

Autoencoder

☑ Trắc nghiệm 12 ☑ Thực hành 1 ⬇ PDF bài

Autoencoder là mạng học cách nén dữ liệu thành ít số hơn rồi dựng lại dữ liệu ban đầu từ bản nén. Ví dụ bốn cảm biến cùng thay đổi theo mức hoạt động của máy thì có thể nén thành một số mà vẫn dựng lại gần đúng. Mẫu nào dựng lại sai nhiều là mẫu khác với dữ liệu đã học, nên autoencoder hay được dùng để phát hiện bất thường. Nhưng sai số dựng lại lớn chỉ là dấu hiệu cần kiểm, chưa phải kết luận máy hỏng.

MỤC TIÊU

- Hiểu encoder, mã nén, decoder và loss dựng lại.
- So autoencoder tuyến tính với PCA.
- Dùng sai số dựng lại để phát hiện bất thường; chọn ngưỡng trên validation.

Nén là một ràng buộc thông tin

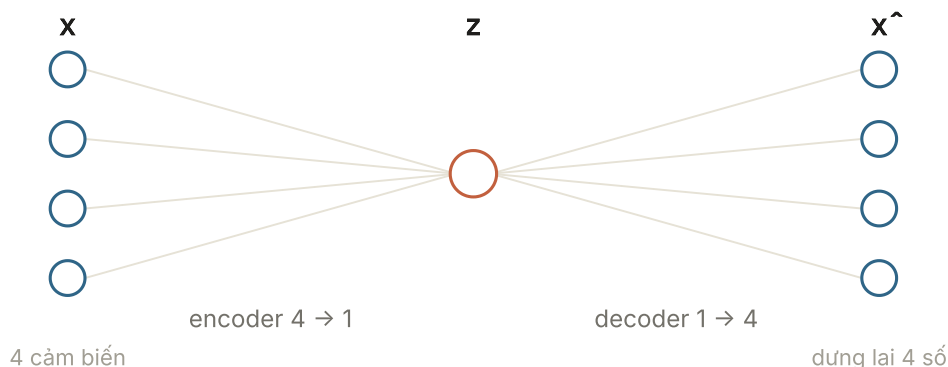
Encoder f_θ đổi $\mathbf{x}$ thành code $\mathbf{z}$ ít chiều hơn; decoder g_ϕ đổi code thành $\hat{\mathbf{x}}$. Training cực tiểu sai khác giữa input và reconstruction, trung bình trên batch B mẫu, mỗi mẫu d feature:

$$\mathbf{z} = f_\theta(\mathbf{x}), \quad \hat{\mathbf{x}} = g_\phi(\mathbf{z}), \quad J = \frac{1}{Bd} \sum_{i,j} (\hat{x}_{i,j} - x_{i,j})^2.$$

Code ít chiều, còn gọi là biểu diễn ẩn (latent), tạo nút thắt (bottleneck): model không đủ chỗ chép mọi chi tiết nên phải giữ cấu trúc chung. Ảnh chữ số 28×28 nén về 32 chiều là bớt từ 784 xuống 32 số, khoảng 24.5 lần. Với ảnh, lỗi MSE cộng nút thắt hẹp thường làm ảnh tái tạo mờ: chi tiết nào không giữ được, model trả trung bình của chúng.

Nếu latent rộng bằng input hoặc encoder/decoder quá mạnh, model có thể học phép chép lại và tái tạo tốt cả mẫu lạ; vì vậy loss tái tạo nhỏ không tự chứng minh code hữu

ích. Autoencoder bỏ thông tin theo loss, nên có thể bỏ luôn thứ mà một nhiệm vụ khác cần.



Autoencoder 4 $\rightarrow$ 1 $\rightarrow$ 4: nén bốn số thành một, rồi dựng lại.

Hai nửa của model có thể chạy ở hai nơi. Khi nén để truyền, thiết bị chạy encoder và chỉ gửi $\mathbf{z}$; nơi nhận giữ decoder để dựng $\hat{\mathbf{x}}$. Khi phát hiện bất thường, hai nửa chạy liền nhau, vì score phải so $\hat{\mathbf{x}}$ với chính $\mathbf{x}$.

HỎI NHANH

Một trạm có 100 cảm biến, đường truyền chỉ đủ gửi 8 số mỗi giây, còn máy chủ cần dựng lại đủ 100 số. Dùng autoencoder 100 $\rightarrow$ 8 $\rightarrow$ 100 thế nào?

- A Trạm chạy cả hai nửa rồi gửi $\hat{\mathbf{x}}$ đủ 100 số
- B Trạm chạy encoder và gửi $\mathbf{z}$, máy chủ chạy decoder
- C Trạm gửi 8 cảm biến đầu, máy chủ chạy encoder
- D Trạm chạy decoder và gửi 8 số, máy chủ chạy encoder

Dữ liệu có một cấu trúc biết trước

Ta sinh bốn cảm biến cùng phụ thuộc một số t : $[t, 2t, -t, 0.5t]$ cộng nhiễu nhỏ. Cấu trúc sạch thật sự chỉ có một bậc tự do, nên nén về một chiều có lý do. Train, validation và test

được sinh riêng; mean/std chỉ lấy train. Các vector không bị ép vào 0–1, vì vậy decoder cuối để linear, không dùng sigmoid.

Python

```
import numpy as np
import torch
from torch import nn

torch.set_num_threads(1)
torch.manual_seed(43)
g = torch.Generator().manual_seed(43)
direction = torch.tensor([1., 2., -1., .5])

def make_normal(n):
    t = torch.randn(n, 1, generator=g)
    clean = t * direction
    noisy = clean + .03 * torch.randn(n, 4, generator=g)
    return clean, noisy

clean_train, raw_train = make_normal(128)
_, raw_valid = make_normal(48)
_, raw_normal_test = make_normal(32)
_, raw_anomaly_test = make_normal(16)
raw_anomaly_test[:, 1] += 1.5
mean = raw_train.mean(dim=0)
scale = raw_train.std(dim=0, unbiased=False).clamp_min(1e-6)
train = (raw_train - mean) / scale
valid = (raw_valid - mean) / scale
test = (torch.cat([raw_normal_test, raw_anomaly_test]) - mean) / scale
test_labels = torch.cat([torch.zeros(32), torch.ones(16)]).long()
print(tuple(train.shape), tuple(valid.shape), tuple(test.shape))
```

Các mẫu bất thường (anomaly) được tạo bằng cách lệch riêng cảm biến thứ hai. Đây là định nghĩa bất thường trong bộ mô phỏng này. Trong dữ liệu thật, nhãn hoặc hiểu biết chuyên môn phải quyết định điều gì đáng báo; một giá trị xa trung bình vẫn có thể là hoạt động hợp lệ.

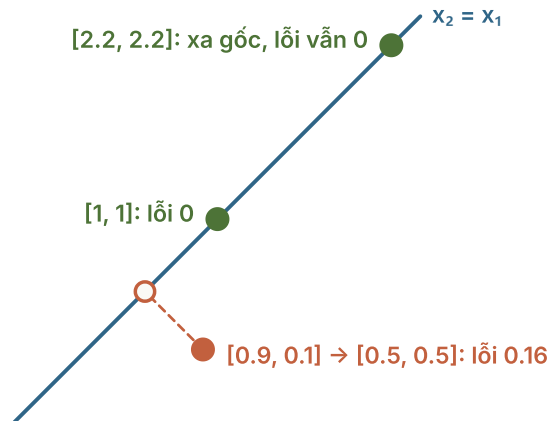
Chuẩn hóa quan trọng vì MSE cộng theo feature: cảm biến có đơn vị lớn sẽ chi phối nếu không chọn thang đo hợp lý.

HỎI NHANH

Target của autoencoder đã chuẩn hóa z-score, nên có giá trị âm và giá trị lớn hơn 1. Vì sao decoder để lớp cuối linear lại hợp hơn sigmoid?

- A Sigmoid không cho gradient khi input của nó âm
- B MSE chỉ tính được khi decoder không có bias
- C Linear tự bảo đảm phát hiện được mọi bất thường
- D Sigmoid chỉ cho ra số trong $(0, 1)$, không phủ được miền của target

Tính thử với một encoder đặt tay. Với hai cảm biến sạch trên đường $x_2 = x_1$, chọn encoder $z = (x_1 + x_2)/2$ và decoder (z, z) . Mẫu $[1, 1]$ tái tạo đúng. Mẫu $[0.9, 0.1]$ cho $z = 0.5$, tái tạo $[0.5, 0.5]$, $\text{MSE}((0.4)^2 + (-0.4)^2)/2 = 0.16$. Phép chiếu bỏ phần lệch khỏi cấu trúc chung.



Điểm trên đường $x_2 = x_1$ được dựng lại đúng; điểm lệch khỏi đường thì có lỗi.

Nếu một mẫu lạ vẫn nằm trên đường ấy, ví dụ $[100, 100]$, công thức trên tái tạo đúng. Một anomaly về **độ lớn dọc cấu trúc** có thể không được reconstruction error phát hiện. Đây là giới hạn cần hiểu trước khi gọi model là bộ phát hiện bất thường.

HỎI NHANH

Vẫn dùng encoder $z = (x_1 + x_2)/2$ và decoder (z, z) . Xếp ba mẫu $A = [5, 5]$, $B = [1.2, 0.8]$, $C = [0, 0.6]$ theo MSE tái tạo, từ lớn đến nhỏ.

A $A > B > C$

B $A > C > B$

C $C > B > A$

D $B > C > A$

Tự cài baseline PCA trước

Với dữ liệu đã trừ mean, PCA một chiều chính là một autoencoder tuyến tính viết sẵn: encoder $z = \mathbf{e}^\top \mathbf{x}$ chiếu lên hướng chính $\mathbf{e}$ (vector đơn vị), decoder $\hat{\mathbf{x}} = z \mathbf{e}$ chiếu ngược. Theo định lý Eckart–Young, trong mọi cách nén tuyến tính về k chiều rồi dựng lại, k thành phần chính đầu tiên cho MSE trên train nhỏ nhất.

Vì vậy PCA vừa là baseline vừa là cận dưới cho autoencoder tuyến tính: mạng tối ưu tốt sẽ học cùng phép chiếu, dù latent của nó có thể bằng tọa độ PCA nhân một hệ số khác 0.

Python

```
train_np = train.numpy()
_, singular_values, vt = np.linalg.svd(train_np, full_matrices=False)
axis = vt[0]
latent_np = train_np @ axis
reconstruction_np = latent_np[:, None] * axis[None, :]
print('PCA latent shape:', latent_np.shape)
print('PCA train MSE:', round(float(np.mean((reconstruction_np - train_np) ** 2)), 6))
print('variance fraction:', round(float(singular_values[0] ** 2 /
                                         np.sum(singular_values ** 2)), 6))
```

Hai số ăn khớp nhau. Sau chuẩn hóa, mỗi cột có phương sai 1, tổng phương sai là 4; MSE trung bình trên bốn cột đúng bằng phần phương sai bị bỏ: $1 - 0.998994 = 0.001006$. Tỷ lệ

giữ lại cao phù hợp thiết kế một biến t điều khiển bốn cảm biến.

Nhưng phương sai lớn chưa chắc mang thông tin cho nhãn (bài 2.9): nhãn có thể phụ thuộc đúng một hướng ít phương sai mà phép nén đã bỏ. Baseline này kiểm mục tiêu reconstruction, không thay cho đánh giá của nhiệm vụ dùng code về sau.

HỎI NHANH

Hai linear autoencoder $4 \rightarrow 1 \rightarrow 4$ không bias cho cùng một reconstruction trên mọi mẫu, nhưng latent của model B luôn gấp đôi latent của model A. Điều này có mâu thuẫn không?

- A Không, vì decoder của B chia đôi để bù lại
- B Có, vì latent tối ưu phải trùng tọa độ PCA
- C Có, vì latent gấp đôi thì output cũng gấp đôi
- D Có, vì chỉ đổi seed mới làm latent khác đi

HỎI NHANH

Dữ liệu 5 cột đã chuẩn hóa, mỗi cột có phương sai 1. PCA giữ 2 thành phần, giải thích được 90% phương sai. MSE tái tạo trên train, lấy trung bình theo mẫu và theo cột, bằng bao nhiêu?

- A 0.02
- B 0.1
- C 0.5
- D 0.9

Học encoder/decoder bằng PyTorch

Ta dùng mạng linear $4 \rightarrow 1 \rightarrow 4$ không bias, vì dữ liệu đã center theo train và cấu trúc thật chỉ có một bậc tự do. Muốn học cấu trúc cong, autoencoder cần tầng ẩn có activation như ReLU; xếp thêm tầng linear mà không có activation chỉ tạo ra một phép tuyến tính khác, không vượt được PCA. Với dữ liệu thẳng như ở đây, mạng nhỏ nhất là đủ để kiểm pipeline.

Python

```
class LinearAutoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Linear(4, 1, bias=False)
        self.decoder = nn.Linear(1, 4, bias=False)

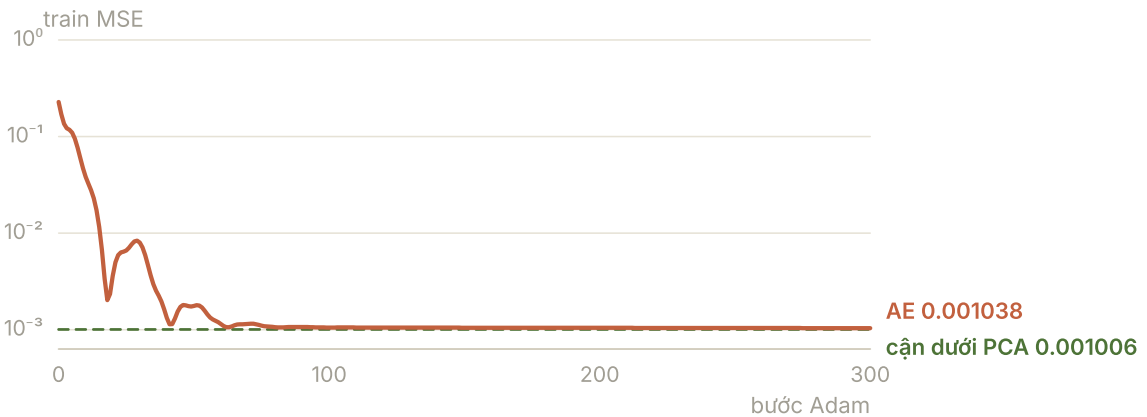
    def forward(self, x):
        return self.decoder(self.encoder(x))

autoencoder = LinearAutoencoder()
optimizer = torch.optim.Adam(autoencoder.parameters(), lr=.03)
initial_loss = (autoencoder(train) - train).square().mean().item()
for _ in range(300):
    autoencoder.train()
    optimizer.zero_grad(set_to_none=True)
    reconstructed = autoencoder(train)
    assert reconstructed.shape == train.shape
    loss = (reconstructed - train).square().mean()
    loss.backward()
    optimizer.step()
autoencoder.eval()
with torch.no_grad():
    final_loss = (autoencoder(train) - train).square().mean().item()
    valid_loss = (autoencoder(valid) - valid).square().mean().item()
    codes = autoencoder.encoder(valid)
print('train MSE:', round(initial_loss, 6), '->', round(final_loss, 6))
print('validation MSE:', round(valid_loss, 6))
print('latent:', tuple(codes.shape))
```

Train MSE giảm từ 0.22791 xuống 0.001038, chỉ cao hơn cận dưới PCA 0.001006 khoảng 3%: 300 bước Adam đã tới rất gần phép chiếu tối ưu. Validation 0.00094 thấp hơn train

một chút; với 48 mẫu, chênh lệch cỡ này là dao động ngẫu nhiên của nhiễu, không phải dấu hiệu rõ ràng.

codes có shape $(48, 1)$, một số mỗi mẫu; dấu và thang của nó có thể đổi theo seed, nên khi dùng code làm feature cho clustering hay classifier, phải đánh giá lại trên split của nhiệm vụ ấy.



Loss của linear autoencoder giảm sát mức PCA nhưng không xuống dưới.

HỎI NHANH

Sau khi train lâu hơn, một bạn báo linear autoencoder $4 \rightarrow 1 \rightarrow 4$ không bias đạt MSE 0.0009 trên train, trong khi trên đúng tập train ấy, PCA một chiều cho 0.001006. Nên nghĩ gì trước tiên?

- A Hợp lý, vì mạng có 8 weight nên khớp tốt hơn PCA
- B Hợp lý, vì Adam tìm được nghiệm mà SVD bỏ sót
- C Đáng ngờ, vì PCA là cận dưới, nên phải kiểm lại phép đo
- D Đáng ngờ, vì linear AE luôn có MSE gấp đôi PCA

Từ reconstruction error tới ngưỡng báo

Bài toán này gọi là phát hiện bất thường (anomaly detection): hợp khi gần như chỉ có dữ liệu bình thường, còn lỗi thì hiếm và mỗi lần một kiểu, như máy hỏng, vết xước trên dây chuyền hay giao dịch gian lận.

Tính lỗi cho **từng mẫu**, trung bình theo feature, không gộp mất chiều batch. Chọn ngưỡng trên validation bình thường, chẳng hạn phân vị 95%; đây là cách đặt mức báo tương đối, không bảo đảm tỉ lệ báo nhầm đúng 5% trên mọi test. Validation nhỏ làm ngưỡng dao động; dữ liệu lệch phân phối có thể tăng số báo sai.

Cách cổ điển hơn là khớp một phân phối chuẩn cho từng đặc trưng rồi báo khi $p(\mathbf{x}) < \varepsilon$; ε càng lớn thì càng ít bỏ sót nhưng càng báo nhầm nhiều.

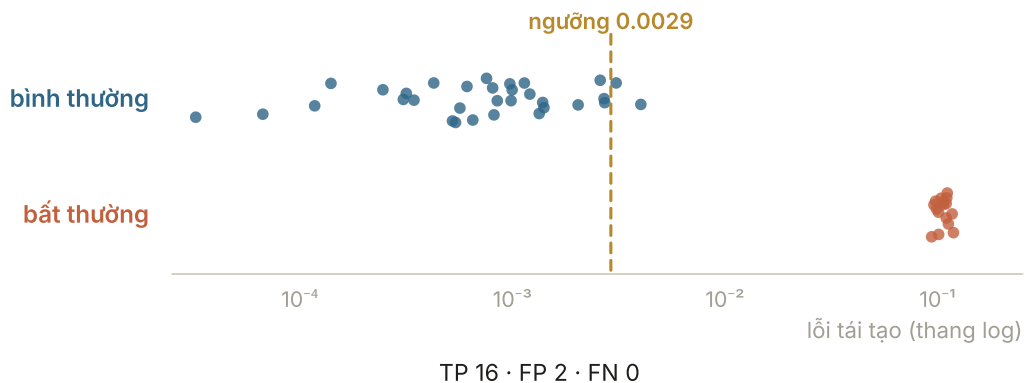
Python

```
def reconstruction_error(model, x):
    model.eval()
    with torch.no_grad():
        return (model(x) - x).square().mean(dim=1)

valid_error = reconstruction_error(autoencoder, valid)
threshold = torch.quantile(valid_error, .95).item()
test_error = reconstruction_error(autoencoder, test)
flag = test_error > threshold
true_positive = (flag & test_labels.eq(1)).sum().item()
false_positive = (flag & test_labels.eq(0)).sum().item()
false_negative = ((~flag) & test_labels.eq(1)).sum().item()
print('threshold:', round(threshold, 6))
print('normal/anomaly median error:', round(test_error[:32].median().item(), 6),
      round(test_error[32:].median().item(), 6))
print('TP FP FN:', true_positive, false_positive, false_negative)
```

Hai median cách nhau hơn trăm lần (0.000808 so với 0.106747), nên score tách hai nhóm rất rõ. Ngưỡng bắt đủ 16 mẫu bất thường (FN = 0) và báo nhầm 2 trong 32 mẫu bình thường, tức 6.25%, gần mức 5% mà phân vị 95% nhắm tới; với 32 mẫu, mỗi mẫu đã là hơn 3 điểm phần trăm.

Nhãn test chỉ dùng để chấm, không dùng chọn ngưỡng. Nếu có validation chứa anomaly đã gán nhãn, chọn ngưỡng tối đa F1 trên đó cũng hợp lệ, nhưng khi ấy bài toán đã dùng nhãn, không còn là phát hiện bất thường không giám sát.



Lỗi tái tạo trên test; vạch đứt là ngưỡng 95% của validation.

Trên thang log, nhóm bất thường nằm cách ngưỡng hơn một bậc độ lớn, còn ngưỡng nằm sát đuôi của nhóm bình thường. Vì vậy dời ngưỡng trong vùng này chủ yếu đổi số FP, hầu như không đổi TP.

HỎI NHANH

Bình phương sai số reconstruction có shape $(B, 4)$. Muốn mỗi mẫu có một MSE anomaly score, phải gộp theo chiều nào?

- A Lấy mean theo $\text{dim}=0$, giữ lại 4 số
- B Lấy sum theo $\text{dim}=0$, rồi chia cho B
- C Lấy mean trên mọi chiều, chỉ còn một số
- D Lấy mean theo $\text{dim}=1$, giữ lại B số

HỎI NHANH

Đổi ngưỡng từ phân vị 95% lên phân vị 99% của cùng một tập validation, giữ nguyên model. Trên cùng tập test, câu nào về TP và FP luôn đúng?

A TP không giảm, FP không tăng

B TP không giảm, FP không giảm

C TP không tăng, FP không giảm

D TP không tăng, FP không tăng

Denoising thay target của việc học

Autoencoder thông thường nhận $\mathbf{x}$ và tái tạo $\mathbf{x}$; denoising autoencoder nhận bản bị nhiễu $\tilde{\mathbf{x}}$ nhưng target là $\mathbf{x}$ sạch. Nếu target cũng nhiễu, model được thưởng khi giữ nhiễu. Lọc tiếng gõ phím, tiếng quạt mà giữ giọng nói trong ứng dụng họp trực tuyến là cùng bài toán trên âm thanh. Khi có cặp noisy/clean hoặc cơ chế mô phỏng nhiễu hợp lệ, ta cực tiểu $\|g(f(\tilde{\mathbf{x}})) - \mathbf{x}\|^2$.

Mô phỏng có sẵn `clean_train`, nên mỗi step ta tạo một bản nhiễu mới với độ lệch chuẩn 0.15 trên thang đã chuẩn hóa; model không thể học thuộc một bản nhiễu cố định. Đánh giá trên 24 cặp clean/noisy mới, dùng lại mean/std của train, và so với cách đơn giản nhất là giữ nguyên noisy input:

Python

```
torch.manual_seed(47)
denoiser = LinearAutoencoder()
denoise_optimizer = torch.optim.Adam(denoiser.parameters(), lr=.03)
clean_scaled = (clean_train - mean) / scale
noise_generator = torch.Generator().manual_seed(47)
for _ in range(200):
    corrupted = clean_scaled + .15 * torch.randn(clean_scaled.shape,
generator=noise_generator)
    denoise_optimizer.zero_grad(set_to_none=True)
```

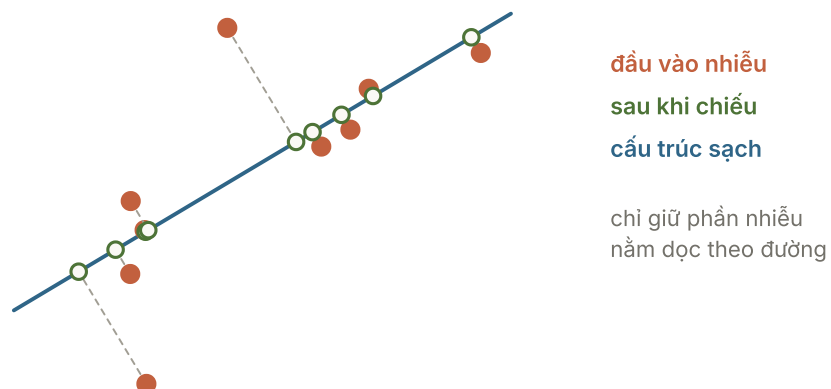
```

loss = (denoiser(corrupted) - clean_scaled).square().mean()
loss.backward()
denoise_optimizer.step()
clean_eval, _ = make_normal(24)
clean_eval = (clean_eval - mean) / scale
noisy_eval = clean_eval + .15 * torch.randn(clean_eval.shape, generator=noise_generator)
denoiser.eval()
with torch.no_grad():
    baseline = (noisy_eval - clean_eval).square().mean().item()
    denoised = (denoiser(noisy_eval) - clean_eval).square().mean().item()
print('noisy input MSE:', round(baseline, 6))
print('denoised MSE:', round(denoised, 6))

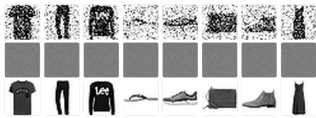
```

Lỗi giảm từ 0.019573 xuống 0.005335, khoảng 3.7 lần. Hình học giải thích con số này: nhiều Gaussian đẳng hướng rải đều năng lượng lên cả bốn chiều, còn tín hiệu sạch nằm trên một đường. Chiếu lên đường ấy chỉ giữ lại khoảng 1/4 năng lượng nhiễu, $0.019573/4 \approx 0.0049$, gần mức 0.005335 đo được.

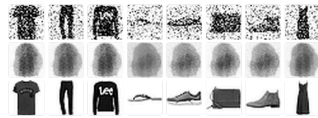
Cơ chế này cần hai điều kiện: cấu trúc sạch ít chiều, và nhiễu lệch khỏi cấu trúc ấy. Với một dạng nhiễu khác, hoặc dữ liệu có chi tiết thật nằm ngoài cấu trúc đã học, phải đo lại trên đúng dữ liệu đó: phép chiếu xóa mọi thứ lệch khỏi đường, kể cả chi tiết thật.



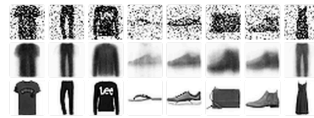
Chiếu lên đường tín hiệu bỏ được phần nhiễu vuông góc với nó.



bước 0: MSE output so với ảnh
sạch 0.1919 (input nhiễu:
0.0655)



bước 30: MSE output so với ảnh
sạch 0.0909 (input nhiễu:
0.0655)



bước 120: MSE output so với
ảnh sạch 0.0344 (input nhiễu:
0.0655)

Denoising autoencoder: hàng trên có nhiễu, hàng giữa là output khi học,
hàng dưới là ảnh sạch.

HỎI NHANH

Một đội viết vòng train denoising, nhưng dòng loss lại là `loss = (model(corrupted) - corrupted).square().mean()`. Mục tiêu học đã thành gì?

- A Vẫn dựng lại bản sạch, chỉ là hội tụ chậm hơn
- B Dự đoán riêng phần nhiễu để trừ đi sau
- C Dựng lại chính input có nhiễu, như autoencoder thường
- D Không học gì, vì loss bằng 0 ngay từ đầu

HỎI NHANH

Tín hiệu sạch nằm trên một mặt phẳng 2 chiều trong không gian 10 chiều. Nhiễu Gaussian đều theo mọi hướng cho MSE 0.05 so với bản sạch. Chiếu đúng lên mặt phẳng ấy thì MSE còn khoảng bao nhiêu?

- A 0.005
- B 0.01
- C 0.025
- D 0.04

Khi đi thi

- Đề chấm AUC hay average precision thì nộp score liên tục; đề chấm F1 thì chọn ngưỡng trên validation.
- Chạy PCA trước: vài giây trên CPU, và autoencoder chỉ đáng giữ khi thắng nó.
- Kiểm chiều của score: đề quy ước score lớn là bất thường hay ngược lại.

Tóm tắt

- Autoencoder học nén rồi dựng lại; loss quyết định thông tin nào được giữ.
- Autoencoder tuyến tính tối ưu học đúng phép chiếu PCA, nên PCA vừa là baseline vừa là cận dưới MSE.
- Lỗi tái tạo là một điểm số: cần ngưỡng chọn trên validation và đánh giá bằng FP/FN; lỗi lớn chưa phải kết luận máy hỏng.
- Denoising dùng input nhiễu và target sạch, và chỉ được kiểm trên đúng loại nhiễu đã đo.

Kiểm tra cuối bài

CÂU 1

Một autoencoder tuyến tính một chiều dùng encoder $z = \hat{\mathbf{e}}^\top \mathbf{x}$, với $\hat{\mathbf{e}}$ là vector đơn vị cùng hướng với $(1, -2, 3)$. Với $\mathbf{x} = (-1, -4, 6)$, latent z bằng bao nhiêu?

A $25/\sqrt{14}$

B $25/14$

C $9/\sqrt{14}$

D 25

ĐỀ THI USA-NA-AIO 2026 · Vòng 1 · Part 2.2

CÂU 2

Tiếp Câu 1, decoder dựng $\hat{\mathbf{x}} = z \hat{\mathbf{e}}$. Vector sai số tái tạo $\mathbf{r} = \mathbf{x} - \hat{\mathbf{x}}$ là gì?

A $\frac{1}{14}(-39, -6, 9)$

B $\frac{1}{14}(39, 6, -9)$

C $\frac{1}{14}(25, -50, 75)$

D $\frac{1}{14}(-23, -38, 57)$

ĐỀ THI USA-NA-AIO 2026 · Vòng 1 · Part 2.3

CÂU 3

Một đội muốn nén mạnh hơn. Trên cùng một tập train đã trừ trung bình, họ đổi PCA từ k thành phần, với MSE tái tạo M_k , xuống $k - 1$ thành phần đầu, với MSE M_{k-1} . Câu nào luôn đúng?

- A $M_{k-1} \geq M_k$
- B $M_{k-1} \leq M_k$
- C $M_{k-1} = M_k$ nếu dữ liệu đã chuẩn hóa
- D Không so được nếu không biết nhãn

PHÒNG THEO CMU 10-601 · Exam 3 Practice · Q9.2(i)

CÂU 4

Trong một bộ dữ liệu, phương sai lớn nhất nằm theo hướng $\mathbf{u}$, còn nhãn chỉ phụ thuộc vào hướng $\mathbf{v} \perp \mathbf{u}$, nơi phương sai rất nhỏ. Lấy latent của một autoencoder tuyến tính một chiều đã train tốt làm feature duy nhất cho classifier. Kết quả dễ thấy nhất là gì?

- A Classifier tốt, vì latent giữ gần hết phương sai
- B Gần như đoán mò, vì latent bám theo $\mathbf{u}$
- C Classifier tốt, vì autoencoder tự giữ hướng có ích cho nhãn
- D Gần như đoán mò, vì latent một chiều thì luôn vô dụng

PHÒNG THEO CMU 10-601 · Exam 3 Practice · Q9.2(iii)

CÂU 5

Trên dữ liệu 20 cột đã trừ trung bình, một đội giữ đủ cả 20 thành phần PCA, rồi lấy lỗi tái tạo của từng mẫu làm score bất thường. Kết quả là gì?

- A Báo lỗi, vì PCA phải giữ ít chiều hơn input
- B Tách tốt nhất, vì giữ đủ 100% phương sai
- C Mẫu nào cũng được dựng đúng, nên score đều bằng 0
- D Chỉ mẫu bất thường còn có lỗi tái tạo khác 0

PHÒNG THEO CMU 10-601 · Exam 3 Practice · Q9.2(iv)

CÂU 6

Hai cảm biến: nhiệt độ đo bằng °C, độ lệch chuẩn khoảng 2, và áp suất đo bằng Pa, độ lệch chuẩn khoảng 500. Train autoencoder trên số liệu thô với MSE. Lỗi tái tạo sẽ chủ yếu phản ánh điều gì?

- A Cột áp suất, vì sai số đo theo Pa lớn hơn nhiều
- B Cột nhiệt độ, vì nó biến động tương đối nhiều hơn
- C Cả hai như nhau, vì MSE lấy trung bình theo cột
- D Không cột nào, vì autoencoder tự chuẩn hóa input

CÂU 7

Ngưỡng là phân vị 95% của score trên một tập validation gồm toàn mẫu bình thường. Sau đó nhà máy thay một loạt cảm biến mới, và phân bố dữ liệu chưa được kiểm lại. Ta kết luận được gì?

- A Ngưỡng chỉ mô tả phần đuôi score của tập validation cũ
- B Tỷ lệ báo nhầm về sau đúng bằng 5%
- C Recall với bất thường bằng 95%, vì chọn phân vị 95
- D Mẫu nào vượt ngưỡng cũng chắc chắn là bất thường

CÂU 8

Một đội viết autoencoder $\text{Linear}(10, 32) \rightarrow \text{Linear}(32, 2) \rightarrow \text{Linear}(2, 32) \rightarrow \text{Linear}(32, 10)$ nhưng quên đặt activation giữa các lớp. So với PCA 2 thành phần trên cùng tập train, MSE tái tạo tốt nhất mà mạng này đạt được là bao nhiêu?

- A Thấp hơn PCA, nhờ có lớp ẩn rộng 32 chiều
- B Không thấp hơn PCA, vì mạng vẫn tuyến tính và hạng không quá 2
- C Thấp hơn PCA, nếu train đủ lâu bằng Adam
- D Bằng 0, vì tổng số weight lớn hơn số feature

CÂU 9

Denoiser của bài được train với nhiễu Gaussian nhỏ cộng vào cả bốn cảm biến. Khi đem dùng thật, lỗi hay gặp lại là một cảm biến thỉnh thoảng bị kẹt và trả về 0.

Trước khi dùng denoiser, ta nên làm gì?

- A Dừng ngay, vì denoiser đã giảm MSE gần 4 lần
- B Train lại với nhiễu Gaussian lớn hơn nhiễu
- C Mô phỏng lỗi kẹt về 0, rồi đo lại so với baseline
- D Bỏ denoiser, vì autoencoder chỉ khử được nhiễu Gaussian

CÂU 10

Ở phần ngưỡng của bài, test có 32 mẫu bình thường và 16 mẫu bất thường, output in ra TP FP FN: 16 2 0. Precision và recall của lớp bất thường bằng bao nhiêu?

- A Precision 1.0, recall 0.89
- B Precision 0.94, recall 1.0
- C Precision 0.89, recall 0.94
- D Precision 0.89, recall 1.0

Đọc thêm

- Goodfellow, Bengio, Courville — [Deep Learning, chương 14](#): đọc nút thắt, giới hạn capacity và quan hệ với PCA.
- Vincent và cộng sự — [Stacked Denoising Autoencoders](#): xem mục tiêu noisy input → clean target và học biểu diễn.
- Baldi, Hornik (1989) — *Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima*, Neural Networks 2(1). Chứng minh autoencoder tuyến tính tối ưu học đúng không gian PCA.

- [PyTorch — Optimizing Model Parameters](#): dùng lại training loop đã học cho reconstruction loss.