HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BỘ MÔN THỰC TẬP CƠ SỞ

THỰC TẬP CƠ SỞ

Giảng viên hướng dẫn : **KIM NGỌC BÁCH**
Họ và tên sinh viên : **NGUYỄN XUÂN HẢI**
Mã sinh viên : **B22DCCN271**
Lớp : **D22CQCN07-B**
Nhóm : **13**

Báo Cáo Hàng Tuần Lần 2 ngày (16-22/3/2025)

Hà Nội – 2025

BÁO CÁO TIẾN ĐỘ HÀNG TUẦN

I. TẠO BỘ DATASET

- Tìm kiếm ảnh trên mạng về động vật, chia ra làm 10 folder về bộ động vật: bướm, mèo, gà, bò, chó, voi, ngựa, cừu, nhện, sóc.

II. LINEAR CLASSIFIER

1. Linear Classifier hoạt động như thế nào?

Linear Classifier dựa trên một hàm số tuyến tính để chuyển đầu vào (ảnh đã được vector hóa) thành điểm số (**confidence scores**) cho từng nhãn. Công thức cơ bản:

$$f(x_i,W,b)=Wx_i+b$$

Trong đó:

- xi là vector đầu vào có kích thước [D×1], chứa toàn bộ pixel của ảnh.
- W là ma trận trọng số có kích thước [K×D], giúp ánh xạ từ không gian đầu vào sang không gian nhãn.
- b là vector bias có kích thước [K×1], giúp mô hình linh hoạt hơn.

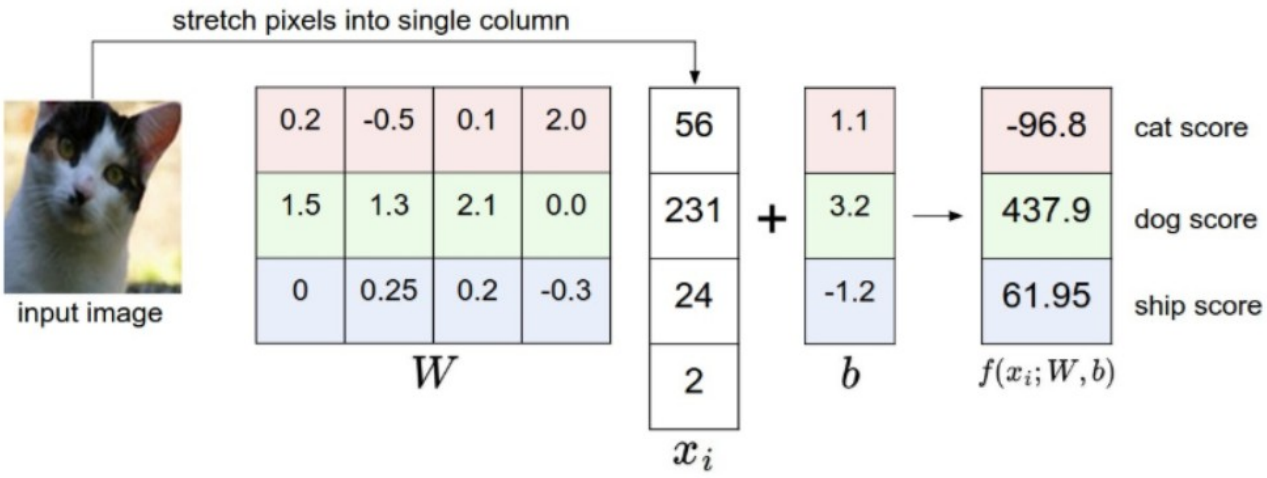
Ví dụ với **CIFAR-10**:

- **N = 50,000** (số ảnh trong tập huấn luyện)
- **D = 32 × 32 × 3 = 3072** (số chiều của mỗi ảnh)
- **K = 10** (số nhãn: máy bay, xe hơi, chim, mèo, hươu, chó, ếch, ngựa, tàu, xe tải)

=> W sẽ có kích thước [10×3072], b sẽ có kích thước [10×1].

Kết quả đầu ra là một vector có **K giá trị**, mỗi giá trị thể hiện mức độ tự tin (confidence score) của mô hình cho từng nhãn. Nhãn có giá trị cao nhất sẽ được chọn.

Ví dụ cho linear classifier



Ví dụ như ở bức hình trên, giả sử bức tranh đầu vào chỉ có kích thước 2*2 pixel, ta sẽ duỗi thẳng bức hình này ra thành vector *xi*=4*1. Lúc này, D = 4. Giả sử trong trường hợp này số nhãn phân loại chỉ là 3 nhãn (cat, dog,ship) thì K = 3. Như vậy W lúc này sẽ là ma trận K*D = 3*4, b là vector K*1 tức là b = 3*1. Sau quá trình tính toán, kết quả đầu ra sẽ là 3 giá trị tương ứng với 3 nhãn. Vì dog có **giá trị cao nhất** nên được chọn. Và bạn cũng thấy được đây là kết quả sai, vì cat mới là nhãn đúng. Nhưng không sao, đây chỉ là ví dụ cho bạn hiểu được khái niệm linear classifier là gì.

Ghép chung W và b trong toán học

Để tiện cho việc tính toán, thay vì tách riêng W và b, ta có thể viết lại phương trình:

$$\mathbf{f(x_i,W,b)=Wx_i+b}$$

Thành:

$$\mathbf{f(x_i',W')=W'x_i'}$$

Với:

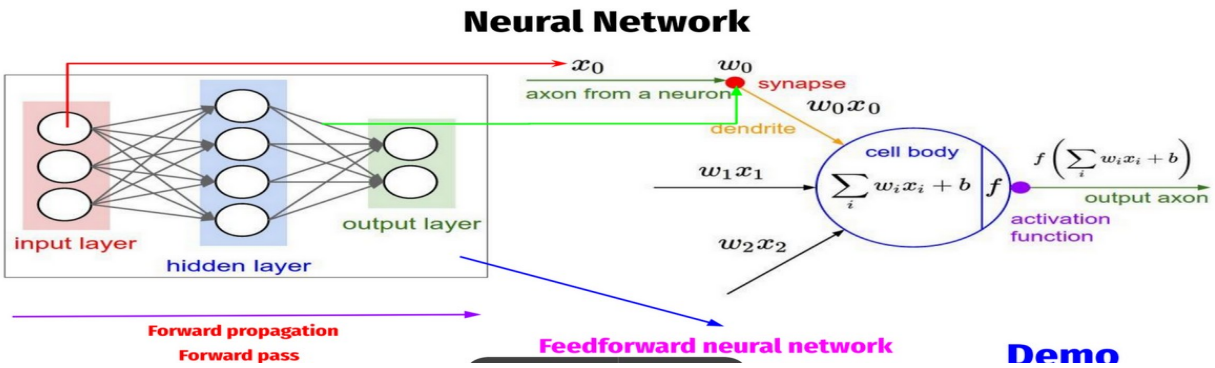
- $\mathbf{x_i'}$: Thêm một phần tử 1 vào cuối vector đầu vào xi, kích thước [D+1×1].
- $\mathbf{W'}$: Gộp W và b thành một ma trận duy nhất, kích thước [K×(D+1)].

Ví dụ:

- Ban đầu: x_i=[D×1], W=[K×D], b=[K×1].
- Sau khi gộp: x_i'=[D+1×1], W′=[K×(D+1)].

Điều này giúp đơn giản hóa tính toán ma trận khi lập trình.

Áp dụng Linear Classifier cho mạng Neural Networks



III. HÀM MẤT MÁT (LOSS FUNCTION)

Hàm **mất mát** là một hàm toán học đo lường mức độ phù hợp giữa dự đoán của mô hình với kết quả thực tế. Nó cung cấp một số liệu định lượng về độ chính xác của dự đoán của mô hình, có thể được sử dụng để hướng dẫn quá trình đào tạo của mô hình. Mục tiêu của hàm mất mát là hướng dẫn các thuật toán tối ưu hóa trong việc điều chỉnh các tham số mô hình để giảm tổn thất này theo thời gian.

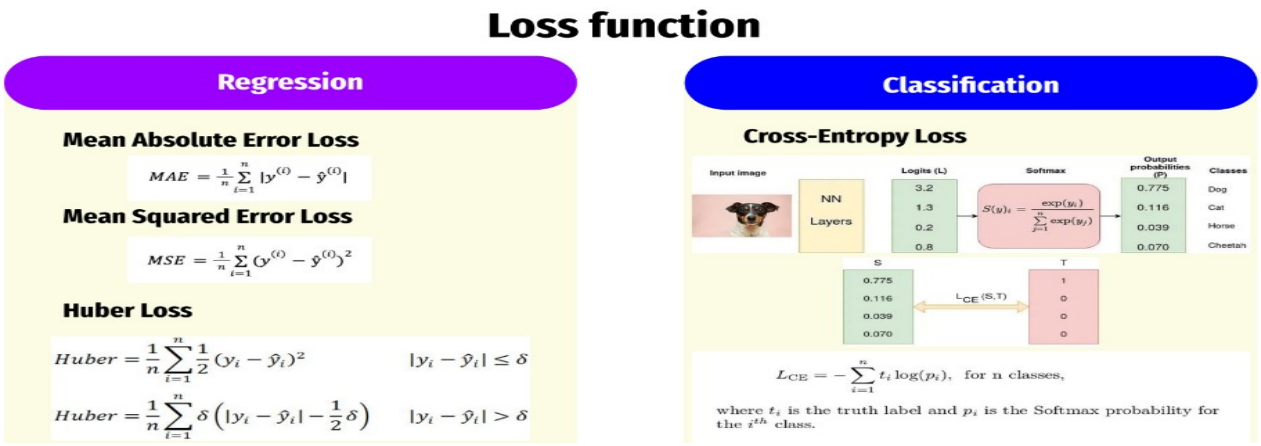
Các hàm mất mát rất quan trọng vì chúng:

1. **Hướng dẫn đào tạo mô hình:** cơ sở cho quá trình tối ưu hóa. Trong quá trình đào tạo, các thuật toán như Gradient Descent sử dụng hàm mất mát để điều chỉnh các tham số của mô hình, nhằm mục đích giảm lỗi và cải thiện dự đoán của mô hình.
2. **Đo lường hiệu suất:** Bằng cách định lượng sự khác biệt giữa giá trị dự đoán và giá trị thực tế, hàm mất mát cung cấp chuẩn mực để đánh giá hiệu suất của mô hình. Giá trị mất mát thấp hơn thường chỉ ra hiệu suất tốt hơn.
3. **Ảnh hưởng đến động lực học tập:** Việc lựa chọn hàm mất mát ảnh hưởng đến động lực học tập, bao gồm tốc độ học của mô hình và loại lỗi nào bị phạt nặng hơn. Các hàm mất mát khác nhau có thể dẫn đến các hành vi và kết quả học tập khác nhau.

Hàm mất mát hoạt động như thế nào?

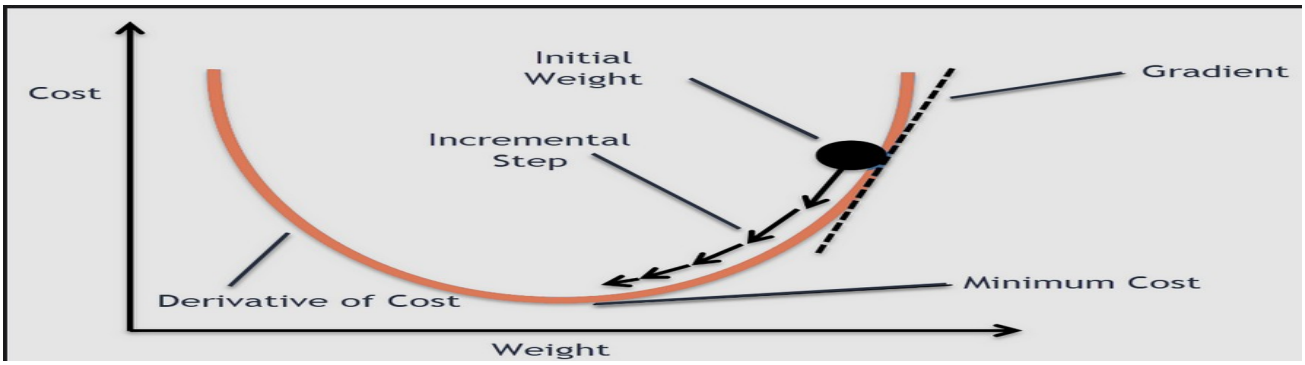
1. **Dự đoán so với Giá trị thực** :
    - Mô hình đưa ra dự đoán dựa trên các tham số hiện tại.
    - Hàm mất mát tính toán lỗi giữa giá trị dự đoán và giá trị thực tế.
2. **Đo lường lỗi** :
    - Lỗi được định lượng bằng hàm mất mát dưới dạng số thực biểu thị "chi phí" hoặc "hình phạt" cho những dự đoán không chính xác.
    - Lỗi này sau đó có thể được sử dụng để điều chỉnh các tham số của mô hình theo cách làm giảm lỗi trong các dự đoán trong tương lai.
3. **Tối ưu hóa** :
    - **Gradient Descent** : Hầu hết các mô hình sử dụng gradient descent hoặc các biến thể của nó để giảm thiểu hàm mất mát. Thuật toán tính toán gradient của hàm mất mát theo các tham số mô hình và cập nhật các tham số theo hướng ngược lại của gradient.

- **Hàm mục tiêu** : Hàm mất mát là thành phần chính của hàm mục tiêu mà các thuật toán hướng tới để giảm thiểu.



IV. GRADIENT DESCENT

Gradient Descent (GD) là thuật toán tìm tối ưu chung cho các hàm số. Ý tưởng chung của GD là điều chỉnh các tham số để lặp đi lặp lại thông qua mỗi dữ liệu huấn luyện để giảm thiểu hàm chi phí.



Ý tưởng của Gradient Descent thực hiện, tại mỗi điểm của hàm số, nó sẽ xác định độ dốc sau đó đi ngược lại với hướng của độ dốc đến khi nào độ dốc tại chỗ đó bằng 0 (cực tiểu)

Gradient Descent là một thuật toán tối ưu lặp được sử dụng trong các bài toán Machine Learning và Deep Learning với mục tiêu là tìm một tập các biến nội tại cho việc tối ưu models. Trong đó:

- Gradient: là tỷ lệ độ nghiêng của đường dốc (rate of inclination or declination of a slope). Về mặt toán học, Gradient của một hàm số là đạo hàm của hàm số đó tương ứng với mỗi biến của hàm. Đối với hàm số đơn biến, chúng ta sử dụng khái niệm Derivative thay cho Gradient.
- Descent: là từ viết tắt của descending, nghĩa là giảm dần.

Gradient Descent có nhiều dạng khác nhau như Stochastic Gradient Descent (SGD), Mini-batch SDG. Nhưng về cơ bản thì đều được thực thi như sau:

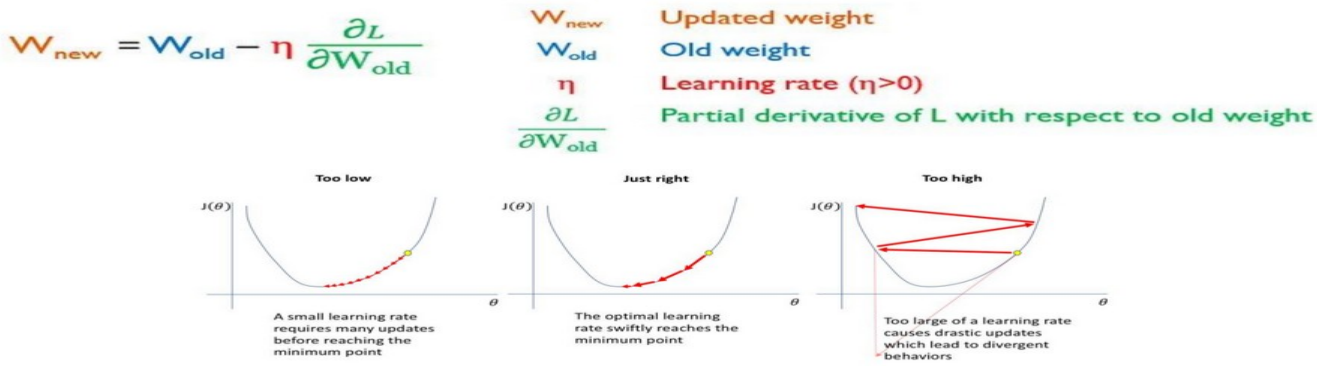
- Khởi tạo biến nội tại.
- Đánh giá model dựa vào biến nội tại và hàm mất mát (Loss function).
- Cập nhật các biến nội tại theo hướng tối ưu hàm mất mát (finding optimal points).
- Lặp lại bước 2, 3 cho tới khi thỏa điều kiện dừng.

- Công thức cập nhật cho GD có thể được viết là:

  $\theta^{(t+1)}=\theta^{(t)}-\alpha\nabla_\theta f(\theta)$

trong đó

θ là tập các biến cần cập nhật, α là tốc độ học (learning rate), $\nabla_\theta f(\theta)$ là Gradient của hàm mất mát f theo tập θ.



Batch Gradient Descent (GD):

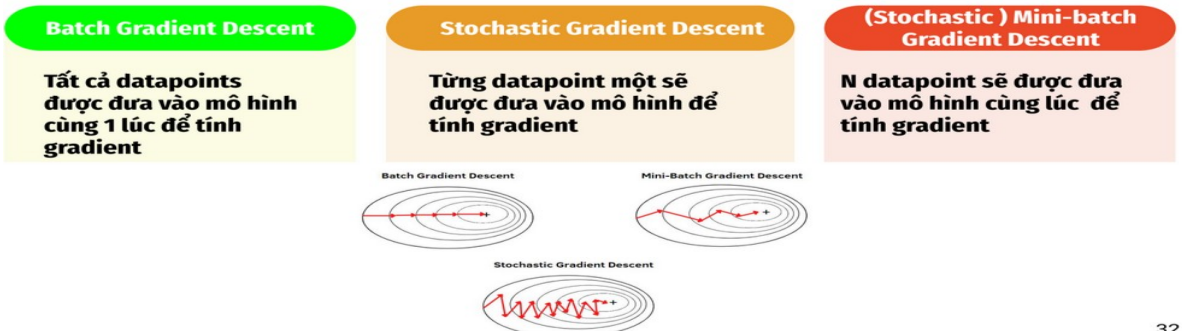
- Sử dụng toàn bộ dữ liệu để tính toán gradient cho mỗi lần cập nhật.
- Ổn định nhưng có thể chậm nếu dữ liệu quá lớn.

Stochastic Gradient Descent (SGD):

- Cập nhật dựa trên từng điểm dữ liệu (một mẫu) thay vì toàn bộ dataset.
- Tốc độ cập nhật nhanh hơn nhưng có thể dao động mạnh (không hội tụ mượt mà như GD).

Mini-batch Gradient Descent:

- Kết hợp giữa GD và SGD, cập nhật theo một nhóm nhỏ (mini-batch) thay vì toàn bộ dataset hoặc từng điểm dữ liệu.
- Cân bằng giữa tốc độ và sự ổn định trong hội tụ.



V. BATCH NORMALIZATION

Batch Normalization (**BatchNorm**) là một kỹ thuật được sử dụng để tăng tốc độ huấn luyện và ổn định quá trình tối ưu hóa trong mạng neural. Nó giúp chuẩn hóa đầu vào của mỗi lớp để giảm sự thay đổi (internal covariate shift) và làm cho mạng hội tụ nhanh hơn.

Cách hoạt động của BatchNorm

BatchNorm hoạt động bằng cách chuẩn hóa đầu ra của một lớp (thường là sau một lớp fully connected hoặc convolutional) theo công thức:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

Trong đó:

- x_i là đầu vào của một neuron tại batch hiện tại.
- μ_B là trung bình của batch hiện tại.
- σ^2_B là phương sai của batch hiện tại.
- ϵ là một giá trị rất nhỏ để tránh chia cho 0.
- x^i là đầu vào sau khi được chuẩn hóa.
- γ và β là các tham số có thể học để duy trì khả năng biểu diễn của mô hình.

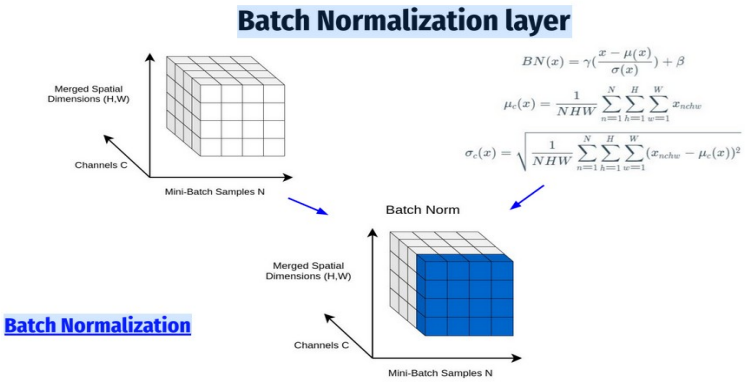
Lợi ích của Batch Normalization

1. **Tăng tốc độ huấn luyện**
    - Giúp mô hình hội tụ nhanh hơn do giảm sự thay đổi của phân phối dữ liệu giữa các lớp.
2. **Giảm sự phụ thuộc vào Learning Rate**
    - Cho phép sử dụng learning rate cao hơn mà vẫn đảm bảo tính ổn định.
3. **Giảm hiện tượng Vanishing/Exploding Gradients**
    - Do dữ liệu được chuẩn hóa, các giá trị đầu vào không bị đẩy về quá nhỏ hoặc quá lớn.
4. **Giảm sự phụ thuộc vào Khởi tạo Trọng số**
    - Giúp mô hình ổn định hơn ngay cả khi khởi tạo trọng số không tối ưu.
5. **Hoạt động như một Regularizer**
    - Có tác dụng tương tự Dropout ở một mức độ nào đó, giúp giảm overfitting.

BatchNorm được sử dụng ở đâu?

BatchNorm có thể được sử dụng:

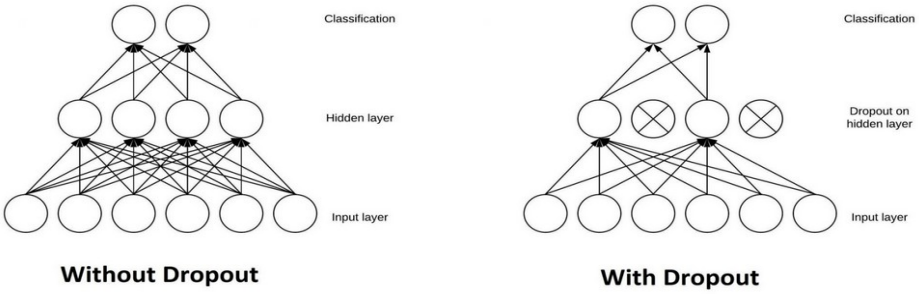
- Sau một lớp Fully Connected
- Sau một lớp CNN
- Trước hoặc sau hàm kích hoạt (ReLU, LeakyReLU, v.v.)



VI. DROP OUT

Trong neural network, việc cuối cùng là tối ưu các tham số để làm cho giảm loss function, nhưng đôi khi có unit thay đổi theo cách sửa lại lỗi của các unit khác dẫn đến việc hòa trộn làm giảm tính dự đoán của model, hay còn gọi là overfitting.

- Dropout là cách thức mà chúng ta giả định một phần các unit bị ẩn đi trong quá trình training, qua đó làm giảm tích hòa trộn (1 hidden unit không thể dựa vào 1 unit khác để sửa lỗi lầm của nó, cho thấy các hidden unit không đáng tin cậy).
- Tại mỗi step trong quá trình training, khi thực hiện Forward Propagation (Lan truyền xuôi) đến layer sử dụng Drop-Out, thay vì tính toán tất cả unit có trên layer, tại mỗi unit ta “gieo xúc xắc” xem có được tính hay không dựa trên xác suất p
- Cách thức hoạt động của dropout là để đạt được kết qủa trung bình của việc train nhiều mạng con trong network (bằng việc giả định ẩn đi % unit) thay vì chỉ lấy kết quả dựa trên việc train 1 mạng mẹ.



Tài liệu tham khảo:

https://viblo.asia/p/dropout-trong-neural-network-E375zevdlGW

https://viblo.asia/p/3-cap-do-hieu-ve-batch-normalization-bai-dich-johann-huber-Yym40mRmJ91

https://cuonglv1109.blogspot.com/2018/11/score-function.html

https://ndquy.github.io/posts/gradient-descent-2/