ML and Optimization Lecture 21

- Tools for using RNNs for NLP problems:
  Embedding layer & tokenization
- RNN example: AG_NEWS classification
- Long Short Term Memory (LSTM) networks
- Bidirectional & deep RNNs

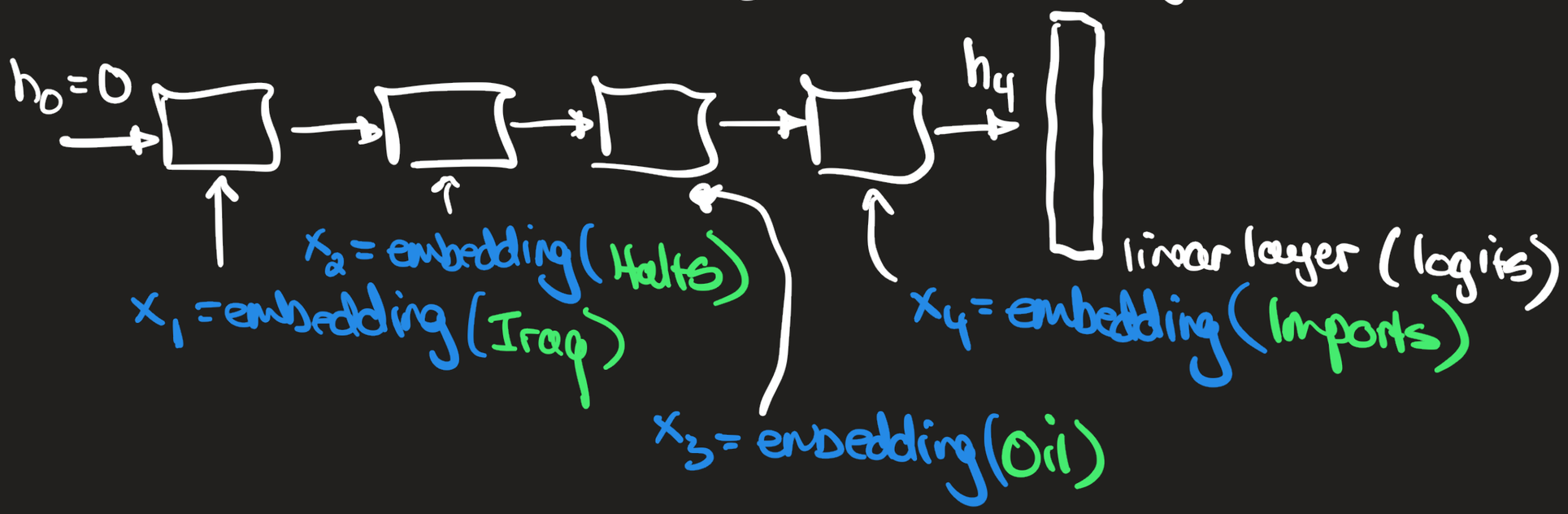
Example Application (NLP)

Classification of text sequences

Input: "Iraq Halts Oil Imports"

Output: class 3 (business/economic news)

Idea: split input sentence into tokens (tokenization),
convert them to vectors via a lookup table (word embeddings),
then we use the many-to-one RNN design followed by a softmax layer



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

<BOS>	Iraq	Halts	Oil	Imports	<EOS>
x_1	x_{23}	x_{532}	x_{11}	x_{72}	x_2

We use a three step process:

preprocessing
- tokenize the input
- lookup the index of each token into our vocabulary

inside of the model
- use a look-up table (embedding layer) to find the vectors corresponding to each word

Ex:

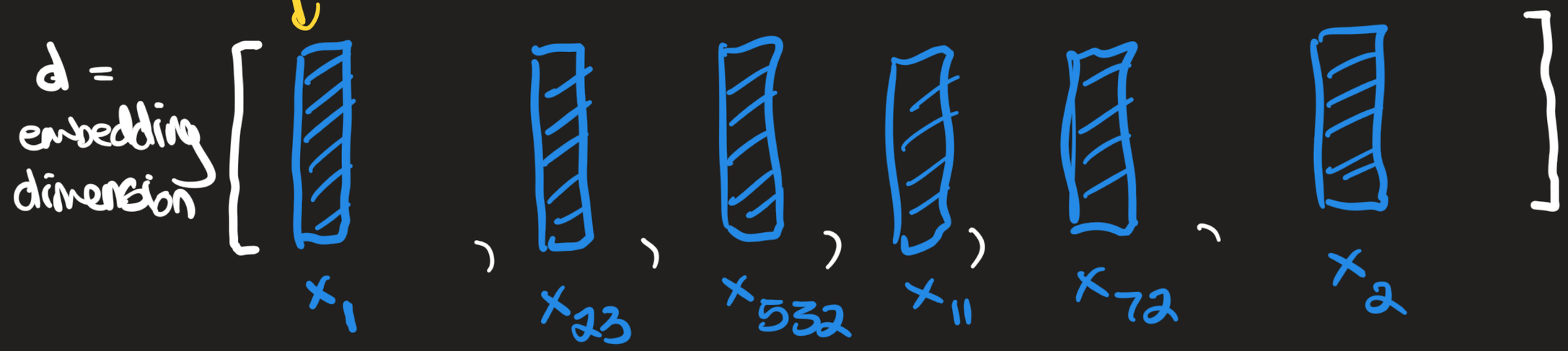
"Iraq Halts Oil Imports"

["<BOS7", "Iraq", "Halts", "Oil", "Imports", "<EOS>"]

input to the RNN → [1 , 23, 532, 11 , 72 , 2]

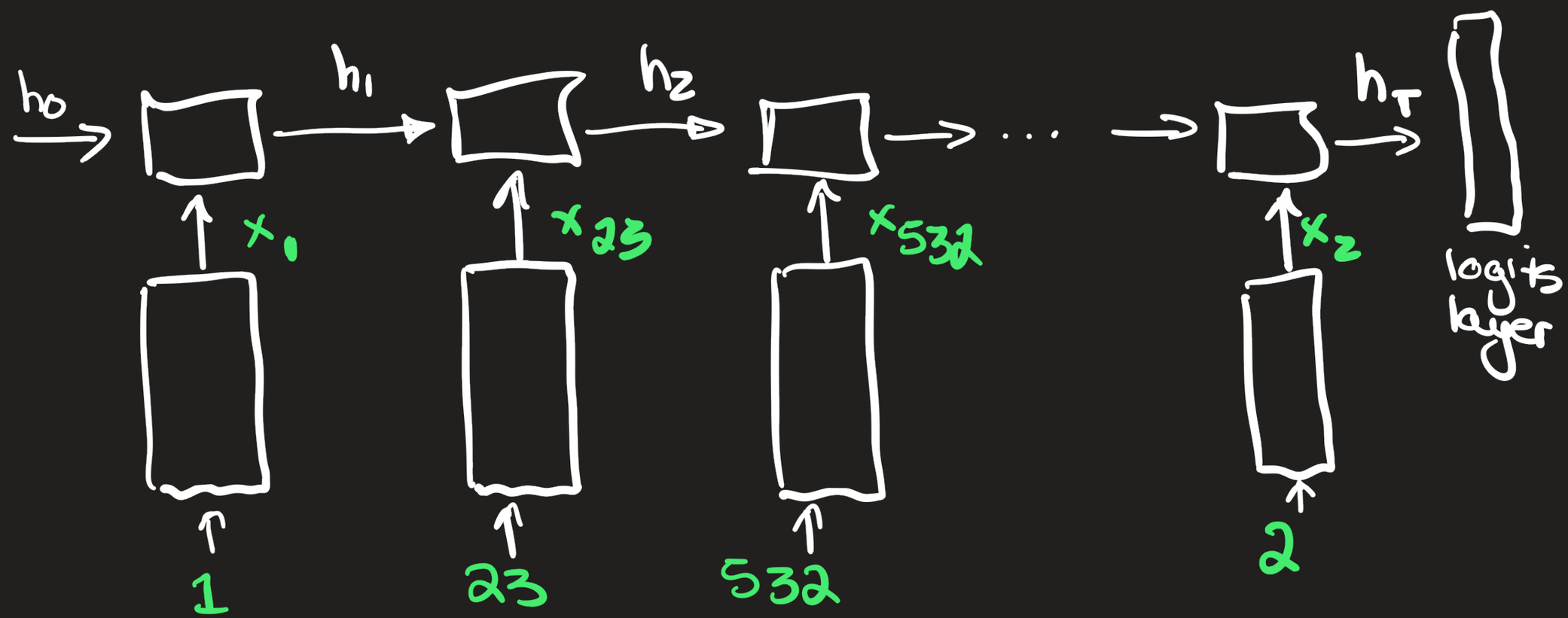
in each minibatch we use the updated embeddings for the tokens

d = embedding dimension



Let $V = \begin{bmatrix} v_1^T \\ \hline \vdots \\ \hline v_m^T \end{bmatrix} \in \mathbb{R}^{m \times d}$ be the look-up table for our vocabulary

The final architecture for this NLP task looks like



To train, we backprop (BPTT) over the logit layer, the parameters for the RNN, and the embedding layer

Practical considerations:

- to embed out-of-vocabulary word, use a special UNKNOWN token in V
- because we often want to learn over minibatches and this is very inefficient when the sequences have different lengths, we:
  1) chop sequences to have a finite maximum length
  2) pad shorter sequences with a special <PAD> token to have the maximum length, e.g.

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

Problem (Vanishing Gradients)

Simple RNNs work well for short sequences, but the hidden state acts as a primitive memory, so we can run into issues of the gradient vanishing or exploding for longer sequences. This is because they attempt to retain all information on the sequence prefix seen before.

Resolution (in practice, by consensus, the most popular RNN architectures)

LSTM (Long-Short Term Memories; 1997)

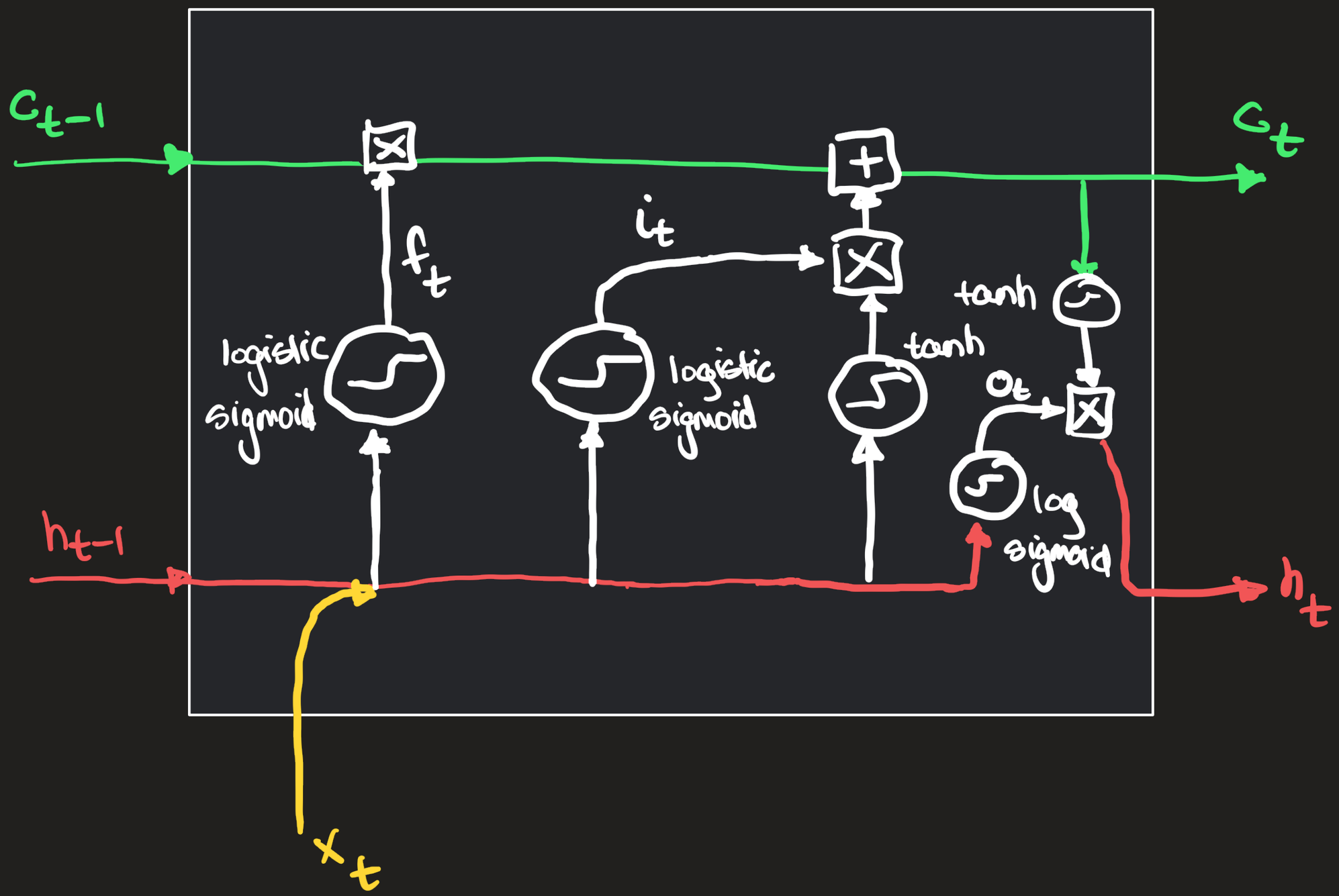
GRUs (Gated Recurrent Units; 2014)

focus on LSTMs: basic intuition is to selectively choose what the network will remember or forget

Introduce:

- cell state to represent our global memory that we can update to remember or forget, C_t
- forget gate - determines what we forget in the cell state at each time t, f_t
- input gate - determines what we remember in our cell state at time t, i_t

Flow for LSTM



Corresponding formulas:

$$i_t = \text{sigmoid}\left(W_{h,i}\, h_{t-1} + W_{x,i}\, x_t\right)$$

$$f_t = \text{sigmoid}\left(W_{h,f}\, h_{t-1} + W_{x,f}\, x_t\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh\left(W_{h,c}\, h_{t-1} + W_{x,c}\, x_t\right)$$

$$o_t = \text{sigmoid}\left(W_{h,o}\, h_{t-1} + W_{x,o}\, x_t\right)$$

$$h_t = o_t \odot \tanh\left(W_{c,h}\, c_t\right)$$

Learn via BPTT all the parameters for these gates

LSTMs are very popular (among RNN models).

GRUs were introduced to simplify LSTMs:

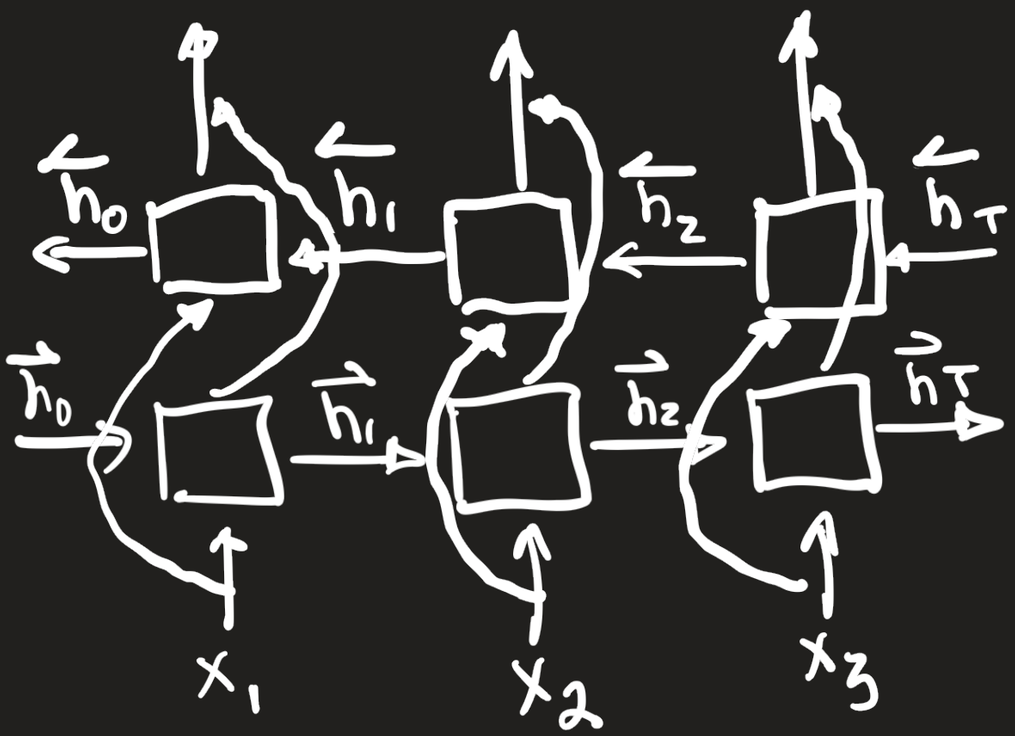
- have a hidden state, no cell state
- only have two gates: reset and update gate

Faster to train, but LSTMs are more popular

Bidirectional RNN

"Very ___, how are you?"

↑ complete w/ well, good, etc.



The "feature" can provide useful information in many tasks.

Bidirectional RNNs predict the current output $\hat{y}_t$ using the "past" and the "feature"

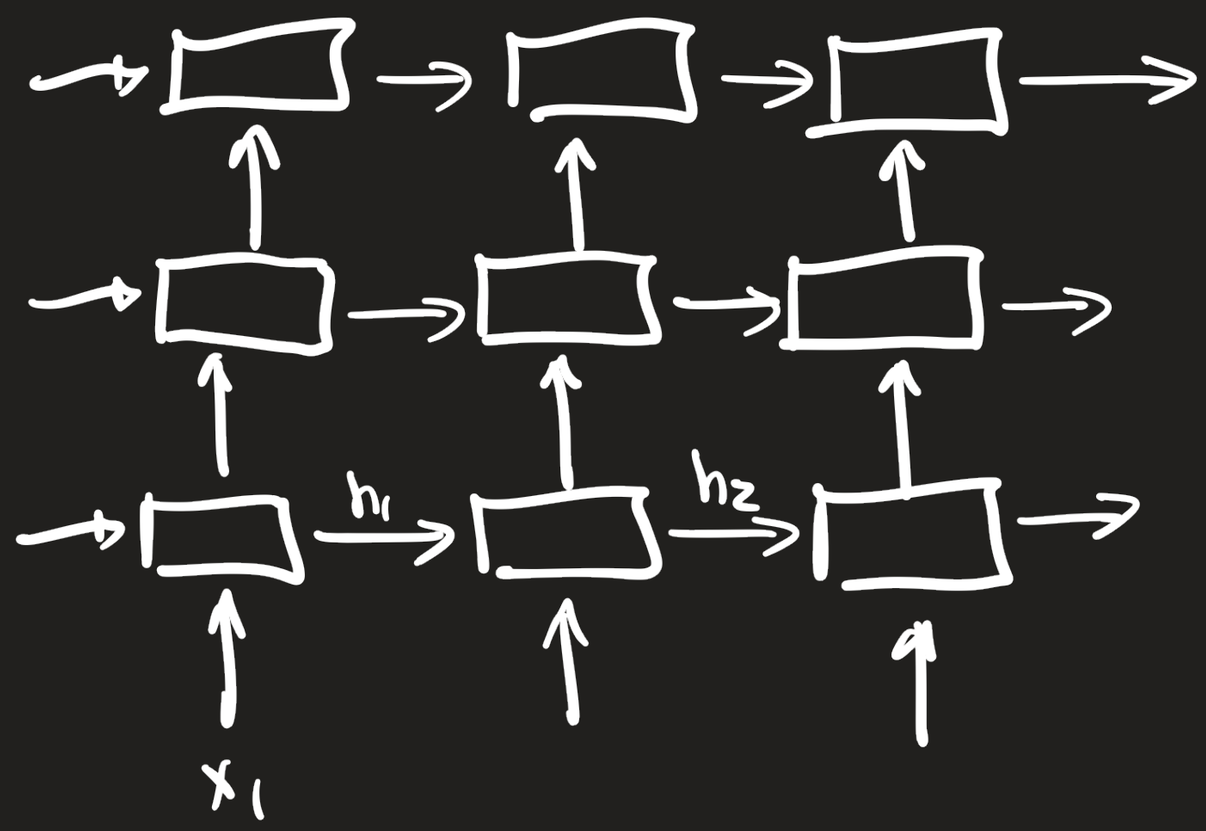
$$\overrightarrow{h}_t = f(x_t, \overrightarrow{h}_{t-1}, \overrightarrow{w}_h)$$

$$\overleftarrow{h}_t = f(x_t, \overleftarrow{h}_{t+1}, \overleftarrow{w}_h)$$

$$h_t = \begin{bmatrix} \overrightarrow{h}_t \\ \overleftarrow{h}_t \end{bmatrix}$$

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

Deep RNNs (typically used when "LSTMs" or "RNNs" are mentioned)



- train layer-wise, by computing all outputs from layer l so we have the inputs for layer $l+1$
- modify BPTT to account for dependencies between layers