Báo cáo tuần 4

1. Entropy, Cross Entropy và KL Divergence

1.1 Lượng đo thông tin (self information)

Trong lý thuyết thông tin của Shannon, ta muốn đo lượng thông tin mà một sự kiện(tập hợp các thông tin) với một biến cố E nhất định mang lại.

Công thức :

$$\mathrm{I}(E) = \log_b \frac{1}{\Pr(E)}$$

-Nguyên tắc: **sự kiện càng ít xảy ra → càng bất ngờ → chứa nhiều thông tin(I(E) tăng)**.

-Log b thường lấy cơ số 2 (bit) hoặc e (nat).

Tính chất :

1. **Không âm**

2. **Nghịch biến với Pr(E)**: xác suất càng nhỏ → lượng thông tin càng lớn và ngược lại.

3. **Tính cộng dồn cho các sự kiện độc lập**: sự kiện độc lập → thông tin cộng lại

   Nếu hai sự kiện độc lập E1,E2 thì:

$$I(E_1 \cap E_2) = I(E_1) + I(E_2)$$

Nghĩa là thông tin của hai sự kiện độc lập bằng tổng thông tin từng sự kiện.

4. **Chuẩn hóa đơn vị**: có thể chọn bit, nat, hartley.

5. **Trực giác**: nếu p=1/2 → 1 bit (như tung đồng xu công bằng).

1.2 Entropy

Entropy đo **mức bất định trung bình** của một nguồn thông tin, và chính là "lượng thông tin trung bình" mà ta nhận được từ các quan sát.

Công thức: Với biến ngẫu nhiên rời rạc X nhận các giá trị {x1,x2,..,xn} thì entropy của X là:

$$\mathrm{H}(X) = \mathbb{E}(\mathrm{I}(X)) = \sum_{i=1}^{n} \Pr(x_i)\mathrm{I}(x_i) = \sum_{i=1}^{n} \Pr(x_i) \log_b \frac{1}{\Pr(x_i)}$$

Viết lại:

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \Pr(x_i) \log_b \Pr(x_i)$$

Gọi p=(p1,p2,...,p_n) với p_i=Pr(X=x_i) là vecto xác suất của biến cố X. Khi đó entropy được viết thành:

$$\mathrm{H}(p) = -\sum_{i=1}^{n} p_i \log_b p_i$$

Tính chất của Entropy:

- Hàm Entropy là một hàm không âm (H(X) ≥ 0).
- Hàm Entropy đạt cực đại khi phân bố xác suất là phân bố đều, với H(X) ≤ log_b(n). Dấu "=" xảy ra khi và chỉ khi $p_i = \Pr(X = x_i) = 1/n$ với mọi i.

⇒ Entropy là phương pháp đo độ bất xác định khi dự đoán trạng thái của một biến ngẫu nhiên X. Entropy thông tin của biến ngẫu nhiên X càng cao (càng nhiều thông tin chứa trong thông điệp) thì càng khó dự đoán.

1.3 Cross-Entropy

Là độ đo giữa hai phân bố p (phân bố thực – *true distribution*) và q (phân bố hiện tại) để đo lượng trung bình thông tin khi dùng mã hoá thông tin của phân bố qqq thay cho mã hoá thông tin của phân bố ppp.

Công thức: Với hai phân bố xác suất rời rạc PPP và QQQ và các vector xác suất tương ứng của phân bố p=(p1,p2,…,pn), q=(q1,q2,…,qn) thì Cross-Entropy được tính như sau:

$$H(p, q) = -\sum_{i=1}^{n} p_i \log_b q_i$$

Tính chất:

- Cross-Entropy dùng q để mã hoá p luôn luôn lớn hơn hoặc bằng Entropy của p (H(p,q)≥H(p))
- Rất nhạy cảm với sự sai khác giữa p_i và q_i, càng khác nhau, giá trị Cross-Entropy càng tăng nhanh.
- Cross-Entropy không có tính đối xứng. Trong thực tế:
    - Thường ta coi p là phân bố đúng (ground truth).
    - q là phân bố mà mô hình dự đoán.
    - Ta muốn tối thiểu hóa H(p,q) để cải thiện mô hình.

Ứng dụng:
Cross-Entropy thường được sử dụng như một **hàm mất mát (loss function)** trong các mô hình học máy.

- Đặc biệt phổ biến trong phân loại nhị phân hoặc đa lớp.
- Đôi khi còn được gọi là **Logistic Loss** (trong bài toán phân loại nhị phân).

1.4 KL Divergence

KL Divergence (Kullback–Leibler Divergence), hay còn gọi là **relative entropy**, là một thước đo mức độ khác biệt giữa hai phân bố xác suất PPP (phân bố thật/chuẩn) và QQQ (phân bố xấp xỉ/mô hình).

Công thức:

$$D_{\mathrm{KL}}(p||q) = H(p,q) - H(p)$$

$$\Leftrightarrow$$

$$D_{\mathrm{KL}}(p||q) = -\sum_{i=1}^{n} p_i \log_b \frac{q_i}{p_i} = \sum_{i=1}^{n} p_i \log_b \frac{p_i}{q_i}$$

Tính chất của KL DIvergence:

- Không có tính đối xứng
- Tương tự CrossEntropy, DL cũng rất nhạy cảm với sự sai khác giữa pi và qi

Trong các bài toán Machine Learning/Deep Learning, do Entropy(P) là không đổi (P là phân phối target -> không đổi) nên tối ưu hàm KL_Divergence cũng tương đương với tối ưu hàm CrossEntropy. Nói các khác, việc bạn dùng KL_Divergence hay CrossEntropy trong Machine Learning là như nhau (trong các lĩnh vực khác thì không).

2. Bias-Variance

2.1 Bias-Variance

Trong học máy, mục tiêu của chúng ta là tìm một mô hình dự đoán f^(x) càng gần với quan hệ thực f(x) càng tốt. Tuy nhiên, bất kỳ mô hình nào cũng phải đối mặt với **sai số dự đoán**. Sai số này không chỉ đến từ dữ liệu có nhiễu mà còn do đặc tính của mô hình.

Sai số dự đoán có thể được phân rã thành:

- **Bias (Độ chệch):** Sai khác giữa giá trị dự báo của mô hình và giá trị thực (ground truth).

- **Variance (Phương sai):** Mức độ dao động của mô hình khi huấn luyện trên các tập dữ liệu khác nhau.

- **Irreducible error (sai số không thể giảm):** Nhiễu vốn có trong dữ liệu, không thể loại bỏ.

2.2 Bias-Variance Trade off

Giả sử có tập huấn luyện D gồm n điểm dữ liệu (xi,yi)

Từ tập D, ta huấn luyện một mô hình f^(x;D).Ký hiệu như vậy để nhấn mạnh rằng mô hình phụ thuộc vào tập dữ liệu huấn luyện cụ thể D.

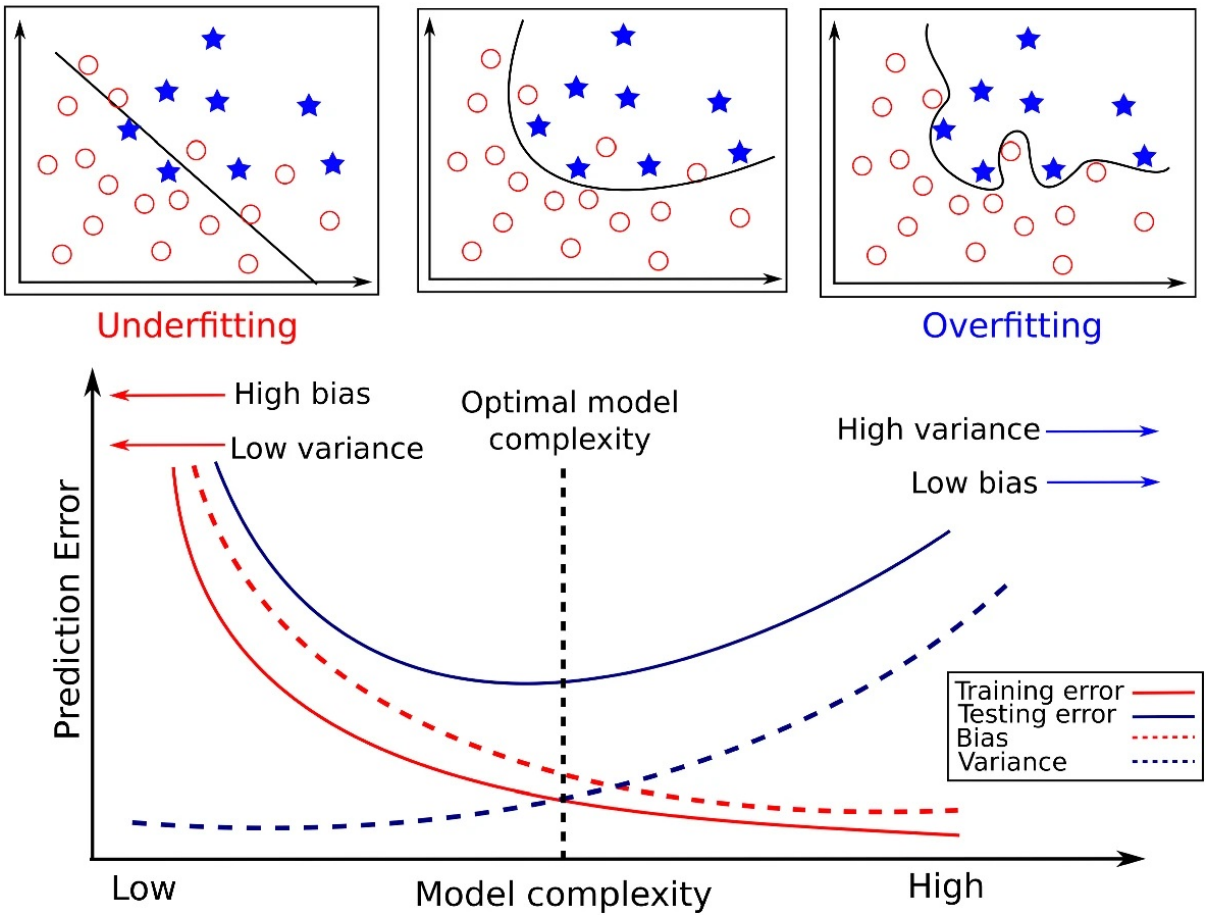
Hàm thực tế (ground truth) là f(x). Giá trị quan sát được:

$$y_i = f(x_i) + \epsilon_i$$

Tổng bình phương sai số giữa giá trị dự báo và giá trị thực tế được biểu diễn qua tổng của độ chệch (*bias*) và phương sai (*variance*) như sau:

$$\mathbf{E}[(y - \hat{f}(x;D))^2] = \underbrace{\mathbf{E}[(\hat{f}(x;D) - f(x))^2]}_{\text{bias error}} + \underbrace{\mathbf{E}[(\hat{f}(x;D) - \mathbf{E}(\hat{f}(x;D)))^2]}_{\text{variance error}} + \underbrace{\sigma_\epsilon^2}_{\text{irreducible error}}$$

Công thức này còn được gọi là công thức phân rã độ chệch-phương sai (*bias-variance decomposition*). Thành phần phương sai nhiễu có độ lớn không đáng kể nên ta có thể xem như tổng bình phương sai số chỉ phục thuộc phần lớn vào *độ chệch* và *phương sai*. Sự đánh đổi giữa *độ chệch* và *phương sai* thể hiện qua: đối với các mô hình có cùng mức độ sai số, nếu muốn một mô hình dự báo ít chệch hơn thì sẽ cần phương sai lớn hơn và ngược lại.



Bias–Variance Trade-off là nguyên lý cốt lõi giải thích tại sao mô hình ML thất bại khi triển khai thực tế.

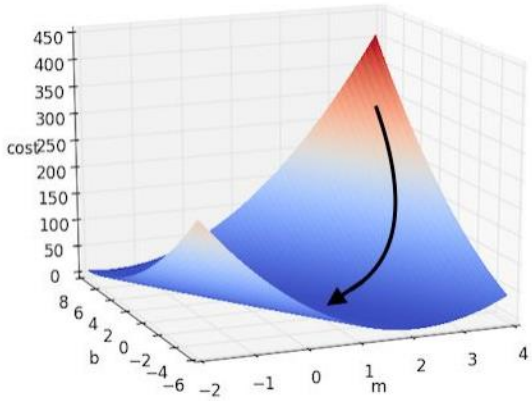
Hiểu rõ nó giúp ta thiết kế mô hình **không quá đơn giản, không quá phức tạp** – đạt điểm cân bằng tối ưu để tổng quát hóa.

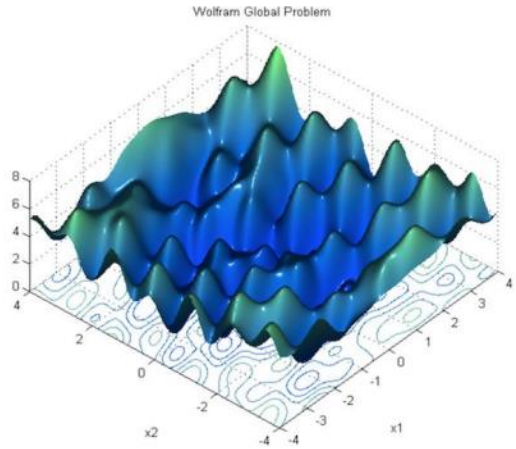
3. Optimizer

3.1 Momentum

Thuật toán Gradient Descent còn nhiều hạn chế như phụ thuộc vào nghiệm khởi tạo ban đầu và learning rate

Gradient Descent in Linear Classifier vs Neural network





Để khắc phục các hạn chế trên của thuật toán Gradient Descent người ta dùng gradient descent with momentum.

-Giới thiệu: Momentum là một kỹ thuật tối ưu mở rộng từ Gradient Descent, nhằm giải quyết được vấn đề Gradient Descent không tiến được tới điểm global minimum mà chỉ dừng lại ở local minimum

-Công thức toán học:
Gradient Descent chuẩn:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta_t)$$

Gradient Descent với Momentum:

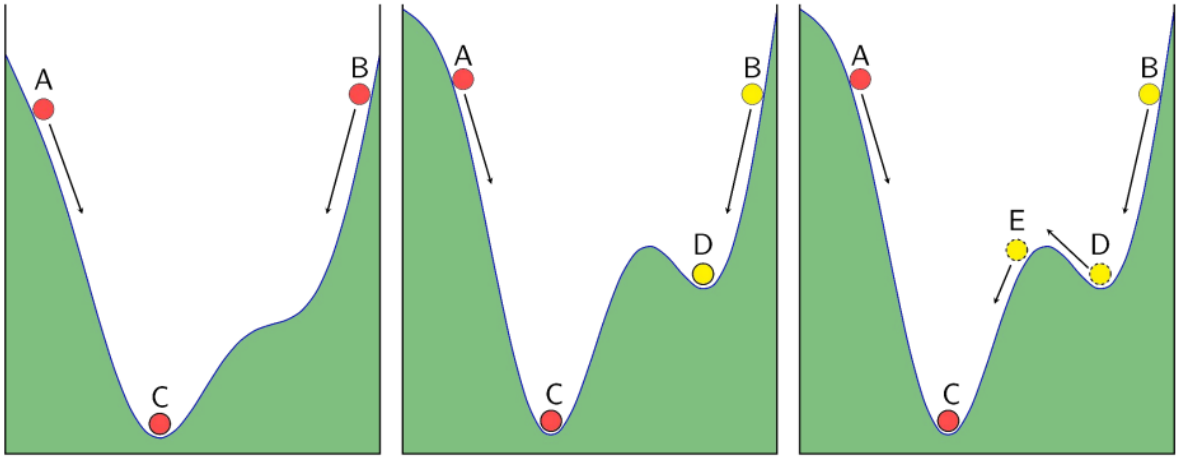
$$v_t = \beta v_{t-1} + (1 - \beta)\nabla J(\theta_t)$$

$$\theta_{t+1} = \theta_t - \eta v_t$$

Trong đó:

- θt: tham số tại bước t.
- η: learning rate.
- $\nabla J(\theta t)$: gradient tại bước t.
- vt: “tốc độ” tích lũy từ các bước trước.

- β: hệ số momentum, thường chọn 0.9.



3.2 Adagrad

-Adagrad (Adaptive Gradient Algorithm) là một thuật toán tối ưu sử dụng **learning rate thích nghi** cho từng tham số. Ý tưởng chính: tham số nào thường có gradient lớn thì sẽ giảm learning rate nhanh hơn, còn tham số ít thay đổi thì giữ learning rate cao hơn.

-Với tham số θ tại bước t:

$$g_t = \nabla J(\theta_t)$$

$$G_t = G_{t-1} + g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

Trong đó:

- gt: gradient tại bước t.
- Gt: tổng tích lũy bình phương gradient.
- η: learning rate

- ϵ: số rất nhỏ để tránh chia cho 0.

3.3 RMSprop(Root Mean Square Propagation)

-Adagrad tích lũy tổng bình phương gradient từ bước đầu → learning rate giảm liên tục → cuối cùng rất nhỏ, làm quá trình học bị chậm.

-Thay vì tích lũy toàn bộ gradient, RMSProp sử dụng trung bình động (exponential moving average) của bình phương gradient, nhờ đó learning rate không giảm quá mức → học ổn định hơn.

-Công thức toán học:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(1-\gamma)g^2{}_{t-1} + \gamma g_t + \varepsilon}} \cdot g_t$$

Trong đó:

- gt: gradient tại bước t.
- Gt: tổng tích lũy bình phương gradient.
- η: learning rate
- ϵ: số rất nhỏ để tránh chia cho 0.
- γ : hệ số decay (thường 0.9)

3.4 Adam(Adaptive Moment Estimation)

Adam là sự kết hợp của Momentum và RMSprop.

Khi học, ta không muốn **dao động quá mạnh** (Momentum giải quyết bằng cách lấy trung bình động). Ta cũng không muốn **learning rate cố định** cho tất cả tham số (RMSProp giải quyết bằng cách dùng adaptive learning rate).

=> Adam = **Momentum + RMSProp** → giúp hội tụ nhanh, ổn định, ít phải tinh chỉnh learning rate.

-Công thức toán học:

Cho một hàm mất mát $J(\theta)$, với tham số cần tối ưu θ, gradient tại bước t là:

$$g_t = \nabla_\theta J(\theta_t)$$

Bước 1: Tính momentum (ước lượng bậc 1)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Bước 2: Tính ước lượng bậc 2 (RMSProp style)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Bước 3: Hiệu chỉnh bias correction

Do m_t, v_t ban đầu có thiên lệch về 0, ta hiệu chỉnh:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Bước 4: Cập nhật tham số

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Trong đó:

- α: learning rate (thường = 0.001)
- β1≈0.9 → điều khiển momentum
- β2≈0.999 → điều khiển RMSProp
- ϵ≈10^−8→ tránh chia 0.