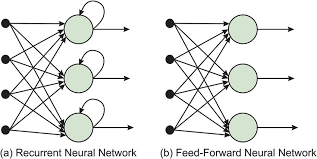
Sự khác nhau của RNN vs FFNN

Recurrent Neural Networks (RNN) và Feedforward Neural Networks (FFNN) là hai loại mạng nơ-ron khác nhau với cấu trúc và ứng dụng khác nhau. Dưới đây là sự khác nhau và ý nghĩa của từng loại mạng:



Cấu trúc

Feedforward Neural Networks (FFNN):

* Cấu trúc: FFNN có cấu trúc đơn giản hơn, nơi mà thông tin chỉ di chuyển theo một chiều, từ đầu vào qua các lớp ẩn đến đầu ra. Không có vòng lặp trong mạng.
* Hoạt động: Các đơn vị nơ-ron trong mỗi lớp chỉ liên kết với các đơn vị ở lớp kế tiếp. Mạng không có trạng thái bên trong để lưu trữ thông tin từ các bước trước.

Recurrent Neural Networks (RNN):

* Cấu trúc: RNN có cấu trúc phức tạp hơn với các vòng lặp, cho phép thông tin được lưu trữ và sử dụng lại trong các bước tiếp theo của mạng. Mỗi nơ-ron trong lớp ẩn có thể nhận đầu vào từ chính nó hoặc từ các nơ-ron khác trong cùng lớp.
* Hoạt động: Các đơn vị nơ-ron trong RNN có trạng thái bên trong (hidden state) được cập nhật tại mỗi bước thời gian, cho phép mạng nhớ thông tin từ các bước trước đó.

Ý nghĩa

Feedforward Neural Networks (FFNN):

* Ứng dụng: FFNN thường được sử dụng cho các bài toán phân loại và hồi quy đơn giản, nơi mà mỗi đầu vào độc lập với nhau. Chúng thường được áp dụng trong các bài toán như nhận dạng hình ảnh, phân loại văn bản, và các tác vụ không phụ thuộc vào trình tự.
* Đơn giản và hiệu quả: FFNN dễ thiết kế và huấn luyện hơn vì không có trạng thái bên trong hoặc thông tin phản hồi, giúp giảm bớt độ phức tạp trong việc tính toán và tối ưu hóa.

Recurrent Neural Networks (RNN):

* Ứng dụng: RNN phù hợp với các bài toán liên quan đến dữ liệu tuần tự hoặc chuỗi thời gian, chẳng hạn như dự báo chuỗi thời gian, xử lý ngôn ngữ tự nhiên (NLP), nhận dạng giọng nói, và dịch máy. Khả năng nhớ thông tin trước đó giúp RNN xử lý tốt các bài toán phụ thuộc vào ngữ cảnh.
* Khả năng ghi nhớ: RNN có khả năng lưu giữ thông tin từ các bước thời gian trước đó, cho phép chúng phân tích và dự đoán dữ liệu tuần tự một cách hiệu quả hơn.

So sánh cụ thể:

Trạng thái:

* FFNN: Không có trạng thái bên trong, mỗi đầu vào được xử lý độc lập.
* RNN: Có trạng thái bên trong, giữ thông tin từ các bước trước đó.

Khả năng xử lý:

* FFNN: Xử lý tốt các bài toán không phụ thuộc vào trình tự.
* RNN: Xử lý tốt các bài toán có phụ thuộc vào trình tự và ngữ cảnh.

Độ phức tạp:

* FFNN: Đơn giản và dễ huấn luyện.
* RNN: Phức tạp hơn, có thể gặp vấn đề về gradient (như vanishing gradient) khi huấn luyện với các chuỗi dài.

### ****Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP)****:

* **Phân loại văn bản**: RNN có thể phân loại văn bản dựa trên nội dung, chẳng hạn như phân loại email spam, phân loại cảm xúc (sentiment analysis), và nhận diện chủ đề (topic detection).
* **Dịch máy (Machine Translation)**: Dịch văn bản từ ngôn ngữ này sang ngôn ngữ khác bằng cách sử dụng mô hình RNN hai chiều (encoder-decoder).
* **Tóm tắt văn bản (Text Summarization)**: Tạo ra một phiên bản ngắn gọn của một văn bản dài hơn.
* **Nhận diện thực thể có tên (Named Entity Recognition - NER)**: Xác định các thực thể quan trọng (như tên người, địa điểm, tổ chức) trong văn bản.

### 2. ****Nhận dạng giọng nói (Speech Recognition)****:

* **Chuyển đổi giọng nói thành văn bản (Speech-to-Text)**: Chuyển đổi tín hiệu âm thanh thành văn bản bằng cách phân tích chuỗi tín hiệu âm thanh.
* **Hiểu lệnh giọng nói**: Hiểu và xử lý các lệnh giọng nói trong các ứng dụng như trợ lý ảo (Siri, Google Assistant).

### 3. ****Dự báo chuỗi thời gian (Time Series Forecasting)****:

* **Dự báo tài chính**: Dự đoán giá cổ phiếu, tỷ giá hối đoái, và các chỉ số kinh tế dựa trên dữ liệu lịch sử.
* **Dự báo thời tiết**: Dự đoán các yếu tố thời tiết như nhiệt độ, lượng mưa, và gió dựa trên dữ liệu thời gian trước đó.
* **Dự báo nhu cầu**: Dự đoán nhu cầu sản phẩm trong tương lai để quản lý hàng tồn kho và chuỗi cung ứng.

### 4. ****Phân tích chuỗi thời gian (Time Series Analysis)****:

* **Phân tích hành vi người dùng**: Phân tích dữ liệu hành vi người dùng trong các ứng dụng như e-commerce để hiểu và dự đoán các xu hướng mua sắm.
* **Phân tích dữ liệu IoT**: Phân tích dữ liệu từ các thiết bị Internet of Things (IoT) để phát hiện bất thường và dự đoán bảo trì.

### 5. ****Tạo ra dữ liệu (Data Generation)****:

* **Sinh văn bản (Text Generation)**: Tạo ra văn bản mới dựa trên mẫu đã học, như viết tiếp câu chuyện hoặc sinh văn bản tự động.
* **Sinh nhạc (Music Generation)**: Tạo ra các bản nhạc mới dựa trên các mẫu âm nhạc đã học.

### 6. ****Phân tích video và hình ảnh tuần tự****:

* **Nhận diện hoạt động (Activity Recognition)**: Nhận diện và phân loại các hoạt động trong video, chẳng hạn như đi bộ, chạy, hoặc lái xe.
* **Chú thích video (Video Captioning)**: Tạo các mô tả văn bản cho các đoạn video.

### 7. ****Ứng dụng y tế****:

* **Dự đoán bệnh tật**: Dự đoán diễn biến của bệnh dựa trên dữ liệu y tế theo thời gian, chẳng hạn như dữ liệu từ máy theo dõi sức khỏe.

### 8. ****Hệ thống khuyến nghị (Recommendation Systems)****:

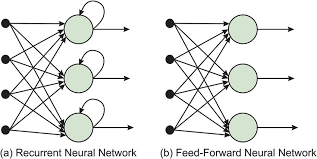
* **Khuyến nghị sản phẩm**: Đề xuất sản phẩm dựa trên lịch sử duyệt web và hành vi mua sắm của người dùng.
  1. Feed Forward Neural Network

Trong bài viết này, chúng ta sẽ tìm hiểu về Mạng nơ-ron tiếp nối (feedforward Neural Networks- FFNN), còn được gọi là Mạng tiếp nối sâu hoặc Perceptron nhiều lớp (Deep feedforward Networks, Multi-layer Perceptrons - MLP). Chúng tạo thành nền tảng của nhiều Mạng thần kinh quan trọng đang được sử dụng trong thời gian gần đây, chẳng hạn như CNN (được sử dụng rộng rãi trong các ứng dụng thị giác máy tính), RNN (được sử dụng rộng rãi trong hiểu ngôn ngữ tự nhiên và học theo trình tự).

Feed Forward Neural Network (FFNN) là một loại mạng neural cơ bản và phổ biến trong học sâu. Đây là loại mạng neural đơn giản nhất, trong đó dữ liệu di chuyển theo một hướng từ đầu vào qua các lớp ẩn đến đầu ra mà không có chu kỳ (loop).

A diagram of a machine

Description automatically generated



* + 1. Cấu trúc của FFNN

2.1.1.1 Các tầng - Layers

FFNN bao gồm ba loại lớp chính:

* **Lớp đầu vào (Input Layer):** Nhận dữ liệu đầu vào. Số lượng nút trong lớp này bằng với số lượng tính năng trong dữ liệu đầu vào.
* **Lớp ẩn (Hidden Layers):** Các lớp giữa đầu vào và đầu ra. Chúng thực hiện các phép biến đổi và tính toán phức tạp. Một FFNN có thể có một hoặc nhiều lớp ẩn.
* **Lớp đầu ra (Output Layer):** Cung cấp đầu ra của mạng. Số lượng nút trong lớp này tùy thuộc vào bài toán cụ thể (ví dụ: một nút cho bài toán hồi quy, nhiều nút cho bài toán phân loại đa lớp).

A group of blue circles with black arrows

Description automatically generated

Số lượng layer trong một MLP được tính bằng số hidden layers + output layer. Như hình trên số lượng layers trong MLP: L = 3.

2.1.1.2 Nodes – Units

Một node hình tròn trong một layer được gọi là một unit. Đầu vào của các hidden layer được ký hiệu bởi z, đầu ra của mỗi unit thường được ký hiệu là a.

A diagram of a network

Description automatically generated

* Trong đó:
  + Đầu ra của unit thứ i trong layer thứ được ký hiệu
  + là ma trận trọng số thể thể hiện liên kết layer từ đến
  + b là độ lệch. Khi bỏ bias, phương trình giờ có dạng: w1\*x + w2∗y=0, sẽ luôn đi qua gốc tọa độ và nó không tổng quát hóa phương trình đường thẳng nên có thể không tìm được phương trình mong muốn

2.1.1.3 Activation functions

Các hàm kích hoạt có thể là các hàm phi tuyến khác nhau, như là hàm sigmoid hoặc hàm tanh. Điều quan trọng là các hàm kích hoạt phải là phi tuyến(non-linear), vì nếu không, nhiều layer hoặc một layer cũng sẽ không có sự khác biệt. Ví dụ, nếu hàm kích hoạt là một hàm tuyến tính (giả sử hàm f(s) = s), thì cả hai layer có thể được thay thế bằng một layer duy nhất với ma trận hệ số . Ở phía dưới, chúng mình sẽ giới thiệu:

* Sigmoid
* Tanh
* Relu
* Softmax

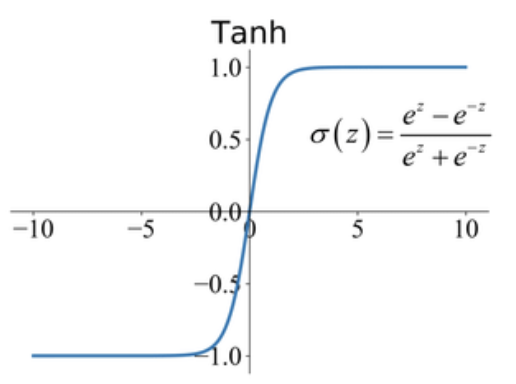
Hàm sigmoid được sử dụng cho hồi quy logistic hai lớp. Nó được sử dụng để phân loại xem một mẫu nhất định là Đúng hay Sai dựa trên xác suất được tính toán. Mẫu được phân loại là True nếu giá trị lớn hơn 0,5 và ngược lại. Nếu đầu vào lớn, hàm số sẽ cho đầu ra gần với 1. Với đầu vào nhỏ (rất âm), hàm số sẽ cho đầu ra gần với 0. Một vấn đề dễ nhận thấy là khi giá trị đầu vào rất lớn (âm hoặc dương), gradient của hàm số sẽ gần bằng 0. Điều này có nghĩa là các hệ số tương ứng với đơn vị đang xét sẽ gần như không được cập nhật (**vanishing gradient problem**).

A graph of a function

Description automatically generated

Khác với Sigmoid giá trị chạy trong khỏi từ (0,1) thì tanh chạy trong khỏi (-1, 1).

Tanh: tanh(z) =



ReLU (Rectified Linear Unit) được sử dụng rộng rãi gần đây vì tính đơn giản của nó. Công thức:

ReLU(z)=max(0,z)

* Với = z
* Với = 0

A graph of a function

Description automatically generated A graph of a function

Description automatically generated with medium confidence

Ở 2 hình trên so sánh sự hội tụ của SGD khi sử dụng hai activation function khác nhau: ReLU và tanh. Cải thiện tốc độ tính toán được cho là do ReLU có thể tính toán gần như ngay lập tức và gradient của nó cũng được tính toán nhanh chóng, với giá trị gradient bằng 1 nếu đầu vào lớn hơn 0 và bằng 0 nếu đầu vào nhỏ hơn 0. Hàm ReLU có nhiều biến thể khác như Noisy ReLU, Leaky ReLu, ELUs.

Tiếp theo, softmax activation function thường được sử dụng trong lớp đầu ra của các mạng neural để giải quyết các bài toán phân loại đa lớp (multiclass classification). Softmax chuyển đổi một vector các giá trị thành một vector các xác suất, trong đó tổng tất cả các xác suất là 1.

Cho 1 vector đầu vào:

Công thức Softmax:

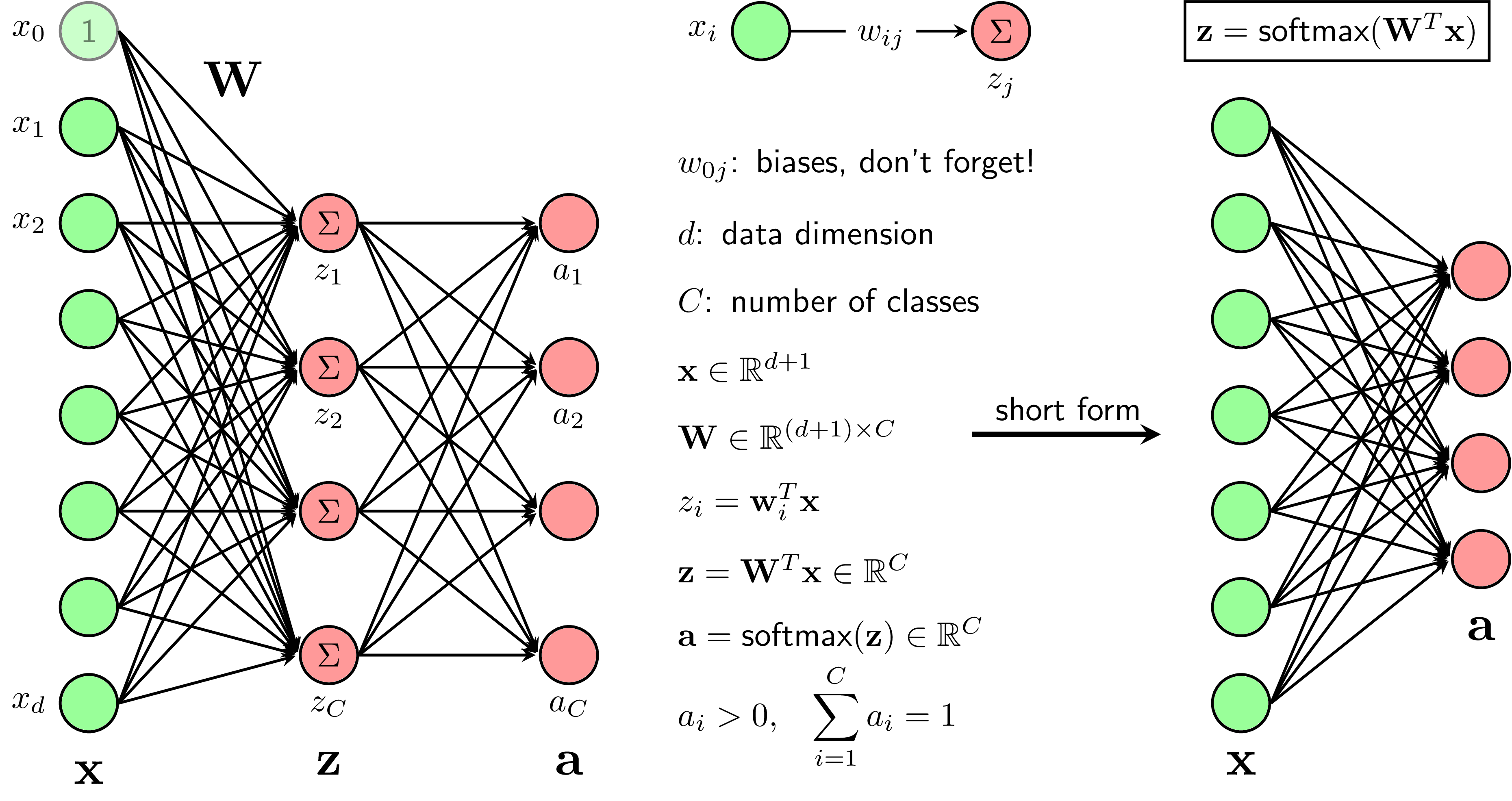
Trong đó :

* thể hiện xác suất để input đó rơi vào class i, với lớn hơn 0 và có tổng bằng 1
* , càng lớn thì xác suất dữ liệu rơi vào class i càng cao. có thể nhận giá trị âm hoặc dương.
* là một hàm mượt, có nghĩa là nó liên tục và khả vi vô hạn. Nghĩ là là 1 giá trị dương thì sẽ trở thành 1 hàm đồng biến(đơn điệu tăng) với tăng thì tăng.
* A graph of a function

  Description automatically generated

Ta có thể viết thành

* được hiểu là xác suất để một điểm dữ liệu x rơi vào class thứ i nếu biết tham số mô hình (ma trận trọng số) là W

Hình vẽ dưới đây thể hiện mạng Softmax Regression dưới dạng neural network: 

2.1.2 Quá trình hoạt động của FFNN

**Lan truyền tiến (Forward Propagation):** Dữ liệu đầu vào di chuyển qua các lớp của mạng, mỗi lớp áp dụng một phép biến đổi tuyến tính và sau đó là một hàm kích hoạt (activation function). Kết quả của mỗi lớp trở thành đầu vào cho lớp tiếp theo.

A math equations on a white background

Description automatically generated

* Trong đó:
* 𝑥 là vector đầu vào.
* 𝑊 là ma trận trọng số.
* 𝑏 là vector hệ số chặn (bias).
* 𝜎 là hàm kích hoạt.

**Hàm kích hoạt (Activation Function):** Áp dụng một hàm phi tuyến để giới thiệu tính phi tuyến vào mô hình, giúp mạng học được các mối quan hệ phi tuyến trong dữ liệu.

Các hàm kích hoạt phổ biến:

* Sigmoid: ​
* Tanh: tanh(z) =
* ReLU: ReLU(z)=max(0,z)

2.1.3 Hàm mất mát (Loss Function):

Hàm mất mát đo lường sự khác biệt giữa giá trị dự đoán của mạng và giá trị thực tế. Mục tiêu của việc huấn luyện là tối thiểu hóa hàm mất mát.

Một số hàm mất mát phổ biến:

* Mean Squared Error (MSE) cho bài toán hồi quy.
* A math equations and symbols

  Description automatically generated with medium confidence
* Với (X, Y) là cặp dữ liệu trong training
* Cross-Entropy Loss cho bài toán phân loại: Như vậy, Entropy của tín hiệu này có phân phối P (tính kì vọng dựa vào P) nhưng lại được mã hóa dựa vào phân phối Q.
* A black text on a white background

  Description automatically generated
* Tính chất của CrossEntropy
  + - Đạt giá trị minimu P == Q
    - Rất nhạy cảm với sự sai khác giữa và . Khi càng khác nhau thì cross entropy càng tăng. Vì vậy bài toán cần xây dựng output càng gần target càng tốt.
    - Công thức Binary cross entropy loss – BCE

A black text on a white background

Description automatically generated

Ví dụ: tính BCE, hình 1 với output và target gần nhau và hình 2 thì ngược lại.

A math equations with numbers

Description automatically generated

A math equations with numbers

Description automatically generated

2.1.4 Lan truyền ngược (Backpropagation):

Lan truyền ngược là quá trình điều chỉnh trọng số của mạng dựa trên lỗi của dự đoán. Nó bao gồm hai bước chính:

* **Lan truyền ngược gradient:** Tính gradient của hàm mất mát đối với từng trọng số bằng cách sử dụng quy tắc chuỗi. Gradient này cho biết mức độ ảnh hưởng của từng trọng số đến lỗi.
* **Cập nhật trọng số:** Điều chỉnh trọng số bằng cách sử dụng gradient đã tính toán. Quá trình này thường sử dụng một thuật toán tối ưu hóa như Gradient Descent.
* Quy tắc cập nhật trọng số:

A black and white math symbols

Description automatically generated with medium confidence

* Trong đó:
* η là tốc độ học (learning rate).

* ​ là gradient của hàm mất mát đối với trọng số.

Cụ thể ta cần tính được 2 thứ đó, tách nhỏ ra từ công thức ở trên. Với W, b là tập hợp ma trận trọng số giữa các layers và biases của mỗi layer:

A black and white math symbols

Description automatically generated with medium confidence

Trong đó có cần biết đến đạo hàm hợp Chain rule:



2.1.5 Thuật toán tối ưu hóa

Các thuật toán tối ưu hóa được sử dụng để cập nhật trọng số nhằm giảm thiểu hàm mất mát. Các thuật toán phổ biến bao gồm:

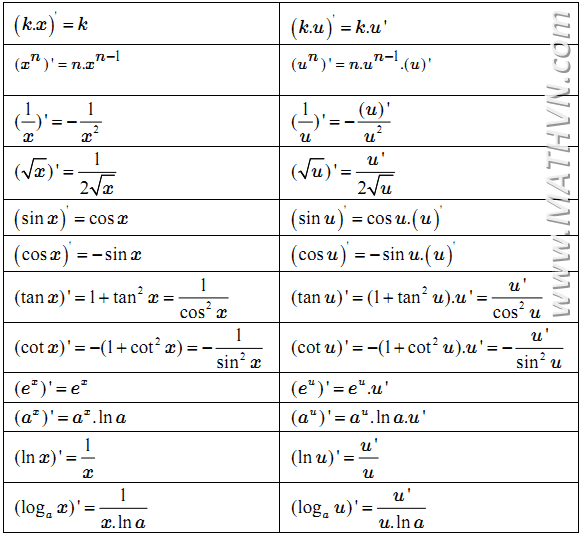
* **Gradient Descent**
* **Stochastic Gradient Descent (SGD)**
* **Adagrad**
* **RMSprop**
* **Adam**

2.1.6 Ví dụ chạy

Ở ví dụ này xin mượn ví dụ của trang nttuan8 để nói về backpropagation.

Link: https://nttuan8.com/bai-4-backpropagation/

Công thức đạo hàm:



Mô hình

* A diagram of a network

  Description automatically generated
* Với L = 2
* Mô hình: 2-2-1, nghĩa là 2 node trong input layer, 1 hidden layer có 2 node và output layer có 1 node.

Tiếp theo là công thức tính chạy trong feedforward:

* A math equations on a white background

  Description automatically generated

Nó sẽ có dạng tổng quát:

* A group of math equations

  Description automatically generated

Từ đó chúng ta dùng hàm loss: Binary cross entropy loss – BCE

* A math equations on a white background

  Description automatically generated
* A number and symbol on a white background

  Description automatically generated
* là giá trị model dự đoán (train), còn là giá trị thực tế

Dựa vào mô hình ở trên, tính toán W, b của tầng 1 và 2. Vì khi backpropagration thì tầng 2 sẽ tính trước rồi tới tầng 1. Ở trên đã giới thiệu đến đạo hàm hợp Chain rule.

Trước khi vô đạo hàm W, b của từng tầng thì có chúng mình có để giải thích thêm hình ở bên dưới:

Giải thích từ dấu = thứ nhất, thì nó là công thức sigmoid, sau khi đạo hàm thì cộng và trừ 1 với tử. Nhóm nhân tử chung lại.

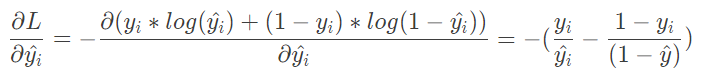
Dựa vào đây có thể giải thích được đến bước ảnh ở dưới.

Tiếp theo chúng ta sẽ đạo hàm Loss với W, b tầng 2.

Áp dụng chain rule: A math equation with black text

Description automatically generated with medium confidence

Ta tính được đạo hàm loss với kết quả dự đoán.

* + 

A diagram of a graph

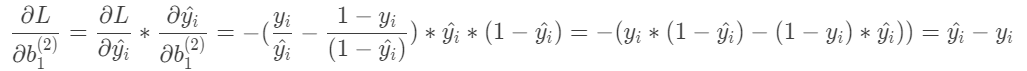
Description automatically generated

Dựa vào tính toán ở trên kia.

A math equations on a white background

Description automatically generated

Khi đó:



Tương tự với ở trên thế vào là được: tương tự với đạo L với a.

A math equations with numbers and symbols

Description automatically generated with medium confidence

Tiếp theo chúng ta sẽ đạo hàm Loss với W, b tầng 1.



Áp dụng chain rule: A mathematical equation with numbers and lines

Description automatically generated

Ta có:

A math equations on a white background

Description automatically generated

Công thức đạo hàm a1 với b1 (l=1) thì tương tự, với B.

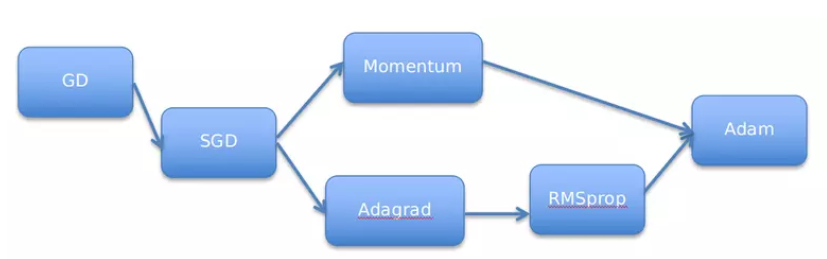
A group of math equations

Description automatically generated

* 1. Các phương pháp optimization

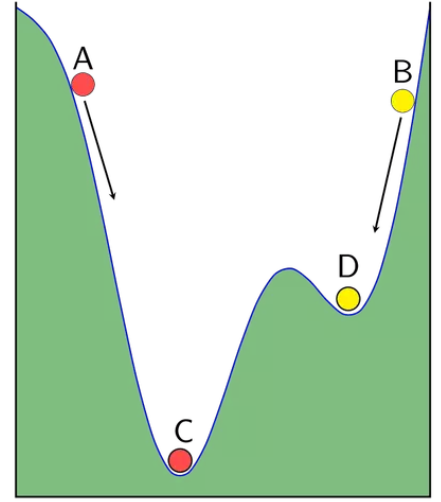
Trước khi đi sâu vào vấn đề, chúng ta cần hiểu thuật toán tối ưu (optimizers) là gì và chúng được sử dụng để làm gì. Thuật toán tối ưu là công cụ quan trọng trong việc xây dựng mô hình neural network. Chúng giúp mô hình **"học"** được các **đặc trưng (features) và mẫu (patterns)** từ dữ liệu đầu vào. Mục tiêu là tìm ra một cặp **weights** và **bias** phù hợp để tối ưu hóa hiệu suất của mô hình.

Nhưng làm thế nào để "học" được weights và bias? Không thể chỉ đơn giản là chọn ngẫu nhiên một số lần hữu hạn và hy vọng rằng ở một bước nào đó sẽ tìm được lời giải. Cách làm này không khả thi và lãng phí tài nguyên. Do đó, chúng ta cần một phương pháp hệ thống để cải thiện weights và bias theo từng bước. Đây chính là lý do mà các thuật toán tối ưu ra đời: để hướng dẫn quá trình điều chỉnh weights và bias sao cho mô hình ngày càng chính xác hơn trong việc dự đoán.



2.2.1 Gradient Descent

Gradient Descent cổ điển: Cập nhật tham số sau khi tính toán gradient dựa trên toàn bộ tập dữ liệu.

* Dự đoán 1 điểm khởi tạo:
* Cập nhật đến khi đạt được kết quả chấp nhận được:
* là đạo hàm của hàm mất mát tại
* Ưu Điểm:
* Đơn Giản và Dễ Hiểu: Thuật toán Gradient Descent cơ bản rất dễ hiểu và triển khai. Nó cập nhật trọng số sau mỗi vòng lặp dựa trên gradient của hàm loss, giúp tối ưu hóa mô hình neural network.
* Hiệu Quả: Bằng cách cập nhật trọng số dần dần, Gradient Descent giúp mô hình học được từ dữ liệu và cải thiện độ chính xác của dự đoán.
* Nhược Điểm:
* Phụ Thuộc Vào Điểm Khởi Tạo Ban Đầu: Vị trí khởi tạo của điểm ban đầu có thể ảnh hưởng lớn đến quá trình hội tụ. Nếu chọn điểm x ban đầu khác nhau, kết quả cuối cùng có thể khác nhau, đặc biệt là trong các hàm có nhiều cực trị.
* Phụ Thuộc Vào Learning Rate:
  + - Learning Rate Quá Nhỏ: Nếu tốc độ học quá nhỏ, quá trình hội tụ sẽ rất chậm, làm ảnh hưởng đến thời gian huấn luyện mô hình. Mô hình có thể mất nhiều thời gian để đạt được kết quả tốt.
    - Learning Rate Quá Lớn: Nếu tốc độ học quá lớn, mô hình có thể tiến nhanh đến gần điểm tối ưu chỉ sau vài vòng lặp. Tuy nhiên, nó có thể không hội tụ và dao động xung quanh điểm tối ưu mà không đạt được giá trị tối ưu chính xác, do bước nhảy quá lớn.
* Phụ Thuộc Vào Nghiệm Khởi Tạo: Ví dụ, nếu hàm số có nhiều điểm cực trị toàn cục (global minimum), tùy thuộc vào điểm khởi tạo ban đầu, thuật toán có thể hội tụ đến các nghiệm cuối cùng khác nhau(local minimum). Điều này có thể dẫn đến các kết quả không nhất quán.
* 

2.2.2 Stochastic Gradient Descent

Cập nhật tham số dựa trên một mẫu ngẫu nhiên (mini-batch) từ tập dữ liệu.

* Mỗi lần duyệt một lượt qua tất cả các điểm trên toàn bộ dữ liệu được gọi là một **epoch**. Với **GD thông thường** thì mỗi **epoch** ứng với **1 lần** cập nhật θ, với **SGD** thì mỗi epoch ứng với **N lần cập nhật** θ với N là số điểm dữ liệu (sau mỗi epoch, chúng ta cần shuffle (xáo trộn) thứ tự của các dữ liệu để đảm bảo tính ngẫu nhiên).
* Nhìn vào một mặt, việc cập nhật từng điểm một như thế này có thể làm giảm đi tốc độ thực hiện 1 epoch. Nhưng nhìn vào một mặt khác, SGD chỉ yêu cầu một lượng epoch rất nhỏ (thường là 10 cho lần đầu tiên, sau đó khi có dữ liệu mới thì chỉ cần chạy dưới một epoch là đã có nghiệm tốt).
* là hàm mất mát với chỉ 1 cặp điểm dữ liệu (input, label) là

A black and white image of a black circle with a red line

Description automatically generated

GD có hạn chế đối với cơ sở dữ liệu lớn (vài triệu dữ liệu) thì việc tính toán đạo hàm trên toàn bộ dữ liệu qua mỗi vòng lặp trở nên cồng kềnh. Bên cạnh đó GD không phù hợp với **online learning**.

* Online learning: là khi dữ liệu cập nhật liên tục (ví dụ như thêm người dùng đăng kí) thì mỗi lần thêm dữ liệu ta phải tính lại đạo hàm trên toàn bộ dữ liệu => thời gian tính toán lâu, thuật toán không online nữa. Vì thế SGD ra đời để giải quyết vấn đề đó, vì mỗi lần thêm dữ liệu mới vào chỉ cần cập nhật trên 1 điểm dữ liệu đó thôi, phù hợp với online learning.
* Ưu điểm :
* Thuật toán giải quyết được đối với cơ sở dữ liệu lớn mà GD không làm được. Thuật toán tối ưu này hiên nay vẫn hay được sử dụng.
* Nhược điểm :
* Thuật toán vẫn chưa giải quyết được 2 nhược điểm lớn của gradient descent ( learning rate, điểm dữ liệu ban đầu )

2.2.2 Gradient Descent với Momentum

Trong Gradient Descent (GD), chúng ta cần tính lượng thay đổi tại thời điểm t để cập nhật vị trí mới của nghiệm. Nếu coi đại lượng này như vận tốc trong vật lý, vị trí mới sẽ là:

Dấu trừ biểu thị việc di chuyển ngược với gradient. Nhiệm vụ của chúng ta là tính sao cho nó chứa cả thông tin về độ dốc (gradient) và đà (momentum) tức vận tốc trước (với vận tốc ban đầu ). .

A diagram of a graph

Description automatically generated

Ở hình b, với GD thì nếu chạy từ A thì có thể đến C là (global minimum) còn nếu từ B thì nó chỉ chạy đến D(local minimum). Còn với hình c, có 1 điểm khác là khi chạy từ B thì nó sẽ dựa vào công thức (vận tốc trước đó và đà ) sẽ giúp nó vượt qua đỉnh gần đó sẽ tới điểm E rồi tới C.

* Ưu điểm : Thuật toán tối ưu giải quyết được vấn đề: Gradient Descent không tiến được tới điểm global minimum mà chỉ dừng lại ở local minimum.
* Nhược điểm: Tuy momentum giúp hòn bi vượt dốc tiến tới điểm đích, tuy nhiên khi tới gần đích, nó vẫn mất khá nhiều thời gian giao động qua lại trước khi dừng hẳn, điều này được giải thích vì viên bi có đà.

2.2.4 Adagrad

Khác với các thuật toán trước đó, trong quá trình huấn luyện, learning rate thường không thay đổi (được xem là một hằng số). Tuy nhiên, Adagrad coi learning rate là một tham số và cho phép nó biến thiên sau mỗi thời điểm t.

A square root of a square root of a square

Description automatically generated

* Trong đó :
* n : hằng số
* gt : gradient tại thời điểm t
* ϵ : hệ số tránh lỗi ( chia cho mẫu bằng 0), default 1e-8
* G : là ma trận chéo mà mỗi phần tử trên đường chéo (i,i) là bình phương của đạo hàm vectơ tham số tại thời điểm t.
  + A diagram of a mathematical equation

    Description automatically generated
* Ưu điểm: Một lơi ích dễ thấy của Adagrad là tránh việc điều chỉnh learning rate bằng tay, chỉ cần để tốc độ học default là 0.01 thì thuật toán sẽ tự động điều chỉnh.
* Nhược điểm: Yếu điểm của Adagrad là tổng bình phương biến thiên sẽ lớn dần theo thời gian cho đến khi nó làm tốc độ học cực kì nhỏ, làm việc training trở nên đóng băng.

2.2.5 RMSprop

RMSprop giải quyết vấn đề giảm tốc độ học của Adagrad bằng cách chia tốc độ học cho trung bình của bình phương gradient.

A mathematical equation with numbers and symbols

Description automatically generated

* Ưu điểm: Giải quyết được vấn đề tốc độ học giảm dần của Adagrad (vấn đề tốc độ học giảm dần theo thời gian sẽ làm quá trình đào tạo chậm lại, có khả năng dẫn đến trì trệ).
* Nhược điểm: Thuật toán RMSprop có thể chỉ tìm ra các giải pháp tối thiểu cục bộ thay vì đạt đến mức tối thiểu toàn cầu như Momentum. Do đó, người ta kết hợp cả thuật toán Momentum và RMSprop để tạo ra thuật toán tối ưu mang tên Adam. Chúng ta sẽ thảo luận về nó trong phần sau.

2.2.5 Adam

Như đã đề cập ở phía trên, Adam là sự kết hợp của Momentum và RMSprop. Nếu diễn giải theo hiện tượng vật lí, Momentum tương tự như quả cầu lao xuống dốc, trong khi Adam giống như quả cầu rất nặng có ma sát (friction). Do đó, Adam dễ dàng vượt qua local minimum để đạt tới global minimum và khi đạt tới global minimum, nó không mất nhiều thời gian dao động quanh điểm cuối vì có ma sát, dễ dừng lại hơn.

A graph of a ball with a number of points

Description automatically generated

Công thức update:

* A math equations on a white background

  Description automatically generated
  1. RNN (Recurrent Neural Networks)

Recurrent Neural Networks (RNN) là một loại mạng neural có kiến trúc cho phép xử lý dữ liệu tuần tự và duy trì thông tin qua các bước thời gian. Khả năng này làm cho RNN trở nên hữu ích trong các ứng dụng liên quan đến dữ liệu tuần tự như xử lý ngôn ngữ tự nhiên, phân tích chuỗi thời gian, và nhận dạng giọng nói.

A diagram of a machine

Description automatically generated

2.3.1. Cấu trúc của RNN

RNN có cấu trúc đặc biệt với các kết nối tuần hoàn (recurrent connections), cho phép thông tin được truyền qua các bước thời gian. Mỗi nút trong mạng không chỉ nhận đầu vào từ các bước thời gian trước mà còn từ trạng thái ẩn (hidden state) của chính nó ở bước thời gian trước đó.

* **Lớp đầu vào (Input Layer)**: Nhận dữ liệu tuần tự.
* **Lớp ẩn (Hidden Layer)**: Lưu giữ trạng thái ẩn, cập nhật qua mỗi bước thời gian.
* **Lớp đầu ra (Output Layer)**: Cung cấp đầu ra của mạng.

2.3.2. Lan truyền tiến (Forward Propagation) trong RNN

Ở mỗi bước thời gian t, RNN nhận đầu vào xt​ và trạng thái ẩn trước đó ht−1. Trạng thái ẩn hiện tại ht​ được tính toán như sau:

A black text on a white background

Description automatically generated

Trong đó:

* Whh là ma trận trọng số cho trạng thái ẩn.
* Wxh là ma trận trọng số cho đầu vào.
* bh là hệ số chặn (bias).
* σ là hàm kích hoạt (thường dùng hàm tanh hoặc ReLU).

Đầu ra yt​ được tính như sau:

A black letter on a white background

Description automatically generated

Trong đó:

* Why là ma trận trọng số cho đầu ra.
* by​ là hệ số chặn cho đầu ra.
* ϕ là hàm kích hoạt đầu ra (thường dùng hàm softmax cho phân loại).

2.3.3. Hàm mất mát (Loss Function)

Hàm mất mát trong RNN thường được tính toán dựa trên toàn bộ chuỗi đầu ra so với chuỗi mục tiêu. Đối với các bài toán phân loại, hàm mất mát phổ biến là Cross-Entropy Loss.

2.3.4. Lan truyền ngược qua thời gian (Backpropagation Through Time - BPTT)

Lan truyền ngược qua thời gian là phương pháp mở rộng của lan truyền ngược (Backpropagation) để áp dụng cho RNN. BPTT tính toán gradient của hàm mất mát đối với các trọng số của mạng qua tất cả các bước thời gian:

1. **Lan truyền tiến**: Tính toán trạng thái ẩn và đầu ra cho mỗi bước thời gian.
2. **Tính toán lỗi**: Đo lường lỗi của đầu ra so với giá trị thực.
3. **Lan truyền ngược**: Tính toán gradient của lỗi đối với từng trạng thái ẩn và trọng số, sau đó cập nhật trọng số bằng cách sử dụng một thuật toán tối ưu hóa (ví dụ: Gradient Descent).

2.3.5. Các vấn đề và giải pháp trong RNN

**Vanishing Gradient**: Gradient có thể trở nên rất nhỏ qua các bước thời gian dài, làm cho mạng khó học các phụ thuộc dài hạn.

**Exploding Gradient**: Gradient có thể trở nên rất lớn, làm cho việc cập nhật trọng số không ổn định.

**Giải pháp**:

* **LSTM (Long Short-Term Memory)**: Một loại RNN với các cổng đặc biệt để kiểm soát luồng thông tin và duy trì thông tin dài hạn.
* **GRU (Gated Recurrent Unit)**: Một biến thể đơn giản hơn của LSTM với hiệu suất tương tự.
* **Gradient Clipping**: Kỹ thuật để giới hạn giá trị của gradient nhằm tránh exploding gradient.