Recurrent Neural Networks

M. Soleymani
Sharif University of Technology
Spring 2023

Most slides have been adopted from Fei Fei Li and colleagues lectures, cs231n, Stanford, 2022
and some slides from Bhiksha Raj, 11-785, CMU, 2019.

Sequences in the wild

Source: Ava Amini & Alex Amini, Deep learning course, MIT

Modelling Series

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

- Examples: ..

Example I

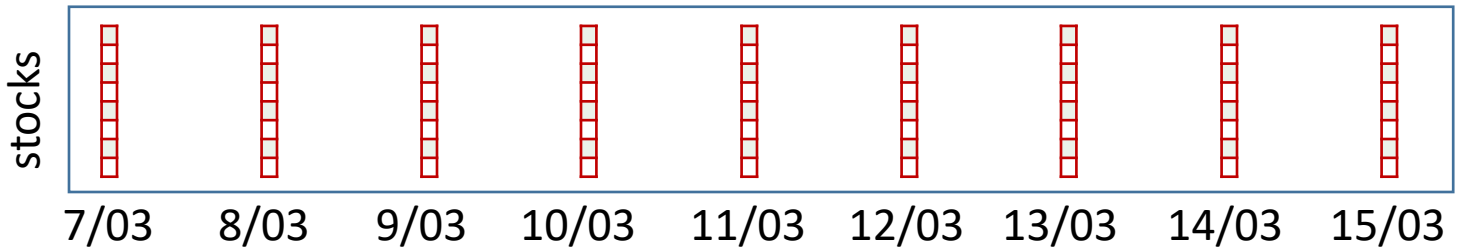
"Football" or "basketball"?

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

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

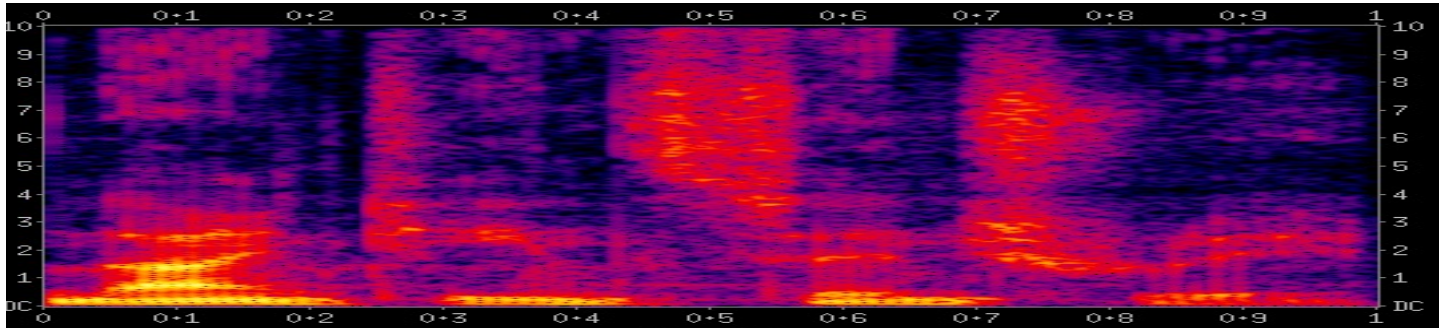
Example II

To invest or not to invest?



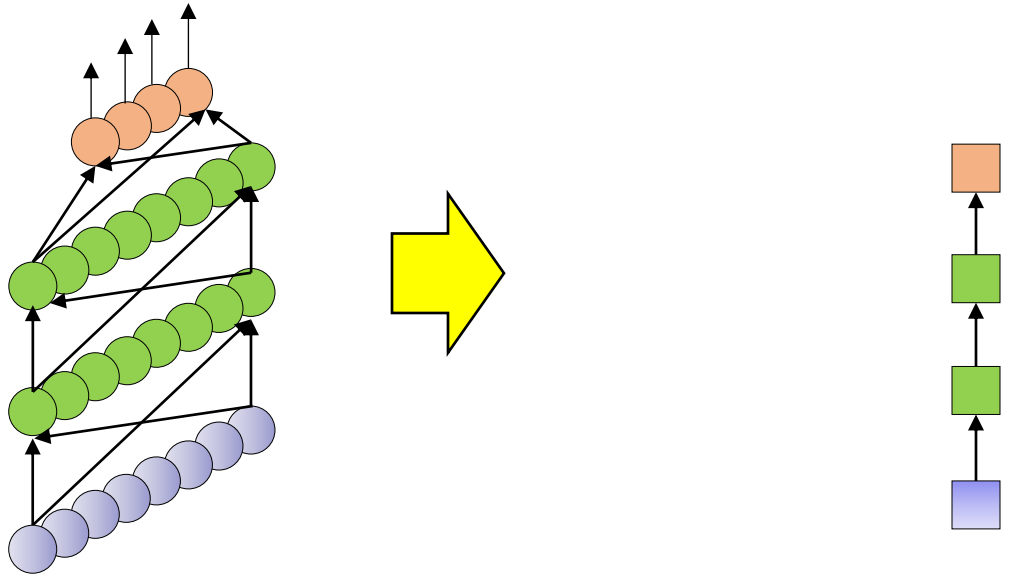
- Stock market
  - Must consider the series of stock values in the past several days to decide if it is wise to invest today
    - Ideally consider *all* of history
- Note: Inputs are vectors. Output may be scalar or vector
  - Should I invest, vs. should I invest in X

Example III



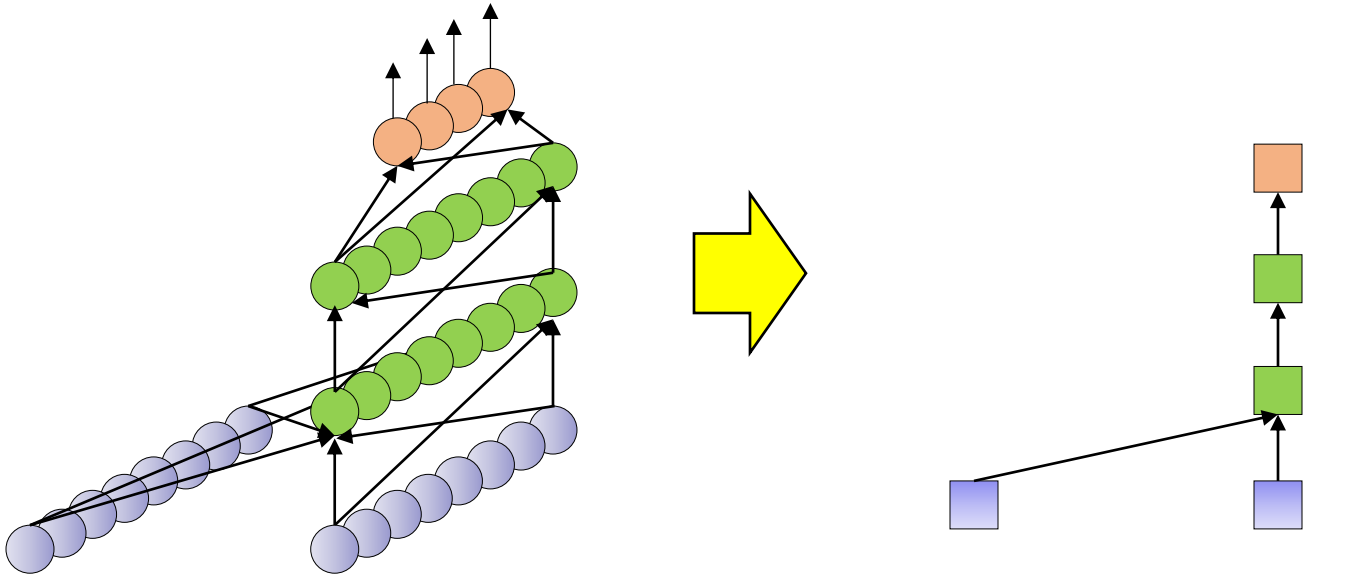
- Speech Recognition
  - Analyze a series of spectral vectors, determine what was said

Representational shortcut



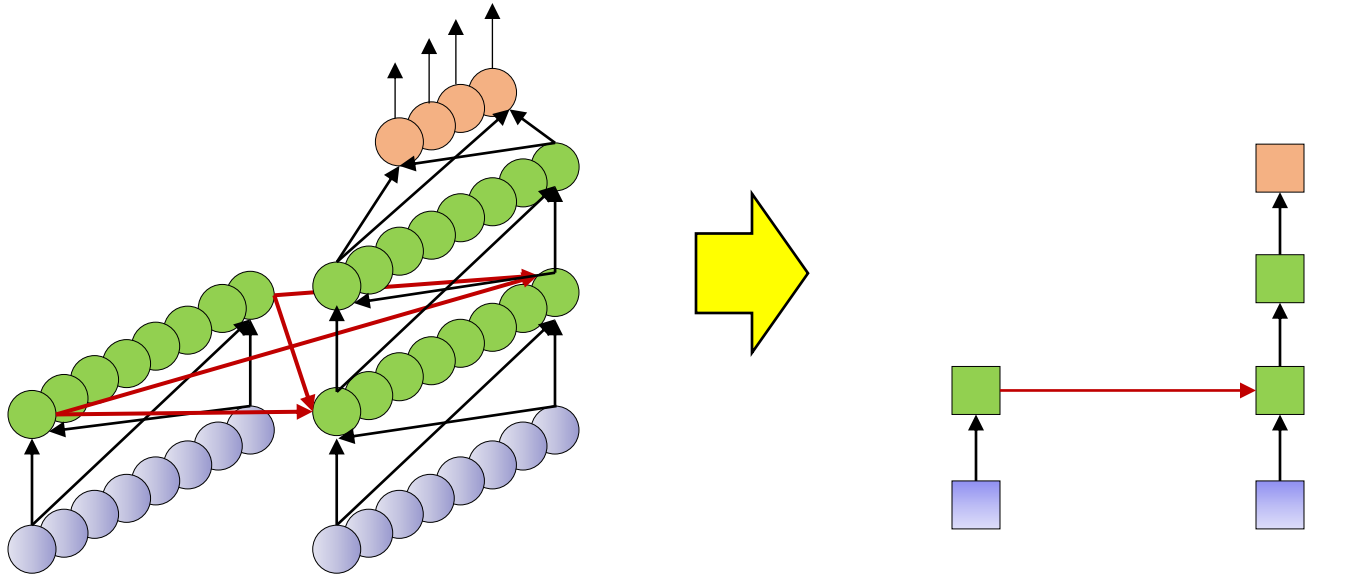
- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything by simple boxes
  - Each box actually represents an entire *layer with many units*

Representational shortcut



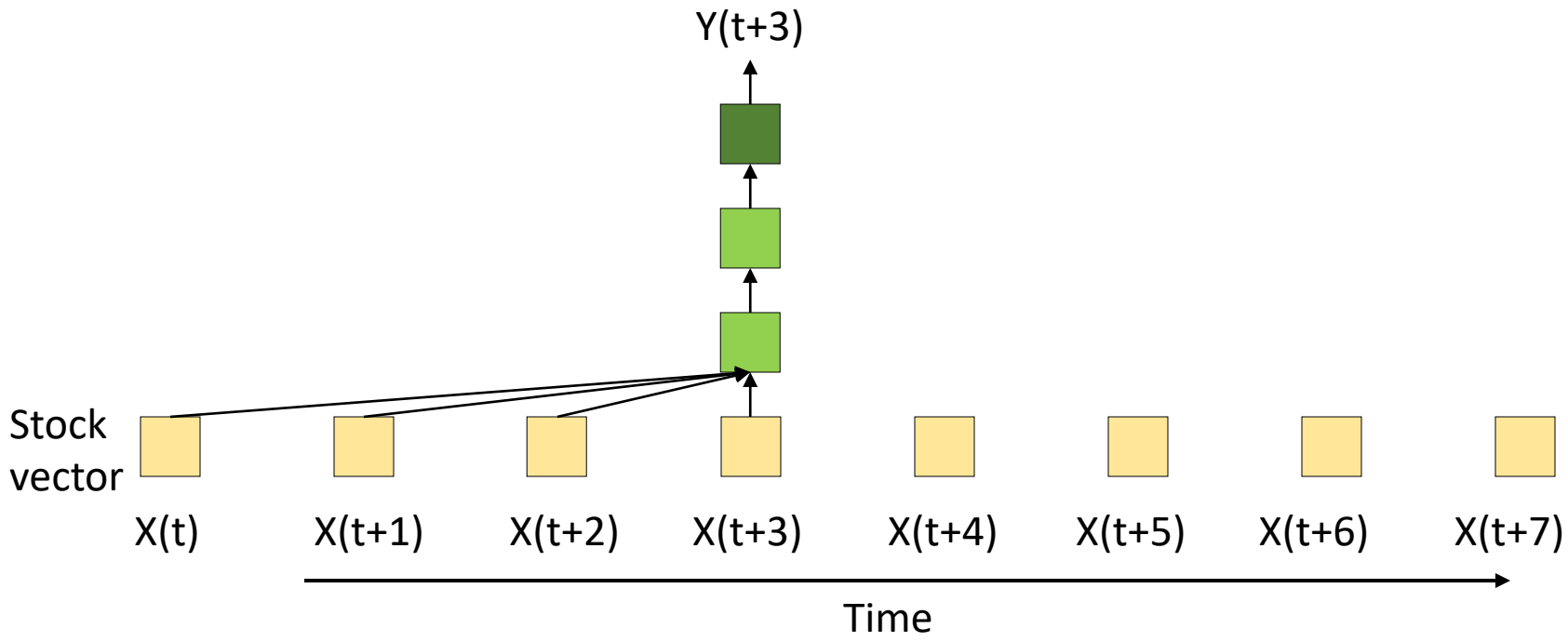
- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything by simple boxes
  - Each box actually represents an entire *layer with many units*

Representational shortcut



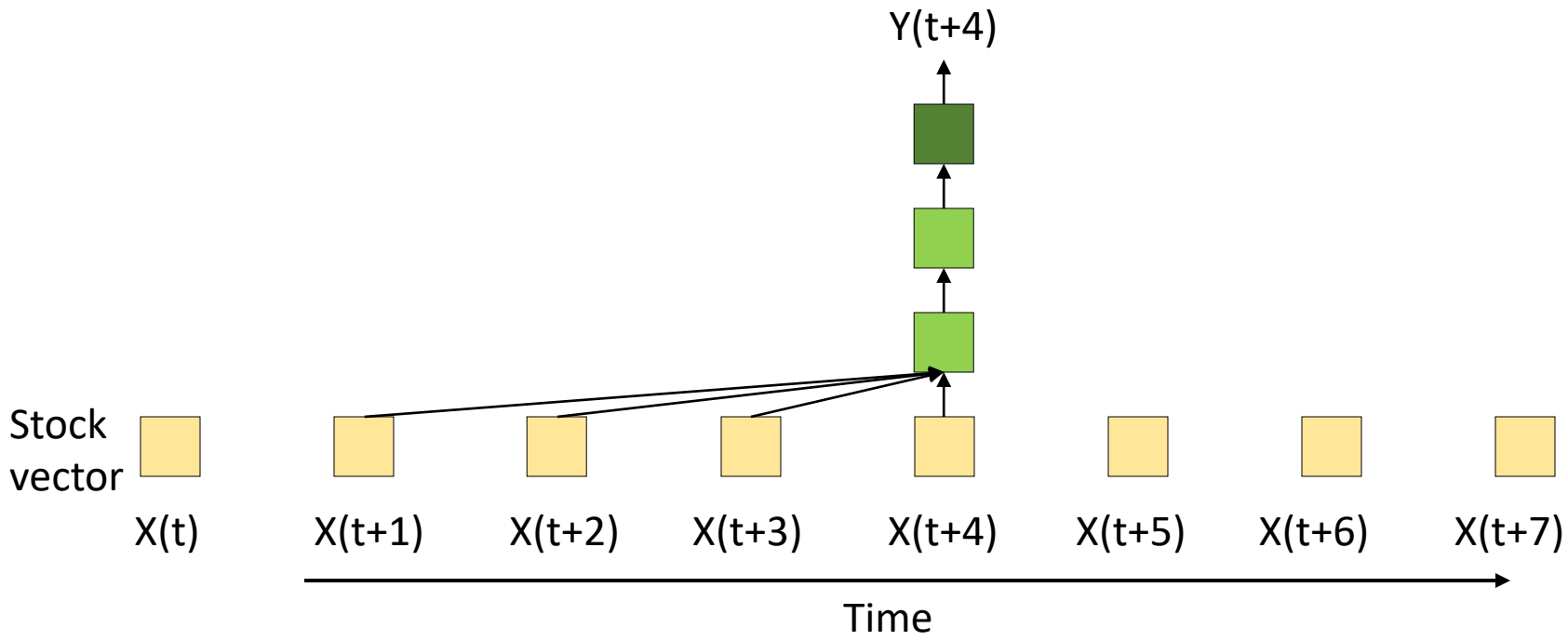
- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything by simple boxes
  - Each box actually represents an entire *layer with many units*

The stock predictor



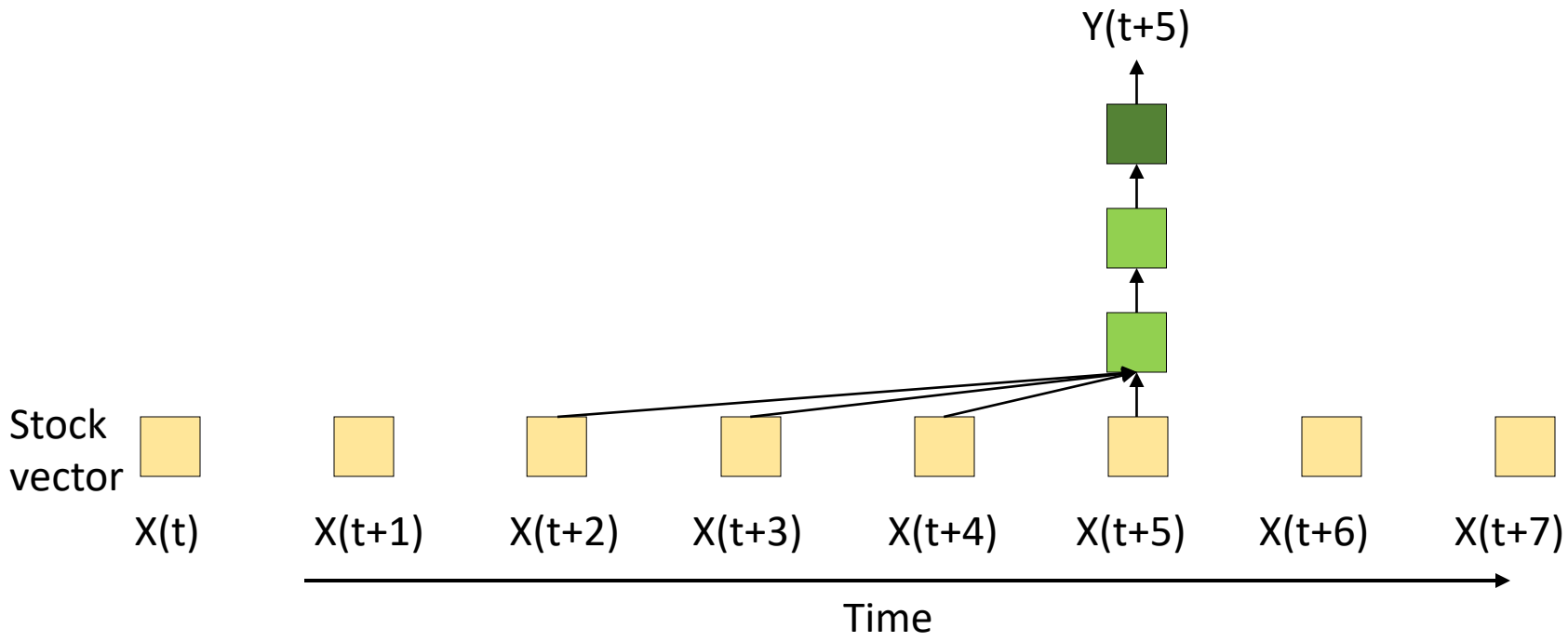
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay Neural Network (TDNN)*

The stock predictor



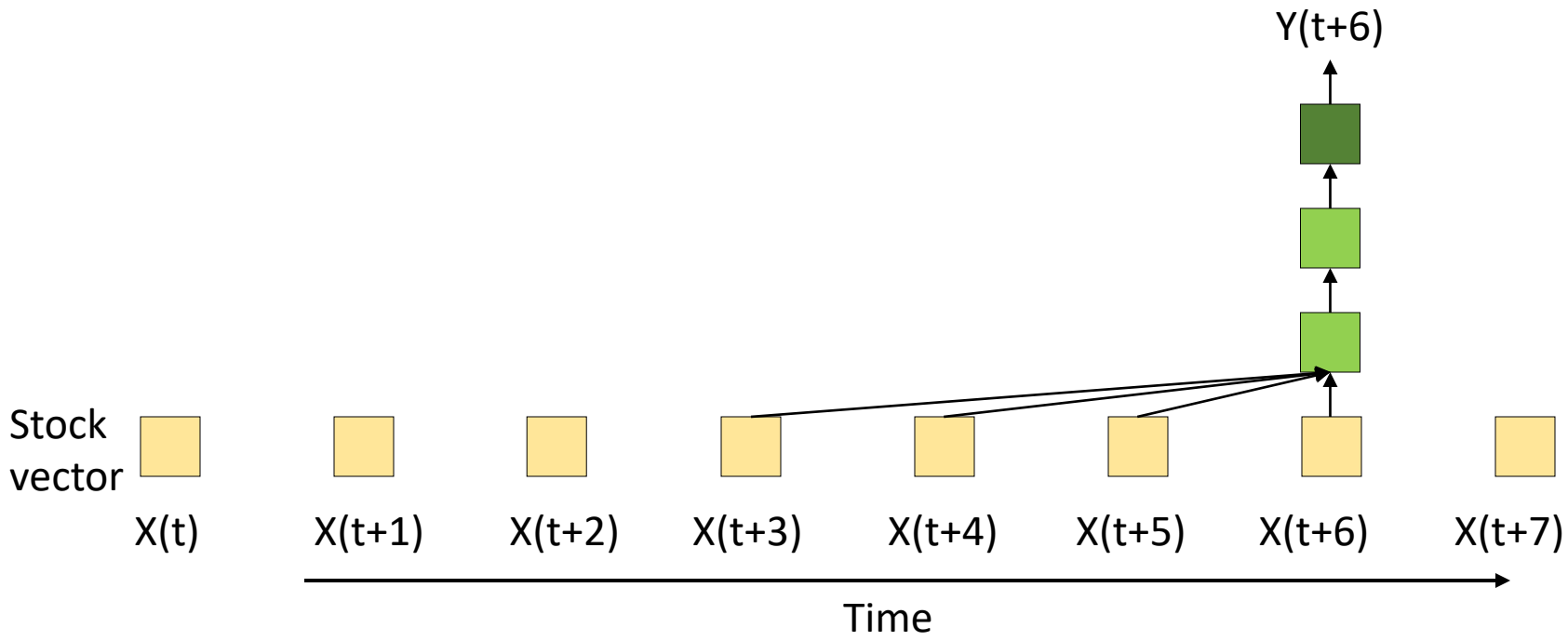
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay Neural Network (TDNN)*

The stock predictor



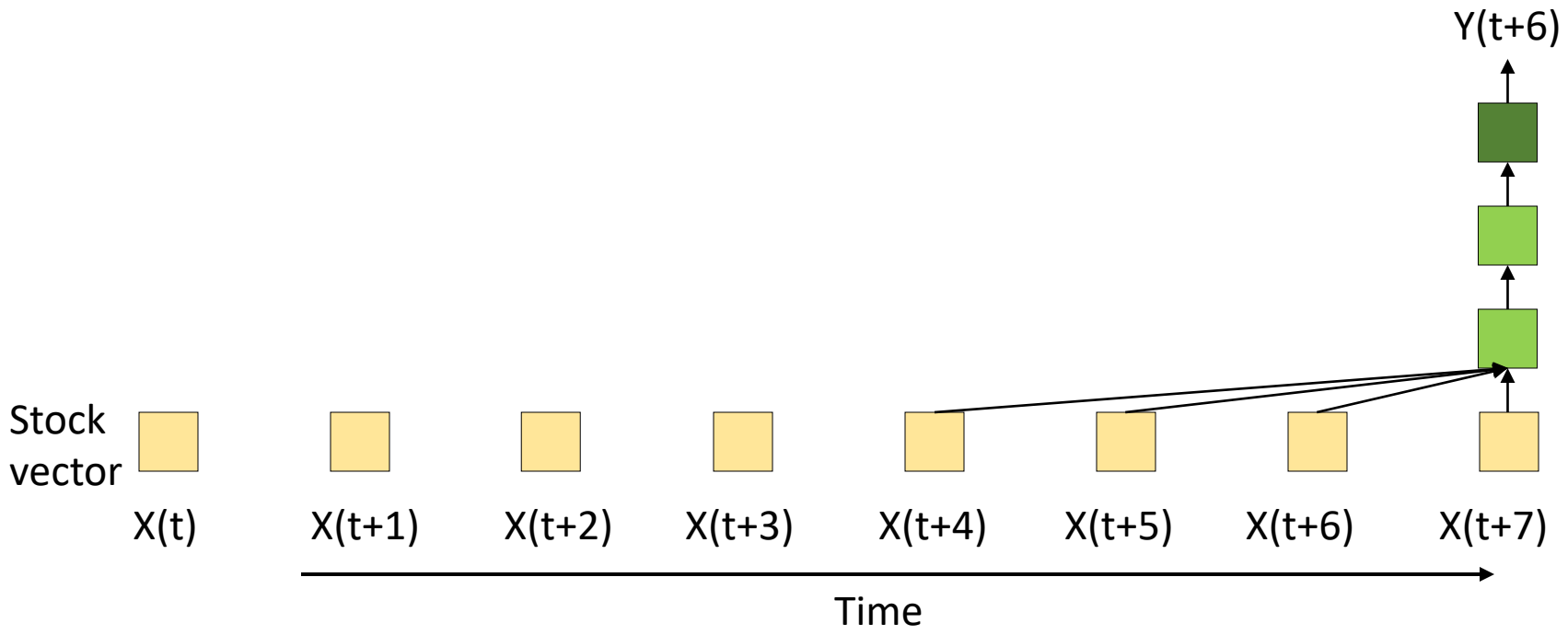
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay Neural Network (TDNN)*

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 (TDNN)*

The stock predictor



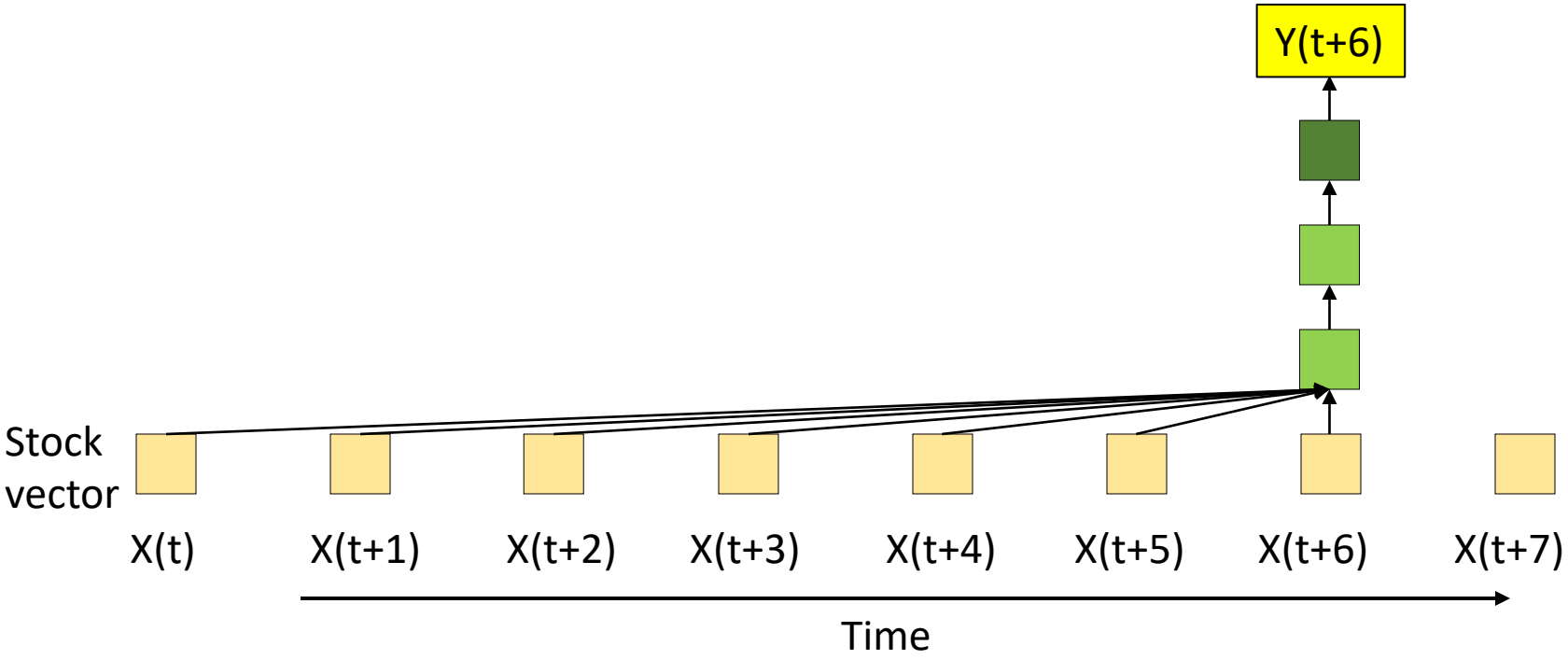
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay Neural Network (TDNN)*

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

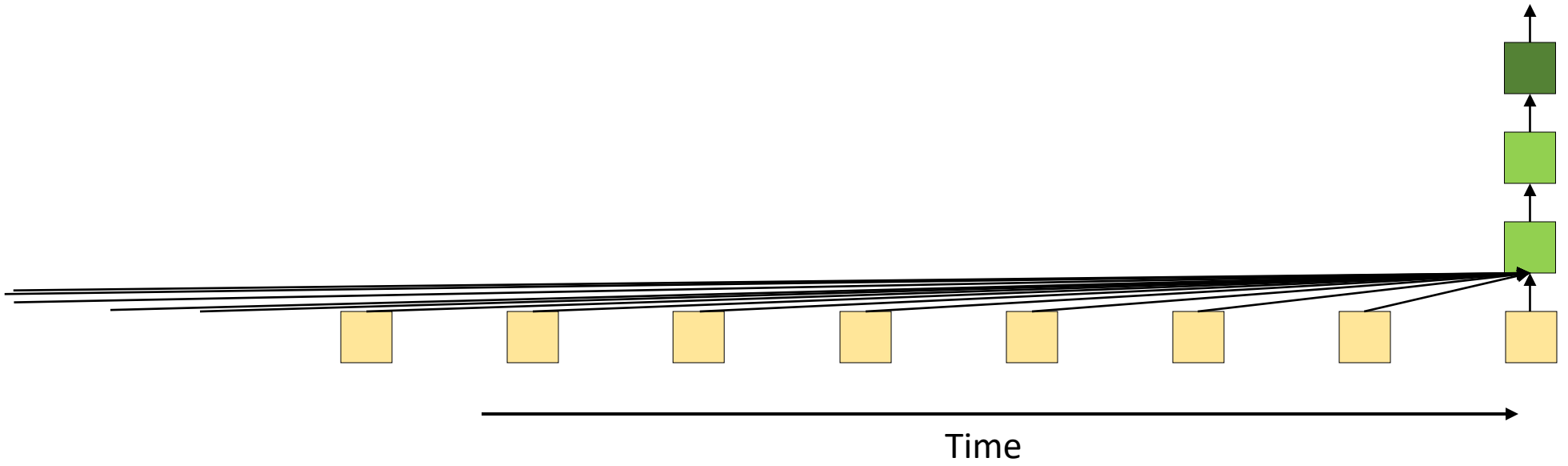


- Problem: Increasing the "history" makes the network more complex
  - No worries, we have the CPU and memory
    - Or do we?

Systems often have long-term dependencies

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

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

An alternate model for infinite response: state-space model

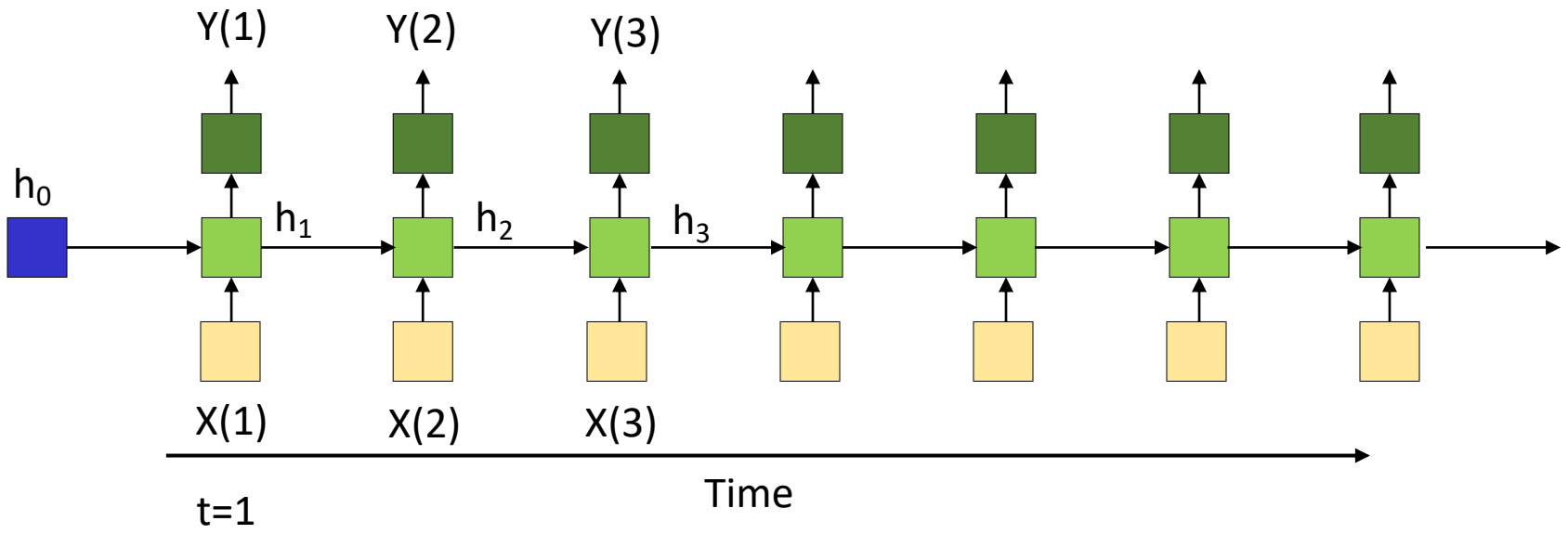
- State-space model:

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

$$y_t = g(h_t)$$

- h_t is the *state* of the network
  - Model directly embeds the memory in the state
- Need to define initial state h_0
- This is a *fully recurrent* neural network
  - Or simply a *recurrent neural network*
- *State* summarizes information about the entire past

The simple state-space model



- The state (green) at any time is determined by the input at that time, and the state at the previous time
- Also known as a recurrent neural net

Recurrent Neural Network

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

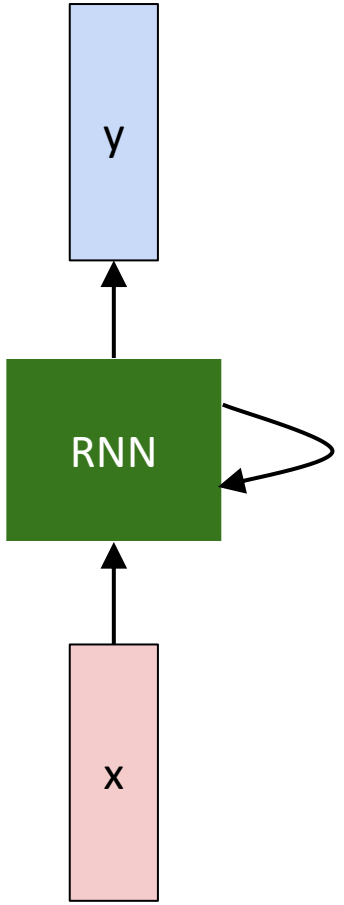
$$h_t = f_W(h_{t-1}, x_t)$$

new state

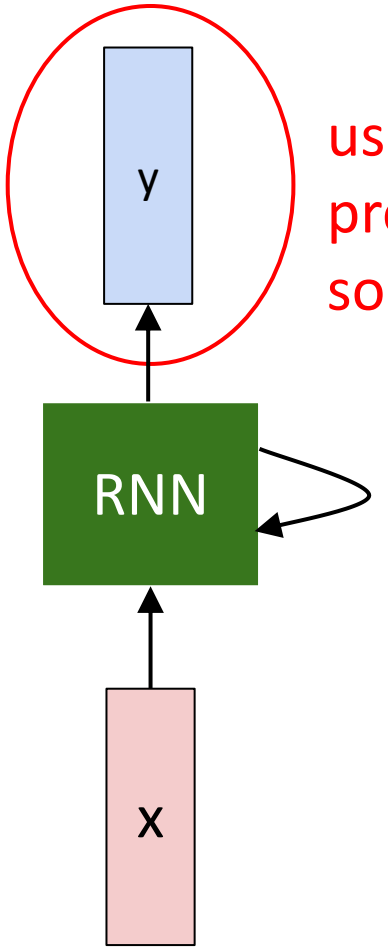
some function with parameters W

old state

input vector at some time step



Recurrent Neural Network

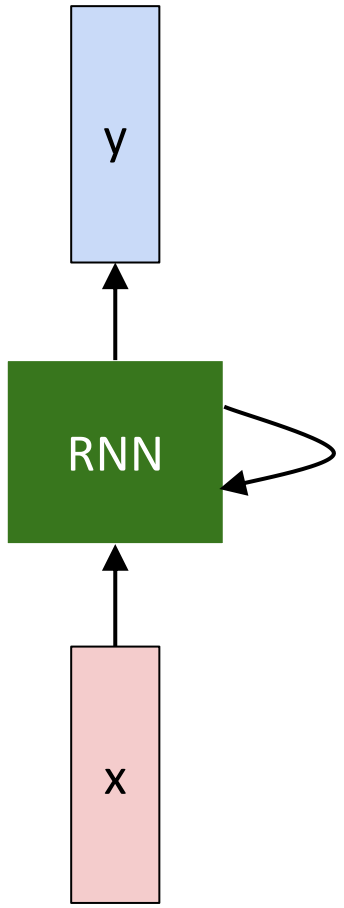


usually want to predict a vector at some time steps

Recurrent Neural Network

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

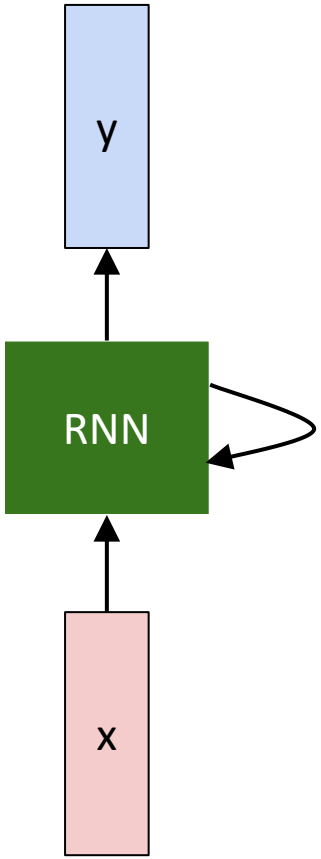
$$h_t = f_W(h_{t-1}, x_t)$$



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

(Vanilla) Recurrent Neural Network

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

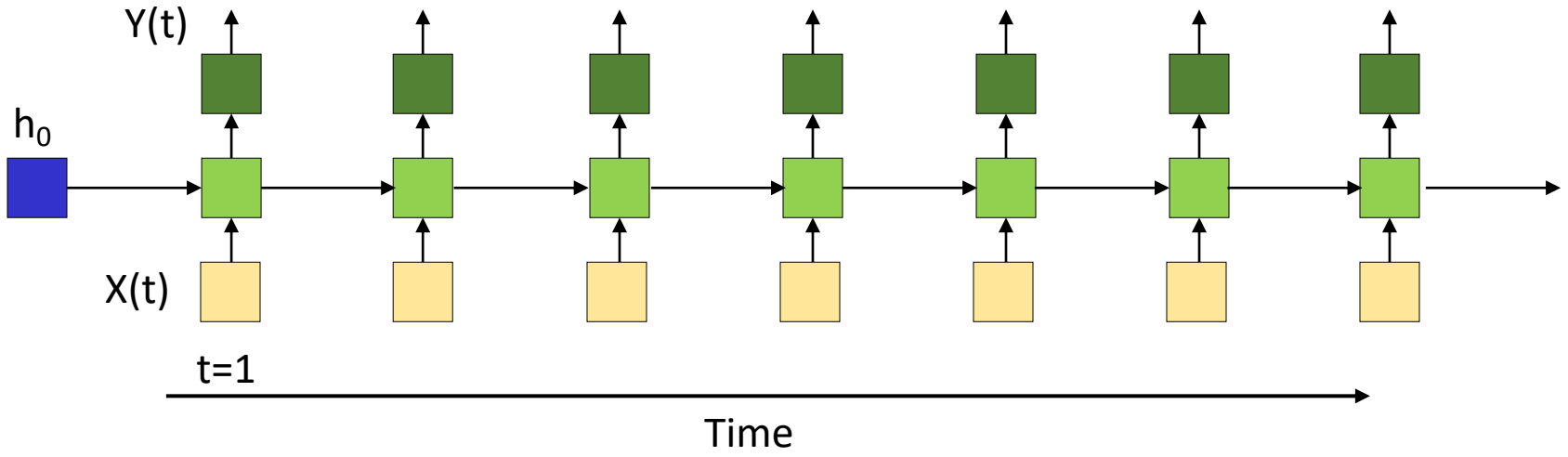


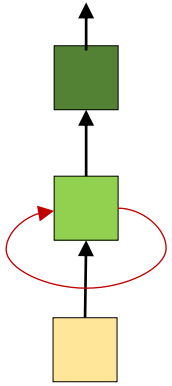
$$h_t = f_W(h_{t-1}, x_t)$$

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

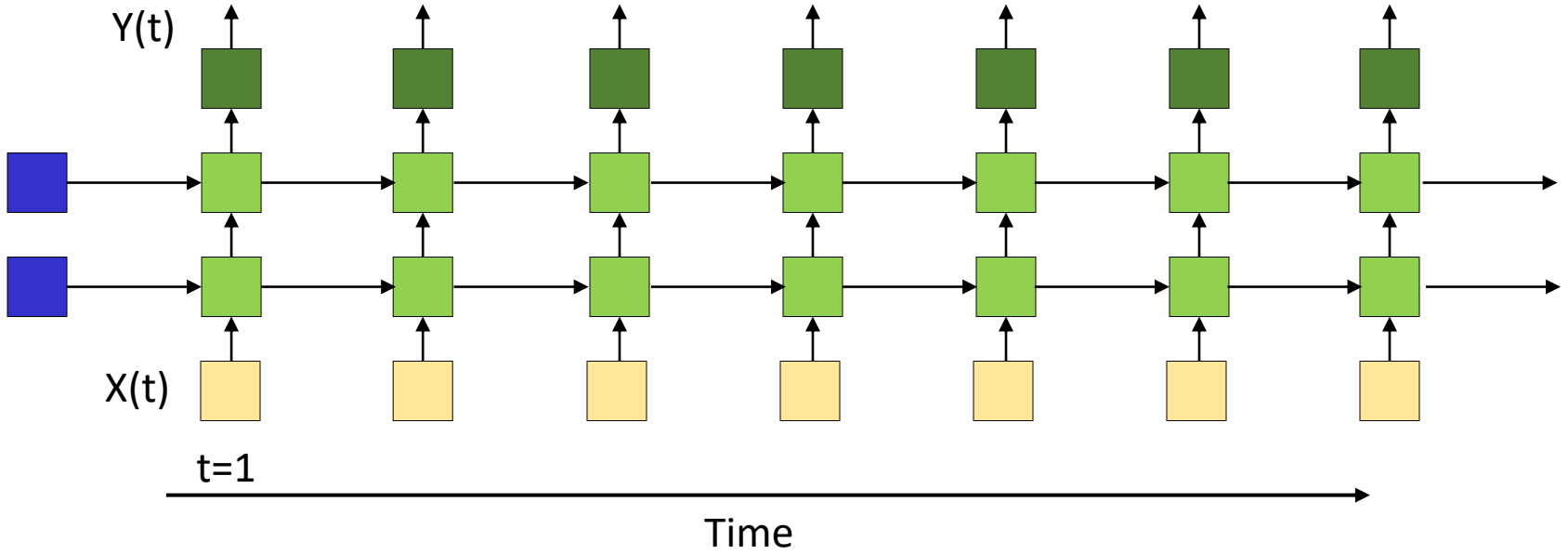
$$y_t = W_{hy}h_t$$

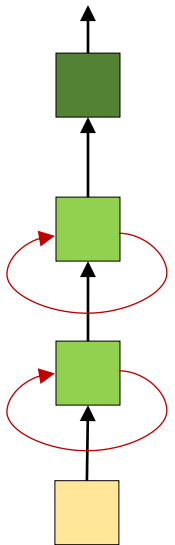
Single hidden layer RNN



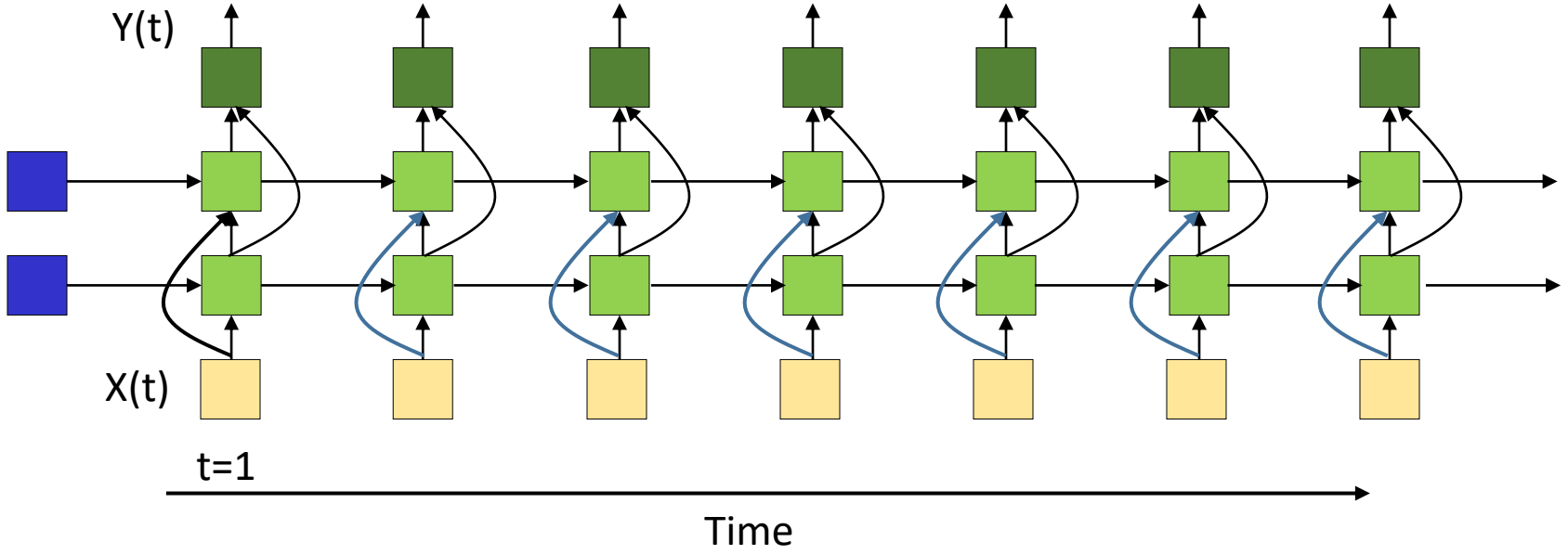


Multiple recurrent layer RNN

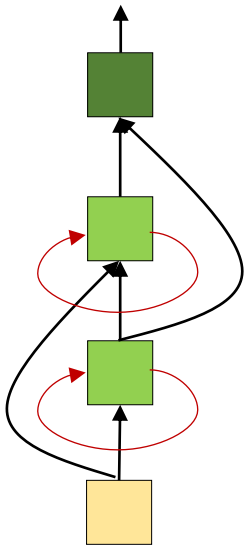




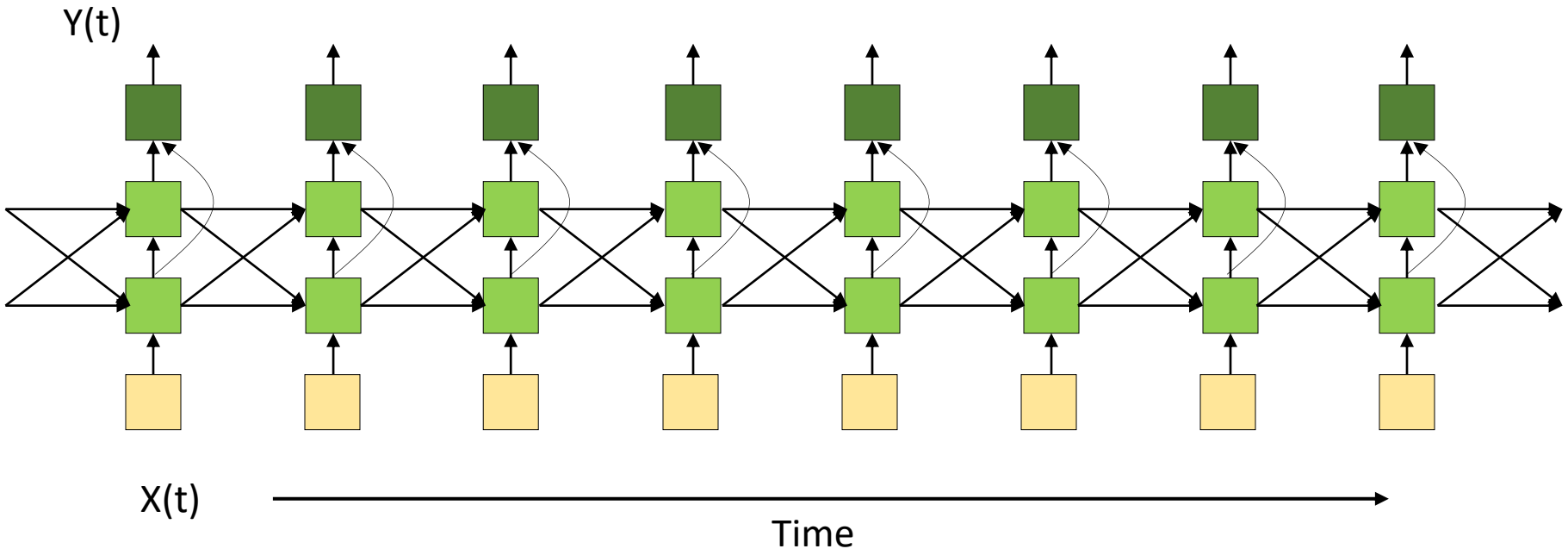
Multiple recurrent layer RNN



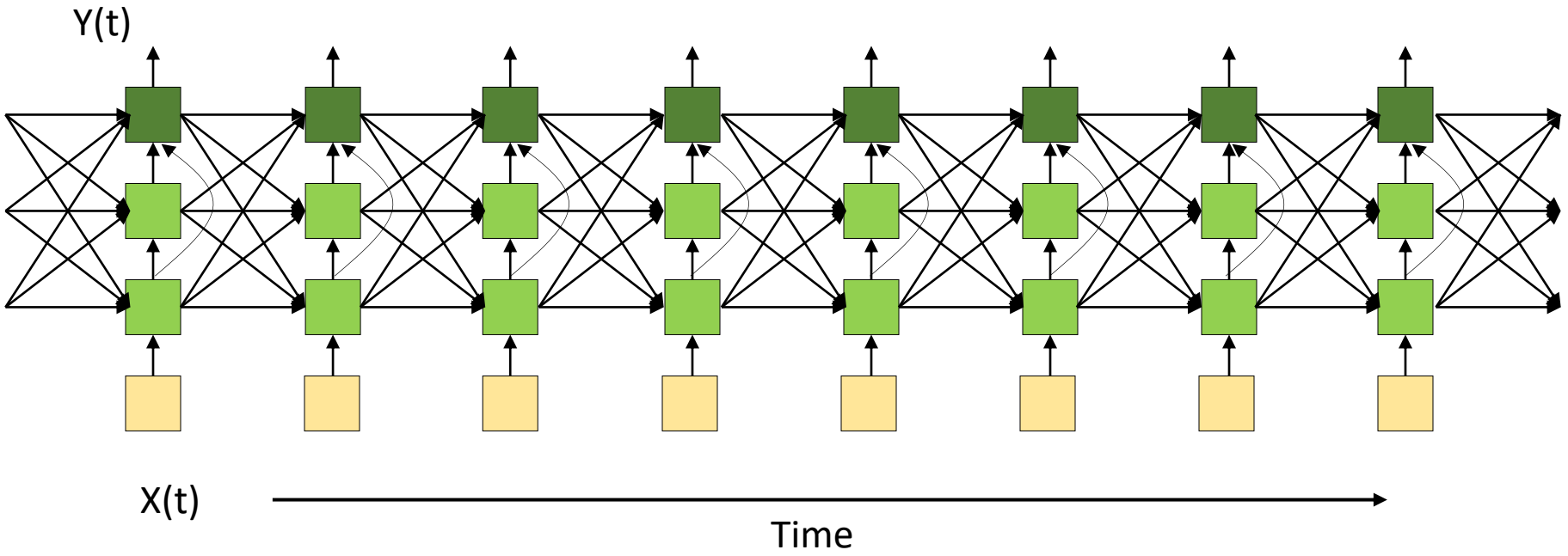
- We can also have skips..



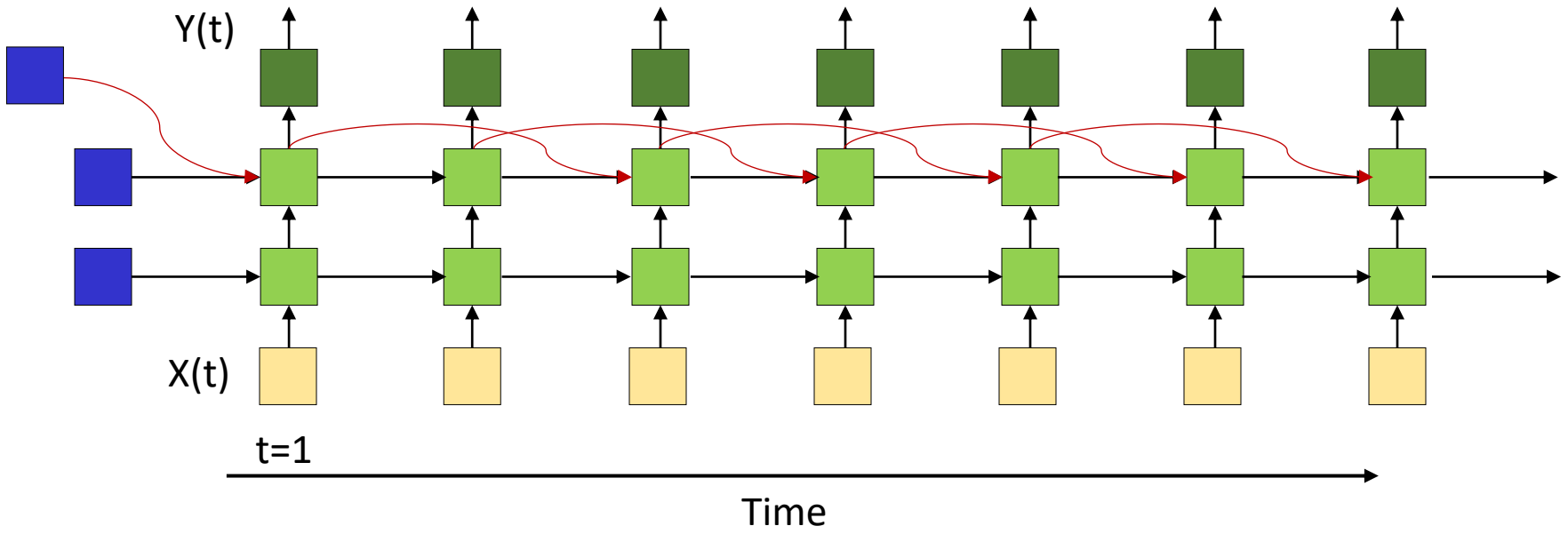
A more complex state



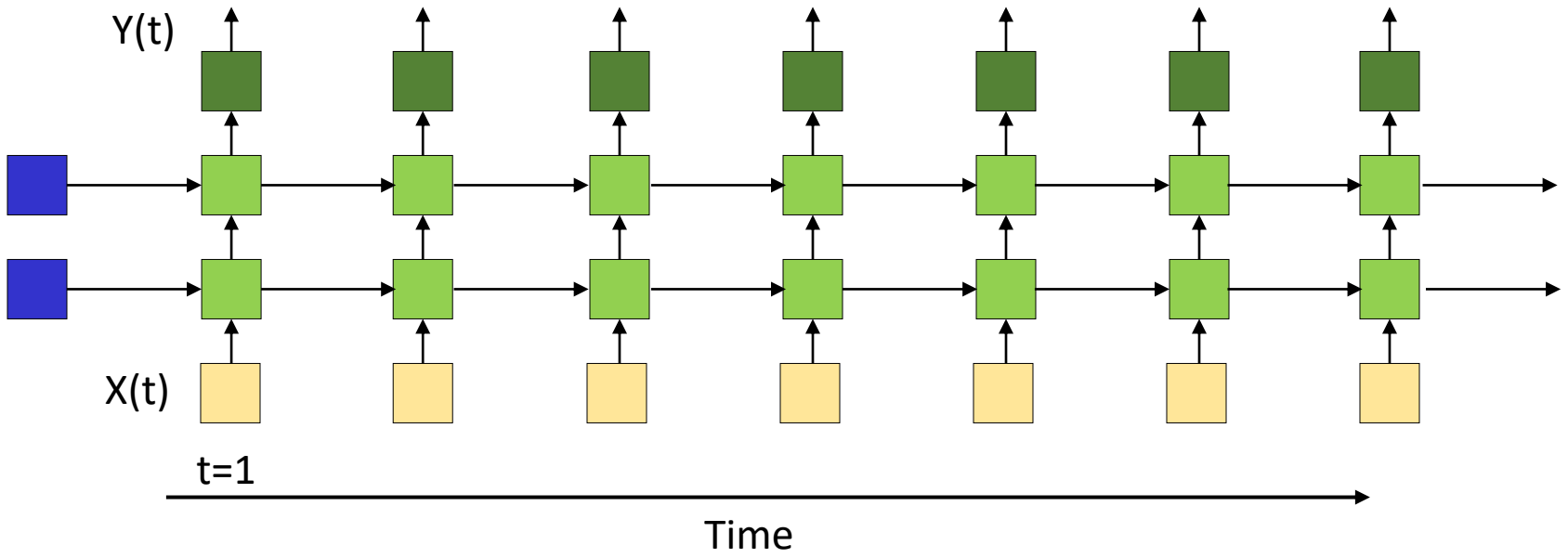
Or the network may be even more complicated



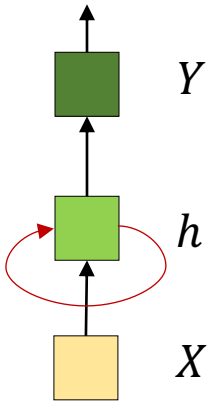
Generalization with other recurrences



The simplest structures are most popular



Multi-layer equations



$$h_i(0) = part\ of\ network\ parameters$$

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

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

- 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

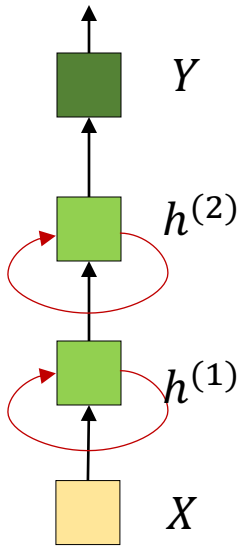
$w_{ji}^{(2)}$: weights from $h^{(2)}$ to Y

$w_{ji}^{(22)}$: weights from $h^{(2)}$ to $h^{(2)}$

$w_{ji}^{(1)}$: weights from $h^{(1)}$ to $h^{(2)}$

$w_{ji}^{(11)}$: weights from $h^{(1)}$ to $h^{(1)}$

$w_{ji}^{(0)}$: weights from input to $h^{(1)}$



Note superscript in indexing, which indicates layer of network from which inputs are obtained

$$h_i^{(1)}(0) = part\ of\ network\ parameters$$

$$h_i^{(2)}(0) = part\ of\ network\ parameters$$

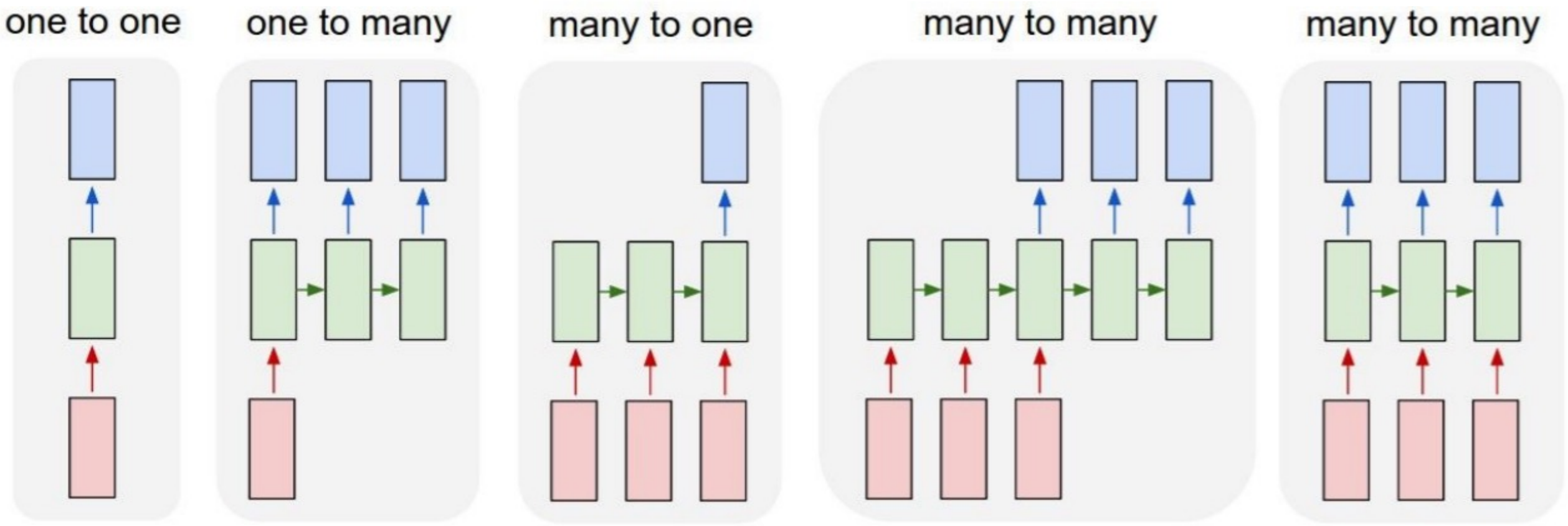
$$h_i^{(1)}(t) = f_1\left(\sum_j w_{ji}^{(0)} X_j(t) + \sum_j w_{ji}^{(11)} h_i^{(1)}(t-1) + b_i^{(1)}\right)$$

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

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

- 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

Recurrent Neural Networks: Process Sequences



"Vanilla" NN

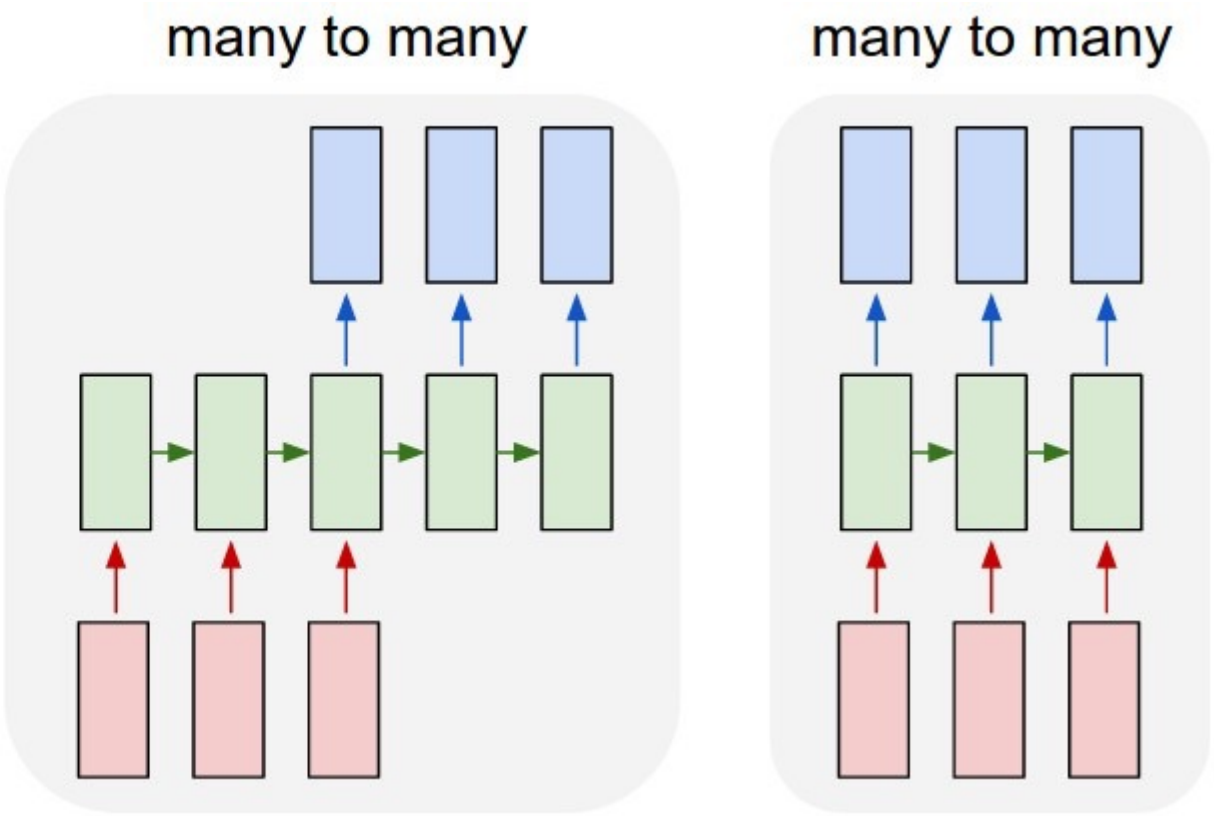
e.g. Image Captioning
image -> seq of words

e.g. Sentiment Classification
seq of words -> sentiment

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

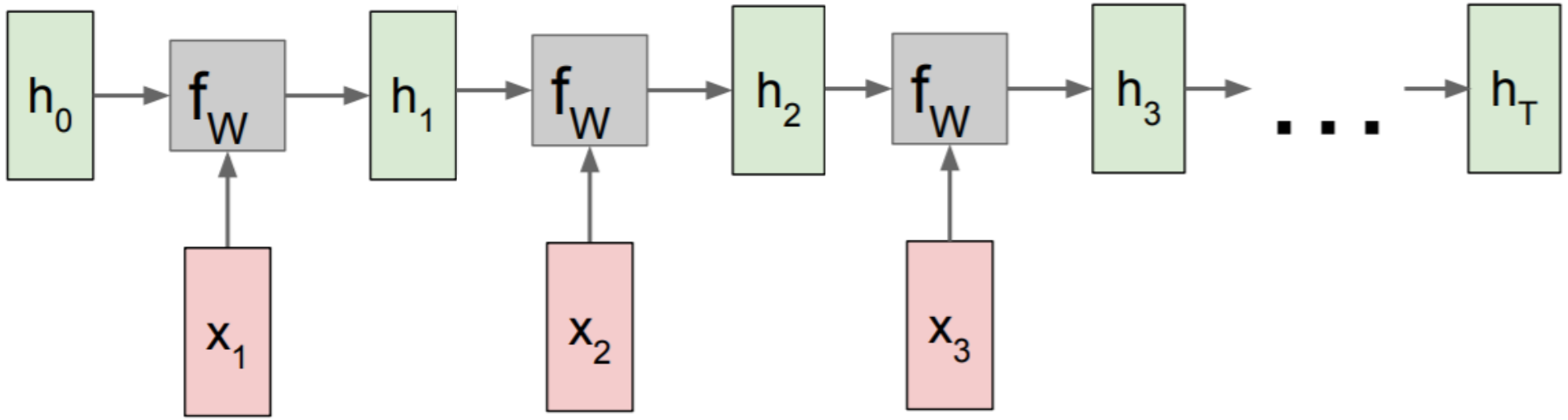
e.g. Video classification on frame level

Variants



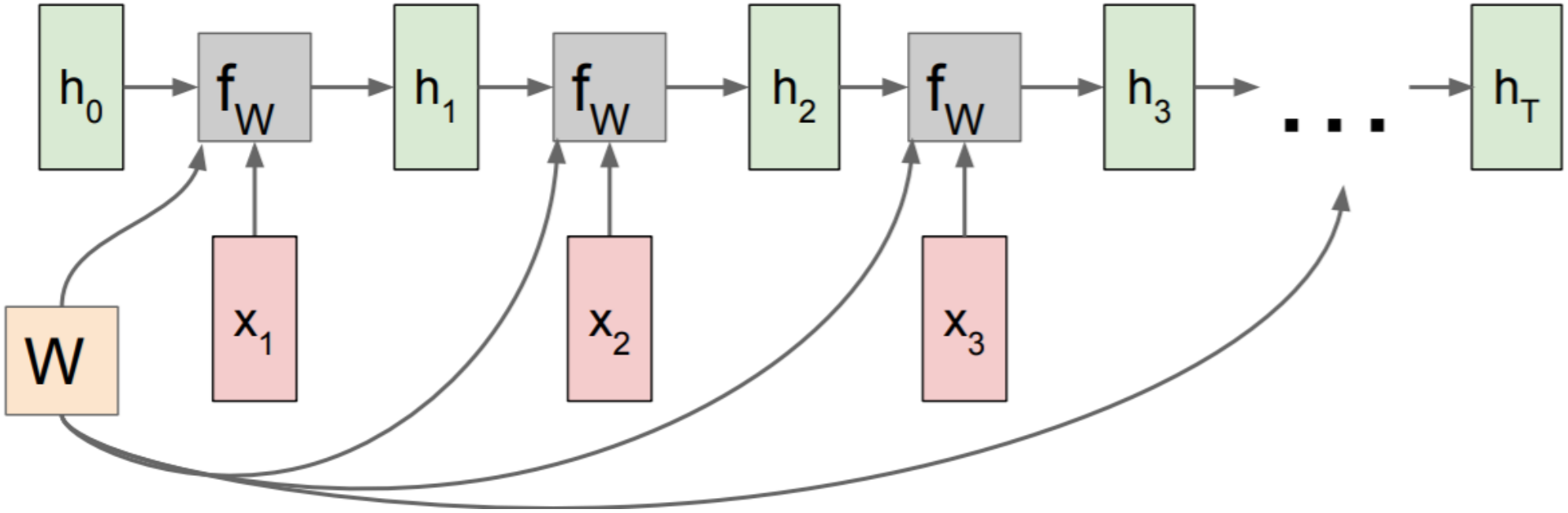
- 1: *Delayed* sequence to sequence
- 2: Sequence to sequence, e.g. stock problem, label prediction
- Etc…

RNN: Computational Graph

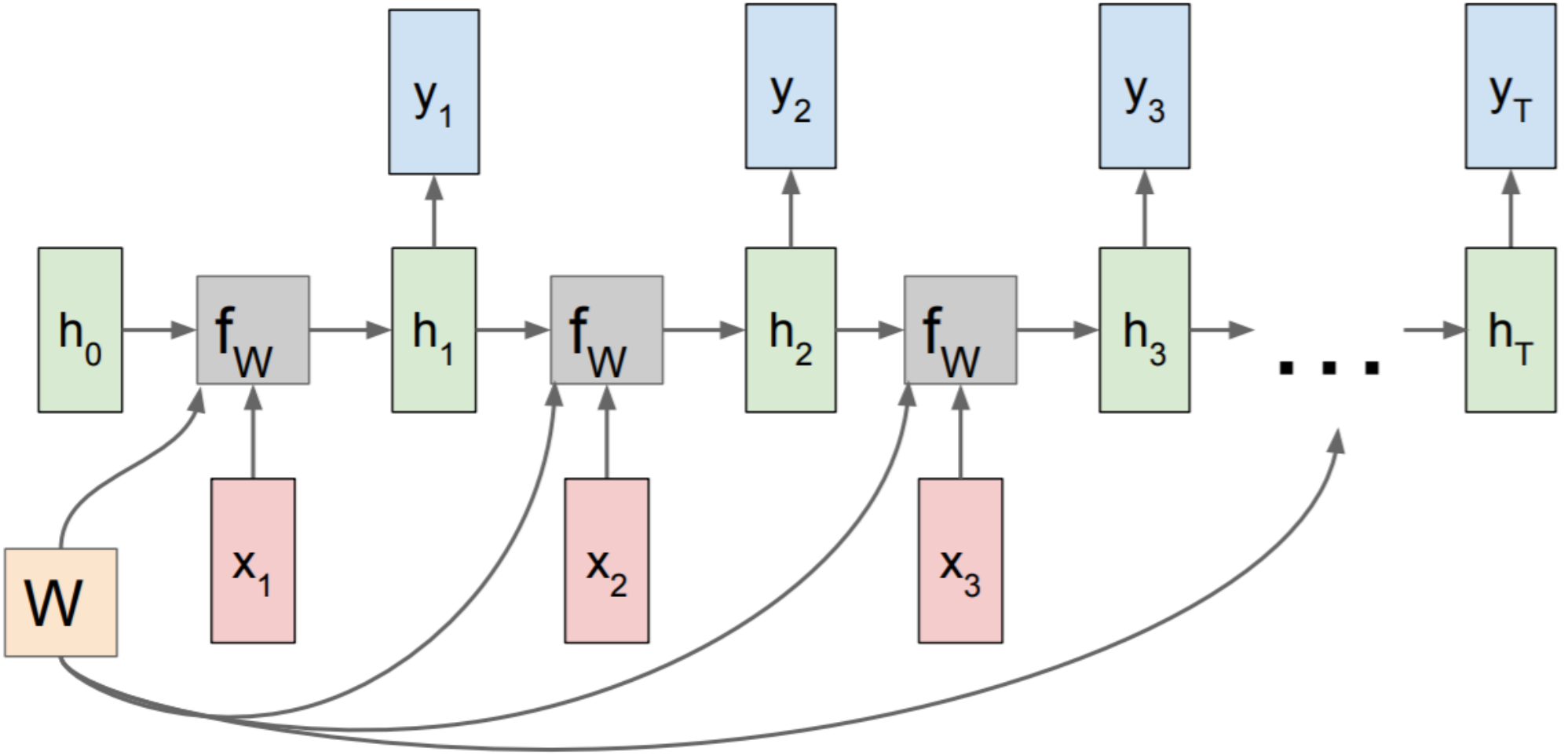


RNN: Computational Graph

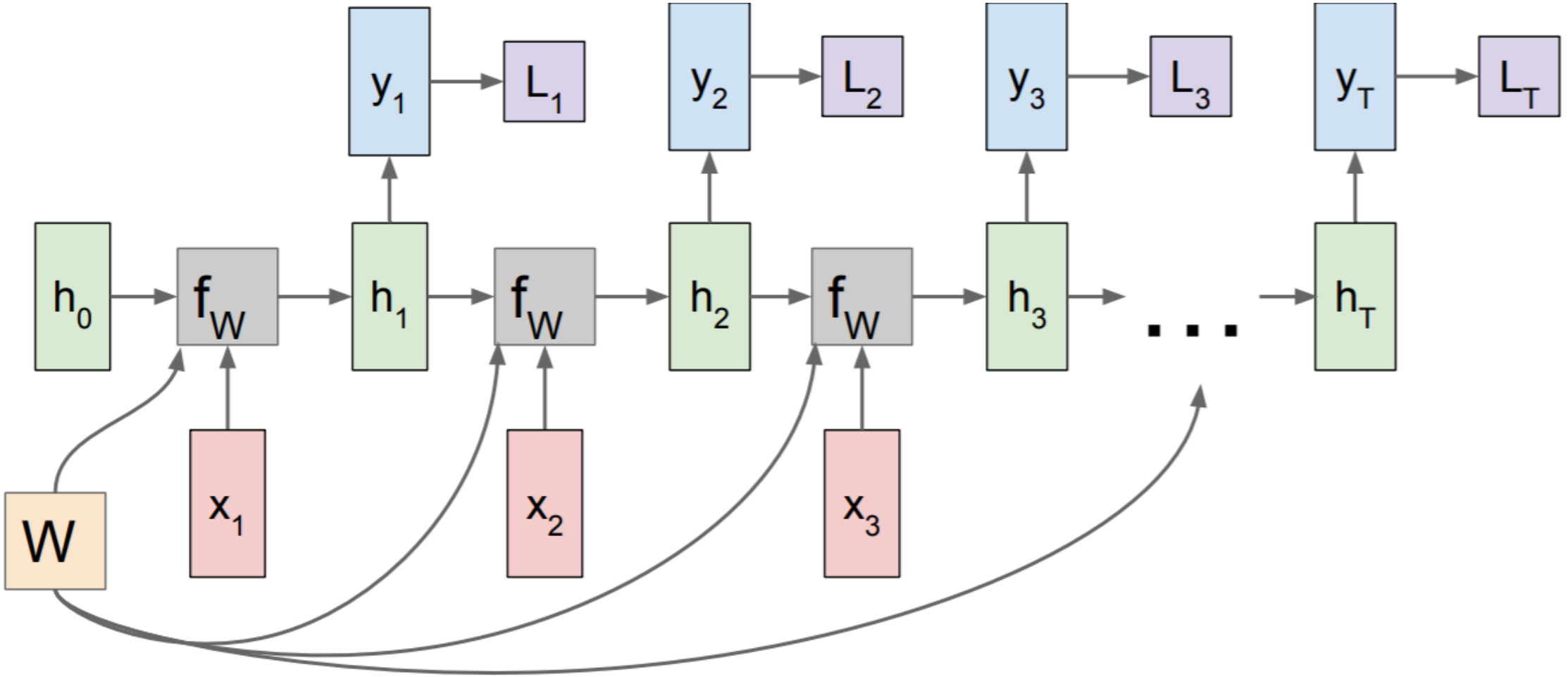
Re-use the same weight matrix at every time-step



RNN: Computational Graph: Many to Many

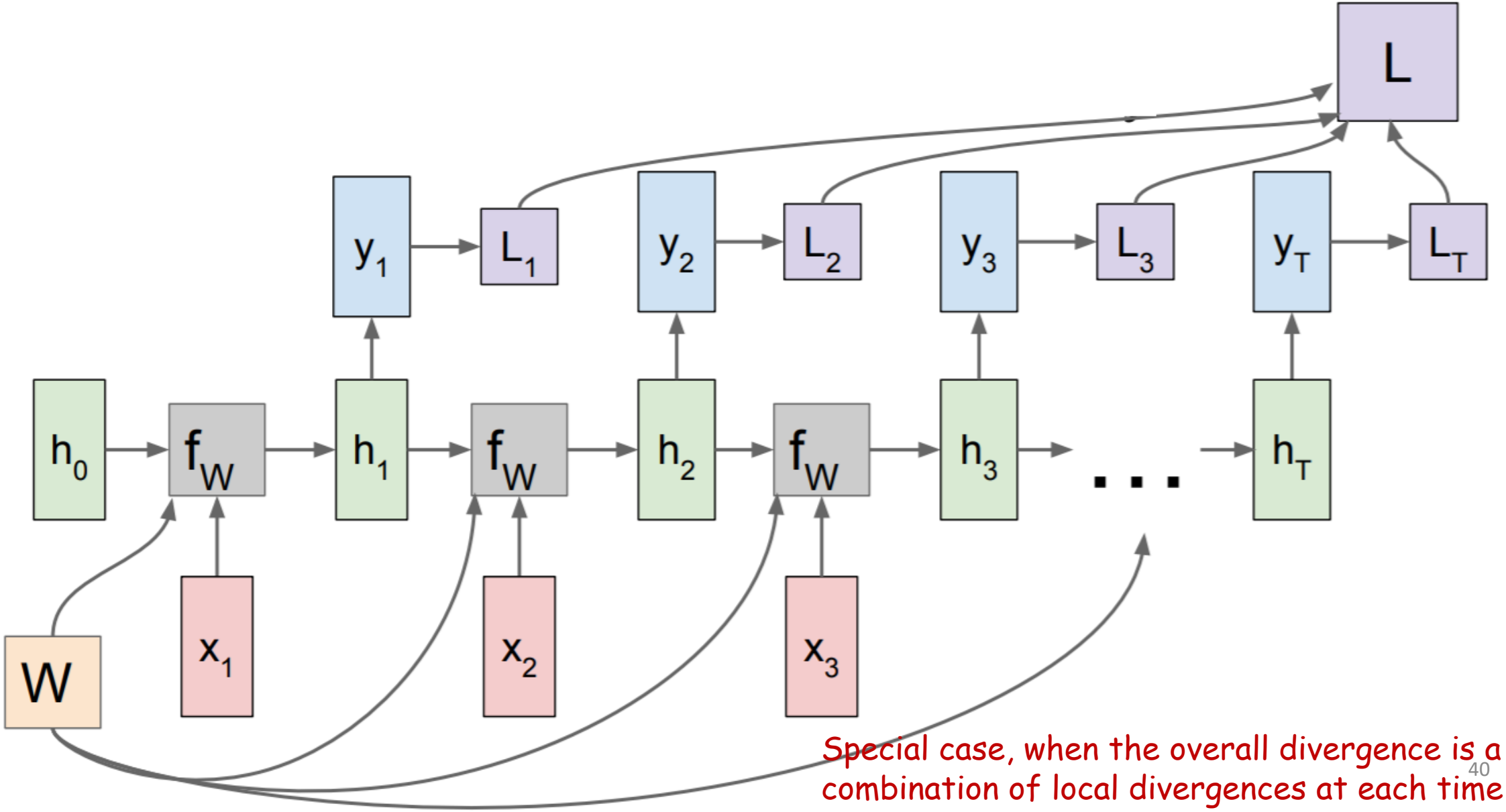


RNN: Computational Graph: Many to Many



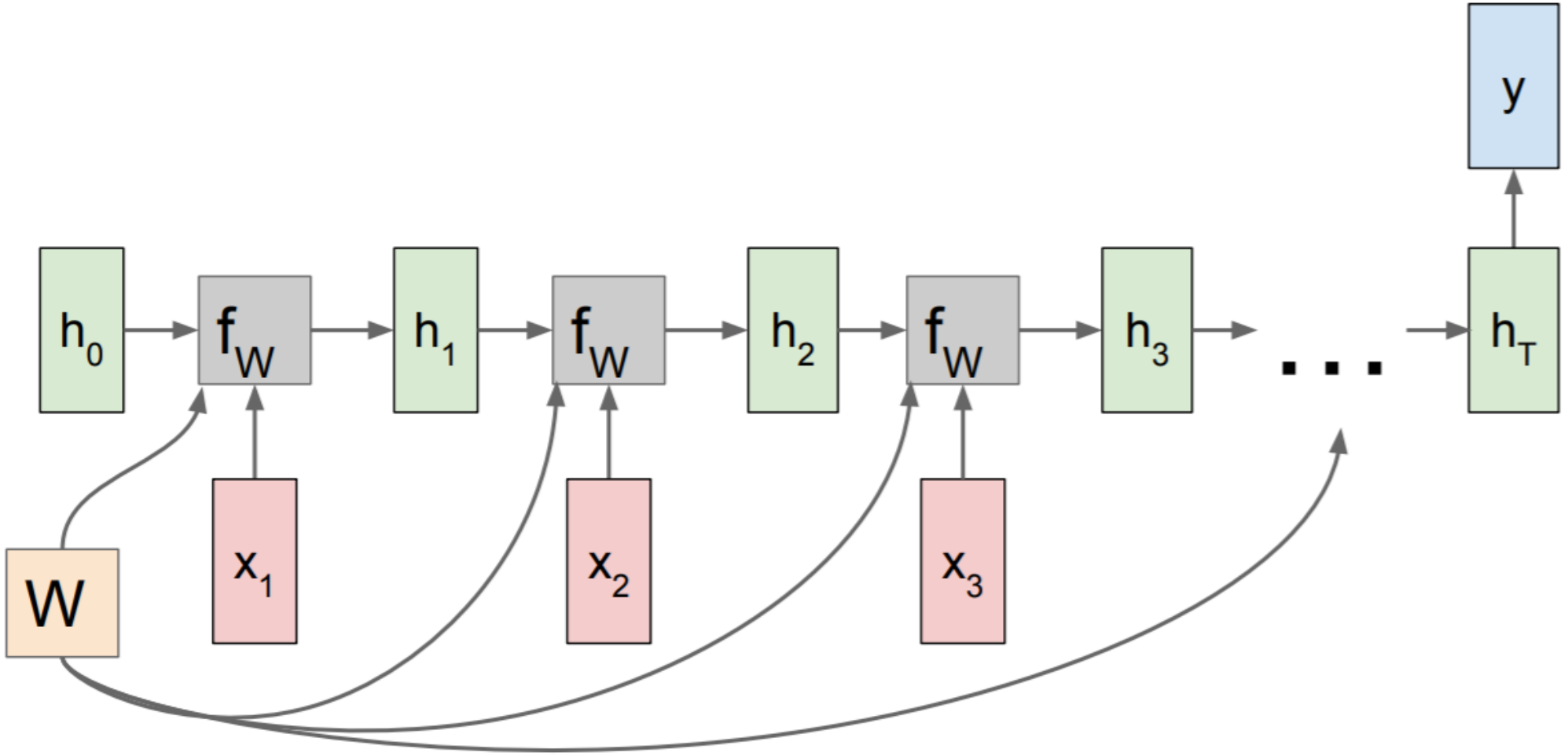
Special case, when the overall divergence is a simple combination of local divergences at each time

RNN: Computational Graph: Many to Many

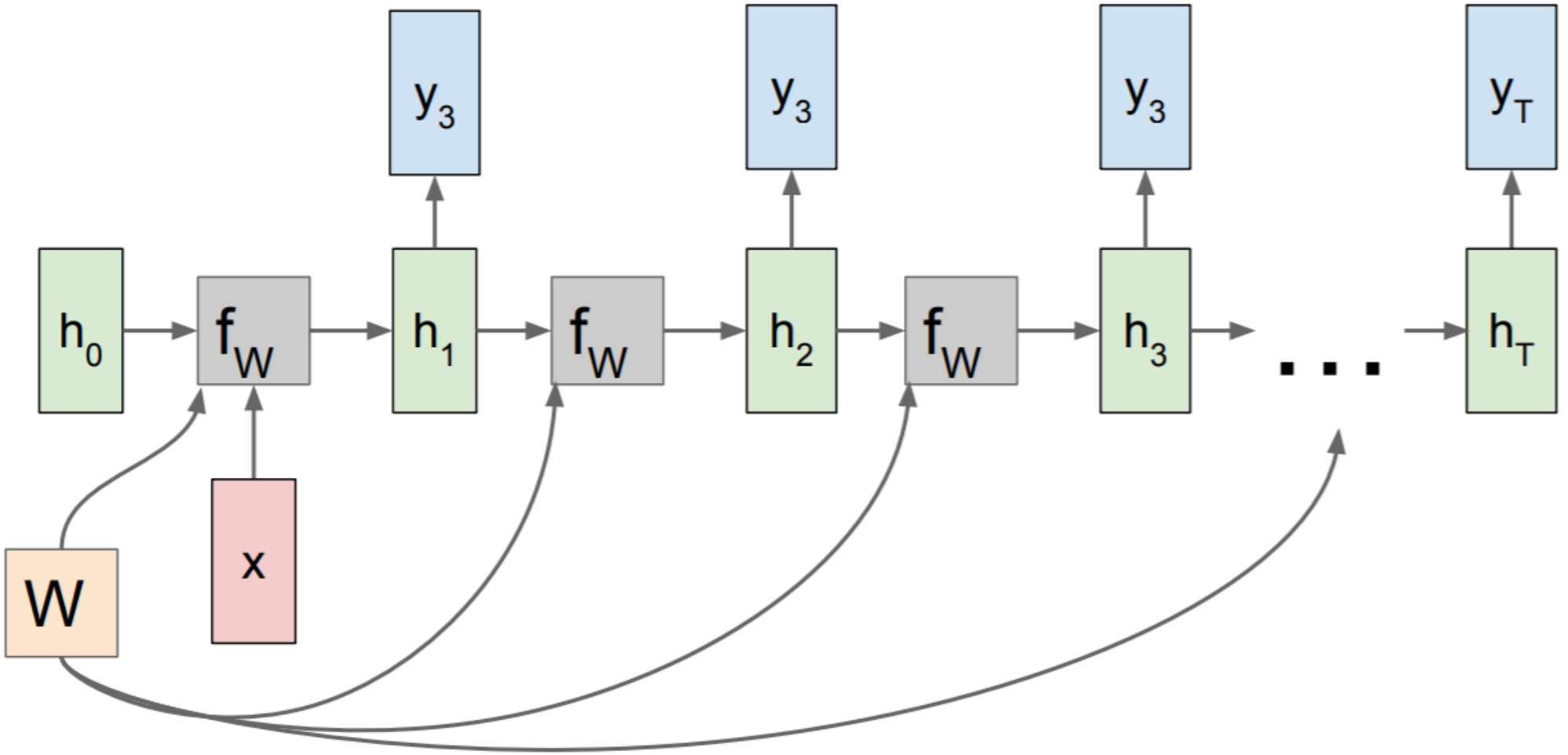


Special case, when the overall divergence is a simple combination of local divergences at each time

RNN: Computational Graph: Many to One



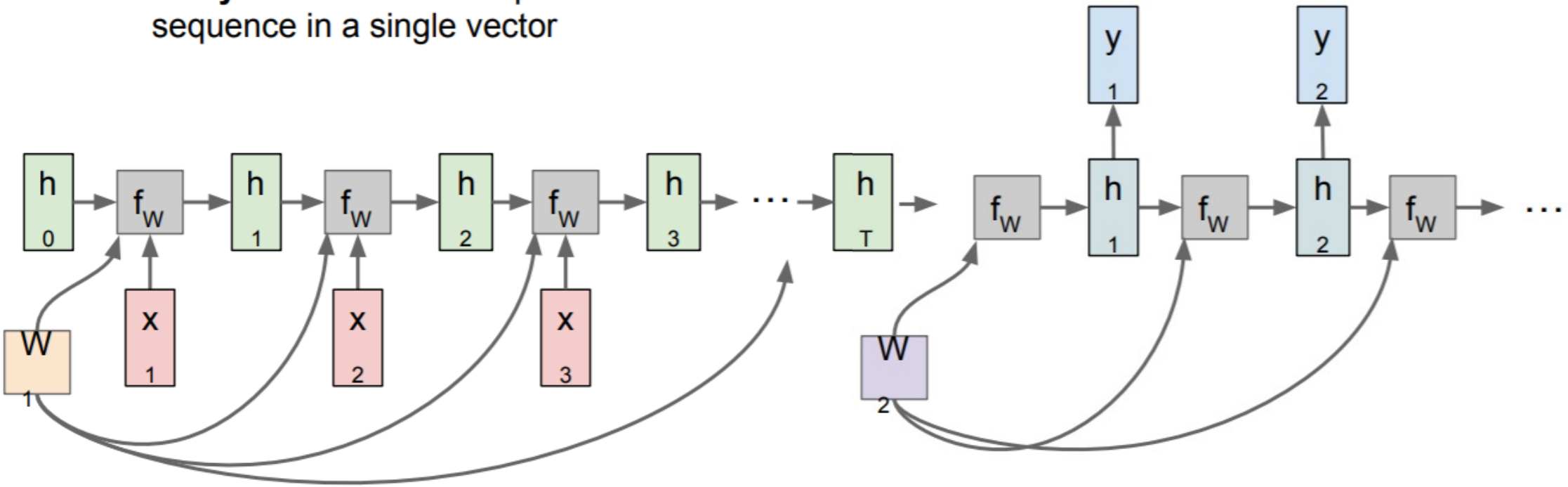
RNN: Computational Graph: One to Many



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

Many to one: Encode input sequence in a single vector

One to many: Produce output sequence from single input vector

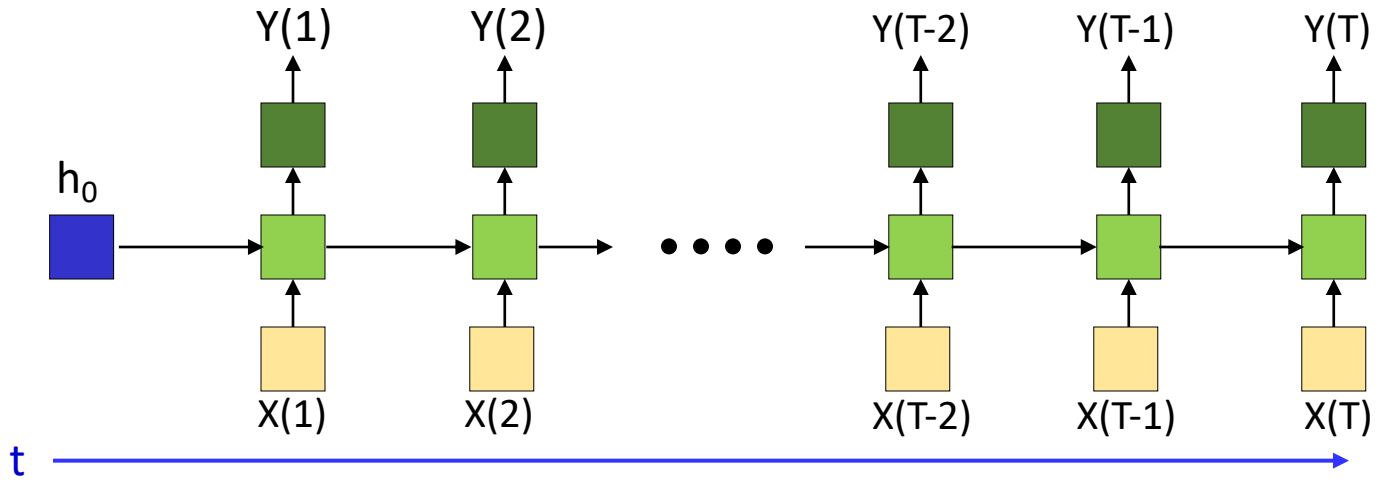


Story so far

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

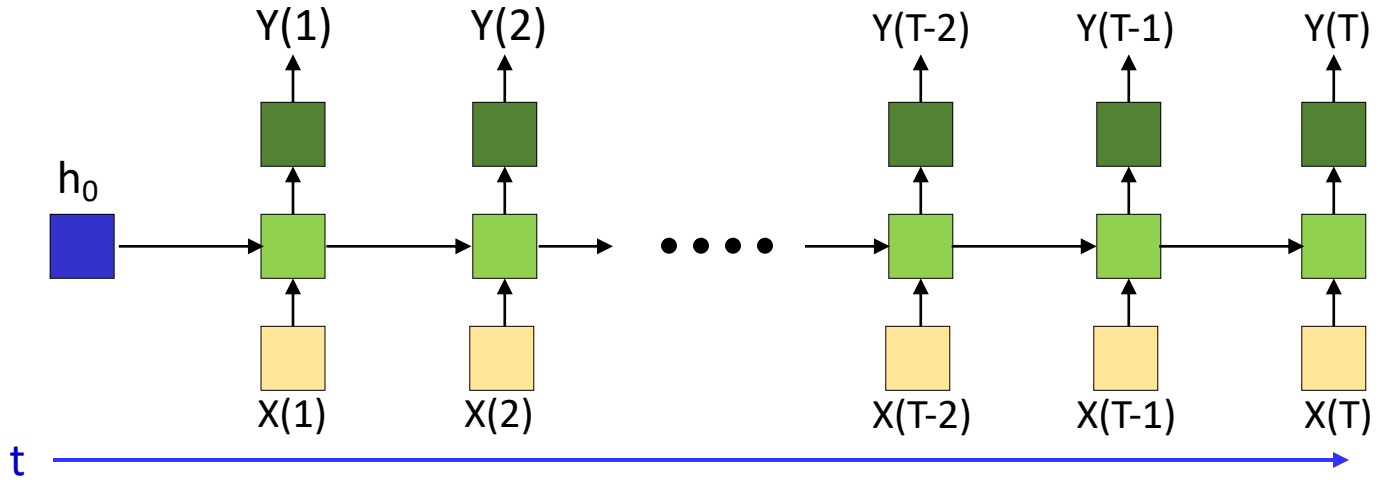
- State-space models retain information about the past through recurrent hidden states
  - These are "fully recurrent" networks
  - The initial values of the hidden states are generally learnable parameters as well

How do we *train* the network



- Back propagation through time (BPTT)
- Given a collection of *sequence* inputs
  - $(\mathbf{X}_i, \mathbf{Y}_i^{target})$, where
  - $\mathbf{X}_i = X_{i,0}, \ldots, X_{i,T}$
  - $\mathbf{Y}_i^{target} = Y_{i,0}^{target}, \ldots, Y_{i,T}^{target}$
- 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

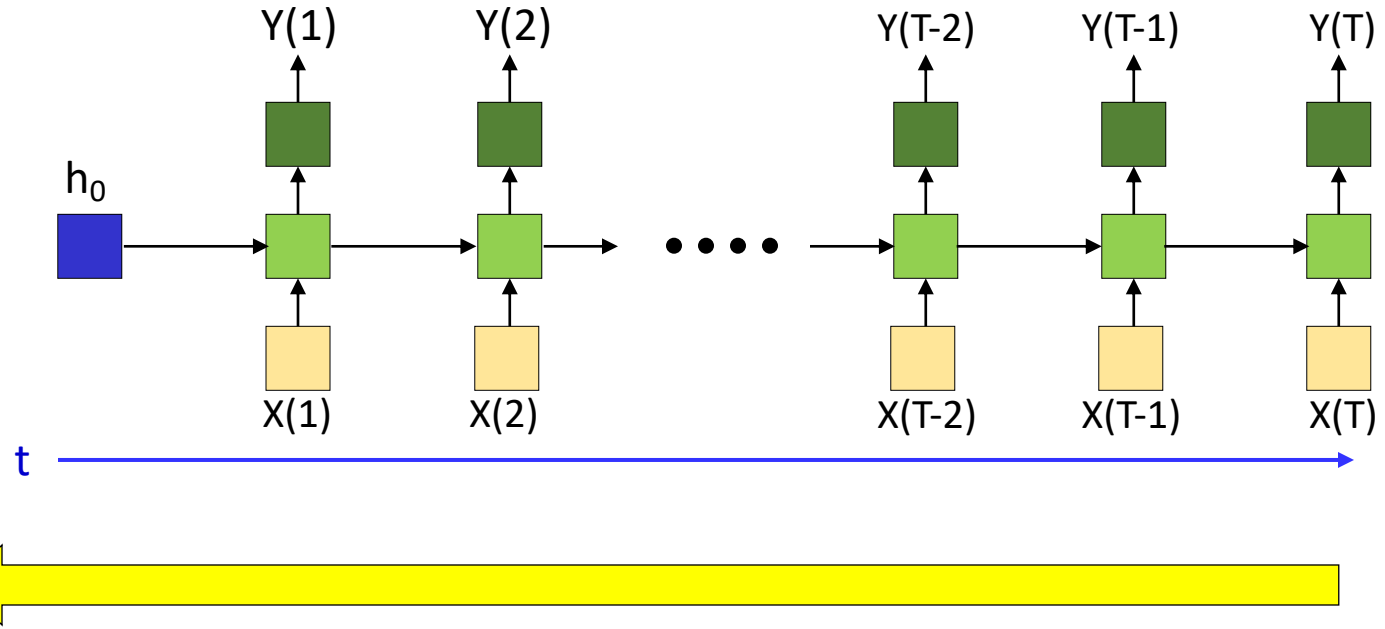


- For each training input:
- Forward pass: pass the entire data sequence through the network, generate outputs

Recurrent Neural Net: Assuming time-synchronous output

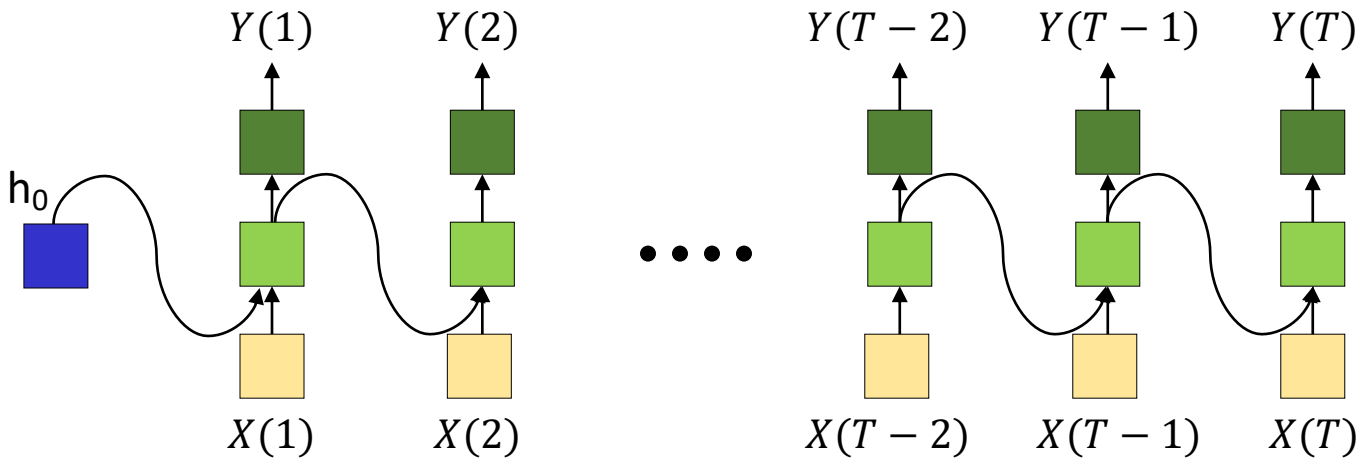
```
# Assuming h(0,*) is known
# Assuming L hidden-state layers and an output layer
# W_c(*) and W_r(*) are matrics, b(*) are vectors
# W_c are weights for inputs from current time
# W_r is recurrent weight applied to the previous time
# W_o are output layre weights

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

Training: Computing gradients



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

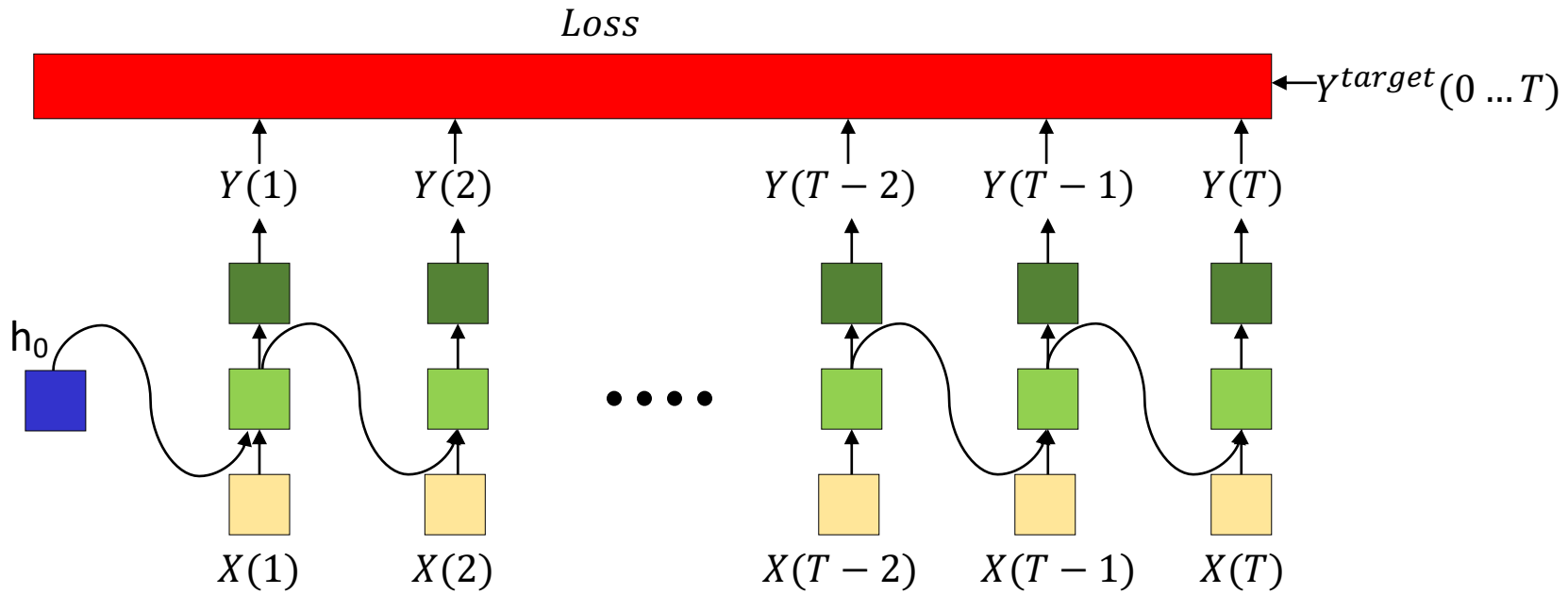
Back Propagation Through Time



Will only focus on *one* training instance

All subscripts represent *components* and not training instance index

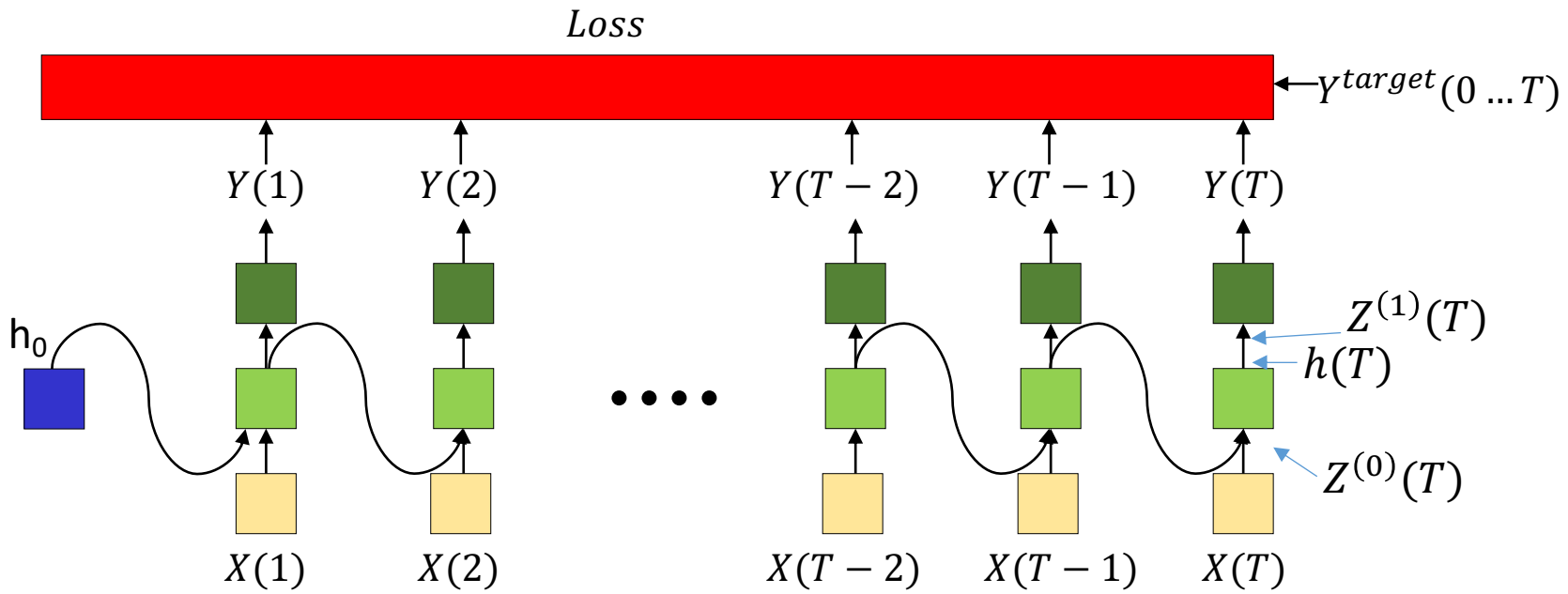
Back Propagation Through Time



- The divergence computed is between the *sequence of outputs* by the network and the *desired sequence of outputs*
  - Loss is a scalar function of a series of vectors!

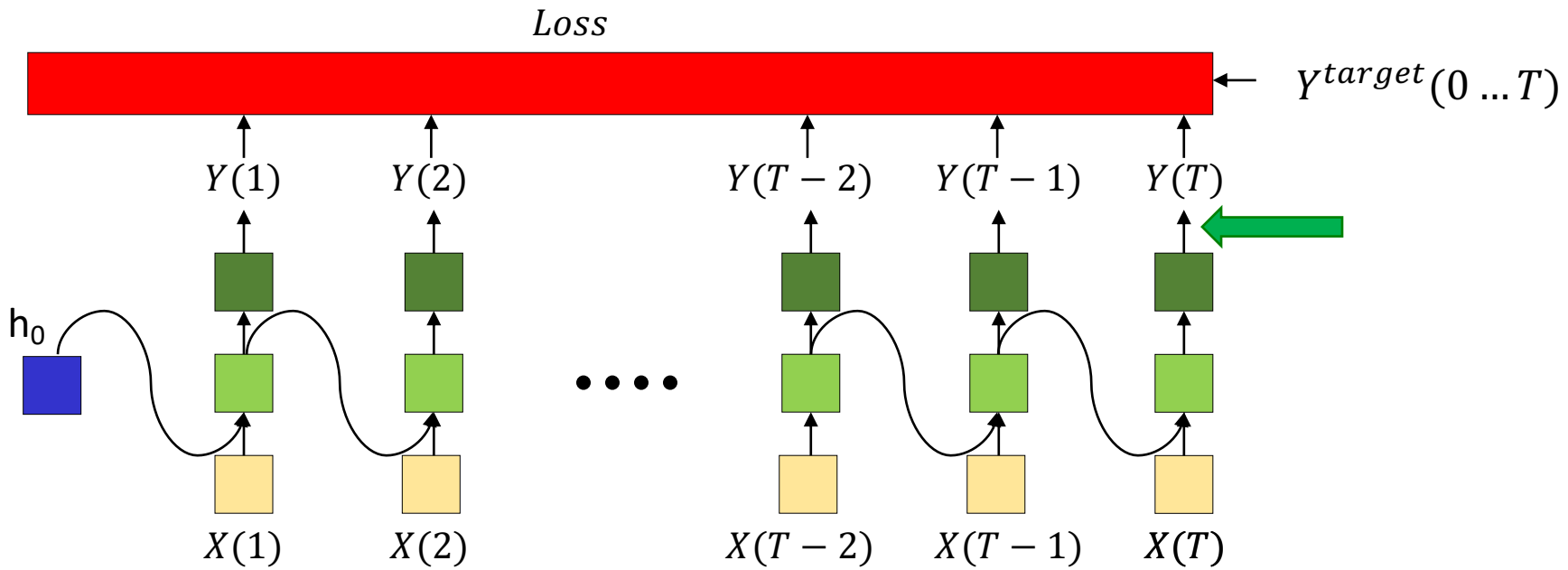
- This is *not* just the sum of the losses at individual times
  - Unless we explicitly define it that way

Notation



- $Y(t)$ is the output at time t
  - $Y_i(t)$ is the ith output
- $Z^{(1)}(t)$ is the pre-activation value of the neurons at the output layer at time t
- $h(t)$ is the output of the hidden layer at time t
  - Assuming only one hidden layer in this example
- $Z^{(0)}(t)$ is the pre-activation value of the hidden layer at time t

Back Propagation Through Time

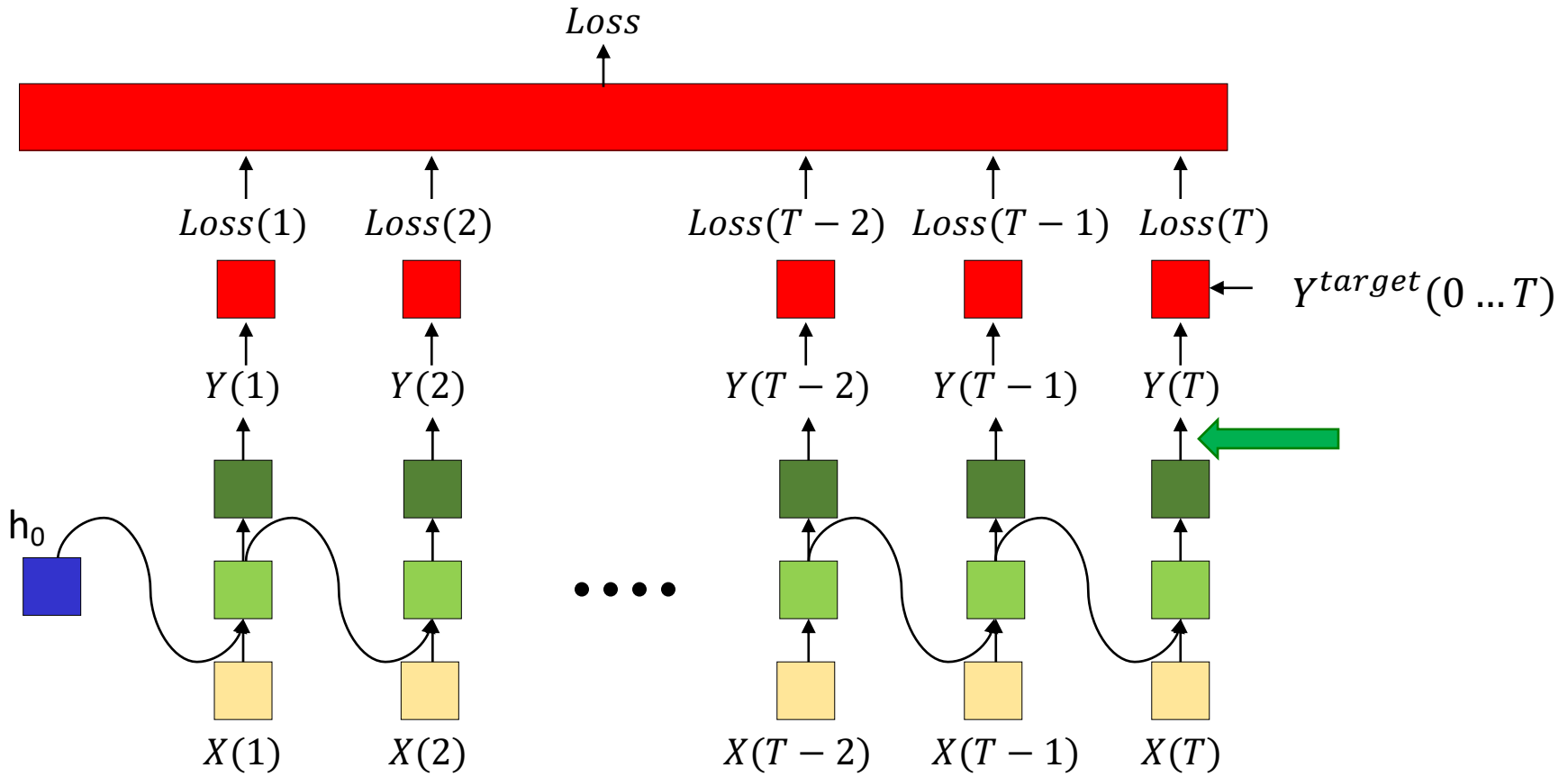


First step of backprop: Compute $\frac{dLoss}{dY_i(T)}$ for all i

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

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

Back Propagation Through Time



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

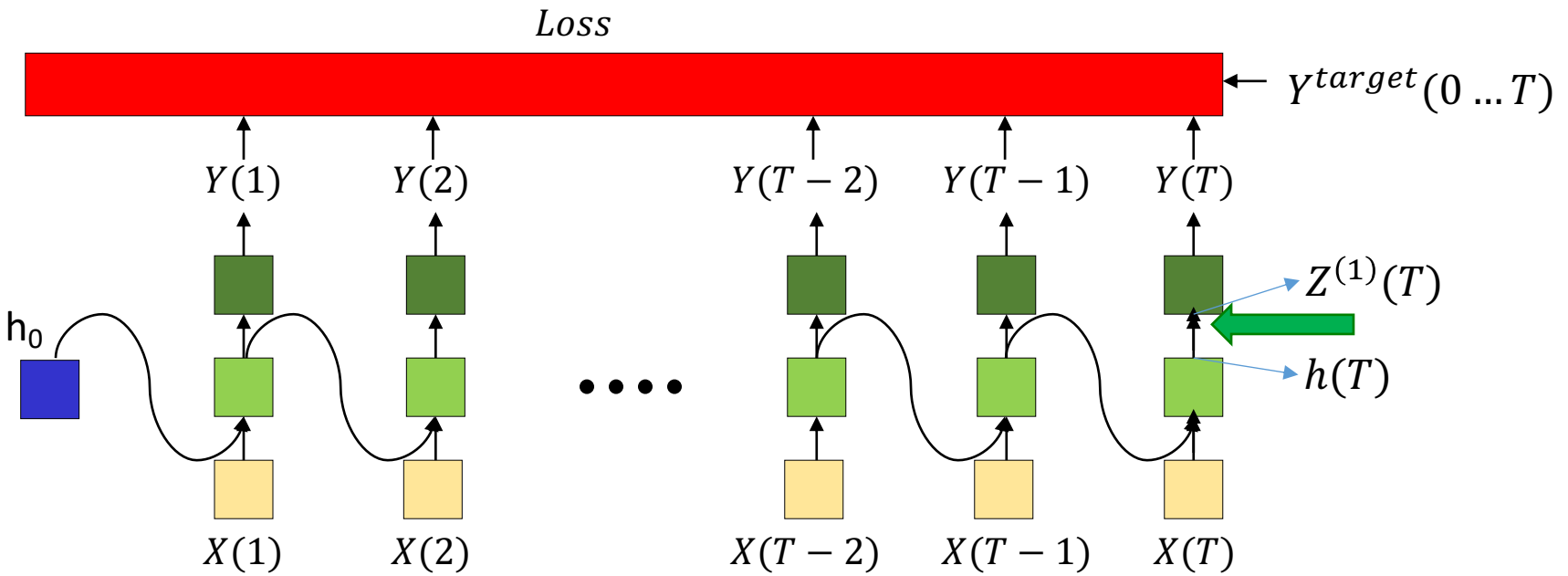
Must compute

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

Will usually get

$$\frac{dLoss}{dY_i(t)} = \frac{dLoss(t)}{dY_i(t)}$$

Back Propagation Through Time

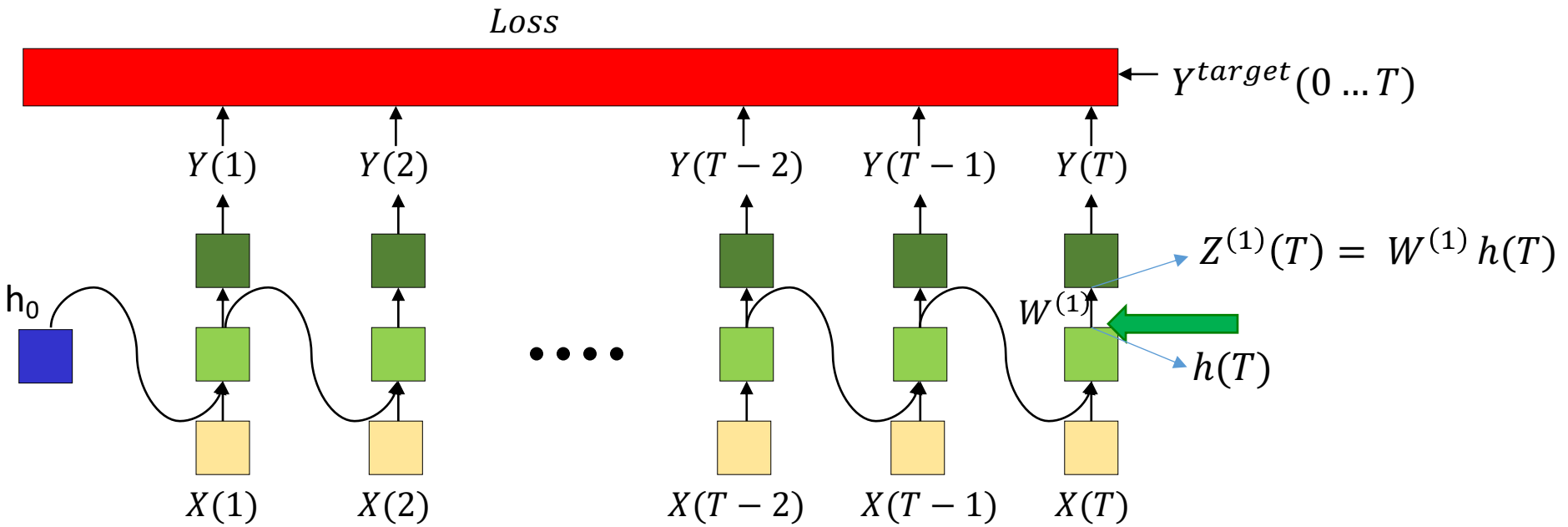


First step of backprop: Compute $\frac{dLoss}{dY_i(T)}$ for all i

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

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

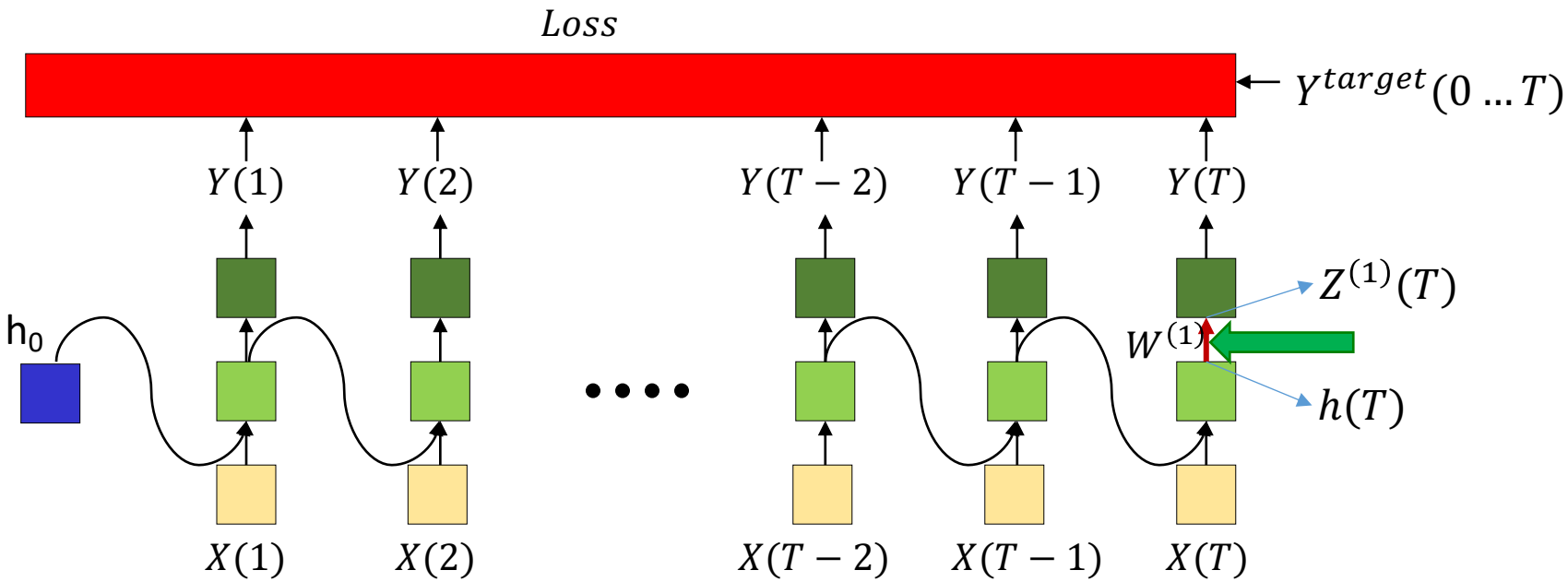
Back Propagation Through Time



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

$$\frac{dLoss}{dh_i(T)} = \sum_j \frac{dZ_j^{(1)}(T)}{dh_i(T)} \frac{dLoss}{dZ_j^{(1)}(T)} = \sum_j w_{ji}^{(1)} \frac{dLoss}{dZ_j^{(1)}(T)}$$

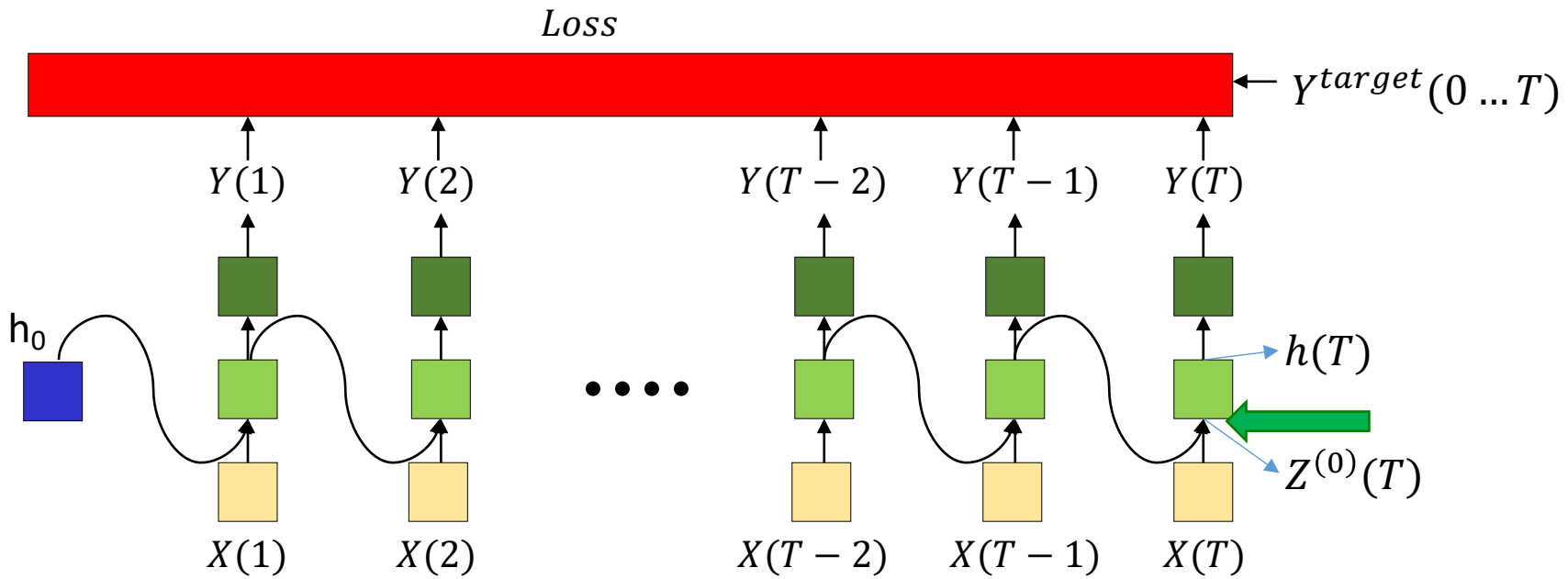
Back Propagation Through Time



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

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

Back Propagation Through Time



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

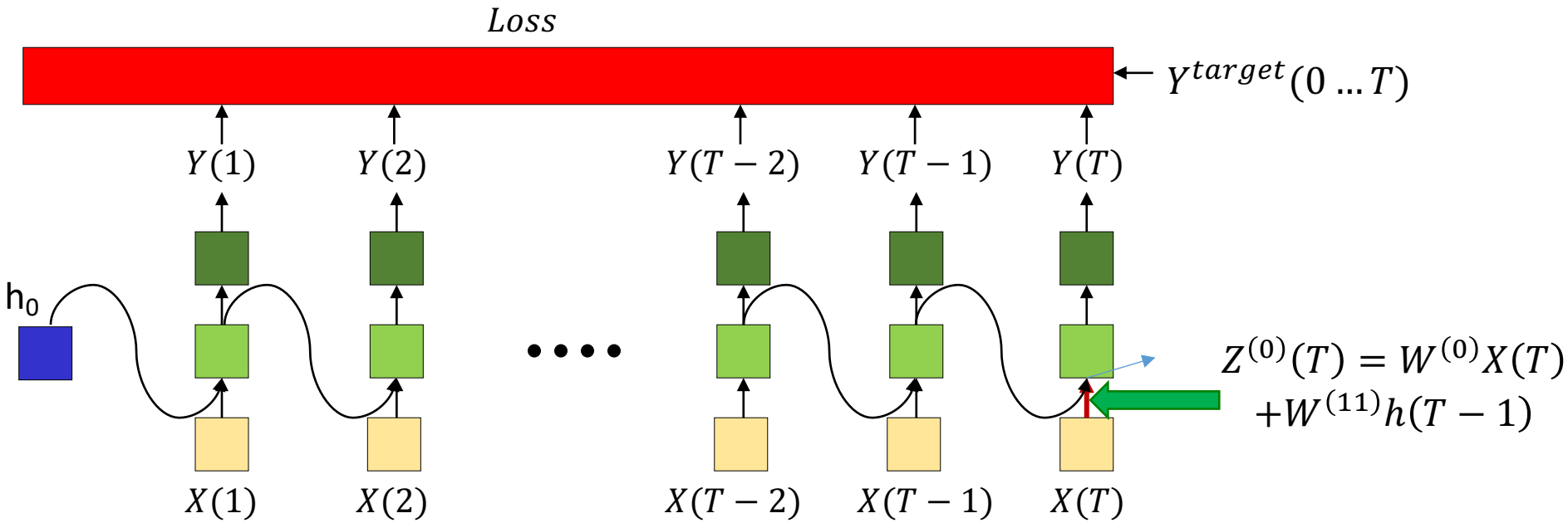
$$\frac{dLoss}{dh_i(T)} = \sum_j w_{ij}^{(1)} \frac{dLoss}{dZ_j^{(1)}(T)}$$

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

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

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

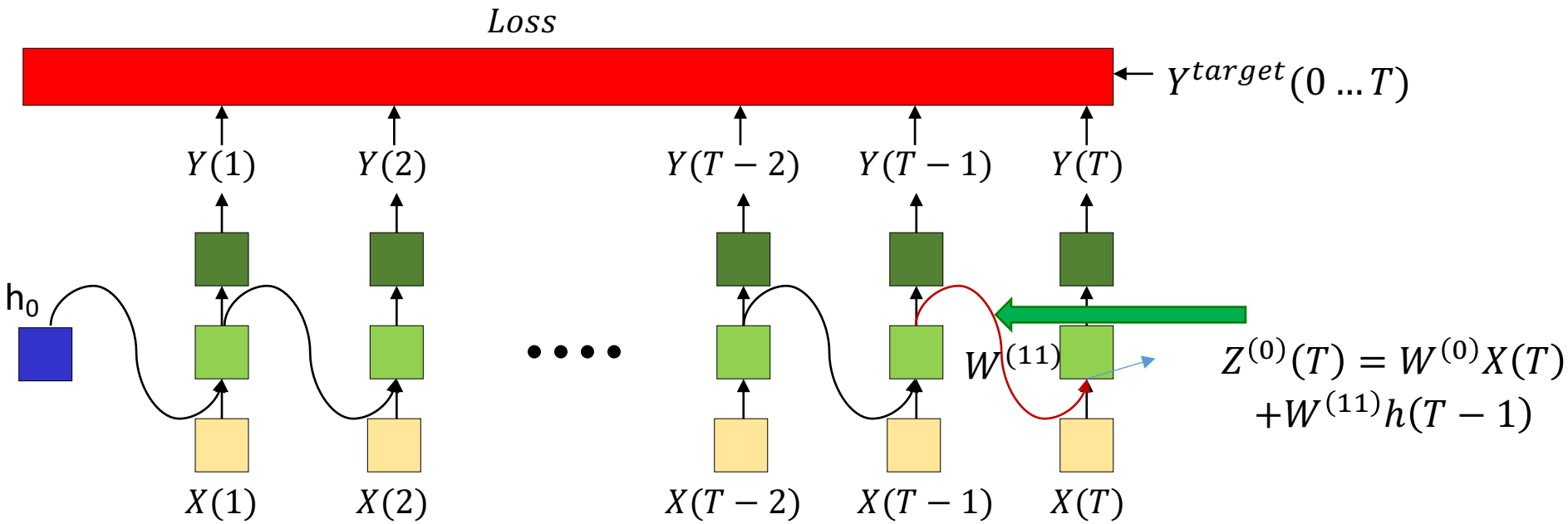
Back Propagation Through Time



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

$$\frac{dloss}{dw_{ij}^{(0)}} = X_j(T) \frac{dLoss}{dZ_i^{(0)}(T)}$$

Back Propagation Through Time

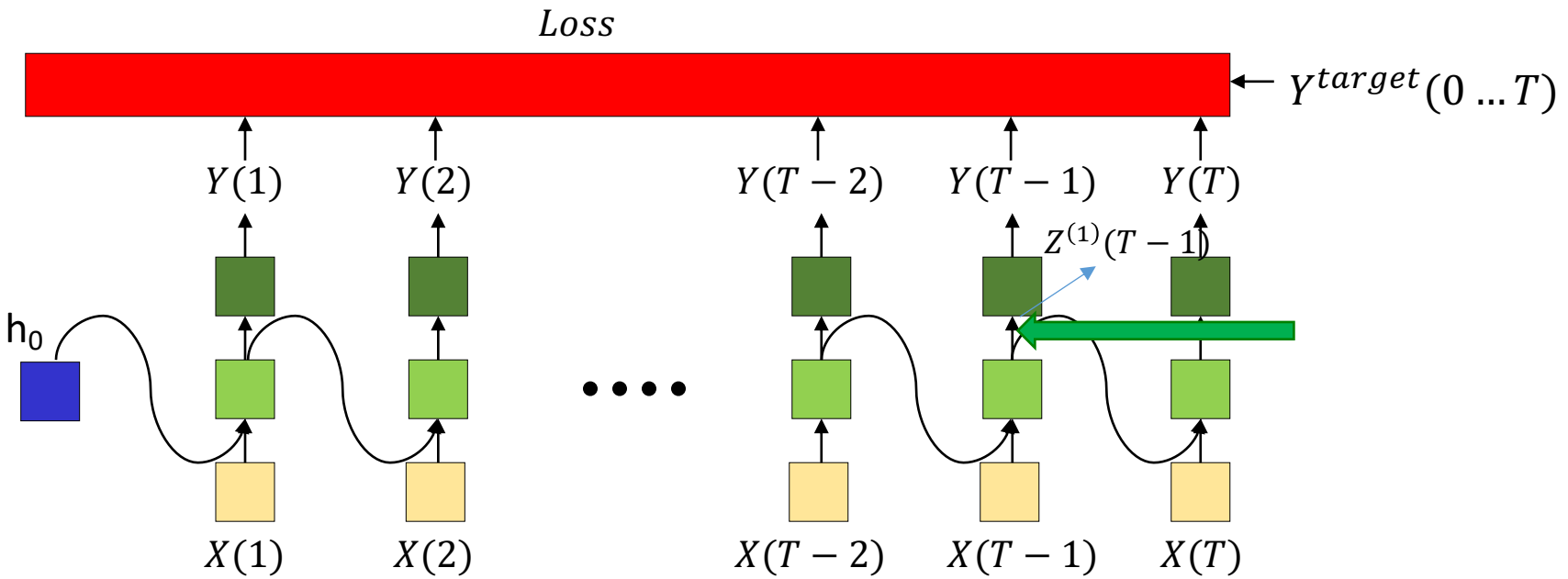


$$\nabla_{W^{(11)}} Loss = \nabla_{Z^{(0)}(T)} Loss \;\; h(T-1)^T$$

$$\frac{dLoss}{dw_{ij}^{(0)}} = X_j(T) \frac{dLoss}{dZ_i^{(0)}(T)}$$

$$\frac{dLoss}{dw_{ij}^{(11)}} = h_j(T-1) \frac{dLoss}{dZ_i^{(0)}(T)}$$

Back Propagation Through Time



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

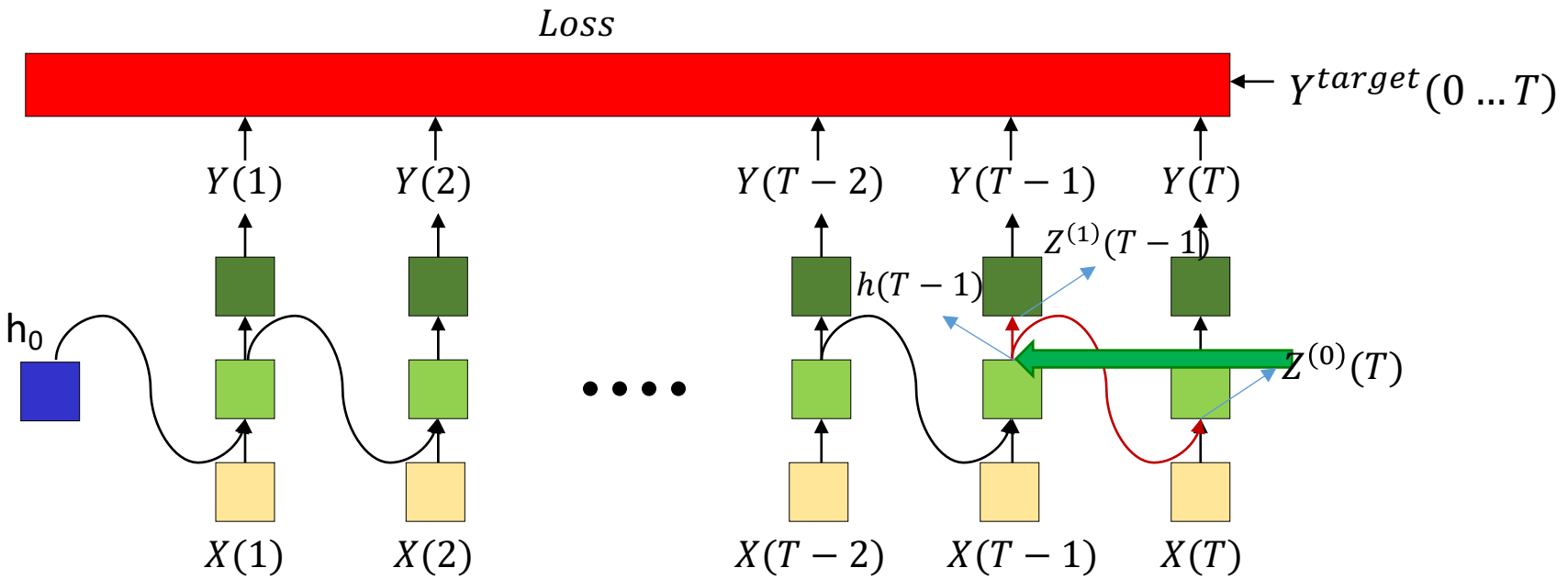
$$\frac{dLoss}{dZ_i^{(1)}(T-1)} = \frac{dY_i(T-1)}{dZ_i^{(1)}(T-1)} \frac{dLoss}{dY_i(T-1)}$$

OR

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

Vector output activation

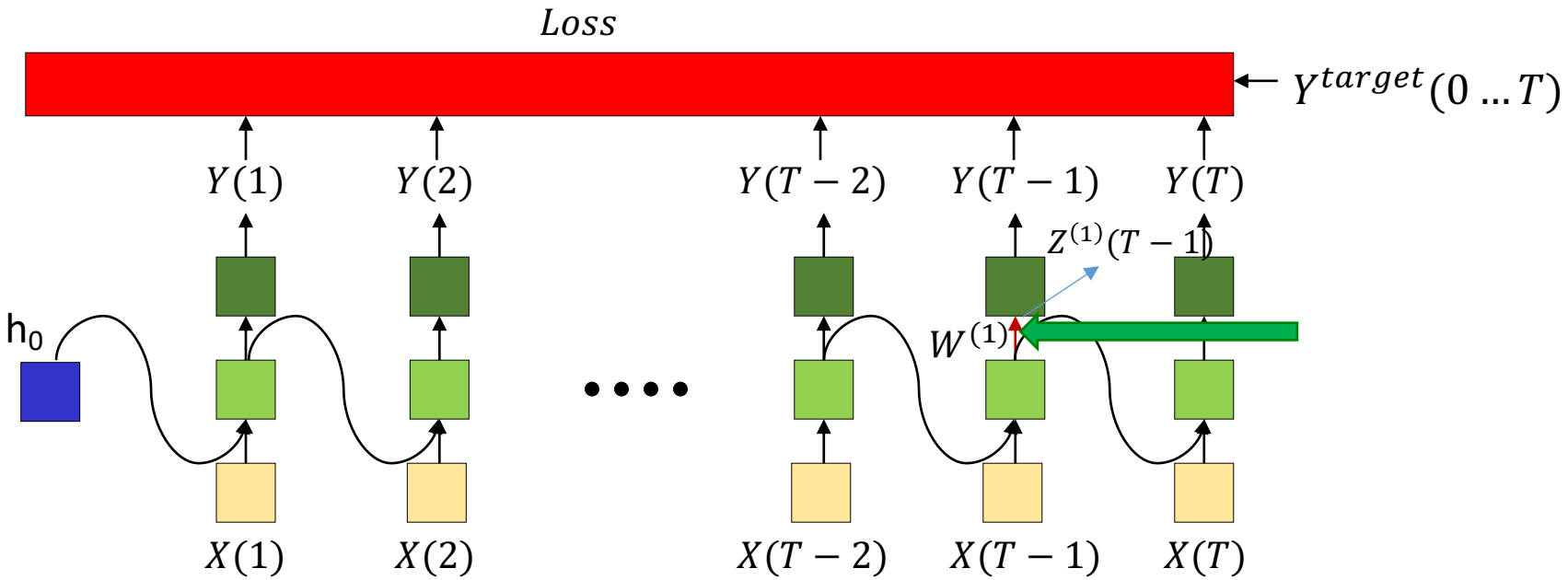
Back Propagation Through Time



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

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

Back Propagation Through Time

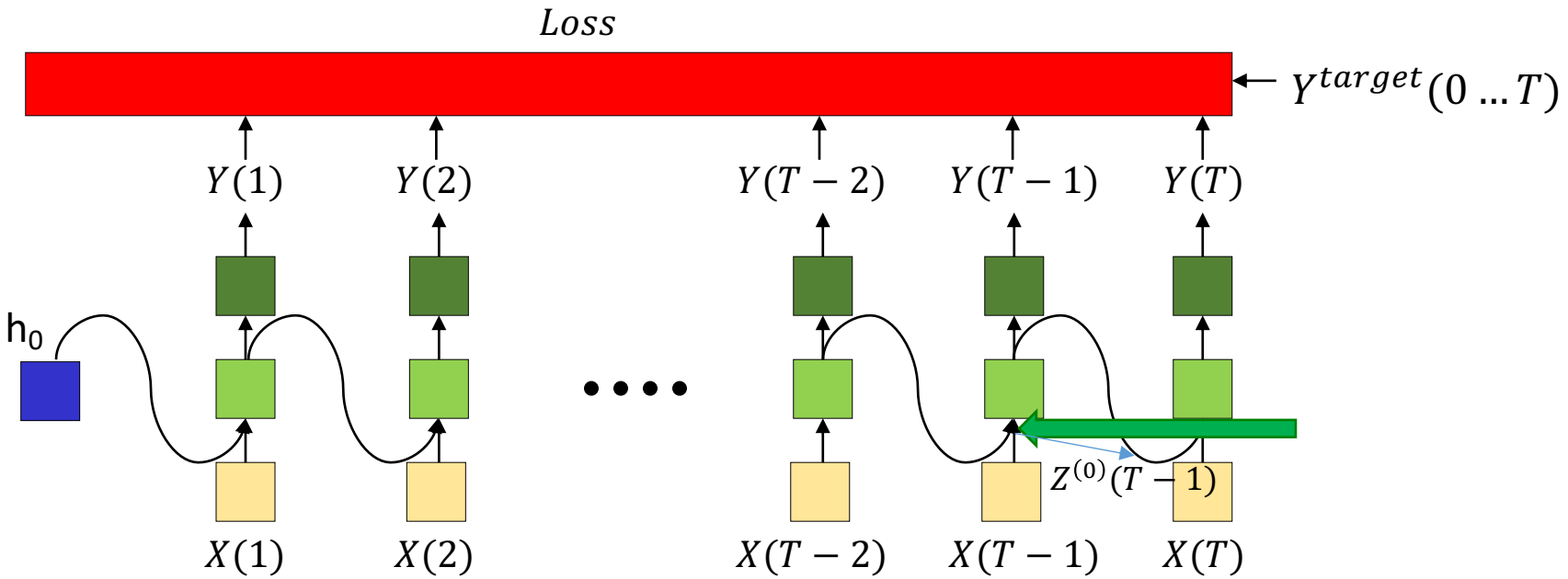


$$\nabla_{W^{(1)}} Loss \mathrel{+}= \nabla_{Z^{(1)}(T-1)} Loss \;\; h(T-1)^T$$

Note the addition

$$\frac{dLoss}{dw_{ij}^{(1)}} \mathrel{+}= h_j(T-1) \frac{dLoss}{dZ_i^{(1)}(T-1)}$$

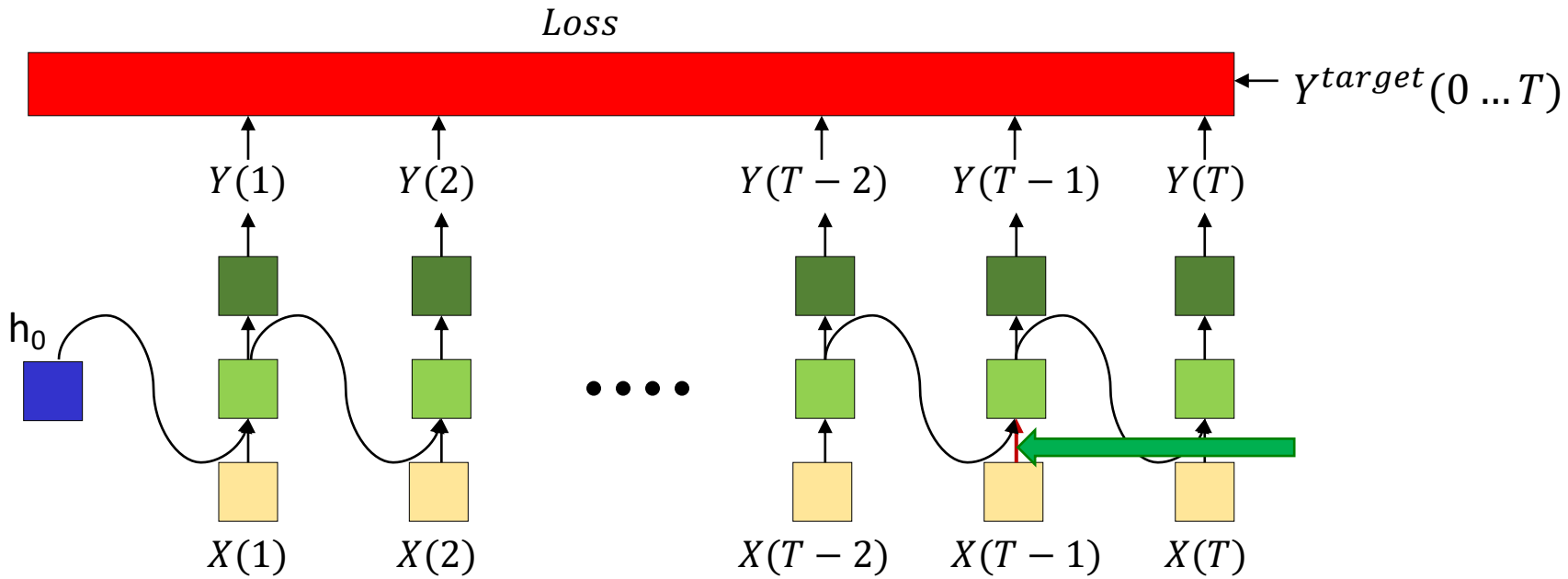
Back Propagation Through Time



$$\nabla_{Z^{(0)}(T-1)} Loss = \nabla_{Z^{(0)}(T-1)} h(T-1)\, \nabla_{h(T-1)} Loss$$

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

Back Propagation Through Time

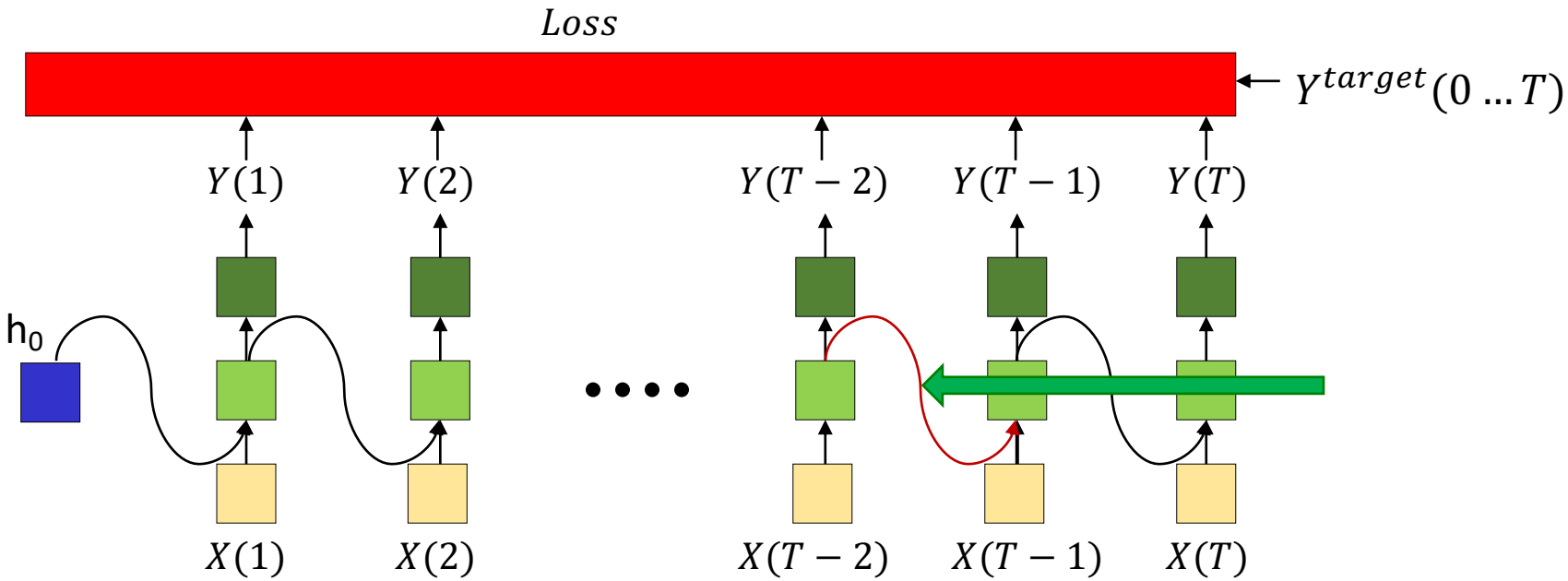


Note the addition

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

$$\frac{dLoss}{dw_{ij}^{(0)}} \mathrel{+}= X_j(T-1) \frac{dLoss}{dZ_i^{(0)}(T-1)}$$

Back Propagation Through Time

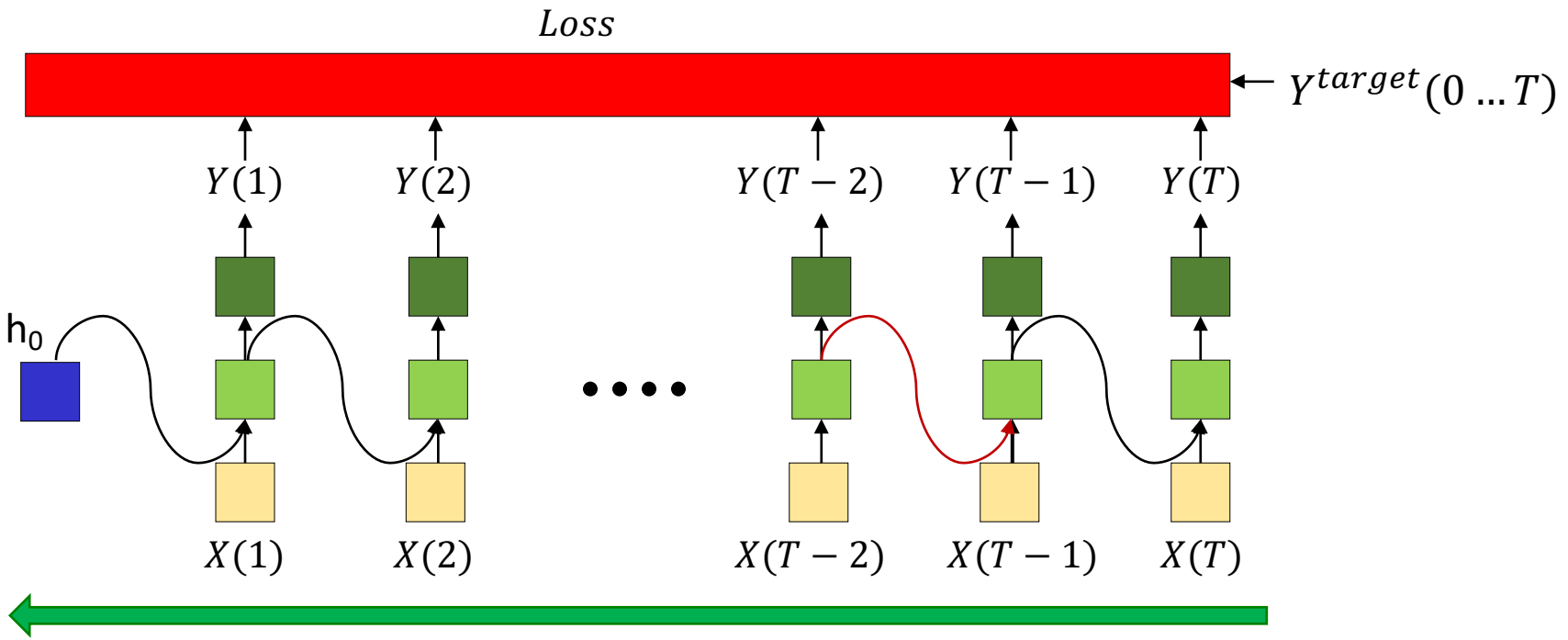


$$\nabla_{W^{(11)}} Loss \mathrel{+}= \nabla_{Z^{(0)}(T-1)} Loss \;\; h(T-2)^T$$

Note the addition

$$\frac{dLoss}{dw_{ij}^{(11)}} \mathrel{+}= h_j(T-2)\frac{dLoss}{dZ_i^{(0)}(T-1)}$$

Back Propagation Through Time



$$\nabla_{h_0} Loss = W^{(11)^T} \nabla_{Z^{(1)}(0)} Loss$$

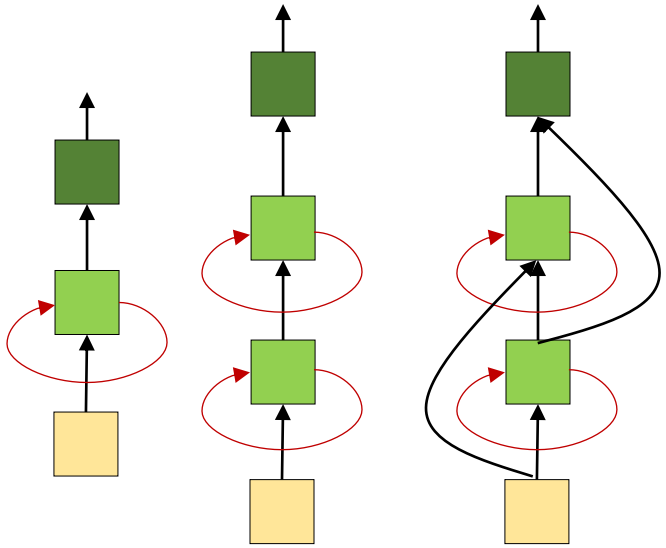
Continue computing derivatives going backward through time until..

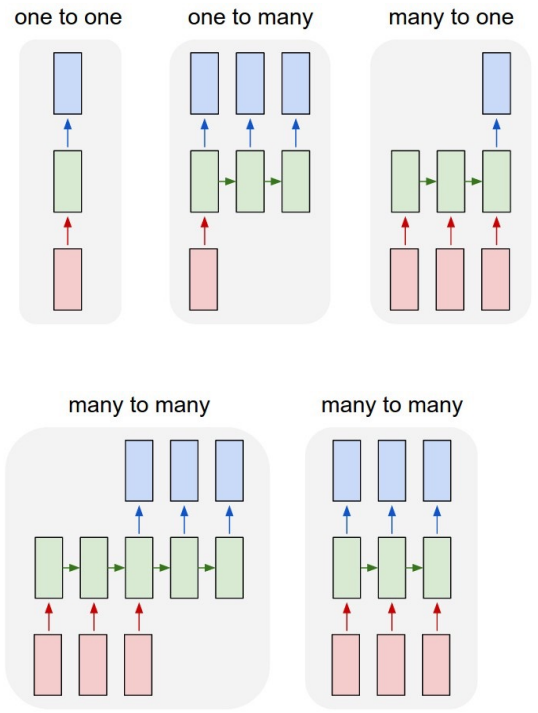
$$\frac{dLoss}{dh_{0,i}} = \sum_j w_{ji}^{(11)} \frac{dLoss}{dZ_j^{(1)}(0)}$$

BPTT

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

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

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

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

BPTT

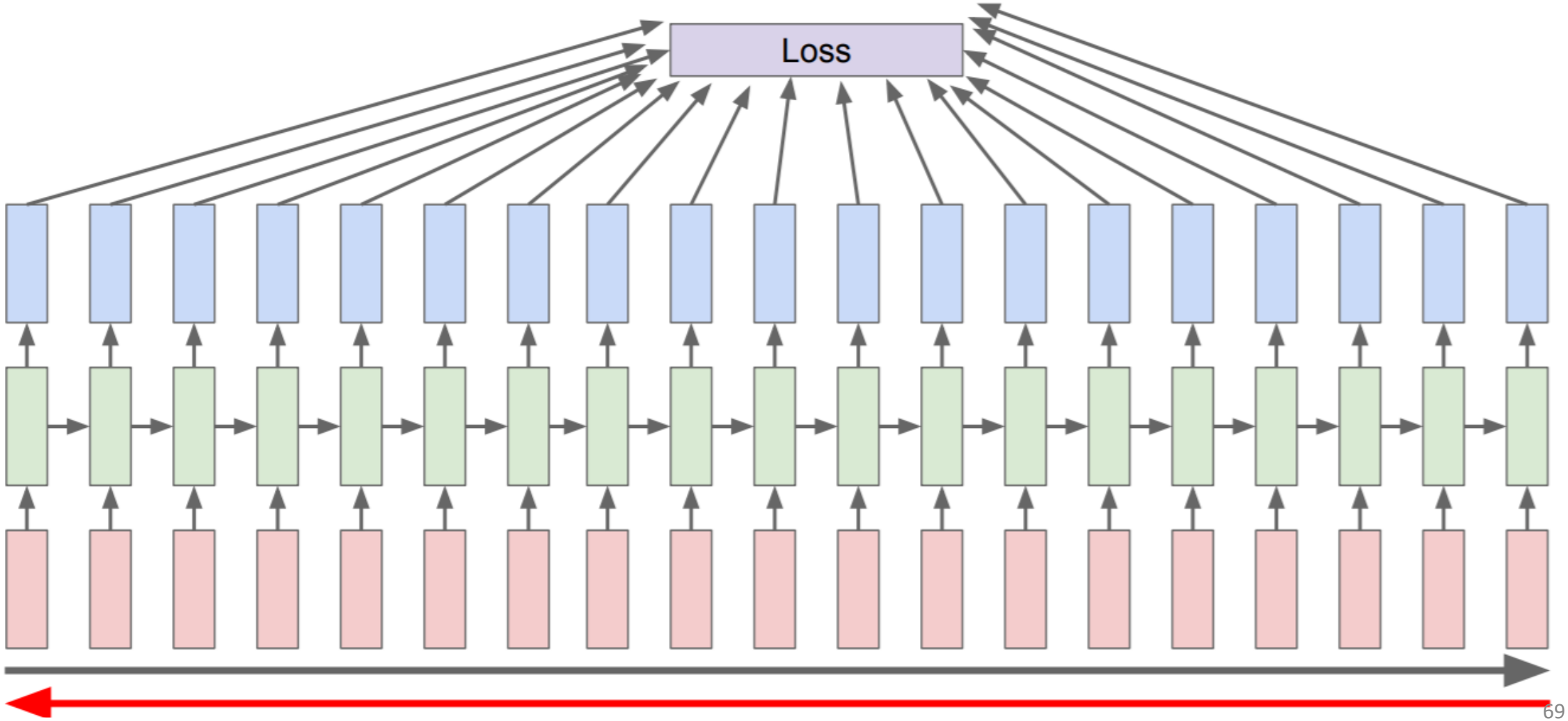




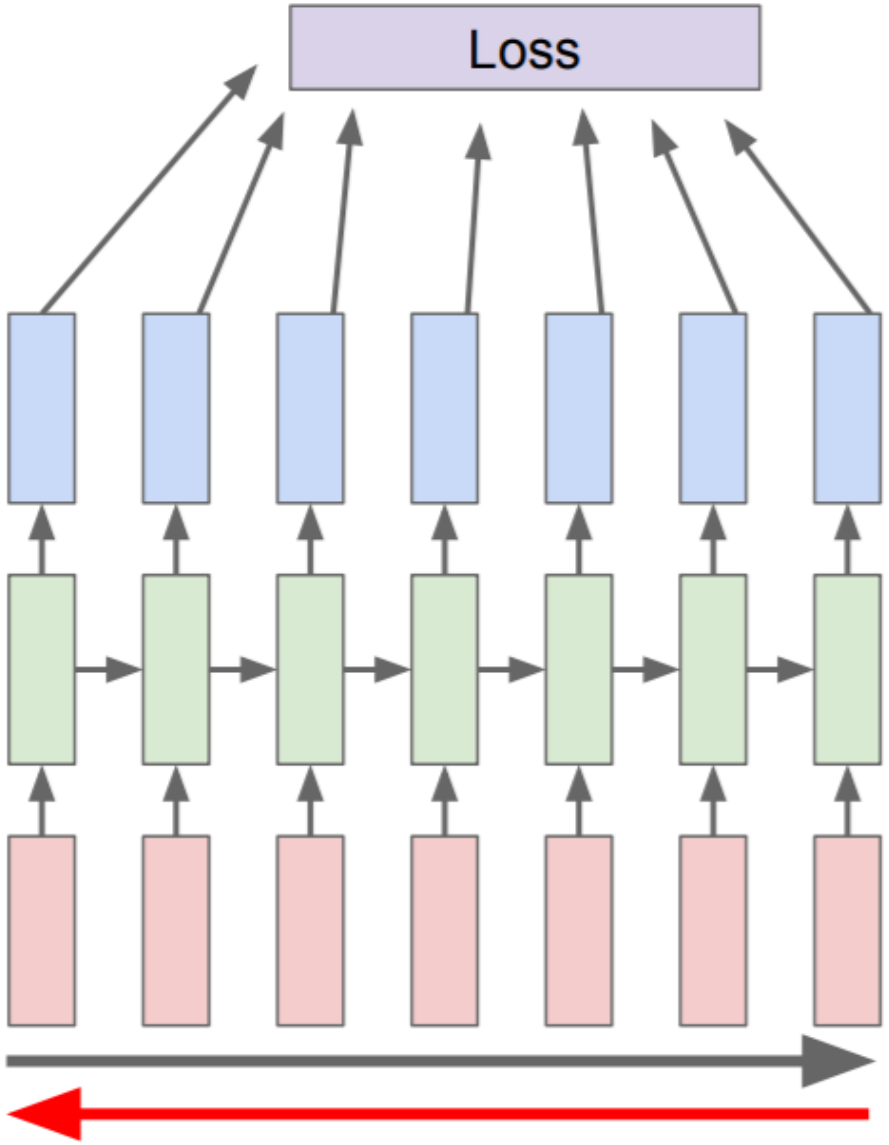
- Can be generalized to any architecture

Backpropagation through time

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

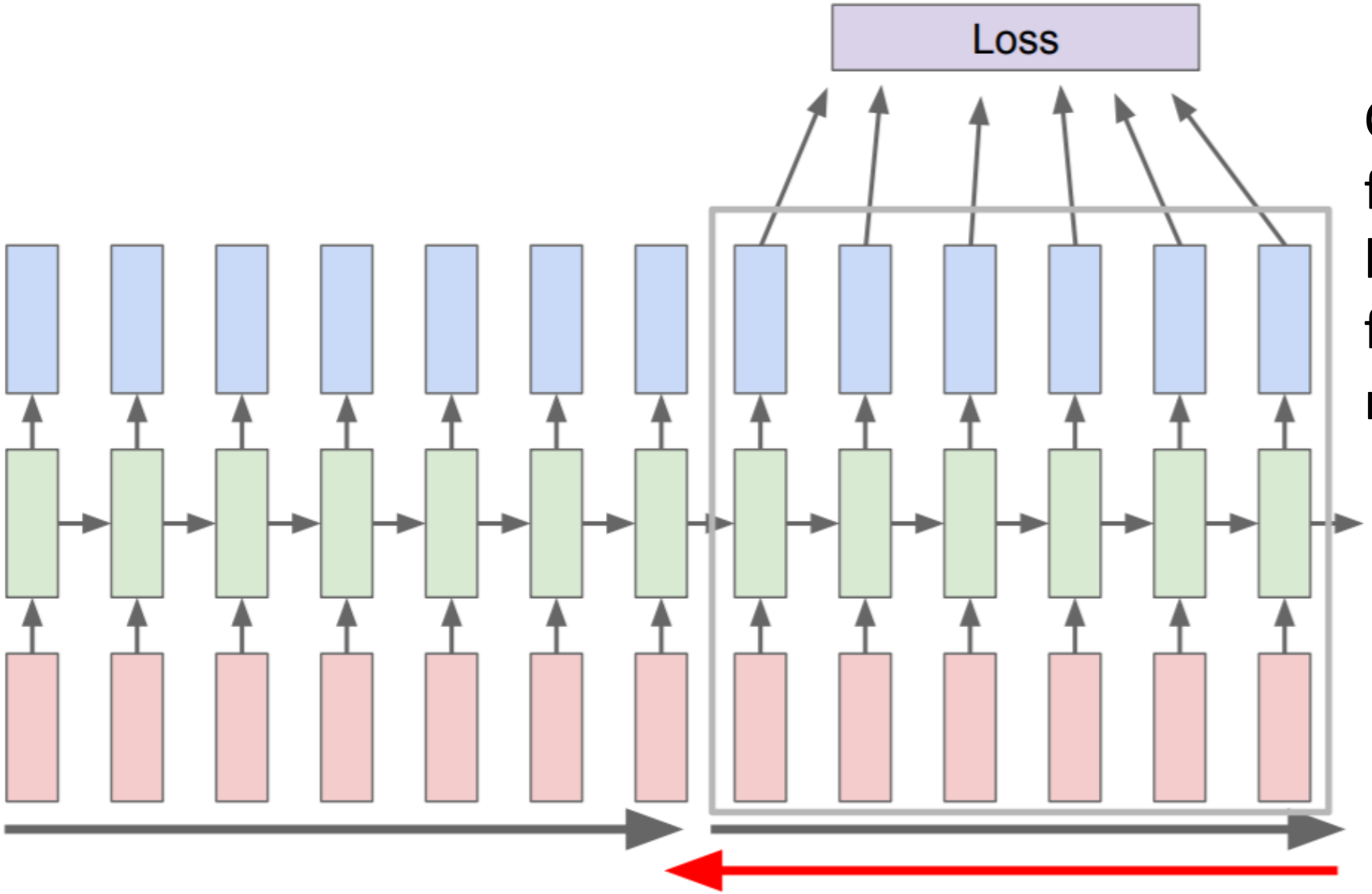


Truncated Backpropagation through time



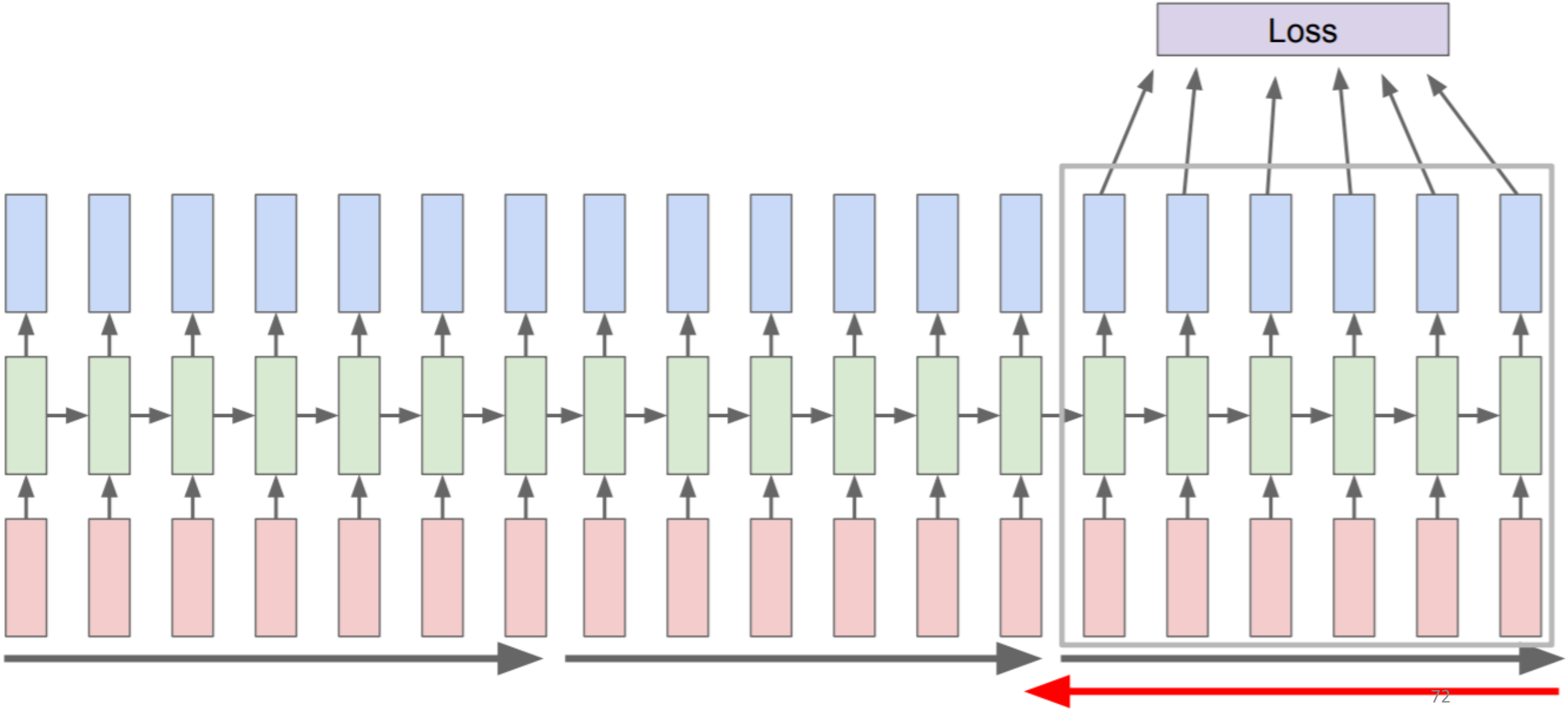
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time

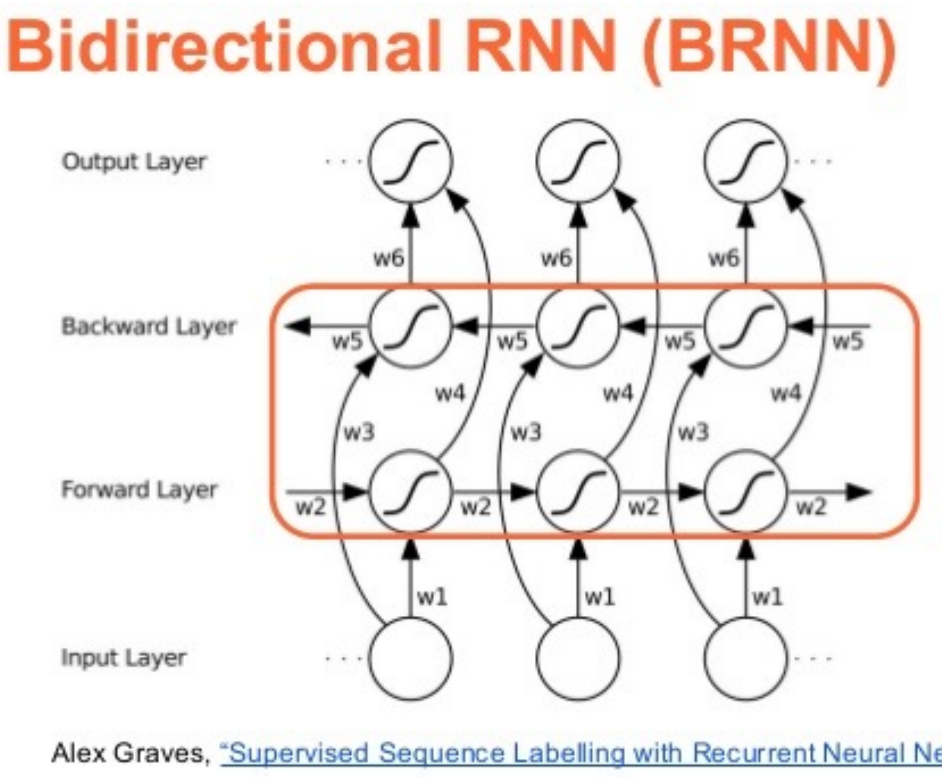


Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation through time



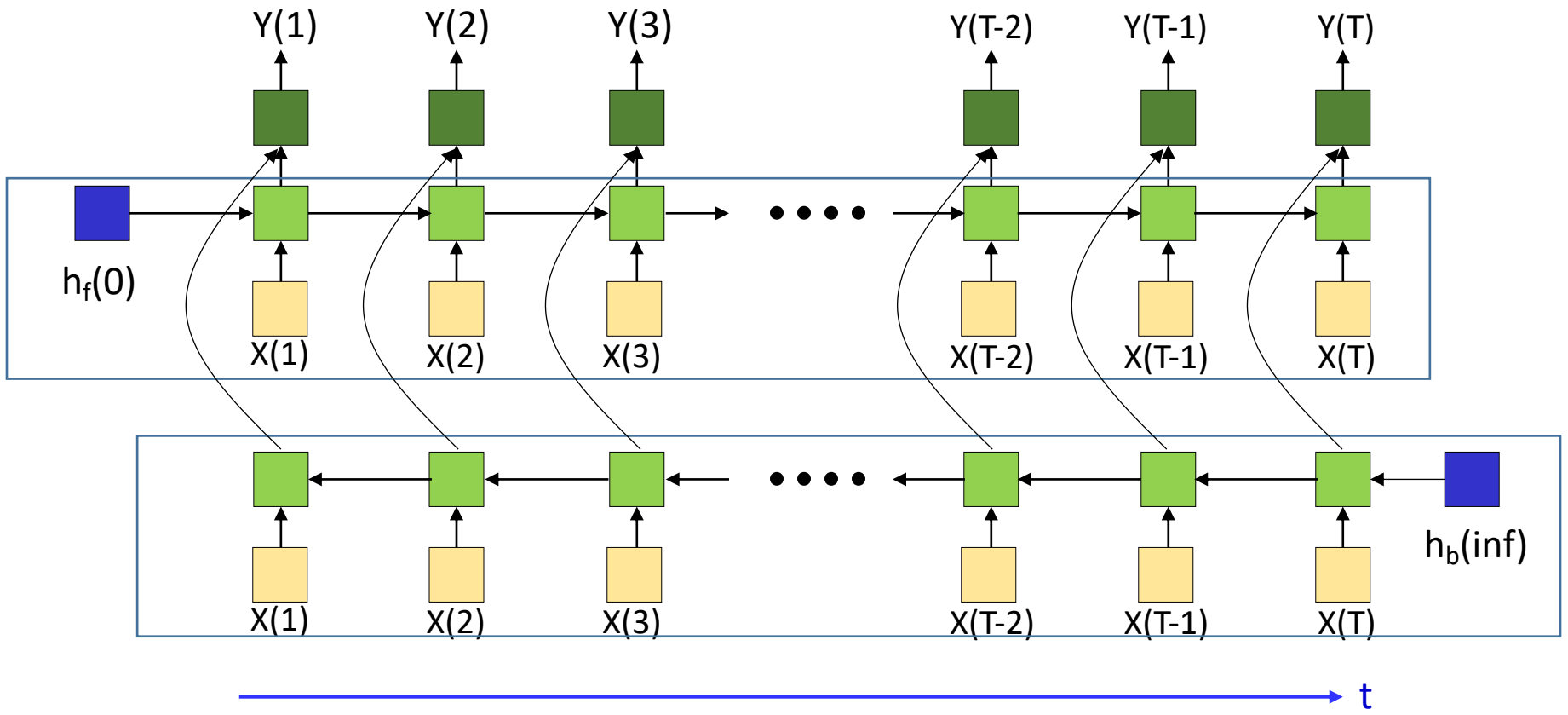
Extensions to the RNN: *Bidirectional RNN*



Proposed by Schuster and Paliwal 1997

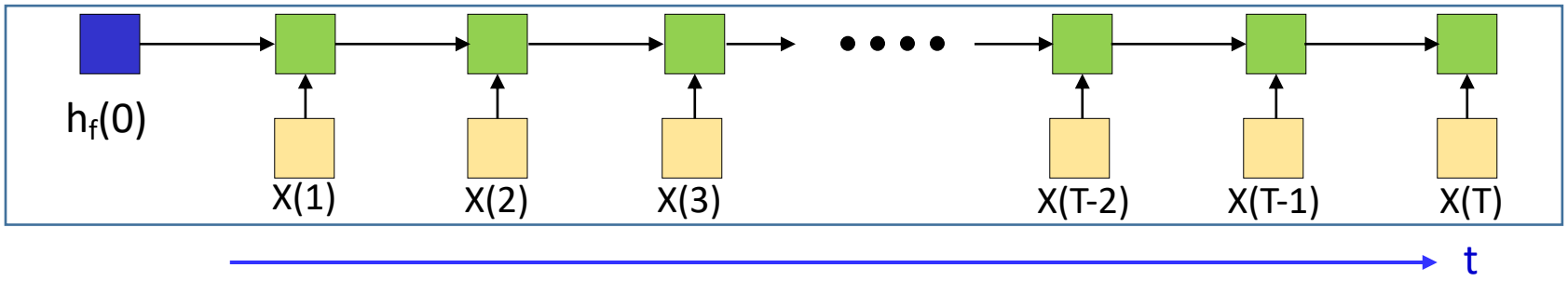
- RNN with both forward and backward recursion
  - Explicitly models the fact that just as the future can be predicted from the past, the past can be deduced from the future

Bidirectional RNN



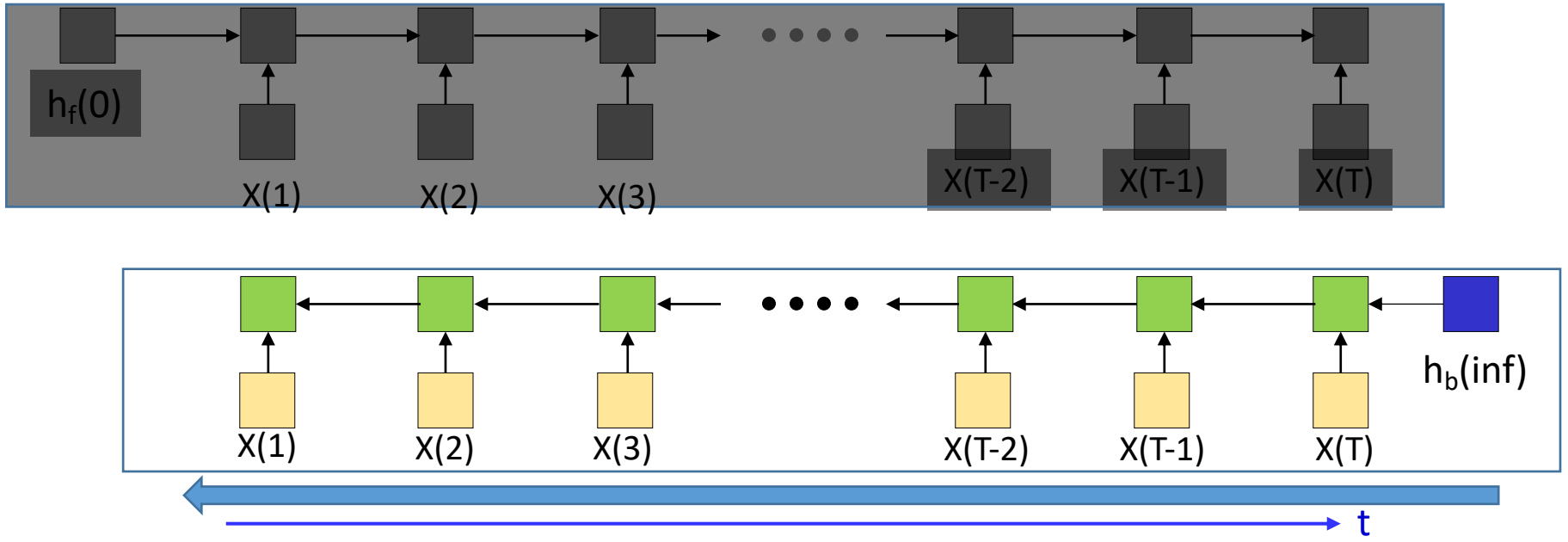
- A forward net process the data from t=1 to t=T
- A backward net processes it backward from t=T down to t=1

Bidirectional RNN: Processing an input string



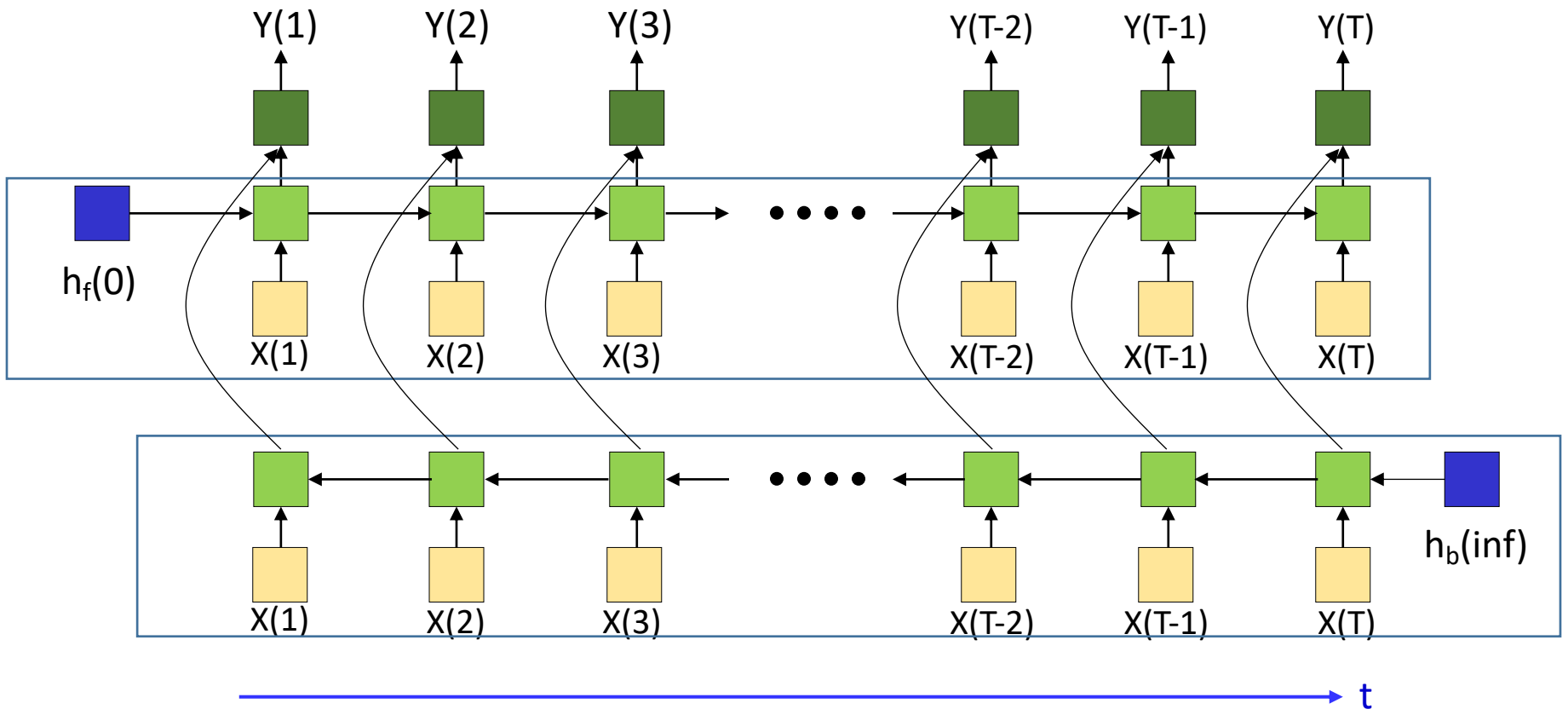
- The forward net process the data from t=1 to t=T
  - Only computing the hidden states, initially

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

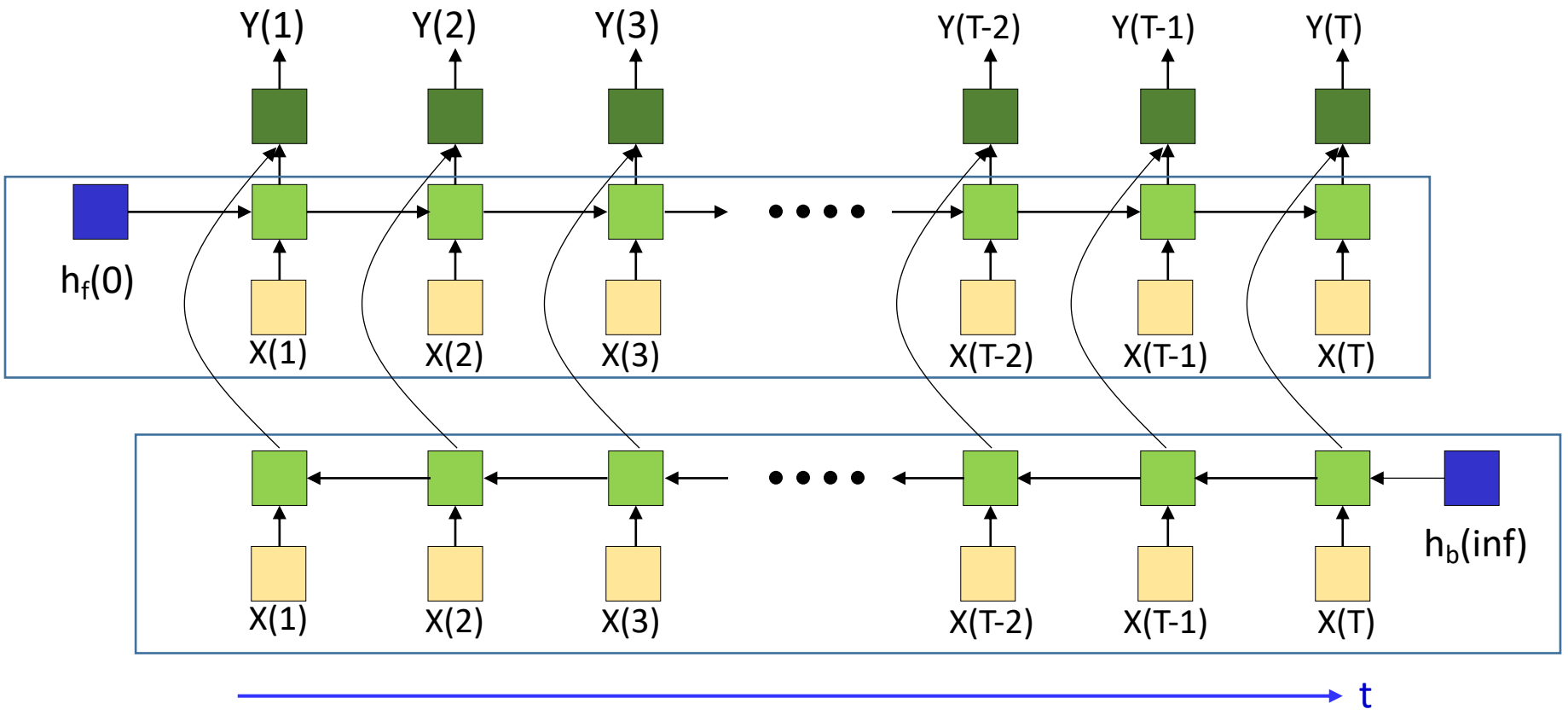
First: Define basic RNN with only hidden units

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


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

Bidirectional RNN: Assuming time-synchronous output

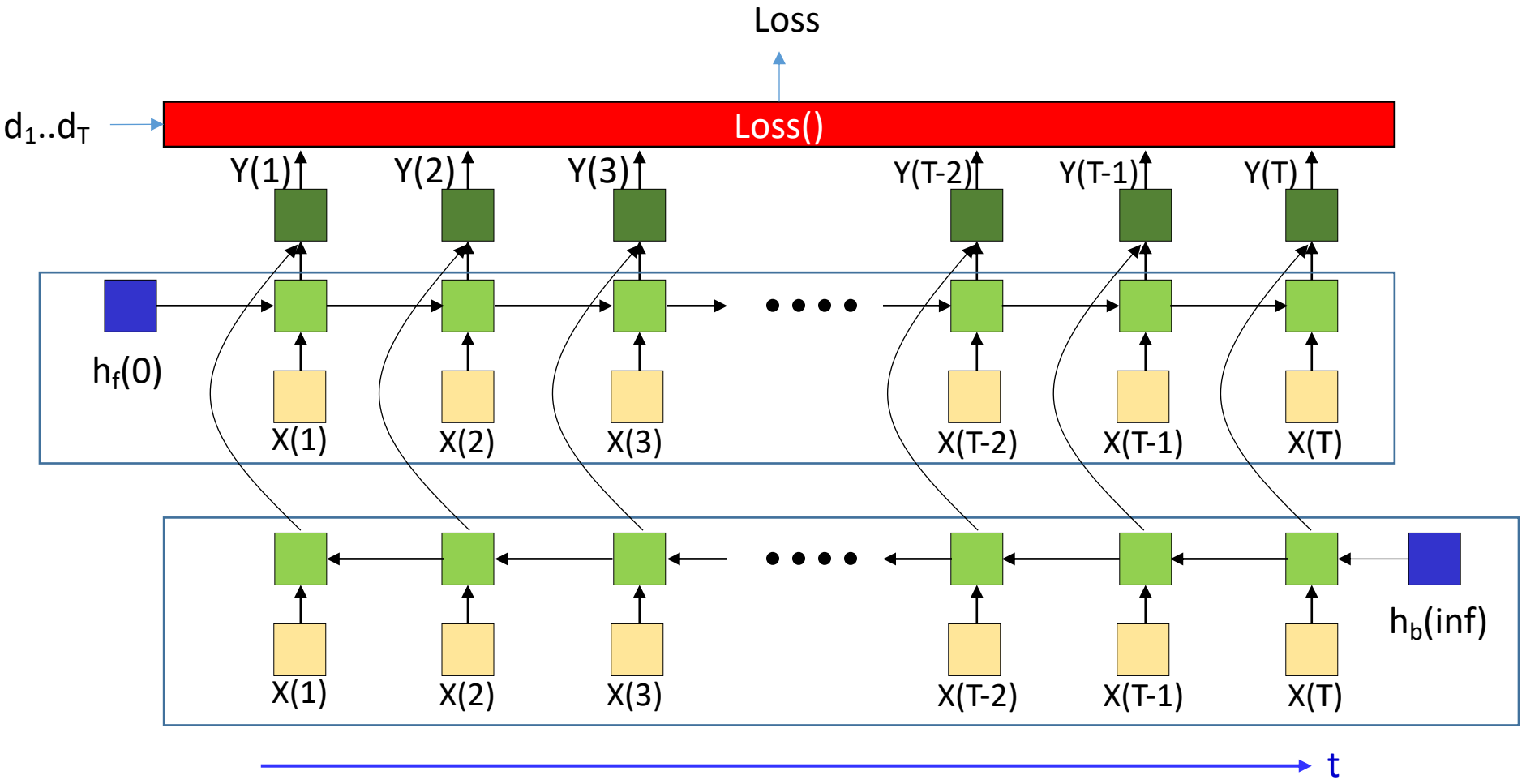
```
# Subscript f represents forward net, b is backward net
# Assuming h_f(-1,*) and h_b(inf,*) are known

#forward pass
[h_f, z_f] = RNN_forward(L_f, W_fc, W_fr, b_f, h(0,:), x, T)

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

#combine the two for the output
for t = 1:T  # The output combines forward and backward
     z_o(t) = W_fo h_f(t,L_f) +  W_bo h_b(t,L_b) + b_o
     Y(t) = softmax( z_o(t) )
```

Backpropagation in BRNNs



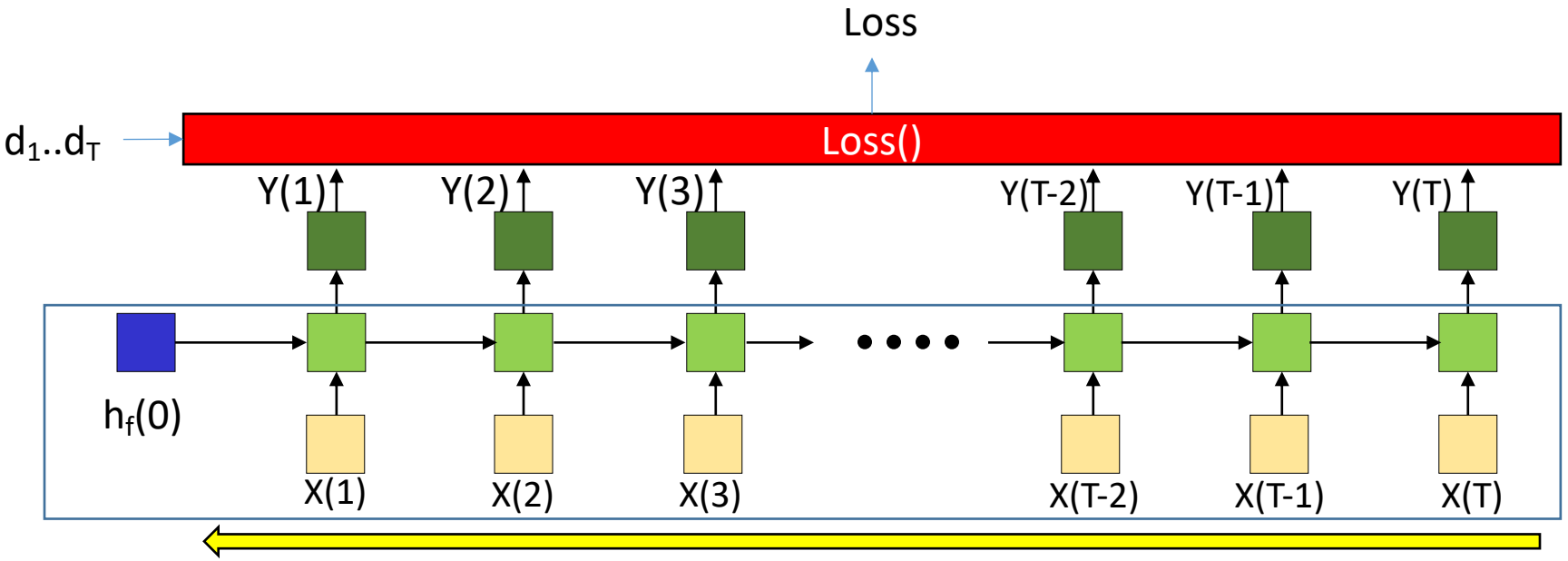
- Forward pass: Compute both forward and backward networks and final output

Backpropagation in BRNNs



- Backward pass: Define a divergence from the desired output

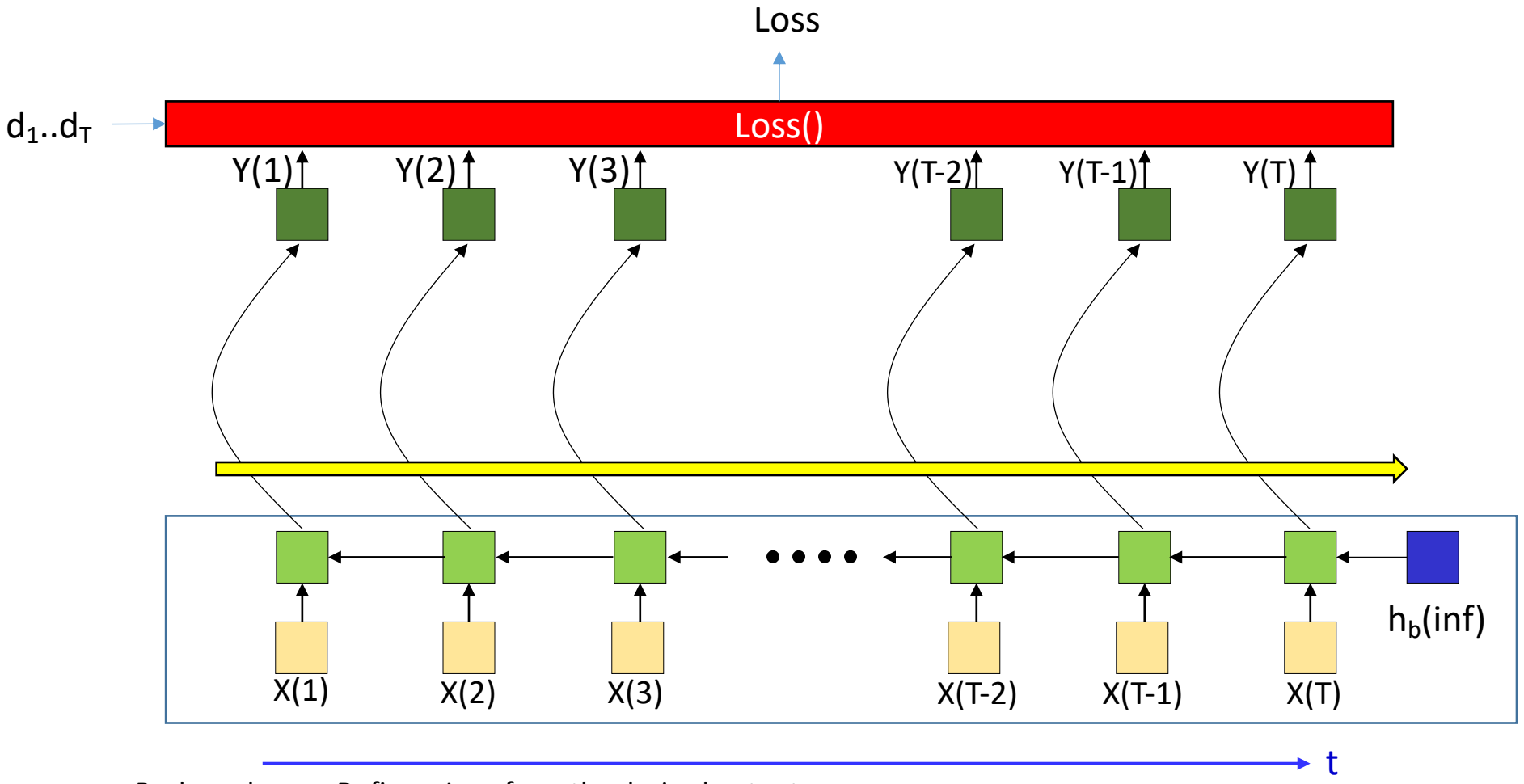
Backpropagation in BRNNs



t

- Backward pass: Define a Loss 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 Loss from the desired output
- Separately perform back propagation on both nets
  - From t=T down to t=1 for the forward net
  - **From t=1 up to t=T for the backward net**

Story so far

- Time series analysis must consider past inputs along with current input
- Recurrent networks look into the infinite past through a state-space framework
  - Hidden states that recurse on themselves
- Training recurrent networks requires
  - Defining a loss between the actual and desired output *sequences*
  - Backpropagating gradients over the entire chain of recursion
    - Backpropagation through time
  - Pooling gradients with respect to individual parameters over time
- Bidirectional networks analyze data both ways, begin→end and end→beginning to make predictions
  - In these networks, backprop must follow the chain of recursion (and gradient pooling) separately in the forward and reverse nets

Story so far

- RNNs for sequence modeling
  - Handle inputs with variable lengths
  - In theory, can track long term history
  - Share parameters across the sequence
  - Consider the order of inputs

RNNs..

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

A sequence modeling problem:

Language Modeling (predict the next word)

A generated Wikipedia page

```
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.]]
```

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

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

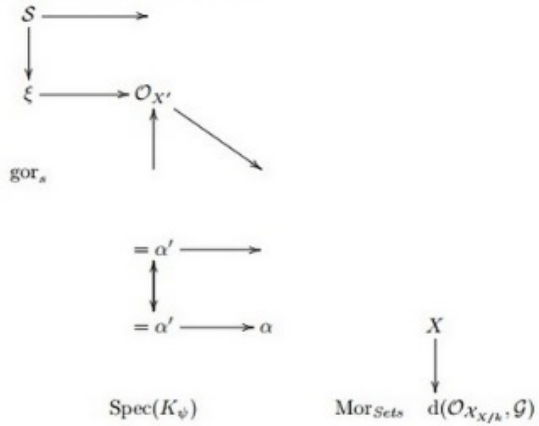
be a morphism of algebraic spaces over S and Y.

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.

The unreasonable effectiveness of RNN

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

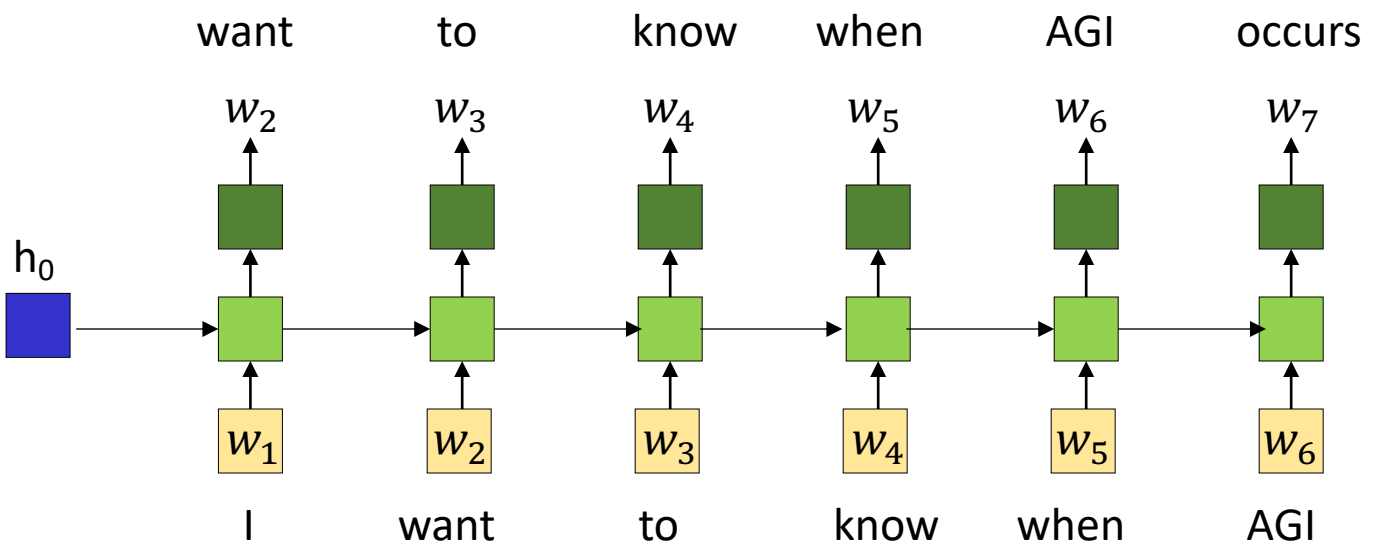
Language Models

- A language model computes a probability for a sequence of tokens: $P(w_1, \dots, w_T)$:
  - It is usually modeled by conditional distributions $P(w_i|w_1, \dots, w_{i-1})$
  - $P(w_1, \dots, w_T) = P(w_1) \prod_{i=2}^{T} P(w_i|w_1, \dots, w_{i-1})$

- Useful for machine translation, spelling correction, and …
  - Word ordering: p(the dog is big) > p(big the is dog)
  - Word choice: p(It was late afternoon.) > p(It was today.)

$P(w_i|w_1, \dots, w_{i-1})$

Simple recurrence example: Text Modelling



- Learn a model that can predict the next character given a sequence of characters
  - Or, at a higher level, words
- After observing inputs w_1 … w_k it predicts w_{k+1}

Recurrent Neural Network language model

- Given a sequence of vectors:

$$x_1, x_2, \dots, x_T$$

- At a single time step:

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

- Output:

$$\hat{y}_t = \text{softmax}\left(W_{hy} h_t\right)$$

$$P(y_t = v_j | x_1, \dots, x_t) \approx \hat{y}_{t,j}$$

Recurrent Neural Network language model

- Given a sequence of vectors:

$$x_1, x_2, \dots, x_T$$

- At a single time step:

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

- Output:

$$\hat{y}_t = \text{softmax}\left(W_{hy}h_t\right)$$

$$P(y_t = v_j | x_1, \dots, x_t) \approx \hat{y}_{t,j}$$

h_0 is some initialization vector for the hidden layer at time step 0

x_t is the column vector at time step t

Recurrent Neural Network language model: loss

- $\hat{y} \in \mathbb{R}^{|V|}$ is a probability distribution over the vocabulary
- Cross entropy loss function at location t of the sequence:

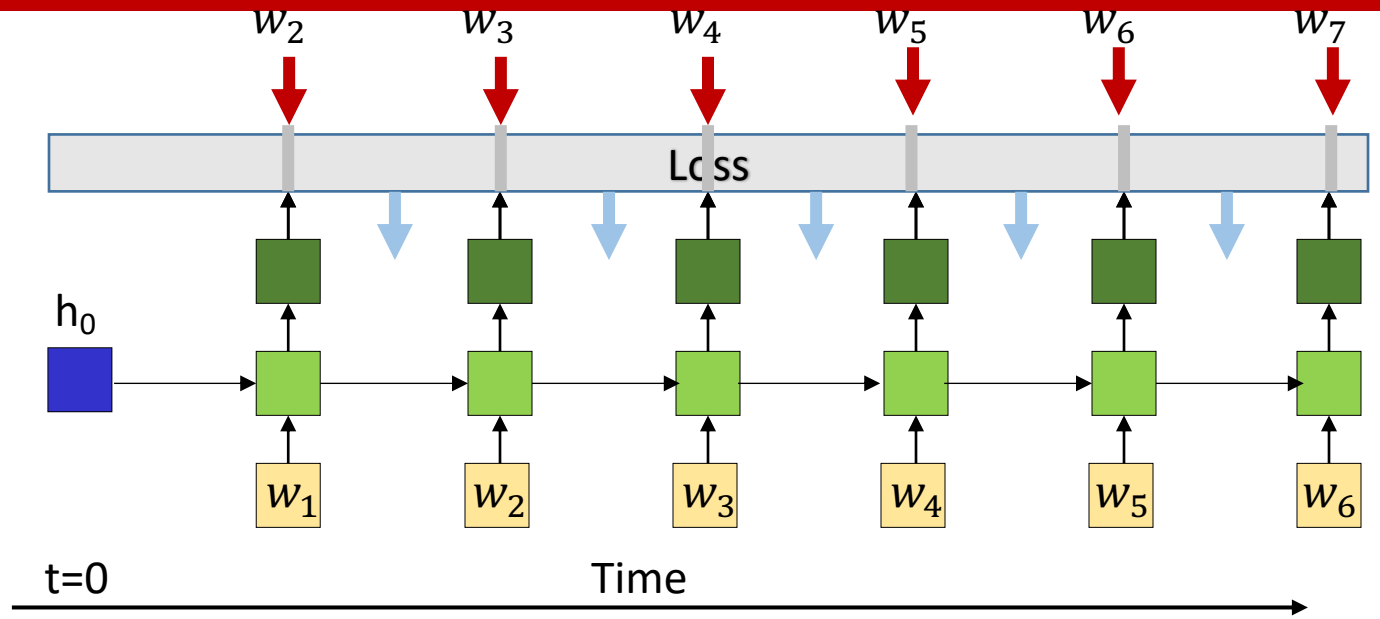
$$E_t = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

$y_{t,j} = 1$ when w_t must be the word j of vocabulary

- Cost function over the entire sequence:

$$E = -\frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

Training



- Input: symbols as one-hot vectors
  - Dimensionality of the vector is the size of the "vocabulary"
- Output: Probability distribution over symbols

$$\hat{y}(t,i) = P(V_i | w_1 \dots w_{t-1})$$

  - V_i is the i-th symbol in the vocabulary
- Divergence

The probability assigned to the correct next word

$$Loss\big(Y_{target}(1 \dots T), Y(1 \dots T)\big) = \sum_t cross\_entropy\big(Y_{target}(t), Y(t)\big) = -\sum_t \log \hat{y}(t, w_{t+1})$$

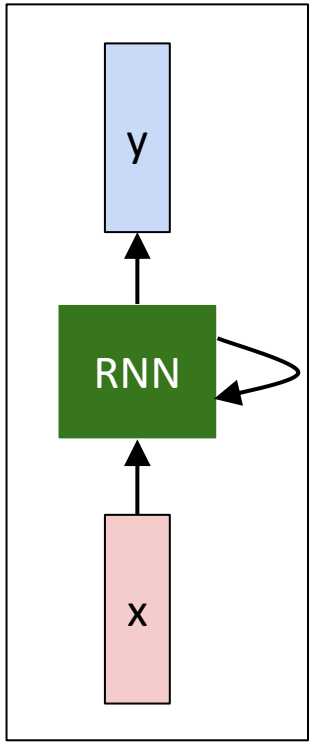
Character-level language model example

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

Input presented as one-hot vectors

Output: probability distribution over characters

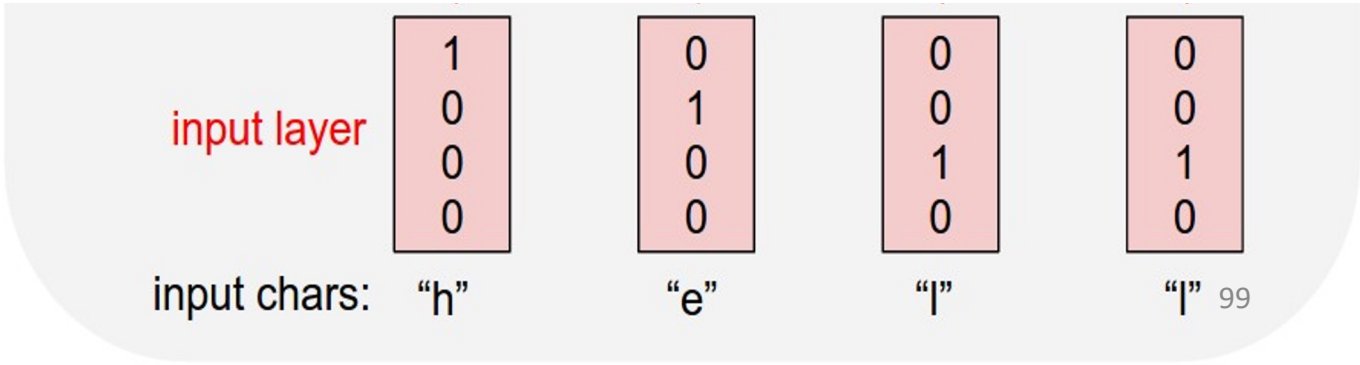
Must ideally peak at the target character



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

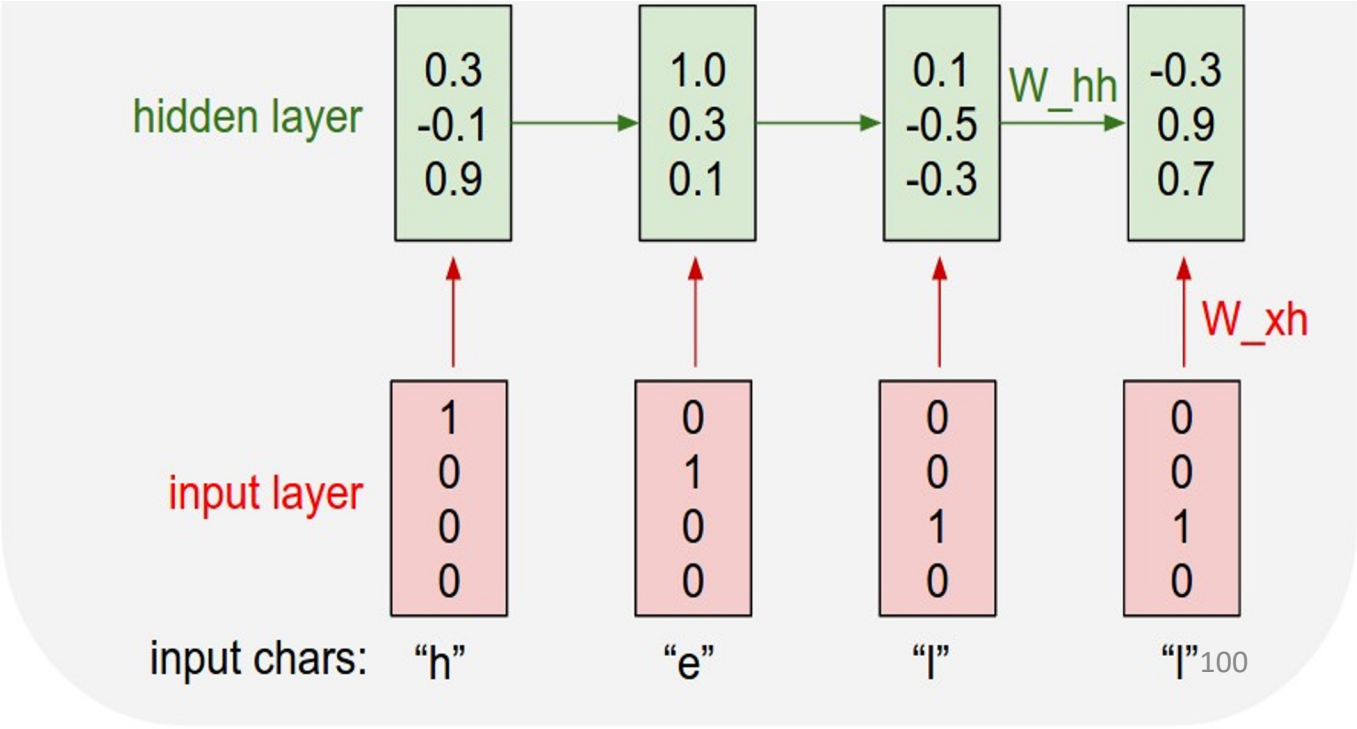


Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

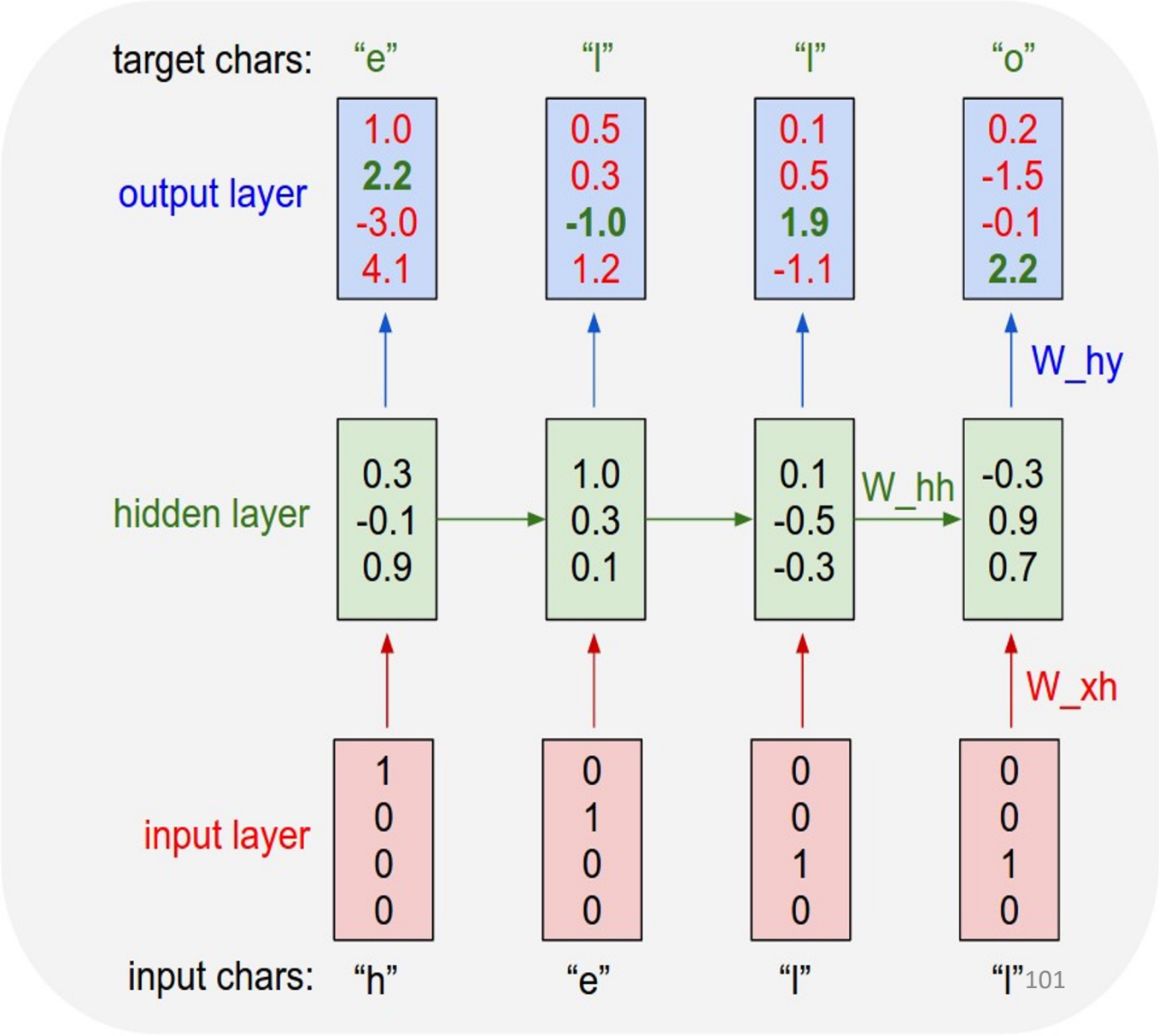
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Character-level language model example

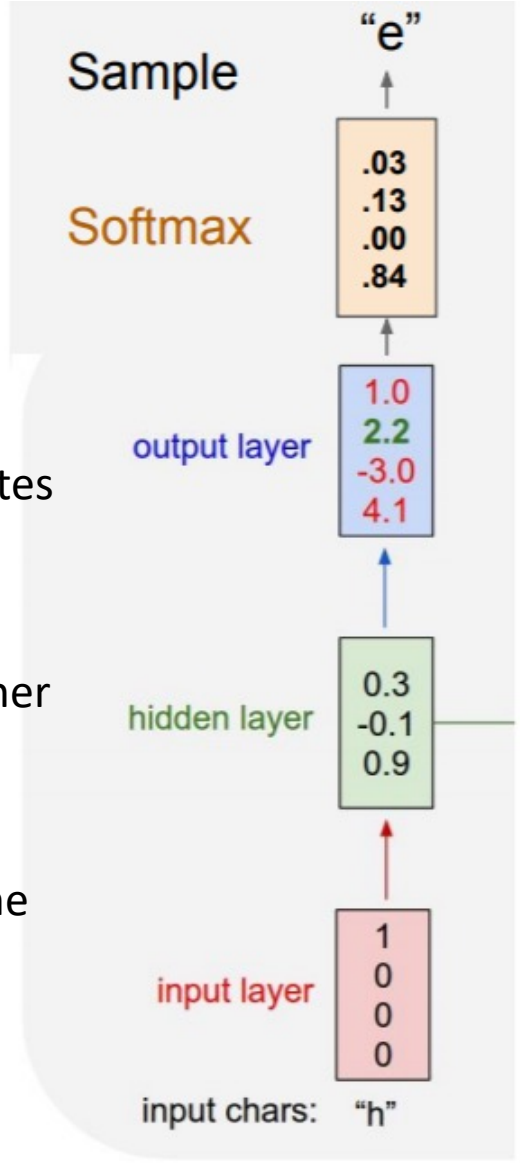
Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



Example: Character-level Language Model Sampling

- Vocabulary: [h,e,l,o]
- At test-time sample characters one at a time, feed back to model



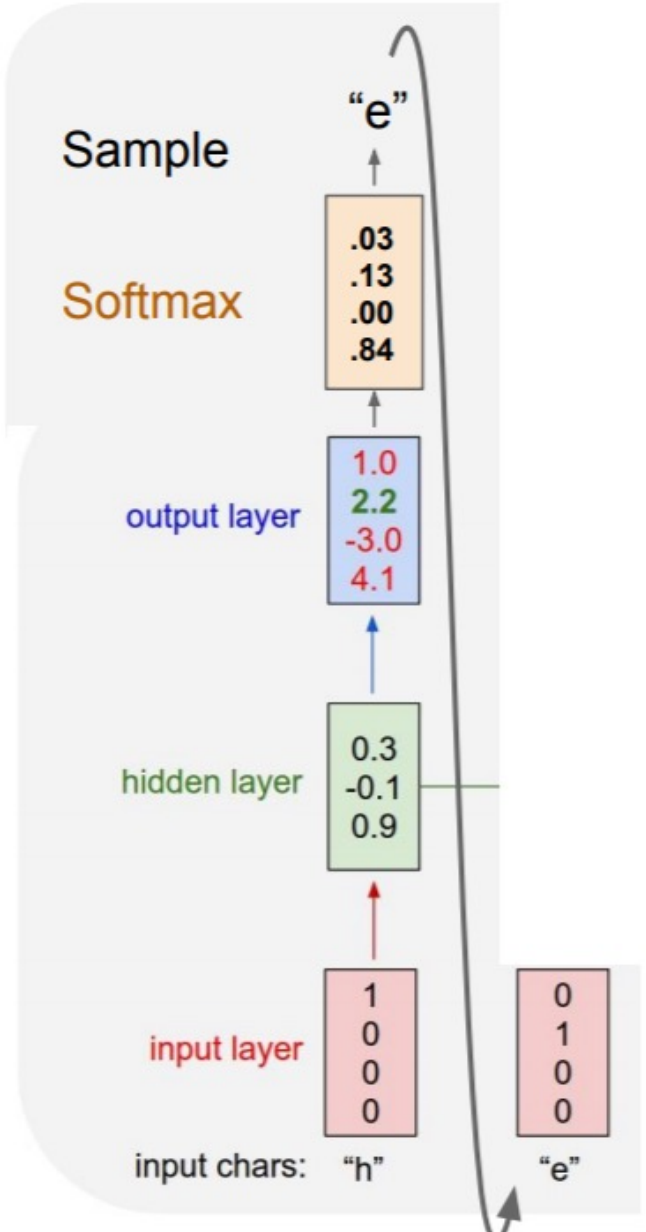
After feeding the first input, the network generates a probability distribution over vocabulary

Outputs an N-valued probability distribution rather than a one-hot vector

Draw a char from the distribution and set it as the next one in the series

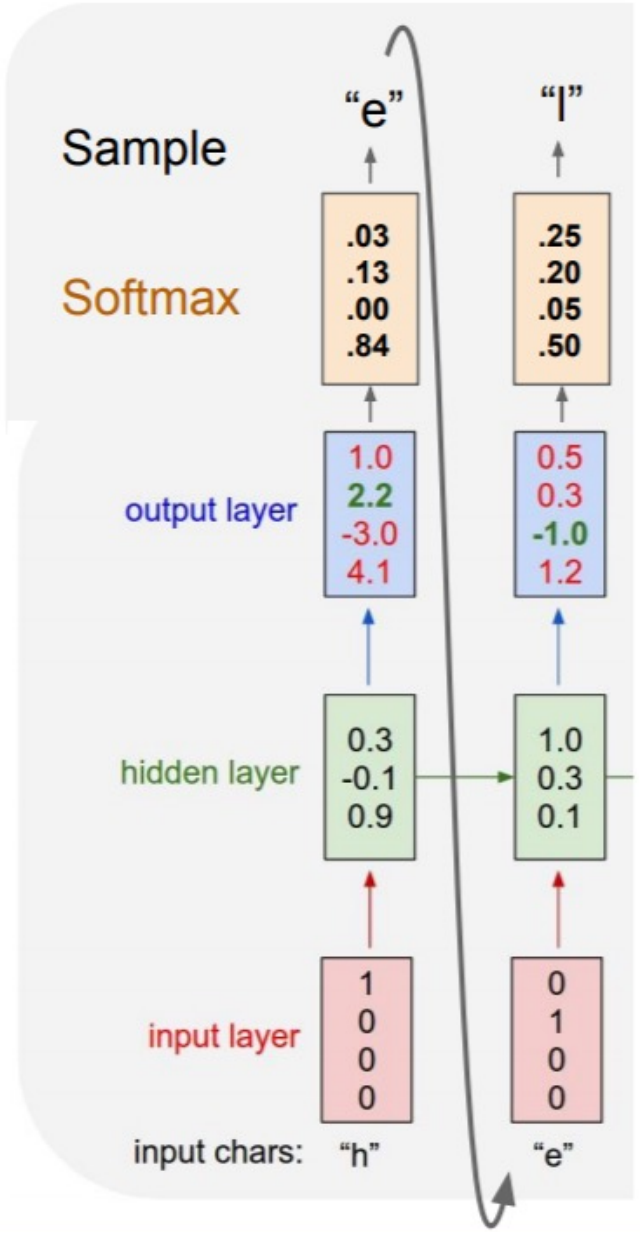
Example: Character-level Language Model Sampling

- Vocabulary: [h,e,l,o]
- At test-time sample characters one at a time, feed back to model



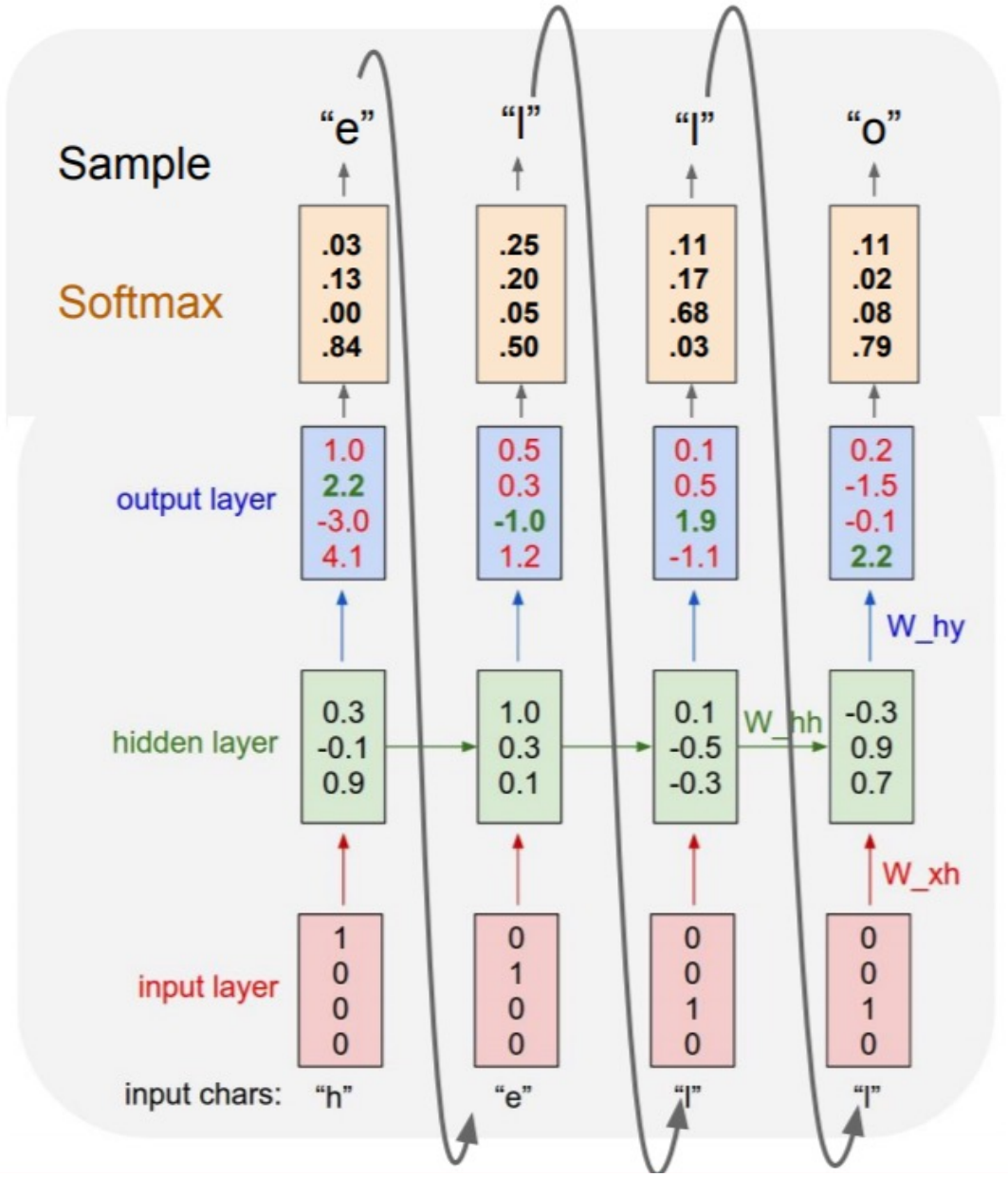
Example: Character-level Language Model Sampling

- Vocabulary: [h,e,l,o]
- At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

- Vocabulary: [h,e,l,o]
- At test-time sample characters one at a time, feed back to model



min-char-rnn.py gist:

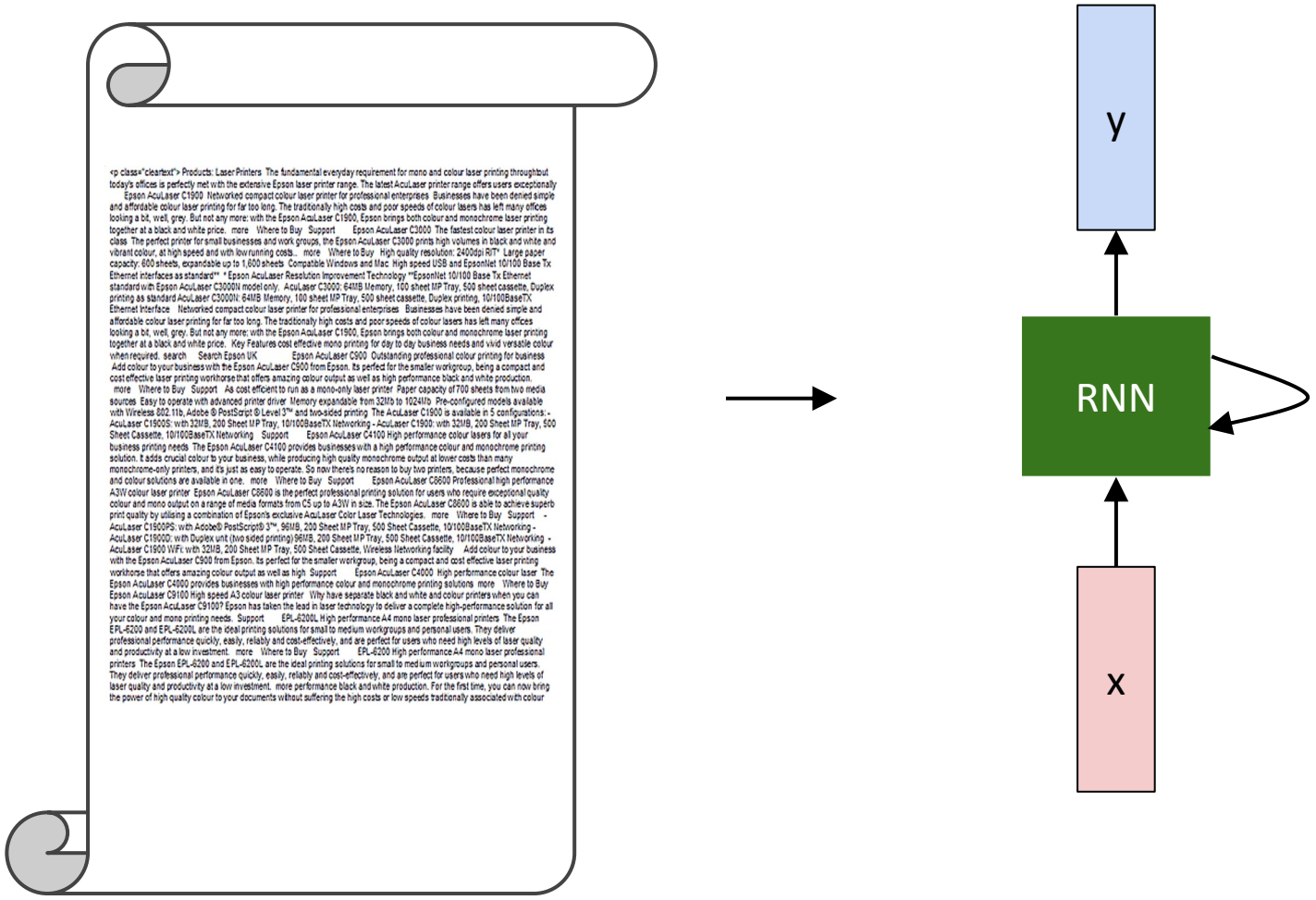
112 lines of Python

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

```python
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

(https://gist.github.com/karpathy/d4dee566867f8291f086)

Language Modeling: Example I



Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no! it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

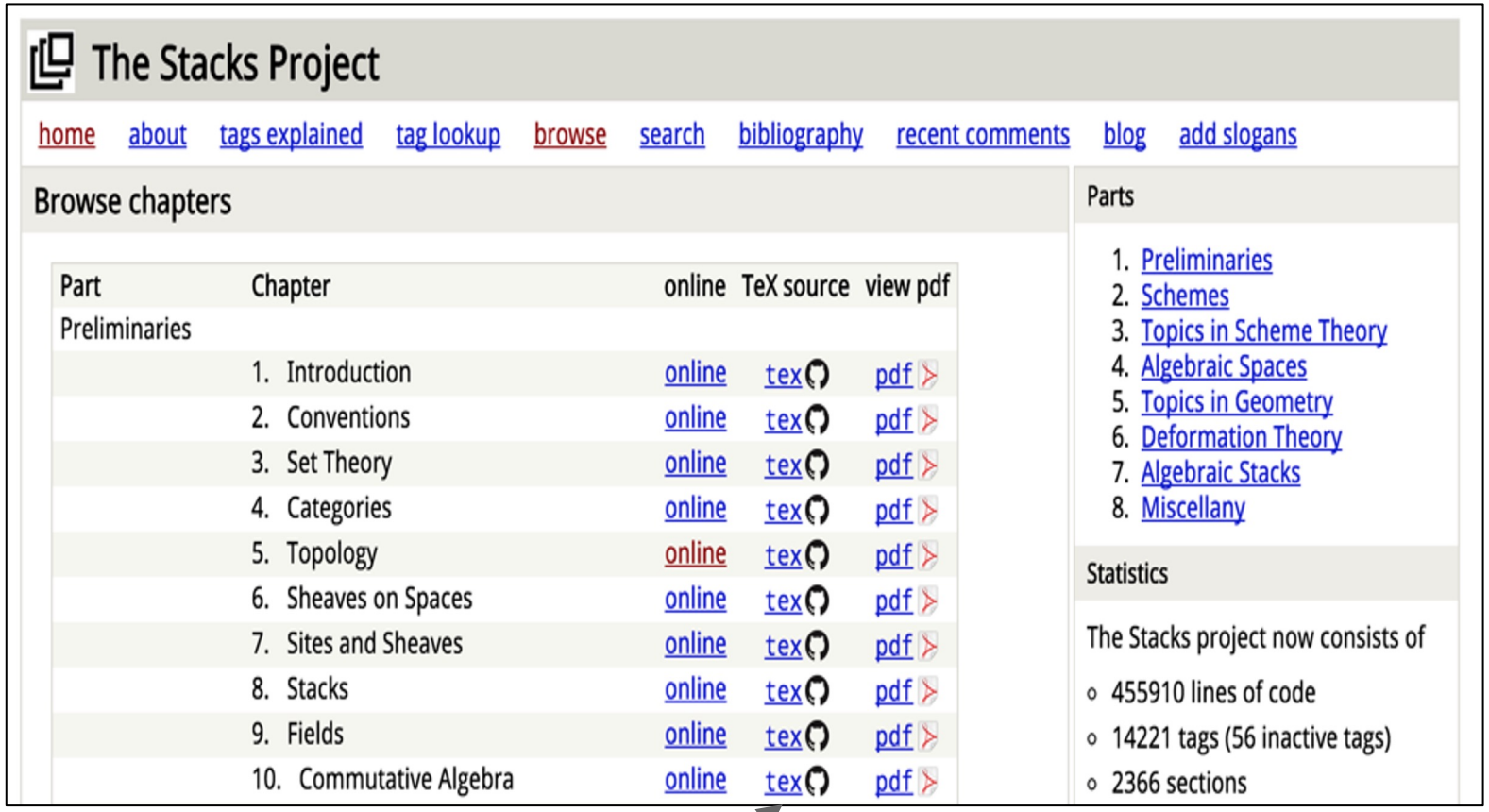
Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.



Latex source

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciy in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M^\bullet} = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

Lemma 0.3. *In Situation* ??. *Hence we may assume* $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

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

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

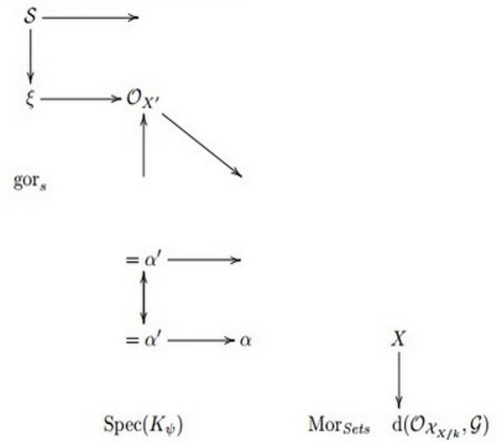
be a morphism of algebraic spaces over S and Y.

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.

□

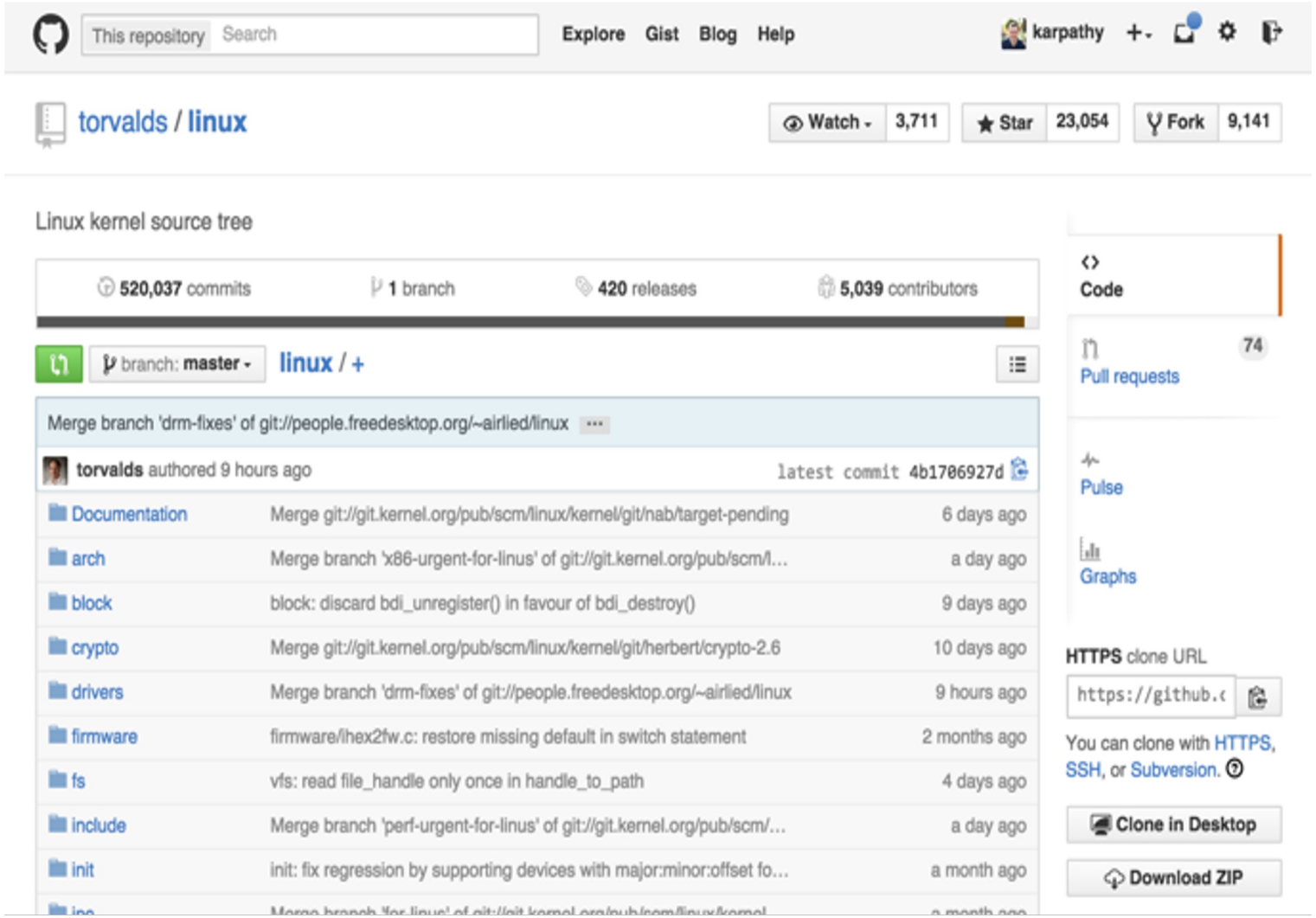
Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

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

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

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.


This repository Search
Explore Gist Blog Help
karpathy
torvalds / linux
Watch 3,711
Star 23,054
Fork 9,141
Linux kernel source tree
520,037 commits
1 branch
420 releases
5,039 contributors
branch: master
linux / +
Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux
torvalds authored 9 hours ago
latest commit 4b1706927d
Documentation Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending 6 days ago
arch Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l... a day ago
block block: discard bdi_unregister() in favour of bdi_destroy() 9 days ago
crypto Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6 10 days ago
drivers Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux 9 hours ago
firmware firmware/ihex2fw.c: restore missing default in switch statement 2 months ago
fs vfs: read file_handle only once in handle_to_path 4 days ago
include Merge branch 'perf-urgent-for-linus' of git://git.kernel.org/pub/scm/... a day ago
init init: fix regression by supporting devices with major:minor:offset fo... a month ago
Code
74
Pull requests
Pulse
Graphs
HTTPS clone URL
https://github.c
You can clone with HTTPS, SSH, or Subversion.
Clone in Desktop
Download ZIP

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000ffffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Generated C code

```
/*
 *  Copyright (c) 2006-2010, Intel Mobile Communications.  All rights reserved.
 *
 *   This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 *        This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software Foundation,
 *  Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>
```

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)     (func)

#define SWAP_ALLOCATE(nr)     (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0));   \
  if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
          pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
  PUT_PARAM_RAID(2, sel) = get_state_state();
  set_pid_sum((unsigned long)state, current_state_str(),
           (unsigned long)-1->lr_full; low;
}
```

OpenAI Codex

translates natural language to code

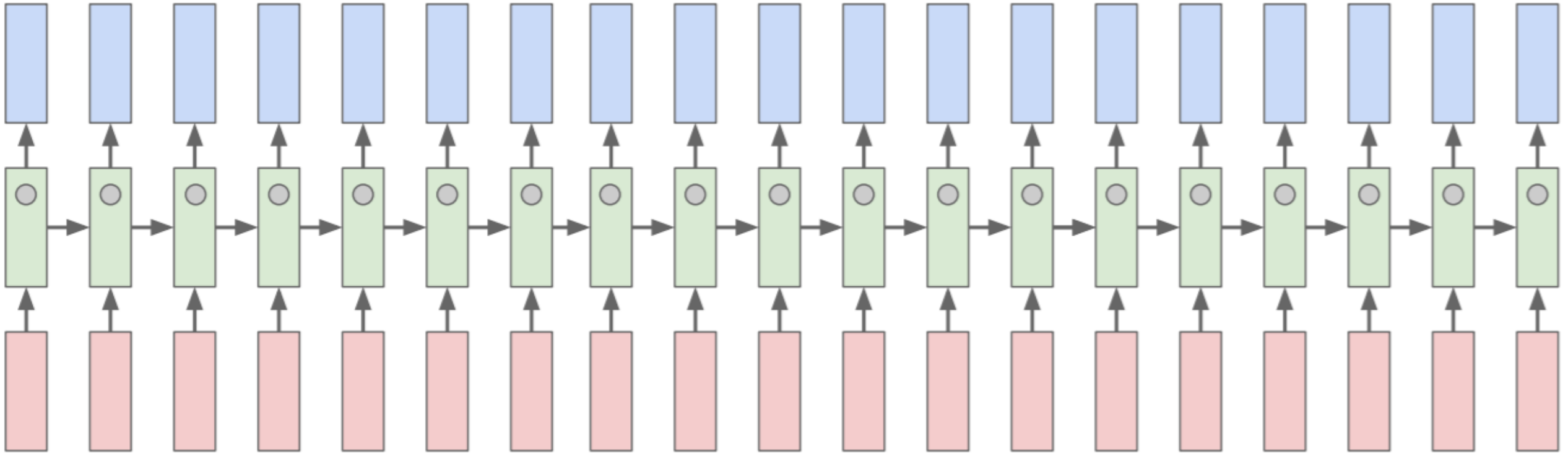

```
/* Add this image of a
rocketship:
https://i1.sndcdn.com/artworks
-j8xjG7zc1wmTeO7b-O6l83w-
t500x500.jpg */
var rocketship =
document.createElement('img');
rocketship.src =
'https://i1.sndcdn.com/artwork
s-j8xjG7zc1wmTeO7b-O6l83w-
t500x500.jpg';
document.body.appendChild(rock
etship);
```

Add this image of a rocketship:
https://i1.sndcdn.com/artworks-j8xjG7zc1wmTeO7b-O6l83w-t500x500.jpg

Codex is the model that powers GitHub Copilot

https://openai.com/blog/openai-codex/

Searching for interpretable cells



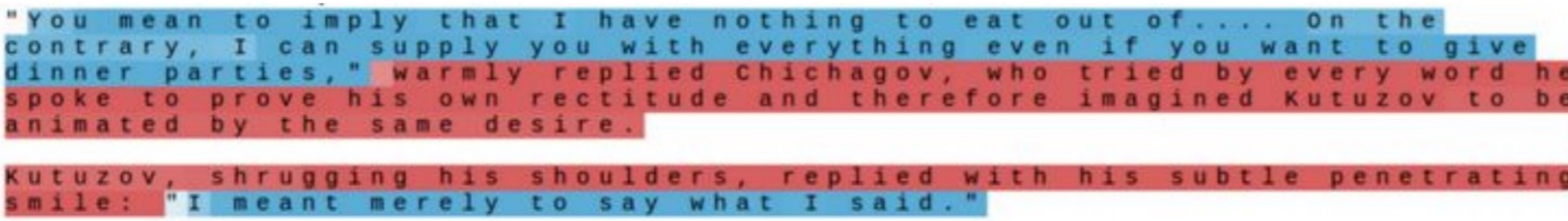
Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
```

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells

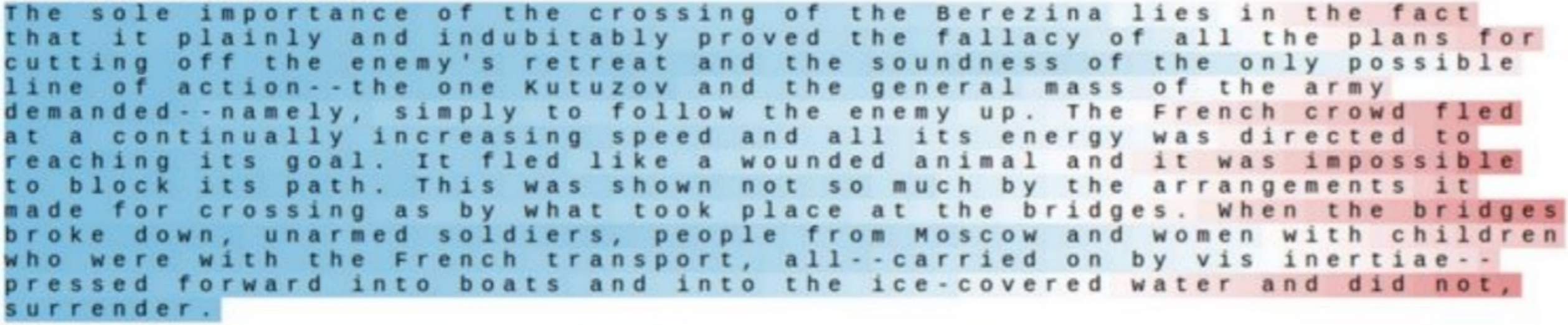


quote detection cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells

Cell sensitive to position in line:



line length tracking cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

if statement cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells

Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                       struct audit_field *sf)
{
  int ret = 0;
  char *lsm_str;
  /* our own copy of lsm_str */
  lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
  if (unlikely(!lsm_str))
    return -ENOMEM;
  df->lsm_str = lsm_str;
  /* our own (refreshed) copy of lsm_rule */
  ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                 (void **)&df->lsm_rule);
  /* Keep currently invalid fields around in case they
   * become valid after a policy reload. */
  if (ret == -EINVAL) {
    pr_warn("audit rule for LSM \'%s\' is invalid\n",
            df->lsm_str);
    ret = 0;
  }
  return ret;
}
```

quote/comment cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

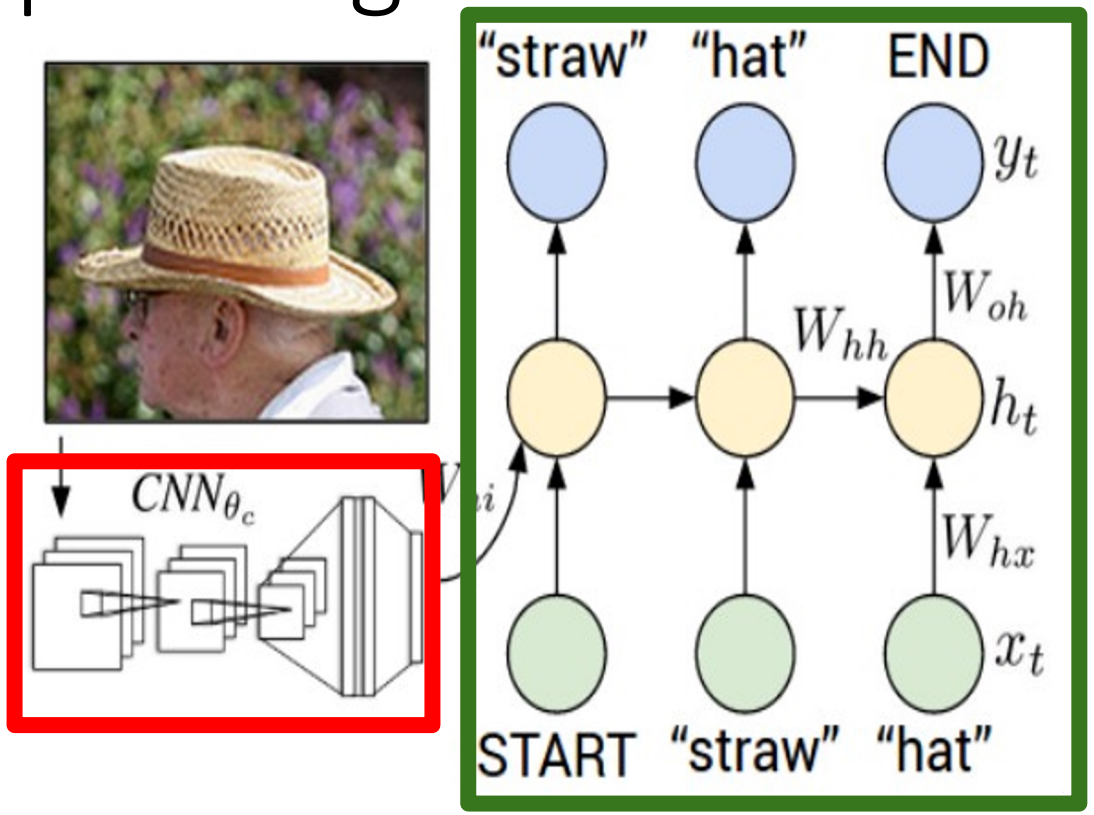
code depth cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

RNN tradeoffs

- RNN Advantages:
  - Can process any length input
  - Computation for step t can (in theory) use information from many steps back
  - Model size doesn't increase for longer input
  - Same weights applied on every timestep, so there is symmetry in how inputs are processed.
- RNN Disadvantages:
  - Recurrent computation is slow
  - In practice, difficult to access information from many steps back

Image Captioning



Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei, CVPR 2016.

test image

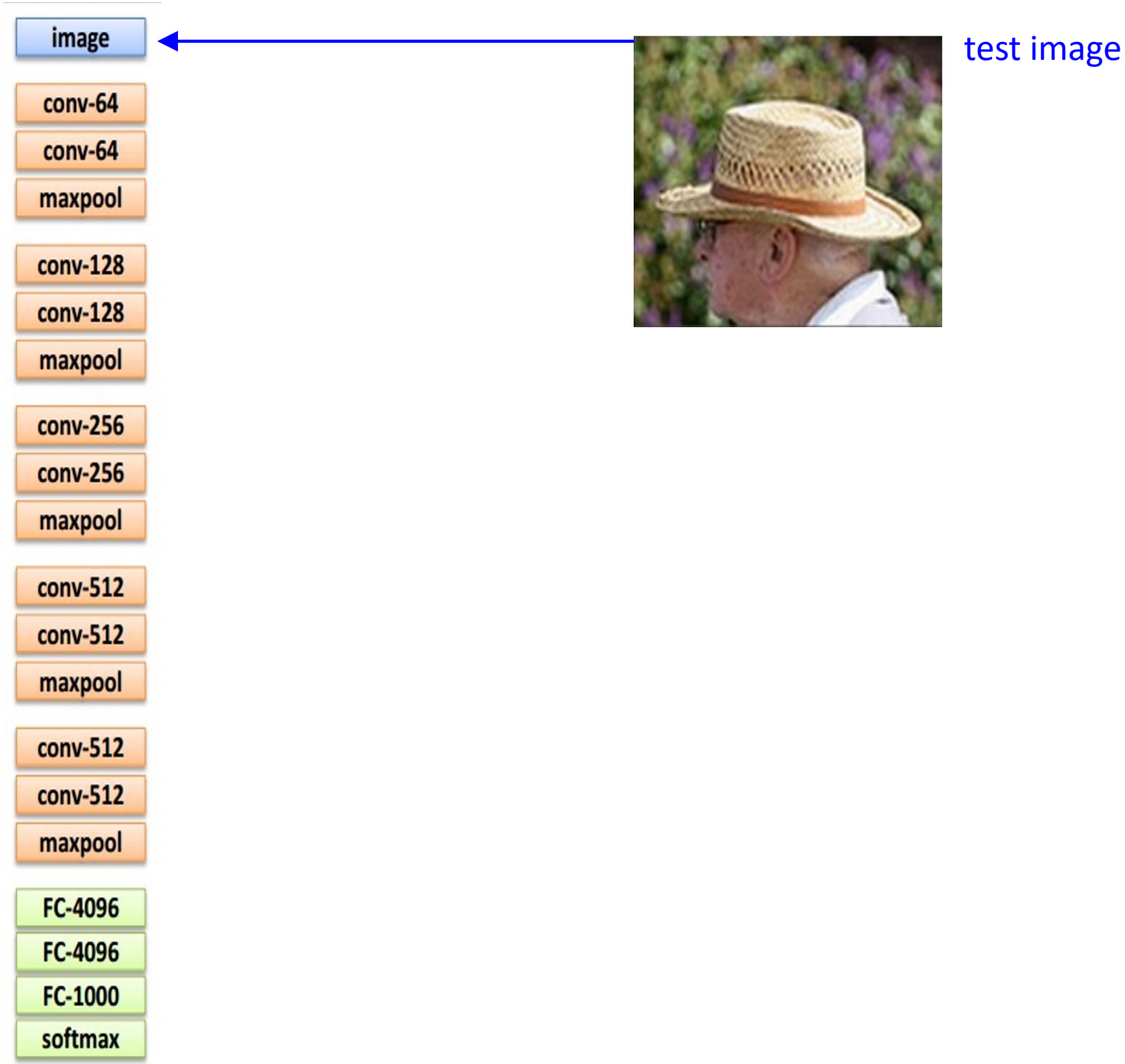


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax
test image

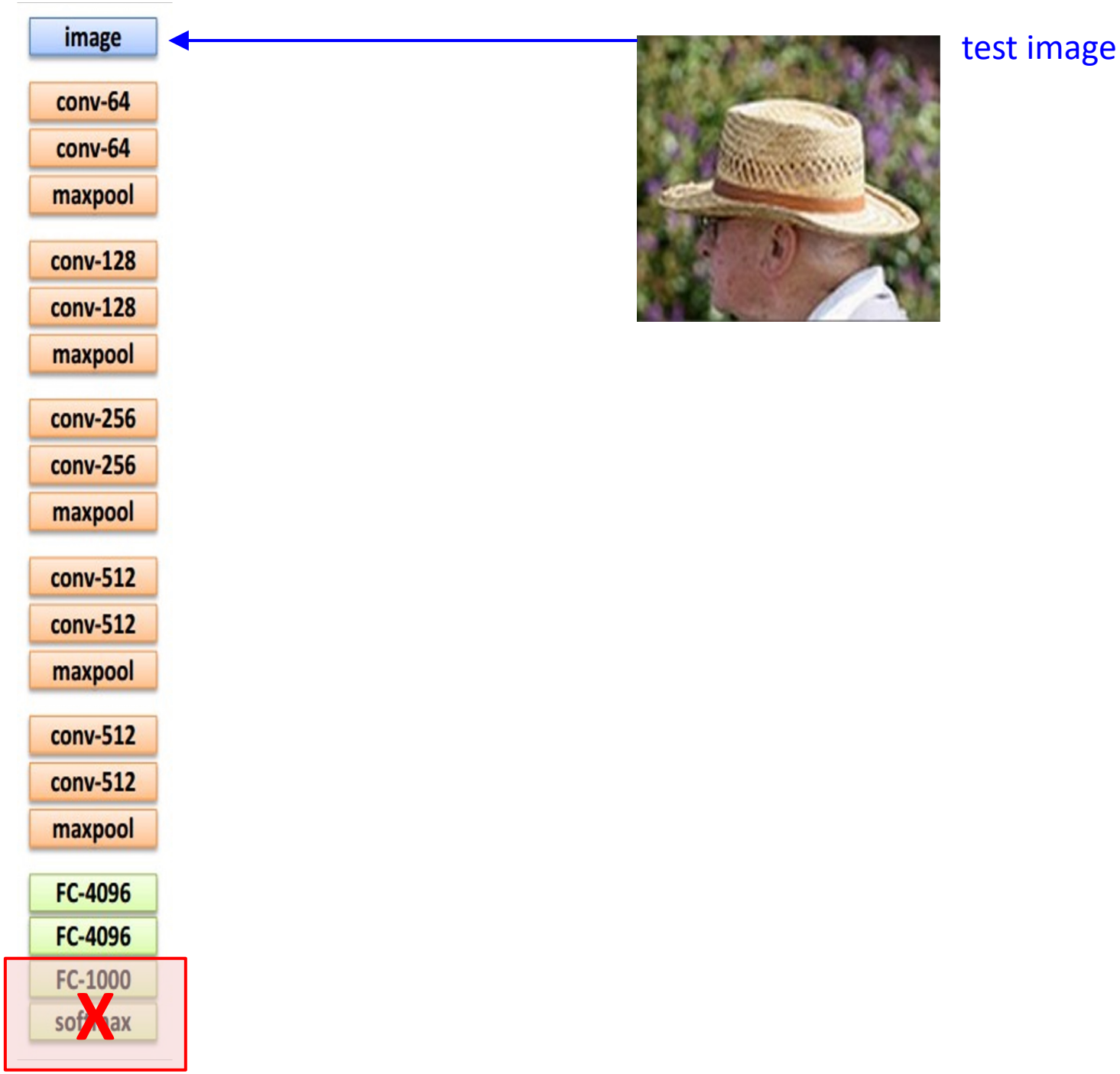


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax
X
test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

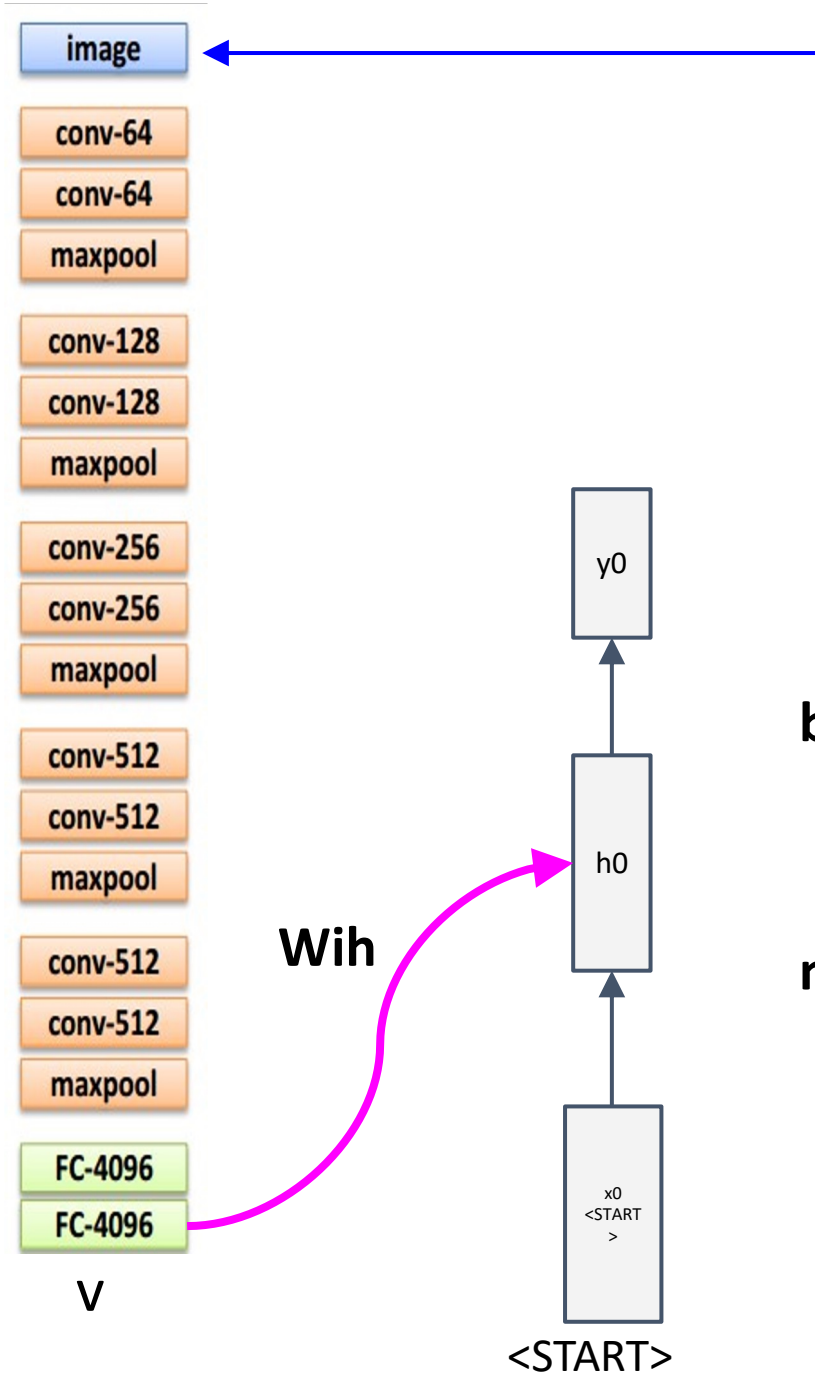
FC-4096

FC-4096

test image

x0
<START
>

<START>



test image

before:

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

now:

$$h = \tanh(W_{xh} * x + W_{hh} * h + \boldsymbol{W_{ih}} * \boldsymbol{v})$$

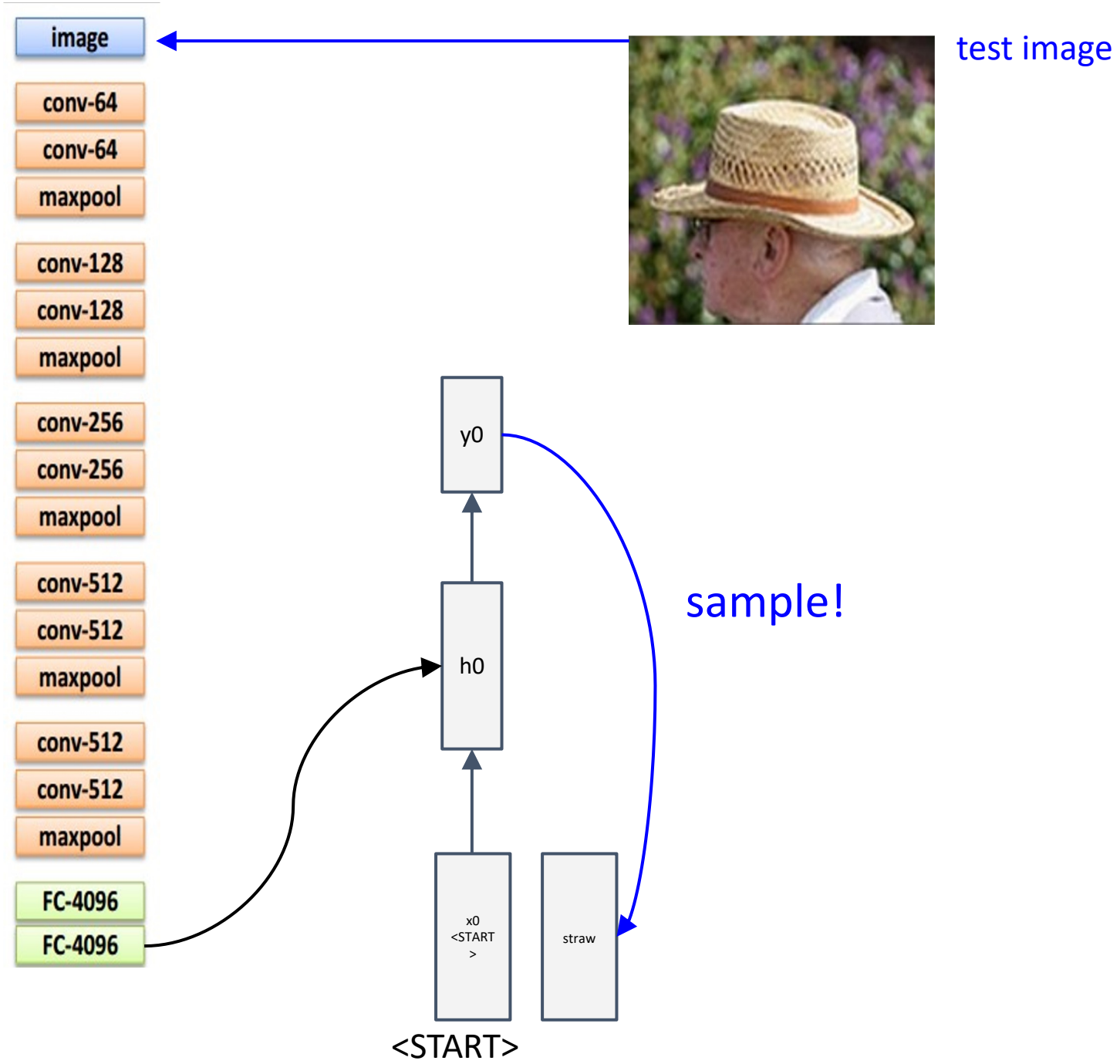


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
h0
x0
<START
>
straw
<START>
sample!

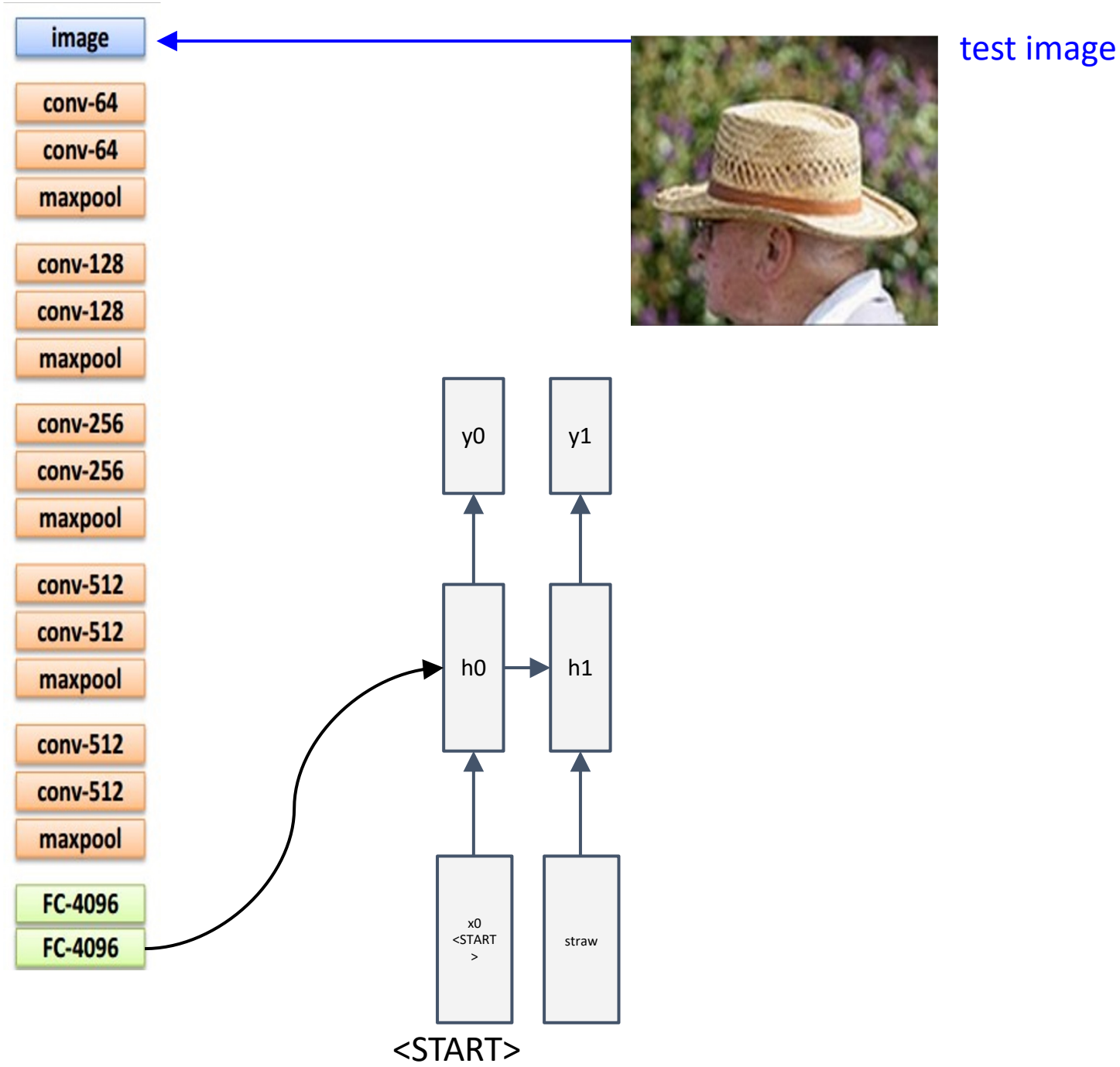


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
x0
<START
>
straw
<START>

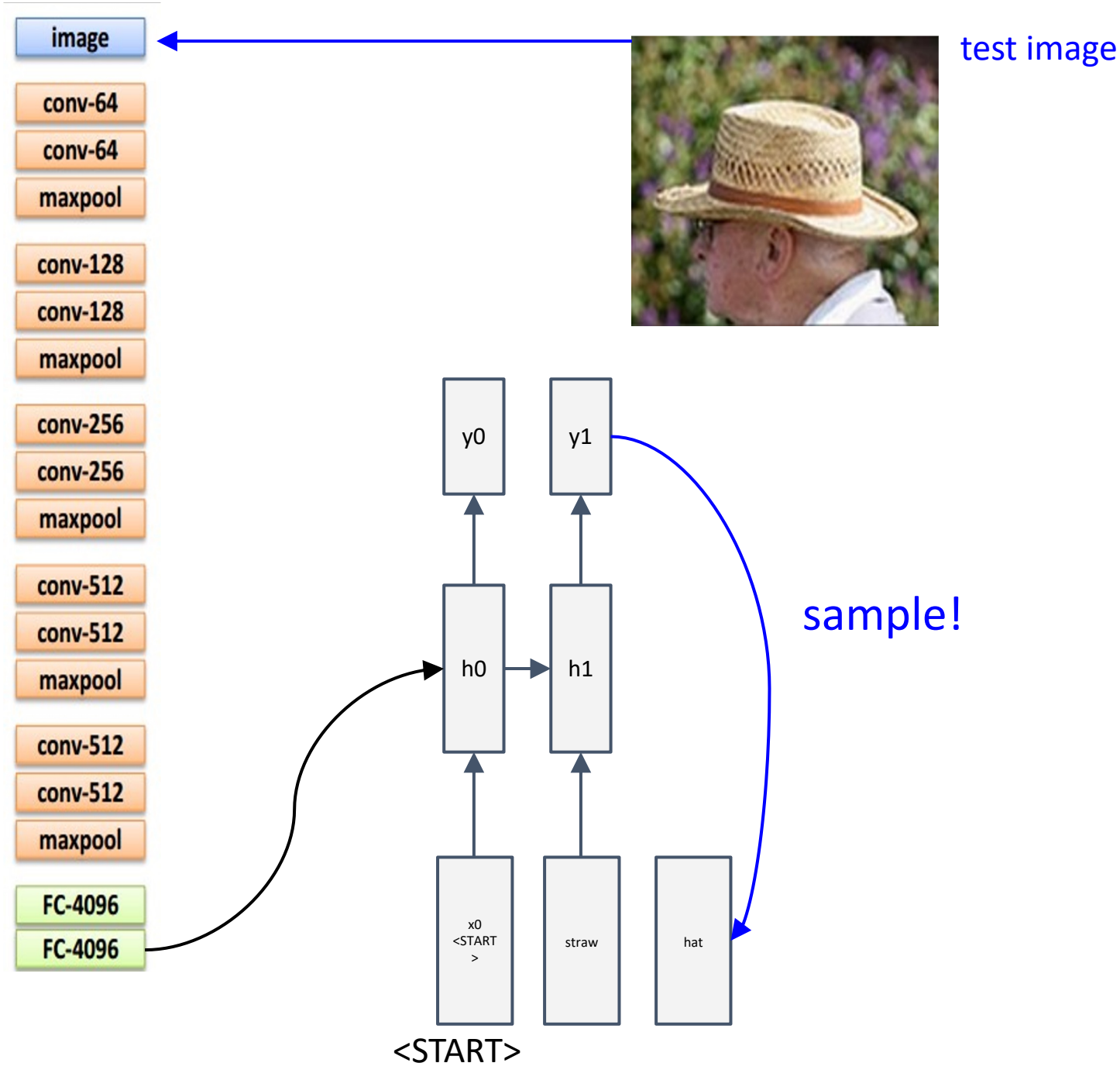


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
sample!
x0
<START
>
straw
hat
<START>

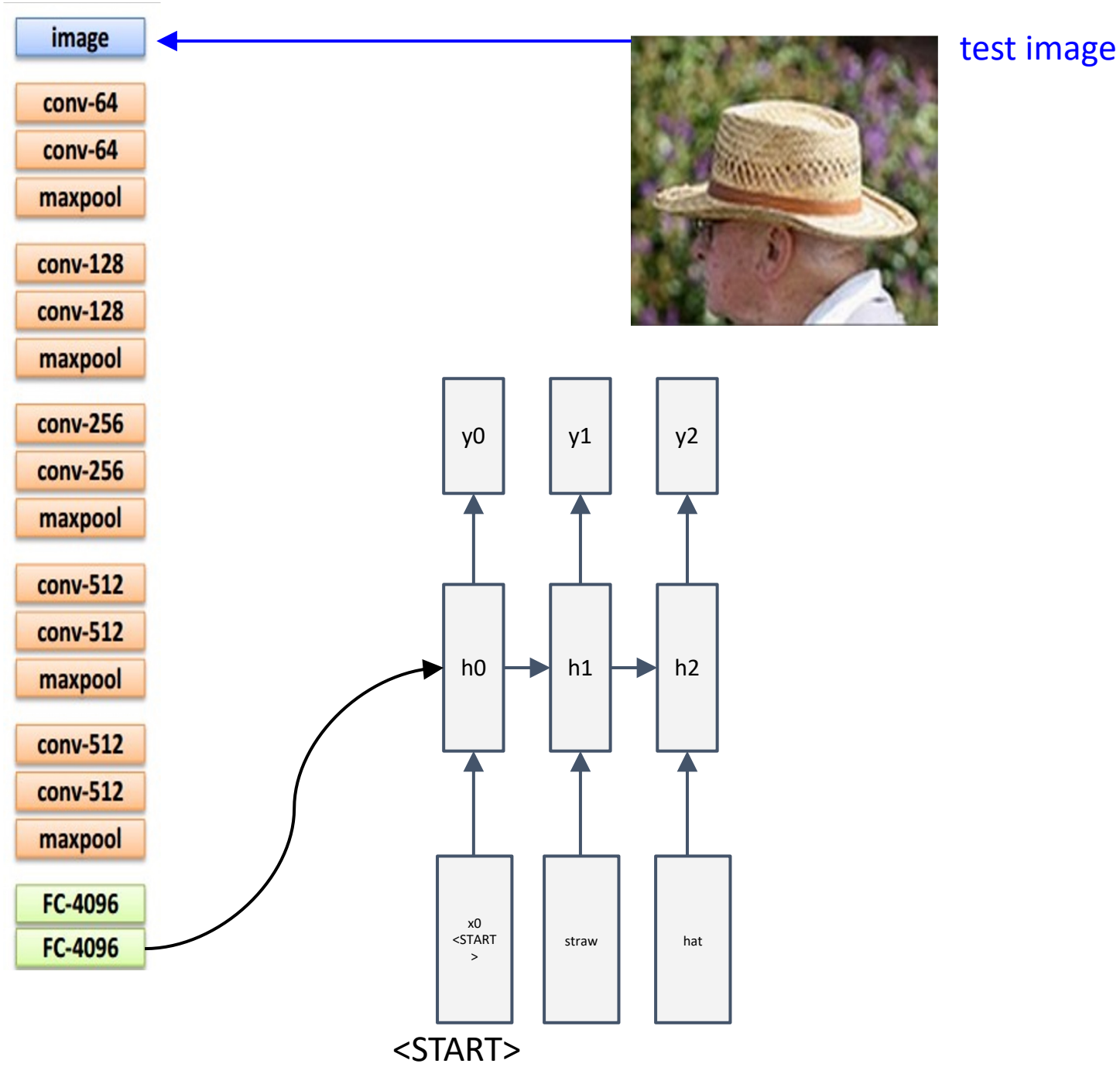


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0
<START
>
straw
hat
<START>

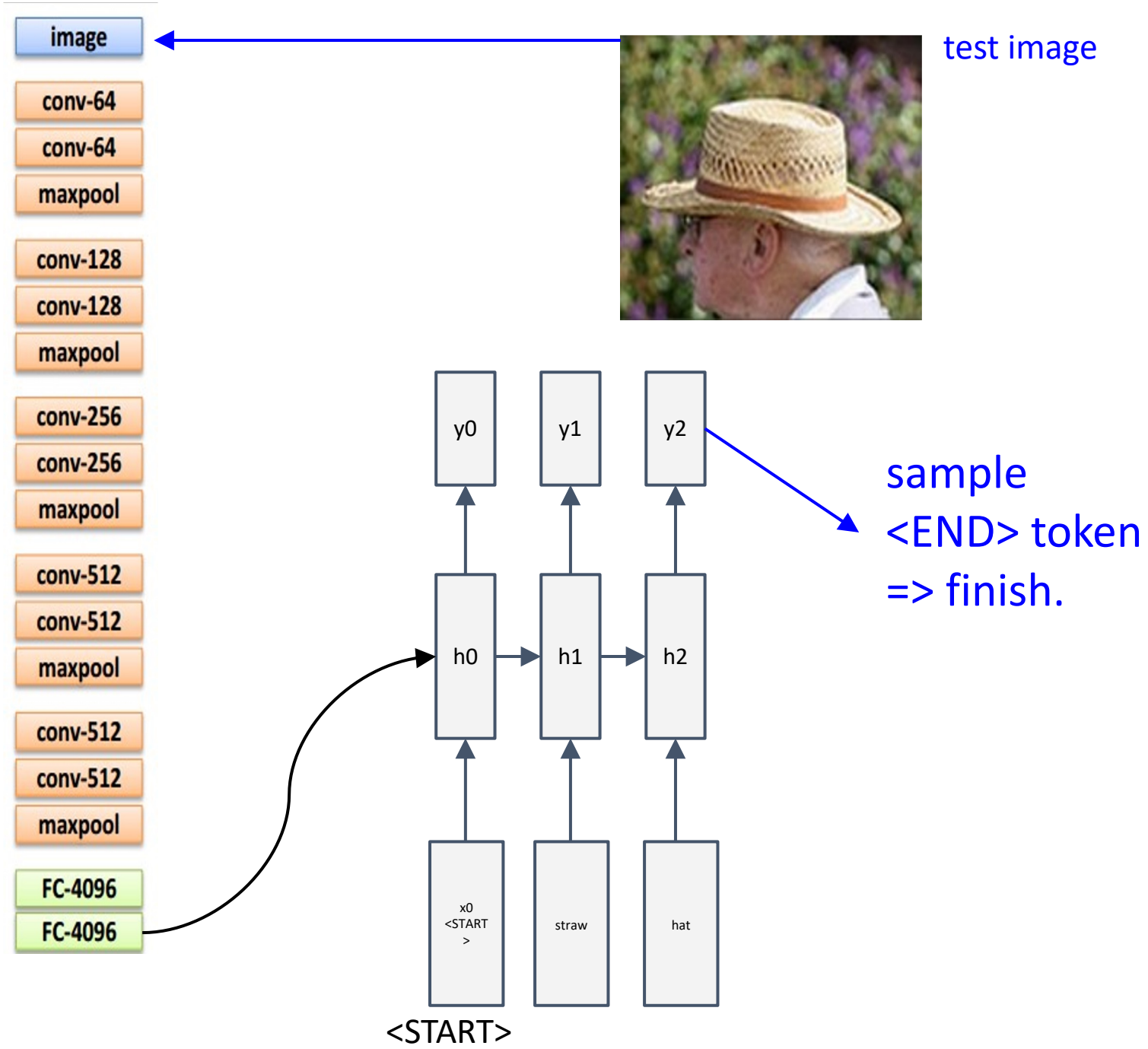


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0
<START
>
straw
hat
<START>
sample
<END> token
=> finish.

Image Sentence Datasets

a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.

Microsoft COCO

[Tsung-Yi Lin et al. 2014]

mscoco.org

currently:

~120K images

~5 sentences each

Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain:
cat suitcase, cat tree, dog, bear,
surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

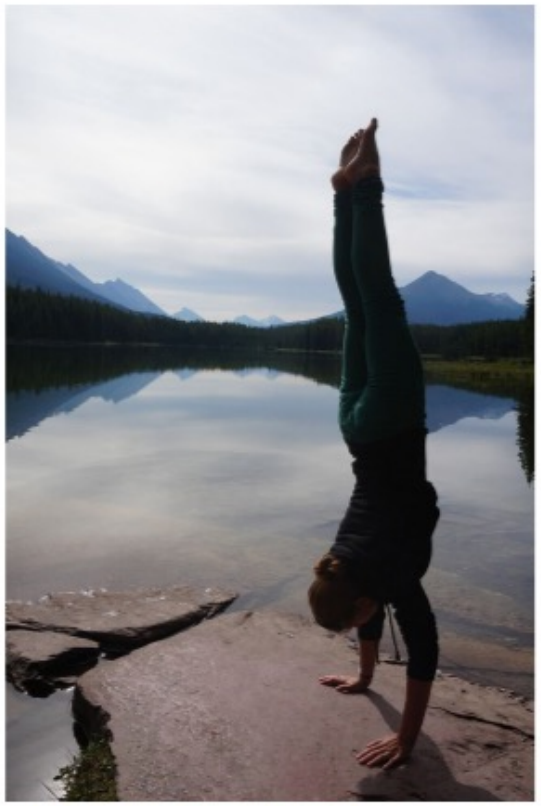
A man riding a dirt bike on a dirt track

Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei, CVPR 2016.

Image Captioning: Failure Cases

Captions generated using neuraltalk2
All images are CC0 Public domain:
cat suitcase, cat tree, dog, bear,
surfers, tennis, giraffe, motorcycle

A woman is holding a cat in her hand



A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A person holding a computer mouse on a desk

A man in a baseball uniform throwing a ball

Visual Question Answering (VQA)

Q: What endangered animal is featured on the truck?

A: **A bald eagle.**
A: A sparrow.
A: A humming bird.
A: A raven.

Q: Where will the driver go if turning right?

A: **Onto 24 ¾ Rd.**
A: Onto 25 ¾ Rd.
A: Onto 23 ¾ Rd.
A: Onto Main Street.

Q: When was the picture taken?

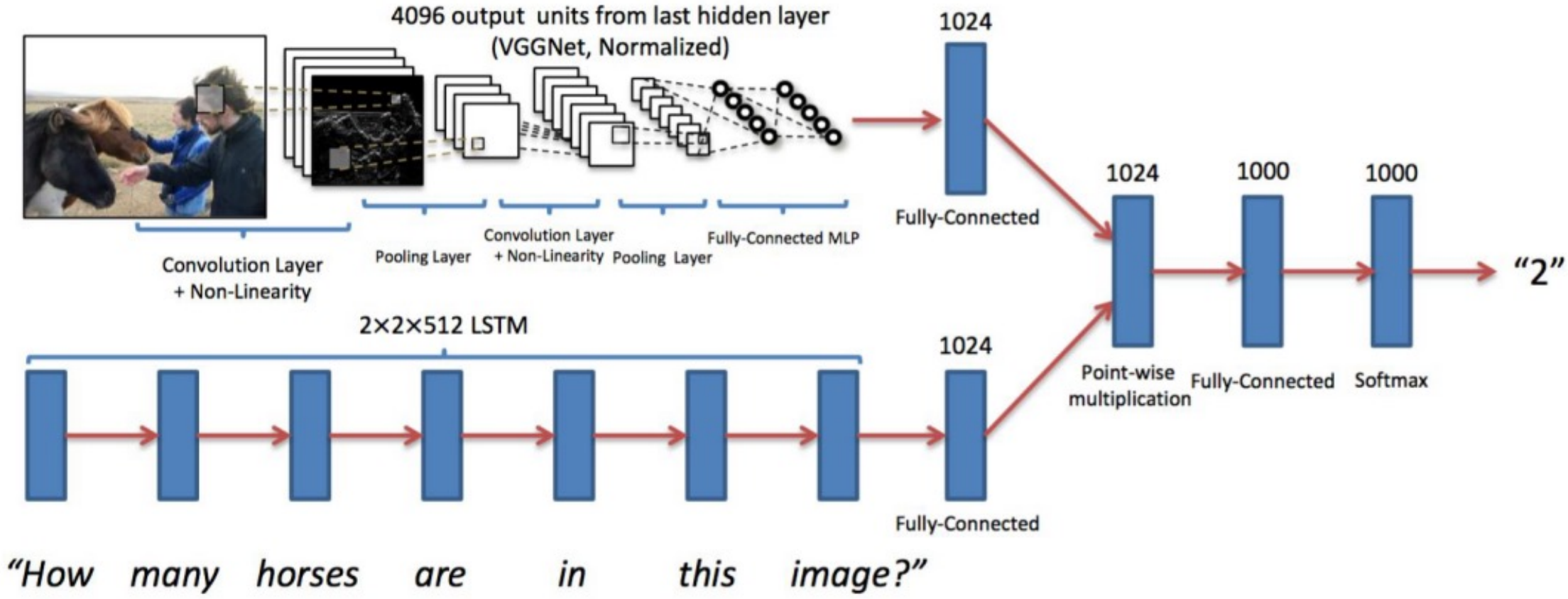
A: **During a wedding.**
A: During a bar mitzvah.
A: During a funeral.
A: During a Sunday church service

Q: Who is under the umbrella?

A: **Two women.**
A: A child.
A: An old man.
A: A husband and a wife.

Agrawal et al, "VQA: Visual Question Answering", ICCV 2015
Zhu et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016

Visual Question Answering (VQA)



Agrawal et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2015
Figures from Agrawal et al, copyright IEEE 2015. Reproduced for educational purposes.