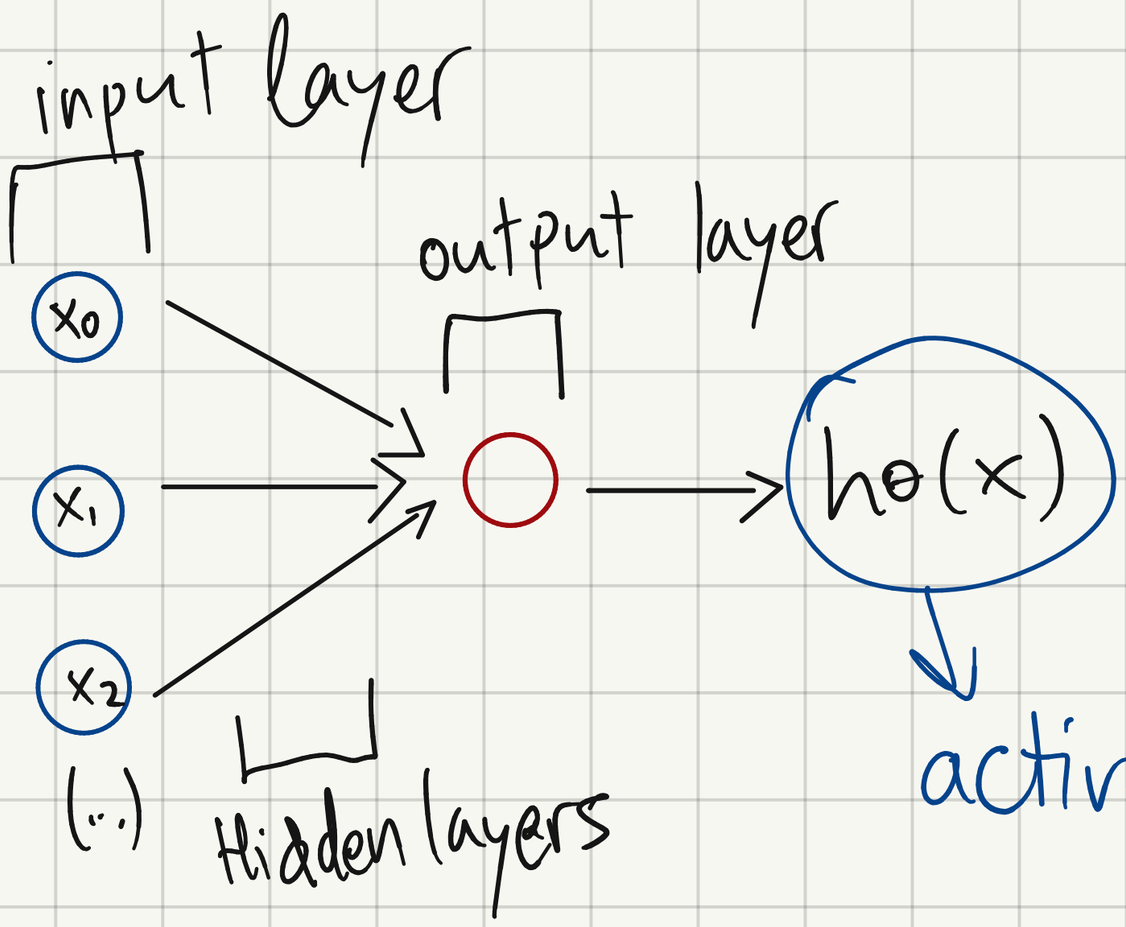
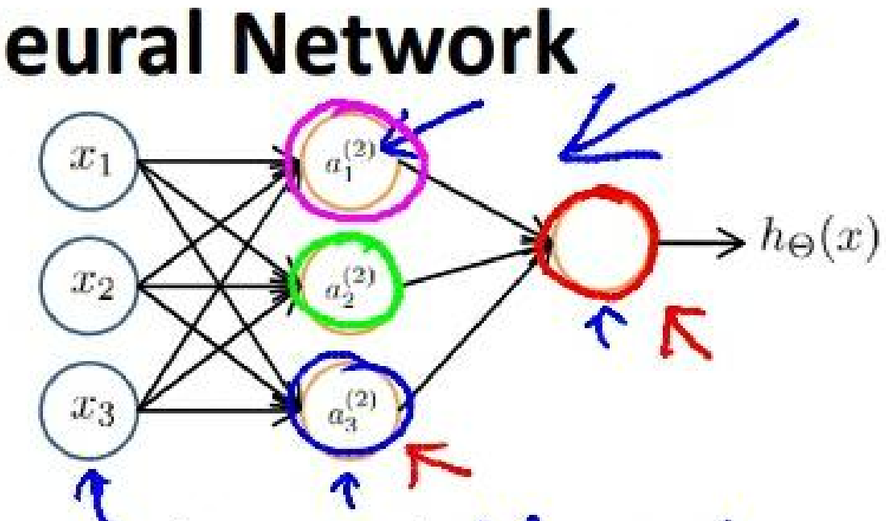


$(x_0 = 1 \quad \text{always})$

Neural Network



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

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

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

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

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

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

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

$\Theta^{(2)}$

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

$s_{j+1} \times (s_j + 1)$

So, for each value of activation, we compute:

$$\left[a_i^{(j)} = g\left(\sum_{k=0}^{m} \Theta_{ik}^{(j)} x_k \right) \right] ; \Theta^{(j)} \in \mathbb{R}^{s_{j+1} \times (s_j + 1)}$$

because we have to take into account the bias parameter x_0 to the input layers.

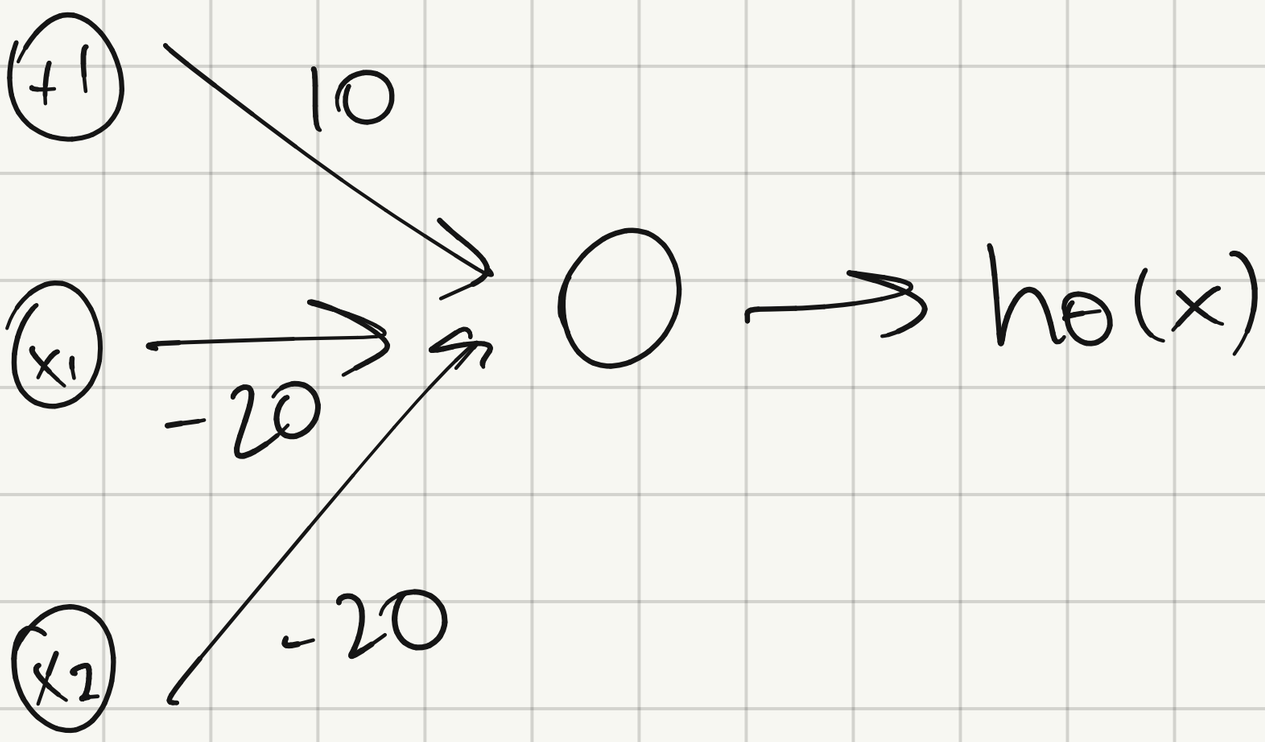
Forward propagation:

Each summation from $a_i^{(j)} = g(\ \ )$ will be renamed as $z_i^{(j)}$, so we have:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad z^{(j)} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_n \end{bmatrix}; \quad z^{(j)} = \Theta^{(j-1)} x, \quad a^{(j)} = g(z^{(j)})$$

Having finally $\rightarrow$ $\boxed{a^{(j)} = g(z^{(j)}) = g(\Theta^{(j-1)} a^{(j-1)})}$

- Example: (not x_1) AND (not x_2)



Truth table:

x_1	x_2	$h_\theta(x)$
0	0	1
0	1	0
1	0	0
1	1	0

$$h_\theta(x) = g(10 - 20x_1 - 20x_2)$$

x_1	x_2	$h_\theta(x)$
0	0	1
0	1	0
1	0	0
1	1	0

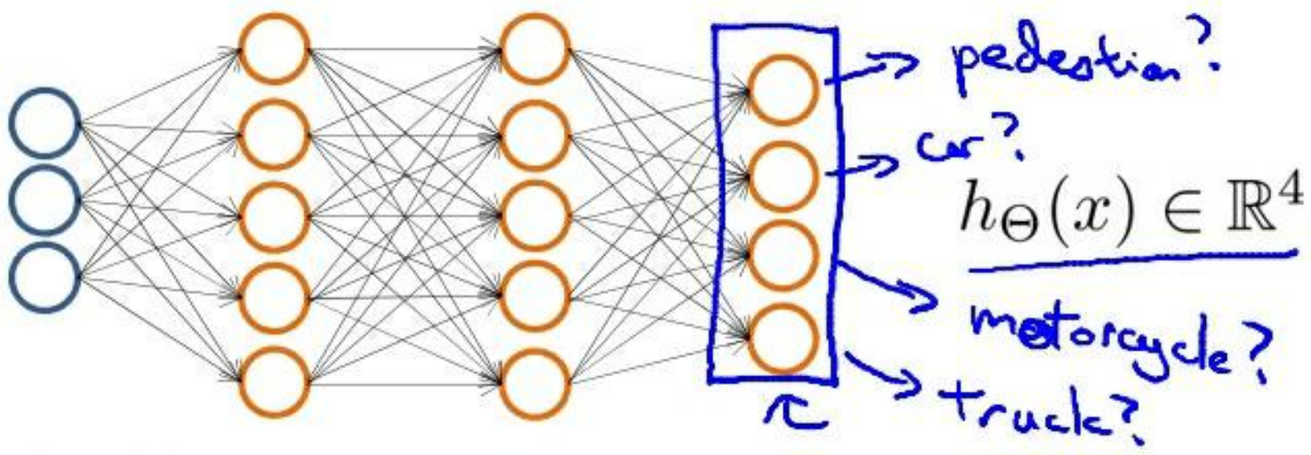
✓

x_1 XNOR x_2: $(x_1 \text{ AND } x_2) \text{ OR } ((\text{NOT } x_1) \text{ AND } (\text{NOT } x_2))$

For multiclass classification:

Multiple output units: One-vs-all.

Pedestrian Car Motorcycle Truck

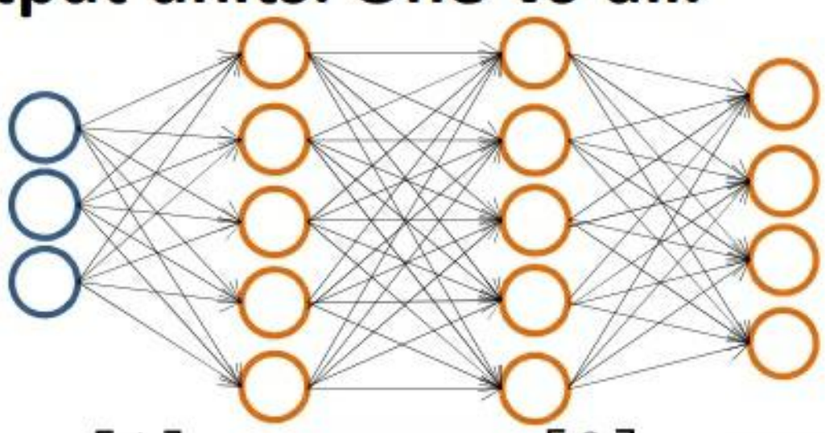


Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

Andre

Multiple output units: One-vs-all.



$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

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

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\1\end{bmatrix}$

pedestrian car motorcycle truck

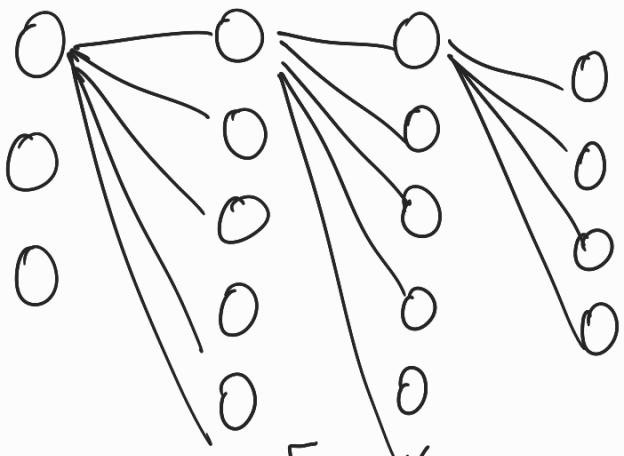
$(x^{(i)}, y^{(i)})$

Previously: $y \in \{1, 2, 3, 4\}$

$h_\Theta(x^{(i)}) \approx y^{(i)}$

$\in \mathbb{R}^4$

Cost function:



L = nº layers

S_L = nº neurons by layer

Classification → $h_\theta(x) \in \mathbb{R}^1$ (binary)

→ $h_\theta(x) \in \mathbb{R}^K$ (multi)

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\theta(x^{(i)})\right)_k + \left(1-y_k^{(i)}\right)\log\left(1-\left(h_\theta(x^{(i)})\right)_k\right)\right]$$

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(\theta_{ji}^{(l)}\right)^2$$

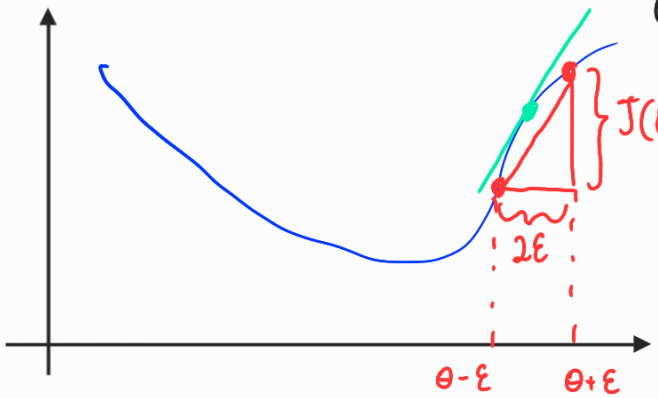
Minimize cost function: → Find $J(\theta)$

→ Find partial derivative of $J(\theta)$

BACKPROPAGATION

$\delta_j^{(l)}$ = "error" of node j in layer l. $= a_j^{(l)} - y_j \Rightarrow \delta^{(l)} = a^{(l)} - y$

We compute this backwards.



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

$(\varepsilon \approx 10^{-4})$

General case: $\theta \in \mathbb{R}^n$; $\frac{d}{d\theta_n}J(\theta) \approx \frac{J(\ldots \theta_{n-1}, \theta_n+\varepsilon, \theta_{n+1}\ldots) - J(\ldots \theta_{n-1}, \theta_n-\varepsilon, \ldots)}{2\varepsilon}$

Back propagation Algorithm

Given training set $\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$

- Set $\Delta_{i,j}^{(l)}$:= 0 for all (l,i,j), (hence you end up having a matrix full of zeros)

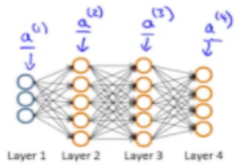
For training example t =1 to m:

1. Set $a^{(1)} := x^{(t)}$
2. Perform forward propagation to compute $a^{(l)}$ for l=2,3,...,L

Gradient computation

Given one training example (x, y):
Forward propagation:

$a^{(1)} = x$
$z^{(2)} = \Theta^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)})$ $(\text{add } a_0^{(2)})$
$z^{(3)} = \Theta^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(3)})$ $(\text{add } a_0^{(3)})$
$z^{(4)} = \Theta^{(3)} a^{(3)}$
$a^{(4)} = h_\Theta(x) = g(z^{(4)})$



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

Where L is our total number of layers and $a^{(L)}$ is the vector of outputs of the activation units for the last layer. So our "error values" for the last layer are simply the differences of our actual results in the last layer and the correct outputs in y. To get the delta values of the layers before the last layer, we can use an equation that steps us back from right to left:

4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$

The delta values of layer l are calculated by multiplying the delta values in the next layer with the theta matrix of layer l. We then element-wise multiply that with a function called g', or g-prime, which is the derivative of the activation function g evaluated with the input values given by $z^{(l)}$.

The g-prime derivative terms can also be written out as:

$$g'(z^{(l)}) = a^{(l)} .* (1 - a^{(l)})$$

5. $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ or with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

Hence we update our new Δ matrix.

- $D_{i,j}^{(l)} := \frac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)} \right)$, if j≠0.
- $D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)}$ If j=0

The capital-delta matrix D is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative. Thus we get $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$

How to choose initial θ?

- Zeros ⇒ weights of each node change, but all get same values. Consequently, the range of combinations is limited.

- Random in range $[-\varepsilon, \varepsilon]$ ⇒ being ε a value close to zero

Conclusion

1st: pick an architecture.

- N° input: dimension of features (number of columns)
- N° output: n° classes $\begin{bmatrix} 1 \\ 0 \\ 0 \\ \dots \end{bmatrix}$
- N° hidden layers: usually 1
- N° hidden units: the more the better

2nd: train neural network

1 - Randomly init weights.

2 - Forward propagation: get $h_\Theta(x^{(i)})$ $\forall$ $x^{(i)}$

3 - Compute $J(\Theta)$

4 - Backpropagation for getting $\frac{d}{d\Theta^{(e)}_{jk}} J(\Theta)$

5 - Compare derivatives of backpropagation vs num.estimate (slow)

6 - Gradient descent