OLS

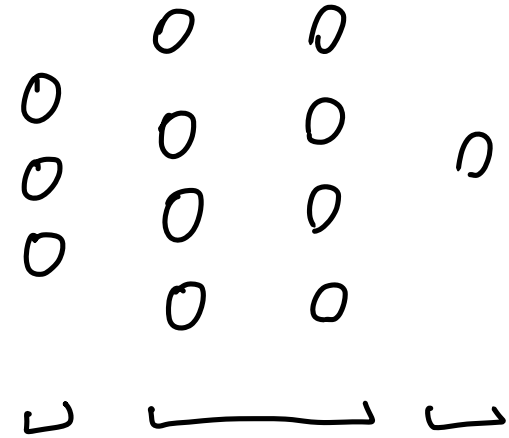
|

Logistic Regression

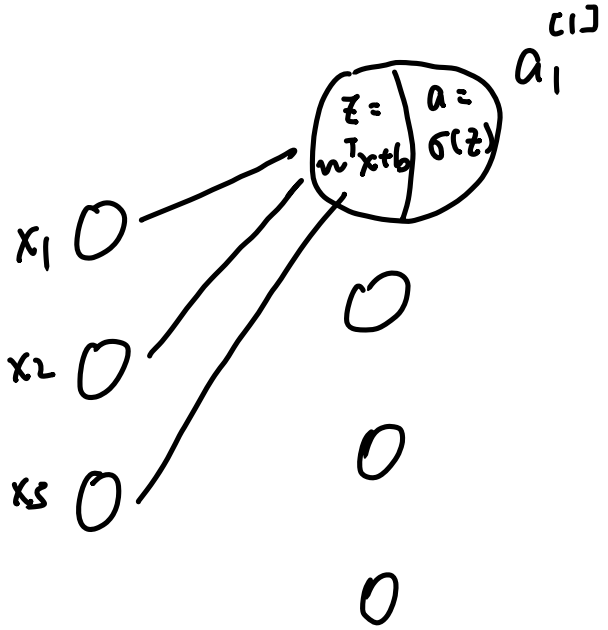
① $\sigma(w^T x + b)$

② Cost fct $J(w, b \mid X, Y)$ → Optim : Gradient Descent

$$w = w - \alpha \cdot \frac{\partial J}{\partial w}$$



Input Layer | Hidden Layer ↑ relu, tanh | Output Layer ↑ sigmoid



$$\underset{4\times m}{\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}} = \underset{\uparrow 1\times m}{\begin{bmatrix} w_1^T x + b_1 \\ w_2^T x + b_2 \\ w_3^T x + b_3 \\ w_4^T x + b_4 \end{bmatrix}} = \underset{1\times 3 \;\; 3\times m}{\begin{bmatrix} w_1^T x \\ w_2^T x \\ w_3^T x \\ w_4^T x \end{bmatrix}} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$$\underset{m\times p\ p\times 1}{Ax} = \begin{bmatrix} c_1 & c_2 & \cdots & c_p \end{bmatrix} x = x_1 c_1 + \cdots + x_p c_p$$

$$= \begin{bmatrix} r_1 \\ r_2 \\ \cdots \\ r_m \end{bmatrix} x = \begin{bmatrix} r_1 x \\ r_2 x \\ \cdots \\ r_m x \end{bmatrix}$$

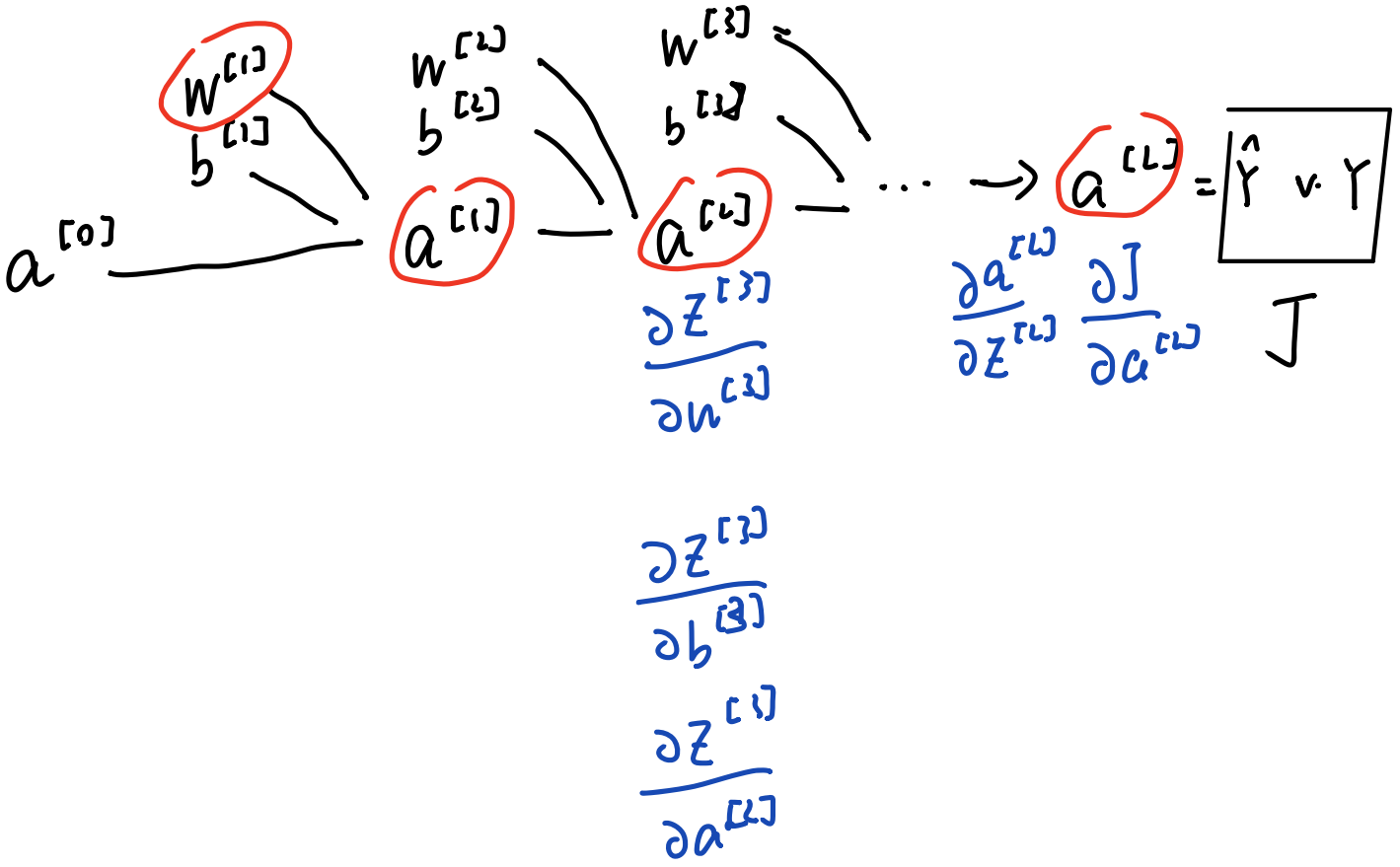
$$\underset{m\times n\ n\times p}{AB} = A \begin{bmatrix} c_1 & c_2 & \cdots & c_p \end{bmatrix} = \begin{bmatrix} Ac_1 & Ac_2 & \cdots & Ac_p \end{bmatrix}$$

$$= \begin{bmatrix} r_1 \\ r_2 \\ \cdots \\ r_m \end{bmatrix} B = \begin{bmatrix} r_1 B \\ r_2 B \\ \cdots \\ r_m B \end{bmatrix}$$

$$\begin{bmatrix} \underset{1\times 3\ \ 3\times m}{w_1^T x} \\ w_2^T x \\ w_3^T x \\ w_4^T x \end{bmatrix} = \begin{bmatrix} w_1^T \\ w_2^T \\ w_3^T \\ w_4^T \end{bmatrix} B = \underset{3\times 4}{W} B$$

$$\underset{1\times m}{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}} \quad 4\times 1 \Rightarrow 4\times m \quad \text{Broadcasting}$$

Computation Graph



In every iteration, $dw^{[l]}$

$db^{[l]}$

Logistic Regression

```
initialize

for i in num_iter:
    forward propagate
        Z = w.T @ X + b
        A = sigmoid(Z)
    compute cost
        J = -sum( Y * log(A2) + (1-Y) * log(1-A2)) / m
    backward
        dw = X @ (A-Y).T / m
        db = np.sum(A-Y) / m
    update params
        w = w - learning_rate * dw
        b = b - learning_rate * db
get w, b
predict
```

Shallow Neural Net

```
initialize

for i in num_iter:
    forward propagate
        Z1 = W1 @ X + b1
        A1 = np.tanh(Z1)
        Z2 = W2 @ A1 + b2
        A2 = sigmoid(Z2)
    compute cost
        J = -sum( Y * log(A2) + (1-Y) * log(1-A2)) / m
    backward propagate
        dZ2, dW2, db2
        dZ1, dW1, db1
    update params
        W1 = W1 - learning_rate * dW1
        b1 = b1 - learning_rate * db1
        W2 = W2 - learning_rate * dW2
        b2 = b2 - learning_rate * db2
get w, b
predict
```

Deep Neural Net

```
initialize

for i in num_iter:
    forward propagate
        for l in range(1,L):
            A, cache = linear_activation_forward(A_prev, W_l, b_l)
    compute cost
        J = -sum( Y * log(AL) + (1-Y) * log(1-AL)) / m
    backward propagate
        dAL = - (np.divide(Y, AL) - np.divide(1 - Y, 1 - AL))
        for l in reversed(range(L-1)):
            dA_prev, dW, db = linear_activation_backward(dA_prev)
    update params
        for l in range(L):
            W_l = W_l - learning_rate * dW_l
            b_l = b_l - learning_rate * db_l
get all w, b
predict
```