

Activation và loss

☑ Trắc nghiệm 46 ⬇ PDF bài

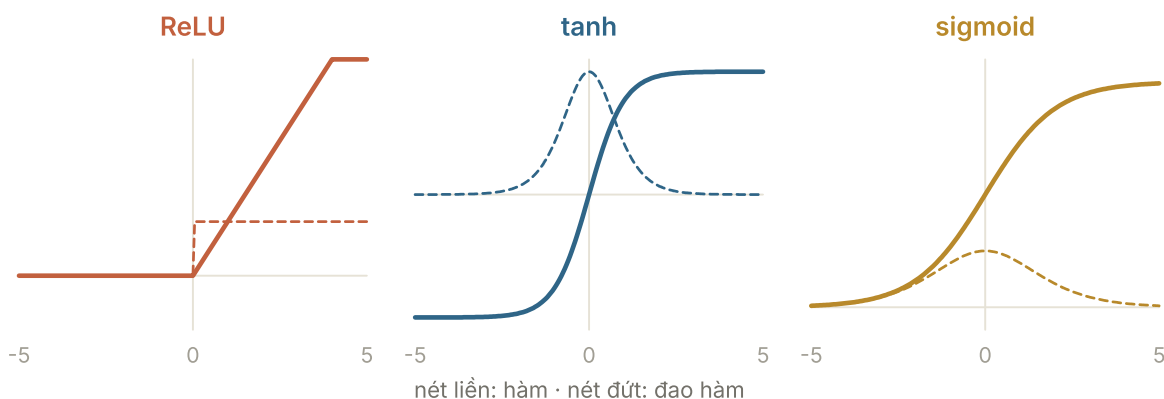
Lớp cuối của mạng phải hợp với loại đầu ra cần dự đoán. Đoán một số thực, như số phút làm bài, thì để nguyên đầu ra và dùng MSE; đoán có/không thì dùng sigmoid với binary cross-entropy; chọn một trong nhiều lớp thì dùng softmax với cross-entropy. Ghép sai thì code vẫn chạy và loss vẫn giảm, nhưng model đang học sai bài toán. Bài này học các activation thường dùng ở lớp ẩn và lớp cuối, và cách chọn loss theo ý nghĩa của nhãn.

MỤC TIÊU

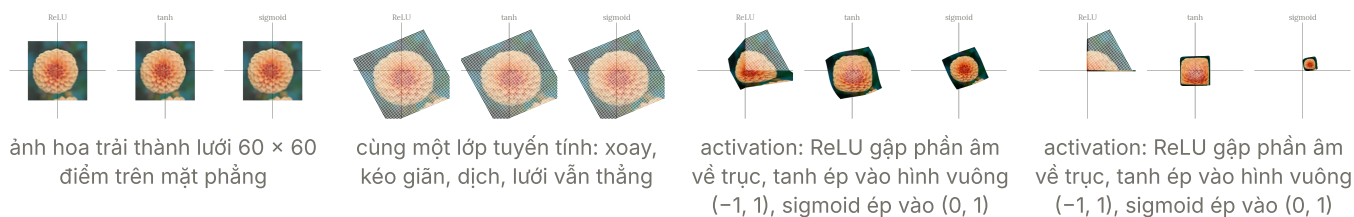
- Chọn activation cho lớp ẩn và cách khởi tạo trọng số.
- Phân biệt logit, xác suất và nhãn.
- Chọn loss theo loại đầu ra: MSE, BCEWithLogitsLoss, CrossEntropyLoss.
- Tránh lỗi áp sigmoid hoặc softmax hai lần.

Activation là một phép biến đổi có mục đích

MLP cần phi tuyến giữa các lớp. ReLU giữ phần dương, đặt phần âm về 0. Tanh đưa số về khoảng $(-1, 1)$; sigmoid đưa về $(0, 1)$. Hãy nhìn đồ thị trước khi thuộc tên: ReLU có một cổng mở hoặc đóng, còn tanh và sigmoid dần nằm ngang ở hai đầu. Khi đầu ra gần như không đổi dù đầu vào đổi, gradient cũng nhỏ.



ReLU, tanh, sigmoid (nét liền) và đạo hàm (nét đứt).



Phép tuyến tính giữ lưới thẳng; activation uốn cong mặt phẳng.

Các công thức là $\text{ReLU}(z) = \max(0, z)$, $\sigma(z) = 1/(1 + e^{-z})$ và $\tanh z = (e^z - e^{-z})/(e^z + e^{-z})$. Đạo hàm của sigmoid và tanh đọc được ngay từ đầu ra: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$, lớn nhất $1/4$ tại 0 ; $\tanh'(z) = 1 - \tanh^2 z$, lớn nhất 1 tại 0 . ReLU có đạo hàm 1 khi $z > 0$ và 0 khi $z < 0$.

Ví dụ một nơ-ron có đầu vào $(2, -1, 3)$, trọng số $(0.5, 2, -1)$, bias 1 : $z = 1 - 2 - 3 + 1 = -3$, ReLU cho 0 . Sigmoid đang ra 0.4 thì đạo hàm là $0.4 \cdot 0.6 = 0.24$. Đạo hàm tanh là $1 - \tanh^2 z \leq 1$, nên đi ngược qua tanh, gradient không bao giờ lớn thêm.

Python

```
import numpy as np
import torch
from torch import nn
from torch.nn import functional as F

torch.set_num_threads(1)
torch.manual_seed(13)
z = torch.tensor([-10., 0., 10.], requires_grad=True)
torch.sigmoid(z).sum().backward()
print('sigmoid:', [round(v, 6) for v in torch.sigmoid(z).tolist()])
print("sigmoid'", [round(v, 6) for v in z.grad.tolist()])
z.grad = None # xóa gradient cũ trước lần backward thứ hai
torch.tanh(z).sum().backward()
print("tanh'", [round(v, 6) for v in z.grad.tolist()])
r = torch.tensor([-2., 0., 3.], requires_grad=True)
torch.relu(r).sum().backward()
print('relu:', torch.relu(r).tolist(), "relu'", r.grad.tolist())
```

Hàng sigmoid' cho 0.25 ở giữa và $4.5 \cdot 10^{-5}$ ở hai đầu; tanh' bằng 1 ở giữa và tròn về 0 trong float32 tại $|z| = 10$. Một neuron ẩn bão hòa (đầu ra nằm ở vùng phẳng) vì thế gần như ngừng học: mọi gradient đi qua nó bị nhân với một số gần 0 .

Đây là độ nhạy của riêng activation, chưa có loss. Ở lớp đầu ra, sigmoid ghép với BCE cho đạo hàm theo logit rút gọn thành $\sigma(z) - y$ (kiểm ở mục nhị phân), nên bão hòa ở lớp cuối không gây cùng vấn đề.

ReLU không bão hòa ở phía dương và tính rất rẻ, vì thế là lựa chọn mặc định cho hidden layer.

Đổi lại, một neuron có pre-activation âm trên mọi mẫu, chẳng hạn sau một bước cập nhật quá lớn đẩy bias xuống rất âm, cho đầu ra 0 và gradient 0 cho chính trọng số của nó. Neuron ấy “chết” và không tự hồi lại. LeakyReLU, $\max(0.01z, z)$, giữ một độ dốc nhỏ phía âm để neuron vẫn nhận gradient. Khi nhiều neuron sigmoid đã bão hòa, giảm learning rate không giúp: gradient nhỏ vì đạo hàm của activation, không phải vì bước đi.

HỎI NHANH

Một neuron ẩn dùng sigmoid có $z \approx 8$ trên hầu hết các mẫu train. Trọng số đi vào neuron ấy được cập nhật thế nào?

- A Rất mạnh, vì z lớn thì gradient lớn
- B Gần như đứng yên, vì $\sigma'(8)$ rất nhỏ
- C Bình thường, vì BCE đã triệt tiêu thừa số σ'
- D Đứng yên hoàn toàn, giống một neuron ReLU đã chết

HỎI NHANH

ReLU không có đạo hàm tại $z = 0$. Vì sao huấn luyện bằng autograd vẫn chạy bình thường?

- A Autograd quy ước một giá trị tại 0, và z rất hiếm khi đúng bằng 0
- B Đạo hàm tại 0 bằng $1/2$, trung bình của hai phía
- C Optimizer tự bỏ qua các mẫu có $z = 0$ trong batch
- D Float32 không biểu diễn được đúng số 0

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 47

Qua nhiều lớp: thang đo và khởi tạo

Một hidden layer tính $z_j = \sum_{i=1}^n w_{ji} a_i$ với n đầu vào (fan-in). Giả sử các trọng số độc lập, trung bình 0, phương sai σ_w^2 , và độc lập với đầu vào. Khi đó các số hạng không tương quan với nhau và

$$\text{Var}(z_j) = n \sigma_w^2 \mathbb{E}[a_i^2].$$

Với ReLU và pre-activation đối xứng quanh 0, nửa âm bị đặt về 0 nên $\mathbb{E}[a^2]$ bằng một nửa phương sai pre-activation của lớp trước. Qua mỗi lớp, phương sai vì thế nhân với $n\sigma_w^2/2$. Hệ số này lặp lại ở mọi lớp nên tác dụng là lũy thừa: nhỏ hơn 1 thì tín hiệu co theo cấp số nhân, lớn hơn 1 thì nổ. Backward đi qua chính các ma trận ấy (chuyển vị) và các mặt nạ ReLU, nên gradient về các lớp đầu cũng co hoặc nổ theo cùng cách.

Đặt hệ số bằng 1 cho

$$\sigma_w = \sqrt{2/n},$$

gọi là khởi tạo He (Kaiming), lựa chọn chuẩn cho mạng ReLU. Phương sai chỉ quyết định thang đo; bản thân các trọng số vẫn phải rút ngẫu nhiên, vì các neuron khởi tạo giống

hệ nhau nhận cùng gradient và giống nhau mãi (bài 3.2).

Với activation khác:

- Tanh gần như là z quanh 0 nên cần $\sigma_w = \sqrt{1/n}$; khởi tạo Xavier (Glorot) lấy trung bình fan-in và fan-out theo cùng tinh thần.
- Sigmoid còn thêm một hệ số: mỗi lớp nhân gradient với $\sigma' \leq 1/4$, nên chồng nhiều lớp sigmoid làm gradient của các lớp đầu tắt dần ngay cả khi trọng số vừa phải.

Vài ví dụ tính fan-in (số đầu vào của mỗi neuron):

- Lớp tuyến tính 512 đầu vào, theo He: $\sigma_w = \sqrt{2/512} = 0.0625$.
- Lớp tích chập 3×3 với 128 kênh vào (bài 5.1): fan-in $9 \cdot 128 = 1152$, nên $\sigma_w = \sqrt{2/1152} \approx 0.042$.
- Xavier dạng chuẩn dùng $\text{Var}(w) = 2/(n_{\text{in}} + n_{\text{out}})$: lớp 300 vào, 100 ra có $\sigma_w = \sqrt{2/400} \approx 0.071$.

Cũng phép tính phương sai ấy cho thấy tích vô hướng của hai vector d chiều độc lập, mỗi thành phần trung bình 0 và phương sai 1, có độ lệch chuẩn $\sqrt{d}$; attention ở bài 3.9 chia cho $\sqrt{d_k}$ vì vậy.

Thí nghiệm dưới đây dùng 10 lớp ReLU rộng 256, không bias, 512 mẫu đầu vào chuẩn. Hai cấu hình dùng cùng các ma trận ngẫu nhiên, chỉ khác hệ số nhân: $\mathcal{N}(0, 0.01^2)$ hoặc He. Loss là trung bình đầu ra cuối, đủ để đo gradient chảy về lớp 1.

Python

```
def depth_probe(std, depth=10, width=256):
    torch.manual_seed(0)  # cùng dữ liệu và cùng ma trận cho mọi std
    h = torch.randn(512, width)
    weights = [(torch.randn(width, width) * std).requires_grad_() for _ in range(depth)]
    scales = []
    for w in weights:
        pre = h @ w  # pre-activation, shape (512, 256)
        scales.append(pre.std().item())
        h = torch.relu(pre)
    h.mean().backward()  # loss đơn giản để đo gradient về lớp 1
    return scales, weights[0].grad.norm().item()

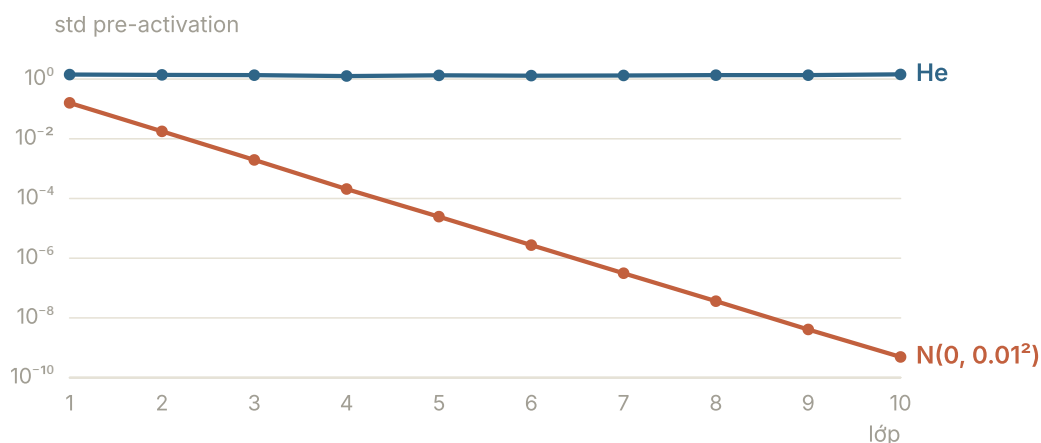
scales_small, grad_small = depth_probe(0.01)
```

```
scales_he, grad_he = depth_probe((2 / 256) ** 0.5)
print('std pre-activation ở lớp 1, 4, 7, 10 và ||grad W1||:')
for name, scales, grad in [('N(0,0.01²)', scales_small, grad_small), ('He', scales_he, grad_he)]:
    print(f'{name:10s}', [float(f'{v:.2g}') for v in scales[::3]], f'{grad:.2g}')
```

Lý thuyết dự đoán được các con số này. Với $\sigma_w = 0.01$ và $n = 256$, phương sai nhân $256 \cdot 10^{-4}/2 = 0.0128$ mỗi lớp, tức độ lệch chuẩn nhân khoảng 0.113. Lớp 1 nhận đầu vào chuẩn nên có std $\sqrt{256} \cdot 0.01 = 0.16$; sau chín lớp nữa còn $0.16 \cdot 0.113^9 \approx 5 \cdot 10^{-10}$, đúng như output. Gradient về lớp 1 nhỏ hơn bản He khoảng $3 \cdot 10^8$ lần, nên mọi bước cập nhật gần như bằng 0 và loss đứng yên. Đầu ra không bằng 0, cũng còn rất xa giới hạn của float32 (cỡ 10^{-38}): tín hiệu chỉ bị co tới mức vô dụng.

Với He, std giữ quanh 1.4 ở mọi lớp.

Hai cấu hình khác nhau đúng hệ số $0.01/\sqrt{2/256} \approx 0.113$ ở mỗi lớp. Vì ReLU thuần nhất dương, $\text{ReLU}(cz) = c \text{ReLU}(z)$ với mọi $c > 0$, nên trong mạng không bias, nhân trọng số của một lớp với c thì mọi lớp phía sau nhân đúng c . Trên trục log, đường của $\mathcal{N}(0, 0.01^2)$ vì thế gần như một đường thẳng đi xuống, cùng độ dốc ở mọi lớp.



Qua 10 lớp ReLU: khởi tạo quá nhỏ làm tín hiệu co dần; He giữ ổn định.

`nn.Linear` của PyTorch mặc định lấy trọng số đều trong $[-1/\sqrt{n}, 1/\sqrt{n}]$, tức độ lệch chuẩn $1/\sqrt{3n}$, nhỏ hơn He. Với mạng ReLU sâu tự viết, gọi

`nn.init.kaiming_normal_(layer.weight, nonlinearity='relu')` để giữ thang đo.

BatchNorm (bài 3.6) và kết nối residual (bài 5.2) là hai cách khác giữ thang đo khi mạng rất sâu.

HỎI NHANH

Mạng ReLU 10 lớp, không có bias. Giữ nguyên đầu vào, nhân mọi trọng số của cả 10 lớp với 2. Đầu ra cuối thay đổi thế nào?

- A Nhân 2, vì mỗi trọng số chỉ nhân 2
- B Nhân 20, vì hệ số cộng dồn qua 10 lớp
- C Nhân $2^{10} = 1024$
- D Không đoán được, vì ReLU là hàm phi tuyến

HỎI NHANH

Gradient đi ngược qua 10 lớp sigmoid. Bỏ qua trọng số, coi chúng bằng 1, thì tích các đạo hàm sigmoid trên đường đi lớn nhất là bao nhiêu?

- A 2.5, tức mười lần $1/4$ cộng lại
- B 0.25, vì chỉ lớp cuối được tính
- C 0, vì sigmoid lúc nào cũng bão hòa
- D $(1/4)^{10} \approx 10^{-6}$

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 55

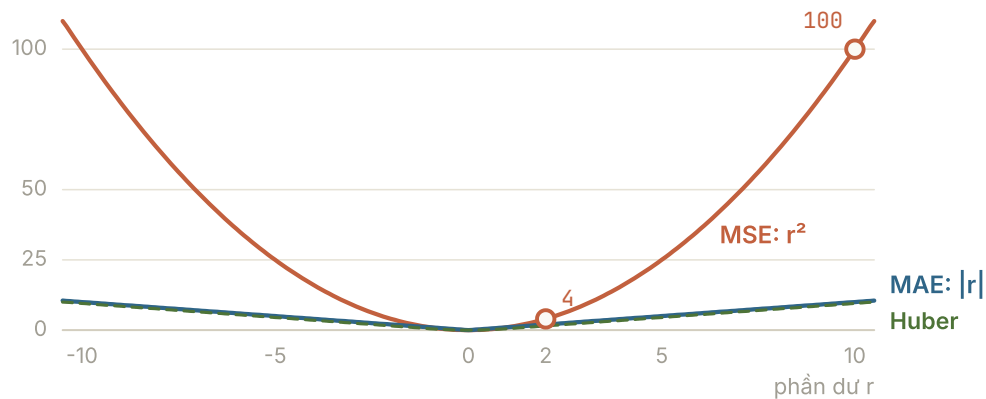
Hồi quy: đo mức sai bằng đơn vị

Đoán thời gian là hồi quy; đầu ra cuối thường để linear, không qua activation nào. Thêm sigmoid ở cuối là ép số phút vào $(0, 1)$ mà không có lý do.

Với phần dư $r = \hat{y} - y$, MSE trung bình r^2 , MAE trung bình $|r|$. MSE có đơn vị phút bình phương, MAE có đơn vị phút. MSE phạt một sai số 10 nặng gấp 25 lần sai số 2, còn MAE

nặng gấp 5 lần: chọn loss là chọn mức phạt cho lỗi lớn.

Nếu model chỉ được đoán một hằng số, MSE chọn trung bình của target, còn MAE chọn trung vị; vài bài có thời gian bất thường vì thế kéo model MSE đi xa hơn nhiều.



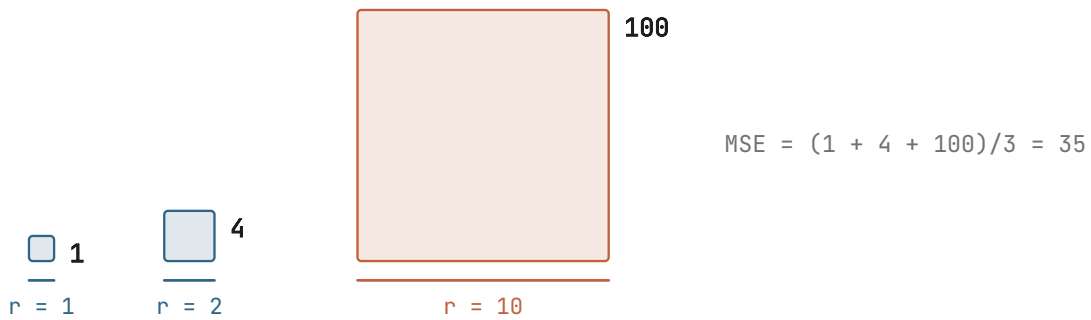
MSE phạt theo bình phương, MAE tuyến tính, Huber ở giữa.

Python

```
prediction = torch.tensor([1., 5., 12.])
target = torch.tensor([2., 3., 2.])
assert prediction.shape == target.shape
print('residual:', (prediction - target).tolist())
print('MSE:', round(F.mse_loss(prediction, target).item(), 4))
print('MAE:', round(F.l1_loss(prediction, target).item(), 4))
print('per item:', F.mse_loss(prediction, target, reduction='none').tolist())
```

Phần dư $(-1, 2, 10)$ cho MSE $(1 + 4 + 100)/3 = 35$ và MAE $(1 + 2 + 10)/3 = 13/3$. Một mẫu cuối đóng phần lớn loss MSE. Kiểm `reduction='none'` giúp phát hiện một dòng dữ liệu bất thường thay vì chỉ nhìn con số trung bình.

Huber loss nối hai cách: gần 0 giống bình phương, xa giống trị tuyệt đối. Khi đề chấm MAE, train bằng L1 hoặc Huber thường khớp metric hơn MSE; chọn cụ thể bằng validation.



Phần dư 1, 2, 10 bình phương thành 1, 4, 100: một mẫu chiếm gần hết MSE.

HỎI NHANH

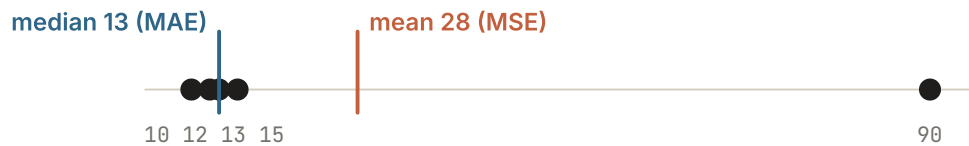
Đổi đơn vị nên target và dự đoán đều nhân 10. MSE thay đổi thế nào?

- A Nhân 10, giống phần dư
- B Nhân 20, vì cả hai đại lượng cùng đổi
- C Giữ nguyên, vì MSE là một trung bình
- D Nhân 100

HỎI NHANH

Năm bài có thời gian hoàn thành là 10, 12, 13, 15 và 90 phút. Nếu mô hình chỉ được đoán một hằng số, thì hằng số tốt nhất theo MSE và theo MAE lần lượt là bao nhiêu?

- A Theo MSE là 28, theo MAE là 13
- B Theo MSE là 13, theo MAE là 28
- C Cả hai đều là 28
- D Cả hai đều là 13



Hàng số tốt nhất: theo MSE là trung bình 28, theo MAE là trung vị 13.

Nhị phân: đầu ra một logit

Đoán “cần đồ thị” dùng nhãn $y \in \{0, 1\}$. Mạng cho một số thực z , gọi là logit. Sigmoid biến z thành xác suất p . Logit 0 tương ứng xác suất 0.5; logit dương nghiêng về lớp 1, âm nghiêng về lớp 0. Logit không bắt buộc nằm trong khoảng 0–1.

Binary cross-entropy là $-[y \log p + (1 - y) \log(1 - p)]$. Nếu tự tính sigmoid rồi log với z rất lớn, làm tròn có thể cho $p = 1$ và $\log(1 - p)$ gặp lỗi số. Dùng BCEWithLogitsLoss để ghép hai phép bằng dạng ổn định:

$$\ell(z, y) = \max(z, 0) - zy + \log(1 + e^{-|z|}).$$

$\log_{1p}$ tính $\log(1 + t)$ ổn hơn khi t rất nhỏ. Ta tự cài dạng này bằng NumPy rồi so thư viện:

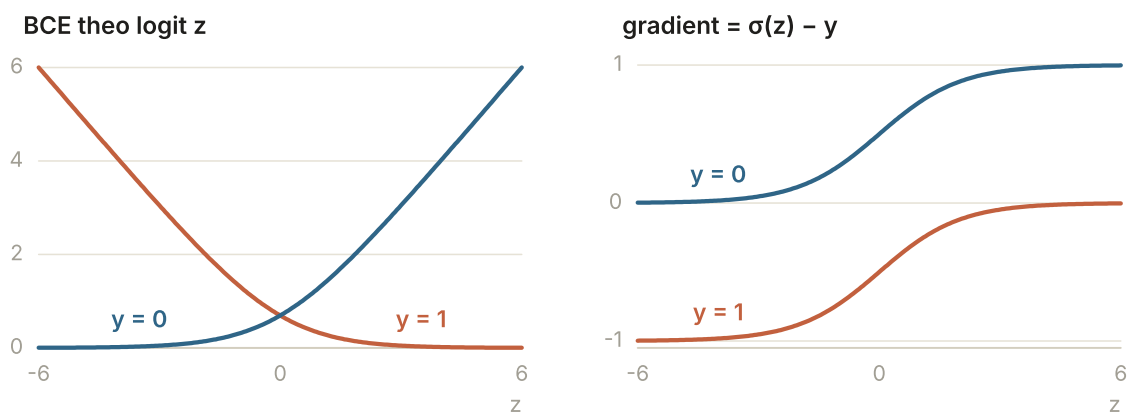
Python

```
logits_np = np.array([-2., 0., 2., -1000., 1000.])
labels_np = np.array([0., 1., 1., 0., 1.])
stable_bce = np.maximum(logits_np, 0) - logits_np * labels_np
stable_bce += np.log1p(np.exp(-np.abs(logits_np)))
logits = torch.tensor(logits_np, requires_grad=True)
labels = torch.tensor(labels_np)
individual = F.binary_cross_entropy_with_logits(logits, labels, reduction='none')
print(np.round(stable_bce, 6).tolist())
print('match:', np.allclose(stable_bce, individual.detach().numpy()))
individual.sum().backward()
print('gradient:', [round(v, 6) for v in logits.grad.tolist()])
```

Loss tại hai logit cực lớn nhưng đoán đúng vẫn hữu hạn. Gradient của tổng loss theo từng logit là $p - y$, nên tại $z = 0, y = 1$ nó là -0.5 . Khi logit rất dương mà target là 0, gradient gần 1: mô hình sai rất tự tin vẫn nhận tín hiệu chỉnh rõ.

Từ đó có ba quy tắc cho lớp cuối của bài nhị phân:

- Lớp cuối là `nn.Linear(d, 1)`, không có sigmoid; sigmoid chỉ gọi lúc suy luận, khi cần xác suất. Ngưỡng 0.5 cho xác suất tương đương ngưỡng 0 cho logit.
- Không đặt ReLU ngay trước đầu ra: logit không bao giờ âm nên $p \geq 0.5$, mô hình không thể đoán lớp 0.
- `nn.Linear(d, 2)` đi với `CrossEntropyLoss`, không đi với `BCEWithLogitsLoss`.



Đoán sai càng tự tin, loss càng lớn và gradient không tắt.

HỎI NHANH

Mô hình có `nn.Sigmoid()` ở lớp cuối, và đầu ra ấy lại được đưa vào `BCEWithLogitsLoss`. Với mẫu có nhãn 0, loss không bao giờ xuống dưới mức nào?

- A 0, khi mô hình rất tự tin đoán đúng
- B 0.313, tức $\log(1 + e^{-1})$
- C Không xác định, vì code báo lỗi kiểu dữ liệu
- D $\log 2 \approx 0.693$

Nhiều lớp: mỗi hàng là một bộ logits

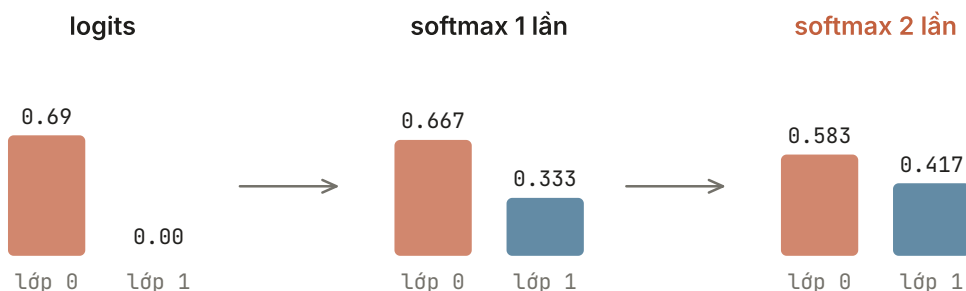
Chọn đúng một độ khó trong ba lớp là multi-class. Mạng trả (B, C) logits. Softmax theo chiều lớp tạo $p_c = e^{z_c} / \sum_j e^{z_j}$, tổng bằng 1. CrossEntropyLoss nhận logits và target là chỉ số lớp (B,) kiểu long; nó đã gộp sẵn log-softmax bên trong. (Loss này cũng nhận target là phân phối xác suất (B, C) kiểu float, dùng cho nhãn mềm; bài này dùng chỉ số lớp.)

Với logits (log 2, 0), softmax là (2/3, 1/3). Nếu nhãn đúng là 0, loss $-\log(2/3) = \log(3/2)$. Hai cách tính sau phải khớp:

Python

```
scores = torch.tensor([[np.log(2.), 0.], [0., np.log(2.)]], dtype=torch.float64)
classes = torch.tensor([0, 1], dtype=torch.long)
probabilities = scores.softmax(dim=1)
hand_loss = -probabilities[torch.arange(2), classes].log().mean()
correct_loss = F.cross_entropy(scores, classes)
double_softmax_loss = F.cross_entropy(probabilities, classes)
print(probabilities.tolist())
print(round(hand_loss.item(), 6), round(correct_loss.item(), 6))
print('softmax twice:', round(double_softmax_loss.item(), 6))
print('prediction:', scores.argmax(dim=1).tolist())
```

Hai loss đều bằng nhau; dòng “softmax twice” khác vì thư viện coi hai xác suất 2/3 và 1/3 như logits mới. Code vẫn chạy, nhưng biến đổi đã nén chênh lệch trước khi loss thực hiện log-softmax. Để lấy lớp có xác suất lớn nhất, `scores.argmax(dim=1)` đủ: softmax giữ thứ tự, không cần tốn thêm phép tính. Bài 10 lớp thì lớp ra có 10 neuron, mỗi neuron một logit.



Softmax hai lần nén (0.667, 0.333) thành (0.583, 0.417).

HỎI NHANH

Đầu ra $(B, 3)$ là logits cho ba lớp loại trừ nhau. Ghép với hàm loss thế nào cho đúng?

- A Qua softmax theo chiều lớp, rồi vào CrossEntropyLoss
- B Qua sigmoid từng logit, rồi vào CrossEntropyLoss
- C Đưa thẳng logits vào CrossEntropyLoss
- D Qua softmax, rồi vào BCEWithLogitsLoss với nhãn one-hot

Một mốc kiểm nhanh: khi mọi logit bằng nhau, mỗi lớp có xác suất $1/C$ và CE bằng $\log C$. Model mới khởi tạo cho logits nhỏ nên loss ở batch đầu nằm gần mốc này: $\log 3 \approx 1.10$ với ba mức khó, $\log 10 \approx 2.30$ với mười lớp. Loss đầu tiên xa hẳn mốc thường báo logits khởi đầu quá lớn hoặc nhãn bị ghép sai.

HỎI NHANH

Bài phân loại 10 lớp, mô hình vừa khởi tạo. CrossEntropyLoss mặc định, lấy trung bình, ở batch đầu tiên bằng 17.5. Nhận định nào hợp lý nhất?

- A Bình thường, có 10 lớp thì loss đầu luôn trên 10
- B Logits khởi đầu quá lớn, vì mức bình thường là gần $\log 10$
- C Learning rate quá nhỏ nên loss chưa kịp giảm
- D Nhãn kiểu long phải đổi thành one-hot kiểu float trước

Nhiều thuộc tính cùng lúc

Một bài có thể thuộc cả “đồ thị” lẫn “DP”: hai thuộc tính không loại trừ nhau. Khi đó mỗi thuộc tính là một nhãn nhị phân riêng, model cho (B, K) logits và BCEWithLogitsLoss với target float cùng shape. Softmax sẽ ép các thuộc tính tranh nhau tổng xác suất 1, trái với ý nghĩa “cả hai đều đúng”.

Khi suy luận, mỗi thuộc tính bật độc lập khi sigmoid của nó vượt ngưỡng; với ngưỡng 0.5, điều đó tương đương logit dương. Lấy argmax chỉ trả về một thuộc tính và làm mất đúng trường hợp “cả hai”.

Tóm lại bốn kiểu đầu ra của bài:

Bài toán	Đầu ra model	Target thông thường	Loss
Hồi quy một số	$(B,)$ float	$(B,)$ float	MSE hoặc MAE
Nhị phân	$(B,)$ logits	$(B,)$ float 0/1	BCEWithLogitsLoss
Một trong C lớp	(B, C) logits	$(B,)$ long	CrossEntropyLoss
K thuộc tính độc lập	(B, K) logits	(B, K) float 0/1	BCEWithLogitsLoss

Tính tay với hai thuộc tính đều đúng và logits bằng 0: mỗi sigmoid là $1/2$, mỗi BCE là $\log 2$. Reduction mean trên hai phần tử vẫn là $\log 2$, còn sum là $2 \log 2$. Mean chia cho số phần tử tham gia: với target (B, K) , đó là BK , không phải B .

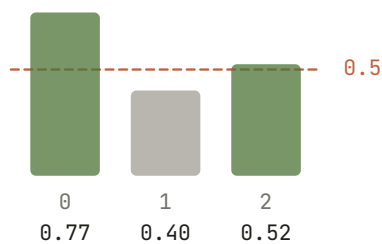
Hai reduction chỉ khác nhau một hằng số nhân, nên có cùng điểm tối ưu; nhân loss với một hằng số dương không đổi nghiệm mà chỉ đổi độ lớn gradient. Với SGD, đổi reduction vì thế tương đương nhân learning rate với hằng số ấy.

HỎI NHANH

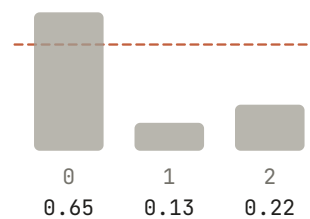
Mô hình đa nhãn cho ba thuộc tính với logits $(1.2, -0.4, 0.1)$, ngưỡng xác suất là 0.5. Những thuộc tính nào được dự đoán?

- A Chỉ $\{0\}$, vì thuộc tính 0 có logit lớn nhất
- B Chỉ $\{0\}$, vì chỉ softmax của nó vượt 0.5
- C $\{0, 2\}$
- D $\{0, 1, 2\}$, vì sigmoid luôn cho số dương

sigmoid từng logit (đa nhãn)



softmax (loại trừ nhau)



Sigmoid bật được nhiều nhãn; softmax bắt các nhãn chia nhau tổng 1.

Khi đi thi

- Model trả logits cho loss; chỉ gọi sigmoid hay softmax khi cần xuất xác suất.
- Kiểm loss ở batch đầu: C lớp thì loss bắt đầu gần $\log C$. Lệch xa thì xem lại logits và nhãn.
- Bài đa nhãn thì chọn ngưỡng cho từng thuộc tính trên validation, không giữ 0.5.

Tóm tắt

- Activation ở lớp ẩn tạo phi tuyến; đạo hàm của nó quyết định gradient đi qua được bao nhiêu. ReLU là mặc định; sigmoid, tanh bão hòa ở hai đầu.
- Khởi tạo phải giữ thang đo qua nhiều lớp: He cho ReLU.
- Hồi quy: đầu ra linear, loss MSE hoặc MAE (mức phạt lỗi lớn khác nhau).
- Nhị phân và đa nhãn: BCEWithLogitsLoss; nhiều lớp loại trừ nhau: CrossEntropyLoss. Cả hai nhận logits, không qua sigmoid hay softmax trước.

Kiểm tra cuối bài

CÂU 1

Target hồi quy là số phút trong khoảng $[0, 120]$, không đổi thang đo. Vì sao lớp cuối dùng sigmoid không phù hợp?

- A Sigmoid không có đạo hàm nên không train được
- B MSE không nhận đầu ra đã qua sigmoid
- C Sigmoid luôn đoán sai các mẫu có target bằng 0
- D Đầu ra bị ép vào $(0, 1)$, không phủ được miền của target

CÂU 2

Tại một điểm x có $\tanh x = 0.6$. Đạo hàm của $\tanh x$ tại điểm ấy bằng bao nhiêu?

A 0.64

B 0.36

C 0.4

D 0.8

PHÒNG THEO USA-NA-AIO 2026 · Vòng 1 · Part 4.1

CÂU 3

Một mạng ReLU 10 lớp được khởi tạo mọi trọng số theo $\mathcal{N}(0, 0.01^2)$. Loss gần như không đổi qua các epoch, còn đổi sang khởi tạo He thì mạng học được. Giải thích nào đúng?

A Trọng số nhỏ khiến ReLU trả toàn số 0, chặn đường đi của gradient

B Trọng số nhỏ gây underflow trong độ chính xác float32

C ReLU chỉ hoạt động đúng khi mọi trọng số đều dương

D Activation co lại theo cấp số nhân, nên gradient tắt dần

ĐỀ THI NOAI 2026 Singapore · Vòng sơ loại · Câu 188

CÂU 4

Xét $y = \sum_{c=0}^{C-1} w_c x_c$, với mọi w_c, x_c độc lập, $\mathbb{E}[w_c] = 0$ và $\text{Var}(w_c) = \sigma_w^2$. Trường hợp (i): x_c có trung bình 0, phương sai σ^2 . Trường hợp (ii): $x_c = \text{ReLU}(u_c)$, với u_c đối xứng quanh 0 và có phương sai σ^2 . Muốn $\text{Var}(y) = \sigma^2$, thì σ_w phải bằng bao nhiêu ở mỗi trường hợp?

A (i) $1/\sqrt{C}$, (ii) $1/\sqrt{2C}$

B (i) $1/\sqrt{C}$, (ii) $\sqrt{2/C}$

C (i) $1/C$, (ii) $2/C$

D (i) $\sqrt{2/C}$, (ii) $1/\sqrt{C}$

PHÒNG THEO USA-NA-AIO 2026 · Vòng 1 · Part 7.2

CÂU 5

Dự đoán số người dự một sự kiện, với g là số đoán và a là số thật. Loss A bằng $\lambda(g - a)^2$ nếu $g > a$, bằng $(g - a)^2$ nếu ngược lại. Loss B bằng $\lambda_1(g - a)^2$ nếu $g > a$, bằng $\lambda_2(g - a)^2$ nếu ngược lại. Mọi hệ số đều dương. Loss B có tạo ra lựa chọn g tối ưu nào mà loss A không tạo được không?

A Có, vì hai tham số luôn mô tả được nhiều hơn một

B Không, vì B chỉ là A nhân với một hằng số dương

C Có, trừ trường hợp đặc biệt $\lambda_1 = \lambda_2$

D Có, vì chỉ B phạt được việc đoán thiếu người

PHÒNG THEO MIT 6.036 · Final, Spring 2022 · Q5(c)

CÂU 6

Bài nhị phân. Mô hình A đưa hai logits $(z_0, z_1) = (1, 3)$ vào CrossEntropyLoss. Mô hình B đưa một logit s vào BCEWithLogitsLoss. s bằng bao nhiêu thì B cho cùng xác suất lớp 1 và cùng loss với A?

A 2

B 3

C 4

D 0.88

PHÒNG THEO CMU 10-601 · Exam 2 Practice · Q2.1

CÂU 7

Sau một giai đoạn train với learning rate lớn, 40% neuron ReLU của một lớp ẩn cho ra 0 trên mọi mẫu train, và không hồi lại qua nhiều epoch. Đây là hiện tượng gì, và giảm nó bằng cách nào?

A Overfitting, cần thêm dữ liệu

B Learning rate quá nhỏ, cần tăng learning rate

C Neuron chết, có thể dùng LeakyReLU hoặc giảm learning rate

D Vanishing gradient, nên đổi sang sigmoid cho mượt

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 192

CÂU 8

Bài đa nhãn: logits và targets cùng shape (B, K), target là 0 hoặc 1. Có ba cách viết loss:

```
# A
p = torch.sigmoid(logits)
loss_a = -(targets * p.log() + (1 - targets) * (1 - p).log())).mean()
# B
loss_b = F.binary_cross_entropy_with_logits(logits, targets)
# C
q = torch.softmax(logits, dim=1)
loss_c = -(targets * q.log()).sum(dim=1)).mean()
```

Dùng float32, và có logit lên tới 120. Nên dùng cách nào để train?

- A A, vì viết rõ công thức nên dễ kiểm hơn B
- B A hoặc C, vì cả hai đều là cross-entropy
- C B, cùng một hàm với A nhưng ổn định về số
- D C, vì softmax bảo đảm các xác suất cộng lại bằng 1

PHÒNG THEO [NOAI 2026 Singapore](#) · [Chung kết](#) · [Câu 7](#)

CÂU 9

Mô hình đa nhãn train với batch (64, 5) và BCEWithLogitsLoss mặc định, tức `reduction='mean'`. Đổi sang `reduction='sum'`, giữ nguyên SGD và learning rate. Bước cập nhật đầu tiên lớn gấp bao nhiêu lần?

- A 1 lần
- B 5 lần
- C 64 lần
- D 320 lần

CÂU 10

Một mẫu có nhãn lớp 0. Xác suất của hai mô hình lần lượt là A: (0.6, 0.4) và B: (0.8, 0.2), cả hai đều đoán đúng theo argmax. So cross-entropy và accuracy trên mẫu này thế nào?

- A B có accuracy cao hơn, vì 0.8 lớn hơn 0.6
- B B có CE thấp hơn, còn accuracy bằng nhau
- C A có CE thấp hơn, còn accuracy bằng nhau
- D CE bằng nhau, vì nhãn dự đoán giống nhau

Đọc thêm

- [PyTorch 2.9 – CrossEntropyLoss](#): tra quy ước logits, target index và reduction.
- [PyTorch 2.9 – BCEWithLogitsLoss](#): đọc vì sao gộp hai phép ổn định hơn và cách dùng pos_weight.
- [D2L – MLP](#): xem phi tuyến, activation và mạng nhiều lớp từ các ví dụ nhỏ.
- [D2L – Numerical Stability and Initialization](#): vanishing/exploding gradient và lập luận phương sai dẫn tới Xavier.
- [PyTorch 2.9 – torch.nn.init](#): tra kaiming_normal_, xavier_uniform_ và tham số nonlinearity.
- He, Zhang, Ren, Sun (2015) – [Delving Deep into Rectifiers](#): bài báo gốc của khởi tạo He cho mạng ReLU sâu.