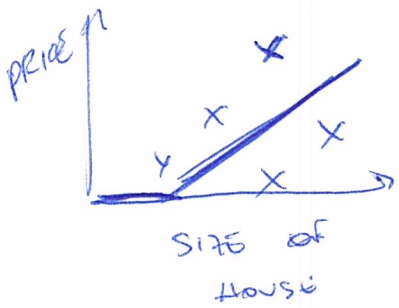
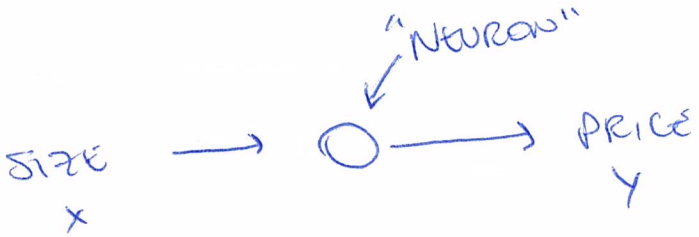
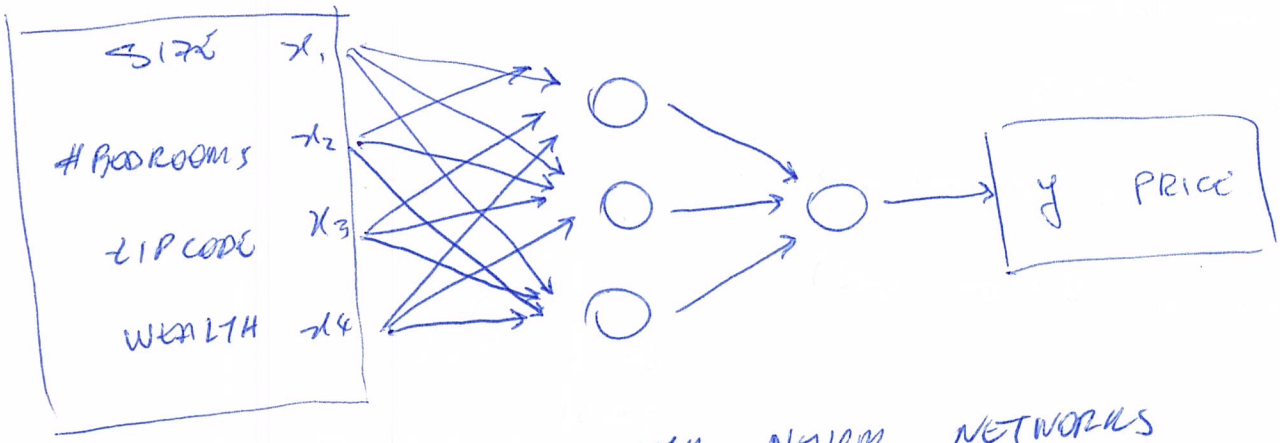
DEEP LEARNING . AI

COURSE 1 - NEURAL NETWORKS AND DEEP LEARNING

- WHAT IS A NEURAL NETWORK?



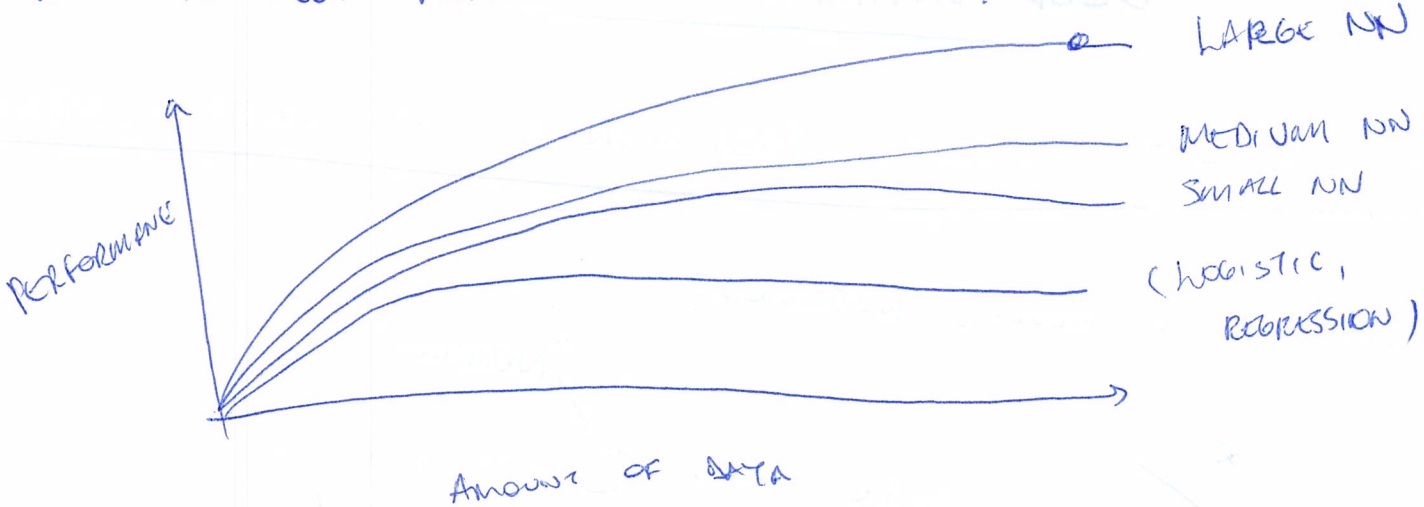




- SUPERVISED LEARNING WITH NEURAL NETWORKS

INPUT (x)	OUTPUT (y)	APPLICATION	
HOME FEATURES AD, USER INFO	PRICE	REAL ESTATE	} NN
	CLICK ON AD	ONLINE ADVERTISING	
IMAGE	OBJECT (1, ..., 1000)	PHOTO TAGGING	} CNN
AUDIO	TEXT TRANSCRIPT	SPEECH RECOGNITION	} RNN
ENGLISH	CHINESE	MACHINE TRANSLATION	
IMAGE, RADAR INFO	POSITION OF OTHER CARS	AUTONOMOUS DRIVING	} CUSTOM / HYBRID NN

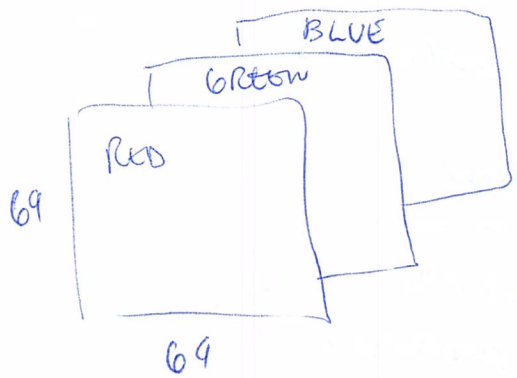
- WHY IS DEEP LEARNING TAKING OFF?



* SCALE DRIVES DEEP LEARNING PROGRESS
  - DATA
  - COMPUTATION
  - ALGORITHMS

- BINARY CLASSIFICATION

1 (CAT) VS. 0 (NON CAT)



$$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 255 \\ 134 \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix} \quad n = n_x = 12288$$

$64 \times 64 \times 3 = 12288$

$(x, y) \Rightarrow x \in \mathbb{R}^{n_x}$

$y \in \{0, 1\}$

TRAINING EXAMPLES: $\{ (x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)}) \}$

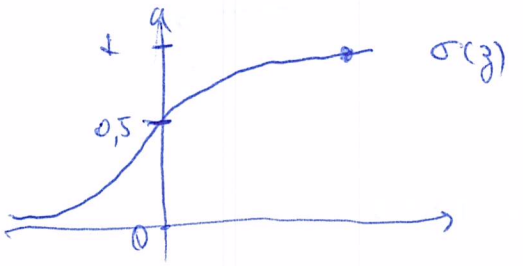
$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad \downarrow n_x$$

- LOGISTIC REGRESSION

x, $\hat{y} = P(y=1|x)$, $0 \le \hat{y} \le 1$, $x \in \mathbb{R}^{n_x}$

PARAMETERS: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

OUTPUT: $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$, $z = w_1 x_1 + w_2 x_2 + b$ (FOR EXAMPLE)



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

- LOGISTIC REGRESSION COST FUNCTION

LOSS (ERROR) FUNCTION:

$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ $\Rightarrow$  (LOCAL MAXIMA)

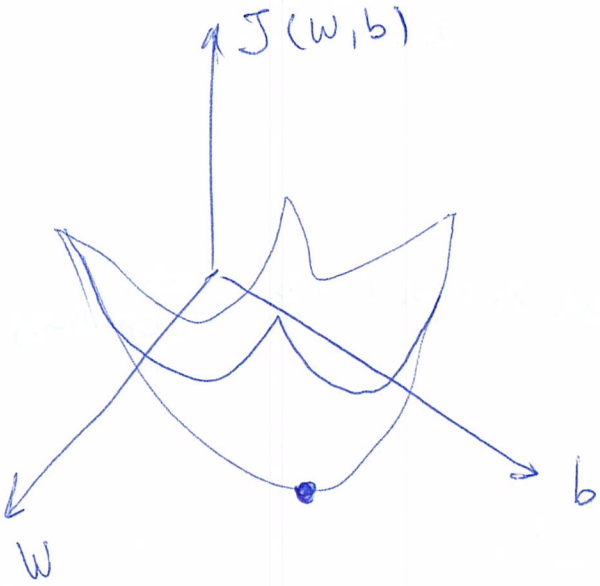
$L(\hat{y}, y) = -\left[y \log \hat{y} + (1-y) \log(1-\hat{y}) \right]$

IF $y=1$: $L(\hat{y}, 1) = -\log \hat{y}$ { WANT $\log \hat{y}$ LARGE $\Rightarrow$ $\hat{y}$ LARGE

IF $y=0$: $L(\hat{y}, 0) = \log(1-\hat{y})$ { WANT $\log 1-\hat{y}$ LARGE $\Rightarrow$ $\hat{y}$ SMALL

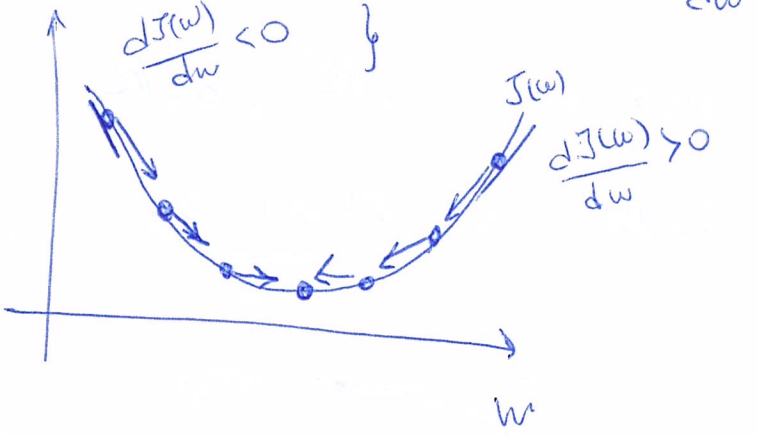
$\therefore$ COST FUNCTION: $J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

- GRADIENT DESCENT



REPEAT {

$$w := w - \alpha \frac{dJ(w)}{dw}$$



- LOGISTIC REGRESSION ON m EXAMPLES

$J=0 \; ; \; dw_1 = 0 \; ; \; dw_2 = 0 \; ; \; db = 0$

FOR $i = 1$ TO m

$$z^{(i)} = w^T x^{(i)} + b$$

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

$$J \mathrel{+}= -\left[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log (1 - a^{(i)}) \right]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$

$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$

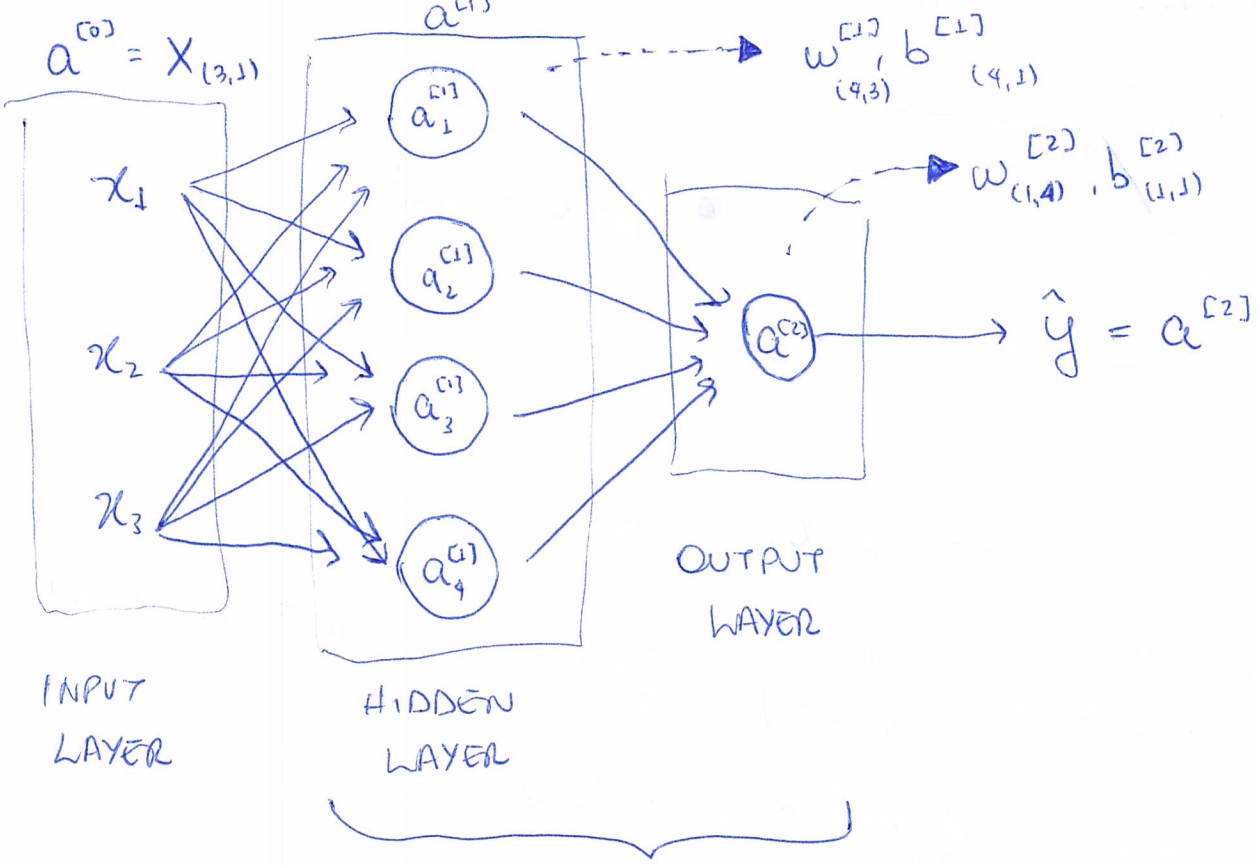
$n = 2$

$$db \mathrel{+}= dz^{(i)}$$

$J \mathrel{/}= m$

$dw_1 \mathrel{/}= m \; ; \; dw_2 \mathrel{/}= m \; ; \; db \mathrel{/}= m$

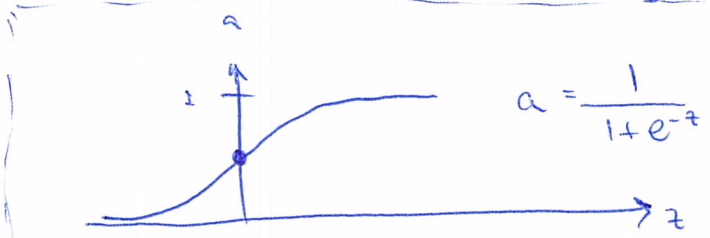
- NEURAL NETWORK REPRESENTATION



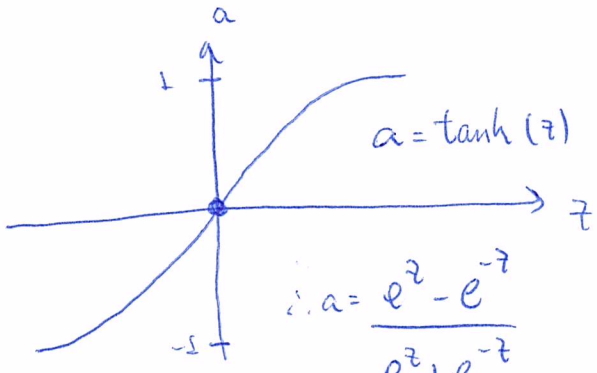
"2 LAYER NN"

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

$$\therefore a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \\ \sigma(z_4^{[1]}) \end{bmatrix} = \sigma(z^{[1]})$$

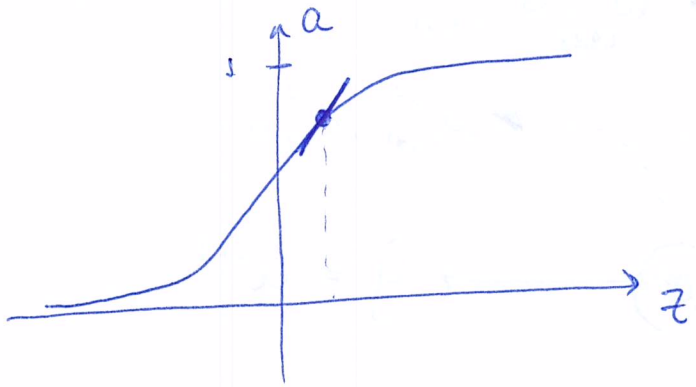


$0 \leq a \leq 1$



$-1 \leq a \leq 1$

- DERIVATIVES OF ACTIVATION FUNCTIONS



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

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

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

$z = 10$: $g(z) \approx 1 \Rightarrow \frac{d}{dz}(g(z)) \approx 1 \cdot (1-1) \approx 0$

$z = -10$: $g(z) \approx 0 \Rightarrow \frac{d}{dz}(g(z)) \approx 0 \cdot (1-0) \approx 0$

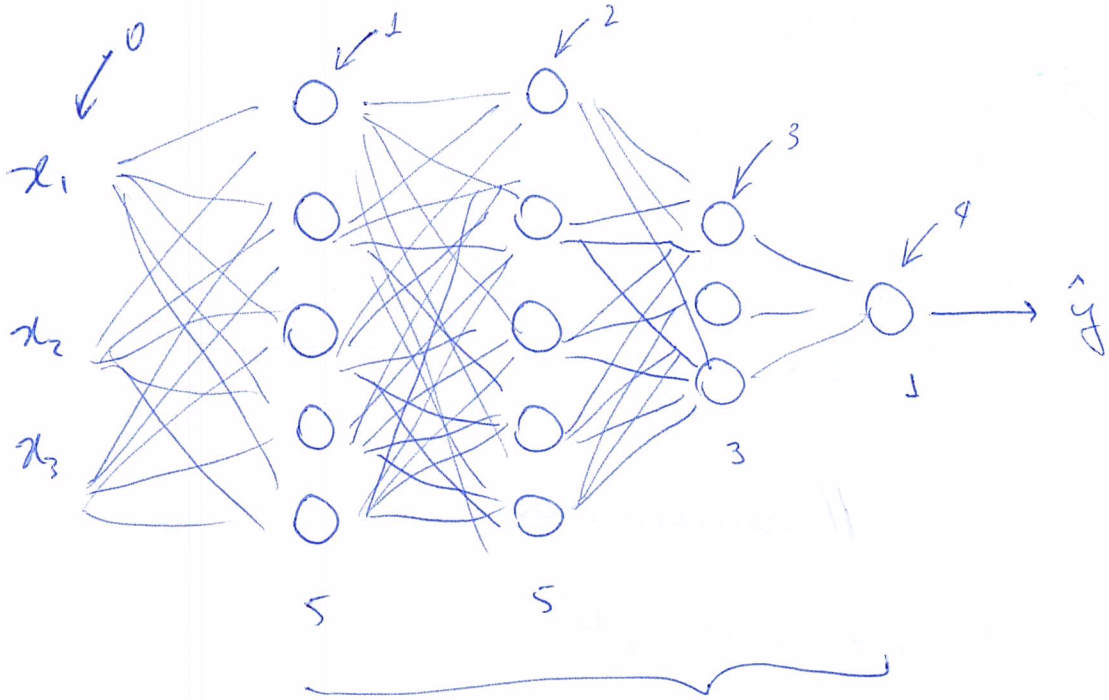
$z = 0$: $g(z) = 1/2 \Rightarrow \frac{d}{dz}(g(z)) = \frac{1}{2} \cdot (1 - 1/2) = \frac{1}{4}$

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

- $g(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$ $\quad g'(z) = \frac{d}{dz} g(z) = 1 - (\tanh(z))^2$

$a = g(z)$, $g'(z) = 1 - a^2$

- DEEP L-LAYER NETWORK



$L = 4$ (# LAYERS)

$n^{[l]}$ = # UNITS IN LAYER l

$a^{[l]}$ = activations in LAYER l

$n^{[1]} = 5, \quad n^{[2]} = 5, \quad n^{[3]} = 3, \quad n^{[4]} = n^{[L]} = 1, \quad n^{[0]} = 3$

- FORWARD PROPAGATION IN A DEEP NETWORK

USING THE LAST EXAMPLE: $x = a^{[0]}$

$$z^{[1]} = w^{[1]} x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = w^{[2]} \cdot a^{[1]} + b^{[1]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$
$$\vdots$$
$$z^{[4]} = w^{[4]} \cdot a^{[3]} + b^{[4]}$$
$$a^{[4]} = g^{[4]}(z^{[4]}) = \hat{y}$$

$$z^{[l]} = w^{[l]} \cdot a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$

$$z^{[1]} = W^{[1]} \cdot x + b^{[1]}$$

$$(3,1) \leftarrow (3,2) \cdot (2,1) + (3,1)$$

$$(n^{[1]},1) \leftarrow (n^{[1]},n^{[0]})(n^{[0]},1) + (n^{[1]},1)$$

$\therefore$ $dW^{[l]} = (n^{[l]}, n^{[l-1]})$

$db^{[l]} = (n^{[l]}, 1)$ } DIMENSIONS

- PARAMETERS VS. HYPERPARAMETERS

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

HYPERPARAMETERS:

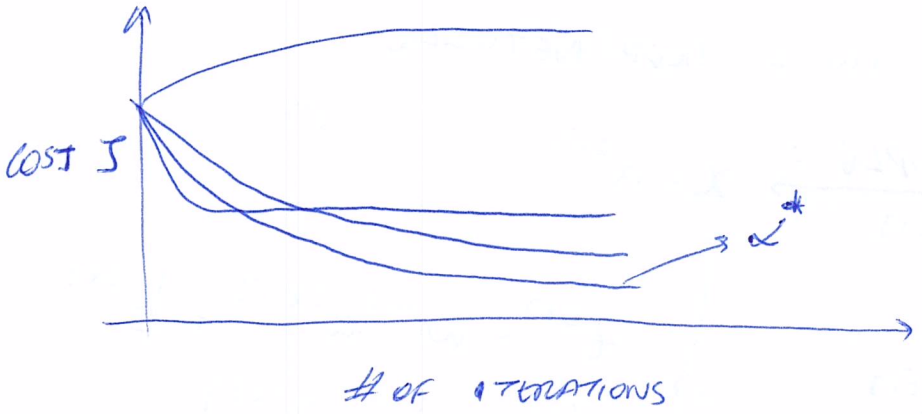
LEARNING RATE α

ITERATIONS

HIDDEN LAYERS L

HIDDEN UNITS $n^{[1]}, n^{[2]}, \ldots$

CHOICE OF ACTIVATION FUNCTION



APPLIED DEEP LEARNING IS A VERY EMPIRICAL PROCESS

COURSE 2 - IMPROVING DEEP NEURAL NETWORKS

- TRAIN / VALID / TEST SETS SIZES

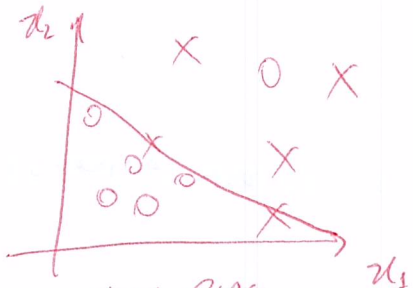
BEFORE: 70% (TRAIN) / 30% (TEST)

OR

60% (TRAIN) / 20% (VALID) / 20% (TEST)

BIG DATA ERA: 1,000,000 + => 98% / 1% / 1%

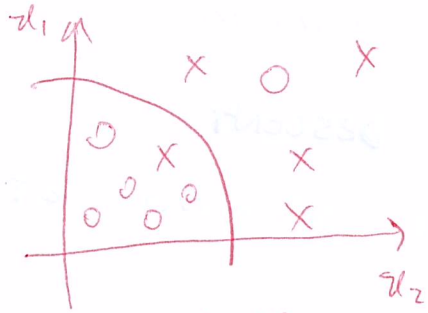
- BIAS AND VARIANCE



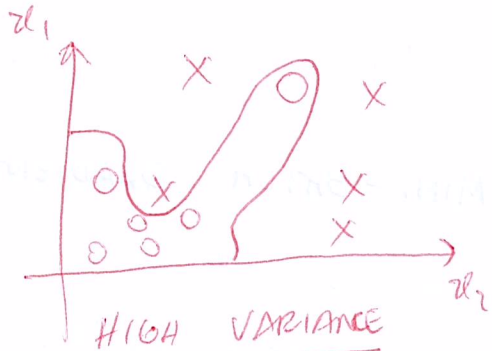
HIGH BIAS

||

UNDERFITTING



JUST RIGHT



HIGH VARIANCE

||

OVERFITTING

- SOLVE BIAS / VARIANCE PROBLEMS

HIGH BIAS? ⟶
- BIGGER NETWORK
- BIGGER TRAINING SET

HIGH VARIANCE? ⟶
- MORE DATA (MORE VARIABILITY)
- REGULARIZATION

- REGULARIZATION (L2)

$$J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2n} \sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

COST FUNCTION — LOSS FUNCTION — REGULARIZATION FUNCTION

ELIMINATES SOME "EXTRA" NEURONS.

(λ := REGULARIZATION PARAMETER)

- NORMALIZING INPUTS

x_1 : 1 ... 100
x_2 : 0 ... 1

} $w_1 \gg w_2$ => MORE TIME / ITERATIONS FOR GRADIENT DESCENT

- REGULARIZATION WITH DROPOUT

DURING TRAINING ELIMINATES (RANDOMLY) NODES, USING A VARIABLE OF KEEP-PROB.

* IMPORTANT : ONLY IN TRAINING

- MINI-BATCH GRADIENT DESCENT

CREATES BATCHS FOR TRAINING SET (M OBSERVATIONS), AND DIVIDES SETS.

MINI-BATCH SIZE NOT TOO BIG / SMALL

⇓

FASTEST LEARNING
- VECTORIZATION
- MAKE PROGRESS WITHOUT PROCESSING ENTIRE SET

OR
- BATCH GRADIENT DESCENT (SIZE=m)
  ⇓
  TOO LONG PER ITERATION
- STOCHASTIC GRADIENT DESCENT (SIZE=1)
  ⇓
  LOSE SPEED

IF SMALL TRAINING SET : USE BATCH GRADIENT DESCENT
($m \leq 2000$)

TYPICAL SIZES OF MINI-BATCHS: 64, 128, 256, 512
2^6, 2^7, 2^8, 2^9

- ADAM OPTIMIZATION ALGORITHM

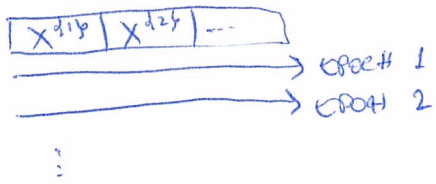
MOMENTUM ⊕ RMSPROP = ADAM

"GRADIENT DESCENT MOMENTUM"

ADAM := ADAPTED MOMENT ESTIMATION

- LEARNING RATE DECAY

1 EPOCH = 1 PASS THROUGH THE DATA



$$\alpha = \frac{1}{1 + \text{DECAY\_RATE} * \text{EPOCH\_NUM}} \alpha_0$$

$\alpha_0 = 0.2$

DECAY_RATE = 1

EPOCH	α
1	0.1
2	0.67
3	0.5
⋮	⋮

- HYPERPARAMETER TUNING

1° α

2° β

β_1, β_2, ε
0.9, 0.999, 10^{-8}

3° # LAYERS

2° # HIDDEN UNITS

3° LEARNING RATE DECAY

2° MINI-BATCH SIZE

BABYSITTING ONE MODEL vs. TRAINING MANY MODELS IN PARALLEL

- BATCH NORMALIZATION

FEATURE NORMALIZATION $\Rightarrow$ SPEED UP LEARNING

USING BATCH NORMALIZATION :

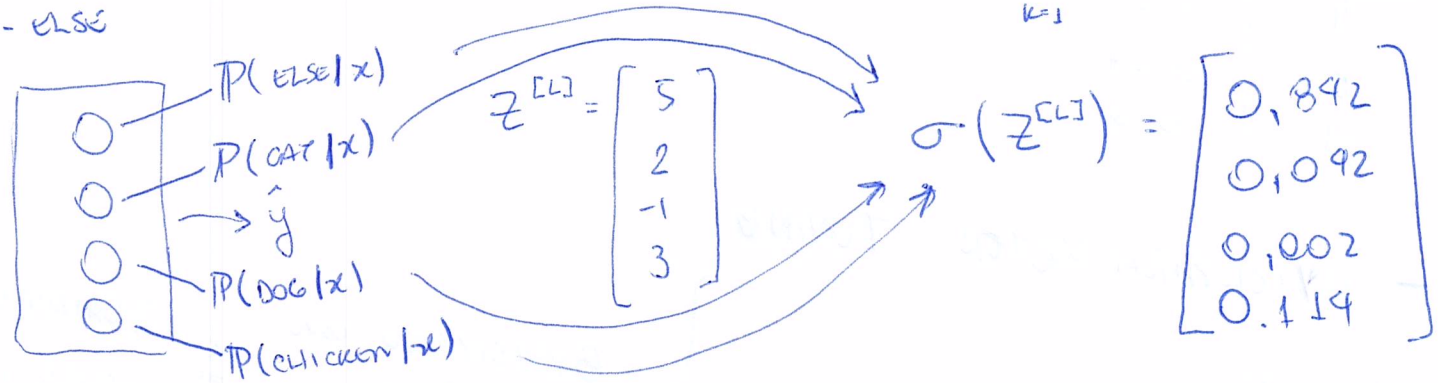
- WE CAN USE HIGHER LEARNING RATES, CAUSE ACTIVATION VALUES ARE NOW
- IT REDUCES OVERFITTING BECAUSE IT HAS A SLIGHT REGULARIZATION EFFECTS (ADD NOISE TO EACH HIDDEN LAYER)

- SOFTMAX REGRESSION (MULTI-CLASS CLASSIFICATION)

1- CAT
2- DOG
3- CHICKEN
0- ELSE

$C = \#\text{CLASSES} = 4$

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad j = 1, \dots, K$$



"HARD MAX" $\Rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ AND IF $C = 2$ $\Rightarrow$ SOFTMAX = LOGISTIC REGRESSION

- loss function

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{4} y_j \cdot \log \hat{y}_j \qquad y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \ \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$$

$$= -\log \hat{y}_2$$

$$\text{MIN } \mathcal{L}(\hat{y}, y) = \text{MIN} - \log \hat{y}_2 \Rightarrow \text{MAX } \hat{y}_2$$

COURSE 3 - STRUCTURING ML PROJECTS

WAYS TO ANALYZE PROBLEMS OF ML IN A WAY TO FIND SOLUTIONS (GOOD ONES).

CHAIN OF ASSUMPTIONS IN ML:

- FIT TRAINING SET WELL
- " DEV " "
- " TEST " "
- PERFORMS WELL IN REAL WORLD

USING A SINGLE NUMBER EVALUATION METRIC:

CLASSIFIER	PRECISION*	RECALL*	F1 SCORE
A	95%	90%	92.4%
B	98%	85%	91%

$$F1\ SCORE = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$ "HARMONIC MEAN"

↳ TO COMPARE PRECISION AND RECALL IN ONLY ONE NUMBER

*

		PREDICTED: NEGATIVE	PREDICTED: POSITIVE
ACTUAL	NEGATIVE	TN	FP
ACTUAL	POSITIVE	FN	TP

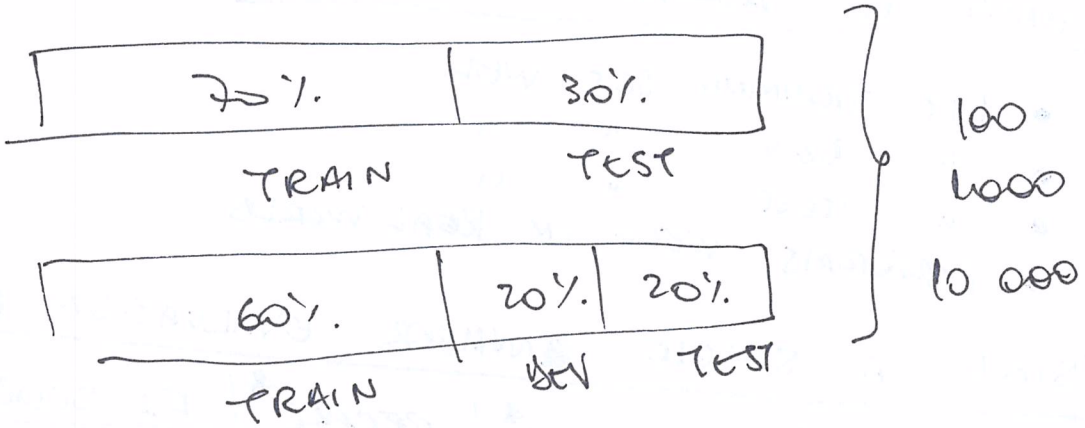
$$PRECISION = \frac{TP}{TP + FP}$$

$$RECALL = \frac{TP}{TP + FN}$$ (SENSITIVITY)

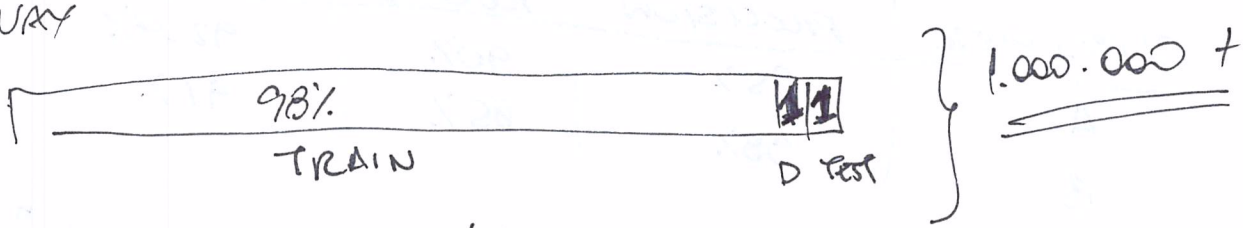
$$SPECIFICITY = \frac{TN}{TN + FP}$$

SIZE OF DEV/TRAIN/TEST SET

- OLD WAY OF SPLITTING



- NEW WAY



WHEN TO CHANGE DEV/TESTS SETS AND METRICS

→ IMPORTANT TO ANALYZE ERROR PUBLIC OVER TAX ERROR

$$\text{ERROR} = \frac{1}{\sum w^{(i)}} \frac{1}{M_{dev}} \sum_{i=1}^{M_{dev}} w^{(i)} \, \mathbb{1}\{y_{PRED}^{(i)} \neq y^{(i)}\}$$

$$w^{(i)} = \begin{cases} 1, & \text{IF } x^{(i)} \text{ IS NON-PORN} \\ 10, & \text{IF } x^{(i)} \text{ IS PORN} \end{cases}$$

UNDERSTANDING HUMAN-LEVEL PERFORMANCE

- HUMAN-LEVEL ERROR (PROXY FOR BAYES ERROR)

↕ "AVOIDABLE BIAS"

- TRAINING ERROR

↕ "VARIANCE"

- DEV ERROR

SURPASSING HUMAN-LEVEL PERFORMANCE

- TEAM OF HUMANS — 0.5% → PROBABLY IS THE BAYES ERROR

1% — 0.1% "AVOIDABLE BIAS"

- ONE HUMAN

- TRAINING ERROR — 0.6%

0.2% "VARIANCE"

- DEV ERROR — 0.8%

IMPROVING YOUR MODEL PERFORMANCE

HUMAN-LEVEL

↕ AVOIDABLE BIAS →
- TRAIN BIGGER MODEL
- TRAIN LONGER/BETTER OPTIMIZATION ALGORITHMS
- NN ARCHITECTURE / HYPERPARAMETERS SEARCH (RNN, CNN) *

TRAINING ERROR

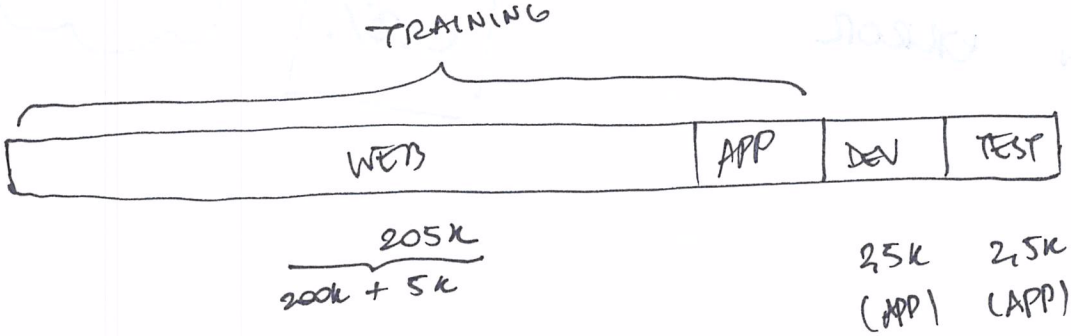
↕ VARIANCE →
- MORE DATA
- REGULARIZATION L2, DROPOUT, DATA AUGMENTATION

DEV ERROR

- (*)

MISMATCHED TRAINING AND DEV/~~SELF~~ TEST SET

- 200k obs from WEB
- 10k obs from APP (THE ONES YOU WANT TO IDENTIFY)

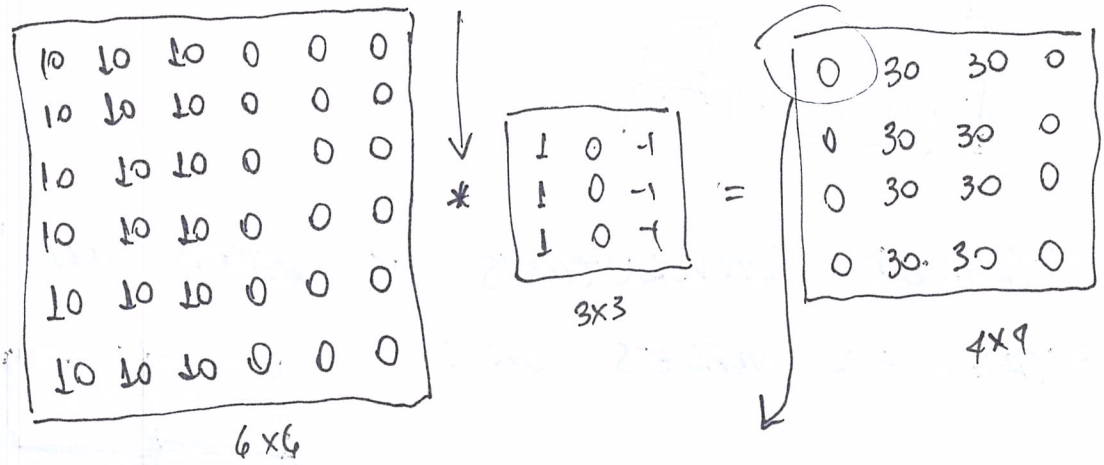


COURSE 4 - CONVOLUTIONAL NEURAL NETWORKS

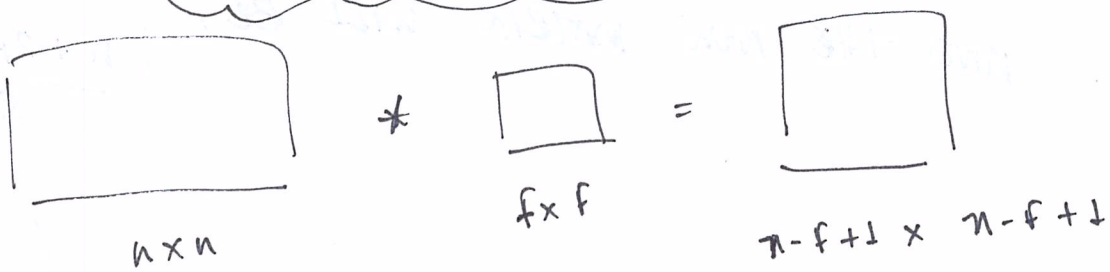
VERY USED ON COMPUTER VISION

How convolution works?

CONVOLUTION OPERATOR



$10 \cdot 1 + 10 \cdot 1 + 10 \cdot 1 + 10 \cdot 0 + 10 \cdot 0 + 10 \cdot 0 + 10 \cdot (-1) + 10 \cdot (-1) + 10 \cdot (-1) = 0$



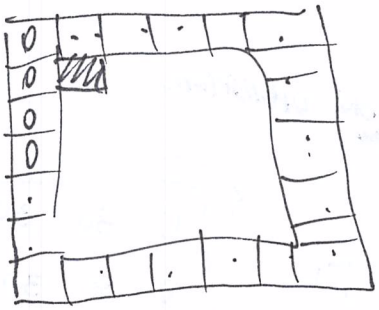
"VALID" AND "SAME" CONVOLUTIONS

- "VALID": $n \times n \quad * \quad f \times f \rightarrow n-f+1 \times n-f+1$
- "SAME": PAD (1) SO THE OUTPUT SIZE IS THE SAME AS THE INPUT SIZE.

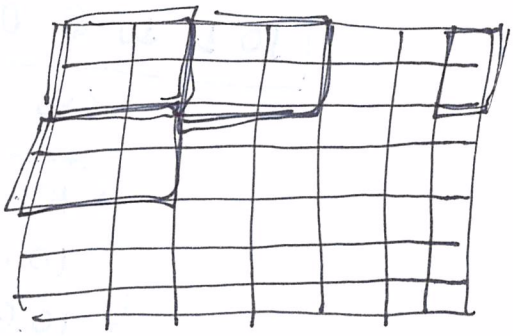
$$n + 2p - f + 1 = n \Rightarrow 2p = f - 1 \Rightarrow$$

$$\boxed{p = \frac{f-1}{2}}$$

(1) PADDING IS TO ADD COLUMNS TO THE IMAGES IN ORDER TO ENHANCE INFLUENCE OF CERTAIN VALUE OF THE IMAGE.

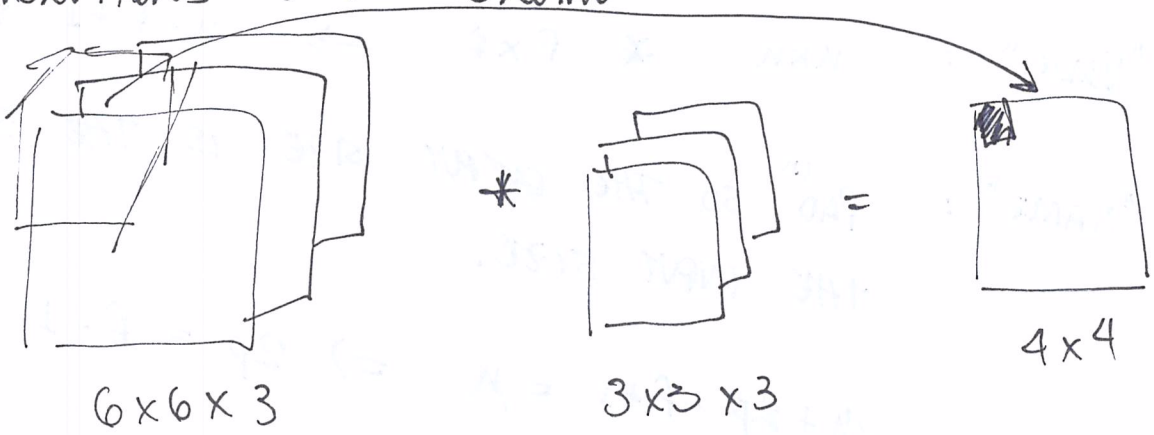


STRIDED CONVOLUTIONS IS BASED ON "JUMPS", SO STRIDE = 2 IMPLIES ON:



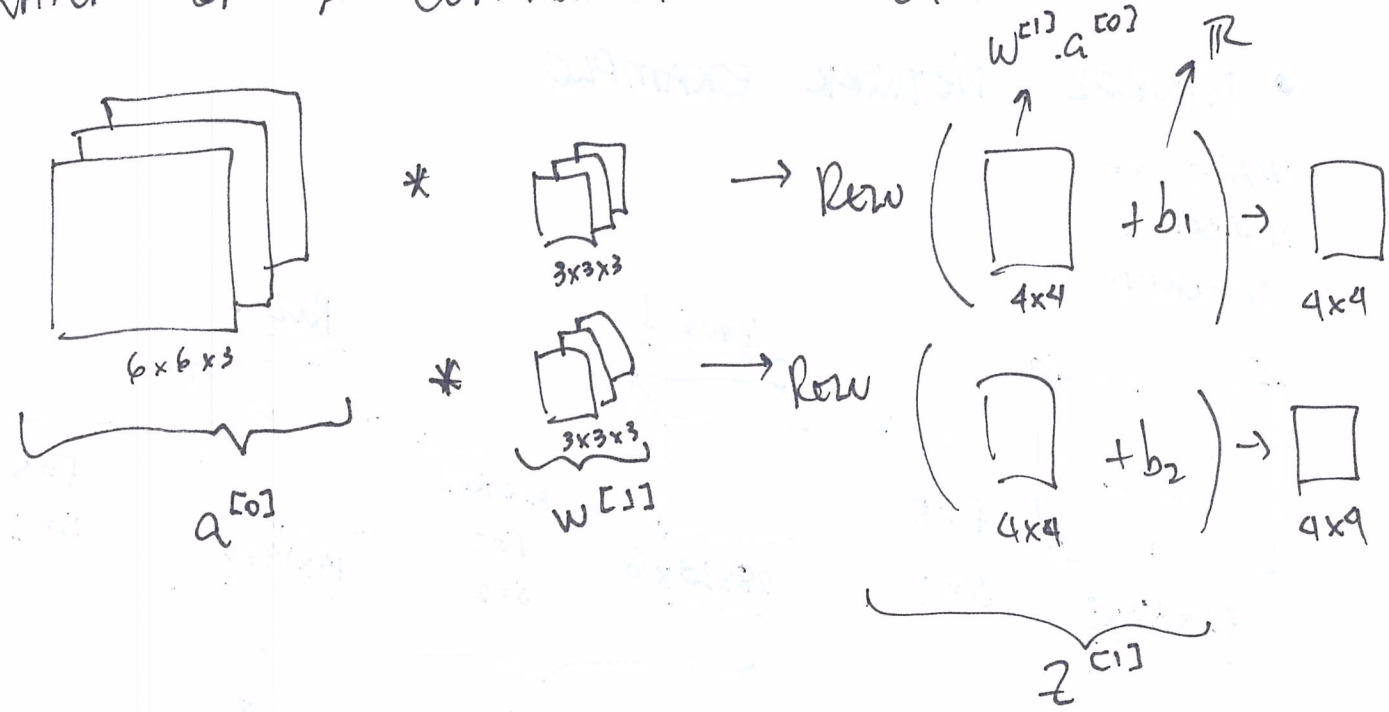
AND THE NEW MATRIX WILL BE: $\left(\frac{n+2p-f}{s}+1\right) \times \left(\frac{n+2p-f}{s}+1\right)$

CONVOLUTIONS OVER VOLUME:



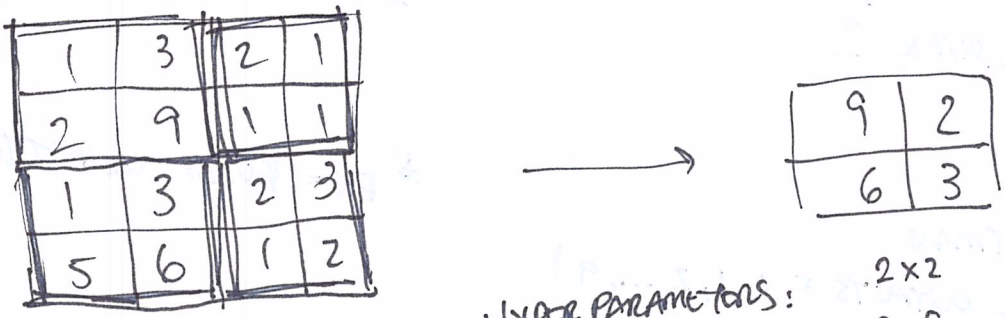
"MULTIPLE FILTERS"

- LAYER OF A CONVOLUTIONAL NETWORK



$$\begin{cases} z^{[1]} = W^{[1]} a^{[0]} + b^{[1]} \\ a^{[1]} = g(z^{[1]}) \end{cases}$$

- POOLING LAYERS : MAX POOLING

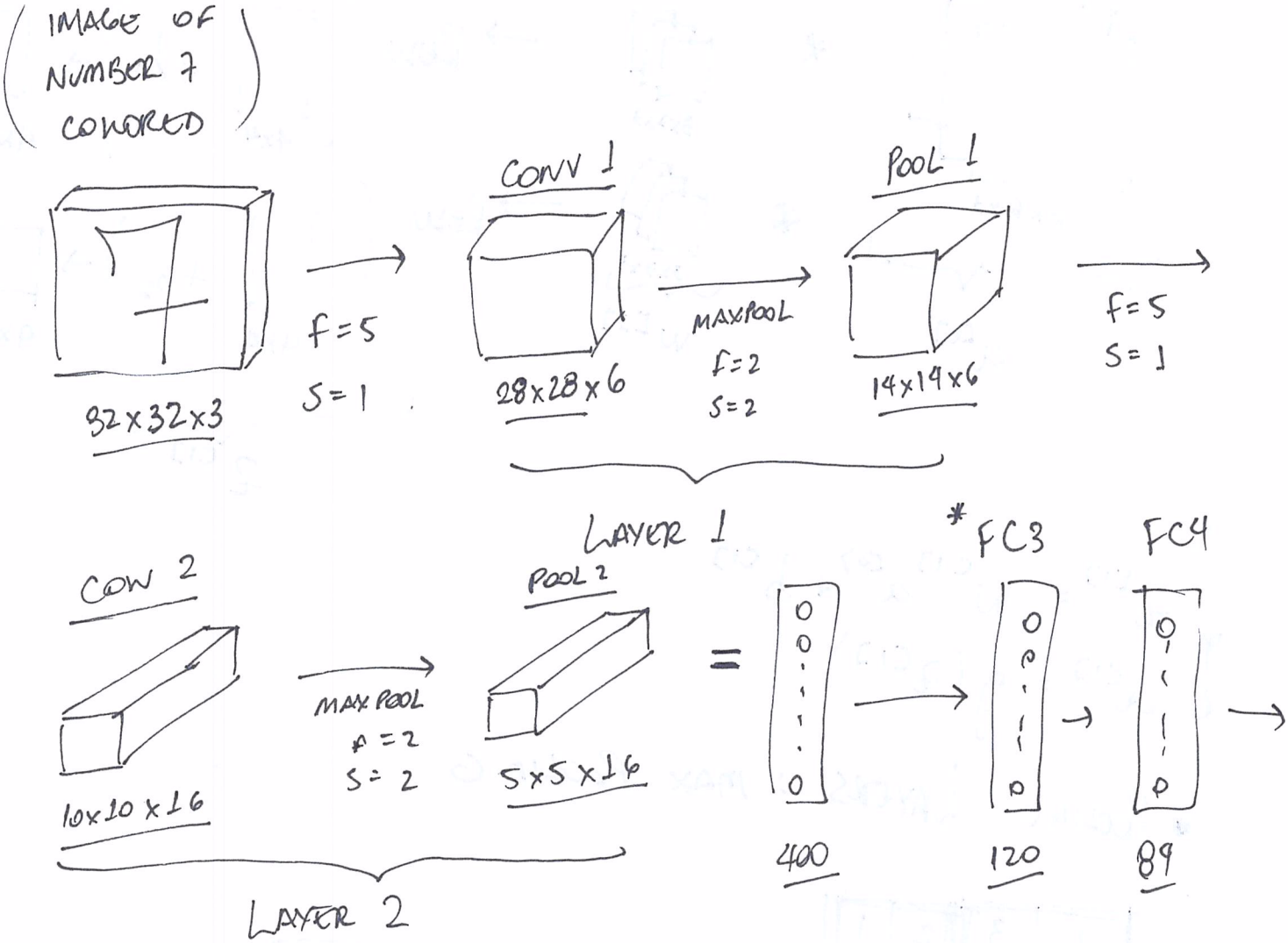


HYPERPARAMETERS: f×f

f = 2 (FILTER DIMENSION)

S = 2 (STRIDE)

- NEURAL NETWORK EXAMPLE



* FC = FULLY CONNECTED LAYER

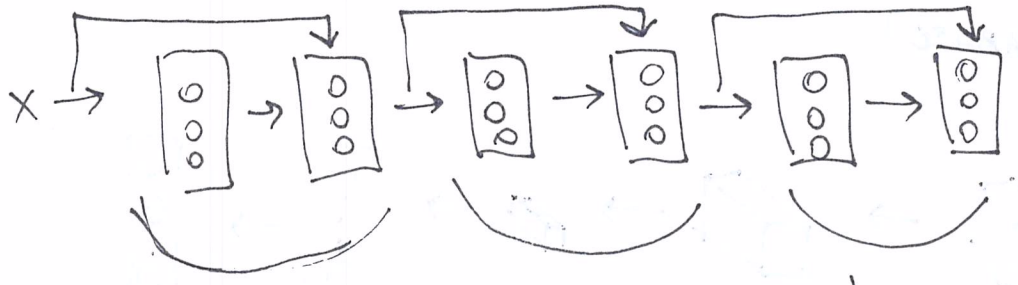
→ ○ SOFTMAX
(10 OUTPUTS = 0, 1, 2, ..., 9)

- WHY CONVOLUTIONS?

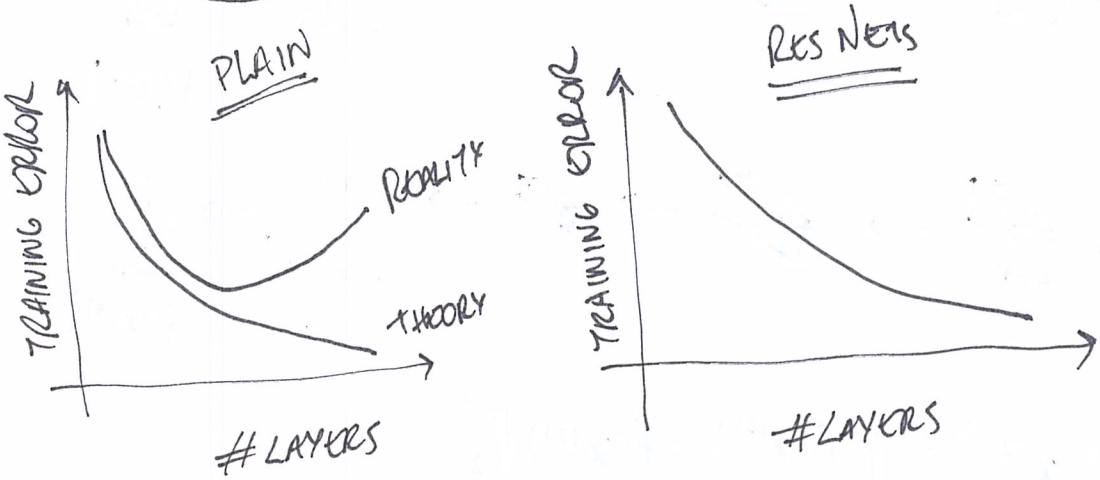
→ PARAMETER SHARING: A FEATURE DETECTOR THAT'S USEFUL IN ONE PART OF THE IMAGE MIGHT BE USEFUL IN ANOTHER PART OF THE IMAGE.

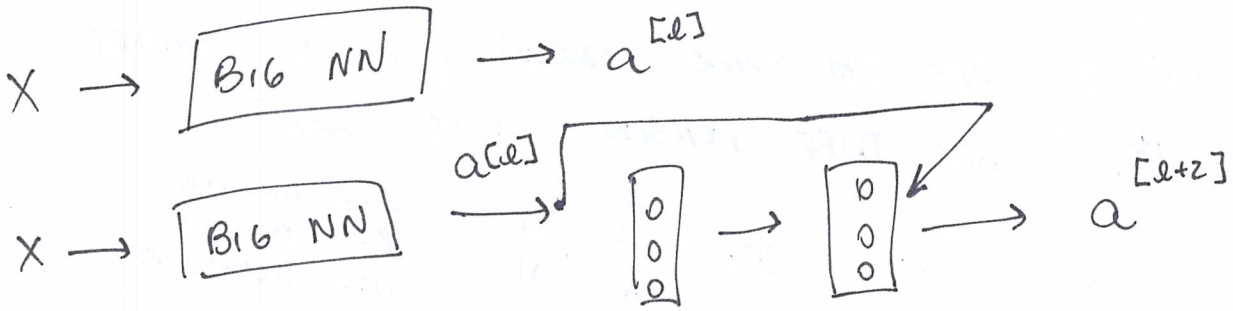
→ SPARSITY CONNECTIONS: IN EACH LAYER, EACH OUTPUT VALUE DEPENDS ONLY ON A SMALL NUMBER OF INPUTS.

- RESIDUAL NETWORK (RESNETS)



3 RESIDUAL BLOCKS





$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$$

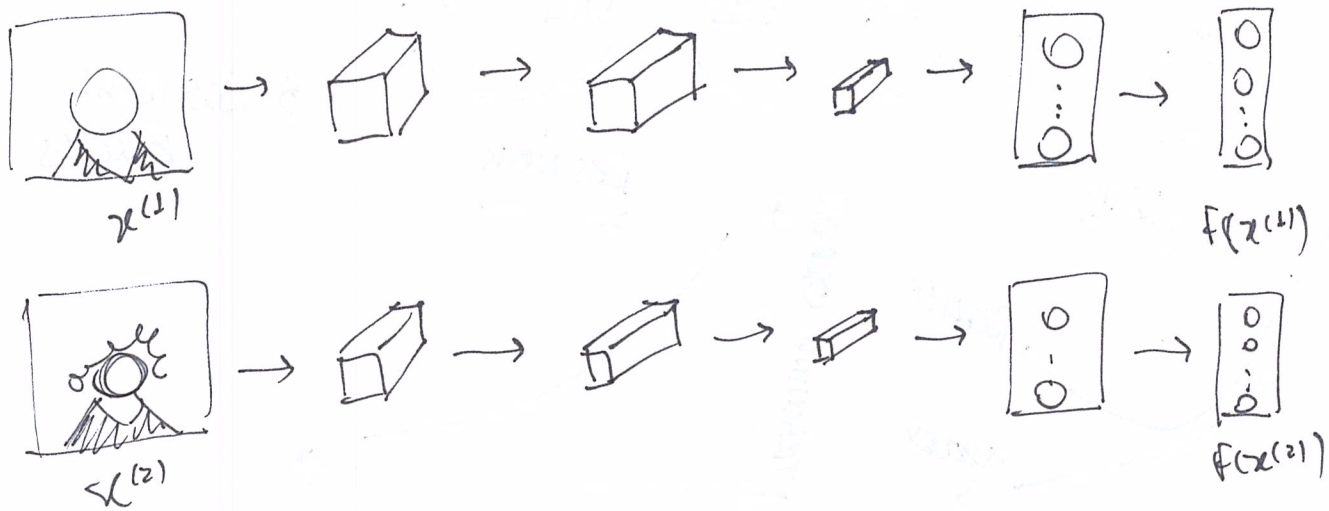
$$= g\left(\underbrace{w^{[l+2]} a^{[l+1]} + b^{[l+2]}}_{=0} + a^{[l]}\right) \quad \overset{(*)\ \text{RELU}}{=} g(a^{[l]}) = a^{[l]}$$

RESIDUAL NETWORKS SKIP CONNECTIONS OR SHORT-CUTS TO JUMP OVER SOME LAYERS, THE MOTIVATION IS TO AVOID THE PROBLEM OF VANISHING GRADIENTS BY REUSING ACTIVATION FROM A PREVIOUS LAYER UNTIL THE LAYER NEXT TO THE CURRENT ONE HAVE LEARNED ITS WEIGHTS.

THE INTUITION ON WHY THIS WORKS IS THAT THE NEURAL NETWORK COLLAPSES INTO FEWER LAYERS INITIALLY, AND AFTER EXPAND

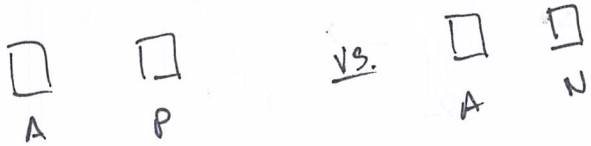
- SIAMESE NETWORK

USED TO SOLVE ONE SHOT LEARNING (IDENTIFY IF PHOTO EXISTS ON DATABASE)



$$d(x^{(1)}, x^{(2)}) = \| f(x^{(1)}) - f(x^{(2)}) \|_2^2$$

- IF $x^{(1)}, x^{(2)}$ ARE THE SAME PERSON, d IS SMALL
- IF $x^{(2)}, x^{(2)}$ ARE DIFF. PERSON, d IS LARGE



A := ANCHOR
P := POSITIVE
N := NEGATIVE

LOSS FUNCTION (= "TRIPLET LOSS") : $\mathcal{L}(A,P,N) = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0)$

$\|f(A) - f(P)\|^2 + \alpha \leq \|f(A) - f(N)\|^2$

$d(A,P)$ $\qquad$ $d(A,N)$