Deep Learning
Recurrent Networks

10/11/2017

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 $\mathcal{F} \to \mathcal{F}$ 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

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

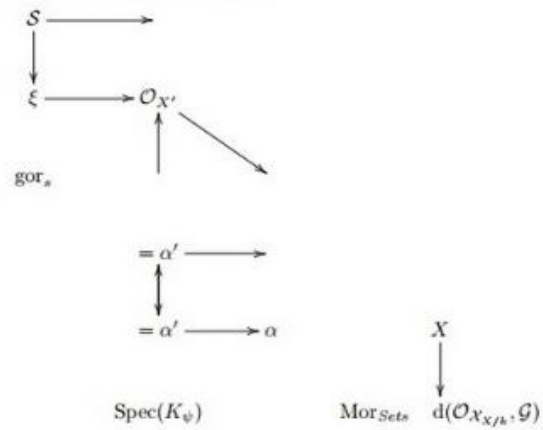
be a morphism of algebraic spaces over S and Y.

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

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

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

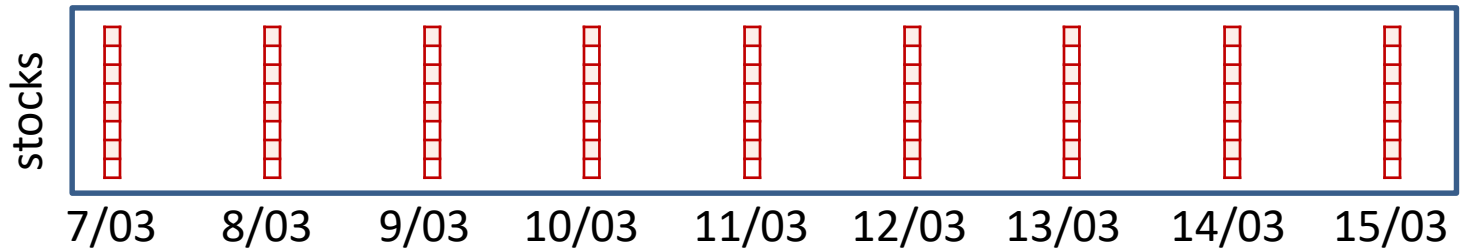
Modelling Series

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

- Examples: ..

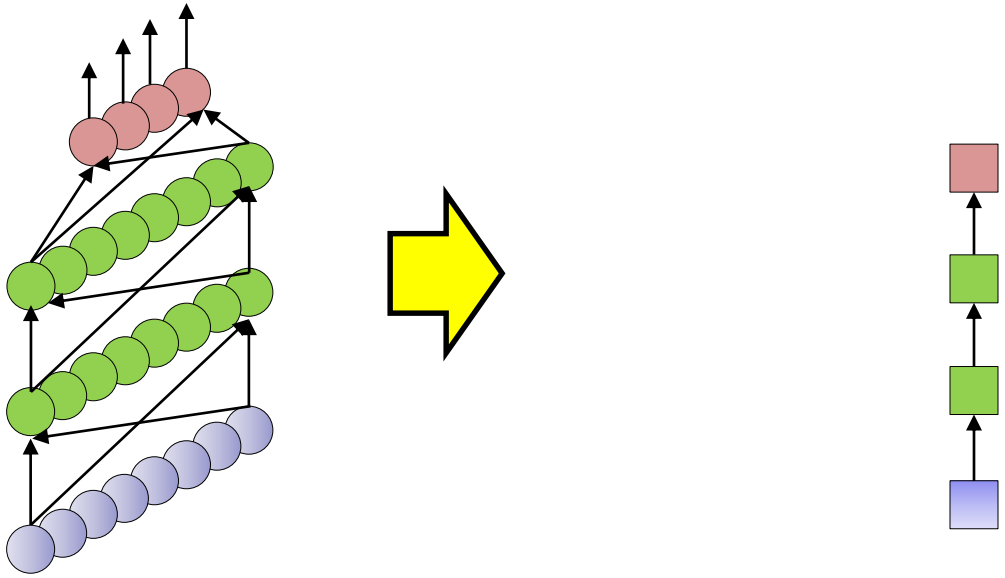
Should I invest..

To invest or not to invest?



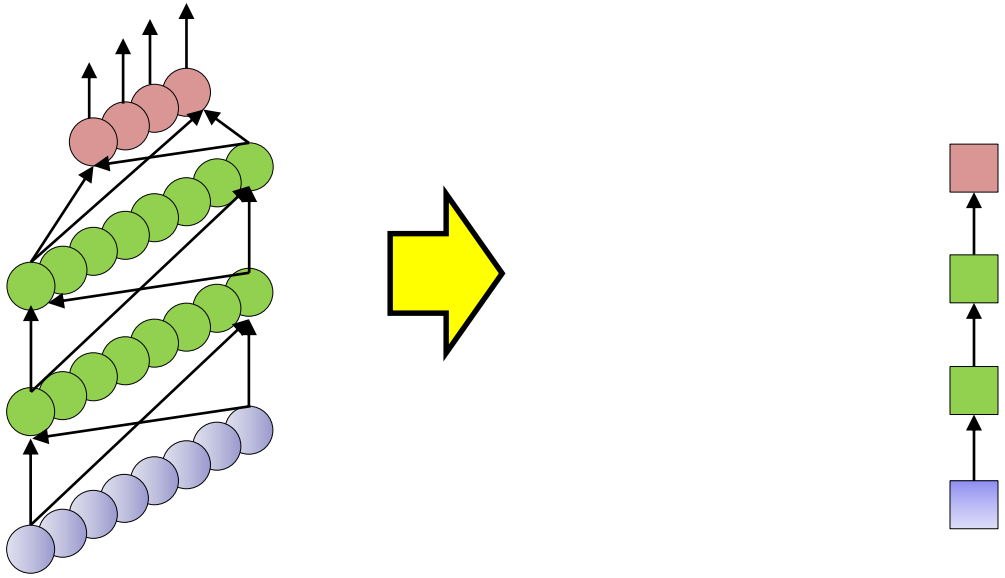
- Stock market
  - Must consider the series of stock values in the past several days to decide if it is wise to invest today
    - Ideally consider *all* of history
- Note: Inputs are vectors. Output may be scalar or vector
  - Should I invest, vs. should I invest in X

Representational shortcut



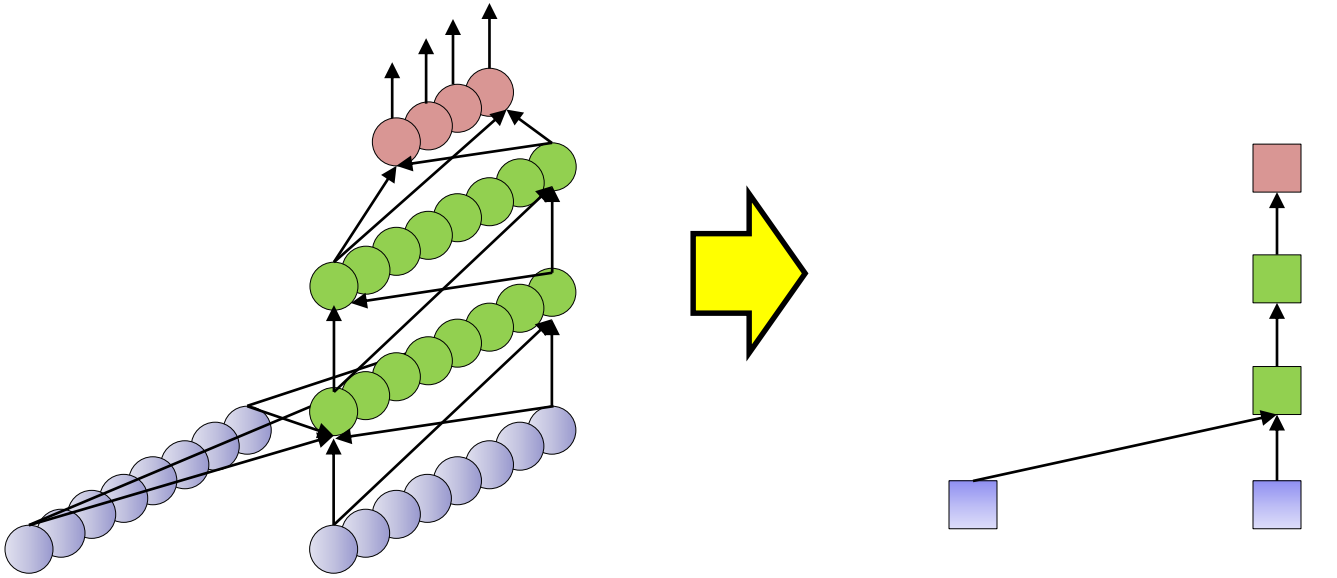
- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything simple boxes
  - Each box actually represents an entire *layer with many units*

Representational shortcut



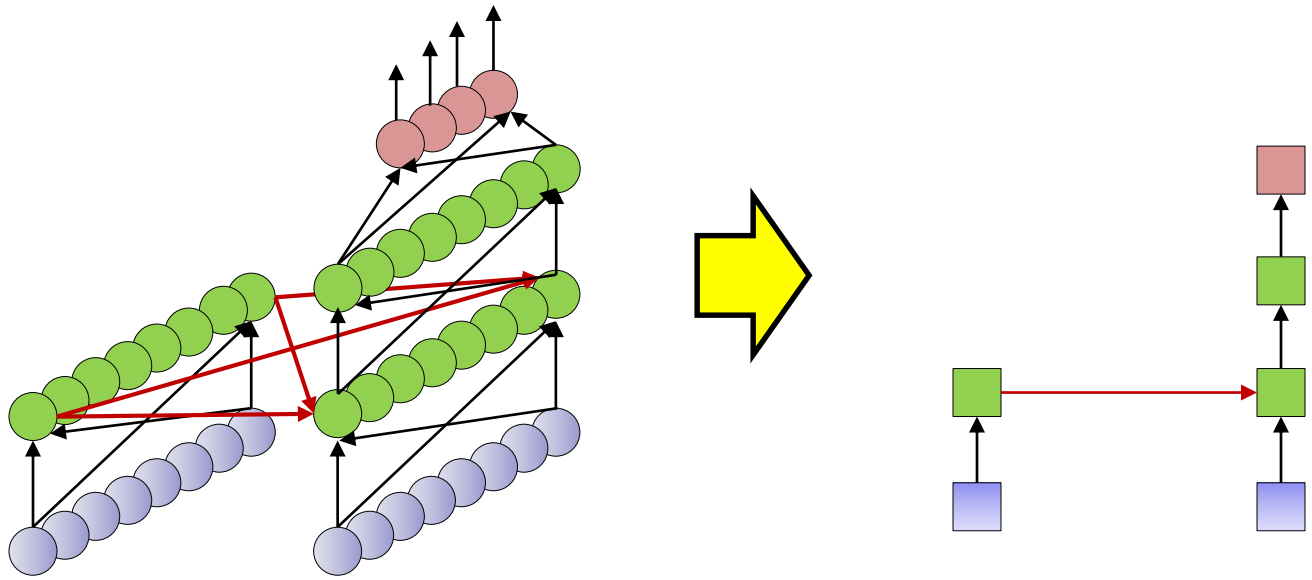
- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything simple boxes
  - Each box actually represents an entire *layer with many units*

Representational shortcut



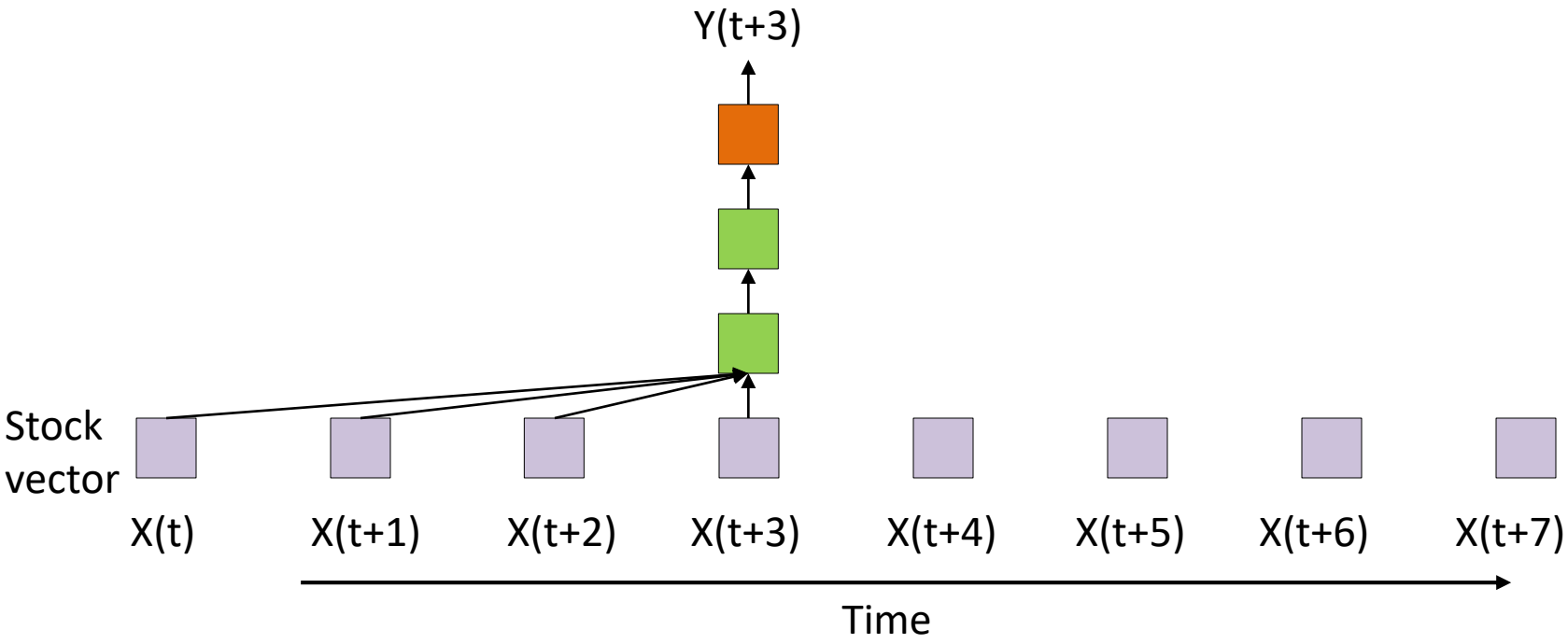
- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything simple boxes
  - Each box actually represents an entire *layer with many units*

Representational shortcut



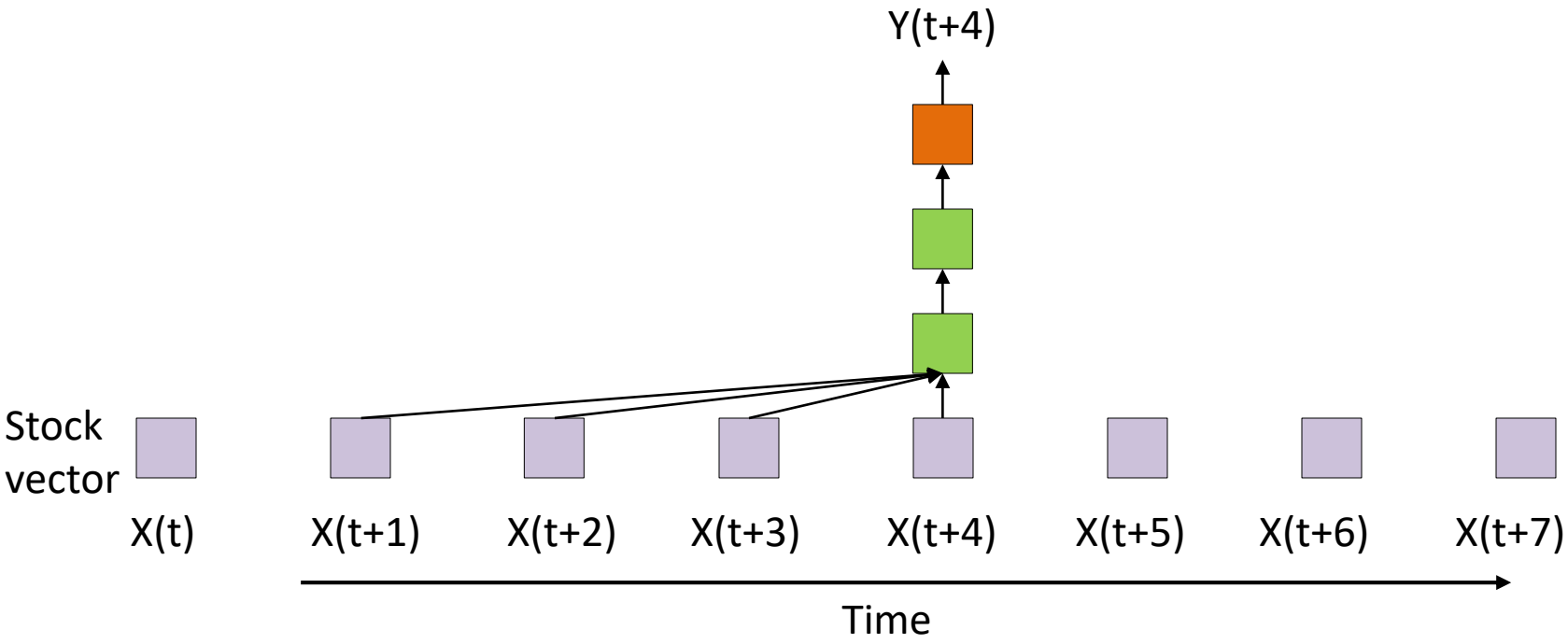
- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything simple boxes
  - Each box actually represents an entire *layer with many units*

The stock predictor



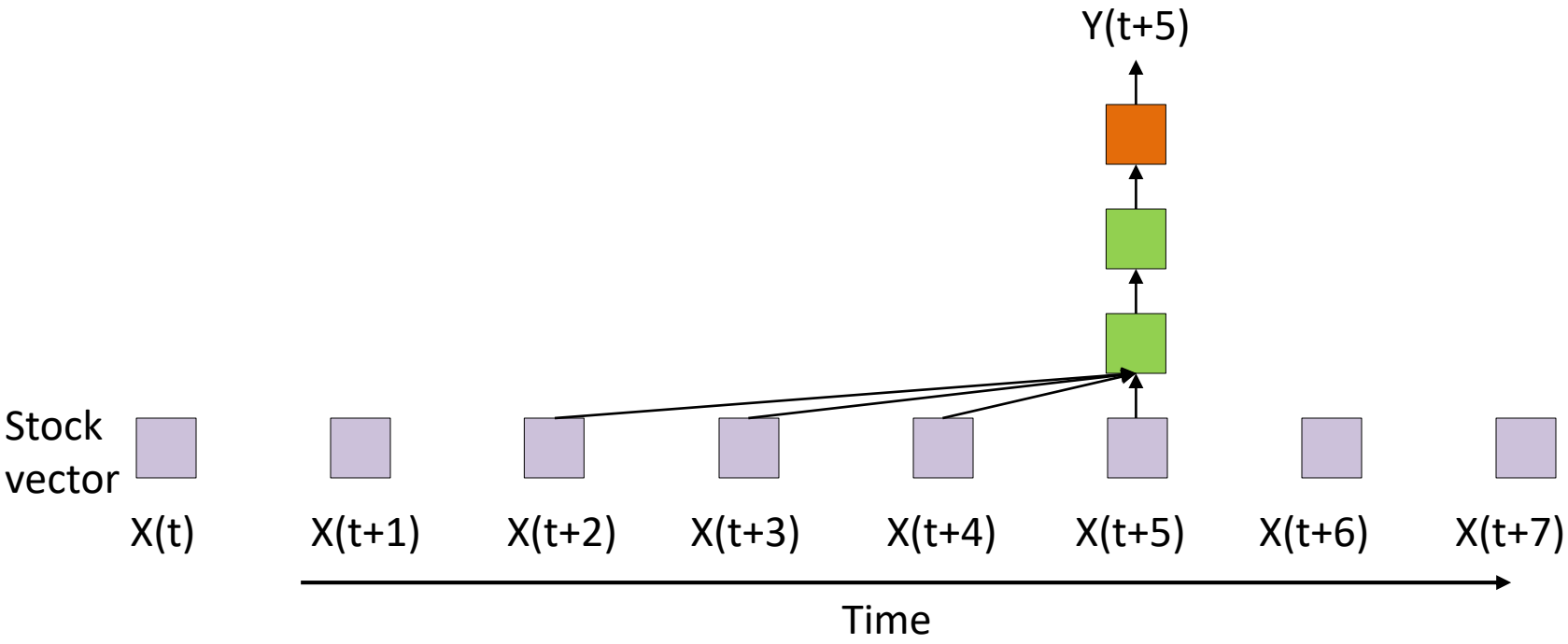
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

The stock predictor



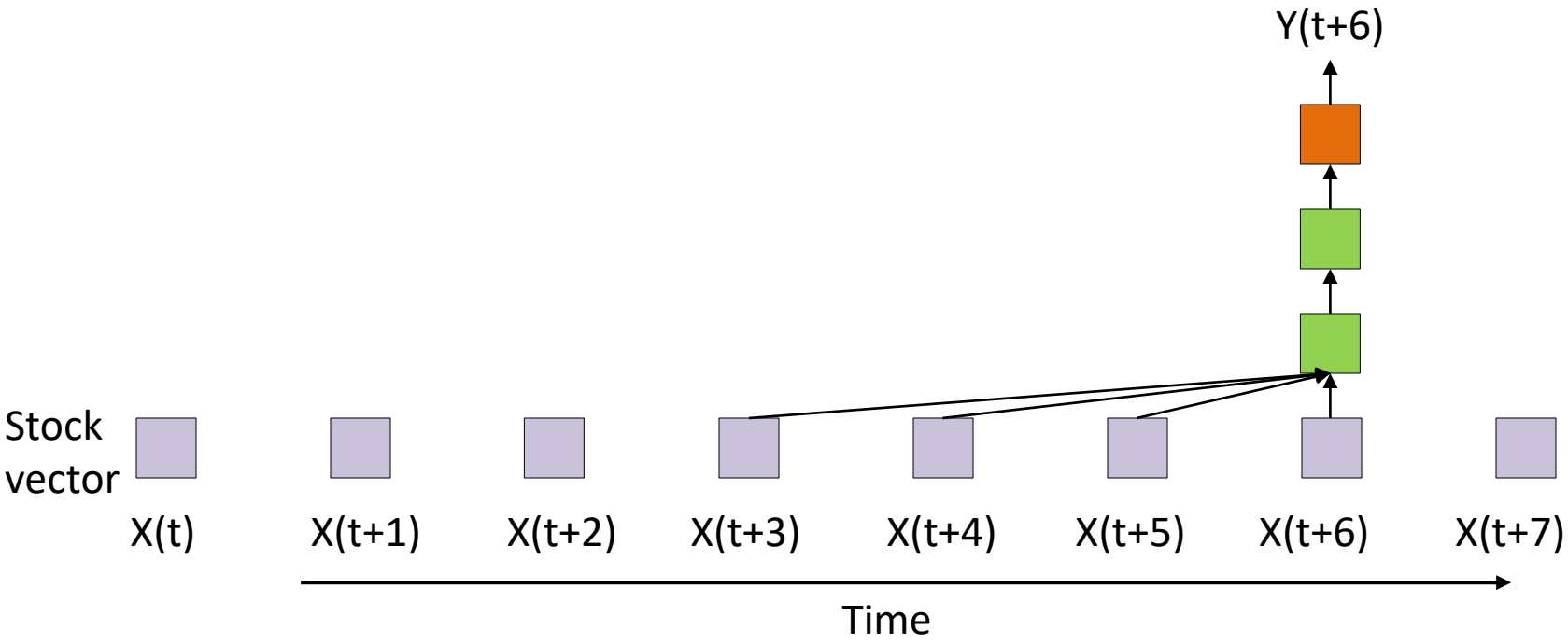
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

The stock predictor



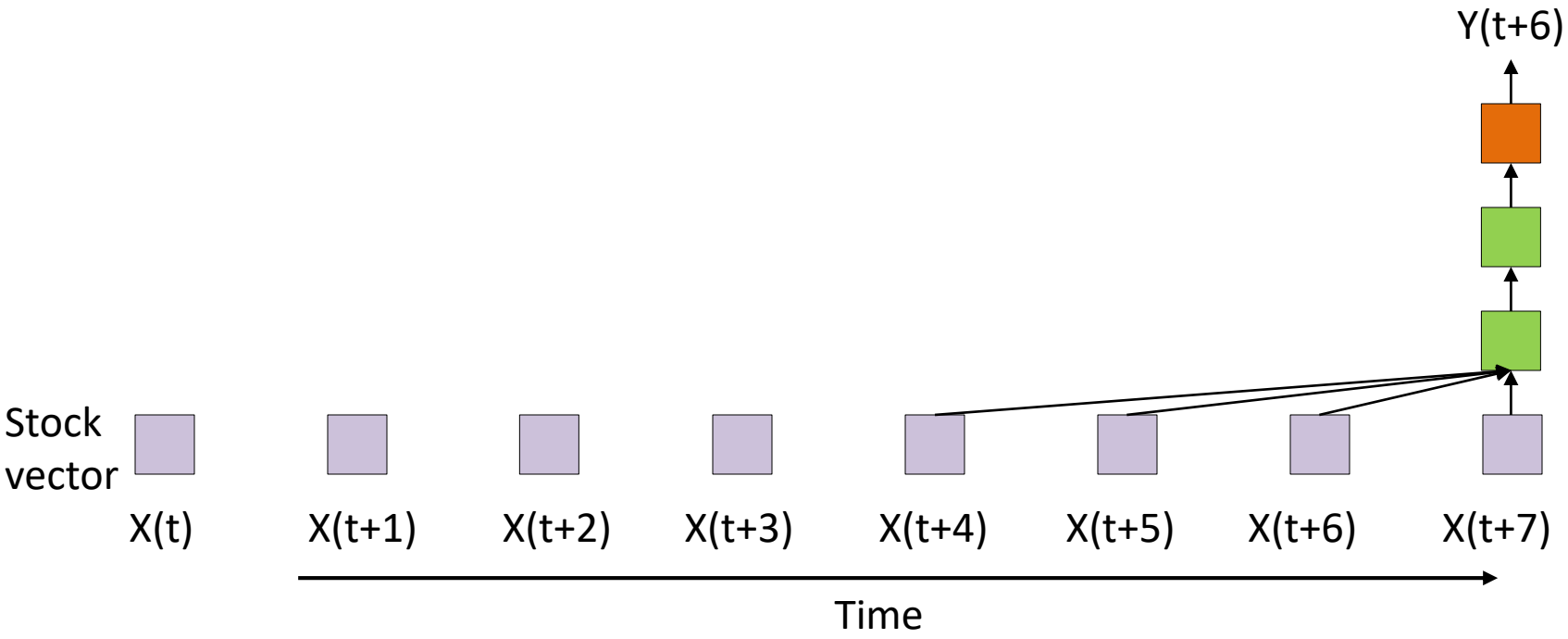
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

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



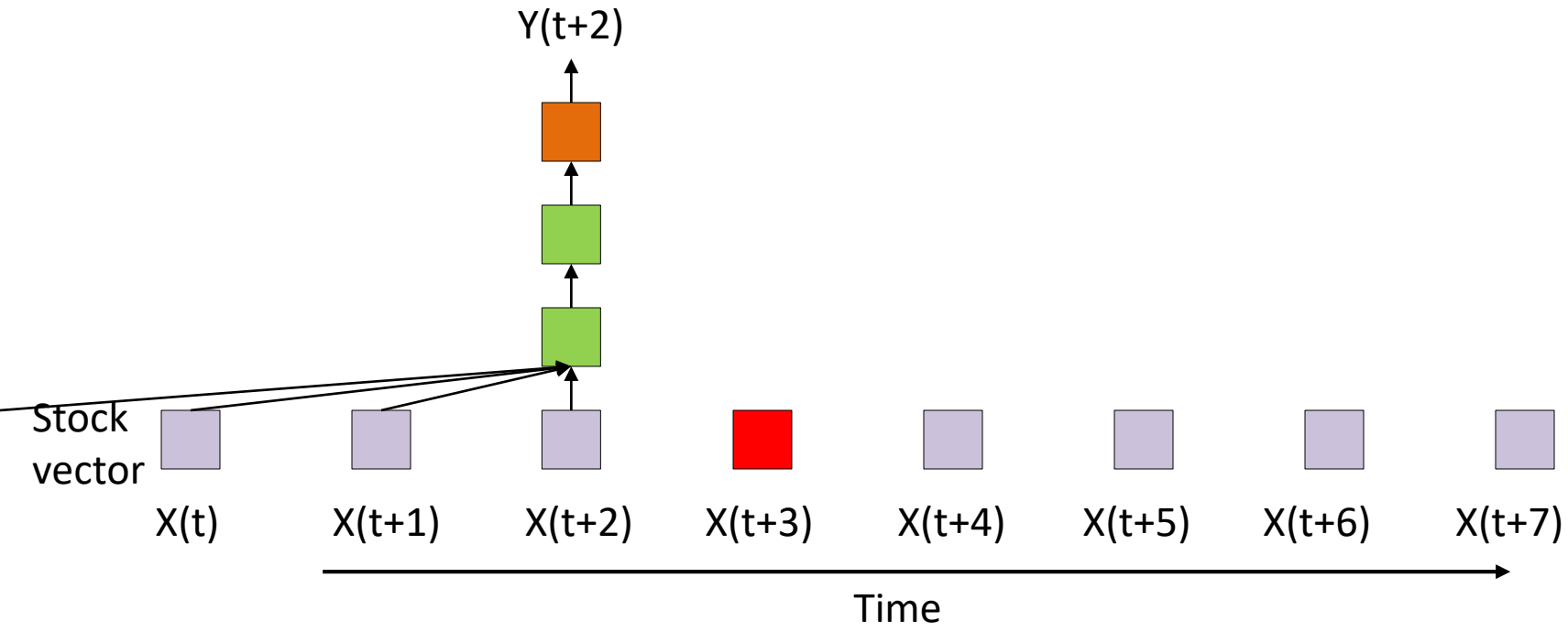
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

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

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

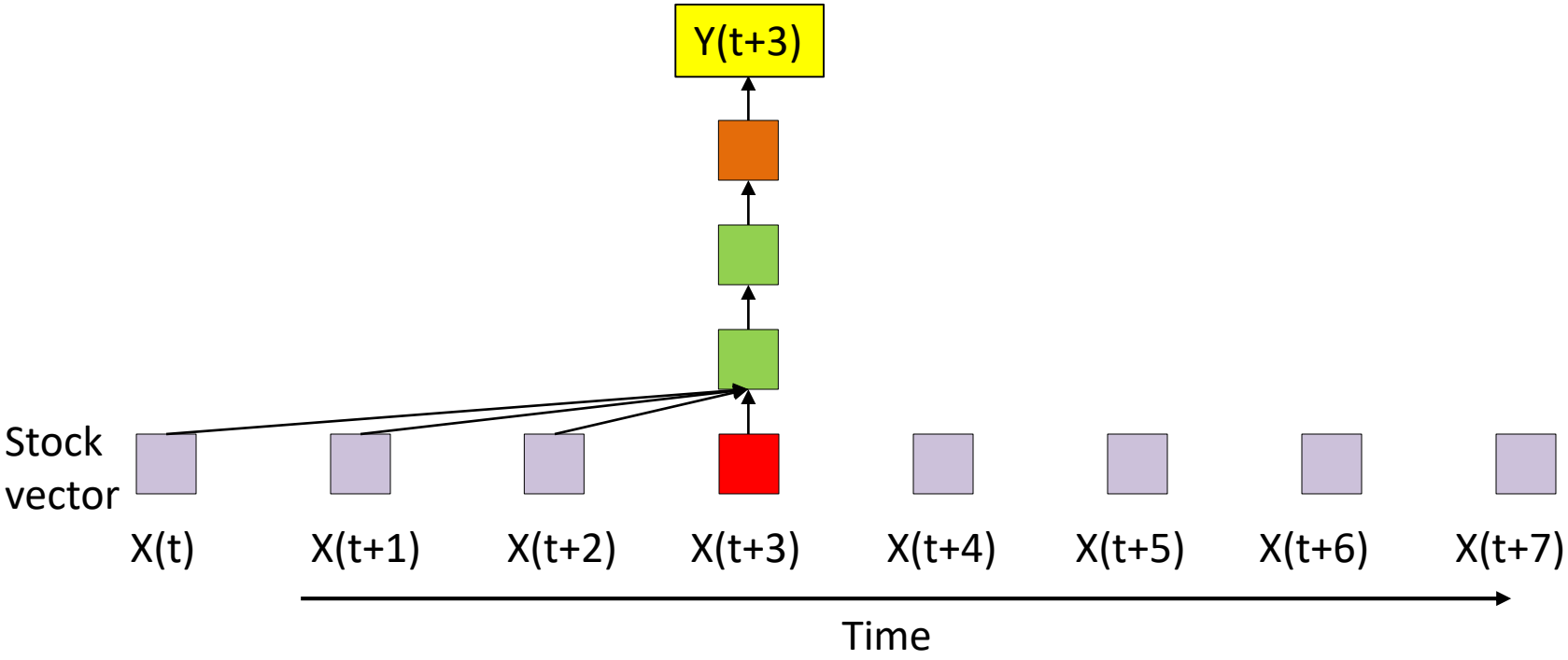
The stock predictor



- 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

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

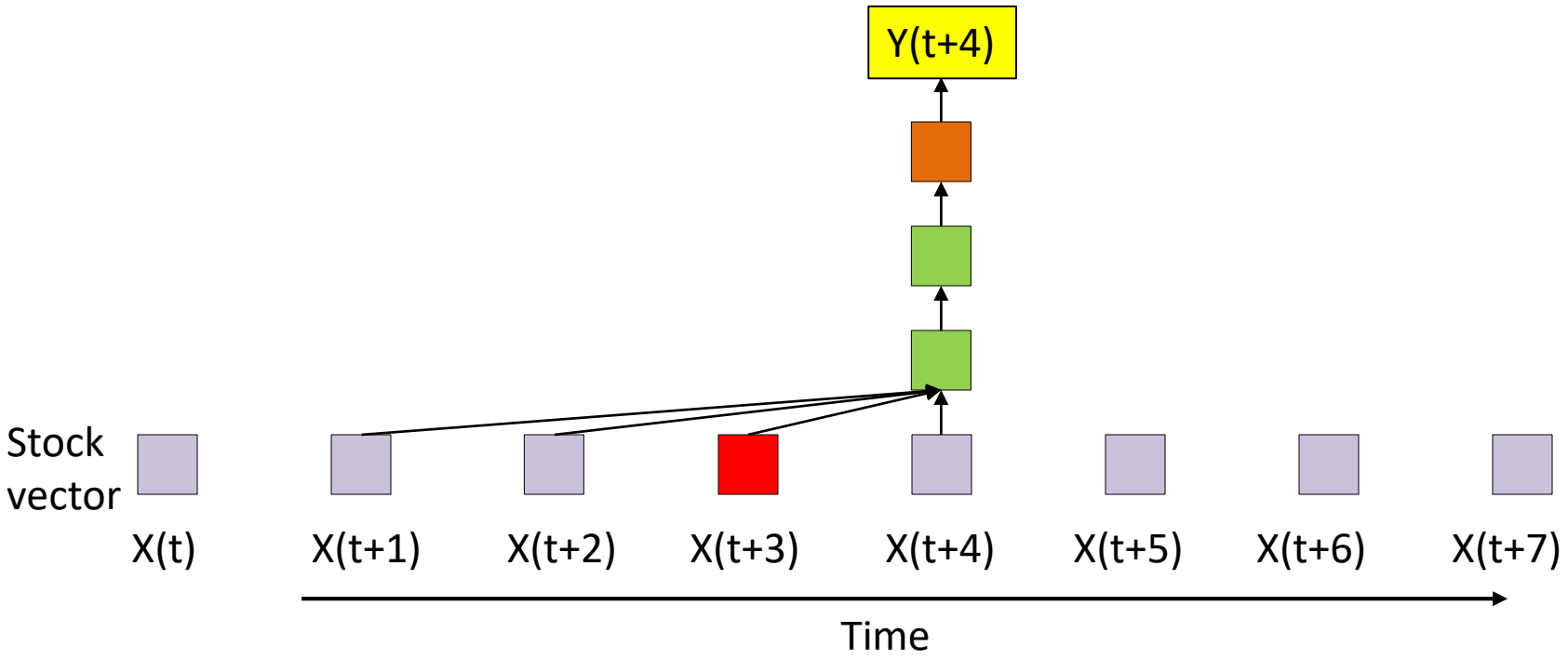
The stock predictor



- 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

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

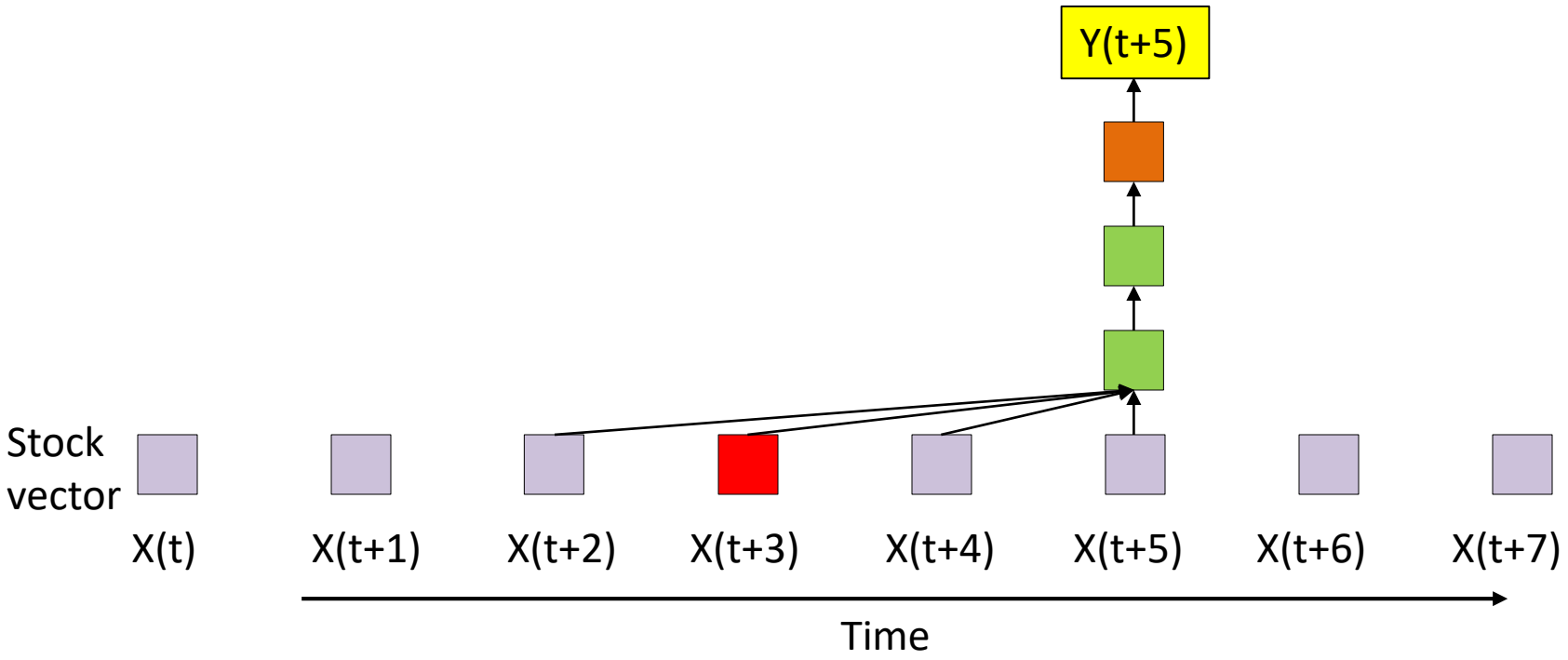
The stock predictor



- 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

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

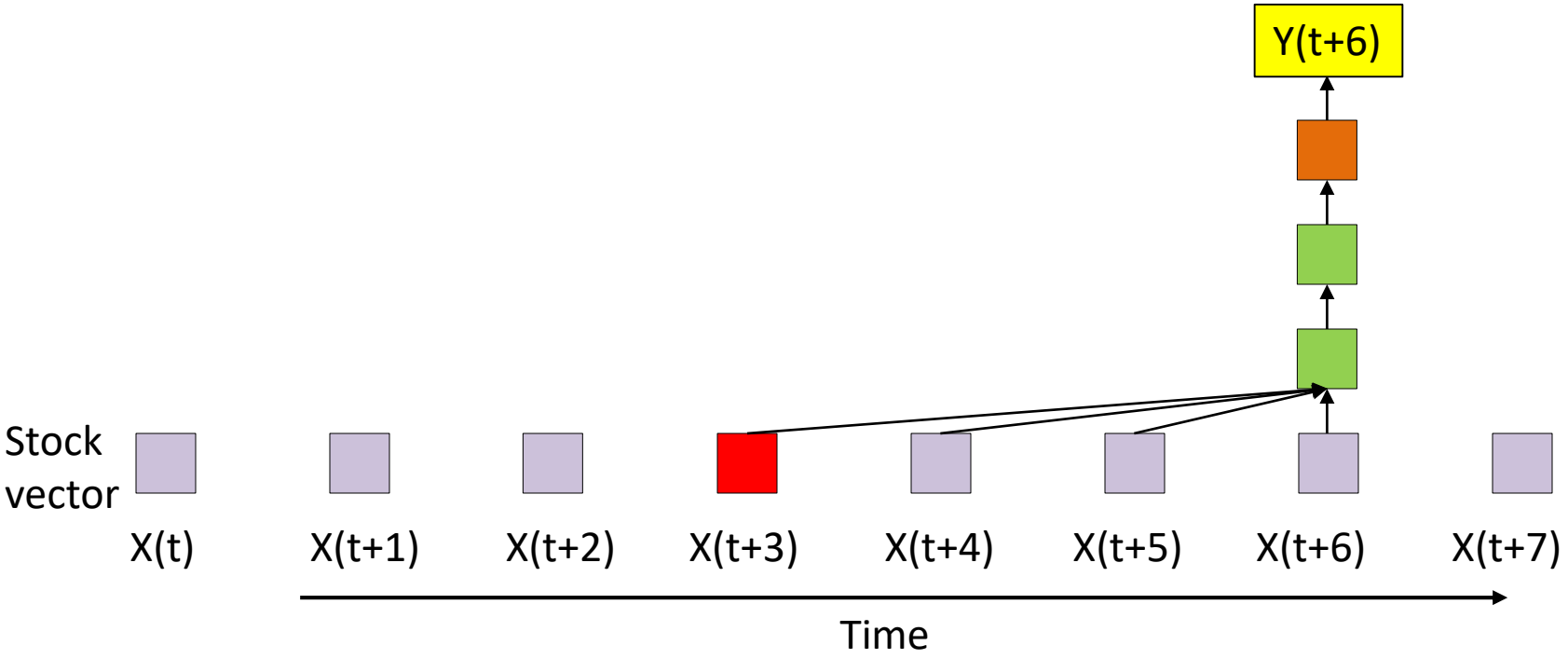
The stock predictor



- 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

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

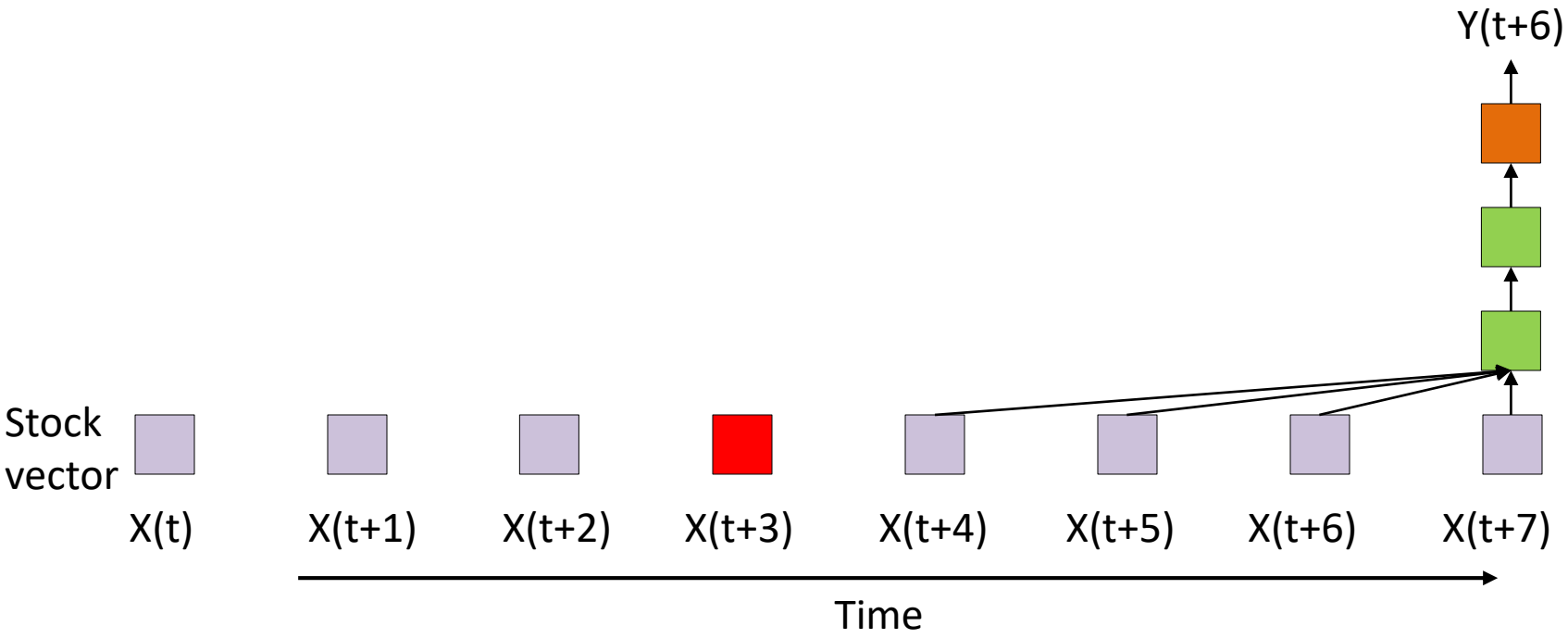
The stock predictor



- 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

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

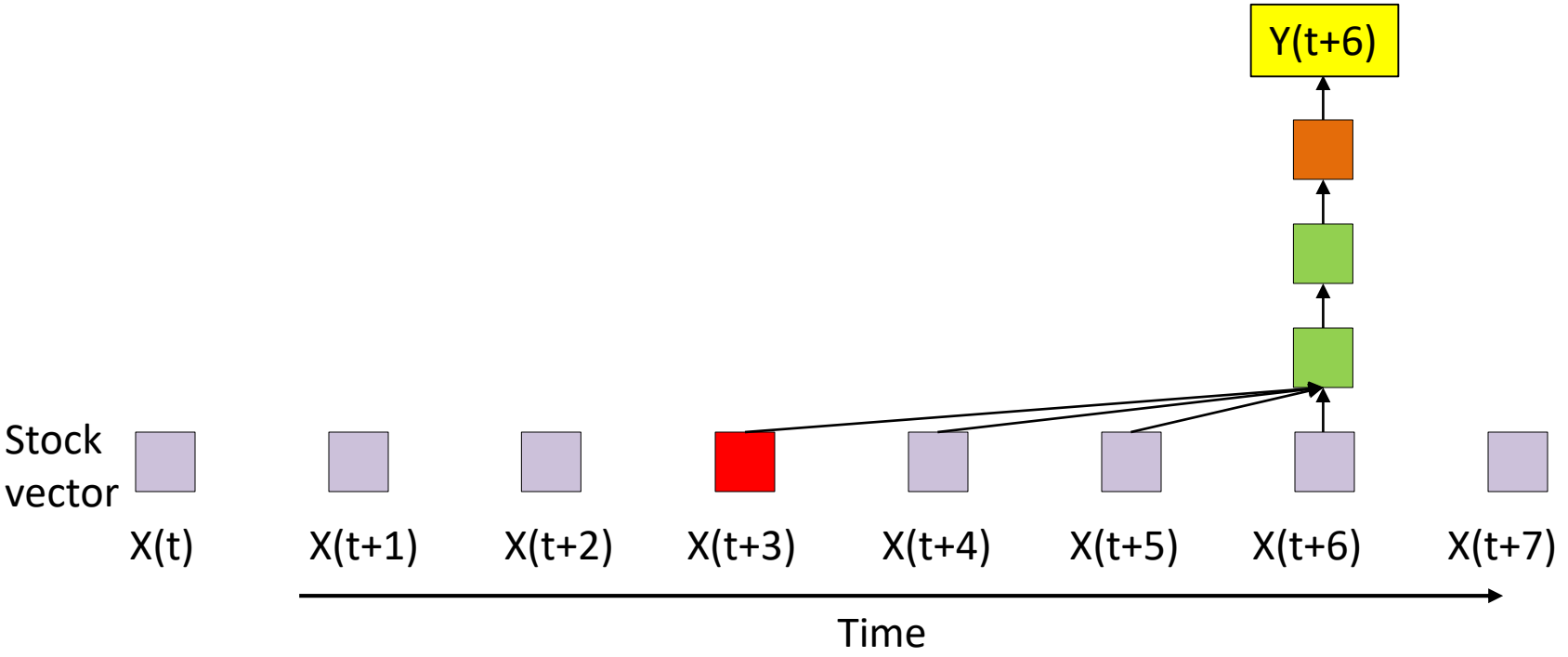
The stock predictor



- 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

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

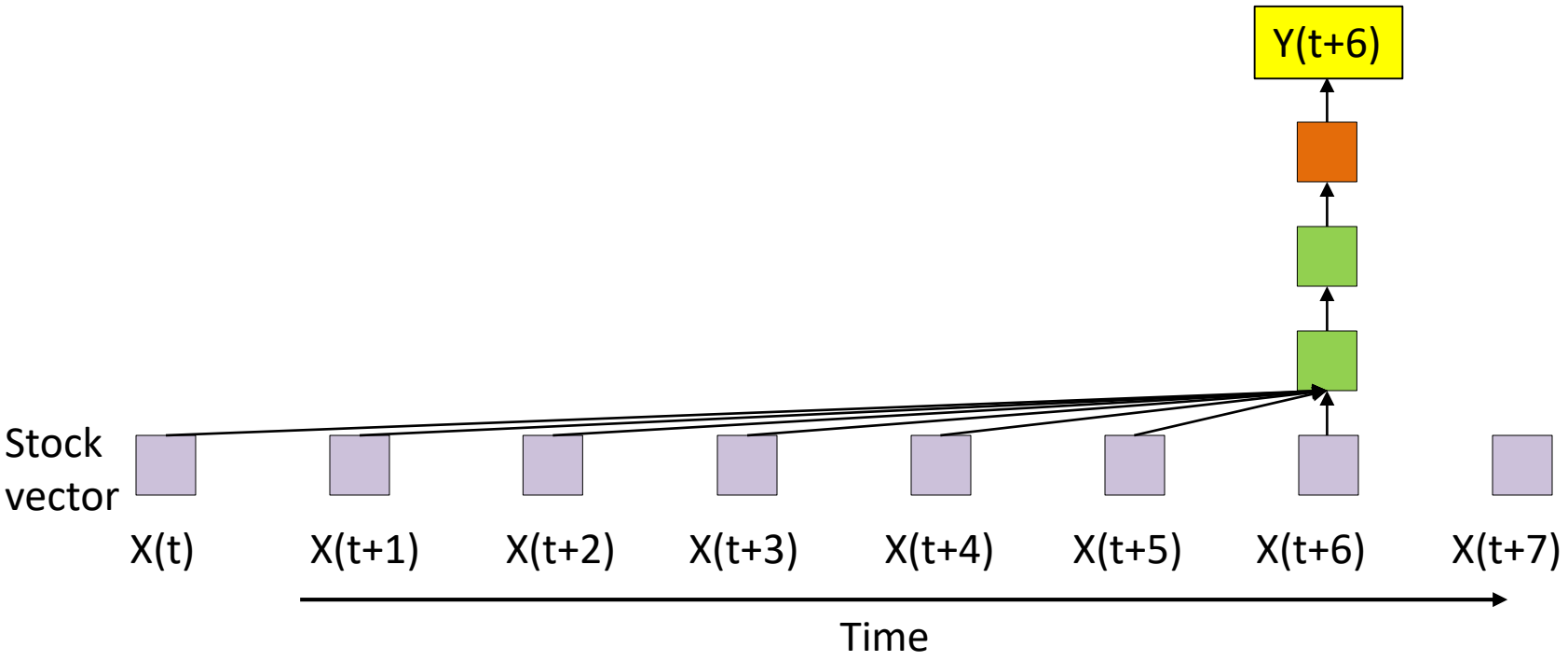
Finite-response model



- 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

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

Finite-response

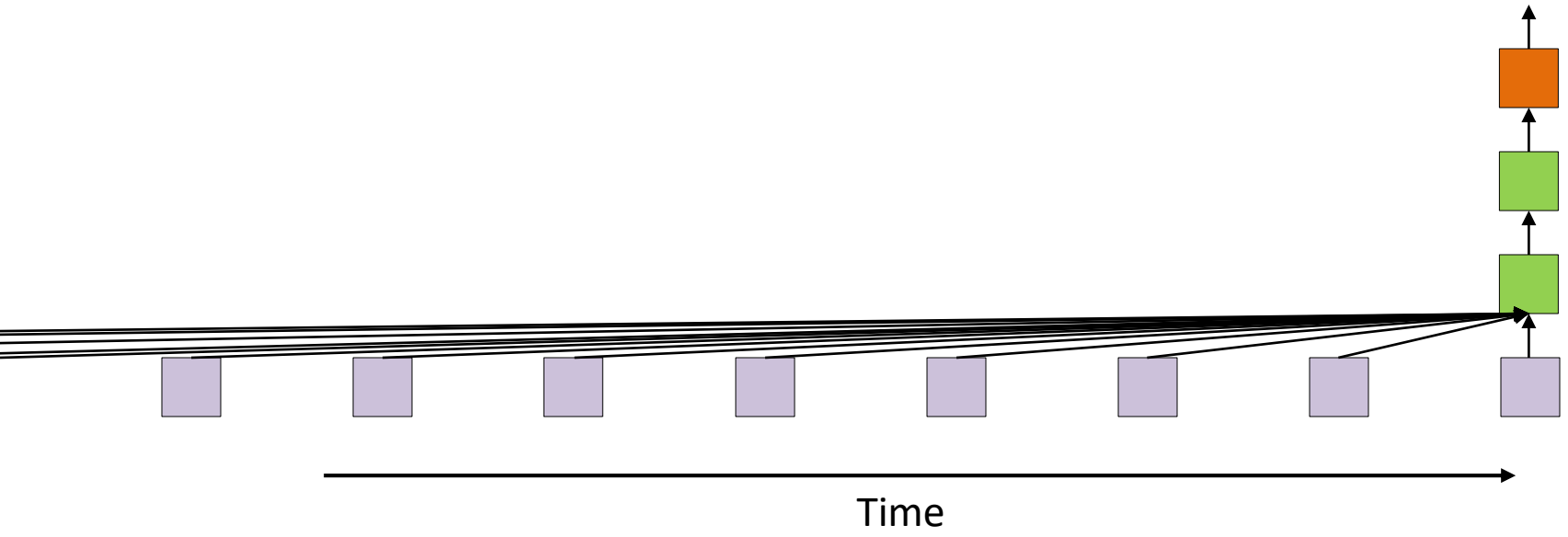


- Problem: Increasing the "history" makes the network more complex
  - No worries, we have the CPU and memory
    - Or do we?

Systems often have long-term dependencies

- Longer-term trends –
  - Weekly trends in the market
  - Monthly trends in the market
  - Annual trends
  - Though longer history tends 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

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

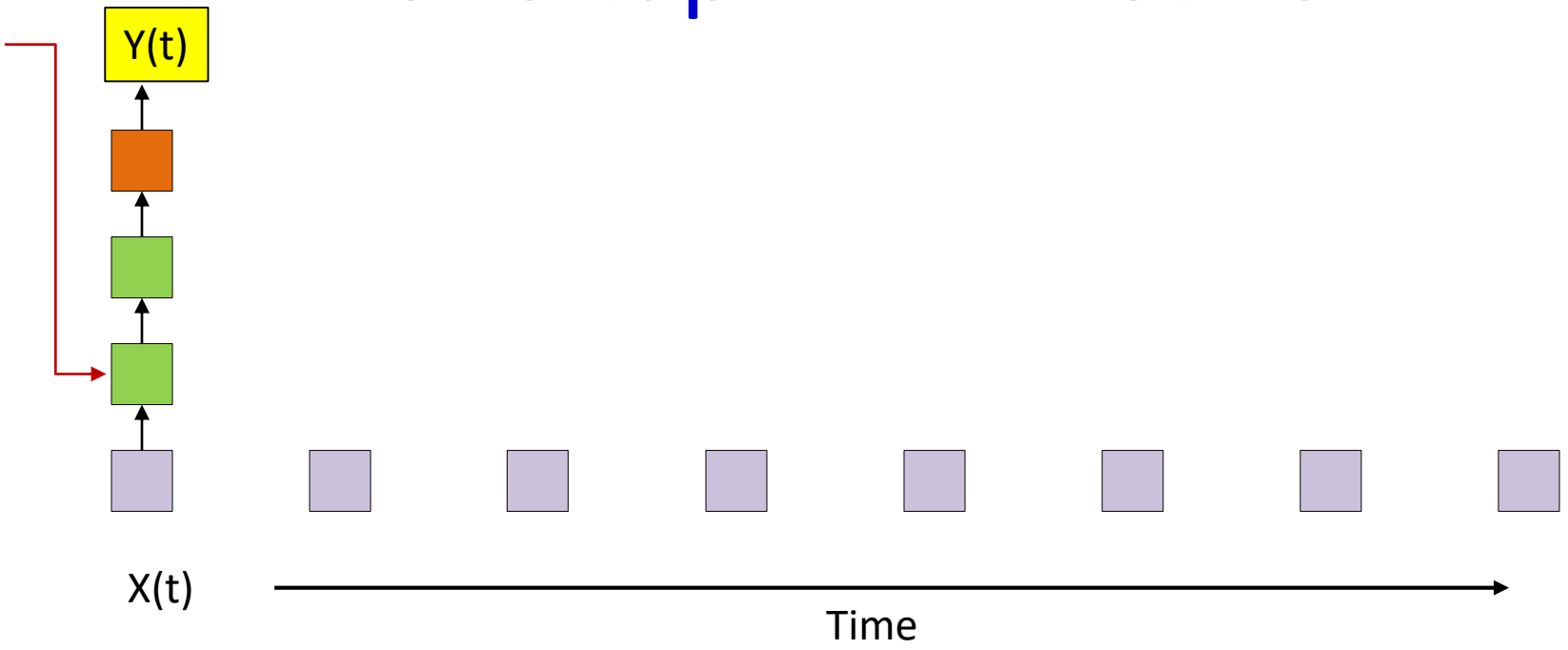
Examples of infinite response systems

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

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

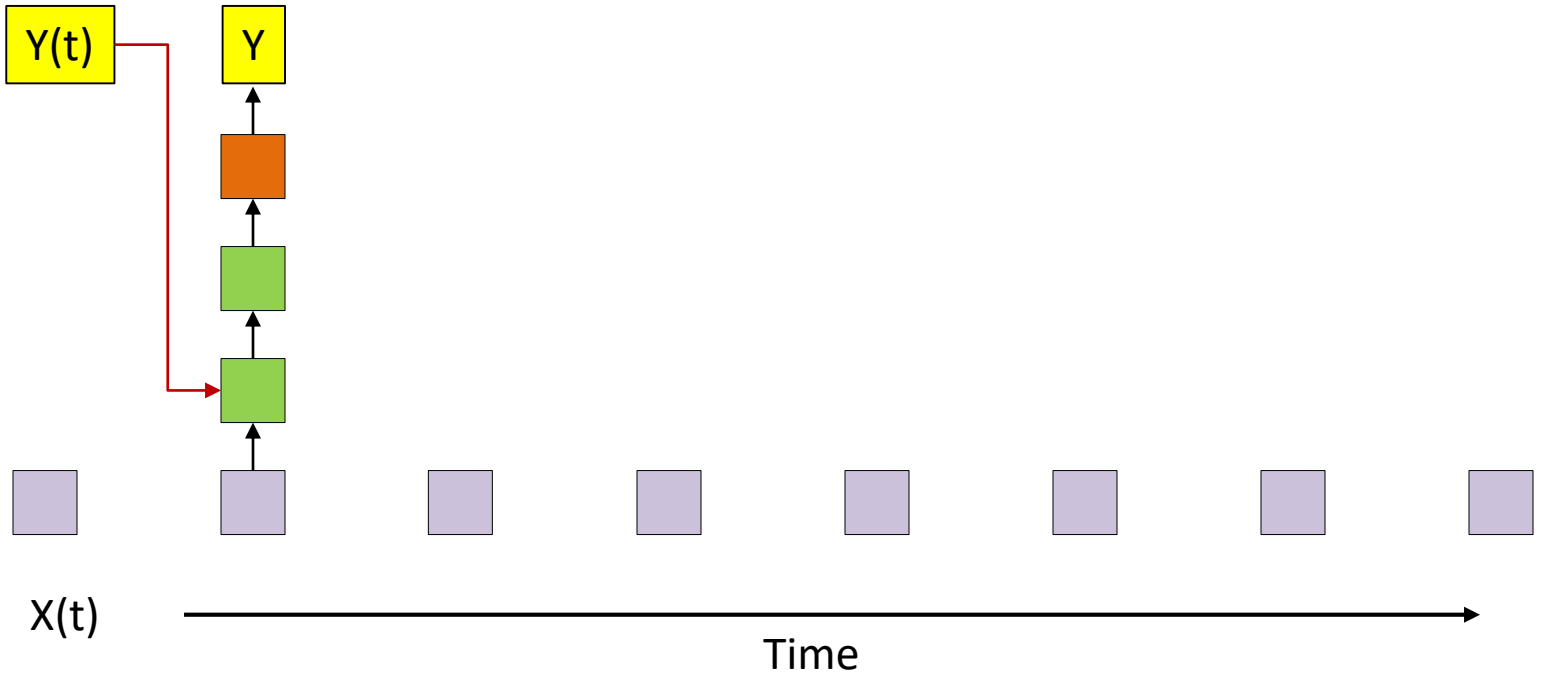
- This is an instance of a NARX network
  - "nonlinear autoregressive network with exogenous inputs"
  - $Y_t = f(X_{0:t}, Y_{0:t-1})$
- *Output* contains information about the entire past

A one-tap NARX network



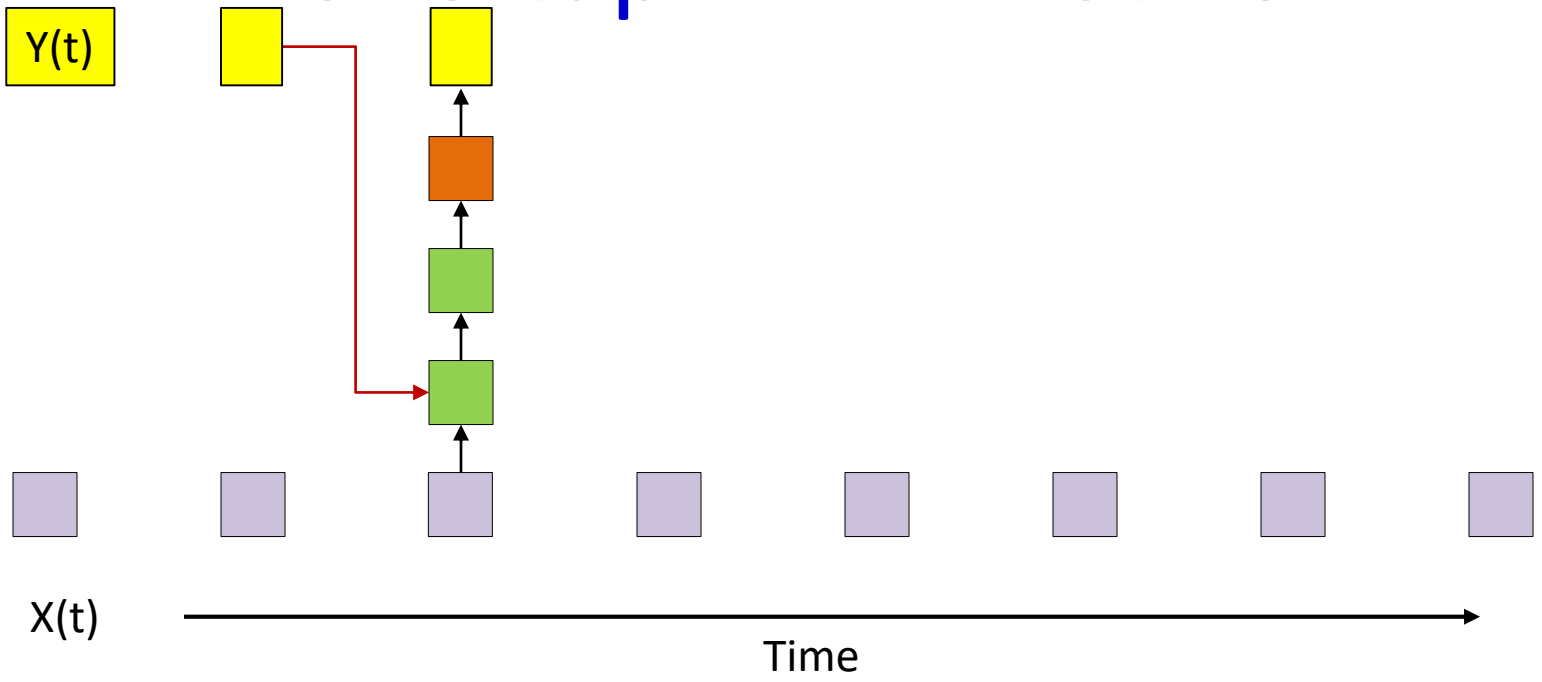
- A NARX net with recursion from the output

A NARX network



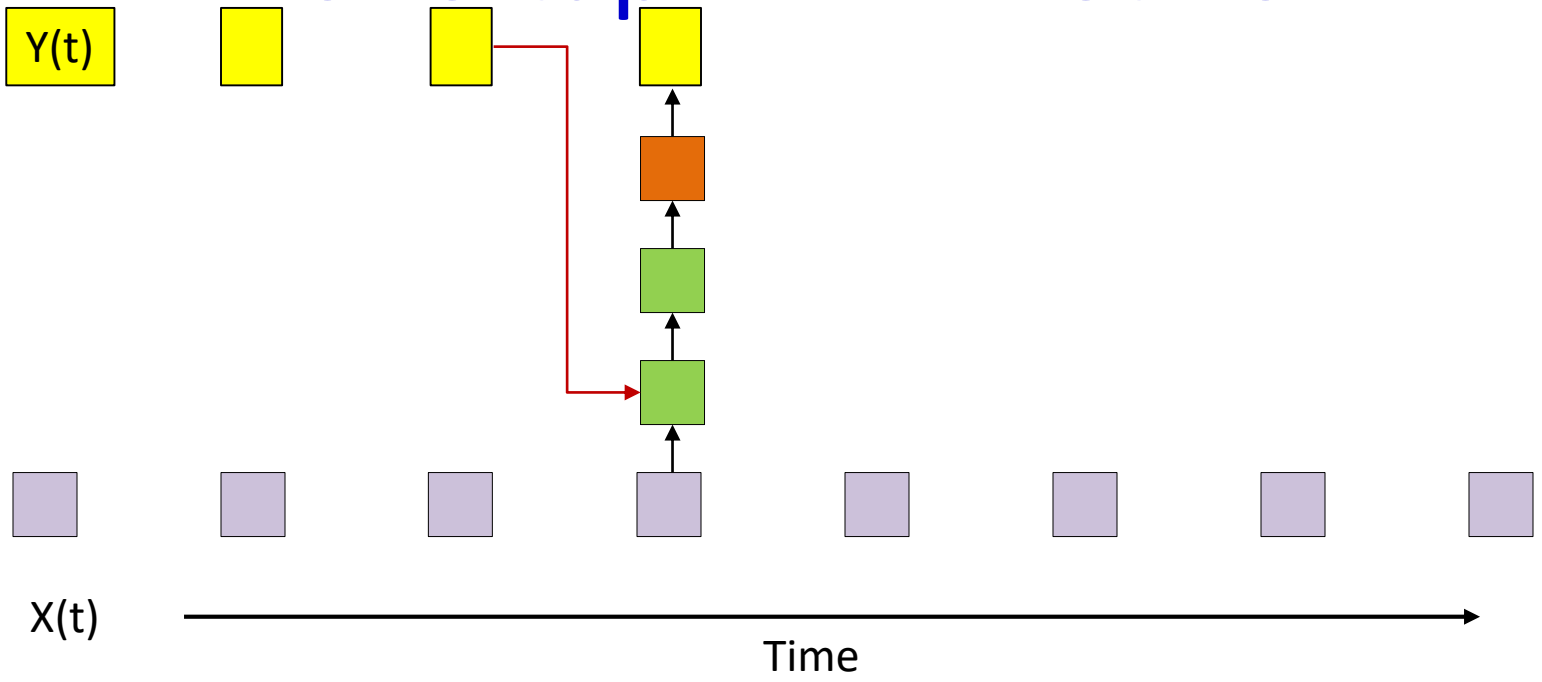
- A NARX net with recursion from the output

A one-tap NARX network



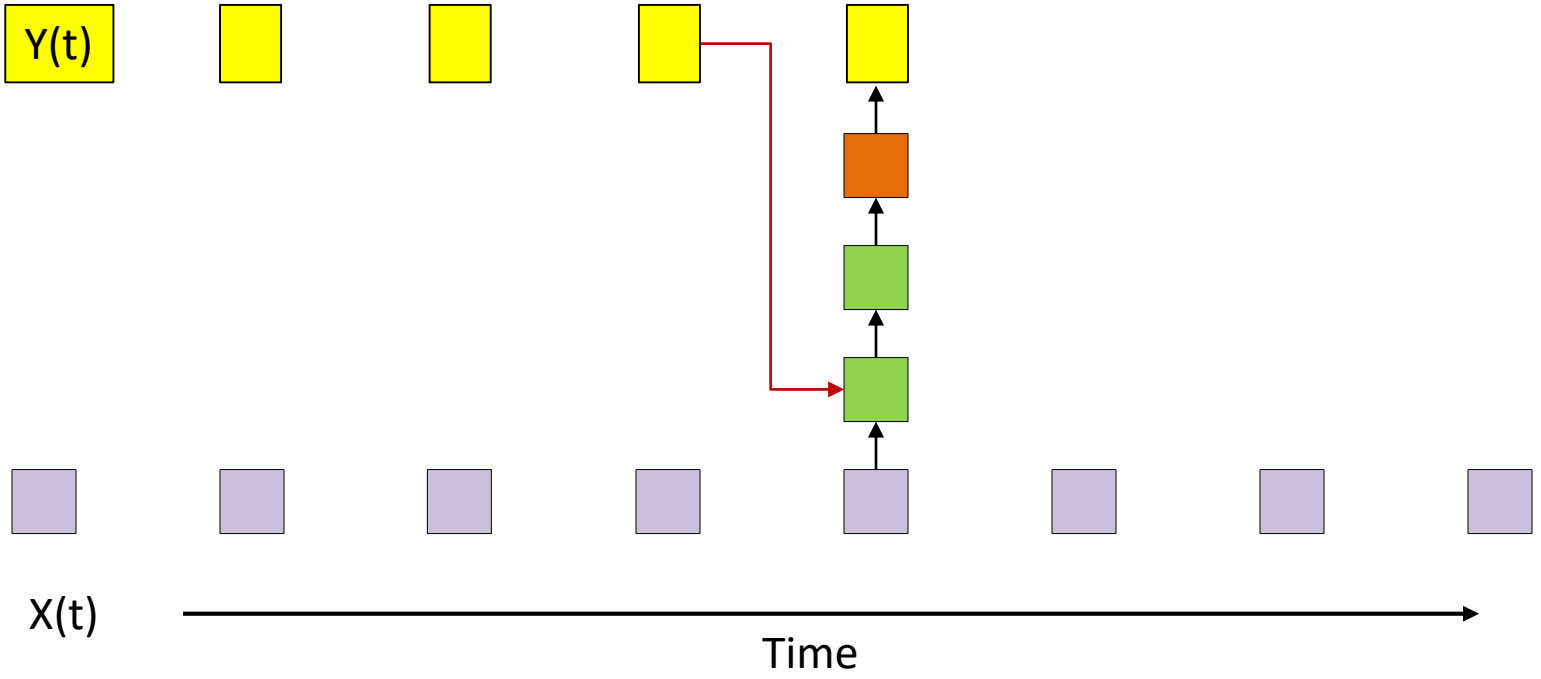
- A NARX net with recursion from the output

A one-tap NARX network



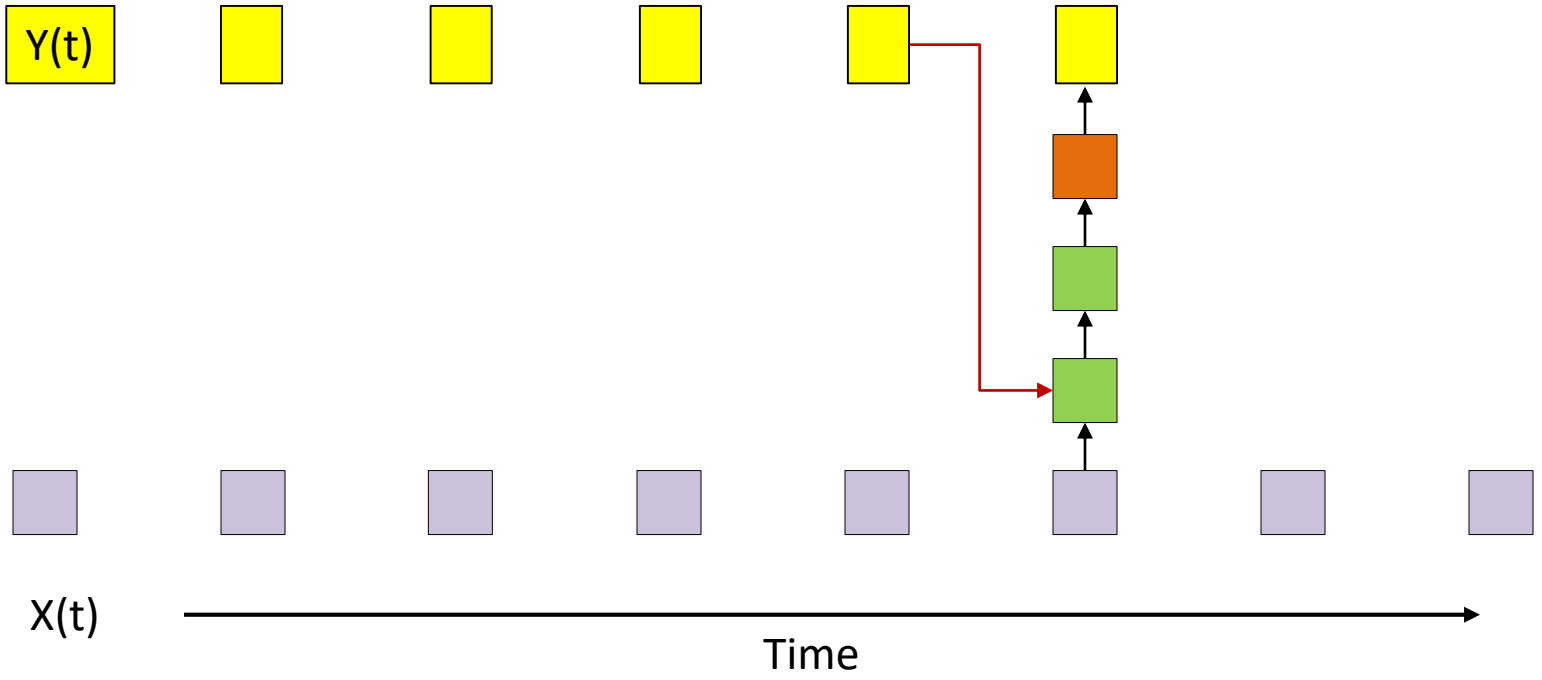
- A NARX net with recursion from the output

A one-tap NARX network



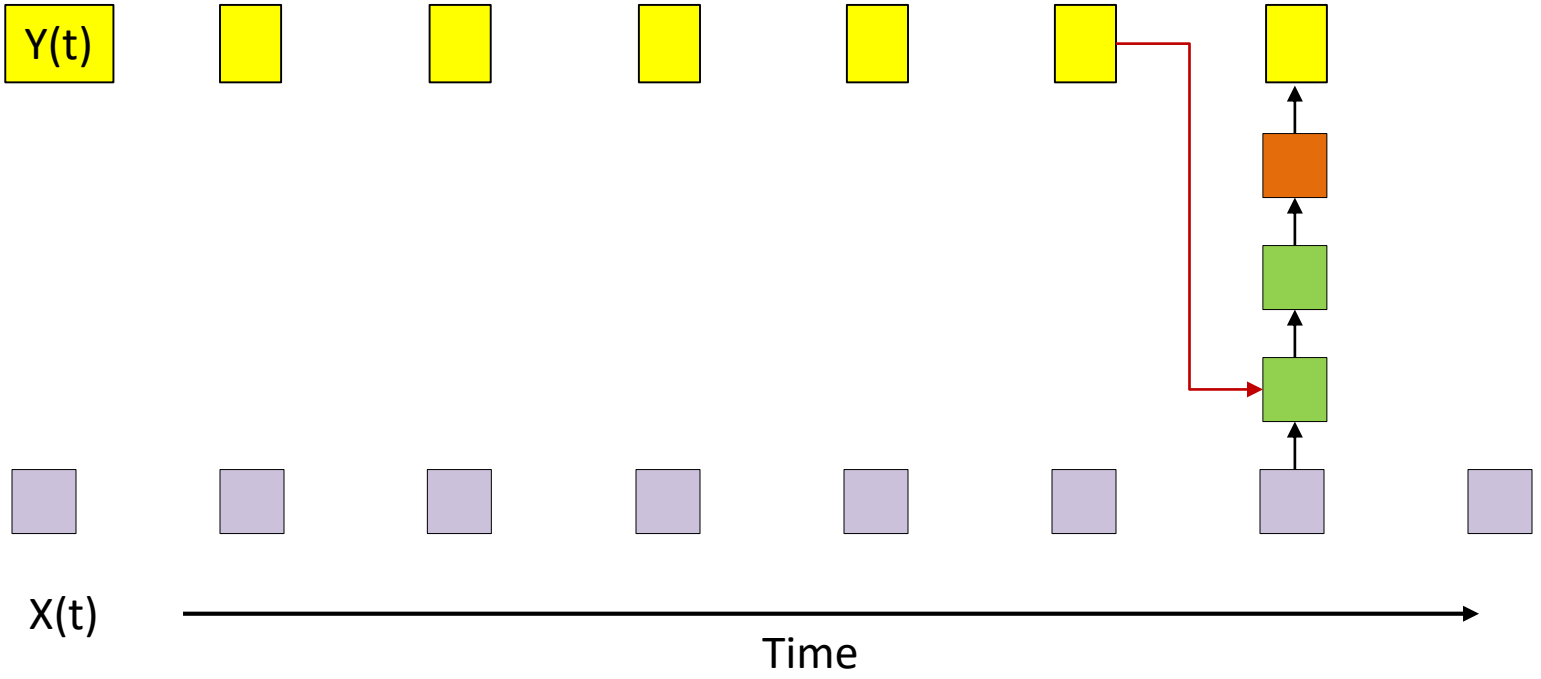
- A NARX net with recursion from the output

A one-tap NARX network



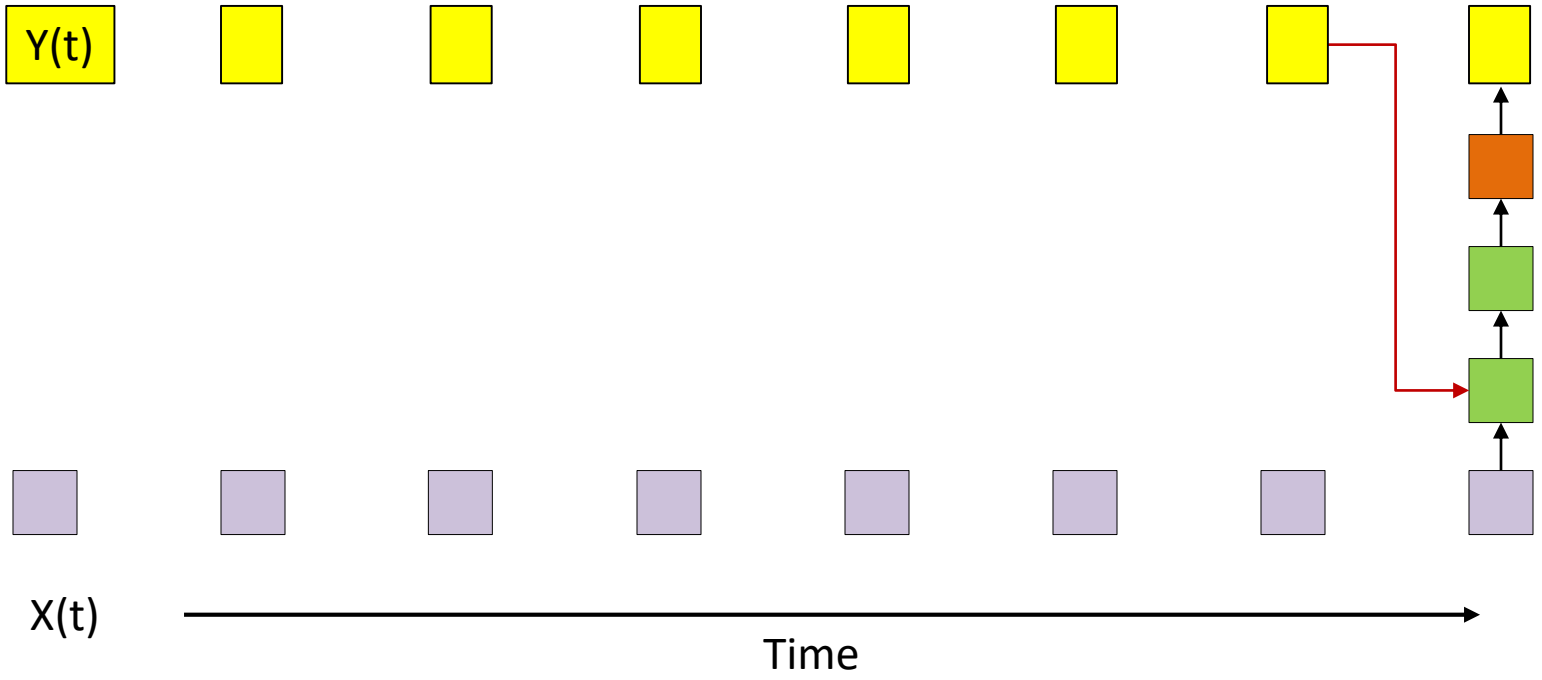
- A NARX net with recursion from the output

A one-tap NARX network



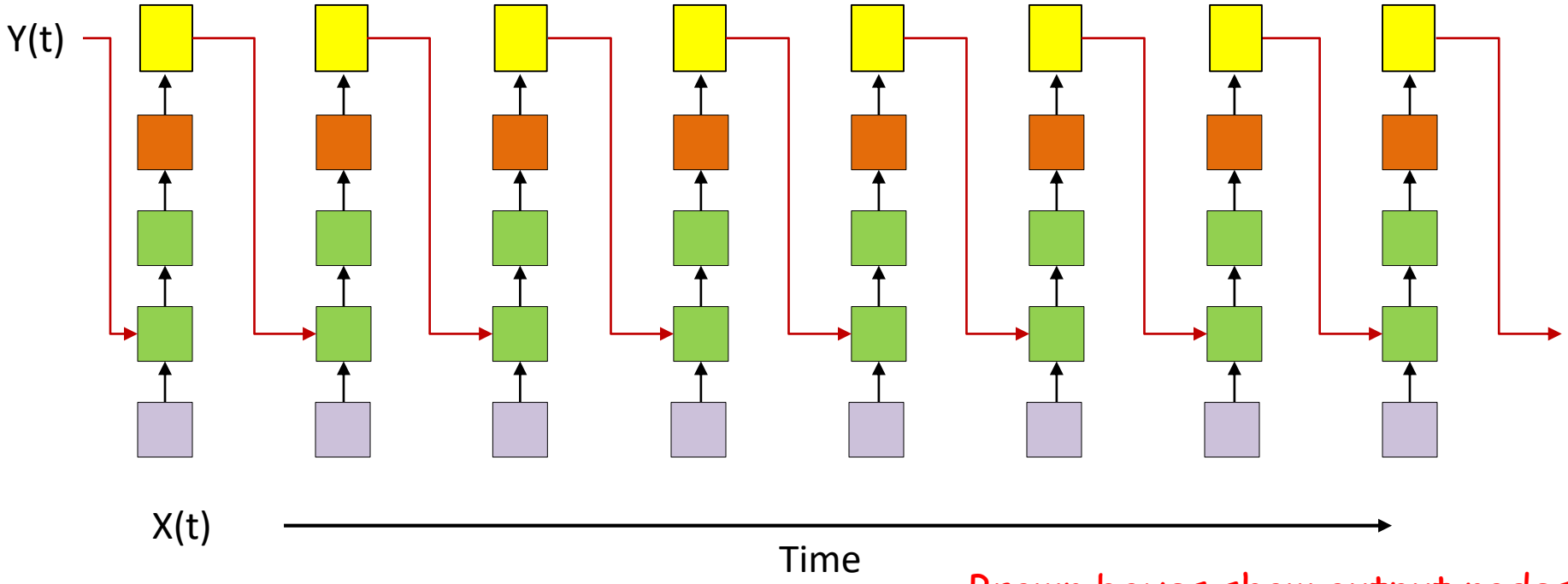
- A NARX net with recursion from the output

A one-tap NARX network



- A NARX net with recursion from the output

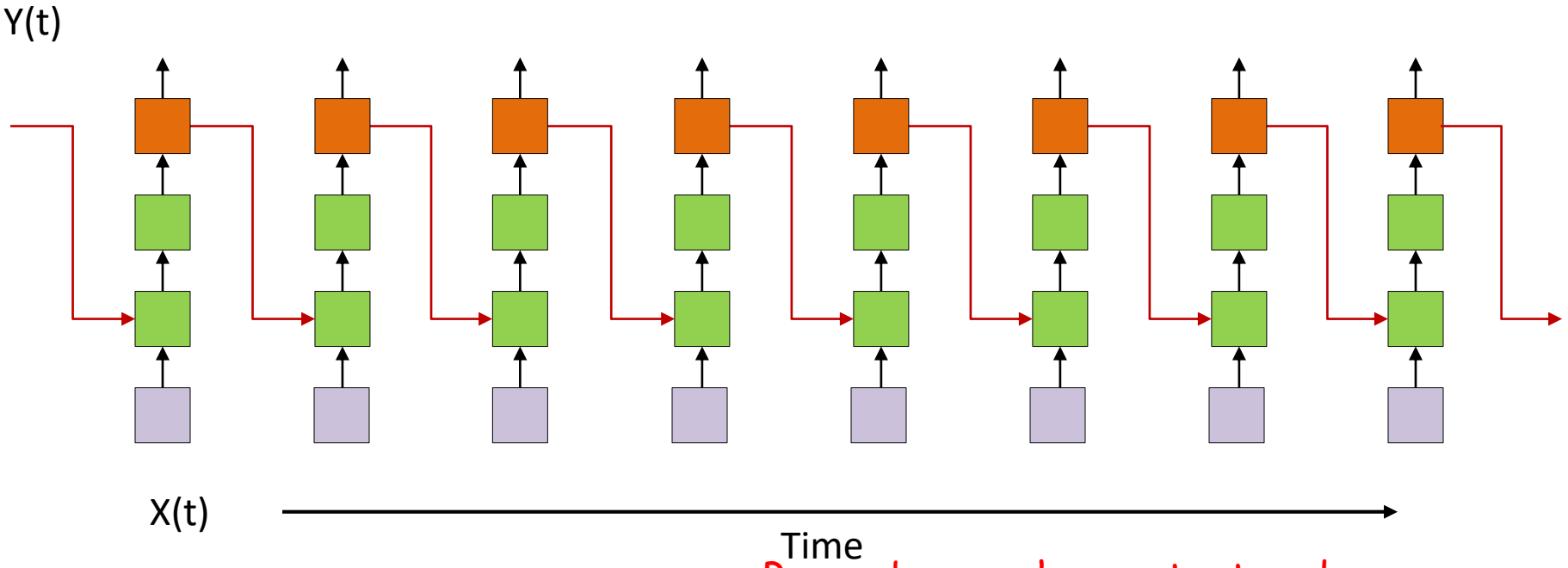
A more complete representation



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*

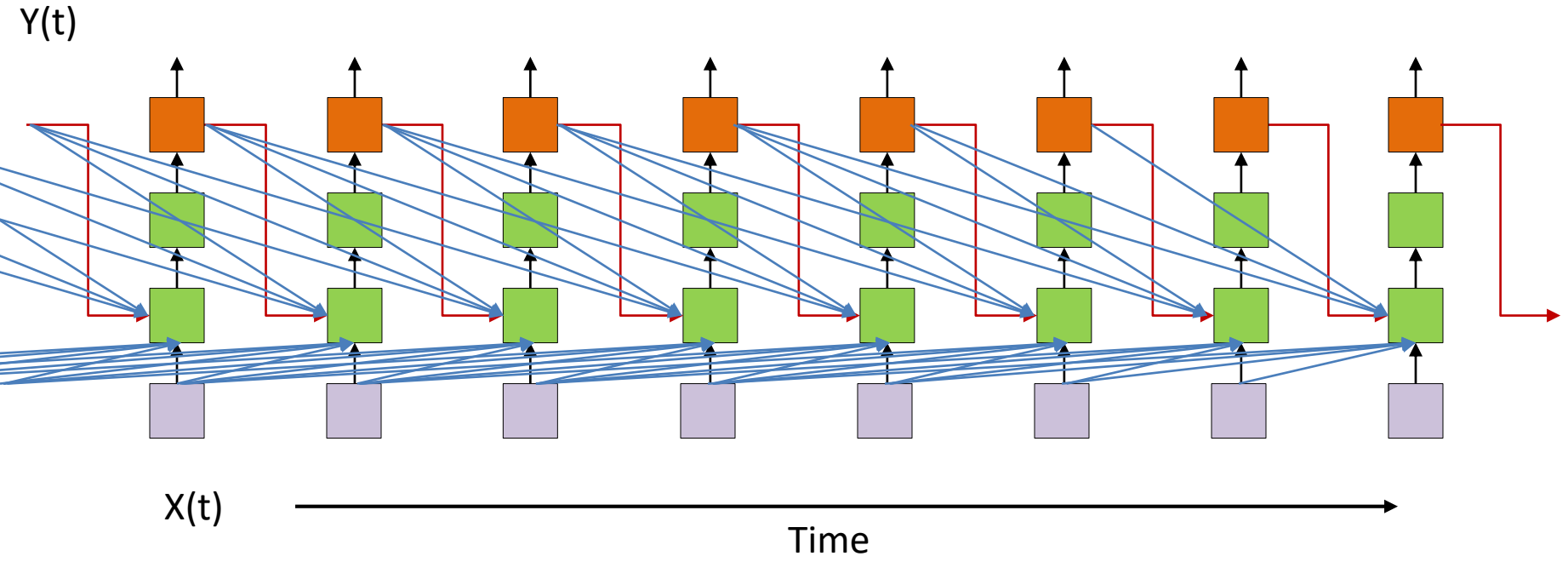
Same figure redrawn



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

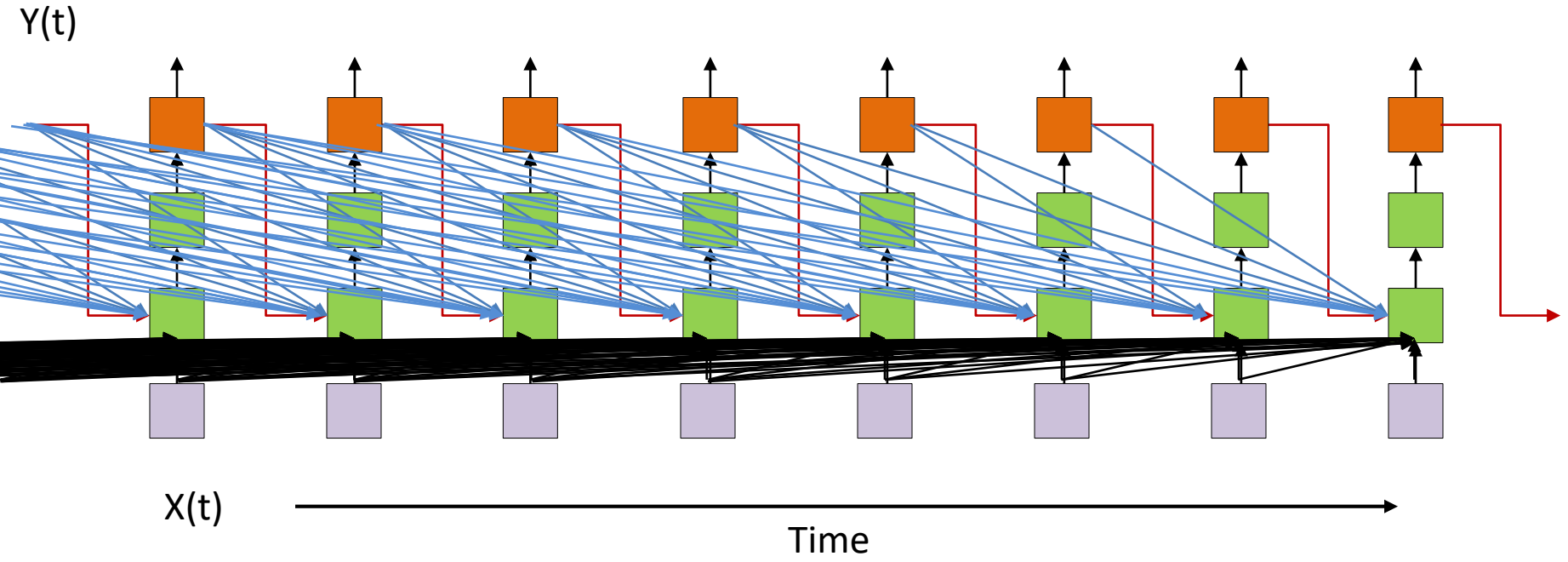
- A NARX net with recursion from the output
- Showing all computations
- All columns are identical
- *An input at t=0 affects outputs forever*

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 $X_t, \ldots, X_{t-L}$

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

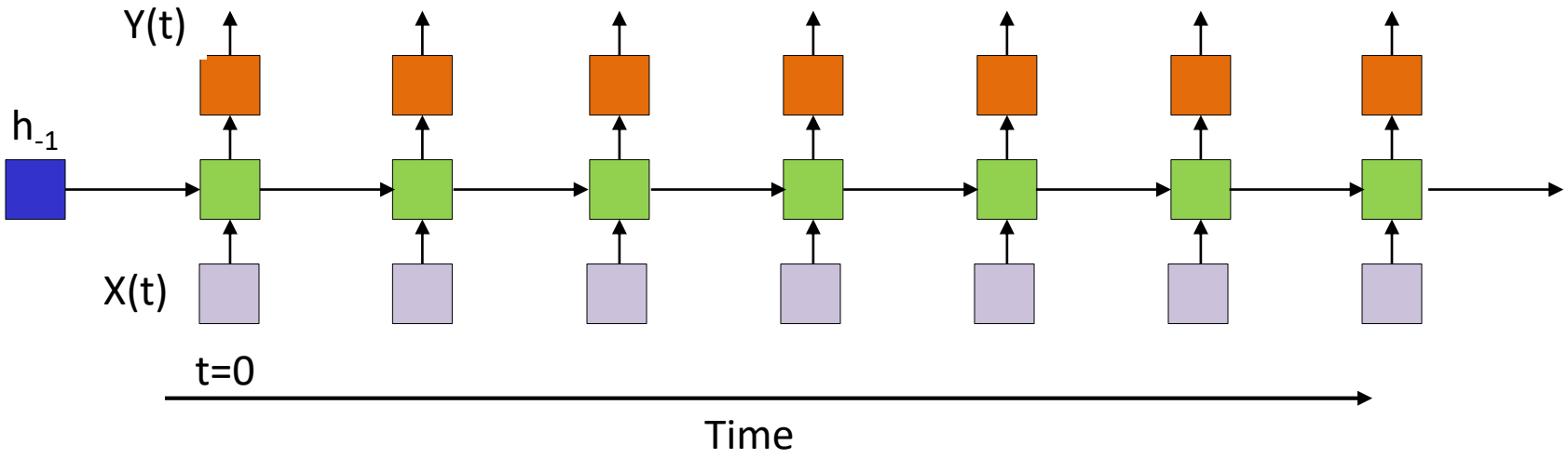
An alternate model for infinite response systems: the state-space model

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

- h_t is the *state* of the network
- Need to define initial state h_{-1}

- This is a *recurrent* neural network
- *State* summarizes information about the entire past

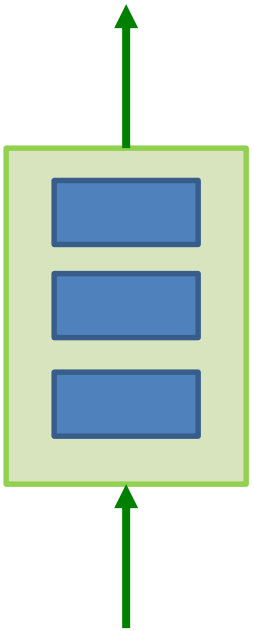
The simple state-space model



- 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

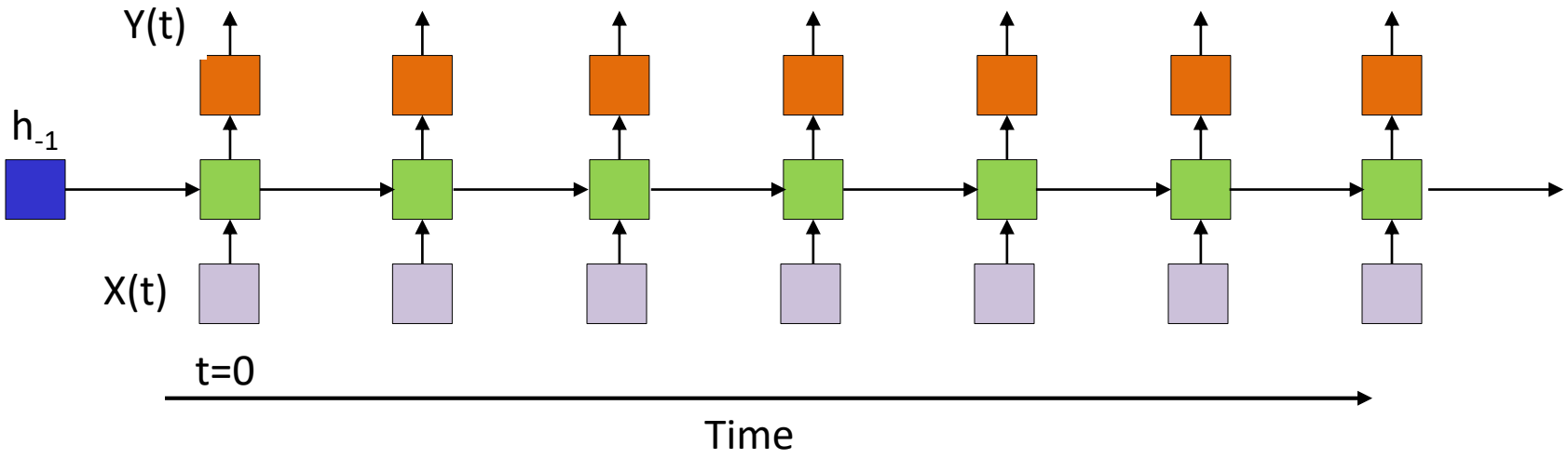
$$h_t = f(x_t, h_{t-1})$$
$$y_t = g(h_t)$$



- h_t is the *state* of the network
- Need to define initial state h_{-1}

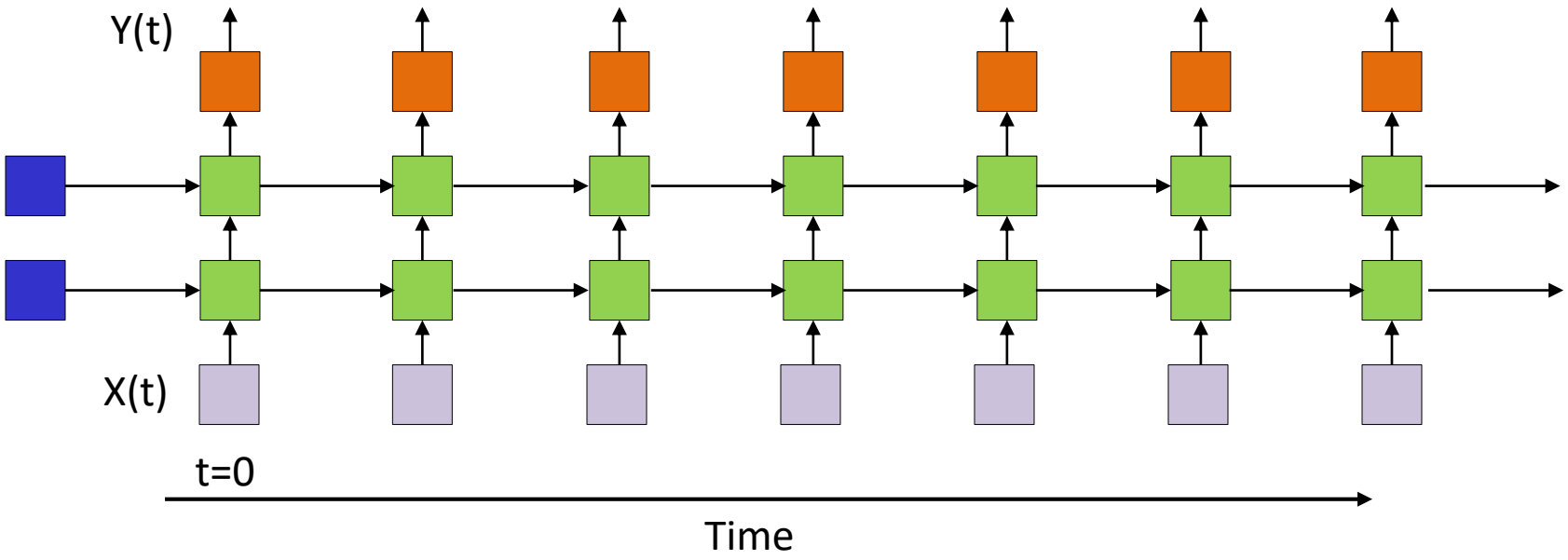
- The state an be arbitrarily complex

Single hidden layer RNN



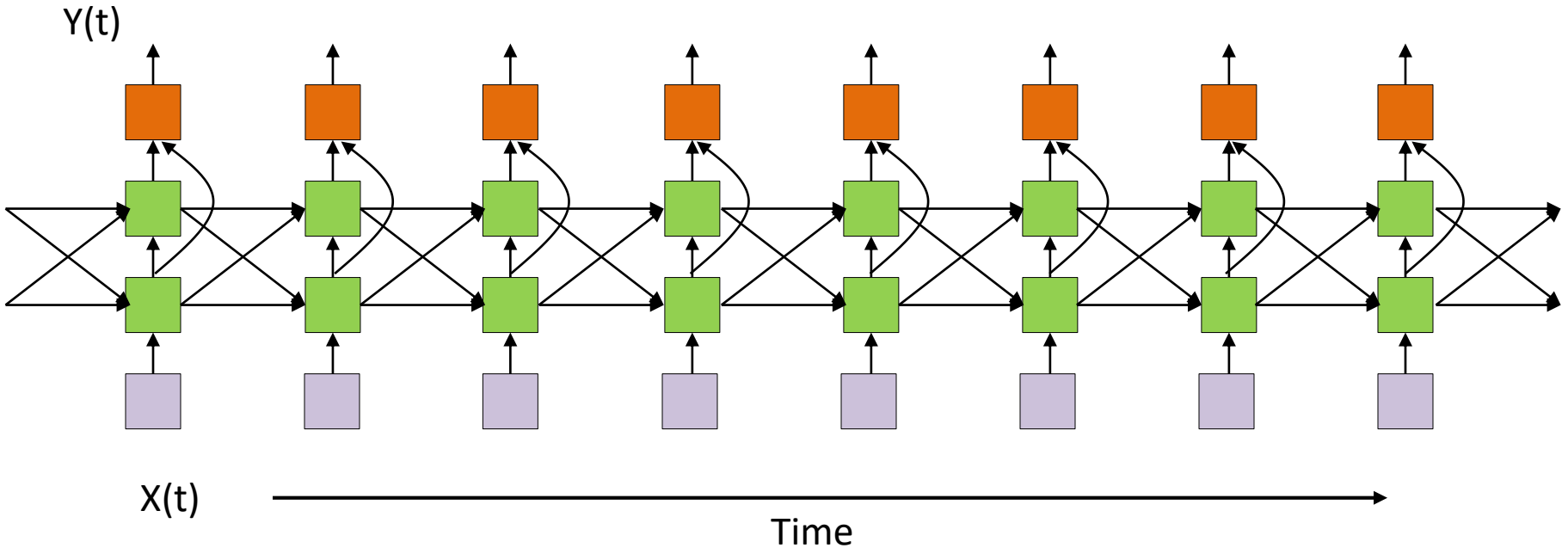
- Recurrent neural network
- All columns are identical
- *An input at t=0 affects outputs forever*

Multiple recurrent layer RNN



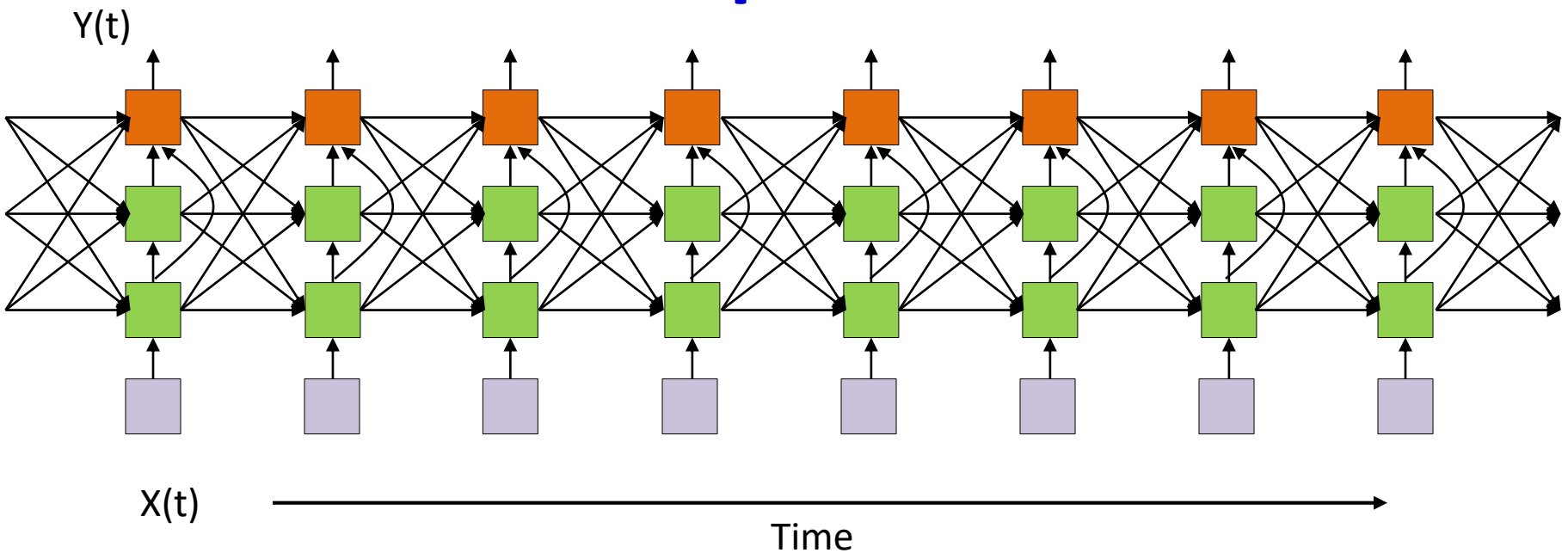
- Recurrent neural network
- All columns are identical
- *An input at t=0 affects outputs forever*

A more complex state



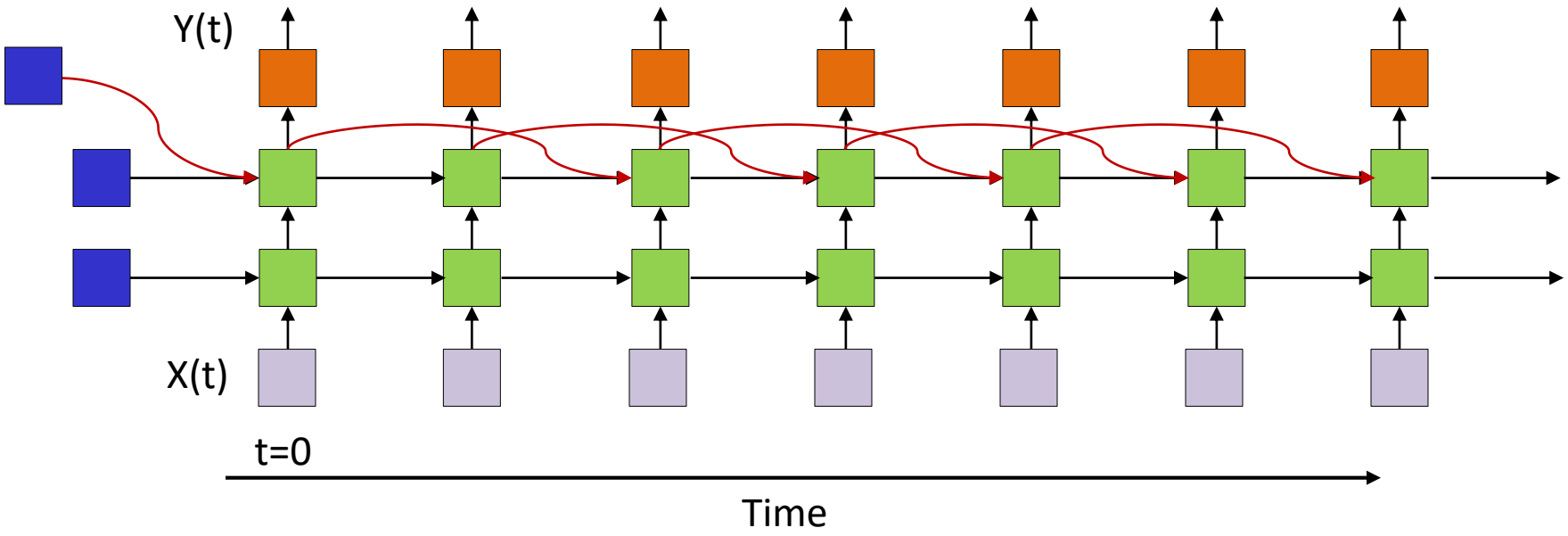
- All columns are identical
- *An input at t=0 affects outputs forever*

Or the network may be even more complicated



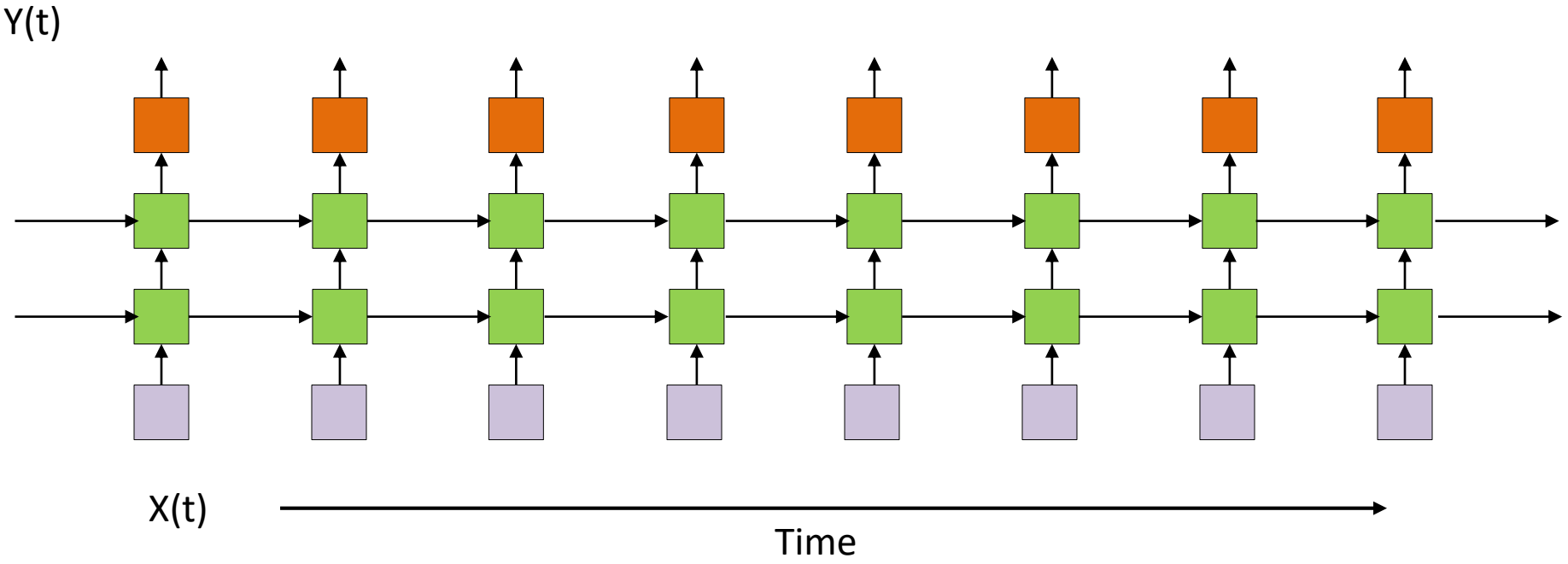
- Shades of NARX
- All columns are identical
- *An input at t=0 affects outputs forever*

Generalization with other recurrences



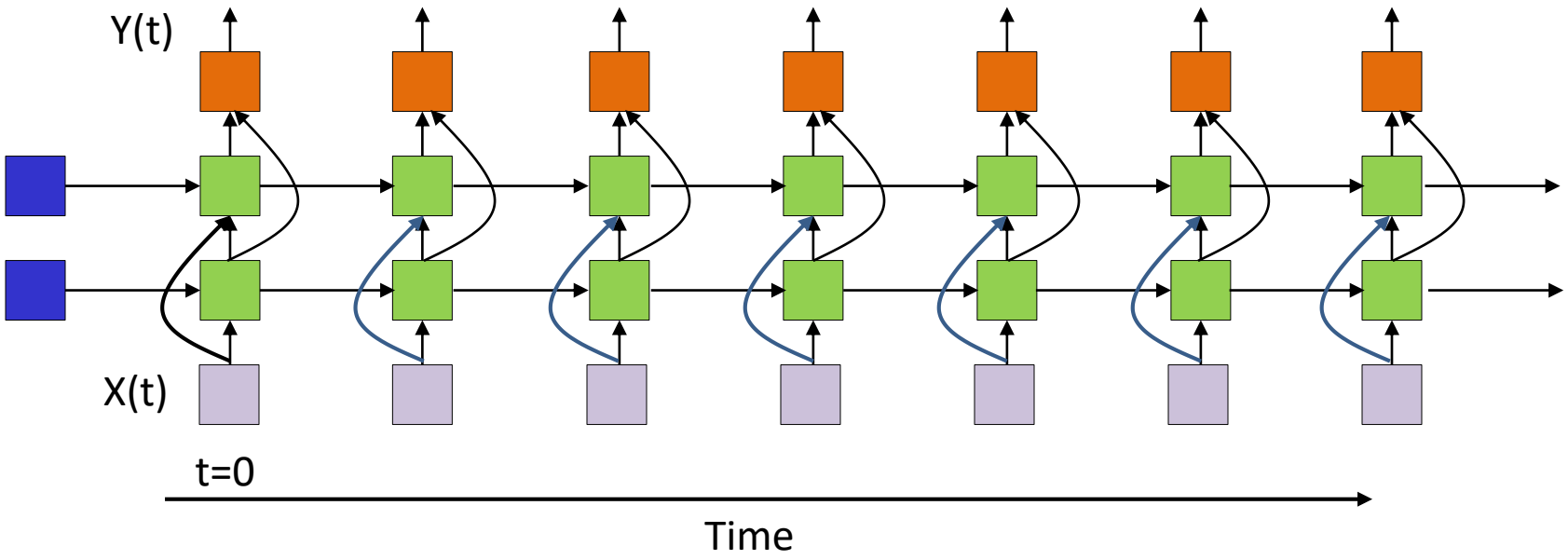
- All column (including incoming edges) are identical

State dependencies may be simpler



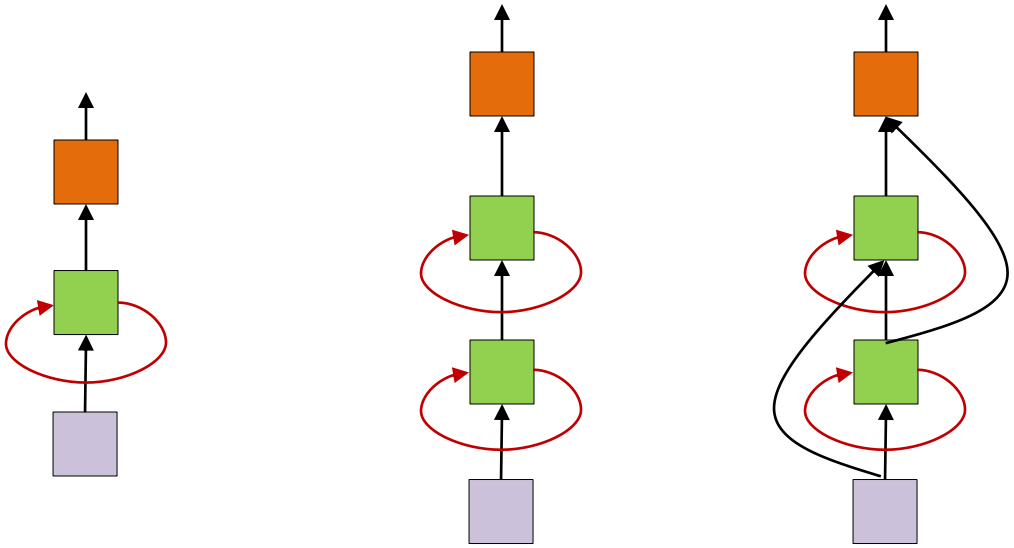
- Recurrent neural network
- All columns are identical
- *An input at t=0 affects outputs forever*

Multiple recurrent layer RNN



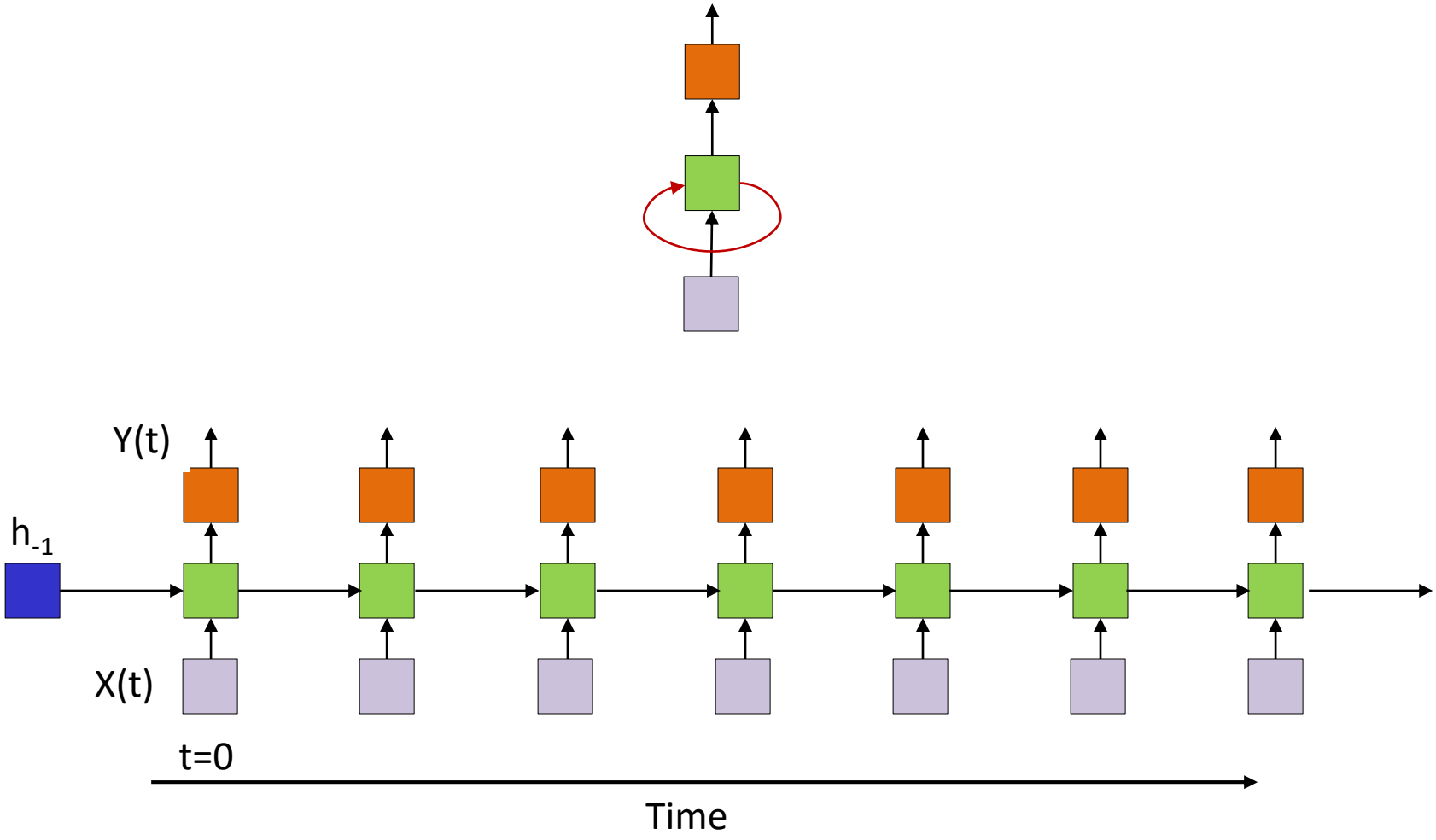
- We can also have skips..

A Recurrent Neural Network

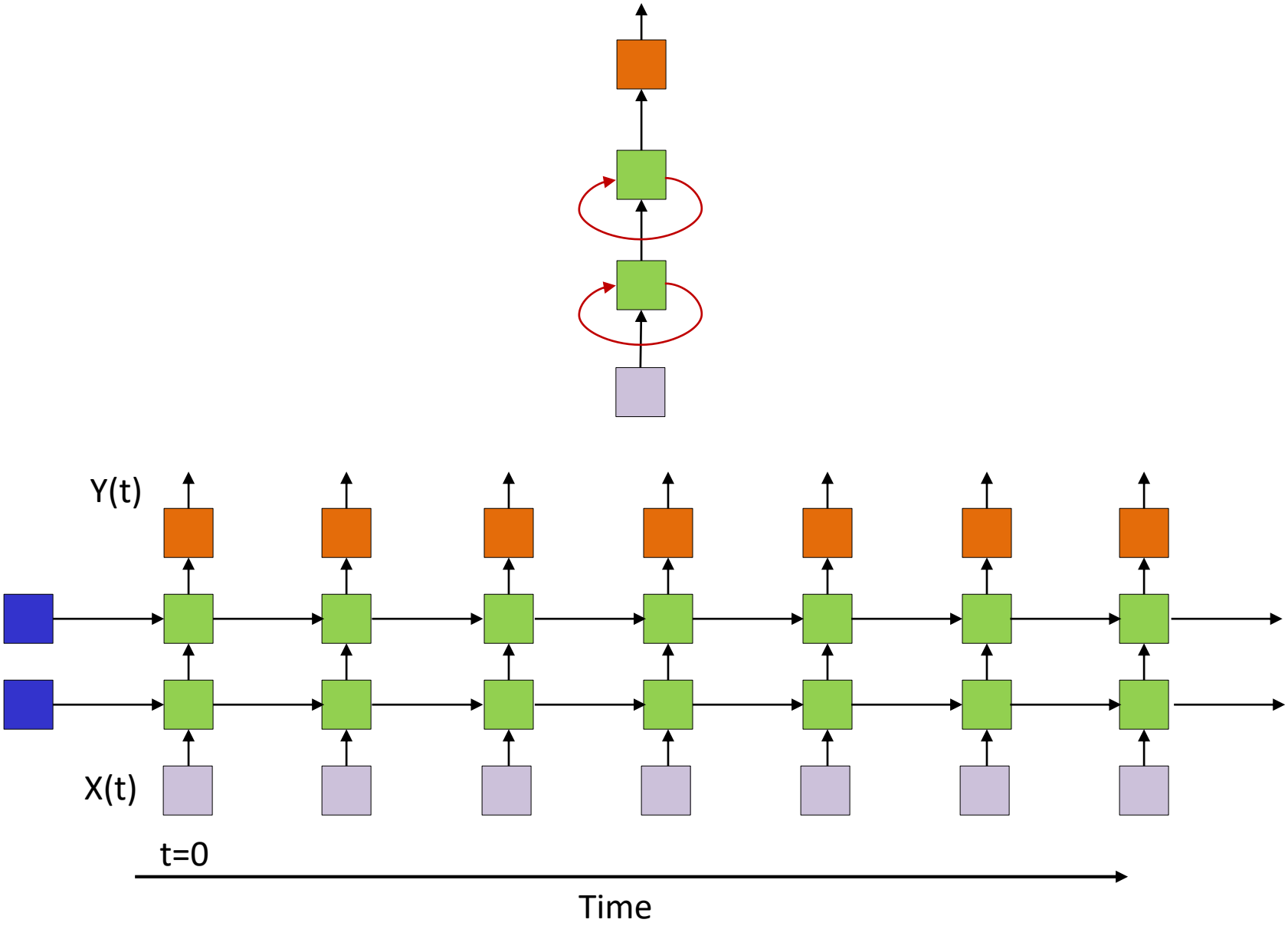


- Simplified models often drawn
- The loops imply recurrence

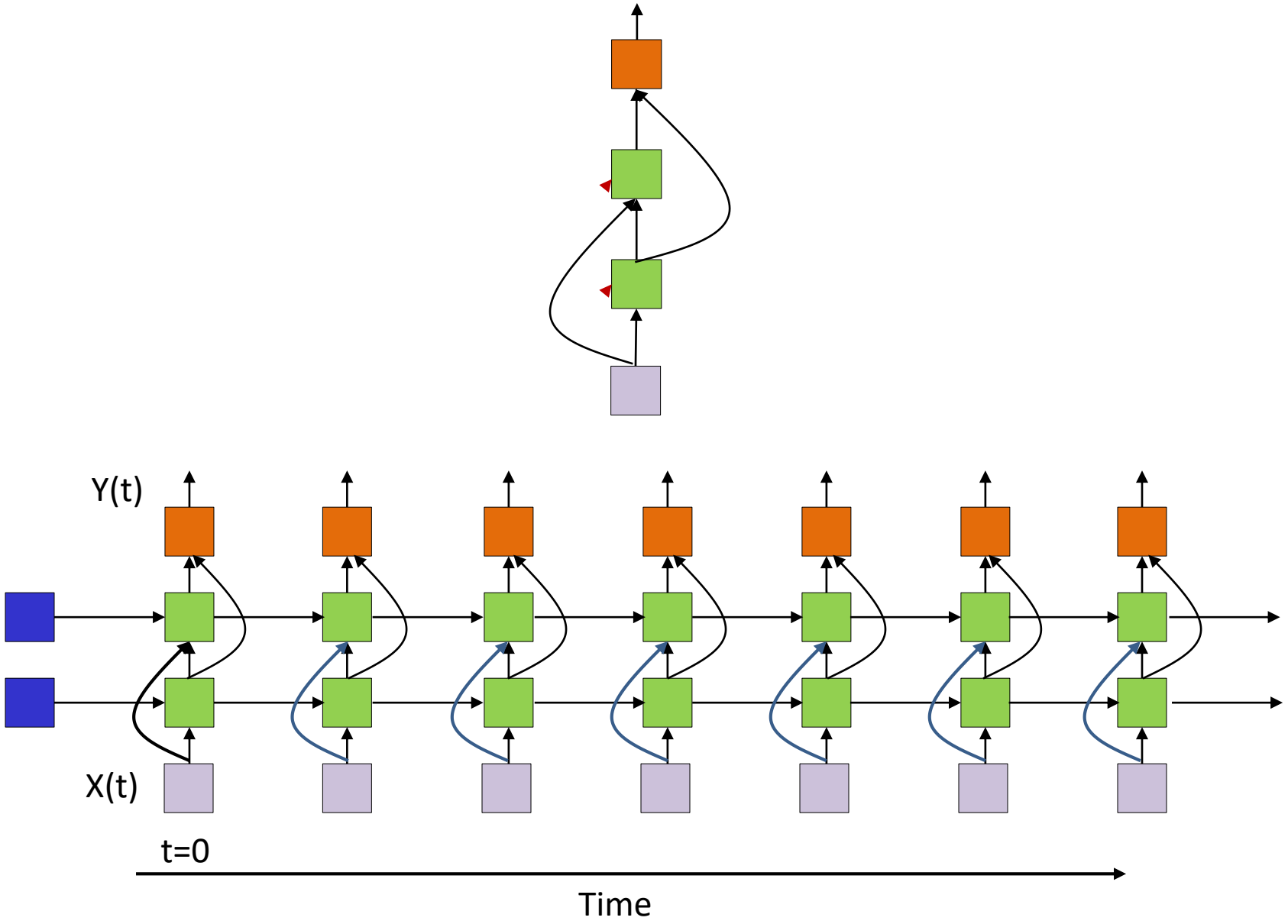
The detailed version of the simplified representation



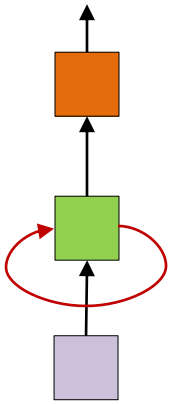
Multiple recurrent layer RNN



Multiple recurrent layer RNN



Equations



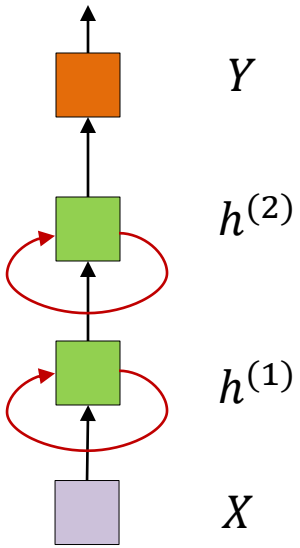
$$h_i^{(1)}(-1) = part\ of\ network\ parameters$$

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

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

- 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



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

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

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

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

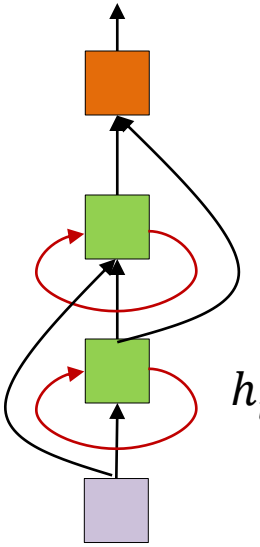
$$Y(t) = f_3\left(\sum_j w_{jk}^{(2)} h_j^{(2)}(t) + b_k^{(3)}, k = 1..M\right)$$

- 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

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

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

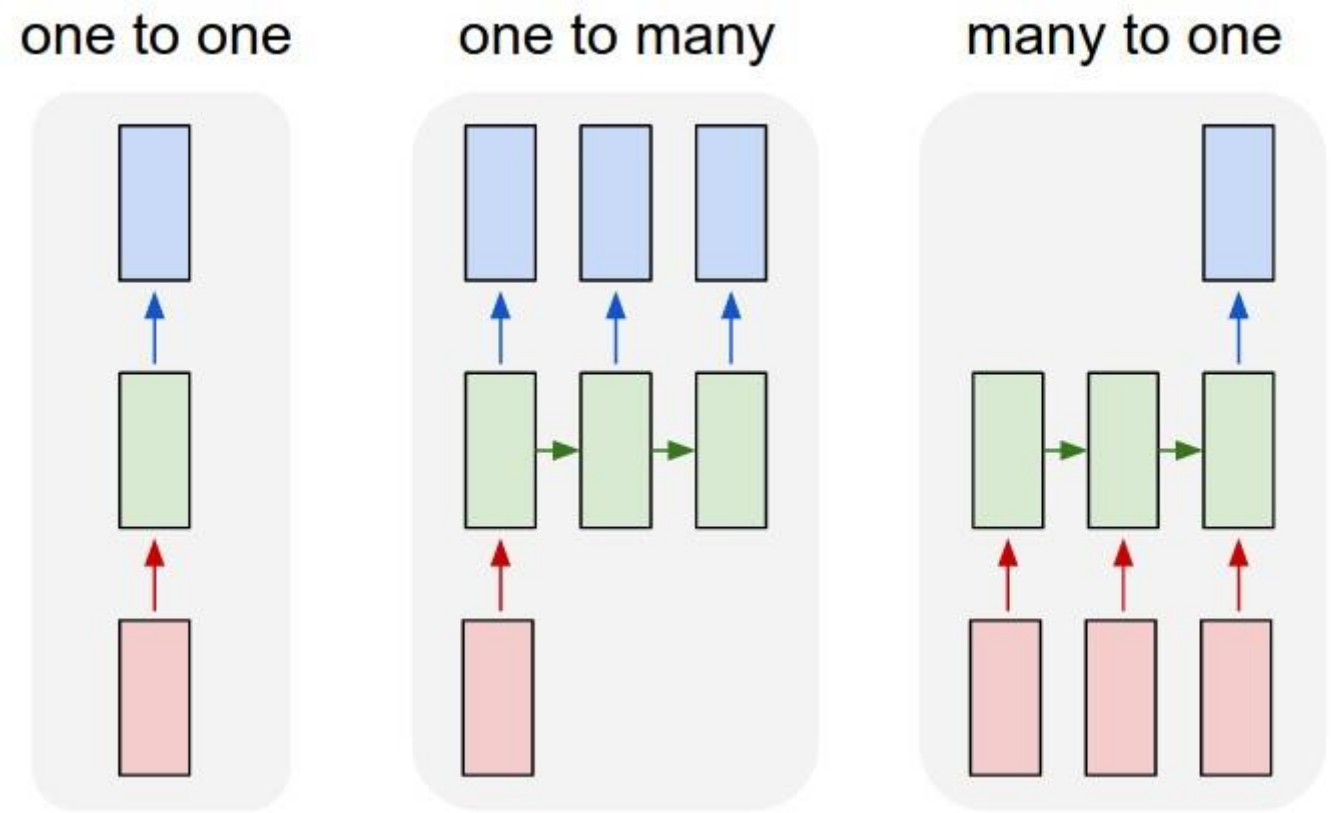


$$h_i^{(1)}(t) = f_1\left(\sum_j w_{ji}^{(0,1)} X_j(t) + \sum_i w_{ii}^{(1,1)} h_i^{(1)}(t-1) + b_i^{(1)}\right)$$

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

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

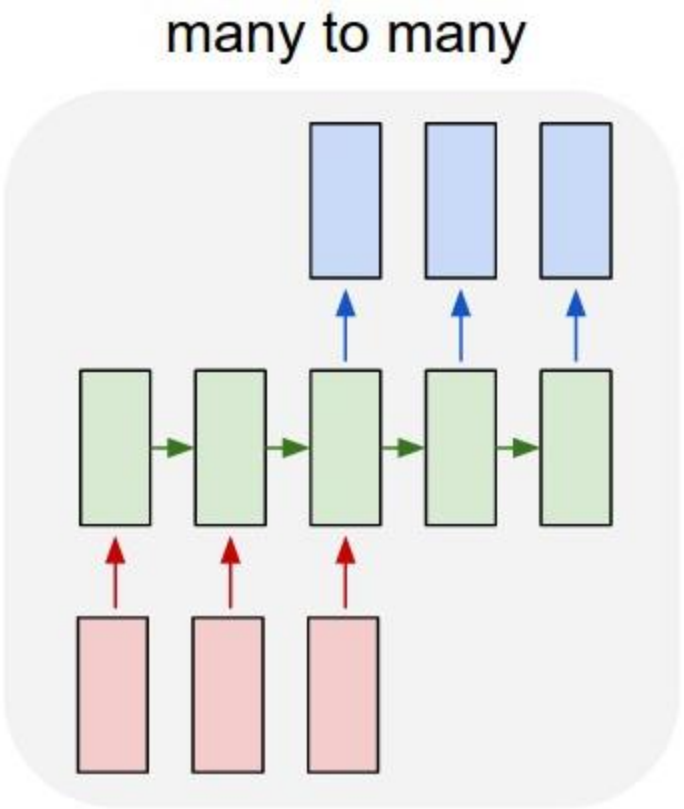
Variants on recurrent nets

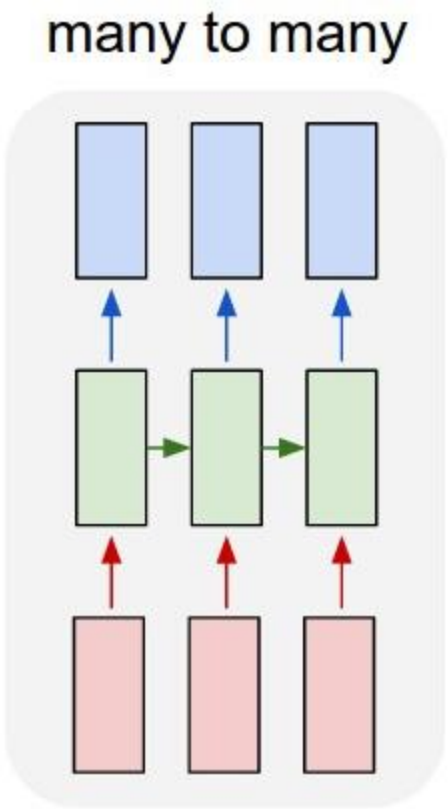


Images from Karpathy

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

Variants

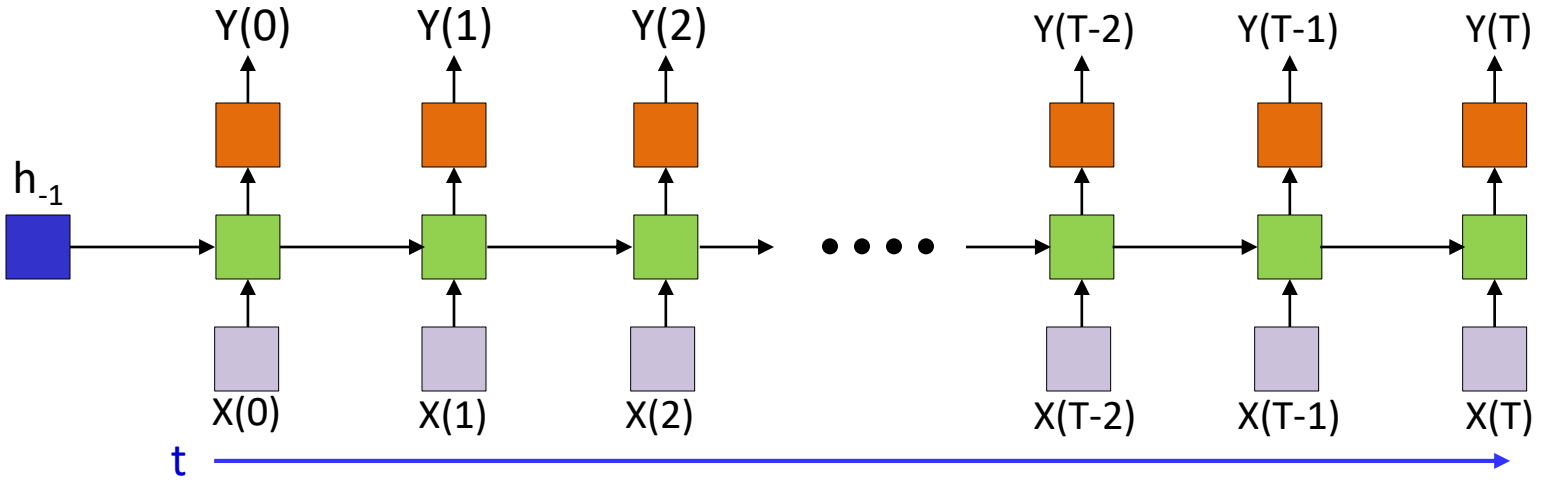




Images from Karpathy

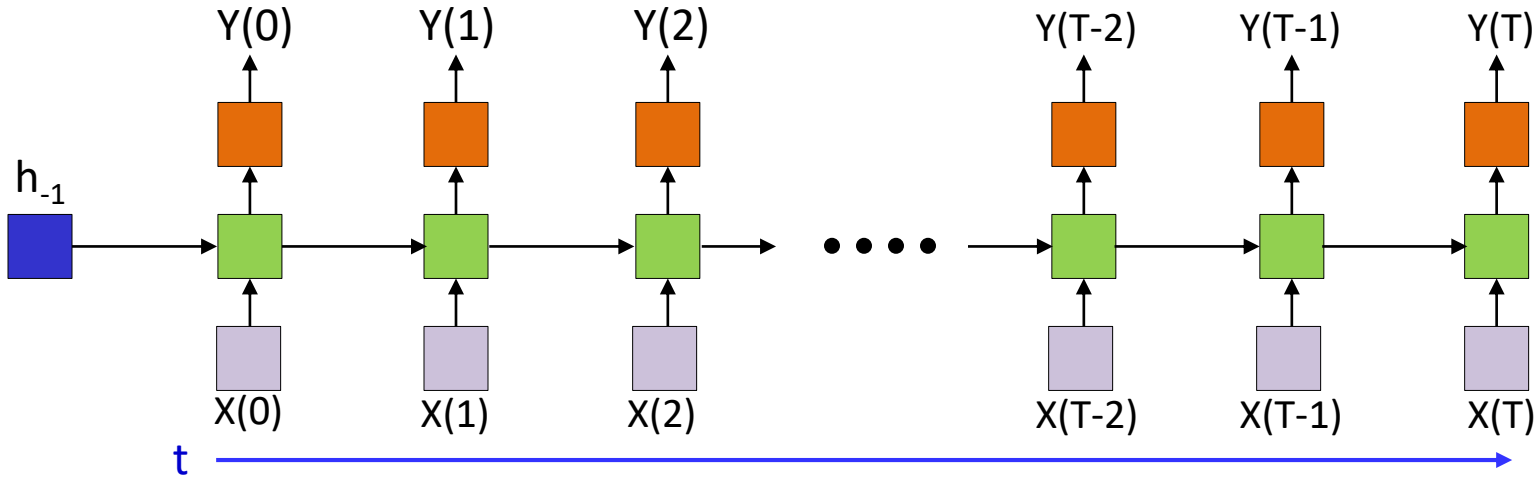
- 2: Sequence to sequence, e.g. stock problem, label prediction
- 1: *Delayed* sequence to sequence
- Etc…

How do we *train* the network



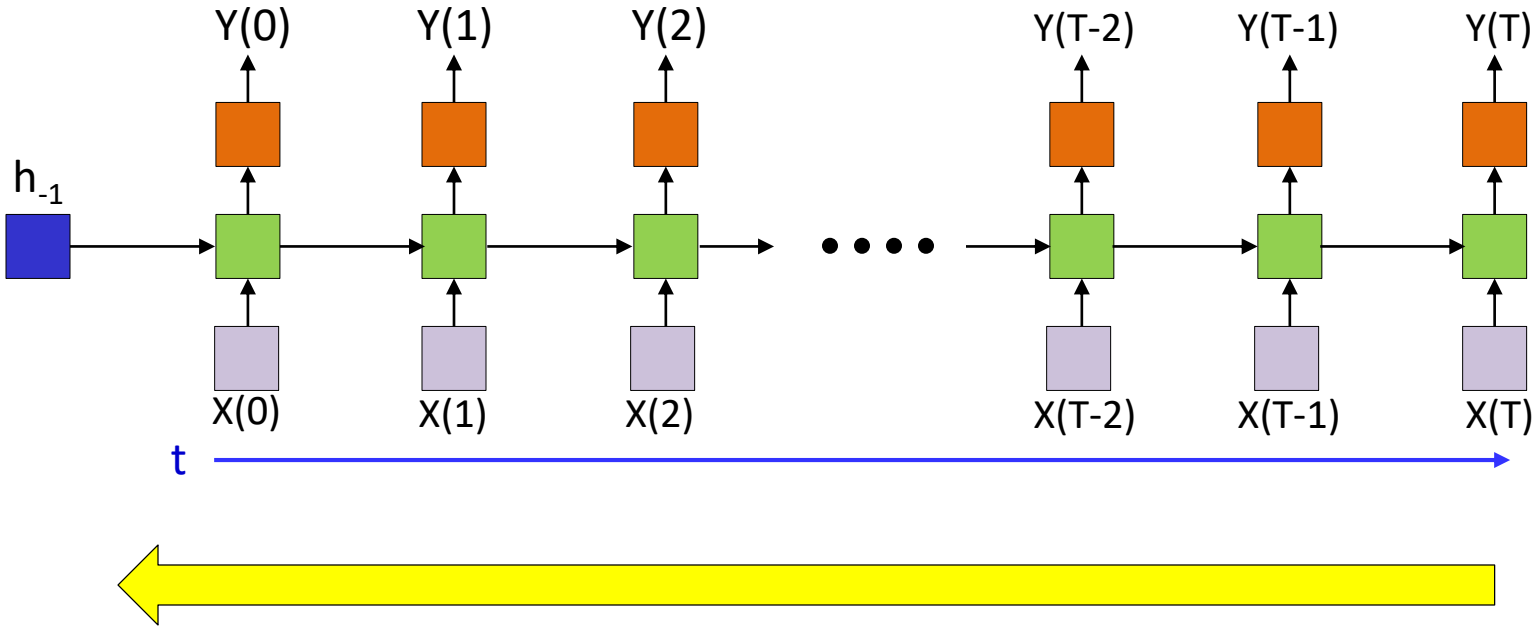
- Back propagation through time (BPTT)
- Given a collection of *sequence* inputs
  - $(\mathbf{X}_i, \mathbf{D}_i)$, where
  - $\mathbf{X}_i = X_{i,0}, \ldots, X_{i,T}$
  - $\mathbf{D}_i = D_{i,0}, \ldots, D_{i,T}$
- Train network parameters to minimize the error between the output of the network $\mathbf{Y}_i = Y_{i,0}, \ldots, Y_{i,T}$ 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



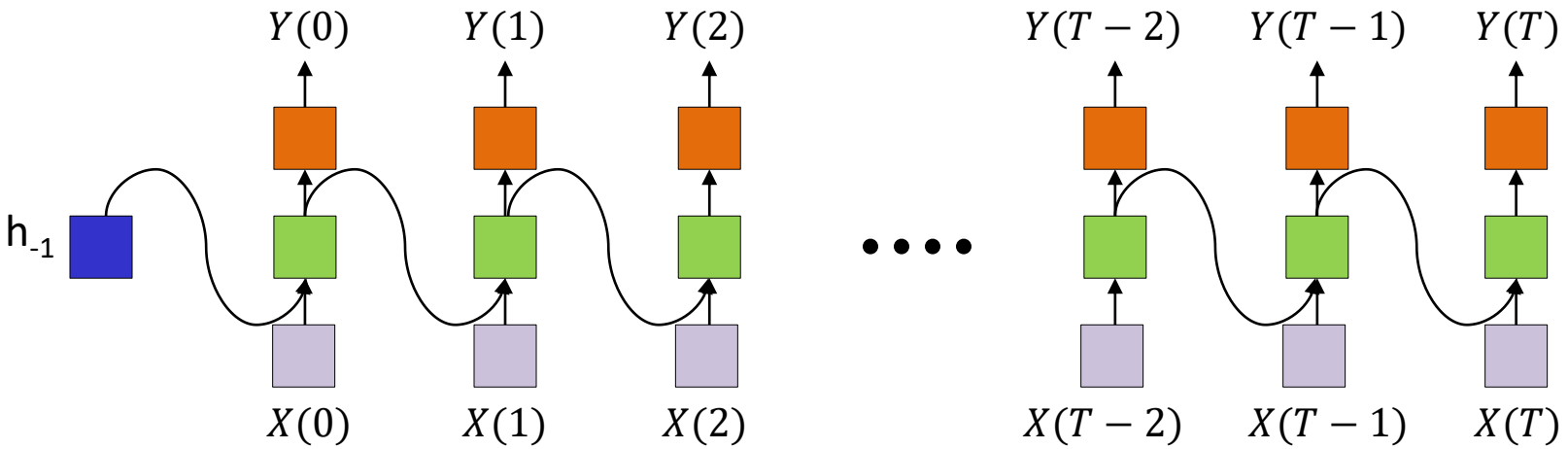
- For each training input:
- Forward pass: pass the entire data sequence through the network, generate outputs

Training: Computing gradients



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

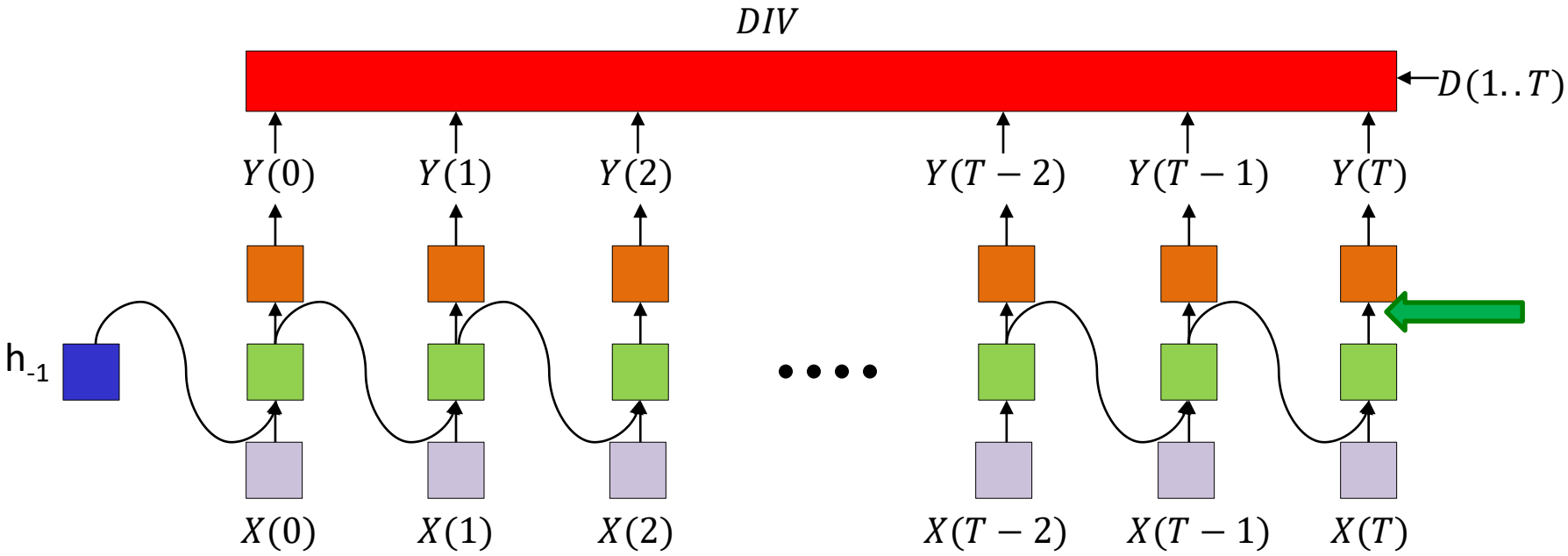
Back Propagation Through Time



Will only focus on *one* training instance

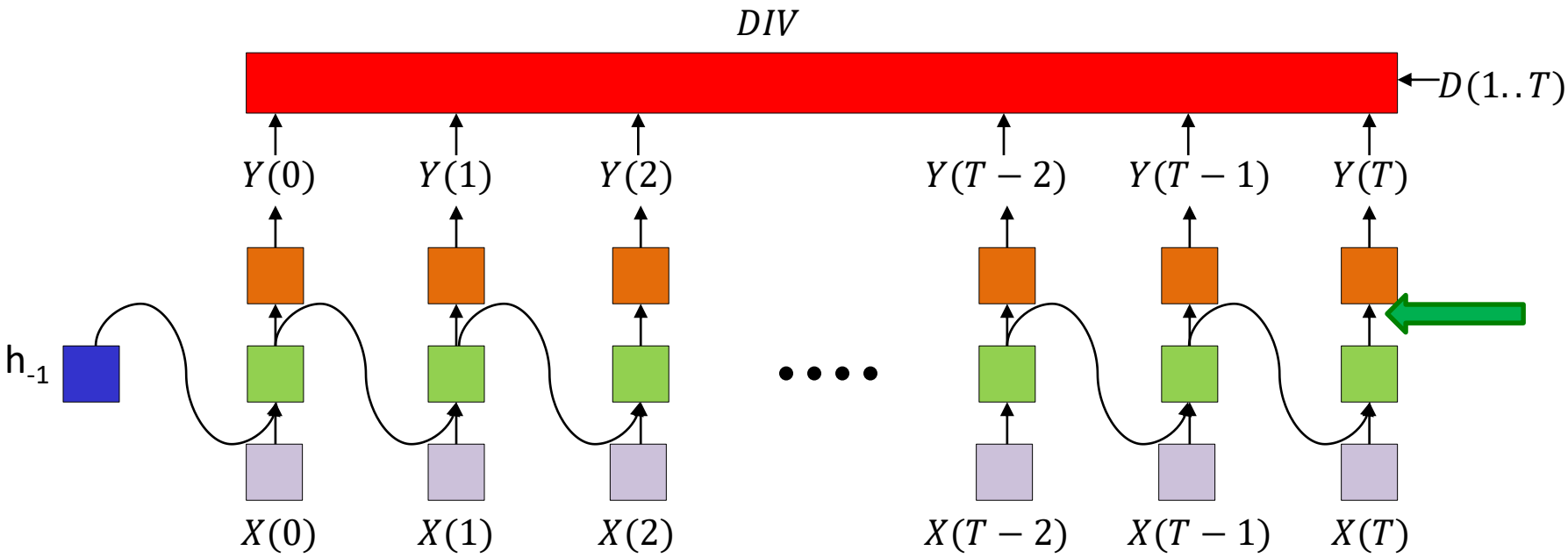
All subscripts represent *components* and not training instance index

Back Propagation Through Time



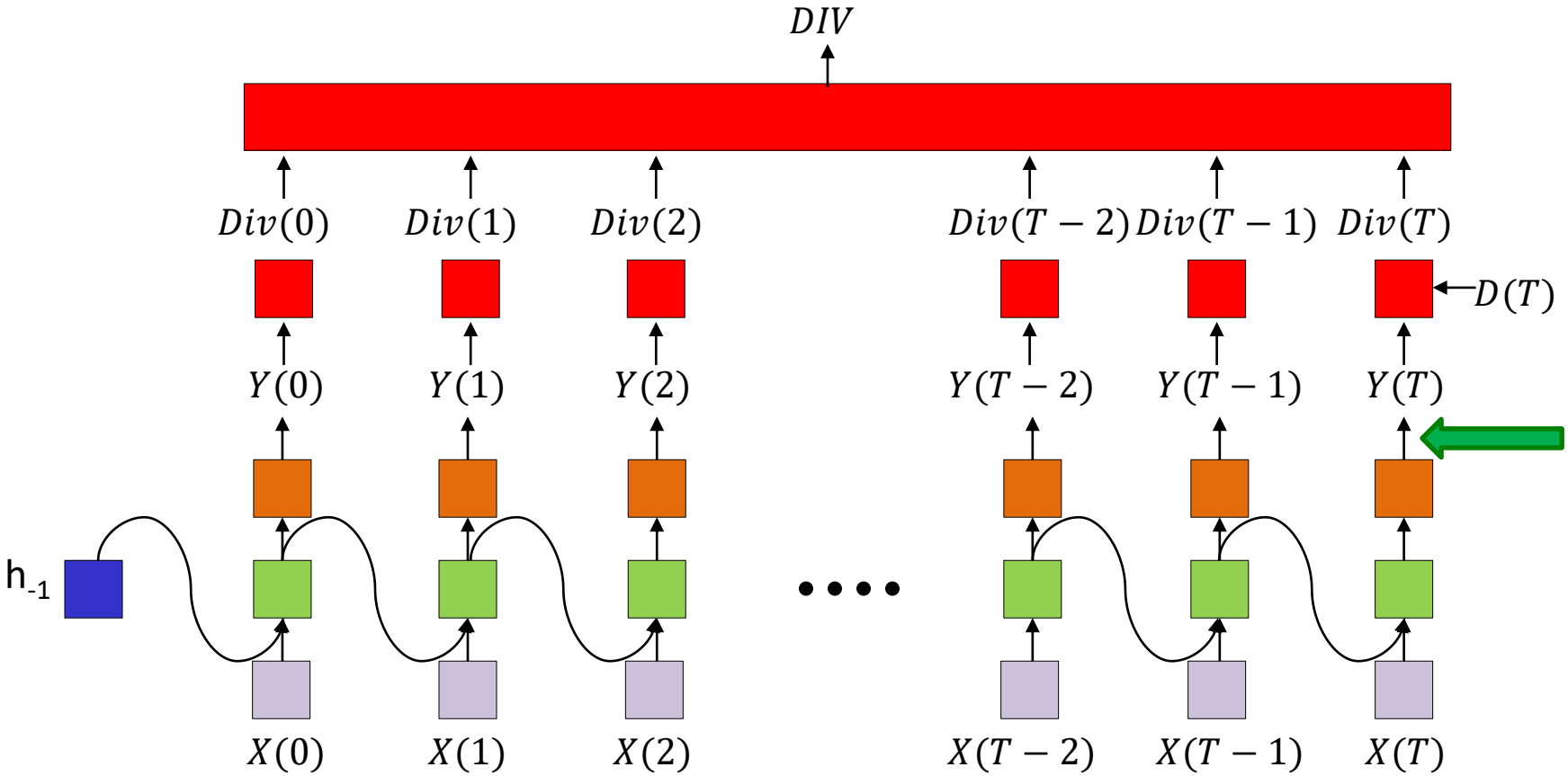
- The divergence computed is between the *sequence of outputs* by the network and the *desired sequence of outputs*
- This is *not* just the sum of the divergences at individual times
  - Unless we explicitly define it that way

Back Propagation Through Time



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

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:

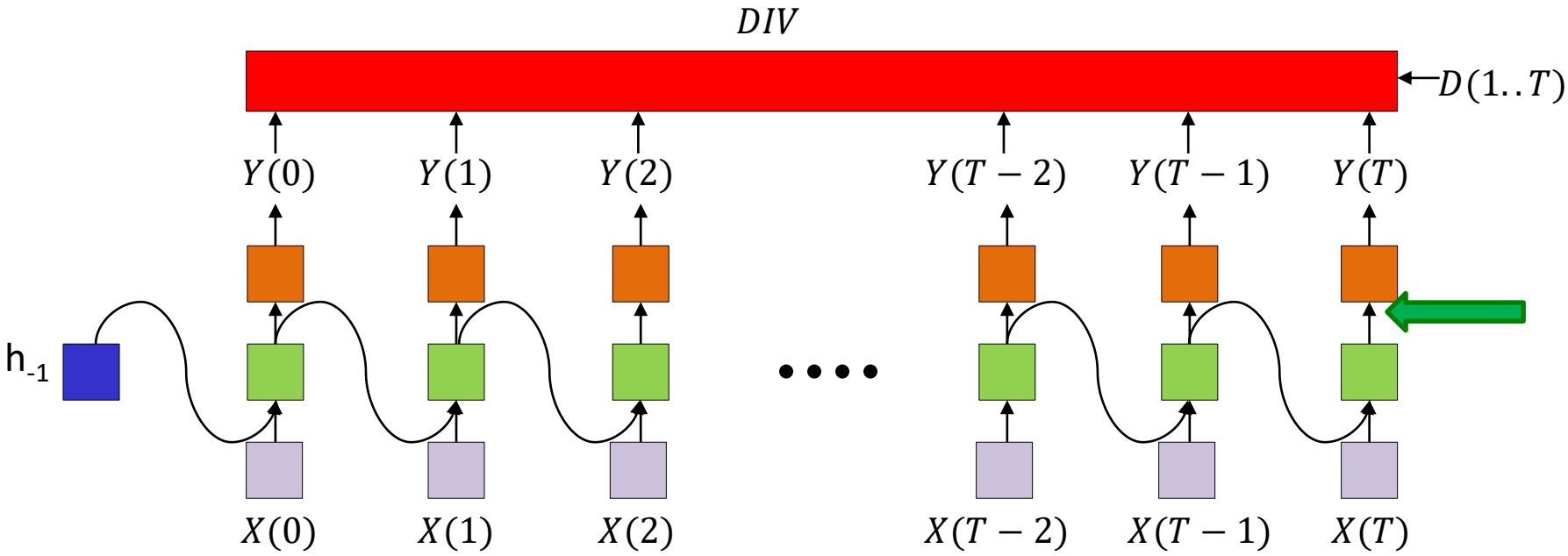
Must compute

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

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

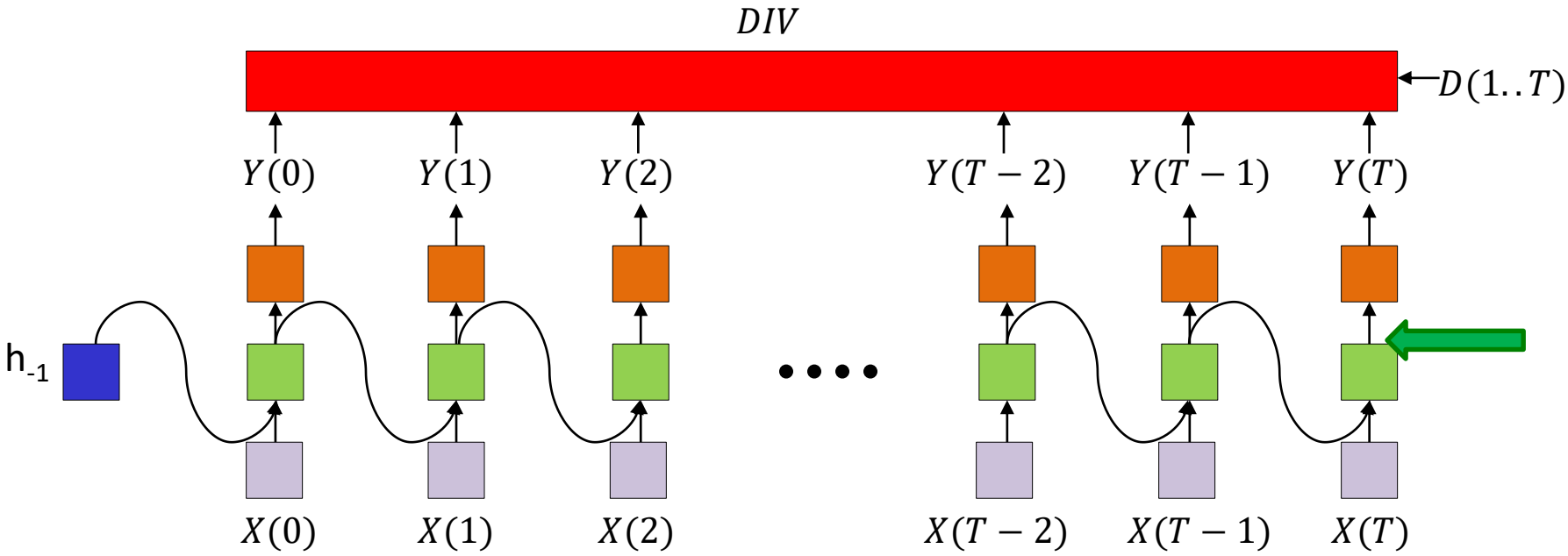
Vector output activation

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

OR

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

Back Propagation Through Time

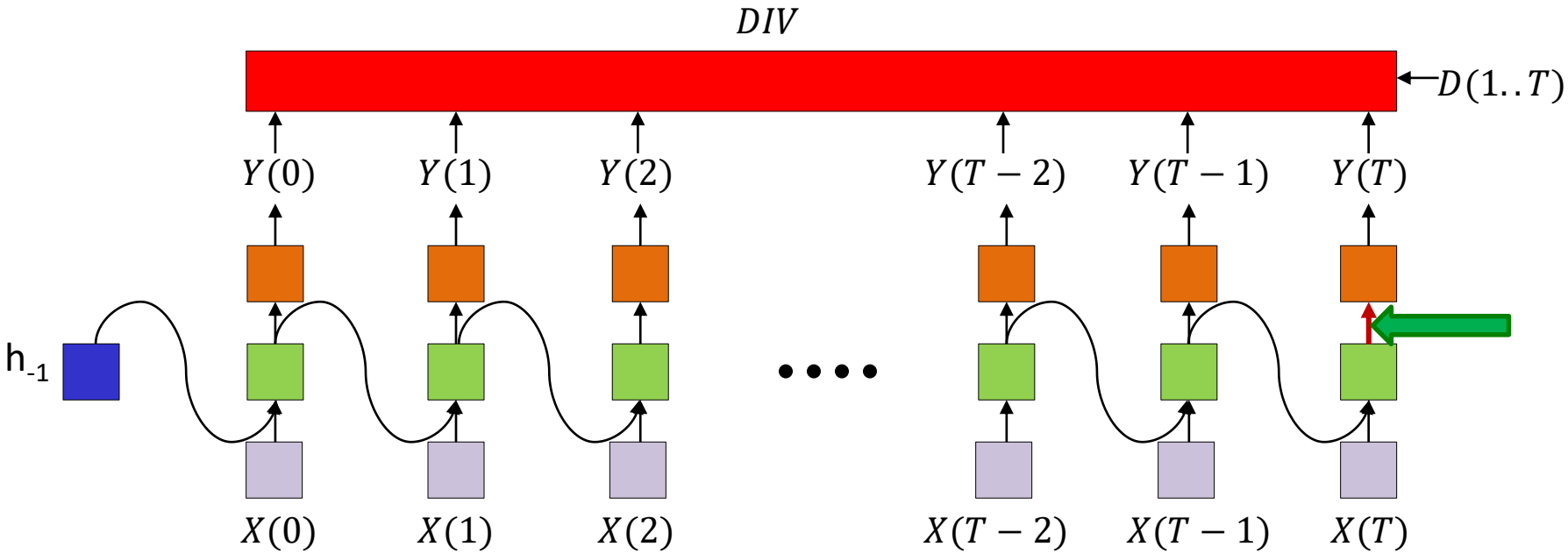


$$\frac{dDIV}{dh_i(T)} = \sum_j \frac{dDIV}{dZ_j^{(1)}(T)} \frac{dZ_j^{(1)}(T)}{dh_i(T)} = \sum_j w_{ij}^{(1)} \frac{dDIV}{dZ_j^{(1)}(T)}$$

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

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

Back Propagation Through Time

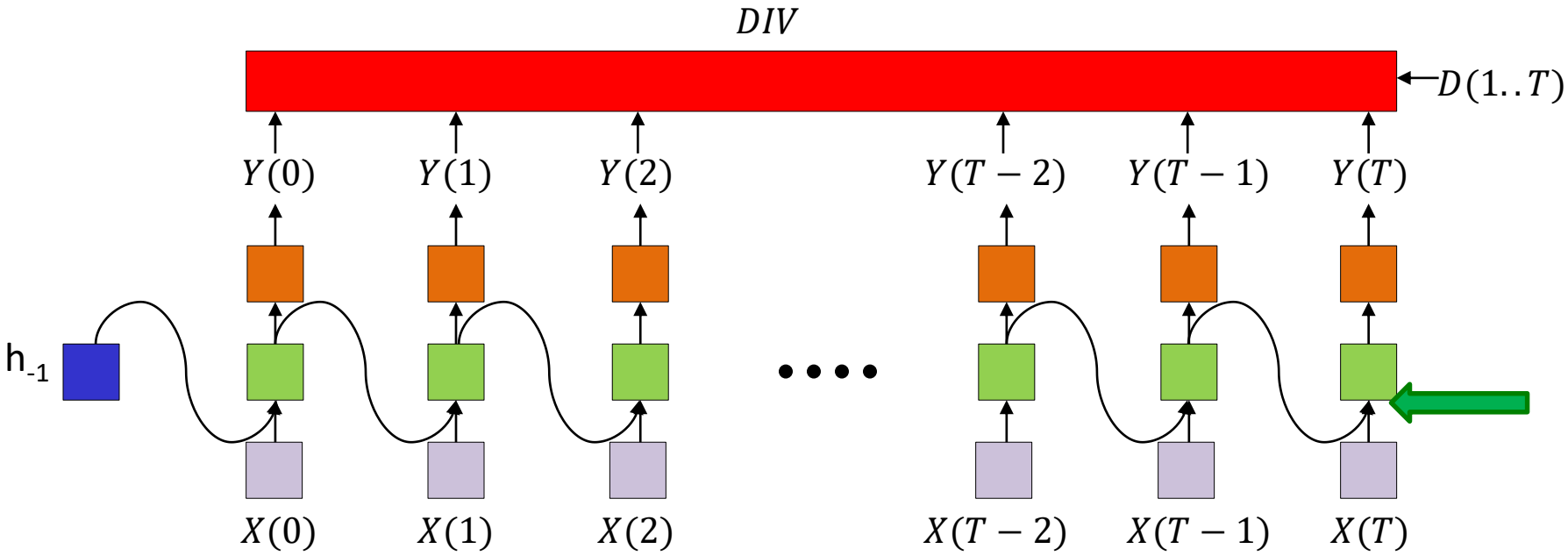


$$\frac{dDIV}{dZ_i^{(1)}(T)} = \frac{dDiv(T)}{dY_i(T)} \frac{dY_i(T)}{dZ_i^{(1)}(T)}$$

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

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

Back Propagation Through Time



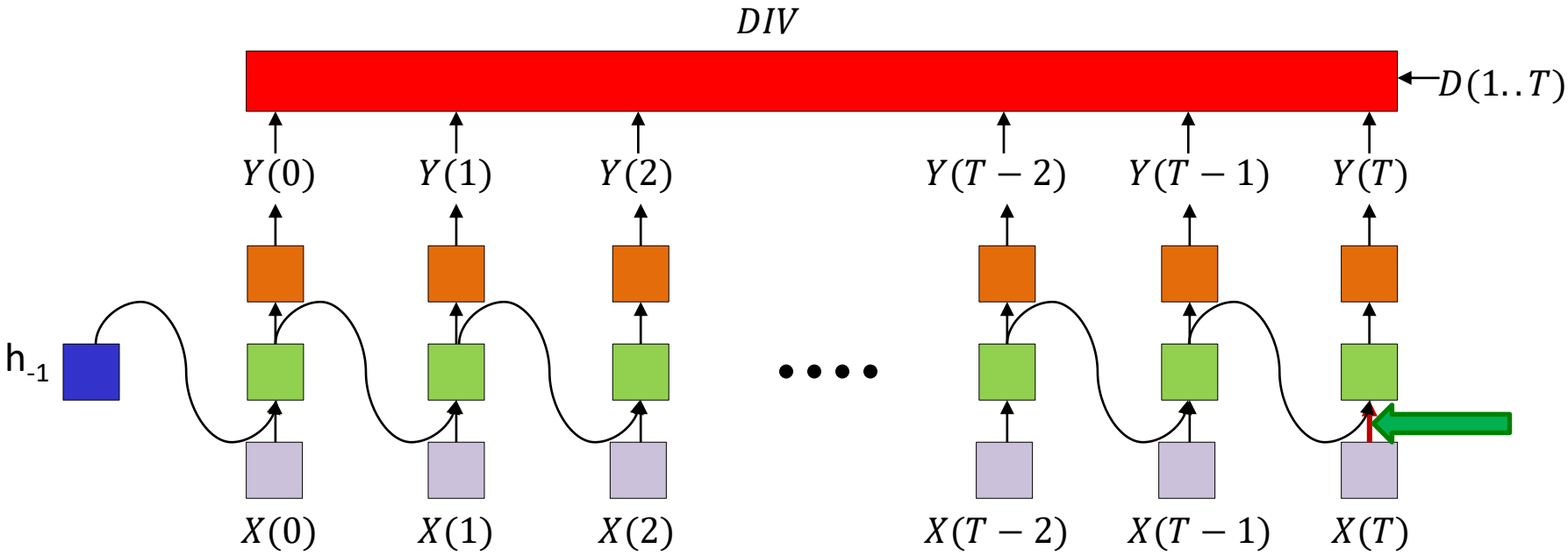
$$\frac{dDIV}{dZ_i^{(0)}(T)} = \frac{dDIV}{dh_i(T)} \frac{dh_i(T)}{dZ_i^{(0)}(T)}$$

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

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

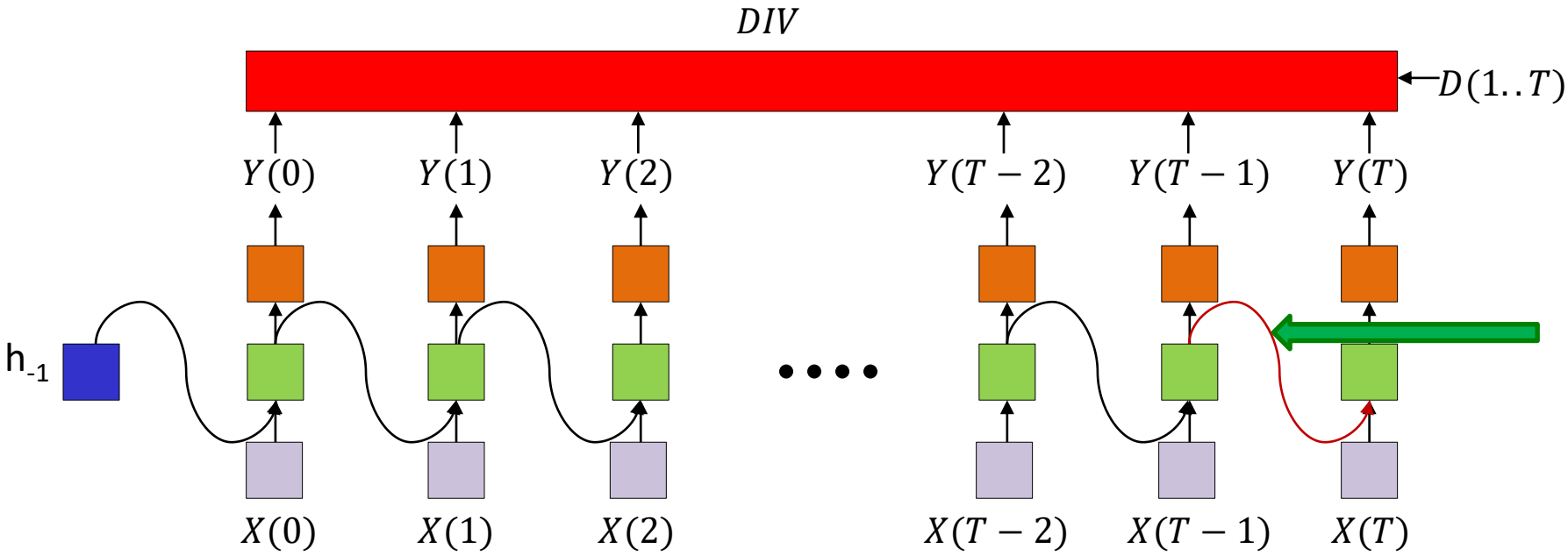
$$\frac{dDIV}{dw_{ij}^{(1)}} = \frac{dDIV}{dZ_j^{(1)}(T)} h_i(T)$$

Back Propagation Through Time



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

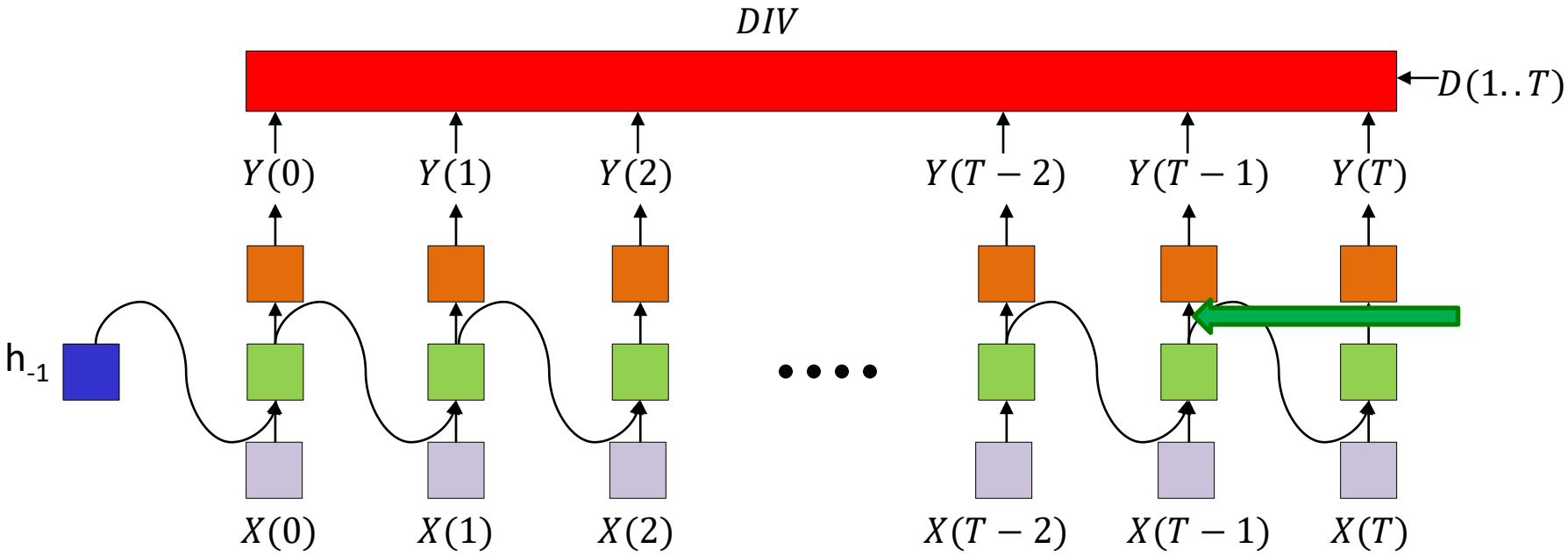
Back Propagation Through Time



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

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

Back Propagation Through Time



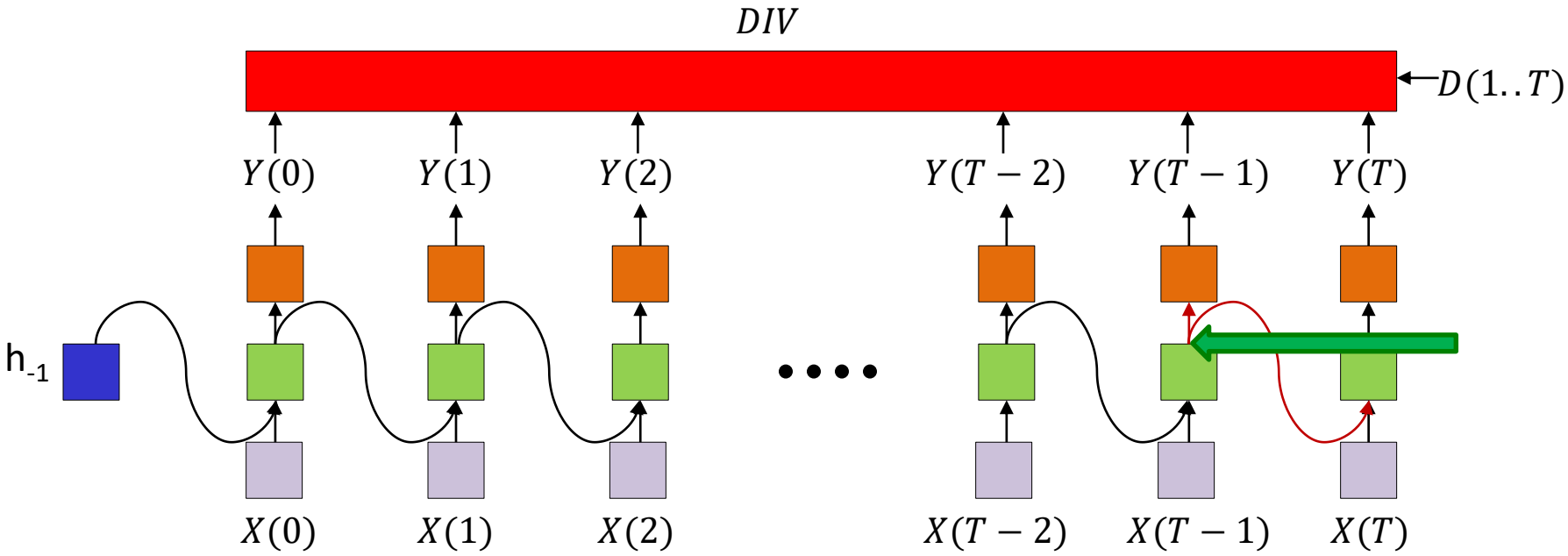
Vector output activation

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

OR

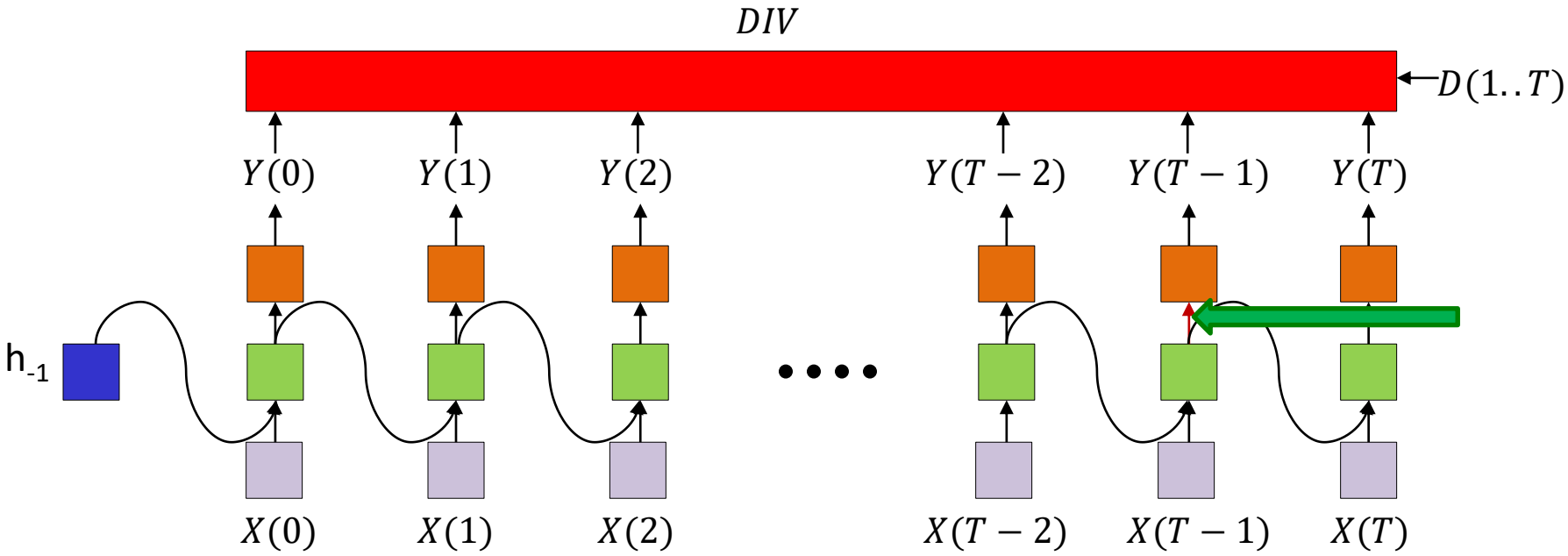
$$\frac{dDIV}{dZ_i^{(1)}(T-1)} = \sum_j \frac{dDIV}{dY_j(T-1)}\frac{dY_j(T-1)}{dZ_i^{(1)}(T-1)}$$

Back Propagation Through Time



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

Back Propagation Through Time

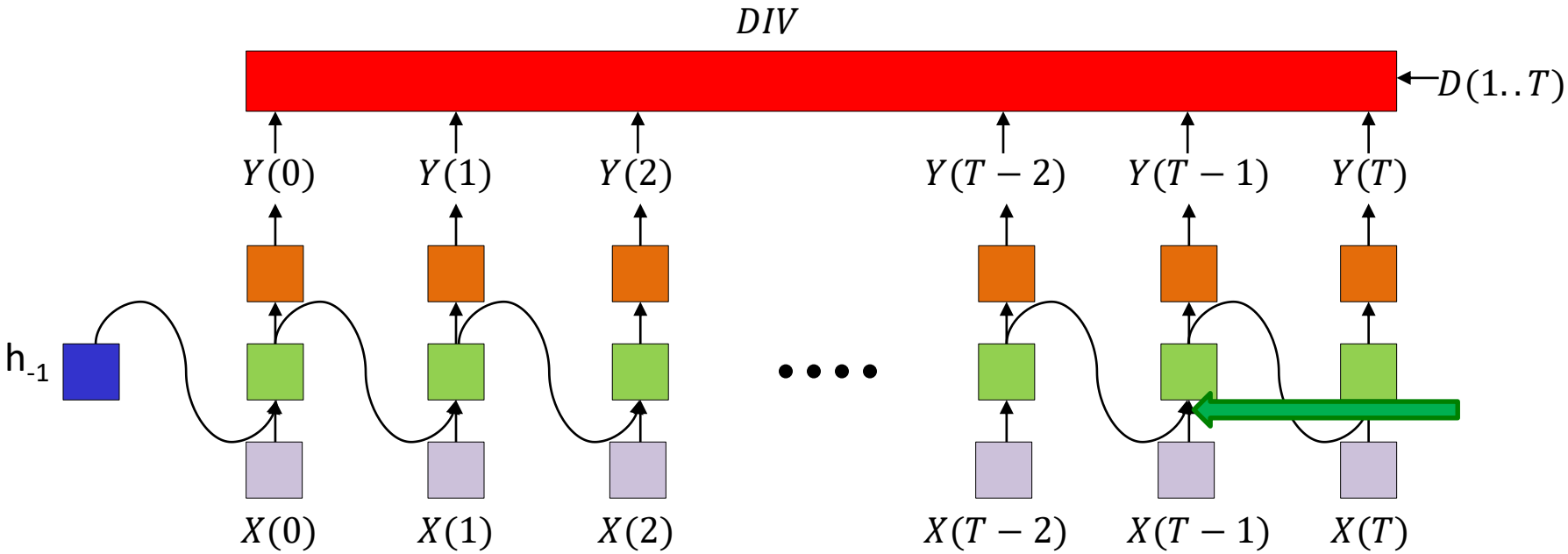


$$\frac{dDIV}{dh_i(T-1)} = \sum_j w_{ij}^{(1)} \frac{dDIV}{dZ_j^{(1)}(T-1)} + \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(0)}(T)}$$

Note the addition

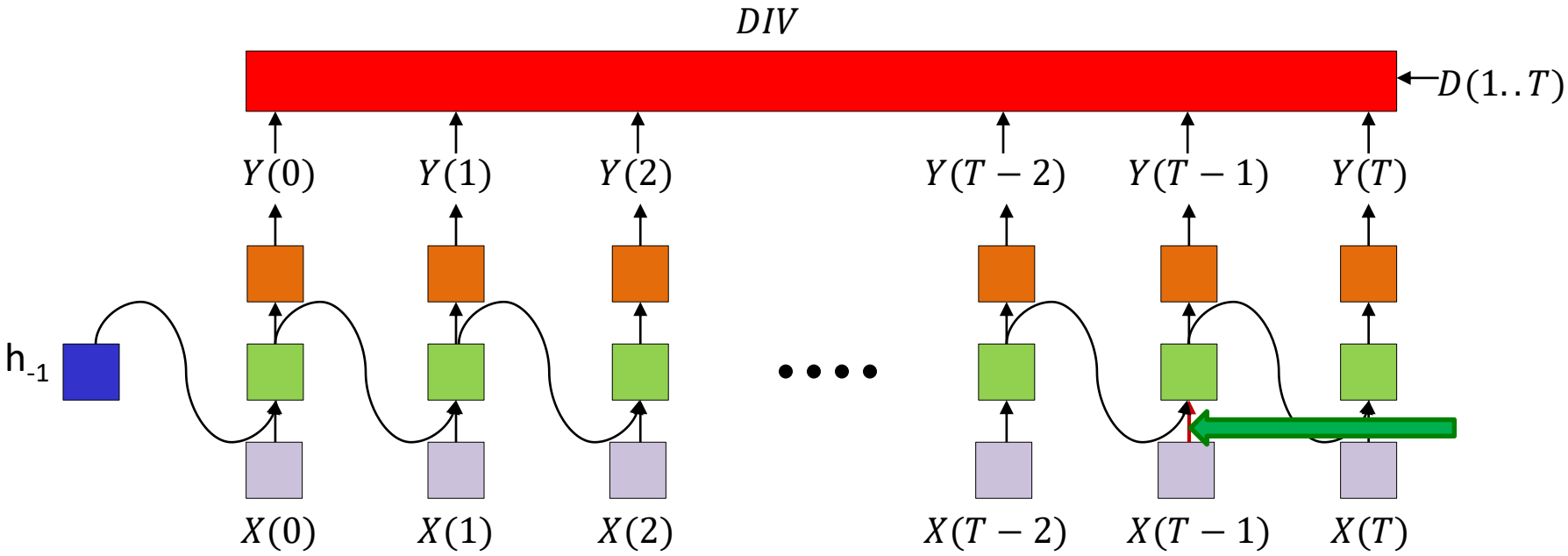
$$\frac{dDIV}{dw_{ij}^{(1)}} \mathrel{+}= \frac{dDIV}{dZ_j^{(1)}(T-1)} h_i(T-1)$$

Back Propagation Through Time



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

Back Propagation Through Time

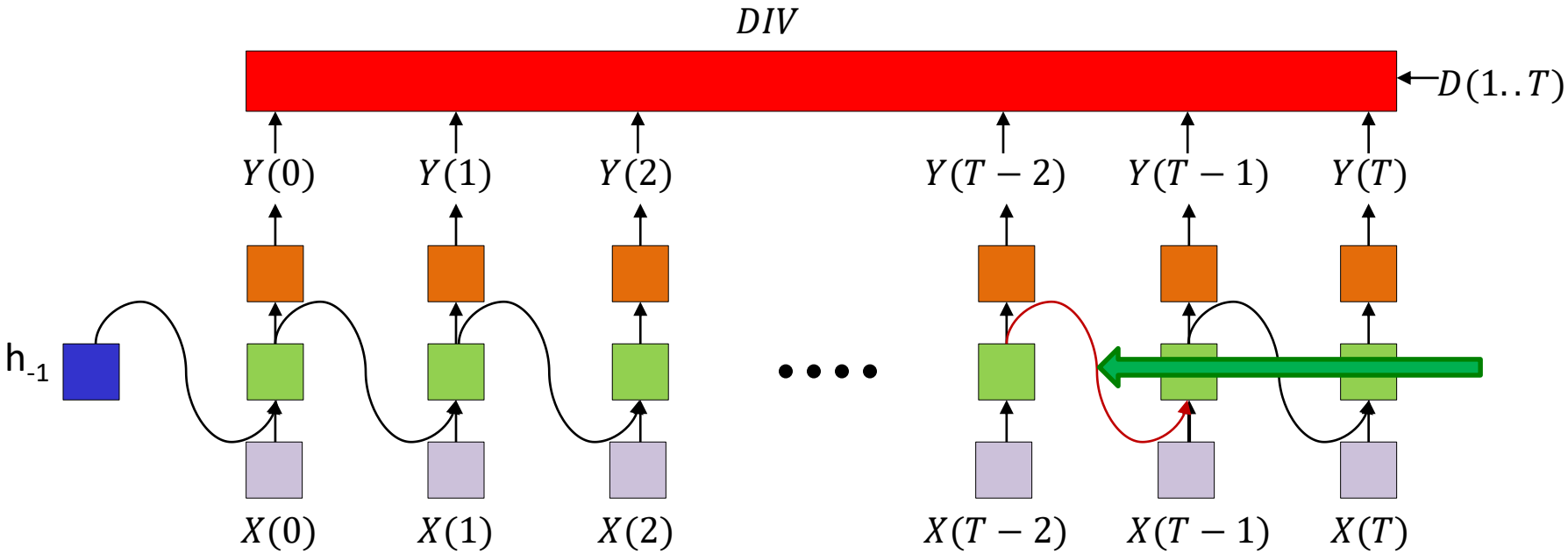


$$\frac{dDIV}{dZ_i^{(0)}(T-1)} = \frac{dDIV}{dh_i(T-1)} \frac{dh_i(T-1)}{dZ_i^{(0)}(T-1)}$$

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

Note the addition

Back Propagation Through Time

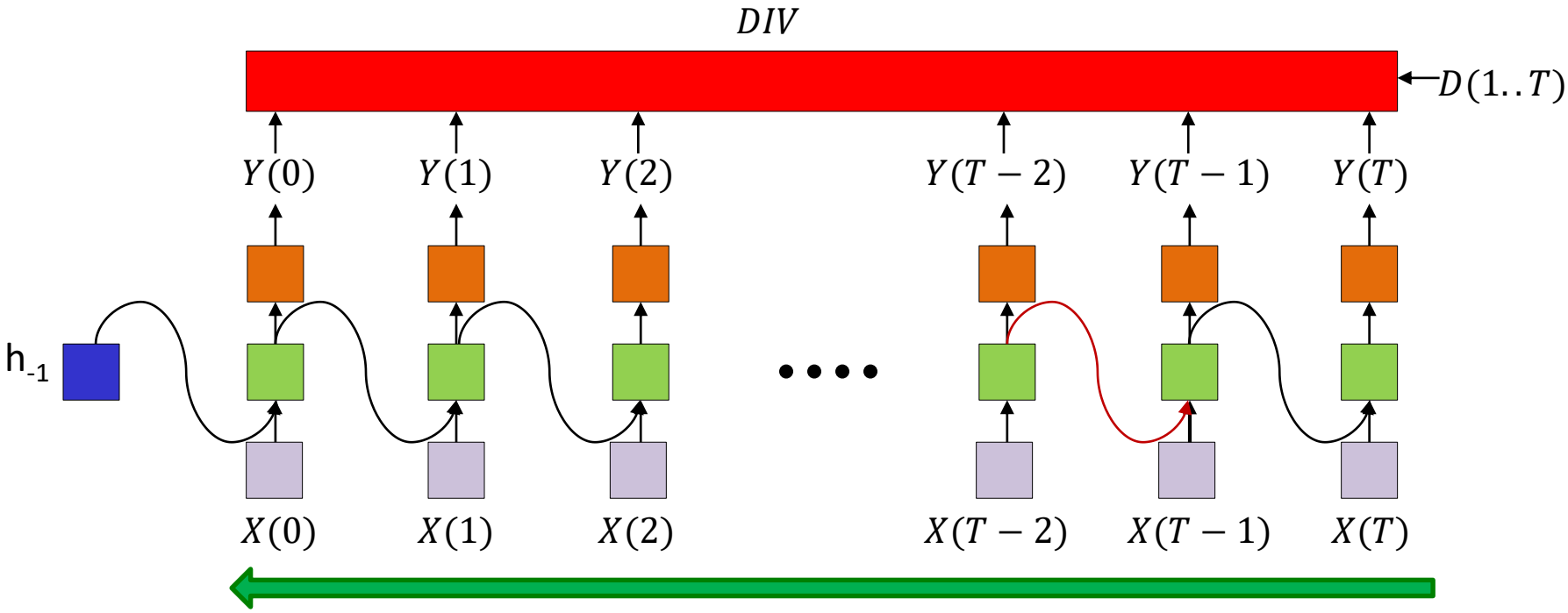


$$\frac{dDIV}{dw_{ij}^{(0)}} \mathrel{+}= \frac{dDIV}{dZ_j^{(0)}(T-1)} X_i(T-1)$$

Note the addition

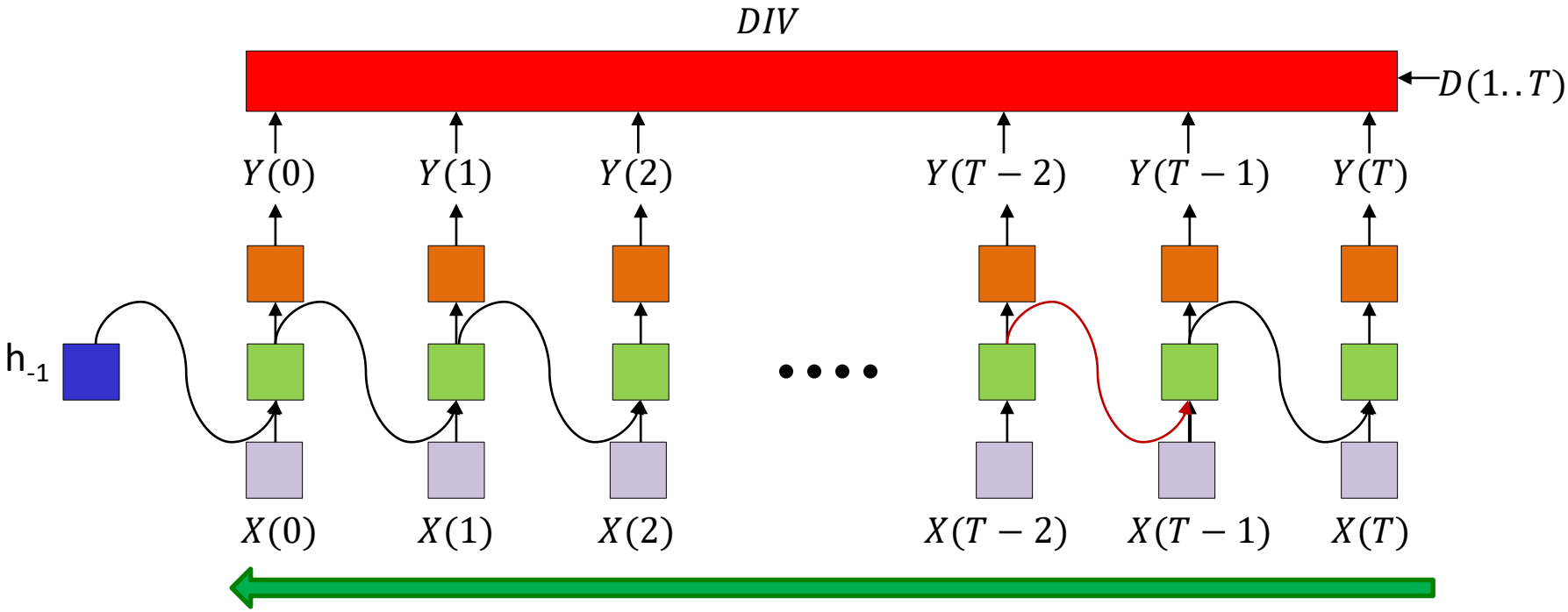
$$\frac{dDIV}{dw_{ij}^{(11)}} \mathrel{+}= \frac{dDIV}{dZ_j^{(0)}(T-1)} h_i(T-2)$$

Back Propagation Through Time



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

Back Propagation Through Time

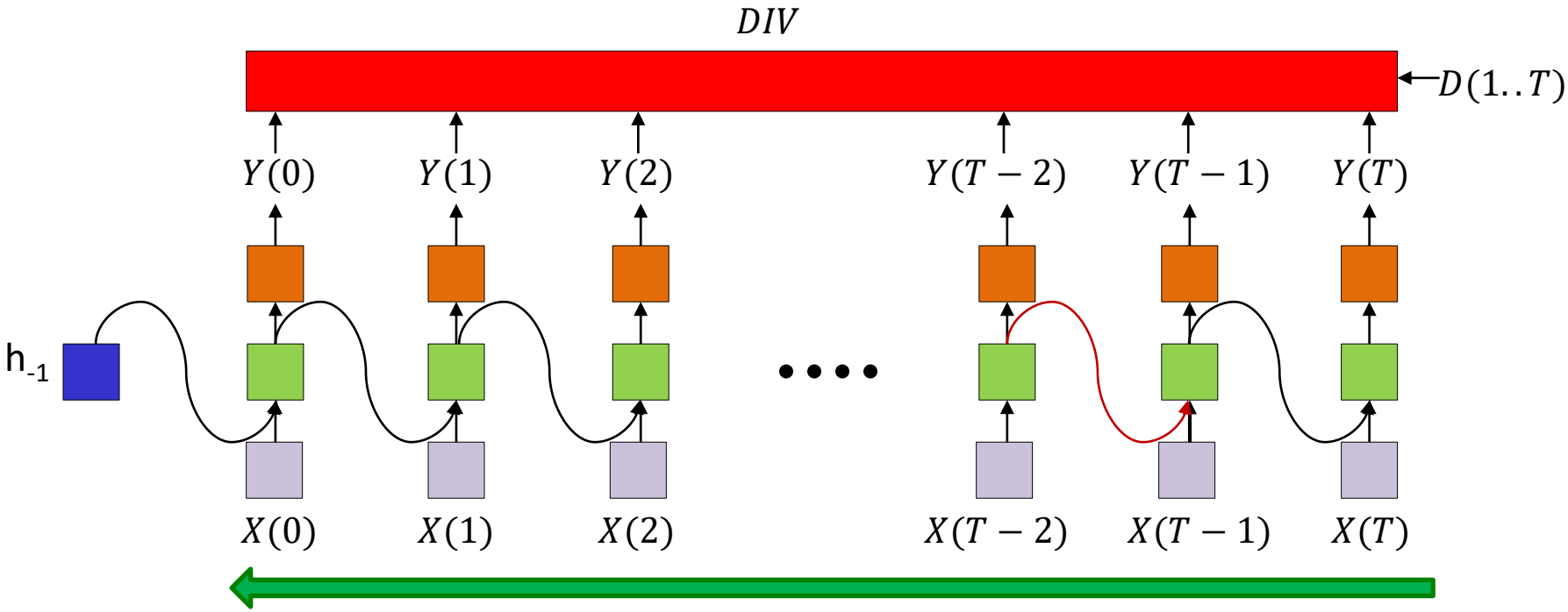


$$\frac{dDIV}{dh_i^{(k)}(t)} = \sum_j w_{i,j}^{(k)} \frac{dDIV}{dZ_j^{(k+1)}(t)} + \sum_j w_{i,j}^{(k,k)} \frac{dDIV}{dZ_j^{(k)}(t+1)}$$

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

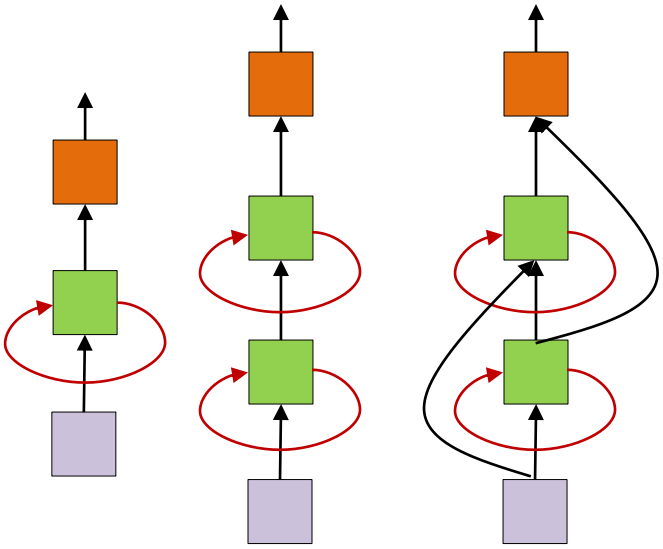


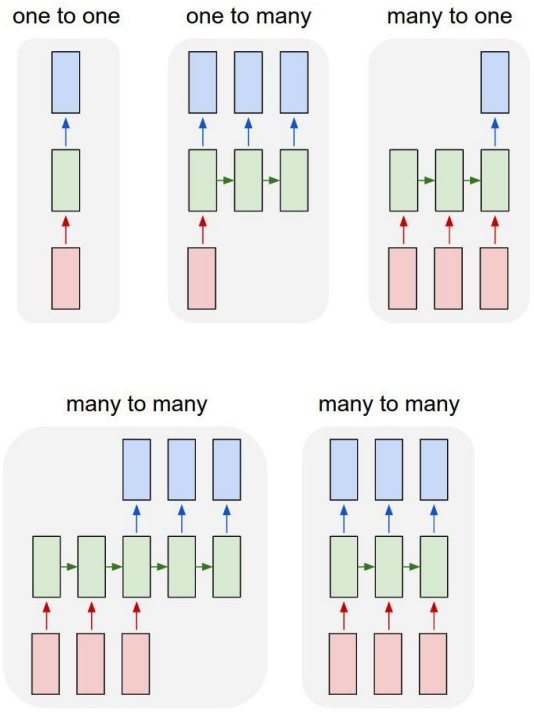
$$\frac{dDIV}{dh_{-1}} = \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(1)}(0)}$$

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

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

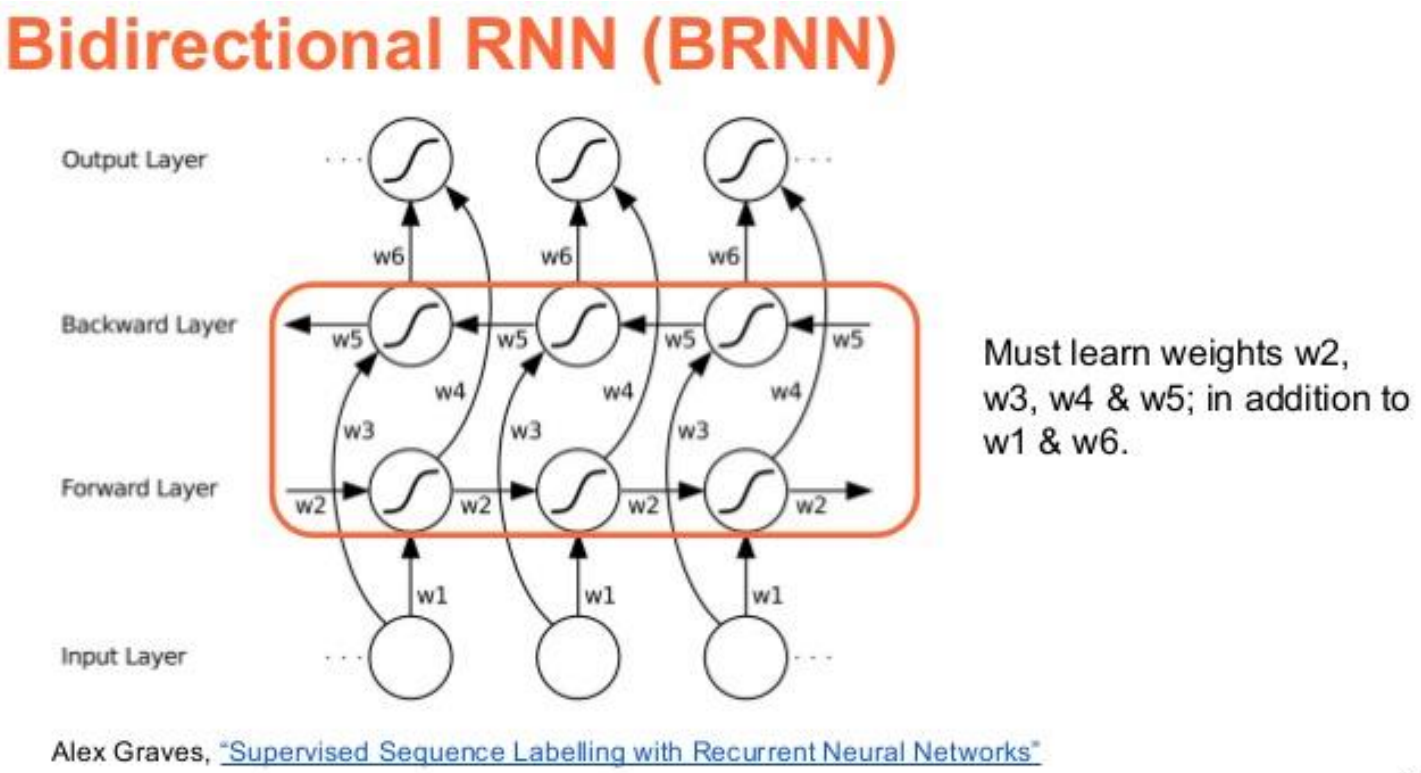
BPTT





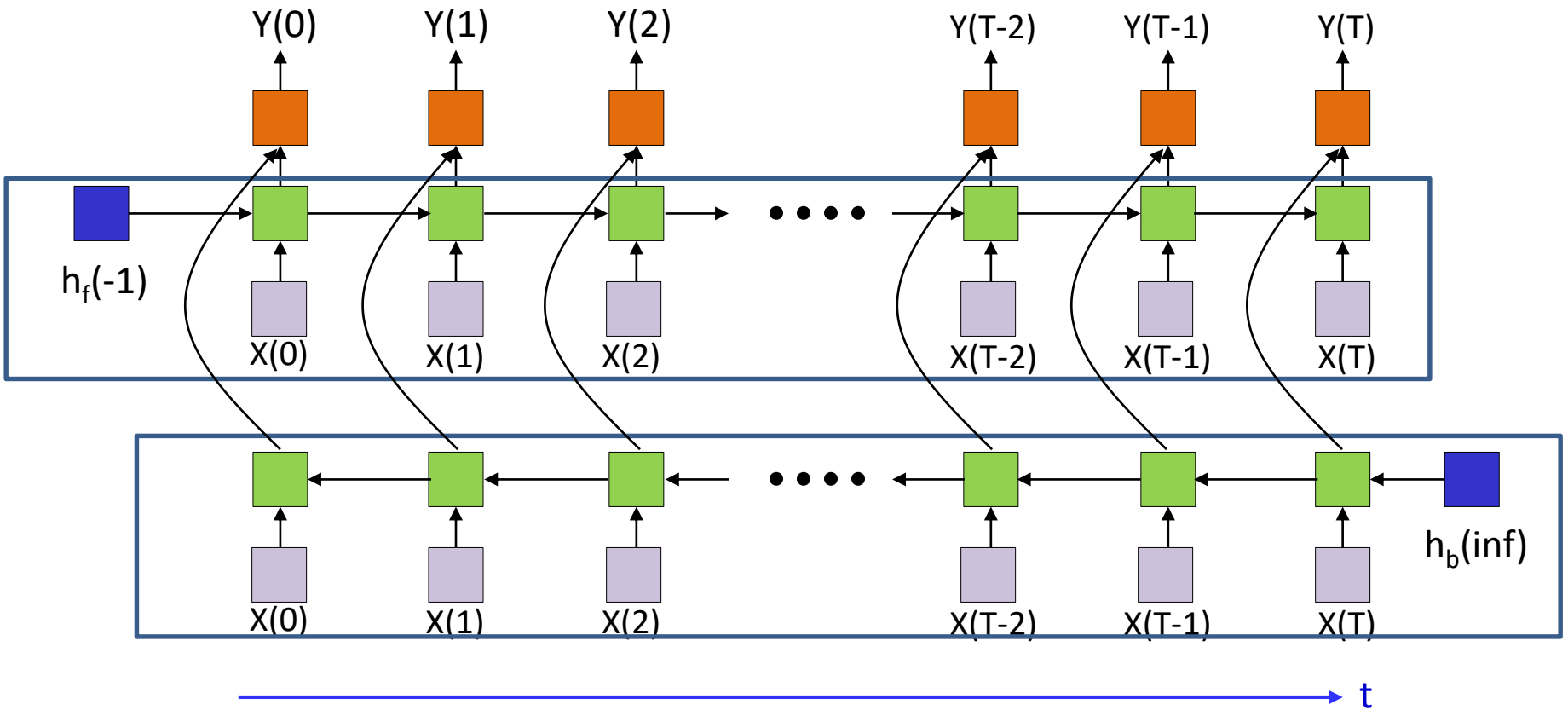
- Can be generalized to any architecture

Extensions to the RNN: *Bidirectional RNN*



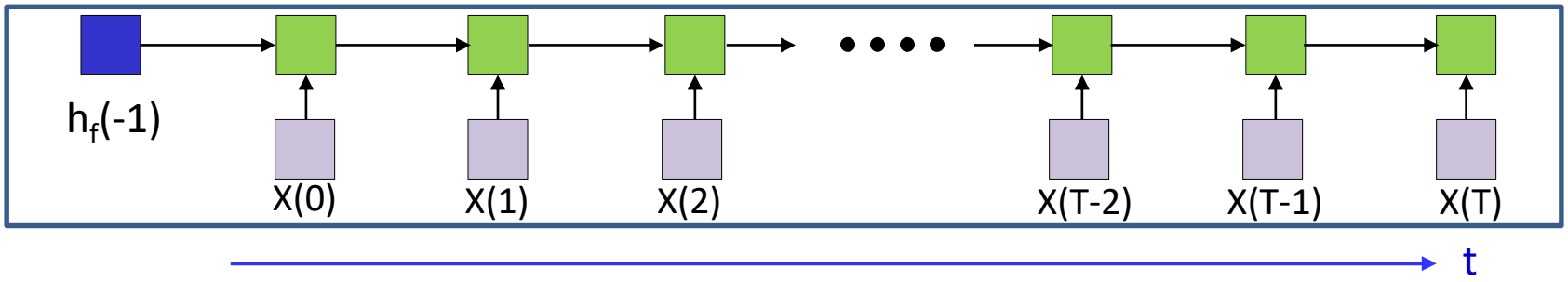
- RNN with both forward and backward recursion
  - Explicitly models the fact that just as the future can be predicted from the past, the past can be deduced from the future

Bidirectional RNN



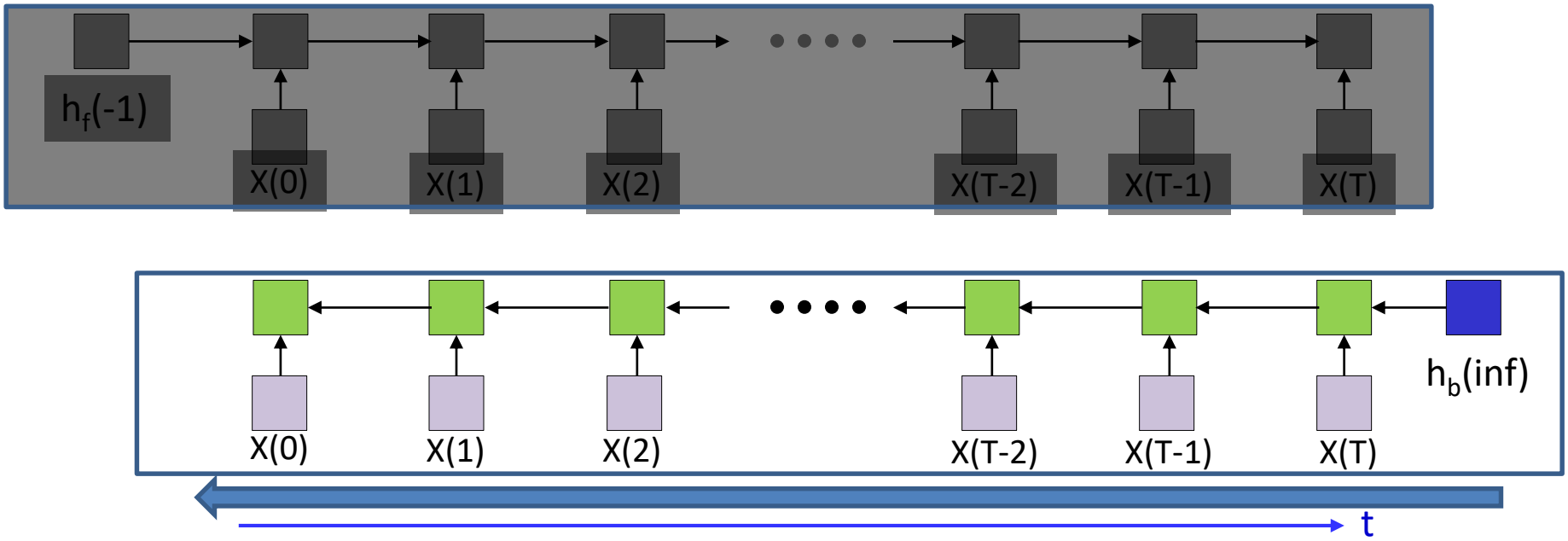
- A forward net process the data from t=0 to t=T
- A backward net processes it backward from t=T down to t=0

Bidirectional RNN: Processing an input string



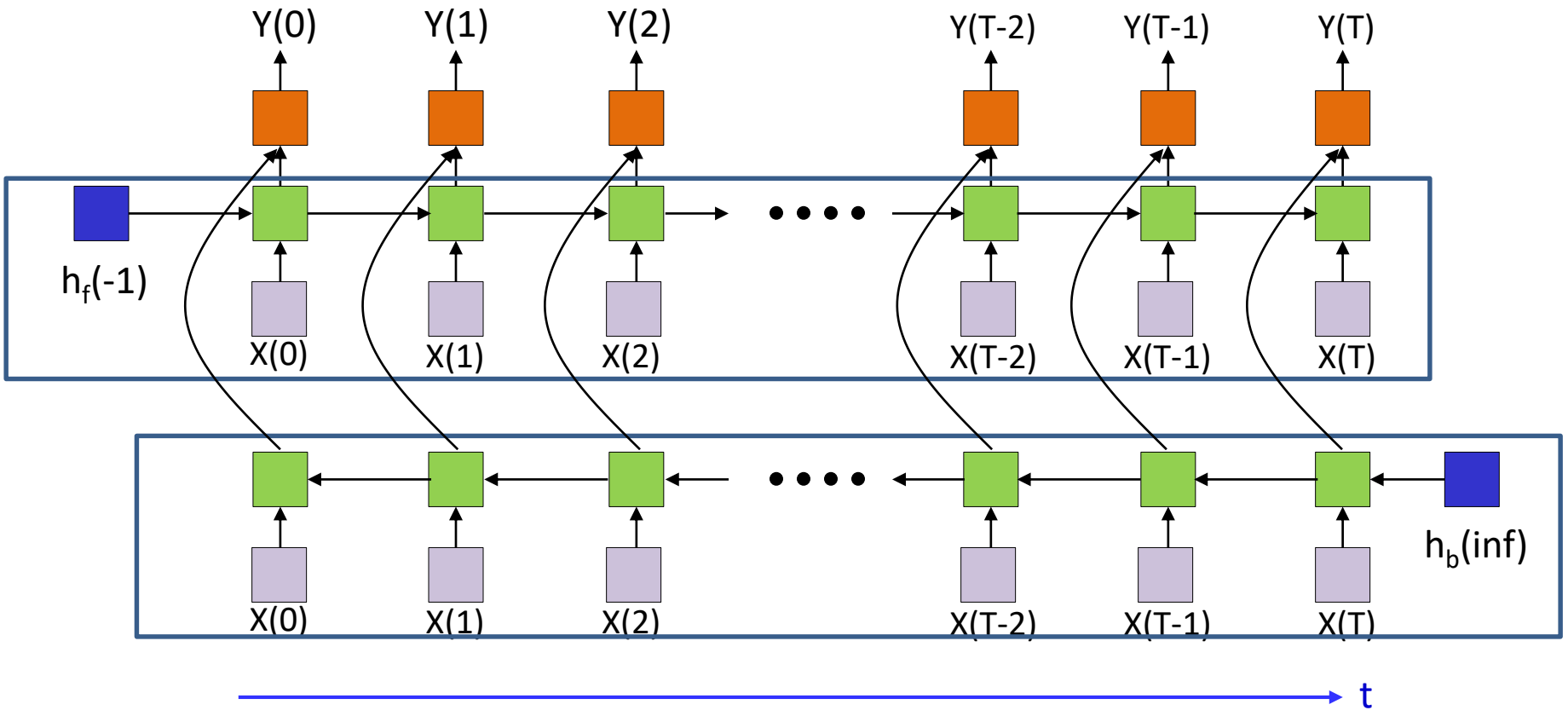
- The forward net process the data from t=0 to t=T
  - Only computing the hidden states, initially

Bidirectional RNN: Processing an input string



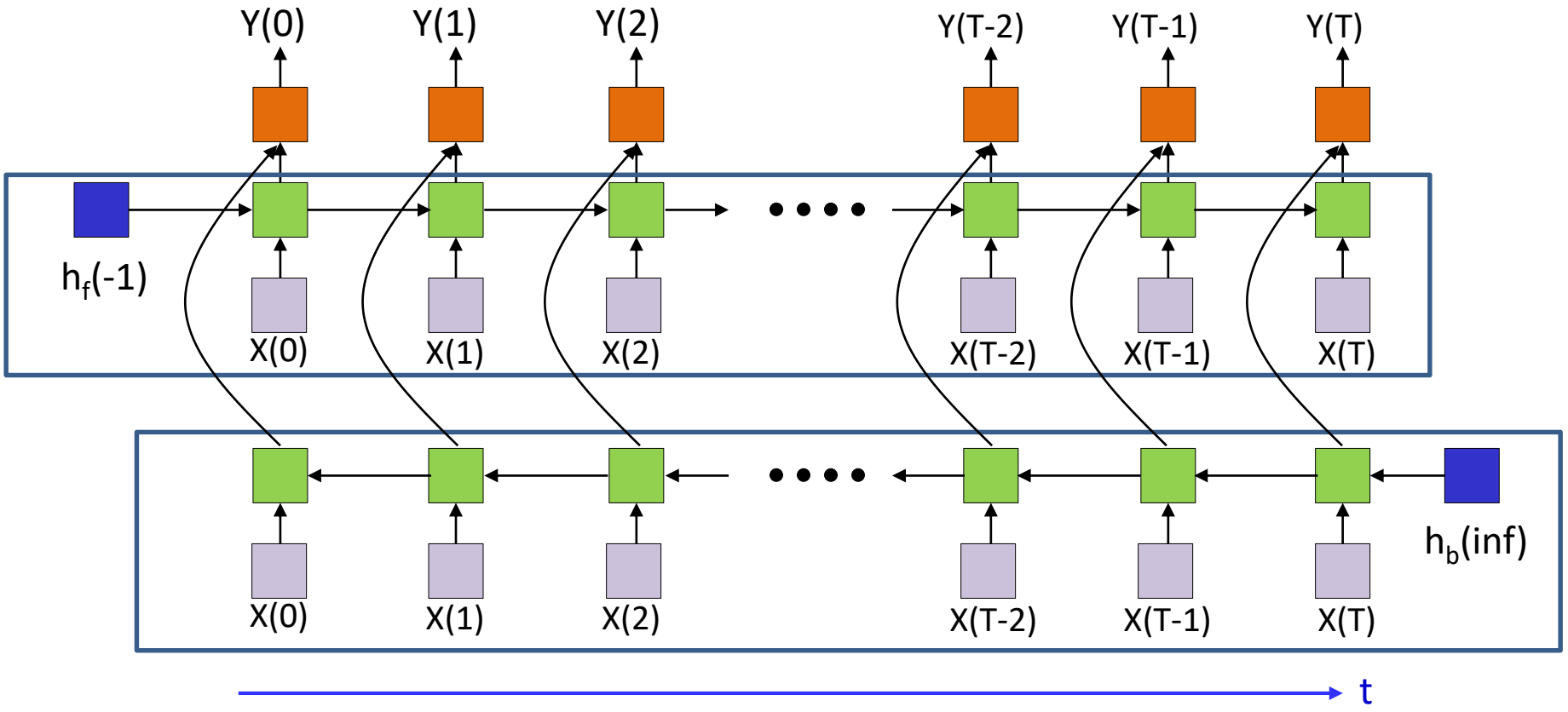
- The backward nets processes the input data in *reverse time,* end to beginning
  - Initially only the hidden state values are computed
  - Clearly, this is not an online process and requires the *entire* input data
  - Note: *This is not the backward pass of backprop.*

Bidirectional RNN: Processing an input string



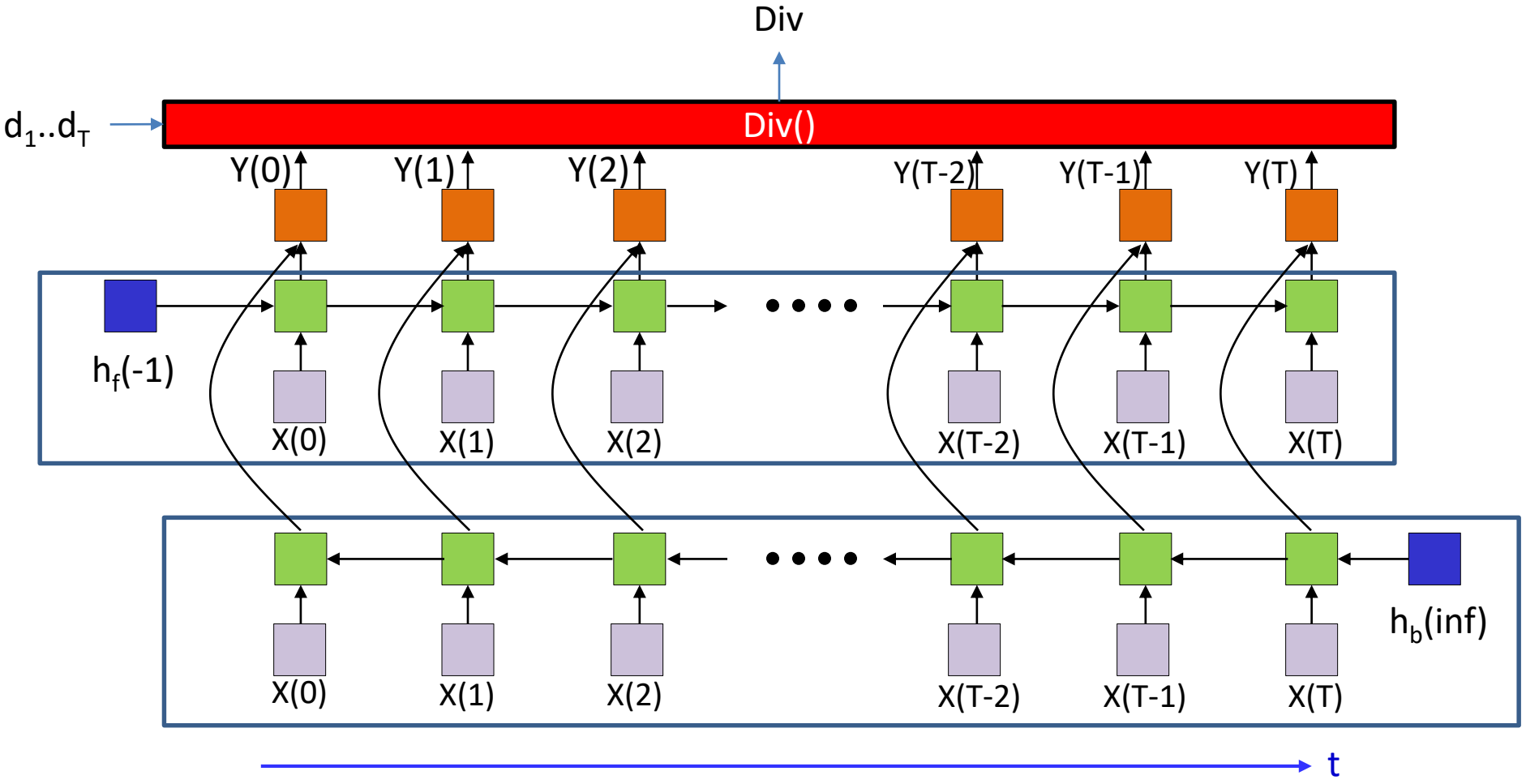
- The computed states of both networks are used to compute the final output at each time

Backpropagation in BRNNs



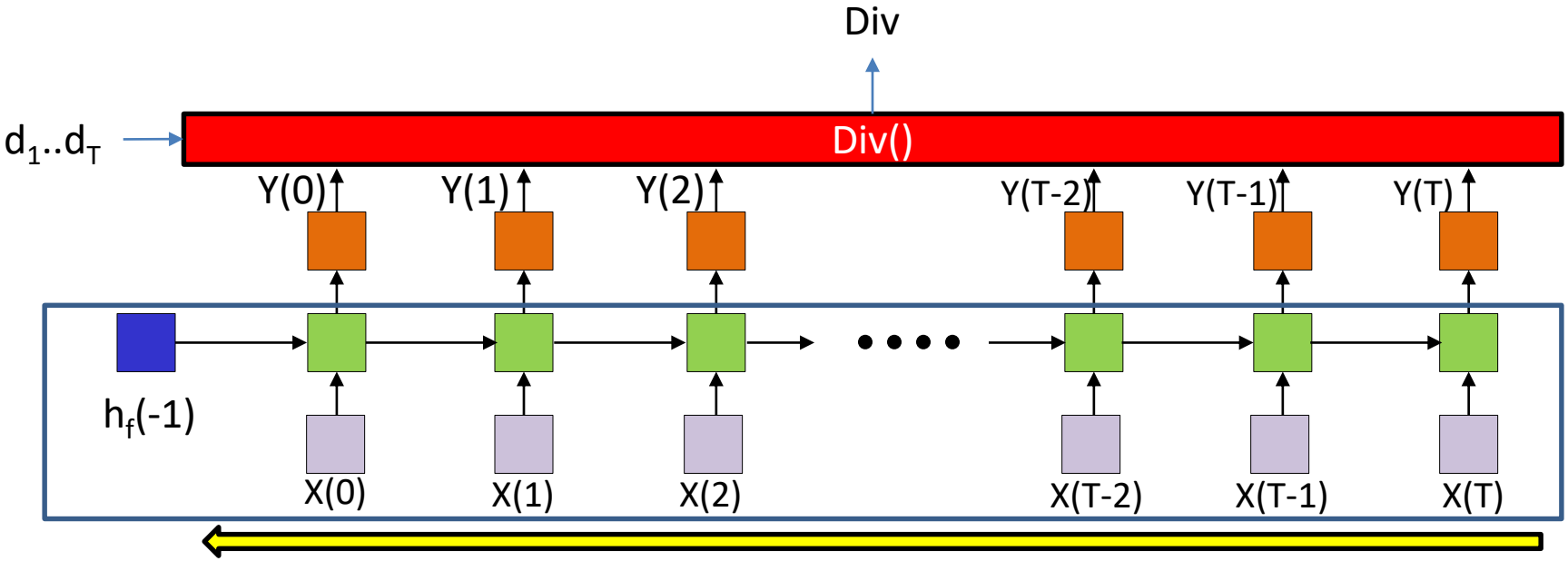
- Forward pass: Compute both forward and backward networks and final output

Backpropagation in BRNNs



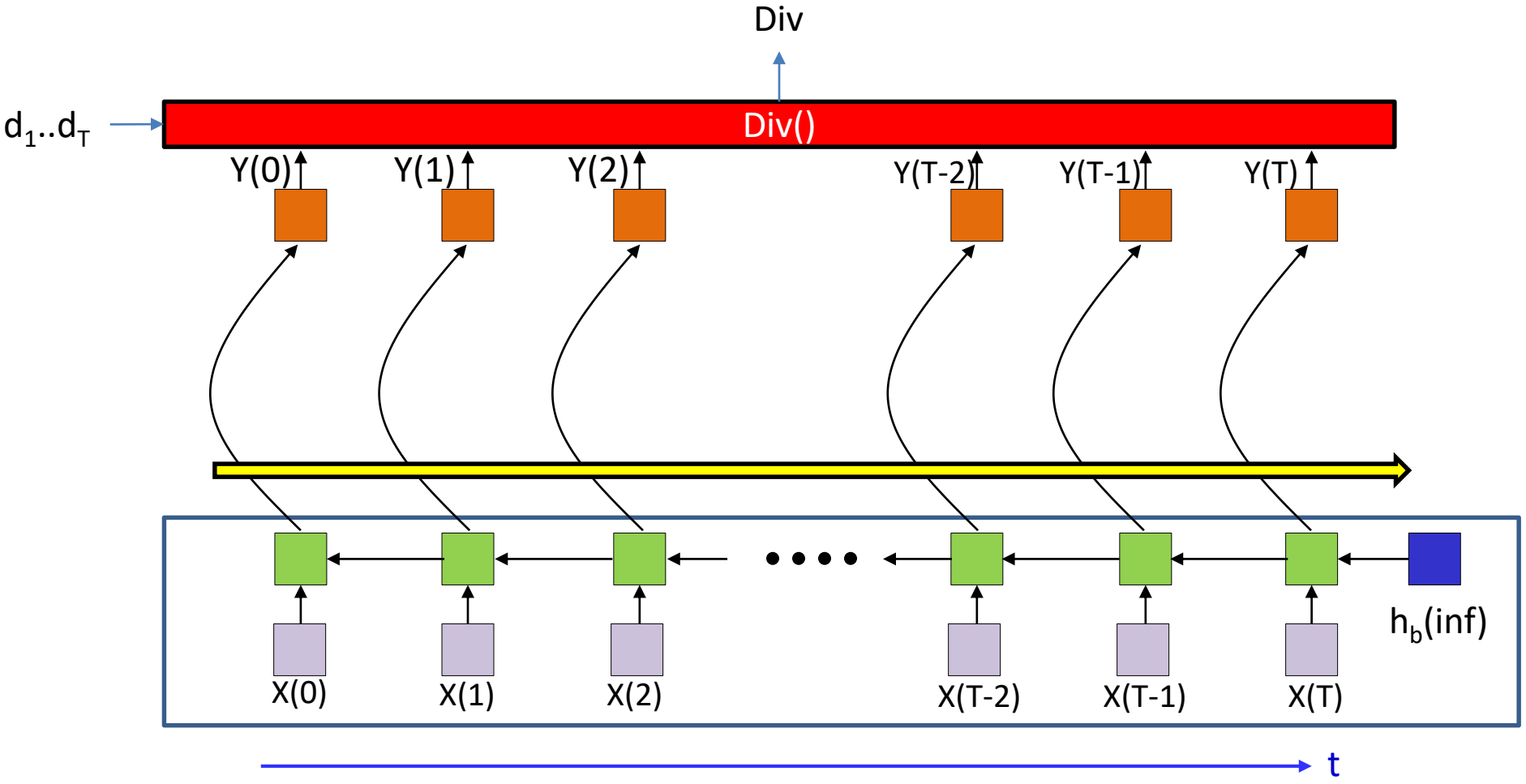
- Backward pass: Define a divergence from the desired output

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**

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**

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

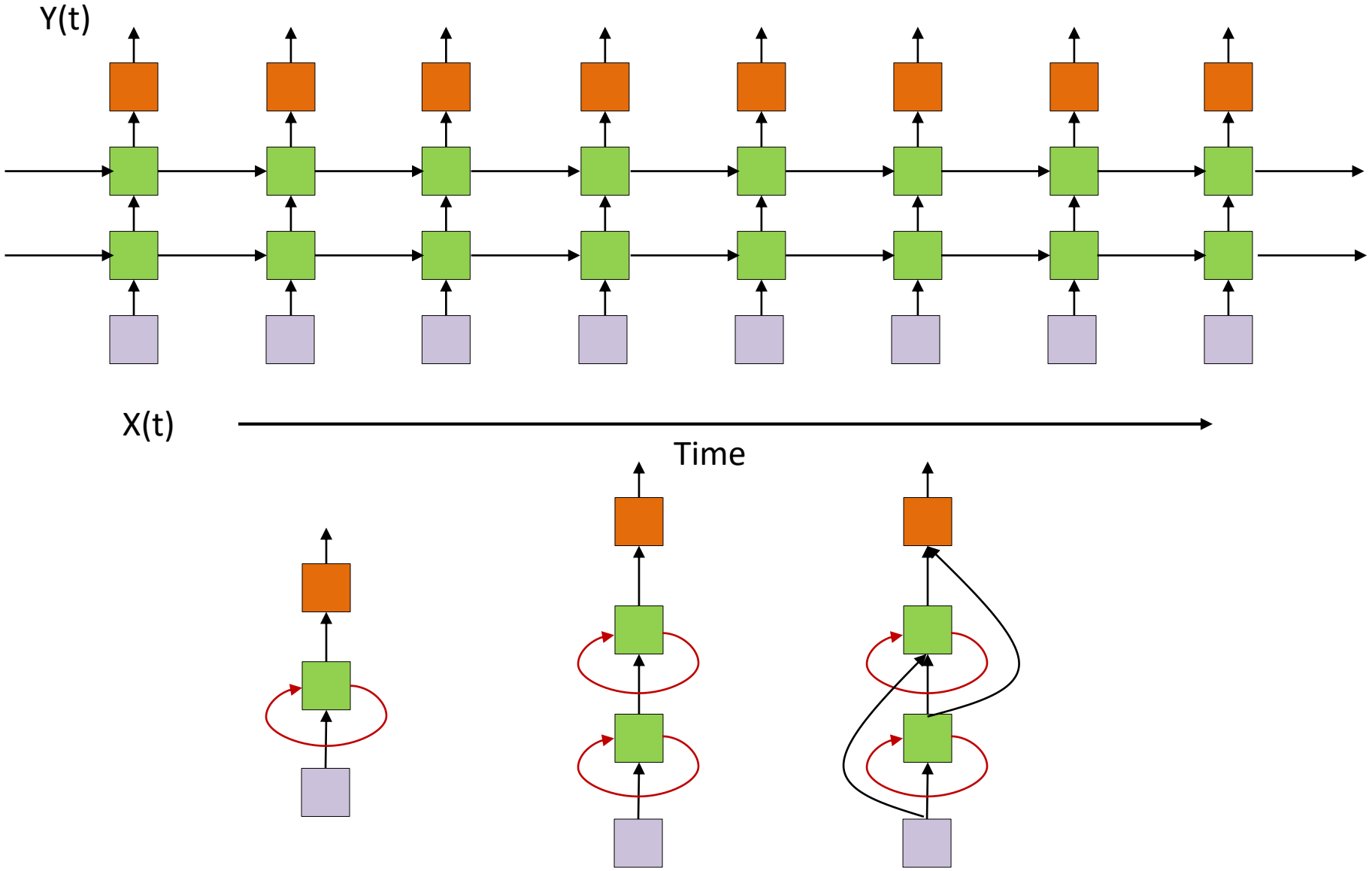
```
Naturalism and decision for the majority of Arab countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could 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..

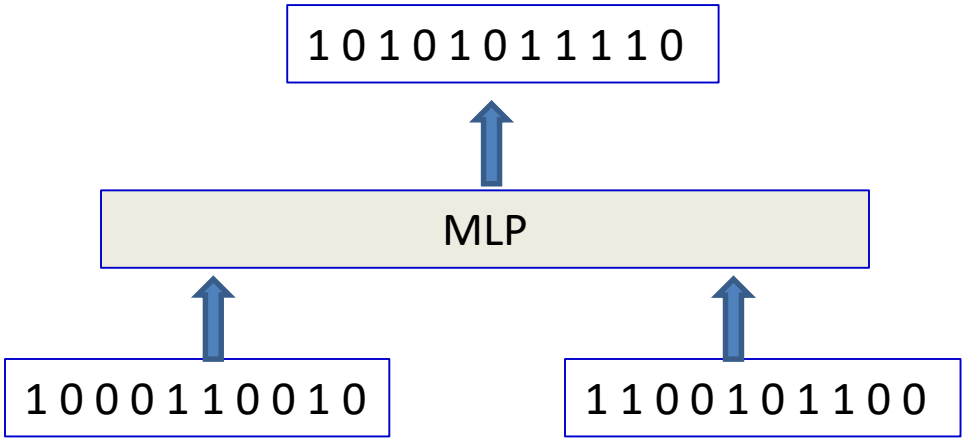
RNNs..

- 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

Recall: A Recurrent Neural Network

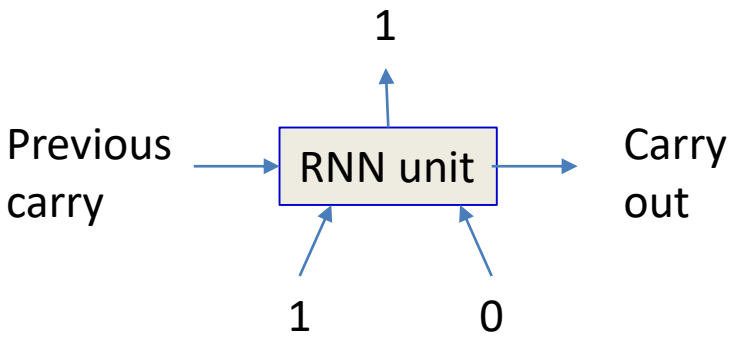


MLPs vs RNNs



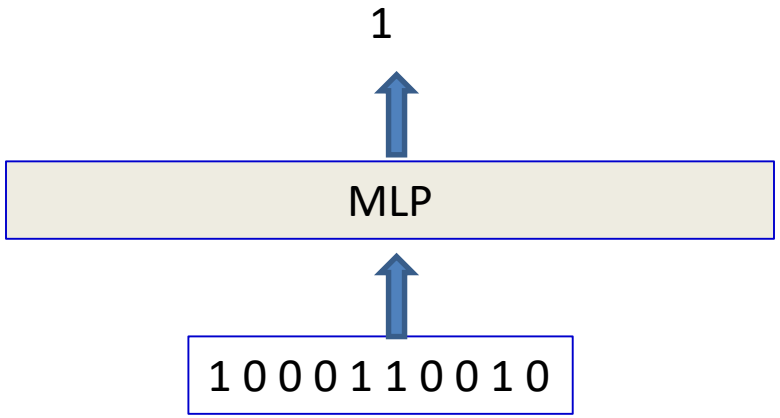
- The addition problem: Add two N-bit numbers to produce a N+1-bit number
  - Input is binary
  - Will require large number of training instances
    - Output must be specified for every pair of inputs
    - Weights that generalize will make errors
  - Network trained for N-bit numbers will not work for N+1 bit numbers

MLPs vs RNNs



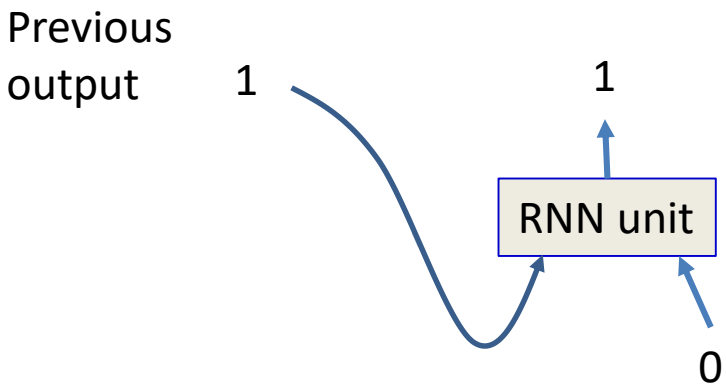
- The addition problem: Add two N-bit numbers to produce a N+1-bit number
- **RNN solution:** Very simple, can add two numbers of any size

MLP: The parity problem



- Is the number of “ones” even or odd
- Network must be complex to capture all patterns
  - At least one hidden layer of size N plus an output neuron
  - Fixed input size

RNN: The parity problem



- Trivial solution
- Generalizes to input of any size

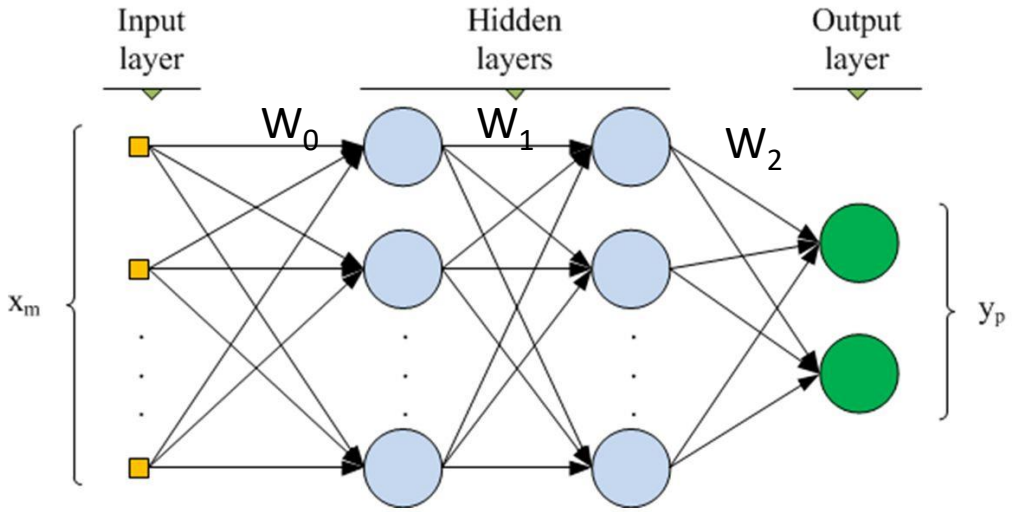
RNNs..

- Excellent models for time-series analysis tasks
  - Time-series prediction
  - Time-series classification
  - Sequence prediction..
  - They can even simplify problems that are difficult for MLPs
- But first – a problem..

The vanishing gradient problem

- A particular problem with training deep networks..
  - The gradient of the error with respect to weights is unstable..

Some useful preliminary math: The problem with training deep networks



- A multilayer perceptron is a nested function

$$Y = f_N\left(W_{N-1}f_{N-1}\left(W_{N-2}f_{N-2}(\ldots W_0X)\right)\right)$$

- W_k is the weights *matrix* at the k[th] layer
- The *error* for X can be written as

$$Div(X) = D\left(f_N\left(W_{N-1}f_{N-1}\left(W_{N-2}f_{N-2}(\ldots W_0X)\right)\right)\right)$$

Training deep networks

- Vector derivative chain rule: for any $f\big(Wg(X)\big)$:

$$\frac{df\big(Wg(X)\big)}{dX} = \frac{df\big(Wg(X)\big)}{dWg(X)}\frac{dWg(X)}{dg(X)}\frac{dg(X)}{dX}$$

Poor notation

$$\nabla_X f = \nabla_Z f . W . \nabla_X g$$

- Where
  - $Z = Wg(X)$
  - $\nabla_Z f$ is the *jacobian* ***matrix*** of $f(Z)$w.r.t Z
    - Using the notation $\nabla_Z f$ instead of $J_f(z)$ for consistency

Training deep networks

- For

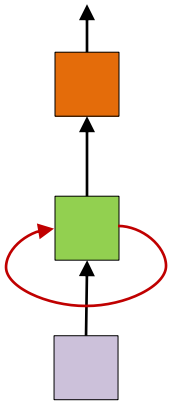
$$Div(X) = D\left(f_N\left(W_{N-1}f_{N-1}\left(W_{N-2}f_{N-2}(\ldots W_0 X)\right)\right)\right)$$

- We get:

$$\nabla_{f_k} Div = \nabla D.\nabla f_N.W_{N-1}.\nabla f_{N-1}.W_{N-2}\ldots\nabla f_{k+1}W_k$$

- Where
  - $\nabla_{f_k} Div$ is the gradient $Div(X)$ of the error w.r.t the output of the kth layer of the network
    - Needed to compute the gradient of the error w.r.t W_{k-1}
  - ∇f_n is *jacobian* of $f_N()$ w.r.t. to its current input
  - All blue terms are matrices

The Jacobian of the hidden layers



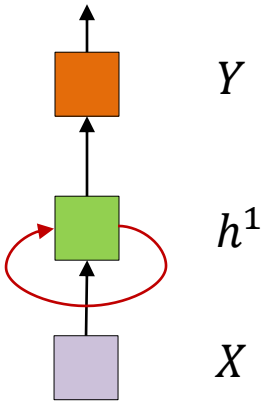
$$\nabla f_t(z_i) = \begin{bmatrix} f'_{t,1}(z_1) & 0 & \cdots & 0 \\ 0 & f'_{t,2}(z_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'_{t,N}(z_N) \end{bmatrix}$$

$$h_i^{(1)}(t) = f_1\left(z_i^{(1)}(t)\right)$$

- $\nabla f_t()$ is the derivative of the output of the (layer of) hidden recurrent neurons with respect to their input
  - A matrix where the diagonal entries are the derivatives of the *activation* of the recurrent hidden layer

The Jacobian

$$h_i^{(1)}(t) = f_1\left(z_i^{(1)}(t)\right)$$

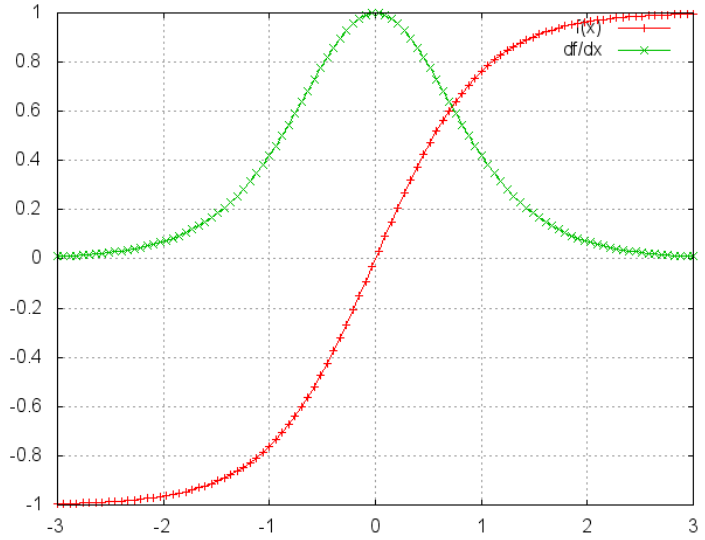


$$\nabla f_t(z_i) = \begin{bmatrix} f'_{t,1}(z_1) & 0 & \cdots & 0 \\ 0 & f'_{t,2}(z_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'_{t,N}(z_N) \end{bmatrix}$$

- The derivative (or subgradient) of the activation function is always bounded
  - The diagonals of the Jacobian are bounded
- There is a limit on how much multiplying a vector by the Jacobian will scale it

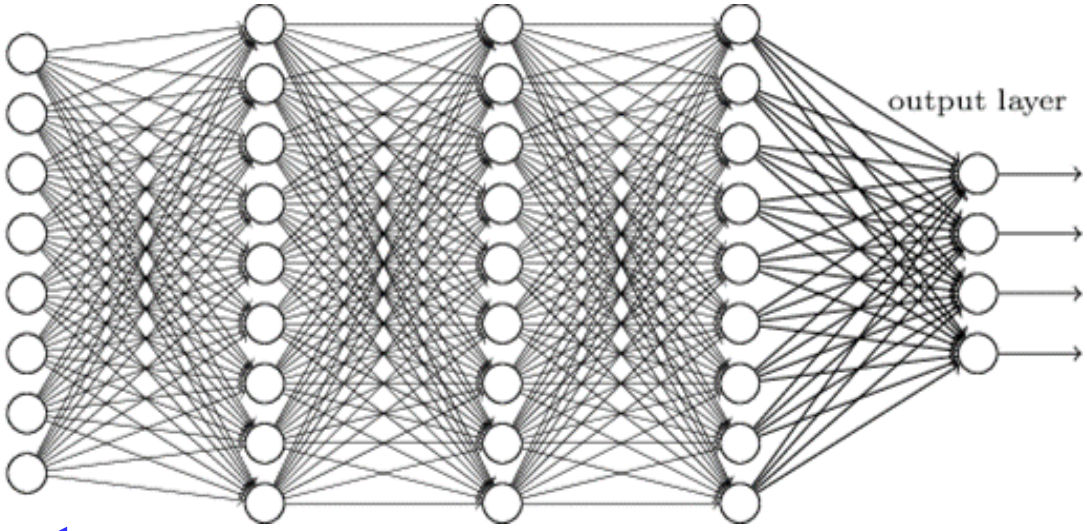
The derivative of the hidden state activation

$$\nabla f_t(z_i) = \begin{bmatrix} f'_{t,1}(z_1) & 0 & \cdots & 0 \\ 0 & f'_{t,2}(z_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'_{t,N}(z_N) \end{bmatrix}$$



- Most common activation functions, such as sigmoid, tanh() and RELU have derivatives that are always less than 1
- The most common activation for the hidden units in an RNN is the tanh()
  - The derivative of $\tanh()$is always less than 1
- Multiplication by the Jacobian is always a *shrinking* operation

Training deep networks



- As we go back in layers, the Jacobians of the activations constantly *shrink* the derivative
  - After a few instants the derivative of the divergence at any time is totally "forgotten"

What about the weights

$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_{N-1} . \nabla f_{N-1} . W_{N-2} \ldots \nabla f_{k+1} W_k$$

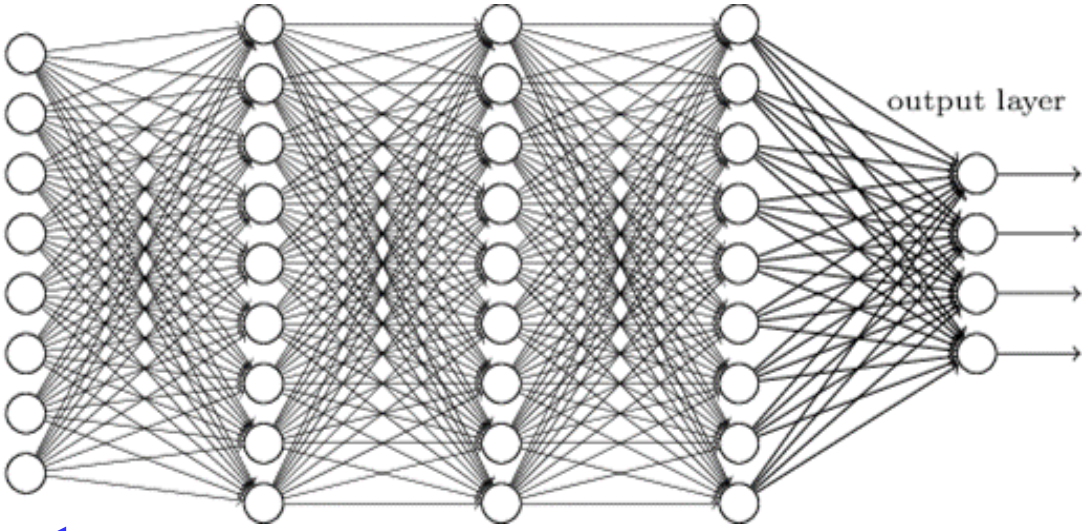
- In a single-layer RNN, the weight matrices are identical
- The chain product for $\nabla_{f_k} Div$ will
  - E*xpand* ∇D along directions in which the singular values of the weight matrices are greater than 1
  - S*hrink* ∇D in directions where the singular values ae less than 1
  - **Exploding** or **vanishing** gradients

Exploding/Vanishing gradients

$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_{N-1} . \nabla f_{N-1} . W_{N-2} \ldots \nabla f_{k+1} W_k$$

- Every blue term is a matrix
- ∇D is proportional to the actual error
  - Particularly for L_2 and KL divergence
- The chain product for $\nabla_{f_k} Div$ will
  - E*xpand* ∇D in directions where each stage has singular values greater than 1
  - S*hrink* ∇D in directions where each stage has singular values less than 1

Gradient problems in deep networks

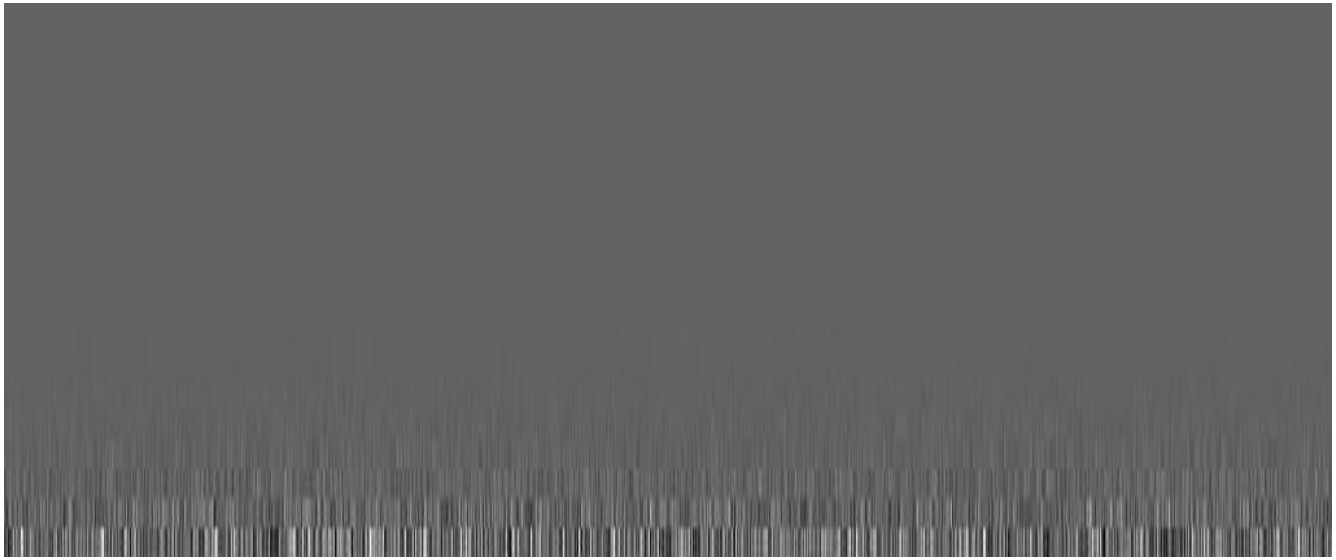


- The gradients in the lower/earlier layers can *explode* or *vanish*
  - Resulting in insignificant or unstable gradient descent updates
  - Problem gets worse as network depth increases

Vanishing gradient examples..

ELU activation, Batch gradients

Input layer



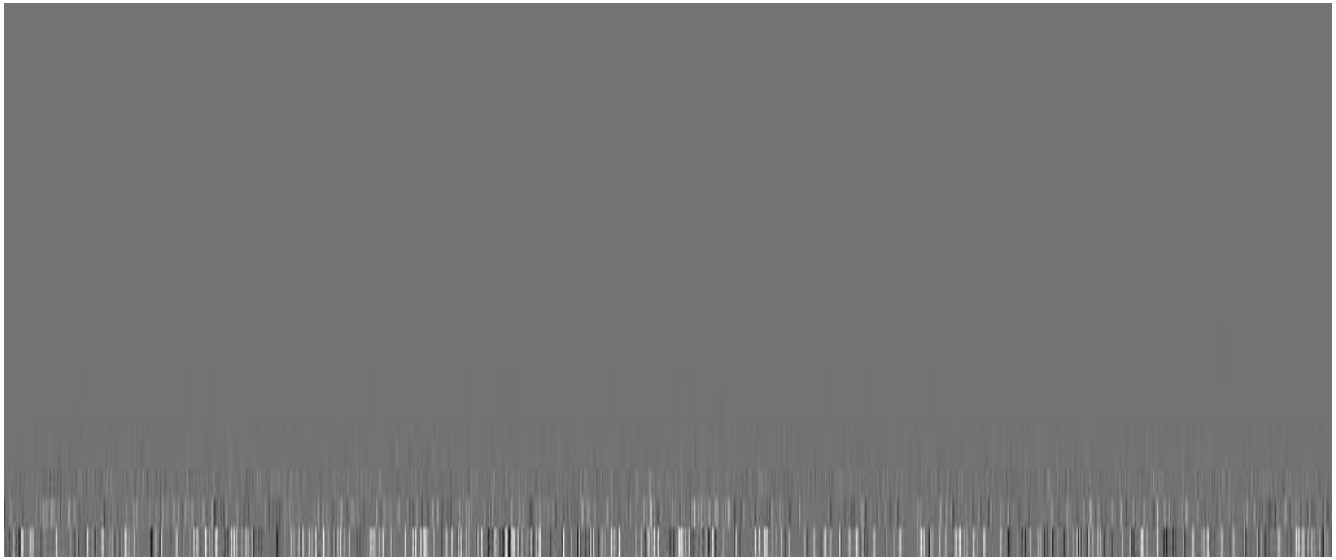
Output layer

- 19 layer MNIST model
  - Different activations: Exponential linear units, RELU, sigmoid, than
  - Each layer is 1024 layers wide
  - Gradients shown at initialization
    - Will actually *decrease* with additional training
- Figure shows $\log\left|\nabla_{W_{neuron}} E\right|$ where W_{neuron} is the vector of incoming weights to each neuron
  - I.e. the gradient of the loss w.r.t. the entire set of weights to each neuron

Vanishing gradient examples..

RELU activation, Batch gradients

Input layer



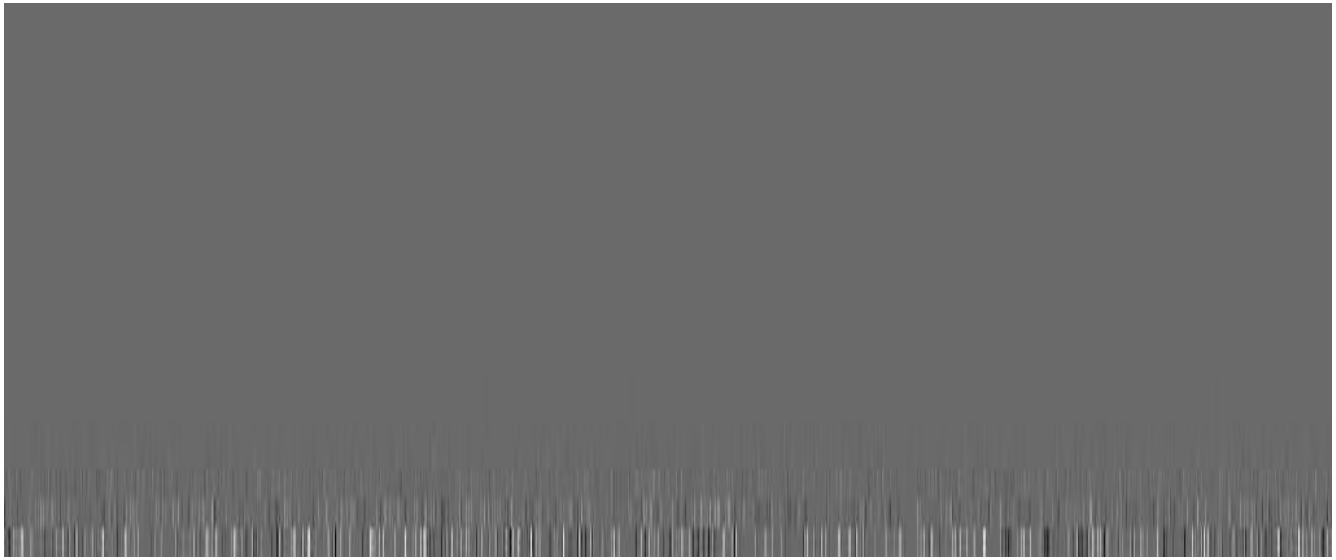
Output layer

- 19 layer MNIST model
  - Different activations: Exponential linear units, RELU, sigmoid, than
  - Each layer is 1024 layers wide
  - Gradients shown at initialization
    - Will actually *decrease* with additional training
- Figure shows $\log\left|\nabla_{W_{neuron}} E\right|$ where W_{neuron} is the vector of incoming weights to each neuron
  - I.e. the gradient of the loss w.r.t. the entire set of weights to each neuron

Vanishing gradient examples..

Sigmoid activation, Batch gradients

Input layer



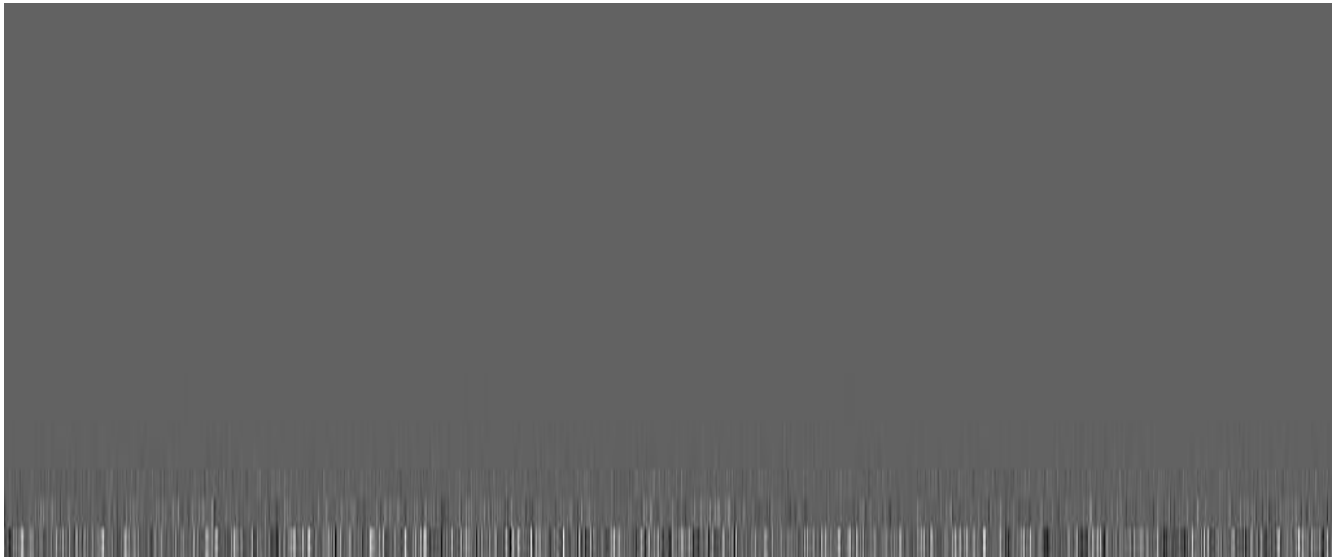
Output layer

- 19 layer MNIST model
  - Different activations: Exponential linear units, RELU, sigmoid, than
  - Each layer is 1024 layers wide
  - Gradients shown at initialization
    - Will actually *decrease* with additional training
- Figure shows $\log\left|\nabla_{W_{neuron}} E\right|$ where W_{neuron} is the vector of incoming weights to each neuron
  - I.e. the gradient of the loss w.r.t. the entire set of weights to each neuron

Vanishing gradient examples..

Tanh activation, Batch gradients

Input layer

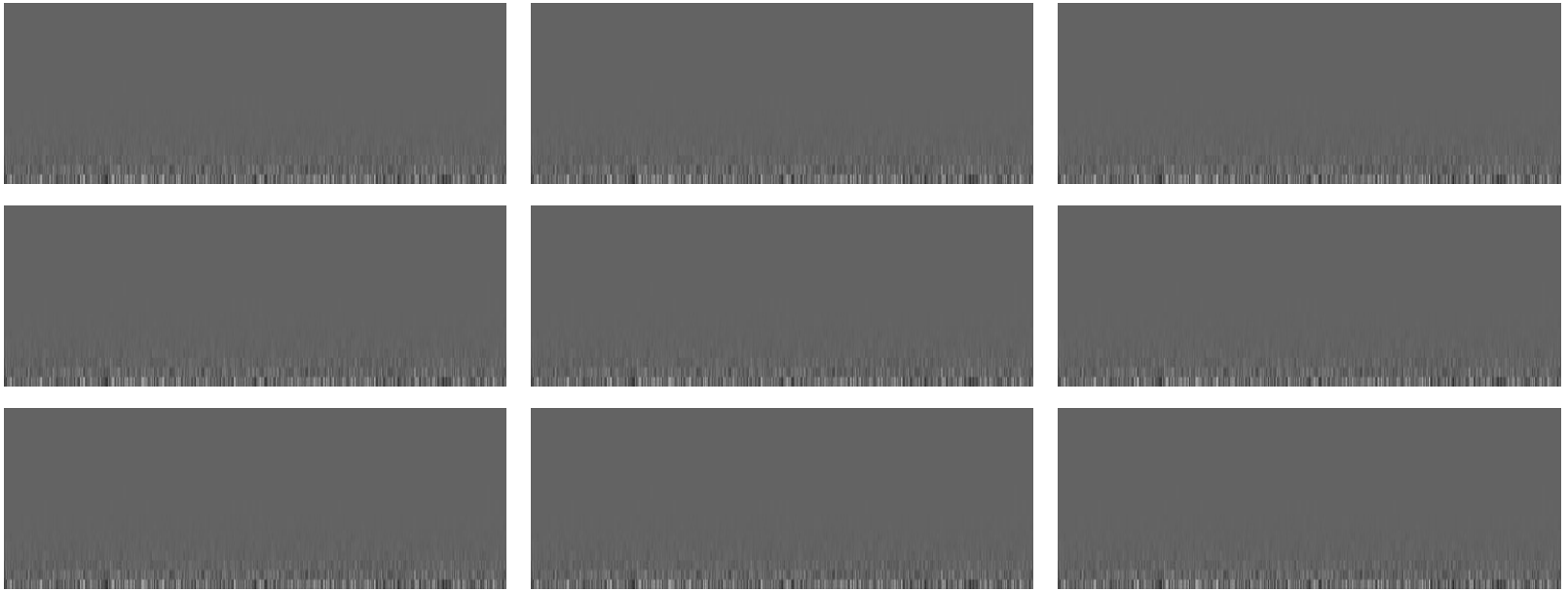


Output layer

- 19 layer MNIST model
  - Different activations: Exponential linear units, RELU, sigmoid, than
  - Each layer is 1024 layers wide
  - Gradients shown at initialization
    - Will actually *decrease* with additional training
- Figure shows $\log\left|\nabla_{W_{neuron}} E\right|$ where W_{neuron} is the vector of incoming weights to each neuron
  - I.e. the gradient of the loss w.r.t. the entire set of weights to each neuron

Vanishing gradient examples..

ELU activation, Individual instances



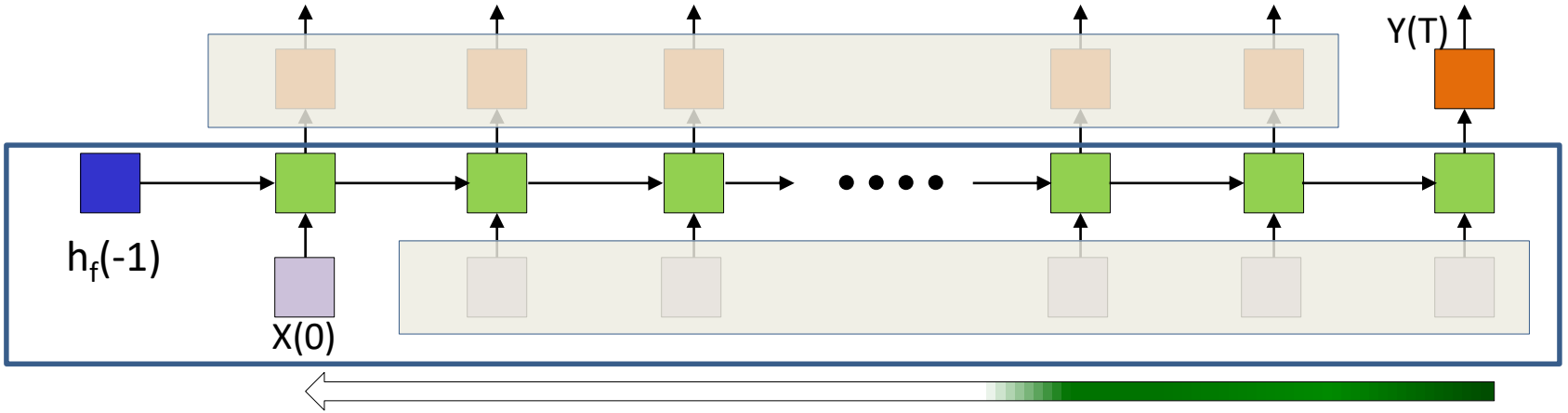
- 19 layer MNIST model
  - Different activations: Exponential linear units, RELU, sigmoid, than
  - Each layer is 1024 layers wide
  - Gradients shown at initialization
    - Will actually *decrease* with additional training
- Figure shows $\log\left|\nabla_{W_{neuron}} E\right|$ where W_{neuron} is the vector of incoming weights to each neuron
  - I.e. the gradient of the loss w.r.t. the entire set of weights to each neuron

Vanishing gradients

- ELU activations maintain gradients longest
- But in all cases gradients effectively vanish after about 10 layers!
  - Your results may vary

- Both batch gradients and gradients for individual instances disappear
  - In reality a tiny number may actually blow up.

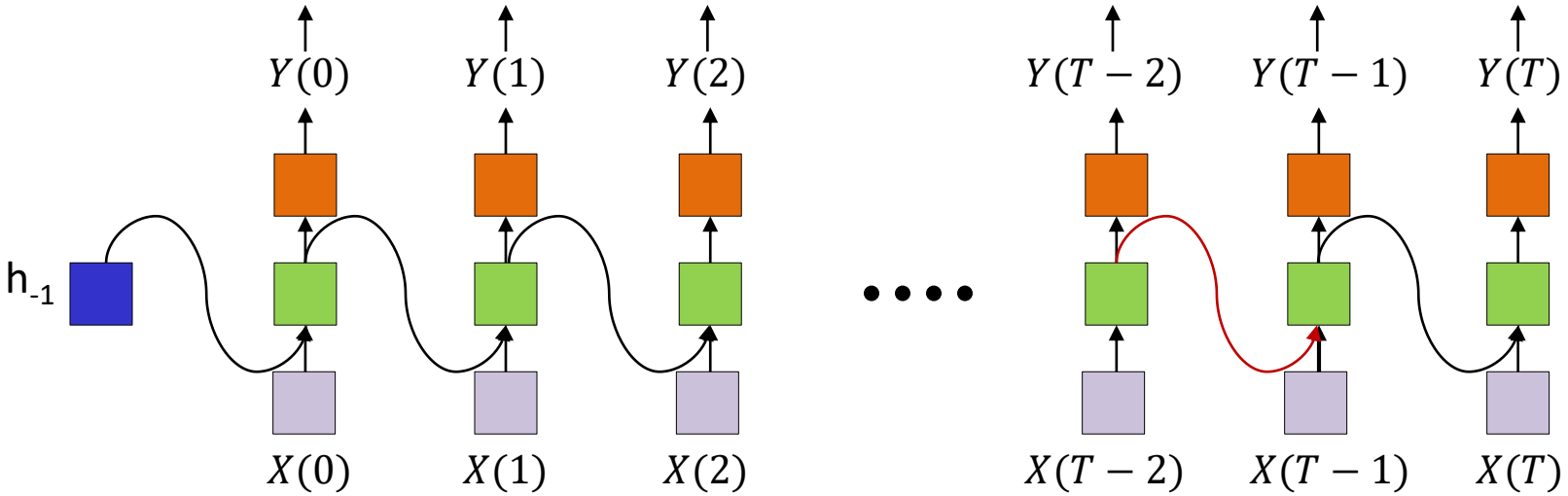
Recurrent nets are very deep nets



$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_{N-1} . \nabla f_{N-1} . W_{N-2} \ldots \nabla f_{k+1} W_k$$

- The relation between $X(0)$ and $Y(T)$ is one of a very deep network
  - Gradients from errors at t $= T$ will vanish by the time they're propagated to $t = 0$

Vanishing stuff..



- Not merely during back propagation
- Stuff gets forgotten in the forward pass too
  - Otherwise outputs would saturate or blow up
  - Typically forgotten in a dozen or so timesteps

The long-term dependency problem

PATTERN1 [……..] PATTERN 2

Jane had a quick lunch in the bistro. Then *she..*

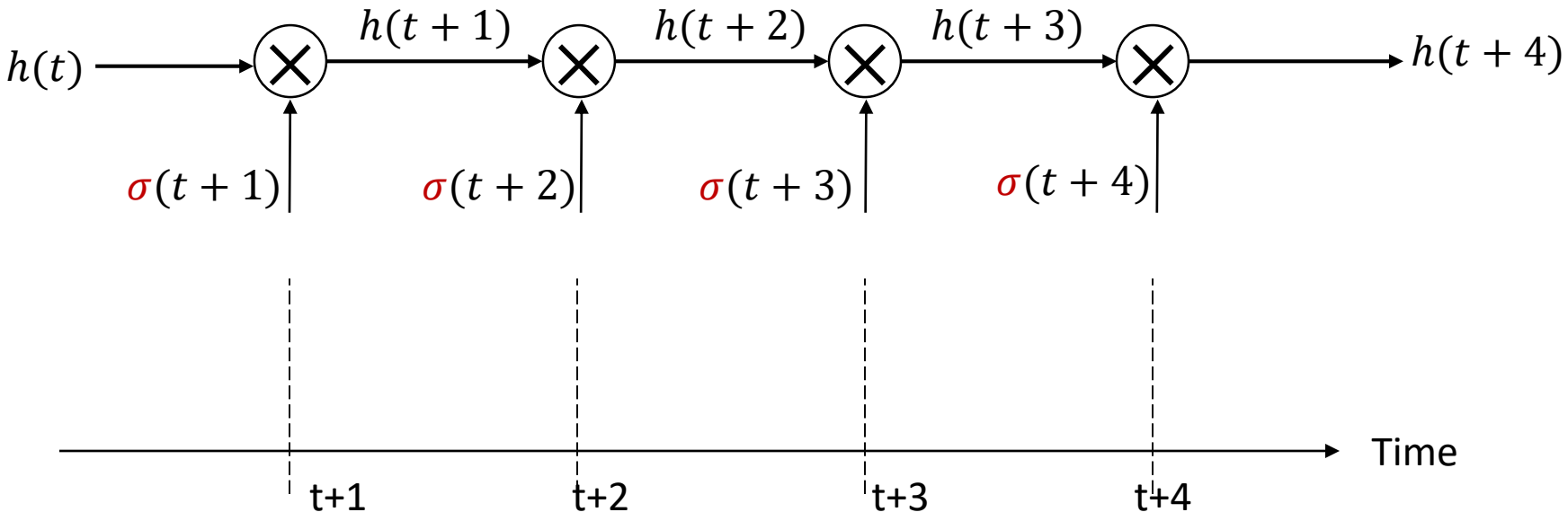
- Any other pattern of any length can happen between pattern 1 and pattern 2
  - RNN will "forget" pattern 1 if intermediate stuff is too long
  - "Jane" → the next pronoun referring to her will be "she"
- *Can* learn such dependencies in theory; in practice *will not*
  - Vanishing gradient problem

Exploding/Vanishing gradients

$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_{N-1} . \nabla f_{N-1} . W_{N-2} \ldots \nabla f_{k+1} W_k$$

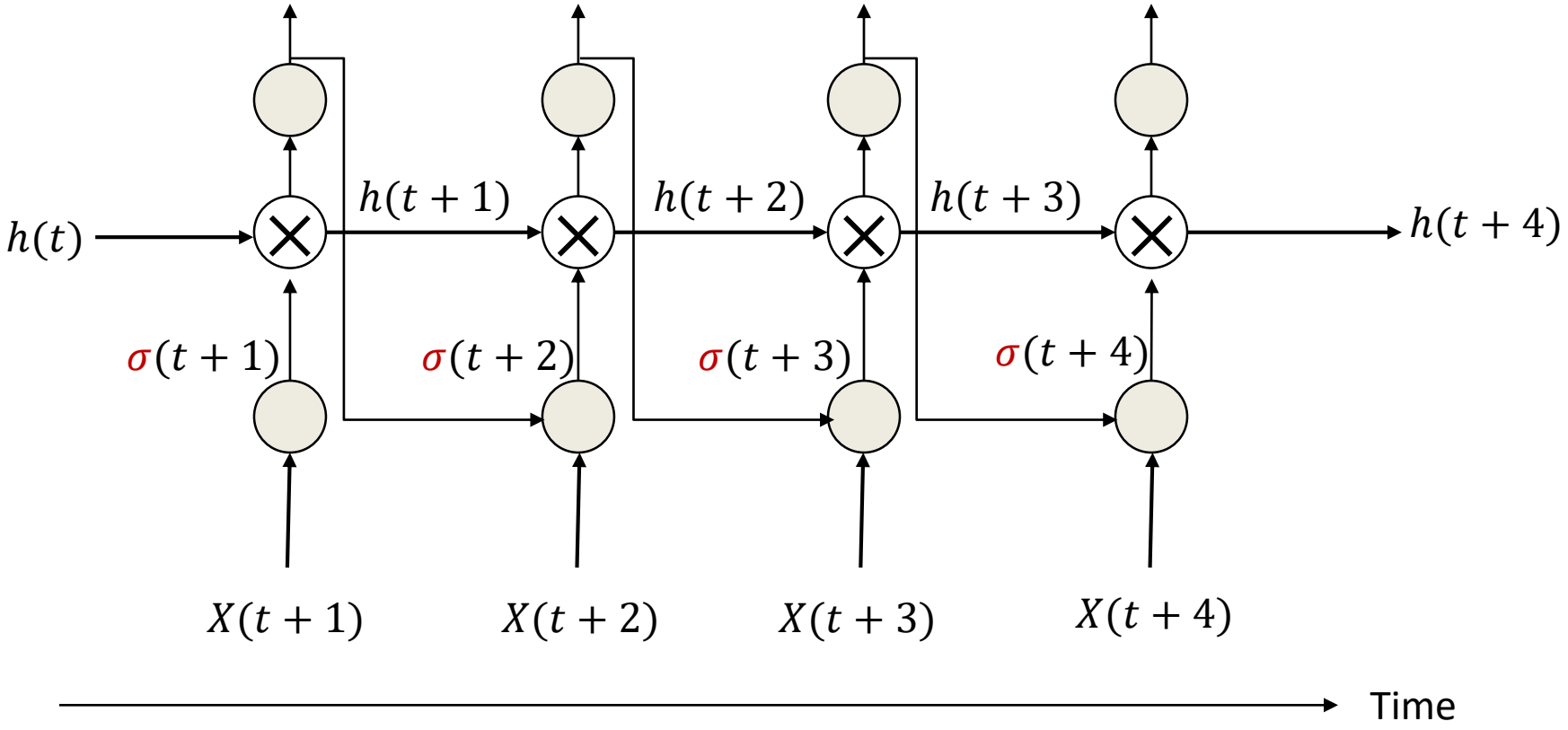
- Can we replace this with something that doesn't fade or blow up?
- $\nabla_{f_k} Div = \nabla D \, C\sigma_N \, C\sigma_{N-1} C \ldots \sigma_k$

Enter – the constant error carousel



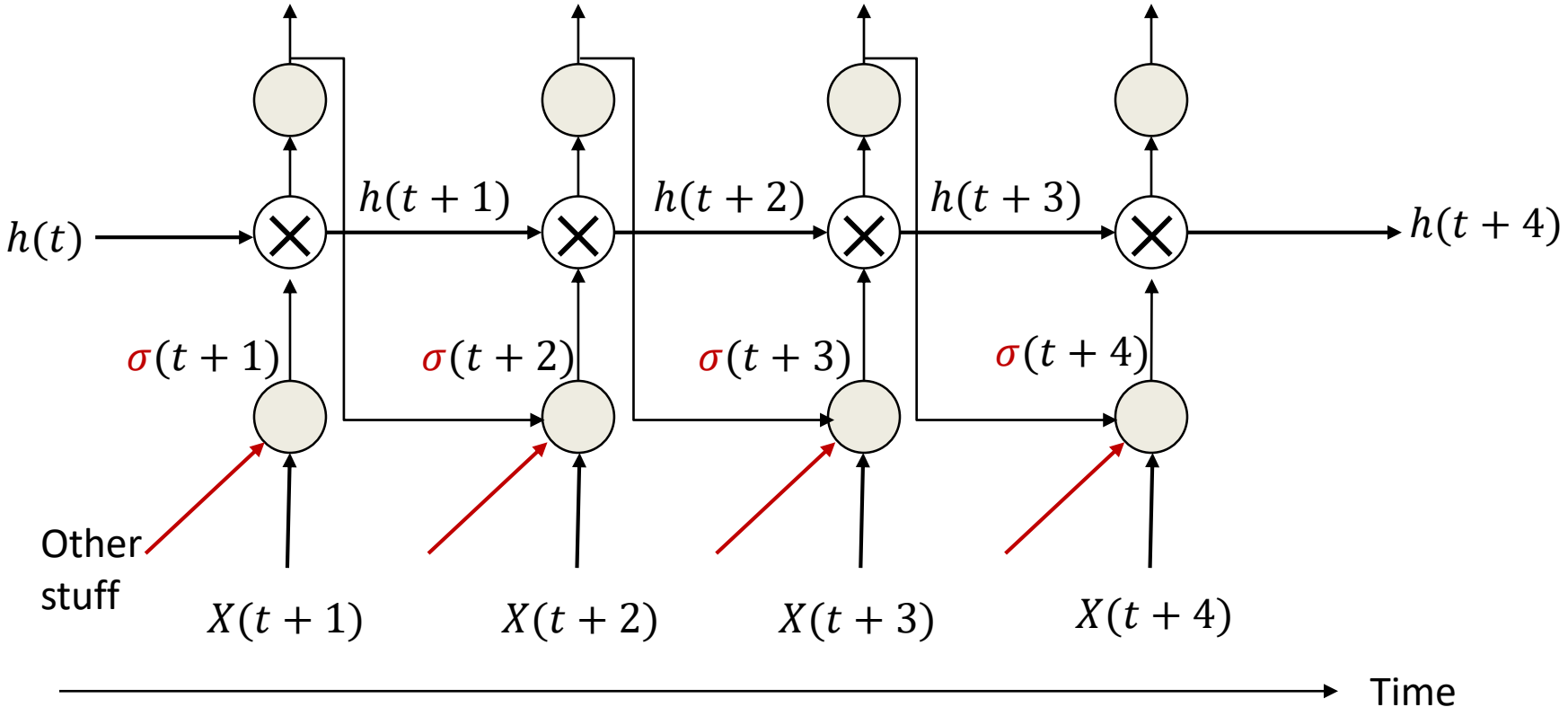
- History is carried through uncompressed
  - No weights, no nonlinearities
  - Only scaling is through the σ "gating" term that captures other triggers
  - E.g. "Have I seen Pattern2"?

Enter – the constant error carousel



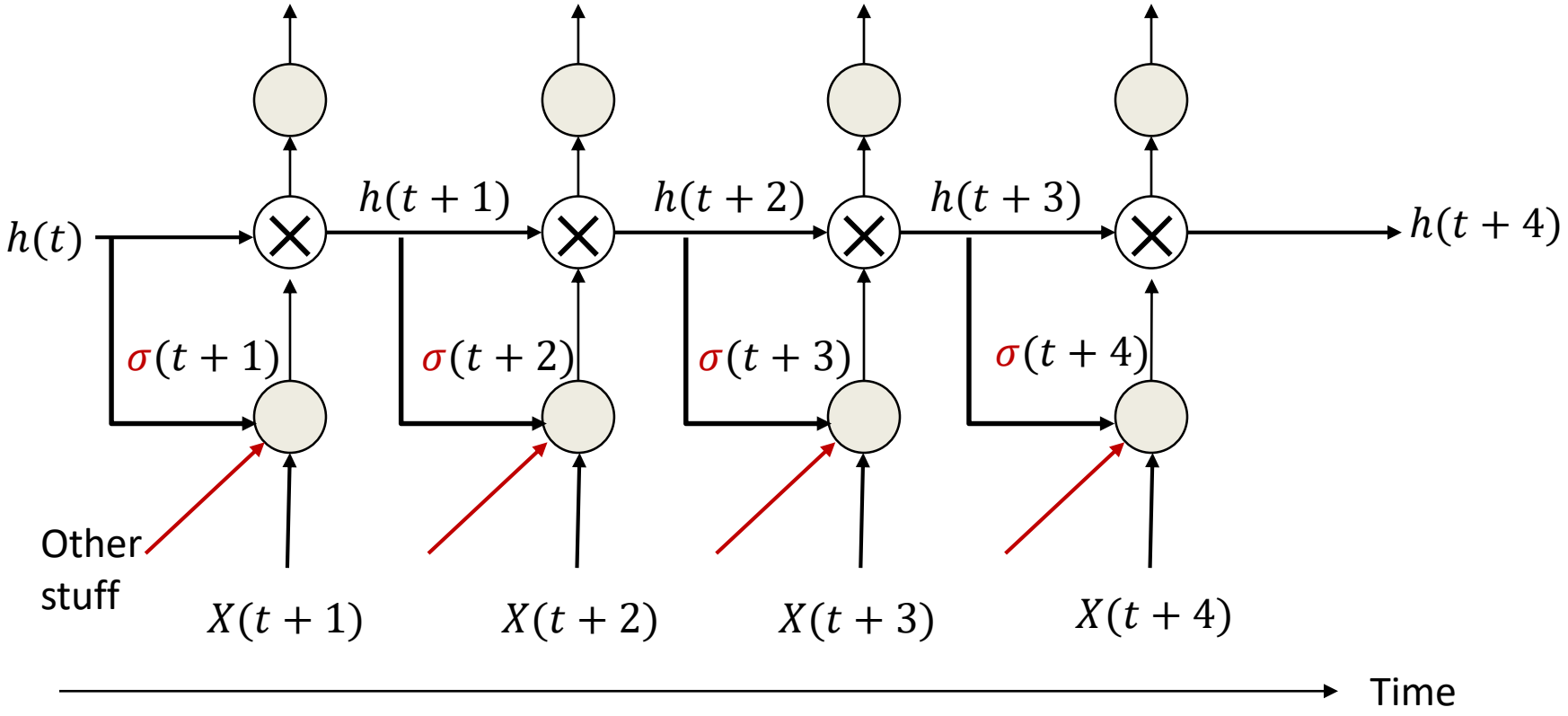
- Actual non-linear work is done by other portions of the network

Enter – the constant error carousel



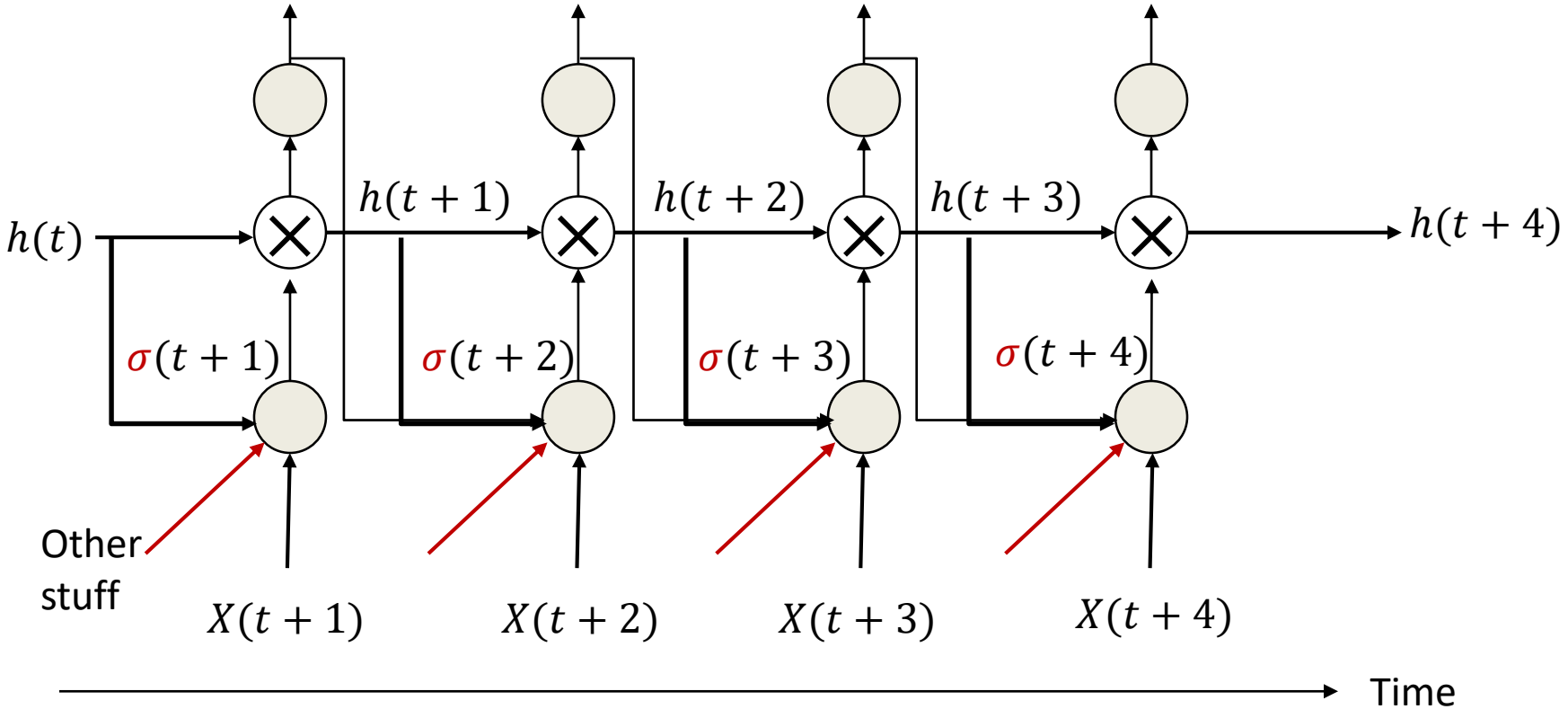
- Actual non-linear work is done by other portions of the network

Enter – the constant error carousel



- Actual non-linear work is done by other portions of the network

Enter – the constant error carousel



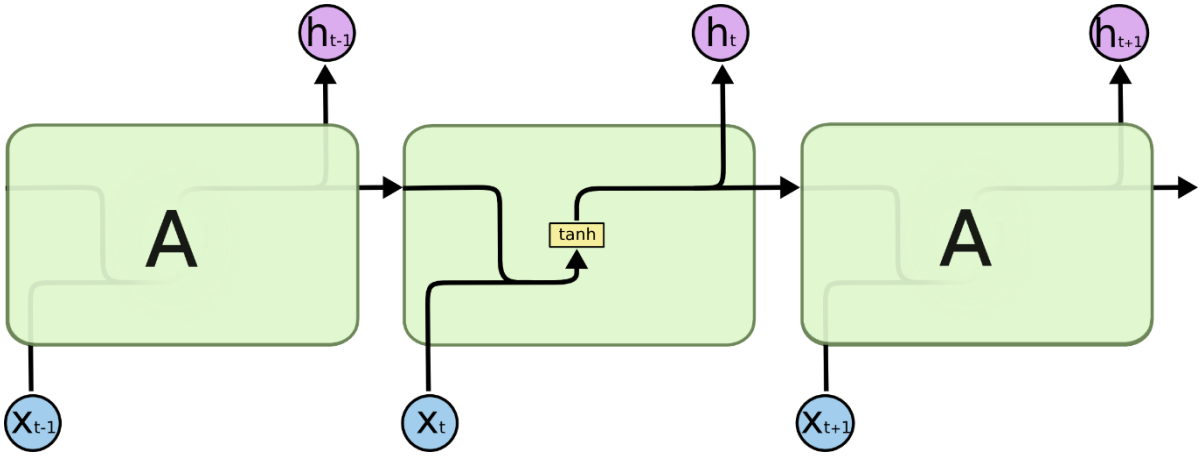
- Actual non-linear work is done by other portions of the network

Enter the *LSTM*

- *Long Short-Term Memory*
- Explicitly latch information to prevent decay / blowup

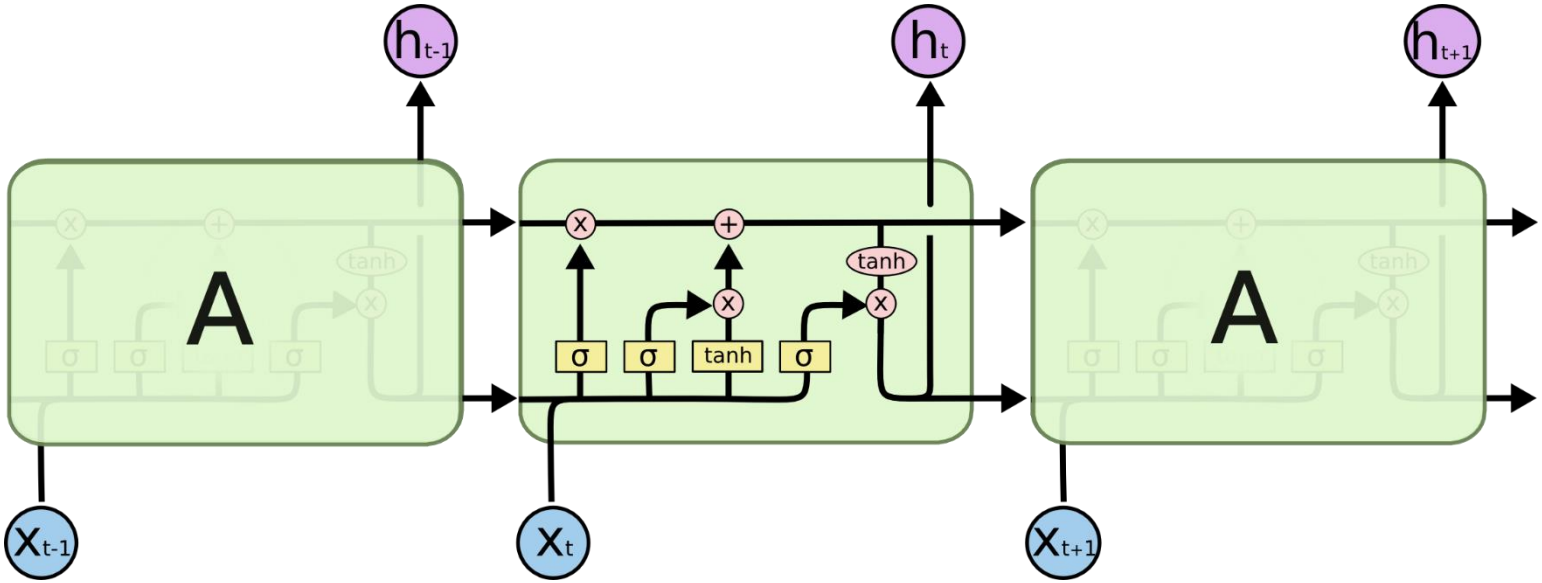
- Following notes borrow liberally from
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Standard RNN



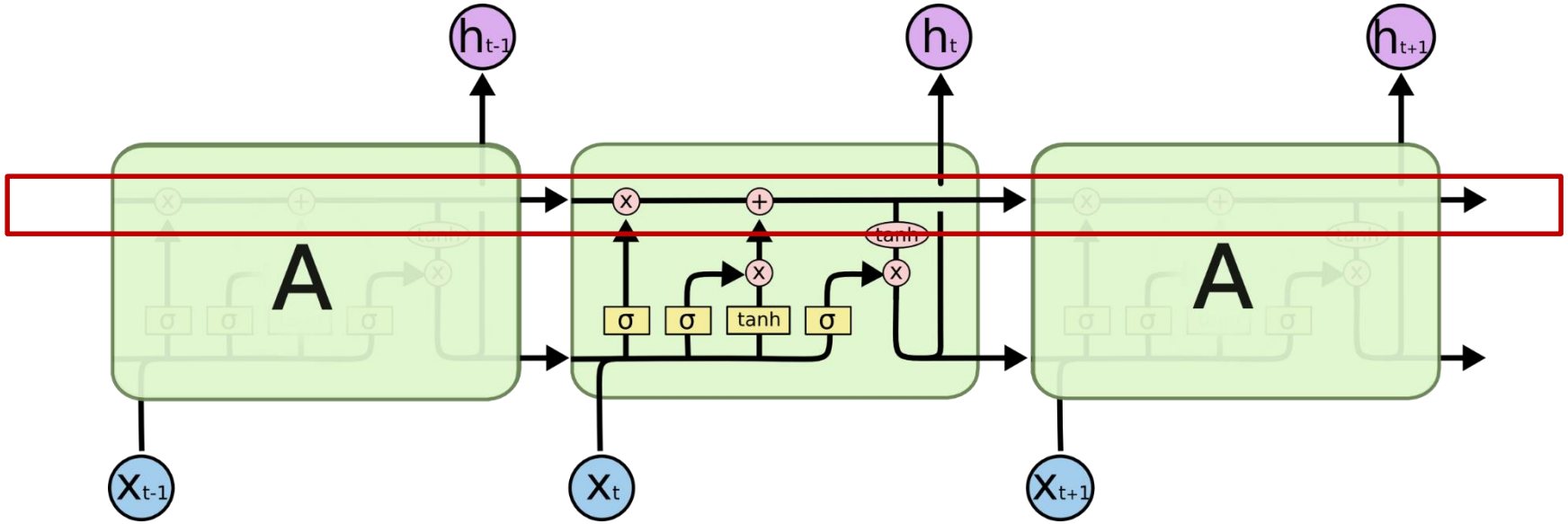
- Recurrent neurons receive past recurrent outputs and current input as inputs
- Processed through a tanh() activation function
  - As mentioned earlier, tanh() is the generally used activation for the hidden layer
- Current recurrent output passed to next higher layer and next time instant

Long Short-Term Memory



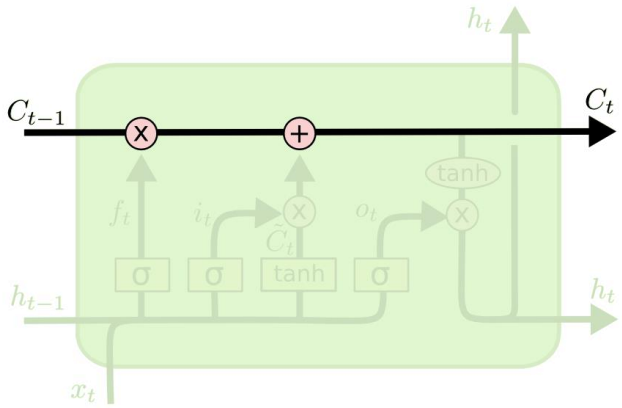
- The $\sigma()$ are *multiplicative gates* that decide if something is important or not
- Remember, every line actually represents a *vector*

LSTM: Constant Error Carousel



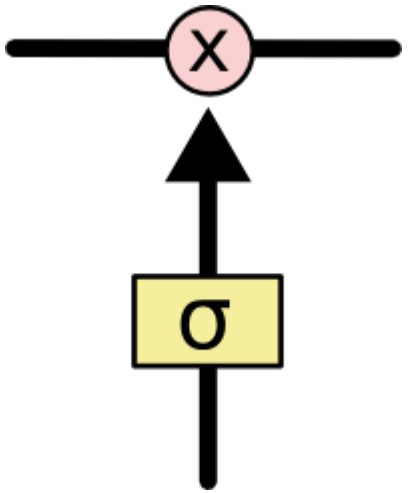
- Key component: a *remembered cell state*

LSTM: CEC



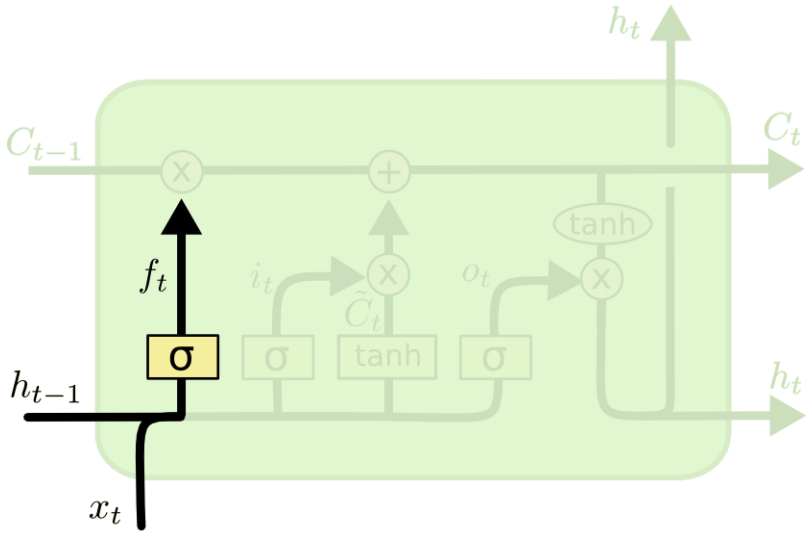
- C_t is the linear history carried by the *constant-error carousel*
- Carries information through, only affected by a gate
  - And *addition of history,* which too is gated..

LSTM: Gates



- Gates are simple sigmoidal units with outputs in the range (0,1)
- Controls how much of the information is to be let through

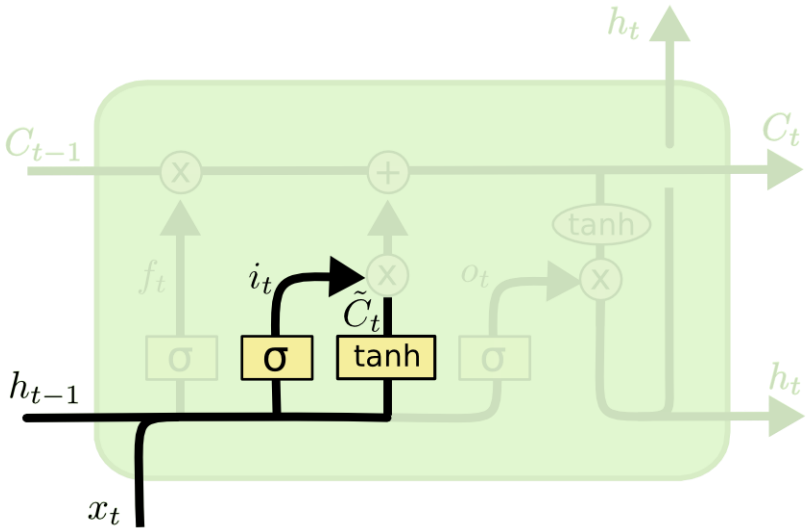
LSTM: Forget gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- The first gate determines whether to carry over the history or to forget it
  - More precisely, how much of the history to carry over
  - Also called the "forget" gate
  - Note, we're actually distinguishing between the cell memory C and the state h that is coming over time! They're related though

LSTM: Input gate

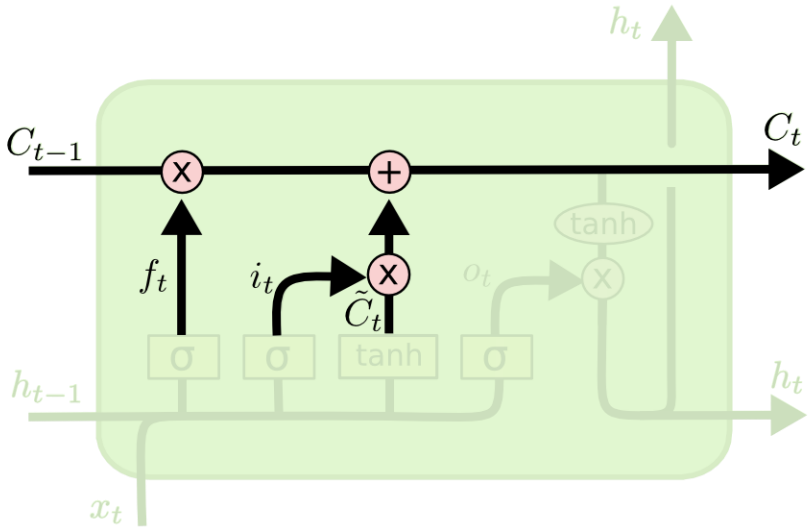


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- The second gate has two parts
  - A perceptron layer that determines if there's something interesting in the input
  - A gate that decides if its worth remembering

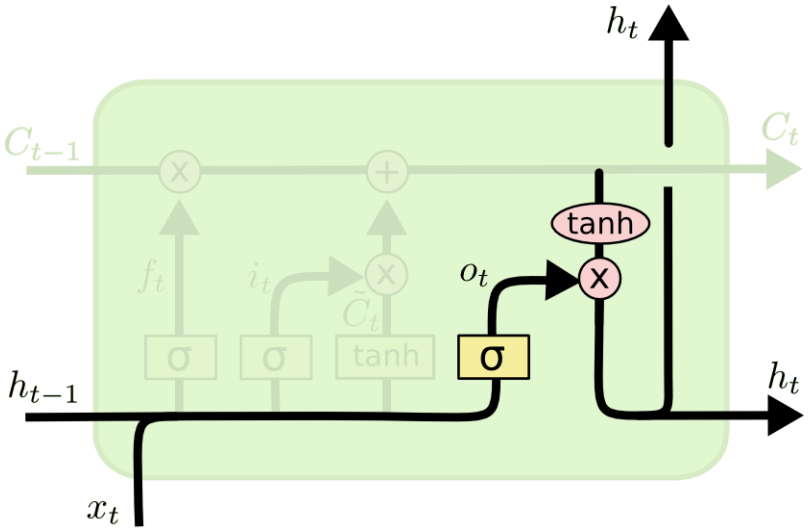
LSTM: Memory cell update



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- The second gate has two parts
  - A perceptron layer that determines if there's something interesting in the input
  - A gate that decides if its worth remembering
  - **If so its added to the current memory cell**

LSTM: Output and Output gate

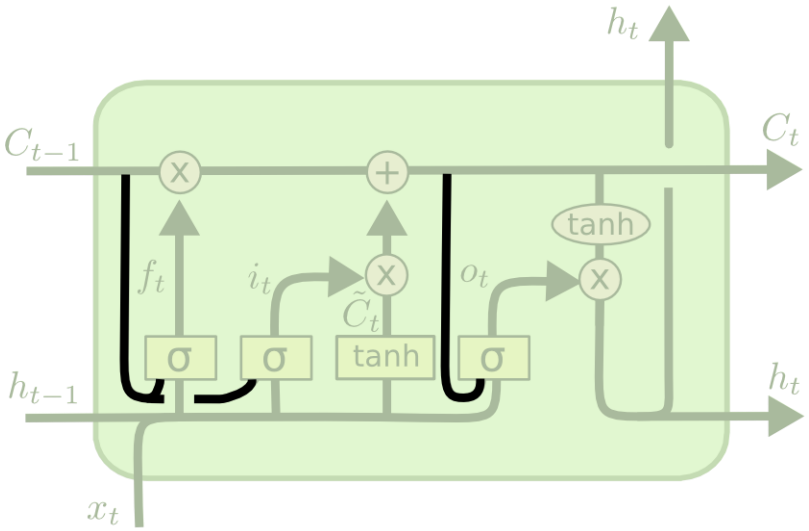


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

- The *output* of the cell
  - Simply compress it with tanh to make it lie between 1 and -1
    - Note that this compression no longer affects our ability to *carry* memory forward
  - While we're at it, lets toss in an output gate
    - To decide if the memory contents are worth reporting at *this* time

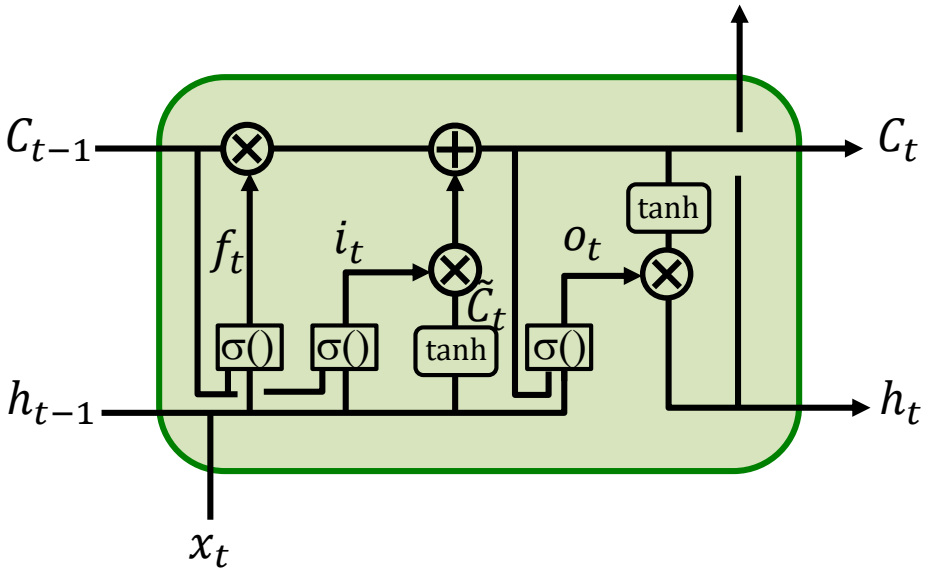
LSTM: The "Peephole" Connection



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

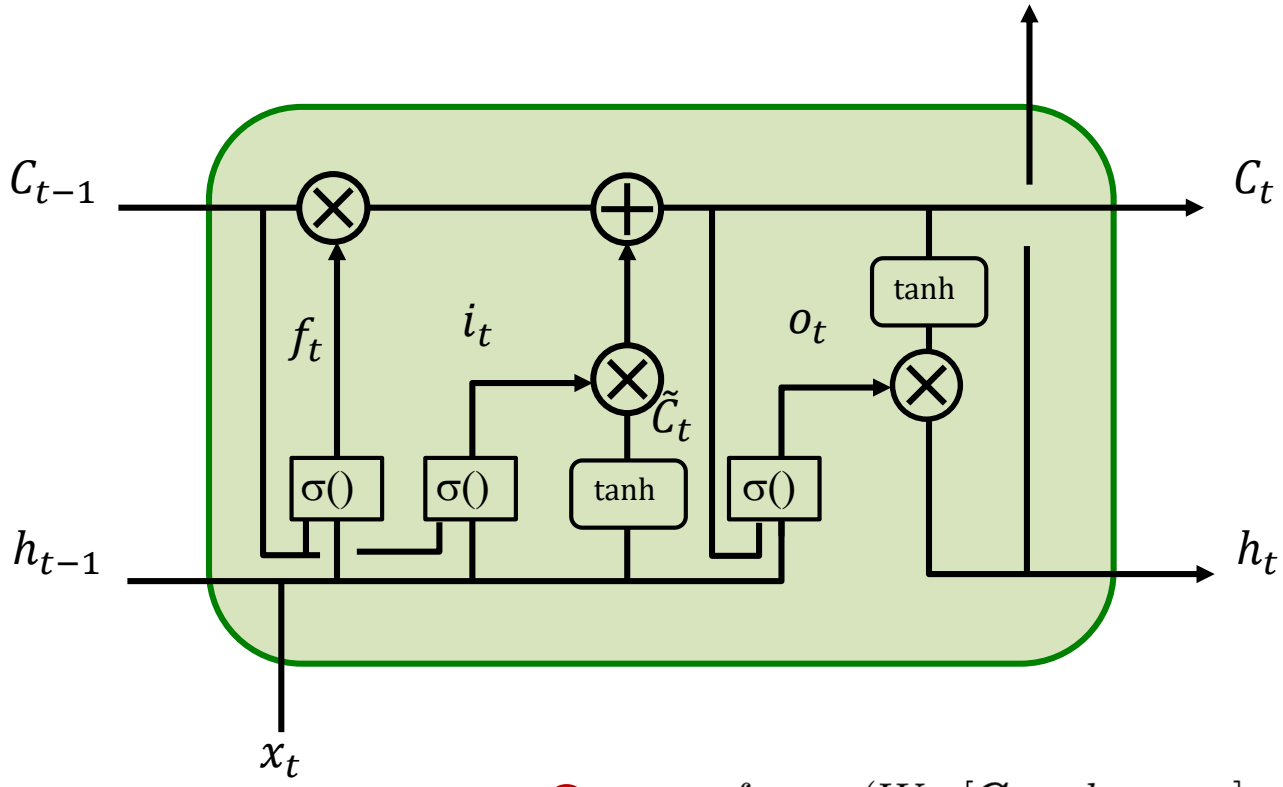
- Why not just let the cell directly influence the gates while at it
  - Party!!

The complete LSTM unit



- With input, output, and forget gates and the peephole connection..

Backpropagation rules: Forward



- Forward rules:

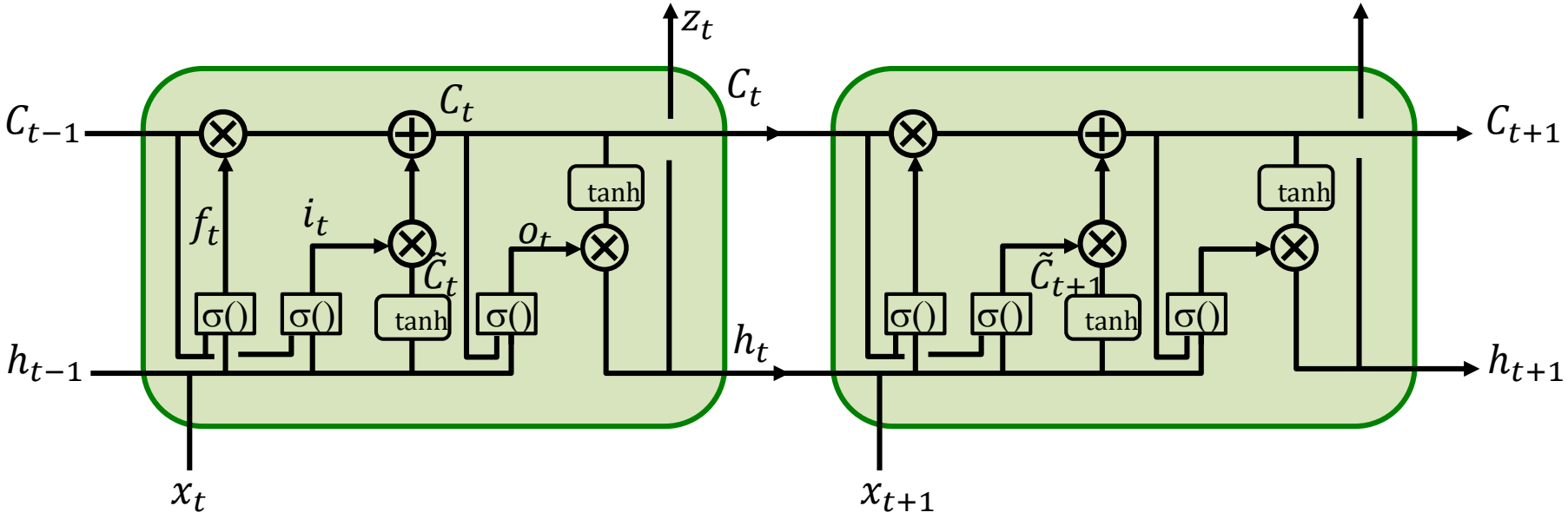
Gates

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Variables

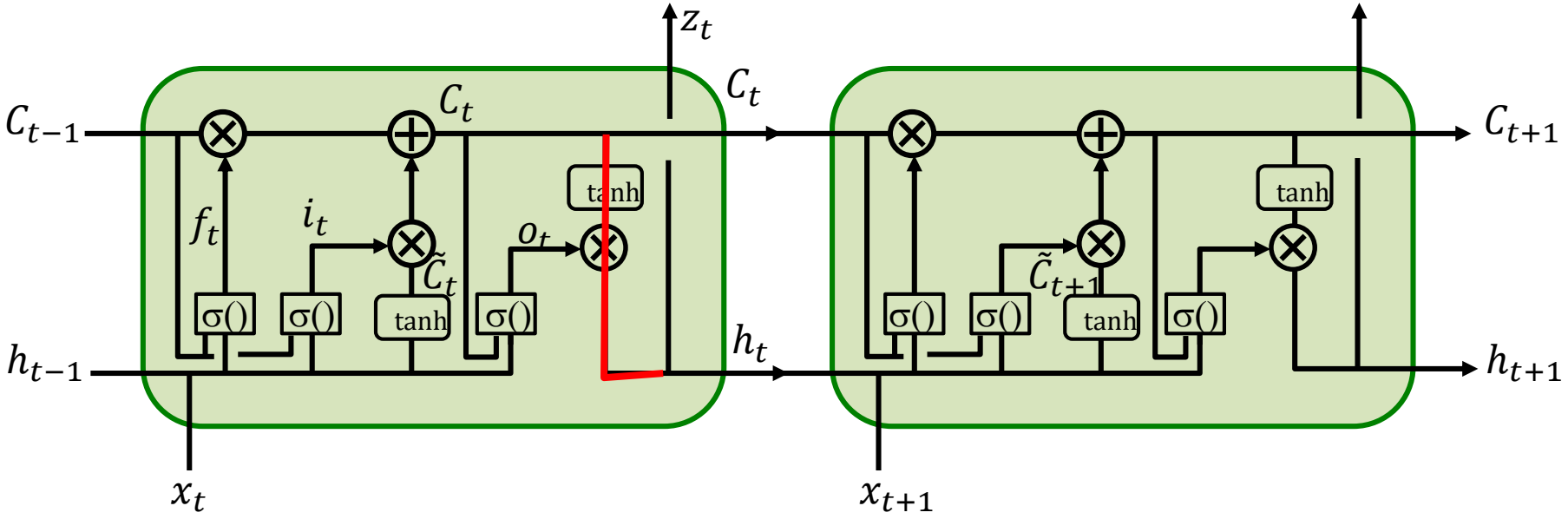
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Backpropagation rules: Backward



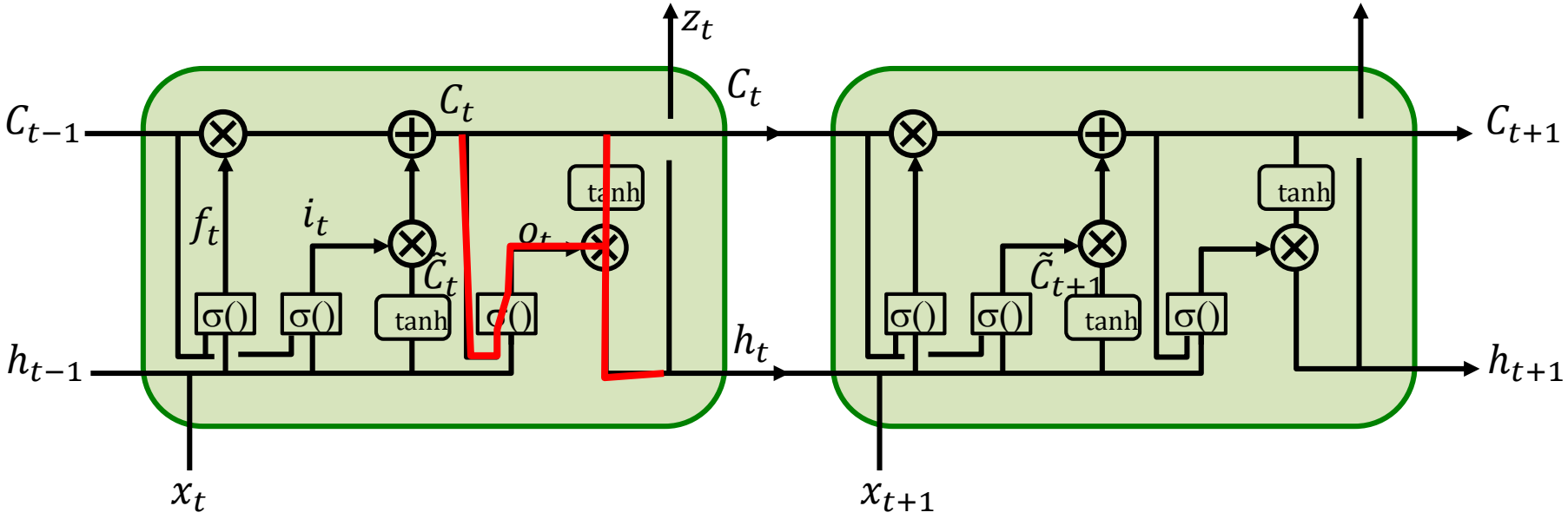
$\nabla_{C_t} Div =$

Backpropagation rules: Backward



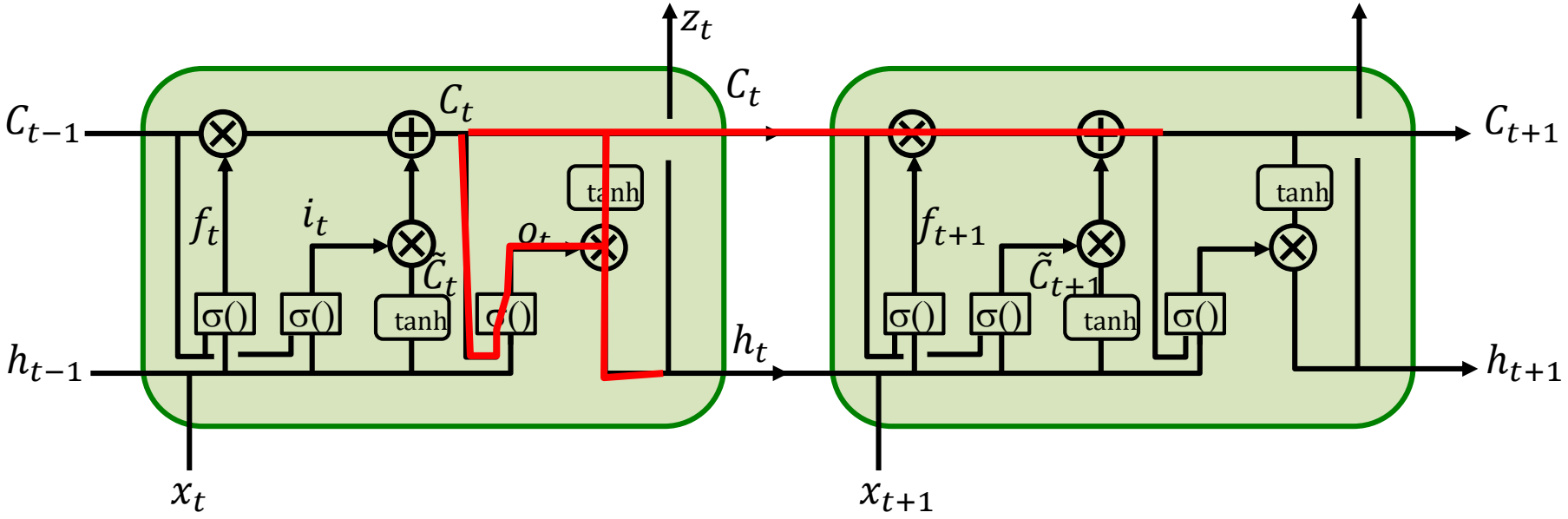
$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ o_t \circ tanh'(.) W_{Ch}$$

Backpropagation rules: Backward



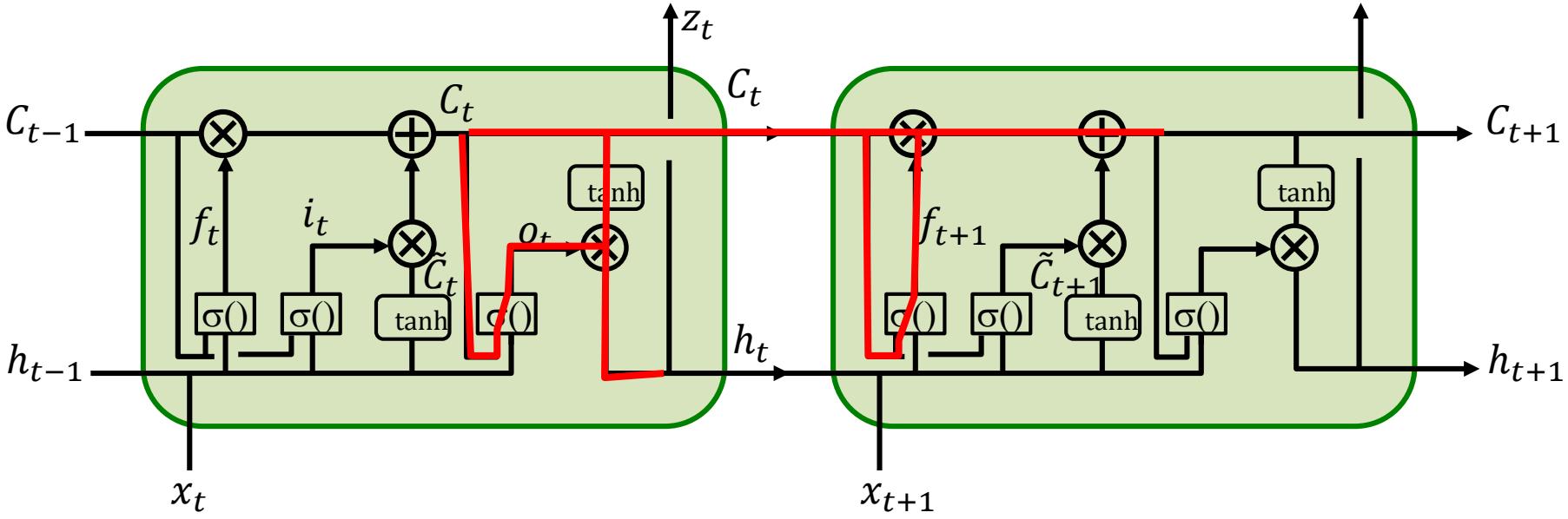
$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ (o_t \circ tanh'(.) W_{Ch} + tanh(.) \circ \sigma'(.) W_{Co})$$

Backpropagation rules: Backward



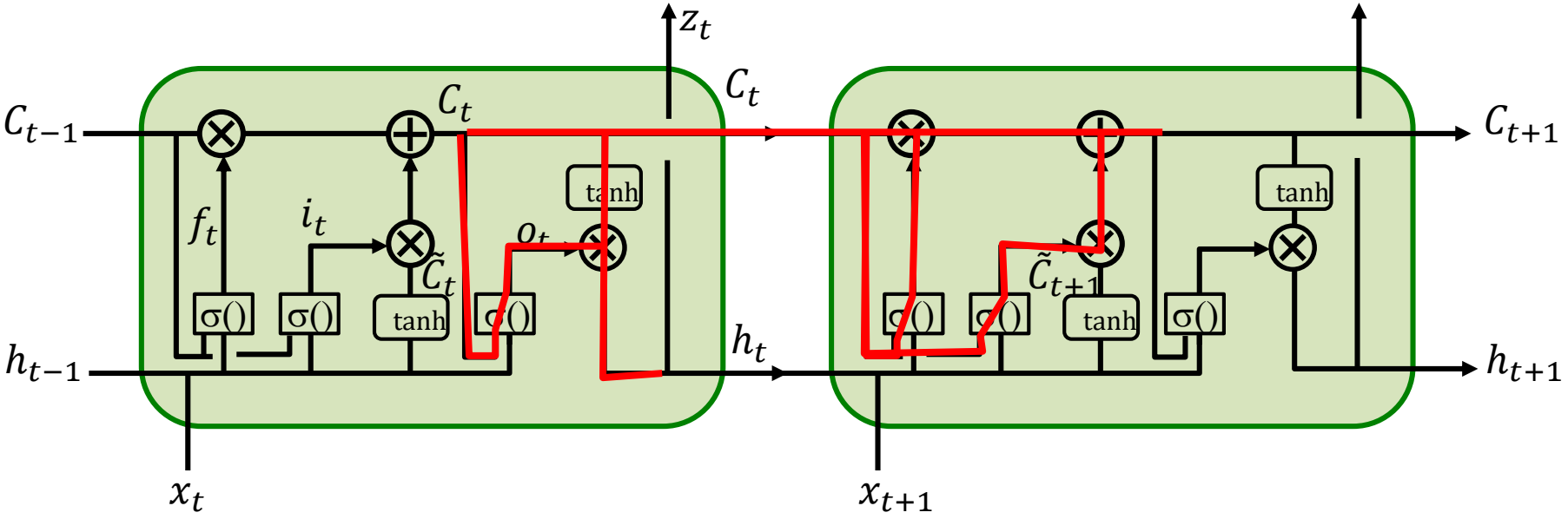
$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ (o_t \circ tanh'(.) W_{Ch} + tanh(.) \circ \sigma'(.) W_{Co}) +$$
$$\nabla_{h_t} C_{t+1} \circ f_{t+1} +$$

Backpropagation rules: Backward



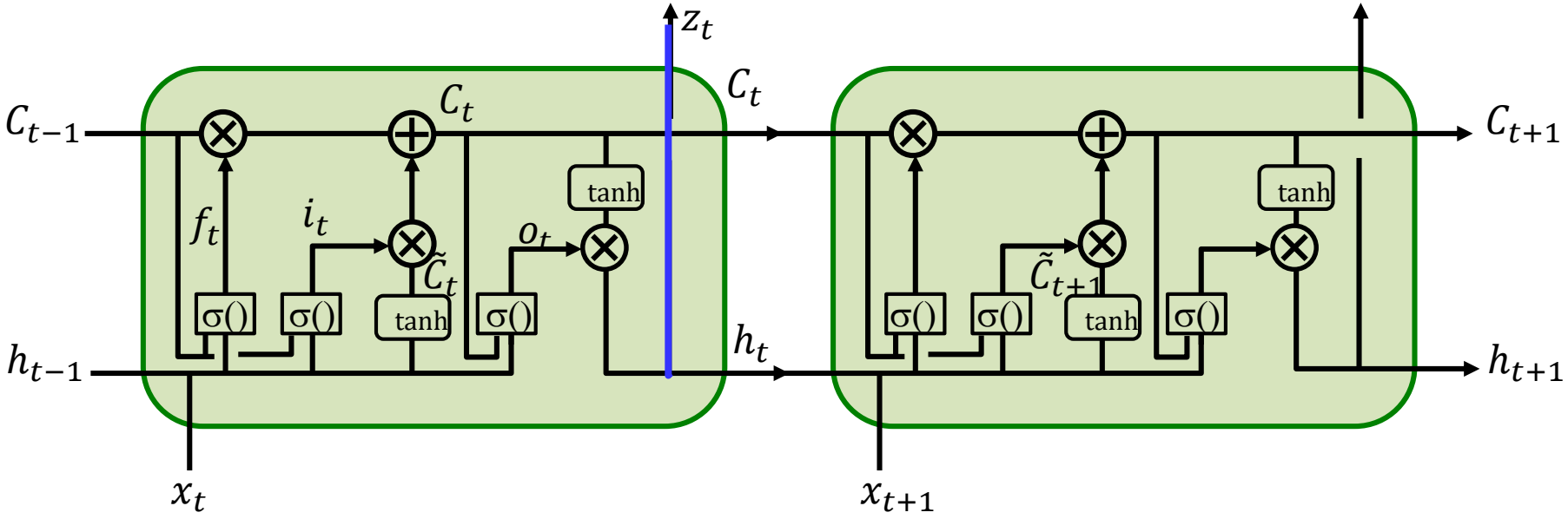
$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ (o_t \circ tanh'(.) W_{Ch} + tanh(.) \circ \sigma'(.) W_{Co}) +$$
$$\nabla_{h_t} C_{t+1} \circ \left(f_{t+1} + C_t \circ \sigma'(.) W_{Cf}\right)$$

Backpropagation rules: Backward



$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ (o_t \circ tanh'(.) W_{Ch} + tanh(.) \circ \sigma'(.) W_{Co}) +$$
$$\nabla_{h_t} C_{t+1} \circ \left(f_{t+1} + C_t \circ \sigma'(.) W_{Cf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{Ci}\right)$$

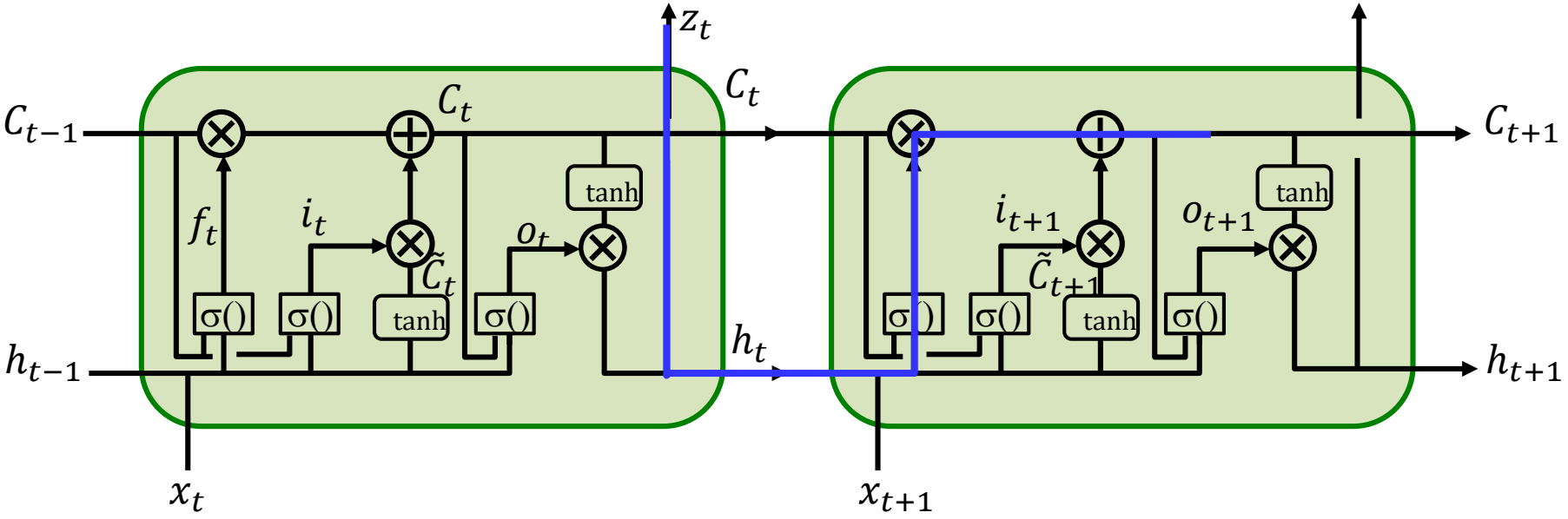
Backpropagation rules: Backward



$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ (o_t \circ tanh'(.) W_{Ch} + tanh(.) \circ \sigma'(.) W_{Co}) +$$
$$\nabla_{h_t} C_{t+1} \circ \left(f_{t+1} + C_t \circ \sigma'(.) W_{Cf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{Ci}\right)$$

$$\nabla_{h_t} Div = \nabla_{z_t} Div \nabla_{h_t} z_t$$

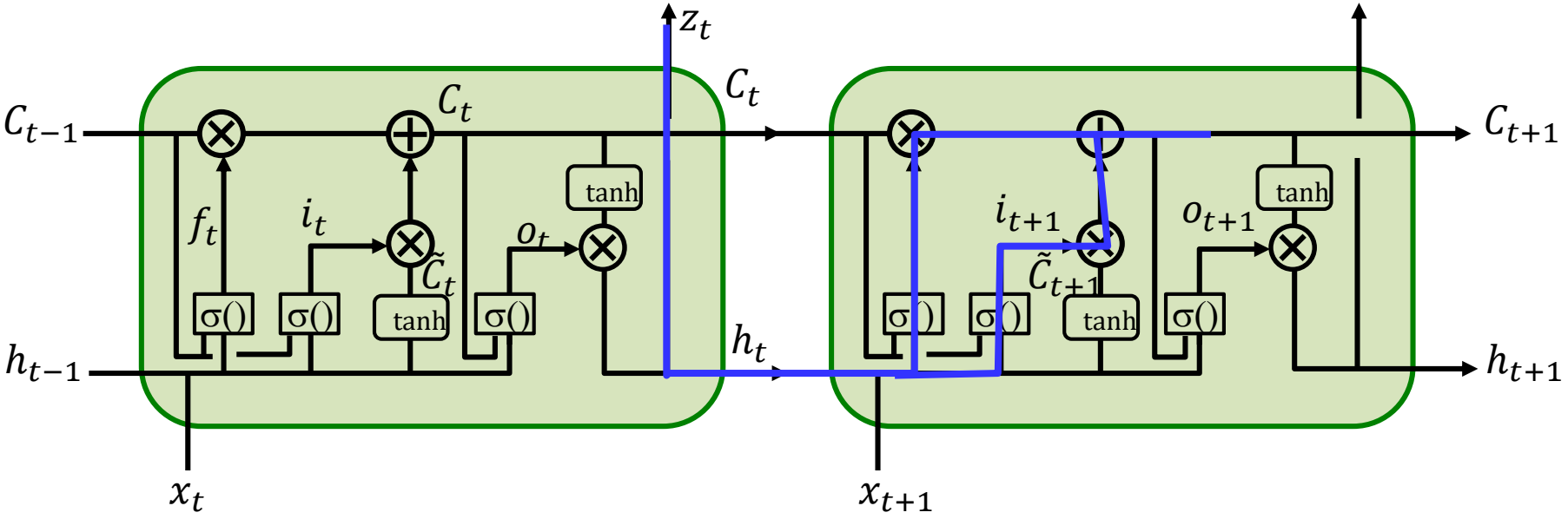
Backpropagation rules: Backward



$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ (o_t \circ tanh'(.) W_{Ch} + tanh(.) \circ \sigma'(.) W_{Co}) +$$
$$\nabla_{h_t} C_{t+1} \circ \left(f_{t+1} + C_t \circ \sigma'(.) W_{Cf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{Ci}\right)$$

$$\nabla_{h_t} Div = \nabla_{z_t} Div \nabla_{h_t} z_t + \nabla_{h_t} C_{t+1} \circ C_t \circ \sigma'(.) W_{hf}$$

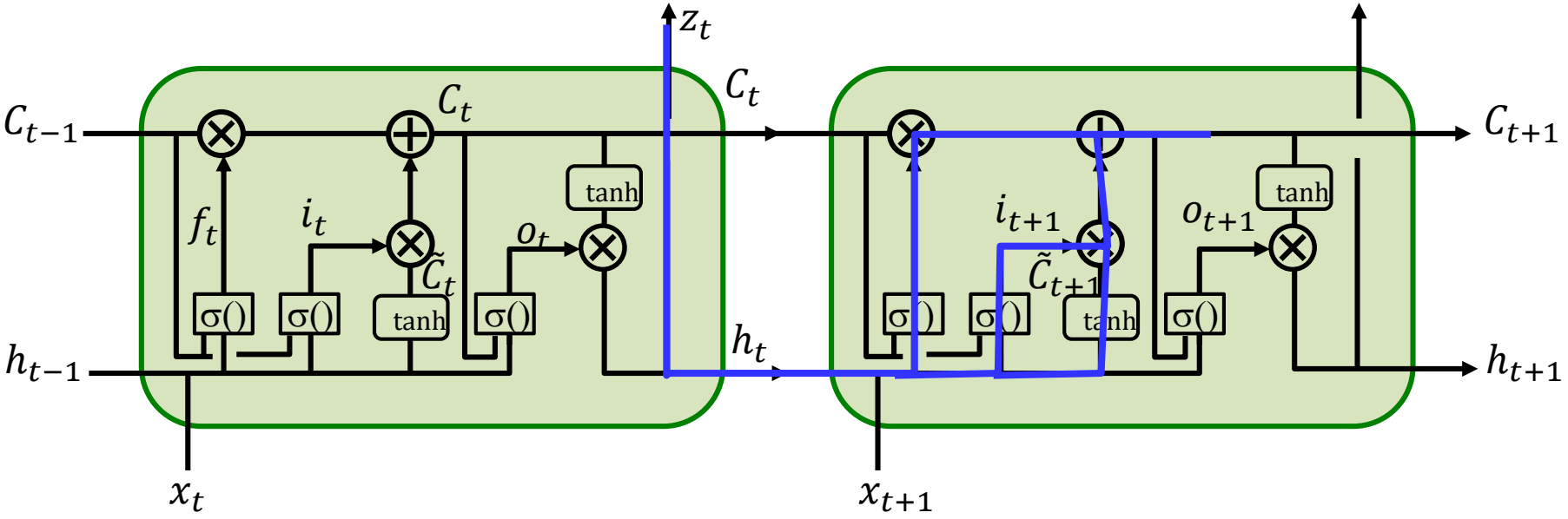
Backpropagation rules: Backward



$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ (o_t \circ tanh'(.) W_{Ch} + tanh(.) \circ \sigma'(.) W_{Co}) +$$
$$\nabla_{h_t} C_{t+1} \circ \left(f_{t+1} + C_t \circ \sigma'(.) W_{Cf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{Ci}\right)$$

$$\nabla_{h_t} Div = \nabla_{z_t} Div \nabla_{h_t} z_t + \nabla_{h_t} C_{t+1} \circ \left(C_t \circ \sigma'(.) W_{hf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{hi}\right)$$

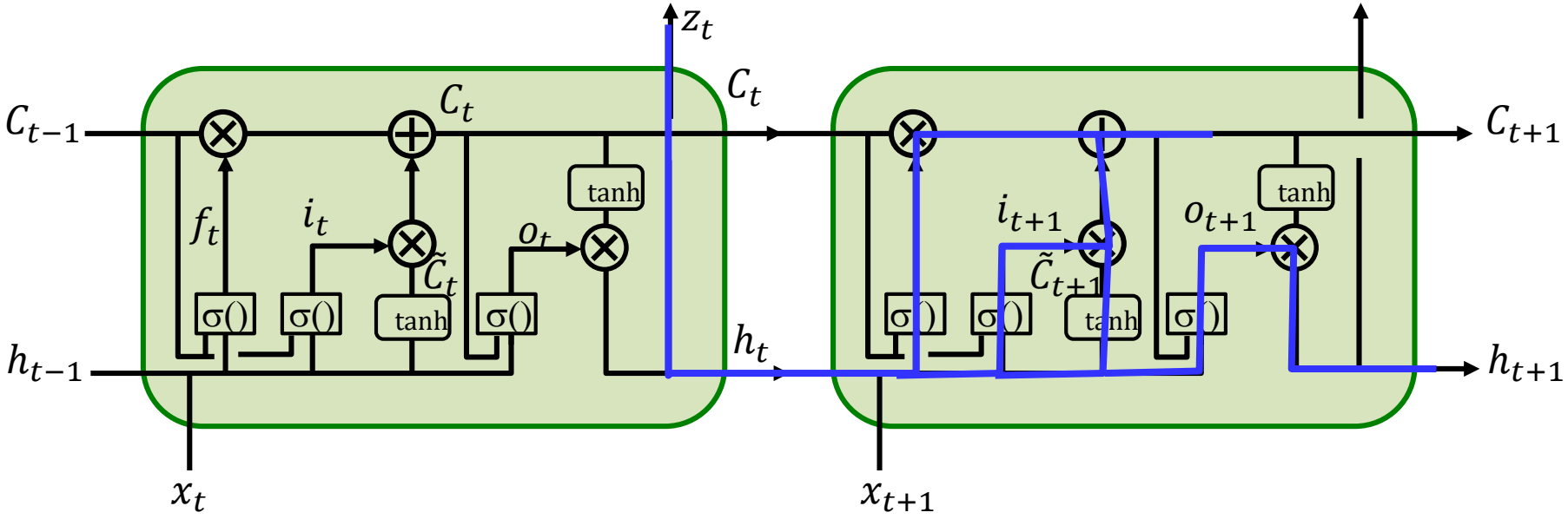
Backpropagation rules: Backward



$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ (o_t \circ tanh'(.) W_{Ch} + tanh(.) \circ \sigma'(.) W_{Co}) +$$
$$\nabla_{h_t} C_{t+1} \circ \left(f_{t+1} + C_t \circ \sigma'(.) W_{Cf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{Ci}\right)$$

$$\nabla_{h_t} Div = \nabla_{z_t} Div \nabla_{h_t} z_t + \nabla_{h_t} C_{t+1} \circ \left(C_t \circ \sigma'(.) W_{hf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{hi}\right) +$$
$$\nabla_{C_{t+1}} Div \circ i_{t+1} \circ tanh'(.) W_{hi}$$

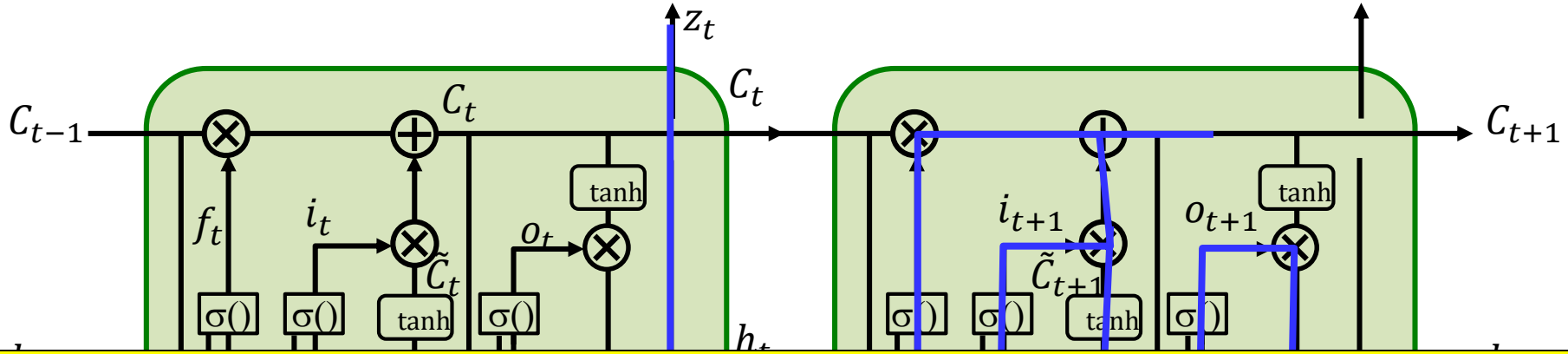
Backpropagation rules: Backward



$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ (o_t \circ tanh'(.) W_{Ch} + tanh(.) \circ \sigma'(.) W_{Co}) + \nabla_{h_t} C_{t+1} \circ \left(f_{t+1} + C_t \circ \sigma'(.) W_{Cf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{Ci}\right)$$

$$\nabla_{h_t} Div = \nabla_{z_t} Div \nabla_{h_t} z_t + \nabla_{h_t} C_{t+1} \circ \left(C_t \circ \sigma'(.) W_{hf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{hi}\right) + \nabla_{C_{t+1}} Div \circ o_{t+1} \circ tanh'(.) W_{hi} + \nabla_{h_{t+1}} Div \circ tanh(.) \circ \sigma'(.) W_{ho}$$

Backpropagation rules: Backward

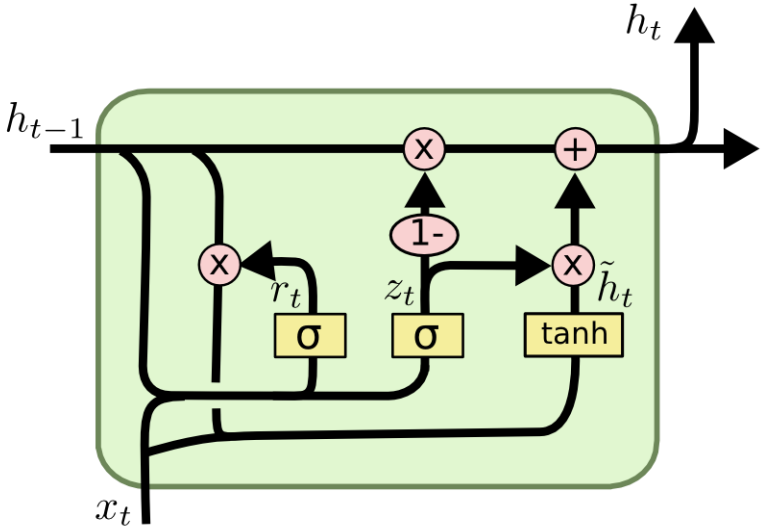


Not explicitly deriving the derivatives w.r.t weights; Left as an exercise

$$\nabla_{C_t} Div = \nabla_{h_t} Div \circ (o_t \circ tanh'(.) W_{Ch} + tanh(.) \circ \sigma'(.) W_{Co}) + \nabla_{h_t} C_{t+1} \circ \left(f_{t+1} + C_t \circ \sigma'(.) W_{Cf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{Ci}\right)$$

$$\nabla_{h_t} Div = \nabla_{z_t} Div \nabla_{h_t} z_t + \nabla_{h_t} C_{t+1} \circ \left(C_t \circ \sigma'(.) W_{hf} + \tilde{C}_{t+1} \circ \sigma'(.) W_{hi}\right) + \nabla_{C_{t+1}} Div \circ o_{t+1} \circ tanh'(.) W_{hi} + \nabla_{h_{t+1}} Div \circ tanh(.) \circ \sigma'(.) W_{ho}$$

Gated Recurrent Units: Lets simplify the LSTM



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

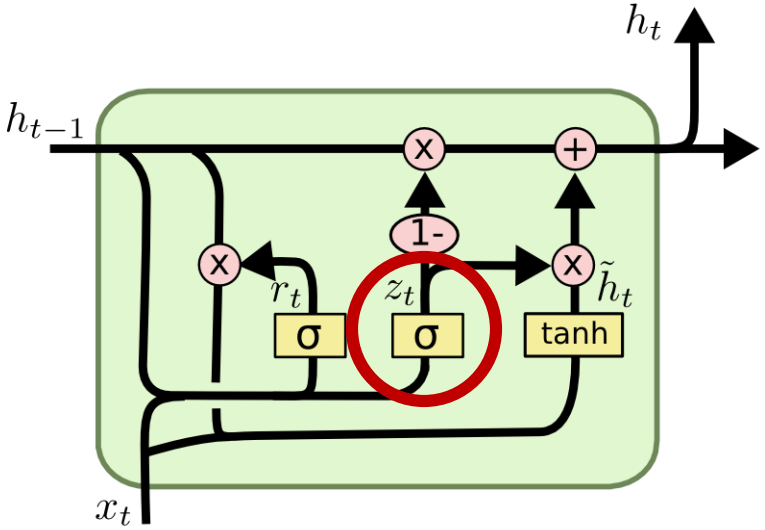
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Simplified LSTM which addresses some of your concerns of *why*

Gated Recurrent Units: Lets simplify the LSTM



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

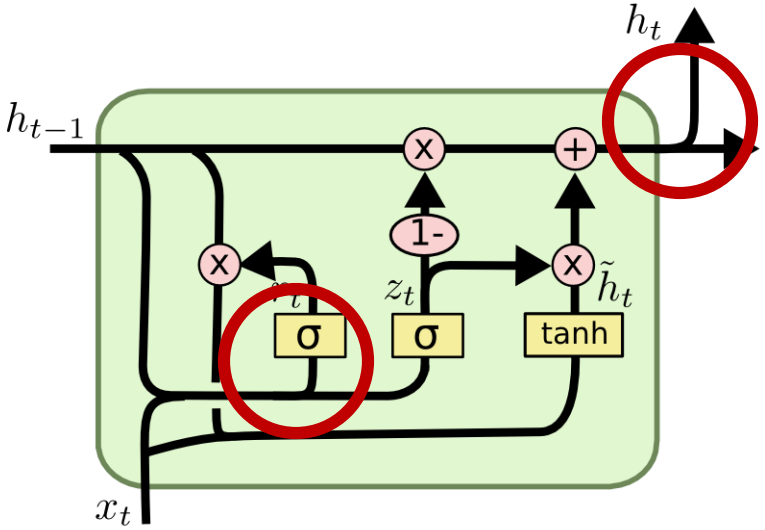
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Combine forget and input gates
  - In new input is to be remembered, then this means old memory is to be forgotten
    - Why compute twice?

Gated Recurrent Units: Lets simplify the LSTM



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

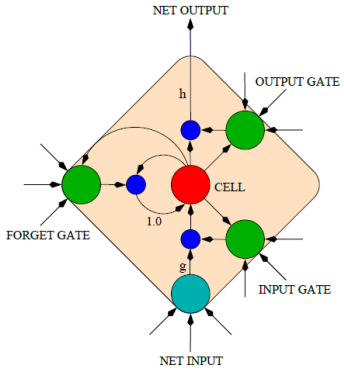
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Don't bother to separately maintain compressed and regular memories
  - Pointless computation!
- But compress it before using it to decide on the usefulness of the current input!

LSTM Equations

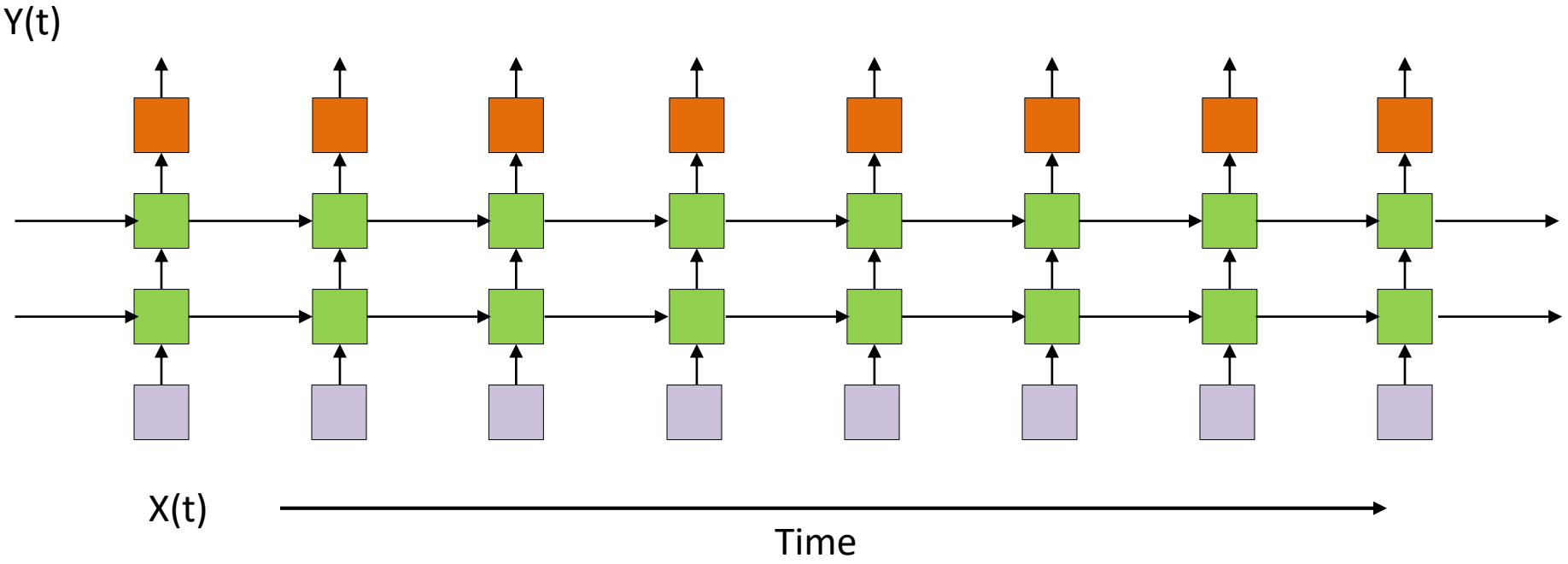
- $i = \sigma(x_t U^i + s_{t-1} W^i)$
- $f = \sigma(x_t U^f + s_{t-1} W^f)$
- $o = \sigma(x_t U^o + s_{t-1} W^o)$
- $g = \tanh(x_t U^g + s_{t-1} W^g)$
- $c_t = c_{t-1} \circ f + g \circ i$
- $s_t = \tanh(c_t) \circ o$
- $y = softmax(V s_t)$

- i: input gate, how much of the new information will be let through the memory cell.
- f: forget gate, responsible for information should be thrown away from memory cell.
- o: output gate, how much of the information will be passed to expose to the next time step.
- g: self-recurrent which is equal to standard RNN
- c_t: internal memory of the memory cell
- s_t: hidden state
- y: final output



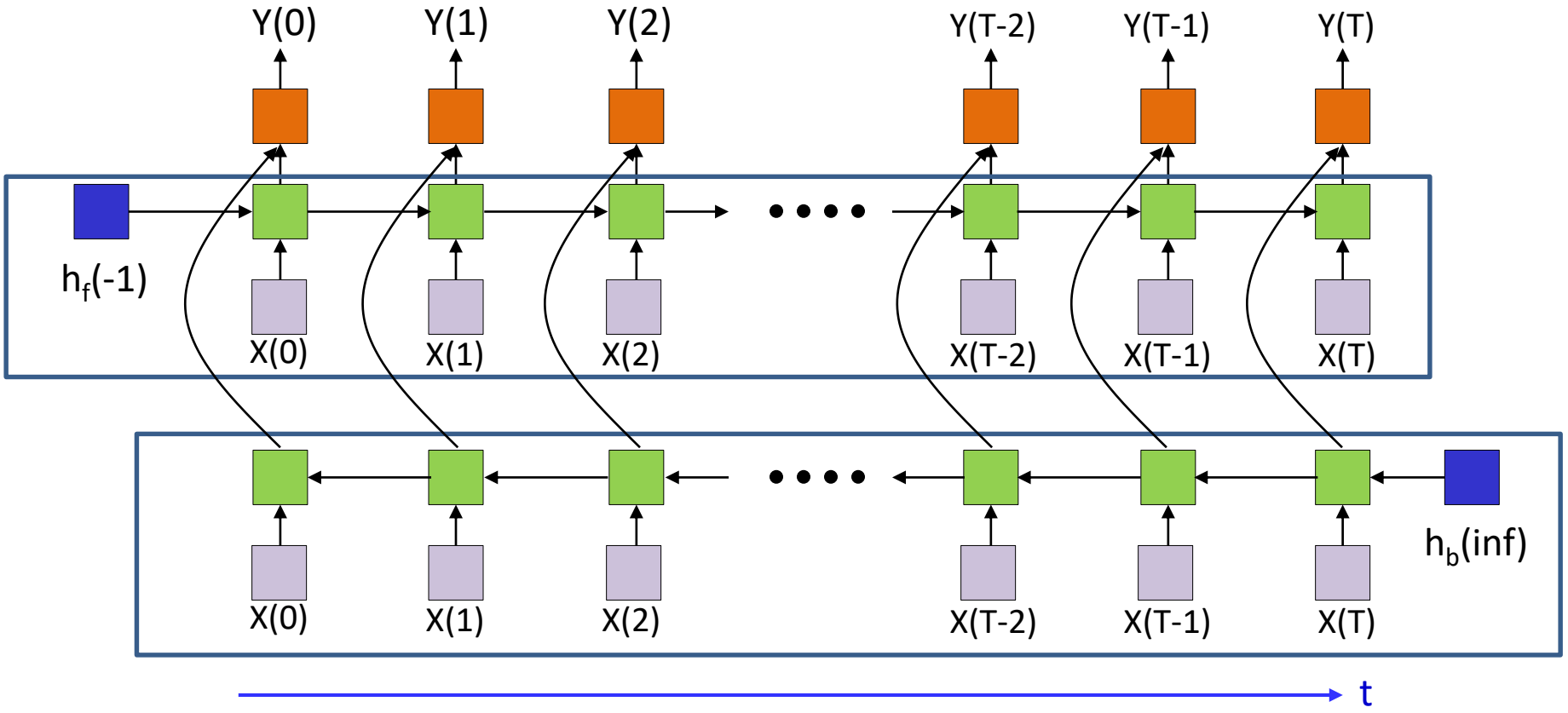
LSTM Memory Cell

LSTM architectures example



- Each green box is now an entire LSTM or GRU unit
- Also keep in mind each box is an *array* of units

Bidirectional LSTM



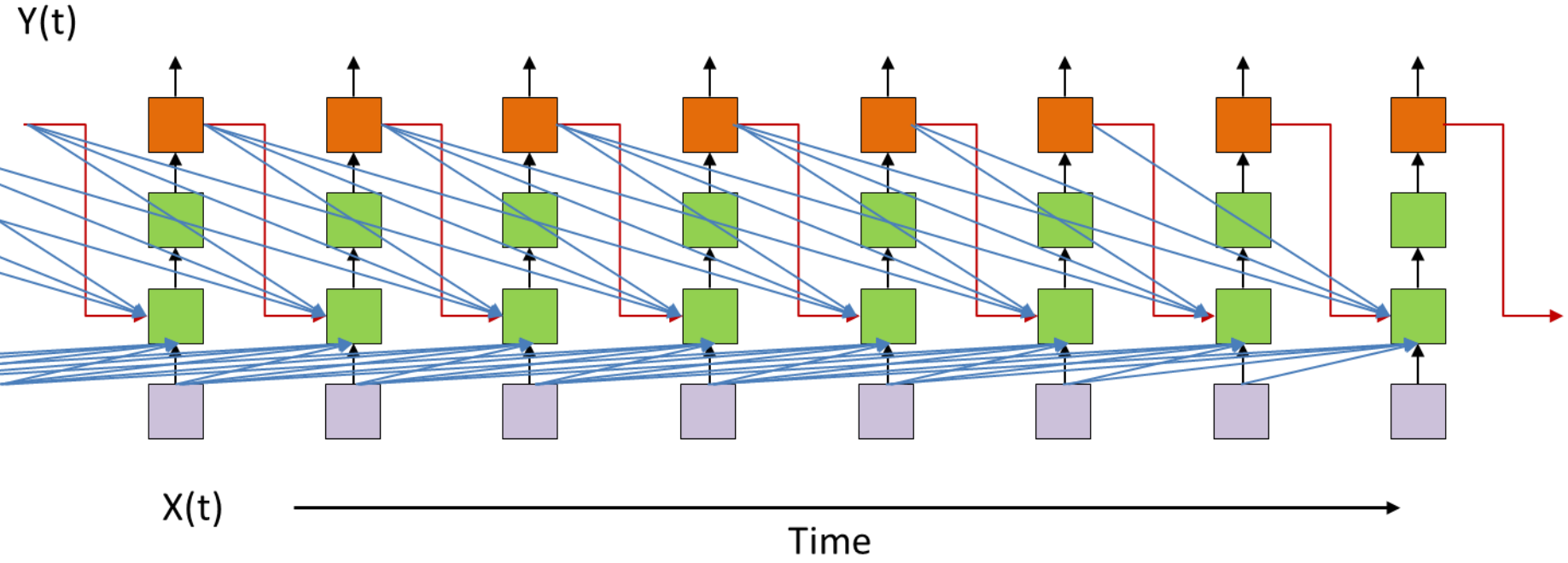
- Like the BRNN, but now the hidden nodes are LSTM units.
- Can have multiple layers of LSTM units in either direction
  - Its also possible to have MLP feed-forward layers between the hidden layers..
- The output nodes (orange boxes) may be complete MLPs

Some typical problem settings

- Lets consider a few typical problems
- Issues:
  - How to define the divergence()
  - How to compute the gradient
  - How to backpropagate

  - Specific problem: The constant error carousel..

Time series prediction using NARX nets



- NARX networks are commonly used for *scalar* time series prediction
  - All boxes are scalar
  - Sigmoid activations are commonly used in the hidden layer(s)
    - Linear activation in output layer
- The network is trained to minimize the L_2 divergence between desired and actual output
  - NARX networks are less susceptible to vanishing gradients than conventional RNNs
  - Training often uses methods *other* than backprop/gradient descent, e.g. simulated annealing or genetic algorithms

Example of Narx Network

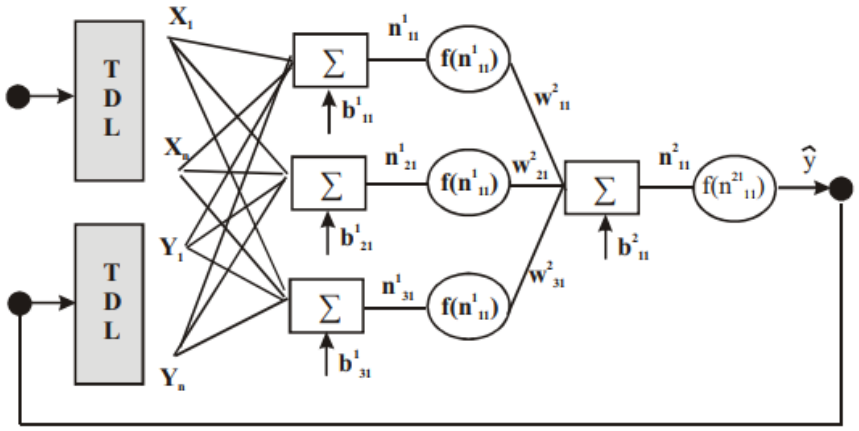
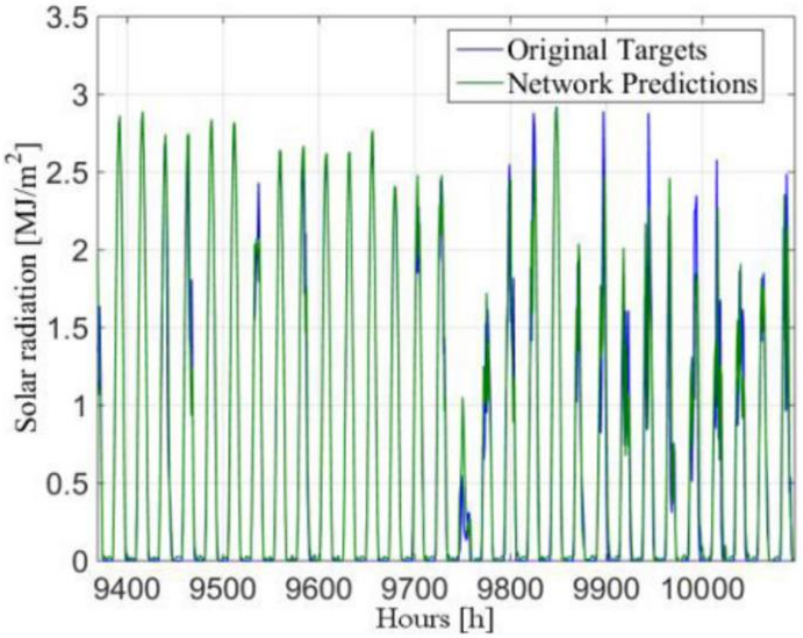


Fig. 1. Chosen structures of the NARX network: closed-loop.

Inputs may use either past *predicted* output values, or past *true* values or the past error in prediction

- "Solar and wind forecasting by NARX neural networks," Piazza, Piazza and Vitale, 2016
- Data: hourly global solar irradiation (MJ/m2), hourly wind speed (m/s) measured at two meters above ground level and the mean hourly temperature recorded during seven years, from 2002 to 2008
- Target: Predict solar irradiation and wind speed from temperature readings

Example of NARX Network: Results



- Used GA to train the net.
- NARX nets are generally the structure of choice for time series prediction problems

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;
}
```

Language modelling using RNNs

Four score and seven years ???

A B R A H A M L I N C O L ??

- Problem: Given a sequence of words (or characters) predict the next one

Language modelling: Representing words

- Represent words as one-hot vectors
  - Pre-specify a vocabulary of N words in fixed (e.g. lexical) order
    - *E.g. [A AARDVARD AARON ABACK ABACUS… ZZYP]*
  - Represent each word by an N-dimensional vector with N-1 zeros and a single 1 (in the position of the word in the ordered list of words)

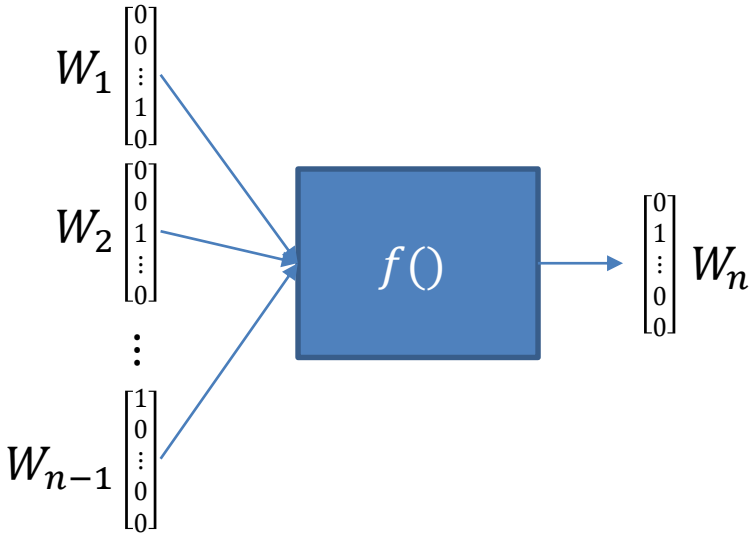
- Characters can be similarly represented
  - English will require about 100 characters, to include both cases, special characters such as commas, hyphens, apostrophes, etc., and the space character

Predicting words

Four score and seven years ???

$W_n = f(\text{W\_1}, W_1, \ldots, W_{n-1})$

Nx1 one-hot vectors



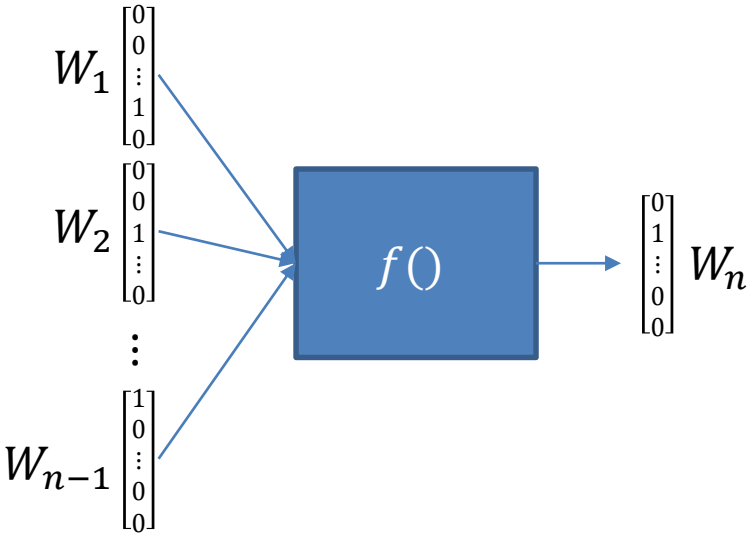
- Given one-hot representations of $W_1 \ldots W_{n-1}$, predict W_n

Predicting words

Four score and seven years ???

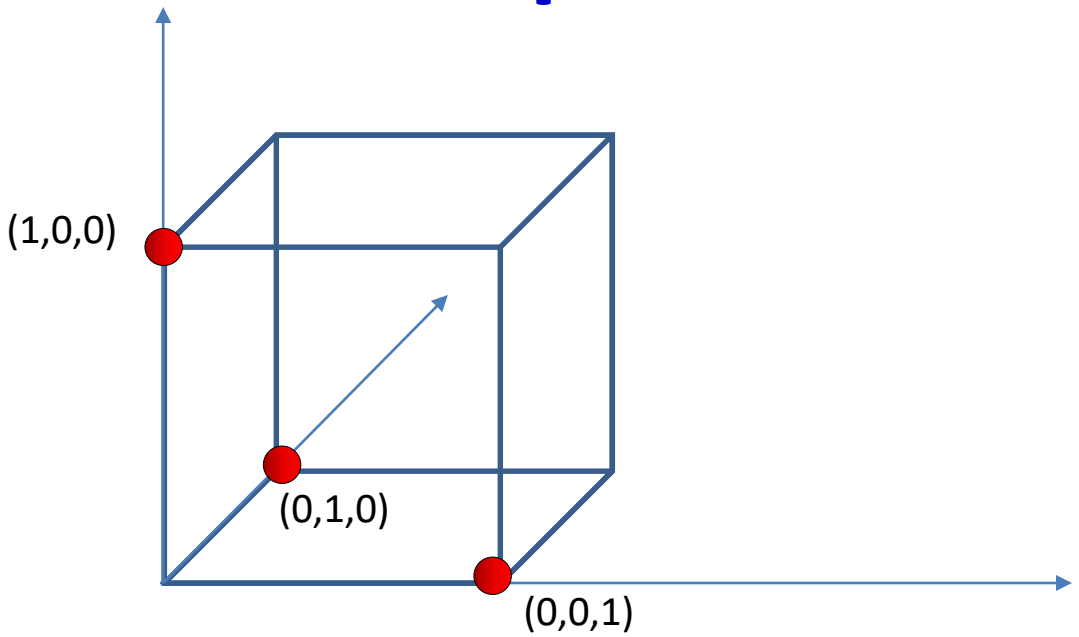
$W_n = f(\text{W\_1}, W_1, \ldots, W_{n-1})$

Nx1 one-hot vectors



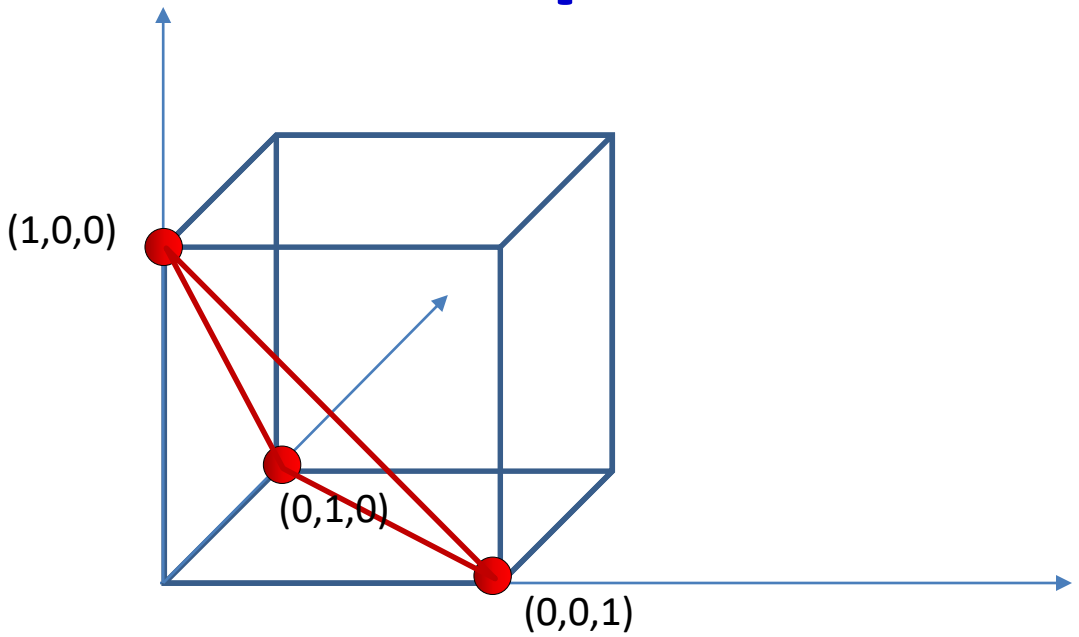
- Given one-hot representations of $W_1 \ldots W_{n-1}$, predict W_n
- **Dimensionality problem:** All inputs $W_1 \ldots W_{n-1}$ are both very high-dimensional and very sparse

The one-hot representation



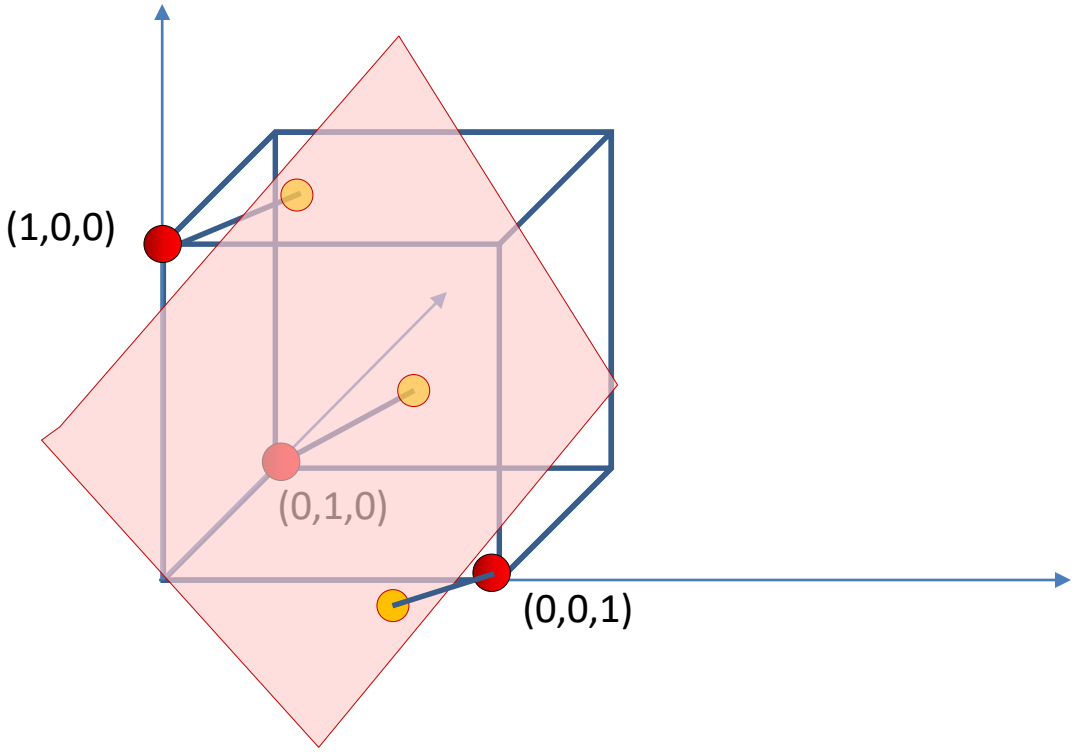
- The one hot representation uses only N corners of the 2^N corners of a unit cube
  - Actual volume of space used = 0
    - $(1, \varepsilon, \delta)$ has no meaning except for $\varepsilon = \delta = 0$
  - Density of points: $\mathcal{O}\left(\frac{N}{2^N}\right)$
- This is a tremendously inefficient use of dimensions

Why one-hot representation



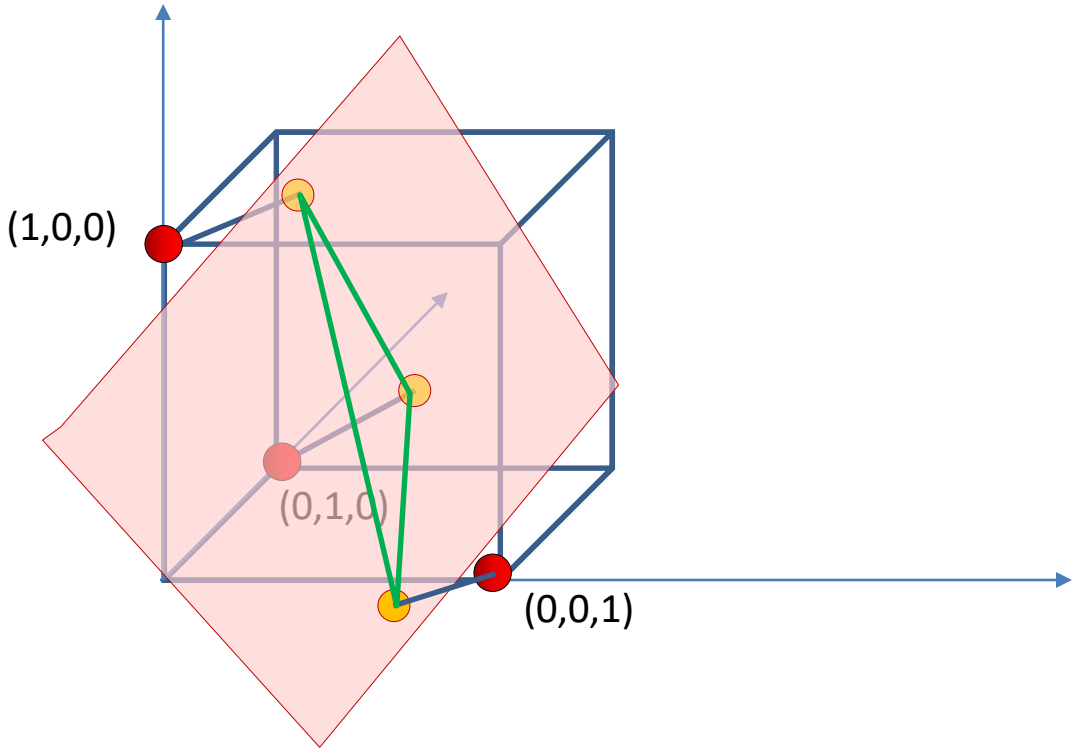
- The one-hot representation makes no assumptions about the relative importance of words
  - All word vectors are the same length
- It makes no assumptions about the relationships between words
  - The distance between every pair of words is the same

Solution to dimensionality problem



- Project the points onto a lower-dimensional subspace
  - The volume used is still 0, but density can go up by many orders of magnitude
    - Density of points: $\mathcal{O}\left(\frac{N}{2^M}\right)$

Solution to dimensionality problem

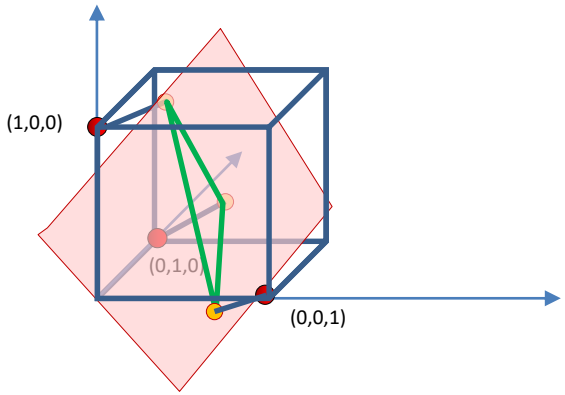


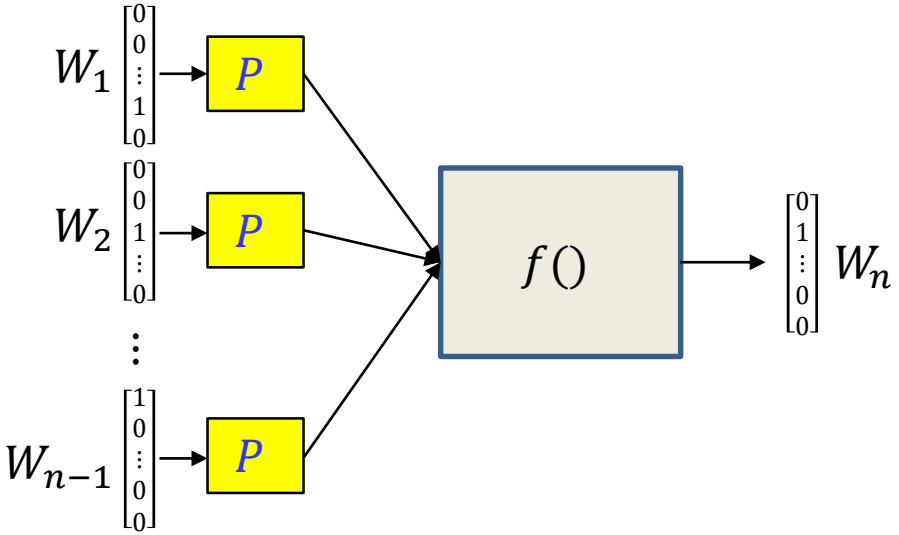
- Project the points onto a lower-dimensional subspace
  - The volume used is still 0, but density can go up by many orders of magnitude
    - Density of points: $\mathcal{O}\left(\frac{N}{2^M}\right)$
  - If properly learned, the distances between projected points will capture semantic relations between the words
    - This will also require linear transformation (stretching/shrinking/rotation) of the subspace

The *Projected* word vectors

Four score and seven years ???

$$W_n = f(PW_1, PW_2, \ldots, PW_{n-1})$$

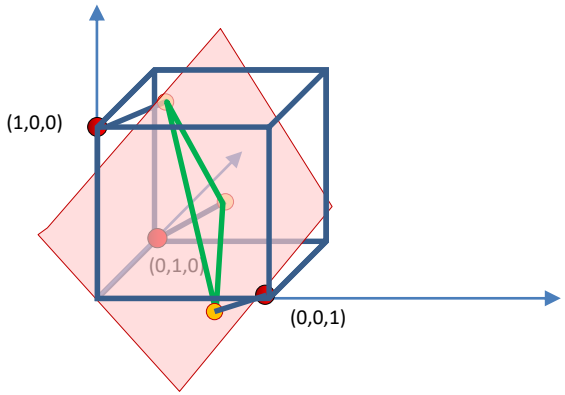


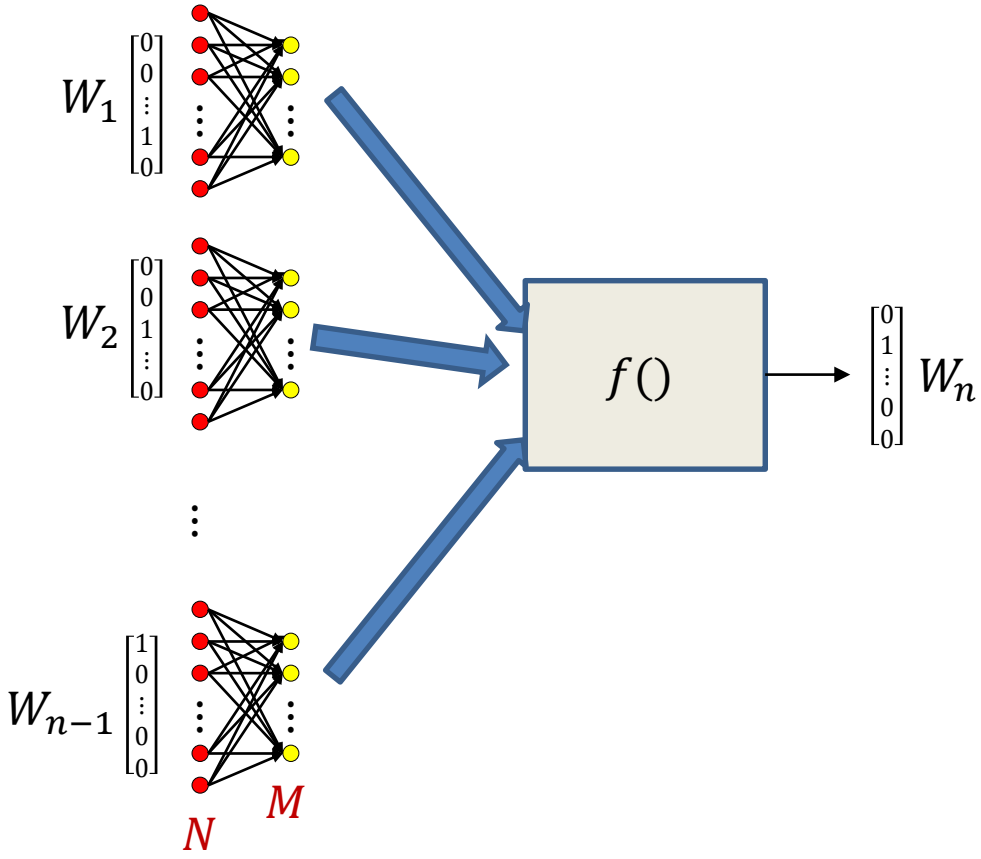


- *Project* the N-dimensional one-hot word vectors into a lower-dimensional space
  - Replace every one-hot vector W_i by PW_i
  - P is an $M \times N$ matrix
  - PW_i is now an M-dimensional vector
  - *Learn P* using an appropriate objective
    - Distances in the projected space will reflect relationships imposed by the objective

“Projection”

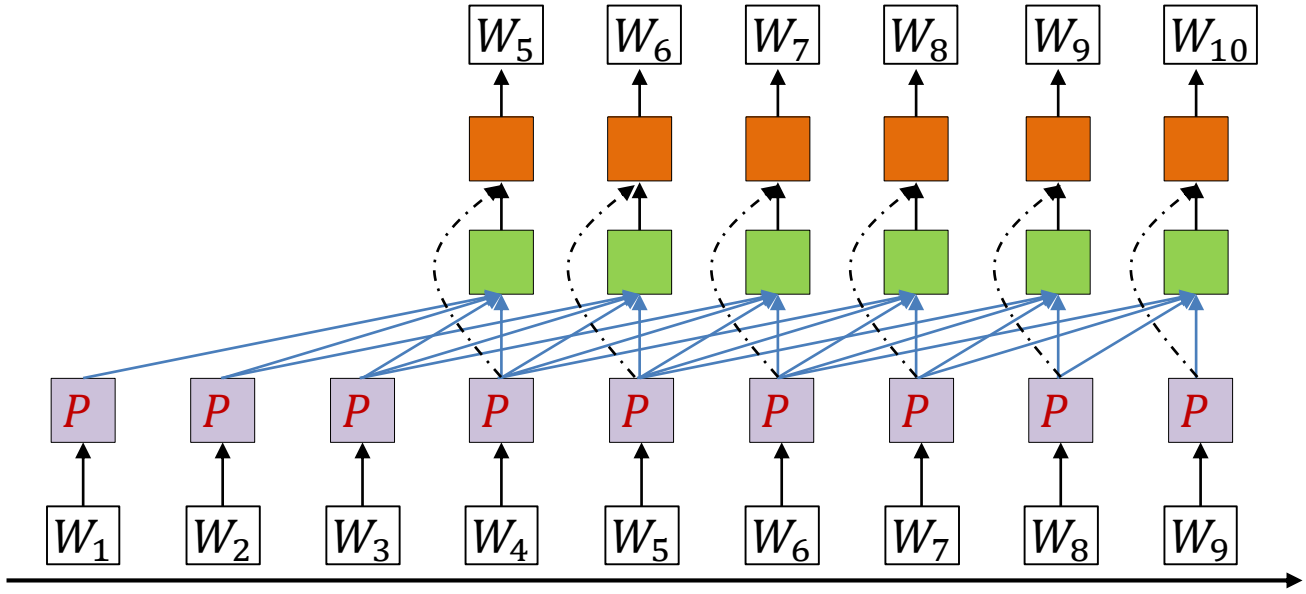
$$W_n = f(PW_1, PW_2, \ldots, PW_{n-1})$$





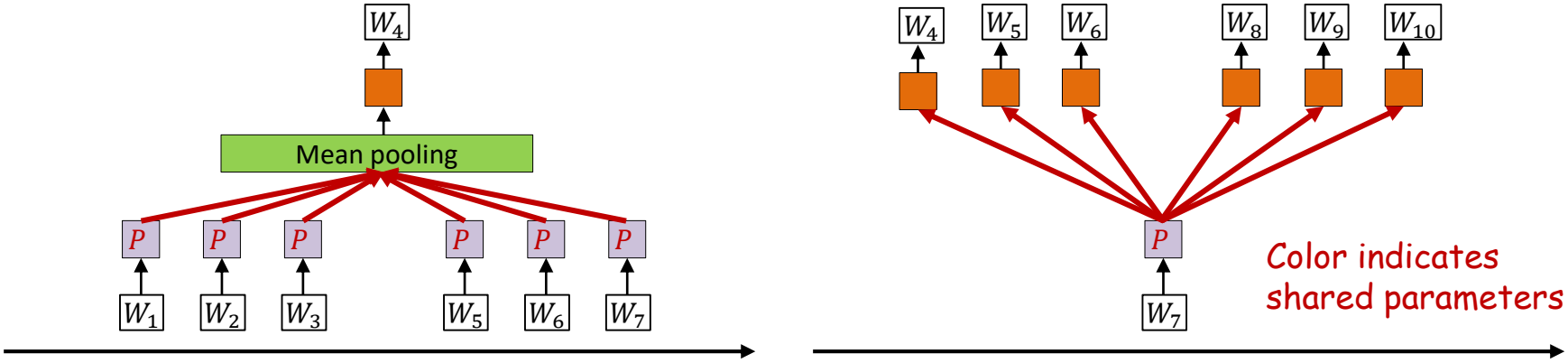
- *P* is a simple linear transform
- A single transform can be implemented as a layer of M neurons with linear activation
- The transforms that apply to the individual inputs are all M-neuron linear-activation subnets with tied weights

Predicting words: The TDNN model



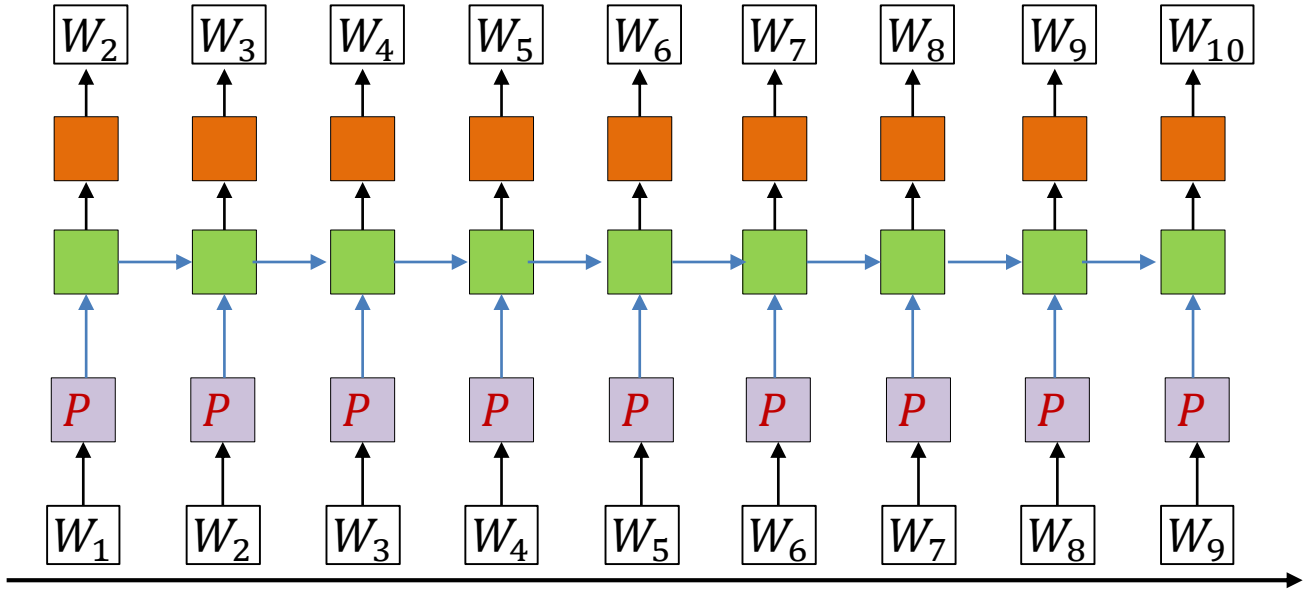
- Predict each word based on the past N words
  - "A neural probabilistic language model", Bengio et al. 2003
  - Hidden layer has Tanh() activation, output is softmax
- One of the outcomes of learning this model is that we also learn low-dimensional representations PW of words

Alternative models to learn projections



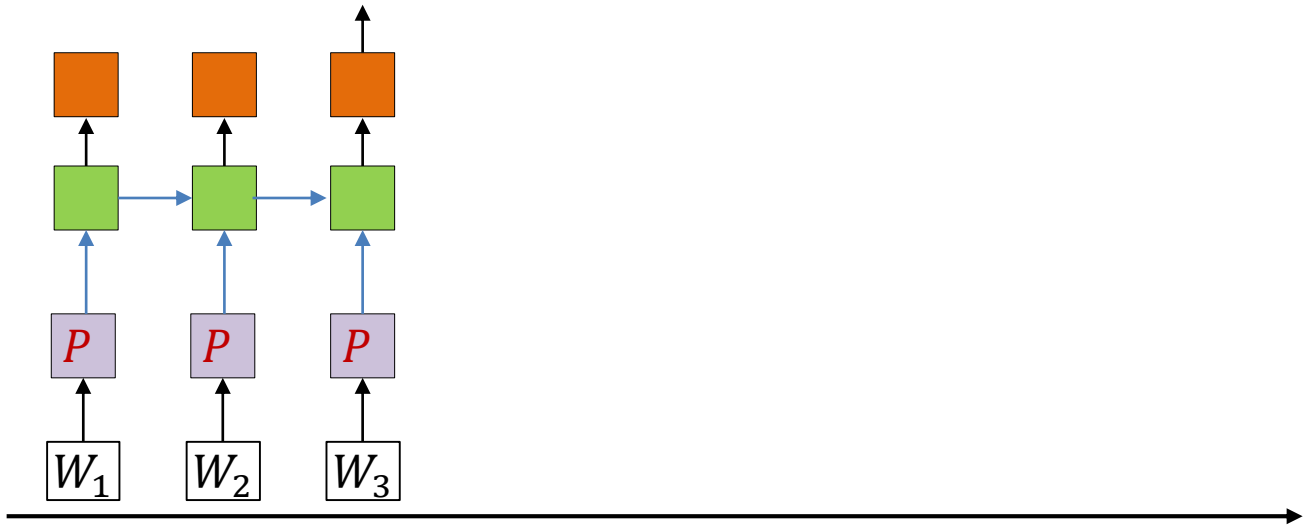
- Soft bag of words: Predict word based on words in immediate context
  - Without considering specific position
- Skip-grams: Predict adjacent words based on current word
- More on these in a future lecture

Generating Language: The model



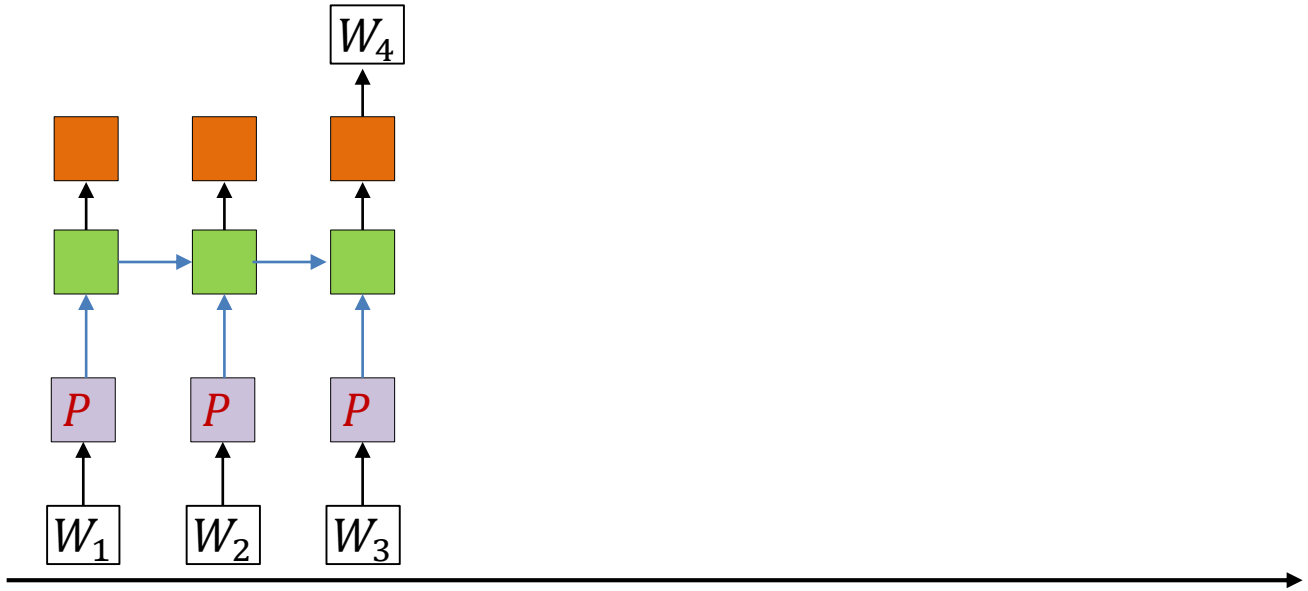
- The hidden units are (one or more layers of) LSTM units
- Trained via backpropagation from a lot of text

Generating Language: Synthesis



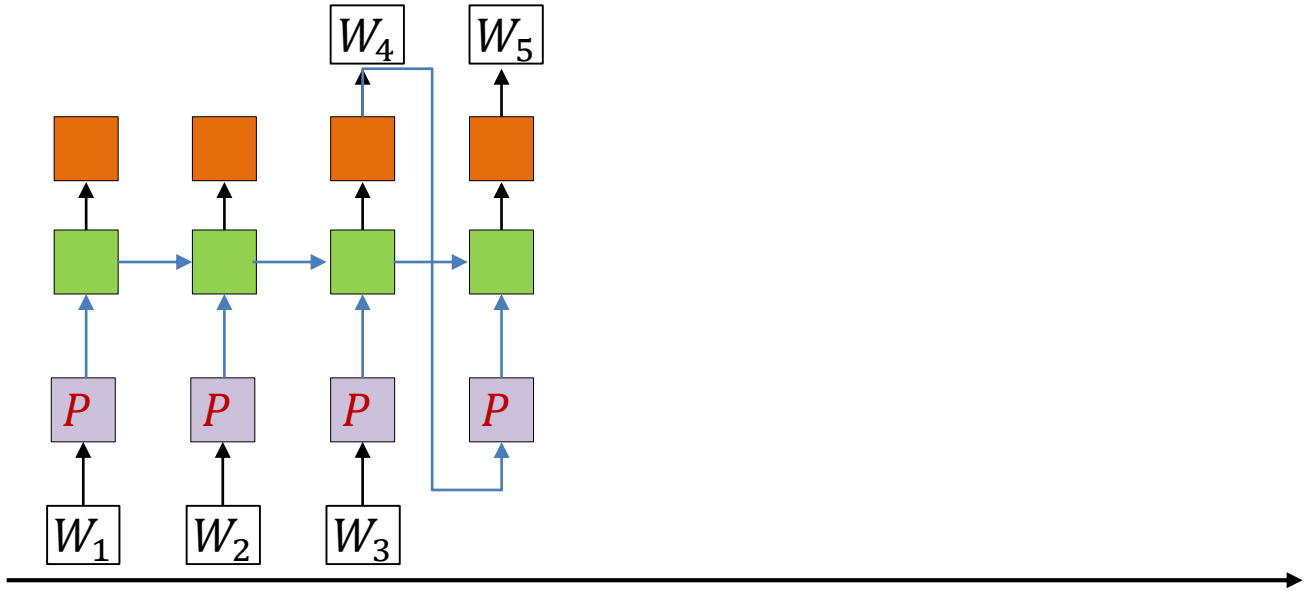
- On trained model : Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector

Generating Language: Synthesis



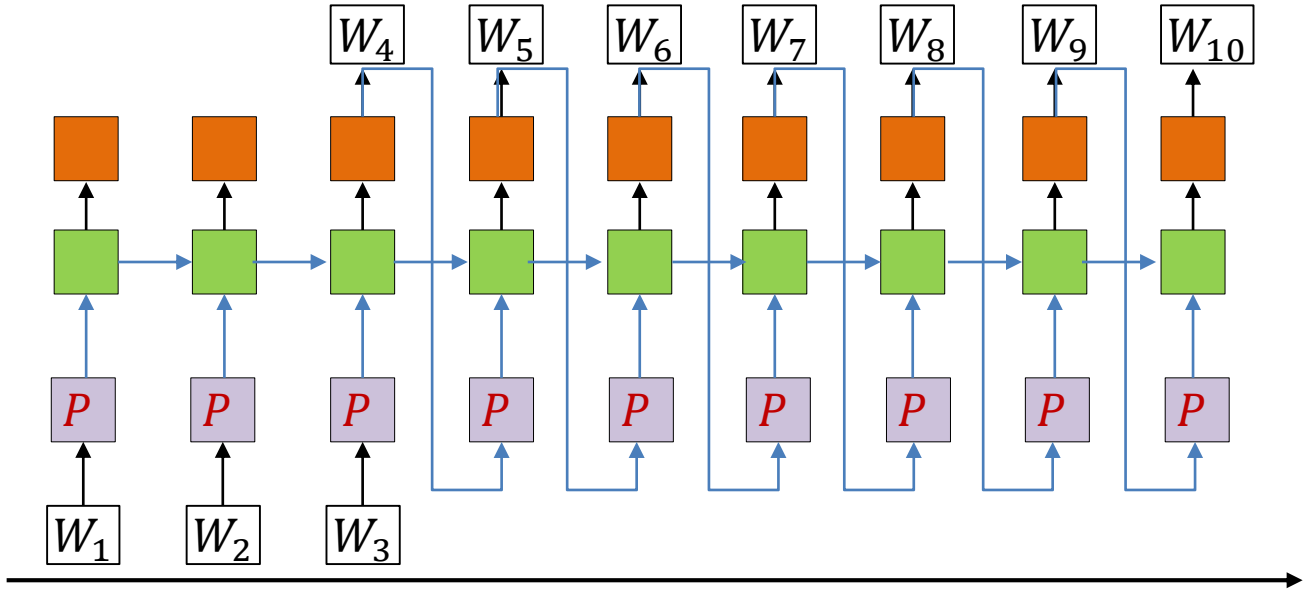
- On trained model : Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector
- Draw a word from the distribution
  - And set it as the next word in the series

Generating Language: Synthesis



- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution

Generating Language: Synthesis



- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution
- Continue this process until we terminate generation
  - In some cases, e.g. generating programs, there may be a natural termination

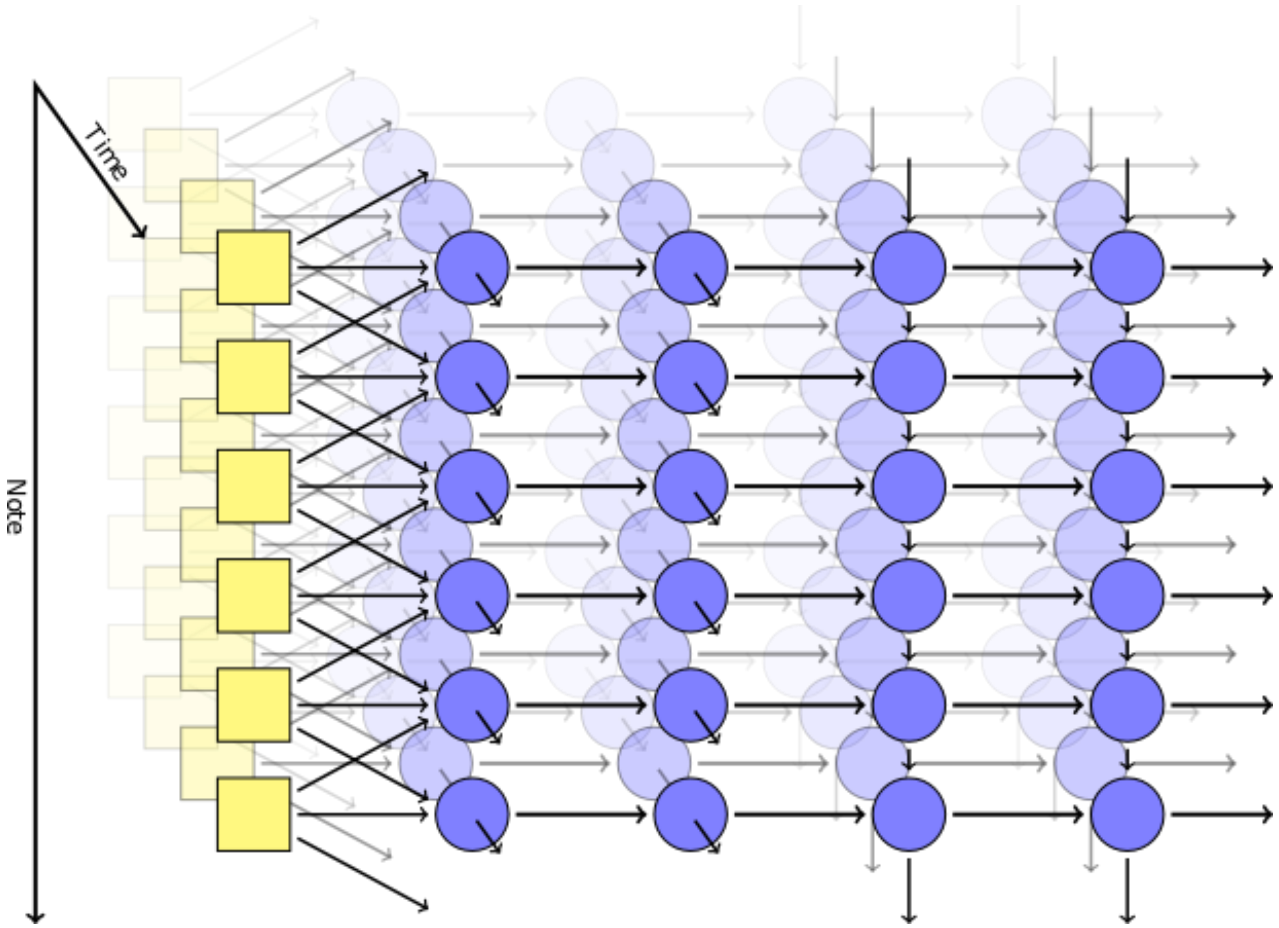
Which open source project?

```
/*
 * 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;
}
```

Trained on linux source code

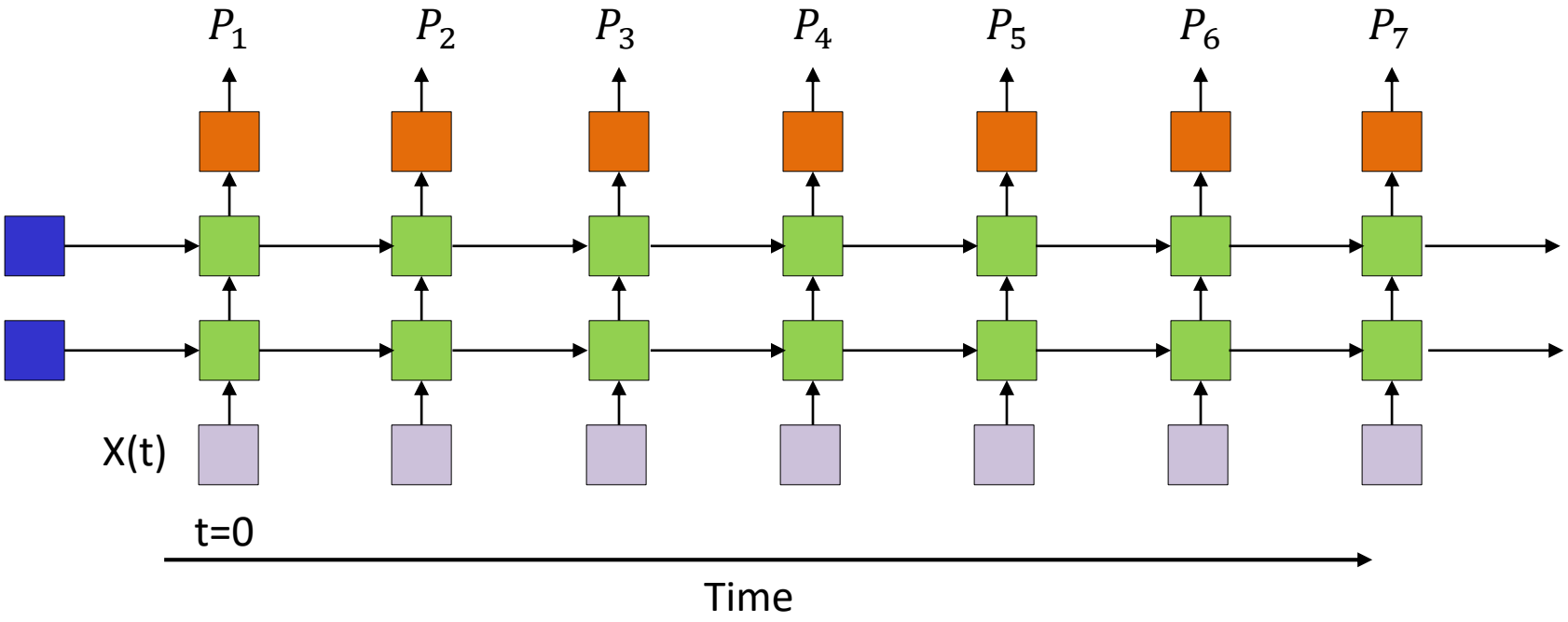
Actually uses a *character-level* model (predicts character sequences)

Composing music with RNN



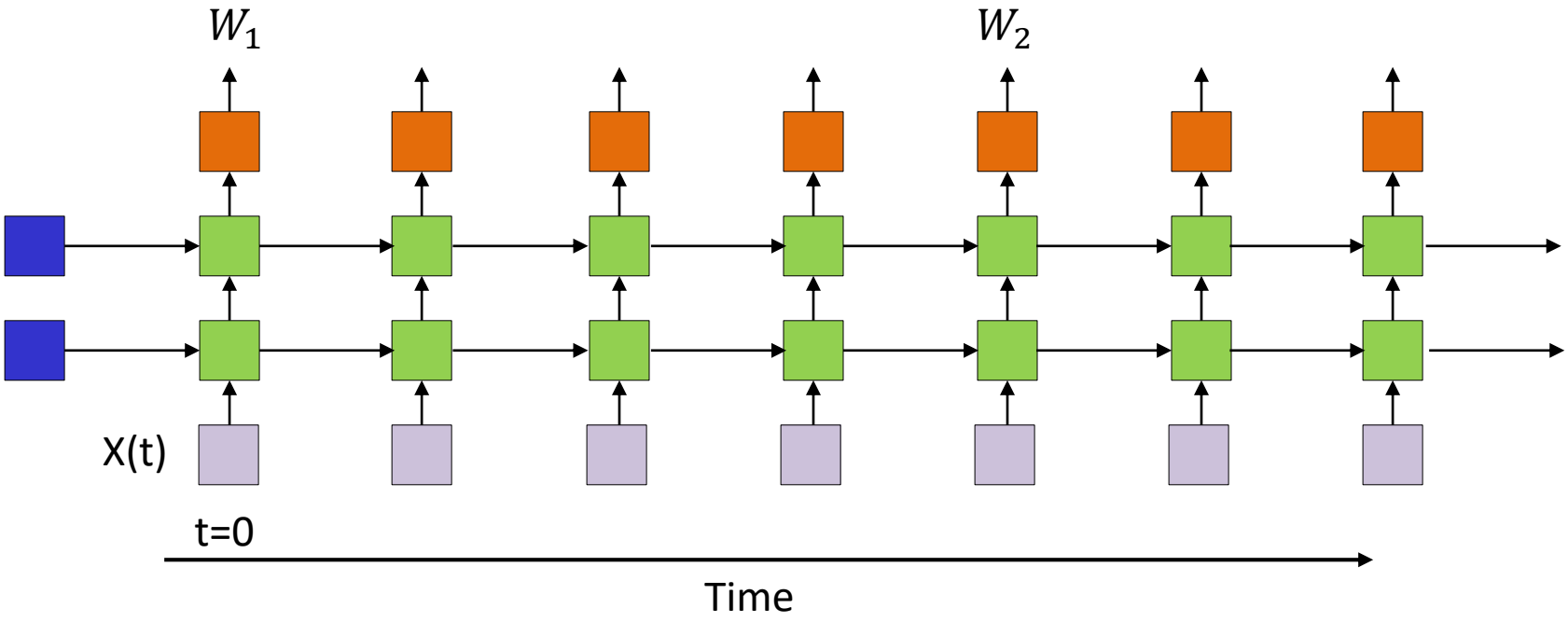
http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/

Speech recognition using Recurrent Nets



- Recurrent neural networks (with LSTMs) can be used to perform speech recognition
  - Input: Sequences of audio feature vectors
  - Output: Phonetic label of each vector

Speech recognition using Recurrent Nets



- Alternative: Directly output phoneme, character or word sequence
- Challenge: How to define the loss function to optimize for training
  - Future lecture
  - Also homework

CNN-LSTM-DNN for speech recognition

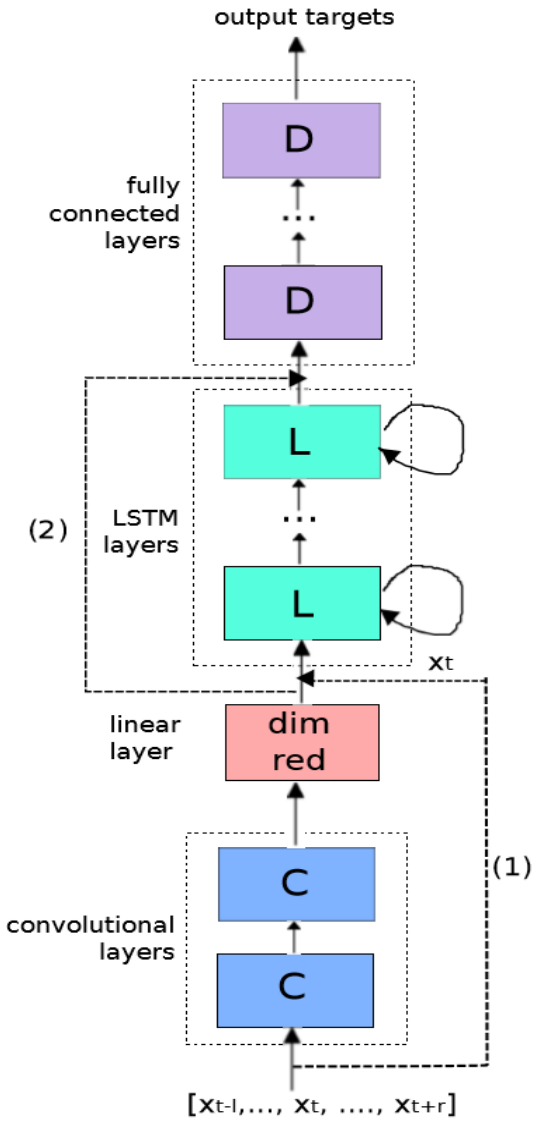
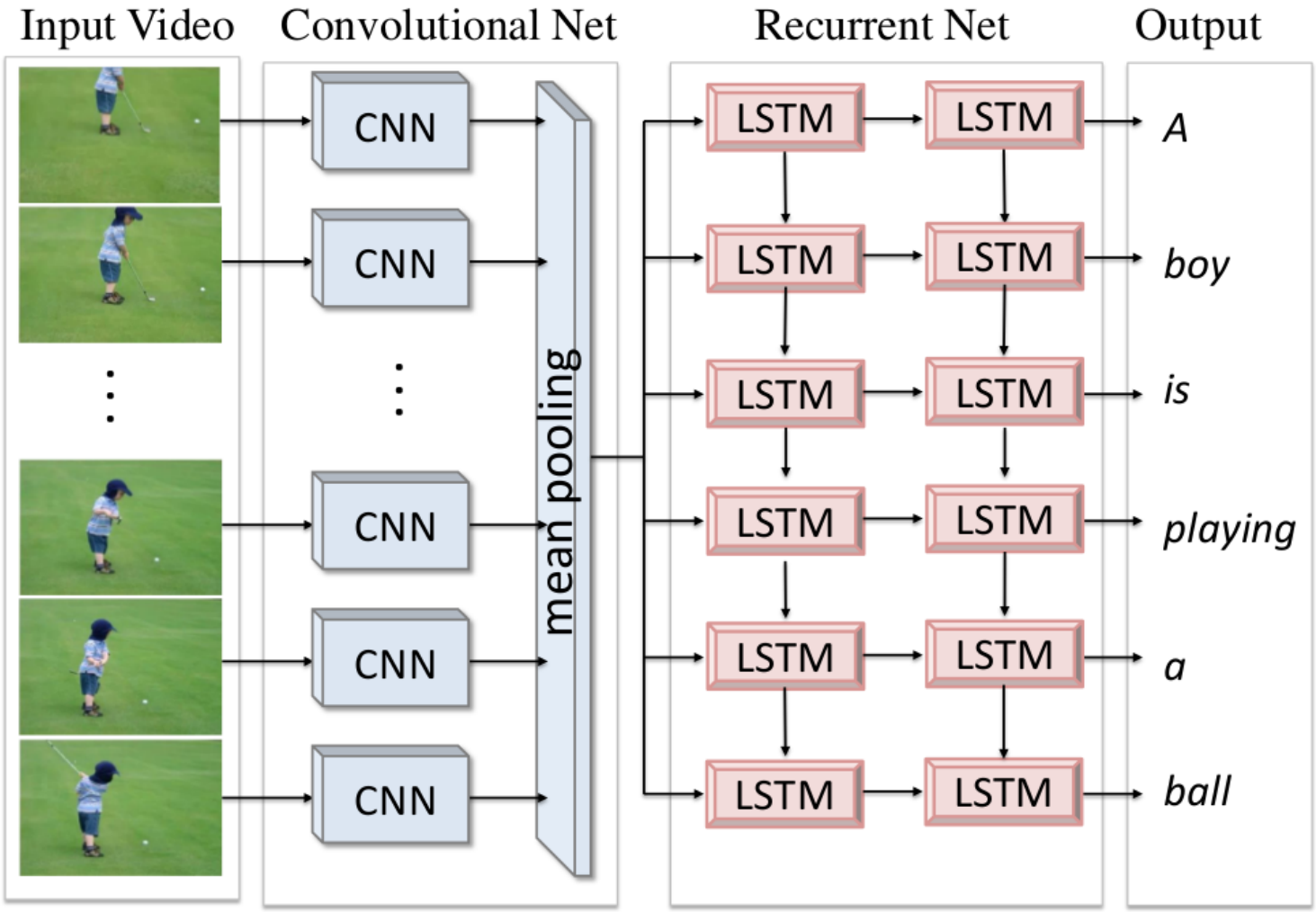


Fig. 1. CLDNN Architecture

Ensembles of RNN/LSTM, DNN, & Conv Nets (CNN) :

T. Sainath, O. Vinyals, A. Senior, H. Sak. "Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks," ICASSP 2015.

Translating Videos to Natural Language Using Deep Recurrent Neural Networks



Translating Videos to Natural Language Using Deep Recurrent Neural Networks
Subhashini Venugopalan, Huijun Xu, Jeff Donahue, Marcus Rohrbach, Raymond Mooney, Kate Saenko
North American Chapter of the Association for Computational Linguistics, Denver, Colorado, June 2015.

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

Summary

- Recurrent neural networks are more powerful than MLPs
  - Can use causal (one-direction) or non-causal (bidirectional) context to make predictions
  - Potentially Turing complete
- LSTM structures are more powerful than vanilla RNNs
  - Can “hold” memory for arbitrary durations
- Many applications
  - Language modelling
    - And generation
  - Machine translation
  - Speech recognition
  - Time-series prediction
  - Stock prediction
  - Many others..

Not explained

- Can be combined with CNNs
  - Lower-layer CNNs to extract features for RNN
- Can be used in tracking
  - Incremental prediction