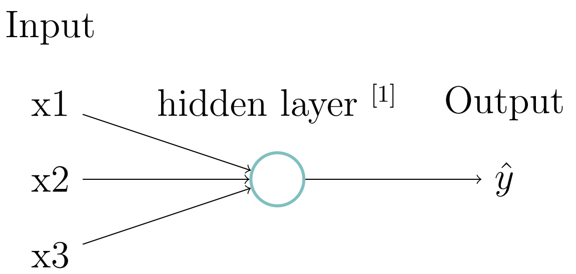
1 Week3

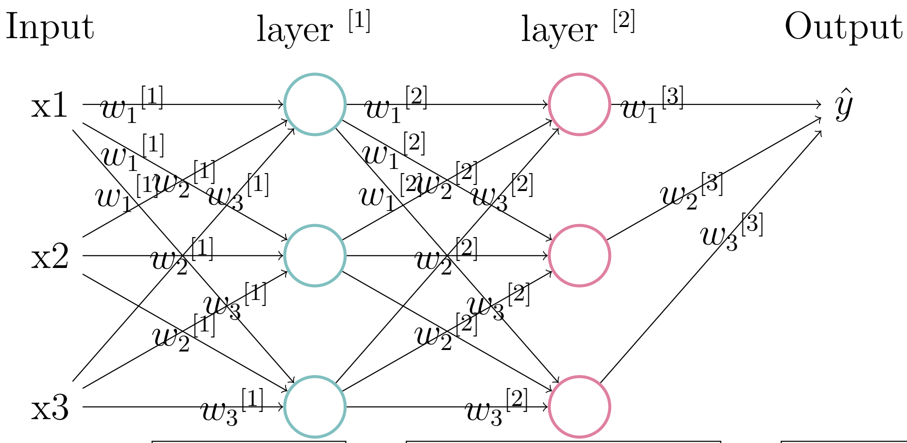
Shallow Neural Network.

1.1 Neural Network Overview

What is a Neural Network?



$\boxed{x, w, b} \rightarrow \boxed{z = w^T x + b} \rightarrow \boxed{a = \sigma(z)}$



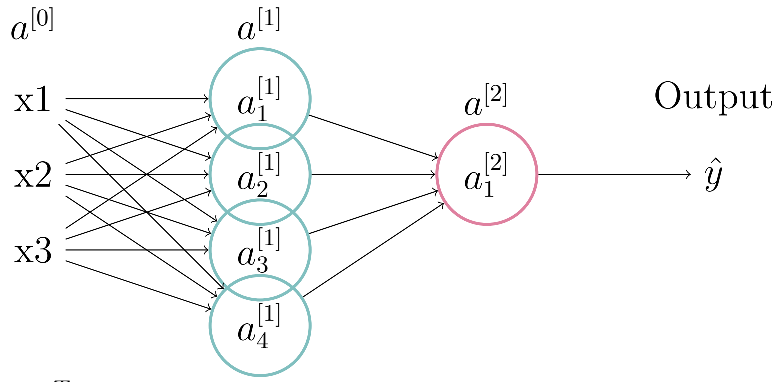
$Layer^{[1]} : \boxed{X, W^{[1]}, b^{[1]}} \rightarrow \boxed{Z^{[1]} = W^{[1]}X + b^{[1]}} \rightarrow \boxed{A^{[1]} = \sigma(Z^{[1]})}$

$Layer^{[2]} : \boxed{Z^{[2]} = W^{[2]}X + b^{[2]}} \rightarrow \boxed{A^{[2]} = \sigma(Z^{[2]})} \rightarrow \boxed{\mathcal{L}(A^{[2]}, Y)}$

1.2 Neural Network Representations

Values of the input features (activation): $X = a^{[0]}$

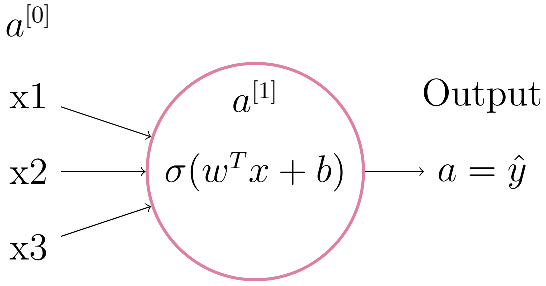
The following is the 2-Layer Neural Network:



${w^{[1]}}^T$ in $(4, 3)$, $b^{[1]}$ in $(4, 1)$

${w^{[2]}}^T$ in $(1, 4)$, $b^{[2]}$ in $(1, 1)$

1.3 Computing Neural Network Output

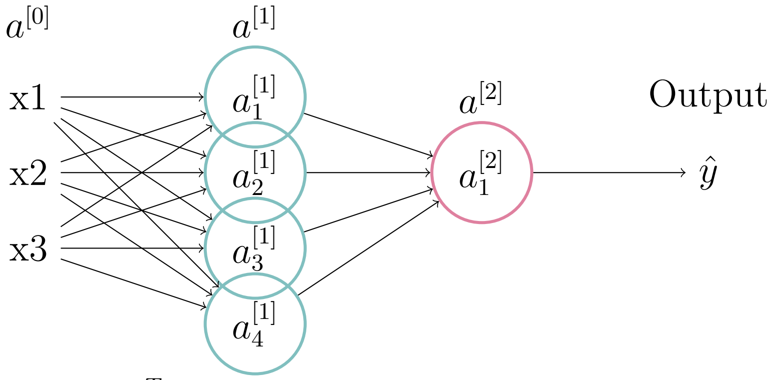


Each circle(node) represents 2 steps of calculation:

$$\boxed{z = w^T x + b}$$

$$\boxed{a = \sigma(z)}$$

1. The weighted sum of the inputs is calculated.
2. The bias is added.
3. The result is fed to an activation function.
4. Specific neuron is activated.

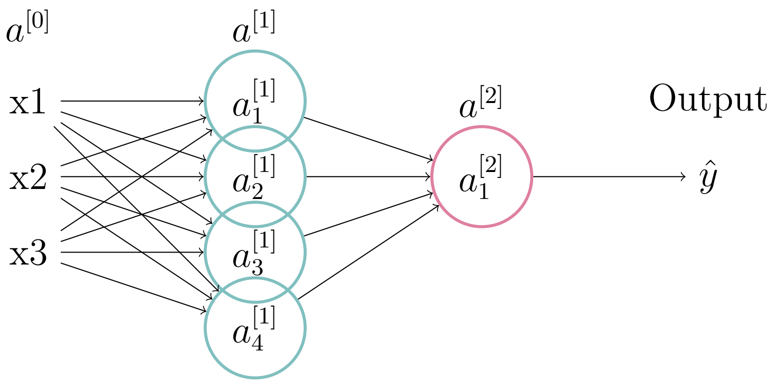


$$z_1^{[1]} = {w_1^{[1]}}^T X + b_1^{[1]} \quad \rightarrow \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = {w_2^{[1]}}^T X + b_2^{[1]} \quad \rightarrow \quad a_2^{[1]} = \sigma(z_2^{[1]}), \quad \text{where } a_i^{[l]} \begin{array}{l} \scriptstyle\leftarrow \text{layer} \\ \scriptstyle\leftarrow \text{node in layer} \end{array}$$

$$z^{[1]} = \begin{bmatrix} {w_1^{[1]}}^T X + b_1^{[1]} \\ {w_2^{[1]}}^T X + b_2^{[1]} \\ {w_3^{[1]}}^T X + b_3^{[1]} \\ {w_4^{[1]}}^T X + b_4^{[1]} \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \sigma({w_1^{[1]}}^T X + b_1^{[1]}) \\ \sigma({w_2^{[1]}}^T X + b_2^{[1]}) \\ \sigma({w_3^{[1]}}^T X + b_3^{[1]}) \\ \sigma({w_4^{[1]}}^T X + b_4^{[1]}) \end{bmatrix}$$

$$a^{[1]} = \sigma(z^{[1]}) = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \\ \sigma(z_4^{[1]}) \end{bmatrix} = \sigma[\begin{bmatrix} \dots & {w_1^{[1]}}^T & \dots \\ \dots & {w_2^{[1]}}^T & \dots \\ \dots & {w_3^{[1]}}^T & \dots \\ \dots & {w_4^{[1]}}^T & \dots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}]$$



Given input x:

$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$, where $_{(4,1)=(4,3)(3,1)+(4,1)}$

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

$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$, where $_{(1,1)=(1,4)(4,1)+(1,1)}$

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

1.4 Vectorizing Across Multiple Examples

$x^{(i)} \ \rightarrow \ a^{[2](i)} = \hat{y}^{(i)}$

for $i = 1$ to m:

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

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

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

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

For m examples:

$$X = \begin{bmatrix} | & | & | & | \\ X^{(1)} & X^{(2)} & \dots & X^{(m)} \\ | & | & | & | \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} | & | & | & | \\ z^{[1](1)} & z^{[1](2)} & \dots & z^{[1](m)} \\ | & | & | & | \end{bmatrix} = {W^{[1]}}^T X + b^{[1]}$$

$$A^{[1]} = \begin{bmatrix} | & | & | & | \\ a^{[1](1)} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & | & | \end{bmatrix} = \sigma(Z^{[1]}) = \sigma({W^{[1]}}^T X + b^{[1]})$$

$\updownarrow$: across hidden units in the i^{th} training example; $\sharp$ of units/node

$\leftrightarrow$: across m training examples

1.5 Justification for vectorized implementation

Given $m = 3$, let

$$W^{[1]}X^{(1)} = \begin{bmatrix} | \\ \text{A} \\ | \end{bmatrix}, W^{[1]}X^{(2)} = \begin{bmatrix} | \\ \text{B} \\ | \end{bmatrix}, W^{[1]}X^{(3)} = \begin{bmatrix} | \\ \text{C} \\ | \end{bmatrix}$$

$$\text{then, } Z^{[1]} = W^{[1]} \begin{bmatrix} | & | & | \\ X^{(1)} & X^{(2)} & X^{(3)} \\ | & | & | \end{bmatrix} = \begin{bmatrix} | & | & | \\ A & B & C \\ | & | & | \end{bmatrix} = \begin{bmatrix} | & | & | \\ Z^{[1](1)} & Z^{[1](2)} & Z^{[1](3)} \\ | & | & | \end{bmatrix}$$

1. Using for-loops:

for $i = 1$ to m:

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

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

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

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

2. Using Matrix(without for-loop):

Given X:

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

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

$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$

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

1.6 Activation Functions

There're other activation functions other than sigmoid and you can use different activation functions for different layers;
$\sigma(z) \;\rightarrow\; g(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

$tanh$ always works better on hidden layers than sigmoid function, except for output layer (use sigmoid for output layer) where it requires to output be either 0 or 1 (binary classification)

Use different activation functions $g^{[l]}(z^{[i]})$, where $l = 1, 2, \ldots, L$ (number of layers in the network)
Both activation functions have down side; If z is either very small or large, the gradient(slope of the function) is very small which will slow down the gradient descent(backward propagation/learning speed).

ReLU function is a popular choice for the activation function; $a = max(0, z)$ that does not have above downsides.

In case of Binary Classification, use sigmoid for the output layer, and ReLu for other hidden layer's activation function. Or Leaky ReLU ($a = max(.01z, z)$) recently.(not commonly used in practice)

1.7 Why Non-linear Activation Functions

Linear Activation through Hidden layers become meaningless. Use $g(z)$ as non-linear function.
Using linear activation for hidden layers and sigmoid for the output layer will simply become a Logistic Regression problem..

1.8 Derivatives of Activation Functions.

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

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

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

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

$ReLU$ activation function $g(z) = max(0, z)$

$g^{'}(z) = 0$ if $z < 0$

$g^{'}(z) = 1$ if $z \geq 0$

$LeakyReLU$ activation function $g(z) = max(.01z, z)$

$g^{'}(z) = .01$ if $z < 0$

$g^{'}(z) = 1$ if $z \geq 0$

1.9 Gradient Descent For Neural Networks

Backward propagation.

Parameters:

$w^{[1]}$, $b^{[1]}$, $w^{[2]}$, $b^{[2]}$ matrices with dimensions:

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

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

Cost Funcion:

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

Gradien Descent:

Repeat:
Compute predicts $\hat{y}^{(i)}$ for $i = 1, ..., m$
$dw^{[1]} = \frac{\partial J}{\partial w^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$
$w^{[1]} := w^{[1]} - \alpha dw^{[1]}$
$b^{[1]} := b^{[1]} - \alpha db^{[1]}$
$w^{[2]} := w^{[2]} - \alpha dw^{[2]}$
$b^{[2]} := b^{[2]} - \alpha db^{[2]}$

Formulas for computing derivatives:

Forward propagtion:

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

$A^{[1]} = g^{[1]}(Z^{[1]})$

$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$, sigmoid for binary classification.

Backward propagtion:

$dZ^{[2]} = A^{[2]} - Y = [y^{(1)}, \ldots, y^{(m)}]$

$dW^{[2]} = \frac{1}{m} dZ^{[2]} {A^{[1]}}^T$

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

, where keepdims ensures $(n^{[2]}, 1)$ instead of $(n,)$

$dZ^{[1]} = {W^{[2]}}^T dZ^{[2]} * {g^{[1]}}^{'}(Z^{[1]})$; elementwise product in $_{(n^{[1]}, m)}$

$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

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

1.10 BackPropagation Intuition

Logistic Regression Recap

Forward Propagation in Logistic Regression:

$$
\begin{tikzcd}
x \arrow[dr] & & & \\
w \arrow[r] & \boxed{z = w^T x + b} \arrow[r] & \boxed{a = \sigma(z)} \arrow[r] & \mathcal{L}(a, y) \\
b \arrow[ur] & & &
\end{tikzcd}
$$

Backward Propagation in Logistic Regression:

$$da = \frac{d}{da}\mathcal{L}(a, y) = \frac{d}{da}[-y \log a - (1 - y)\log(1 - a)] = -\frac{y}{a} + \frac{1-y}{1-a} = \frac{a-y}{a(1-a)}$$

$$dz = \frac{d}{dz}\mathcal{L}(a, y) = \frac{d}{da}\mathcal{L}(a, y)\frac{dz}{da} = da \cdot \frac{dz}{da} = da \cdot g^{'}(z)$$

, where $a=g(z)$ and therefore $\frac{dz}{da}=g^{'}(z)$ In case of logistic regression: $g(z)=\sigma(z)$,therefore $g^{'}(z)=g(z)(1-g(z))$

$$dw = \frac{d}{dz}\mathcal{L}(a, y)\frac{dz}{dw} = dz \cdot x$$

$$db = \frac{d}{dz}\mathcal{L}(a, y)\frac{dz}{db} = dz$$

Neural network gradients:

Compute forward:

$$\begin{tikzcd}
x \arrow[dr] & & & \\
W^{[1]} \arrow[r] & \boxed{z^{[1]} = W^{[1]}x + b^{[1]}} \arrow[r] & \boxed{a^{[1]} = \sigma(z^{[1]})} \arrow[r] & \cdots \\
b^{[1]} \arrow[ur] & {\color{red} dz^{[1]}} & {\color{red} da^{[1]}} &
\end{tikzcd}$$

$$\begin{tikzcd}
a^{[1]} \arrow[dr] & & & \\
W^{[2]} \arrow[r] & \boxed{z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}} \arrow[r] & \boxed{a^{[2]} = \sigma(z^{[2]})} \arrow[r] & \mathcal{L}(a^{[2]}, y) \\
b^{[2]} \arrow[ur] & {\color{red} dz^{[2]}, dw^{[2]}, db^{[2]}} & {\color{red} da^{[2]}} &
\end{tikzcd}$$

Compute backward $(n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1)$

$$da^{[2]} \to dz^{[2]} \to dw^{[2]}, db^{[2]} \to da^{[1]} \to dz^{[1]} \to dw^{[1]}, db^{[1]}$$

$\boxed{dz^{[2]}} = a^{[2]} - y$ (logistic regression: activation for output layer is sigmoid) $a^{[2]} = \sigma(z^{[2]})$

$\boxed{dw^{[2]}} = dz^{[2]} {a^{[1]}}^T$ (similar to logistic regression problem) $dw = dz \cdot x$

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

$\boxed{dz^{[1]}} = {w^{[2]}}^T dz^{[2]} * {g^{[1]}}'(z^{[1]})$ where $_{(n^{[1]},1)} =_{(n^{[1]},n^{[2]})} \cdot_{(n^{[2]},1)} *_{(n^{[1]},1)}$

$\boxed{dw^{[1]}} = dz^{[1]} X^T = dz^{[1]} {a^{[0]}}^T$

$\boxed{db^{[1]}} = dz^{[1]}$

Summary of gradient descent

For a simple training example: (from the previouse slide)

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

$dW^{[2]} = dz^{[2]} - {a^{[1]}}^T$

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

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

$dW^{[1]} = dz^{[1]} x^T$

$db^{[1]} = dz^{[1]}$

For m training examples:

$dZ^{[2]} = A^{[2]} - Y$

$dW^{[2]} = \frac{1}{m} dZ^{[2]} {A^{[1]}}^T$

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

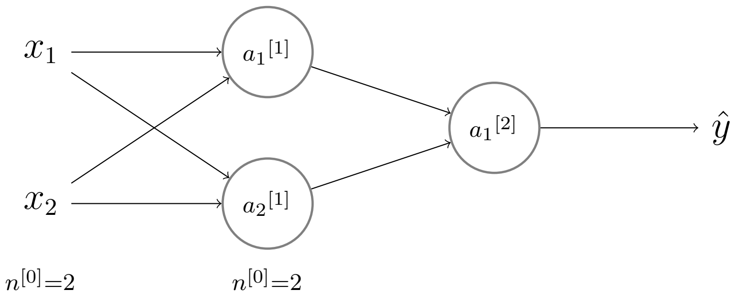
$dZ^{[1]} = {W^{[2]}}^T dZ^{[2]} * {g^{[1]}}^{'}(Z^{[1]})$, elementwise in $_{(n^{[1]}, m)}$ dimension.

$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

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

1.11 Random Initialization

What happens if you initialize weights to zero?



$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ gives $w^{[2]} = \begin{bmatrix} 0 & 0 \end{bmatrix}$ and hidden units in $^{[1]}$ are identical. So having more than one hidden units become meaningless.

$${a_1}^{[1]} = {a_2}^{[1]}$$

$${dz_1}^{[1]} = {dz_2}^{[1]}$$

Random Initialization

$$w^{[1]} = np.random.randn((2, 2)) * .01$$

$$b^{[1]} = np.zeros((2, 1))$$

$$w^{[2]} = np.random.randn((1, 2)) * .01$$

$$b^{[2]} = 0$$

(big multiplier will make slope of the gradient descent is very small, so choose it as a small value.)