DEEP LEARNING SPECIALIZATION
COURSE 1/5
NEURAL NETWORKS AND DEEP LEARNING

→ WHAT IS A NEURAL NETWORK?

* NEURON

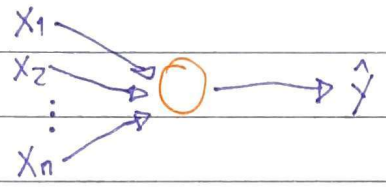
- SIMPLE ELEMENT THAT TAKES INPUTS AND CONVERT THEM TO AN OUTPUT

* NEURAL NETWORKS

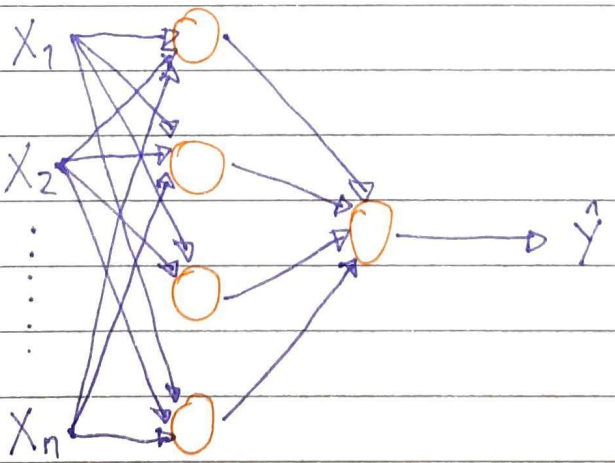
- A LOT OF NEURONS INTERCONNECTED, GENERALLY ORGANIZED IN LAYERS

* EXAMPLE:

- LINEAR REGRESSION: A SINGLE NEURON



- NEURAL NETWORK:



→ EXAMPLES OF NEURAL NETWORKS AND SOME APPLICATIONS

* COMMON DATASETS: STANDARD N.N.
* IMAGE: CONVOLUTIONAL N.N.
* AUDIO AND SEQUENCED DATA: RECURRENT N.N.
* THERE MAY BE CUSTOM AND HYBRID ARCHITECTURES

→ STRUCTURED VS. UNSTRUCTURED DATA

* STRUCTURED DATA

- DATABASES, DATASETS, DATAFRAMES
- LABELED FEATURES

* UNSTRUCTURED DATA

- RAW DATA, LIKE AUDIO, IMAGE AND TEXT

→ LOGISTIC REGRESSION (BINARY CLASSIFICATION)

* NOTATION

- $X = \begin{bmatrix} \vdots & \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(m)} \\ \vdots & \vdots & \vdots & & \vdots \end{bmatrix}$

$$Y = [y^{(1)}\ y^{(2)}\ y^{(3)} \ldots y^{(m)}]$$

- $X.shape = (n, m)$
- $Y.shape = (1, m)$
- EACH FEATURE VECTOR OF EACH TRAINING EXAMPLE IS STORED AS A COLUMN OF X

* IDEA

- GIVEN X, WANT $\hat{y} = P(Y=1|X)$
- $X \in \mathbb{R}^{n_x}$
- PARAMETERS: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$
- SIGMOID FUNCTION
  - $g(z) = \frac{1}{1+e^{-z}}$
- GOAL: FIND BETTER w AND b

* LOSS FUNCTION

- $L(\hat{y}, y) = -[y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})]$

* COST FUNCTION

- $J(w, b) = \frac{1}{m} \cdot \sum_{i=1}^{m} L(\hat{y}^{(i)}, y)$

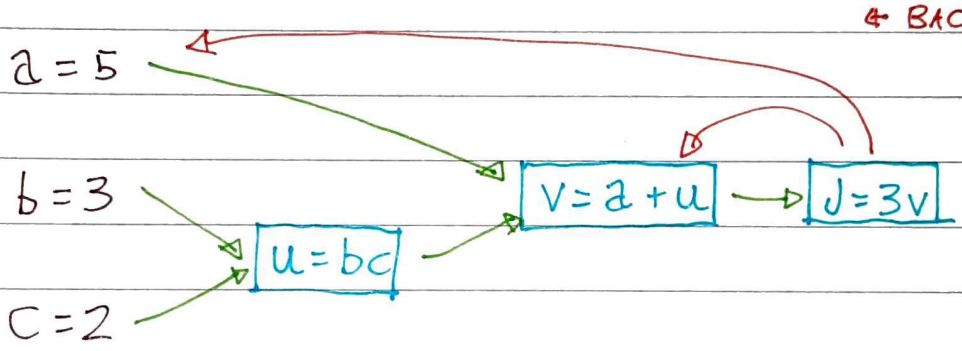
* GRADIENT DESCENT (SIMULTANEOUS UPDATE)

$$w = w - \alpha \cdot \frac{\partial J(w,b)}{\partial w} \qquad b = b - \alpha \cdot \frac{\partial J(w,b)}{\partial b}$$

* DERIVATIVES WITH COMPUTATION GRAPH (EXAMPLE)

- $J(a,b,c) = 3(a+bc)$

→ FORWARD PROPAGATION
← BACK PROPAGATION



$J = 3v$

$v = 11 \xrightarrow{\text{INCREASE}} v = 11.001$
$J = 33 \xrightarrow{\text{INCREASE}} J = 33.003$

$$\frac{\partial J}{\partial v} = \frac{.003}{.001} = 3$$

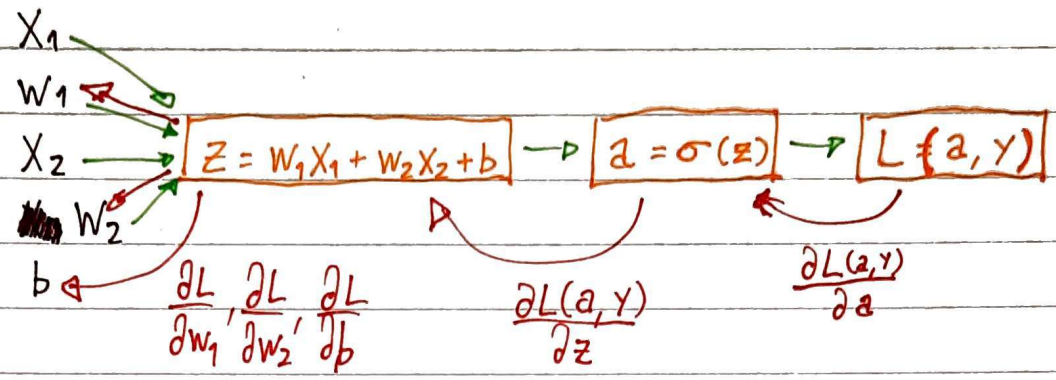
$a = 5 \xrightarrow{\text{INCREASE}} a = 5.001$
$v = 11 \xrightarrow{\text{INCREASE}} v = 11.001$
$J = 33 \xrightarrow{\text{INCREASE}} J = 33.003$

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial v} \cdot \frac{\partial v}{\partial a} = 3 \cdot \frac{.001}{.001} = 3$$

- BACK PROPAGATION USED TO CALCULATE DERIVATIVES OF FINAL OUTPUT VARIABLE

→ LOGISTIC REGRESSION GRADIENT DESCENT (DERIVATIVES WITH BACK PROPAGATION)

* EXAMPLE



- $\frac{\partial L(a,y)}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}$

- $\frac{\partial L(a,y)}{\partial z} = a - y$

- $\frac{\partial L(a,y)}{\partial w_1} = x_1 \cdot \frac{\partial L(a,y)}{\partial z} = x_1 \cdot (a-y)$

- $\frac{\partial L(a,y)}{\partial w_2} = x_2 \cdot \frac{\partial L(a,y)}{\partial z} = x_2 \cdot (a-y)$

- $\frac{\partial L(a,y)}{\partial b} = \frac{\partial L(a,y)}{\partial z} = a - y$

* ON m EXAMPLES: CALCULATE AVERAGE OF SUM OF EXAMPLES

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

→ VECTORIZATION

* PARALLELISM
  - POSSIBILITY TO RUN SIMD (SINGLE INSTRUCTION, MULTIPLE DATA
  - MAKES CODE RUNS FASTER
* AVOID EXPLICIT FOR-LOOPS WHENEVER POSSIBLE
* NUMPY HAVE MANY FUNCTIONS THAT USE PARALLELISM

→ VECTORIZE LOGISTIC REGRESSION

* CONSIDERING NOTATION:
  - $z^{(i)} = w^T x^{(i)} + b$
  - $a^{(i)} = \sigma(z^{(i)})$
  - $X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ X_1 & X_2 & \cdots & X_m \\ \vdots & \vdots & & \vdots \end{bmatrix}$ $w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$ $X \in \mathbb{R}^{n_x \cdot m}$ $w \in \mathbb{R}^m$ $b \in \mathbb{R}$

* CALCULATE VECTORIZED LOGISTIC REGRESSION
  - $z = np.dot(np.transpose(w), X) + b$
  - $\hat{y} = sigmoid(z)$

* CALCULATE VECTORIZED GRADIENT DESCENT
  - $db = \frac{1}{m} \cdot np.sum(dz)$
  - $dz = \hat{y} - y$
  - $dw = \frac{1}{m} \cdot np.dot(X\_matrix, np.transpose(dz))$

→ BROADCASTING IN PYTHON

* VECTORS AND MATRIXES RESHAPING FOR SOME OPERATIONS

(m, n) MATRIX $\quad \overset{+}{-}\ \underset{\div}{\times} \quad (1, n) \rightsquigarrow (m, n)$

$\qquad\qquad\qquad\quad (m, 1) \rightsquigarrow (m, n)$

$(m, 1) \quad \overset{+}{-}\ \underset{\div}{\times} \quad \mathbb{R} \rightsquigarrow (m, 1)$

* EXAMPLES

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + 100 \rightsquigarrow \begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \begin{bmatrix}100\\100\\100\\100\end{bmatrix} = \begin{bmatrix}101\\102\\103\\104\end{bmatrix}$$

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix} = \begin{bmatrix}101 & 202 & 303\\104 & 205 & 306\end{bmatrix}$$

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 100 & 100\\200 & 200 & 200\end{bmatrix} = \begin{bmatrix}101 & 102 & 103\\204 & 205 & 206\end{bmatrix}$$

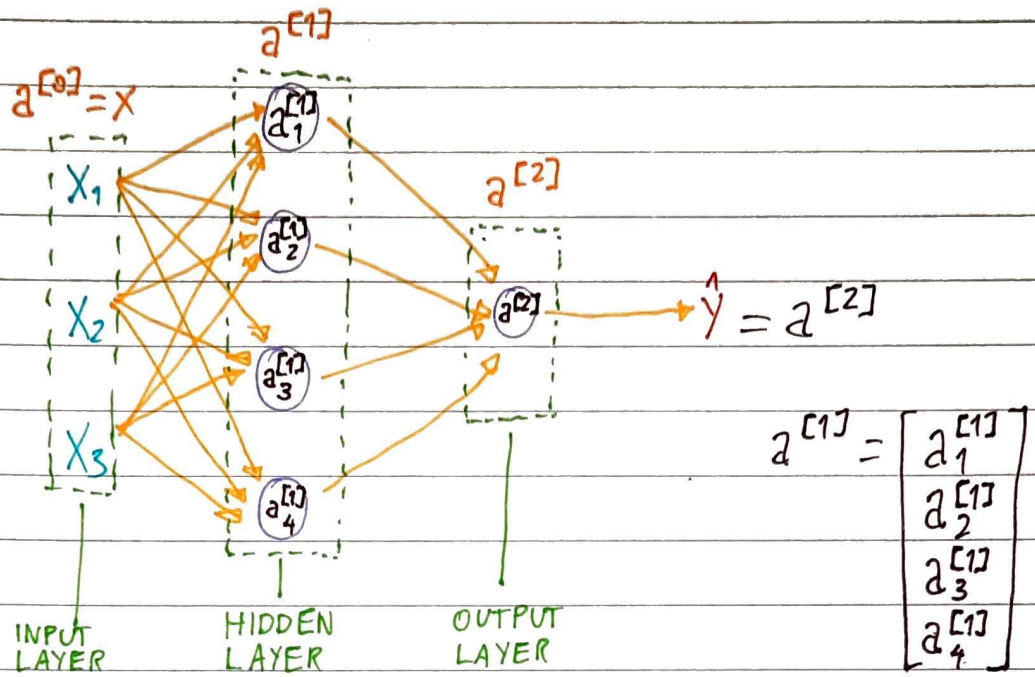
→ TIPS AND TRICKS FOR AVOIDING BUGS WITH NUMPY

* DON'T USE RANK 1 ARRAYS (1D-ARRAY)

- SHAPE OF A RANK 1 ARRAY: (X_1)
- SHAPE OF A RANK 2 ARRAY: (X_1, X_2)

NEURAL NETWORK REPRESENTATION AND NOTATION

* TWO-LAYER NEURAL NETWORK



$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

* COMPUTATION FOR TWO-LAYER N.N. (HIDDEN LAYER)

- $$z^{[1]} = \begin{bmatrix} \cdots\cdots w_1^{[1]T} \cdots\cdots \\ \cdots\cdots w_2^{[1]T} \cdots\cdots \\ \cdots\cdots w_3^{[1]T} \cdots\cdots \\ \cdots\cdots w_4^{[1]T} \cdots\cdots \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

- $$z^{[1]} = \begin{bmatrix} w_1^{[1]T} X + b_1^{[1]} \\ w_2^{[1]T} X + b_2^{[1]} \\ w_3^{[1]T} X + b_3^{[1]} \\ w_4^{[1]T} X + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

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

* COMPUTATION FOR TWO-LAYER N.N. (OUTPUT LAYER)
  - $z^{[2]} = W^{[2]T} \cdot a^{[1]} + b^{[2]}$
  - $a^{[2]} = \sigma(z^{[2]})$

→ VECTORIZING TWO-LAYER N.N. ACROSS MULTIPLE EXAMPLES

* EXAMPLES
  - $x^{(1)} \rightarrow a^{[2](1)} = \hat{y}^{(1)}$
    $x^{(2)} \rightarrow a^{[2](2)} = \hat{y}^{(2)}$
    $\vdots$
    $x^{(m)} \rightarrow a^{[2](m)} = \hat{y}^{(m)}$

$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$

  - $z^{[1]} = W^{[1]} X + b^{[1]}$
    $A^{[1]} = \sigma(z^{[1]})$
    $z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$
    $A^{[2]} = \sigma(z^{[2]})$

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

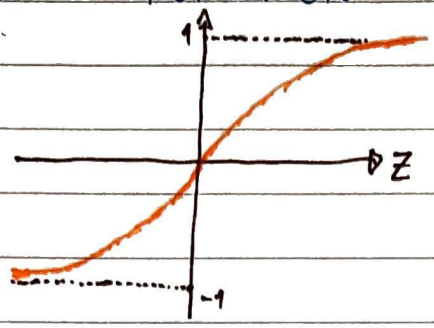
  - $A^{[1]} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$ } HIDDEN UNITS

TRAINING EXAMPLES

* VECTORIZED IMPLEMENTATIONS ALLOWS US TO CALCULATE THE ENTIRE MATRIX OPERATIONS FASTER USING PARALLELISM INSTEAD OF USING FOR-LOOPS FOR ITERATE AT EACH ELEMENT OF MATRIX

→ ACTIVATION FUNCTIONS

* TANH FUNCTION



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

- "SHIFTED" VERSION OF SIGMOID FUNCTION WHICH VARIES FROM -1 TO 1
- MEAN OF THIS ACTIVATION IS ZERO
- "CENTERING DATA"
- LEARNING FOR THE NEXT LAYERS IS ~~EASY~~ EASIER

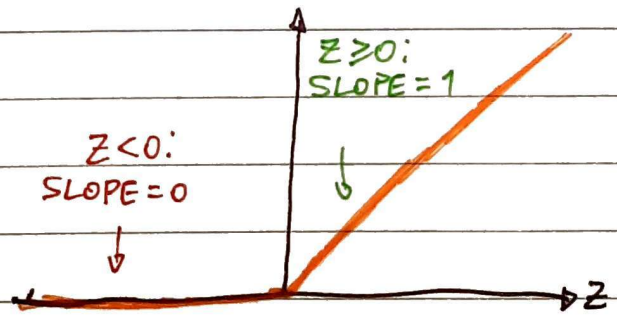
* EXCEPTION FOR SIGMOID FUNCTION

- OUTPUT LAYER: $Y = \{0, 1\}$ FOR BINARY CLASSIFICATION

* CONS OF TANH() AND SIGMOID()

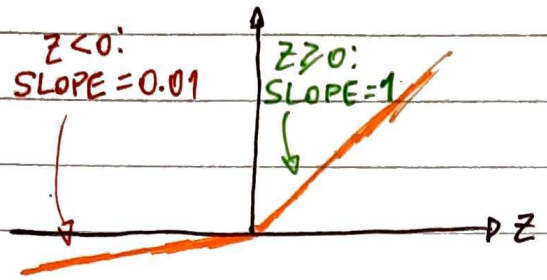
- AT VERY LARGE OR VERY SMALL Z VALUES, THE SLOPE OF TANH AND SIGMOID ARE ALMOST ZERO: IT MAKES GRADIENT DESCENT SLOWER

* RECTIFIED LINEAR UNIT (ReLU)



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

* LEAKY ReLU



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

- ADVANTAGE FOR ReLU AND LEAKY ReLU: IN A LOT OF SPACE OF z, THE SLOPE WILL BE DIFFERENT FROM ZERO: THE NEURAL NETWORK WILL LEARN FASTER

→ WHY USE NON-LINEAR ACTIVATION FUNCTIONS?

* IF WE USE ONLY LINEAR ACTIVATION FUNCTIONS, THE OUTPUT RESULT WILL BE A LINEAR COMBINATION, AND THAT'S NO BETTER THAN USE A SINGLE NEURON
* LINEAR FUNCTIONS ARE INTERESTING FOR REGRESSION IF USED AT THE OUTPUT LAYER

→ DERIVATIVES OF ACTIVATION FUNCTIONS

* SIGMOID FUNCTION
  - $\frac{\partial \sigma}{\partial z} = \sigma(z) \cdot (1 - \sigma(z))$
* TANH FUNCTION
  - $\frac{\partial \tanh(z)}{\partial z} = 1 - (\tanh(z))^2$

* ReLU AND LEAKY ReLU

- $\frac{\partial}{\partial z} \max(0, z) = \begin{cases} 0, & \text{IF } z < 0 \\ 1, & \text{IF } z \geq 0 \end{cases}$ (ReLU)

- $\frac{\partial}{\partial z} \max(0.01z, z) = \begin{cases} 0.01, & \text{IF } z < 0 \\ 1, & \text{IF } z \geq 0 \end{cases}$ (LEAKY ReLU)

→ GRADIENT DESCENT FOR NEURAL NETWORKS

* PARAMETERS (TWO LAYER N.N.)

$w^{[1]}$, $b^{[1]}$, $w^{[2]}$, $b^{[2]}$ — $n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$

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

* COST FUNCTION

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

* INITIALIZATION OF PARAMETERS

- INITIALIZE ALL WITH RANDOM NUMBERS, NOT ZEROS

* FORWARD PROPAGATION

REPEAT: {

→ COMPUTE PREDICTIONS ($\hat{y}^{(i)}$, $i = 1, 2, \ldots, m$)

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

→ $dw^{[2]} = \frac{\partial J}{\partial w^{[2]}}$, $db^{[2]} = \frac{\partial J}{\partial b^{[2]}}$

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

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

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

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

}

* BACKPROPAGATION

$$dz^{[2]} = A^{[2]} - Y$$
$$dw^{[2]} = \frac{1}{m} \cdot dz^{[2]} \cdot A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} \cdot np.sum(dz^{[2]}, axis=1, keepdims=True)$$
$$dz^{[1]} = (w^{[2]} dz^{[2]}) * g^{[1]'}(z^{[1]})$$

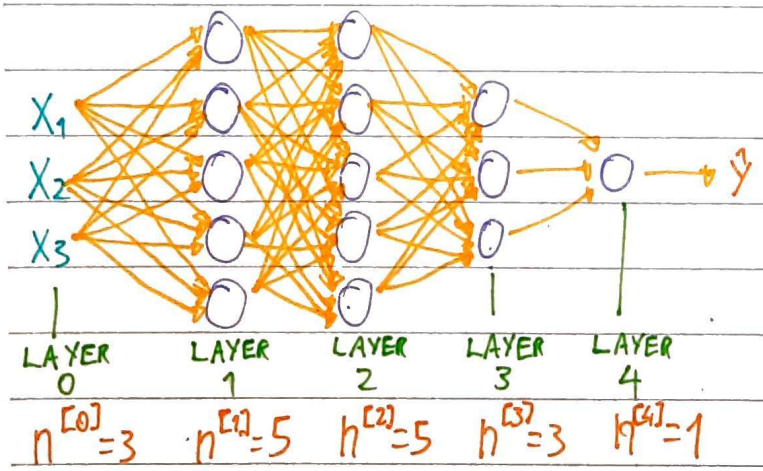
↑ ELEMENT-WISE: np.multiply()

$$dw^{[1]} = \frac{1}{m} dz^{[1]} \cdot X^T$$
$$db^{[1]} = \frac{1}{m} \cdot np.sum(dz^{[1]}, axis=1, keepdims=True)$$

→ WHAT HAPPENS IF YOU INITIALIZE YOUR WEIGHTS TO ZERO?

* EVERY SINGLE ELEMENT OF THE LAYER WILL COMPUTE THE EXACT SAME FUNCTION
* INITIALIZE $w^{[i]}$ WITH np.random.randn() * 0.01 AND $b^{[i]}$ WITH np.zeros()

→ DEEP NEURAL NETWORK NOTATION (EXAMPLE: 4-LAYER N.N.)



$L = 4$ (LAYERS)

$n^{[l]}$ = NO. OF UNITS IN LAYER l

$n^{[0]} = n_x = 3 \{ n^{[1]} = 5$

$n^{[2]} = 5 \{ n^{[3]} = 3 \{ n^{[4]} = n^{[L]} = 1$

$a^{[l]}$ = ACTIVATIONS IN LAYER l

$a^{[l]} = g^{[l]}(z^{[l]})$

$w^{[l]}$ = WEIGHTS FOR LAYER l

$b^{[l]}$ = BIASES FOR LAYER l

$a^{[0]} = X$

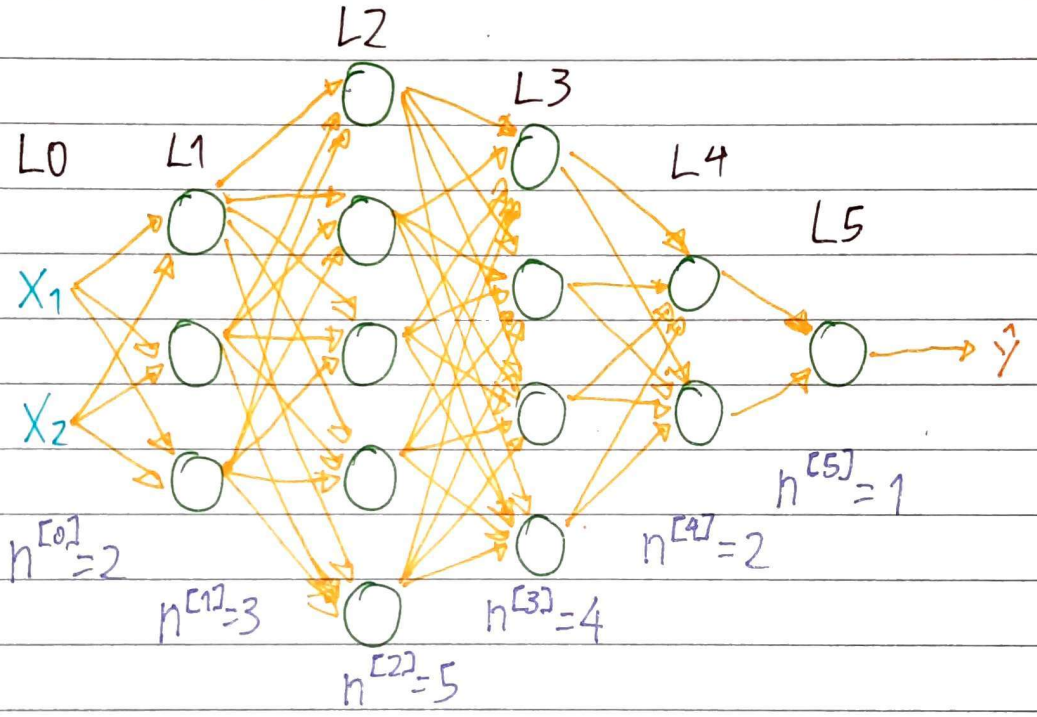
$a^{[L]} = \hat{y}$

→ FORWARD PROPAGATION IN A DEEP NETWORK

* $z^{[l]} = w^{[l]} \cdot a^{[l-1]} + b^{[l]}$
* $a^{[l]} = g^{[l]}(z^{[l]})$
* CAN IMPLEMENT THIS ON A VECTORIZED WAY

→ GETTING YOUR MATRIX DIMENSIONS RIGHT

* EXAMPLE



EXAMPLE

$z^{[1]}.shape = (n^{[1]}, 1) = (3, 1)$

$X.shape = (n^{[0]}, 1) = (2, 1)$

$w^{[1]}.shape = (n^{[1]}, n^{[0]}) = (3, 2)$

$b^{[1]}.shape = (n^{[1]}, 1) = (3, 1)$

VECTORIZED EXAMPLE

$Z^{[1]}.shape = (n^{[1]}, m)$

$X.shape = (n^{[0]}, m)$

GENERAL

$w^{[l]}.shape = (n^{[l]}, n^{[l-1]})$

$b^{[l]}.shape = (n^{[l]}, 1)$

$dw^{[l]}.shape = w^{[l]}.shape$

$db^{[l]}.shape = b^{[l]}.shape$

$dz^{[l]}.shape = (n^{[l]}, m)$

$dA^{[l]}.shape = dz^{[l]}.shape$

→ WHY DEEP REPRESENTATIONS?

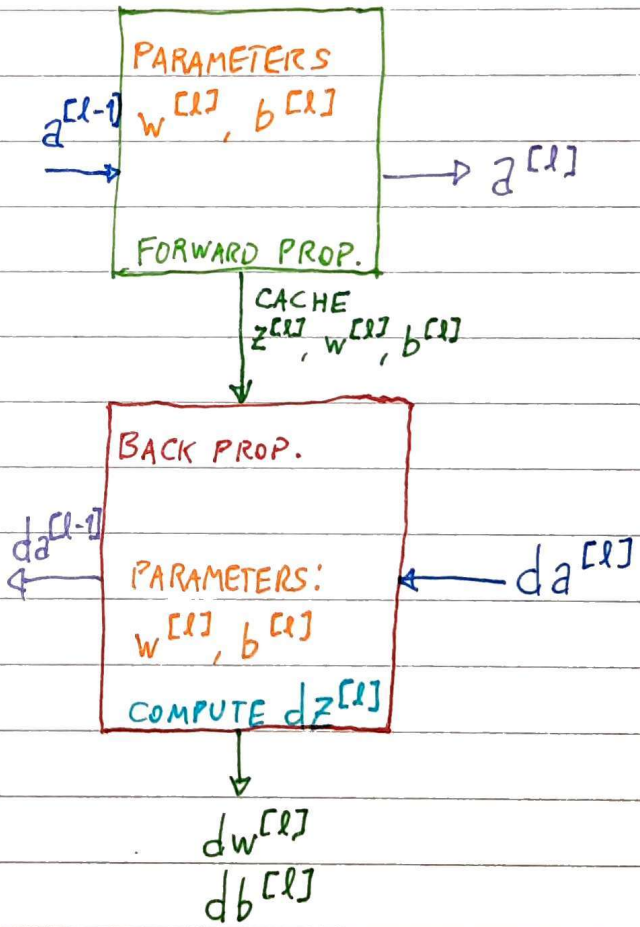
* LAYERS ARE CONNECTED IN A WAY THAT THE FIRST LAYERS ARE RESPONSIBLE FOR DETECTING SIMPLE FEATURES/ PATTERNS, AND THE NEXT LAYERS DETECTS MORE COMPLEX PATTERNS
* THERE ARE FUNCTIONS YOU CAN COMPUTE WITH A "SMALL" L-LAYER DEEP N.N. THAT SHALLOWER NETWORKS REQUIRE EXPONENTIALLY MORE HIDDEN UNITS TO COMPUTE

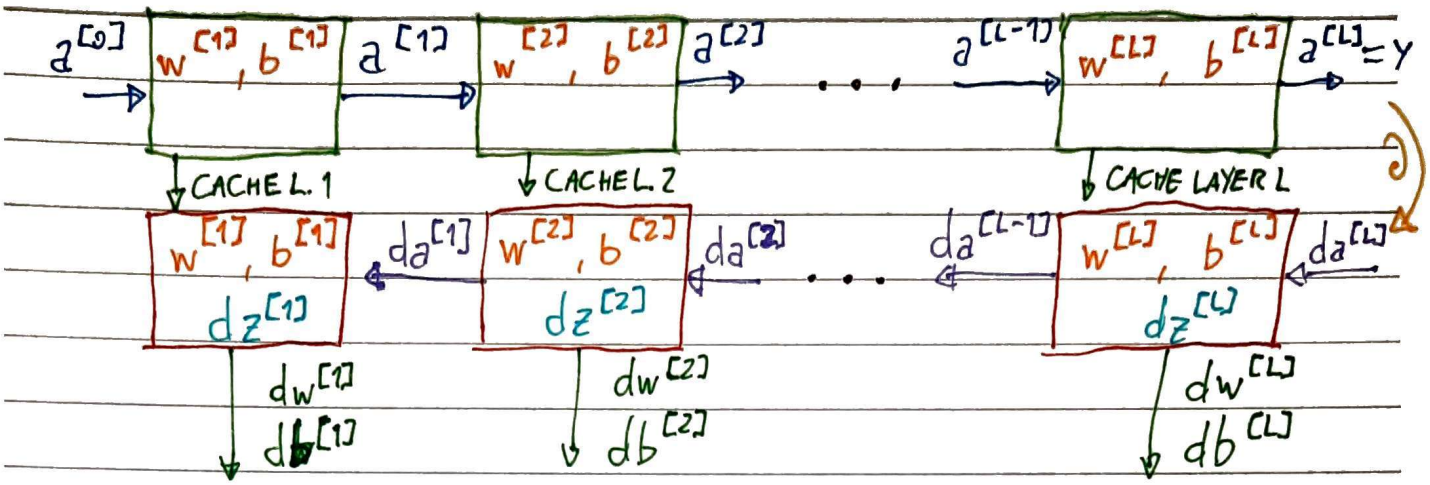
→ BUILDING BLOCKS OF DEEP NEURAL NETWORKS

LAYER l:

FORWARD PROPAGATION BLOCK RECEIVES $a^{[l-1]}$ AND USES $W^{[l]}$ AND $b^{[l]}$ TO CALCULATE $a^{[l]}$.

BACK PROPAGATION BLOCK RECEIVES $da^{[l]}$ AND THE CACHE OF $z^{[l]}$, AND USES $W^{[l]}$ AND $b^{[l]}$ TO COMPUTE $dz^{[l]}$, $da^{[l-1]}$, $dw^{[l]}$ AND $db^{[l]}$.





→ FORWARD PROPAGATION FOR LAYER l

* INPUT: $a^{[l-1]}$
* OUTPUT: $a^{[l]}$, CACHE ($z^{[l]}$, $w^{[l]}$, $b^{[l]}$)

* VECTORIZED IMPLEMENTATION

$$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

→ BACKWARD PROPAGATION FOR LAYER l

* INPUT: $da^{[l]}$
* OUTPUT: $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$

* VECTORIZED IMPLEMENTATION

ELEMENT-WISE MULTIPLICATION

$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis=1, keepdims=True)$$
$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$
$$da^{[L]} = -\frac{Y}{a} + \frac{(1-Y)}{(1-a)}$$
$$dA^{[L]} = [da_1^{[L]}, da_2^{[L]}, da_3^{[L]}, \dots, da_m^{[L]}]$$

→ HYPERPARAMETERS

* PARAMETERS

$W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}, \ldots$

* HYPERPARAMETERS
  - LEARNING RATE α
  - NUMBER OF ITERATIONS
  - NUMBER OF HIDDEN LAYERS L
  - NUMBER OF HIDDEN UNITS $n^{[1]}, n^{[2]}, n^{[3]}, \ldots, n^{[L]}$
  - CHOICE OF ACTIVATION FUNCTION
* OTHER HYPERPARAMETERS
  - MOMENTUM
  - MINIBATCH SIZE
  - REGULARIZATION, ...
* APPLIED DEEP LEARNING IS A VERY EMPIRICAL PROCESS