Deep Learning
Recurrent Networks : 1
Fall 2022

Instructor: Bhiksha Raj

Which open source project?

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECON
  return segtable;
}
```

Related math. What is it talking about?

Proof. Omitted. □

Lemma 0.1. *Let C be a set of the construction.*
Let C be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram
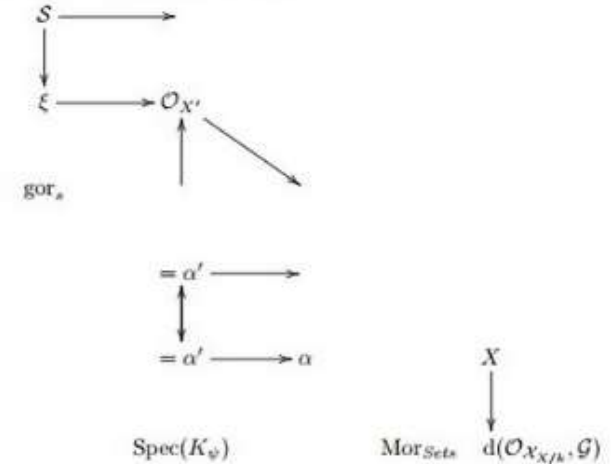
is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings. □

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??. A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad -1(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_{\bar{x}}}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\vee})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

And a Wikipedia page explaining it all

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servicious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS)[http://www.humah.yahoo.com/guardian. cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile.]]

The unreasonable effectiveness of recurrent neural networks..

- All previous examples were *generated* blindly by a *recurrent* neural network..
  - With simple architectures

- http://karpathy.github.io/2015/05/21/rnn-effectiveness/

# Modern text generation is a lot more sophisticated that that

- One of the many sages of the time, the Bodhisattva Bodhisattva Sakyamuni (1575-1611) was a popular religious figure in India and around the world. This Bodhisattva Buddha was said to have passed his life peacefully and joyfully, without passion and anger. For over twenty years he lived as a lay man and dedicated himself toward the welfare, prosperity, and welfare of others. Among the many spiritual and philosophical teachings he wrote, three are most important; the first, titled the "Three Treatises of Avalokiteśvara"; the second, the teachings of the "Ten Questions;" and the third, "The Eightfold Path of Discipline."
  - Entirely randomly generated

Modelling Series

- In many situations one must consider a *series* of inputs to produce an output
  - Outputs too may be a series

- Examples: ..

What did I say?

"To be" or not "to be"??
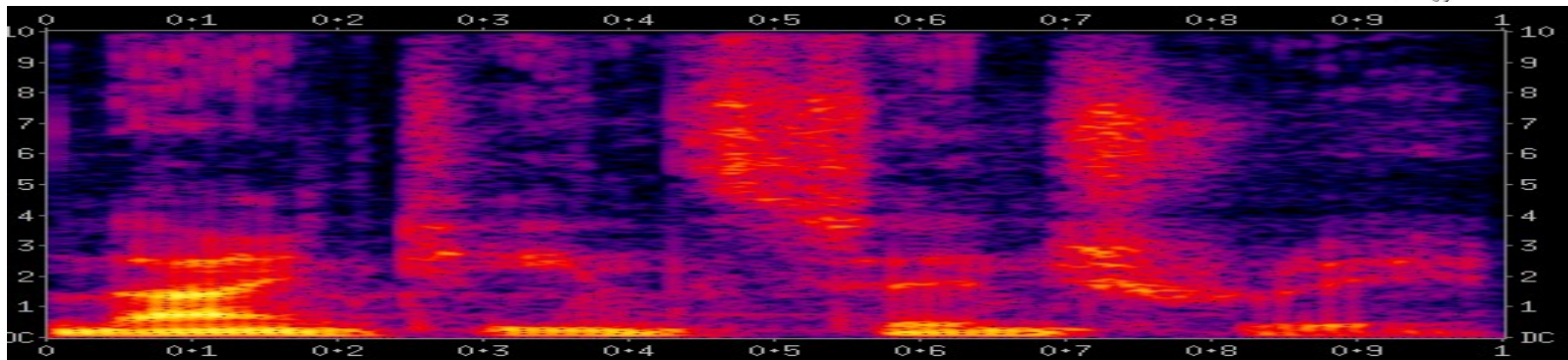
- Speech Recognition
  - Analyze a series of spectral vectors, determine what was said
- Note: Inputs are sequences of vectors. Output is a classification result

What is he talking about?

"Football" or "basketball"?

> The Steelers, meanwhile, continue to struggle to make stops on defense. They've allowed, on average, 30 points a game, and have shown no signs of improving anytime soon.

- Text analysis
  - E.g. analyze document, identify topic
    - Input series of words, output classification output
  - E.g. read English, output French
    - Input series of words, output series of words

Should I invest..

To invest or not to invest?

stocks

7/03 8/03 9/03 10/03 11/03 12/03 13/03 14/03 15/03
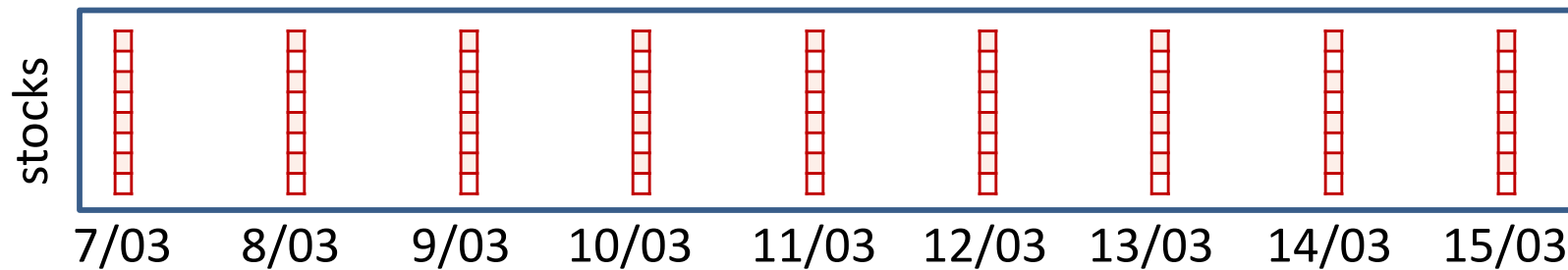
- Note: Inputs are sequences of vectors. Output may be scalar or vector
  - Should I invest, vs. should I not invest in X?
  - Decision must be taken considering how things have fared over time

These are classification and prediction problems

- Consider a sequence of inputs
  - Input vectors

- Produce one or more outputs

- This can be done with neural networks
  - Obviously

Representational shortcut
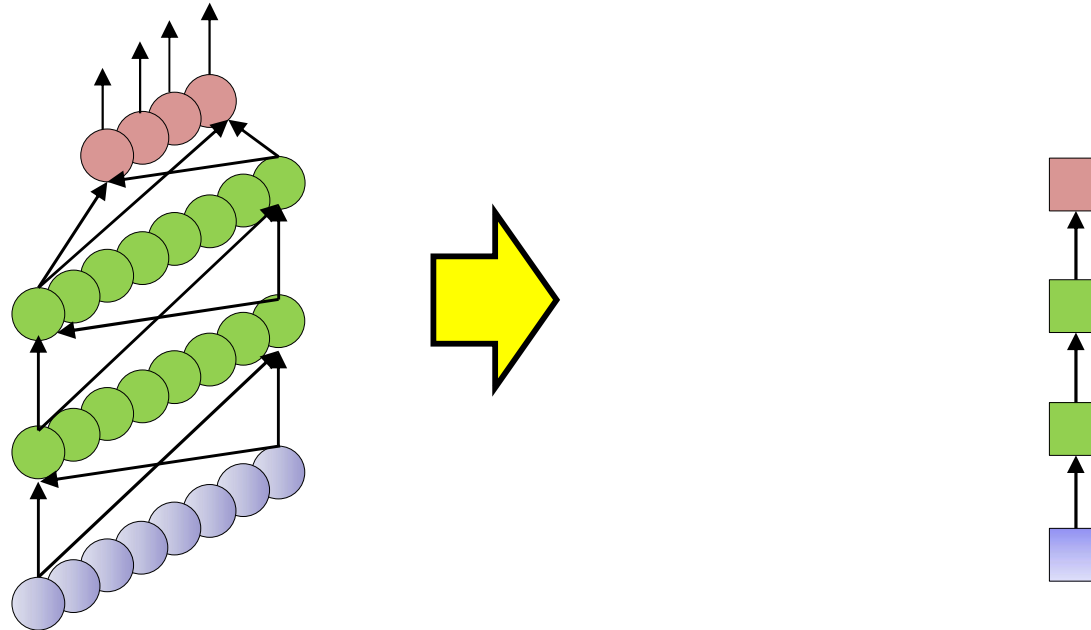
- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything by simple boxes
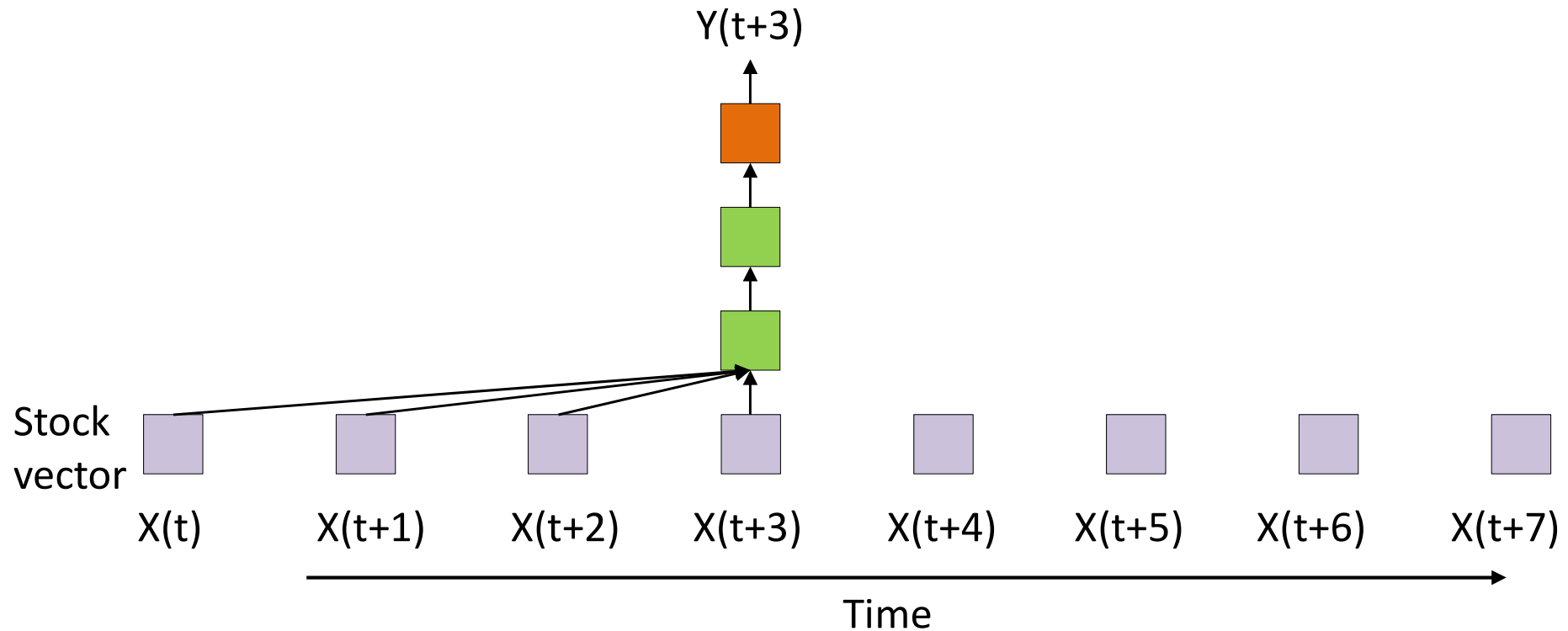 - Each box actually represents an entire *layer with many units*

Representational shortcut
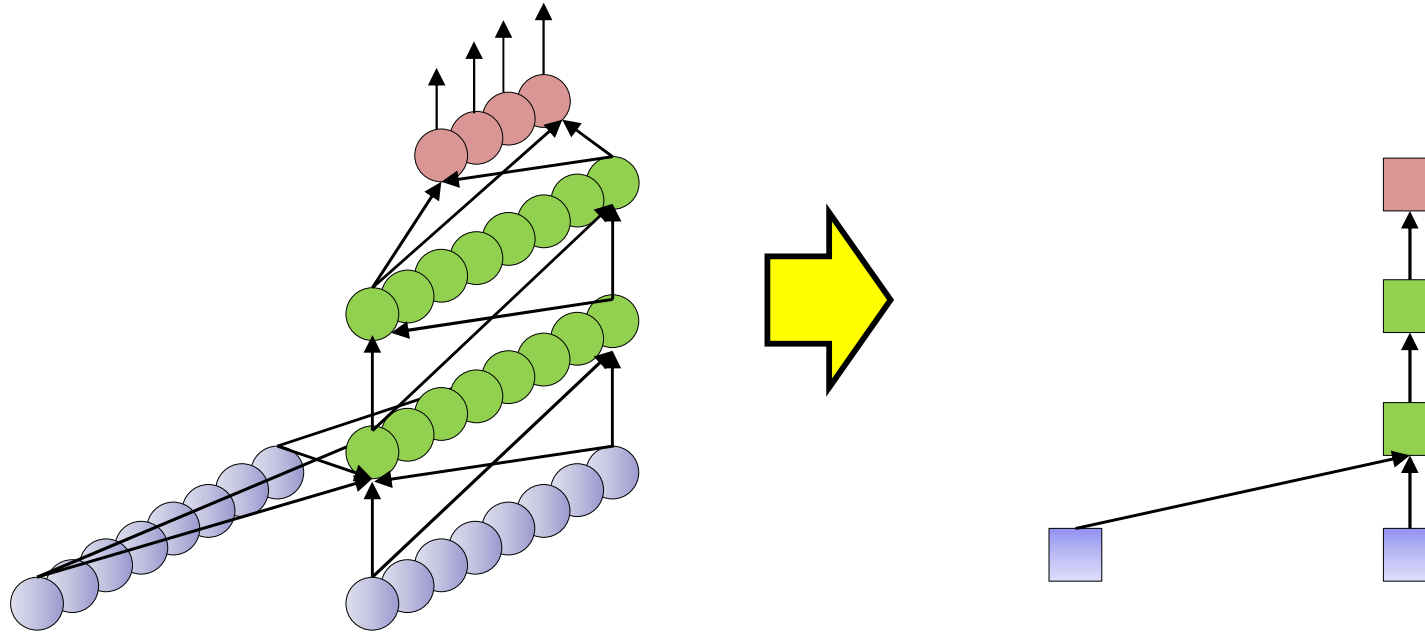
- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything by simple boxes
  - Each box actually represents an entire *layer with many units*

Representational shortcut

- Input at each time is a *vector*

- Each layer has many neurons
  - Output layer too may have many neurons

- But will represent everything as simple boxes
  - Each box actually represents an entire *layer with many units*

The stock prediction problem...

To invest or not to invest?

stocks

7/03 8/03 9/03 10/03 11/03 12/03 13/03 14/03 15/03
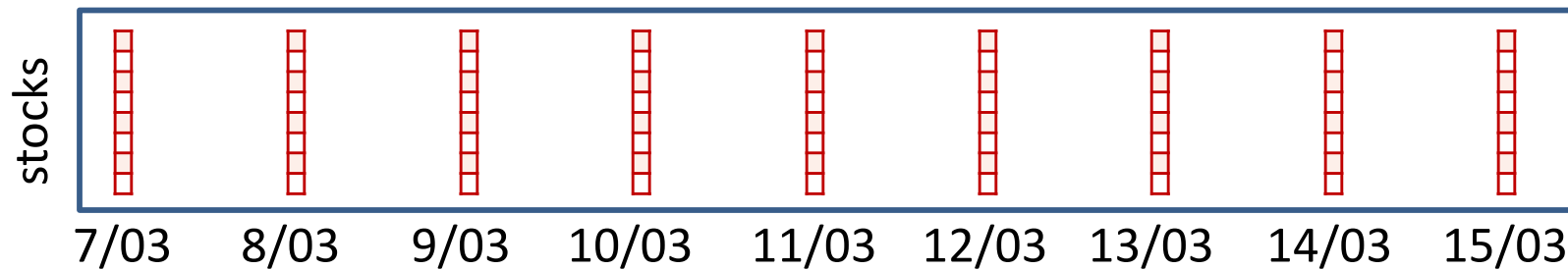
- Stock market
  - Must consider the series of stock values in the past several days to decide if it is wise to invest today

The stock predictor network

Y(t+3)

Stock
vector

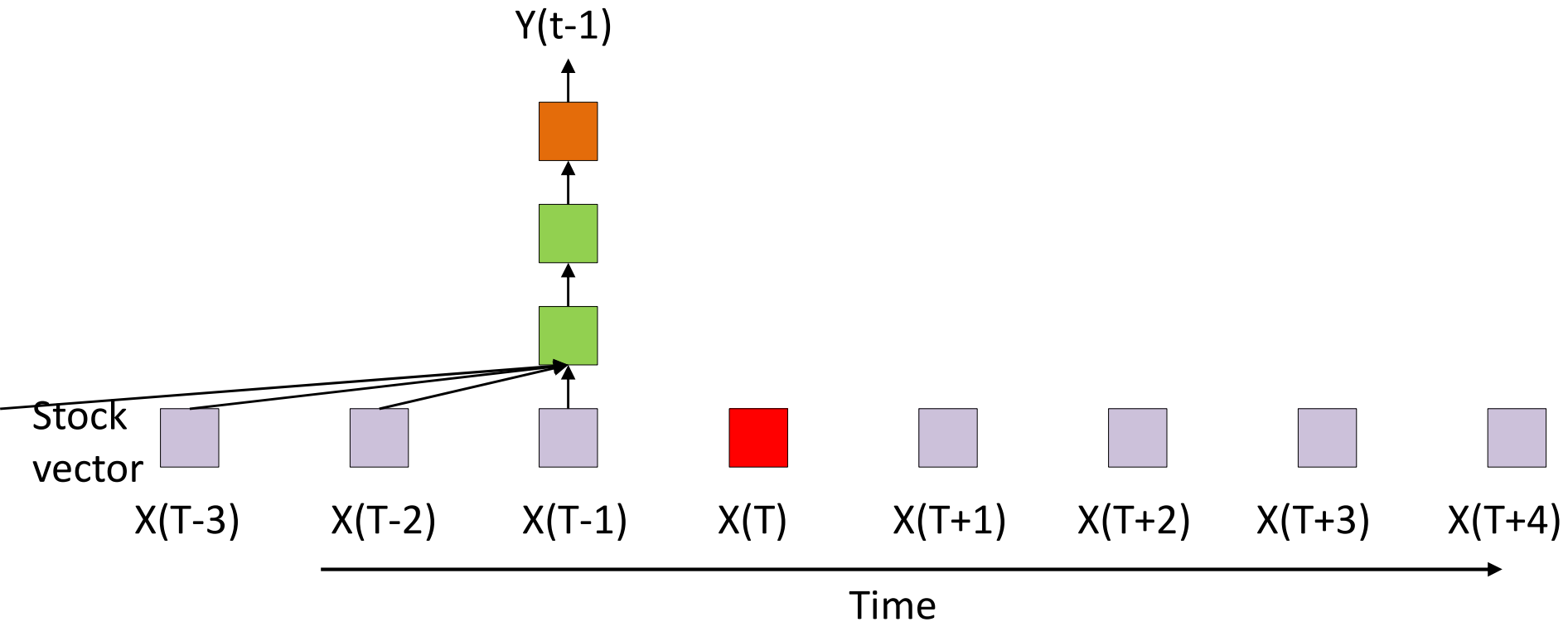X(t)　　X(t+1)　　X(t+2)　　X(t+3)　　X(t+4)　　X(t+5)　　X(t+6)　　X(t+7)

Time

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

The stock predictor network

- ## The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

The stock predictor network

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

The stock predictor network

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

The stock predictor network

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

Finite-response model

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for N days into the future
    - N is the *width* of the system
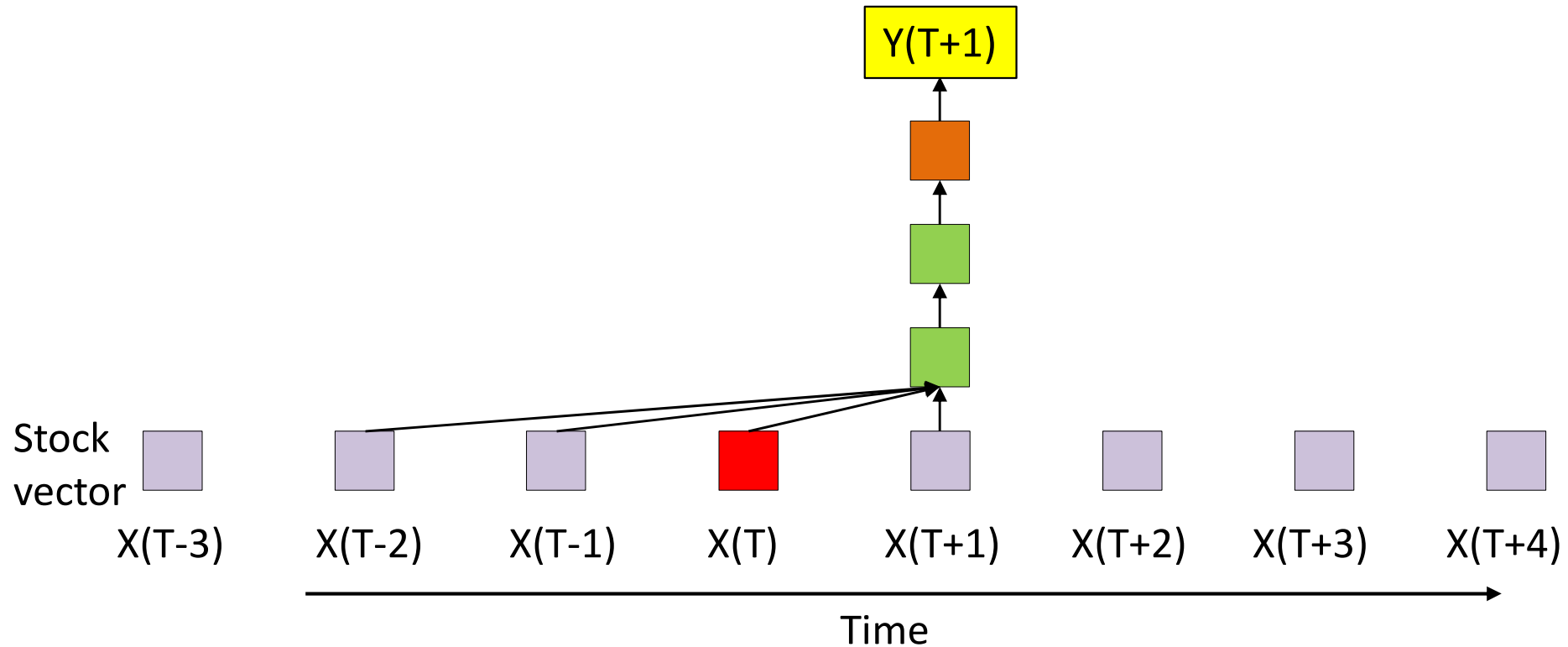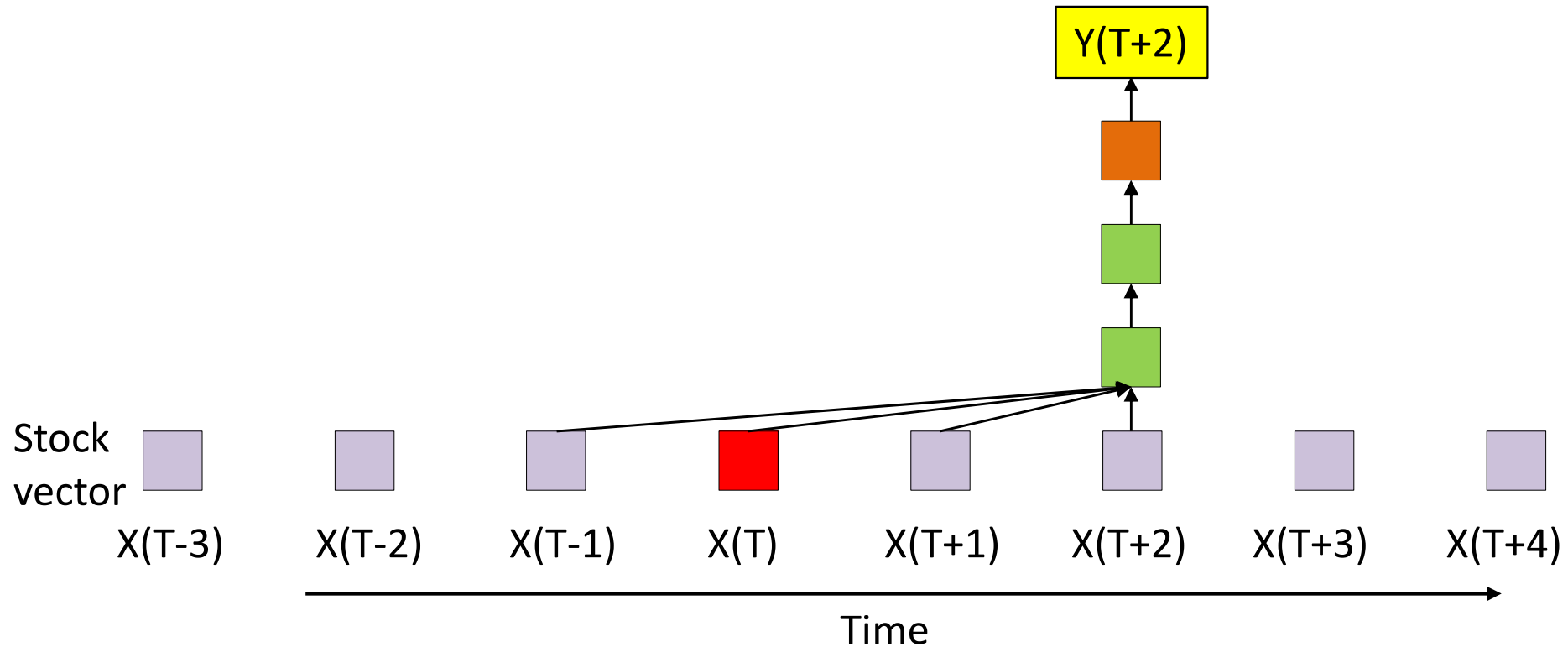
$$Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})$$

The stock predictor

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for N days into the future
    - N is the *width* of the system
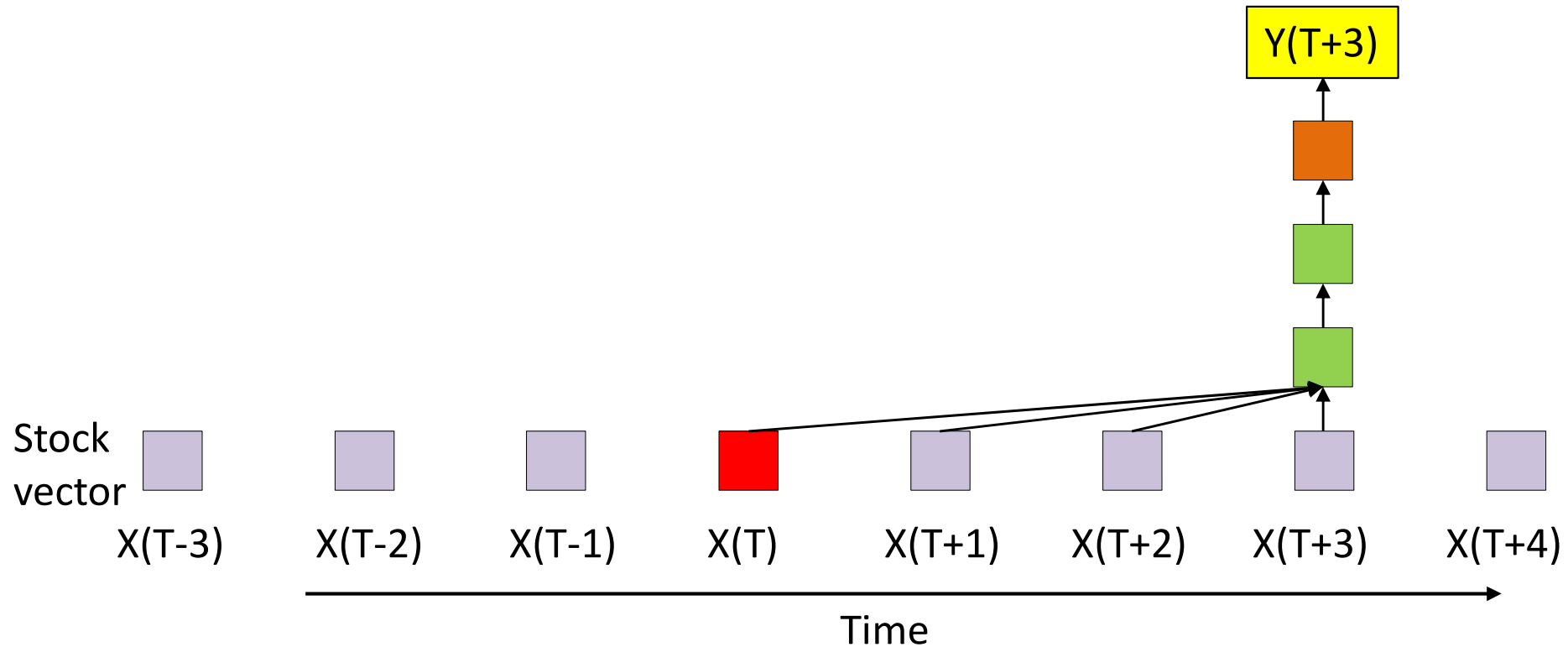
$$Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})$$

The stock predictor

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for N days into the future
    - N is the *width* of the system
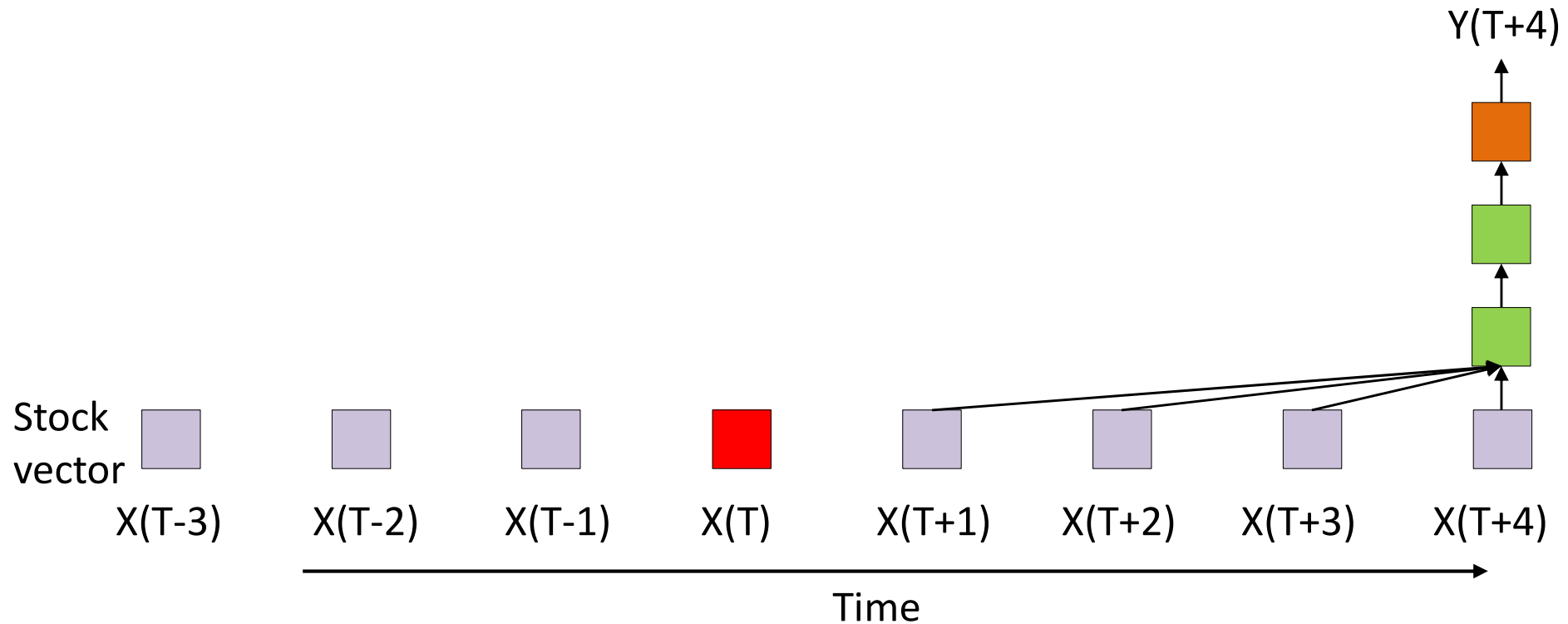
$$Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})$$

The stock predictor

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for N days into the future
    - N is the *width* of the system

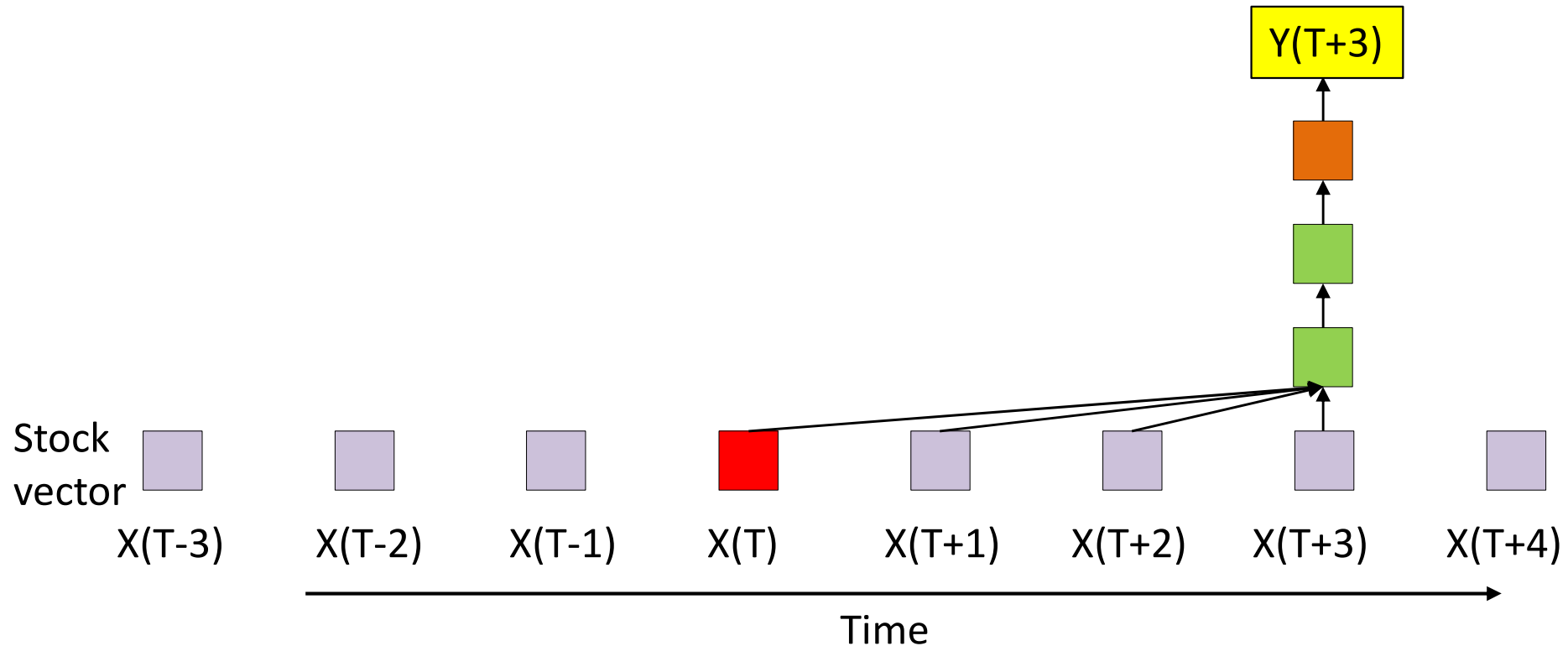$$Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})$$

The stock predictor

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for N days into the future
    - N is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, ..., X_{t-N})$$

The stock predictor

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for N days into the future
    - N is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})$$

The stock predictor

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for N days into the future
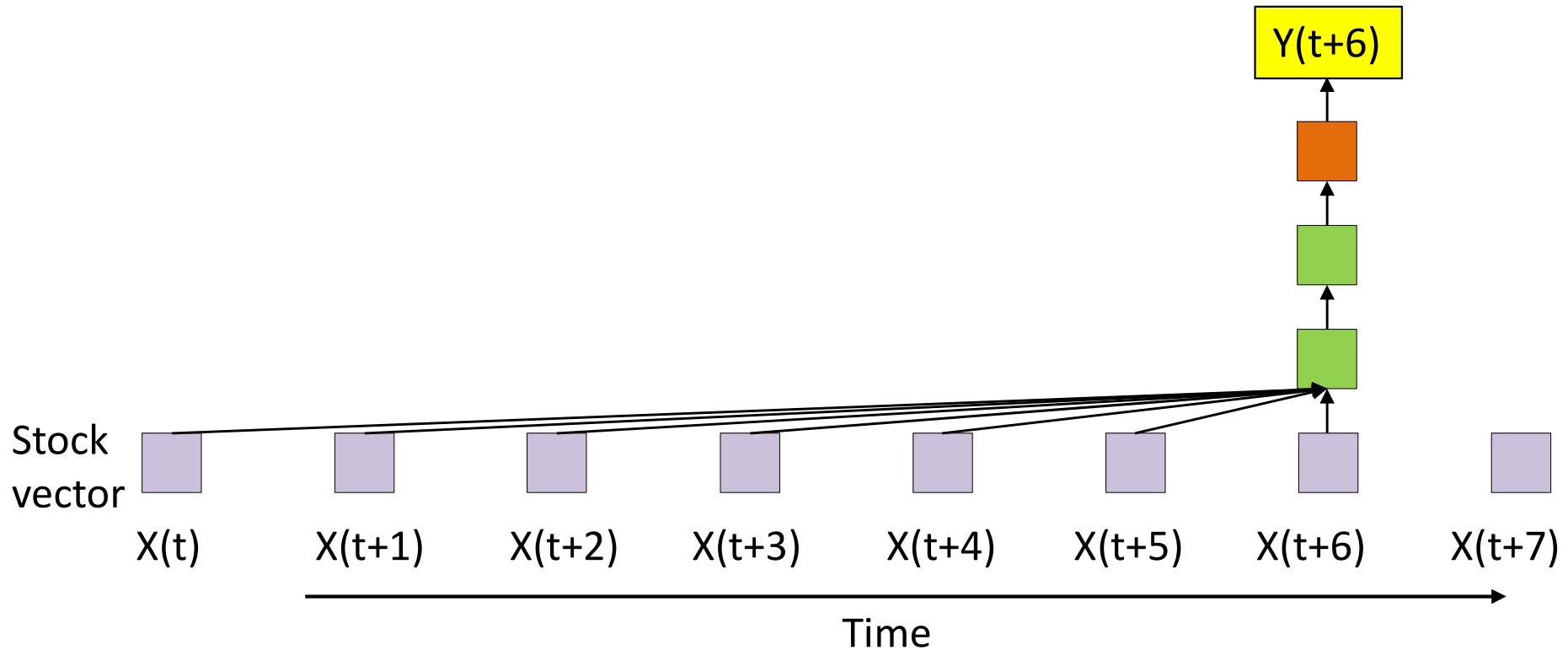   - N is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})$$

Finite-response model

- Something that happens *today* only affects the output of the system for N days into the future
  - **Predictions consider *N* days of history**

- To consider more of the past to make predictions, you must increase the "history" considered by the system
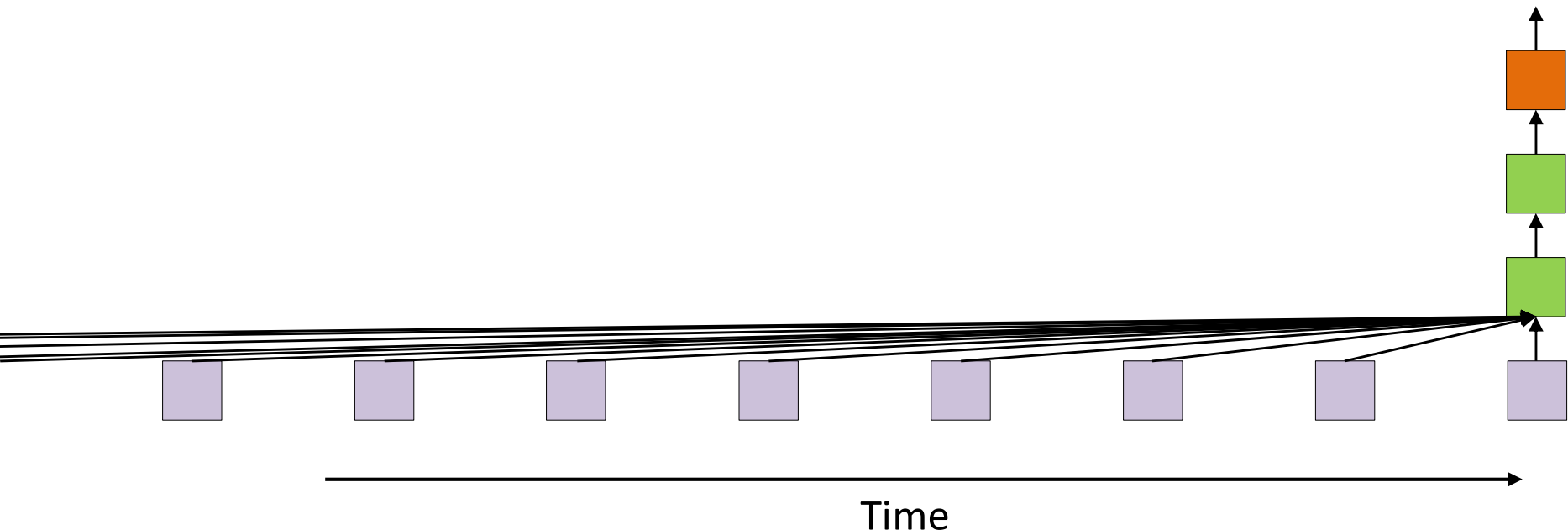
Finite-response

- Problem: Increasing the "history" makes the network more complex
  - No worries, we have the CPU and memory
    - Or do we?

Systems often have long-term dependencies

- Longer-term trends –
  - Weekly trends in the market
  - Monthly trends in the market
  - Annual trends
  - Though longer historic tends to affect us less than more recent events..

We want *infinite* memory

Time

- Required: *Infinite* response systems
  - What happens today can continue to affect the output forever
    - Possibly with weaker and weaker influence

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-\infty})$$

Poll 1

- @864, @865

Slide 32: Finite response system vs infinite response system

Convolutional neural networks are finite response systems, true or false

- True
- False

An input at time T affects the output of the convolutional layers of the network for all time, true or false

- True
- False

Poll 1

Convolutional neural networks are finite response systems, true or false

- **True**
- False

An input at time T affects the output of the convolutional layers of the network for all time, true or false

- True
- **False**

# Examples of infinite response systems

$$Y_t = f(X_t, Y_{t-1})$$

– Required: Define initial state: Y_{-1} for $t = 0$

– An input at X_0 at $t = 0$ produces Y_0

– Y_0 produces Y_1 which produces Y_2 and so on until Y_∞ *even if* $X_1 \ldots X_\infty$ *are 0*

   • i.e. even if there are no further inputs!

– **A single input influences the output for the rest of time!**

• This is an instance of a NARX network

   – "nonlinear autoregressive network with exogenous inputs"

   – $Y_t = f(X_{0:t}, Y_{0:t-1})$

• *Output* contains information about the entire past
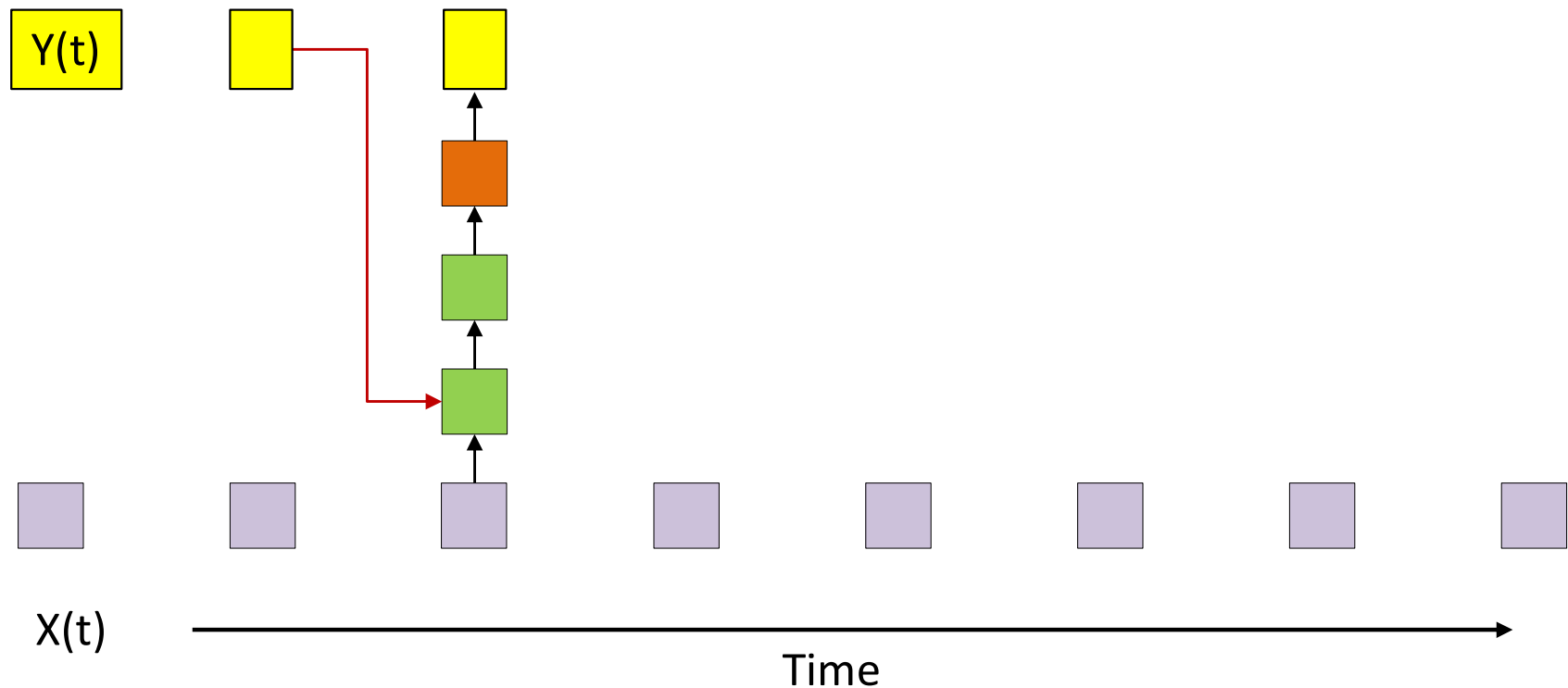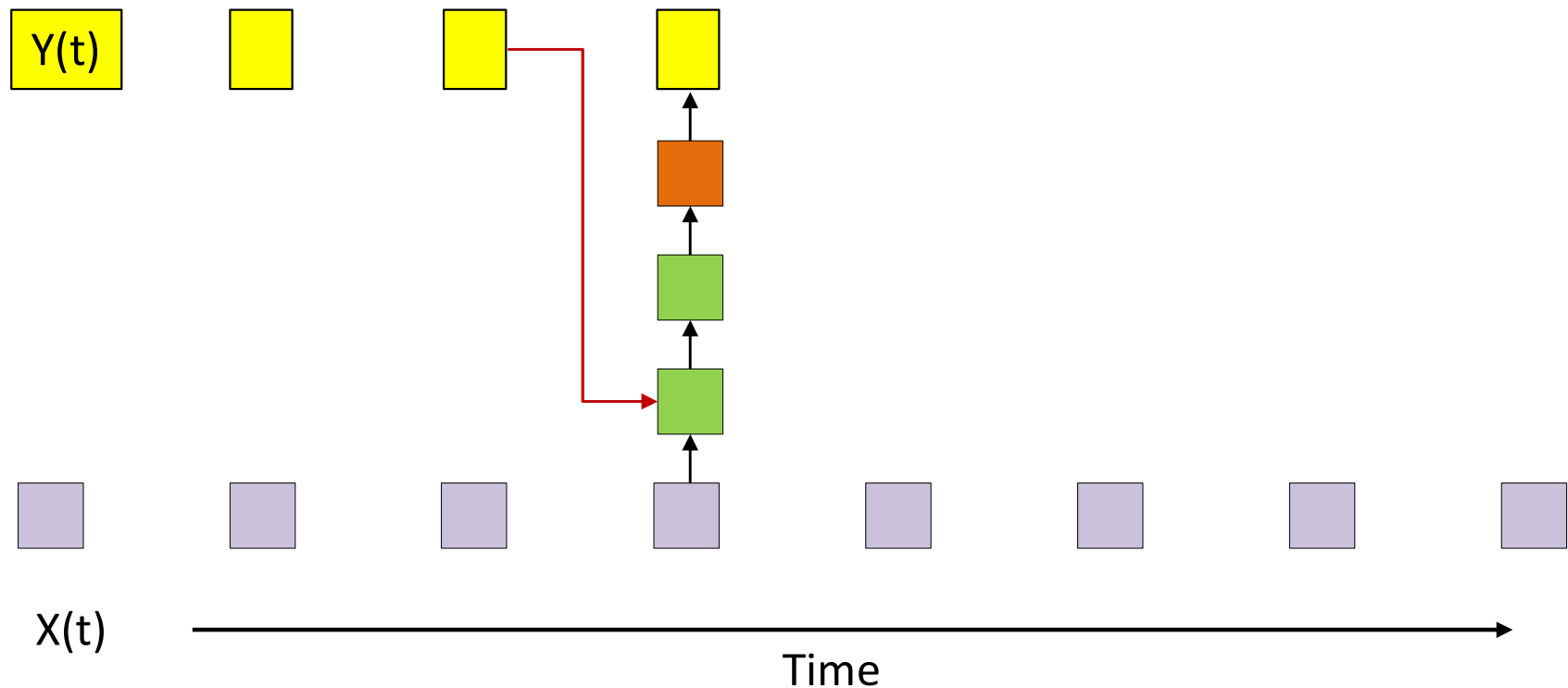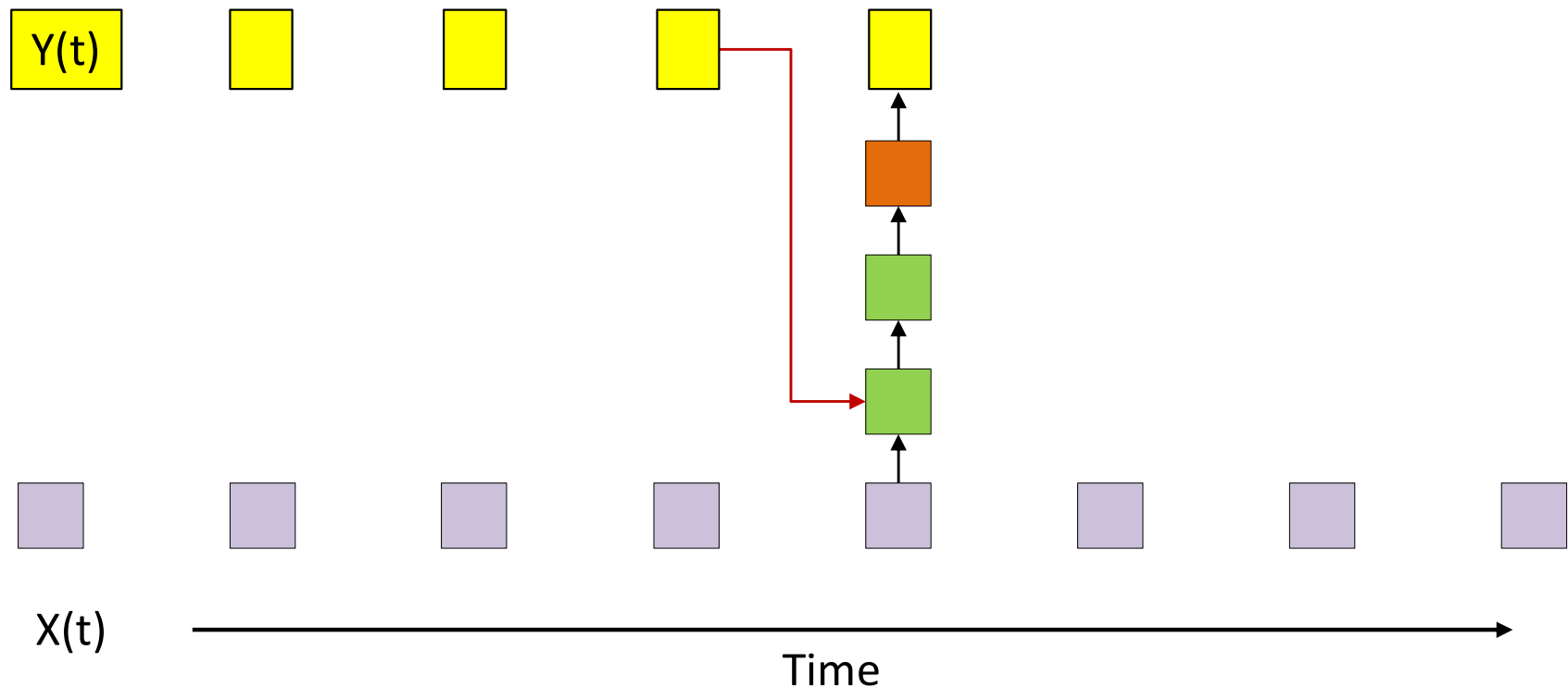
A one-tap NARX network
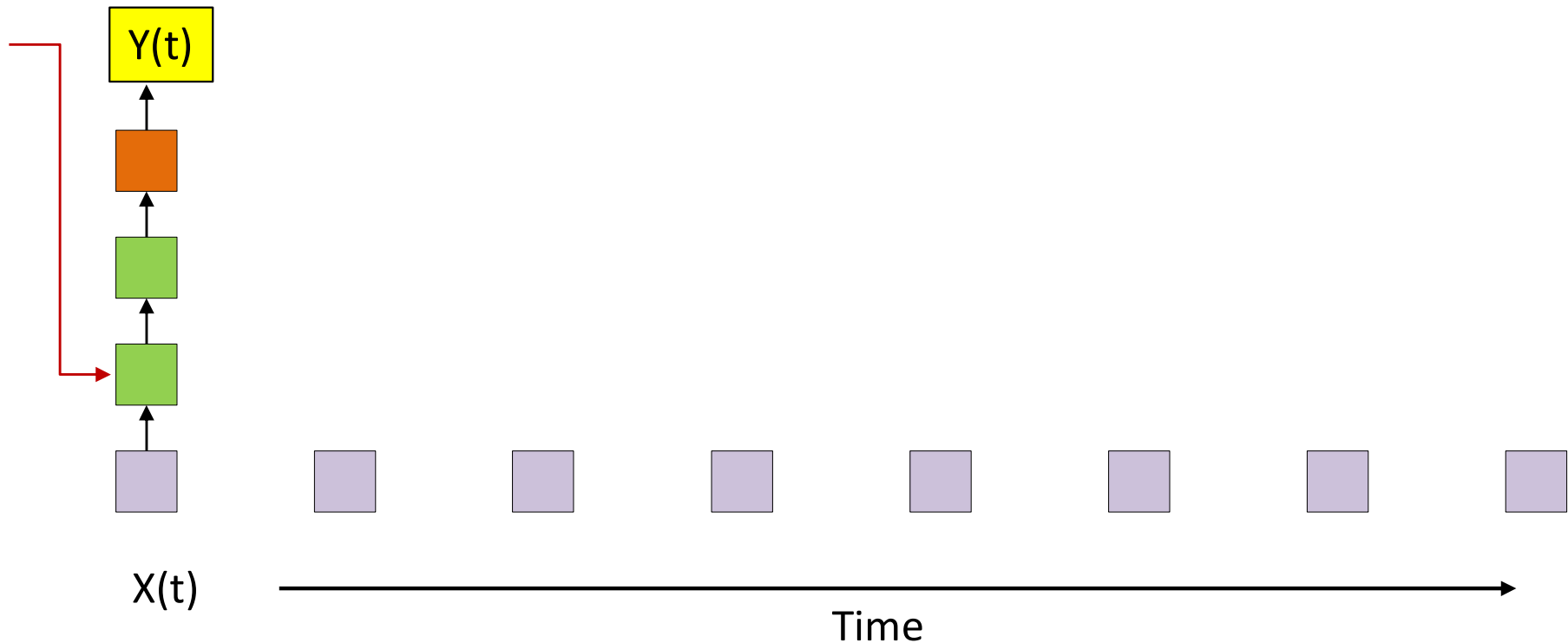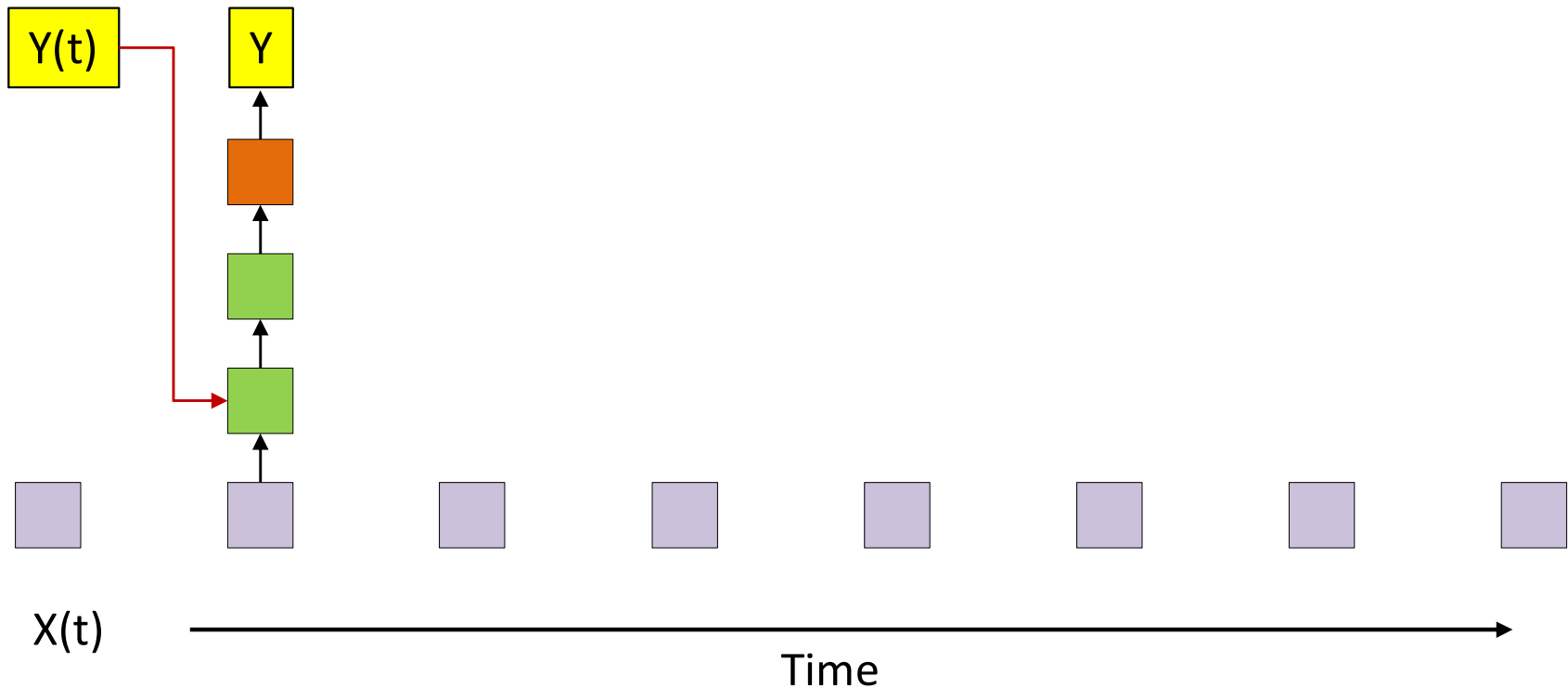
- A NARX net with recursion from the output

A one-tap NARX network

- A NARX net with recursion from the output

A one-tap NARX network

- A NARX net with recursion from the output
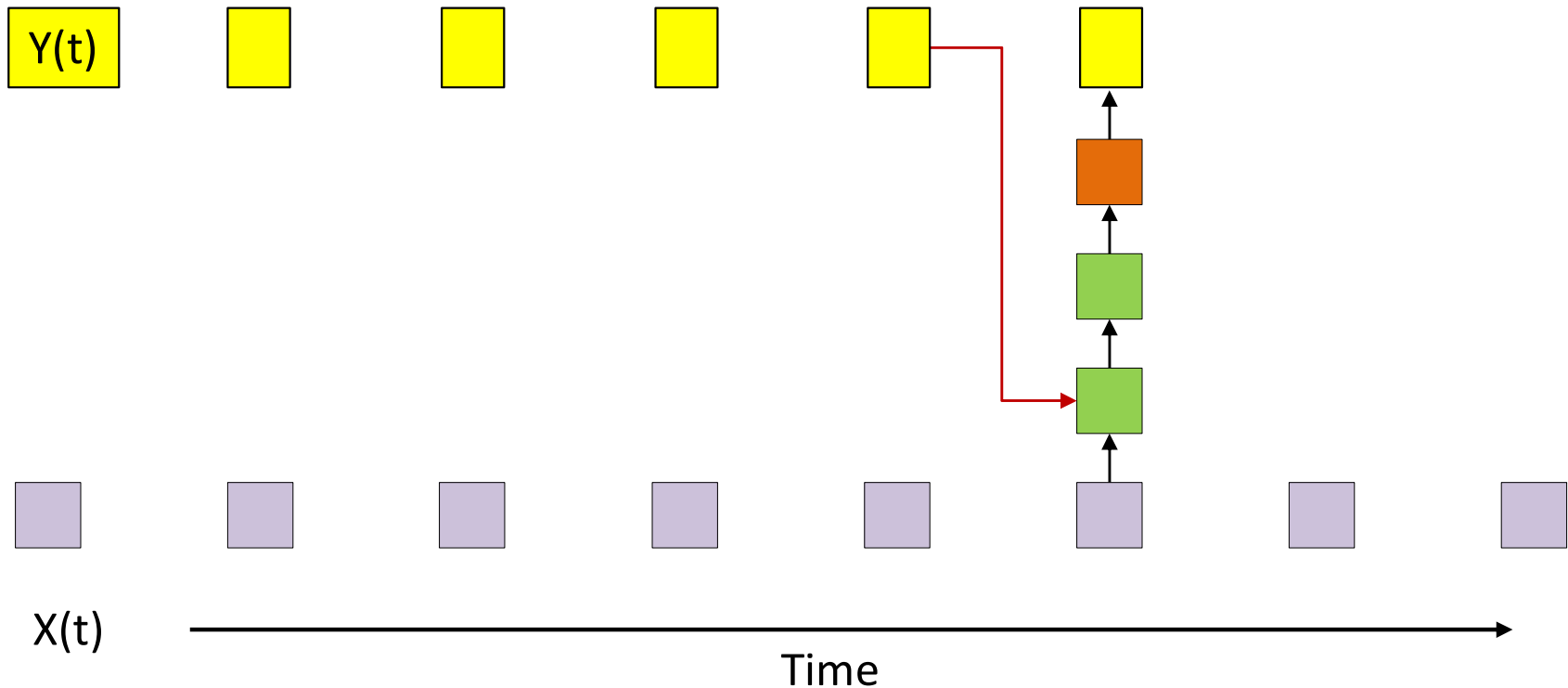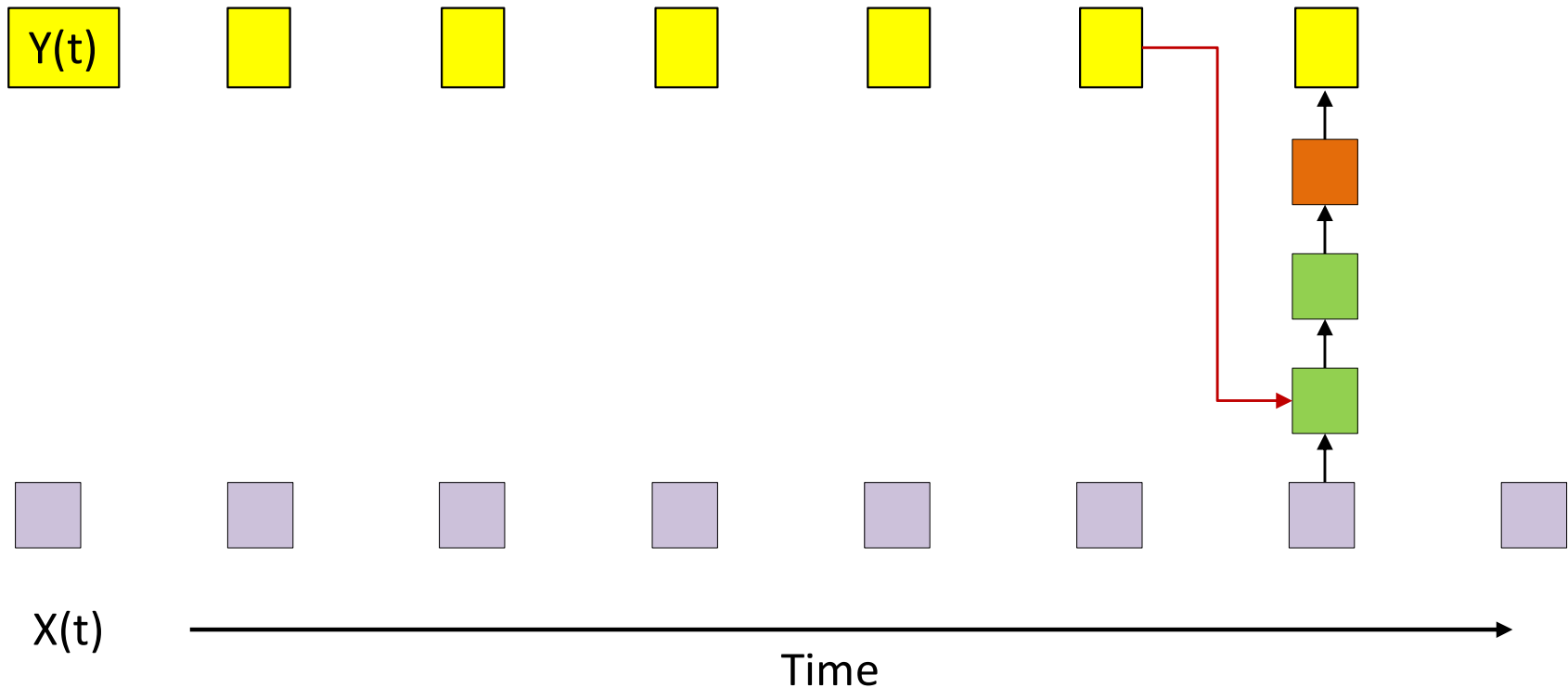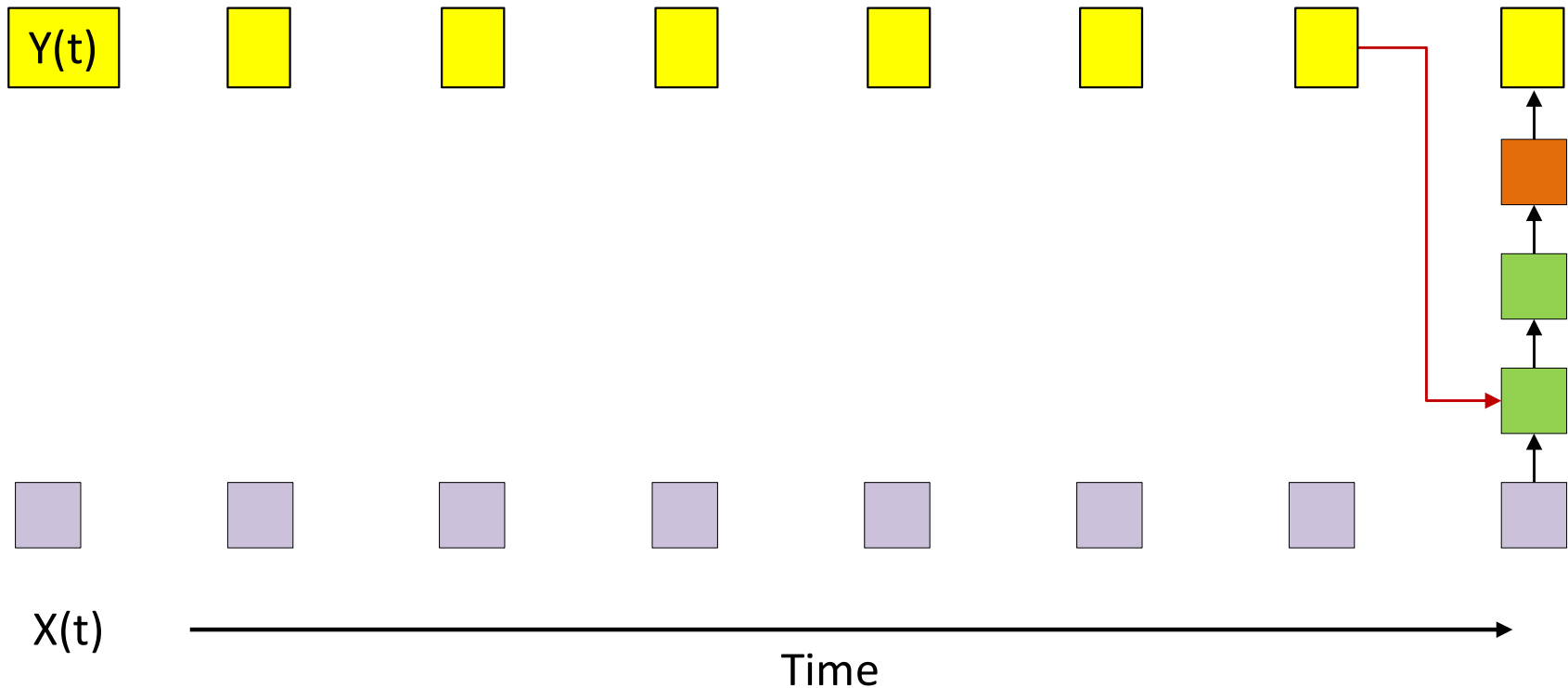
A one-tap NARX network

- A NARX net with recursion from the output

A one-tap NARX network

- A NARX net with recursion from the output
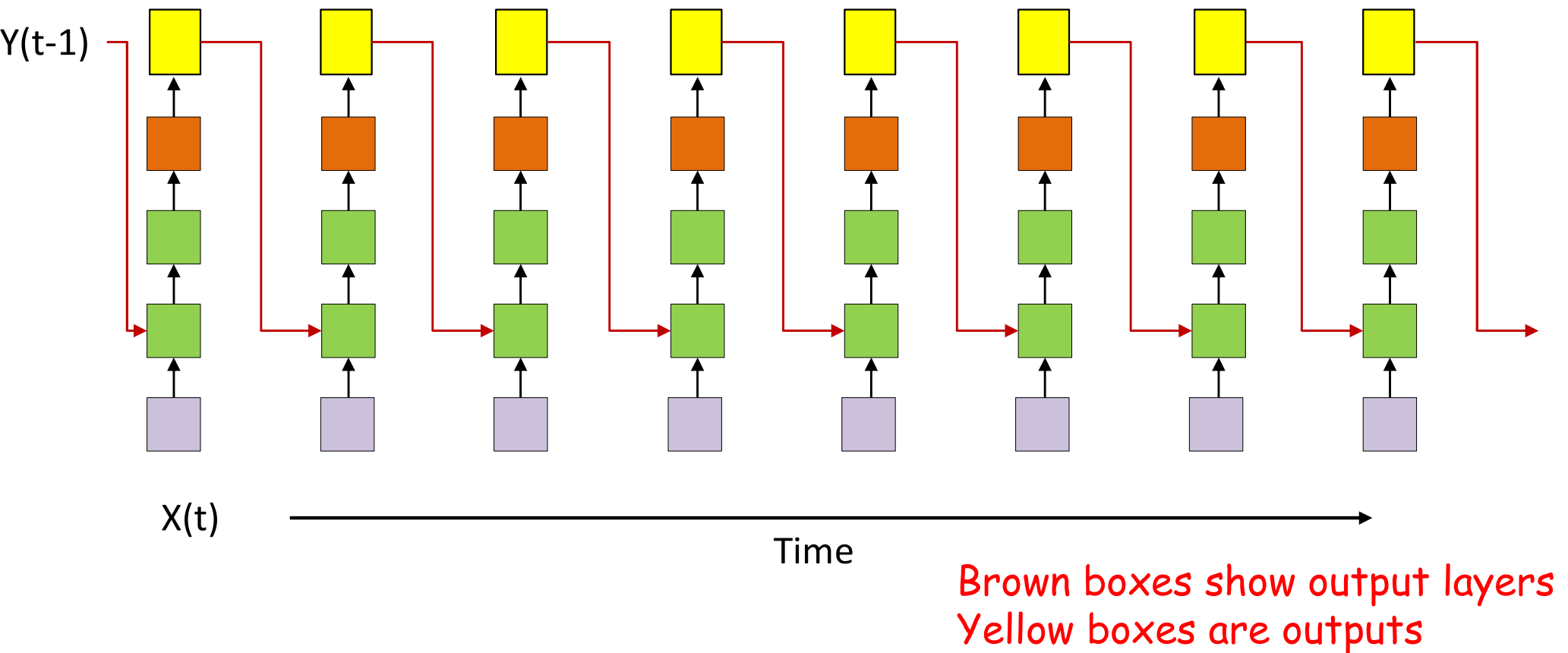
A one-tap NARX network

- A NARX net with recursion from the output

A one-tap NARX network

- A NARX net with recursion from the output

A one-tap NARX network

Y(t)

X(t)

Time

- A NARX net with recursion from the output
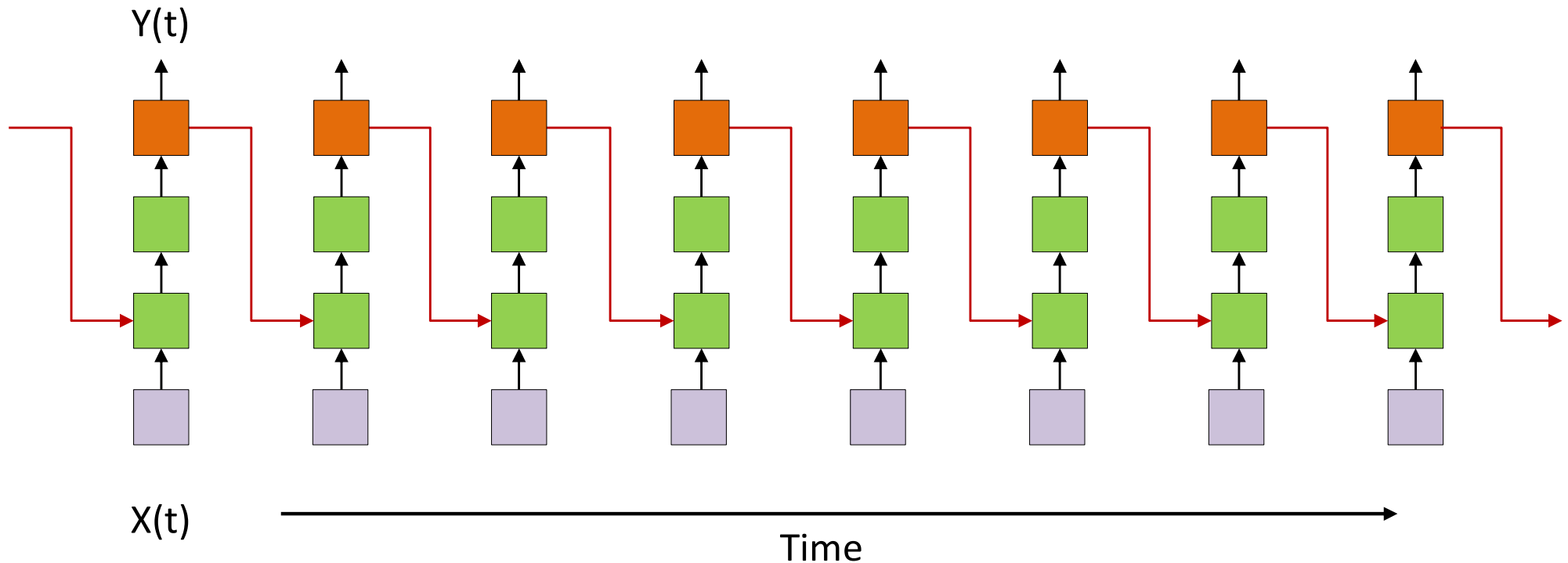
A more complete representation

Brown boxes show output layers
Yellow boxes are outputs

- A NARX net with recursion from the output
- Showing all computations
- All columns are identical
- *An input at t=0 affects outputs forever*

43

Same figure redrawn

Y(t)

X(t)

Time

Brown boxes show output layers
All outgoing arrows are the same output

- A NARX net with recursion from the output

- Showing all computations

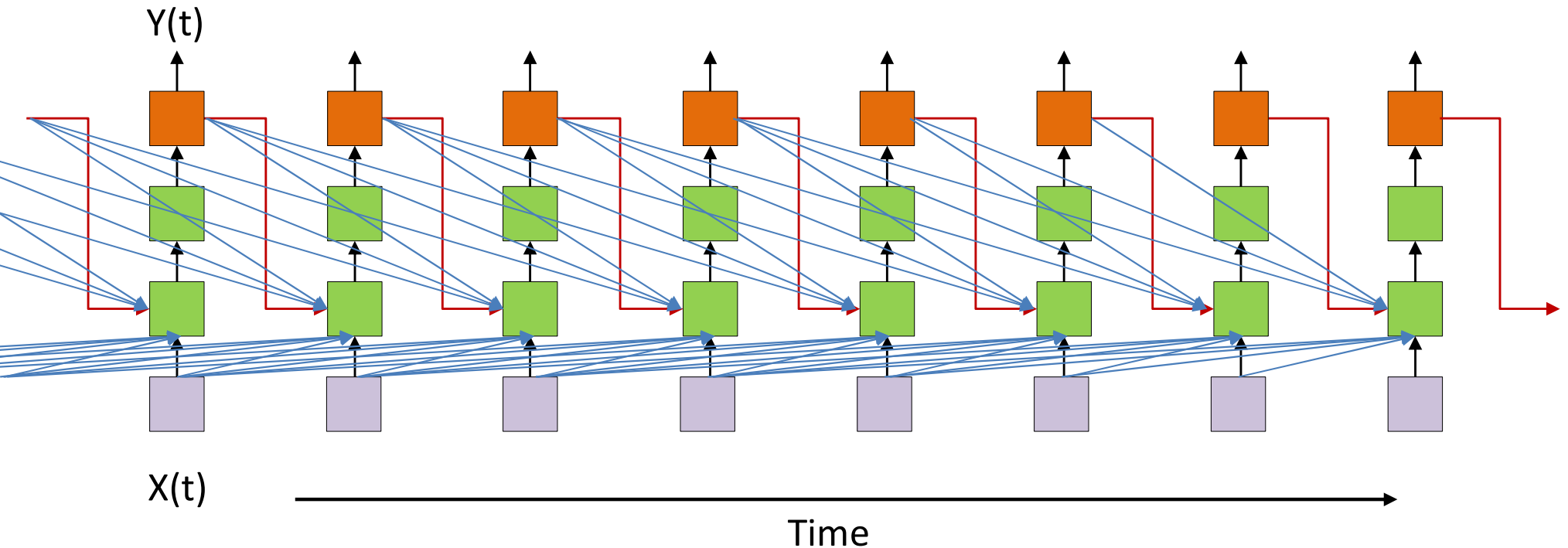- All columns are identical

- *An input at t=0 affects outputs forever*

A more generic NARX network

- The output Y_t at time t is computed from the past K outputs $Y_{t-1}, \ldots, Y_{t-K}$ and the current and past L inputs $X_t, \ldots, X_{t-L}$

45

NARX Networks

- Very popular for time-series prediction
  - Weather
  - Stock markets
  - As alternate system models in tracking systems

- Any phenomena with distinct "innovations" that "drive" an output

- Note: here the "memory" of the past is in the output itself, and not in the network

Let's make memory more explicit

- Task is to "remember" the past

- Introduce an explicit *memory* variable whose *job* it is to remember
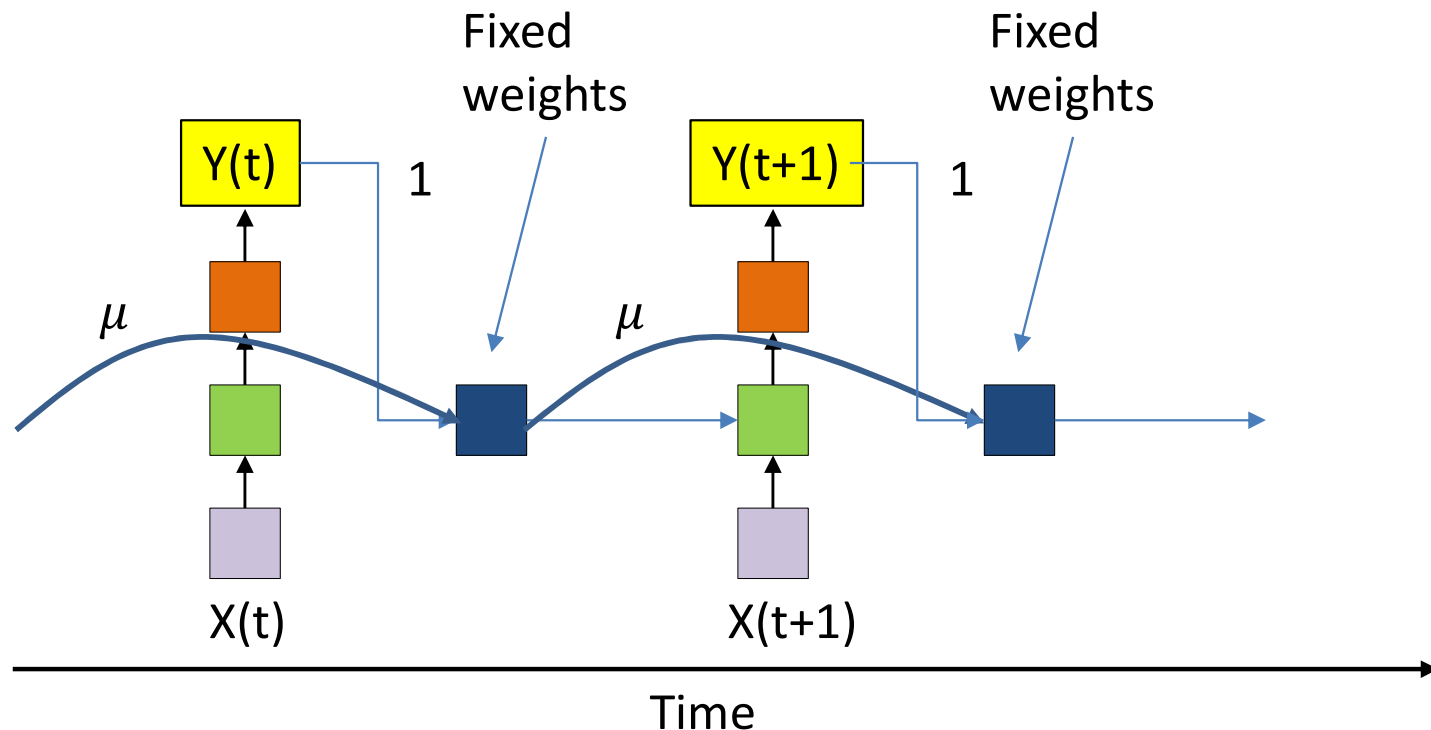
$$m_t = r(y_{t-1}, h_{t-1}, m_{t-1})$$
$$h_t = f(x_t, m_t)$$
$$y_t = g(h_t)$$

- m_t is a "memory" variable
  - Generally stored in a "memory" unit
  - Used to "remember" the past

Jordan Network

- Memory unit simply retains a running average of past outputs
  - "Serial order: A parallel distributed processing approach", M.I.Jordan, 1986
    - Input is constant (called a "plan")
    - Objective is to train net to produce a specific output, given an input plan
  - Memory has fixed structure; does not "learn" to remember
    - The running average of outputs considers entire past, rather than immediate past

Elman Networks
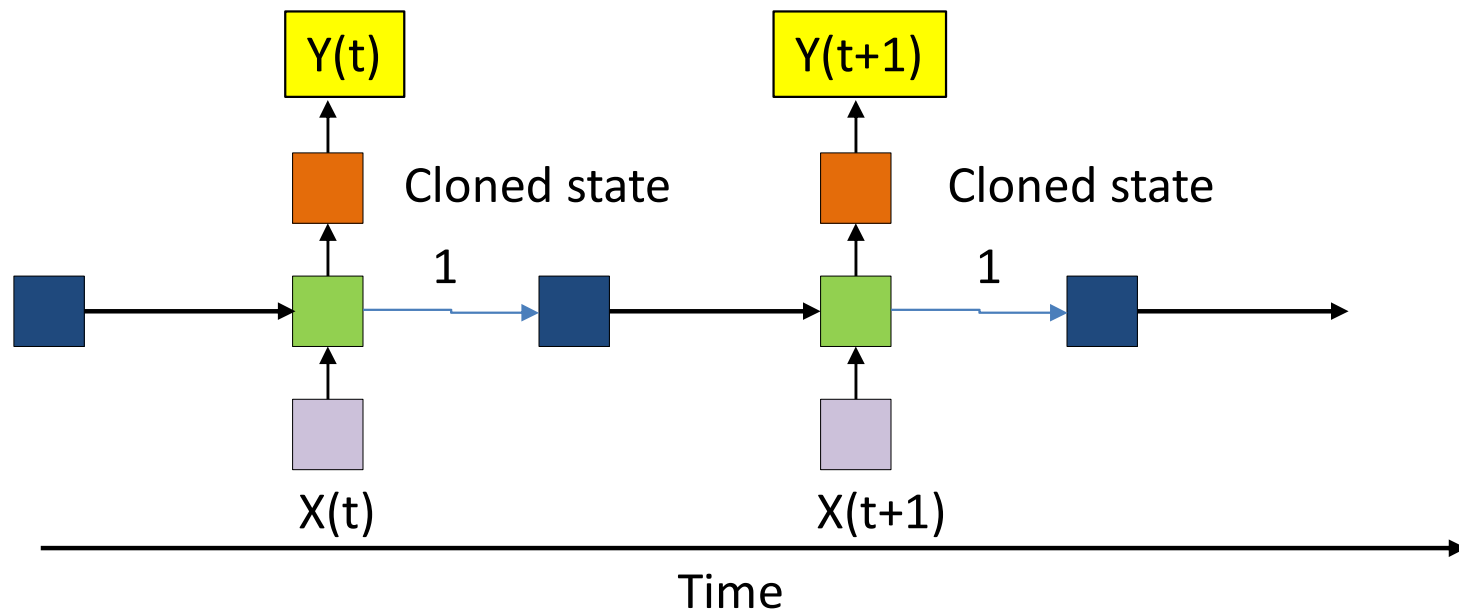
- Separate memory state from output
  - "Context" units that carry historical state
  - "Finding structure in time", Jeffrey Elman, Cognitive Science, 1990
    - For the purpose of training, this was approximated as a set of T independent 1-step history nets
- Only the weight *from* the memory unit to the hidden unit is learned
  - But during training no gradient is backpropagated over the "1" link

Story so far

- In time series analysis, models must look at past inputs along with current input
  - Looking at a finite horizon of past inputs gives us a convolutional network
- Looking into the infinite past requires recursion

- NARX networks recurse by feeding back the output to the input
  - May feed back a finite horizon of outputs

- "Simple" recurrent networks:
  - Jordon networks maintain a running average of outputs in a "memory" unit
  - Elman networks store *hidden* unit values for one time instant in a "context" unit
  - "Simple" (or partially recurrent) because during *learning* current error does not actually propagate to the past
    - "Blocked" at the memory units in Jordan networks
    - "Blocked" at the "context" unit in Elman networks

Poll 2

- @866, @867

Memory neuron models have true recurrence, true or false

- True
- False

Memory neuron networks dedicate neurons specifically to store past history, true or false

- True
- False

Poll 2

Memory neuron models have true recurrence, true or false

- True
- **False**

Memory neuron networks dedicate neurons specifically to store past history, true or false
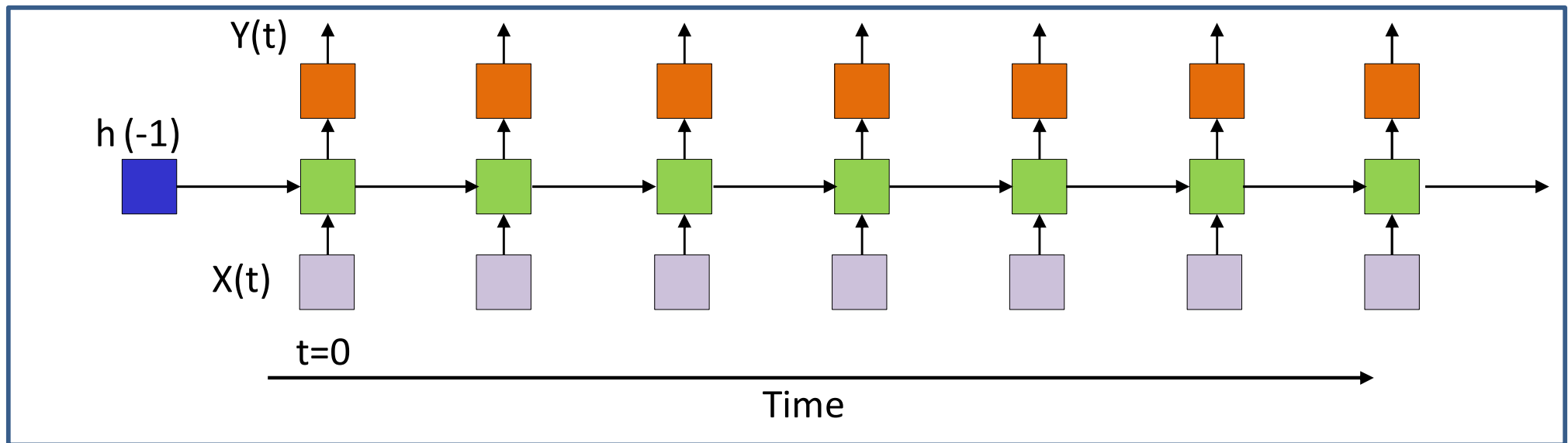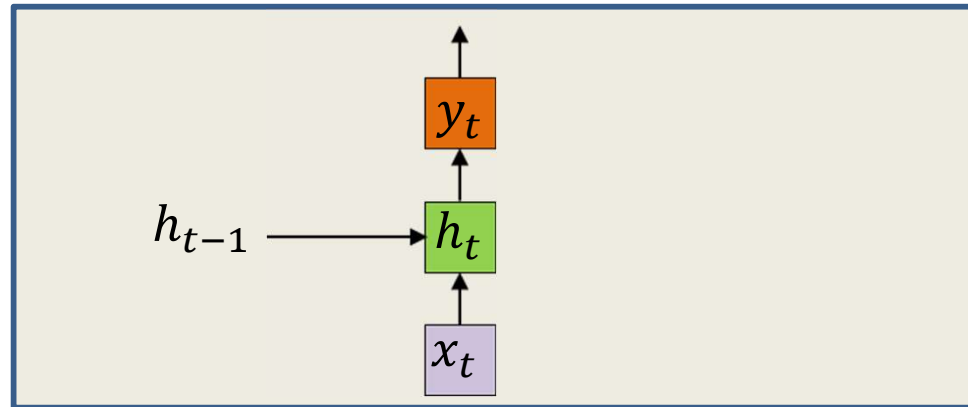
- **True**
- False

An alternate model for infinite response systems: the state-space model

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

- h_t is the *state* of the network
  - *State* summarizes information about the entire past
    - Model directly embeds the memory in the state
- Need to define initial state h_{-1}

- This is a *fully recurrent* neural network
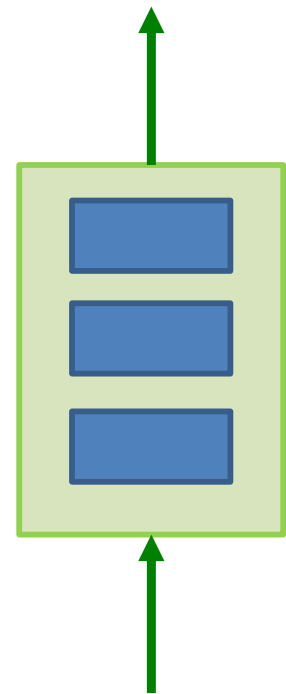 - Or simply a *recurrent neural network*

The simple state-space model

- The state (green) at any time is determined by the input at that time, and the state at the previous time

- *An input at t=0 affects outputs forever*

- Also known as a recurrent neural net
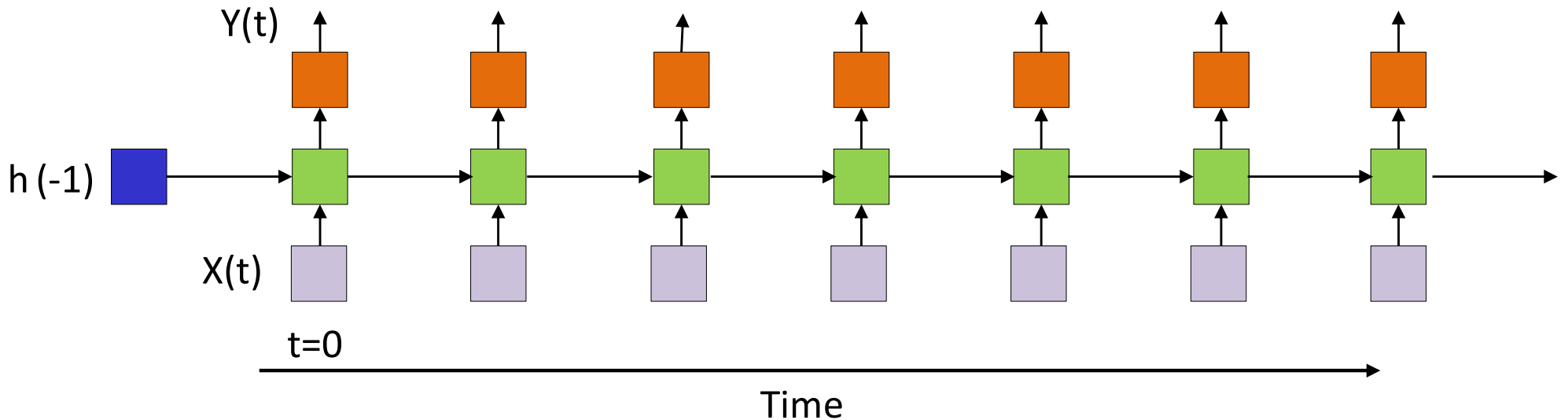
An alternate model for infinite response systems: the state-space model

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

- h_t is the *state* of the network

- Need to define initial state h_{-1}
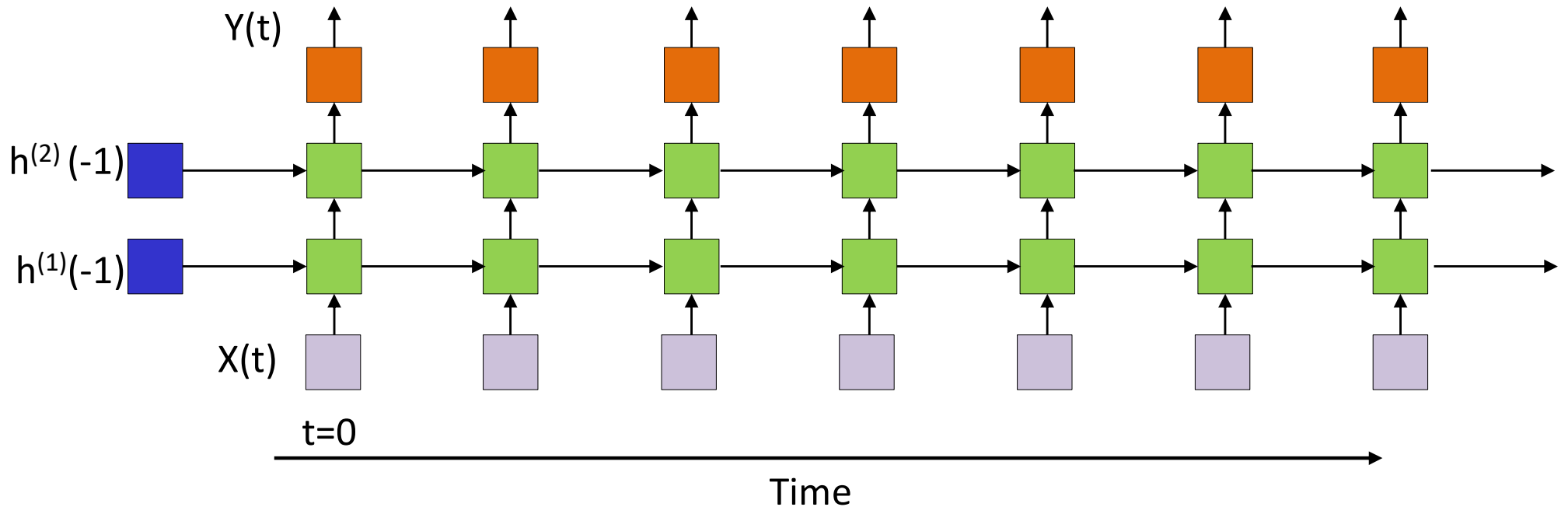
- The state an be arbitrarily complex

Single hidden layer RNN
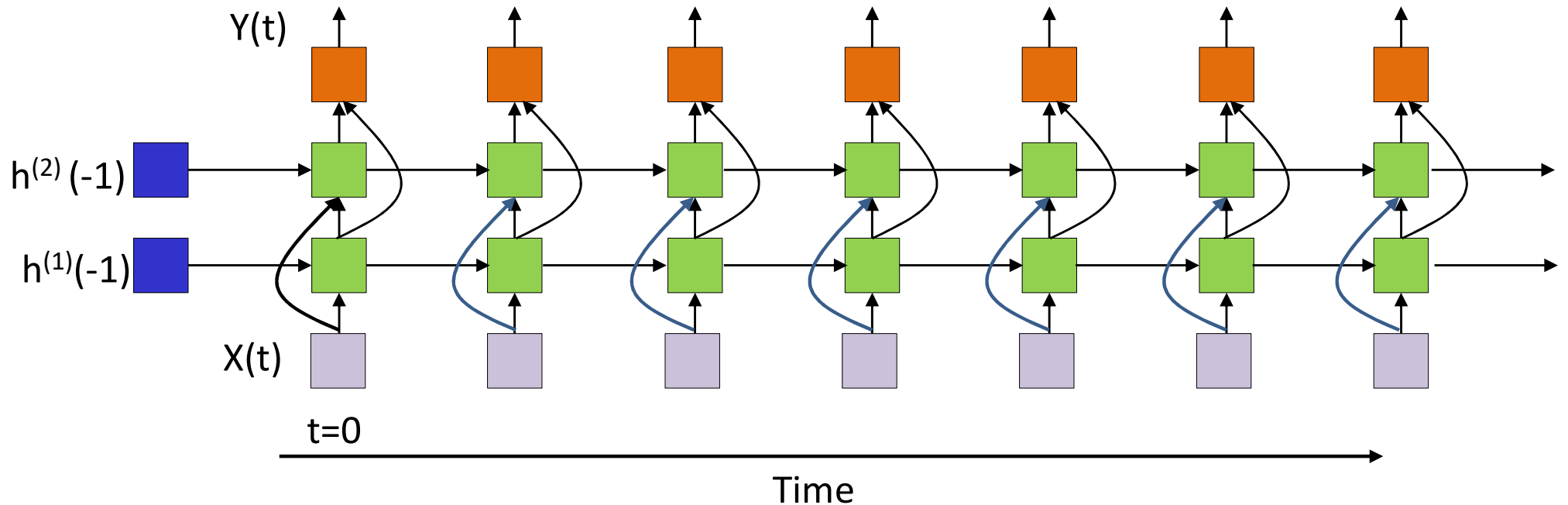

- Recurrent neural network
- All columns are identical
- *An input at t=0 affects outputs forever*

Multiple recurrent layer RNN

- Recurrent neural network
- All columns are identical
- *An input at t=0 affects outputs forever*

Multiple recurrent layer RNN

- We can also have skips..

A more complex state

- All columns are identical
- *An input at t=0 affects outputs forever*

Or the network may be even more complicated

Y(t)

X(t)

Time

- Shades of NARX

- All columns are identical

- *An input at t=0 affects outputs forever*

Generalization with other recurrences
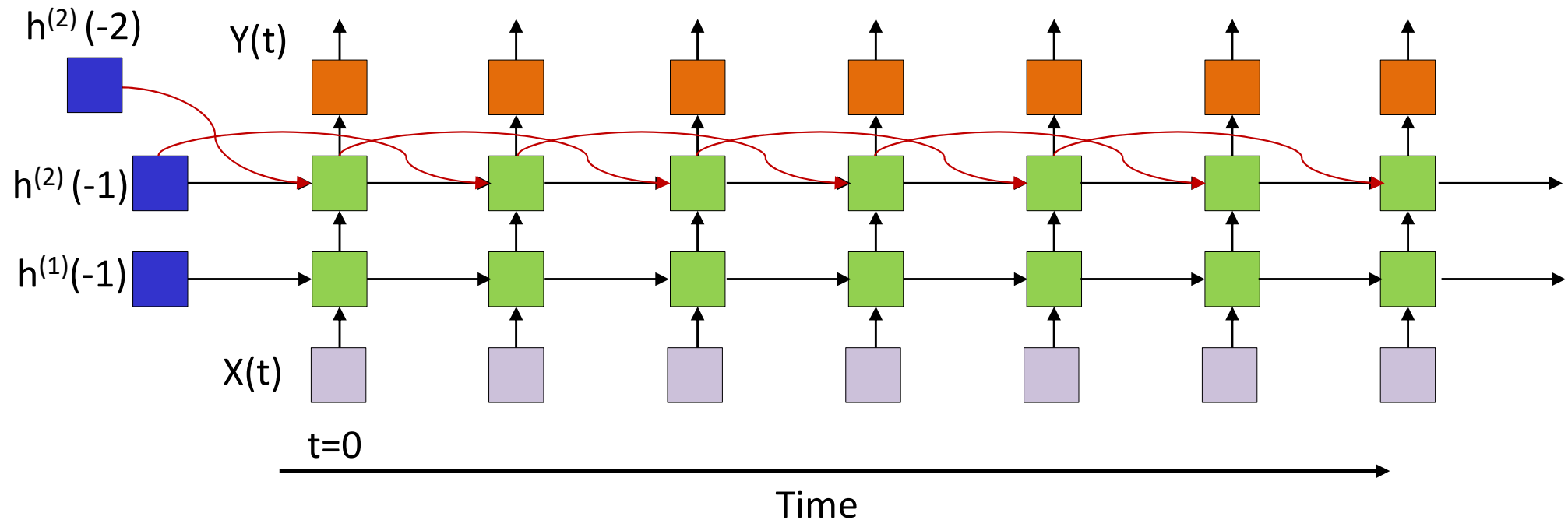
- All columns (including incoming edges) are identical

The simplest structures are most popular
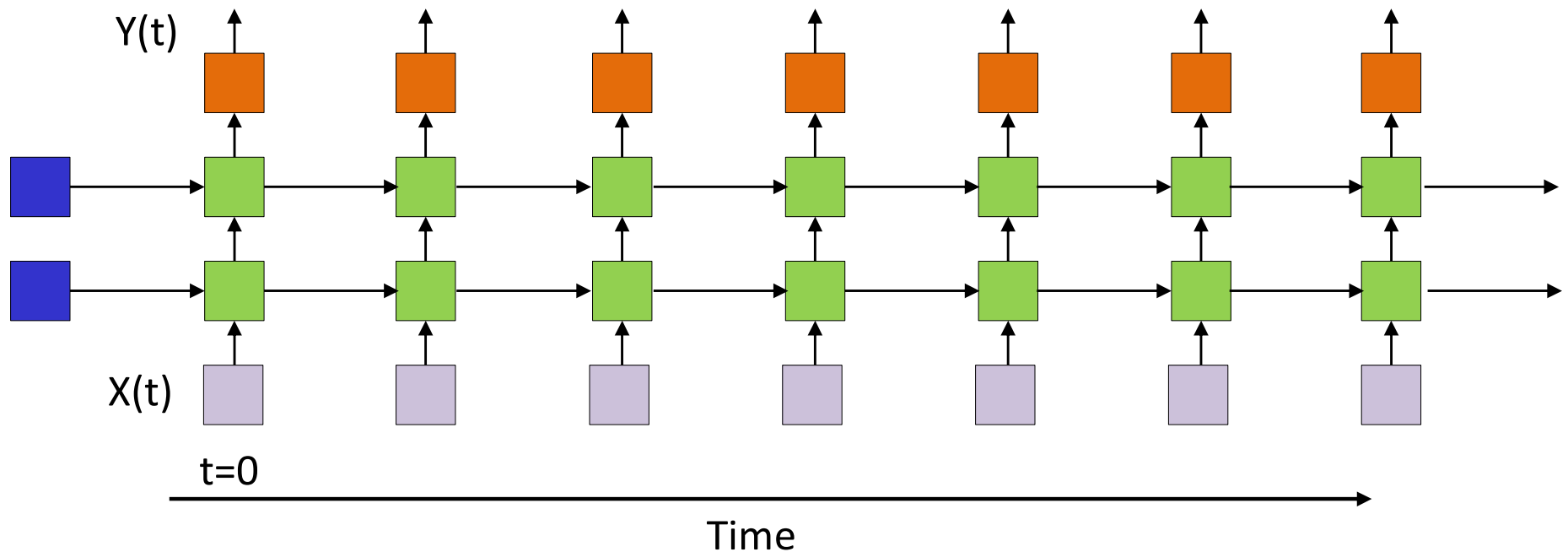
- Recurrent neural network
- All columns are identical
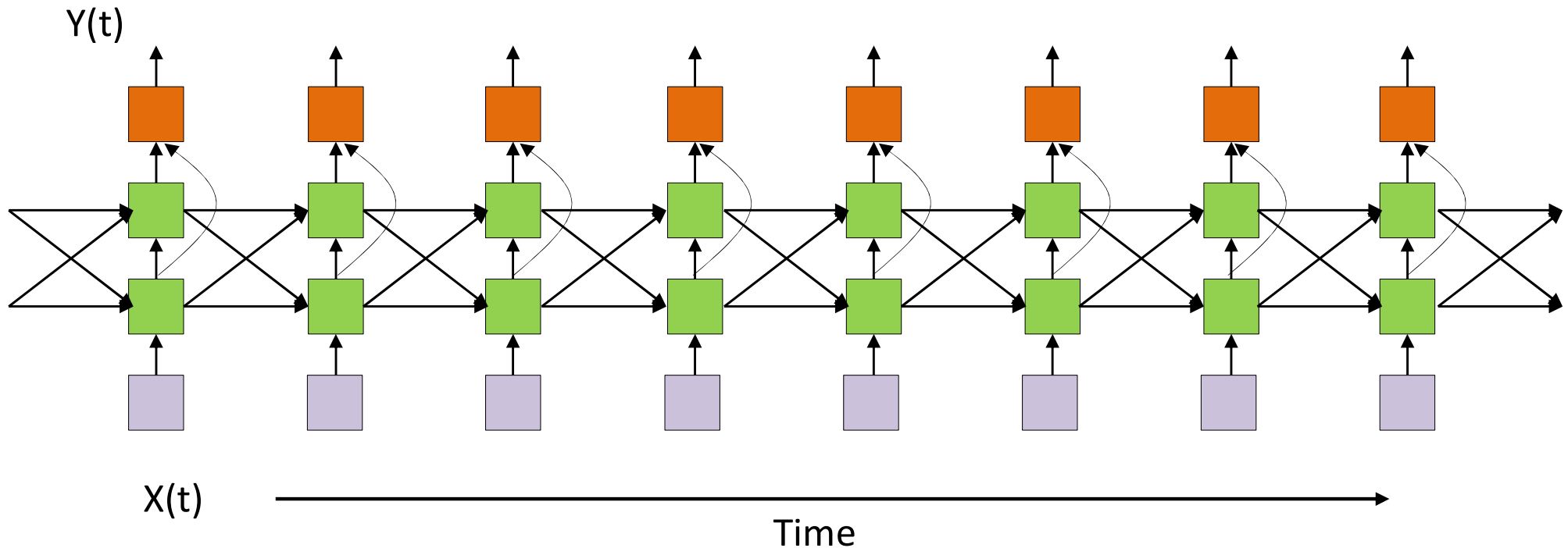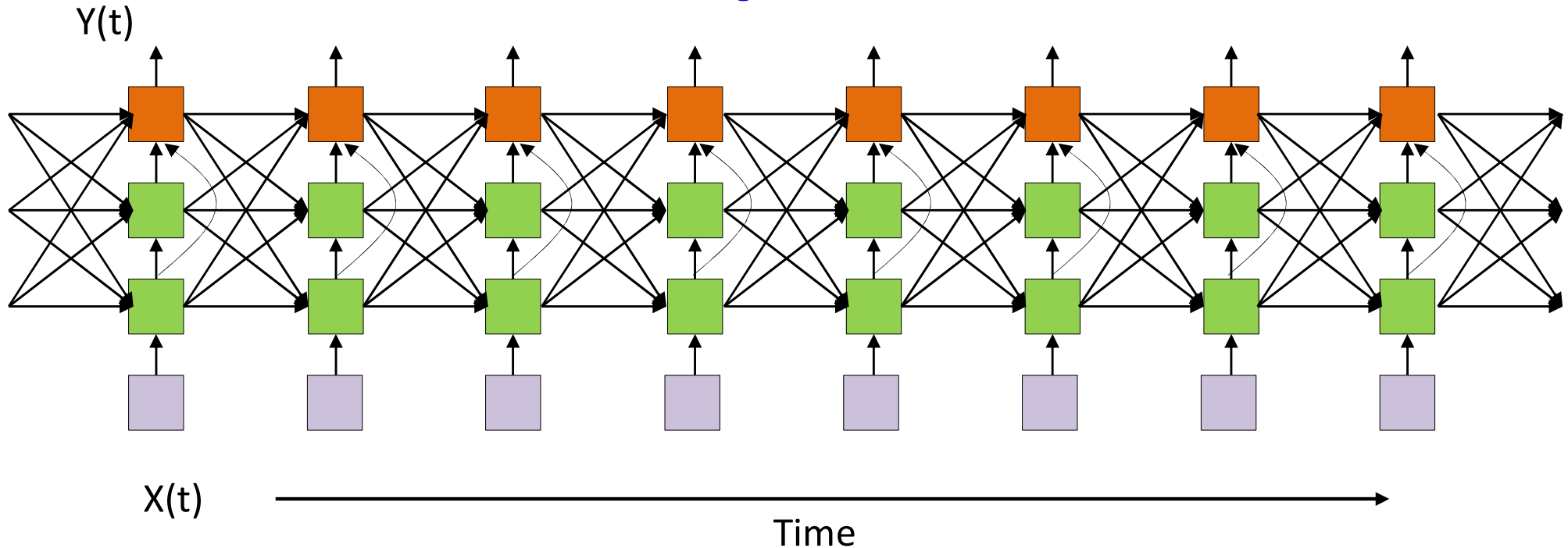- *An input at t=0 affects outputs forever*

A Recurrent Neural Network
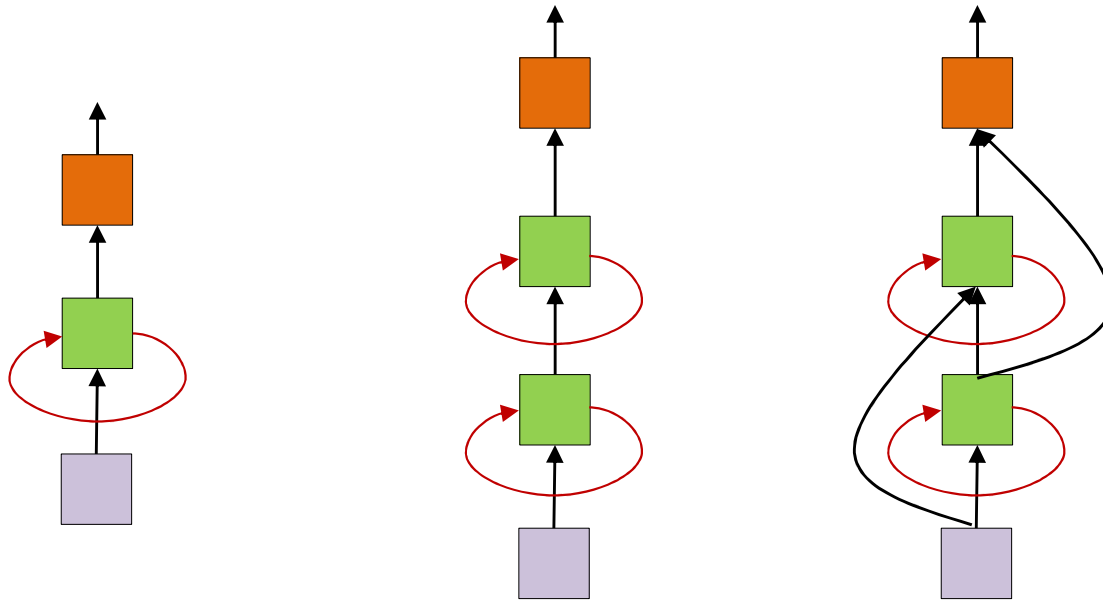
- Simplified models often drawn
- The loops imply recurrence

The detailed version of the simplified representation

Y(t)

h (-1)

X(t)

t=0

Time

Multiple recurrent layer RNN

Y(t)

$h^{(2)}(-1)$

$h^{(1)}(-1)$

X(t)

t=0

Time
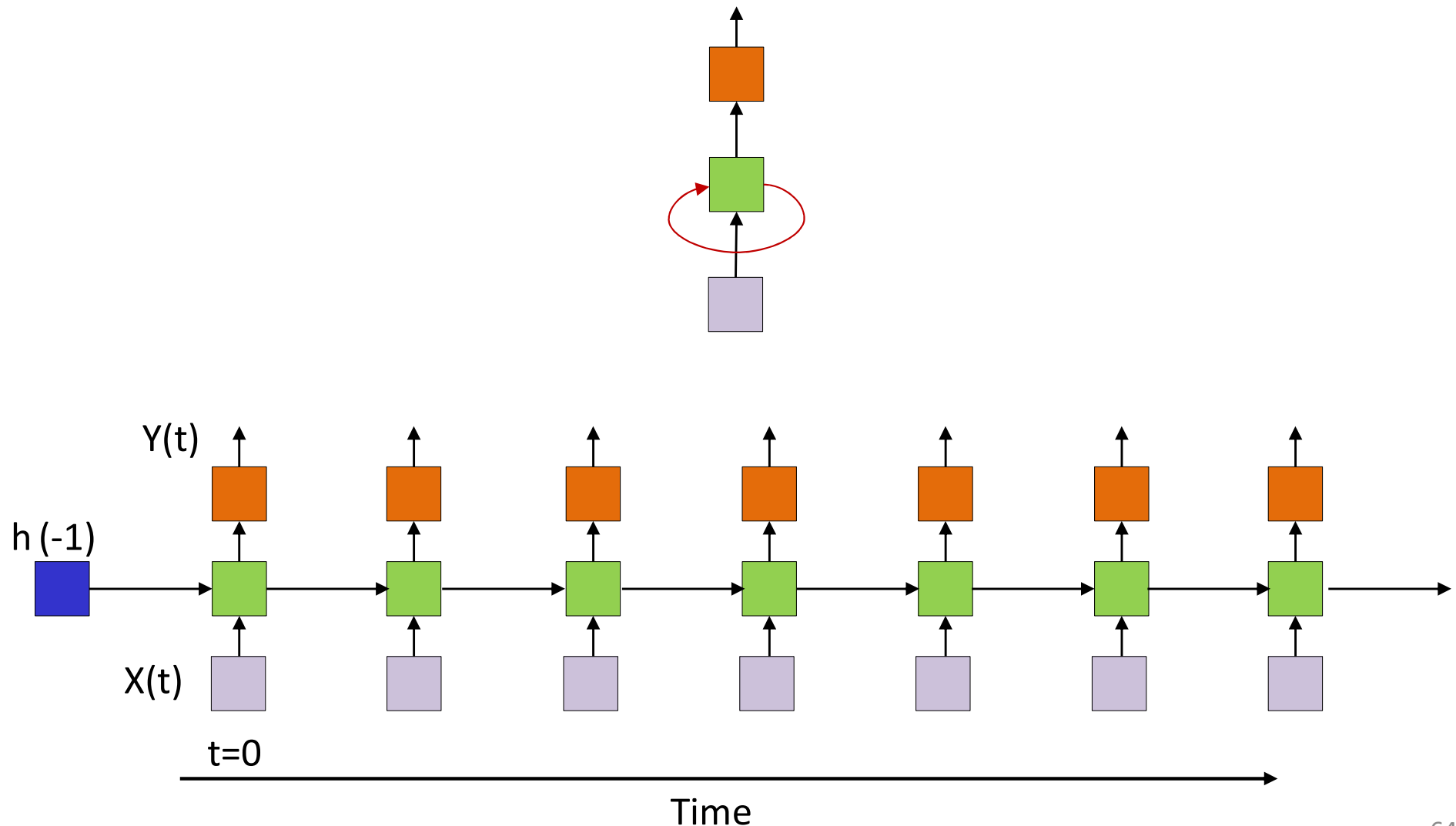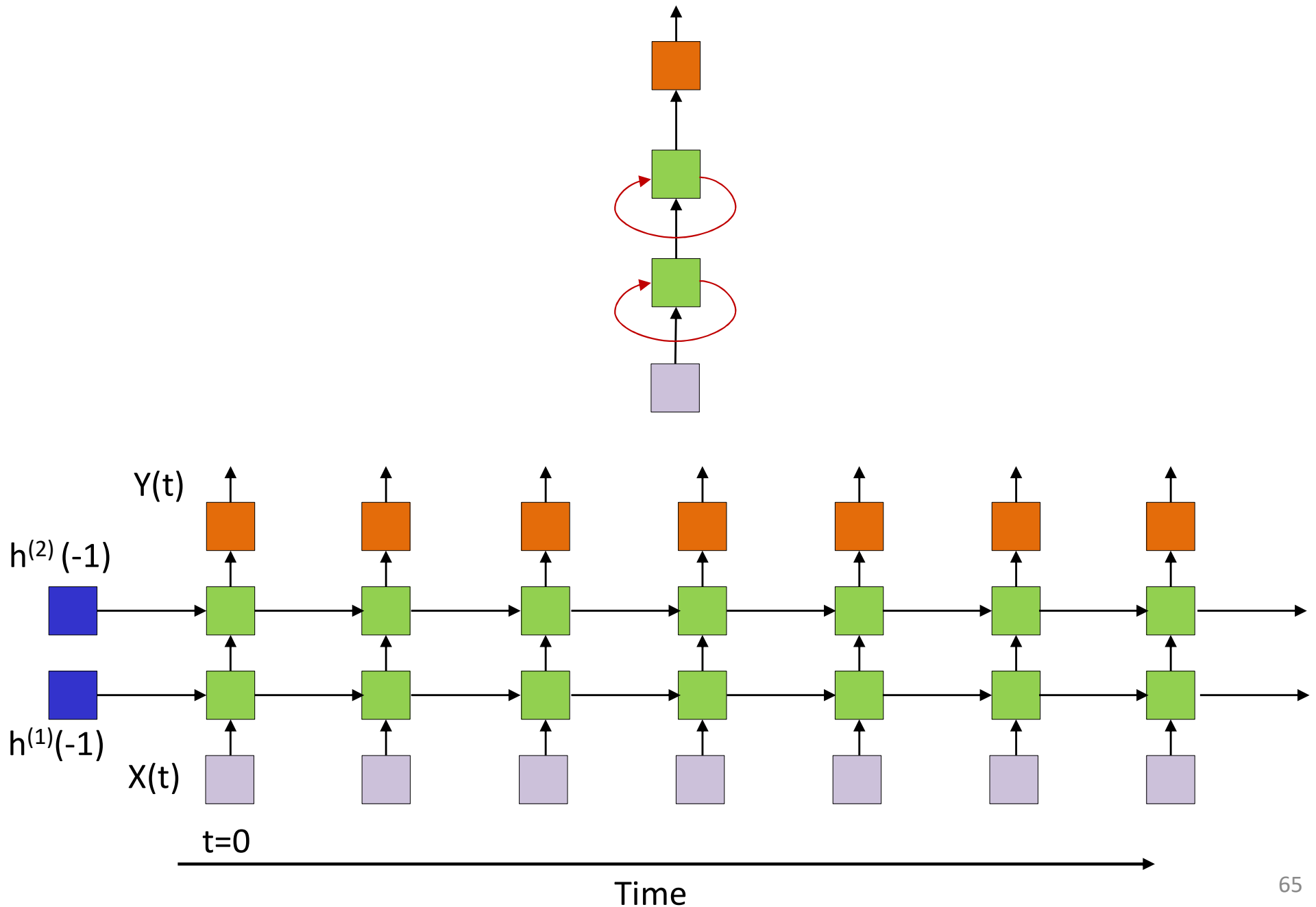
Multiple recurrent layer RNN

Equations

Y

$h^{(1)}$

X

$$h_i^{(1)}(-1) = part\ of\ network\ parameters$$

$$h_i^{(1)}(t) = f_1\left(\sum_j w_{ji}^{(1)} X_j(t) + \sum_j w_{ji}^{(11)} h_j^{(1)}(t-1) + b_i^{(1)}\right)$$

$$Y(t) = f_2\left(\sum_j w_{jk}^{(2)} h_j^{(1)}(t) + b_k^{(2)}, k = 1..M\right)$$
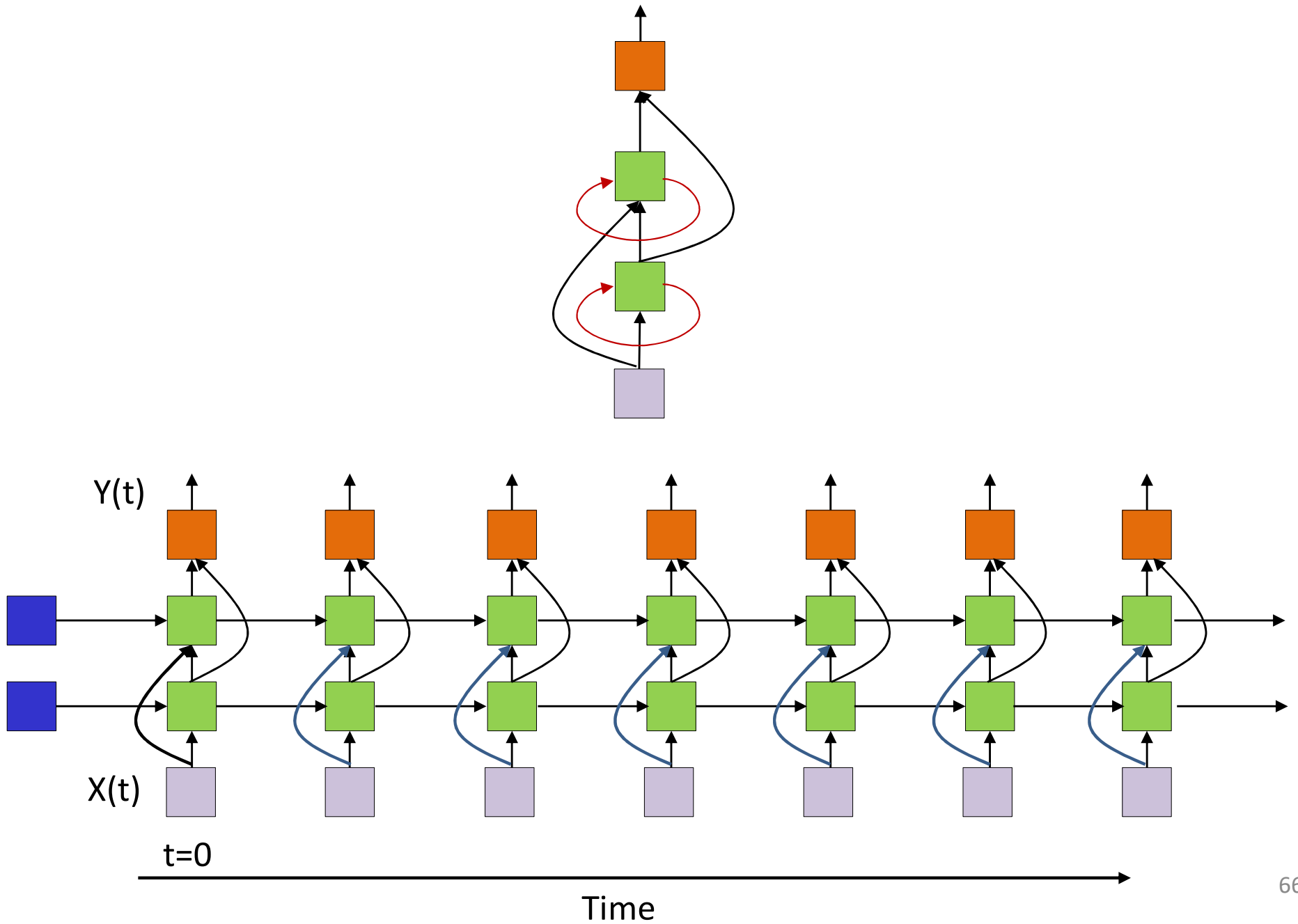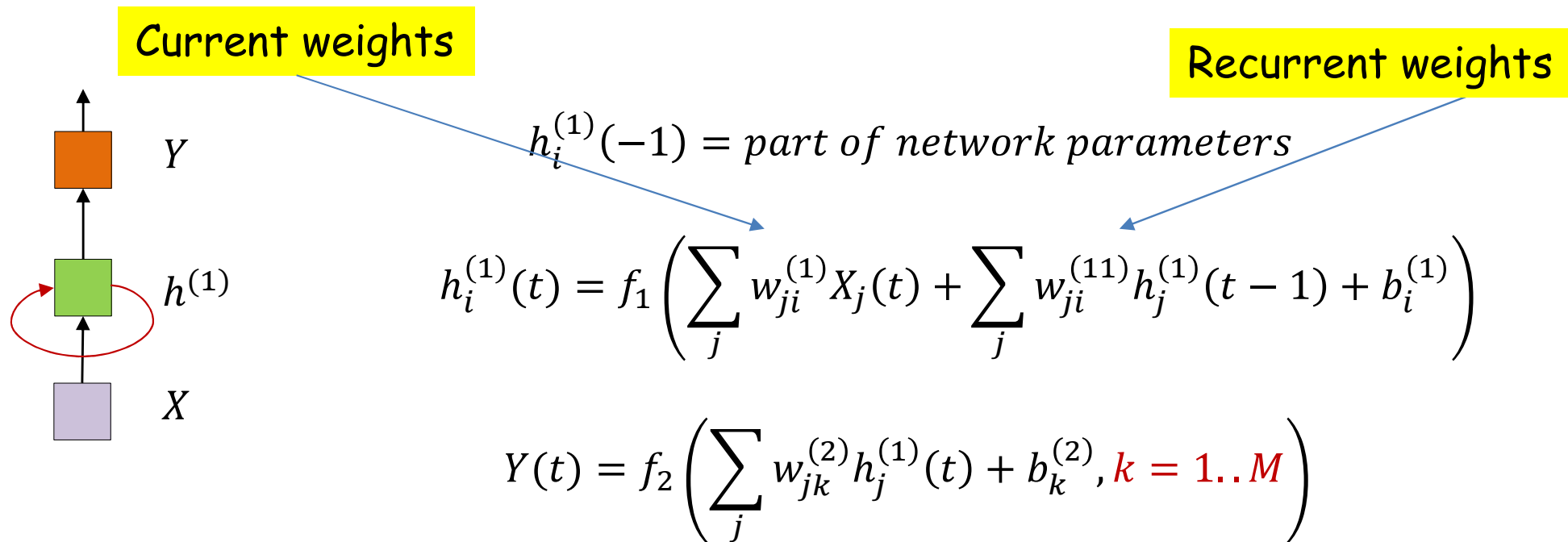
- Note superscript in indexing, which indicates layer of network from which inputs are obtained

- Assuming vector function at output, e.g. softmax

- The *state* node activation, $f_1()$ is typically $\tanh()$

- Every neuron also has a *bias* input
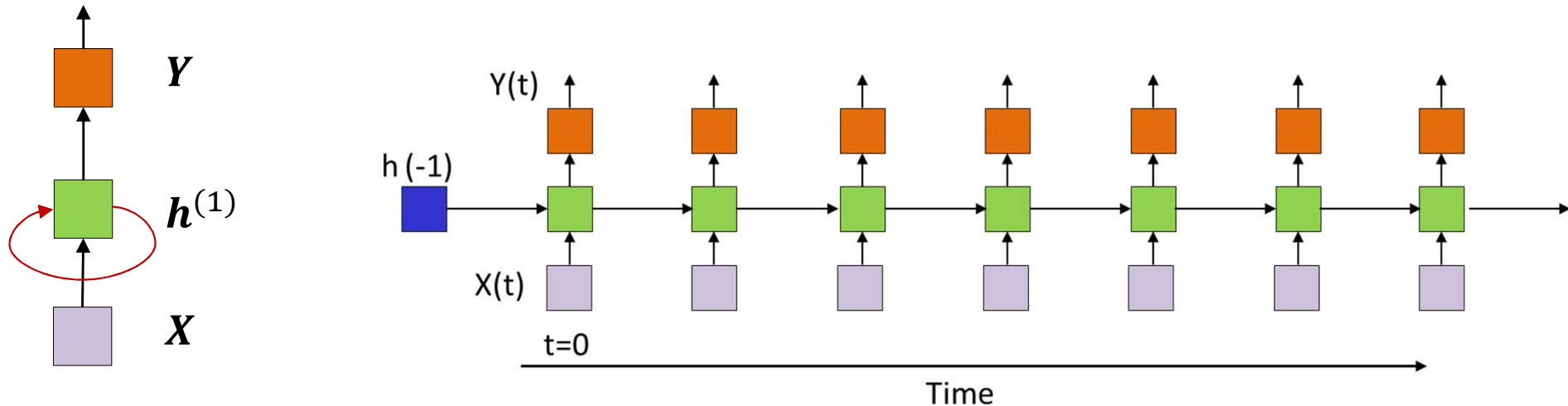
Equations

$$h^{(1)}(-1) = part\ of\ network\ parameters$$

- Computation:

$$h^{(1)}(t) = f_1\big(W^{(1)}X(t) + W^{(11)}h^{(1)}(t-1) + b^{(1)}\big)$$

$$Y(t) = f_2\big(W^{(2)}h^{(1)}(t) + b^{(2)}\big)$$

- The recurrent state activation $f_1()$ is typically tanh()

68
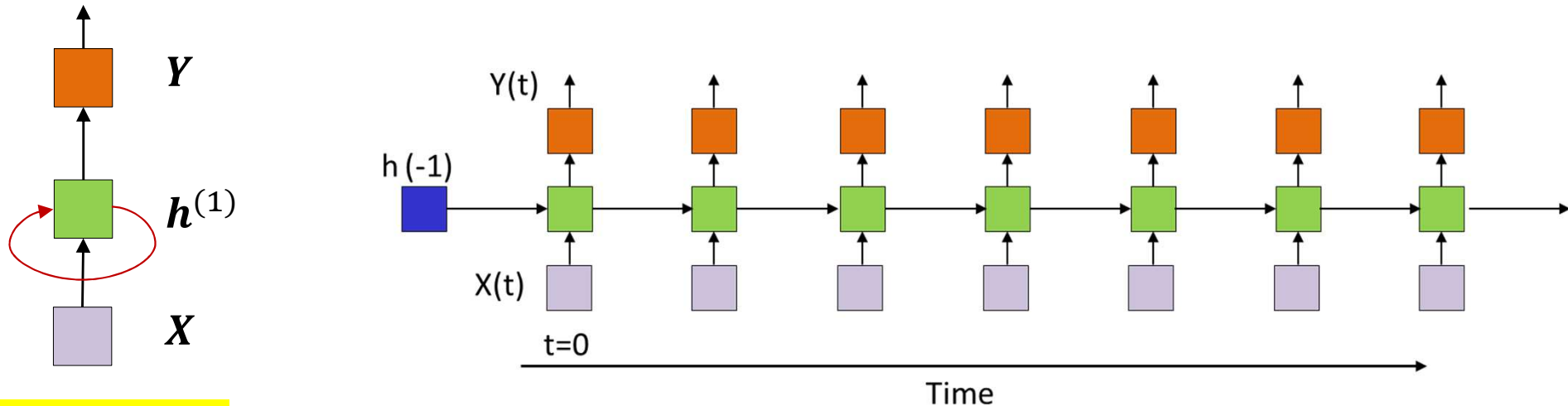
Equations

Current weights $= part\ of\ network\ parameters$

Recurrent weights

- Computation:

$$h^{(1)}(t) = f_1\big(W^{(1)}X(t) + W^{(11)}h^{(1)}(t-1) + b^{(1)}\big)$$

$$Y(t) = f_2\big(W^{(2)}h^{(1)}(t) + b^{(2)}\big)$$

- The recurrent state activation $f_1()$ is typically tanh()

Equations
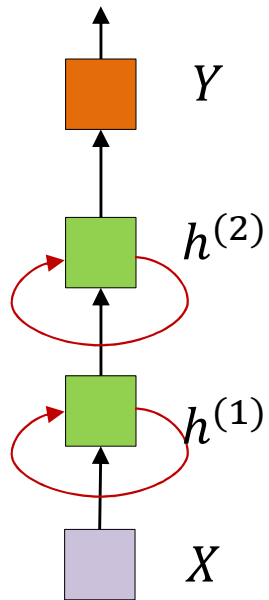
$$h_i^{(1)}(-1) = part\ of\ network\ parameters$$

$$h_i^{(2)}(-1) = part\ of\ network\ parameters$$

$$h_i^{(1)}(t) = f_1\left(\sum_j w_{ji}^{(1)} X_j(t) + \sum_j w_{ji}^{(11)} h_j^{(1)}(t-1) + b_i^{(1)}\right)$$

$$h_i^{(2)}(t) = f_2\left(\sum_j w_{ji}^{(2)} h_j^{(1)}(t) + \sum_j w_{ji}^{(22)} h_j^{(2)}(t-1) + b_i^{(2)}\right)$$

$$Y(t) = f_3\left(\sum_j w_{jk}^{(3)} h_j^{(2)}(t) + b_k^{(3)}, k = 1..M\right)$$

- Assuming vector function at output, e.g. softmax $f_3()$
- The *state* node activations, $f_k()$ are typically $\tanh()$
- Every neuron also has a *bias* input
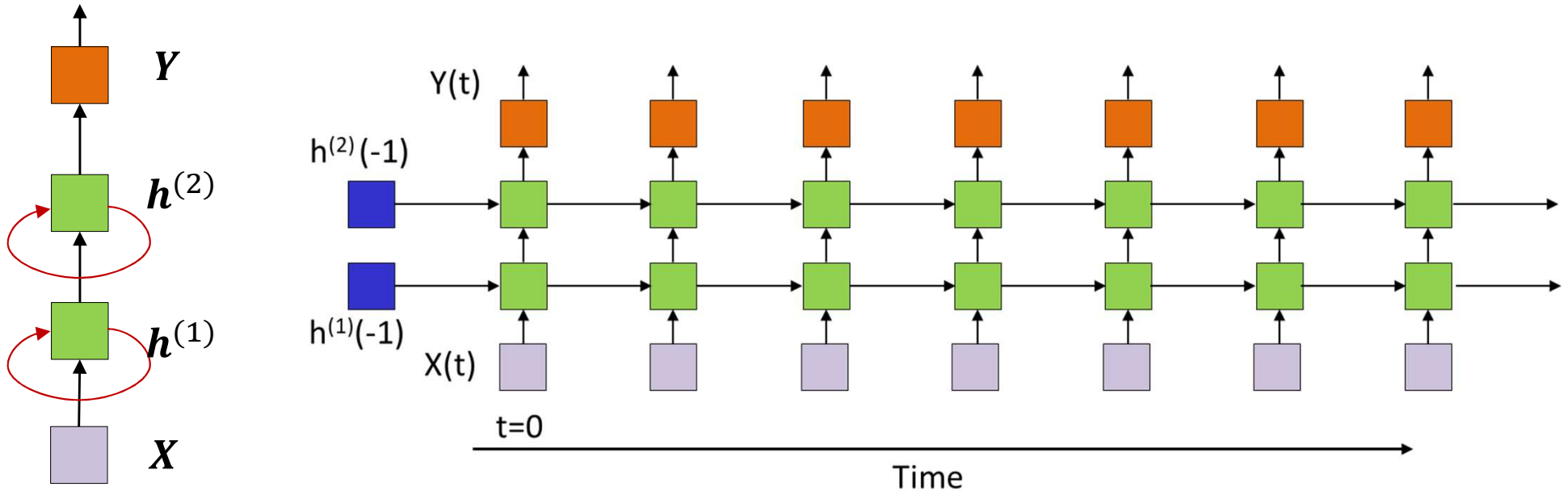
Equations

$$\boldsymbol{h}^{(1)}(-1) \text{ and } \boldsymbol{h}^{(2)}(-1) = part \ of \ network \ parameters$$

- Computation:

$$\boldsymbol{h}^{(1)}(t) = f_1\big(\boldsymbol{W}^{(1)}\boldsymbol{X}(t) + \boldsymbol{W}^{(11)}\boldsymbol{h}^{(1)}(t-1) + \boldsymbol{b}^{(1)}\big)$$

$$\boldsymbol{h}^{(2)}(t) = f_2\big(\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)}(t) + \boldsymbol{W}^{(22)}\boldsymbol{h}^{(2)}(t-1) + \boldsymbol{b}^{(2)}\big)$$

$$\boldsymbol{Y}(t) = f_3\big(\boldsymbol{W}^{(3)}\boldsymbol{h}^{(2)}(t) + \boldsymbol{b}^{(3)}\big)$$

- The recurrent state activation is typically tanh()

Equations

$$\boldsymbol{h}^{(1)}(-1) = part\ of\ network\ parameters$$

$$\boldsymbol{h}^{(2)}(-1) = part\ of\ network\ parameters$$
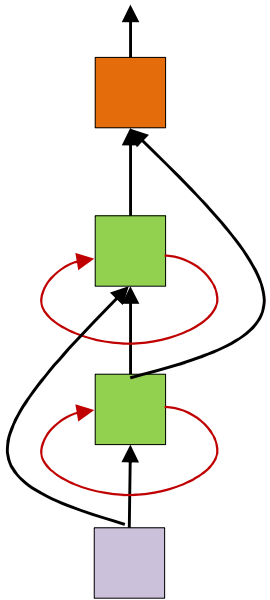
$$\boldsymbol{h}^{(1)}(t) = f_1\big(\boldsymbol{W}^{(0,1)}\boldsymbol{X}(t) + \boldsymbol{W}^{(11)}\boldsymbol{h}^{(1)}(t-1) + \boldsymbol{b}^{(1)}\big)$$

$$\boldsymbol{h}^{(2)}(t) = f_2\big(\boldsymbol{W}^{(12)}\boldsymbol{h}^{(1)}(t) + \boldsymbol{W}^{(0,2)}\boldsymbol{X}(t) + \boldsymbol{W}^{(22)}\boldsymbol{h}^{(2)}(t-1) + \boldsymbol{b}^{(2)}\big)$$

$$\boldsymbol{Y}(t) = f_3\big(\boldsymbol{W}^{(3)}\boldsymbol{h}^{(2)}(t) + \boldsymbol{W}^{(13)}\boldsymbol{h}^{(1)}(t) + \boldsymbol{b}^{(3)}\big)$$

Variants on recurrent nets
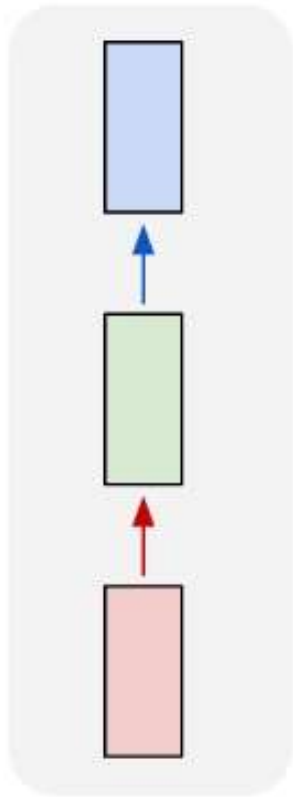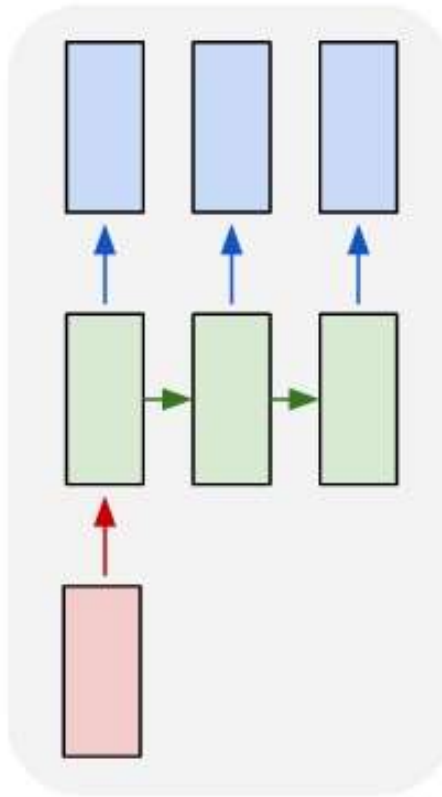
one to one   one to many   many to one
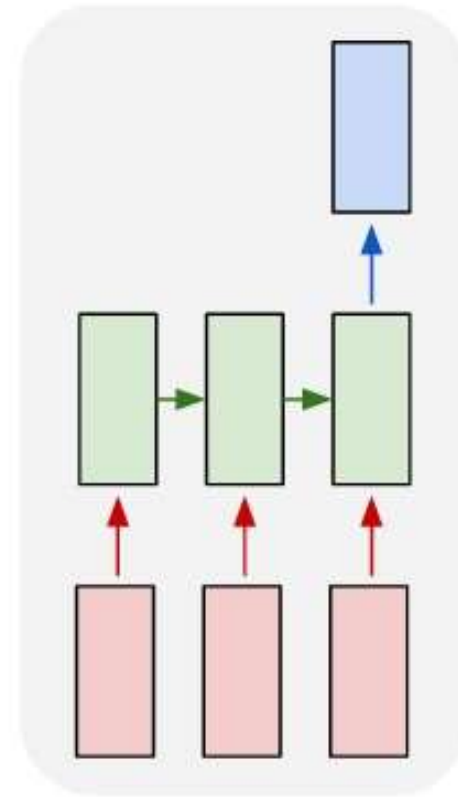
Images from Karpathy

- 1: Conventional MLP
- 2: Sequence *generation*, e.g. image to caption
- 3: Sequence based *prediction or classification,* e.g. Speech recognition, text classification

Variants

Images from Karpathy
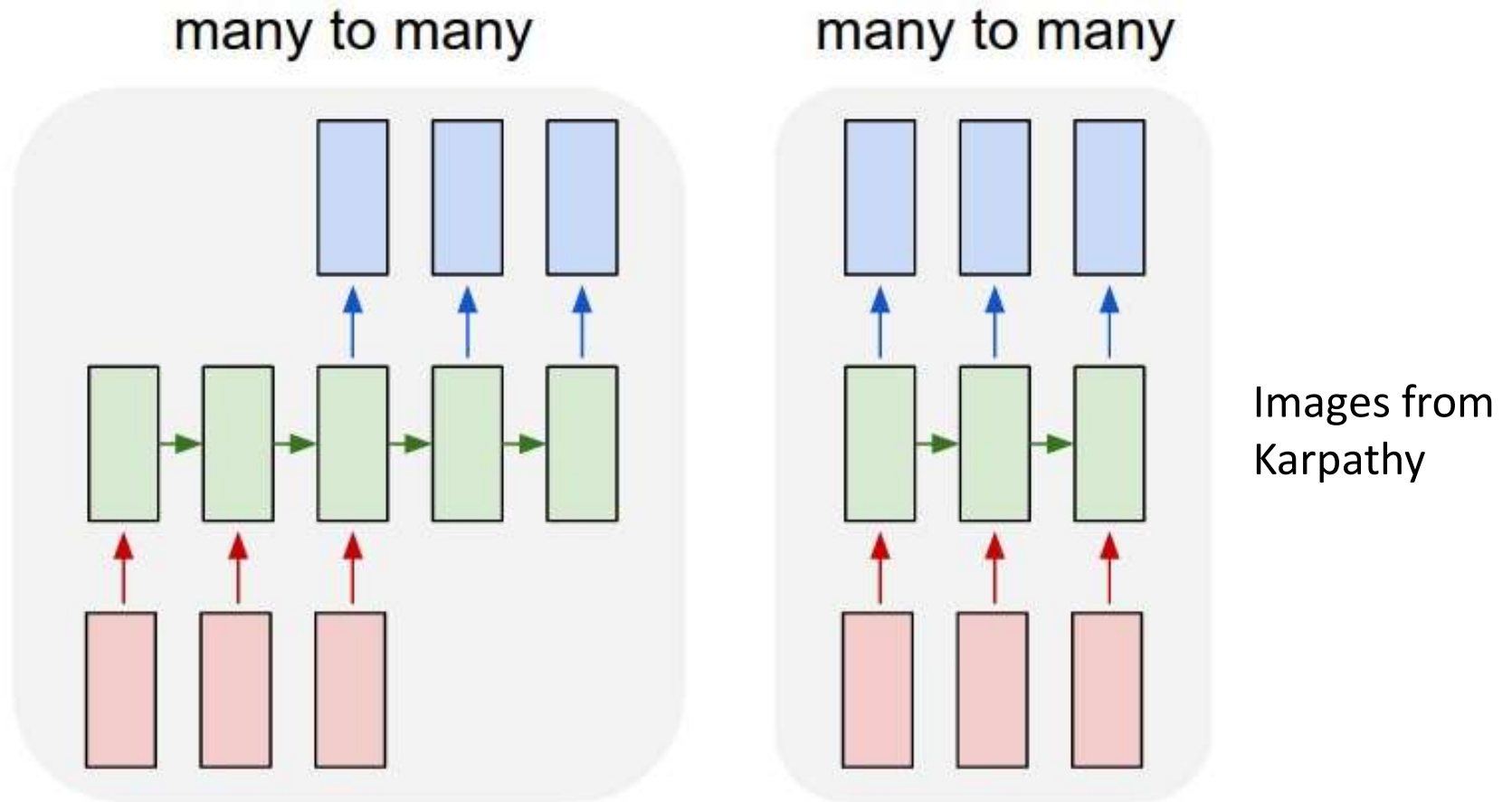
- 1: *Delayed* sequence to sequence, e.g. machine translation
- 2: Sequence to sequence, e.g. stock problem, label prediction
- Etc…

Story so far

- Time series analysis must consider past inputs along with current input
- Looking into the infinite past requires recursion

- NARX networks achieve this by feeding back the output to the input

- "Simple" recurrent networks maintain separate "memory" or "context" units to retain some information about the past
    - But during learning the current error does not influence the past

- State-space models retain information about the past through recurrent hidden states
    - These are "fully recurrent" networks
    - The initial values of the hidden states are generally learnable parameters as well

- State-space models enable current error to update parameters in the past

How do we *train* the network
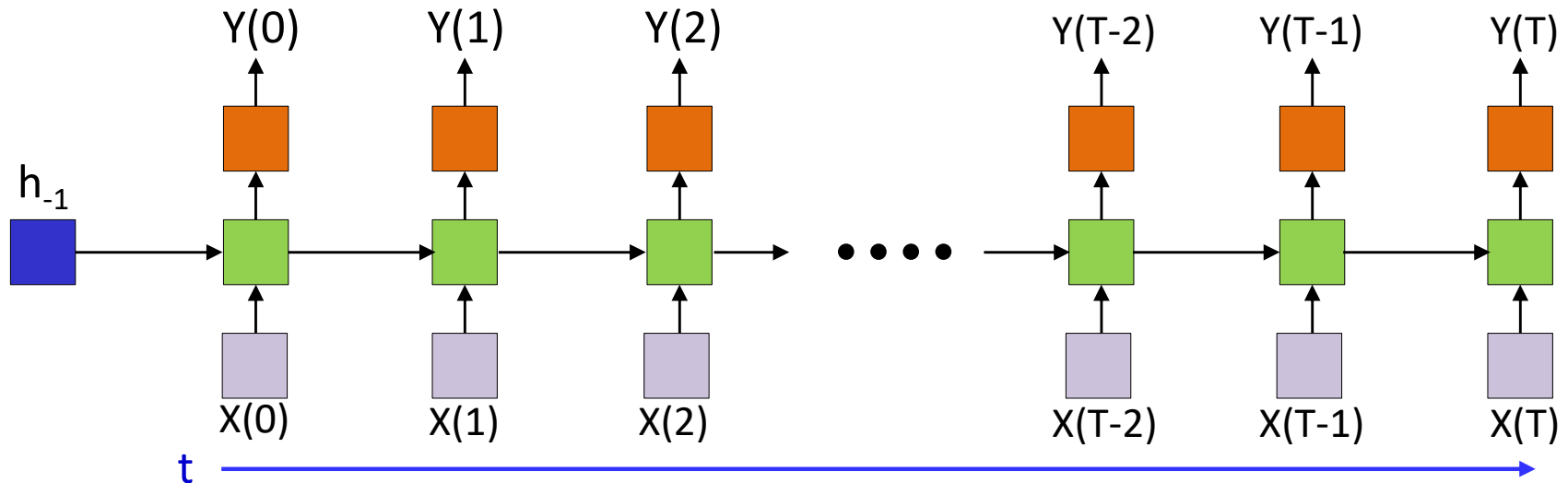
- Back propagation through time (BPTT)

- Given a collection of *sequence* inputs
  - $(\mathbf{X}_i, \mathbf{D}_i)$, where
  - $\mathbf{X}_i = X_{i,0}, \dots, X_{i,T}$
  - $\mathbf{D}_i = D_{i,0}, \dots, D_{i,T}$
- Train network parameters to minimize the error between the output of the network $\mathbf{Y}_i = Y_{i,0}, \dots, Y_{i,T}$ and the desired outputs
  - This is the most generic setting. In other settings we just "remove" some of the input or output entries

Training the RNN
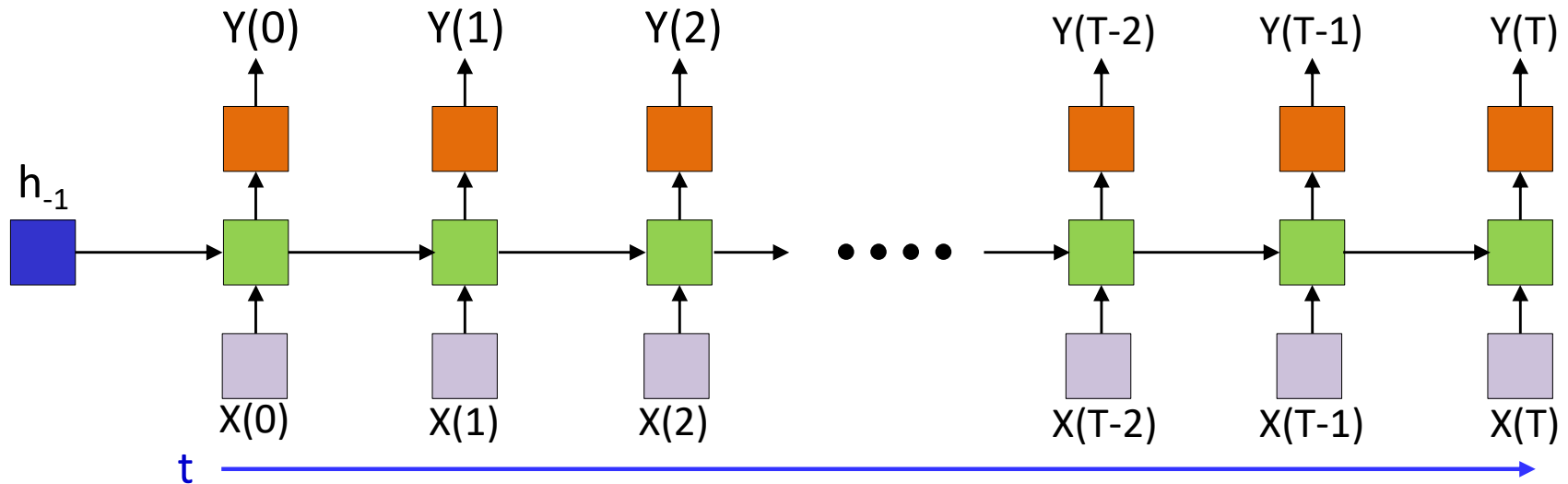
- The "unrolled" computation is just a giant shared-parameter neural network
  - All columns are identical and share parameters

- Network parameters can be trained via gradient-descent (or its variants) using shared-parameter gradient descent rules
  - Gradient computation requires a forward pass, back propagation, and pooling of gradients (for parameter sharing)

Training: Forward pass
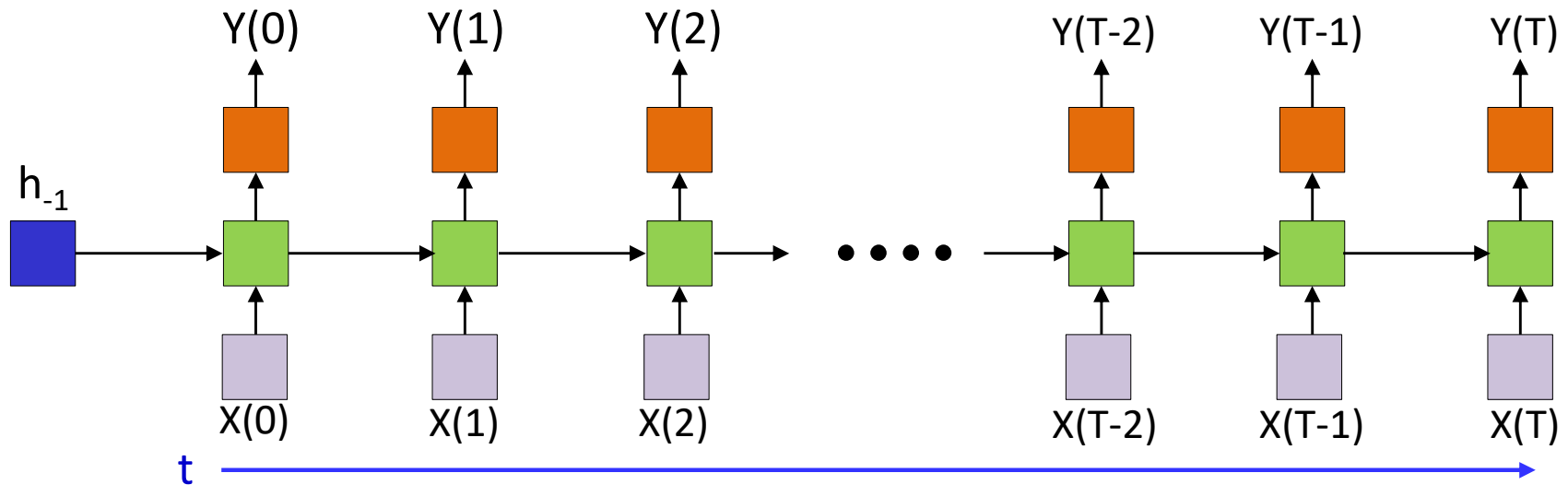
- For each training input:
- Forward pass: pass the entire data sequence through the network, generate outputs

Recurrent Neural Net
Assuming time-synchronous output

```
# Assuming h(-1,*) is known
# Assuming L hidden-state layers and an output layer
# W_c(*) and W_r(*) are matrics, b(*) are vectors
# W_c are weights for inputs from current time
# W_r is recurrent weight applied to the previous time
# W_o are output layre weights


for t = 0:T-1  # Including both ends of the index
    h(t,0) = x(t) # Vectors. Initialize h(0) to input
    for l = 1:L  # hidden layers operate at time t
        z(t,l) = W_c(l)h(t,l-1) + W_r(l)h(t-1,l) + b(l)
        h(t,l) = tanh(z(t,l)) # Assuming tanh activ.
    z_o(t) = W_o h(t,L) + b_o
    Y(t) = softmax( z_o(t) )
```

Subscript "c" – current
Subscript "r" – recurrent

Training: Computing gradients
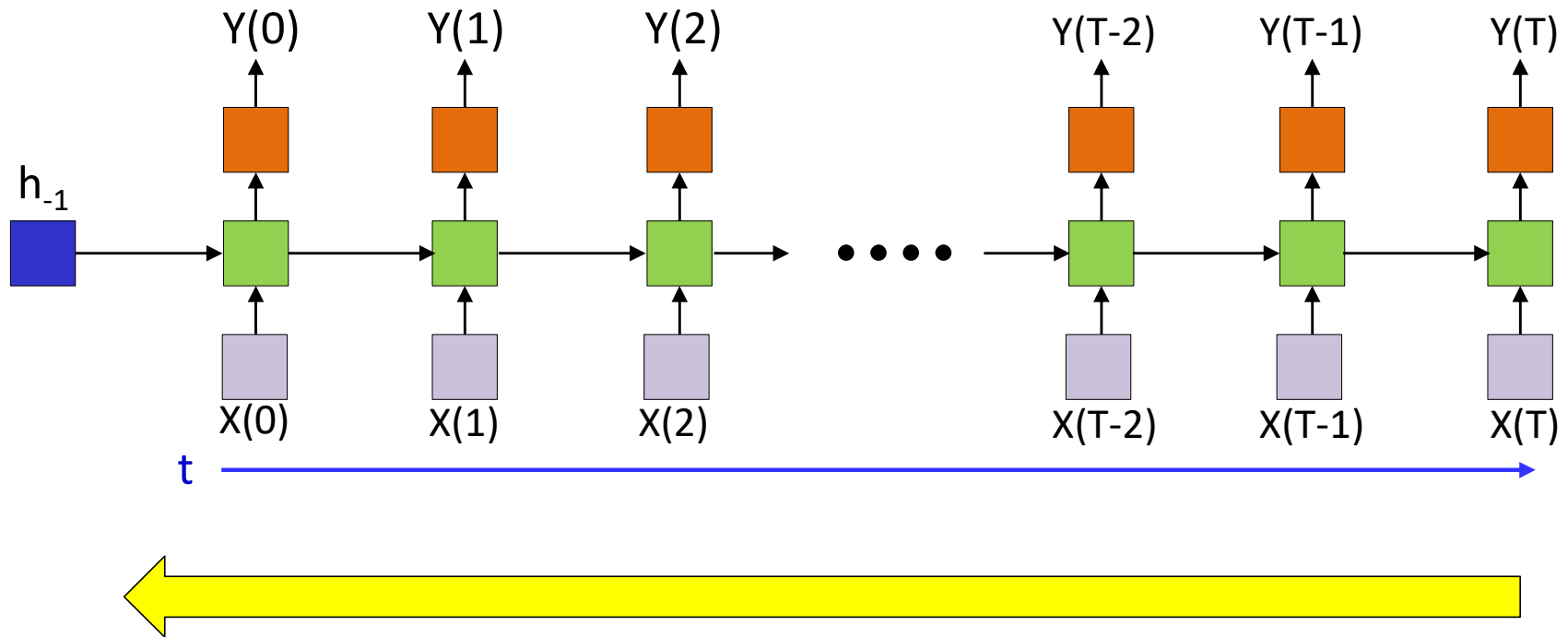
- For each training input:
- Backward pass: Compute gradients via backpropagation
  - *Back Propagation Through Time*

Back Propagation Through Time
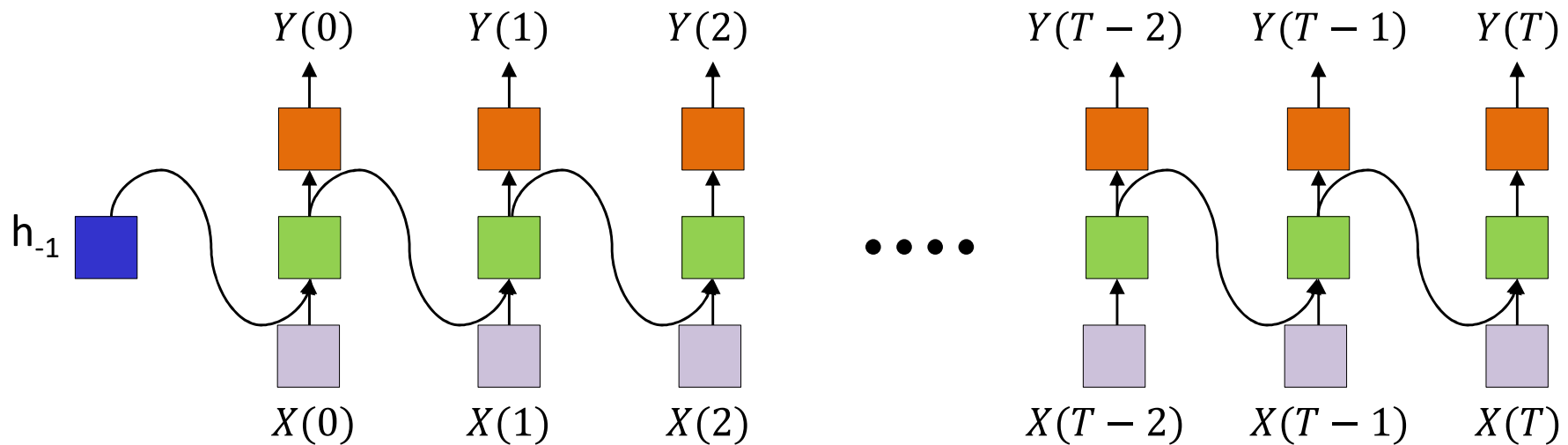
Will only focus on *one* training instance

All subscripts represent *components* and not training instance index

Back Propagation Through Time
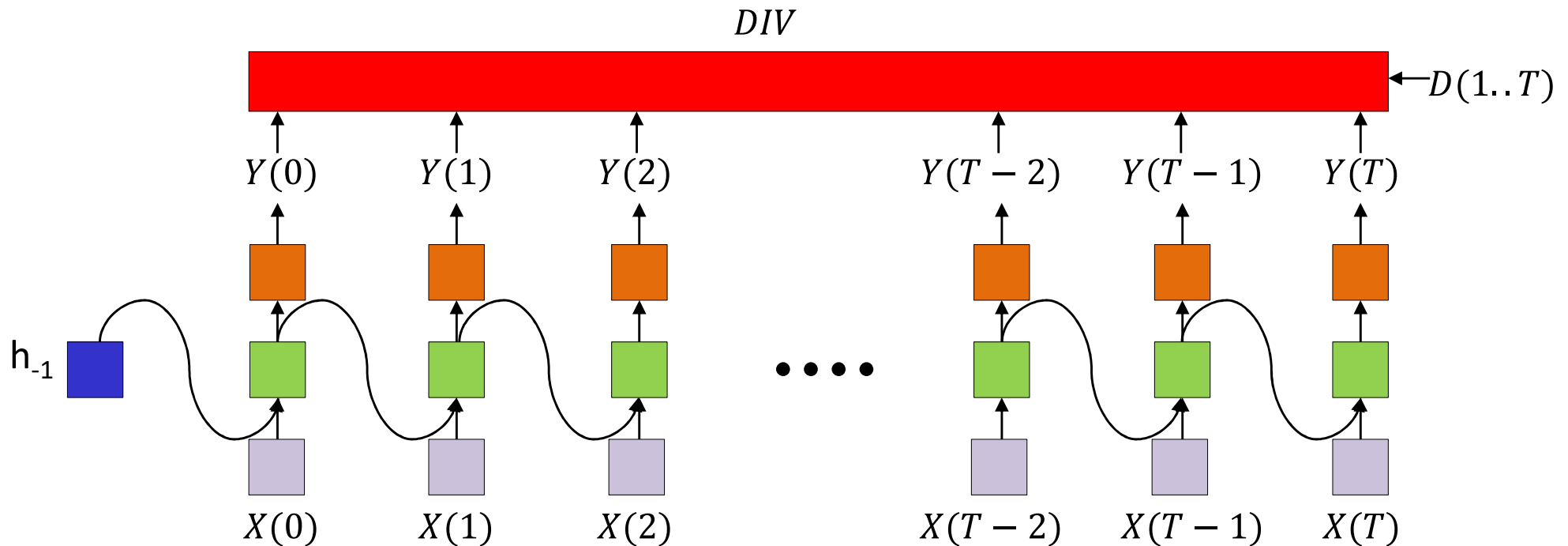
- The divergence computed is between the *sequence of outputs* by the network and the *desired sequence of outputs*
  - DIV is a scalar function of a series of vectors!

- This is *not* just the sum of the divergences at individual times
  - Unless we explicitly define it that way

Notation
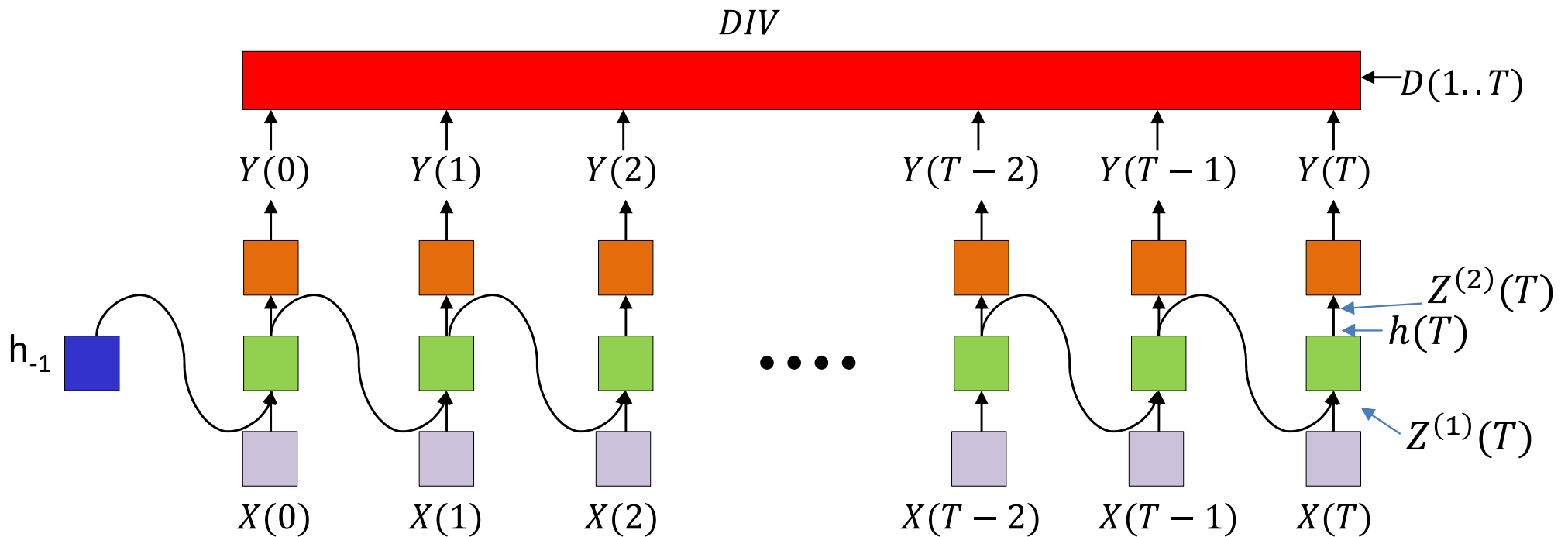
- $Y(t)$ is the output at time t
  - $Y_i(t)$ is the ith output
- $Z^{(2)}(t)$ is the pre-activation value of the neurons at the output layer at time t
- $h(t)$ is the output of the hidden layer at time t
  - Assuming only one hidden layer in this example
- $Z^{(1)}(t)$ is the pre-activation value of the hidden layer at time t

Notation
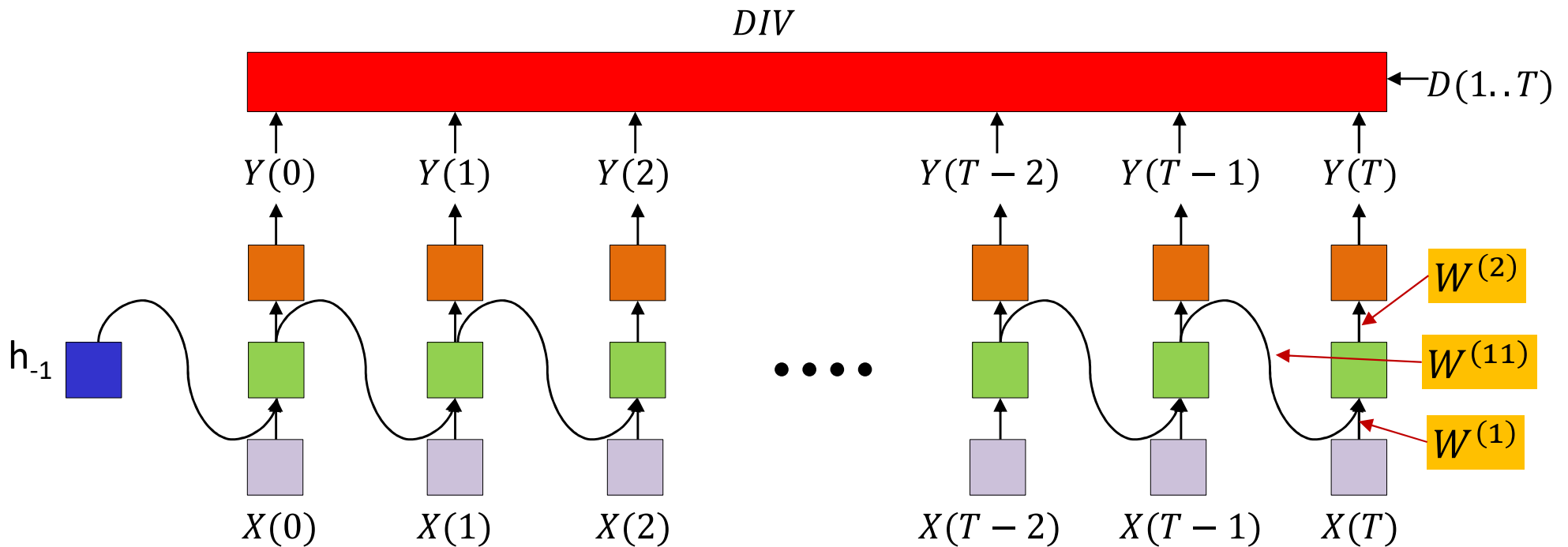
- $W^{(1)} = \left[w_{ij}^{(1)} \right]$ is the matrix of *current* weights from the input to the hidden layer.

- $W^{(2)} = \left[w_{ij}^{(2)} \right]$ is the matrix of *current* weights from the hidden layer to the output layer

- $W^{(11)} = \left[w_{ij}^{(11)} \right]$ is the matrix of *recurrent* weights from the hidden layer to itself

Back Propagation Through Time

DIV

$Y(0)$ $Y(1)$ $Y(2)$ $Y(T-2)$ $Y(T-1)$ $Y(T)$

$D(1..T)$

h_{-1}

$X(0)$ $X(1)$ $X(2)$ $X(T-2)$ $X(T-1)$ $X(T)$
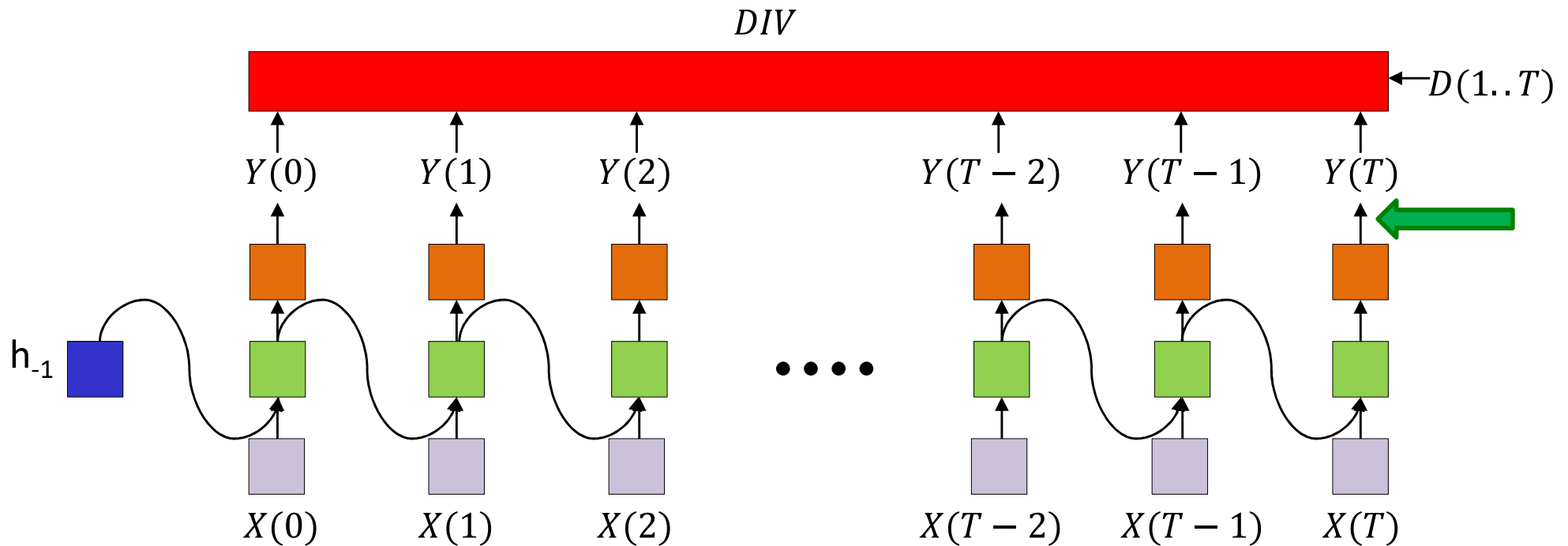
First step of backprop: Compute $\nabla_{Y(T)} DIV$ (Compute $\frac{dDIV}{dY_i(T)}$ for all i)

Note: DIV is a function of *all* outputs Y(0) … Y(T)

In general we will be required to compute $\frac{dDIV}{dY_i(t)}$ for all i and t as we will see. This can be a source of significant difficulty in many scenarios.

85

DIV

$Div(0)$ $Div(1)$ $Div(2)$ $Div(T-2)$ $Div(T-1)$ $Div(T)$

$\leftarrow D(T)$

$Y(0)$ $Y(1)$ $Y(2)$ $Y(T-2)$ $Y(T-1)$ $Y(T)$

h$_{-1}$

• • • •

$X(0)$ $X(1)$ $X(2)$ $X(T-2)$ $X(T-1)$ $X(T)$
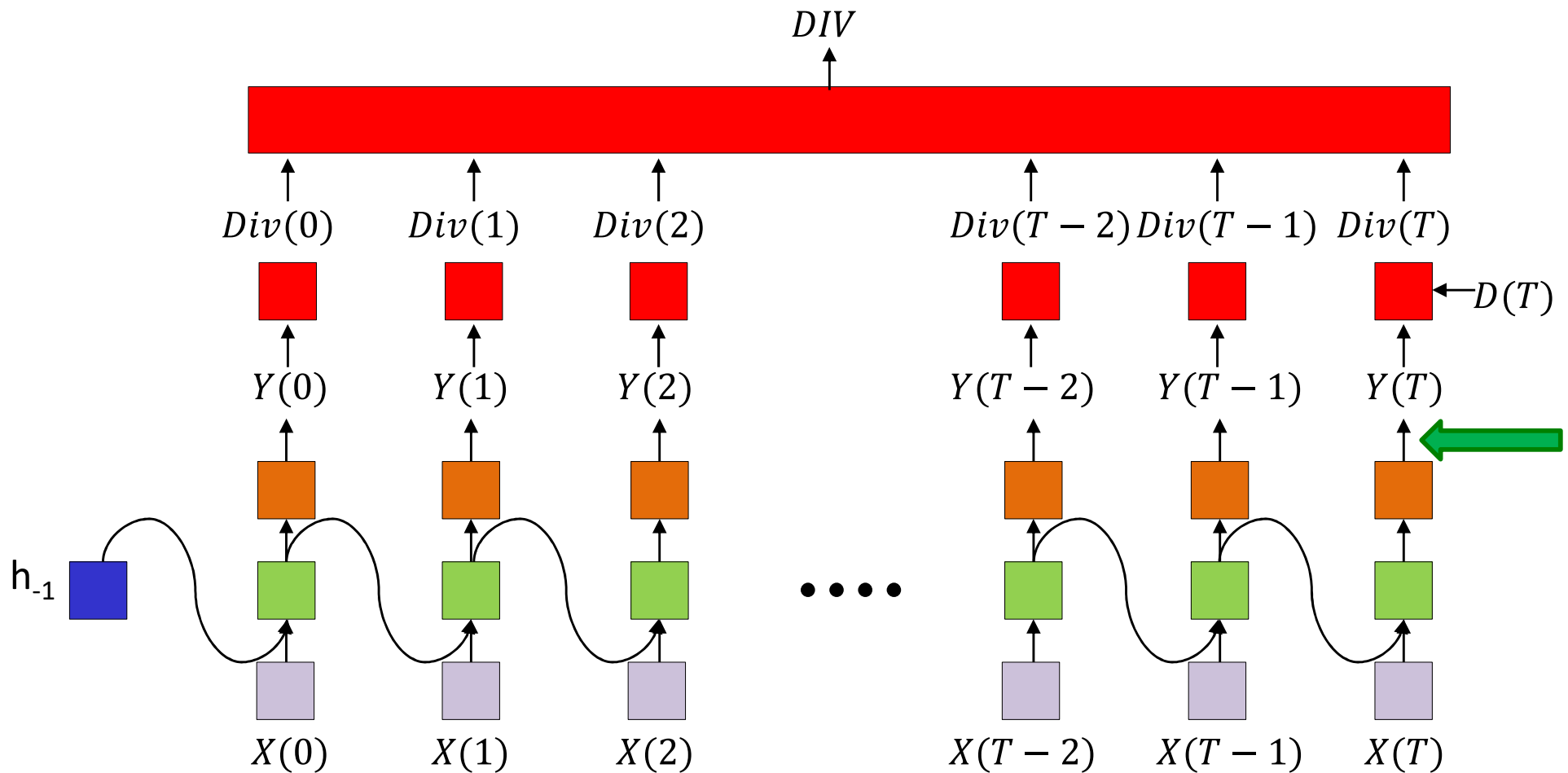
Special case, when the overall divergence is a simple sum of local divergences at each time: $DIV = \sum_t Div(t)$

Must compute $\nabla_{Y(t)} DIV$

$$\frac{\partial DIV}{\partial Y_i(t)} \text{ for all i for all T}$$

Will get $\nabla_{Y(t)} Div(t)$

$$\frac{\partial DIV}{\partial Y_i(t)} = \frac{\partial Div(t)}{\partial Y_i(t)}$$

86

Back Propagation Through Time
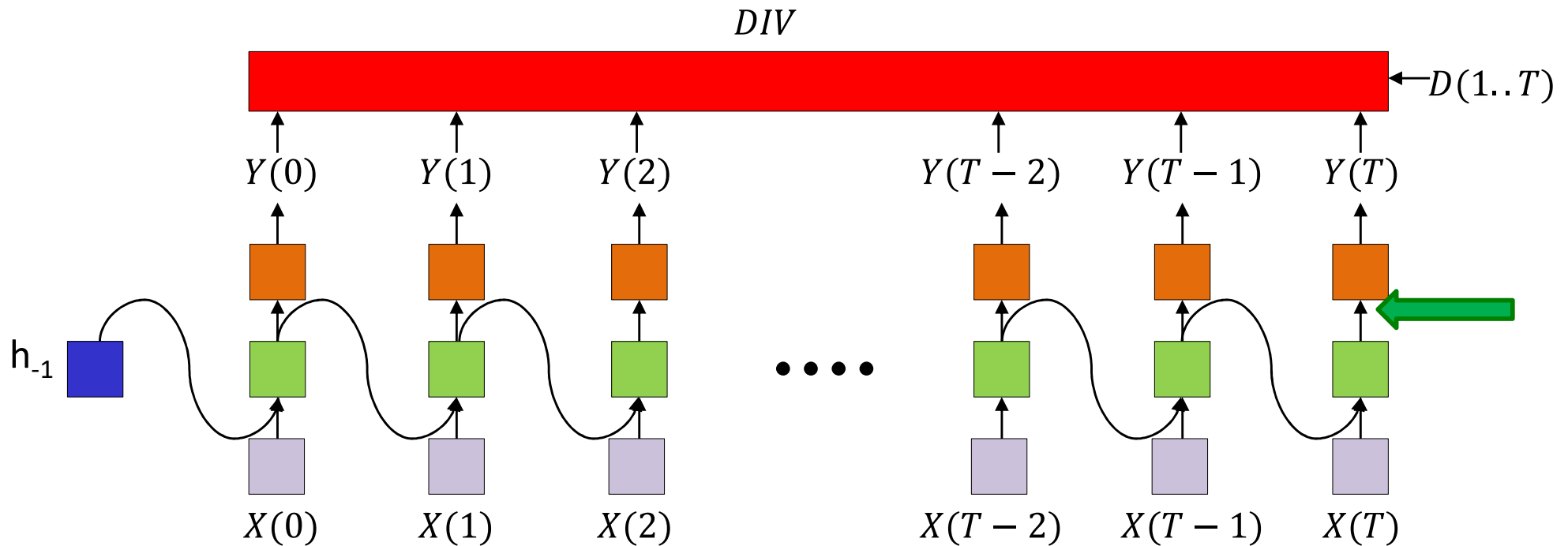
First step of backprop: Compute $\frac{dDIV}{dY_i(T)}$ for all i

$$\nabla_{Z^{(2)}(T)}DIV = \nabla_{Y(T)}DIV\nabla_{Z^{(2)}(T)}Y(T)$$

Vector output activation

$$\frac{dDIV}{dZ_i^{(2)}(T)} = \frac{dDIV}{dY_i(T)}\frac{dY_i(T)}{dZ_i^{(2)}(T)}$$ OR $$\frac{dDIV}{dZ_i^{(2)}(T)} = \sum_j \frac{dDIV}{dY_j(T)}\frac{dY_j(T)}{dZ_i^{(2)}(T)}$$

Back Propagation Through Time

DIV
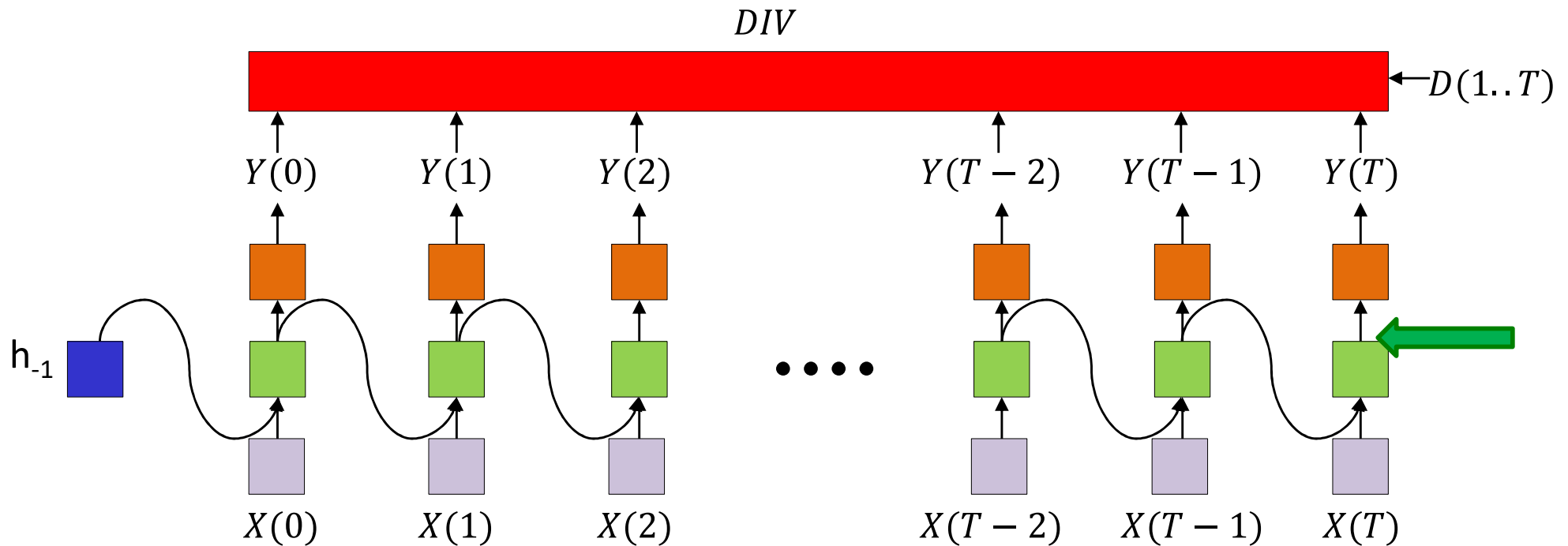
$\dfrac{dDIV}{dY_i(T)}$ for all i

$\dfrac{dDIV}{dZ_i^{(2)}(T)} = \dfrac{dDiv(T)}{dY_i(T)} \dfrac{dY_i(T)}{dZ_i^{(2)}(T)}$

$$\frac{dDIV}{dh_i(T)} = \sum_j \frac{dDIV}{dZ_j^{(2)}(T)} \frac{dZ_j^{(2)}(T)}{dh_i(T)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dZ_j^{(2)}(T)}$$

$$\nabla_{h(T)} DIV = \nabla_{Z^{(2)}(T)} DIV \, W^{(2)}$$

Back Propagation Through Time

$$\frac{dDIV}{dZ_i^{(2)}(T)} = \frac{dDiv(T)}{dY_i(T)} \frac{dY_i(T)}{dZ_i^{(2)}(T)} \qquad \frac{dDIV}{dh_i(T)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dZ_j^{(2)}(T)}$$

$$\frac{dDIV}{dw_{ij}^{(2)}} = \frac{dDIV}{dZ_j^{(2)}(T)} h_i(T)$$

$$\nabla_{W^{(2)}} DIV = h(T) \nabla_{Z^{(2)}(T)} DIV$$

Back Propagation Through Time

DIV

$D(1..T)$

$Y(0)$ $Y(1)$ $Y(2)$ $Y(T-2)$ $Y(T-1)$ $Y(T)$

h_{-1}

$X(0)$ $X(1)$ $X(2)$ $X(T-2)$ $X(T-1)$ $X(T)$

jacobian

$$\nabla_{Z^{(1)}{}_{(T)}} DIV = \nabla_{h(T)} DIV \; \nabla_{Z^{(1)}(T)} h(T)$$

$$\frac{dDIV}{dZ_i^{(1)}(T)} = \frac{dDIV}{dh_i(T)} \frac{dh_i(T)}{dZ_i^{(1)}(T)}$$
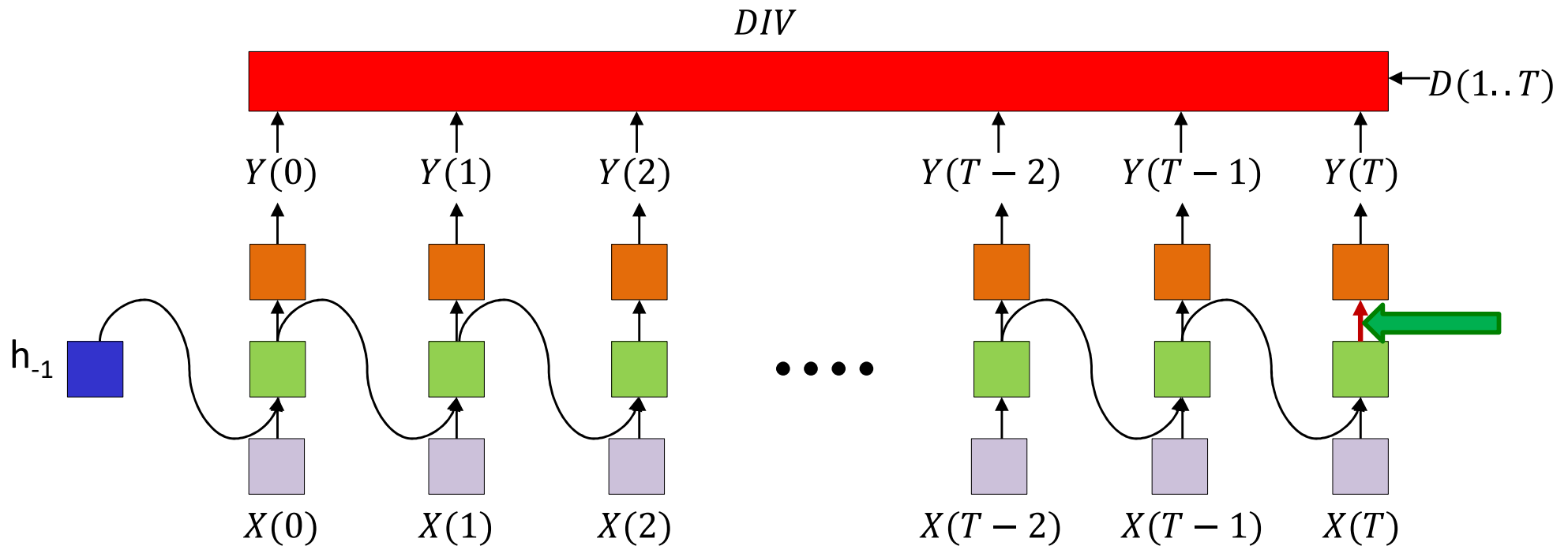
$$\frac{dDIV}{dZ_i^{(2)}(T)} = \frac{dDIV}{dY_i(T)} \frac{dY_i(T)}{dZ_i^{(2)}(T)}$$

$$\frac{dDIV}{dh_i(T)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dZ_j^{(2)}(T)}$$

$$\frac{dDIV}{dw_{ij}^{(2)}} = \frac{dDIV}{dZ_j^{(2)}(T)} h_i(T)$$

Back Propagation Through Time

DIV

$\leftarrow D(1..T)$

$Y(0)$ $Y(1)$ $Y(2)$ $Y(T-2)$ $Y(T-1)$ $Y(T)$

h$_{-1}$

• • • •

$X(0)$ $X(1)$ $X(2)$ $X(T-2)$ $X(T-1)$ $X(T)$

$$\frac{dDIV}{dw_{ij}^{(1)}} = \frac{dDIV}{dZ_j^{(1)}(T)} X_i(T)$$

$$\nabla_{W^{(1)}} DIV = X(T) \nabla_{Z^{(1)}(T)} DIV$$

Back Propagation Through Time

DIV

$D(1..T)$

$Y(0)$ $Y(1)$ $Y(2)$ $Y(T-2)$ $Y(T-1)$ $Y(T)$

h_{-1}

$X(0)$ $X(1)$ $X(2)$ $X(T-2)$ $X(T-1)$ $X(T)$

$$\nabla_{W^{(11)}} DIV = h(T-1) \nabla_{Z^{(1)}(T)} DIV$$

$$\frac{dDIV}{dw_{ij}^{(1)}} = \frac{dDIV}{dZ_j^{(1)}(T)} X_i(T)$$

$$\frac{dDIV}{dw_{ij}^{(11)}} = \frac{dDIV}{dZ_j^{(1)}(T)} h_i(T-1)$$

92

Back Propagation Through Time
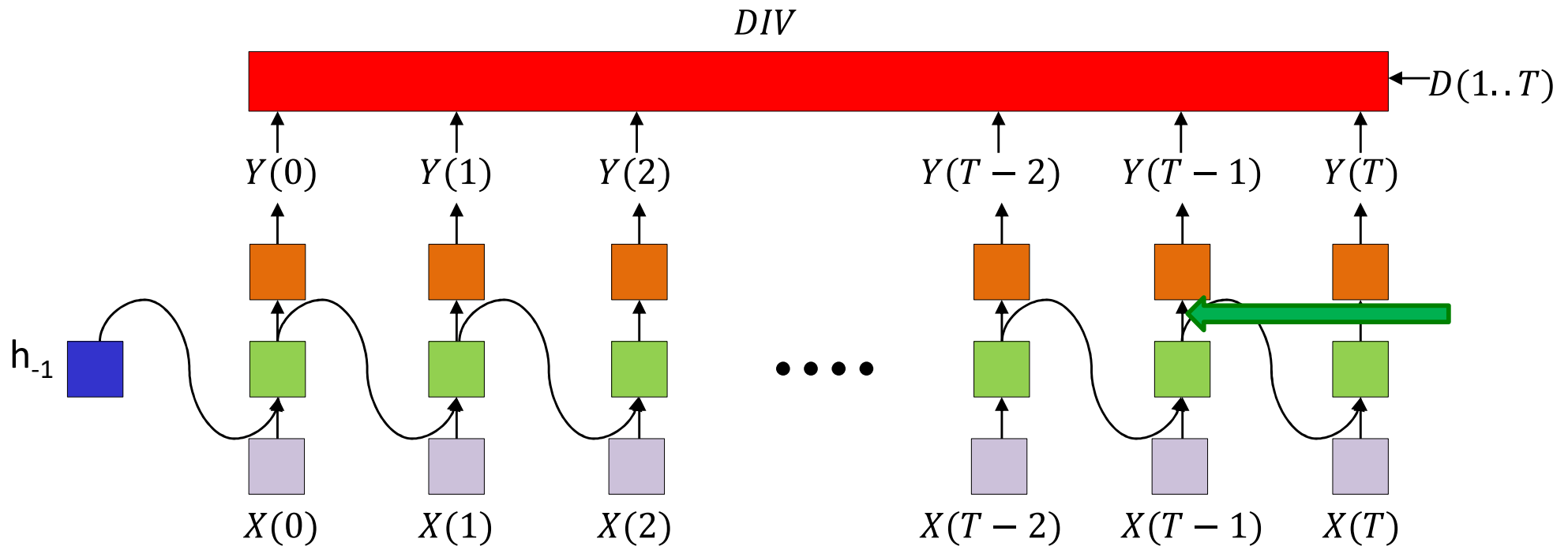
$$\nabla_{Z^{(2)}(T-1)} DIV = \nabla_{Y(T-1)} DIV \; \nabla_{Z^{(2)}(T-1)} Y(T-1)$$
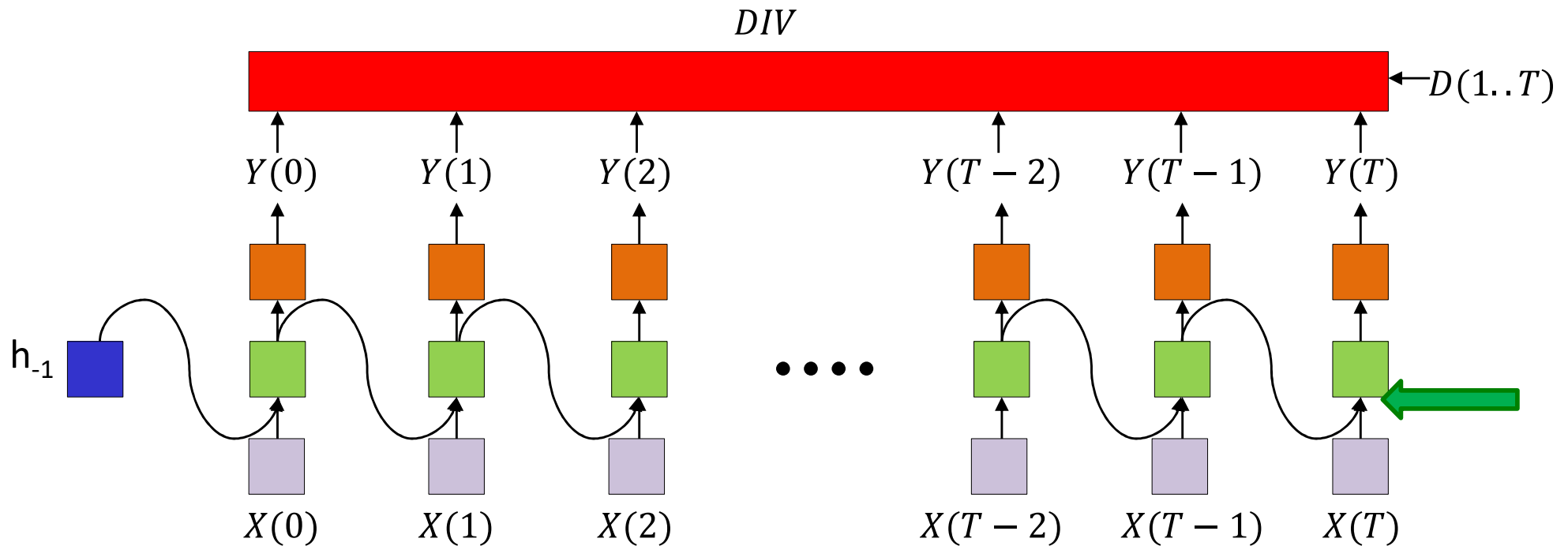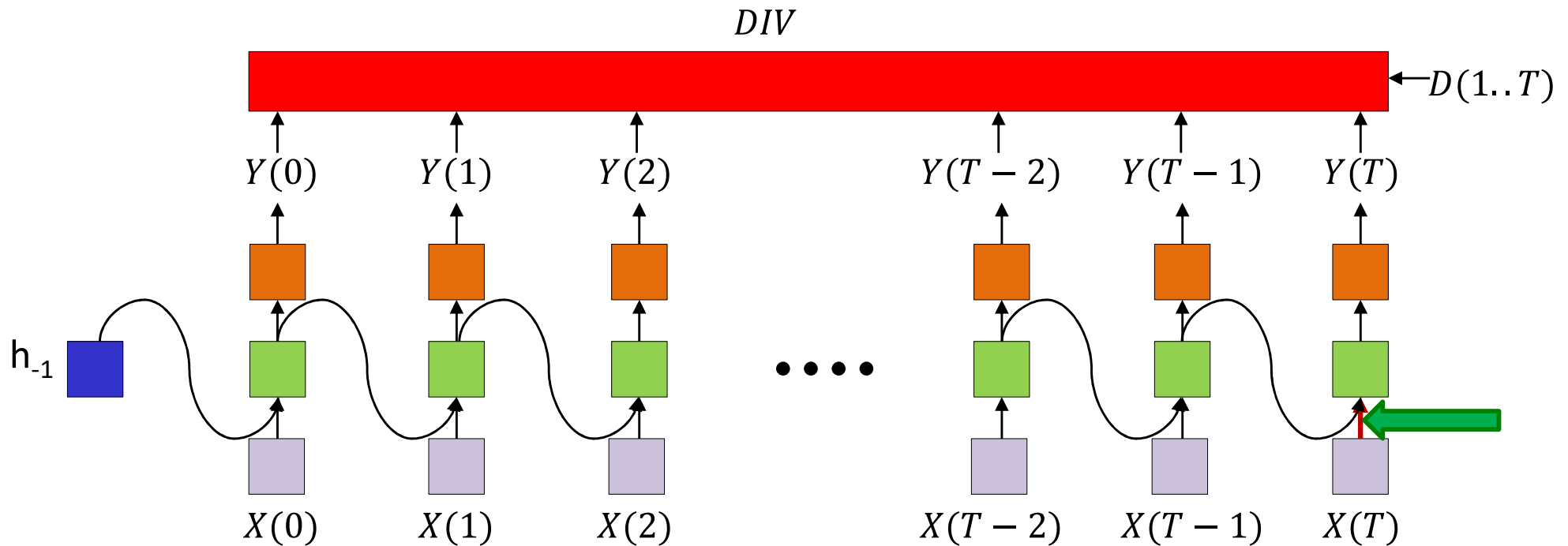
Vector output activation

$$\frac{dDIV}{dZ_i^{(2)}(T-1)} = \frac{dDIV}{dY_i(T-1)} \frac{dY_i(T-1)}{dZ_i^{(2)}(T-1)}$$

OR

$$\frac{dDIV}{dZ_i^{(2)}(T-1)} = \sum_j \frac{dDIV}{dY_j(T-1)} \frac{dY_j(T-1)}{dZ_i^{(2)}(T-1)}$$

Back Propagation Through Time

DIV
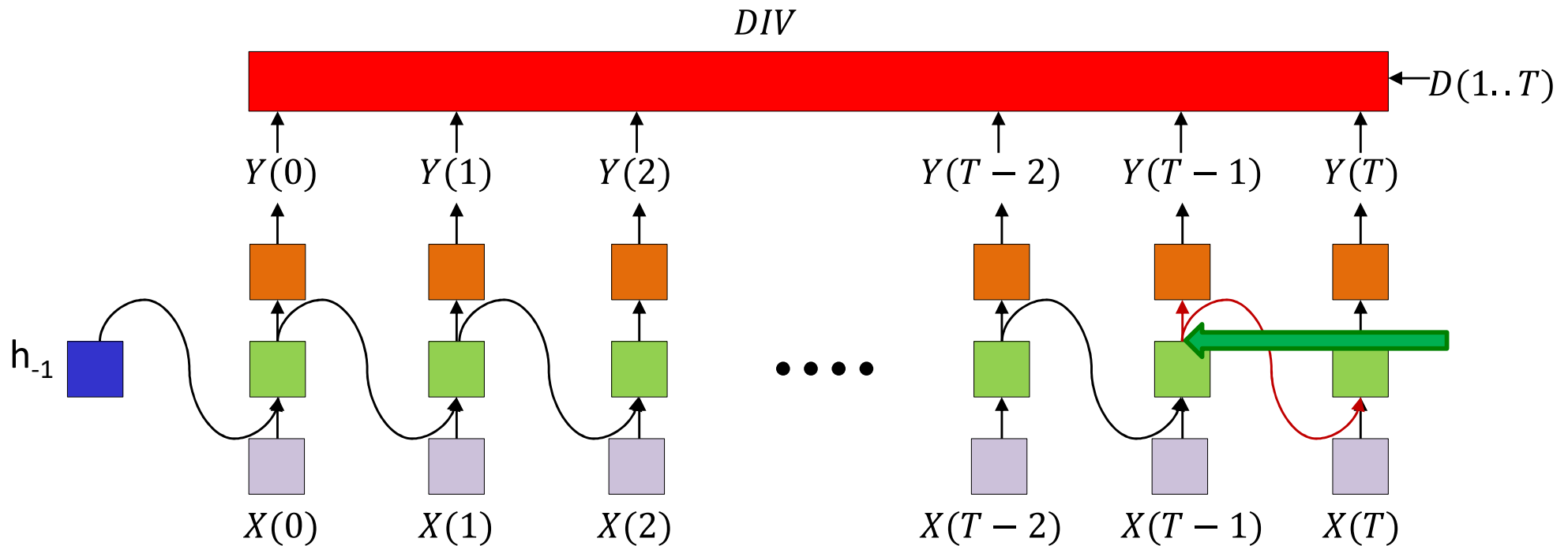
$$\frac{dDIV}{dh_i(T-1)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dZ_j^{(2)}(T-1)} + \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(1)}(T)}$$

$$\nabla_{h(T-1)}DIV = \nabla_{Z^{(2)}(T-1)}DIV \, W^{(2)} + \nabla_{Z^{(1)}(T)}DIV \, W^{(11)}$$

94

Back Propagation Through Time

DIV

$\leftarrow D(1..T)$

$Y(0)$ $Y(1)$ $Y(2)$ $Y(T-2)$ $Y(T-1)$ $Y(T)$

h_{-1}

$X(0)$ $X(1)$ $X(2)$ $X(T-2)$ $X(T-1)$ $X(T)$
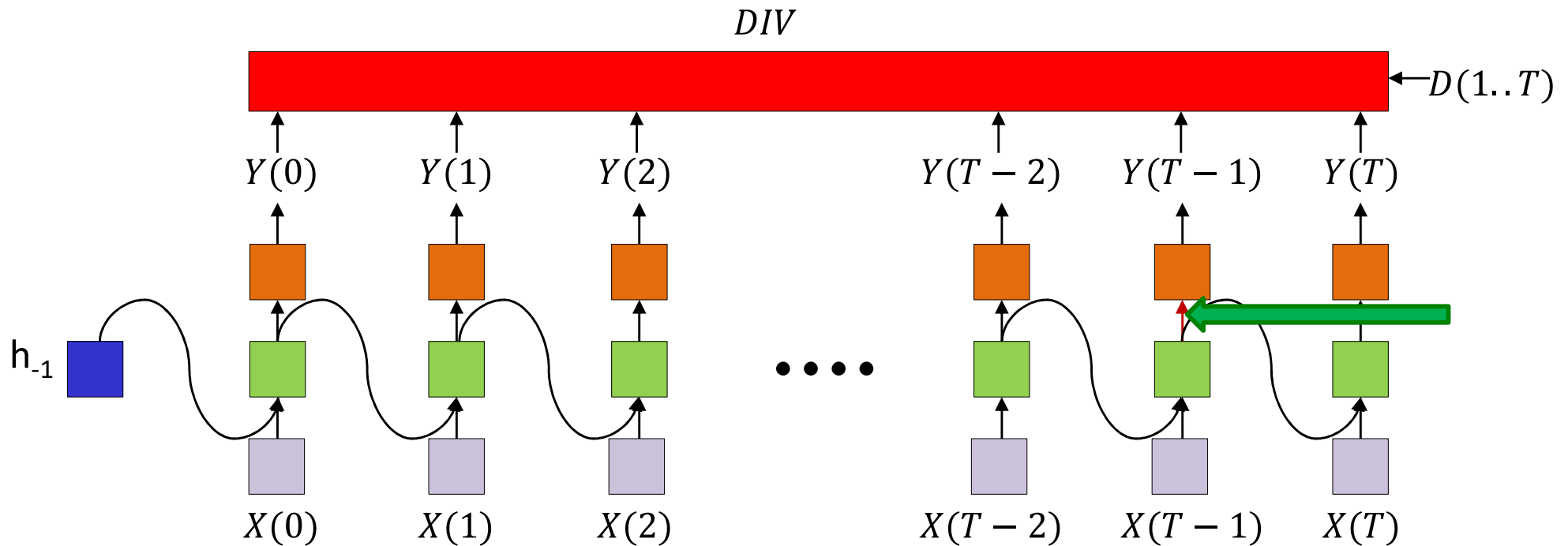
$$\frac{dDIV}{dh_i(T-1)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dZ_j^{(2)}(T-1)} + \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(1)}(T)}$$

Note the addition $\Rightarrow$

$$\frac{dDIV}{dw_{ij}^{(2)}} \mathrel{+}= \frac{dDIV}{dZ_j^{(2)}(T-1)} h_i(T-1)$$

Note the addition $\Rightarrow$

$$\nabla_{W^{(2)}} DIV \mathrel{+}= h(T-1)\nabla_{Z^{(2)}(T-1)} DIV$$

95

Back Propagation Through Time

$$\frac{dDIV}{dZ_i^{(1)}(T-1)} = \frac{dDIV}{dh_i(T-1)}\frac{dh_i(T-1)}{dZ_i^{(1)}(T-1)}$$

$$\nabla_{Z^{(1)}(T-1)}DIV = \nabla_{h(T-1)}DIV \ \nabla_{Z^{(1)}(T-1)}h(T-1)$$

Back Propagation Through Time

$$DIV$$

$$\frac{dDIV}{dZ_i^{(1)}(T-1)} = \frac{dDIV}{dh_i(T-1)} \frac{dh_i(T-1)}{dZ_i^{(1)}(T-1)}$$

$$\frac{dDIV}{dw_{ij}^{(1)}} += \frac{dDIV}{dZ_j^{(1)}(T-1)} X_i(T-1)$$

Note the addition

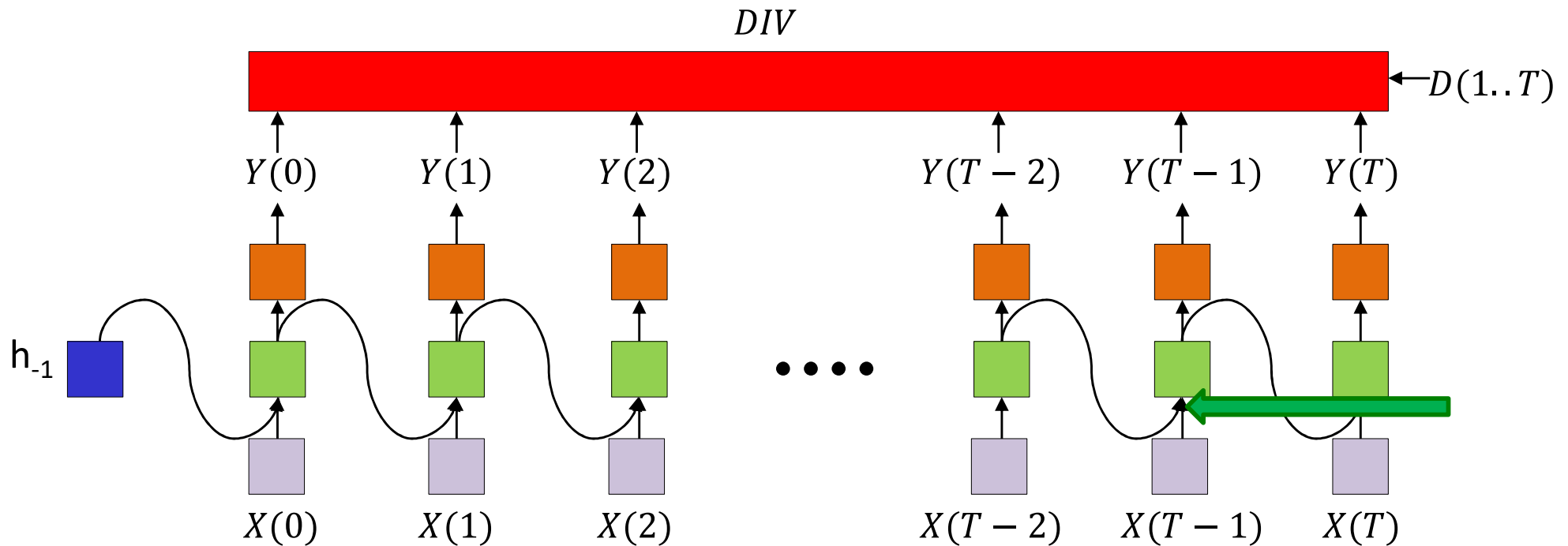$$\nabla_{W^{(1)}} DIV += X(T-1) \nabla_{Z^{(1)}(T-1)} DIV$$

Back Propagation Through Time

DIV

$\leftarrow D(1..T)$

$Y(0)$ $Y(1)$ $Y(2)$ $Y(T-2)$ $Y(T-1)$ $Y(T)$

h_{-1}

$X(0)$ $X(1)$ $X(2)$ $X(T-2)$ $X(T-1)$ $X(T)$
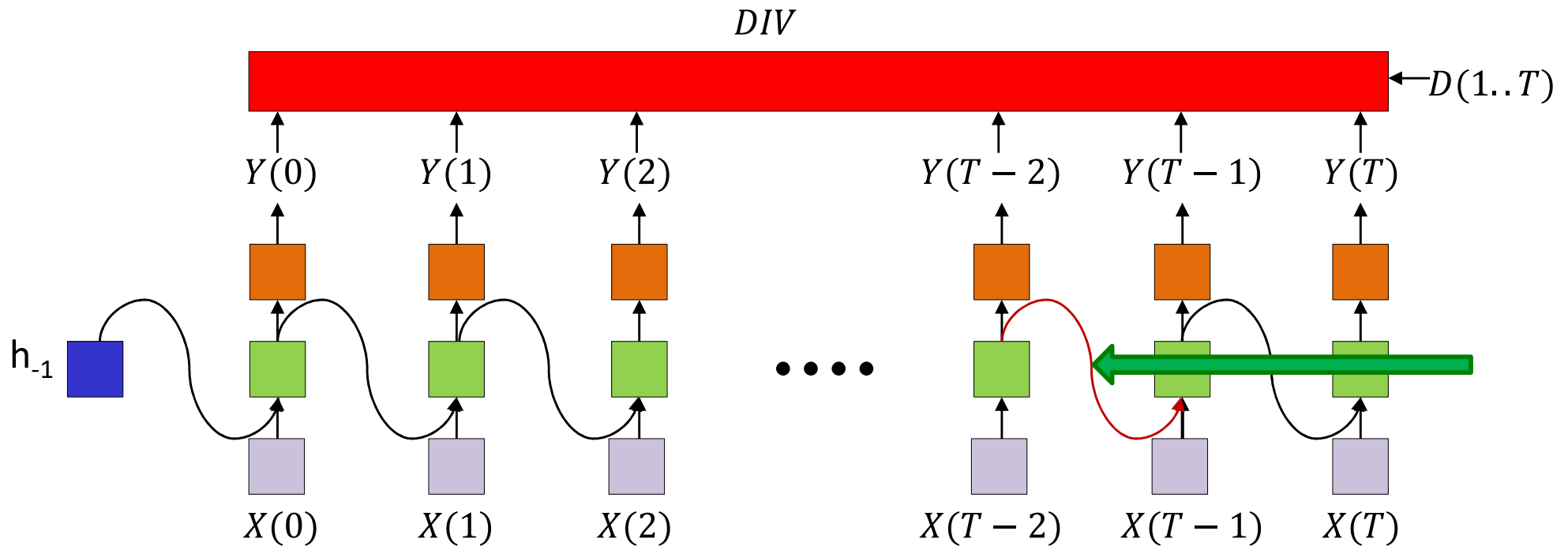
$$\frac{dDIV}{dw_{ij}^{(1)}} += \frac{dDIV}{dZ_j^{(1)}(T-1)} X_i(T-1)$$

Note the addition $\Rightarrow$

$$\frac{dDIV}{dw_{ij}^{(11)}} += \frac{dDIV}{dZ_j^{(1)}(T-1)} h_i(T-2)$$

Note the addition $\Rightarrow$
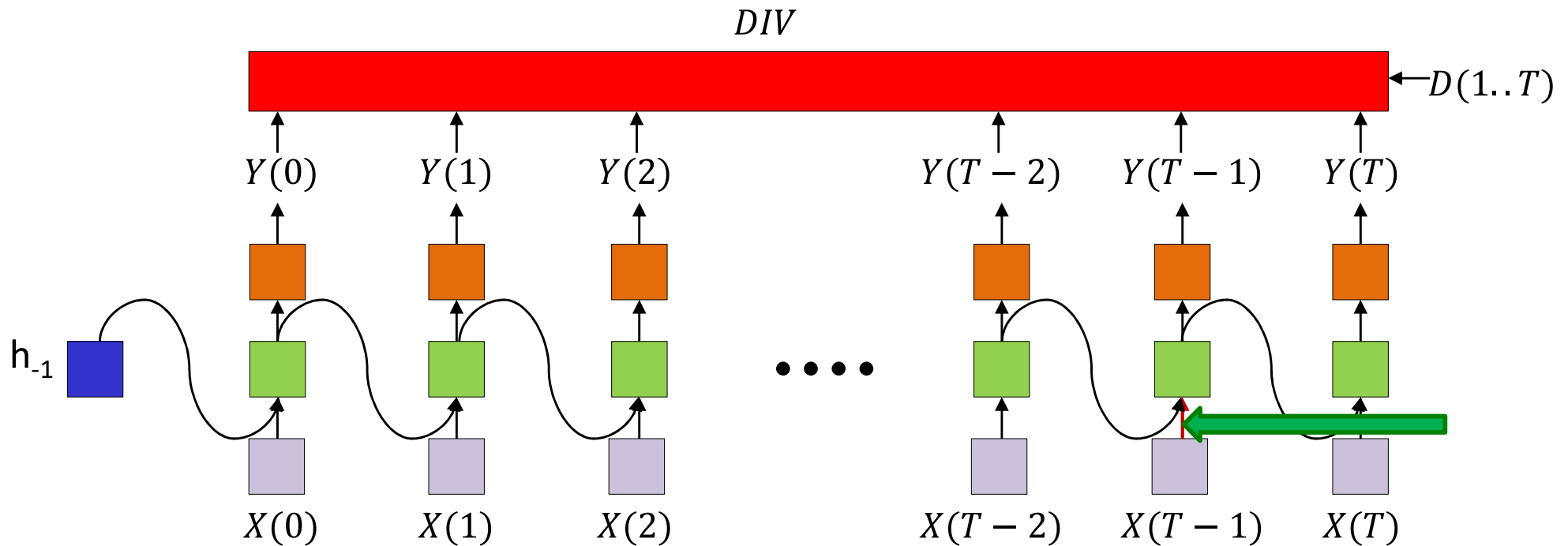
$$\nabla_{W^{(11)}} DIV += h(T-2) \nabla_{Z^{(1)}(T-1)} DIV$$

8

Back Propagation Through Time

Continue computing derivatives going backward through time until..

$$\frac{dDIV}{dh_i(-1)} = \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(1)}(0)}$$

$$\nabla_{h_{-1}} DIV = \nabla_{Z^{(1)}(0)} DIV W^{(11)}$$

Back Propagation Through Time

DIV

$\leftarrow D(1..T)$

$Y(0)$ $Y(1)$ $Y(2)$ $Y(T-2)$ $Y(T-1)$ $Y(T)$

h₋₁

$X(0)$ $X(1)$ $X(2)$ $X(T-2)$ $X(T-1)$ $X(T)$
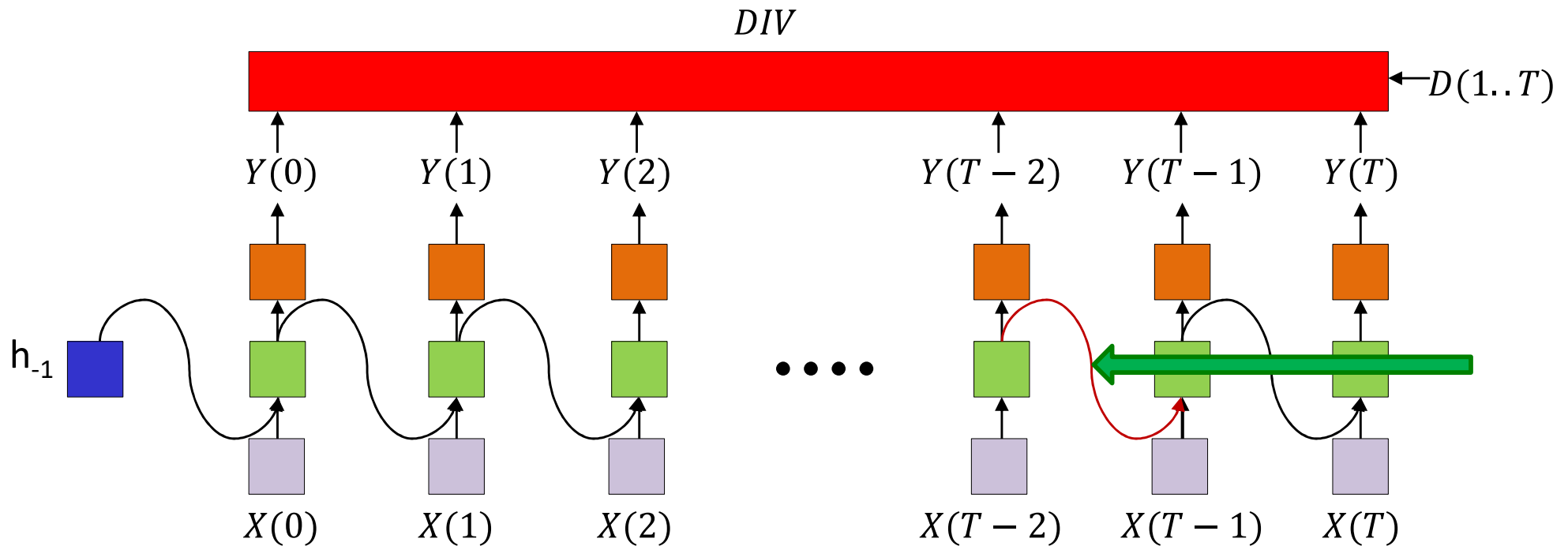
Initialize all derivatives to 0

For t = T downto 0

$$\nabla_{Z^{(2)}(t)} DIV = \nabla_{Y(t)} DIV\, \nabla_{Z^{(2)}(t)} Y(t)$$

$$\nabla_{h(t)} DIV = \nabla_{Z^{(2)}(t)} DIV\, W^{(2)} + \nabla_{Z^{(1)}(t+1)} DIV\, W^{(11)}$$

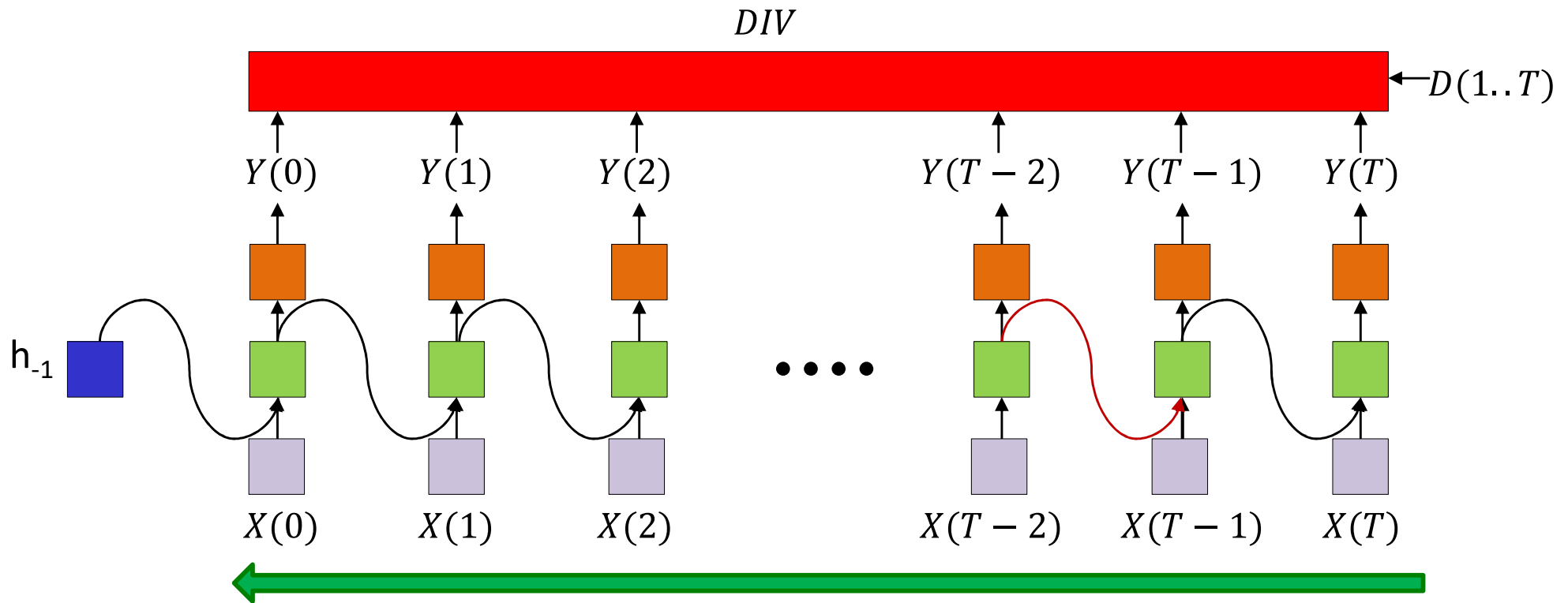$$\nabla_{Z^{(1)}(t)} DIV = \nabla_{h(t)} DIV\, \nabla_{Z^{(1)}(t)} h(t)$$

$$\nabla_{W^{(2)}} DIV \mathrel{+}= h(t)\nabla_{Z^{(2)}(t)} DIV$$

$$\nabla_{W^{(11)}} DIV \mathrel{+}= h(t-1)\nabla_{Z^{(1)}(t)} DIV$$

$$\nabla_{W^{(1)}} DIV \mathrel{+}= X(t)\nabla_{Z^{(1)}(t)} DIV$$

$$\nabla_{h_{-1}} DIV = \nabla_{Z^{(1)}(0)} DIV\, W^{(11)}$$

0

Back Propagation Through Time
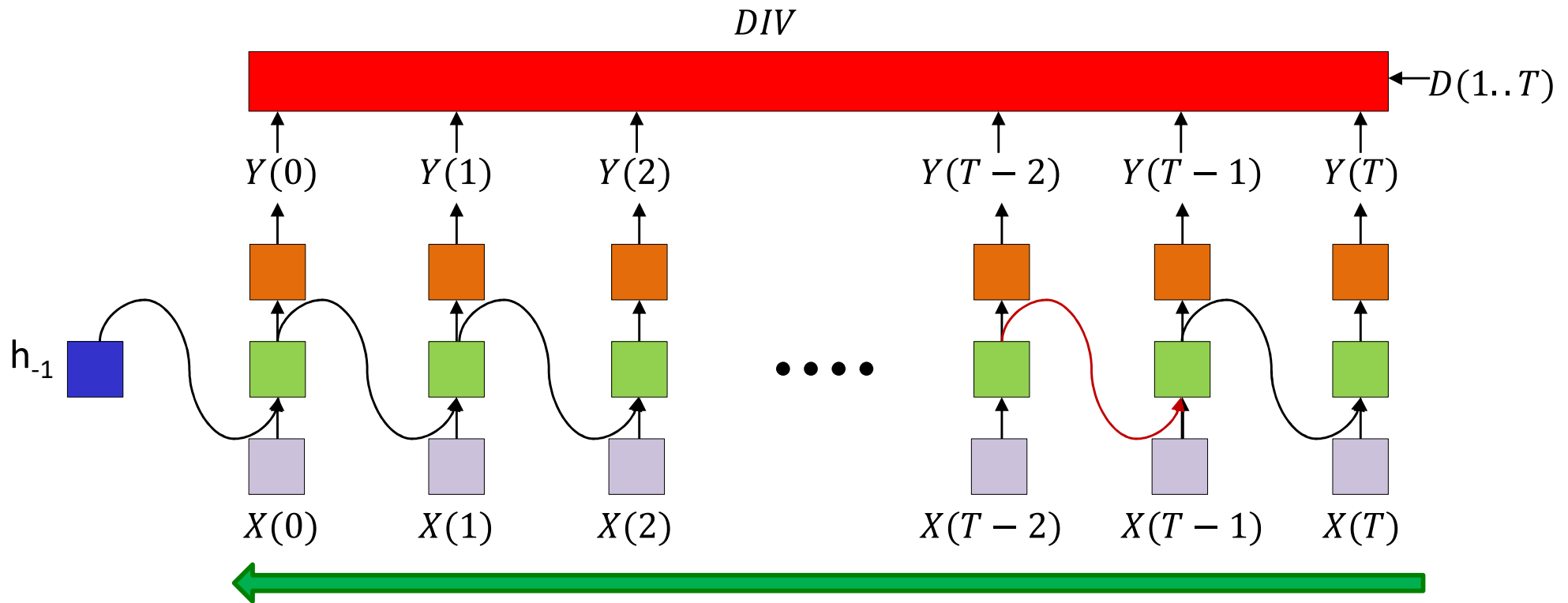
$$\frac{dDIV}{dh_i^{(k)}(t)} = \sum_j w_{i,j}^{(k+1)} \frac{dDIV}{dZ_j^{(k+1)}(t)} + \sum_j w_{i,j}^{(k,k)} \frac{dDIV}{dZ_j^{(k)}(t+1)}$$

Not showing derivatives
at output neurons

$$\frac{dDIV}{dZ_i^{(k)}(t)} = \frac{dDIV}{dh_i^{(k)}(t)} f_k'\left(Z_i^{(k)}(t)\right)$$

Back Propagation Through Time
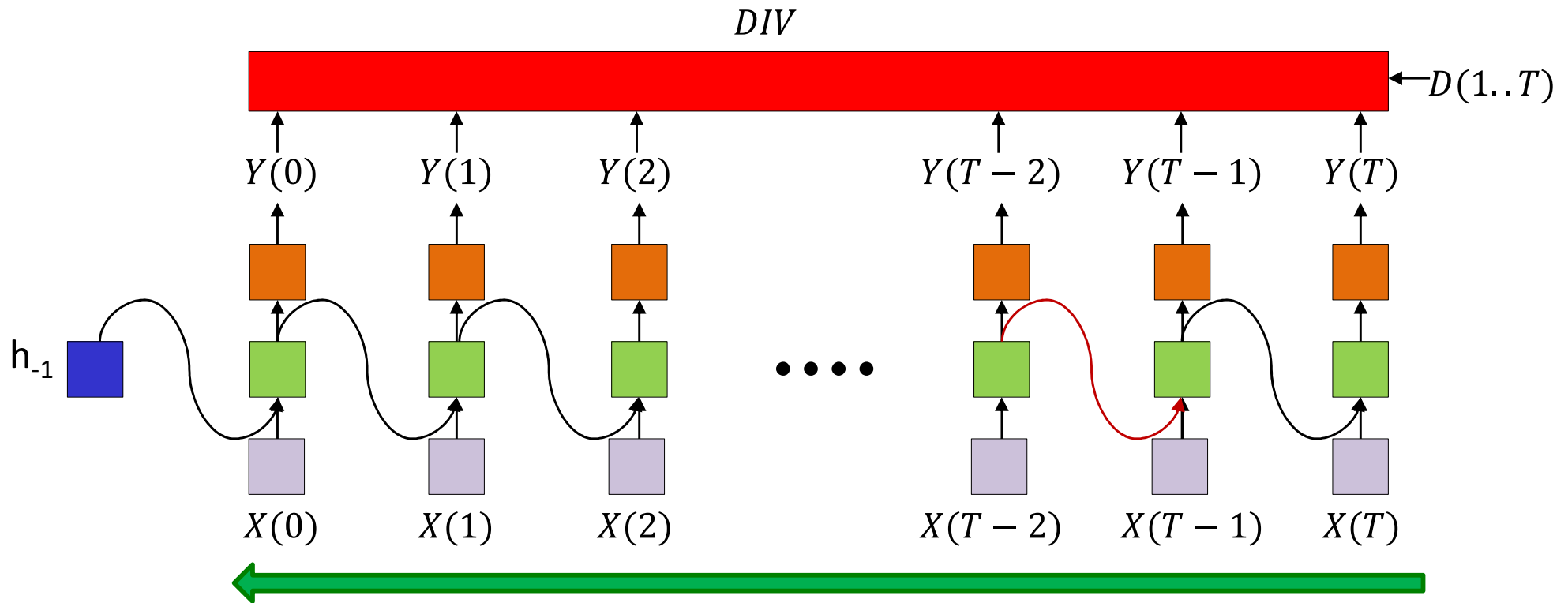
$$\frac{dDIV}{dh_i(-1)} = \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(1)}(0)}$$

$$\frac{dDIV}{dw_{ij}^{(1)}} = \sum_t \frac{dDIV}{dZ_j^{(1)}(t)} X_i(t)$$

$$\frac{dDIV}{dw_{ij}^{(11)}} = \sum_t \frac{dDIV}{dZ_j^{(1)}(t)} h_i(t-1)$$

BPTT

```
# Assuming forward pass has been completed
# Jacobian(x,y) is the jacobian of x w.r.t. y
# Assuming dY(t) = gradient(div,Y(t)) available for all t
# Assuming all dz, dh, dW and db are initialized to 0

for t = T-1:downto:0   # Backward through time
    dz_o(t) = dY(t)Jacobian(Y(t),z_o(t))
    dW_o += h(t,L)dz_o(t)
    db_o += dz_o(t)
    dh(t,L) += dz_o(t)W_o

    for l = L:1   # Reverse through layers
        dz(t,l) = dh(t,l)Jacobian(h(t,l),z(t,l))
        dh(t,l-1) += dz(t,l) W_c(l)
        dh(t-1,l) = dz(t,l) W_r(l)


        dW_c(l) += h(t,l-1)dz(t,l)
        dW_r(l) += h(t-1,l)dz(t,l)
        db(l) += dz(t,l)
```

Subscript "c" – current
Subscript "r" – recurrent

BPTT

one to one

one to many

many to one

many to many

many to many
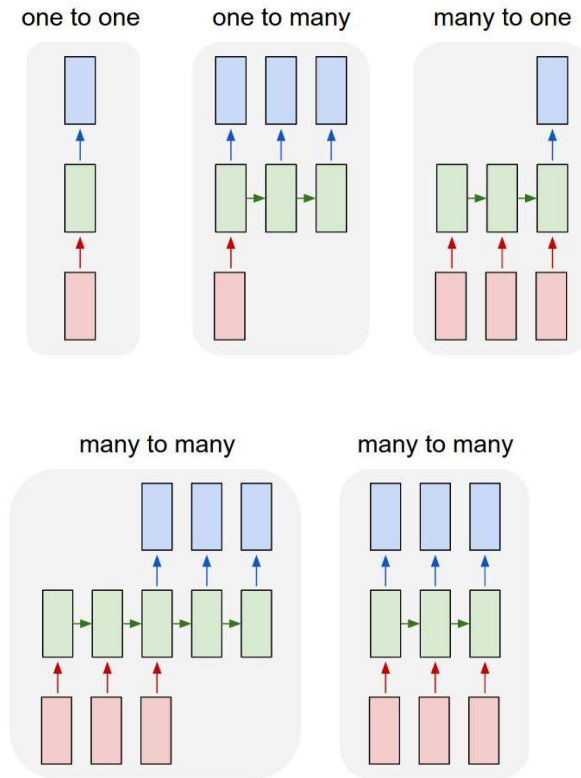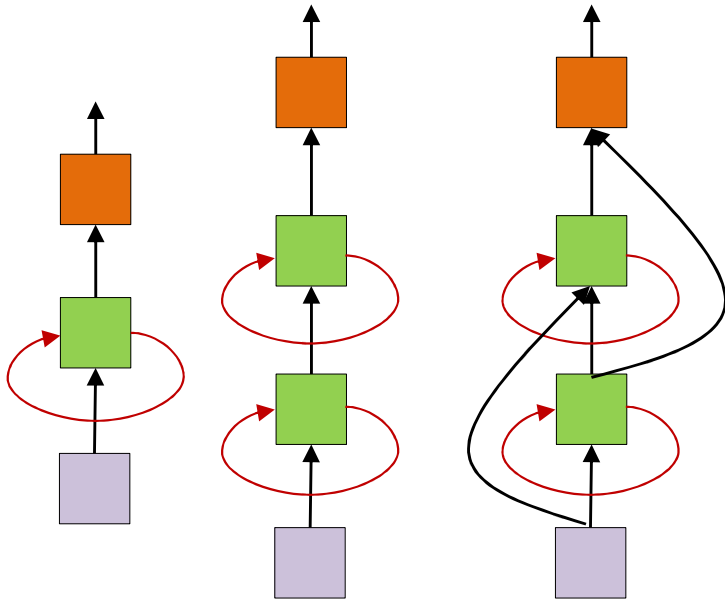
- Can be generalized to any architecture

Poll 3

- @868, @869

SGD trains neural networks one input at a time, rather than over batches of inputs. The corresponding equivalent for RNNs would be to update the network after each input vector: True or False

- True
- False

Select all that are true:

- In RNNs the divergence we minimize is the sum of the divergences for the individual inputs in the time series
- The divergence is the divergence between the actual sequence of outputs and the desired sequence of outputs and cannot always be decomposed into the sum of divergences at individual time steps.

Poll 3

SGD trains neural networks one input at a time, rather than over batches of inputs. The corresponding equivalent for RNNs would be to update the network after each input vector: True or False

- True
- **False**

Select all that are true:

- In RNNs the divergence we minimize is the sum of the divergences for the individual inputs in the time series
- **The divergence is the divergence between the actual sequence of outputs and the desired sequence of outputs and cannot always be decomposed into the sum of divergences at individual time steps.**

Extensions to the RNN: *Bidirectional RNN*

Bidirectional RNN (BRNN)

Output Layer

Backward Layer

Forward Layer

Input Layer

w6   w6   w6
w5   w5   w5   w5
w4   w4   w4
w3   w3   w3
w2   w2   w2   w2
w1   w1   w1
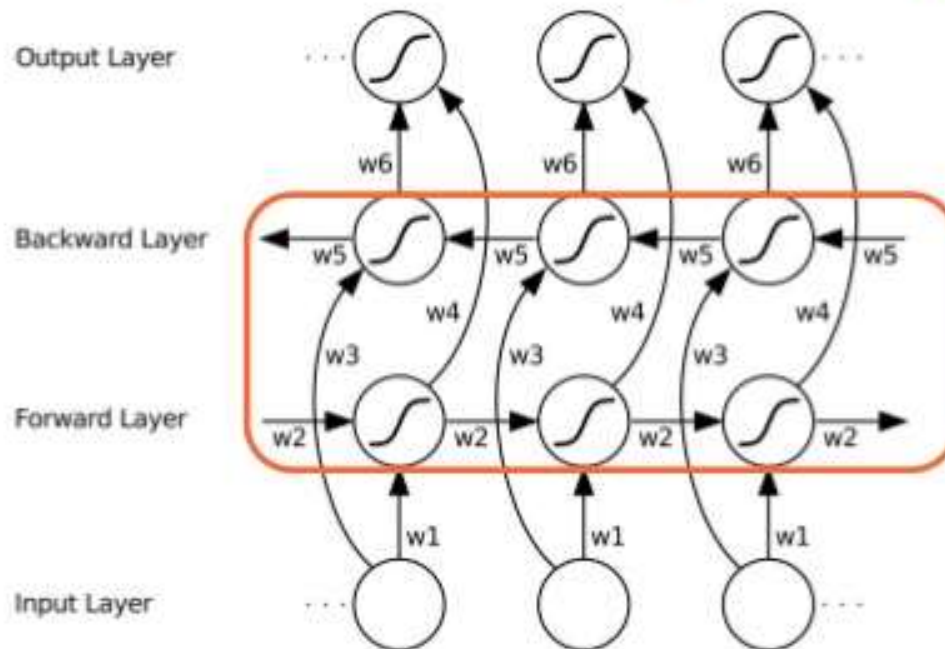
Must learn weights w2, w3, w4 & w5; in addition to w1 & w6.

Proposed by Schuster and Paliwal 1997

Alex Graves, "Supervised Sequence Labelling with Recurrent Neural Networks"

- In problems where the entire input sequence is available before we compute the output, RNNs can be bidirectional

- RNN with both forward and backward recursion
  - Explicitly models the fact that just as the future can be predicted from the past, the past can be deduced from the future

Bidirectional RNN
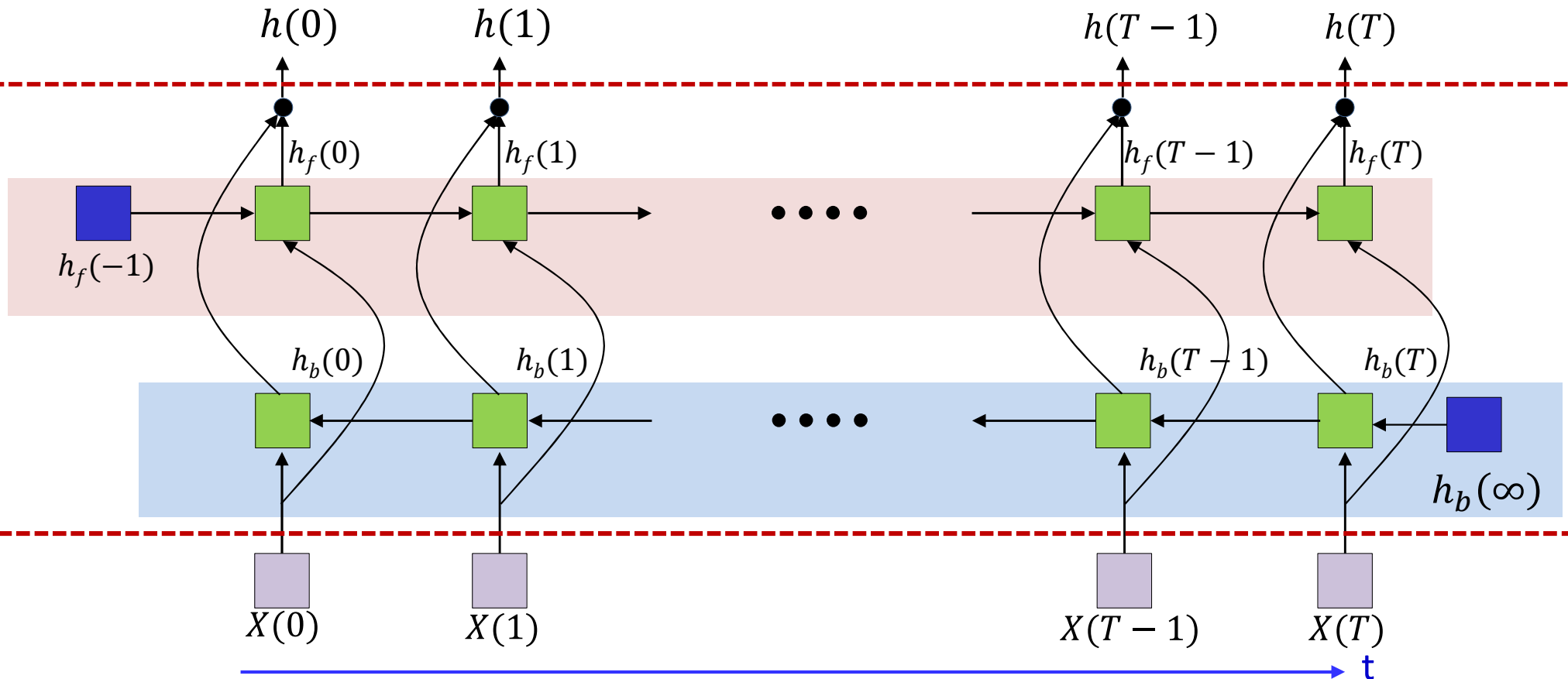
- "Block" performs bidirectional inference on input
  - "Input" could be input series X(0)...X(T) or the output of a previous layer (or block)

- The Block has two components
  - A forward net process the data from t=0 to t=T
  - A backward net processes it backward from t=T down to t=0

Bidirectional RNN block
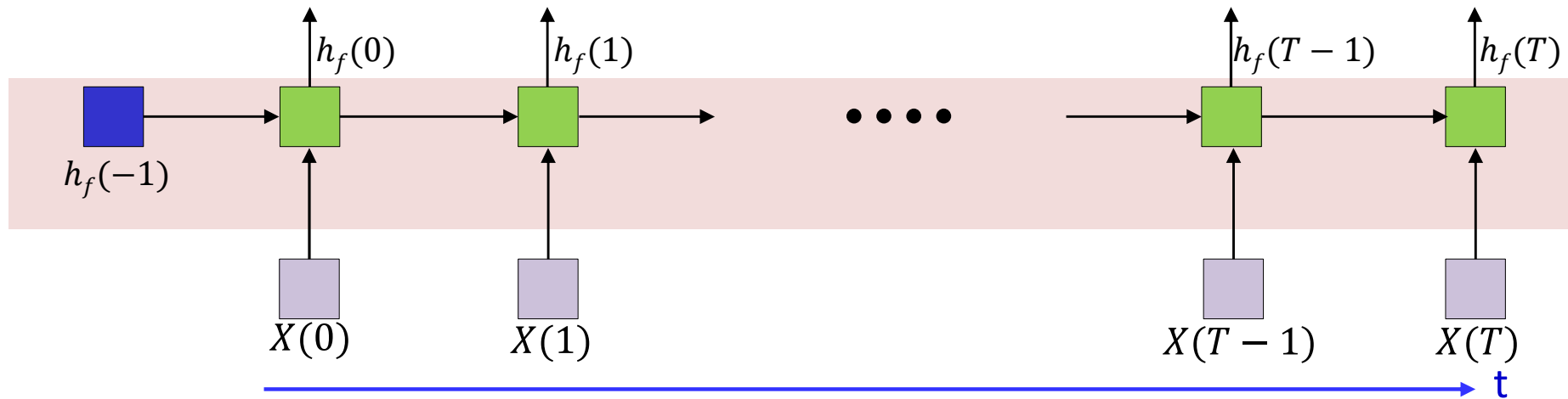
- The forward net process the data from t=0 to t=T
  - Only computing the hidden state values.

Bidirectional RNN block
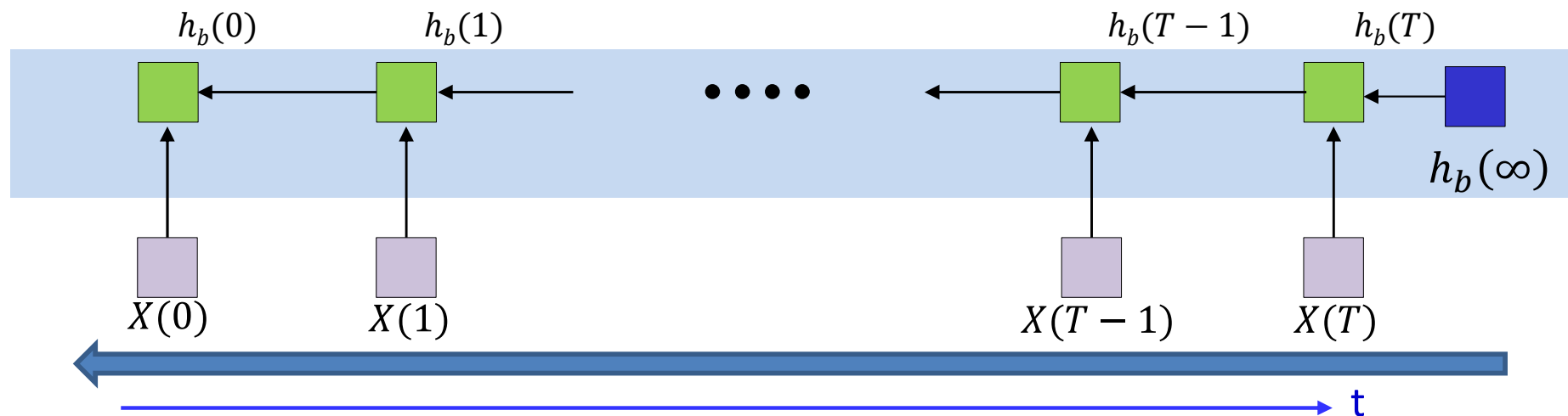
- The backward nets processes the input data in *reverse time,* end to beginning
  - Initially only the hidden state values are computed
    - Clearly, this is not an online process and requires the *entire* input data
  - Note: *This is not the backward pass of backprop.*

Bidirectional RNN block
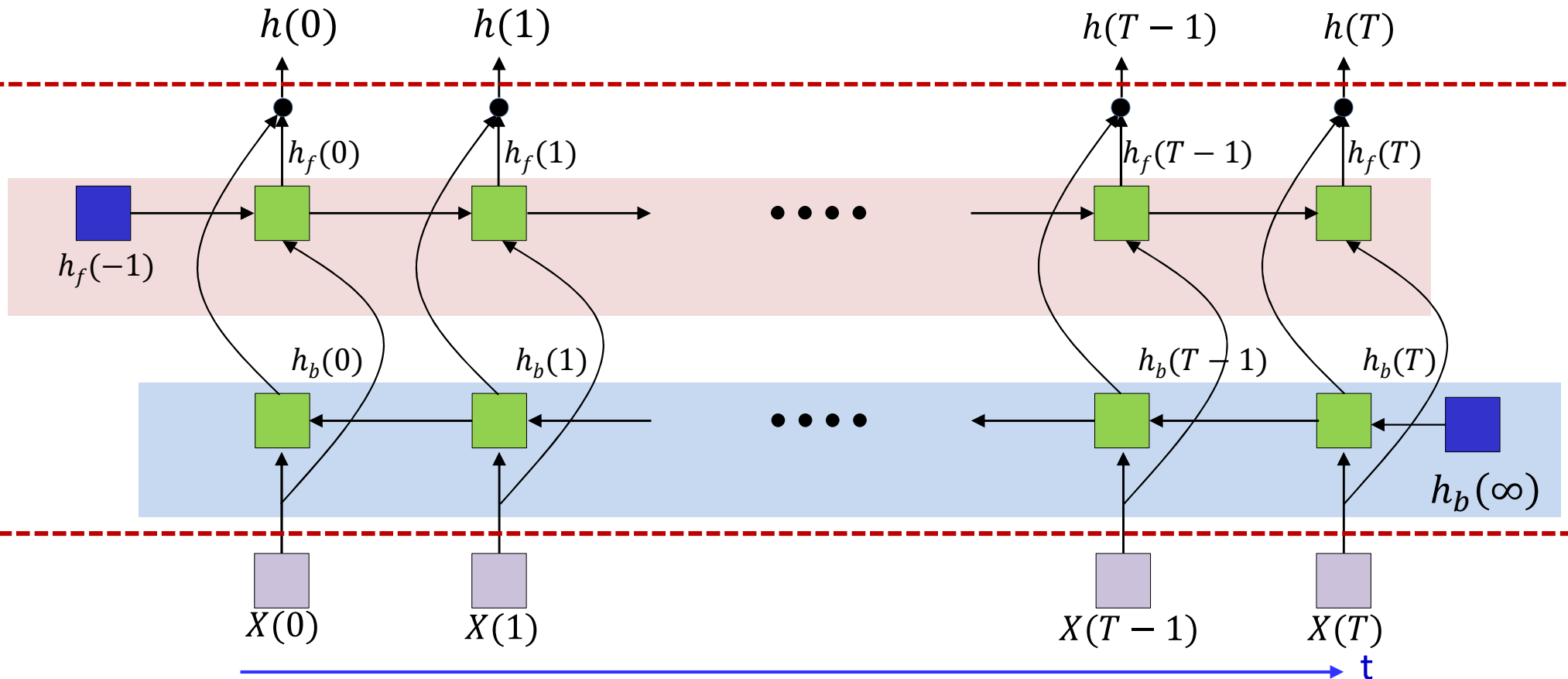
- The computed states of both networks are combined to give you the output of the bidirectional block
  - Typically just concatenate them

$$h(t) = [h_f(t); h_b(t)]$$

Bidirectional RNN
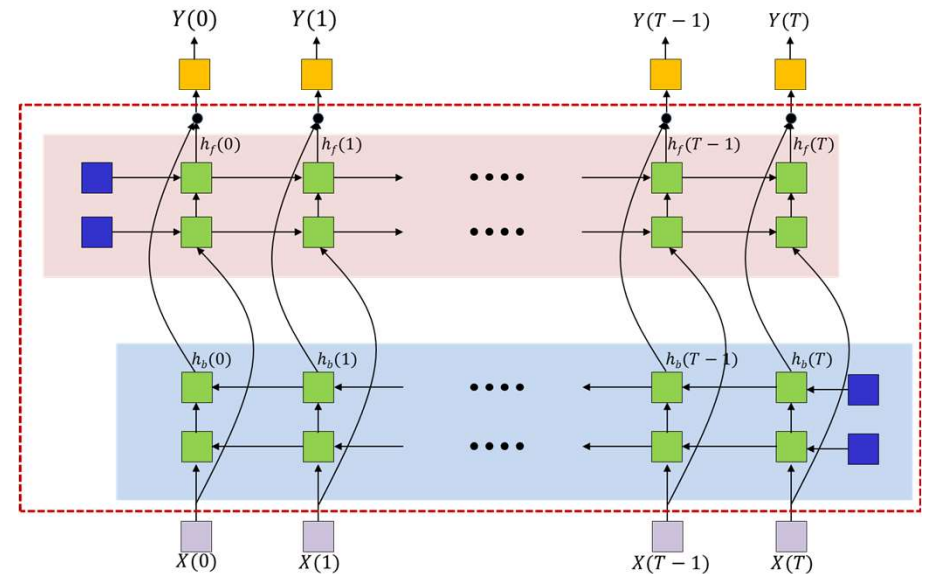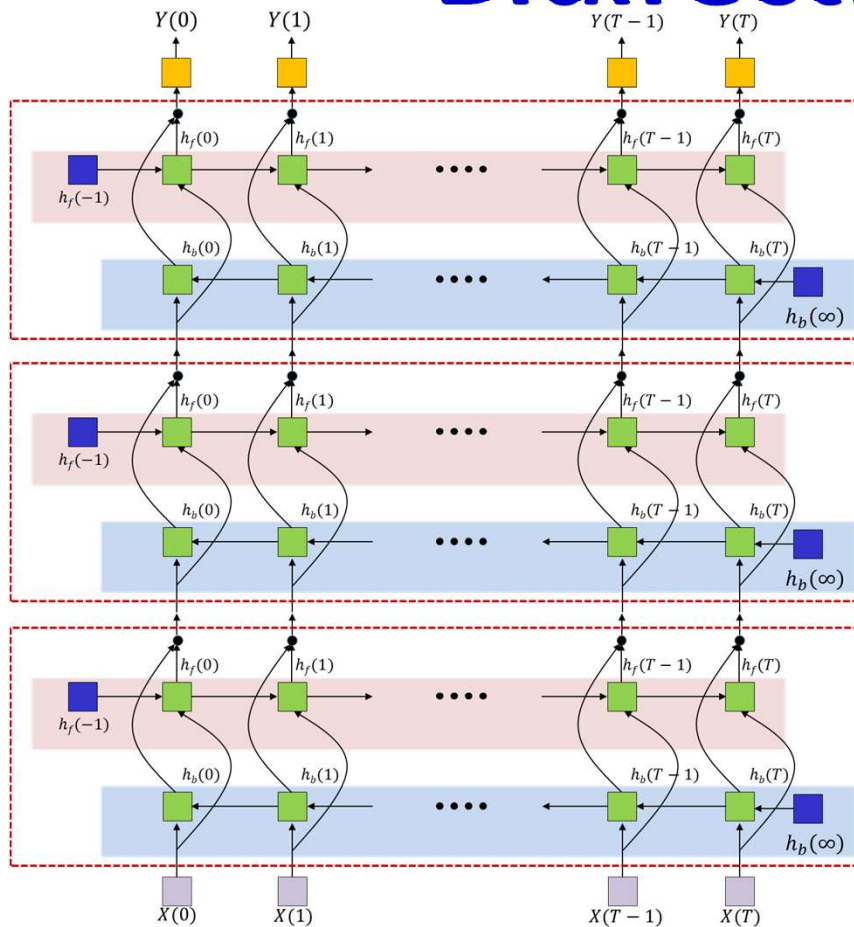
- Actual network may be formed by stacking many independent bidirectional blocks followed by an output layer
  - Forward and backward nets in each block are a single layer
- Or by a single bidirectional block followed by an output layer
  - The forward and backward nets may have several layers
- In either case, it's sufficient to understand forward inference and backprop rules for a single block
  - Full forward or backprop computation simply requires repeated application of these rules

Poll 4

- @870

A day trader in the stock exchange uses RNNs to make predictions from historical market data to decide which stocks to buy. He can use a bidirectional RNN for his task.

- True
- False

Poll 4

A day trader in the stock exchange uses RNNs to make predictions from historical market data to decide which stocks to buy. He can use a bidirectional RNN for his task.

- True
- **False**

Bidirectional RNN block: inference

```
# Subscript f represents forward net, b is backward net
# Assuming h_f(-1,*) and h_b(inf,*) are known
# x(t) is the input to the block (which could be from a lower layer)


#forward recurrence
for t = 0:T-1 # Going forward in time
    h_f(t,0) = x(t) # Vectors. Initialize h_f(0) to input
    for l = 1:L_f  # L_f is depth of forward network hidden layers
        z_f(t,l) = W_fc(l)h_f(t,l-1) + W_fr(l)h_f(t-1,l) + b_f(l)
        h_f(t,l) = tanh(z_f(t,l)) # Assuming tanh activ.


#backward recurrence
h_b(T,:,:) = h_b(inf,:,:) # Just the initial value
for t = T-1:downto:0 # Going backward in time
    h_b(t,0) = x(t) # Vectors. Initialize h_b(0) to input
    for l = 1:L_b # L_b is depth of backward network hidden layers
        z_b(t,l) = W_bc(l)h_b(t,l-1) + W_br(l)h_b(t+1,l) + b_b(l)
        h_b(t,l) = tanh(z_b(t,l)) # Assuming tanh activ.


for t = 0:T-1  # The output combines forward and backward
    h(t) = [h_f(t,L_f); h_b(t,L_b)]
```

Bidirectional RNN: Simplified code

- Code can be made modular and simplified for better interpretability...

First: Define forward recurrence

```
# Inputs:
#    L : Number of hidden layers
#    Wc,Wr,b: current weights,  recurrent weights, biases
#    hinit:  initial value of h(representing h(-1,*))
#    x: input vector sequence
#    T: Length of input vector sequence
# Output:
#    h, z: sequence of pre-and post activation hidden
#          representations from all layers of the RNN


function RNN_forward(L, Wc, Wr, b, hinit, x, T)
    h(-1,:) = hinit  # hinit is the initial value for all layers
    for t = 0:T-1 # Going forward in time
        h(t,0) = x(t) # Vectors. Initialize h(0) to input
        for l = 1:L
            z(t,l) = Wc(l)h(t,l-1) + Wr(l)h(t-1,l) + b(l)
            h(t,l) = tanh(z(t,l)) # Assuming tanh activ.
    return h
```

Bidirectional RNN block

```
# Subscript f represents forward net, b is backward net
# Assuming h_f(-1,*) and h_b(inf,*) are known


#forward pass
h_f = RNN_forward(L_f, W_fc, W_fr, b_f, h_f(-1,:), x, T)


#backward pass
x_rev = fliplr(x)   # Flip it in time
h_brev = RNN_forward(L_b, W_bc, W_br, b_b, h_b(inf,:), x_rev, T)
h_b = fliplr(h_brev)   # Flip back to straighten time


#combine the two for the output
for t = 0:T-1  # The output combines forward and backward
    h(t) = [h_f(t,L_f); h_b(t,L_b)]
```

Backpropagation in BRNNs
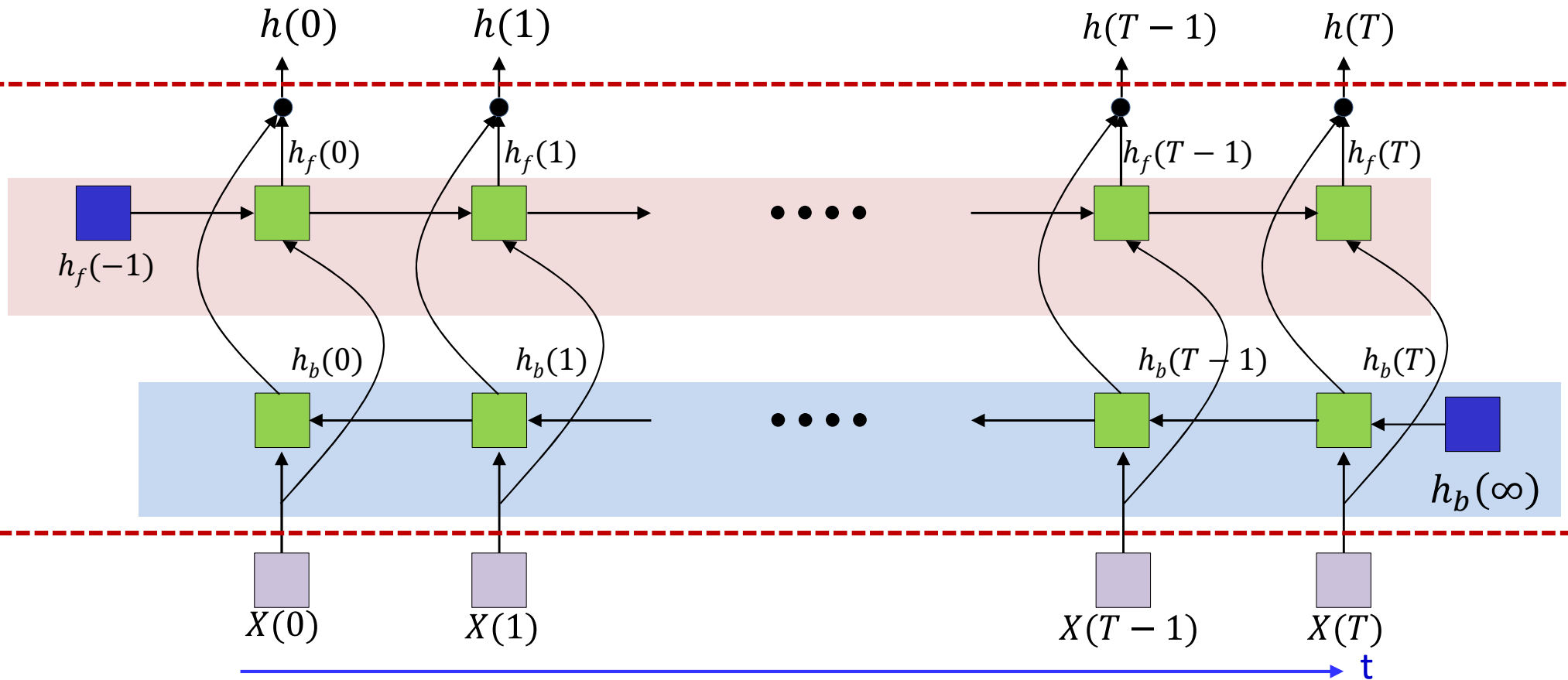
- Forward pass: Compute both forward and backward networks and final output

Backpropagation in BRNNs
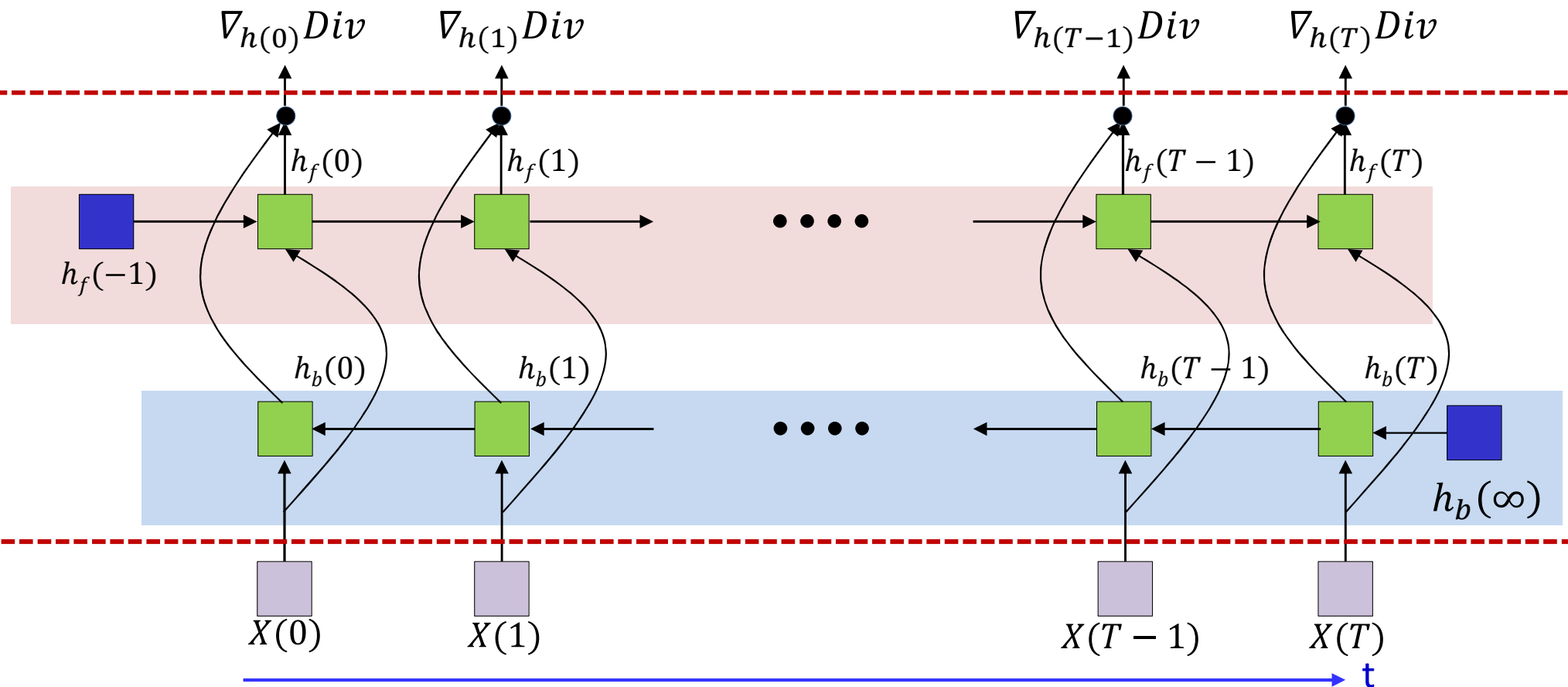
- Backward pass: Assume gradients of the divergence are available for the block outputs $h(t)$
  - Obtained via backpropagation from network output
  - Will have the same dimension (length) as $h(t)$
    - Which is the sum of the dimensions of $h_f(t)$ and $h_b(t)$

Backpropagation in BRNNs
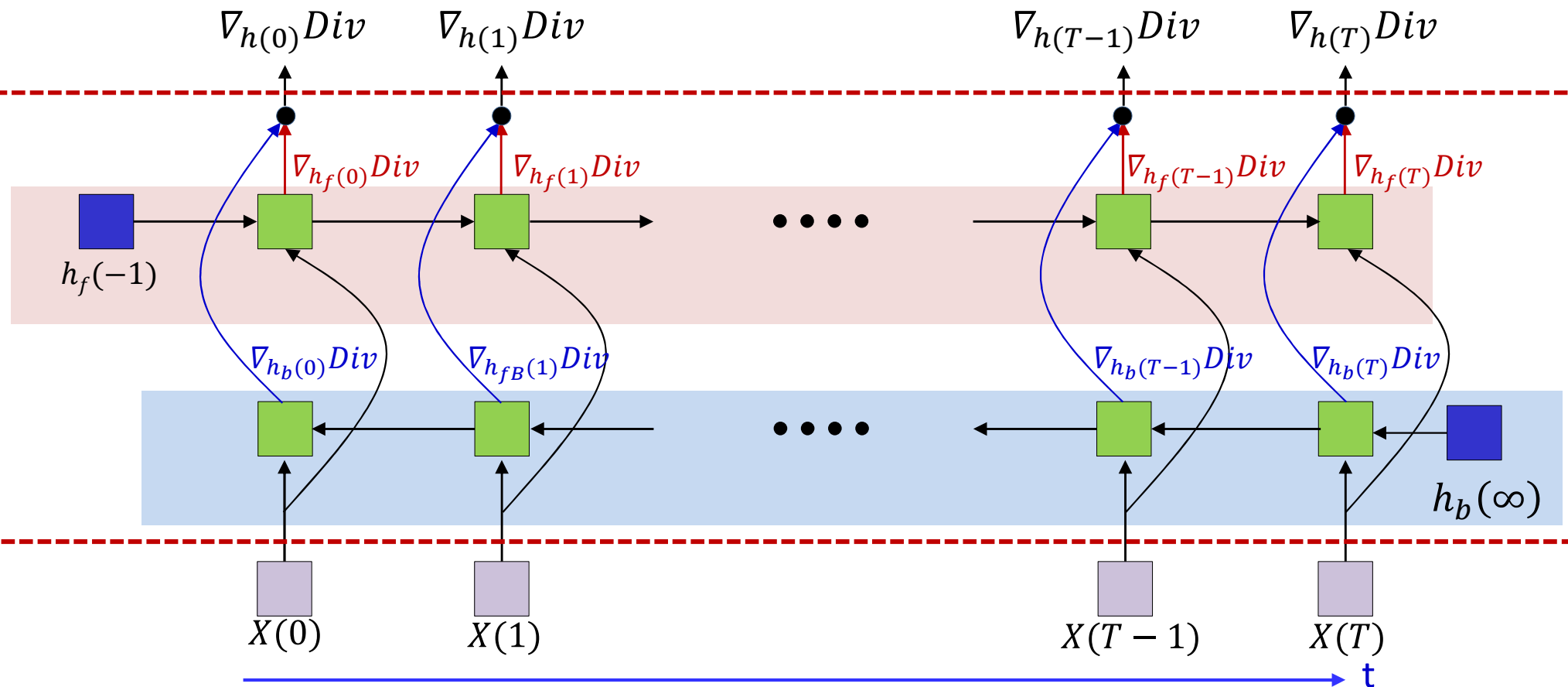
- Separate gradient into forward and backward components

$$\nabla_{h(t)} Div = [\nabla_{h_f(t)} Div; \nabla_{h_b(t)} Div]$$

  - Extract $\nabla_{h_f(t)} Div$ and $\nabla_{h_b(t)} Div$ from $\nabla_{h(t)} Div$.

- Separately perform backprop on the forward and backward nets

Backpropagation in BRNNs
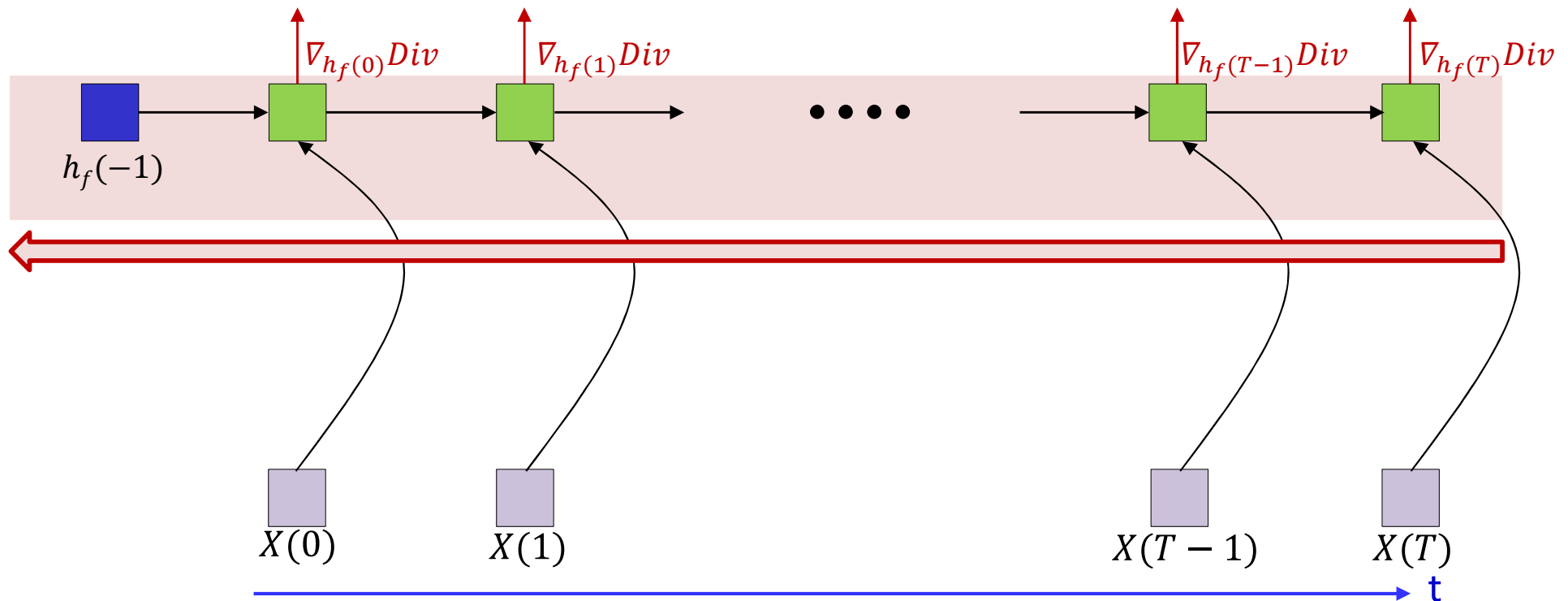
- Backprop for forward net:
  - Backpropagate $\nabla_{h_f(t)} Div$ from $t = T$ down to $t = 0$ in the usual way
  - Will obtain derivatives for all the parameters of the forward net
  - Will also get $\nabla_{X(t)} Div_{forward}$
    - Partial derivative of the gradient for $X(t)$ computed through the forward net

Backpropagation in BRNNs
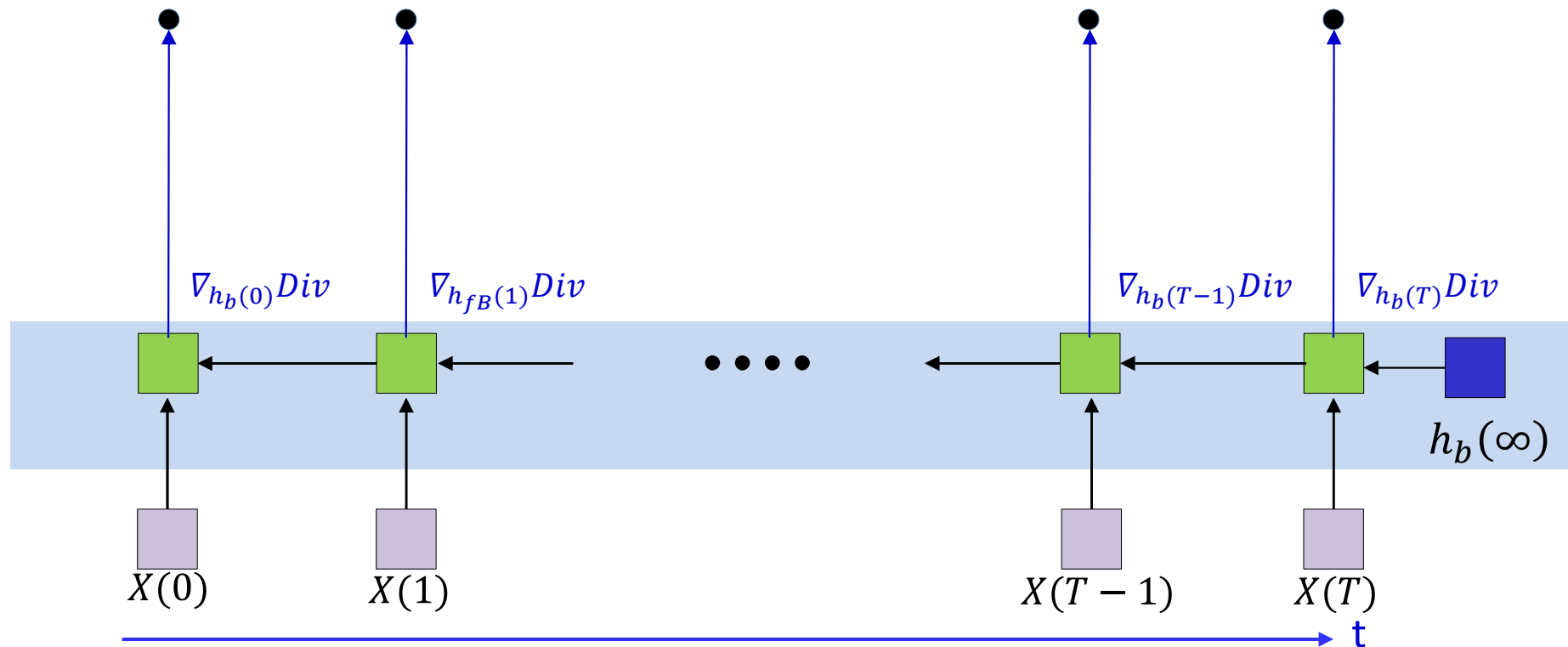
- Backprop for backward net:
  - Backpropagate $\nabla_{h_b(t)}Div$ *forward* from $t = 0$ *up* to $t = T$
  - Will obtain derivatives for all the parameters of the forward net
  - Will also get $\nabla_{X(t)}Div_{backward}$
    - Partial derivative of the gradient for $X(t)$ computed through the backward net

Backpropagation in BRNNs
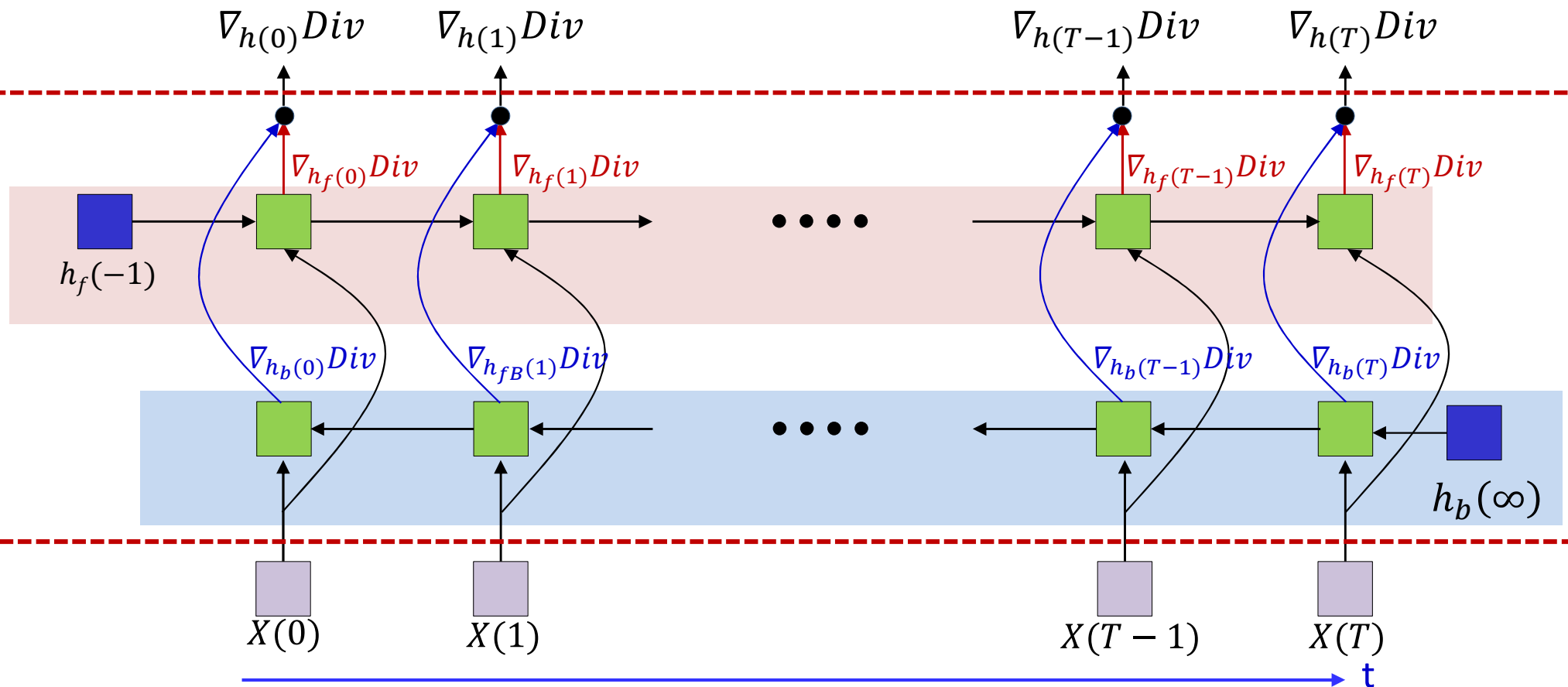
- Finally add up the forward and backward partial derivatives to get the full gradient for $X(t)$

$$\nabla_{X(t)} Div = \nabla_{X(t)} Div_{backward} + \nabla_{X(t)} Div_{forward}$$

Backpropagation: Pseudocode

- As before we will use a 2-step code:
  - A basic backprop routine that we will call
  - Two calls to the routine within a higher-level wrapper

First: backprop through a recurrent net

```
# Inputs:
#     (In addition to inputs used by L : Number of hidden layers
#     dh_top:  derivatives ddiv/dh_*(t,L) at each time (* may be f or b)
#     h, z:  h and z values returned by the forward pass
#     T: Length of input vector sequence
# Output:
#     dW_c, dW_b, db dh_init: derivatives w.r.t current and recurrent weights,
#                             biases, and initial h.
# Assuming all dz, dh, dW_c, dW_r and db are initialized to 0


function RNN_bptt(L, W_c, W_r, b, hinit, x, T, dh_top, h, z)
    dh = zeros
    for t = T-1:downto:0  # Backward through time
        dh(t,L) += dh_top(t)
        h(t,0) = x(t)
        for l = L:1  # Reverse through layers
            dz(t,l) = dh(t,l)Jacobian(h(t,l),z(t,l))
            dh(t,l-1) += dz(t,l) W_c(l)
            dh(t-1,l) += dz(t,l) W_r(l)

            dW_c(l) += h(t,l-1)dz(t,l)
            dW_r(l) += h(t-1,l)dz(t,l)
            db(l) += dz(t,l)
        dx(t)= dh(t,0)

    return dx, dWc, dWr, db, dh(-1)  # dh(-1) is actually dh(-1,1:L,:)
```

BRNN block: gradient computation

```
# Subscript f represents forward net, b is backward net
# Given dh(t), t=0…T-1 : The sequence of gradients from the upper layer
# Also assumed available:
#      x(t), t=0…T-1 : the input to the BRNN block
#      z_f(t), h_f(t) : Complete forward-computation outputs for all layers of the forward net
#      z_b(t), h_b(t) : Complete backward-computation outputs for all layers of the backward net
# L_f and L_b are the number of components in h_f(t) and h_b(t)


for t = 0:T-1  # Separate out forward and backward net gradients
    dh_f(t) = dh(t,1:L_f)
    dh_b(t) = dh(t,L_f+1:L_f+L_b)


#forward net
[dx_f dW_fc,dW_fr,db_f,dh_f(-1)] = RNN_bptt(L, W_fc, W_fr, b_f, h_f(-1), x, T, dh_f, h_f, z_f)


#backward net
x_rev = fliplr(x)   # Flip it in time
dh_brev = fliplr(dh_b)
h_brev = fliplr(h_b)
z_brev = fliplr(z_b)
[dx_brev, dW_bc,dW_br,db_b,dh_b(inf)] = RNN_bptt(L, W_bc, W_br, b_b, h_b(inf), x_rev, T, dh_brev, h_brev, z_brev)
dx_b = fliplr(dx_brev)


for t = 0:T-1  # Add the partials
    dx(t) = dx_f(t) + dx_b(t)
```

Story so far

- Time series analysis must consider past inputs along with current input

- Recurrent networks look into the infinite past through a state-space framework
  - Hidden states that recurse on themselves

- Training recurrent networks requires
  - Defining a divergence between the actual and desired output *sequences*
  - Backpropagating gradients over the entire chain of recursion
    - Backpropagation through time
  - Pooling gradients with respect to individual parameters over time

- Bidirectional networks analyze data both ways, begin→end and end→beginning to make predictions
  - In these networks, backprop must follow the chain of recursion (and gradient pooling) separately in the forward and reverse nets

RNNs..

- Excellent models for series data analysis tasks
  - Time-series prediction
  - Time-series classification
  - Sequence generation..

So how did this happen

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servicious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS)[http://www.humah.yahoo.com/guardian. cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile.]]

So how did this happen

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servicious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS)[http://www.humah.yahoo.com/guardian. cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile.]]

More on this later..

RNNs..

- Excellent models for series data analysis tasks
  - Time-series prediction
  - Time-series classification
  - Sequence generation..
  - They can even simplify some problems that are difficult for MLPs
    - Next class..