DeepLearning.ai

https://www.coursera.org/specializations/deep-learning

course 1

Bassalayev Maxim

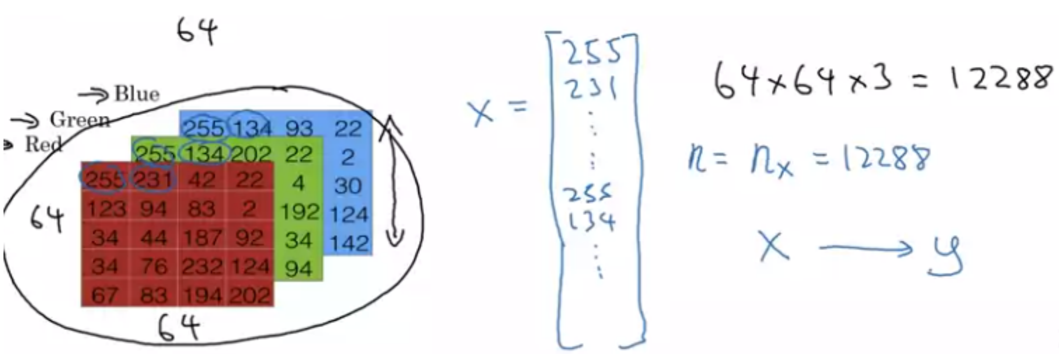
Notations

- $m = m_{train}-$train examples
- $m_{test}-$ test examples
- $n^{[l]}-$number of hidden units on layer l
- $g^{[l]}-$ activation function on layer l
- $w^{[l]}, b^{[l]}$ - parameters on layer l
- $dW^{[l]}, dB^{[l]} \dots dA^{[l]}$ - derivatives on layer l
- $par^{[l](i)}-$ the value of example i parameter on layer l
- $\alpha-$ learning rate

Binary classification, Logistic Regression

Input : picture(x) $\rightarrow$ Output : 1(cat) or 0(non cat)

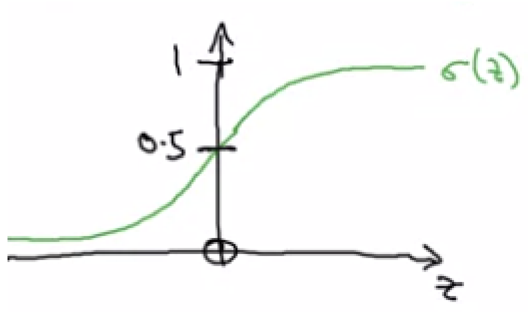
Interpret the picture like a RGB matrix.



RGB matrix

Let, $n_x = 12288$.
m training examples : $\{(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})\}, \forall i : x^{(i)} \in \mathbb{R}^{n_x}, y^{(i)} \in \{0,1\}$.
Let, $X = \left(x^{(1)}, ..x^{(m)}\right) \in Mat_{n_x \times m}, Y = \left(y^{(1)}, ..y^{(m)}\right) \in Mat_{1 \times m}$.

Given, x, want $\hat{y} = p(y=1|x), \hat{y} \in [0;1]$.
Parameters : $w \in \mathbb{R}^{n_x}, b \in \mathbb{R}$
Output $\hat{y} = \sigma(\omega^T x + b)$:



Sigma function

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

$$z \to \infty \Rightarrow, \sigma(z) \to 1$$

$$z \to 0 \Rightarrow, \sigma(z) \to 0$$

Logistic regression, cost and error function

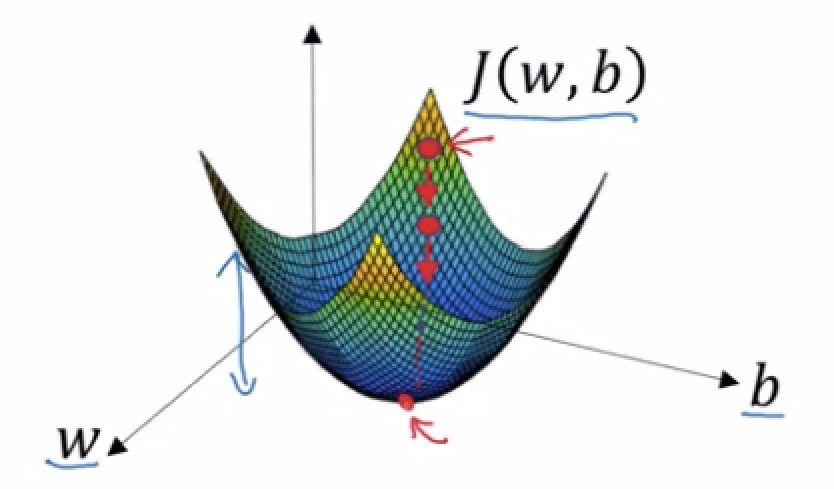
Let, for i-th example $\hat{y}^{(i)} = \sigma(\omega^T x^{(i)} + b), z^{(i)} = \omega^T x^{(i)} + b$

$$\begin{cases} x^{(i)} \\ y^{(i)} \\ z^{(i)} \end{cases} - \text{for i} - \text{th example}$$

Loss(error) function : $L(\hat{y}, y) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$ - use for a single test.
Cost function : $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$ - use for train/test dataset(shows how good parameters w and b).

Gradient Descent

Want to find, w, b, than minimize $J(w,b)$. We will use gradient descent.



Gradient descent

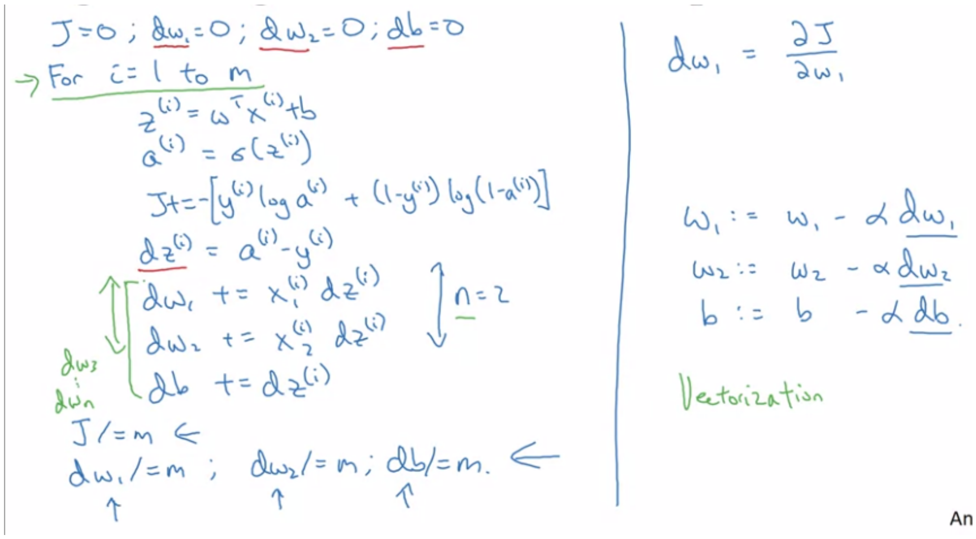
Algorithm
Repeat :
$w = w - \alpha \frac{\partial J(w,b)}{\partial w}$
$b = b - \alpha \frac{\partial J(w,b)}{\partial b}$
$\alpha-$ learning rate.

Logistic regression on m examples

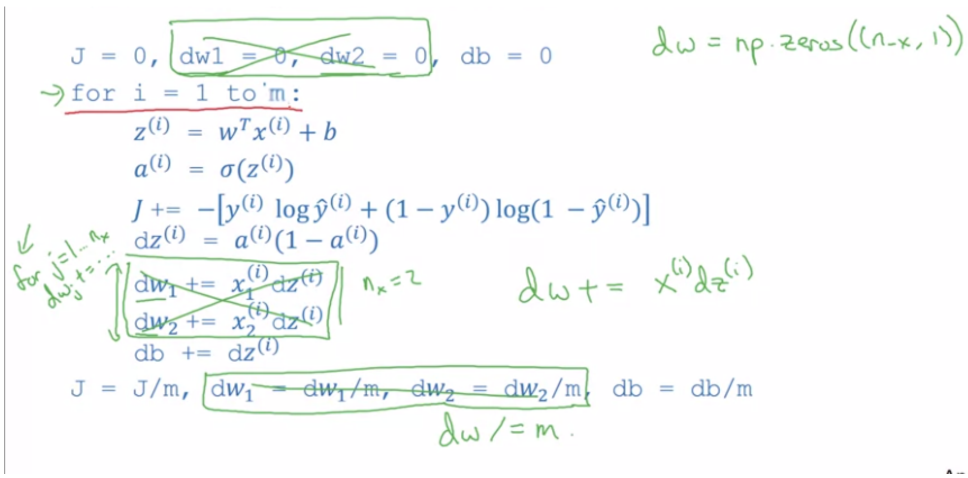
$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(a^{(i)}, y^{(i)}),\ a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(\omega^T x^{(i)} + b)$

$$\frac{\partial}{\partial w_1} J(w,b) = \underbrace{\frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial w_1} L(a^{(i)}, y^{(i)})}_{dw_1^{(i)} - (x^{(i)}, y^{(i)})}$$

Algorithm



Pseudocode



Pseudocode with vectorization

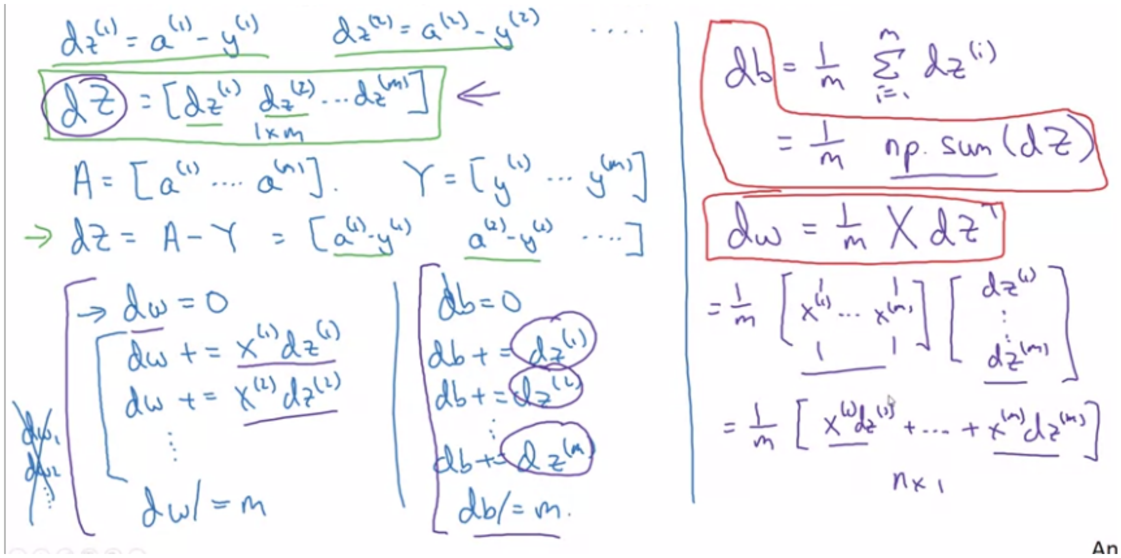
Calculate without for-loop

$X = \left(x^{(1)}, ..x^{(m)}\right) \in Mat_{n_x \times m}$

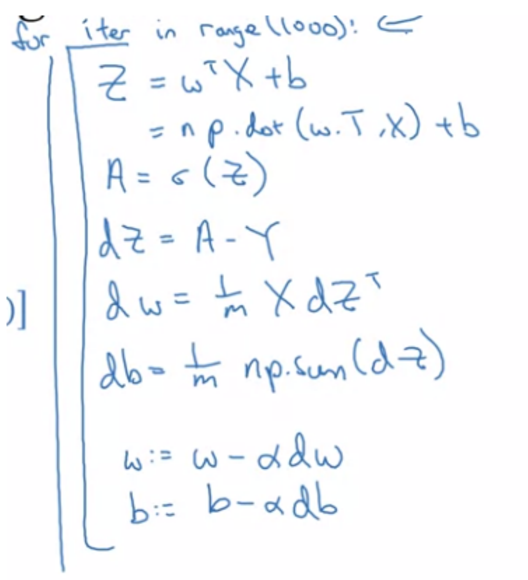
$a^{(i)} = \sigma(z^{(i)}) = \sigma(\omega^T x^{(i)} + b)$

$Z = \left(z^{(1)}, ..z^{(m)}\right) = \omega^T X + \left(\underbrace{b \dots b}_{m}\right) = \left(\omega^T x^{(1)} + b \dots \omega^T x^{(m)} + b\right)$

$A = \sigma(Z)$



Calculate dz and db and dw without for-loop



Upgraded logistic regression algorithm

Notes

Common steps for pre-processing a new dataset are:

- Figure out the dimensions and shapes of the problem (Mtrain, Mtest, numpx, ...)
- Reshape the datasets such that each example is now a vector of size (numpx $*$ numpx $*$ 3, 1)
- "Standardize"the data

2 layer NN

$var_j^{[i]}$ means i-th layer and j-th node in the i-th layer. (picture 2)

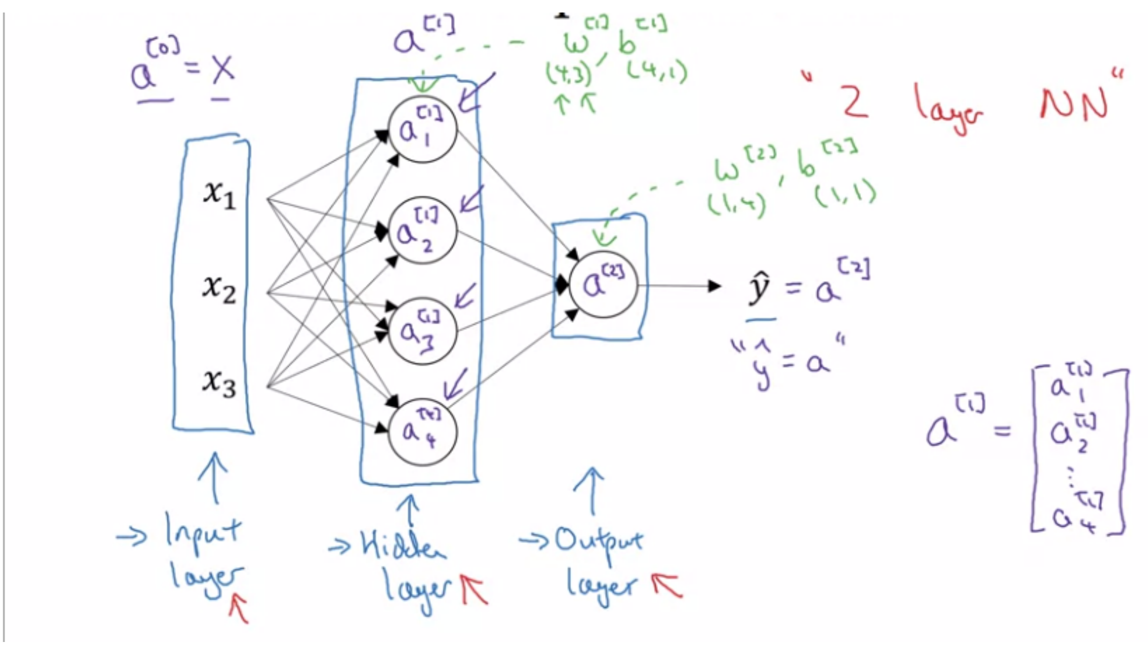
$$\underbrace{x}_{(3\times1)} = \underbrace{a^{[0]}}_{(3\times1)}$$

$$\underbrace{z^{[1]}}_{(4\times1)} = \underbrace{W^{[1]}x}_{(4\times3)\times(3\times1)} + \underbrace{b^{[1]}}_{(4\times1)}$$

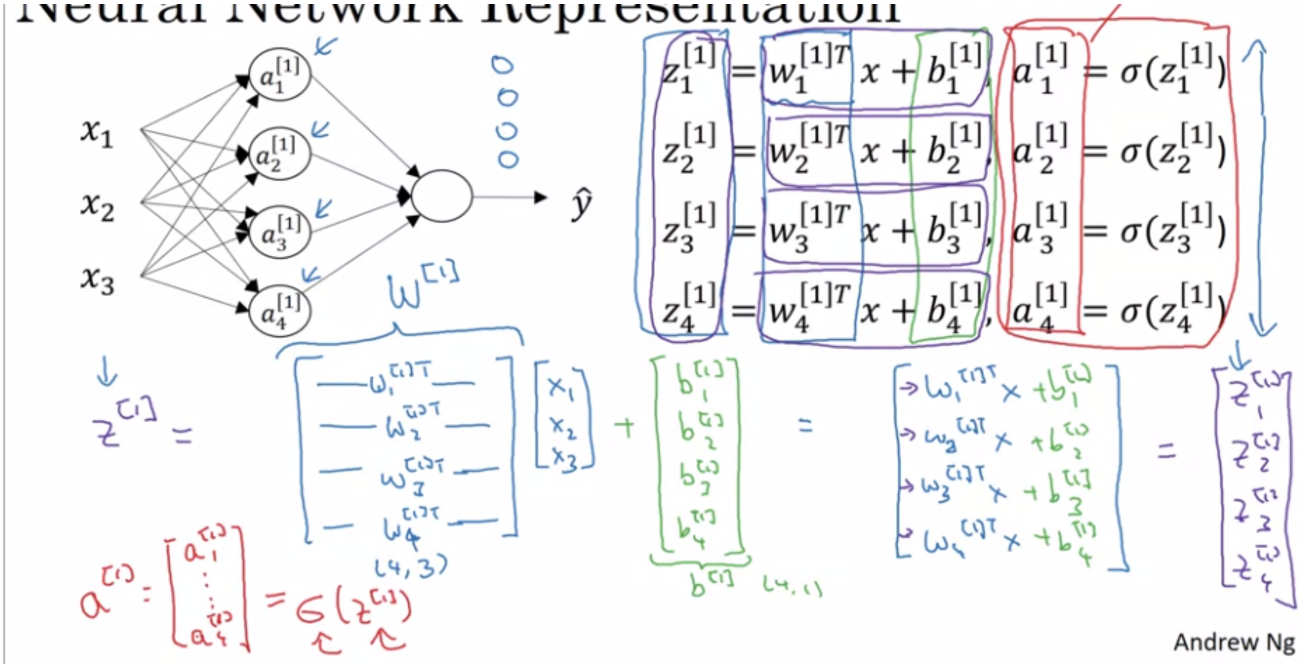
$$\underbrace{a^{[1]}}_{(4\times1)} = \underbrace{\sigma(z^{[1]})}_{(4\times1)}$$

$$\underbrace{z^{[2]}}_{(1\times1)} = \underbrace{W^{[2]}a^{[1]}}_{(1\times4)\times(4\times1)} + \underbrace{b^{[2]}}_{(1\times1)}$$

$$\underbrace{a^{[2]}}_{(1\times1)} = \underbrace{\sigma(z^{[2]})}_{(1\times1)}$$

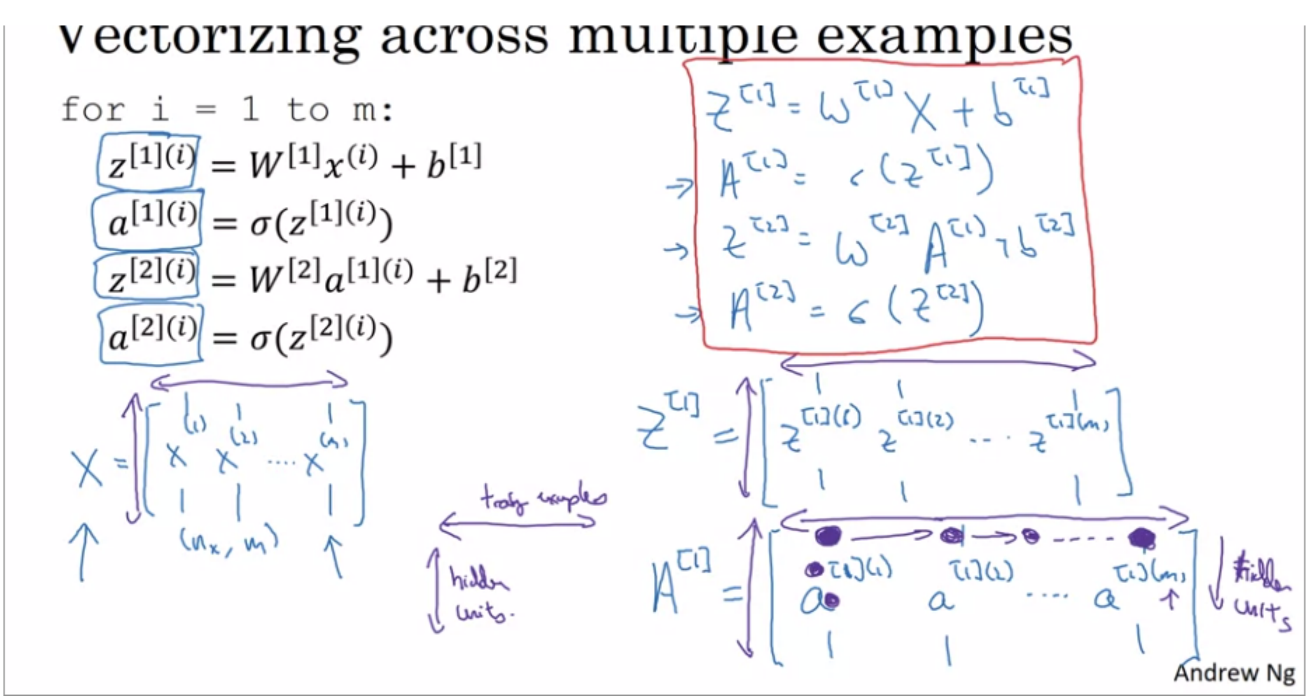


2 layer NN (1)

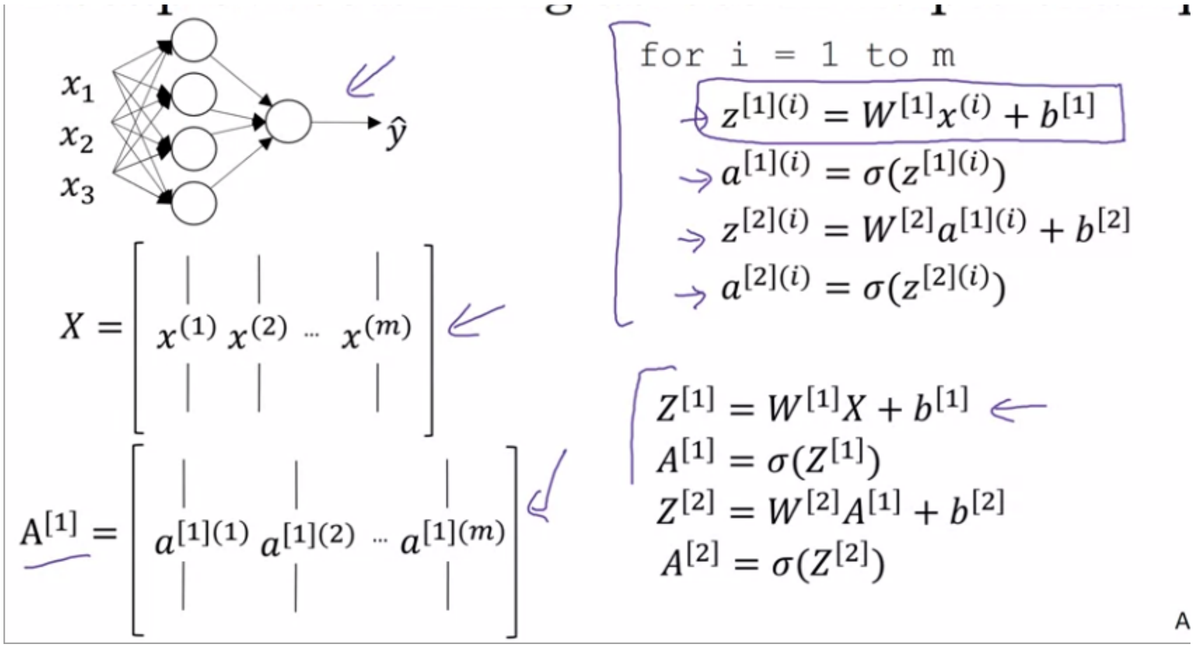


2 layer NN (2)

Vectorizing across multiple examples



Vectorizing across multiple examples

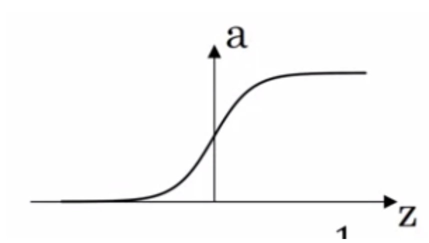


Vectorizing across multiple examples

Activation functions

Sigmoid :

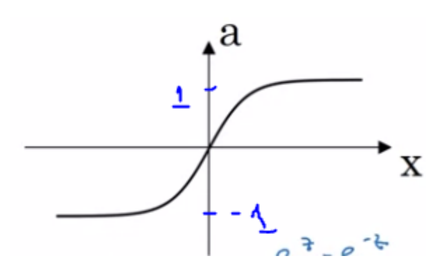
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
$$\sigma(x)' = \sigma(x) \cdot (1 - \sigma(x))$$



Sigmoid

Hyperbolic tangent :

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$\tanh(x)' = 1 - (\tanh(x))^2$$

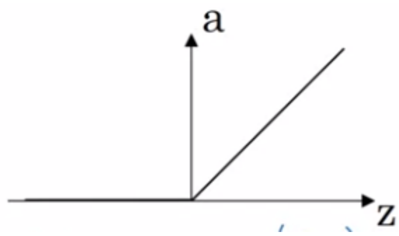


Hyperbolic tangent

Relu :

$$relu(x) = max(0, x)$$

$$relu(x)' = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \\ \text{undefined}, & \text{x} = 0 \end{cases}$$

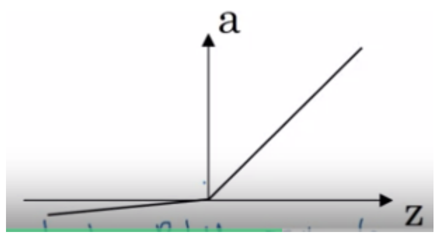


Relu

Leaky Relu :

$$lrelu(x) = max(0.01 \cdot x, x)$$

$$lrelu(x)' = \begin{cases} 0.01, & x < 0 \\ 1, & x \geqslant 0 \end{cases}$$



Leaky Relu

Forward and back - propagation for 2-layer NN

Prarameters : $\underbrace{w^{[1]}}_{(n^{[1]} \times n^{[0]})}, \underbrace{b^{[1]}}_{(n^{[1]} \times 1)}, \underbrace{w^{[2]}}_{(n^{[2]} \times n^{[1]})}, \underbrace{b^{[2]}}_{(n^{[2]} \times 1)}$

Cost function : $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \dfrac{1}{m} \sum_{i=1}^{m} L(\widehat{y}, y), \widehat{y} = a^{[2]}$

Forward propagation :

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$

Back propagation :

$$Y = (y^{(1)} \dots y^{(m)}) \text{ (stacked in column)}$$
$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$dB^{[2]} = \frac{1}{m} \cdot \text{np.sum}(dZ^{[2]}, \text{ axis=1, keepdims=True})$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$
$$dB^{[1]} = \frac{1}{m} \cdot \text{np.sum}(dZ^{[1]}, \text{ axis=1, keepdims=True})$$

Random Initialization

```
W1 = np.random.randn((n[1], n[0])) * 0.01
b1 = np.zeros((n[1], 1))
W2 = np.random.randn((n[2], n[1])) * 0.01
b2 = np.zeros((n[2], 1))
```

Deep L - layer neural network

Parameters : $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]} \dots w^{[L]}, b^{[L]}$
Dimensions : $\underbrace{w^{[l]}}_{(n^{[l]} \times n^{[l-1]})}, \underbrace{dW^{[l]}}_{(n^{[l]} \times n^{[l-1]})}, \underbrace{b^{[l]}}_{(n^{[l]} \times 1)}, \underbrace{db^{[l]}}_{(n^{[l]} \times 1)}, \underbrace{Z^{[l]}}_{(n^{[l]} \times m}, \underbrace{dZ^{[l]}}_{(n^{[l]} \times m)}, \underbrace{A^{[l]}}_{(n^{[l]} \times m)}, \underbrace{dA^{[l]}}_{(n^{[l]} \times m)}, \underbrace{X}_{(n^{[0]} \times m)}$

Forward propagation :

$$\underbrace{Z^{[1]}}_{(n^{[1]} \times m)} = \underbrace{W^{[1]}}_{(n^{[1]} \times n^{[0]})} \underbrace{X}_{(n^{[0]} \times m)} + \underbrace{b^{[1]}}_{(n^{[1]} \times m)} = (z^{[1](1)} \dots z^{[1](m)}) \text{ (stacked in column)}$$
$$A^{[1]} = g^{[1]}(Z^{[1]}) = (a^{[1](1)} \dots a^{[1](m)}) \text{ (stacked in column)}$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\dots\dots$$
$$Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]}$$
$$A^{[L]} = g^{[L]}(Z^{[L]})$$

Back propagation :

$$\text{input} : dA^{[l]} \rightarrow output : dA^{[l-1]}, dW^{[l]}, db^{[l]}$$
$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} \text{np.sum}(dZ^{[l]}, \text{ axis=1, keepdims=True})$$
$$dA^{[l-1]} = W^{[l]T} dZ^{[l]}$$

Hyperparameters

- Learning rate α
- iterations
- hidden layers L
- hidden units $n^{[1]}, n^{[2]} \dots n^{[L]}$
- choice of activation functions