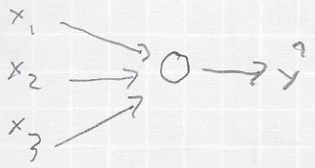
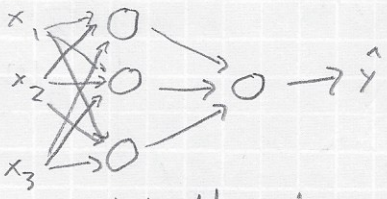
Week 4 Deep NN

Shallow vs Deep is a matter of degree.

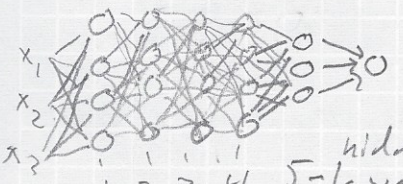


logistic regression "shallow"



"2 layer"

1 hidden layer



"deep"

5-hidden layers

There are functions that very deep NN can learn that more shallow models cannot perform. It can be hard to predict the number of hidden layers that will work the best. So it can be good to try logistic regression, 1 layer, 2 layer, etc and treat the number of hidden layers as another hyperparameter to evaluate against.

Notation

L = # of layers

$n^{(l)}$ = # of nodes (units)

$n^{[0]} = n_x$

$a^{[l]}$ = activations in layer l

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

$w^{[l]}$ = weights for $z^{[l]}$

$b^{[l]}$ = biases for $z^{[l]}$

$x = a^{[0]}$

$a^{[L]} = \hat{y}$ $a^{[L]}$ = is the prediction of the final layer

Forward Propagation in a Deep NN

For a single training example x

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$
$$\dots$$
$$z^{[4]} = W^{[4]}a^{[3]} + b^{[4]}$$
$$a^{[4]} = g^{[4]}(z^{[4]}) = \hat{y}$$

$g^{[l]}$ has a superscript because each layer could have a different activation function

general forward propagation equation

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$

Vectorizing forward propagation across all training examples

$$Z^{[1]} = W^{[1]}X + b^{[1]}, \quad X = A^{[0]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\ldots$$
$$\hat{Y} = g(Z^{[4]}) = A^{[4]}$$

general form of vectorized forward prop

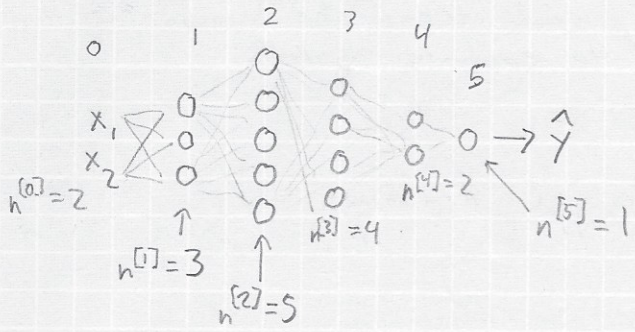
$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

this is a for loop over l, don't think there is a way to get rid of this for loop from 1 to L this is okay to have a " "

To avoid errors and debug it is helpful to think systematically about your matrix dimensions.

Getting your dimensions right

Parameter $W^{[l]}$ and $b^{[l]}$



$L = 5$

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

$(3,1) \quad (2,1)$
$(n^{[1]},1) \quad (n^{[0]},1)$

$$\begin{bmatrix} \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} ? \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$$

$$\begin{bmatrix} \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots \\ \vdots & \vdots \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$$

$(3,2)$
$(n^{[1]}, n^{[0]})$

W = The dimension of the next layer comma the dimension of the previous layer.

- $Z^{[1]}$ is the vector of activations for the 1st hidden layer = 3
- we know we have 2 input features for X
- so we need a W that when multiplied by a (2,1) matrix X, we get a (3,1) activations vector $Z^{[1]}$
- the rules of matrix multiplication tell us that in this case we need a (3, 2) matrix

$$W^{[1]} = (n^{[1]}, n^{[0]})$$
$$W^{[2]} = (5,3) \quad (n^{[2]}, n^{[1]})$$
$$Z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$
$(5,1) \quad (5,3) \quad (3,1)$

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

dimensions for $W^{[l]} : (n^{[l]}, n^{[l-1]})$

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

we just determined that the dot product of these two terms will be a (3,1) dimensional matrix so to perform matrix addition $b^{[1]}$ must be the same (3,1).

$$b^{[l]} : (n^{[l]}, 1)$$

In back propagation the dimensions of dW should be the same as W

$$dW^{[l]} : (n^{[l]}, n^{[l-1]})$$

same with b and db

$$db^{[l]} : (n^{[l]}, 1)$$

Vectorized Implementation

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

$(n^{[1]}, m)$ $(n^{[1]}, n^{[0]})$ (same) $(n^{[0]}, m)$

because we stack across all examples m when you multiply X with dimension $(n^{[0]}, m)$ by $(n^{[1]}, n^{[0]})$ you will get a matrix of size $(n^{[1]}, m)$ which is what we want

$b^{[1]}$ is still $(n^{[1]}, 1)$ but through python broadcasting it will be duplicated into an $(n^{[1]}, m)$ matrix and then added element-wise.

$$z^{[l]}, a^{[l]} : (n^{[l]}, 1)$$

$$Z^{[l]}, A^{[l]} : (n^{[l]}, m)$$

special case is when $l=0$ $A^{[0]} = X = (n^{[0]}, m)$

(same) $dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$

Why Deep Representation? (are they effective)

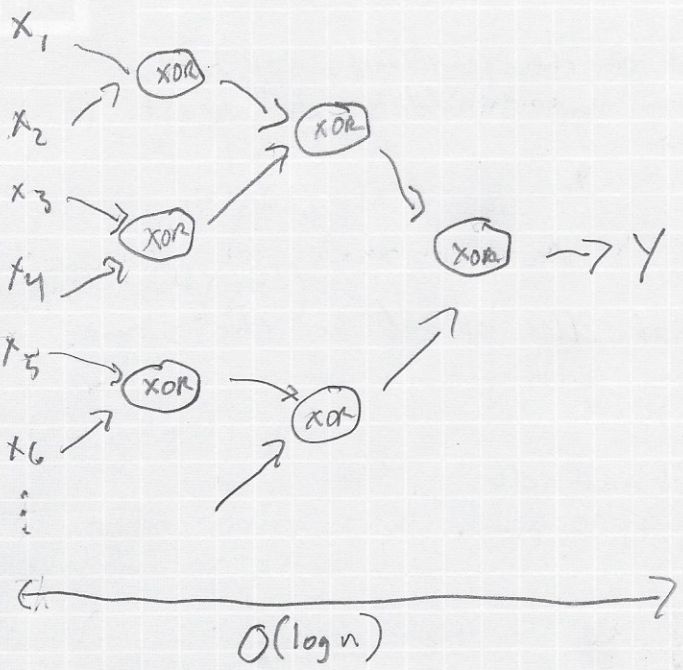
the first hidden layer will work on simple problems then will use those to build or compose more complex functions in the later layers

waves → sounds / phonemes → words → sentence / phrase

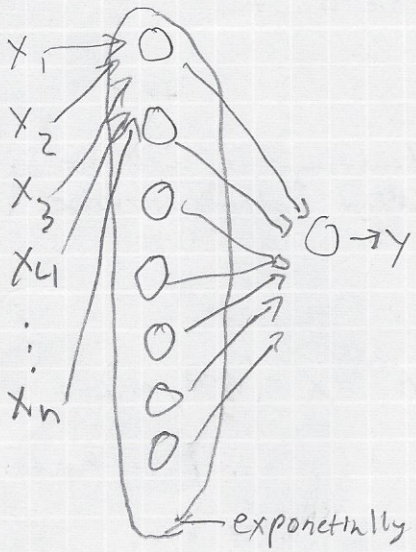
edges → facial features → faces

Using circuit theory (logic gates) you can show that more layers is superior to more nodes. 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.

Problem: $y = x_1 \text{ XOR } x_2 \text{ XOR } x_3 \ldots \text{XOR } x_n$



vs



this is a 2^{n-1} operation which is $O(2^n)$

which is obviously far costlier
helps explain the value of <u>deep</u> NN

"Deep Learning" is great branding, but deep networks do work well. But still A.N. still usually starts with L.R. then 2 layers, then more using L as a hyperparameter.