ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

MẠNG NƠRON NHÂN TẠO

Tài liệu tham khảo để hiện thực bài tập lớn
Môn học Cấu trúc dữ liệu và Giải thuật (CO2003)

TP. HỒ CHÍ MINH, THÁNG 09/2024

Neural Networks

Lê Thành Sách

Mục lục

1 Dữ liệu và phân tích dữ liệu

1.1 Dữ liệu và dữ liệu số

Dữ liệu là một dạng biểu diễn thông tin. Trong thực tiễn, dữ liệu có thể tồn tại/lưu trữ ở các dạng vật lý khác nhau như giấy và phim nhựa. Ví dụ, giấy và các phim nhựa chứa bản viết hay bản in của văn bản, hình ảnh, và các biểu đồ. Dạng dữ liệu này *chưa sẵn sàng* để được xử lý và phân tích bởi máy tính nhằm cung cấp những tiện ích cho người dùng. Do đó, một trong những nhiệm vụ của **chuyển đổi số** mà nhiều quốc gia đang quan tâm là chuyển đổi sang dạng số [1] cho các dạng lưu trữ tuyền thống đang có, cũng những áp dụng những công nghệ và quy trình mới để dữ liệu tạo phải ở sẵn dạng số.

Dữ liệu được quan tâm trong lĩnh vực Trí tuệ nhân tạo và Khoa học dữ liệu là dữ liệu số, ví dụ như tập tin hình ảnh, âm thanh, và video (ở tất cả các định dạng); các tập tin văn bản và các tập tin định dạng CSV và Excel, v.v. Từ đây, khi đề cập đến dữ liệu thì đó là dữ liệu số.

1.2 Ứng dụng dựa trên dữ liệu

Dựa trên các dạng dữ liệu số được cung cấp, lĩnh vực Trí tuệ nhân tạo (Ariticial Intelligence, AI) có mục đích là phân tích dữ liệu này để từ đó **hiểu** dữ liệu và ra các quyết định tương ứng để có thể *thay thế* hay *hỗ trợ* con người. Một số ứng dụng điển hình như sau:

1. Ứng dụng chạy trên iPhone:
    - Cài đặt giờ: Để đặt giờ hẹn, người dùng có thể ra lệnh bằng giọng nói cho Siri như "set an alarm for 4PM". Siri sẽ nhận dạng, hiểu câu lệnh và xác nhận giờ hẹn. Nếu không hiểu Siri sẽ hỏi lại các bạn.
    - Tương tự người dùng có thể bật/tắt đèn pin của iPhone bằng câu lệnh "Lumos" và "Nox" tương ứng.
2. Với ChatGPT và các phiên bản liên quan: người dùng có thể tương tác với ChatGPT thông qua hỏi-đáp, để yêu cầu nó thực hiện nhiều việc, ví dụ như sau.
    - Sửa chữa lại bản viết để rõ nghĩa hơn.
    - Dịch qua lại giữa các ngôn ngữ.
    - Hỏi đáp với nhiều nội dung phong phú khác.

[1] Dữ liệu số (digital data): dữ liệu được biểu diễn bởi chuỗi các **bit** 0 và 1; được lưu trữ ở các tập tin trong máy tính

3. Định danh khuôn mặt và ứng dụng: Hiện nay xác thực bằng khuôn mặt đã có độ chính xác cao và dựa trên đó, có nhiều ứng dụng như:
    - Mở khoá màn hình, mở/đóng cửa/cổng vật lý hay cổng trên phần mềm (nghĩa là xác thực tài khoản).
    - Chuyển khoản trong nhân hàng
    - Định danh và truy vết tội phạm.

Ứng dụng hay vận dụng các công nghệ AI vào thực tế rất phong phú; tuy nhiên, tất cả ứng dụng như vậy phải dựa vào những bài toán **cốt lõi** của trong ngành Trí tuệ nhân tạo.

Theo truyền thống, ngành Trí tuệ nhân tạo có những hướng nghiên cứu chuyên biệt để cố gắng **sao chép** được những khả năng của con người. Các khả năng của con người được quan tâm gồm:

1. Khả năng về thị giác (**Computer Vision**): khả năng hiểu được hình ảnh và video;
2. Khả năng về thính giác (**Voice recognition and Synthesis**): bao gồm, nhận dạng và tổng hợp tiếng nói.
3. Khả năng ngôn ngữ (**Natural Language Processing**): khả năng hiểu và dùng được ký hiệu như con người.
4. Khả năng vận động (**Robotics**): khả năng vận động như con người; bao gồm các ứng dụng điển hình như robot người, xe tự hành, v.v.
5. Khả năng suy diễn (**Reasoning**): khả năng thực hiện các loại suy diễn như con người.
6. Khả năng học (**Learning**): khả năng học từ dữ liệu. Đây là khả năng quan trọng nhất, phân biệt con người và các động vật khác. Tiêu biểu theo hướng này là các kỹ thuật học máy (machine learning và học sâu (deep learning).

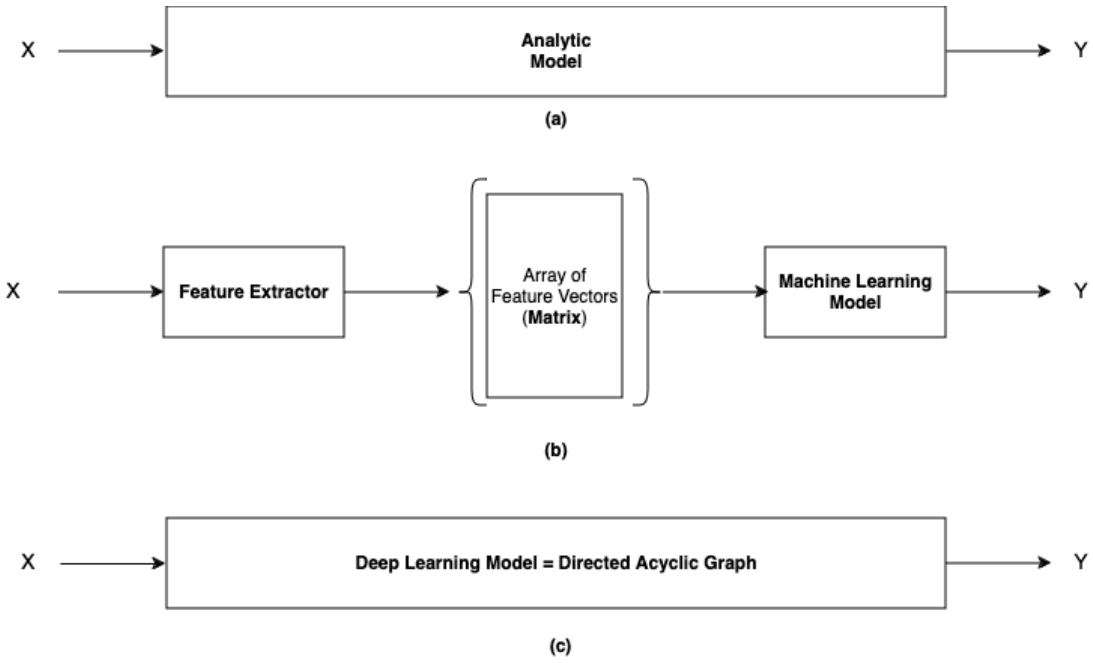
1.3 Tổng quan về phân tích dữ liệu

Ở mức tổng quan, việc phân tích dữ liệu được minh hoạ trong Hình 1 (a). Ở đó, X là dữ liệu đầu vào, có thể là tập tin hình ảnh, video, âm thanh, và văn bản; còn Y là đầu ra của việc phân tích. Hãy xem xét một số ứng dụng sau đây để hiểu rõ hơn về X và Y.

- Nếu X là đoạn văn bản bình luận về một sản phẩm và mô hình phân tích là **phân loại** xem bình luận này thuộc nhóm nào sau đây: "trung tính", "tiêu cực", và "tích cực". Trong trường hợp này Y sẽ là một trong ba giá trị được liệt kê ở trên.
- Nếu X là hình ảnh của con vật mô hình phân tích là xác định xem hình ảnh con vật trong ảnh X là loại nào sau đây (**phân loại**): "Chó", "Mèo", và "Gà". Trong trường hợp

này Y sẽ là một trong ba giá trị được liệt kê ở trên.

- Nếu X là hình ảnh chứa một hay nhiều con vật và con người. Các con vật có thể là "Chó", "Mèo", và "Gà". Nếu mô hình phân tích là **phát hiện** xem các đối tượng (người và các con vật) nằm ở đâu trên khung hình. Trong trường hợp này Y sẽ gồm:
  1. **Về vị trí:** Danh sách các các hộp giới hạn (hình chữ nhật) để cho biết đối tượng ở đâu trên hình. Mỗi hộp sẽ được mô tả bởi 4 giá trị: c_x, c_y, w, h, tương ứng là toạ độ tâm của hộp trên ảnh, độ rộng và chiều cao của hộp.
  2. **Về phân loại:** Mỗi hộp giới hạn ở trên được đính kèm một giá trị (trong tập: "Con người", "Chó", "Mèo", và "Gà") để cho biết hộp đó chứa người hay con vật nào.
- Nếu X là đoạn văn trong ngôn ngữ Anh và mô hình là dịch ngôn ngữ, cụ thể là dịch đoạn văn X sang tiếng Việt, thì Y là đoạn văn tương ứng trong ngôn ngữ Tiếng Việt.



Hình 1: Mô hình phân tích dữ liệu

1.3.1 Dữ liệu, nhãn, mẫu dữ liệu

Trong thực tế, để phát triển các mô hình học máy chúng ta cần thu thập dữ liệu và gán nhãn cho chúng. Dữ liệu và nhãn của nó là các cặp $< X, Y >$ như vừa được trình bày ở trên. Cụ thể X được gọi là dữ liệu và Y được gọi là nhãn.

Ví dụ, chúng ta cần phân văn bản vào một trong các loại: "trung tính", "tích cực" và "tiêu cực". Gọi X là một đoạn văn bản nào đó thu thập được. Chúng ta phải dùng con người làm

mẫu, **gán nhãn cho X là loại nào trong 3 loại ở trên**. Giá trị thu được từ người làm mẫu [2] chúng ta gọi nhãn của X, ký hiệu là Y (hay t).

Việc dùng máy tính để học từ dữ liệu và nhãn có thể được hiểu như sau: chúng ta dùng con người làm mẫu cho một số lượng đủ lớn dữ liệu và bắt máy tính học để "copy" quá trình làm việc của con người. Hay nói cách khác, theo toán học, là xây dựng một hàm toán để ánh xạ từ X sang Y.

Hãy xem xét một số ví dụ sau:

- Với bài toán phân loại bình luận ở trên, chúng ta cần thu thập bộ dữ liệu gồm nhiều bình luận và gán nhãn ("trung tính", "tiêu cực", và "tích cực") cho mỗi bình luận. Giả sử ta có được $N = 1000$ bình luận, mỗi bình luận là một đoạn văn bằng Tiếng Việt. Ta nói, ta có N mẫu dữ liệu (**sample**). Vì mỗi mẫu dữ liệu được gán một nhãn, nên sau khi gán nhãn (làm nhãn) ta được một danh sách của N cặp, mỗi cặp gồm bình luận và nhãn của bình luận đó.
- Với bài toán phát hiện đối tượng trong ảnh ở trên, giả sử chúng ta thu thập được $N = 10^6$ hình, hay 10^6 mẫu dữ liệu. Quá trình làm nhãn là vẽ hộp giới hạn cho mỗi con vật và con người có trong ảnh, cũng như gán nhãn loại cho hộp giới hạn. Do đó, nhãn của mỗi hình là danh sách các hộp giới hạn và tên nhãn phân loại cho hộp.
- Với bài toán dịch ngôn ngữ ở trên, giả sử chúng ta thu thập được $N = 10^3$ đoạn văn trong Tiếng Anh. Quá trình làm nhãn là dịch chúng sang Tiếng Việt. Do đó, ta nói ta có một tập dữ liệu gồm N cặp, mỗi cặp gồm một đoạn văn trong Tiếng Anh và một đoạn văn cùng nghĩa trong Tiếng Việt.

1.3.2 Phương pháp truyền thống

Thực tế rằng, các dạng dữ liệu của X (Hình 1 (a)) là rất phong phú, có thể là văn bản, hình ảnh, video, âm thanh, tập tin Excel và CSV. Do đó ý tưởng chủ đạo của việc phân tích bao gồm hai bước cơ bản:

1. **Rút trích đặc trưng:** Biểu diễn mỗi mẫu dữ liệu của X bằng một vector mô tả cho mẫu dữ liệu. Vector này được gọi là vector đặc trưng (**feature vector**). Cụ thể, vector đặc trưng chỉ là một dãy con con số thực [3].
   Vector đặc trưng của N mẫu dữ liệu được ghép thành một ma trận đặc trưng. Mỗi hàng của ma trận này là vector đặc trưng của mẫu dữ liệu tương ứng. Nhãn của dữ liệu cũng

[2] Người làm mẫu, người làm nhãn: annotator
[3] Thường dùng các thư viện sau đây để biểu diễn dạng vector: **Numpy**, **Pytorch**, **Tensorflow**, **xtensor**

được biểu diễn thành một vector có N phần tử hay ma trận có N hàng. Nhãn của mẫu dữ liệu thứ k là hàng thứ k trong ma trận đặc trưng, và nhãn của nó là phần tử thứ k trong vector nhãn hay hàng thứ k của ma trận nhãn.

2. **Sử dụng các kỹ thuật trong học máy:** Sử dụng các kỹ thuật có trong lĩnh vực học máy để thực hiện việc phân tích dữ liệu. Cụ thể, thực hiện các công việc như phân loại, hồi quy, gôm cụm, v.v. Một số kỹ thuật tiêu biểu bao gồm, mạng nơron truyền thẳng nhiều lớp (MultiLayer Perceptron, MLP), Support Vector Machines (SVM), Dicision Trees and Random Forest, Naive Bayes, v.v. Môn học "Học máy" sẽ cung cấp cho sinh viên hiểu biết và vận dụng được các kỹ thuật như vậy.

Để có được biểu diễn dạng vector cho nhiều loại dữ liệu khác nhau, trước đây khoảng trên 12 năm, các nhà nghiên cứu hay phát triển ứng dụng sẽ xây dựng mới hay dùng lại các giải thuật có sẵn để thực hiện việc rút trích và biểu diễn này, xem Hình 1 (b). Các môn học "Xử lý ảnh và Thị giác Máy tính", "Xử lý tín hiệu số", "Xử lý ngôn ngữ tự nhiên", ngày đó dạy cho người học các kỹ thuật khác nhau để trích và biểu diễn đặc trưng của từng dạng dữ liệu hình ảnh/video, âm thanh và văn bản.

Cho dù là dạng dữ liệu ở đầu vào là gì thì bộ trích đặc trưng (Feature Extractor), Hình 1 (b) cũng có biểu diễn vào dạng chung, đó là ma trận của các đặc trưng; từng hàng trong ma trận đó là vector đặc trưng của mẫu dữ liệu. Chính vì vậy, các thư viện học máy truyền thống như **sklearn**[4] đều yêu cầu đầu vào cho các kỹ thuật là ma trận như trong Hình 1 (b).

1.3.3 Phương pháp hiện đại

Từ năm 2012, khởi đầu là mô hình có tên là AlexNet[5], học máy đã chuyển sang một hướng **dễ dàng hơn** và **mạnh mẽ hơn** để phân tích dữ liệu. Đó là, thay vì dùng các hàm được thiết kế đặc biệt để rút trích đặc trưng thì xu hướng mới "**học**" luôn vector đặc trưng cho các mẫu dữ liệu thô (gốc). Kỹ thuật như vậy có tên là **học sâu (deep learning)**. Do đó, học sâu có khả năng học từ dữ liệu đầu vào ra kết quả ở đầu ra, xem Hình 1(c). Thực tế, nếu phân tích các mô hình học sâu hiện nay thì chúng có thể được tách thành 2 phần nối tiếp: (a) phần học vector đặc trưng cho các điểm dữ liệu và (b) phần thực hiện các bài toán (còn gọi là nhiệm vụ) cốt lõi của học máy.

Trên góc nhìn tính toán thì mô hình hâu sâu chỉ là **một đồ thị có hướng và không có vòng, Driected Acyclic Graph (DAG)**. Ở đó, các nút trong đồ thị có thể là nút chứa dữ liệu **đầu vào** (in-degree =0), **đầu ra** (out-degree =0), hoặc các **nút tính toán**; còn các cạnh

[4] **sklearn**: Thư viện của các kỹ thuật học máy truyền thống.
[5] **AlexNet**: ImageNet Classification with Deep Convolutional Neural Networks

có mục đích định hướng dòng dữ liệu chạy trong đồ thị.

Do đó, nếu cho sẵn dữ liệu ở các nút đầu vào, chúng ta có thể theo dòng mũi tên của đồ thị thì có thể tính toán ra các kết quả trung gian và đáp ứng cuối cùng của mô hình. Với các DAG phức tạp thì chắc chắn chúng ta sẽ cần dùng giải thuật **TopologicalSort** để tuyến tính hoá các nút và theo thứ tự trong đầu ra của **TopologicalSort** để tính toán đáp ứng.

Mạng nơron truyền thẳng nhiều lớp (Multi-Layer Perceptron, MLP) là trường hợp đặc biệt và đơn giản của mô hình học sâu; ở đó, các nút tính toán đã được thiết kế thành một dãy, có thự tự trước sau. Do đó, việc tính toán đáp ứng đầu ra của MLP rất đơn giản, bằng cách thực hiện lần lượt việc tính toán cho từng nút có trong dãy. Phần 5 sẽ trình bày sâu hơn về những chủ đề trong MLP.

1.4 Các nhiệm vụ cốt lõi trong học máy và học sâu

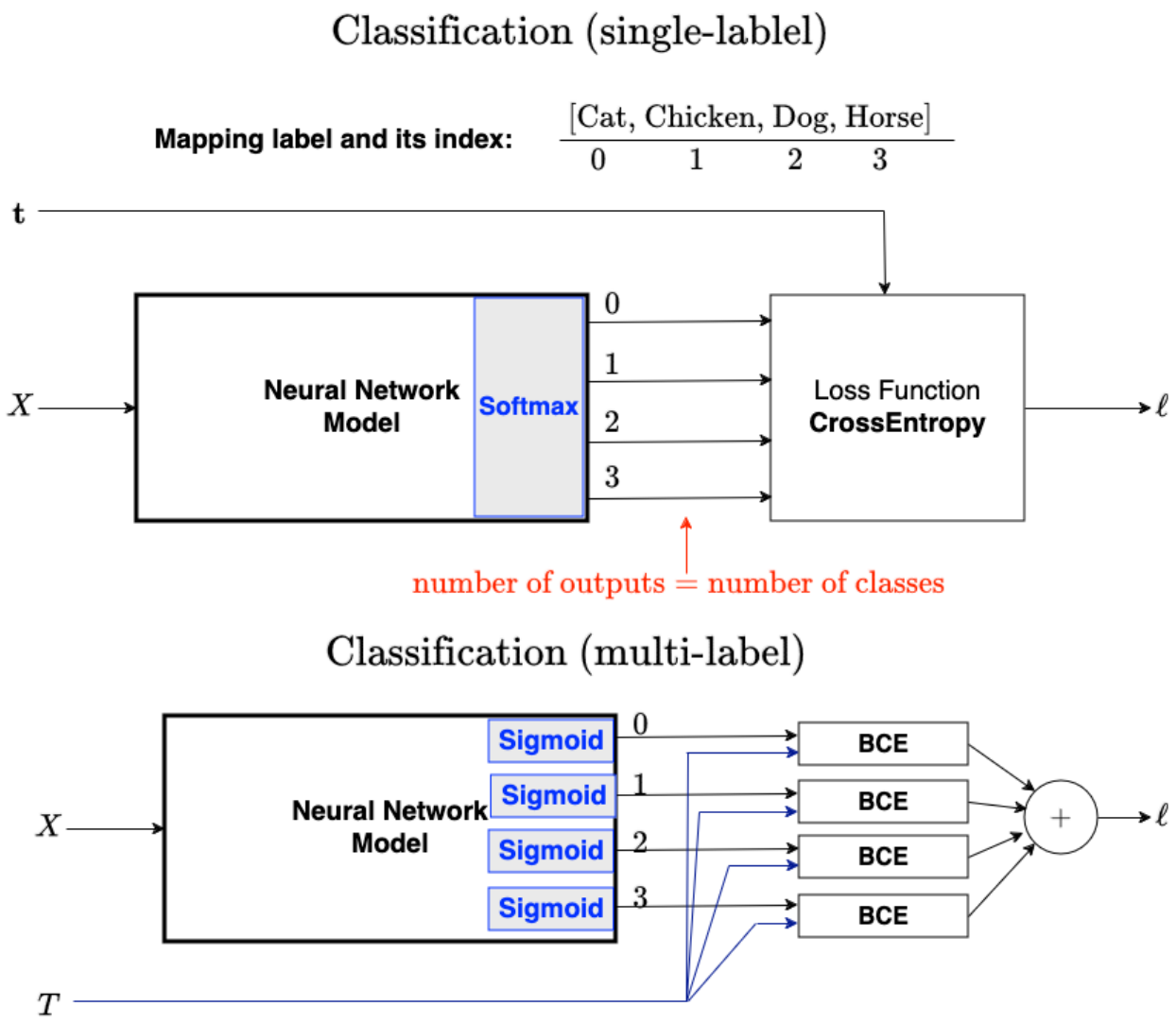
Khi phân tích dữ liệu trong Trí tuệ nhân tạo, chúng ta sẽ phân rã bài toán cần thực hiện thành các **nhiệm vụ cốt lõi**[6] trong lĩnh vực học máy và học sâu.

Các nhiệm vụ cốt lõi bao gồm: **phân loại**, **hồi quy** và **định danh/phân biệt**.

Xem xét một số ví dụ sau:

- Bài toán dịch Anh-Việt: bài toán này được quy về bài toán đoán từ; cụ thể, nếu đã dịch được k từ đầu tiên trong đoạn văn ở đầu ra Tiếng Việt và tất nhiên có sẵn đoạn văn bản trong Tiếng Anh, bài toán quan tâm ở đây là từ thứ $(k+1)$ ở đầu ra là từ nào trong tiếng Việt? Để giải quyết việc này, kỹ thuật hiện này là ước lượng một phân phối xác suất trên **D** từ của bộ từ vựng[7] Tiếng Việt. Về mặt toán học đó là $P(\mathbf{D} \,|\, \textbf{phần đã dịch có } k \textbf{ từ và đoạn văn trong Tiếng Anh})$. Đây là nhiệm vụ phân loại, và phân phối xác suất ở trên chính là đầu ra của hàm softmax, xem Phần 5.6.
- Bài toán phát hiện đối tượng trong ảnh: được quy về hai nhiệm vụ sau.
  - Hồi quy: có nhiệm vụ dự báo ra toạ độ của các hộp giới hạn;
  - Phân loại: dự báo ra loại của đối tượng nằm trong hộp giới hạn.

[6] **nhiệm vụ cốt lõi**: cũng được gọi là bài toán cốt lõi.
[7] **Bộ từ vựng (Vocabulary)** : là danh mục tất cả những từng có trong ngôn ngữ nào đó.



Hình 2: Mô hình phân loại

1.4.1 Phân loại

Nhiệm vụ của mô hình phân loại là chọn **một hay nhiều** tên trong tập các tên [8] để gán cho mẫu dữ liệu đầu vào. Nếu mô hình chỉ chọn một tên trong tập thì chúng ta gọi là **phân loại đơn nhãn** (**single-lalel classification**). Ngược lại, nếu mô hình có thể chọn từ 0 (không chọn) đến một số lượng tên nhiều hơn 1 thì ta gọi là **phân loại đa nhãn** (**multi-lalel classification**). Phân loại đơn nhãn là bài toán rất phổ biến khi nói về phân loại, nên khi chỉ đề cập là "phân loại" không kèm theo thông tin cụ thể thì đó là phân loại đơn nhãn.

Để thực hiện được nhiệm vụ phân loại, **đầu ra của mô hình mạng nơron** cần được

[8] còn được gọi là tên lớp hay tên loại. Ví dụ: [Cat, Chicken, Dog, Horse]

thiết kế theo cách như trong Hình 2. Cụ thể hơn, hãy xem giải thích sau đây.

- **Với phân loại đơn nhãn**: Gọi số lượng nhãn trong tập tên là K. Mô hình có những đặc điểm như sau:
  1. Mô hình có K ngõ ra. K giá trị này là xác suất của các tên trong tập hợp.
  2. Để có thể xuất ra các giá trị xác suất, mô hình có một lớp **Softmax** ở đầu ra (xem Phần 5.6), nhằm biến các dãy điểm số (giá trị) ở ngõ vào của nó thành phân phối xác suất [9].
  3. Các đầu ra của mô hình **luôn luôn** được đánh số là $0, 1, \cdots, (K-1)$. Các chỉ số này được ánh xạ sang tên gì trong tập tên là trách nhiệm của người phát triển và huấn luyện mô hình. Cách thông thường nhất là sắp xếp danh sách các tên theo thứ tự tăng dần trong bảng chữ cái, ta được danh sách có thứ tự. Lúc đó, chỉ số của tên trong danh sách cũng là chỉ số của đầu ra ở mô hình.
     - Trong Hình 2, danh sách có thứ tự của các tên là `[Cat, Chicken, Dog, Horse]`.Giả sử mô hình xuất ra phân phối xác suất sau: $[0.2, 0.5, 0.1, 0.2]$; ý nghĩa của dãy số này là, mô hình tin rằng có 20% khả năng nhãn của mẫu dữ liệu đầu là "Cat" cũng như "Horse". 50% khả năng là "Chicken" và chỉ 10% là "Dog".
     - Trong trường hợp, chúng ta cần mô hình đưa đến dự đoán cụ thể, thì chúng ta sẽ thực hiện hàm **argmax**[10] trên dãy số trên. **argmax**$([0.2, 0.5, 0.1, 0.2]) = 1$; số 1 cũng có nghĩa là "Chicken", mô hình dự báo là "Chicken".
  4. Sơ đồ huấn luyện: để huấn luyện mạng nơron làm nhiệm vụ phân loại, chúng ta thường kết nối đầu ra của mô hình với hàm tổn thất **Cross-Entropy** (xem Phần 6.1). Hàm này nhận vào hai đại lượng: (a) phân phối xác suất dự đoán của mô hình và (b) phân phối xác suất nhãn; nó tính toán và trả về giá trị số thực không âm. Bộ tối ưu sẽ dùng giá trị này để thực hiện các bước huấn luyện. Các phần theo sau sẽ giải thích rõ hơn về các bước còn lại.
- **Với phân loại đa nhãn**: Thay vì áp một hàm **Softmax** để tính phân phối trên toàn bộ các tên trong tập các lớp; mô hình phân loại đa nhãn sử dụng hàm **Sigmoid** (xem Phần 5.4) trên từng tên trong tập này. Hàm **Sigmoid** nhận vào một giá trị số thực và trả về giá trị thuộc khoảng $(0, 1)$. Với tập tên trong Hình 2, mô hình có chứa 4 hàm **Sigmoid** ở bên trong để xuất ra xác suất **có** mặt của các nhãn. Giả sử đầu ra của mô hình phân loại đa nhãn là dãy $[0.7, 0.3, 0.1, 0.8]$; chúng ta có thể giải thích như sau:
  - Lưu ý: tổng các con số trong dãy trên không bằng 1, vì chúng không phải là đầu ra của hàm **Softmax**.

[9] phân phối xác suất trên các tên (**categorical distribution**)
[10] **argmax**: thay vì trả về giá trị lớn nhất, hàm trả về chỉ số tương ứng giá trị lớn nhất.

- Giá trị đầu tiên là 0.7, nghĩa là 70% khả năng dữ liệu đầu vào **có nhãn** "Cat". Cũng lưu ý, $1 - 0.7 = 0.3$, nghĩa là 30% khả năng dữ liệu đầu vào **không có** "Cat" [11]. Các giá trị còn lại được diễn dịch tương tự.
- Trong trường hợp chúng ta cần mô hình ra quyết định dự báo cụ thể, chúng ta có thể so với các trị xác suất với một giá trị ngưỡng T, ví dụ, $T = 0.5$. Nếu các giá trị xuất suất lớn hơn hay bằng T thì ta bảo mô hình có dự báo ra nhãn tương ứng. Ví dụ, khi so các giá trị dự báo với T, ta thấy chỉ có 0.7 của "Cat" và 0.8 của "Horse" là thoả $>= T$. Do đó, dự báo của mô hình là "Cat" và "Horse"[12].

Sơ đồ huấn luyện: để huấn luyện mô hình phân loại đa nhãn, chúng ta thường kết nối đầu ra của dự báo vào hàm tổn thất **Binary Cross-Entropy**, như Hình 2. Các giá trị tổn thất riêng rẽ cho từng nhãn được tổng lại (có thể có nhân với hệ số) để tạo thành tổn thất toàn thể cho mẫu dữ liệu.

- **Với bài toán phân loại nhị phân (Two-Class classification)**: Phân loại nhị phân là bài toán phân loại khi tập tên chỉ gồm hai lớp, ví dụ như [Cat, Not-a-Cat], [Normal, Abnormal], [Cancer, not Cancer], v.v. Để giải bài toán này, chúng ta có hai cách tiếp cận, đó là:

  1. Dùng làm **Softmax**: chúng ta thiết kế mô hình như phân loại đơn nhãn trong Hình 2. Lúc này, $K = 2$ nên mô hình có hai ngõ ra.
  2. Dùng làm **Sigmoid**: chúng ta thiết kế mô hình chỉ có một ngõ ra, đó là xác suất của của một nhãn nào đó. Khi huấn luyện, chúng ta dùng hàm tổn thất **Binary Cross-Entropy**, xem Phần 6.2.

1.4.2 Hồi quy

Mô hình hồi quy là mô hình dự báo ra số thực. Một số ứng dụng trong thực tiễn,

- Dự báo ra lượng mưa, mức ngập trên đường, nhiệt độ, áp suất, v.v.
- Dự báo ra giá chứng khoán, giá vàng, v.v.
- Dự báo ra độ tuổi, nhịp tim, v.v.

Mạng noron làm nhiệm vụ hồi quy cũng có thiết kế đặc thù. Cụ thể, số lượng đầu ra bằng với số biến cần dự báo. Hàm tổn thất để huấn luyện mô hình hồi quy thường dùng là các hàm **Mean Equared Error, MSE (L2)** và **Mean Absolute Error, MAE (L1)**.

[11] nếu dữ liệu là ảnh, thì nghĩa là 70% khả năng là ảnh có chứa con mèo, và 30% khả năng không chứa mèo.
[12] trong trường hợp ứng dụng là phân loại xem âm thanh của con vật nào, thì dự báo này có nghĩa, bao gồm âm và con mèo và ngựa tín hiệu âm thanh đầu vào.

1.4.3 Định danh

To be updated!

2 Phương pháp đánh giá cho các nhiệm vụ cốt lõi

To be updated!

2.1 Cho nhiệm vụ phân loại

2.1.1 Ma trận nhầm lẫn (Confusion Matrix)

2.1.2 Độ chính xác (Accuracy)

2.1.3 Độ chính xác (Precision)

2.1.4 Độ triệu hồi (Recall)

2.1.5 Điểm số F1 (F1-Score)

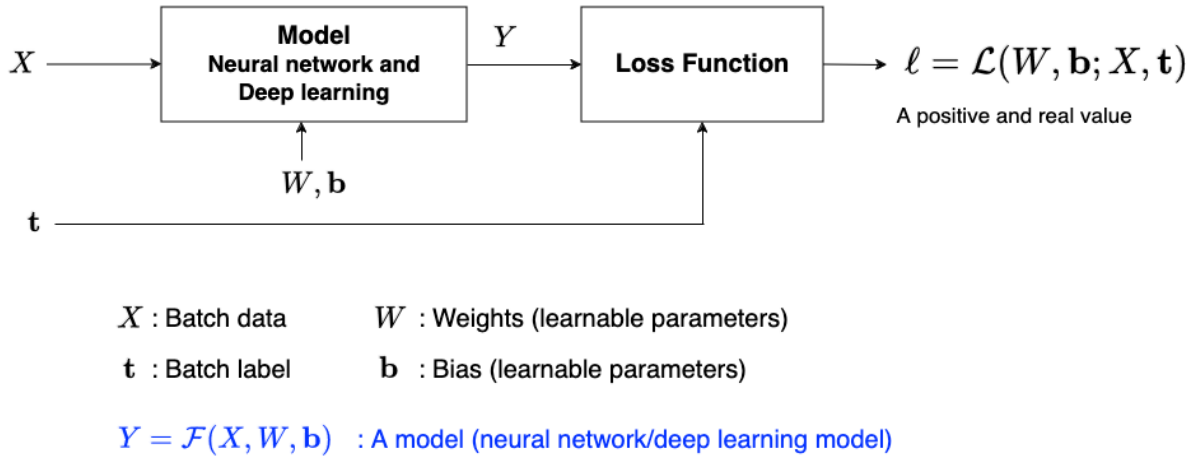
2.2 Cho nhiệm vụ hồi quy

2.2.1 Sai số trung bình bình phương (MSE)

2.2.2 Sai số trung bình tuyệt đối (MAE)

3 Mô hình toán cho mạng học sâu

Mô hình học sâu được xem như một hàm toán $Y = \mathcal{F}(X, W, \mathbf{b})$, xem Hình 3. Một mô hình học sâu có nhiều tham số có thể học là W và $\mathbf{b}$; W được gọi là **weights**, còn b được gọi là **bias**. Cả W và $\mathbf{b}$ đều được học từ dữ liệu; tuy nhiên, trong quá trình học W và $\mathbf{b}$ được đối xử khác nhau nên chúng được tách ra gọi tên riêng.



Hình 3: Mạng nơron và học sâu và sơ đồ huấn luyện

3.1 Quá trình Inference

Với một mô hình học sâu, sau khi đã xác định W và $\mathbf{b}$ từ dữ liệu, nghĩa là W và $\mathbf{b}$ đã sẵn sàng. Khi cung cấp X, nó tính $Y = \mathcal{F}(X, W, \mathbf{b})$; và bước này được gọi là **inference** (suy diễn).

3.2 Cơ sở toán của huấn luyện

Để xác định W và $\mathbf{b}$ của mô hình, chúng ta cần dữ liệu X và nhãn của các mẫu dữ liệu trong X, đó là $\mathbf{t}$.

Mô hình dự báo ra Y từ dữ liệu X, quá trình học cần so dự báo này với nhãn $\mathbf{t}$ để tìm ra sự khác biệt, sự khác biệt này được gọi là tổn thất (loss). Tổn thất là một con số thực dương. $\ell = \mathcal{L}(W, \mathbf{b}; X, \mathbf{t})$ [13]. Cách hiểu bốn tham số W, $\mathbf{b}$; X, $\mathbf{t}$ này như sau. Bên trái dấu ";" là W, $\mathbf{b}$; điều này có nghĩa là W, $\mathbf{b}$ là biến số của hàm $\mathcal{L}(W, \mathbf{b}; X, \mathbf{t})$. Bên phải của dấu ";" là X, $\mathbf{t}$; nghĩa là X và $\mathbf{t}$ được cung cấp khi tính giá trị tổn thất.

Như vậy bài toán huấn luyện mô hình học sâu trở thành bài toán kinh điển trong toán học; đó là, **tìm giá trị của W, $\mathbf{b}$ để hàm số $\mathcal{L}(W, \mathbf{b}; X, \mathbf{t})$ đạt cực tiểu**. Khi $\mathcal{L}(W, \mathbf{b}; X, \mathbf{t})$ đạt cực giá trị nhỏ nhất trên toàn bộ không gian của W và $\mathbf{b}$, ta nói hàm đạt cực tiểu **toàn cục** (global); ngược lại, khi $\mathcal{L}(W, \mathbf{b}; X, \mathbf{t})$ đạt cực giá trị nhỏ nhất trong một vùng hẹp của không gian W và $\mathbf{b}$, ta nói hàm đạt cực tiểu **cục bộ** (local). Cho đến hiện nay, chưa có một giải thuật đang được sử dụng nào có thể đảm bảo tìm ra điểm đạt cực tiểu toàn cục. Các giải thuật huấn luyện đang dùng là tìm ra cực tiểu cục bộ. Xin lưu ý, chúng ta thực sự mong muốn là giá trị W^* và $\mathbf{b}^*$ mà tại đó hàm $\mathcal{L}(W^*, \mathbf{b}^*; X, \mathbf{t})$ đạt cực tiểu thay vì giá trị cực tiểu của hàm

[13] $\ell = \mathcal{L}(W, \mathbf{b}; X, \mathbf{t})$: còn được viết là $\ell = \mathcal{L}(W, \mathbf{b}|X, \mathbf{t})$

$\mathcal{L}(W, \mathbf{b}; X, \mathbf{t})$.

Bài toán tìm cực tiểu không xa lạ với các học sinh phổ thông. Khi cho hàm số $f(w)$, cực tiểu w^* thoả mãn hai ràng buộc:

1. $f^{'}(w^*) = 0$
2. $f^{''}(w^*) > 0$

Với các hàm đơn giản, chỉ gồm có một biến, thì việc dùng giấy và bút để tính toán các đạo hàm bậc 1 và bậc 2 là khả thi. Tuy nhiên, với các mô hình học sâu, số lượng biến số (trong W và $\mathbf{b}$) có thể đến hàng tỉ. Do đó, không thể nào tính toán đạo hàm riêng theo kiểu dùng giấy bút và giải phương trình được. Thay vào đó, chúng ta sử dụng phương pháp số [14] để tính toán các đạo hàm riêng theo các biến. Phương pháp đó được trình bày trong các phần theo sau.

4 Giải thuật huấn luyện

4.1 Tổng quan

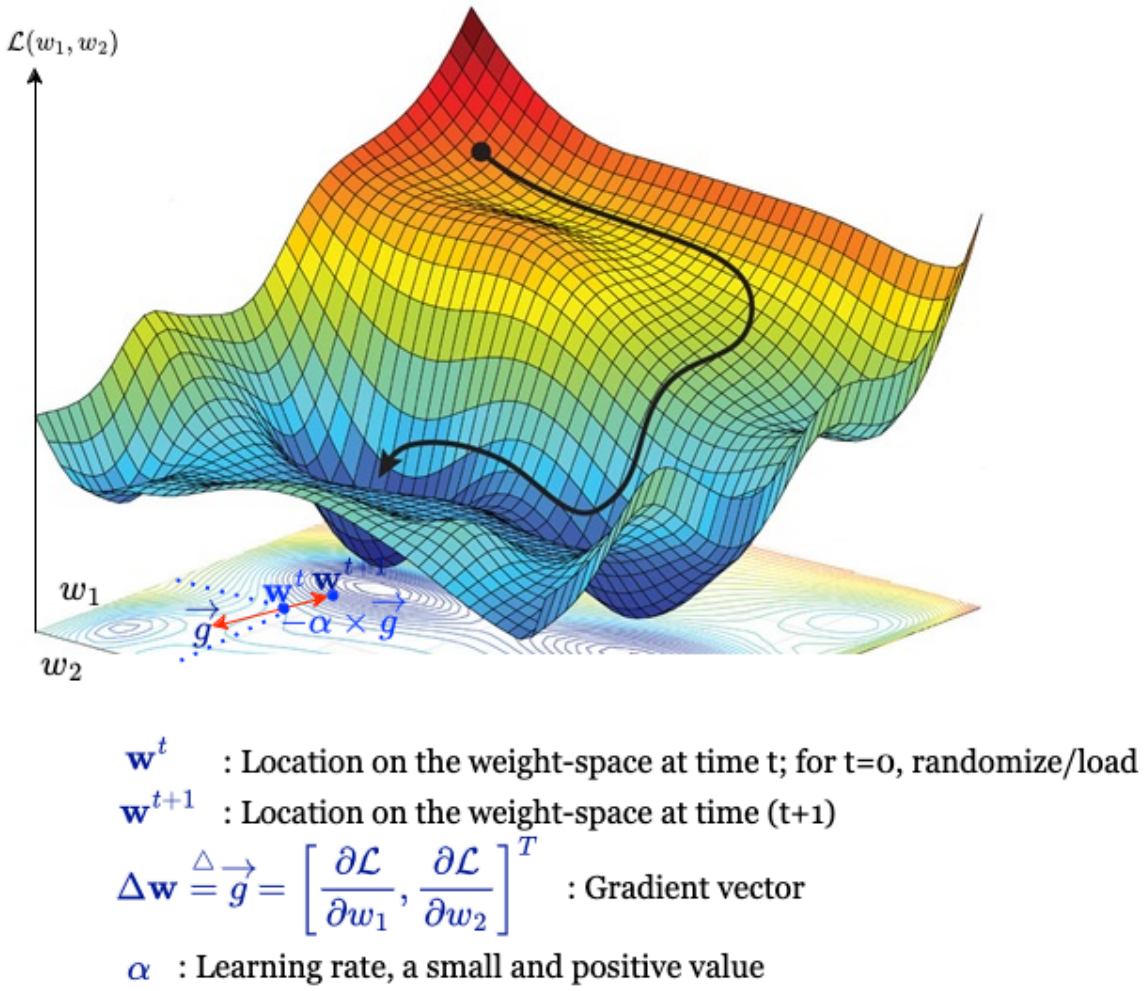
Mục tiêu của quá trình huấn luyện mang nơron học sâu là tìm ra bộ tham số W^* và $\mathbf{b}^*$, tại đó hàm tổn thất $\mathcal{L}(W^*, \mathbf{b}^*; X, \mathbf{t})$ đạt là đủ nhỏ. Xin lưu ý các điểm sau đây:

- $\mathcal{L}(W^*, \mathbf{b}^*; X, \mathbf{t})$ đạt đủ nhỏ thay vì chính xác là cực tiểu cục bộ hay toàn cục.
- Điều quan trọng là W^* và $\mathbf{b}^*$ tìm được làm cho mô hình có khả năng làm việc đúng đắn với dữ liệu không dùng để huấn luyện và kết quả inference của nó chấp nhận được trong thực tế.

Ý tưởng của giải thuật được tóm lược như sau:

1. Khởi động các giá trị **weights** và **bias**, trong W và $\mathbf{b}$, với các giá trị nào đó [15] . Hình 4 minh hoạ cho hàm tổn thất phụ thuộc vào hai tham số được học là w_1 và w_2. Trong trường hợp khởi động, $\mathbf{w}^t$ tương ứng là $\mathbf{w}^0$.
2. Lặp qua nhiều lần, mỗi lần như vậy thì hiệu chỉnh $\mathbf{w}^t$ về vị trí của $\mathbf{w}^{t+1}$ sao cho hàm tổn thất đạt giá trị nhỏ hơn tại điểm $\mathbf{w}^t$ và là nhỏ nhất trong một vùng cục bộ xung quanh điểm $\mathbf{w}^t$; nghĩa là:

[14] Phương pháp số: numerical methods
[15] Có hai cách thông dụng: (1), sinh số ngẫu nhiên, thường là nhỏ và có thể sinh ra dùng phân phối Gauss; (2), tải từ bộ số đã có sẵn



Hình 4: Minh hoạ cho hàm tổn thất. Nguồn hình: https://culturemachine.net/

- $\mathcal{L}(w_1^{t+1}, w_2^{t+1}) \leq \mathcal{L}(w_1^t, w_2^t)$
- $\mathcal{L}(w_1^{t+1}, w_2^{t+1})$: nhỏ nhất trong vùng cục bộ xung quang $\mathbf{w}^t$.

Để thực hiện bước số 2 ở trên, chúng ta cần dùng đến **gradient-vector** ($\vec{\mathbf{g}}$). **Gradient-vector** là vector chứa các đạo hàm riêng của hàm tổn thất $\mathcal{L}$ so với từng tham số cần phải học, nghĩa là các giá trị trong W và $\mathbf{b}$.

Gradient vector có tính chất là nếu trong không gian tham số (nghĩa là mặt phẳng Ow_1w_2 trong Hình 4) chúng ta di chuyển theo hướng của $\vec{g}$ thì hàm tổn thất **tăng nhanh nhất**; nghĩa là có giá trị lớn dần sau mỗi lần dịch chuyển. Bài toán của chúng ta là tìm weights và bias để hàm tổn thất nhỏ hơn sau mỗi lần cập nhập. Do đó, chúng ta **phải đi** theo hướng ngược lại hướng của $\vec{g}$, nghĩa là hướng $-\vec{g}$. Tuy nhiên, *chúng ta nên di chuyển một khoảng cách bao xa tính từ* $\mathbf{w}^t$ *và theo hướng* $-\vec{g}$*?* Chúng ta hoàn toàn không biết!. Do đó, chúng ta sử dụng một siêu tham số, được gọi là **hệ số học (learning rate)** để tính $\mathbf{w}^t$. Hệ số học là con số cần

xác định trước quá trình huấn luyện diễn ra và là số dương đủ nhỏ, ví dụ 10^{-4}. Chúng ta tính $\mathbf{w}^{t+1}$ từ $\mathbf{w}^t$ theo Công thức (1).

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \alpha \times \overrightarrow{g} \tag{1}$$

4.2 Một số khái niệm

4.2.1 Các tập dữ liệu

Dữ liệu thu thập được, bao gồm nhãn của nó cần phải được biểu diễn ở dạng **tensor (ma trận nhiều chiều)**[16] thì quá trình huấn luyện mới bắt đầu được. Toàn bộ dữ liệu được chia tách thành ba tập dữ liệu con, đó là: (a) Tập huấn luyện (Training-set), (b) Tập kiểm thử (Validation-set) và (c) Tập kiểm tra (Testing-set). Công dụng của các tập ở trên được trình bay sau đây:

1. **Tập huấn luyện (Training-set)**: Là tập dữ liệu dùng để tính gradient-vector và cập nhật các tham số của mô hình. Nói cách khác, tập này dùng để **chọn tham số** của mô hình, nghĩa là weights và bias.
2. **Tập kiểm thử (Validation-set)**: Tập này cũng có nhiều tác giả dịch là tập kiểm tra. Tuy nhiên, ở đây "kiểm thử" được dùng với lý do tập này dùng để **thử** xem mô hình đang được học có hiệu quả như thế nào khi làm việc với dữ liệu không nằm trong tập huấn luyện. Tập này được dùng để lựa **chọn siêu tham số**.
3. **Tập tra (Testing-set)**: Tập này được dùng để đánh giá hiệu quả của mô hình và để **công bố** và **so sánh** các kỹ thuật khác nhau, các kiến trúc khác nhau, v.v.

Cách dể dàng nhất để tạo ra ba tập dữ liệu nói trên từ tập chung đó là:

1. Trộn ngẫu nhiên các mẫu dữ liệu theo phân phối đều[17]
2. Chia tách các mẫu vào ba tập theo tỉ lệ $\alpha\%$ (cho tập huấn luyện), $\beta\%$ (cho tập kiểm thử), và $(1 - (\alpha + \beta))\%$ (cho tập huấn kiểm tra). Tuỳ vào độ lớn của tập dữ liệu mà chúng ta có các phần trăm phù hợp. Có thể tham khảo: $\alpha = [70, 80]; \beta = [10, 15]$.

4.2.2 Batch, Epoch và Shuffle

Ta dùng các tập huấn luyện để cập nhật W và $\mathbf{b}$. Cách phổ biến là:

[16]Ma trận nhiều chiều: Trong Python có thư viện **Numpy**, **Pytorch**, **Tensorflow**. Trong C++: xtensor
[17]còn được gọi là **shuffle**

1. Sắp xếp các mẫu dữ liệu trong tập huấn luyện theo một trật tự nào đó (**shuffle**).
2. Chia toàn bộ dữ liệu trong tập thành các bó (batch)[18] có kích thước là **batch-size**, con số này cũng là một siêu tham số.
3. Ta dùng bó dữ liệu để tính hàm tổn thất, tính gradient-vector và cập nhật các weights và bias. Khi xong xử lý xong tất cả các bó trong tập dữ liệu, cũng là lúc mô hình đã "**nhìn**" xong từng mẫu dữ liệu một lần. Lúc này, ta nói đã kết thúc một **epoch**.

Cũng giống con người, rất ít người có khả năng hiểu hết và nhớ hết khi chỉ lướt qua cuốn sách một lần. Huấn luyện mạng nơron cũng vậy, chúng ta cần thực hiện nhiều epoch, bằng cách lặp lại ba bước ở trên. Trong huấn luyện, để việc cập nhật các tham số hiệu quả, chúng ta cần giả lập dữ liệu trong tập huấn luyện để chúng đến theo thứ tự bất kỳ. Để làm việc này, chúng ta cần sắp xếp ngẫu nhiên, theo phân phối đều, các mẫu trong tập huấn luyện ở Bước 1 ở trên. Việc này cũng chính là gán `shuffle=true` trong bộ tải dữ liệu.

Khi làm việc với các tập dữ liệu kiểm thử và kiểm tra thì chúng ta không cần xáo trộn (shuffle) dữ liệu.

4.3 Giải thuật

Minibatch Stochastic Gradient Descent (SGD) là giải thuật quan trọng và hiện đang được dùng để huấn luyện các mô hình mạng nơron và mô hình học sâu. Giải thuật này có thể được tóm lược như trong Algorithm 1.

- **Đầu vào**:
  - `trainloader` và `validloader`: là các bộ tải dữ liệu cho tập huấn luyện và kiểm thử.
  - `lossLayer`: Hàm tính toán tổn thất là bắt buộc phải có trong quá trình huấn luyện, xem Hình 3.
  - `optimizer`: Công thức cơ bản nhất để cập nhật tham số của mô hình được nêu trong Công thức (1). Tuy nhiên, một số biến thể từ đó đã có hiệu quả hơn và dễ dàng hơn cho sử dụng; `optimizer` chính là đề cập đến biến thể nào được sử dụng.
  - `metricLayer`: Trong quá trình huấn luyện, chúng ta cần ghi nhận các giá trị tổn thất và tính hiệu quả của mô hình; vấn đến này được thực hiện bởi `metricLayer`.
  - `nepoches`, α (hệ số học): quá trình huấn luyện diễn ra trong bao nhiêu epoch và dùng hệ số học lớn nhỏ ra sao cũng cần được thiết lập trước.
- **Đầu ra**: Mục tiêu cuối cùng của quá trình huấn luyện là tìm ra các giá trị W^* và $\mathbf{b}^*$ làm cho mô hình làm việc tốt. Chúng cũng chính là đầu ra của mô hình.

[18] batch: bó, lô

Data: (Input)
1. `trainloader`, `validloader`: Các `DataLoader` cho các tập huấn luyện và kiểm thử
2. `optimizer`, `lossLayer`, `metricLayer`: Bộ tối ưu, hàm tổn thất và lớp đánh giá hiệu quả
3. Các siêu tham số: nepochs, learning-rate (α)
Result: (Output) $W^*, \mathbf{b}^*$
1. Khởi động tham số của mô hình
2.. **for** *epoch in range(nepoches)* **do**
  2.1.Bật `train_mode=true`
  2.2. **for** *batch in trainloader* **do**
    2.2.1. X, t = batch.data, batch.label
    2.2.2. Lan truyền thuận (xuyên qua hết `lossLayer`)
    2.2.3. Lan truyền ngược (ngược về, từ đầu ra của `lossLayer`)
    2.2.4. Cập nhật tham số của mô hình (cần đến `optimizer`, α)
    2.2.5. Ghi nhận giá trị tổn thất và hiệu quả của mô hình (cần đến `metricLayer`)
  end
  2.3. Đánh giá mô hình với tập kiểm thử (validloader)
  2.4. In ra thông tin
end

Algorithm 1: Minibatch Stochastic Gradient Descent (SGD)

- **Các bước trong giải thuật**:
  - **1. Khởi động tham số của mô hình**: trước khi quá trình huấn luyện bắt đầu, chúng ta phải khởi động các tham số; nghĩa là gán các tham số với các giá trị nào đó. Có hai cách thường dùng, đó là:
    - (1) Khởi động từ mô hình đã được huấn luyện trước đó (pretrained). Trường hợp này, huấn luyện có ý nghĩa là tinh chỉnh thêm (finetuning) để mô hình tốt hơn so với mô hình cũ.
    - (2) Khởi động để huấn luyện từ đầu. Có nhiều cách khởi động; cách đơn giản là khởi động với các con số nhỏ ngẫu nhiên theo phân phối chuẩn tắc cho weights, và bias thì có thể gán là 0.
    - **Lưu ý**: Về hiện thực cụ thể, bước khởi có thể nằm ngoài giải thuật huấn luyện; chỉ cần đảm bảo các giá trị weights và bias phải được khởi gán.
  - **2. Lặp trên mỗi epoch**: Một epoch là một lần nhìn qua (dùng) mỗi điểm dữ liệu trên tập huấn luyện một lần. Như vậy, một epoch chứa nhiều bó (batch) dữ liệu. Các công việc trong epoch được tóm lược như sau:
    - **Bước 2.1**: Bước này chuyển tất cả các lớp tính toán về mode huấn luyện (train_mode). Lý do của việc này là: quá trình lan truyền thuận có cách làm việc khác nhau giữa train_mode và eval_mode. Trong toàn bộ giải thuật, chỉ

có Bước 2.3 là cần chuyển về eval_mode. Do đó, nếu hiện thực của Bước 2.3 bắt đầu với việc chuyển về eval_mode và kết thúc là phục hồi mode ban đầu, thì Bước 2.1 cũng không nhất thiết phải có.

* **Bước 2.2**: Xử lý các bó dữ liệu trong epoch. Trong đó, các **Bước 2.2.2**, **2.2.3**, **2.2.4** là quan trọng nên được tách giải thích riêng trong các phần theo sau. **Bước 2.2.5** là ghi nhận giá trị tổn thất và hiệu quả trong batch và tích luỹ chúng qua các batch cho từng epoch.

4.4 Lan truyền thuận, forward-pass

Như đã được giới thiệu ở các phần trước, nếu chúng ta xem mô hình học sâu và hàm tổn thất là những hàm toán, $\mathcal{F}(X, W, \mathbf{b})$ và $\mathcal{L}(W, \mathbf{b}; X, \mathbf{t})$ (Xem Hình 3), thì lan truyền thuận là việc tính ra giá trị tổn thất (ℓ) từ bó dữ liệu và nhãn $(X, \mathbf{t})$ và các giá trị hiện tại của weights và bias $(W, \mathbf{b})$.

Còn nếu chúng ta xem mô hình học sâu là một đồ thị tính toán (DAG) thì quá trình lan truyền thuận diễn ra với hai bước sau:

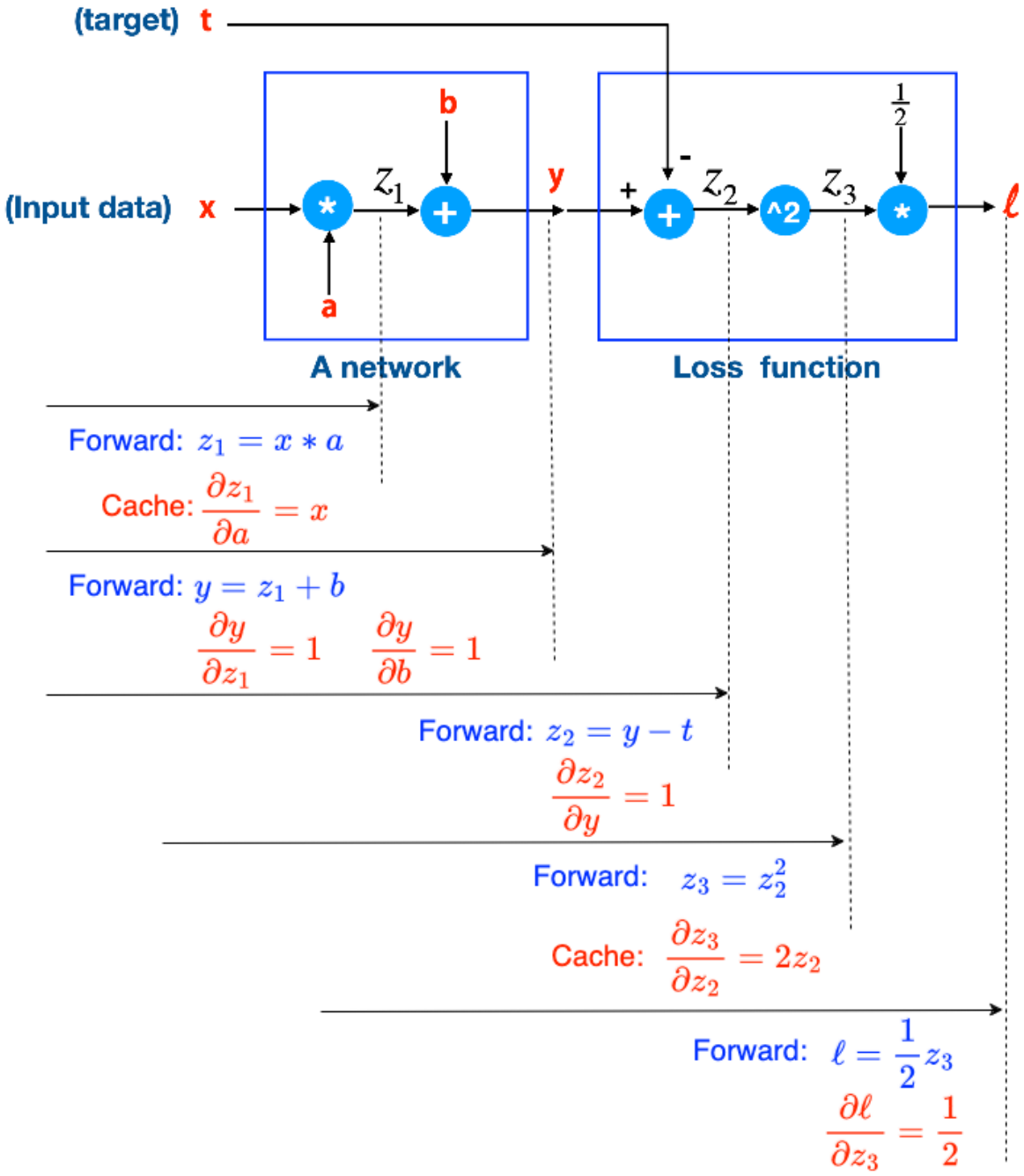
1. Tuyến tính hoá dãy các lớp tính toán trong đồ thị bằng giải thuật **TopologicalSort**. Vì, mô hình học sâu là một DAG nên luôn tồn tại dãy này.
2. Tính toán đáp ứng cho từng lớp trong dãy thu được ở bước trên.

Lan truyền thuận được hiện thực đơn giản theo cách ở trên. Tuy nhiên, có một điểm lưu ý quan trọng, đó là, cần phân biệt rõ hai chế độ `train_mode` và `eval_mode`.

1. `train_mode`: Chế độ này cho biết mô hình đang trong quá trình huấn luyện. Bước lan truyền thuận khi mô hình đang ở chế độ này thì ngoài các việc tính toán như ở trên, nó cũng cần lưu lại các dữ liệu để hỗ trợ việc tính đạo hàm (riêng) ở bước lan truyền ngược, xem thêm bước lan truyền ngược.
2. `eval_mode`: Chế độ này cho biết mô hình không ở quá trình huấn luyện, đang làm việc ở chế độ **inference**. Bước lan truyền thuận khi mô hình đang ở chế độ này thì chỉ cần tính toán đáp theo cách đã đề cập ở trên. Thông thường khi mô hình không ở chế độ huấn luyện thì bước suy diễn (inference) được dùng thay cho lan truyền thuận.

4.4.1 Ví dụ 1: Phép toán trên các số thực

Hình 5 minh hoạ cấu trúc của một mô hình đơn giản, chỉ gồm các phép toán cơ bản trên số thực. Mô hình này có khả năng dự báo ra giá trị y có quan hệ tuyến tính với x ở đầu vào, nghĩa



Hình 5: Mô hình đơn giản, tính toán chỉ trên số thực

là $y = a \times x + b$.

4.4.1.a Chế độ suy diễn (inference), eval-mode

Ở chế độ này, các tham số của mô hình đã được cho sẵn; cụ thể ở đây là a và b. Do đó, bước lan truyền thuận ở đây là theo dòng mũi tên để tính ra lần lượt các giá trị: z_1 và y; sau đó trả

về y, xem thêm ở Hình 5. Ở chế độ này, lan truyền thuận cũng không cần phải lưu lại (cache) giá trị nào cả.

4.4.1.b Chế độ huấn luyện, training-mode

Ở chế độ này, lan truyền thuận vừa theo dòng các mũi tên để tính các giá trị là đầu ra của các phép tính toán trong mô hình, vừa phải lưu lại (cache) một số giá trị để hỗ trợ tính đạo hàm riêng ở bước lan truyền ngược.

Xét một cặp dữ liệu và nhãn $< x, t >$. Dựa theo bộ giá trị này, bước lan truyền thuận thực hiện lần lượt các phép tính từ bên trái sang ở trong Hình 5; cụ thể là:

1. Lần lượt tính ra các đại lượng z_1, y, z_2, z_3 và ℓ.
2. Khi đi xuyên qua mỗi nút tính toán, lan truyền thuận cũng tính ra các đạo hàm riêng cục bộ [19]. Nếu các đạo hàm riêng này khác hằng số thì nó cũng phải lưu lại để phục vụ bước lan truyền ngược ở sau.

4.5 Lan truyền ngược, backward-pass

Mục tiêu của quá trình lan truyền ngược là tính ra gradient vector đã được đề cập ở các phần trước, nhằm hỗ trợ thao tác cập nhập dựa theo Công thức (1).

4.5.1 Lan truyền ngược đơn giản

Xét đồ thị tính toán như Hình 5. Mục tiêu của lan truyền ngược ở ví dụ này chính là tính $\overrightarrow{g} = [\frac{\partial \ell}{\partial a}, \frac{\partial \ell}{\partial b}]^T$. Sử dụng **chain-rule** ta có thể tính $\frac{\partial \ell}{\partial a}$ và $\frac{\partial \ell}{\partial b}$ theo Công thức (2) và (3). Tuy nhiên, bằng cách nào chúng ta biết nên phân rã theo dãy các biến trong các công thức trên? Thật ra chúng ta không phân rã theo cách toán học; chúng ta **lần ngược theo mũi tên** trong Hình 5 để nhân lần lượt các đạo hàm cục bộ với nhau. Các đạo hàm cục bộ này đã được tính và lưu lại sẵn khi thực hiện bước lan truyền thuận. Theo cách này, ta có thể tính $\frac{\partial \ell}{\partial a}$ và $\frac{\partial \ell}{\partial b}$ theo Công thức (4) và (5). Lưu ý, máy tính sử dụng các giá trị của các biến như z_2 và x trong các công thức (4) và (5) để tính thay cho dùng ký hiệu tên biến.

[19] **Đạo hàm riêng cục bộ**: là đạo hàm riêng của đầu ra so với đầu vào của nút tính toán

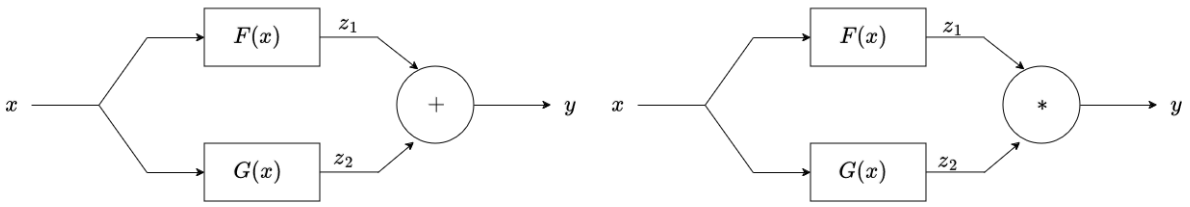
$$\frac{\partial \ell}{\partial a} = \frac{\partial \ell}{\partial z_3} \times \frac{\partial z_3}{\partial z_2} \times \frac{\partial z_2}{\partial y} \times \frac{\partial y}{\partial z_1} \times \frac{\partial z_1}{\partial a} \tag{2}$$

$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial z_3} \times \frac{\partial z_3}{\partial z_2} \times \frac{\partial z_2}{\partial y} \times \frac{\partial y}{\partial b} \tag{3}$$

$$\frac{\partial \ell}{\partial a} = \frac{1}{2} \times 2 \times z_2 \times 1 \times 1 \times x \tag{4}$$

$$\frac{\partial \ell}{\partial b} = \frac{1}{2} \times 2 \times z_2 \times 1 \times 1 \tag{5}$$

4.5.2 Lan truyền ngược qua nút tách-nhập



$y = F(x) + G(x) => y' = F'(x) + G'(x)$

(a) Aggregation: sum

$y = F(x) * G(x) => y' = F'(x)G(x) + F(x)G'(x)$

(b) Aggregation: multiply

Hình 6: Tách-nhập và lan truyền ngược

Các kiến trúc mạng nơron phức tạp thường bao gồm việc phân phối dữ liệu qua các nhánh và tính toán bởi các mô-đun riêng biệt, sau đó tổng hợp (aggregation) lại với nhau, như được minh hoạ bằng hai sơ đồ trong Hình 6. Ở trường hợp Hình 6 (a), x được phân phối vào hai hàm $F(x)$ và $G(x)$, sau đó tổng hợp bằng phép cộng. Trường hợp Hình 6 (b) giống ở Hình 6 (a) về việc phân phối, nhưng khác nhau ở phép tổng hợp. Cả hai trường hợp ta thấy giống nhau ở bước lan truyền ngược. Đó là, **khi ở bước lan truyền thuận ta phân phối một biến vào các hàm khác nhau, thì ở lan truyền ngược chúng ta cộng các đạo hàm lại với nhau**.

Một trường hợp khác, khi lan truyền thuận ta nhập nhiều nhánh bằng nút tổng hợp; ví dụ như nút $+$ và $*$ trong Hình 6, thì **khi lan truyền ngược ta cần tính đạo hàm theo từng nhánh đầu vào tuỳ cơ chế tổng hợp**. Ví dụ, ở Hình 6 (a), ta có, $y = z_1 + z_2$. Do đó. $\partial y/\partial z_1 = \partial y/\partial z_2 = 1$; trong khi đó, ở Hình 6 (b), $y = z_1 * z_2$. Do đó, $\partial y/\partial z_1 = z_2 = G(x)$ và $\partial y/\partial z_2 = z_1 = F(x)$

4.6 Chiến lược cập nhật tham số

Tham số của mô hình hay tham số có thể học gồm weights (W) và bias ($\mathbf{b}$). Công thức cập nhật của weights và bias là giống nhau. Để đơn giản cho việc trình bày ta giới thiệu biến $\mathbf{x}$; $\mathbf{x}$ có thể là W hay $\mathbf{b}$ hay bất kỳ thông số cần học nào.

Các chiến lược cập nhật thông số trong phần này đều cần ở đầu vào ít nhất là các đại lượng sau đây:

1. $\mathbf{x}$: tham số cần học;
2. $\Delta\mathbf{x}$: đạo hàm của hàm tổn thất so với $\mathbf{x}$.
3. α: hệ số học, là số thực dương đủ nhỏ, ví dụ, 10^{-4}.
4. Một số siêu tham số khác, được giới thiệu riêng cho từng chiến lược.

4.6.1 SGD

SGD là chiến lược cơ bản nhất, nó cập nhật thông số $\mathbf{x}$ theo Công thức (6).

$$\Delta\mathbf{x}^{(t+1)} = \Delta\mathbf{x}^{t} - \alpha \times \Delta\mathbf{x} \tag{6}$$

4.6.2 Momentum

Chiến lược Momentum giới thiệu thêm một biến $\mathbf{v}$ (velocity), làm nhiệm vụ tích luỹ các đạo hàm theo thời gian[20]. $\mathbf{v}$ được khởi động là 0, nó được tích luỹ qua các lần cập nhật theo Công thức (7). Ở đó, ρ là một siêu tham số, được dùng để kiểm soát việc tích luỹ đạo hàm ở biến $\mathbf{v}$. ρ có thể được gán tiêu biểu là 0.9 hay 0.99. Nếu $\rho = 0$ thì chiến lược Momentum làm việc như SGD gốc.

Dựa vào $\mathbf{v}$ đã cập nhật, Momentum tính lại $\Delta\mathbf{x}$ sử dụng Công thức (8).

$$\mathbf{v}^{(t+1)} = \rho\mathbf{v}^{t} + \Delta\mathbf{x} \tag{7}$$

$$\Delta\mathbf{x}^{(t+1)} = \Delta\mathbf{x}^{t} - \alpha \times \mathbf{v}^{(t+1)} \tag{8}$$

[20] nghĩa là, qua các lần cập nhật.

4.6.3 Adagrad

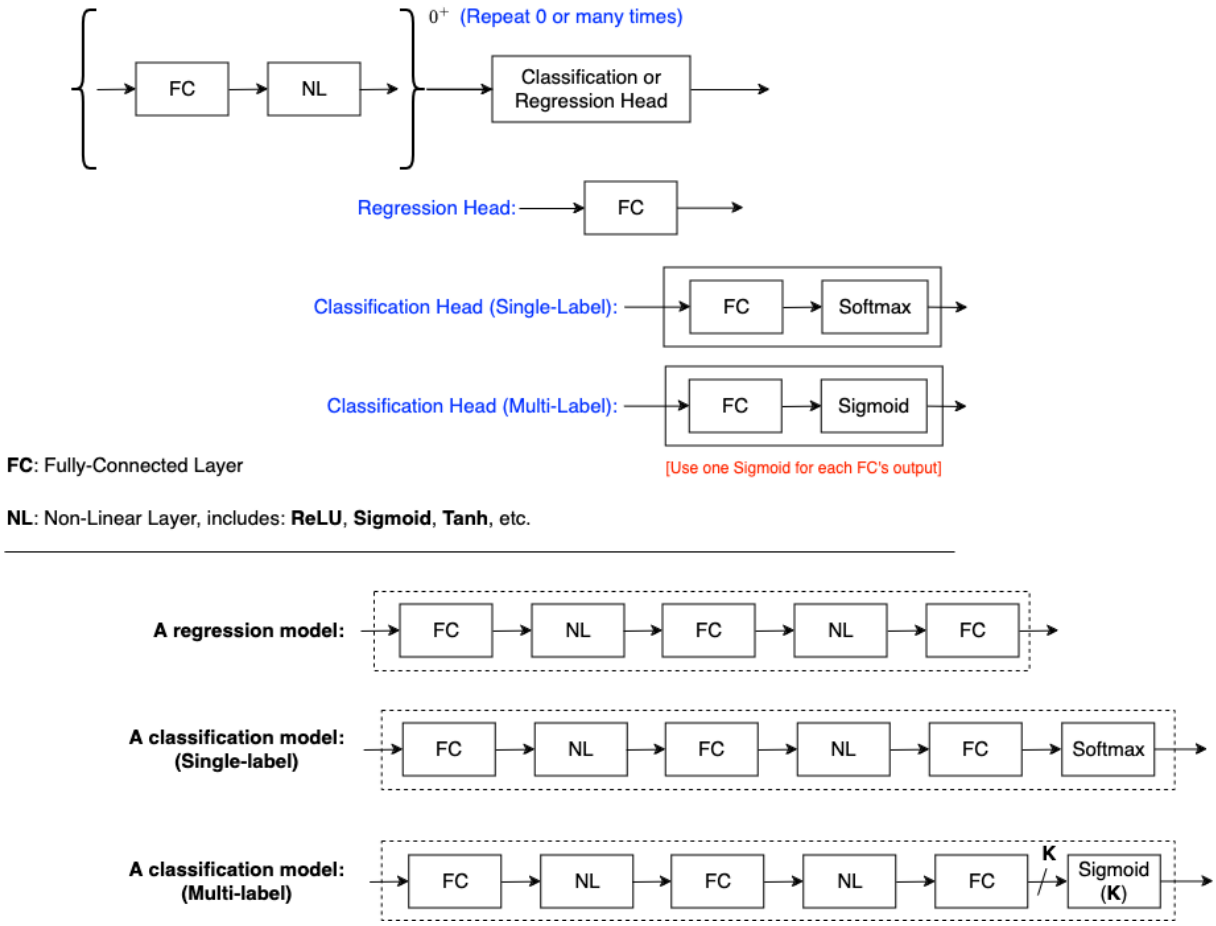
To be updated!

4.6.4 Adam

To be updated!

5 Mạng nơron truyền thẳng nhiều lớp

5.1 Kiến trúc tổng quan



Hình 7: Mạng nơron truyền thẳng nhiều lớp (MultiLayer Perceptron, MLP)

Mạng nơron truyền thẳng nhiều lớp là một cấu trúc tính toán gồm một dãy các nút (còn được là layer), được minh hoạ như ba mô hình ở cuối của Hình 7. Nói cách khác, ta có thể

dùng cấu trúc danh sách để lưu trữ dãy các nút tính toán trong MLP.

5.1.1 Tổng quan về các lớp tính toán

Các nút tính toán trong MLP chia làm hai nhóm:

1. **Tuyến tính**: Cho đến hiện nay, chỉ có hai lớp thuộc nhóm này; đó là, lớp kết nối đầy đủ (Fully-Connected, được viết tắt là FC) và lớp tích chập (Convolution, được viết tắt là Conv). Tuy nhiên, trong chủ đề về MLP thì chỉ dùng đến lớp FC. Conv thì được dùng rất phổ biến trong mô hình học sâu.
2. **Phi tuyến**: Trong Hình 7, nhóm này có tên viết tắt là NL. Hiện nay, nhóm này gồm nhiều lớp, phổ biến trong chủ đề về MLP là những lớp như ReLU, Sigmoid, và Tanh. Softmax cũng là một lớp phi tuyến; tuy nhiên, nó chỉ được dùng như lớp cuối cùng của mô hình phân loại (đơn nhãn).

5.1.2 Kiến trúc

Có thể chia dãy các nút tính toán làm hai phần, như được minh hoạ ở phần trên của Hình 7.

1. **Phần làm nhiệm vụ phân loại hay hồi quy**: Phần này trực tiếp giải quyết nhiệm vụ phân loại hay hồi quy. Chúng được đặt tên là "đầu phân loại" (classification head) hay "đầu hồi quy" (regression head). Kiến trúc của các đầu này cũng đơn giản.
    - **Đầu hồi quy**: Hồi quy là bài toán dự báo ra giá trị số thực của một đại lượng được quan tâm. Do đó, đầu này chỉ gồm một lớp tuyến tính (FC).
    - **Đầu phân loại (đơn nhãn)**: Phân loại đơn nhãn là bài toán rất phổ biến, ở đó mô hình chỉ có ý định dự báo ra một nhãn duy nhất cho từng điểm dữ liệu ở đầu vào. Do đó, cấu trúc tính toán của đầu này gồm một lớp FC để biến đổi từ đặc trưng sang điểm số và dùng một lớp Softmax để chuyển số sang một phân phối xác suất (categorial distribution). Để ra quyết định là nhãn nào ở đầu ra, bước inference cần phải thực hiện **Argmax** trên phân phối này để chọn ra nhãn có độ tin cậy (xác suất) cao nhất. Trong các tài liệu, đầu này còn được gọi với tên là **Softmax Regression**.
    - **Đầu phân loại (đa nhãn)**: Phân loại đa nhãn hay còn được gọi là "tagging", là bài toán ở đó số nhãn mà mô hình có thể dự báo cho từng điểm dữ liệu ở đầu đi từ 0 đến tổng số nhãn của bài toán. Do đó, cấu trúc tính toán của đầu này gồm một lớp FC để biến đổi từ đặc trưng sang điểm số và dùng hàm Sigmoid kết nối vào từng đầu ra của lớp FC. Đầu ra của Sigmoid là một giá trị thuộc khoảng $(0, 1)$ và có ý nghĩa

là xác suất có nhãn tương ứng là bao nhiêu. Bước inference cần phải thực hiện: so các giá trị đầu ra của Sigmoid với một giá trị ngưỡng (là siêu tham số, chọn sẵn, ví dụ là 0.5) để xem có nhãn nào đó gán được cho dữ liệu đầu vào hay không?

2. **Phần làm nhiệm vụ biến đổi đặc trưng**: Biến đổi đặc trưng là công việc đặc biệt quan trọng; nó giúp cho mô hình có năng lực lớn hơn. Cụ thể,
   - Đầu hồi quy chỉ là một biến đổi tuyến tính. Nếu mô hình chỉ gồm các lớp trong đầu này thì không thể kỳ vọng mô hình có thể tìm ra một quan hệ phức tạp (phi tuyến) giữa đầu ra và đầu vào.
   - Tương tự, đầu phân loại có đường phân tách các lớp là tuyến tính. Do vậy, nếu chỉ dùng đầu này trực tiếp và không có khối biến đổi đặc trưng thì cũng không thể kỳ vọng mô hình có đường phân tách phi tuyến được.
   - Để giải quyết hai vấn đề trên và làm cho mô hình có năng lực mạnh mẽ hơn thì phải cần đến khối biến đổi đặc trưng. Trong MLP, như được minh hoạ ở Hình 7, khối biến đổi đặc trưng được tạo thành bằng cách chồng một hay nhiều khối con, mỗi khối con là kết nối của lớp FC và một lớp trong nhóm Phi tuyến. Mạng nơron học sâu và cụ thể là kiến trúc của các phiên bản ChatGPT [21] phổ biến hiện nay, có khối biến đổi đặc trưng nổi tiếng là Transformer[22], có nhiệm như các khối biến đổi đặc trưng được đề cập ở đây.

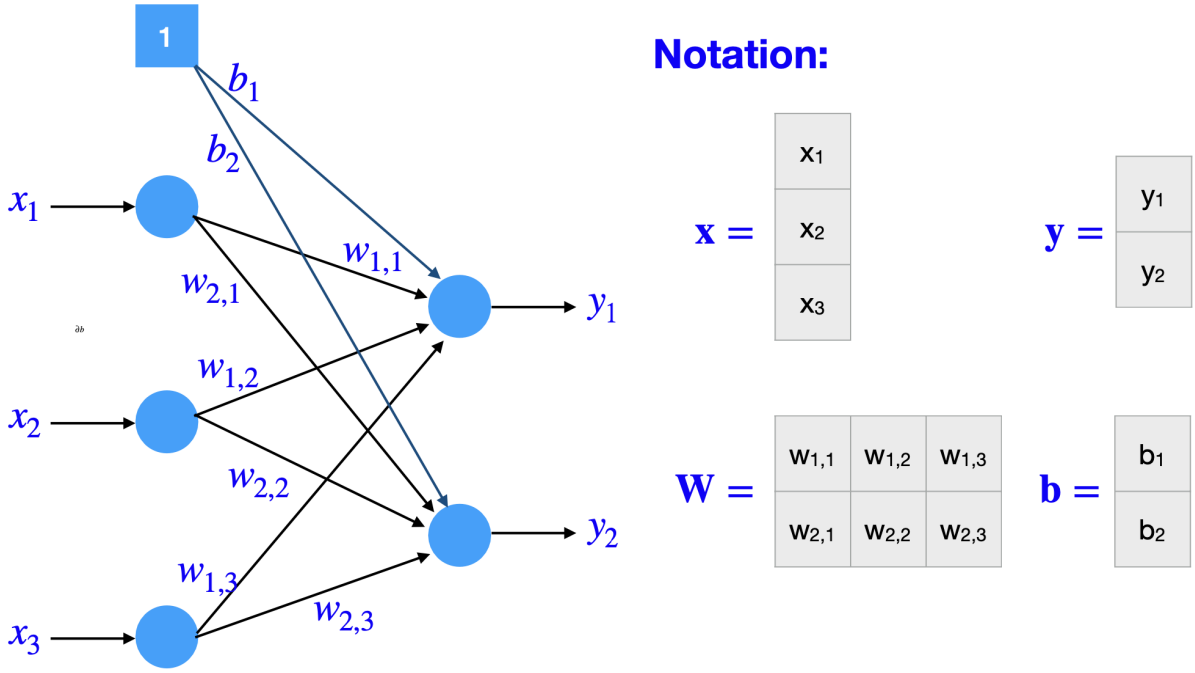
5.2 Lớp kết nối đầy đủ

Lớp kết nối đầy đủ là một biến đổi tuyến tính, như được minh hoạ trong Hình 8; ở đó, các giá trị đầu ra y_1 và y_2 được tính theo Công thức (9) và (10) tương ứng. Nếu ta đặt $\mathbf{x} = [x_1, x_2, x_3]^T$, đặt $\mathbf{w}_1 = [w_{1,1}, w_{1,2}, w_{1,3}]^T$ và đặt $\mathbf{w}_2 = [w_{2,1}, w_{2,2}, w_{2,3}]^T$, thì có thể thu được dạng ngắn gọn hơn, đó là, $y_1 = \mathbf{w}_1^T\mathbf{x} + b_1$ và $y_2 = \mathbf{w}_2^T\mathbf{x} + b_2$. Với lớp FC, $\mathbf{w}_1$, $\mathbf{w}_2$, b_1 và b_2 được gọi chung là thông số của lớp hay thông số có thể học (learnable parameters); trong đó, $\mathbf{w}_1$ và $\mathbf{w}_2$ được là "weights", còn b_1 và b_2 được gọi là "bias".

Một số lưu ý:

1. Số lượng "weights" bằng số cạnh nối từ đầu vào đến đầu ra. Vì FC là kết nối đầy đủ nên số cạnh bằng tích $N_{in} \times N_{out}$; ở đó, N_{in} và N_{out} tương ứng là số đầu vào và số đầu ra của lớp FC. Trong trường hợp FC ở trong Hình 8, số weights là $3 \times 2 = 6$.
2. Số lượng "bias" bằng số đầu ra của lớp, nghĩa là N_{out}. Trong trường hợp FC ở trong Hình

[21] **ChatGPT**
[22] **Transformer**: Attention Is All You Need



Hình 8: Minh hoạ lớp kết nối đầy đủ (Fully-Connected) có ba đầu vào hai đầu ra

8, số lượng bias là 2. Trong một số trường hợp, người dùng có thể không dùng bias, nghĩa là không có b_1 và b_2 như trong Hình 8.

3. Tổng số lượng thông số có thể học của mô hình là $(N_{in} \times N_{out})$ nếu không dùng bias và $(N_{in} \times N_{out} + N_{out})$ khi có dùng bias.

$$y_1 = w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + b_1 \tag{9}$$

$$y_2 = w_{2,1}x_1 + w_{2,2}x_2 + w_{2,3}x_3 + b_2 \tag{10}$$

5.2.1 Ma trận weights và vector bias

Các weights và bias của FC được tổ chức tương ứng ở dạng ma trận và vector. Ví dụ, như ma trận W và vector $\mathbf{b}$ được minh hoạ trong Hình 8; ở đó, W là ma trận có kích thước 2×3 (là $N_{out} \times N_{in}$ trong trường hợp tổng quát[23]); còn $\mathbf{b}$ là vector (cột) chứa hai bias là b_1 và b_2 (là N_{out} bias trong trường hợp tổng quát và có dùng đến bias).

[23]Việc tổ chức weights thành $N_{out} \times N_{in}$ hay $N_{in} \times N_{out}$ rất tuỳ vào thư viện được sử dụng.

5.2.2 Lan truyền thuận

5.2.2.a Cho một mẫu dữ liệu đơn x

Ta cũng biểu diễn dữ liệu đầu vào cho một mẫu dữ thành vector $\mathbf{x}$ như trong Hình 8. Lúc đó, đầu ra $\mathbf{y}$ thu được bằng Công thức (11); vector $\mathbf{y}$ chứa hai giá trị y_1 và y_2 như được tính bởi Công thức (9 và 10). Công thức (11) cần đến phép nhân ma trận W với vector $\mathbf{x}$ và cộng kết quả trung gian (cũng là vector) với vector $\mathbf{b}$.

$$\mathbf{y} = W\mathbf{x} + \mathbf{b} \tag{11}$$

5.2.2.b Cho một bó (batch) dữ liệu X

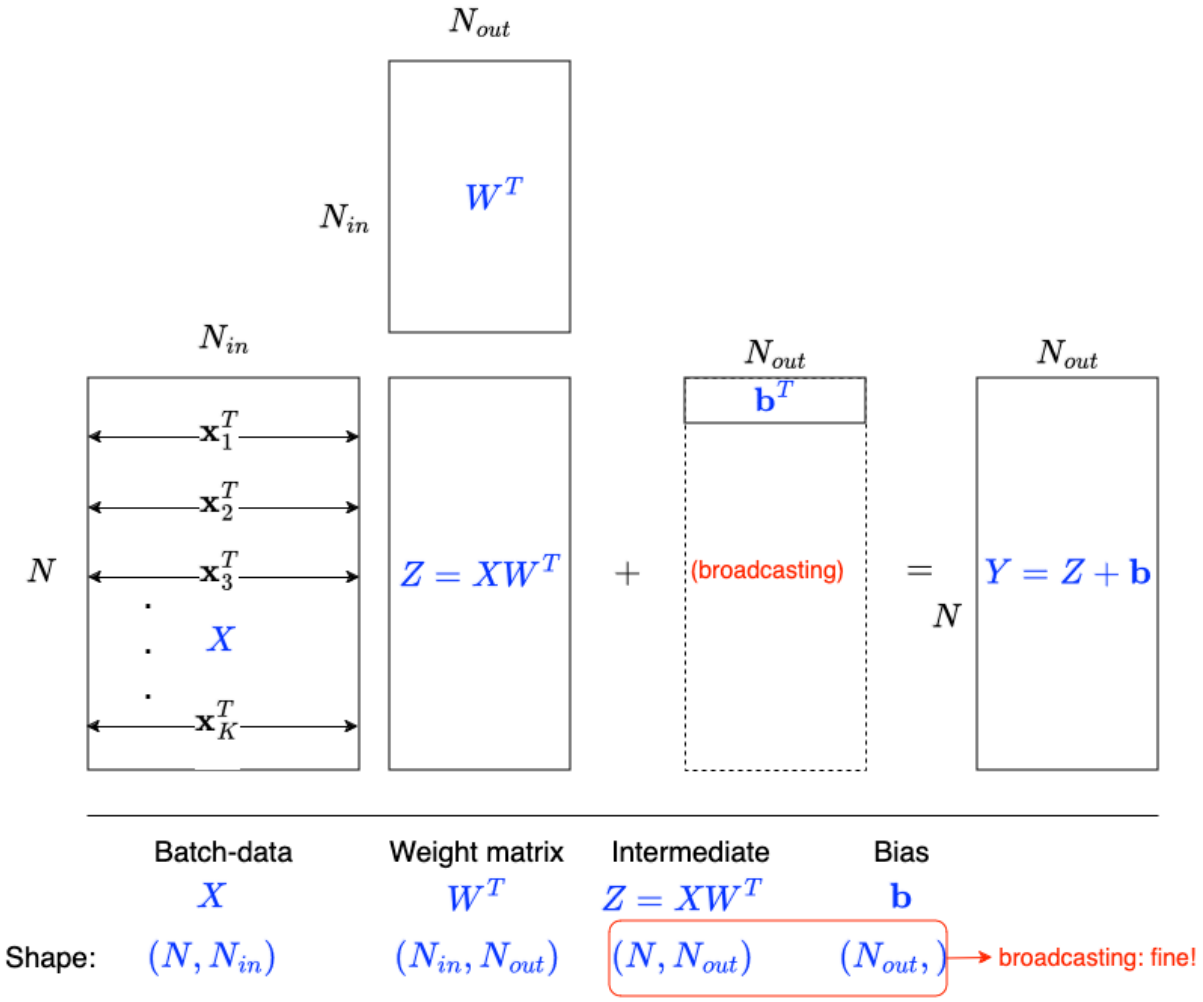
Trong mạng nơron và mô hình học sâu chúng ta thường cho cả bó (batch) dữ liệu chạy xuyên qua mạng thay vì chỉ một mẫu dữ liệu. Theo minh hoạ trong Hình 9, chúng ta có thể biến đổi toàn bộ bó dữ liệu X sang đầu ra Y bằng Công thức (12). Công thức đó cần đến phép nhân hai ma trận X và W^T và sau đó cộng với vector $\mathbf{b}$. Gọi $Z = XW^T$ là kết quả trung gian, Z là ma trận có hình dạng (N, N_{out}). Do vậy, chúng ta cần cộng ma trận Z với vector $\mathbf{b}$ (có hình dạng $(N_{out},)$). Phép cộng này cần đến một khái niệm được gọi là "broadcast". May mắn, các thư viện về ma trận nhiều chiều đều hỗ trợ, kể cả thư viện **xtensor** được dùng trong bài tập lớn.

$$Y = XW^T + \mathbf{b} \tag{12}$$

5.2.3 Lan truyền ngược

5.2.3.a Ký hiệu

- Trong tài liệu này, hàm tổn thất được ký hiệu là $\ell = \mathcal{L}(W, \mathbf{b}; X, \mathbf{t})$.
- Nhằm tránh viết ký hiệu đạo hàm riêng dài dòng, chúng ta sử dụng ký hiệu δ và Δ theo ý nghĩa được quy định như sau:
  - Nếu z là biến vô hướng (số thực) thì đạo hàm riêng của hàm tổn thất so với z được ký hiệu là δz; nghĩa là, $\delta z \stackrel{\text{def}}{=} \partial\ell/\partial z$.
  - Đạo hàm riêng của hàm tổn thất so với các biến trong vector $\mathbf{z}$ hay so với các biến trong ma trận và mảng nhiều chiều Z ta dùng ký hiệu $\Delta\mathbf{z}$ và ΔZ tương ứng.

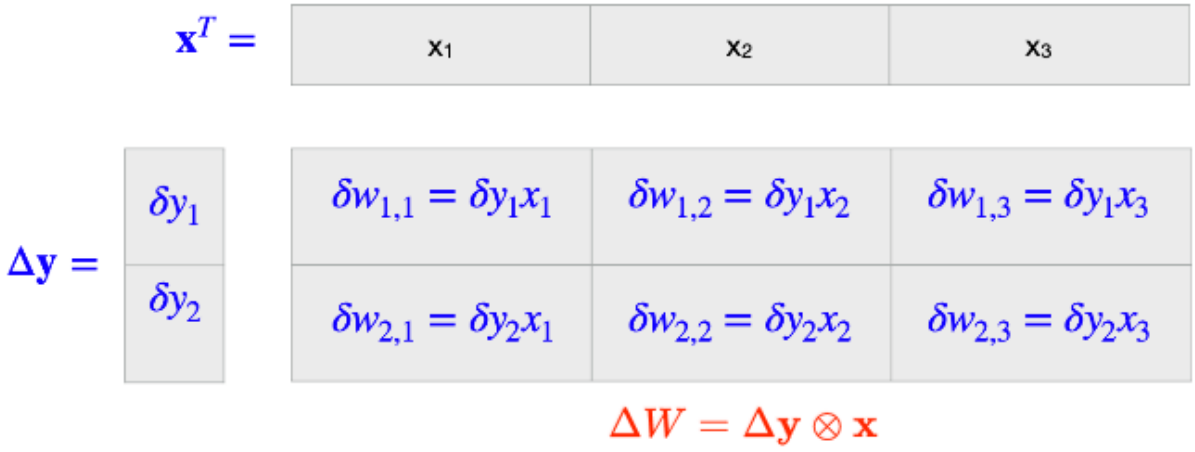


Hình 9: FC với bó dữ liệu

5.2.3.b Cách tính ΔW, cho một mẫu dữ liệu

Với một mẫu dữ liệu $\mathbf{x}$, đáp ứng $\mathbf{y}$ được tính bởi Công thức (11). Giả sử rằng, quá trình lan truyền ngược đã đến $\mathbf{y}$; nghĩa là ta đã có $\Delta\mathbf{y}$. Lúc đó, ΔW có thể được tính bởi Công thức (13); ở đó, $\otimes$ là phép tích ngoài (outer-product) giữa hai vector. Hình 10 minh hoạ cách tính này cho trường của lớp FC trong Hình 8. Một số lưu ý:

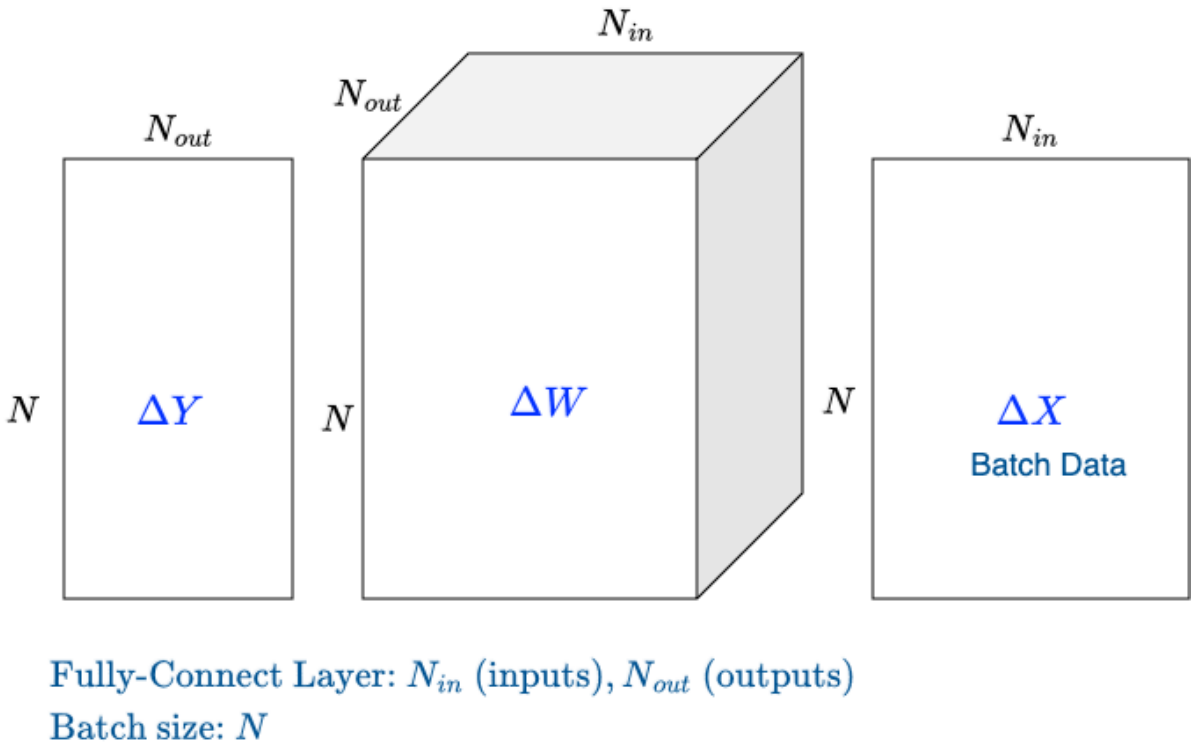
1. Công thức (13) cũng cho thấy rằng để tính toán ΔW, mẫu dữ liệu $\mathbf{x}$ cần được lưu lại (cached) trong quá trình thực hiện lan truyền thuận.
2. Hình dạng của ΔW giống với hình dạng của W, là $(N_{out} \times N_{in})$. Tương tự, với bất kỳ mảng 1 hay nhiều chiều $\mathbf{Z}$ thì hình dạng của nó khớp với hình dạng của $\Delta\mathbf{Z}$.
3. Sinh viên có thể tham khảo thêm cách dẫn ra Công thức (13) ở phần cuối của tài liệu. Tuy nhiên, để hiện thực các tính năng của lớp FC thì Công thức (13) là đủ.



Hình 10: Cách tính ΔW cho một mẫu dữ liệu

$$\Delta W = \Delta\mathbf{y} \otimes \mathbf{x}^T \tag{13}$$

5.2.3.c Cách tính ΔW, cho bó dữ liệu



Hình 11: Cách tính ΔW cho một bó dữ liệu

Trong phần trên chúng ta thấy rằng ΔW khi tính cho một mẫu dữ liệu đã là một ma trận.

Do vậy, để lưu trữ ΔW cho cả bó dữ liệu ta cần một khối, hay mảng 3 chiều. Chiều đầu tiên có kích thước là N bằng với kích thước của bó dữ liệu. Hai chiều còn lại có kích thước bằng với kích thước ma trận W. Nghĩa là, ΔW có hình dạng: (N, N_{out}, N_{in}).

Tuy vậy, sau khi tính ra khối nói trên, chúng ta lấy trung bình theo trục đầu tiên và chỉ lưu trữ ma trận ΔW trung bình. Chúng ta cũng nên lưu tổng số mẫu dữ liệu trong bó, N, để khi cần thiết có thể dùng nó nhằm tính ma trận ΔW trung bình qua nhiều bó dữ liệu.

5.2.3.d Cách tính Δb

$\Delta\mathbf{b}$ được tính toán đơn giản nhiều so với tính ΔW. Khi cho một vector đặc trưng chạy qua lớp, $\Delta\mathbf{b}$ được tính theo Công thức (14).

Khi cho bó dữ liệu gồm N vector đặc trưng chạy qua lớp thì $\Delta\mathbf{b}$ được theo Công thức (15). Tuy nhiên, sau đó, chúng ta lấy trung bình trên N vector trong $\Delta\mathbf{b}$, chỉ để lưu lại vector $\Delta\mathbf{b}$ trung bình; chúng ta cũng cần phải lưu lại số lượng mẫu trong bó để hỗ trợ tích luỹ đạo hàm xuyên qua nhiều bó dữ liệu.

$$\Delta\mathbf{b} = \Delta\mathbf{y} \tag{14}$$

$$\Delta\mathbf{b} = \Delta Y \tag{15}$$

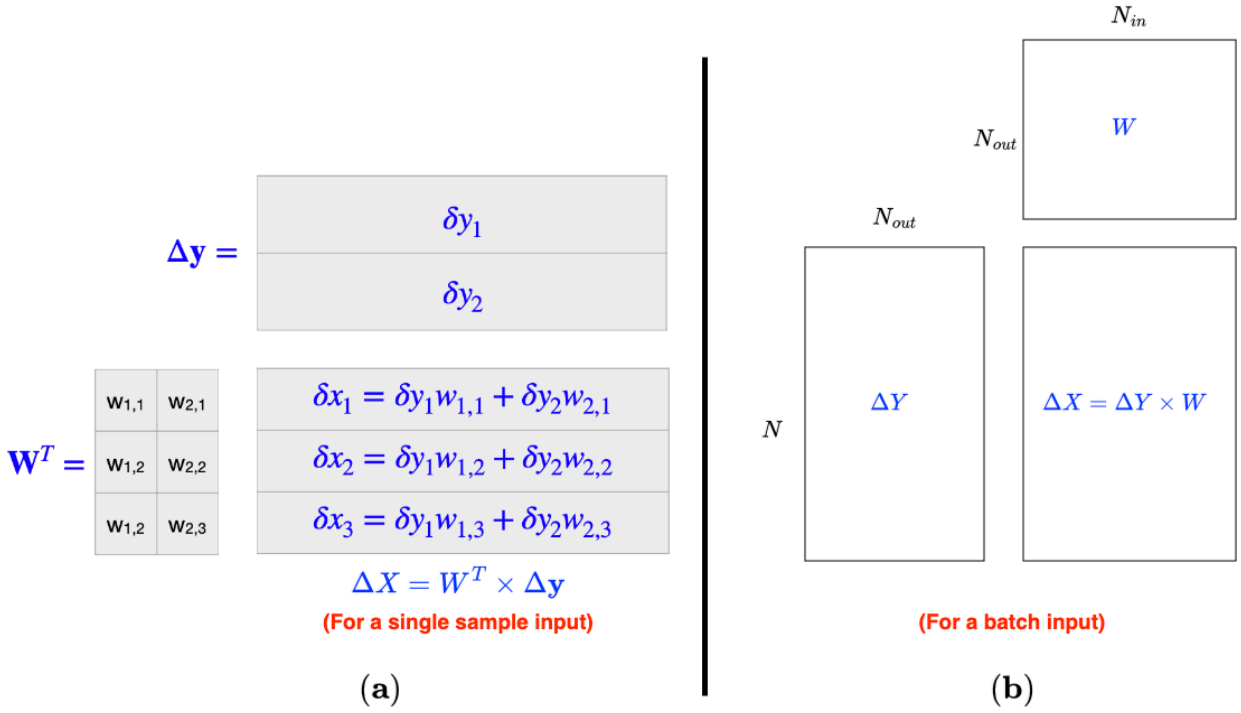
5.2.3.e Cách tính ΔX

Khi cho một vector đặc trưng qua lớp FC; ở bước lan truyền ngược, cách tính ΔX được minh hoạ trong Hình 12 (a). Cụ thể, ta lấy ma trận W^T nhân với vector $\Delta\mathbf{y}$ (đã được cung cấp sẵn khi lan truyền ngược đến lớp FC).

Trường hợp ta cho một bó dữ liệu chạy qua lớp FC thì tại bước lan truyền ngược ΔY là một ma trận cho kích thước $N \times N_{out}$. Công thức (16) được dùng để tính ΔX từ ΔY; cụ thể là nhân ma trận ΔY với ma trận W. Kết quả thu được, ΔX, có kích thước $N \times N_{in}$.

Một số lưu ý:

- Với ΔW và $\Delta\mathbf{b}$, chúng ta cần tính và lưu lại ΔW và $\Delta\mathbf{b}$ trung bình cho N mẫu trong bó. Tuy nhiên, với ΔX chúng ta không được lấy trung bình theo số mẫu, cần để nguyên vậy và lan truyền ngược tiếp tục.
- Công thức (16) cho thấy ΔX chỉ phụ thuộc vào ΔY nhận được ở bước lan truyền ngược và ma trận weights W (luôn sẵn sàng có trong lớp FC). Do đó, chúng ta không cần lưu



Hình 12: Cách tính ΔX

lại gì thêm để hỗ trợ tính ΔX.

$$\Delta X = \Delta Y \times W \tag{16}$$

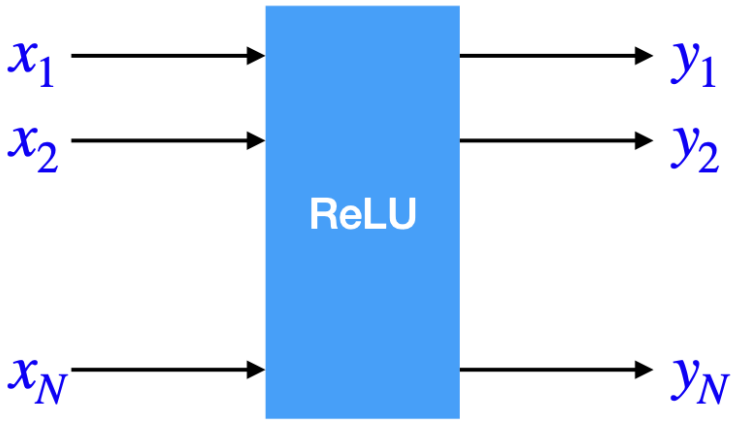
5.3 Lớp ReLU

ReLU là một hàm thuộc nhóm phi tuyến và không có thông số nào cần phải học. Khi hàm này nhận vào một vector đặc trưng hay một bó dữ liệu có hình dạng bất kỳ, nó thực hiện việc biến đổi như sau và được Minh hoạ trong Hình 13:

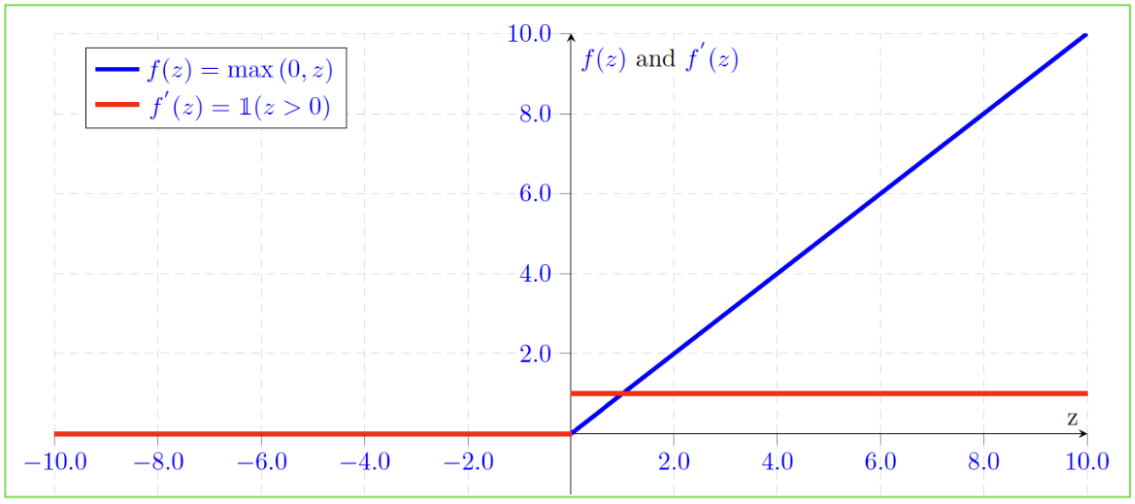
- Cho từng con số trong bó, nếu số đó lớn không hay bằng không thì được giữ nguyên ở đầu ra; ngược lại, thì đầu ra tương ứng được gán là 0.
- Lưu ý: ReLU giữ nguyên hình dạng dữ liệu ở đầu vào cho đầu ra.

RELU khá đơn giản về mặt hiện thực, được trình bày như sau:

1. **Lan truyền thuận**: Gồm hai bước chính,
   (a) Tạo ra mặt nạ M theo Công thức (17). Mặt nạ này chỉ gồm các giá trị `true (1)` và `false (0)` tương ứng các giá trị lớn hay bằng không và các trị âm ở đầu X. Mặt



$$y_i = x_i, \textbf{ if } x_i \geq 0$$
$$= 0, \textbf{ otherwise}$$



Hình 13: Nguyên lý hoạt động của hàm ReLU

nạ này có hình dạng giống như hình dạng của X. Nếu mạng nơron đang trong giai đoạn huấn luyện thì nó cũng cần được lưu lại (cached) để hỗ trợ tính DY.

(b) Đầu ra Y của ReLU được tính bằng Công thức (18); ở đó, $\odot$ là **phép nhân phần tử với phần tử tương ứng**. Phép này cũng chính là phép toán $*$ trong nhiều thư viện về mảng nhiều chiều.

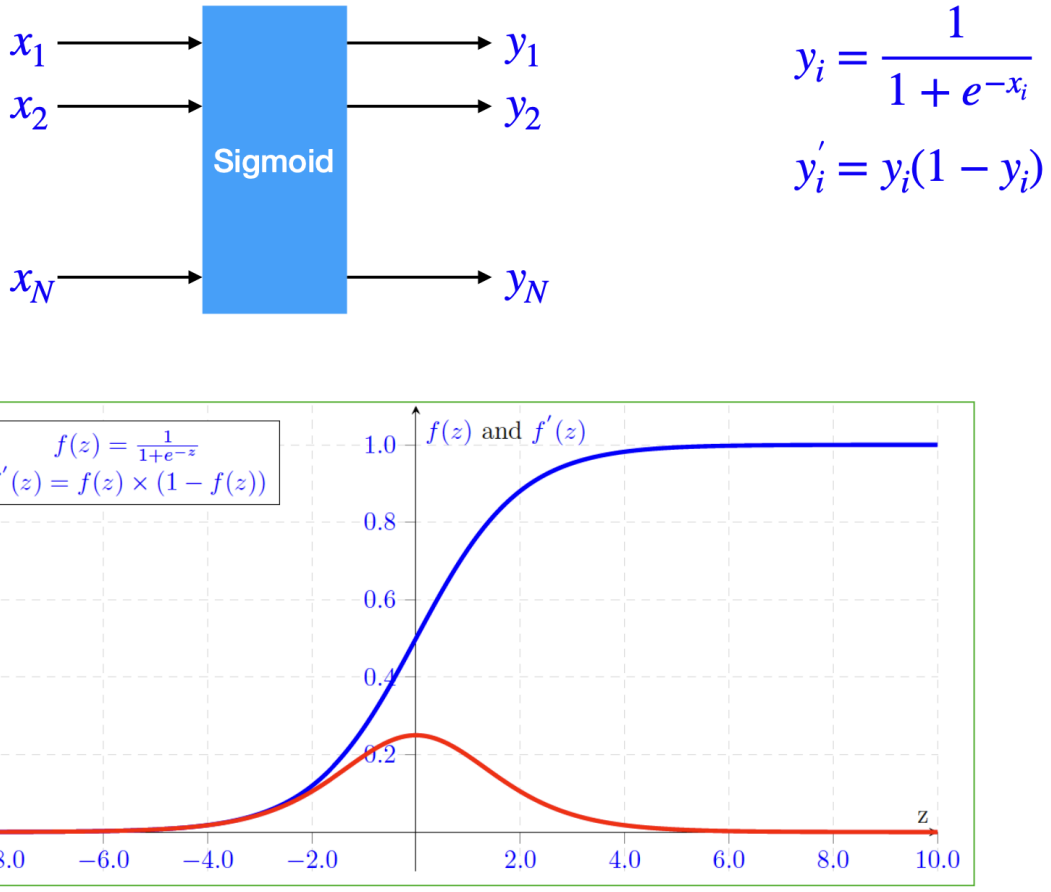
2. **Lan truyền ngược**: Với mặt nạ M được lưu lại, DX được tính bởi Công thức (19). Ý nghĩa của phép nhân này là, chỉ cho đạo hàm chạy xuyên qua các noron được kích hoạt (có giá trị lớn hơn hay bằng không, lúc tính lan truyền thuận).

$$M = X >= 0 \quad \text{(M: một mặt nạ, cached)} \tag{17}$$

$$Y = M \odot X \quad \text{(forward)} \tag{18}$$

$$DX = M \odot DY \quad \text{(backward)} \tag{19}$$

5.4 Lớp Sigmoid



Hình 14: Nguyên lý hoạt động của hàm Sigmoid

Sigmoid là một hàm thuộc nhóm phi tuyến và không có thông số nào cần phải học. Cũng giống như nhiều hàm khác trong nhóm này, nó xử lý các con số ở đầu vào độc lập nhau và giữ nguyên hình dạng của X ở đầu vào sang cho Y ở đầu ra.

Khi cho một bó dữ liệu X ở đầu vào, các bước lan truyền thuận và ngược được tính bởi Công thức (20) và (21) tương.

$$Y = \frac{1}{1 + \exp(-X)} \quad \text{(forward)} \tag{20}$$

$$DX = DY \odot Y \odot (1 - Y) \quad \text{(backward)} \tag{21}$$

Như trình bày trong Hình 14, đạo hàm của hàm Sigmoid tương đối nhỏ, giá trị lớn nhất cũng chưa đến 0.3 (tại $z = 0$); và gần như là 0 khi đầu vào nằm ngoài khoảng $(-6.0, +6.0)$. Trong khi đó, giá trị đạo hàm cục này phải nhân với các đạo hàm cục khác (ở bước lan truyền ngược) để tính đạo hàm cuối cùng. Giá trị nhỏ cũng như vậy cũng là nguyên nhân gây ra sự biến mất của đạo hàm và gây khó khăn cho việc huấn luyện mạng nơron.

Hàm Sigmoid biến đổi giá trị trong phạm vi bất kỳ ở đầu vào khoảng $(0, 1)$ ở đầu ra nên ngoài khả năng dùng như một lớp phi tuyến ở trung gian nó cũng được dùng làm đầu ra của mô hình phân loại đa nhãn. Cụ thể, là mô hình dự báo xác suất có (hay không có) nhãn nào đó cho mẫu dữ liệu đầu vào.

5.5 Lớp Tanh

Hình 15 và 14 cho thấy đồ thị của hàm Tanh khá giống với hàm Sigmoid, chỉ có đều Tanh sẽ ánh xạ dữ liệu đầu đến khoảng $(-1, 1)$ thay cho $(0, 1)$ của hàm Sigmoid. Về mặt tính toán, khi có bó dữ liệu X ở đầu vào thì lan truyền thuận và ngược được tính như các công thức (22) và (23).

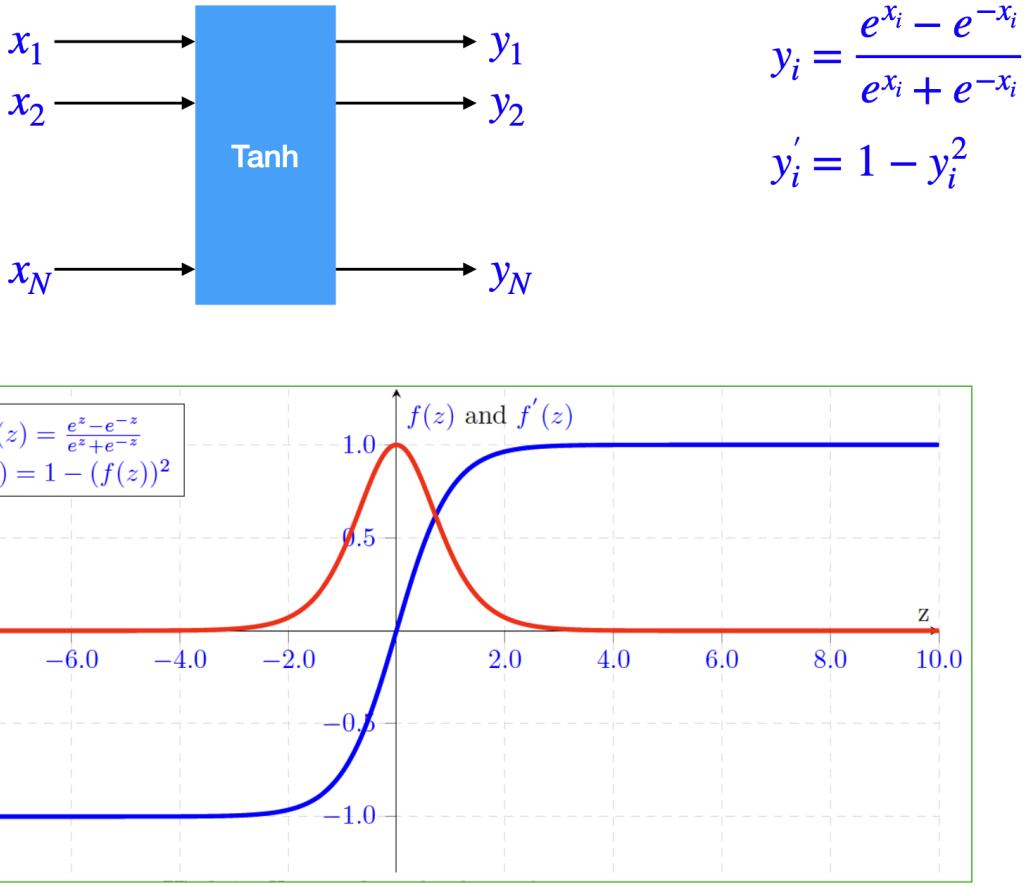
$$Y = \frac{\exp(X) - \exp(-X)}{\exp(X) + \exp(-X)} \quad \text{(forward)} \tag{22}$$

$$DX = DY \odot (1 - Y \odot Y) \quad \text{(backward)} \tag{23}$$

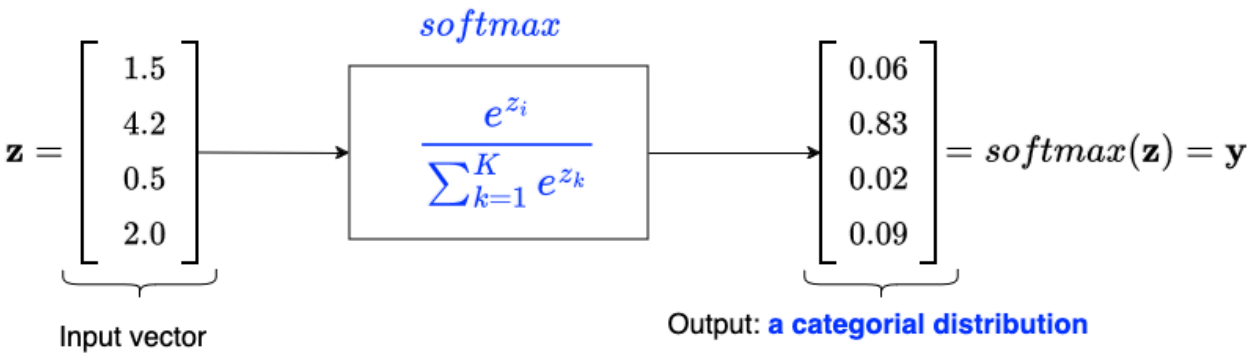
5.6 Lớp Softmax

Softmax là một hàm toán, được định nghĩa trong Công thức (24). Ý nghĩa và cách tính toán cho một vector đầu vào được minh hoạ trong Hình 16. Theo đó, tính chất quan trọng của hàm Softmax là biến một vector ở đầu vào (có giá trị bất kỳ) về một phối xác suất ở đầu ra. Do đó, các con số trong vector trả về thoả mãn hai ràng buộc sau:

1. Từng giá trị thuộc khoảng $(0, 1)$.



Hình 15: Nguyên lý hoạt động của hàm Tanh



Hình 16: Hàm softmax

2. Tổng các giá trị là 1.

Phân phối mà Softmax trả về có tên là **categorical distribution**, với ý nghĩa rằng mỗi con số trong đó là xác suất của một lớp nào đó trong bài toán phân loại đơn nhãn. Giả sử, đó là bài toán phân loại của bốn con vật sau đây (theo thứ tự): `Cat`, `Chicken`, `Dog` và `Horse`; thì phân phối xác suất ở đầu ra trong trong Hình 16 có ý nghĩa là: mô hình tin rằng có 6% khả

năng mẫu dữ liệu có nhãn là `Cat`, 83% khả năng là `Chicken`, 2% khả năng là `Dog` và 9% khả năng là `Horse`.

$$\mathbf{y} = softmax(\mathbf{z}) \quad | \quad \mathbf{z} = [z_1, z_2, \cdots, z_K]^T, y_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}, \mathbf{y} = [y_1, y_2, \cdots, y_K]^T \tag{24}$$

$$\Delta\mathbf{z} = (\text{DIAG}(\mathbf{y}) - \mathbf{y} \otimes \mathbf{y}^T) \times \Delta\mathbf{y} \tag{25}$$

Ở giai đoạn lan truyền ngược, $\Delta\mathbf{z}$ ($\mathbf{z}$: vector đầu vào của softmax) được tính theo Công thức (25); ở đó, DIAG($\mathbf{y}$) là hàm tạo ra một ma trận vuông có các giá trị là 0, ngoại trừ đường chéo chính của nó là vector $\mathbf{y}$; $\otimes$ là phép tích ngoài, và $\times$ là phép nhân của ma trận (DIAG($\mathbf{y}$)$-\mathbf{y}\otimes\mathbf{y}^T$) và vector $\mathbf{y}$. Trường hợp đầu vào là một bó dữ liệu, ví dụ là ma trận; chúng ta tính Công thức (24) và (25) cho từng hàng độc lập nhau.

6 Hàm tổn thất

6.1 Hàm Cross-Entropy

Cross-Entropy là hàm toán được sử dụng trong việc huấn luyện các mạng nơron và học sâu làm nhiệm vụ phân loại (đơn nhãn). Như được định nghĩa trong Công thức (26), hàm này nhận vào hai ma trận là Y và T; ở đó, Y là dự báo của mô hình mạng cho một bó dữ liệu ở đầu vào, và T là nhãn cho bó dữ liệu đó. Cụ thể hơn,

- Y:
  - Y là ma trận có N và K cột; với N là số lượng mẫu dữ liệu có trong bó và K là số lớp trong bài toán phân loại. Chúng ta cũng thường biểu diễn ma trận Y là dãy của các vector $\mathbf{y}_i$; mỗi $\mathbf{y}_i^T$ là một hàng trong ma trận Y; ở đây, $i = 1, 2, \cdots, N$.
  - Hàm Cross-Entropy giả định rằng mô hình có lớp cuối cùng trong đó là Softmax để đầu ra là phân phối xác suất trên các lớp của bài toán. Do đó, các hàng của ma trận Y là những con số trong khoảng $(0, 1)$ và có tổng bằng 1.
- T:
  - T cũng là ma trận có kích thước giống với Y, là $N \times K$. Mỗi hàng là được gọi là vector $\mathbf{t}_i^T$, $i = 1, 2, \cdots, N$; nó cũng là một phân phối xác suất.
  - $\mathbf{t}_i$ là mã hoá cho nhãn ở dạng xác suất. Ví dụ, giả sử rằng $K = 4$, và danh sách nhãn là [“Cat“, “Chicken“, “Dog“, “Horse“] (có thứ tự) thì mã hoá nhãn có thể được hiểu

như sau:

* Nếu mẫu dữ liệu thứ i có nhãn duy nhất là "Dog" thì $\mathbf{t}_i = [0, 0, 1, 0]^T$; ý nghĩa là, 100% người làm nhãn tin rằng nhãn phải là "Dog". Trường hợp này, chúng ta gọi là "hard-label". Nếu tất cả các mẫu trong tập dữ liệu là "hard-label" thì Công thức (27) cũng có thể được dùng để tính cross-entropy.
* Trường hợp người dùng không chắc chắn về việc gán nhãn cho mẫu dữ liệu thứ i; cụ thể, 80% họ tin là "Cat" và 20% khả năng là "Dog" thì $\mathbf{t}_i = [0.8, 0, 0.2, 0]^T$. Trường hợp này chúng ta gọi là "soft-label", và phải dùng Công thức (26) để tính cross-entropy.

Công thức (26) và (27) đều trả về giá trị thực, là tổn thất tính được cho bó dữ liệu có N mẫu; trong đó, N_{norm} có thể là 1 nếu chúng ta muốn tổng tổn thất của các mẫu (còn được gọi là "reduce=sum"). Nếu chúng ta muốn lấy trung bình ("reduce=mean") thì $N_{norm} = N$. Ở Công thức (26), chúng ta cần phép toán $\cdot$, là phép tích vô hướng của hai vector. Trong Công thức (27), thành phần y_{i,t_i} là phần tử tại hàng i và cột t_i trong ma trận Y; ở đó, t_i là nhãn của mẫu dữ liệu thứ i. Lưu ý, Công thức (27) dùng cho trường hợp "hard-label", nên vector nhãn $\mathbf{t}$ chỉ lưu **chỉ số** của nhãn trong danh sách nhãn (có thứ tự); nghĩa là, t_i là con số nguyên trong $\{0, 1, \cdots, (K-1)\}$.

Bước lan truyền ngược tính $\Delta\mathbf{y}$ cho từng mẫu dữ liệu như trong Công thức (28). Ở đó, phép chia giữa hai vector là phép chia từng cặp phần tử tương ứng và EPSILON là một giá trị nhỏ nhằm tránh chia cho 0, ví dụ EPSILON $= 10^{-7}$.

$$\text{CE}(Y, T) = -\frac{1}{N_{norm}} \sum_{i=1}^{N} \mathbf{t}_i \cdot \log(\mathbf{y}_i) \quad | \quad Y = [\mathbf{y}_1^T, \mathbf{y}_2^T, \cdots, \mathbf{y}_N^T], T = [\mathbf{t}_1^T, \mathbf{t}_2^T, \cdots, \mathbf{t}_N^T], \tag{26}$$

$$\text{CE}(Y, \mathbf{t}) = -\frac{1}{N_{norm}} \sum_{i=1}^{N} \log(y_{i,t_i}) \quad | \quad Y = [\mathbf{y}_1^T, \mathbf{y}_2^T, \cdots, \mathbf{y}_N^T], \mathbf{t} = [t_1, \cdots, t_i, \cdots t_N], \tag{27}$$

$$\Delta y = -\frac{1}{N_{norm}} \times \frac{\mathbf{t}}{\mathbf{y} + \text{EPSILON}} \tag{28}$$

6.2 Binary Cross-Entropy (BCE)

Binary Cross-Entropy được dùng trong việc huấn luyện các mô hình mạng nơron và học làm nhiệm vụ phân hai lớp (two-class classification) hay phân loại đa nhãn. Hàm này nhận vào hai vector $\mathbf{y}$ và $\mathbf{t}$, là dự báo và nhãn. Vector $\mathbf{y}$ chứa các giá trị xác suất thuộc $(0, 1)$; nghĩa là, BCE

giả thiết rằng mô hình đã dùng hàm Sigmoid để chuyển về xác suất ở đầu ra. Các giá trị trong **t** chứa mã hoá dạng xác suất cho nhãn. Nếu mà "hard-label" thì các giá trị trong **t** hoặc là số 0 (xác suất = 0) hoặc là số 1 (xác suất = 1).

Binary Cross-Entropy được tính theo Công thức (29); ở đó, N_{norm} bằng 1 nếu tính tổng tổn thất và bằng N nếu tính trung bình của tổn thất.

Ở bước lan truyền ngược, BCE được tính theo Công thức (30).

$$\text{BCE}(\mathbf{y}, \mathbf{t}) = -\frac{1}{N_{norm}} \times \sum_{i=1}^{N} \left[t_i \times \log y_i + (1 - t_i) \times \log (1 - y_i) \right] \tag{29}$$

$$\Delta \mathbf{y} = [\delta y_1, \cdots, \delta y_i, \cdots, \delta y_N]^T \quad | \quad \delta y_i = -\frac{1}{N_{norm}} \times \left[\frac{t_i}{y_i} - \frac{1 - t_i}{1 - y_i} \right] \tag{30}$$

6.3 Mean Squared Error

To be updated!

7 Hướng dẫn hiện thực

7.1 Các lớp tính toán

7.1.1 Lớp FCLayer

Lớp `FCLayer` được định nghĩa trong Hình 17. Các biến thành viên được giải thích như sau:

- `m_nNin` và `m_nNout`: hai biến này tương ứng lưu giữa số lượng ngõ vào vào và ngõ ra của lớp kết nối đầy đủ. Chúng chính là N_{in} và N_{out} trong phần 5.2.1. Hai biến này phải được khởi động trong hàm khởi tạo bằng các thông số có tên tương ứng.
- `m_bUse_Bias`: biến có kiểu `bool` này cho biết lớp kết nối đầy đủ có dùng bias hay không; nó phải được khởi gán trong hàm khởi tạo, từ biến có tên tương ứng.
- `m_aWeights` và `m_aBias`: hai biến này làm weights và bias của lớp kết nối đầy đủ. Chúng phải được tạo ra và khởi gán trong hàm khởi tạo.
  - `m_aWeights`: Phải được tạo ra để có hình dạng (N_{out}, N_{in}), và phải khởi gán là các số sinh ngẫu nhiên theo phân phối chuẩn tắc[24].

[24] dùng hàm `xt::random::randn<double>(.)`

```cpp
#include "ann/Layer.h"

class FCLayer: public Layer {
public:
    FCLayer(int Nin=2, int Nout=10, bool use_bias=true);
    FCLayer(const FCLayer& orig);
    virtual ~FCLayer();

    xt::xarray<double> forward(xt::xarray<double> X);
    xt::xarray<double> backward(xt::xarray<double> DY);
    int register_params(IParamGroup* ptr_group);
    bool save(string file);
    bool load(string filename, bool use_bias);


protected:
    virtual void init_weights();

private:
    int m_nNin, m_nNout;
    bool m_bUse_Bias;

    xt::xarray<double> m_aWeights; //N_out x N_in
    xt::xarray<double> m_aBias;

    xt::xarray<double> m_aGrad_W;
    xt::xarray<double> m_aGrad_b;
    xt::xarray<double> m_aCached_X;
    unsigned long long m_unSample_Counter;
};
```

Hình 17: **FCLayer**: lớp kết nối đầy đủ

  - `m_aBias`: **Nếu biến `m_bUse_Bias` là true**, nó phải được tạo ra để có hình dạng $(N_{out},)$, và phải khởi gán là các số 0 [25].

- `m_aGrad_W`: chính là ΔW trong phần 5.2.3.c. Biến này phải được tạo ra và khởi động bằng 0 trong hàm khởi tạo[26].
- `m_aGrad_b`: chính là $\Delta \mathbf{b}$ trong phần 5.2.3.d. **Nếu biến `m_bUse_Bias` là true**, nó phải được tạo ra và khởi động bằng 0 trong hàm khởi tạo[27].
- `m_aCached_X`: **Nếu biến `is_training` là true**, thì hàm `forward` phải lưu trữ X ở đầu vào cho biến này, để dùng đến khi tính ΔW ở hàm `backward`.
- `m_unSample_Counter`: Biến này giữa tổng số mẫu đã được tính **backward**. **Lưu ý**: chỉ cộng dồn số mẫu dữ liệu qua các bó, khi có thực hiện **backward** trên bó. Nghĩa là, `m_unSample_Counter` phải được cập nhật trong hàm **backward** thay cho **forward**.

[25] dùng hàm `xt::zeros<double>(.)`
[26] dùng hàm `xt::random::zeros<double>(.)`
[27] dùng hàm `xt::random::zeros<double>(.)`

```cpp
#include "ann/Layer.h"

class ReLU: public Layer {
public:
    ReLU();
    ReLU(const ReLU& orig);
    virtual ~ReLU();

    xt::xarray<double> forward(xt::xarray<double> X);
    xt::xarray<double> backward(xt::xarray<double> DY);
private:
    xt::xarray<bool> m_aMask;
};
```

Hình 18: **ReLU**: Lớp phi tuyến

Các phương thức của lớp `FCLayer` được giải thích như sau:

- `forward`:
  - Nhận vào `xt::xarray<double> X`
  - Thực hiện:
    - Gán `X` vào `m_aCached_X` nếu như ở chế độ huấn luyện.
    - Tính đáp ứng theo Công thức 12. Lưu ý, không cộng **b** nếu không sử dụng bias. Nên sử dụng hàm **xt::linalg::tensordot** để tính XW^T trong Công thức 12.
- `backward`:
  - Ngõ vào: `xt::xarray<double> DY`. `DY` là đạo hàm của hàm tổn thất so với đầu ra của `FCLayer`.
  - Thực hiện:
    - Cập nhật biến `m_unSample_Counter`
    - Cập nhật biến `m_aGrad_W`, theo Công thức 13.
    - Cập nhật biến `m_aGrad_b` (nếu biến `m_bUse_Bias`) là true), theo Công thức 15
    - Tính và trả về ΔX, theo Công thức 16.

7.1.2 Lớp ReLU

To be updated!

7.1.3 Lớp Sigmoid

```cpp
#include "ann/Layer.h"

class Sigmoid: public Layer {
public:
    Sigmoid();
    Sigmoid(const Sigmoid& orig);
    virtual ~Sigmoid();

    xt::xarray<double> forward(xt::xarray<double> X);
    xt::xarray<double> backward(xt::xarray<double> DY);
private:
    xt::xarray<double> m_aCached_Y;

};
```

Hình 19: **Sigmoid**: Lớp phi tuyến

```cpp
#include "ann/Layer.h"

class Tanh: public Layer  {
public:
    Tanh();
    Tanh(const Tanh& orig);
    virtual ~Tanh();

    xt::xarray<double> forward(xt::xarray<double> X);
    xt::xarray<double> backward(xt::xarray<double> DY);
private:
    xt::xarray<double> m_aCached_Y;
};
```

Hình 20: **Tanh**: Lớp phi tuyến

7.1.4 Lớp Tanh

7.1.5 Lớp Softmax

7.2 Các lớp tổn thất

7.2.1 LossLayer

7.3 LossLayer

```cpp
#include "ann/Layer.h"

class Softmax: public Layer {
public:
    Softmax(int axis=-1);
    Softmax(const Softmax& orig);
    virtual ~Softmax();

    virtual xt::xarray<double> forward(xt::xarray<double> X);
    virtual xt::xarray<double> backward(xt::xarray<double> DY);

private:
    int m_nAxis;
    xt::xarray<double> m_aCached_Y;
};
```

Hình 21: **Softmax**: đầu ra của bộ phân loại

```cpp
#include "ann/xtensor_lib.h"
enum LossReduction{
    REDUCE_NONE = 0,
    REDUCE_SUM,
    REDUCE_MEAN
};

class LossLayer {
public:
    LossLayer(LossReduction reduction=REDUCE_MEAN);
    LossLayer(const LossLayer& orig);
    virtual ~LossLayer();

    virtual double forward(xt::xarray<double> X, xt::xarray<double> t)=0;
    virtual xt::xarray<double> backward()=0;
private:
    LossReduction reduction;
};
```

Hình 22: **LossLayer**: Lớp cha của mọi lớp tổn thất

```cpp
#include "tensor/xtensor_lib.h"

enum LossReduction{
    REDUCE_NONE = 0,
    REDUCE_SUM,
    REDUCE_MEAN
};

class LossLayer {
public:
    LossLayer(LossReduction reduction=REDUCE_MEAN);
    LossLayer(const LossLayer& orig);
    virtual ~LossLayer();

    virtual double forward(xt::xarray<double> X, xt::xarray<double> t)=0;
    virtual xt::xarray<double> backward()=0;
private:
    LossReduction m_eReduction;
};
```

Hình 23: **CrossEntropy**

7.4 Mô hình

7.5 Bộ tối ưu

——————————HẾT——————————