Neural Network And Deep Learning

* Week 2.

→ Binary Classification

Logistic regression is an algorithm for binary classification.

In binary classification our goal is to learn a classifier that can input an image represented by a feature vector x and predict the corresponding label y is 1 or 0.

For example, if the image is of a cat, y will be 1 if it's a cat & else it will be 0.

→ Notation (Logistic regression):

(I) A training example is denoted by a pair (x, y) where, $x \in \mathbb{R}^{n_x}$ (n_x-dimensional feature vector)

$y \in \{0, 1\}$ (Label)

(II) m (training examples) : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

m_{train} : to emphasize the number of training examples

m_{test} : testing examples.

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad n_x, \quad \leftarrow m \rightarrow$$

$X \in \mathbb{R}^{n_x \times m}$

X.shape = (n_x, m)

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix} \quad Y \in \mathbb{R}^{1 \times m}$$

Y.shape = $(1, m)$.

→ Logistic Regression:

Given x, want $\hat{y} = P(y = 1 | x)$

$\hat{y}$ ⇒ estimate of y, probability of the chance that y is equal to one given the input features x.

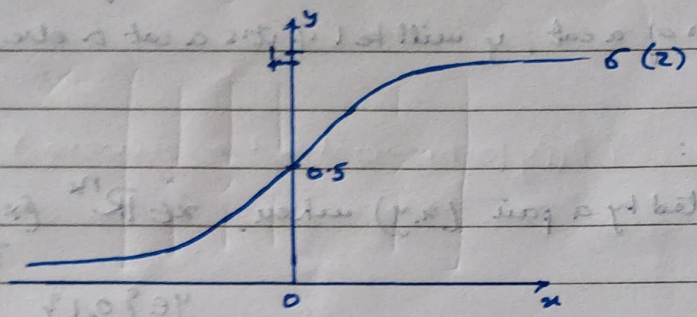
$x \in \mathbb{R}^{n_x}$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

Output $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$ ($w^T \Rightarrow$ transpose of matrix w)

↓ Sigmoid

$\hat{y}$



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

if z large (+ve)

$\sigma(z) \approx \frac{1}{1+0} = 1$.

if z large (-ve)

$\sigma(z) \approx \frac{1}{1+\text{Bignumber}} \approx 0$

→ To train the parameters, we need to define a cost function.

$\hat{y}^{(i)} = \sigma(\underbrace{w^T x^{(i)} + b}_{z^{(i)}})$, where $\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$

Given $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$ (training set values)

↑ output

Loss (error) function: $L(\hat{y}, y)$

$L \Rightarrow$ how good the output $\hat{y}$ is when the true label is y.

↑ function used to measure how accurate the output is.

$$L(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$$

If $y=1$: $L(\hat{y}, y) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ large (max 1)

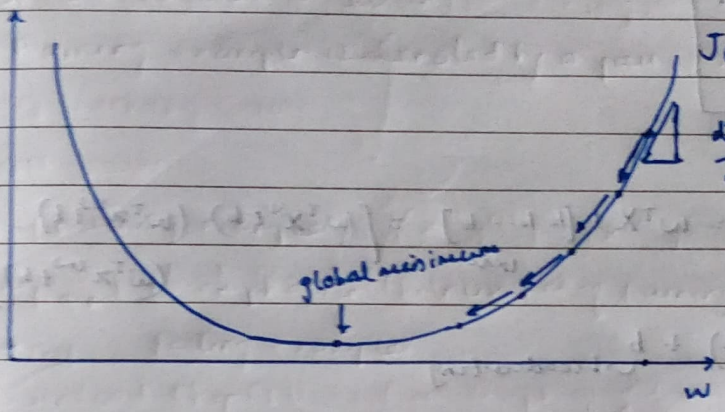
If $y=0$: $L(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log(1-\hat{y})$ large, want $\hat{y}$ small (min 0).

Loss function is defined for a single training example, cost function is defined for a set of training examples (cost of the parameters).

Cost Function: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$

→ Since we want the cost function to be minimum, we find w, b which minimize the cost function

For this we use the concept of gradient descent.



Repeat:

$\{ w := w - \alpha \frac{dJ(w)}{dw} \}$

update — learning rate

$w := w - \alpha\, dw$

↓ represents the derivative term

$J(w,b) \rightarrow$

$w := w - \alpha \frac{\partial(J(w,b))}{\partial w}$

$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$

} for parameters w and b is the cost fn

→ Computation graph

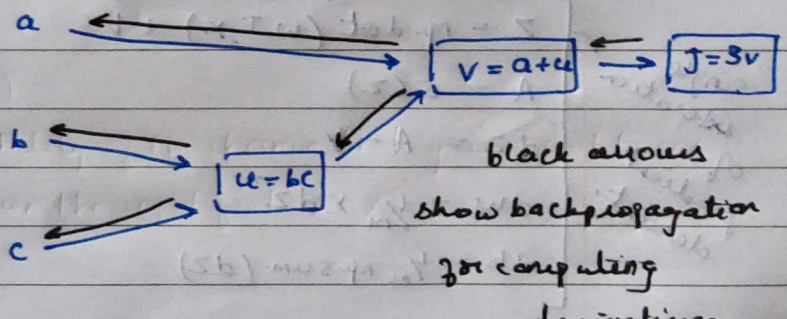
Example:

$J(a,b,c) = 3(a+bc)$

$u = bc$

$v = a+u$

$J = 3v$



black arrows show backpropagation for computing derivatives

→ Vectorization: It is the art of removing explicit for loops from your code. In deep learning code, if there are more for loops, it slows down the program hence decreasing efficiency, so vectorization is used.

In Python: np.dot(w,b) → vectorization.

→ Vectorising logistic regression:

$$X = \begin{bmatrix} | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \dots & x^{(m)} \\ | & | & | & & | \end{bmatrix}$$

$$Z = [z^{(1)}\ z^{(2)} \dots z^{(m)}] = w^T X + [b\ b \dots b]_{1\times m} = [(w^T x^{(1)} + b)\ (w^T x^{(2)} + b) \dots (w^T x^{(m)} + b)]$$

In Python: np.dot(w.T, X) + b → broadcasting

→ Vectorising logistic regression's gradient descent.

$$dZ = [dz^{(1)}\ dz^{(2)} \dots dz^{(m)}]_{1\times m}$$

$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$$

in python: db = 1/m np.sum(dz).

$$dw = \frac{1}{m} X dZ^T$$

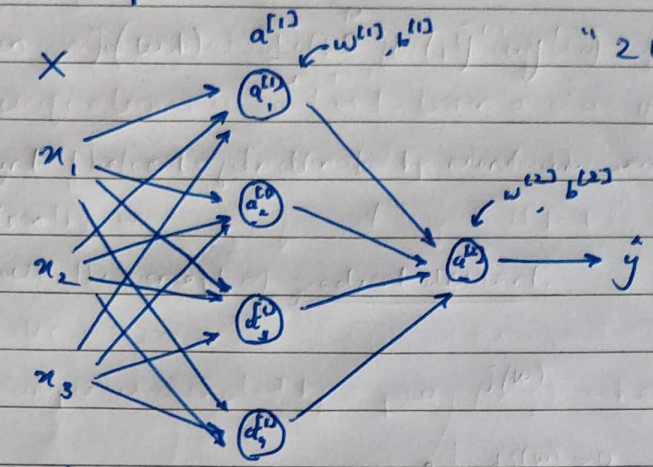
→ Implementing logistic Regression:

single iteration of gradient descent.

$$\begin{bmatrix} Z = np.dot(w.T, X) + b & w := w - \alpha\, dw \\ A = \sigma(Z) & b := b - \alpha\, db \\ dZ = A - Y & \\ dw = \frac{1}{m} X dZ^T & \\ db = \frac{1}{m} np.sum(dZ) & \end{bmatrix}$$

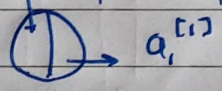
★ Week 3

→ Neural Network Representation:



"2 layer NN" (as input layer is not counted)

- Considering the first node of the hidden layer:



$$z_1^{[1]} = w_1^{[1]T} X + b_1^{[1]}$$

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$a_i^{[l]}$ → layer, i → node in layer

→ Formulas for computing derivatives

- Forward Propagation

$$z^{[1]} = w^{[1]} X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = w^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$$

Back Propagation

$$dz^{[2]} = A^{[2]} - Y \qquad Y = [y^{(1)} y^{(2)} \dots y^{(m)}]$$
$$dw^{[2]} = \frac{1}{m} dz^{[1]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis=1, keepdims=True)$$
$$dz^{[1]} = \underbrace{w^{[2]T} dz^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]'}(z^{[1]})}_{(n^{[1]}, m)}$$

→ element wise

$$dw^{[1]} = \frac{1}{m} dz^{[1]} X^T$$
$$db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1, keepdims=True)$$