

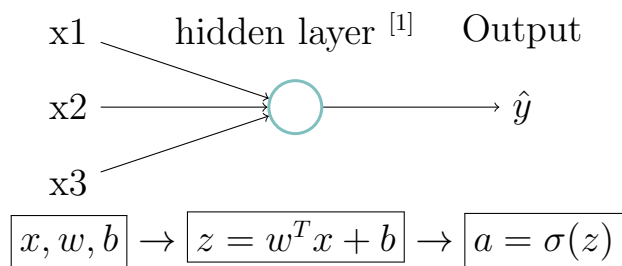
1 Week3

Shallow Neural Network.

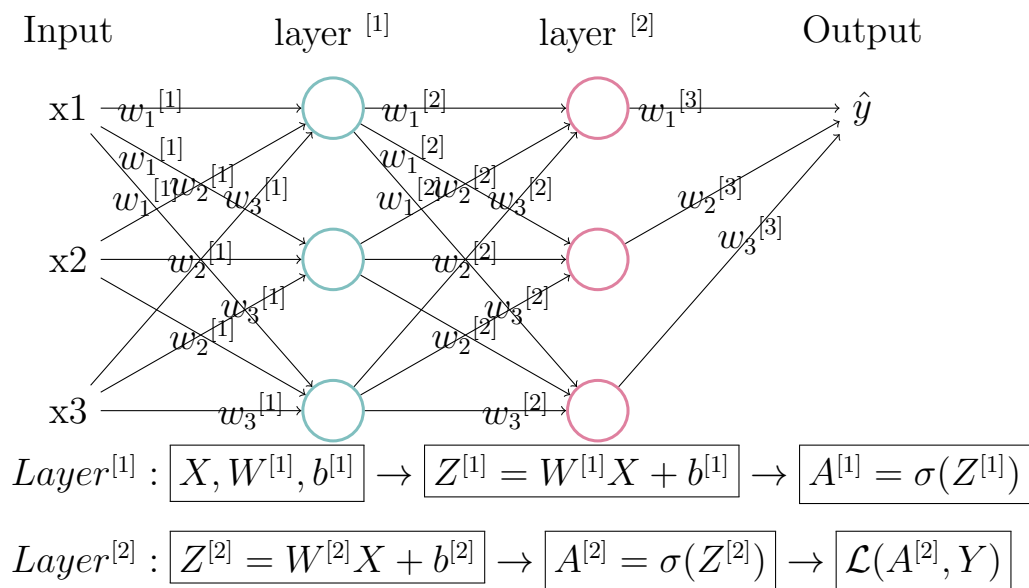
1.1 Neural Network Overview

What is a Neural Network?

Input



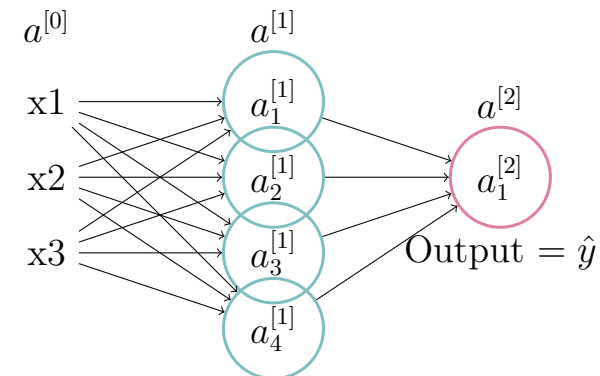
Input



1.2 Neural Network Representations

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

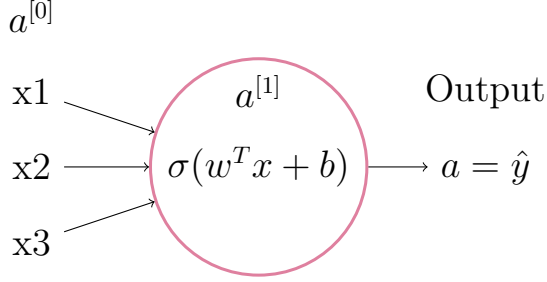
The following is the 2-Layer Neural Network:



$w^{[1]T}$ 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

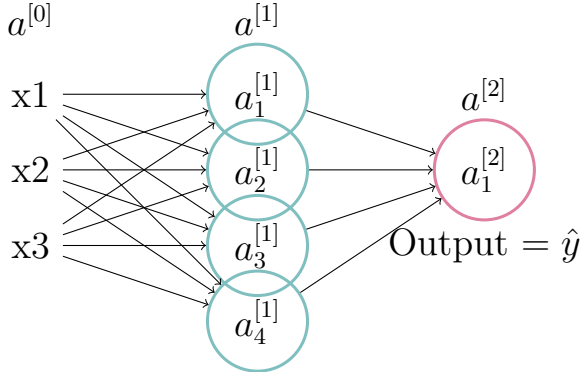


Each circle(node) represents 2 steps of calculation:

$$z = w^T x + b$$

$$a = \sigma(z)$$

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.

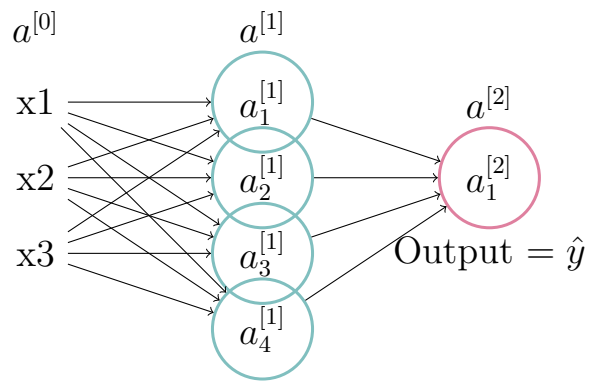


$$z_1^{[1]} = w_1^{[1]T} X + b_1^{[1]} \rightarrow a_1^{[1]} = \sigma(z_1^{[1]})$$

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

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

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



Given input x :

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

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

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

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

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:

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

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

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

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

\leftrightarrow : across m training examples

1.5 Justification for vectorized implementation

Given $m = 3$, let

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

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

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 $l = 1, 2, \dots, L$ (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 $g(z) = \max(0, z)$

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

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

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

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

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

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 Function:

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

Gradient Descent:

Repeat:

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

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

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

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

$$w^{[2]} := w^{[2]} - \alpha dw^{[2]}$$

$$b^{[2]} := b^{[2]} - \alpha db^{[2]}$$

Formulas for computing derivatives:

Forward propagation:

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

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

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

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

Backward propagation:

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

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

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

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

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

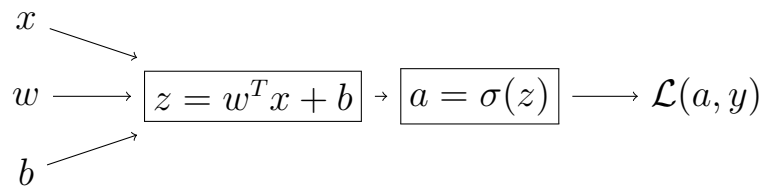
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

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

1.10 BackPropagation Intuition

Logistic Regression Recap

Forward Propagation in Logistic Regression:



Backward Propagation in Logistic Regression:

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

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

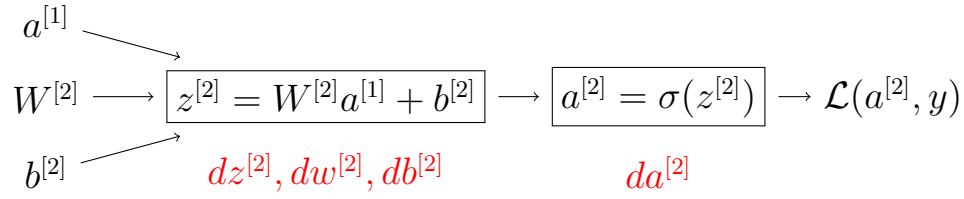
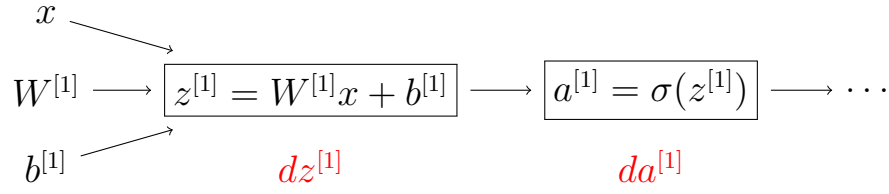
, where $a=g(z)$ and therefore $\frac{dz}{da}=g'(z)$ In case of logistic regression: $g(z)=\sigma(z)$, therefore $g'(z)=g(z)(1-g(z))$

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

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

Neural network gradients:

Compute forward:



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

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

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

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

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

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

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

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

Summary of gradient descent

For a simple training example: (from the previous 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:

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

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

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

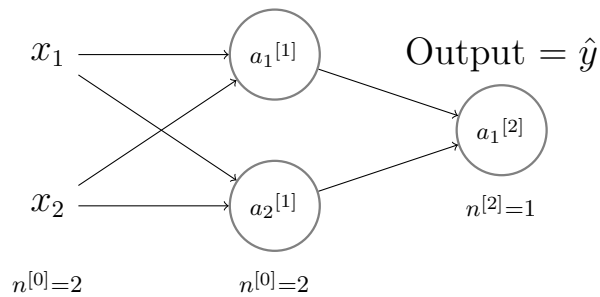
$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]}), \text{ elementwise in } (n^{[1]}, m) \text{ dimension.}$$

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

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

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.)