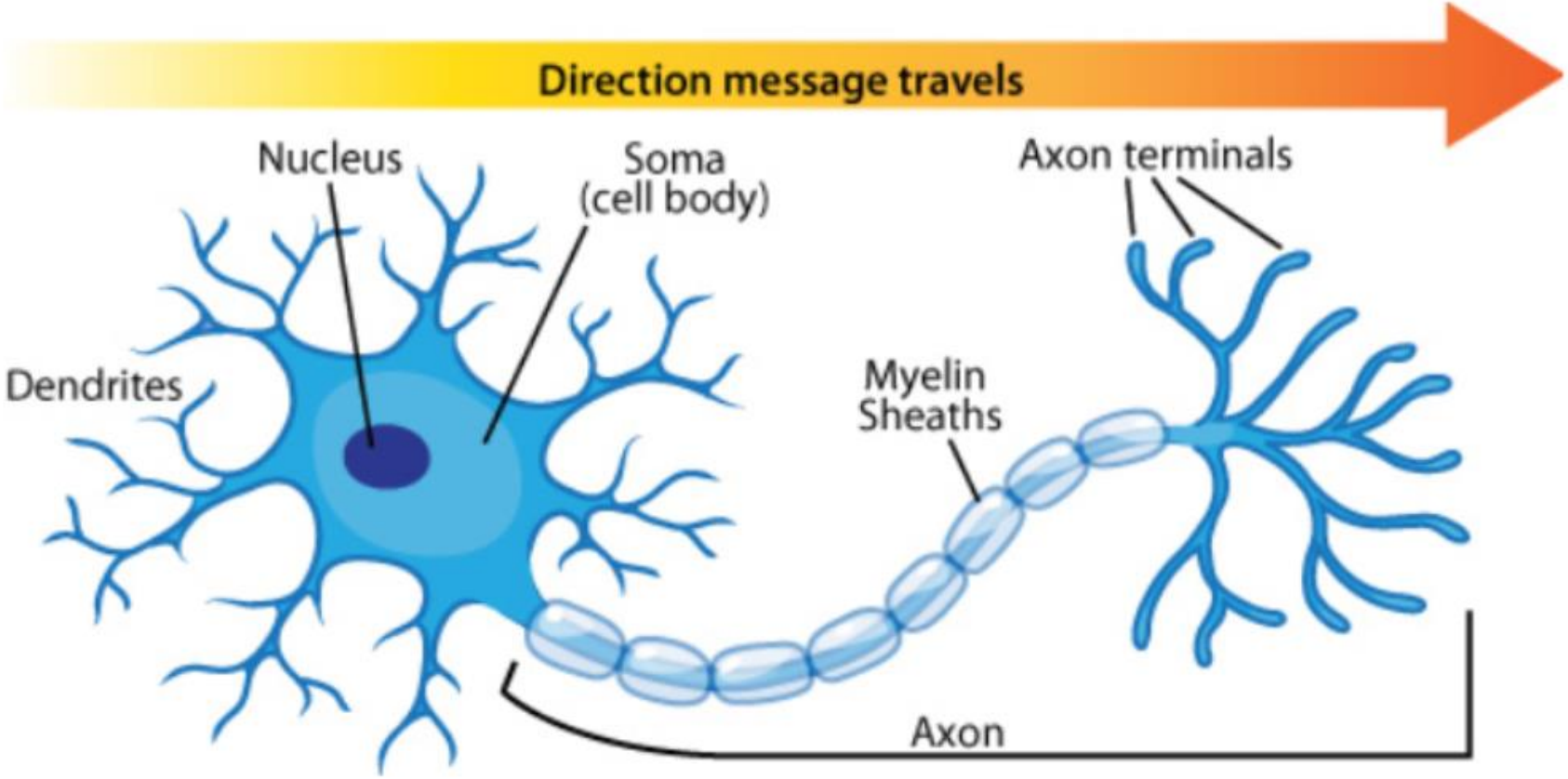
HỌC SÂU

BÀI 3. MẠNG NEURON (NEURAL NETWORK)

Neural network là gì?

Neural là tính từ của neuron (nơ-ron), network chỉ cấu trúc, cách các nơ-ron đó liên kết với nhau, nên neural network (NN) là một hệ thống tính toán lấy cảm hứng từ sự hoạt động của các nơ-ron trong hệ thần kinh.

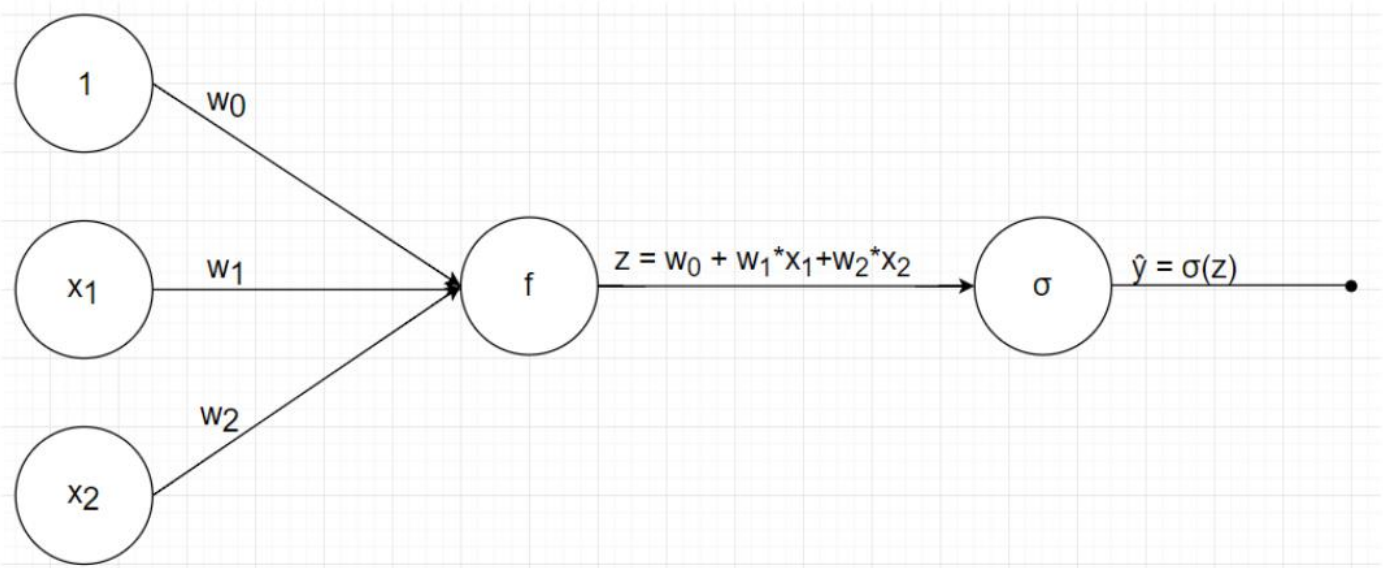
Neuron Anatomy



Mô hình neural network

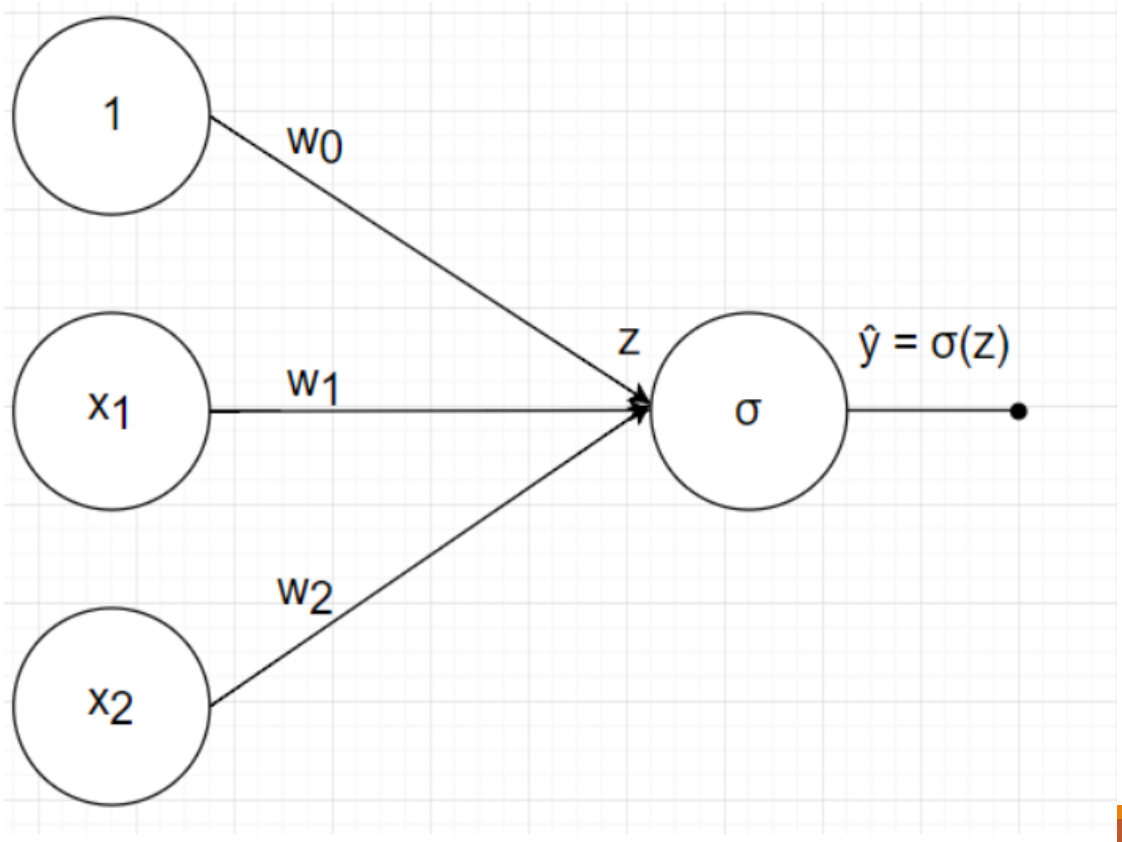
Logistic regression là mô hình neural network đơn giản nhất chỉ với input layer và output layer. Mô hình của logistic regression từ bài trước là: $\hat{y} = s(w0 + w1 * x1 + w2 * x2)$. Có 2 bước:

- Tính tổng linear: $z = 1 * w0 + x1 * w1 + x2 * w2$
- Áp dụng sigmoid function: $\hat{y} = s(z)$



Mô hình neural network

Để biểu diễn gọn lại ta sẽ gộp hai bước trên thành một trên biểu đồ như hình:



Mô hình neural network

Hệ số $w0$ được gọi là bias. Để ý từ những bài trước đến giờ dữ liệu khi tính toán luôn được thêm 1 để tính hệ số bias $w0$.

Tại sao lại cần hệ số bias? Quay lại với bài 1, phương trình đường thẳng sẽ thế nào nếu bỏ $w0$, phương trình giờ có dạng: $y = w1 * x$, sẽ luôn đi qua gốc tọa độ và nó không tổng quát hóa phương trình đường thẳng nên có thể không tìm được phương trình mong muốn.

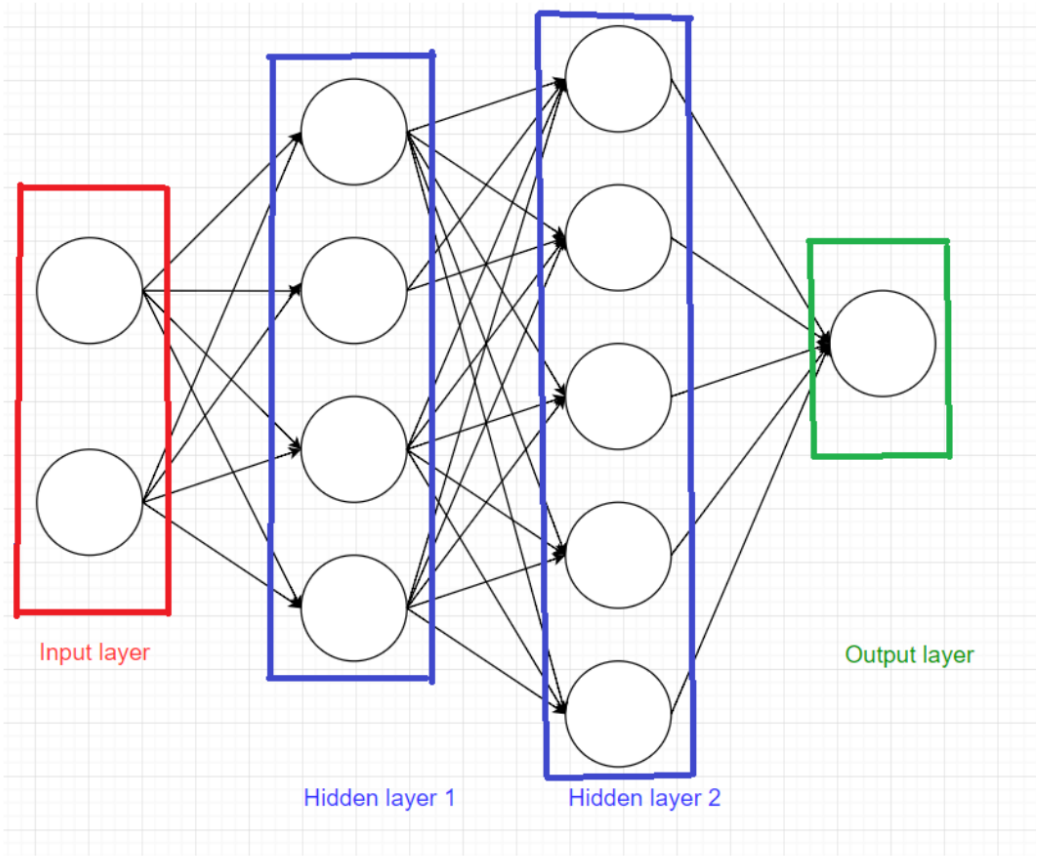
Hàm sigmoid ở đây được gọi là activation function.

Mô hình tổng quát

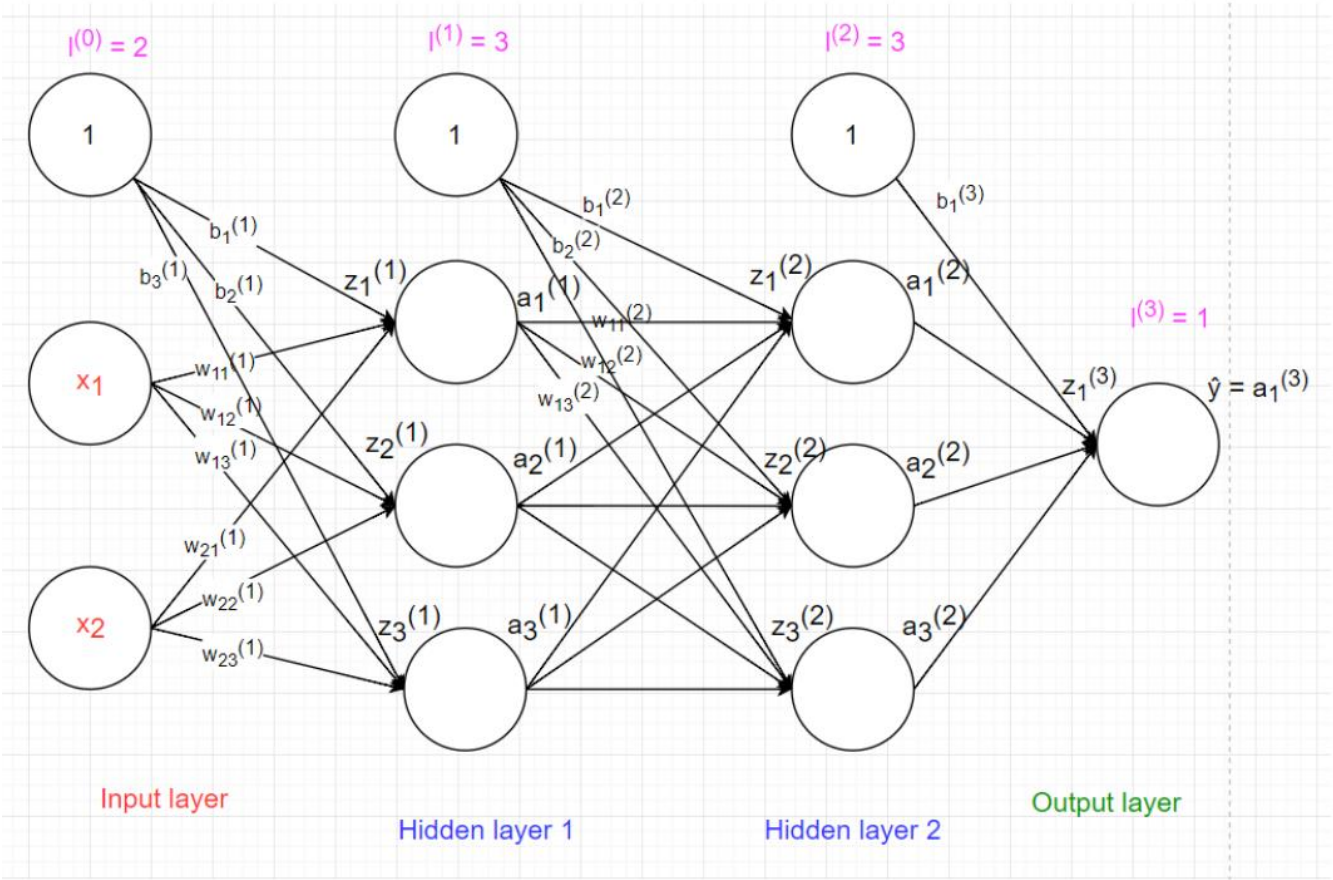
Layer đầu tiên là input layer, các layer ở giữa được gọi là hidden layer, layer cuối cùng được gọi là output layer. Các hình tròn được gọi là node.

Mỗi mô hình luôn có 1 input layer, 1 output layer, có thể có hoặc không các hidden layer. Tổng số layer trong mô hình được quy ước là số layer - 1 (không tính input layer).

Mô hình tổng quát



Kí hiệu

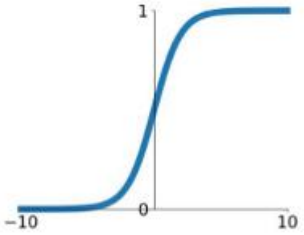

$l^{(0)} = 2$
$l^{(1)} = 3$
$l^{(2)} = 3$
$l^{(3)} = 1$
1
1
1
x_1
x_2
$b_1^{(1)}$
$b_2^{(1)}$
$b_3^{(1)}$
$w_{11}^{(1)}$
$w_{12}^{(1)}$
$w_{13}^{(1)}$
$w_{21}^{(1)}$
$w_{22}^{(1)}$
$w_{23}^{(1)}$
$z_1^{(1)}$
$z_2^{(1)}$
$z_3^{(1)}$
$a_1^{(1)}$
$a_2^{(1)}$
$a_3^{(1)}$
$b_1^{(2)}$
$b_2^{(2)}$
$w_{11}^{(2)}$
$w_{12}^{(2)}$
$w_{13}^{(2)}$
$z_1^{(2)}$
$z_2^{(2)}$
$z_3^{(2)}$
$a_1^{(2)}$
$a_2^{(2)}$
$a_3^{(2)}$
$b_1^{(3)}$
$z_1^{(3)}$
$\hat{y} = a_1^{(3)}$
Input layer
Hidden layer 1
Hidden layer 2
Output layer

Hàm kích hoạt

linear
step function
logistic (sigmoid) function
tanh function
rectified linear unit (ReLu) function

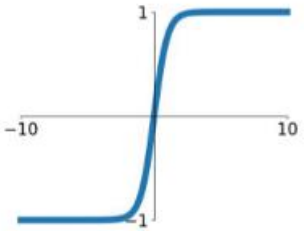
Sigmoid

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



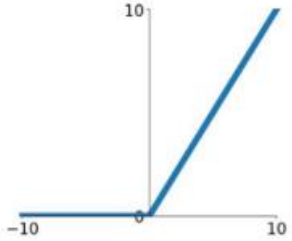
tanh

$$\tanh(x)$$



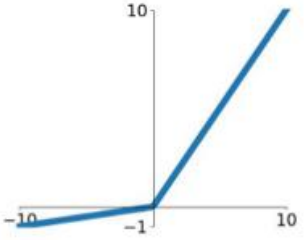
ReLU

$$\max(0, x)$$



Leaky ReLU

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

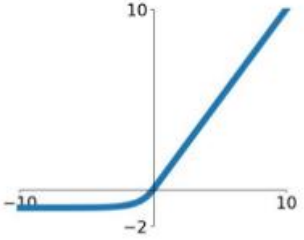


Maxout

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

ELU

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



Quá trình học của mạng nơ-ron

Khái niệm về học (Learning):

Học trong mạng nơ-ron là quá trình điều chỉnh các trọng số (weights) và độ lệch (biases) để giảm thiểu sai số giữa đầu ra dự đoán và đầu ra mong muốn. Quá trình này còn được gọi là "huấn luyện" (training).

Quá trình học của mạng nơ-ron

Hàm mất mát (Loss Function):

Hàm mất mát đo lường sai số giữa giá trị dự đoán và giá trị thực tế. Một số hàm mất mát phổ biến:

Bình phương sai số trung bình (Mean Squared Error - MSE):

$MSE = (1/n) \times \Sigma(y\_true - y\_pred)^2$ Bài toán hồi quy

Entropy chéo (Cross-Entropy):

$CE = -\Sigma(y\_true \times \log(y\_pred))$ Bài toán phân loại

Quá trình học của mạng nơ-ron

Ví dụ 4: Tính MSE

Dự đoán: [0.2, 0.7]

Thực tế: [0, 1]

MSE = $(1/2) \times [(0-0.2)^2 + (1-0.7)^2] = (1/2) \times [0.04 + 0.09] = 0.065$

Quá trình học của mạng nơ-ron

Thuật toán Gradient Descent:

Gradient Descent là kỹ thuật tối ưu hóa dùng để điều chỉnh trọng số trong mạng nơ-ron:

1. Bắt đầu với các trọng số ngẫu nhiên
2. Tính toán gradient (đạo hàm) của hàm mất mát theo mỗi trọng số
3. Điều chỉnh trọng số ngược với hướng gradient (để giảm giá trị hàm mất mát)
4. Lặp lại quá trình cho đến khi hàm mất mát đủ nhỏ hoặc đạt số lần lặp tối đa

Công thức cập nhật trọng số: $w\_new = w\_old - learning\_rate \times gradient$

Quá trình học của mạng nơ-ron

Ví dụ 5: Gradient Descent đơn giản

Giả sử ta có hàm mất mát $J(w) = w^2$ và trọng số ban đầu $w = 5$:

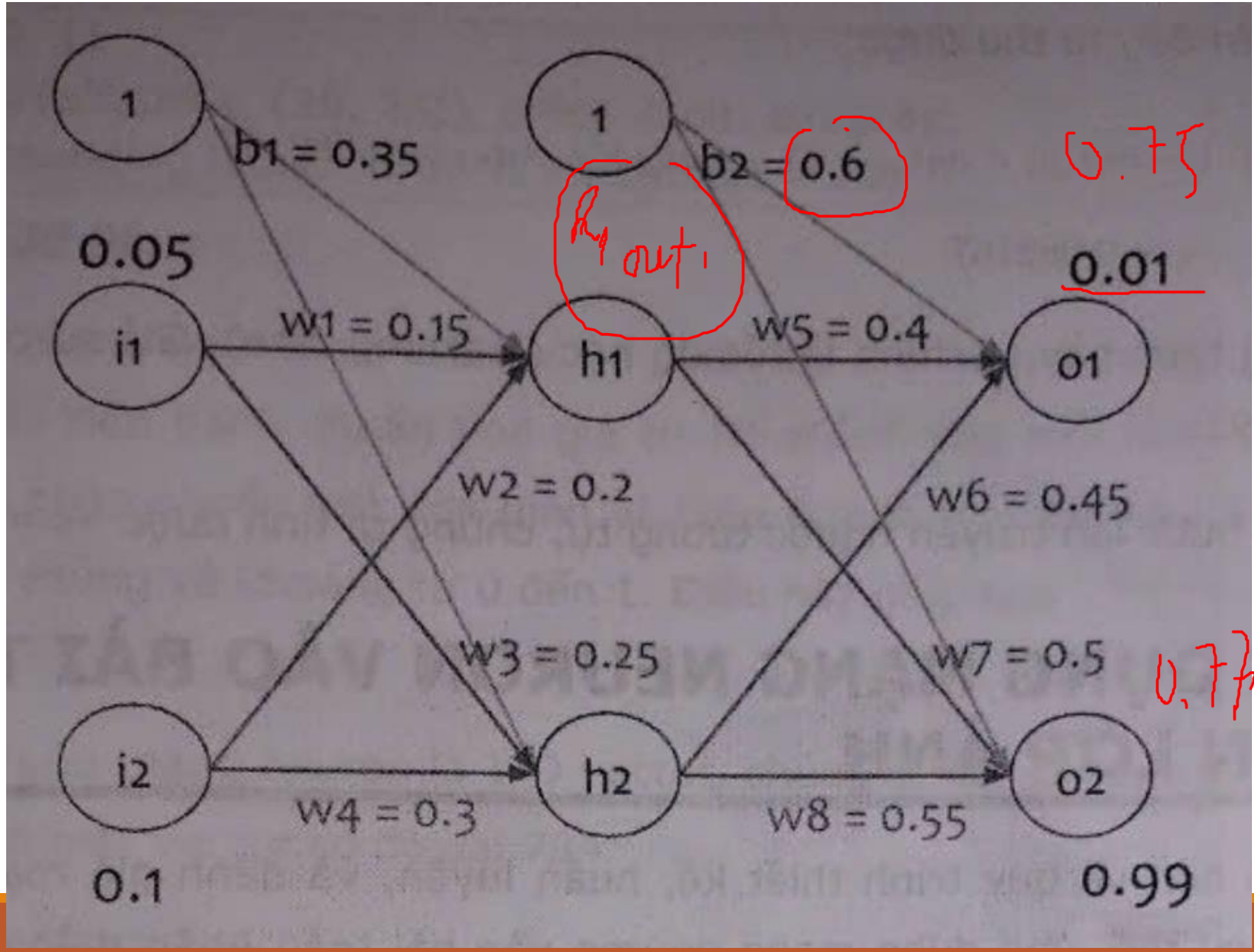
- Gradient: $\partial J/\partial w = 2w = 2 \times 5 = 10$
- Learning rate: 0.1
- Cập nhật w: $w\_new = 5 - 0.1 \times 10 = 4$
- Lặp lại quá trình này, w sẽ dần tiến về 0 (giá trị tối thiểu của hàm w^2)

Quá trình học của mạng nơ-ron

Thuật toán Backpropagation:

Backpropagation (lan truyền ngược) là thuật toán hiệu quả để tính gradient trong mạng nơ-ron nhiều lớp:

1. **Lan truyền thuận (Forward Pass)**: Tính đầu ra của mạng từ đầu vào
2. **Tính lỗi (Error Calculation)**: So sánh đầu ra với giá trị mong muốn
3. **Lan truyền ngược (Backward Pass)**: Truyền lỗi ngược từ output về input, tính gradient cho mỗi trọng số
4. **Cập nhật trọng số (Weight Update)**: Điều chỉnh trọng số dựa trên gradient


1
1
b1 = 0.35
b2 = 0.6
0.05
i1
w1 = 0.15
h1
w5 = 0.4
o1
0.01
w2 = 0.2
w6 = 0.45
w3 = 0.25
w7 = 0.5
i2
h2
o2
w4 = 0.3
w8 = 0.55
0.1
0.99

- Lan truyền tiến:

$$in_{h1} = w_1 \times i_1 + w_2 \times i_2 + b_1 = 0.3775$$

$$out_{h1} = sigmoid(in_{h1}) = 0.593$$

$$in_{h2} = w_3 \times i_1 + w_4 \times i_2 + b_1 = 0.3925$$

$$out_{h2} = sigmoid(in_{h2}) = 0.597$$

$$in_{o1} = w_5 \times out_{h1} + w_6 \times out_{h2} + b_2 = 0.$$

$$out_{o1} = sigmoid(in_{o1}) = 0.751$$

$$in_{o2} = w_7 \times out_{h1} + w_8 \times out_{h2} + b_2$$

$$out_{o2} = sigmoid(in_{o2}) = 0.773$$

$$J(w) = \frac{1}{2}((out_{o1} - o1)^2 + (out_{o2} - o2)^2) = 0.3$$

- Lan truyền ngược:

$$w_5 \Rightarrow w_5 - \alpha \times \frac{\partial J(w)}{\partial w_5}$$

$$\frac{\partial J(w)}{\partial w_5} = \frac{\partial J(w)}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

Sau khi biến đổi, ta thu được:

$$\frac{\partial J(w)}{\partial w_5} = -(o1 - out_{o1}) \times out_{o1} \times (1 - out_{o1}) \times out_{h1} = 0.082167$$

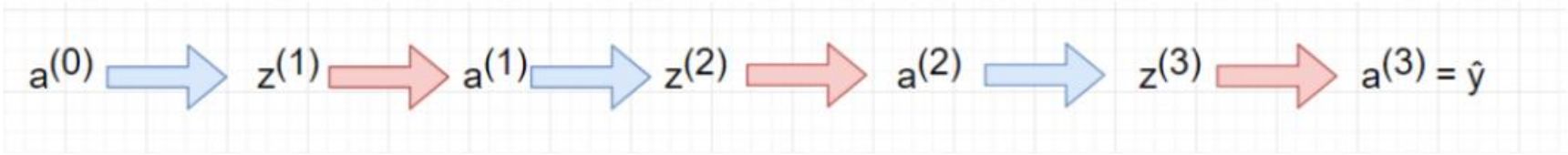
$$\rightarrow w_5 = 0.4 - \alpha \times 0.082167$$

Trong công thức này, α chính là tốc độ học (learning rate). Giả sử chọn $\alpha = 0.5$, khi đó: $w_5 = 0.35892$

Lặp lại các bước lan truyền ngược tương tự, chúng ta tính được w_6, w_7, và w_8

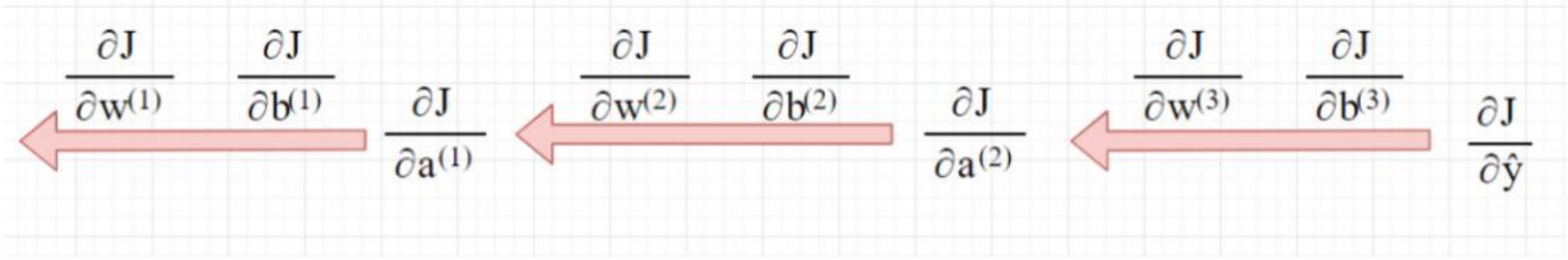
Nếu hàm activation là sigmoid thì $\frac{\partial A^{(i)}}{\partial Z^{(i)}} = A^{(i)} \otimes (1 - A^{(i)})$

Ở bài trước quá trình feedforward



Hình 6.3: Quá trình feedforward

Thì ở bài này quá trình tính đạo hàm ngược lại



Hình 6.4: Quá trình backpropagation

Ví dụ

```python
# Import thư viện numpy để tính toán với mảng và ma trận
import numpy as np

# Bài toán XOR: Có 4 mẫu đầu vào và đầu ra tương ứng
# Tạo mảng X chứa 4 mẫu đầu vào: [0,0], [0,1], [1,0], [1,1]
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Đầu vào
# Tạo mảng y chứa kết quả đầu ra tương ứng với 4 mẫu đầu vào theo phép XOR: 0, 1, 1, 0
y = np.array([[0], [1], [1], [0]])              # Đầu ra mong muốn

# Hàm kích hoạt sigmoid và đạo hàm của nó
def sigmoid(x):
    # Hàm sigmoid: f(x) = 1/(1+e^(-x)), giới hạn đầu ra trong khoảng (0,1)
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    # Đạo hàm của hàm sigmoid: f'(x) = f(x) * (1 - f(x))
    # Lưu ý: x ở đây đã là giá trị của hàm sigmoid, không phải đầu vào gốc
    return x * (1 - x)
```

```python
# Định nghĩa lớp mạng nơ-ron
class NeuralNetwork:
    def __init__(self, x, y):
        # Khởi tạo mạng với dữ liệu đầu vào x và đầu ra mong muốn y
        self.input = x
        # Khởi tạo ngẫu nhiên trọng số kết nối từ lớp đầu vào (2 node) đến lớp ẩn (4 node)
        self.weights1 = np.random.rand(self.input.shape[1], 4)  # Trọng số lớp đầu vào -> lớp ẩn
        # Khởi tạo ngẫu nhiên trọng số kết nối từ lớp ẩn (4 node) đến lớp đầu ra (1 node)
        self.weights2 = np.random.rand(4, 1)                    # Trọng số lớp ẩn -> lớp đầu ra
        # Lưu trữ đầu ra mong muốn
        self.y = y
        # Khởi tạo mảng đầu ra với kích thước giống y và giá trị ban đầu bằng 0
        self.output = np.zeros(y.shape)

    def feedforward(self):
        # Lan truyền thuận - tính đầu ra của mạng với trọng số hiện tại
        # Tính đầu ra của lớp ẩn: input * weights1 qua hàm sigmoid
        self.layer1 = sigmoid(np.dot(self.input, self.weights1))  # Đầu ra của lớp ẩn
        # Tính đầu ra của lớp output: layer1 * weights2 qua hàm sigmoid
        self.output = sigmoid(np.dot(self.layer1, self.weights2)) # Đầu ra của lớp output
```

```
41      def backprop(self):
42          # Lan truyền ngược - cập nhật trọng số dựa trên lỗi
43          # Tính gradient
44          # Tính đạo hàm của lỗi theo weights2:
45          # (layer1.T là chuyển vị của layer1) nhân với (2 * lỗi * đạo hàm sigmoid tại output)
46          # Hệ số 2 từ đạo hàm của hàm lỗi bình phương
47          d_weights2 = np.dot(self.layer1.T,
48                              (2*(self.y - self.output) * sigmoid_derivative(self.output)))
49
50          # Tính đạo hàm của lỗi theo weights1:
51          # (input.T là chuyển vị của input) nhân với (đạo hàm lỗi theo layer1 * đạo hàm sigmoid tại layer1)
52          # Đạo hàm lỗi theo layer1 = đạo hàm lỗi theo output nhân với weights2.T
53          d_weights1 = np.dot(self.input.T,
54                              (np.dot(2*(self.y - self.output) * sigmoid_derivative(self.output),
55                                      self.weights2.T) * sigmoid_derivative(self.layer1)))
56
57          # Cập nhật trọng số bằng cách cộng với gradient (learning rate mặc định là 1)
58          self.weights1 += d_weights1
59          self.weights2 += d_weights2
60
61  # Khởi tạo đối tượng mạng neural với dữ liệu X và y
62  nn = NeuralNetwork(X, y)
```

```python
# Khởi tạo đối tượng mạng neural với dữ liệu X và y
nn = NeuralNetwork(X, y)

# Huấn luyện mạng neural qua 10000 vòng lặp (epochs)
for i in range(10000):
    # Tính đầu ra với trọng số hiện tại
    nn.feedforward()
    # Cập nhật trọng số dựa trên lỗi
    nn.backprop()

    # In ra lỗi sau mỗi 1000 epochs để theo dõi quá trình huấn luyện
    # Lỗi được tính bằng trung bình bình phương sai khác giữa y và output
    if i % 1000 == 0:
        print(f"Epoch {i}: Loss = {np.mean(np.square(y - nn.output))}")

# Tính đầu ra cuối cùng với trọng số đã huấn luyện
nn.feedforward()
# In kết quả dự đoán của mạng neural
print("\nKết quả dự đoán:")
print(nn.output)
```

Cải thiện mạng nơ-ron cơ bản

Kỹ thuật Regularization

Regularization giúp tránh overfitting (quá khớp) bằng cách thêm ràng buộc vào trọng số:

L1 Regularization

Thêm tổng giá trị tuyệt đối của trọng số vào hàm mất mát:

$J\_new = J + \lambda \times \Sigma|w|$

L2 Regularization

Thêm tổng bình phương trọng số vào hàm mất mát:

$J\_new = J + \lambda \times \Sigma w^2$

Trong đó λ là hệ số regularization.

Cải thiện mạng nơ-ron cơ bản

Dropout

Dropout là kỹ thuật ngẫu nhiên tắt một số nơ-ron trong quá trình huấn luyện, giúp:

- Tránh overfitting
- Tạo ra mạng "robust" hơn, không phụ thuộc vào nơ-ron cụ thể

```
# Ví dụ dropout đơn giản
def apply_dropout(layer, dropout_rate=0.5):
    mask = np.random.binomial(1, 1-dropout_rate, size=layer.shape) / (1-dropout_rate)
    return layer * mask
```

Cải thiện mạng nơ-ron cơ bản

Batch Normalization

Batch Normalization chuẩn hóa đầu vào của mỗi lớp, giúp:

- Tăng tốc độ hội tụ
- Giảm sự phụ thuộc vào khởi tạo trọng số
- Cho phép sử dụng learning rate lớn hơn

```
1  # Ví dụ batch normalization đơn giản
2  def batch_normalize(x, epsilon=1e-8):
3      mean = np.mean(x, axis=0)
4      var = np.var(x, axis=0)
5      return (x - mean) / np.sqrt(var + epsilon)
```

Ứng dụng

Ứng dụng mạng NN trong phân lớp ảnh

Ứng dụng mạng NN trong bài toán hồi quy

Bài thực hành 1: Nhận dạng chữ số viết tay với MNIST

Bài thực hành này hướng dẫn xây dựng một mạng nơ-ron để nhận dạng chữ số viết tay từ bộ dữ liệu MNIST. Mã nguồn bao gồm:

- **Tải và khám phá dữ liệu MNIST**: 70.000 hình ảnh chữ số viết tay (60.000 mẫu huấn luyện, 10.000 mẫu kiểm tra)
- **Tiền xử lý dữ liệu**: Chuẩn hóa pixel về khoảng [0,1], chuyển đổi nhãn sang one-hot encoding
- **Xây dựng mô hình**: Mạng nơ-ron với một lớp ẩn 128 nơ-ron
- **Huấn luyện và đánh giá**: Theo dõi quá trình huấn luyện và đánh giá độ chính xác
- **Trực quan hóa kết quả**: Hiển thị dự đoán và ma trận nhầm lẫn

Bài thực hành 2: Dự đoán giá nhà với Mạng nơ-ron

Bài thực hành này hướng dẫn xây dựng mạng nơ-ron để giải quyết bài toán hồi quy - dự đoán giá nhà dựa trên bộ dữ liệu California Housing. Mã nguồn bao gồm:

- **Tải và khám phá dữ liệu**: Sử dụng bộ dữ liệu California Housing từ scikit-learn
- **Phân tích dữ liệu (EDA)**: Phân tích mối tương quan giữa các đặc trưng và giá nhà
- **Tiền xử lý dữ liệu**: Chuẩn hóa đặc trưng bằng StandardScaler
- **Xây dựng mô hình**: Mạng nơ-ron với hai lớp ẩn (64 nơ-ron mỗi lớp)
- **Huấn luyện và đánh giá**: Huấn luyện mô hình và đánh giá bằng các chỉ số MSE, RMSE, MAE và R^2
- **Cải thiện mô hình**: Thử nghiệm kiến trúc phức tạp hơn và so sánh hiệu suất