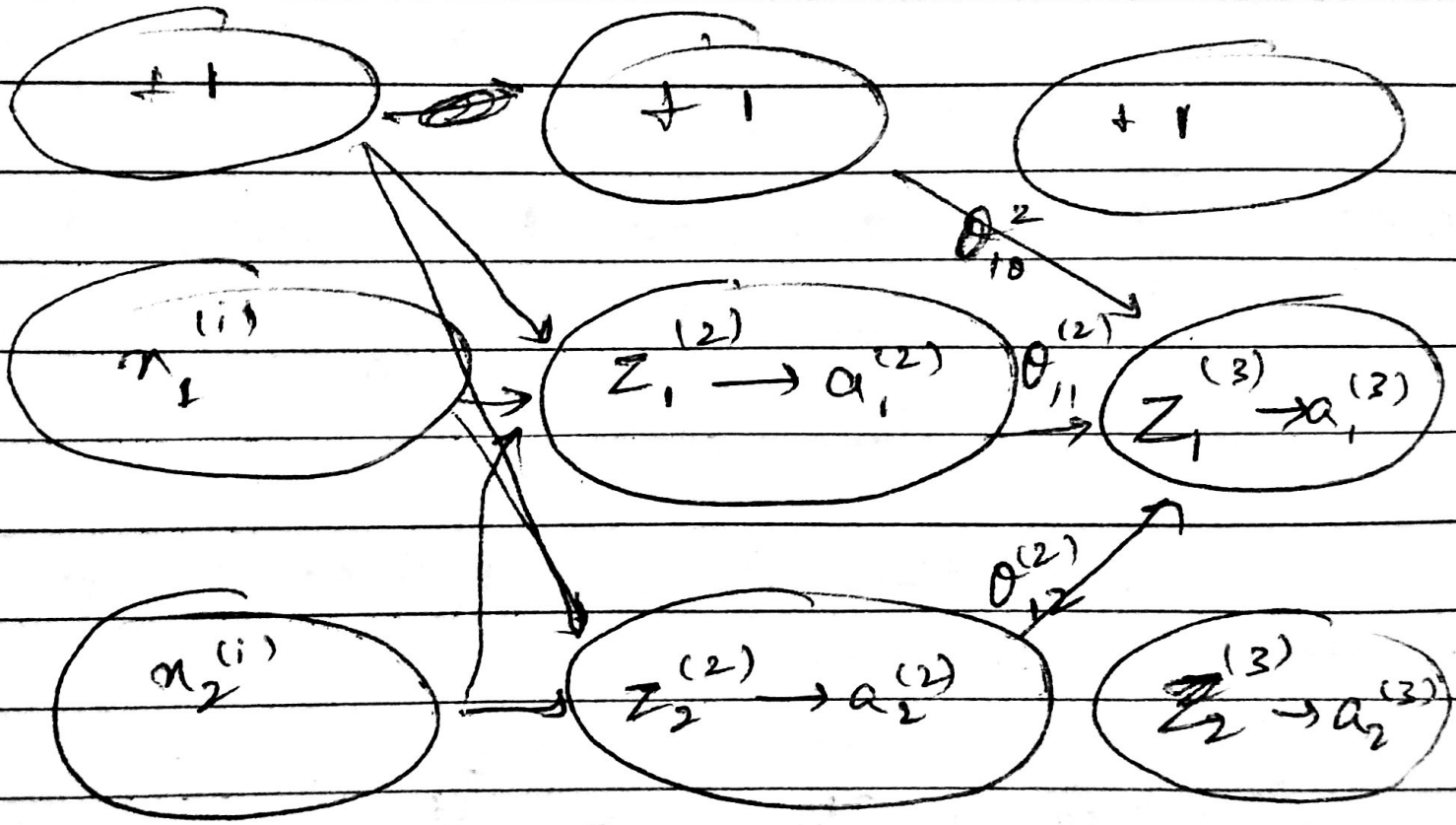
Forward propagation



$\therefore$ for $z_1^{(3)} = \theta_{10}^{(2)} \cdot 1 + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)}$

Unrolling Parameters

Normally, we have

function [jVal, gradient] = costFunction(theta)
{
}

jVal → cost function
gradient → parameters (vector)
theta → vector

optTheta = fminunc(@costFunction, initialTheta, options)

initialTheta → vector

Neural network,

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ } all matrices
$D^{(1)}, D^{(2)}, D^{(3)}$ }

Suppose, $S_1 = 10, S_2 = 10, S_3 = 1$

$\Theta^{(1)} \in R^{10 \times 11}$ $\Theta^{(2)} \in R^{10 \times 11}$ $\Theta^{(3)} \in R^{1 \times 11}$

$D^{(1)}$ " " "

unroll: thetaVec = [Theta1(:); Theta2(:); Theta3(:)]

DVec = [D1(:); D2(:); D3(:)];

for getting back, Theta1 = reshape(thetaVec(1:110), 10, 11)

Theta2 = reshape(thetaVec(111:220), 10, 11);

Theta3 = reshape(thetaVec(221:231), 1, 11);

Now,
for function,

function[jval, gradientVec] = costfunction (thetaVec)

→ From thetaVec,
get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$
→ use forward prop/back prop to
compute $D^{(1)}, D^{(2)}, D^{(2)}$ & $J(\Theta)$
→ unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get gradientVec

vector

GRADIENT CHECKING (Bug fixes)

$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}$$

$$\boxed{\epsilon \approx 10^{-4}}$$

for implementation,

gradApprox = (J(thetaVec + Epsilon) - J(thetaVec - Epsilon))
/ 2 * Epsilon

$\theta \in \mathbb{R}^n$

$\theta = \theta_1, \theta_2, \theta_3, \ldots \theta_n$

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3 \ldots) - J(\theta_1 - \epsilon, \theta}{2\epsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2 + \epsilon, \theta_3 \ldots) - J(\theta_1, \theta_2 - \epsilon, \theta_3 \ldots)}{2\epsilon}$$

Implementation

for i = 1 : n

thetaPlus = theta;

thetaplus(i) = thetaPlus(i) + Epsilon;

thetaminus = theta;

thetaminus = thetaMinus(i) - Epsilon

gradApprox(i) = (J(thetaplus) - J(thetaminus))
/ 2 * EPSILON

end;

↕

check gradApprox ≈ Dvec

↑ from backpropagation

→ Turn off gradient checking after you have checked the Dvec & gradApprox because gradient checking is very slow.

RANDOM INITIALIZATION

Some random θ

optTheta = fminunc(@costFunction, initialTheta, options)

we usually set,

initialTheta = zeros(n,1)

for regression, & it should not be used for Neural Networks.

for neural network, we randomly initialize weights in between $-\epsilon$ & $+\epsilon$

i.e, $-\epsilon \leq \theta_{ij}^{(l)} \leq \epsilon$

Theta1 = rand(10,11) * (2*INIT_EPSILON) - INIT_EPSILON.

Theta2 = rand(1,11) * (2*INIT_EPSILON) - INIT_EPSILON.

How to :

1) We start by picking a network architecture (connectivity pattern b/w neurons)

i/p is $x^{(i)}$

o/p is no. of classes. if y takes value as $y \in \{1, 2, 3 \ldots 10\}$ then, you have a vector of dimension ten.

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

* Reasonable default is take 1 hidden layer or if you take more, same no. of units in each layer.