# Building a Text Classification Model


**Janani Ravi**

CO-FOUNDER, LOONYCORN

www.loonycorn.com

# Overview

Recurrent Neural Networks (RNNs) are another common NN architecture

Great for working with sequences

Classification using RNNs

PyTorch has built-in support for RNNs

# Recurrent Neural Networks (RNNs)

```
y = f(x)
```

# Machine Learning

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

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

# Learning the Past

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

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

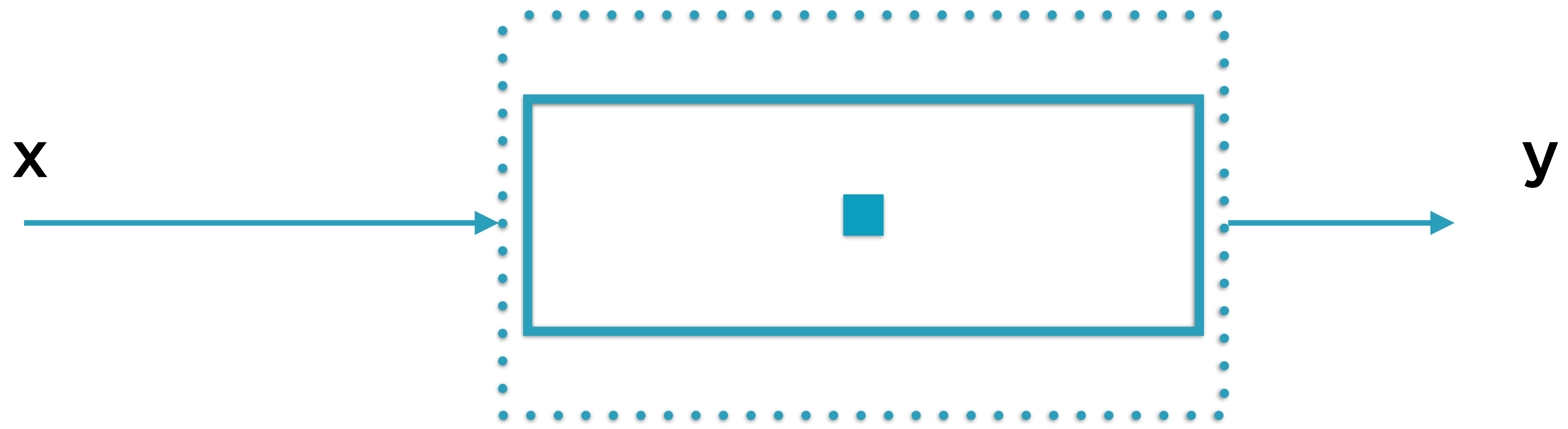
# Learning the Past

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

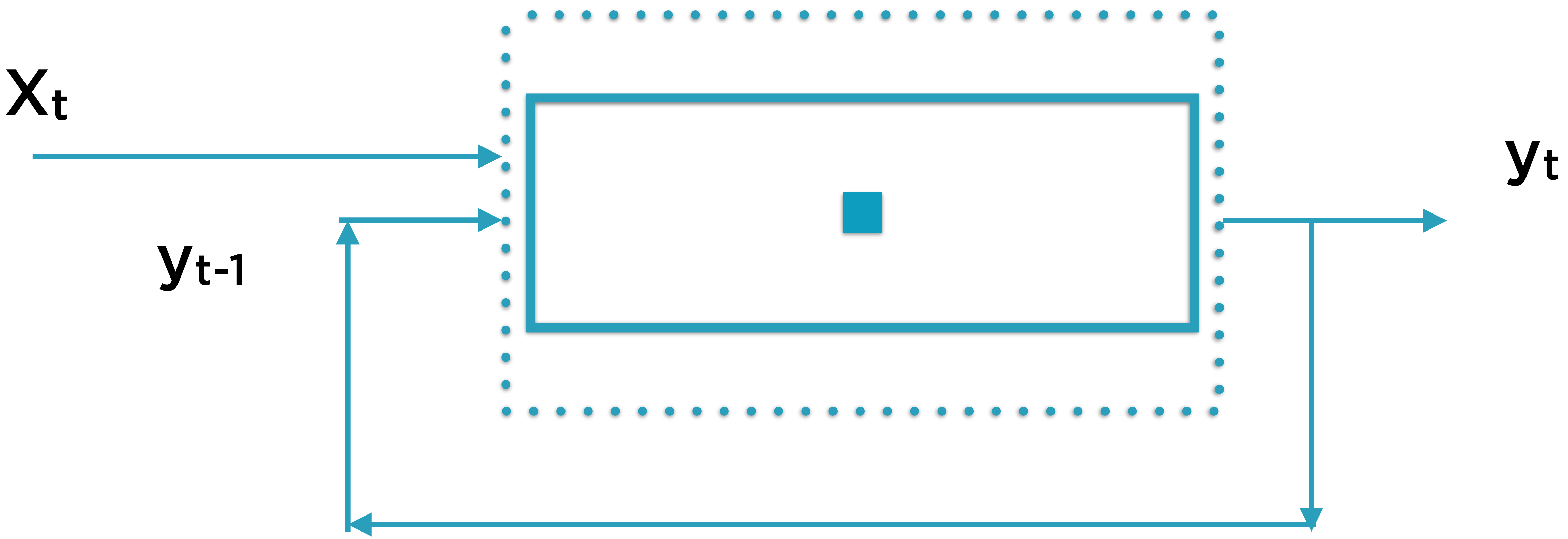
Feed-forward networks cannot learn from the **past**

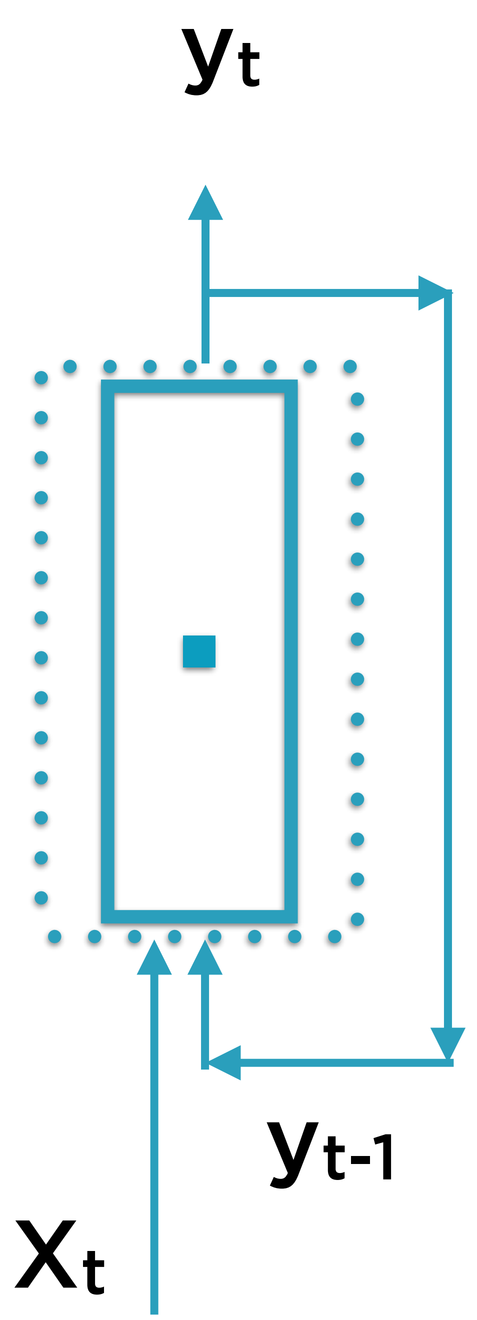
Recurrent neural networks can

# Simplest Feed-forward Neuron



# Simplest Recurrent Neuron





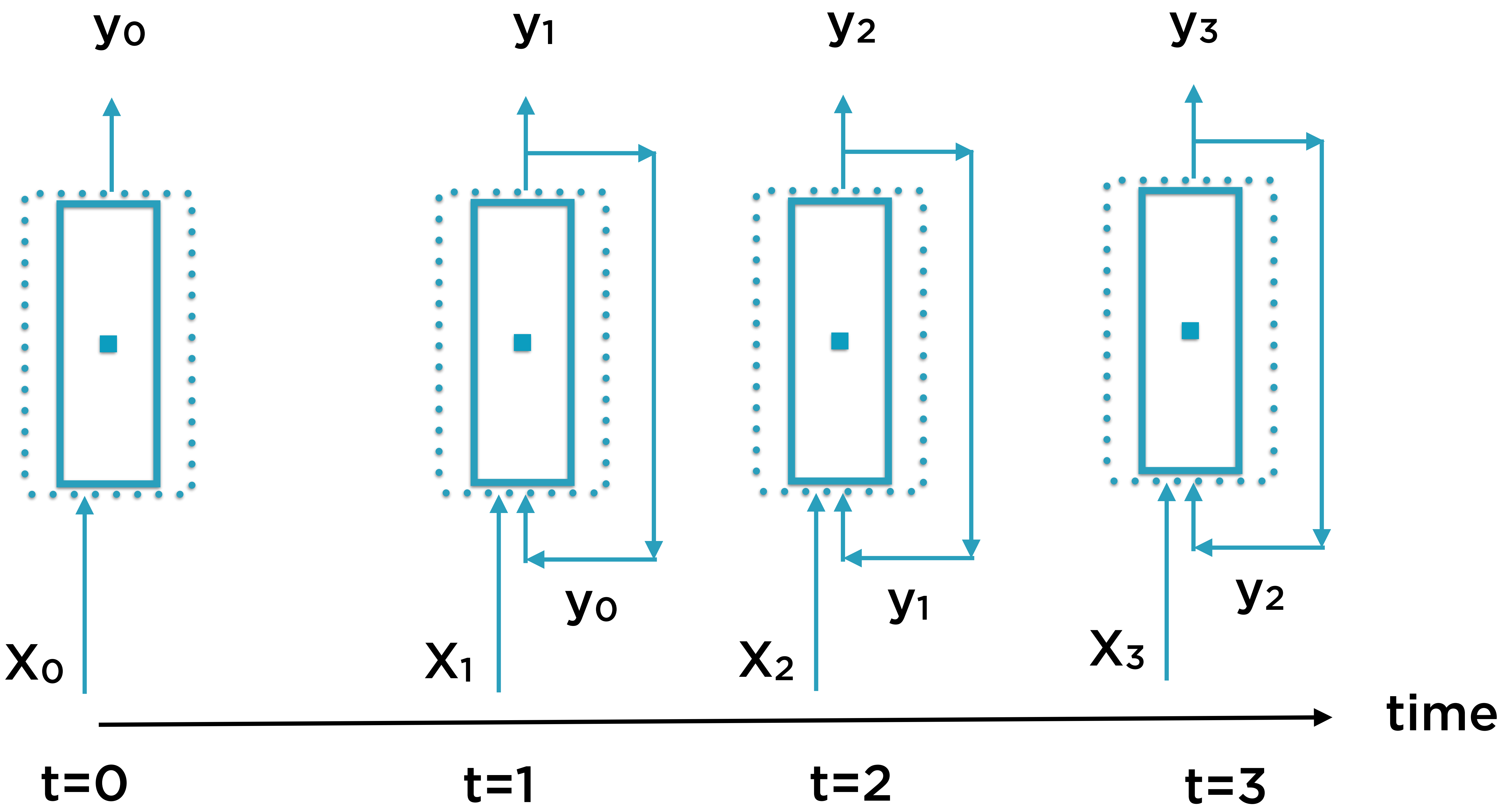

# Recurrent Neuron

$y_t$ = Output at time t

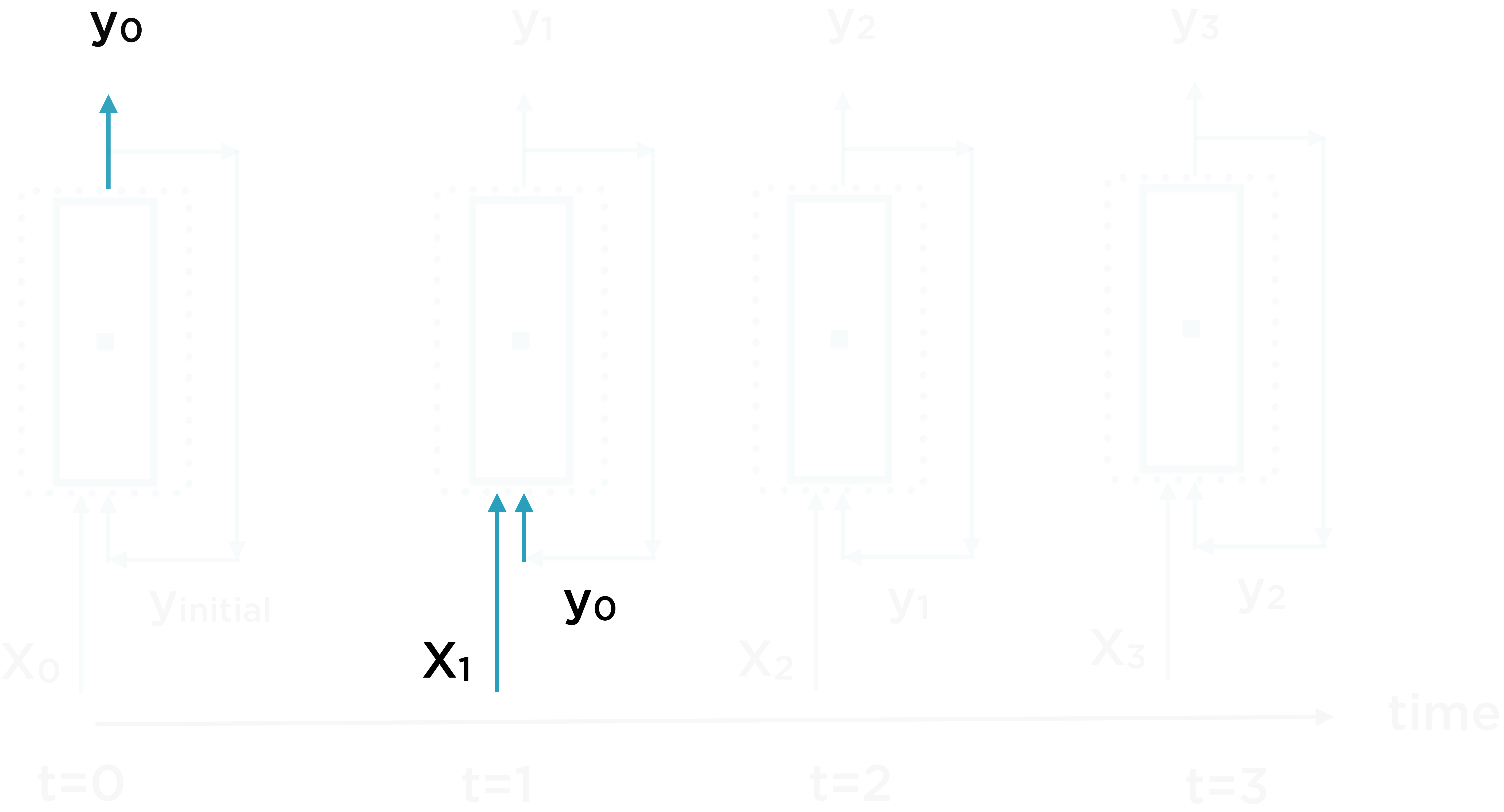
Depends upon

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

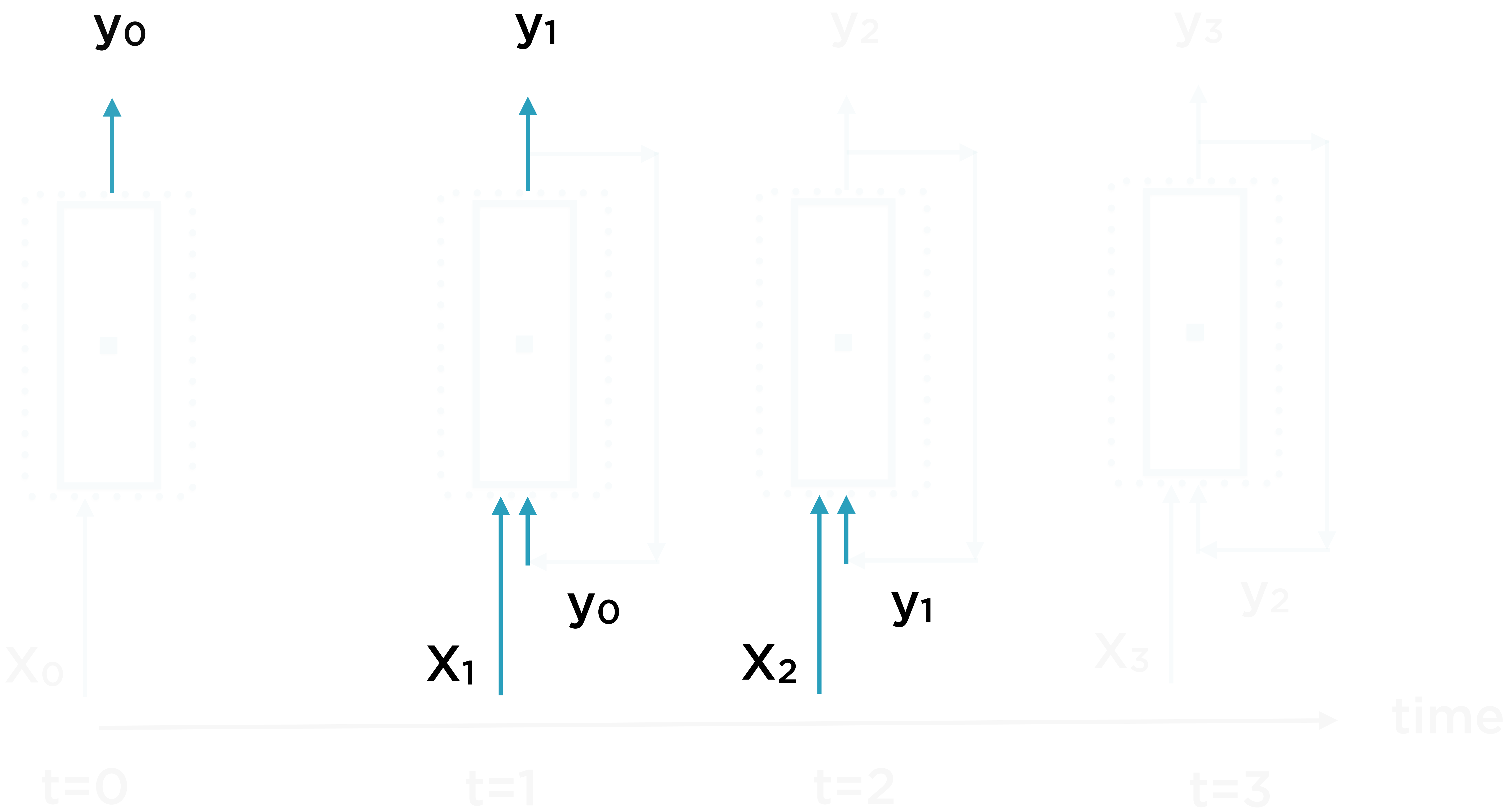
# Unrolling Through Time



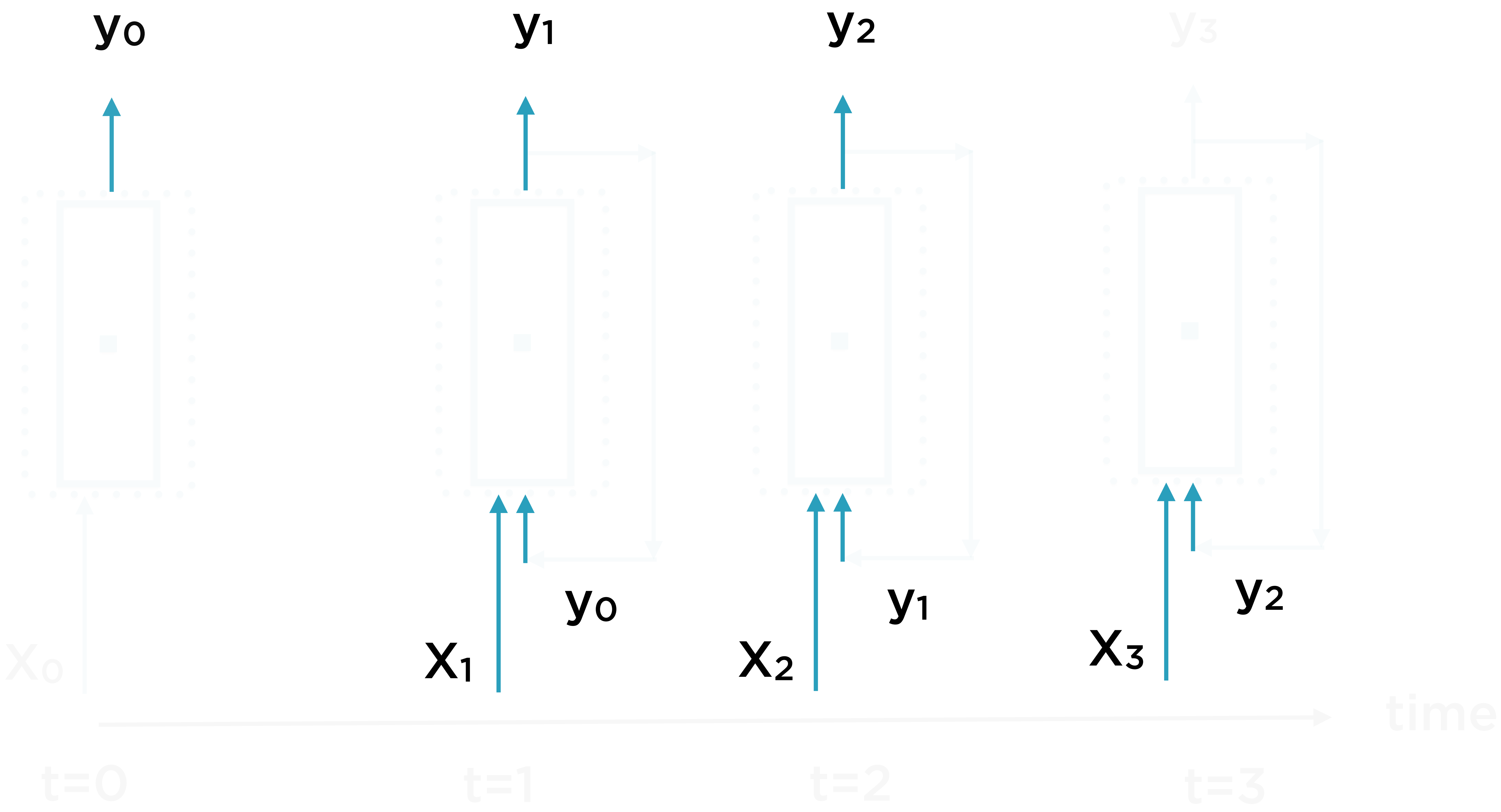
# Unrolling Through Time



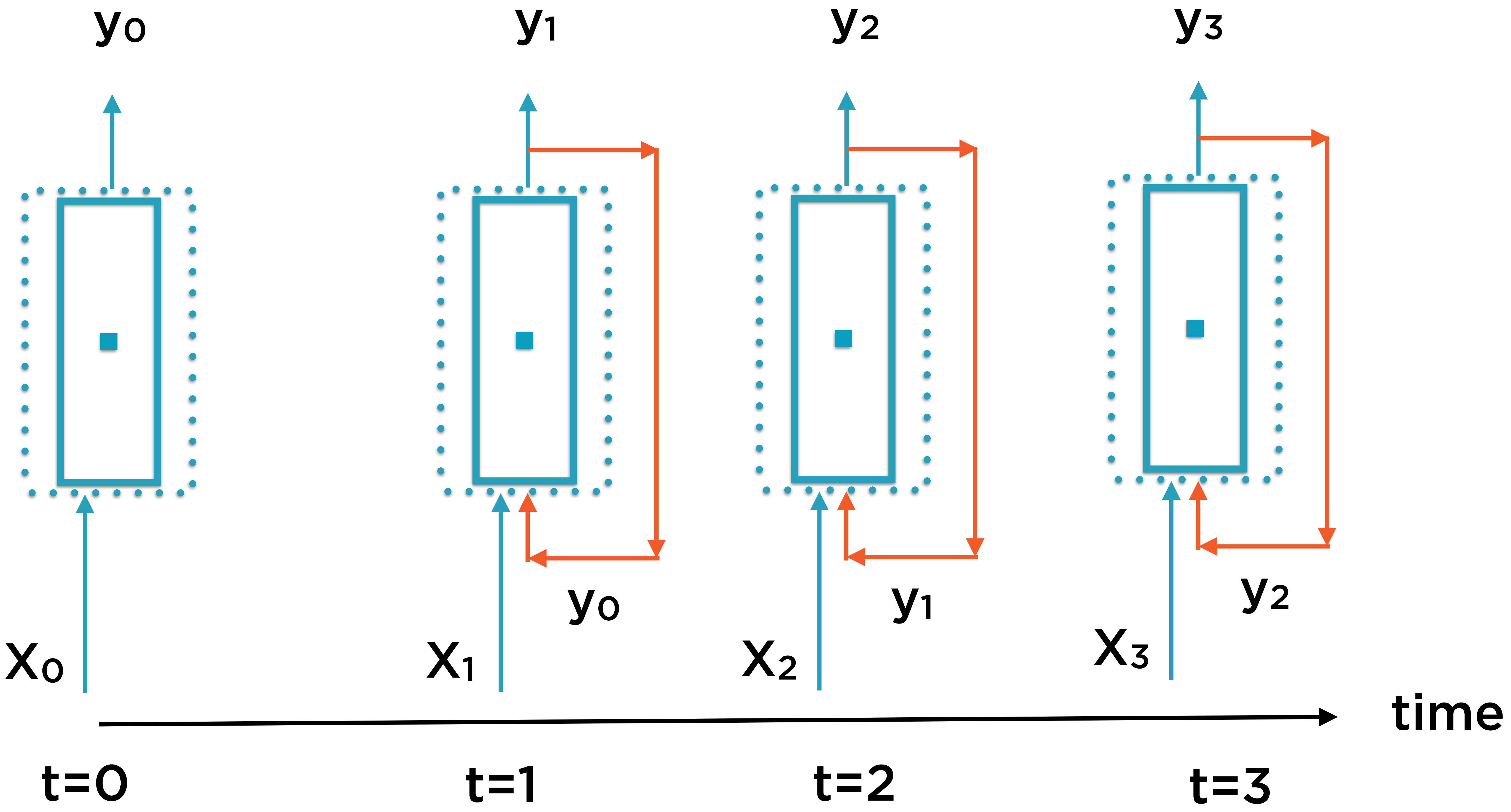
# Unrolling Through Time



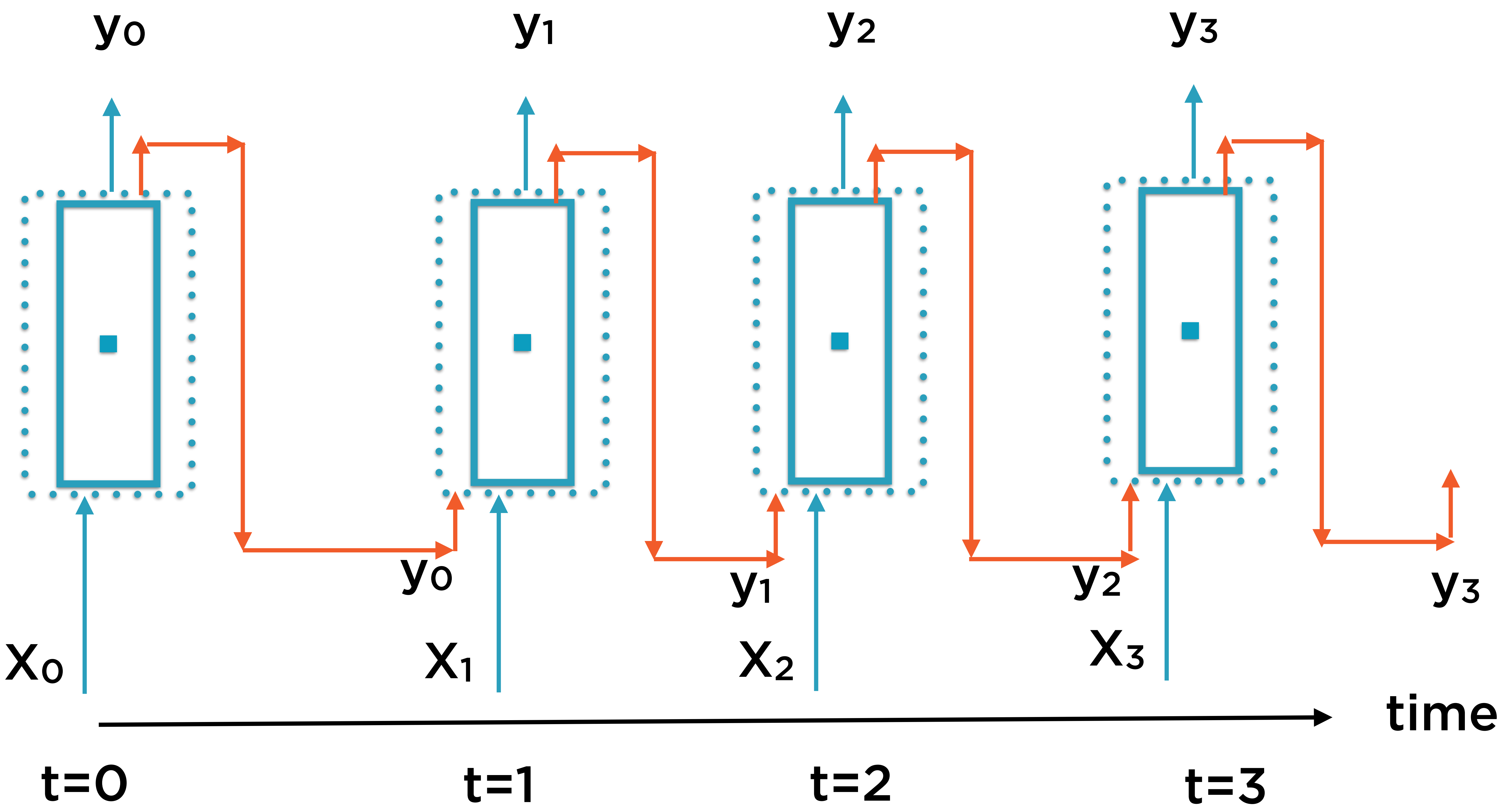
# Unrolling Through Time

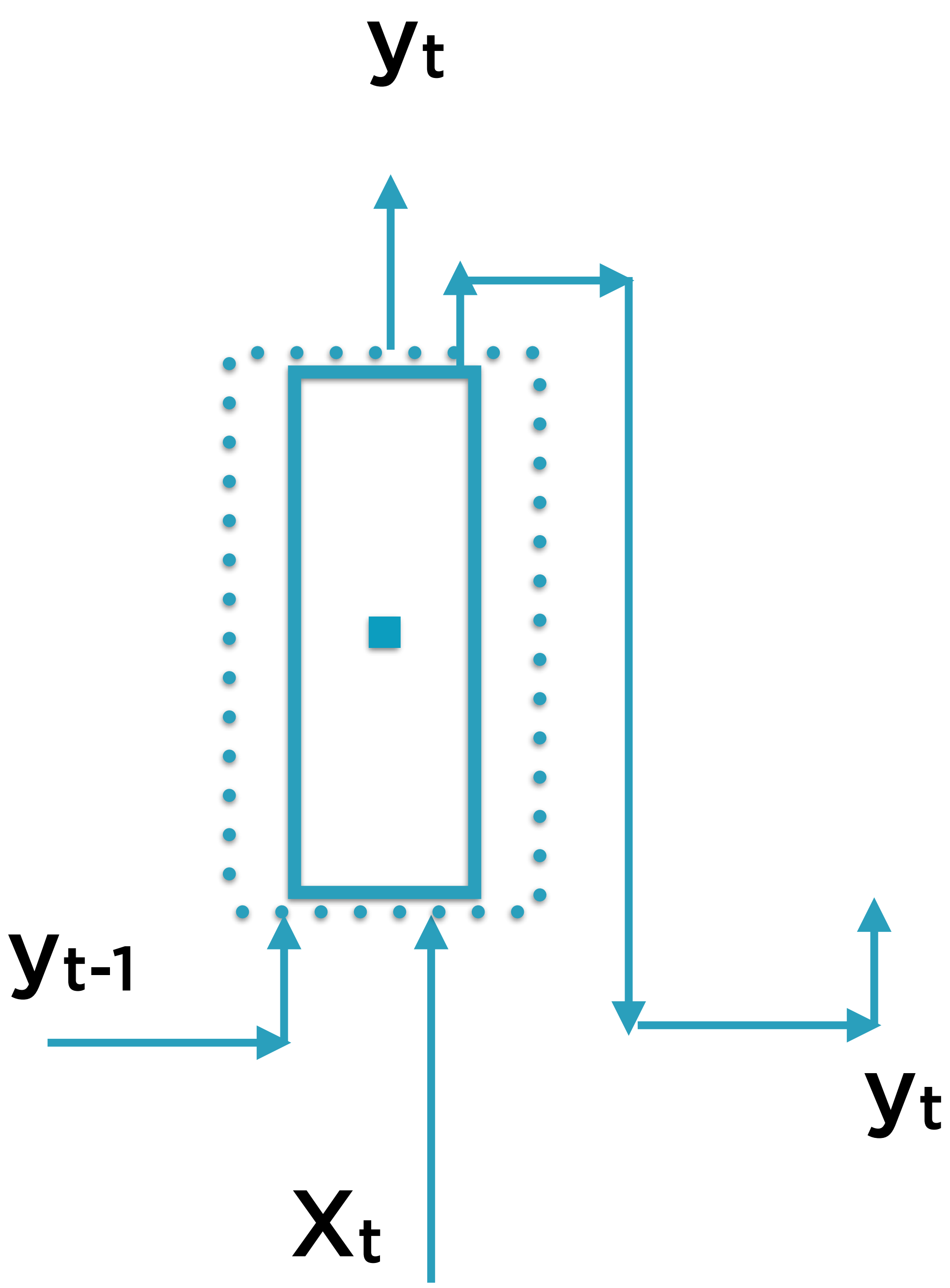


# Unrolling Through Time



# Unrolling Through Time






# Recurrent Neuron

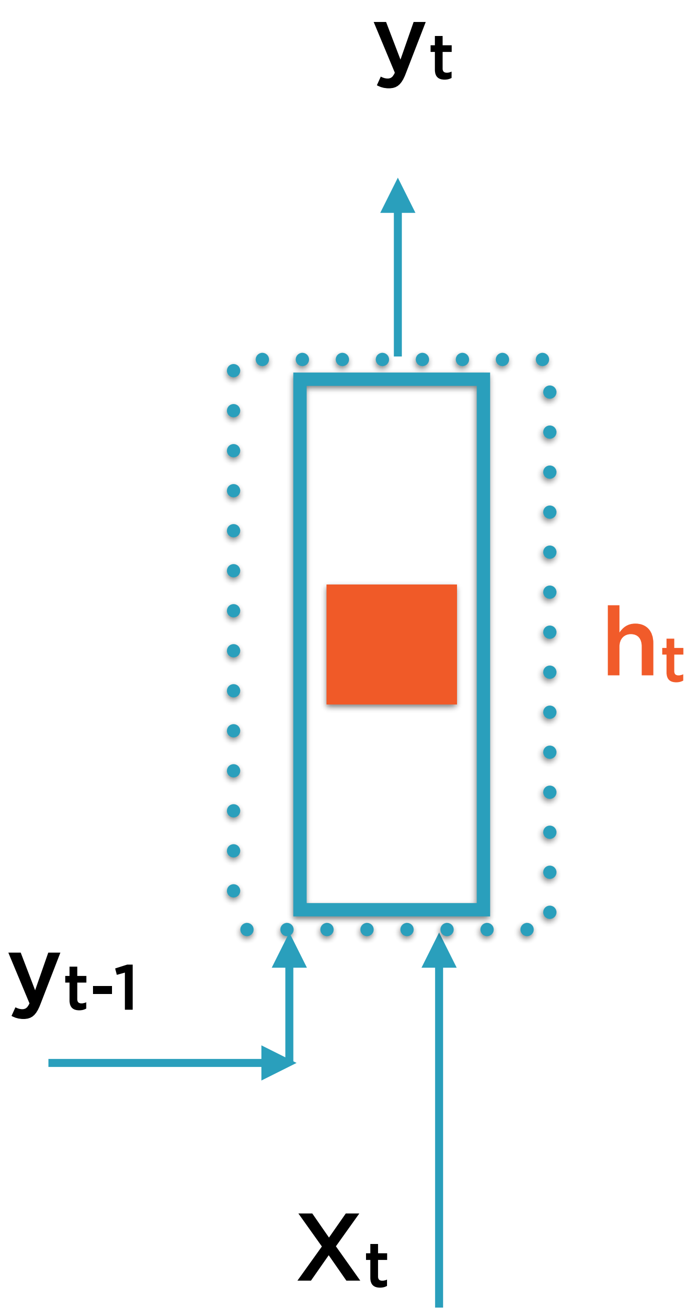
**Regular neuron: input is feature vector, output is scalar**

`Y = activation(Wx + b)`

**Recurrent neuron: output is vector too**

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

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



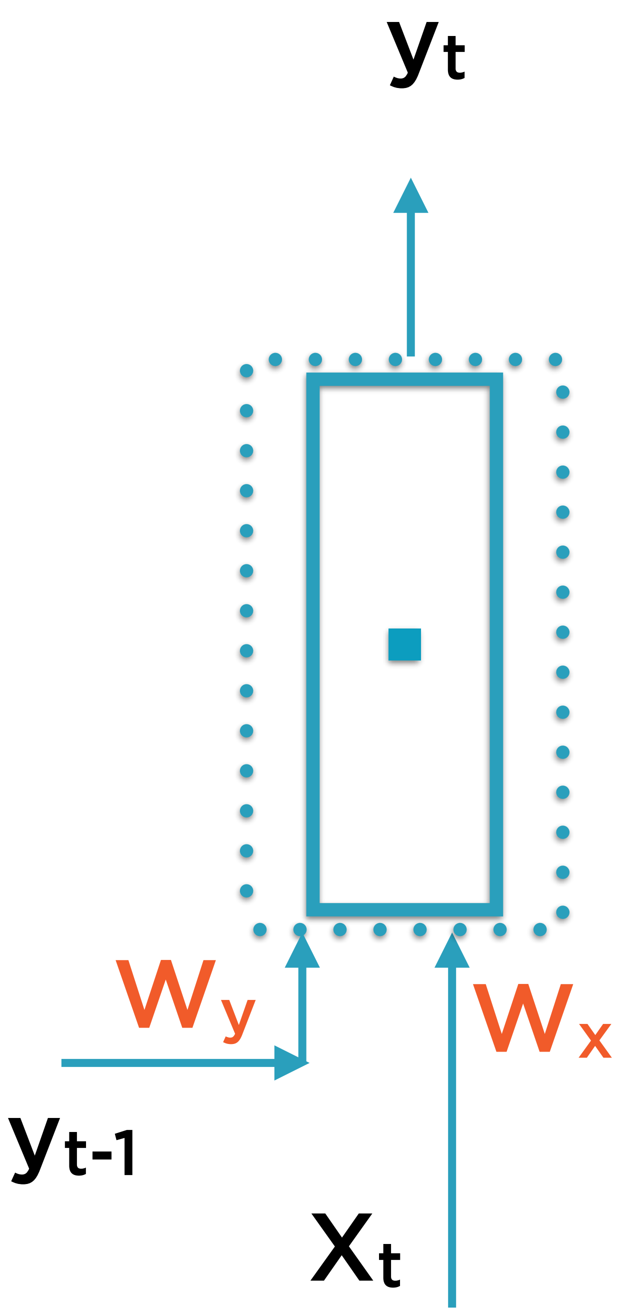

# Memory and State

Recurrent neurons remember the past

They possess ‘memory’

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

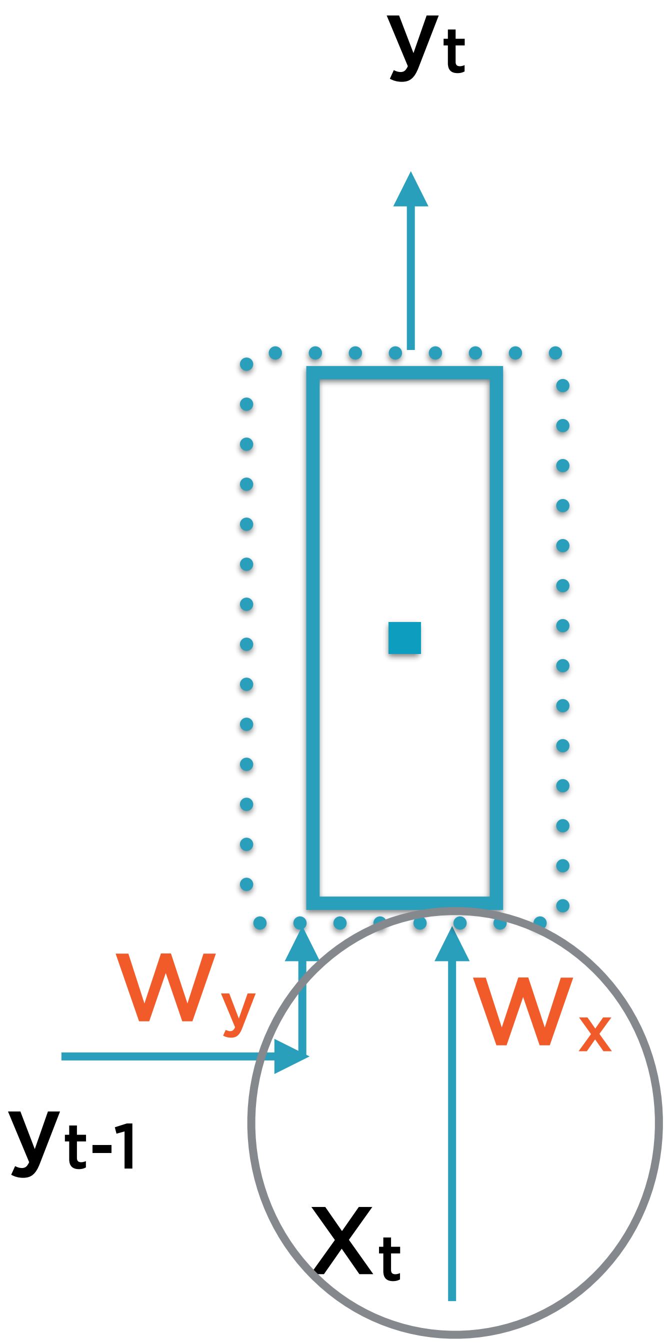
The internal state is represented by $h_t$




# Recurrent Neuron

Now, each neuron has two weight vectors

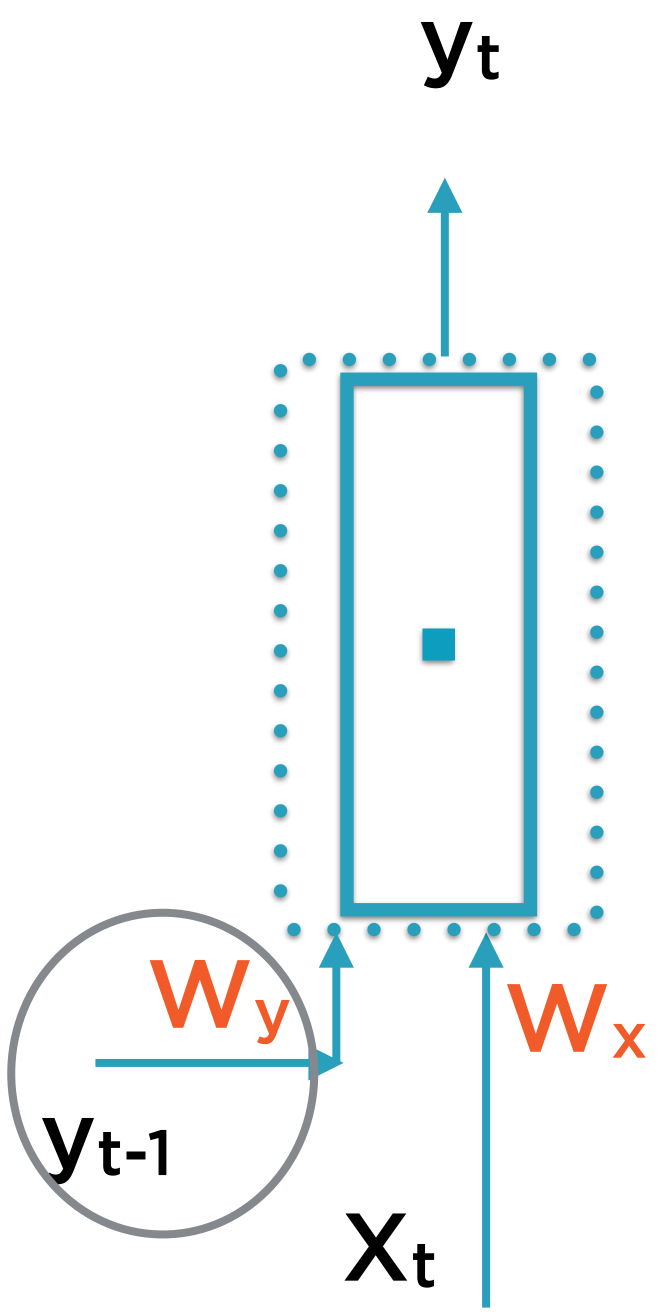
$W_x$, $W_y$




# Recurrent Neuron

Now, each neuron has two weight vectors

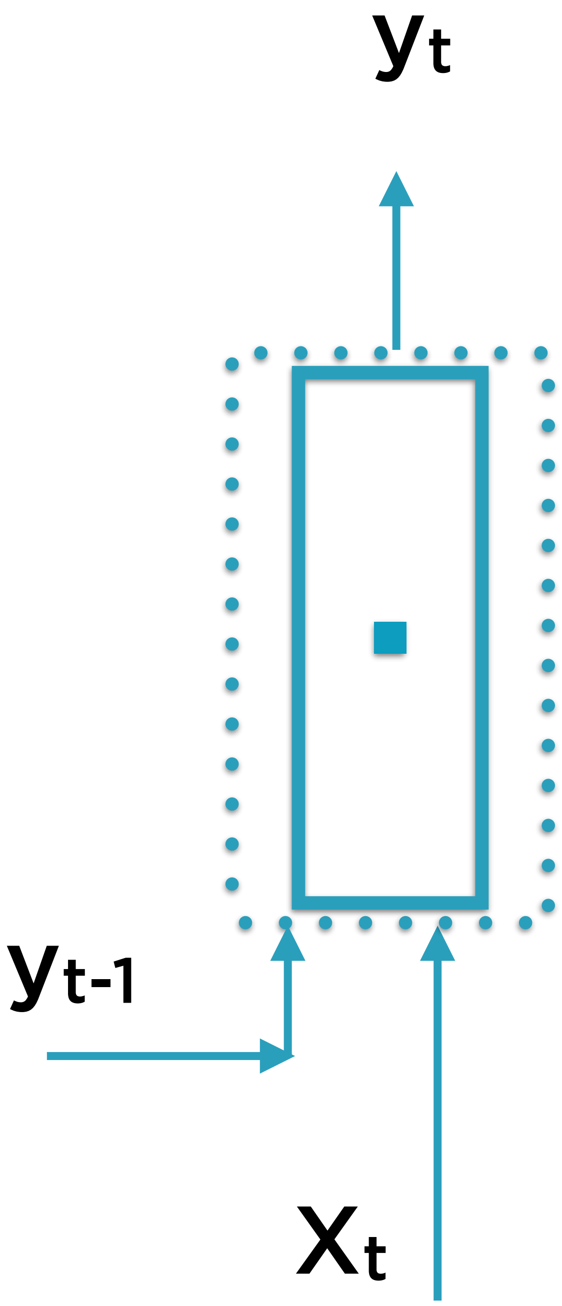
$W_x$, $W_y$




# Recurrent Neuron

Now, each neuron has two weight vectors

$W_x$, $W_y$




# Recurrent Neuron

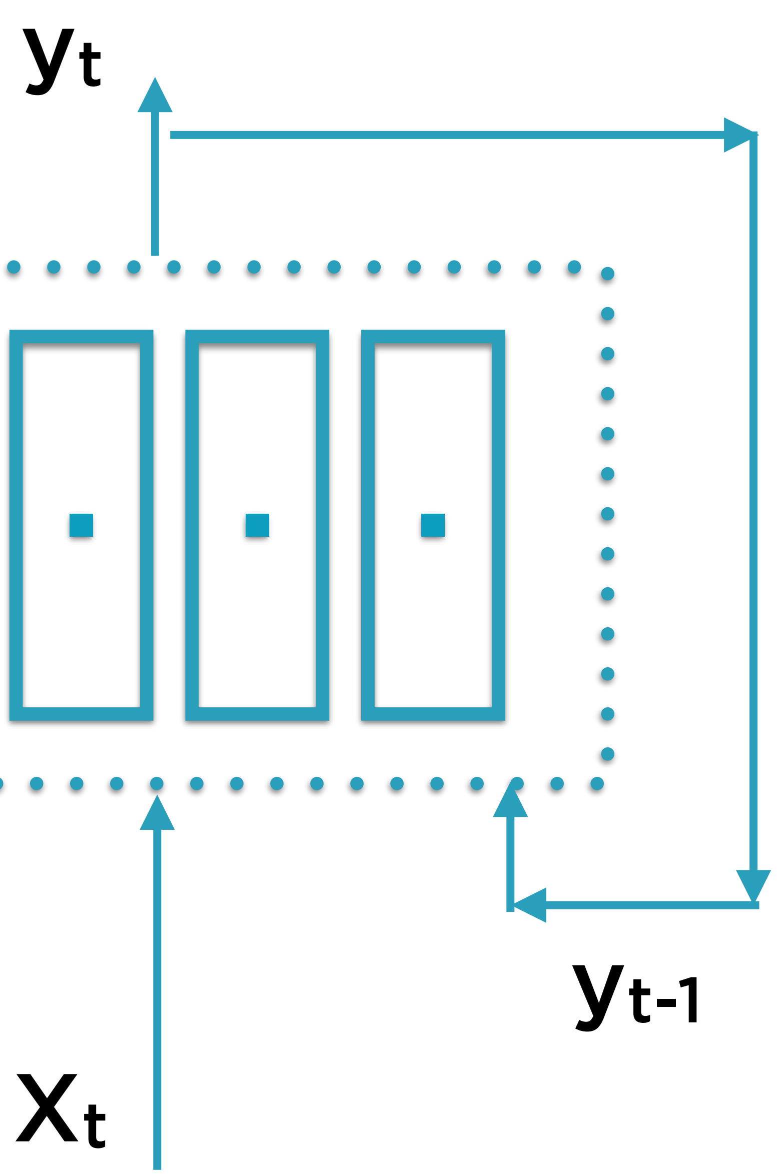
Output of neuron as a whole is given as

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

($\Phi$ is the activation function)

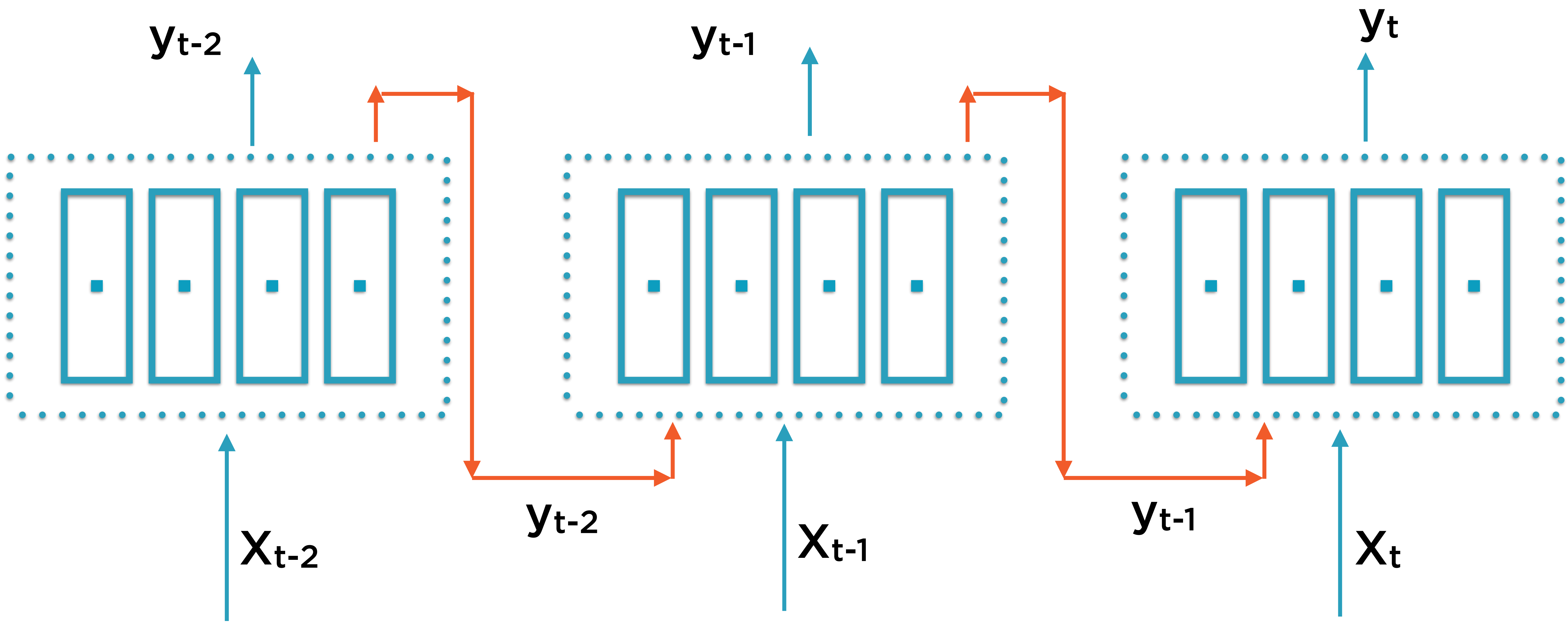
# Training a Recurrent Neural Network

# 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

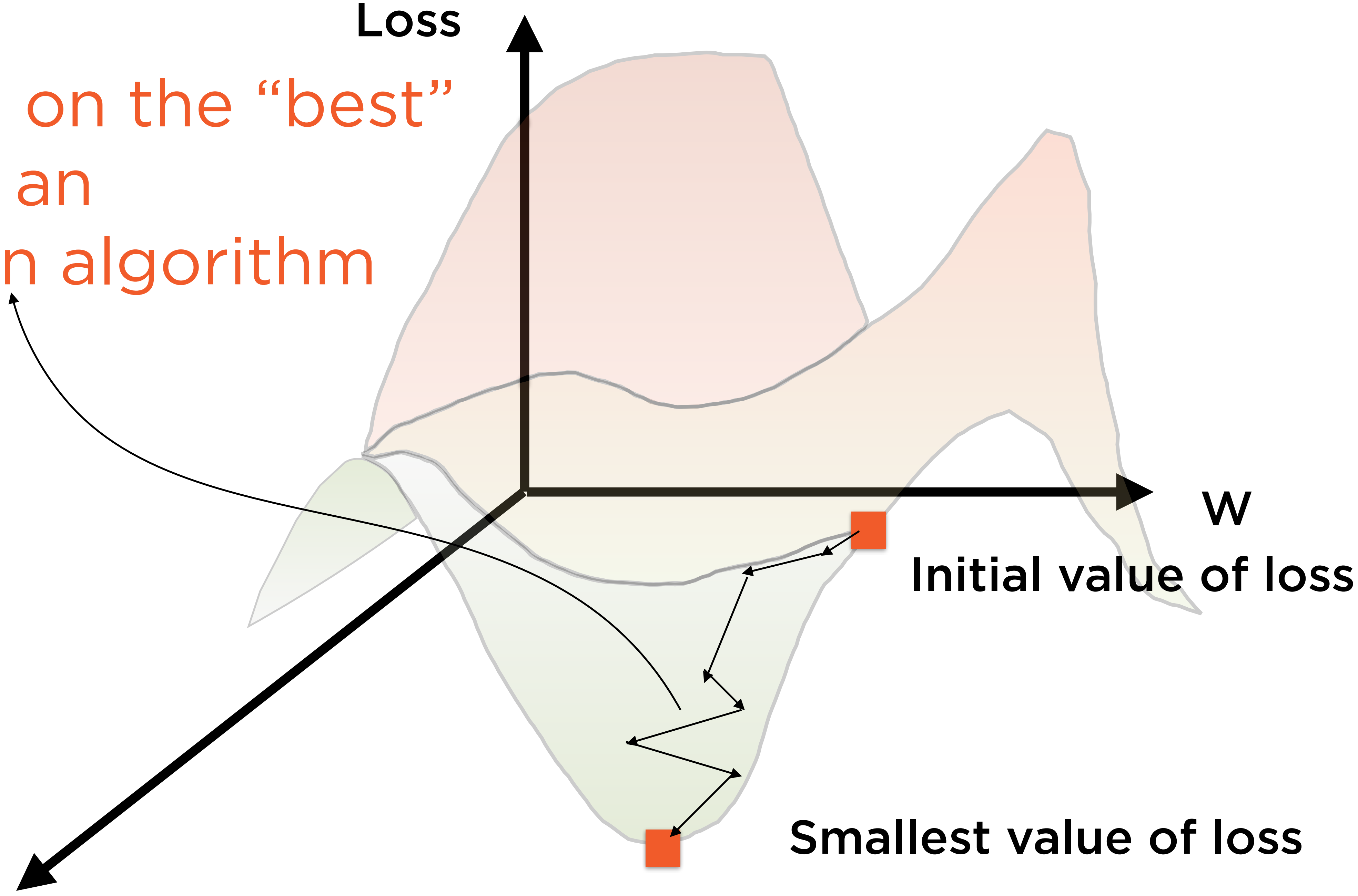



**The cells unrolled through time form the layers of the neural network**

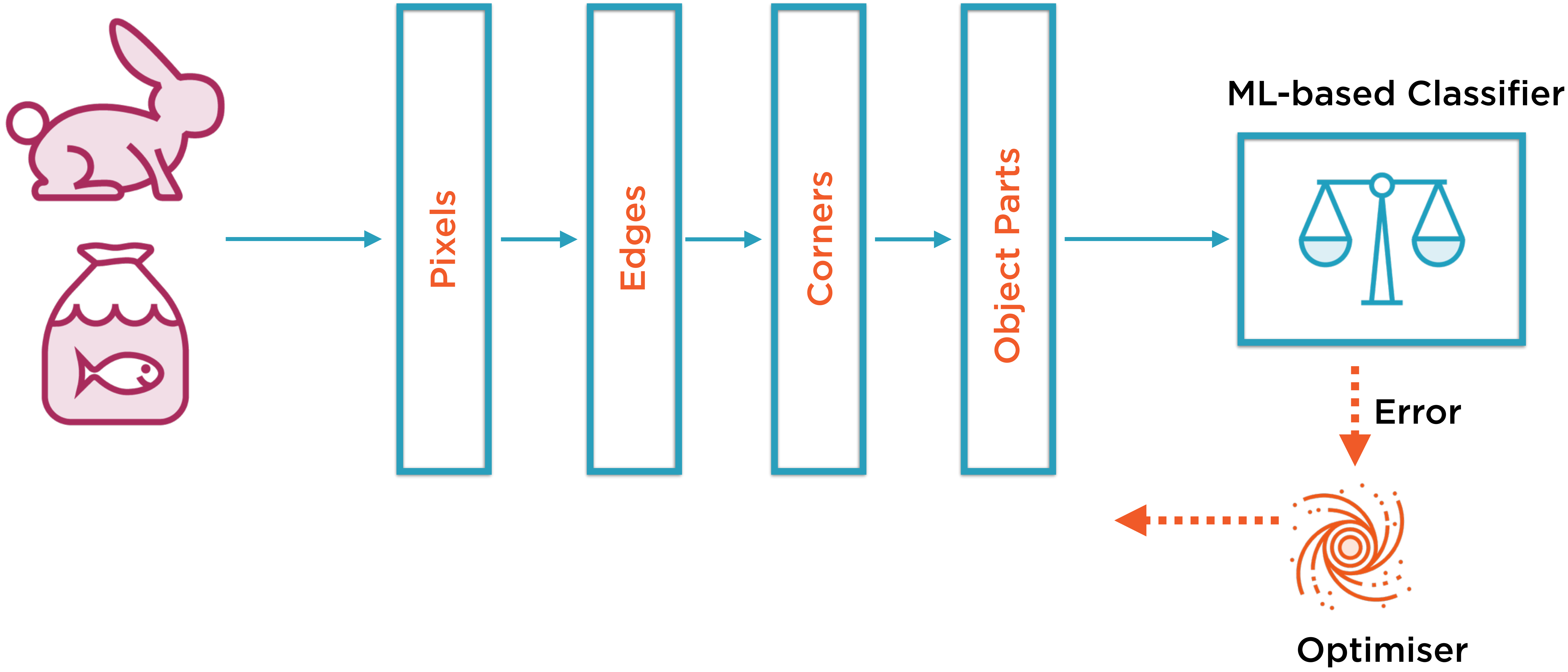
The actual training of a neural network happens via Gradient Descent Optimization

# Gradient Descent

Converging on the “best” value using an optimization algorithm



# Back Propagation Through Time



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

# Learning the Distant Past

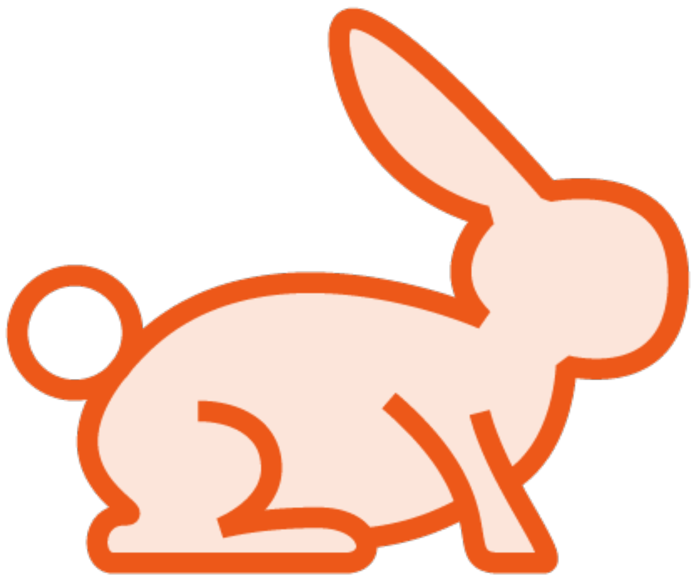
**The unrolled RNN will be very, very deep - many layers to train, the gradient has to be propagated a long way**

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

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

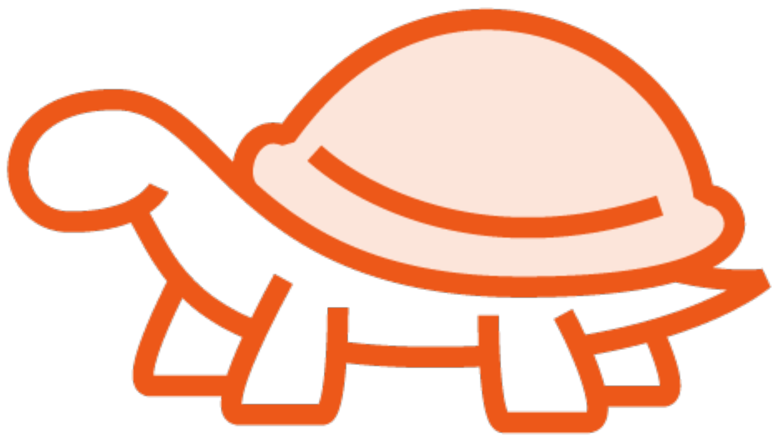
Deep neural networks are prone to overfitting

# Overfitting



**Low Training Error**

Model does very well in training...



**High Test Error**

...but poorly with real data

# Preventing Overfitting

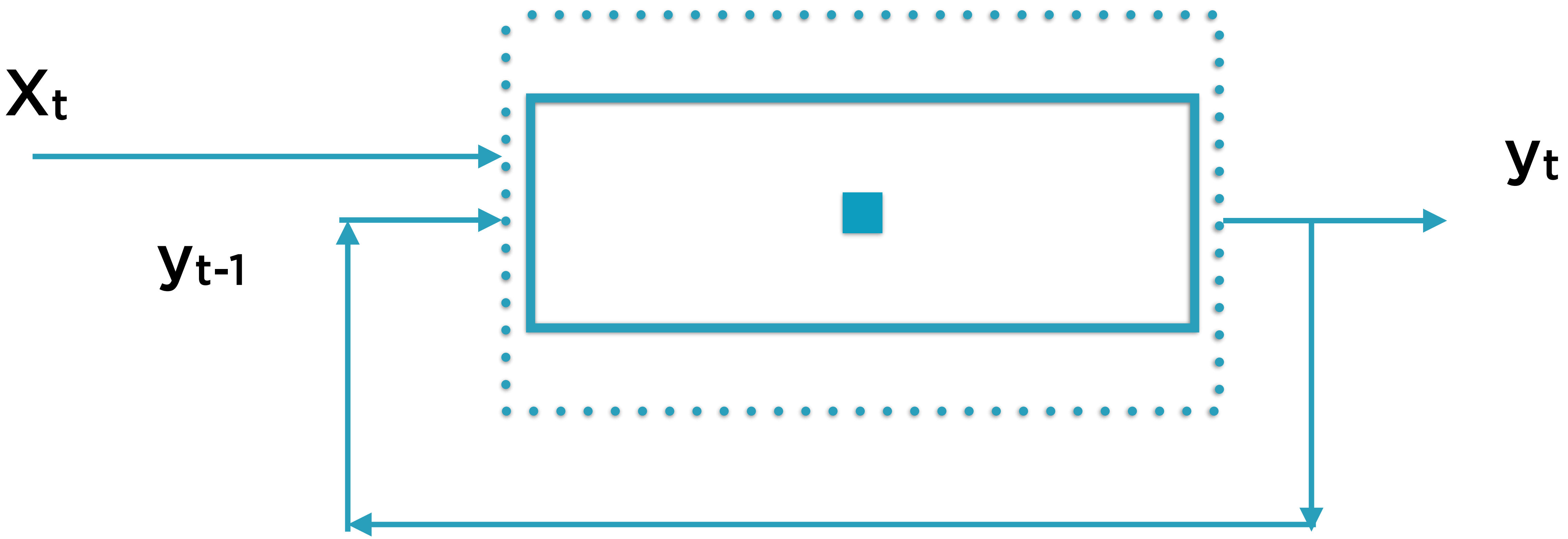

**Regularization - Penalize complex models**

**Cross-validation - Distinct training and validation phases**

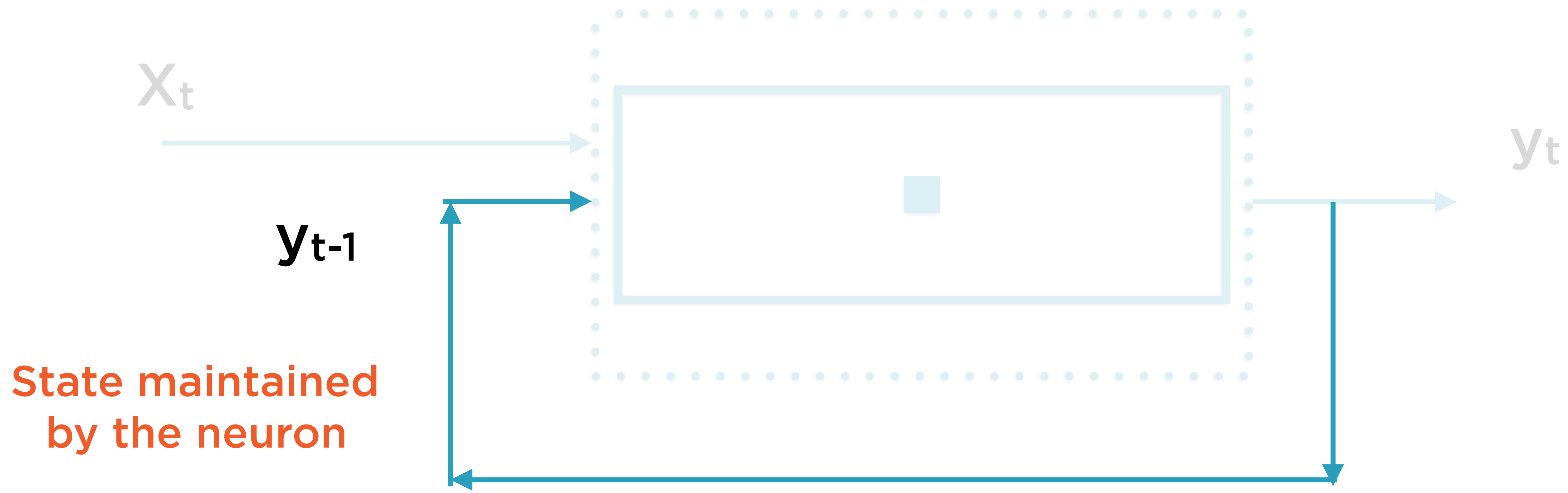

**Dropout - Intentionally turn off some neurons during training**

# Long Memory RNN Cells

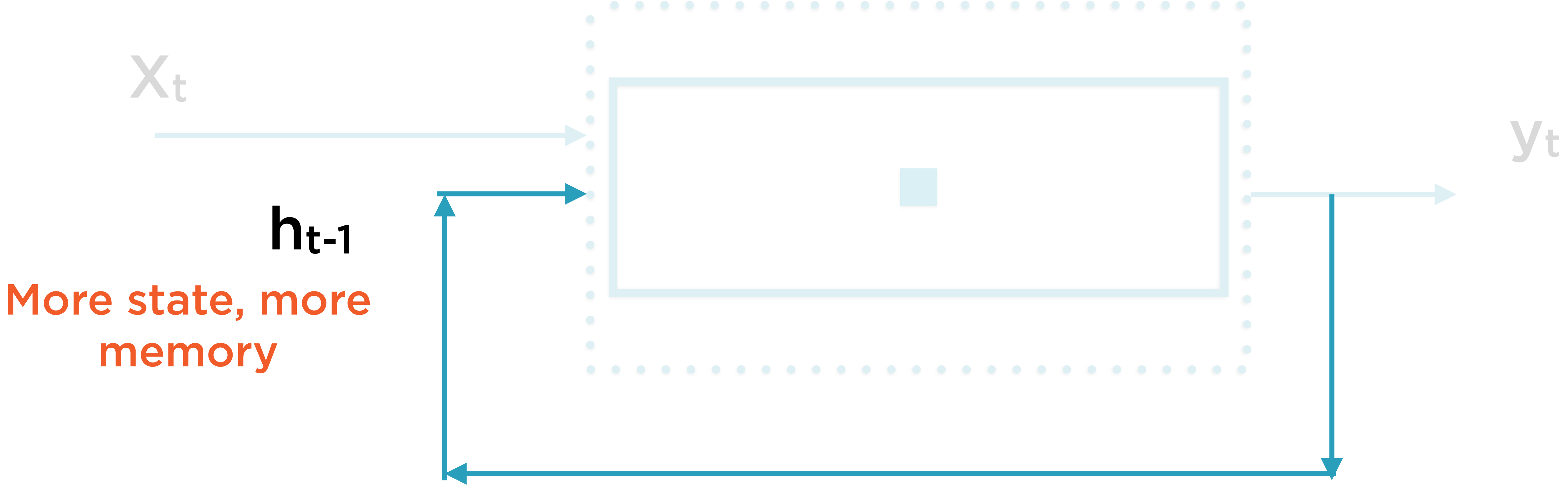
# Simplest Recurrent Neuron

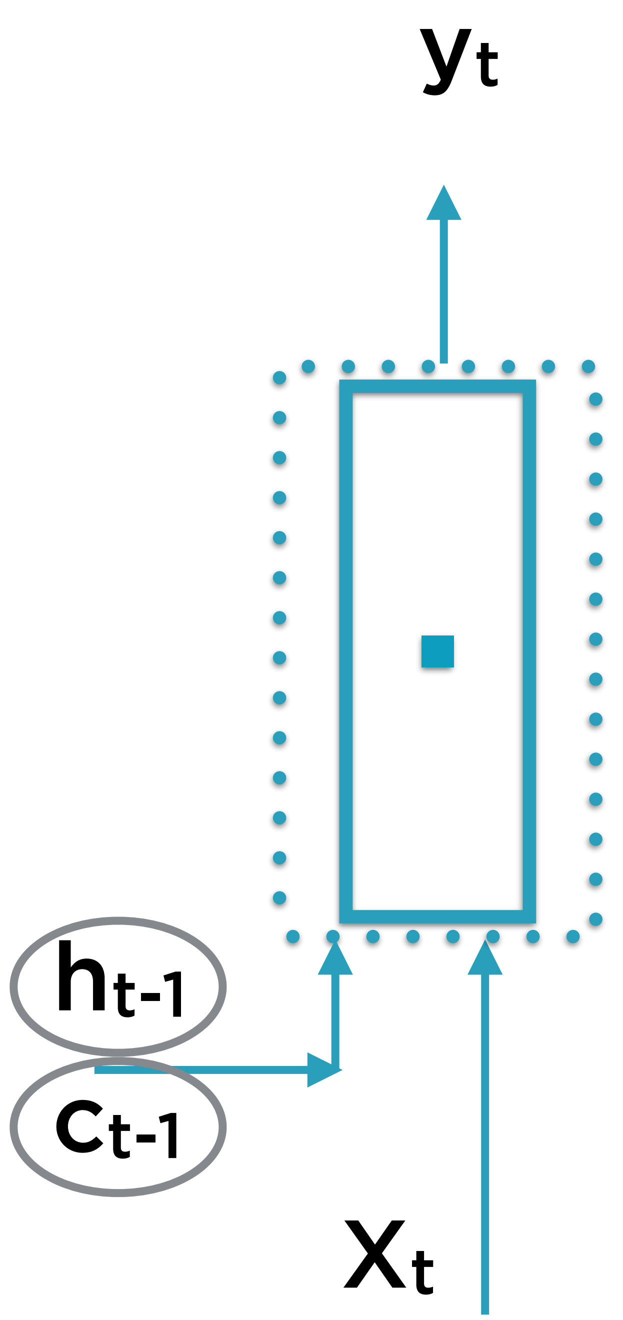


# Simplest Recurrent Neuron



# Long Memory Recurrent Neuron






# Long Memory RNNs

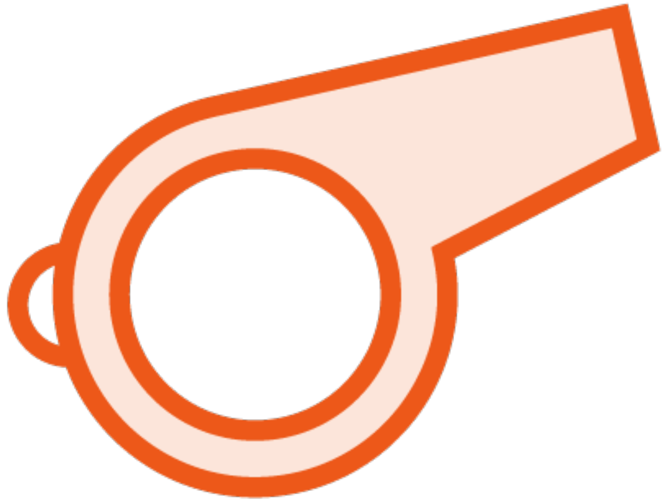
**Increase the amount of state in neuron**

**Effect is to increase memory of neuron**

**Could explicitly add:**

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

# Long Memory RNNs



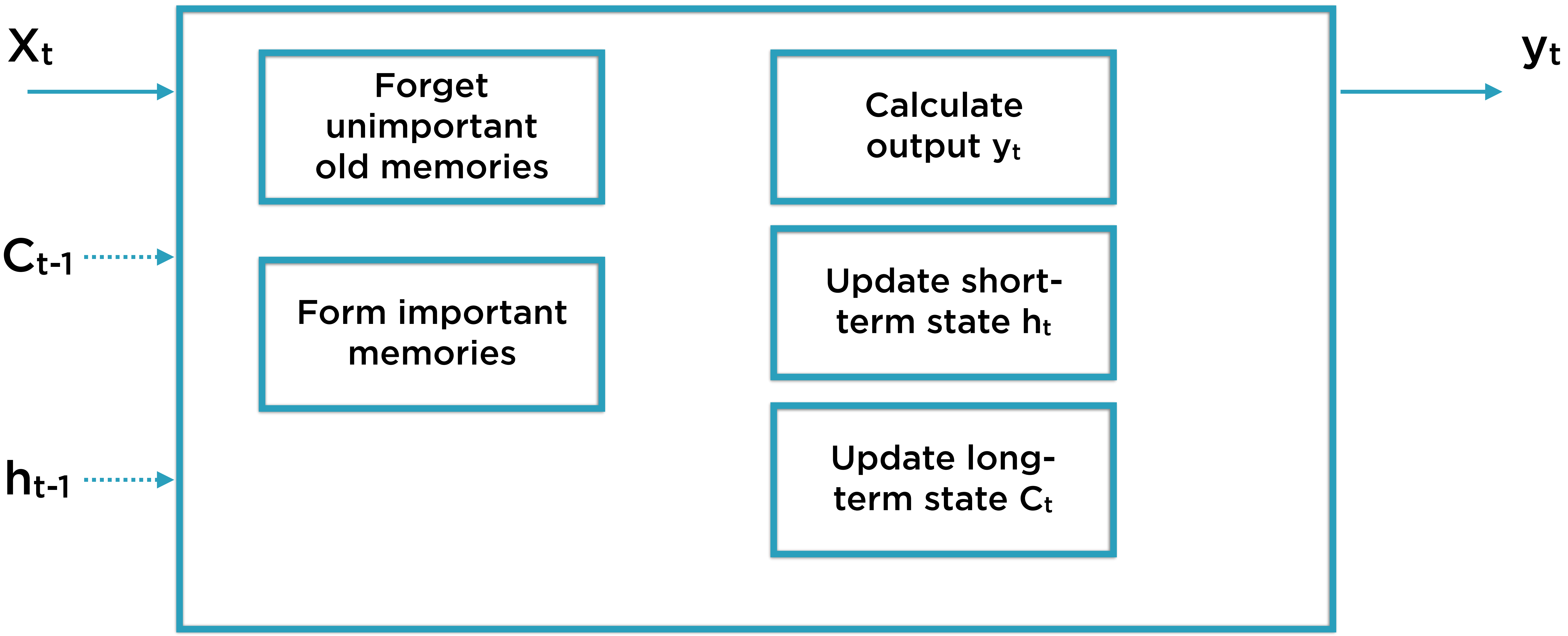
**Advantages in Training**

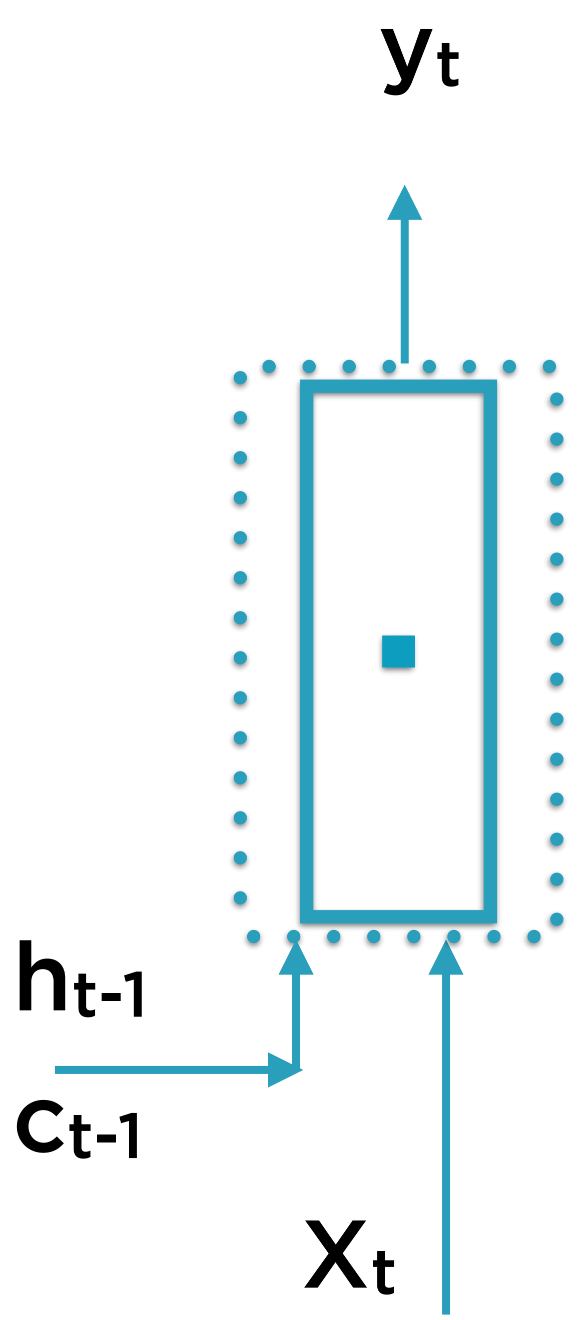
Faster training, nicer gradients

**Advantages in Prediction**

Works well with even very deep NNs

# Long/Short-Term Memory Cell (LSTM)






# LSTM Cells

Functionally like basic RNN cell

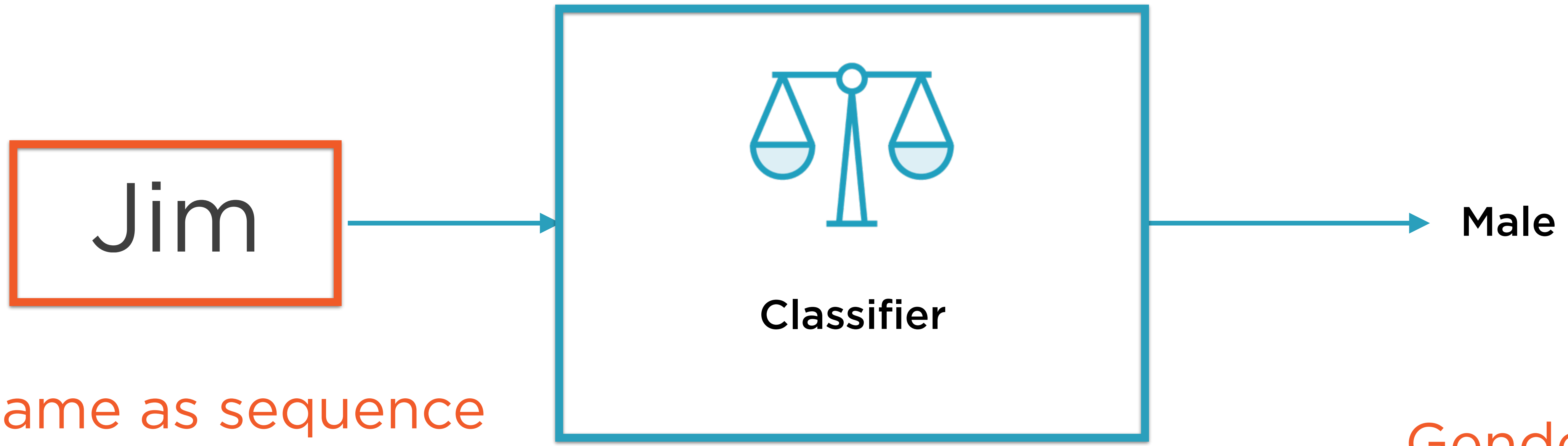
Performance far better

Amazing success at long-term patterns

Long text sequences, time series…

# Gender Prediction Based on Names

# Gender Prediction Based on Names



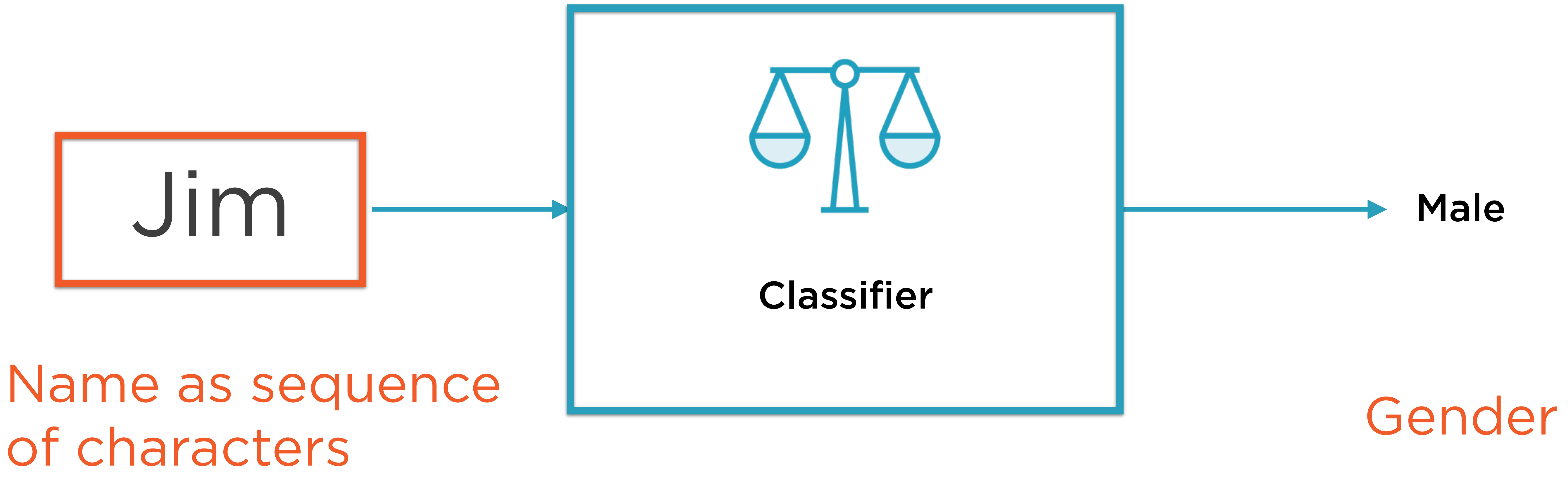

Name as sequence of characters

Gender

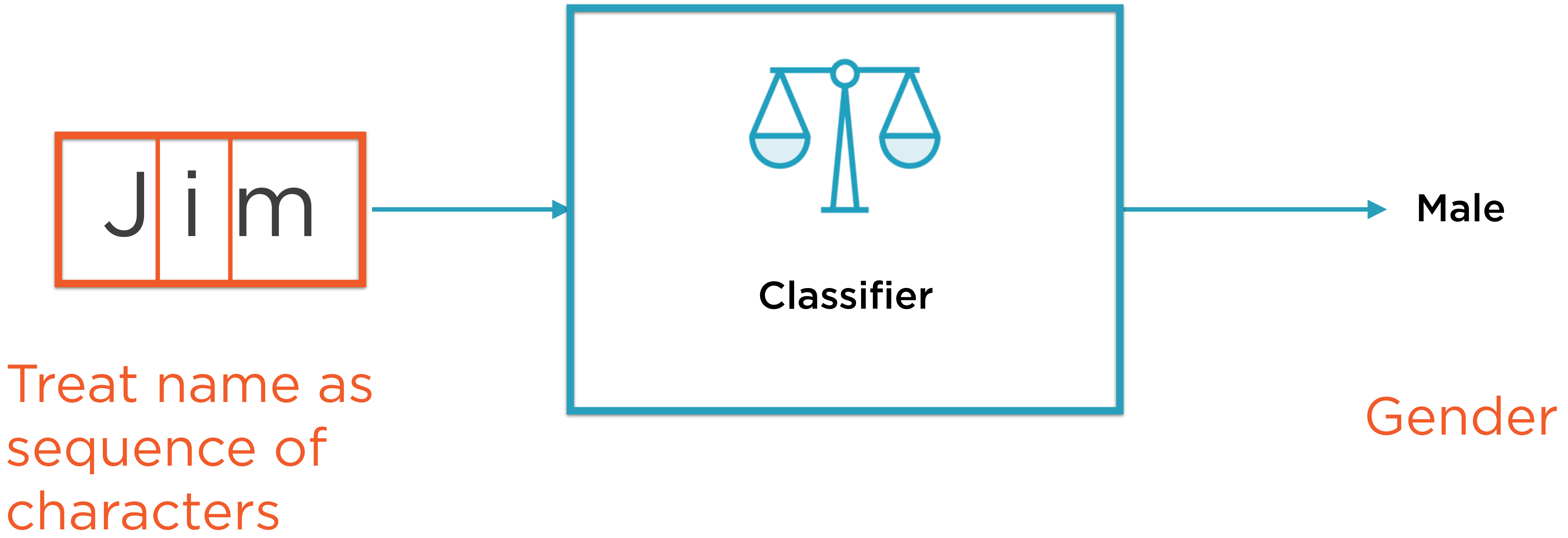
In PyTorch, because of dynamic computation graphs we do not need to pad names to be of the same length

This is a significant improvement over TensorFlow

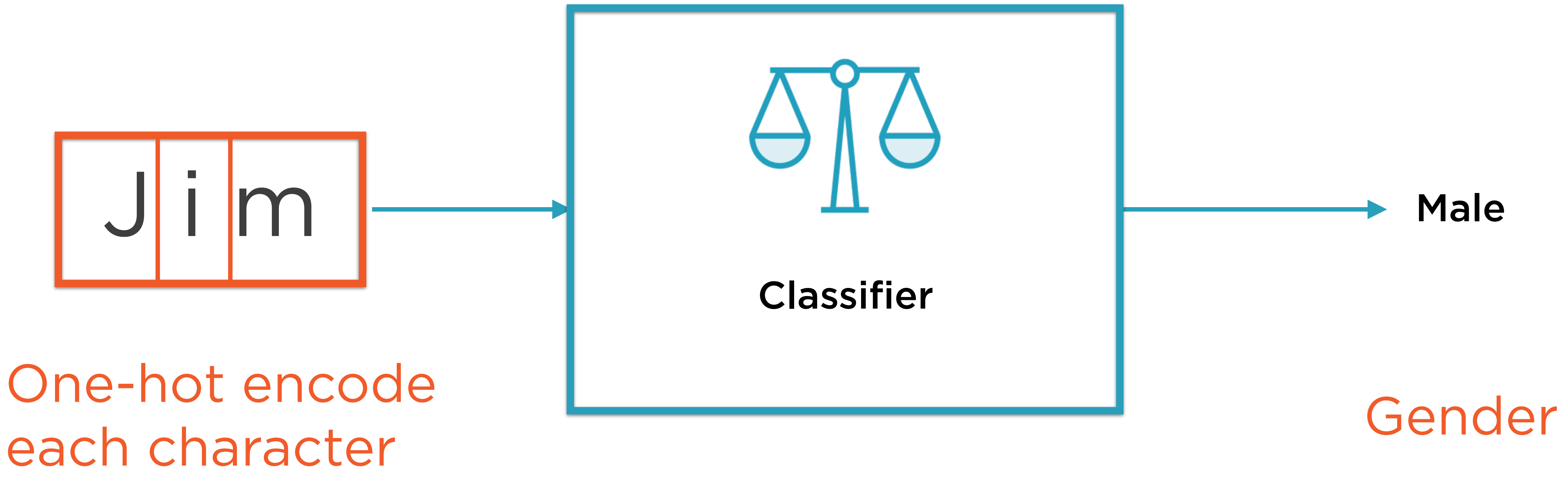
# Gender Prediction Based on Names



# Gender Prediction Based on Names



# Gender Prediction Based on Names



# One-hot Encoded Characters

| Letter | a | … | j | … | i | … | m | … |
|---|---|---|---|---|---|---|---|---|
| J | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| i | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| m | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

26 elements

# One-hot Encoded Characters

| Letter | a | ... | j | ... | i | ... | m | ... |
|---|---|---|---|---|---|---|---|---|
| J | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| i | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| m | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

26 elements

# One-hot Encoded Characters

| Letter | a | ... | j | ... | i | ... | m | ... |
|---|---|---|---|---|---|---|---|---|
| J | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| i | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| m | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

26 elements

# One-hot Encoded Characters

| Letter | a | ... | j | ... | i | ... | m | ... |
|---|---|---|---|---|---|---|---|---|
| J | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| i | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| m | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

26 elements

# Gender Prediction Using RNNs

**Prepare Data**

Download and extract

File manipulation - simple

**Encode Names as Tensors**

No padding needed in PyTorch!

Sequence of one-hot encoded characters

**Create Neural Network**

Two linear layers

LogSoftMax output layer

**Train Model**

Backprop with loss.backward()

NLL loss to go with LogSoftMax

**Evaluate Model**

Confusion Matrix

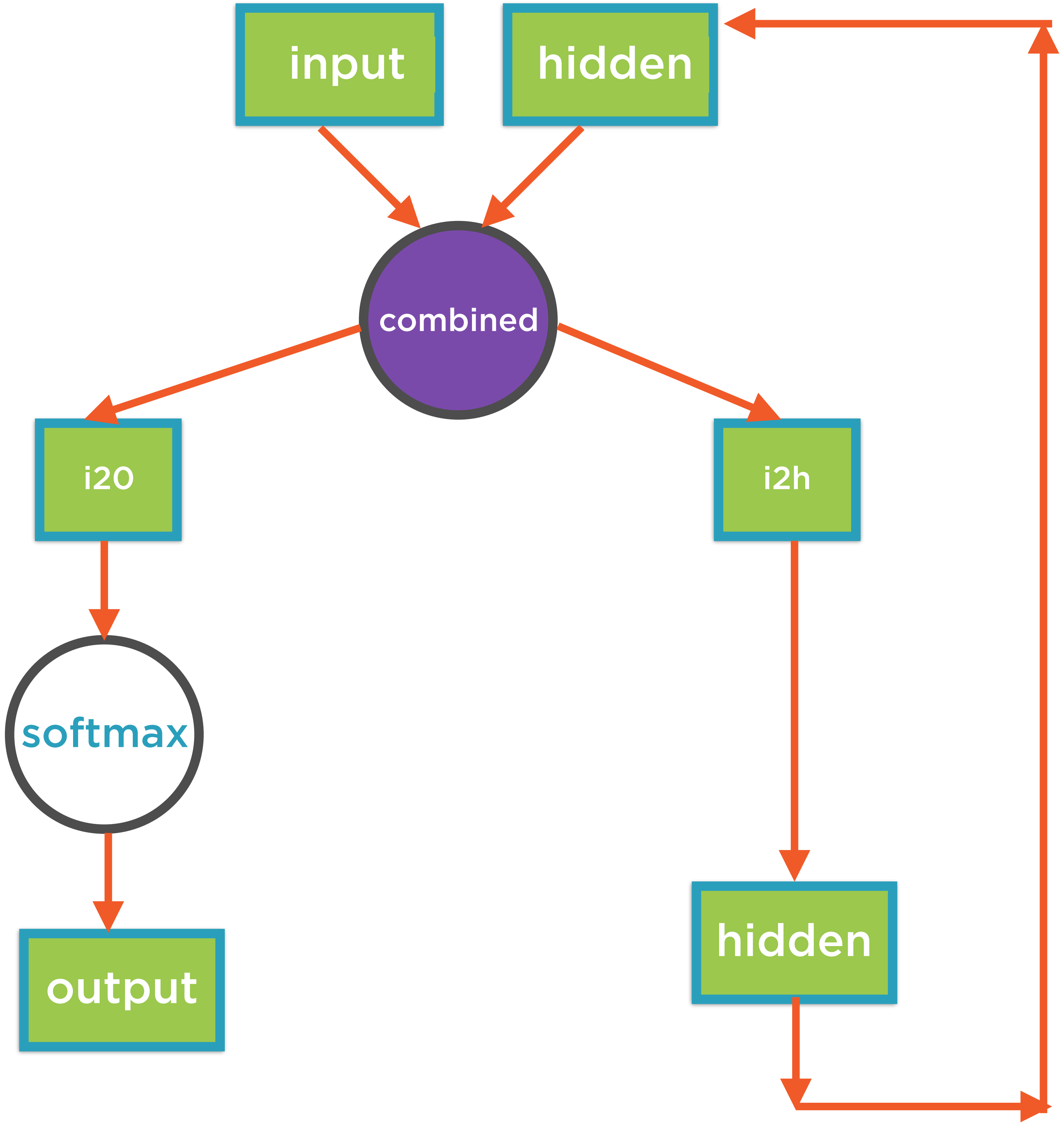
Similar to training, but no backprop

**Use in Prediction**

Ready to go!

Could add REST API server

```
class RNN(nn.Module):

# In constructor

  self.i2h = nn.Linear()

  self.i2o = nn.Linear()

  self.softmax = nn.LogSoftmax()

# Forward method

  def forward(self, input, hidden):

        combined = torch.cat((input, hidden), 1)

        hidden = self.i2h(combined)

        output = self.i2o(combined)

        output = self.softmax(output)

        return output, hidden
```



# Demo

**Gender prediction/classification from names using RNNs**

# Confusion Matrix in Evaluating Classifiers

# Confusion Matrix

Predicted Labels

Actual Label

| | Cancer | No Cancer |
|---|---|---|
| **Cancer** | **10 instances** | **4 instances** |
| **No Cancer** | **5 instances** | **1000 instances** |

# Confusion Matrix

Predicted Labels

Actual Label

| | Cancer | No Cancer |
|---|---|---|
| Cancer | **10** instances | **4** instances |
| No Cancer | **5** instances | **1000** instances |

# Confusion Matrix

Predicted Labels

Actual Label

| | Cancer | No Cancer |
|---|---|---|
| Cancer | 10 | 4 |
| No Cancer | 5 | 1000 |

# True Positive

Predicted Labels

Actual Label

| | Cancer | No Cancer |
|---|---|---|
| Cancer | 10 TP | 4 |
| No Cancer | 5 | 1000 |

Actual Label = Predicted Label

# False Positive

| Actual Label \ Predicted Labels | Cancer | No Cancer |
|---|---|---|
| Cancer | 10 | 4 |
| No Cancer | 5 FP | 1000 |

Actual Label ≠ Predicted Label

# True Negative

Predicted Labels

Actual Label

| | Cancer | No Cancer |
|---|---|---|
| Cancer | 10 | 4 |
| No Cancer | 5 | 1000 TN |

Actual Label = Predicted Label

# False Negative

Predicted Labels

Actual Label

| | Cancer | No Cancer |
|---|---|---|
| Cancer | 10 | 4 FN |
| No Cancer | 5 | 1000 |

Actual Label ≠ Predicted Label

# Accuracy

Predicted Labels

Actual Label

| | Cancer | No Cancer |
|---|---|---|
| Cancer | TP 10 | FN 4 |
| No Cancer | FP 5 | TN 1000 |

$$\text{Accuracy} = \frac{\text{TP + TN}}{\text{Num Instances}} = \frac{1010}{1019} = 99.12\%$$

# Precision

| Actual Label | Predicted Labels: Cancer | Predicted Labels: No Cancer |
|---|---|---|
| Cancer | 10 TP | 4 FN |
| No Cancer | 5 FP | 1000 TN |

Precision = Accuracy when classifier flags cancer

# Recall

| Actual Label \ Predicted Labels | Cancer | No Cancer |
|---|---|---|
| Cancer | TP 10 | FN 4 |
| No Cancer | FP 5 | TN 1000 |

Recall = Accuracy when cancer actually present

# Summary

Recurrent Neural Networks (RNNs) are another common NN architecture

Great for working with sequences

Classification using RNNs

PyTorch has built-in support for RNNs

# Related Courses

Building Classification Models with TensorFlow

Building Unsupervised Learning Models with TensorFlow

Deep Learning Using TensorFlow and Apache MXNet on AWS SageMaker

Building and Deploying Keras in a Multi-cloud Environment