

Recurrent Neural Networks

Overview

RNNs consist of recurrent neurons i.e. neurons which have memory

Recurrent neurons feed back their output as an input

This makes them naturally well suited to learning time series data

Training RNNs relies on Back Propagation Through Time (BPTT)

Recurrent Neural Networks

$$y = f(x)$$

Machine Learning

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

X Causes Y



Cause

Explanatory variable



Effect

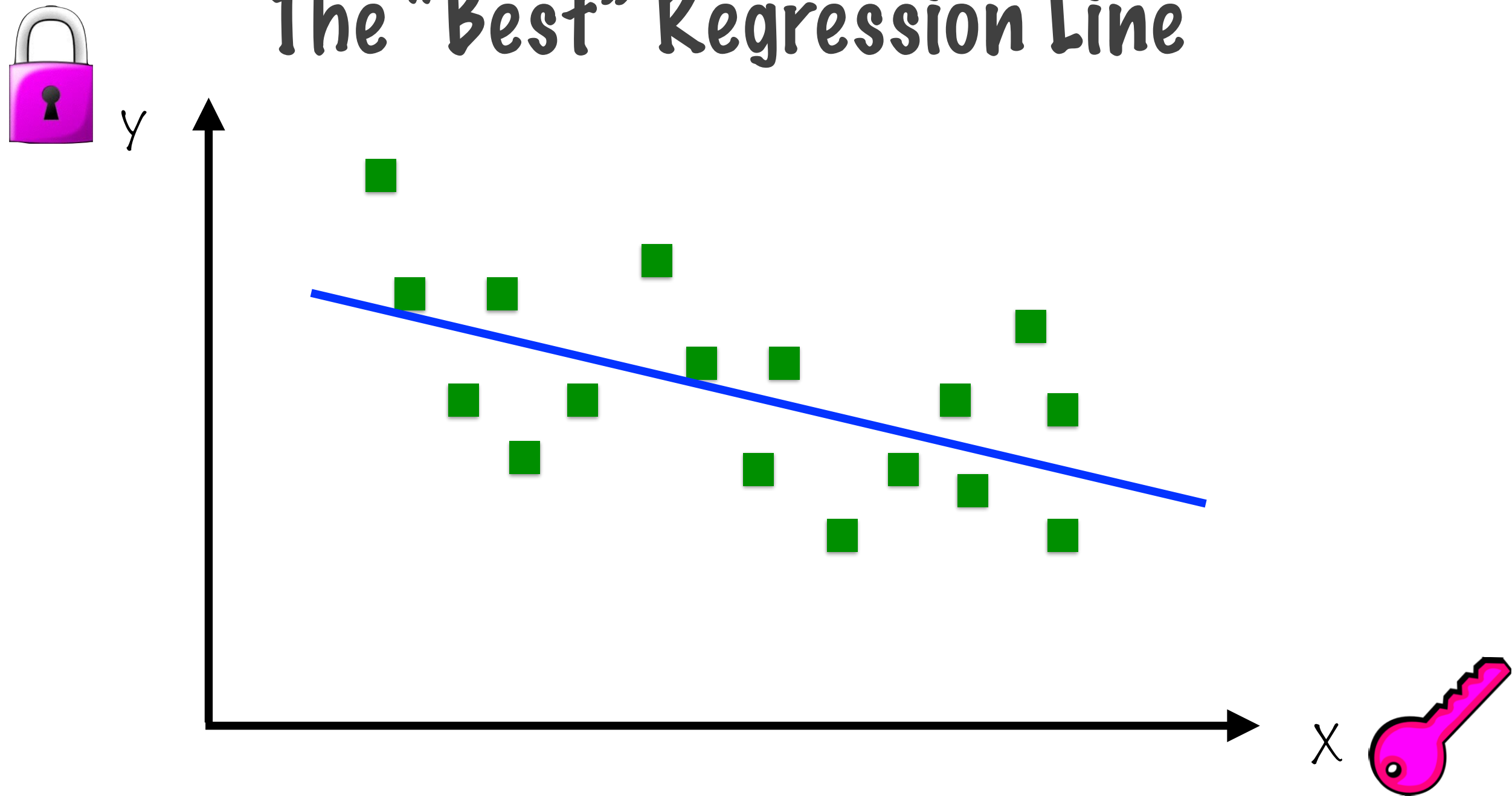
Dependent variable

$$y = Wx + b$$

$$\mathbf{f(x) = Wx + b}$$

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

The "Best" Regression Line



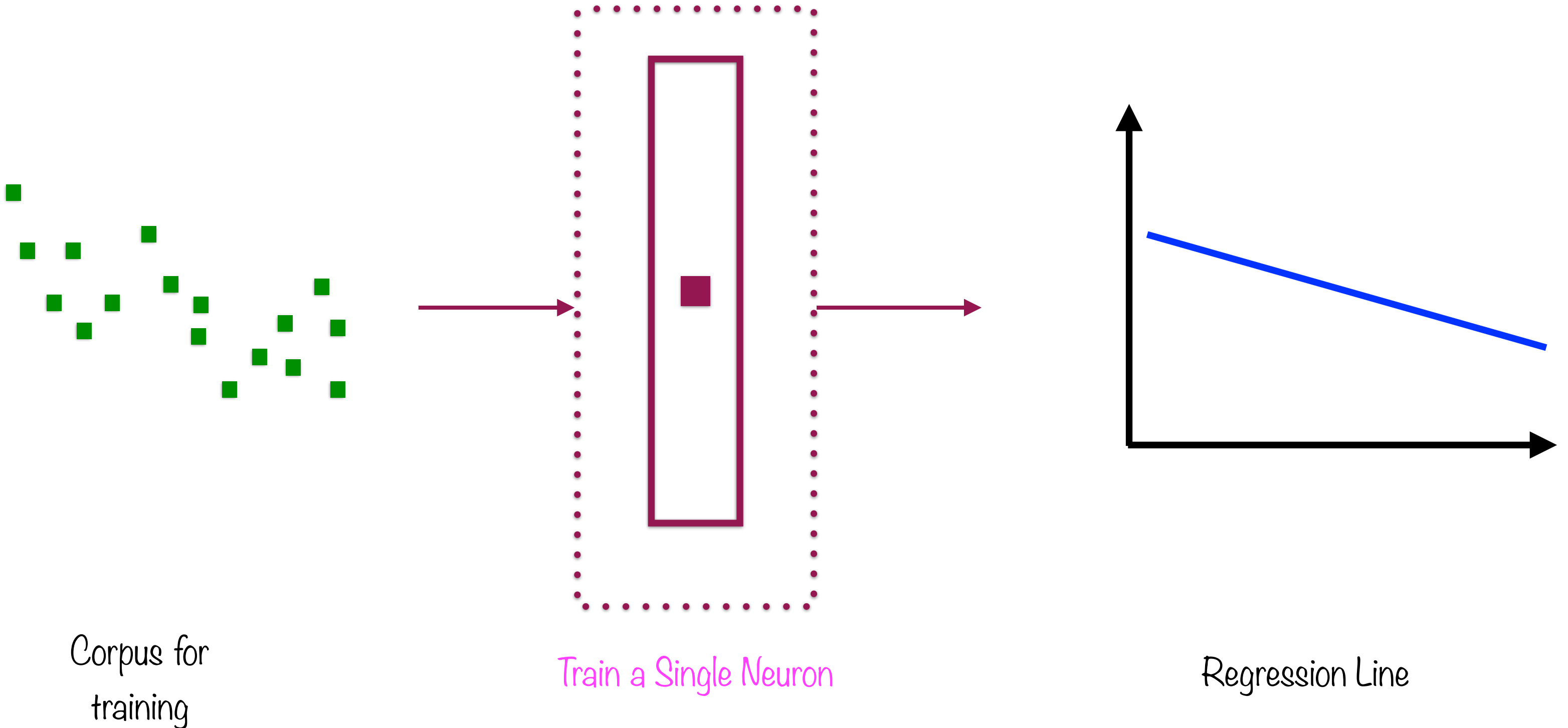
Linear Regression involves finding the "best fit" line via a training process

$$y = Wx + b$$

“Learning” Regression

Regression can be learnt by a single neuron using an affine transformation alone

Regression: The Simplest Neural Network



```
def doSomethingReallyComplicated(x1,x2...):
```

```
...
```

```
...
```

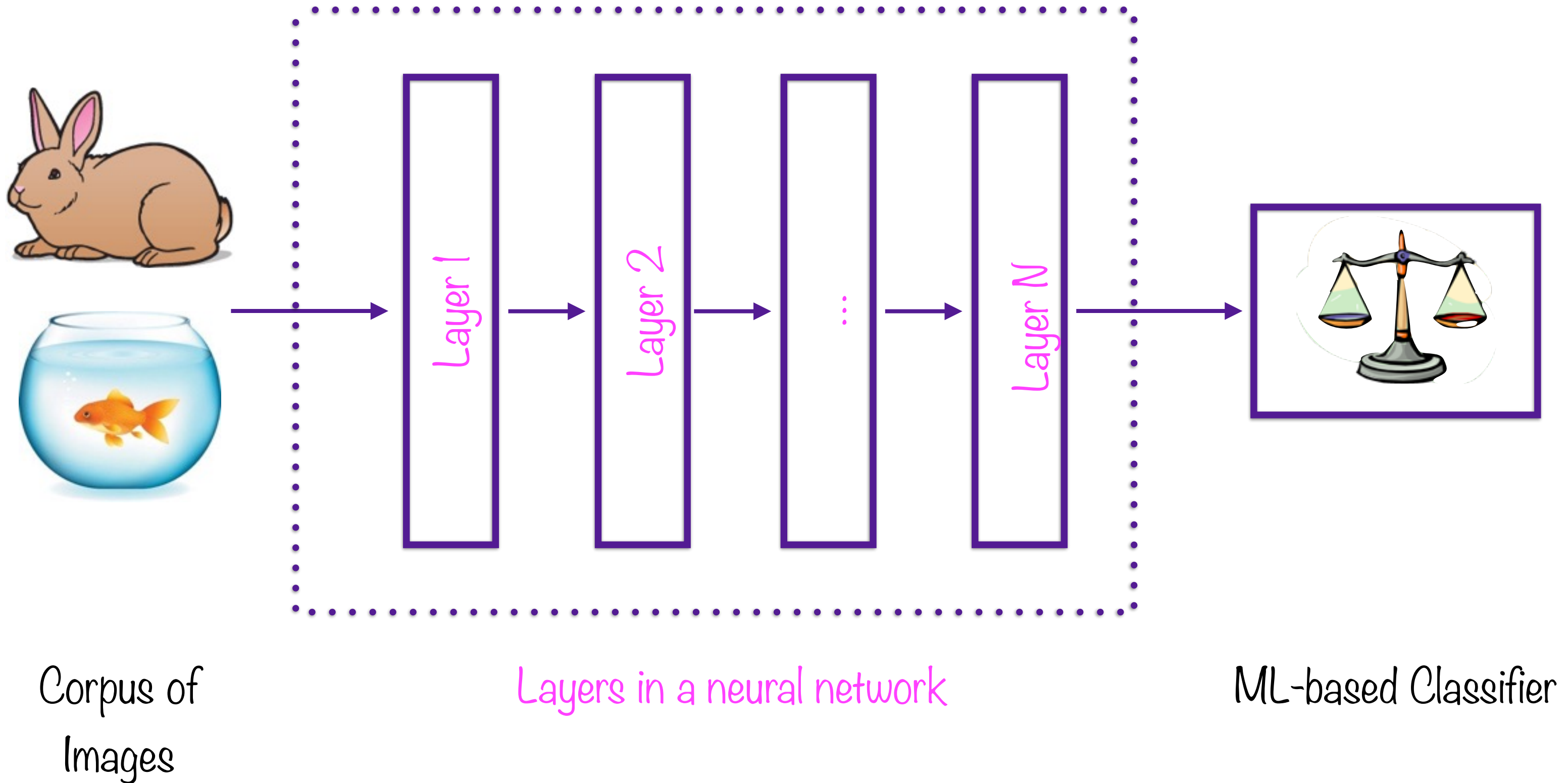
```
...
```

```
    return complicatedResult
```

$f(x) = \text{doSomethingReallyComplicated}(x)$

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

Feed-Forward Networks for Complex Functions



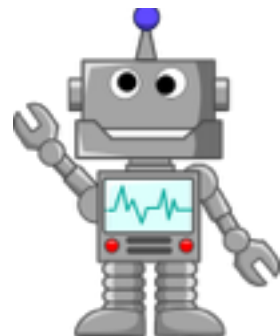
**Sometimes time relationships in data have
special meaning**

Why is The Past Important?



Assess stock prices over the last week, month and year

Will stocks be up or down in the coming week?



Autocomplete sentences

“The 2012 Olympics were held in ...”



A car is moving in a parabolic arc for the last 15 seconds

Is it headed for a collision?

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

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

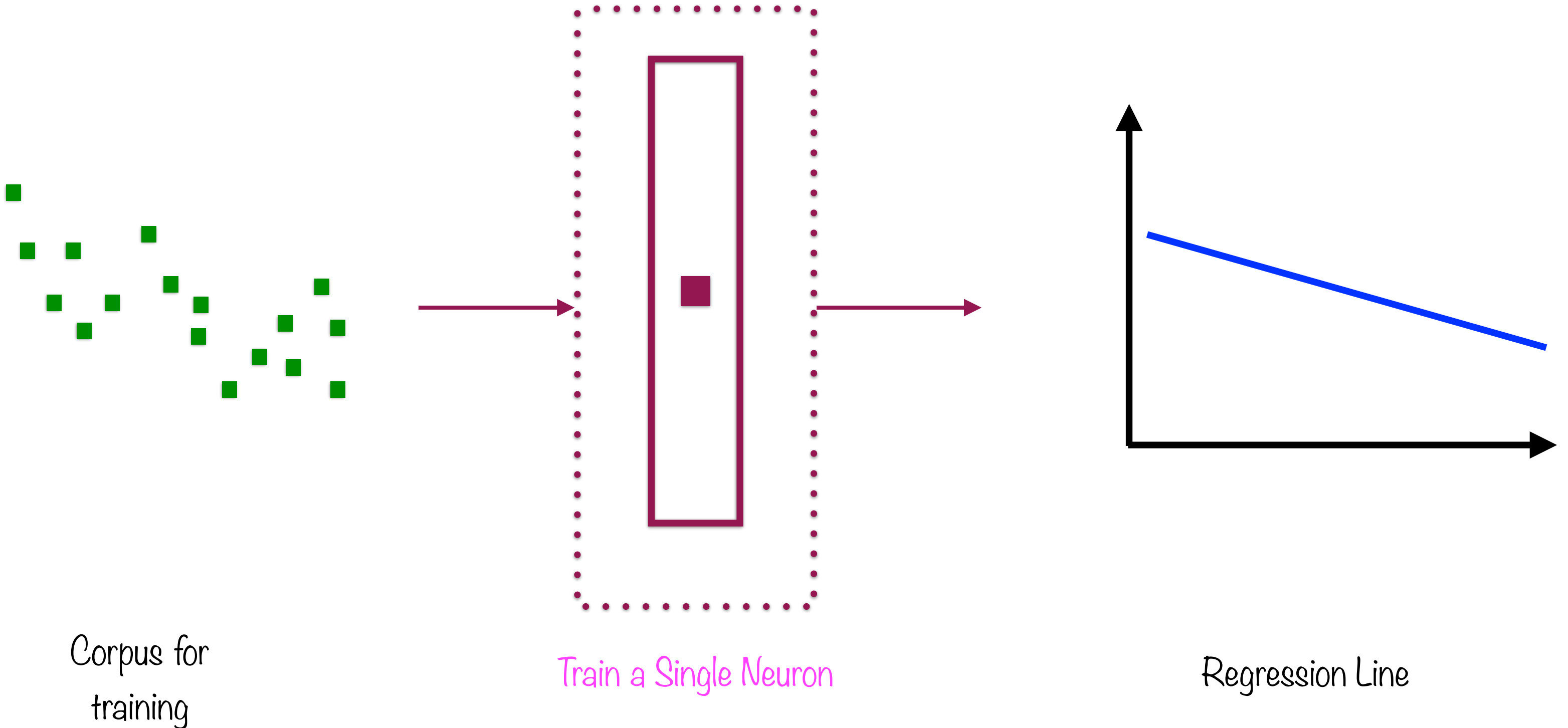
Relationships where past values of the effect variable drive current values are called auto-regressive

Feed-forward networks cannot learn from the
past

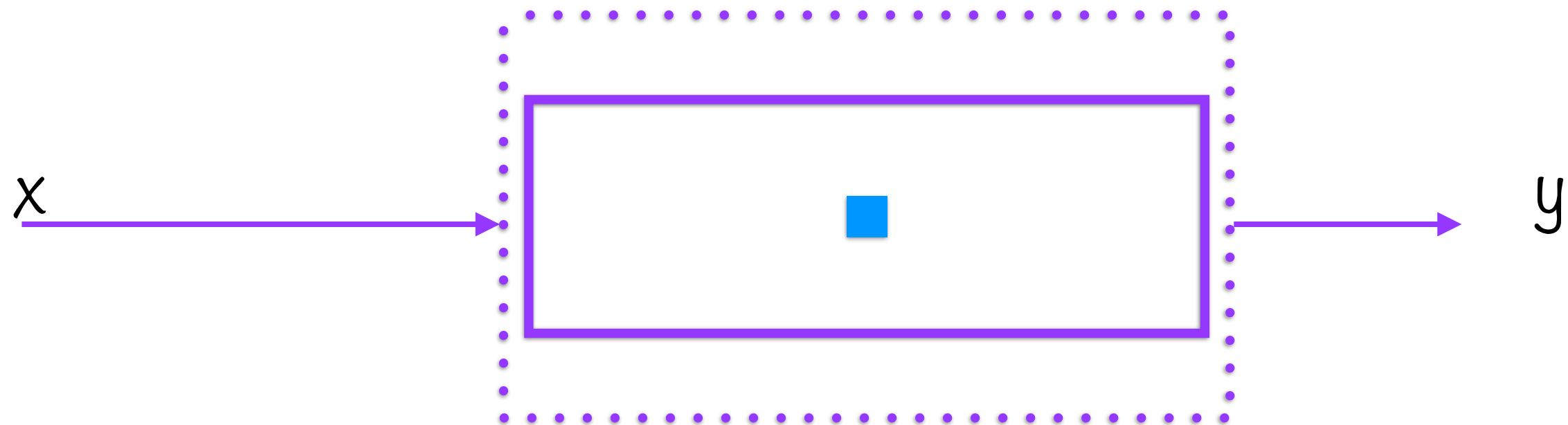
Recurrent neural networks can

Recurrent Neurons

Simplest Feed-forward Network

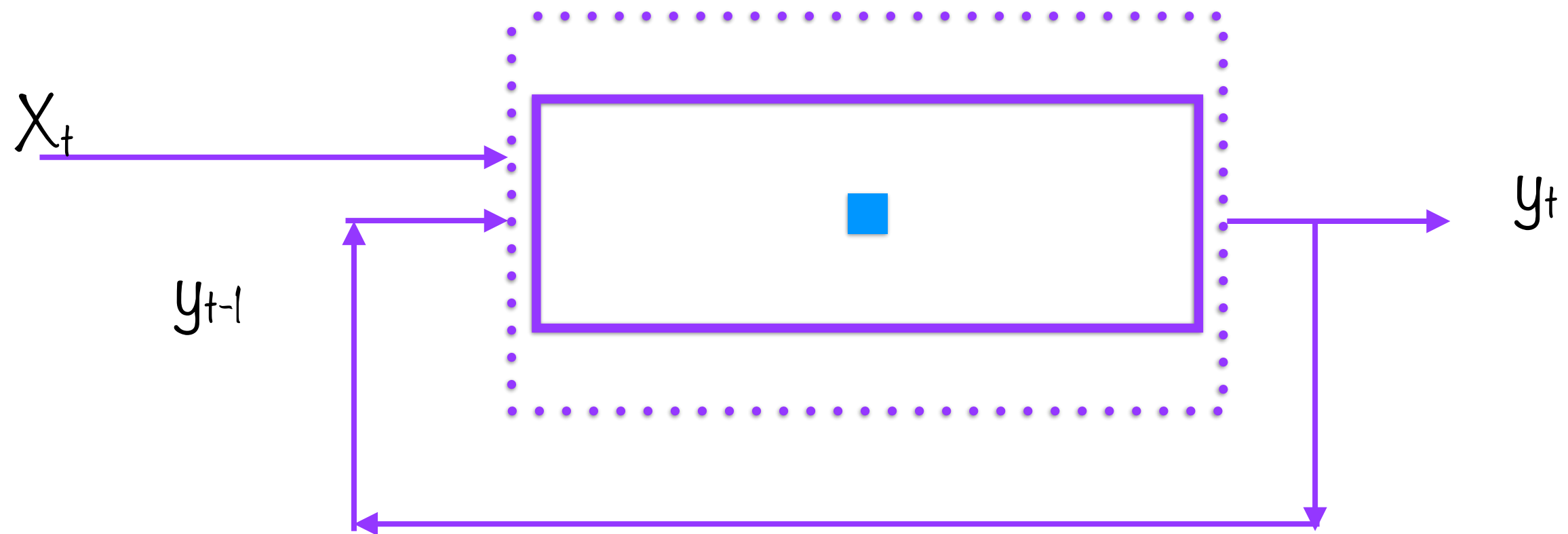


Simplest Feed-forward Neuron



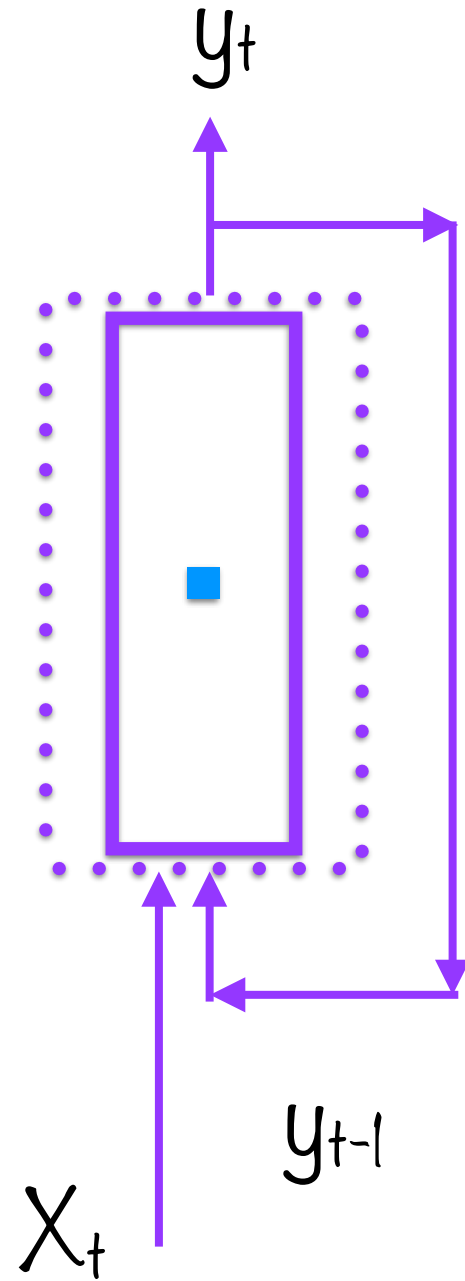
Neuron

Simplest Recurrent Neuron



Recurrent Neuron

Recurrent Neuron

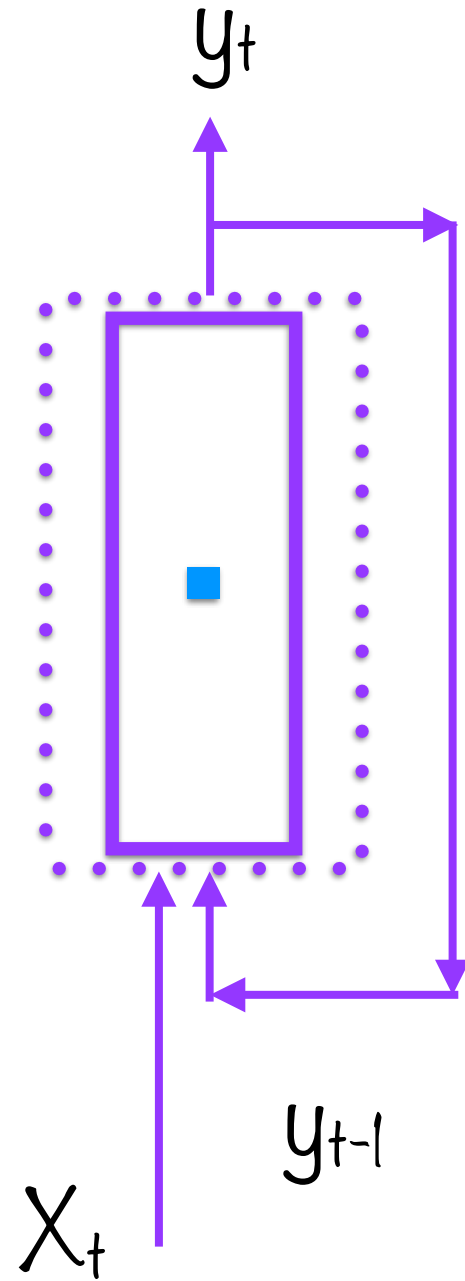


y_t = Output at time t

Depends upon

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

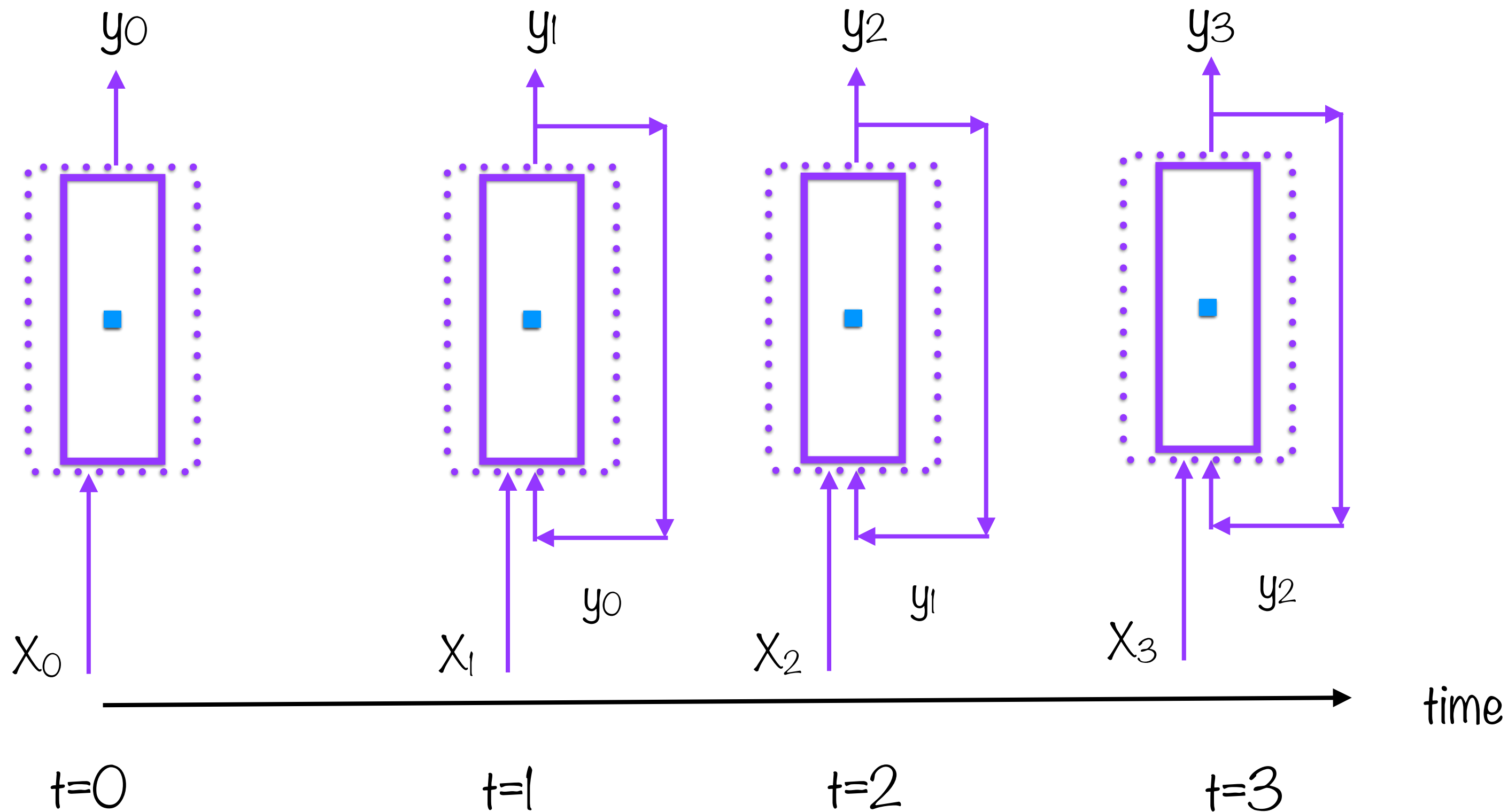
Recurrent Neuron



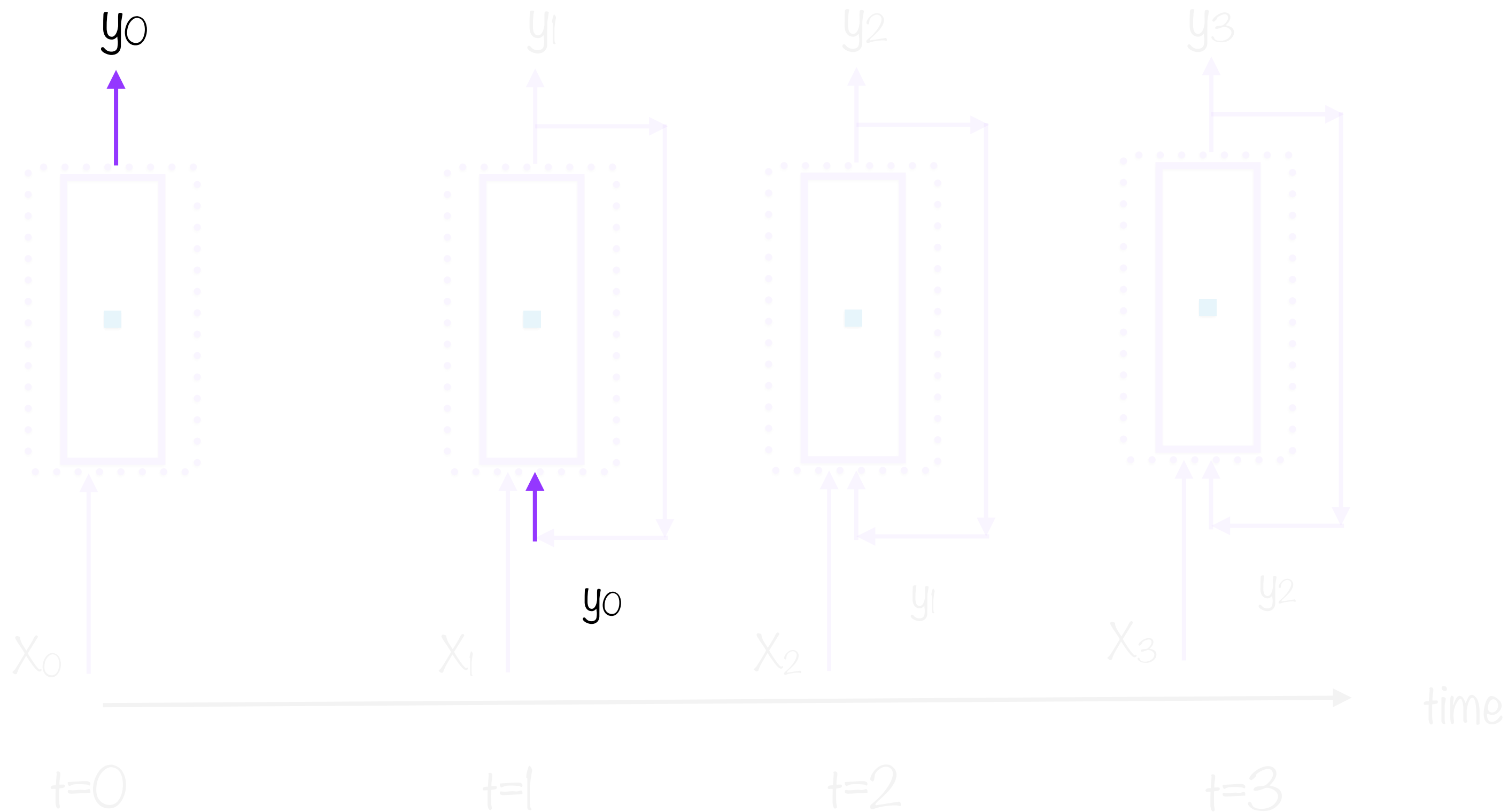
Substitute different values of t

Lay out the resulting neural network on a time axis

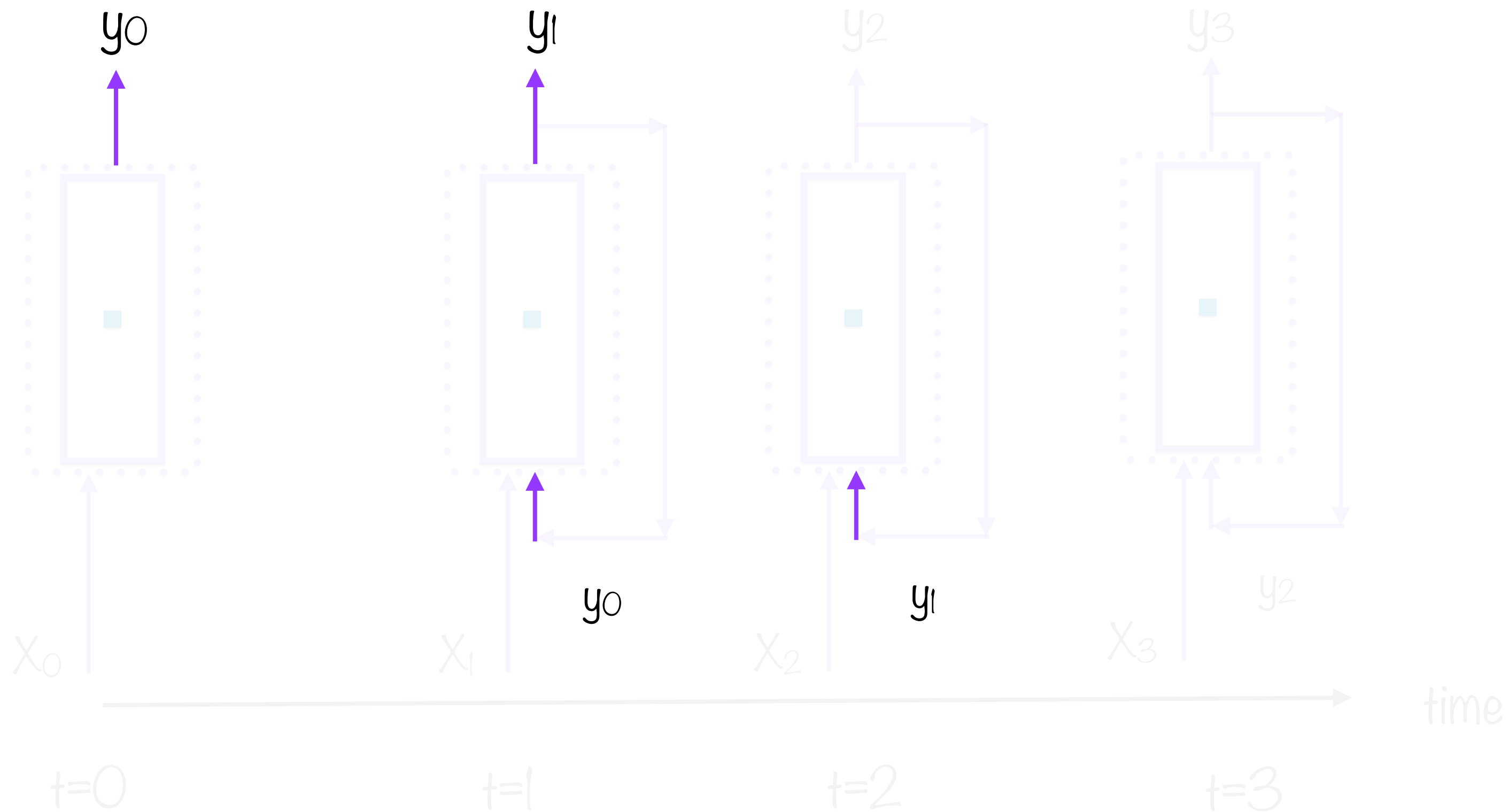
Unrolling Through Time



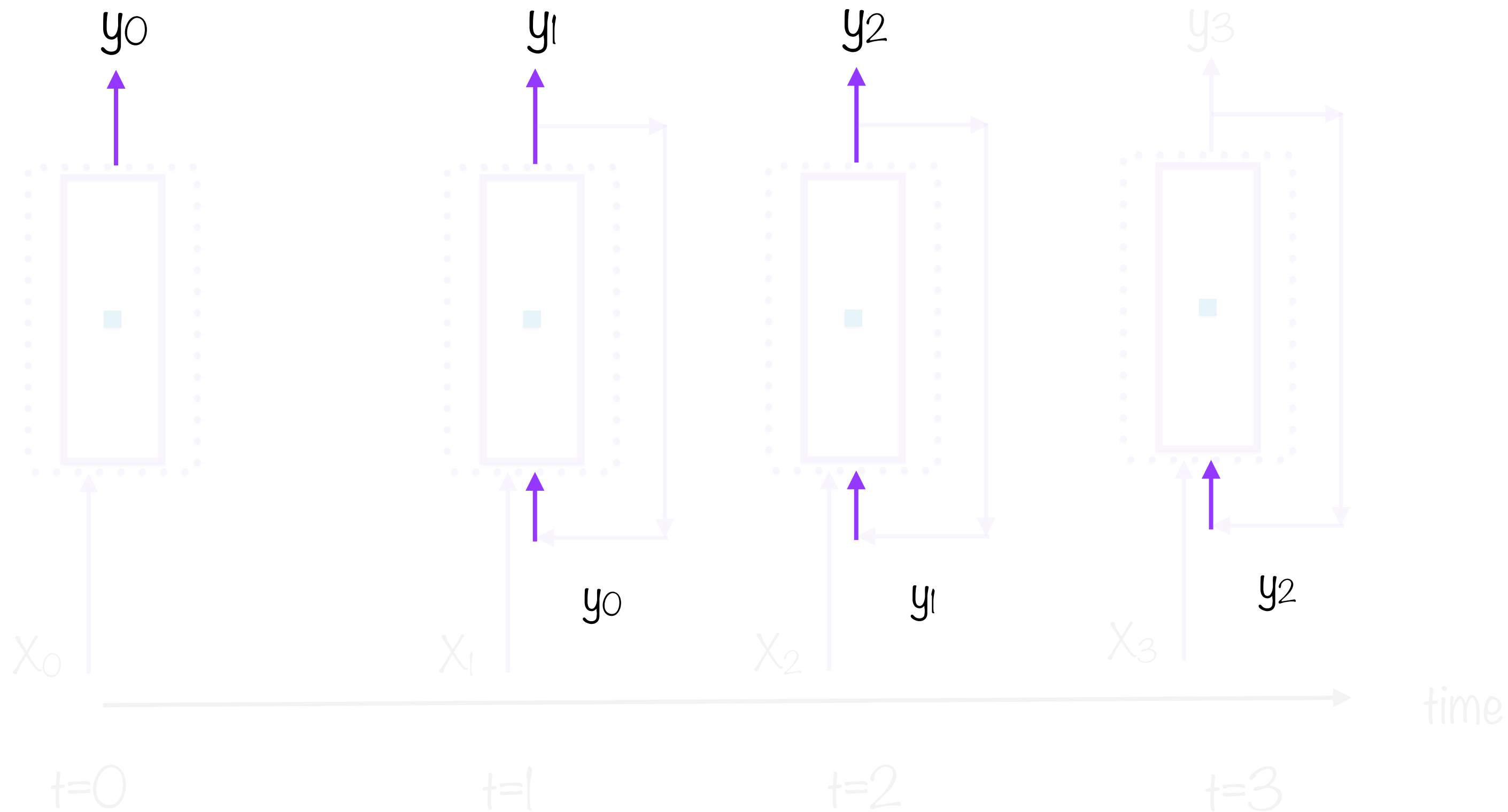
Unrolling Through Time



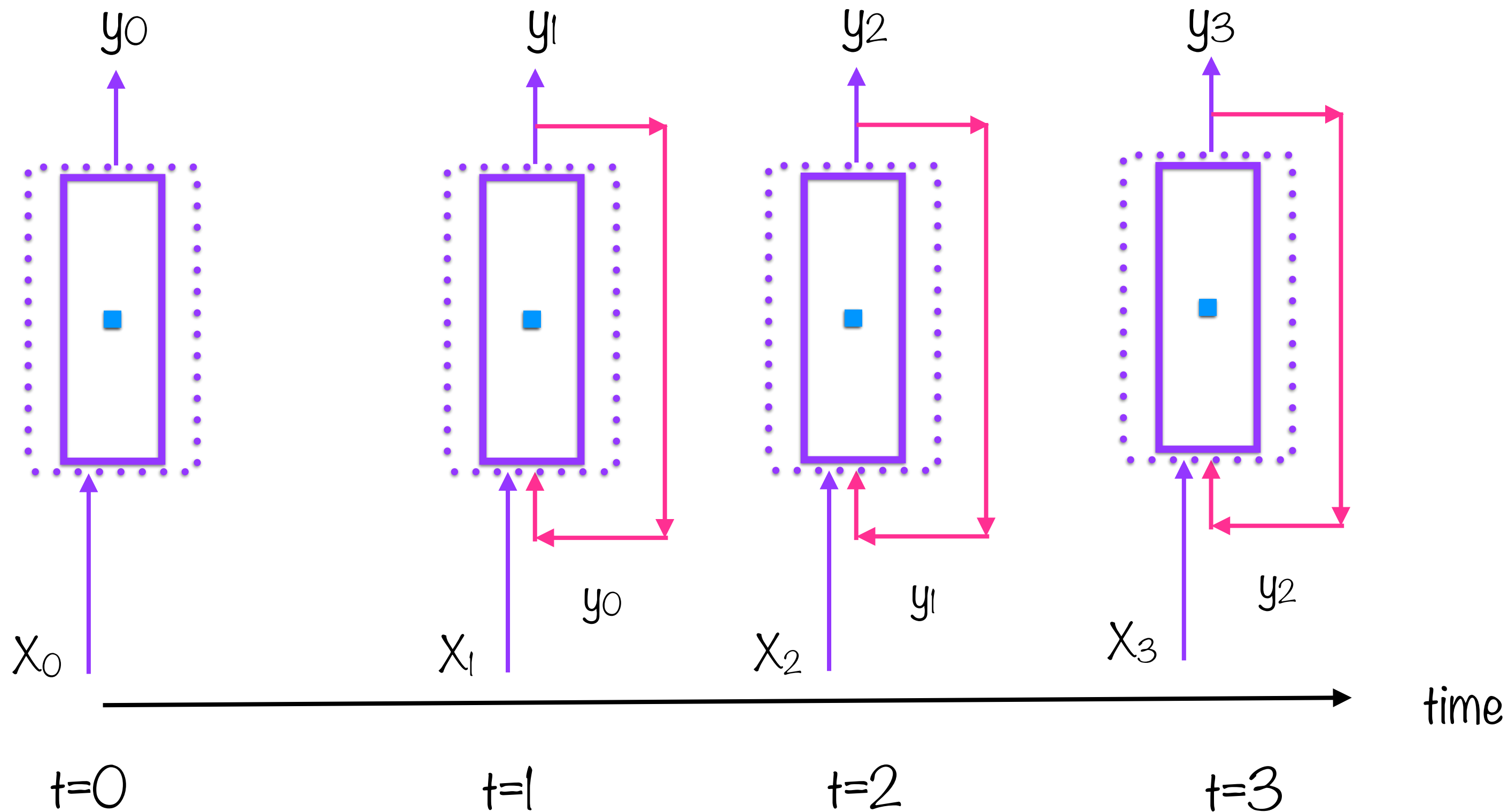
Unrolling Through Time



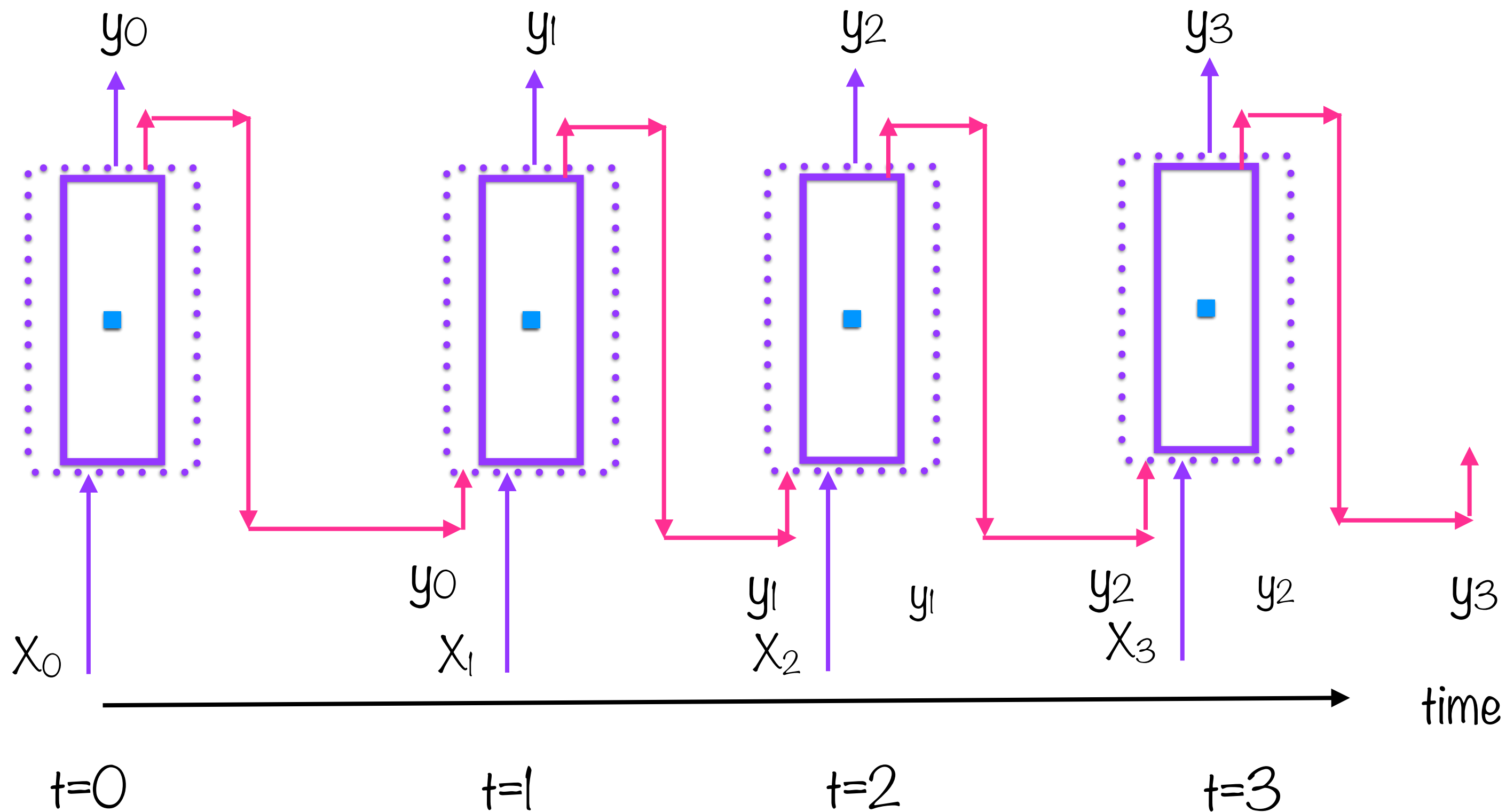
Unrolling Through Time



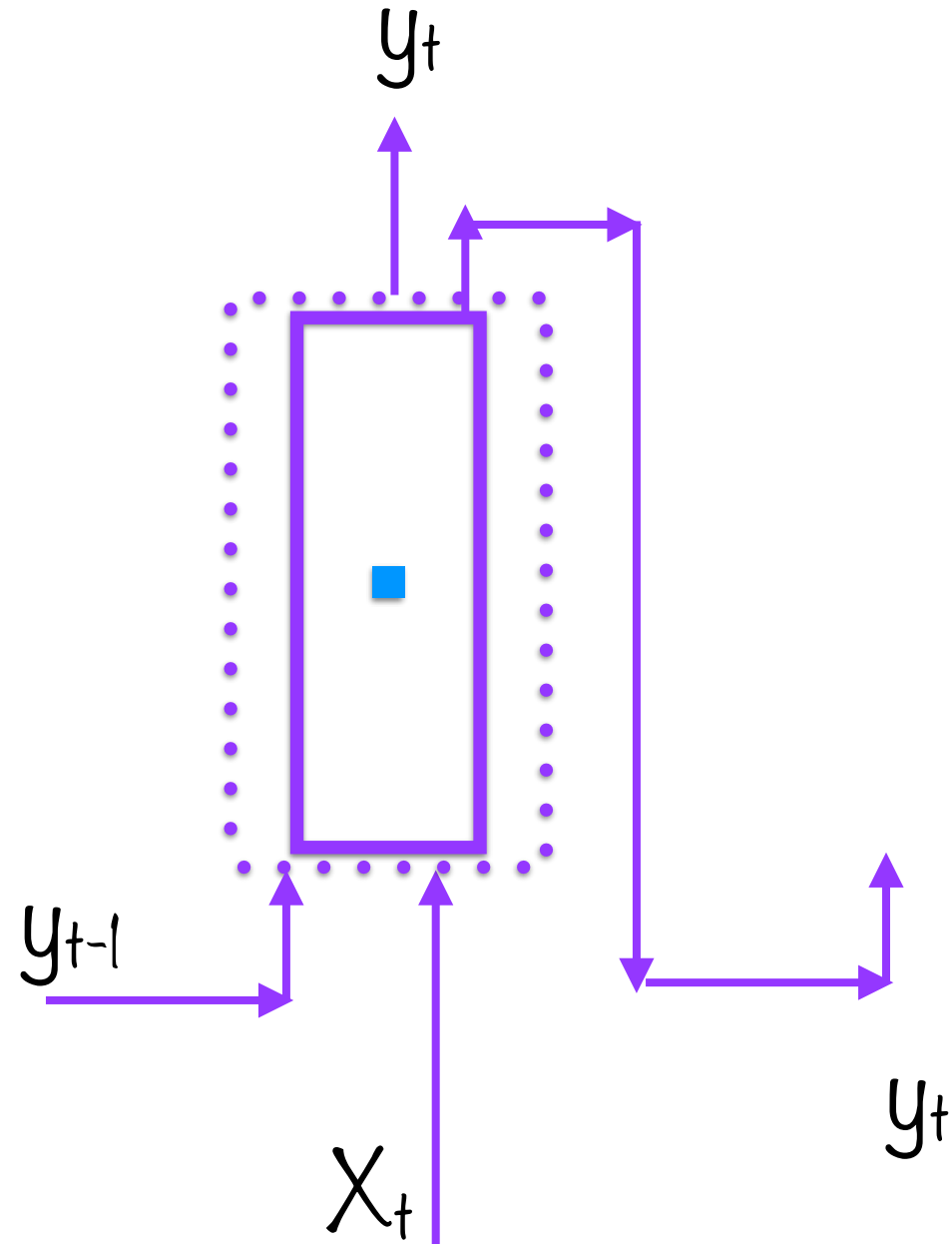
Unrolling Through Time



Unrolling Through Time



Recurrent Neuron



Regular neuron: input is feature vector, output is scalar

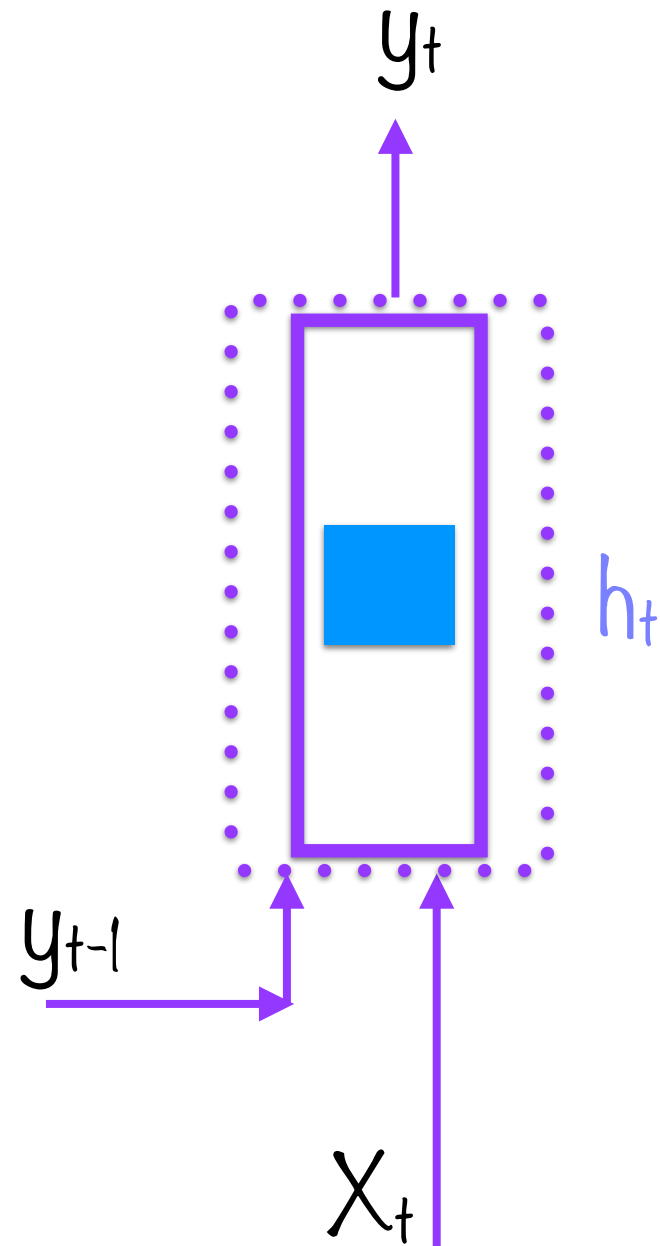
$$Y = Wx + b$$

Recurrent neuron: output is vector too

Input: $[X_0, X_1, \dots, X_t]$

Output: $[Y_0, Y_1, \dots, Y_t]$

Memory and State



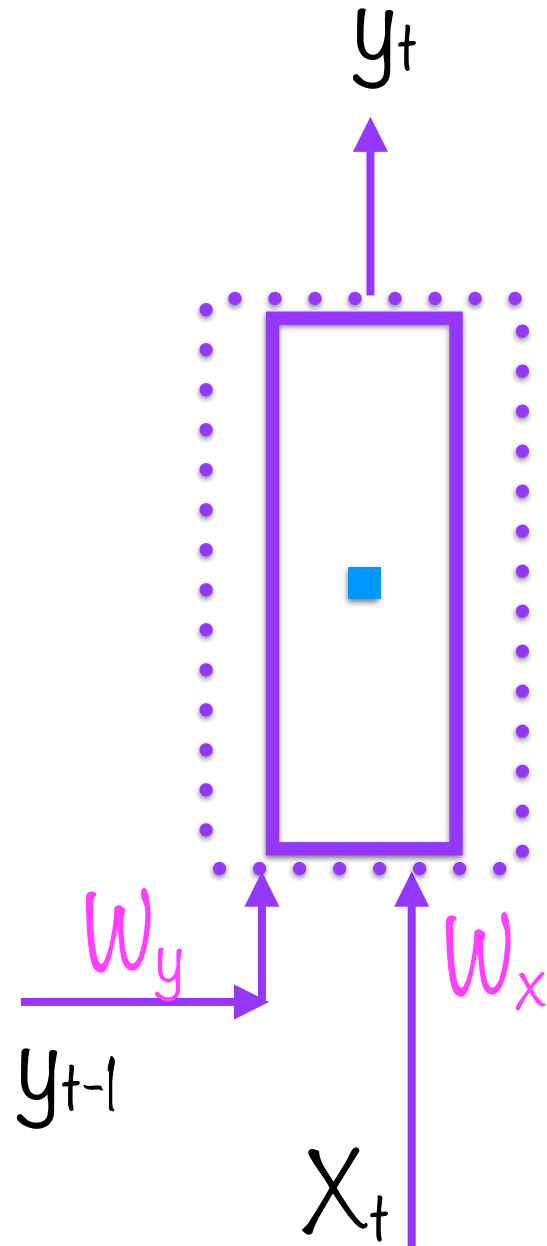
Recurrent neurons remember the past

They possess 'memory'

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

The internal state is represented by h_t

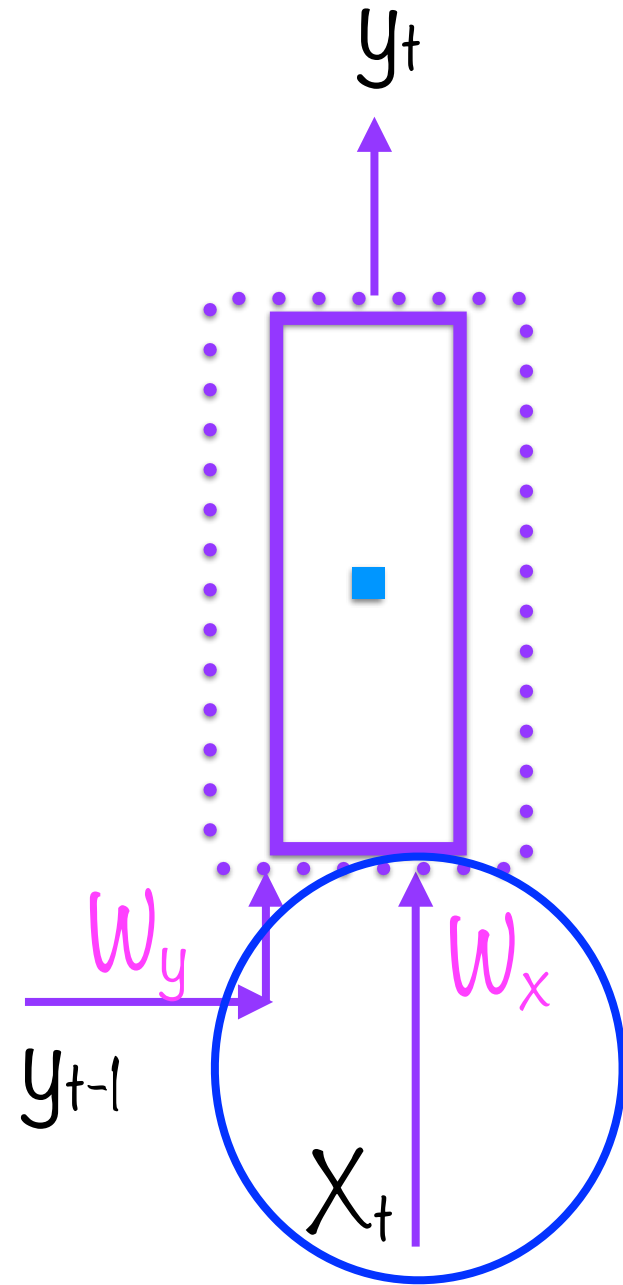
Recurrent Neuron



Now, each neuron has two weight vectors

w_x, w_y

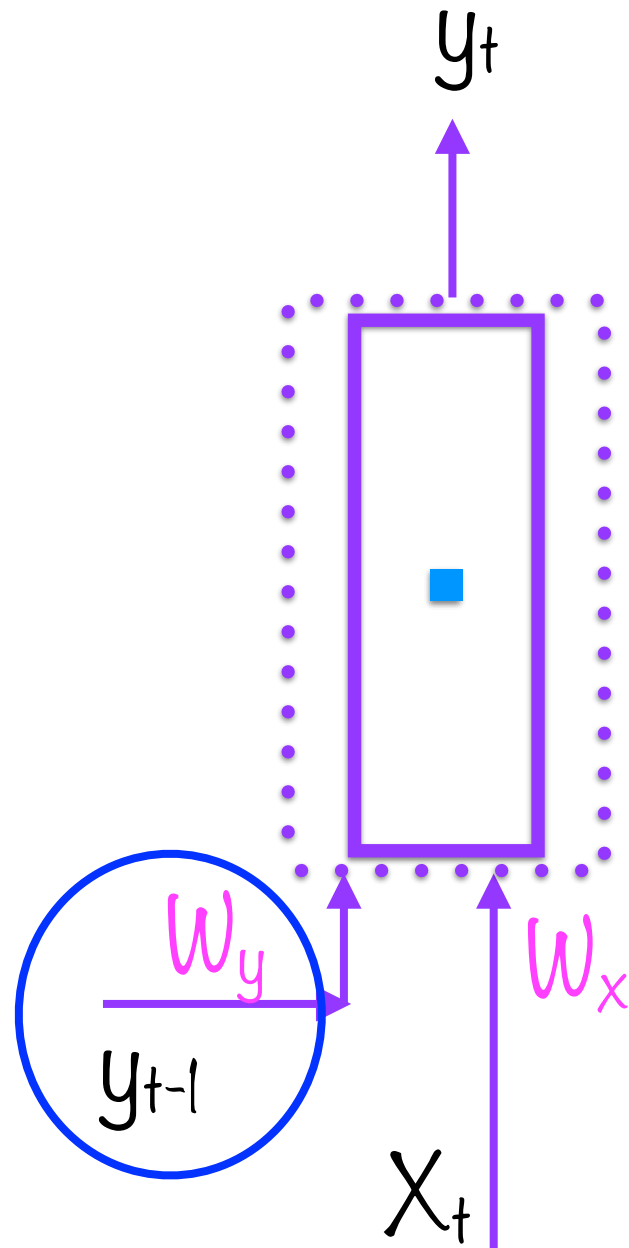
Recurrent Neuron



Now, each neuron has two weight vectors

w_x, w_y

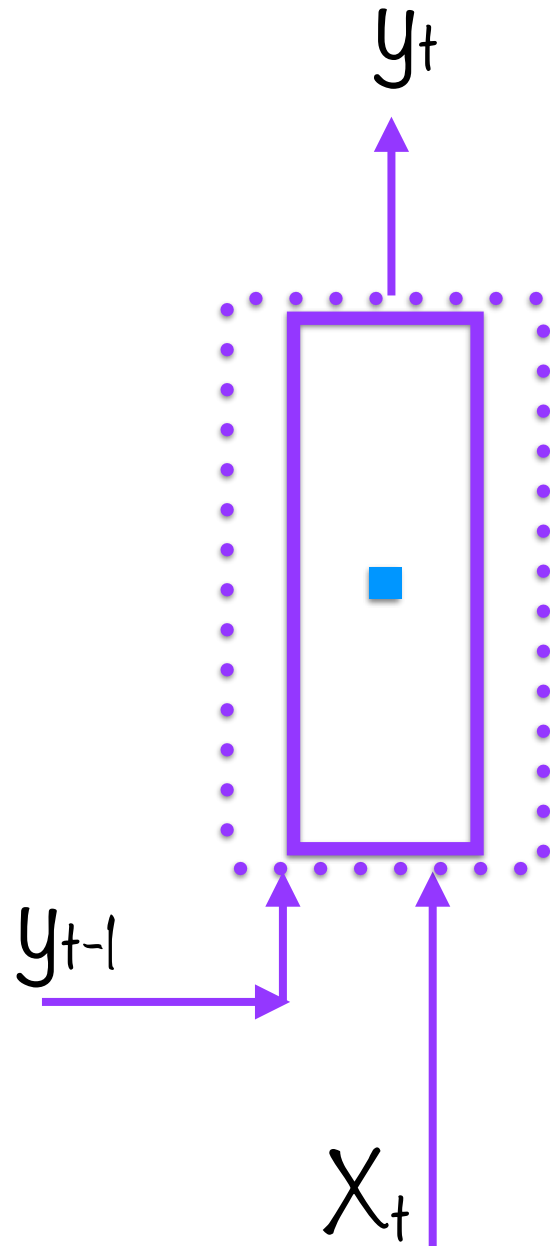
Recurrent Neuron



Now, each neuron has two weight vectors

w_x, w_y

Recurrent Neuron



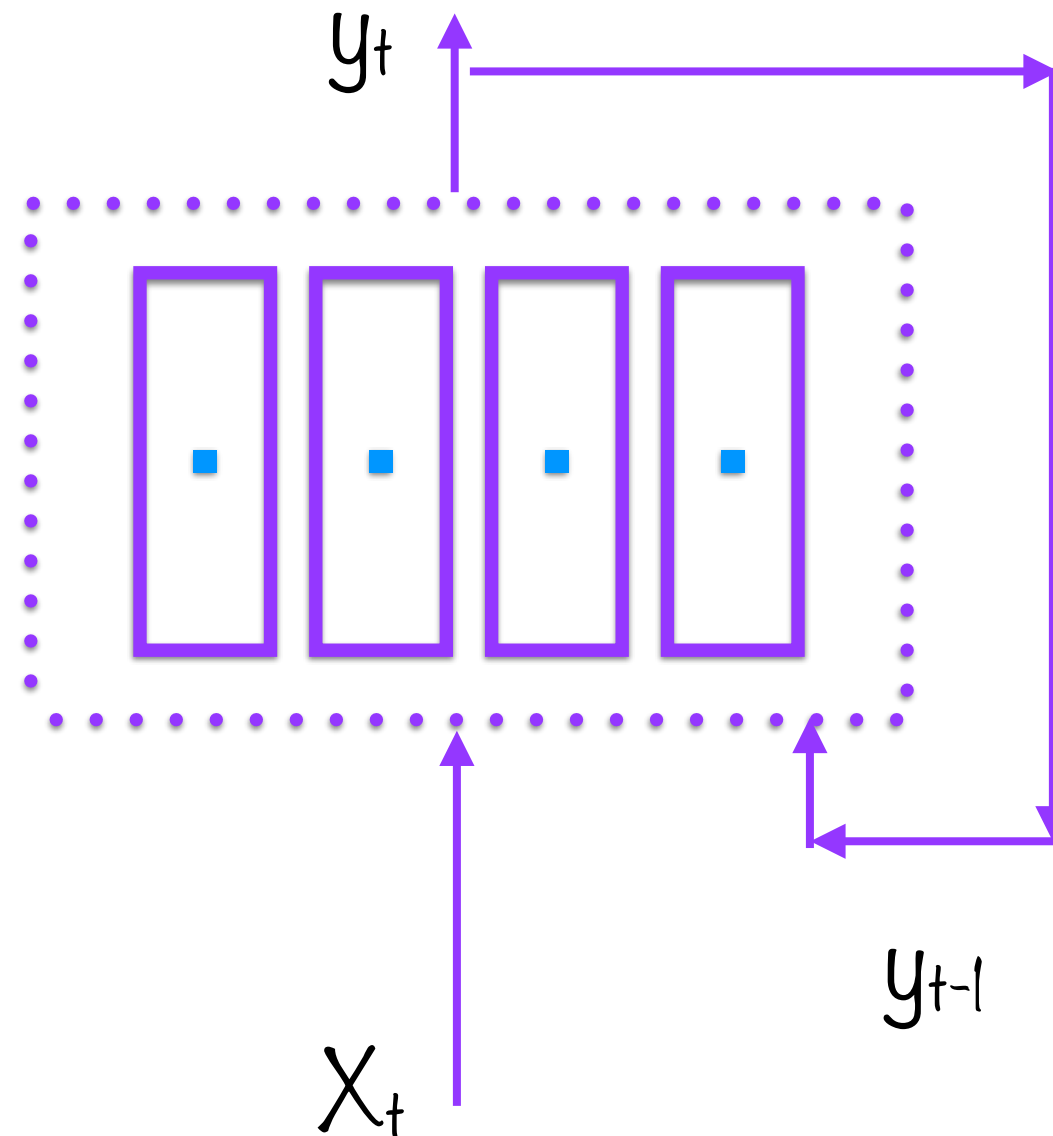
Output of neuron as a whole is given as

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

(Φ is the activation function)

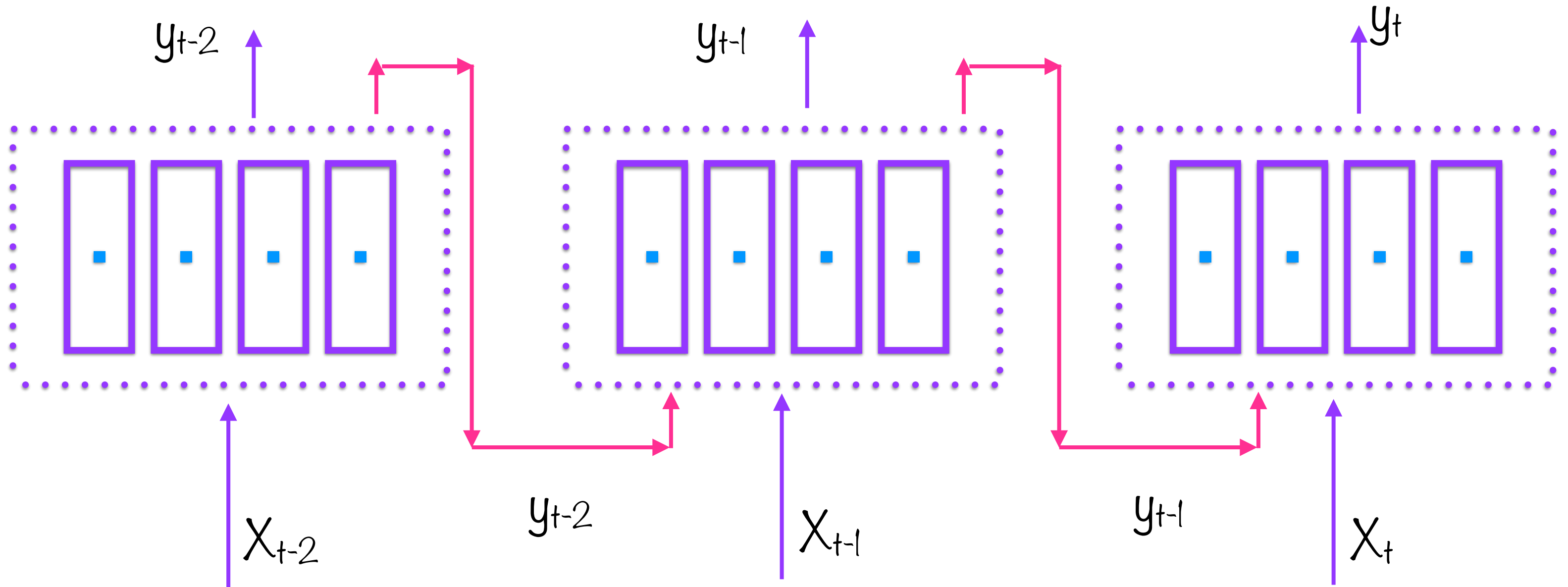
Training a Recurrent Neural Network

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 cell unrolled through time form the layers of the **neural network**

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

$$y = Wx + b$$

Linear Regression

Specifies, up-front, that the function f is linear

$$y = w x + b$$

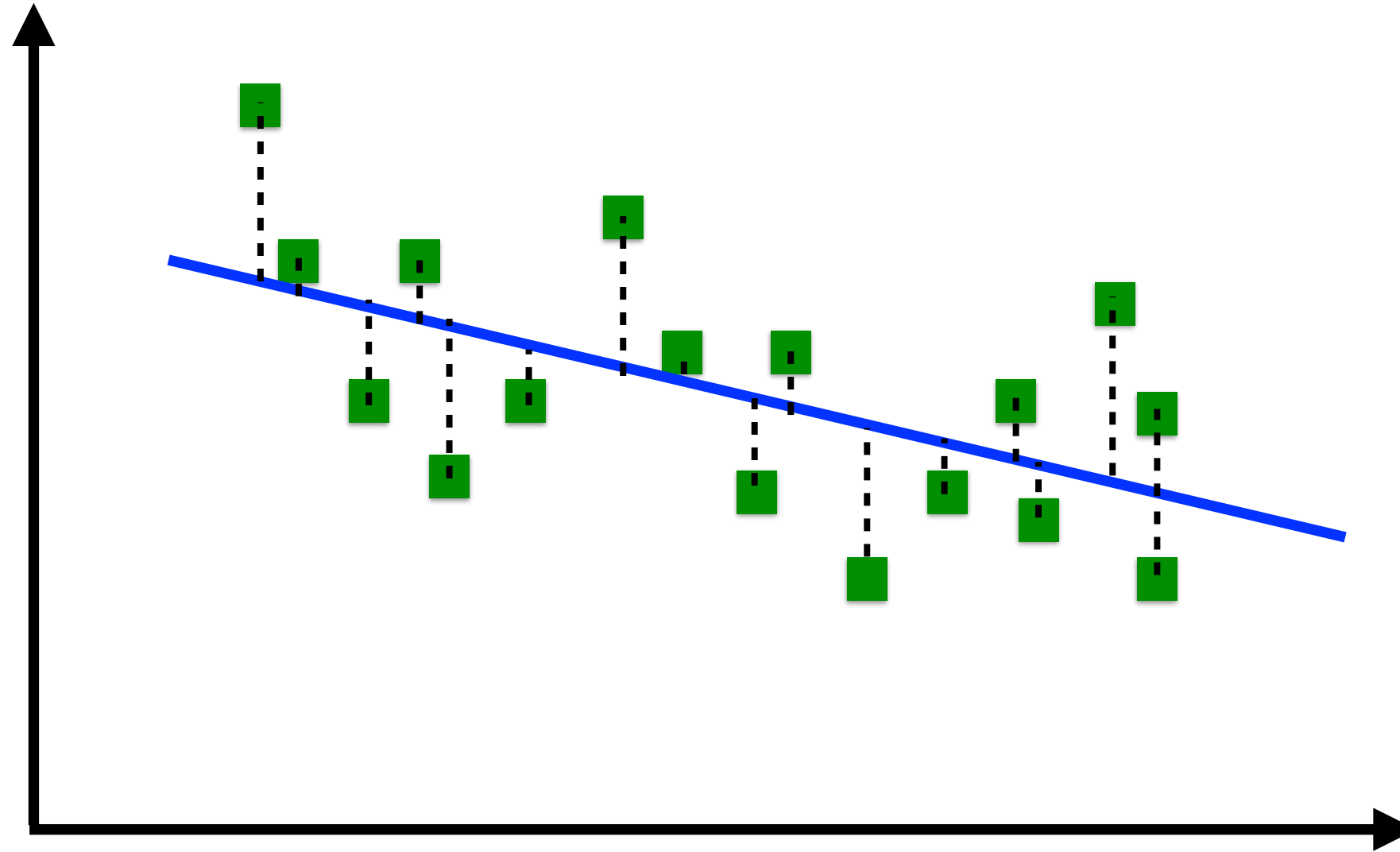
Linear Regression

Find the weights (w) and biases (b) for the best fit regression line

The "Best" Regression Line



y

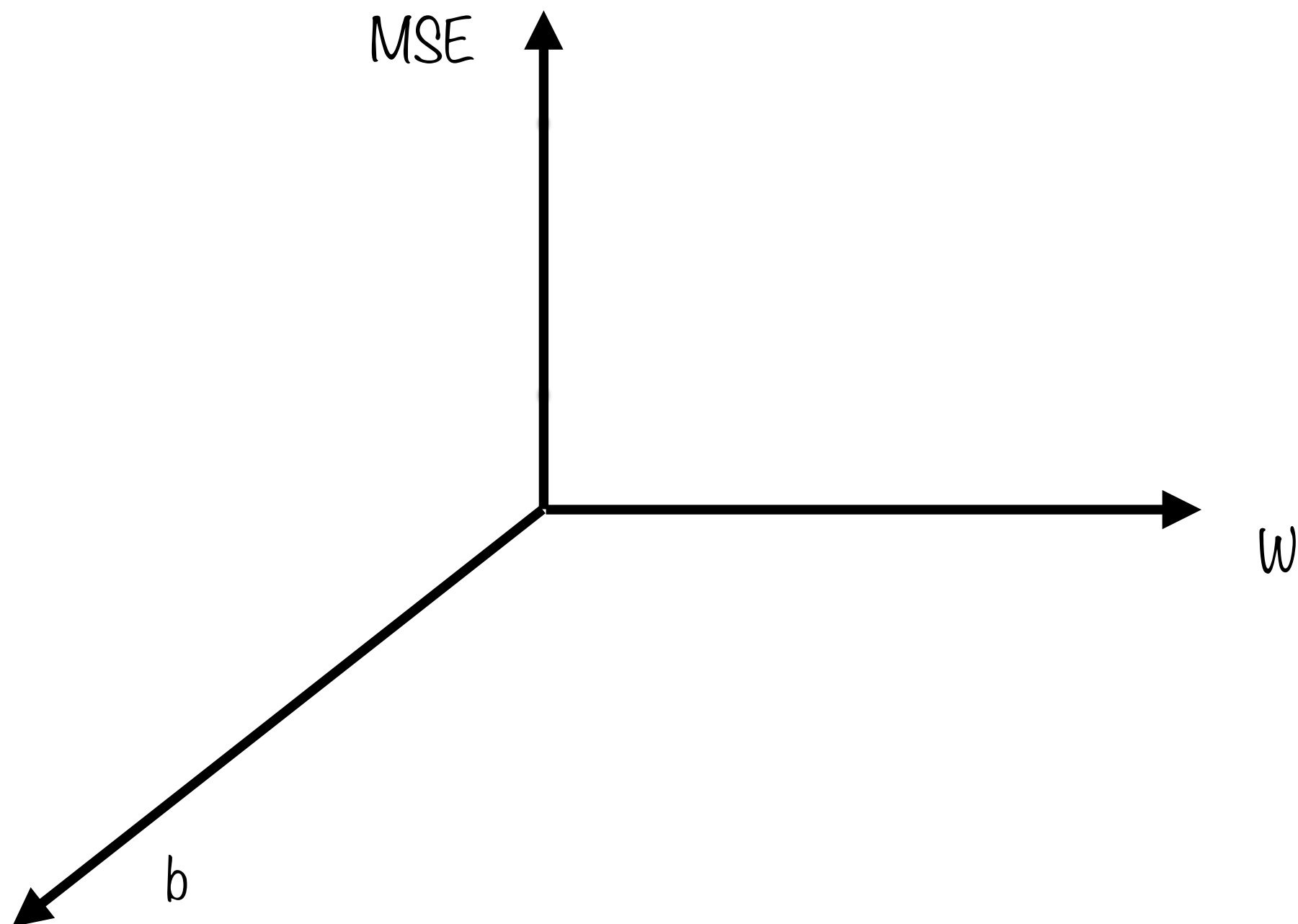


x

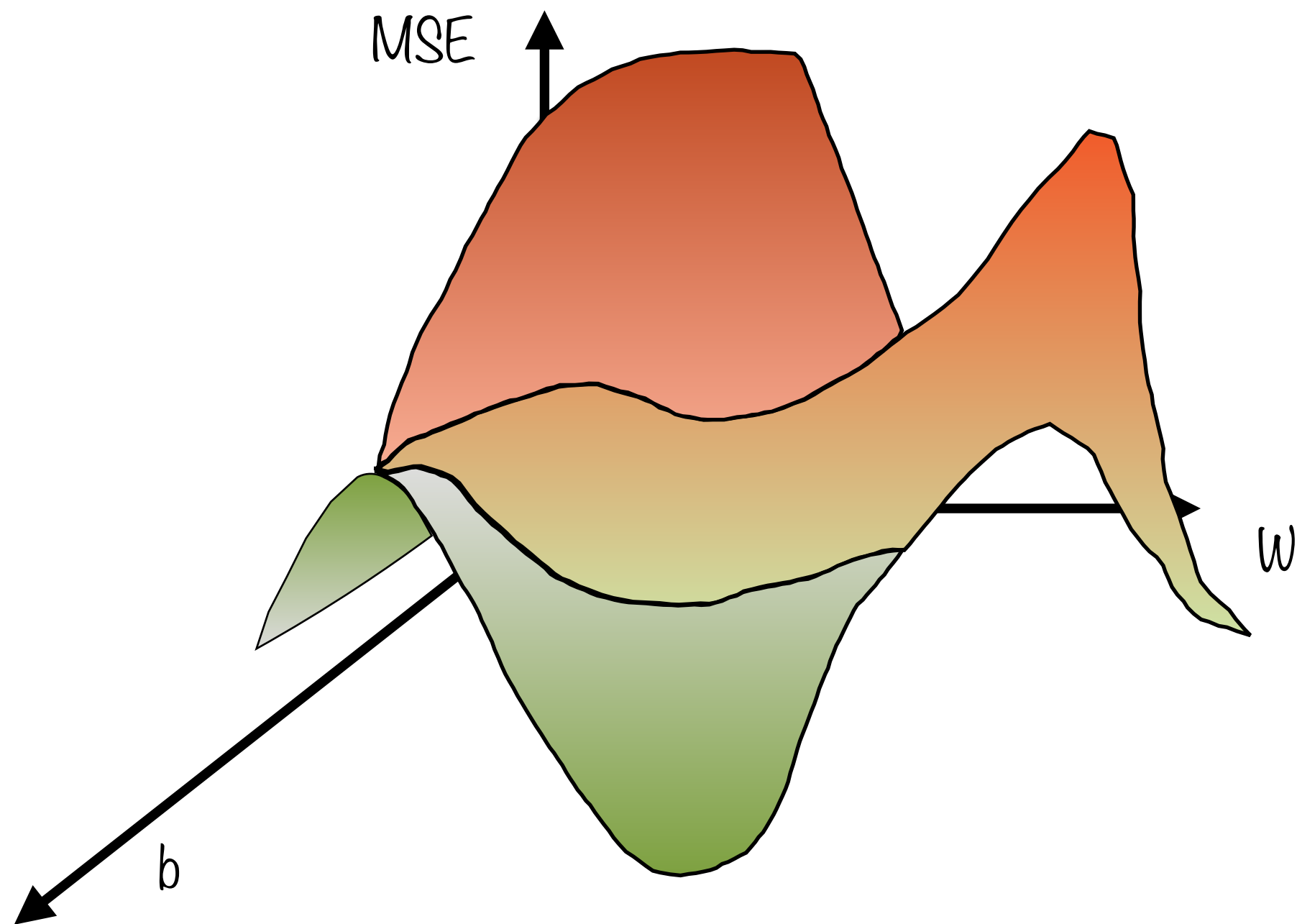


The "best fit" line minimizes the **Mean Square Error** of the points from the line

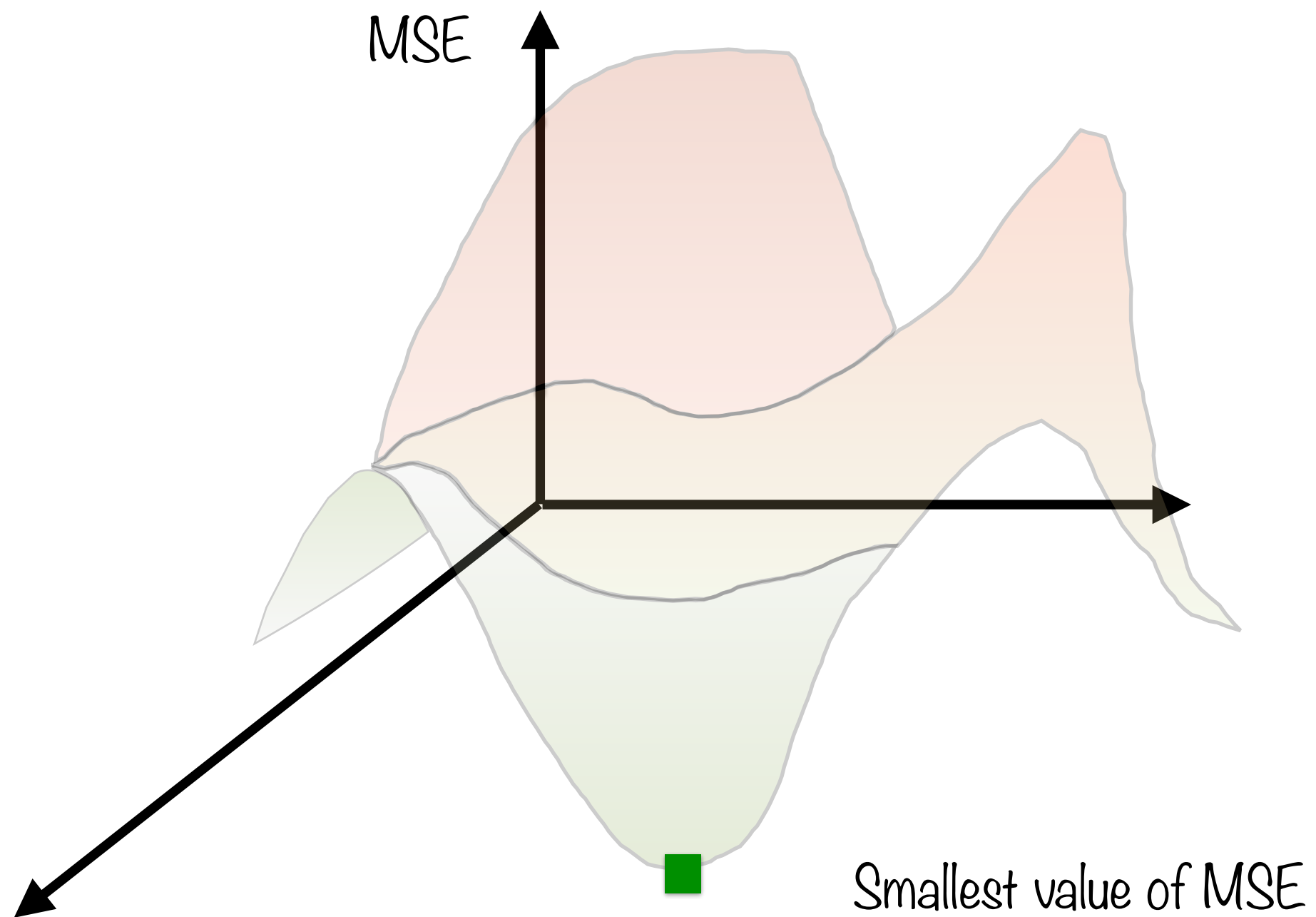
Minimizing MSE



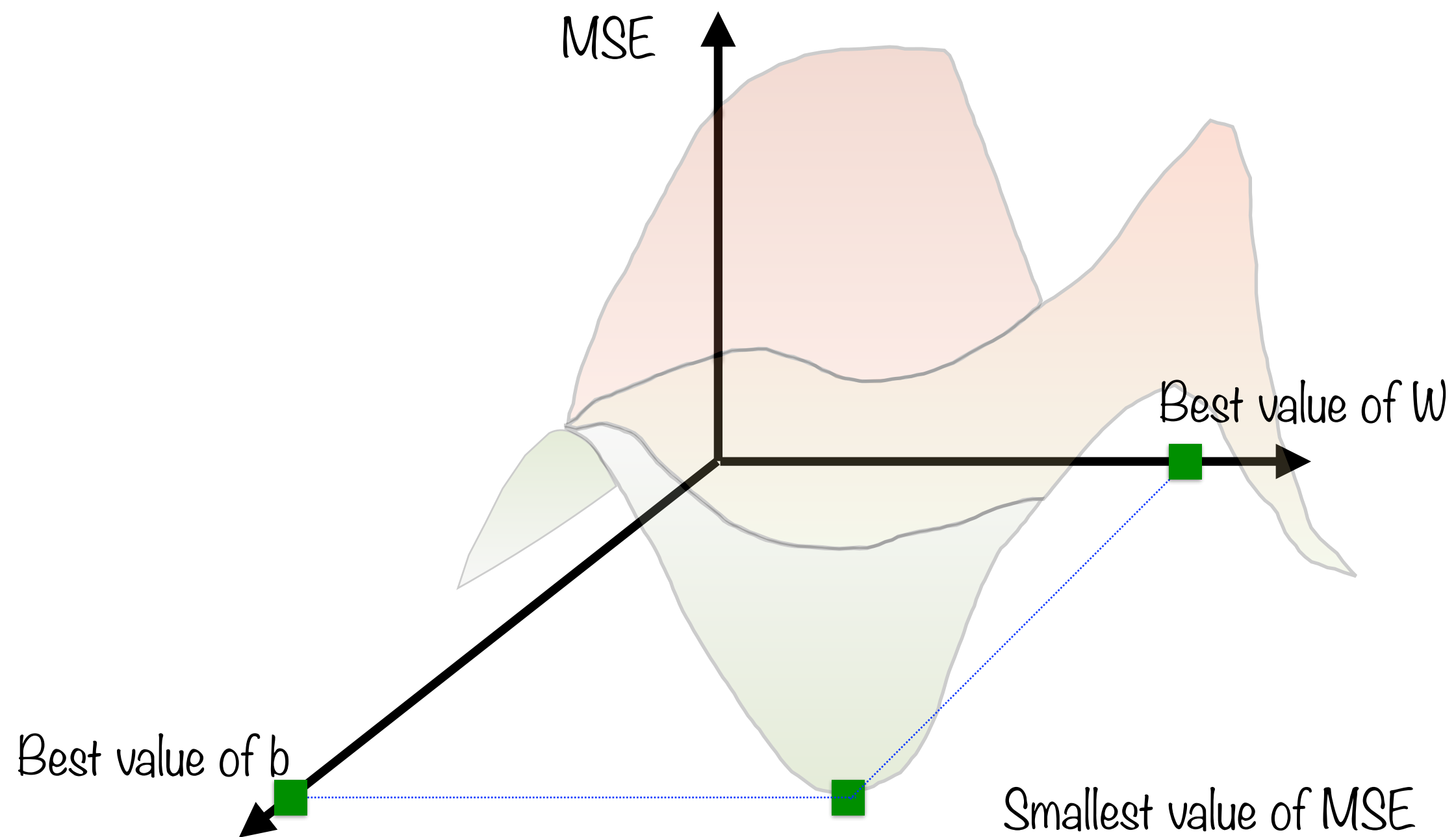
Minimizing MSE



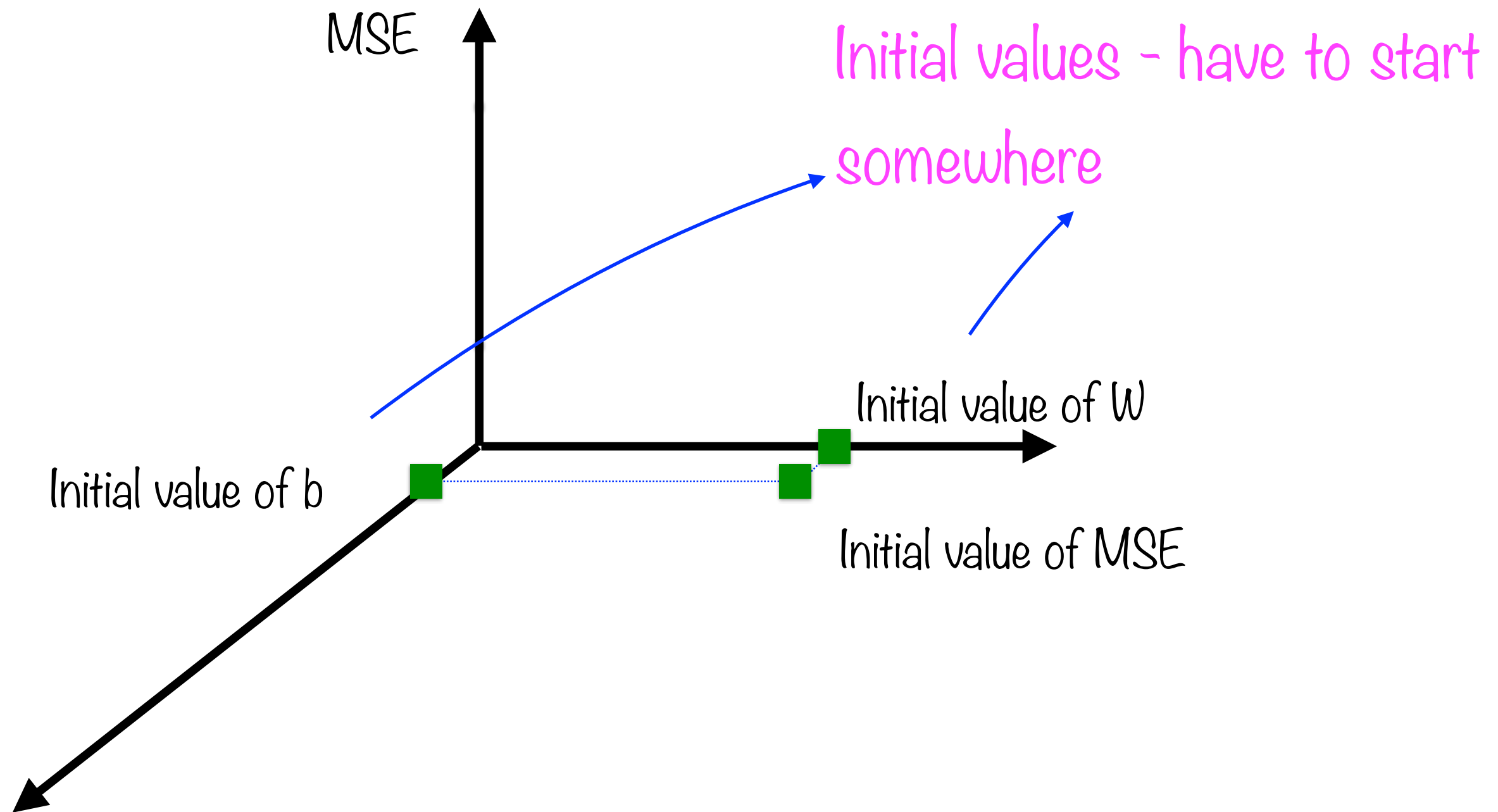
Minimizing MSE



Minimizing MSE

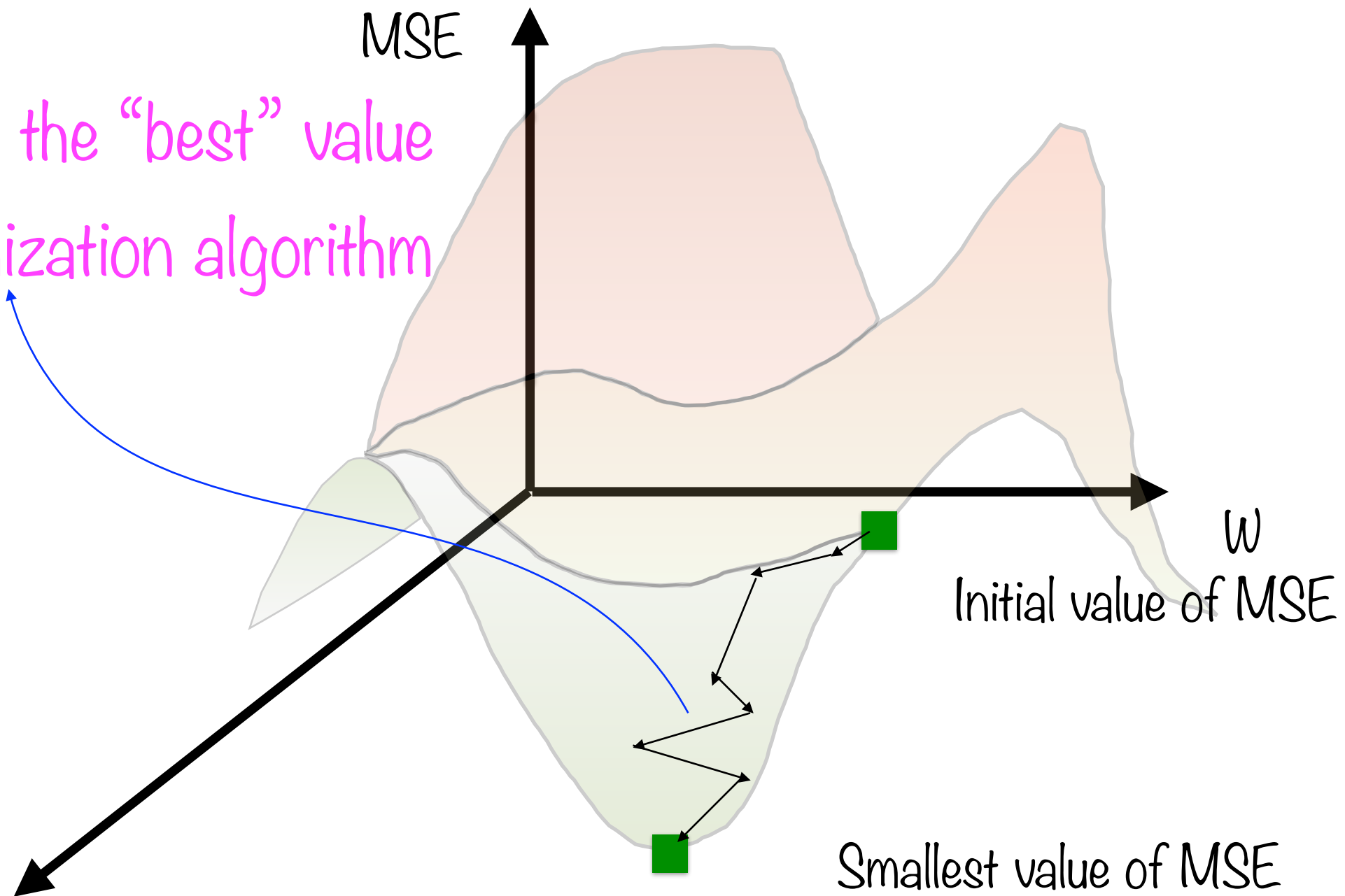


Start Somewhere

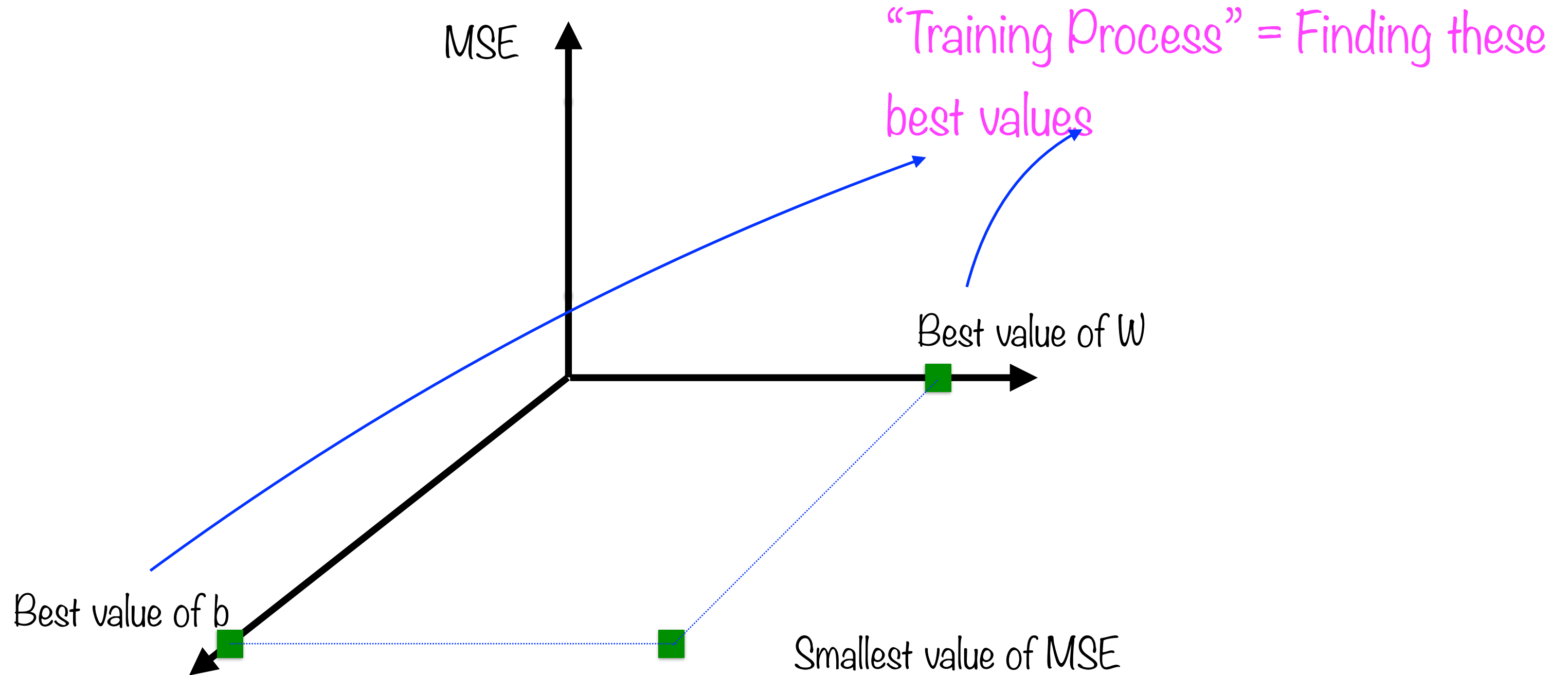


"Gradient Descent"

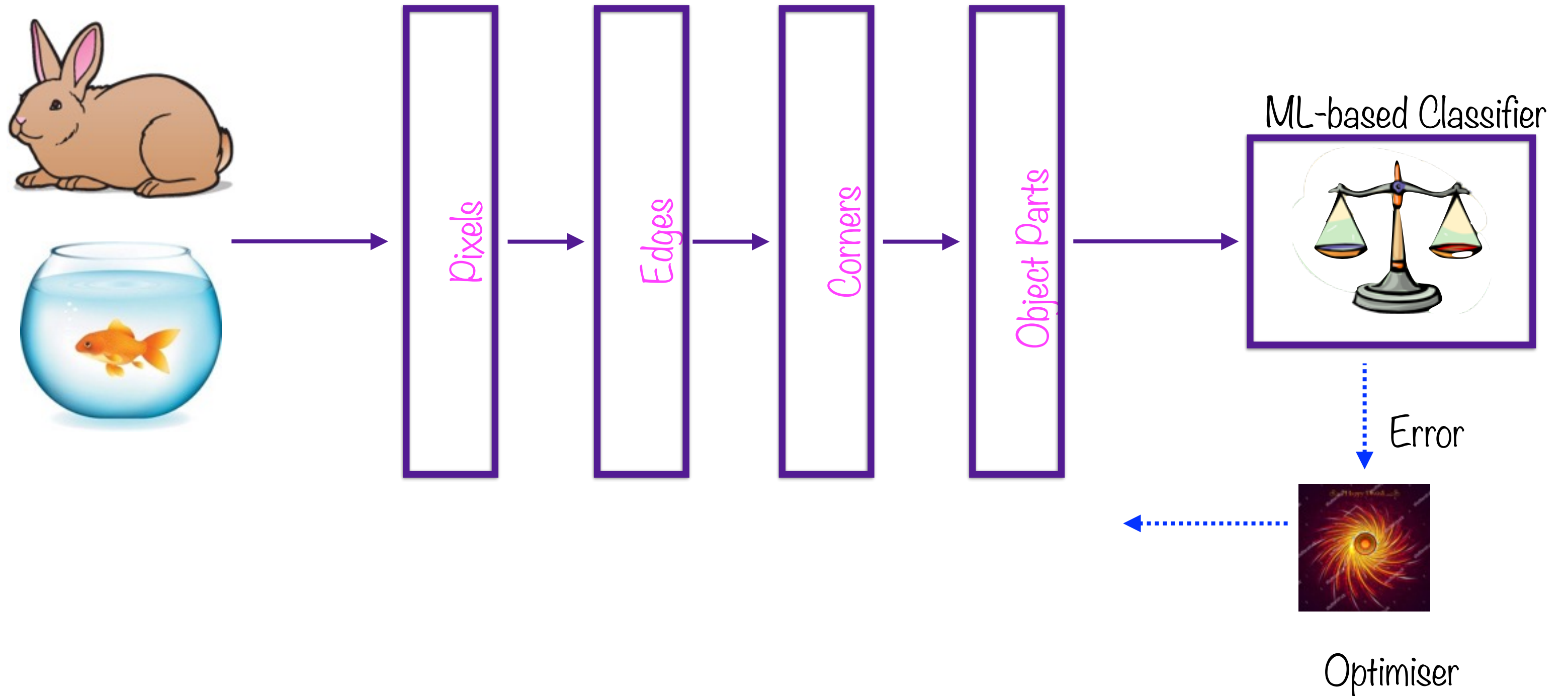
Converging on the "best" value
using an optimization algorithm



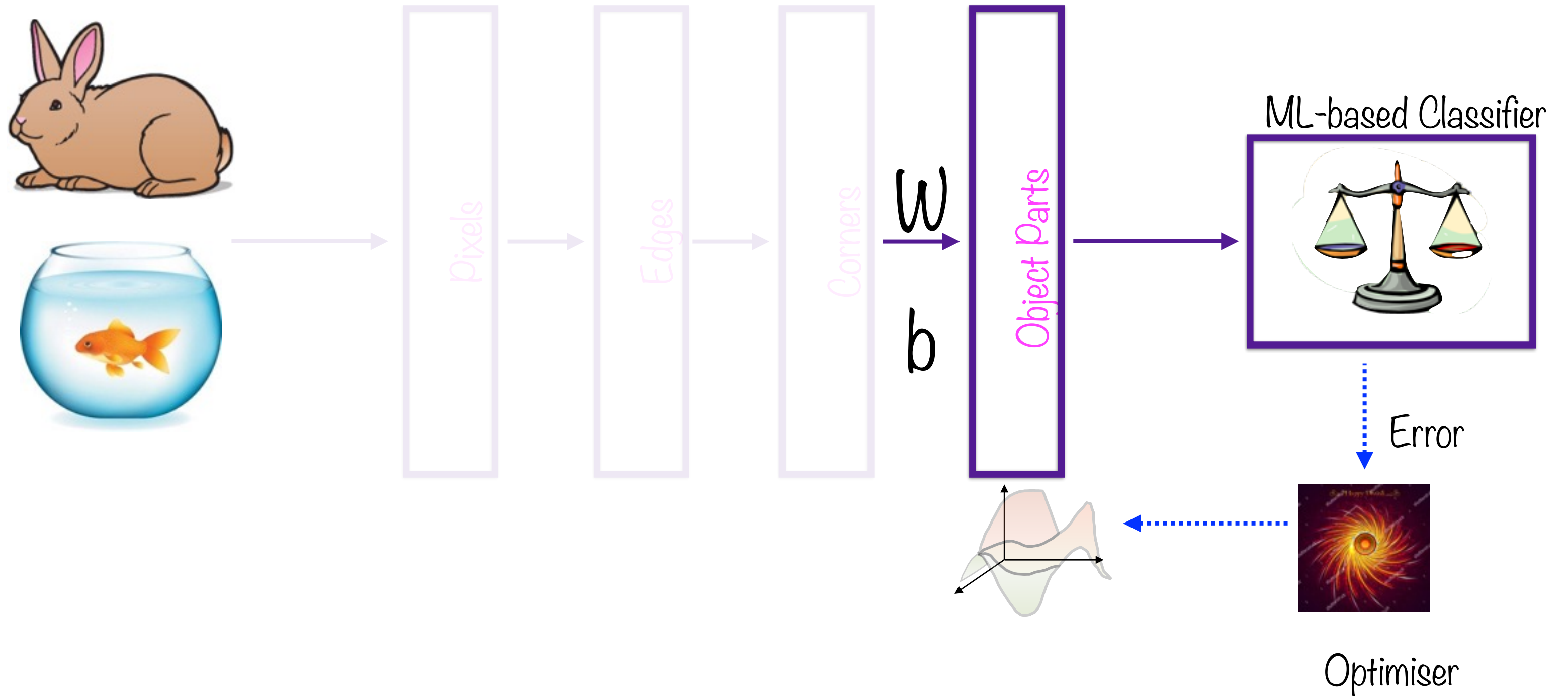
“Training” the Algorithm



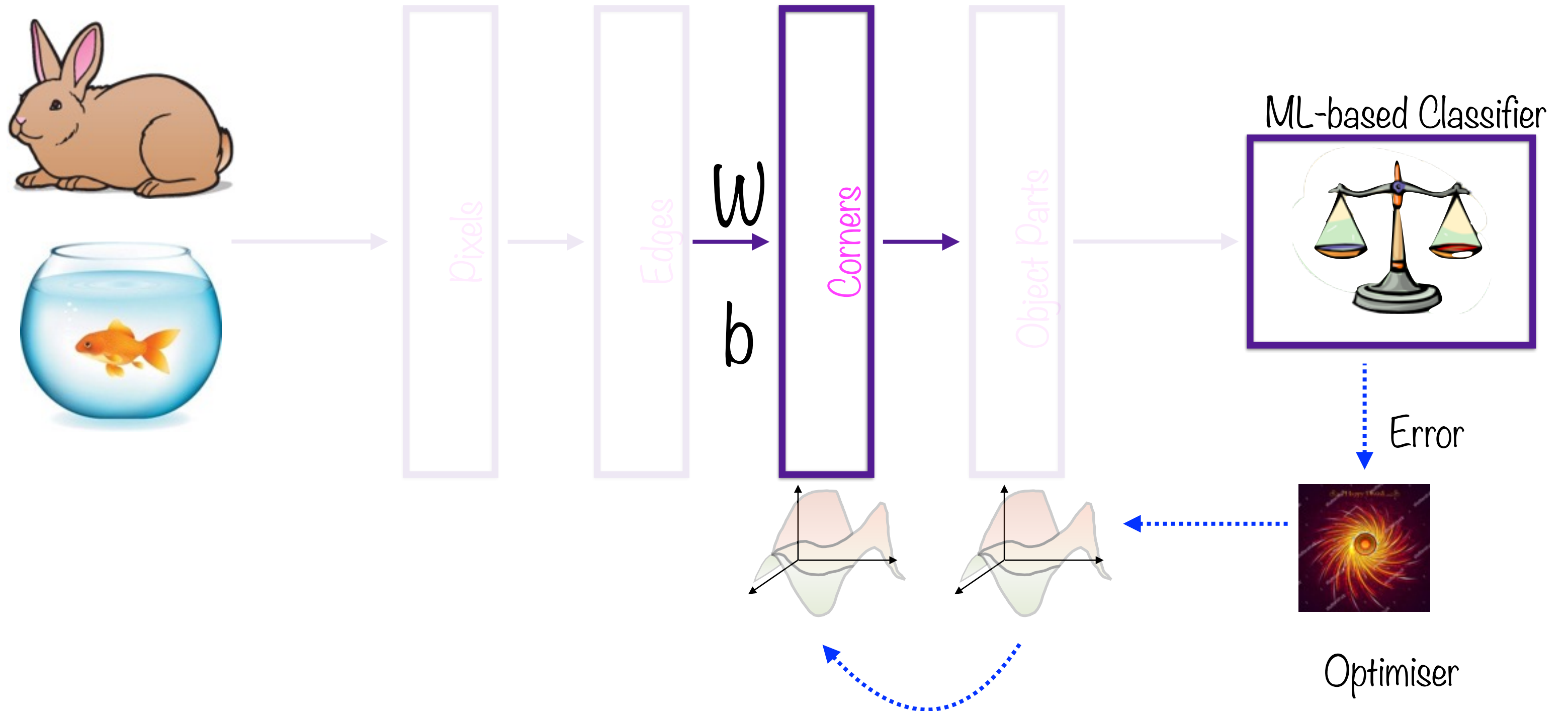
Training via Back Propagation



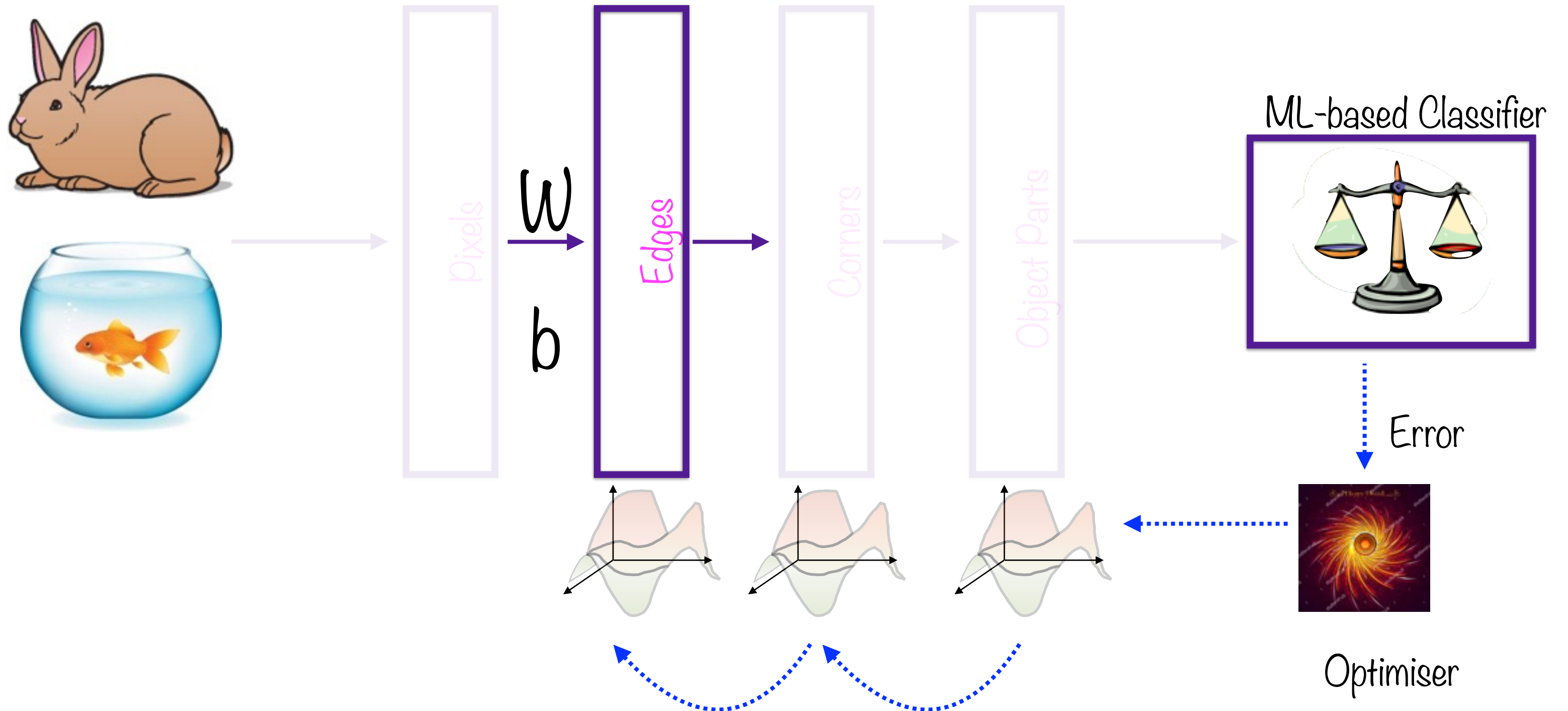
Training via Back Propagation



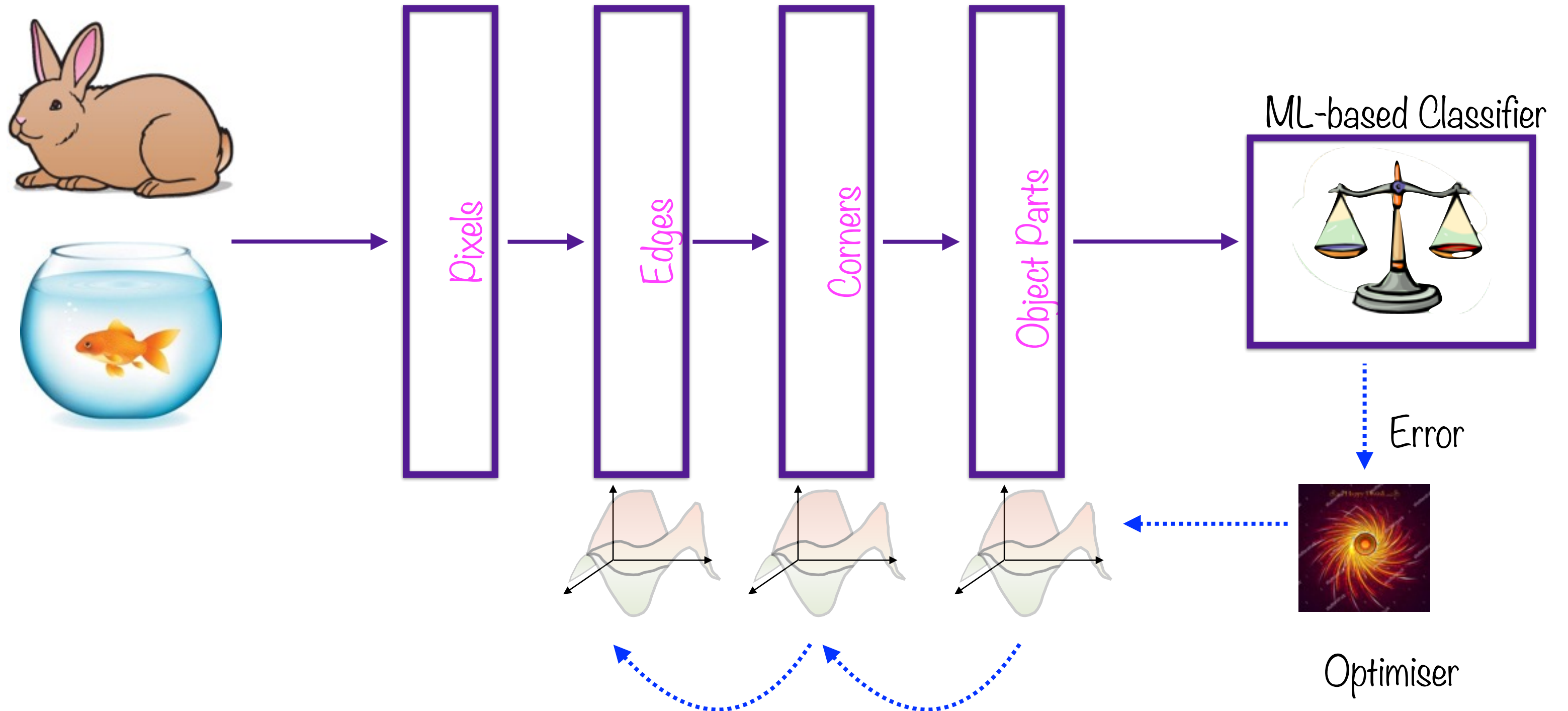
Training via Back Propagation



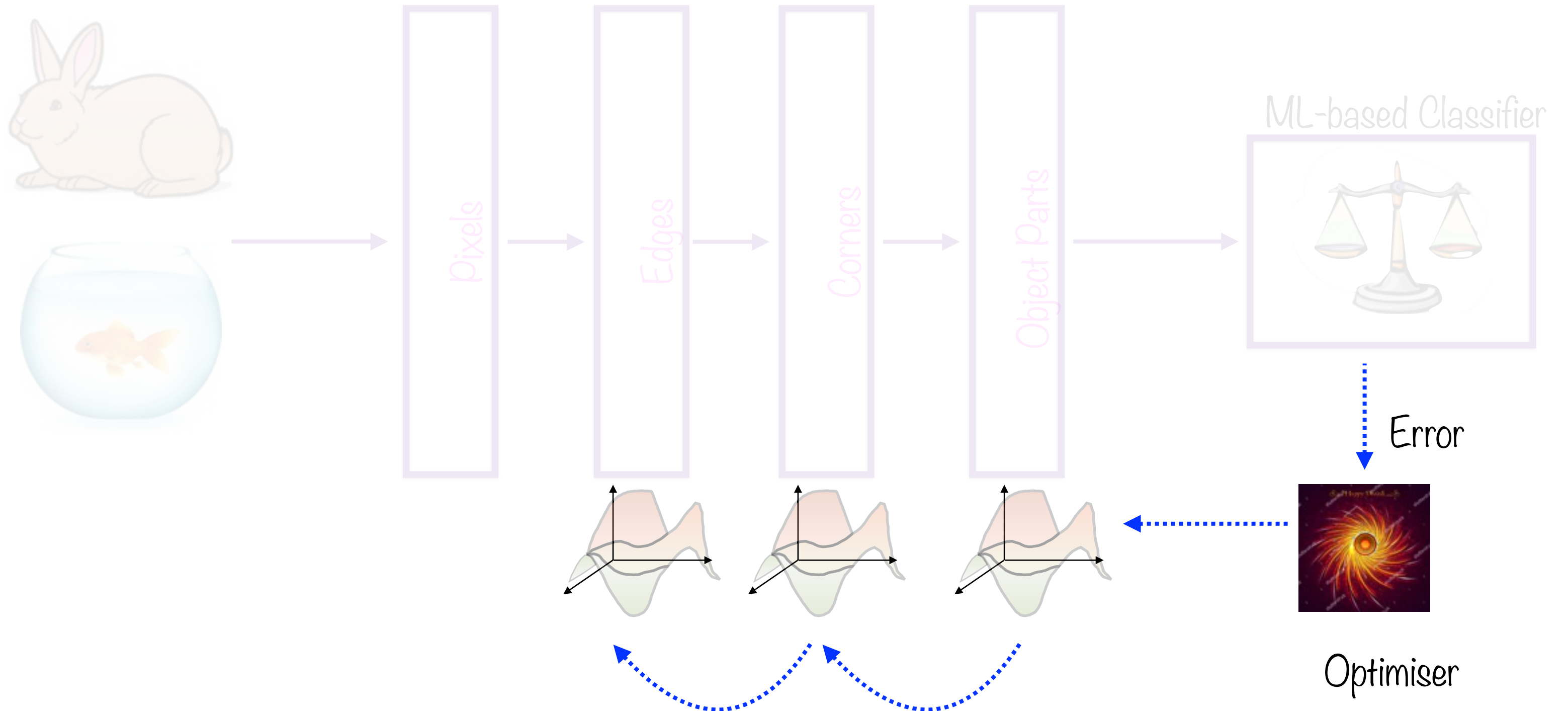
Training via Back Propagation



Training via Back Propagation



Training via Back Propagation



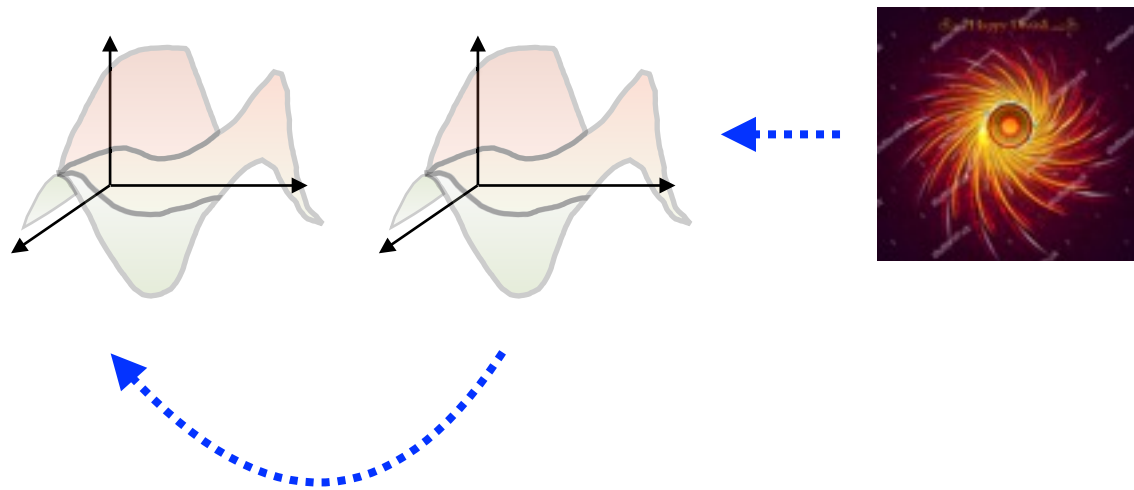
Back propagation allows the weights and biases of the neurons to **converge** to their final values

Back Propagation

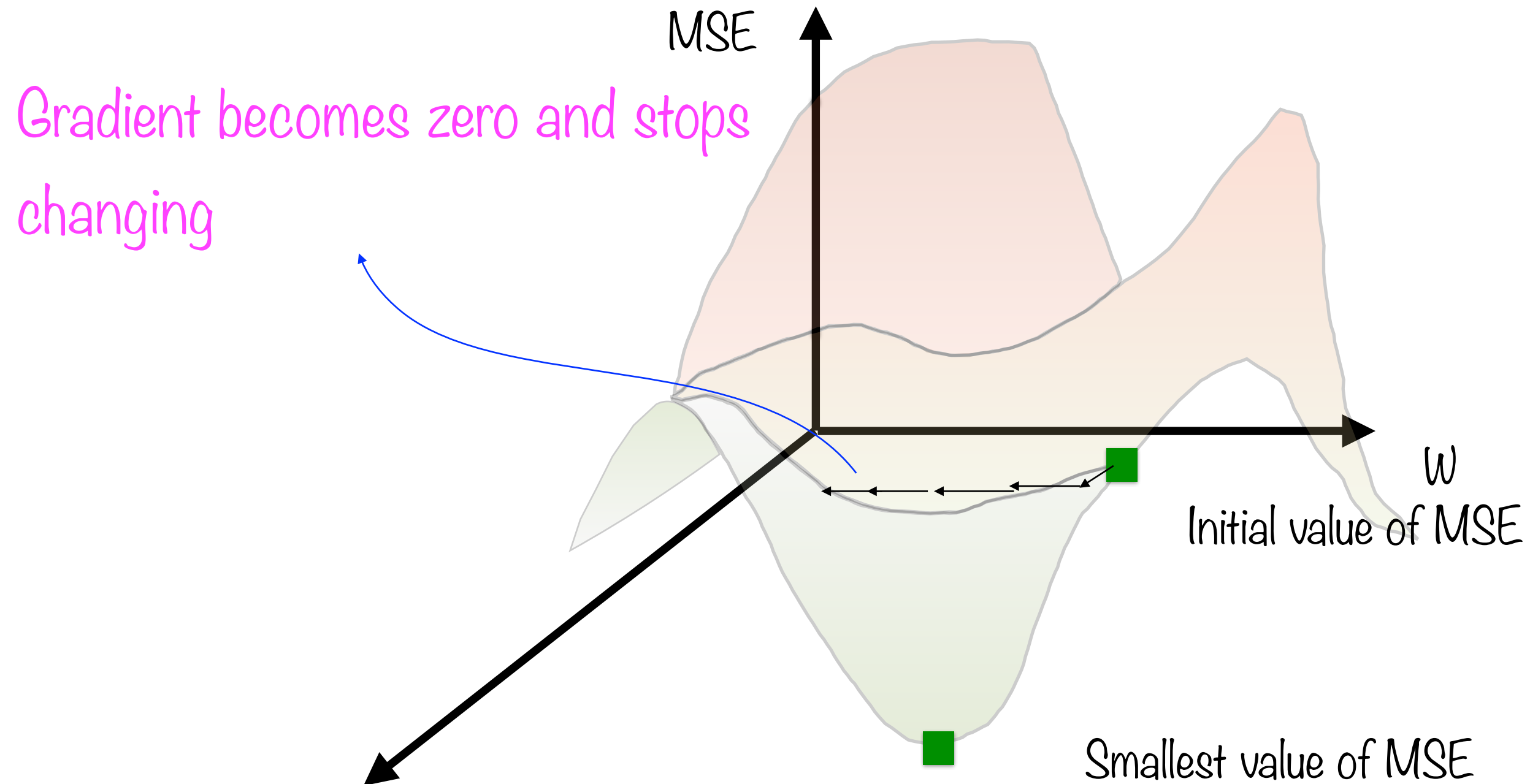
This is an iterative process

Fails either if

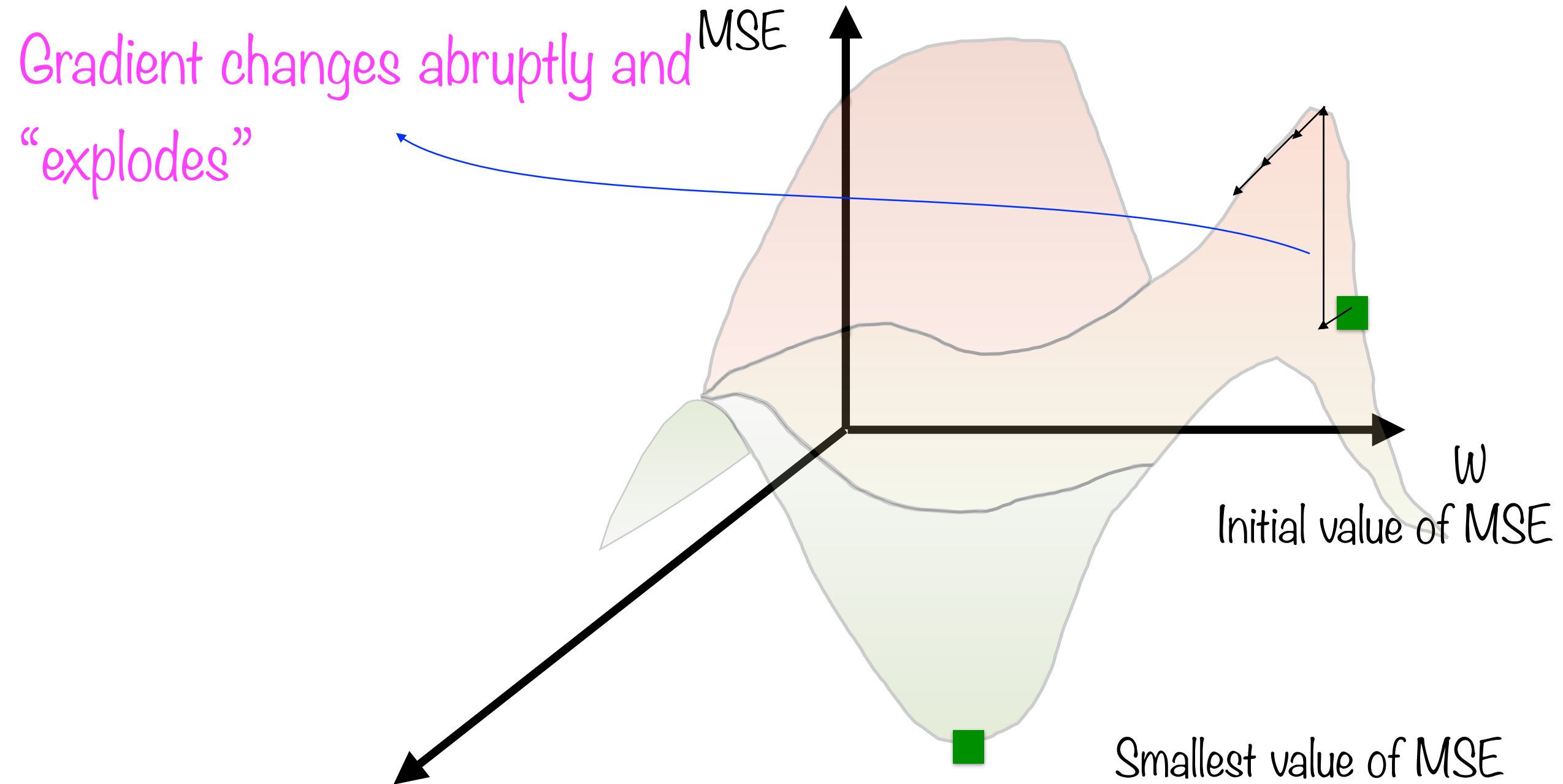
- gradients don't change at all
- gradients change too fast



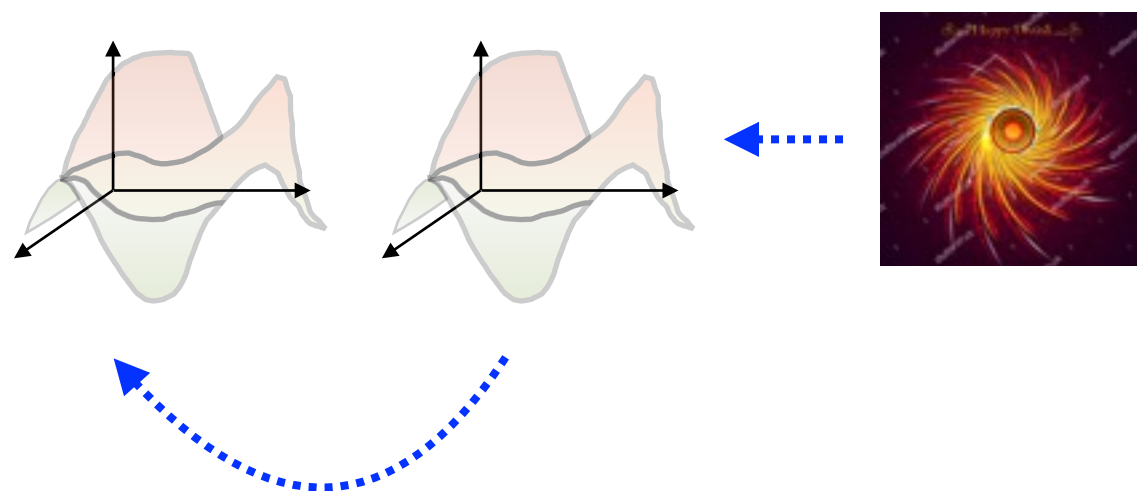
“Vanishing Gradient Problem”



“Exploding Gradient Problem”



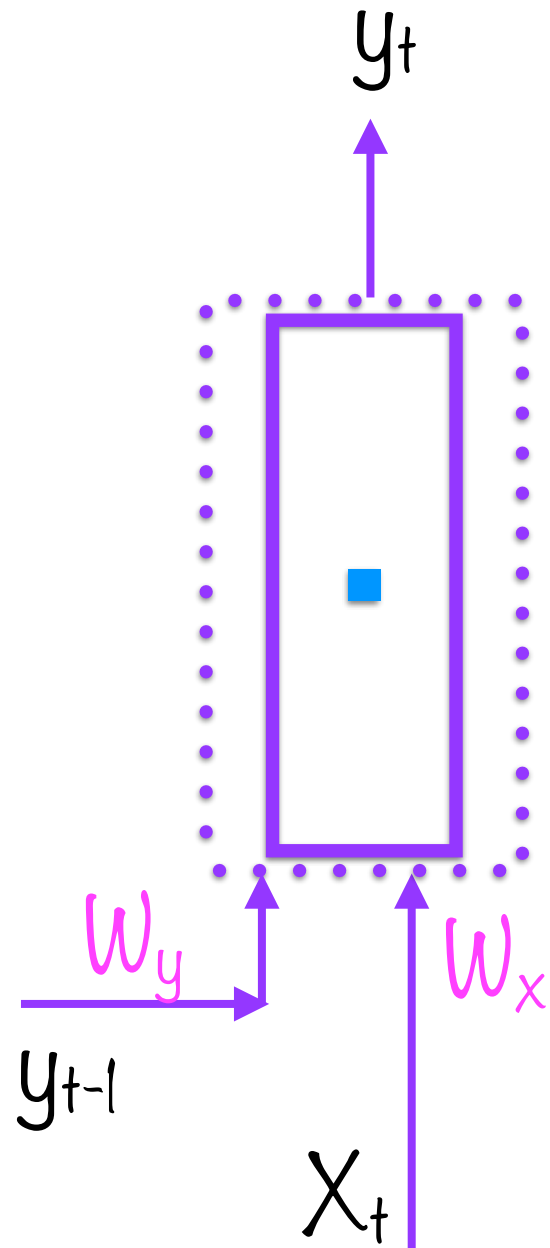
Vanishing and Exploding Gradients



Back propagation fails if

- gradients are **vanishing**
- gradients are **exploding**

Training RNNs



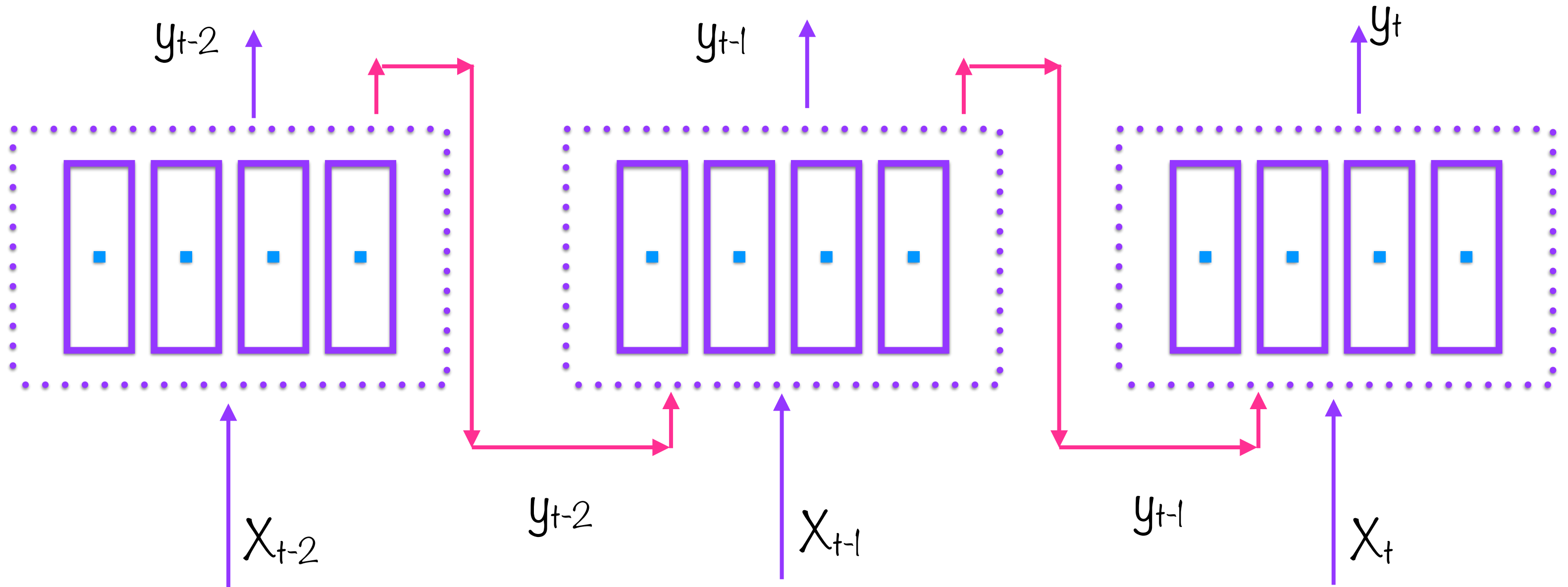
Training RNNs poses some specific challenges

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

Learning the (Recent) Past

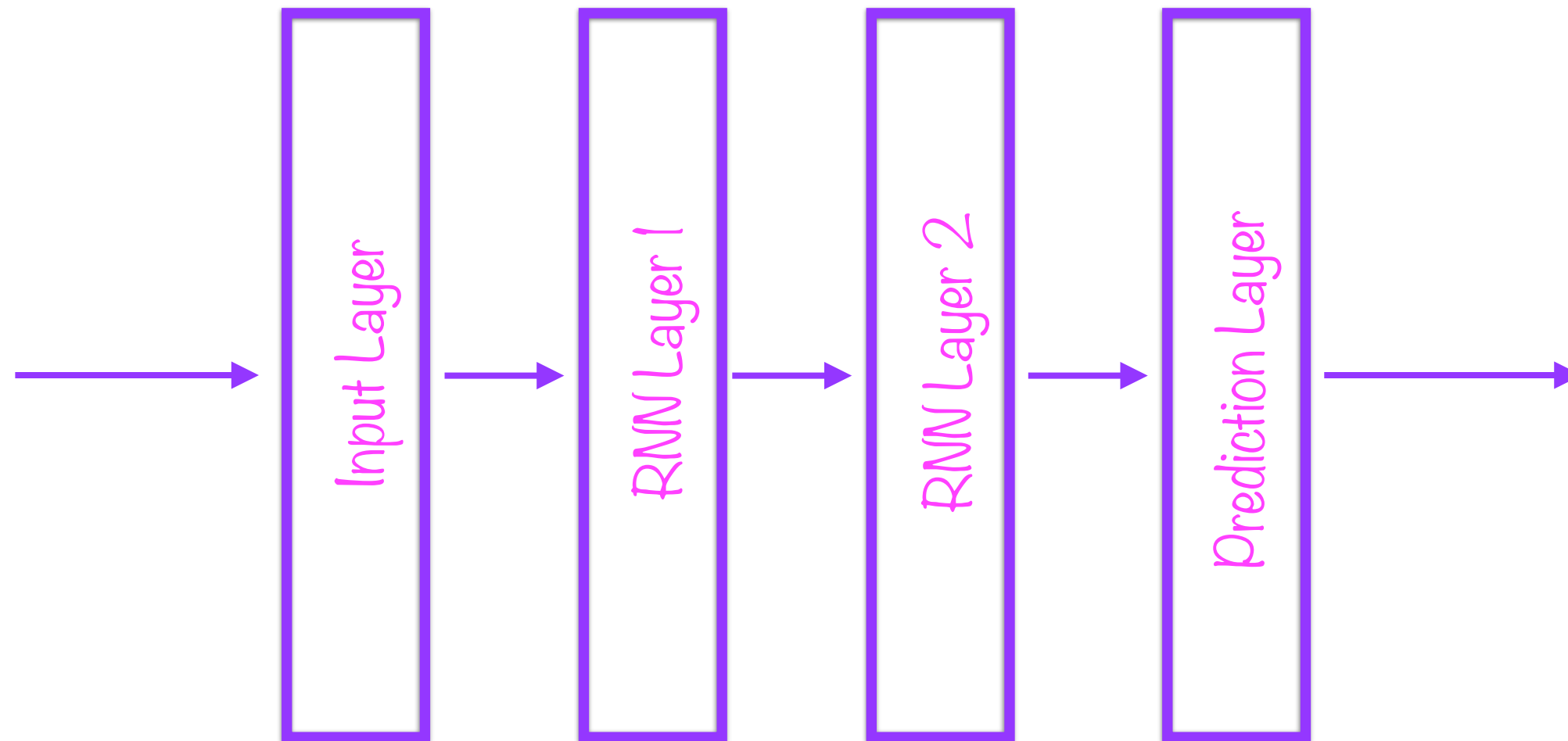
Unrolling the RNN through time helps learn the past

Layer of Recurrent Neurons



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

Layer of Recurrent Neurons



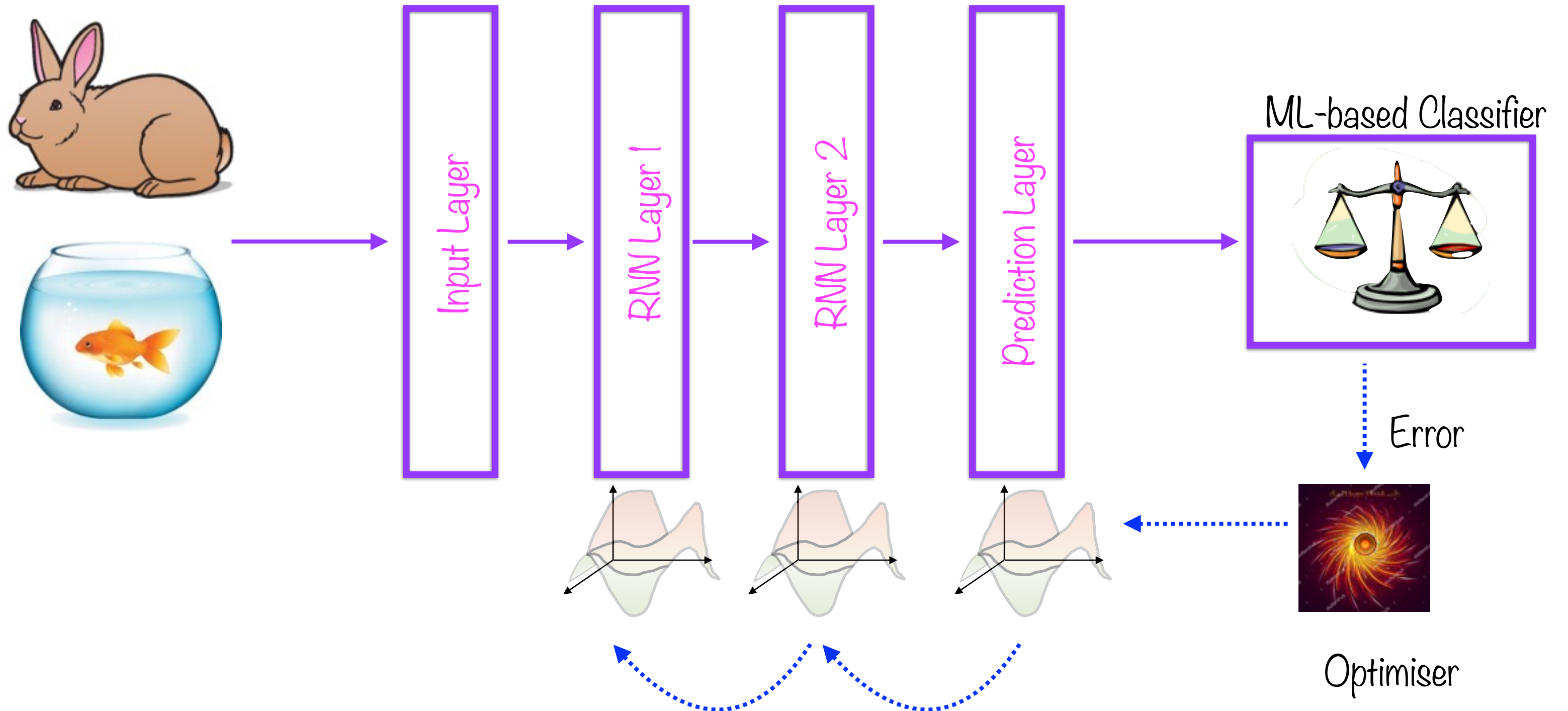
$t=0$

$t=1$

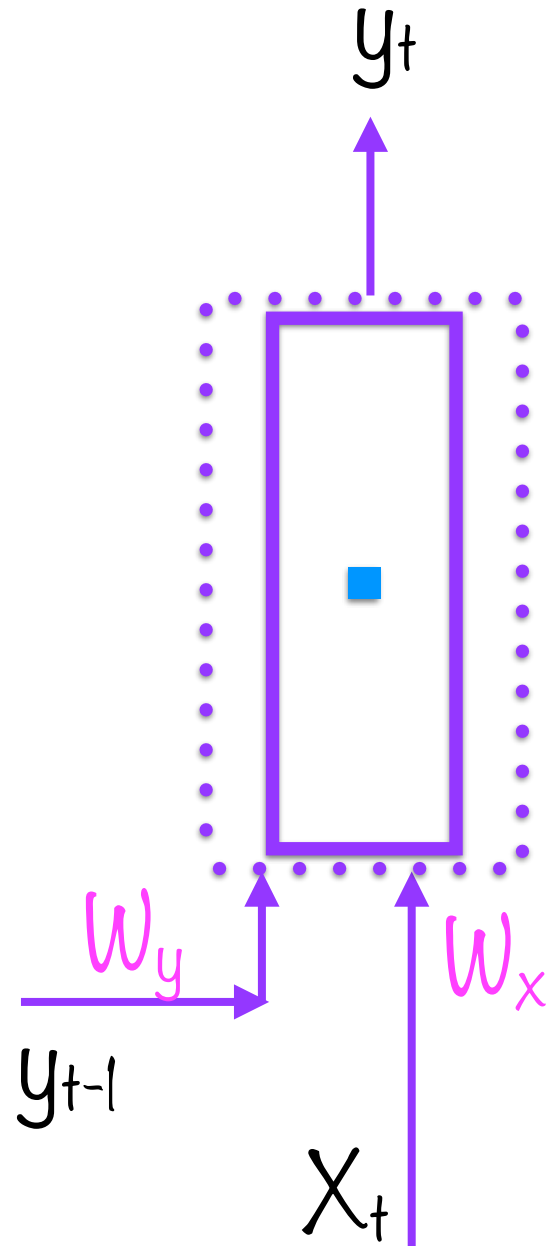
time

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

Back Propagation Through Time (BPTT)



BPTT



Back Propagation Through Time is the *tweaked* version of Back Propagation for RNN training

Need as many layers as past time periods

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

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**

Coping with Vanishing/Exploding Gradients

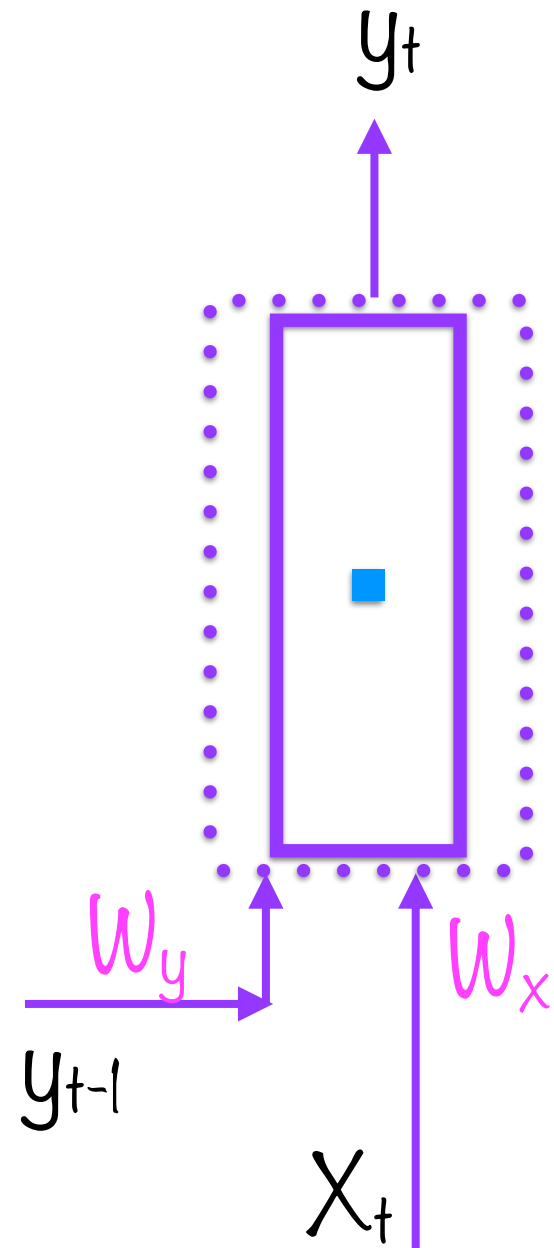
Proper initialisation

Non-saturating activation function

Batch normalisation

Gradient clipping

BPTT



If output relies on distant past...

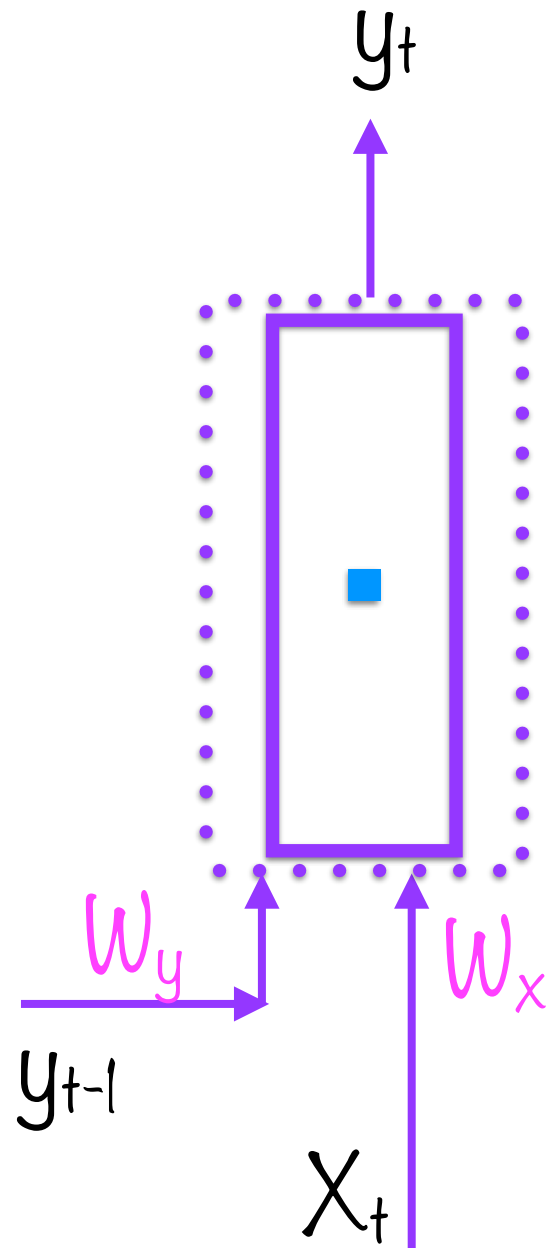
...Vanishing/exploding gradients very likely

One option - truncated BPTT

(Just discard distant past)

Not ideal - need to store long-term state

Memory and State



Vanishing and exploding gradients are mitigated by adding **long term memory** to RNN cells

This is the basic idea behind LSTM cells

[None, n_steps, n_inputs]

Shape of RNN Inputs

Batch size, steps in time, dimensionality of inputs

[None, n_steps, n_inputs]

Batch Size

Number of instances to be fed into the RNN

[None, **n_steps**, n_inputs]

Number of Steps in Time

Discrete time instances for which inputs are available

[None, n_steps, n_inputs]

Dimensionality of Inputs

Vector size representing one input at a particular time instance

Representing Images As Steps in Time

0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8	0.7	0.8	0.9	0
0	0	0	0.2	0.8	0
0	0	0	0.2	0.2	0

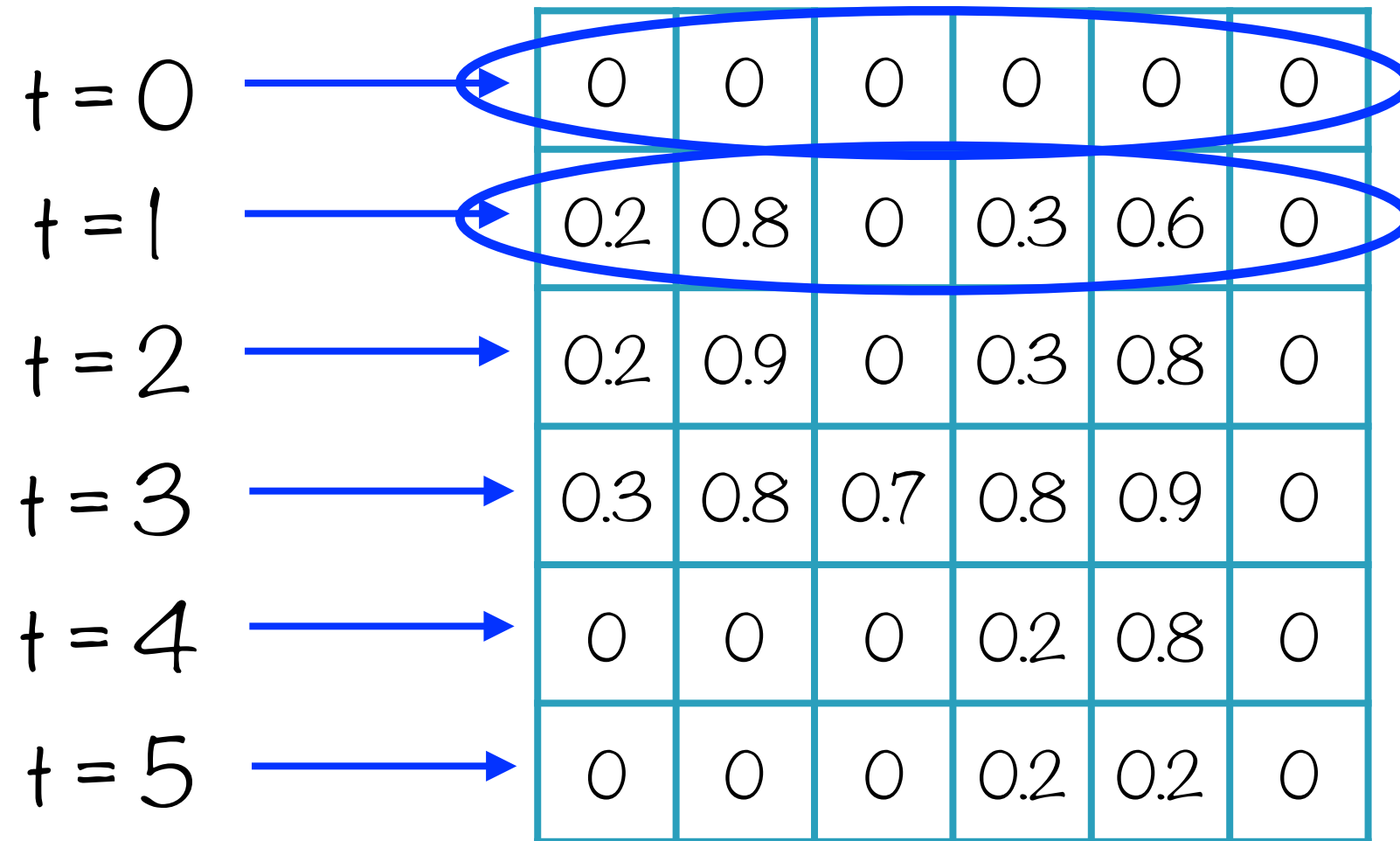
Structure the image array so that every row is one step
in time

Representing Images As Steps in Time

$t = 0$	→	0	0	0	0	0	0
$t = 1$	→	0.2	0.8	0	0.3	0.6	0
$t = 2$	→	0.2	0.9	0	0.3	0.8	0
$t = 3$	→	0.3	0.8	0.7	0.8	0.9	0
$t = 4$	→	0	0	0	0.2	0.8	0
$t = 5$	→	0	0	0	0.2	0.2	0

Structure the image array so that every row is one step
in time

Representing Images As Steps in Time

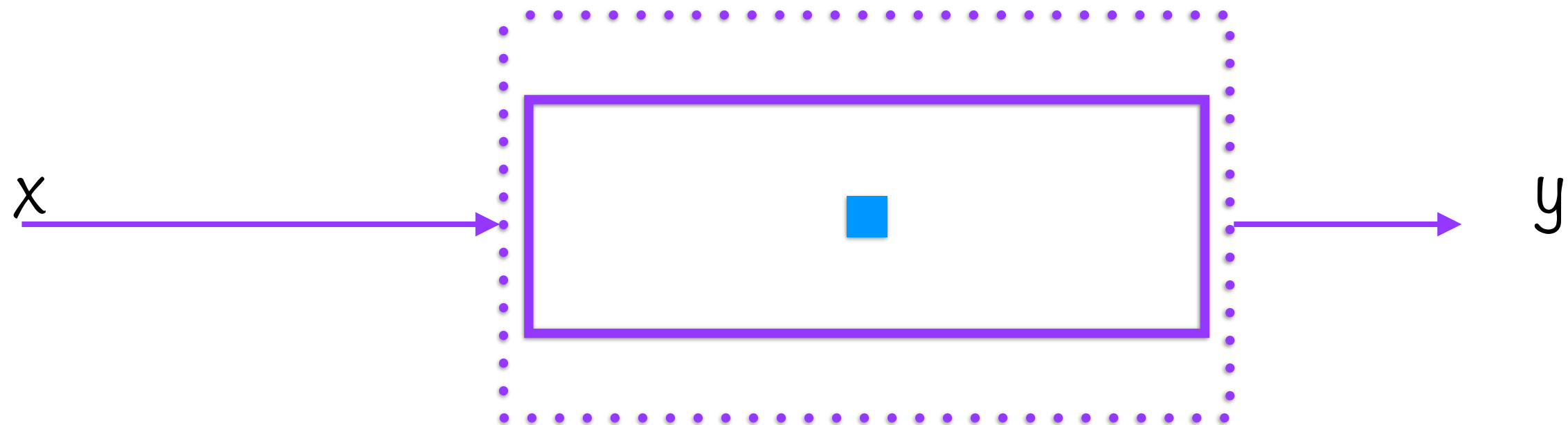


$t = 0$	→	0	0	0	0	0	0
$t = 1$	→	0.2	0.8	0	0.3	0.6	0
$t = 2$	→	0.2	0.9	0	0.3	0.8	0
$t = 3$	→	0.3	0.8	0.7	0.8	0.9	0
$t = 4$	→	0	0	0	0.2	0.8	0
$t = 5$	→	0	0	0	0.2	0.2	0

The dimensions of each input is the width of the input

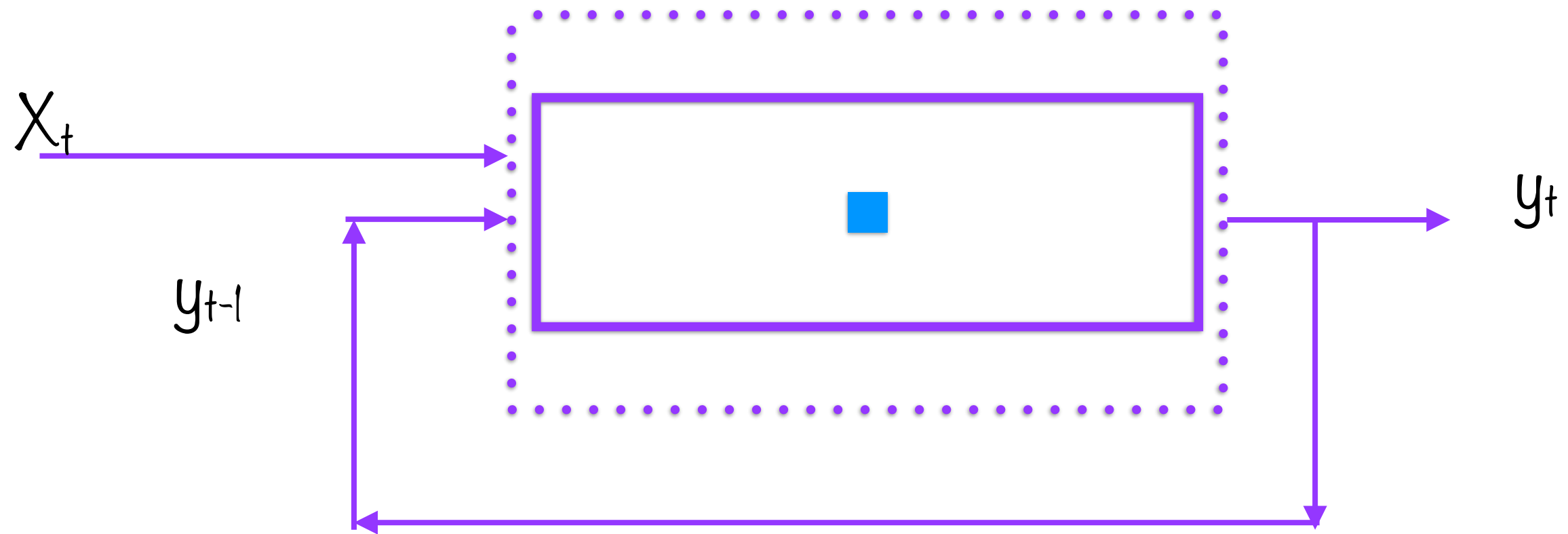
Long Memory RNN Cells

Simplest Feed-forward Neuron



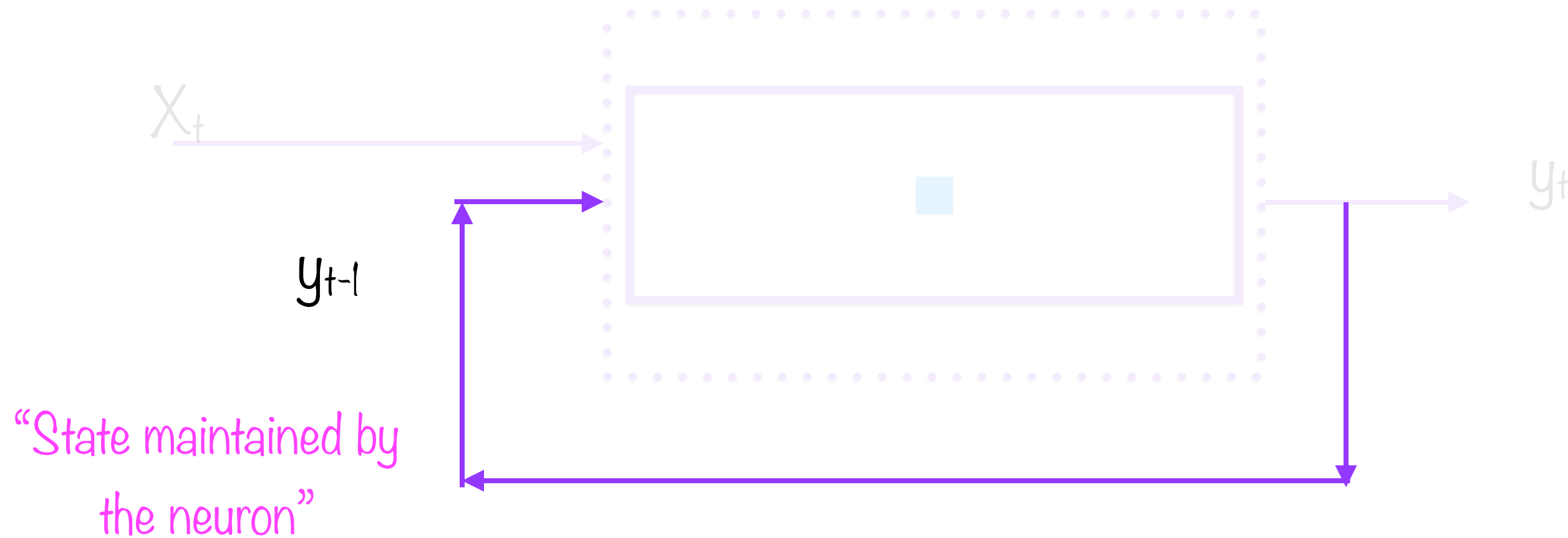
Neuron

Simplest Recurrent Neuron

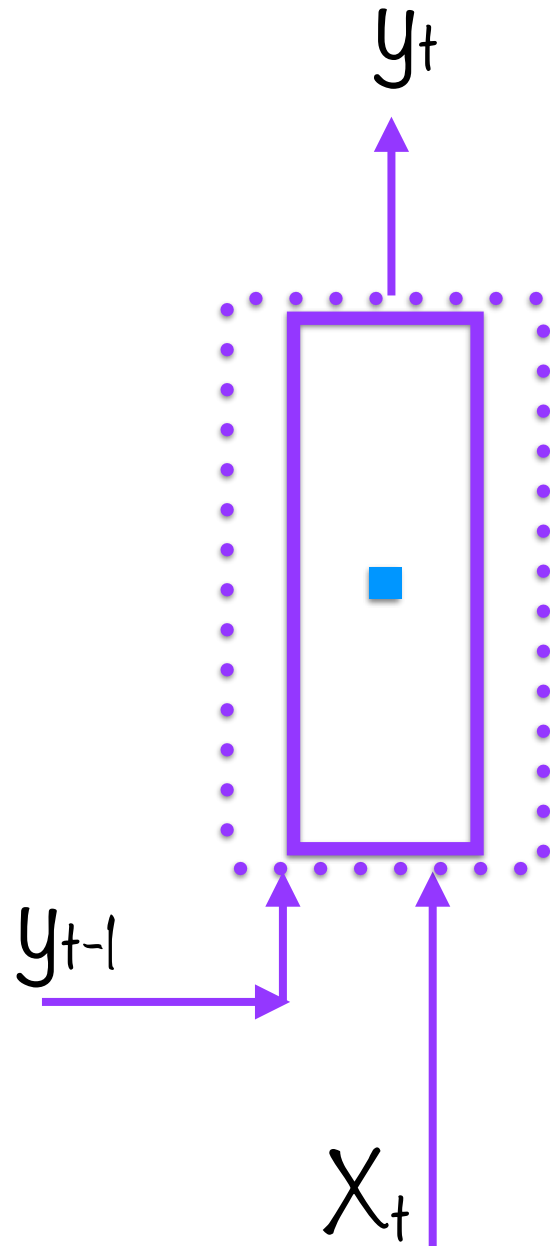


Recurrent Neuron

Simplest Recurrent Neuron



Memory and State



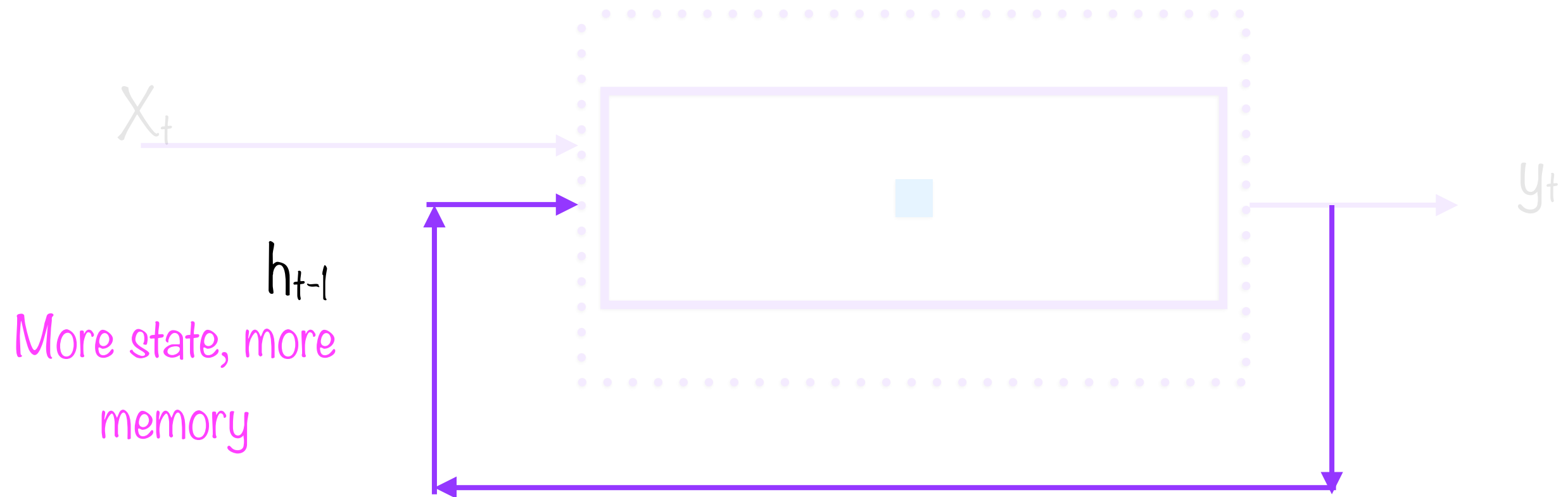
The output at time t depends on output at time $t-1$

The output is effectively state maintained by the recurrent neuron

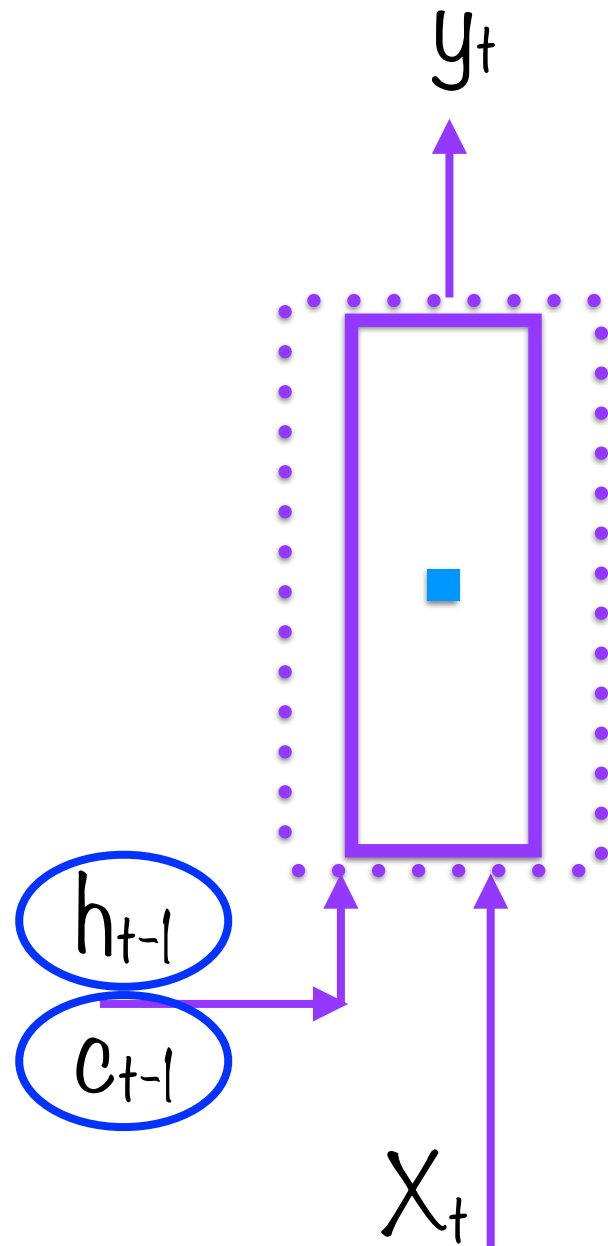
The neuron “remembers”...

...it has **memory**

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 neurons have several advantages over
basic RNNs**

Long Memory RNNs



Advantages in Training

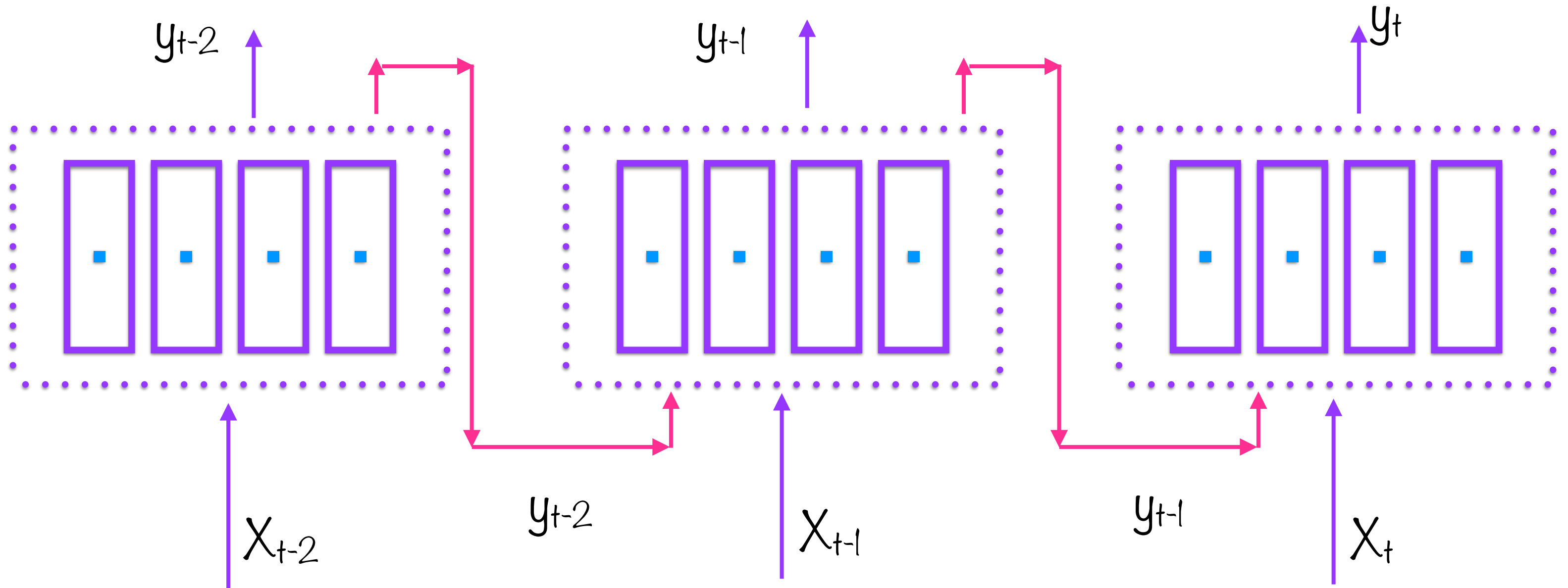
Faster training, nicer gradients



Advantages in Prediction

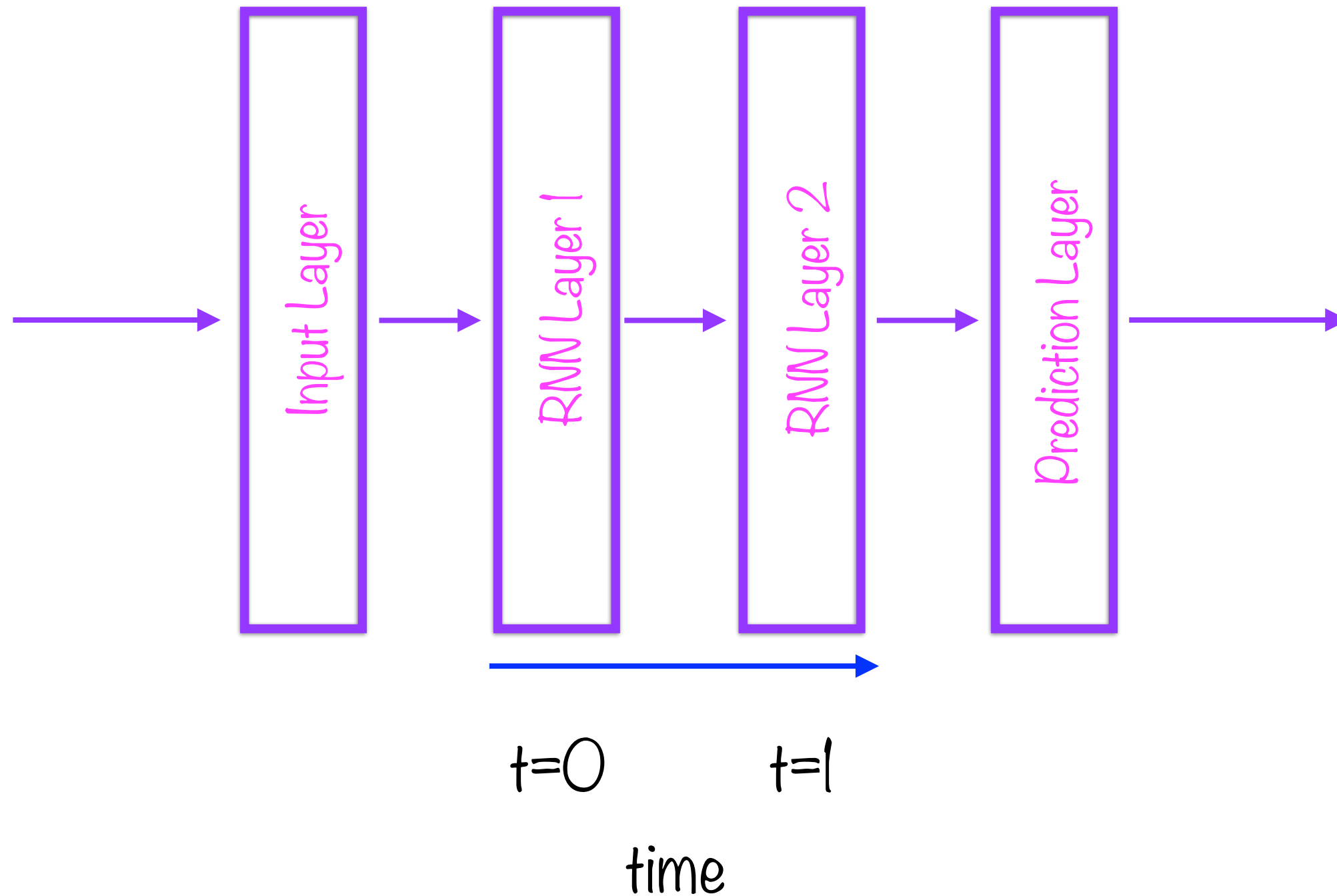
No need to truncate BPTT

Layer of Recurrent Neurons

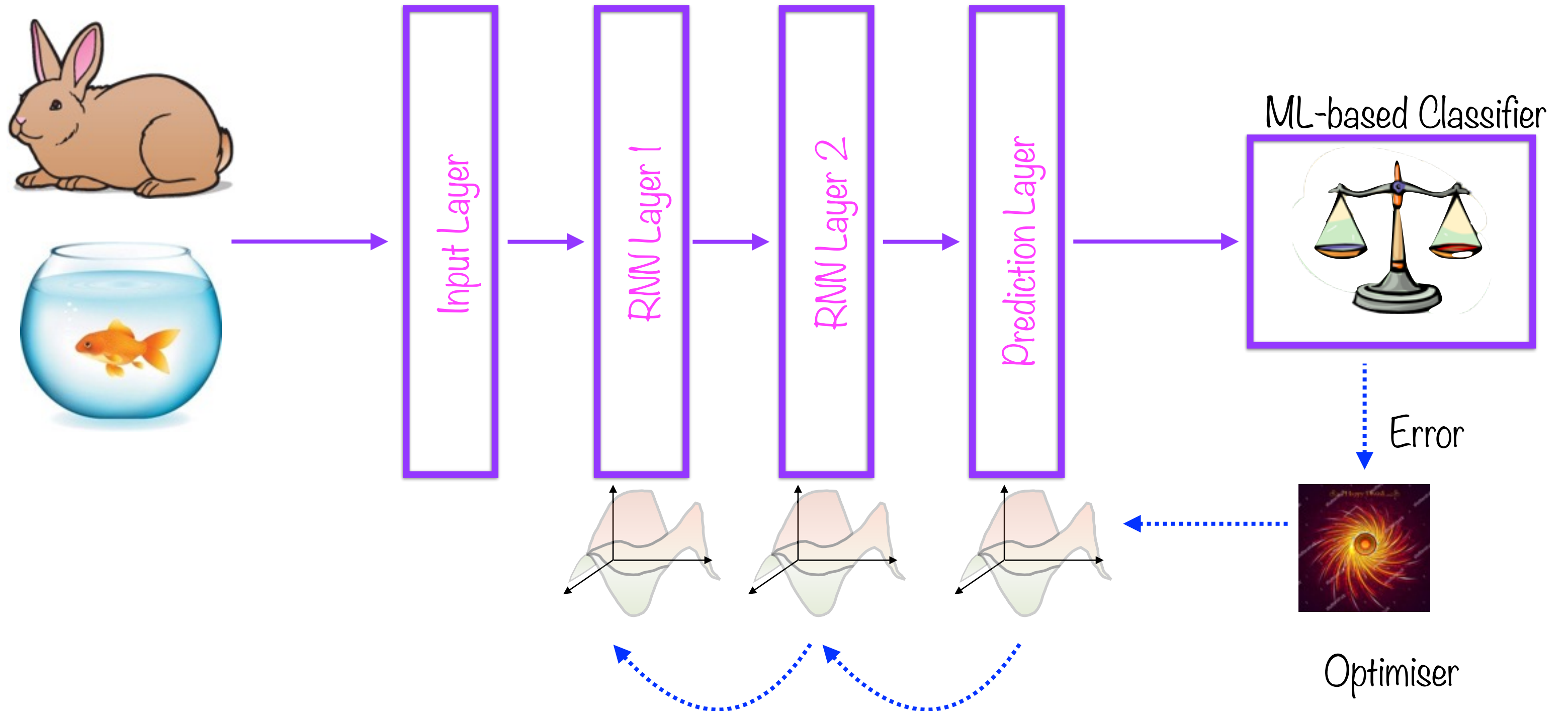


Each unit is a cell, unrolled through time to form the layers of a neural network

Layer of Recurrent Neurons



Back Propagation Through Time (BPTT)



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

Learning the Recent Past

RNN is still relatively shallow and can still learn the past

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

Learning the Distant Past

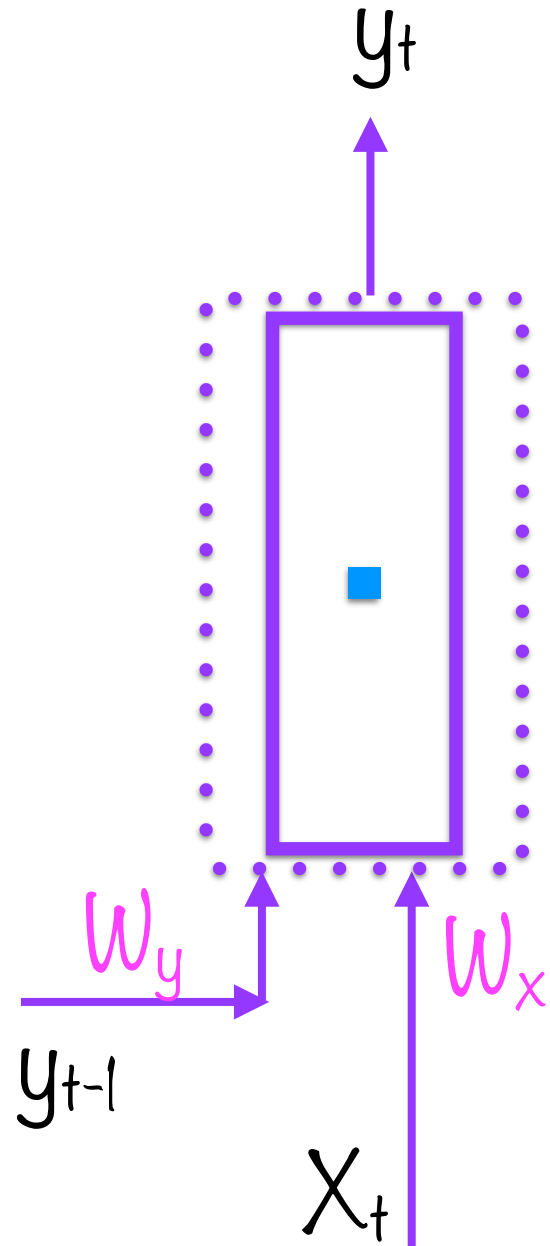
The unrolled RNN will be very very deep - vanishing gradients very likely (and hence so are exploding gradients)

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

Learning the Distant Past

Training such a deep RNN also takes forever

BPTT



If output relies on distant past...

...Vanishing/exploding gradients very likely

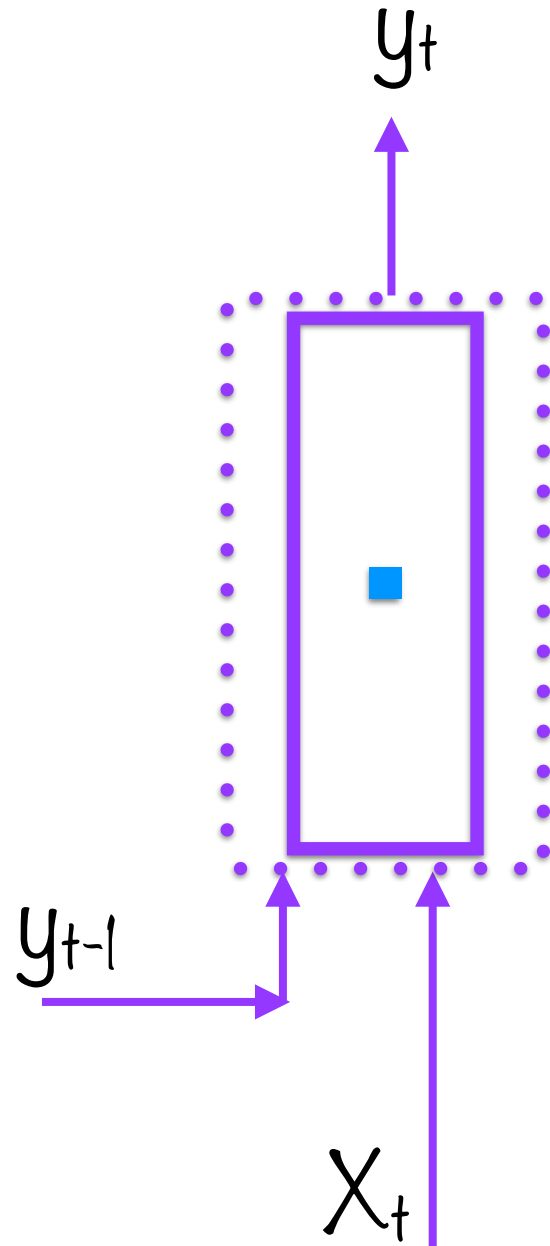
One option - truncated BPTT

Truncated BPTT

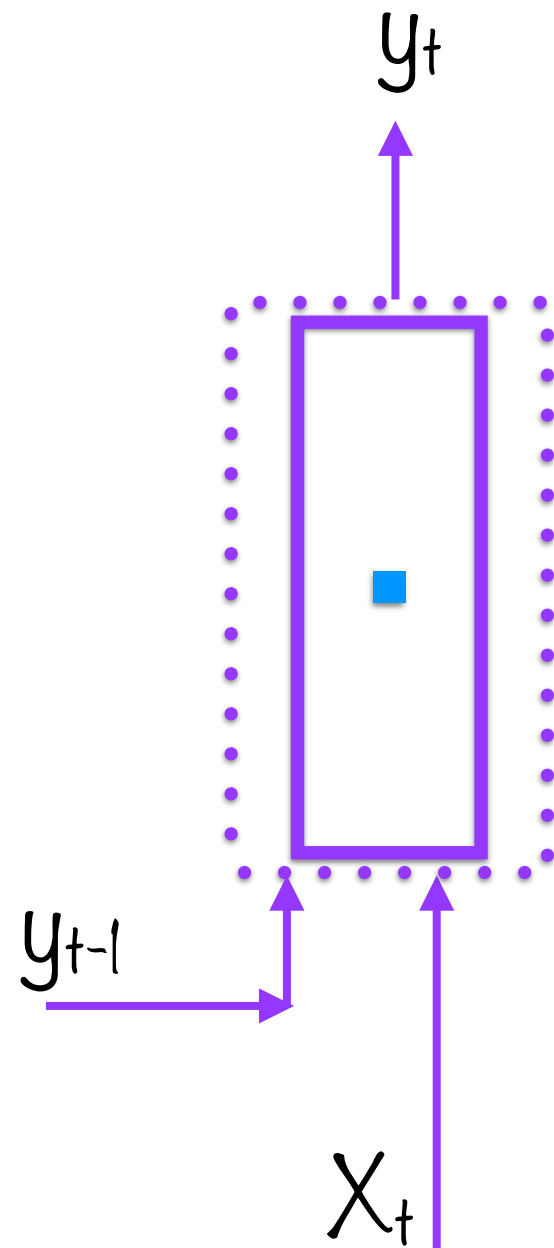
Simply truncate input sequence

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



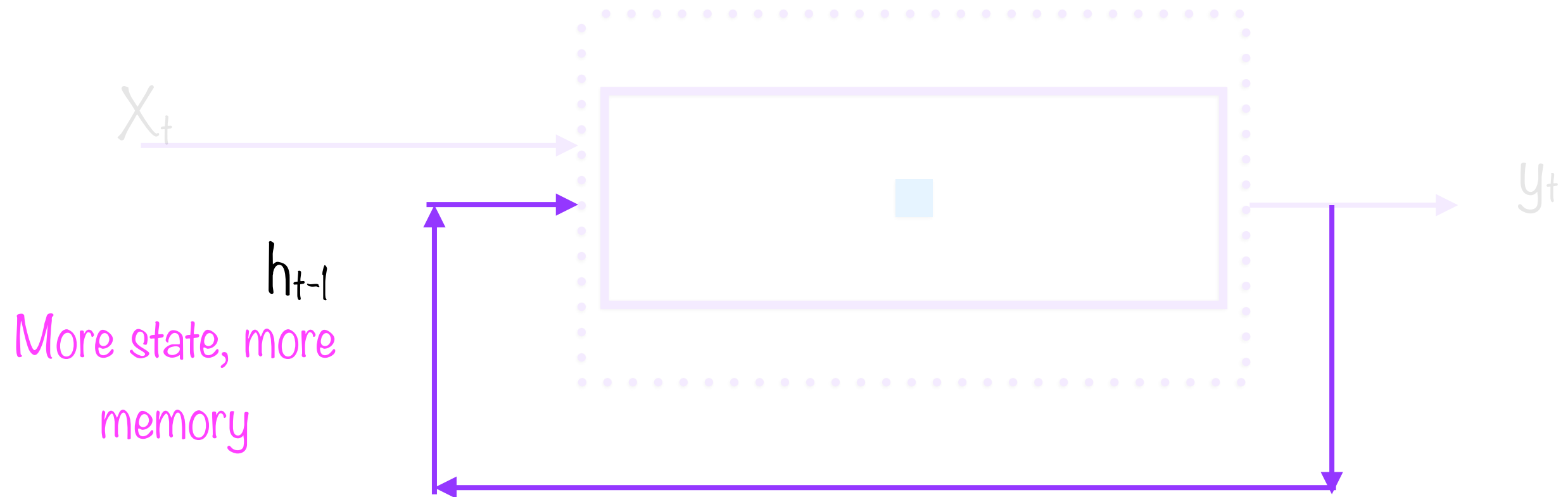
Truncated BPTT



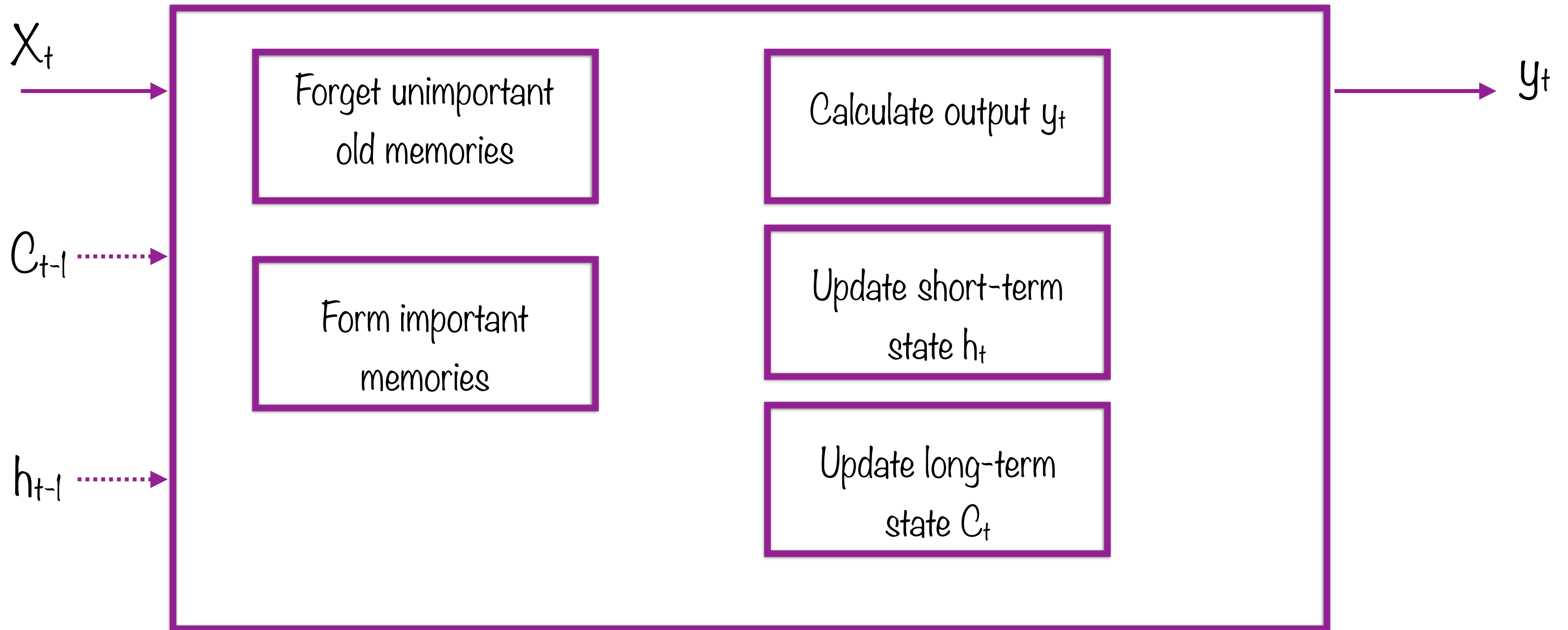
Truncated BPTT can kill prediction performance

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

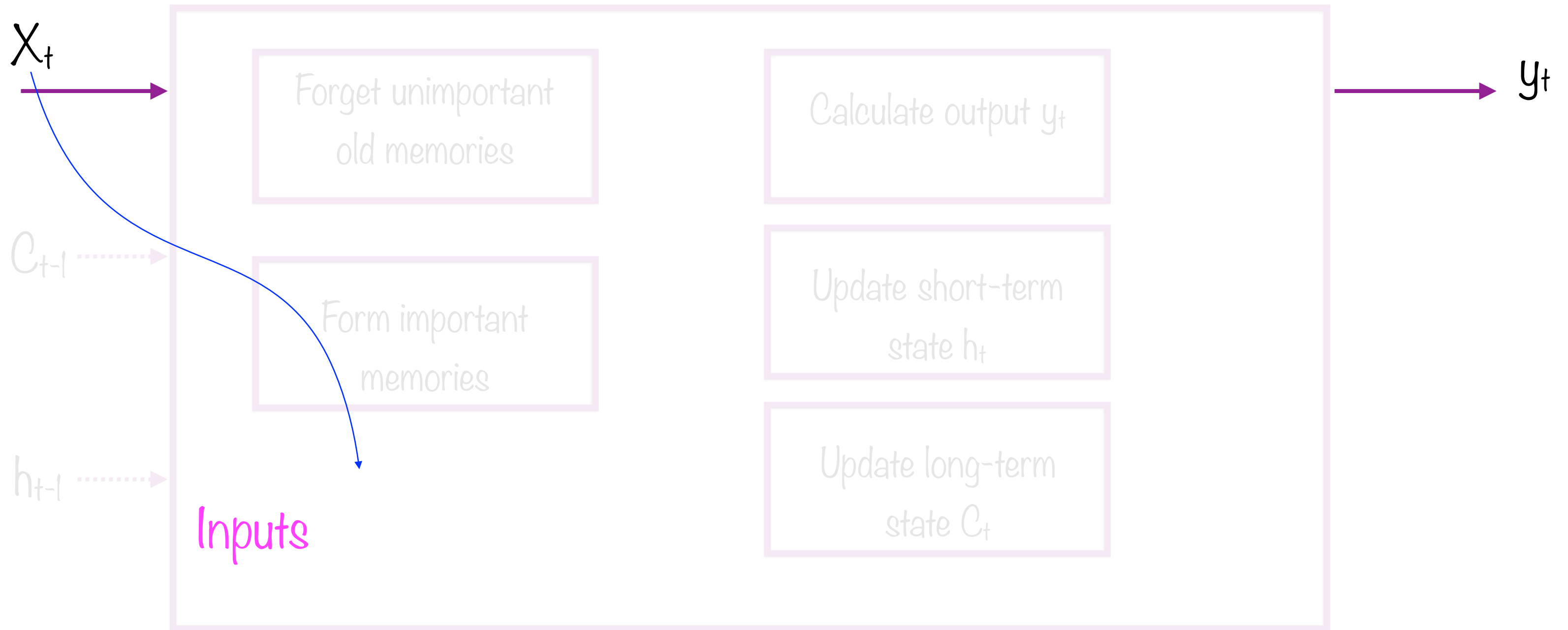
Long Memory Recurrent Neuron



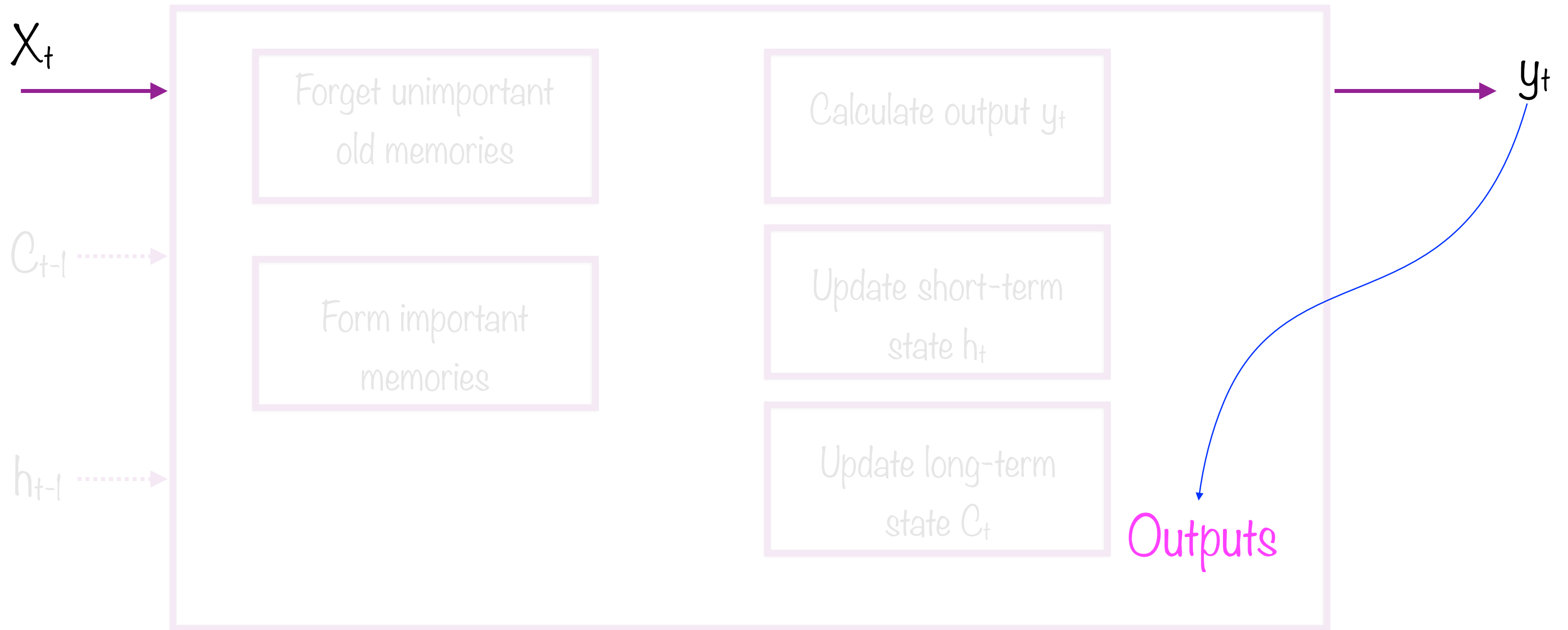
Long/Short-Term Memory Cell (LSTM)



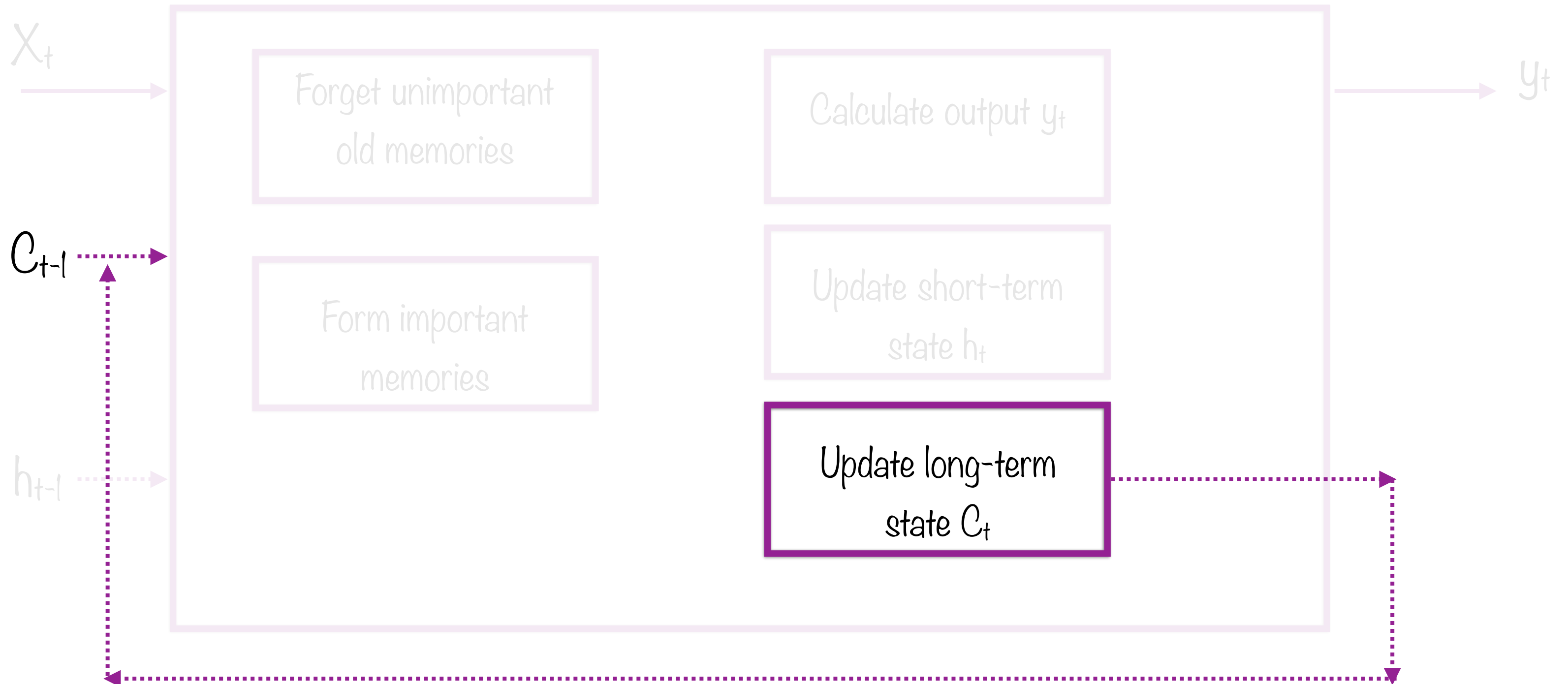
Long/Short-Term Memory Cell (LSTM)



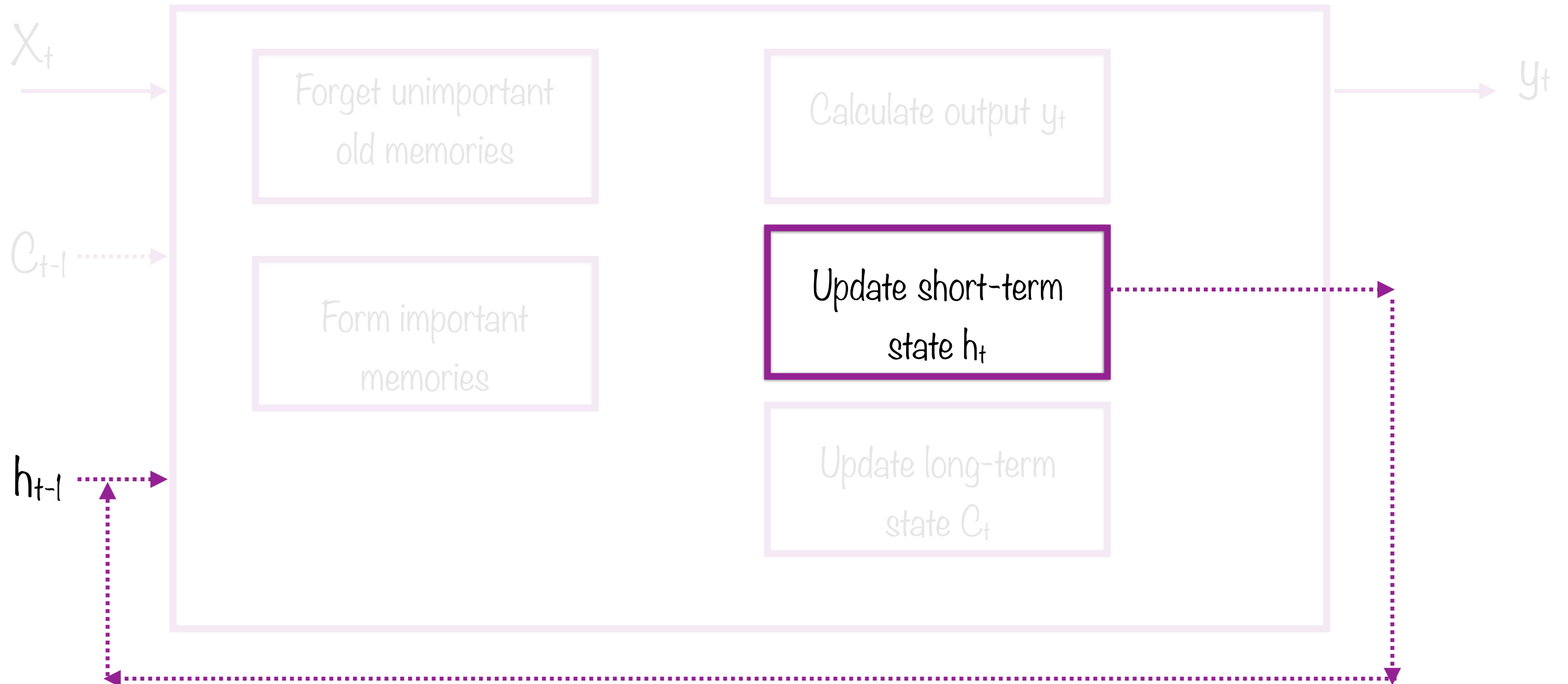
Long/Short-Term Memory Cell (LSTM)



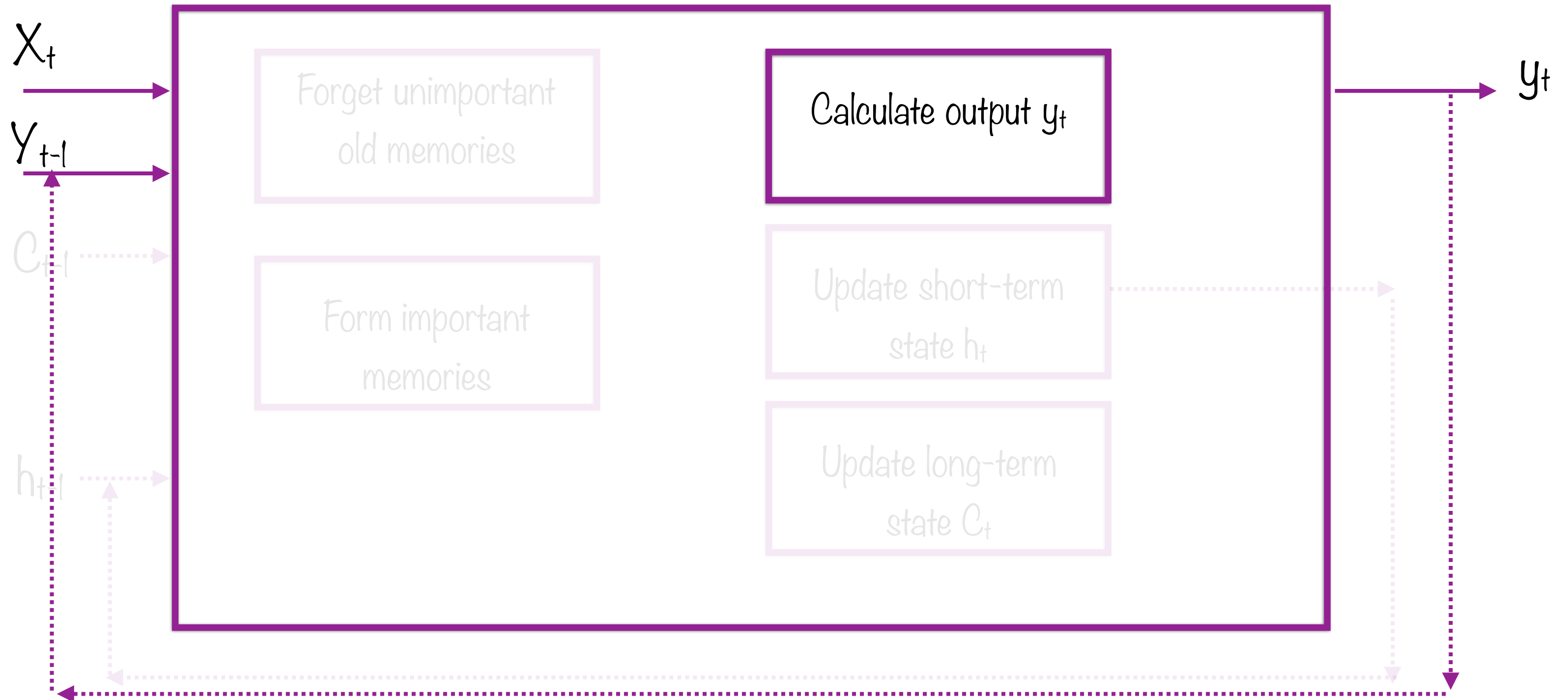
Long/Short-Term Memory Cell (LSTM)



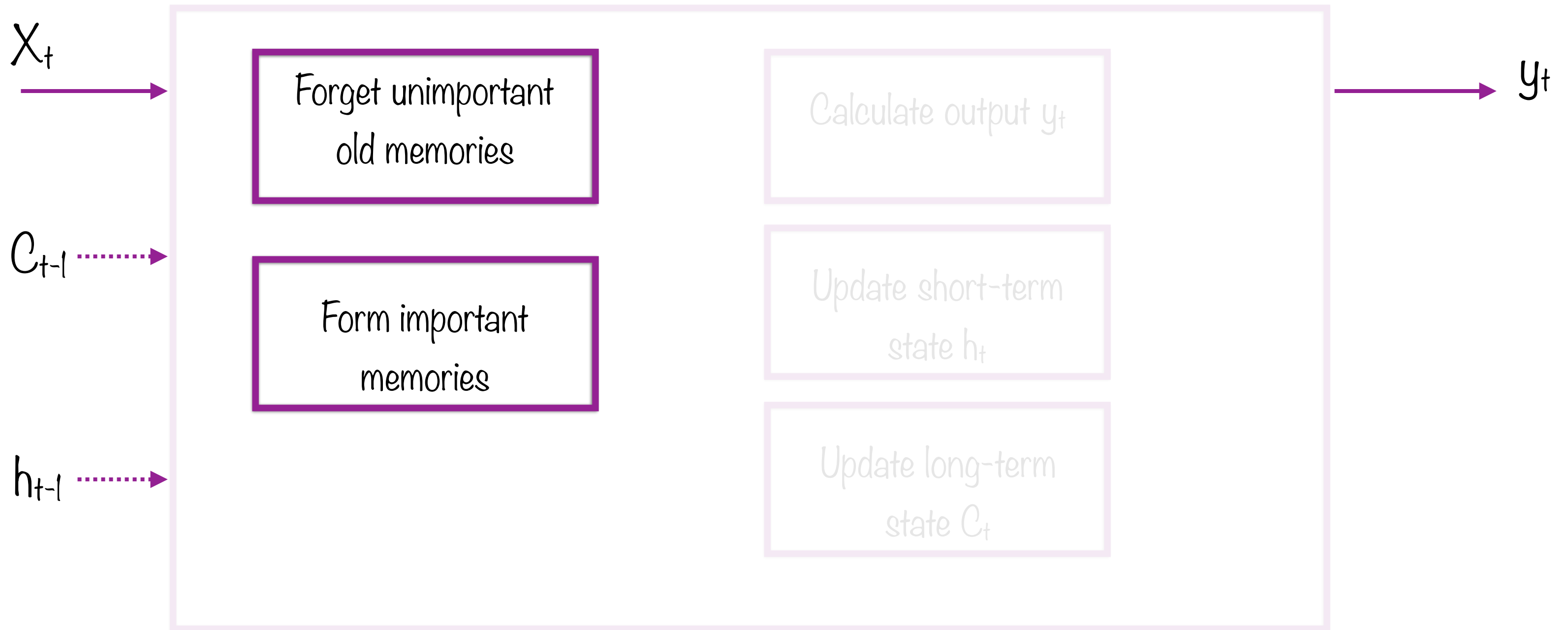
Long/Short-Term Memory Cell (LSTM)



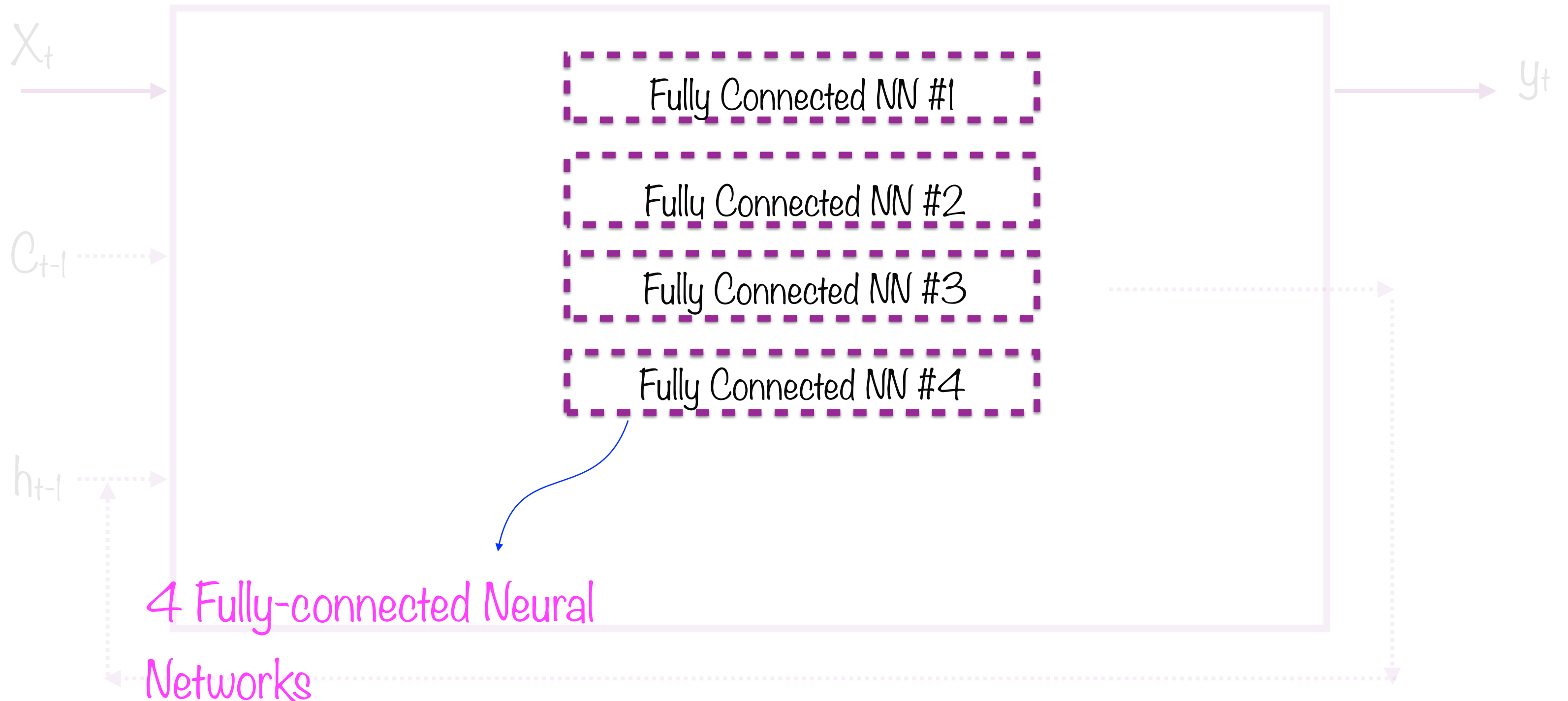
Basic RNN Cell



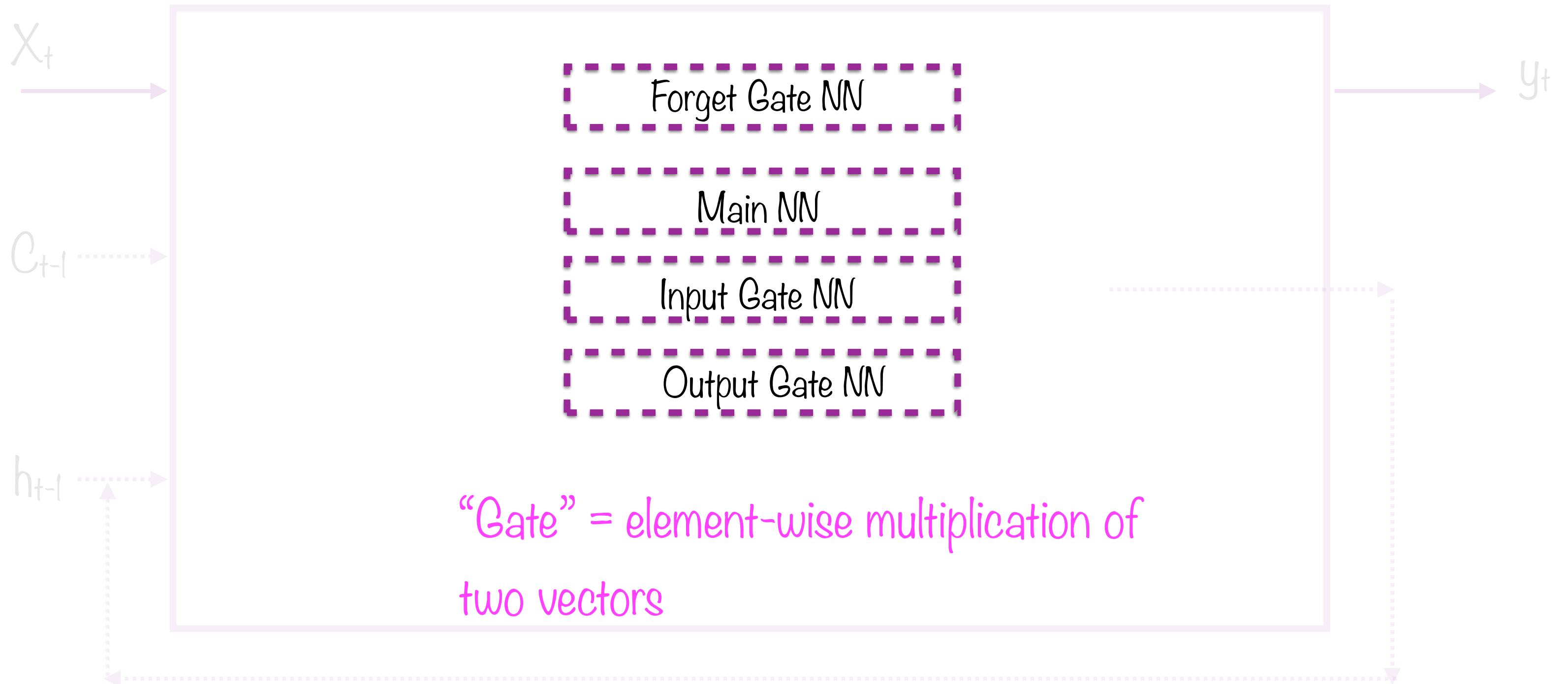
Long/Short-Term Memory Cell (LSTM)



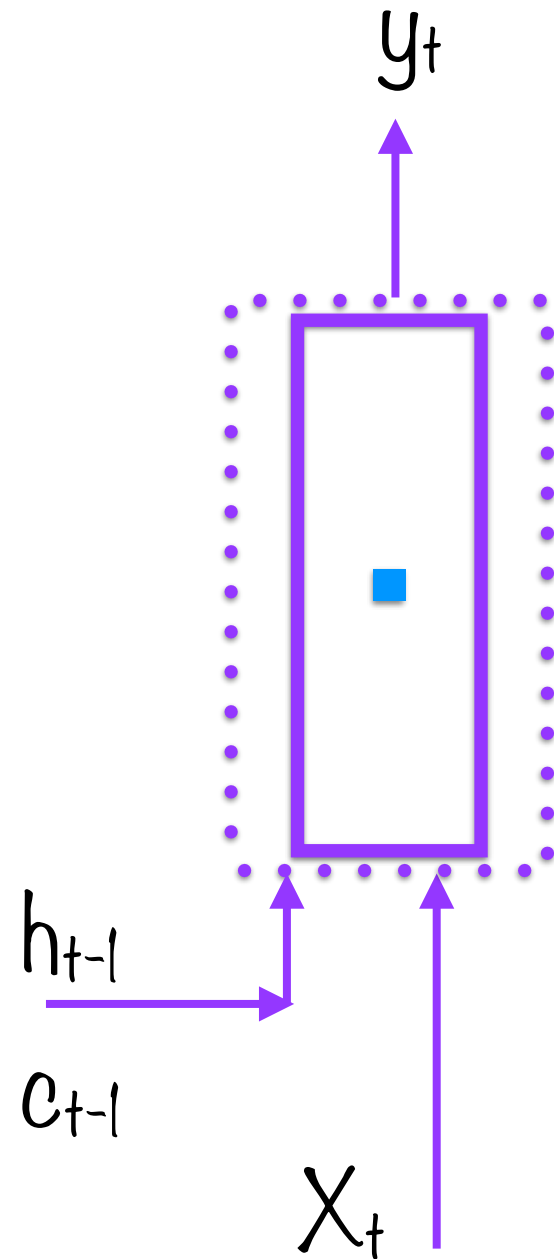
Long/Short-Term Memory Cell (LSTM)



Long/Short-Term Memory Cell (LSTM)



LSTM Cells



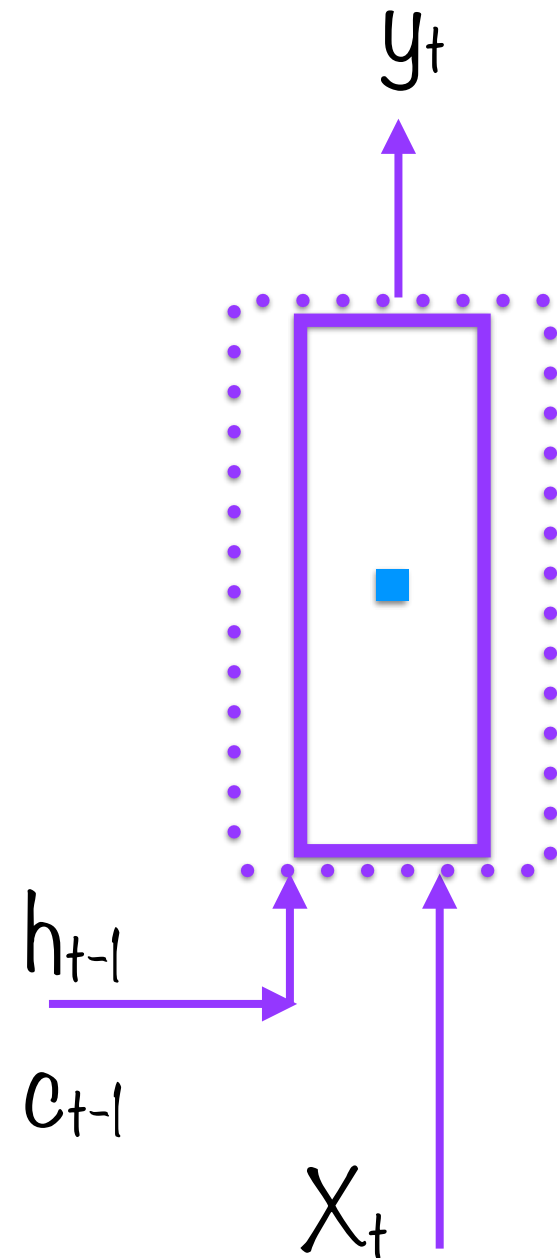
Functionally like basic RNN cell

Performance far better

Amazing success at long-term patterns

Long text sequences, time series...

Variants



Deephole 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

Text as Sequential Data

RNNs are great at learning sequential data

Text is Sequential Data



Predict the next word in a sequence (autocomplete)

“The tallest building in the world is ...”



Language translations

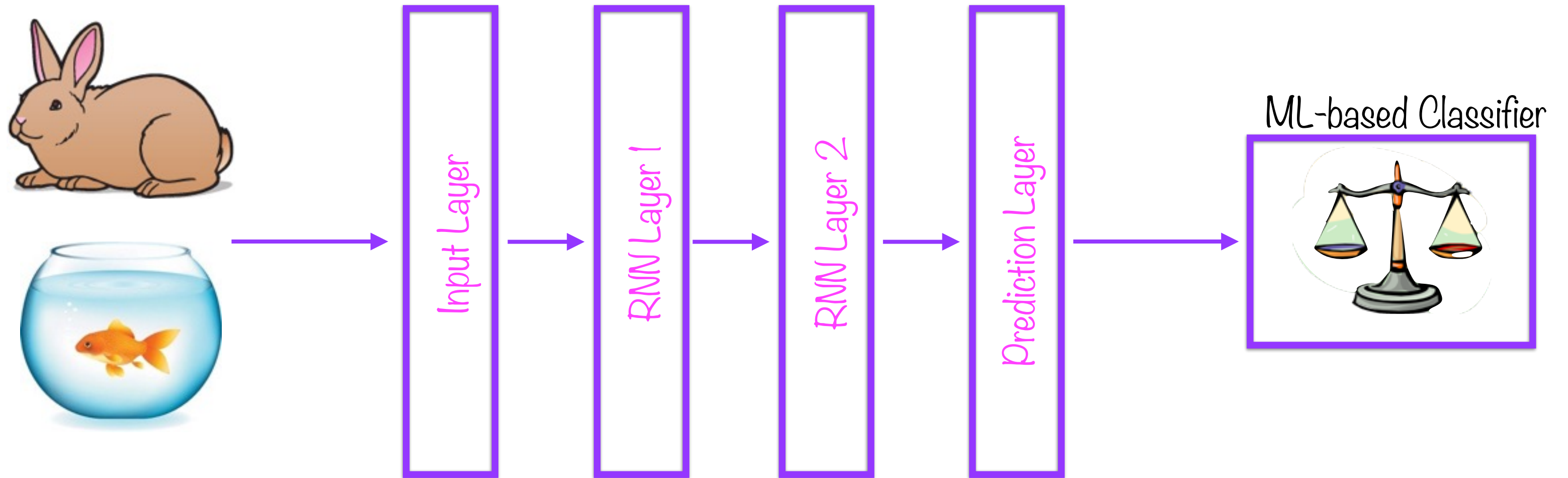
“how are you” -> “Comment allez-vous”



Text classification, sentiment analysis, natural language processing

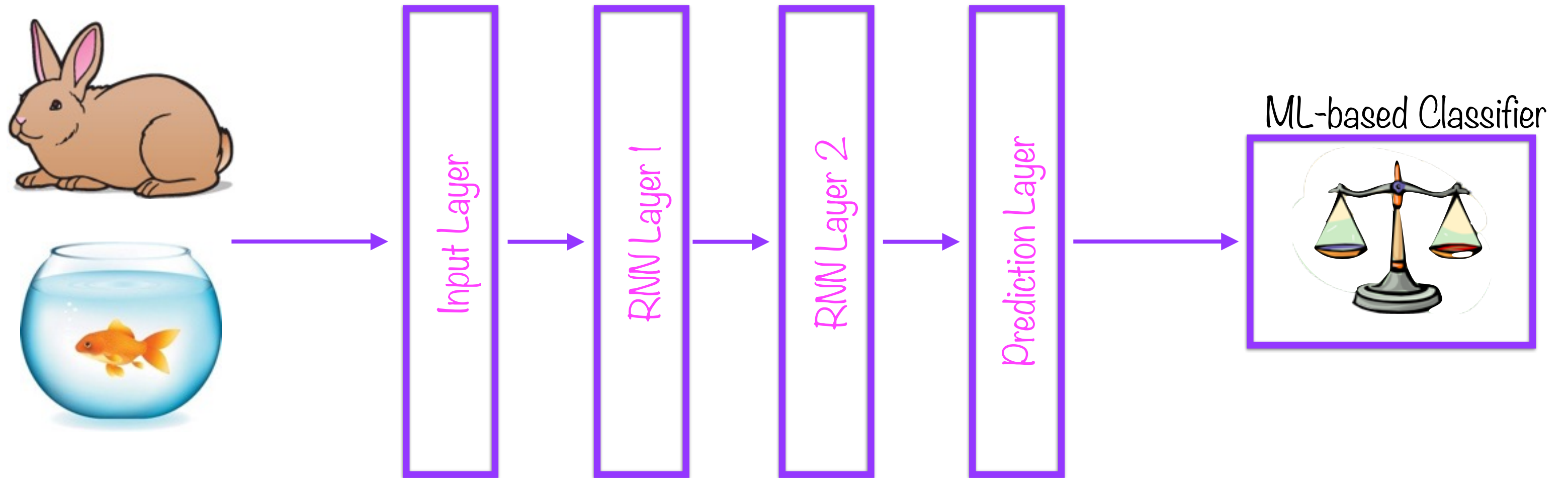
“This is not the worst restaurant not by a long way”

A Recurrent Neural Network



A neural network works on numeric data

A Recurrent Neural Network



How would you represent text as numbers in a meaningful manner?

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

Document as Word Sequence

Here d is a document, represented as a tensor of word encodings

$d =$ “This is not the worst restaurant in the metropolis, not by a long way”

Text as Sequential Data

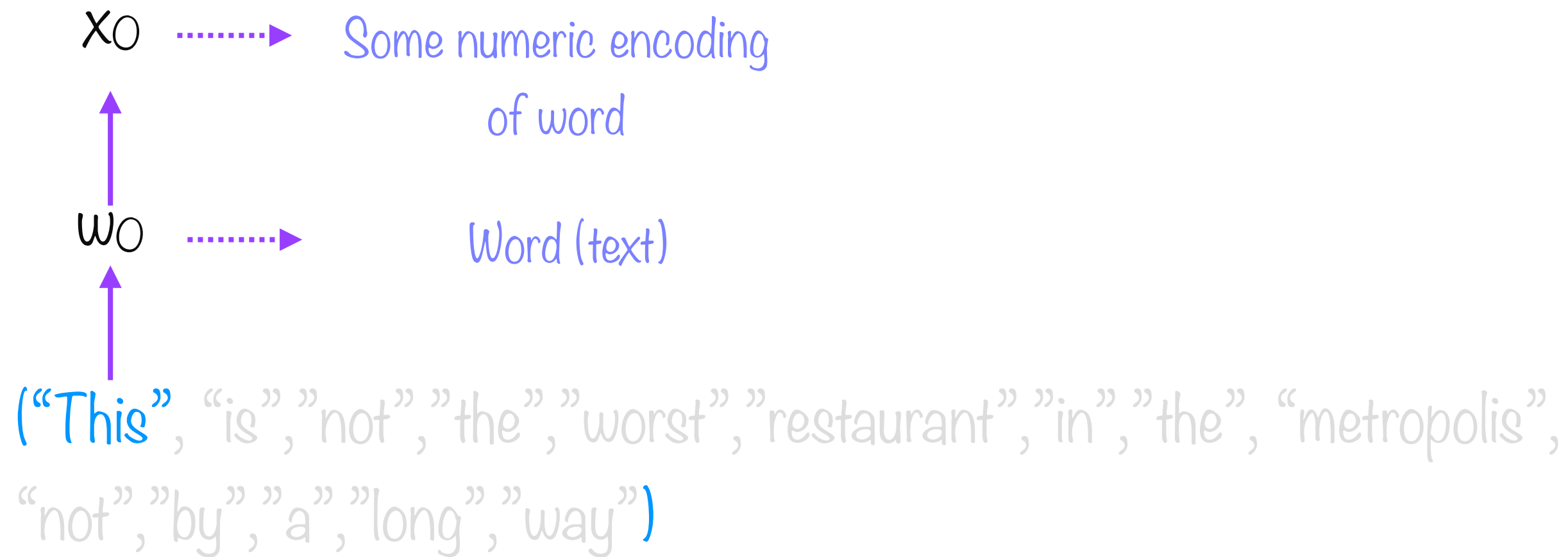
Model a document as an ordered sequence of words

$d =$ “This is not the worst restaurant in the metropolis, not by a long way”

(“This”, “is”, “not”, “the”, “worst”, “restaurant”, “in”, “the”, “metropolis”, “not”, “by”, “a”, “long”, “way”)

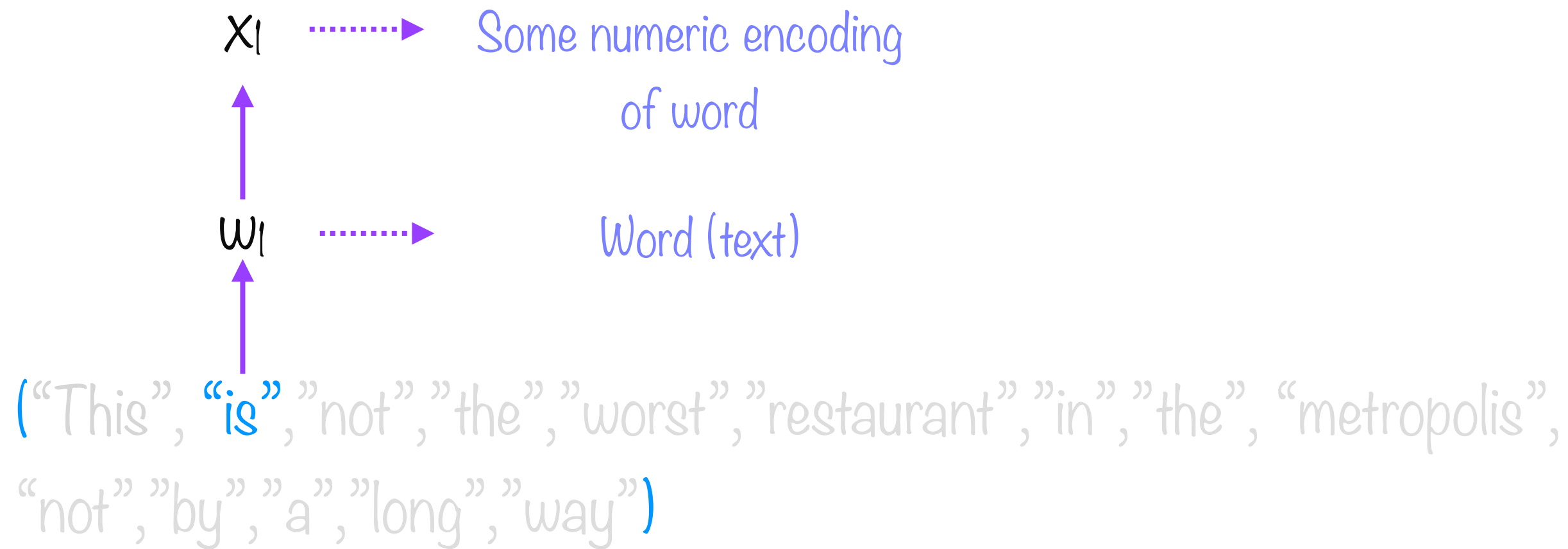
Document as Word Sequence

Tokenise document and feed into RNN



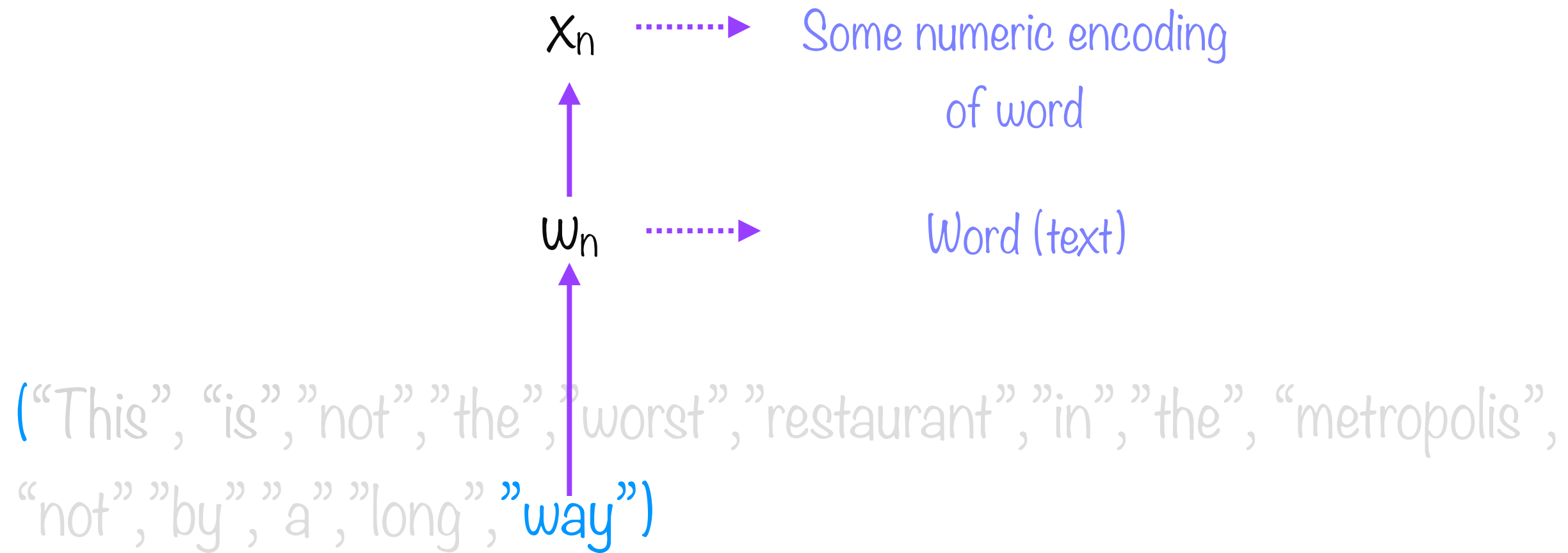
Document as Word Sequence

Tokenise document and feed into RNN



Document as Word Sequence

Tokenise document and feed into RNN



Document as Word Sequence

Tokenise document and feed into RNN

$$d = [x_0, x_1, \dots, x_n]$$

Document as Tensor

Represent each word as numeric data, aggregate into tensor

$$x_i = [?]$$

The Big Question

How best can words be represented as numeric data?

$d = [[?], [?], \dots [?]]$

The Big Question

How best can words be represented as numeric data?

Generating Text Embeddings

One-hot

TF-IDF

Word Embeddings

Generating Text Embeddings

One-hot

TF-IDF

Word Embeddings

Documents and Corpus

Reviews

Amazing!
Worst movie ever
Two thumbs up
Part 2 was bad, 3 the worst
Up there with the greats

D = Entire corpus

d_i = One document in corpus

One-hot Encoding

Reviews

Amazing!
Worst movie ever
Two thumbs up
Part 2 was bad, 3 the worst
Up there with the greats

All Words

amazing
worst
movie
ever
two
thumbs
up
Part
was
bad
3
the
there
with
greats

Create a set of all words (all across the corpus)

One-hot Encoding

	Amazing!	Worst movie ever	Two thumbs up
amazing	1	0	0
worst	0	1	1
movie	0	1	1
ever	0	1	1
two	0	0	1
thumbs	0	0	1
up	0	0	1
Part	0	0	0
was	0	0	0
bad	0	0	0
3	0	0	0
the	0	0	0
there	0	0	0
with	0	0	0
greats	0	0	0

Express each review as a tuple of 1,0 elements

One-hot Encoding

Reviews

Amazing!
Worst movie ever
Two thumbs up
Part 2 was bad, 3 the worst
Up there with the greats

Feature Vector

[1,0,0,0,0,0,0 ...]
[0,1,1,1,0,0,0 ...]
[0,0,0,0,1,1,1 ...]
...
...

Now compare, measure distance using simple geometry

Flaws of One-hot Encoding



Large vocabulary - enormous feature vectors

Unordered - Lost all context

Binary - Lost frequency information

Generating Text Embeddings

One-hot

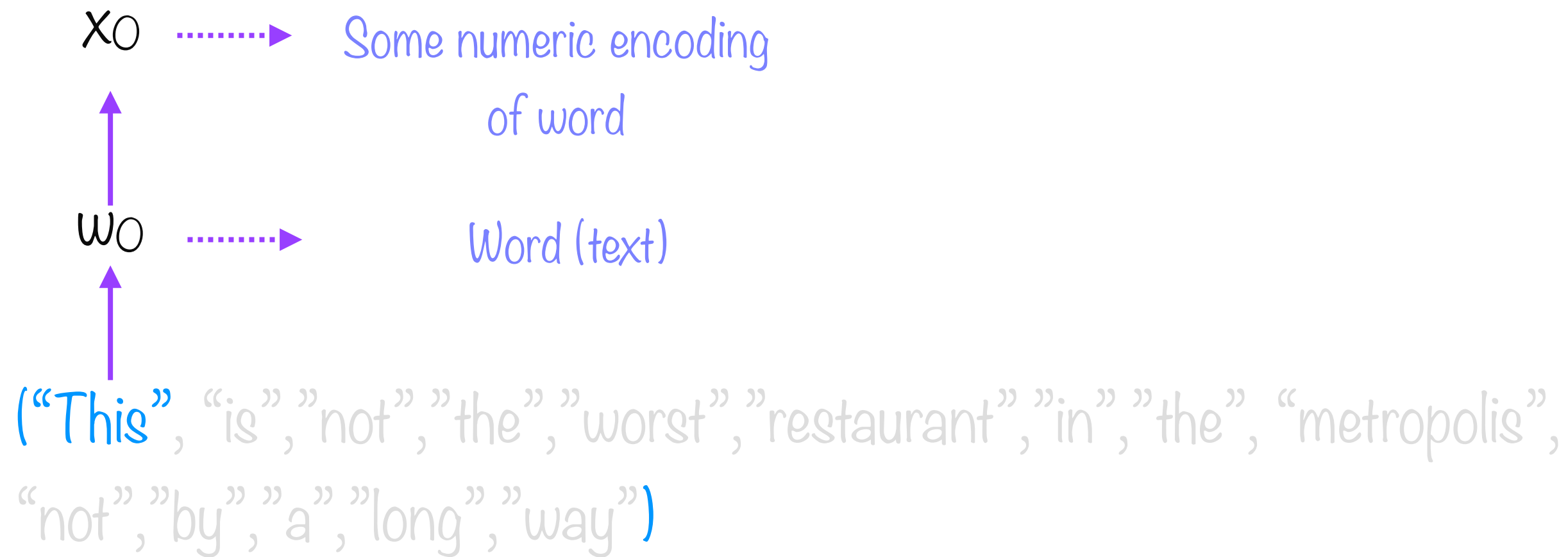
TF-IDF

Word Embeddings

$$d = [x_0, x_1, \dots, x_n]$$

Document as Tensor

Represent each word as numeric data, aggregate into tensor



Document as Word Sequence

Tokenise document and feed into RNN

$$x_i = \text{tf}(w_i) \times \text{idf}(w_i)$$

TF-IDF

Represent each word with the product of two terms - tf and idf

$$x_i = tf(w_i) \times idf(w_i)$$

TF-IDF

TF = Term Frequency; IDF = Inverse Document Frequency

Documents and Corpus

Reviews

Amazing!
Worst movie ever
Two thumbs up
Part 2 was bad, 3 the worst
Up there with the greats

D = Entire corpus

d_i = One document in corpus

$$x_{i,j} = \text{tf}(w_i, d_j) \times \text{idf}(w_i, D)$$

TF-IDF

Encoding of word i in document j depends on word, document and also on entire corpus

$$x_{i,j} = \text{tf}(w_i, d_j) \times \text{idf}(w_i, D)$$

TF = Term Frequency

Measure of how frequently word i occurs in document j

$$x_{i,j} = \text{tf}(w_i, d_j) \times \text{idf}(w_i, D)$$

IDF = Inverse Document Frequency

Measure of how **infrequently** word i occurs in corpus D

$$x_{i,j} = \text{tf}(w_i, d_j) \times \text{idf}(w_i, D)$$

TF-IDF

High weight for word i in document j if word occurs a lot in this document, but rarely elsewhere

Evaluating TF-IDF



Important advantages

- Feature vector much more tractable in size
- Frequency and relevance captured

One big drawback

- Context still not captured

Generating Text Embeddings

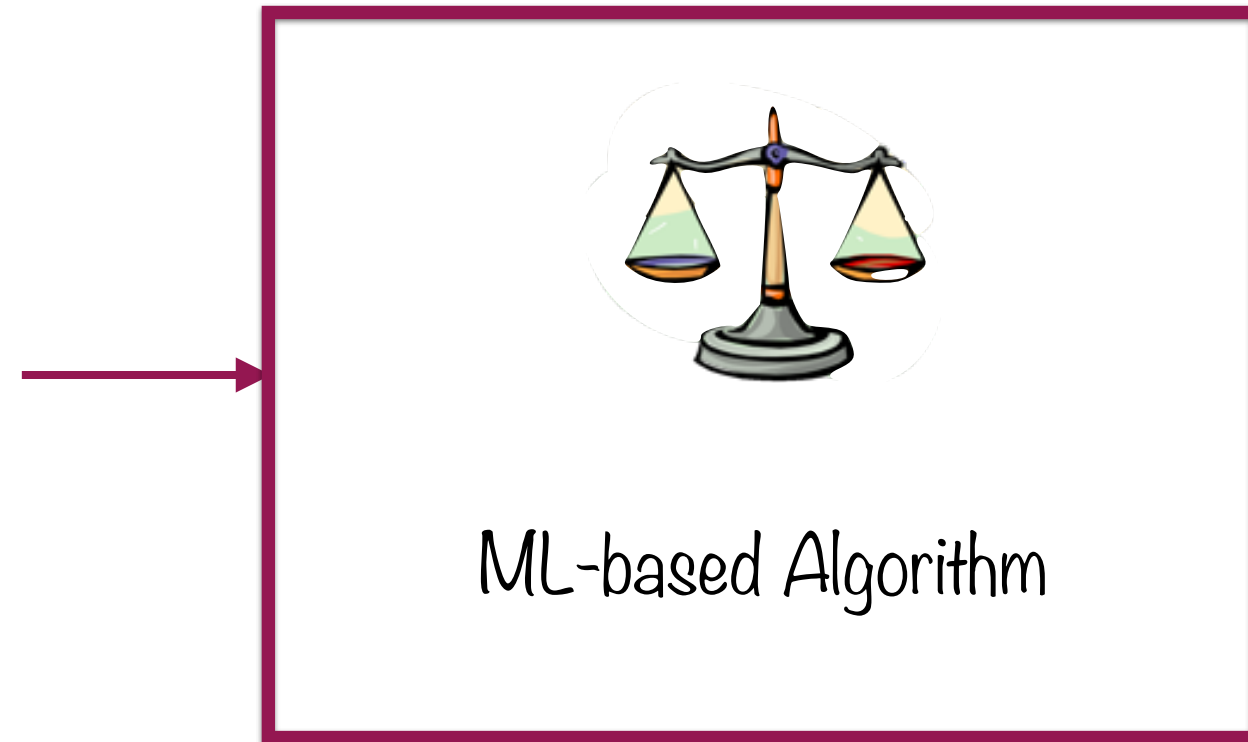
One-hot

TF-IDF

Word Embeddings

Word Embeddings

“London”



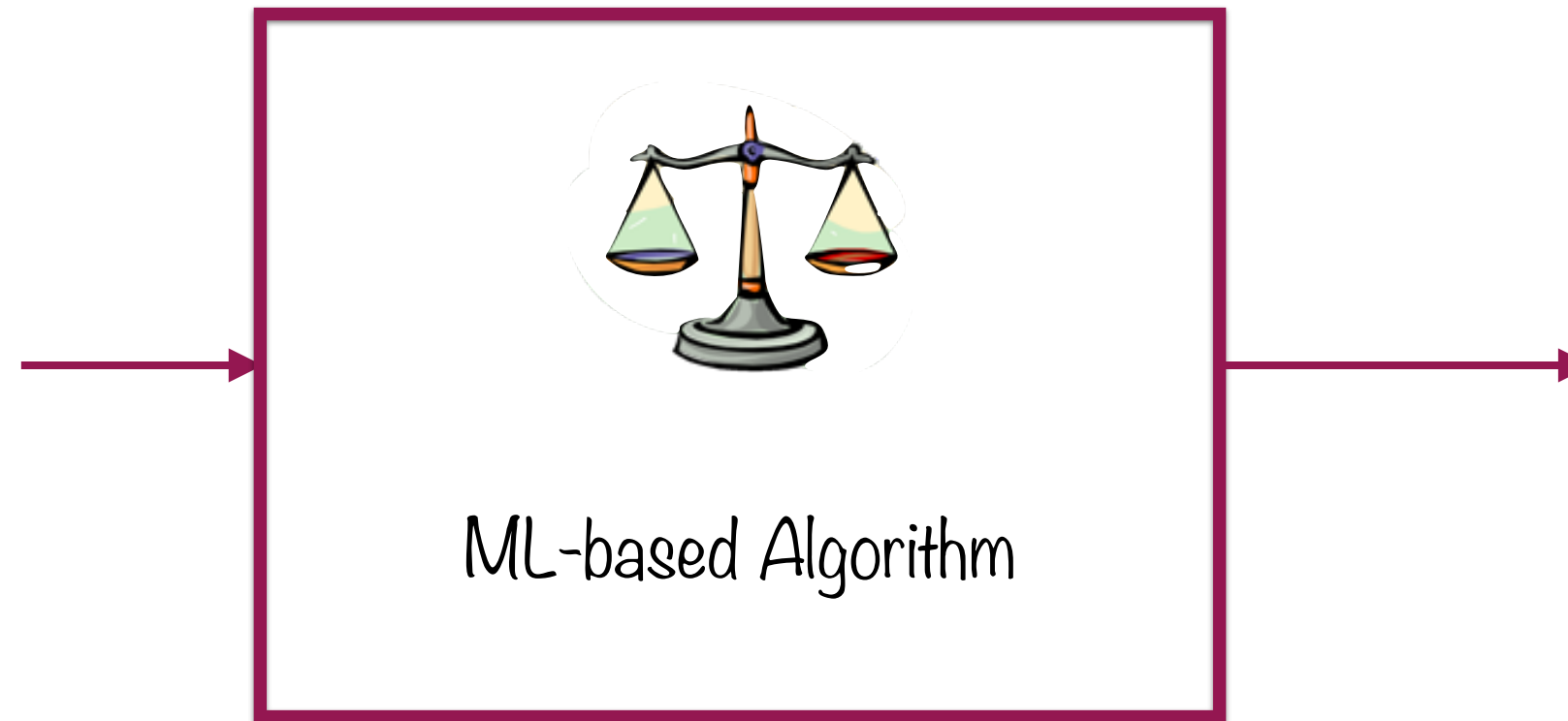
[
“345698000”,
]



Large corpus

Word Embeddings

[
“New York”,
“Paris”,
“London”
]

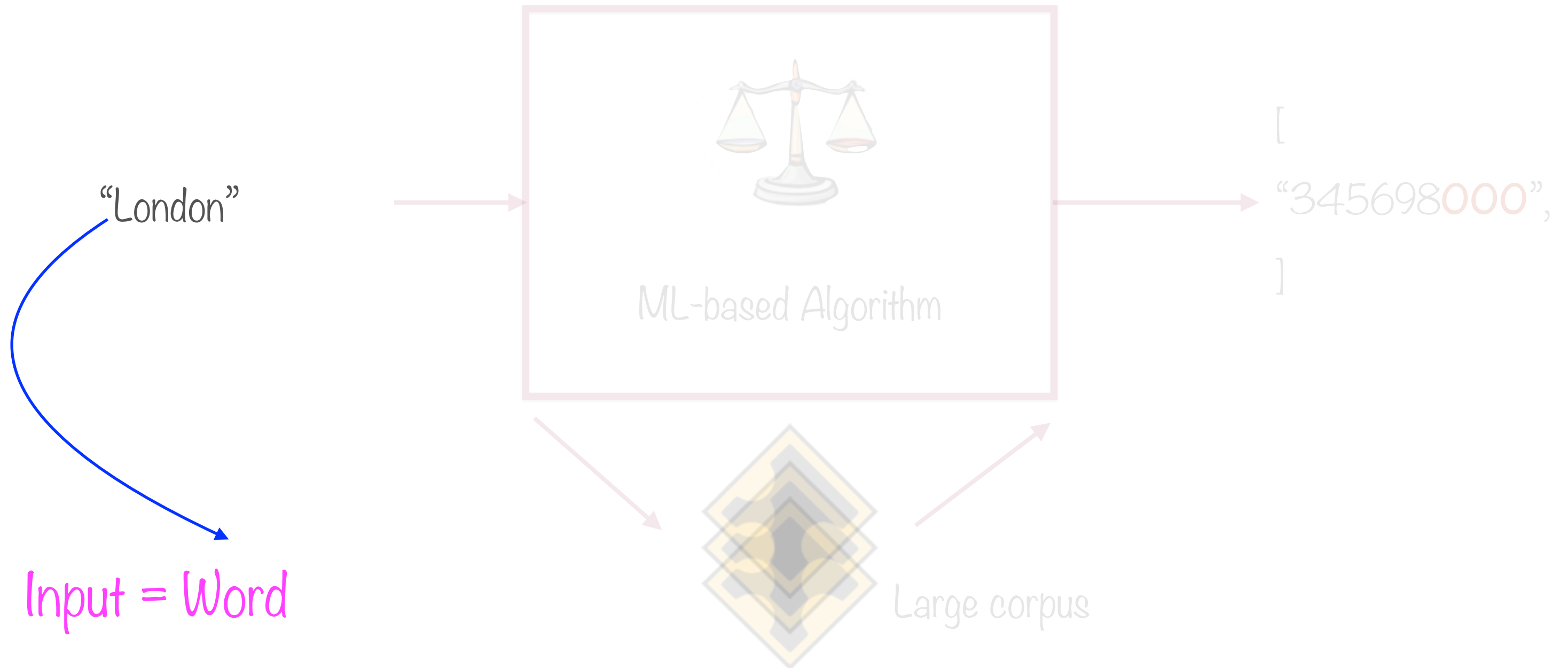


[
“345698234”,
“345698100”,
“345698000”,
]

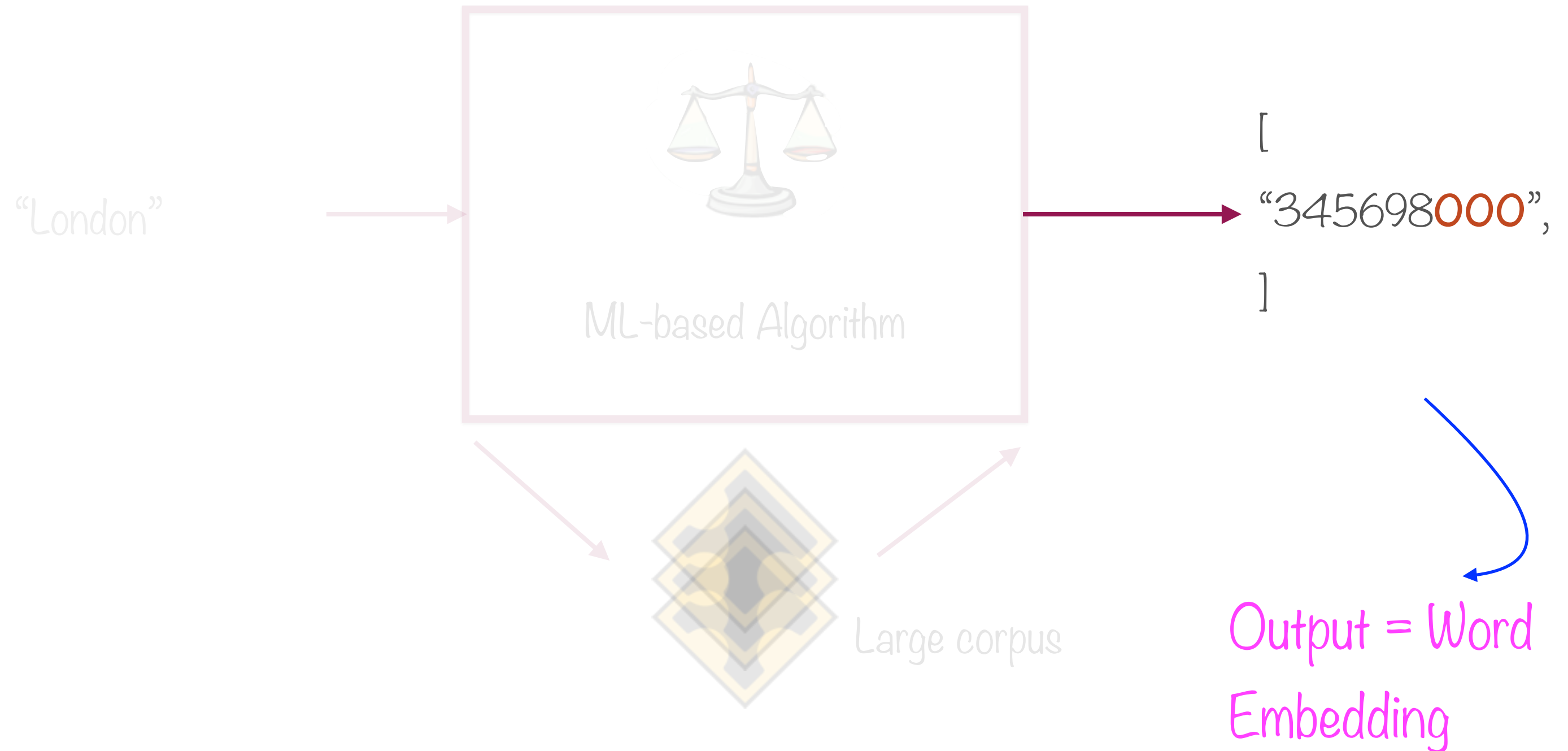


Large corpus

Word Embeddings



Word Embeddings



Word Embeddings

“London”



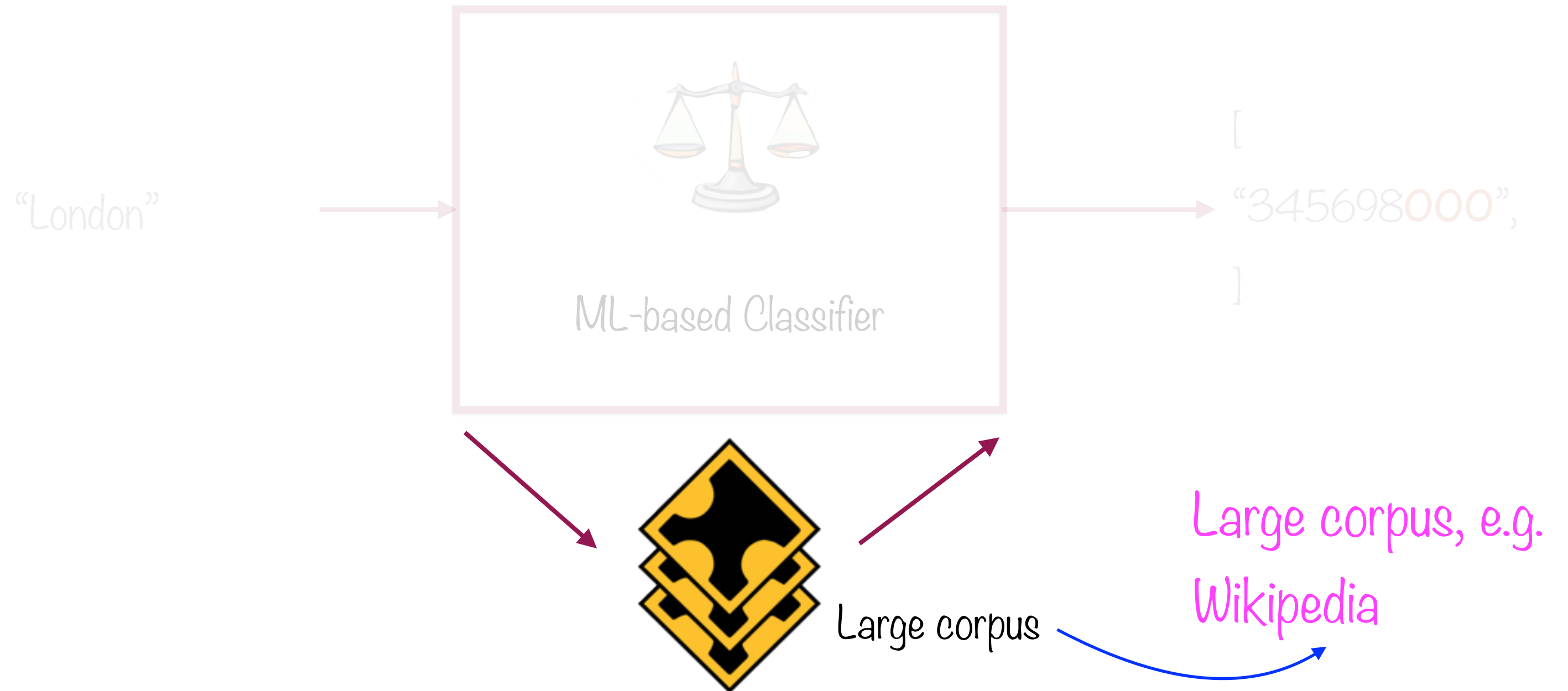
Very low
dimensionality

[
“345698000”,
]

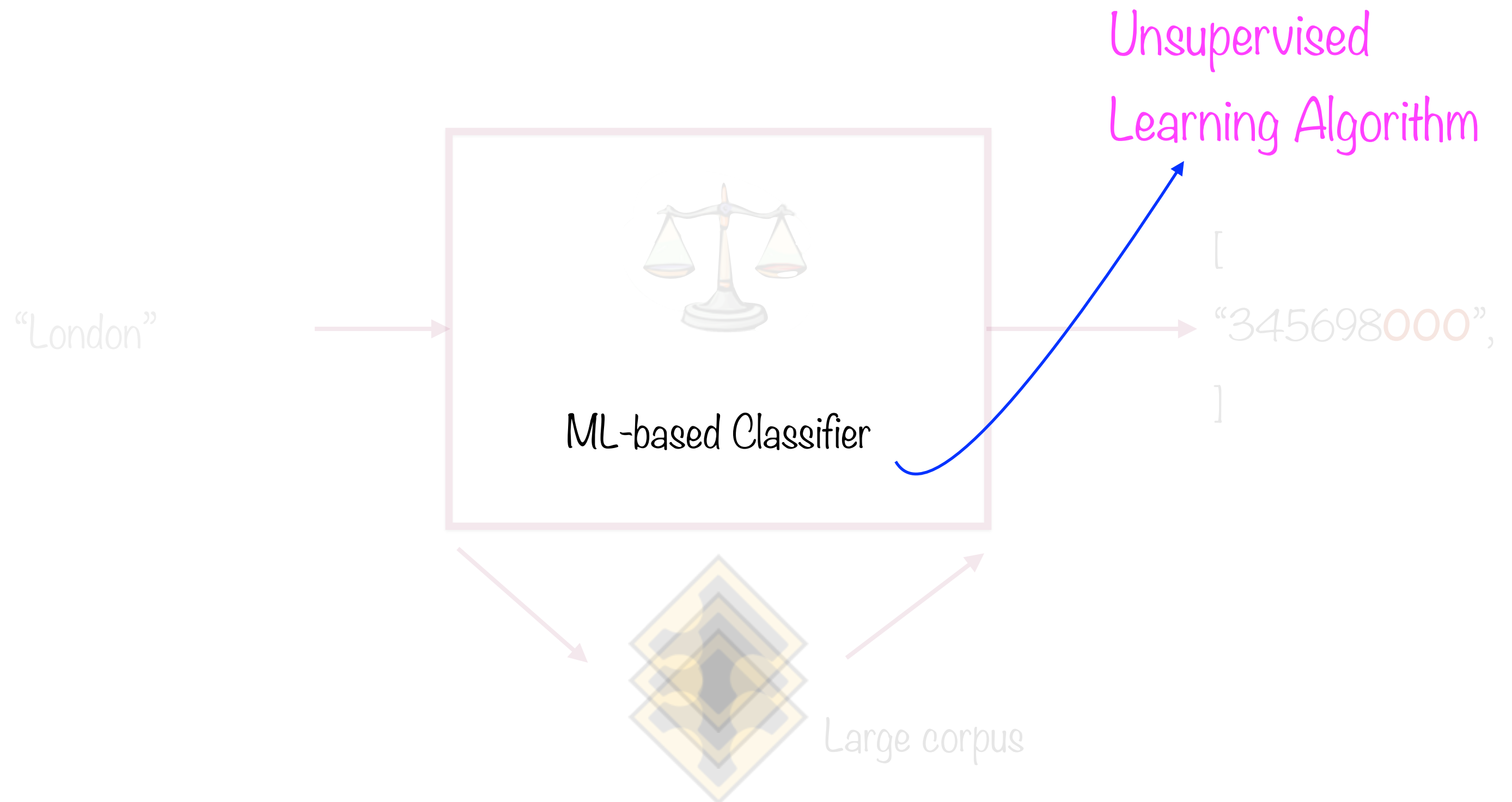


Large corpus

Word Embeddings



Word Embeddings



Magic



Word embeddings capture meaning

“Queen” ~ “King” + “Woman” - “Man”

“Paris” ~ “France” + “London” - “England”

Dramatic dimensionality reduction

Word2Vec



Most popular word embedding model

Mikolov (Google), 2013

Use simple NN (not deep) to learn embeddings

GloVe



Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning,
(Stanford) 2014

Uses word-word co-occurrence matrix, nearest-neighbors for
word relationships

Documents and Corpus

Reviews

Amazing!
Worst movie ever
Two thumbs up
Part 2 was bad, 3 the worst
Up there with the greats

Consider all the words in the entire corpus - this is the
vocabulary

Split into Words

Reviews

Amazing!
Worst movie ever
Two thumbs up
Part 2 was bad, 3 the worst
Up there with the greats

All Words

amazing
worst
movie
ever
two
thumbs
up
Part
was
bad
3
the
there
with
greats

Create a set of all words (all across the corpus)

Assign Unique Identifiers

amazing	1
worst	2
movie	3
ever	4
two	5
thumbs	6
up	7
Part	8
was	9
bad	10
3	11
the	12
there	13
with	14
greats	15

Each word has a unique **numeric** identifier

$d =$ “This is not the worst restaurant in the metropolis, not by a long way”

$d = [1, 2, \textcircled{3}, 4, 5, 6, 7, 4, 8, \textcircled{3}, 9, 10, 11, 12]$

Vector of Unique Identifiers

Convert Identifiers to Encodings

1	123243
2	492584
3	254698
4	885412
5	524763
6	148248
7	...
8	...
9	
10	
11	
12	
13	
14	
15	

Each word id is encoded into vectors of the same
dimensionality