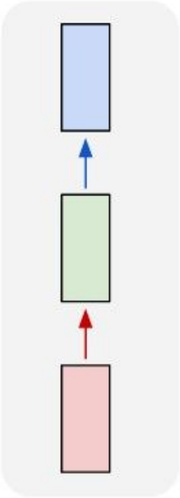
# Lecture 10: Recurrent Neural Networks

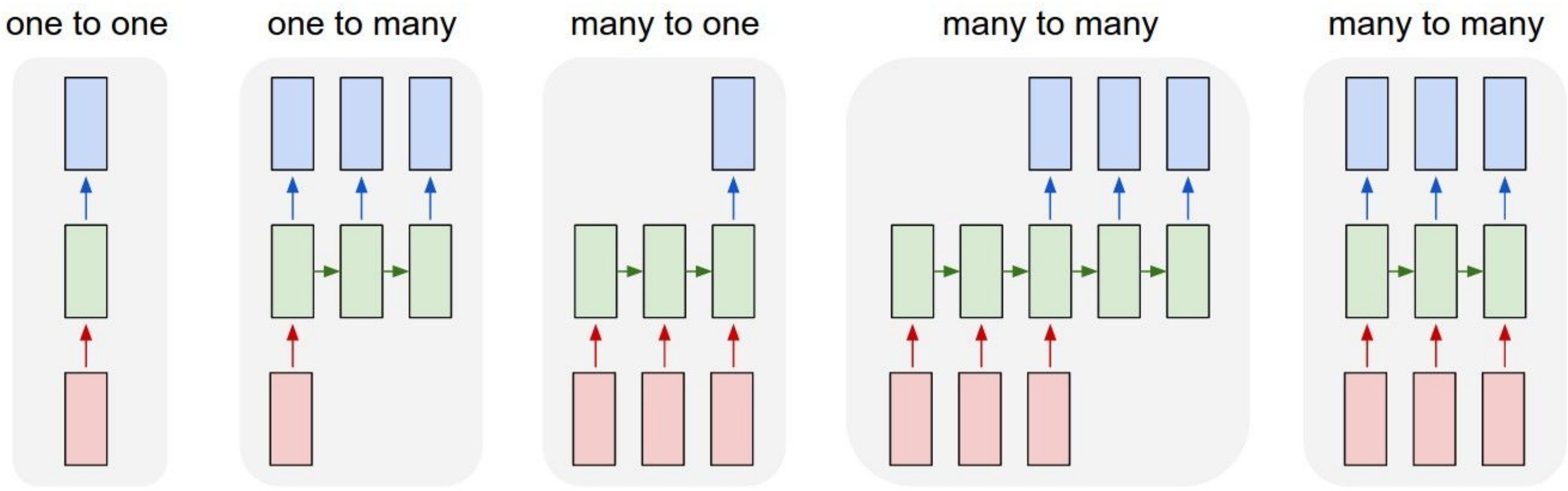
# “Vanilla” Neural Network

one to one



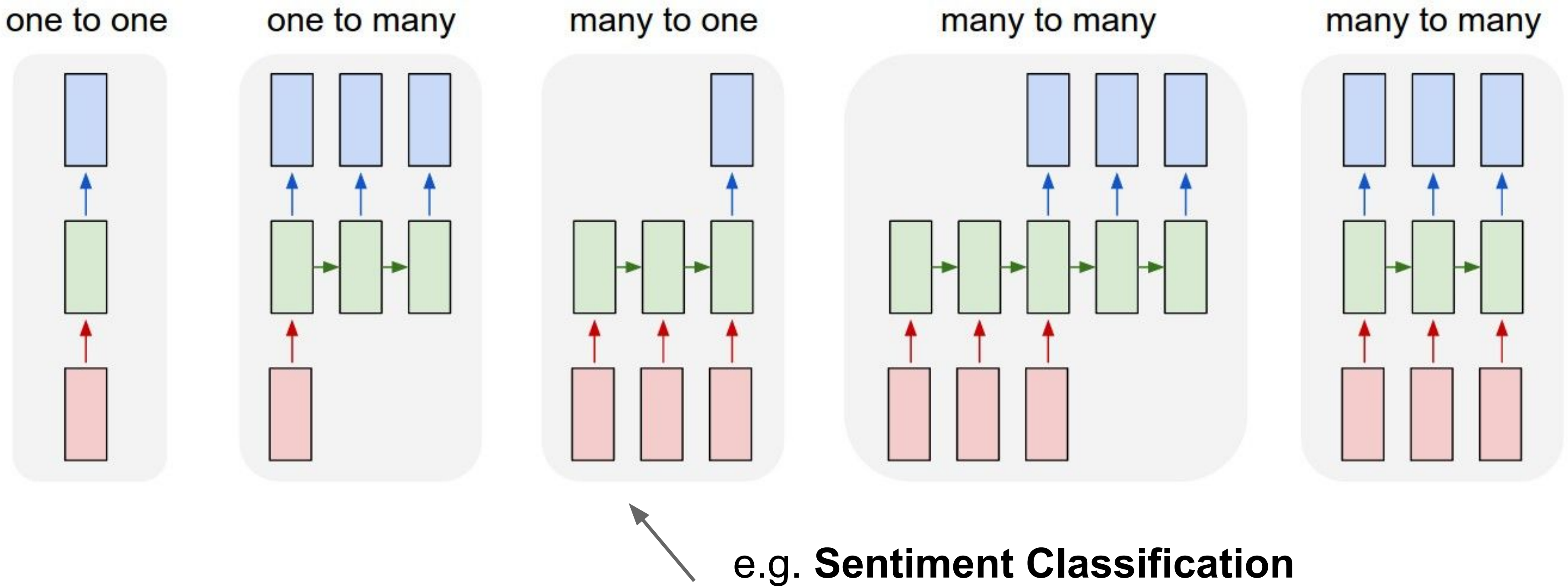
Vanilla Neural Networks

# Recurrent Neural Networks: Process Sequences



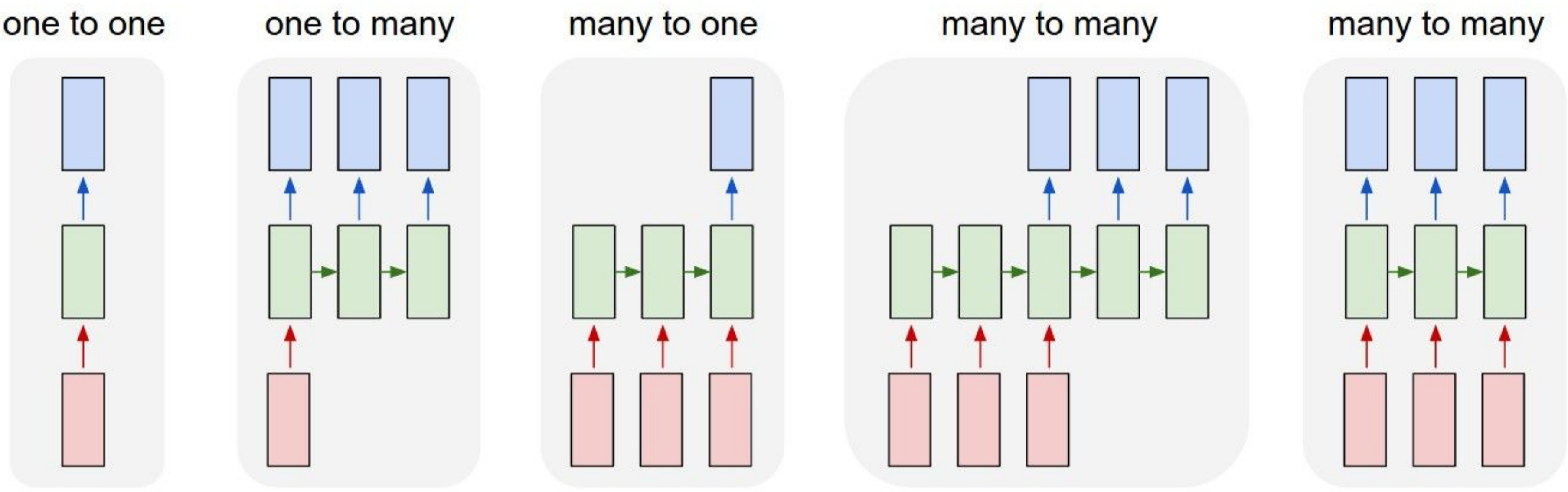

e.g. **Image Captioning**
image -> sequence of words

# Recurrent Neural Networks: Process Sequences



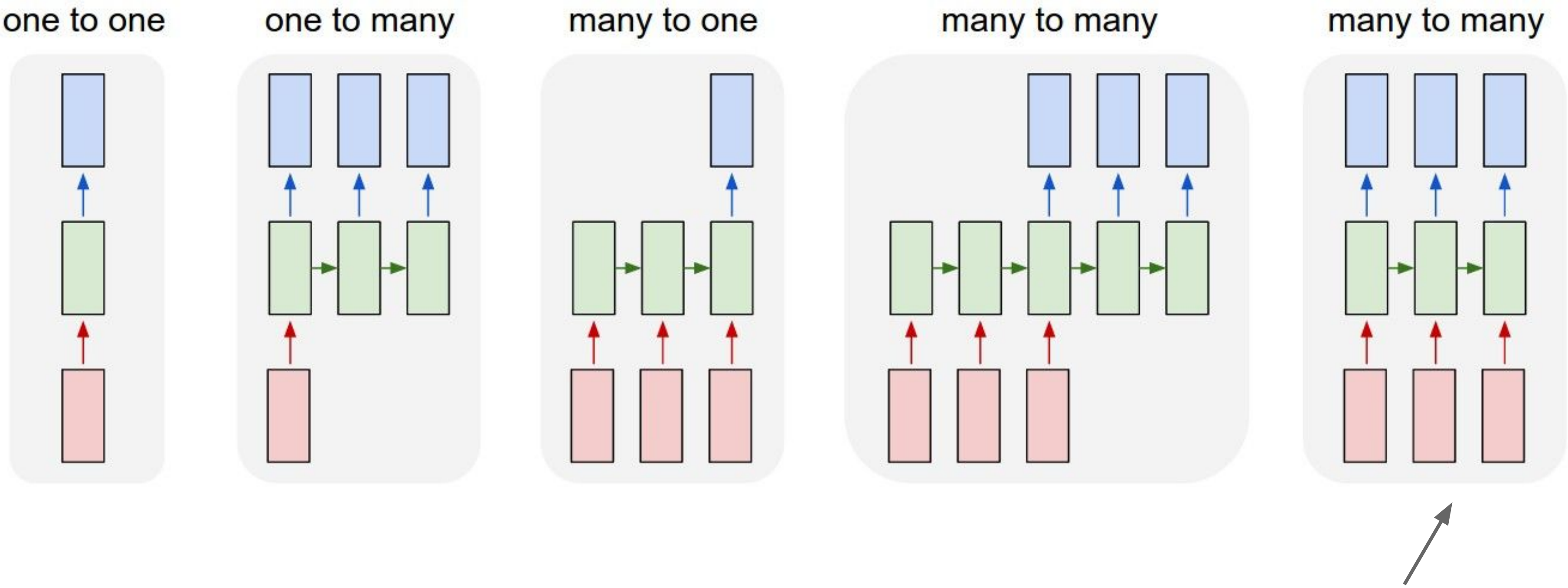

e.g. **Sentiment Classification**
sequence of words -> sentiment

# Recurrent Neural Networks: Process Sequences



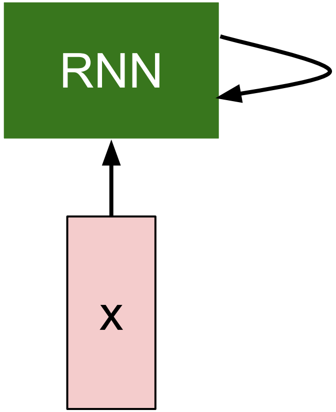

e.g. **Machine Translation**
seq of words -> seq of words

# Recurrent Neural Networks: Process Sequences

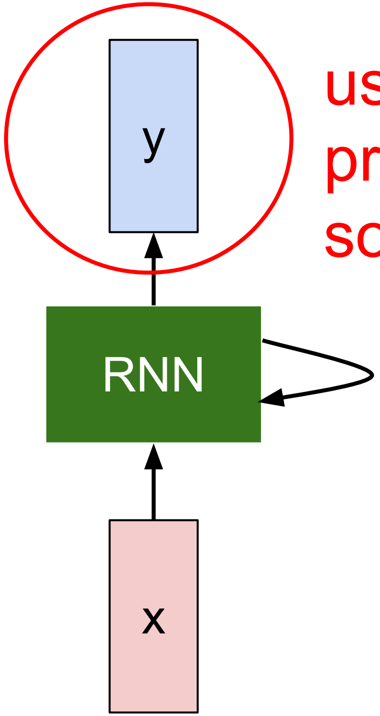



e.g. **Video classification on frame level**

# Recurrent Neural Network



# Recurrent Neural Network




usually want to predict a vector at some time steps

# Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

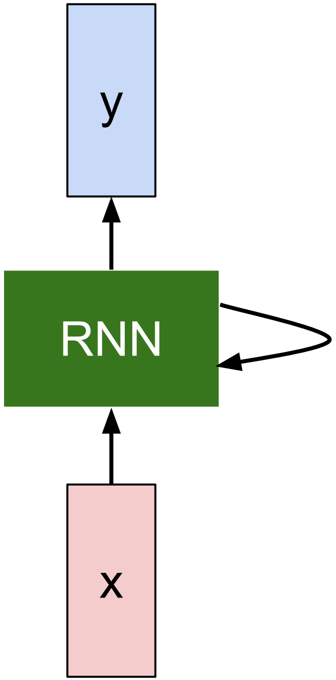
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W

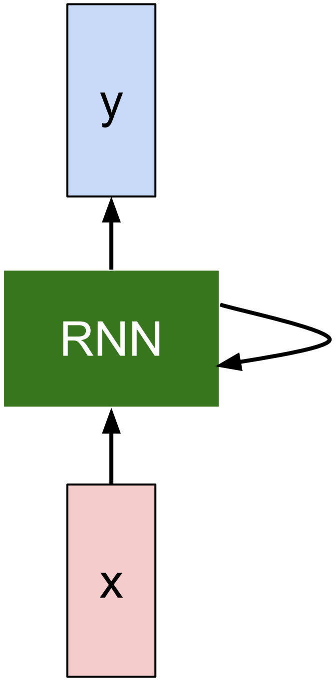


# Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

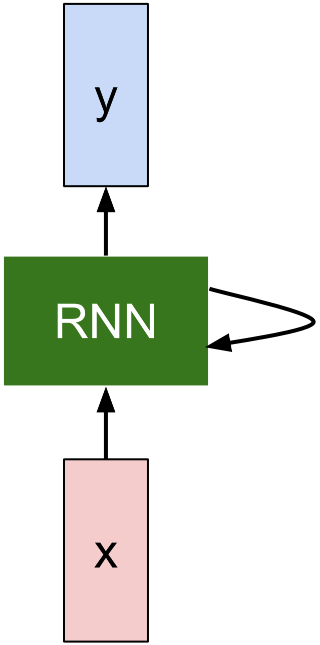
$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.



# (Vanilla) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



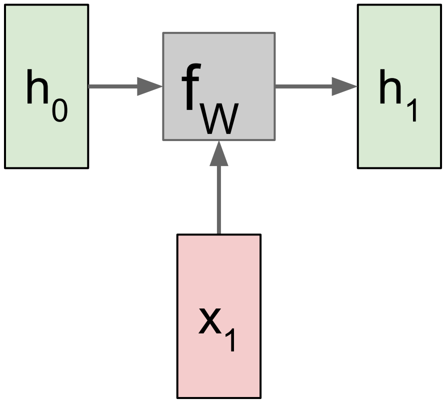

$$h_t = f_W(h_{t-1}, x_t)$$

$$\downarrow$$

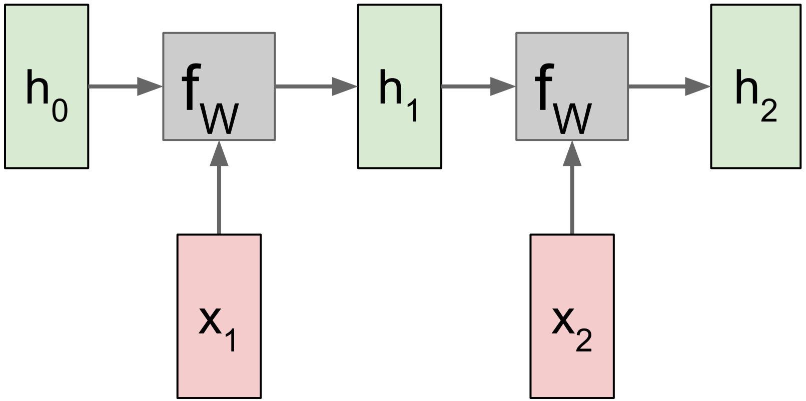
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

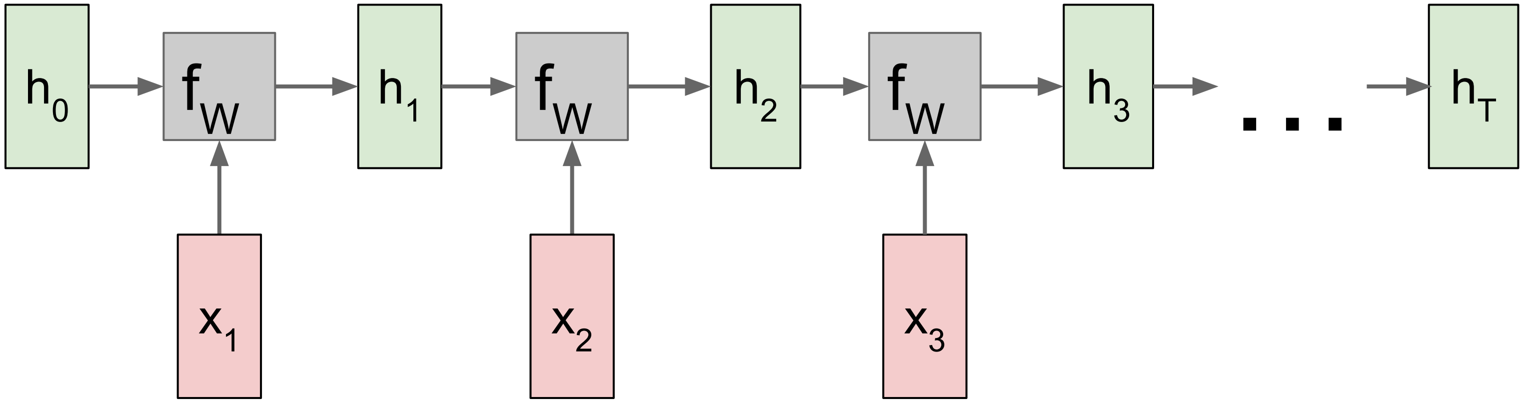
# RNN: Computational Graph



# RNN: Computational Graph

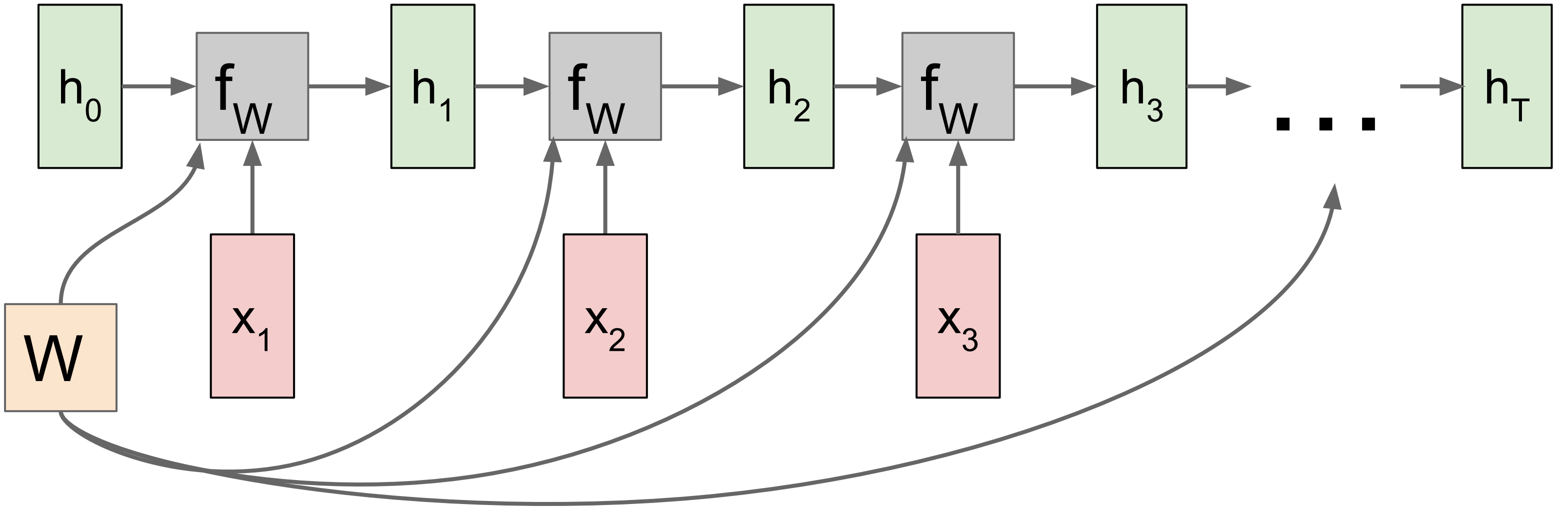


# RNN: Computational Graph

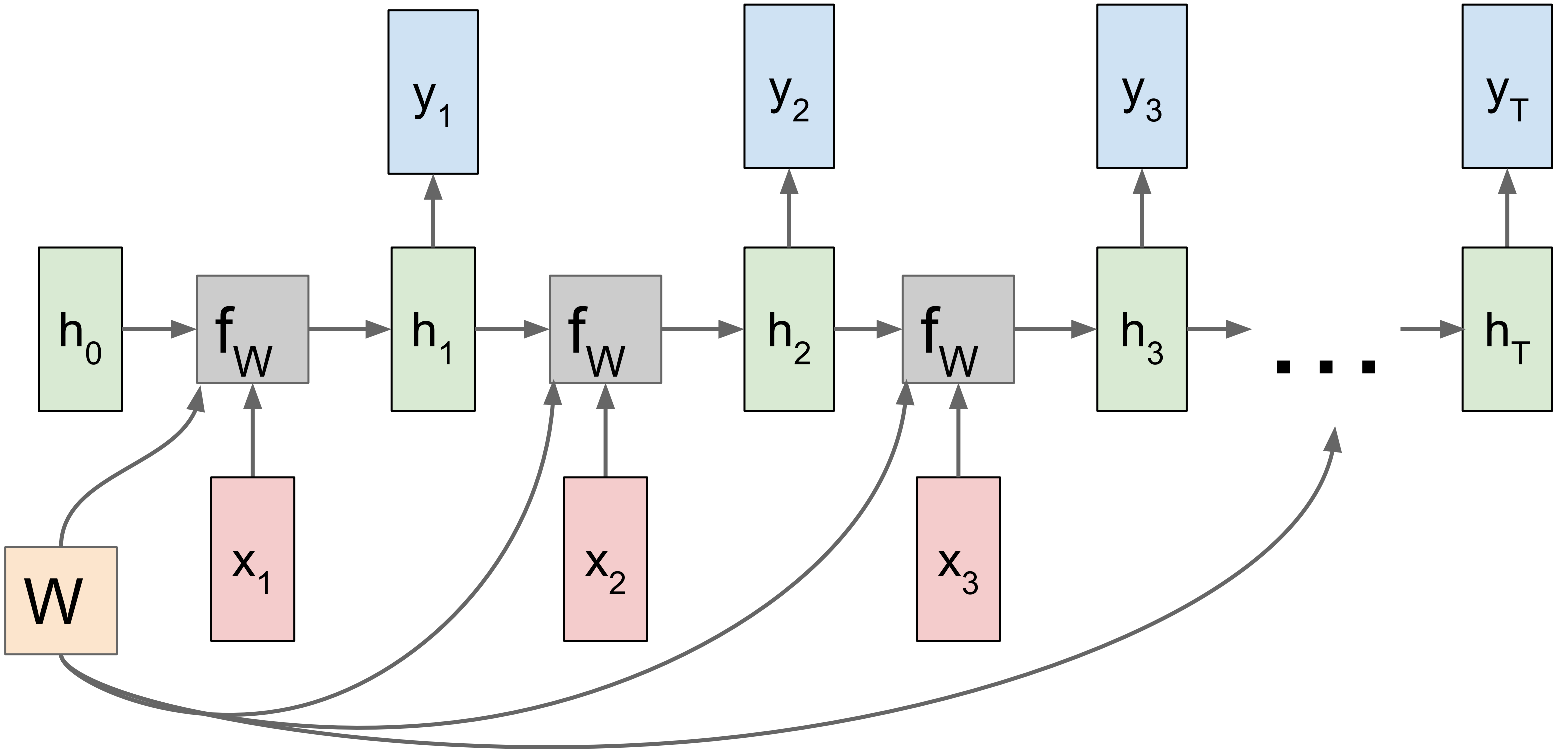


# RNN: Computational Graph

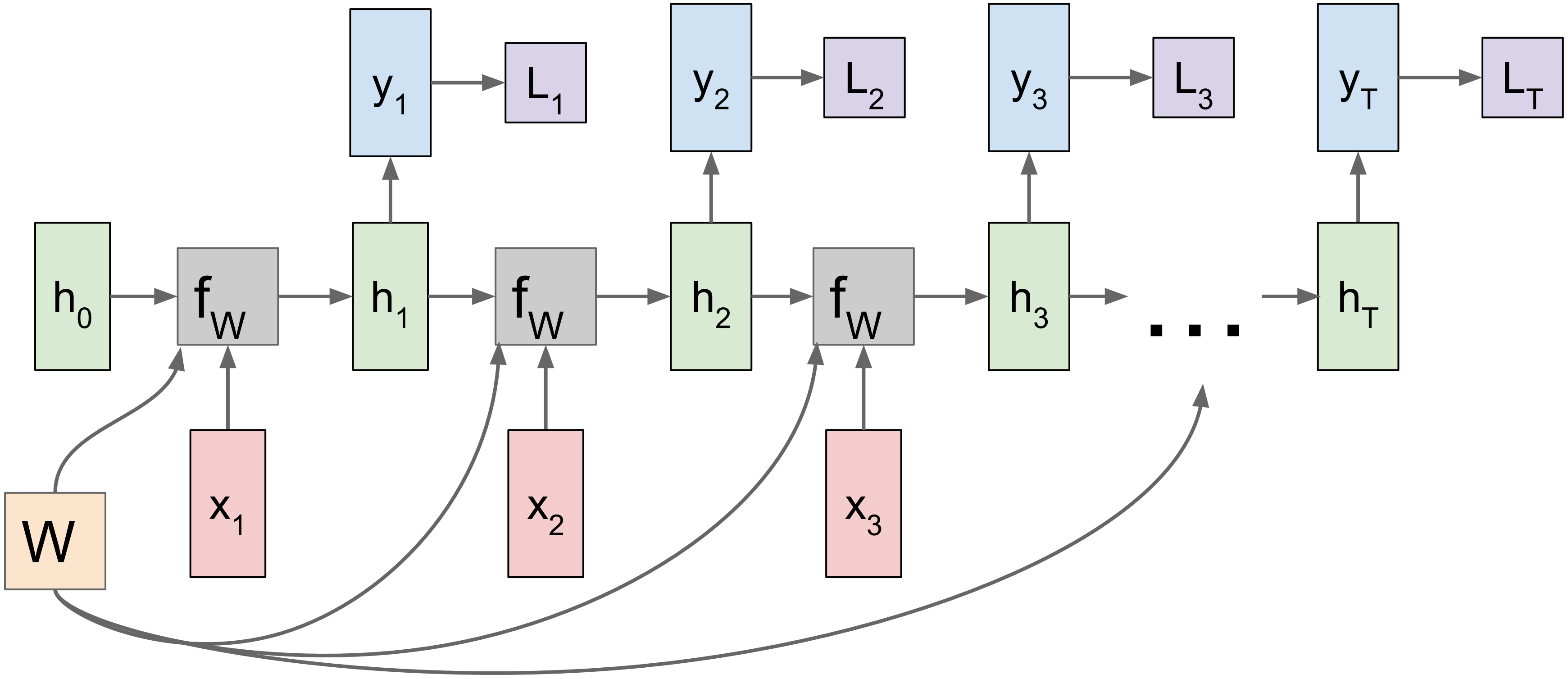
Re-use the same weight matrix at every time-step



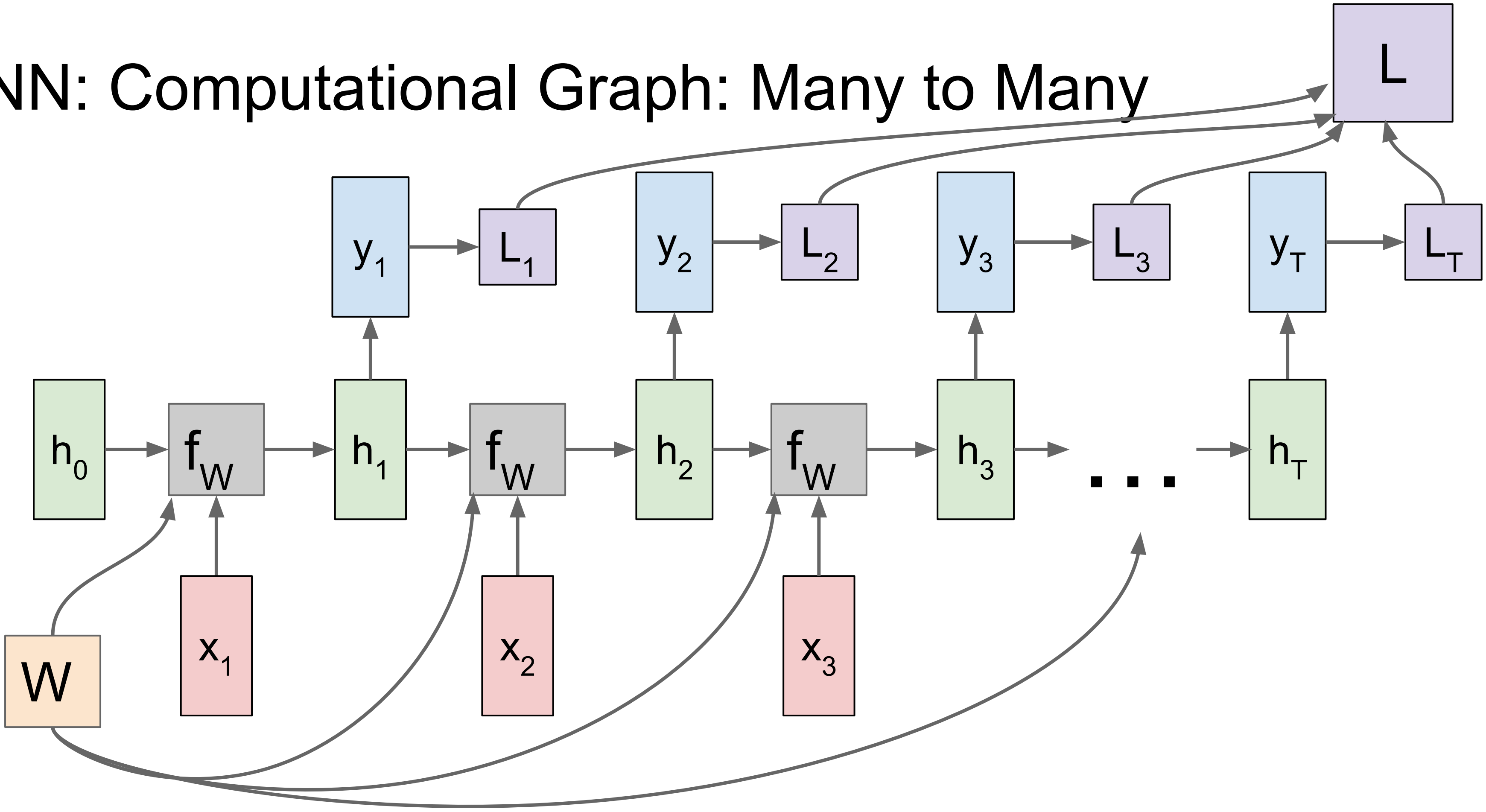
# RNN: Computational Graph: Many to Many



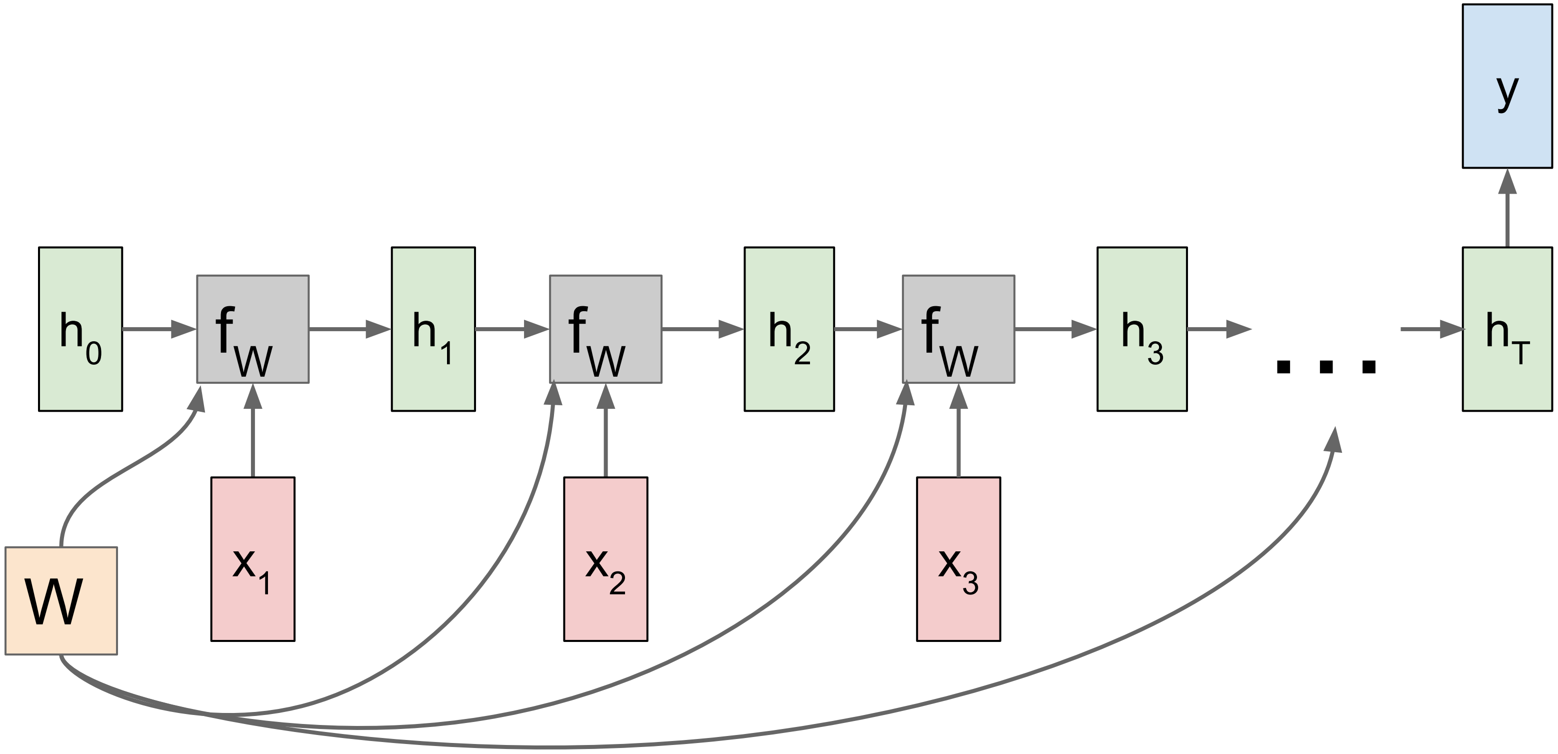
# RNN: Computational Graph: Many to Many



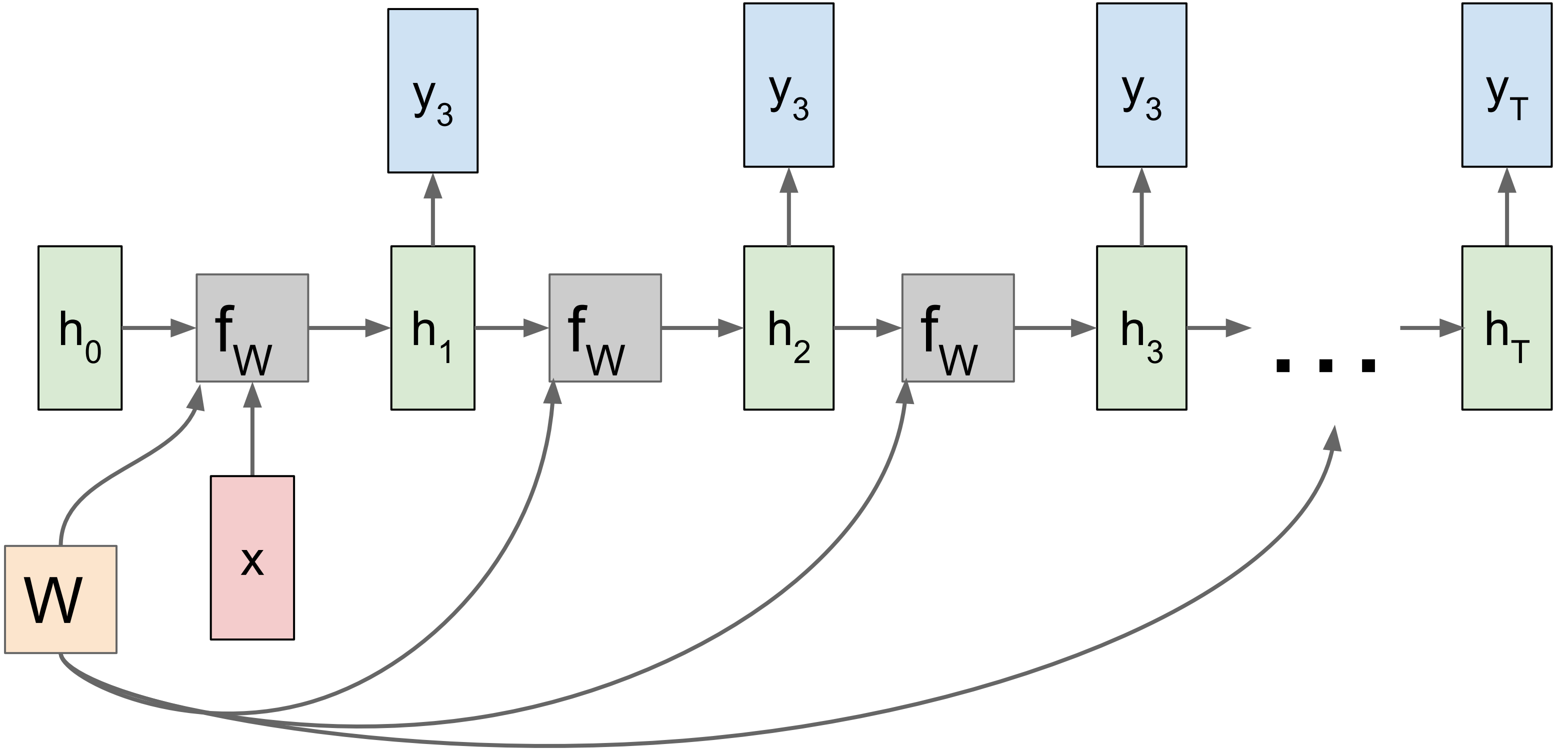
# RNN: Computational Graph: Many to Many



# RNN: Computational Graph: Many to One

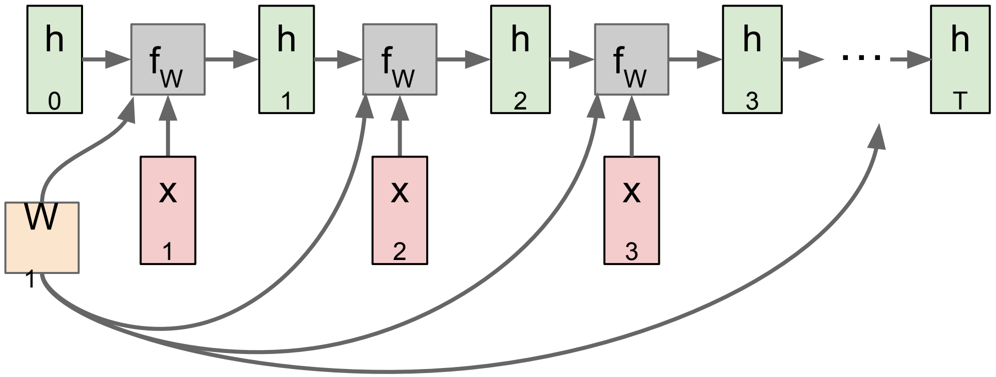


# RNN: Computational Graph: One to Many



# Sequence to Sequence: Many-to-one + one-to-many

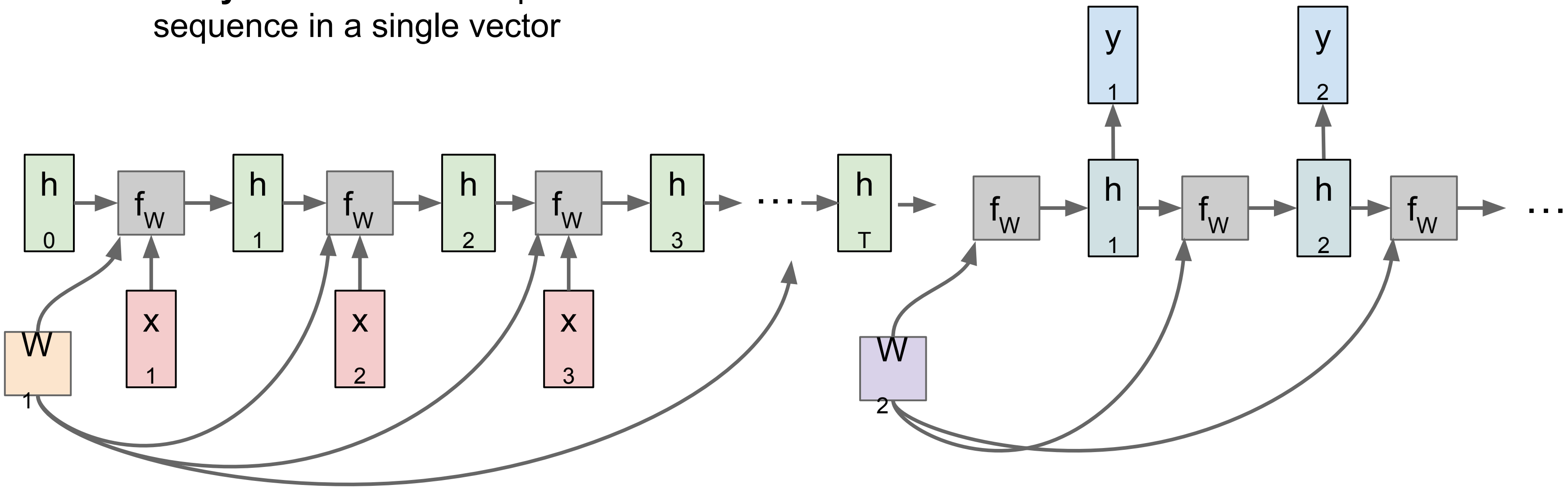
**Many to one**: Encode input sequence in a single vector



# Sequence to Sequence: Many-to-one + one-to-many

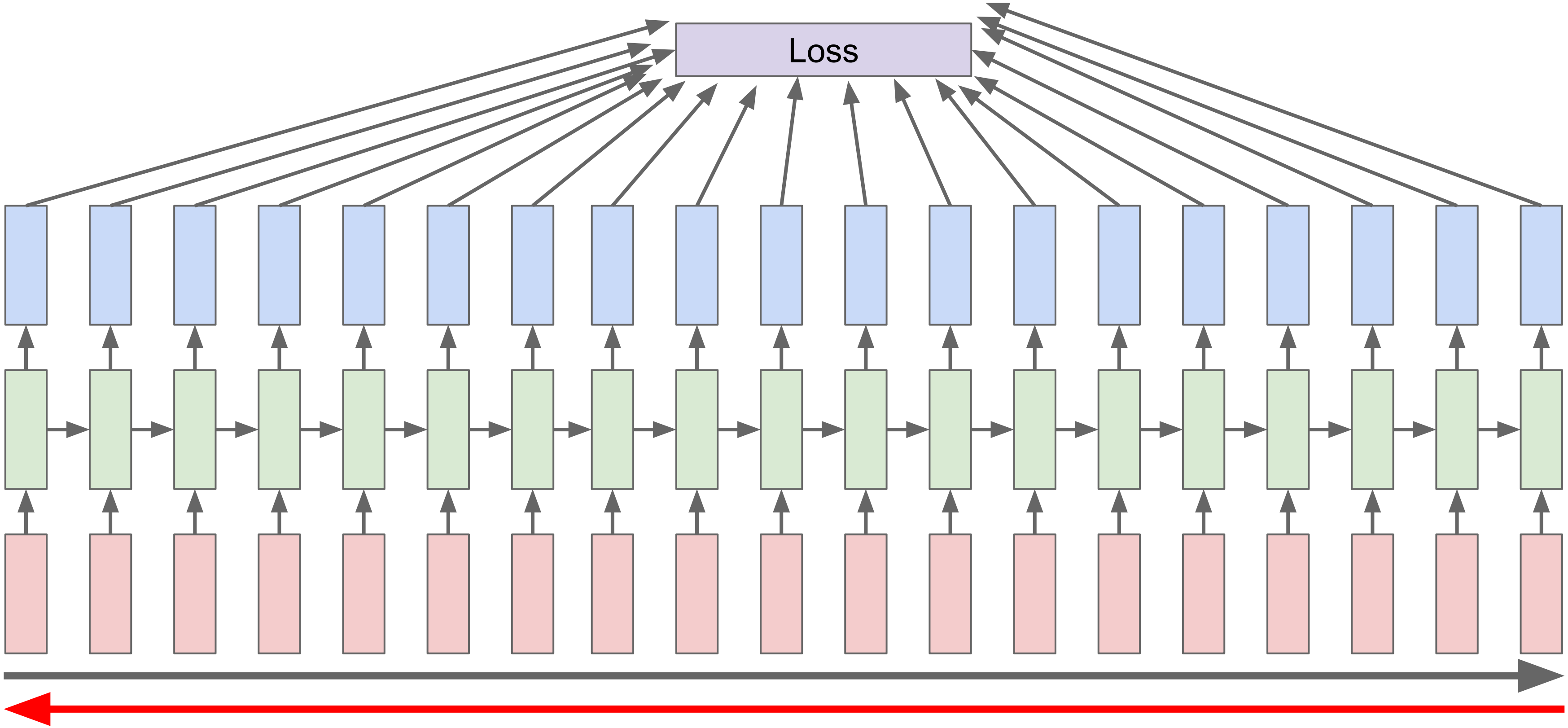
**One to many**: Produce output sequence from single input vector

**Many to one**: Encode input sequence in a single vector

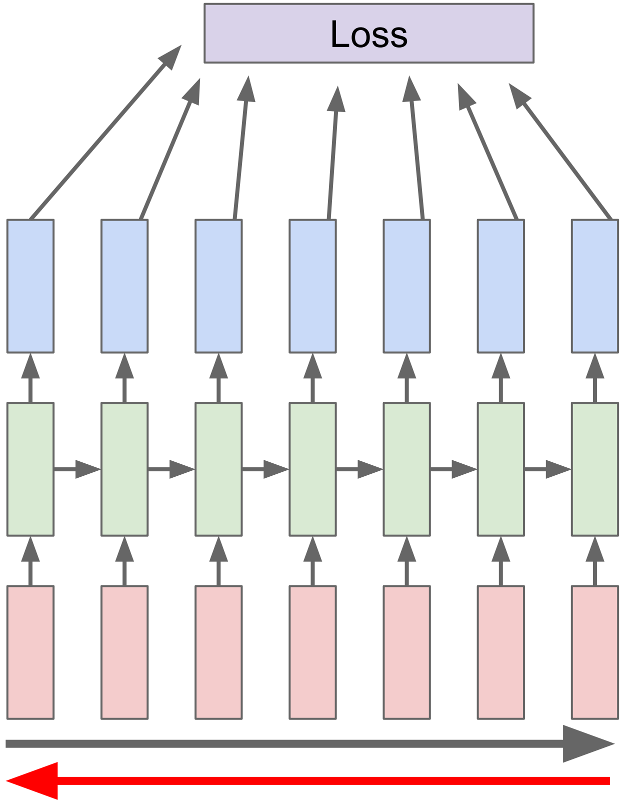


# Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

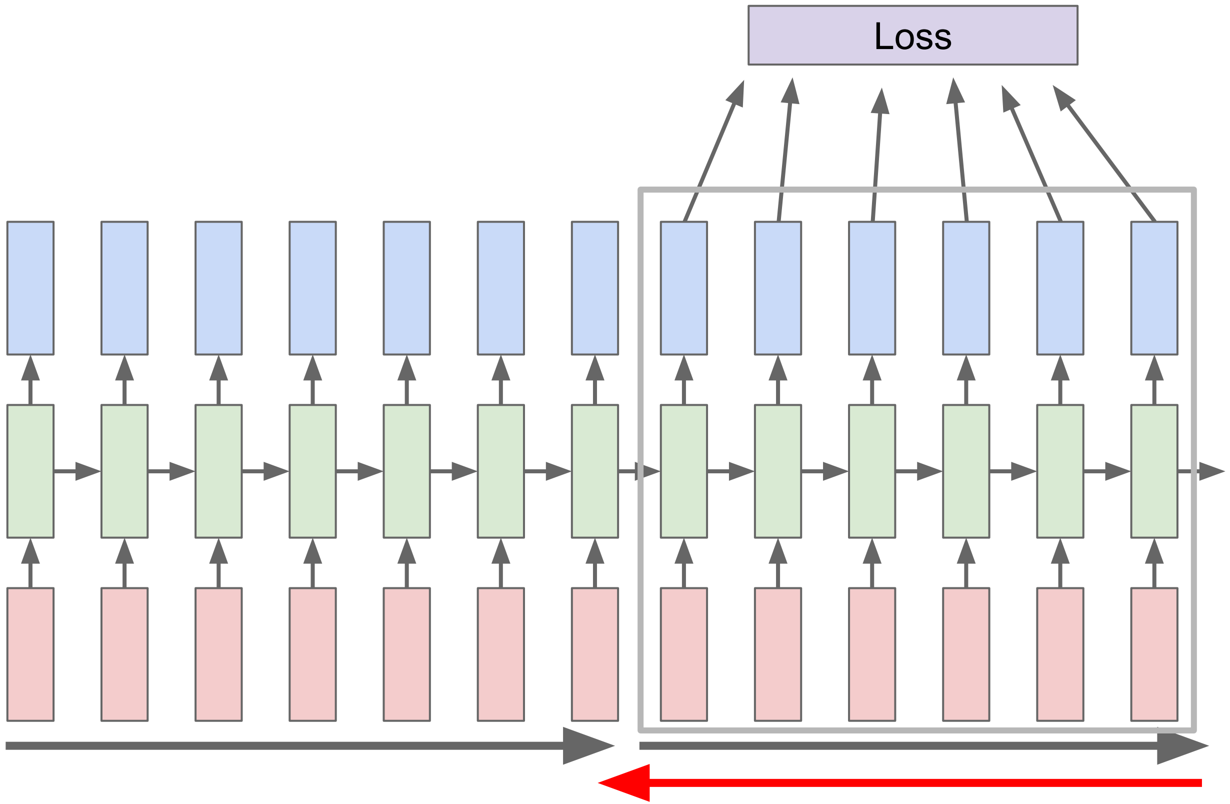


# **Truncated** Backpropagation through time




Run forward and backward through chunks of the sequence instead of whole sequence

# **Truncated** Backpropagation through time




Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

# **Truncated** Backpropagation through time