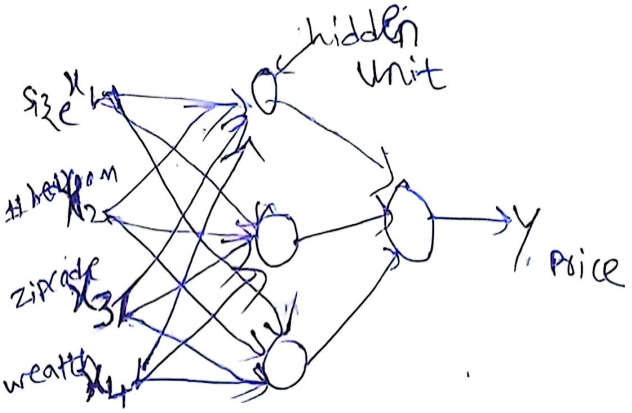
Deep learning coursera course

→ Neural network comprises an input layer, and an output layer

→ Deep learning is made up of several hidden layers of neural networks that perform complex operations on massive amounts of structured and unstructured data



What you'll learn

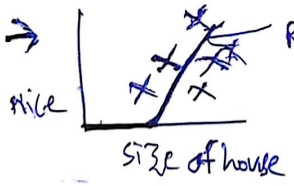
1. Neural networks and Deep learning
2. Improving Deep Neural Networks: Hyper Parameter tuning, Regularization and Optimization.
3. Structuring your ML Project
4. Convolution Neural Networks
5. NLP : Building sequence models.

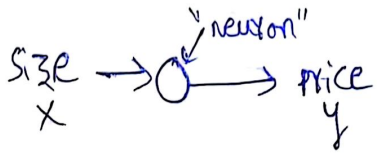
Supervised learning

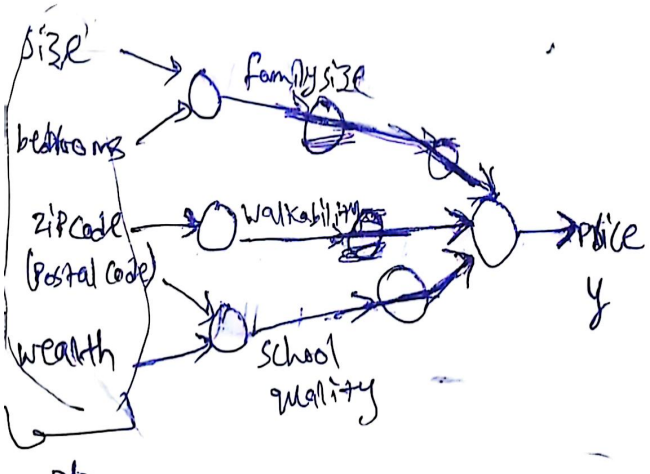
Input (x)	o/p (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on add (0/1)	online Advertising
Image	object (1, ..., 1000)	Photo tagging
Audio	text transcript	speech recognition
English	chinese	machine translation
Image, Radar info	position of other cars	Autonomous driving

standard NN
CNN
RNN
RNN
CNN
custom / Hybrid NN

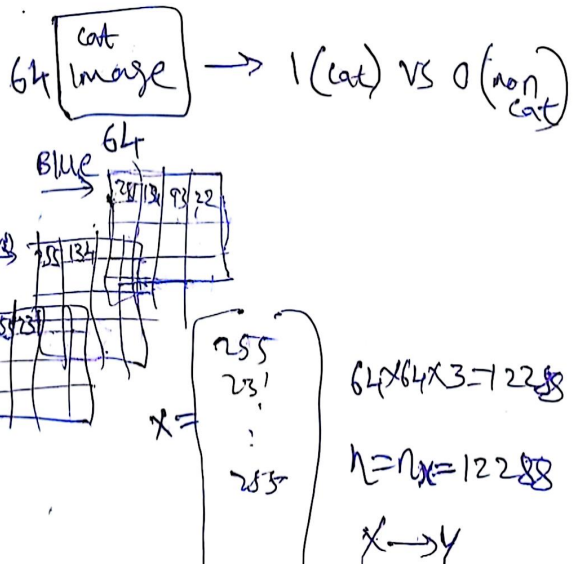
Housing price prediction







Binary classification



Notation 'm' → small m is used to represent data

(x, y) $x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

x → input, y → output

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

$M = M_{train}$

M_{test} = # test examples

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

$\longleftarrow m \longrightarrow$

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

$X.shape = (n_x, m)$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

$Y \in \mathbb{R}^{1 \times m}$

$Y.shape = (1, m)$

deals with a single data instance.

→ A cost function is a measure of how inaccurate the model is in estimating the connection b/w X and Y

→ A loss function is a function that compares the target & predicted o/p values; measures how well the neural network models the training data. When training, we aim to minimize this loss b/w the predicted & target o/p's.

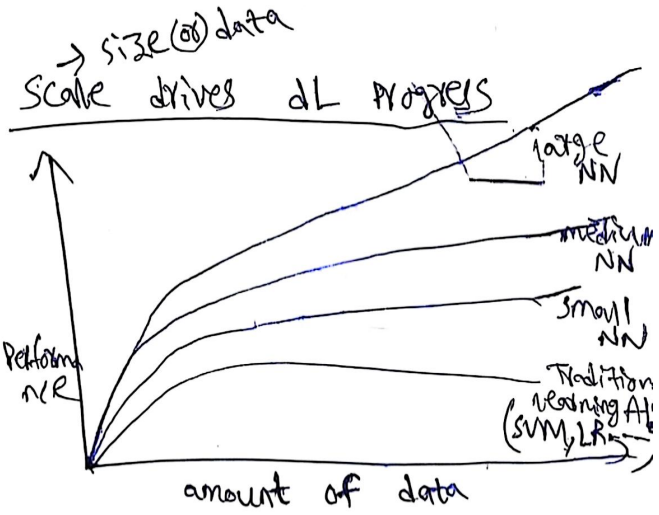
→ The cost function of a neural network will be the sum of errors in each layer. This is done by finding the error at each layer first and then summing the individual error to get the total error

Loss function & cost function

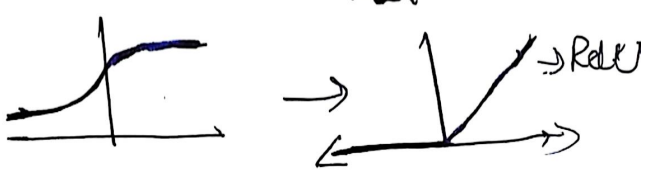
→ The loss function is to capture the difference between the actual & predicted values for a single record whereas cost functions aggregate the difference for the entire training dataset.

→ The most commonly used loss functions are mean-squared error and hinge loss.

→ The cost function measures the model's error on a group of objects, whereas the loss function



→ Gradient descent converts sigmoid function to ReLU



Logistic Regression:

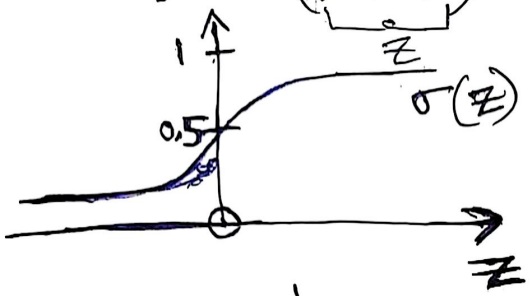
Given x, want $\hat{y} = P(y=1|x)$

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

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

$0 \le \hat{y} \le 1$

output $\hat{y} = \sigma(w^T x + b)$



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

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

If z is small (large -ve number) $\sigma(z) = \frac{1}{1+e^{-z}} \simeq \frac{1}{1+\text{big}} = 0$

$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$

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

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

Loss (error) function:

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

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

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

want $\log \hat{y}$ large, for that $\hat{y}$ should be large

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

want $\log(1-\hat{y})$ large, for that $\hat{y}$ should be small.

$x_0 = 1$, $x \in \mathbb{R}^{n_x+1}$

$\hat{y} = \sigma(\theta^T x)$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} b \\ \\ \\ \\ \} w \end{matrix}$$

Cost function:

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

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

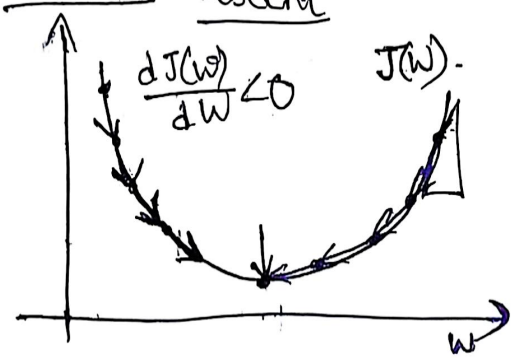
LR Cost function:

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

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

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

Gradient Descent



Repeat {

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

(learning rate)

}

In logistic Regression

$$J(w,b) \begin{cases} w := w - \alpha \frac{dJ(w,b)}{dw} \\ b := b - \alpha \frac{dJ(w,b)}{db} \end{cases}$$

More derivative examples :

$f(a) = a^2$ $\quad \frac{d}{da} f(a) = 2a$ $\quad a=2$ $f(a)=4$

$a=2.001$ $f(a) \approx 4.004$

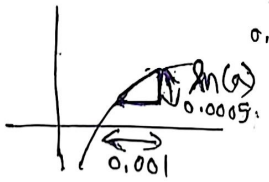
$f(a) = a^3$ $\quad \frac{d}{da} f(a) = 3a^2$ $\quad a=2$ $f(a)=8$

$a=2.001$ $f(a) \approx 8.012$

$f(a) = \log_e(a) = \ln(a)$ $\quad \frac{d}{da} f(a) = \frac{1}{a}$ $\quad a=1$ $f(a) \approx 0.6935$

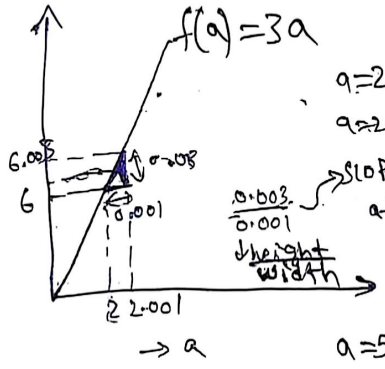
$a=2.001$ $f(a) \approx 0.6925$

0.0005 ⟷ 0.0005



$\frac{d}{da} f(a) = \frac{1}{2}$

Intuition about derivatives :



$a=2$ $f(a)=6$

$a=2.001$ $f(a)=6.003$

slope (derivative) of $f(a)$ at $a=2$ is 3

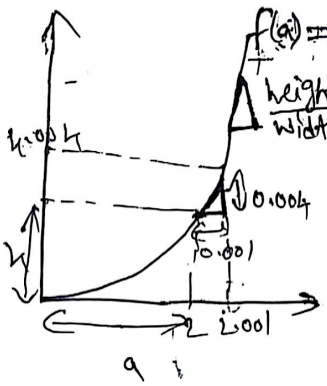
$\frac{0.003}{0.001}$ height / width

$a=5$ $f(a)=15$

$a=5.001$ $f(a)=15.003$

slope at $a=5$ is 3

$\frac{df(a)}{da} = 3 = \frac{d}{da} f(a)$



$f(a) = a^2$

$a=2$ $f(a)=4$

$a=2.001$ $f(a) \approx 4.004$ (4.004001)

slope (derivative) of $f(a)$ at $a=2$ is 4

$\frac{d}{da} f(a) = 4$ when $a=2$

$a=5$ $f(a)=25$

$a=5.001$ $f(a) \approx 25.010$

$\frac{d}{da} f(a) = 10$ when $a=5$

$\frac{d}{da} a^2 = 2a$

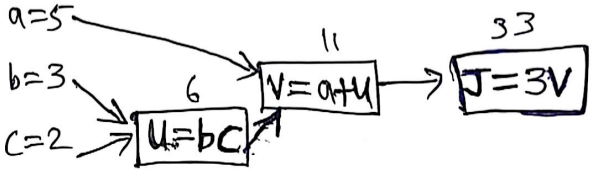
Computation Graph

$J(a,b,c) = 3(a+bc) = 3(5+3\times2) = 33$

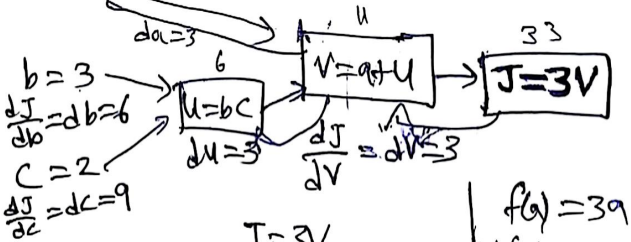
$u = bc$

$v = a+u$

$J = 3v$



Computing derivatives !



$\frac{dJ}{da} = da = 3$

$\frac{dJ}{db} = db = 6$

$\frac{dJ}{dc} = dc = 9$

$a \to v \to J$

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

$J = 3v$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$a = 5 \to 5.001$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \frac{dv}{da}$ (3 × 1)

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

$f(a) = 3a$

$\frac{df(a)}{da} = 3$

$J = 3v$

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

$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \cdot \frac{dv}{du} \quad (3 \cdot 1)$$

$$\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db} = 6 \quad (3 \cdot 2)$$

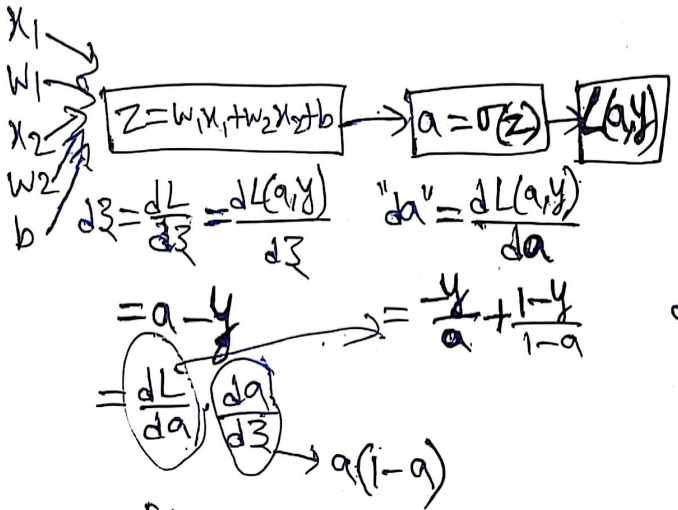
$$\frac{dJ}{dc} = \frac{dJ}{du} \cdot \frac{du}{dc} = 9$$

Logistic regression derivatives

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

$$z = w^T x + b$$

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



By multiplying above two we get $\frac{dL}{dz} = a - y = dz$

$$\frac{\partial L}{\partial w_1} = "dw_1" = x_1 \cdot dz$$

$$\frac{\partial L}{\partial w_2} = "dw_2" = x_2 \cdot dz$$

$$db = dz$$

$$w_1 := w_1 - \alpha\, dw_1$$

$$w_2 := w_2 - \alpha\, dw_2$$

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

Logistic Regression on m examples:

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

$$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

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

$$dw_1^{(i)}, dw_2^{(i)}, db^{(i)}$$

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

$$= dw_1^{(i)}$$

Logistic Regression derivatives

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

for $i = 1$ to m:

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

$$a^{(i)} = \sigma(z^{(i)})$$

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

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dw_1\ += x_1^{(i)} dz^{(i)}$$

$$dw_2\ += x_2^{(i)} dz^{(i)}$$

$$db\ += dz^{(i)}$$

$n_x = 2$

$$J /= m$$

$$dw_1 /= m;\ dw_2 /= m;\ db /= m$$

$$dw_1 = \frac{\partial J}{\partial w_1}$$

$$w_1 := w_1 - \alpha\, dw_1$$

$$w_2 := w_2 - \alpha\, dw_2$$

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

What is vectorization?

$z = w^T x + b$.

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

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

non-vectorized

```
z=0
for i in range(n-x):
    z+=w[i]*x[i]
z+=b
```

vectorized

```
z=np.dot(w,x)+b
```

($w^T x$)

one is faster vectorized version or ~~for loop~~ non-vectorized version.

Neural Network Programming guideline

→ whenever possible, avoid explicit for-loops.

non-vectorized

```
u=Av
u_i = Σ_j A_ij v_j
u=np.zeros((0,1))
for i---
    for j---
        u[i]+=A[i][j]*v[j]
```

vectorized

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

Vectors & matrix valued functions

Say you need to apply the exponential operation on every element of a matrix / vector.

$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$

non-vectorized

```
u=np.zeros((n,1))
for i in range(n):
    u[i]=math.exp(v[i])
```

vectorized

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

```
import numpy as np
import time

a=np.random.rand(1000000)
b=np.random.rand(1000000)
tic=time.time()
c=np.dot(a,b)
toc=time.time()
print(c)
print("vectorized version:"+str(1000*(toc-tic))+"ms")

c=0
tic=time.time()
for i in range(1000000):
    c+=a[i]*b[i]
toc=time.time()
print(c)
print("for loop:"+str(1000*(toc-tic))+"ms")
```

Output

```
250286.989866
vectorized version: 1.50275230 ms
250286.989866
for loop: 474.29513931 ms
```

Your program is to know which

Vectorizing logistic Regression

$z^{(1)} = w^T x^{(1)} + b$ $z^{(2)} = w^T x^{(2)} + b$ $z^{(3)} = w^T x^{(3)} + b$

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

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

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

$= [w^T x^{(1)} + b\ \ w^T x^{(2)} + b\ \ \dots\ \ w^T x^{(m)} + b]$ (1×m)

$w^T X = [\dots] \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$

$Z = np.dot(w.T, X) + b$ (b ↑ (1,1) "Broadcasting")

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

$dz^{(1)} = a^{(1)} - y^{(1)}$ $dz^{(2)} = a^{(2)} - y^{(2)}$ ...

$dz = [dz^{(1)}\ dz^{(2)}\ \dots\ dz^{(m)}]$ (1×m)

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

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

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

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

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

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

Implementing LR

$Z = w^T X + b$

$= np.dot(w.T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

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

$db = \frac{1}{m}$ np.sum(dZ)

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

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

Broadcasting example

Calories from Carbs, Proteins, fats in 100g of different foods:

	Apples	Beef	Eggs	Potatoes
Carb	56.0	0.0	4.4	68.0
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

3×4

Calculate % of Calories from Carb, Protein, fat. Can you do this without using explicit for-loop

Program)

```
import numpy as np
A = np.array([[56.0, 0.0, 4.4, 68.0],
              [1.2, 104.0, 52.0, 8.0],
              [1.8, 135.0, 99.0, 0.9]])
print(A)
```

```
cal = A.sum(axis=0)
print(cal)
```

O/P:-

[59. 239. 155.4 76.9]

```
percentage = 100 * A/cal
print(percentage)
```

O/P:-

[[94.9152 0. -- --]
 [2.0338 -- -- --]
 [3.0508 -- -- --]]

```
import numpy as np
a = np.random.randn(5)
print(a)
```

O/P:-

[0.5829 -0.2969 0.9572 -0.8212
-1.4626]

```
print(a.shape)
```

O/P:- (5,)

```
print(a.T)
```

O/P:- O/P is same as above

```
print(np.dot(a, a.T))
```

O/P:- 4.0657

Broadcasting example

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

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

$$\underset{(m,n)}{\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}} + \underset{(m,1)\rightarrow(m,n)}{\begin{bmatrix}100\\200\end{bmatrix}} = \begin{bmatrix}101&102&103\\204&205&206\end{bmatrix}$$

General principle:-

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

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

```
a = np.random.randn(5,1)
print(a)
```

O/P:-

[[-0.0967]
 [-2.3861]
 [-0.3243]
 [-0.9821]
 [0.5441]]

```
print(a.T)
```

[[-0.0967 -2.3861 ...]]

```
print(np.dot(a, a.T))
```

O/P:-

[[0.0093 0.0238 ...]
 [0.2308 ...]
 [0.0313 ...]
 [0.0950 ...]]

Packages

→ numpy is the fundamental package for scientific computing with python.

→ h5py is a common package to interact with a dataset that is stored on an H5 file.

→ matplotlib is a famous library to plot graphs in python.

Logistic regression cost function.

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

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

If $y=1$: $P(y|x) = \hat{y}$

If $y=0$: $P(y|x) = 1-\hat{y}$

} $P(y|x)$

$P(y|x) = \hat{y}^y (1-\hat{y})^{(1-y)}$

If $y=1$: $P(y|x) = \hat{y}$

If $y=0$: $P(y|x) = 1-\hat{y}$

increases

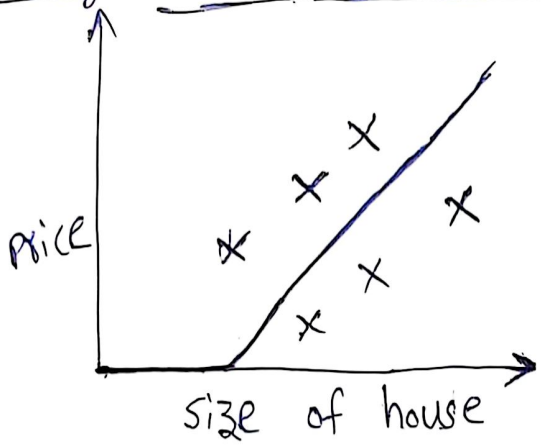
$\log P(y|x) = \log \hat{y}^y (1-\hat{y})^{(1-y)}$

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

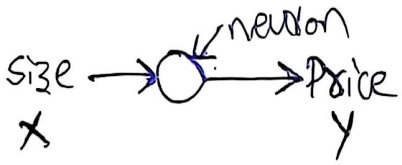
$= -L(\hat{y}, y)$ ↓ decreases.

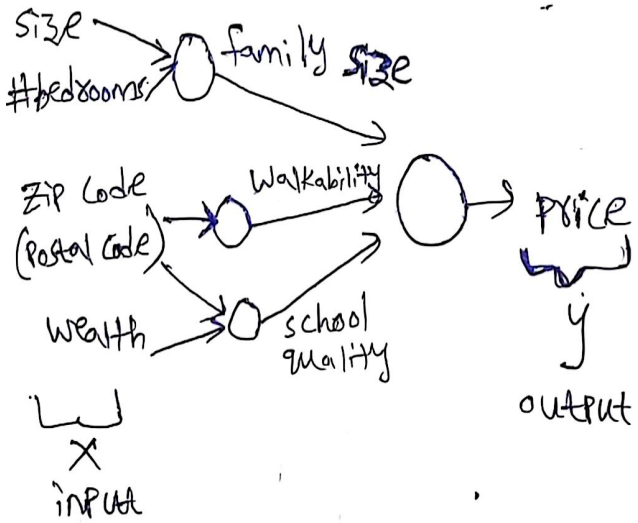
Andrew Ng Lectures!

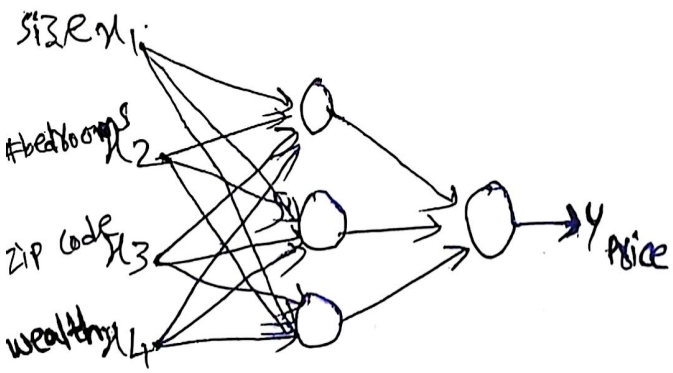
Housing Price Prediction



ReLU → Rectified Linear Unit function



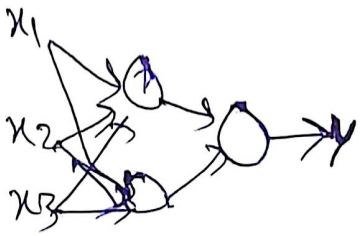




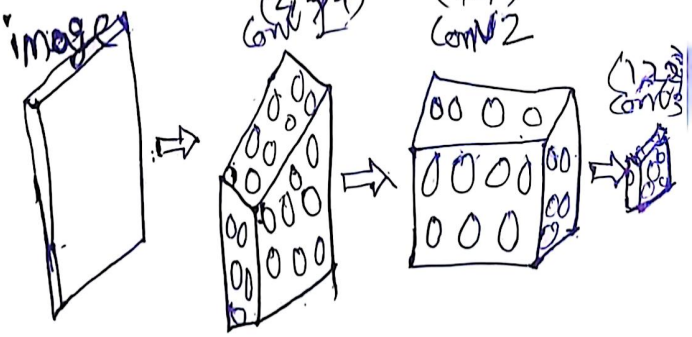
Supervised Learning

Input (x)	Output (y)	Application	
Home features	Price	Real Estate	Standard neural network
Ad, user info	Click on ad? (0/1)	Online Advertising	
Image	Object (1,...1000)	Photo tagging	CNN
Audio	Text transcript	speech recognition	
English	Chinese	Machine translation	RNN
Image, Radar info	position of other cars	Autonomous driving	Custom/ hybrid neural network

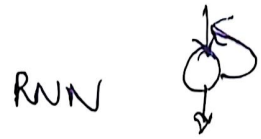
Neural Network examples



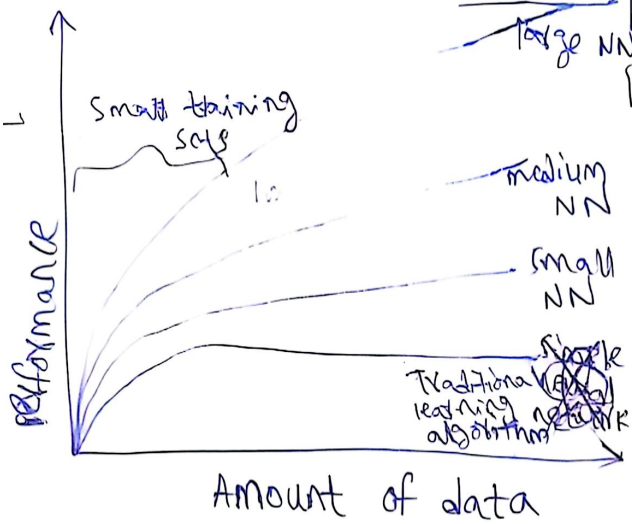
Standard Neural Network



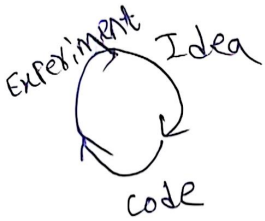
Convolutional Neural Network



Scale drives deep learning progress:



- Data
- Computation
- Algorithms



Course: Neural Networks and Deep Learning:

Week 1: Introduction

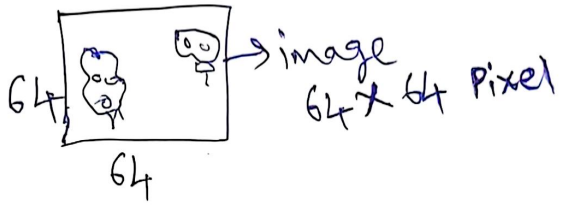
Week 2: Basics of Neural Network Programming

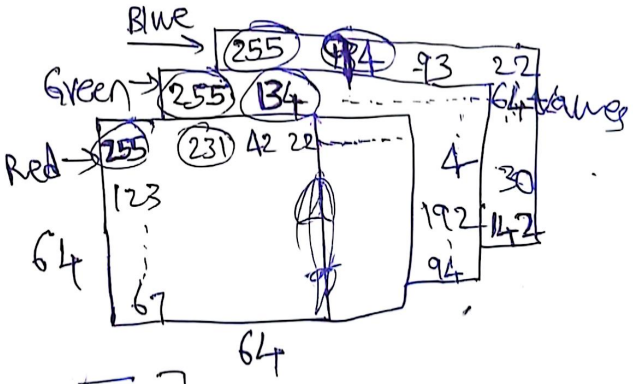
Week 3: One hidden layer Neural Networks

Week 4: Deep Neural Networks

Binary Classification

→ Logistic Regression is an algorithm for binary classification.





$$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 255 \\ 134 \\ \vdots \\ 255 \\ 94 \\ \vdots \end{bmatrix}$$

64 values belongs to Red

64 values belongs to Green

64 values belongs to Blue

$64 \times 64 \times 3 = 12288$

Courses in this Specialization:

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization.
3. Structuring your Machine Learning Project.
4. Convolutional Neural Networks
5. Natural Language Processing: Building sequence Models

→ Per one 64×64 pixels image there are three matrices (Red, Green, Blue) (RGB), each of size 64×64.

Per one image
- Red → 64×64
- Green → 64×64
- Blue → 64×64

→ The resultant feature vector contains 64×64×3 values

→ The unfold of the matrices happens like, first row of Red matrices happens and then second row and so on like that until 64th row of Red matrices and then Green matrices unfolds in the same way and then Blue matrices.

→ 64×64×3 = 12288

→ The feature vector is represented as $n_x = 12288$

dimension of input feature vector

$M = m_{train}$ $M_{test} = \#test$ examples

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

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

$X.shape = (n_x, m)$

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

$Y \in \mathbb{R}^{1 \times m}$

$Y.shape = (1, m)$

Notation

(x, y), $x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

↙ Single training example

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

Problem: Cat (1) or non-cat (0)

Logistic Regression:

Given x, want $\hat{y} = P(y=1|x)$ → $0 \le \hat{y} \le 1$

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

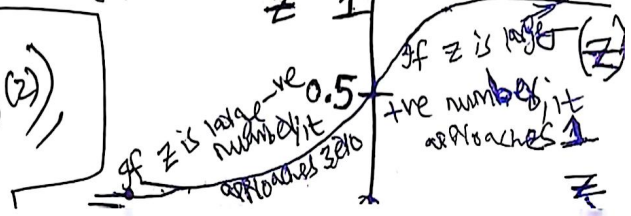
output $\hat{y} = w^T x + b$

↳ Linear regression

It shows the probability of image, either cat or non-cat

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

z



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

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

→ If z is large negative number

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+\infty} \approx 0$$

$x_0 = 1$, $x \in \mathbb{R}^{n_x+1}$

$\hat{y} = \sigma(\theta^T x)$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{n_x} \end{bmatrix}$ $\theta_0 \leftarrow b$, $\theta_1, \ldots, \theta_{n_x} \leftarrow w$

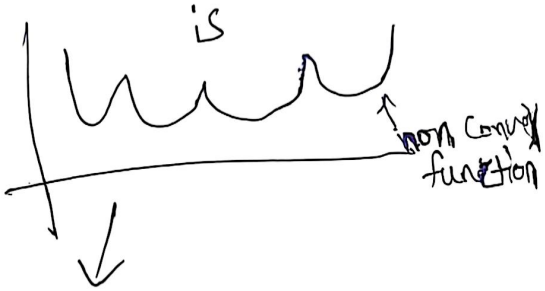
This notation is not important

Loss (error) function:

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

↳ square error

The graph of square error is



So, square error curve is not possible to find global minimum or global maximum.

→ So, in logistic Regression we use ∈ convex function

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

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

want $\log \hat{y}$ large, want $\hat{y}$ large

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

want $\log(1-\hat{y})$ large, want $\hat{y}$ small

Logistic Regression Cost function:

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

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

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

for training examples

$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, where

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

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

For ith training example, it is represented as

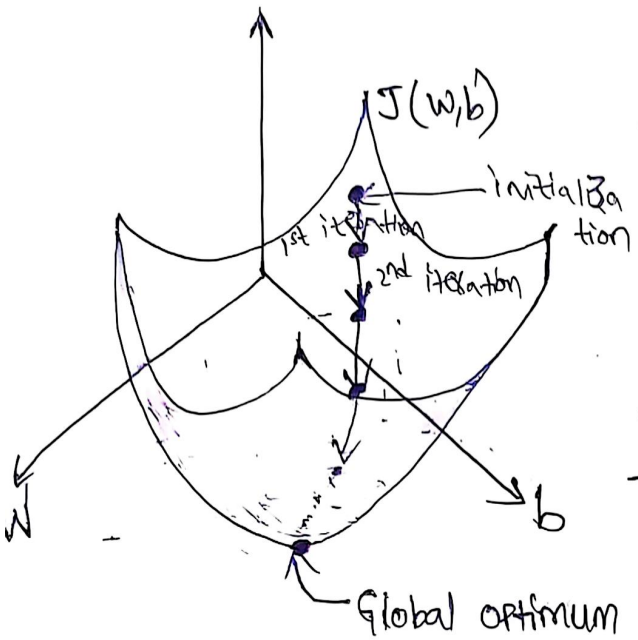
Cost Function:

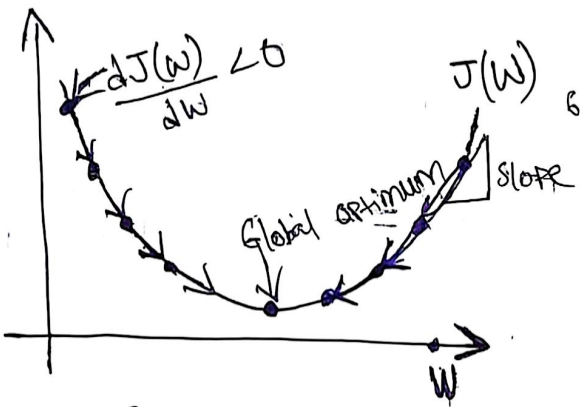
$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

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

Gradient Descent:

Want to find w, b, that minimize $J(w,b)$





Repeat {

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

(learning rate; $\frac{dJ(w)}{dw}$ = "dw")

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

$\frac{dJ(w)}{dw}$ ← Slope $= ?$

$J(w,b)$ → 'w' updates as

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

→ 'b' updates as

$$b := b - \alpha \frac{dJ(w,b)}{db}$$

$\frac{\partial J(w,b)}{\partial w} \Rightarrow dw$ → In code it is shown as 'dw'

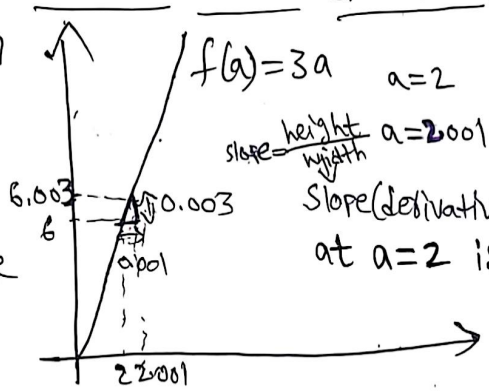
↳ This shows the slope of $J(w,b)$ in the direction of 'w'.

$\frac{\partial J(w,b)}{\partial b} \Rightarrow db$ → In code it is shown as 'db'

↳ This shows the slope of $J(w,b)$ in the direction of 'b'

15/01/2024

Intuition about derivatives



$f(a) = 3a$ $a=2$ $f(a)=6$

$a=2.001$ $f(a)=6.003$

slope = height / width

Slope (derivative) of $f(a)$ at $a=2$ is 3.

$a=5$ $f(a)=15$

$a=5.001$ $f(a)=15.003$

Slope at $a=5$ & also 3

$$\frac{d\,f(a)}{da} = 3 = \frac{d}{da} f(a)$$

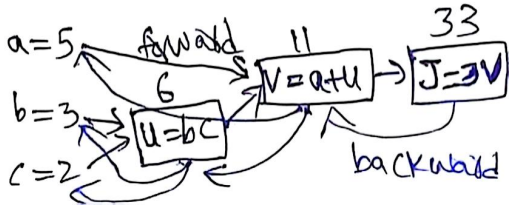
Computation Graph

$J(a,b,c) = 3(a+bc) = 3(5+3\times2) = 33$

$u = bc$

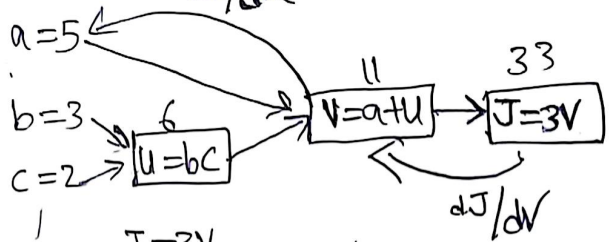
$v = a+u$

$J = 3v$



Derivatives with a Computation

Graph:



$J = 3V$

$\frac{dJ}{dV} = 3$

↓

Increase in 'V' increases 'J' (three times)

$J = 3V$

$V = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$\frac{dJ}{da} = 3 = \frac{dJ}{dV} \cdot \frac{dV}{da}$

$= 3 \cdot \frac{dV}{da}$

$= 3(1)$

$= 3$

$\frac{dJ}{da} = 3$

$\frac{dV}{da} = \frac{d(a+u)}{da}$

$= (1)$

d final output variable

↙ d variable

→ when we are implementing it in software, we can given this as "dJdvar"

→ In the code we can write it as "dvar"

$\frac{dJ}{dV} \Rightarrow dV = 3$, $\frac{dJ}{da} \Rightarrow da = 3$

$\frac{dJ}{du} = 3$

$= \frac{dJ}{dV} \cdot \frac{dV}{du}$

$= (3)(1)$

$= 3$

$u = 6 \xrightarrow[\text{to}]{\text{increases}} 6.001$

$V = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db}$ (3 · 2)

$= 6$

$b = 3 \rightarrow 3.001$

$u = bc = 6 \rightarrow 6.002$

$c = 2$

$V = u + 0.002$

$J = 33.006$

$\frac{dJ}{dc} = \frac{dJ}{du} \cdot \frac{du}{dc}$ (3 · 3)

$= 9$

18/01/2024

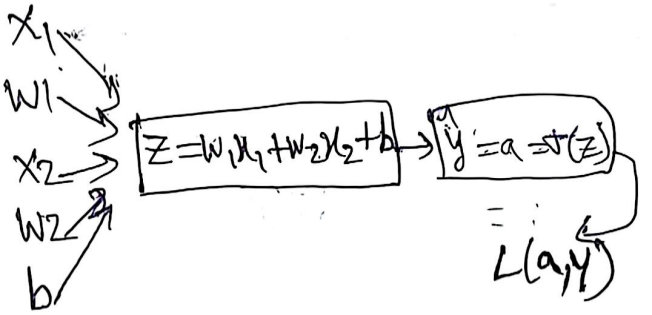
Logistic Regression Gradient Descent

Recap:

$z = w^T x + b$

$\hat{y} = a = \sigma(z)$

$L(a, y) = -(y \log a + (1-y) \log(1-a))$



"da" $= \frac{dL(a,y)}{da} = -\left(\frac{d}{da}(y \log a + (1-y) \log(1-a))\right)$

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

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

"dz" $= a - y = \frac{dL}{dz}$

$\Rightarrow \frac{dL}{da} \cdot \frac{da}{dz} \Rightarrow -\left(\frac{y}{a} - \frac{1-y}{1-a}\right) \cdot \frac{da}{dz}$

$\Rightarrow -\left(\frac{y}{a} - \frac{1-y}{1-a}\right) \cdot (a(1-a))$

$\Rightarrow \frac{da}{dz} = \frac{d}{dz}\sigma(z)$ $\left[\because a = \sigma(z) = \frac{1}{1+e^{-z}}\right]$

$= -\left(y(1-a) - a(1-y)\right)$

$= \frac{d}{dz}\left(\frac{1}{1+e^{-(z)}}\right)$

$= -\left(y - ay - a + ay\right)$

$\left[\because \frac{d}{dx}\left(\frac{1}{A}\right) = \frac{-1}{A^2}\frac{dA}{dx}\right]$

$= -(y-a)$

$= a - y$

$= \frac{-1}{(1+e^{-z})^2} \cdot \frac{d}{dz}(1+e^{-z})$

$dz = a - y = \frac{dL}{dz}$

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

$= \frac{e^{-z}}{(1+e^{-z})^2}$

At final step to adjust weights and 'b'

$\frac{\partial L}{\partial w1} = dw1 = x_1 \cdot dz$

$dw2 = x_2 \cdot dz$

$db = dz$

Add 1 to numerator & subtract 1 to numerator

$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$ [split numerator]

$w1 := w1 - \alpha dw1$

$w2 := w2 - \alpha dw2$

$b := b - \alpha db$

$= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2}$

$= \frac{1}{(1+e^{-z})} - \frac{1}{(1+e^{-z})^2}$

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

$= a(1-a)$

$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w1}$

$= \left(\frac{-y}{a} + \frac{1-y}{1-a}\right) \cdot (a(1-a)) \cdot \frac{\partial z}{\partial w1}$

$= (a-y) \cdot \frac{\partial z}{\partial w1}$

$= dz \cdot \frac{\partial (w_1 x_1 + w_2 x_2 + b)}{\partial w1}$

$= dz \cdot x_1$

$\frac{\partial L}{\partial w1} = x_1 \cdot dz$

Gradient Descent on m examples:

Logistic Regression on m examples:

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

$$\rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

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

$dw_1^{(i)}$ — $(x^{(i)}, y^{(i)})$

Lets initialize

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

for $i=1$ to m

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

$a^{(i)} = \sigma(z^{(i)})$

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

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} \cdot dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} \, dz^{(i)}$

$dw_n \mathrel{+}= x_n^{(i)} \cdot dz^{(i)}$ (features)

$db \mathrel{+}= dz^{(i)}$

So, we are calculating averages we need to divide each with 'm'

$J /= m, \; dw_1 /= m;$

$dw_2 /= m \; ; \; db /= m$

$$dw_1 = \frac{\partial J}{\partial w_1}$$

$w_1 := w_1 - \alpha \, dw_1$

$w_2 := w_2 - \alpha \, dw_2$

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

Vectorization:

→ It is used to get rid of extra for loops in the code

What is vectorization?

$$z = w^T x + b$$

$$w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$$

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

non-vectorial implementation

```
z=0
for i in range (n-x):
    z += w[i] * x[i]
z += b
```

Vectorial implementation

```
z = np.dot(w,x) + b
```

$w^T x$

Neural Network Programming Guideline

Note:

Whenever possible, avoid explicit for-loops.

Ex:-

$u = A.V$

$u_i = \sum_j A_{ij} V_j$

Non-vectorized implementation

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

Vectorized implementation:

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

Vectors and matrix valued functions

```
1) import numpy as np
   import time

a = np.random.rand(1000000)
b = np.random.rand(1000000)
tic = time.time()
c = np.dot(a,b)
toc = time.time()
print(c)
Print("vectorized version:" + str(1000*(toc-tic)) + "ms")

c = 0
tic = time.time()
for i in range(1000000):
    c += a[i]*b[i]
toc = time.time()
Print(c)
Print("for loop:" + str(1000*(toc-tic)) + "ms")
```

→ Say

O/P)

```
249660.39691457726
vectorized version: 1.5258789062 5ms
249660.39691457726
for loop: 1200.25777--ms
```

→ In above code vectorized method takes less time to compute compared to non-vectorized method.