

Deep Learning
Recurrent Networks : 1
Spring 2019

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 \mathcal{C} be a set of the construction.*

Let \mathcal{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 $\mathcal{F} \rightarrow \mathcal{F}$ 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

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

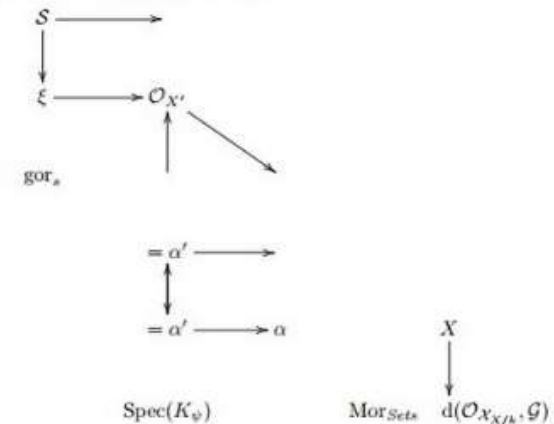
be a morphism of algebraic spaces over S and Y .

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 $X = \text{Spec}(R)$ 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

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_x \rightarrow \mathcal{O}_{X_{\acute{e}tale}}^{-1} \rightarrow \mathcal{O}_{X_t}^{-1} \mathcal{O}_{X_\lambda}(\mathcal{O}_{X_n}^v)$$

is an isomorphism of covering of \mathcal{O}_{X_t} . 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_λ} 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..

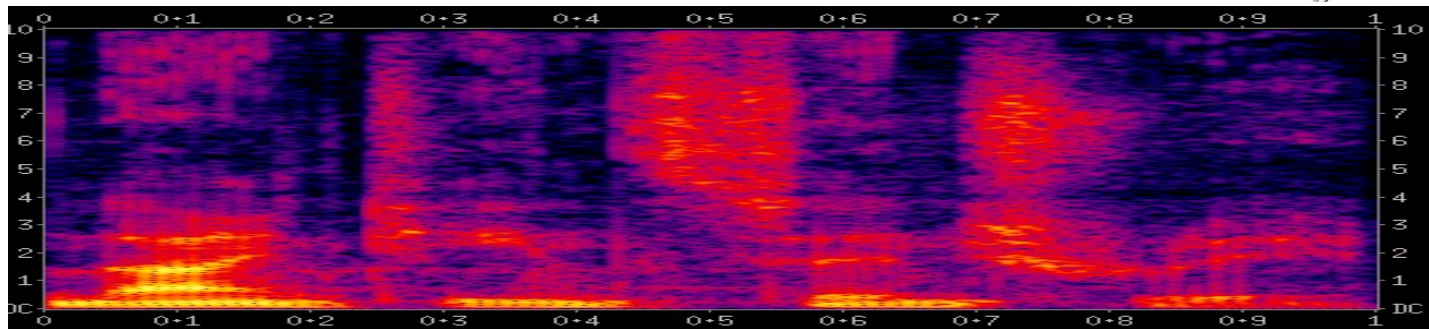
- All previous examples were *generated* blindly by a *recurrent* neural network..
- <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

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 vectors. Output is a classification result

What is he talking about?

“Football” or “basketball”?

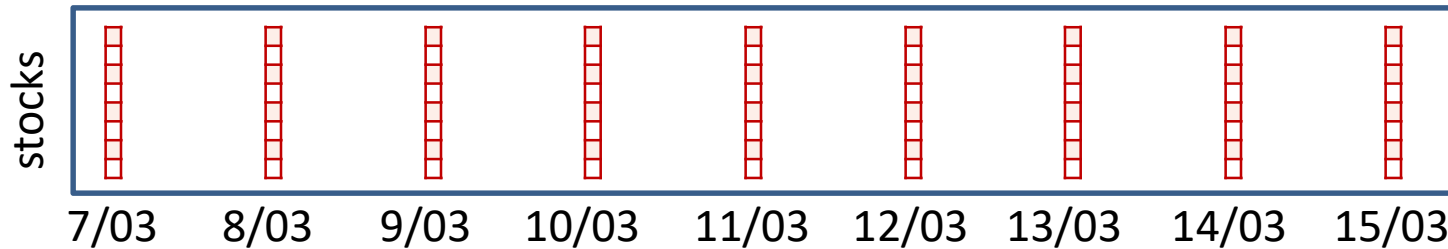


The Steelers, meanwhile, continue to struggle to make stops on defense. They've allowed, on average, 30 points a game, and have shown no signs of improving anytime soon.

- 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

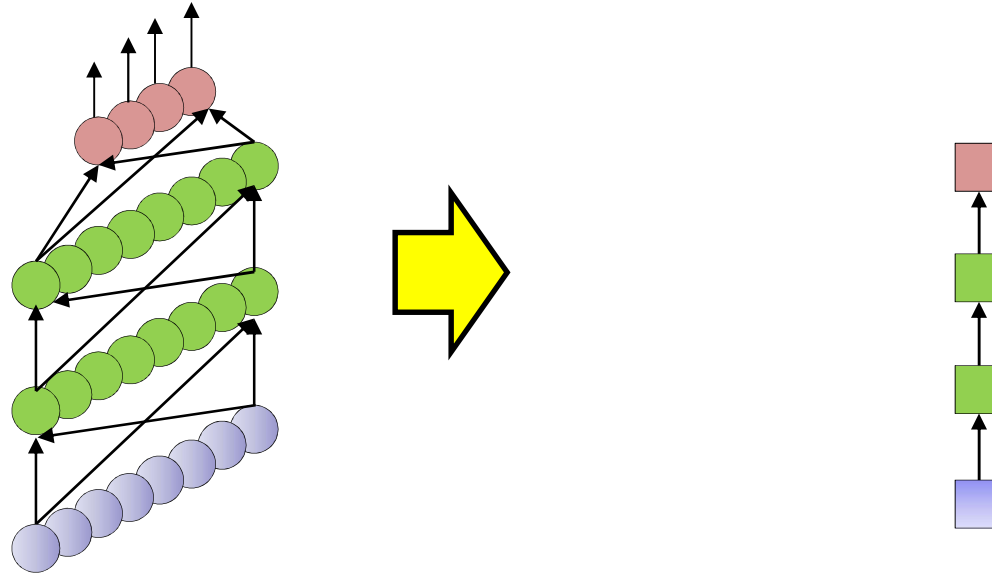
Should I invest..

To invest or not to invest?



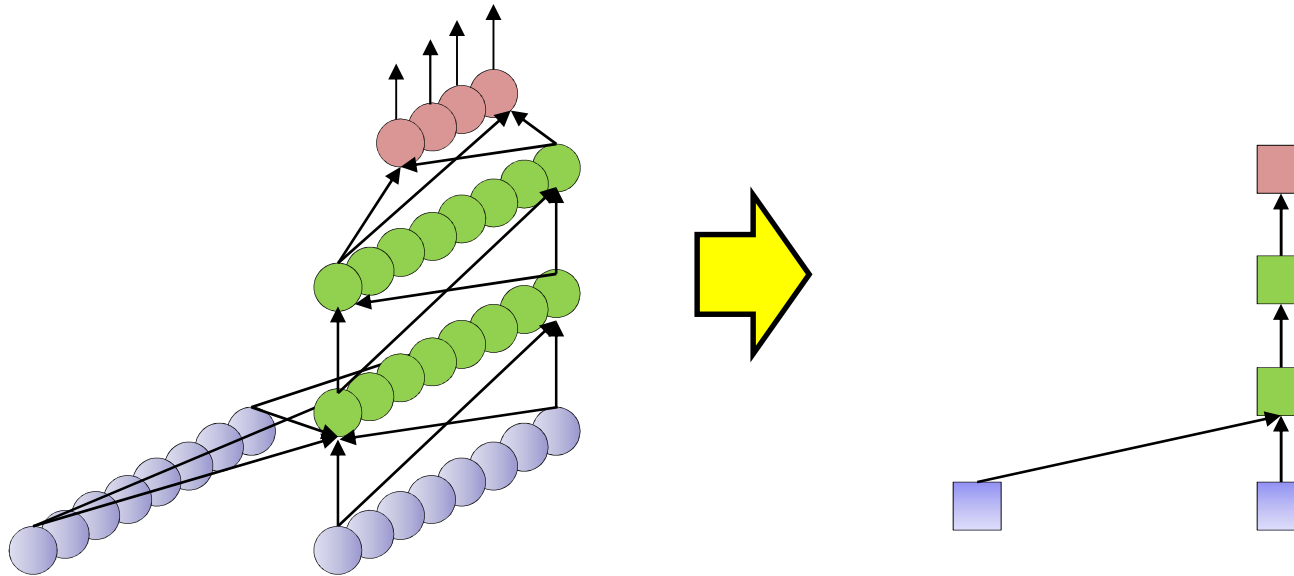
- Stock market
 - Must consider the series of stock values in the past several days to decide if it is wise to invest today
 - Ideally consider *all* of history
- Note: Inputs are vectors. Output may be scalar or vector
 - Should I invest, vs. should I invest in X

Representational shortcut



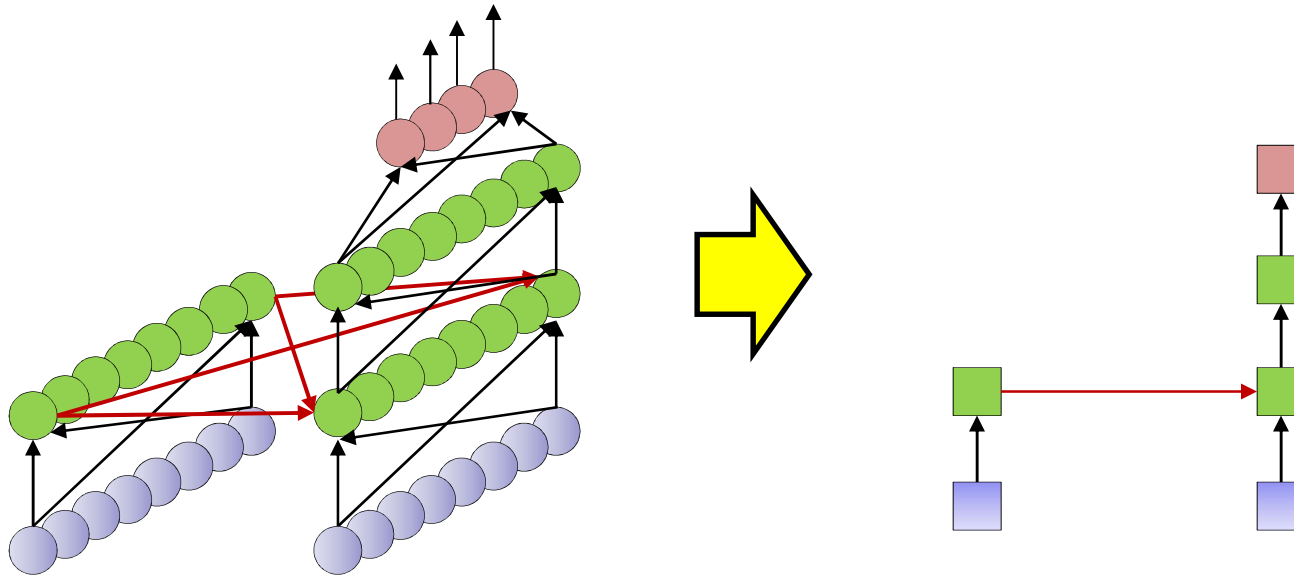
- Input at each time is a *vector*
- Each layer has many neurons
 - Output layer too may have many neurons
- But will represent everything by simple boxes
 - Each box actually represents an entire *layer with many units*

Representational shortcut



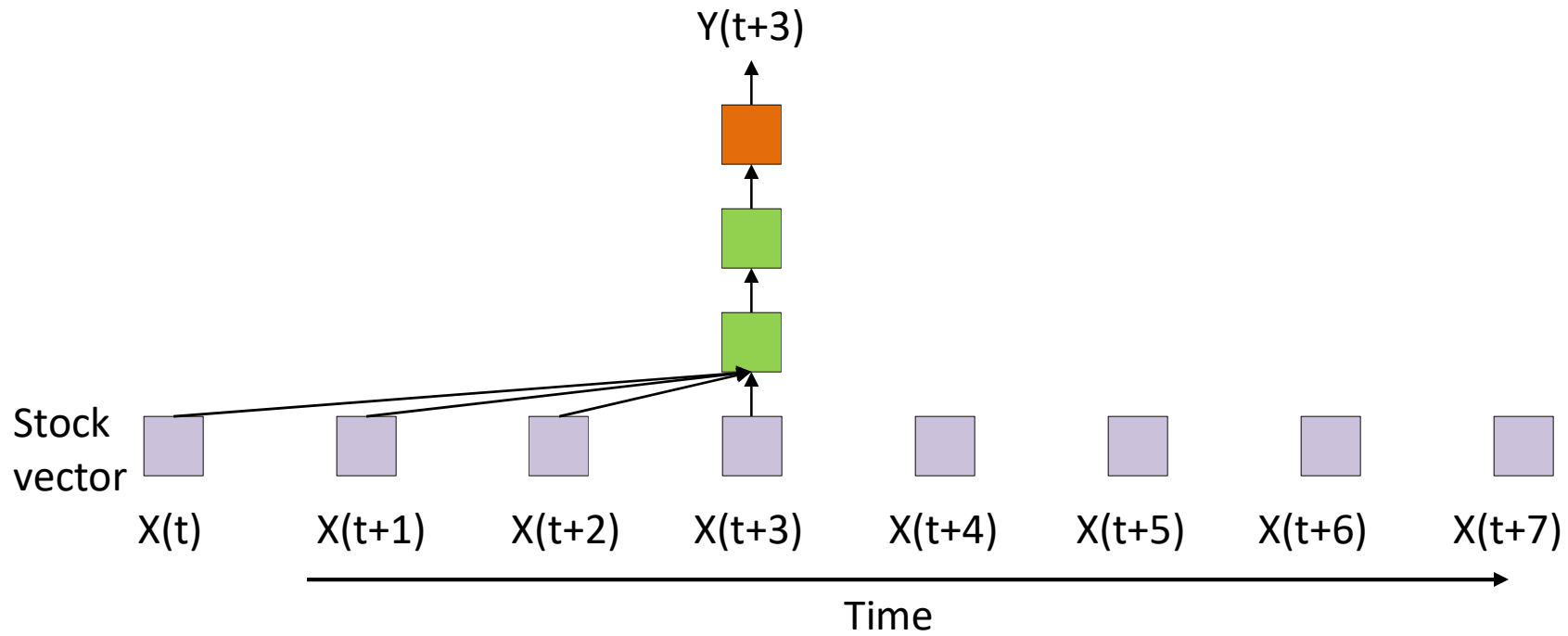
- Input at each time is a *vector*
- Each layer has many neurons
 - Output layer too may have many neurons
- But will represent everything by simple boxes
 - Each box actually represents an entire *layer with many units*

Representational shortcut



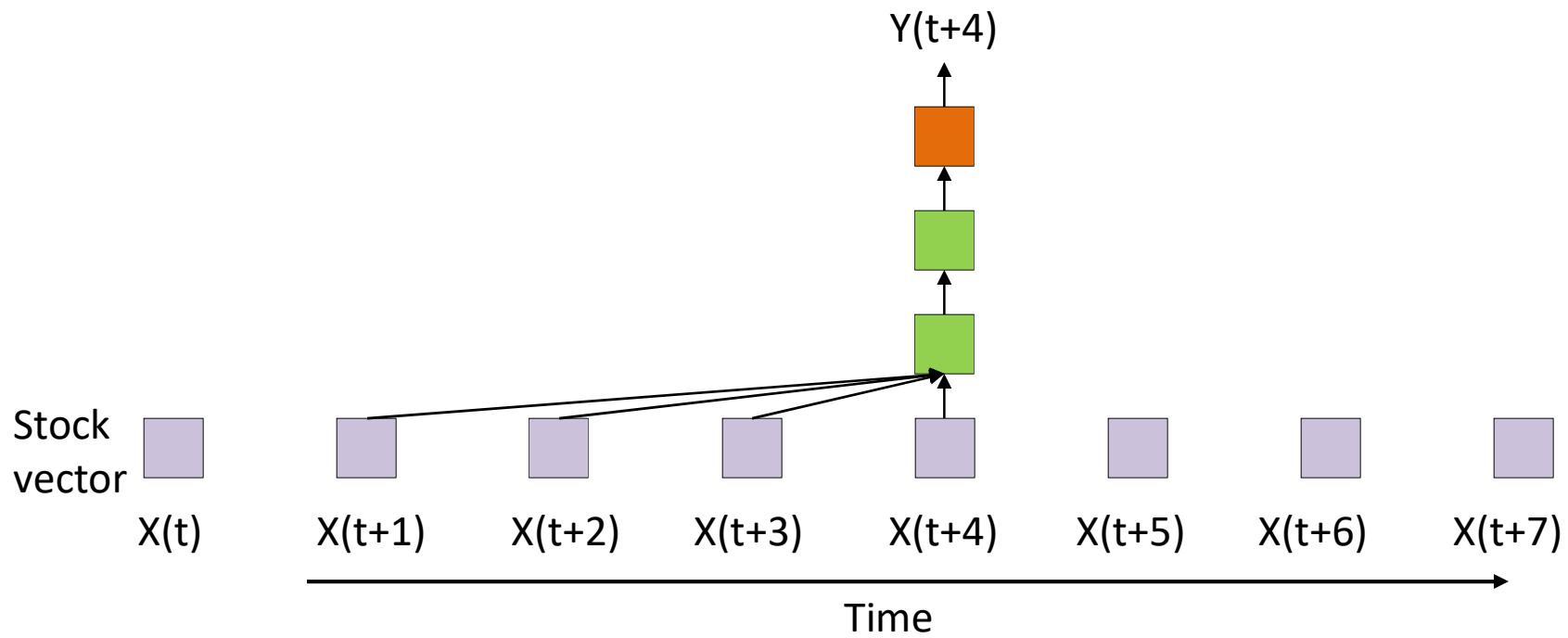
- Input at each time is a *vector*
- Each layer has many neurons
 - Output layer too may have many neurons
- But will represent everything simple boxes
 - Each box actually represents an entire *layer with many units*

The stock predictor



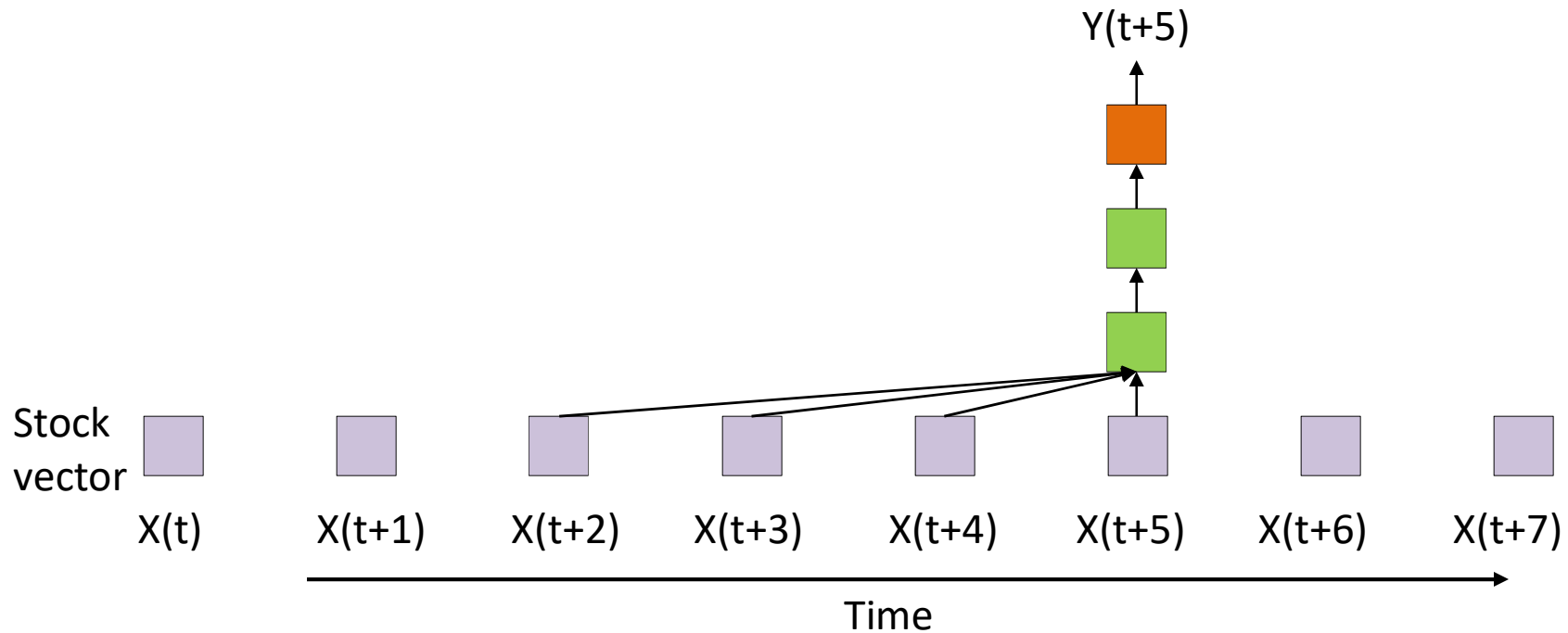
- The sliding predictor
 - Look at the last few days
 - This is just a convolutional neural net applied to series data
 - Also called a *Time-Delay neural network*

The stock predictor



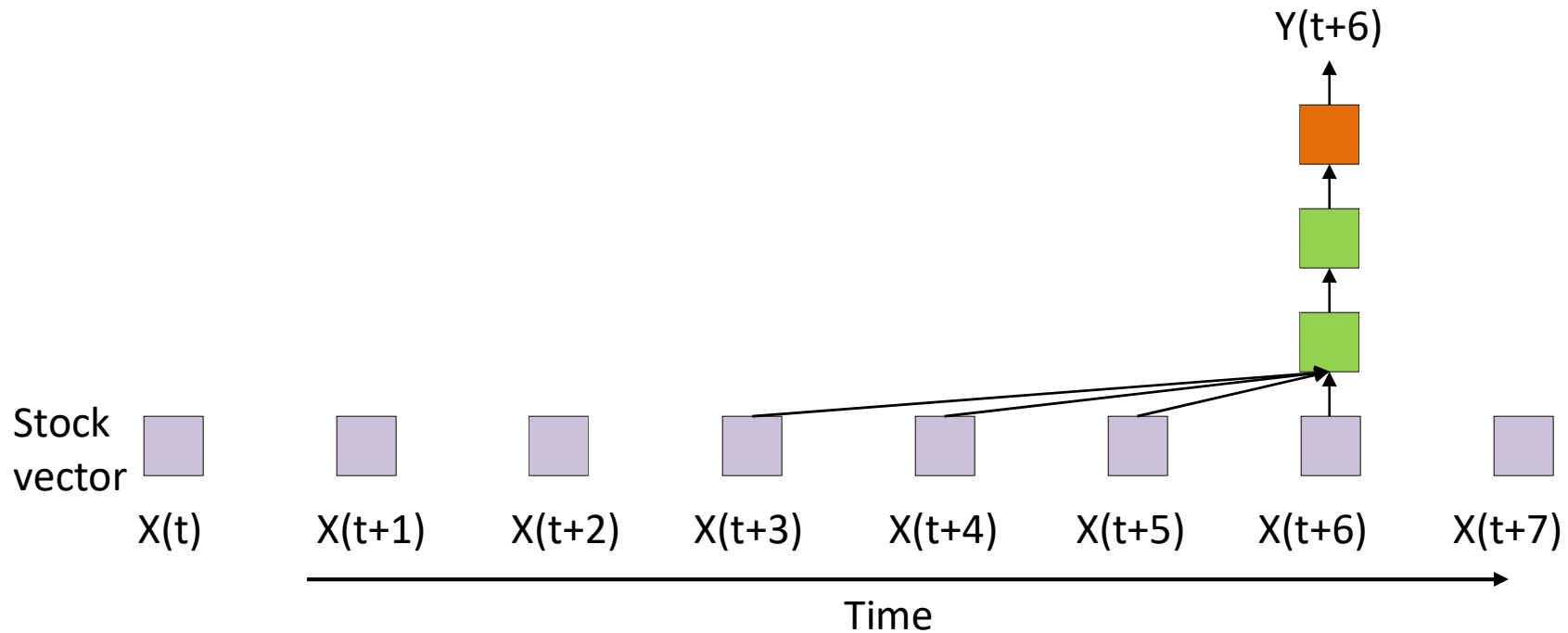
- The sliding predictor
 - Look at the last few days
 - This is just a convolutional neural net applied to series data
 - Also called a *Time-Delay neural network*

The stock predictor



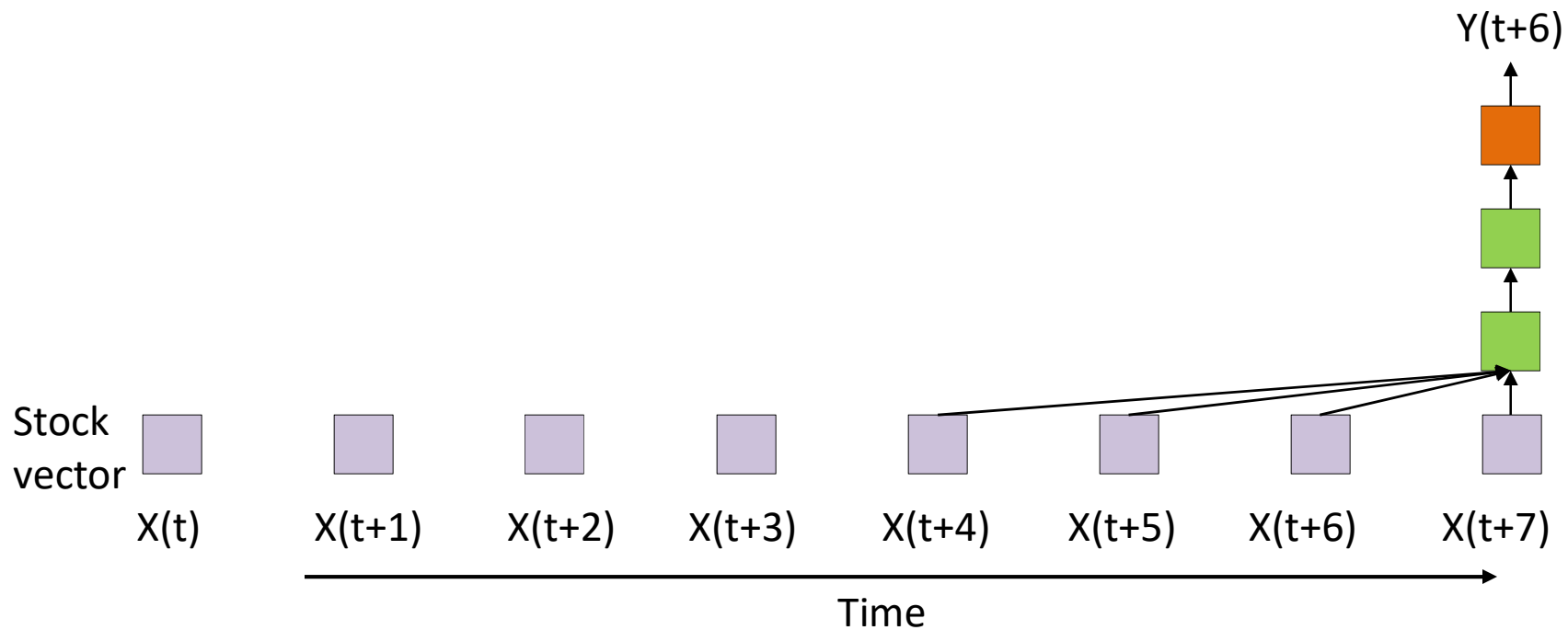
- The sliding predictor
 - Look at the last few days
 - This is just a convolutional neural net applied to series data
 - Also called a *Time-Delay neural network*

The stock predictor



- 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



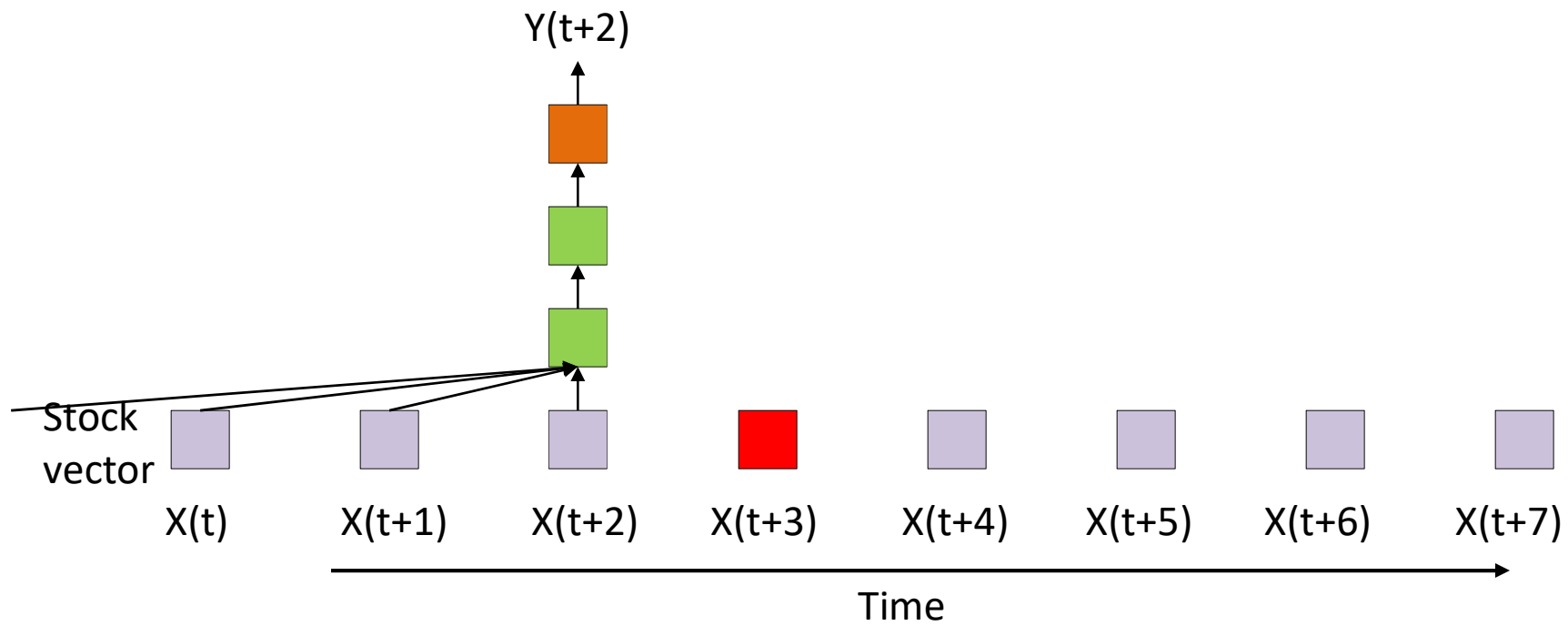
- The sliding predictor
 - Look at the last few days
 - This is just a convolutional neural net applied to series data
 - Also called a *Time-Delay neural network*

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

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

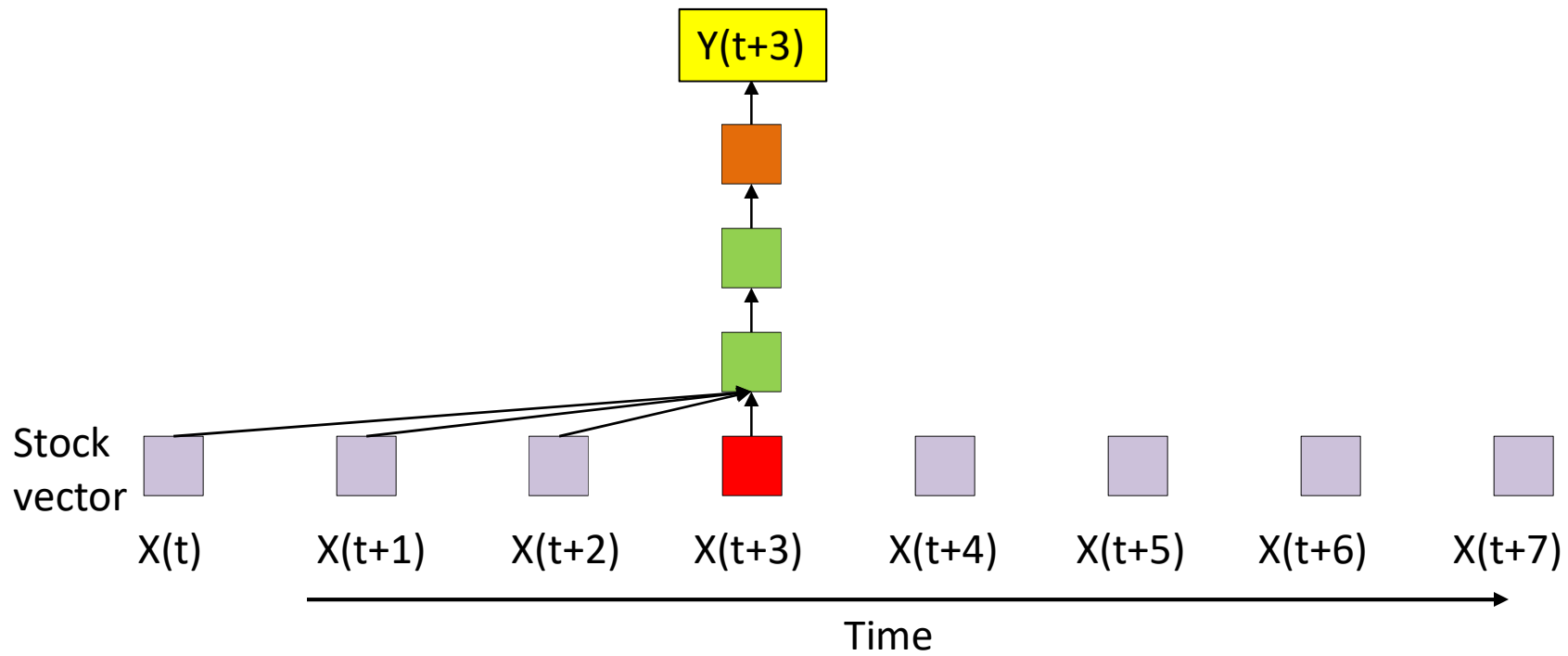
The stock predictor



- 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

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

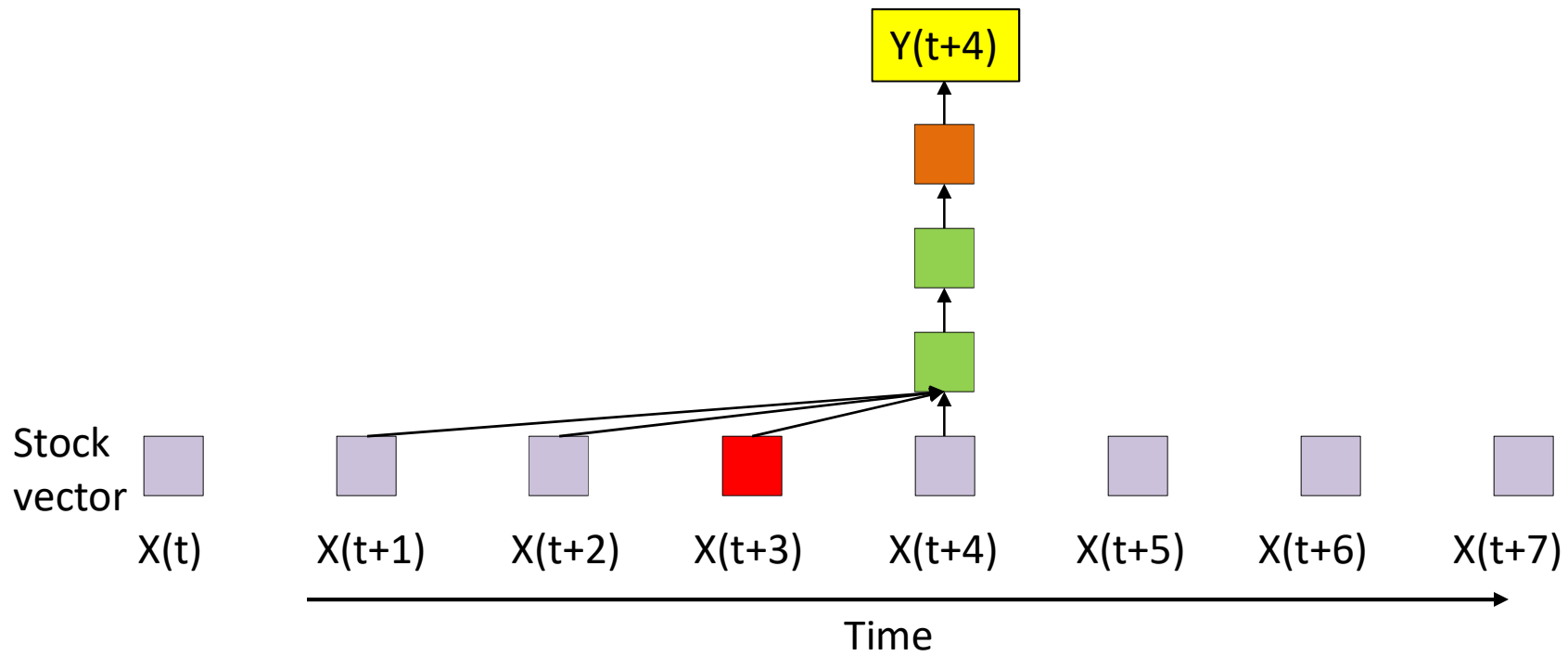
The stock predictor



- 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

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

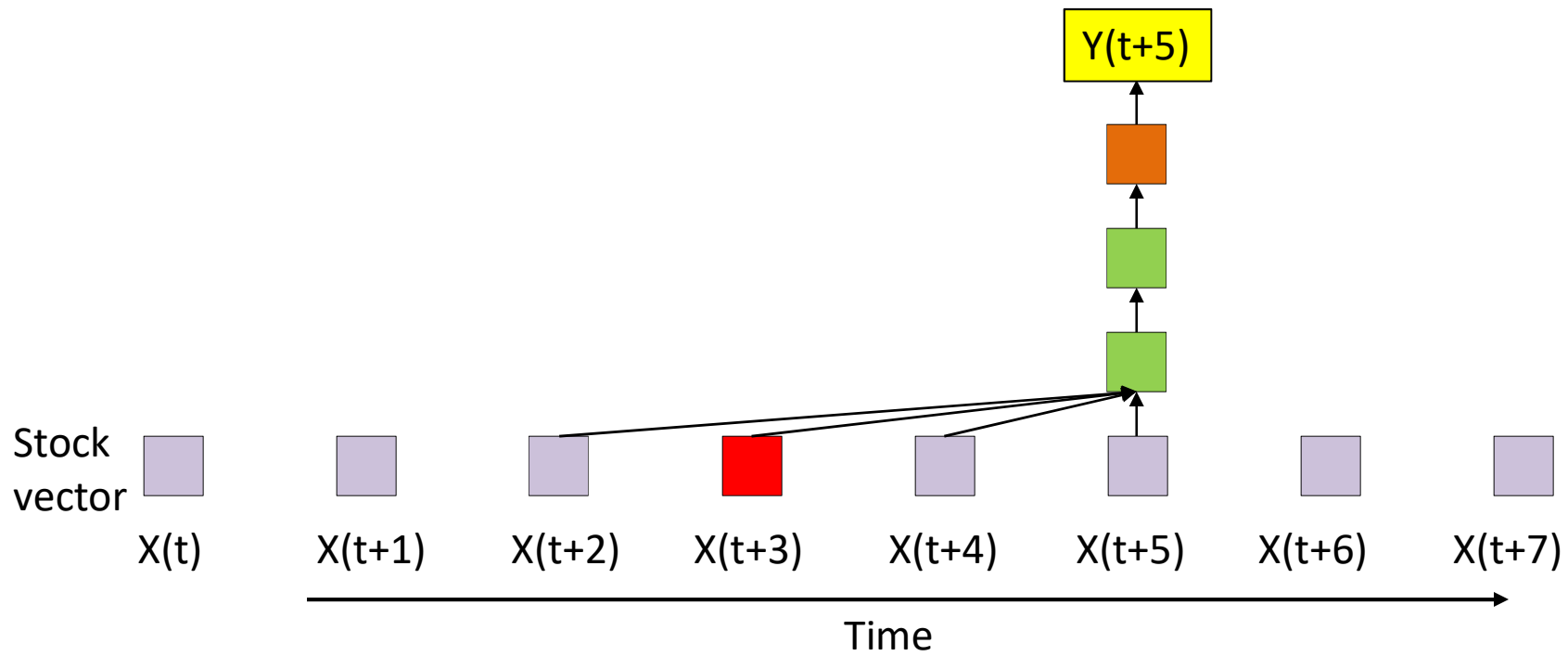
The stock predictor



- 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

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

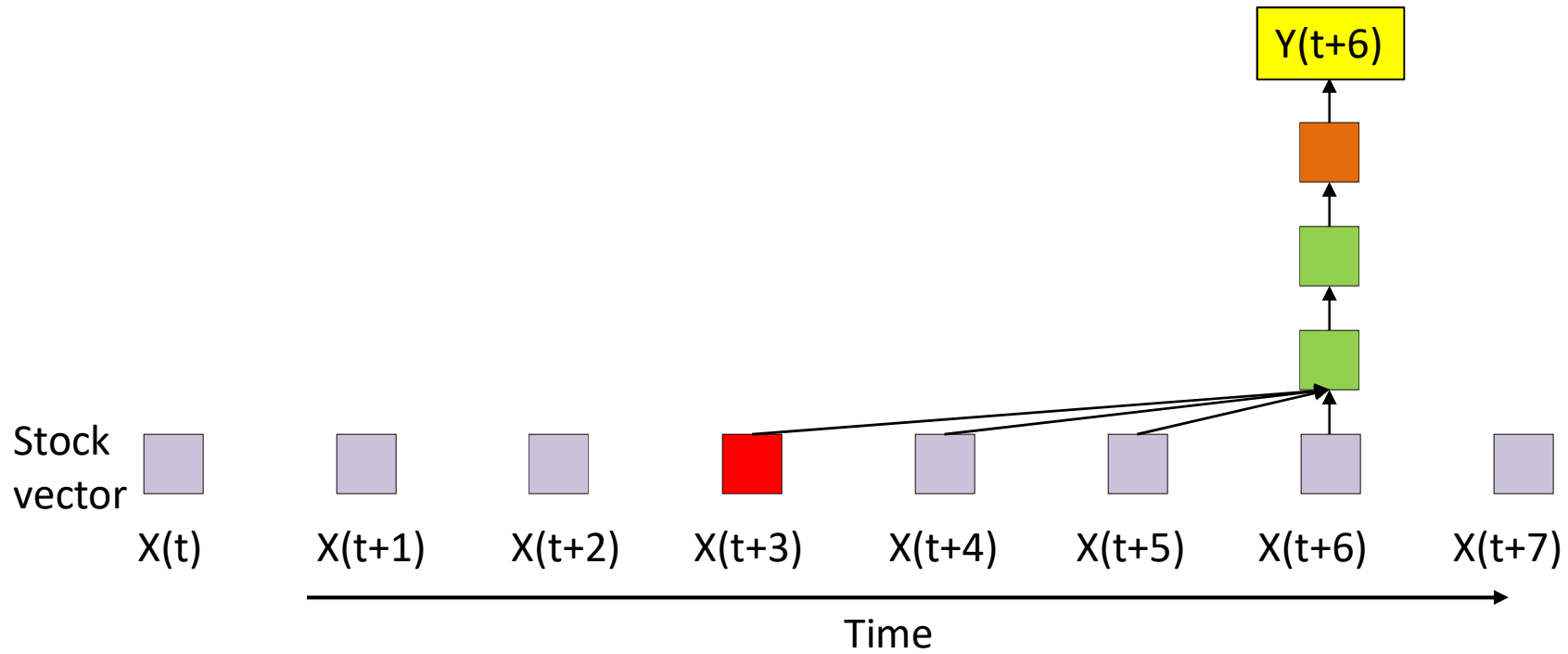
The stock predictor



- 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

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

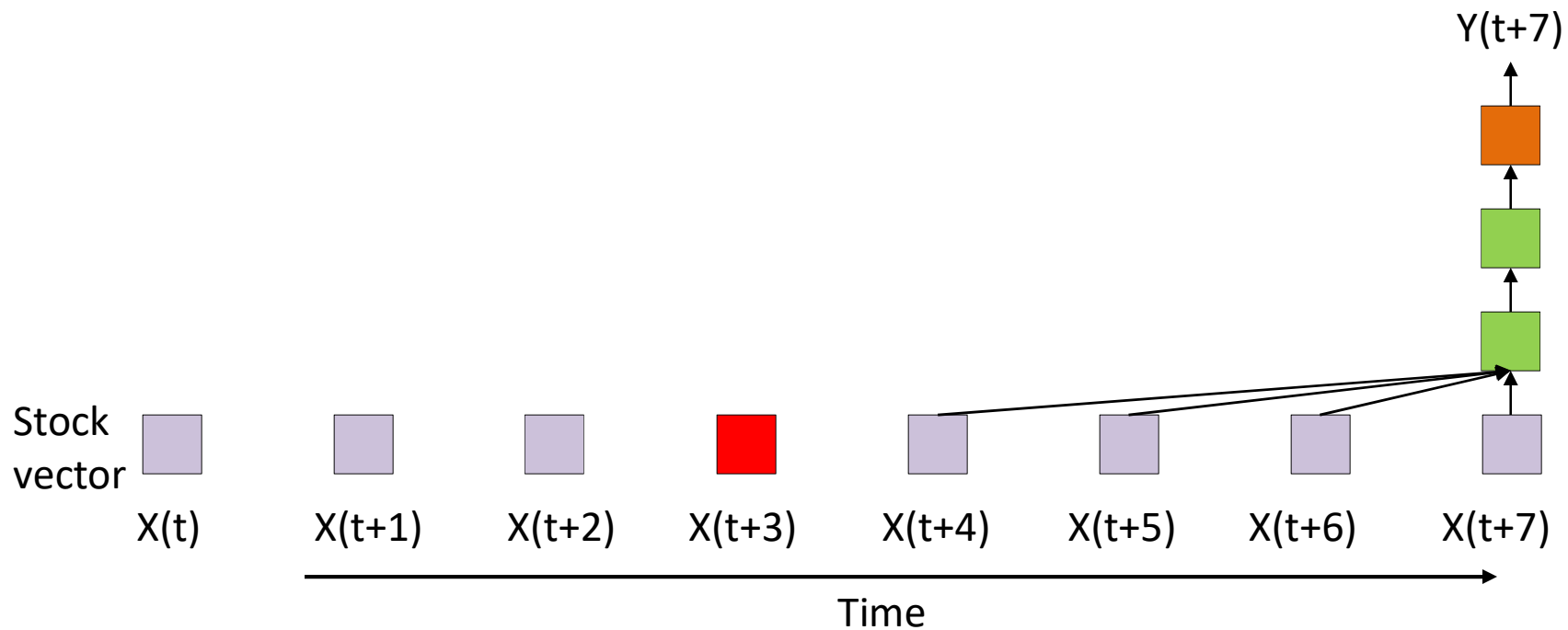
The stock predictor



- 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

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

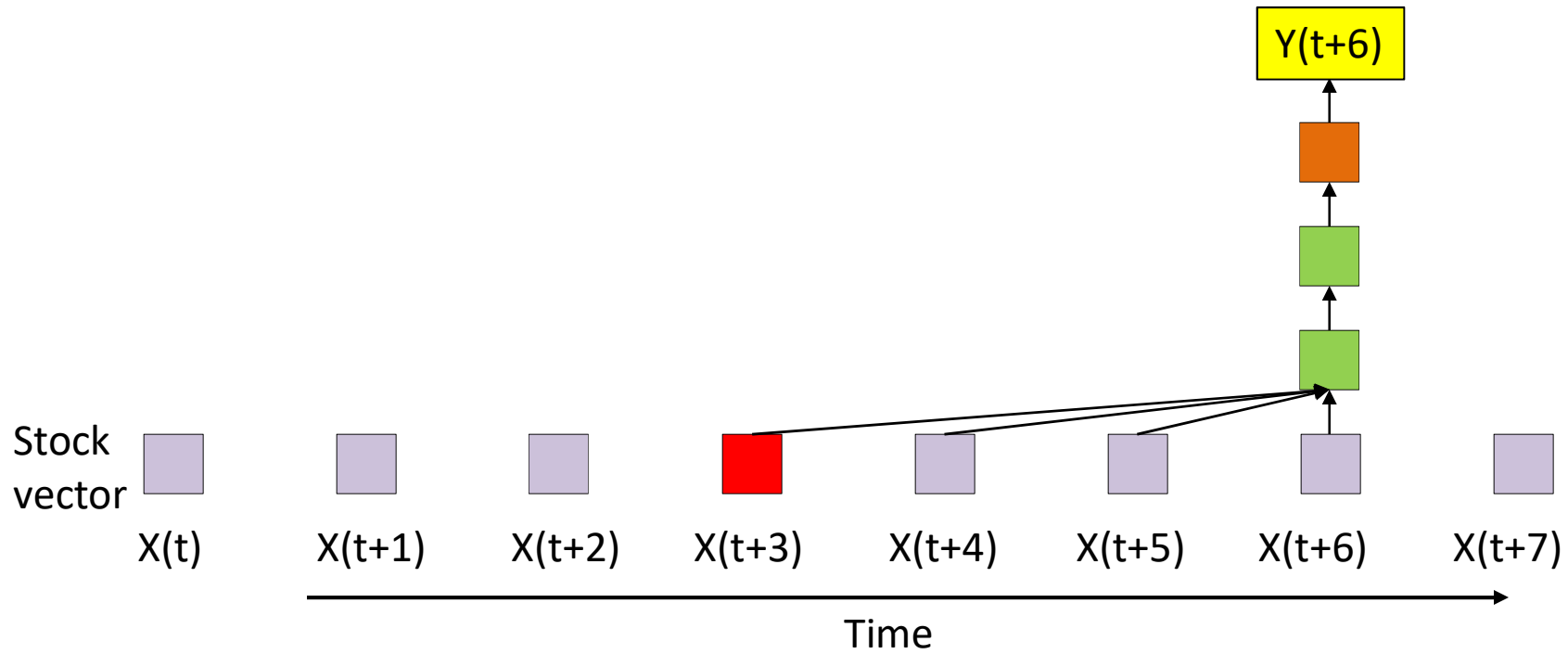
The stock predictor



- 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

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

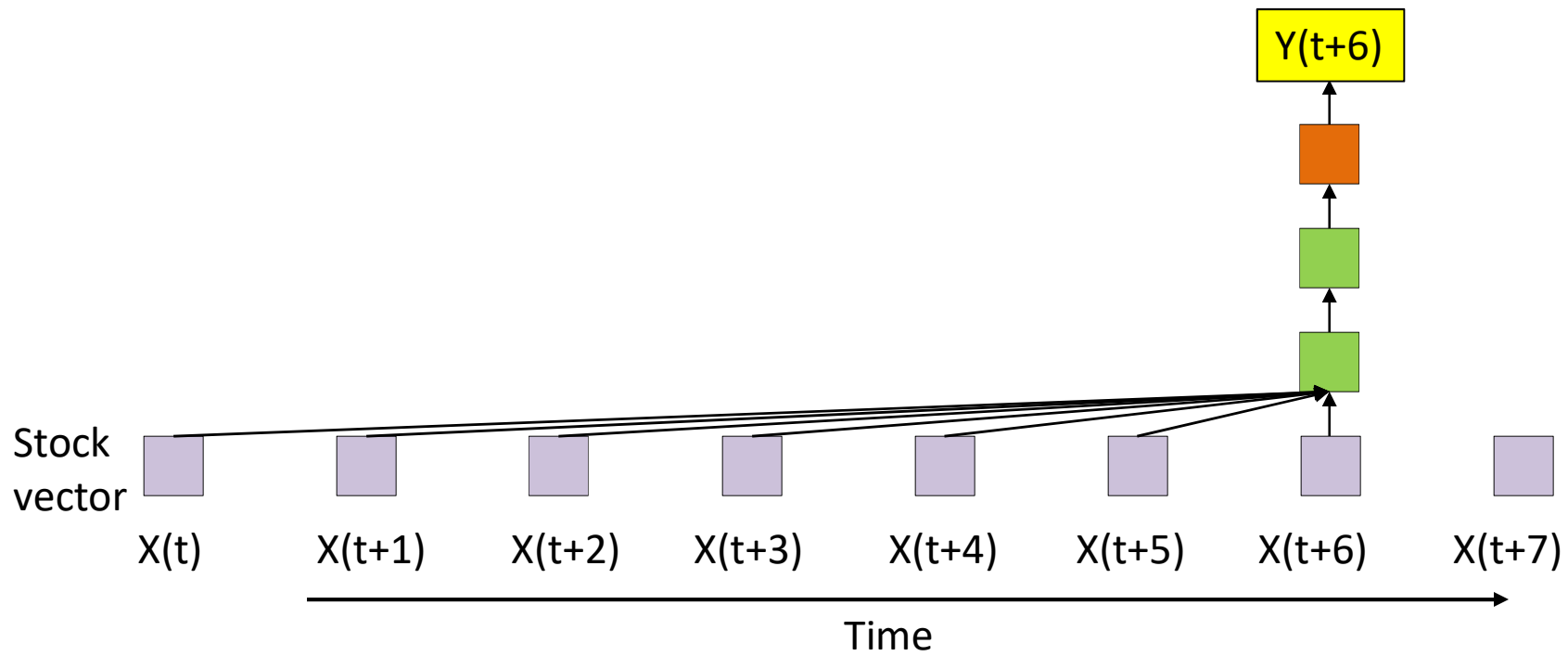
Finite-response model



- 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

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

Finite-response



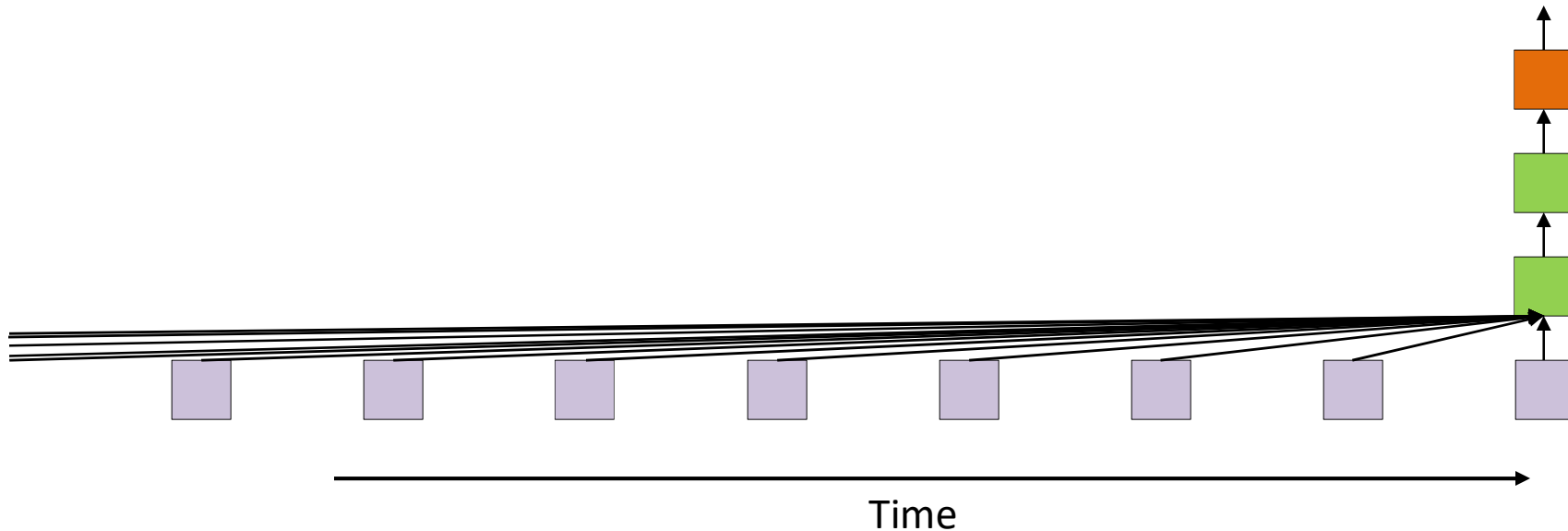
- Problem: Increasing the “history” makes the network more complex
 - No worries, we have the CPU and memory
 - Or do we?

Systems often have long-term dependencies



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

We want *infinite* memory



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

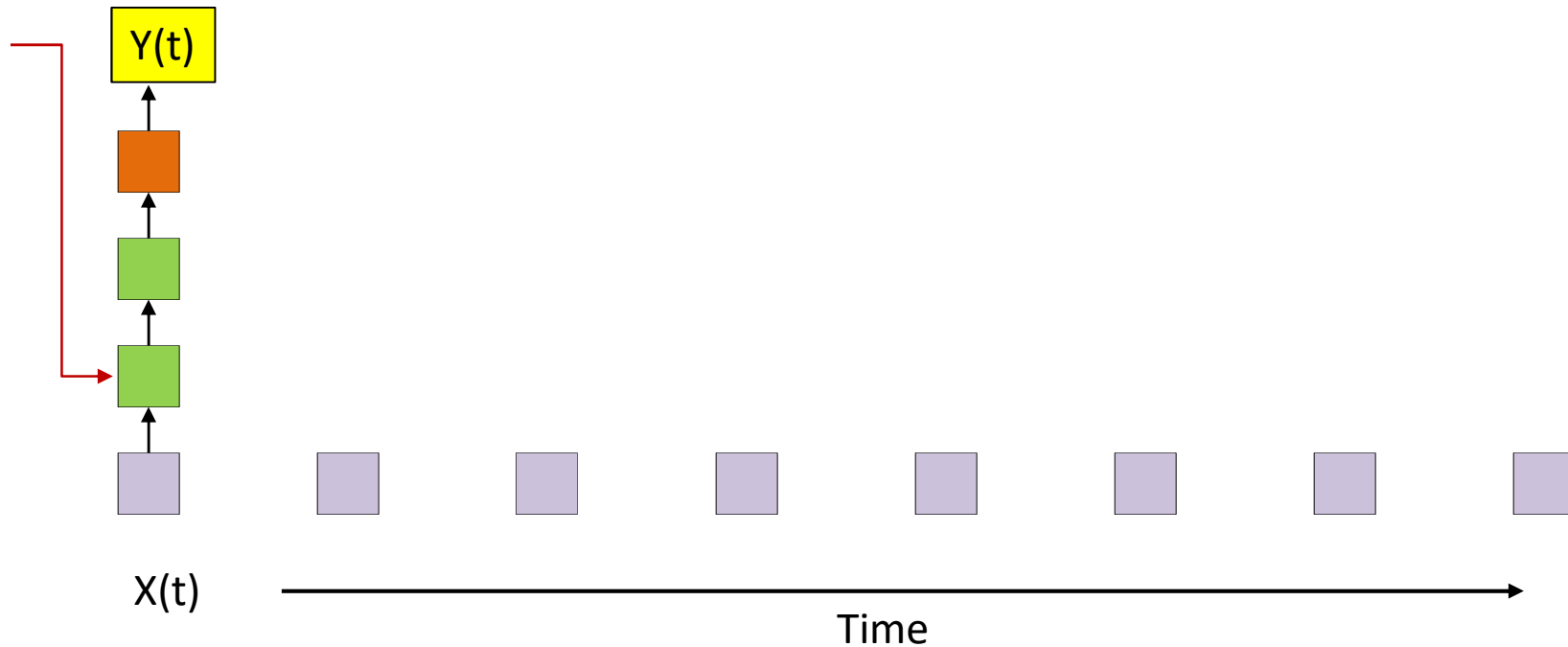
$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-\infty})$$

Examples of infinite response systems

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

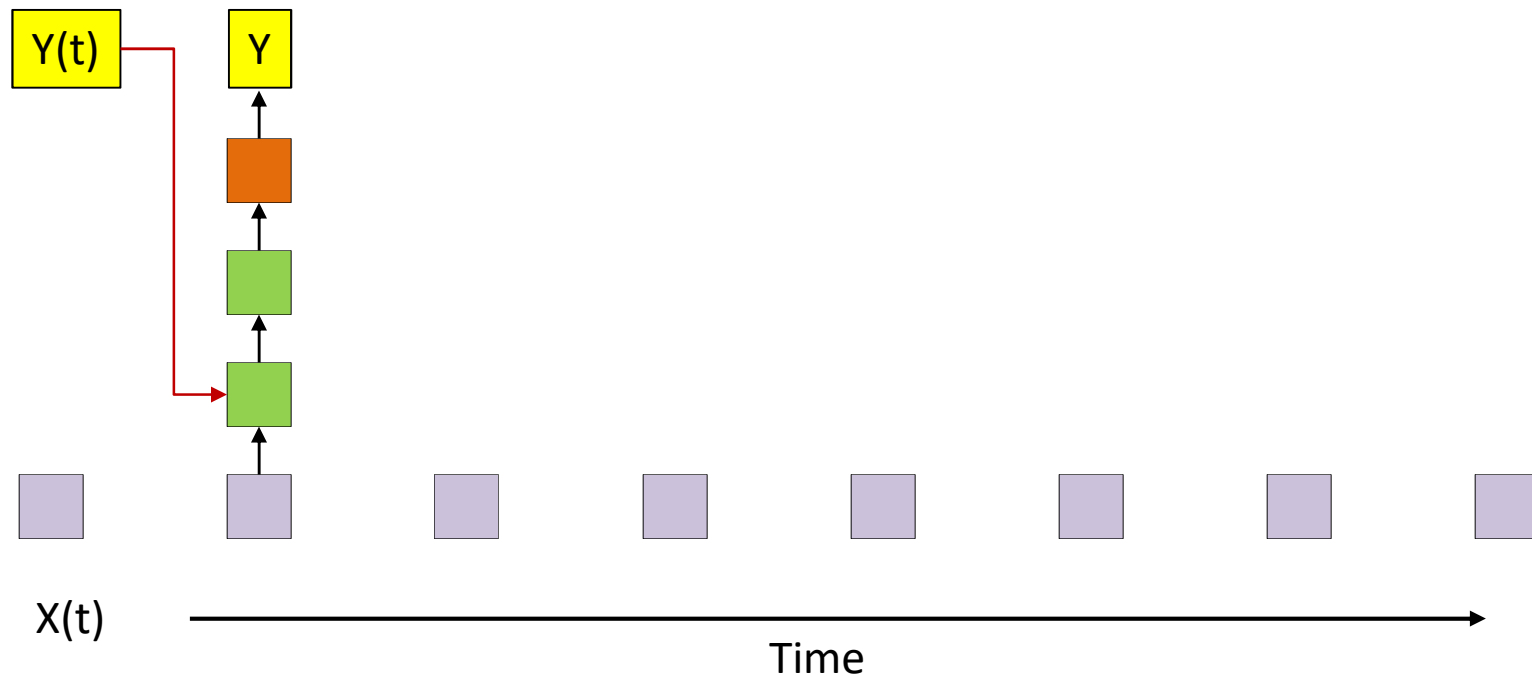
- Required: Define initial state: Y_{-1} for $t = 0$
- An input at X_0 at $t = 0$ produces Y_0
- Y_0 produces Y_1 which produces Y_2 and so on until Y_∞ even if $X_1 \dots X_\infty$ are 0
 - i.e. even if there are no further inputs!
- This is an instance of a NARX network
 - “nonlinear autoregressive 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



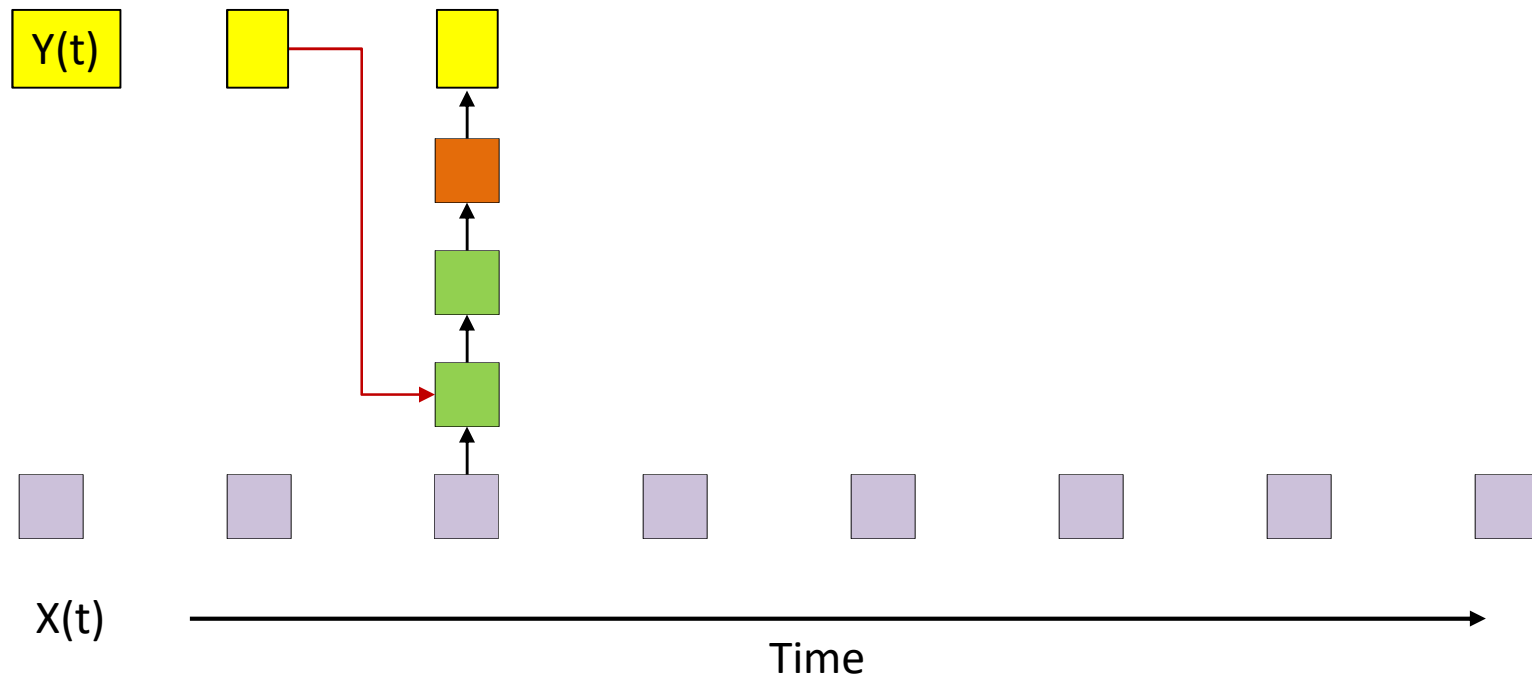
- A NARX net with recursion from the output

A one-tap NARX network



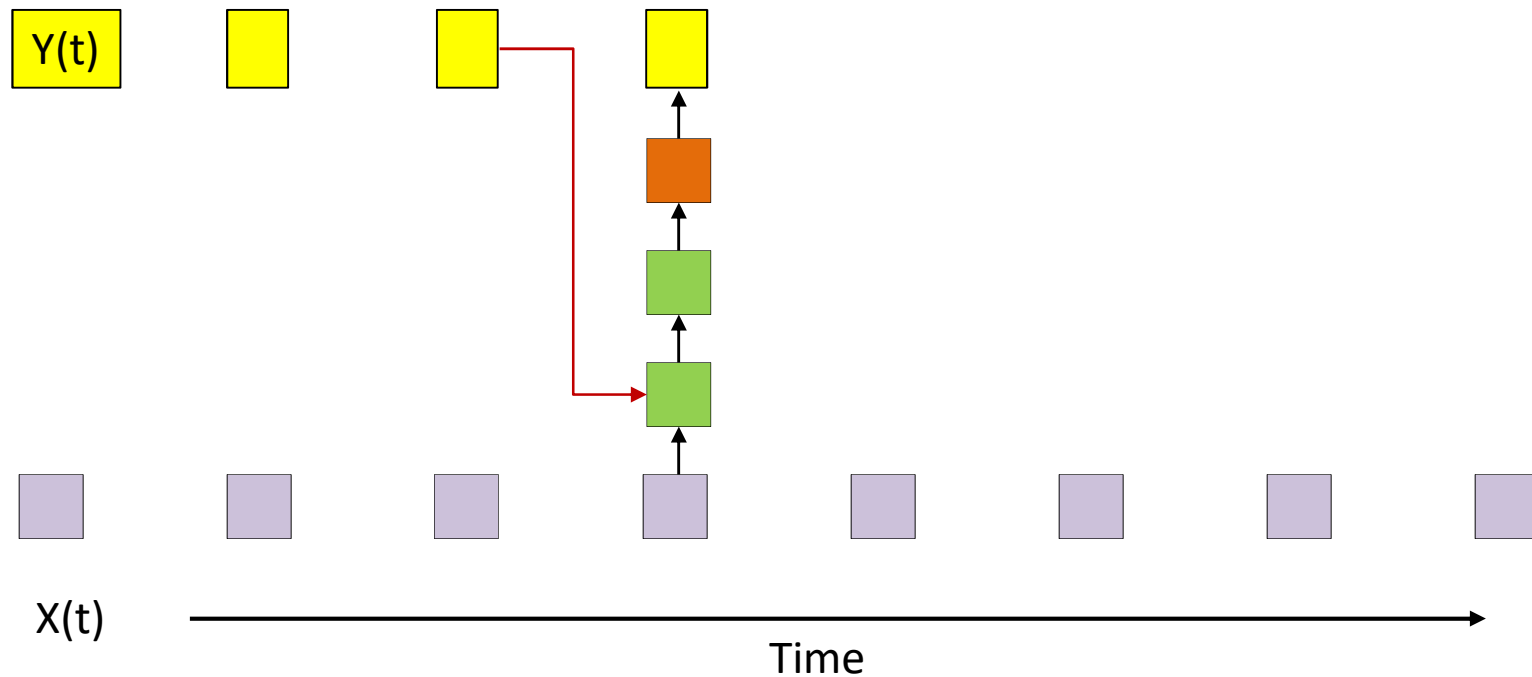
- A NARX net with recursion from the output

A one-tap NARX network



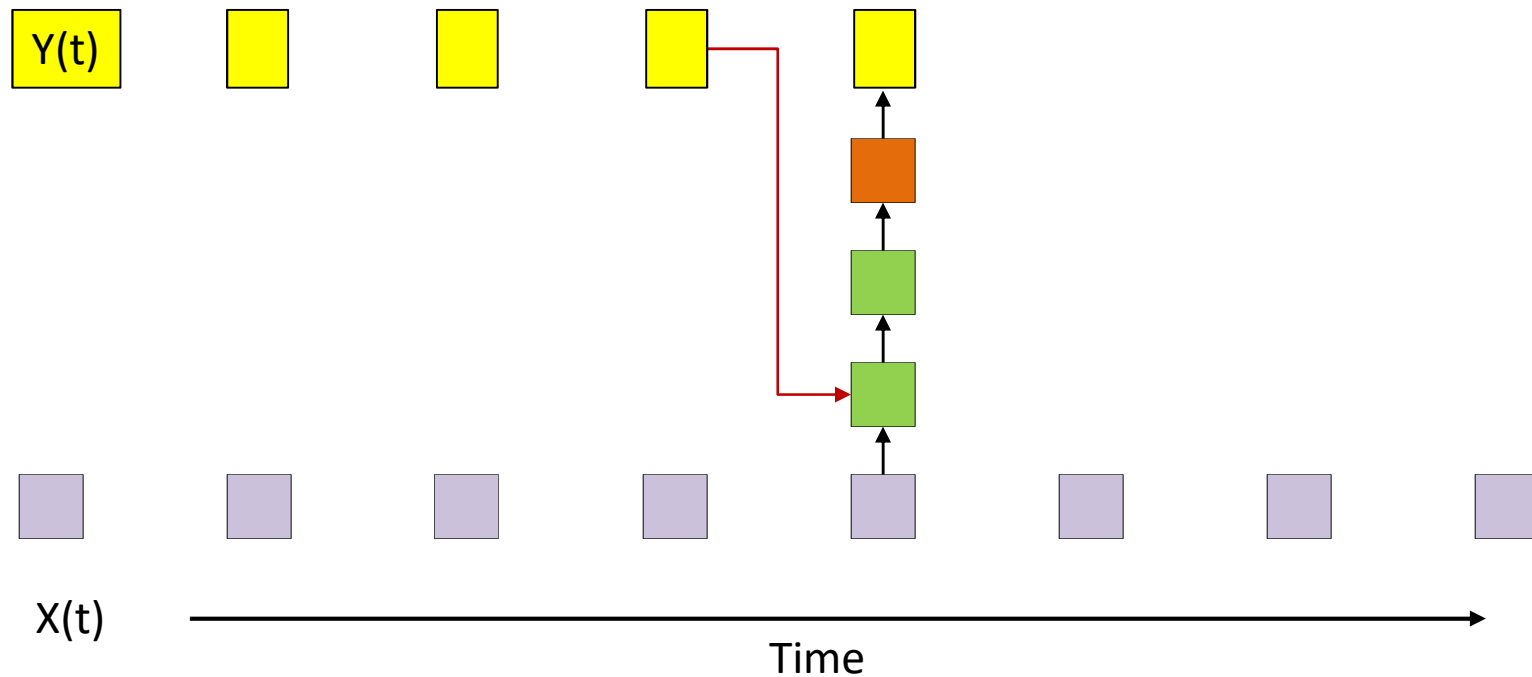
- A NARX net with recursion from the output

A one-tap NARX network



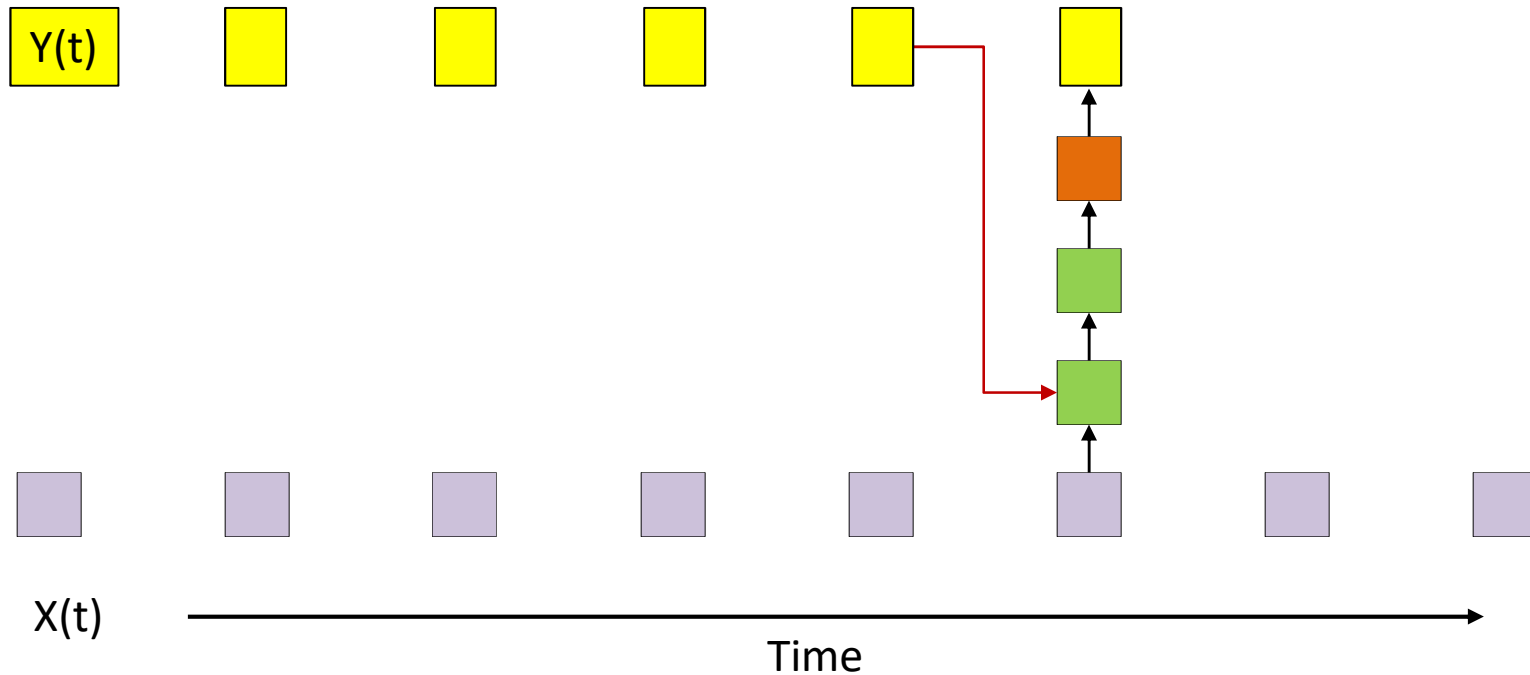
- A NARX net with recursion from the output

A one-tap NARX network



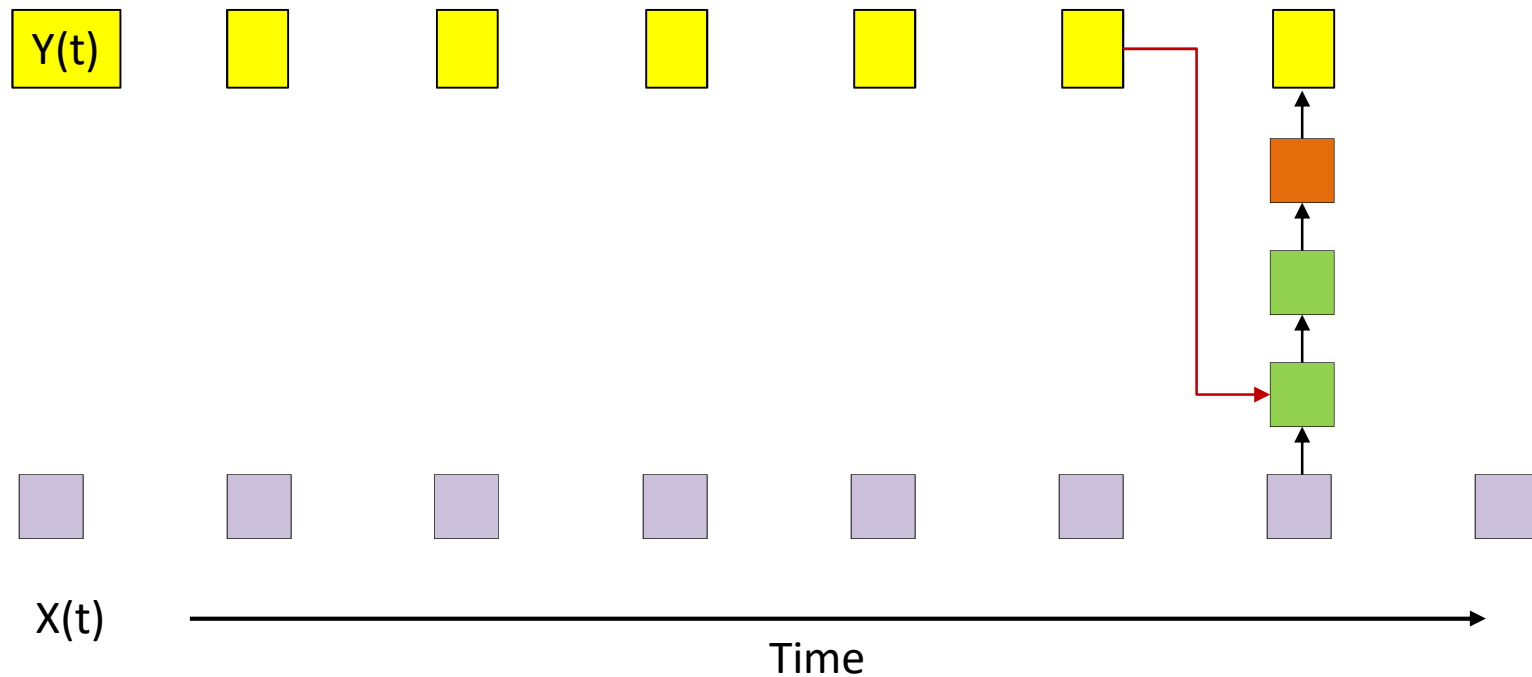
- A NARX net with recursion from the output

A one-tap NARX network



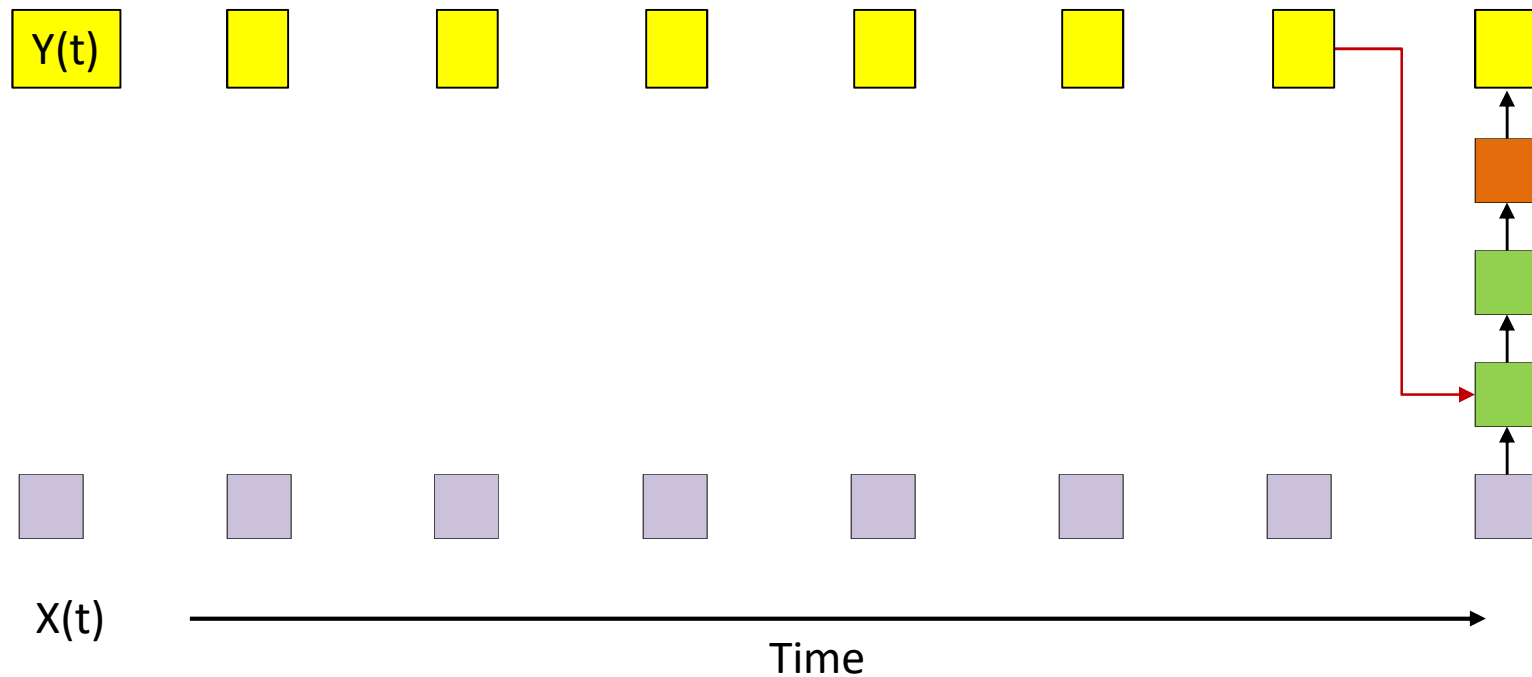
- A NARX net with recursion from the output

A one-tap NARX network



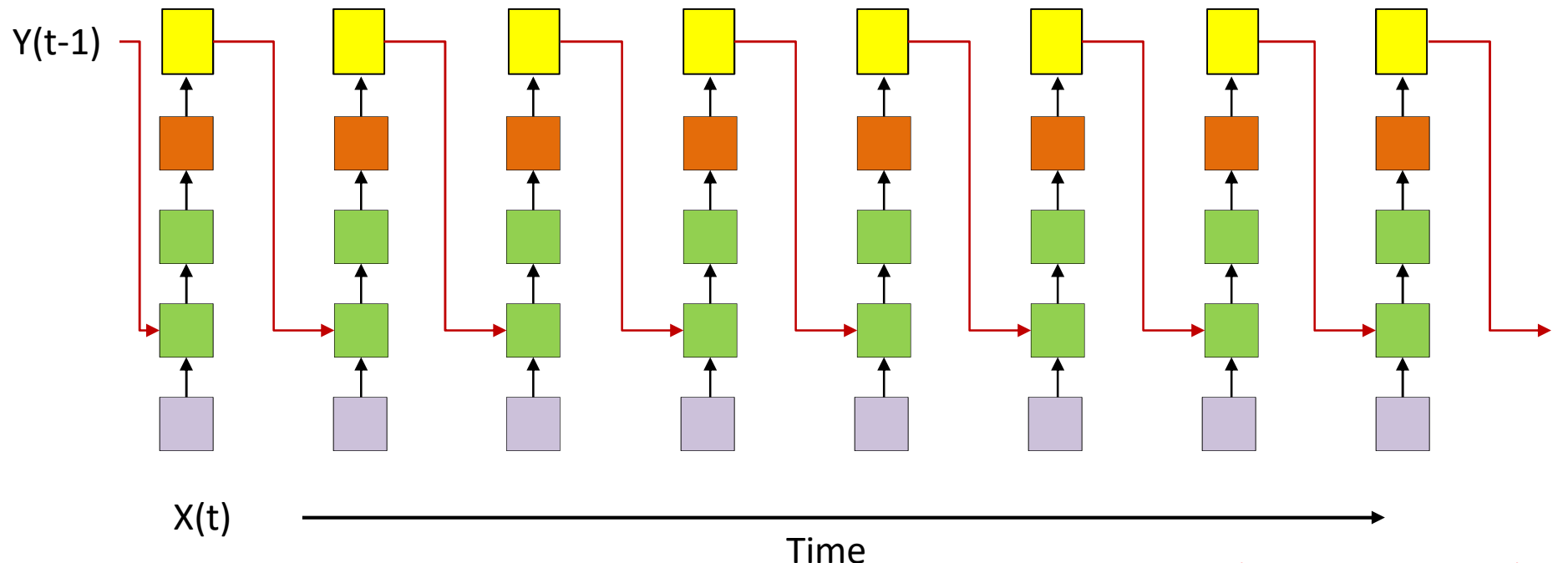
- A NARX net with recursion from the output

A one-tap NARX network



- A NARX net with recursion from the output

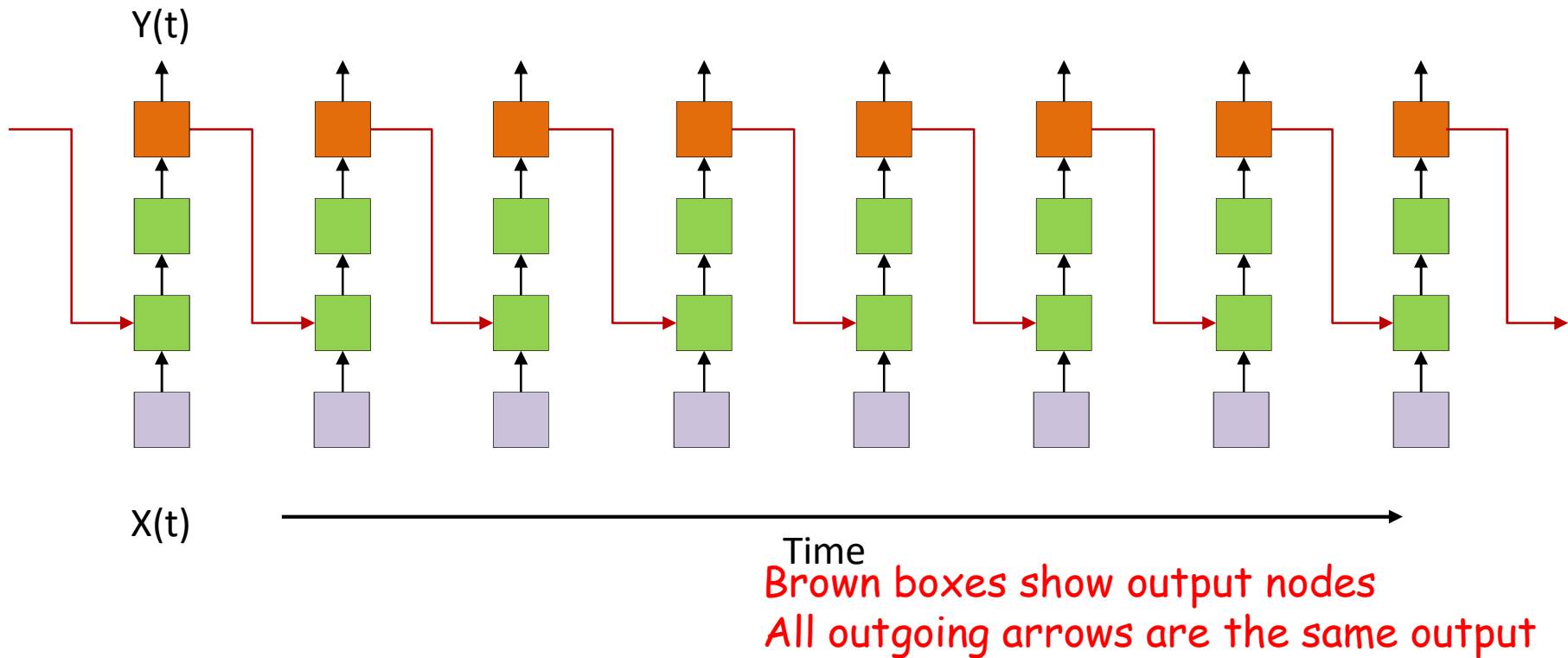
A more complete representation



Brown boxes show output nodes
Yellow boxes are outputs

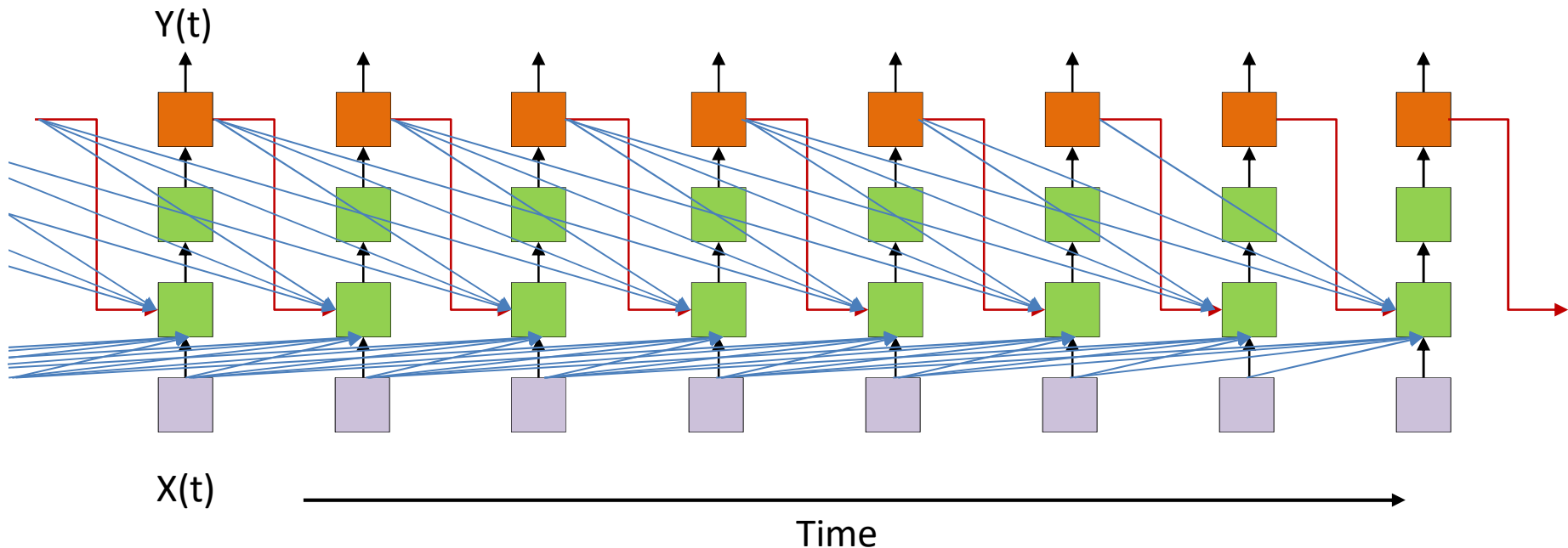
- A NARX net with recursion from the output
- Showing all computations
- All columns are identical
- *An input at $t=0$ affects outputs forever*

Same figure redrawn



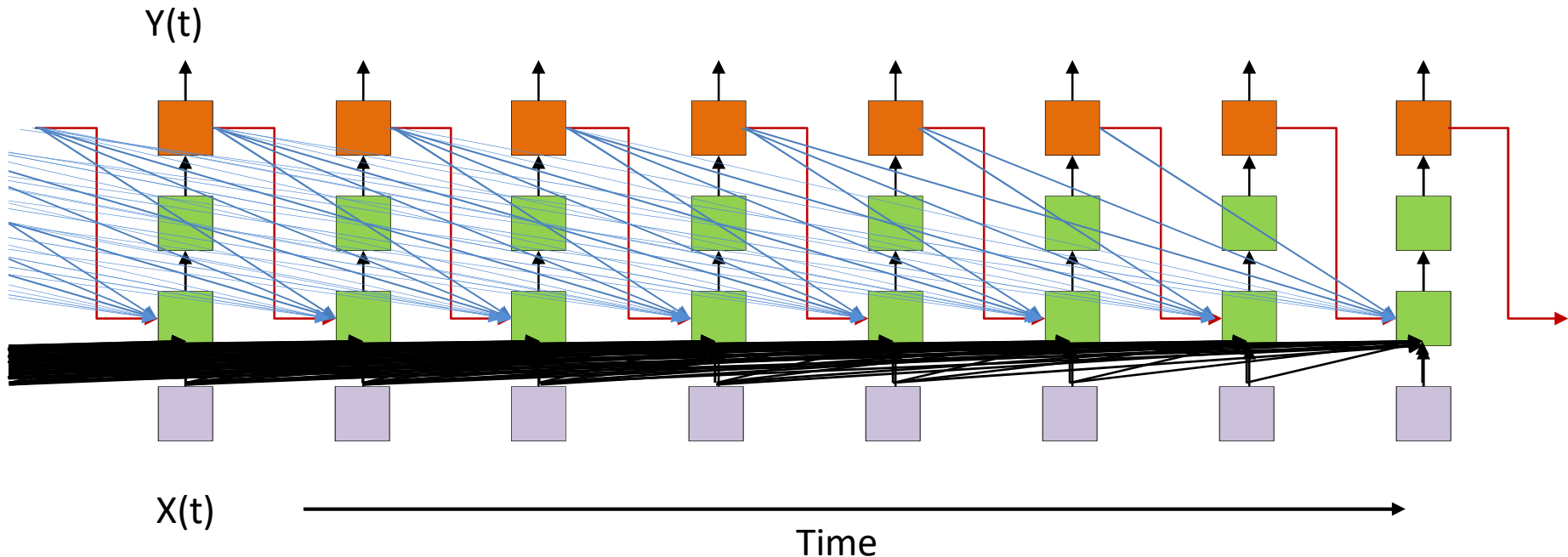
- A NARX net with recursion from the output
- Showing all computations
- All columns are identical
- *An input at $t=0$ affects outputs forever*

A more generic NARX network



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

A “complete” NARX network



- The output Y_t at time t is computed from *all* past outputs and *all* inputs until time t
 - Not really a practical model

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

Lets make memory more explicit

- Task is to “remember” the past
- Introduce an explicit *memory* variable whose *job* it is to remember

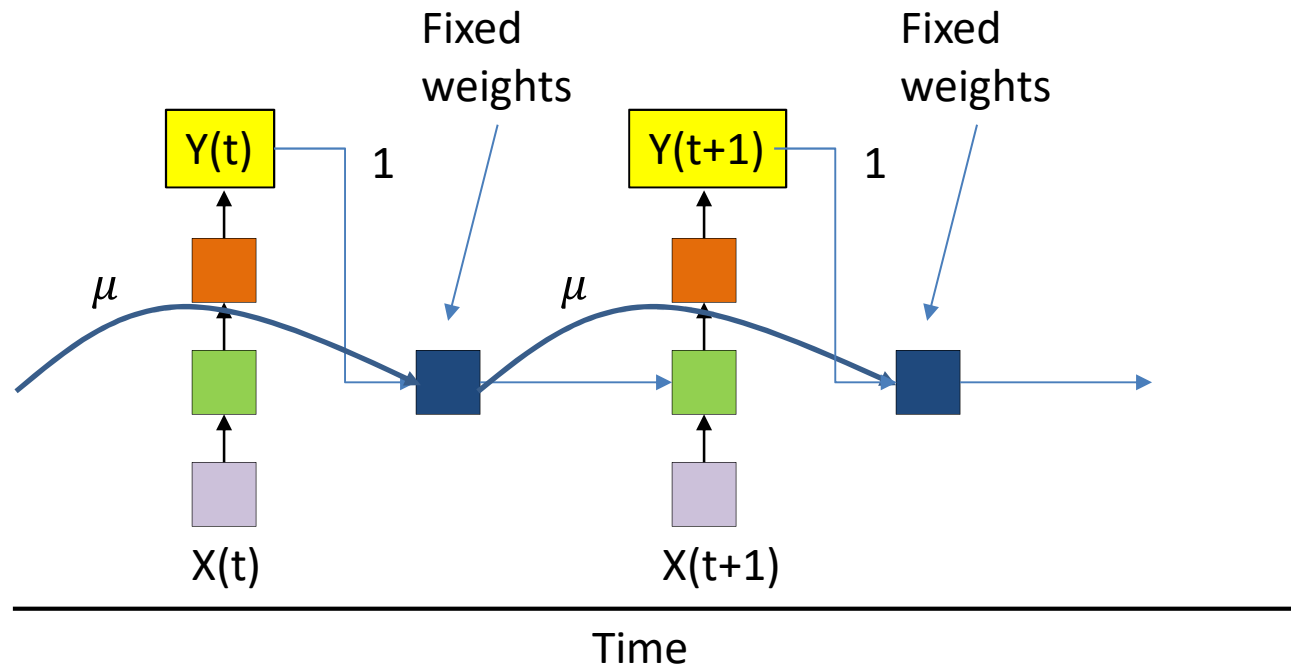
$$m_t = r(y_{t-1}, h_{t-1}, m_{t-1})$$

$$h_t = f(x_t, m_t)$$

$$y_t = g(h_t)$$

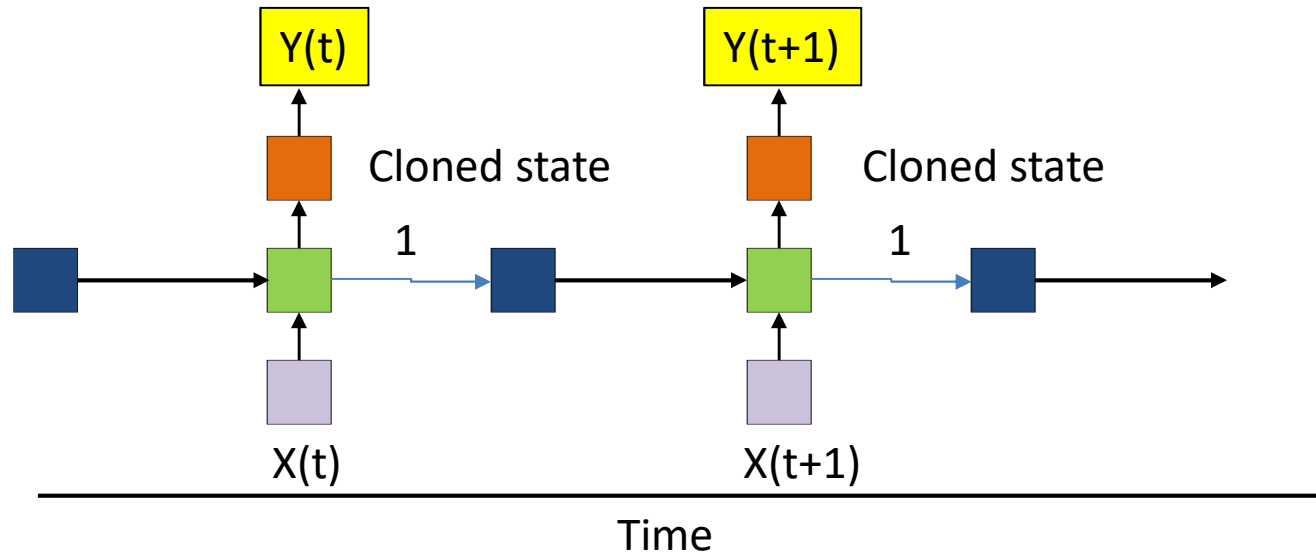
- m_t is a “memory” variable
 - Generally stored in a “memory” unit
 - Used to “remember” the past

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

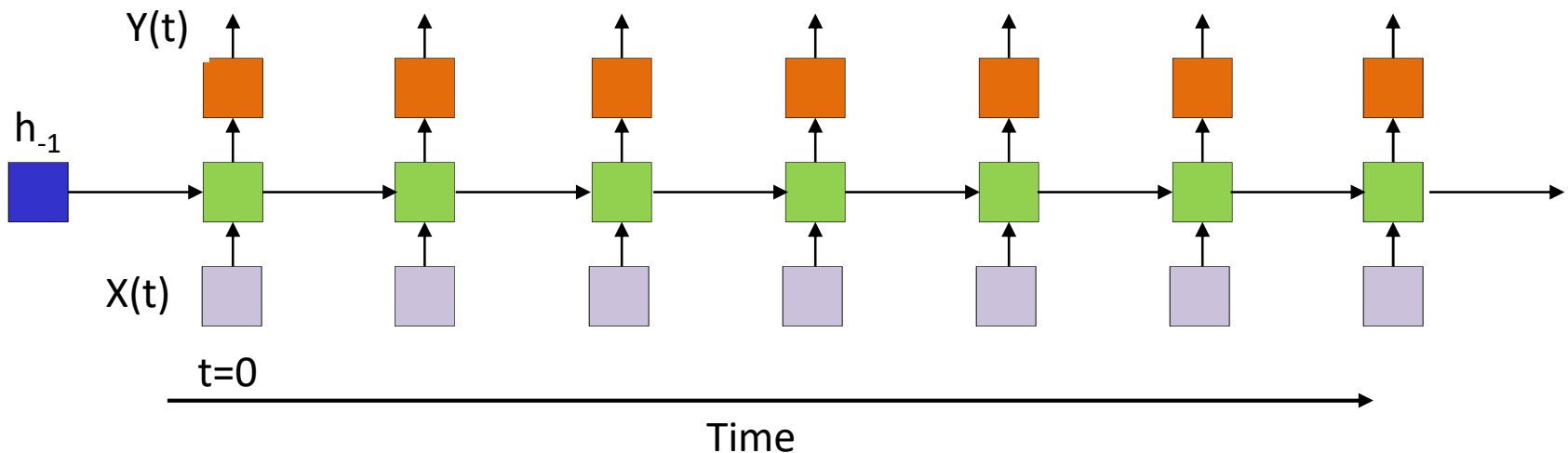
- In time series analysis, models must look at past inputs along with current input
 - Looking at a finite horizon of past inputs gives us a convolutional network
- Looking into the infinite past requires recursion
- NARX networks recurse by feeding back the output to the input
 - May feed back a finite horizon of outputs
- “Simple” recurrent networks:
 - Jordan 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

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
 - 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*
- *State* summarizes information about the entire past

The simple state-space model

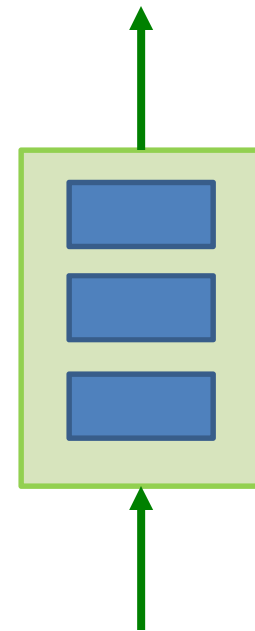


- The state (green) at any time is determined by the input at that time, and the state at the previous time
- *An input at $t=0$ affects outputs forever*
- Also known as a recurrent neural net

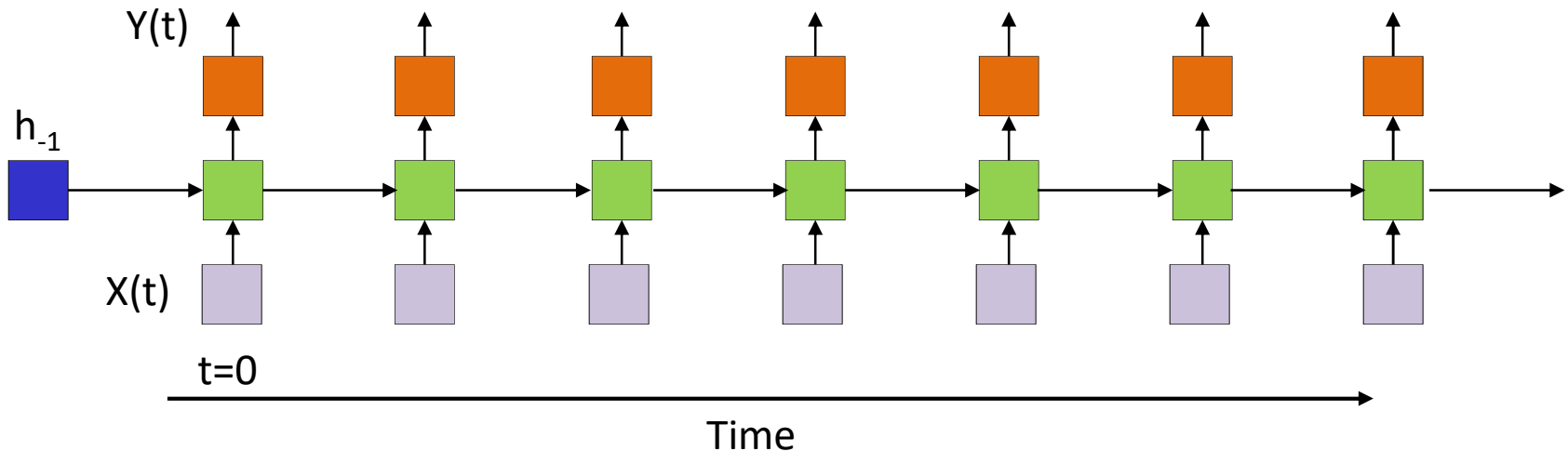
An alternate model for infinite response systems: **the state-space model**

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

- h_t is the *state* of the network
- Need to define initial state h_{-1}
- The state can be arbitrarily complex

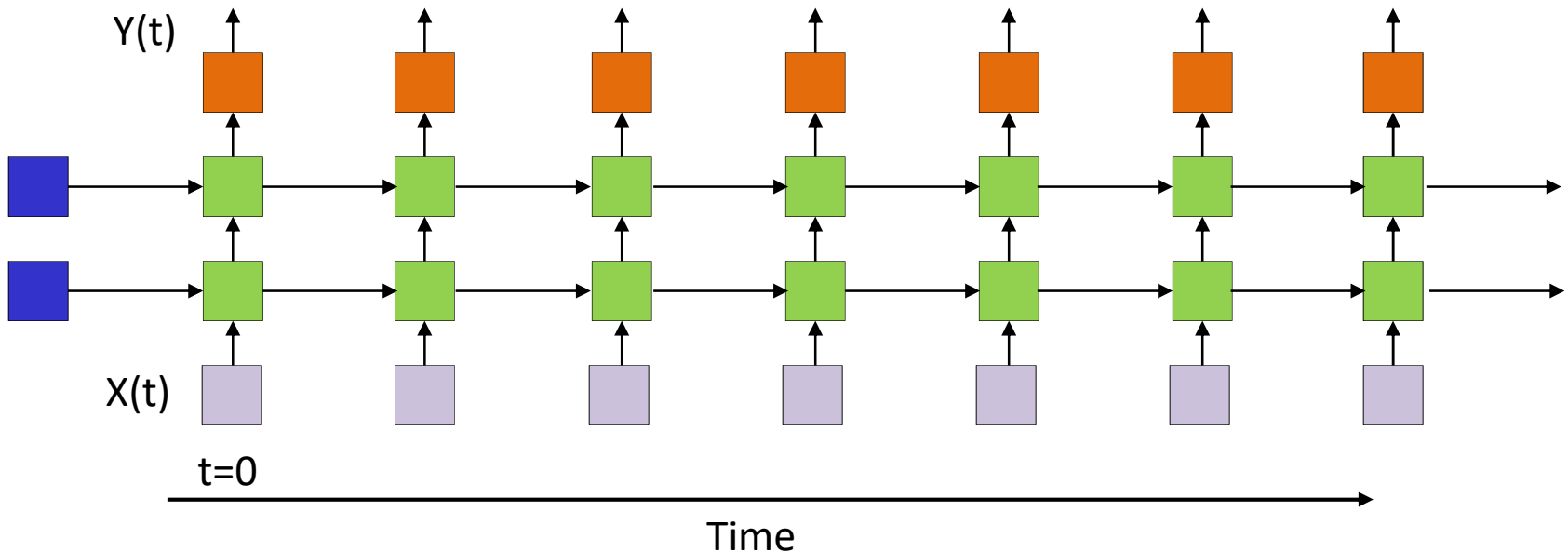


Single hidden layer RNN



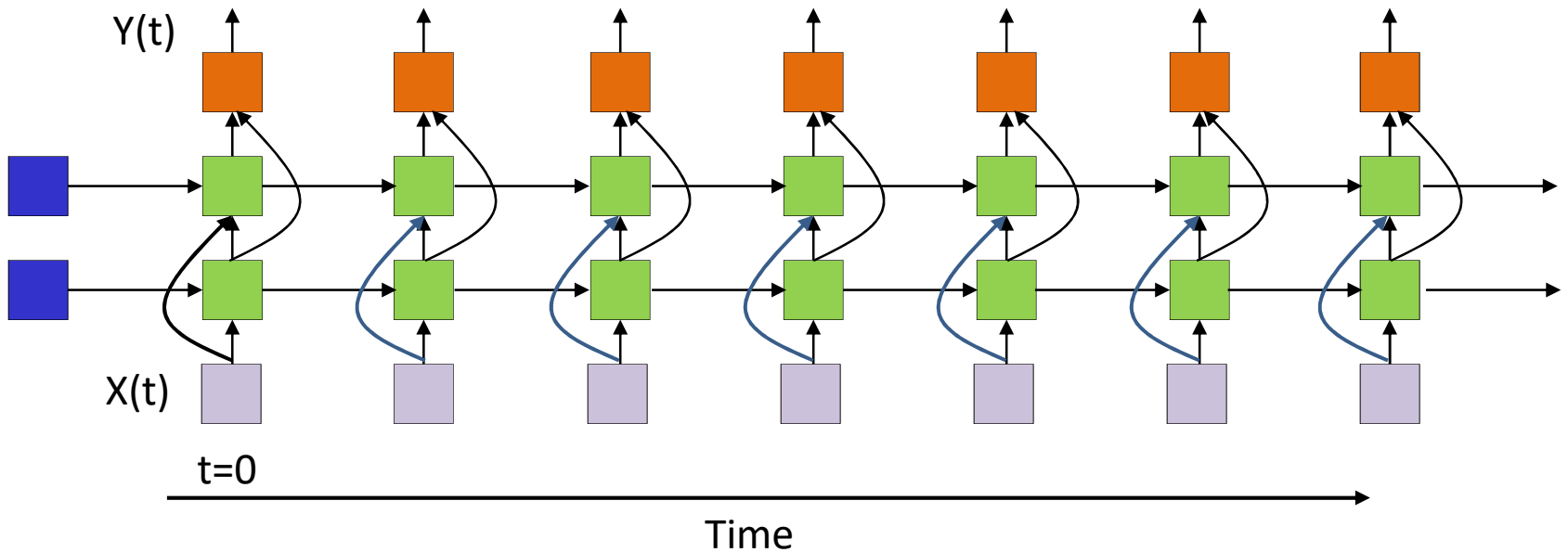
- Recurrent neural network
- All columns are identical
- *An input at $t=0$ affects outputs forever*

Multiple recurrent layer RNN



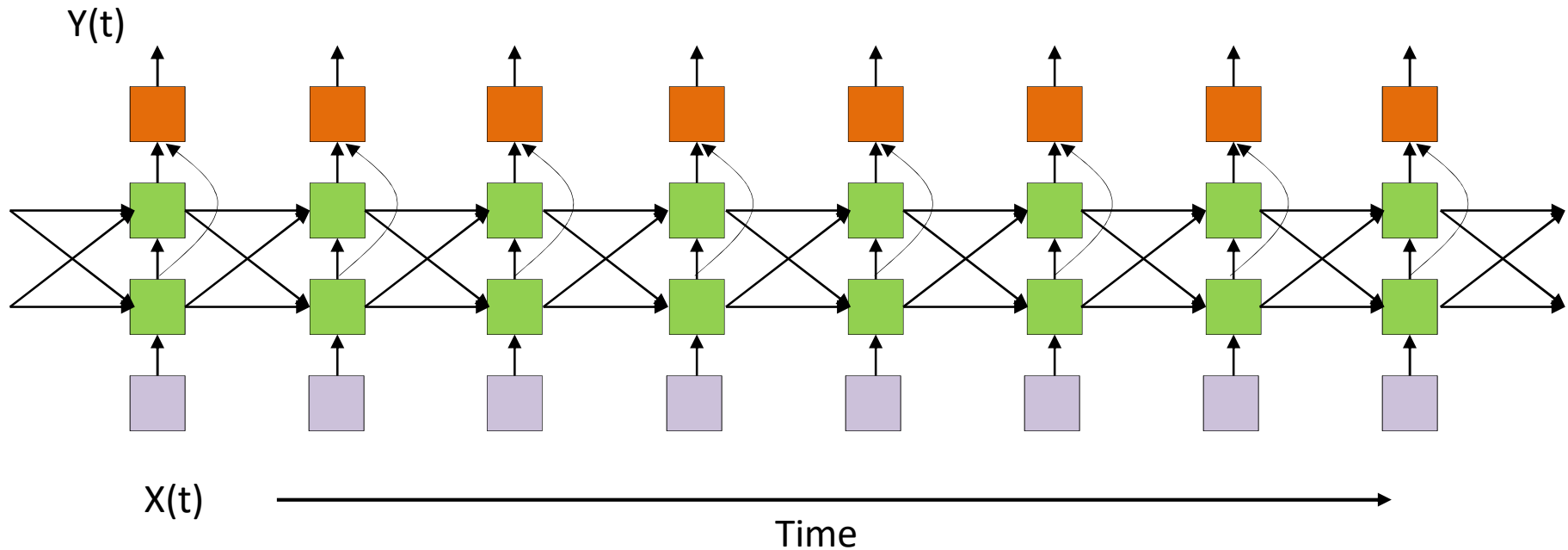
- Recurrent neural network
- All columns are identical
- *An input at $t=0$ affects outputs forever*

Multiple recurrent layer RNN



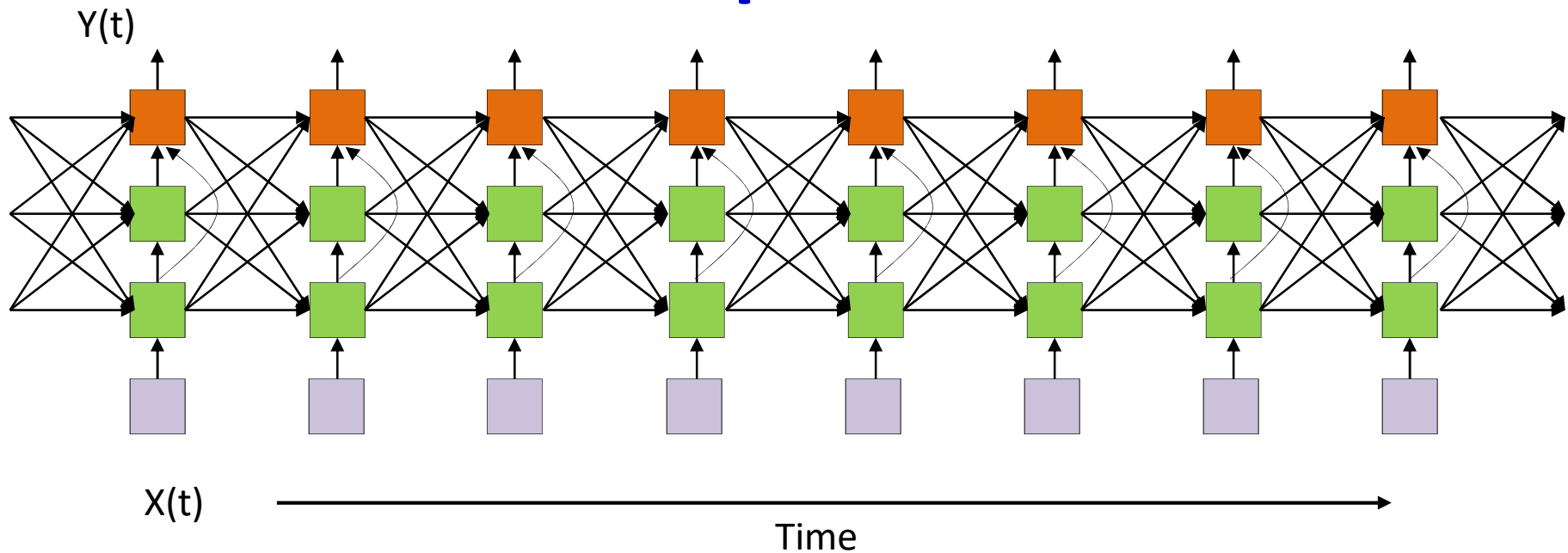
- We can also have skips..

A more complex state



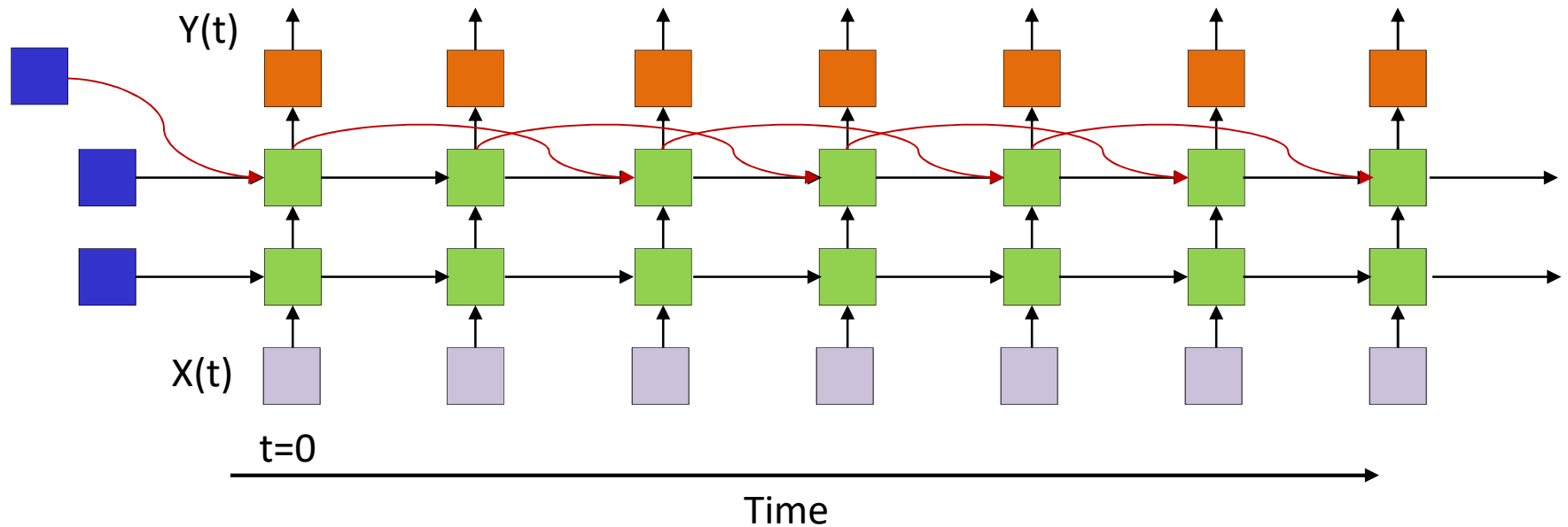
- All columns are identical
- *An input at $t=0$ affects outputs forever*

Or the network may be even more complicated



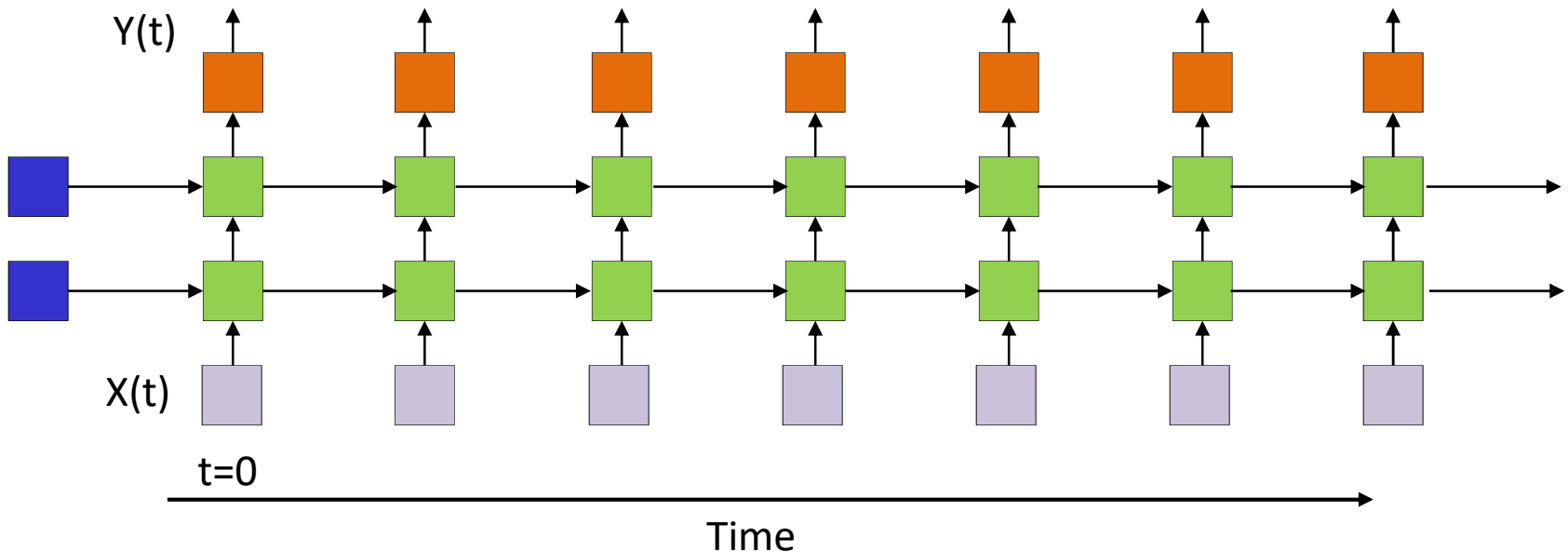
- Shades of NARX
- All columns are identical
- *An input at $t=0$ affects outputs forever*

Generalization with other recurrences



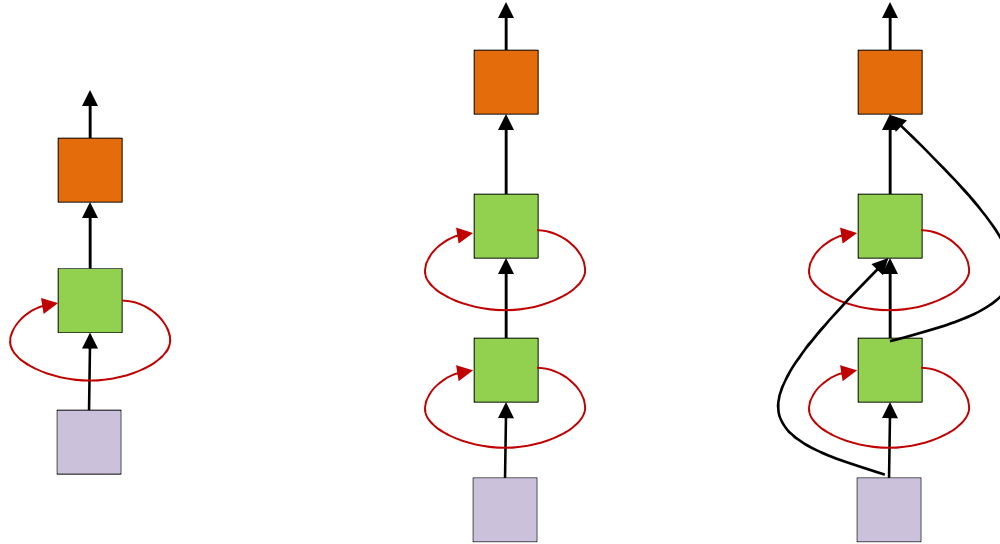
- All columns (including incoming edges) are identical

The simplest structures are most popular



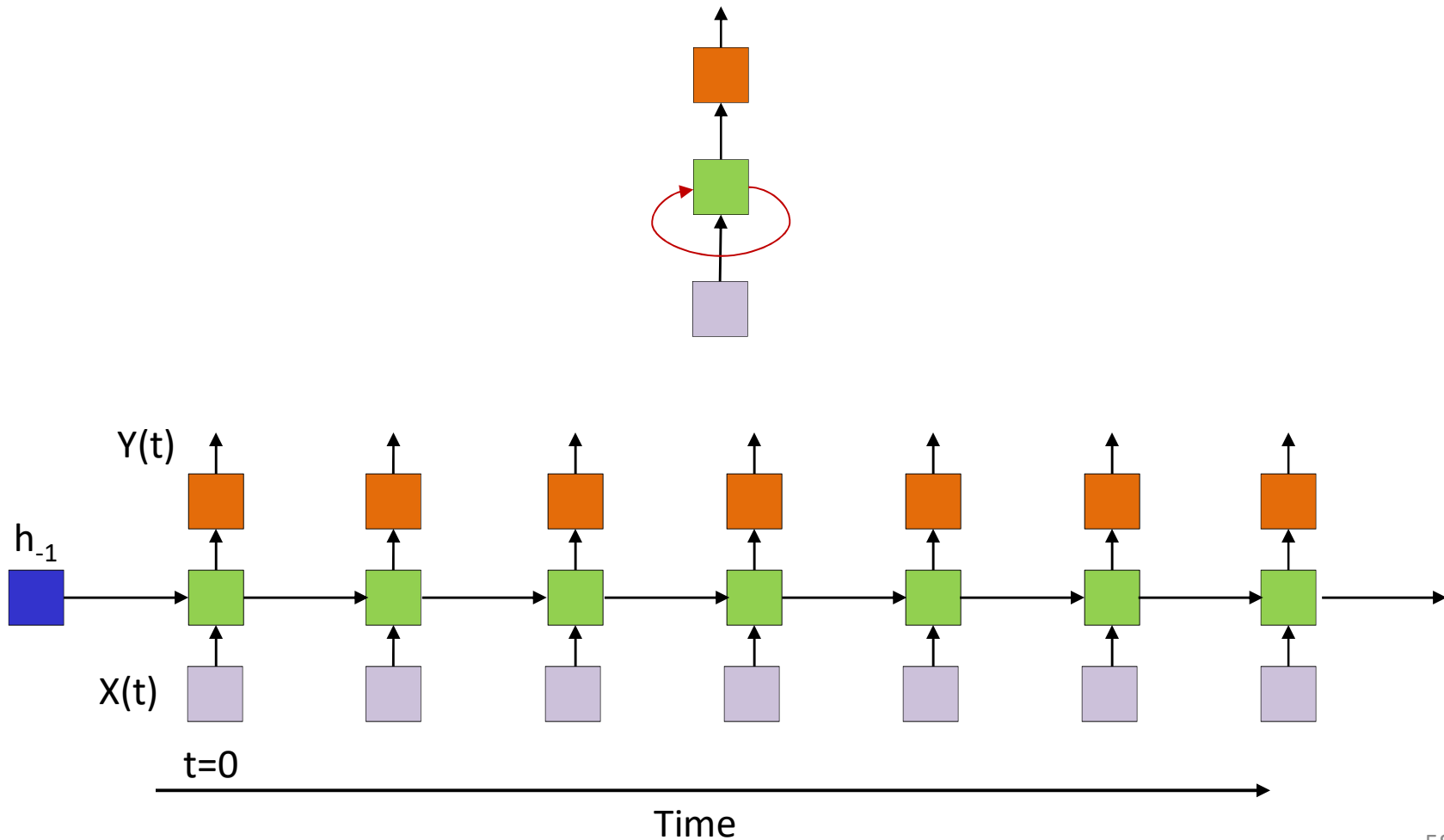
- Recurrent neural network
- All columns are identical
- *An input at $t=0$ affects outputs forever*

A Recurrent Neural Network

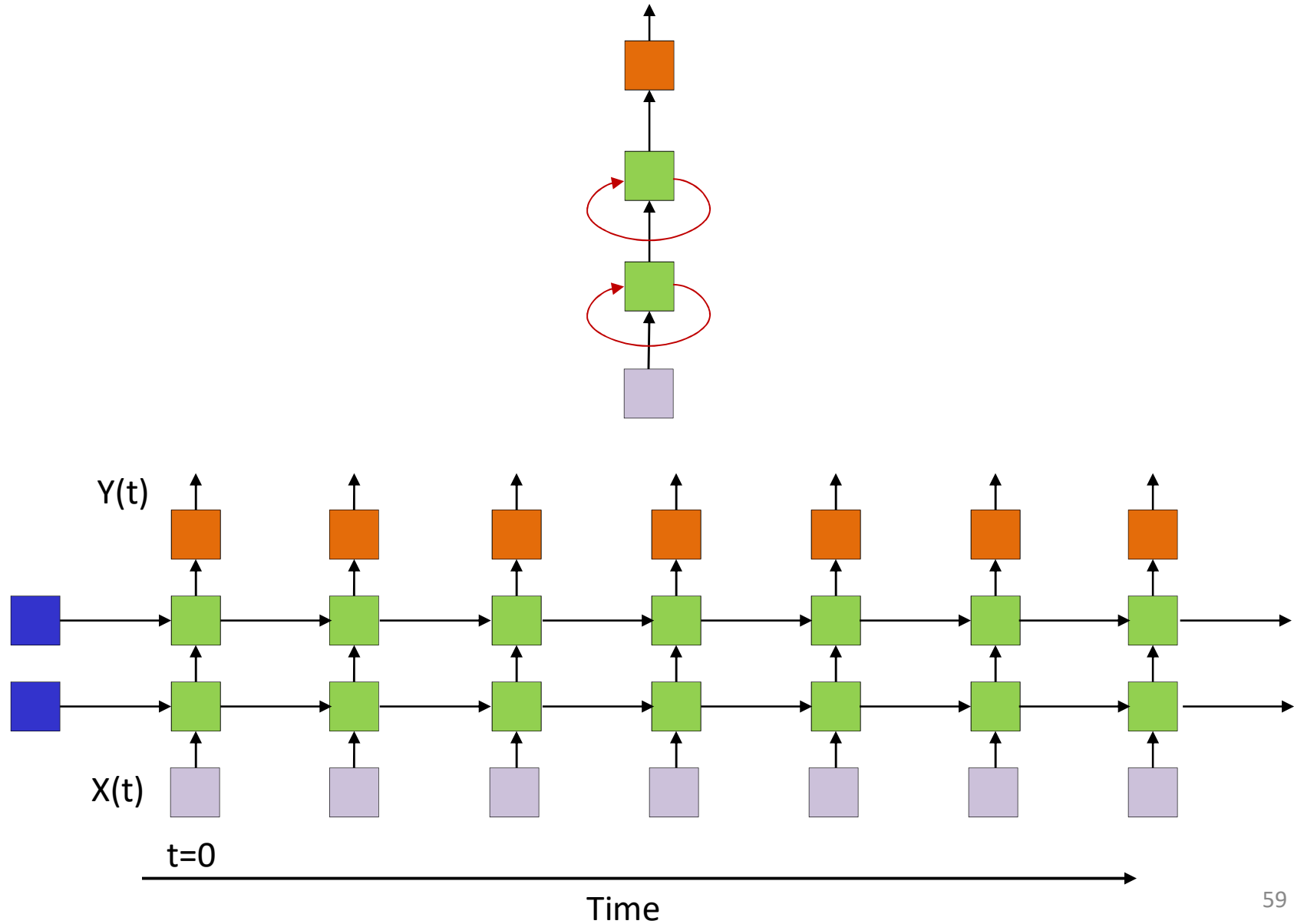


- Simplified models often drawn
- The loops imply recurrence

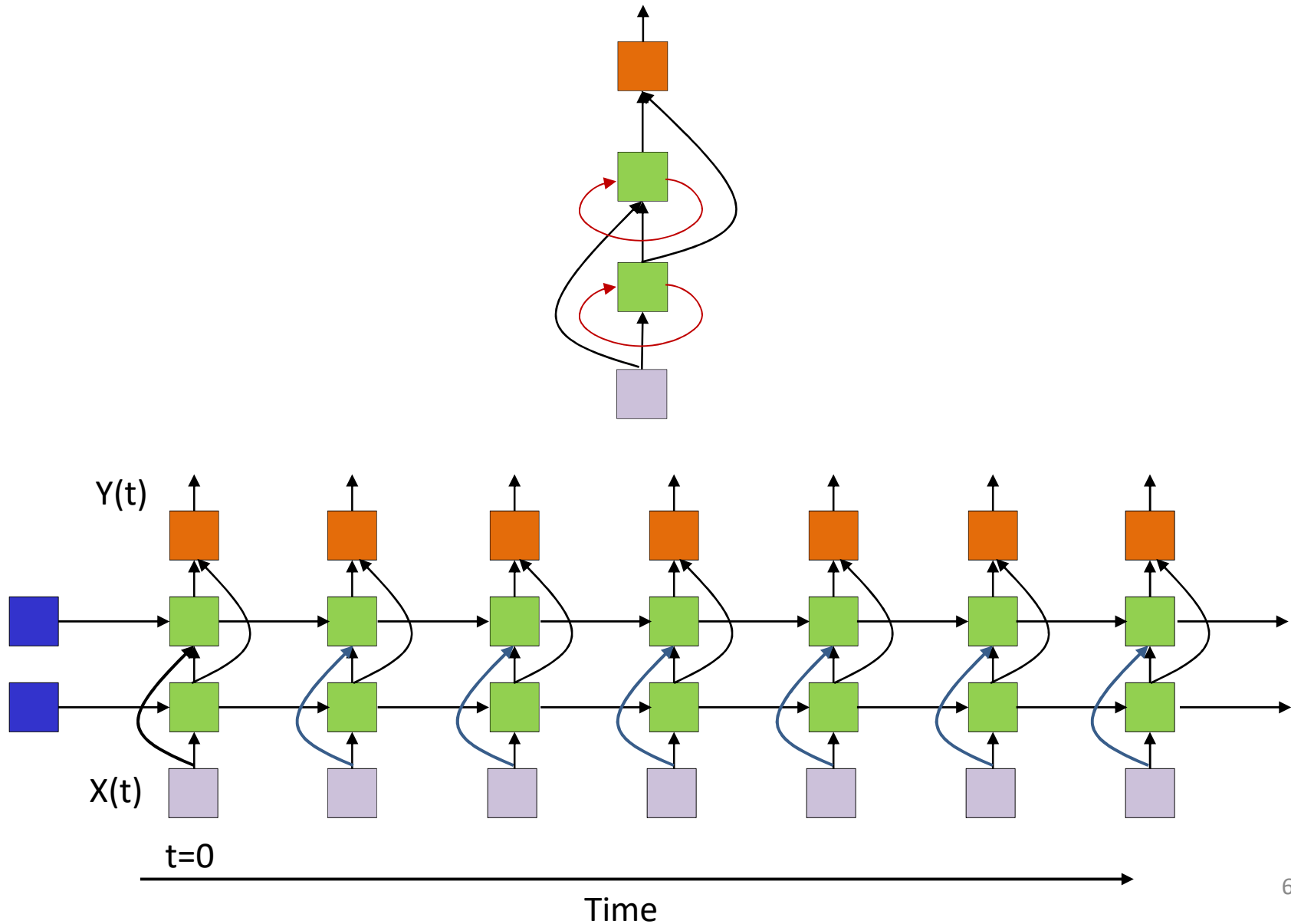
The detailed version of the simplified representation



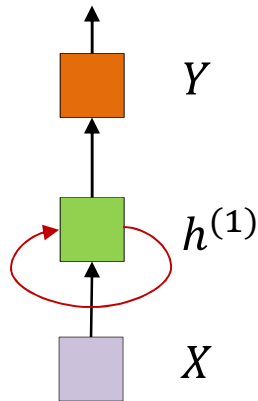
Multiple recurrent layer RNN



Multiple recurrent layer RNN



Equations



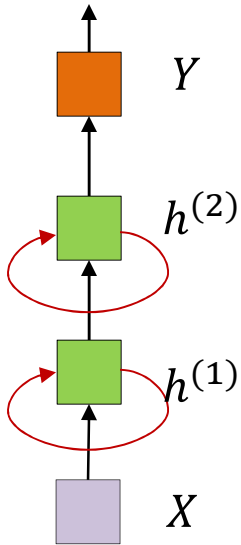
$h_i^{(1)}(-1) = \text{part of network parameters}$

$$h_i^{(1)}(t) = f_1 \left(\sum_j w_{ji}^{(0)} X_j(t) + \sum_j w_{ji}^{(11)} h_i^{(1)}(t-1) + b_i^{(1)} \right)$$

$$Y(t) = f_2 \left(\sum_j w_{jk}^{(1)} h_j^{(1)}(t) + b_k^{(1)}, k = 1..M \right)$$

- 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



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

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

$$h_i^{(1)}(t) = f_1 \left(\sum_j w_{ji}^{(0)} X_j(t) + \sum_j w_{ji}^{(11)} h_i^{(1)}(t-1) + b_i^{(1)} \right)$$

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

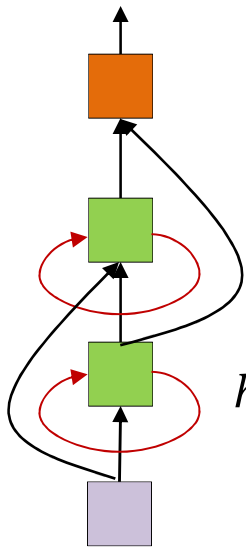
$$Y(t) = f_3 \left(\sum_j w_{jk}^{(2)} h_j^{(2)}(t) + b_k^{(3)}, k = 1..M \right)$$

- 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

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

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



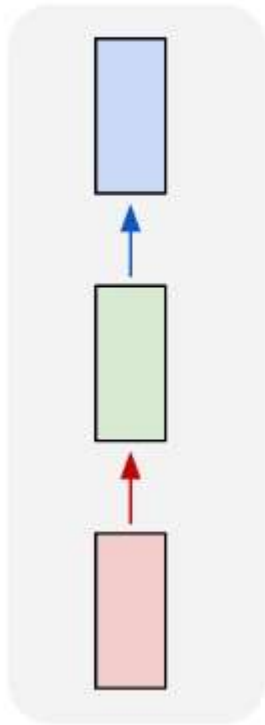
$$h_i^{(1)}(t) = f_1 \left(\sum_j w_{ji}^{(0,1)} X_j(t) + \sum_i w_{ii}^{(1,1)} h_i^{(1)}(t-1) + b_i^{(1)} \right)$$

$$h_i^{(2)}(t) = f_2 \left(\sum_j w_{ji}^{(1,2)} h_j^{(1)}(t) + \sum_j w_{ji}^{(0,2)} X_j(t) + \sum_i w_{ii}^{(2,2)} h_i^{(2)}(t-1) + b_i^{(2)} \right)$$

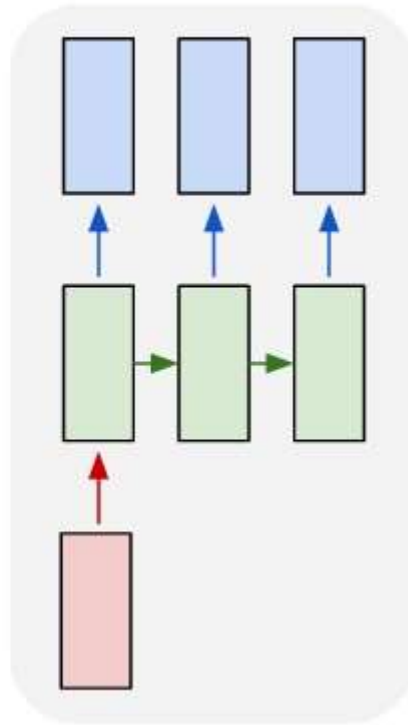
$$Y_i(t) = f_3 \left(\sum_j w_{jk}^{(2)} h_j^{(2)}(t) + \sum_j w_{jk}^{(1,3)} h_j^{(1)}(t) + b_k^{(3)}, k = 1..M \right)$$

Variants on recurrent nets

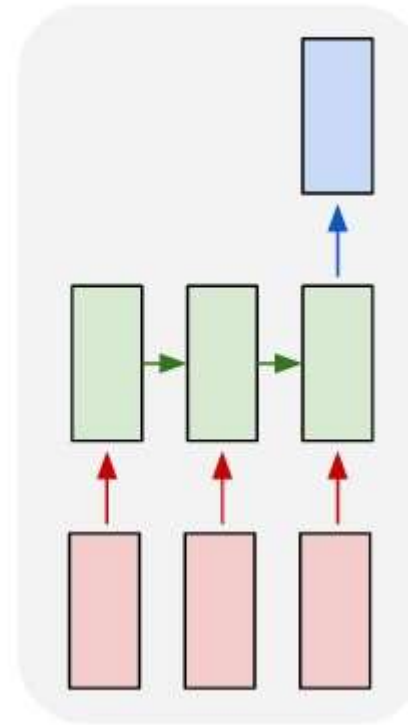
one to one



one to many



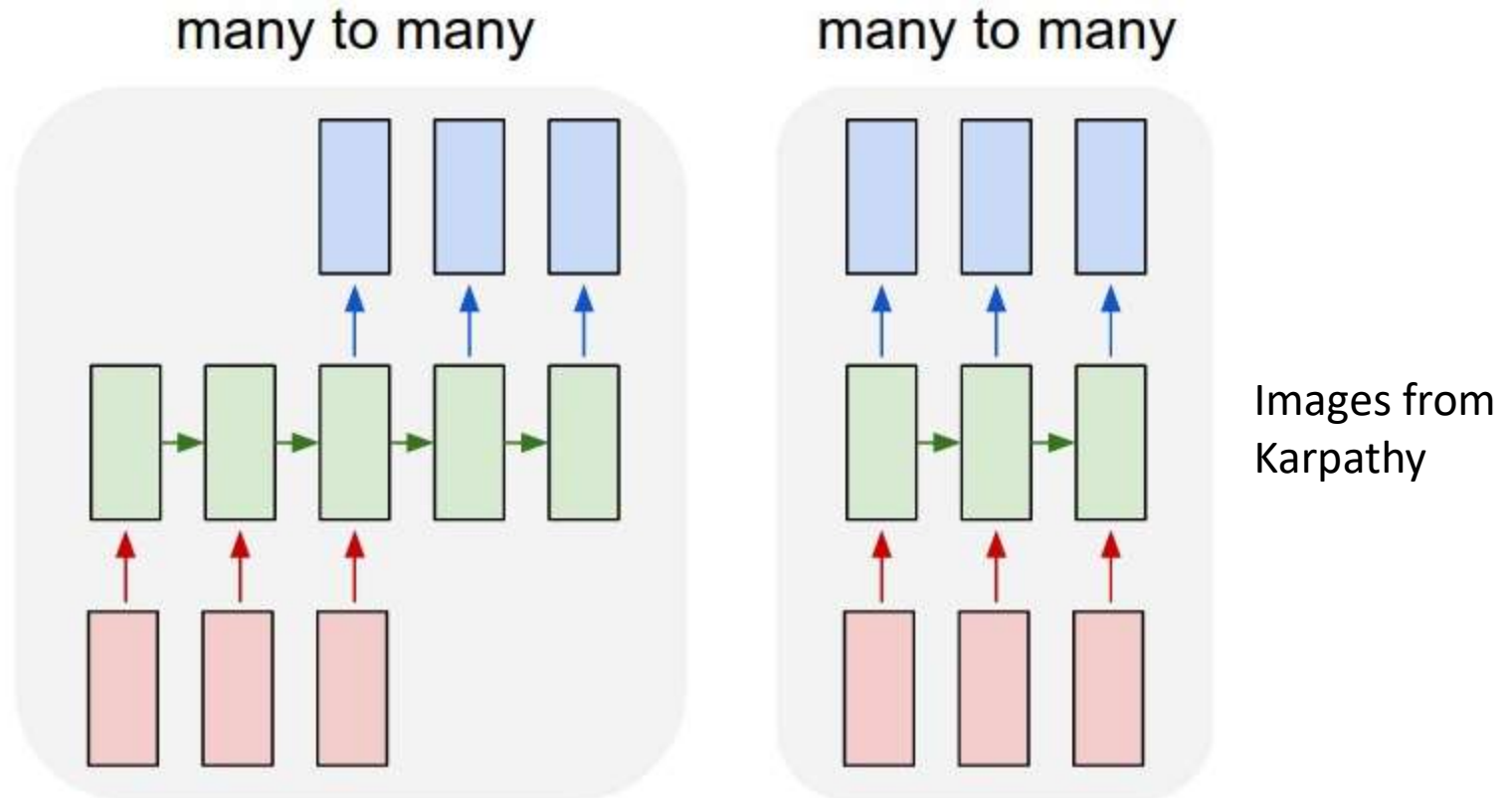
many to one



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

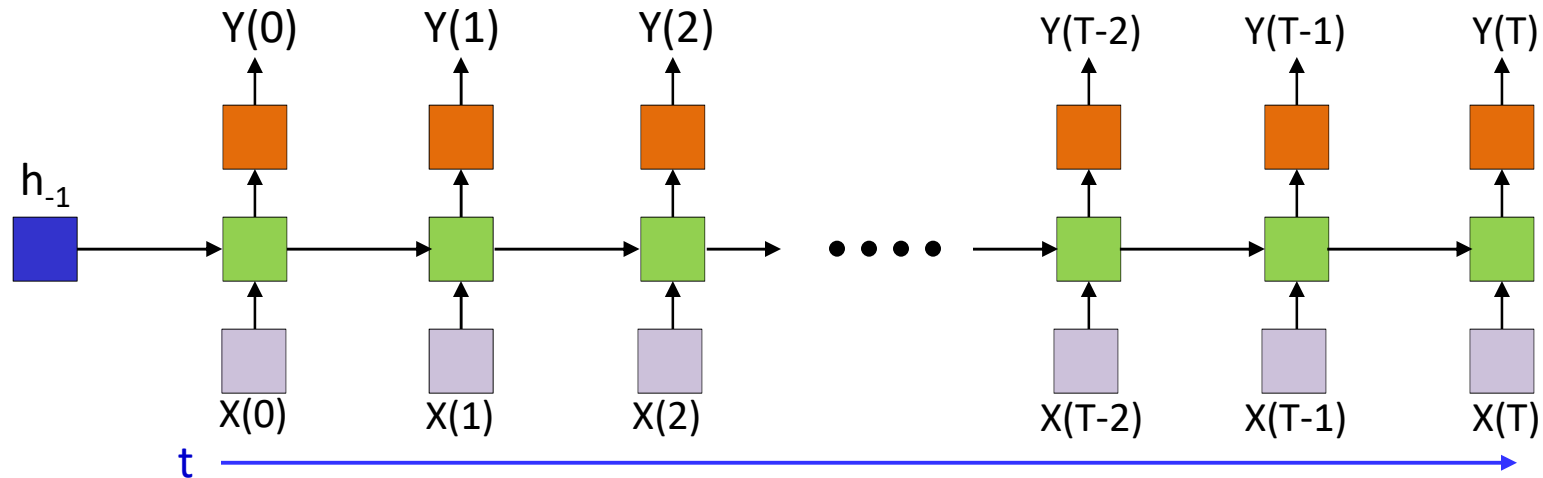


- 1: *Delayed* sequence to sequence
- 2: Sequence to sequence, e.g. stock problem, label prediction
- Etc...

Story so far

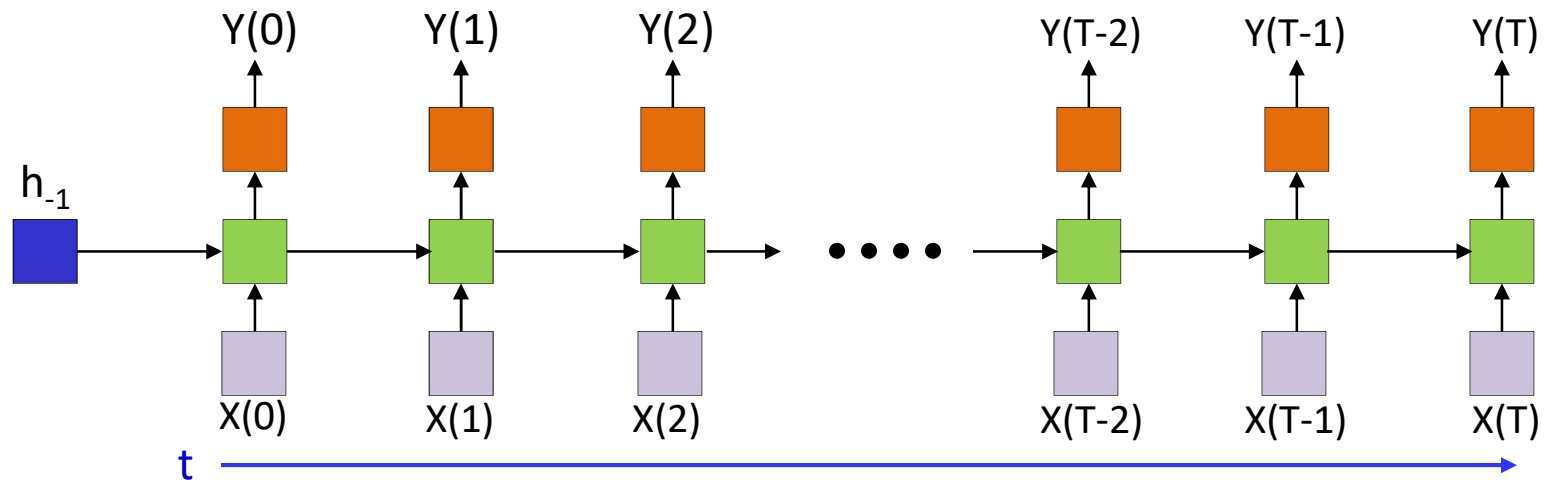
- Time series analysis must consider past inputs along with current input
- Looking into the infinite past requires recursion
- NARX networks achieve this by feeding back the output to the input
- “Simple” recurrent networks maintain separate “memory” or “context” units to retain some information about 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: Forward pass



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

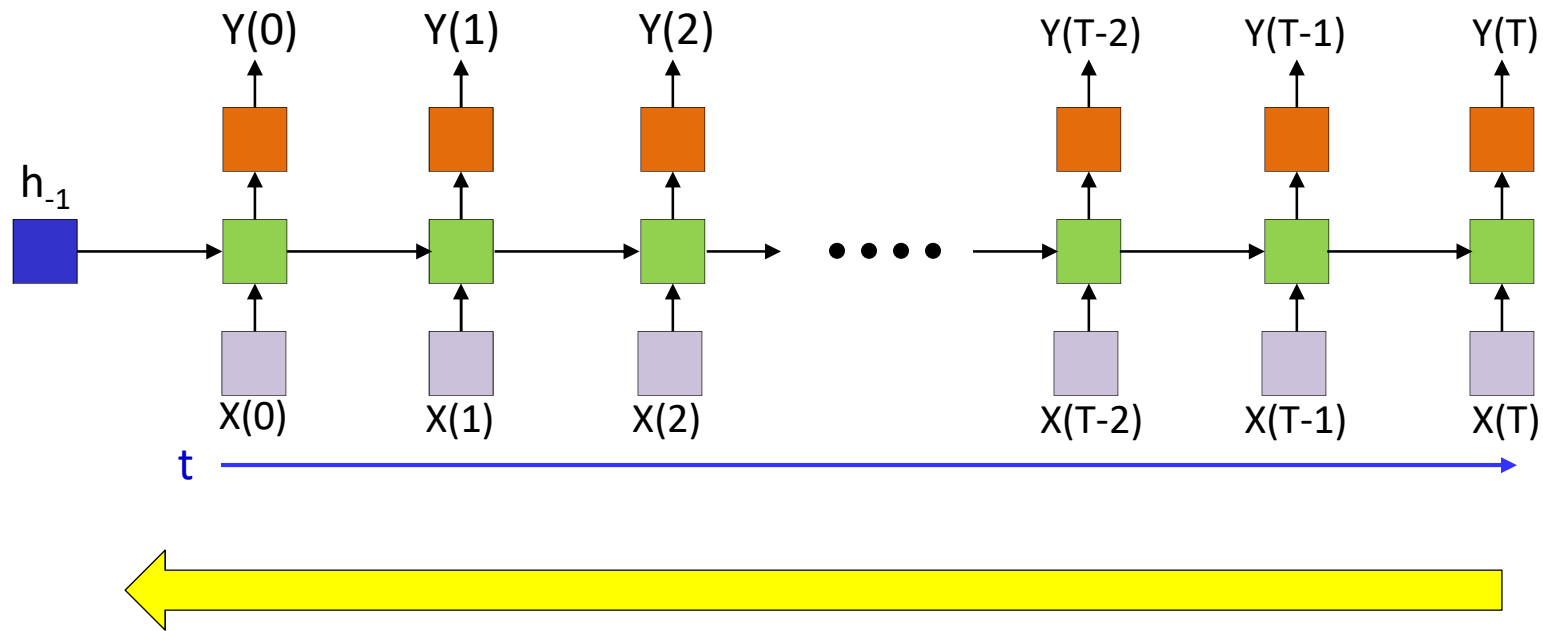
Recurrent Neural Net

Assuming time-synchronous output

```
# Assuming  $h(-1,*)$  is known
# Assuming L hidden-state layers and an output layer
#  $W_c(*)$  and  $W_r(*)$  are matrices,  $b(*)$  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 layer weights
```

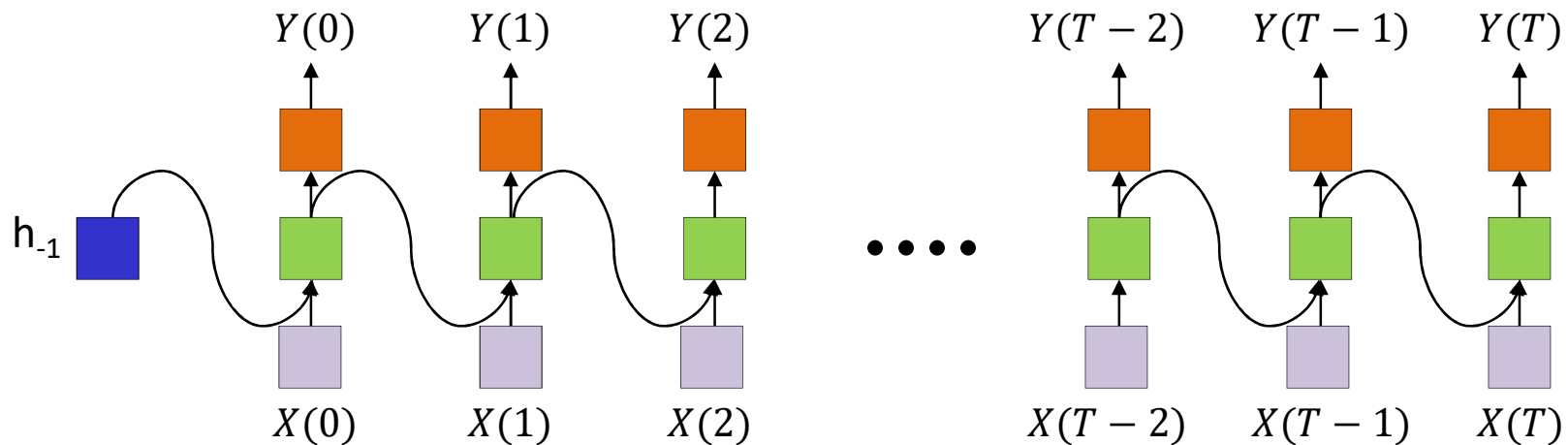
```
for t = 0:T-1 # Including both ends of the index
     $h(t,0) = x(t)$  # Vectors. Initialize  $h(0)$  to input
    for l = 1:L # hidden layers operate at time t
         $z(t,l) = W_c(l)h(t,l-1) + W_r(l)h(t-1,l) + b(l)$ 
         $h(t,l) = \tanh(z(t,l))$  # Assuming tanh activ.
     $z_o(t) = W_o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

Training: Computing gradients



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

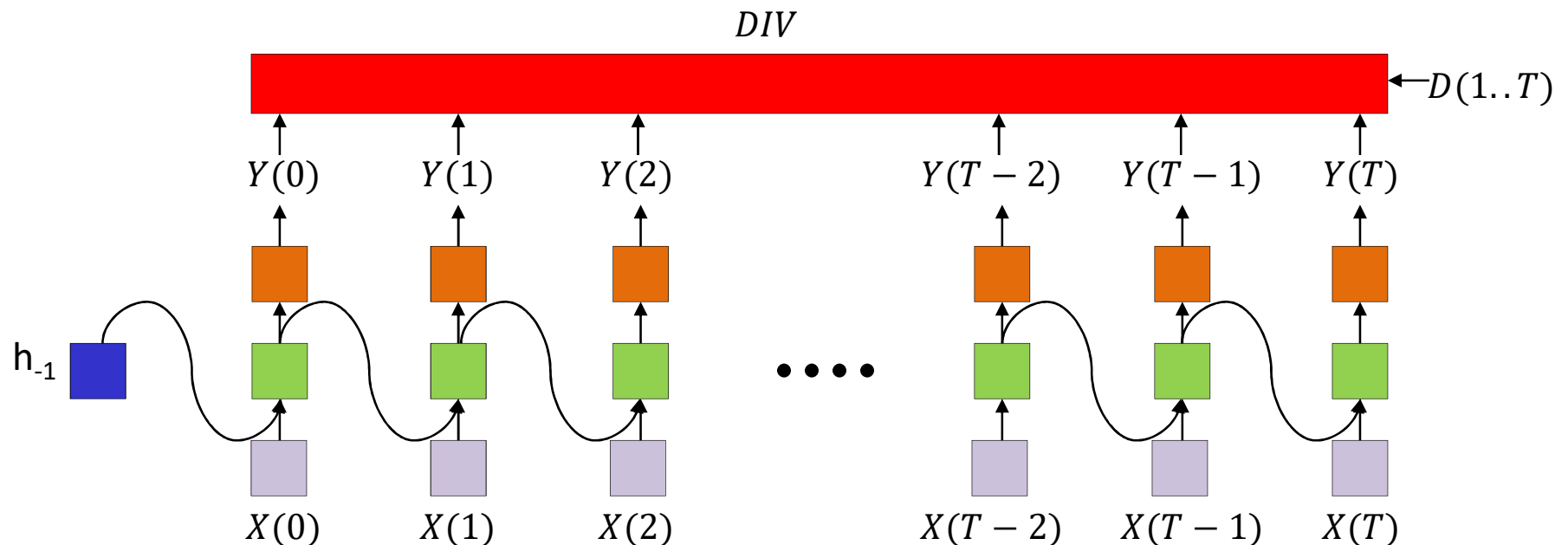
Back Propagation Through Time



Will only focus on *one* training instance

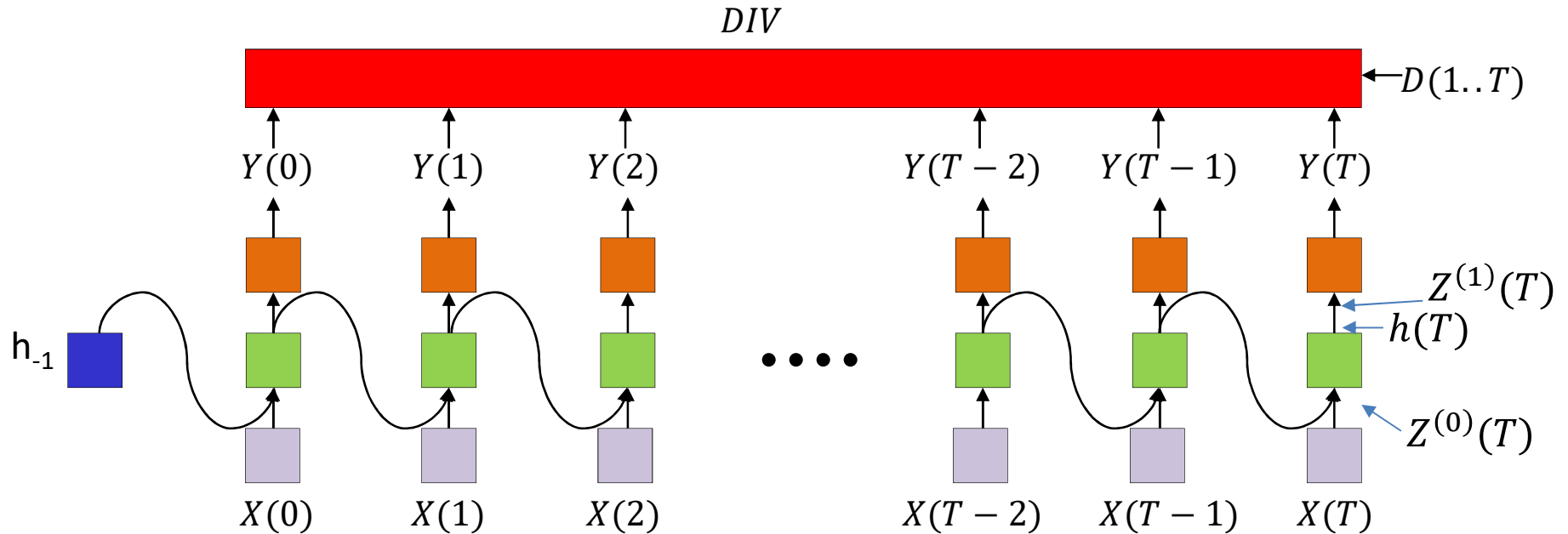
All subscripts represent *components* and not training instance index

Back Propagation Through Time



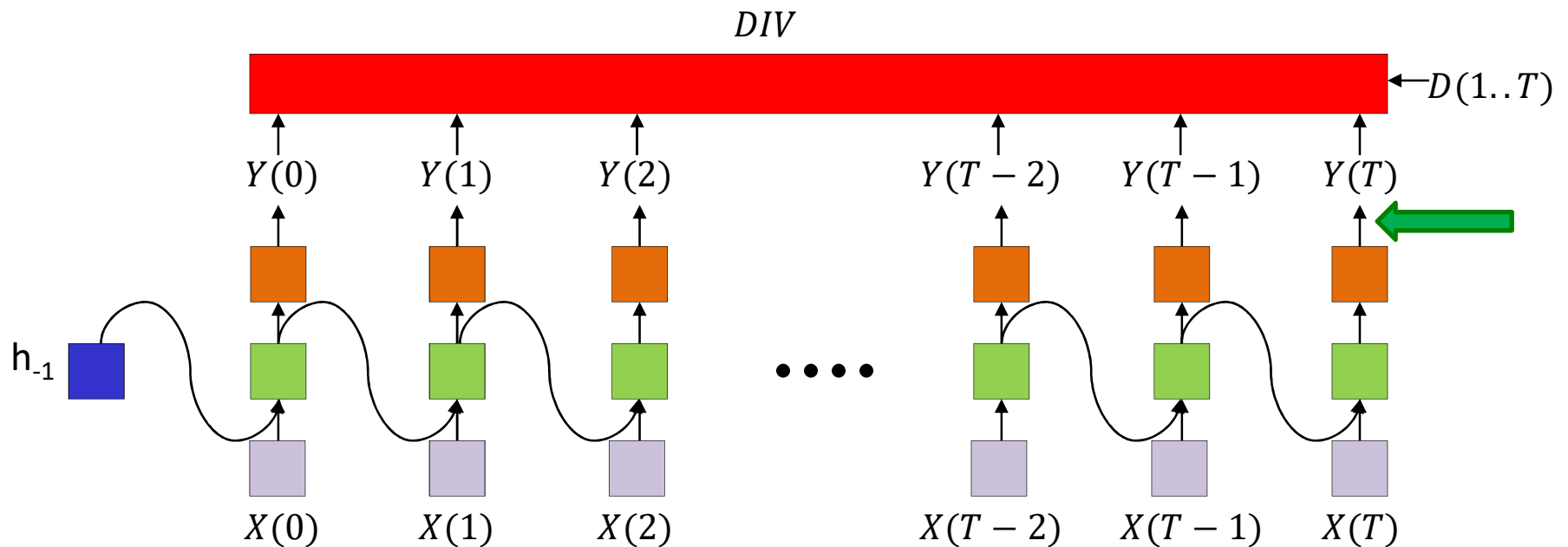
- The divergence computed is between the *sequence of outputs* by the network and the *desired sequence of outputs*
 - DIV is a scalar function of a series of vectors!
- This is *not* just the sum of the divergences at individual times
 - Unless we explicitly define it that way

Notation



- $Y(t)$ is the output at time t
 - $Y_i(t)$ is the i th output
- $Z^{(1)}(t)$ 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
- $Z^{(0)}(t)$ is the pre-activation value of the hidden layer at time t

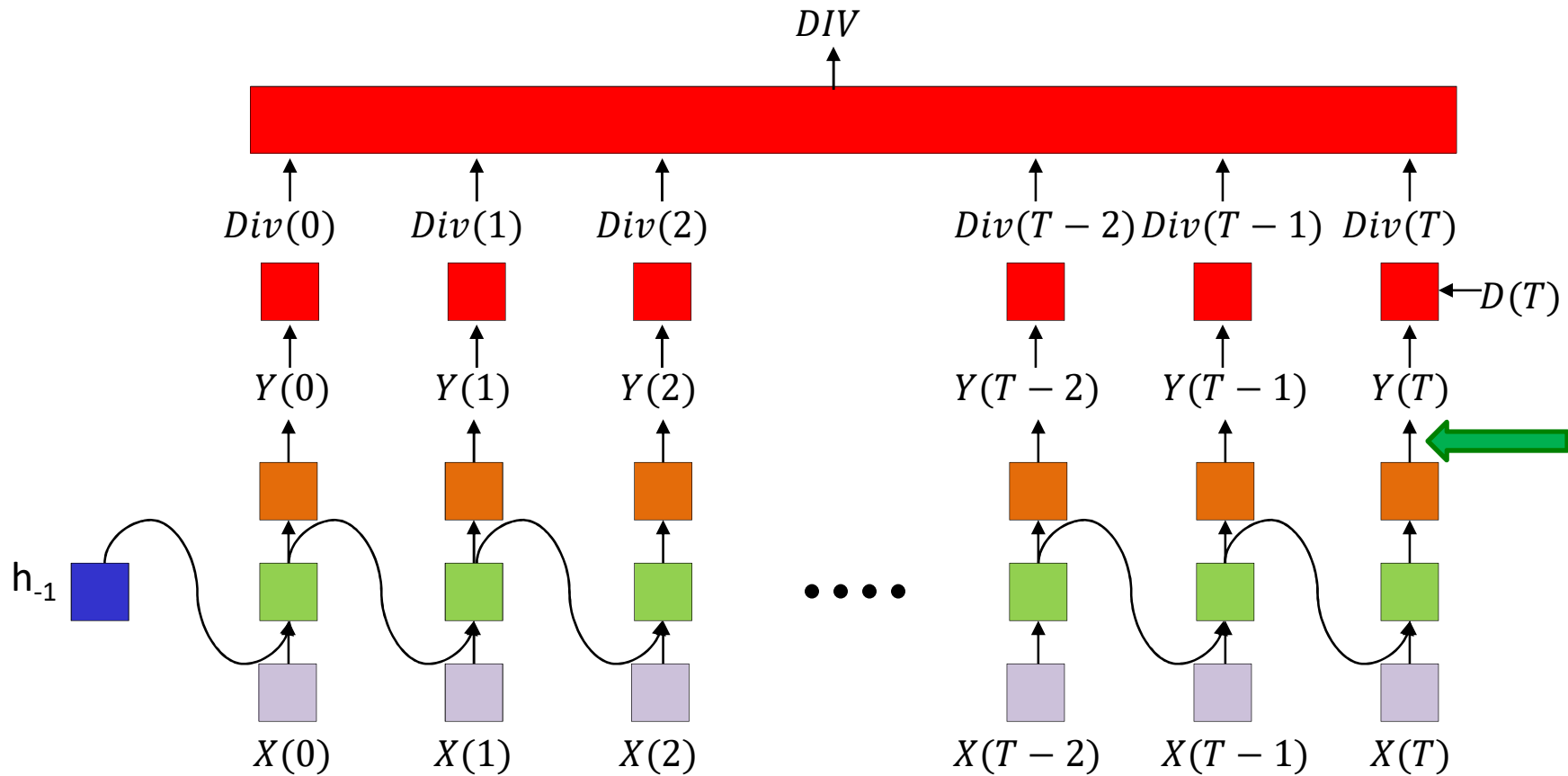
Back Propagation Through Time



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

Note: DIV is a function of *all* outputs $Y(0) \dots Y(T)$

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.



Special case, when the overall divergence is a simple combination of local divergences at each time:

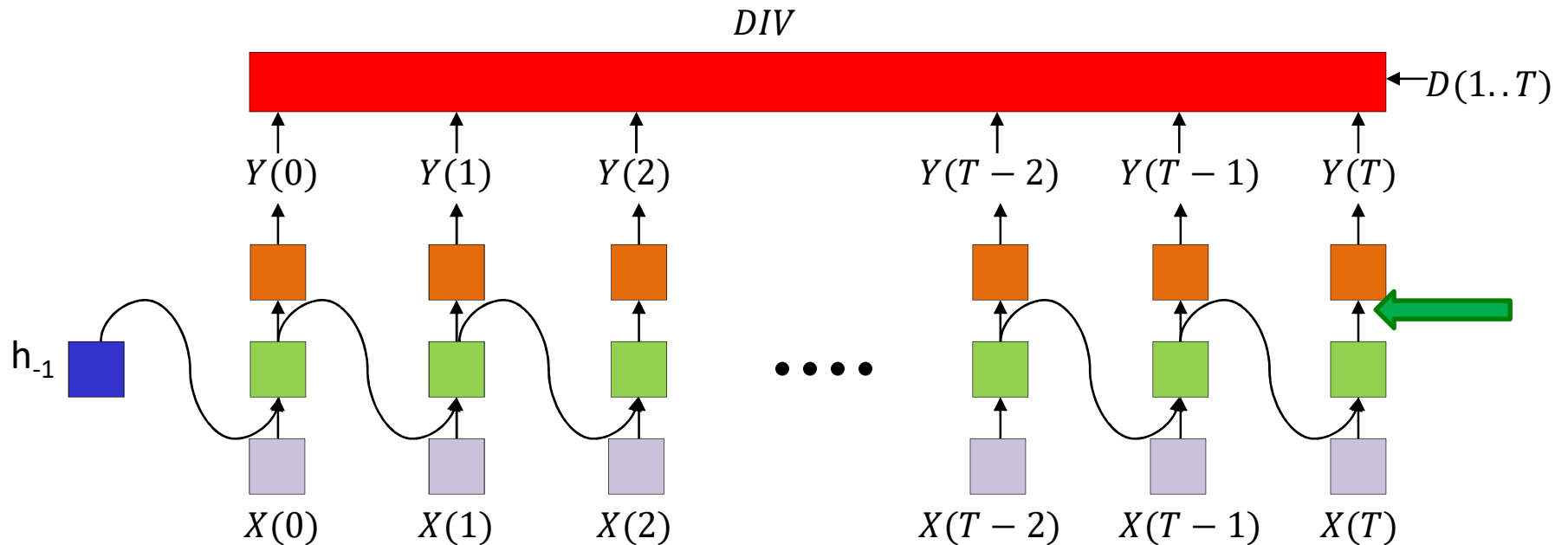
Must compute

$$\frac{dDIV}{dY_i(t)} \text{ for all } i \text{ for all } T$$

Will usually get

$$\frac{dDIV}{dY_i(t)} = \frac{dDiv(t)}{dY_i(t)}$$

Back Propagation Through Time



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

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

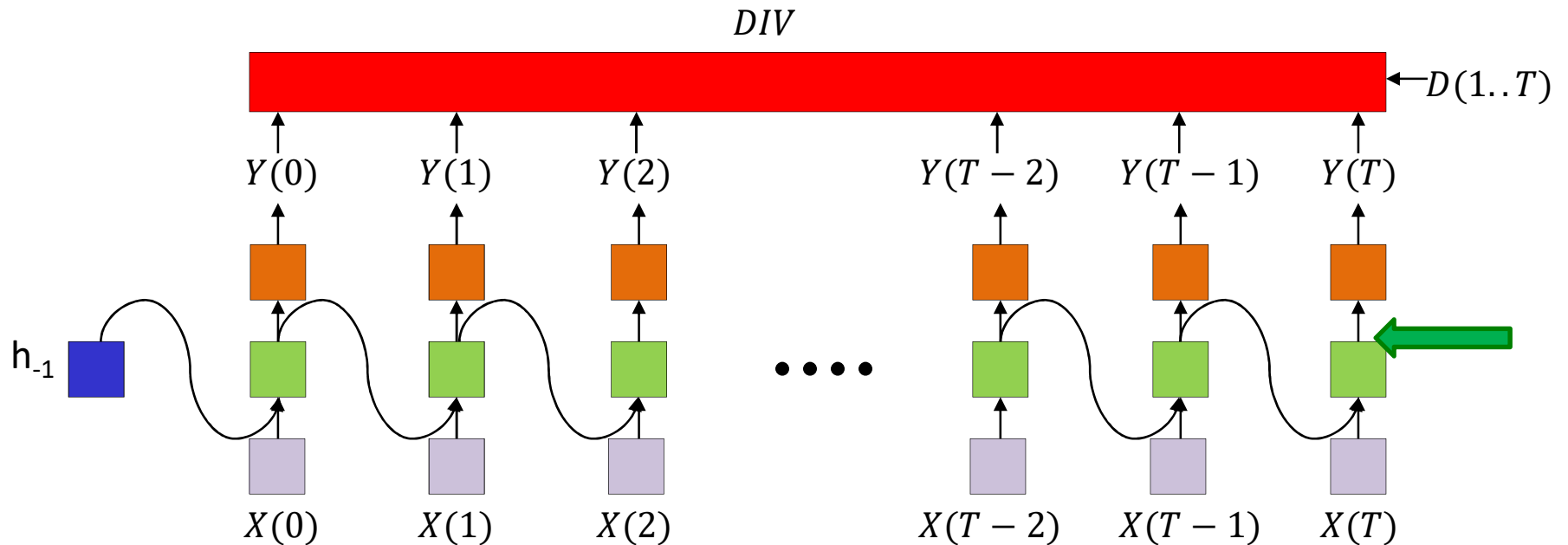
Vector output activation

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

OR

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

Back Propagation Through Time



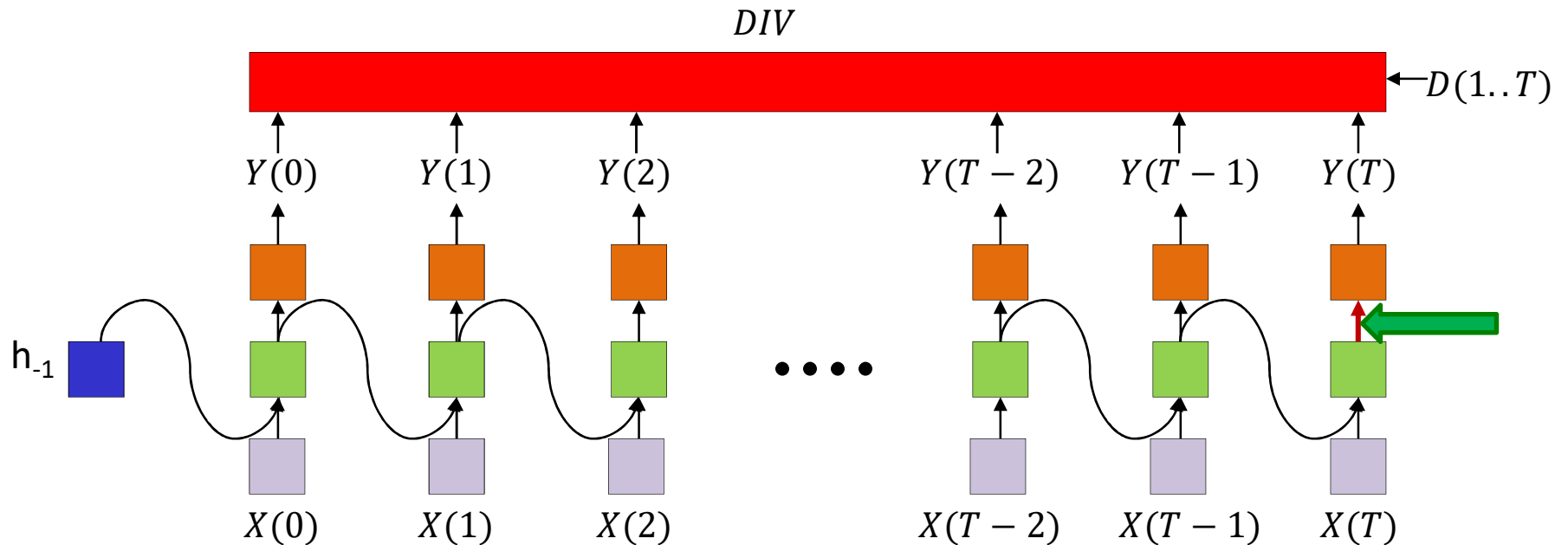
$$\frac{dDIV}{dY_i(T)} \text{ for all } i$$

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

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

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

Back Propagation Through Time



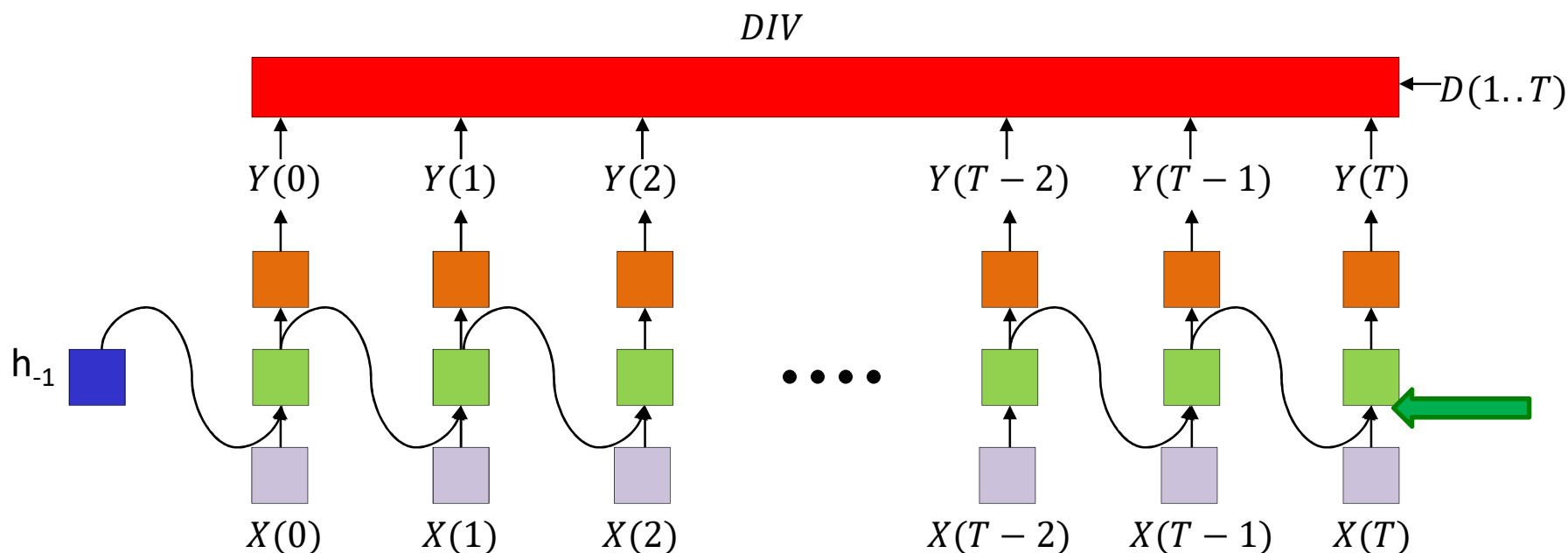
$$\frac{dDIV}{dZ_i^{(1)}(T)} = \frac{dDiv(T)}{dY_i(T)} \frac{dY_i(T)}{dZ_i^{(1)}(T)}$$

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

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

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

Back Propagation Through Time



$$\nabla_{Z^{(0)}(T)} DIV = \nabla_{h(T)} DIV \nabla_{Z^{(0)}(T)} h(T)$$

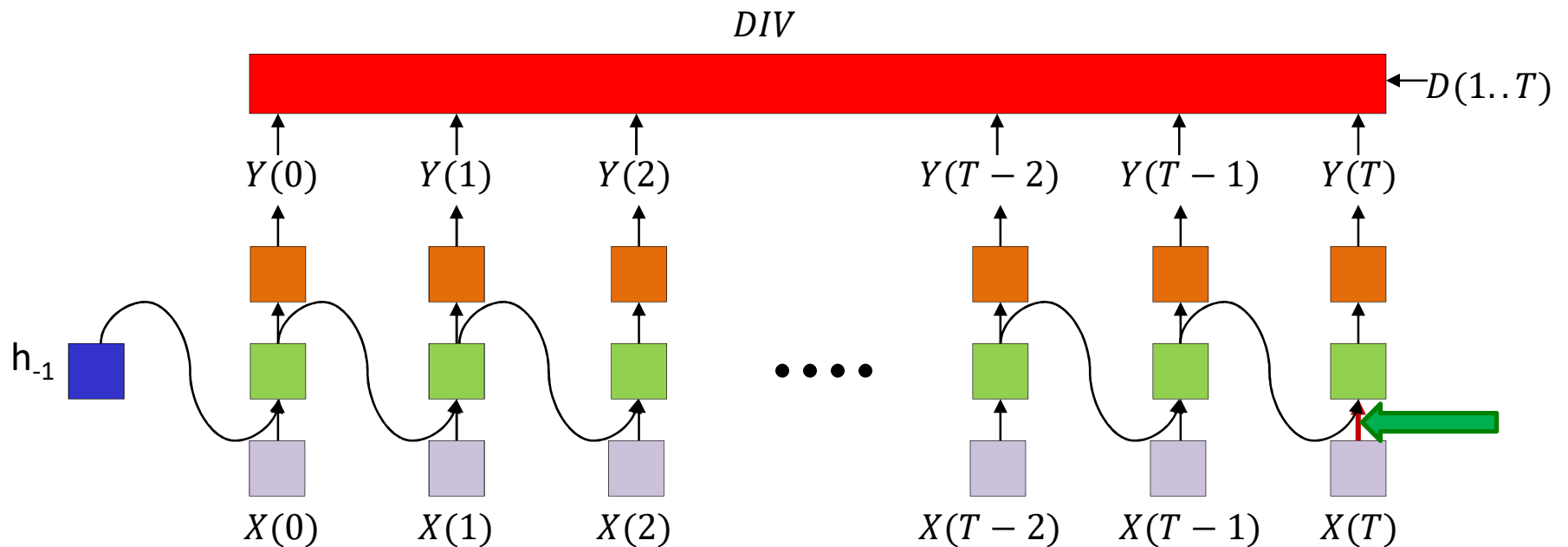
$$\frac{dDIV}{dZ_i^{(0)}(T)} = \frac{dDIV}{dh_i(T)} \frac{dh_i(T)}{dZ_i^{(0)}(T)}$$

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

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

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

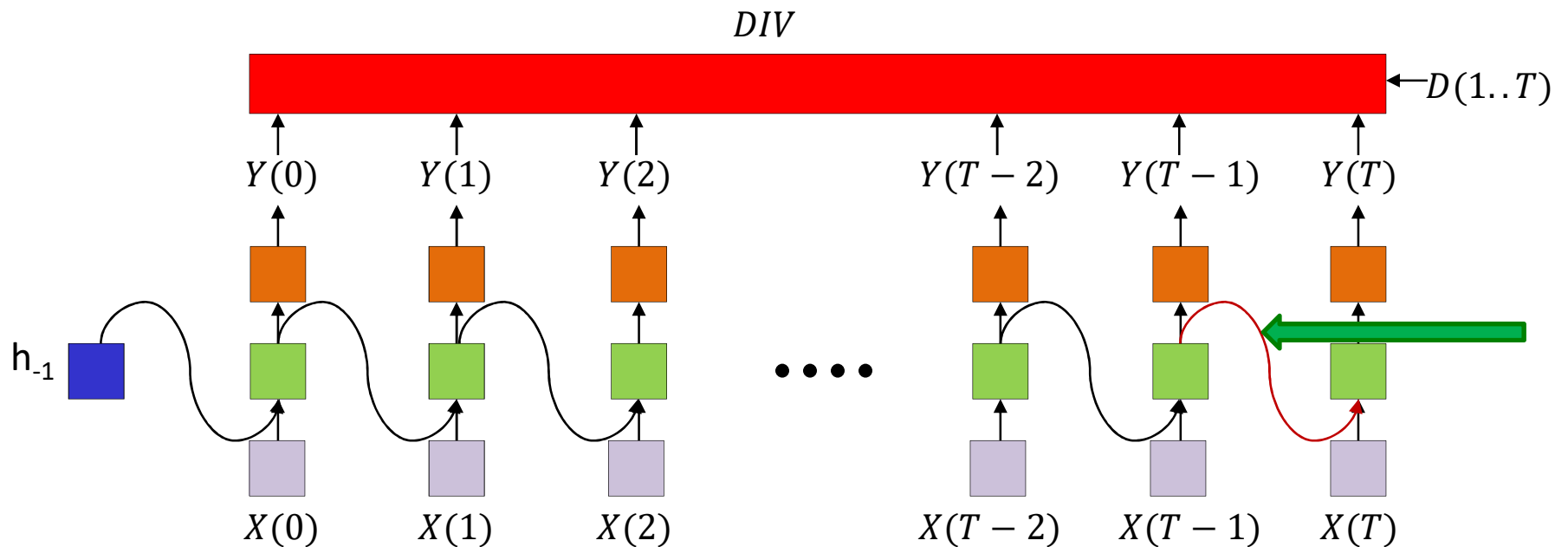
Back Propagation Through Time



$$\nabla_{W^{(0)}} DIV = X(T) \nabla_{Z^{(0)}(T)} DIV$$

$$\frac{dDIV}{dw_{ij}^{(0)}} = \frac{dDIV}{dZ_j^{(0)}(T)} X_i(T)$$

Back Propagation Through Time

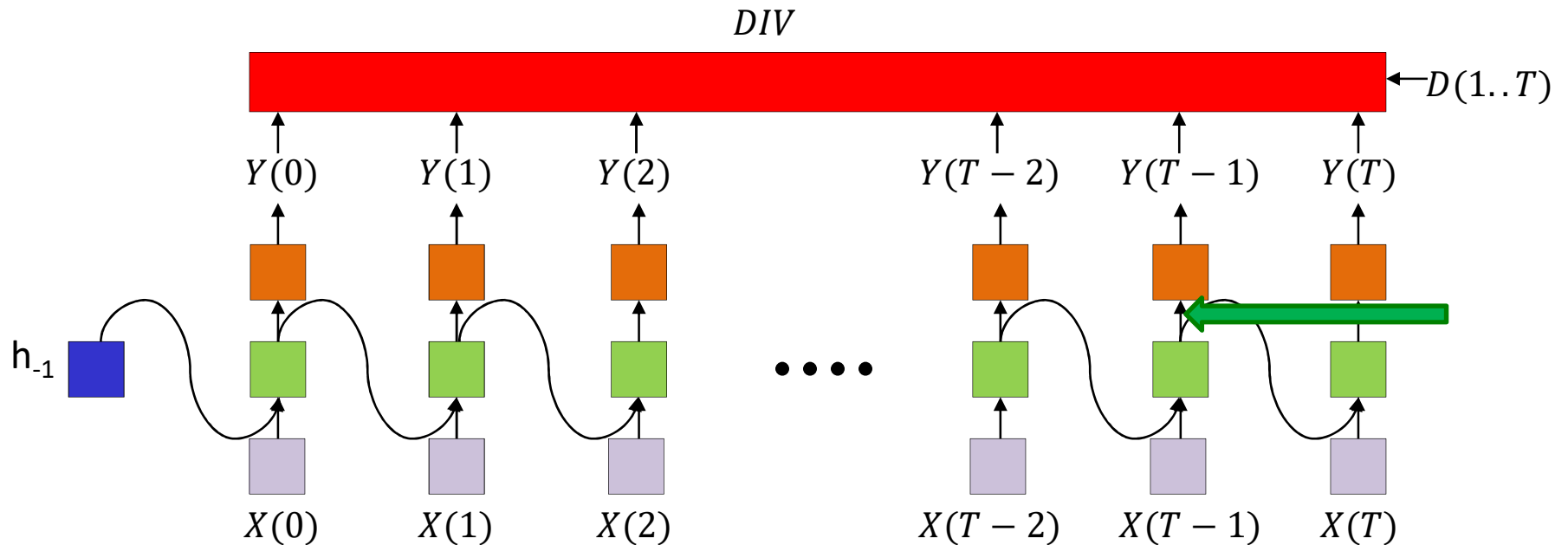


$$\nabla_{W^{(11)}} DIV = h(T-1) \nabla_{Z^{(0)}(T)} DIV$$

$$\frac{dDIV}{dw_{ij}^{(0)}} = \frac{dDIV}{dZ_j^{(0)}(T)} X_i(T)$$

$$\frac{dDIV}{dw_{ij}^{(11)}} = \frac{dDIV}{dZ_j^{(0)}(T)} h_i(T-1)$$

Back Propagation Through Time



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

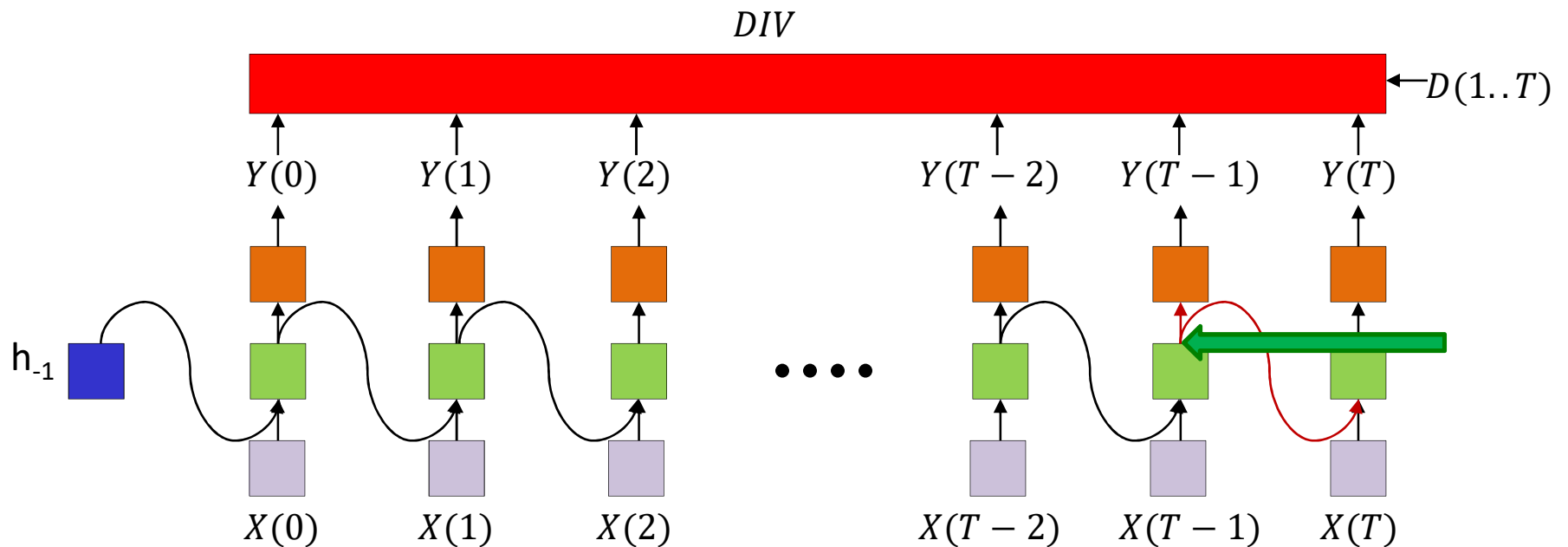
Vector output activation

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

OR

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

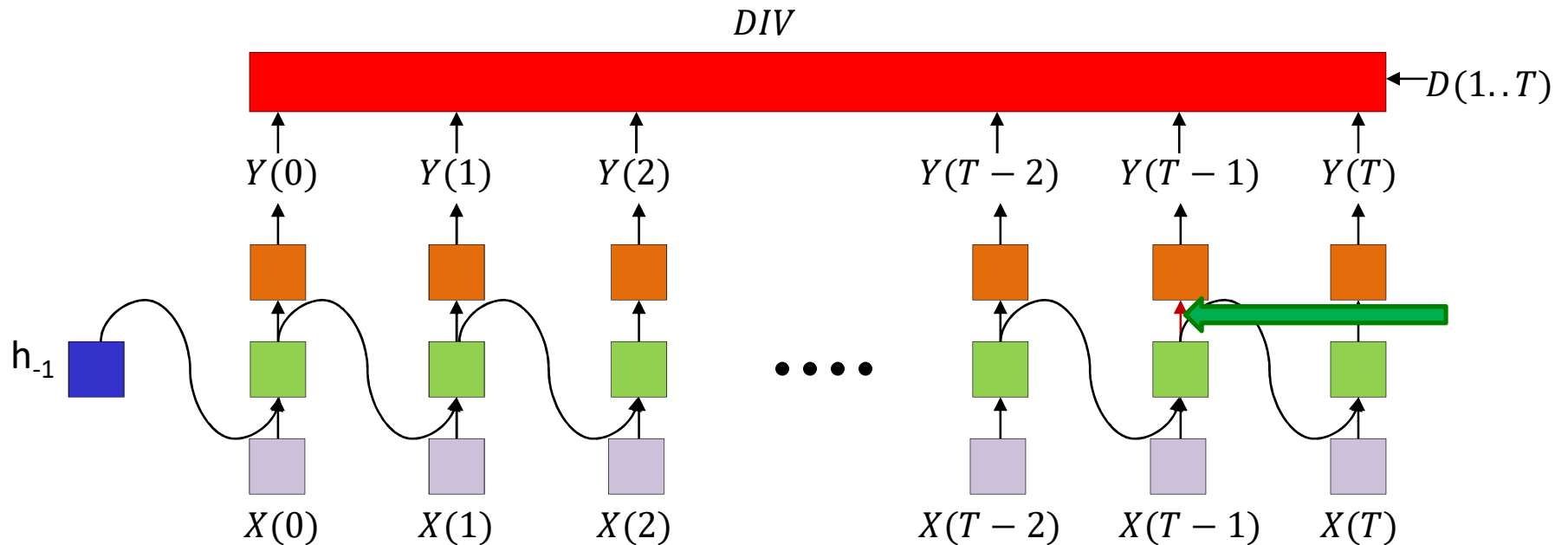
Back Propagation Through Time



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

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

Back Propagation Through Time

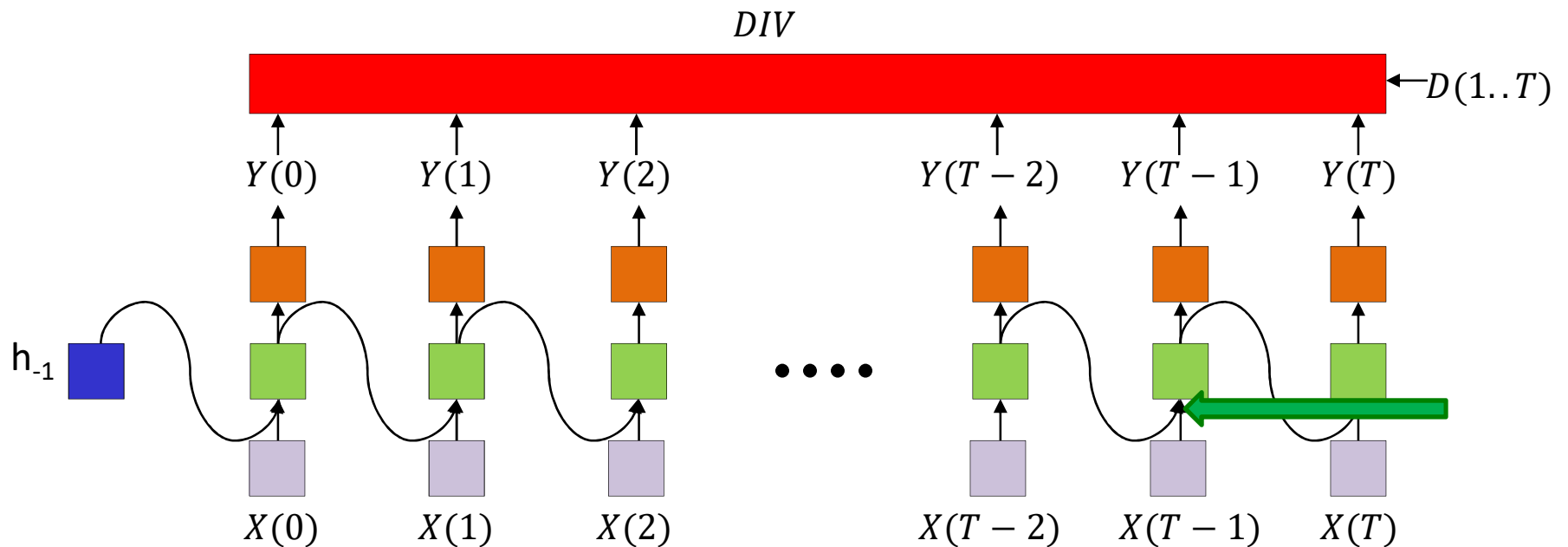


$$\frac{dDIV}{dh_i(T-1)} = \sum_j w_{ij}^{(1)} \frac{dDIV}{dZ_j^{(1)}(T-1)} + \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(0)}(T)}$$

Note the addition \rightarrow $\frac{dDIV}{dw_{ij}^{(1)}} += \frac{dDIV}{dZ_j^{(1)}(T-1)} h_i(T-1)$

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

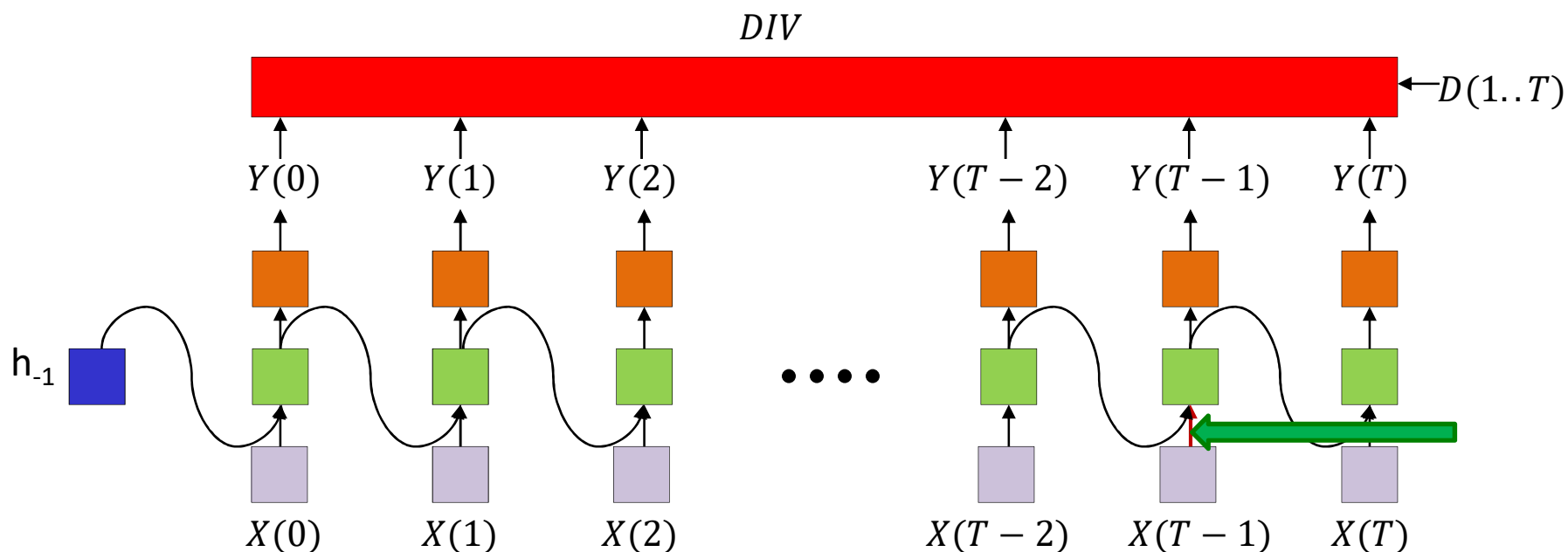
Back Propagation Through Time



$$\frac{dDIV}{dZ_i^{(0)}(T-1)} = \frac{dDIV}{dh_i(T-1)} \frac{dh_i(T-1)}{dZ_i^{(0)}(T-1)}$$

$$\nabla_{Z_i^{(0)}(T-1)} DIV = \nabla_{h(T-1)} DIV \nabla_{Z_i^{(0)}(T-1)} h(T-1)$$

Back Propagation Through Time



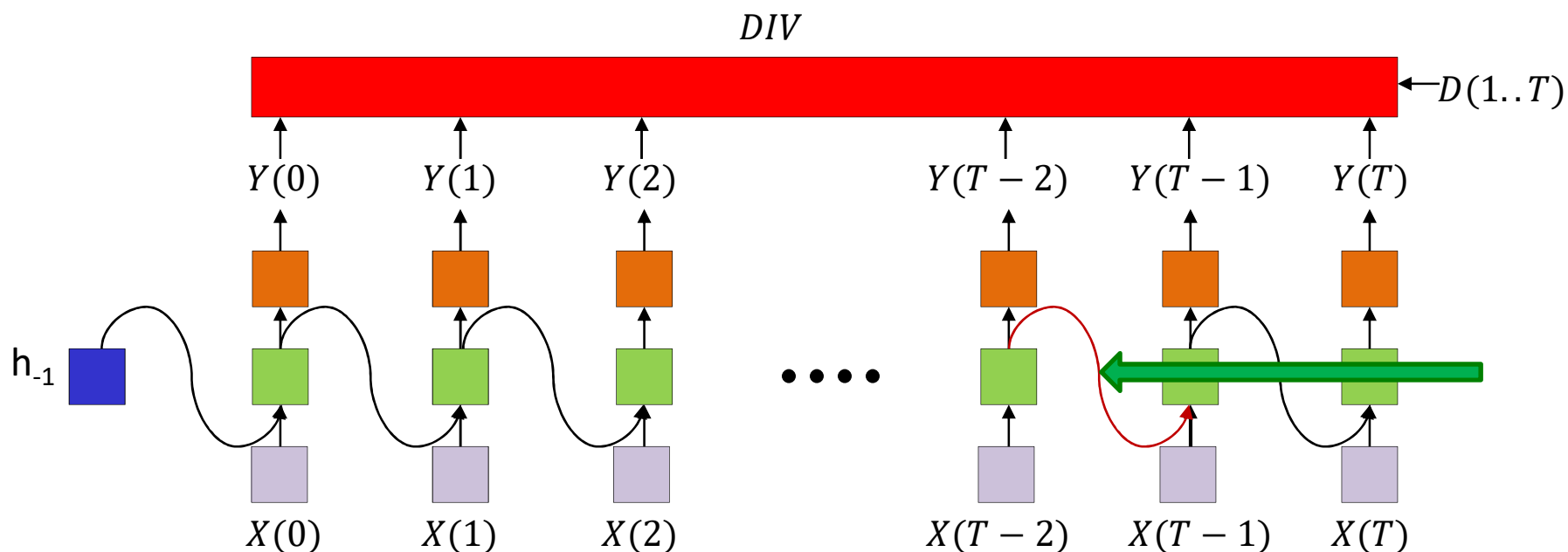
$$\frac{dDIV}{dZ_i^{(0)}(T-1)} = \frac{dDIV}{dh_i(T-1)} \frac{dh_i(T-1)}{dZ_i^{(0)}(T-1)}$$

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

Note the addition

$$\nabla_{W^{(0)}} DIV += X(T-1) \nabla_{Z^{(0)}(T-1)} DIV$$

Back Propagation Through Time



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

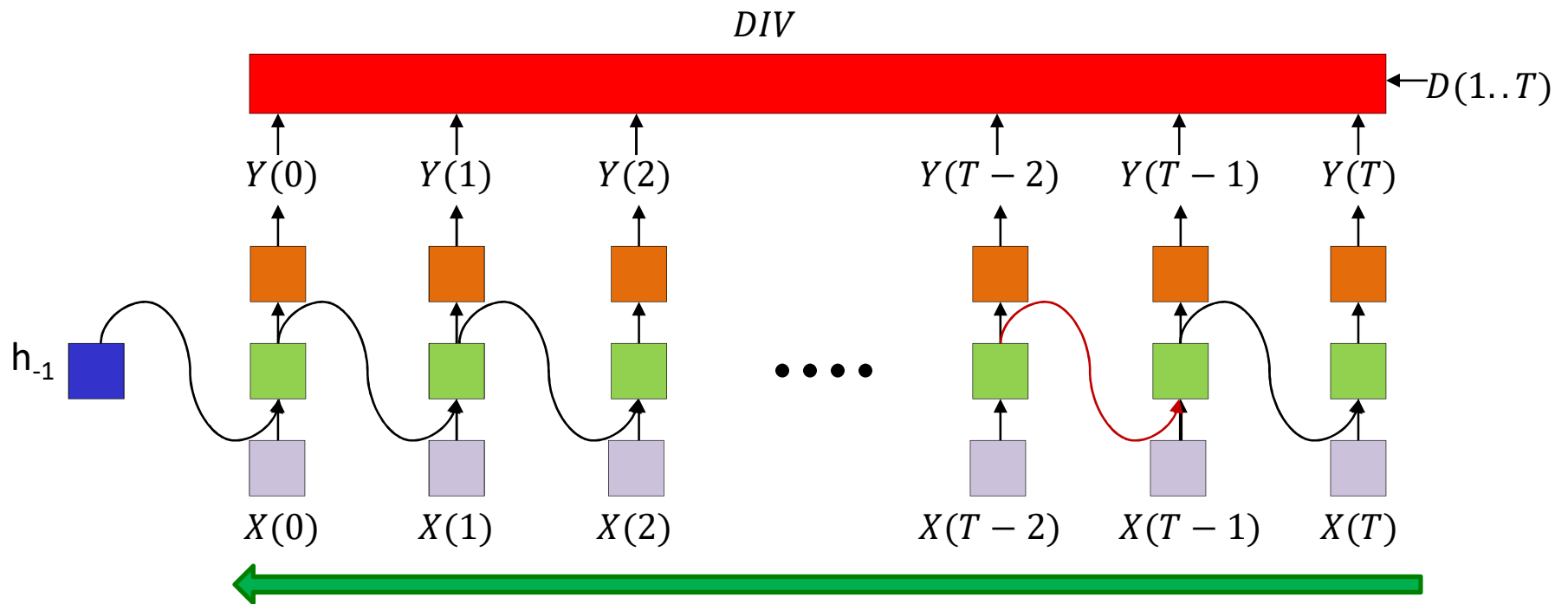
Note the addition



$$\frac{dDIV}{dw_{ij}^{(11)}} += \frac{dDIV}{dZ_j^{(0)}(T-1)} h_i(T-2)$$

$$\nabla_{W^{(11)}} DIV += h(T-2) \nabla_{Z^{(0)}(T-1)} DIV$$

Back Propagation Through Time

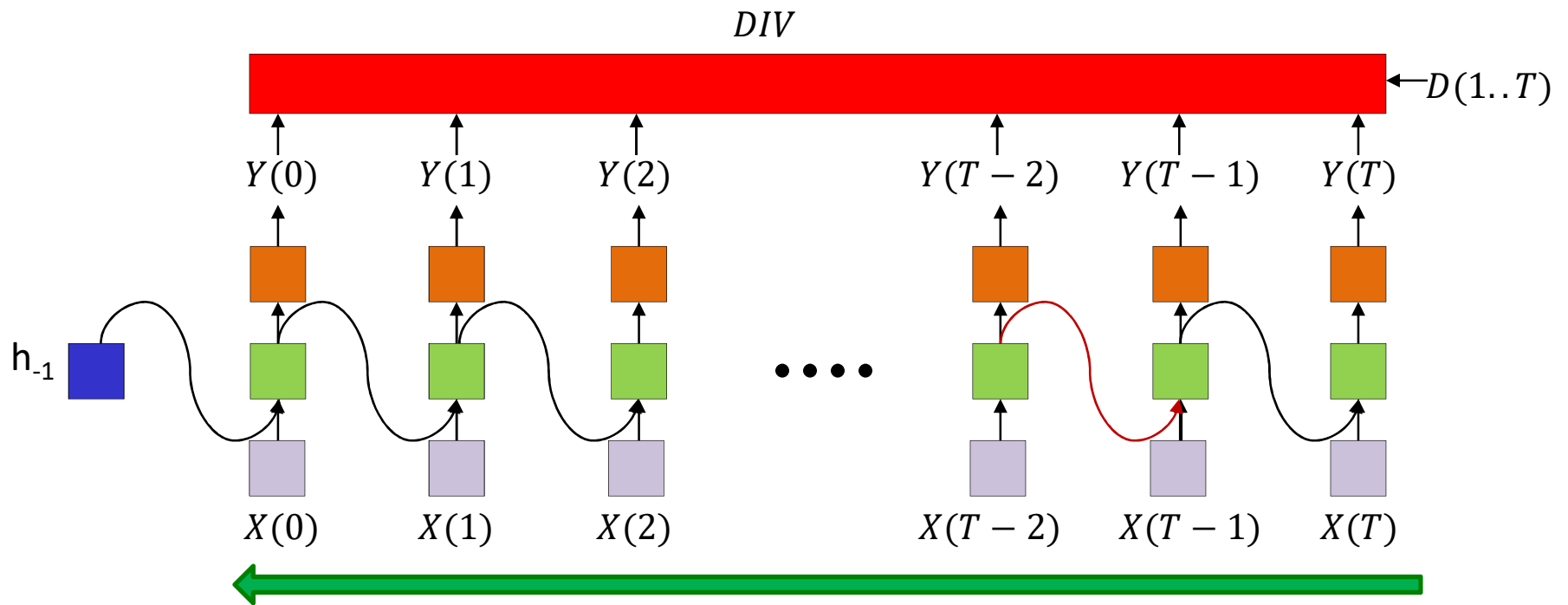


Continue computing derivatives going backward through time until..

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

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

Back Propagation Through Time

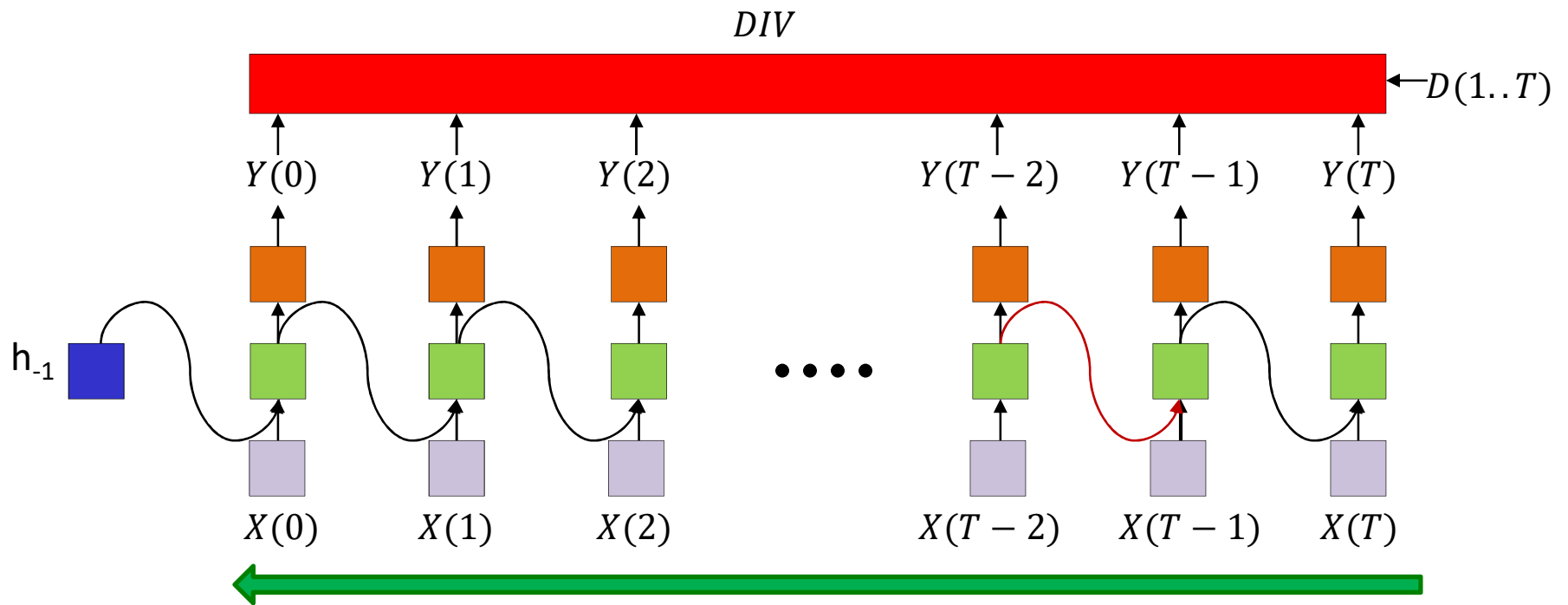


$$\frac{dDIV}{dh_i^{(k)}(t)} = \sum_j w_{i,j}^{(k)} \frac{dDIV}{dZ_j^{(k+1)}(t)} + \sum_j w_{i,j}^{(k,k)} \frac{dDIV}{dZ_j^{(k)}(t+1)}$$

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



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

$$\frac{dDIV}{dw_{ij}^{(0)}} = \sum_t \frac{dDIV}{dZ_j^{(0)}(t)} X_i(t)$$

$$\frac{dDIV}{dw_{ij}^{(11)}} = \sum_t \frac{dDIV}{dZ_j^{(0)}(t)} h_i(t-1)$$

BPTT

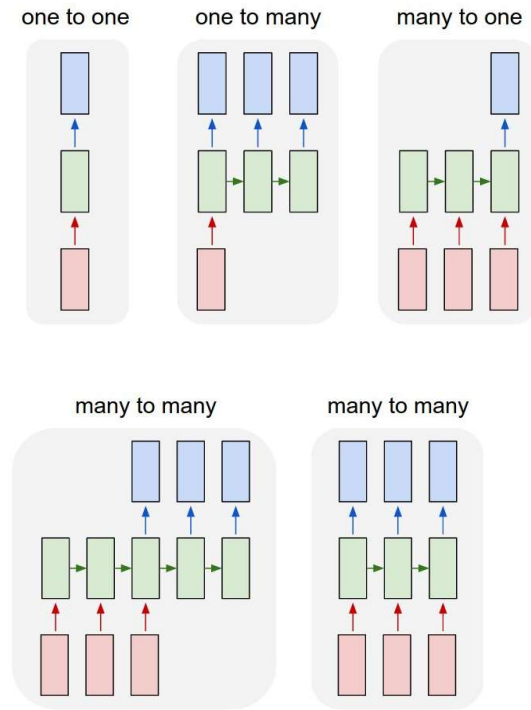
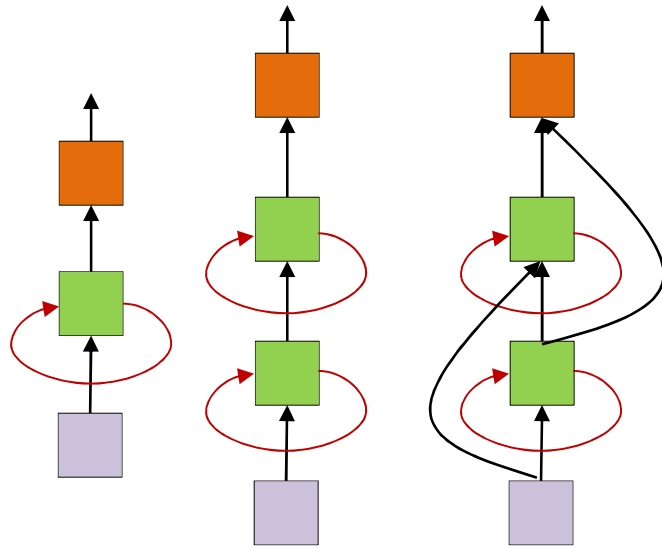
```
# Assuming forward pass has been completed
# Jacobian(x,y) is the jacobian of x w.r.t. y
# Assuming dY(t) = gradient(div,Y(t)) available for all t
# Assuming all dz, dh, dW and db are initialized to 0
```

```
for t = T-1:downto:0 # Backward through time
    dzo(t) = dY(t)Jacobian(Y(t),zo(t))
    dWo += h(t,L)dzo(t)
    db(L) += dzo(t)
    dh(t,L) += dzo(t)Wo
```

```
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)Wc(l)
    dh(t-1,l) += dz(t,l)Wr(l)

    dWc(l) += h(t,l-1)dz(t,l)
    dWr(l) += h(t-1,l)dz(t,l)
    db(l) += dz(t,l)
```

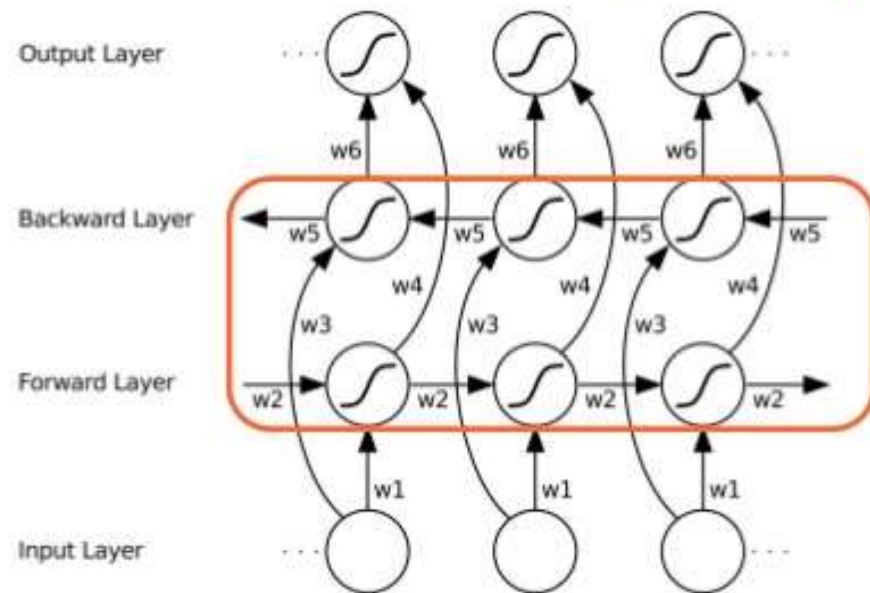
BPTT



- Can be generalized to any architecture

Extensions to the RNN: *Bidirectional RNN*

Bidirectional RNN (BRNN)



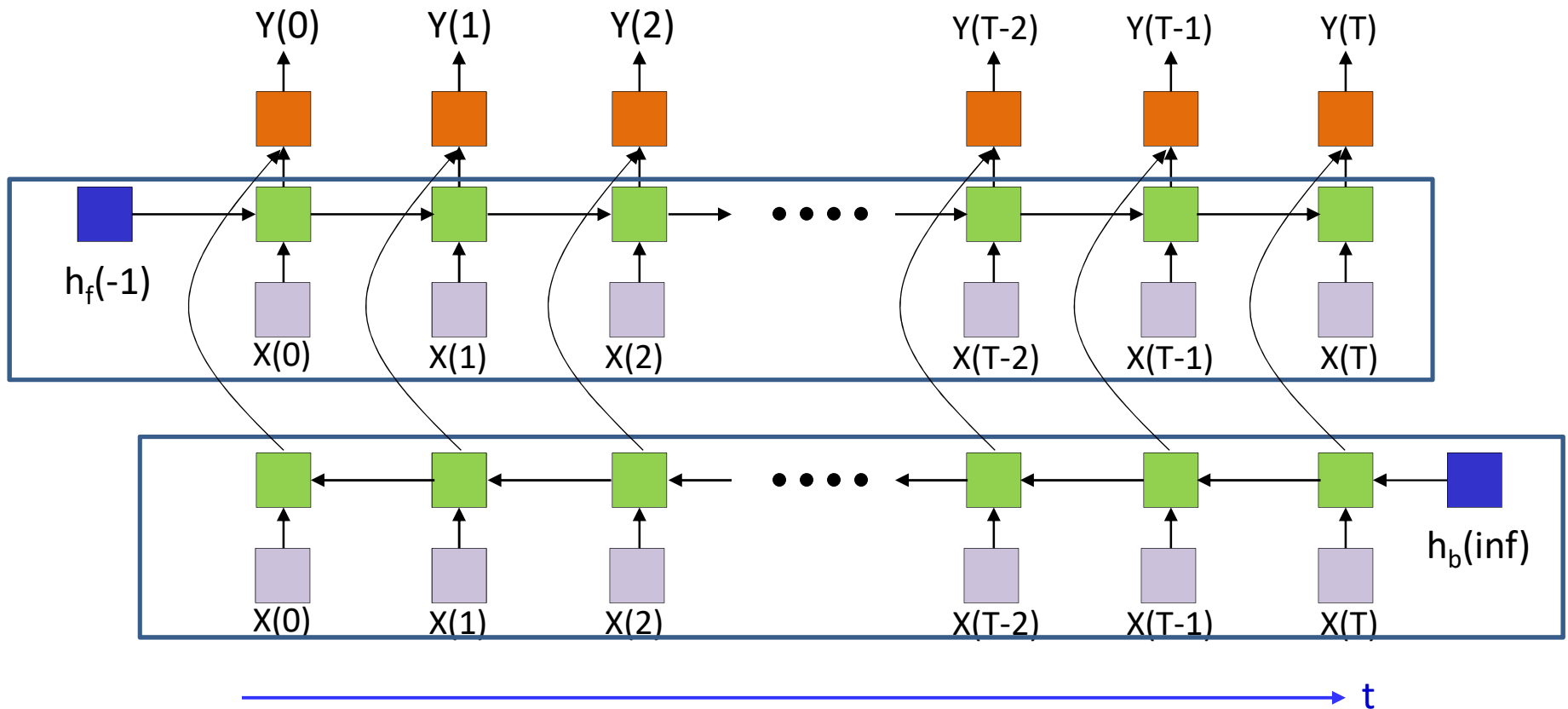
Must learn weights w_2 , w_3 , w_4 & w_5 ; in addition to w_1 & w_6 .

Proposed by Schuster and Paliwal
1997

Alex Graves, ["Supervised Sequence Labelling with Recurrent Neural Networks"](#)

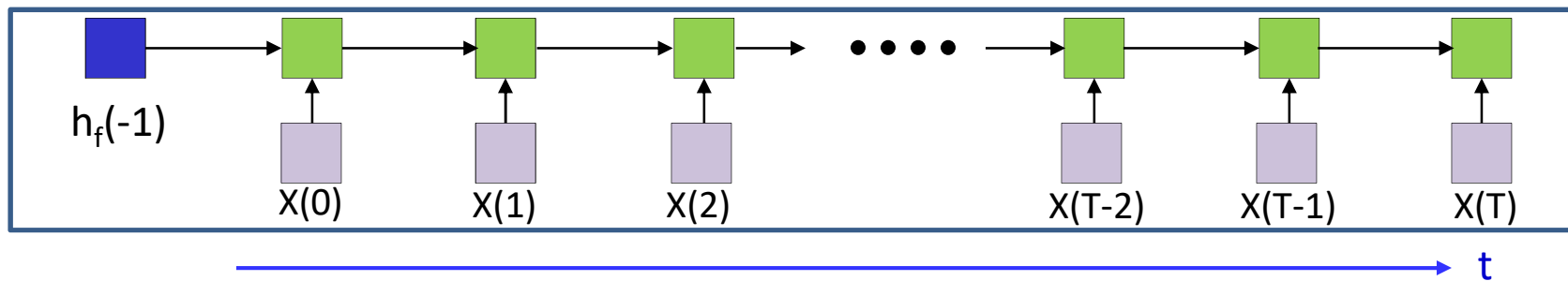
- RNN with both forward and backward recursion
 - Explicitly models the fact that just as the future can be predicted from the past, the past can be deduced from the future

Bidirectional RNN



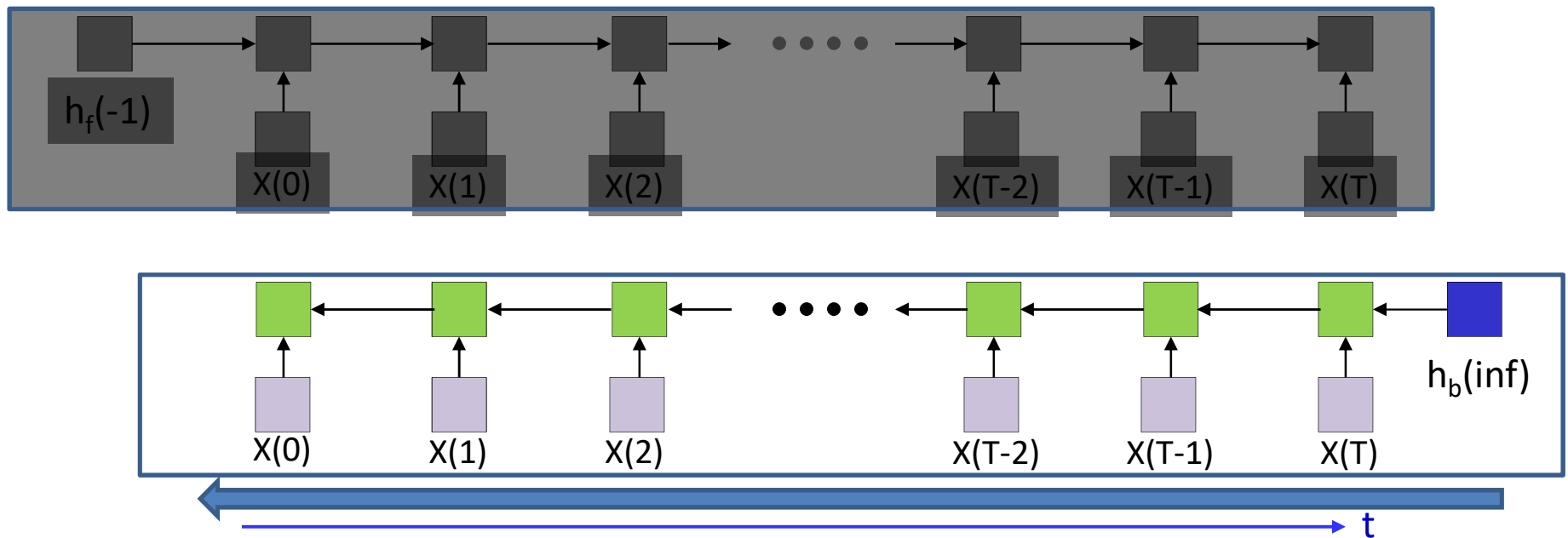
- A forward net process the data from $t=0$ to $t=T$
- A backward net processes it backward from $t=T$ down to $t=0$

Bidirectional RNN: Processing an input string



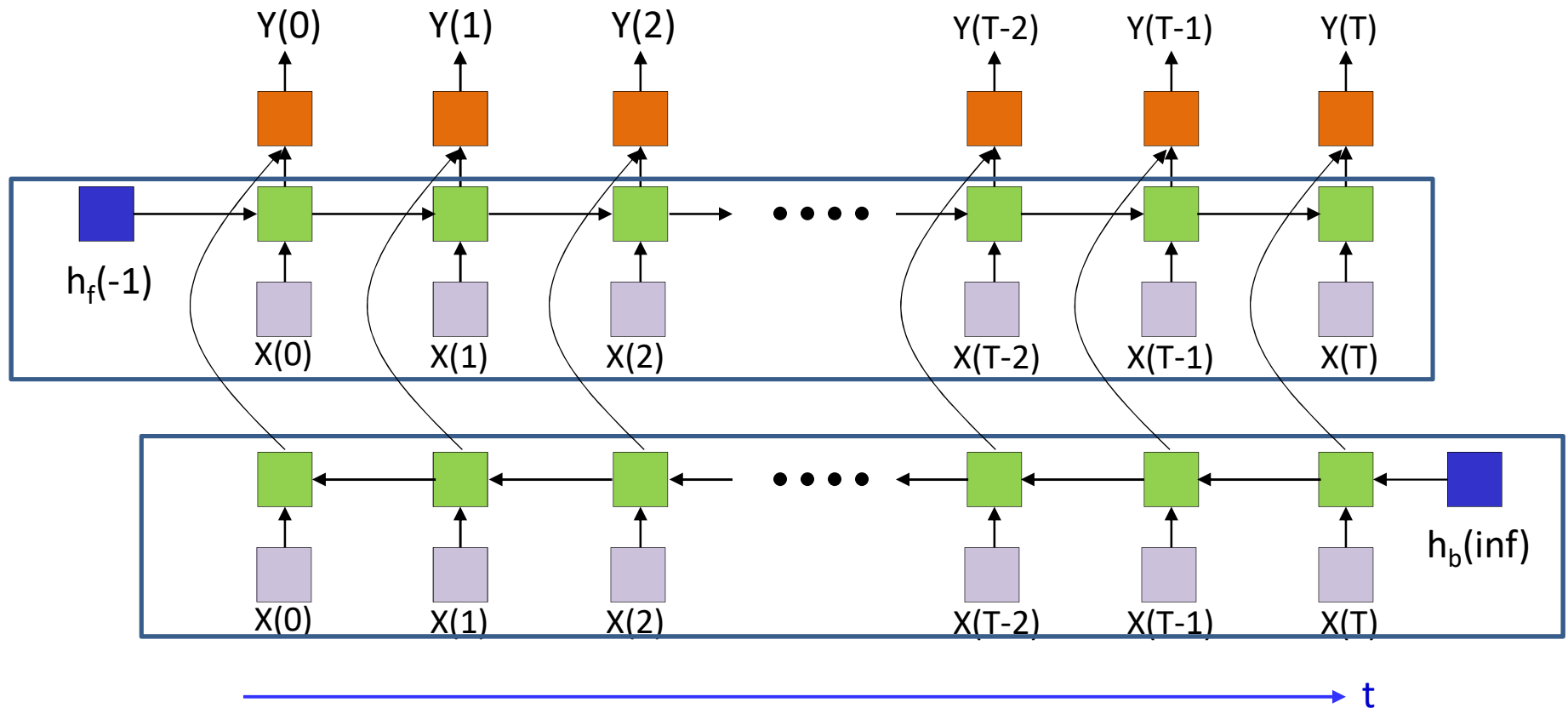
- The forward net process the data from $t=0$ to $t=T$
 - Only computing the hidden states, initially

Bidirectional RNN: Processing an input string



- 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: Processing an input string



- The computed states of both networks are used to compute the final output at each time

Bidirectional RNN

Assuming time-synchronous output

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

#forward pass
for t = 0:T-1 # Going forward in time
     $h_f(t,0) = x(t)$  # Vectors. Initialize  $h(0)$  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
 $h(T, :, :) = h(\text{inf}, :, :)$  # Just the initial value
for t = T-1:downto:0 # Going backward in time
     $h_b(t,0) = x(t)$  # Vectors. Initialize  $h(0)$  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
     $z_o(t) = W_{fo}h_f(t, L_f) + W_{bo}h_b(t, L_b) + b_o$ 
     $Y(t) = \text{softmax}(z_o(t))$ 
```

Bidirectional RNN: Simplified code

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

First: Define basic RNN with only hidden units

```
# 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 [h,z] = 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.
        end
    end
    return h,z
```

Bidirectional RNN

Assuming time-synchronous output

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

```
#forward pass
```

```
[ $h_f$ ,  $z_f$ ] = RNN_forward( $L_f$ ,  $W_{fc}$ ,  $W_{fr}$ ,  $b_f$ ,  $h(-1, :)$ ,  $x$ , T)
```

```
#backward pass
```

```
 $x_{\text{rev}} = \text{fliplr}(x)$  # Flip it in time
```

```
[ $h_{\text{brev}}$ ,  $z_{\text{brev}}$ ] = RNN_forward( $L_b$ ,  $W_{bc}$ ,  $W_{br}$ ,  $b_b$ ,  $h(\text{inf}, :)$ ,  $x_{\text{rev}}$ , T)
```

```
 $h_b = \text{fliplr}(h_{\text{brev}})$  # Flip back to straighten time
```

```
 $z_b = \text{fliplr}(z_{\text{brev}})$ 
```

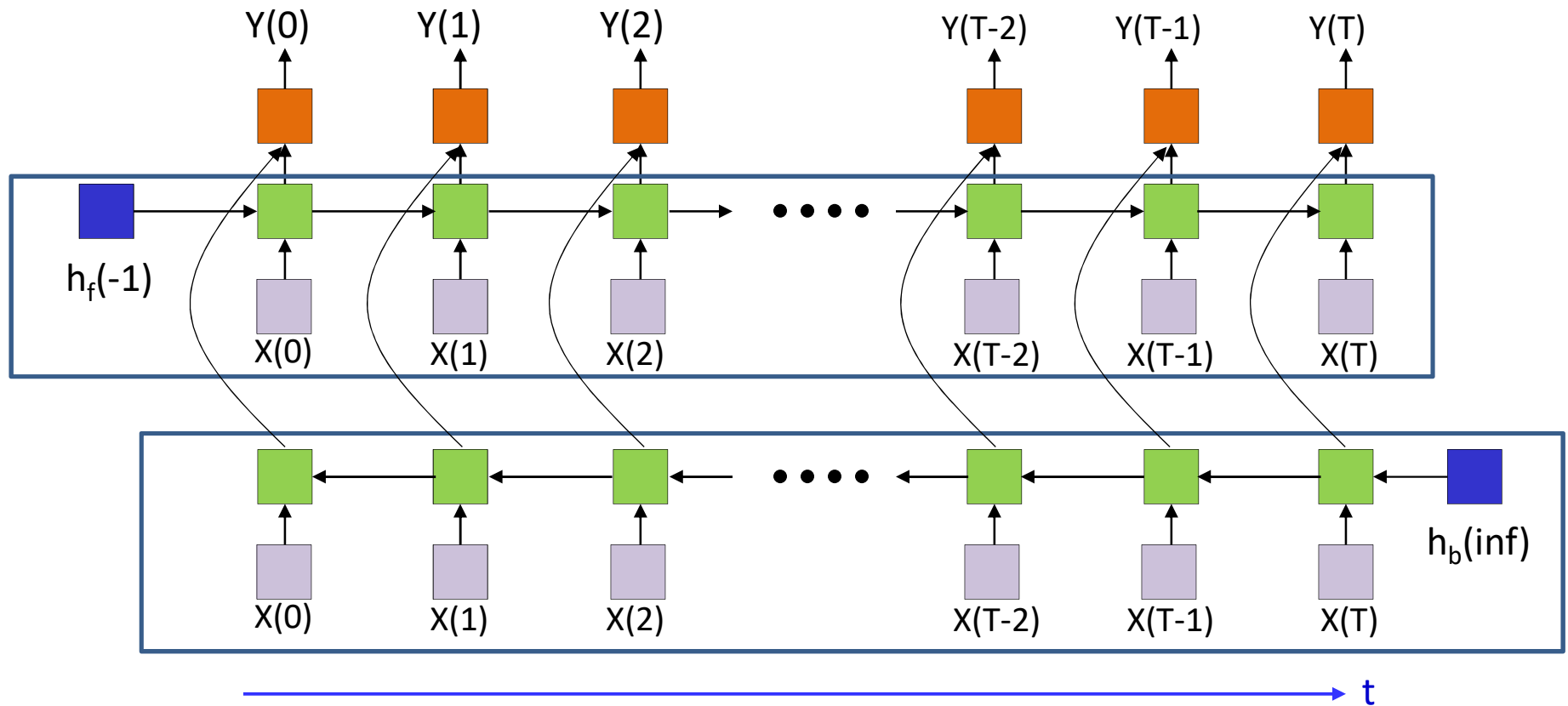
```
#combine the two for the output
```

```
for t = 0:T-1 # The output combines forward and backward
```

```
     $z_o(t) = W_{fo}h_f(t, L_f) + W_{bo}h_b(t, L_b) + b_o$ 
```

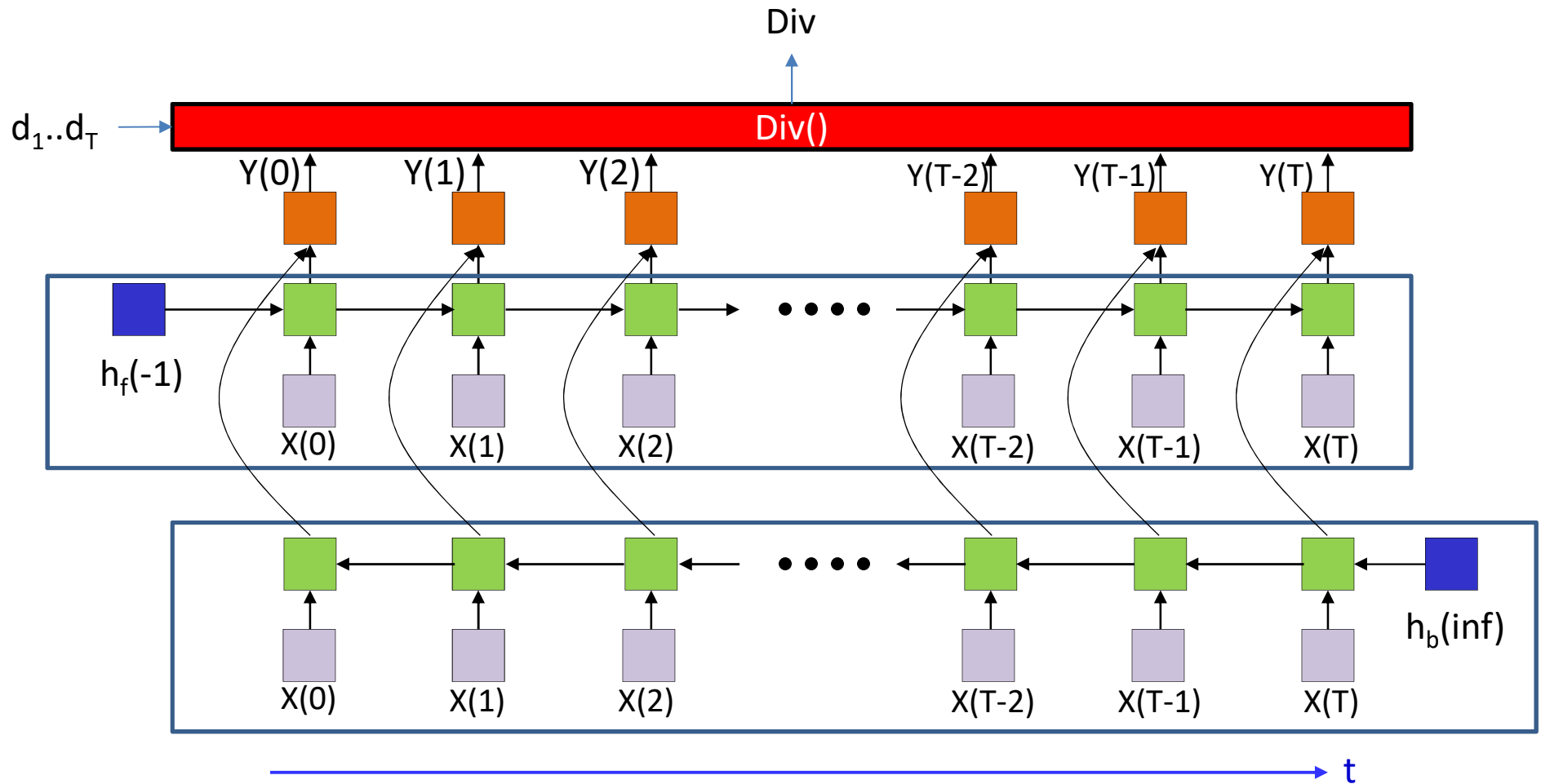
```
     $Y(t) = \text{softmax}(z_o(t))$ 
```

Backpropagation in BRNNs



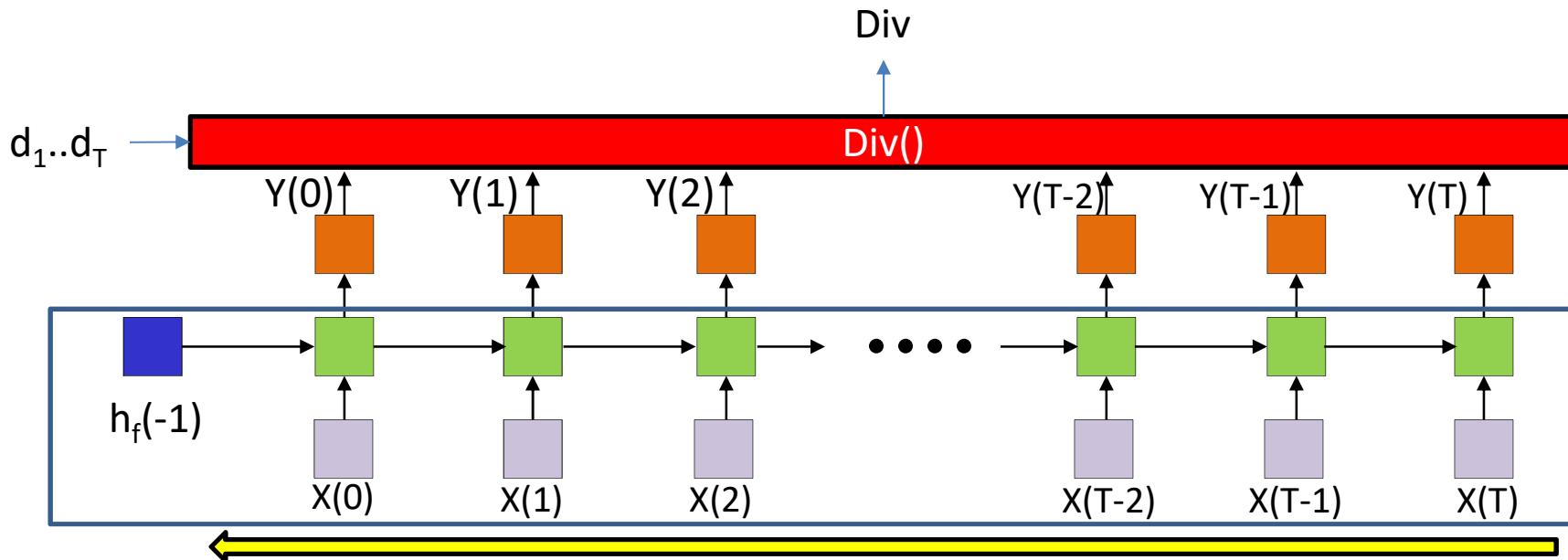
- Forward pass: Compute both forward and backward networks and final output

Backpropagation in BRNNs



- Backward pass: Define a divergence from the desired output

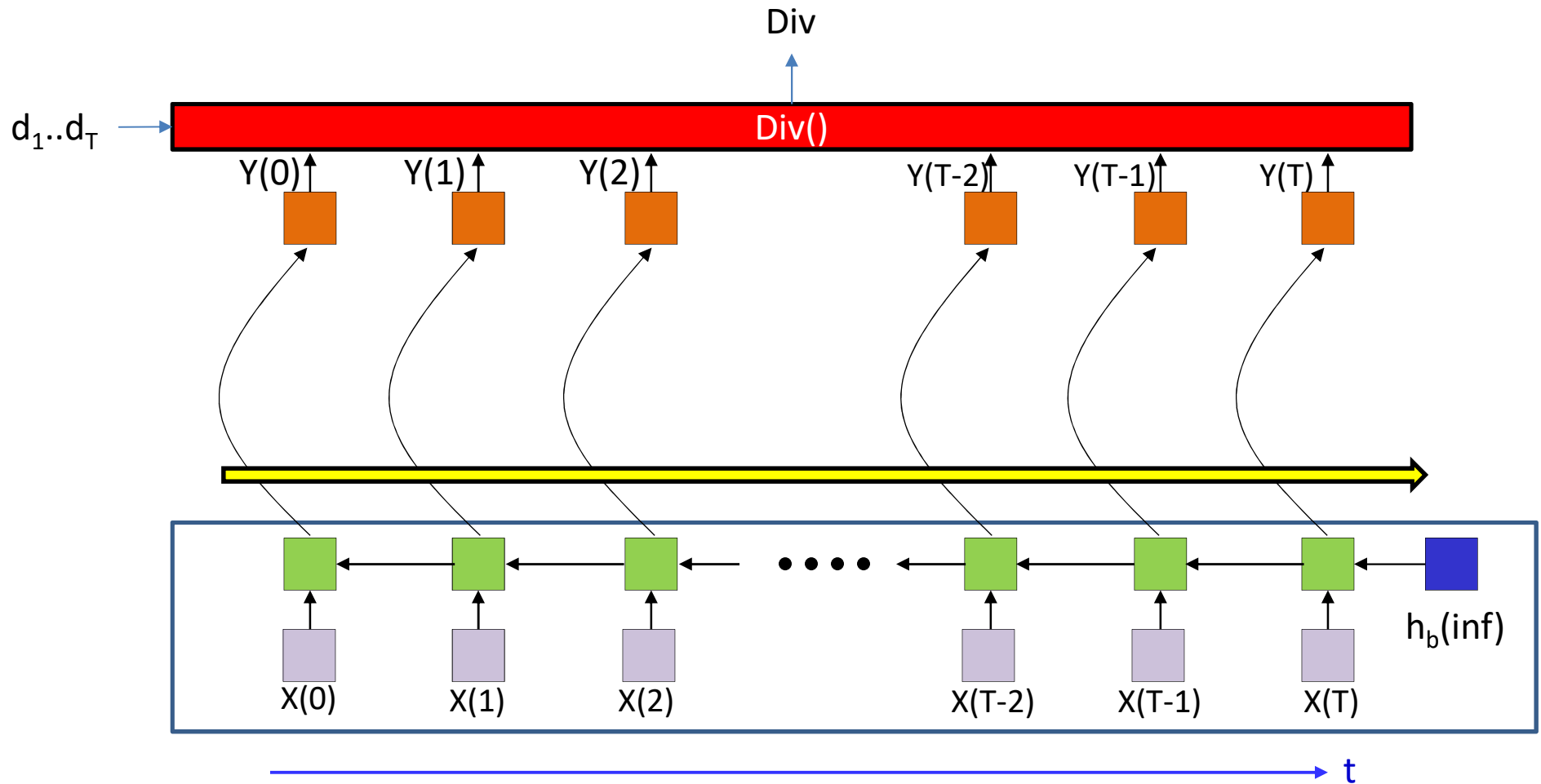
Backpropagation in BRNNs



- Backward pass: Define a divergence from the desired output
- Separately perform back propagation on both nets
 - **From $t=T$ down to $t=0$ for the forward net**

t

Backpropagation in BRNNs



- Backward pass: Define a divergence from the desired output
- Separately perform back propagation on both nets
 - From $t=T$ down to $t=0$ for the forward net
 - **From $t=0$ up to $t=T$ for the backward net**

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_r, 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 [dW_c,dW_r,db,dh_init] = 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)
        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)

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

Bi-RNN gradient computation

Assuming time-synchronous output

```
# Subscript f represents forward net, b is backward net
# First compute derivatives that directly relate to dY(t) for all t,
# then pass the derivatives into RNN_bpptt to compute forward and backward
# parameter derivatives

for t = 0:T-1 # The output combines forward and backward
    dz_o(t) = dY(t)Jacobian(Y(t), z_o(t))
    dh_fo(t) = dz_o(t)W_fo
    dh_bo(t) = dz_o(t)W_bo
    db_o += dz_o(t)
    dW_fo += h_f(t,L) dz_o(t)
    dW_bo += h_b(t,L) dz_o(t)

#forward net
[dW_fc, dW_fr, db_f, dh_f(-1)] = RNN_bpptt(L, W_fc, W_fr, b_f, h_f(-1), x, T, dh_fo, h_f, z_f)
#backward net
x_rev = fliplr(x) # Flip it in time
[dW_bc, dW_br, db_b, dh_b(inf)] = RNN_bpptt(L, W_bc, W_br, b_b, h_b(inf), x_rev, T, dh_bo, h_b, z_b)
```

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 time-series analysis tasks
 - Time-series prediction
 - Time-series classification
 - Sequence prediction..

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