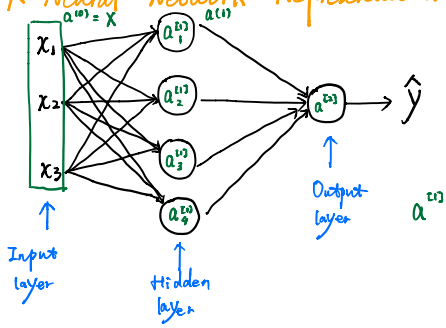
1. Neural Network Representation



2 layer network (don't count input layer)

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}, \quad a^{[0]} = X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad a^{[2]} = \hat{y}$$

2. Computing a NN's output.

e.g. $\begin{cases} z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]} \\ a_1^{[1]} = \sigma(z_1^{[1]}) \end{cases} \quad \begin{cases} z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]} \\ a_2^{[1]} = \sigma(z_2^{[1]}) \end{cases}$ ...

$a_i^{[l]}$ ← layer; ↖ node in layer $= \sigma[w_i^{[l]T} x + b_i^{[l]}]$

If we vectorize the results:

$$\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]} x + b_2^{[1]} \\ w_3^{[1]} x + b_3^{[1]} \\ w_4^{[1]} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = z^{[1]}$$

$W^{[1]}$ (4×3) $\qquad b^{[1]}$ (4×1) $\qquad z^{[1]}$ (4×1)

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

Given input X:

$$\begin{cases} z^{[1]} = W^{[1]} x + b^{[1]} \\ \quad (4\times1) \quad (4\times3)\ (3\times1) \quad (4\times1) \\ a^{[1]} = \sigma(z^{[1]}) \\ \quad (4\times1) \quad (4\times1) \\ z^{[2]} = W^{[2]} x + b^{[2]} \\ \quad (1\times1) \quad (1\times4)\ (4\times1) \quad (1\times1) \\ a^{[2]} = \sigma(z^{[2]}) \end{cases}$$

3. Vectorizing across multiple examples

$$\begin{cases} X \longrightarrow a^{[2]} = \hat{y} \\ x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)} \\ x^{(2)} \longrightarrow a^{[2](2)} = \hat{y}^{(2)} \\ \vdots \\ x^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)} \end{cases}$$

$a^{[2](i)}$: ↖ layer 2, ← example i

for $i = 1$ to m,

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

$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$ $n_x \times m$ (training examples $\rightarrow$, hidden units $\downarrow$)

$$\begin{cases} z^{[1]} = W^{[1]} X + b^{[1]} \\ A^{[1]} = \theta(z^{[1]}) \\ z^{[2]} = W^{[2]} A^{[1]} + b^{[2]} \\ A^{[2]} = \theta(z^{[2]}) \end{cases}$$ ($W^{[1]}X$: $n_x \times m$; layers $\times$ n_x)

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

4. Explanation for vectorized implementation

$z^{[1](1)} = w^{[1]} x^{(1)} + b^{[1]}$, $z^{[1](2)} = w^{[1]} x^{(2)} + b^{[1]}$, $z^{[1](3)} = w^{[1]} x^{(3)} + b^{[1]}$ (3 training examples)

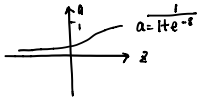
$w^{[1]} = \begin{bmatrix} — \\ — \\ — \end{bmatrix}$, $w^{[1]} x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$, $w^{[1]} x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$, $w^{[1]} x^{(3)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$

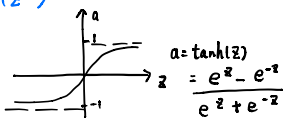
$$w^{[1]} \begin{bmatrix} | & | & | & \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots \\ | & | & | & \end{bmatrix} = \begin{bmatrix} | & | & | & \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots \\ | & | & | & \end{bmatrix} = Z^{[1]}$$

More training examples, more columns.

5. Activation functions

$a^{[1]} = \theta(z^{[1]}) \rightarrow a^{[1]} = g(z^{[1]})$





For the input layer to hidden layer, tangent function always works much better.

However, if you want to do binary classification, since y should $\in [0,1]$, this time we use sigmoid function.

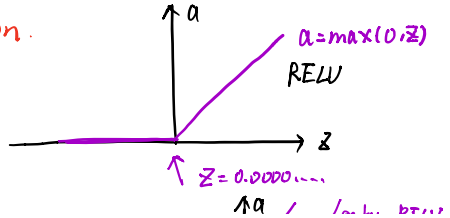
$$\therefore \begin{cases} a^{[1]} = g(z^{[1]}) = \tanh(z^{[1]}) \\ a^{[2]} = \theta(z^{[2]}) \quad \text{(sigmoid)} \end{cases} \rightarrow \begin{cases} a^{[1]} = g^{(1)}(z^{[1]}) = \tanh(z^{[1]}) \\ a^{[2]} = g^{(2)}(z^{[2]}) = \theta(z^{[2]}) \end{cases}$$

Activation functions differ with the different layers.

One downside of sigmoid & tangent function is that:

if z is very large / very small, the gradient descent will be very slow

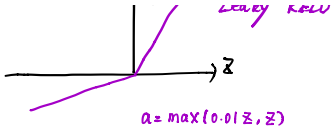
Here is another activation function: RELU function.



Some rules about activation functions:

1) Output is binary classification $\in \{0,1\}$, output layer use sigmoid function

2) other layers use RELU function, more efficient.



6. Why do we need non-linear activation functions?

e.g. for $a = g(z)$, we just use the linear activation, $g(z) = z$.

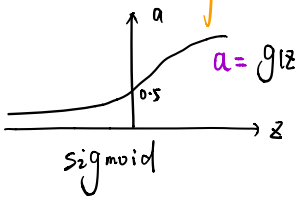
$$a^{[1]} = z^{[1]} = w^{[1]}x + b^{[1]}$$
$$a^{[2]} = z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$$
$$= w^{[2]}(w^{[1]}x + b^{[1]}) + b^{[2]}$$
$$= w^{[2]}w^{[1]}x + (w^{[2]}b^{[1]} + b^{[2]})$$
$$= w'x + b'$$

If we use linear activation function, the output just the linear combination of input.

∴ No matter how many hidden layers the NN have, it will always compute the linear combination of inputs. Then the hidden layers may useless.

Hint: Here is an exception when we need linear functions.
If you are doing machine learning on the regression problem.
Then y is a real number. e.g. if y is a house price.
Also, since house price should always ≥ 0, we can also use RELU function.

7. Derivatives of activation functions



$a = g(z) = \frac{1}{1+e^{-z}}$

$\frac{d}{dz}g(z)$ = slope of $g(x)$ at z $\downarrow g'(z)$

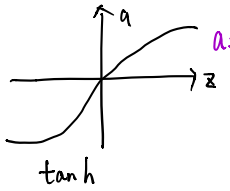
$= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$

$= g(z)(1-g(z))$

$= a(1-a)$

∴ $g'(z) = a(1-a)$

e.g. $z = 10$, $g(z) \approx 1$
slope $\approx 1 \times (1-1) \approx 0$
$z = -10$, $g(z) \approx 0$
slope $\approx 0 \times (1-0) \approx 0$
$z = 0$, $g(z) = \frac{1}{2}$
slope $\approx \frac{1}{2} \times (1 - \frac{1}{2}) = \frac{1}{4}$



$a = g(z) = \tanh(z)$

$g'(z) = \frac{d}{dz}g(z)$ = slope of $g(z)$ at z

$= 1 - (\tanh(z))^2$

∴ $g'(z) = 1 - a^2$

$z = 10$, $\tanh(z) \approx 1$
$g'(z) \approx 0$
$z = -10$, $\tanh(z) \approx -1$
$g(z) \approx 0$
$z = 0$, $\tanh(z) = 0$
$g'(z) = 1$

$g(z) = \max(0, z)$

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

$\downarrow z = 0.00\ldots0$

$g(z) = \max(0.01z, z)$

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

8. Gradient descent for Neural Networks

One hidden layer.

Parameters: $w^{[1]}$ $(n^{[1]}, n^{[0]})$, $b^{[1]}$ $(n^{[1]}, 1)$, $w^{[2]}$ $(n^{[2]}, n^{[1]})$, $b^{[2]}$, $\rightarrow (n^{[2]}, 1)$ $\quad n_x = n^{[0]}$, $n^{[1]}, n^{[2]} = 1$

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

Gradient descent:

Repeat {
  Compute predicts $(\hat{y}^{(i)}, i = 1, 2, \cdots m)$
  $dw^{[1]} = \frac{dJ}{dw^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$, ...
  $w^{[1]} = w^{[1]} - \alpha dw^{[1]}$
  $b^{[1]} = b^{[1]} - \alpha db^{[1]}$
  ⋮
  Until parameters reach Convergence }

Formulas for computing derivatives.

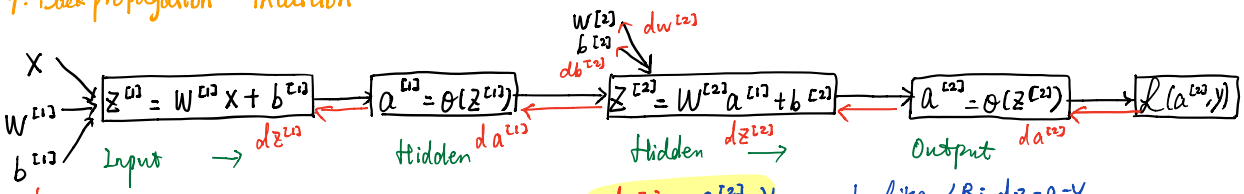
Forward propagation:

$z^{[1]} = w^{[1]} x + b^{[1]}$
$A^{[1]} = g^{[1]}(z^{[1]})$
$z^{[2]} = w^{[2]} A^{[1]} + b^{[2]}$
$A^{[2]} = g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$

Back propagation

$dz^{[2]} = A^{[2]} - Y$ $\quad Y = [y^{(1)}, y^{(2)}, \cdots y^{(m)}]$
$dw^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$
$db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis=1, keepdims=True)$
$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$ → make sure array $(n^{[2]}, 1)$
$\quad (n^{[1]}, m) \quad (n^{[1]}, m)$
$dw^{[1]} = \frac{1}{m} dz^{[1]} X^T$
$db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1, keepdims=True) \rightarrow (n^{[1]}, 1)$

9. Back propagation intuition



$dw^{[1]} = dz^{[1]} \cdot X^T$
$db^{[1]} = dz^{[1]}$

$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$
$(n^{[1]}, 1) = (n^{(1)}, n^{[2]}) \times (n^{[2]}, 1)$
$* \ (n^{[1]}, 1) \times (n^{[1]}, 1)$
element-wise product 元素积

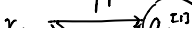
$dz^{[2]} = a^{[2]} - y$
$dw^{[2]} = dz^{[2]} a^{[1]T}$
$db^{[2]} = dz^{[2]}$
$W^{[1]} = [\quad\quad]$

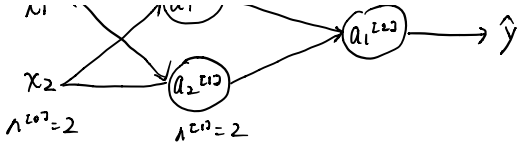
Like LR: $dz = a - y$
$dw = \frac{1}{m} dz X^T$
$db = \frac{1}{m} np.sum(dz)$

For dimension,
$n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$
$w^{[2]} : (n^{[2]}, n^{[1]})$
$z^{[2]}, dz^{[2]} : (n^{[2]}, 1) \rightarrow (1, 1)$
$w^{[1]}, dz^{[1]} : (n^{[1]}, 1)$

10. Random Initialization

→ What happens if you initialize weights to 0?





If we initialize $W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Then $a_1^{[1]} = a_2^{[1]}$, $dz_1^{[1]} = dz_2^{[1]}$ → Then compelety symmetric

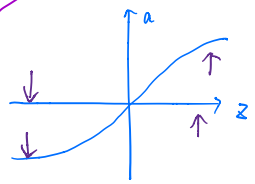
It turns out:

after every single iteration of training your two hidden units are still computing exactly the same function.

Then $dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$, $W^{[1]} = W^{[1]} - \alpha dw$, W的每一行都是相同的，所以每个 x_2 乘在的 weights 是相同的. 不管有多少 layers.

Random Initialization

e.g.
$W^{[1]} = np.random.randn((2,2)) * \frac{0.01}{100?}$
$b^{[1]} = np.zero((2,1))$
$W^{[2]} = np.random.randn((1,2)) * 0.01$
$b^{[2]} = 0$



If w is large, z is large, a自动会 large / small

gd 会变得 slow.