- Representation

LR might not be the scalable solution for large problems with non-linear hypothesis

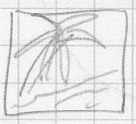
Why NN? not enough with LR?

when fitting or classifying non-linear problems the number of features might increase a lot:

the problem becomes too large :(

$x_1, \dots, x_{100}$ original features ($n = 100$)

- quadratic features → $x_1^2, x_1x_2, x_1x_3, \dots, x_1x_{100}, x_2^2, x_2x_3, \dots$ — $O(n^2) \approx 5000$ features
- cubic features → $x_1^3, x_1x_2x_3, x_1^2x_2, \dots$ — ≈ 170.000 features, $O(n^3)$



Images is another example where LR is not enough because each pixel is treated as a feature...

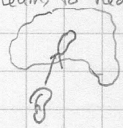
50×50 px → 2500 features (7500 if RGB) → $n = 2500$ → quadratic features ≈ 3 milion :(

? → but, LR also follows this principle, right? so this "one learning algorithm" description could also apply to LR...

Neurons and brain

The brain is based on the "one learning algorithm" hypothesis because part is prepared to learn from any input signals.

auditory cortex learns to hear



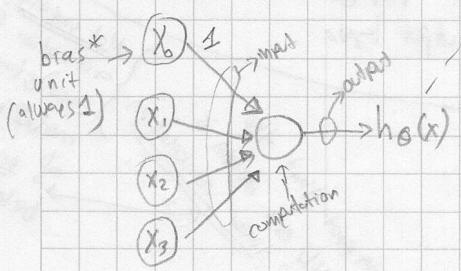
auditory cortex learns to see

we cut the ear connection and plug the eye instead

this is the idea of NN, one algorithm to fit all problems.

Model Representation

Single neuron model



$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} \quad \text{where} \quad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ and } \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

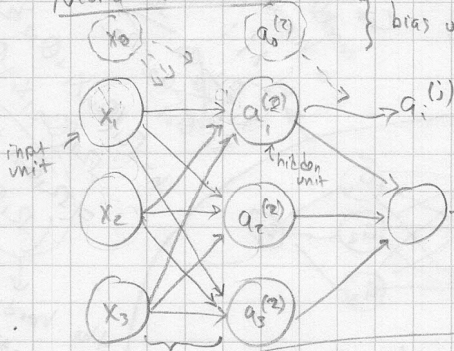
(sigmoid; or logistic, activation function)

I guess it is a sigmoid () to model 'activation' of that unit.

* the bias unit is sometimes not drawn ($x_0 = 1$ ...)

? Is Logistic Regression a single neuron?

Neural Networks



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

$$\begin{cases} a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\ a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\ \vdots \end{cases}$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \dots)$$

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

dimension of $\Theta^{(j)}$ is $s_{j+1} \times (s_j + 1)$

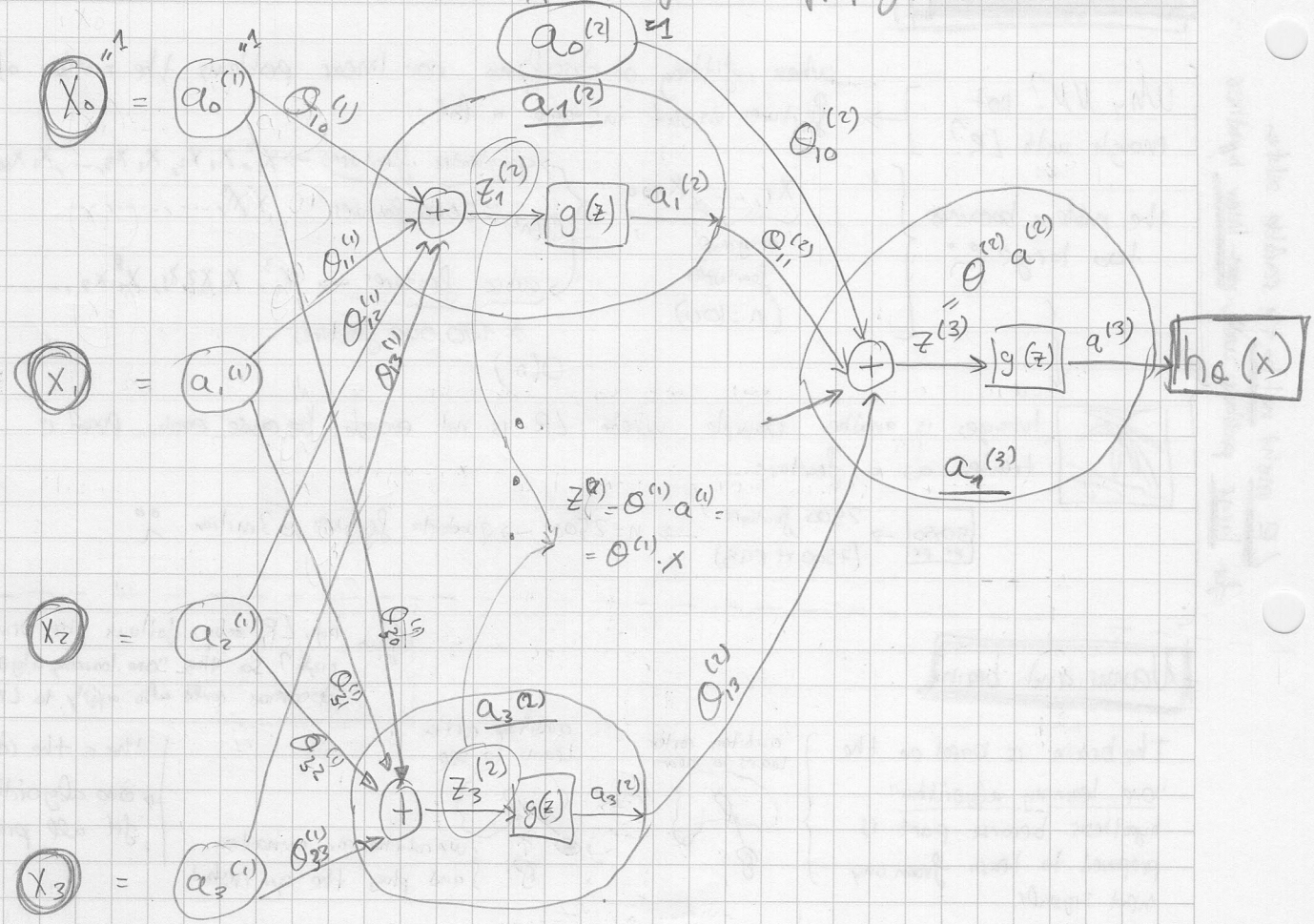
- s_{j+1}: # units in layer $j+1$
- s_j: # units in layer j

e.g. dim of $\Theta^{(1)}$: $3 \times (3+1)$ (not counting a_0; not counting x_0)

A more detailed diagram

→ to show vectorization implementation

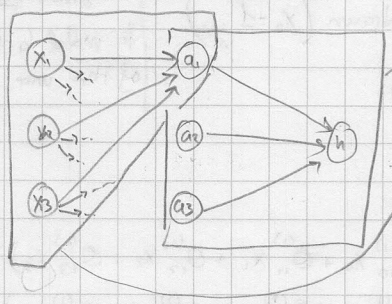
→ to explain 'forward propagation'



Forward Propagation → start with activation of input unit and propagate to hidden layer (computing activations of hidden items), and forward propagate to compute the activation of the output layer

Using $g(z)$ helps with the idea of activating each layer element

Interpretation of steps



this is just applying logistic regression where features are $a^{(2)}$

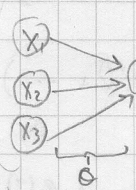
this is just applying logistic regression where features are x_i and output is $a_1^{(2)}$

I interpret that the NN can obtain a hierarchical linear combination of x_i

Not exactly! The use of activation functions at each step allows for modelling of non linear behavior!

$h = \Omega_1(\Theta_{11}x_1 + \Theta_{12}x_2) + \Omega_2(\Theta_{21}x_1 + \Theta_{22}x_2) =$
$= (\Omega_1\Theta_{11} + \Omega_2\Theta_{21})x_1 + (\Omega_1\Theta_{12} + \Omega_2\Theta_{22})x_2$

the hard thing is to compute the right parameter values.

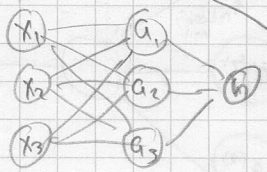


original logistic regression

⇒ We add a hidden layer in the middle and the features of this hidden layer are learnt by the neural network

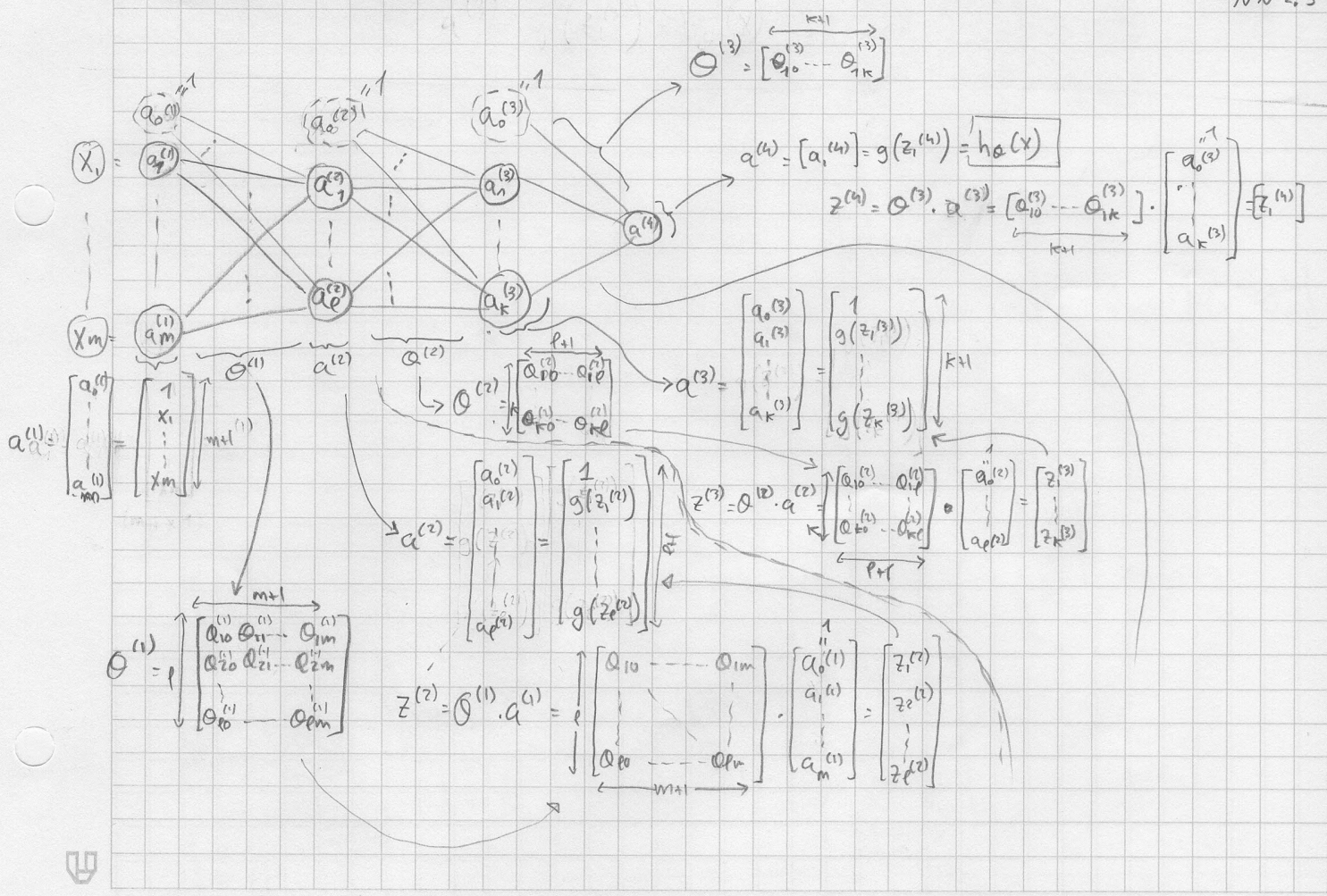
↓

this is the nice thing about NN

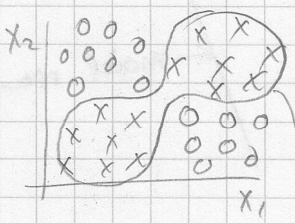


⇒ allows for more complex combinations, but better hypothesis

WRONG! I missed to consider the activation functions at each $a_j^{(i)}$


$\Theta^{(3)} = \begin{bmatrix} \Theta_{10}^{(3)} & \cdots & \Theta_{1K}^{(3)} \end{bmatrix}$
$a^{(4)} = \left[a_1^{(4)}\right] = g\left(z_1^{(4)}\right) = h_\Theta(x)$
$z^{(4)} = \Theta^{(3)} \cdot a^{(3)} = \begin{bmatrix} \Theta_{10}^{(3)} & \cdots & \Theta_{1K}^{(3)} \end{bmatrix} \cdot \begin{bmatrix} a_0^{(3)} \\ \vdots \\ a_K^{(3)} \end{bmatrix} = \left[z_1^{(4)}\right]$
$a^{(3)} = \begin{bmatrix} a_0^{(3)} \\ a_1^{(3)} \\ \vdots \\ a_K^{(3)} \end{bmatrix} = \begin{bmatrix} 1 \\ g(z_1^{(3)}) \\ \vdots \\ g(z_K^{(3)}) \end{bmatrix}$
$\Theta^{(2)} = \begin{bmatrix} \Theta_{10}^{(2)} & \cdots & \Theta_{1P}^{(2)} \\ \Theta_{K0}^{(2)} & \cdots & \Theta_{KP}^{(2)} \end{bmatrix}$
$z^{(3)} = \Theta^{(2)} \cdot a^{(2)} = \begin{bmatrix} \Theta_{10}^{(2)} & \cdots & \Theta_{1P}^{(2)} \\ \Theta_{K0}^{(2)} & \cdots & \Theta_{KP}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} a_0^{(2)} \\ \vdots \\ a_P^{(2)} \end{bmatrix} = \begin{bmatrix} z_1^{(3)} \\ \vdots \\ z_K^{(3)} \end{bmatrix}$
$a^{(2)} = \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ \vdots \\ a_P^{(2)} \end{bmatrix} = \begin{bmatrix} 1 \\ g(z_1^{(2)}) \\ \vdots \\ g(z_P^{(2)}) \end{bmatrix}$
$\Theta^{(1)} = \begin{bmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \cdots & \Theta_{1m}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \cdots & \Theta_{2m}^{(1)} \\ \vdots & & & \vdots \\ \Theta_{P0}^{(1)} & \cdots & & \Theta_{Pm}^{(1)} \end{bmatrix}$
$z^{(2)} = \Theta^{(1)} \cdot a^{(1)} = \begin{bmatrix} \Theta_{10} & \cdots & \Theta_{1m} \\ \vdots & & \vdots \\ \Theta_{P0} & \cdots & \Theta_{Pm} \end{bmatrix} \cdot \begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_m^{(1)} \end{bmatrix} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ \vdots \\ z_P^{(2)} \end{bmatrix}$
$a^{(1)} = \begin{bmatrix} a_0^{(1)} \\ \vdots \\ a_m^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_m \end{bmatrix}$
x_1
x_m
$a_0^{(1)}$
$a_1^{(1)}$
$a_m^{(1)}$
$a_0^{(2)}$
$a_1^{(2)}$
$a_P^{(2)}$
$a_0^{(3)}$
$a_1^{(3)}$
$a_K^{(3)}$
$a^{(4)}$
$\Theta^{(1)}$
$\Theta^{(2)}$
$a^{(2)}$
$a^{(3)}$
$m+1$
$P+1$
$K+1$

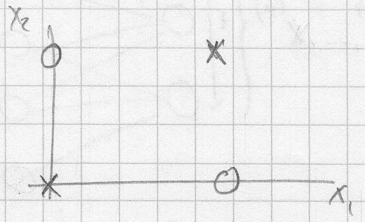
Example and intuition



$x \to y = 1$
$0 \to y = 0$

→ difficult for LR, NN can manage it...

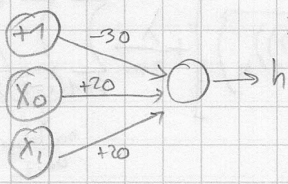
simplification →



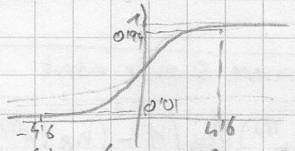
x_1 XNOR x_2
(1 only if $x_1 == x_2$)

He reviews how to model different operators using NN, but in all cases he gives the parameters, so it is more an exercise to understand how NN compute values (forward propag.) rather than understanding how to come up with Θ.

AND



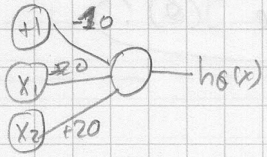
$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$ →



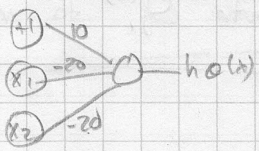
← remember this is $P(y=1 \mid x_1, x_2)$

x_1	x_2	$h_\Theta(x)$	
0	0	$g(-30) \approx 0$	x_1 AND x_2
0	1	$g(-10) \approx 0$	
1	0	$g(-10) \approx 0$	
1	1	$g(10) \approx 1$	

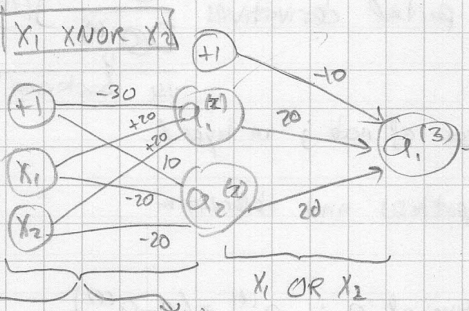
OR



(NOT x_1) AND (NOT x_2)



x_1 XNOR x_2

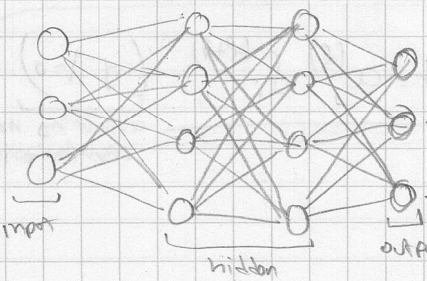


x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

✓

Multiple classification

(same idea as in one-vs-all)

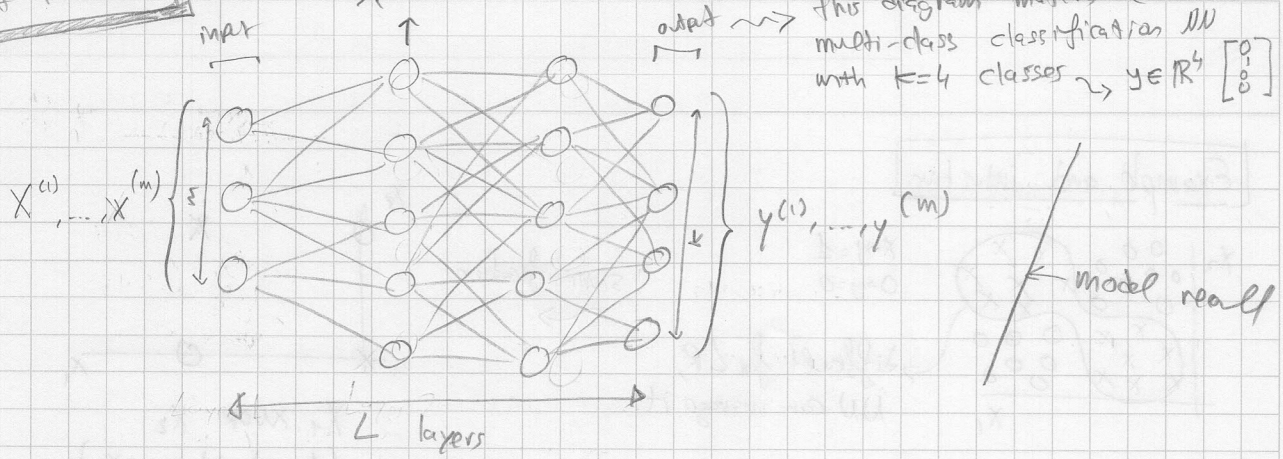


→ pedestrian → $h_\Theta(x) = \begin{bmatrix}1\\0\\0\end{bmatrix}$

→ car → $h_\Theta(x) = \begin{bmatrix}0\\1\\0\end{bmatrix}$

→ motorcycle → $h_\Theta(x) = \begin{bmatrix}0\\0\\1\end{bmatrix}$

Cost Function



The cost function will be a generalization of the Log. Reg cost function:

Logistic Regression → $J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log\left(1-h_\theta(x^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta(x^{(i)})\right)_k + (1-y_k^{(i)}) \log\left(1-(h_\Theta(x^{(i)}))_k\right)\right] + \frac{\lambda}{2m}\sum_{\ell=1}^{L-1}\sum_{i=1}^{s_\ell}\sum_{j=1}^{s_{\ell+1}} \left(\Theta_{ji}^{(\ell)}\right)^2$$

- cost of single sample & single output
- $\sum_{i=1}^{m}$: all m samples
- $\sum_{k=1}^{K}$: all K outputs
- $y_k^{(i)}$: k^{th} output observed
- $(h_\Theta(x^{(i)}))_k$: k^{th} output predicted
- $\sum_{i=1}^{s_\ell}$: no bias unit
- $\Theta_{ji}^{(\ell)}$: j = from layer $\ell+1$, i = to layer ℓ; layer ℓ

↓

but how to compute $\Theta_{ij}^{(\ell)}$ to minimize $J(\Theta)$?

Back propagation Algorithm

To minimize $J(\Theta)$, we need $J(\Theta)$ (already have it!) and the partial derivatives $\frac{\partial}{\partial \Theta_{ij}^{(\ell)}} J(\Theta)$ (use back prop. alg.)

the algorithm is based on computing $\delta_j^{(\ell)}$ ⇒ error of node j in layer ℓ.

- $\delta_j^{(L)} = a_j^{(L)} - y_j$ ⇒ difference between hypothesis and observation
- $\delta^{(L)} = a^{(L)} - y$ ← vectorized
- $\delta^{(\ell)} = (\Theta^{(\ell)})^T \delta^{(\ell+1)} \,.*\, g'(z^{(\ell)})$ → derivative of g: $a^{(\ell)} .* (1-a^{(\ell)})$

($.*$ = element-wise)

(no $\delta^{(1)}$ → layer 1 is observations, no error...)

Having $\delta_j^{(\ell)}$, it can be proved that $\frac{\partial}{\partial \Theta_{ij}^{(\ell)}} J(\Theta) = a_j^{(\ell)} \delta_i^{(\ell+1)}$ (if $\lambda = 0$ — assuming no regularization)

↓

actually, $\delta_j^{(\ell)} = \frac{\partial}{\partial z_j^{(\ell)}} \text{cost}(i)$,

so it measures how the cost changes when we change $z_j^{(\ell)}$, and the change z by changing Θ...

the algorithm

Set $\Delta_{ij}^{(l)} = 0$

For $i=1$ to m ← loop all training set

Set $a^{(1)} = x^{(i)}$ ← introduce input observations

Forward propagation to compute $a^{(2)}, a^{(3)}, \dots a^{(L)}$ |← but with which $\Theta_{ij}^{(l)}$???

Compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Back propagation to compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$, ~~$\delta^{(1)}$~~

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ ← accumulate the partial derivative terms in δ

$$D_{ij}^{(l)} := \begin{cases} \frac{1}{m}\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}, & \text{if } j \neq 0 \\ \frac{1}{m}\Delta_{ij}^{(l)}, & \text{if } j = 0 \end{cases}$$

$$\boxed{D_{ij}^{(l)} = \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)} \quad |\rightarrow \text{so we have the partial derivative of the cost function!!!}$$

if initialized to 0, all a end up computing the same function

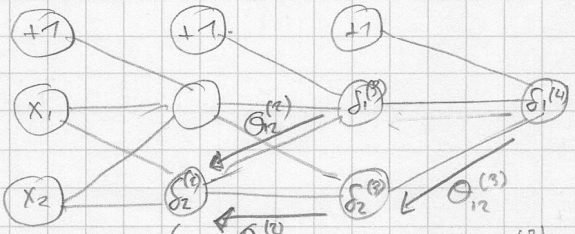
↑

usually initialized randomly to $[-\varepsilon, \varepsilon]$. There is a later video about this issue.

↑

the intuition

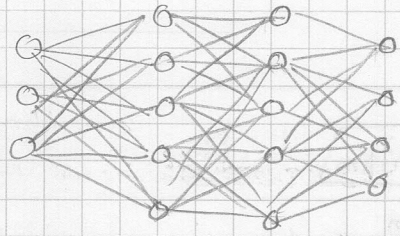
It is just like forward propagation, but backwards...



$\rightarrow \delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}$

$\delta_2^{(2)} = \Theta_{12}^{(2)} \cdot \delta_1^{(3)} + \Theta_{22}^{(2)} \cdot \delta_2^{(3)}$

} but what about $g'(z)$ in the previous page ???



A. Pick a network architecture

input units = dimensions of x

output units = number of classes

hidden layers = default 1 (if >1, have same number of units per layer)

hidden units = slightly larger than # input units (2-3-4 times is ok)

B. Train the neural network

1. Randomly initialize weyths
2. Forward Propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute $J(\Theta)$
4. Back Prop to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

(5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{ij}}$ from back prop vs. numerical estimate of gradient of $J(\Theta)$)

6. Use gradient descent or advanced optimization to minimize $J(\Theta)$ as a function of Θ.