# Natural Language Processing with PyTorch

IMPLEMENTING RECURRENT NEURAL NETWORKS (RNNS) IN PYTORCH


**Janani Ravi**
CO-FOUNDER, LOONYCORN
www.loonycorn.com

# Overview

Modifying neurons to endow them with state and memory

Understand Recurrent Neural Networks (RNNs)

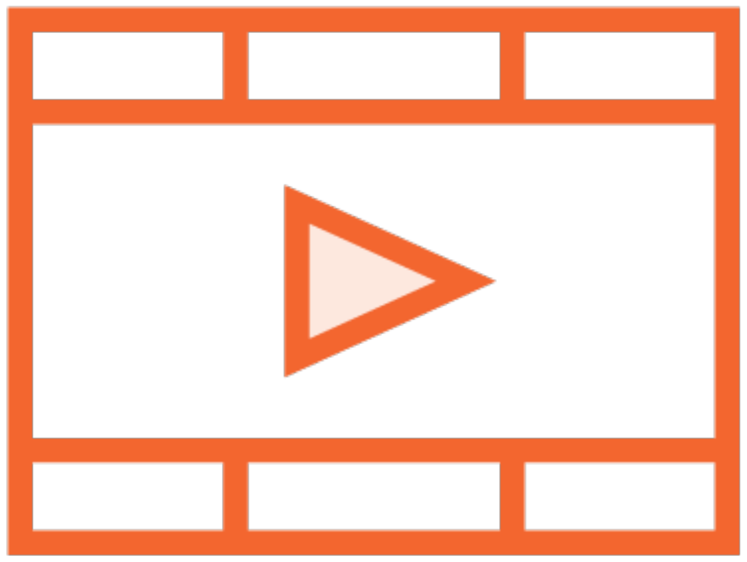
Mitigate problems of vanishing and exploding gradients in training RNNs

Working with long-memory recurrent cells

Use RNNs in language modeling

# Prerequisites and Course Outline

# Prerequisites



Comfortable programming in Python

Good understanding of neural networks

Used PyTorch to build and train neural networks

# Prerequisite Courses

**Foundations of PyTorch**

**Building Your First PyTorch Solution**

**Image Classification with PyTorch**

# Course Outline

- Recurrent neural networks (RNNs)
- Binary classification using words
- Multi-class classification using characters
- Sentiment analysis using word embeddings
- Sequence-to-sequence models

# RNNs and Natural Language Processing

$$y = f(x)$$

# Machine Learning

**Machine learning algorithms seek to “learn” the function f that links the features and the labels**

```
y = Wx + b
```

$$f(x) = Wx + b$$

**Linear regression specifies, up-front, that the function f is linear**

```
def doSomethingReallyComplicated(x1,x2…):
  …
  …
  …
  return complicatedResult
```

$$f(x) = doSomethingReallyComplicated(x)$$

**ML algorithms such as neural network can “learn” (reverse-engineer) pretty much anything given the right training data**

Sometimes **time** relationships in data have special meaning

$$y_t = f(x_t, y_{t-1})$$

# Learning the Past

**Relationships where past values of the effect variable drive current values are called auto-regressive**

$$y_t = f(x_t, y_{t-1})$$

# Learning the Past

**The output at one time instance depends on the current input at that time instance**

$$y_t = f(x_t, y_{t-1})$$

# Learning the Past

**And on the output from the previous time instance**

Feed-forward networks cannot learn from the past

**Recurrent neural networks can**

# Text Is Sequential Data

**Predict the next word in a sequence (autocomplete)**

**“The tallest building in the world is …”**

**Language translations**

**“how are you” -> “Comment allez-vous”**

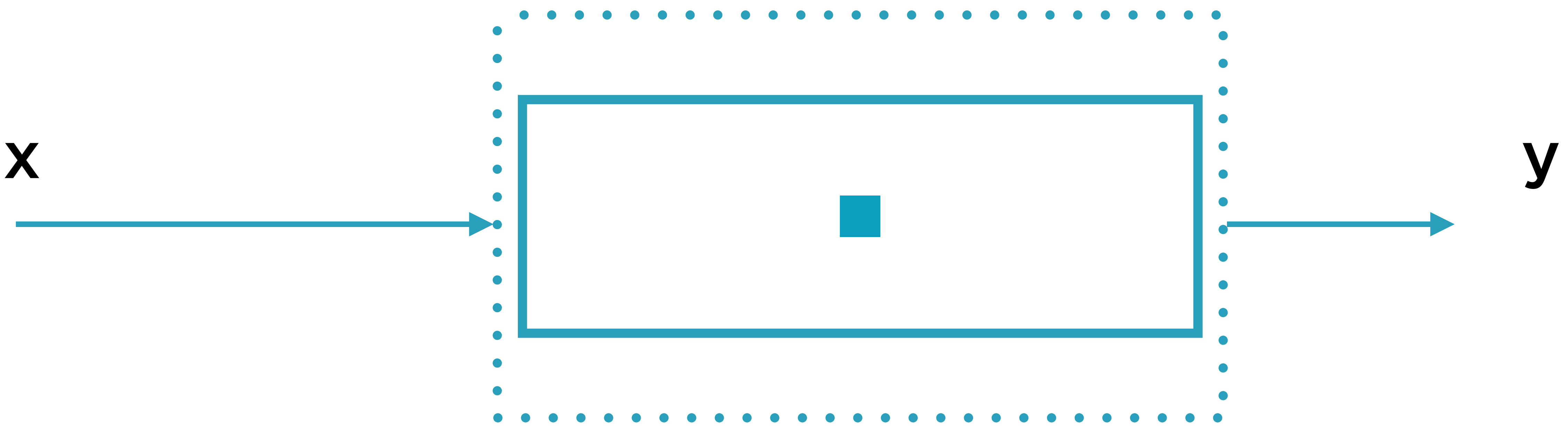
**Text classification, sentiment analysis, natural language processing**

**“This is not the worst restaurant not by a long way”**

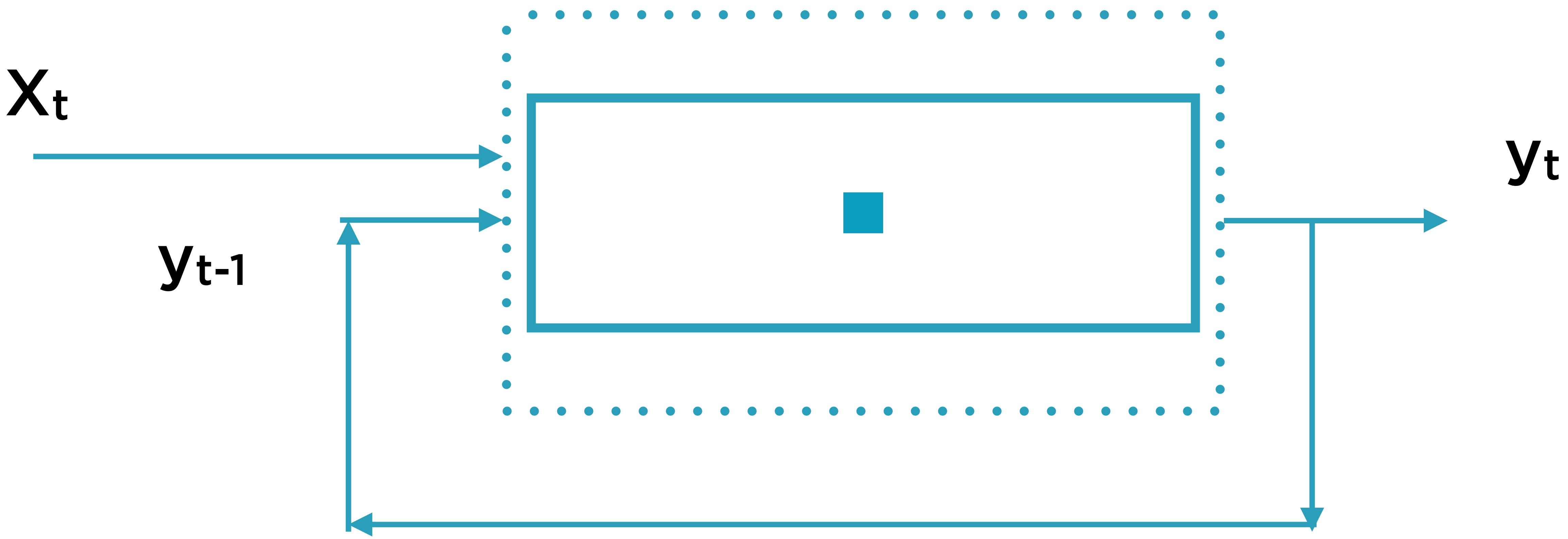
RNNs are great at learning sequential data

# Recurrent Neurons

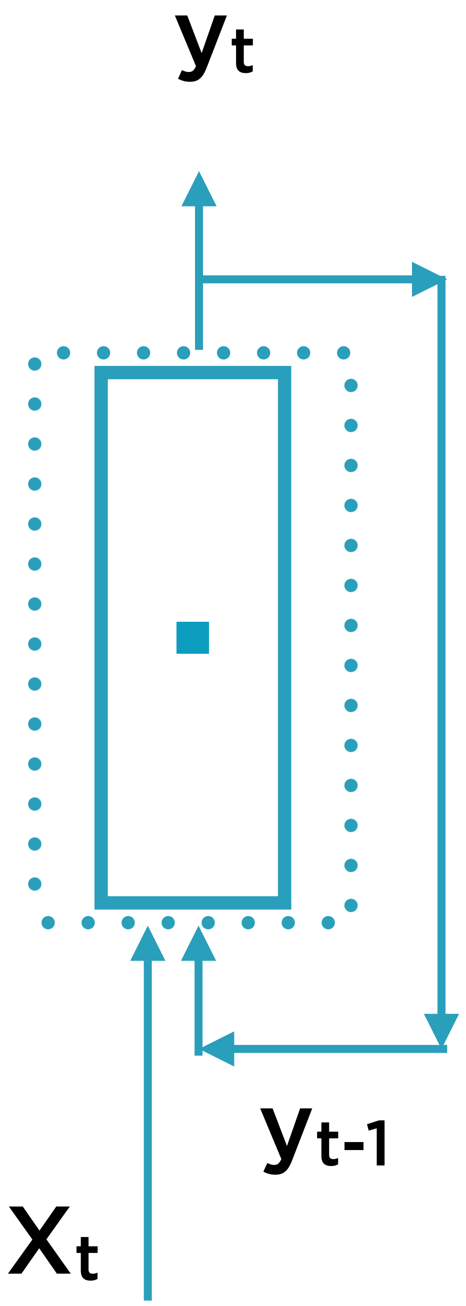
# Simplest Feed-forward Neuron



# Simplest Recurrent Neuron



# Recurrent Neuron

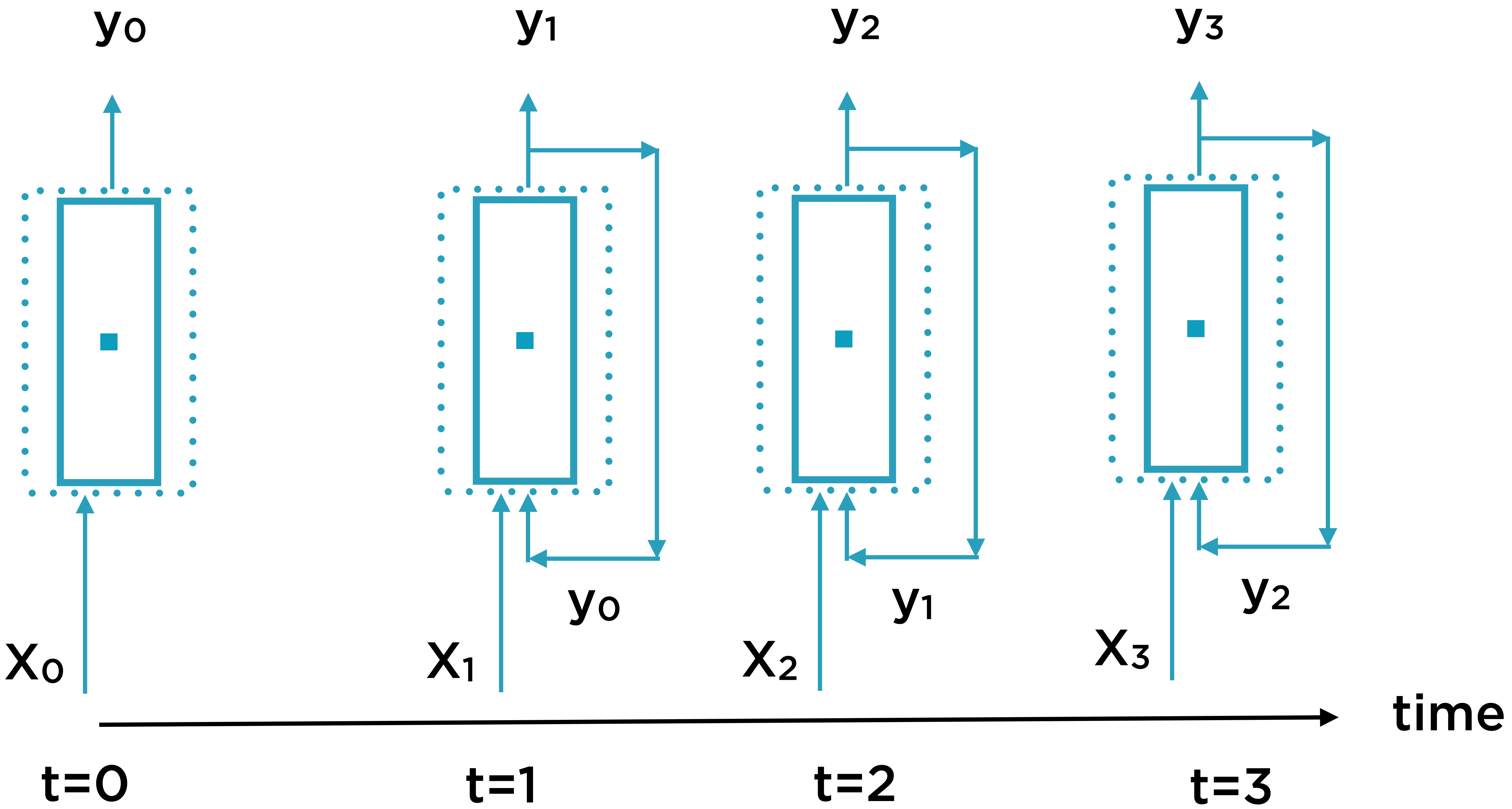



**$y_t$ = Output at time t**

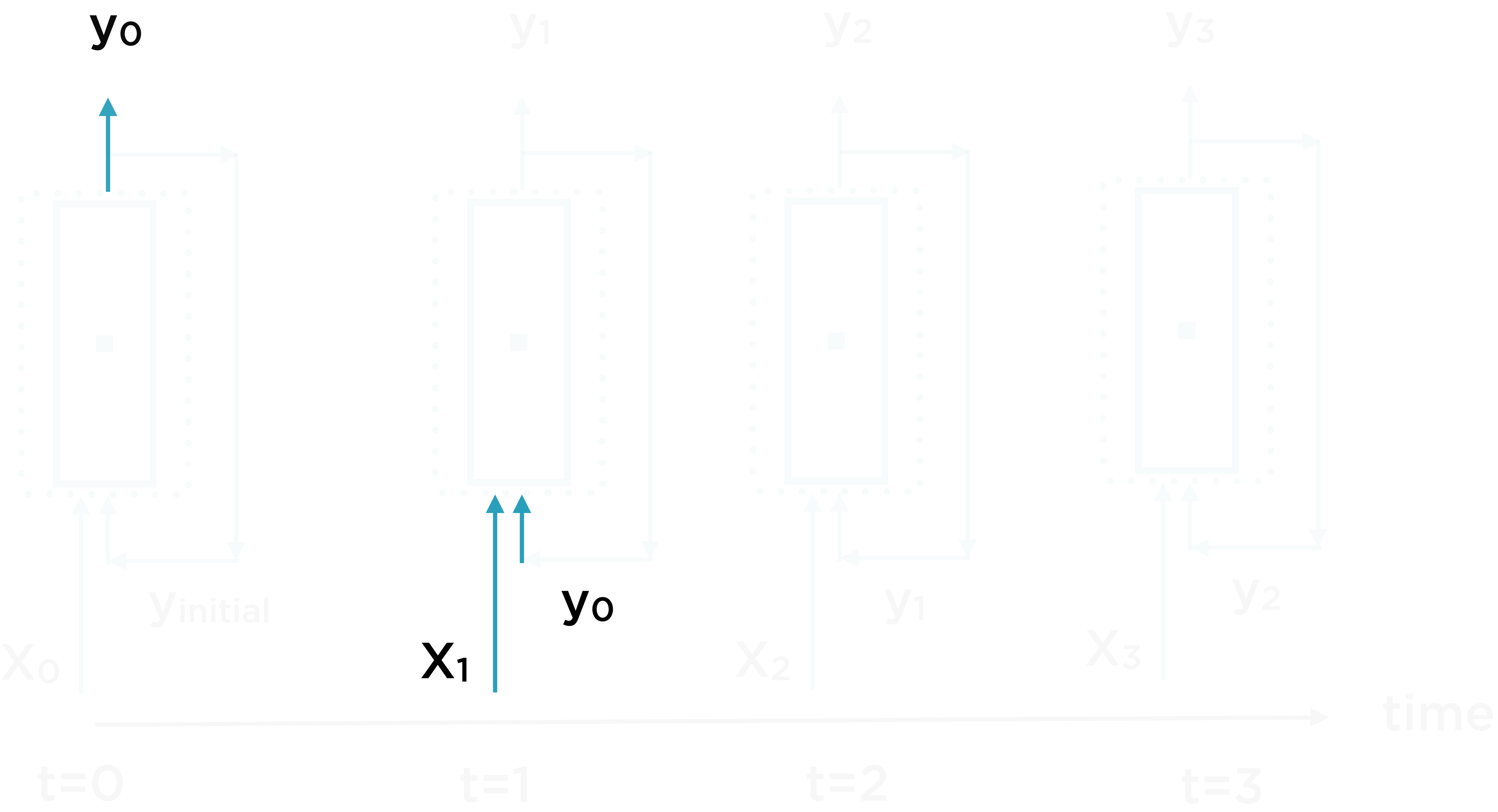
**Depends upon**

- $y_{t-1}$ = Output at time t - 1
- $x_t$ = New inputs available only at time t

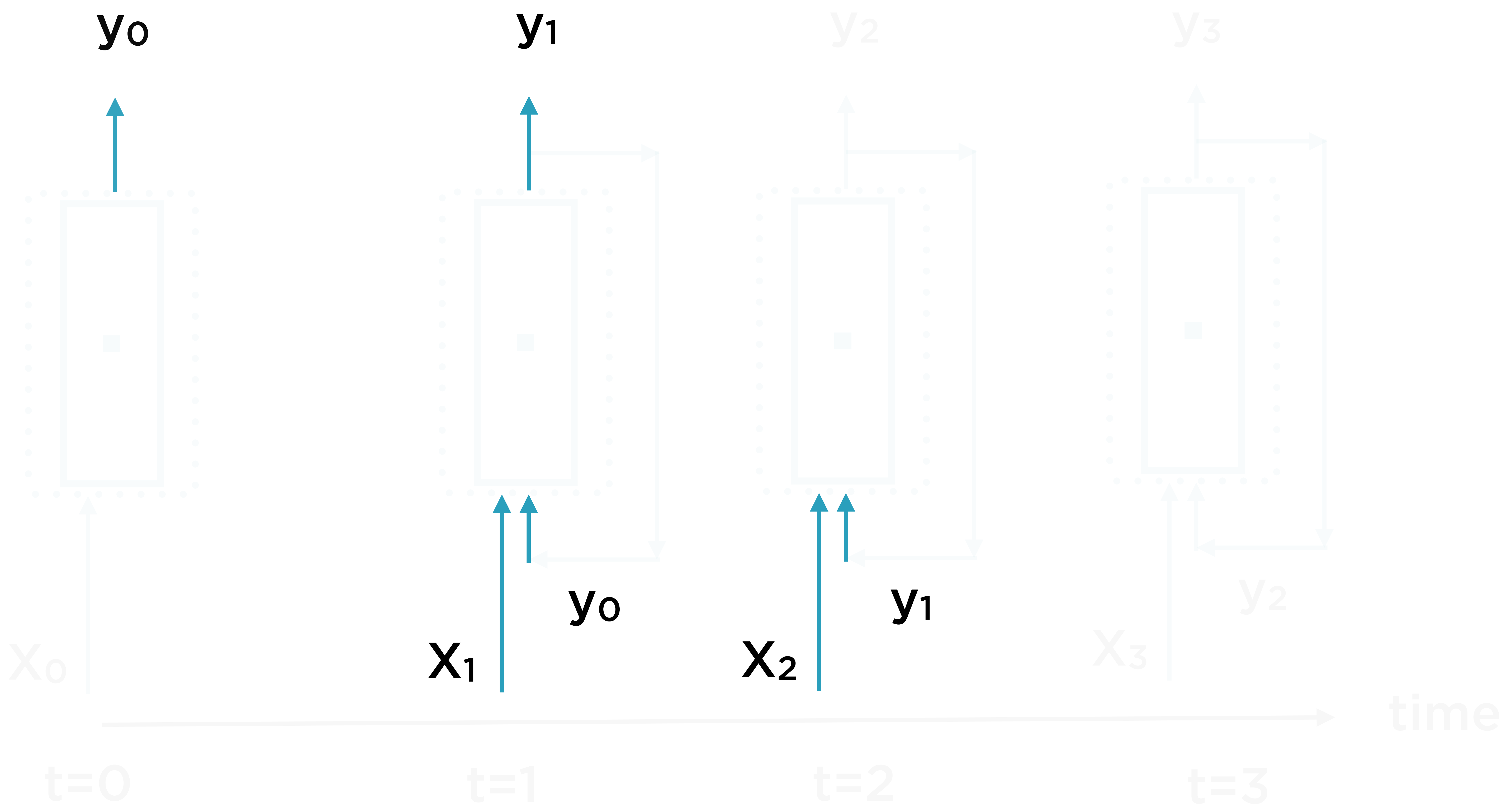
# Unrolling Through Time



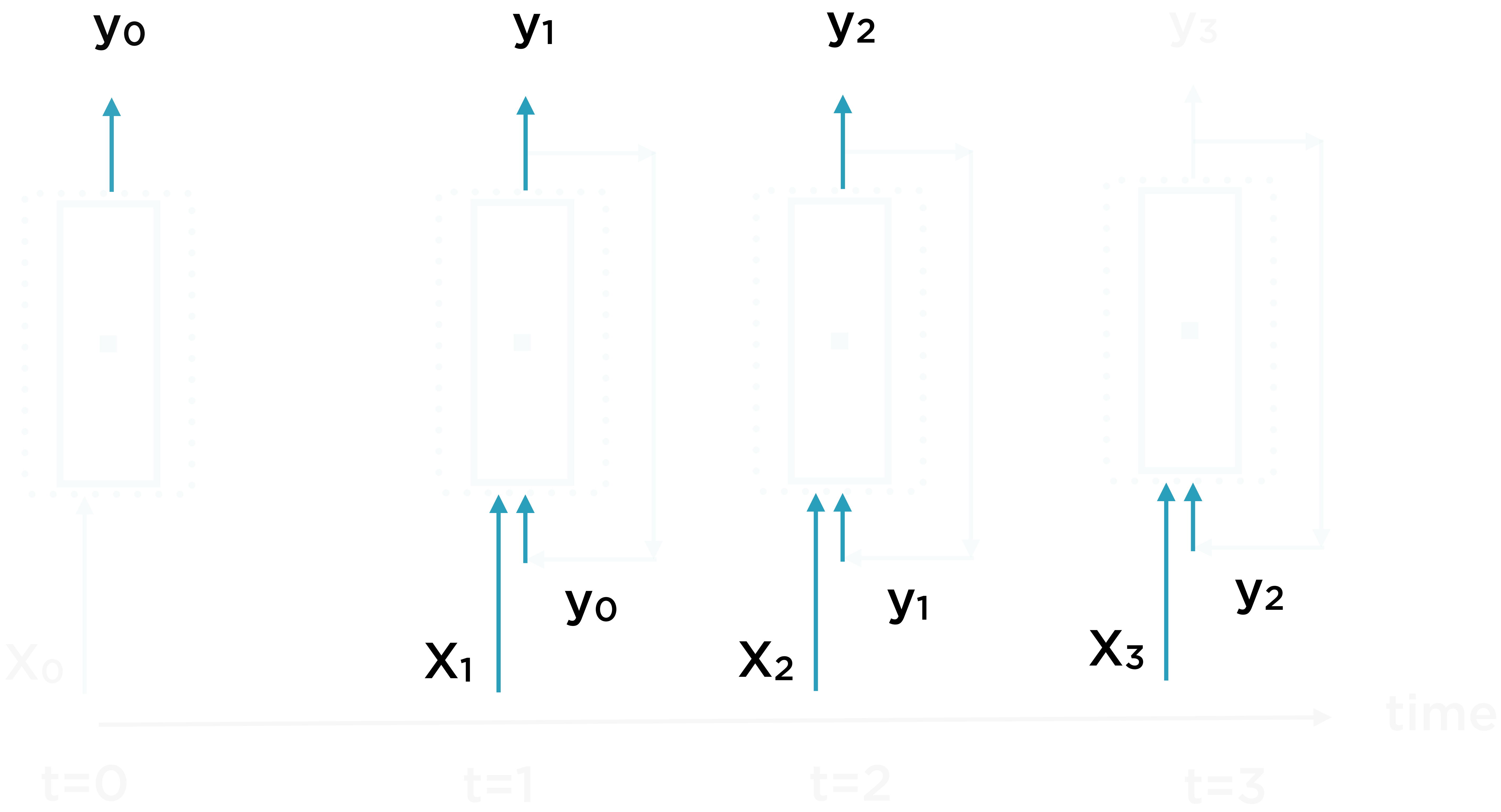
# Unrolling Through Time



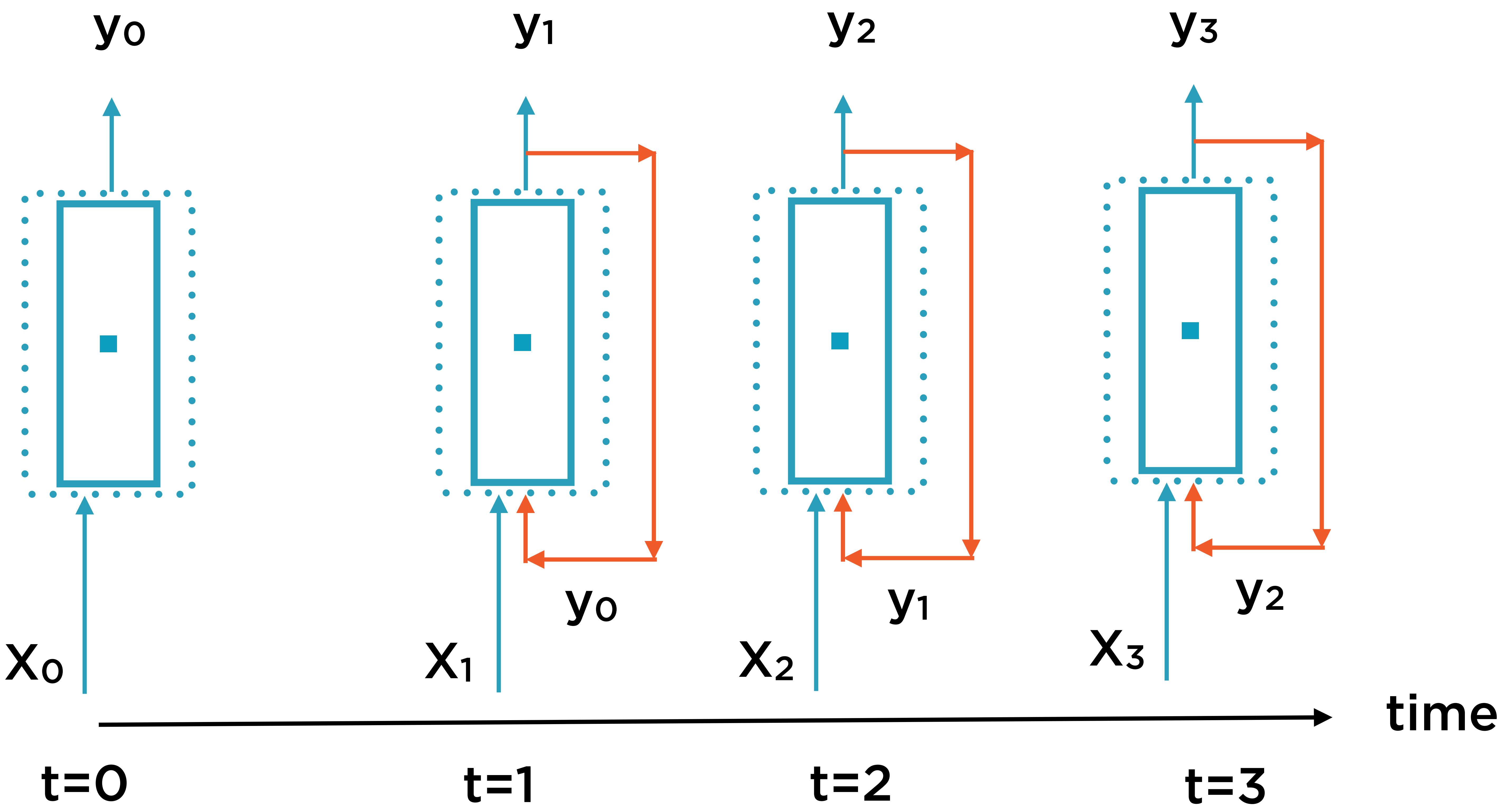
# Unrolling Through Time



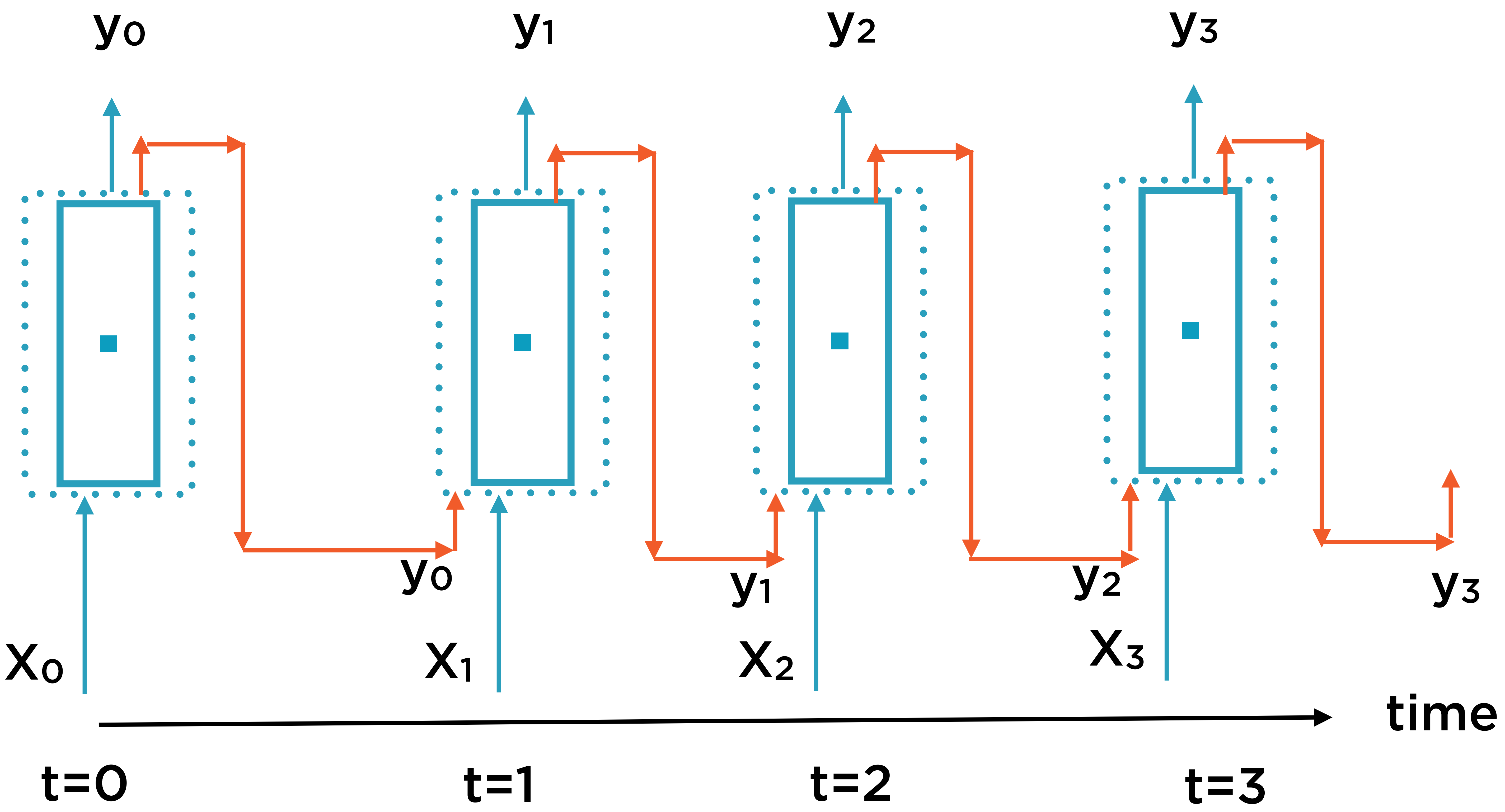
# Unrolling Through Time



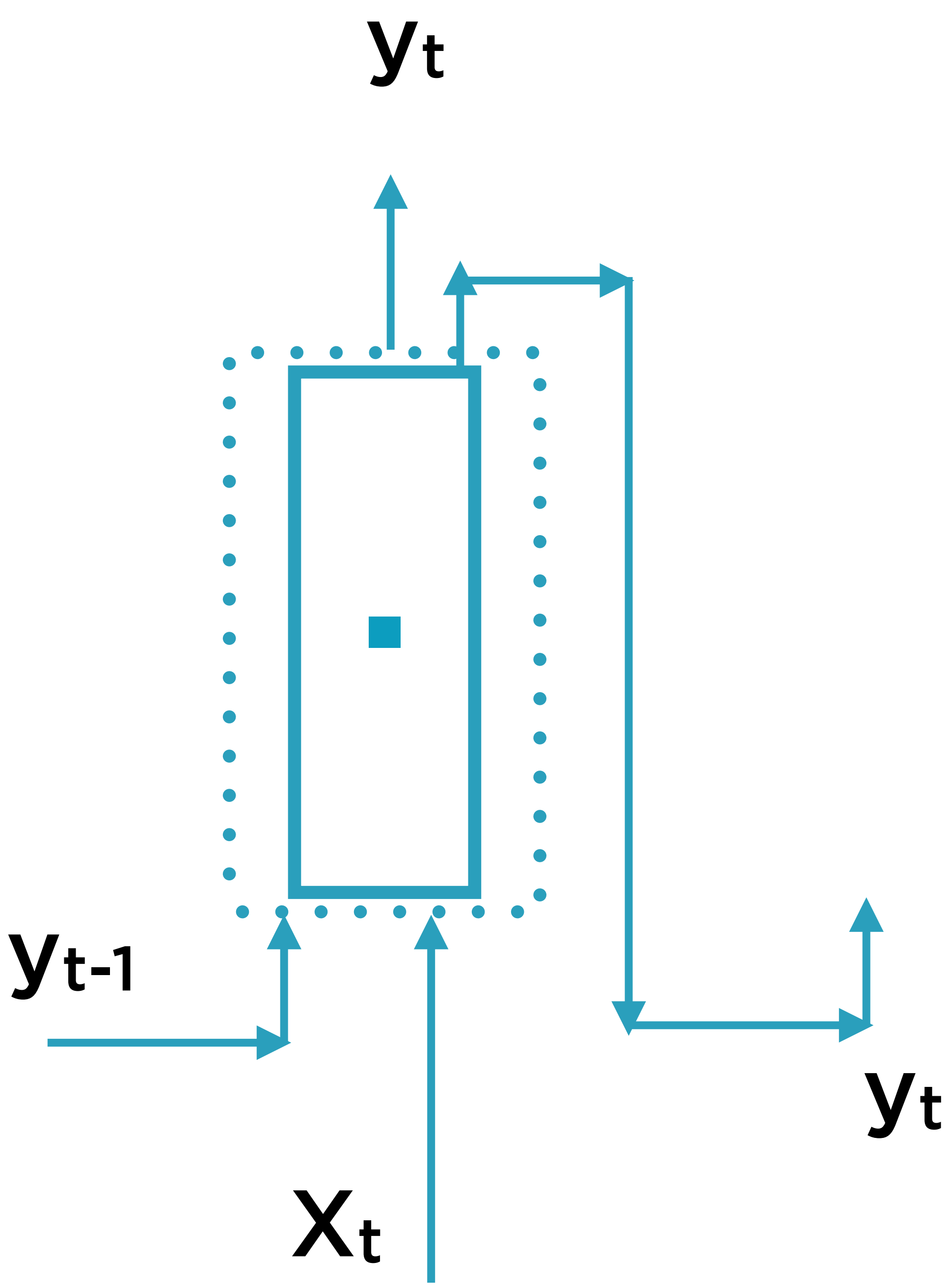
# Unrolling Through Time



# Output of a Layer Fed to Next Layer



# Recurrent Neuron




**Regular neuron: input is feature vector, output is scalar**

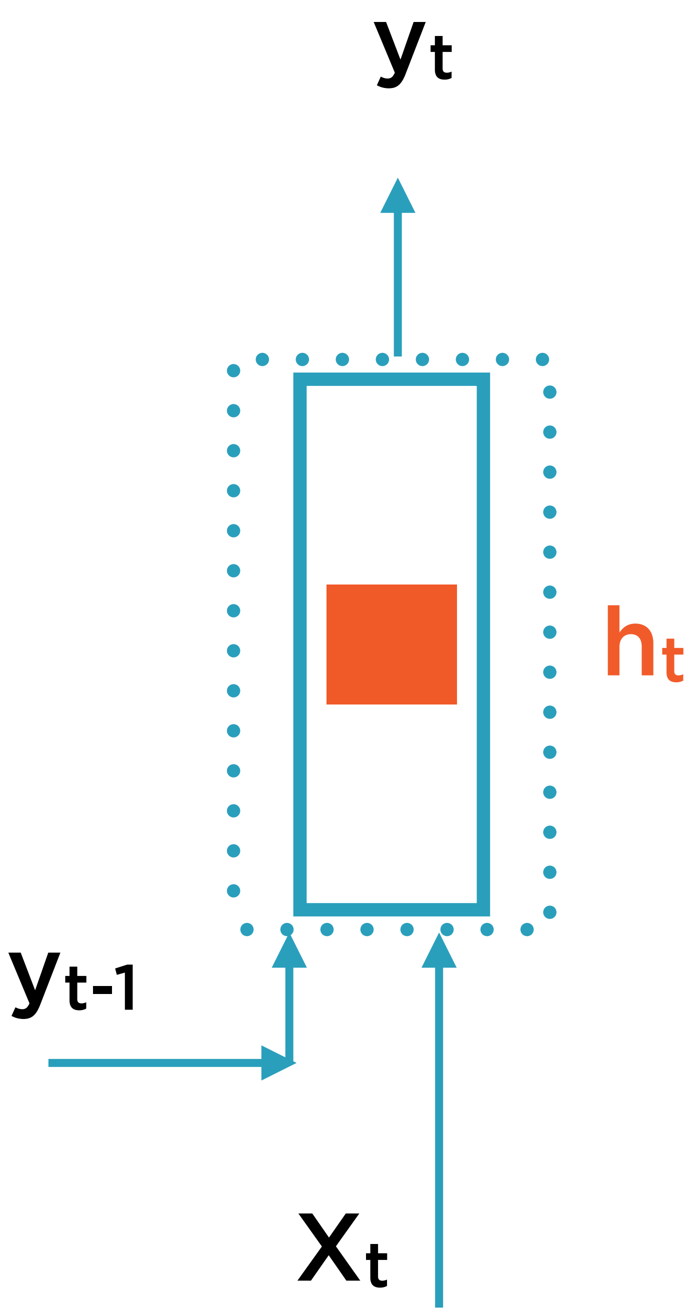
$Y = Wx + b$

**Recurrent neuron: output is vector too**

**Input:** $[X_0, X_1, \ldots X_t]$

**Output:** $[Y_0, Y_1, \ldots Y_t]$

# Memory and State



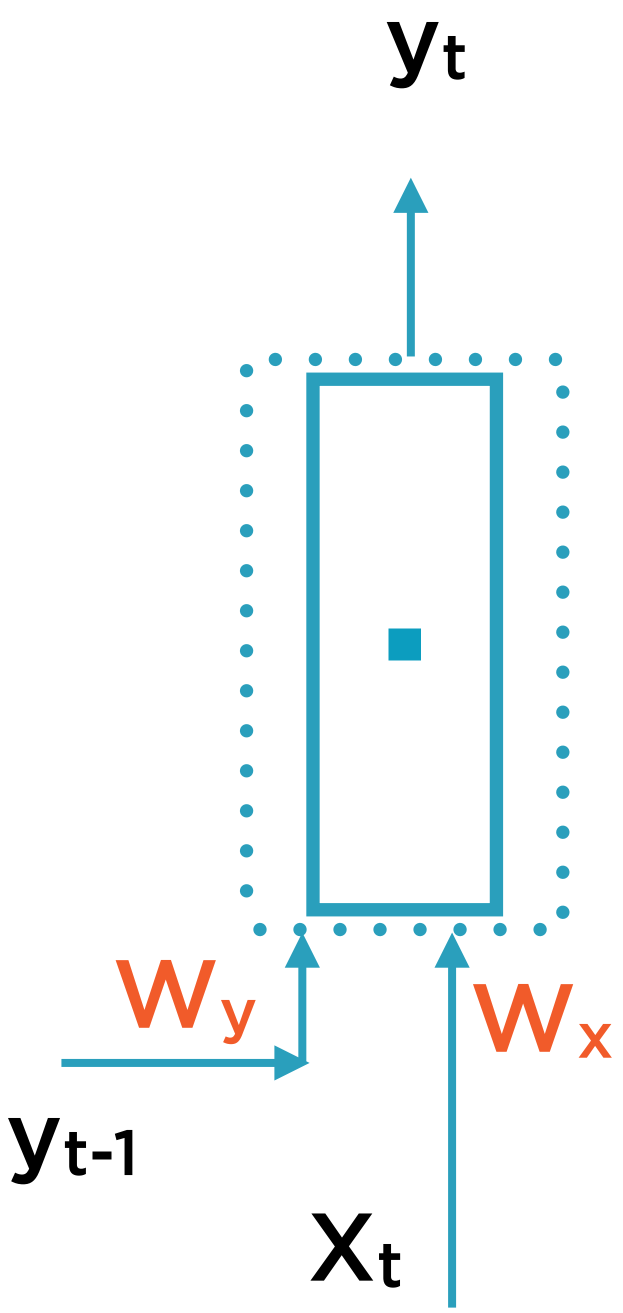

Recurrent neurons remember the past

They possess 'memory'

The stored state could be more complex than simply $y_{t-1}$

The internal state is represented by $h_t$

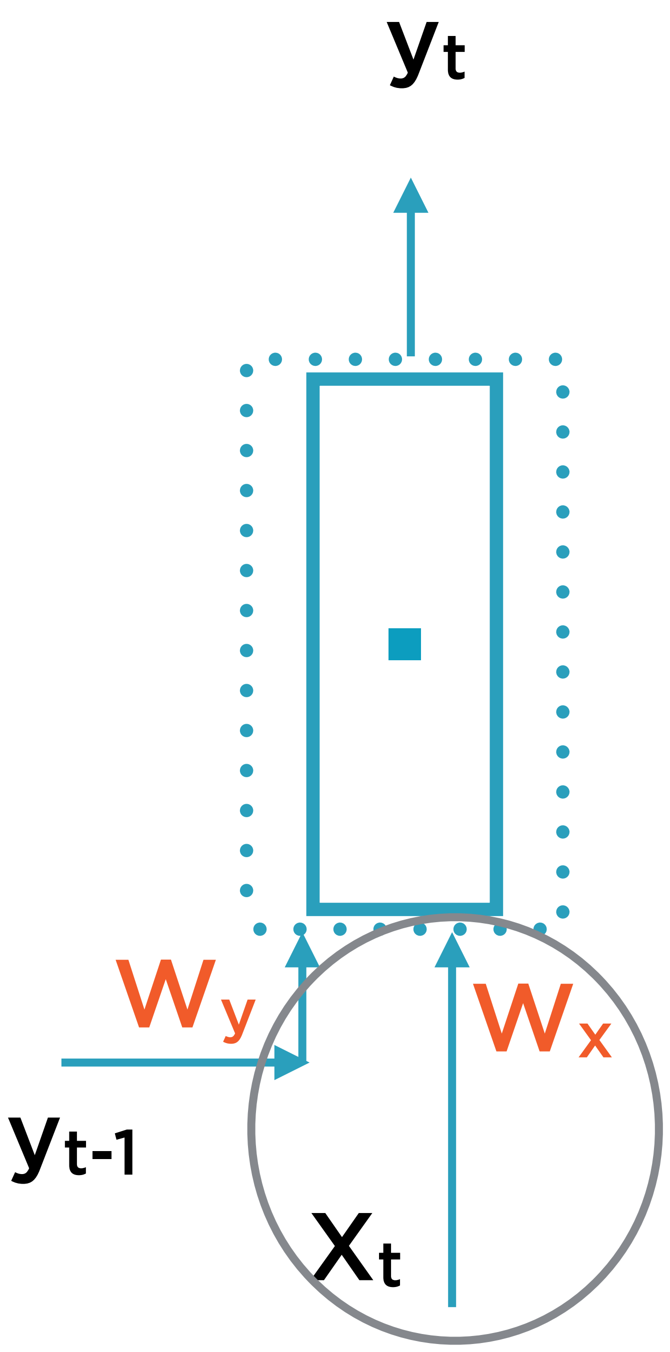
# Recurrent Neuron




Now, each neuron has two weight vectors

$W_x, W_y$

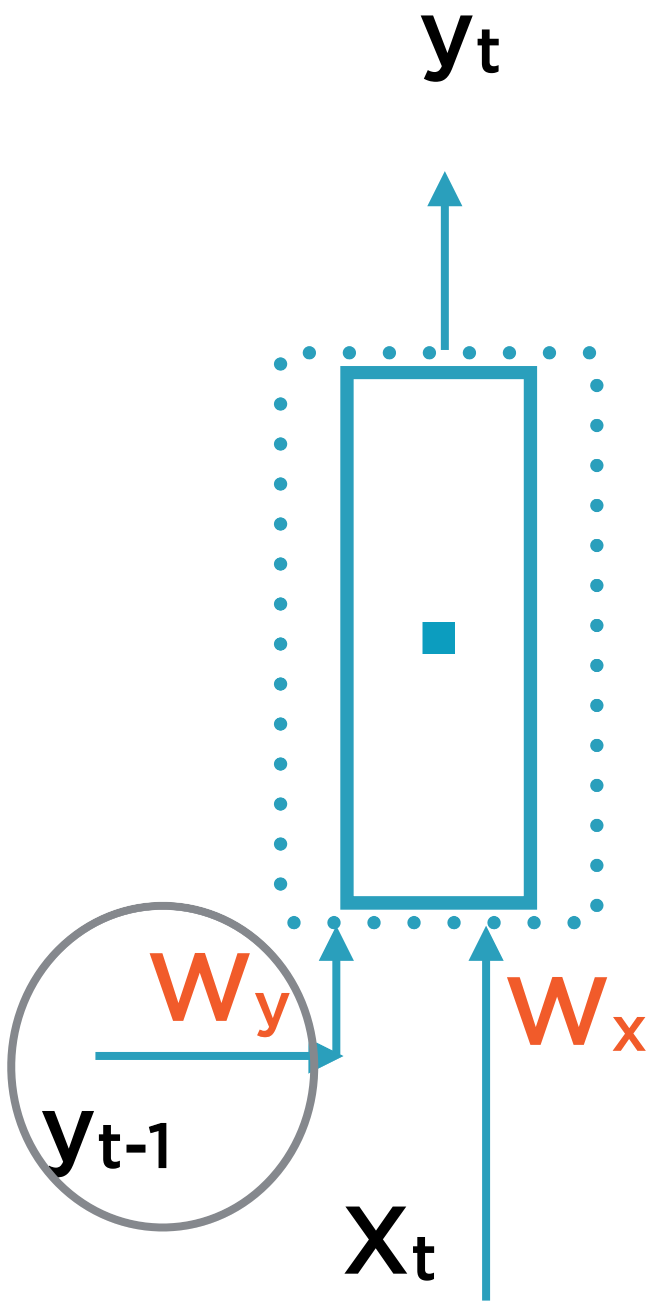
# Recurrent Neuron




Now, each neuron has two weight vectors

$W_x$, $W_y$

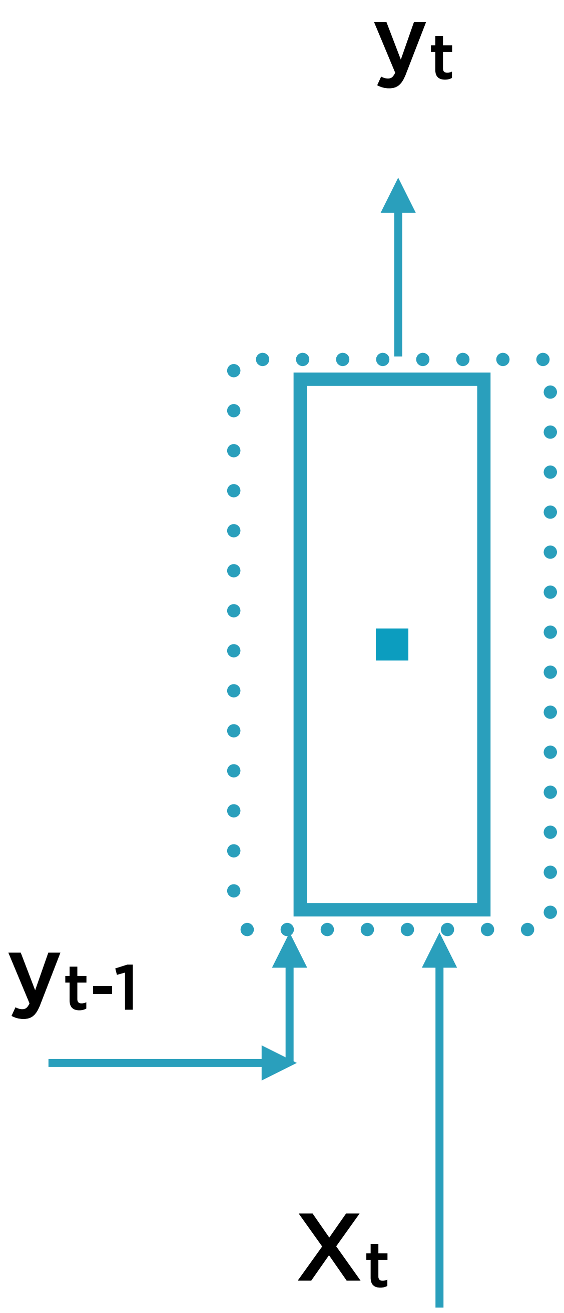
# Recurrent Neuron




Now, each neuron has two weight vectors

$W_x$, $W_y$

# Recurrent Neuron



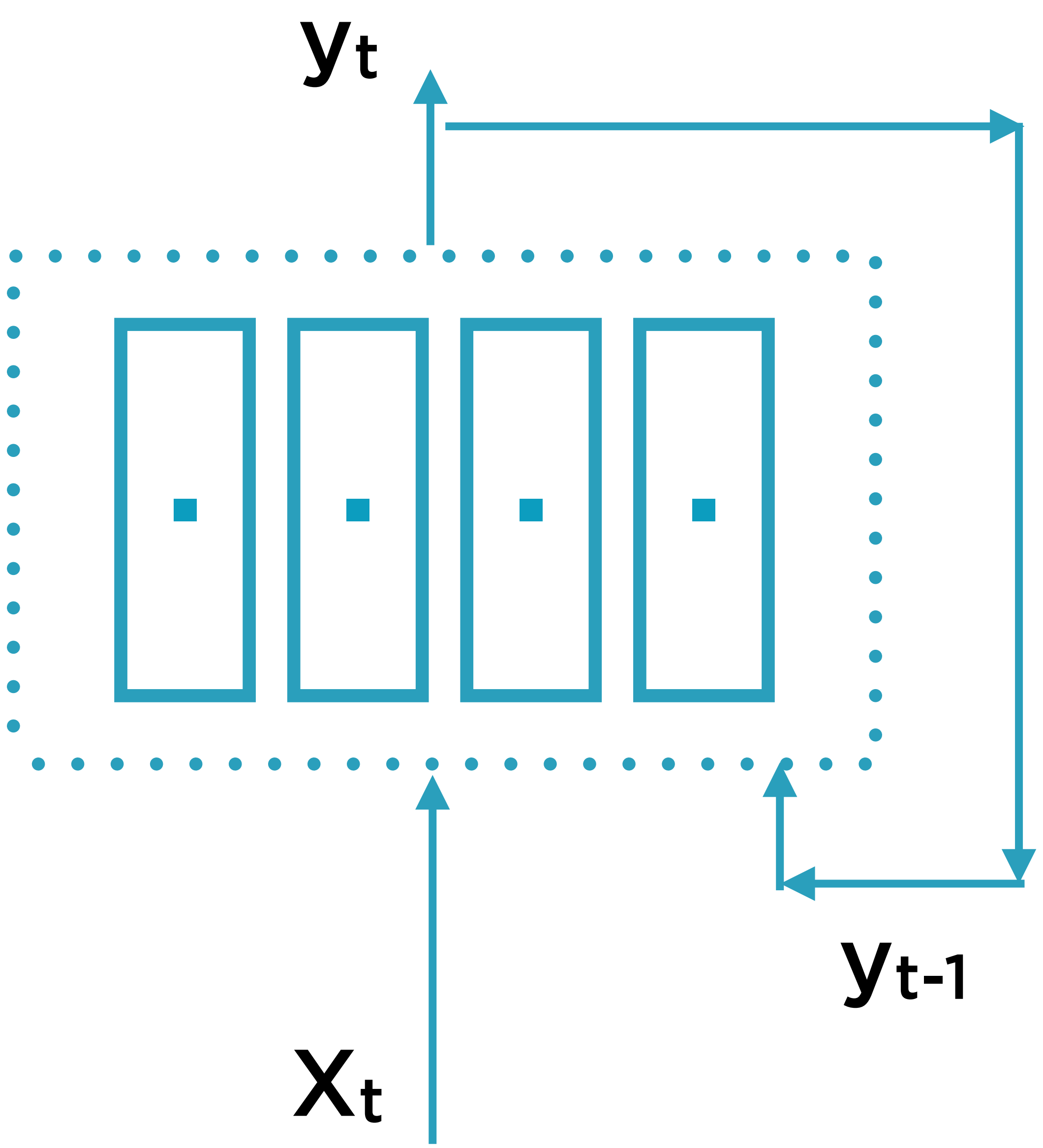

Output of neuron as a whole is given as

$$y_t = \Phi(X_t W_x + y_{t-1} W_y + b)$$

($\Phi$ is the activation function)

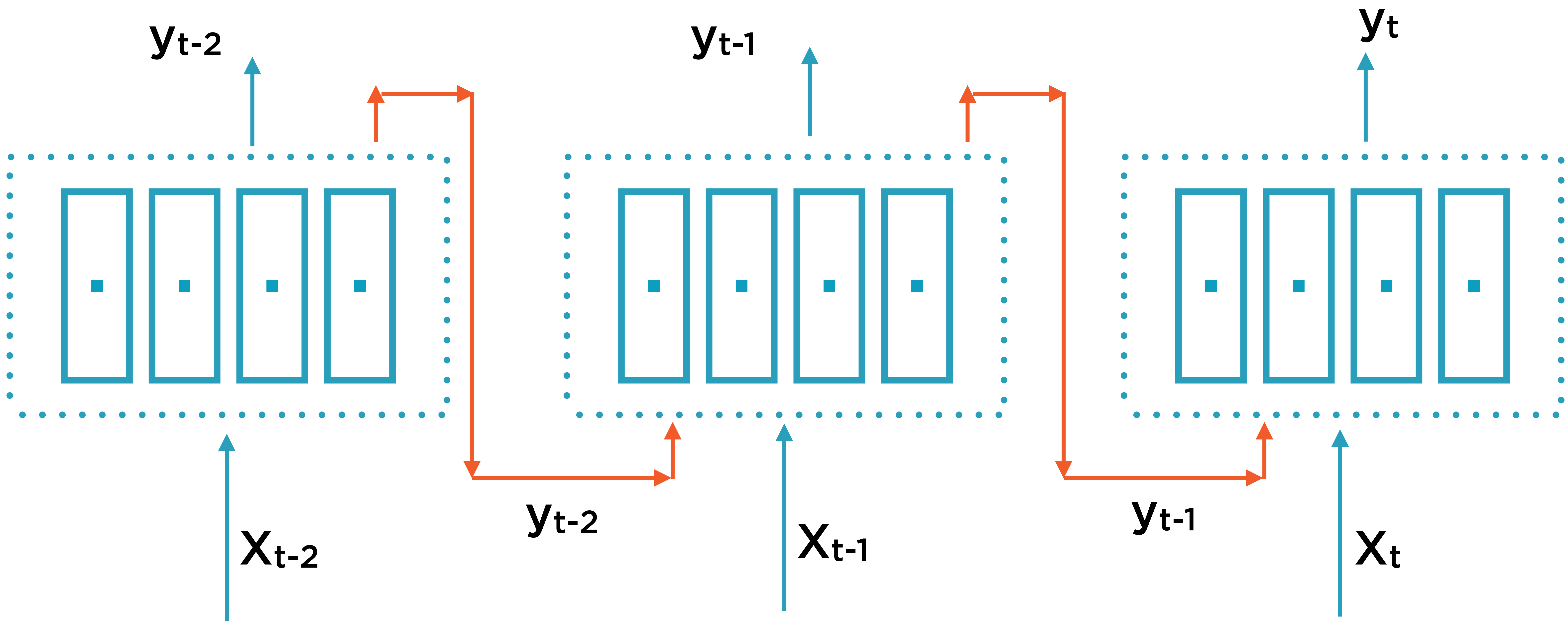
# Training a Recurrent Neural Network

# Layer of Recurrent Neurons




**A layer of neurons forms an RNN cell - basic cell, LSTM cell, GRU cell (more on these later)**

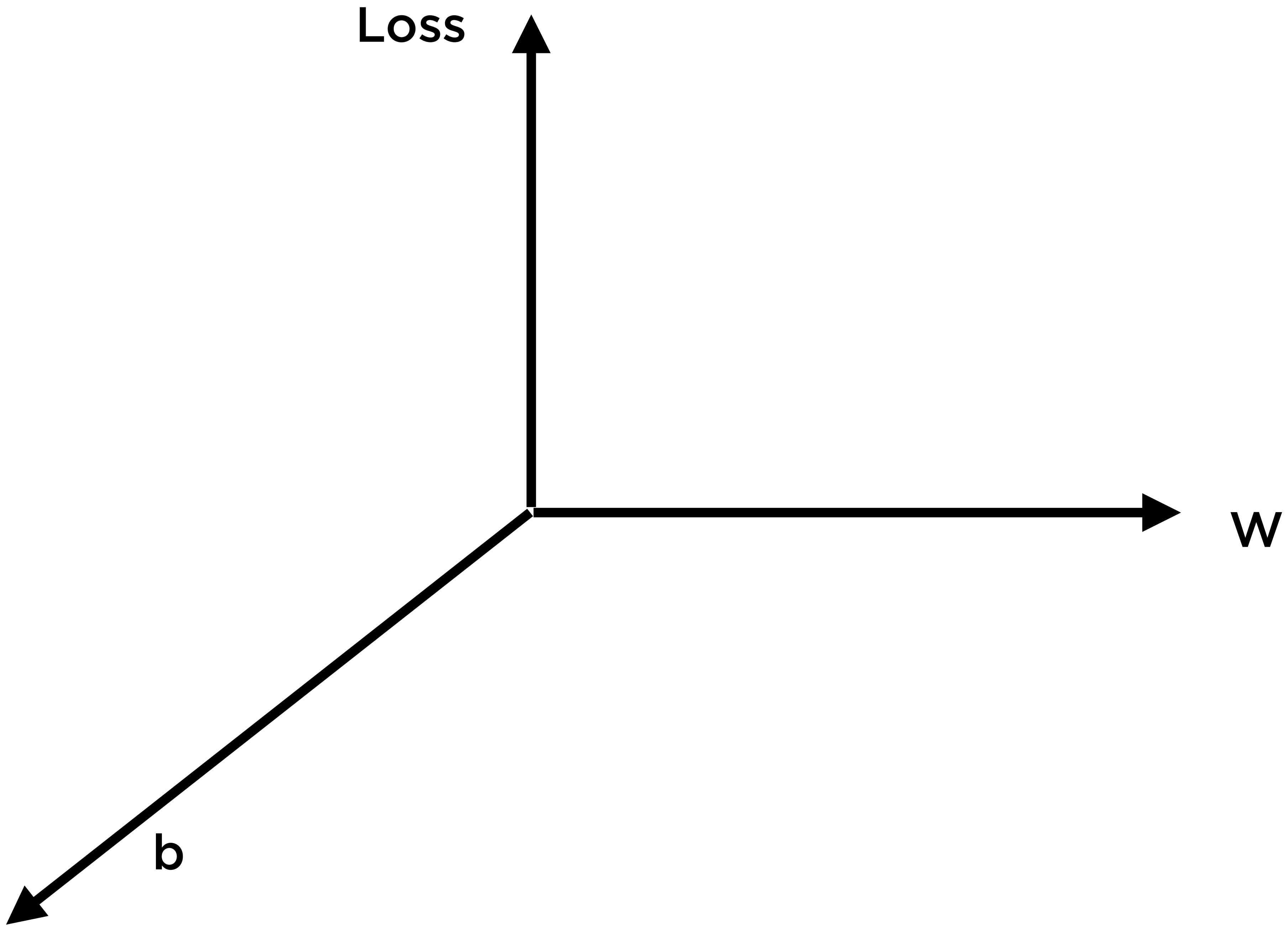
# Layer of Recurrent Neurons



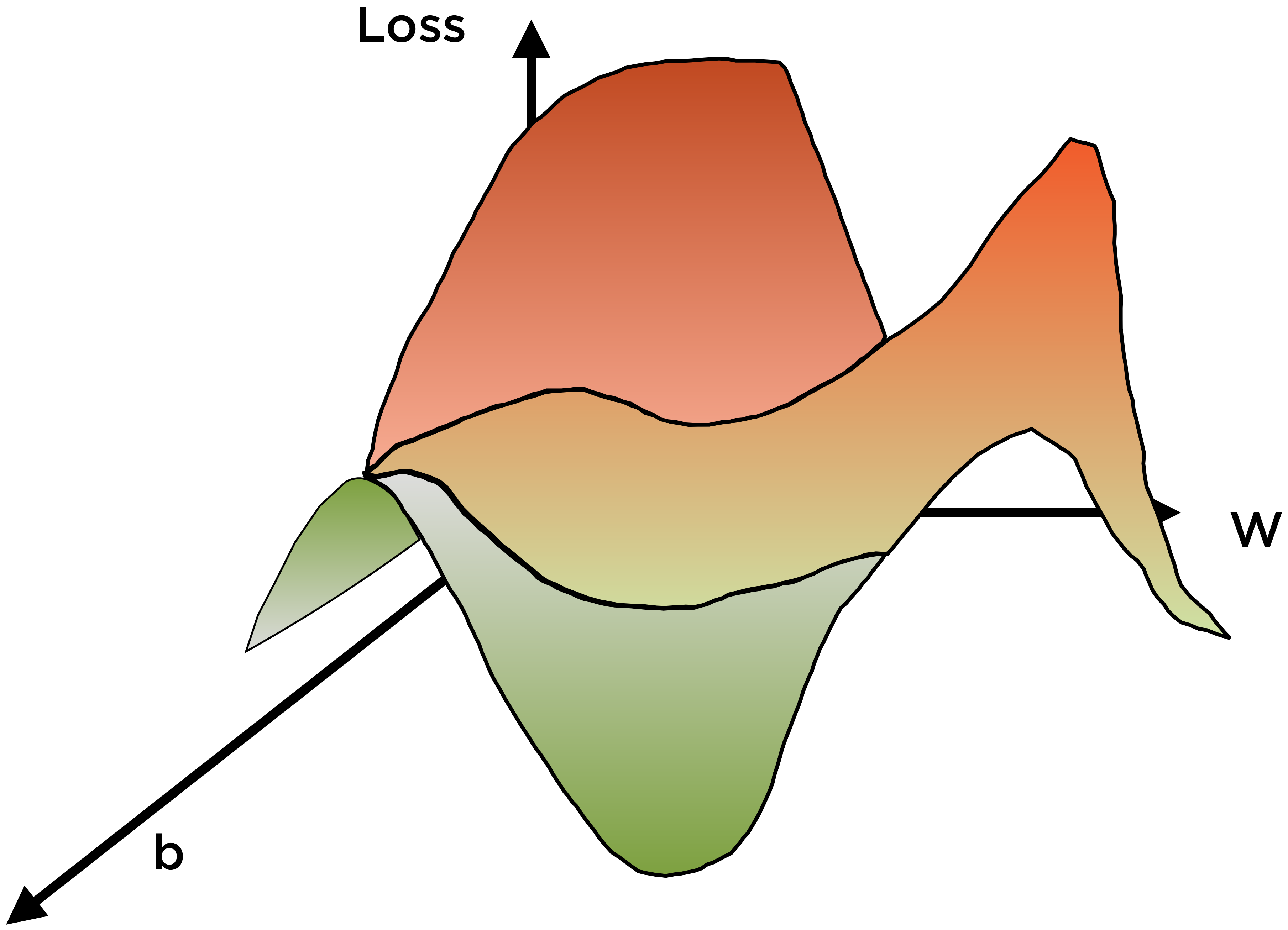

**The cells unrolled through time form the layers of the neural network**

The actual training of a neural network happens via Gradient Descent Optimization

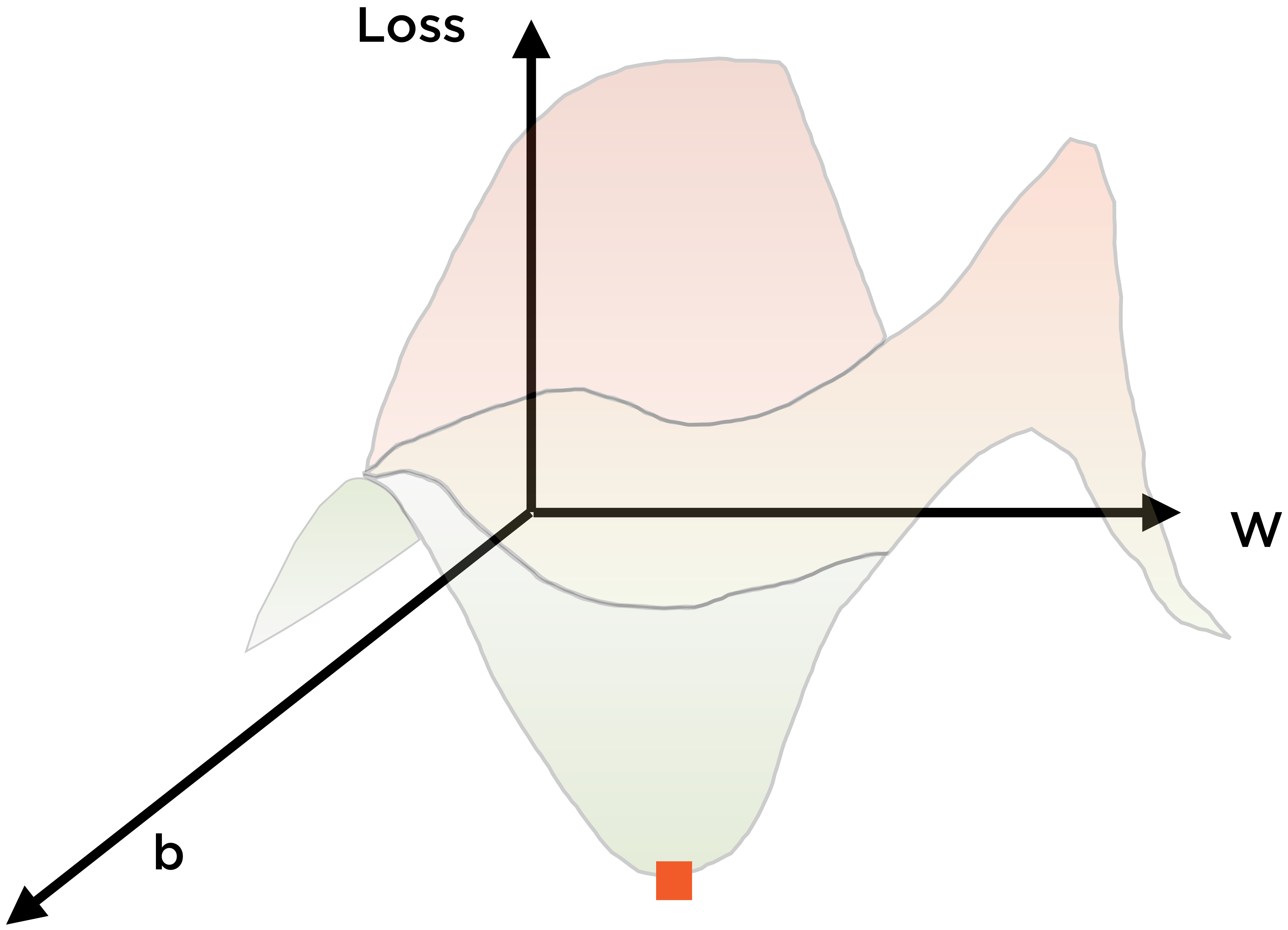
# Minimizing Loss



# Minimizing Loss

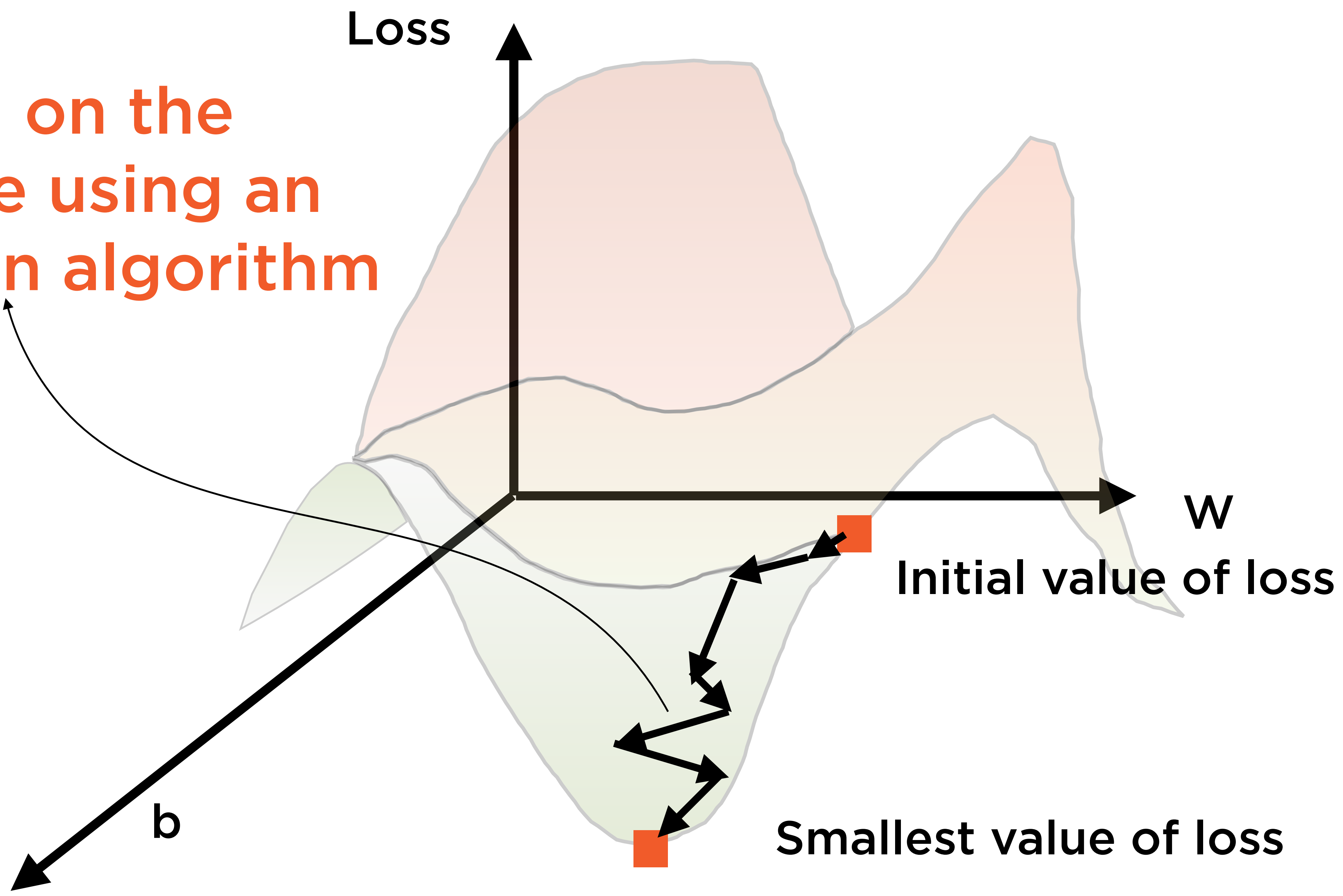


# Minimizing Loss

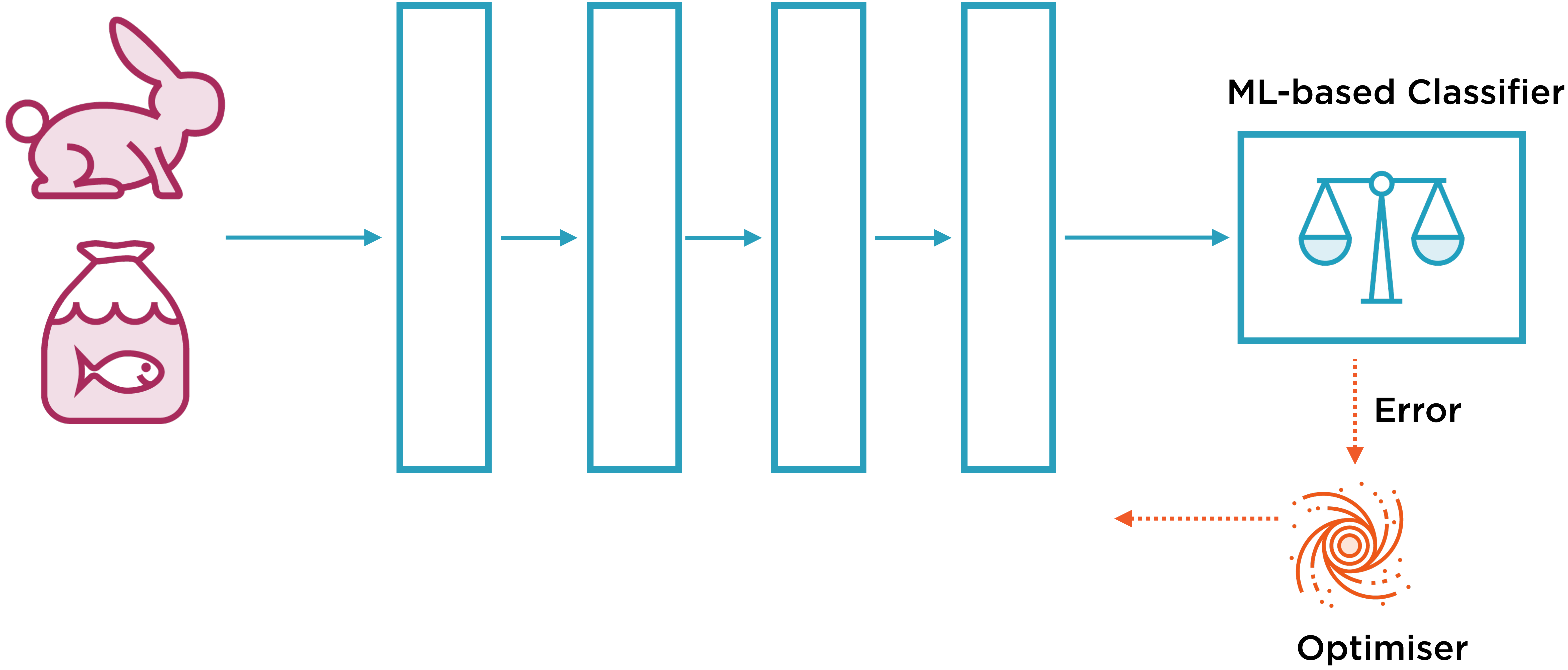


# Gradient Descent

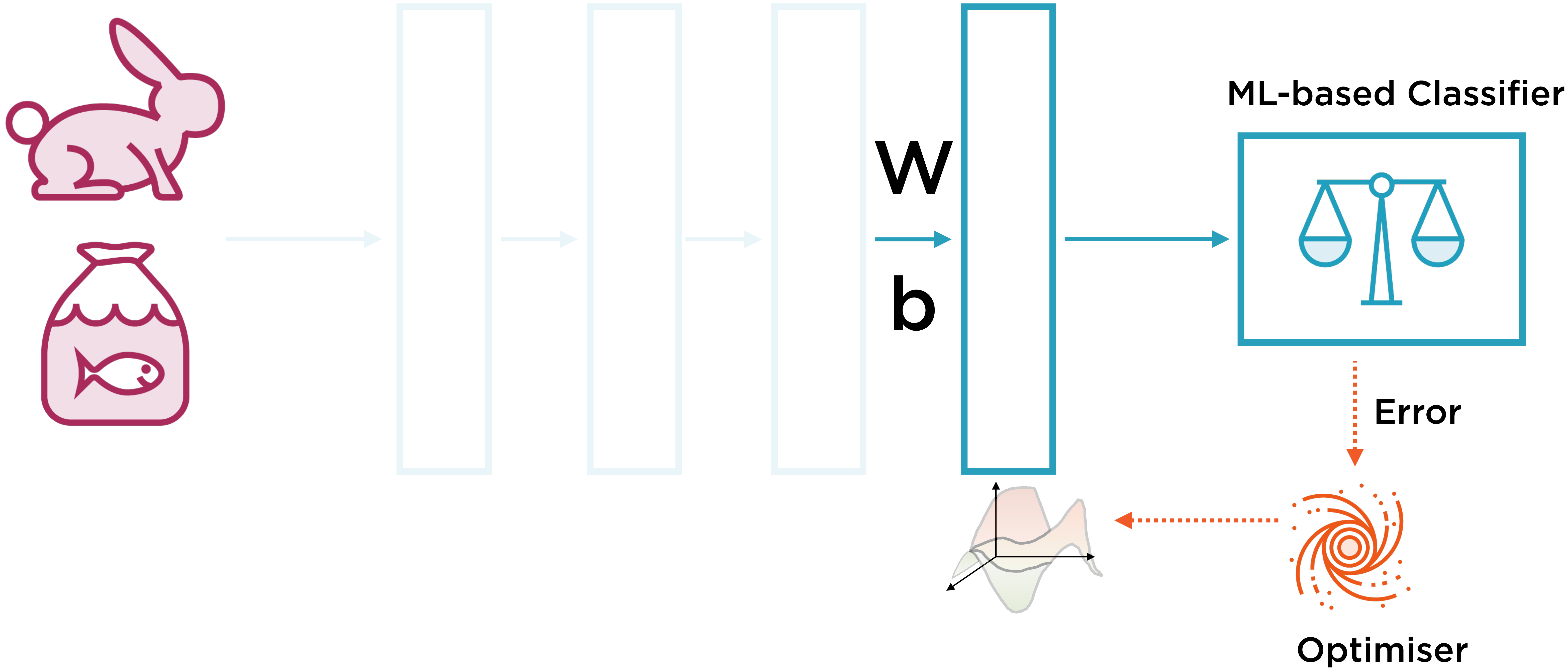
Converging on the "best" value using an optimization algorithm



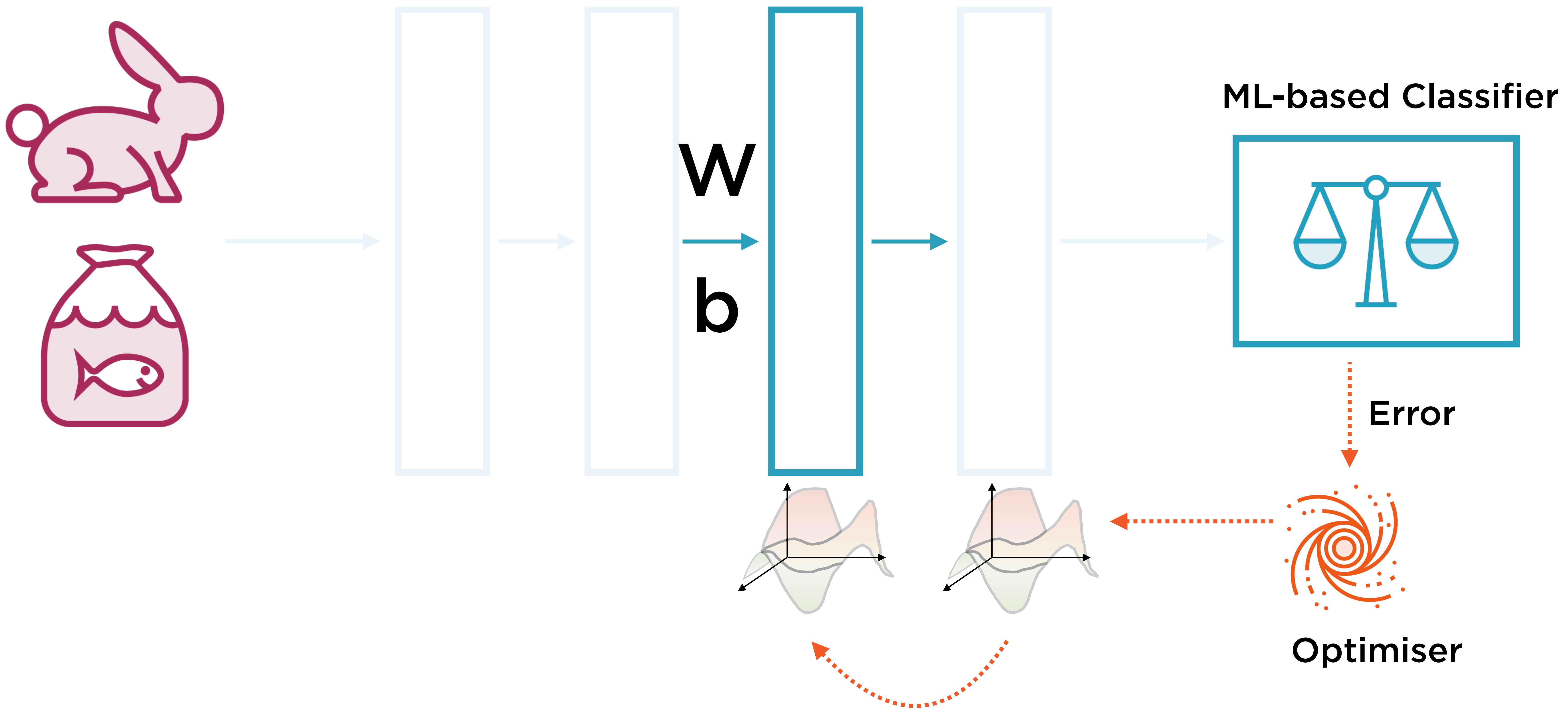
# Back Propagation Through Time



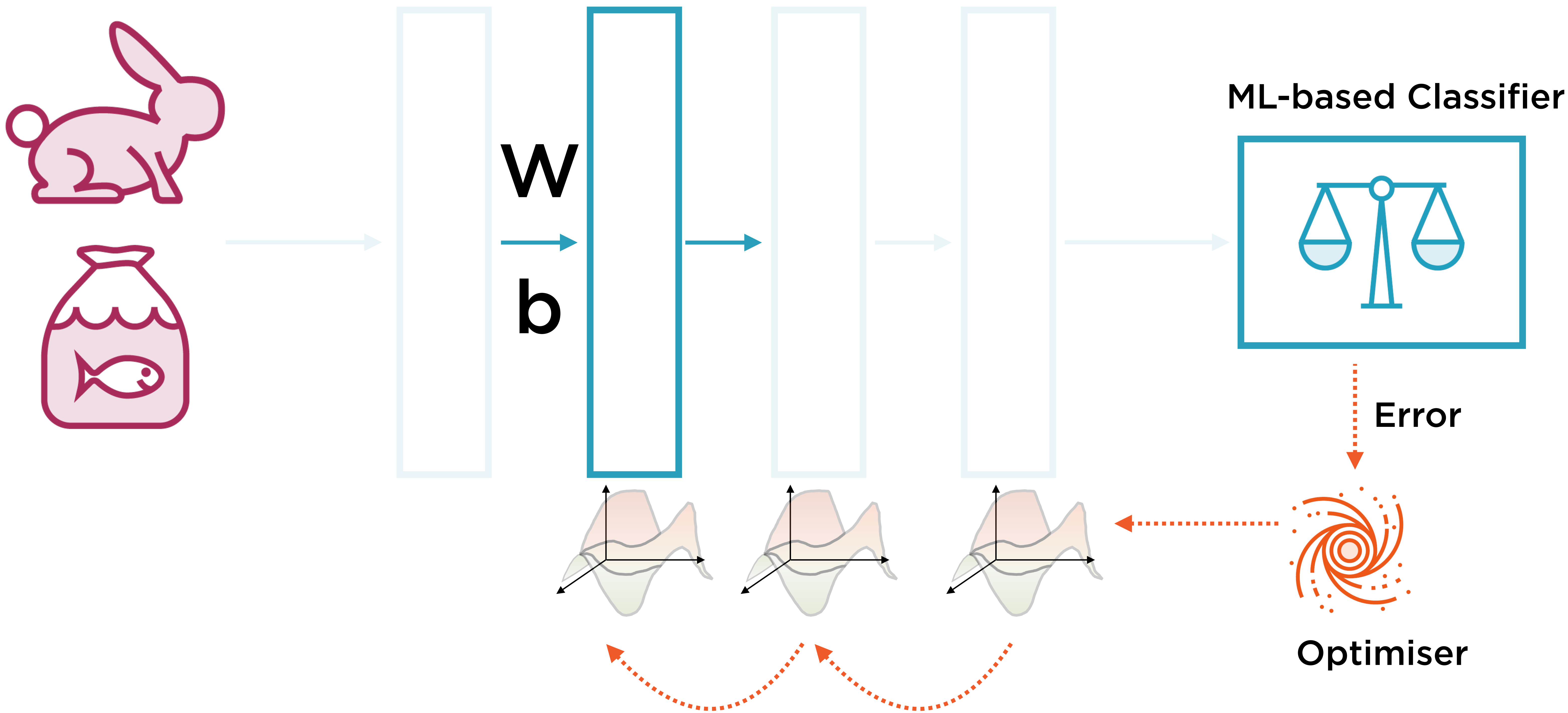
# Back Propagation Through Time



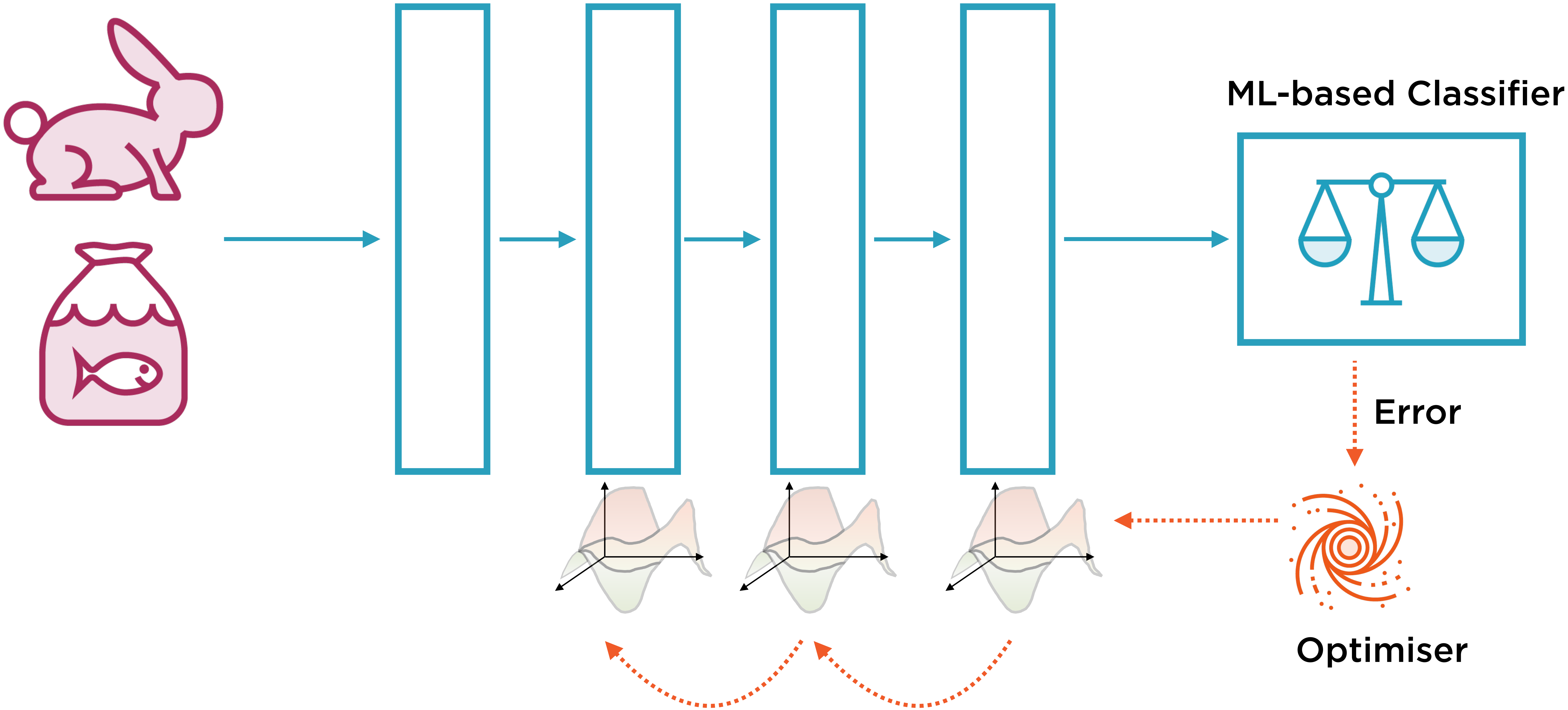
# Back Propagation Through Time



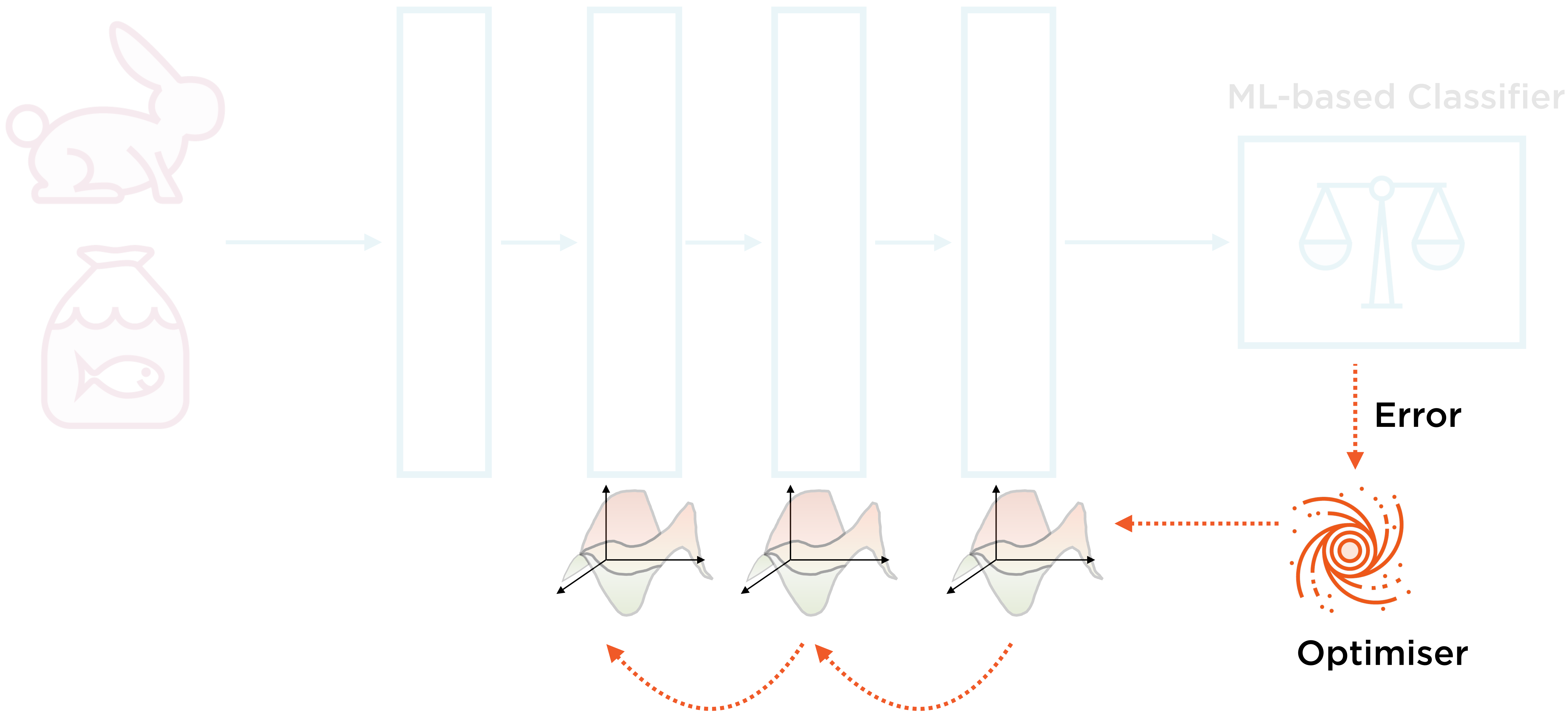
# Back Propagation Through Time



# Back Propagation Through Time

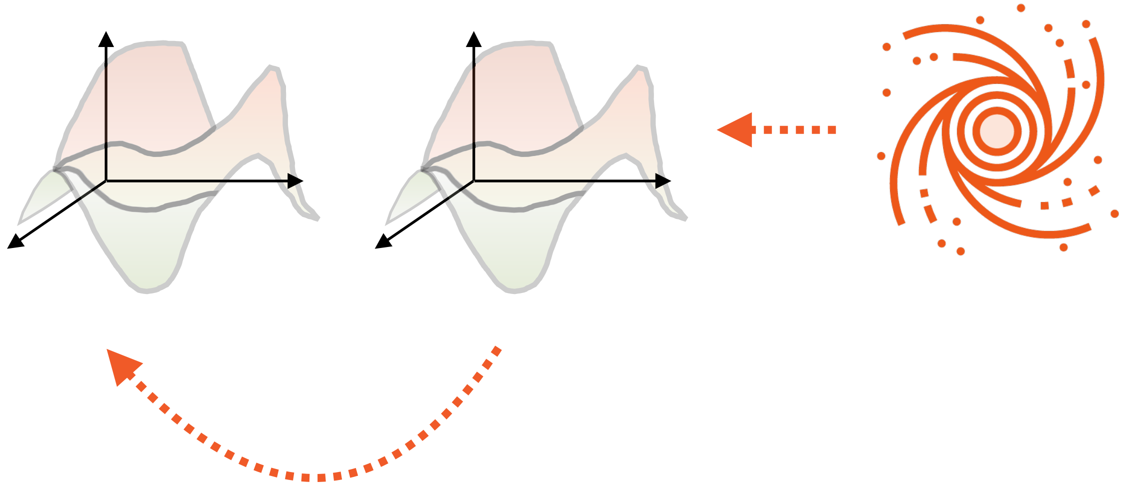


# Back Propagation Through Time



Back propagation allows the weights and biases of the neurons to **converge** to their final values

# Back Propagation

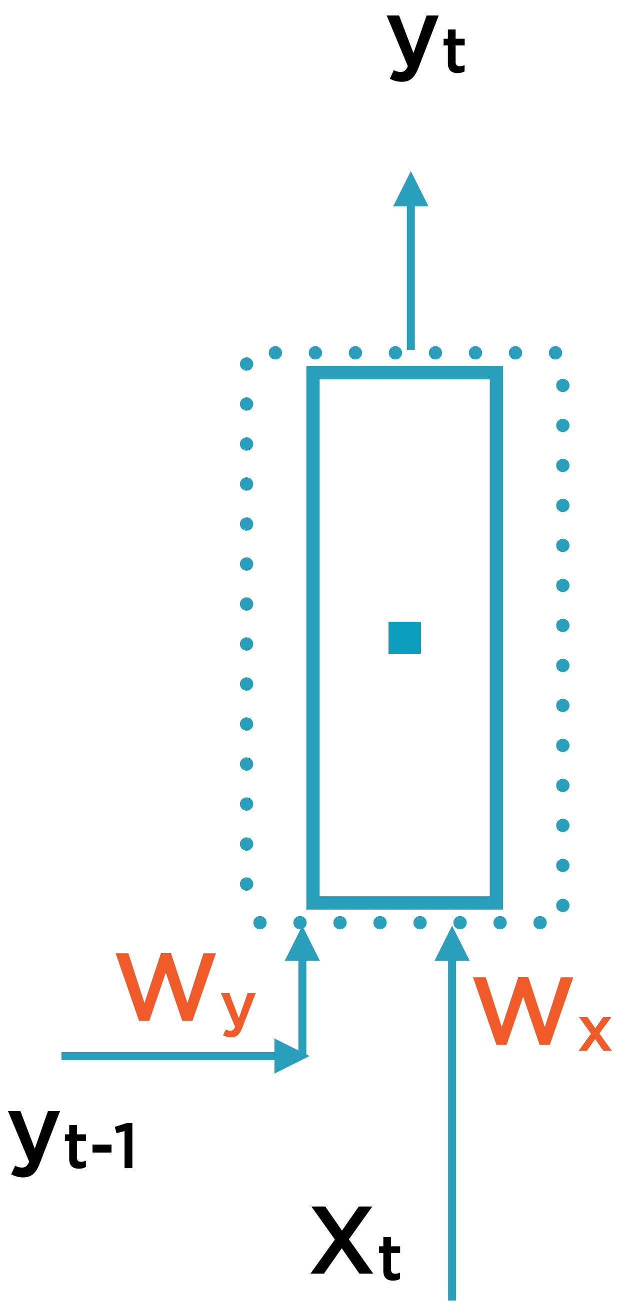


**This is an iterative process**

**Fails either if:**

- gradients don't change at all
- gradients change too fast

# BPTT

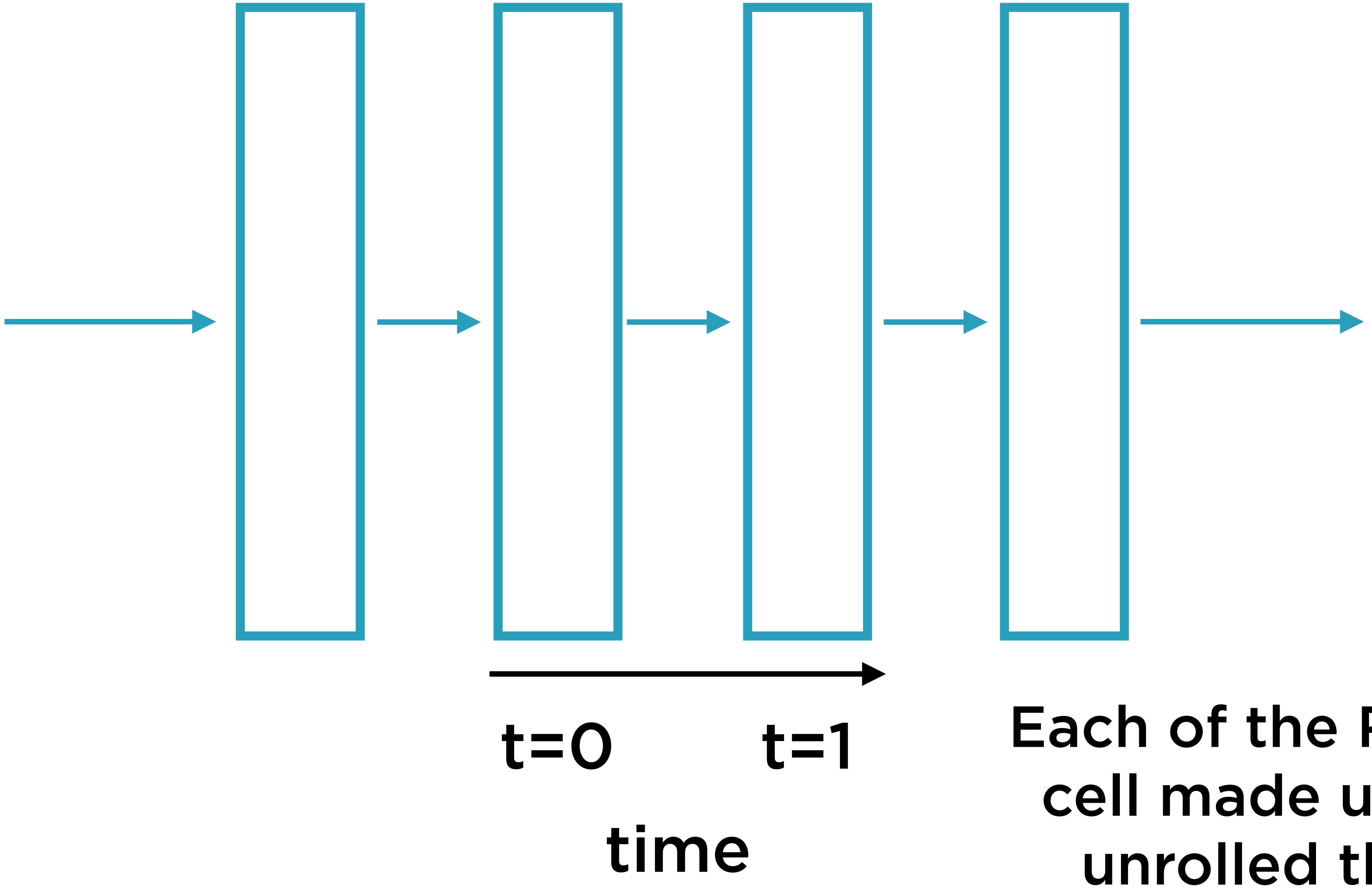



BPTT is the tweaked version of Back Propagation for RNN training

Need as many layers as past time periods

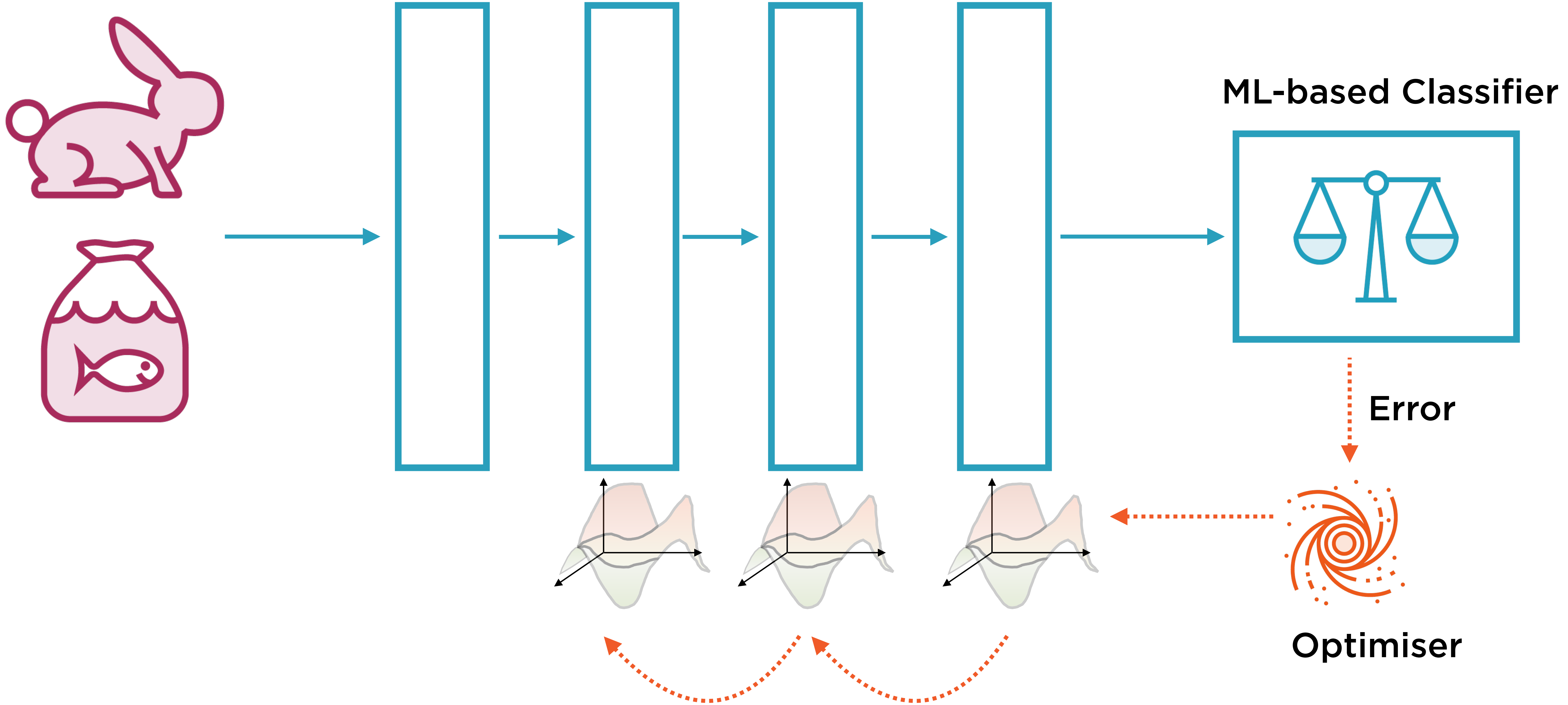
# Vanishing and Exploding Gradients

# Layer of Recurrent Neurons

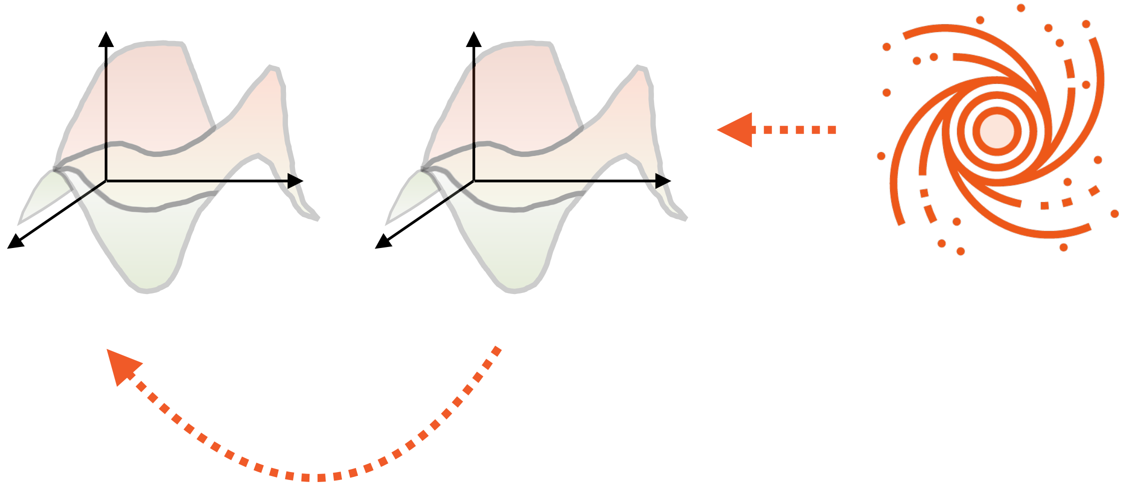



Each of the RNN layers is a cell made up of neurons, unrolled through time

# Back Propagation Through Time (BPTT)



# Back Propagation

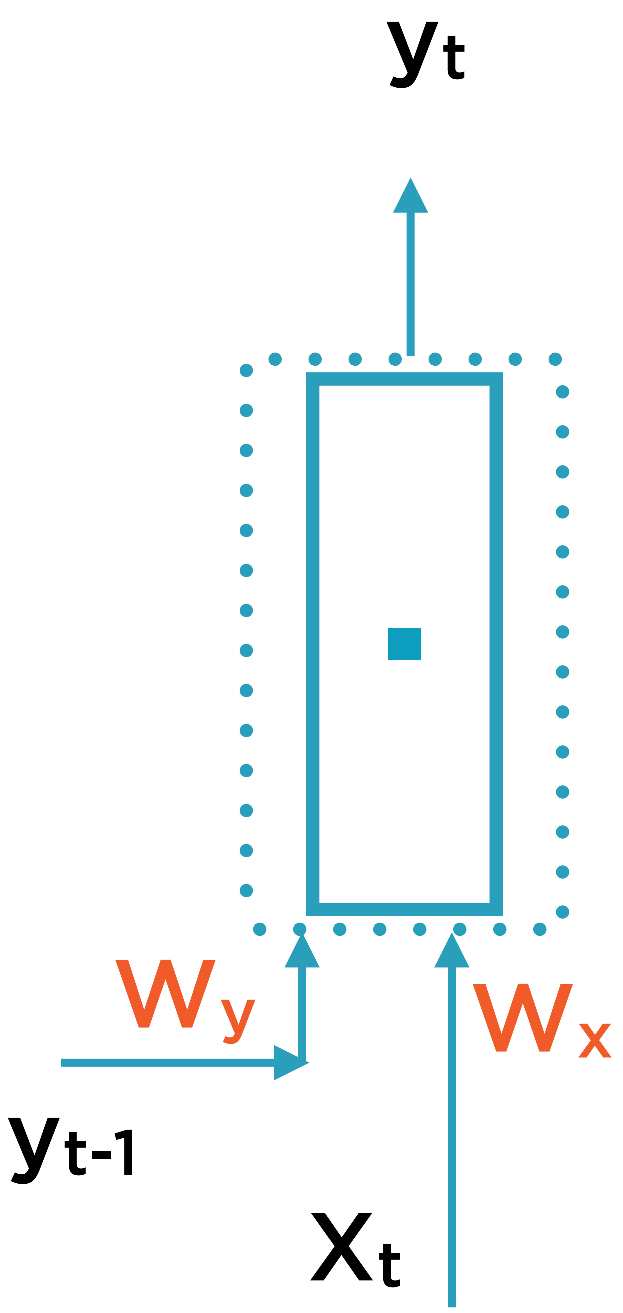


**This is an iterative process**

**Fails either if:**

- gradients don't change at all
- gradients change too fast

# BPTT




BPTT is the tweaked version of Back Propagation for RNN training

Need as many layers as past time periods

$$y_t = f(x_t, y_{t-1}, y_{t-2})$$

# Learning the (Recent) Past

**Unrolling the RNN through time helps learn the past**

$y_t = f(x_t, y_{t-1}, y_{t-2} \ldots, y_{t-1000})$

# Learning the Distant Past

**The unrolled RNN will be very, very deep - many layers to train, the gradient has to be propagated a long way**

$$y_t = f(x_t, y_{t-1}, y_{t-2} \ldots, y_{t-1000})$$

# Learning the Distant Past

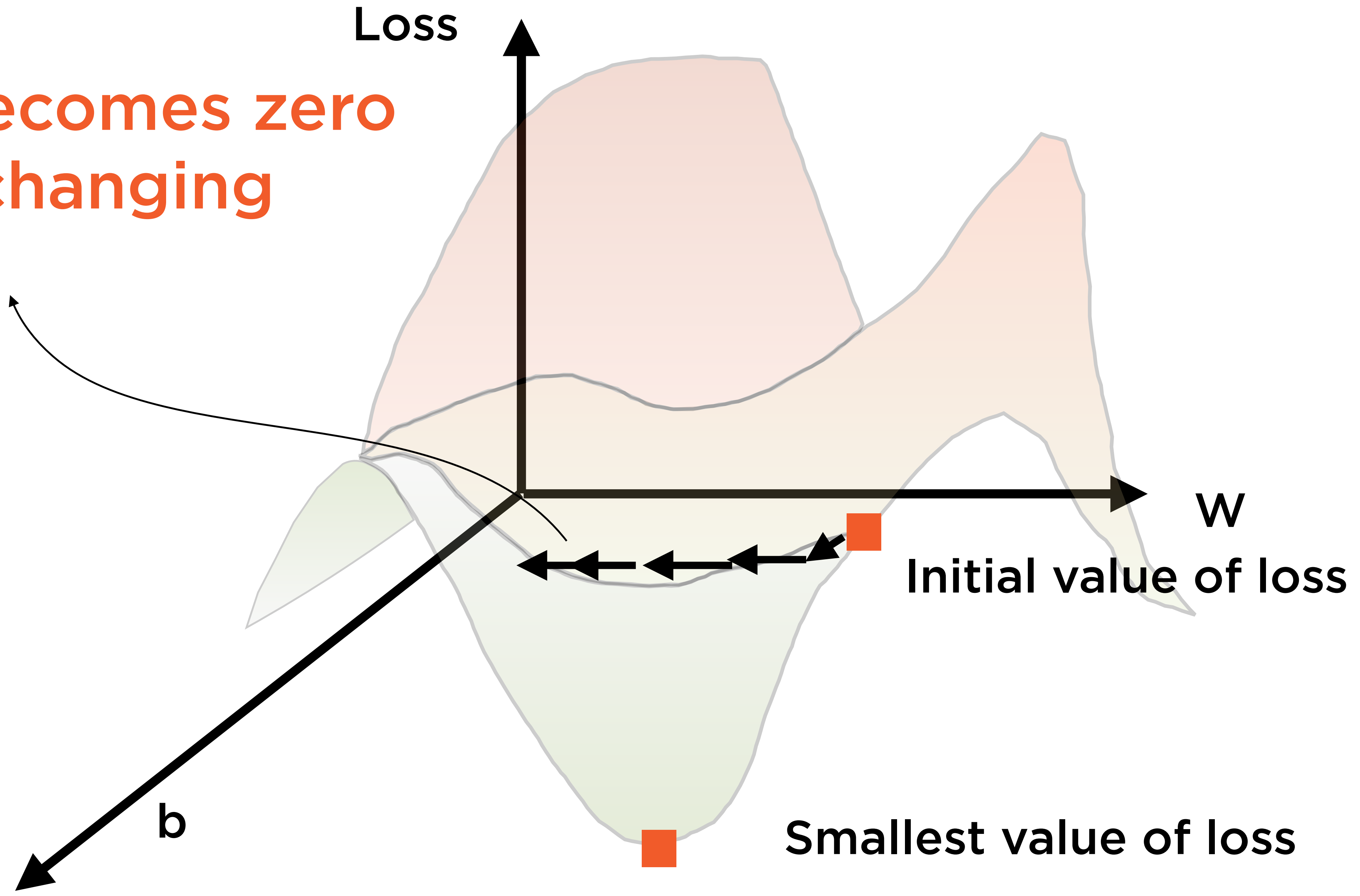
**The unrolled RNN will be very, very deep - many layers to train, the gradient has to be propagated a long way**

Recurrent neural networks may be unrolled **very far back in time**

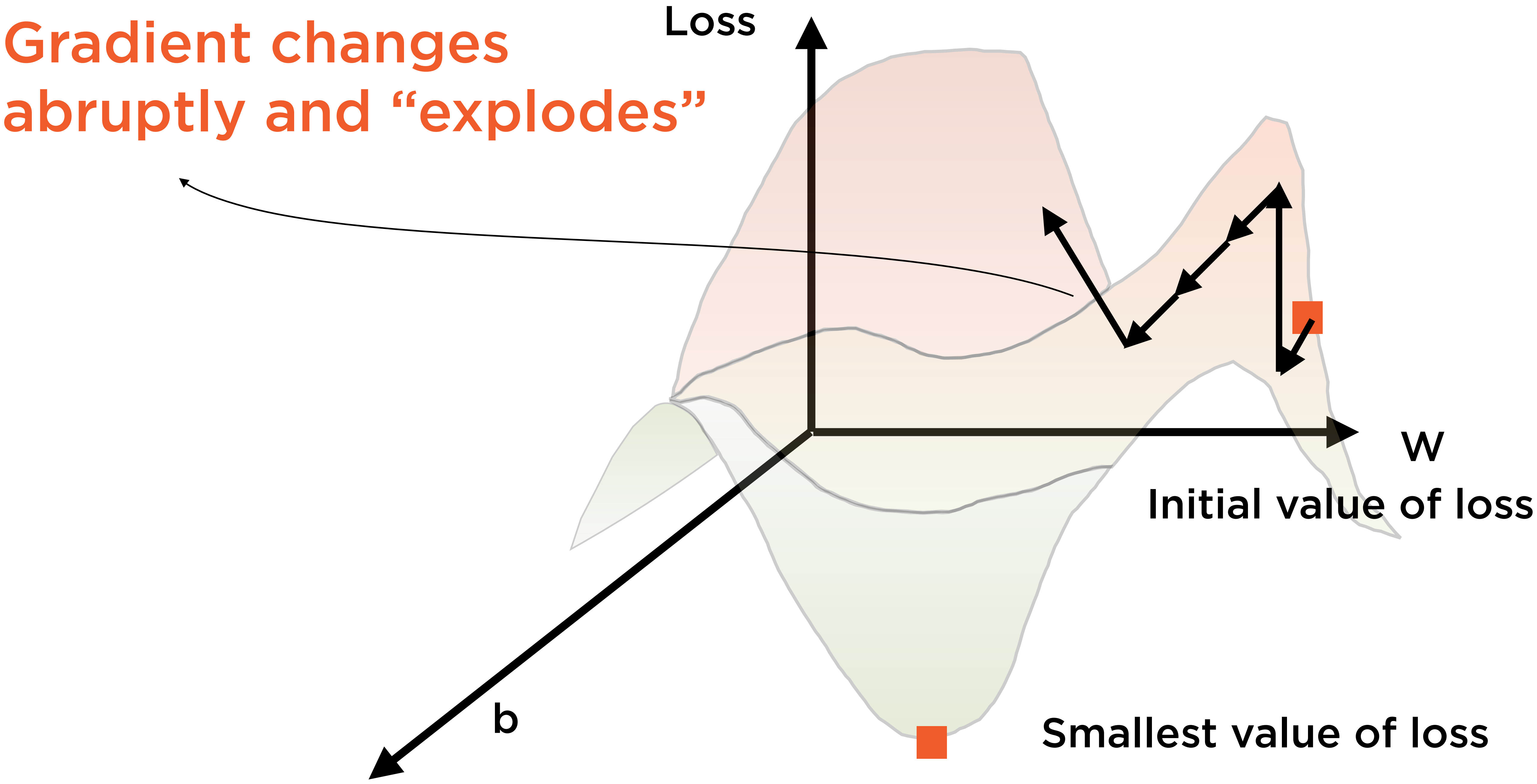
They're prone to the **vanishing** and **exploding** gradients issue

# Vanishing Gradient Problem

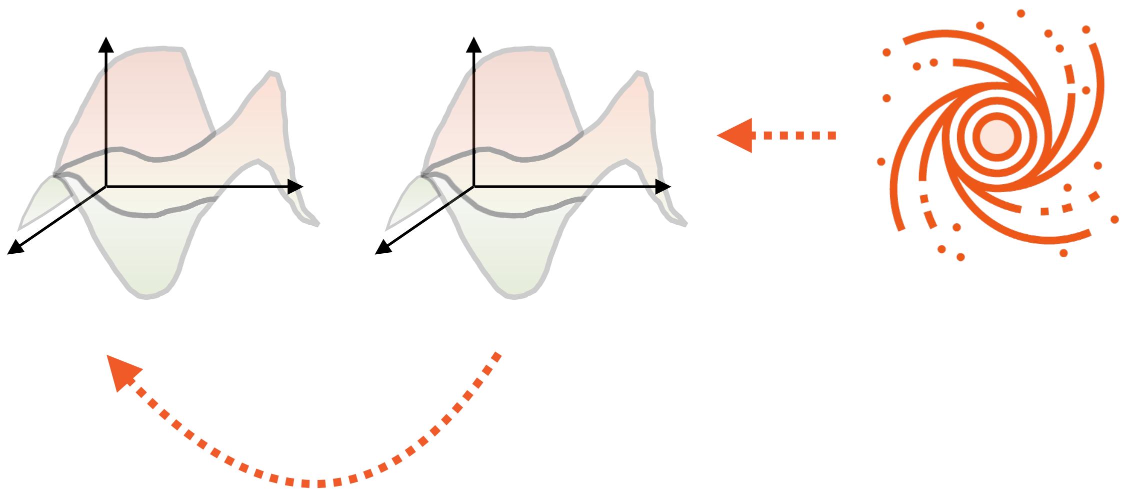
Gradient becomes zero and stops changing



# Exploding Gradient Problem



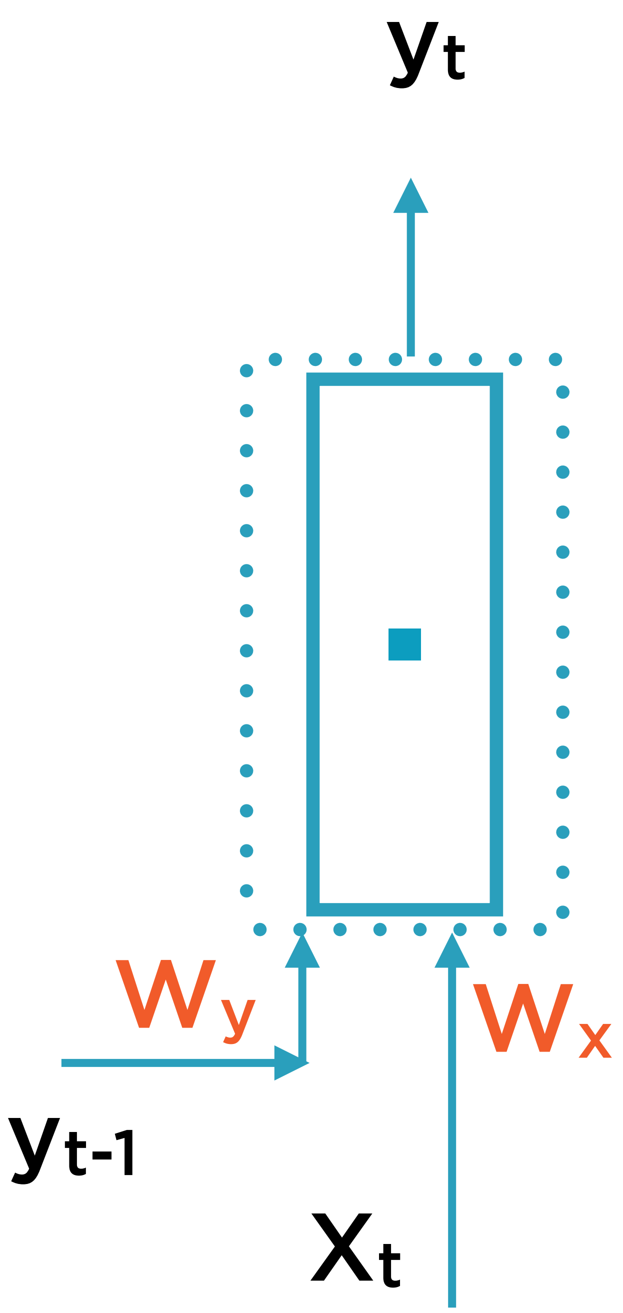
# Vanishing and Exploding Gradients



**Back propagation fails if:**

- gradients are vanishing
- gradients are exploding

# Training RNNs

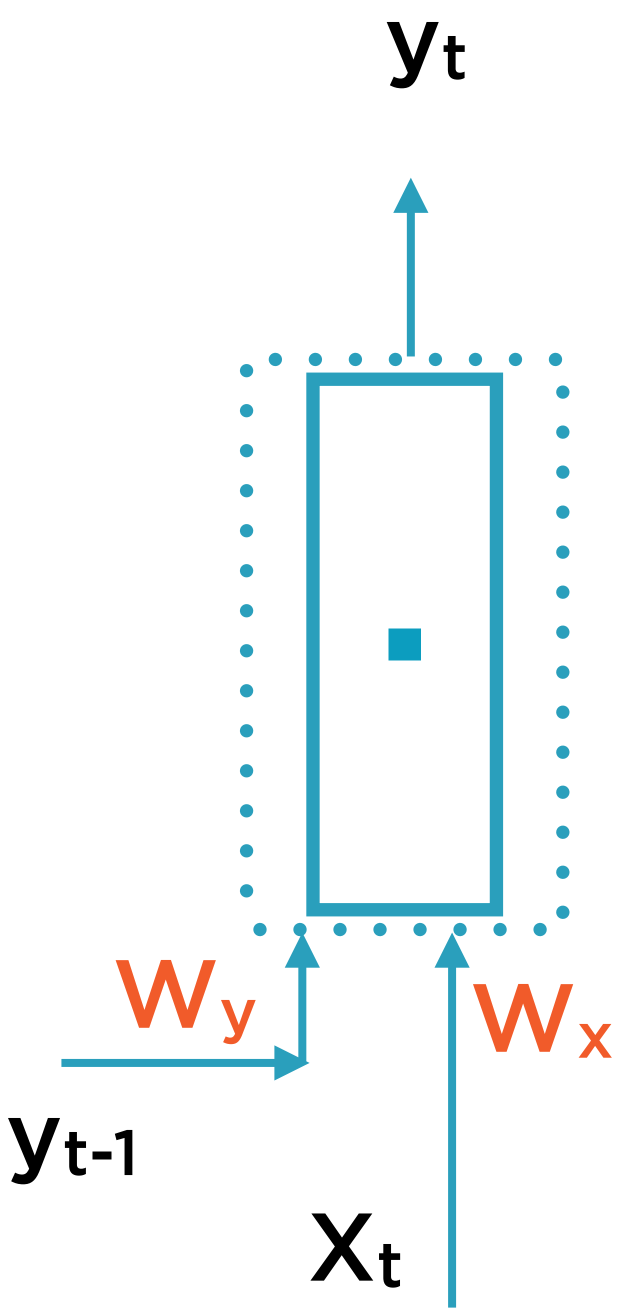



Training RNNs poses some specific challenges

# Coping with Vanishing/Exploding Gradients

| | |
|---|---|
| Proper initialization | Non-saturating activation function |
| Batch normalization | Gradient clipping |

# BPTT

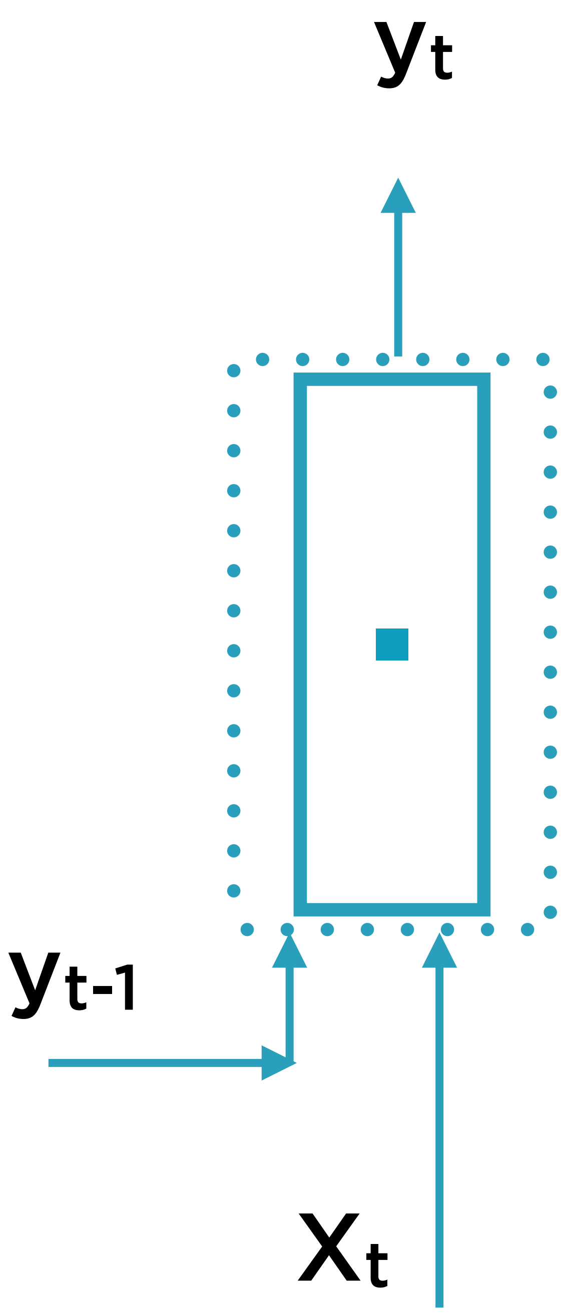



If output relies on distant past

Vanishing/exploding gradients very likely

One option - truncated BPTT

# Truncated BPTT

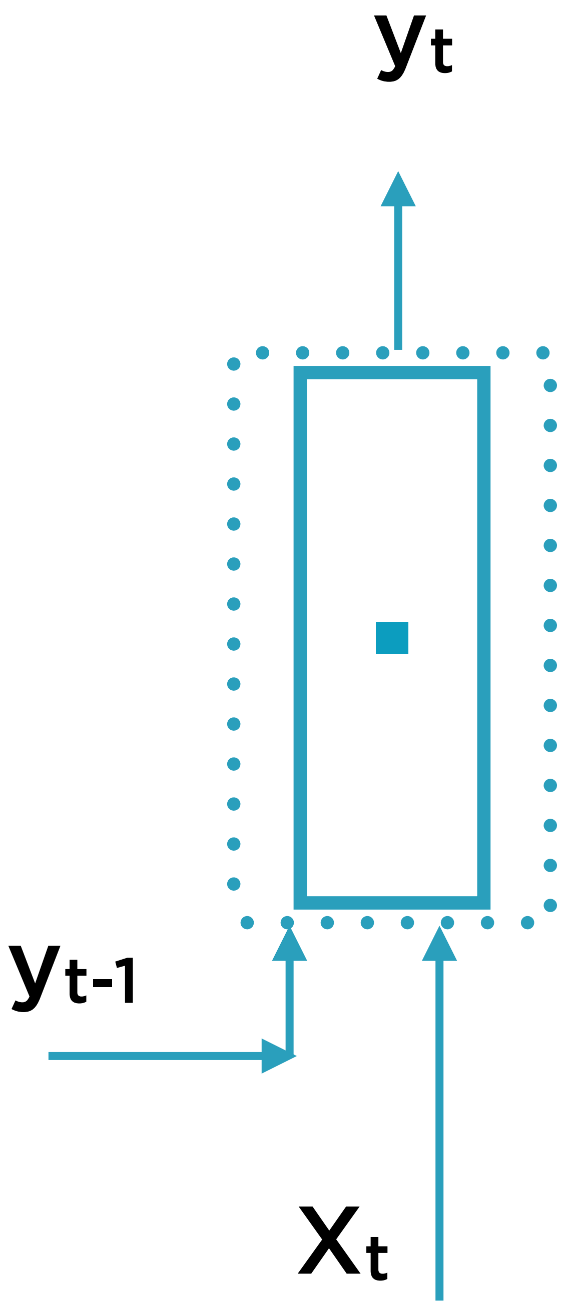



**Simply truncate input sequence**

**E.g. predict stock movement tomorrow:**

- Use only last week’s data
- Do not use data for last year at all
- Daily data for last week, monthly before that

# Truncated BPTT



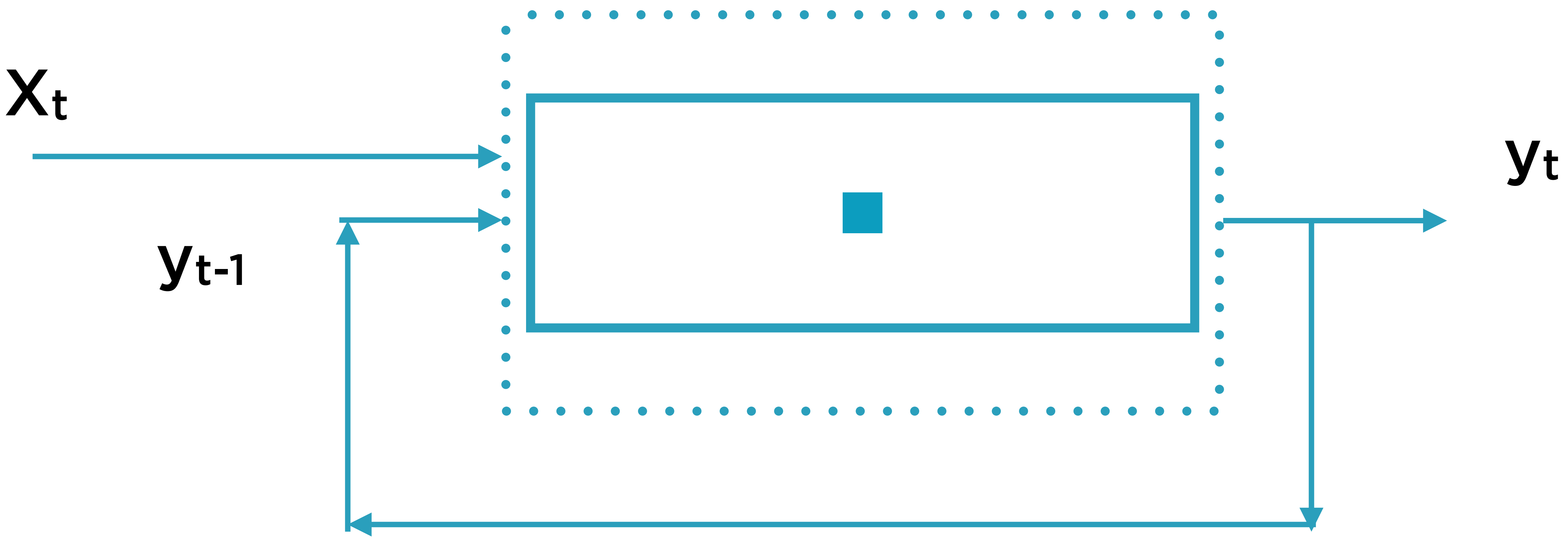

Truncated BPTT can kill prediction performance

What if stock move tomorrow depends on stock move on last quarter-ending date?

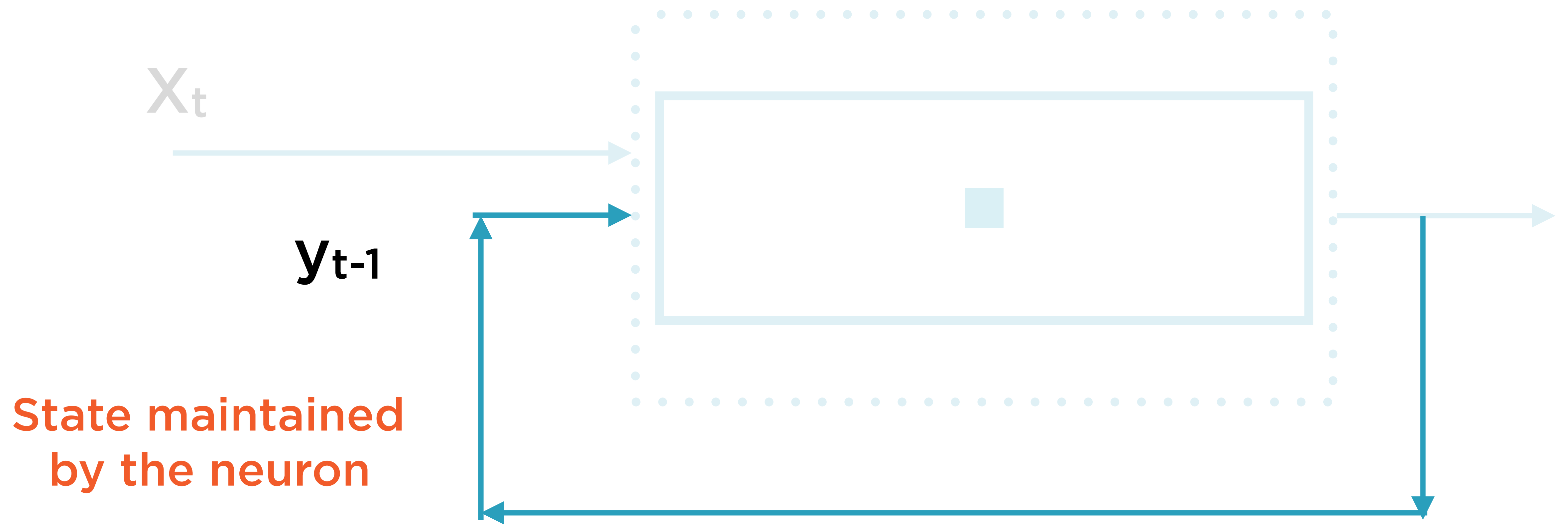
Use **long-memory** cells to store additional state in neuron

# Long-memory Cells

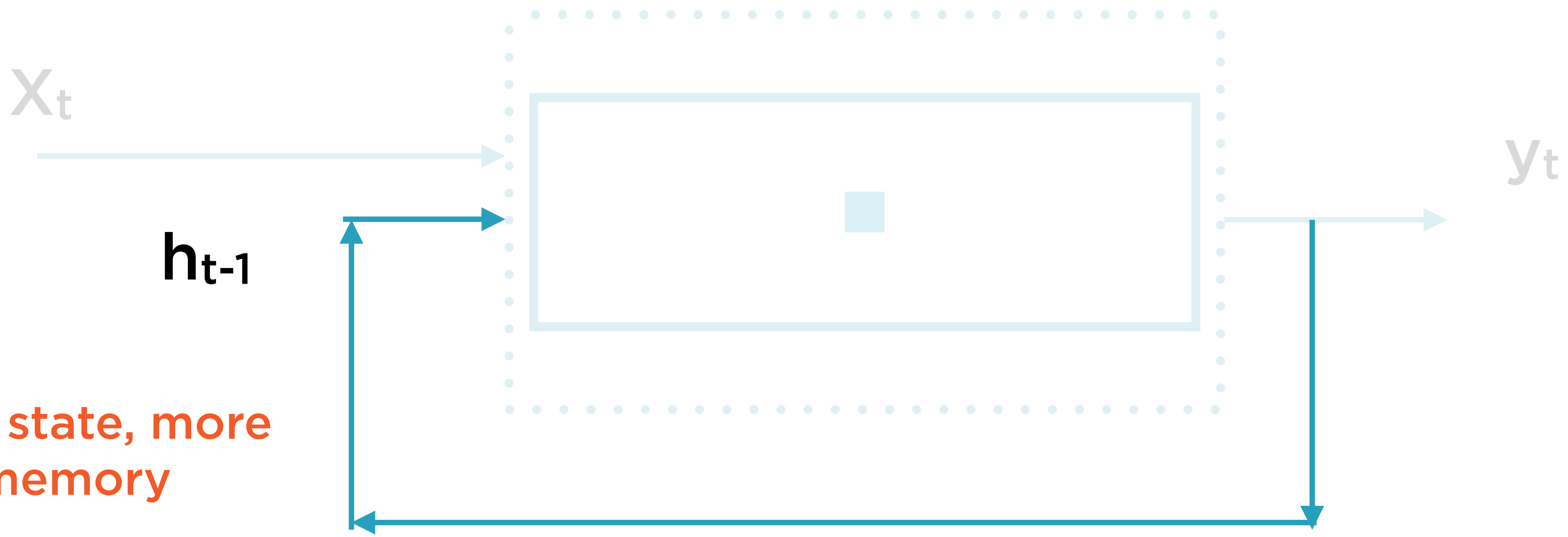
# Simplest Recurrent Neuron



# Simplest Recurrent Neuron

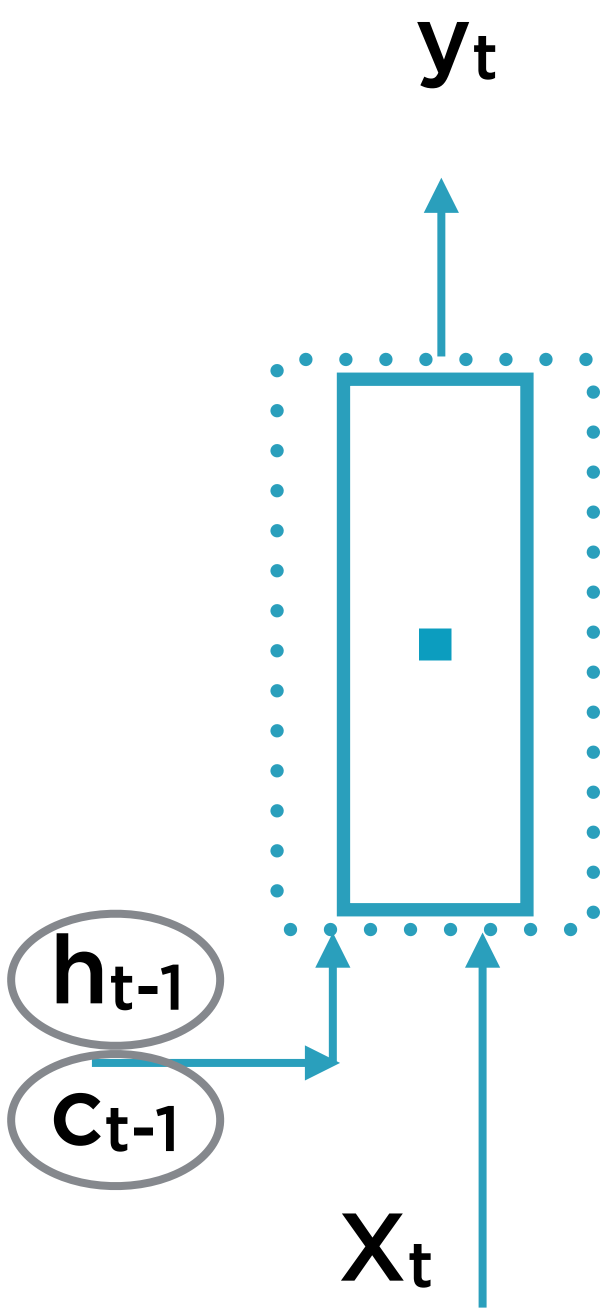


# Long Memory Recurrent Neuron




More state, more memory

# Long Memory RNNs




**Increase the amount of state in neuron**

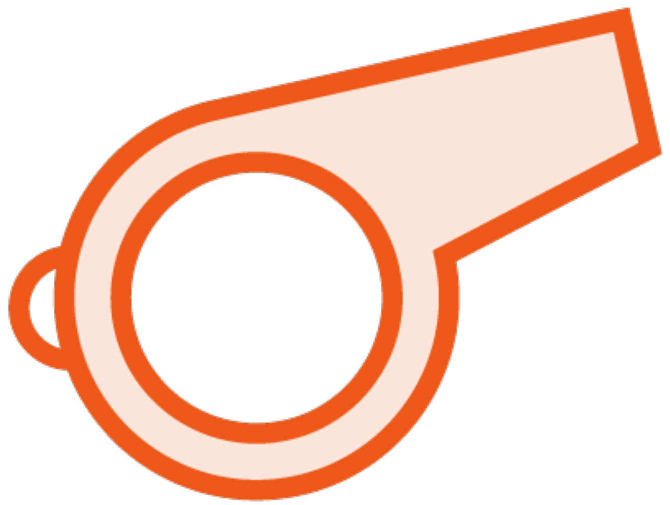
**Effect is to increase memory of neuron**

**Could explicitly add:**

- long-term state (c)
- short-term state (h)

Long memory neurons have several advantages over basic RNNs

# Long Memory RNNs



**Advantages in Training**

Faster training, nicer gradients

**Advantages in Prediction**

No need to truncate BPTT

# Long-term Dependencies in Text

The sky is $\boxed{\text{cloudy,}}$ it looks like $\boxed{\_\_\_\_\_\_}$

The gap between the relevant information needed to predict the next word is small

# Long-term Dependencies in Text

**I've lived in France a long time. I first went there as a tourist, then applied for a job and I've been here ever since. I now speak fluent ________**

The context for which language to predict here is much farther back in the sentence

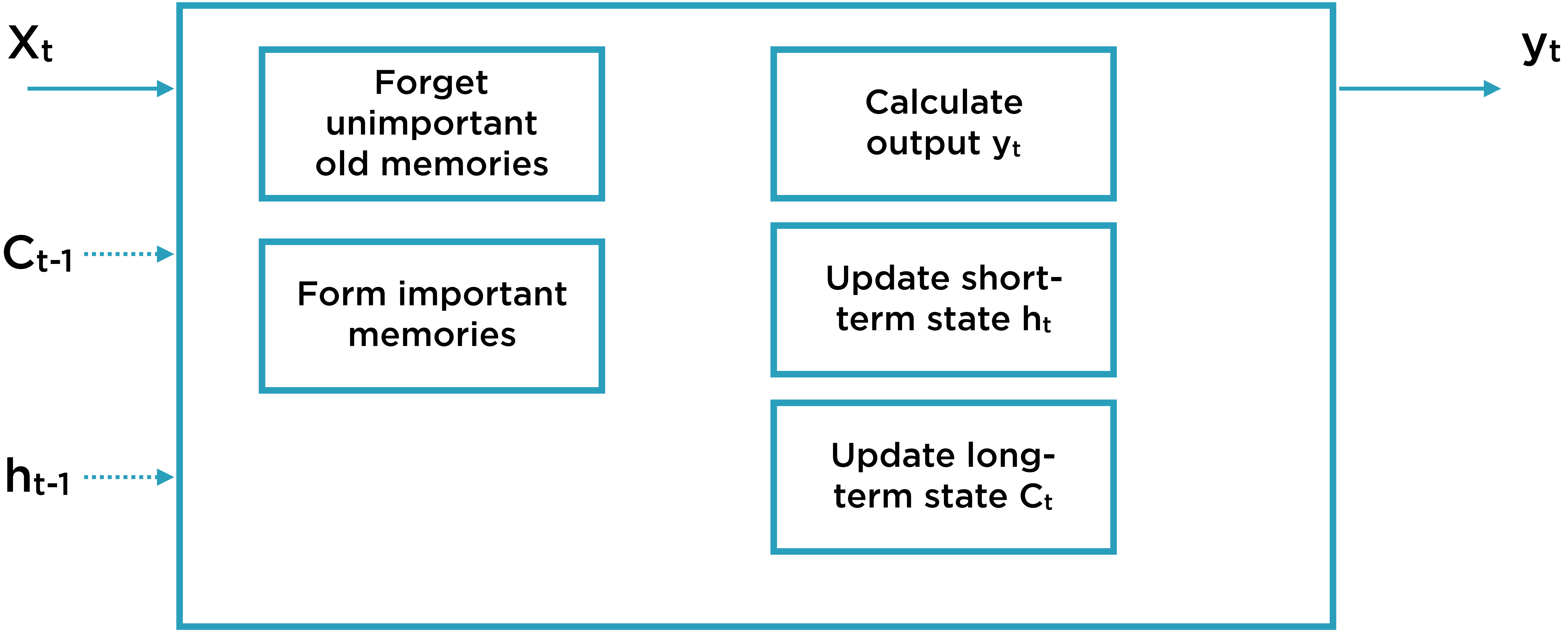
# Long-term Dependencies in Text

**I’ve lived in France a long time. I first went there as a tourist, then applied for a job and I’ve been here ever since. I now speak fluent ________**

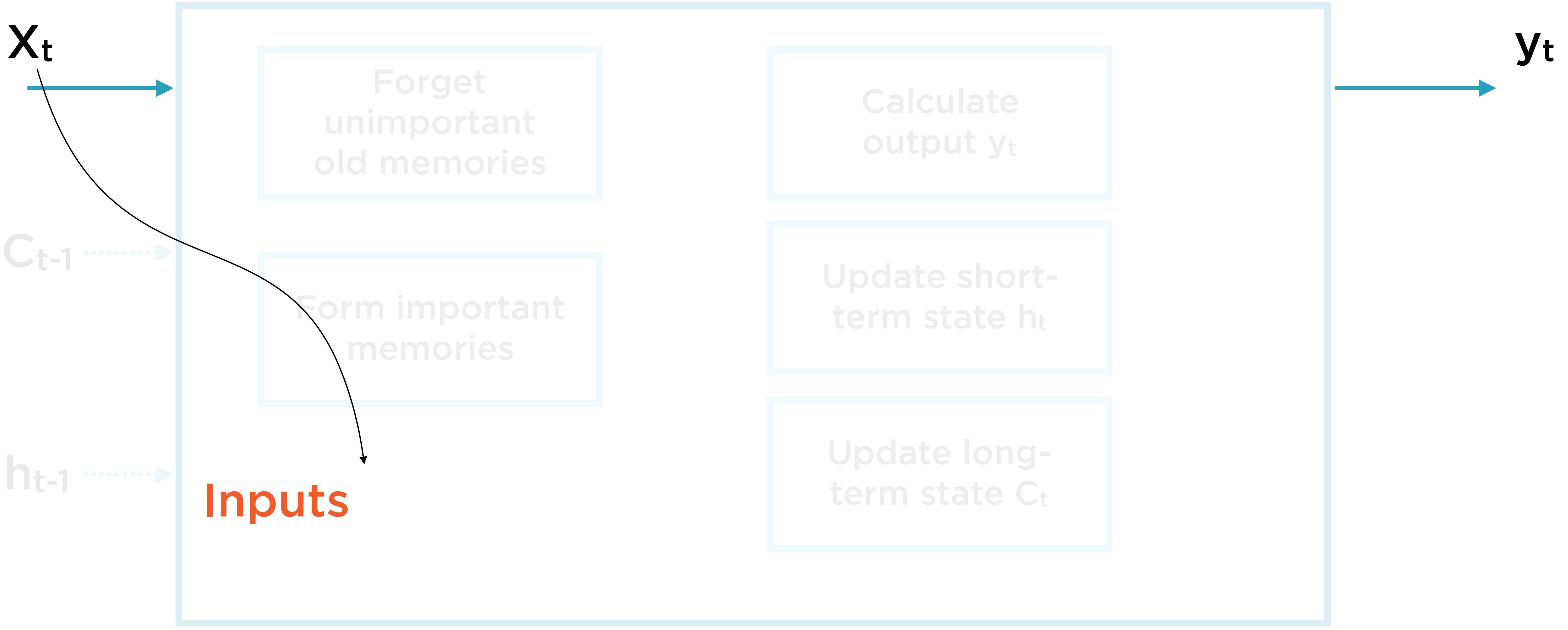
As this gap grows, RNNs become unable to learn to connect the information

Long/Short-Term Memory Cell (**LSTM**) - a popularly used long memory cell in RNNs

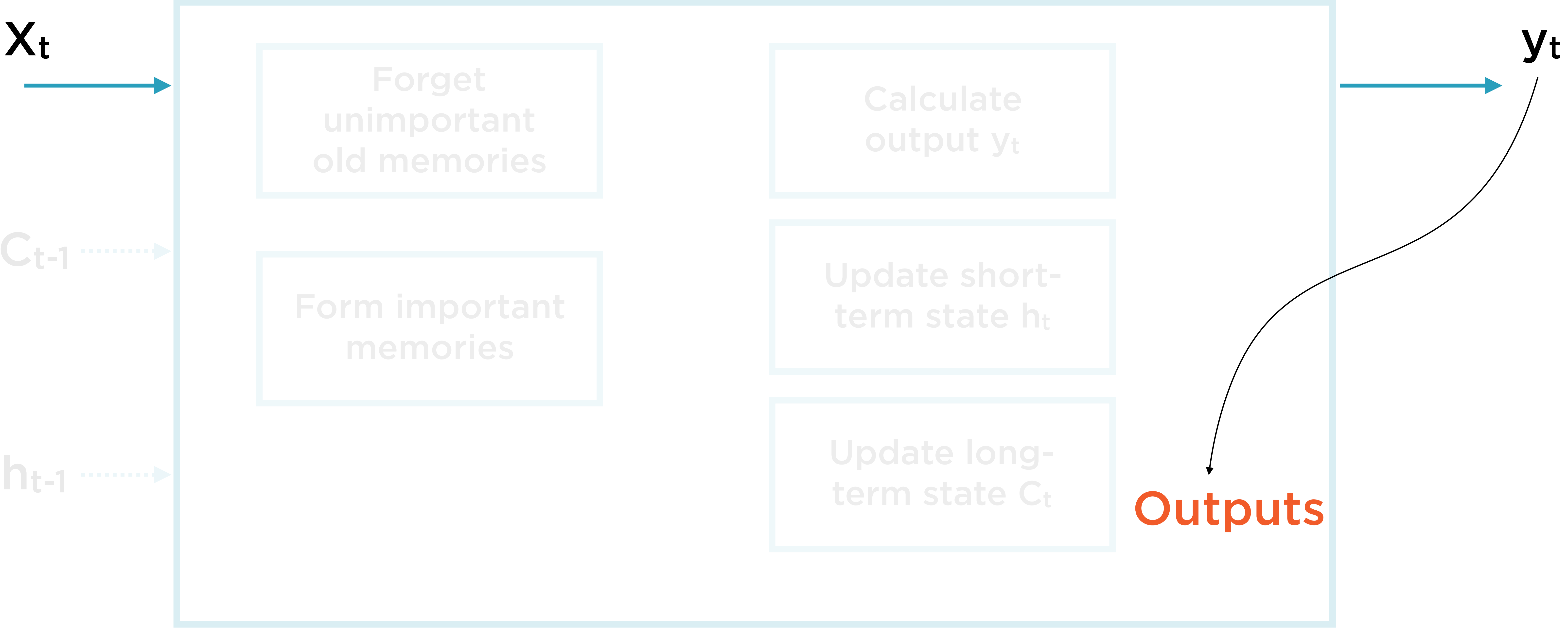
# LSTM


$X_t$
$C_{t-1}$
$h_{t-1}$
Forget unimportant old memories
Form important memories
Calculate output $y_t$
Update short-term state $h_t$
Update long-term state $C_t$
$y_t$

# LSTM

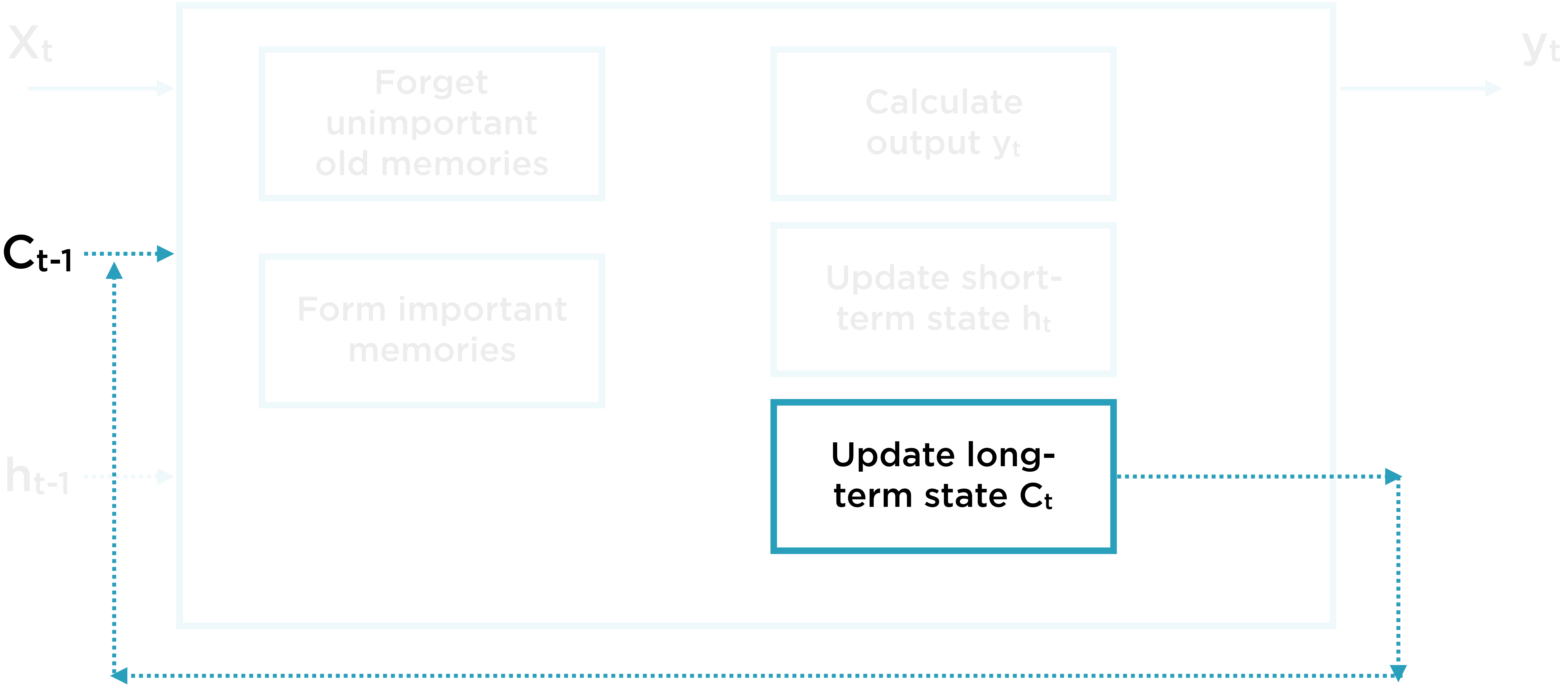

Xt
yt
Forget unimportant old memories
Calculate output yt
Ct-1
Form important memories
Update short-term state ht
ht-1
Inputs
Update long-term state Ct

# LSTM



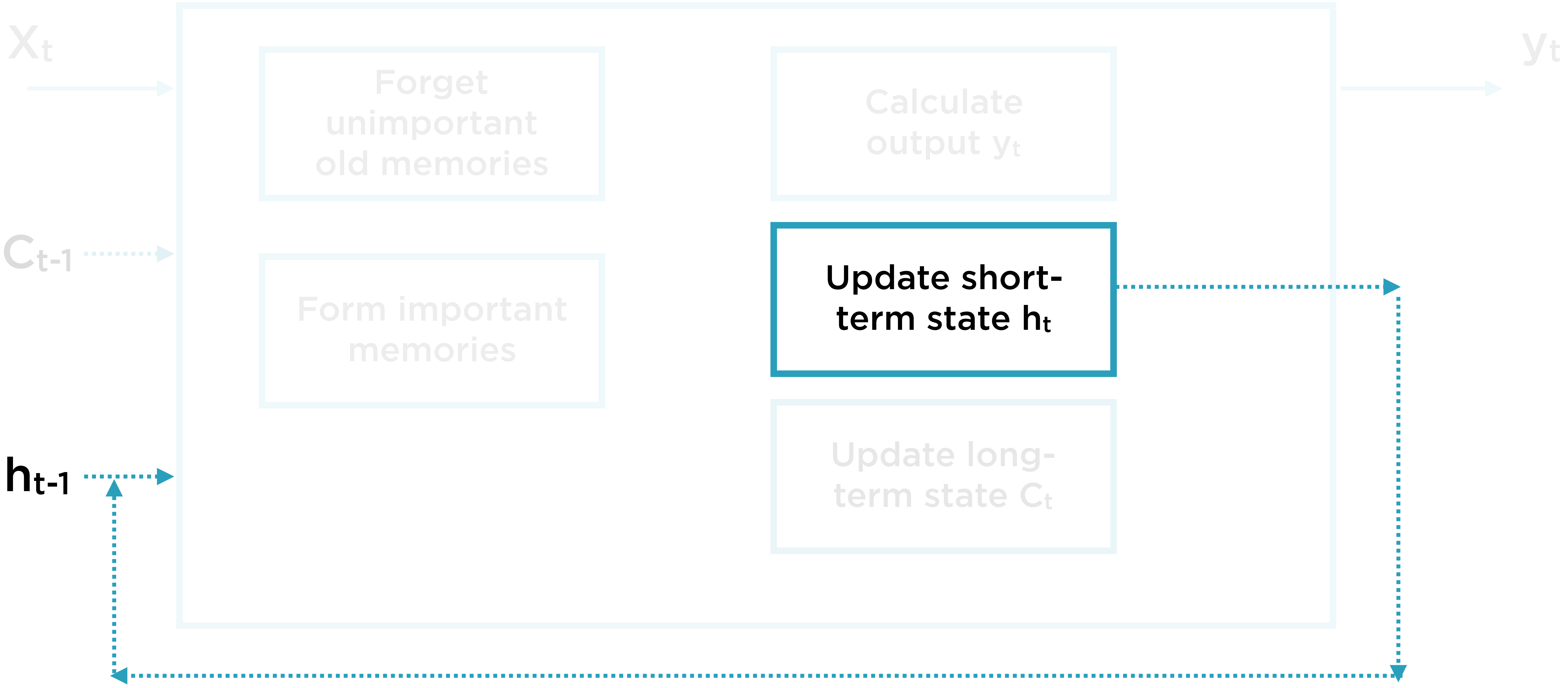
$x_t$
Forget unimportant old memories
Calculate output $y_t$
$y_t$
$C_{t-1}$
Update short-term state $h_t$
Form important memories
$h_{t-1}$
Update long-term state $C_t$
Outputs

# LSTM



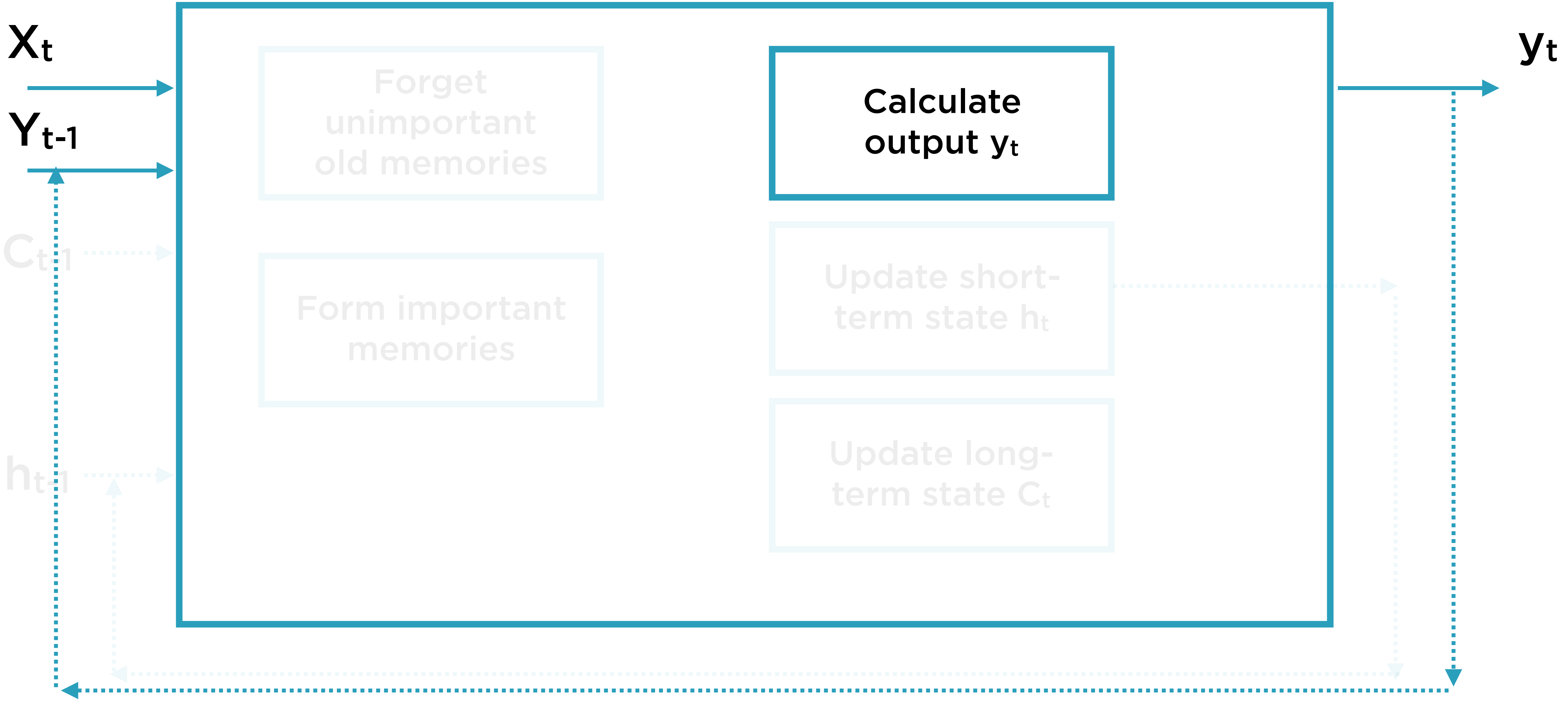
$X_t$
Forget unimportant old memories
Form important memories
Calculate output $y_t$
Update short-term state $h_t$
Update long-term state $C_t$
$y_t$
$C_{t-1}$
$h_{t-1}$

# LSTM

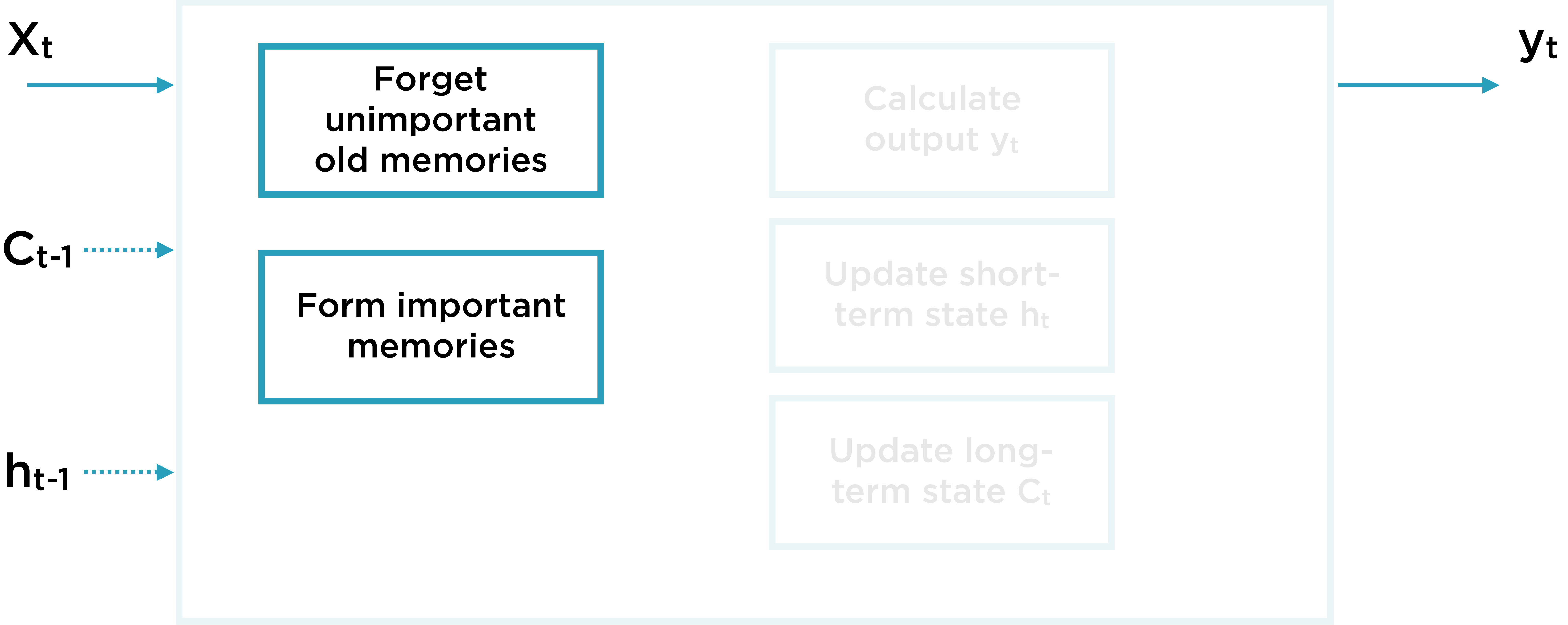


$X_t$
$C_{t-1}$
$h_{t-1}$
Forget unimportant old memories
Form important memories
Calculate output $y_t$
Update short-term state $h_t$
Update long-term state $C_t$
$y_t$

# Basic RNN Cell

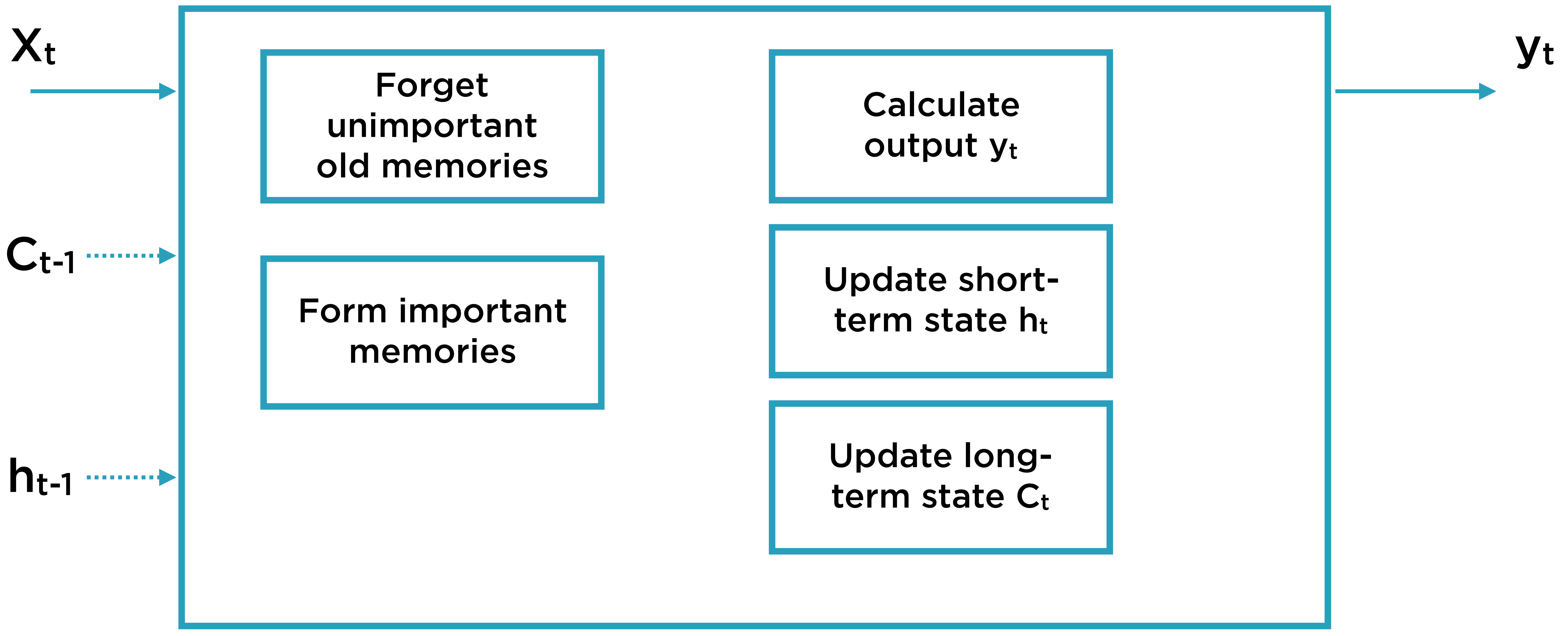


# LSTM

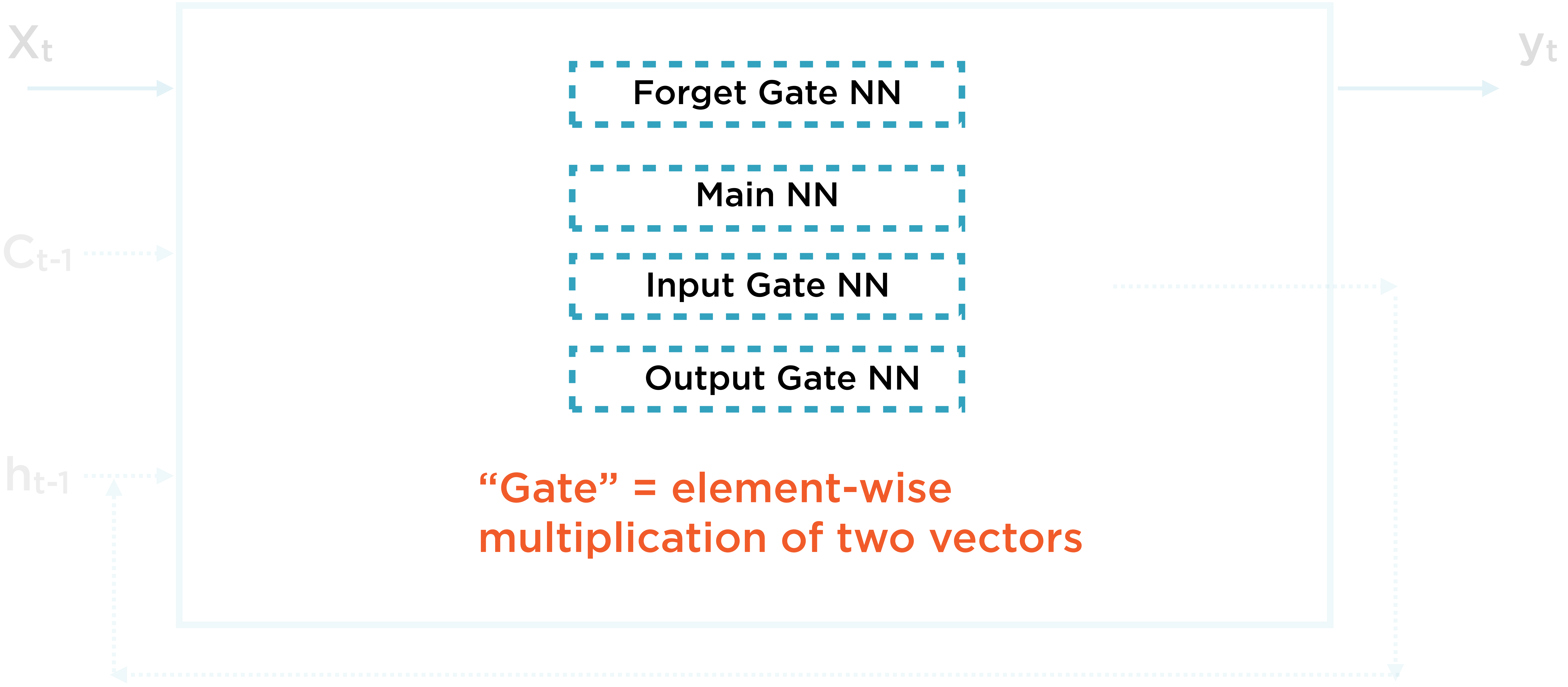


$X_t$
$C_{t-1}$
$h_{t-1}$
Forget unimportant old memories
Form important memories
Calculate output $y_t$
Update short-term state $h_t$
Update long-term state $C_t$
$y_t$

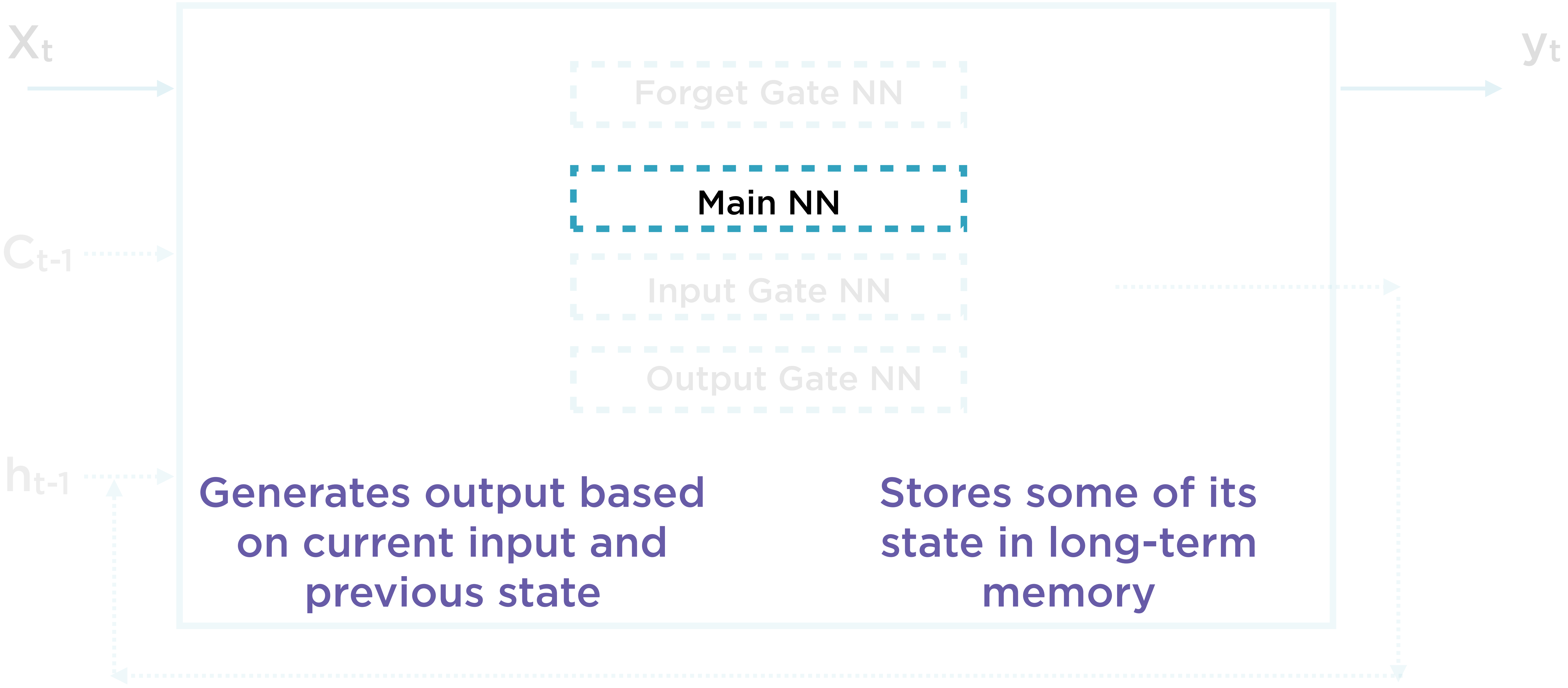
# LSTM


$X_t$
$C_{t-1}$
$h_{t-1}$
Forget unimportant old memories
Form important memories
Calculate output $y_t$
Update short-term state $h_t$
Update long-term state $C_t$
$y_t$

# LSTM

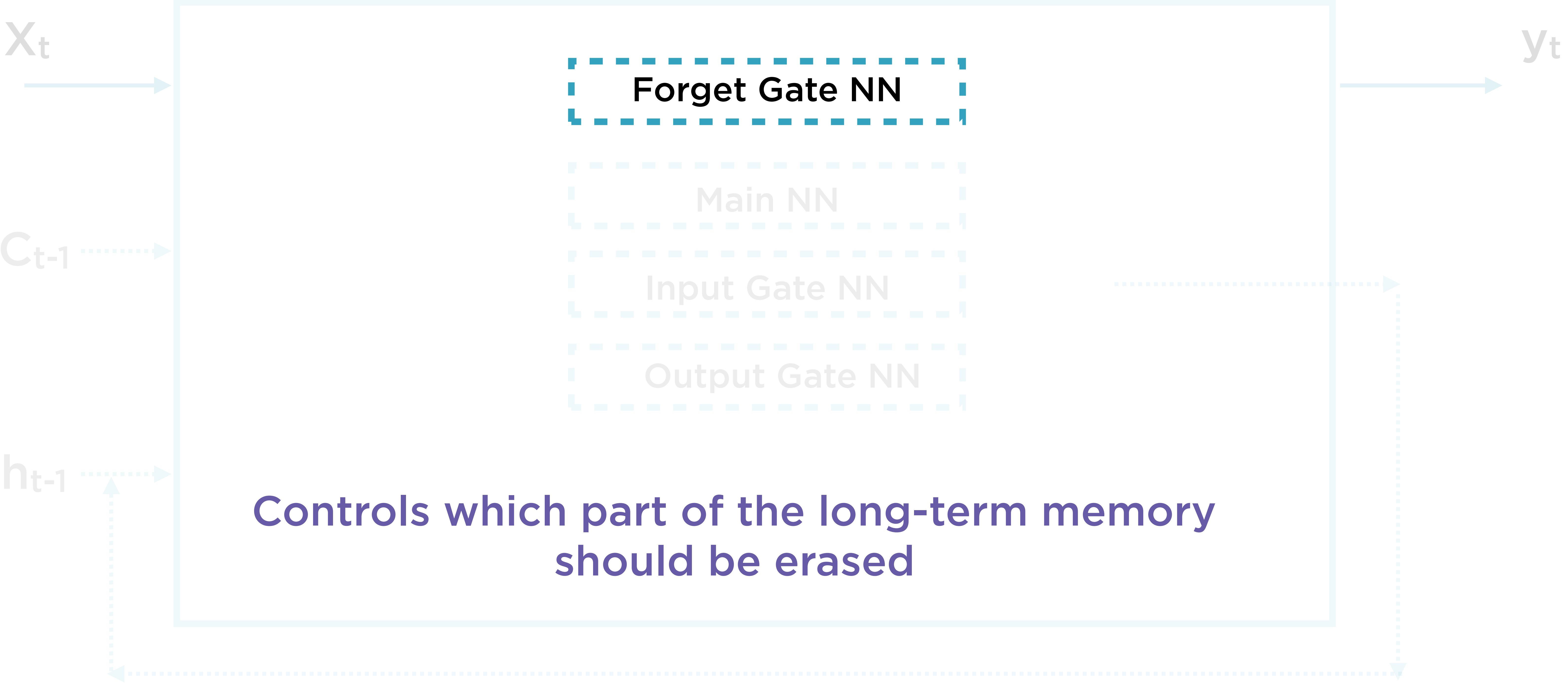

$x_t$
$y_t$
Forget Gate NN
Main NN
$C_{t-1}$
Input Gate NN
Output Gate NN
$h_{t-1}$
“Gate” = element-wise multiplication of two vectors

# LSTM



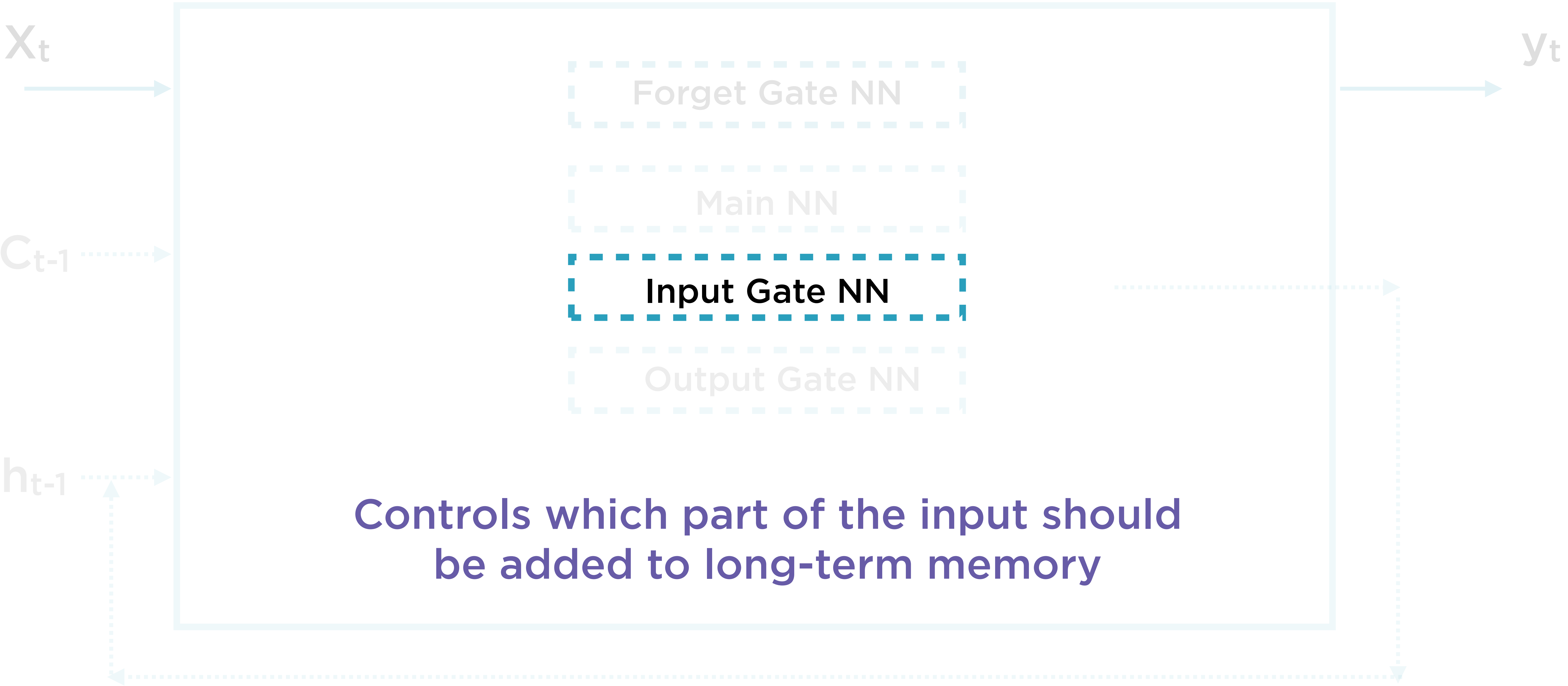
$x_t$
$y_t$
$C_{t-1}$
$h_{t-1}$
Forget Gate NN
Main NN
Input Gate NN
Output Gate NN
Generates output based on current input and previous state
Stores some of its state in long-term memory

# LSTM



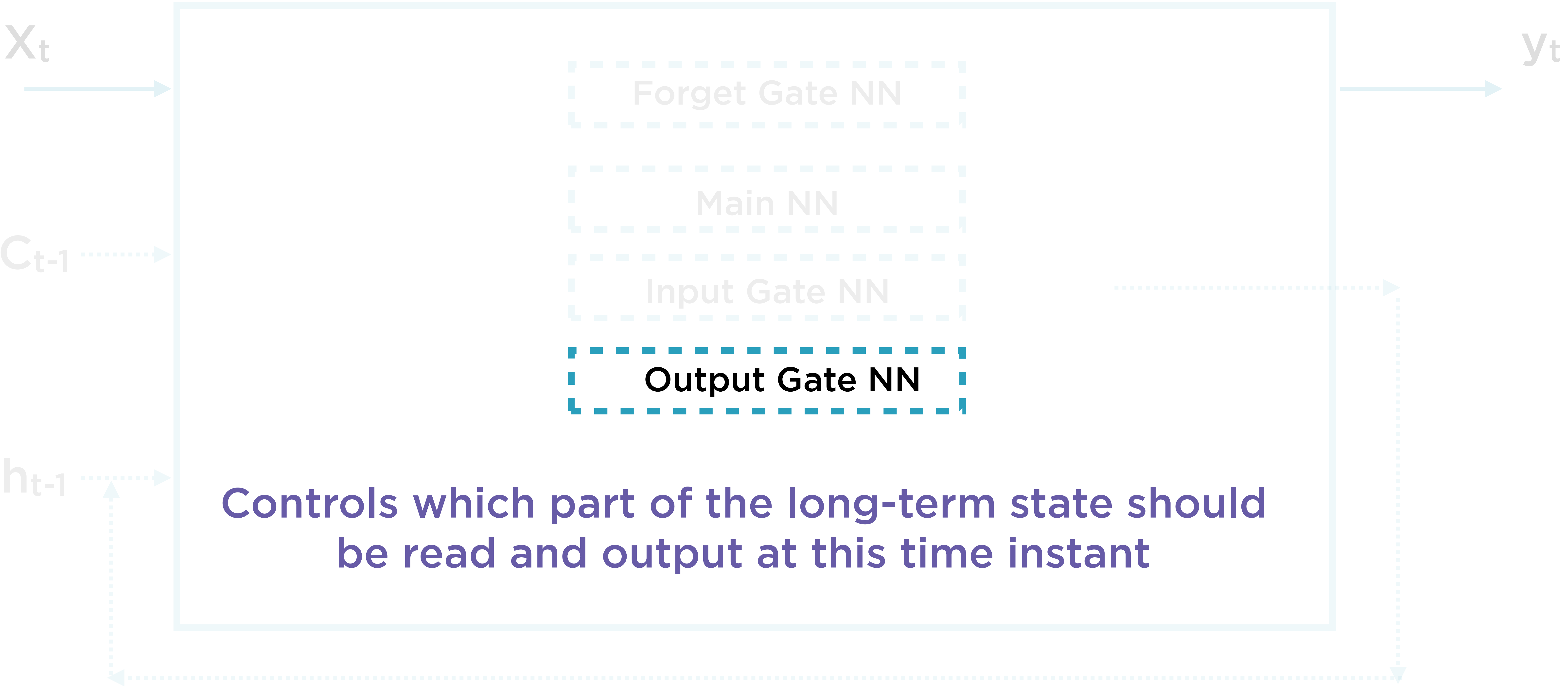

Controls which part of the long-term memory should be erased

# LSTM



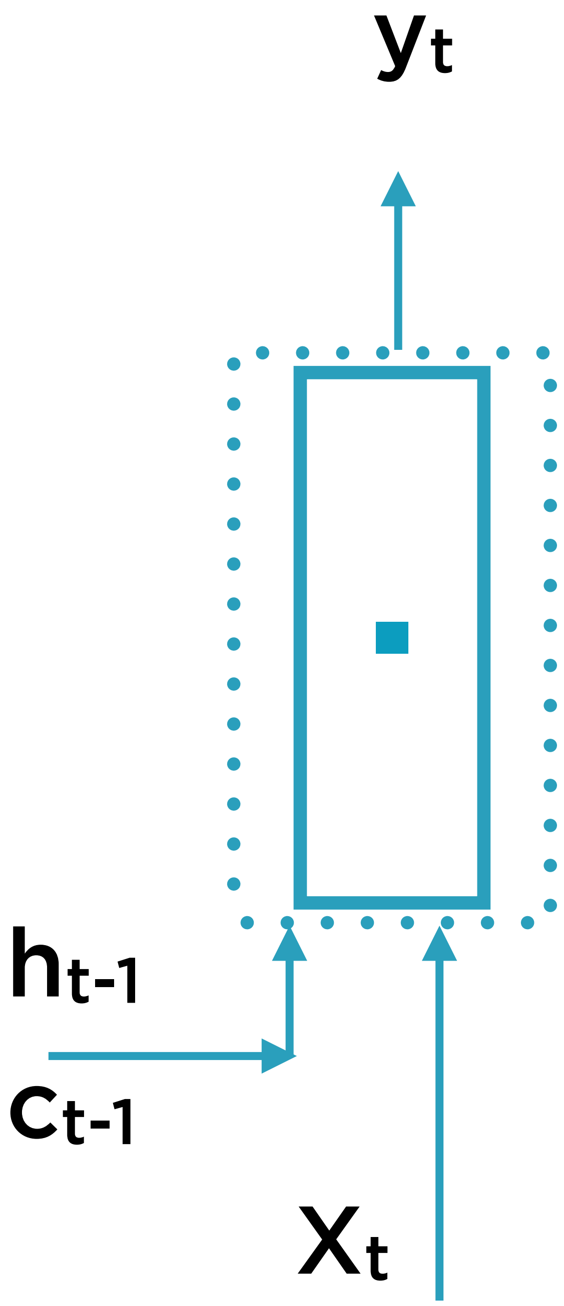

Controls which part of the input should be added to long-term memory

# LSTM




Controls which part of the long-term state should be read and output at this time instant

# LSTM Cells



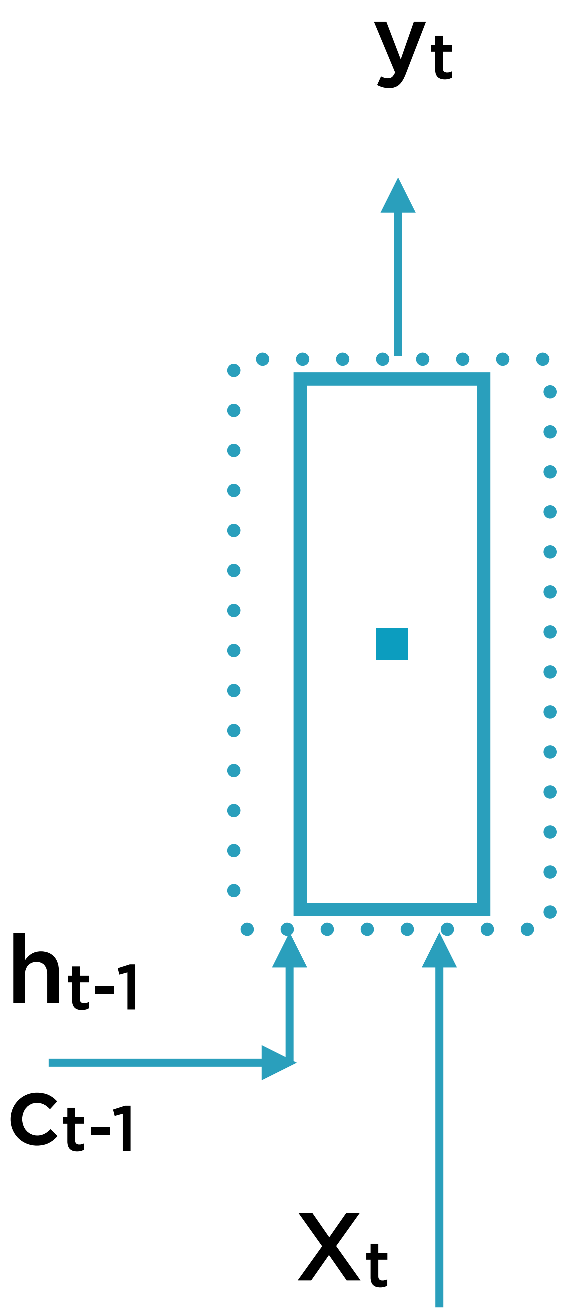

Functionally like basic RNN cell

Performance far better

Amazing success at long-term patterns

Long text sequences, time series

# Variants




**Peephole connections:** LSTM cells that store state for more than 1 period

**Gated Recurrent Unit (GRU):** Simplified LSTM with better performance

- Only 1 state vector
- Fewer internal gates and NNs

# Summary

Modifying neurons to endow them with state and memory

Understand Recurrent Neural Networks

Mitigate problems of vanishing and exploding gradients in training RNNs

LSTM and GRU neurons in RNNs

Use RNNs in language modeling