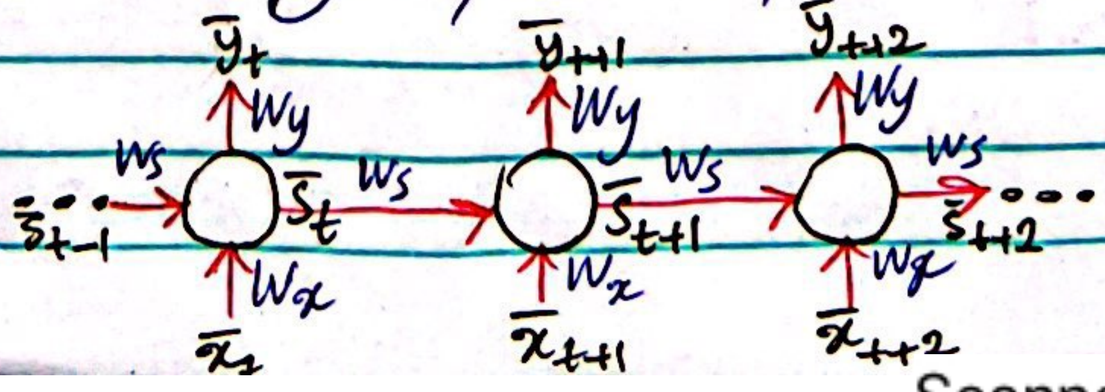
"RNN"

- Mini batch training: Updating the weights once every N steps: $\delta = \frac{1}{N} \sum_{k=1}^{N} \delta_{ij_k}$
  - It helps reducing the complexity of training
  - Good for noise reduction in training stage
- Recurrent means occurring often or frequently.
  - in RNN, we perform the same task for inputs in a sequence.
- RNN captures the information from the previous input by maintaining an internal memory elements called states.
- It is useful for applications with temporal dependency = the current output depends not only on the current input but also on a memory element which takes into account past inputs. E.g., predicting the next word in a sentence: Sentiment analysis, Speech recognition, time-series analysis, NLP, and gesture recognition.

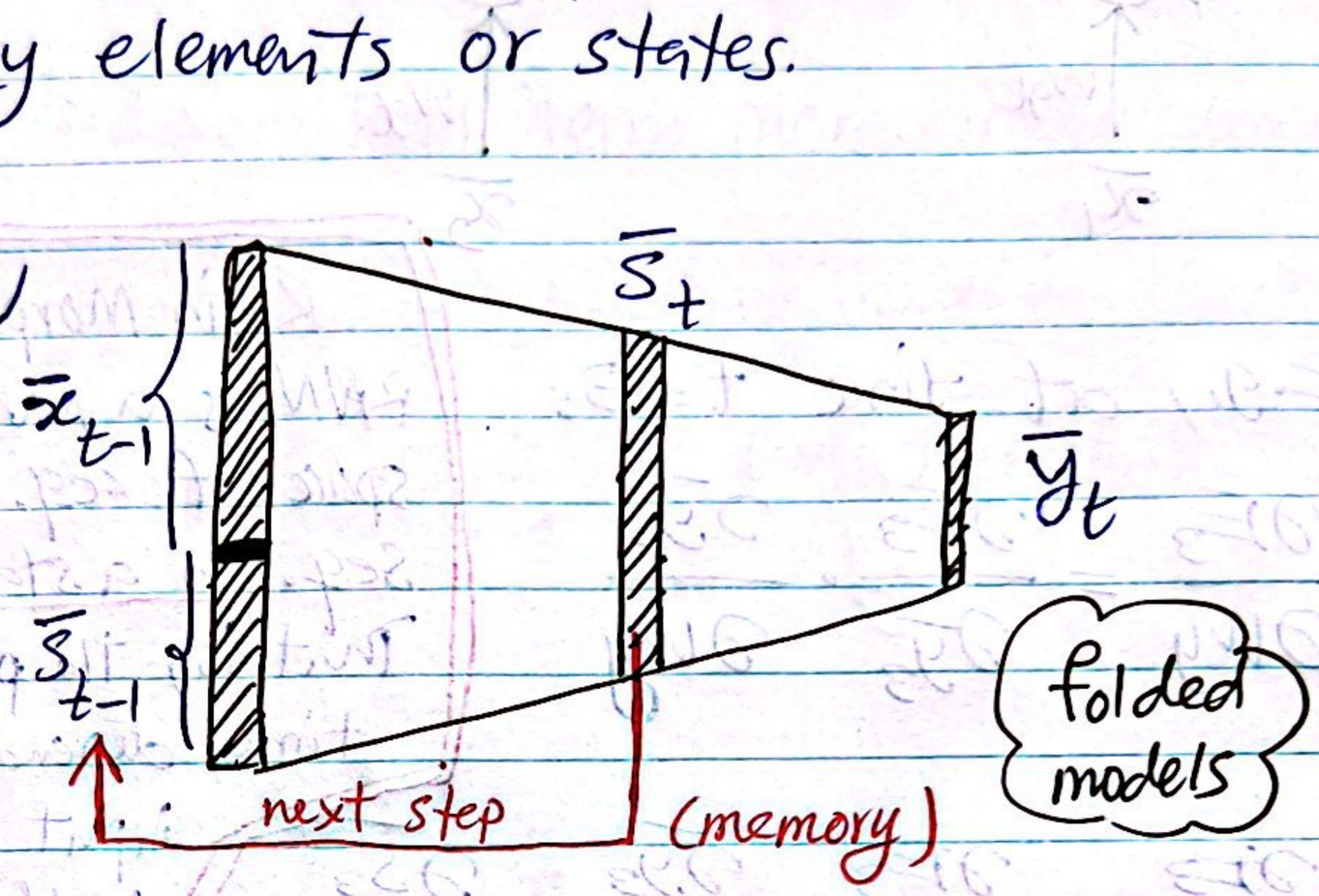


* The core differences between RNN and FFNN are:

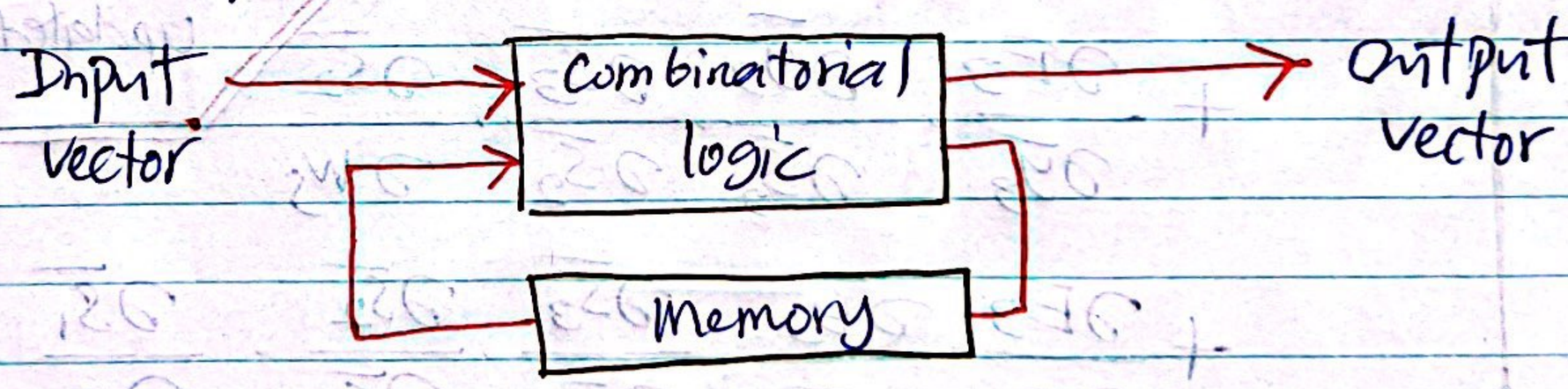
1/ the manner we define our outputs:

2/ memory elements or states.

* Simple RNN (Elman Network)



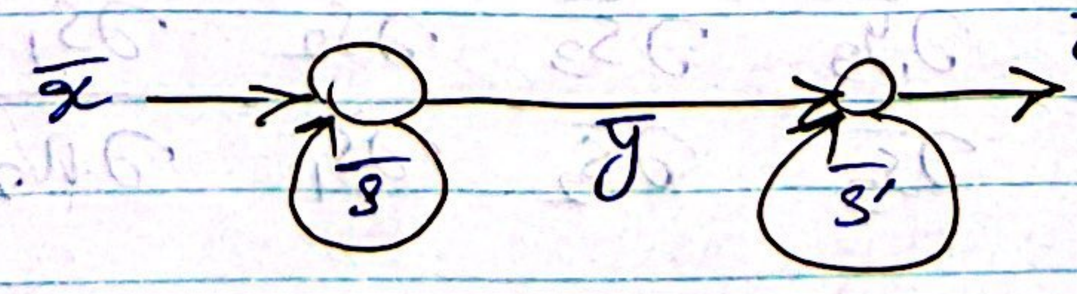
It is equivalent to a state machine:



$$\begin{cases} \bar{s}_t = \phi(\bar{x}_t \cdot W_x + \bar{s}_{t-1} \cdot W_s) \\ \bar{y}_t = \bar{s}_t \cdot W_y \quad \text{or} \quad \sigma(\bar{s}_t \cdot W_y) \end{cases}$$

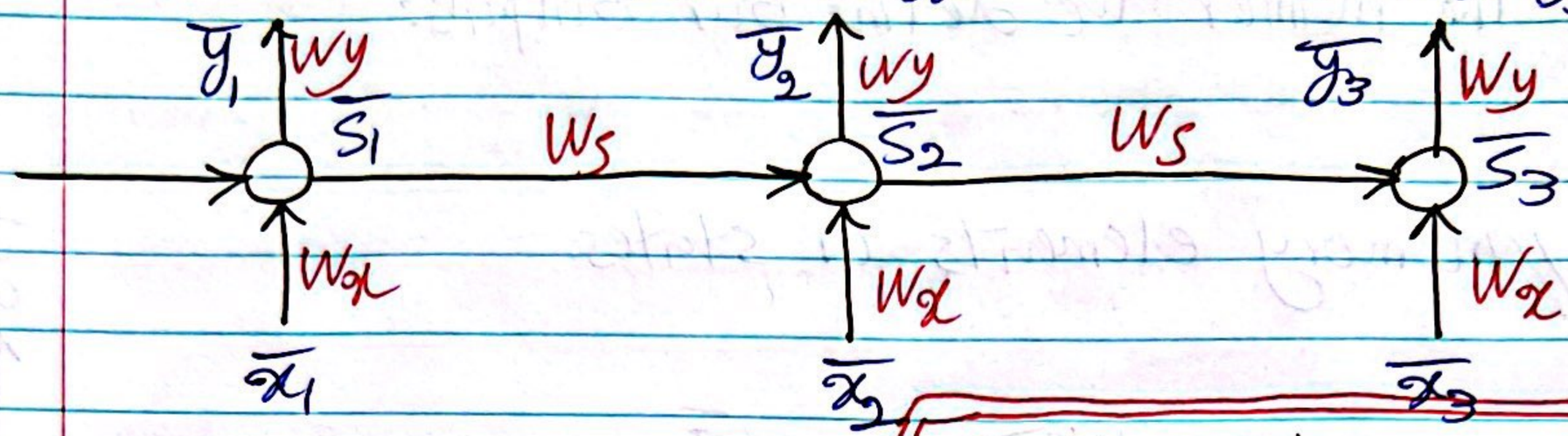
weights W remain the same for a given sequence of input!

* We can stack RNNs:



⊗ Back Propagation Through Time (BPTT)

$E_1 = (\bar{d}_1 - \bar{y}_1)^2$ $E_2 = (\bar{d}_2 - \bar{y}_2)^2$ $E_3 = (\bar{d}_3 - \bar{y}_3)^2$



E.g., at time $t=3$:

$$\frac{\partial E_3}{\partial W_y} = \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial W_y}$$

$$\frac{\partial E_3}{\partial W_s} = \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial W_s}$$

$$+ \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial \bar{s}_2} \cdot \frac{\partial \bar{s}_2}{\partial W_s}$$

$$+ \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial \bar{s}_2} \cdot \frac{\partial \bar{s}_2}{\partial \bar{s}_1} \cdot \frac{\partial \bar{s}_1}{\partial W_s}$$

$$\frac{\partial E_3}{\partial W_x} = \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial W_x}$$

$$+ \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial \bar{s}_2} \cdot \frac{\partial \bar{s}_2}{\partial W_x}$$

$$+ \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial \bar{s}_2} \cdot \frac{\partial \bar{s}_2}{\partial \bar{s}_1} \cdot \frac{\partial \bar{s}_1}{\partial W_x}$$

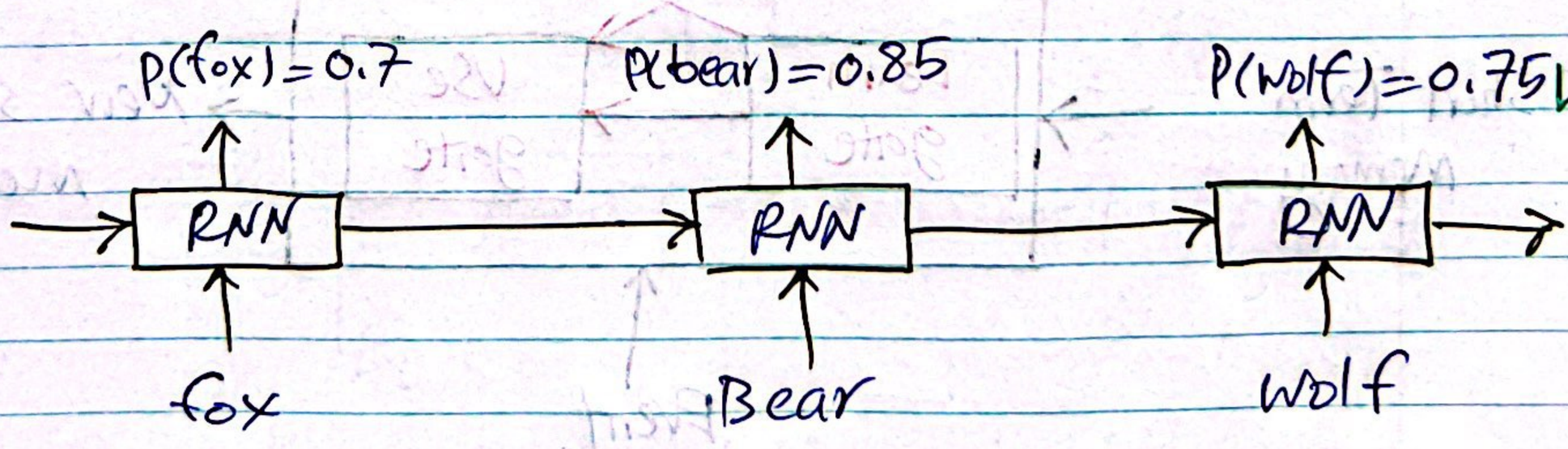
Kevin Morphy:
RNN is a NN that maps the input space of seq. to output space of seq. in a stateful way!
That is, the prediction of y_t over time, depends not only on the input x_t, but also on the hidden state of the sys, h_t, which gets updated over time!

* Problems with RNN: 1/ Vanishing Gradient —remedy→ LSTM
2/ Exploding Gradient —remedy→ Gradient clipping

LSTM: Long Short Term Memory Networks

These nets are useful when we need to switch between remembering recent things, and things from long time ago.

* Let's say, we have an image classifier and it has a picture of wolf → There is a high risk of missclassification of being dog! If we feed the net with a video which previous images are fox and bear, it is more probable that net classifies the image as wolf!

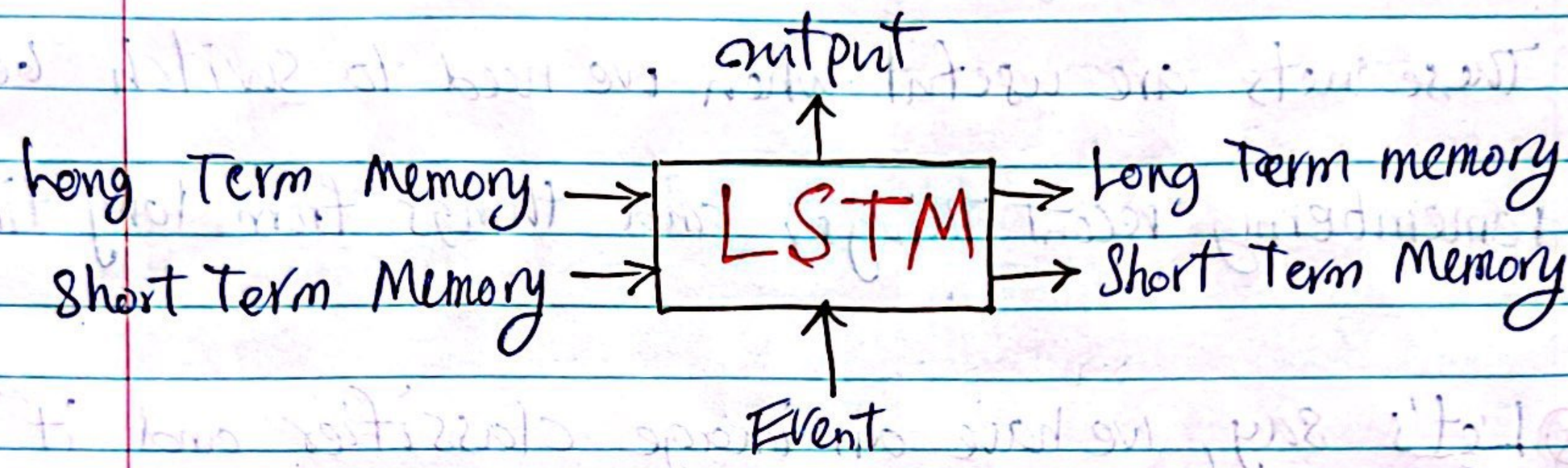


* This is because the network knows they are wild!

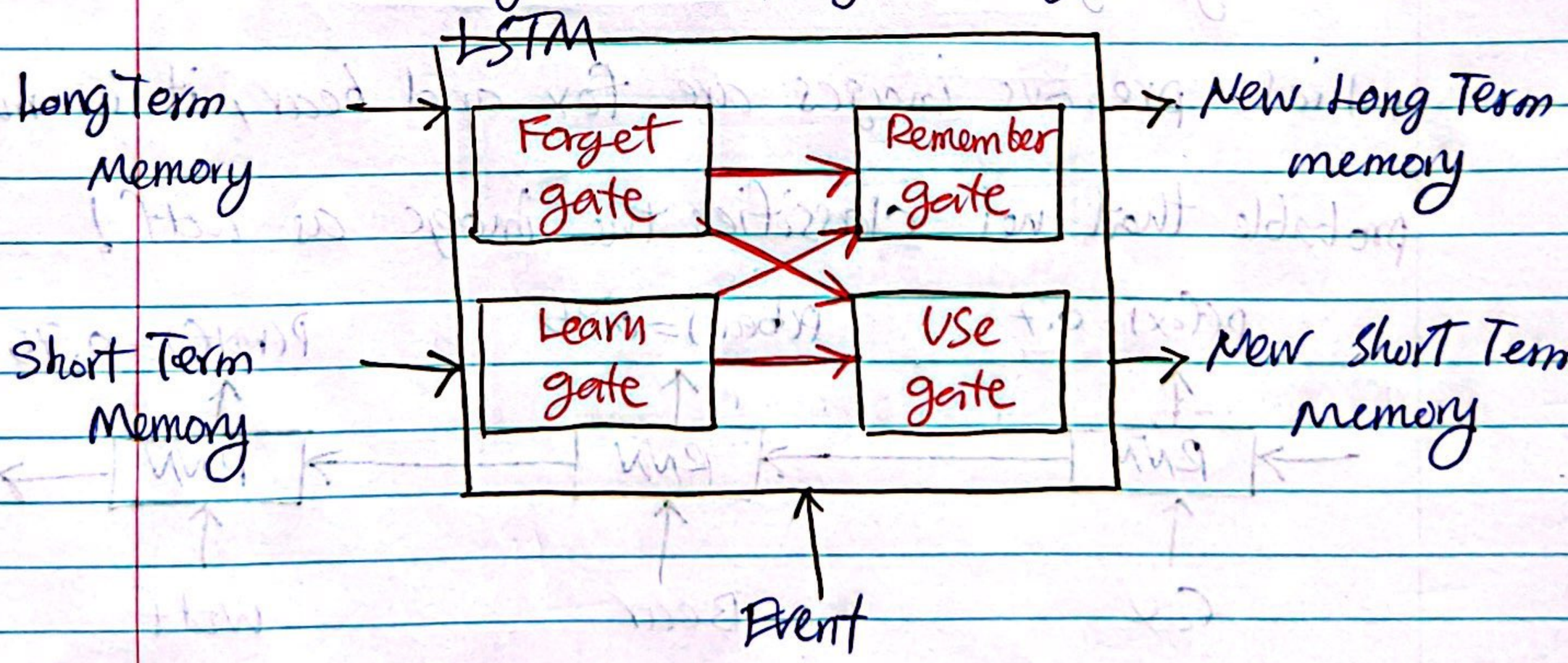
But, if we have these frames:

Bear → tree → squirrel → wolf

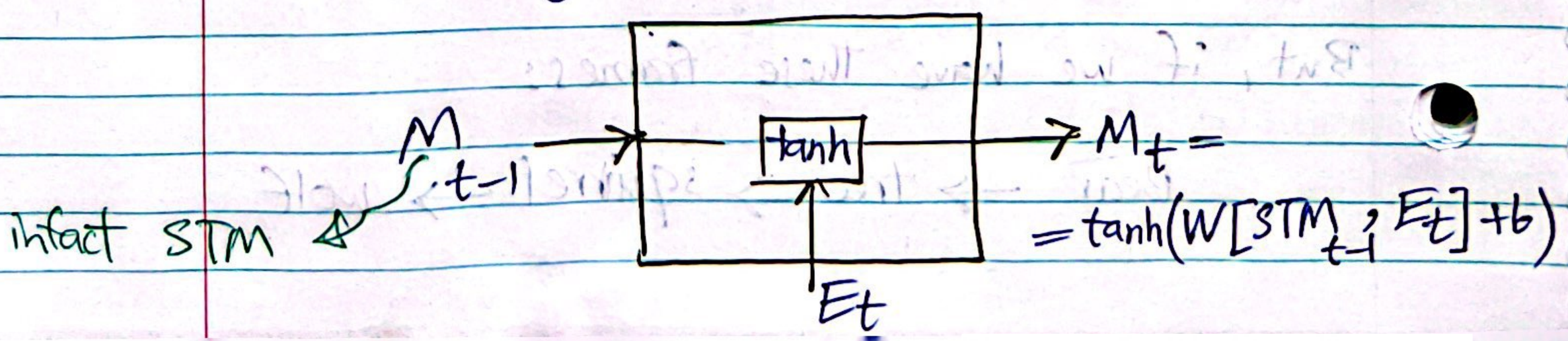
in this case, the net may forget about wild animals and misclassify wolf as dog! This is the problem with RNN! Due to vanishing gradient. This is where LSTM helps us!



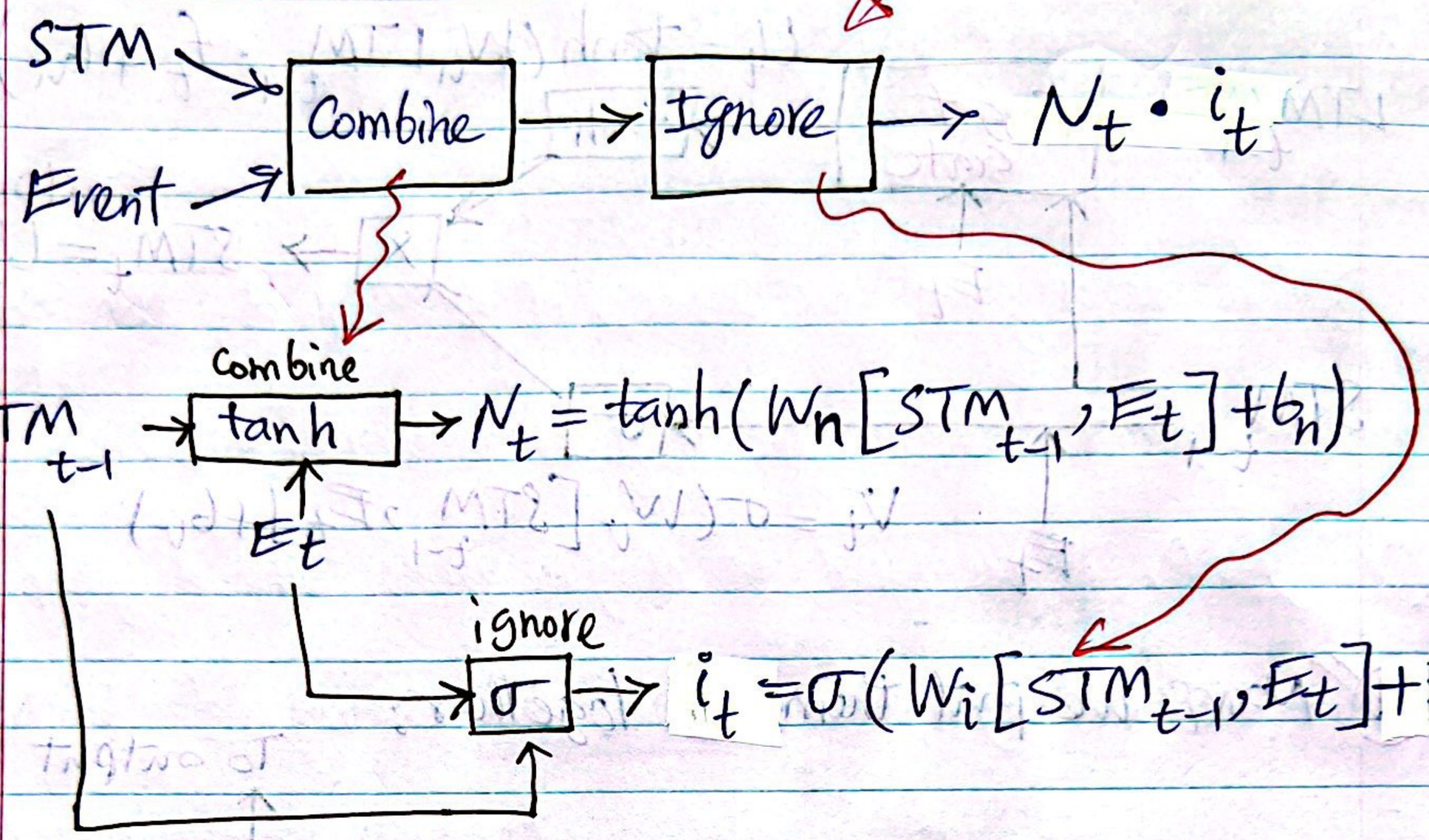
in LSTM, if we deem it is necessary, the network will remember thing from a long time ago!



RNN; revisiting:

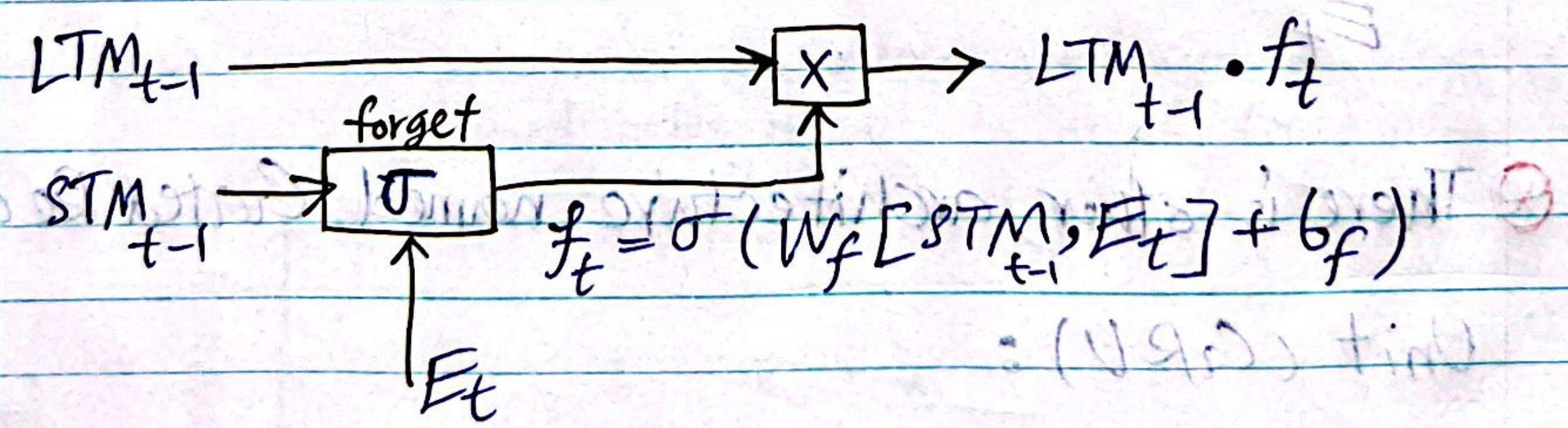


① We will start from the **learn gate**:

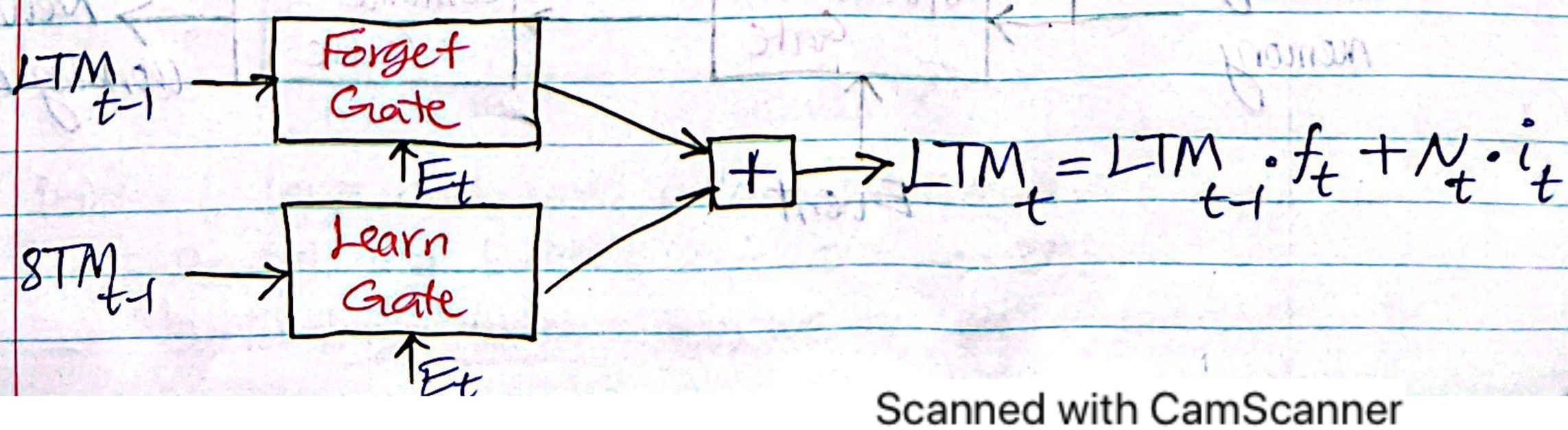


② The result of learn gate is element-wise product of N_t and i_t!

② Now, we will explain the **Forget Gate**:

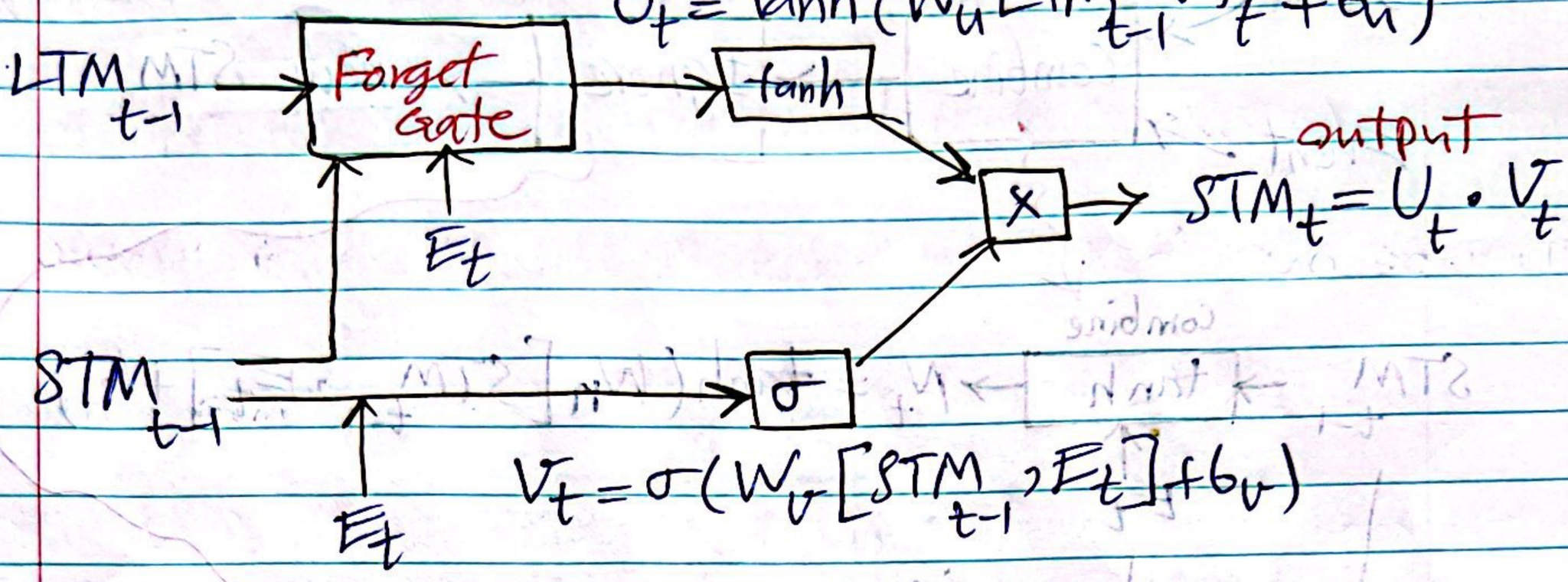


③ Remember Gate is relatively straight-forward:

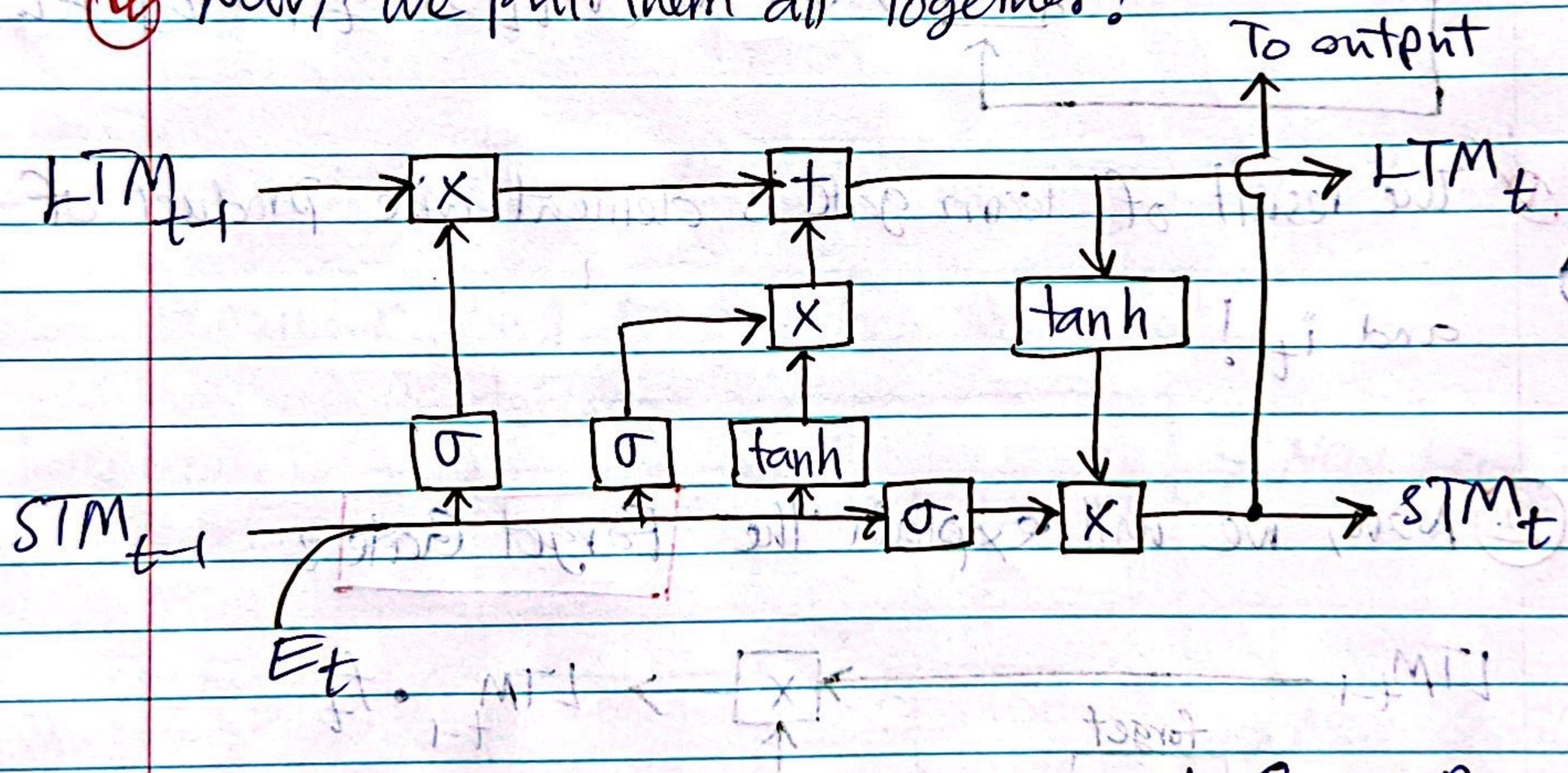


④ Use Gate or Output gate:

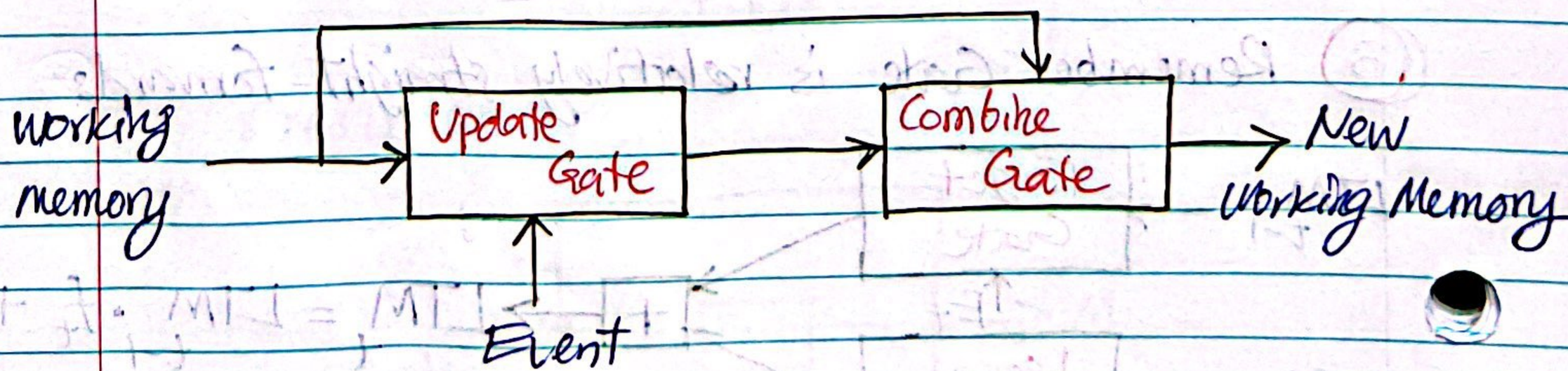
$$U_t = \tanh(W_U LTM_{t-1} \cdot f_t + b_U)$$



⑤ Now, we put them all together:



⑥ There is other architecture named Gated Recurrent Unit (GRU):



⊛ An important consideration in DL is hyperparameter tuning.

Hyperparameters ↗ 1/ optimizer hyperparameter
- learnig rate
- minibatch size
- # of training epochs

↘ 2/ Model hyperparmeters
- # of layers
- # of hidden units
- model's specific parametes

& depend on the task and data set

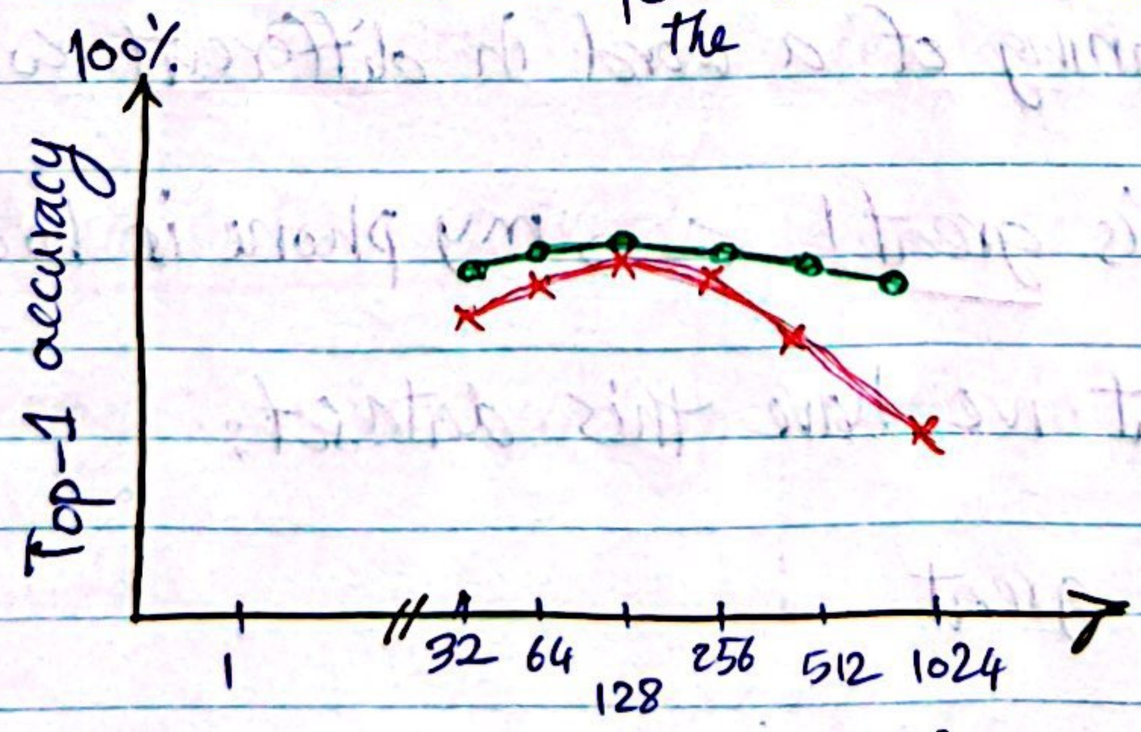
Minibatch size: 1, 2, 4, 8, 16, **32, 64, 128, 256**, 512, 1024, ...
(good)

- less memory demand
- more noisy learning.
- long learning process

- allows computational boost
- need large memories
  ⇒ more compntational resources

⊛ the compntational boost comes at a cost: decreasing model accuracy

⊛ This is because noise in smaller batches introduce noise into the learning and helps model to evade local minimas!



Points: 1/ large and small batch sizes are bad!
2/ optimum batch size is 32 → 256
⊛ 3/ for larger batch sizes, we have to increase the lr!