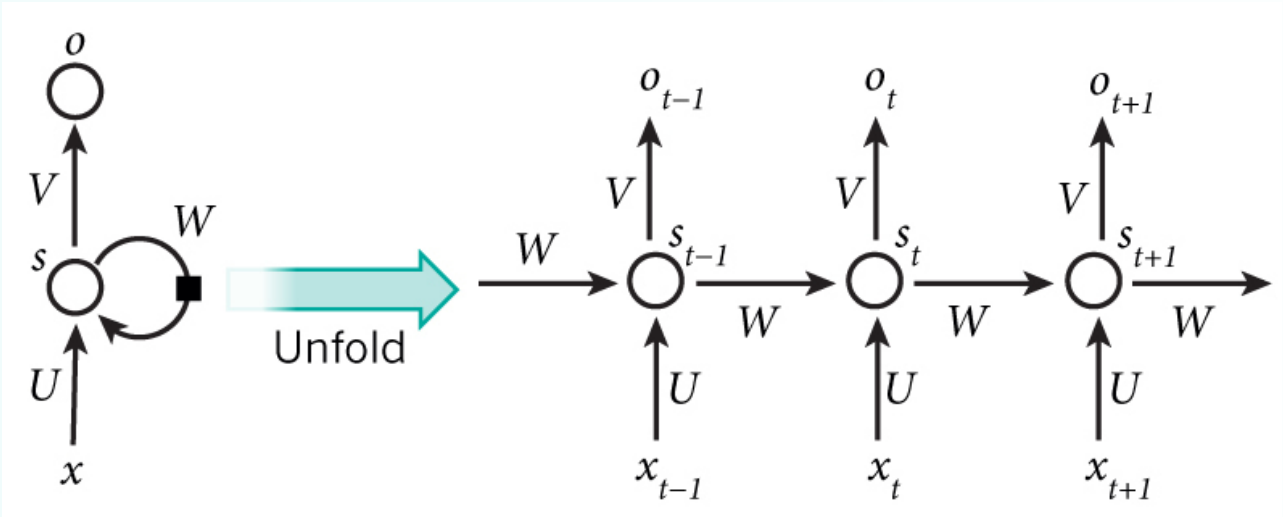
What is Recurrent Neural Networks (RNN)?

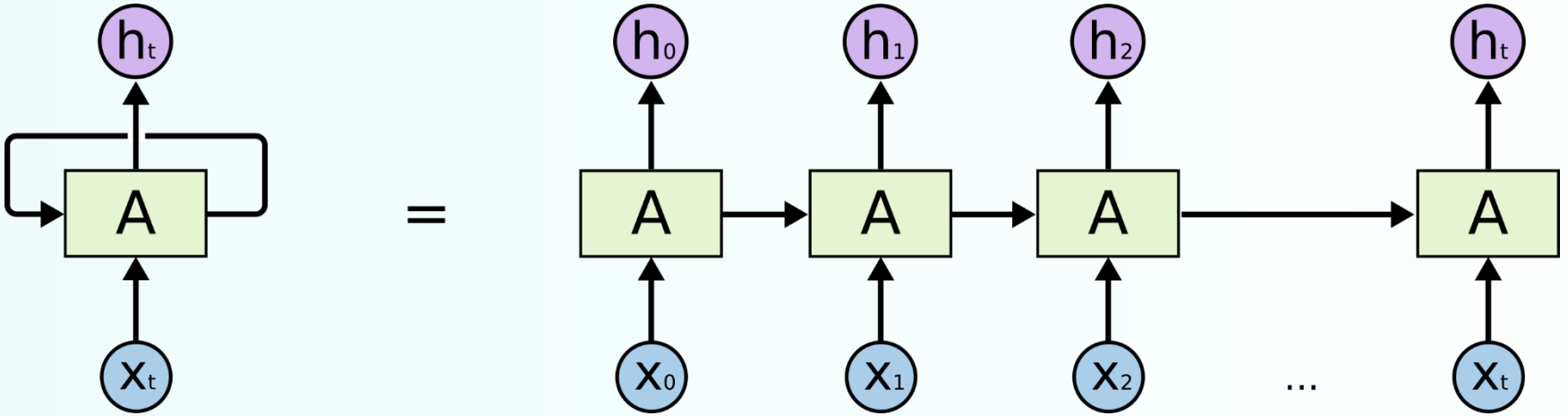
RNN (Recurrent Neural Network) is a type of artificial neural network designed to process sequential data. Unlike traditional feedforward neural networks, RNNs have a feedback loop, enabling them to handle data that changes over time or has a sequential nature. They operate by utilizing the current input and the hidden state from the previous step at each iteration.

Recurrent

- RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output depending on the previous values.
- RNNs have a “memory” which captures information about what has been calculated so far.
- In theory RNNs can make use of information in arbitrarily long sequences.

Recurrent Neural Network





Hidden Units

- The hidden state s_t represents the memory of the network.
- s_t captures information about what happened in all the previous time steps.
- The output o_t is calculated solely based on the memory at time t.
- s_t typically can't capture information from too many time steps ago.
- Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters (U, V, W above) across all steps.
- This reflects the fact that we are performing the same task at each step, just with different inputs. This greatly reduces the total number of parameters we need to learn.

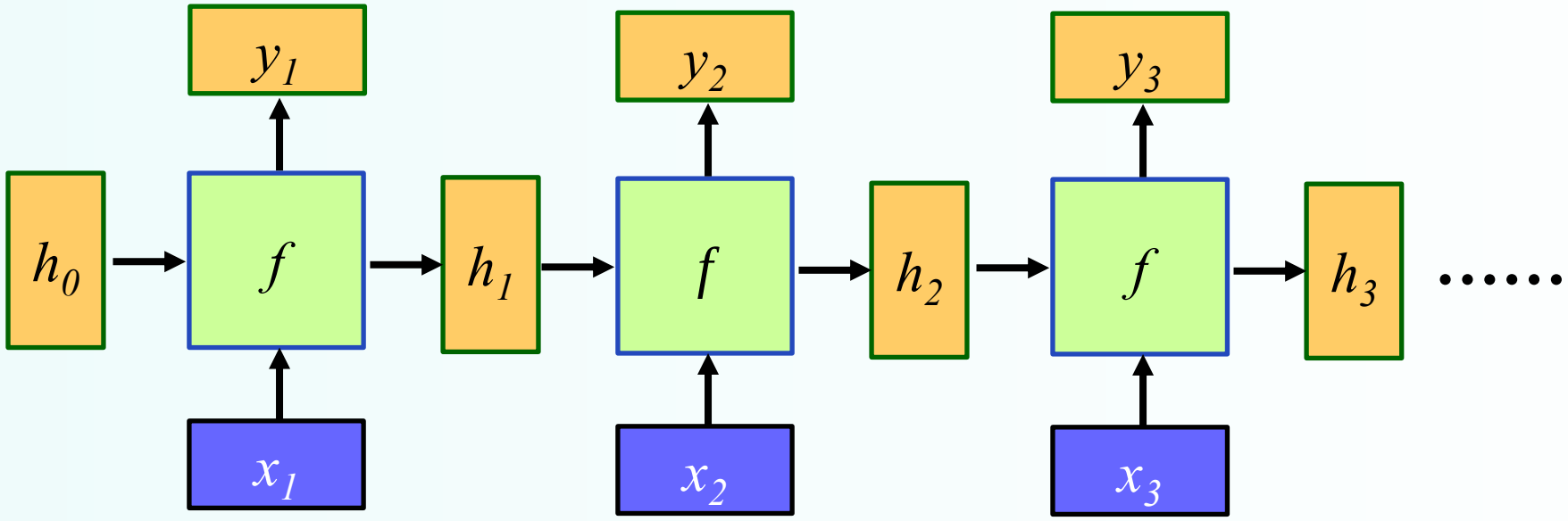
RNN Pro/Cons

- RNN **Advantages**:
  - Can process input of any length
  - Model size doesn't increase for longer input
  - Computation for step *t* can (in theory) use information from many steps back
  - Weights are shared across timesteps → representations are shared
- RNN **Disadvantages**:
  - Recurrent computation is slow
  - In practice, difficult to access information from many steps back

How do RNN reduce complexity?

Given function f: $h', y = f(h, x)$

h and h' are vectors with the same dimension

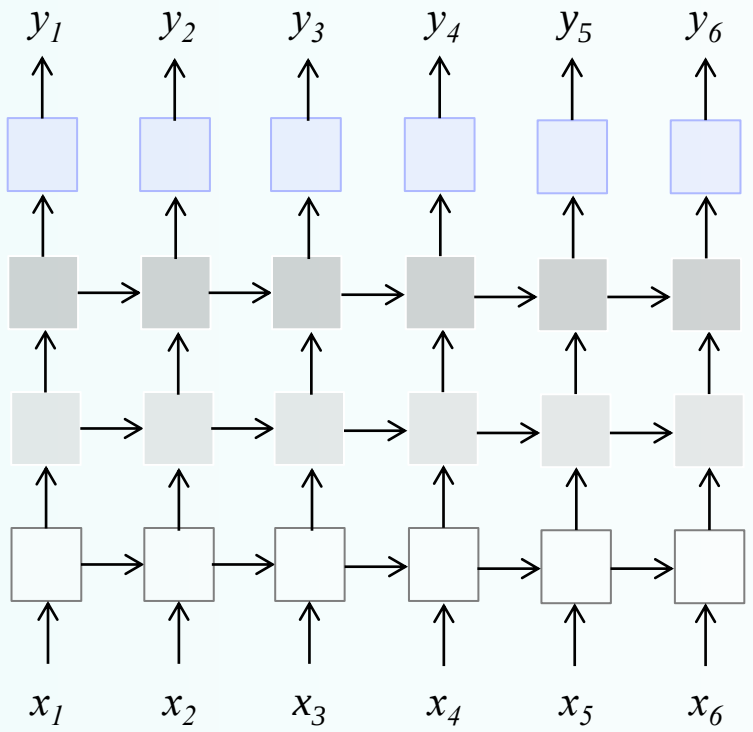


No matter how long the input/output sequence is, we only need **one function** f.

If f's are different, then it becomes a feedforward NN.

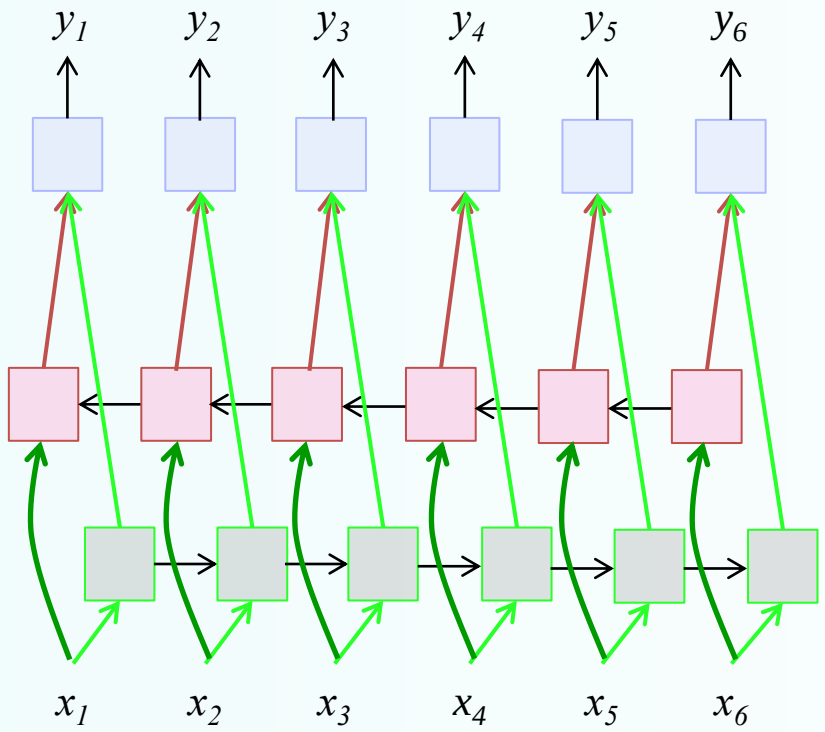
Deep RNN

RNNs with multiple hidden layers



Bidirectional RNN

RNNs can process the input sequence in both forward and reverse direction



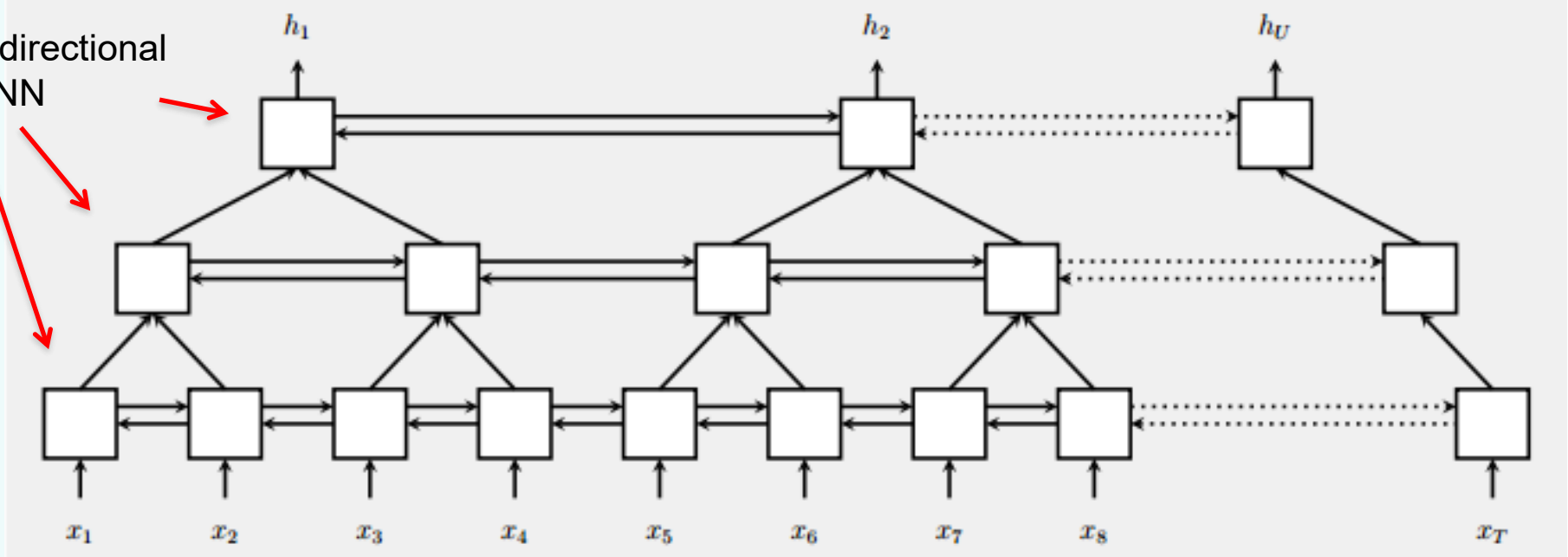
Popular in speech recognition and NLP

Pyramid RNN

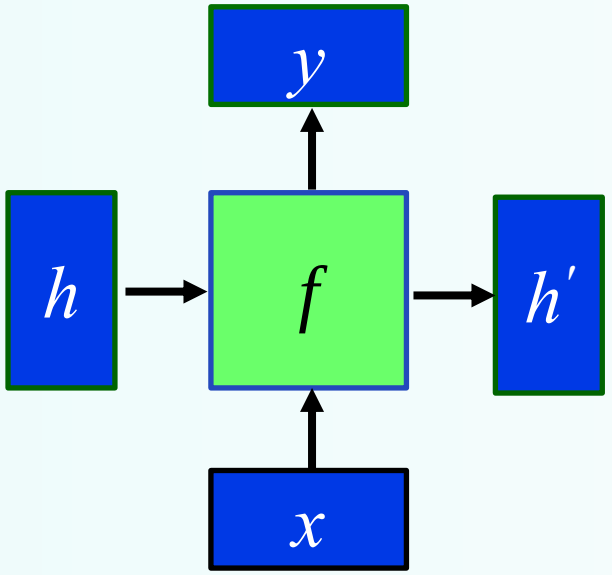
Significantly speed up training

Reducing the number of time steps

Bidirectional RNN



Vanilla RNN



$$h' = \sigma(W h + U x + b)$$

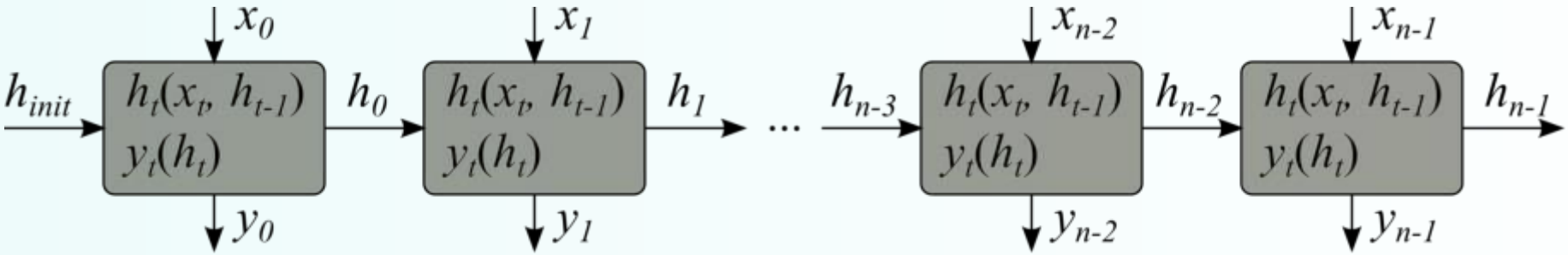
$$y = \sigma(V h')$$

Note, y is computed from h'

Vanilla RNN

$$h_t = \sigma(\boldsymbol{W}_{ih}\, x_t + \boldsymbol{U}\, h_{t-1} + b)$$
$$y_t = \sigma(\boldsymbol{W}_0\, h_t)$$

Unfolding the network in time



RNN Training

RNN Training

Target: obtain the network parameters that optimize the cost function
Cost functions: log loss, mean squared root error etc…

Tasks:

- For **each timestep** of the input sequence x predict output y (synchronously)
- For the input sequence x predict the **scalar value** of y (e.g., at end of sequence)
- For the input sequence x of length L_x generate the output sequence y of different length L_y

Methods

Backpropagation:

- Reliable and controlled convergence
- Supported by most of ML frameworks

Others:

- Evolutionary methods, expectation maximization, non-parametric methods, particle swarm optimization

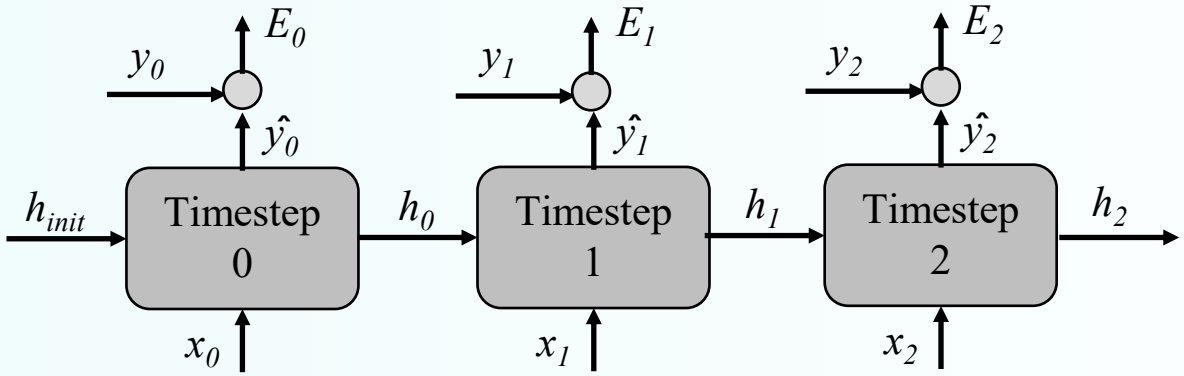
Back-propagation Through Time

$$h_t = \sigma(\boldsymbol{W}_{ih}\, x_t + \boldsymbol{U}\, h_{t-1} + b)$$
$$y_t = \sigma(\boldsymbol{W}_0\, h_t)$$

Cross Entropy Loss: $E(\hat{y}, y) = -\sum_t y \log \hat{y}_t$

1. *Unfold the network.*
2. *Repeat* for the train data:
    1. Given some input sequence $\boldsymbol{x}$
    2. *For t in* 0, *N*-1*:*
        1. *Forward-propagate*
        2. Initialize hidden state to the past value $\boldsymbol{h}_{t-1}$
        3. Obtain output sequence $\boldsymbol{y}$
        4. Calculate error $\boldsymbol{E}(\boldsymbol{y}, \boldsymbol{y})$
        5. *Back-propagate* error across the unfolded network
        6. Average the weights
        7. Compute next hidden state value $\boldsymbol{h}_t$

Back-propagation Through Time



Apply chain rule: $$\frac{\partial h_2}{\partial h_0} = \frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial h_0}$$

For time 2: $$\frac{\partial E_2}{\partial \theta} = \sum_{k=0}^{2} \frac{\partial E_2}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_k} \cdot \frac{\partial h_k}{\partial \theta}$$

$\boldsymbol{\theta}$ - Network parameters

Back-propagation Through Time