CHƯƠNG TRÌNH ƯỚC LƯỢNG GIÁ TRỊ NGÕ RA SỬ DỤNG MÔ HÌNH MẠNG HỌC SÂU

Trần Minh Đô, Nguyễn Đặng Mai Thy
Trường Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh
Bộ môn Kỹ thuật Máy tính và Viễn thông
Môn Thực tập Học máy và Trí tuệ nhân tạo

I. GIỚI THIỆU

Trong nhiều bài toán trong lĩnh vực dự đoán và phân loại, việc ước lượng giá trị ngõ ra là một yếu tố quan trọng để hiểu và dự đoán kết quả của một hệ thống. Tuy nhiên, việc ước lượng này có thể phức tạp và đòi hỏi một mô hình mạnh mẽ để đạt được độ chính xác và hiệu suất cao.

Xây dựng và thiết kế mô hình mạng học sâu đã trở thành một phương pháp phổ biến và hiệu quả để ước lượng giá trị ngõ ra trong các bài toán phức tạp. Mạng học sâu, còn được gọi là mạng nơ-ron nhân tạo đa tầng (Artificial Neural Network - ANN), là một mô hình tính toán lấy cảm hứng từ cấu trúc và hoạt động của hệ thống nơ-ron trong não người.

Qua việc sử dụng mô hình mạng học sâu, chúng ta có thể xây dựng một kiến trúc mạng phức tạp với nhiều lớp nơ-ron, mỗi lớp thực hiện một phép tính toán riêng. Các lớp này tương tác với nhau thông qua trọng số và hàm kích hoạt để biểu diễn và học các đặc trưng phức tạp từ dữ liệu đầu vào. Qua quá trình huấn luyện, mô hình học cách tối ưu các trọng số để tối đa hoá hiệu suất dự đoán và ước lượng giá trị ngõ ra.

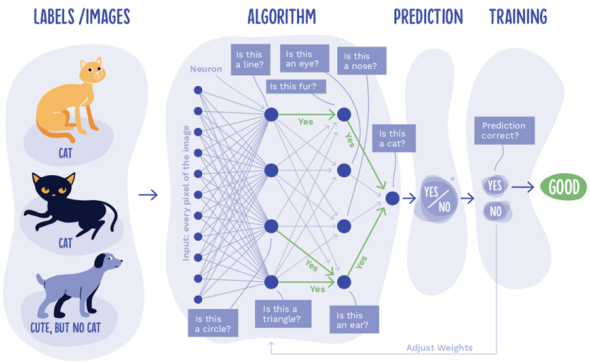
Việc xây dựng và thiết kế mô hình mạng học sâu cho ước lượng giá trị ngõ ra đòi hỏi sự hiểu biết sâu về cấu trúc mạng, các kỹ thuật huấn luyện và tối ưu mô hình. Ngoài ra, việc xử lý và chuẩn bị dữ liệu đầu vào cũng đóng vai trò quan trọng trong việc đảm bảo hiệu suất và độ chính xác của mô hình.

Tổng quan, việc sử dụng mô hình mạng học sâu để ước lượng giá trị ngõ ra là một phương pháp mạnh mẽ và linh hoạt, có thể áp dụng cho nhiều bài toán khác nhau. Qua việc xây dựng và thiết kế mô hình mạng học sâu, chúng ta có thể đạt được những dự đoán chính xác và đáng tin cậy về giá trị ngõ ra, đóng góp quan trọng vào việc hiểu và ứng dụng trong các lĩnh vực thực tế.

II. CƠ SỞ LÝ THUYẾT

A. Tổng quan về học sâu

Lý thuyết học sâu (deep learning) là một lĩnh vực trong machine learning (học máy) tập trung vào việc xây dựng và huấn luyện các mạng nơ-ron nhân tạo sâu. Nó được xem là một trong những phương pháp mạnh nhất và hiệu quả nhất trong lĩnh vực nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên, và nhiều ứng dụng khác. Một trong những thành tựu quan trọng của lý thuyết Học sâu là khả năng học biểu diễn thông qua việc xây dựng mạng nơ-ron nhân tạo sâu, trong đó mỗi lớp nơ-ron tiếp nhận đầu vào từ lớp trước và tạo ra đầu ra thông qua việc tính toán và học các trọng số của mạng. Các mạng nơ-ron sâu thường có hàng trăm hoặc hàng nghìn lớp và triệu hoặc thậm chí tỷ lệ tỷ lệ nơ-ron (mô tả ở hình 1), cho phép chúng mô hình hóa được những đặc trưng phức tạp và trừu tượng.



Hình 1: Hình ảnh minh họa bài toán phân loại dựa trên deep learning.

Một trong những kiến trúc mạng nơ-ron sâu phổ biến là mạng nơ-ron tích chập (Convolutional Neural Network

- CNN), nó đặc biệt phù hợp cho xử lý hình ảnh. Mạng CNN sử dụng các lớp tích chập để học các đặc trưng cục bộ trong hình ảnh và các lớp gộp để giảm kích thước của đầu vào. Điều này giúp nó tìm hiểu và trích xuất các đặc trưng phức tạp từ hình ảnh một cách hiệu quả. Trong lý thuyết Học sâu, mô hình UNet [**?**] là một kiến trúc mạng nơ-ron sâu đáng chú ý được phát triển bởi Olaf Ronneberger et al. vào năm 2015. Kiến trúc UNet là một sự kết hợp giữa mạng tích chập và mạng UNet, được thiết kế đặc biệt để xử lý nhiệm vụ phân đoạn hình ảnh. Nó có một kiến trúc encoder-decoder với các kết nối bỏ qua để giữ lại thông tin bổ sung từ các lớp encoder và kết hợp chúng với lớp decoder tương ứng.

B. Mạng nơ-ron nhân tạo - Artificial Neural Network (ANN)

Mạng nơ-ron nhân tạo là gì?

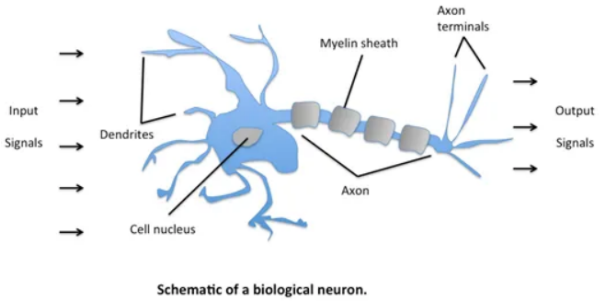
Mạng nơ-ron nhân tạo là một trong những công cụ chính được sử dụng trong học máy. Như phần "thần kinh" – “neural” trong tên của chúng cho thấy, chúng là những hệ thống lấy cảm hứng từ não nhằm tái tạo cách con người chúng ta học. Mạng nơ-ron bao gồm các lớp đầu vào và đầu ra, cũng như (trong hầu hết các trường hợp) một lớp ẩn bao gồm các đơn vị biến đổi đầu vào thành thứ mà lớp đầu ra có thể sử dụng. Chúng là những công cụ tuyệt vời để tìm kiếm các mẫu quá phức tạp hoặc nhiều để một lập trình viên con người trích xuất và dạy máy nhận ra.

Mặc dù mạng lưới thần kinh - neural networks (còn được gọi là "perceptron") đã xuất hiện từ những năm 1940, nhưng chỉ trong vài thập kỷ qua, chúng mới trở thành một phần chính của trí tuệ nhân tạo. Điều này là do sự xuất hiện của một kỹ thuật gọi là "lan truyền ngược" – “backpropagation”, cho phép các mạng điều chỉnh các lớp tế bào thần kinh ẩn của chúng trong các tình huống mà kết quả không phù hợp với những gì người sáng tạo đang hy vọng - như một mạng được thiết kế để nhận ra chó, ví dụ như xác định sai một con mèo.

Một tiến bộ quan trọng khác là sự xuất hiện của các mạng nơ-ron học sâu, trong đó các lớp khác nhau của mạng nhiều lớp trích xuất các tính năng khác nhau cho đến khi nó có thể nhận ra những gì nó đang tìm kiếm.

Ý tưởng về ANN dựa trên niềm tin rằng hoạt động của bộ não con người bằng cách tạo ra các kết nối phù hợp có thể được bắt chước bằng cách sử dụng silicon và dây làm tế bào thần kinh - neurons và đuôi gai sống - dendrites.

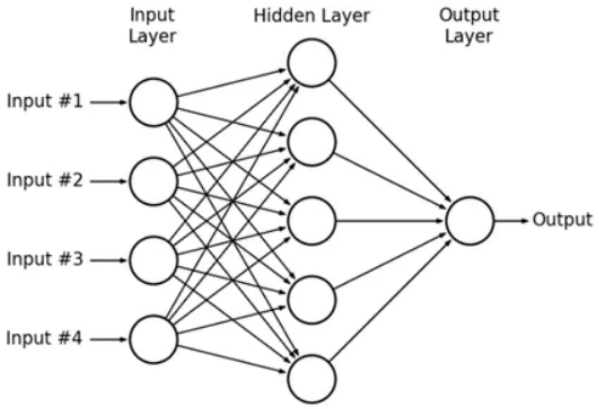
Bộ não con người bao gồm 86 tỷ tế bào thần kinh được gọi là tế bào thần kinh - neurons. Chúng được kết nối với hàng nghìn tế bào khác bằng sợi trục - Axons. Các kích thích từ môi trường bên ngoài hoặc đầu vào từ các cơ quan cảm giác được chấp nhận bởi đuôi gai. Những đầu vào này tạo ra các xung điện, nhanh chóng đi qua mạng lưới thần kinh. Một tế bào thần kinh sau đó có thể gửi tin nhắn đến một tế bào thần kinh khác để xử lý vấn đề hoặc không gửi nó về phía trước.



Hình 2: Sơ đồ mạng nơ-ron sinh học

ANN bao gồm nhiều nút - nodes, bắt chước các tế bào thần kinh sinh học - neurons của não người. Các tế bào thần kinh được kết nối bởi các liên kết và chúng tương tác với nhau. Các nút có thể lấy dữ liệu đầu vào và thực hiện các thao tác đơn giản trên dữ liệu. Kết quả của các hoạt động này được chuyển đến các tế bào thần kinh khác. Đầu ra tại mỗi nút được gọi là giá trị kích hoạt - activation hoặc giá trị nút của nó - node value.

Mỗi liên kết được liên kết với trọng số - weight . ANN có khả năng học hỏi, diễn ra bằng cách thay đổi giá trị trọng lượng. Hình minh họa sau đây cho thấy một ANN đơn giản

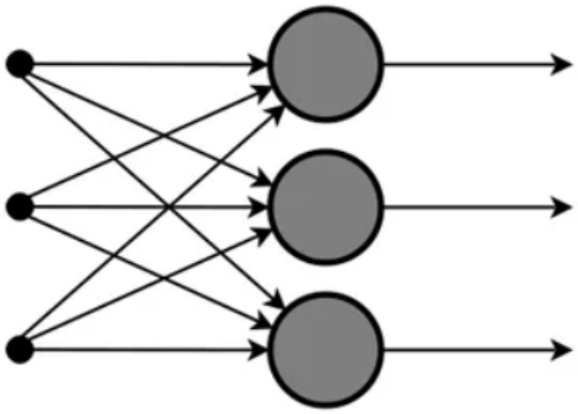


Hình 3: Hình minh họa một ANN đơn giản

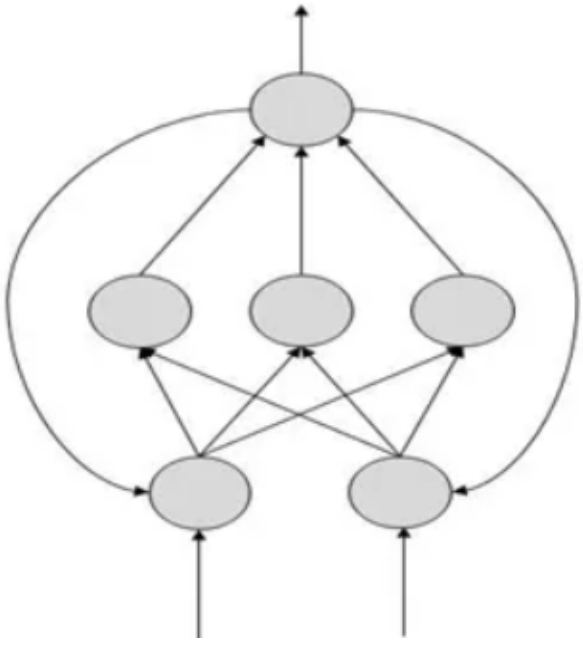
Các loại mạng thần kinh nhân tạo

Có hai cấu trúc liên kết Mạng nơ-ron nhân tạo - chuyển tiếp nguồn cấp dữ liệu ‘FeedForward’ và phản hồi ‘Feedback’.

- *ANN chuyển tiếp nguồn cấp dữ liệu - FeedForward:* Trong ANN này, luồng thông tin là một chiều. Một đơn vị gửi thông tin đến một đơn vị khác mà từ đó nó không nhận được bất kỳ thông tin nào. Không có vòng phản hồi. Chúng được sử dụng trong việc tạo / nhận dạng / phân loại mẫu (generation/recognition/classification). Họ có đầu vào và đầu ra cố định.



Hình 4: Hình minh họa ANN chuyển tiếp nguồn cấp dữ liệu



Hình 5: Hình minh họa ANN phản hồi

- *ANN phản hồi - FeedBack :* Ở đây, các vòng phản hồi được cho phép. Chúng được sử dụng trong các bộ nhớ có địa chỉ nội dung - content-addressable memories.

Chức năng kích hoạt (Activation Functions) và có các loại?

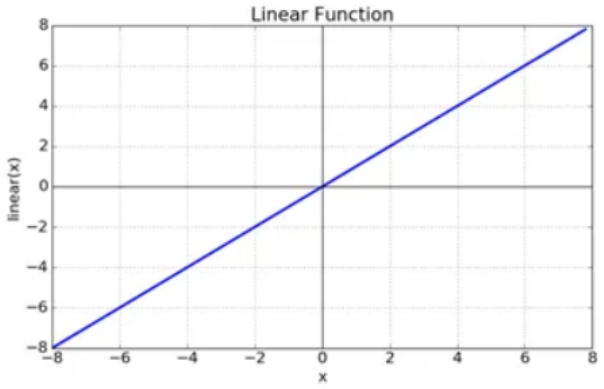
Chức năng kích hoạt chỉ là một hàm thing mà bạn sử dụng để lấy đầu ra của nút. Nó còn được gọi là Chức năng chuyển (Transfer Function).

Các chức năng kích hoạt (Activation functions) thực sự quan trọng đối với Mạng nơ-ron nhân tạo (Artificial Neural Network) để tìm hiểu và hiểu ý nghĩa của một cái gì đó được phân bổ lại và ánh xạ chức năng phức tạp phi tuyến tính giữa các đầu vào và biến phản hồi. Họ giới thiệu các thuộc tính phi tuyến tính (non-linear properties) cho Mạng của chúng tôi. Mục đích chính của chúng là chuyển đổi tín hiệu đầu vào của một nút trong ANN thành tín hiệu đầu ra. Tín hiệu đầu ra đó bây giờ được sử dụng làm đầu vào trong lớp tiếp theo trong ngăn xếp.

Cụ thể trong A-NN, chúng ta thực hiện tổng các tích của đầu vào - inputs (X) và Trọng số tương ứng của chúng - Weights (W) và áp dụng hàm Kích hoạt Activation function f (x) cho nó để lấy đầu ra của lớp đó và đưa nó làm đầu vào cho lớp tiếp theo.

Nó được sử dụng để xác định đầu ra của mạng nơ-ron như có hoặc không. Nó ánh xạ các giá trị kết quả trong khoảng từ 0 đến 1 hoặc -1 đến 1, v.v. (tùy thuộc vào hàm).

Chức năng kích hoạt có thể dựa trên 2 loại: Chức năng kích hoạt tuyến tính (Linear Activation Function) và Chức năng kích hoạt phi tuyến tính (Non-linear Activation Functions)



Hình 6: Hình chức năng kích hoạt tuyến tính

- *Chức năng kích hoạt tuyến tính (Linear Activation Function)*
  Như bạn có thể thấy, hàm là một đường thẳng hoặc tuyến tính. Do đó, đầu ra của các chức năng sẽ

không bị giới hạn giữa bất kỳ phạm vi nào.
Tóm tắt: f(x) = x, Phạm vi: (-vô cực đến vô cực). Nó không giúp ích gì cho sự phức tạp của các tham số khác nhau của dữ liệu thông thường được đưa vào mạng thần kinh.

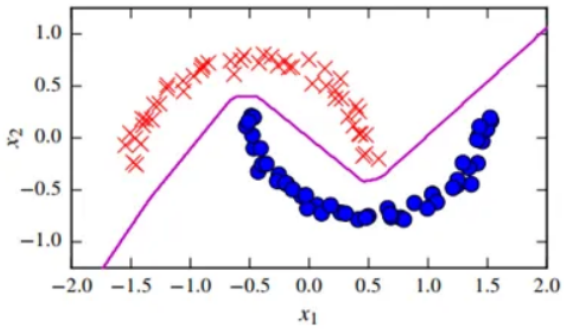


Fig: Non-linear Activation Function

Hình 7: Hình chức năng kích hoạt phi tuyến tính

- *Chức năng kích hoạt phi tuyến tính (Non-linear Activation Function)*
  Chức năng kích hoạt phi tuyến là các chức năng kích hoạt được sử dụng nhiều nhất. Tính phi tuyến giúp làm cho biểu đồ trông giống như thế này.
  Nó giúp mô hình dễ dàng khái quát hóa hoặc thích ứng (generalize or adapt) với nhiều loại dữ liệu và phân biệt giữa các đầu ra.
  Các thuật ngữ chính cần thiết để hiểu cho các hàm phi tuyến là: Đạo hàm hoặc vi phân - Derivative or Differential: Thay đổi trục y w.r.t. thay đổi trục x. Nó còn được gọi là một con dốc. Chức năng đơn điệu - Monotonic function: Một chức năng hoàn toàn không tăng hoặc không giảm. Các hàm kích hoạt phi tuyến chủ yếu được chia dựa trên phạm vi hoặc đường cong- range or curves

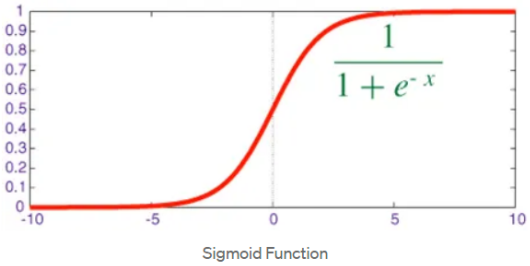
Các loại chức năng kích hoạt khác nhau trong phi tuyến tính

- *Chức năng kích hoạt Sigmoid - Sigmoid Activation Function*
  Đường cong hàm Sigmoid trông giống như hình chữ S.
  Lý do chính tại sao chúng ta sử dụng hàm sigmoid là nó tồn tại giữa (0 đến 1). Do đó, nó đặc biệt được sử dụng cho các mô hình mà chúng ta phải dự đoán xác suất - predict the probability dưới dạng đầu ra. Vì xác suất của bất cứ thứ gì chỉ tồn tại

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Hình 8: Công thức hàm Sigmoid

trong khoảng từ 0 đến 1, sigmoid là lựa chọn đúng đắn.



Sigmoid Function

Hình 9: Đồ thị hàm Sigmoid

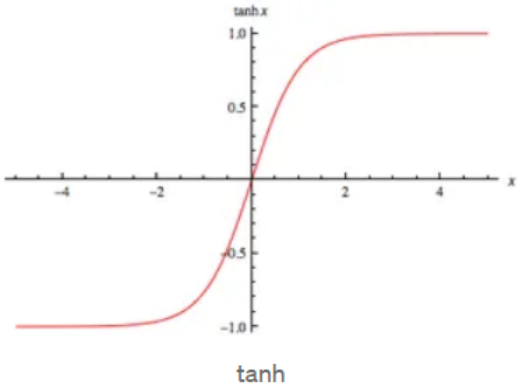
Chức năng này có thể phân biệt được - differentiable. Điều đó có nghĩa là, chúng ta có thể tìm thấy độ dốc của đường cong sigma tại hai điểm bất kỳ.
Hàm là đơn điệu - monotonic nhưng đạo hàm của hàm thì không.
Chức năng sigmoid logistic có thể khiến mạng lưới thần kinh bị kẹt tại thời điểm đào tạo.
Hàm softmax là một hàm kích hoạt logistic tổng quát hơn được sử dụng để phân loại nhiều lớp.

- *Chức năng kích hoạt Tanh - Tanh Activation Function*

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

Hình 10: Công thức hàm Tanh

Tanh cũng giống như Logistic Sigmoid nhưng tốt hơn. Phạm vi của hàm tanh là từ (-1 đến 1). Tanh cũng là sigma (hình chữ S).
Ưu điểm là các đầu vào âm sẽ được ánh xạ âm mạnh và các đầu vào bằng 0 sẽ được ánh xạ gần



Hình 11: Đồ thị hàm Tanh

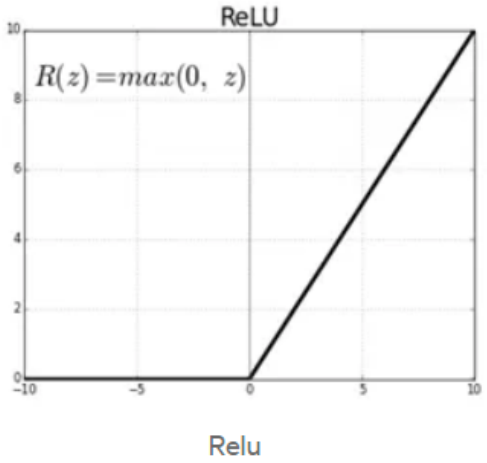
bằng 0 trong biểu đồ tanh. Chức năng này có thể phân biệt được - differentiable.
Chức năng này là đơn điệu - monotonic trong khi dẫn xuất của nó không phải là đơn điệu - derivative is not monotonic.
Hàm tanh chủ yếu được sử dụng để phân loại giữa hai lớp.
Cả hai chức năng kích hoạt sigmoid tanh và logistic đều được sử dụng trong lưới chuyển tiếp thức ăn.

- *Chức năng kích hoạt ReLU (Bộ tuyến tính chỉnh lưu) - (Rectified Linear Unit)*

$$f(x) = max(0, x)$$

Hình 12: Công thức hàm ReLU

ReLU là chức năng kích hoạt được sử dụng nhiều nhất trên thế giới hiện nay. Vì nó được sử dụng trong hầu hết các mạng nơ-ron tích chập hoặc học sâu.
Như bạn có thể thấy, ReLU được chỉnh lưu một nửa (từ dưới lên). f(z) bằng 0 khi z nhỏ hơn 0 và f(z) bằng z khi z ở trên hoặc bằng không.
Phạm vi: [0 đến vô cùng)
Chức năng và nó là đạo hàm cả hai đều là đơn trương - both are monotonic.
Nhưng vấn đề là tất cả các giá trị âm trở thành 0 ngay lập tức, điều này làm giảm khả năng của mô



Hình 13: Đồ thị hàm ReLU

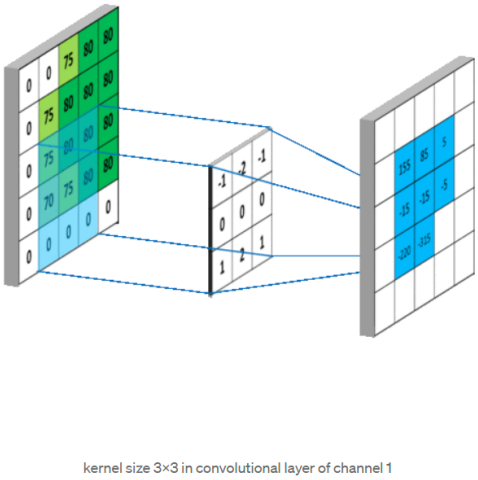
hình để phù hợp hoặc đào tạo từ dữ liệu đúng cách. Điều đó có nghĩa là bất kỳ đầu vào âm nào được cung cấp cho hàm kích hoạt ReLU đều biến giá trị thành 0 ngay lập tức trong biểu đồ, từ đó ảnh hưởng đến biểu đồ kết quả bằng cách không ánh xạ các giá trị âm một cách thích hợp.

Có những lớp ẩn (Hidden layers) nào?

- *Lớp dày đặc - Dense Layers* Các lớp dày đặc là các lớp (nhiều nút) có mỗi nút của nó được kết nối với tất cả các nút trong lớp trước. Ví dụ: hình 1: Lớp dày đặc và hình 2: Lớp không đậm đặc
Lưu ý: Đôi khi trong sơ đồ NN, các nút từ lớp n-1 không nhất thiết phải được vẽ để được liên kết với một số nút trong lớp n nhưng tại sao Keras chỉ triển khai các lớp dày đặc? Vâng, một câu trả lời đơn giản là trọng số trên các liên kết có thể được đặt thành 0 và lớp dày đặc sẽ dẫn đến lớp không dày đặc. Các lớp dày đặc có một sức mạnh đại diện tuyệt vời.
- *Lớp Lambda - Lambda Layers* Từ lớp n-1, giá trị N của một nút được ánh xạ đơn giản đến một nút khác trong lớp n của giá trị f (N). Lambda đang định nghĩa hàm f (tương tự như lambda trong python).
//Keras triển khai lớp lambda bình phương các nút lớp valuemodel = Sequential()
model.add(Dense(32, input-dim=32))
model.add(Lambda(lambda x: x**2))
Các lớp Lambda có thể trông đơn giản trong ví dụ này nhưng mang lại nhiều tự do cho người dùng để áp dụng các thao tác cụ thể theo trường hợp hơn.

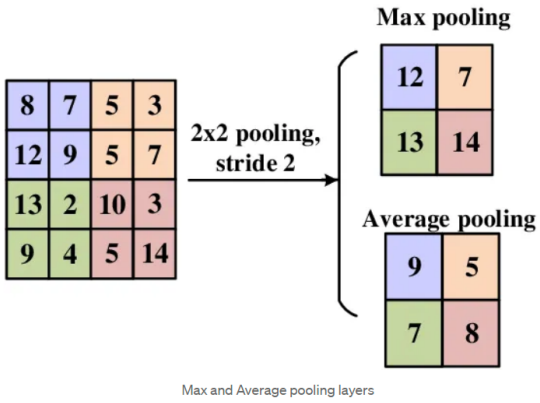
Chúng ta có thể nghĩ về chuẩn hóa, giá trị tuyệt đối và thậm chí ứng dụng lớp này để tạo phép nhân ma trận tích chập trên các lớp trước đó (các "bộ lọc" tích chập này sẽ được phát triển thêm trong các bài báo Deep Learning: CNN sắp được xuất bản)

- *Regularization Layers* Cũng giống như việc thêm một quy tắc vào bình phương nhỏ nhất làm cho hồi quy sườn núi, chúng ta thêm một giá trị bổ sung: lần vô hướng định mức lasso của x hoặc chuẩn euclid của x; đến chức năng khách quan.
  //Keras triển khai
  keras.layers.ActivityRegularization(l1=1.0, l2=0.0)
  Và một lần nữa, mục tiêu trước hết là tránh quá mức vì nó có thể hoạt động như một tiếng ồn nhưng nó cũng cho phép các trọng lượng mà chúng tôi tối ưu hóa không phát nổ về mặt giá trị.
- *Lớp nhiễu Gaussian - Gaussian Noise Layers*
  Một lớp nhiễu Gaussian thêm vào các nút lớp trước đó một giá trị Gaussian (trung bình bằng không) cho mỗi nút.
  //độ lệch chuẩn làm đầu vào
  keras.layers.GaussianNoise(std-deviation)
  Một lần nữa, vì chúng tôi đang thêm tiếng ồn, chúng tôi giảm thiểu quá mức, nó tương tự như một lớp chính quy nhưng điều đó không phụ thuộc vào đầu vào.
- *Dropout Layers*
  Các lớp bỏ học là các lớp không ảnh hưởng đến giá trị đầu ra mà thay vào đó tác động lên các bài tập tạ. Sẽ chọn ngẫu nhiên một "phần" các nút sẽ được đặt thành 0.
  // tỷ lệ trong [0,1]
  keras.layers.Dropout(rate)
  Có một vài nhận xét được đưa ra ở đây:
  Việc các nút được đặt thành 0 sẽ dừng luồng / đường dẫn đi qua nút được đặt thành 0. Trong đào tạo, nó ngụ ý rằng việc tối ưu hóa được thực hiện trong một không gian dự kiến của không gian ban đầu, thay vì một gradient đầy đủ, chúng ta có được một đạo hàm một phần.
  Thả các phần của tạ làm cho quá trình lặp lại đào tạo nhanh hơn
  Và việc thả các nút "ngẫu nhiên" trong một lớp ngăn chặn việc dựa vào một nút cụ thể để thực hiện suy luận / đầu ra, do đó ngăn ngừa quá mức.
- *Lớp tích chập - Convolutional layer*
  Lọc hình ảnh (kernel) là quá trình sửa đổi hình ảnh bằng cách thay đổi sắc thái hoặc màu sắc của pixel. Nó cũng được sử dụng cho độ sáng và độ tương phản.



Hình 14: Hình minh họa lớp tích chập - Convolutional layer

- *Lớp gộp - Pooling layer*

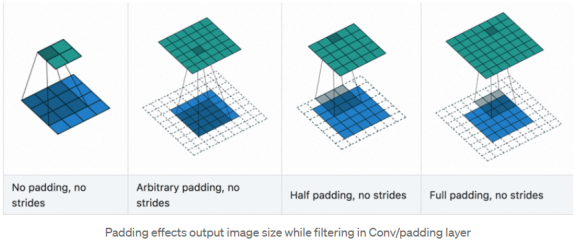


Hình 15: Hình minh họa lớp gộp - Pooling layer

Lớp gộp được sử dụng để giảm kích thước bản đồ tính năng. Do đó, nó làm giảm không có tham số để tìm hiểu và số lượng tính toán được thực hiện trong mạng. Lớp tổng hợp tóm tắt các tính năng có trong một vùng của bản đồ địa vật được tạo bởi lớp tích chập.
Trong hình ảnh này, kích thước hạt nhân là 2x2 và sải chân 2. có nghĩa là hạt nhân bước hai lần.
Lớp gộp tối đa tìm thấy tối đa trong hạt nhân 2x2 của hình ảnh đầu vào (như tối đa trong khu vực hạt nhân màu xanh nhạt trong số [8,7,12,9] là 12)
Lớp gộp trung bình lấy trung bình hạt nhân 2x2

(như trong các khu vực màu xanh lam [8 + 7 + 12 + 9] / 4 = 9)

- *Lớp đệm - Padding layer*
  Hiệu ứng đệm xuất kích thước hình ảnh trong khi lọc trong lớp Conv / đệm



Hình 16: Hình minh họa Lớp đệm - Padding layer

Nó tương tự như lớp tích chập vì nó đề cập đến số lượng pixel được thêm vào hình ảnh khi nó đang được xử lý bởi kernel hoặc bộ lọc. Khi không sử dụng Stride thì theo mặc định là 1. Một nửa đệm có nghĩa là một nửa kích thước bộ lọc và đệm trung bình đệm đầy đủ bằng kích thước của bộ lọc / nhân. Đệm được thực hiện để giảm mất dữ liệu giữa các cạnh / ranh giới của hình ảnh.
Kích thước đầu ra của hình ảnh được tính bằng công thức này [(W-K+2P)/S]+1.
W là âm lượng đầu vào
K là kích thước Kernel
P là đệm
S là sải chân

- *Flatten layer*
  Trực giác đằng sau lớp làm phẳng là chuyển đổi dữ liệu thành mảng 1 dimentional để cung cấp cho lớp tiếp theo. Chúng tôi làm phẳng đầu ra của lớp tích chập thành vectơ tính năng dài đơn. được kết nối với mô hình phân loại cuối cùng, được gọi là lớp được kết nối đầy đủ. Giả sử chúng ta đã [5,5,5] Bản đồ tính năng gộp được làm phẳng thành vectơ đơn 1x125. Vì vậy, các lớp phẳng chuyển đổi mảng đa chiều thành vectơ đơn chiều.
- *Connected Components*
  Mô hình lấy hình ảnh đầu vào có kích thước 28x28 và áp dụng lớp Conv đầu tiên với hạt nhân 5x5, sải chân 1 và đệm không đầu ra n1 kênh có kích thước 24x24 được tính bằng đầu ra của lớp gộp là (Kích thước đầu vào - Kích thước hồ bơi + 2 * Padding) / Stride + 1.. thì lớp poling giống như conv nhưng lần này kích thước bộ lọc 2x2 và sải chân 2, khi chúng ta tính toán bằng cách sử dụng đầu ra công thức lớp Conv là 12x12 của cùng một kênh n1. Tôi

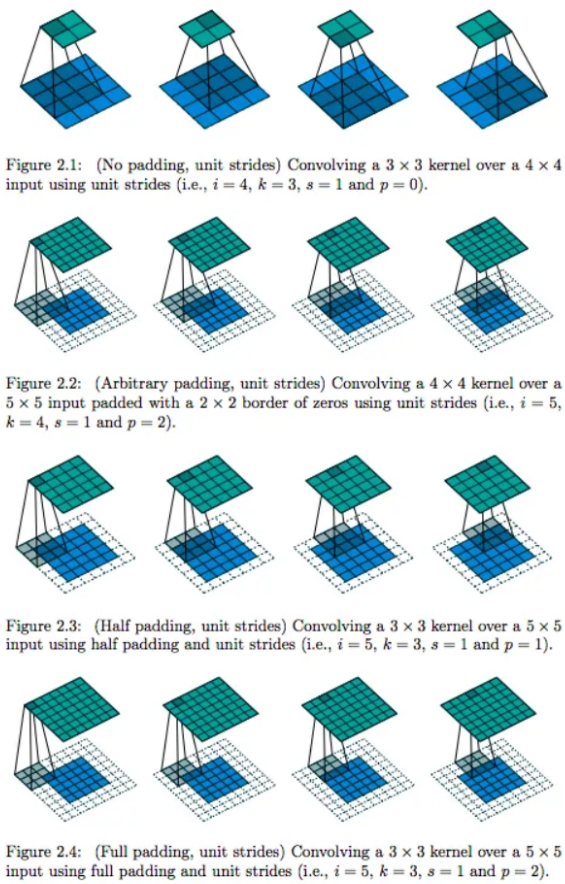


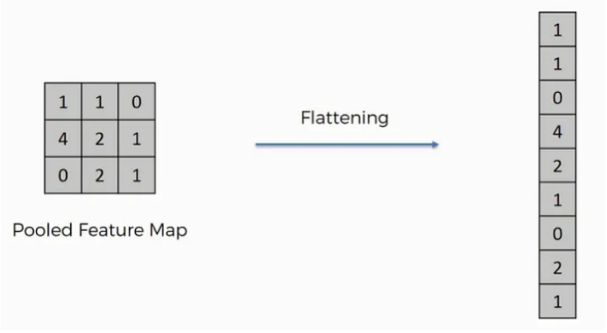
Figure 2.1: (No padding, unit strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

Figure 2.2: (Arbitrary padding, unit strides) Convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$).

Figure 2.3: (Half padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$).

Figure 2.4: (Full padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$).

Hình 17: Hình minh họa Lớp đệm - Padding layer

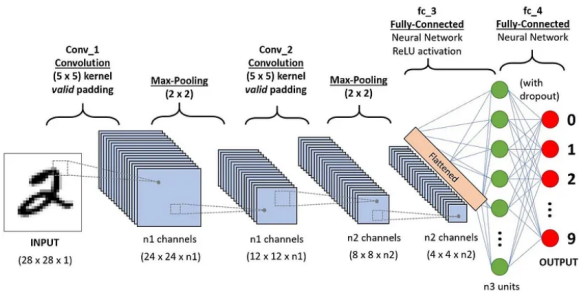


Hình 18: Hình minh họa Lớp đệm - Padding layer

lặp lại cách tương tự một lần nữa và ở cuối lớp phẳng chuyển đổi mảng hai chiều thành vectơ một chiều.

III. QUÁ TRÌNH THỰC HIỆN – GIẢI THÍCH CODE

Phân tích Đề bài:



Hình 19: Hình minh họa Lớp đệm - Padding layer

Viết chương trinh ước lượng giá trị ngõ ra sử dụng thông qua xây dựng và thiết kế mô hình mạng học sâu. Đánh giá hiệu năng trên tập dữ liệu cho trước dựa trên metric RMSE (root mean spuared error)

Yêu cầu:

Dataset: Input 11 attributes (A – K), Output 2 attributes (L – M)

Total params < 100K

50% training, 50% validation (random state khác 0)

Accuracy in validation set

RMSE nhỏ nhất

Mẫu tập dữ liệu – Dataset:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	1	5	4	3	3	10	6	10	330	500	2	0.016606	1.245632	
2	5	2	1	4	3	10	5	31	340	590	1	0	0.622222	
3	3	5	6	9	4	12	6	26	380	550	1.5	0.011124	0.857026	
4	5	5	4	5	2	10	4	27	350	570	0.5	0	0.305556	
5	2	1	1	6	0	11	6	16	350	580	0.5	0.384398	0.188101	
6	5	1	6	9	3	9	3	33	400	550	1.5	1	0	
7	4	3	6	7	3	7	2	5	310	510	1	0.383899	0.403888	
8	6	6	5	5	2	9	6	7	340	520	1.5	1.91E-07	0.933333	
9	5	6	4	5	3	9	4	23	360	570	1.5	0	0.9	
10	1	5	5	4	1	8	6	11	340	510	2	0.002719	1.241061	
11	2	4	1	7	0	9	4	8	320	540	0.5	0.143559	0.275964	
12	3	1	1	6	0	10	2	7	370	540	2	0.681034	0.375672	
13	3	4	2	3	3	11	5	31	390	560	1.5	0	0.85	
14	5	1	3	4	3	9	3	26	300	580	0.5	0	0.333333	
15	2	1	5	6	3	9	5	0	340	530	0.5	0.999764	7.34E-05	
16	2	6	3	9	4	12	3	26	400	600	2	0.087168	1.014258	
17	3	2	4	6	3	11	4	20	320	500	1.5	9.21E-05	0.966578	
18	1	5	4	10	3	9	6	9	360	530	0.5	0.921714	0.023486	
19	1	2	6	9	4	9	4	17	400	510	1.5	1	0	
20	5	5	3	6	3	8	6	9	350	530	1.5	0.000161	0.916519	
21	4	5	5	10	0	10	4	0	330	510	0.5	0.999999	2.98E-07	
22	3	1	5	3	3	7	5	2	350	580	1.5	0.56196	0.401537	
23	2	2	4	6	4	10	4	29	340	570	2	0.000138	1.244272	
24	2	6	3	2	0	8	6	33	320	550	1.5	0	0.966667	
25	5	6	6	9	3	8	6	22	380	540	1	1.13E-06	0.577777	
26	2	6	4	6	2	7	2	31	330	550	0.5	0.136558	0.273423	
27	3	4	3	10	0	11	3	18	380	510	0.5	0.045136	0.27585	

Hình 20: Mẫu tập dữ liệu – Dataset

A. Tiền xử lý dữ liệu

Nhập thư viện

Trong bước này, chúng ta import ba Libraries trong phần Data Preprocessing. Thư viện là một công cụ mà bạn có thể sử dụng để thực hiện một công việc cụ thể. Trước hết, chúng ta import thư viện pandas - là một thư viện mã nguồn mở dùng để xử lý và phân tích dữ liệu dạng bảng. Nó cung cấp các công cụ để đọc, ghi, lọc, biến đổi và thống kê dữ liệu một cách dễ dàng.

```
# Tiền xử lý dữ liệu
import pandas as pd
```

Nhập tập dữ liệu

Để import tập dữ liệu sử dụng câu lệnh read_csv (‘ đường dẫn đến tập dữ liệu ‘)

```
# Đọc dữ liệu từ CSV vào DataFrame
data = pd.read_csv('/kaggle/input/throughput-prediction/datasetNov1_data_full.csv')
```

Xử lý dữ liệu

Chia tập dữ liệu theo yêu cầu về dataset của đề bài. Chia các giá trị của Input vào X, các giá trị của Output vào y. Sử dụng câu lệnh iloc để lấy các cột dữ liệu tương ứng.

```
# Chia dữ liệu input và output
X = data.iloc[:, 0:11]          # Input 11 attributes ( A - K )
y = data.iloc[:, 11:13]         # Ouput 2 attributes ( L - M )
```

Chuẩn hóa dữ liệu

Sử dụng phương pháp chuẩn hóa dữ liệu MinMaxScaler để đưa dữ liệu về các khoảng phù hợp nhất để tiến hành chia dữ liệu training.

```
# Chuẩn hóa dữ liệu
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X = scaler.fit_transform(X)
y = scaler.fit_transform(y)
```

MinMax Scaler là một phương pháp chuẩn hóa dữ liệu trong Machine Learning, được sử dụng để đưa các giá trị của một biến về khoảng giá trị cụ thể, thường là từ 0 đến 1. Công thức chung của MinMax Scaler được tính như sau:

Trong đó:

Phương pháp MinMax Scaler thường được sử dụng trong các mô hình Machine Learning như Neural Networks, SVM, K-means Clustering và nhiều thuật toán khác, khi cần đưa các biến có đơn vị và phạm vi khác nhau về cùng một khoảng giá trị chuẩn hóa.

$$X_{\text{new}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

- X là giá trị ban đầu của biến.
- X_{new} là giá trị sau khi chuẩn hóa.
- X_{min} là giá trị nhỏ nhất của biến trong tập dữ liệu ban đầu.
- X_{max} là giá trị lớn nhất của biến trong tập dữ liệu ban đầu.

Chia tập dữ liệu thành các tập train và tập validation

Thực hiện theo yêu cầu của đề bài chia tập dữ liệu thành 50% cho tập training và 50% cho tập validation và đặt tham số random_state để đảm bảo khả năng tái sản xuất của việc chia dữ liệu. (randomly)

```
# Chia dữ liệu thành tập train và tập validation
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.5, random_state=42)
```

B. Xây dựng mô hình

Nhập thư viện

Nhập Thư viện Keras sẽ xây dựng một mạng nơ-ron sâu dựa trên Tensorflow.

TensorFlow là một thư viện mã nguồn mở phổ biến cho việc xây dựng và huấn luyện mô hình học máy. Nó cung cấp các công cụ và lớp để xây dựng mạng nơ-ron và thực hiện các phép toán đại số tuyến tính trên dữ liệu.

Keras là một API cao cấp được xây dựng trên TensorFlow và hỗ trợ xây dựng mô hình mạng nơ-ron một cách dễ dàng và trực quan hơn. Nó cung cấp các lớp, hàm và công cụ để xây dựng mô hình học sâu.

Sử dụng Keras thông qua TensorFlow để khởi tạo mô hình Sequential và thêm các lớp Dense vào mô hình.

```
# Xây dựng mô hình mạng học sâu
from tensorflow.keras.models import Sequential      #Khởi tạo mô hình ANN
from tensorflow.keras.layers import Dense           #Cấu trúc các layer khác nhau
```

Khởi tạo mô hình ANN

Khởi tạo mô hình ANN bằng cách sử dụng mô hình Sequential và truyền danh sách các layer vào hàm khởi tạo của nó.

```
model = Sequential([
```

Thêm lớp đầu vào và lớp ẩn đầu tiên

Định nghĩa một layer dense (fully connected) với 640 đơn vị (units). Đây là layer đầu tiên trong mạng nơ-ron và cũng là layer đầu vào (input layer). Hàm kích hoạt được sử dụng là 'relu' (Rectified Linear Unit). Tham số input_shape=(11,) xác định kích thước của đầu vào, với 11 đặc trưng.

```
Dense(640,activation='relu', input_shape=(11,)),    #Input Layer & first hidden layer
```

Thêm các hidden layer khác

```
Dense(64,activation ="relu"),                       #Second hidden layer
Dense(512,activation ="relu"),                      #Third hidden layer
Dense(32,activation ="relu"),                       #Fourth hidden layer
Dense(32,activation ="relu"),                       #Fifth hidden layer
```

Layer thứ hai trong mạng là một layer dense với 64 đơn vị (units).Hàm kích hoạt được sử dụng là 'relu'.

Layer thứ ba trong mạng là một layer dense với 512 đơn vị (units).Hàm kích hoạt được sử dụng là 'relu'.

Layer thứ tư trong mạng là một layer dense với 32 đơn vị (units).Hàm kích hoạt được sử dụng là 'relu'.

Layer thứ năm trong mạng là một layer dense với 32 đơn vị (units).Hàm kích hoạt được sử dụng là 'relu'.

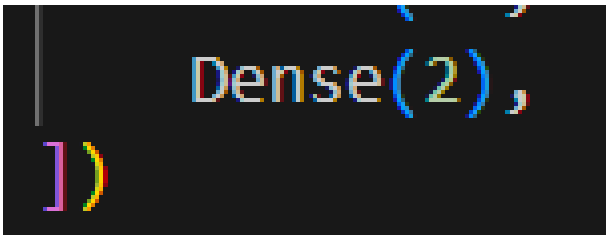
Các dòng code trên định nghĩa kiến trúc của mạng nơ-ron với các hidden layer và hàm kích hoạt 'relu' được sử dụng để tạo các đầu ra phi tuyến của các layer. Các giá trị đơn vị (units) và hàm kích hoạt (activation) có thể được điều chỉnh tùy thuộc vào yêu cầu của bài toán và dữ liệu đang xử lý.

Thêm lớp đầu ra

Định nghĩa một layer dense với 2 đơn vị (units). Đây là layer cuối cùng trong mạng nơ-ron và là layer đầu ra (output layer).

Trong một bài toán phân loại nhị phân (binary classification), thông thường sẽ sử dụng một layer cuối cùng với 1 đơn vị và hàm kích hoạt là 'sigmoid' để đưa ra dự đoán xác suất thuộc vào lớp positive (1) hay negative (0). Tuy nhiên, trong trường hợp này, vì bạn đang sử dụng Dense(2), nên mô hình sẽ đưa ra dự đoán xác suất thuộc vào 2 lớp khác nhau. Cách mà mô hình đưa ra

dự đoán cụ thể phụ thuộc vào hàm kích hoạt mà bạn sử dụng ở layer này.



Kiểm tra trọng số của model

Để xem tóm tắt (summary) của mô hình, sử dụng phương thức summary() trên đối tượng mô hình.

Nó sẽ hiển thị thông tin về các layer trong mô hình, số lượng tham số cần tối ưu (trainable parameters), và kích thước của đầu ra của mỗi layer.

Thông qua tóm tắt này, có thể kiểm tra kiến trúc của mô hình, số lượng tham số và kích thước đầu ra của mỗi layer.

```
model.summary()
```

```
Model: "sequential_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_191 (Dense)           (None, 640)               7680

 dense_192 (Dense)           (None, 64)                41024

 dense_193 (Dense)           (None, 512)               33280

 dense_194 (Dense)           (None, 32)                16416

 dense_195 (Dense)           (None, 32)                1056

 dense_196 (Dense)           (None, 2)                 66

=================================================================
Total params: 99522 (388.76 KB)
Trainable params: 99522 (388.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Ở đây có thể thấy param của từng hidden layer cũng như total params của model bằng 99522 nhỏ hơn 100K đúng với yêu cầu mà đề bài đưa ra.

Biên dịch mô hình ANN

Để compile mô hình ANN đã xây dựng sử dụng phương thức compile và thêm một số tham số.

Trong đó, một tối ưu hóa Adam được sử dụng với tốc độ học (learning rate) là 0.001 và hàm mất mát (loss function) được chọn là mean squared error (MSE).

Với việc sử dụng tối ưu hóa Adam là đang cấu hình mô hình để sử dụng thuật toán Adam để tối ưu hóa các tham số của mô hình trong quá trình huấn luyện. Tốc độ học (learning rate) 0.001 xác định tốc độ cập nhật cho các tham số trong quá trình tối ưu hóa. Có thể điều chỉnh giá trị tốc độ học tùy thuộc vào bài toán cụ thể và hiệu suất huấn luyện.

Hàm mất mát (loss function) được chọn là mean squared error (MSE). Điều này có nghĩa là mô hình sẽ sử dụng MSE để đo lường sự khác biệt giữa đầu ra được dự đoán và đầu ra thực tế trong quá trình huấn luyện. Mục tiêu của mô hình là giảm thiểu giá trị của hàm mất mát này.

Sau khi mô hình được biên dịch, nó đã sẵn sàng để được huấn luyện và đánh giá bằng cách sử dụng dữ liệu.

```
# Compile mô hình
from tensorflow.keras.optimizers import Adam
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')
```

Huấn luyện mô hình trên tập validation

Để huấn luyện mô hình trên tập validation sử dụng fit(), một phương thức được dùng để huấn luyện mô hình.

```
# Huấn luyện mô hình trên tập validation
history = model.fit(X_train, y_train, epochs=500,batch_size=128, validation_data=(X_val, y_val), verbose=1)
```

Dưới đây là giải thích từng tham số trong phương thức fit():

X_train và y_train là dữ liệu huấn luyện, trong đó X_train là các đặc trưng huấn luyện và y_train là nhãn tương ứng.

epochs=500 xác định số lượng epoch, tức là số lần mô hình sẽ được huấn luyện trên toàn bộ tập dữ liệu huấn luyện.

batch_size=128 xác định kích thước của các batch, tức là số lượng mẫu dữ liệu sẽ được sử dụng để tính gradient và cập nhật tham số trong mỗi lần cập nhật.

validation_data=(X_val, y_val) xác định dữ liệu validation, trong đó X_val là các đặc trưng validation và y_val là nhãn tương ứng. Mô hình sẽ được đánh giá trên tập validation sau mỗi epoch để kiểm tra hiệu suất của mô hình trên dữ liệu không được sử dụng trong quá trình huấn luyện.

verbose=1 xác định cách hiển thị quá trình huấn luyện. Trong trường hợp này, giá trị là 1, vì vậy quá trình huấn luyện sẽ được hiển thị trên giao diện dòng lệnh.

Khi chạy đoạn mã trên, mô hình sẽ được huấn luyện trên tập huấn luyện và đánh giá trên tập validation trong

500 epoch. Quá trình huấn luyện sẽ được hiển thị trên giao diện dòng lệnh và kết quả của mỗi epoch sẽ được lưu trong biến history.

```
Epoch 1/500
651/651 [==============================] - 6s 7ms/step - loss: 0.0192 - val_loss: 0.0059
Epoch 2/500
651/651 [==============================] - 4s 6ms/step - loss: 0.0022 - val_loss: 0.0012
Epoch 3/500
651/651 [==============================] - 5s 8ms/step - loss: 0.0013 - val_loss: 0.0031
Epoch 4/500
651/651 [==============================] - 4s 6ms/step - loss: 0.0010 - val_loss: 7.8594e-04
Epoch 5/500
651/651 [==============================] - 4s 6ms/step - loss: 0.0010 - val_loss: 0.0025
Epoch 6/500
651/651 [==============================] - 4s 6ms/step - loss: 7.1118e-04 - val_loss: 4.6030e-04
Epoch 7/500
651/651 [==============================] - 4s 6ms/step - loss: 6.1644e-04 - val_loss: 7.0958e-04
Epoch 8/500
651/651 [==============================] - 5s 8ms/step - loss: 6.3439e-04 - val_loss: 0.0013
Epoch 9/500
651/651 [==============================] - 4s 6ms/step - loss: 5.6864e-04 - val_loss: 0.0012
Epoch 10/500
651/651 [==============================] - 4s 6ms/step - loss: 5.7428e-04 - val_loss: 6.1582e-04
Epoch 11/500
651/651 [==============================] - 5s 8ms/step - loss: 4.8257e-04 - val_loss: 9.2208e-04
Epoch 12/500
651/651 [==============================] - 4s 6ms/step - loss: 4.2570e-04 - val_loss: 5.7823e-04
Epoch 13/500
...
Epoch 499/500
651/651 [==============================] - 4s 7ms/step - loss: 1.1304e-05 - val_loss: 2.1011e-05
Epoch 500/500
651/651 [==============================] - 5s 8ms/step - loss: 1.5612e-05 - val_loss: 1.3527e-05
```

C. Đưa ra dự đoán và kết quả chính xác.

Dự đoán kết quả trên tập Validation

Sử dụng mô hình đã được huấn luyện để dự đoán kết quả trên tập dữ liệu validation.

Khi mô hình được huấn luyện trên tập dữ liệu huấn luyện, ta có thể sử dụng mô hình đó để dự đoán kết quả trên tập dữ liệu validation. Dự đoán này thường được thực hiện bằng cách sử dụng phương thức predict() trên đối tượng mô hình.

```
# Dự đoán kết quả tập Validation
y_pred = model.predict(X_val)
```

Kết quả của dự đoán này sẽ là một mảng (hoặc tensor) chứa các dự đoán cho mỗi mẫu trong tập validation. Bạn có thể sử dụng kết quả này để đánh giá hiệu suất của mô hình trên tập validation hoặc thực hiện các tác vụ khác liên quan đến dự đoán.

```
2602/2602 [==============================] - 4s 2ms/step
```

Đánh giá hiệu năng trên tập validation bằng MSE

Với:

yi là biến độc lập

yb là giá trị ước lượng

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Sử dụng hàm mean_squared_error từ module sklearn.metrics để tính toán giá trị mean squared error (MSE) giữa các dự đoán (y_pred) và nhãn thực tế (y_val) trên tập validation.

Hàm mean_squared_error nhận đầu vào là hai mảng (hoặc danh sách) y_val và y_pred, và tính toán MSE giữa chúng. MSE là một phép đo đánh giá mức độ sai khác trung bình giữa các giá trị dự đoán và giá trị thực tế, trong trường hợp này là trên tập validation.

Sau khi tính toán MSE, đoạn mã in ra giá trị MSE bằng cách sử dụng hàm print().

```
# Đánh giá hiệu năng trên tập validation bằng MSE
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_val, y_pred)
print("MSE:", mse)
```

```
MSE: 1.3527487590557588e-05
```

Đánh giá hiệu năng trên tập validation bằng RMSE

Sử dụng mô-đun numpy (np) để tính toán giá trị Root Mean Square Error (RMSE) từ giá trị Mean Squared Error (MSE) đã tính toán trước đó.

$$RMSE = \sqrt{\sum_{i=1}^{n}\frac{(\hat{y}_i - y_i)^2}{n}}$$

Với:

yî là giá trị ước lượng
yi là biến độc lập
n=(N – k – 1)
N : số tổng lượng quan sát
K : tổng lượng biến

RMSE là căn bậc hai của MSE và được sử dụng để đo lường sự sai khác trung bình giữa các giá trị dự đoán và giá trị thực tế. Để tính toán RMSE, ta có thể sử dụng hàm sqrt() từ mô-đun numpy để tính căn bậc hai của MSE.

```
# Đánh giá hiệu năng trên tập validation bằng RMSE
import numpy as np
rmse = np.sqrt(mean_squared_error(y_val, y_pred))
print("RMSE trên tập validation:", rmse)
```

Đoạn mã trên sẽ tính RMSE giữa y_val (nhãn thực tế) và y_pred (dự đoán) trên tập validation và in ra giá trị RMSE.

```
RMSE trên tập validation: 0.0036779732993263542
```

IV. ĐÁNH GIÁ KẾT QUẢ

Dựa trên phần thực hiện và kết quả có thể nhận xét các điểm sau:

- **Dữ liệu:** Dữ liệu đầu vào có 166,475 mẫu và 11 thuộc tính. Dữ liệu này được sử dụng để huấn luyện mô hình mạng học sâu.
- **Mô hình mạng học sâu:** Mô hình được thiết kế với kiến trúc tuần tự (sequential) và bao gồm nhiều lớp dense (fully connected). Mỗi lớp thực hiện một phép tính toán riêng. Tổng số tham số của mô hình là 99,522.
- **Huấn luyện mô hình:** Mô hình được huấn luyện trong 500 epochs với batch size là 128. Trong quá trình huấn luyện, hàm mất mát (loss function) được sử dụng là mean squared error (MSE). Mô hình được huấn luyện trên tập huấn luyện và được đánh giá trên tập validation.
- **Đánh giá mô hình:** Kết quả đánh giá mô hình trên tập validation cho thấy giá trị root mean squared error (RMSE) là khoảng 0.00368. Kết quả này cho thấy mô hình có độ chính xác tương đối cao trong việc ước lượng giá trị ngõ ra.
- **Các chỉ số đánh giá:** Đánh giá hiệu suất của mô hình dựa trên các chỉ số mean squared error (MSE - sai số bình phương trung bình) và root mean squared error (RMSE - căn bậc hai của sai số bình phương trung bình). Các chỉ số này cung cấp một đo lường về sự chênh lệch trung bình giữa giá trị dự đoán và giá trị thực tế. Giá trị MSE hoặc RMSE thấp hơn cho thấy hiệu suất tốt hơn và sự gần gũi hơn giữa các giá trị dự đoán và thực tế.
- **Quá trình huấn luyện:** Quá trình huấn luyện mô hình liên quan đến tối ưu hóa các tham số của mô hình để giảm thiểu hàm mất mát (MSE trong trường hợp này) thông qua backpropagation và gradient descent. Huấn luyện được thực hiện qua nhiều epochs, trong đó mỗi epoch đại diện cho một lần duyệt qua toàn bộ tập dữ liệu huấn luyện. Tập validation được sử dụng để theo dõi hiệu suất của mô hình và ngăn chặn việc quá khớp.
- **Hiện tượng quá khớp (overfitting):** Hiện tượng quá khớp xảy ra khi mô hình hoạt động tốt trên dữ liệu huấn luyện nhưng không thể tổng quát hóa cho dữ liệu mới, chưa được nhìn thấy trước. Đánh giá hiệu suất của mô hình trên tập validation riêng biệt là quan trọng để phát hiện hiện tượng quá khớp. Nếu hiệu suất của mô hình trên tập validation tệ hơn đáng kể so với tập huấn luyện, có thể chỉ ra hiện tượng quá khớp. Các kỹ thuật regularization, chẳng hạn như dropout hoặc weight decay, có thể được áp dụng để giảm thiểu hiện tượng quá khớp.
- **Độ phức tạp của mô hình:** Kiến trúc của mô hình học sâu được sử dụng trong đánh giá này bao gồm nhiều lớp dense (fully connected). Số lượng lớp, số đơn vị trong mỗi lớp và các lựa chọn kiến trúc khác có thể ảnh hưởng đến khả năng học của mô hình và khả năng tổng quát hóa. Quan trọng là đạt được sự cân bằng giữa độ phức tạp của mô hình và hiện tượng quá khớp, vì mô hình quá phức tạp có nguy cơ quá khớp cao hơn.
- **Khả năng tổng quát hóa:** Phần đánh giá được cung cấp ở trên tập trung vào hiệu suất của mô hình trên tập validation. Quan trọng là đánh giá hiệu suất của mô hình trên các tập dữ liệu bổ sung hoặc dữ liệu thực tế khác để xác định khả năng tổng quát hóa. Một mô hình hoạt động tốt trên các tập dữ liệu đa dạng sẽ có khả năng cung cấp dự đoán đáng tin cậy trong các tình huống khác nhau.
- **Giải thích kết quả:** Kết quả của việc đánh giá chỉ ra rằng mô hình đạt được giá trị RMSE thấp trên tập validation, cho thấy nó có khả năng dự đoán chính xác. Tuy nhiên, quan trọng là giải thích kết quả trong ngữ cảnh của nhiệm vụ hoặc ứng dụng cụ thể. Ý nghĩa và tính hữu ích của hiệu suất mô hình phụ thuộc vào yêu cầu và ràng buộc cụ thể

của lĩnh vực vấn đề.

- **Xem xét để cải thiện thêm:** Kết quả đánh giá có thể được sử dụng làm điểm khởi đầu cho phân tích và cải thiện tiếp theo. Nếu hiệu suất của mô hình không đạt yêu cầu, có thể khám phá nhiều khía cạnh, như điều chỉnh siêu tham số của mô hình, thử các kiến trúc khác, tăng kích thước tập dữ liệu hoặc kết hợp các đặc trưng hoặc nguồn dữ liệu bổ sung.

V. TÀI LIỆU THAM KHẢO

[1] Classical Neural Networks: What hidden layers are there?, link: https://medium.com/swlh/classical-neural-networks-what-hidden-layers-are-there-9c0dc59a6d0f, ngày truy cập (09/12/2023).

[2] Artificial Neural Network (ANN) with Practical Implementation, link: https://medium.com/machine-learning-researcher/artificial-neural-network-ann-4481fa33d85a, ngày truy cập (09/12/2023).

[3] Flattening CNN layers for Neural Network and basic concepts,

link: https://medium.com/@muhammadshoaibali/flattening-cnn-layers-for-neural-network-694a232eda6a, ngày truy cập (09/12/2023).

[4] Deep Regression Neural Networks for Proportion Judgment, link: https://www.mdpi.com/1999-5903/14/4/100, ngày truy cập (09/12/2023).