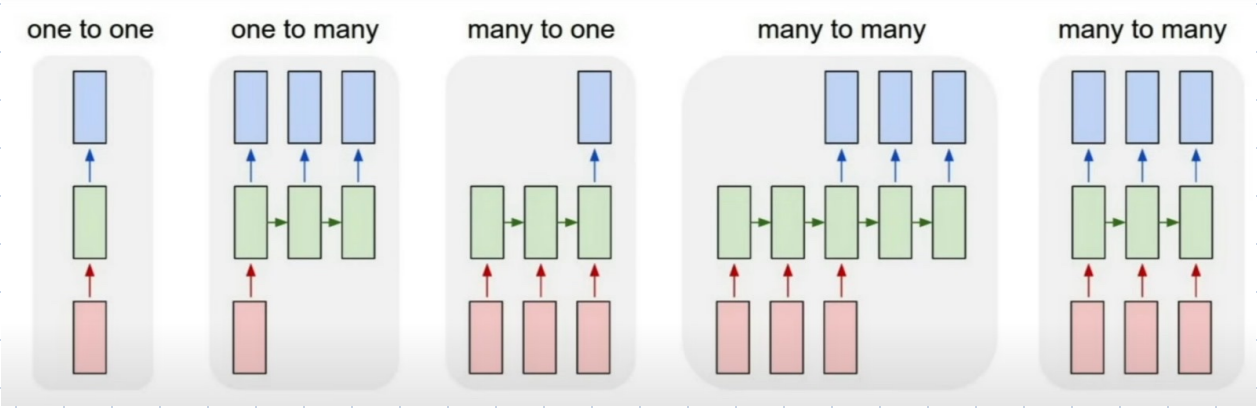


↳ image → sequence of words

↳ sequence of words → sentiment ↳ +/-

↳ seq of words → Translated seq of words

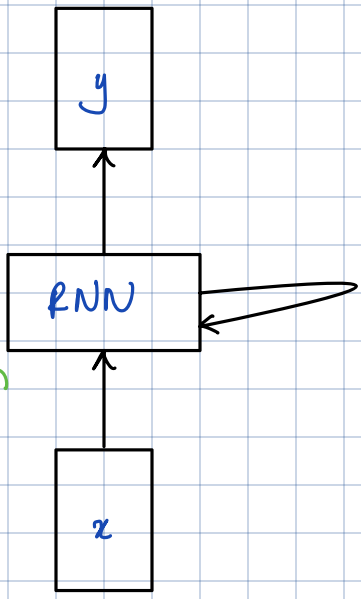
↳ Video classification on frame level

Process sequence:

↳ Take a fixed input
↳ Network looks at glimpses of image
↳ After a series of glimpes, it makes a decision

RNN:

↳ Read an input
↳ update it's internal hidden state
↳ Produce an output



old state ↑

{ functional form of recurrence relation

↓ new state

↓ some func with params W

↳ input vector at current time step

Vanilla RNN:

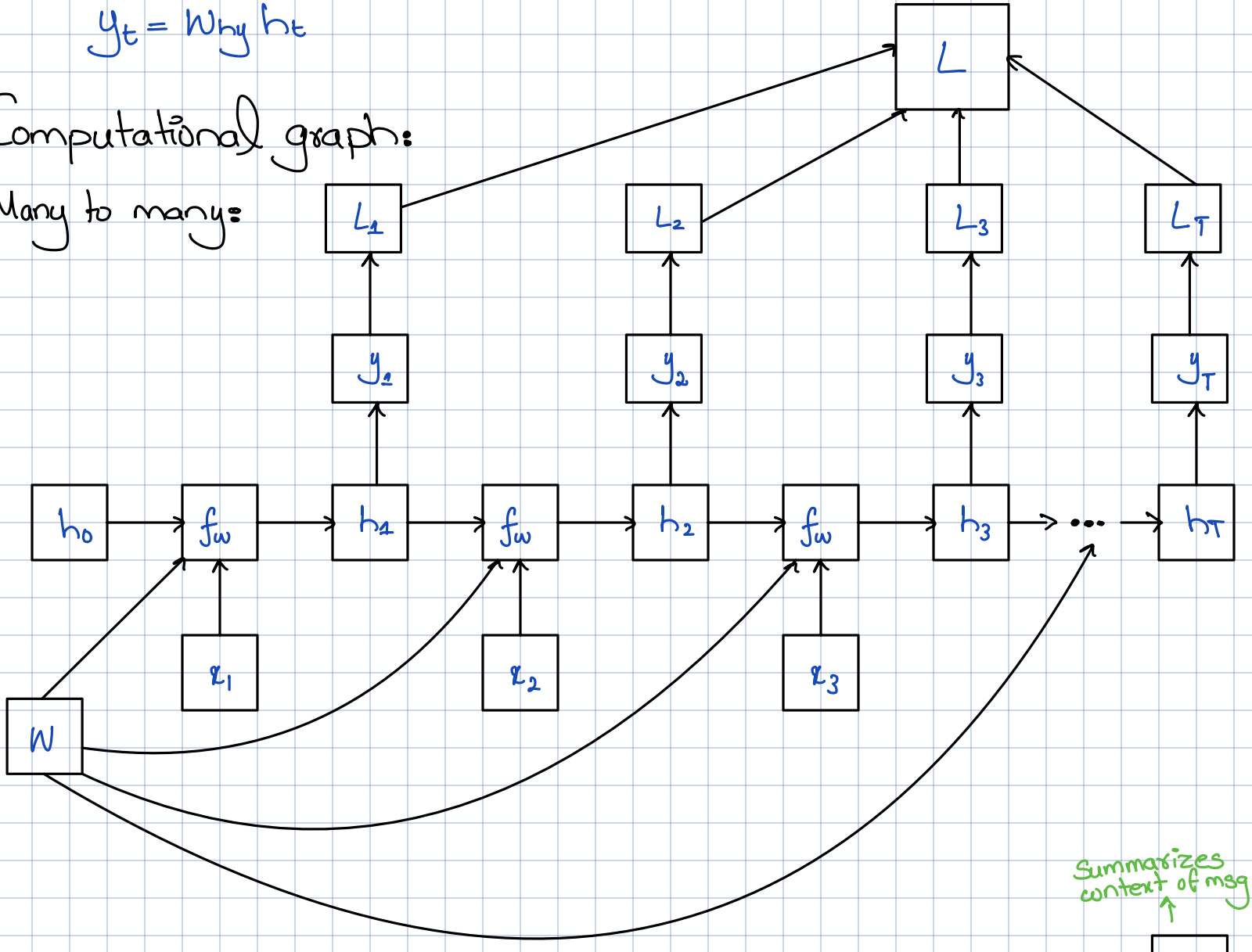
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$ { Non linearity

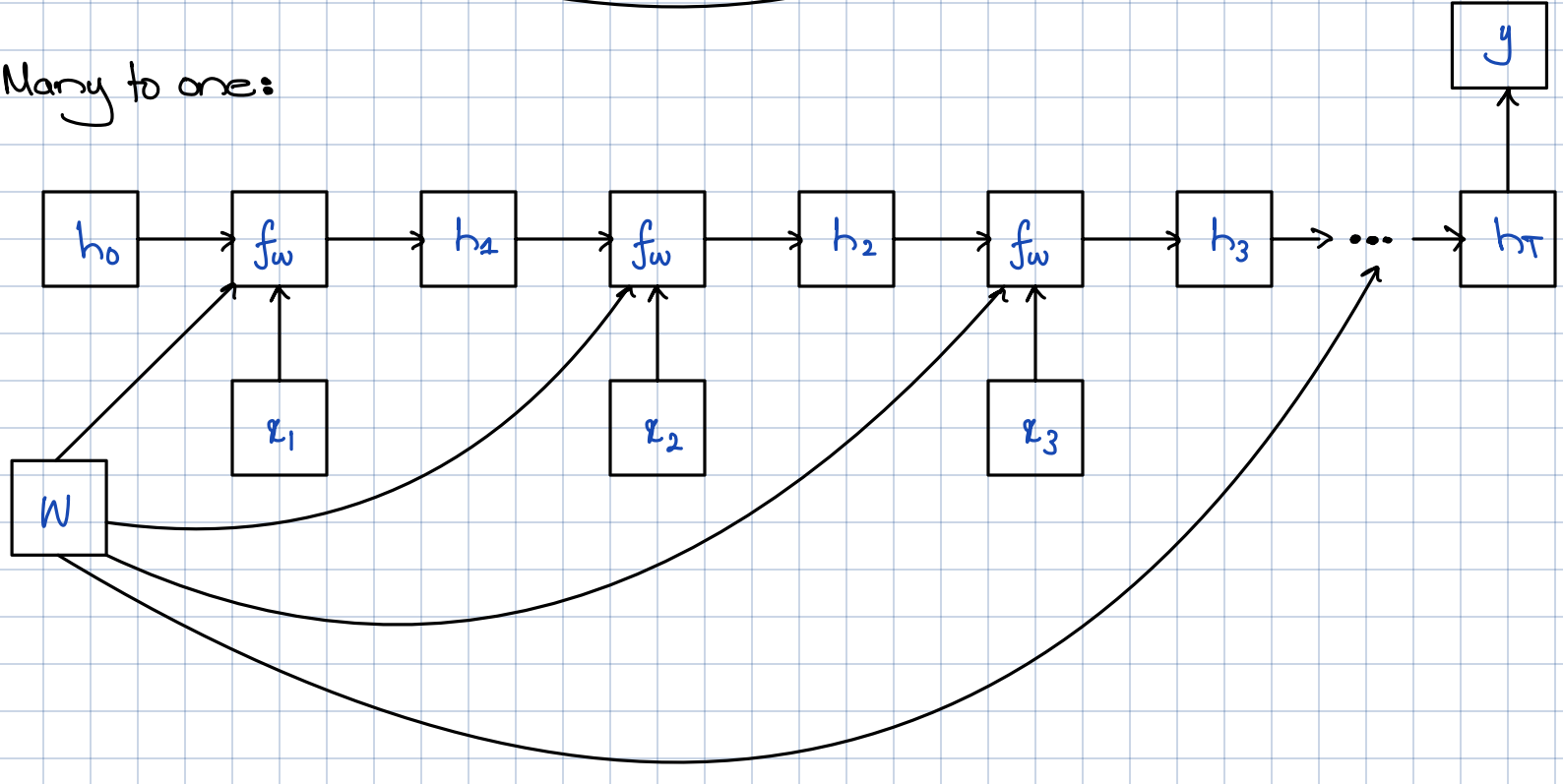
$y_t = W_{hy} h_t$

Computational graph:

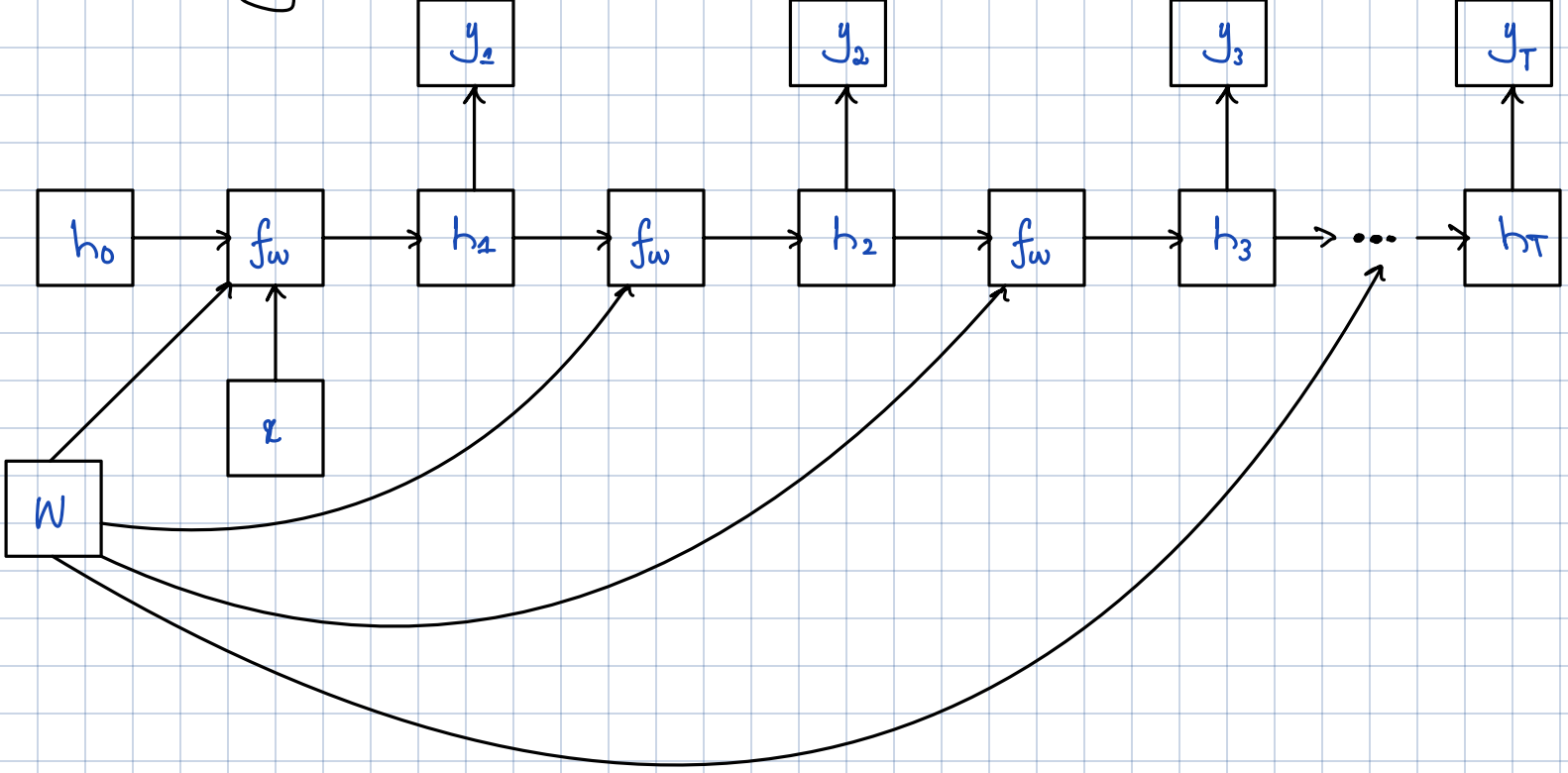
Many to many:



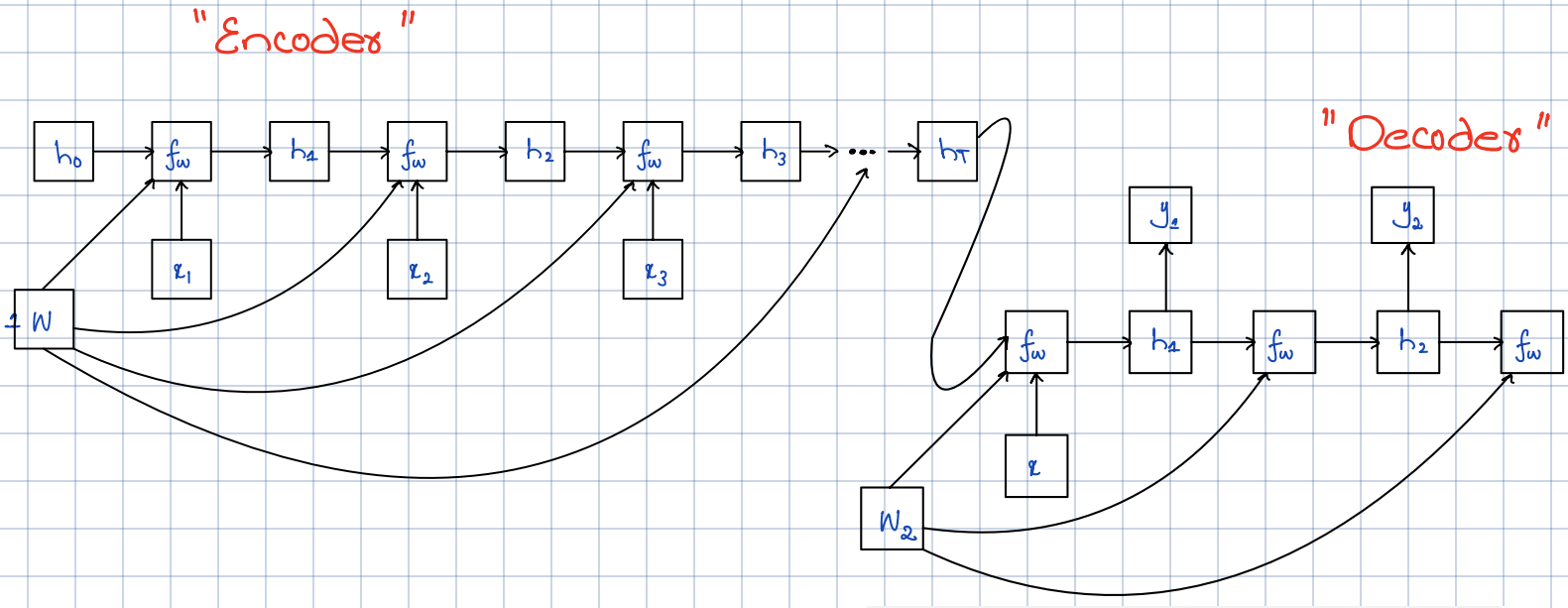
Many to one:



One to many:



Sequence to Sequence (Many to one + One to many)

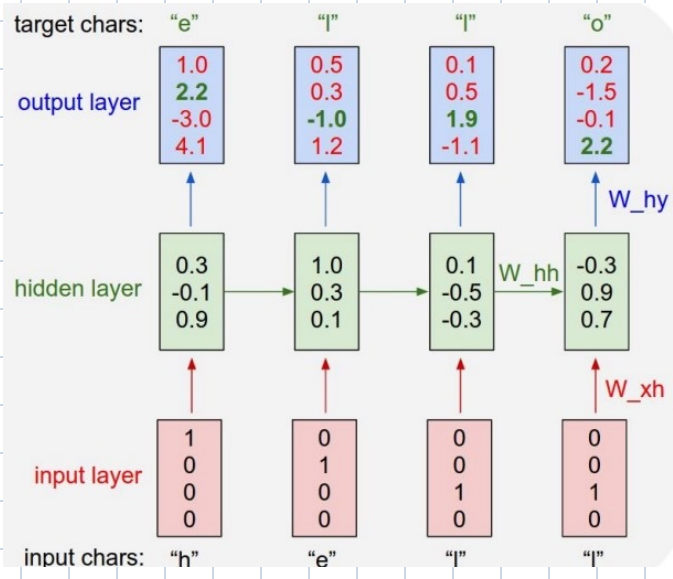


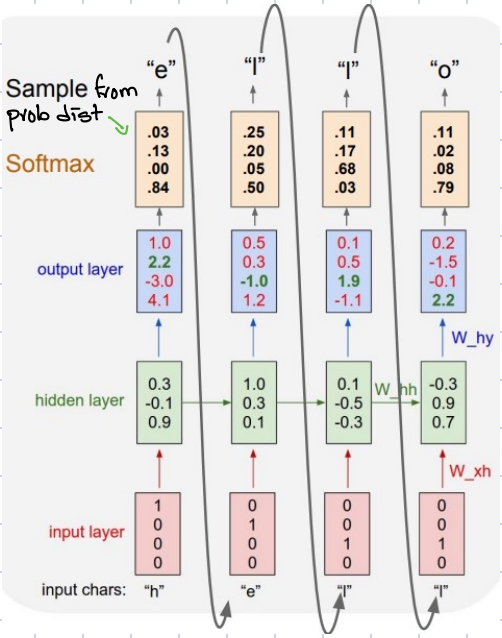
Character level language model:

Vocabulary: [h, e, l, o]

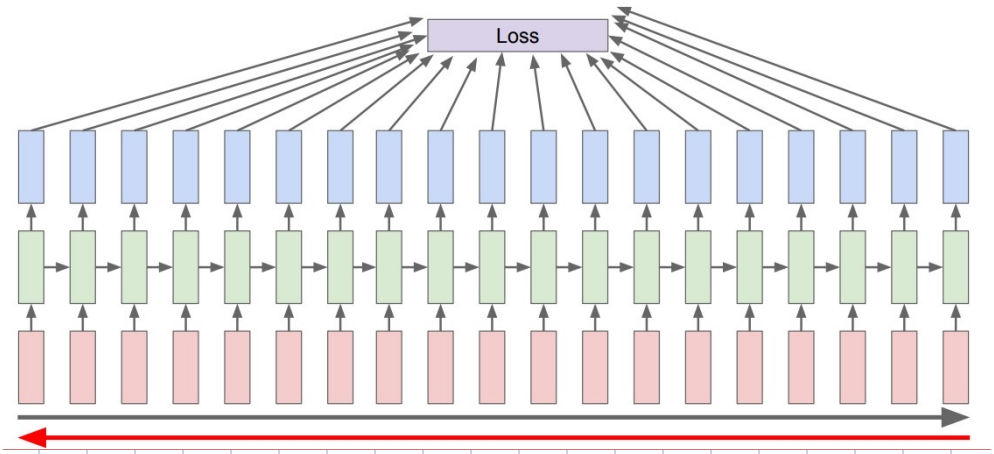
Example training sequence: "hello"

At test time, sample chars one at a time & feed back to model





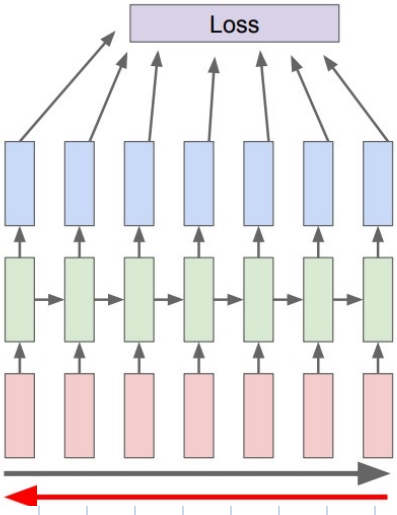
Backpropagation through time:



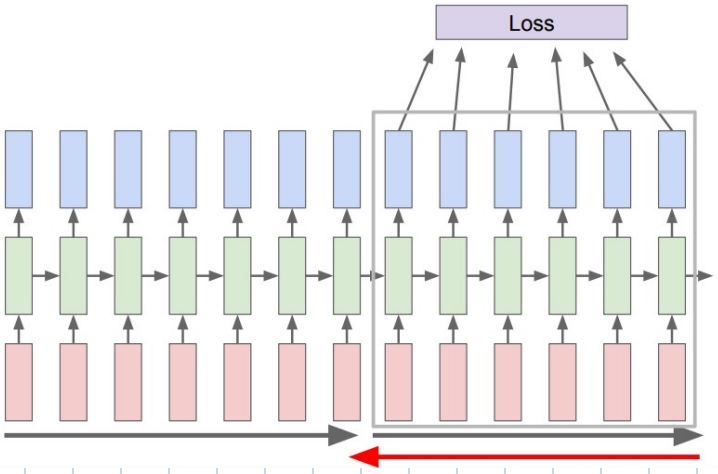
↳ forward through a entire sequence to compute loss, then backward through entire sequence to compute gradient

↳ Very bad. Very large sequence can slow model. So,

Truncated Backpropagation through time:

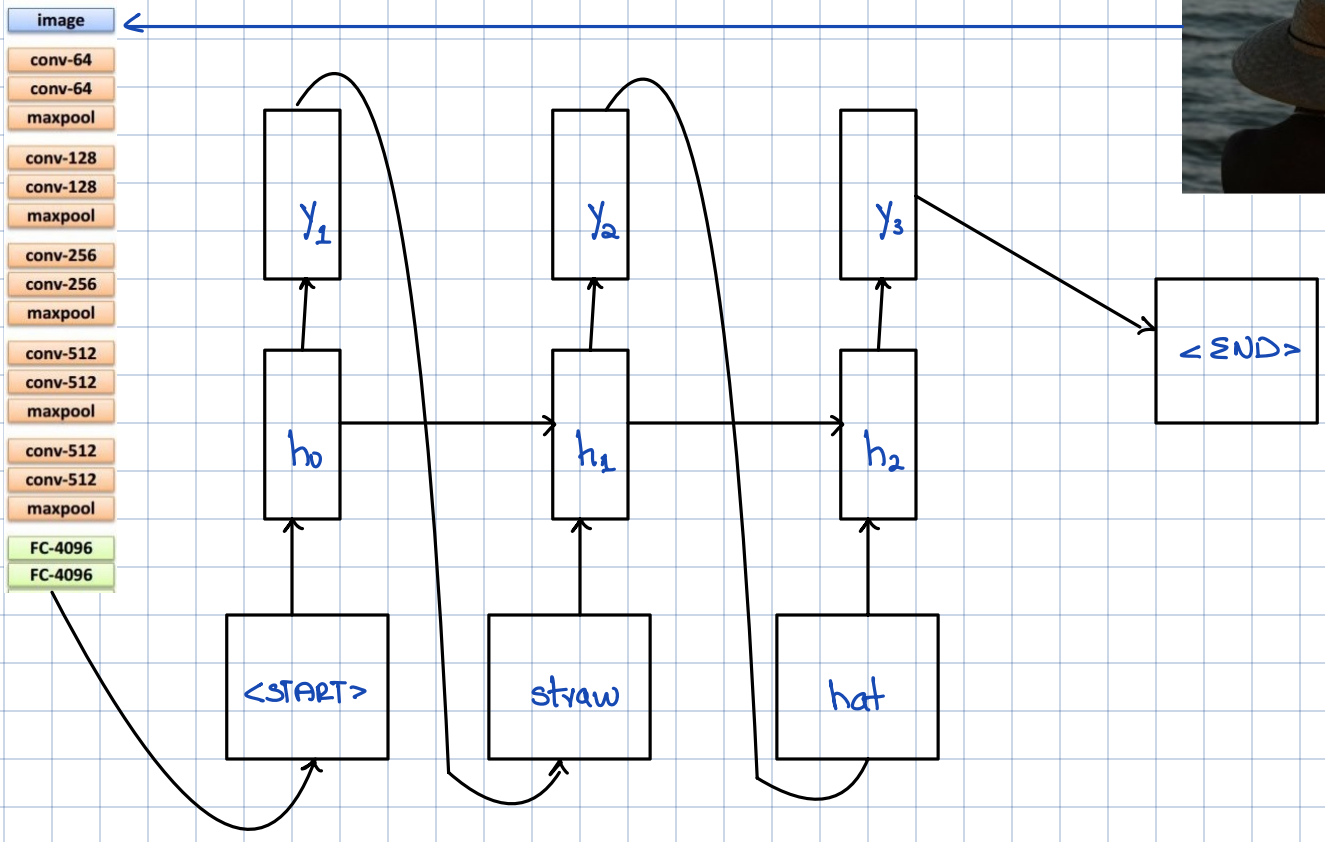


↳ Run forward & backward through chunks of the sequence instead of whole sequence



↳ Carry hidden states forward in time forever, but only backprop for some smaller no. of steps

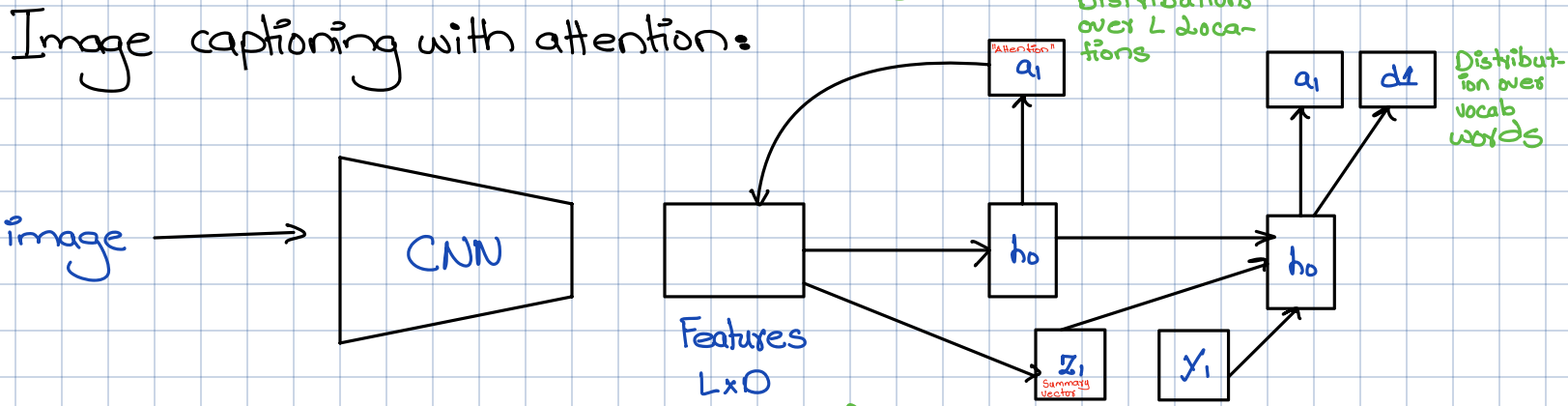
Image Captioning:



$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

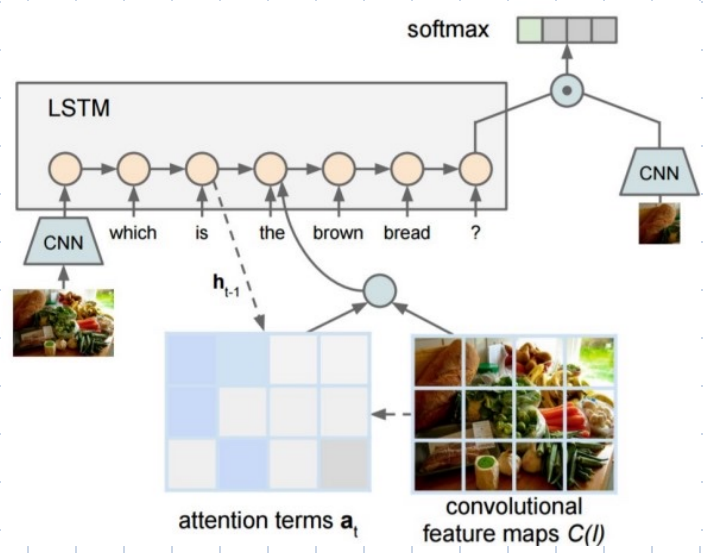
↳ image contribution

Image captioning with attention:



↳ One vector for each spatial loc in image $z = \sum_{i=1}^{L} p_i v_i$

Visual Question Answering



Multilayer RNN:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

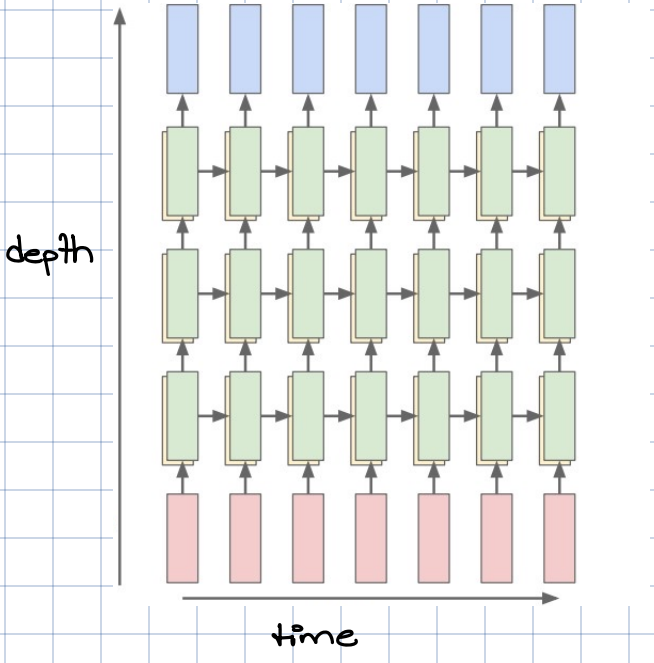
$h \in \mathbb{R}^n \qquad W^l [n \times 2n]$

LSTM:

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

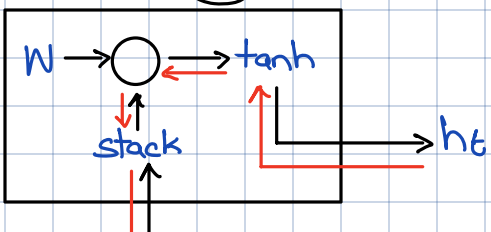
$$c_t^l = f \cdot c_{t-1}^l + i \cdot g$$

$$h_t^l = o \cdot \tanh(c_t^l)$$



↳ input goes through RNN
↳ produces sequence of hidden states
↳ use hidden states sequence as input

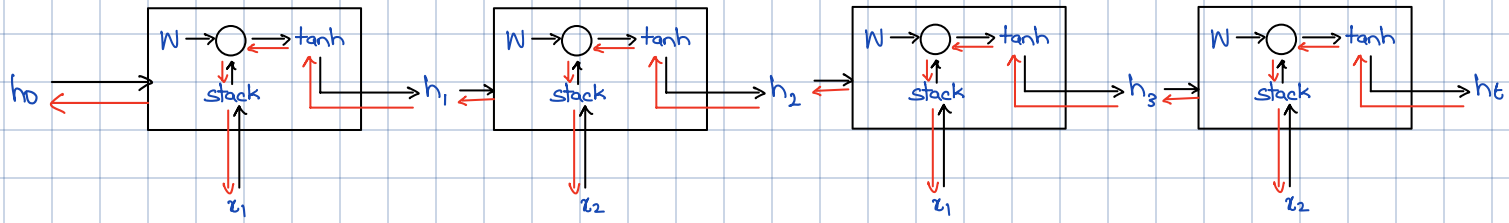
Vanilla RNN gradient flow:



$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$= \tanh\left((W_{hh} \; W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

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



Largest singular value > 1 ⟶ Grad clipping:

Exploding grads

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```

Largest singular value < 1 ⟶ Change architecture (LSTM)

Vanishing grads

2STM:

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

cell state

$c_t = f \cdot c_{t-1} + i \cdot g$ ⟶ How much of prev cell state we need to forget + How much to write if $i=1$ vice versa

$h_t^l = o \cdot \tanh(c_t)$

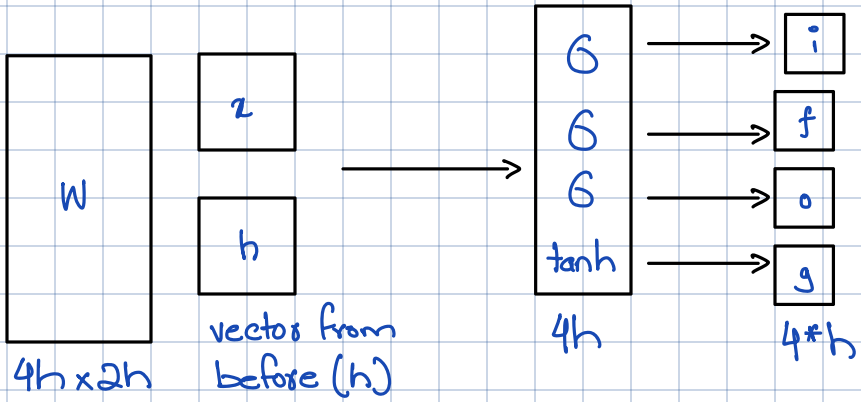
↳ hidden state

f: forget gate, whether to erase cell

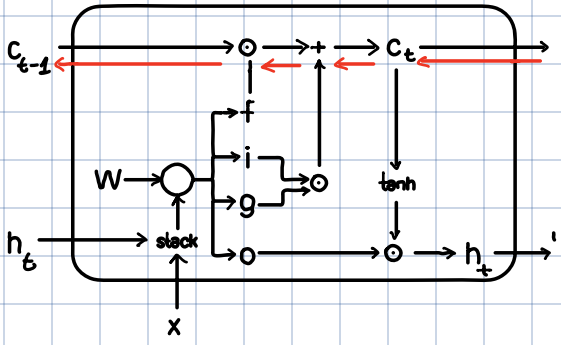
i: input gate, how much do we input into cell

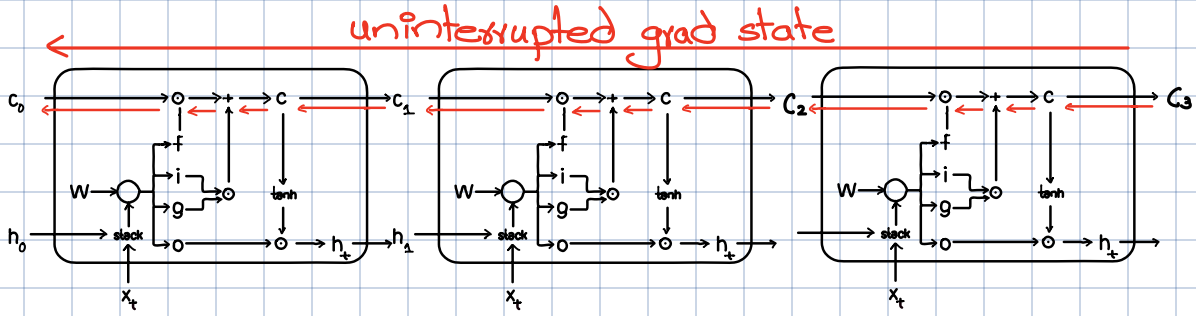
g: Gate gate, How much to write to cell

o: Output gate, How much to reveal cell

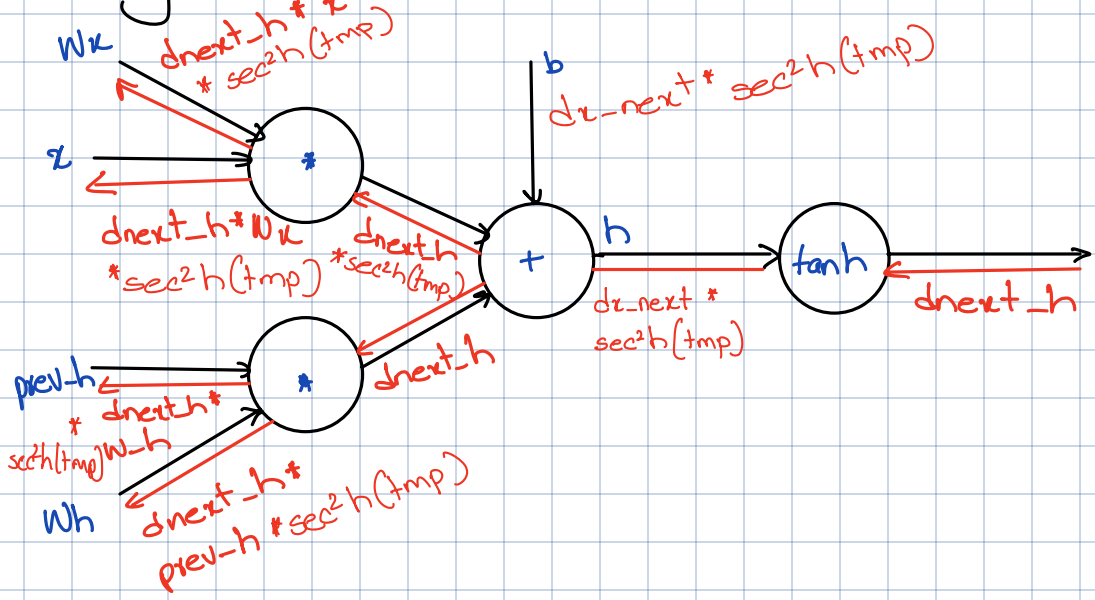


Computational graphs:



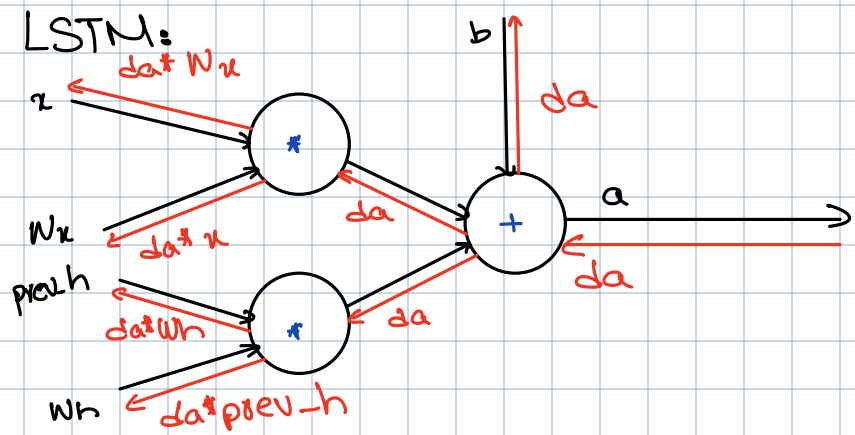


Assignment helper:

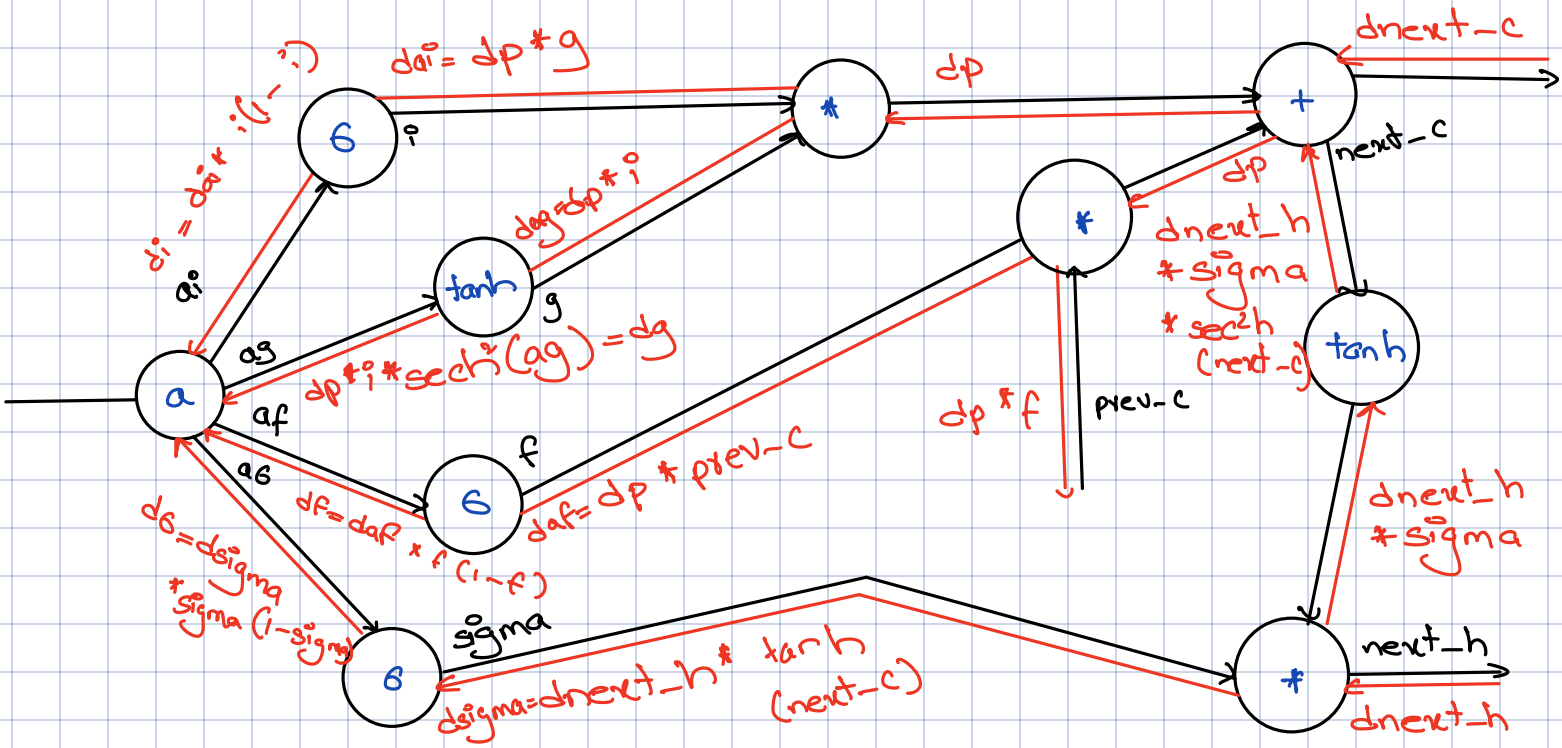


$\frac{\partial}{\partial x} \tanh x = \operatorname{sech}^2 x$

$tmp = Wh * prev\_h + Wx * x + b$



$dp = dnext\_h * sigma * \sec^2 h(\,) + dnext\_c$



$$da = di + df + d\sigma + dg$$