Machine Learning

Week-3 (Classification problem)

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

m training examples.

each $x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}_{(n \times 1)} \Rightarrow x_0^{(i)} = 1, \quad \forall\ i = 1(1)m$

$\theta \in \mathbb{R}^{n+1}$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}_{(n+1) \times 1}$

$y \in \{0, 1\}$

$h_\theta(x) = \dfrac{1}{1 + e^{-\theta^T x}}$

$[\theta_0\ \theta_1 \ldots\ldots \theta_n] * \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{1}$

$(\theta_0 x_0 + \theta_1 x_1 + \ldots\ldots + \theta_n x_n)$

→ how do I choose parameters θ?

Cost fn in linear regression:

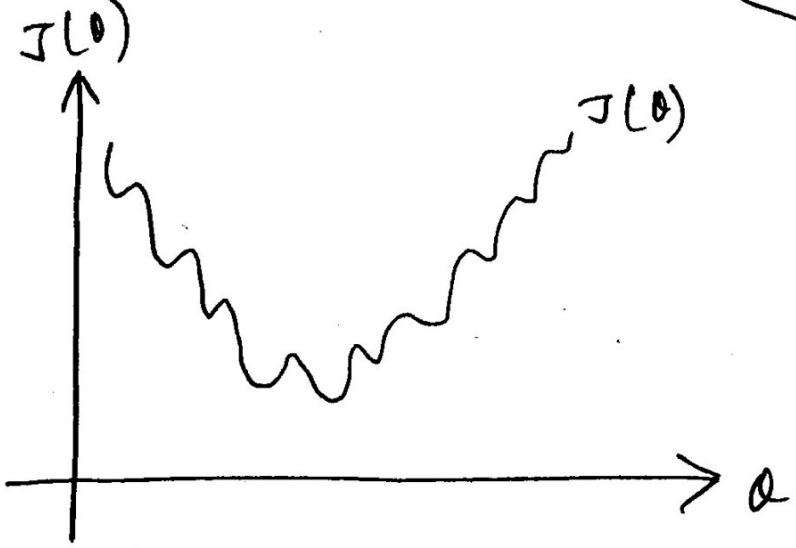
$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$= \frac{1}{m} \sum_{i=1}^{m} \text{cost}\left(h_\theta(x^{(i)}), y \right)$$

If we use $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}\left(h_\theta(x^{(i)}), y^{(i)}\right)$

$$\text{Cost}\left(h_\theta(x), y\right) = \frac{1}{2}\left(h_\theta(x) - y\right)^2$$

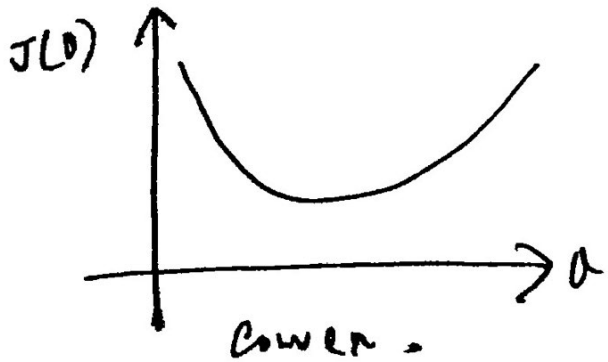
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



non-convex.

gradient descent will not always converge to global minimum in a convex fn.

we want something like

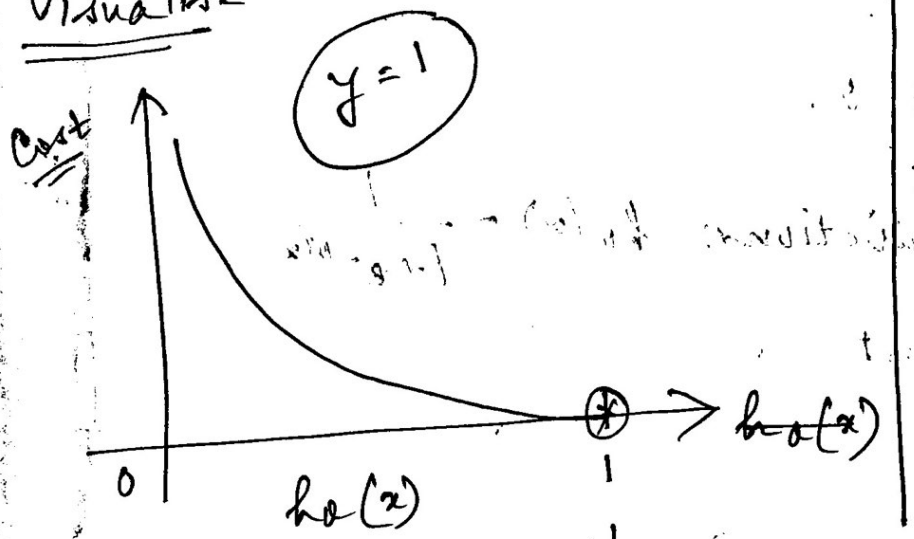


Cover.

New cost fn

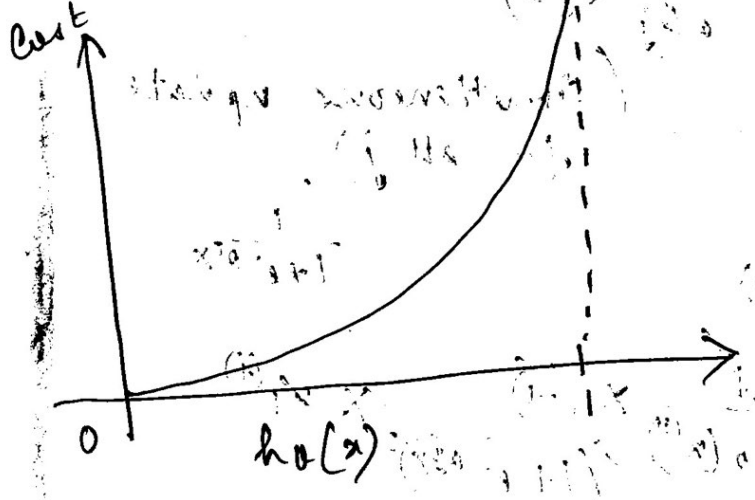
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Visualise



As $h_\theta(x) = 1$, if $y = 1$, Cost = 0.

If $h_\theta(x) = p(y=1 \mid x, \theta) = 0$, but $y = 1$, Cost $\to \infty$.



Simplified cost function

$$\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y) \log(1 - h_\theta$$

→ Maximum likelihood estimation

→ ~~Concave~~. Convex.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}\left(h_\theta(x^{(i)}), y^{(i)}\right)$$

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

$\min J(\theta)$ w.r.t θ.

To make predictions: $h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

gradient descent

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

(simultaneous update for all j).

}

$\frac{1}{1+e^{-\theta^T x}}$.

$$\frac{\partial}{\partial \theta_j} J(\theta) = -\frac{1}{m} \sum y^{(i)} \times \frac{1}{h_\theta(x^{(i)})} \times \frac{(-1)}{(1+e^{-\theta^T x})^2} \times x_j^{(i)}$$

{

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} \left(e^{(i)}\right) x_j^{(i)}$$

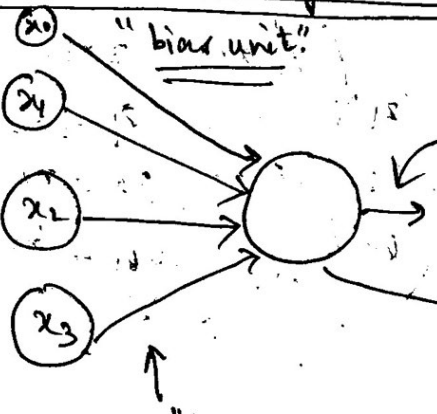
}.

Week-4 (Neural Networks)

Model representation (1)

Modelling how neurons in the brain work

Neuron model: logistic unit



$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \rightarrow \text{"weights" (parameters)}$$

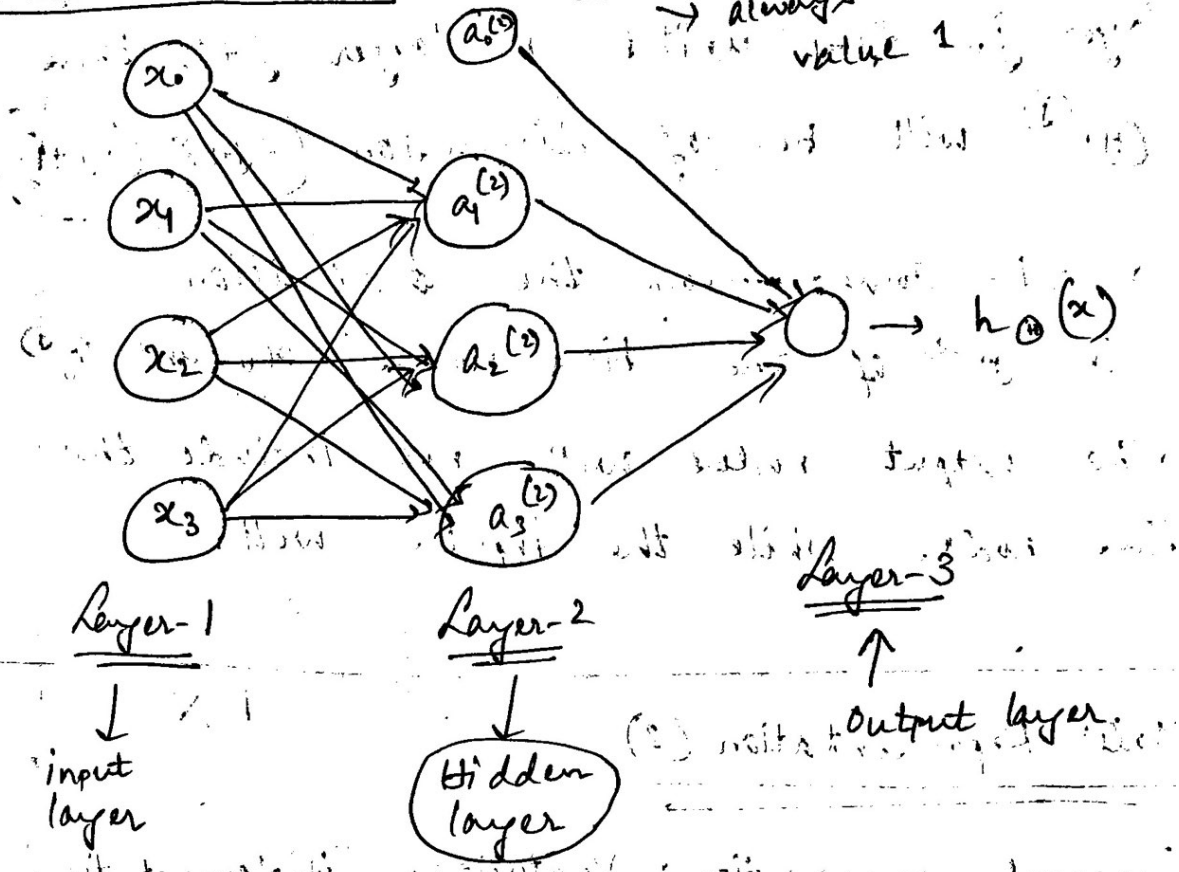
$x_0 \rightarrow$ bias unit $\rightarrow$ always $= 1$.

Sigmoid (logistic) activation function

This above diagram represents a single neuron.

Neural network $\rightarrow$ group of different neurons strong together.

Neural Network



$a_i^{(j)}$ = "activation" of unit i in layer j.

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $(j+1)$

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right)$$

$$h_\Theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$$

If network has s_j units in layer j and s_{j+1} units in layer $(j+1)$, then $\Theta^{(j)}$ is a $s_{j+1} \times (s_j + 1)$ dimension matrix.

$\Theta^{(1)} \in \mathbb{R}^{3 \times 4}$

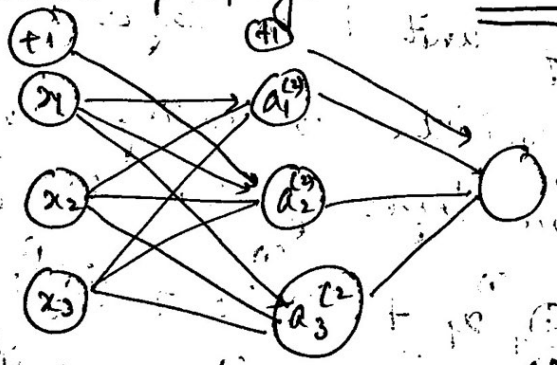
→ If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

→ ~~+1 comes from the addition in $\Theta^{(j)}$ of the "bias nodes", x_0 and $\Theta_0^{(j)}$~~

(*) The output nodes will not include the bias nodes while the inputs will.

Model Representation (2)

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right)$$

$$\hookrightarrow z_1^{(2)}$$

$$a_1^{(2)} = g(z_1^{(2)})$$

$$a_2^{(2)} = g(z_2^{(2)})$$

$$a_3^{(2)} = g(z_3^{(2)})$$

$$a^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \qquad a^{(1)} = x$$

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

$$a^{(2)} = g(z^{(2)}) \quad \text{(} a^{(2)} \in \mathbb{R}^3,\ z^{(2)} \in \mathbb{R}^3 \text{)} \rightarrow \text{elementwise sigmoid fct.}$$

Add $a_0^{(2)} = 1$ to get $a^{(2)} \in \mathbb{R}^4$

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

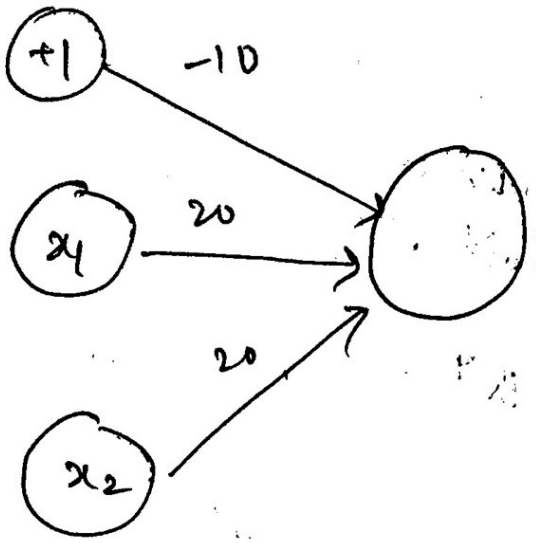
$$h_\Theta(x) = a^{(3)} = g(z^{(3)}) \quad \rightarrow \mathbb{R}$$

This process is called forward propagation

$$a^{(3)} = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$$

Applications

Examples and intuitions I



$$g(-10 + 20x_1 + 20x_2)$$

x_1	x_2	$h_\Theta(x)$
0	0	$\simeq 0$
0	1	$\simeq 1$
1	0	$\simeq 1$
1	1	$\simeq 1$

A simple example of applying neural networks is by predicting x_1 AND x_2 which is the logical 'and' operator and is only true if both x_1 and x_2 are 1.

Applications & intuitions - II

(Not x_1 and Not x_2) (NOR)

$y = 1$ if and only if $x_1 = x_2 = 0$

→ For negation just put a large weight in front of the number you want to negate.

$\Theta^{(1)}$ matrices for AND, NOR, and OR are:

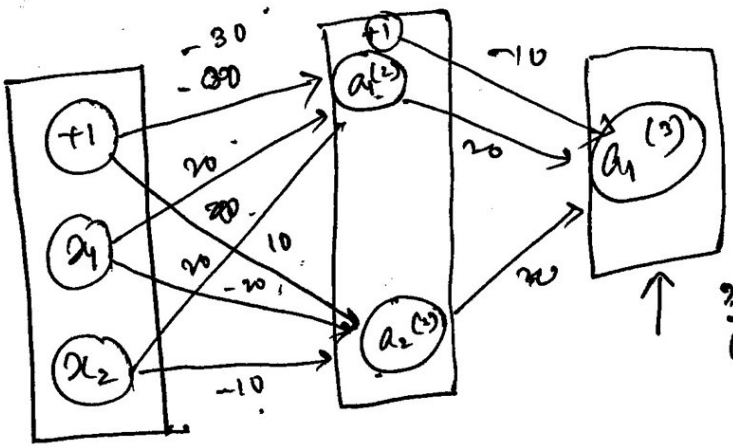
AND: $\Theta^{(1)} = [-30 \quad 20 \quad 20] \rightarrow$ (AND)

$\Theta^{(1)} = [10 \quad -20 \quad -20] \rightarrow$ (NOR)

$\Theta^{(1)} = [-10 \quad 20 \quad 20] \rightarrow$ (OR)

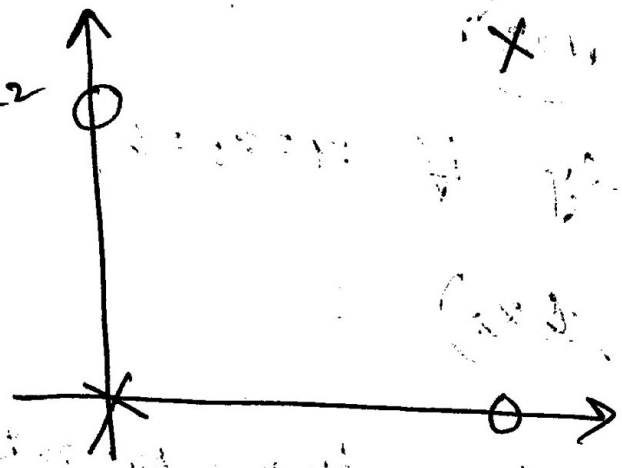
We combine these to get the XNOR operator

x_1 (XNOR) $x_2 \rightarrow 1$ if $x_1 \& x_2 = 0$ or $x_1 \& x_2 = 0$



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

$y = x_1$ XNOR x_2



X → positive examples [True]

O → negative examples [False]

Multiclass classification

Output $h_\Theta(x)$ is a vector of the number of classes.

Quiz → Answer.

5 hidden layers.

Vectorization

$n = 2$

gradient descent

$$\begin{cases} \theta_0 := \theta_0 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)} \\ \theta_1 := \theta_1 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_1^{(i)} \\ \theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_2^{(i)} \end{cases}$$

$$\theta := \theta - \alpha \, \delta \quad \text{(vector subtraction)}$$

$\theta \in \mathbb{R}^{n+1}$, $\theta \in \mathbb{R}^{n+1}$, $\alpha \in \mathbb{R}$, $\delta \in \mathbb{R}^{n+1}$

$$\delta = \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \end{bmatrix} \qquad \delta_\theta = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\left(h_\theta(x^{(i)}) - y^{(i)} \right)}_{\in \mathbb{R}} x_\theta^{(i)} \quad x_\theta^{(i)} \in \mathbb{R}^{n+1}$$

$u = 2v + 5w$.

$$v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}, \quad w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$$\begin{bmatrix} 2v_1 + 5w_1 \\ 2v_2 + 5w_2 \\ 2v_3 + 5w_3 \\ \vdots \end{bmatrix} = e^{(1)} \begin{bmatrix} x_0^{(1)} \\ x_1^{(1)} \\ \vdots \\ x_n^{(1)} \end{bmatrix} + e^{(2)} \begin{bmatrix} x_0^{(2)} \\ x_1^{(2)} \\ \vdots \\ x_m^{(2)} \end{bmatrix}$$

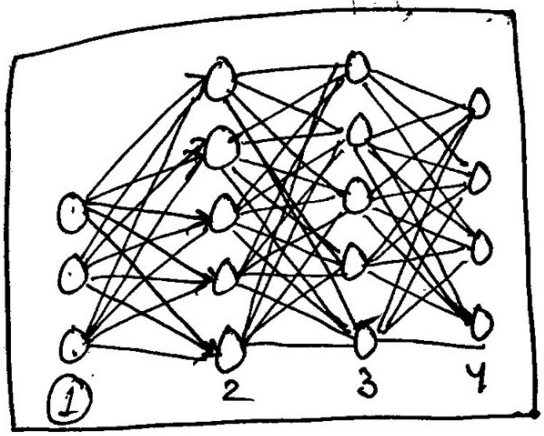
Week-5 (Neural Networks: Learning)

Cost function

Application of neural network to classification problem.

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

→ L = total no. of layers in the network



→ s_l = no. of units (excl. bias unit) in layer l.

$s_1 = 3, \; s_2 = 5, \; s_3 = 5, \; s_4 = s_L = 4$

Binary ($K = 1$)

$y = 0$ or 1

1 output unit

$h_\Theta(x) \in \mathbb{R}$

$s_L = 1$ ~~$K = 1$~~

Multi-class (K classes)

$y \in \mathbb{R}^K$

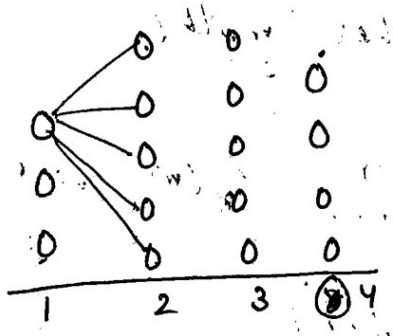
K output units

$h_\Theta(x) \in \mathbb{R}^K$

$s_L = K, \quad (K \geq 3)$

Backpropagation Algorithm

$\{(x, y)\}$ → only one training ~~set~~ example



Forward propagation

$x = a^{(1)}$; $\Theta^{(1)} \in \mathbb{R}^{5 \times 4}$

$z^{(2)} \in \mathbb{R}^{5 \times 1}$

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

$a^{(2)} = g(z^{(2)})$; (add $a_0^{(2)}$)

$a^{(2)} \in \mathbb{R}^{6 \times 1}$

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

$a^{(3)} = g(z^{(3)})$; (add $a_0^{(3)}$)

$z^{(4)} = \Theta^{(3)} a^{(3)}$

$a^{(4)} = g(z^{(4)})$

Backward propagation

δ → error

$\delta_1^{(4)} = y_1 - (h_\Theta(x))_1 - y_1$

Vectorized $\delta^{(4)} = (h_\Theta(x) - y)$

$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} . * g'(z^{(3)})$ → $a^{(3)} * (1 - a^{(3)})$

$\delta^{(2)} = (\Theta^{(2)})^T . \delta^{(3)} . * g'(z^{(3)})$

(No $\delta^{(1)}$)

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{K=1}^{K}\left[y_K^{(i)} \log h_\Theta(x^{(i)})_K + (1-y_K^{(i)}) \log(1-\right.$$

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{j=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

$$\min_{\Theta} J(\Theta)$$

Need to compute → (1) $J(\Theta)$

(2) $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ $\quad \Theta_{ij}^{(l)} \in$

→ main aim: gradient computation, for cost minimisation

Simple case: One training example - $(x^{(1)}, y)$

Forward propagation:

$a^{(1)} = x$

$z^{(2)} = \Theta^{(1)} a^{(1)}$ $\quad \Theta^{(1)} \in \mathbb{R}^{4\times 3}$.

$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

$z^{(3)} = \Theta^{(2)} a^{(2)}$ $\quad \Theta^{(2)} \in \mathbb{R}^{1\times}$

$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)

$z^{(4)} = \Theta^{(3)} a^{(3)}$

$a^{(4)} = h_\Theta(x) = g(z^{(4)})$

→ Gradient computation: backprop algo

$\delta_j^{(l)}$ = "error" of node j in layer l.

For each output unit, we find error:

$$\delta_1^{(4)} = a_1^{(4)} - y_1 \quad \left(h_\Theta(x)\right)_1$$

Vectorise: $\delta^{(4)} = a^{(4)} - y$

→ use this formula: $\delta^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)} .* g'(z^{(3)})$

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'(z^{(2)}) \quad \rightarrow \quad a^{(2)} .* (1 - a^{(2)})$$

(No $\delta^{(1)}$) → input has no error.

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$ (ignoring λ)

→ this can be proved mathematically

Remember — ① how to compute $\delta^{(l)}$

② $\frac{\partial}{\partial \Theta_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)}$

→ Steps in backpropagation algorithm:

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

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j)

→ used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

for $t = 1$ to m — accumulator

— Set $a^{(1)} = x^{(t)}$

— Perform forward propagation to

— Compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

— Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$

— Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ ~~$\delta^{(1)}$~~

— $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ → using backprop

— $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

vectorize · assuming $\Delta^{(l)}$ is a matrix subscripted by (i, j)

→ accumulate the partial derivative terms from previous slide.

Note: j → denotes node number.

- Go outside for loop.

$$D_{ij}^{(l)} := \frac{1}{m}\left(\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}\right) \text{ if } j \neq 0$$

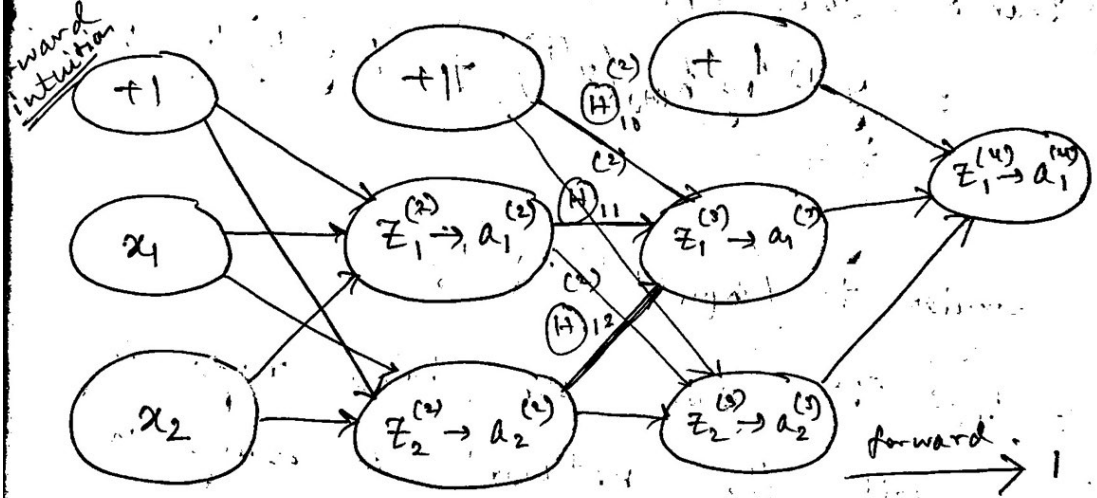
$$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)}$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

→ can be proved mathematical

*

→ Backpropagation intuition:



(excluding bias unit)

$$z_1^{(3)} = \left(\Theta_{10}^{(2)} \times 1\right) + \left(\Theta_{11}^{(2)} \times a_1^{(2)}\right) + \left(\Theta_{12}^{(2)} \times a_2^{(2)}\right)$$

~~For every node calculation~~

Every node is a weighted sum of nodes in previous layer.

→ What is backpropagation doing ?

for K = 1

Cost function is

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)}) \log(1-h_\theta \right.$$

$$+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(\Theta_{ji}^{(l)} \right)^2$$

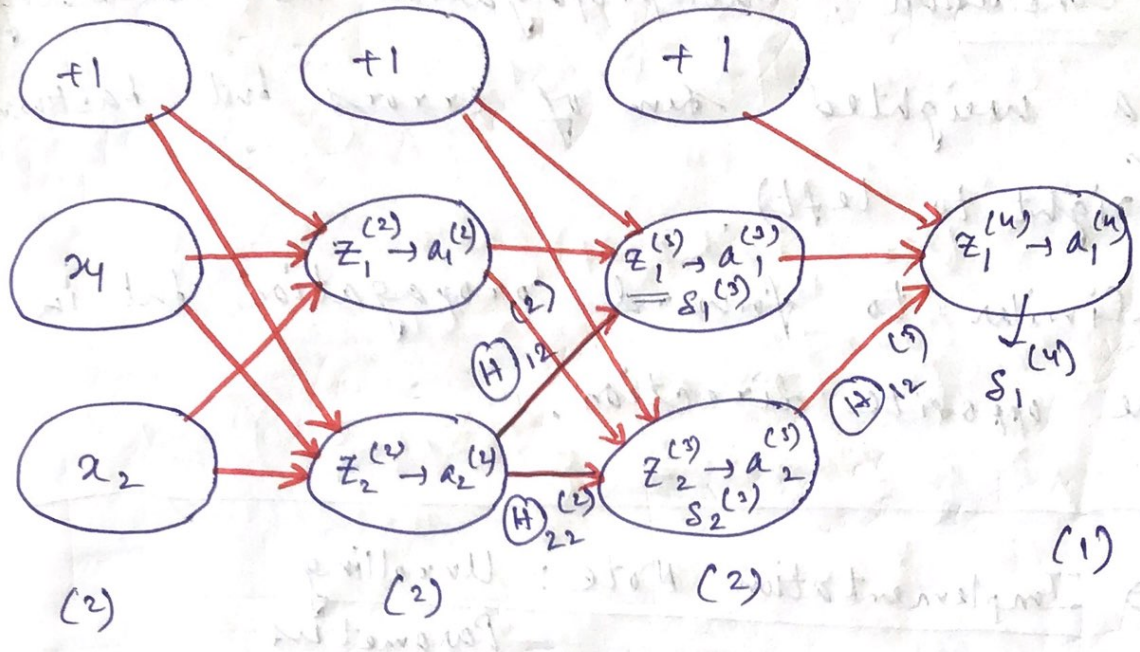
For one training example $(x^{(i)}, y^{(i)})$

→ think Cost (i)

$$= y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\Theta(x^{(i)}))$$

similar to $\left(h_\Theta(x^{(i)}) - y^{(i)}\right)^2$

→ Cost (i) gives an idea regarding how well the network is doing on example i. ?



→ $\delta_j^{(l)}$ = "error" of ~~node j in layer l~~.
cost for $a_j^{(l)}$ (unit j in layer l)

→ [Formally. $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(i)$ (for $j \geq 0$)

$\text{cost}(t) = y^{(t)} \log h_\Theta(x^{(t)}) + (1-y^{(t)}) \log (1-h_\Theta(x^{(t)}$

$\delta_1^{(3)} = \frac{\partial}{\partial z_1^{(3)}} \text{cost}(t)$

→ basically $\delta_j^{(l)}$ measures the impact on of a change in $z_j^{(l)}$ on cost(t)] $t = 1, \ldots, m$

→ ~~$\delta_1^{(4)} = a_1^{(4)} - y$~~

$\delta_1^{(4)} = y^{(i)} - a_1^{(4)}$ $\quad \delta^{(4)} = a_1^{(4)} - y$

→ $\delta_2^{(2)} = (H)_{12}^{(2)} \times \delta_1^{(3)} + (H)_{22}^{(2)} \times \delta_2^{(3)}$

→ Conclusion : Backpropagation is calculating a weighted sum of errors but backward (right to left).

Similar to forward propagation but in the opposite direction.

→ Implementation Note : Unrolling Parameters

Parameter $\Theta^{(1)}$, $\Theta^{(2)}$ ... into a big vector so that it can be used into advanced optimisation algorithms.

From vectors → reshape to get the original matrices

Neural networks learning : Gradient Checking

$J(\theta) = \theta^3$

$$\frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon} = \frac{(1.01)^3 - (0.99)^3}{2 \times (0.01)}$$

$$= \frac{0.060002}{0.02} \approx 3.0001$$

Gradient interpretation : $\frac{d}{dx} f(x) \approx \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\varepsilon}$

ε is very small.

$\varepsilon \approx 1.e-04$

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

$$\frac{\partial J(\theta)}{\partial \theta_i} = \approx \frac{J(\theta_1, \ldots, \boxed{\theta_i + \varepsilon}, \ldots, \theta_n) - J(\theta_1, \ldots, \boxed{\theta_i - \varepsilon}, \ldots, \theta_n)}{2\varepsilon}$$

→ unrolled version of $(H)^{(1)}$, $(H)^{(2)}$, $(H)^{(3)}$

Numerically compute of derivative

```
for i = 1 : n,
    thetaPlus = theta
    thetaPlus(i) = theta(i) + Epsilon;
    thetaminus = theta
    thetaminus(i) = theta(i) - Epsilon.
    gradApprox(i) = J(θ+) - J(θ-)
                    ---------------
                        2 * ε
End,
```

Check if gradApprox ≈ DVec (from backprop)

↓ gradient checking.

Note: After gradient checking, turn ~~off~~ it off. (disable gradient checking code)

→ Use backpropagation code for computing gradients.

→ Numerical gradient computation is very slow & inefficient.

→ Random Initialization # symmetry breaking

rand(m,n) → initialise matrix with all values $\in [0,1]$

$\epsilon \in [-\epsilon, \epsilon]$

$r \in [0,1]$

$(2\epsilon) * r \approx 0 - r$

$2\epsilon r < 2\epsilon$

$2\epsilon r - 2\epsilon$

$= 2\epsilon(r-1)$ → (0,1)

$r = 0.5$ $\quad$ $r \neq 0$

$r = 0.5$

$\epsilon = 0.001$

$2\epsilon = 0.002$

$2\epsilon r = 0.002 \times 0.5$

$= 0.0010 - 0.001$

$= 0.$

$r = 0.1$

$x \in [0,1]$

$2(0.0001) \times 1$

-1

$[0, 2\epsilon] - \epsilon$

$(-0.001, 0.001)$

$\epsilon \in [-\epsilon, \epsilon]$

0.002×0.5

0.0010

-0.001

→ Putting it together

Training a neural network

→ Pick a network architecture (connectivity pattern between neurons)

No. of input units: dimension of features $x^{(i)}$

No. of output units: Number of classes.

Hidden layers ??

Reasonable default: 1 hidden layer, or if > 1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

→ Steps in training a neural network -

1. Randomly initialise weights $\Theta^{(l)}$
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backpropagation to compute partial derivatives $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$