Course 1: Neural Networks and Deep Learning.

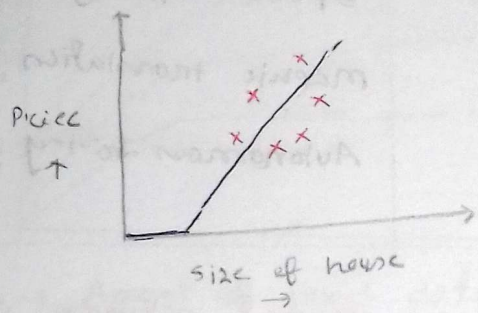
Started Date: 30-11-2018 3:20 PM.

week 1: Introduction to Deep Learning.

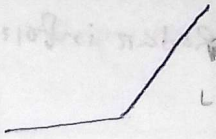
L-1: what is neural network. 11:05 Am (1-12-18)

Neural network is a powerful learning algorithm inspired by how the brain works. Deep learning refers to train neural network sometimes very large neural network.

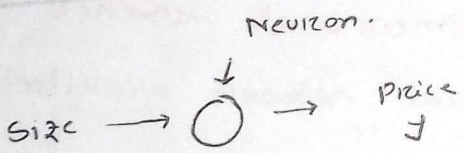
Housing price prediction.



As price can never be negative so we are creating a function called rectified Linear unit (ReLU).



Simple Network.

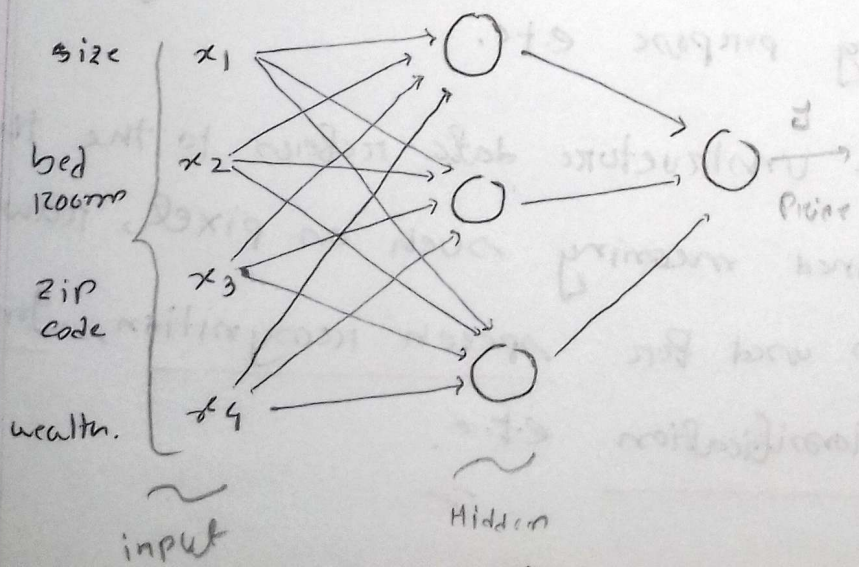


x (input) → size of house.

y (output) → Price.

The neuron implements the function ReLU.

Large network.



magic of neural network is that it automatically generate the hidden units we only need to give input x and output y.

L-2: Supervised Learning with Neural Networks (11:35 Am 01-12-18)

In supervised learning, we are given a dataset and already know what our correct output should look like. Some Examples.

Input (x)	output (y)	Application.	
Home features.	Price	Real Estate.	Standard NN
Ad, user info.	click on Ad? (0/1)	online Advertising.	Standard NN
Image	object detection.	photo tagging.	CNN
Audio	Text transcript	speech recognition	RNN
English.	chinese	machie translation	RNN
Image, Radar info.	position of other cars.	Autonomous driving.	complex RNN

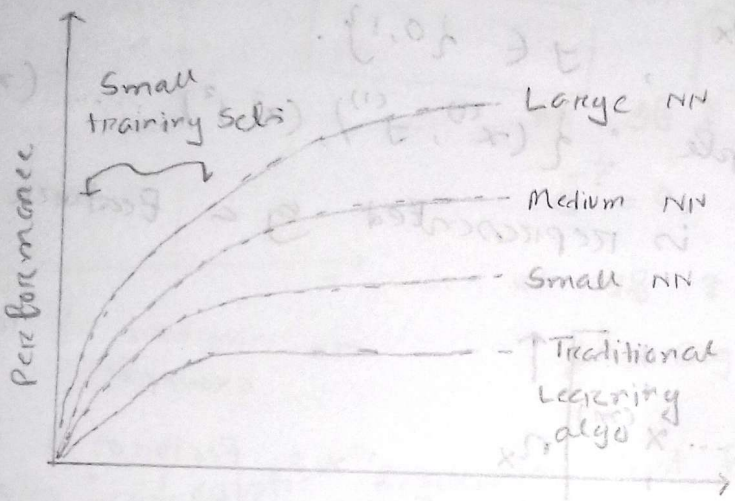
** standard NN used for standard neural network application.
** CNN used for image application.
** RNN used for one dimensional sequence data and temporal element. { Hybrid neural network architecture is also used }

Structure data :- Structure data refers to the things that has a defined meaning such as price, age. e.t.c. Structure data in database of data. It is used for housing price Prediction, advertising purpose e.t.c.

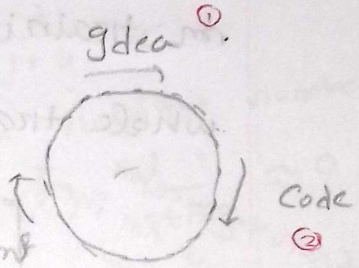
unstructure data :- unstructure data refers to the things that has not defined meaning such as pixel, raw audio, text. It is used for speech recognition, Image recognition, Text classification e.t.c.

L 3 :- why Deep Learning taking off ? (12.10 pm 1-12-18)

Deep learning is taking off due to, ⓐ Large amount of data
ⓑ Faster computation.
Scale drive deep learning Progress. ⓒ Innovation in NN algorithm



The process of training neural network is iterative.



L 4 :- About this course. (12.25 pm 1-12-18).

week 2 :

P1 :- Logistic Regression as a Neural Network.

E1 :- Binary classification. (12:05 pm 02-12-18)

In binary classification result is discrete value output.

Here, $x \longrightarrow y$. (corresponding label of training example)

(Feature of T.E.) — dimension of input feature vector

A single training example → (x, y) $\boxed{x \in \mathbb{R}^{n_x}}$, $\boxed{y \in \{0, 1\}}$.

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

whole training set is represented by a feature matrix

Feature vector which will our input.

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

$\longleftarrow m \longrightarrow$

m = number of training example.

n_x = Features of each training examples.

Label of the training example is denoted by a row vector. $Y = [y^{(1)}, y^{(2)} \ldots y^{(m)}]$ $\Rightarrow Y = \mathbb{R}^{1 \times m}$

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

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

- $X \in \mathbb{R}^{n_x \times m}$ is the input matrix.
- $x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th} example represented as a column vector.
- $Y \in \mathbb{R}^{n_y \times m}$ is the label matrix.
- $y^{(i)} \in \mathbb{R}^{n_y}$ is the output label for the i^{th} example

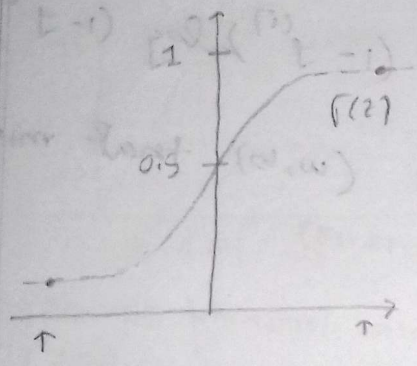
(Can be viewed as a very very small neural network)

L2 : Logistic Regression. (12.30 pm 02-12-18).

Given x, we want to predict $\hat{y} = P(y=1|x)$. $[0 \le \hat{y} \le 1]$.

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

output $\hat{y} = \sigma(w^T x + b)$. Now, $\boxed{w^T x + b = z}$



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

** If z is a large positive number.

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

** If z is a large negative number.

$e^{-z} =$ Big number. $\sigma(z) = \frac{1}{1+\text{Big}} \approx 0$

L3 : Logistic Regression cost function. (12.45 pm 02-12-18)

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

$\hat{y}^{(i)} \approx y^{(i)}$ [Predicted function $\hat{y}^{(i)}$ to be as close as actual output $y^{(i)}$]

Loss (error) Function : $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ [squared error function]

we can not use this squared error function. Because if we use it we do not get global optima because there are multiple local optima.

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

If $y=1$: $L(\hat{y}, y) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ large. [close to 1].

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

* In neural network w and b will be in separate matrix

Loss Function calculates on a single training set.

Cost Function measures how well our algorithm is doing in entire training set.

Cost function : $J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \ell(\hat{y}^{(i)}, y^{(i)})$

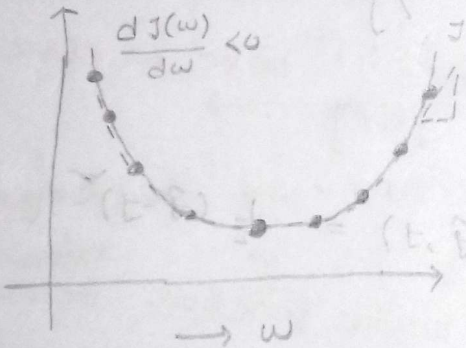
$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)}) \right]$$

we are going to find parameters (w,b) that minimizes cost function.

(10.45 Am 03-12-18)

L4 : Gradient Descent. (machine learning [illegible] veido [illegible])

* Gradient descent is an algorithm that helps to find (w and b) that minimizes the cost function.



Repeat {

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

"dw"

}

$w := w - \alpha dw$

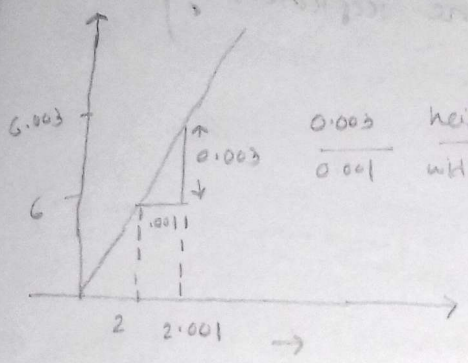
$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$ → dw — The amount by which you update w

$J(w,b)$

$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$ → db — The amount by which you update b

L 4 : Derivatives (11.10 Am 03-12-18).

In mathematics, the derivative is a way to show rate of change. The amount by which a function is changing at one given point.



$f(a) = 3a \quad a = 2, \quad f(a) = 6$

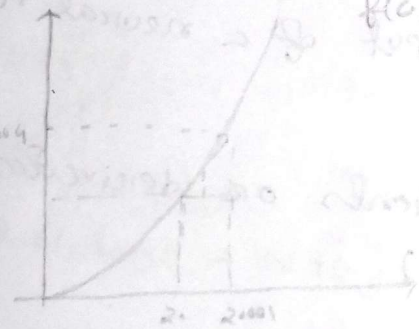
$a = 2.001 \quad f(a) = 6.003.$

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

[$\frac{df(a)}{da}$ (a) The amount of change in course change in $f(a)$]

L 5 : More derivative examples. (11.40 Am 03-12-18)

Derivatives can be different at different points of a line.



$f(a) = a^2 \quad a = 2, \quad f(a) = 4$

$a = 2.001 \quad f(a) = 4.004.$

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

$f(a) = 25$

$a = 5.001 \quad f(a) = 25.010.$

slope of $f(a)$ at $a = 5$ is 10.

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

আরো কিছু Derivative example. এরপর আর Problem নাই.

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

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

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

L6 : Computation Graph. (12:15 pm 03-12-18)
(Help to find more complex derivatives)

In neural network computation can be represented by a data flow graph called computation graph. Computation graph consists of a set of node each one representing an operation.

Example :- $J(a,b,c) = 3(a+bc)$.

$u = bc$

$v = a+u$

$J = 3v$.

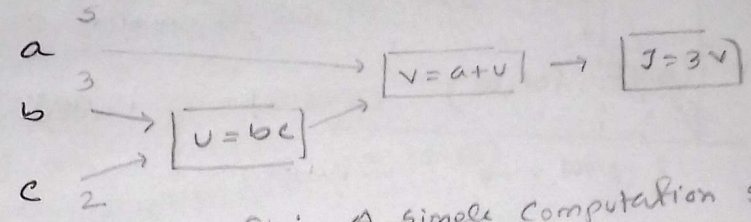


Fig:- A simple computation graph.

** Computation graph is useful when we want to optimize a special variable in this case that is J.

v.v.g

A neural network is organized in terms of

i) Forward Pass. (Compute output of a neural network) [It is a left to right pass]

ii) Backward Pass (Compute gradients or derivatives of a neural network) [It is a right to left pass]

L7 : Derivatives with a computation Graph. (3.50pm 16-12-18)

$\frac{dJ}{dv} = ?$ [It means that if we change the value of v. how does it effect the value of J].

[chain rule is main concept over v.v.J] [Successive derivatives]

$\frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da}$

The change in (J) due to change in (a) can be calculated by.

First computing how much (v) changes due to changing (a) multiplied by how much (J) changes due to change in (v)

* blue lines compute the output of neural network.
* Red lines compute the derivative of neural network.

(Layer 1 এর জন্য 2nd chain rule use করে derivative calculate করা হয়।)

$\frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da} = "da" = 3.$

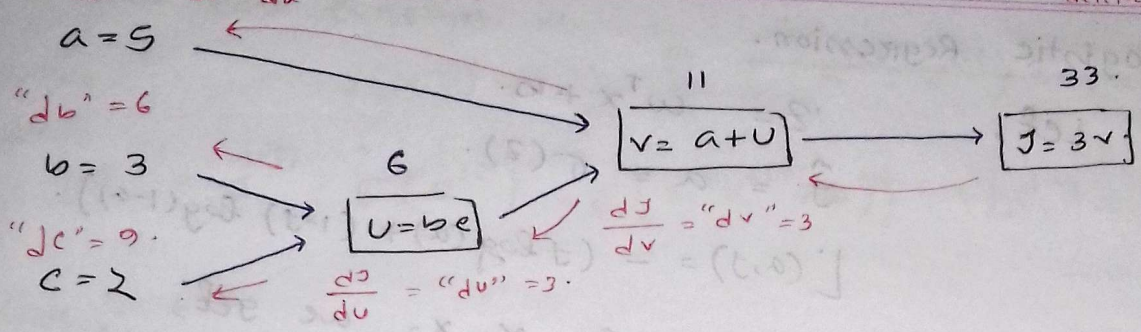


Fig:- Computation graph.

notation:- we have to compute the derivative of final output with respect to some intermediate variable. It can be represented by,

$$\frac{d\,\text{finaloutputvar}}{d\,\text{var}} = \text{"d var"}$$

- we will use it

$\frac{dJ}{dv} = 3.$ → [chain rule. মানে layer a value change করলে পরের layer J কি change হয়]

$\frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da} = 3 \cdot 1 = 3.$

$\frac{dJ}{du} = \frac{dJ}{dv} \cdot \frac{dv}{du} = 3 \cdot 1 = 3.$

$\frac{dJ}{db} = \frac{dJ}{dv} \cdot \frac{dv}{du} \cdot \frac{du}{db} = 3 \cdot 1 \cdot 2 = 6$

$\frac{dJ}{dc} = \frac{dJ}{dv} \cdot \frac{dv}{du} \cdot \frac{du}{dc} = 3 \cdot 3 \cdot 3 = 9.$

Example value put করে দেখানো হয়েছে।

"db" or "dc" mean derivatives of final output with respect to intermediate variable "b" and "c".

L8 :- Logistic Regression Gradient Descent. [4.30 pm 16.12.18]

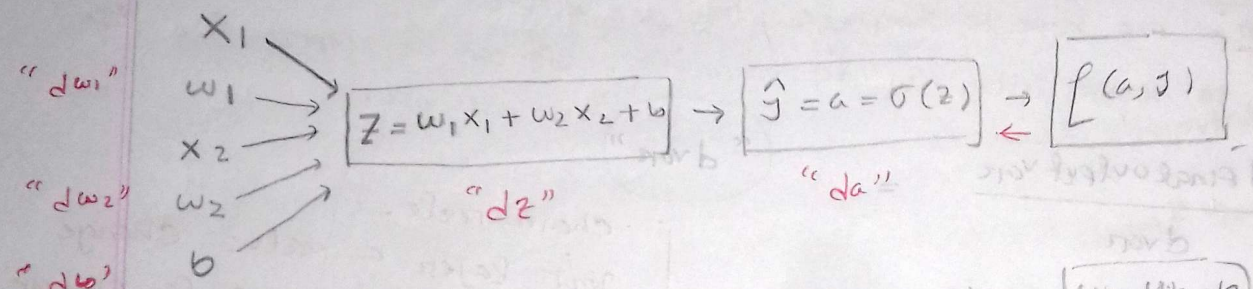
Logistic Regression.

Let, $z = w^T x + b$.

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

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

Now considering only two variables x_1, x_2 we get,



** our objective is to modify the parameters w_1, w_2, b in order to reduce loss L

Now, $"da" = \frac{dL(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$

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

$= a - y.$

$"dw_1" = \frac{dL}{dw_1} = x_1 \cdot dz.$

$"dw_2" = \frac{dL}{dw_2} = x_2 \cdot dz.$

$"db" = dz.$

(**) one step of gradient descent is,

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

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

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

L9: Gradient Descent of m Examples. [5:00 pm 16.12.18]

Overall cost function is.

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

prediction for ith example $\left[a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)\right]$ $\left[dw_1^{(i)}, dw_2^{(i)}, db^{(i)}\right]$

Code :- let

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

For $i=1$ to m

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

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

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

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

features বেশি হলে loop দিয়ে করা যাবে. { $dw_1 += x_1^{(i)} dz^{(i)}$; $dw_2 += x_2^{(i)} dz^{(i)}$ } [$n=2$ for 2 feature.] [vectorization use করে Easily implement করা যাবে.]

$db += dz^{(i)}$ end of loop

$J /= m.$

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

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

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

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

[This is sample code for one iteration]

Scanned by CamScanner

P2 : python and vectorization.

L1 : vectorization. (10.30 Am, 17-12-18)

* Faster implementation এর জন্য vectorized version ব্যবহার করা হয়।
vectorization just matrix multiplication এর উপর নির্ভর।

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

$$w = \begin{bmatrix} \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \end{bmatrix} \quad w \leftarrow \mathbb{R}^{n_x} \quad x \leftarrow \mathbb{R}^{n_x}.$$

Non-vectorized.

```
z = 0
For i in range (0, nx)
    z += w[i] * x[i]
z += b.
```

time required = 475 ms

vectorized.

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

time required = 1.5 ms.

** Built in function use করলে অনেক faster output পাওয়া যায় তাই built in function use করার Try করতে

L2 : more vectorization Example (10.45 Am, 17-12-18)

V.V.G

** for neural network programming whenever possible avoid explicit for loop. (built in function ব্যবহার)

Let, $v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$ we have to find $u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$.

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).
```

** Elementwise operation ব্যবহার করে আমরা vectorize implementation এর ক্ষেত্রে অনেক।

- np.log(v) → Elementwise log.
- np.abs(v) → Elementwise absolute value
- np.maximum(v,0) → Maximum of v.
- v**2 → Elementwise square

L3: Vectorizing Logistic Regression. (11.00 Am 17-12-18)

Using Logistic regression predictions are,

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

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

This implementation can be vectorized,

n_x = number of features
m = number of Training examples.

$$Z = w^T X + b$$

$$[z^1 \; z^2 \; \cdots \; z^m] = [w_1 \; w_2 \; \cdots \; w_{n_x}] \times \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} + [b \; b \; \cdots \; b]$$

$1 \times n_x$ $\quad n_x \times m$ $\quad 1 \times m$

$$= [w^T x^{(1)} + b \quad w^T x^{(2)} + b \quad \cdots \quad w^T x^{(3)} + b]$$

$1 \times m$

Computing Forward propagation

vectorized

$$Z = np.dot(w.T, X) + b$$
$$A = [a^{(1)} \; a^{(2)} \; \cdots \; a^{(m)}] = \sigma(Z)$$

L4: Vectorizing Logistic Regression Gradient output (11.15 Am 17-12-18)

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

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

$dz^{(1)} = a^{(1)} - y^{(1)} \quad \cdots \quad dz^{(m)} = a^{(m)} - y^{(m)}$

$\therefore$ $dz = A - Y$ vectorized

$dw = 0$

$dw += x^{(1)} dz^{(1)}$

$dw += x^{(2)} dz^{(2)}$

$\vdots$

$dw /= m$

vectorized

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

$$= \frac{1}{m} \underbrace{\begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix}}_{(n_x \times m)} \underbrace{\begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}}_{m \times 1}$$

$$= \frac{1}{m} [x^{(1)} dz^{(1)} + \cdots + x^{(m)} dz^{(m)}]$$

$db = 0$

$db \mathrel{+}= dz^{(1)}$

$db \mathrel{+}= dz^{(2)}$

$\vdots$

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

$db /= m.$

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

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

Gradient এর জন্য vectorize code

** vectorize implementation of logistic regression.

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

for i = 1 to m:

$z^{(i)} = w^T x^{(i)} + b$ → vectorization for m training examples → $Z = w^T X + b$ $= np.dot(w.T, X) + b$

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

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

$dz^{(i)} = a^{(i)} - y^{(i)}$ → $dZ = A - Y$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$
$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$ → $dw = \frac{1}{m} X dZ^T$

$db \mathrel{+}= dz^{(i)}$ → $db = \frac{1}{m} \, np.sum(dZ)$

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

এই [illegible] vectorization
এ [illegible].

for iter in range(1000):

1000 iteration of gradient descent.

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

This the code of Iteration

L4: Broadcasting in python. [10:00 pm 17-12-18]

Normally speed বাড়ে

* For loop ছাড়া এভাবে না করলে বড় হয়, এই কাজ কোনো technique ব্যবহার হয়না।

cal = A.sum(axis=0)

[Axis 0 → sum vertically ↑ 0
Axis 1 → sum horizontally → 1]

percentage = 100 * A/cal.reshape(1,4)

Genecral principle of broadcasting.

[.reshape matrix size ঠিক রাখার জন্য ব্যবহার করা হয়।]

1. একটা (m,n) matrix এর সাথে যদি (1,n) matrix এর operation করি তাহলে (1,n) matrix broadcast হবে, (m,n) matrix হবে

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

(3,1) (1,1) → (3,1)

[Broadcasting normally +, −, *, / এ ব্যবহার হয়।]

2. একটা (m,n) matrix এর সাথে যদি (m,1) matrix এর operation করি তাহলে (m,1) matrix broadcast হবে (m,n) matrix হবে।

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} + 100 \rightarrow \begin{bmatrix} 100 & 100 & 100 \end{bmatrix} \rightarrow \begin{bmatrix} 101 & 102 & 103 \end{bmatrix}.$$

(1x3)

L5: A note on python/numpy vectors. [10:25 17-12-18]

** Do not make a rank 1 array. → This will lead to error,

{ a = np.random.randn(5)
a.shape = (5,)
Do not use it }

use,

a = np.random.randn(5,1).
(column vector)

a = np.random.randn(1,5)
[row vector].

assert(a.shape == (5,1)).

⊛ যদি rank 1 Array তৈরি হয় তাহলে reshape দিয়ে row or column vector তৈরি করে নিতে হবে।

week 3 : Shallow neural networks. [other mathematical function are very easy]

L1 :- Neural Networks overview. [11:45 pm 18-12-18]

Logistic regression is a simple neural network, it is.

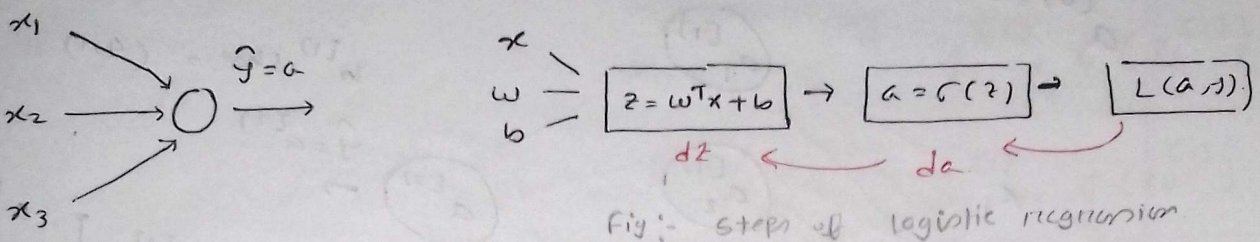
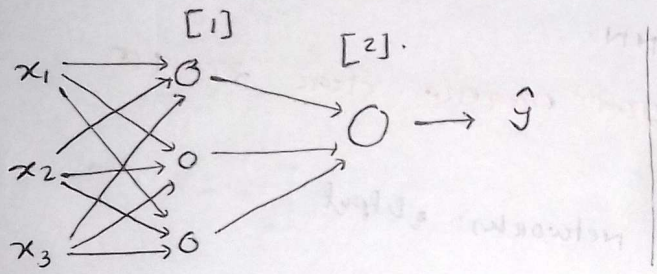


Fig :- steps of logistic regression

Typically a neural network has multiple hidden layer, output of one layer is taken input of another Layer.

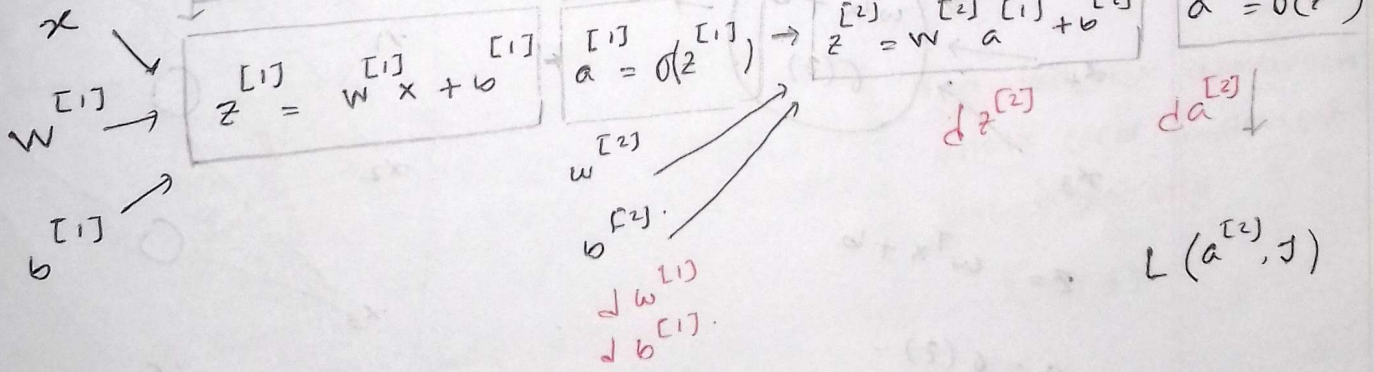


$x^{(i)}$ → This means ith training example

$x^{[i]}$ → This means ith hidden layer.

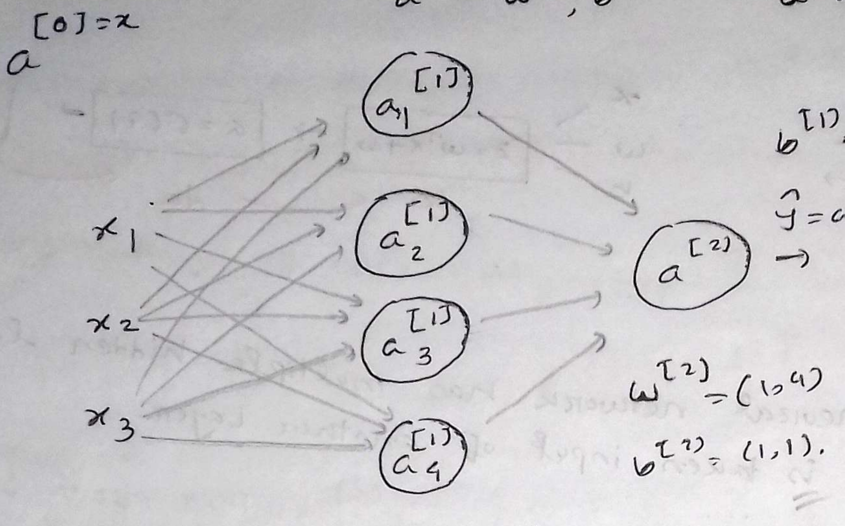
** round bracket (i) represents ith training example.
square bracket [i] represents ith hidden layer of NN.

Simple calculation of this neural network,



*** $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$ এগুলো হল weight and bias. Neural network backpropagation এ সবচেয়ে গুরুত্বপূর্ণ weight গুলো নিয়েই আমাদের কাজ করা।

L2: Neural Network Representation [12:06 Am 19.12.18]



$w^{[1]}.shape = (4,3)$

Hidden layer এ 4 টা Neuron
3 3 টা input variable

$b^{[1]}.shape = (4,1)$

$\hat{y} = a^{[2]}$ →

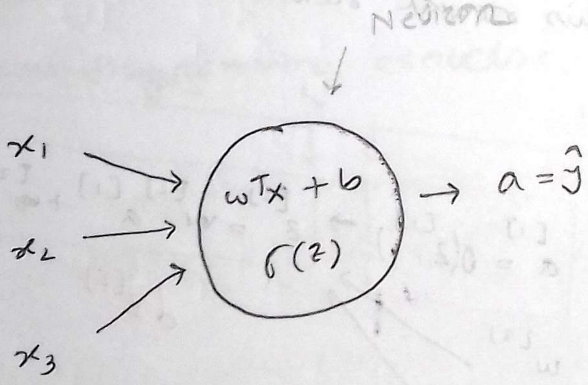
$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$$

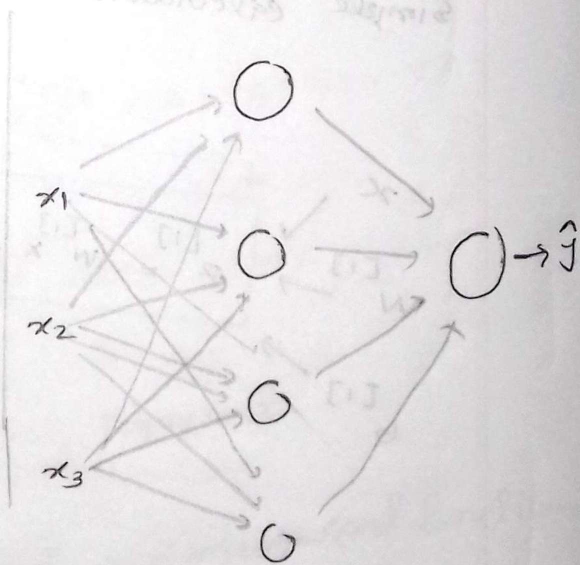
fig :- 2 Layer NN.

** Layer এর dimension গুলো আমাকে clear হতে হবে.

L3: Computing a Neural Networks' output [12:20 Am 19.12.18]

Computation on a single neuron,





$z = w^Tx + b$

$a = \sigma(z)$.

→ Simply a neural network do this computation for each neuron in each layer.

(Easy concept)

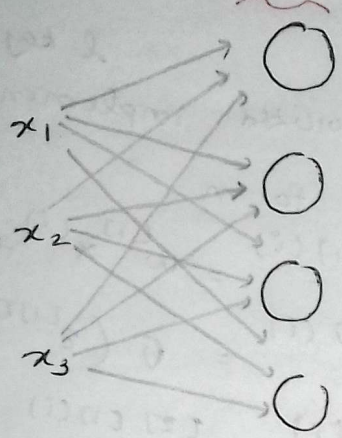
fig :- 2 Layer NN

যখন $w_i^{[1]}$ → 1st dimension 2nd layer
এরকম যতগুলো input থাকবে, তাদের
$w_1^{[1]} = (3,1)$

[l] ← Layer no.
a_i ← node in the layer

Now Computation of each part

Layer 1.



$$z_1^{[1]} = w_1^{[1]T} + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]\cdot T} + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]\cdot T} + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]\cdot T} + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]}).$$

vectorization

$$\underbrace{\begin{bmatrix} - w_1^{[1]\cdot T} - \\ - w_2^{[1]\cdot T} - \\ - w_3^{[1]\cdot T} - \\ - w_4^{[1]\cdot T} - \end{bmatrix}}_{W^{[1]}} \cdot \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x = a^{[0]}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]}} = \underbrace{\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]}}$$

(4×3) (3×1) (4×1) (4×1)

$$\sigma(z^{[1]}) = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = a^{[1]}.$$

Equations of 2-layer NN. :-

$$z^{[1]} = W^{[1]} \cdot a^{[0]} + b^{[1]}$$
(4,1) (4,3) (3,1) (4,1)

$$a^{[1]} = \sigma(z^{[1]})$$
(4,1) (4,1)

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
(1,1) (1,4) (4,1) (1,1)

$$a^{[2]} = \sigma(z^{[2]})$$
(1,1) (1,1)

Note

* matrix এর dimension সমস্যা হলে implement করার আগে হিসাব করুন.

(v.v.y)

Easy

L4: Vectorizing Across Multiple Examples. [6.15 pm 19.12.18]

For a training example what we get from neural network,

$$X \longrightarrow a^{[2]} = \hat{y}$$

$a^{[l](i)}$ → ith training example
↳ l layer.

For m training examples,

$$x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$$

$$x^{(2)} \longrightarrow a^{[2](2)} = \hat{y}^{(2)}$$

— — — — — —

$$x^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$$

unvectorized implementation.

for $i = 1$ to m,

$$z^{[1](i)} = w^{[1]} x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

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

$$a^{[2](i)} = \sigma(z^{[2](i)}).$$

→ Just একটা equation [illegible] training example এর [illegible]

** vectorized implementation.

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

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]}).$$

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

(n_x, m) ↳ number of input feature

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

row → # Number of training example

col ↓ # Number of unit in a hidden layer

** Matrix এর dimension [illegible] easily implement [illegible] **

L5 :- Explanation for vectorized implementation. [6.35 pm 19.12.18]

[Vectorize করে implement করলে যে সঠিক output দিবে হবে তা ধারণা করতে করে কাজ করে দেখা যায় = [Easy]]

v.v.g

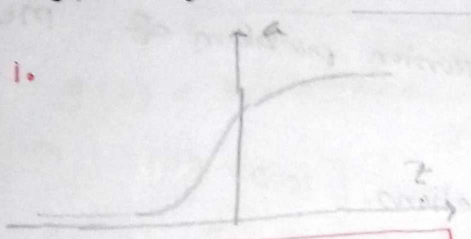
L6 : Activation functions. [6.45 pm 19.12.18]

(wiki)

→ The activation function of a node defines the output of that node given an input or set of inputs. ~~Few~~ Several types of activation functions are available. (wiki).

** different activation function use করে final result evaluate করা

1.



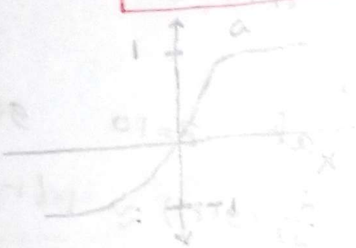
* Almost never use it because it is slow.

** If the problem is a binary class problem it can be used in final layer.

sigmoid : $a = \frac{1}{1+e^{-z}}$

$\tanh = \frac{e^{z} - e^{-z}}{e^{z}+e^{-z}}$

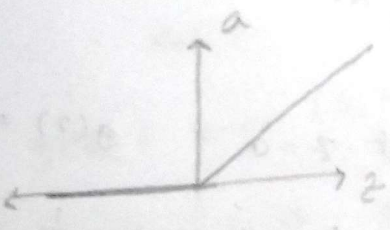
2.



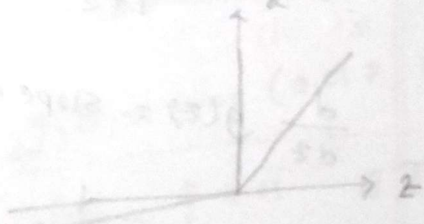
** It is superior than sigmoid.

→ The main drawback of ~~gradient descent~~ Sigmoid and tanh is that if z is very small or very large gradient descent will be very slow.

3.



4.



ReLU = max (0, z)

Leaky ReLU = max (0.01 z, z).

v.v.g

** In both of these algorithm our network will learn much faster. (It is better to use ReLU or Leaky ReLU).

very imp

** we can use different activation function at different layer of neural network.

Scanned by CamScanner

(দুবার দেখা আছে (ব))

L7 : why do you need non-linear activation functions [7.20 pm 19.12.18]

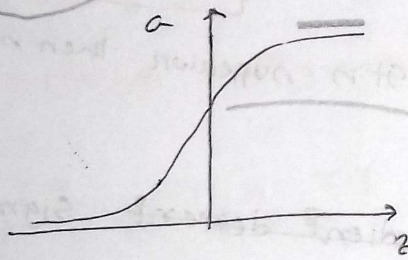
→ If we use linear activation function then hidden layers in the network will be useless. because composition of two linear function is a linear function.
(slide এ দেয়া mathematical explanation দেখা আছে)

→ ~~I~~ unless we throw some non-linearity in there we do not computing more interesting function even if we go deeper in network.

→ linear function is used in regression problem of ML

L8 : Derivatives of Activation Functions. [10.00 pm 19.12.18]

sigmoid activation function.



at $z = 10$ $\quad g(z) \approx 1$

$\frac{d}{dz}(g(z)) \approx 1(1-1) \approx 0$

at $z = -10$ $\quad g(z) \approx 0$

$\frac{d}{dz}(g(z)) \approx 0 \cdot (1-0) \approx 0$

at $z = 0$ $\quad g(z) = 1/2$

$\frac{d}{dz}(g(z)) = \frac{1}{2}(1 - 1/2) = 1/4.$

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

$\frac{d}{dz} g(z) =$ slope of $g(z)$ at z

→ ধরি.

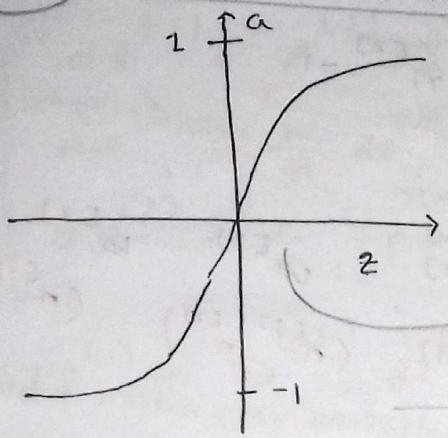
$$= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= g(z)\,(1 - g(z))$$

$$g'(z) = a(1-a)$$

$$\left[\; g'(z) = a(1-a) \;,\quad a = \frac{1}{1+e^{-z}} \;\right]$$

tanh activation function.



at $z=10$, $\tanh(z) \approx 1$

$g'(z) \approx 0$

at $z=0$ $\tanh(z) = 0$

$g'(z) = 1$

at $z=-10$ $\tanh(z) = 0$

$g'(z) = 1$

$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = \frac{d}{dz}(g(z)) = \text{slop of } g(z) \text{ at } z.$$

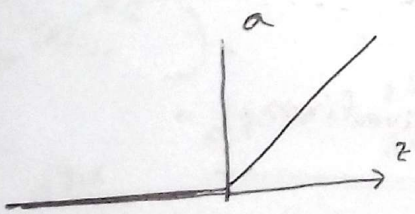
$$= 1 - (\tanh(z))^2$$

$$= 1 - a^2.$$

$g'(z) = 1 - a^2$

$a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

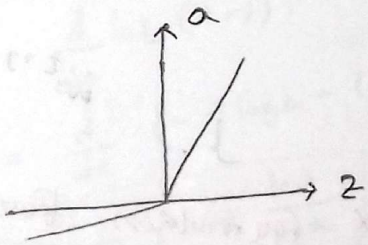
ReLU and Leaky ReLU.



ReLU.

$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$



$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0. \end{cases}$$

L-7 : Gradient descent for Neural Networks. [10:20 19.12.18]

Consider a neural network with one hidden layer,

$n_x = n^{[0]}$ → Input features, $n^{[1]}$ → number of units in hidden layer, $n^{[2]} = 1$ → output layer.

Parameters : $w^{[1]}$, $b^{[1]}$, $w^{[2]}$, $b^{[2]}$.

Dimension → $(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$

* In neural network we have to initialize parameters randomly.

Cost function : $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}, y)$.

Gradient Descent,

Repeat {

Compute prediction ($\hat{y}^{(1)}, \hat{y}^{(2)}, \dots \hat{y}^{(m)}$) ← Forward propagation ও calculate cost.

$dw^{[1]} = \frac{dJ}{dw^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$.

$w^{[1]} = w^{[1]} - \alpha\, dw^{[1]}$.

$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$.

$w^{[2]} = \dots$ $b^{[2]} \dots$ (... week ...)

}.

** Formulas for computing derivatives.

Forward Propagation :-

$z^{[1]} = w^{[1]} x + b^{[1]}$

$A^{[1]} = g^{[1]}(z^{[1]})$

$z^{[2]} = w^{[2]} A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(z^{[2]})$

$= \sigma(z^{[2]})$.

$g(z)$ → Activation function. + binary classification এর জন্য sigmoid function ব্যবহার করা হয়েছে

Backpropagation :-

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

Similar as logistic regression:

$$dz^{[2]} = A^{[2]} - Y$$

$$dw^{[2]} = \frac{1}{m} dz^{[2]} \cdot A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} \cdot np.sum(dz^{[2]}, axis=1, keedim=True)$$

Prevent python to produce Rank 1 array

$$dz^{[1]} = w^{[2]T} \cdot dz^{[2]} * g^{[1]'}(z^{[1]})$$

$(n^{[1]}, m)$ $(n^{[1]}, m)$ Element wise

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

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

$(n^{[1]}, 1)$

L10 : Backpropagation Intuition.

Logistic regression.

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

$$da = \frac{d}{da} L(a, y)$$

$$= \frac{d}{da} (-y \log a - (1-y) \log(1-a))$$

$$= -\frac{y}{a} + \frac{1-y}{1-a}$$

$$"dz" = \frac{\partial L}{\partial a} \cdot \frac{da}{dz}$$

$$= \frac{\partial L}{\partial a} \cdot \frac{d}{dz}(g(z))$$

$$dz = da \cdot g'(z)$$

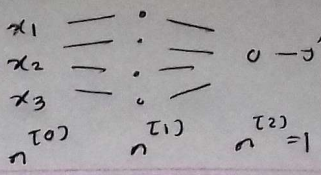
$dw = dz \cdot x$

$db = dz$

$x_1, x_2, x_3 \rightarrow (a) \rightarrow \hat{y} = a$

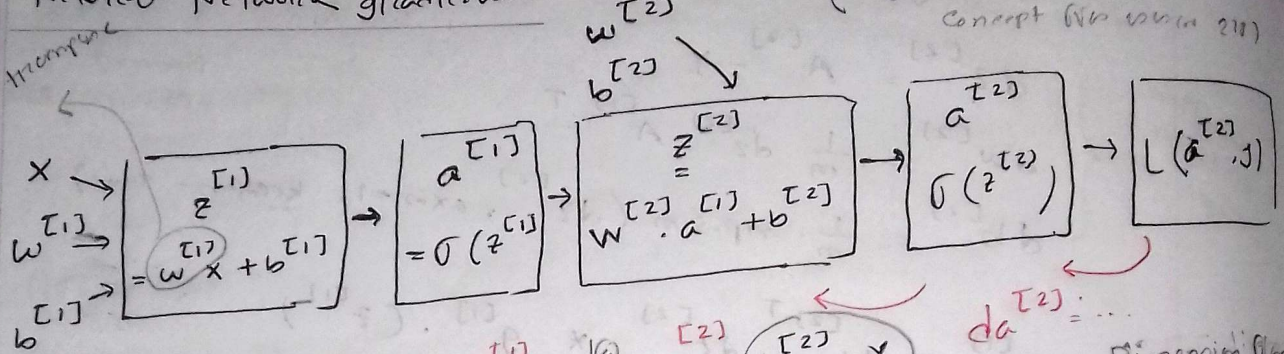
Neural network ची forward calculation

* most important, matrix dimension সবসময় খেয়াল রাখতে হবে.



Neural Network gradients.

(logistic regression এর concept দিয়ে বোঝান হচ্ছে)



(1) $dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$

(4) $dz^{[2]} = a^{[2]} - Y$

(5) $dw^{[2]} = dz^{[2]} \cdot a^{[1]T}$

(6) $db^{[2]} = dz^{[2]}$.

Dimension matching.

$w^{[2]} \rightarrow (n^{[2]}, n^{[1]})$

$z^{[1]}, dz^{[1]} \rightarrow (n^{[1]}, 1)$

$z^{2}, dz^{(2)} \rightarrow (n^{[2]}, 1)$

$\therefore dz^{[1]} = (n^{[1]}, n^{[2]}) \cdot (n^{[2]}, 1) * (n^{[1]}, 1)$

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

(2) $dw^{[1]} = dz^{[1]} \cdot x^T$.

(3) $db^{[1]} = dz^{[1]}$.

→ Summary.

$dz^{[2]} = a^{[2]} - y$

$dw^{[2]} = dz^{[2]} \cdot a^{[1]T}$

$db^{[2]} = dz^{[2]}$.

$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$

$dw^{[1]} = dz^{[1]} \cdot x^T$.

$db^{[1]} = dz^{[1]}$.

vectorized.

$dz^{[2]} = A^{[2]} - Y$.

$dw^{[2]} = \frac{1}{m} dz^{[2]} \cdot A^{[1]T}$

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

$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$

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

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

m training example এর average.

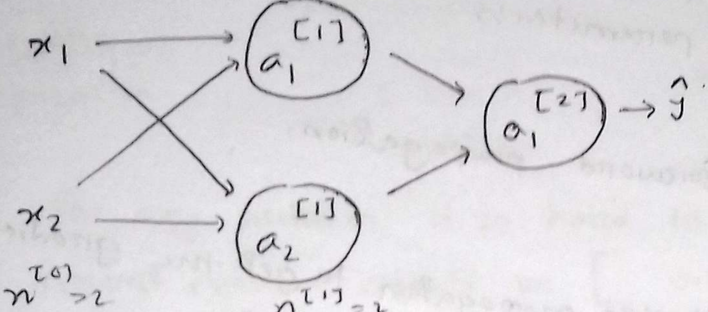
** সবসময় dimension check করে code লেখা উচিত।

Scanned by CamScanner

** Forward propagation (4 equations)
** Backward propagation. (6 equations) > Programming implement করতে হবে.

L11 :- Random initialization. [11.50 pm 19.12.18]

** Neural Network will not work if we initialize weights with <u>0</u>



$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

→ If we initialize weight with zeros, the neurons in hidden unit will ~~compute~~ symmetric because the calculate same function.

So, $a_1^{[1]} = a_2^{[1]} \cdots = a_n^{[1]}$

And the derivatives will also be same. $dz_1^{[1]} = dz_2^{[1]} \cdots dz_n^{[1]}$

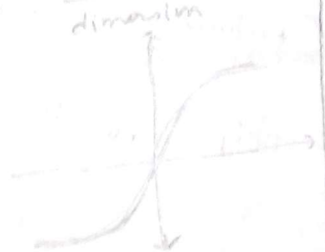
** The solution is to initialize weights with random values.

$W^{[1]}$ = np.random.randn ((2,2)) * 0.01 (dimension)

$b^{[1]}$ = np.zeros ((2,1)).

$W^{[2]}$ = ...

$b^{[2]}$ = ...



→ we initialize weights with small values because if we use large values learning rate of gradient descent will be very slow.

→ Programming Assignment. [will be able to].

1. Develop an intuition of back-propagation.
2. Recognize that more hidden layers can compute more Complex structures.
3. Built all helper functions to implement a full model with one hidden layer.

Scanned by CamScanner

Programming Assignment [week 3] [11.30 Am 26-12-18).

→ General methodology to build a neural Network.

1. Define Neural Network structure (# no of inputs, # of hidden units, etc)
2. Initialize. Model's parameters.
3. Loop :-
   - Implement forward propagation.
   - Compute loss.
   - Implement backward propagation to get the gradients.
   - update parameters. (gradient descent).

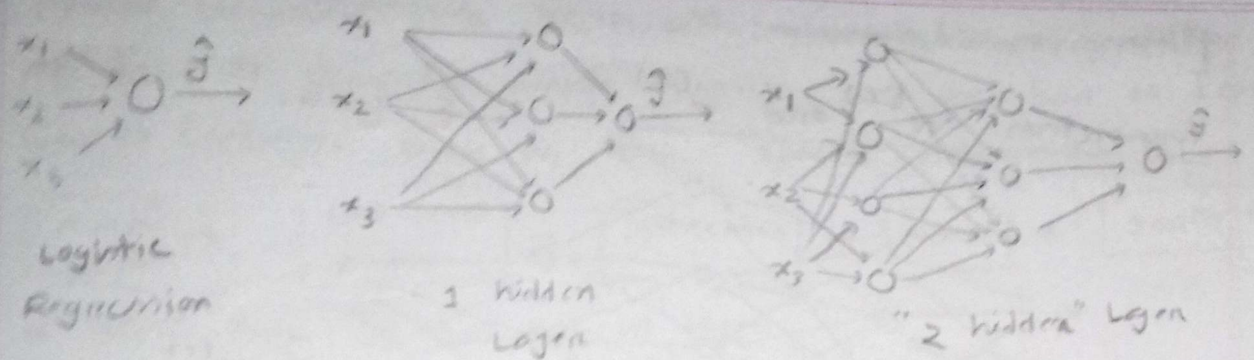
** Don't helper function ... Don't Helper function build ...

△ ... Hidden ... Accuracy ... (Future ...) (Paper ...)

* Don't Helper Function ...

(good)

L-1 Deep L-Layer Neural Network. [9.10 pm 26-12-18]



→ For any problem, it is hard to predict in advance how deep a neural network should be. [আমরা number of Layer এবং প্রতি Layer এ number of neuron change করে implement করে দেখবো].

Notation :-

L = number of hidden Layers.

$n^{[l]}$ = number of units in layer l.

$a^{[l]}$ = activation in layer l.

$w^{[l]}$ = weights in layer l.

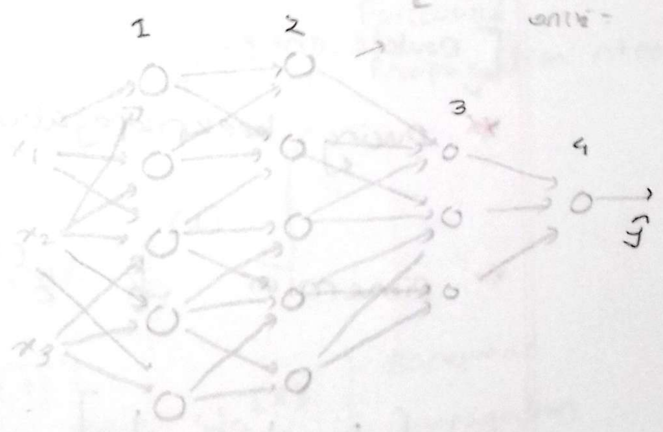
$b^{[l]}$ = Biases in layer l.

$x = a^{[0]}$ = input features.

$\hat{y} = a^{[L]}$ = outputs.

L-2 Forward Propagation in a Deep Network [9.25 pm 26-12-18]

[আগের Connection আছে]



Equation of forward propagation for Layer (i).

$$\begin{cases} Z^{[i]} = W^{[i]} A^{[i-1]} + b^{[i]} \\ A^{[i]} = g^{[i]}(Z^{[i]}) \end{cases}$$

for $i = 1, 2 \ldots L$.

for loop দিয়ে

→ আমার explicit for loop দিয়ে প্রতি hidden Layer এ দিয়ে যাবো implement করতে হবে

L-3 Getting your matrix dimensions right [9.40 pm. 26-12-18]

[note : matrix dimension ঠিক থাকলে Debugging easy হয়]

Parameters $W^{[l]}$ and $b^{[l]}$. [ধরি $L = 5$.

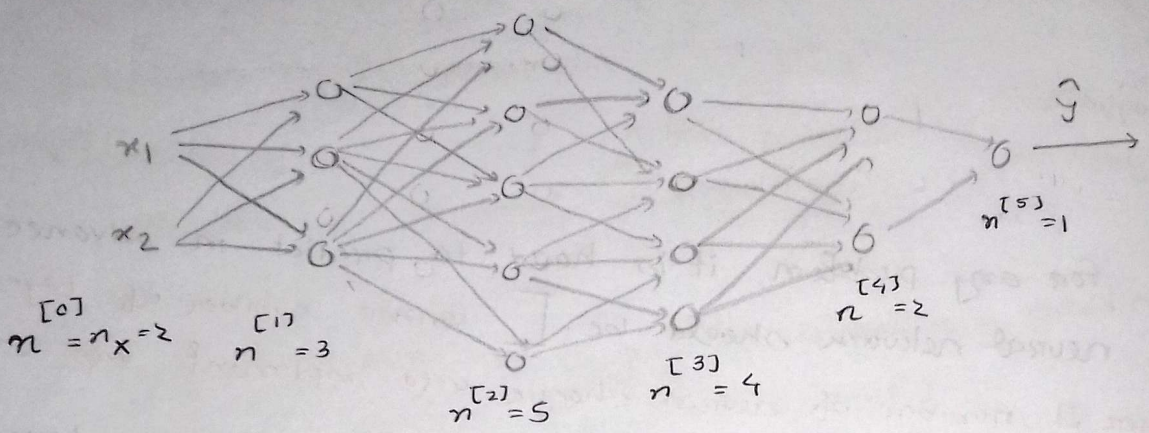


Fig:- Neural Network.

** Dimension of $W^{[l]}$ is : $(n^{[l]}, n^{[l-1]})$

$W^{[1]} = (3,2)$, $W^{[2]} = (5,3)$, $W^{[3]} = (4,3)$., $W^{[4]} = [2,4]$, $W^{[5]} = [1,2]$

** Dimension of $b^{[l]}$ is : $(n^{[l]}, 1)$

$b^{[1]} = (3,1)$, $b^{[2]} = (5,1)$, $b^{[3]} = (4,1)$, $b^{[4]} = (2,1)$, $b^{[5]} = (1,1)$.

[Double check করা হচ্ছে matrix dimension check করে implement করলে ভুল কম হয়]

** During backpropagation dimension e.g. $W^{[l]} = dW^{[l]}$, $b^{[l]} = db^{[l]}$

** Dimension of $Z^{[l]}$ for m training example is, $(n^{[l]}, m)$

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

⊛ number of hidden unit ও hidden layer এর dimension change হয়।

L-4 Why deep representation? [10.00 pm 26-12-18]

Neural Network is a complex hierarchical representation where at first it computes simple function and use this to learn complex features. Such as for "speech recognition":

Audio → Low level audio → phonemes → words → sentence

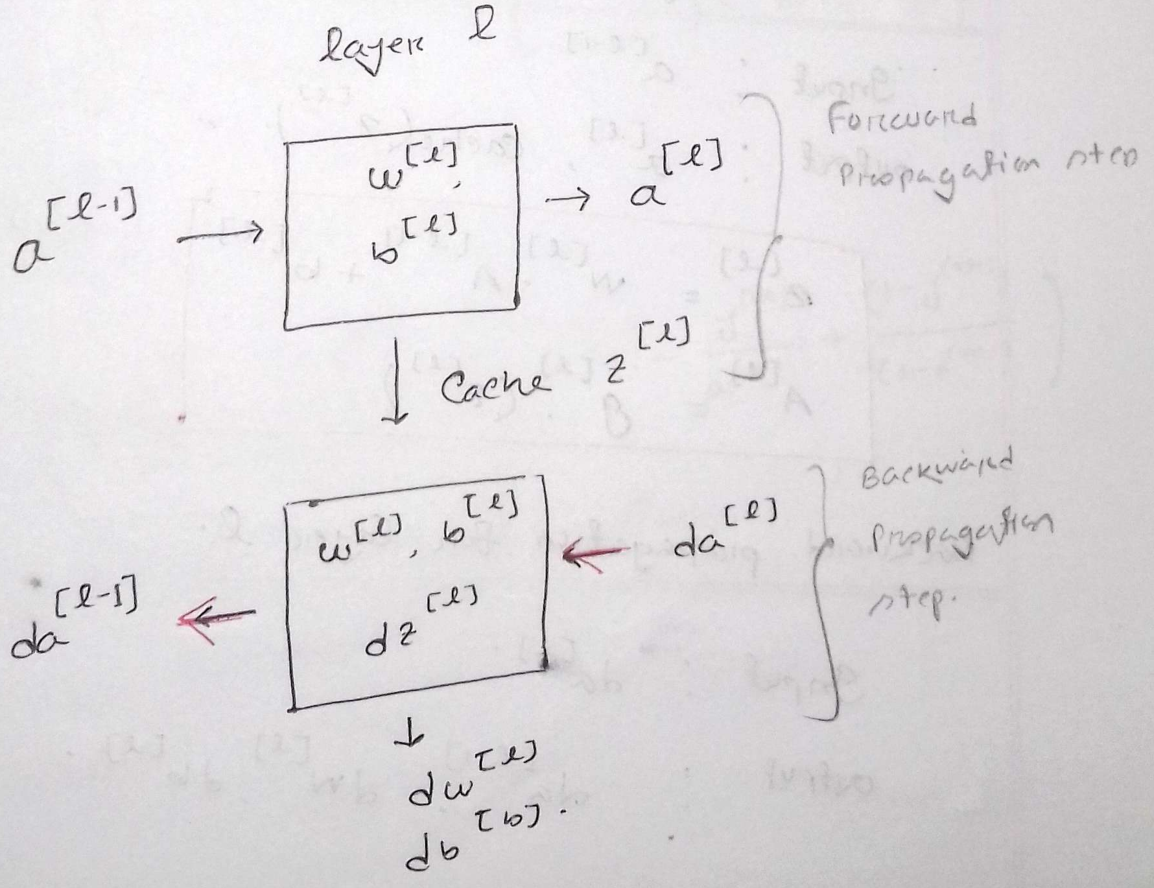
* প্রতি task এর জন্য আলাদা layer

Informally: There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

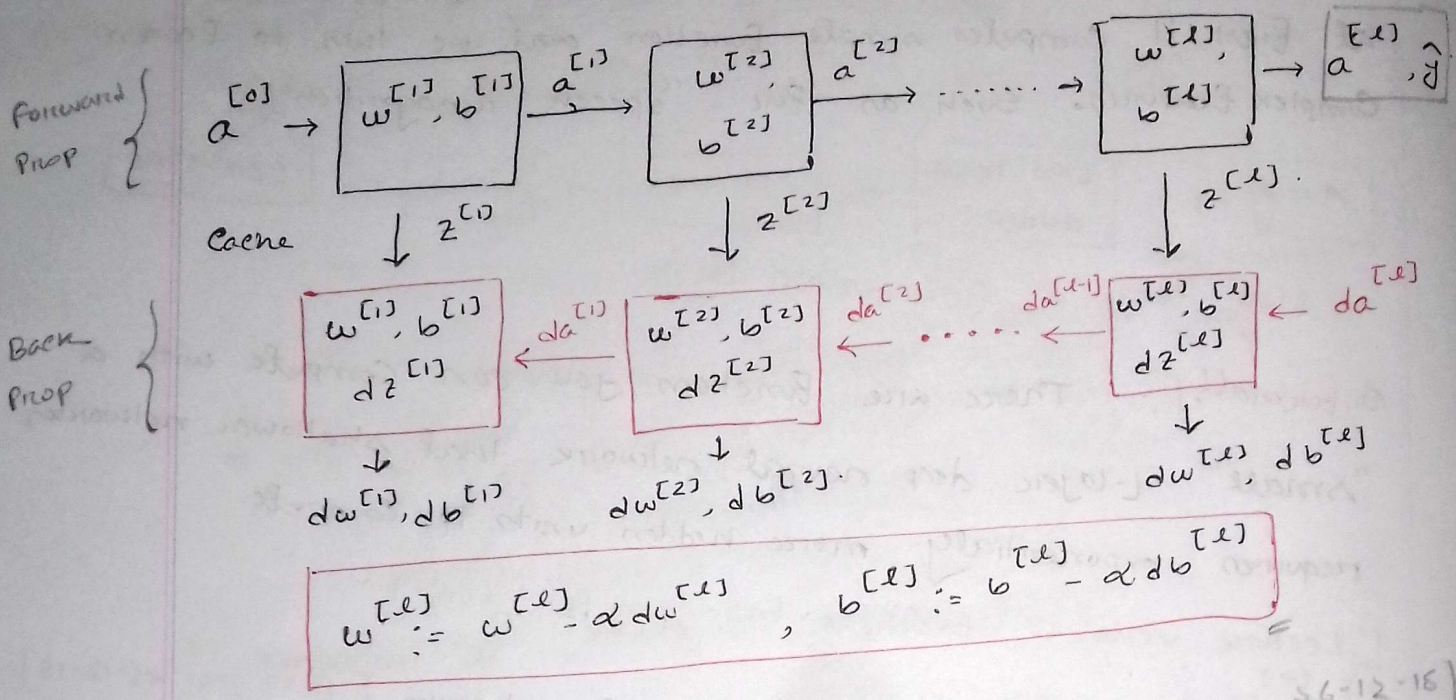
[Lecture veido তে Example দেওয়া আছে [illegible])

L-5 Building blocks of Deep neural networks. [11.30 pm 26-12-18]

Forward and Backward functions:



Iteration of a neural network :-



$$w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}, \quad b^{[l]} := b^{[l]} - \alpha\, db^{[l]}$$

L-6 Forward and backward Propagation [11:50 pm 26-12-16]

Forward propagation for layer l.

Input : $a^{[l-1]}$

output : $a^{[l]}$, cache($z^{[l]}$)

[equation for implementation]

$$z^{[l]} = w^{[l]} \cdot A^{[l-1]} + b^{[l]}$$

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

Backward propagation for layer l.

Input : $da^{[l]}$.

output : $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$.

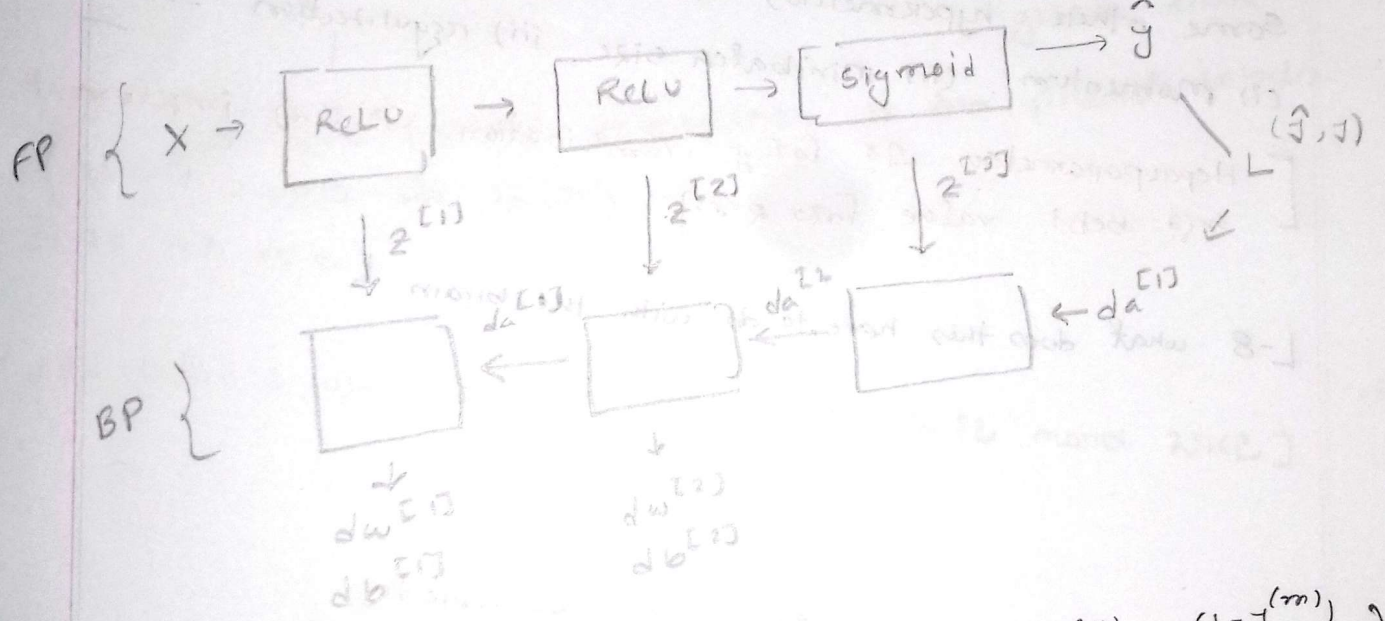
$$dz^{[l]} = dA^{[l]} * g^{[l]'}(z^{[l]})$$

$$dw^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1].T}$$

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

$$dA^{[l-1]} = w^{[l].T} \cdot dz^{[l]}$$

By summarizing we get.



initialize.

$$dA^{[l]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} + \cdots\cdots - \frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})} \right).$$

[implement

L-7 Parameters vs hyperparameters. [12.10 Am 27-12-18]

Parameters : $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$ ....

Hyperparameters : → learning rate α.

→ number of iterations.

→ hidden layer L.

→ number of hidden units

Choice of activation function.

Imp

এই hyperparameter change করে comparison দেখিয়ে paper লেখা যায়

[Imp for paper writing].

* hyperparameters controls parameters.

Some other hyperparameters are,

(i) Momentum. (ii) minibatch size iii) regulization ......

[Hyperparameters এর বিভিন্ন Combination এর জন্য implement করে best value নিতে হবে]. (Trial and error Process)

[12.20 Am 27-12-18]

L-8 what does this have to do with the brain

[এখানে brain এর সাথে তুলনামূলক কথা বলা হয়েছে এবং নিয়ে আলোচনা হয়েছে)

→ Programming Assignment. [will be able to].

Use non-linear units like ReLU to improve your model.

1. Use non-linear units like ReLU to improve your model.
2. Build a deeper neural network.
3. Implement easy to use neural network class

[Exercise এ অনেক Theory দেওয়া আছে Imp গুলো দাগানো আছে].

v.v.g.

** General Methodology for Deep Learning.

1. Initialize parameters / Define hyperparameters.
2. Loop for num_iterations:
   a) Forward propagation.
   b) Compute cost function.
   c) Backward propagation.
   d) update parameters (using parameters and grads from Backprop)
3. Use trained parameters to predict labels.