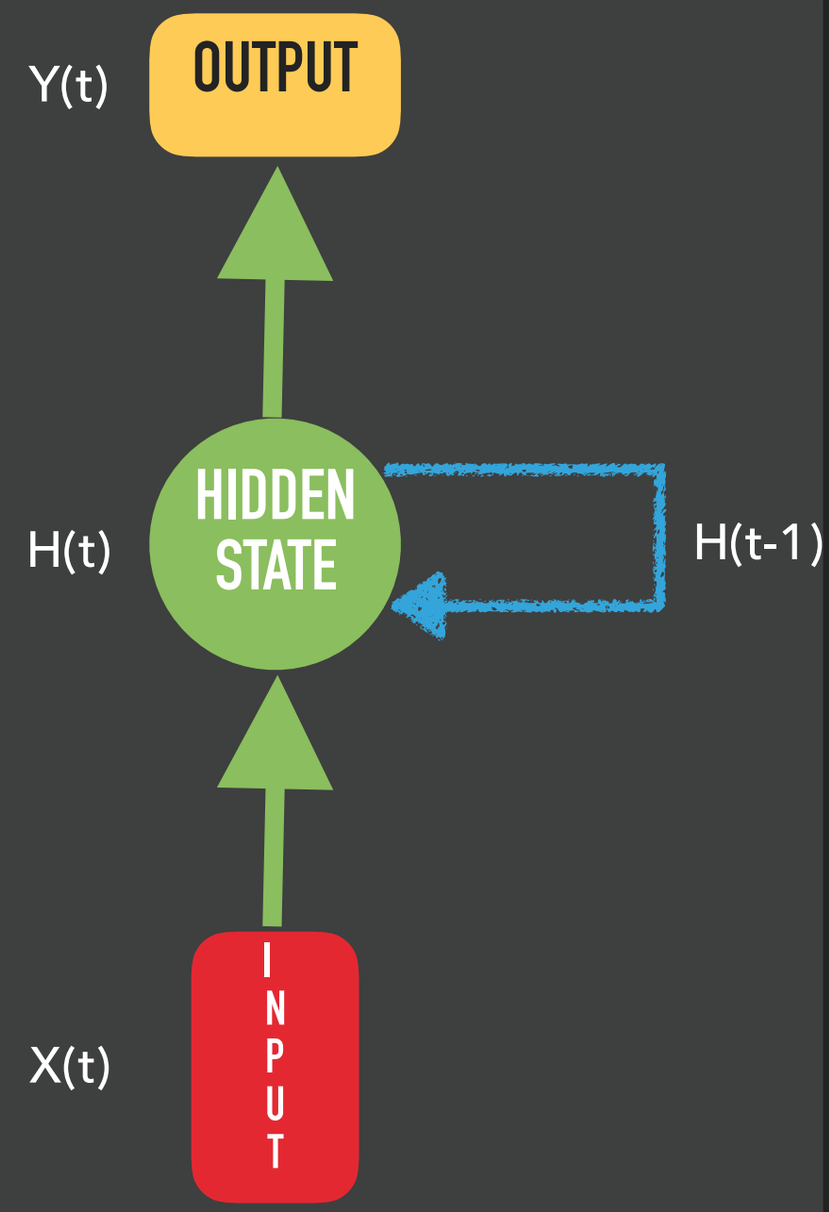
INTRODUCTION TO

RNN

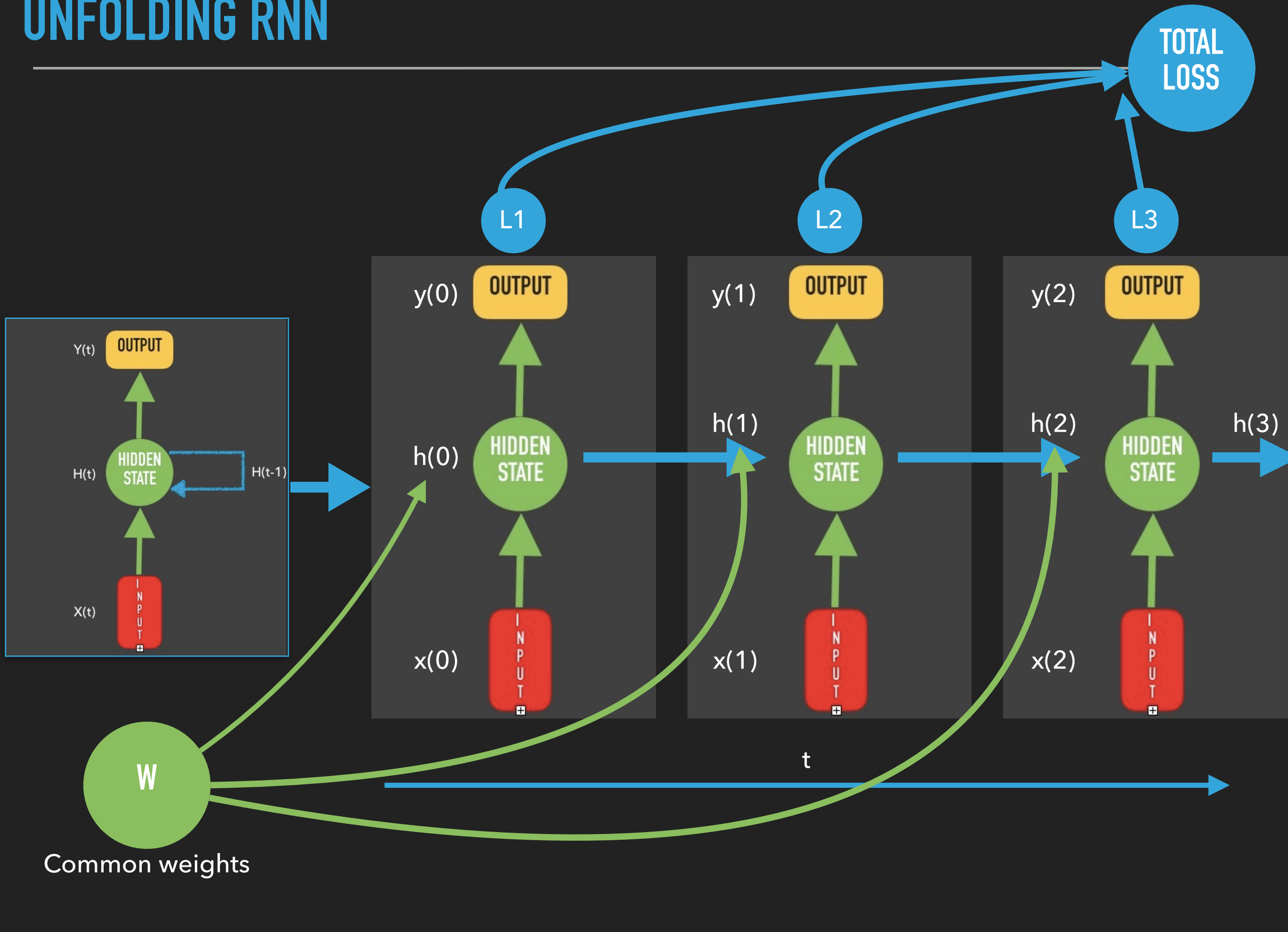
RECURRENT NEURAL NETWORK

- Make use of sequential information to predict next information.
- They are called ‘recurrent’ because they perform similar task for every element of the sequence ,with the output being dependent on the previous computations.
- You can think as RNN’s have “memory” which captures information what has been calculated so far.



Vanilla NN
Vanilla RNN

UNFOLDING RNN



FORMULA FOR H(T) (HIDDEN STATE)

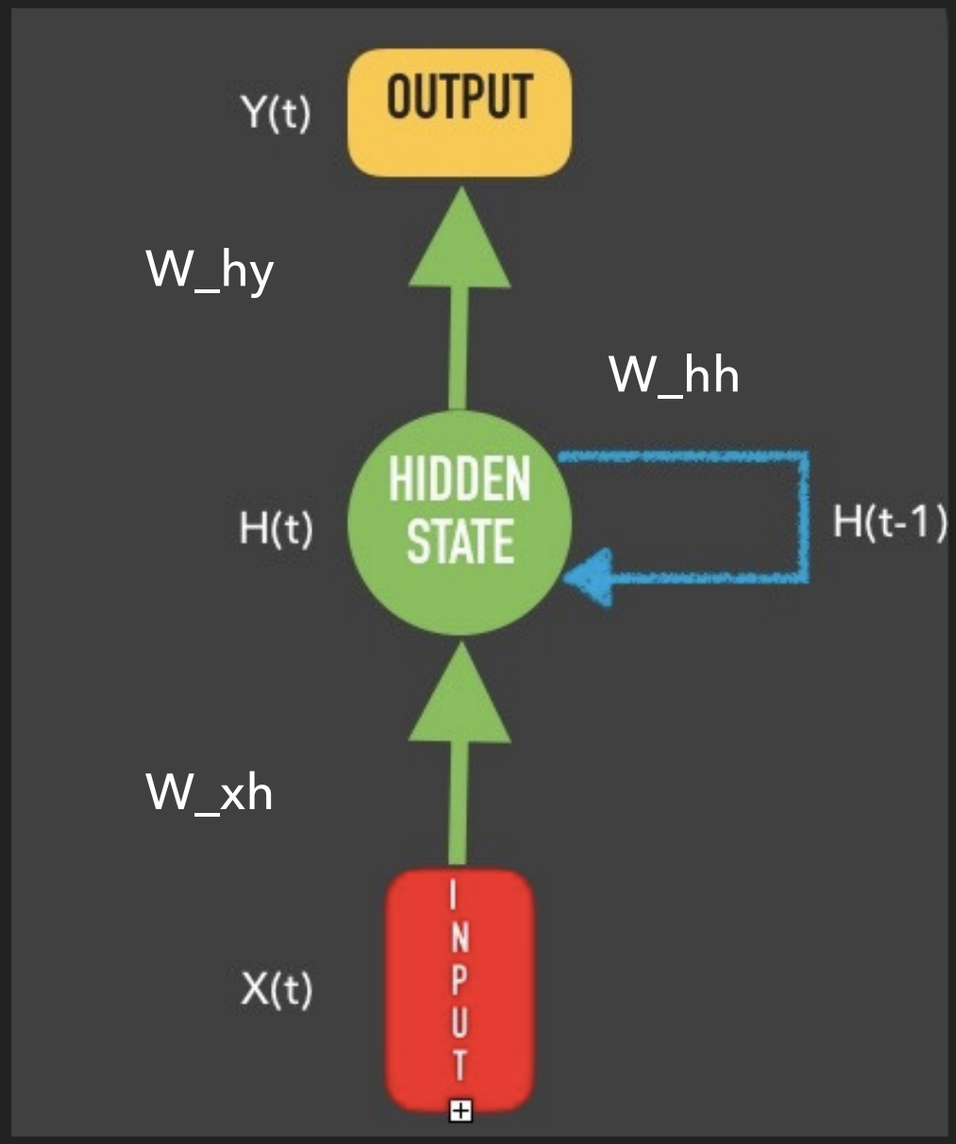
$$H(t) = \tanh(W_{hh} * H(t-1) + W_{xh} * X(t) + b_h)$$

FORMULA FOR Y(T) (OUTPUT)

$$Y(t) = W_{hy} * H(t) + b_y$$

LOSS FORMULA

$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \qquad L_i = -\log(p_{y_i})$$



BACKPROPAGATION THROUGH TIME IN RNN

- Calculate gradient with respect to each weight (W_xh ,W_hh and W_hy)
- Since all output units contribute to the error of each hidden unit we sum up all the gradients calculated at each time step in the sequence and use it to update the parameters.

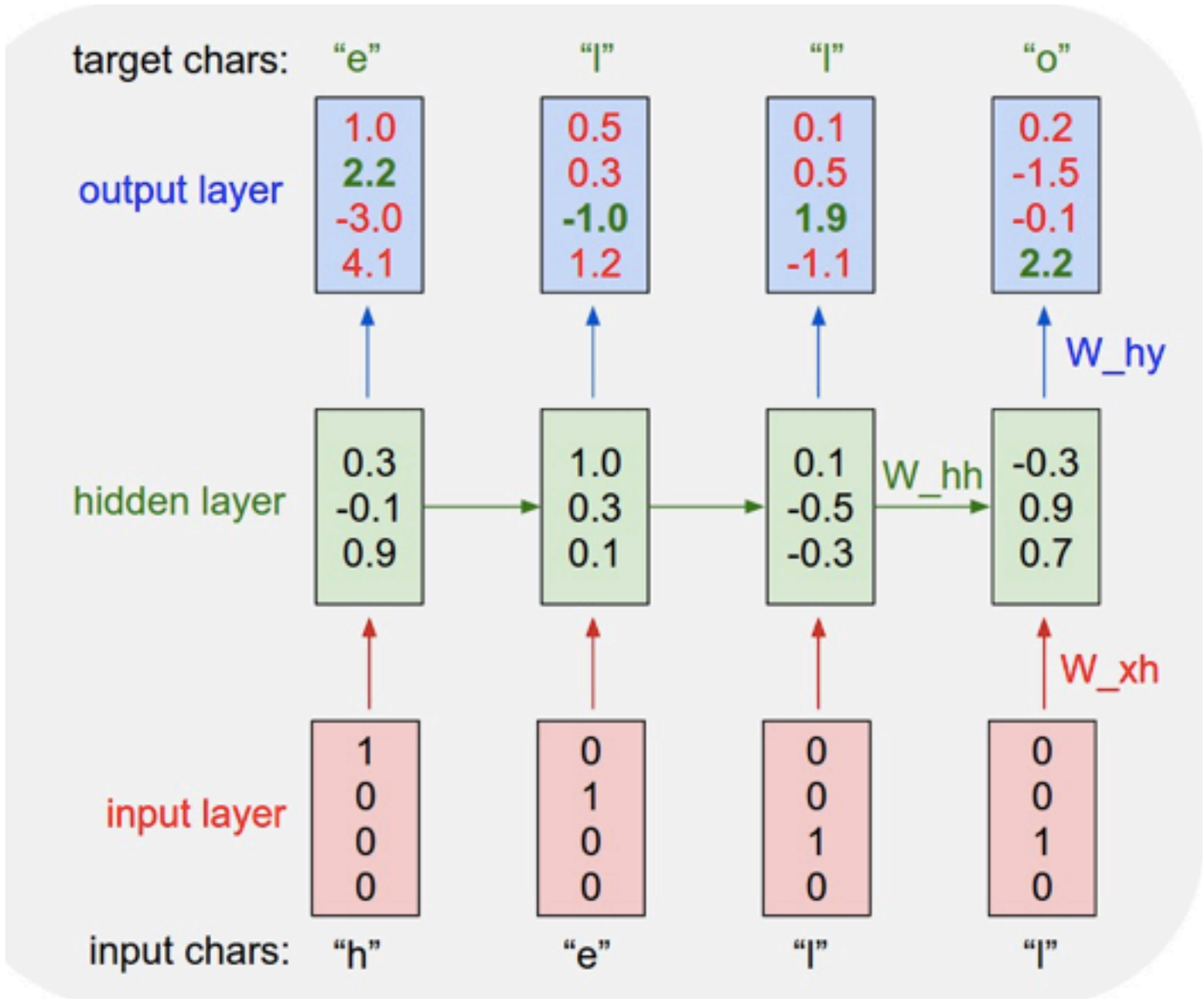
$$\frac{\partial J_t}{\partial W_{hy}} = \sum_t \frac{\partial J}{\partial z_t} * \underbrace{\frac{\partial z_t}{\partial W_{hy}}}_{h_t}$$

$$\frac{\partial J_t}{\partial W_{hh}} = \sum_t \frac{\partial J}{\partial h_t} * \underbrace{\frac{\partial h_t}{\partial W_{hh}}}_{(1-h_t^2)*h_{t-1}}$$

$$\frac{\partial J_t}{\partial W_{xh}} = \sum_t \frac{\partial J}{\partial h_t} * \underbrace{\frac{\partial h_t}{\partial W_{xh}}}_{(1-h_t^2)*x_t}$$

EXAMPLE- CHARACTER-LEVEL LANGUAGE MODELS

Text = “hello” Set of characters = [h , e , l , o]



Actual

GENERATED CODE

```
#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)     (func)

#define SWAP_ALLOCATE(nr)     (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0));   \
  if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
          pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
  PUT_PARAM_RAID(2, sel) = get_state_state();
  set_pid_sum((unsigned long)state, current_state_str(),
           (unsigned long)-1->lr_full; low;
}
```

GENERATED XML

```
<page>
  <title>Antichrist</title>
  <id>865</id>
  <revision>
    <id>15900676</id>
    <timestamp>2002-08-03T18:14:12Z</timestamp>
    <contributor>
      <username>Paris</username>
      <id>23</id>
    </contributor>
    <minor />
    <comment>Automated conversion</comment>
    <text xml:space="preserve">#REDIRECT [[Christianity]]</text>
  </revision>
</page>
```

GENERATED TEXT IN SHAKESPEARE'S STYLE

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

FORMS OF RNN AND THEIR USE CASES

(1) (2) (3) (4) (5)

(1) Vanilla mode of processing from fixed-sized input to fixed-sized output (e.g. image classification).

(2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words).

(3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).

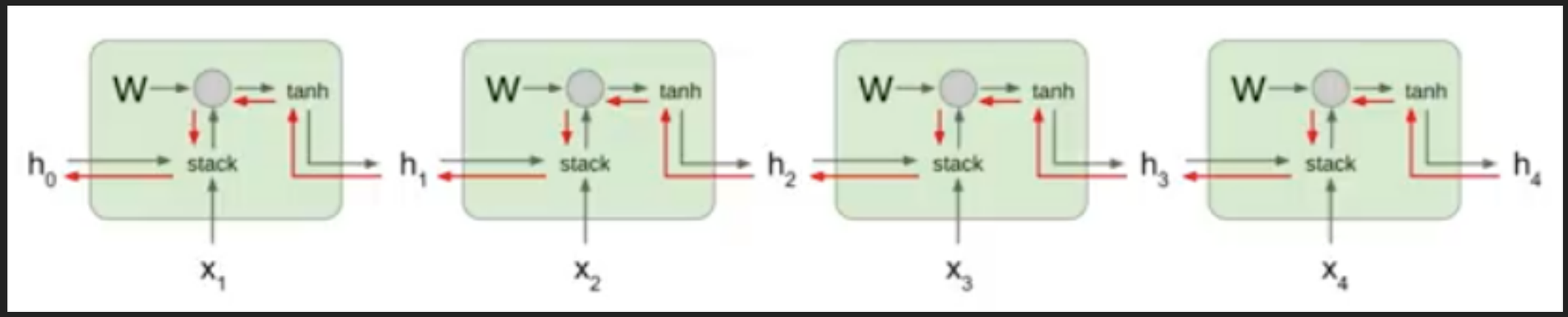
(4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).

(5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).

example-> http://grail.cs.washington.edu/projects/AudioToObama/

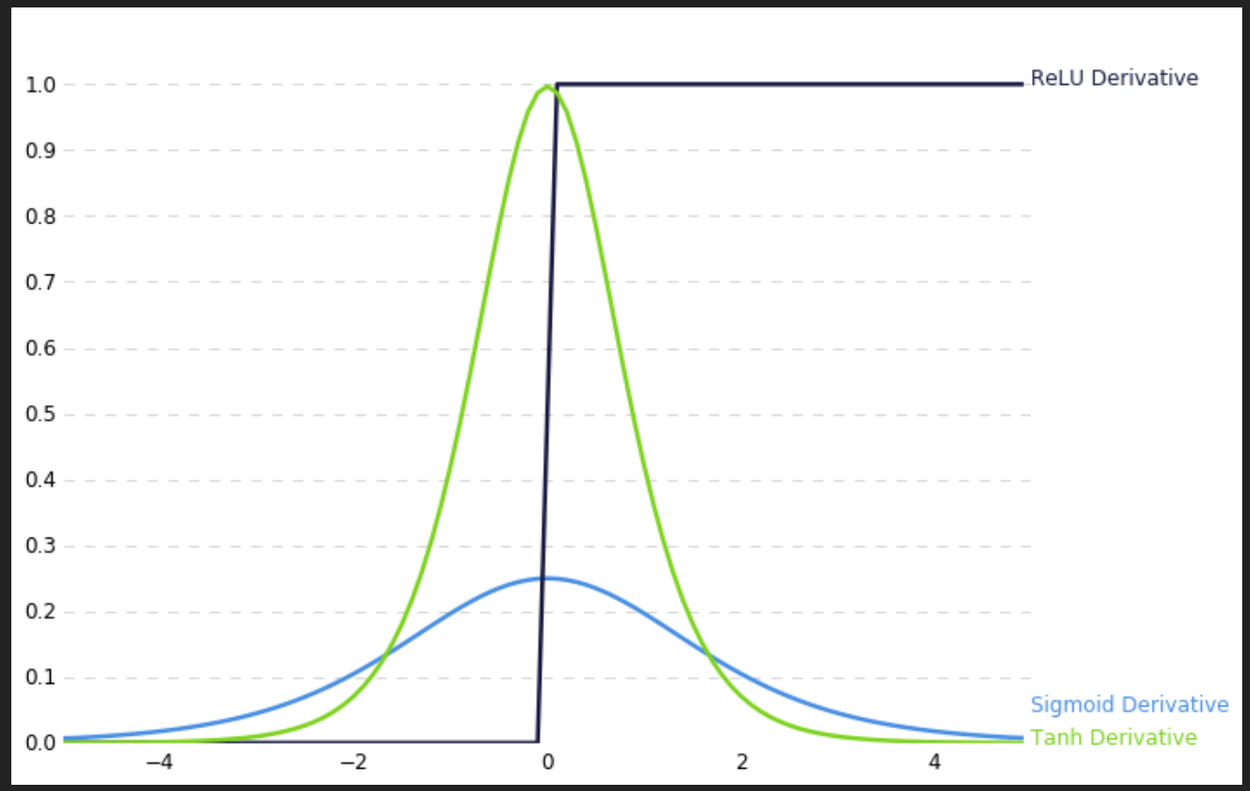
PROBLEM WITH RNN

- RNN is not good at remembering long range dependencies because of “Vanishing Gradient Problem”.
- In Vanishing Gradient Problem the Gradient exponentially decays as its backpropagated.
- Gradient exponentially decays because the derivative of “tanh” ranges between 0 to 1 and if multiplying n of these small numbers while calculating gradient through chain rule then it will exponentially decrease.



SOLUTION

- Change activation function (use ReLU)



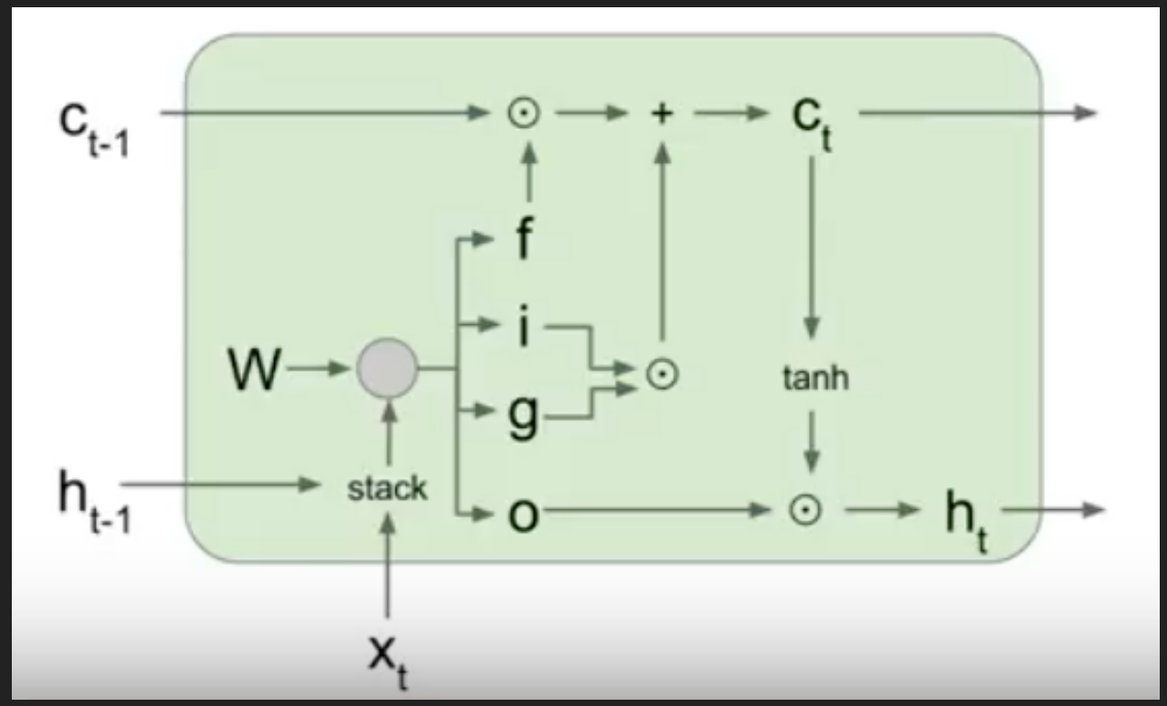
- Use LSTM

LONG SHORT-TERM MEMORY(LSTM)

- LSTMs (Long Short-Term Memory Networks) are a special subset of RNNs that are able to deal with remembering information for much longer periods of time.

- An LSTM does the same as RNN, except it also takes in its old cell state and outputs its new cell state.So cell state is the long term memory.

- So we have now to states
  C(t) -> Cell state
  h(t) -> output



LSTM

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

W=[W_hh W_xh]

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

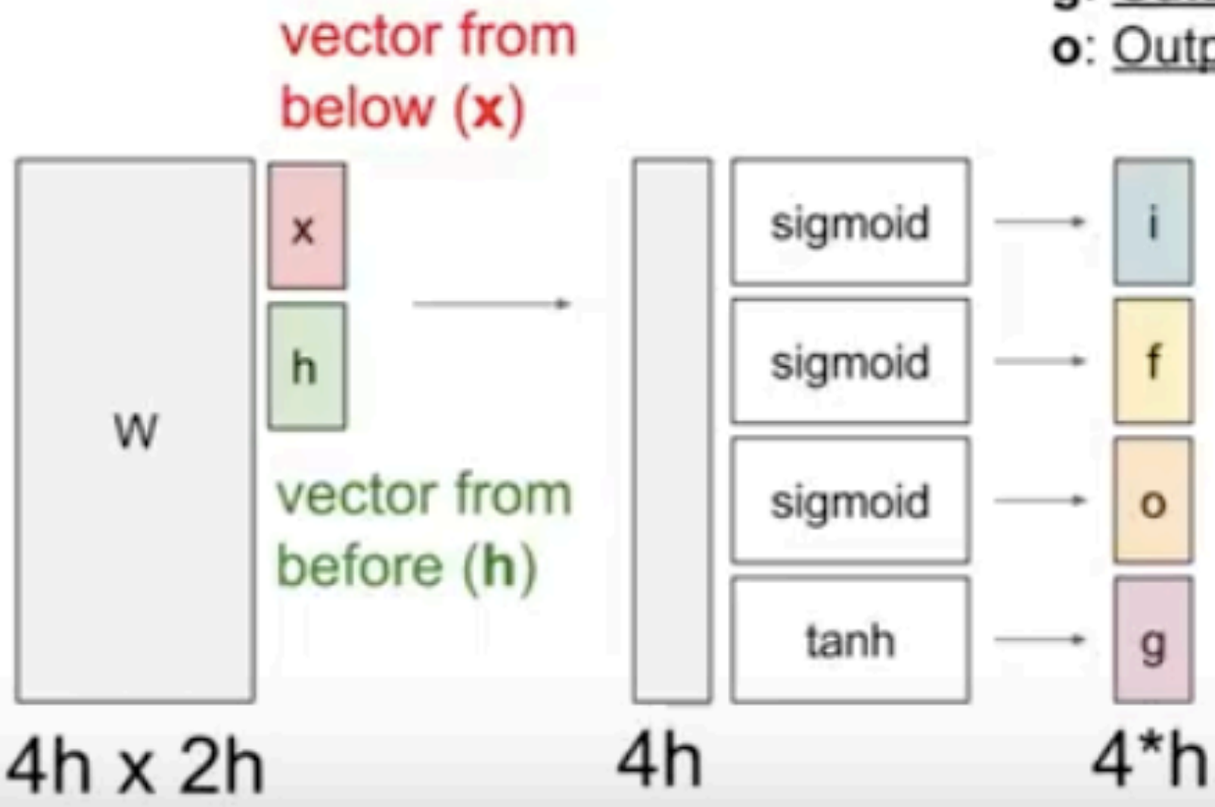
$$h_t = o \odot \tanh(c_t)$$

- In LSTM we first calculate cell state ,which is dependent on previous cell state .
- Using current cell state we calculate h(t)

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

f: Forget gate, Whether to erase cell
i: Input gate, whether to write to cell
g: Gate gate (?), How much to write to cell
o: Output gate, How much to reveal cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

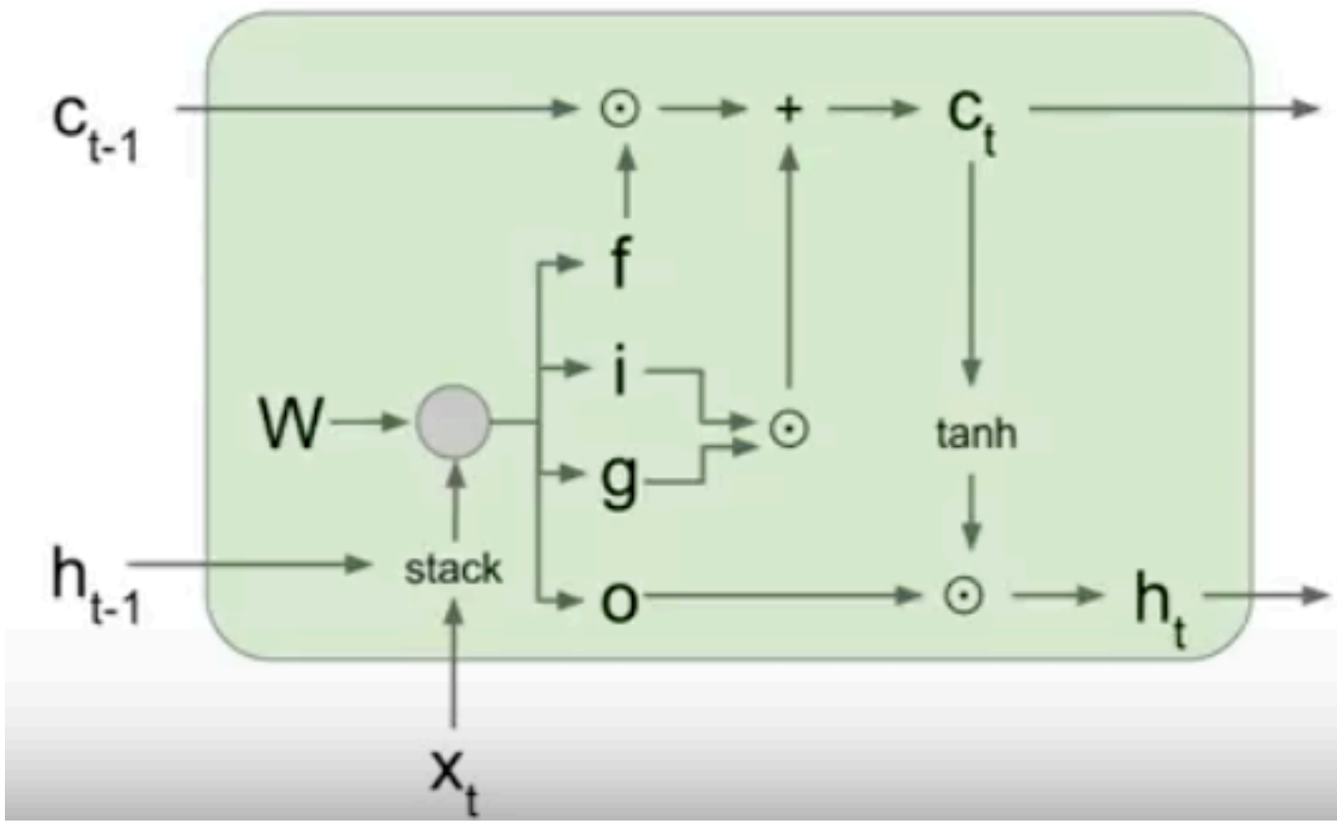
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

FORWARD PROPAGATION

- We decide what from the previous cell state is worth remembering, and tell the cell state to **forget the stuff we decide is irrelevant**.
- We **selectively update** the cell state based on the new input we've just seen.
- We **selectively decide what part of the cell state we want to output** as the new hidden state.

$$\boldsymbol{g}^{(t)} = \phi(W^{\mathrm{gx}}\boldsymbol{x}^{(t)} + W^{\mathrm{gh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_g)$$



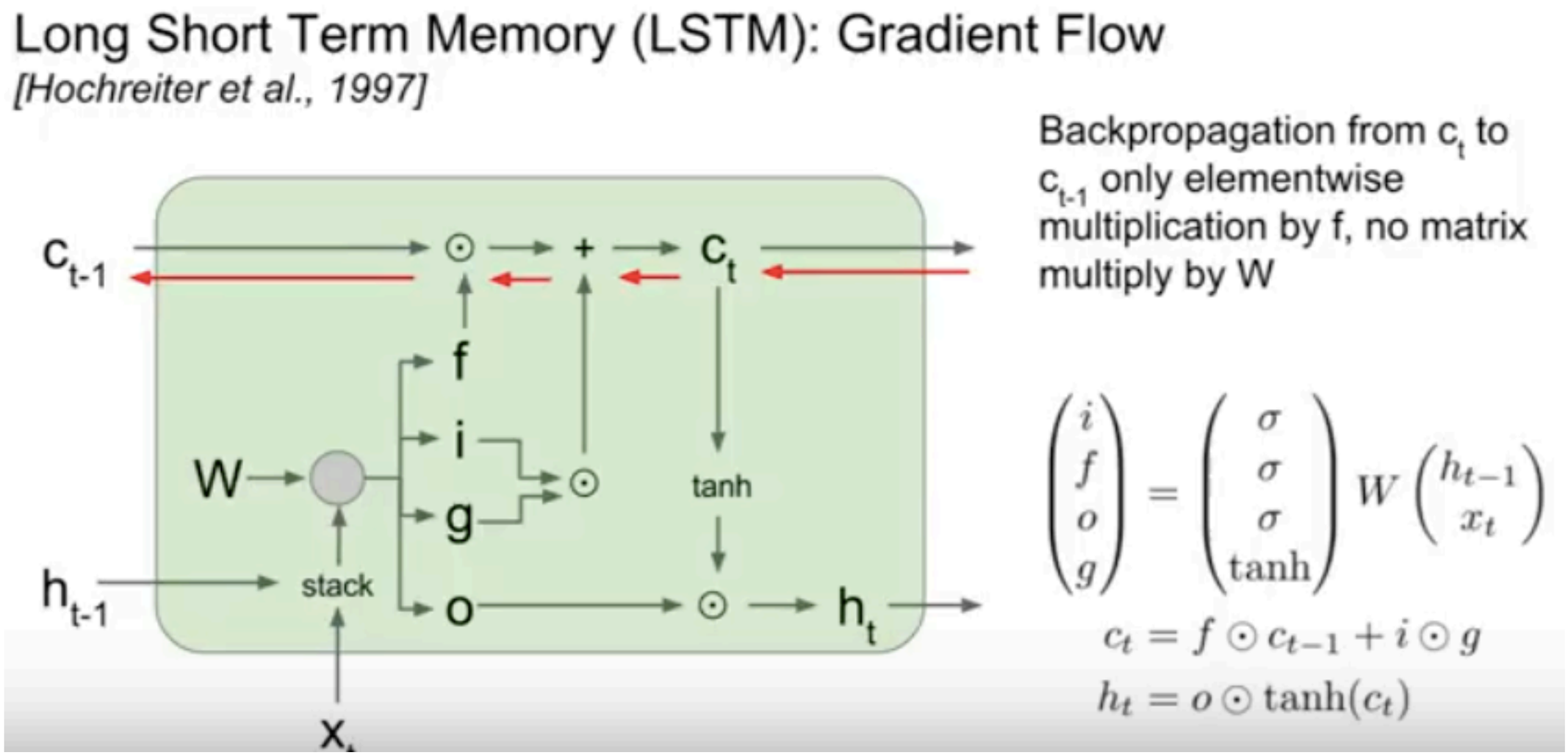
$$\boldsymbol{i}^{(t)} = \sigma(W^{\mathrm{ix}}\boldsymbol{x}^{(t)} + W^{\mathrm{ih}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i)$$
$$\boldsymbol{f}^{(t)} = \sigma(W^{\mathrm{fx}}\boldsymbol{x}^{(t)} + W^{\mathrm{fh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f)$$
$$\boldsymbol{o}^{(t)} = \sigma(W^{\mathrm{ox}}\boldsymbol{x}^{(t)} + W^{\mathrm{oh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o)$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

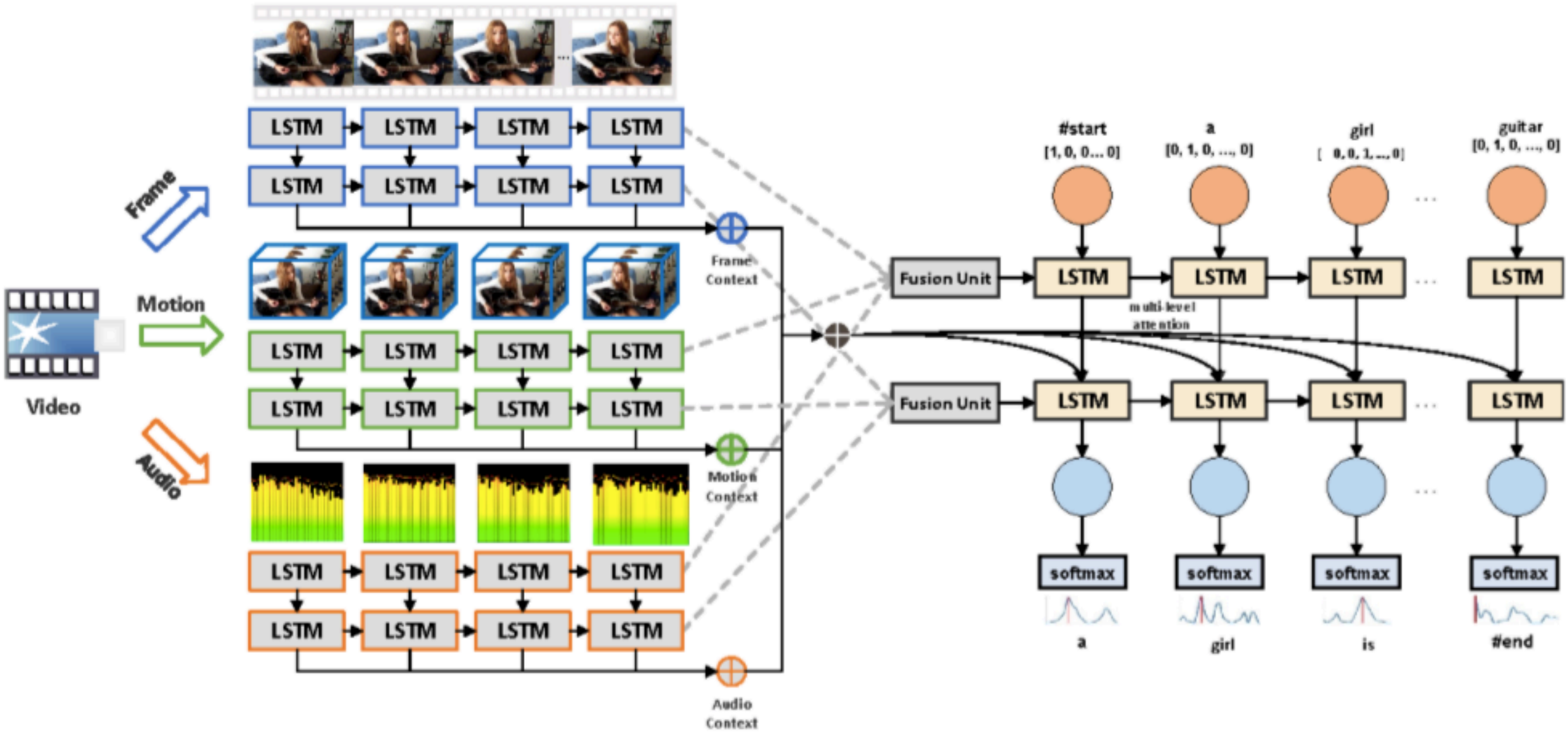
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

BACKPROPAGATION

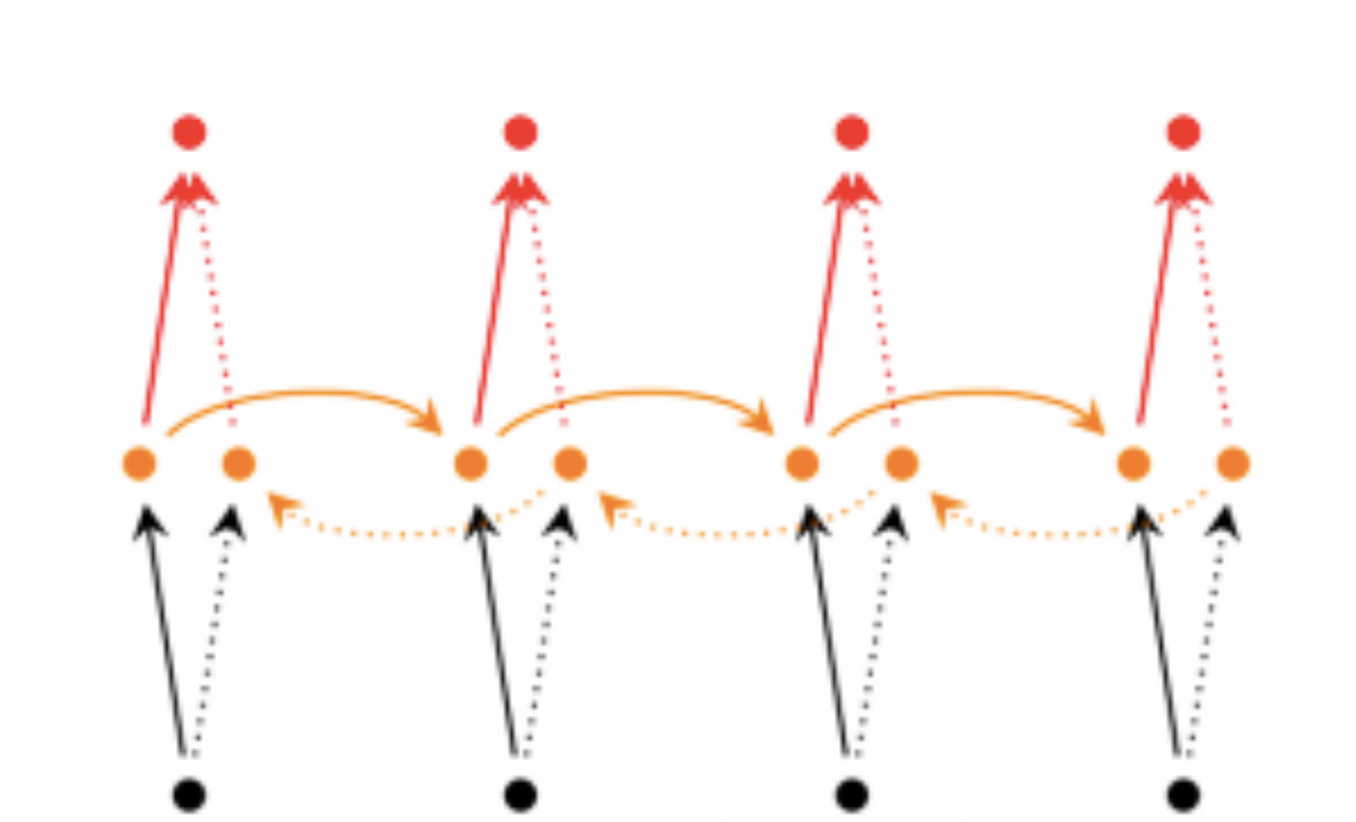
- The constant error carousel enables the gradient to propagate back across many time steps, neither exploding nor vanishing. In a sense, the gates are learning when to let error in, and when to let it out.
- Instead of maintaining cumulative sum of gradients like RNN ,LSTM maintains local set of gradients.



EXAMPLE


Frame
Motion
Audio
Video
LSTM
Frame Context
Motion Context
Audio Context
Fusion Unit
multi-level attention
#start
[1, 0, 0... 0]
a
[0, 1, 0, ..., 0]
girl
[0, 0, 1, ..., 0]
guitar
[0, 1, 0, ..., 0]
softmax
a
girl
is
#end

BIDIRECTIONAL RNNS



THANK YOU