Sequence modeling via RNNs (recurrent neural networks)

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

Ex applications w/ sequential data:

1) NLP (natural language processing):

text $\xrightarrow[\text{ML}]{}$ summary

2) CV + NLP:

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

3) Machine Translation: $\text{text}_{\text{Eng}} \longleftrightarrow \text{text}_{\text{Mongolian}}$

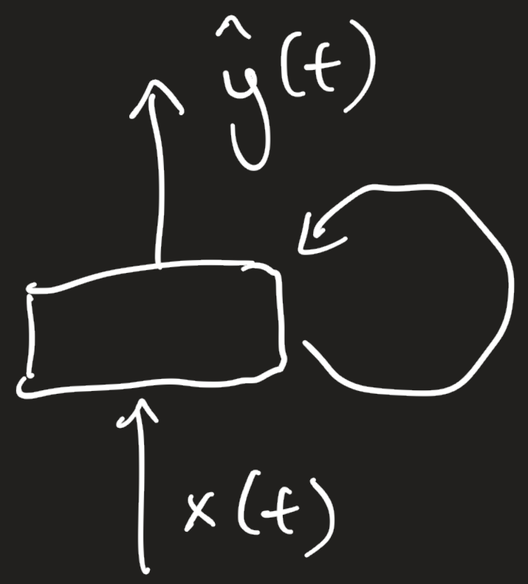
Classical models for sequential data:
(e.g. HMMs) feature:

- a state that describes the dynamical system
- an input to the system
- an update for how the state changes given an input
- an output from the changed system

Typically: ODEs, HMMs, or autoregressive processes, etc.

Recurrent Neural Networks (RNNs)

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



$h(t-1)$: hidden state at time $t-1$

$x(t)$: input at time t

$\hat{y}(t)$: prediction at time t (optional)

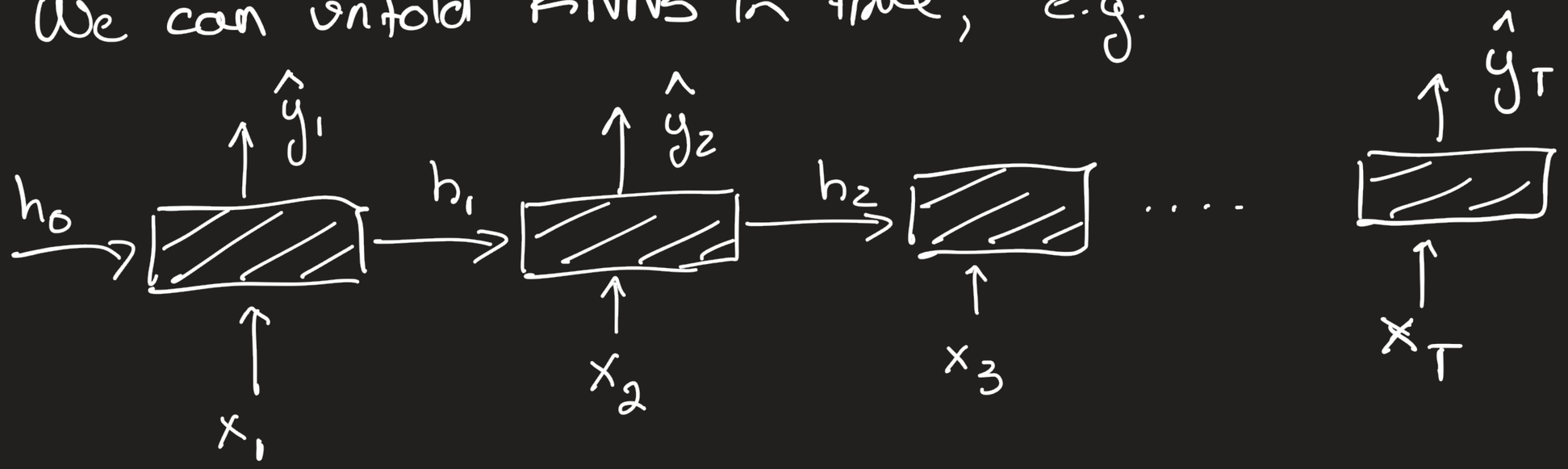
Update formula: $h_t = f(x_t, h_{t-1}, w_h)$

(w_h: NN params for hidden state updates)

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

(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_h and W_o.

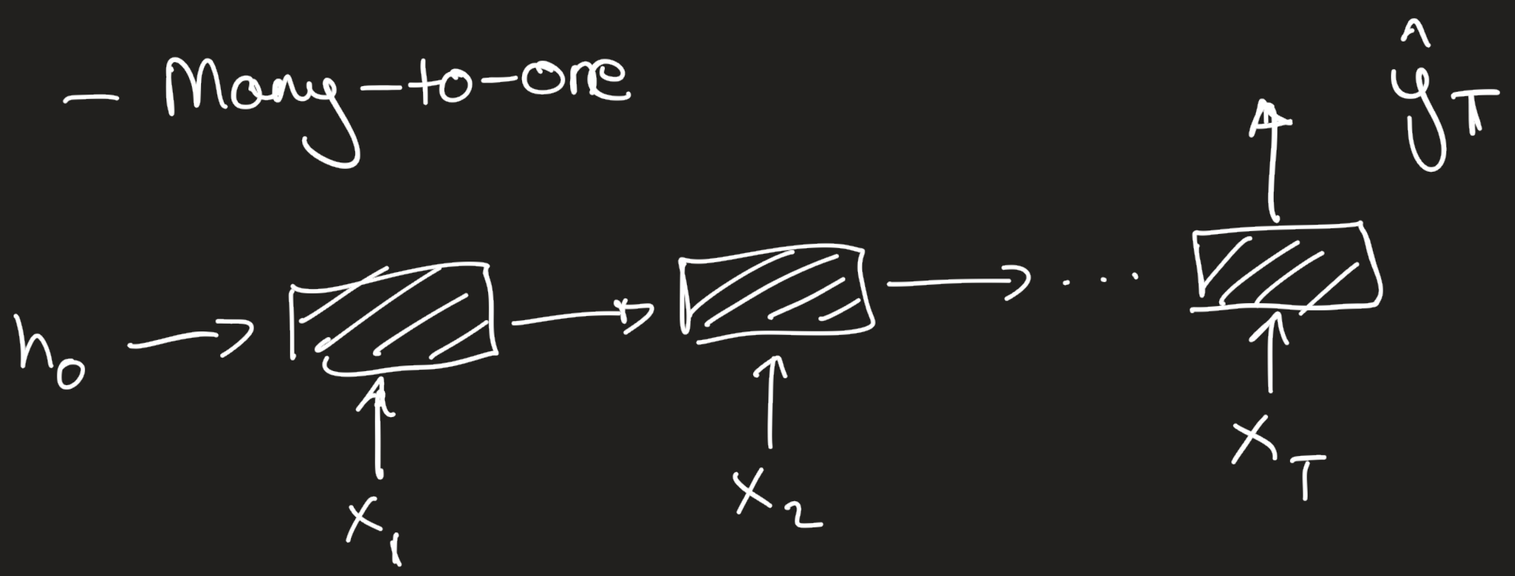
Design Patterns for RNNs

- Many-to-many (seen above)

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

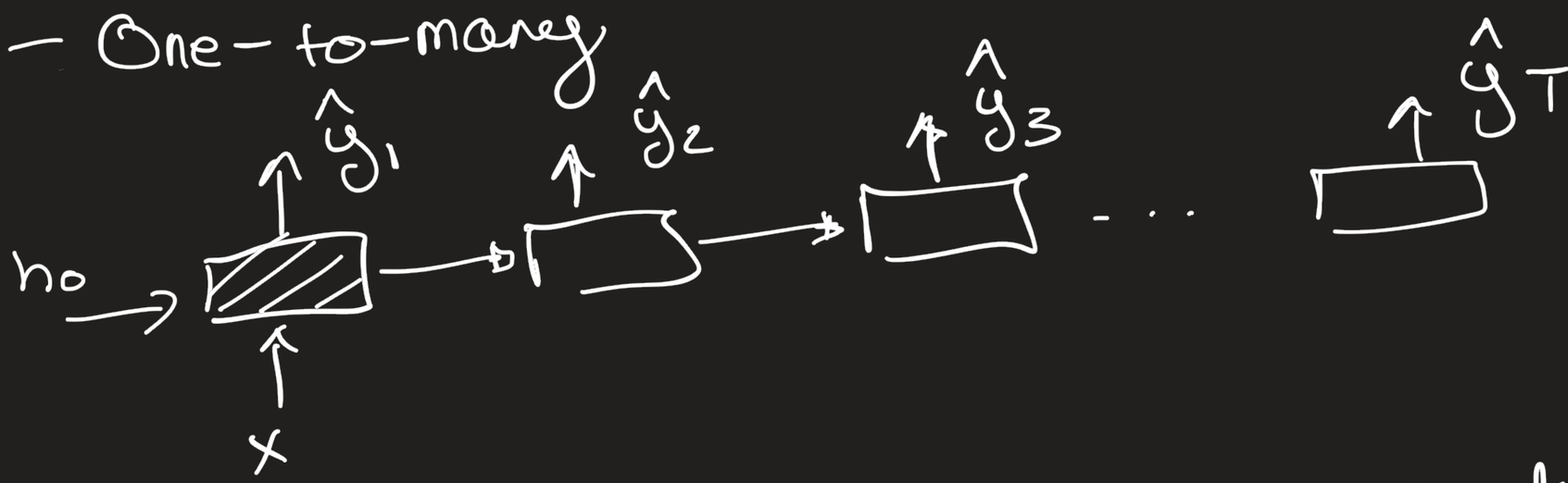
  The cat ate its food
  ART N V PP N

- Many-to-one



useful for say:
- time series prediction
- sentiment analysis

- One-to-many



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 rest of a caption

Vanilla RNNs

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

$$h_t = \sigma(a_t) \quad \text{— use } \sigma = \tanh$$

} update formula

$$o_t = V h_t + c$$

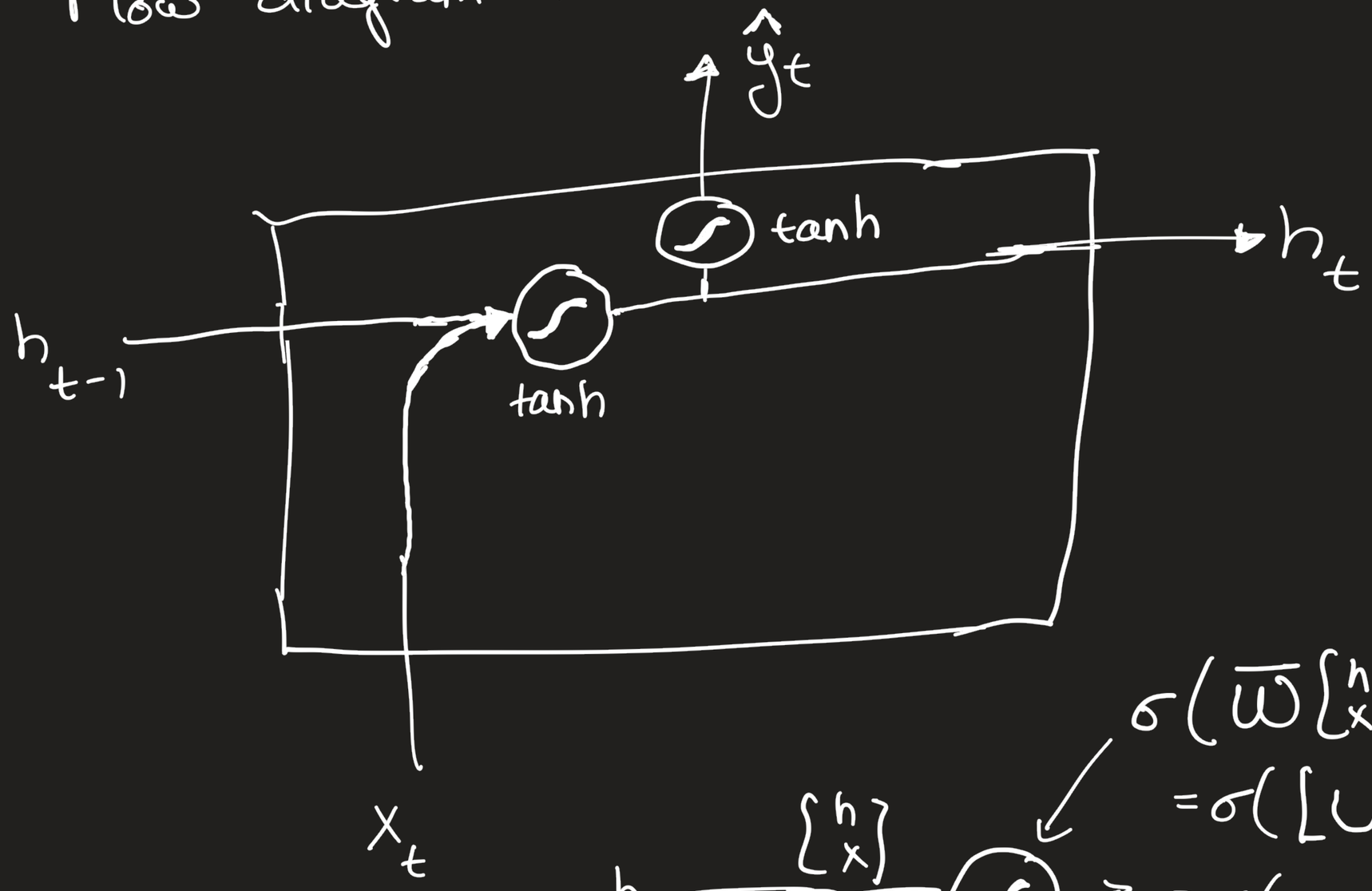
$$\hat{y}_t = \sigma(o_t) \quad \text{— use } \sigma = \tanh$$

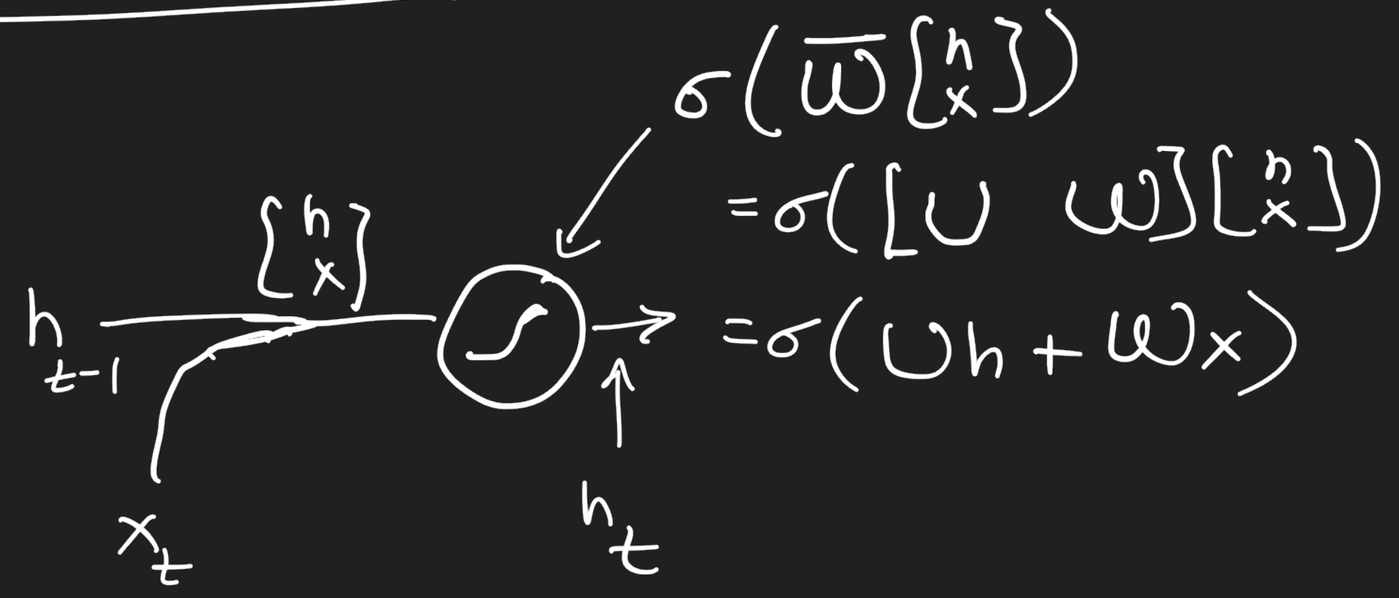
} output formula

parameters are: $w_h = \{W, U, b\}$

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

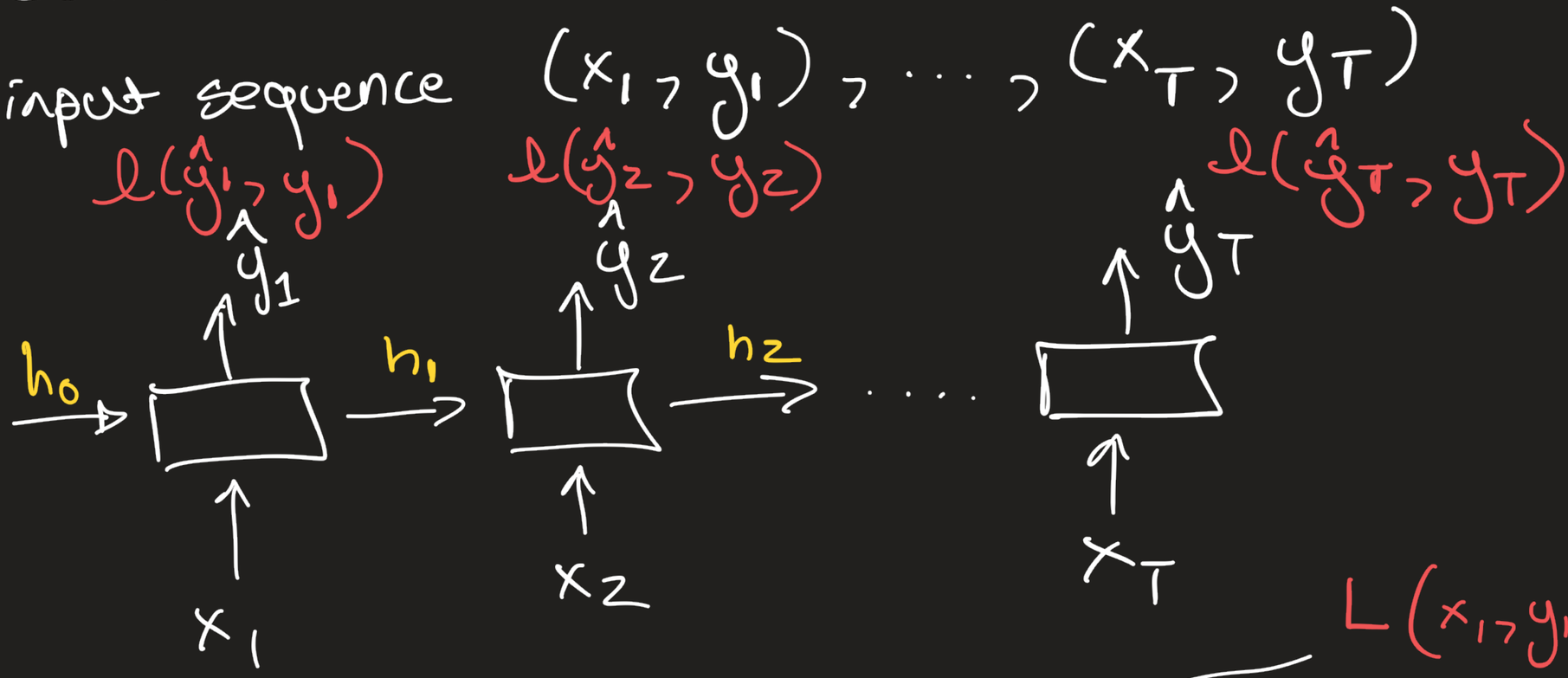
Flow diagram for a vanilla RNN





Loss for RNNs

Consider the unfolded RNN corresponding to an input sequence $(x_1, y_1), \ldots, (x_T, y_T)$



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

$$L(x_1, y_1, \ldots, x_T, y_T, \underline{\underline{w_h}}, \underline{\underline{w_o}})$$

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

Training RNNs Backpropagation Through Time (BTT)

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

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

$$\nabla_{w_0} \ell(\hat{y}_t, y_t)$$

then average these together to get

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

$$\nabla_{w_0} L = \frac{1}{T} \sum_{t=1}^{T} \nabla_{w_0} \ell(\hat{y}_t, y_t)$$

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

For simplicity we will assume w_h is a scalar, so

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

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

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

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

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

Recall

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

so

$$\frac{\partial h_t}{\partial \omega_h} = \underbrace{\frac{\partial f}{\partial \omega_h}(x_t, h_{t-1}, \omega_h)}_{b_t} + \underbrace{\frac{\partial f}{\partial h_{t-1}}(x_t, h_{t-1}, \omega_h)}_{e_t} \underbrace{\frac{\partial h_{t-1}}{\partial \omega_h}}_{\text{complicated}}$$

d_t

easy to compute at time t

d_{t-1}

Let

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

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

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

We now have the difference equation

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

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

We see

$$d_1 = b_1 + e_1 \cdot d_0 = b_1$$

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

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

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

$$d_4 = b_4 + e_4 \cdot d_3 = b_4 + e_4 \cdot (b_3 + e_3 \cdot b_2 + e_3 \cdot e_2 \cdot b_1)$$

$$= b_4 + e_4 \cdot b_3 + e_4 \cdot e_3 \cdot b_2 + e_4 \cdot e_3 \cdot e_2 \cdot b_1$$

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

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

recall $d_t = \frac{\partial h_t}{\partial w_n}$

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

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

This gives us our final expression for

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

which can then be used in $(*)$

Note that this computation is expensive when t is large, because we account for the influence of w_h on $h_1, \dots, h_{t-1}$ to account for how it influences h_t

More consequentially, because of the recursive nature of the updates for $\frac{\partial h_t}{\partial w_h}$, the vanishing and exploding gradients problem becomes worse:

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

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

clearly unstable

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

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

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

This is a form of regularization: it trains the model to depend on recent history and results in more stable training

Next time:

- LSTMs (long short-term memory RNNs)
- GRUs (vaguely)
- Deep RNNs
- Bidirectional RNNs
- Code examples → hand-coded & Keras

Post the lecture to MediaSite & move my office hours
(post the details to Piazza)