→ Neural Network

→ linear regression

→ ReLU max of 0

⇒ Supervised Learning - - - → Standard NN

CNN

RNN

Custom hybrid NN

Structured Data Unstructured

↓ ↓

Table Audio Image Text

Sigmoid → Relu

→ Logistic reg is for binary classification (0 or 1)

→ feature vector (x) n_x = dim of input feature

→ m training examples

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

↳ notation for NN logistic reg

↳ stacked in columns → better to run

→ Logistic Regression.

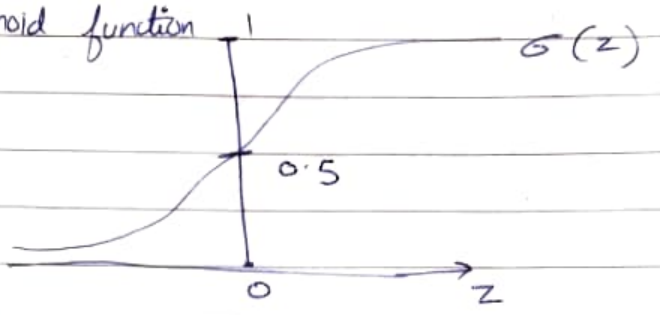
Given $x \rightarrow \hat{y} = P(y=1|x)$

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

(n_x dim vector)

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

Sigmoid function



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

→ Loss function. → only for a single training example

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

→ Cost → entire training set

$$\frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \Sigma \dots$$

→ Gradient descent

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

→ Logistic regression derivatives

$\boxed{dz = a - y}$ $da = \frac{dL}{da} = \frac{-y}{a} + \frac{1-y}{1-a}$

$\frac{dL}{dw_1} = dw_1 = x_1 dz$

$dw_2 = x_2 dz$; $db = dz$

$dz = a - y \quad ; \quad a = \sigma(z)$

$w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

→ The feature vector x has the dimension ~~$n \times 1$~~ $n \times 1$

→ The feature matrix x has the dimension $n \times m$

→ $\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_1} L(a^{(i)}, y^{(i)})$

$dw_1^{(i)} - (x^{(i)}, y^{(i)})$

→ Vectorization → to get rid of for loop

→ Vectorizing Logistic Regression

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

$= w^T x^{(1)} + b + \cdots$

Broadcasting in Python

$A = [a^{(1)} a^{(2)} \cdots a^{(m)}] = \sigma(z)$

Same for $dz \rightarrow dZ = A - Y$

→ Backward → $db = \frac{1}{m} np.sum(dz)$

$dw = \frac{1}{m} x dz^T$

→ Avoid using ~~rank matrix~~

rank 1 array

↳ doesn't behave either like row or column vector

→ 1. Load the dataset
2. Initialize the parameters
3. Forward propogation, A, cost
4. Backward propogation, dw, db gradient descent
5. Then update w and b
6. Then predict A = 1 / ..
   if else conditions

*

3

→ logistic progression.

$$Z = w^T x + b \rightarrow a = \sigma(z) \rightarrow L(a, y)$$

→ Neural network

$$z^{[1]} = w^{[1]} x + b^{[1]} \rightarrow a^{[1]} = \sigma(z^{[1]}) \rightarrow z^2 = \_\_ \rightarrow a^2 \rightarrow L$$

~~2 layers~~

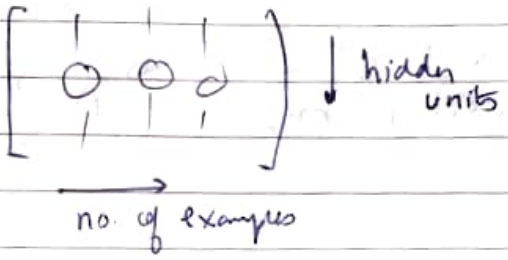
→ Two Layers

- $z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$
  $(4,1) \quad (4,3)\ (3,1) \quad (4,1)$
- $a^{[1]} = \sigma(z^{[1]})$

$$\tan(hz) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- $z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$
- $a^{[2]} = \sigma(z^{[2]})$

ReLU → $a = \max(0, z)$



$X, A^{[1]}$

→ Tanh is preferred, coz we might get mean as zero.

→ Binary → Sigmoid
Other → ReLU

→ Forward propogation.

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

→ Backward propogation.

$$dz^{[2]} = A^{[2]} - Y$$
$$dw^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis=1, keepdims=True)$$

$$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]}(z^{[1]})$$
$$(n^{[1]}, m) \quad \text{element wise product}$$

$$dw^{[1]} = \frac{1}{m} dz^{[1]} x^T$$

$$db^{[1]} = \frac{1}{m} np.sum \ldots$$

→ Deep Network

Forward propagation. $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

$Z = wX + b$

$[n^{[l]}, 1] \quad [n^{[l-1]}, 1] \quad [n^{1}, 1]$

$w^{[l]} = (n^{[l]}, n^{[l-1]})$

$b^{[l]} = (n^{[l]}, 1)$

$(dw) \simeq [w]$

$(db) \simeq [B]$

for $Z = W^{[1]}X + b^{[1]}$

$(n^{[1]}, m) \quad (n^{[1]}, n^{[0]}) \quad (n^{0}, m) \rightarrow (n^{[1]}, m)$

broadcasting.

→

1) Layer l: $w^{[l]}$, $b^{[l]}$

2) Forward: Input $a^{[l-1]}$, output $a^{[l]}$

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ Cache $z^{[l]}$

$a^{[l]} = g^{[l]} z^{[l]}$

3) Backward: Input $da^{[l]}$ output $da^{[l-1]}$

Cache dw

db