thêm các hidden layer vào Model

# Định nghĩa model
model = tf.keras.models.Sequential()

# Flatten layer chuyển từ tensor sang vector
model.add(tf.keras.layers.Flatten())

# Thêm 2 Hidden layer với 128 nodes và dùng hàm ReLU
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))

# Output layer với 10 node và dùng softmax function để chuyển sang xác
xuất.
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

# 5. Compile model, sử dụng phương thức tối ưu Adam, loss function là
sparse_categorical_crossentropy
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 6. Thực hiện train model với data
model.fit(x_train, y_train, epochs=3)

# 7. Lưu model và in thông báo
model.save('epic_num_reader.model')
print("Model saved")

# 8. Đánh giá model với dữ liệu test set
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

- Train Model cho Convolutional Neural Networks (CNN)

```
# 1. Thêm các thư viện cần thiết
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils

# 2. Load data từ MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 3. Reshape dữ liệu thành dạng [samples][width][height][channels]
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1)).astype('float32')
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1)).astype('float32')

# 4. Chuẩn hóa inputs từ 0-255 thành 0-1
X_train = X_train / 255
X_test = X_test / 255

# 5. One-hot encoding chuyển đổi label của ảnh từ giá trị số sang vector
#cùng kích thước với output của model
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]

# 6. Định nghĩa larger model
def larger_model():
        # Tạo model
        model = Sequential()

        # Thêm Convolutional layer với 30 kernel, kích thước kernel 5*5
        #dùng hàm relu làm activation và chỉ rõ input_shape cho layer
        #đầu tiên
        model.add(Conv2D(30, (5, 5), input_shape=(28, 28, 1),
activation='relu'))

        # Thêm Max pooling layer
        model.add(MaxPooling2D())
```

```
        # Thêm Convolutional layer với 15 kernel, kích thước kernel 3*3
        #dùng hàm relu làm activation
        model.add(Conv2D(15, (3, 3), activation='relu'))

        # Thêm MaxPooling Layer
        model.add(MaxPooling2D())

        # Thêm Dropout layer để chống Over-fitting
        model.add(Dropout(0.2))

        # Thêm Flatten Layer chuyển từ tensor sang vector
        model.add(Flatten())

        # Thêm Fully Connected layer với 128 nodes và dùng hàm relu
        model.add(Dense(128, activation='relu'))

        # Thêm Fully Connected layer 2 với 50 nodes và dùng hàm relu
        model.add(Dense(50, activation='relu'))

        # Output layer dùng softmax function để chuyển sang xác xuất.
        model.add(Dense(num_classes, activation='softmax'))

        # Compile model sử dụng phương thức tối ưu Adam và loss
        #function là Categorical_crossentropy
        model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
        return model

# 7. Build model
model = larger_model()

# 8. Train model với data, 10 epochs
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,
batch_size=200)

# 9. Đánh giá model với dữ liệu test set và thông báo kết quả
scores = model.evaluate(X_test, y_test, verbose=0)
print("Large CNN Error: %.2f%%" % (100-scores[1]*100))

# 10. Lưu model vào mnist.h5 và thông báo
model.save('mnist.h5')
print("Saving the model as mnist.h5")
```

- Sử dụng Pygame để tạo giao diện cho người dùng vẽ số vào và đưa ra dự đoán

```
# Hàm dự đoán dựa vào model đã train ở trên và hiển thị ra messageBox
def guess(li):
    model = tf.keras.models.load_model('model')

    predictions = model.predict(li)
    print(predictions[0])
    t = (np.argmax(predictions[0]))
    print("I predict this number is a:", t)
    window = Tk()
    window.withdraw()
    messagebox.showinfo("Prediction", "I predict this number is a: " +
str(t))
    window.destroy()
```

Chương VI: Thực nghiệm và kết quả

Kết quả thực nghiệm

- ANN: Với 10 epoch ta thấy kết quả khi chạy thử với test set MNIST không cao, Test loss khoảng 0.178, Test accuracy khoảng 0.953, tỷ lệ lỗi khoảng 4.63%.

```
Epoch 1/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2479 - accuracy: 0.9244
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1148 - accuracy: 0.9632
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0841 - accuracy: 0.9730
Epoch 4/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0651 - accuracy: 0.9790
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0521 - accuracy: 0.9825
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0437 - accuracy: 0.9854
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0375 - accuracy: 0.9876
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0301 - accuracy: 0.9900
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0281 - accuracy: 0.9905
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0256 - accuracy: 0.9914
INFO:tensorflow:Assets written to: epic_num_reader.model\assets
Model saved
Test loss: 0.17838068306446075
Test accuracy: 0.9537000060081482
Large ANN Error: 4.63%
```

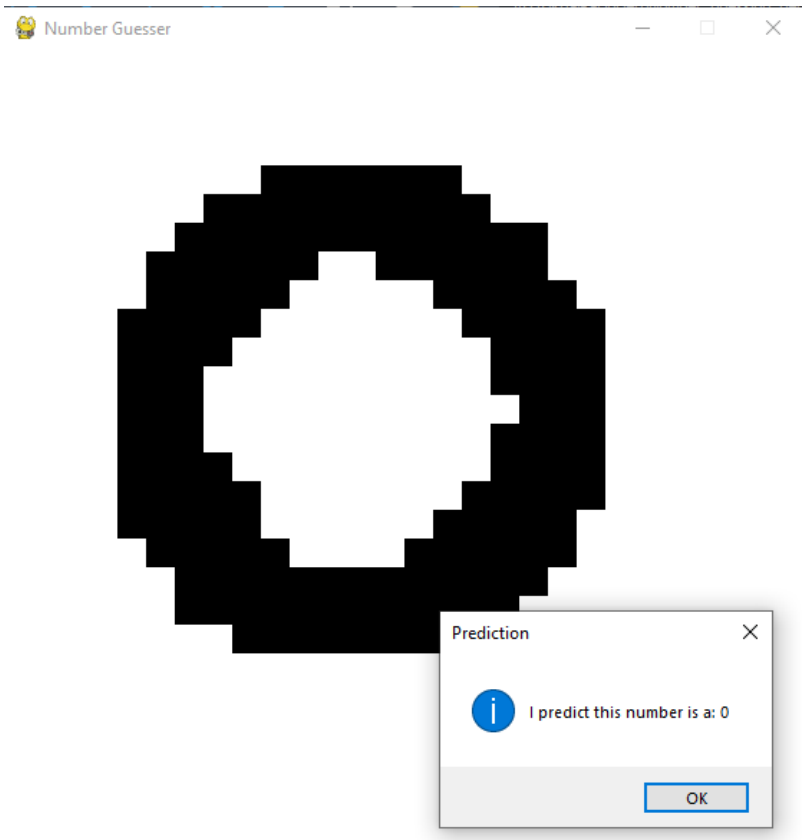
Hình 17. Kết quả training bằng ANN

- CNN: Với 10 epoch ta thấy kết quả khi chạy thử với test set MNIST cho ra khá cao, Test loss khoảng 0.03, test accuracy khoảng 0.9908 và tỷ lệ lỗi 0.92%.

```
300/300 [==============================] - 30s 99ms/step - loss: 0.3940 - accuracy: 0.8803 - val_loss: 0.0786 - val_accuracy: 0.9757
Epoch 2/10
300/300 [==============================] - 28s 93ms/step - loss: 0.0965 - accuracy: 0.9700 - val_loss: 0.0506 - val_accuracy: 0.9840
Epoch 3/10
300/300 [==============================] - 29s 96ms/step - loss: 0.0690 - accuracy: 0.9787 - val_loss: 0.0388 - val_accuracy: 0.9880
Epoch 4/10
300/300 [==============================] - 27s 91ms/step - loss: 0.0587 - accuracy: 0.9819 - val_loss: 0.0297 - val_accuracy: 0.9908
Epoch 5/10
300/300 [==============================] - 30s 100ms/step - loss: 0.0510 - accuracy: 0.9839 - val_loss: 0.0318 - val_accuracy: 0.9900
Epoch 6/10
300/300 [==============================] - 33s 111ms/step - loss: 0.0448 - accuracy: 0.9858 - val_loss: 0.0302 - val_accuracy: 0.9908
Epoch 7/10
300/300 [==============================] - 41s 136ms/step - loss: 0.0391 - accuracy: 0.9874 - val_loss: 0.0352 - val_accuracy: 0.9876
Epoch 8/10
300/300 [==============================] - 30s 101ms/step - loss: 0.0366 - accuracy: 0.9884 - val_loss: 0.0279 - val_accuracy: 0.9904
Epoch 9/10
300/300 [==============================] - 35s 116ms/step - loss: 0.0342 - accuracy: 0.9893 - val_loss: 0.0280 - val_accuracy: 0.9910
Epoch 10/10
300/300 [==============================] - 33s 110ms/step - loss: 0.0310 - accuracy: 0.9901 - val_loss: 0.0303 - val_accuracy: 0.9908
Test loss: 0.03031846322119236
Test accuracy: 0.9908000230789185
Large CNN Error: 0.92%
Saving the model as mnist.h5
```

Hình 18. Kết quả training bằng CNN

- Ảnh demo giao diện: cả 2 phần ANN và CNN đều sử dụng chung code giao diện



Hình 19. Ảnh demo giao diện dự đoán số

So sánh

	ANN	CNN
Tỉ lệ đúng	95,55%	99,08%
Tỷ lệ lỗi	4.63%	0.92%
Hình ảnh	Hình ảnh nhỏ	Hình ảnh lớn

Tài liệu tham khảo

1. *Artificial Neural Network Tutorial*. (n.d.). Javatpoint. Retrieved May 10, 2022, from https://www.javatpoint.com/artificial-neural-network
2. *Bài 3: Neural network*. (2019, March 9). Deep Learning cơ bản. Retrieved May 10, 2022, from https://nttuan8.com/bai-3-neural-network/
3. *Các hàm kích hoạt (activation function) trong neural network*. (2019, September 23). AICurious. Retrieved May 10, 2022, from https://aicurious.io/posts/2019-09-23-cac-ham-kich-hoat-activation-function-trong-neural-networks/
4. *Convolutional Neural Network (CNN) Tutorial*. (n.d.). Kaggle. Retrieved May 10, 2022, from https://www.kaggle.com/kanncaa1/convolutional-neural-network-cnn-tutorial
5. *Convolutional Neural Network là gì? Cách chọn tham số cho Convolutional Neural Network chuẩn chỉnh*. (2021, July 27). Tino Group. Retrieved May 10, 2022, from https://wiki.tino.org/convolutional-neural-network-la-gi/
6. *Deep Neural Network Keras way*. (n.d.). Kaggle. Retrieved May 10, 2022, from https://www.kaggle.com/poonaml/deep-neural-network-keras-way
7. *Deep Neural Networks*. (2022, May 5). Kaggle. Retrieved May 10, 2022, from https://www.kaggle.com/ryanholbrook/deep-neural-networks
8. *FabianGroeger96/cnn-number-detection: Number detection implemented using TensorFlow with custom CNN architecture for fast inference and custom dataset*. (n.d.). GitHub. Retrieved May 10, 2022, from https://github.com/FabianGroeger96/cnn-number-detection

9. Gorman, B. (2017, November 8). *Neural Networks – A Worked Example*. GormAnalysis. Retrieved May 10, 2022, from https://www.gormanalysis.com/blog/neural-networks-a-worked-example/
10. Guide, S. (2022, March 8). *An Ultimate Tutorial to Neural Networks in 2022*. Simplilearn. Retrieved May 10, 2022, from https://www.simplilearn.com/tutorials/deep-learning-tutorial/neural-network
11. *Hàm Kích Hoạt Trong Mạng Neural*. (n.d.). TEK4. Retrieved May 10, 2022, from https://tek4.vn/ham-kich-hoat-trong-mang-neural
12. *JackonYang/captcha-tensorflow: Image Captcha Solving Using TensorFlow and CNN Model. Accuracy 90%+*. (n.d.). GitHub. Retrieved May 10, 2022, from https://github.com/JackonYang/captcha-tensorflow
13. *Kyziridis/Neural-Network: Neural_Network for handwritten numbers detection in Octave (free Matlab)*. (n.d.). GitHub. Retrieved May 10, 2022, from https://github.com/Kyziridis/Neural-Network
14. *luke-dinh/Basic-Deep-Learning: (Written in Vienamese). A summary of some basic theory about Deep Learning: Artificial Neural Network, Backpropagation and Basic Convolutional Neural Network.* (n.d.). GitHub. Retrieved May 10, 2022, from https://github.com/luke-dinh/Basic-Deep-Learning
15. *Machine Learning cơ bản*. (2018, June 22). Machine Learning cơ bản. Retrieved May 10, 2022, from https://machinelearningcoban.com/2018/06/22/deeplearning/
16. *Mạng Neural: Giới Thiệu Và Ví Dụ – Tự Học TensorFlow*. (n.d.). TEK4. Retrieved May 10, 2022, from https://tek4.vn/mang-neural-gioi-thieu-va-vi-du-tu-hoc-tensorflow

17. *Mạng Nơron nhân tạo hướng tiếp cận mới trong công tác nhận dạng và dự báo tài nguyên nước*. (n.d.). nawapi. Retrieved May 10, 2022, from http://www.nawapi.gov.vn/index.php?option=com_content&view=article&id=3498%3Amng-nron-nhan-to-hng-tip-cn-mi-trong-cong-tac-nhn-dng-va-d-bao-tai-nguyen-nc&catid=70%3Anhim-v-chuyen-mon-ang-thc-hin&Itemid=135&lang=vi
18. *NHẬN DẠNG CHỮ SỐ VIẾT TAY DÙNG MẠNG NEURON NHÂN TẠO 1. MỞ ĐẦU Hiện nay, với hệ thống dữ liệu h*. (n.d.). Tạp chí Khoa học & Công nghệ. Retrieved May 10, 2022, from http://joshusc.hueuni.edu.vn/upload/vol_14/no_1/485_fulltext_4.%C4%90TVT%20-%20Quoc%20-%20Pham%20Phu%20Quoc.pdf
19. Rubinetti, V. (2017, October 5). *But what is a neural network? | Chapter 1, Deep learning*. YouTube. Retrieved May 10, 2022, from https://www.youtube.com/watch?v=aircAruvnKk
20. *techwithtim/Number-Guesser-Neural-Net: This program allows the user to draw a number on the screen and have the program take a guess of which digit it is. This uses a basic neural network model.* (n.d.). GitHub. Retrieved May 10, 2022, from https://github.com/techwithtim/Number-Guesser-Neural-Net
21. *Thuật toán CNN - Convolutional Neural Network*. (n.d.). TopDev. Retrieved May 10, 2022, from https://topdev.vn/blog/thuat-toan-cnn-convolutional-neural-network/#convolutional-la-gi
22. *Tìm hiểu thuật toán tối ưu hóa Adabelief Optimizer*. (2021, January 15). Phạm Duy Tùng. Retrieved May 10, 2022, from

https://www.phamduytung.com/blog/2021-01-15---adabelief-optimizer/#ixzz7SsXi5TYu

23. *Tìm hiểu về Convolutional Neural Network và làm một ví dụ nhỏ về phân loại ảnh*. (n.d.). Viblo. Retrieved May 10, 2022, from https://viblo.asia/p/tim-hieu-ve-convolutional-neural-network-va-lam-mot-vi-du-nho-ve-phan-loai-anh-aWj53WXo56m
24. *Tổng quan về Artificial Neural Network*. (n.d.). Viblo. Retrieved May 10, 2022, from https://viblo.asia/p/tong-quan-ve-artificial-neural-network-1VgZvwYrlAw
25. *Tổng quan về công nghệ Neural Network - FIBO.VN*. (n.d.). Fibo.vn. Retrieved May 10, 2022, from https://fibo.vn/tong-quan-ve-cong-nghe-neural-network/
26. *TTV-GIÁO DỤC ỨNG DỤNG*. (n.d.). TTV-GIÁO DỤC ỨNG DỤNG. Retrieved May 10, 2022, from http://trituevietvn.com/chi-tiet/-NGHIEN-CUU-VE-MANG-NEURON-NHAN-TAO-VA-UNG-DUNG-DU-BAO-DOANH-SO-BAN-HANG-60
27. *ỨNG DỤNG MẠNG NEURON NHÂN TẠO (ANN) TRONG DỰ BÁO ĐỘ RỖNG*. (n.d.). Pvn.vn. Retrieved May 10, 2022, from https://pvn.vn/DataStore/Documents/2019/TapchiDK/so%207.%202019/mang%20neuron%20Ta%20Quoc%20Dung.pdf
28. *Xây dựng mô hình Neural Network*. (2019, April 22). Trí tuệ nhân tạo. Retrieved May 10, 2022, from https://trituenhantao.io/kien-thuc/xay-dung-mo-hinh-neural-network/