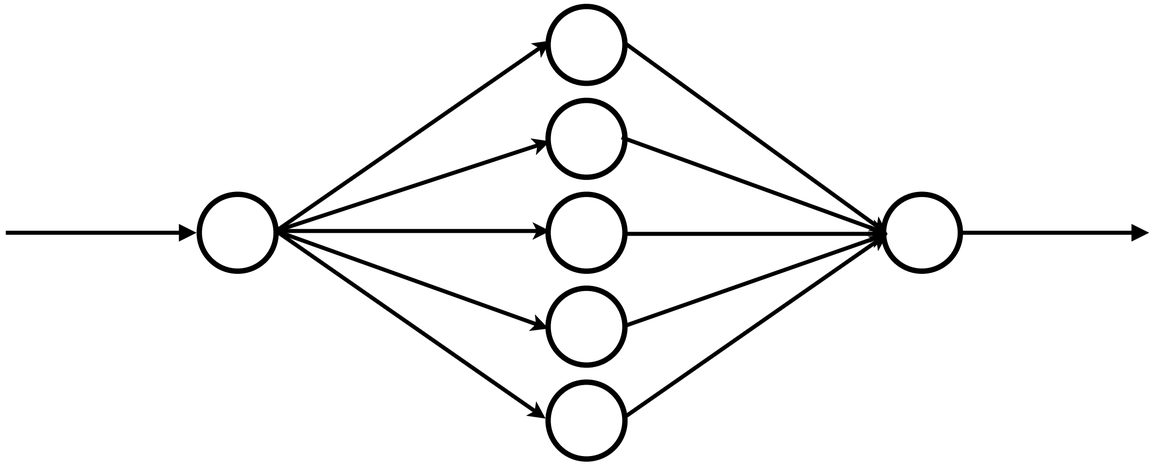
LSTM Networks

Fundamentals

Rahul Singh
rsingh@arrsingh.com

Recurrent Neural Networks

Recurrent Neural Networks can model inputs that have a temporal dependency and ordering

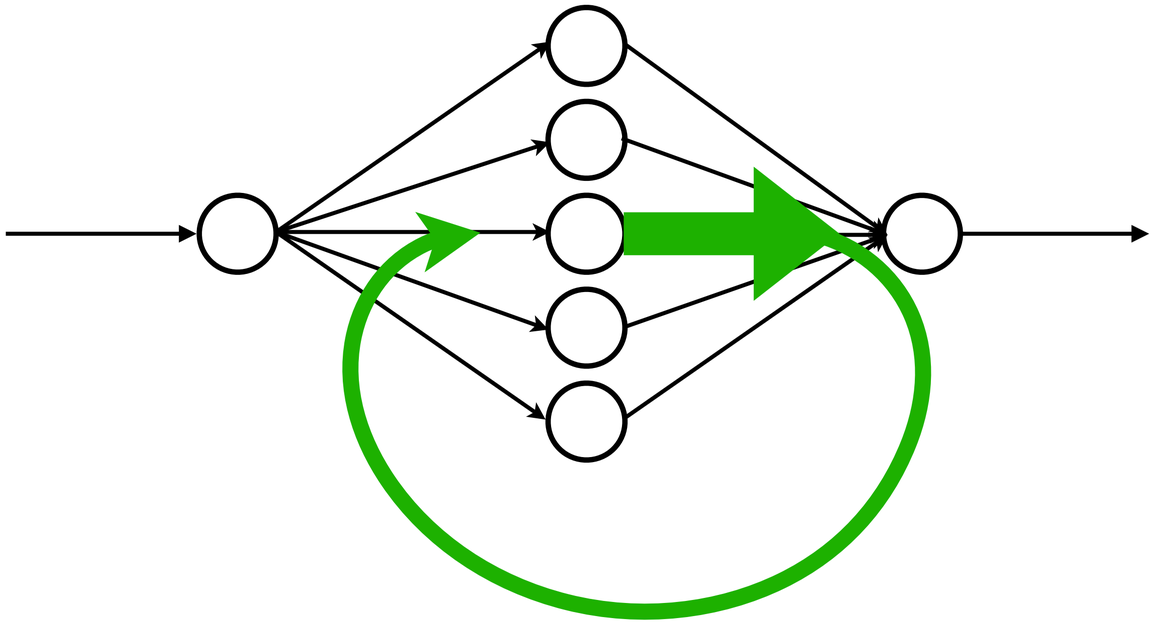


In an RNN, since the inputs have a temporal dependency, the inputs are fed to the network and processed one at a time.

The structure of an RNN is similar to that of a FeedForward Network (input layer, hidden layers and an output layer) with one additional nuance…

Recurrent Neural Networks

Recurrent Neural Networks can model inputs that have a temporal dependency and ordering



In an RNN, since the inputs have a temporal dependency, the inputs are fed to the network and processed one at a time.

The structure of an RNN is similar to that of a FeedForward Network (input layer, hidden layers and an output layer) with one additional nuance…

… the output from the hidden layer at time t, is fed back as input to the hidden layer at time $t + 1$ along with the input at time $t + 1$

Recurrent Neural Networks

Simple Example: **Network Traffic Prediction**

Problem Statement: We have a network service (http) and we want to predict the traffic every hour (for capacity planning). We want to scale up the service (add capacity) if we predict that the traffic is going to increase, and we want to scale down the service (release capacity) if we predict the traffic is going to decrease.

Here are the observations of traffic (Requests Per Hour) for the past 8 hours:

Hour	Traffic
10:00 PM	530
11:00 PM	645
12:00 AM	732
1:00 AM	845
2:00 AM	865
3:00 AM	720
4:00 AM	485
5:00 AM	366

Question:
Can we predict the traffic at 6:00 AM?
Should we add capacity or reduce it?

We can model this as a Recurrent Neural Network?

Recurrent Neural Networks

Simple Example: **Network Traffic Prediction**

Problem Statement: We have a network service (http) and we want to predict the traffic every hour (for capacity planning). We want to scale up the service (add capacity) if we predict that the traffic is going to increase, and we want to scale down the service (release capacity) if we predict the traffic is going to decrease.

Given a sequence (traffic) over 8 time steps…

530,	645,	732,	845,	865,	720,	485,	366	?
t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8

Question:
Can we predict the traffic at t_8 (6:00 AM)?
Should we add capacity or reduce it?

Recurrent Neural Networks

Lets take another example…

Remember the First Element

Problem Statement: Given a sequence of T random numbers, drawn from the set $\{0,1,2...9\}$, train an RNN to predict what the first element of the sequence was.

Example sequence over 8 time steps...

3,	7,	9,	4,	6,	8,	7,	5	?
t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	

The first element is 3

Question:
Can we train an RNN to predict the first element seen - in this case 3?

During training (BPTT) the derivative of any loss term L_t with respect to the recurrent parameter (β_1, W_1, W_{h1}) must chain backward through all previous hidden states.

$$\frac{\partial}{\partial \beta_1} L_t = \frac{\partial}{\partial H_{1t}} L_t \left[\sum_{k=0}^{T} \left(\prod_{j=k+1}^{t} \frac{\partial}{\partial H_{1j-1}} H_{1j} \right) \frac{\partial}{\partial \beta_1} H_{1k} \right]$$

Recurrent Neural Networks

Remember the First Element

Problem Statement: Given a sequence of T random numbers, drawn from the set $\{0,1,2...9\}$, train an RNN to predict what the first element of the sequence was.

Example sequence over 8 time steps…

3,	7,	9,	4,	6,	8,	7,	5	?
t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	

The first element is 3

Question:
Can we train an RNN to predict the first element seen - in this case 3?

During training (BPTT) the derivative of any loss term L_t with respect to the recurrent parameter (β_1, W_1, W_{h1}) must chain backward through all previous hidden states.

$$\frac{\partial}{\partial \beta_1} L_t = \frac{\partial}{\partial H_{1t}} L_t \left[\sum_{k=0}^{T} \left(\prod_{j=k+1}^{t} \frac{\partial}{\partial H_{1j-1}} H_{1j} \right) \frac{\partial}{\partial \beta_1} H_{1k} \right]$$

The $\frac{\partial}{\partial H_{1j-1}} H_{1j}$ terms (Jacobians) get multiplied repeatedly.

If the spectral norm (largest singular value) of each Jacobian is consistently < 1 then as sequence length increases the gradient converges to zero exponentially. If it's consistently > 1 then the gradient explodes exponentially.

Recurrent Neural Networks

Remember the First Element

Problem Statement: Given a sequence of T random numbers, drawn from the set $\{0,1,2...9\}$, train an RNN to predict what the first element of the sequence was.

Example sequence over 8 time steps…

3,	7,	9,	4,	6,	8,	7,	5	?
t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	

The first element is 3

Question:
Can we train an RNN to predict the first element seen - in this case 3?

During training (BPTT) the derivative of any loss term L_t with respect to the recurrent parameter (β_1, W_1, W_{h1}) must chain backward through all previous hidden states.

$$\frac{\partial}{\partial \beta_1} L_t = \frac{\partial}{\partial H_{1t}} L_t \left[\sum_{k=0}^{T} \left(\prod_{j=k+1}^{t} \frac{\partial}{\partial H_{1j-1}} H_{1j} \right) \frac{\partial}{\partial \beta_1} H_{1k} \right]$$

The $\frac{\partial}{\partial H_{1j-1}} H_{1j}$ terms (Jacobians) get multiplied repeatedly.

If the spectral norm (largest singular value) of each Jacobian is consistently < 1 then as sequence length increases the gradient converges to zero exponentially. If it's consistently > 1 then the gradient explodes exponentially.

In either case the training will fail

Recurrent Neural Networks

Remember the First Element

Problem Statement: Given a sequence of T random numbers, drawn from the set $\{0,1,2...9\}$, train an RNN to predict what the first element of the sequence was.

Example sequence over 8 time steps…

3,	7,	9,	4,	6,	8,	7,	5	?
t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	

The first element is 3

Question:
Can we train an RNN to predict the first element seen - in this case 3?

During training (BPTT) the derivative of any loss term L_t with respect to the recurrent parameter (β_1, W_1, W_{h1}) must chain backward through all previous hidden states.

When gradients converge to zero: Vanishing Gradient Problem
Training stalls because the gradients converge to zero and no signal reaches the early layers

When gradients explode to infinity: Exploding Gradient Problem
Training fails because loss becomes NaN and the training crashes

The $\frac{\partial}{\partial H_{1j-1}} H_{1j}$ terms (Jacobians) get multiplied repeatedly.

If the spectral norm (largest singular value) of each Jacobian is consistently < 1 then as sequence length increases the gradient converges to zero exponentially. If it's consistently > 1 then the gradient explodes exponentially.

In either case the training will fail

Recurrent Neural Networks

Remember the First Element

Problem Statement: Given a sequence of T random numbers, drawn from the set $\{0,1,2...9\}$, train an RNN to predict what the first element of the sequence was.

Example sequence over 8 time steps…

3,	7,	9,	4,	6,	8,	7,	5	?
t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	

The first element is 3

Question:
Can we train an RNN to predict the first element seen - in this case 3?

During training (BPTT) the derivative of any loss term L_t with respect to the recurrent parameter (β_1, W_1, W_{h1}) must chain backward through all previous hidden states.

The $\frac{\partial}{\partial H_{1j-1}} H_{1j}$ terms (Jacobians) get multiplied repeatedly.

If the spectral norm (largest singular value) of each Jacobian is consistently < 1 then as sequence length increases the gradient converges to zero exponentially. If it's consistently > 1 then the gradient explodes exponentially.

In either case the training will fail

Let's look at this problem in a bit more detail…

Sequence to Vector RNN over 3 Time Steps

Backpropagation Through Time (BPTT) over 3 time steps

Hidden layer gradients are derivatives of Loss from the final time step

For more details see the Tutorial on RNN Training and BPTT

$$\Rightarrow \frac{\partial}{\partial \beta_1} L = \frac{\partial}{\partial \beta_1} L_2$$

$$\Rightarrow \frac{\partial}{\partial \beta_1} L = \frac{\partial}{\partial \hat{Y}_2} L_2 \frac{\partial}{\partial H_{12}} \hat{Y}_2 \frac{\partial}{\partial \beta_1} H_{12} +$$

$$\frac{\partial}{\partial \hat{Y}_2} L_2 \frac{\partial}{\partial H_{12}} \hat{Y}_2 \frac{\partial}{\partial H_{11}} H_{12} \frac{\partial}{\partial \beta_1} H_{11} +$$

$$\frac{\partial}{\partial \hat{Y}_2} L_2 \frac{\partial}{\partial H_{12}} \hat{Y}_2 \frac{\partial}{\partial H_{11}} H_{12} \frac{\partial}{\partial H_{10}} H_{11} \frac{\partial}{\partial \beta_1} H_{10}$$

Jacobians

$$\Rightarrow \frac{\partial}{\partial \beta_1} L = \frac{\partial}{\partial \hat{Y}_2} L_2 \frac{\partial}{\partial H_{12}} \hat{Y}_2 \left[\frac{\partial}{\partial \beta_1} H_{12} + \frac{\partial}{\partial H_{11}} H_{12} \frac{\partial}{\partial \beta_1} H_{11} + \frac{\partial}{\partial H_{11}} H_{12} \frac{\partial}{\partial H_{10}} H_{11} \frac{\partial}{\partial \beta_1} H_{10} \right]$$

For backpropagation we have to calculate the partial derivative of the Loss function w.r.t the parameters $W_{h1}, W_1, W_2, \beta_1, \beta_2$

$$\frac{\partial}{\partial W_{h1}} L \quad \frac{\partial}{\partial W_1} L \quad \frac{\partial}{\partial W_2} L \quad \frac{\partial}{\partial \beta_1} L \quad \frac{\partial}{\partial \beta_2} L$$

The $\frac{\partial}{\partial H_{1j-1}} H_{1j}$ terms (Jacobians) get multiplied repeatedly.

If the spectral norm (largest singular value) of each Jacobian is consistently < 1 then as sequence length increases the gradient converges to zero exponentially. If it's consistently > 1 then the gradient explodes exponentially.

Let's look at this problem in a bit more detail…

Recurrent Neural Networks

Remember the First Element

Problem Statement: Given a sequence of T random numbers, drawn from the set $\{0,1,2...9\}$, train an RNN to predict what the first element of the sequence was.

3,	7,	9,	4,	6,	8,	7,	5	?
t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	

Question:
Can we train an RNN to predict the first element seen - in this case 3?

In practice, this means that RNNs cannot “remember” elements earlier in the sequence as the sequence length increases

Answer:
For long sequences we cannot train an RNN to predict the first element seen. The training will fail because of the vanishing gradient problem

Let’s look at this problem in a bit more detail…

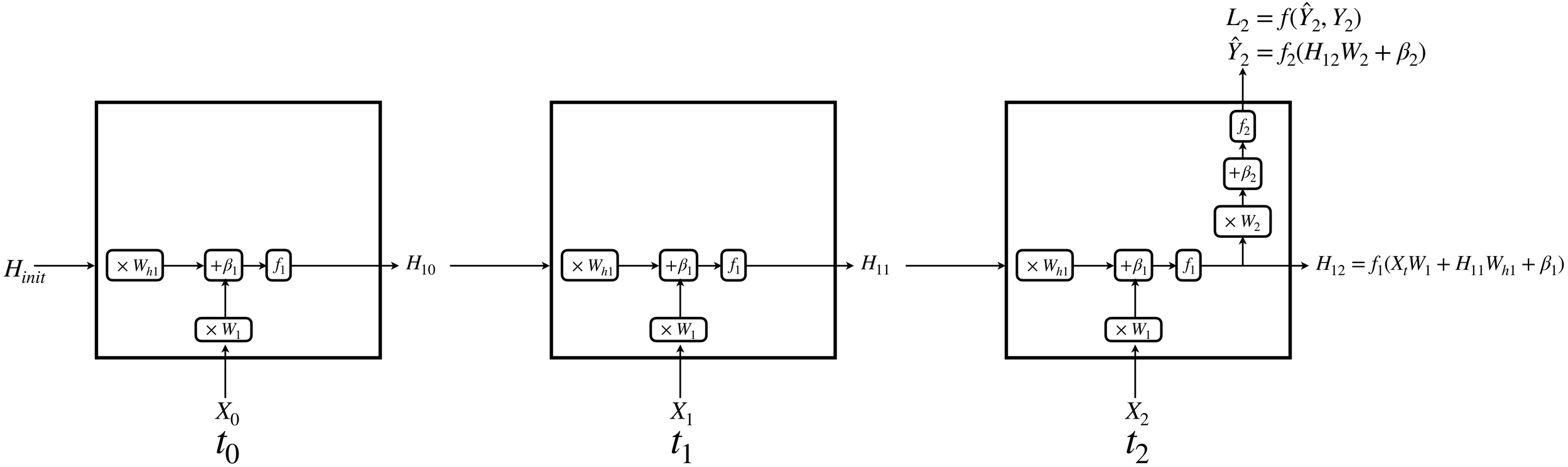
Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.

Let's review an RNN unrolled over 3 time steps…

For the rest of the slides we'll assume a single hidden layer and drop the layer subscript to simplify the notation

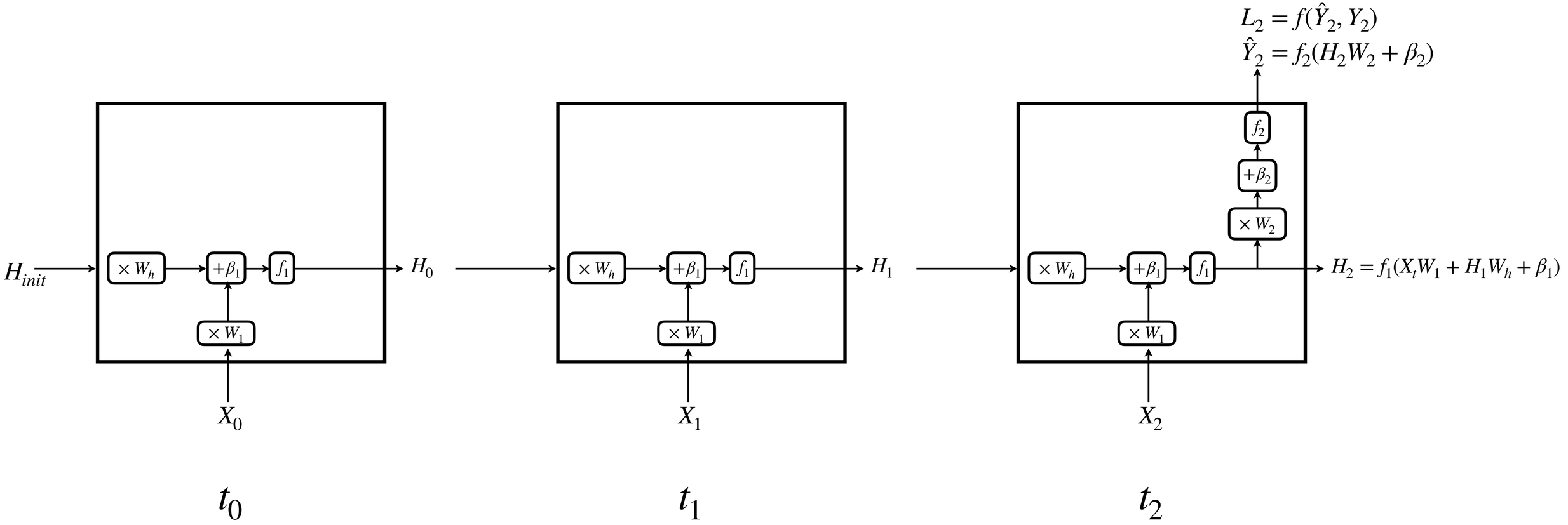
We'll replace H_{11} with H_1 and so on



Let's look at this problem in a bit more detail…

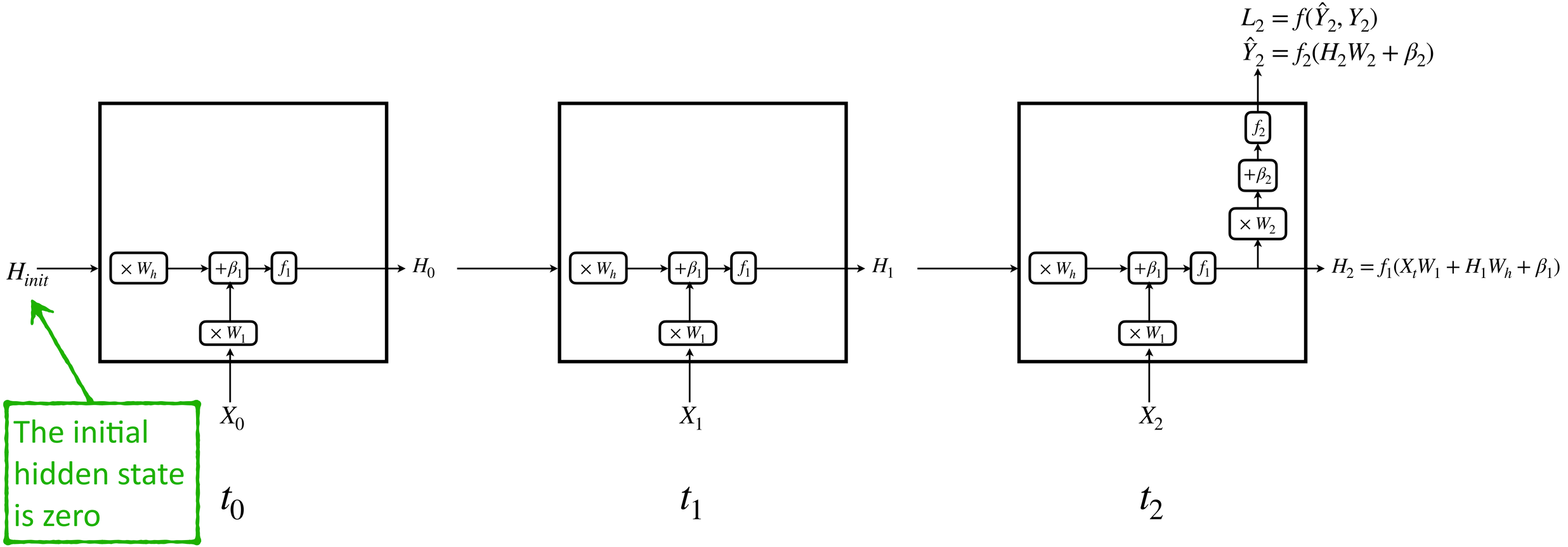
Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.



Recurrent Neural Networks

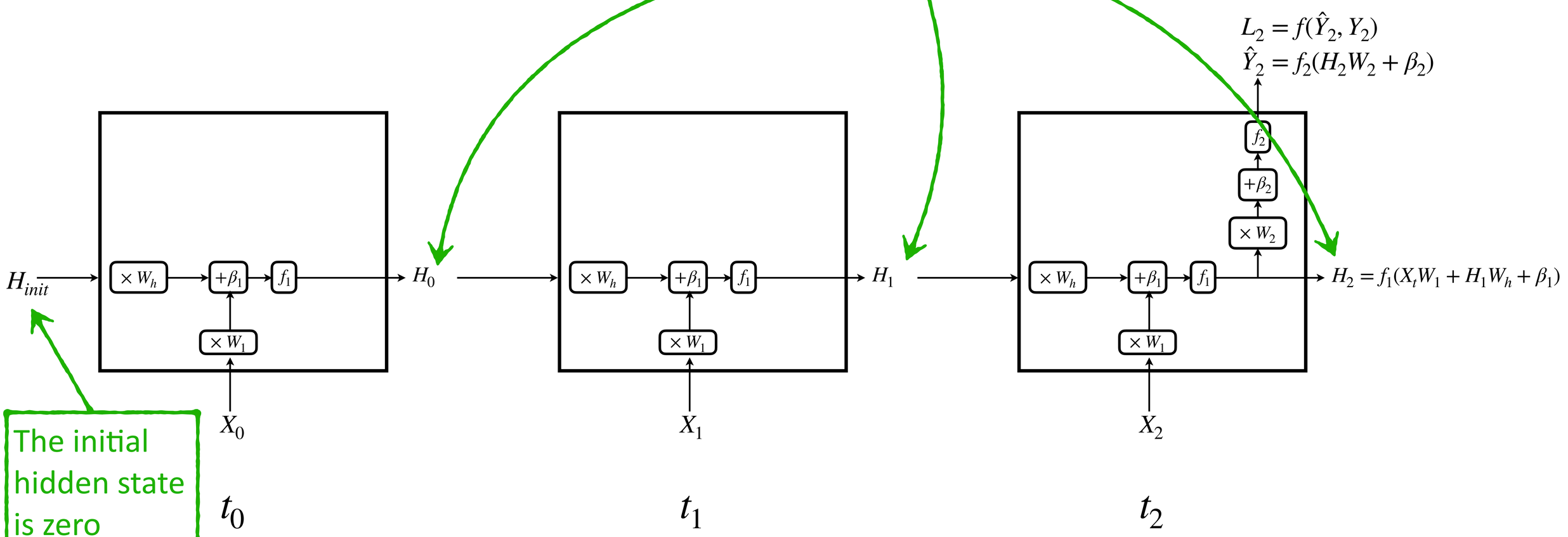
RNN cells store representations of input events over time via the hidden state.



Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.

Subsequent hidden states store current input as well as representations of all previous inputs

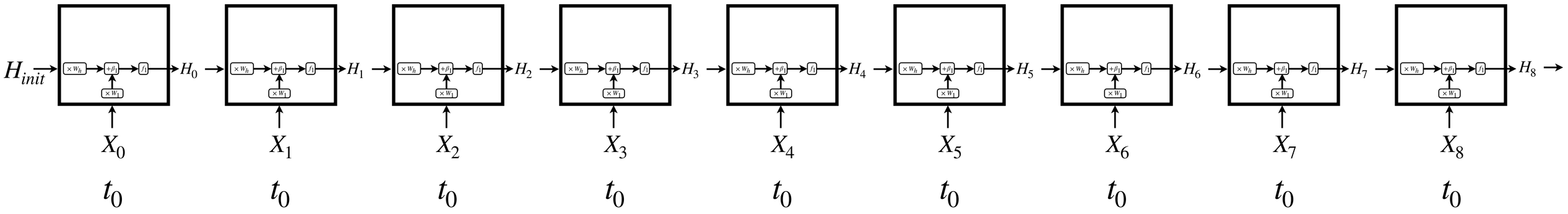


The initial hidden state is zero

RNN cells store representations of input events over time via the hidden state.

Recurrent Neural Networks

As the sequence length increases, the earlier inputs get “washed out” over time

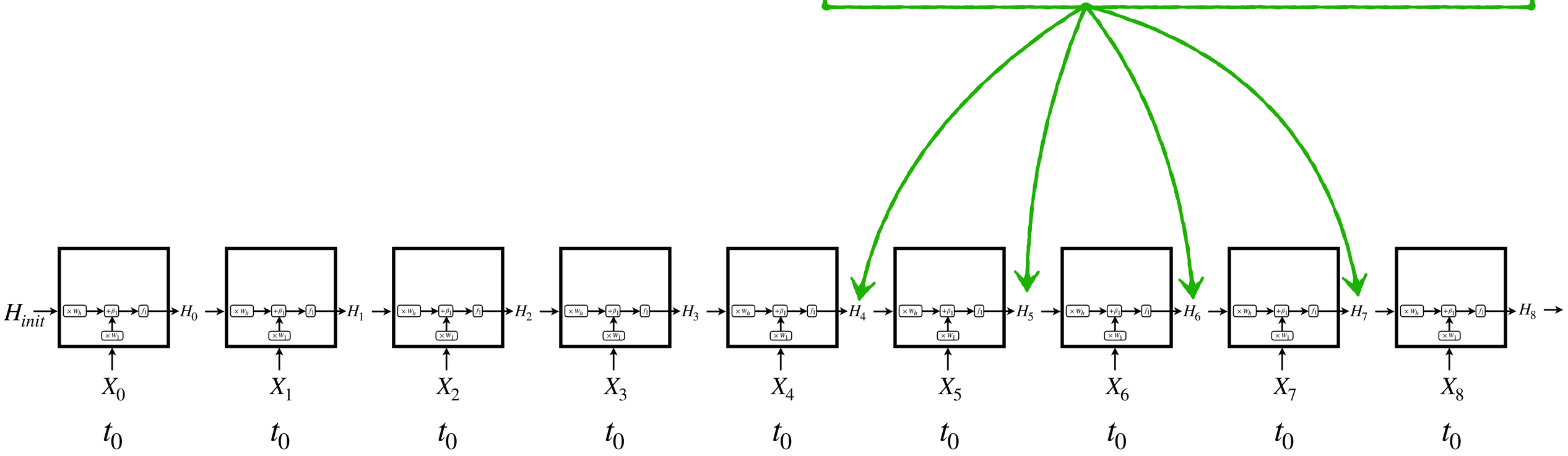


Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.

As the sequence length increases, the earlier inputs get “washed out” over time

Recent inputs dominate the hidden state…



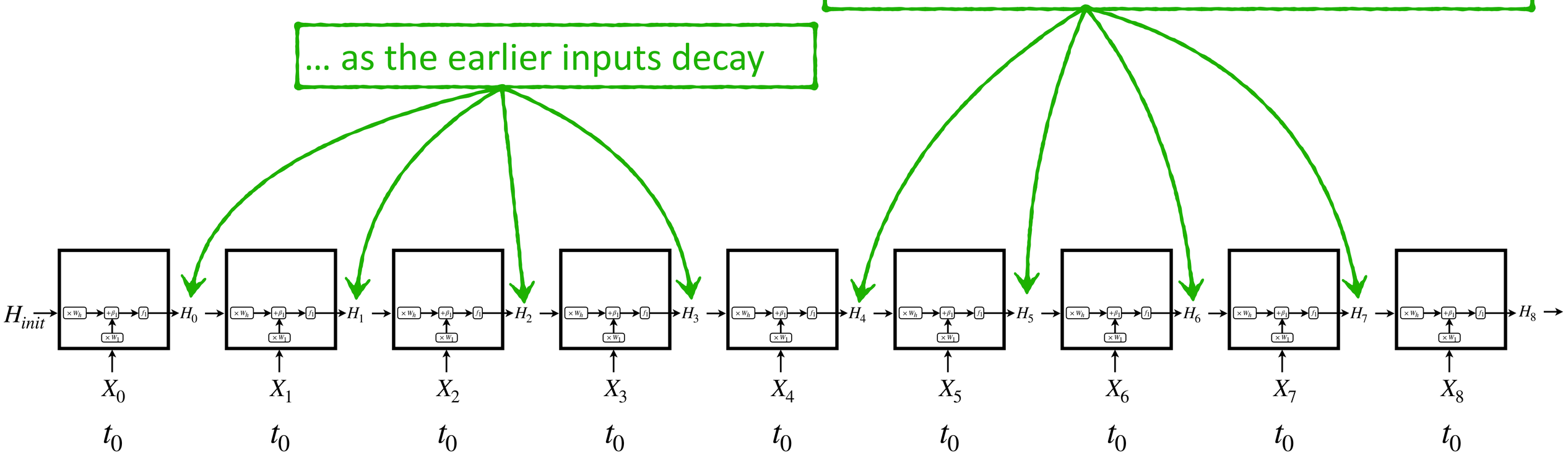
Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.

As the sequence length increases, the earlier inputs get “washed out” over time

Recent inputs dominate the hidden state…

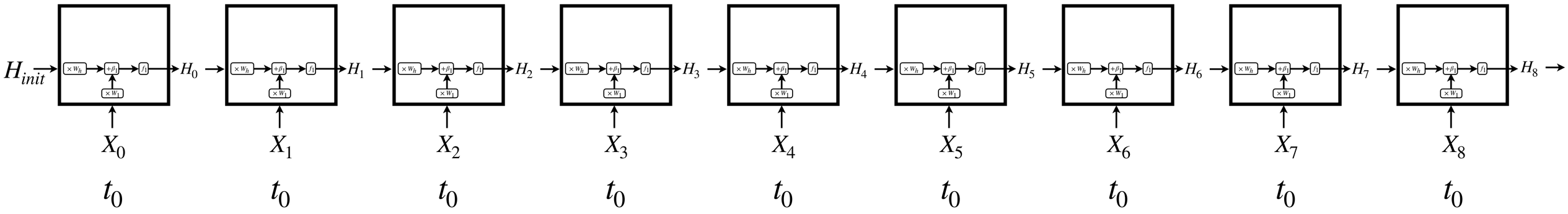
… as the earlier inputs decay



Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.

The RNN has “Short Term Memory”

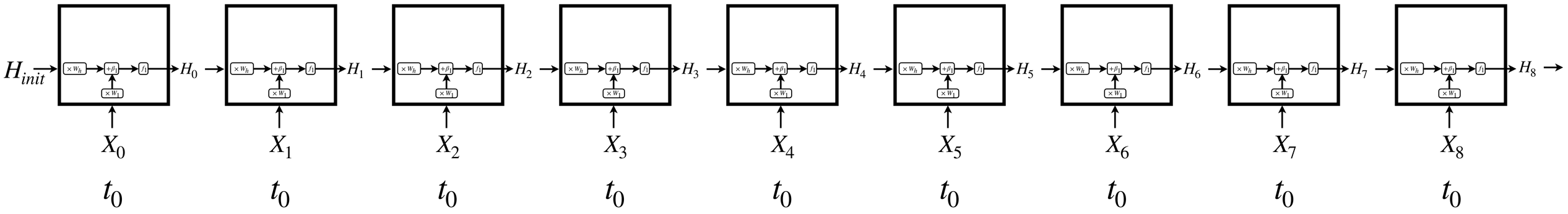


Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.

The RNN has “Short Term Memory”

Result: RNNs cannot “remember” elements earlier in the sequence as the sequence length increases



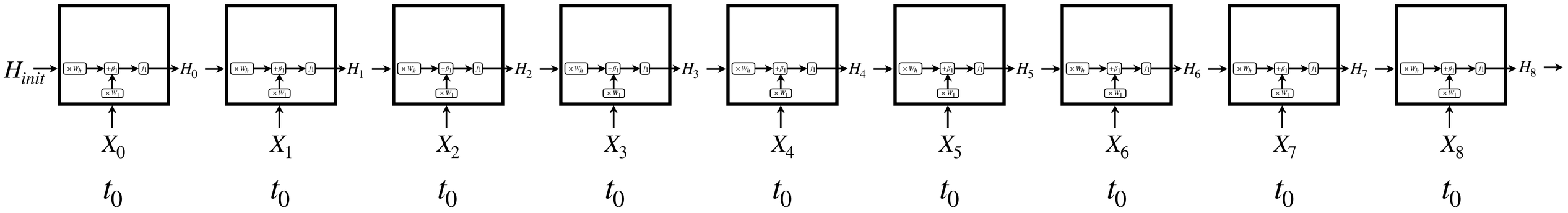
Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.

The RNN has “Short Term Memory”

Result: RNNs cannot “remember” elements earlier in the sequence as the sequence length increases

Question: Can we extend the RNN architecture to solve this problem?



Recurrent Neural Networks

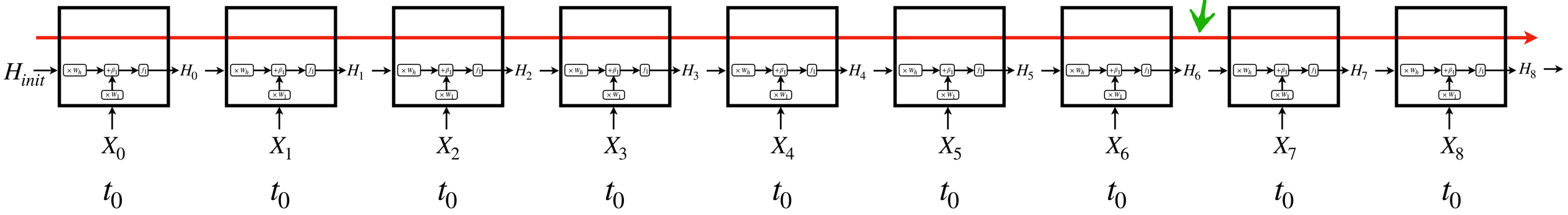
RNN cells store representations of input events over time via the hidden state.

The RNN has “Short Term Memory”

Result: RNNs cannot “remember” elements earlier in the sequence as the sequence length increases

What if we add a second state that preserves earlier inputs without decay?

Question: Can we extend the RNN architecture to solve this problem?



Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.

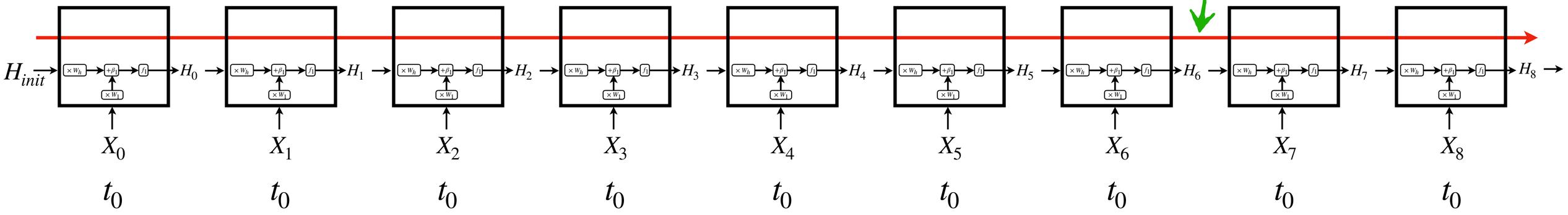
The RNN has “Short Term Memory”

Result: RNNs cannot “remember” elements earlier in the sequence as the sequence length increases

What if we add a second state that preserves earlier inputs without decay?

We’ll call it “Long Term Memory”

Question: Can we extend the RNN architecture to solve this problem?



Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.

The RNN has “Short Term Memory”

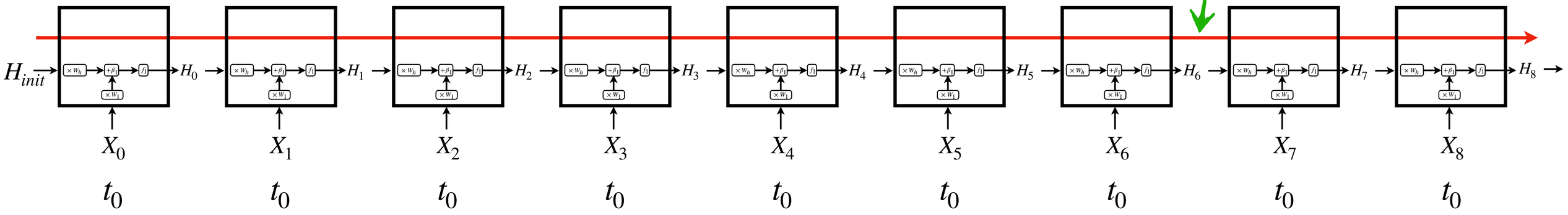
Result: RNNs cannot “remember” elements earlier in the sequence as the sequence length increases

Question: Can we extend the RNN architecture to solve this problem?

What if we add a second state that preserves earlier inputs without decay?

We’ll call it “Long Term Memory”

However its important to be selective. We need mechanisms to decide what to keep, what to forget, and what to add.



Recurrent Neural Networks

RNN cells store representations of input events over time via the hidden state.

The RNN has “Short Term Memory”

Result: RNNs cannot “remember” elements earlier in the sequence as the sequence length increases

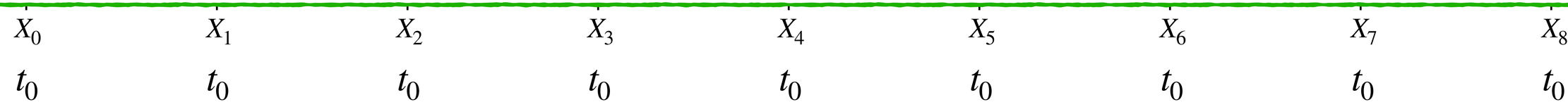
Question: Can we extend the RNN architecture to solve this problem?

What if we add a second state that preserves earlier inputs without decay?

We’ll call it “Long Term Memory”

However its important to be selective. We need mechanisms to decide what to keep, what to forget, and what to add.

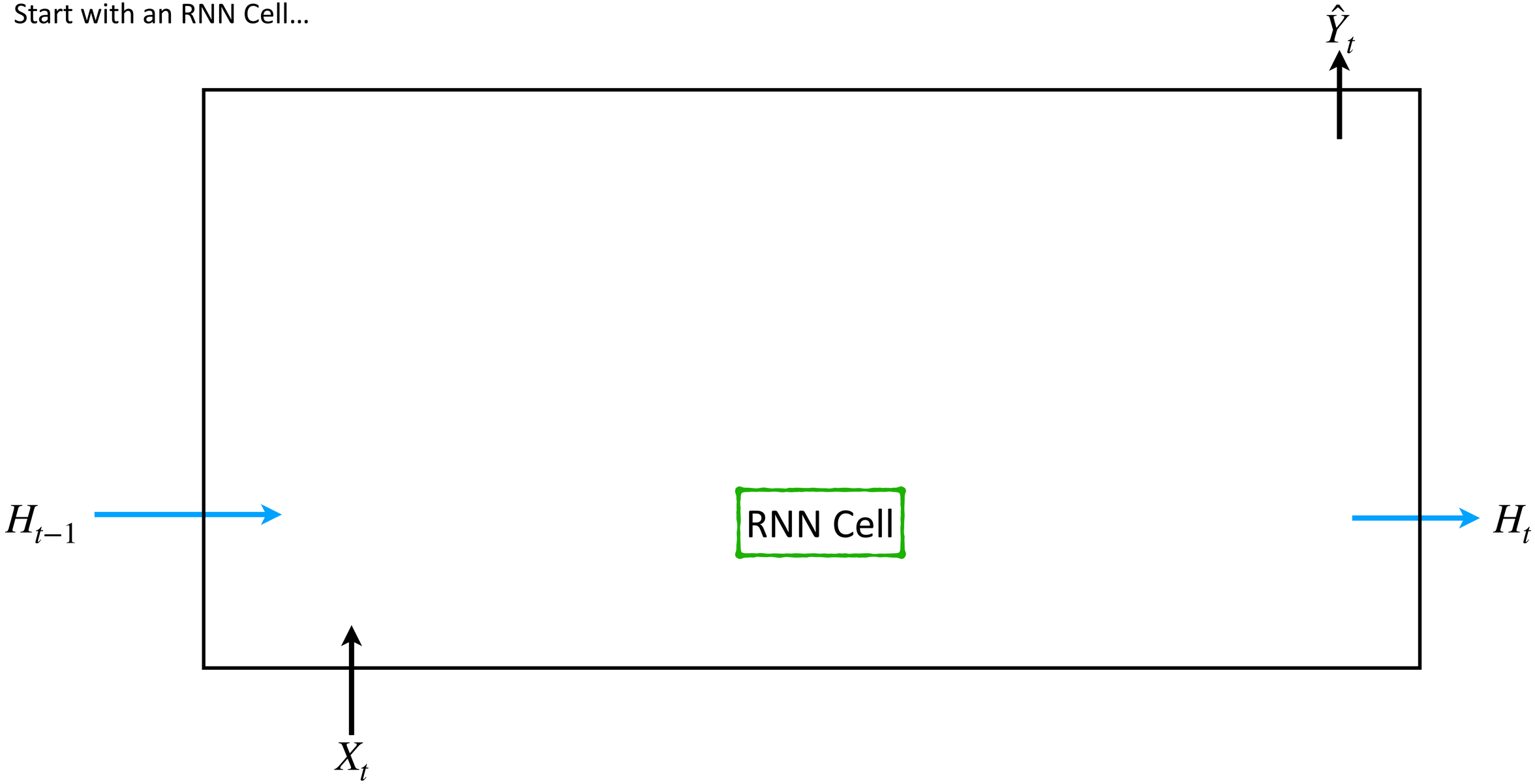
Let’s look at the Cell in a bit more detail



Extending the RNN Architecture for Long Term Memory

LSTM Networks

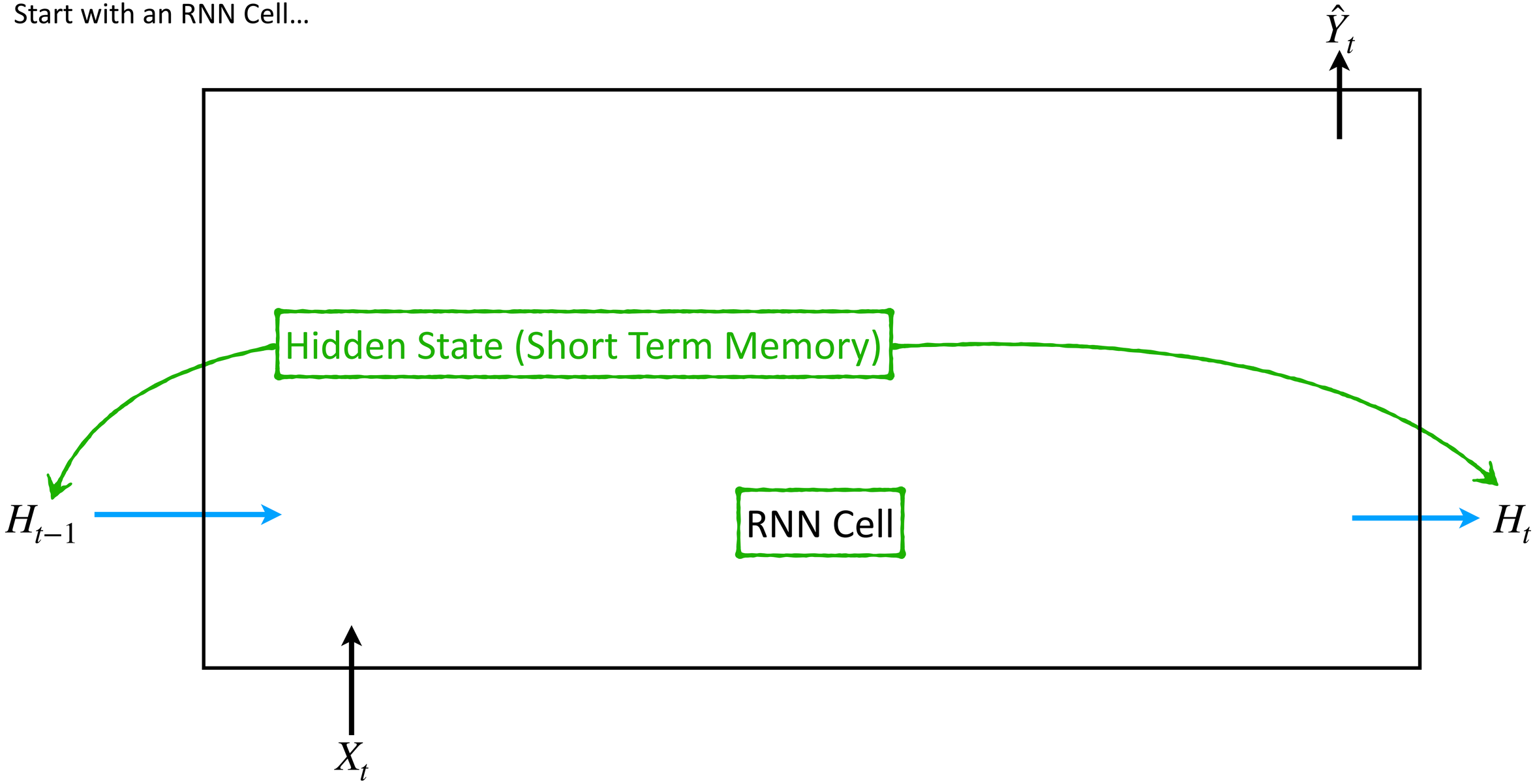
Start with an RNN Cell…



Extending the RNN Architecture for Long Term Memory

LSTM Networks

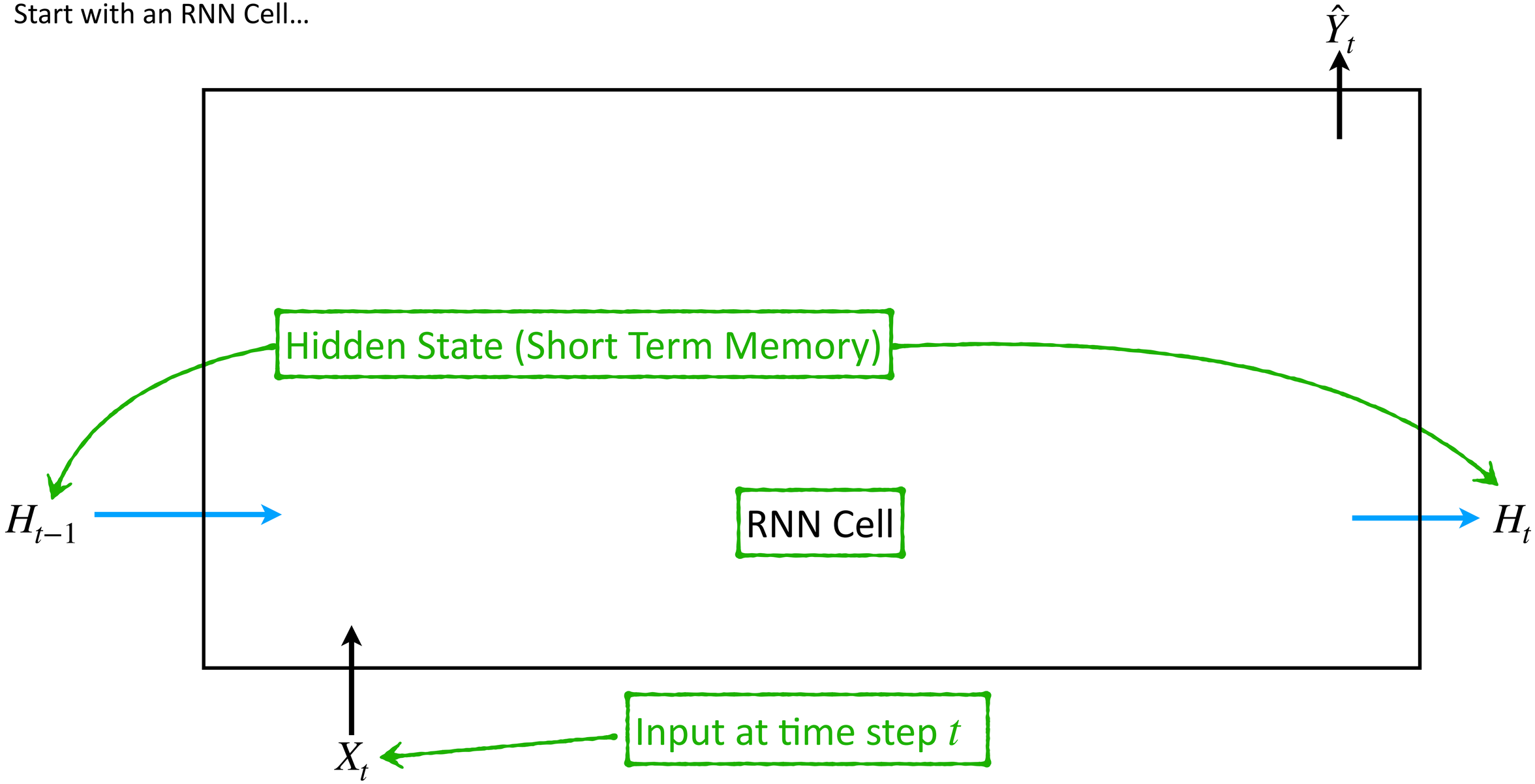
Start with an RNN Cell…



Extending the RNN Architecture for Long Term Memory

Start with an RNN Cell…

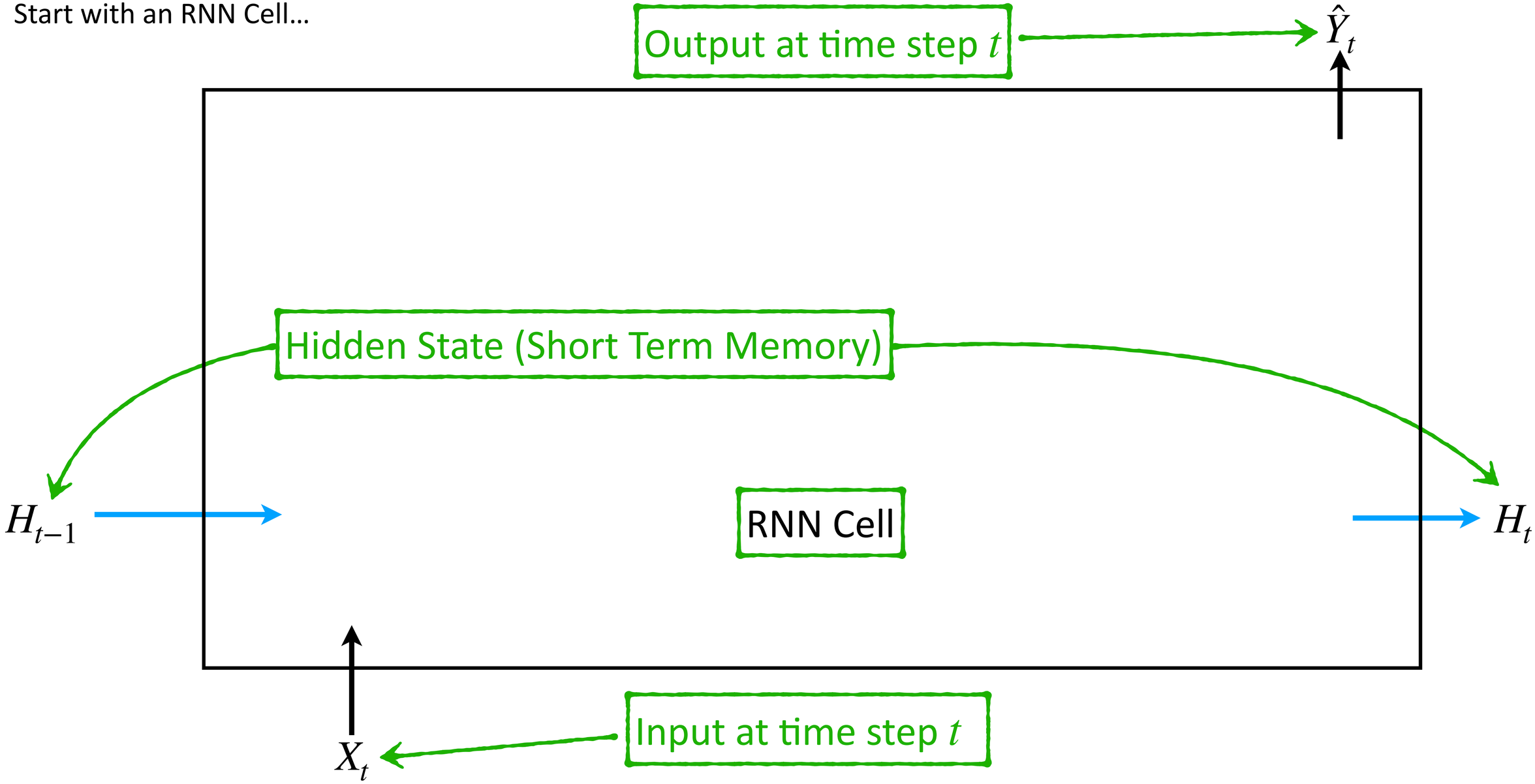
LSTM Networks



Extending the RNN Architecture for Long Term Memory

LSTM Networks

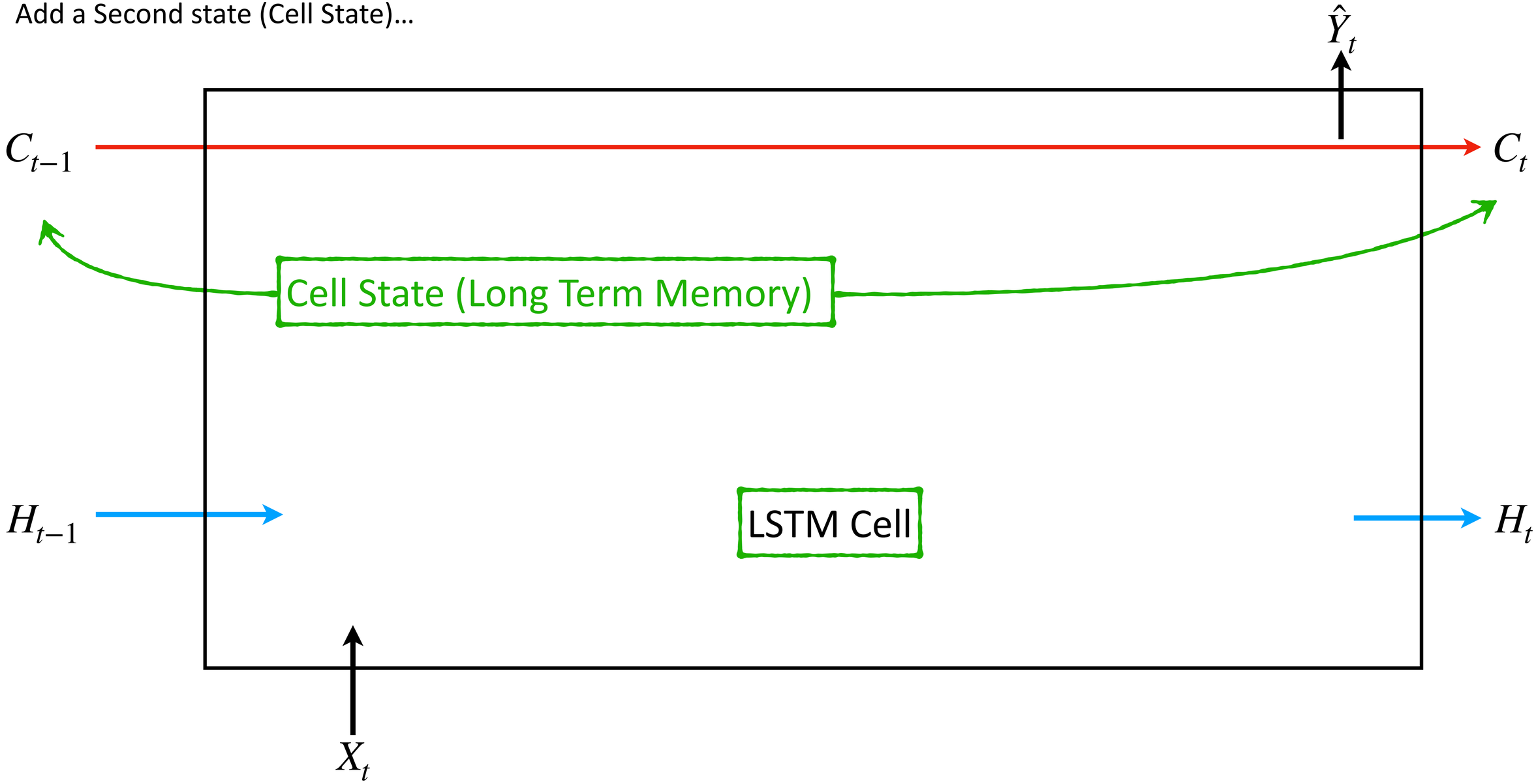
Start with an RNN Cell…



Extending the RNN Architecture for Long Term Memory

LSTM Networks

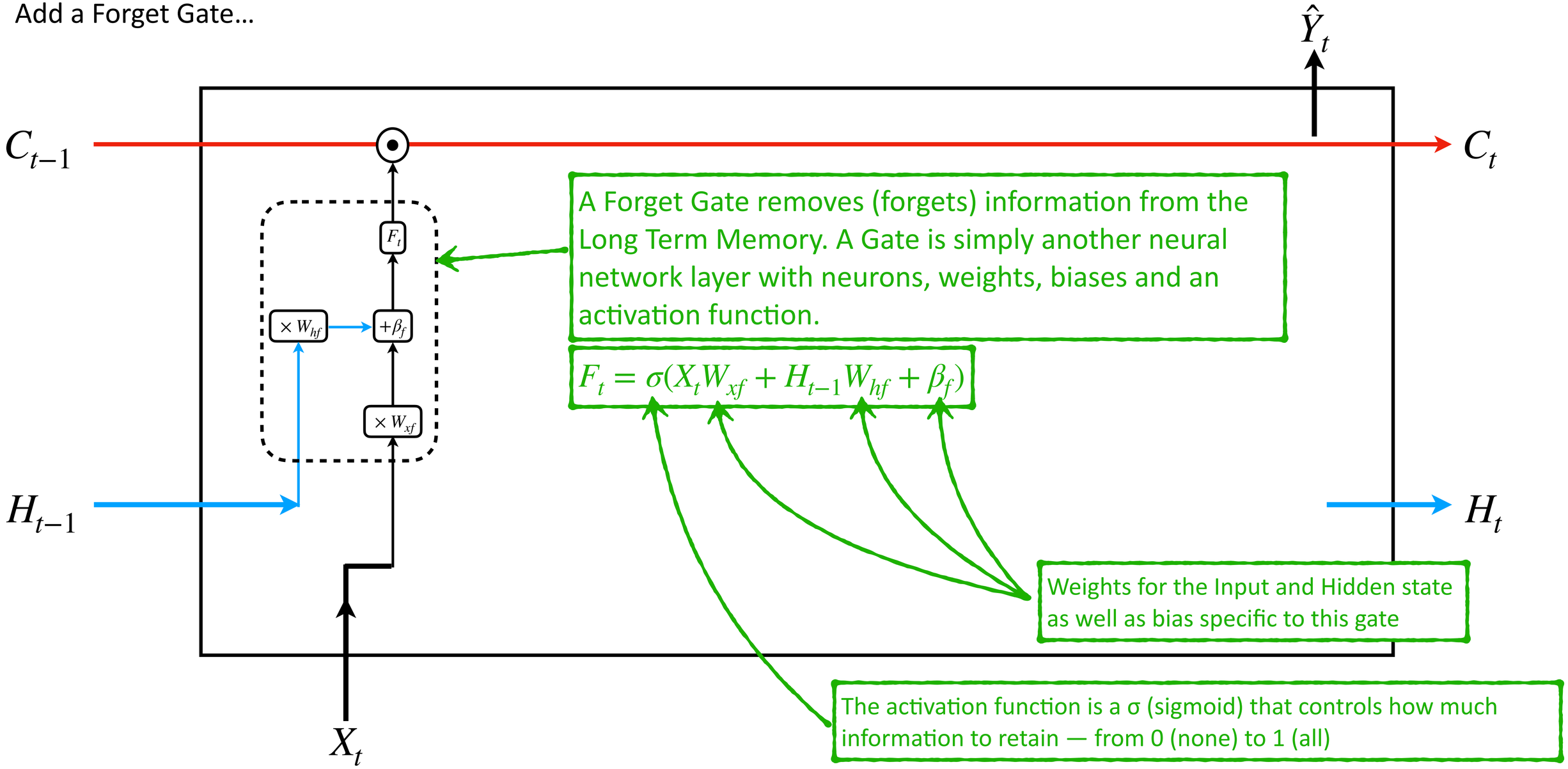
Add a Second state (Cell State)…



Extending the RNN Architecture for Long Term Memory

Add a Forget Gate…

LSTM Networks



A Forget Gate removes (forgets) information from the Long Term Memory. A Gate is simply another neural network layer with neurons, weights, biases and an activation function.

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + \beta_f)$$

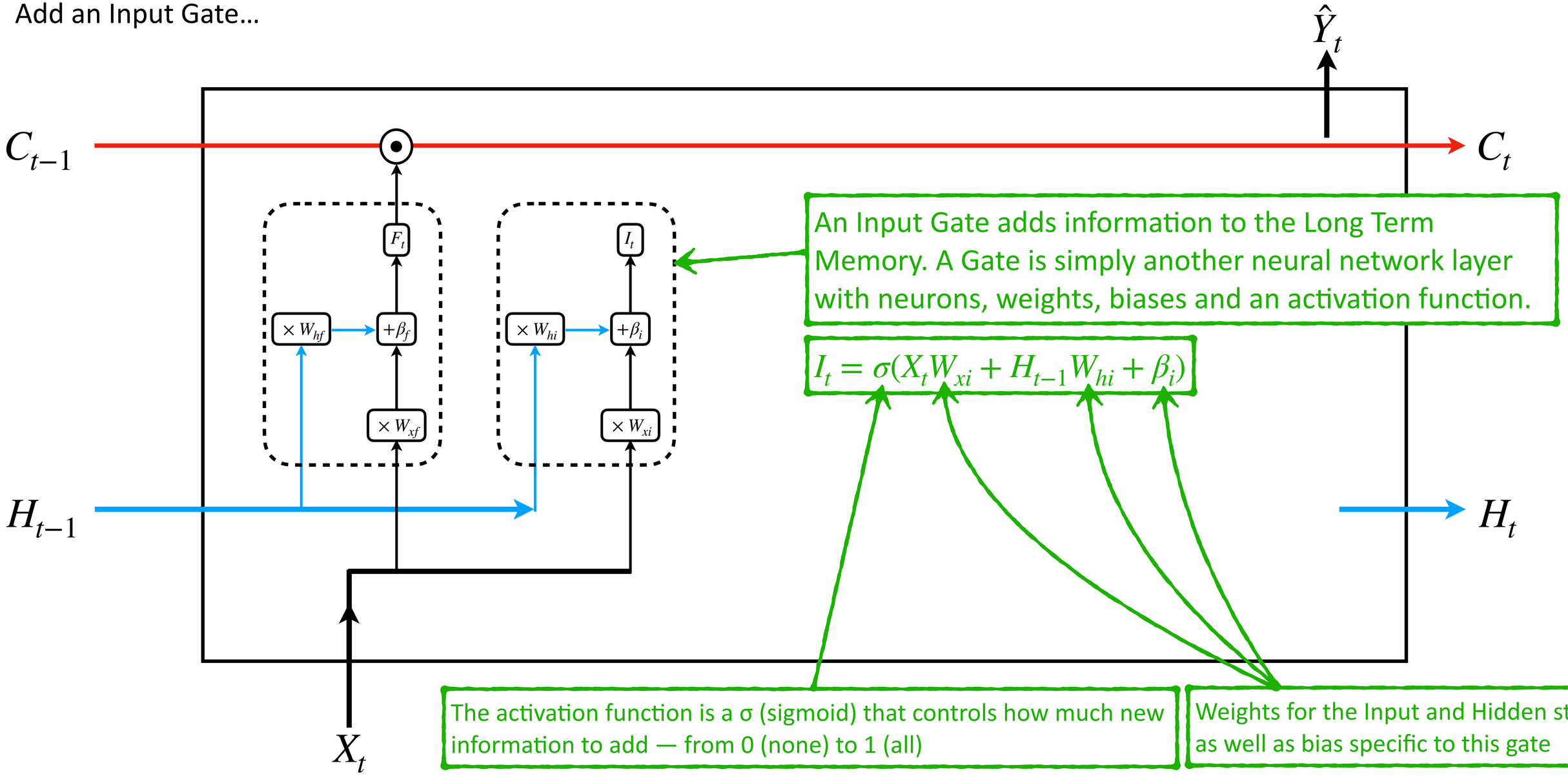
Weights for the Input and Hidden state as well as bias specific to this gate

The activation function is a σ (sigmoid) that controls how much information to retain — from 0 (none) to 1 (all)

Extending the RNN Architecture for Long Term Memory

LSTM Networks

Add an Input Gate…



An Input Gate adds information to the Long Term Memory. A Gate is simply another neural network layer with neurons, weights, biases and an activation function.

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + \beta_i)$$

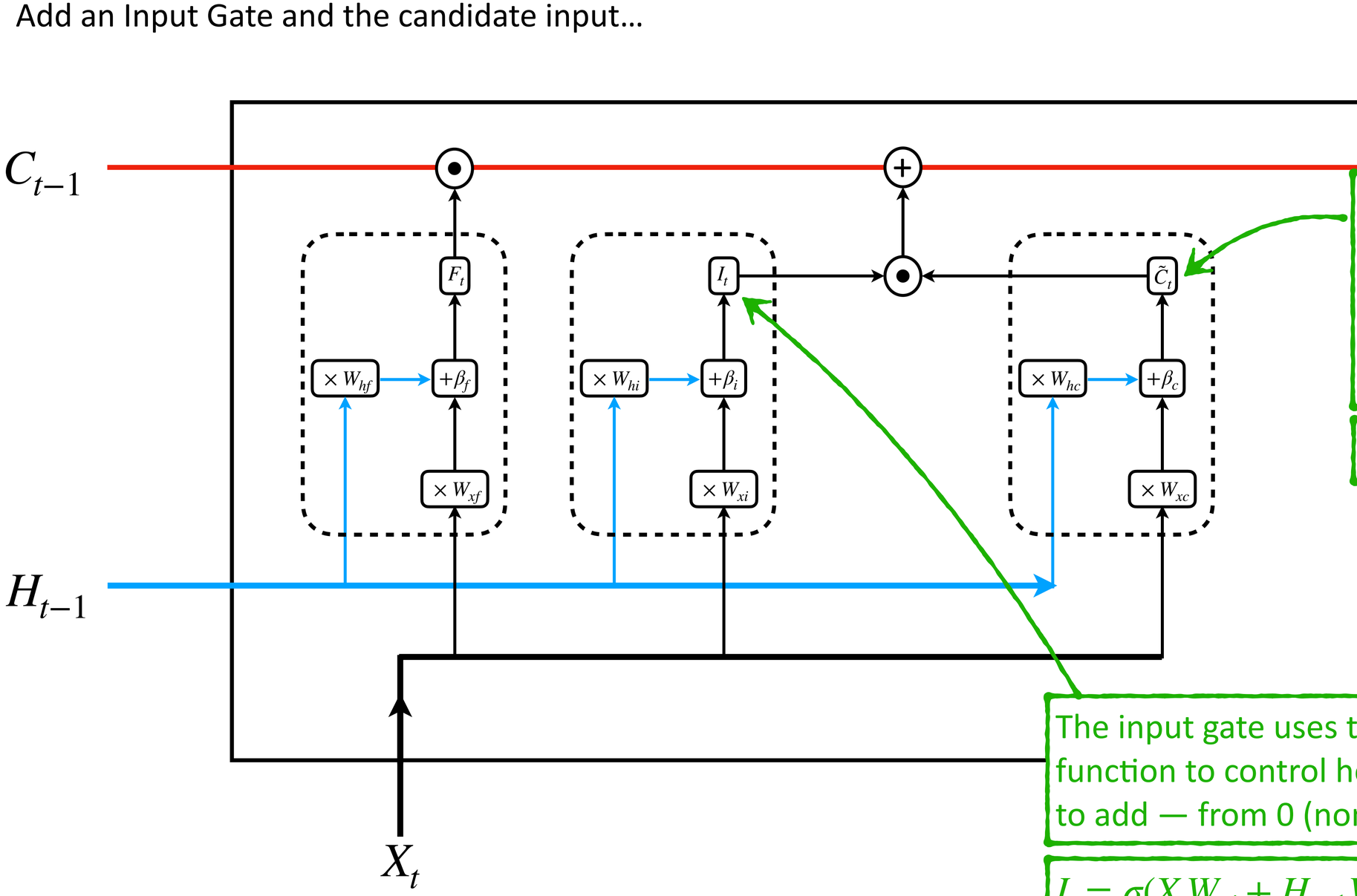
The activation function is a σ (sigmoid) that controls how much new information to add — from 0 (none) to 1 (all)

Weights for the Input and Hidden state as well as bias specific to this gate

Extending the RNN Architecture for Long Term Memory

Add an Input Gate and the candidate input…

LSTM Networks



The candidate ($\tilde{C}$) uses tanh to generate potential new values (from -1 to 1) that the input gate may add to long-term memory

$$\tilde{C}_t = tanh(X_t W_{xc} + H_{t-1} W_{hc} + \beta_c)$$

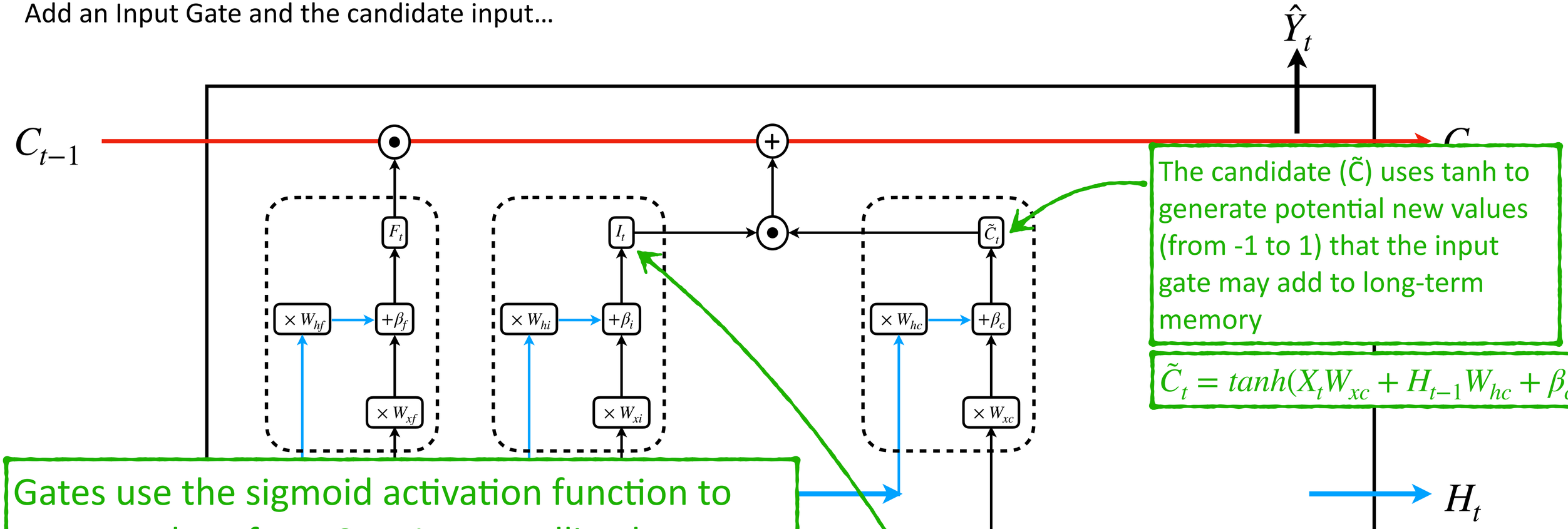
The input gate uses the σ (sigmoid) activation function to control how much new information to add — from 0 (none) to 1 (all)

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + \beta_i)$$

Extending the RNN Architecture for Long Term Memory

Add an Input Gate and the candidate input…

LSTM Networks



The candidate ($\tilde{C}$) uses tanh to generate potential new values (from -1 to 1) that the input gate may add to long-term memory

$$\tilde{C}_t = tanh(X_t W_{xc} + H_{t-1} W_{hc} + \beta_c)$$

Gates use the sigmoid activation function to output values from 0 to 1, controlling how much information passes through.

The candidate uses tanh to output values from -1 to 1, allowing it to increase or decrease cell state values.

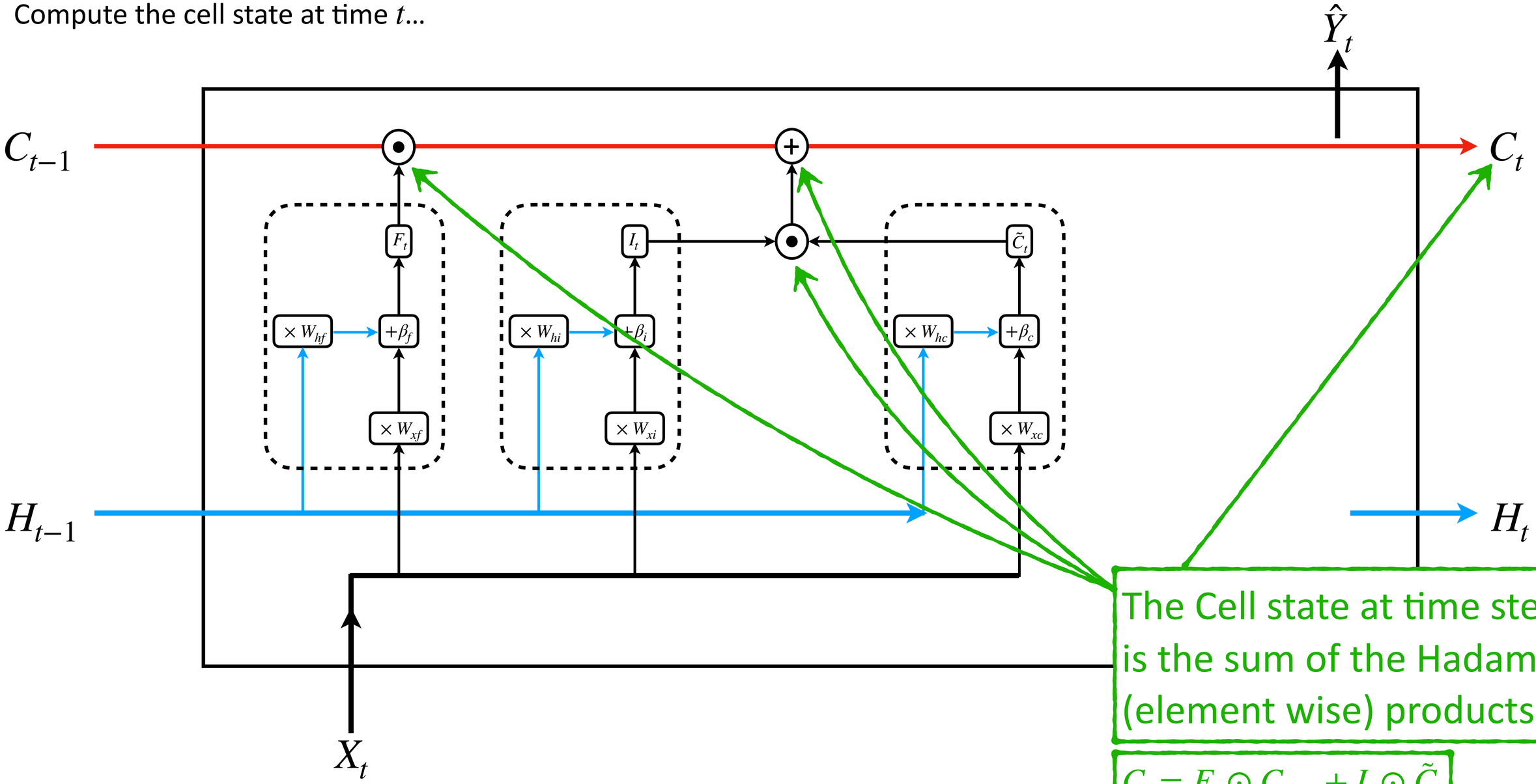
The input gate uses the σ (sigmoid) activation function to control how much new information to add — from 0 (none) to 1 (all)

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + \beta_i)$$

Extending the RNN Architecture for Long Term Memory

LSTM Networks

Compute the cell state at time t...



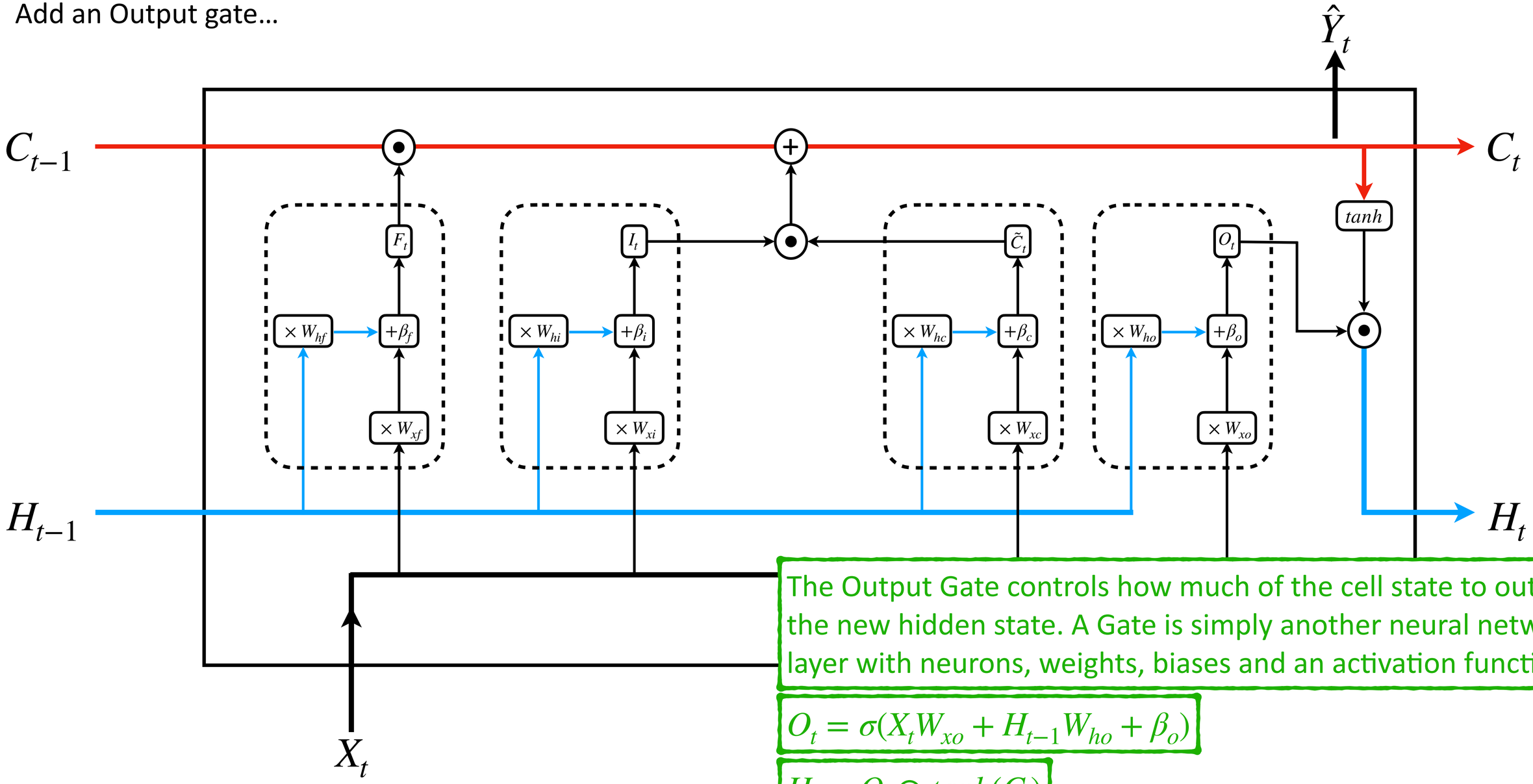
The Cell state at time step t, is the sum of the Hadamard (element wise) products

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

Extending the RNN Architecture for Long Term Memory

LSTM Networks

Add an Output gate…



The Output Gate controls how much of the cell state to output as the new hidden state. A Gate is simply another neural network layer with neurons, weights, biases and an activation function.

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + \beta_o)$$

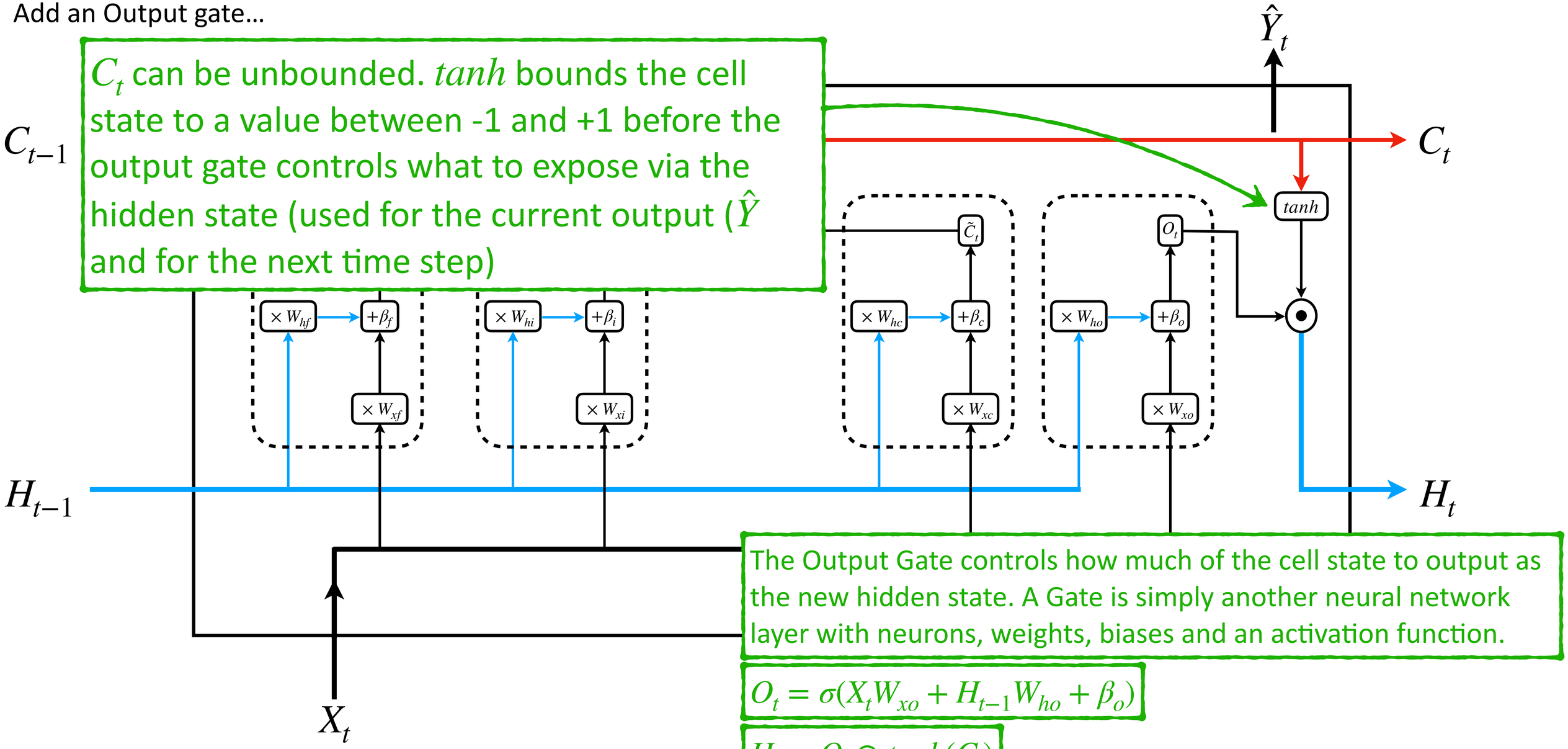
$$H_t = O_t \odot tanh(C_t)$$

Extending the RNN Architecture for Long Term Memory

LSTM Networks

Add an Output gate…

C_t can be unbounded. *tanh* bounds the cell state to a value between -1 and +1 before the output gate controls what to expose via the hidden state (used for the current output ($\hat{Y}$ and for the next time step)



The Output Gate controls how much of the cell state to output as the new hidden state. A Gate is simply another neural network layer with neurons, weights, biases and an activation function.

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + \beta_o)$$

$$H_t = O_t \odot tanh(C_t)$$

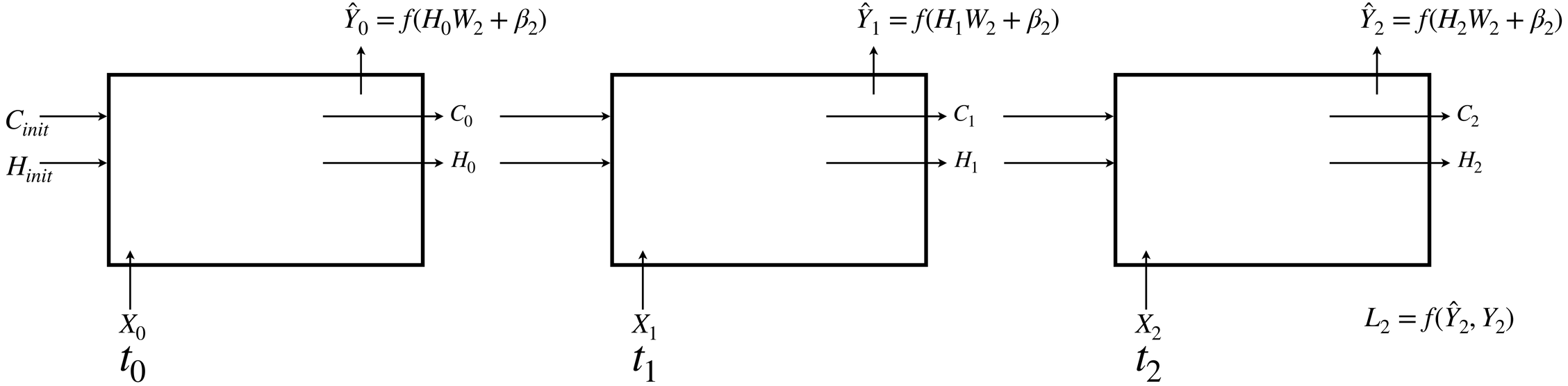
LSTM Networks

Let's review an LSTM unrolled over 3 time steps…

$$F_0 = \sigma(X_0 W_{xf} + H_{init} W_{hf} + \beta_f)$$
$$I_0 = \sigma(X_0 W_{xi} + H_{init} W_{hi} + \beta_i)$$
$$O_0 = \sigma(X_0 W_{xo} + H_{init} W_{ho} + \beta_o)$$
$$\tilde{C}_0 = tanh(X_0 W_{xc} + H_{init} W_{hc} + \beta_c)$$
$$C_0 = F_0 \odot C_{init} + I_0 \odot \tilde{C}_0$$
$$H_0 = O_0 \odot tanh(C_0)$$

$$F_1 = \sigma(X_1 W_{xf} + H_0 W_{hf} + \beta_f)$$
$$I_1 = \sigma(X_1 W_{xi} + H_0 W_{hi} + \beta_i)$$
$$O_1 = \sigma(X_1 W_{xo} + H_0 W_{ho} + \beta_o)$$
$$\tilde{C}_1 = tanh(X_1 W_{xc} + H_0 W_{hc} + \beta_c)$$
$$C_1 = F_1 \odot C_0 + I_1 \odot \tilde{C}_1$$
$$H_1 = O_1 \odot tanh(C_1)$$

$$F_2 = \sigma(X_2 W_{xf} + H_1 W_{hf} + \beta_f)$$
$$I_2 = \sigma(X_2 W_{xi} + H_1 W_{hi} + \beta_i)$$
$$O_2 = \sigma(X_2 W_{xo} + H_1 W_{ho} + \beta_o)$$
$$\tilde{C}_2 = tanh(X_2 W_{xc} + H_1 W_{hc} + \beta_c)$$
$$C_2 = F_2 \odot C_1 + I_2 \odot \tilde{C}_2$$
$$H_2 = O_2 \odot tanh(C_2)$$



Let’s walk through how we train this LSTM unrolled over 3 time steps

Training via Gradient Descent involves a **Forward Pass**, Computing the **Cost Function**, **Backpropagation** and **Parameter Updates**

Training an LSTM uses Backpropagation Through Time (BPTT), computing gradients through each gate (Forget, Input, Output) and the Candidate

LSTM Networks

LSTM unrolled over 3 Time Steps

For backpropagation we have to calculate the partial derivative of the Loss function w.r.t the parameters

$$W_{h1}, W_1, W_2, W_{xf}, W_{xi}, W_{xo}, W_{hf}, W_{hi}, W_{ho}, W_{xc}, W_{hc}, \beta_1, \beta_2, \beta_f, \beta_i, \beta_o, \beta_c$$

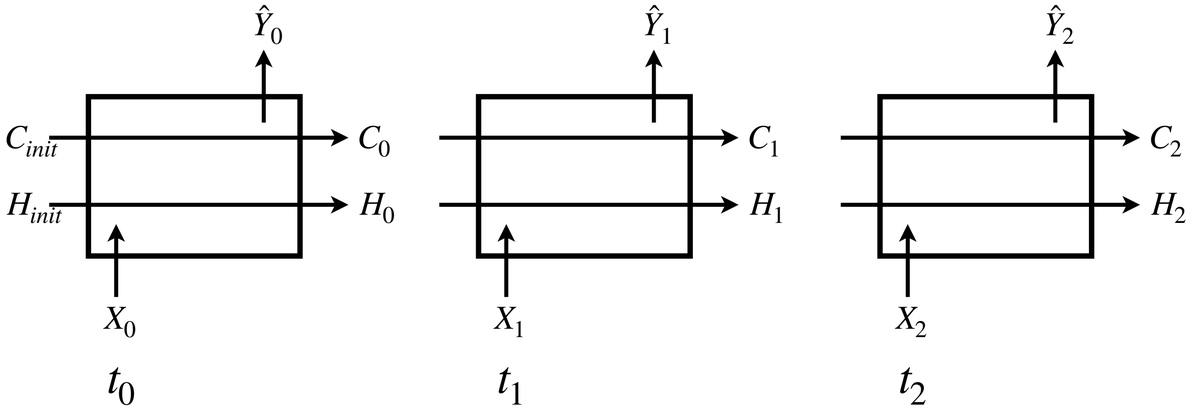
$$\frac{\partial}{\partial W_{h1}}L, \frac{\partial}{\partial W_1}L, \frac{\partial}{\partial W_2}L$$

$$\frac{\partial}{\partial W_{xf}}L, \frac{\partial}{\partial W_{xi}}L, \frac{\partial}{\partial W_{xo}}L,$$

$$\frac{\partial}{\partial W_{hf}}L, \frac{\partial}{\partial W_{hi}}L, \frac{\partial}{\partial W_{ho}}L,$$

$$\frac{\partial}{\partial W_{xc}}L, \frac{\partial}{\partial W_{hc}}L$$

$$\frac{\partial}{\partial \beta_1}L, \frac{\partial}{\partial \beta_2}L, \frac{\partial}{\partial \beta_f}L, \frac{\partial}{\partial \beta_i}L, \frac{\partial}{\partial \beta_o}L, \frac{\partial}{\partial \beta_c}L$$



Gradient Descent for LSTM

Step 1: Start with initial values for the Parameters

$$W_{h1}, W_1, W_2, W_{xf}, W_{xi}, W_{xo}, W_{hf}, W_{hi}, W_{ho}, W_{xc}, W_{hc}, \beta_1, \beta_2, \beta_f, \beta_i, \beta_o, \beta_c$$

Step 2: Forward Propagation (compute)…

$$F_0, I_0, O_0, \tilde{C}_0, C_0, H_0, \hat{Y}_0 \qquad F_2, I_2, O_2, \tilde{C}_2, C_2, H_2, \hat{Y}_2$$

$$F_1, I_1, O_1, \tilde{C}_1, C_1, H_1, \hat{Y}_1 \qquad L = L_0 + L_1 + L_2$$

Step 3: Backpropagation Through Time

$$\frac{\partial}{\partial W_{h1}}L, \frac{\partial}{\partial W_1}L, \frac{\partial}{\partial W_2}L, \frac{\partial}{\partial W_{hf}}L, \frac{\partial}{\partial W_{hi}}L, \frac{\partial}{\partial W_{ho}}L, \frac{\partial}{\partial W_{xc}}L, \frac{\partial}{\partial W_{hc}}L$$

$$\frac{\partial}{\partial W_{xf}}L, \frac{\partial}{\partial W_{xi}}L, \frac{\partial}{\partial W_{xo}}L, \frac{\partial}{\partial \beta_1}L, \frac{\partial}{\partial \beta_2}L, \frac{\partial}{\partial \beta_f}L, \frac{\partial}{\partial \beta_i}L, \frac{\partial}{\partial \beta_o}L, \frac{\partial}{\partial \beta_c}L$$

Step 4: Parameter Updates

$$\beta_2 = \beta_2 - \left(\frac{\partial}{\partial \beta_2}L\right) \times lr \qquad \beta_o = \beta_o - \left(\frac{\partial}{\partial \beta_o}L\right) \times lr \qquad W_{hf} = W_{hf} - \left(\frac{\partial}{\partial W_{hf}}L\right) \times lr \qquad W_{xi} = W_{xi} - \left(\frac{\partial}{\partial W_{xi}}L\right) \times lr$$

$$\beta_1 = \beta_1 - \left(\frac{\partial}{\partial \beta_1}L\right) \times lr \qquad W_{h1} = W_{h1} - \left(\frac{\partial}{\partial W_{h1}}L\right) \times lr \qquad W_{hi} = W_{hi} - \left(\frac{\partial}{\partial W_{hi}}L\right) \times lr \qquad W_{xo} = W_{xo} - \left(\frac{\partial}{\partial W_{xo}}L\right) \times lr$$

$$\beta_f = \beta_f - \left(\frac{\partial}{\partial \beta_f}L\right) \times lr \qquad W_1 = W_1 - \left(\frac{\partial}{\partial W_1}L\right) \times lr \qquad W_{ho} = W_{ho} - \left(\frac{\partial}{\partial W_{ho}}L\right) \times lr \qquad W_{xc} = W_{xc} - \left(\frac{\partial}{\partial W_{xc}}L\right) \times lr$$

$$\beta_i = \beta_i - \left(\frac{\partial}{\partial \beta_i}L\right) \times lr \qquad W_2 = W_2 - \left(\frac{\partial}{\partial W_2}L\right) \times lr \qquad W_{xf} = W_{xf} - \left(\frac{\partial}{\partial W_{xf}}L\right) \times lr \qquad W_{hc} = W_{hc} - \left(\frac{\partial}{\partial W_{hc}}L\right) \times lr$$

$$\beta_c = \beta_c - \left(\frac{\partial}{\partial \beta_c}L\right) \times lr$$

Step 5: Go to step 2 and repeat

Related Tutorials & Textbooks

Neural Networks

An introduction to Neural Networks starting from a foundation of linear regression, logistic classification and multi class classification models along with the matrix representation of a neural network generalized to *l* layers with *n* neurons

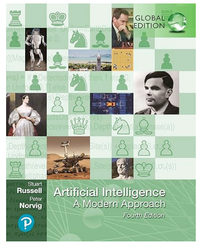
Forward and Back Propagation in Neural Networks

A deep dive into how Neural Networks are trained using Gradient Descent. Output predictions, are compared to observations to calculate loss and Backward propagation then computes gradients by working backward through the network

Gradient Descent for Multiple Regression

Gradient Descent algorithm for multiple regression and how it can be used to optimize k + 1 parameters for a Linear model in multiple dimensions.

Recommended Textbooks



Artificial Intelligence: A Modern Approach
by Peter Norvig, Stuart Russell

For a complete list of tutorials see:
https://arrsingh.com/ai-tutorials