CS60010: Deep Learning

Spring 2021

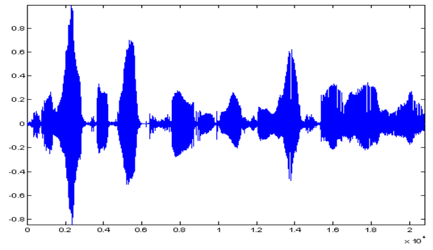
Sudeshna Sarkar

Recurrent Neural Network

15 Feb 2021

Sequences are everywhere

- An RNN models sequences: Time series, Natural Language, Speech



- Sequence data: sentences, speech, stock market, signal data
  - Sequence of words in an English sentence
  - Acoustic features at successive time frames in speech recognition
  - Successive frames in video classification
  - Rainfall measurements on successive days in Hong Kong
  - Daily values of current exchange rate

Why RNNs?

- Can model sequences having variable length
- Inputs, outputs can be different lengths in different examples
- **Efficient**: Weights shared across time-steps

Modeling Sequential Data

- Sample data sequences from a certain distribution

$$P(x_1, x_2, \dots, x_T)$$

- Generate natural sentences to describe an image

$$P(y_1, y_2, \dots, y_T | I)$$

- Activity recognition from a video sequence

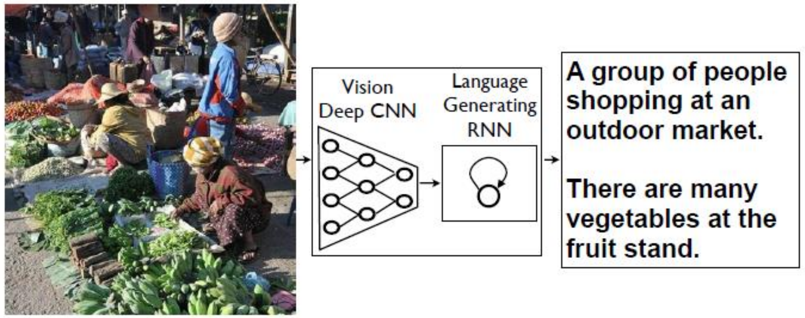
$$P(y | x_1, x_2, \dots, x_T)$$

- Speech Recognition

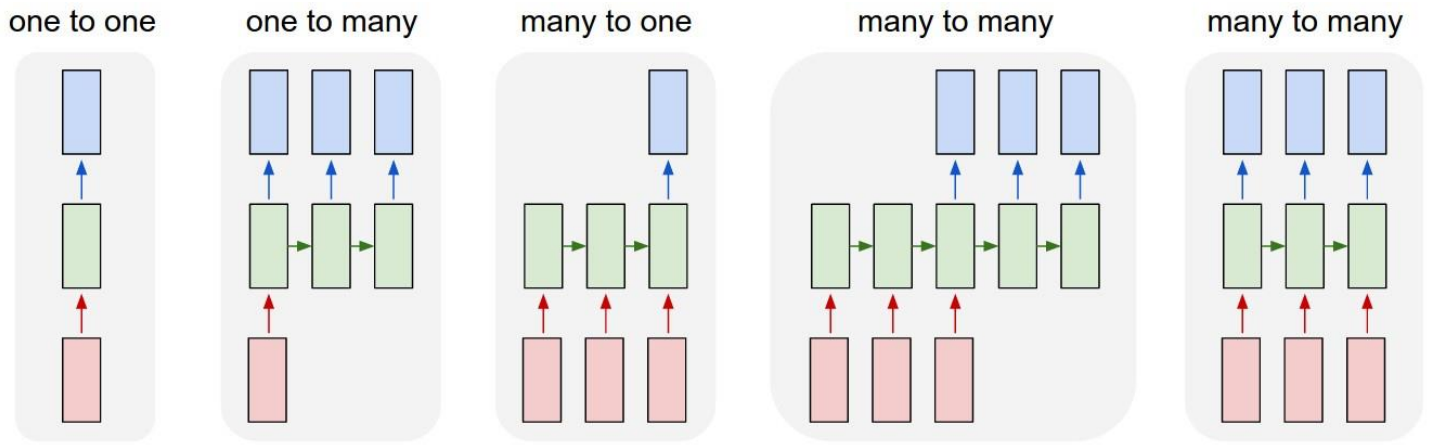
$$P(y_1, y_2, \dots, y_T | x_1, x_2, \dots, x_T)$$

- Machine Translation

$$P(y_1, y_2, \dots, y_T | x_1, x_2, \dots, x_S)$$


Vision
Deep CNN
Language
Generating
RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

Sequences in Input or Output?



Input: No sequence
Output: No sequence
Example: "standard" classification / regression problems

Input: No sequence
Output: Sequence
Example: Im2Caption

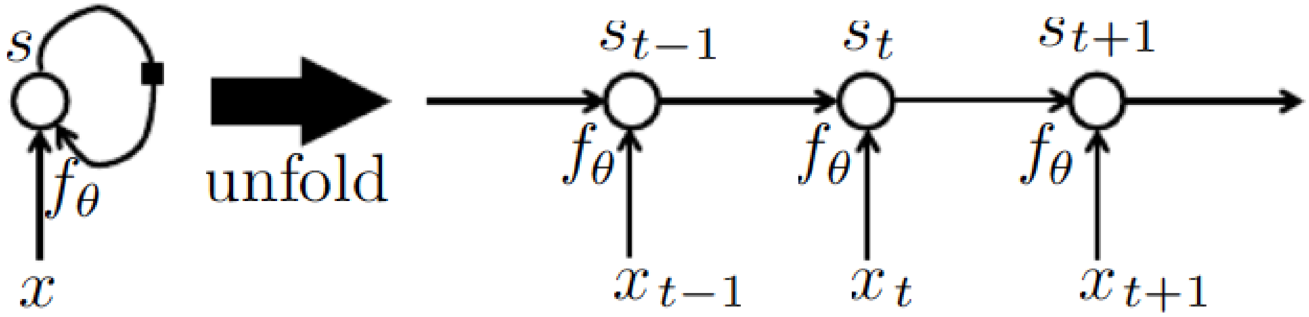
Input: Sequence
Output: No sequence
Example: sentence classification, multiple-choice question answering

Input: Sequence
Output: Sequence
Example: machine translation, video classification, video captioning, open-ended question answering

How do we model sequences?

- With inputs

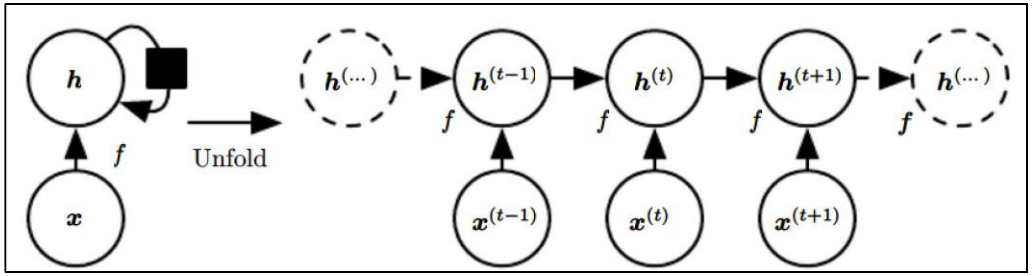
$$\boldsymbol{s}_t = f_\theta(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t)$$



Dynamical system driven by external signal

- Consider a dynamical system driven by external (input) signal $x^{(t)}$: $s^{(t)} = f\left(s^{(t-1)}, x^{(t)}; \theta\right)$
  - The state now contains information about the whole past input sequence. To indicate that the state is hidden rewrite using variable h for state:

$$h^{(t)} = f\left(h^{(t-1)}, x^{(t)}; \theta\right)$$

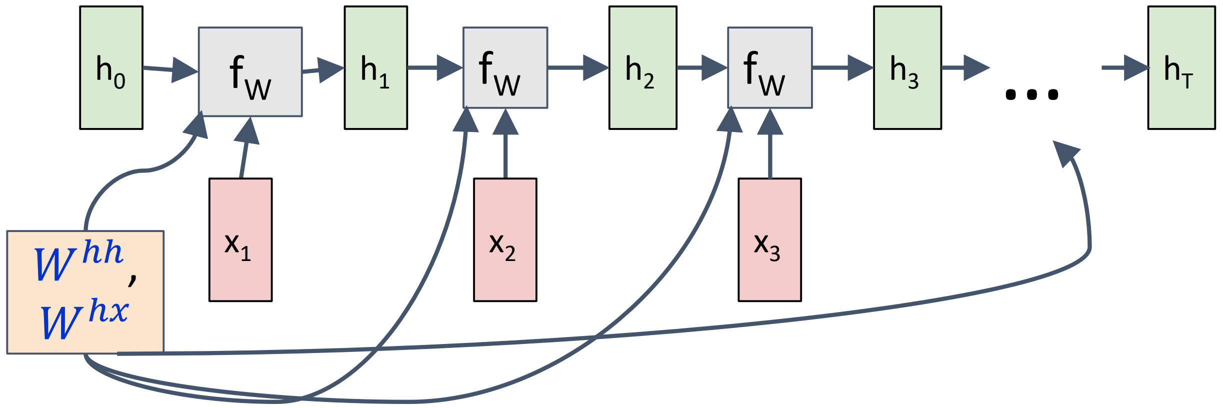


Output prediction by RNN

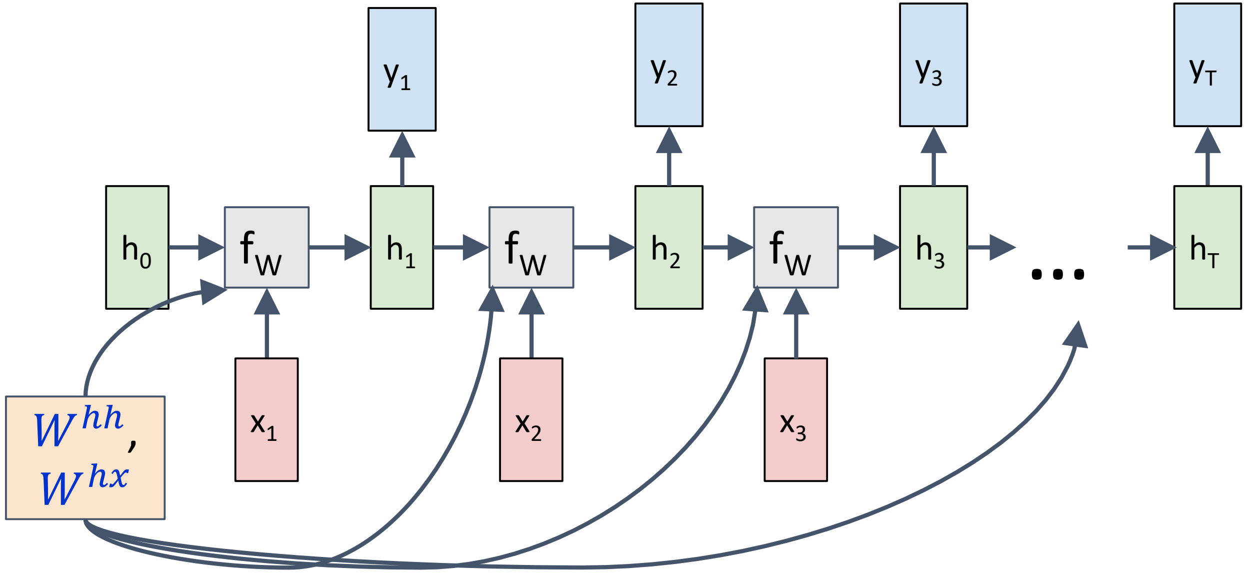
- Task : To predict the future from the past
- The network typically learns to use $\boldsymbol{h}^{(t)}$ as a summary of the task-relevant aspects of the past sequence of inputs upto t
- The summary is in general lossy since it maps a sequence of arbitrary length $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, .., \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ to a fixed length vector $\boldsymbol{h}^{(t)}$
- Depending on the training criterion, the summary keeps some aspects of past sequence more precisely than other aspects

RNN: Computational Graph

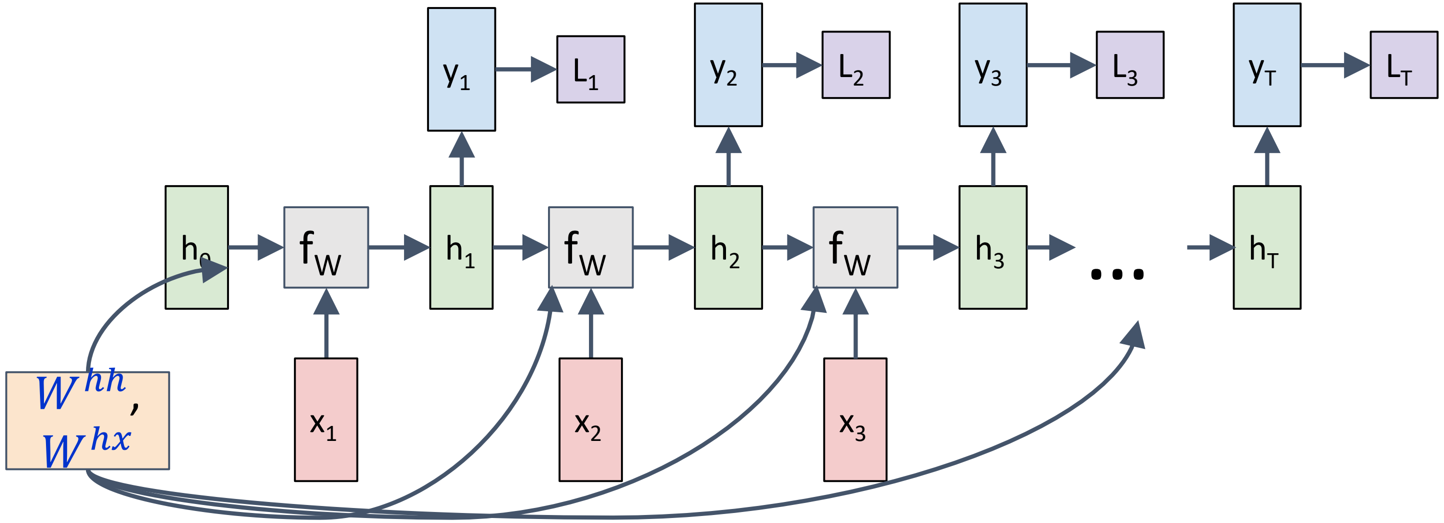
Re-use the same weight matrix at every time-step



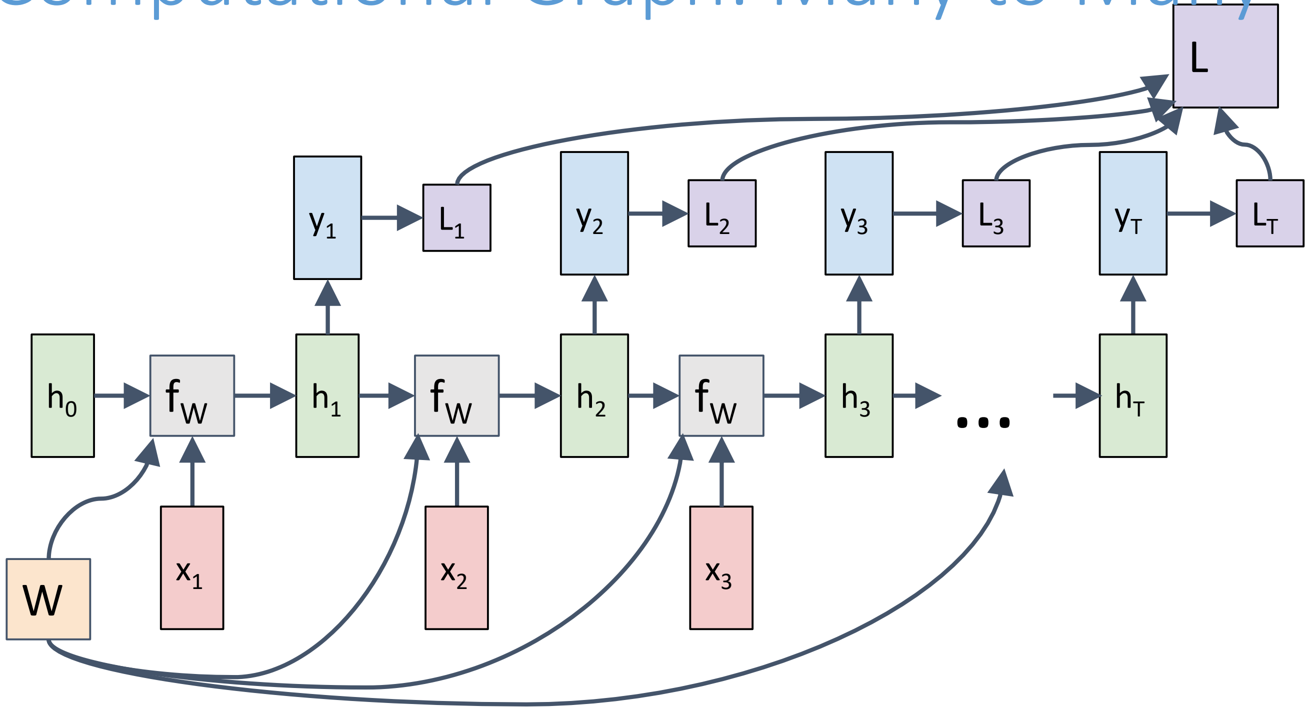
RNN: Computational Graph: Many to Many



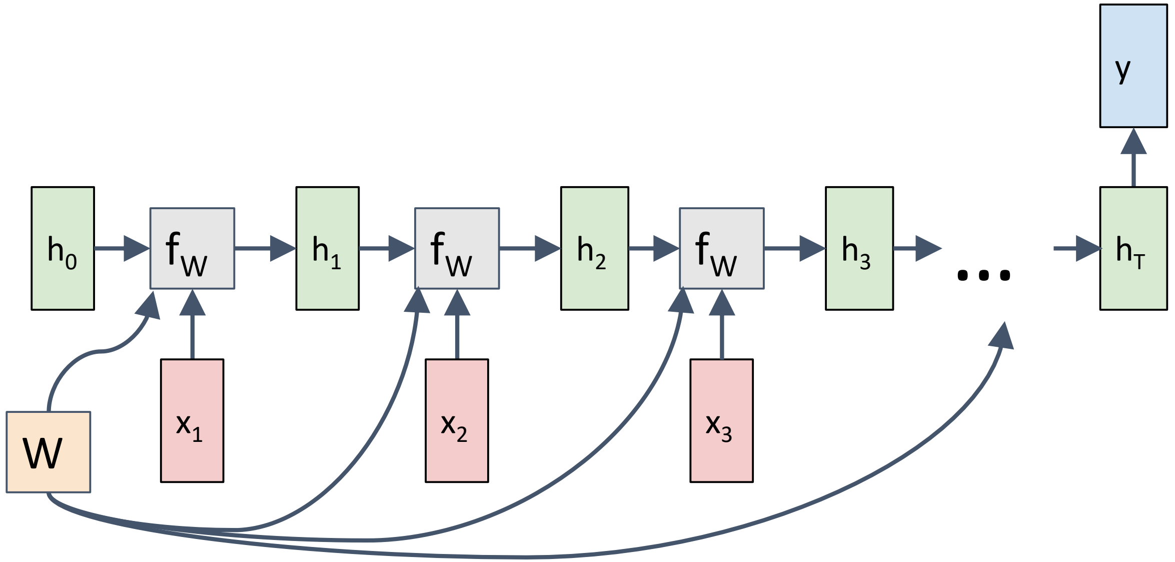
RNN: Computational Graph: Many to Many



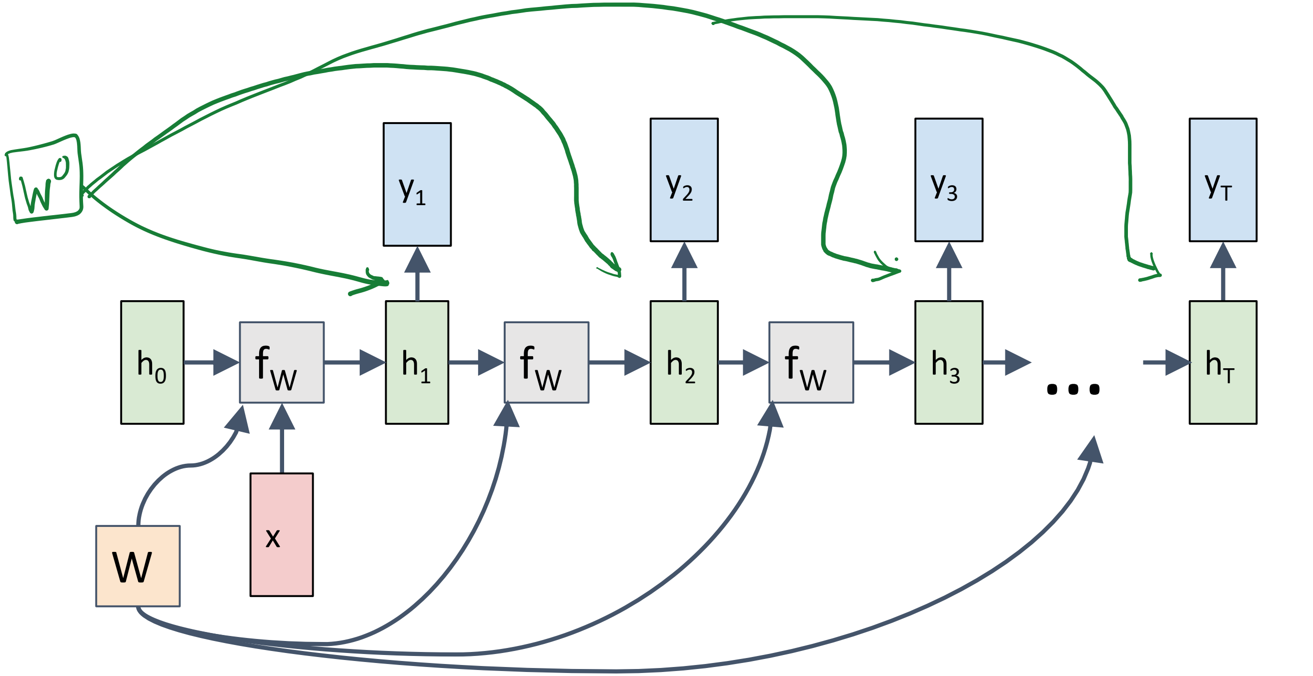
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to One

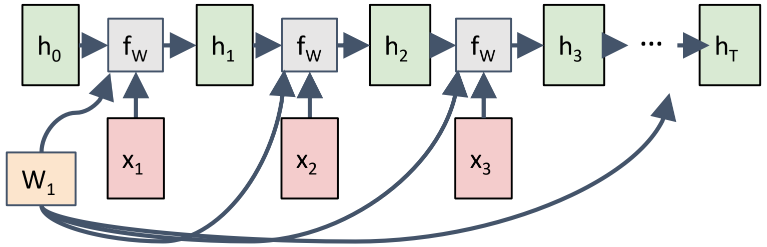


RNN: Computational Graph: One to Many



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

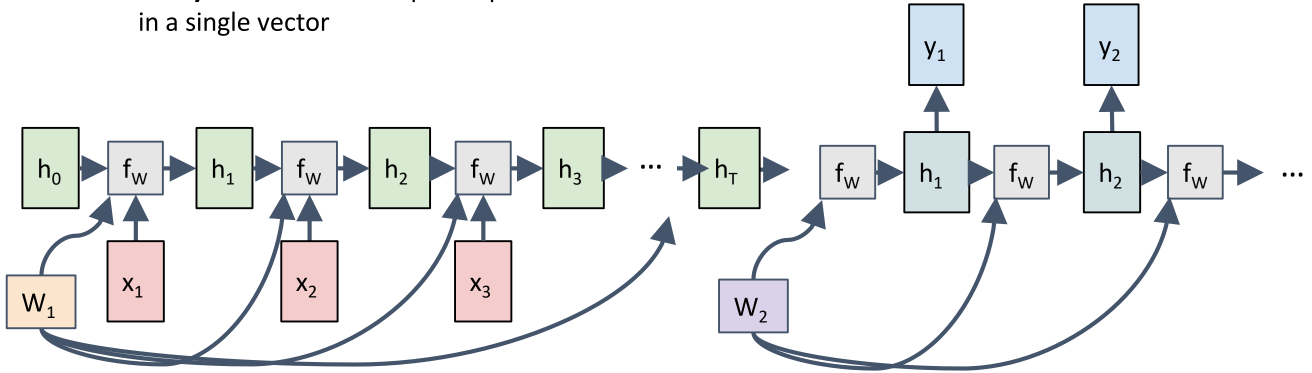
Many to one: Encode input sequence in a single vector



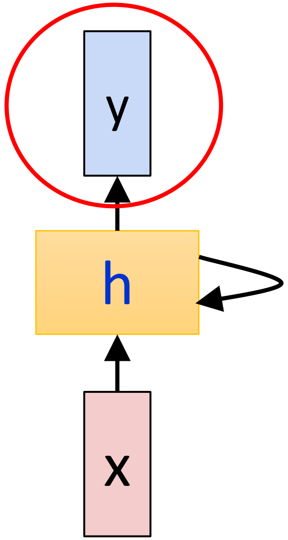
Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

One to many: Produce output sequence from single input vector



Recurrent Neural Network



Recurrent Neural Network

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

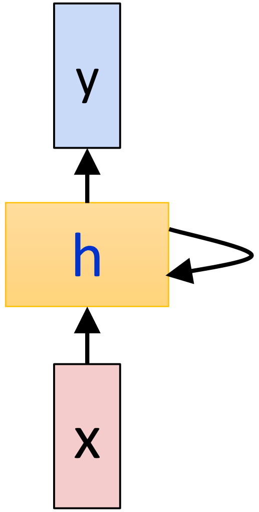
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

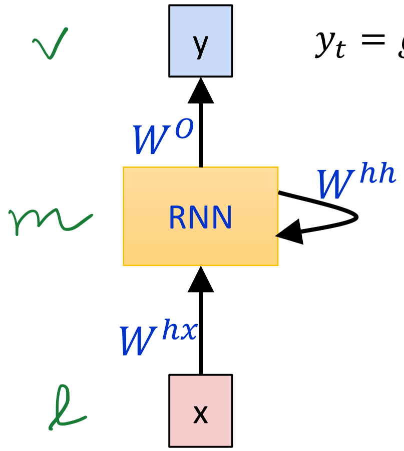
old state

input vector at some time step



Recurrent Neural Network

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



$y_t = g(h_t)$

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

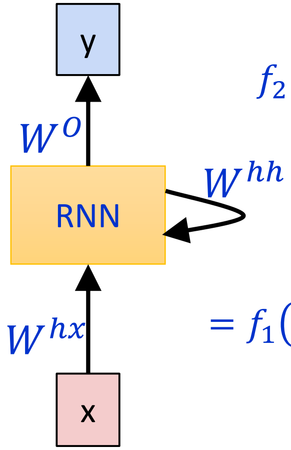
$$y_t = g(h_t) \\ = f_2\left(W^{O} h_t + W_0^{O}\right)$$

$$h_t = f(h_{t-1}, x_t) \\ = f_1\left(W^{hx} x_t + W^{hh} h_{t-1} + W_0^{hh}\right)$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

Recurrent Neural Network

The state consists of a single *“hidden”* vector **h**:



$$y_t = g(h_t)$$
$$f_2\left(W^O h_t + W_0^O\right)$$

$$h_t = f(h_{t-1}, x_t)$$
$$= f_1\left(W^{hx} x_t + W^{hh} h_{t-1} + W_0^{hh}\right)$$

The inputs, outputs, and states are all vector-valued:

$$x_t : l \times 1$$
$$h_t : m \times 1$$
$$y_t : v \times 1$$

Weights in the network:

$$W^{hx} : m \times l$$
$$W^{hh} : m \times m$$
$$W_0^{hh} : m \times 1$$

$$W^O : v \times m$$
$$W_0^O : v \times 1$$

Sequence-to-sequence RNN

- How can we train an RNN to model a transduction on sequences? This problem is sometimes called sequence-to-sequence mapping
- A training set has the form $[x^{(1)}, y^{(1)}, \ldots, x^{(q)}, y^{(q)}]$
- $x^{(i)}$ and $y^{(i)}$ are length $n^{(i)}$ sequences;
- Sequences in the same pair are the same length; and sequences in different pairs may have different lengths

y:	Verb	Det	Noun
x:	Book	the	table

Recurrent Hidden Units

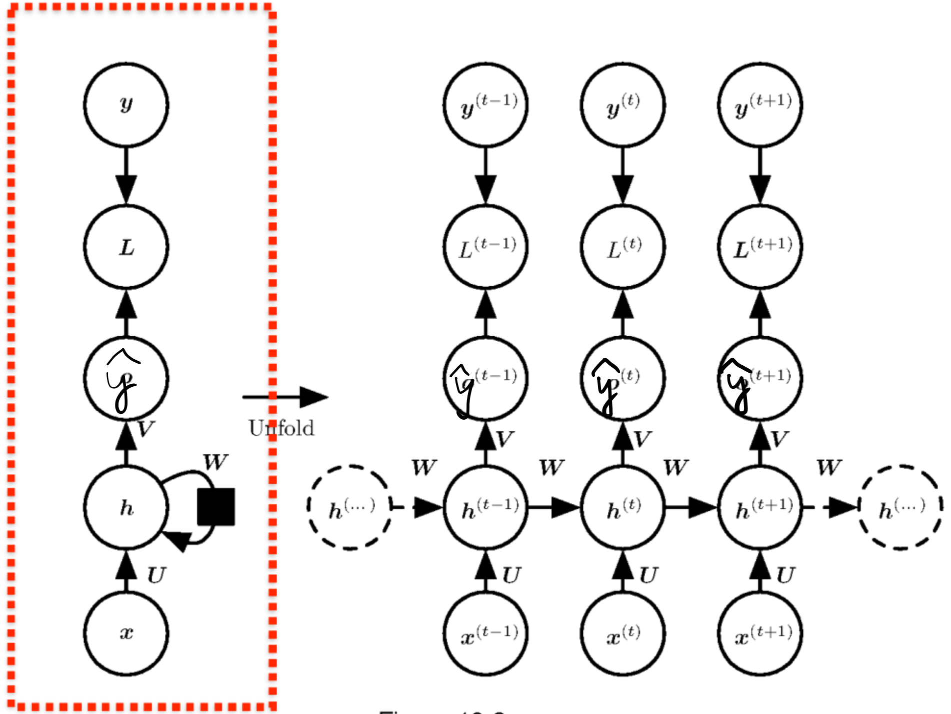


Figure 10.3

Loss function

- Sum up a per-element loss function on each of the output values, where $\hat{\boldsymbol{y}}$ is the predicted sequence and $\boldsymbol{y}$ is the actual one:

$$\mathrm{Loss}_{\mathrm{seq}}\left(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)}\right) = \sum_{t=1}^{n^{(i)}} \mathrm{Loss}_{\mathrm{elt}}\left(\widehat{y_t}^{(i)}, {y_t}^{(i)}\right)$$

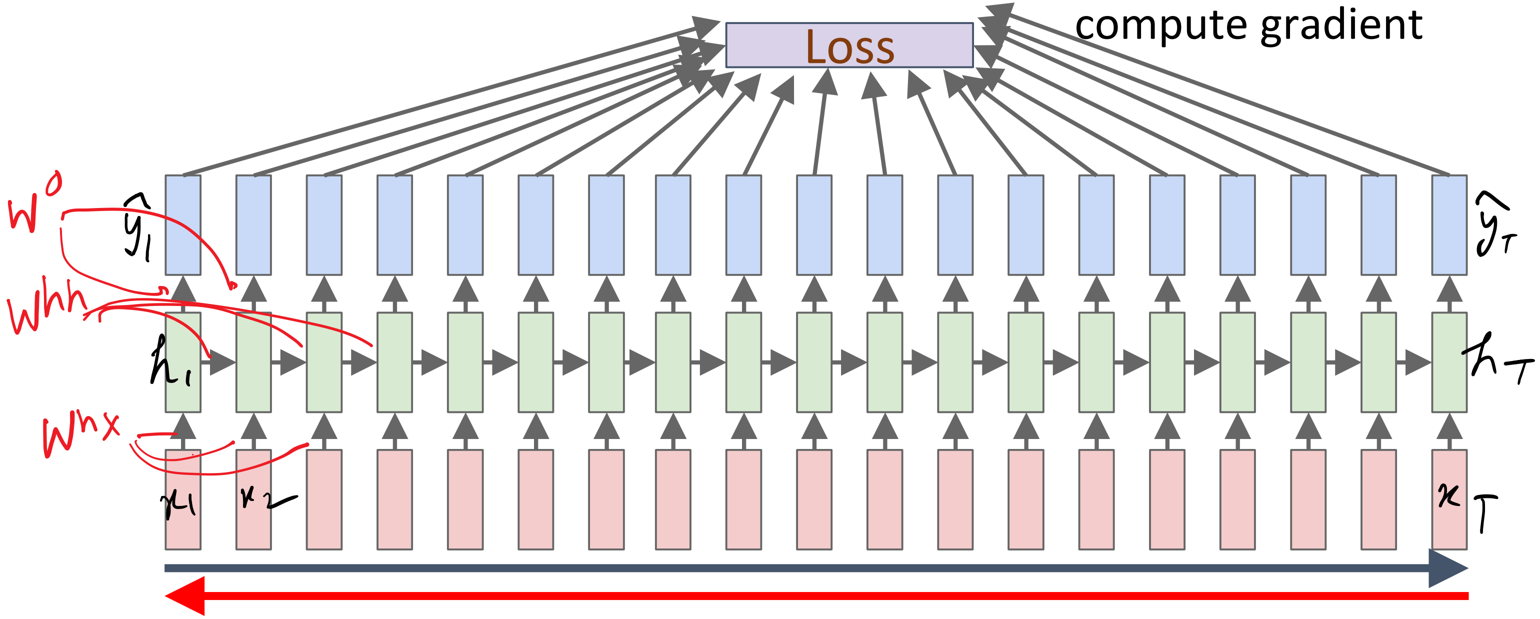
The per-element loss function $\mathrm{Loss}_{\mathrm{elt}}$ will depend on the type of y_t and what information it is encoding.

The overall objective to miminize is :

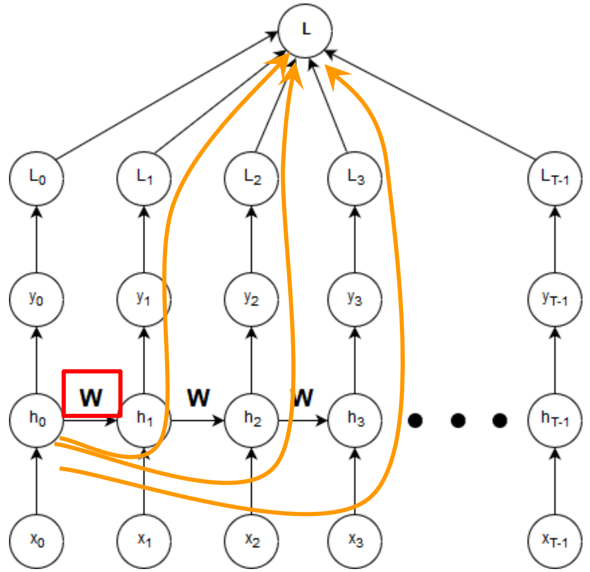
$$J(\theta) = \sum_{t=1}^{q} \mathrm{Loss}_{\mathrm{seq}}\left(\underbrace{\mathrm{RNN}(x^{(i)}; \theta)}_{\hat{y}^{(i)}}, \boldsymbol{y}^{(i)}\right)$$

Backpropagation through time

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



Backpropagation Through Time (BPTT)



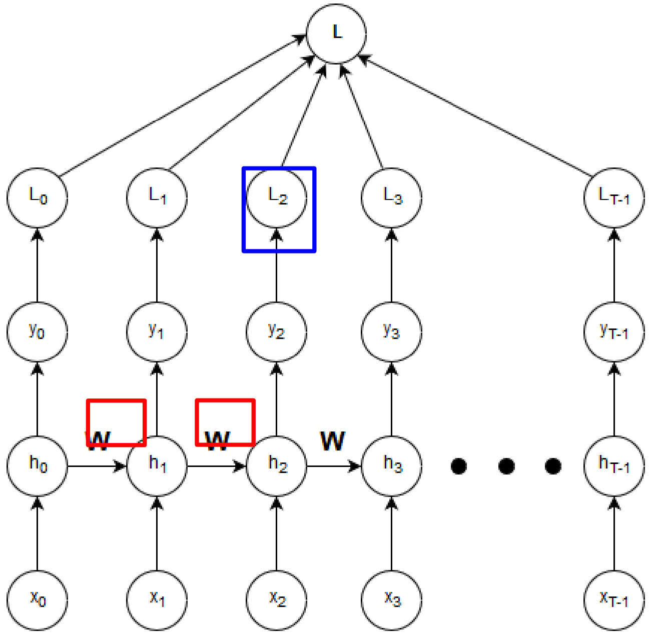
- Update the weight matrix:

$$\mathbf{W} \rightarrow \mathbf{W} - \alpha \frac{\partial L}{\partial \mathbf{W}}$$

- Issue: **W** occurs each timestep
- **Every** path from **W** to L is one dependency
- Find all paths from **W** to L

W_{hh}

Systematically Finding All Paths



- How many paths exist from W to L through L1?
  - 1
- How many paths from W to L through L2?
  - 2 (originating at h0 and h1)

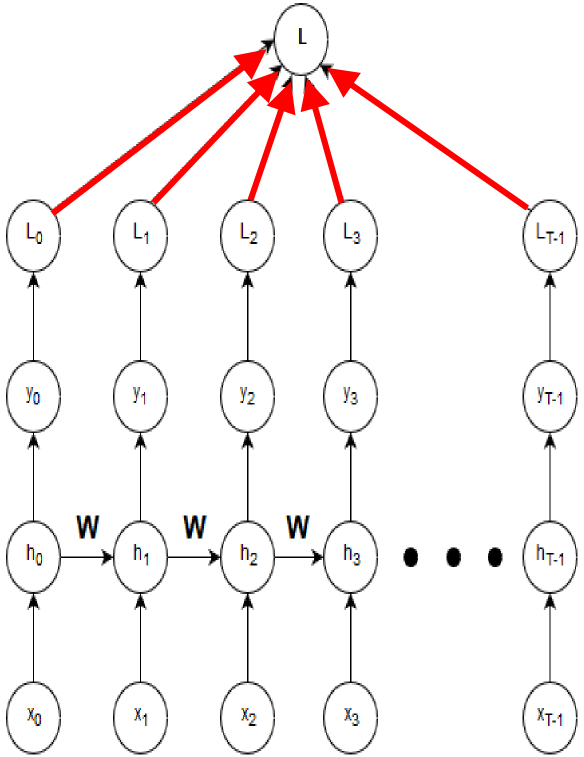
$$\frac{\partial L}{\partial \mathbf{W}}$$

The gradient has two summations:

1: Over L_j

2: Over h_k

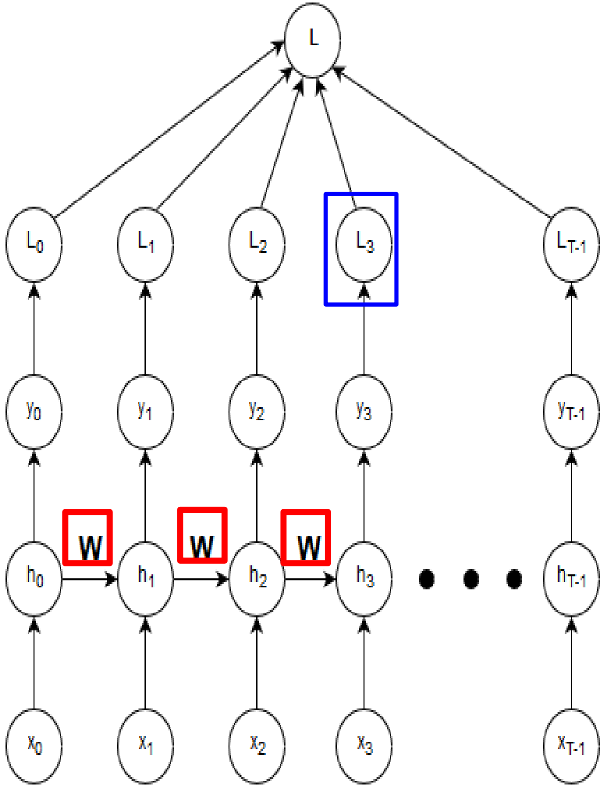
Backpropagation as two summations



First summation over L

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial L_j}{\partial \mathbf{W}}$$

Backpropagation as two summations

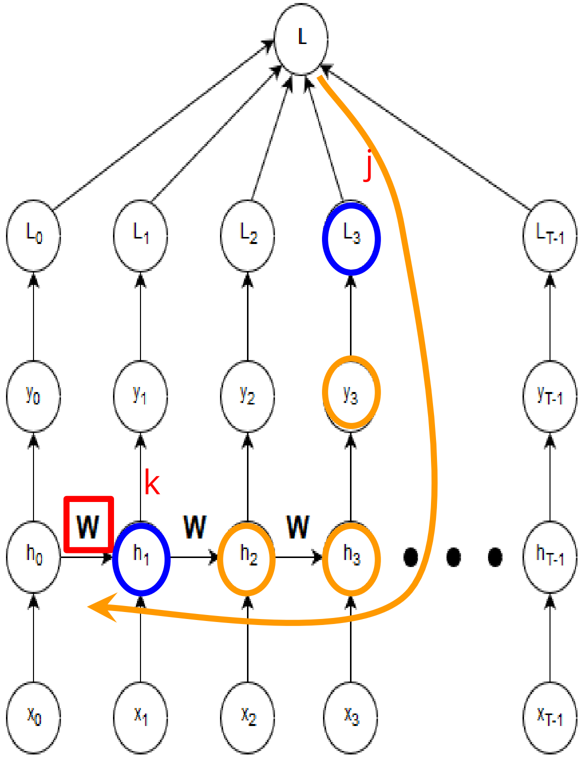


Second summation over h:
Each $\mathbf{L_j}$ depends on the weight matrices *before it*

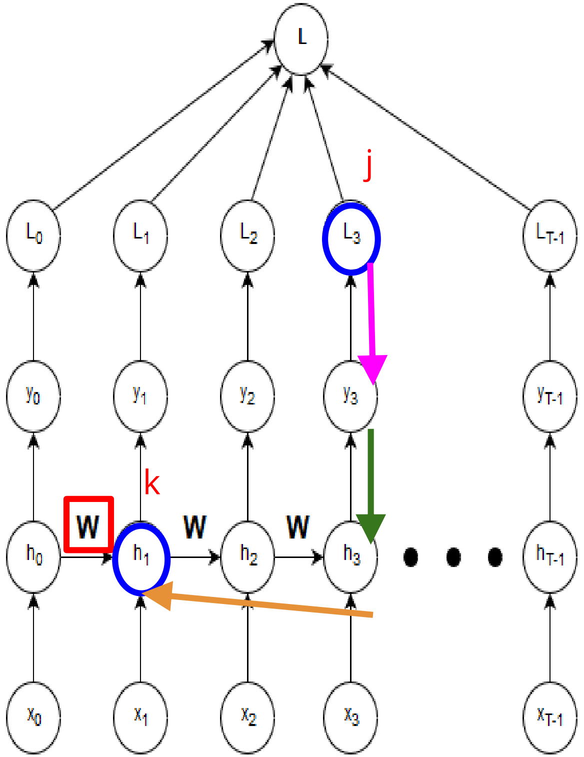
$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \boxed{\frac{\partial L_j}{\partial h_k}} \frac{\partial h_k}{\partial \mathbf{W}}$$

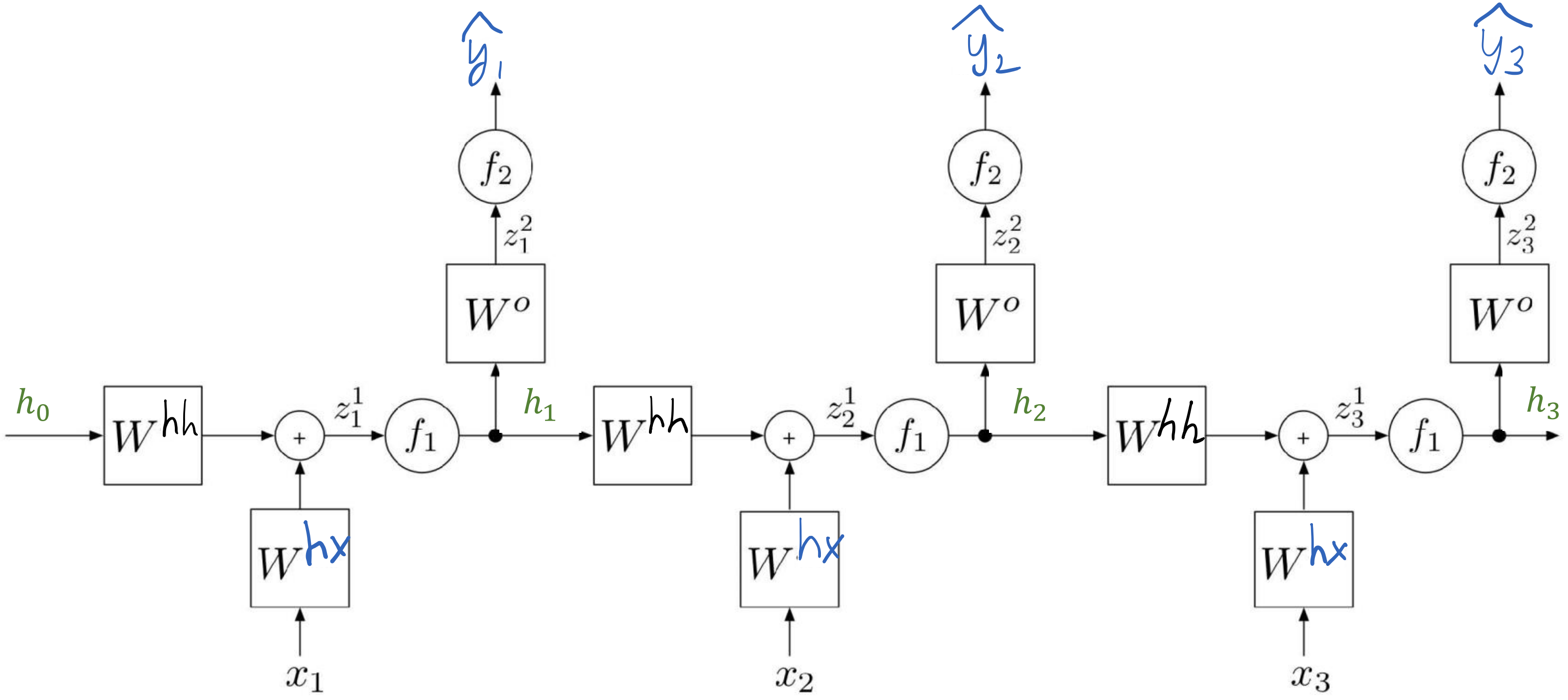
L_j depends on all h_k before it.

Backpropagation as two summations



Backpropagation as two summations




$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
f_2
z_1^2
z_2^2
z_3^2
W^o
h_0
W^{hh}
z_1^1
f_1
h_1
z_2^1
h_2
z_3^1
h_3
W^{hx}
x_1
x_2
x_3

BPTT

1. Sample a training pair of sequences (x, y) ; let their length be n.
2. “Unroll" the RNN to be length n, and initialize h_0

Performing almost an ordinary backpropagation training procedure in a feed-forward neural network, but with the difference that the weight matrices are shared among the layers

1. Do the forward pass, to compute the predicted output sequence $\hat{y}$
2. Do backward pass to compute the gradients. Find

$$\frac{dL_{seq}}{dW} = \sum_{u=1}^{n} \frac{dL_u}{dW} \qquad L_u = \mathrm{Loss}_{\mathrm{elt}}(\hat{y}_u, y_u)$$

$$= \sum_{u=1}^{n} \sum_{t=1}^{n} \frac{\partial L_u}{\partial h_t} \cdot \frac{\partial h_t}{\partial W} = \sum_{t=1}^{n} \frac{\partial h_t}{\partial W} \cdot \sum_{u=1}^{n} \frac{\partial L_u}{\partial h_t}$$

BPTT

$$\frac{dL_{seq}}{dW} = \sum_{t=1}^{n} \frac{\partial h_t}{\partial W} \cdot \sum_{u=1}^{n} \frac{\partial L_u}{\partial h_t}$$

h_t only affects $L_t, L_{t+1}, \dots\ L_n$

$$\frac{dL_{seq}}{dW} = \sum_{t=1}^{n} \frac{\partial h_t}{\partial W} \cdot \sum_{u=t}^{n} \frac{\partial L_u}{\partial h_t}$$

$$= \sum_{t=1}^{n} \frac{\partial h_t}{\partial W} \cdot \left(\frac{\partial L_t}{\partial h_t} + \underbrace{\sum_{u=t+1}^{n} \frac{\partial L_u}{\partial h_t}}_{\delta^{h_t}} \right)$$

δ^{h_t} is the dependence of the loss on steps after t on the state at time t.
We can compute this backwards, with t going from n down to 1.

Define future loss F_t

$$F_t = \sum_{u=t+1}^{n} \mathrm{Loss}_{\mathrm{elt}}(\hat{y}_u, y_u)$$

$$\delta^{h_t} = \frac{\partial F_t}{\partial h_t}$$

$$\partial h_n = 0$$

$$\boldsymbol{\delta}^{\boldsymbol{h_{t-1}}} = \frac{\partial}{\partial h_{t-1}} \sum_{u=t}^{n} \mathrm{Loss}_{\mathrm{elt}}(\hat{y}_u, y_u)$$

$$= \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial}{\partial h_t} \sum_{u=t}^{n} \mathrm{Loss}_{\mathrm{elt}}(\hat{y}_u, y_u)$$

$$= \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial}{\partial h_t} \left[\mathrm{Loss}_{\mathrm{elt}}(\hat{y}_t, y_t) + \sum_{u=t+1}^{n} \mathrm{Loss}_{\mathrm{elt}}(\hat{y}_u, y_u) \right]$$

$$= \frac{\partial h_t}{\partial h_{t-1}} \cdot \left[\frac{\partial \mathrm{Loss}_{\mathrm{elt}}(\hat{y}_t, y_t)}{\partial h_t} + \boldsymbol{\delta}^{\boldsymbol{h_t}} \right]$$

BPTT

$$\delta^{h_{t-1}} = \frac{\partial h_t}{\partial h_{t-1}} \cdot \left[\frac{\partial \mathrm{Loss}_{\mathrm{elt}}(\hat{y}_t, y_t)}{\partial h_t} + \delta^{h_t}\right]$$

- we can use the chain rule again to find the dependence of the element loss at time t on the state at that same time

$$\frac{\partial \mathrm{Loss}_{\mathrm{elt}}(\hat{y}_t, y_t)}{\partial h_t} = \frac{\partial z_t^2}{\partial h_t} \cdot \frac{\partial \mathrm{Loss}_{\mathrm{elt}}\left(\hat{y}_t, y_t\right)}{\partial z_t^2}$$

and the dependence of the state at time t on the state at $t-1$, noting that we are performing an elementwise multiplication between W_t^{hh} and the vector of $f^{1\prime}$ values, $\frac{\partial h_t}{\partial z_t^1}$

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial z_t^1}{\partial h_{t-1}} \cdot \frac{\partial h_t}{\partial z_t^1} = {W^{hh}}^T * f^{1\prime}(z_t^1)$$

Thus, we get

$$\delta^{h_{t-1}} = {W^{hh}}^T * f^{1\prime}(z_t^1) \cdot \left({W^O}^T \frac{\partial L_t}{\partial z_t^2} + \boldsymbol{\delta}^{\boldsymbol{h}_t}\right)$$

BPTT weight updates

$$\frac{dL_{seq}}{dW^{hh}} += \frac{\partial F_{t-1}}{\partial W^{hh}} = \frac{\partial z_t^1}{\partial W^{hh}} \frac{\partial h_t}{\partial z_t^1} \frac{\partial F_{t-1}}{\partial h_t}$$

$$\frac{dL_{seq}}{dW^{hx}} += \frac{\partial F_{t-1}}{\partial W^{hx}} = \frac{\partial z_t^1}{\partial W^{hx}} \frac{\partial h_t}{\partial z_t^1} \frac{\partial F_{t-1}}{\partial h_t}$$

$$\frac{dL_{seq}}{dW^{O}} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial W^{O}} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial z_2^t} \cdot \frac{\partial z_2^t}{\partial W^{O}}$$

Vanishing Gradients

Take a careful look at the backward propagation of the gradient along the sequence:

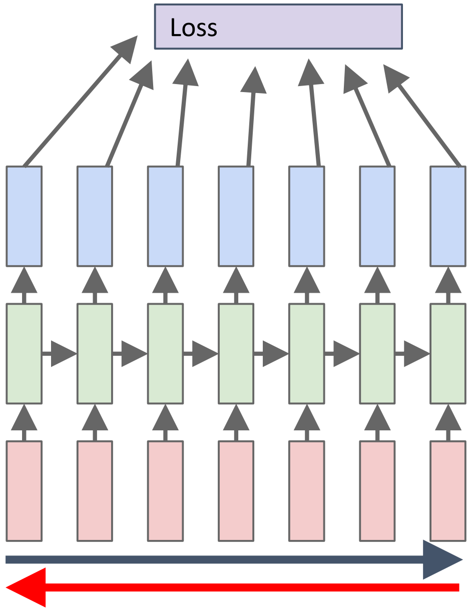
$$\delta^{h_{t-1}} = \frac{\partial h_t}{\partial h_{t-1}} \cdot \left[\frac{\partial \mathrm{Loss}_{\mathrm{elt}}(\hat{y}_t, y_t)}{\partial h_t} + \delta^{h_t}\right]$$

Consider a case where only the output at the end of the sequence is incorrect, but it depends critically, via the weights, on the input at time 1. In this case, we will multiply the loss at step n by

$$\frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_3}{\partial h_2} \cdot \dots \cdot \frac{\partial h_n}{\partial h_{n-1}}$$

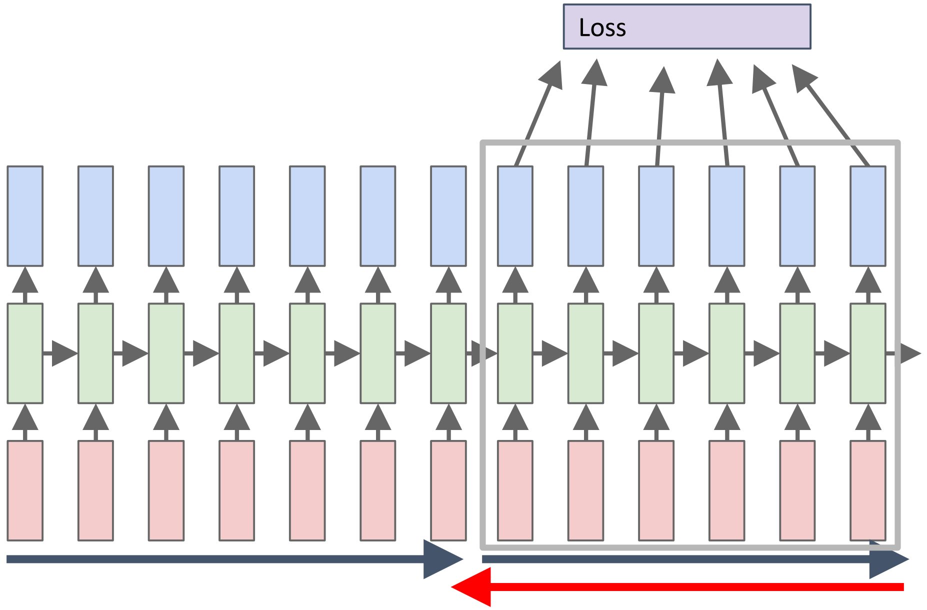
In general, this quantity will either grow or shrink exponentially with the length of the sequence, and make it very difficult to train.

Truncated Backpropagation through time



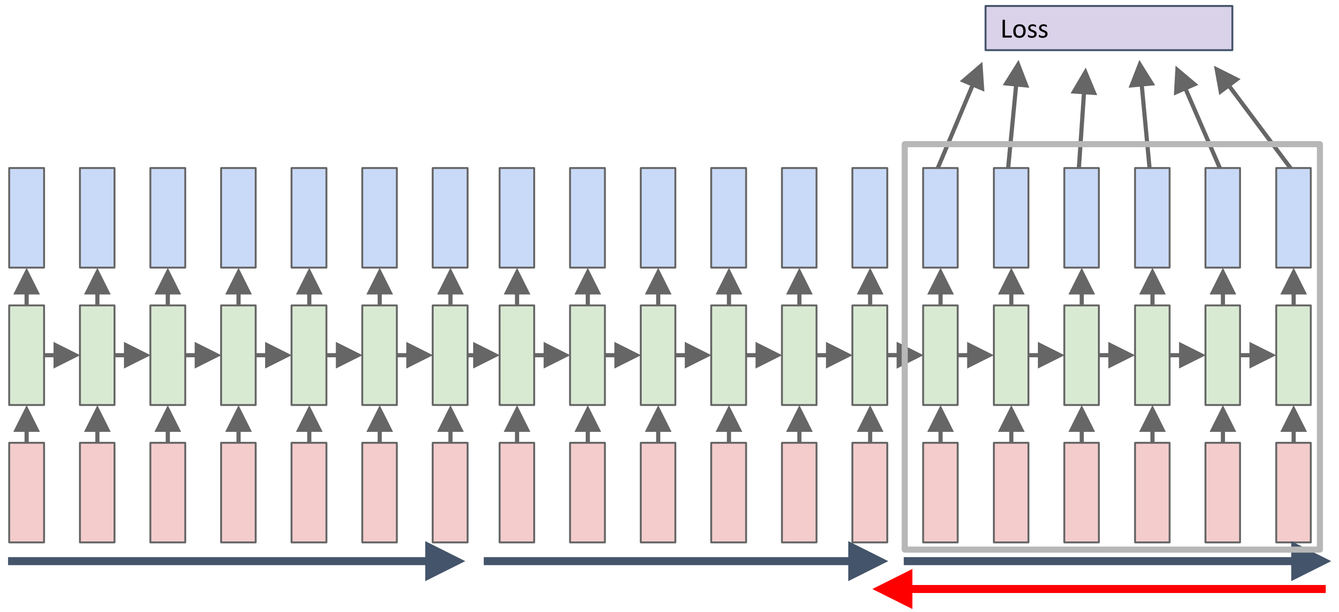
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



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

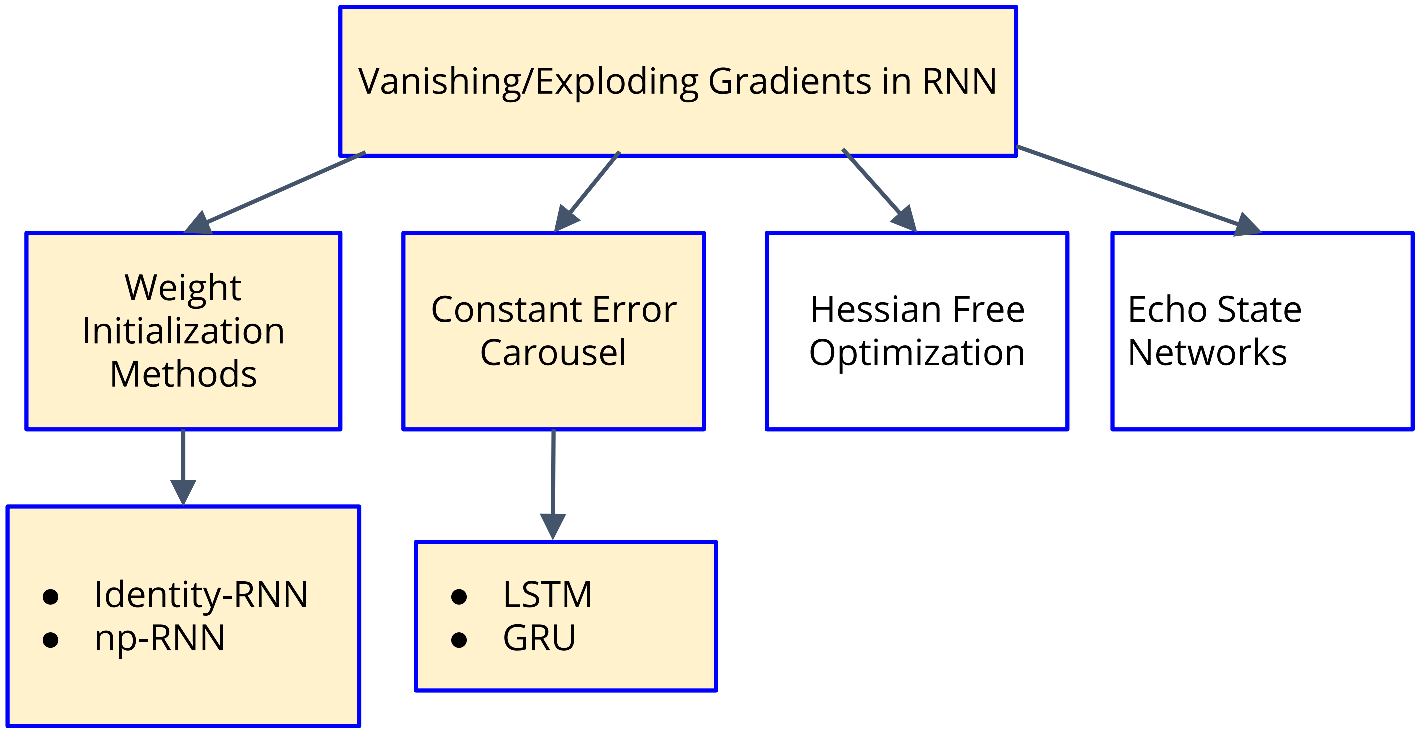
Truncated Backpropagation through time



Complexity of BPTT

- Computing gradient of the loss function wrt parameters is expensive
  - It involves performing a forward propagation pass followed by a backward propagation through the graph
- Run time is $O(\tau)$ and cannot be reduced by parallelization
- States computed during forward pass must be stored until reused in the backward pass
  - So memory cost is also $O(\tau)$
- RNN with hidden unit recurrence is very powerful but also expensive to train

Addressing Vanishing / exloding gradients



Vanishing/Exploding Gradients in RNN
Weight Initialization Methods
Constant Error Carousel
Hessian Free Optimization
Echo State Networks
● Identity-RNN
● np-RNN
● LSTM
● GRU

Conditional independence assumption

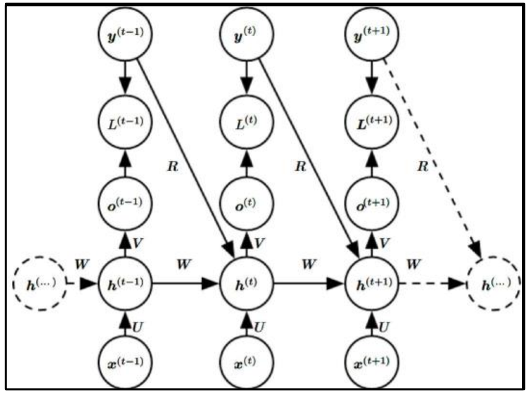
- RNN receives a sequence of vectors $\boldsymbol{x}^{(t)}$ as input
- RNN described by $\boldsymbol{z}^{(t)} = \boldsymbol{b} + W^{hh}\boldsymbol{h}^{(t\text{-}1)} + W^{hx}\boldsymbol{x}^{(t)}$ corresponds to a conditional distribution $P(\boldsymbol{y}^{(1)},..,\ \boldsymbol{y}^{(\tau)} | \boldsymbol{x}^{(1)},..,\ \boldsymbol{x}^{(\tau)})$
- It makes a conditional independence assumption that this distribution factorizes as

$$P(\boldsymbol{y}^{(t)} \overset{t}{|} \ \boldsymbol{x}^{(1)},..,\boldsymbol{x}^{(t)})$$

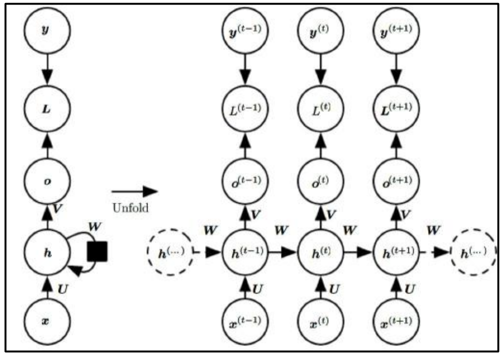
- To remove the conditional independence assumption, we can add connections from the output at time t to the hidden unit at time t+1

Removing conditional independence assumption

Connections from previous output to current state allow RNN to model arbitrary distribution over sequences of y



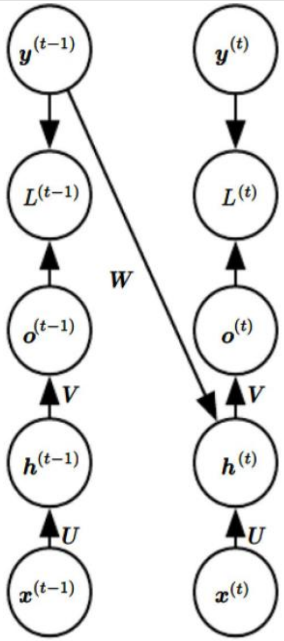
Compare it to model that is only able to represent distributions in which the y values are conditionally independent from each other given x values



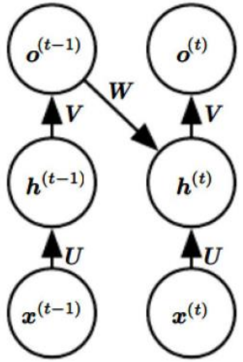
Teacher forcing

Train time: We feed the correct output y(t) (from teacher) drawn from the training set as input to h(t+1)

Test time:
True output is not known.
We approximate the correct output $\boldsymbol{y}^{(t)}$ with the model's output $\boldsymbol{o}^{(t)}$ and feed the output back to the model



Train time

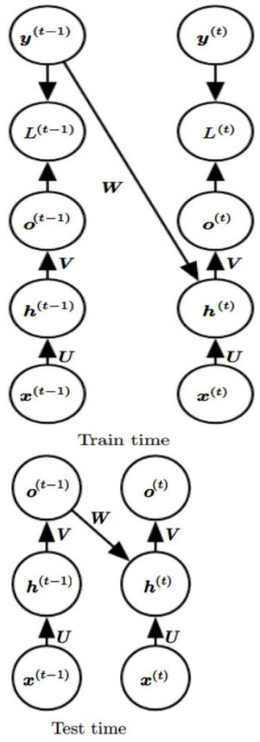


Test time

Training with Teacher forcing

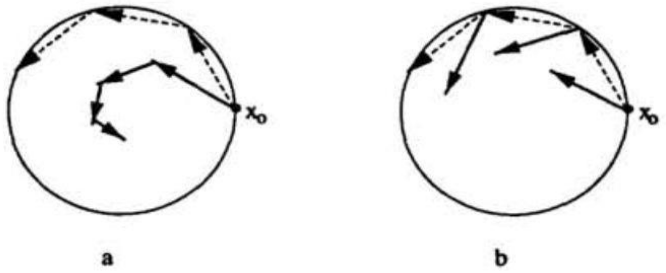
- Teacher forcing: during training the model receives the ground truth output $y^{(t)}$ as input at time $t+1$.
- Advantage
  1. In comparing loss function to output all time steps are decoupled -> each step can be trained in isolation
  2. Training can be parallelized
     - Gradient for each step *t* computed in isolation
     - No need to compute output for the previous step first, because training set provides ideal value of output
  3. Can be trained with teacher forcing

The model is trained to maximize the conditional probability of current output y(t), given both the x sequence so far and the previous output y(t-1)



Visualizing Teacher Forcing

- Imagine that the network is learning to follow a trajectory
- It goes astray (because the weights are wrong) but teacher forcing puts the net back on its trajectory
- By setting the state of all the units to that of teacher's.



(a) Without teacher forcing, trajectory runs astray (solid lines) while the correct trajectory are the dotted lines
(b) With teacher forcing trajectory corrected at each step

Training with both Teacher Forcing and BPTT

Less powerful than with hidden-to- hidden recurrent connections

- It cannot simulate a universal TM
- It requires that the output capture all information of past to predict future

- Some models may be trained with both Teacher forcing and Backward Propagation through time (BPTT)
  - When there are both hidden-to-hidden recurrences as well as output-to- hidden recurrences

RNN 3: hidden2hidden, single output.

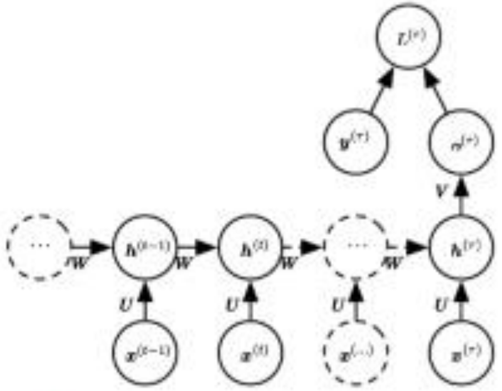


Figure 10.5: Time-unfolded recurrent neural network with a single output at the end of the sequence. Such a network can be used to summarize a sequence and produce a fixed-size representation used as input for further processing. There might be a target

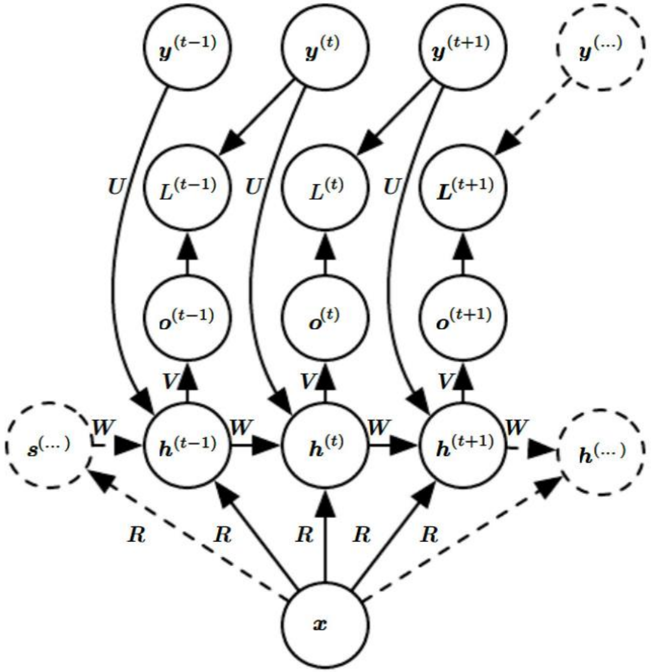
Such a network can be used to summarize a sequence and produce a fixed-size representation used as input for further processing.
There might be a target right at the end or the gradient on the output $o^{(t)}$ can be obtained by backpropagation from further downstream modules

Vector to sequence RNN

If **x** is a fixed-sized vector, we can make it an extra input of the RNN that generates the **y** sequence.

- as an extra input at each time step,
- as the initial state $\mathbf{h}_0$
- Both

- Example: generate caption for an image



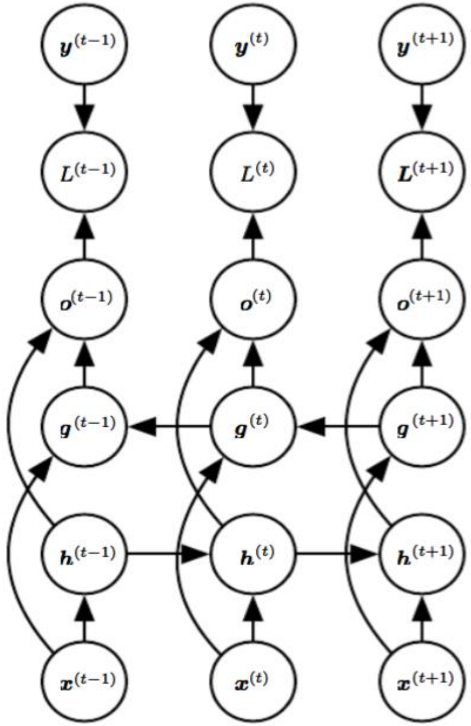
Bidirectional RNNs

- Combine an RNN that moves forward through time from the start of the sequence
- Another RNN that moves backward through time beginning from the end of the sequence

- Need for bidirectionality
  - In speech recognition, the correct interpretation of the current sound may depend on the next few phonemes because of coarticulation and the next few words because of linguistic dependencies
  - handwriting recognition
  - Machine translation

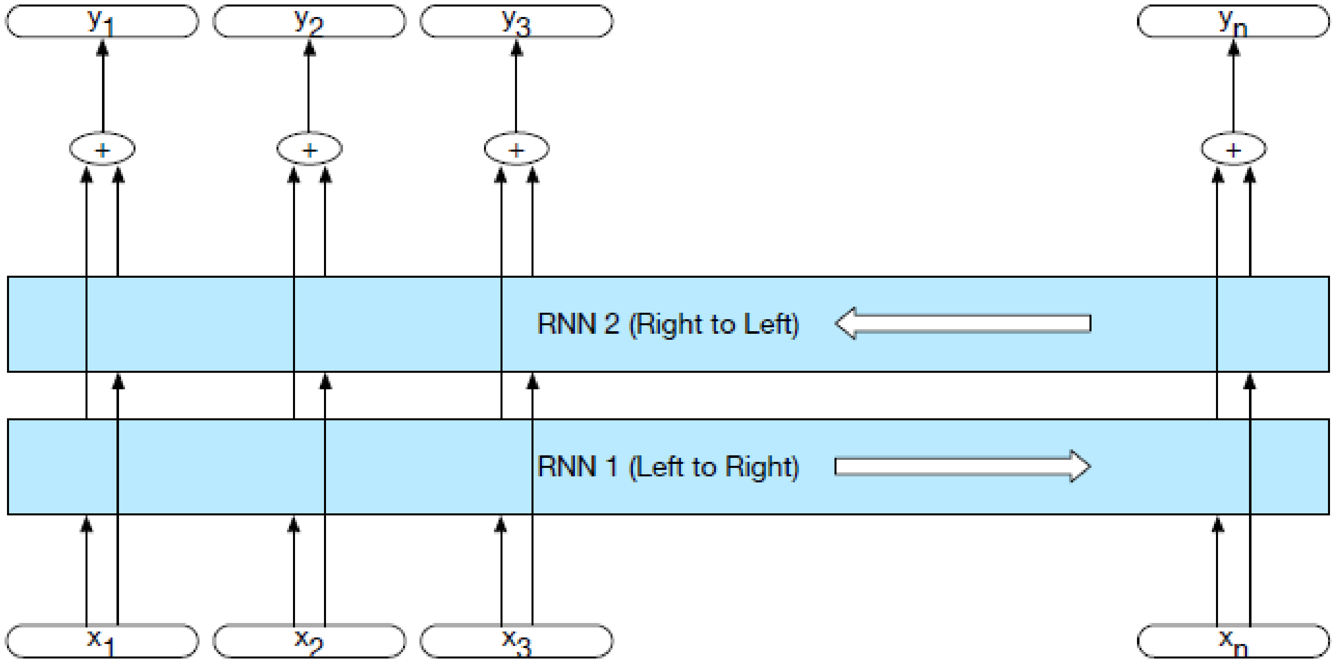
Bidirectional RNNs

- $h^{(t)}$ summaries the information from the past sequence, and
- $g^{(t)}$ summaries the information from the future sequence



Bidirectional RNNs

- Consists of **two independent RNNs**
- outputs of the two networks are combined to capture **both the left and right contexts** of an input at each point in time.



Encoder Decoder RNNs

- Applications such as speech recognition, machine translation or question-answering where the input and output sequences in the training set are generally not of the same length

Encoder-Decoder Sequence to Sequence RNN

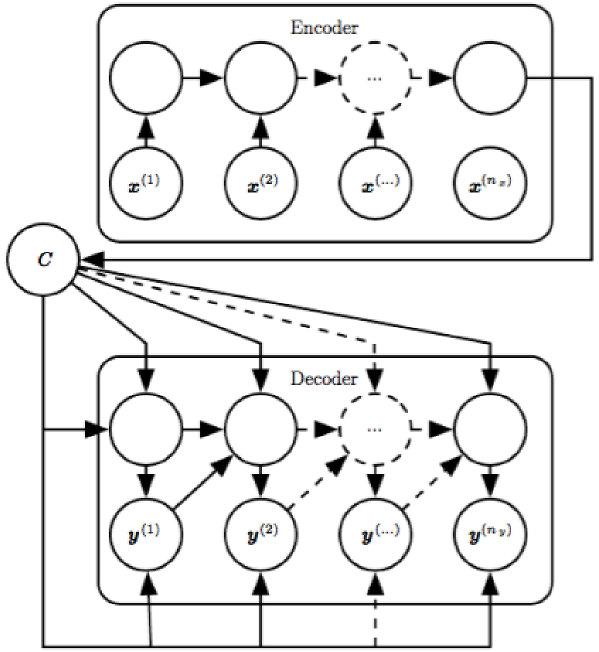
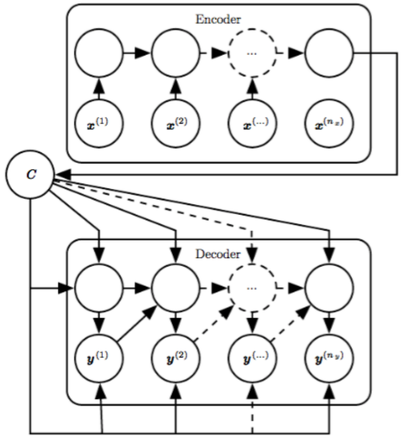


Figure 10.12: Example of an encoder-decoder or sequence-to-sequence RNN architecture, for learning to generate an output sequence $(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n_y)})$ given an input sequence $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n_x)})$. It is composed of an encoder RNN that reads the input sequence and a decoder RNN that generates the output sequence (or computes the probability of a given output sequence). The final hidden state of the encoder RNN is used to compute a generally fixed-size context variable C which represents a semantic summary of the input sequence and is given as input to the decoder RNN.

Encoder-Decoder Sequence to Sequence RNN

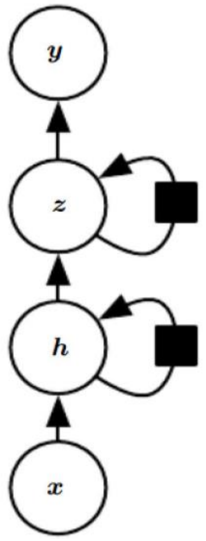


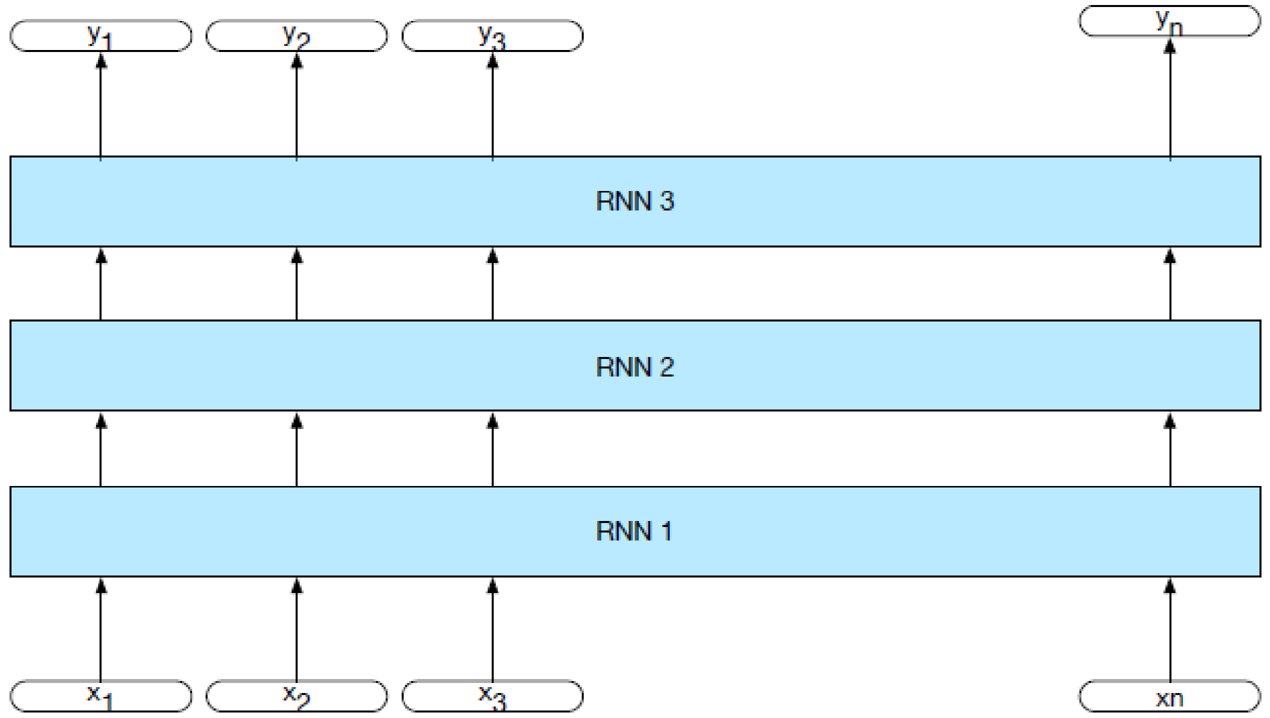
- An encoder or reader or input RNN processes the input sequence. The encoder emits the context C , usually as a simple function of its final hidden state.
- A decoder or writer or output RNN is conditioned on that fixed-length vector to generate the output sequence Y = (y(1) , . . . , y(ny)).
- **Training**: two RNNs are trained jointly to maximize the average of logP(y(1),…,y(ny) |x(1),…,x(nx)) over all the pairs of x and y sequences in the training set.

Deep Recurrent Networks

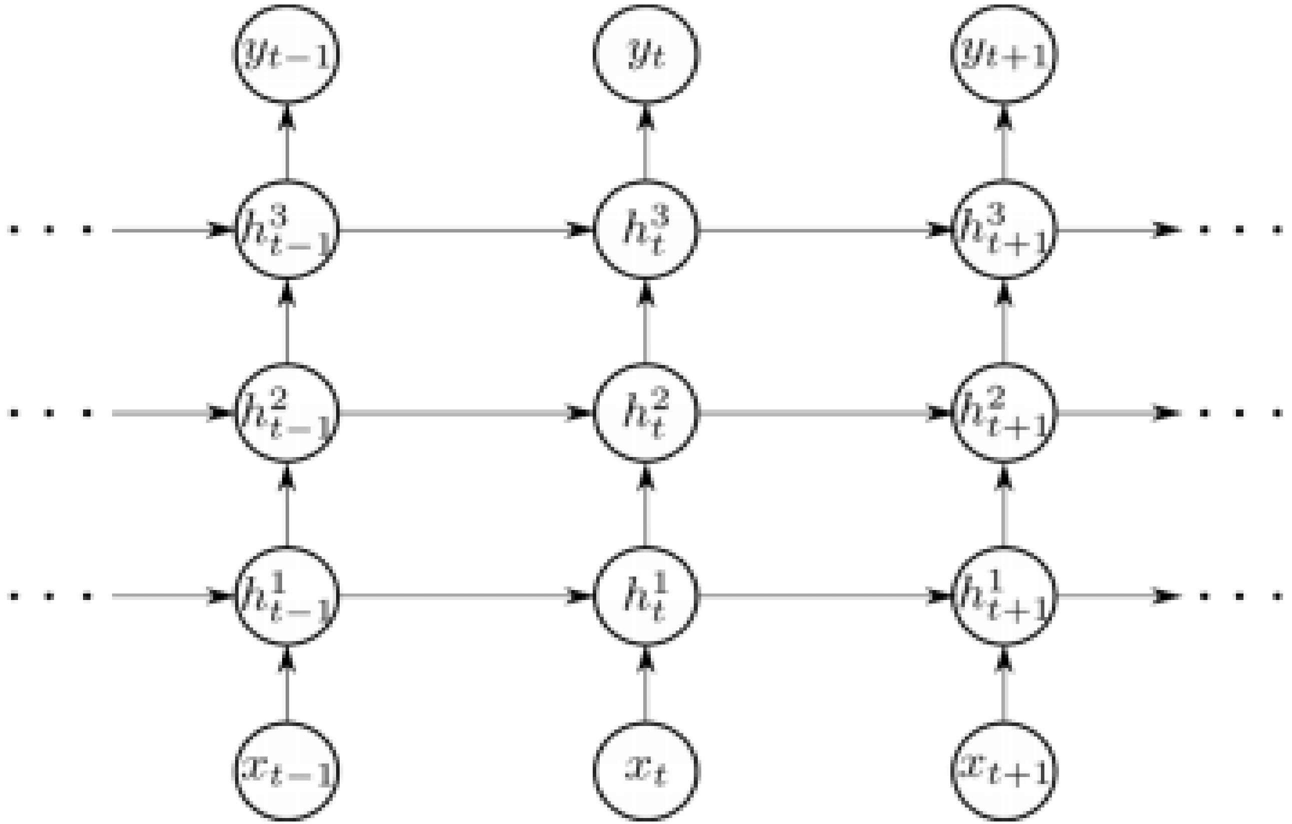
- The computation in most RNNs can be decomposed into three blocks of parameters and associated transformations
  1. From the input to the hidden state
  2. From the previous hidden state to the next hidden state
  3. From the hidden state to the output
- Introduce depth

Deep RNN





Stacked RNN


y_{t-1}
y_t
y_{t+1}
h^3_{t-1}
h^3_t
h^3_{t+1}
h^2_{t-1}
h^2_t
h^2_{t+1}
h^1_{t-1}
h^1_t
h^1_{t+1}
x_{t-1}
x_t
x_{t+1}