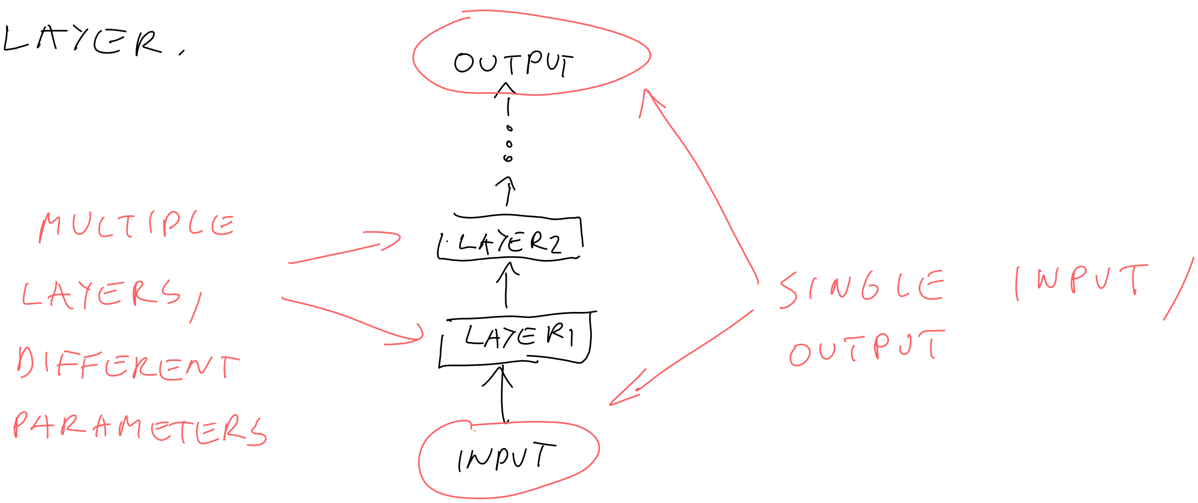
ML TUTORIAL 3 - RNNs

FEEDFORWARD (MLP) VS RECURRENT (RNN):

- FEEDFORWARD HAS NO FEEDBACK CONNECTION. THE INFORMATION FLOWS IN ONE DIRECTION.

THE INPUT IS USUALLY SEEN BY THE NETWORK ONLY IN THE FIRST LAYER.



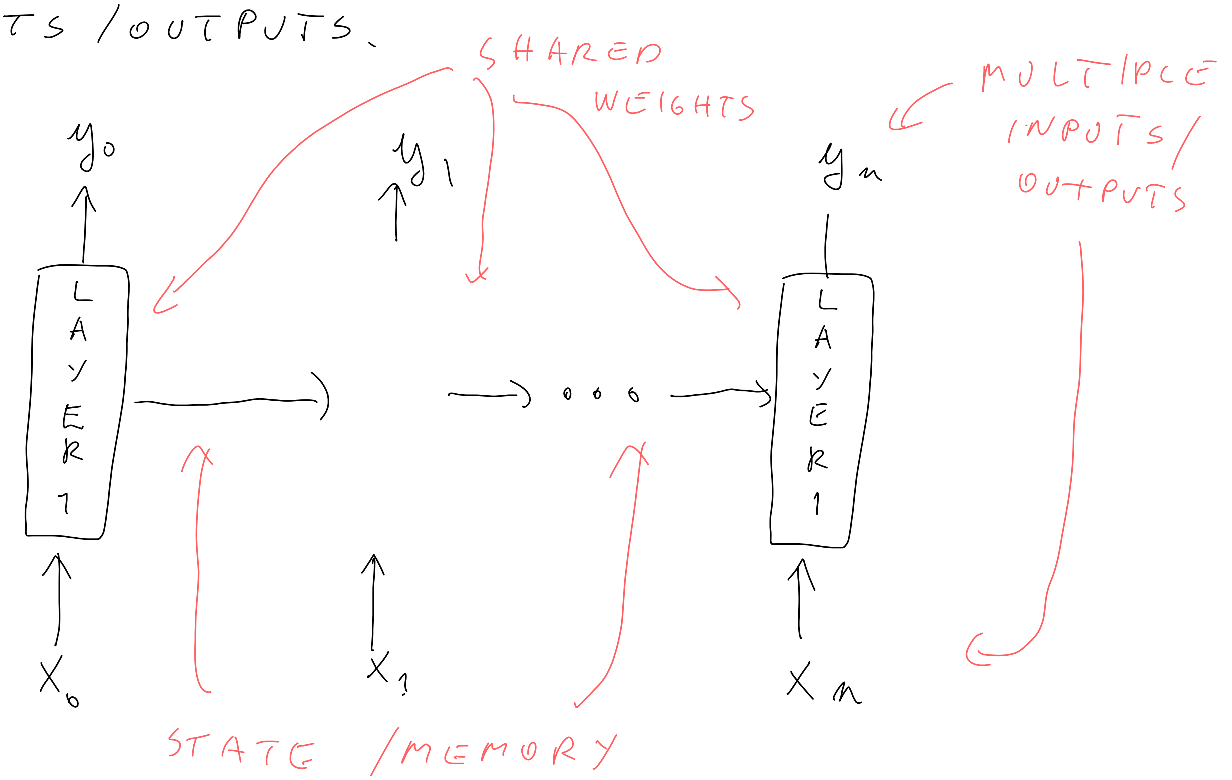
THEY ARE WORKING WITH A SINGLE INPUT AND SINGLE OUTPUT.
THEY CAN WORK WITH FIXED LENGTH SEQUENCES, IF THE WHOLE SEQUENCE IS PRESENTED AS INPUT.

THEY HAVE NO PARAMETER SHARING BETWEEN LAYERS.

- RECURRENT NEURAL NETWORKS (RNNs)

THEY HAVE FEEDBACK CONNECTIONS.
THEY USUALLY HAVE MUCH FEWER LAYERS (IF THEY HAVE MULTIPLE, EACH LAYER PROCESSES THE WHOLE SEQUENCE, BEFORE IT IS PASSED TO THE NEXT LAYER. HERE WE WILL CONSIDER ONLY SINGLE LAYER RNNs).

RNNs RECEIVE MULTIPLE INPUTS AND PRODUCE MULTIPLE OUTPUTS.
THE COMPUTATION IS DIVIDED TO (TIME) STEPS. IN EACH STEP AN INPUT IS READ, THE SAME TRANSFORMATION IS PERFORMED AND AN OUTPUT IS PRODUCED.
THE INPUT → OUTPUT TRANSFORMATION IS DEPENDENT ON THE HISTORY OF PREVIOUS INPUTS/OUTPUTS.



RNNs CAN WORK WITH ARBITRARY SEQUENCE LENGTHS, THAT CAN CHANGE FROM SAMPLE TO SAMPLE.

SIMPLE RNN

$$\bar{h}_t = \sigma\left(\bar{x}_t \underline{W} + \bar{h}_{t-1} \underline{U} + \bar{b}\right) \qquad \bar{h}_{-1} = 0$$

$$\bar{y}_t = \bar{h}_t \underline{V}$$

$\bar{h}_t$ - STATE

$\bar{x}_t$ - INPUTS

$\bar{y}_t$ - OUTPUTS

$\underline{W}$ - INPUT → HIDDEN WEIGHTS

$\underline{U}$ - HIDDEN → HIDDEN WEIGHTS

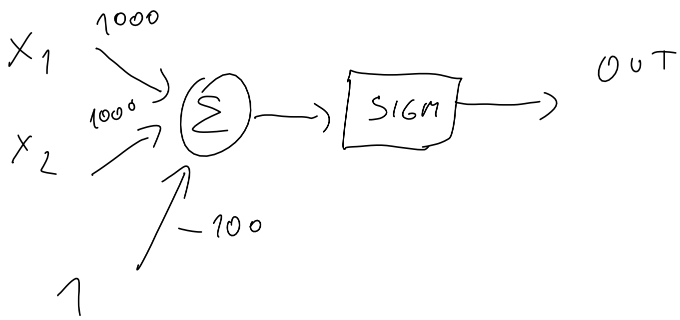
$\bar{b}$ - BIASES

$\underline{V}$ - OUTPUT WEIGHTS

σ - ACTIVATION FUNCTION (USUALLY TANH)

RNNs ARE UNIVERSAL COMPUTERS

IT IS EASY TO SEE. YOU CAN BUILD A NOR GATE:



x_1	x_2	o
1	1	$\sigma(1900) \sim 1$
1	0	$\sigma(900) \sim 1$
0	1	$\sigma(900) \sim 1$
0	0	$\sigma(-100) \sim 0$

ANY LOGIC GATE CAN BE BUILT OUT OF NOR GATES.

NOW YOU CAN TRANSLATE ARBITRARY DIGITAL CIRCUIT (LIKE A COMPUTER) TO AN RNN. MAP EACH WIRE TO AN ELEMENT IN THE STATE, AND EACH GATE TO THE CORRESPONDING PART OF THE WEIGHT MATRICES.

RECURRENCE IS NEEDED HERE BECAUSE THE OUTPUT OF EACH GATE IS CONNECTED TO THE INPUT OF ANOTHER.

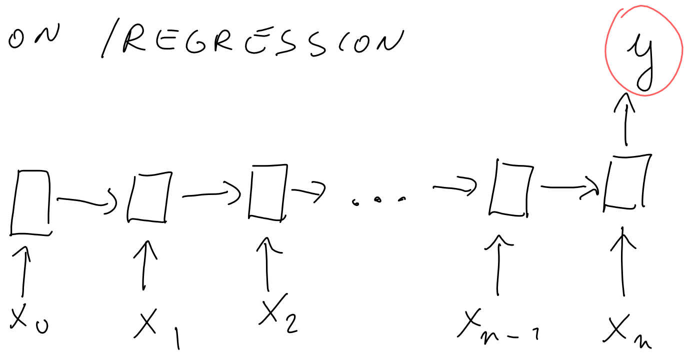
NOTE: THIS IS JUST THE REPRESENTATION POWER. IT DOES NOT MEAN THAT WE CAN LEARN IT WITH GRADIENT DESCENT.

WEIGHT SHARING

IN EACH STEP WE ARE USING THE SAME SET OF WEIGHTS. THIS HAS HUGE ADVANTAGES: THE LEARNED FUNCTIONS CAN BE REUSED. FOR EXAMPLE FOR TRANSLATION, YOU DON'T WANT TO RE-LEARN THE TRANSFORMATION WORK → ARBEIT FOR EACH POSSIBLE POSITION. THEN YOU WOULD NEED MANY SAMPLES OF EACH POSSIBLE WORD IN EACH POSSIBLE POSITION.

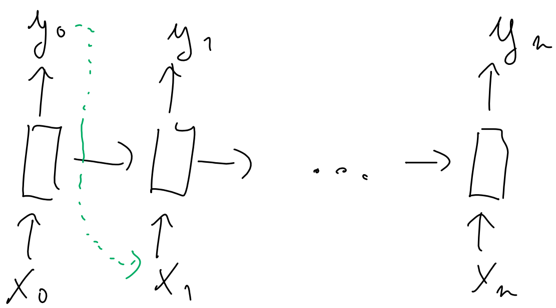
TYPES OF PROBLEMS

- MULTIPLE INPUTS - SINGLE OUTPUT: CLASSIFICATION / REGRESSION



OUTPUT IS ONLY PRODUCED IN THE LAST STEP.

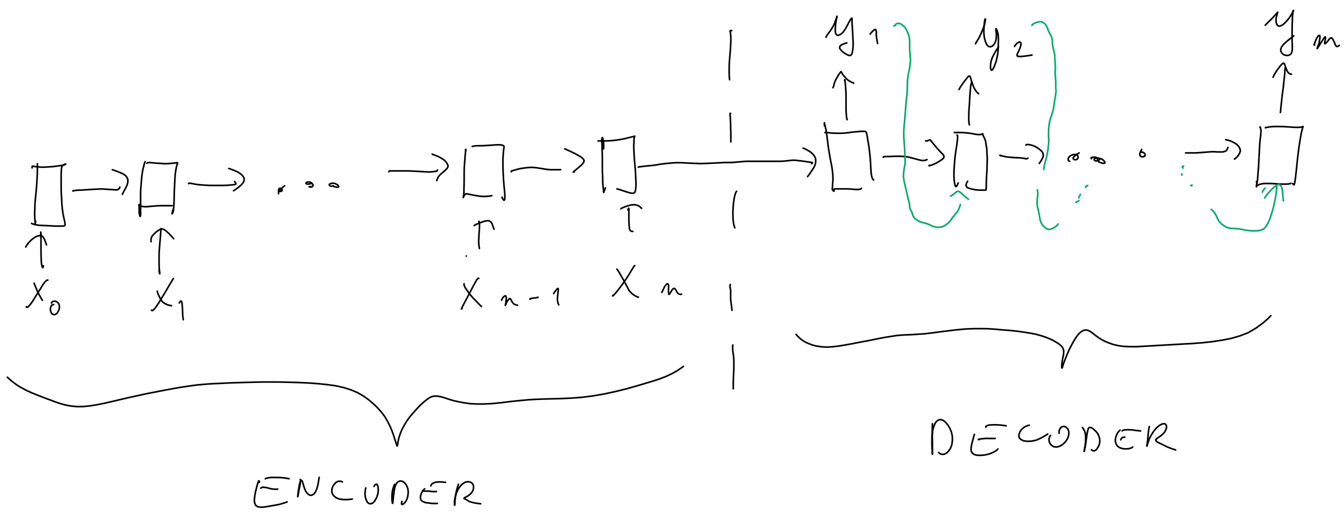
- SAME LENGTH SEQ-TO-SEQ, OR SINGLE STEP PREDICTION:



---> OPTIONAL: FOR SEQUENCE PREDICTION PROBLEMS A SAMPLE FROM THE PREVIOUS OUTPUT IS FED AS AN INPUT FOR THE NEXT STEP. TYPICAL FOR LANGUAGE MODELS.

- DIFFERENT LENGTH SEQ-TO-SEQ

FOR EXAMPLE TRANSLATION (GOOGLE TRANSLATE)



ENCODER AND DECODER DO NOT SHARE WEIGHTS.

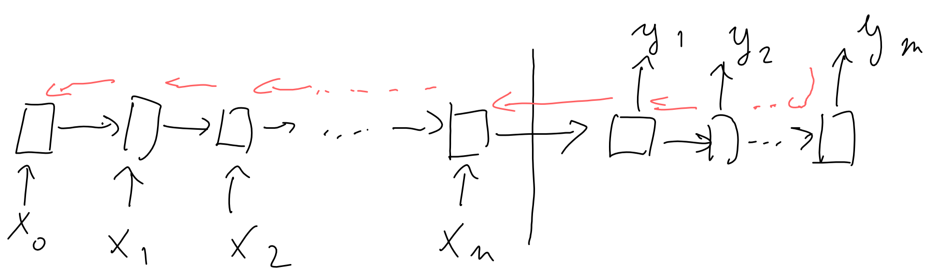
THEY COMMUNICATE WITH THE LAST HIDDEN STATE OF THE ENCODER.

THIS IS A BOTTLENECK. RECENT MODELS USE ATTENTION TO OVERCOME THIS. (OUT OF SCOPE).

THE FUNDAMENTAL DEEP LEARNING PROBLEM: VANISHING AND EXPLODING GRADIENTS.

THIS IS TRUE FOR ALL MULTILAYER NETWORKS, JUST IT IS MORE PRONOUNCED FOR RNNS, BECAUSE THEY USUALLY HAS MANY MORE TIME-STEPS THAN MLPS HAS LAYERS, AND THEY USE THE SAME SET OF WEIGHTS FOR ALL.

CREDIT ASSIGNMENT PATH IS THE PATH THAT THE GRADIENTS SHOULD TAKE THROUGH THE COMPUTATION GRAPH TO REACH THE PARAMETERS OF INTEREST. RNNS HAS VERY LONG CREDIT ASSIGNMENT PATHS, BECAUSE THE GRAD STARTS FROM THE END OF THE SEQUENCE AND HAS TO PROPAGATE ALL THE WAY TO THE BEGINNING



← THE LONGEST CREDIT ASSIGNMENT PATH

THE PROBLEM IS THAT IF THE APPLIED TRANSFORMATION CHANGES THE LENGTH OF THE GRADIENT VECTOR.

INTUITION: IGNORE NONLINEARITIES. IN THIS CASE WE HAVE A MATRIX MULTIPLICATION IN EVERY STEP: $h_1 = \underline{A}\bar{x}$, $h_2 = \underline{A}\,\underline{A}\,\bar{x}$, $h_3 = \underline{A}^3\bar{x}$ ...

THE LENGTH OF h WILL EITHER GROW OR SHRINK. FOR SCALARS:

IF $A = 0.99$ $\lim_{n \to \infty} A^n = 0$

IF $A = 1.01$ $\lim_{n \to \infty} A^n = \infty$

$A = \underline{I}$ IS NOT THAT USEFUL TRANSFORMATION..

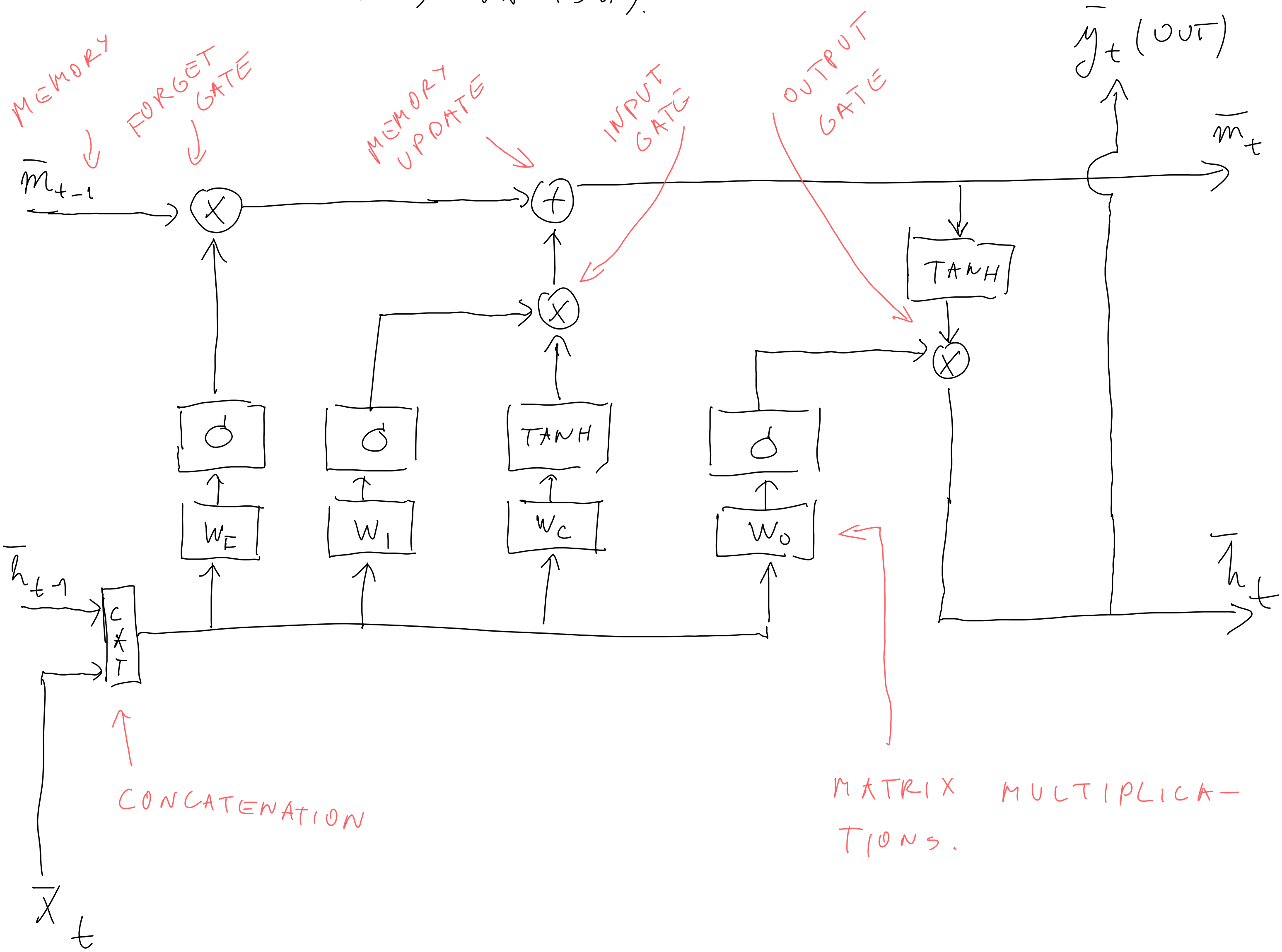
THIS IS CALLED THE VANISHING/EXPLODING GRADIENT PROBLEM: THE GRAD EITHER GROWS TO ∞ OR VANISHES TO 0 AS IT PROPAGATES THROUGH THE NETWORK, AND THE NETWOR WILL EITHER NOT TRAIN (VANISHING CASE) OR EXPLODE (NANs, ∞s, EXPLODING CASE).

THIS IS LESS PRONOUNCED IN MLPs, BECAUSE DIFFERENT TRANSFORMS CAN COMPENSATE EACH OTHER, AND THEIR CREDIT ASSIGNMENT PATH IS SHORTER.

LSTM

IDEA: USE A MEMORY CELL (IDENTITY TRANSFORMATION) TO CARRY THE GRADIENTS WITHOUT EXPLODING / VANISHING.

LSTM DOES <u>NOT SOLVE</u> THE EXPLODING / VANISHING GRAD PROBLEM, BUT IT SIGNIFICANTLY LESSENS ITS NEGATIVE EFFECT (THIS IS BECAUSE WE NEED GATES TO BE ABLE TO PERFORM INTERESTING COMPUTATION, AND THEY STILL CAN MAKE GRAD VANISH).



EQUATIONS

(1) FORGET OLD MEMORIES SELECTIVELY –

– <u>FORGET GATE</u>

$$\bar{f}_t = \sigma\left(\underline{W}_F^h \, \bar{h}_{t-1} + \underline{W}_F^i \, \bar{x}_t + \bar{b}_F\right)$$

(2) CREATE <u>UPDATE GATE</u> AND DIFFERENCE SIGNALS:

$$\bar{i}_t = \sigma\left(\underline{W}_i^h \, \bar{h}_{t-1} + \underline{W}_i^i \, \bar{x}_t + \bar{b}_i\right)$$

$$\bar{d}_t = \tanh\left(\underline{W}_c^h \, \bar{h}_{t-1} + \underline{W}_c^i \, \bar{x}_t + \bar{b}_c\right)$$

(3) UPDATE MEMORY

$$\bar{m}_t = \bar{f}_t \odot \bar{m}_{t-1} + \bar{i}_t \odot \bar{d}_t$$

ELEMENTWISE MULTIPLICATION

(4) PRODUCE OUTPUT

$$\bar{o}_t = \sigma\left(\underline{W}_o^h \, \bar{h}_{t-1} + \underline{W}_o^i \, \bar{x}_t + \bar{b}_o\right)$$

<u>OUTPUT GATE</u>

$$\bar{h}_t = \bar{o}_t \odot \tanh\left(\bar{m}_t\right)$$

STATE OF THE LSTM IS A TUPLE $(\bar{m}_t, \bar{h}_t)$, WHERE $\bar{h}_t$ IS USUALLY USED AS AN OUTPUT.

BOTH $\bar{m}_{-1}$ AND $\bar{h}_{-1}$ ARE USUALLY INITIALIZED TO 0.