ML and Optimization Lecture 23

- Recurrent neural networks for sequential data
  - common model architectures
- Backpropagation through time (BTT)
- An NLP example

Drawbacks to FFNs:
- don't handle sequential data "naturally"

Ex applications w/ sequential data:

1) NLP (natural language processing)

text $\xrightarrow[ML]{}$ summary (ex: seq-to-seq)

text $\xrightarrow[ML]{}$ text in another language

2) CV + NLP:

image captioning: image $\xrightarrow[ML]{}$ caption

Classical models for sequential data (e.g. HMMs) typically feature:

- a state that describes a dynamical system
- an input to the system
- an update rule saying how the state changes given an input
- an (optional) output from the changed system

Examples:

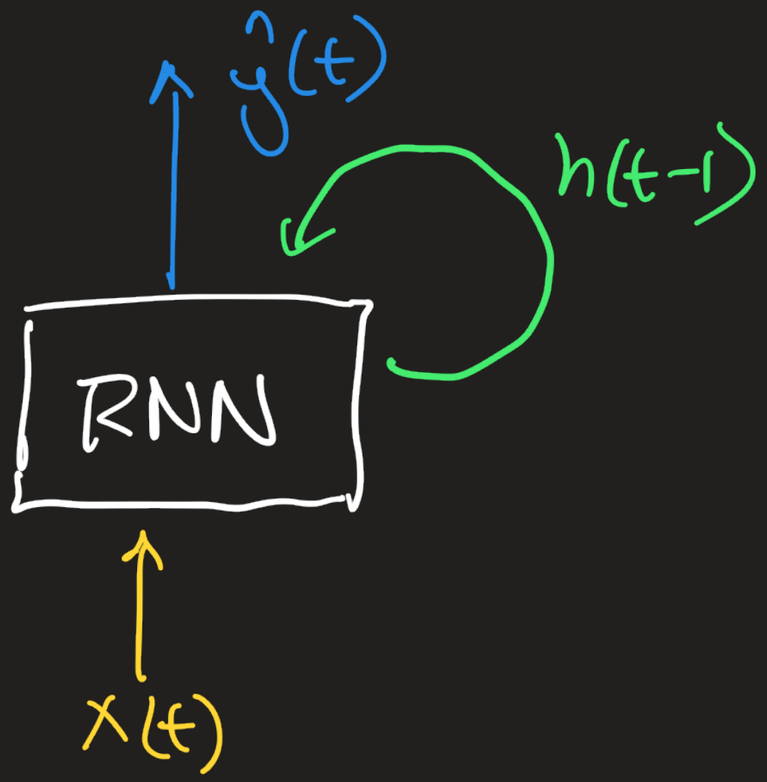
- ODEs
- HMMs
- autoregress processes
- etc.

Recurrent Neural Networks (RNNs)

use NNs to reproduce these features of state-space models of dynamical systems

$h(t)$: hidden state at time t
$x(t)$: input to system at time t
$\hat{y}(t)$: predictions/outputs at time t (optional)

} all vectors! (can be of different sizes)



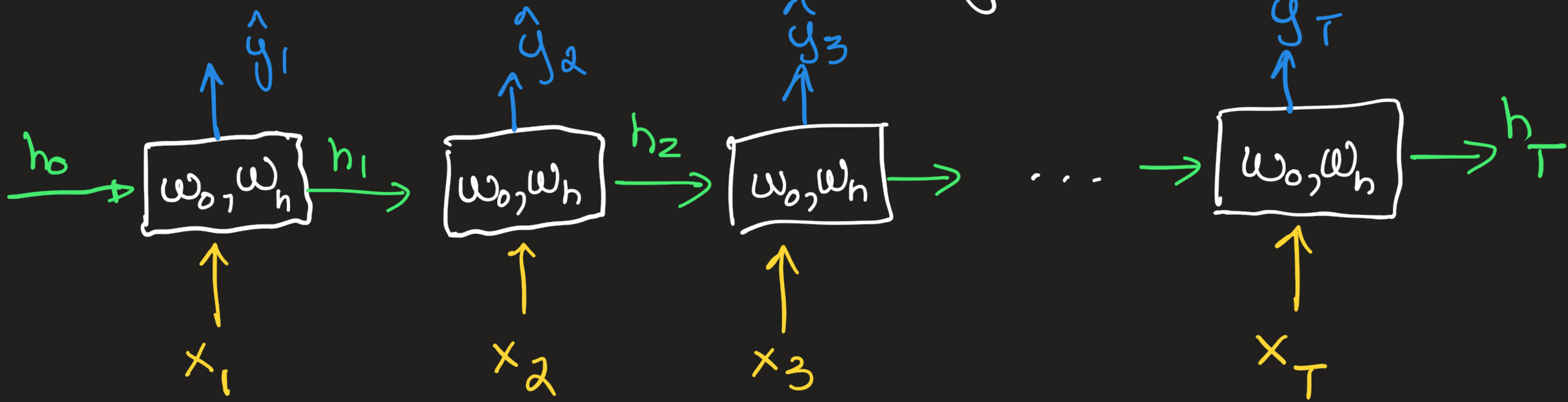
NN parameters for hidden state updates

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

$$\hat{y}_t = g(h_t, w_o)$$

NN params for output formula

We can unfold RNNs in time e.g.



Note that the blocks  all share the same parameters:

W_o — for computing output

W_h — for updating the state

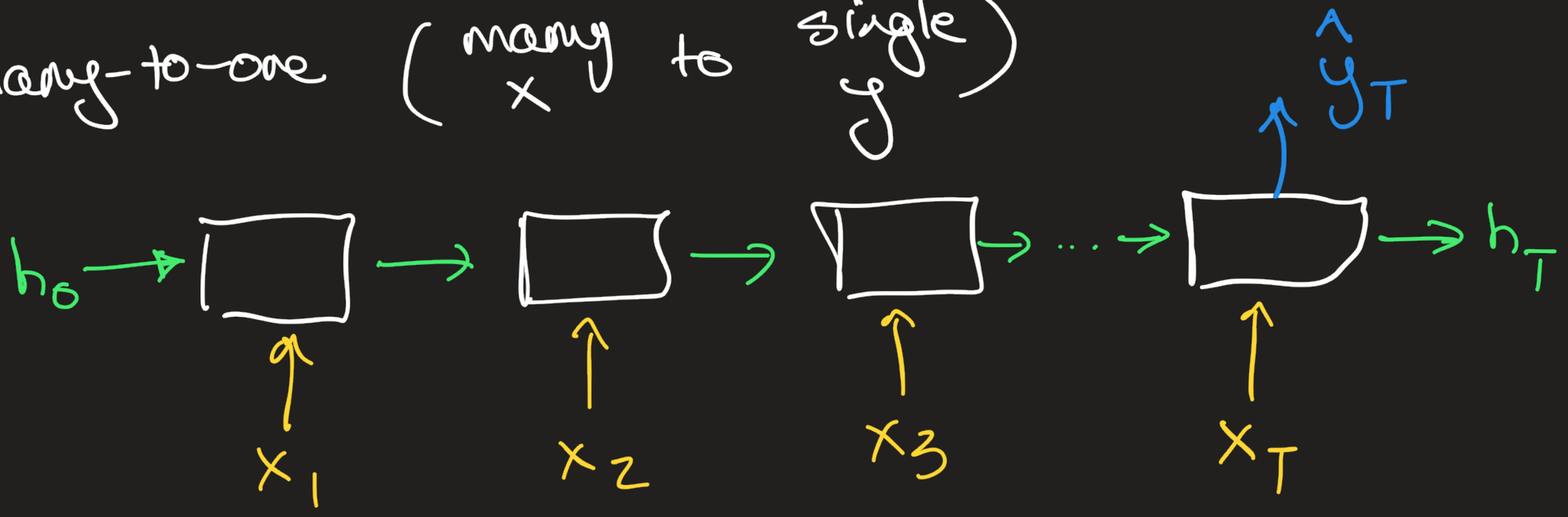
Design Patterns for RNNs

- Many-to-many (seen above) (many x to many $\hat{y}$)

  useful for, e.g., NLP tagging task like part-of-speech determination

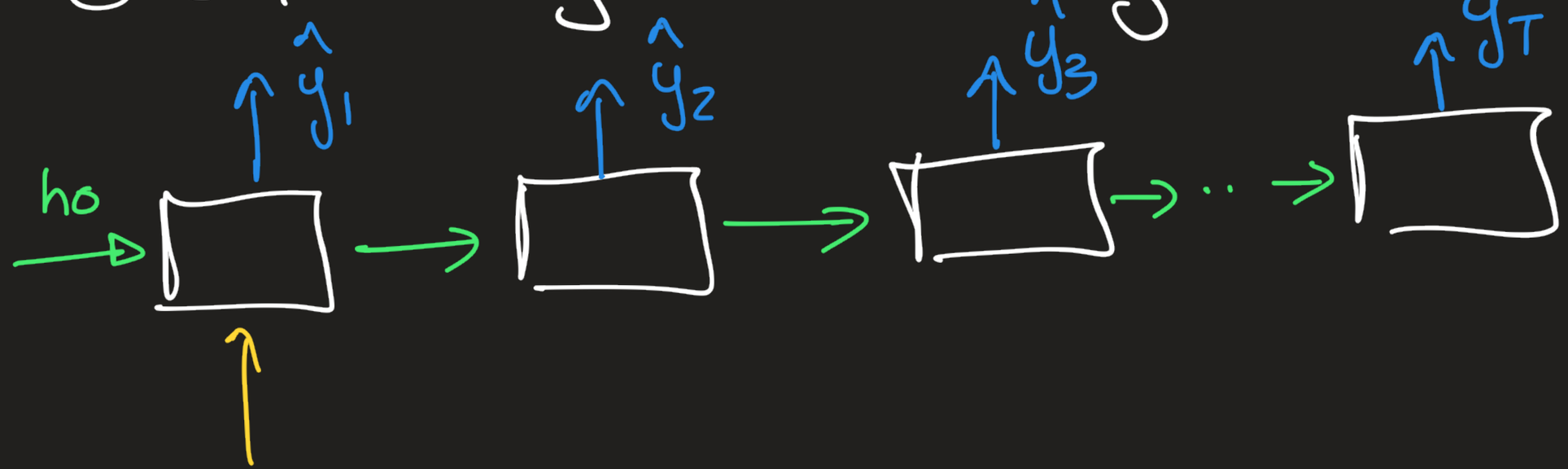
The cat ate its food
ART N V PP N

- Many-to-one (many x to single y)



use cases:
- time series prediction
- sentiment analysis
- sequence classification

- One-to-many (one x to many y)



useful for captioning: x - some representation of an image
(CNN features or autoencoder features)
and $\hat{y}_1, \ldots, \hat{y}_T$ should be the text of a caption

Vanilla RNNs

$$a_t = W x_t + U h_{t-1} + b$$

$$h_t = \sigma(a_t)$$ — use tanh typically so h_t can be neg or pos

} hidden state update network

$$o_t = V h_t + c$$

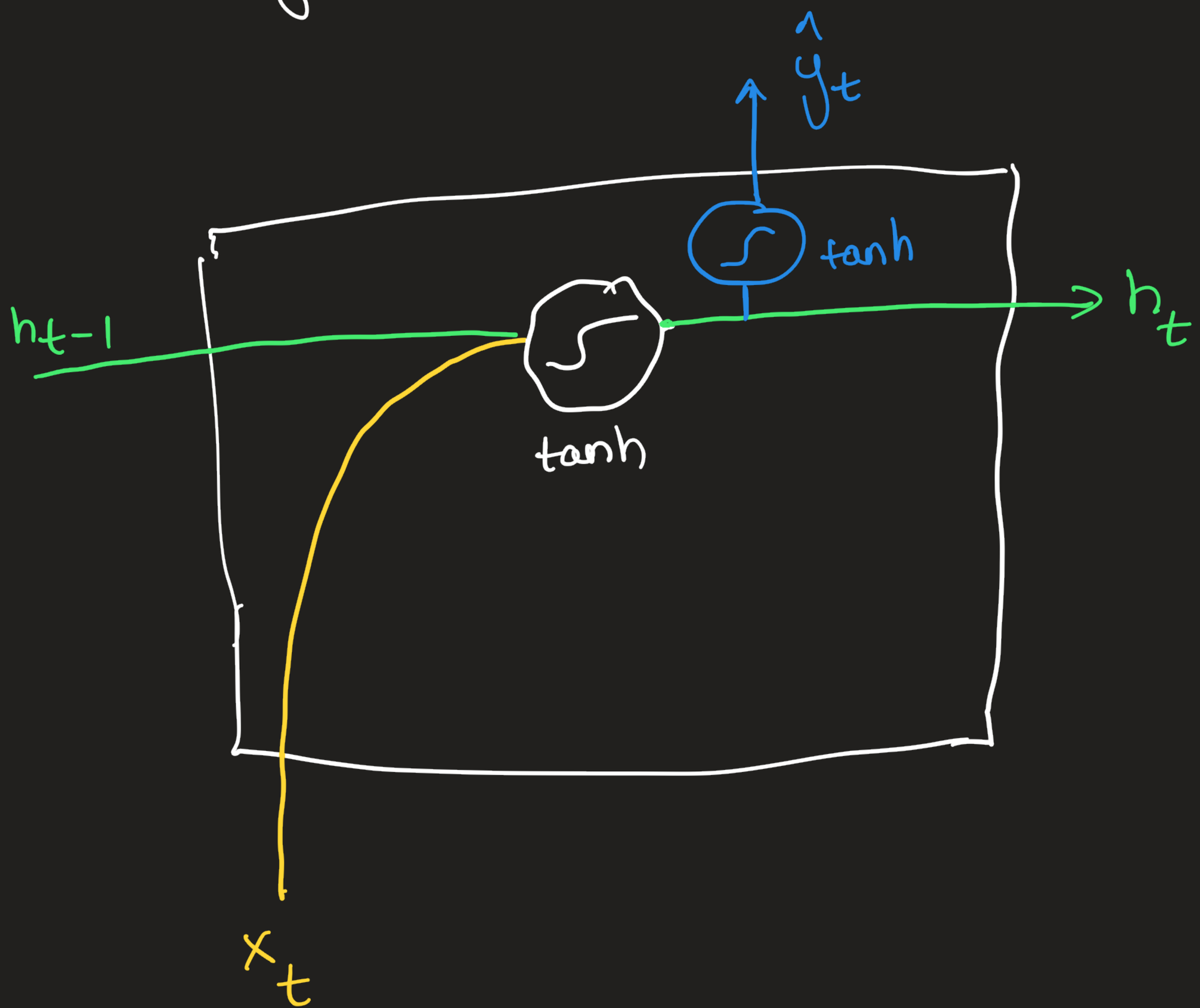
$$\hat{y}_t = \sigma(o_t)$$ — use tanh typically so $\hat{y}_t$ can be neg or pos

} output formula

parameters are $\omega_h = \{W, U, b\}$

$\omega_o = \{V, c\}$

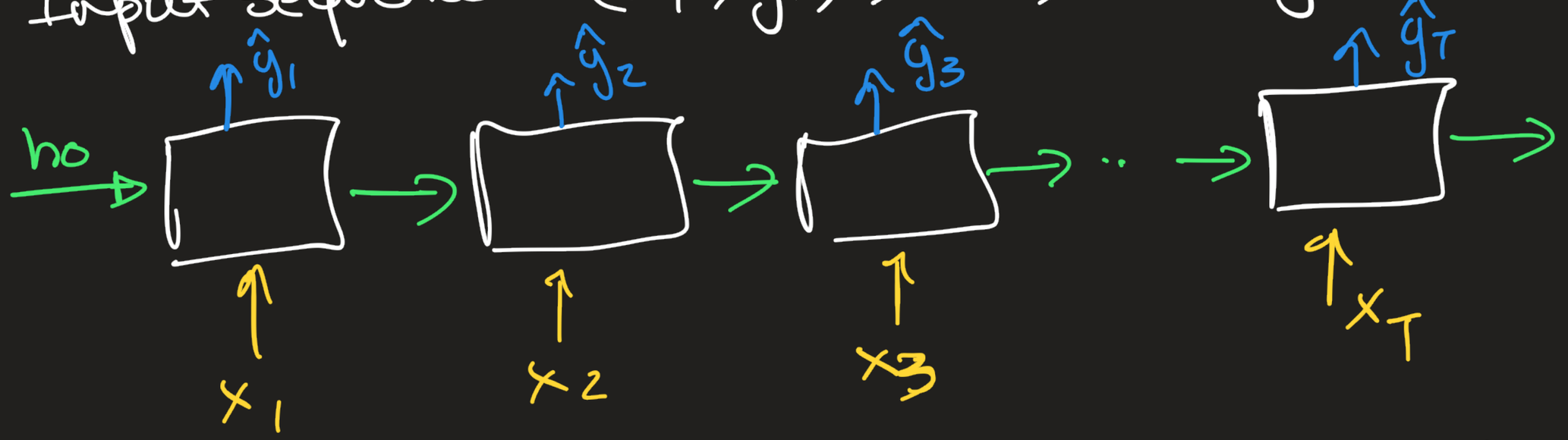
Flow diagram for a vanilla RNNs



Loss for RNNs

many-to-many case (changes are straight-forward for the other cases)

Input sequence: $(x_1, y_1), \dots, (x_T, y_T)$



overall loss of RNN for this input sequence is

$$L = \frac{1}{T} \sum_{t=1}^{T} l(\hat{y}_t, y_t)$$

Note that we can only compute $\hat{y}_t$ if we computed h_t, which requires $h_0, \dots, h_{t-1}$

Training RNNs Backpropagation Through Time (BTT)

Idea: unfold the neural network and at each training point (x_t, y_t) compute

$$\nabla_{w_h} \ell(\hat{y}_t, y_t) \text{ and } \nabla_{w_o} \ell(\hat{y}_t, y_t)$$

then average these together to get

$$\nabla_{w_h} L = \frac{1}{T} \sum_{t=1}^{T} \nabla_{w_h} \ell(\hat{y}_t, y_t)$$

$$\nabla_{w_o} L = \frac{1}{T} \sum_{t=1}^{T} \nabla_{w_o} \ell(\hat{y}_t, y_t)$$

How to compute $\nabla_{w_h} \ell(\hat{y}_t, y_t)$ for $t \geq 1$?

For simplicity we will assume w_h is a scalar, so

$$\nabla_{w_h} \ell(\hat{y}_t, y_t) = \frac{\partial}{\partial w_h} \ell(\hat{y}_t, y_t)$$

For generality we return to our generic RNN model

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

$$\hat{y}_t = g(h_t, w_o)$$

We want to compute

$$(*)\quad \nabla_{w_h} L = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l}{\partial w_h}(\hat{y}_t, y_t)$$

recall $\hat{y}_t = g(h_t, w_o)$ so by the chain rule

$$(*) = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l}{\partial \hat{y}_t}(\hat{y}_t, y_t)\,\frac{\partial \hat{y}_t}{\partial w_h}$$

$$= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l}{\partial \hat{y}_t}(\hat{y}_t, y_t)\,\frac{\partial g}{\partial w_h}(h_t, w_o)$$

$$= \frac{1}{T}\sum_{t=1}^{T} \underbrace{\frac{\partial l}{\partial \hat{y}_t}(\hat{y}_t, y_t)\,\frac{\partial g}{\partial h_t}(h_t, w_o)}_{\text{easy to compute}}\,\underbrace{\frac{\partial h_t}{\partial w_h}}_{\text{more complicated}}$$

Recall

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

so

$$\underbrace{\frac{\partial h_t}{\partial w_h}}_{d_t} = \underbrace{\frac{\partial f}{\partial w_h}(x_t, h_{t-1}, w_h)}_{b_t} + \underbrace{\frac{\partial f}{\partial h_{t-1}}(x_t, h_{t-1}, w_h)}_{e_t} \cdot \underbrace{\frac{\partial h_{t-1}}{\partial w_h}}_{d_{t-1}}$$

Note that b_t and e_t are easy to compute because at time t, we know x_t, h_{t-1} and w_h; further we know f so we have expressions for $\frac{\partial f}{\partial w_h}$ and $\frac{\partial f}{\partial h_{t-1}}$.

Let

$$d_t := \frac{\partial h_t}{\partial w_h}$$

$$b_t := \frac{\partial f}{\partial w_h}(x_t, h_{t-1}, w_h)$$

$$e_t := \frac{\partial f}{\partial h_{t-1}}(x_t, h_{t-1}, w_h)$$

We now have the equations

$$d_t = b_t + e_t d_{t-1} \quad \text{for } T \geq t \geq 1$$

$$d_0 = \frac{\partial h_0}{\partial w_h} = 0$$

We see

$$d_1 = b_1 + e_1 d_0 = b_1$$

$$d_2 = b_2 + e_2 d_1 = b_2 + e_2 b_1$$

$$d_3 = b_3 + e_3 d_2 = b_3 + e_3 (b_2 + e_2 b_1)$$

$$= b_3 + e_3 b_2 + e_3 e_2 b_1$$

$$d_4 = b_4 + e_4 d_3 = b_4 + e_4 b_3 + e_4 e_3 b_2 + e_4 e_3 e_2 b_1$$

By induction we can show that for $T \geq t \geq 1$, the solution to the difference equation satisfies

$$d_t = b_t + \sum_{\Delta=1}^{t-1} b_\Delta \left(\prod_{j=\Delta+1}^{t} e_j \right) \qquad (**)$$

where

$$d_t = \frac{\partial h_t}{\partial w_h}$$

$$b_t = \frac{\partial f}{\partial w_h}(x_t, h_{t-1}, w_h)$$

$$e_t = \frac{\partial f}{\partial h_{t-1}}(x_t, h_{t-1}, w_h)$$

This can be plugged into (*) to get the final expression for $\nabla_{w_h} L$

In practice, on long sequences we use <u>truncated BTT</u> by terminating the sum (**) at τ steps into the past:

$$\frac{\partial h_t}{\partial w_h} \approx \frac{\partial f}{\partial w_h}(x_t, h_{t-1}, w_h) + \sum_{\Delta=t-\tau}^{t} \left[\left(\prod_{j=\Delta+1}^{t} \frac{\partial f}{\partial h_{j-1}}(x_j, h_{j-1}, w_h) \right) \cdot \frac{\partial f}{\partial w_h}(x_\Delta, h_{\Delta-1}, w_h) \right]$$

This both is more computationally cheap <u>and</u> serves as a form of regularization: it trains the model to depend on recent history and results in more stable training.

Notice that, because of the recursive nature of the updates for $\frac{\partial h_t}{\partial w_h}$, the vanishing and exploding gradients problem becomes worse for RNNs

imagine $h_t = f(x_t, h_{t-1}, w_h) = \sigma(x_t + w_h h_{t-1})$

$$\Rightarrow \frac{\partial f}{\partial h_{j-1}} = w_h \sigma'(x_t + w_h h_{t-1})$$

and

$$\prod_{j=\Delta+1}^{t} \frac{\partial f}{\partial h_{j-1}}(x_j, h_{j-1}, w_h) = w_h^{t-\Delta} \prod_{j=\Delta+1}^{t} \sigma'(x_j + w_h h_{j-1})$$

w_h is either >1 or <1, so $w_h^{t-\Delta} \ll 1$ or $w_h^{t-\Delta} \gg 1$

so we either get vanishing or exploding gradients

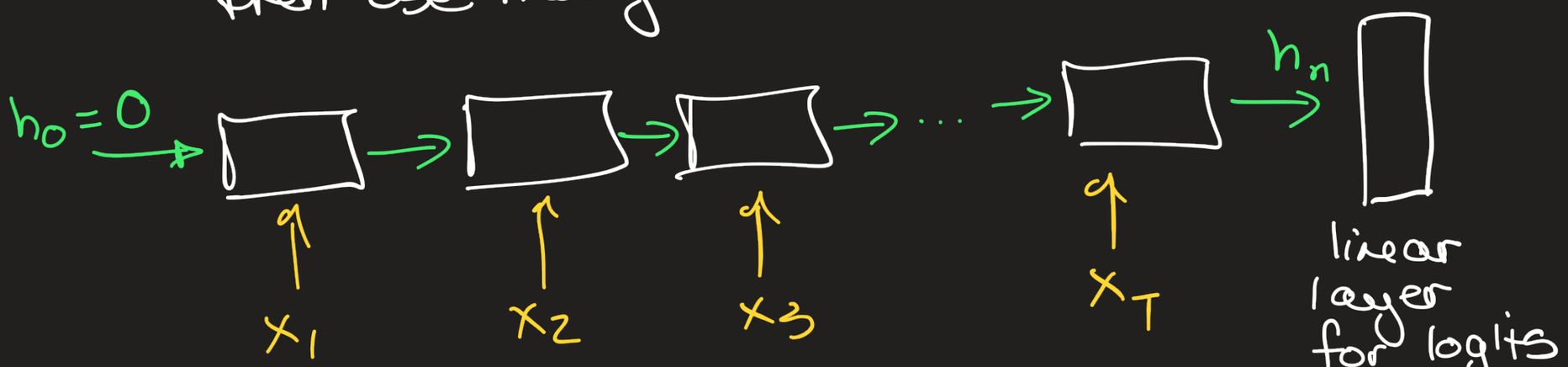
Example Application (NLP)

Classification of text sequences

Input: "Iraq Halts Oil Exports"

Output: class 3 (business/economic news)

Idea: split input into tokens, convert them to vectors, then use many to one RNN framework



First, find it is useful to augment the input sequence to indicate the start and end of inputs, e.g.

<BOS> Iraq Hail Oil Exports <EOS>

x_1 x_2 x_3 x_4 x_5 x_6

but also note that in fact we need the inputs x_i to be vectors

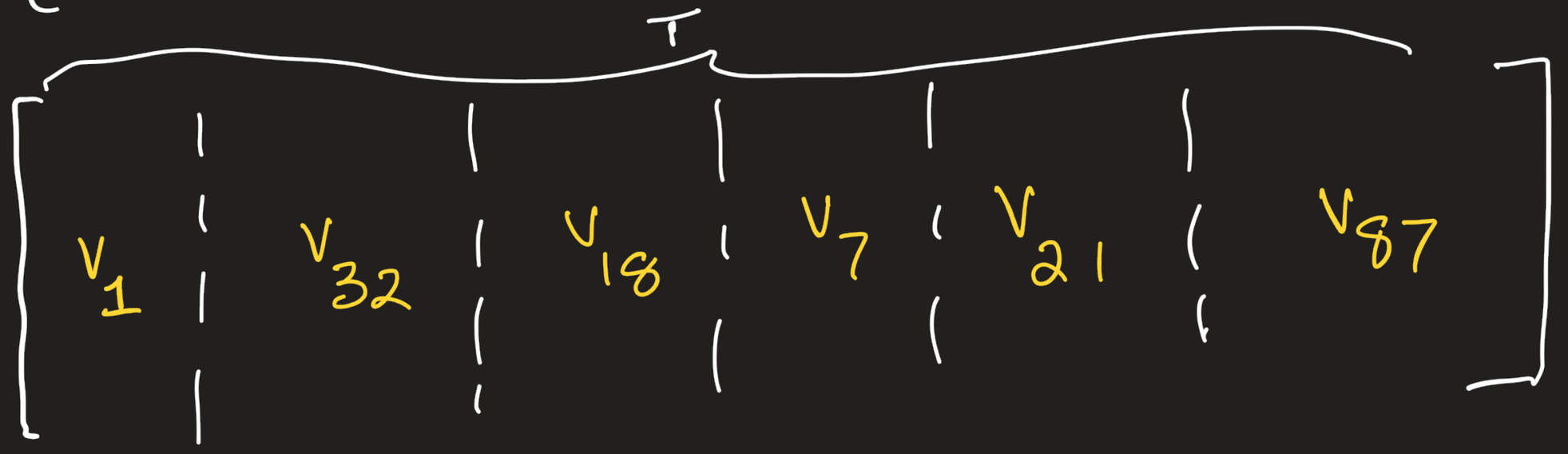
We use an embedding layer for this:

- convert the tokenized input text into indices into a vocabulary
- use a look-up table to find a vector representing each word

Ex:

<BOS> Iraq Halts Oil Exports <EOS>

↳ [1 32 18 7 21 87]

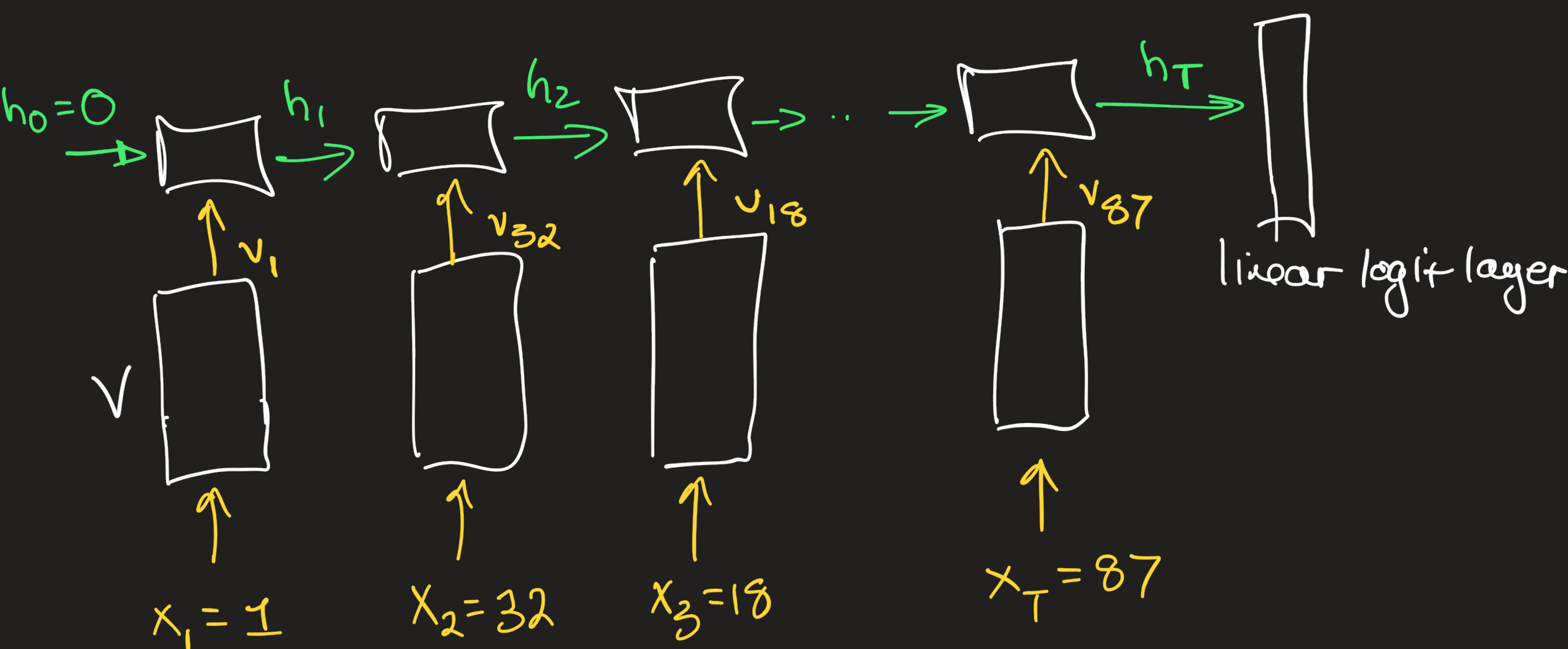
↳ 

$= [\, x_1 \quad x_2 \quad \cdots \quad x_T \,]$

↑ this sequence of T d-dimensional vectors is the input to my RNN

Let $V = \begin{bmatrix} v_1^T \\ \vdots \\ v_m^T \end{bmatrix}$ be the look-up table for our vocabulary (there are m tokens in our vocab)

The final architecture for this NLP task looks like



To train, we backpropagate over the parameters for the RNN, the logit layer, and the embedding layer

Practical considerations:

- to embed out of vocabulary words, use a special UNKNOWN token in V
- because we often want to learn over minibatches and this is most efficient when the sequences have the same lengths, we pad the shorter sequences with a special PAD token to have the same length, e.g.

<BOS> Iraq Halts Oil Exports <EOS>
<BOS> Belarus Shuts Borders <EOS> <PAD>