Page No.
Date

Basics of neural network programming

Week 2 : Course 1

Binary classification

eg: we input an image from which we need to identify if it contains a cat (1) or non cat (0)

Any image is stored in a computer as three seperate matrices corresponding to RGB channels of that particular image. eg: if the input image is 64×64 pixels then there will be three, 64×64 pixel matrices which will correspond to the pixel intensity values.
Then these are fed into an input feature vector **x**.

i.e. $x = \begin{bmatrix} 255 \\ 302 \\ \vdots \\ 101 \\ 105 \\ \vdots \\ 209 \\ 212 \\ \vdots \end{bmatrix}$

$64 \times 64 \times 3 = 12288$

$\therefore n = n_x = 12288$
↳ dimension of input feature X
(for brevity use n)

In short,

$\cancel{x} \longrightarrow 1/0$
(input image) (output) cat / no cat

Notation:

~~data~~ (x, y) $\quad x \in R^{n_x}, \; y \in \{0, 1\}$

A training example is represented by a pair (x, y) where, x is an n-dimentional feature vector and y is either 1 or 0.

The training set consists of ~~n~~ m total examples. ∴ The training sets, will be written as ~~(x1, y1), (x2, y2), ..., xm, ym)~~

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

$m = M_{train}$, M_{test} = # test examples

$$X = \begin{array}{c} \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \updownarrow n_x \\ \longleftarrow m \longrightarrow \end{array}$$

i.e. $X \in R^{n_x \times m}$

$X.\text{shape} = (n_x, m)$

$$Y = [y^{(1)}, y^{(2)}, \dots y^{(m)}]$$

i.e. $Y \in R^{1 \times m}$

$Y.\text{shape} = (1, m)$

Logistic Regression

→ Used when output is either 1 or 0 i.e. in case of binary classification.

Given input feature vector x, we want an algorithm that outputs a prediction ($\hat{y}$) i.e. probability of $y = 1$ given the input feature x.

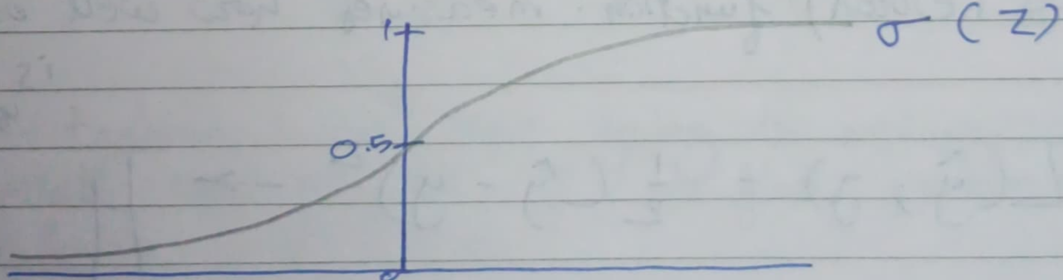
$$\therefore \hat{y} = P(y=1 \mid x) \qquad 0 \le \hat{y} \le 1$$

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

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

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

Only considering "$w^T x + b$" would not make sense to probability as it would result in a very big number or negative number. Hence, we pass it to sigmoid function which will take care that our output is between 0 & 1.



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

Page No.
Date

If z is large, $\sigma(z) \approx \frac{1}{1+0} = 1$

If z is large negative number,

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+\text{Big no}} \approx 0$$

'b' corresponds to inter-spectrum.

To train w & b, we need to define cost f^n.

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

where, $z^{(i)} = w^T x^{(i)} + b$

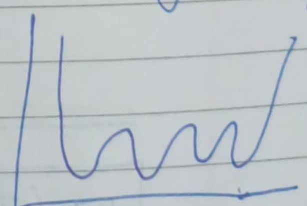
Given $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$,

we want $\hat{y}^{(i)} \approx y^{(i)}$

☆ $x^{(i)}$, $y^{(i)}$, $z^{(i)}$ denote training data's ith example

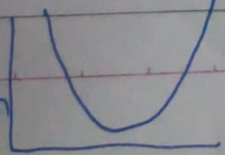
Loss (error) function: measures how well our algorithm is doing on single example

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2 \rightarrow$$



local minima is not well defined.

convex function



→ This would be preffered.

Page No.
Date

Hence we do not use the above function.

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

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

we want $\log \hat{y}$ large i.e. $\hat{y}$ to be large (as close to 1)

If $y=0$: $L(\hat{y}, y) = -\log(1-\hat{y})$

we want $\log(1-\hat{y})$ large i.e. $1-\hat{y}$ to be large $\therefore$ $\hat{y}$ close to 0.

Cost function : how well the algorithm is doing on entire training set.

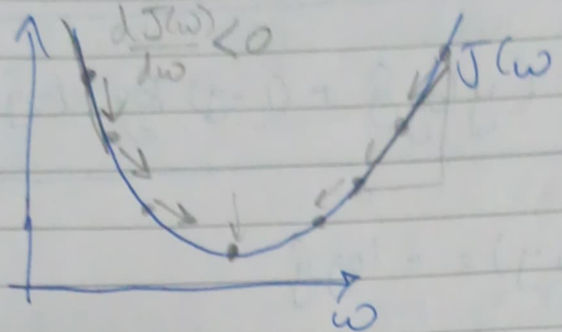
$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right]$$

In training, we are going to minimize ~~to~~ overall cost function J to find parameters w & b.

Gradient descent algorithm is used for it.

Gradient Descent:



Repeat {

learning rate

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

}

this derivate term in code is written as "dw"

$J(w, b)$

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

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

actual formulae

partial derivative

written as "db" in code

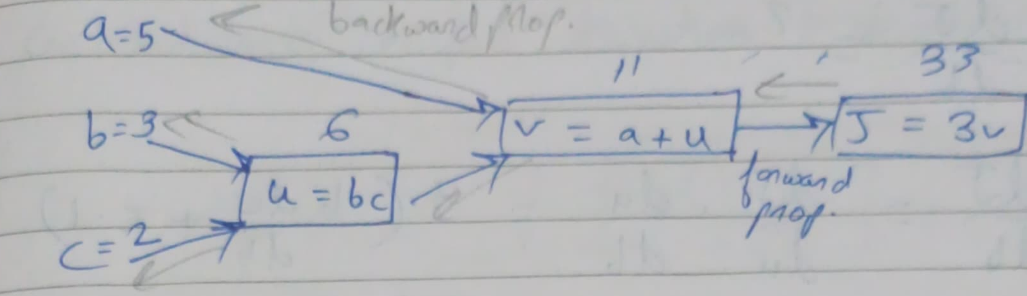
Computation graph:

$$J(a,b,c) = \underbrace{3\underbrace{(a + \underbrace{bc}_{u})}_{v}}_{J}$$

$u = bc$

$v = a + u$

$J = 3v$



Note: one step of backward propogation yields the derivative of final output

~~dJ~~
Computing derivates:

(prev example)

$\frac{dJ}{dv} = ?$ $\qquad J = 3v$ $\qquad \frac{dJ}{du} = 3 = \underbrace{\frac{dJ}{dv}}_{3} \cdot \underbrace{\frac{dv}{du}}_{1}$

$\therefore \frac{dJ}{dv} = 3$

$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \cdot \frac{dv}{da}$ ... (chain rule)

$\frac{dv}{da} = 1$

$\boxed{\frac{d \text{ Final Output Var}}{d\text{var}}}$ → "dvar" in code

similarly, $\frac{dJ}{dv}$ → "dv" in code.
$= 3$

"da" = 3

$$\frac{dJ}{db} = ?$$

$$\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db} = 3\left(b\frac{dc}{db} + c \cdot 1\right)$$

$$= 3(0 + (2)(1))$$

$$= 3 \times 2$$

$$= 6$$

$\therefore$ "db" $= 6$

$||^{ly}$ $\quad \frac{dJ}{dc} = dc = \cancel{6}\ 9 \quad \{\text{Since } b = 3\}$ $\quad \therefore 3 \times 3 = 9$

* In ~~linear~~ logistic regression, to compute final output go from left to right and to find derivatives go from right to left.

Gradient descent for logistic regression :

$$z = w^T x + b$$
$$\hat{y} = a = \sigma(z)$$
$$L(a, y) = -(y \log a + (1-y)\log(1-a))$$

$$\left.\begin{matrix} x_1 \\ x_2 \\ w_1 \\ w_2 \\ b \end{matrix}\right\} \rightarrow \boxed{z = w_1 x_1 + w_2 x_2 + b} \rightarrow \boxed{\hat{y} = a = \sigma(z)} \rightarrow \boxed{L(a,y)}$$

$dz = \frac{dL}{dz} = \frac{dL(a,y)}{dz}$ $\quad$ "da" $= \frac{dL(a,y)}{da}$

Go backwards i.e. find derivatives to decrease loss f^n.

$$da = \frac{dL(a,y)}{da} = -\left[\frac{y}{a} - \frac{(1-y)}{1-a} \right] = \frac{-y}{a} + \frac{1-y}{1-a}$$

$$dz = \frac{dL}{dz} = \frac{dL}{da} \cdot \frac{da}{dz} = \left(\frac{-y}{a} + \frac{1-y}{1-a} \right) \cdot \frac{d}{dz}\left(\frac{1}{1+e^{-z}} \right)$$

$$= \left(\frac{-y}{a} + \frac{1-y}{1-a} \right) \cdot \left(\frac{+e^{-z}}{(1+e^{-z})^2} \right)$$

$$= \left(\frac{-y}{a} + \frac{1-y}{1-a} \right) \left(\frac{\frac{1}{a} - 1}{1/a^2} \right)$$

$$= \frac{-y(1-a) + a(1-y)}{\cancel{a(1-a)}} \cancel{\left(\frac{1-a}{a} \right) a^2}$$

$$= -y + ay + a - ay$$

$$= a - y$$

$$\cancel{\therefore dz = \left(\frac{-y}{a} + \frac{1-y}{1-a} \right) a - y = a(1-a)}$$

$$\cancel{dL} \; \frac{\partial L}{\partial w_1} = \text{"}\partial w_1\text{"} = x_1 \cdot dz$$

$$\frac{\partial L}{\partial w_2} = \text{"}\partial w_2\text{"} = x_2 \cdot dz \quad , \quad db = dz$$

Page No.
Date

Then perform these updates:

$\omega_1 := \omega_1 - \alpha \, d\omega_1$

$\omega_2 := \omega_2 - \alpha \, d\omega_2$

$b := b - \alpha \, db$

Logistic regression for m examples:

Vectorization : the art of getting rid of emplicit for loops in code

$z = \omega^T x + b$

$\omega = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$ $x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$

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

Non-vectorized :

```
z = 0
for i in range n_x :
    z += w[i] * x[i]
    ~~z += b~~
z += b
```

vectorized:

$z = \underbrace{np.dot(\omega, x)}_{\omega^T x} + b$

import numpy as np then execute above code.

* vectorized code also runs faster hence takes less time to execute.

Page No.
Date

examples of vectorization:

* whenever possible, avoid explicit for-loops in logistic regression.

eg: compute $u = Av$

Non-vectorized:

$u_i = \sum_j A_{ij} v_j$

```
u = np.zeroes((n, 1))
for i ...
    for j ...
        u[i] += A[i][j] * v[j]
```

Vectorized:

```
u = np.dot(A, v)
```

eg: $v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$, compute $u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$

Non-vectorized:

```
u = np.zeroes((n, 1))      // initializing zero vector of n-dimension
for i in range(n):
    u[i] = math.exp(v[i])
```

Vectorized:

```
import numpy as np
u = np.exp(v)
```

Page No.
Date

Vectorizing logistic Regression:

$z^{(1)} = \omega^T x^{(1)} + b$ $\quad$ $z^{(2)} = \omega^T x^{(2)} + b$ $\quad$ $z^{(3)} = \omega^T x^{(3)} + b$

$a^{(1)} = \sigma(z^{(1)})$ $\quad$ $a^{(2)} = \sigma(z^{(2)})$ $\quad$ $a^{(3)} = \sigma(z^{(3)})$

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

$\omega^T \to$ row vector according to rules of matrix multiplication

$\to (1 \times m)$

$$Z = [z^{(1)} \; z^{(2)} \ldots z^{(m)}] = \omega^T X + [b \; b \ldots b]$$

$$= [\omega^T x^{(1)} + b \quad \omega^T x^{(2)} + b \;\cdots\; \omega^T x^{(m)} + b]$$

python code: ~~z~~Z = np.dot (w.T, X) + b

↳ when we import the lib. numpy as np, we can use the f^n dot to multiply matrices and use T to find transpose of matrix. Also, note that b is actually a $1 \times m$ vector (m-dimensional row vector) which is written as a real number. But python is smart enough to convert it to a vector when added to ω^T ~~x~~X. This is called "Broadcasting"

$$A = [a^{(1)} \; a^{(2)} \ldots a^{(m)}] = \sigma(Z)$$

Vectorizing LR's gradient computation:

Page No.
Date

$dz^{(1)} = a^{(1)} - y^{(1)} \quad dz^{(2)} = a^{(2)} - y^{(2)} \quad \dots$

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

$A = [\, a^{(1)} \;\dots\; a^{(m)} \,] \quad Y = [\, y^{(1)} \;\dots\; y^{(m)} \,]$

$dZ = A - Y = [\, a^{(1)} - y^{(1)} \;\; a^{(2)} - y^{(2)} \;\; \dots \,]$

$dw = 0$ | $db = 0$
$dw \mathrel{+}= x^{(1)} dz^{(1)}$ | $db \mathrel{+}= dz^{(1)}$
$dw \mathrel{+}= x^{(2)} dz^{(2)}$ | $db \mathrel{+}= dz^{(2)}$
$\vdots$ | $\vdots$
 | $db \mathrel{+}= dz^{(m)}$
$dw /= m$ | $db /= m$

Vectorized:

~~db = np. sum~~

formulae: $dw = \frac{1}{m} X \, dZ^T$ $\qquad db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$

$= \frac{1}{m}$ np.dot(X, dZ.T) $\qquad = \frac{1}{m}$ np.sum(dZ)

$$= \frac{1}{m} \begin{bmatrix} | & | & \\ x^{(1)} & x^{(2)} & \dots \\ | & | & \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ dz^{(2)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} \left[x^{(1)} dz^{(1)} + \dots + x^{(m)} dz^{(m)} \right]^{n \times 1}$$

Implementing logistic Regression using vectorization

$J=0,\ dw_1=0,\ dw_2=0,\ db=0$

for $i=1$ to m:

$z^{(i)} = w^T x^{(i)} + b$ — 1

$a^{(i)} = \sigma(z^{(i)})$ — 2

$J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$ — 3

$\left.\begin{aligned} dw_1 &+= x_1^{(i)} dz^{(i)} \\ dw_2 &+= x_2^{(i)} dz^{(i)} \end{aligned}\right\}$ 4

$db += dz^{(i)}$ — 5

$J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m$
$db = db/m$

vectorization:

1. $Z = w^T X + b$
   $= np.dot(w.T, X) + b$
2. $A = \sigma(Z)$
3. $dZ = A - Y$
4. $dw = \frac{1}{m} X . dZ^T$
5. $db = \frac{1}{m} np.sum(dZ)$

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

Page No.
Date

Broadcasting example:

ex 1) $\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + 100 \Rightarrow \begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \begin{bmatrix}100\\100\\100\\100\end{bmatrix} = \begin{bmatrix}101\\102\\103\\104\end{bmatrix}$

ex 2) $\underset{2\times3}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} + \underset{1\times3}{\begin{bmatrix}100 & 200 & 300\end{bmatrix}} = ?$

$1\times3 \rightarrow 2\times3$

$\Rightarrow$

$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix} = \begin{bmatrix}101 & 202 & 303\\104 & 205 & 306\end{bmatrix}$

ex 3) $\underset{2\times3}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} + \underset{2\times1}{\begin{bmatrix}100\\200\end{bmatrix}} = ?$

$2\times1 \rightarrow 2\times3$

$\Rightarrow \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 100 & 100\\200 & 200 & 200\end{bmatrix} = \begin{bmatrix}101 & 102 & 103\\204 & 205 & 206\end{bmatrix}$

General principle:

(m,n) matrix $\quad +, -, *, / \quad (1,n) \rightsquigarrow (m,n)$ matrix

$(m,1) \rightsquigarrow (m,n)$ matrix

$(m,1) + R$

$\begin{bmatrix}1\\2\\3\end{bmatrix} + 100 = \begin{bmatrix}101\\102\\103\end{bmatrix}$

Page No.
Date

Python/numpy vectors

a = np.random.randn(5)

→ rank 1 array
(neither a row vector nor column vector)
a.shape = (5,)

} Don't use to implement logistic regression

a = np.random.randn(5,1) → a.shape = (5,1) ✓
∴ column vector
a = np.random.randn(1,5) → a.shape = (1,5) ✓
∴ row vector