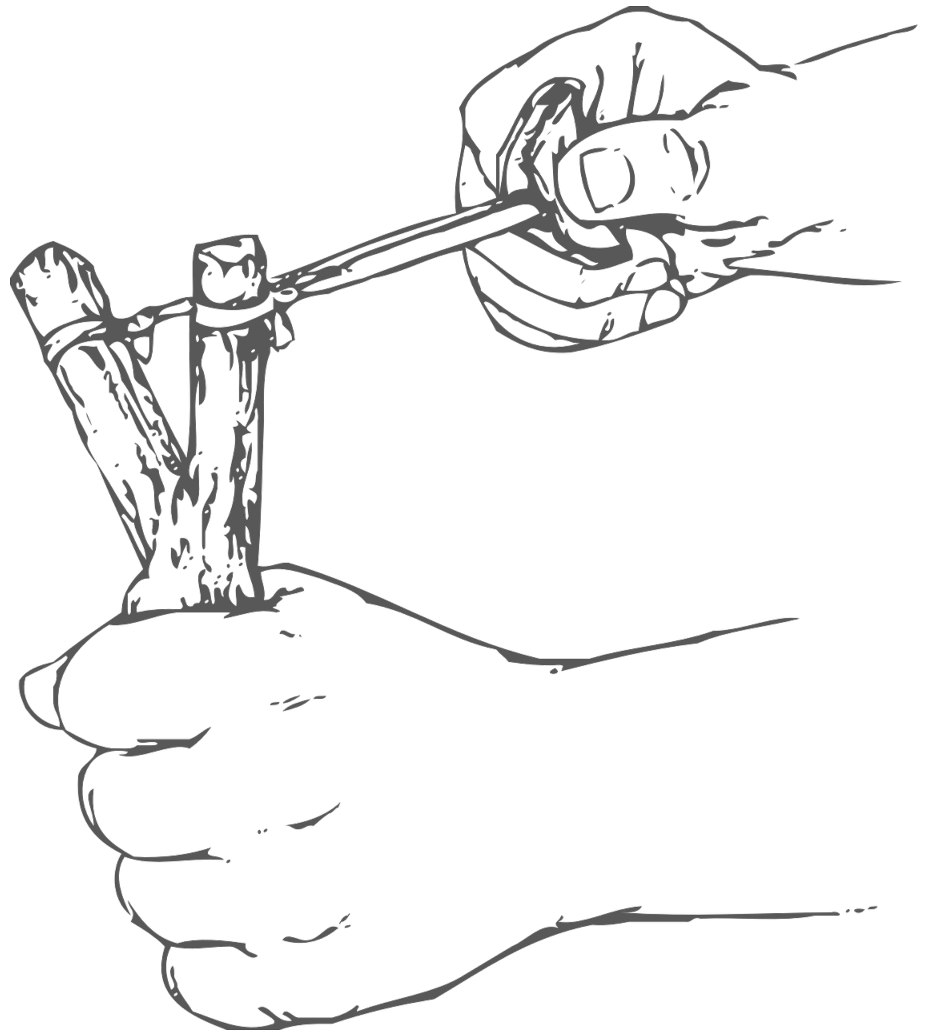
Neural·Pragmatic Natural Language Generation

N·P NLG

Learning goals



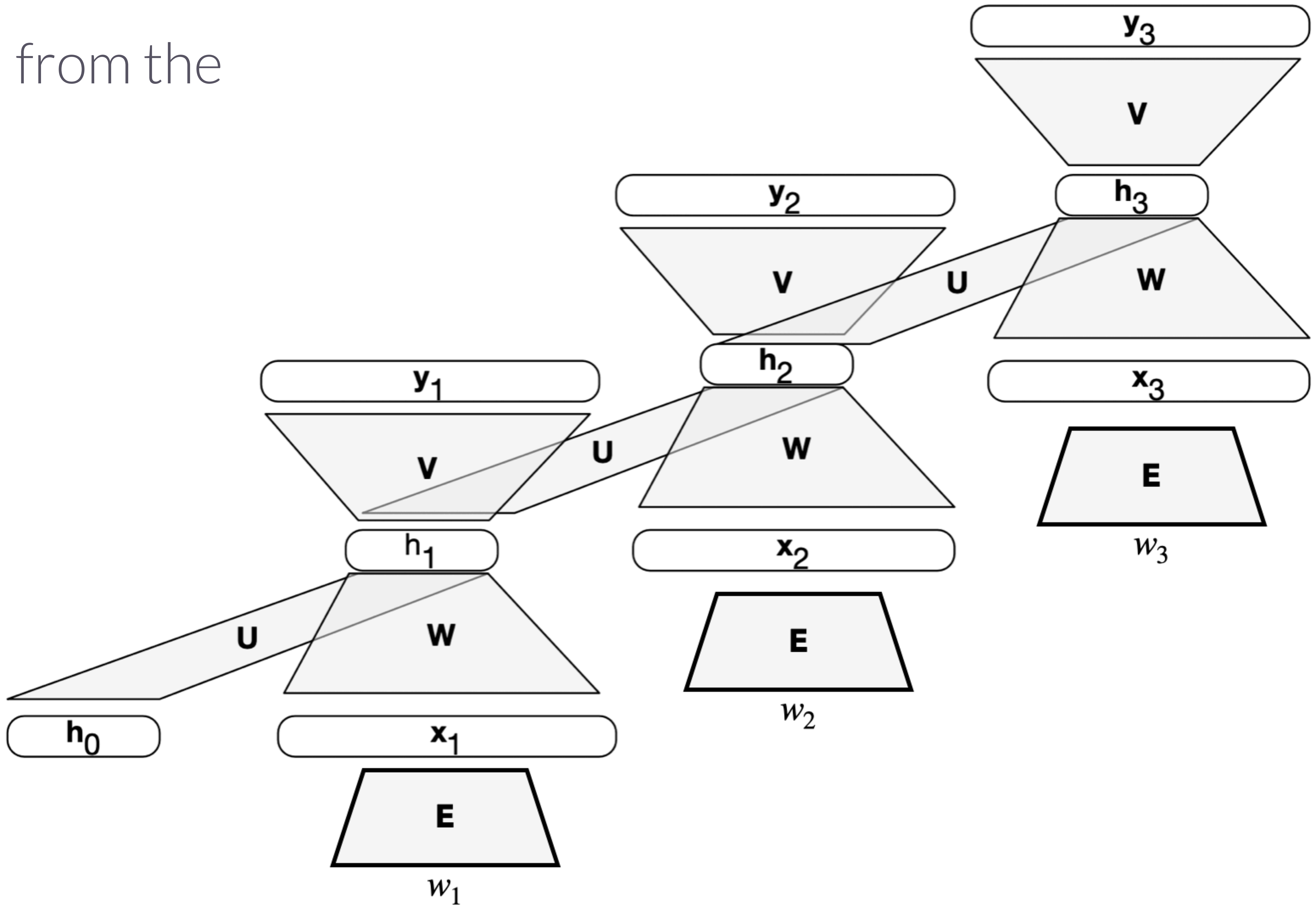
1. understand basic architecture of LSTMs
2. understand how LSTMs improve on RNNs
3. become able to use PyTorch's built-in modules for LMs
4. implement a character-level LSTM
5. learn about different **decoding schemes**
   a. pure & greedy sampling
   b. top-k & top-p sampling
   c. softmax sampling
   d. beam search
6. learn about different **training regimes**
   a. autoregressive training
   b. teacher forcing
   c. curriculum learning
   d. professor forcing

RNNs revisited

RNN-based language model

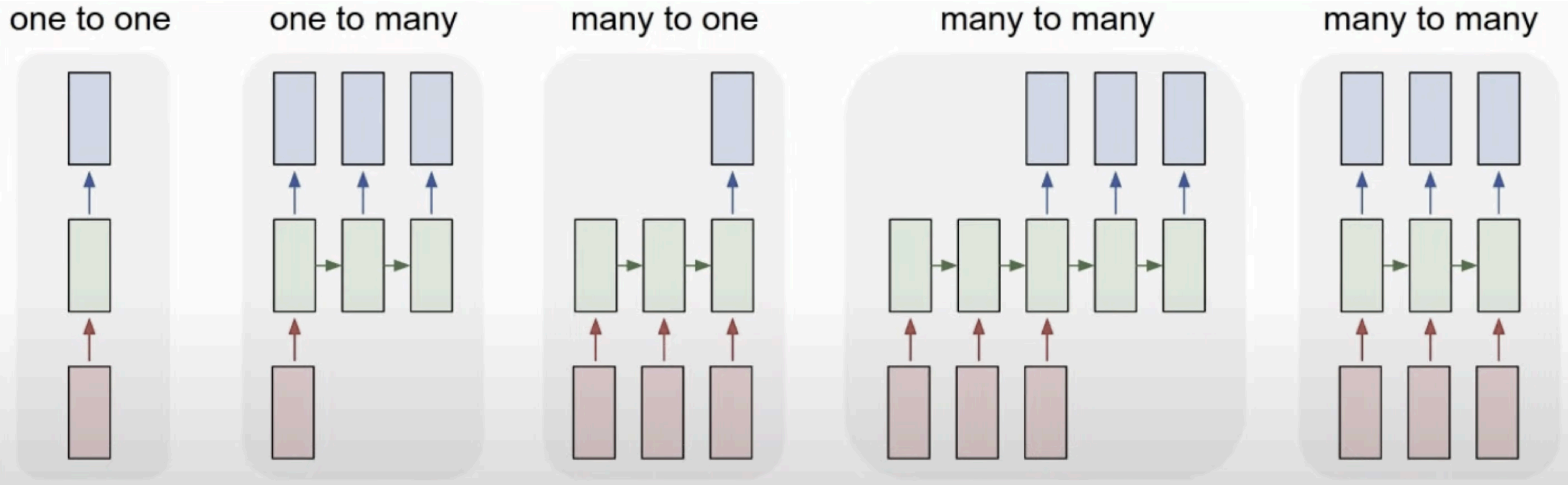
hidden layer as a “memory state”

- hidden layer is a “memory state”
- predictions (at each token) are derived from the hidden layer
  - next-word prediction
  - part-of-speech prediction
  - sentiment analysis
  - ...

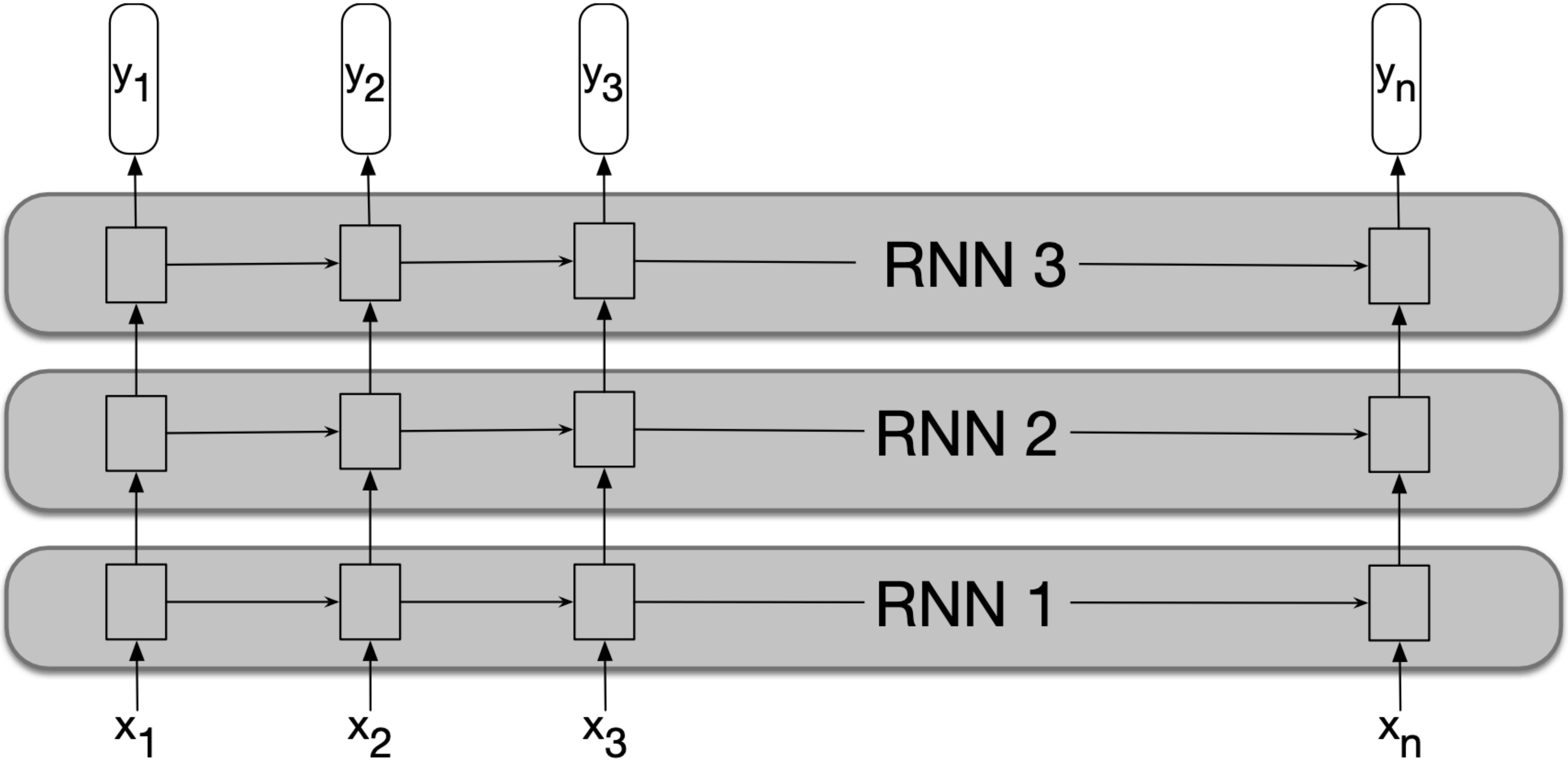


Different kinds of sequence processing models

sequence as input and/or (simultaneous) output

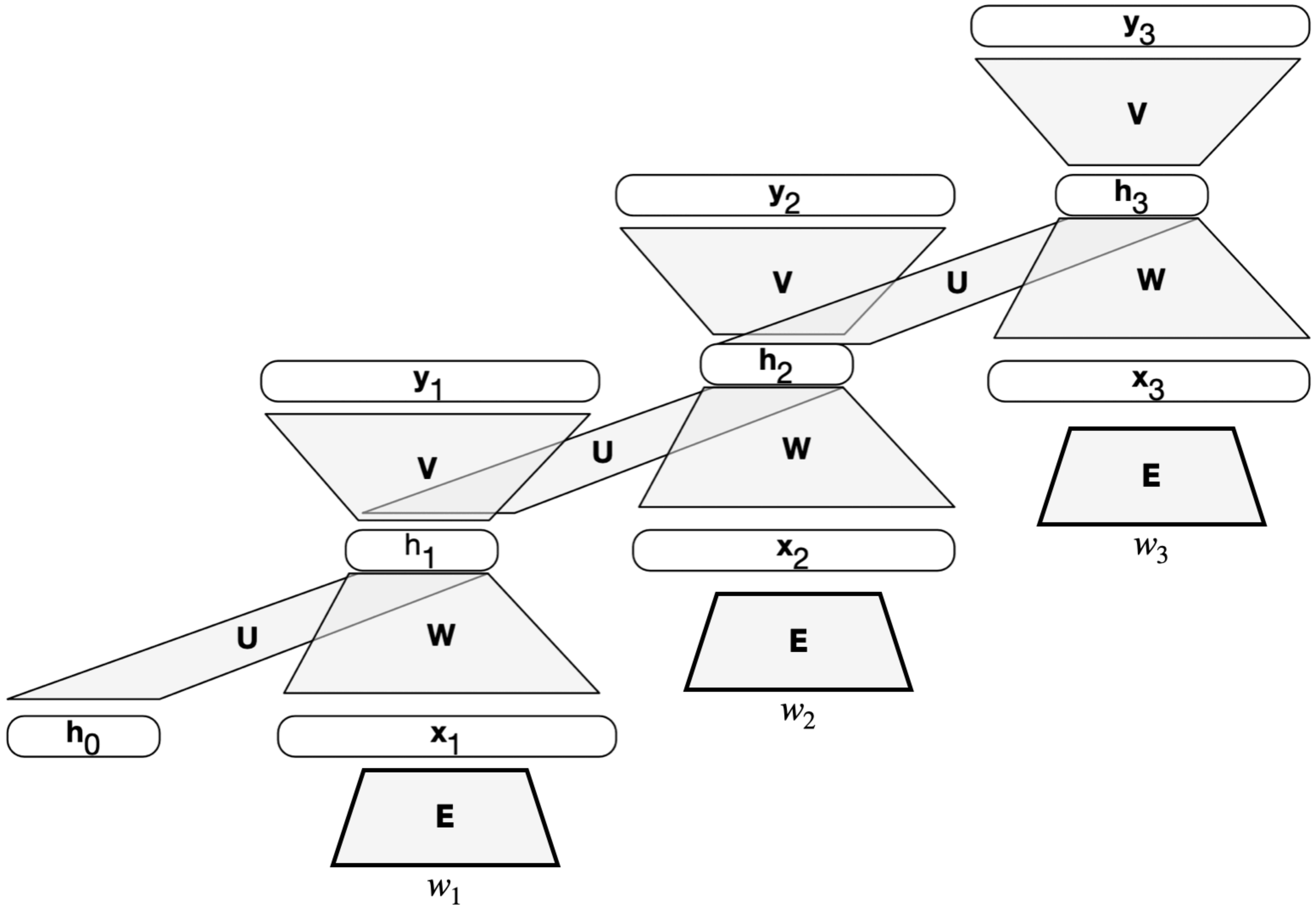


Stacked RNNs
multiple layers



Problems with RNNs

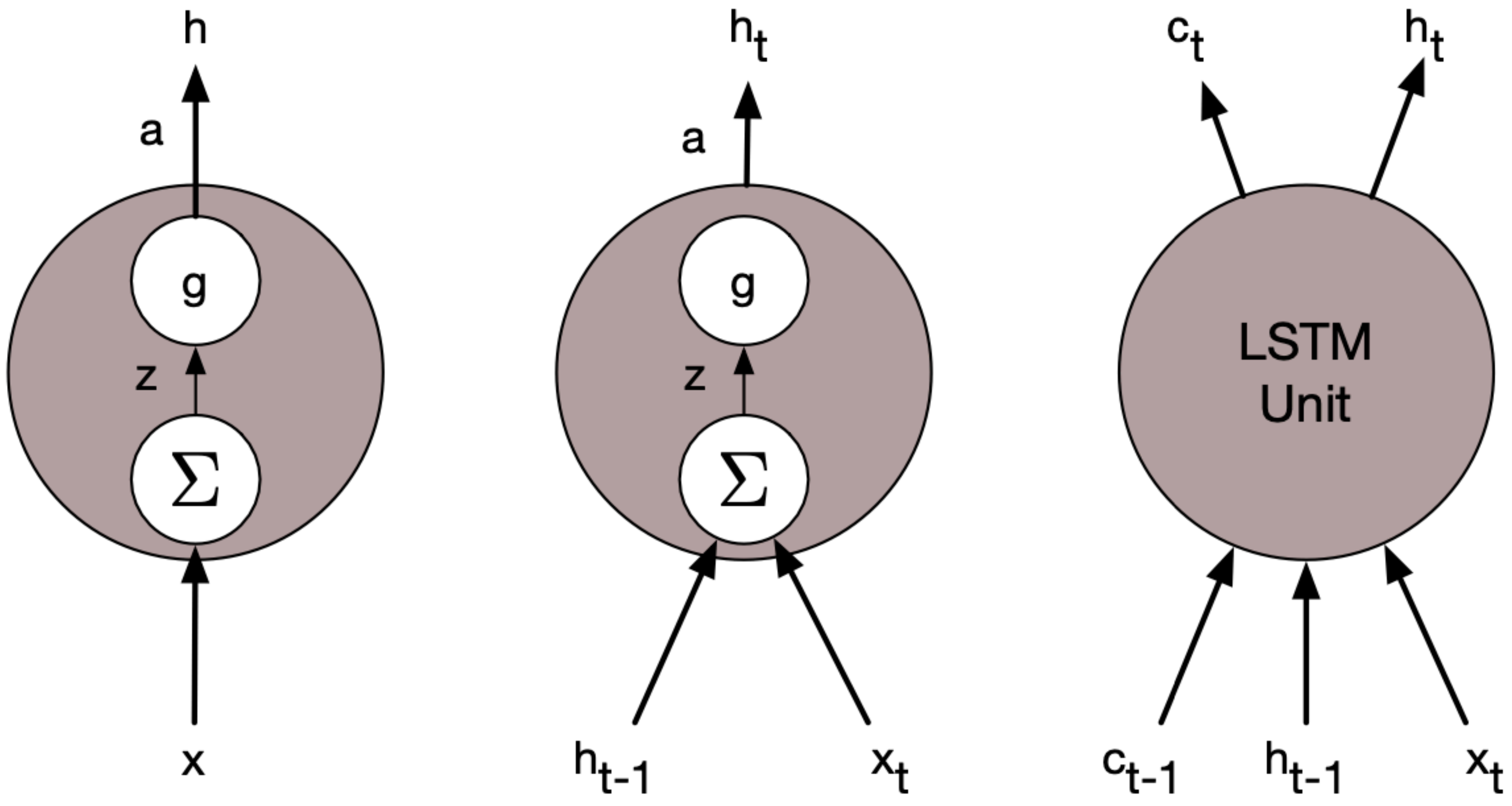
- conceptual problem
  - two-fold role of hidden state:
    - memory for past sequence
    - recommend what to do now
- technical problem
  - vanishing gradients for long past input
    - partial remedy: bidirectional RNNs



Long-Short Term Memory (LSTM) Models

Modular architectures
cells / units

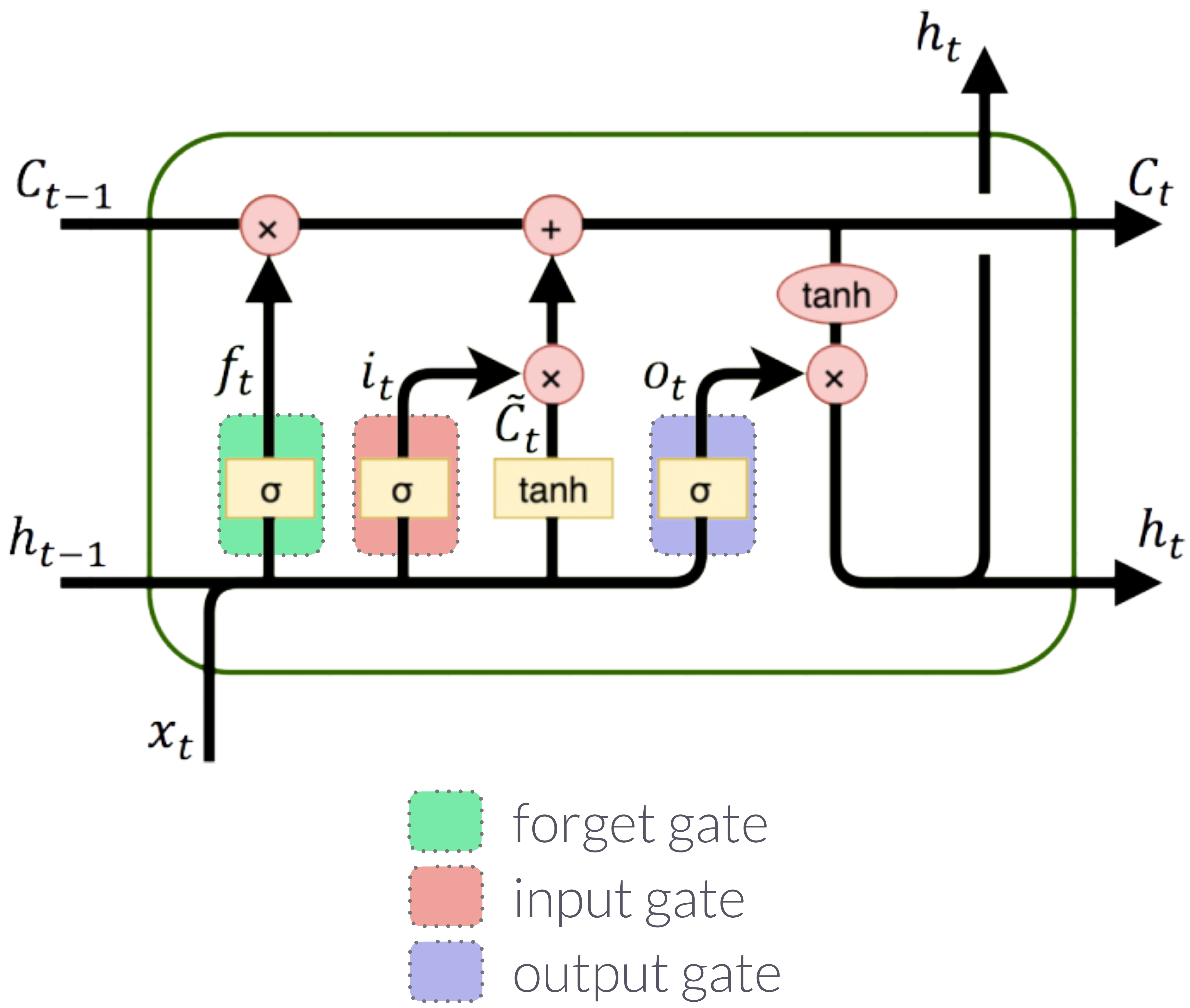
- common mapping
  - input to hidden state: $x \mapsto h$
    - variously referred to as encoding or embedding



MLP RNN LSTM

LSTM cell

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$



Decoding schemes

Decoding schemes

based on next-word probability $P(w_{i+1} \mid w_{1:i})$

- **pure sampling**
  - next word is sampled from next-word probability distribution: $w_{i+1} \sim P(\cdot \mid w_{1:i})$
- **greedy decoding**
  - next word is word with highest probability: $w_{i+1} = \arg\max_{w'} P(w' \mid w_{1:i})$
- **softmax sampling**
  - next word is sampled from softmax of next-word probability distribution: $w_{i+1} \sim SM_{\alpha}\left(P(\cdot \mid w_{1:i})\right)$
- **top-k sampling**
  - next word is sampled from next-word prob. distribution after restricting to the k most likely words
- **top-p sampling**
  - next word is sampled from next-word prob. distribution after restricting to the smallest set of the most likely words which together comprise at least next-word probability p
- **beam search**
  - see blackboard

Training regimes for LMs

Training regimes

- teacher forcing
  - LM is fed true word sequence
  - training signal is next-word assigned to true word
- autoregressive training (aka free-running mode)
  - LM autoregressively generates a sequence
  - training signal is next-word probability assigned to true word
- curriculum learning (aka scheduled sampling)
  - combine teacher-forced and autoregressive training
  - start with mostly teacher forcing, then increase amount of autoregressive training
- professor forcing
  - combines teacher forcing with adversarial training
  - generative adversarial network GAN is trained to discriminate (autoregressive) predictions from actual data
  - LM is trained to minimized this discriminability
- decoding-based
  - use prediction function (decoding scheme) to optimize based on *actual* output