

HỌC MÁY CƠ BẢN – Bài 9

Mạng nơ ron nhân tạo đa lớp (MLP) -
Ứng dụng trong nhận dạng chữ viết tay

Nội dung buổi học

- Neural Network
 - Bộ phân lớp tuyến tính
 - Multiple Perceptron: Neural network
 - Huấn luyện mạng với gradient descent
- Bài toán nhận dạng chữ viết tay
 - Bộ dữ liệu MNIST
 - Định dạng dữ liệu
 - Phát biểu bài toán
 - Xây dựng mạng và thử nghiệm

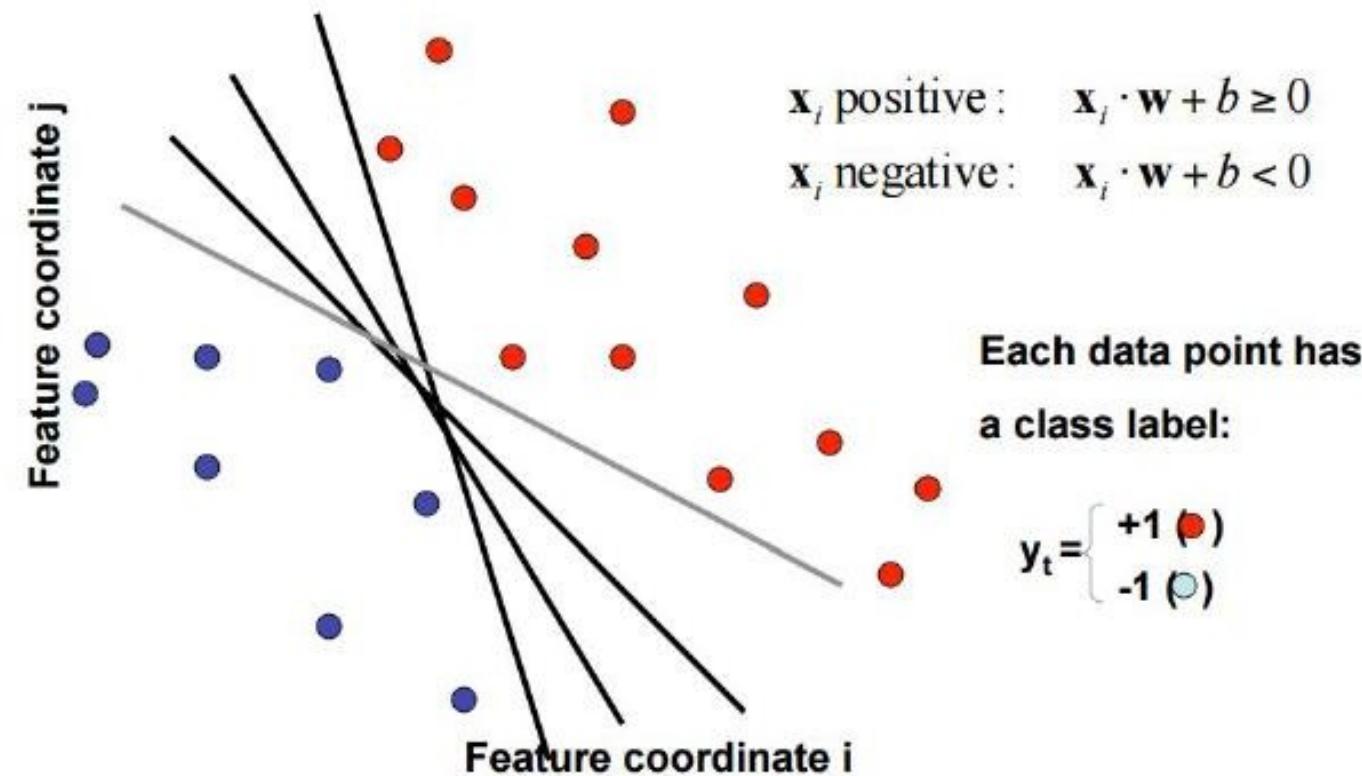




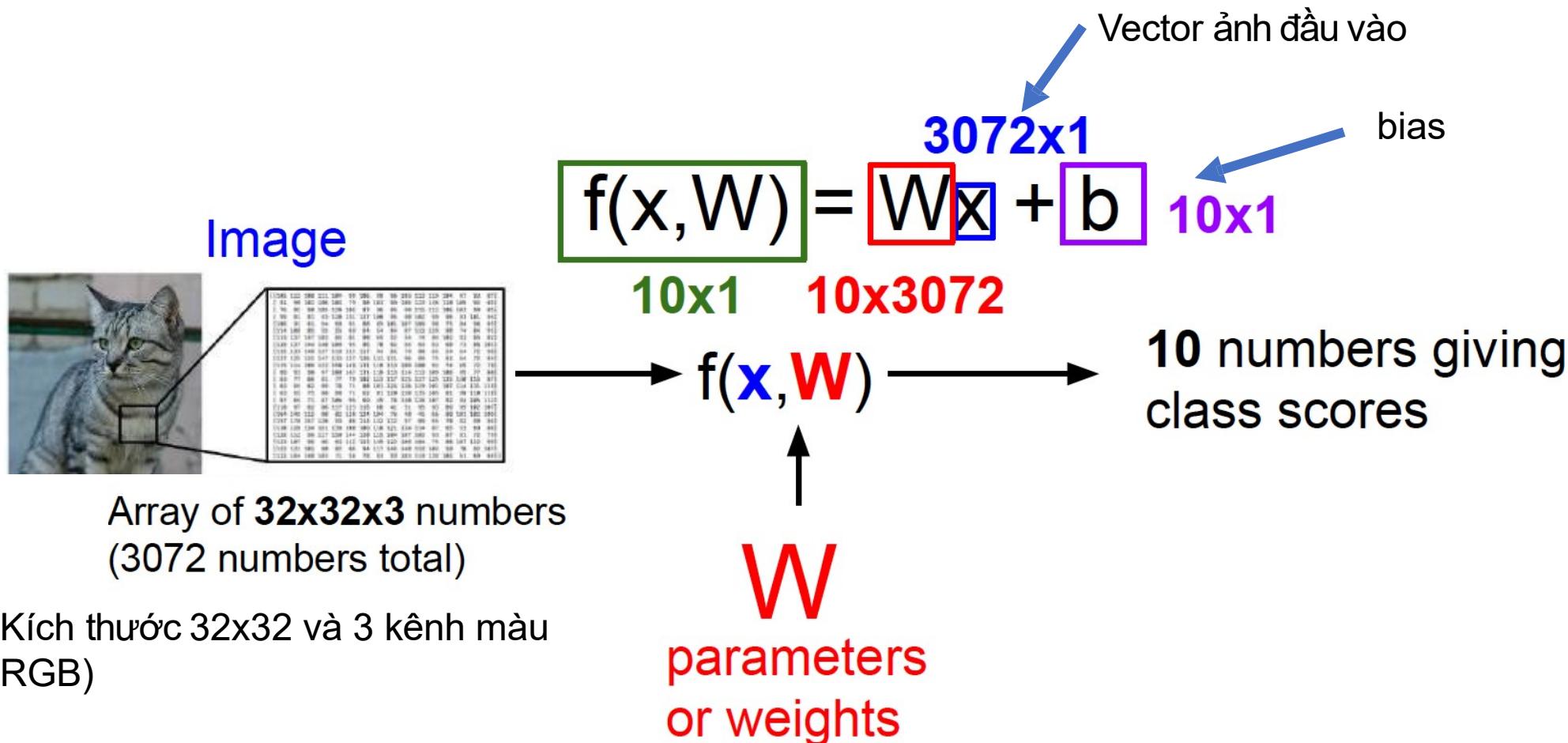
Mạng Nơ-ron đa lớp (Multi-Layer Perception Network)

Phân lớp tuyến tính (Linear Classification)

- Tìm một đường thẳng phân tách giữa hai lớp positive và negative

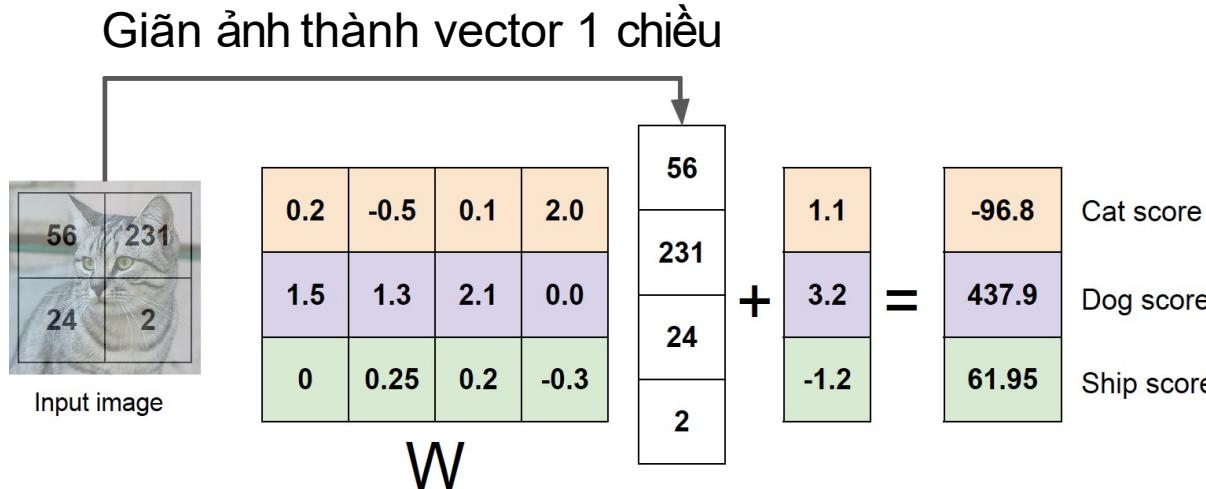


Phân lớp tuyến tính (Linear Classification)

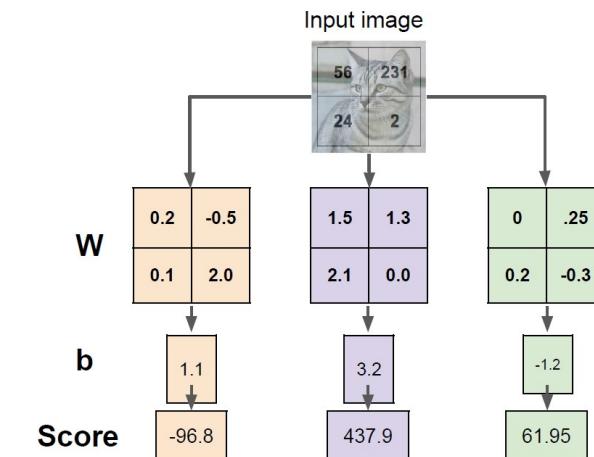


Linear Classification

- Ví dụ với phân loại ảnh có 4 pixels và 3 lớp đối tượng (cat/dog/ship)

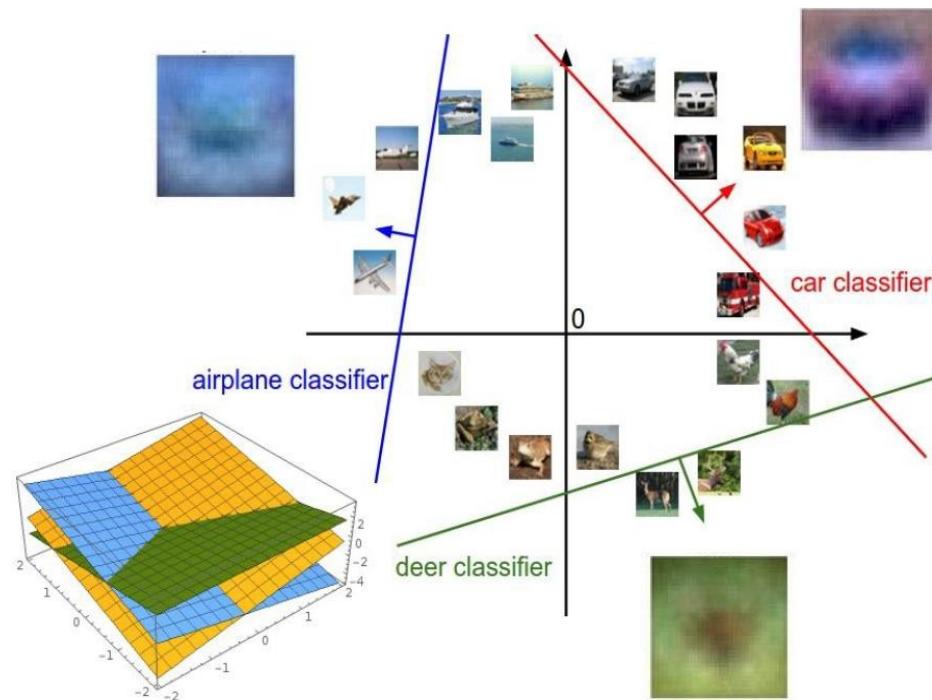


$$f(x, W) = Wx + b$$



Linear Classification

- Trực quan hóa bộ phân lớp với các đường phân tách



$$f(x, W) = Wx + b$$



Array of **32x32x3** numbers
(3072 numbers total)

Part 2: Phân lớp Perception



Mạng Neuron (Neural Network)

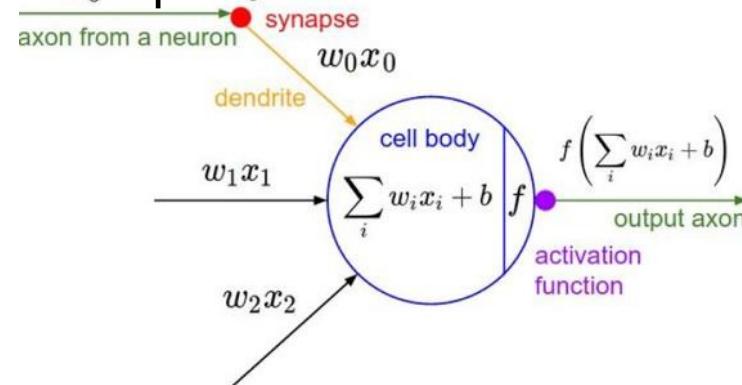
Linear

classification

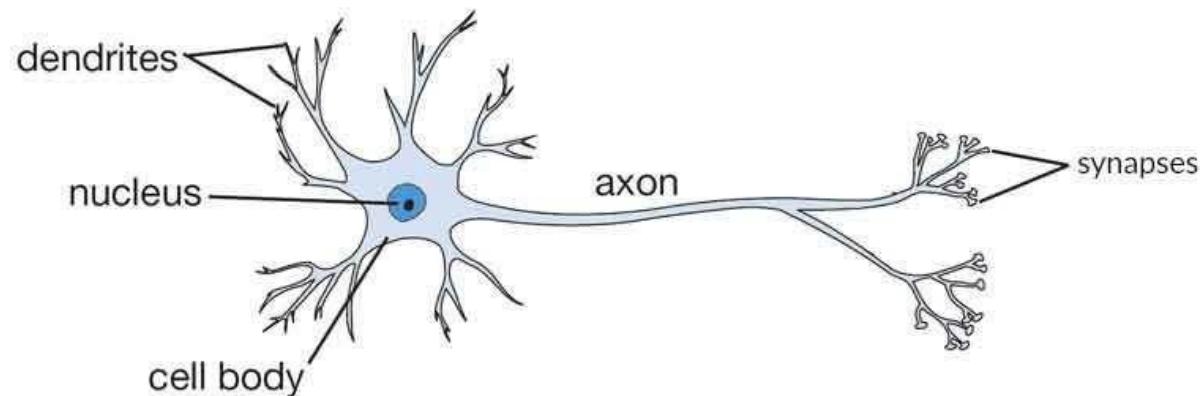
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$$\text{Output} = \text{sign}(Wx+b)$$

Perceptron

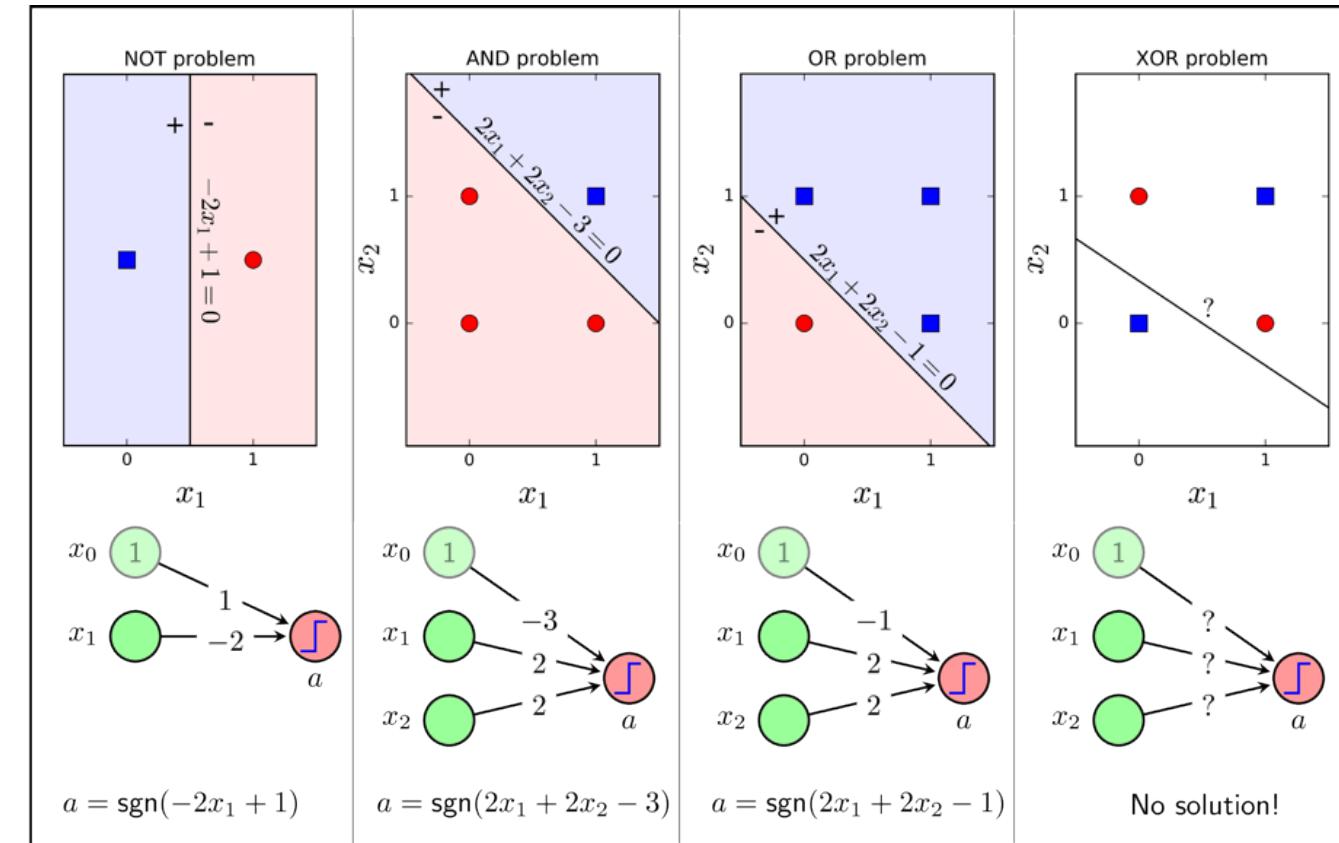


Biological Neuron



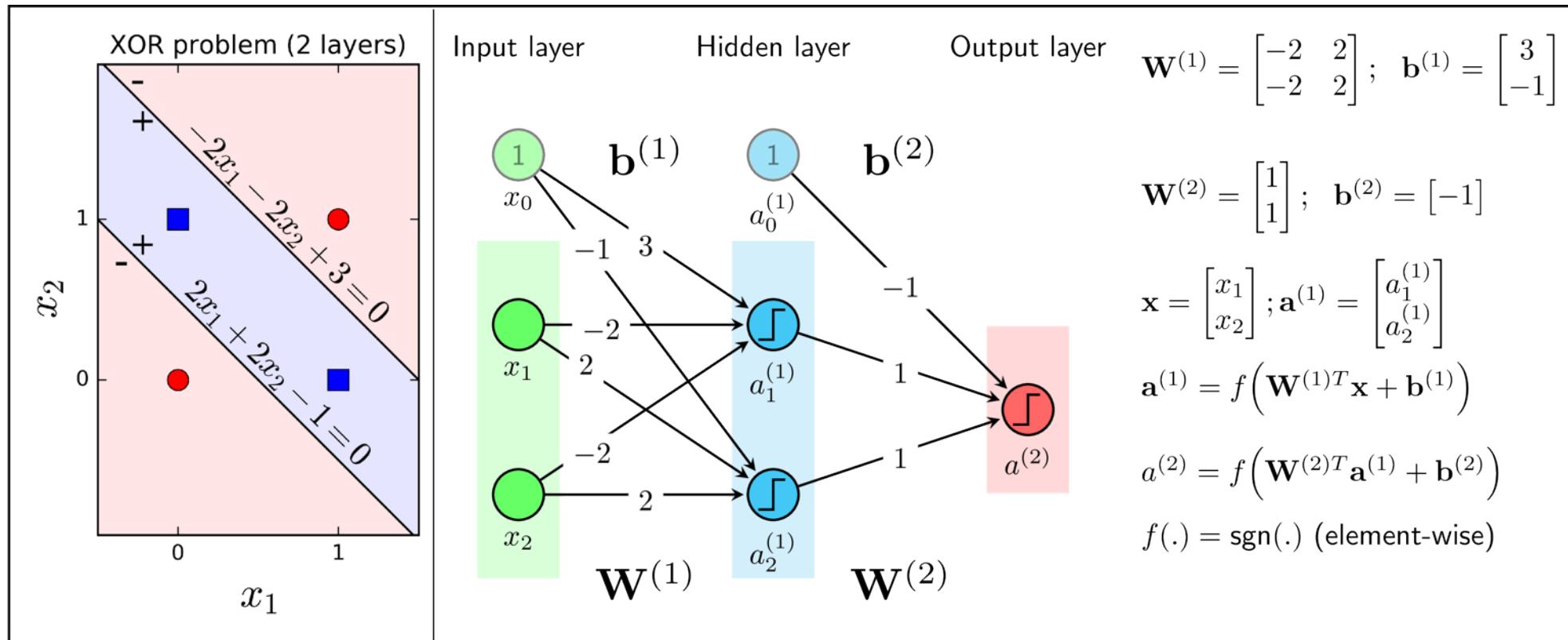
Mạng Neuron (Neural Networks)

- ❖ Vấn đề của Linear Classification
 - ❖ Một số phân bố dữ liệu có thể phân tách bằng các đường thẳng
 - ❖ Vấn đề với XOR: phân tách bằng non-linearly



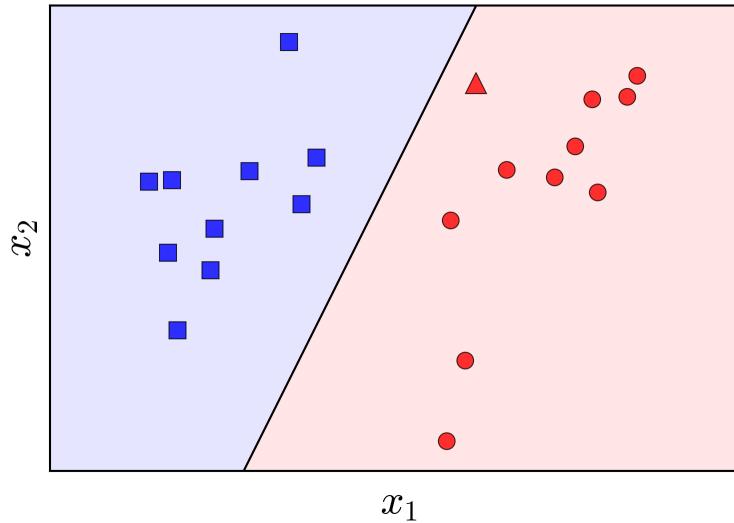
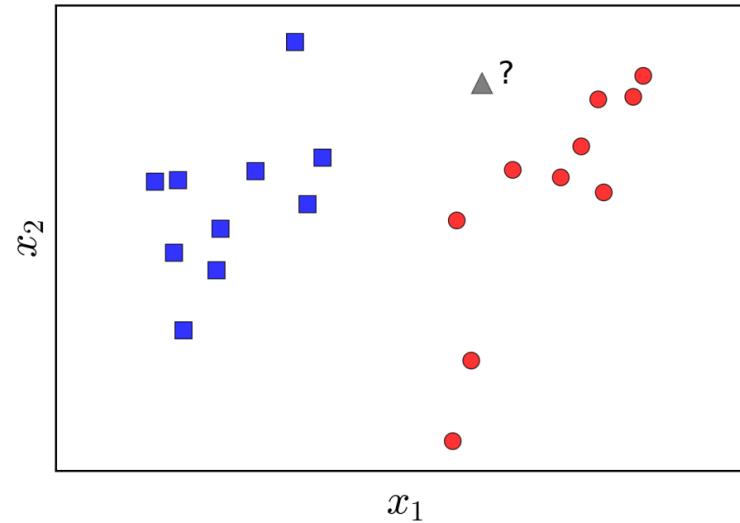
Neural Network

- Multiple perceptron



Phân lớp Perception

- Đơn giản là binary classifier ; 2-D feature vector



- Tồn tại 1 đường thẳng phân tách 2 lớp

$$f_W(x) = w_1x_1 + \dots + w_dx_d + w_0$$

- Phân lớp Perceptron đi tìm hệ số

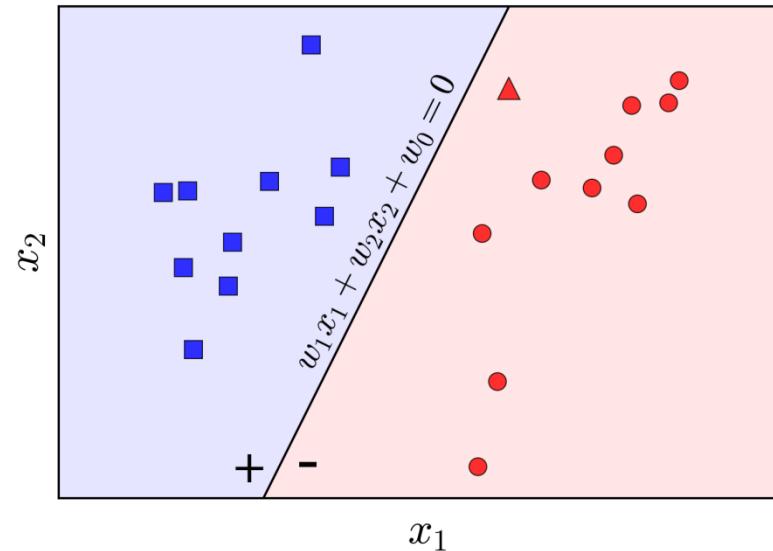
$$w = [w_0, w_1, w_2, \dots, w_N]$$

So sánh K-NN vs. Perception

- Vai trò các thành phần trong feature vector là tương đương nhau
- Khoảng cách định nghĩa tường minh (Euclidian distance)
- Sử dụng toàn bộ dữ liệu training
- Mỗi thành phần được gán 1 trọng số nhất định
- Khoảng cách không định nghĩa tường minh, mà có được thông qua quá trình học
- Có thể sử dụng một phần dữ liệu (SGD)
→ lớp bài toán Metric Learning

Phân lớp perception

- Giả sử tồn tại một phương trình đường thẳng



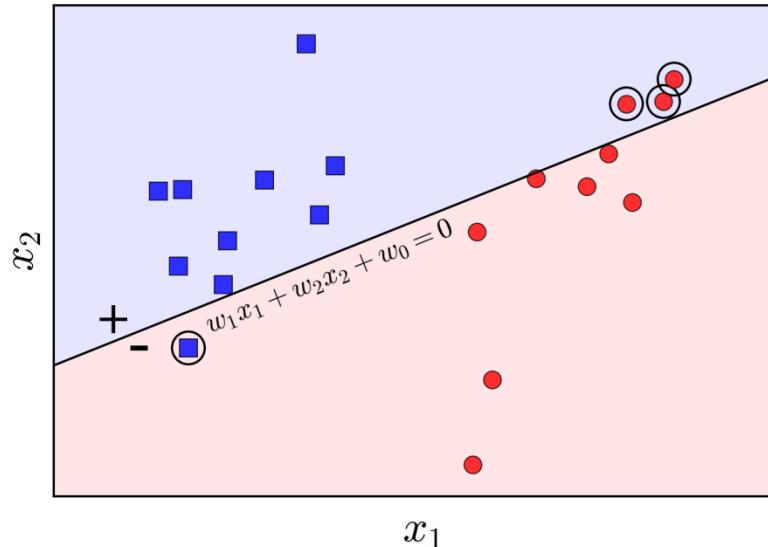
- Tìm bộ tham số w theo phương pháp GD
 - Cần tìm hàm mất mát
 - Tìm cực trị của hàm mất mát

Định nghĩa hàm mất mát

- Gán một bộ tham số bất kỳ w
- Tính (đếm) số điểm phân lớp nhầm (misclassified) với dữ liệu huấn luyện

$$J_1(w) = \sum_{i=1}^N (-y_i \text{sign}(w^T x))$$

- Sign là một hàm dấu
- Giá trị w là tốt nhất khi $J(w)$ nhỏ nhất → bài toán tối ưu
(tổng số các điểm phân lớp nhầm là ít nhất)



So sánh với hàm mất mát của hồi quy

- Hàm lỗi

$$J_1(w) = \sum_{i=1}^N (-y_i \text{sign}(w^T x))$$

- Giá trị x rời rạc
- Khó tính đạo hàm

$$J_1(w) = \sum_{i=1}^N (-y_i w^T x)$$

- Hàm lỗi

$$MSE = \frac{1}{N} \sum_{i=1}^N (y - (mx + b))^2$$

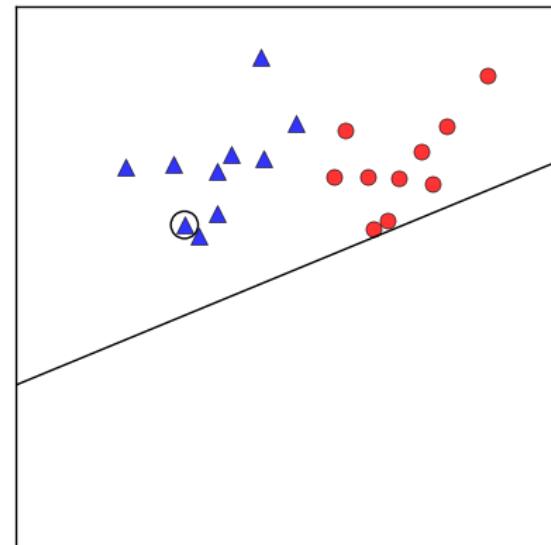
- Giá trị (y,x) liên tục
- Tính đạo hàm là khả thi

$$\nabla_w J(w, x_i, y_i) = -y_i x_i$$

$$w = w + \alpha y_i x_i$$

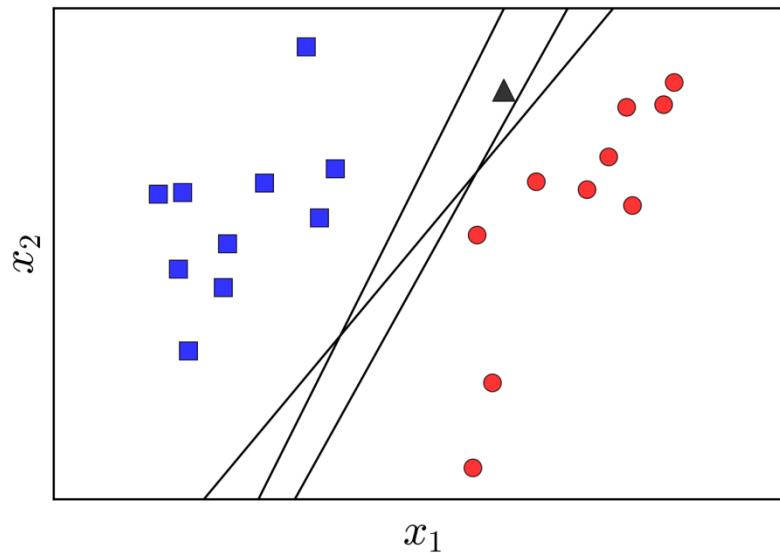
Tóm tắt thuật toán Perception

- Step 1: Chọn ngẫu nhiên vector w
- Step 2: duyệt ngẫu nhiên từng điểm x_i (như SGD)
- Step 3: nếu $-y_i \text{sign}(w^T x) = 1 \rightarrow$ phân lớp sai \rightarrow cập nhật lại $w_{\text{new}} = w + (Learning Rate) \cdot x_i \cdot y_i$
- Step 4: dừng khi ko

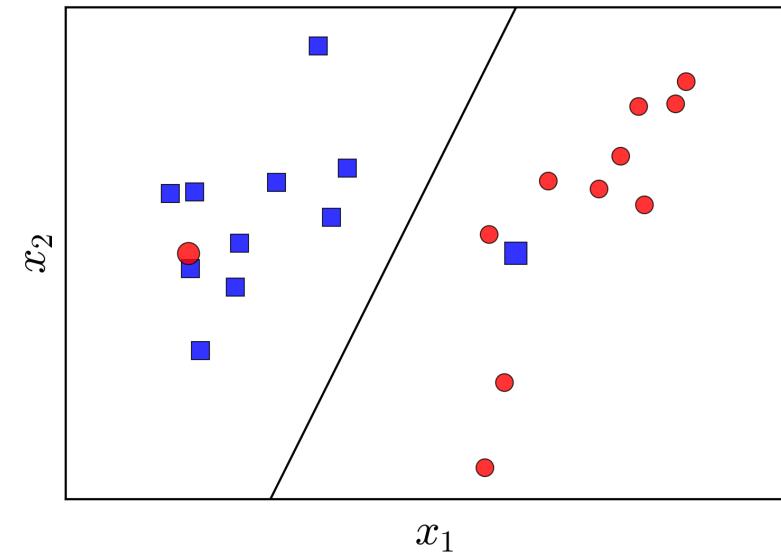


Nhận xét về thuật toán Perception

- Có thể có vô số nghiệm
- Không hội tụ khi dữ liệu không phân tách rõ ràng



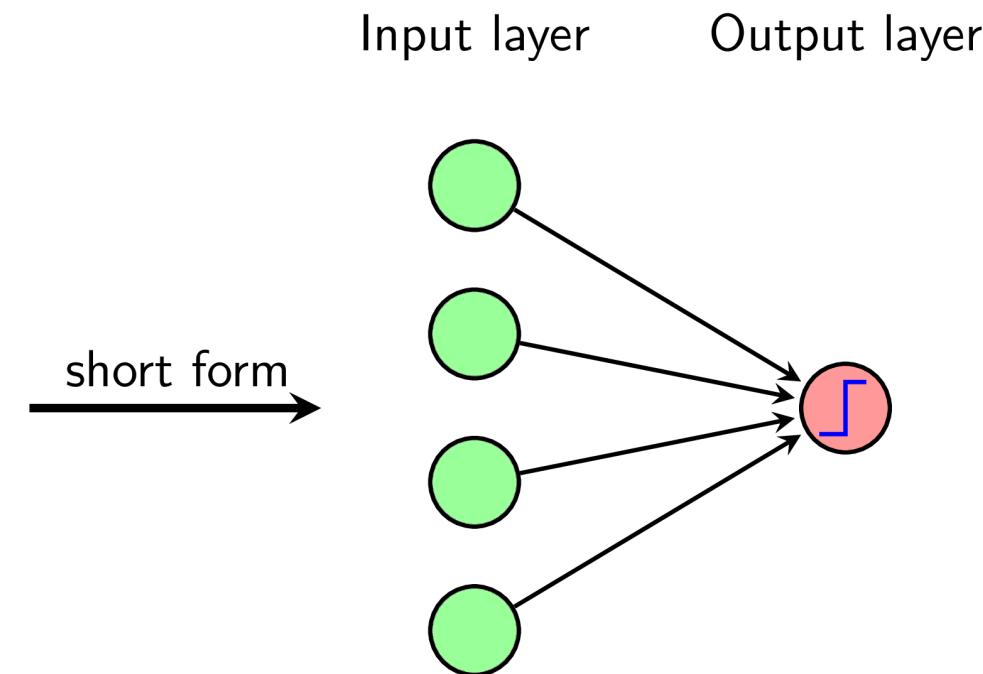
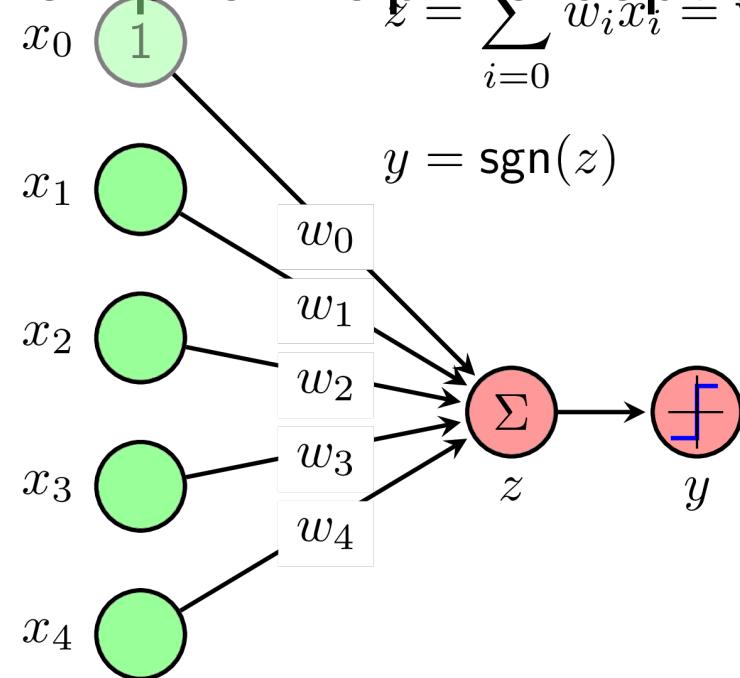
Vô số nghiệm



Dữ liệu luôn có (ít nhất)
1 điểm phân lớp sai

Mạng Neuron network 1 lớp

- Dựa trên phân lớp Perceptron



Loss Function

- Bài toán: Tập huấn luyện gồm 3 ảnh với 3 lớp đối tượng
- Sử dụng parameter W với hàm $f(x, W) = Wx$



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	12.9

- Multiple SVM loss
- Tập huấn luyện (x_i, y_i) với
 - x_i là dữ liệu vector ảnh
 - y_i là nhãn
 - Loss trên từng data i

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

- Loss tổng:

$$L = \frac{1}{N} \sum_{i=1}^N L_i$$

$$\begin{aligned} L &= (2.9 + 0 + 12.9)/3 \\ &= 5.27 \end{aligned}$$

Softmax classifier

- Chuẩn hóa từ score sang miền xác suất



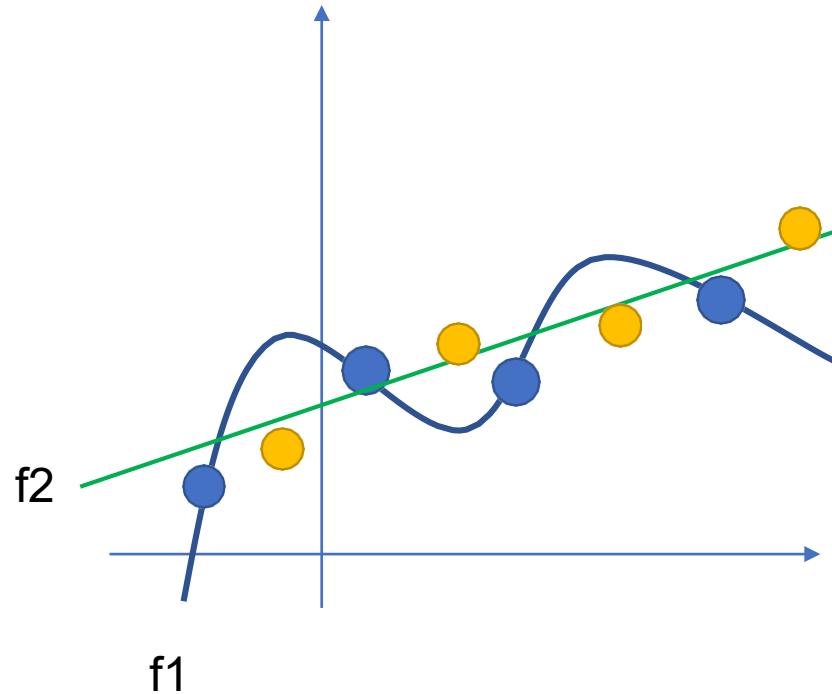
$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

	Score		Xác suất đoán (P)	So sánh ↓	Cross Entropy $H(P, Q)$	Xác suất đúng (Q)
cat	3.2		0.13			1.00
car	5.1	→	0.87			0.00
frog	-1.7		0.00			0.00

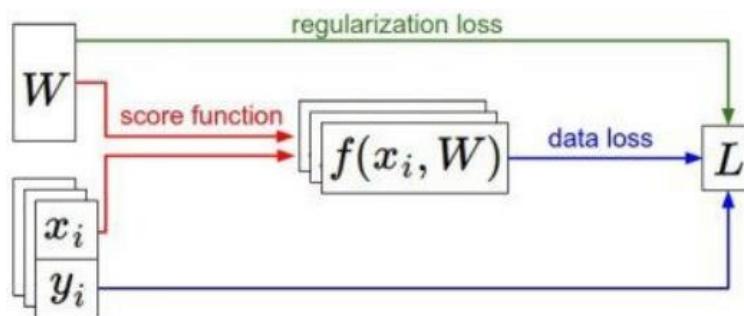
Regularization



$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$$

Data loss:
Dự đoán từ mô hình

Regularization:
Nhằm tránh mô hình biểu diễn quá tốt trên dữ liệu huấn luyện



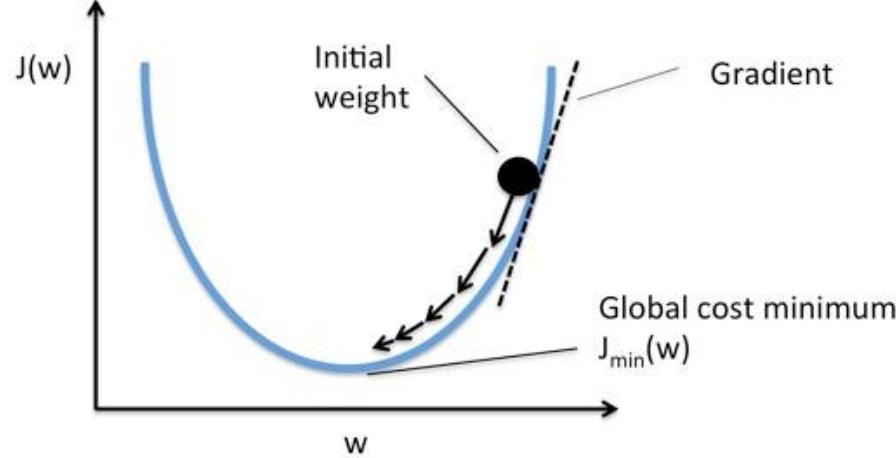
Tham khảo

Regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Gradient Descent



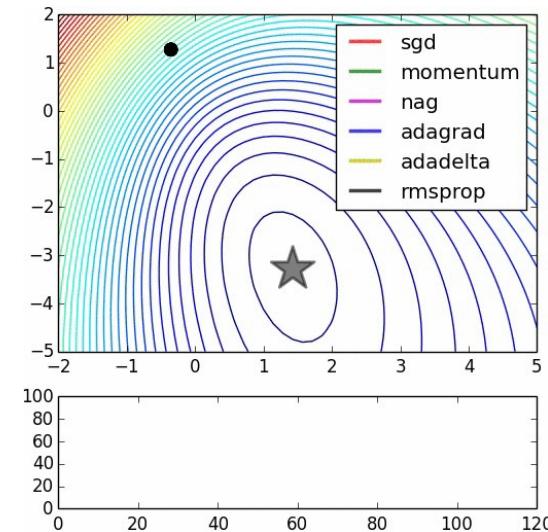
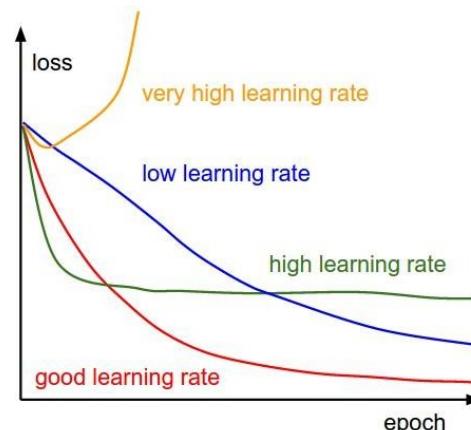
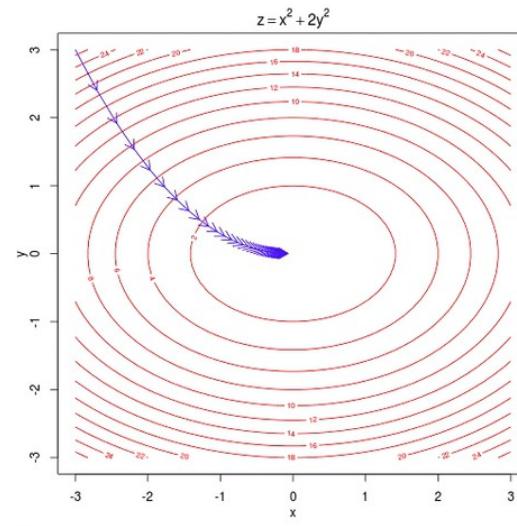
Algorithm: Gradient Descent

Input: Y, Θ, X, α , tolerance, max iterations
Output: Θ

```

1 for i = 0; i < max iterations; i ++ do
2   current cost = Cost(Y, X, Θ)
3   if current cost < tolerance then
4     break
5   else
6     gradient = Gradient(Y, X, Θ)
7     θj ← θj - α · gradient
  
```

Learning rate : bước nhảy dài hay ngắn

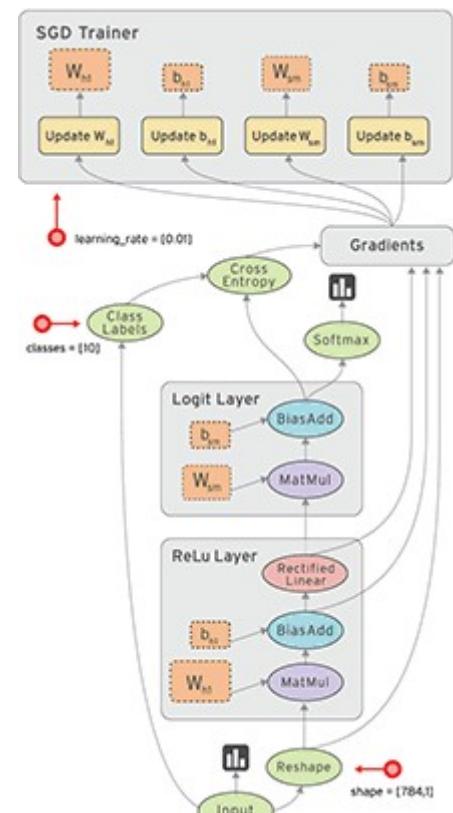
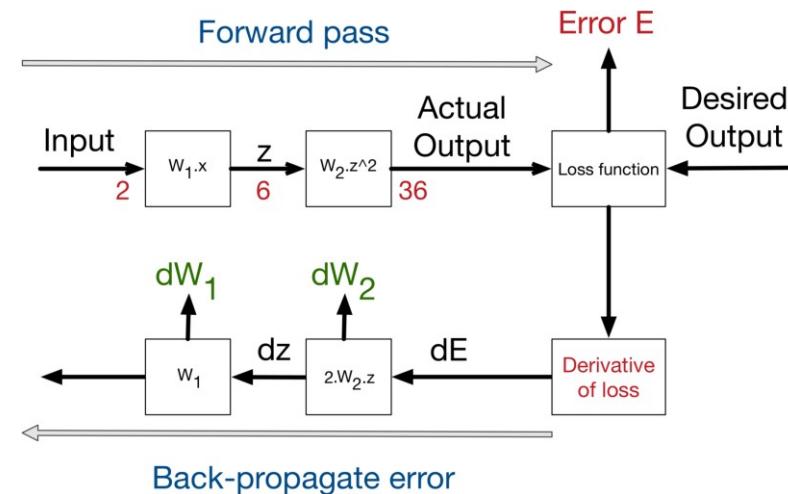
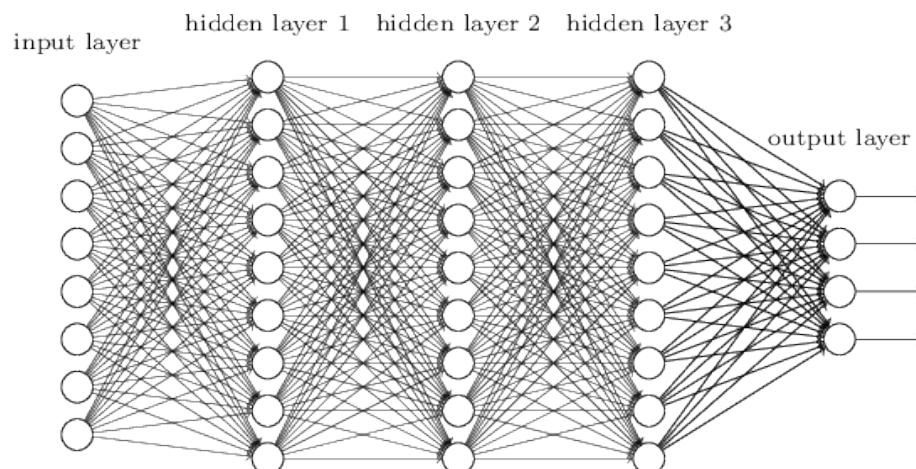


- Batch SGD: thay vì input từng ảnh, input một tập ảnh
- Minibatch: dùng batch nhỏ để huấn luyện

Lan truyền ngược (Back propagation)

- Tính gradient cho từng tham số w_i với mạng nhiều lớp

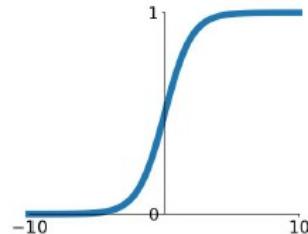
(Có thể coi là hộp đen nếu bạn chưa muộn tìm hiểu ngay)



Hàm kích hoạt (Activate functions)

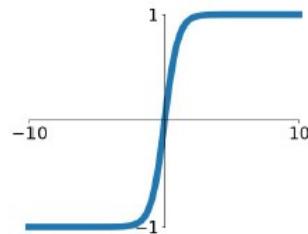
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



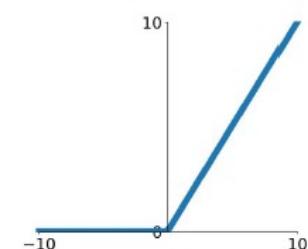
tanh

$$\tanh(x)$$



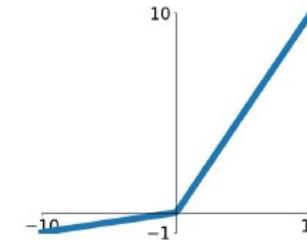
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

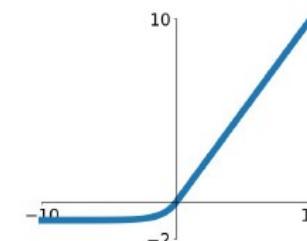


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

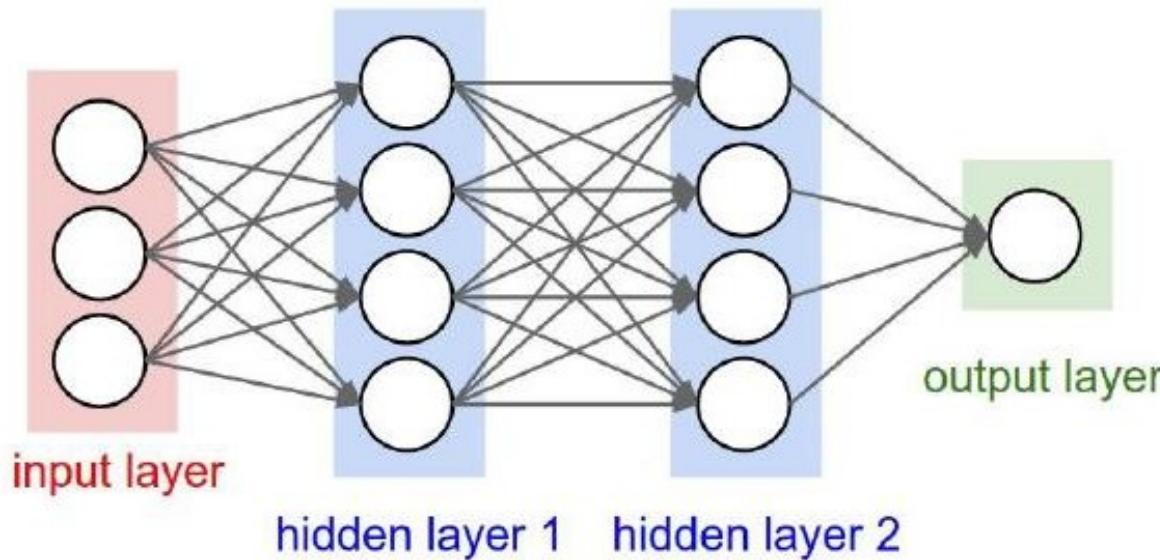
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Thường sử dụng thực tế

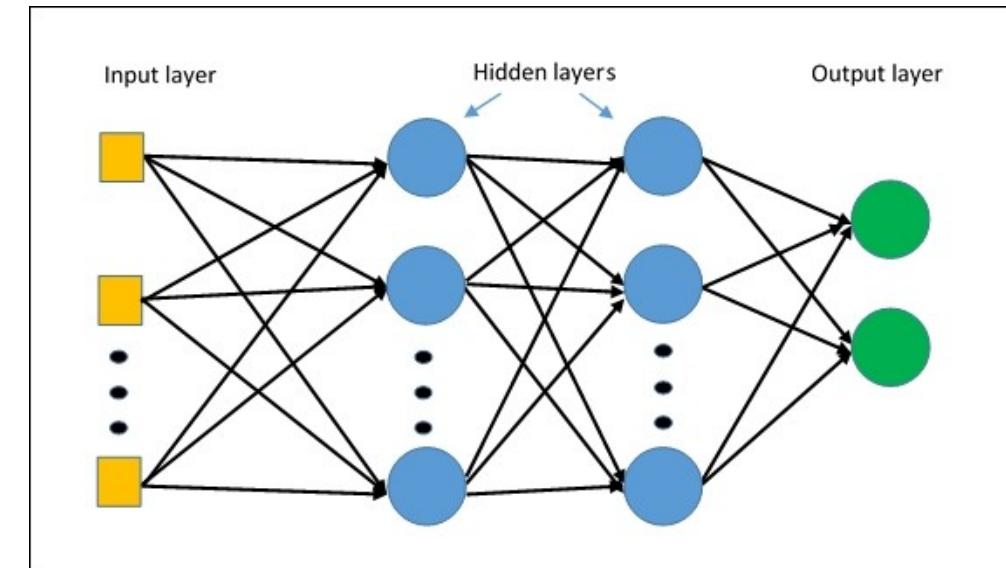
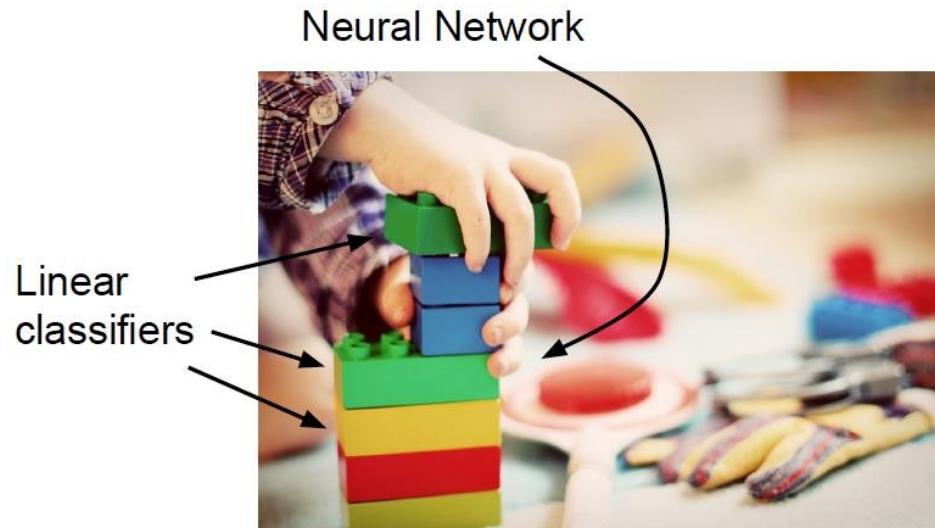
Định nghĩa mạng



```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Neural Network

- Multi-layers Perceptron
 - Layers: Chơi xếp hình với các kiểu kết nối
 - Cấu trúc mạng: Thiết kế các lớp và các kết nối

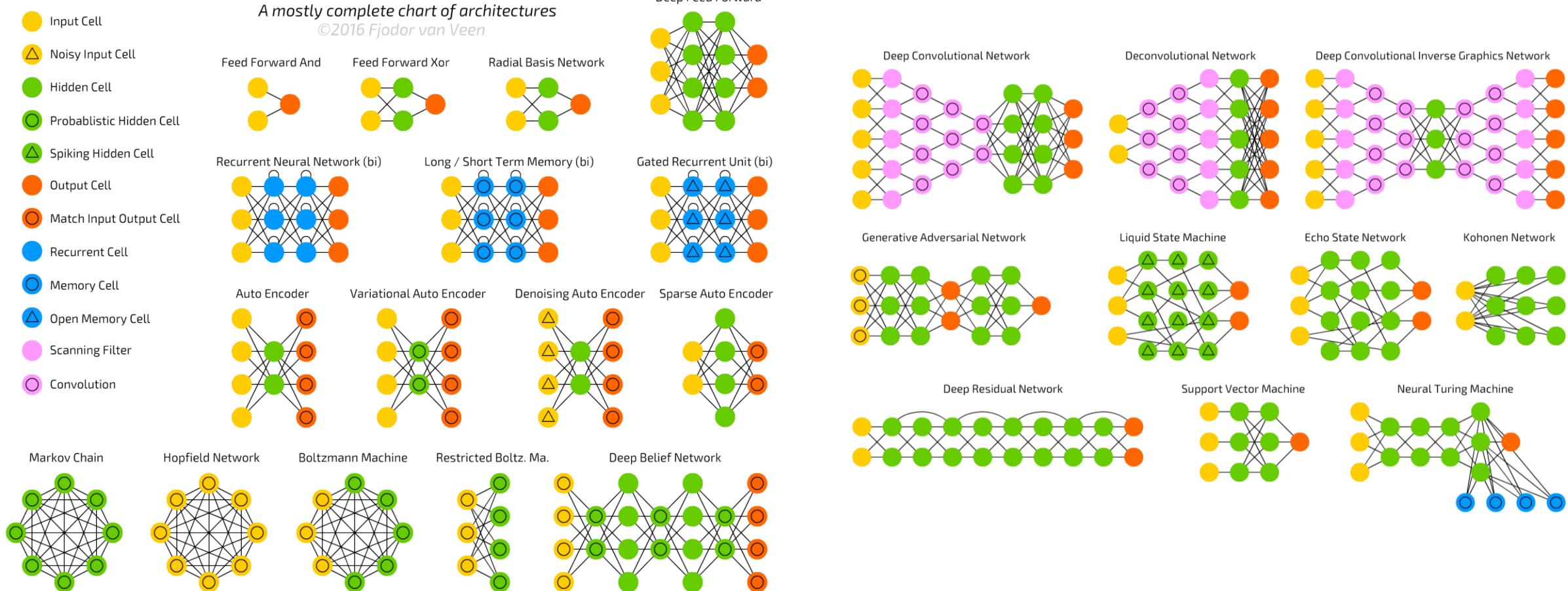


Các kiến trúc kết nối

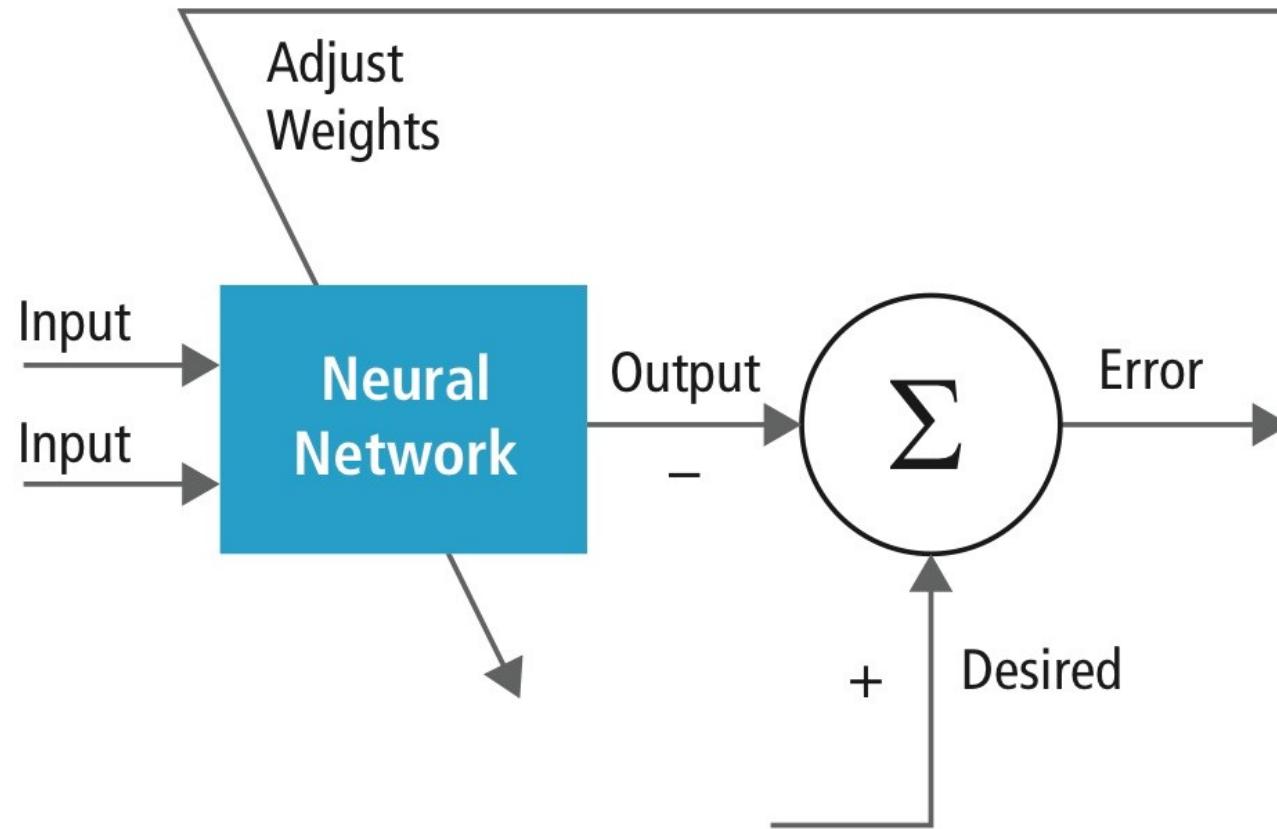
- Backfed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Open Memory Cell
- Scanning Filter
- Convolution

Neural Networks

*A mostly complete chart of architectures
©2016 Fjodor van Veen*



Huấn luyện mạng



Huấn luyện mạng

- Định nghĩa hàm mất mát (**loss function**): định lượng về độ lệch của dự đoán đối với nhãn thực của tập dữ liệu huấn luyện
- Nhiệm vụ: tìm bộ tham số sao cho loss function là nhỏ nhất có thể (tối ưu hóa – **optimization**)



airplane	-3.45	-0.51	3.42
automobile	-8.87	6.04	4.64
bird	0.09	5.31	2.65
cat	2.9	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	-4.34
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

Cat image by Nikita is licensed under CC-BY 2.0; Car image is CC0 1.0 public domain; Frog image is in the public domain

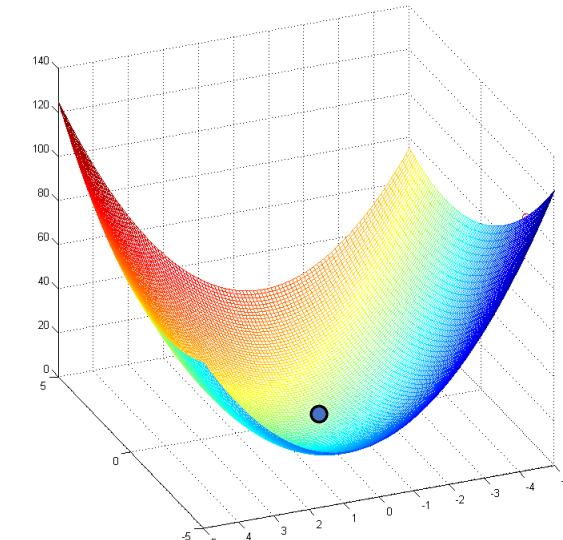
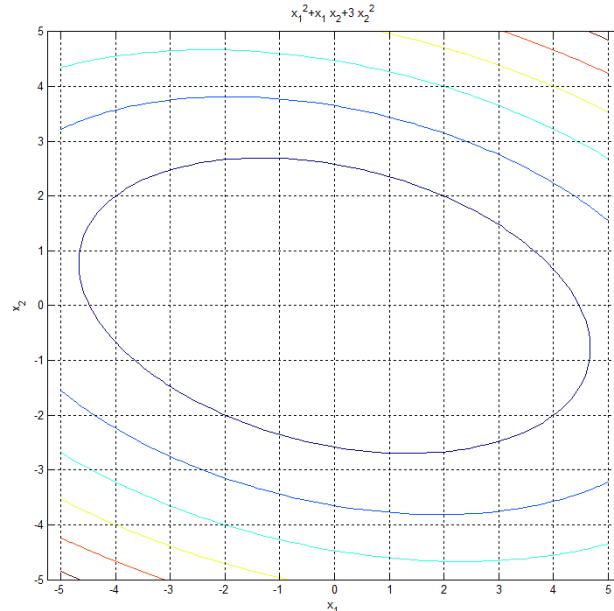
Hướng giải chung của bài toán tối ưu

- $\arg \min(y - f(x_i)) \Leftrightarrow \arg \max(y - f(x_i)) \Leftrightarrow$ Cực trị toàn cục (global)
- Hướng giải chung:
 - Tìm các cực trị địa phương (local), rồi thử từng cực trị địa phương cho đến khi tìm được giá trị global
 - Tương đương với việc tìm nghiệm của **phương trình đạo hàm (riêng nếu là hàm nhiều biến)**
- Một số phương pháp tìm cực trị
 - Gradient Descent (first order)
 - Newton-Raphon Iteration (second order)
- Gradient Descent được sử dụng nhiều/phổ biến trong deep learning vì
 - Hỗ trợ Online learning, và dữ liệu training lớn

Phương pháp suy giảm đạo hàm

$$f(x_1, x_2) = x_1^2 + x_1 x_2 + 3x_2^2$$

- Ví dụ:



Cực trị địa phương = Cực trị toàn cục

- Đạo hàm riêng:

$$\frac{df}{dx_1} = 2x_1 + x_2, \frac{df}{dx_2} = x_1 + 6x_2$$

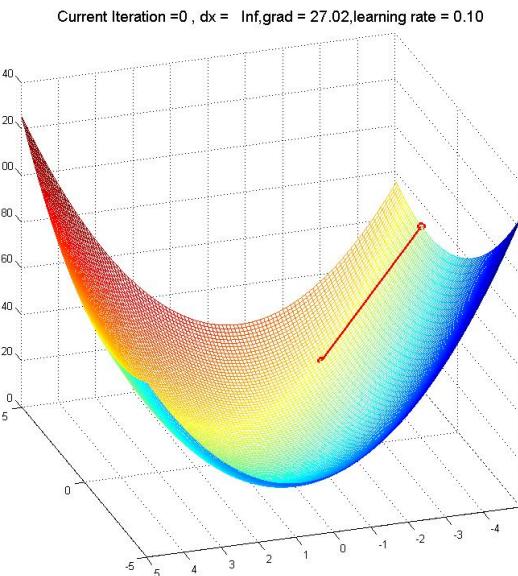
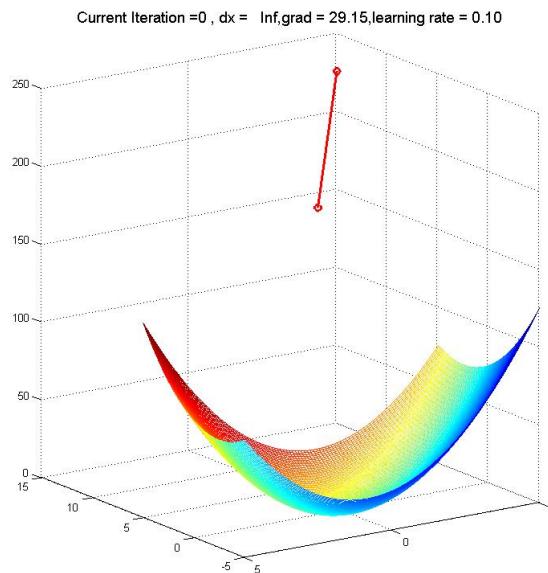
Phương pháp suy giảm đạo hàm

- Ý tưởng
 - Step 1: Lấy 1 điểm ngẫu nhiên trên $f(x)$
 - Step 2: Dịch chuyển tới điểm tiếp theo ngược hướng đạo hàm
 - Step 3: Tính lại đạo hàm ở điểm mới
 - Step 4: Lặp lại bước 2 cho đến khi một trong số các điều kiện sau thỏa mãn
 - Số vòng lặp đến ngưỡng xác định trước
 - Điểm dịch chuyển không đáng kể
 - Giá trị đạo hàm gần sát về 0
- Cách tính điểm dịch chuyển mới (ở Step 2)
$$x_{new} = x - \alpha \text{Grad}(f(x))$$

$\text{Grad}(f(x))$: Đạo hàm tại điểm x ;
 α : Hệ số học – Learning rate \leftarrow tốc độ cập nhật x

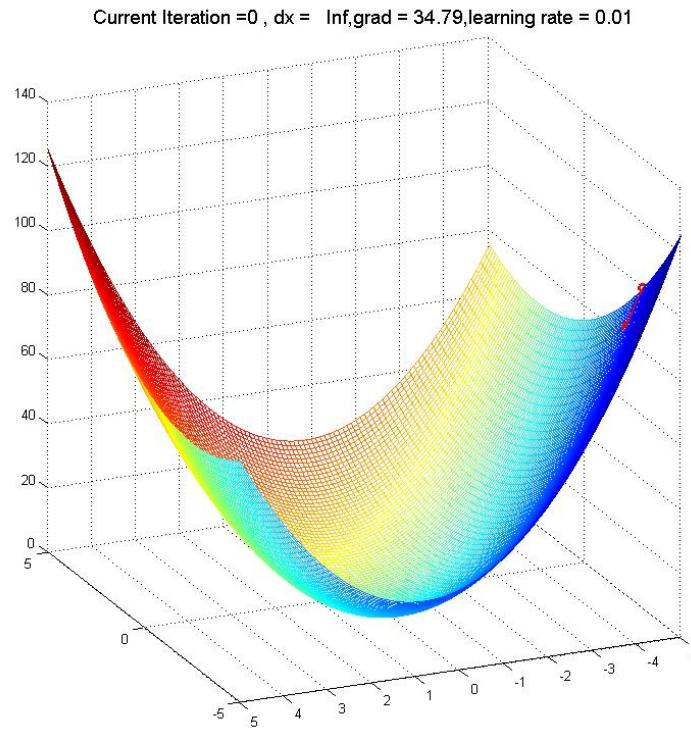
Điểm khởi tạo

- Vòng lặp đầu tiên, x được khởi tạo ngẫu nhiên
 - Có thể dùng một số thông tin biết trước ← tốc độ hội tụ nhanh hơn (giống k-means)
- Tốc độ hội tụ phụ thuộc vào: Điểm khởi tạo

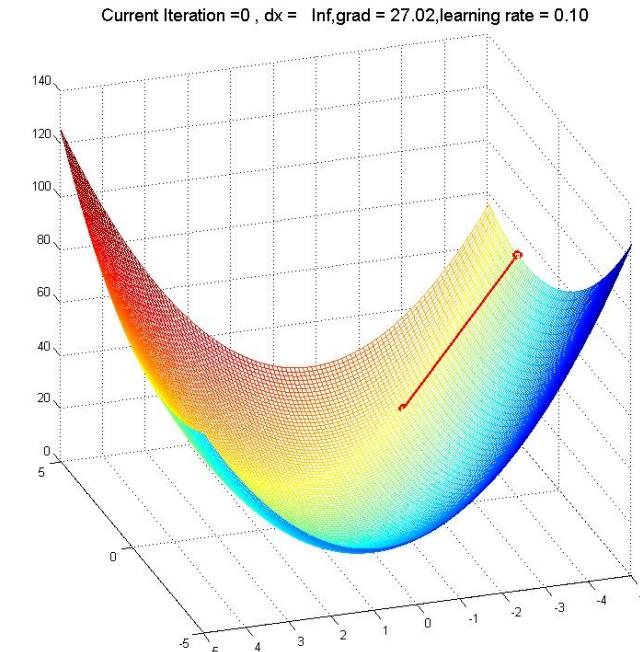


Hệ số học (Learning Rate)

- Learning rate << → số vòng lặp >>;



Learning rate = 0.01



Learning rate = 0.1

Cách tính đạo hàm Grad

$f(x)$

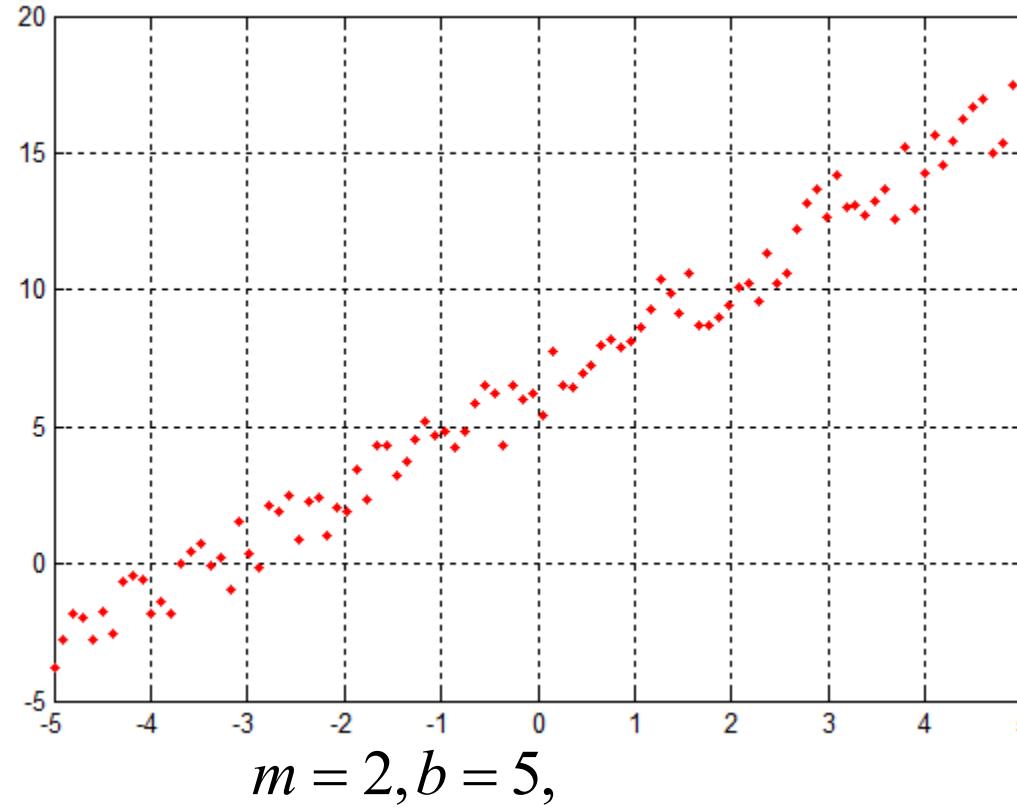
- Hàm $f(x)$ **tường minh** (ví dụ Hồi quy) \rightarrow đạo hàm Grad
(riêng) với hàm nhiều biến là khả thi
- Hàm $f(x)$ **định nghĩa gián tiếp** (ví dụ K-NN, K-Means) \rightarrow đạo hàm Grad (riêng) với hàm nhiều biến tính trực tiếp

Sử dụng GD for Linear Regression

- Cho phương trình đường thẳng

$$\begin{aligned} y &= mx + b \\ m &= 2, b = 5 \end{aligned} \quad \left. \right\}$$

- Thêm 1 lượng noise
- Bài toán: Dự đoán lại bộ tham số (m, b) theo phương pháp GD



$$Noise = 3rand(0,1)$$

Sử dụng GD for Linear Regression

- Phát biểu dưới dạng bài toán tối ưu hàm lỗi

$$\arg \min_{m^*, b^*} (y - (mx + b))^2 \quad (m^*, b^*) \text{ là bộ tham số tối ưu}$$

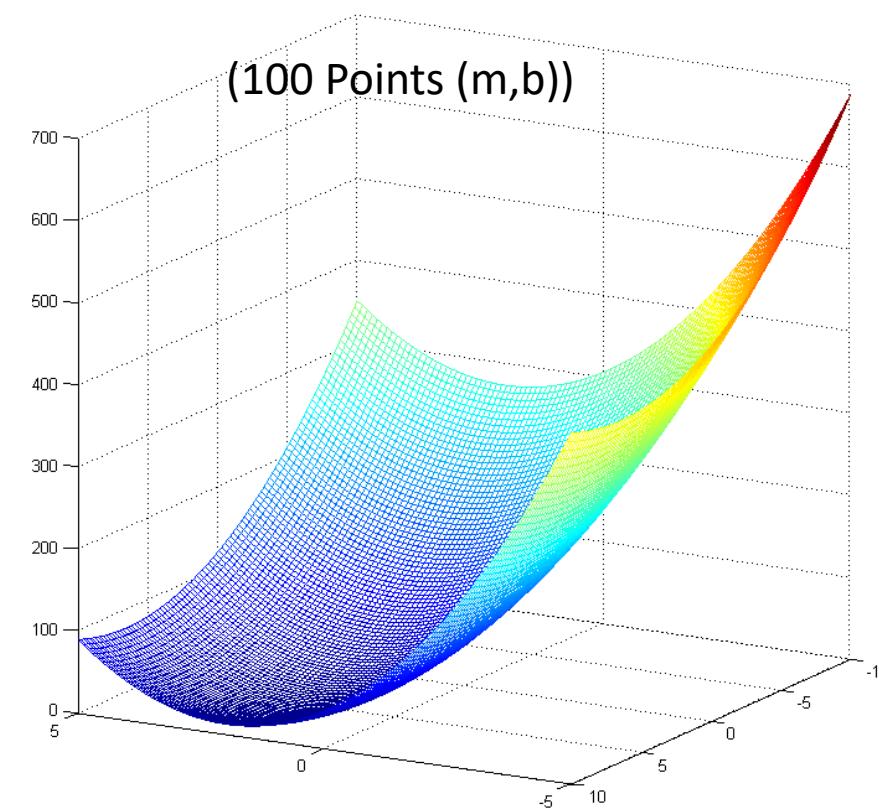
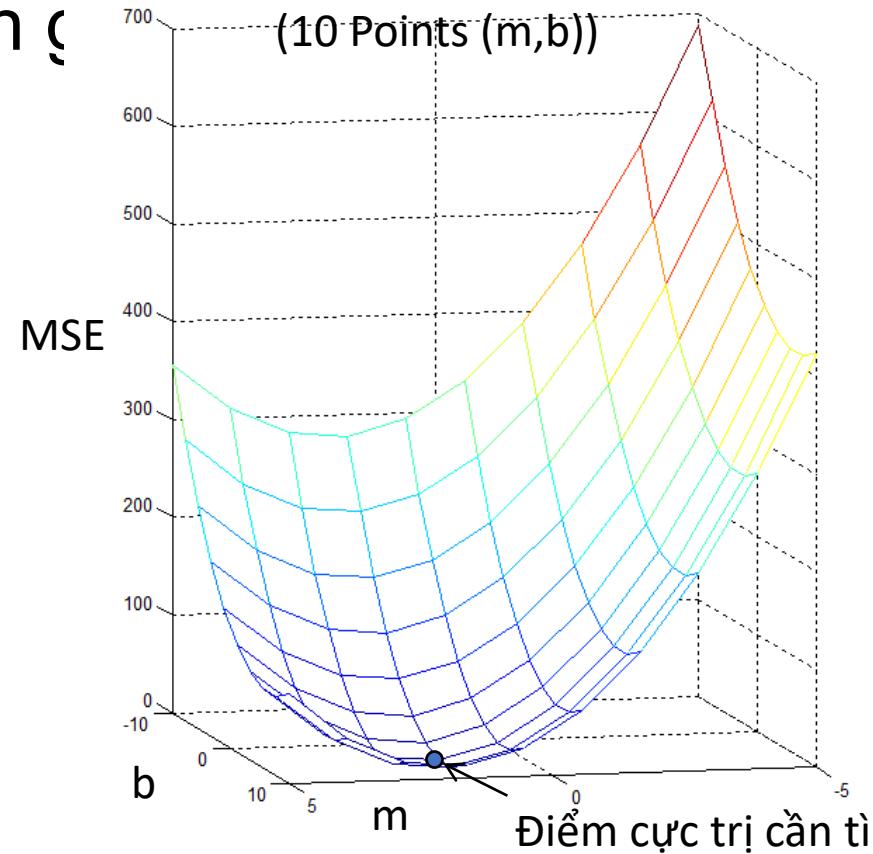
- Thông thường hàm lỗi là Mean Square Error

$$MSE = \frac{1}{N} \sum_{i=1}^N (y - (mx + b))^2$$

- Chuyển thành bài toán tìm điểm cực trị của hàm MSE theo tham số **(m,b)** (**không phải x,y**)
- Biểu diễn trực quan hàm lỗi

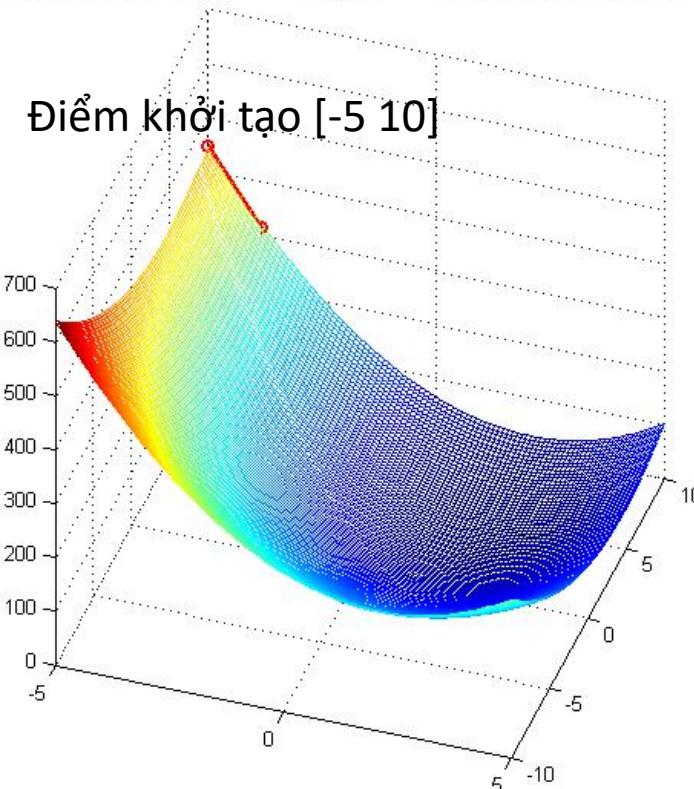
Biểu diễn trực quan hàm lồi

- Cho m chạy trong 1 dải (e.g., $m = [-5 5]$, $b=[-10 , 10]$)
- Tính ζ

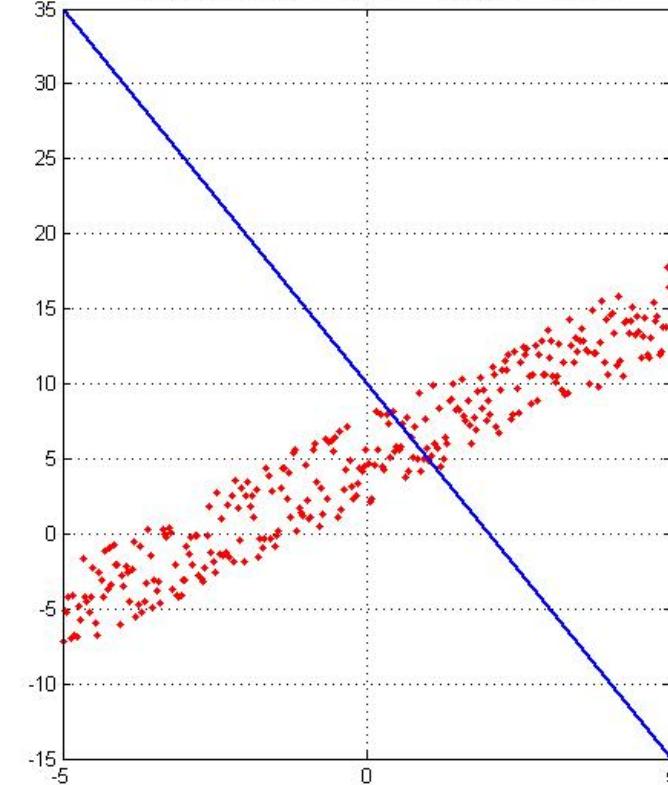


Tìm cực trị sử dụng GD

Current Iteration =0 , dx = Inf,grad = 118.66,learning rate = 0.01



Current Iteration =0 , m= -5.00,b=10.00



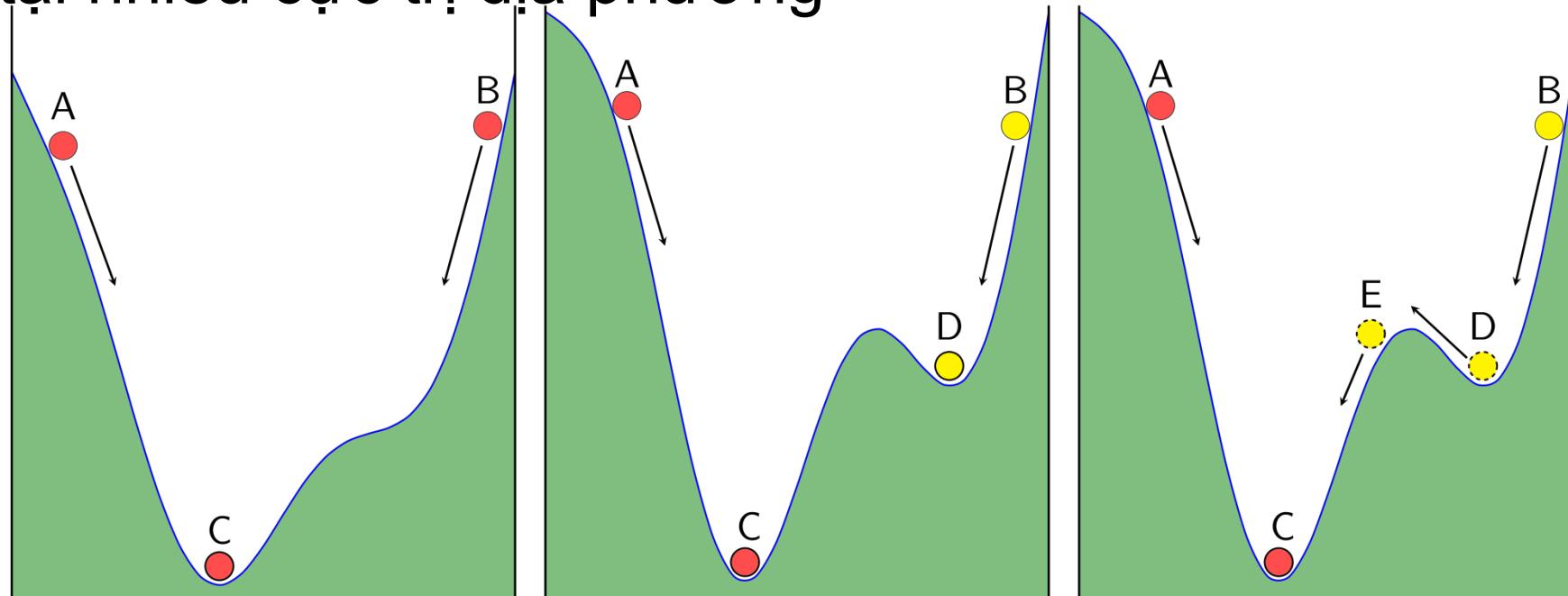
Đạo hàm riêng của m và b

$$\frac{\partial}{\partial m} = \frac{2}{N} \sum_{i=1}^N -x_i(y_i - (mx_i + b))$$

$$\frac{\partial}{\partial b} = \frac{2}{N} \sum_{i=1}^N -(y_i - (mx_i + b))$$

Một số vấn đề của GD

- Tồn tại nhiều cực trị địa phương



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

$$v_t = w_{t-1} + \eta \nabla_{\theta} f(\theta)$$

GD with momentum

$$v_0 = 0, \gamma = 0.9$$

Một số vấn đề của GD

- Đạo hàm riêng luôn tương minh (?)

- Tính theo numerical gradient

$$f'(x) \approx \frac{f(x + \varepsilon) - f(x - \varepsilon)}{2\varepsilon}$$

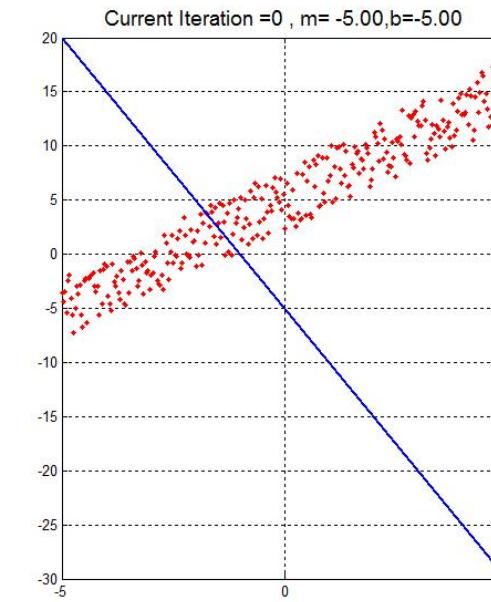
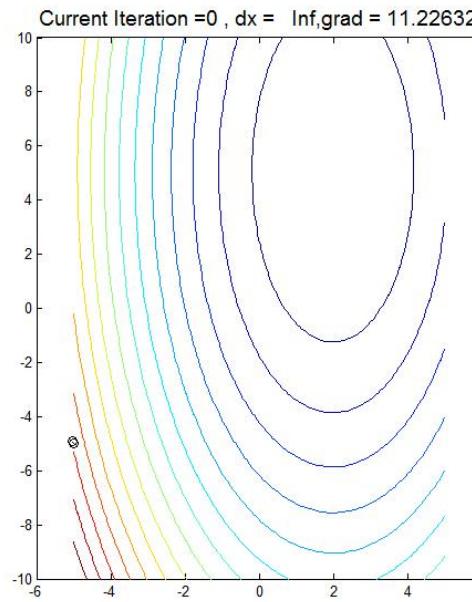
- Với hàm nhiều biến

- Giữ nguyên các biến còn lại, chỉ để 1 biến chạy

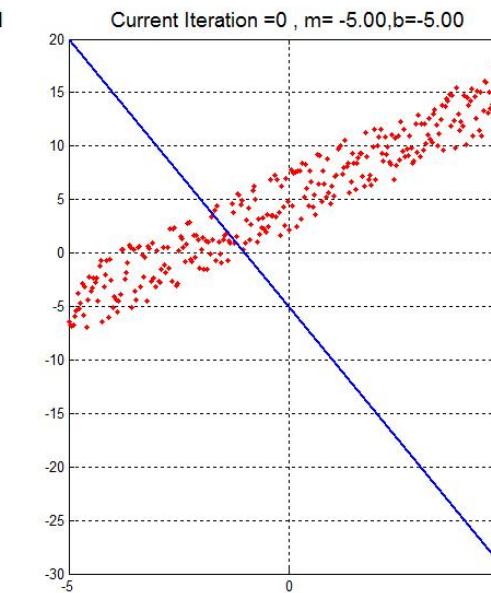
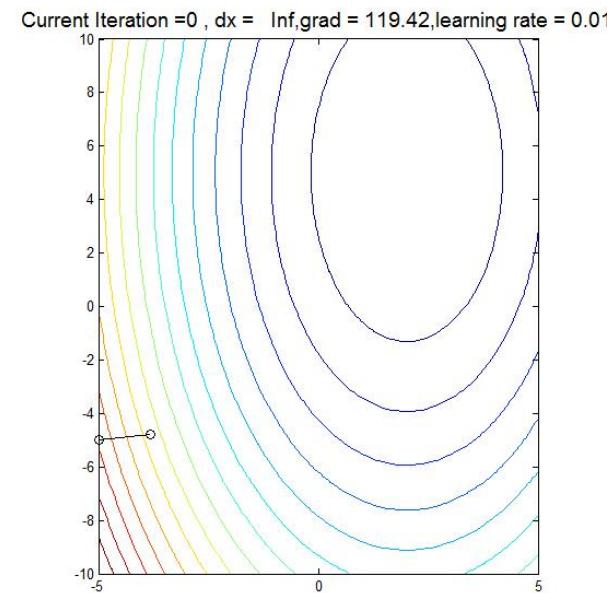
Stochastic Gradient Descent (SGD)

- Nguyên tắc cơ bản:
 - Cập nhật lại giá trị hàm lỗi sử dụng 1 mẫu ngẫu nhiên (ko phải toàn bộ các mẫu)
$$\theta_{new} = \theta - \alpha Grad(f(x_i))$$
 - Nếu bổ sung mẫu online , ko phải duyệt lại toàn bộ các mẫu đã có → rất có tác dụng với tập dữ liệu lớn; vẫn kế thừa được bộ tham số đã học từ các mẫu sẵn có
- Khái niệm epoch
 - 1 epoch: một lần duyệt qua toàn bộ dữ liệu (ví dụ: tính tổng; trung bình ; khoảng cách giữa các mẫu)
 - Với GD: 1 epoch thì 01 lần cập nhật bộ tham số
 - Với SGD: 1 epoch thì cập nhật N lần bộ tham số (N là tổng số mẫu)

SGD



GD



Mini-batch Gradient Descent

- Dùng từng phần dữ liệu để tính toán đạo hàm (SGD dùng từng mẫu)
- Chia tổng số mẫu thành k phần , mỗi phần gồm n mẫu là một mini-batch
- Tính toán/cập nhật tham số cho mỗi mini-batch
- Thường sử dụng trong deep learning

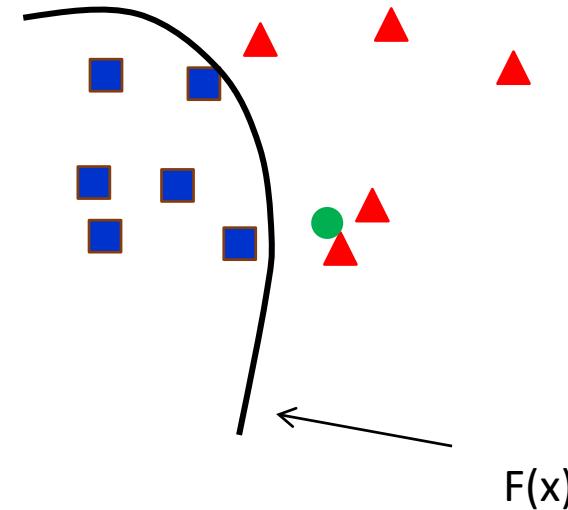
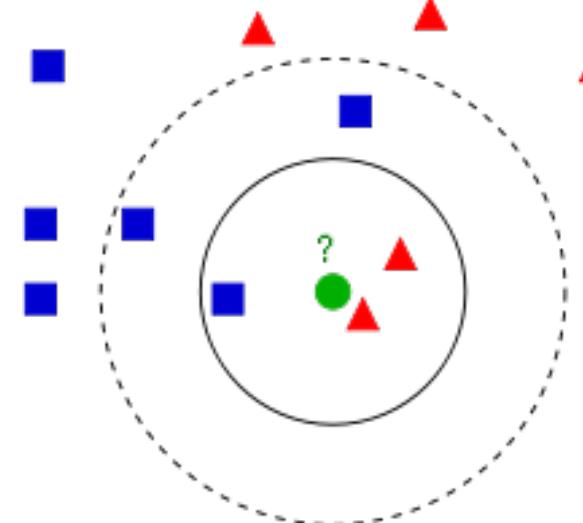
Sử dụng Mini-batch GD for K-Means

- Bài tập

- Thuật toán cơ bản
 - Initialize each $c \in C$ with an x picked randomly from X
 - $v \leftarrow 0$
 - for $i = 1$ to t do
 - $M \leftarrow b$ examples picked randomly from X
 - for $x \in M$ do
 - $d[x] \leftarrow f(C, x)$ // Cache the center nearest to x
 - end for
 - for $x \in M$ do
 - $c \leftarrow d[x]$ // Get cached center for this x
 - $v[c] \leftarrow v[c] + 1$ // Update per-center counts
 - $\eta \leftarrow 1 / v[c]$ // Get per-center learning rate
 - $c \leftarrow (1 - \eta)c + nx$ // Take gradient step
 - end for
 - end for

Sử dụng GD for K-NN

- Xây dựng hàm phân tách đủ tốt



- Đếm số lượng phân lớp nhầm (điểm bên trái $f(x)$ và bên phải $f(x)$)
 - Tối ưu hóa dùng GD
- Không khả thi vì tính toán trên hàm lõi rác;

Ưu nhược điểm của Neural Networks



- Dễ dàng xây dựng được mô hình biểu diễn, tính toán
- Mô hình đa dạng, có thể sử dụng trong nhiều bài toán khác nhau (phân lớp, phân cụm, phân tích chuỗi)
- Sử dụng trong nhiều ứng dụng khác nhau: ảnh, text, voice ...
- Vấn đề hộp đen (blackbox): không giải thích được, không suy diễn được
- Cần nhiều dữ liệu tính toán
- Huấn luyện mô hình phức tạp

BÀI TOÁN PHÂN LOẠI CHỮ VIẾT TAY

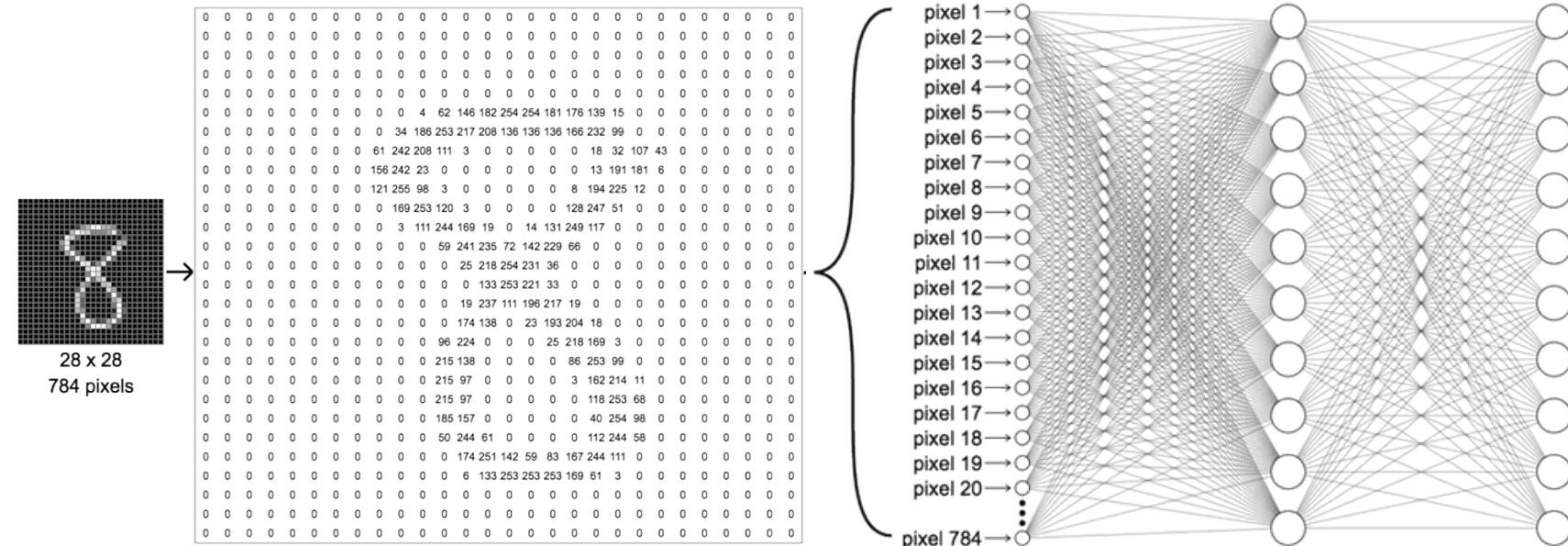
Bài toán phân loại chữ viết tay

- Dữ liệu: Bộ dữ liệu MNIST
- <http://yann.lecun.com/exdb/mnist/>
- 60,000 ảnh cho huấn luyện
- 10,000 ảnh cho kiểm thử mô hình
- Mọi ảnh chữ số viết tay trong tập dữ liệu được chuẩn hóa về kích thước, cụ thể là (28x28) với giá trị pixels nằm trong khoảng 0 đến 1



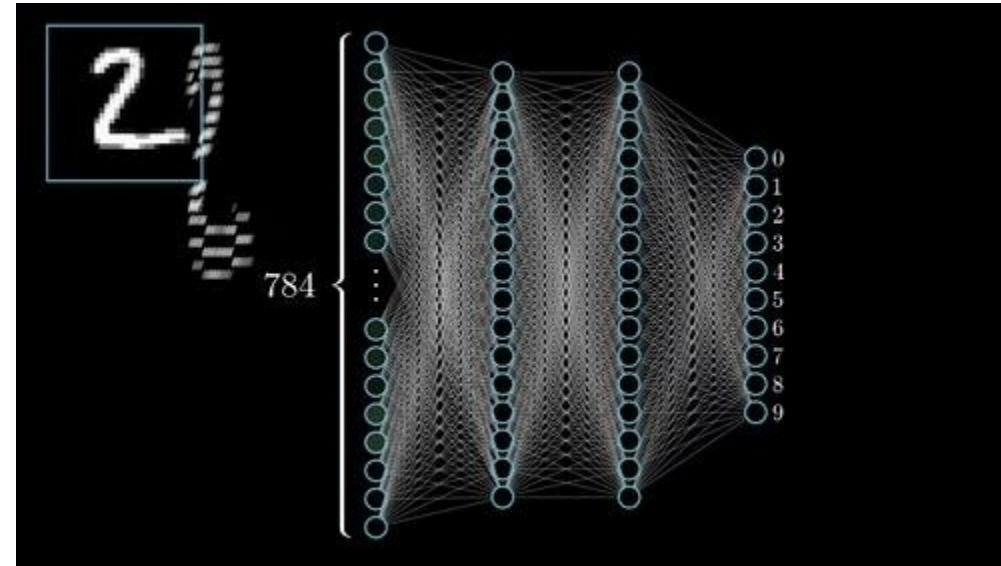
Dữ liệu đầu vào cho mạng neuron

- Vector đầu vào: 784 feature với các số trong khoảng từ 0-255
- Nhãn từ 0 đến 9



Xây dựng mô hình

- Input: vector 784 features
- Hai lớp ẩn với hàm kích hoạt là ReLU
- Lớp output với softmax classifier



```
# build model
from keras.layers import Dense, Flatten
from keras.models import Sequential

model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(n_hidden_1, activation='relu')) # hidden Layer1
model.add(Dense(n_hidden_2, activation='relu')) # hidden Layer2
model.add(Dense(num_classes, activation='softmax')) # output Layer
```

Huấn luyện mô hình

- Hàm mất mát (loss function): cross entropy
- Cập nhật tham số bằng Stochastic Gradient Desent
- Các siêu tham số:
 - learning_rate = 0.1
 - num_epoch = 10
 - batch_size = 128
- Tham khảo các bước còn lại trên github

K Keras

```
# loss, optimizers
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=learning_rate),
              metrics=['accuracy'])

model.fit(x_train, y_train, batch_size=batch_size, epochs=num_epoch)
```

Bài tập

- Quan sát và làm lại mẫu bài tập trên github
- Tìm và thay thế các phần thiếu hoặc thay thế các tham số, các phương pháp huấn luyện
- Thủ với các bài toán trong các buổi học trước



Q&A



Thank you!