TÌM HIỂU VÀ ĐÁNH GIÁ ĐỘ HIỆU QUẢ CỦA HÀM KÍCH HOẠT RELU VÀ LEAKYRELU TRONG MẠNG NORON NHÂN TẠO

Bùi Khánh Huyền, Trần Thu Thủy

Khoa học máy tính, Học viện kỹ thuật Quân sự

ABSTRACT

Đánh giá ảnh hưởng của hàm kích hoạt trong mô hình mạng noron nhân tạo (ANN) lên bài toán phân loại. Trong báo cáo này sẽ tìm hiểu 2 hàm kích hoạt (ReLU, LeakyReLU) và so sánh với hàm kích hoạt Sigmoid thông qua thử nghiệm mô hình ANN trên tập dữ liệu MNIST. Trong các thử nghiệm cho thấy rằng hàm kích hoạt ReLU và LeakyReLU đạt độ chính xác cao, tương đương nhau, đồng thời cũng đưa ra kết qủa tốt hơn Sigmoid.

***Index Terms*—** ANN, hàm kích hoạt, RELU, LeakyRELU, Sigmoid.

1. INTRODUCTION

Trong báo cáo này, nhóm chúng tôi tìm hiểu 3 hàm kích hoạt (ReLU, LeakyReLU, Sigmoid) và áp dụng cho bài toán sử dụng mô hình ANN, đồng thời so sánh ảnh hưởng của từng hàm kích hoạt đến hiệu năng của mô hình.

Các mô hình ANN đã cho những kết quả rất thành công trong lĩnh vực học máy trong những năm gần đây và góp phần quan trọng vào việc đưa các ứng dụng mới vào cuộc sống của chúng ta. Có thể thấy rằng các vấn đề học máy cơ bản như hồi quy, phân loại và phân cụm trong các bộ dữ liệu khác nhau đều được giải quyết bởi ANN. Trong mô hình ANN, nơi các giải pháp hiệu quả như vậy được tạo ra, có rất nhiều yếu tố như hàm Convolution, Embedding, Power Dissipation, Pooling, Full Graph, Dissipation và một trong những yếu tố quan trọng nhất là hàm kích hoạt (AF). Việc tối ưu hóa cấu trúc mô hình mạng, mà điển hình là việc lựa chọn một hàm kích hoạt thích hợp, là một điều rất cần thiết.

Chúng tôi đã tìm hiểu, so sánh 3 hàm kích hoạt (ReLU[1], LeakyReLU[2], Sigmoid[3]); xây dựng mô hình ANN cơ bản với 4 lớp: 1 lớp đầu vào, 2 lớp ẩn và 1 lớp đầu ra.

Cấu trúc tiếp theo của bài viết được trình bày như sau: mục 2 trình bày về phương thức gồm các hàm kích hoạt và mô hình sẽ được khảo sát; mục 3 kết quả thực nghiệm và đánh giá; mục 4 là phần kết luận.

2. METHOD

2.1. Các hàm kích hoạt

- **Hàm ReLU**

  Rectified Linear Unit (ReLU)[1] được định nghĩa:

$$f(x) = \begin{cases} 0 & (x \leq 0) \quad (1a) \\ x & (x > 0) \quad (1b) \end{cases}$$

có đạo hàm:

$$f'(x) = \begin{cases} 0 & (x < 0) \quad (2a) \\ 1 & (x > 0) \quad (2b) \end{cases}$$

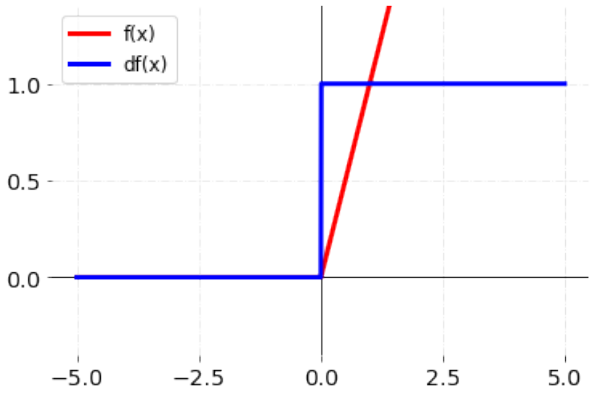

Fig. 1. Hàm ReLU và đạo hàm

Đây là hàm không tuyến tính và không có đạo hàm tại x=0. Tuy nhiên trong tính toán, đạo hàm của ReLU tại x=0 được ngầm định bằng 0 hoặc 1.

ReLU đơn giản lọc các giá trị < 0. Nhìn vào công thức chúng ta dễ dàng hiểu được cách hoạt động của nó. Một số ưu điểm khá vượt trội của nó so với Sigmoid: *Tốc độ hội tụ nhanh hơn hẳn*. ReLU có tốc độ hội tụ nhanh gấp 6 lần Tanh [4]. Điều này có thể do ReLU không bị bão hoà ở 2 đầu như Sigmoid và Tanh; *Tính toán nhanh hơn*. Tanh và Sigmoid sử dụng hàm exp và công thức phức tạp hơn ReLU rất nhiều do vậy sẽ tốn nhiều chi phí hơn để tính toán.

Tuy nhiên ReLU cũng có nhược điểm: Với các node có giá trị nhỏ hơn 0, qua ReLU activation sẽ thành 0, hiện tượng đấy gọi là "Dying ReLU". Nếu các node bị chuyển thành 0 thì sẽ không có ý nghĩa với bước linear activation ở lớp tiếp theo và các hệ số tương ứng từ node đấy cũng không được cập nhật với gradient descent.Vì vậy, Leaky ReLU ra đời. Ngoài ra, khi learning rate lớn, các trọng số (weights) có thể thay đổi theo cách làm tất cả neuron dừng việc cập nhật.

Hàm ReLU có nhiều biến thể khác như *Noisy ReLU, Leaky ReLu, ELUs.*

- **Hàm LeakyReLU**

Leaky Rectified Linear Unit (Leaky ReLU)[2] là một biến thể của ReLU.

Leaky ReLU được định nghĩa như sau:

$$f(x) = \begin{cases} 0.01x & (x \leq 0) \quad (3a) \\ x & (x > 0) \quad (3b) \end{cases}$$

có đạo hàm:

$$f'(x) = \begin{cases} 0.01 & (x < 0) \quad (4a) \\ 1 & (x > 0) \quad (4b) \end{cases}$$

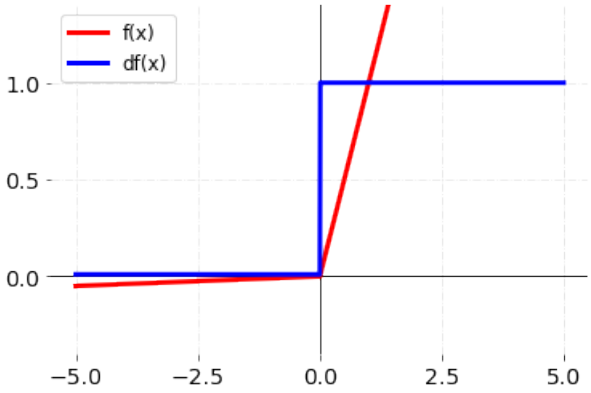

Fig. 2. Hàm LeakyReLU và đạo hàm

Dễ thấy hàm Leaky RELU thay thế phần âm của ReLU bằng một hàm tuyến tính với hệ số góc nhỏ cố định (0.01).

Leaky ReLU là một cố gắng trong việc loại bỏ "dying ReLU". Thay vì trả về giá trị 0 với các đầu vào < 0 thì Leaky ReLU tạo ra một đường xiên có độ dốc nhỏ (Fig2.). Có nhiều báo cáo về việc hiệu Leaky ReLU có hiệu quả tốt hơn ReLU, nhưng hiệu quả này vẫn chưa rõ ràng và nhất quán.

- **Hàm Sigmoid**

Sigmoid được định nghĩa như sau:

$$f(x) = \frac{1}{1+e^{-x}} \quad (5)$$

có đạo hàm:

$$f'(x) = f(x).(1 - f(x)) \quad (6)$$

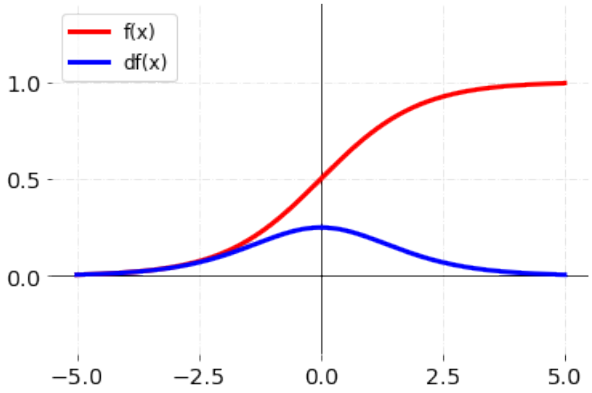

Fig. 3. Hàm Sigmoid và đạo hàm

Hàm Sigmoid nhận đầu vào là một số thực và chuyển thành một giá trị trong khoảng (0;1) (Fig3.). Đầu vào là số thực âm rất nhỏ sẽ cho đầu ra tiệm cận với 0, ngược lại, nếu đầu vào là một số thực dương lớn sẽ cho đầu ra là một số tiệm cận với 1. Trong quá khứ hàm Sigmoid hay được dùng vì có đạo hàm rất đẹp. Tuy nhiên hiện nay hàm Sigmoid rất ít được dùng vì những nhược điểm sau:

- *Hàm Sigmoid bão hào và triệt tiêu gradient:* Khi đầu vào có trị tuyệt đối lớn (rất âm hoặc rất dương, gradient của hàm số này sẽ rất gần với 0. Điều này đồng nghĩa với việc các hệ số tương ứng với unit đang xét sẽ gần như không được cập nhật (còn được gọi là vanishing gradient).
- *Hàm Sigmoid không có trung tâm là 0 gây khó khăn cho việc hội tụ.*

2.2. Mô hình

Chúng tôi xây dựng một mô hình mạng noron cơ bản với tổng cộng 4 lớp: 1 lớp đầu vào, 2 lớp ẩn và 1 lớp đầu ra. Tất cả các lớp đều là full- connected.

Để training mạng noron, chúng tôi sử dụng stochastic gradient descent (SGD); có nghĩa là từng hình ảnh sẽ qua mô hình mạng noron tại một thời điểm.

1. **Lớp đầu vào (Input layer):** Trong lớp này, chúng tôi sử dụng tập dữ liệu của bao gồm các hình ảnh 28x28. Sau đó, làm phẳng (flatten) những hình ảnh này thành một mảng có 28 × 28 = 784 phần tử. Điều này có nghĩa là lớp đầu vào sẽ có 784 nodes.

2. **Lớp ẩn thứ nhất (Hidden layer 1**): Trong lớp này, chúng tôi đã quyết định giảm số lượng nodes từ 784 trong lớp đầu vào xuống còn 128 nodes.

3. **Lớp ẩn thứ hai (Hidden layer 2)**: Trong lớp này, chúng tôi sử dụng 64 nodes, từ 128 nodes trong lớp ẩn đầu tiên.

4. **Lớp đầu ra (Output layer)**: Trong lớp này, chúng tôi giảm 64 nodes xuống tổng số 10 nodes, để có thể đánh giá các nodes dựa trên nhãn.

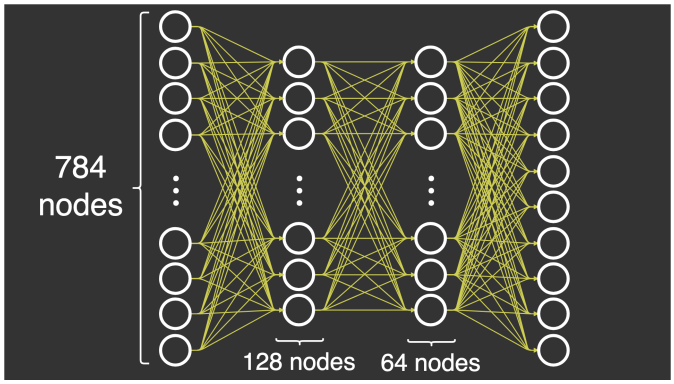

Fig. 4. Mô hình mạng noron

Số lượng nodes trong mỗi lớp giảm từ 784 nodes xuống 128 nodes, sau đó 64 nodes và cuối cùng là còn 10 nodes. Dựa trên các quan sát thực nghiệm, chọn số nodes như vậy sẽ mang lại kết quả tốt để tránh overfitting và underfitting.

Để có thể so sánh được các hàm kích hoạt như đã giới thiệu ở trên, chúng tôi xây dựng lên ba mô hình mạng cho mỗi hàm kích hoạt. Các hàm này sẽ được sử dụng để kích hoạt tại hai lớp ẩn, riêng tại lớp đầu ra ở cả ba mô hình đều sử dụng hàm kích hoạt *softmax* nhằm đánh giá xác suất phân loại của dữ liệu đầu vào.

Chúng tôi chọn sử dụng Stochastic Gradient Descent (SGD) làm trình tối ưu hóa để cập nhật các tham số của mạng.

- Giả mã cho SGD [5]:
  - Chọn vecto khởi tạo cho tham số w và learning rate η.
  - Lặp lại cho đến khi đạt được giá trị tối thiểu gần đúng:
    - Trộn ngẫu nhiên các mẫu trong tập train.
    - For $i = 1, 2, ...n$:
      - $w := w - \eta \nabla Q_i(w)$

Thêm vào đó, hàm loss được sử dụng trong mô hình là *cross-entropy* [6]. Với xác suất đúng $p_i \in \{y, 1-y\}$ là nhãn đúng và phân phối đã cho $q_i \in \{\hat{y}, 1-\hat{y}\}$ là giá trị dự đoán của mô hình hiện tại:

$$-\sum_i p_i \log q_i \;=\; -\,y \log \dot{y} - (1-y)\log(1-\dot{y}) \qquad (7)$$

3. EXPERIMENTAL RESULTS AND ANALYSIS

3.1. Tập dữ liệu

Để đánh giá tác động của các hàm kích hoạt kể trên lên ANN trong bài toán phân lớp hình ảnh, chúng tôi sử dụng tập dữ liệu hình ảnh phổ biến là MNIST

Tập dữ liệu MNIST (Modified National Institute of Standards and Technology)[7] là tập các chữ số viết tay từ 0-9 được số hóa. MNIST bao gồm 60,000 hình ảnh để huấn luyện và 10,000 hình ảnh để kiểm định. Mỗi phần tử của tập là một kí tự số theo hệ màu xám đơn sắc (grayscale) nằm chính giữa một khung hình có kích thước đã được chuẩn hóa (28x28).

Fig. 5. Tập dữ liệu ký tự số MNIST

Nhãn của mỗi phần tử là kí tự số tương ứng trong hình ảnh của phần tử đó. Chúng ta nói rằng có 10 class, vì có 10 nhãn.

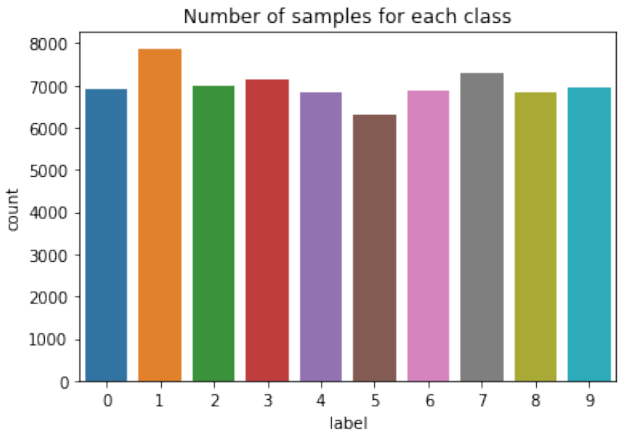

Fig. 6. Số lượng mẫu của mỗi lớp trong tập dữ liệu

Chia tập dữ liệu thành 3 tập con: *train, test, valid* với tỷ lệ *7:2:1*.

3.2. Thử nghiệm

Chúng tôi tiến hành training với *100 epoch* và *batch size* là *64*.

Để đánh giá ảnh hưởng của các hàm kích hoạt chúng tôi dựa trên 4 tham số: *F1, precision, recall và accuracy*.

- **Đánh giá kết quả thực nghiệm**

  Sau khi tiến hành training trên mô hình, chúng tôi so sánh kết quả được thể hiện trong hình dưới đây.

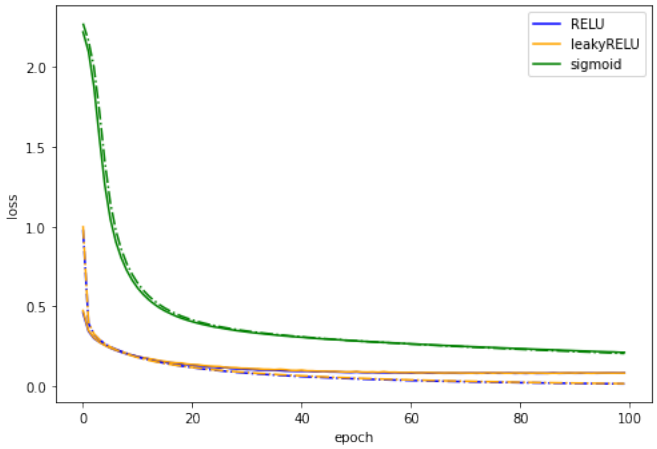

Fig. 7. Giá trị hàm lỗi của ANN trên tập MNIST

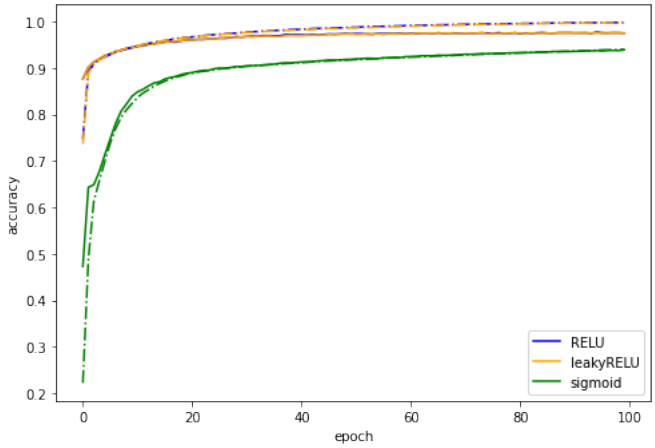

Fig. 8. Độ chính xác của ANN trên tập MNIST

Table 1. Giá trị loss và acc trên tập valid.

	ReLU	LeakyRELU	Sigmoid
Final loss	0.0915	0.0837	0.2138
Final acc	0.9761	0.9763	0.9423

– Việc huấn luyện với ReLU và Leaky ReLU cho tần suất lỗi thấp (tương ứng với độ chính xác cao) khi kiểm thử. Cùng với đó, hàm Sigmoid đem lại tần suất lỗi lớn, độ chính xác cũng thấp hơn 2 hàm trên.

3.3. Phân tích, thảo luận về kết quả

Đánh giá độ hiệu quả của mô hình vừa được training trên tập test, chúng tôi thu được kết quả:

Table 2. Độ hiệu quả dựa trên các tham số.

Model	ReLU	LeakyRELU	Sigmoid
F1	0.9739	0.9758	0.9393
Precision	0.9755	0.9779	0.9526
Recall	0.9723	0.9738	0.9267
Accuracy	0.9738	0.9752	0.9376

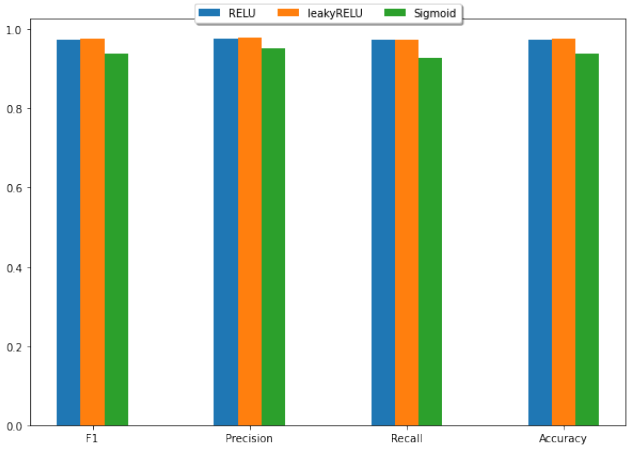

Fig. 9. Độ hiệu quả của các mô hình trên các tham số

Mô hình sử dụng hàm kích hoạt ReLU va LeakyReLU thể hiện rõ độ hiệu quả cao hơn mô hình sử dụng Sigmoid trên cả 4 tham số. Trong khi ReLU và LeakyReLU có sự sai khác không đáng kể, thì Sigmoid lại cho kết quả thấp hơn 2 hàm trên.

Tuy nhiên, ta cũng không thể kết luận rằng Sigmoid là kém hiệu quả, vì trong bài toán phân lớp này, độ chính xác của nó vẫn trên 90% - một kết quả không phải là tệ.

4. CONCLUSION

Qua báo cáo này, chúng tôi đã tìm hiểu về các hàm kích hoạt, cụ thể là ReLU, Leaky ReLU, cũng như Sigmoid. Đồng thời bước đầu đánh giá được tác động của các hàm kích hoạt này đến quá trình huấn luyện mạng noron. Dựa vào kết quả thực nghiệm, đã đánh giá được độ hiệu quả của mỗi hàm và so sánh chúng với nhau. Kết luận lại, hàm ReLU va LeakyReLU đem lại kết quả tốt hơn Sigmoid.

Table 3. Kết luận.

Func	Nhận xét	Sử dụng khi:
ReLU	Hàm được sử dụng phổ biến nhất trong các lớp ẩn. Tuy nhiên, trong những trường hợp hiếm hoi, có thể gặp vấn đề "dying ReLU".	Ưu tiên chọn đầu tiên
LReLU	Là một biến thể của ReLU. Đầu ra không = 0 nên không thể "die".	Khi dự đoán là có "Dying ReLU"
Sigmoid	Bão hòa, khó hội tụ và triệt tiêu gradient	Phân loại nhị phân trong mô hình hồi quy logistic.

5. REFERENCES

[1] Nair, Vinod, and Geoffrey E. Hinton (2010). Rectified linear units improve restricted boltzmann machines.*Proceedings of the 27th international conference on machine learning (ICML-10)*

[2] Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models.*" Proc. icml.* Vol.30.No.1.2013

[3] *https://cs231n.github.io/neural-networks-1/*

[4] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton (2012). *ImageNet Classification with Deep Convolutional Neural Networks.*

[5] Sebastian Ruder, Aylien Ltd., Dublin. *An overview of gradient descent optimization algorithms*

[6] Murphy, Kevin (2012).*Machine Learning: A Probabilistic Perspective*

[7] LeCun, Yann, Corinna Cortes, and C. J. Burges (2010).*MNIST handwritten digit database. ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist 2*