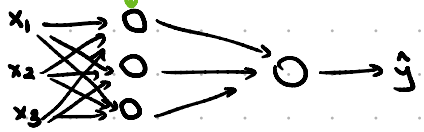
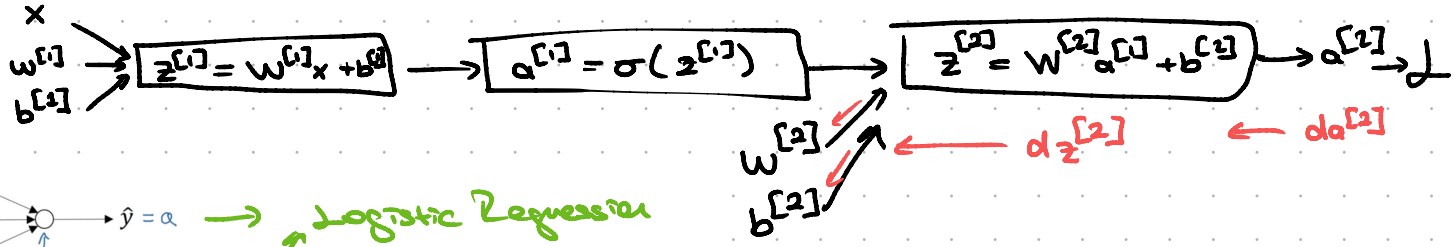
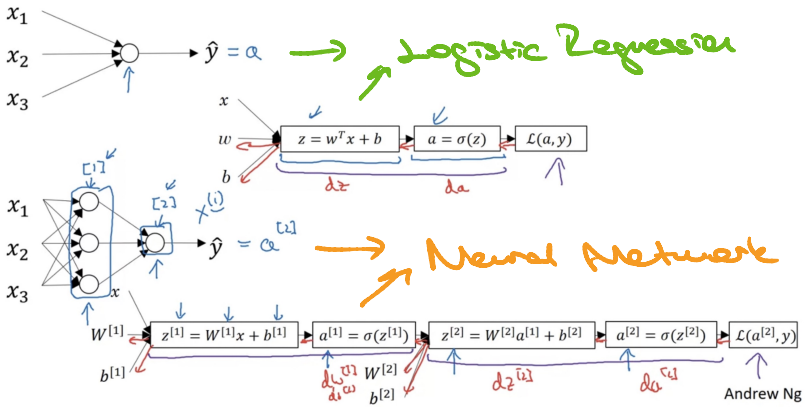
Neural Networks Overview

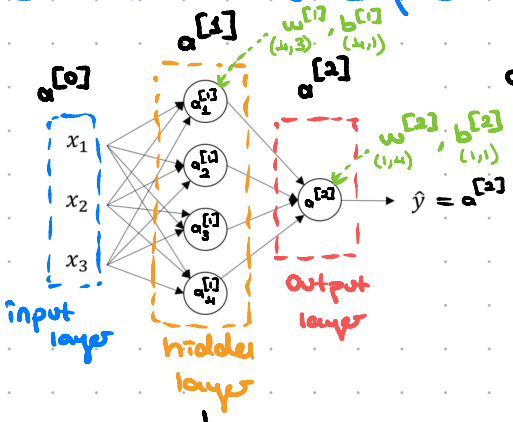
$[1]$ ⇒ new notation, superscript square brackets $x^{[i]}$ ⇒ ith layer







Neural Network Representation



$a^{[0]}$ = input vector x (alternative naming convention)

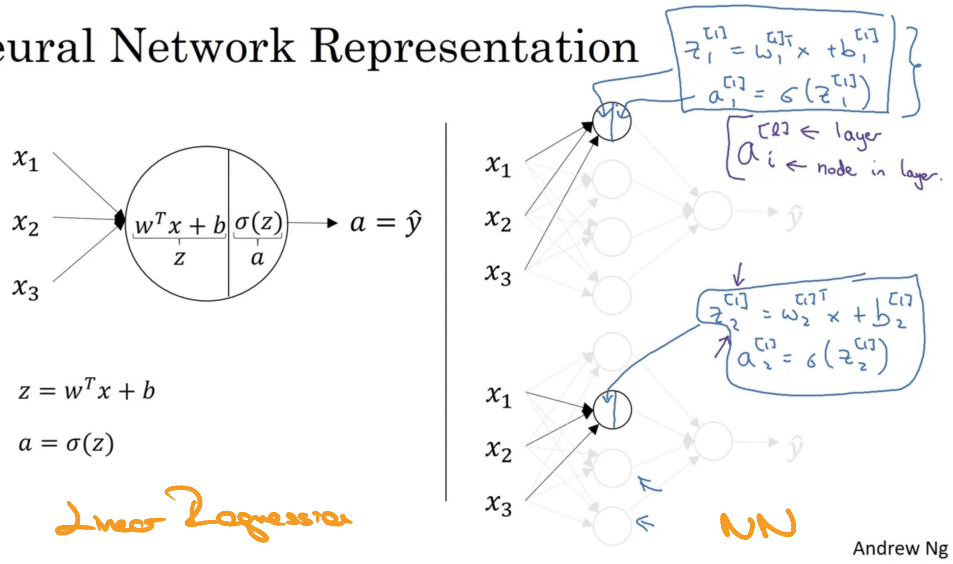
This is a "2 layer NN" ⇒ we don't count input layer

Every layer, except input layer, will have parameters associated with it.

↓

In training set, we don't see what hidden layer should be BUT we see input + output layer values

Computing a NN Output



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

} We need to vectorize this!!

Let $W^{[1]}$ be a matrix such that:

$$W^{[1]} = \begin{bmatrix} — & w_1^{[1]T} & — \\ — & w_2^{[1]T} & — \\ & \vdots & \\ — & w_{n_x}^{[1]T} & — \end{bmatrix}$$

Note

All of this is for a single training example.

Then,

$$Z^{[1]} = \underbrace{\begin{bmatrix} — & w_1^{[1]T} & — \\ — & w_2^{[1]T} & — \\ & \vdots & \\ — & w_{n_x}^{[1]T} & — \end{bmatrix}}_{W^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]}} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ \cdots \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

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

$b^{[1]}$ $(4,1)$

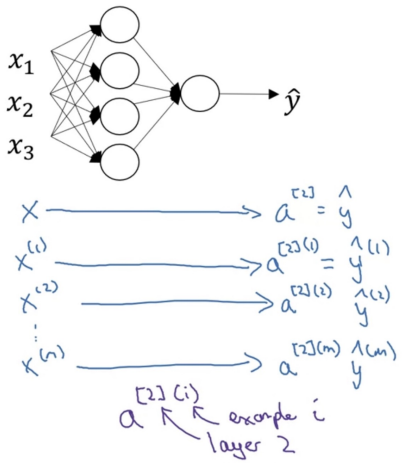
$$a^{[1]} = \sigma(Z^{[1]})$$

Given input x:

$$z^{[1]}_{(4,1)} = W^{[1]}_{(4,3)} x_{(3,1)} + b^{[1]}_{(4,1)}$$
$$a^{[1]}_{(4,1)} = \sigma(z^{[1]}_{(4,1)})$$
$$z^{[2]}_{(1,1)} = W^{[2]}_{(1,4)} a^{[1]}_{(4,1)} + b^{[2]}_{(1,1)}$$
$$a^{[2]}_{(1,1)} = \sigma(z^{[2]}_{(1,1)})$$

} only 1 training example

Vectorizing Across Multiple Examples



without vectorization:

for $i = 1$ to m, (we want to get rid of this)

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$
$$\hat{y}^{(i)} = a^{[2](i)}$$

RECALL

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}_{(n_x, m)}$$

THEN

$$Z^{[1]}_{(4,m)} = W^{[1]}_{(4,n_x)} X_{(n_x,m)} + b^{[1]}_{(4,1)}$$

(4 = hidden units, m = training example)

$$A^{[1]}_{(4,m)} = \sigma(Z^{[1]}_{(4,m)})$$

$$Z^{[2]}_{(1,m)} = W^{[2]}_{(1,4)} A^{[1]}_{(4,m)} + b^{[2]}_{(1,1)}$$

$$A^{[2]}_{(1,m)} = \sigma(Z^{[2]}_{(1,m)})$$

$$Z^{[1]} = \begin{bmatrix} — & w_1^{[1]T} & — \\ — & w_2^{[1]T} & — \\ & \cdots & \\ — & w_{n_x}^{[1]T} & — \end{bmatrix} \begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix} + b^{[1]} = \begin{bmatrix} w_1^{[1]} x^{(1)} & | & | \\ w_2^{[1]} x^{(1)} & | & | \\ \vdots & W^{[1]} x^{(2)} + b^{[1]} & W^{[1]} x^{(m)} + b^{[1]} \\ w_4^{[1]} x^{(1)} & | & | \end{bmatrix}$$

(training examples across; hidden units down)

Activation Functions

Sigmoid function is an activation function
- goes between 0 and 1

tanh ⇒ almost always better than sigmoid
- goes between −1 and +1
- mathematically a shifted version of sigmoid
- better as mean of activation is zero, which makes training easier
  - ↳ ONE EXCEPTION is the output activation
    - → if output is {0,1}, sigmoid might be better.

$$a = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

NOTE your NN can have different activations at different layers. ⇒ $g^{[1]}$: activation at layer 1

PROBLEM if z is either very large or very small, gradient of tanh or sigmoid function is very small, which slows down linear regression.

Rectified Linear Unit (RELU) $a = \max(0, z)$
- problem is derivative is 0 on the left
- advantage : slope is large so train very fast

Leaky ReLU ⇒  small slope at left $a = \max(0.01z, z)$

General Rules
- if output is {0,1} value, sigmoid is natural choice for output
- ReLU is very popular for hidden units

Why Do You Need Activation Functions?

$g(z) = z$: "linear activation function"

if we use a linear activation function:

$$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$= (\underbrace{W^{[2]}W^{[1]}}_{W'})x + (\underbrace{W^{[2]}b^{[1]} + b^{[2]}}_{b'})$$

multiple linear transformations is still one linear transformation

$$= W'x + b' \longrightarrow \text{this is just a linear function}$$

the NN is just outputting a linear function, then

$\Rightarrow$ no matter # of layers we will still calculate a linear function

$\Rightarrow$ might as well have 1 layer. $\Rightarrow$ USELESS

THE ONLY TIME we may want to use a linear activation function

$\Rightarrow$ if we do ML on a REGRESSION problem

Derivatives of Activation Functions

Sigmoid:

$$\frac{d}{dz} g(z) = g'(z) = g(z)(1 - g(z))$$

$$= a \cdot (1 - a)$$

if $z = 10$ $g(z) \approx 0$

if $z = -10$ $g(z) \approx 0$

(as $g(z) = a$)

if we calculated a already, super easy to find $g'(z)$

tanh

$$g'(z) = 1 - (\tanh(z))^2 = 1 - a^2$$

ReLU

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \\ \text{undef} & \text{if } z = 0 \end{cases}$$

Leaky ReLU

$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \\ \text{undef} & \text{if } z = 0 \end{cases}$$

Gradient Descent for Neural Networks

input features → $n_x = n^{[0]}$, hidden units → $n^{[1]}$, output units → $n^{[2]} = 1$

parameters: $w^{[1]}$ $(n^{[1]}, n^{[0]})$, $b^{[1]}$ $(n^{[1]}, 1)$, $w^{[2]}$ $(n^{[2]}, n^{[1]})$, $b^{[2]}$ $(n^{[2]}, 1)$

cost function: (binary classification)

$$J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$$

Gradient Descent

REPEAT {

Compute predictions $(\hat{y}^{(i)}, i = 1, \dots, m)$

$dw^{[1]} = \frac{\partial J}{\partial w^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, ...

$w^{[1]} := w^{[1]} - \alpha\, dw^{[1]}$

$b^{[1]} := b^{[1]} - \alpha\, db^{[1]}$

... }

FORWARD PROPAGATION

$$z^{[1]} = w^{[1]} X + b^{[1]}$$

$$A^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = w^{[2]} A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(z^{[2]})$$

BACK PROPAGATION

$$\underset{(n^{[2]}, m)}{dz^{[2]}} = \underset{(1,m)}{A^{[2]}} - \underset{(1,m)}{Y}$$

$$\underset{(n^{[2]}, n^{[1]})}{dw^{[2]}} = \frac{1}{m} \underset{(n^{[2]}, m)}{dz^{[2]}} \underset{(m, n^{[1]})}{A^{[1]T}}$$

$$\underset{(n^{[2]}, 1)}{db^{[1]}} = \frac{1}{m} np.sum(dz^{[2]}, axis = 1, keepdims = True)$$

horizontal sum; avoids Rank 1

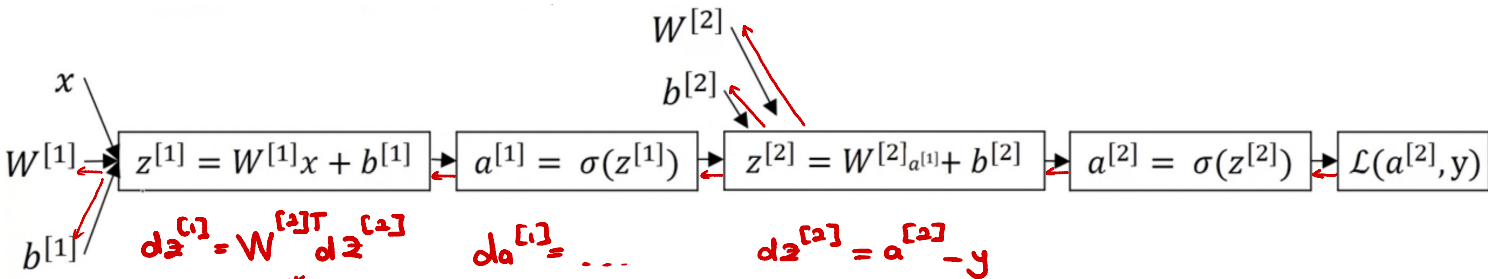
$$\underset{(n^{[1]}, m)}{dz^{[1]}} = \underbrace{w^{[2]T} dz^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]'}(z^{[1]})}_{(n^{[1]}, m)}$$

* = element-wise product

$$\underset{(n^{[1]}, n^{[0]})}{dw^{[1]}} = \frac{1}{m} \underset{(n^{[1]}, m)}{dz^{[1]}} \underset{(m, n^{[0]})}{X^T}$$

$$\underset{(n^{[1]}, 1)}{db^{[1]}} = \frac{1}{m} np.sum(dz^{[1]}, axis = 1, keepdims = 1)$$

BACKPROPAGATION CALCULATIONS



$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$ $\quad da^{[1]} = \ldots$

$dz^{[2]} = a^{[2]} - y$

$dw^{[2]} = dz^{[2]} a^{[1]T}$ ← transpose this it to column vector

$db^{[2]} = dz^{[2]}$

↓

why? remember that the $W^{[2]}$ is a matrix where each row is $w_j^{[2]T}$ (i.e. a row vector)

↑ jth hidden unit

where does this come from?

$$dz^{[1]} = \frac{\partial \mathcal{L}}{\partial z^{[1]}}$$

$$= \frac{\partial \mathcal{L}}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial z^{[1]}}$$

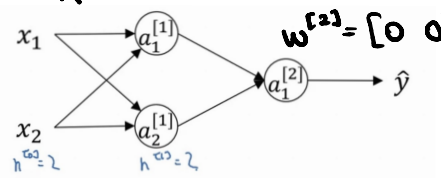
$$= \frac{\partial \mathcal{L}}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial z^{[1]}}$$

$$= \underset{(n^{[2]}, 1)}{dz^{[2]}} \cdot \underset{(n^{[2]}, n^{[1]})}{W^{[2]}} \cdot \underset{(n^{[1]}, 1)}{g^{[1]'}(z^{[1]})}$$

$$= \underset{(n^{[1]}, 1)}{W^{[2]T} dz^{[2]}} * g^{[1]'}(z^{[1]})$$

Random Initialization

what happens if we initialize weights to zero?



$w^{[2]} = [0 \; 0]$

Both hidden units are symmetric

Both compute exactly the same function after gradient descent

$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad \underset{\text{(okay)}}{b^{[1]}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

If we initialize W's to zeros, the hidden units will be and will remain to be symmetric

$a_1^{[1]} = a_2^{[1]}$ $\qquad dz_1^{[1]} = dz_2^{[1]}$

USELESS to have >1 hidden units

Therefore,

we want small values

$$W^{[1]} = np.random.randn((2,2)) * 0.01$$

$$b^{[1]} = np.zeros((1,1))$$

← Symmetry isn't a problem for b

Why do we want small weights?

→ if too large, z is too big or too small

→ we end up in flat parts of tanh or sigmoid

→ gradient is close to zero

→ gradient descent is slow