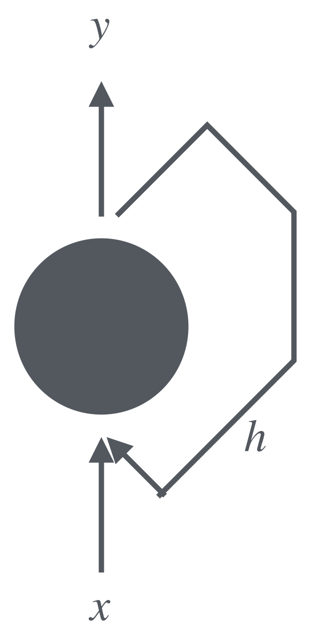
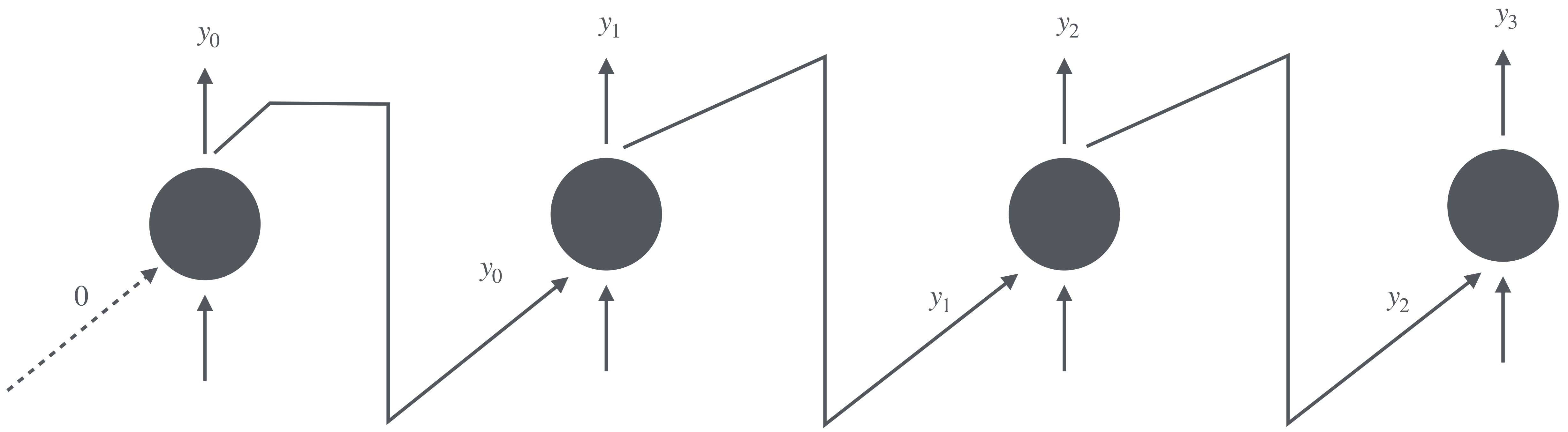
RNN , CNN

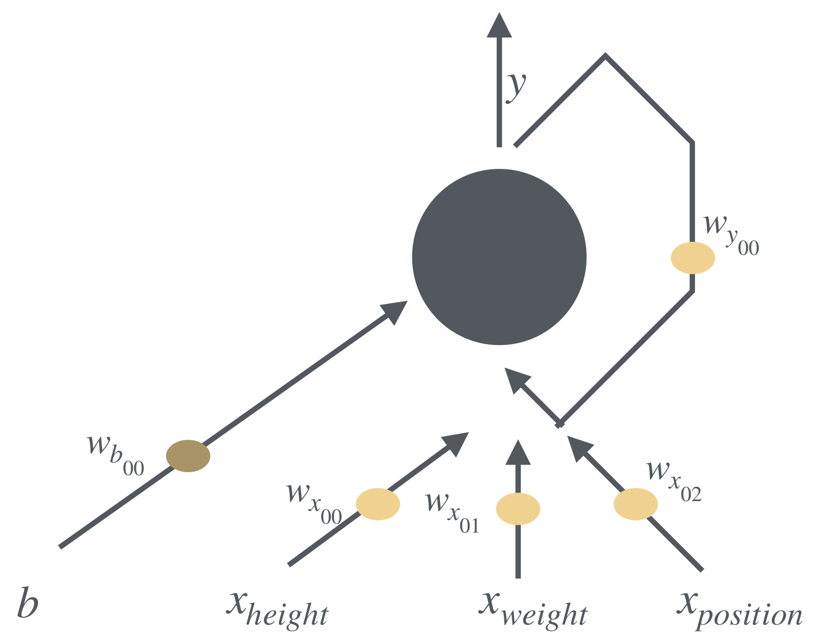
Recurrent Neuron



$x = [height \;\; weight \;\; position]$

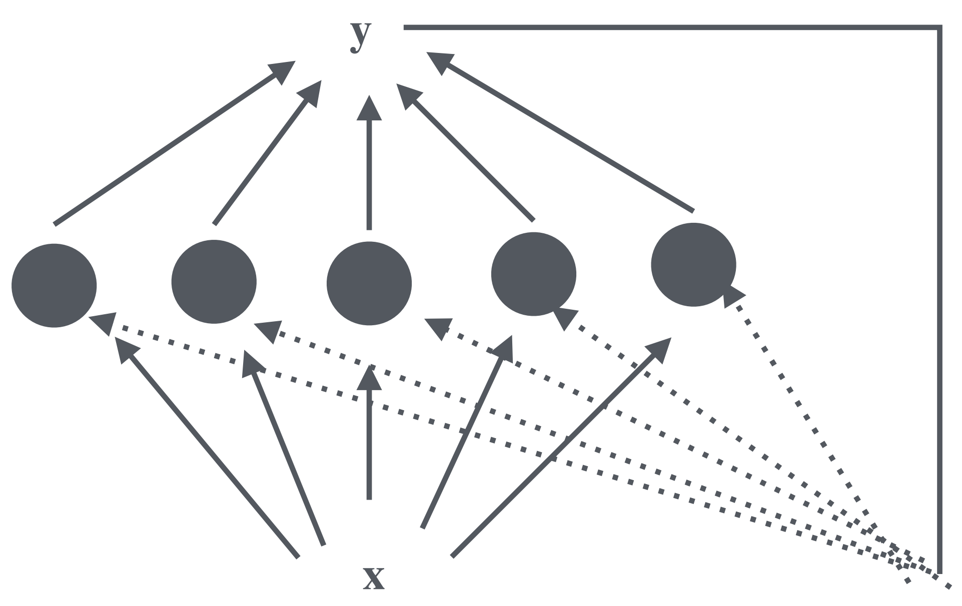


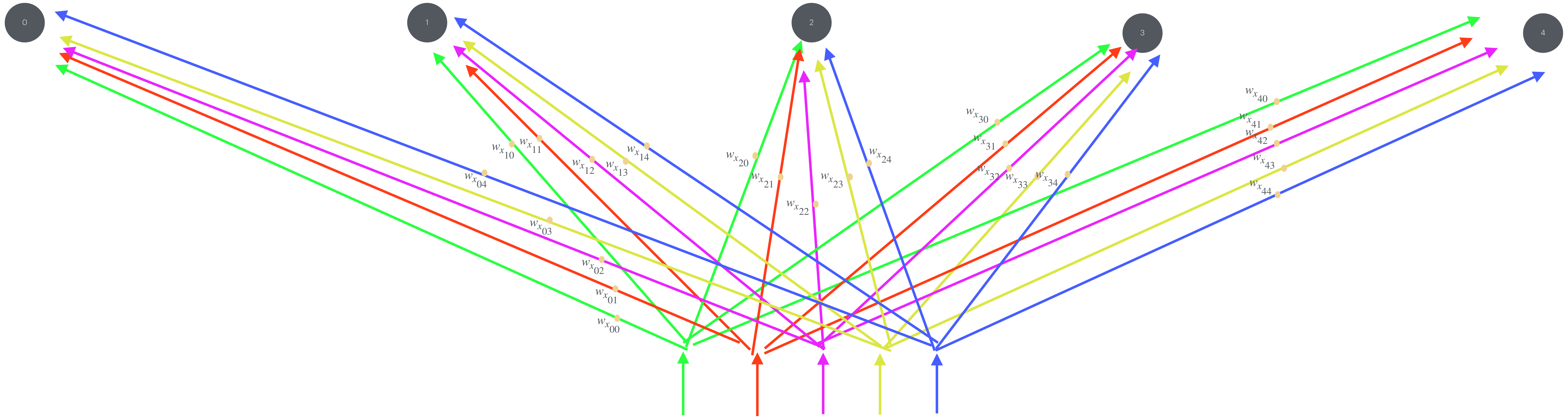
$x_{t=0} = [height \;\; weight \;\; position]$ $x_{t=1} = [height \;\; weight \;\; position]$ $x_{t=2} = [height \;\; weight \;\; position]$ $x_{t=3} = [height \;\; weight \;\; position]$



X	Wx	Y	Wy	b
x_height	wx_00	y	wy_00	wb_00
x_weight	wx_01			
x_position	wx_02			

Recurrent Neuron Layer





$$x_{t=0} = [height_0 \quad weight_1 \quad position_2 \quad date_3 \quad score_4]$$

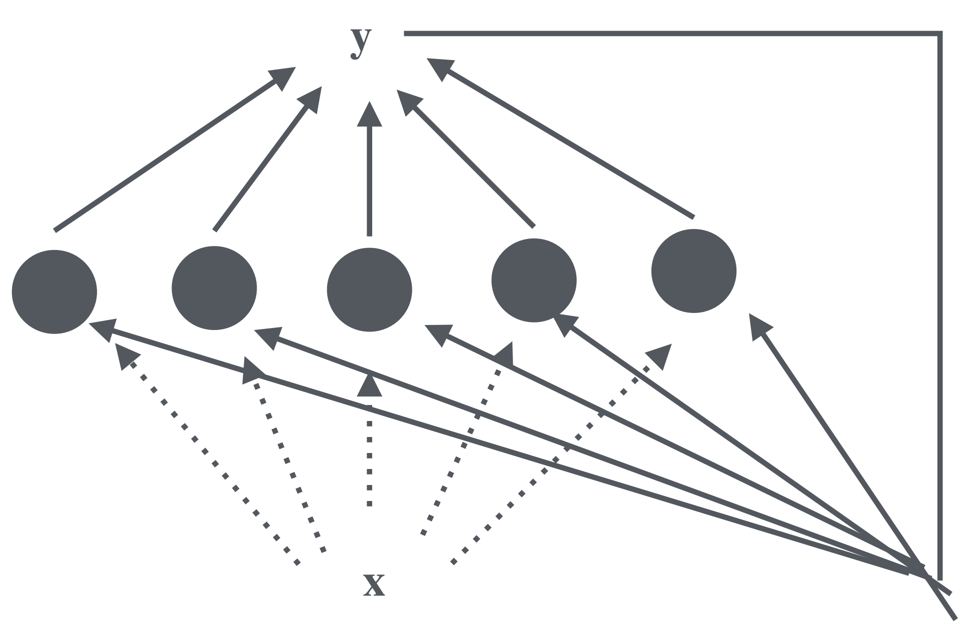
Wx

w_00	w_01	w_02	w_03	w_04
w_10	w_11	w_12	w_13	w_14
w_20	w_21	w_22	w_23	w_24
w_30	w_31	w_32	w_33	w_34
w_40	w_41	w_42	w_43	w_44

X

x_height
x_weight
x_position
x_date
x_score

Recurrent Neuron Layer

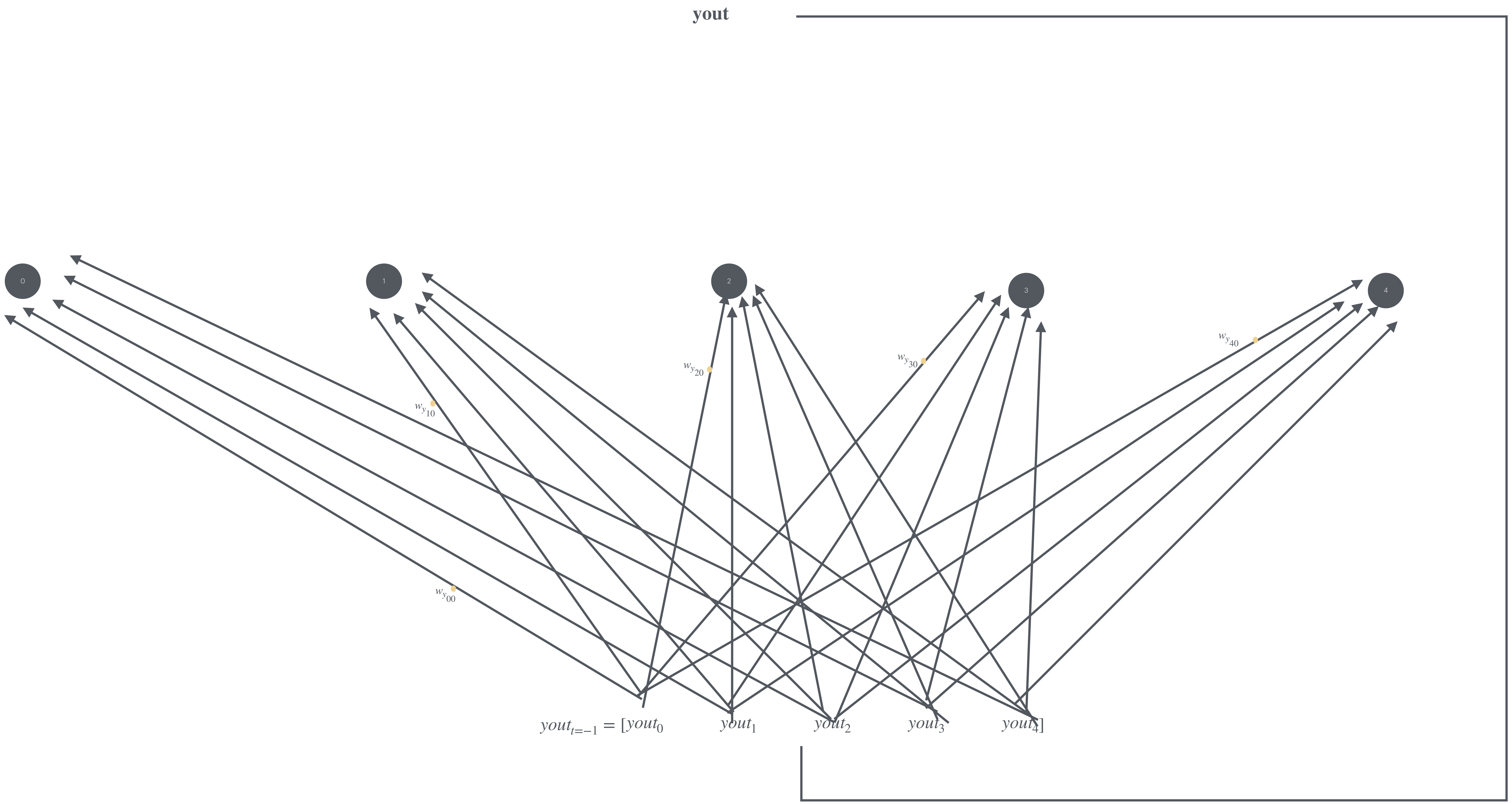


Wy

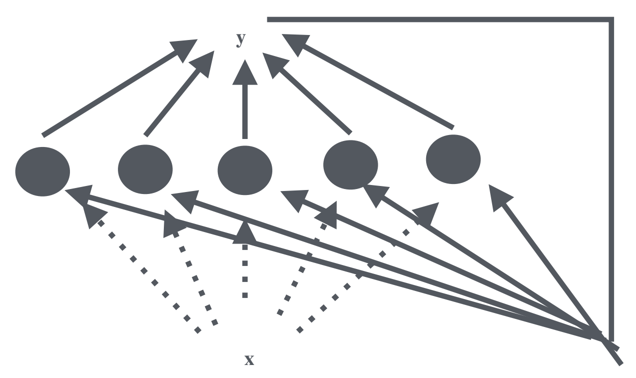
wy_00	wy_01	wy_02	wy_03	wy_04
wy_10	wy_11	wy_12	wy_13	wy_14
wy_20	wy_21	wy_22	wy_23	wy_24
wy_30	wy_31	wy_32	wy_33	wy_34
wy_40	wy_41	wy_42	wy_43	wy_44

Y

yout_0
yout_1
yout_2
yout_3
yout_4



Recurrent Neuron Layer



height_weights

W_x

w_00	w_01	w_02	w_03	w_04
w_10	w_11	w_12	w_13	w_14
w_20	w_21	w_22	w_23	w_24
w_30	w_31	w_32	w_33	w_34
w_40	w_41	w_42	w_43	w_44

height_weights

W_t^x

w_00	w_10	w_20	w_30	w_40
w_01	w_11	w_21	w_31	w_41
w_02	w_12	w_22	w_32	w_42
w_03	w_13	w_23	w_33	w_43
w_04	w_14	w_24	w_34	w_44

X

x_height
x_weight
x_position
x_date
x_score

X^T

x_height	x_weight	x_position	x_date	x_score

$yout_0\ weights$

W_y

wy_00	wy_01	wy_02	wy_03	wy_04
wy_10	wy_11	wy_12	wy_13	wy_14
wy_20	wy_21	wy_22	wy_23	wy_24
wy_30	wy_31	wy_32	wy_33	wy_34
wy_40	wy_41	wy_42	wy_43	wy_44

$yout_0\ weights$

W_y^t

wy_00	wy_10	wy_20	wy_30	wy_40
wy_01	wy_11	wy_21	wy_31	wy_41
wy_02	wy_12	wy_22	wy_32	wy_42
wy_03	wy_13	wy_23	wy_33	wy_43
wy_04	wy_14	wy_24	wy_34	wy_44

Y

yout_0
yout_1
yout_2
yout_3
yout_4

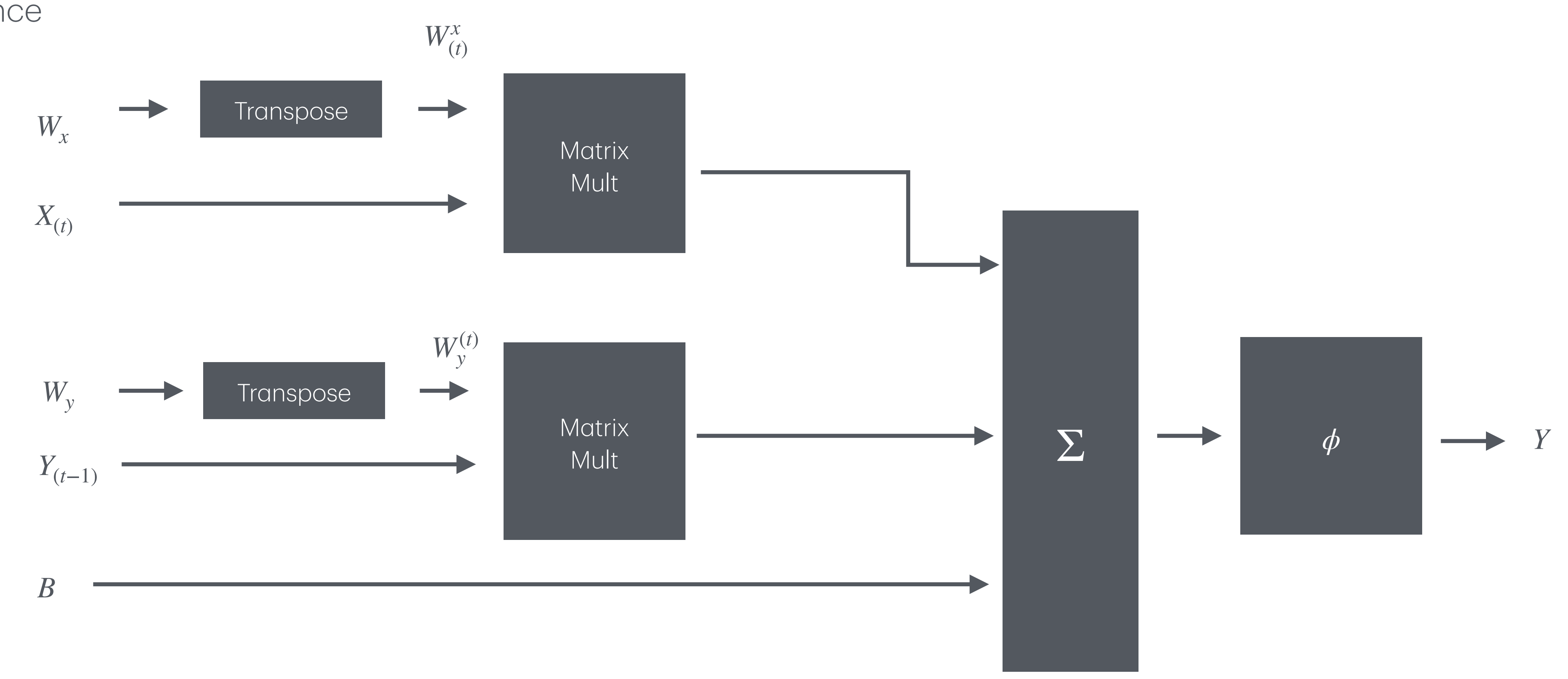
B

bias_0
bias_1
bias_2
bias_3
bias_4

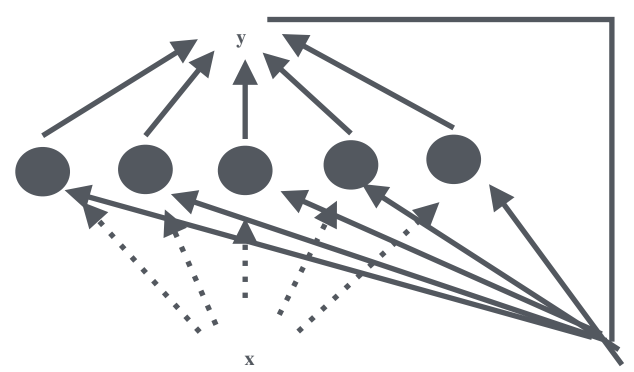
Y^T

yout_0	yout_1	yout_2	yout_3	yout_4

Single Instance



Recurrent Neuron Layer



W_x

w_00	w_01	w_02	w_03	w_04
w_10	w_11	w_12	w_13	w_14
w_20	w_21	w_22	w_23	w_24
w_30	w_31	w_32	w_33	w_34
w_40	w_41	w_42	w_43	w_44

W_t^x

w_00	w_10	w_20	w_30	w_40
w_01	w_11	w_21	w_31	w_41
w_02	w_12	w_22	w_32	w_42
w_03	w_13	w_23	w_33	w_43
w_04	w_14	w_24	w_34	w_44

X

x_height
x_weight
x_position
x_date
x_score

X^T

x_height	x_weight	x_position	x_date	x_score

W_y

wy_00	wy_01	wy_02	wy_03	wy_04
wy_10	wy_11	wy_12	wy_13	wy_14
wy_20	wy_21	wy_22	wy_23	wy_24
wy_30	wy_31	wy_32	wy_33	wy_34
wy_40	wy_41	wy_42	wy_43	wy_44

W_y^t

wy_00	wy_10	wy_20	wy_30	wy_40
wy_01	wy_11	wy_21	wy_31	wy_41
wy_02	wy_12	wy_22	wy_32	wy_42
wy_03	wy_13	wy_23	wy_33	wy_43
wy_04	wy_14	wy_24	wy_34	wy_44

Y

yout_0
yout_1
yout_2
yout_3
yout_4

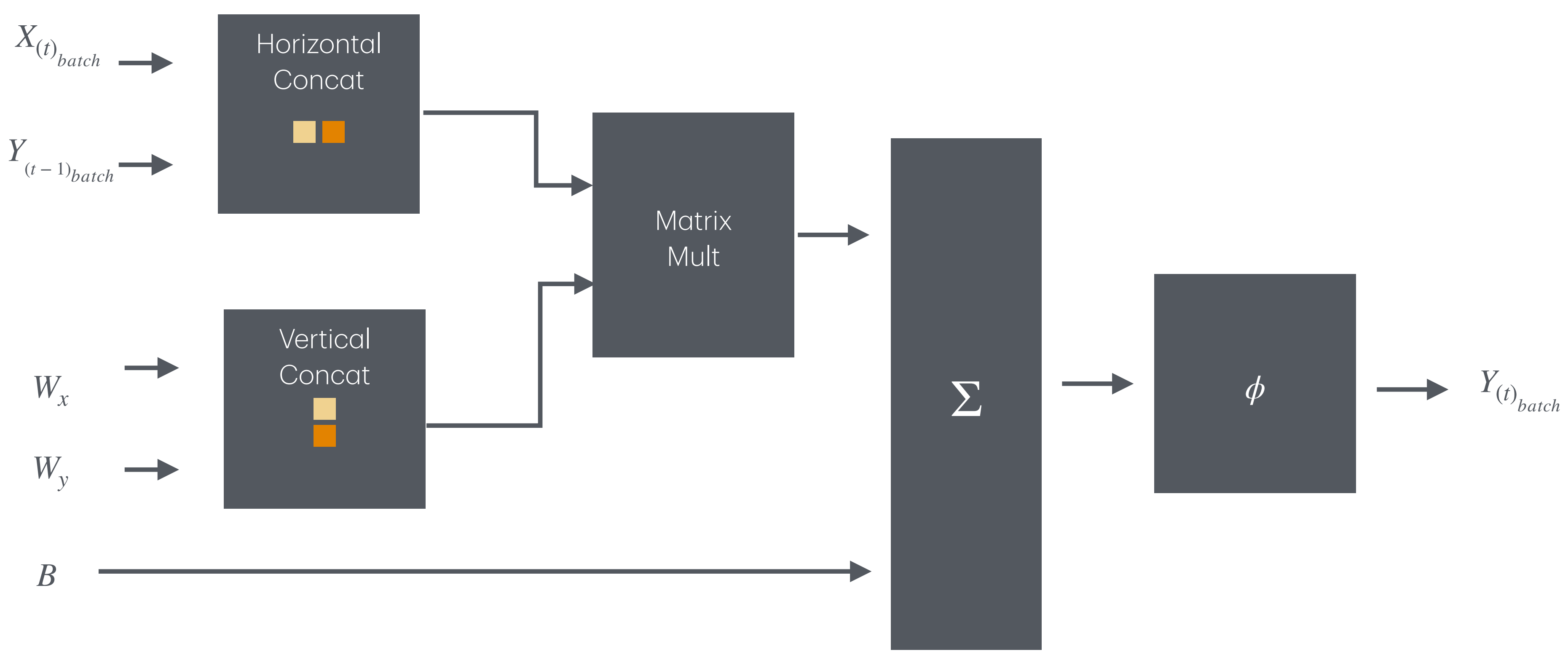
Y^T

yout_0	yout_1	yout_2	yout_3	yout_4

B

bias_0
bias_1
bias_2
bias_3
bias_4

Batch of Instances



$instance_0$

x_height	x_weight	x_position	x_date	x_score	yout_0	yout_1	yout_2	yout_3	yout_4

$instance_1$

⋮

w_00	w_01	w_02	w_03	w_04
w_10	w_11	w_12	w_13	w_14
w_20	w_21	w_22	w_23	w_24
w_30	w_31	w_32	w_33	w_34
w_40	w_41	w_42	w_43	w_44
wy_00	wy_01	wy_02	wy_03	wy_04
wy_10	wy_11	wy_12	wy_13	wy_14
wy_20	wy_21	wy_22	wy_23	wy_24
wy_30	wy_31	wy_32	wy_33	wy_34
wy_40	wy_41	wy_42	wy_43	wy_44

height_weights

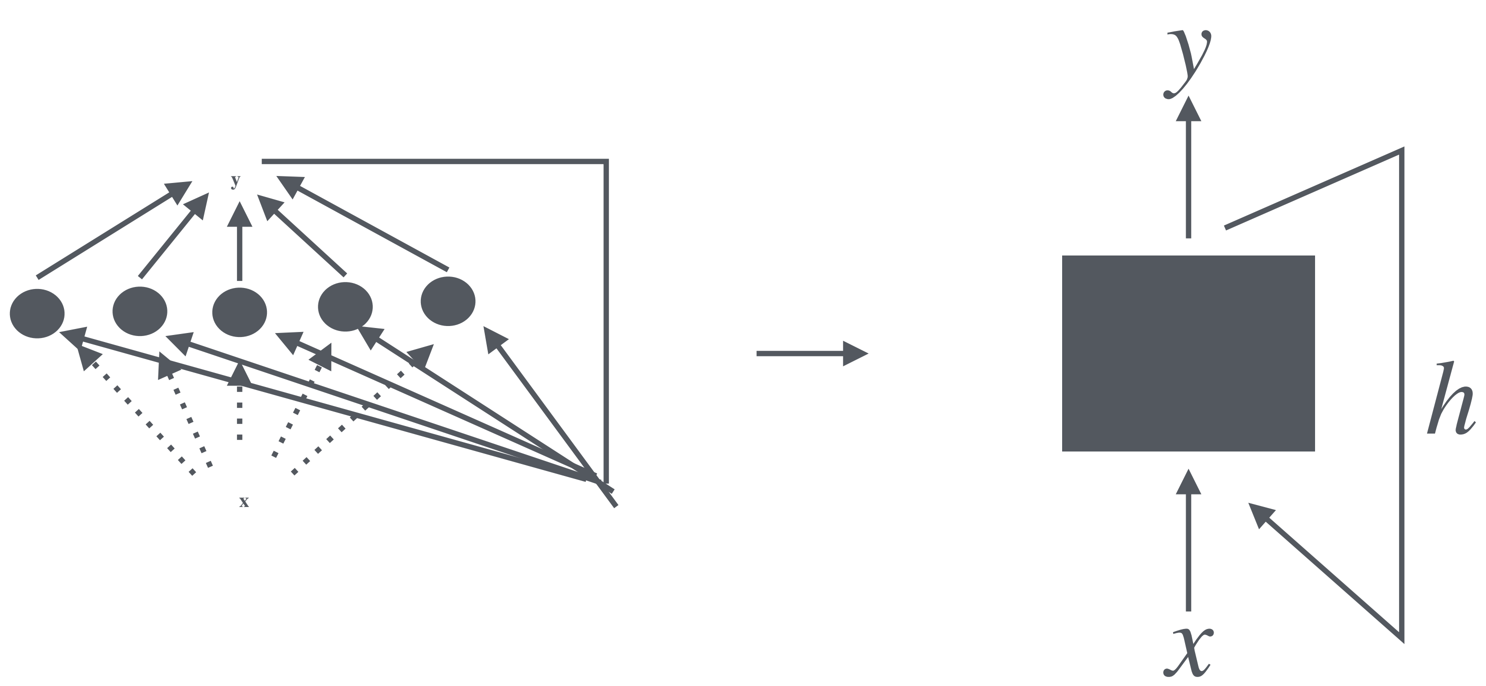
$MatrixOperation_{instance_0}$

$MatrixOperation_{instance_1}$

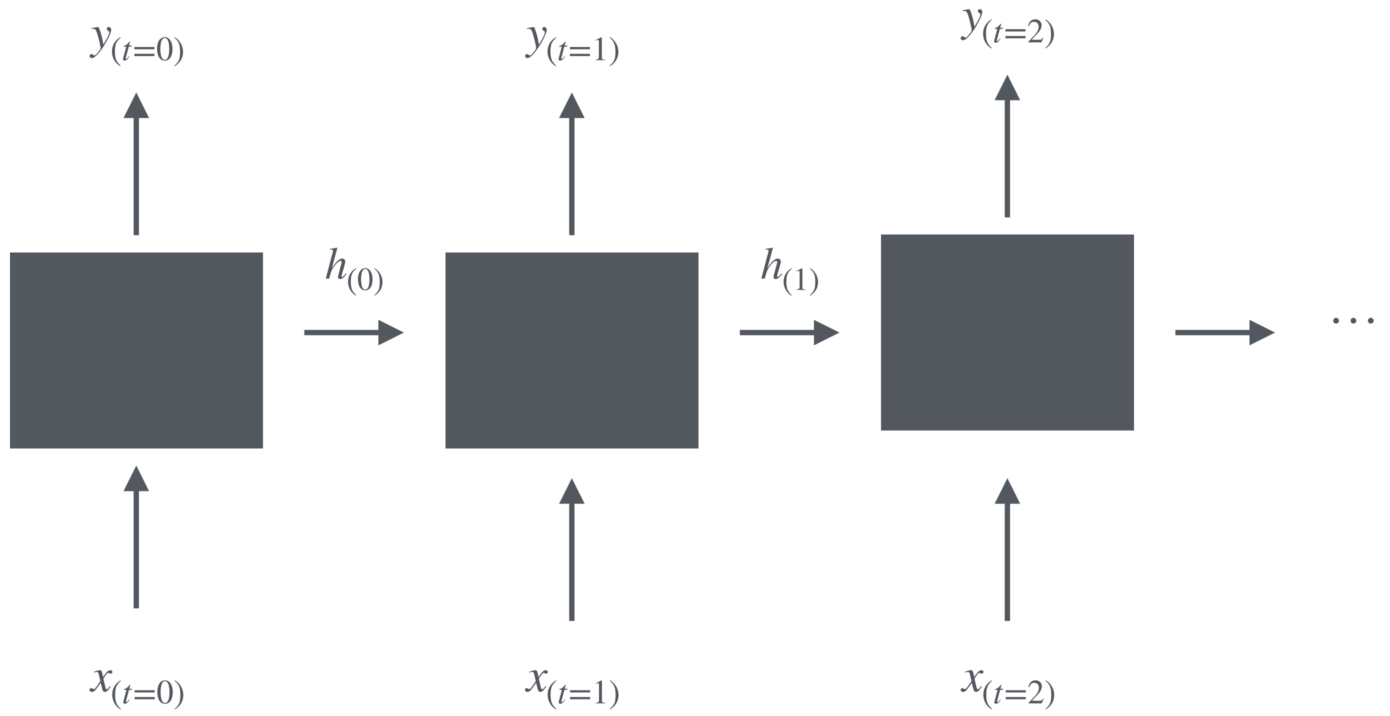
⋮

Memory Cell

Recurrent neuron is a form of memory with state (i.e. h)



State variable may not always be equal to output



Output is function of previous state and input

Each state is a function of previous output and input

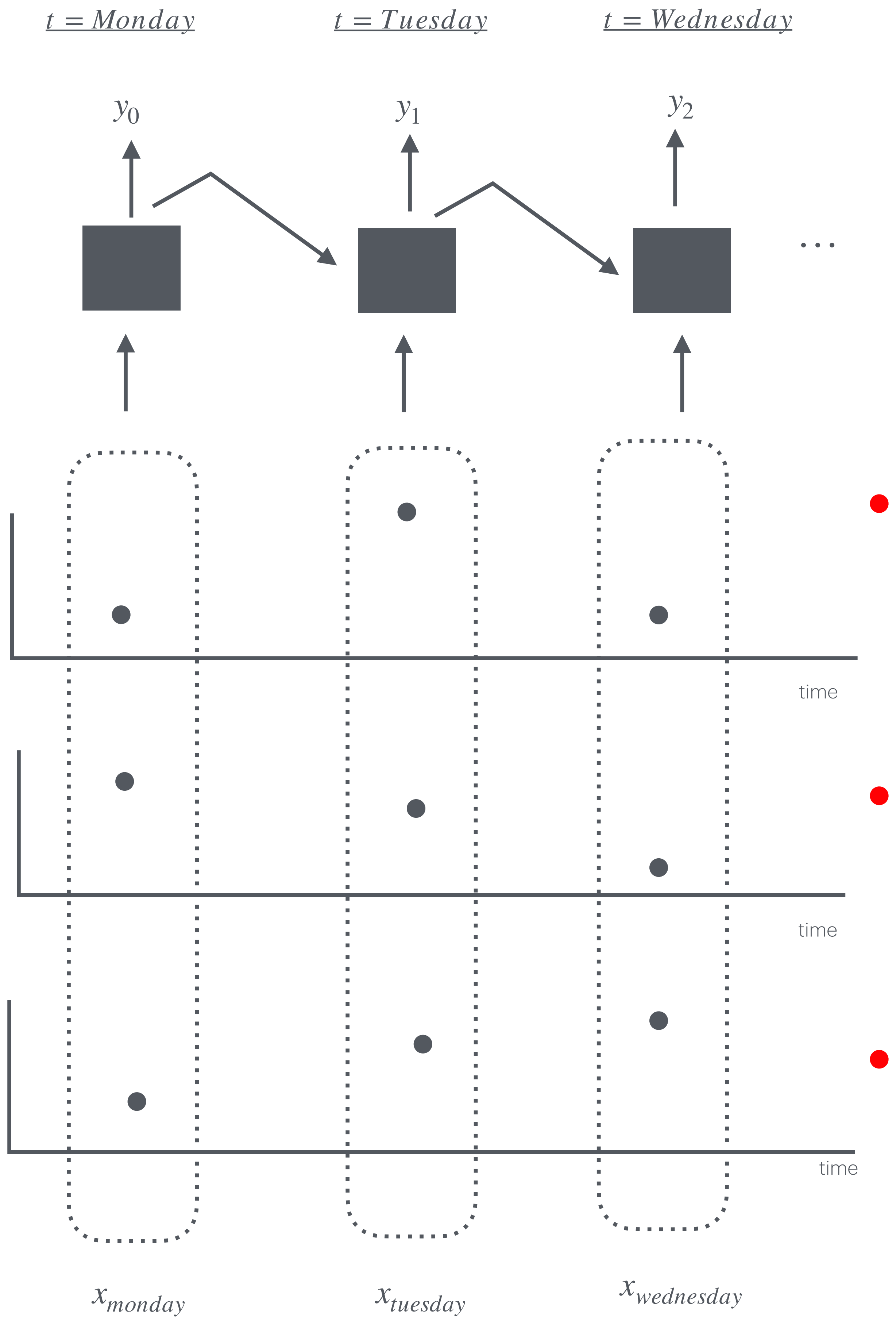
RNN

Simultaneously takes input and produces output

Sequence to Vector

Sequence to Vector

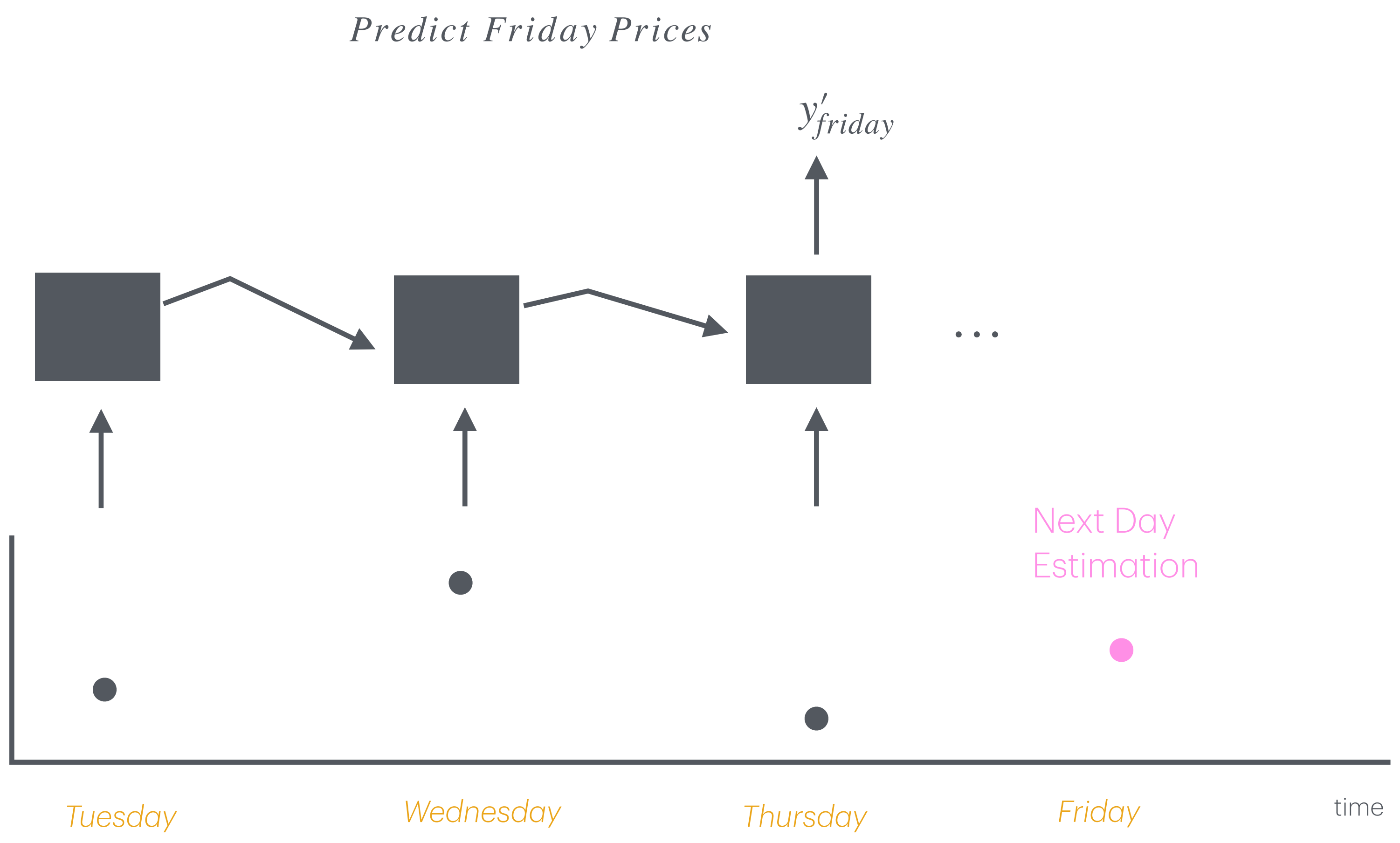
Stock prices



Instances (Train Data)

Target (Thursday Predictions using past data)

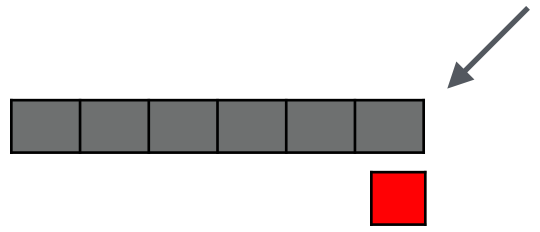
Stock prices (Predict next day using sequence (history) of data)



Recent Prices Data

XSequence

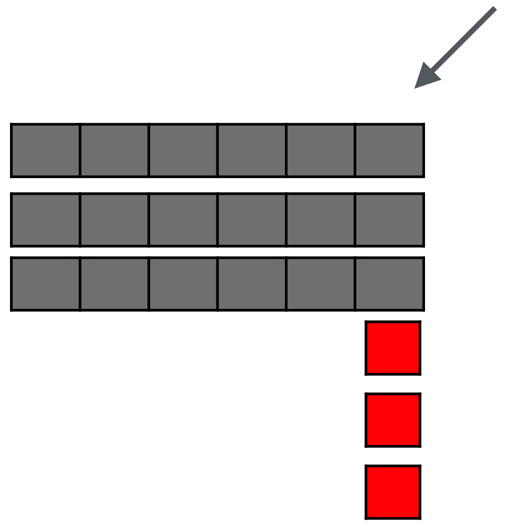
YVector



Recent Prices Data

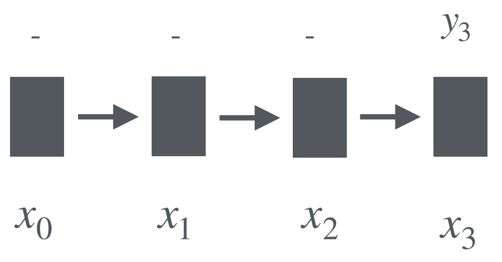
X Batch

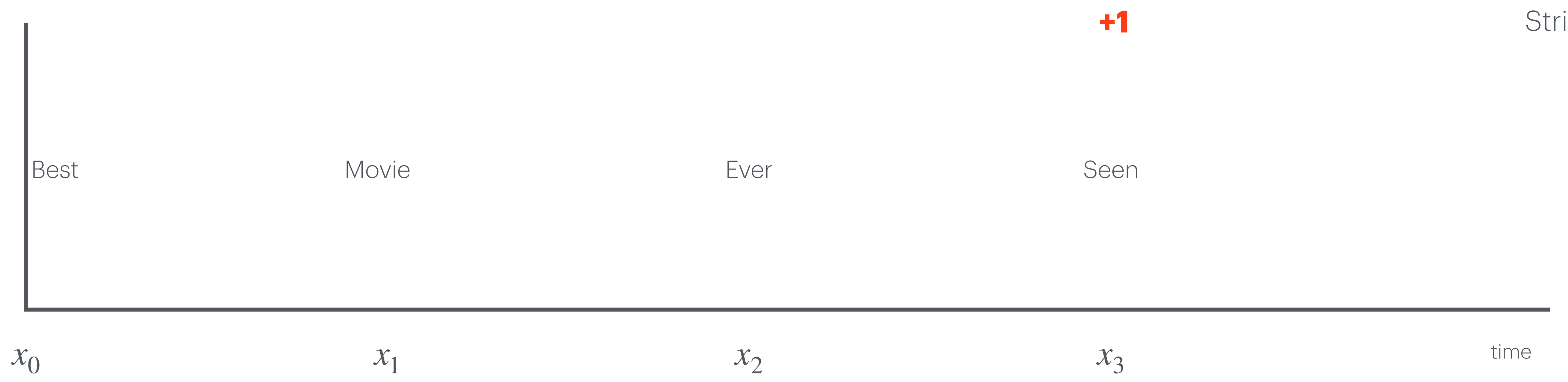
Y Vector

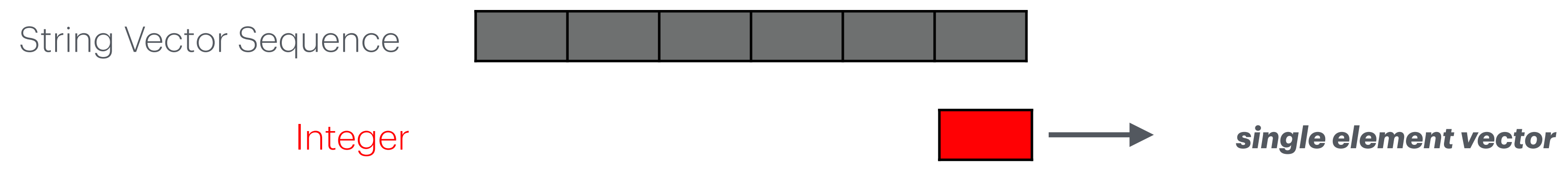


RNN

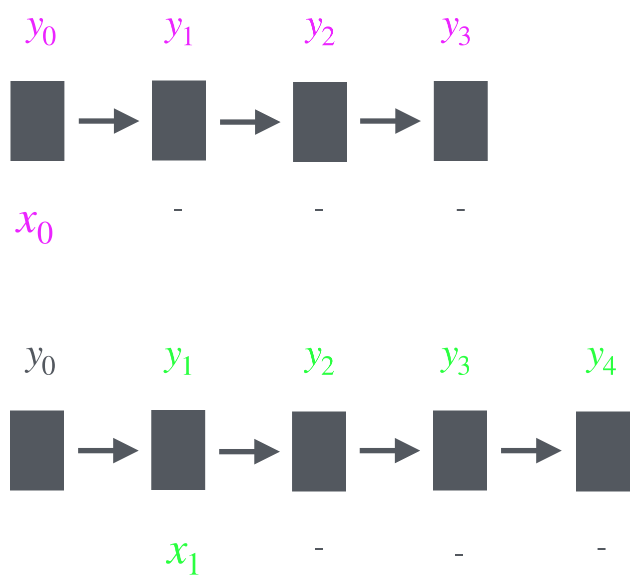
Sequence to vector

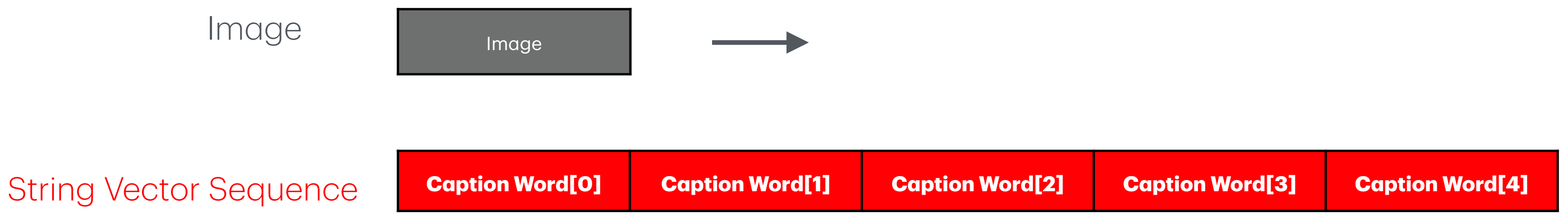


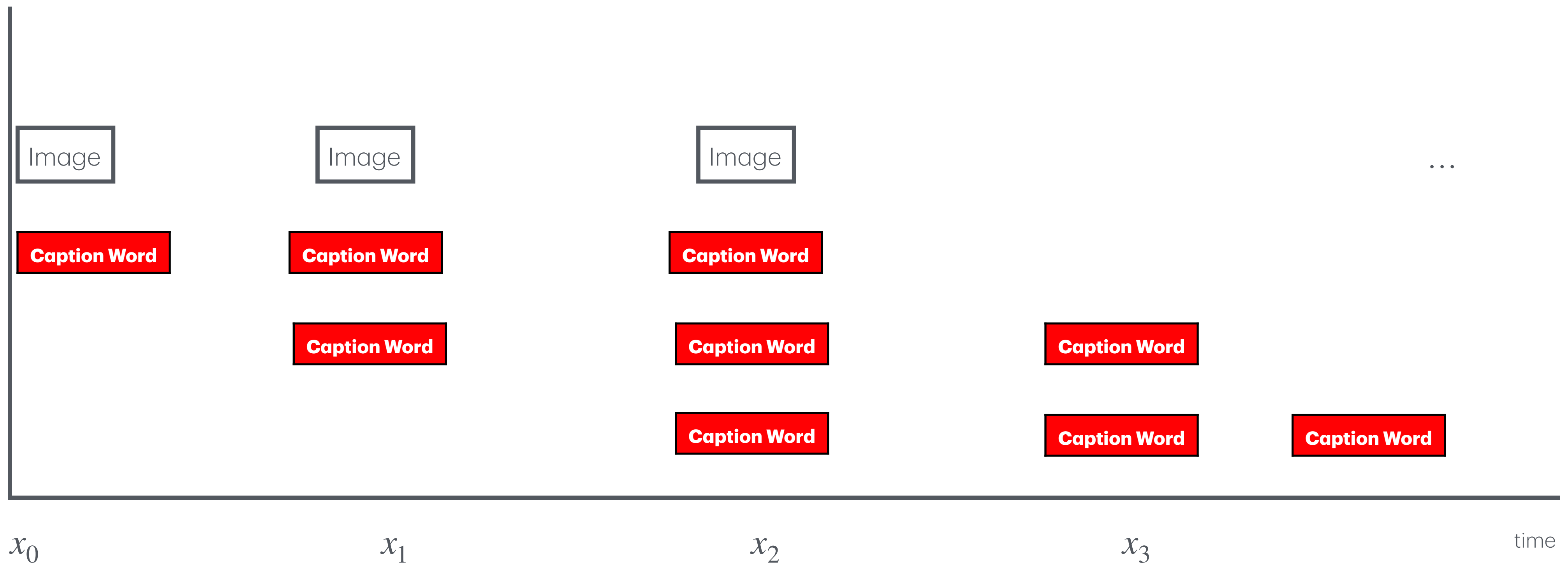




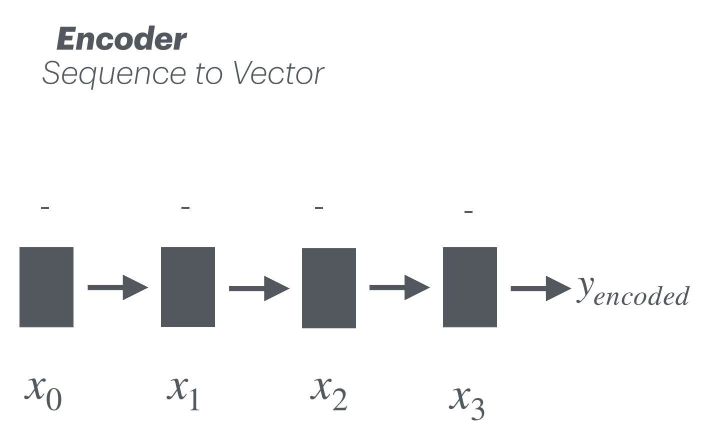
Vector to Sequence

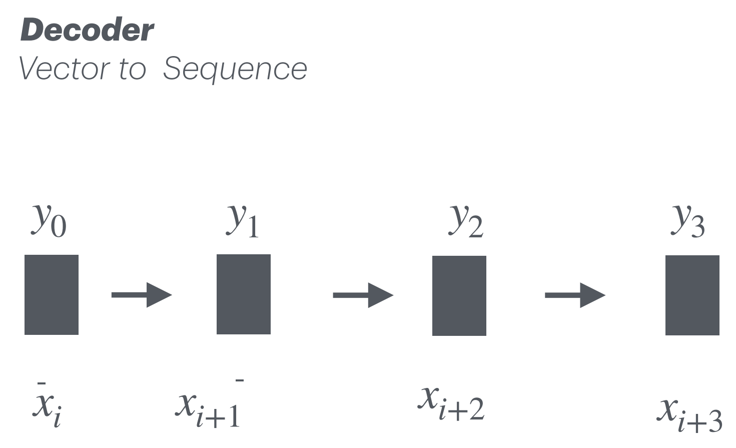


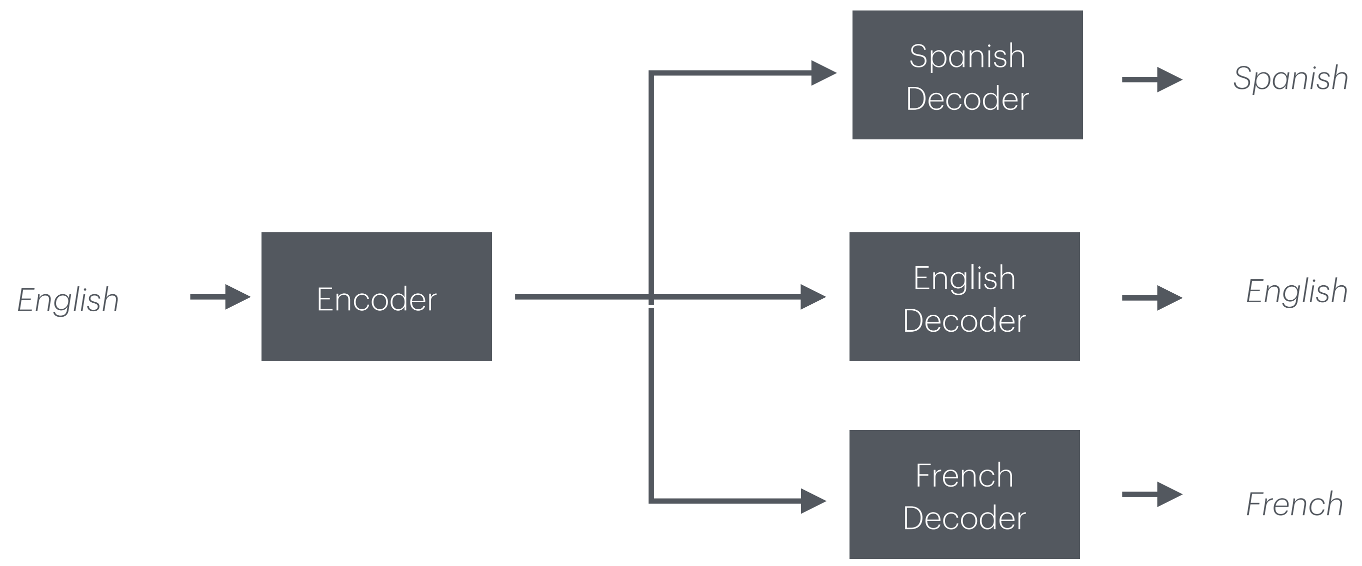


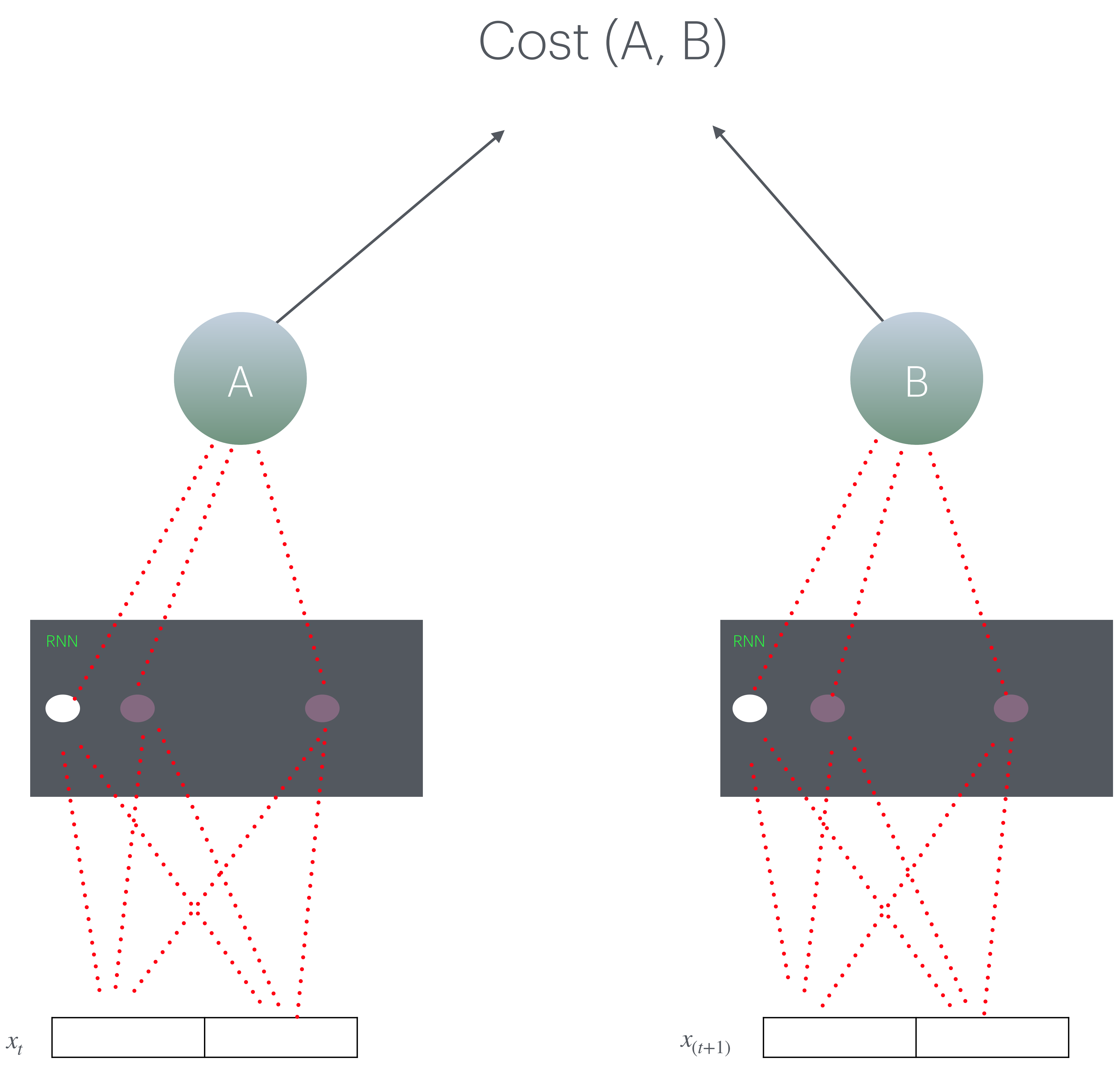


...


Encoder
Sequence to Vector
$y_{encoded}$
x_0
x_1
x_2
x_3

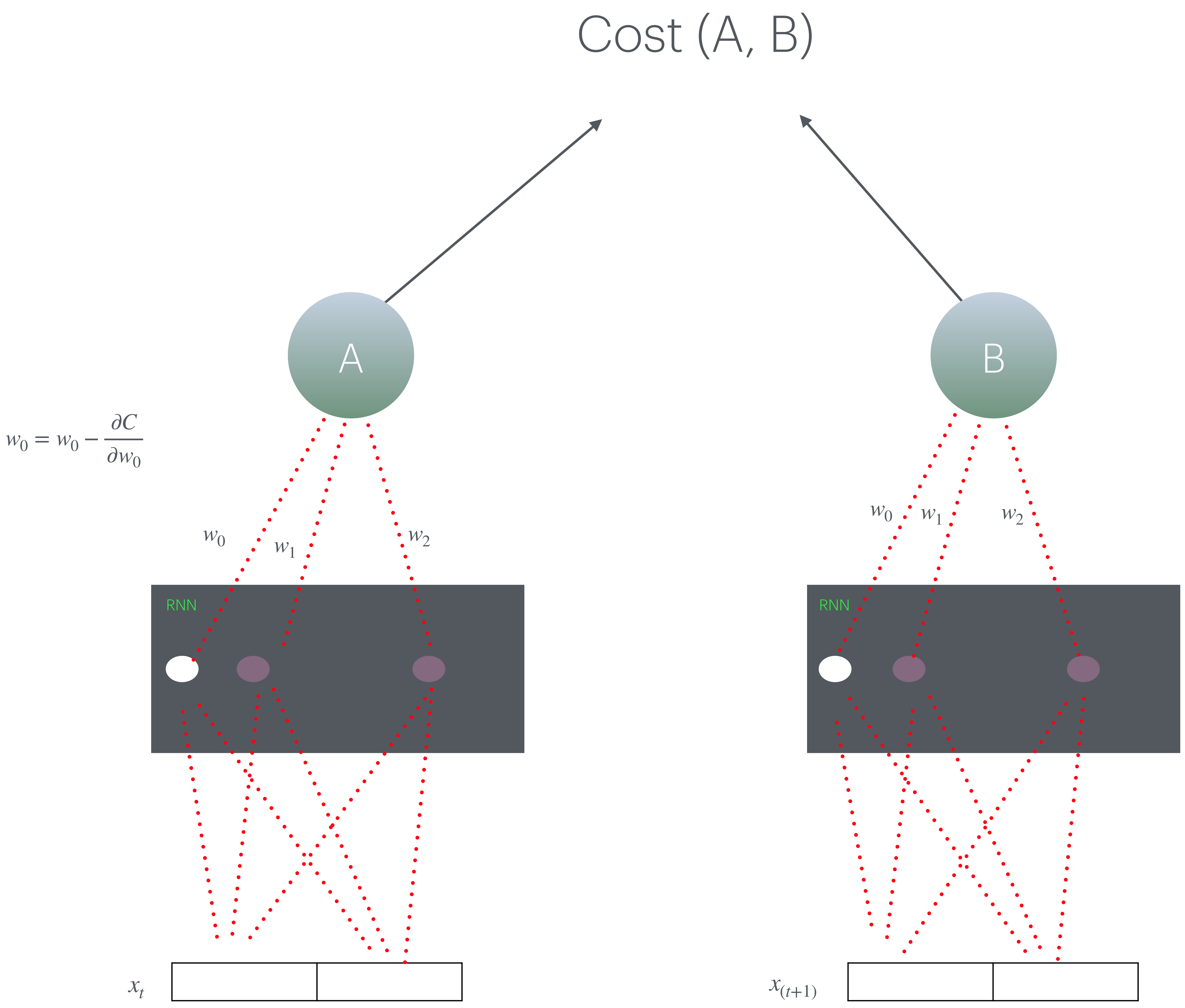

Decoder
Vector to Sequence
y_0
y_1
y_2
y_3
x_i
x_{i+1}
x_{i+2}
x_{i+3}


English
Encoder
Spanish Decoder
English Decoder
French Decoder
Spanish
English
French



Weights are equal among all RNN's

2D vector sample per time sample



$$w_0 = w_0 - \left(\frac{\partial C}{\partial w_{0_t}} + \frac{\partial C}{\partial w_{0_{t+1}}}\right) * \eta$$

Back-propagation sums over all steps

Data Setup Tensor

batch

[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]
[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]
[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]
[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]
[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]	[-]

Shape = batch_size x steps_n x 1

Univariate

steps=11

Data Setup Tensor

batch

[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]
[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]
[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]
[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]
[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]	[~, ~]

Shape = batch_size x steps_n x 2

Multivariate

steps=11

Baseline Metrics

Model which predicts the last value

Validation Set

```
[ 0.67503341 0.66000836 0.58825612 0.79083483 0.7766738  1.33158259 1.69310973 2.41718671 1.65764816 1.40107688 ]

[ 0.42877583 1.47534251 1.57482383 1.24177148 1.48009739 1.14153755 1.45149384 0.30831861 0.32696614 0.12650302 ]

[ 0.55692501  0.957339    1.5249232   1.1005888   0.88698048  0.12466827 -1.06207913 -1.41600146 -1.01998381 -0.71028766 ]
```

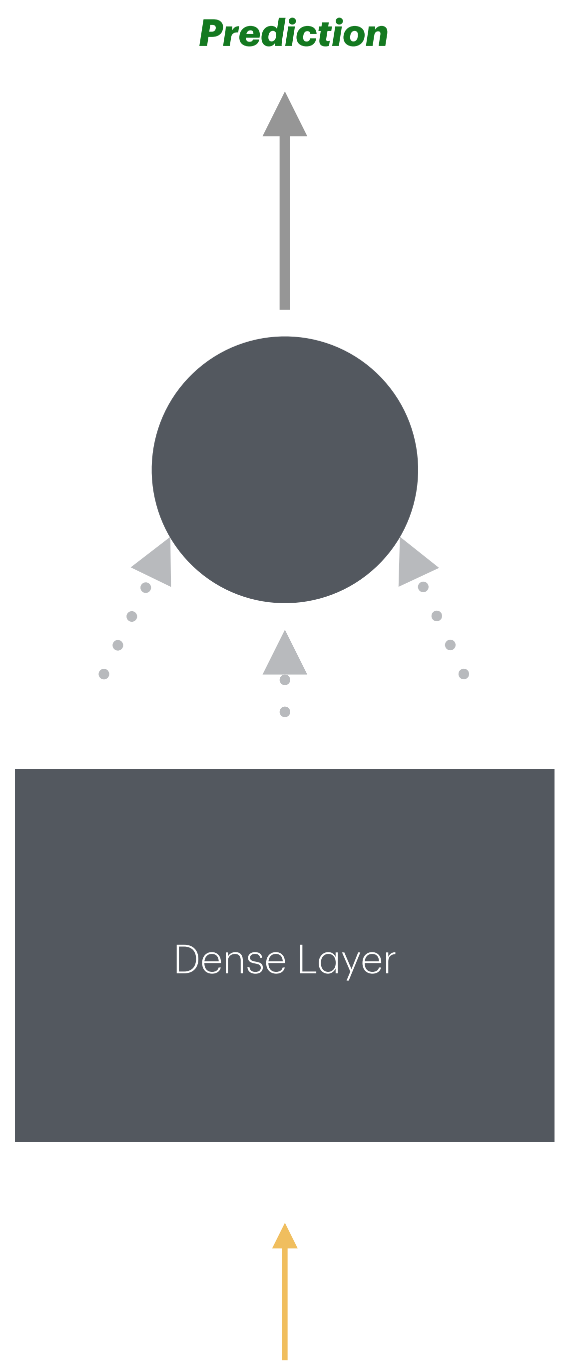
⋮

y_pred

y_target

Avg mean squared error of the obviously incorrect prediction but the lowest acceptable guess will provide a good baseline.

The RNN model should perform better than this baseline MSE score



Train for about 22 epochs and calculate MSE on the validation set. Score (i.e. MSE) will be better than the previous baseline model.

Train Set

[0.67503341 0.66000836 0.58825612 0.79083483 0.7766738 1.33158259 1.69310973 2.41718671 1.65764816 1.40107688]

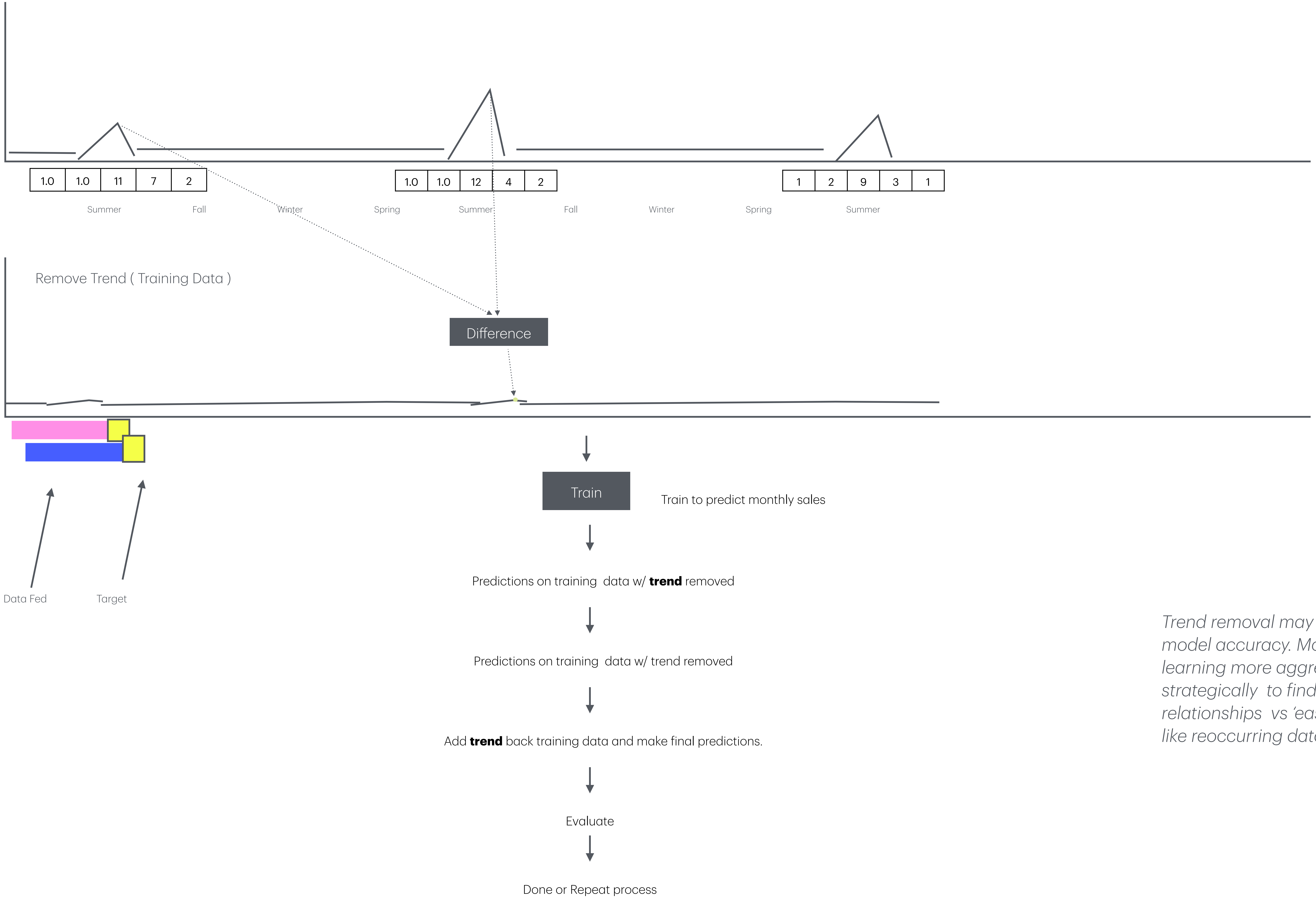
[0.42877583 1.47534251 1.57482383 1.24177148 1.48009739 1.14153755 1.45149384 0.30831861 0.32696614 0.12650302]

[0.55692501 0.957339 1.5249232 1.1005888 0.88698048 0.12466827 -1.06207913 -1.41600146 -1.01998381 -0.71028766]

⋮

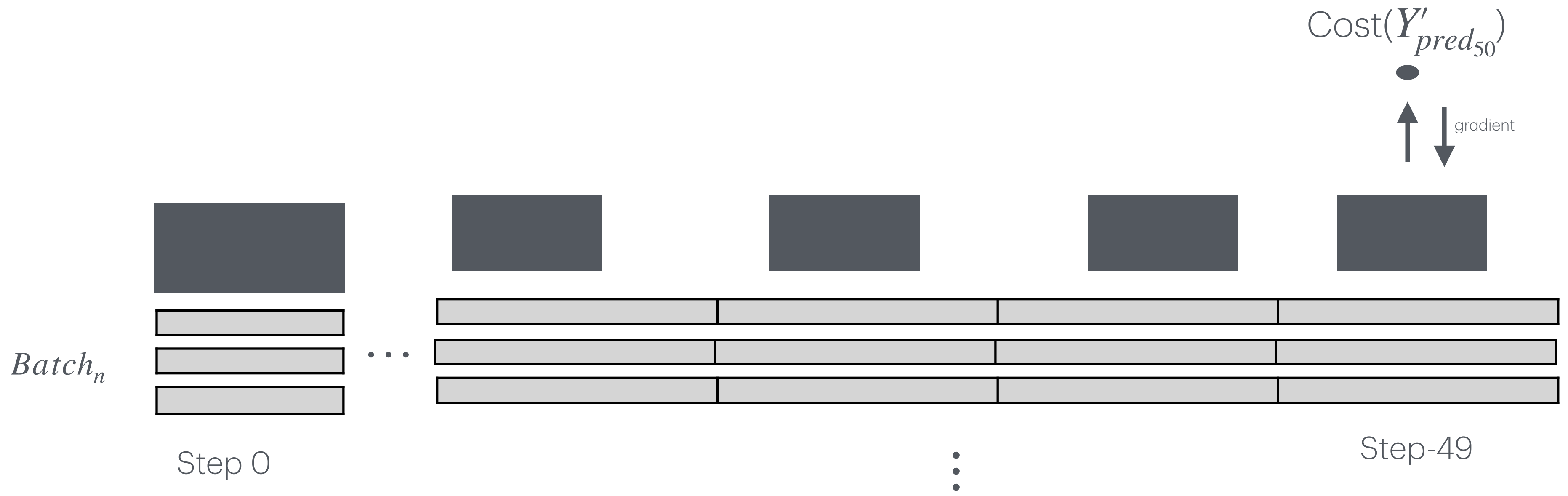
[Predictions Target]

Trendy Data

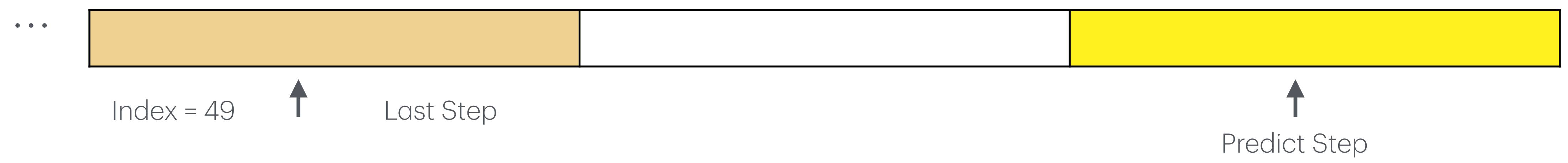


Trend removal may improve model accuracy. Model is learning more aggressively or strategically to find deeper relationships vs 'easy' patterns like reoccurring data trend.

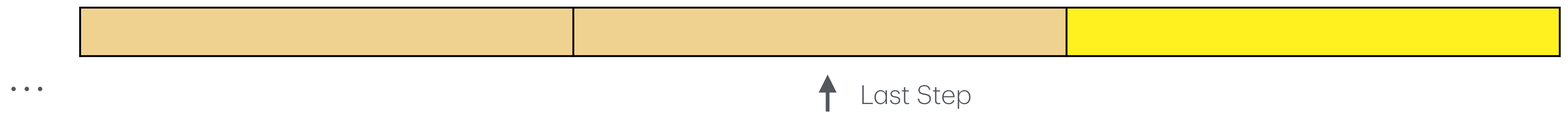
Note: Gradients produced for last step only during training



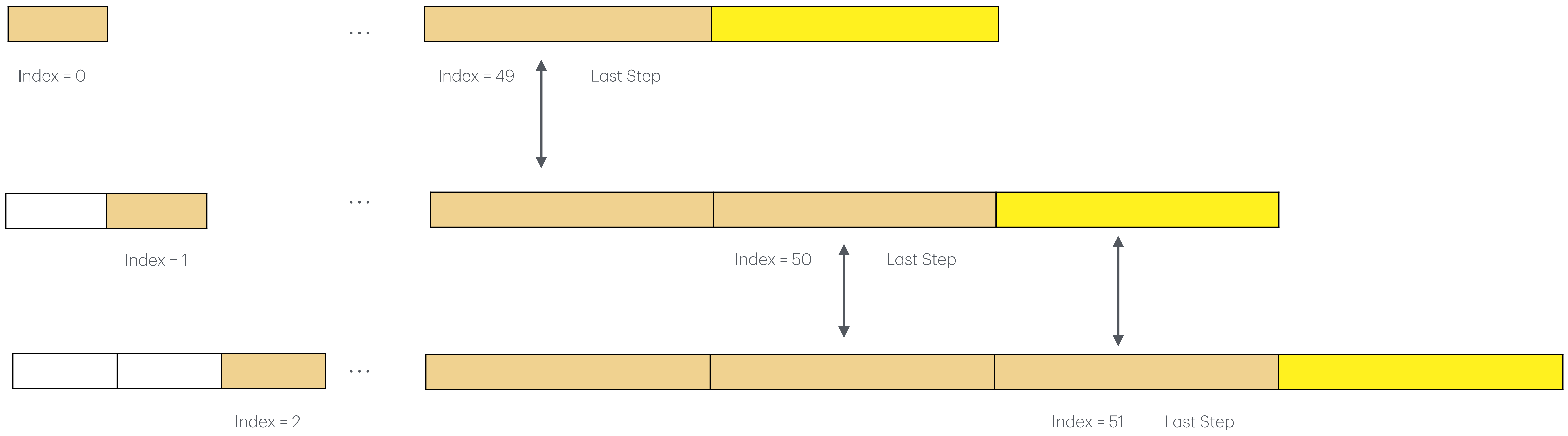
Predict step ahead of last step



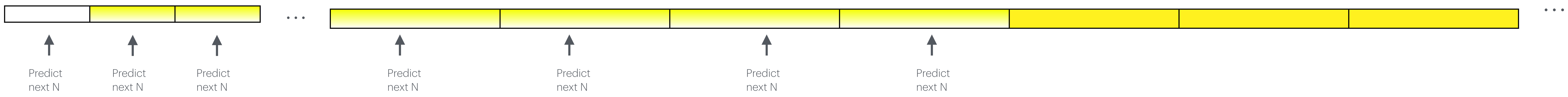
Predict step after last step



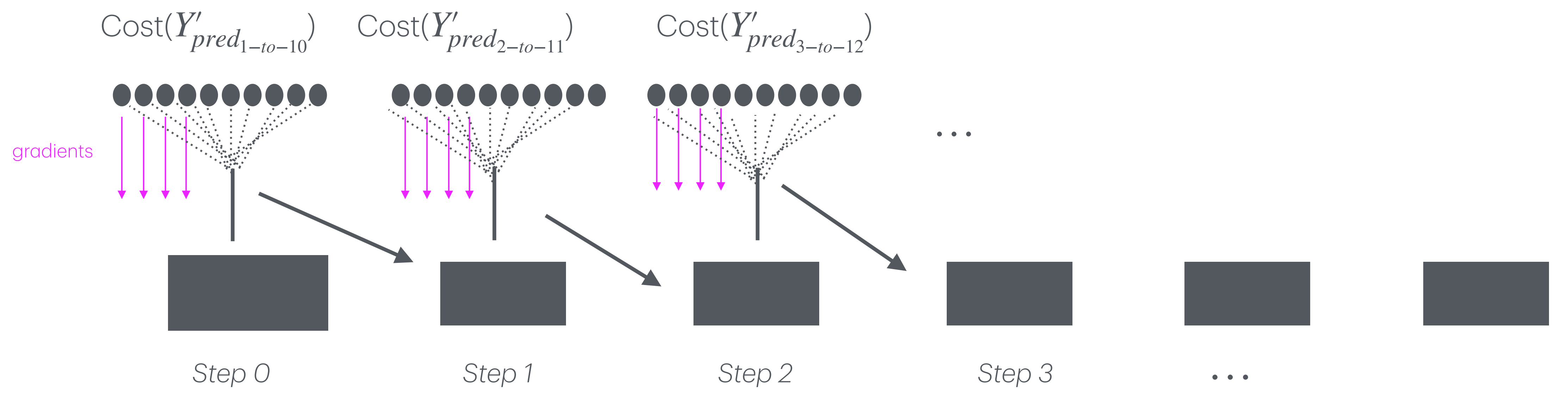
Predict next three values after last step



Sequence to Sequence Model



Note: Gradients produced for every step during training. Lots of gradients which will boost model's learning speed, and stabilize gradients (eliminate unstable gradients problem)



Training Data

batch_size x 50 x 1

	x_0	x_1	x_2
$X_{instance_0}$	[value_00]	[value_01]	[value_02]
$X_{instance_1}$	[value_10]	[value_11]	[value_12]
$X_{instance_2}$	[value_20]	[value_21]	[value_22]
	[value_30]	[value_31]	[value_32]

Target

batch_size x 50 x 10

Time Distributed

instance_0

10

50

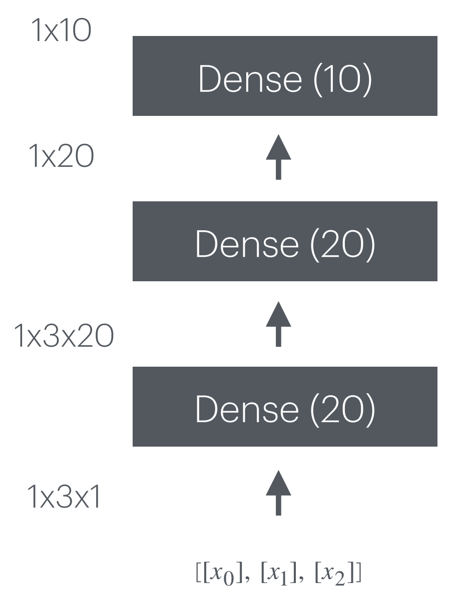
Y_{x_0}	[value_01]	[value_02]	[value_03]
Y_{x_1}	[value_11]	[value_12]	[value_13]
Y_{x_2}	[value_21]	[value_22]	[value_23]
Y_{x_3}	[value_31]	[value_32]	[value_33]

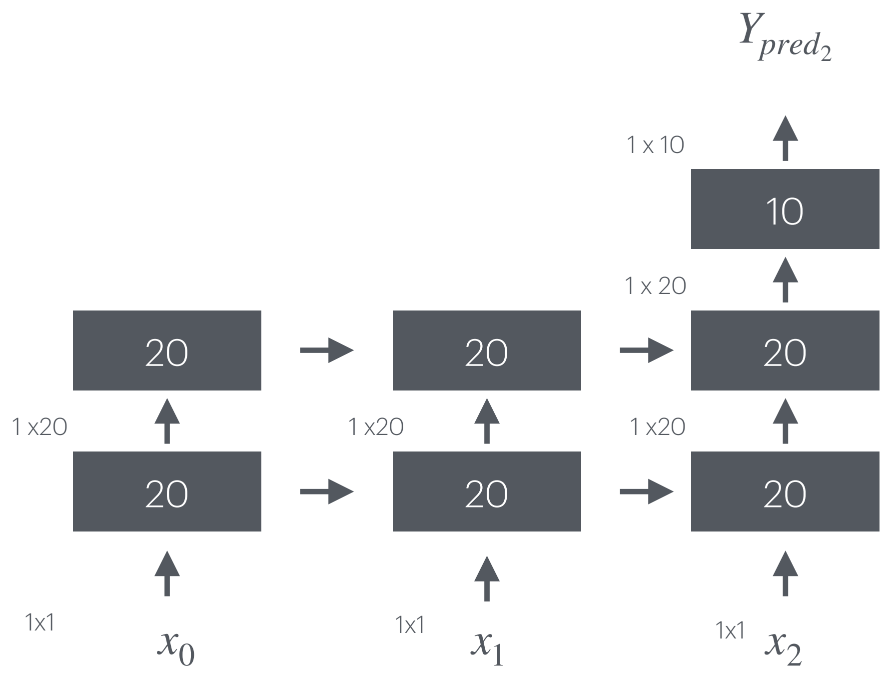
50 target vectors per instance. Each target(Y_{x_i}) is a prediction at step x_i

instance_1

instance_2

Sequence to Vector Model

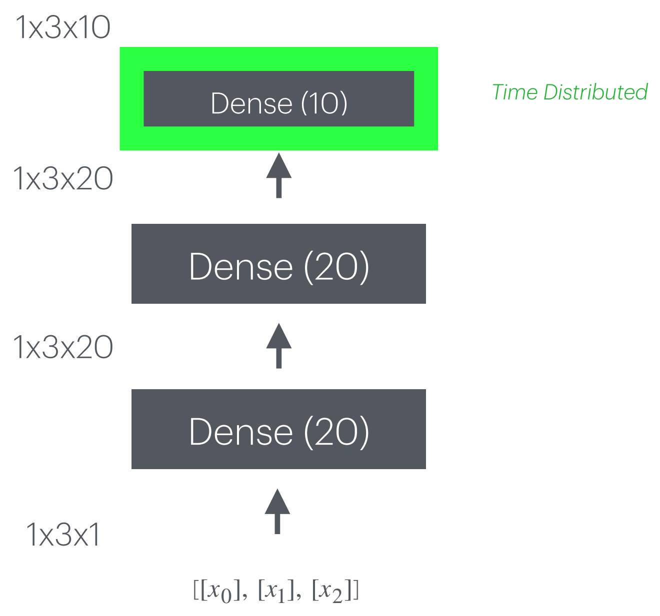


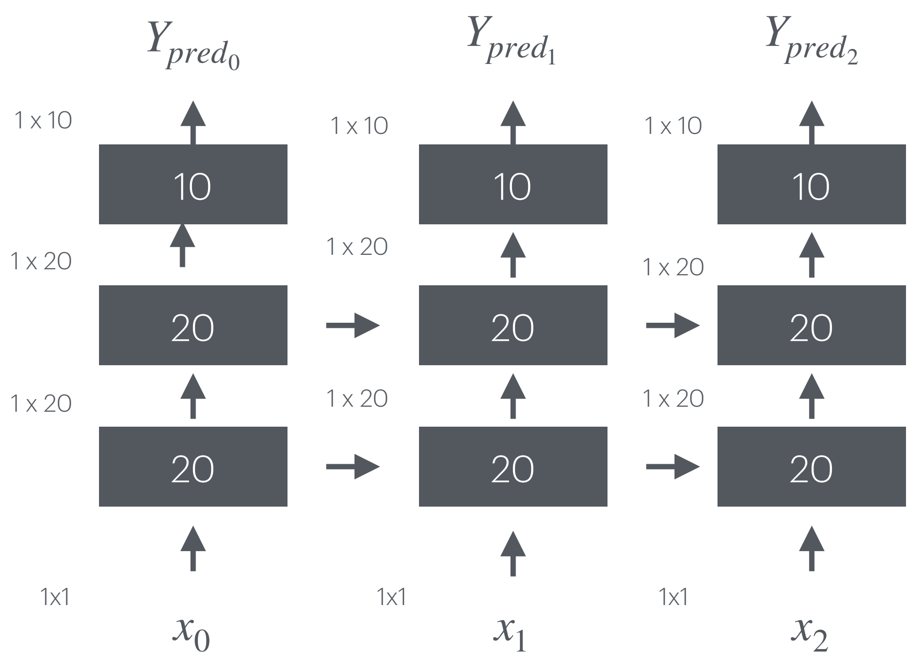


Tensorflow: return sequences up until input of layer feeding output layer. **The last step forecasts**

Note: Output Dense layer applied independently at each time step.

Sequence to Sequence Model



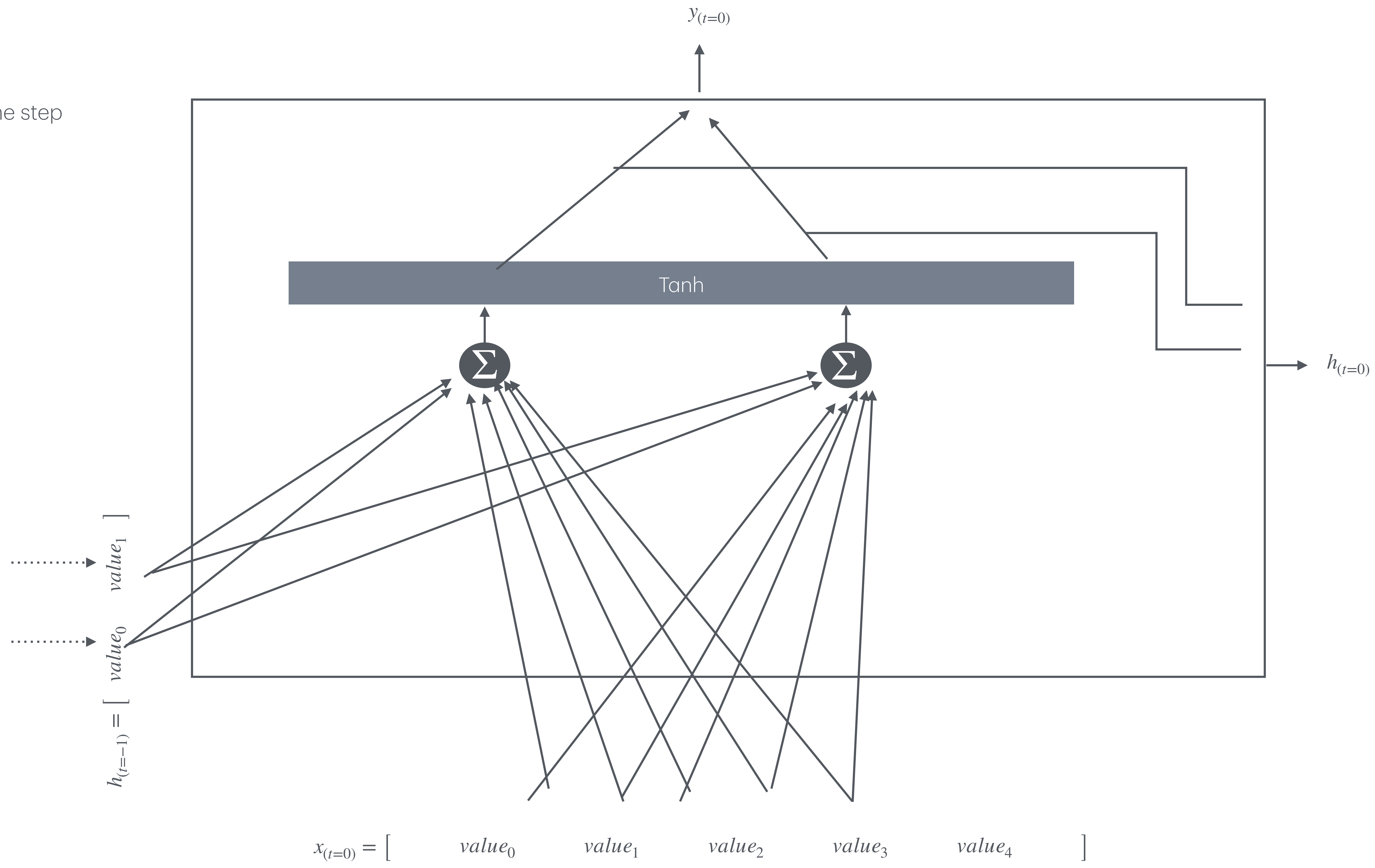


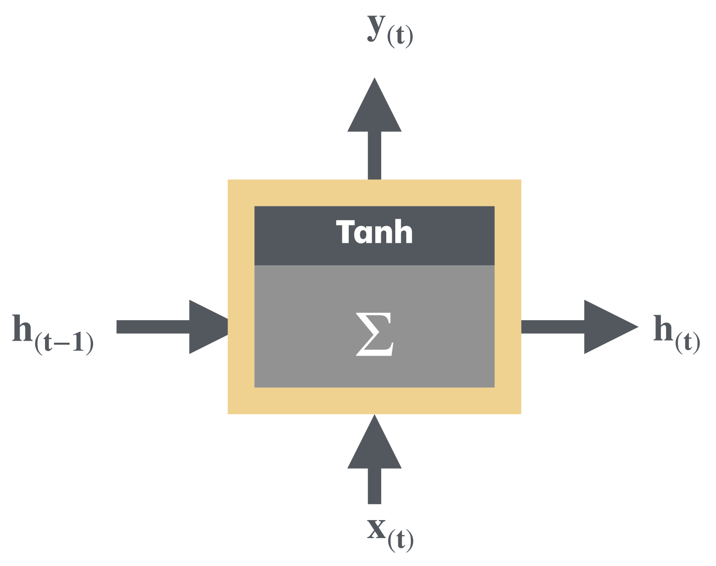
Tensorflow: return sequences for each RNN layer: **Each step forecasts next values**

- All outputs needed during training
- Last step needed for predictions and evaluations. Custom metric required validation and test set evaluations
-

2 Units

5 Features per time step

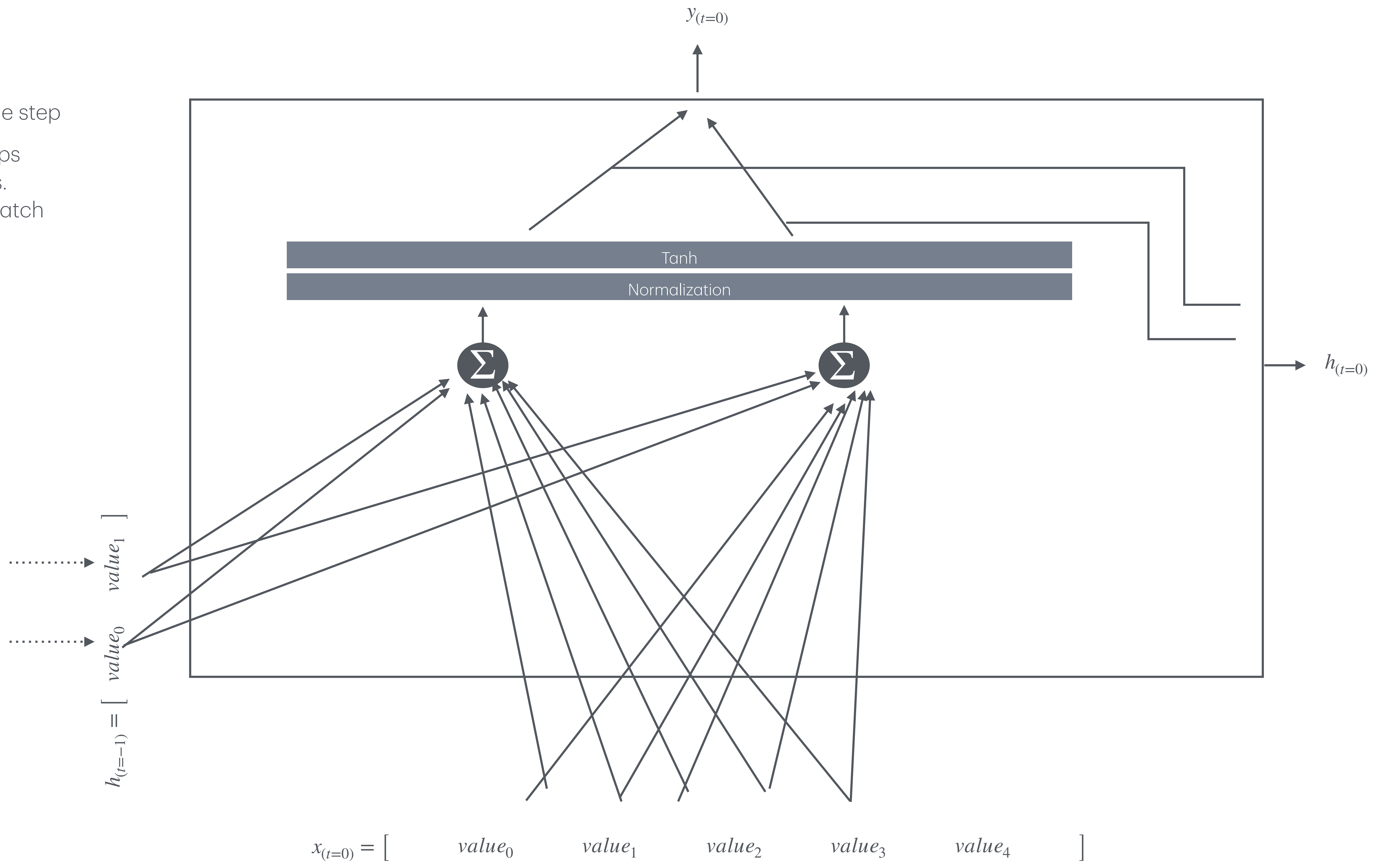


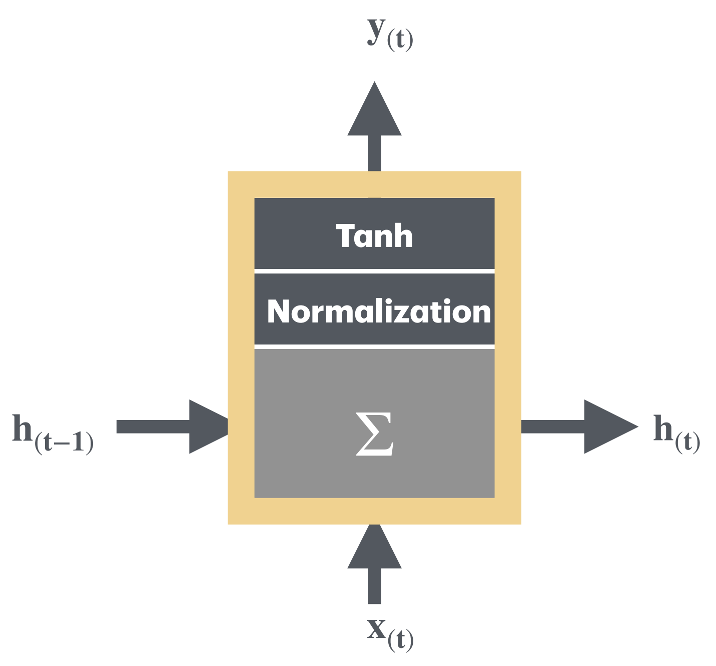


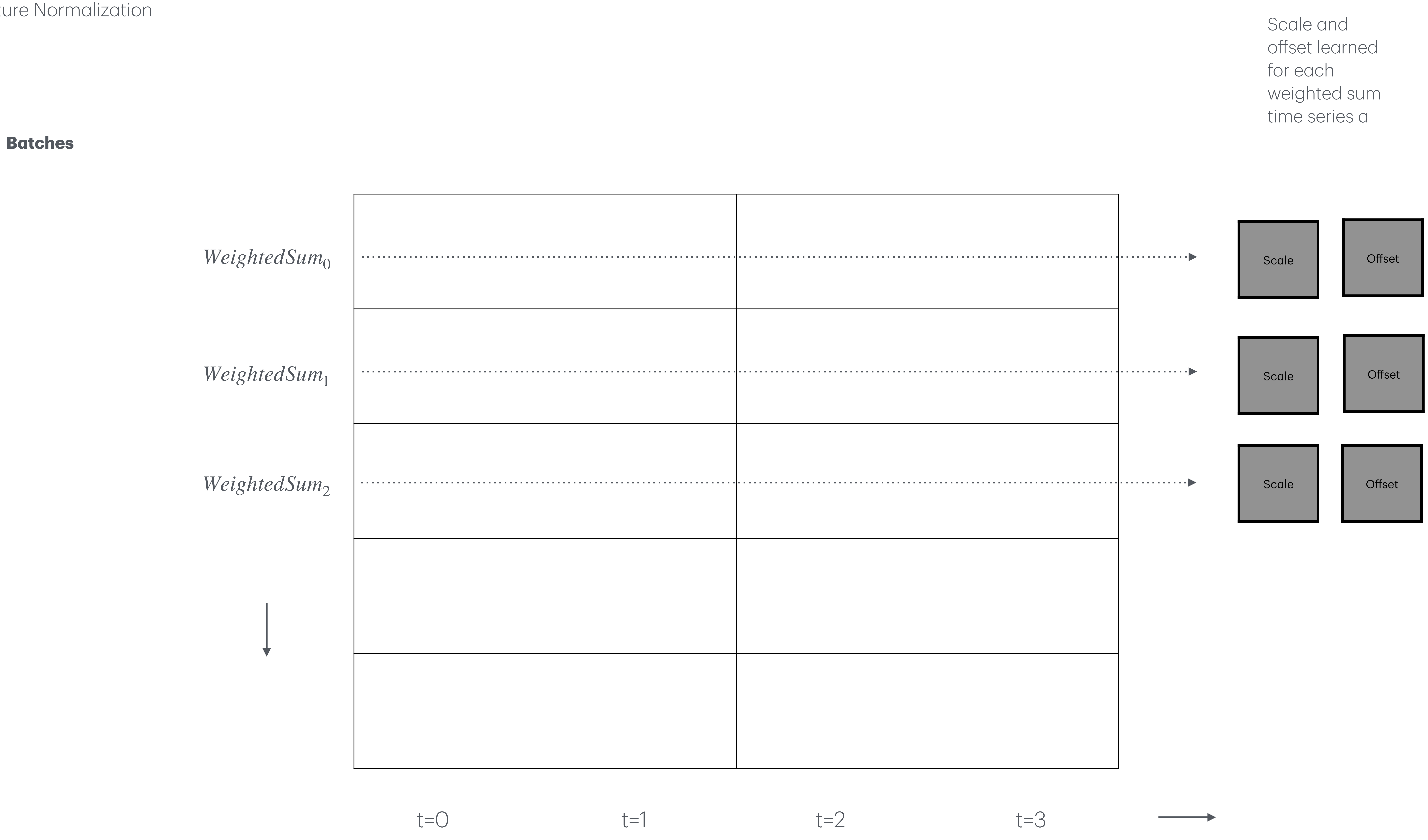
2 Units

5 Features per time step

Normalization helps stabilize gradients. Similar effect as Batch Normalization

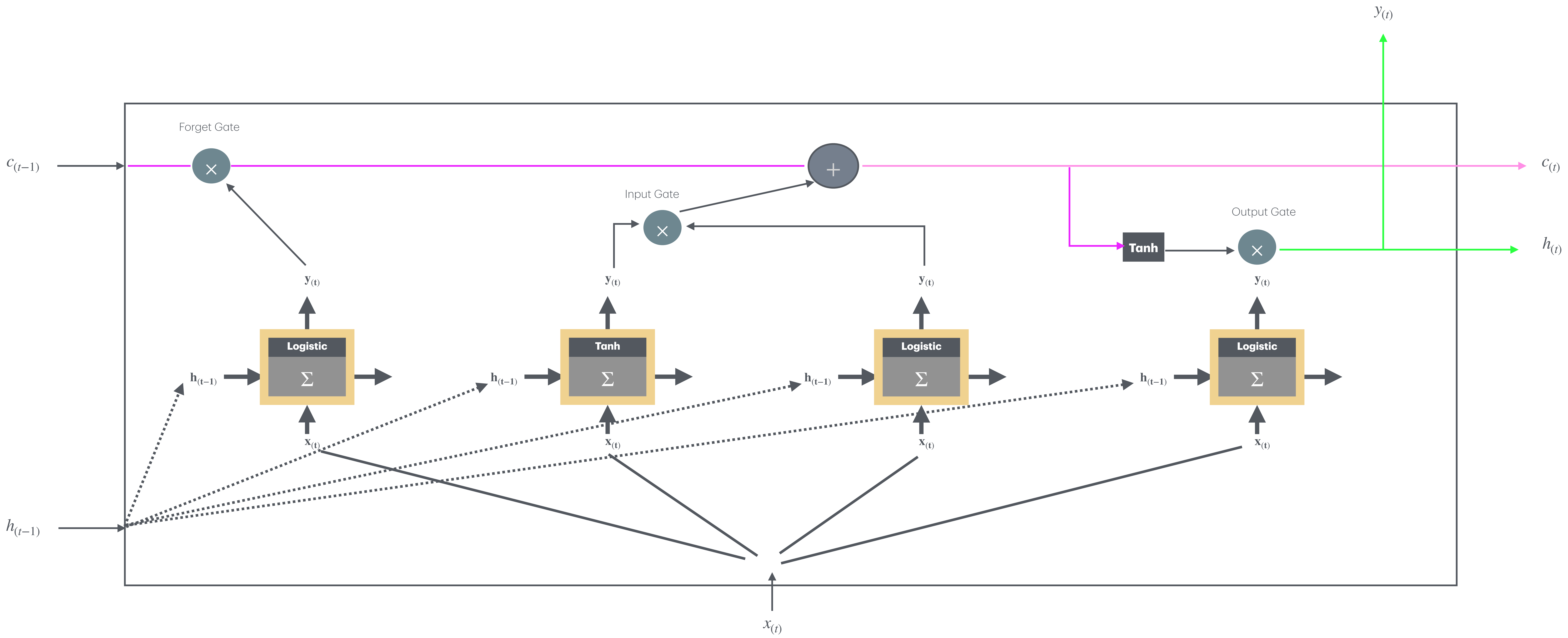





Scale and offset learned for each weighted sum time series a
Batches
$WeightedSum_0$
$WeightedSum_1$
$WeightedSum_2$
Scale
Offset
Scale
Offset
Scale
Offset
t=0
t=1
t=2
t=3

Translate sentence….while translating you **forget** the initial sentence translation which influences all aspects of sentence structure. Only remember most recent translation.

Deep RNNs: Long Short Term Memory LSTM Cell



- Input Gate : Which part of input to add to long-term state (important inputs artifacts are preserved for current or future outputs)
- Output Gate - Which part of long term state should be read and output at time step (i.e. estimation at time step)
- Forget Gate - Which part of long term state should be erased (remove useless input artifacts)

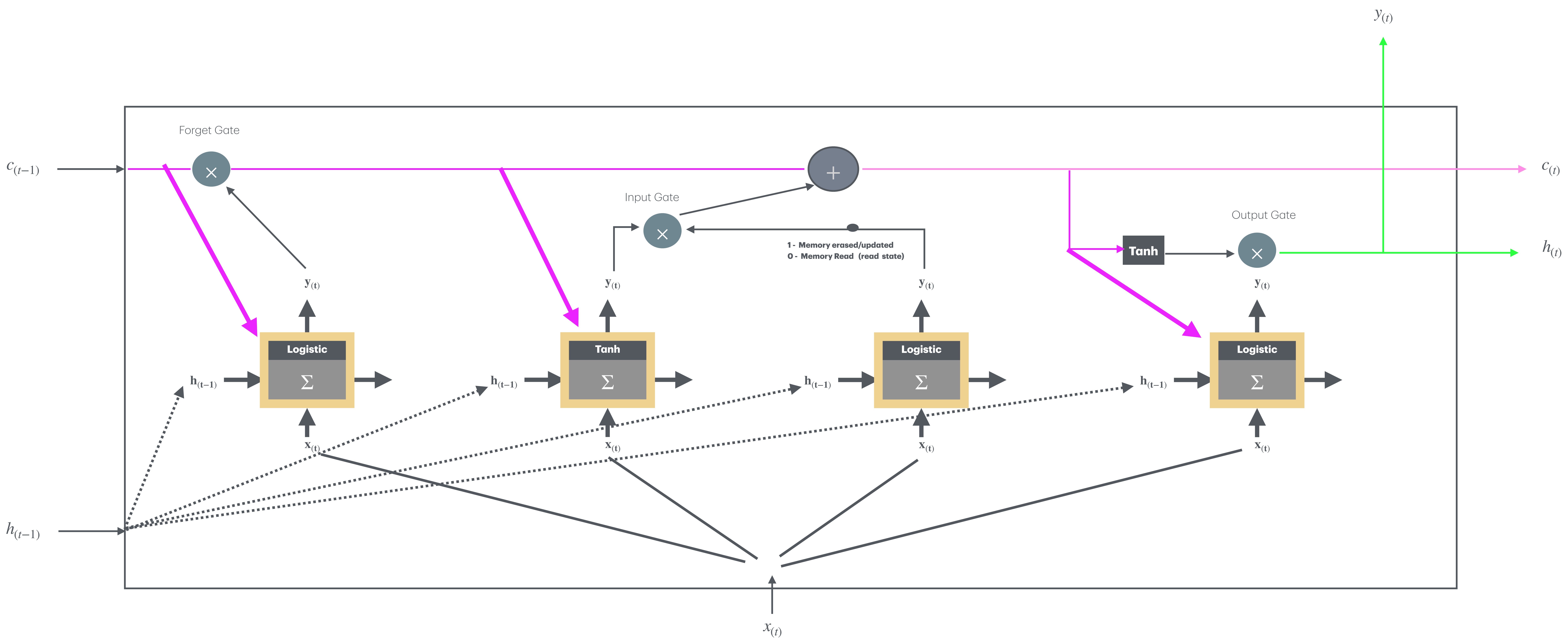
Long Term State
Output State

Gate Controller Outputs:
0 - Close
1 -Open

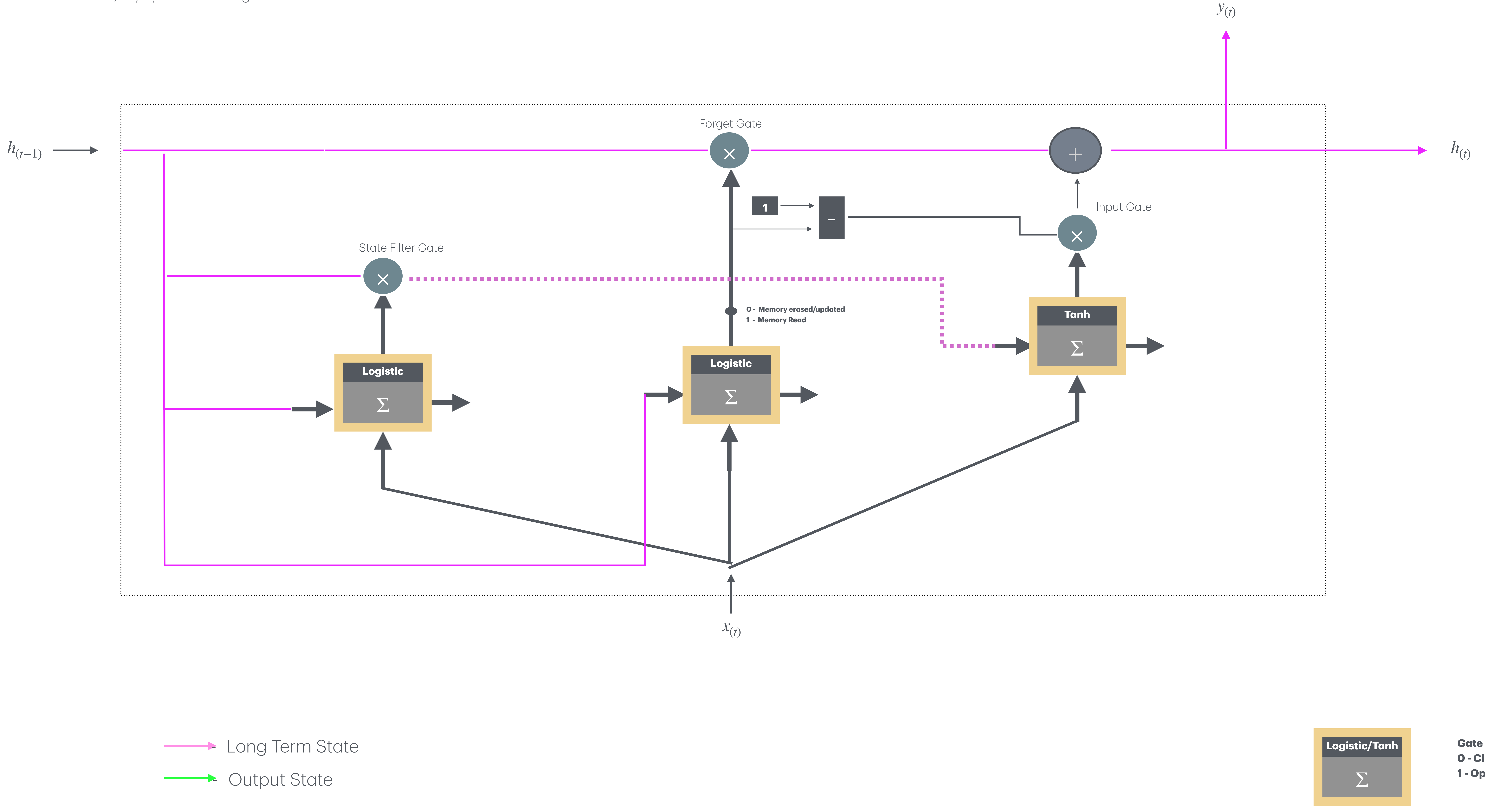
- Input Gate : Which part of input to add to long-term state (important inputs artifacts are preserved for current or future outputs)
- Output Gate - Which part of long term state should be read and output at time step
- Forget Gate - Which part of long term state should be erased (remove useless input artifacts)

Gate Controller Outputs:
0 - Close
1 - Open

Deep RNNs: LSTM Cell Peephole connections



Introduced in 2014, in paper introducing Encoder-Decoder network

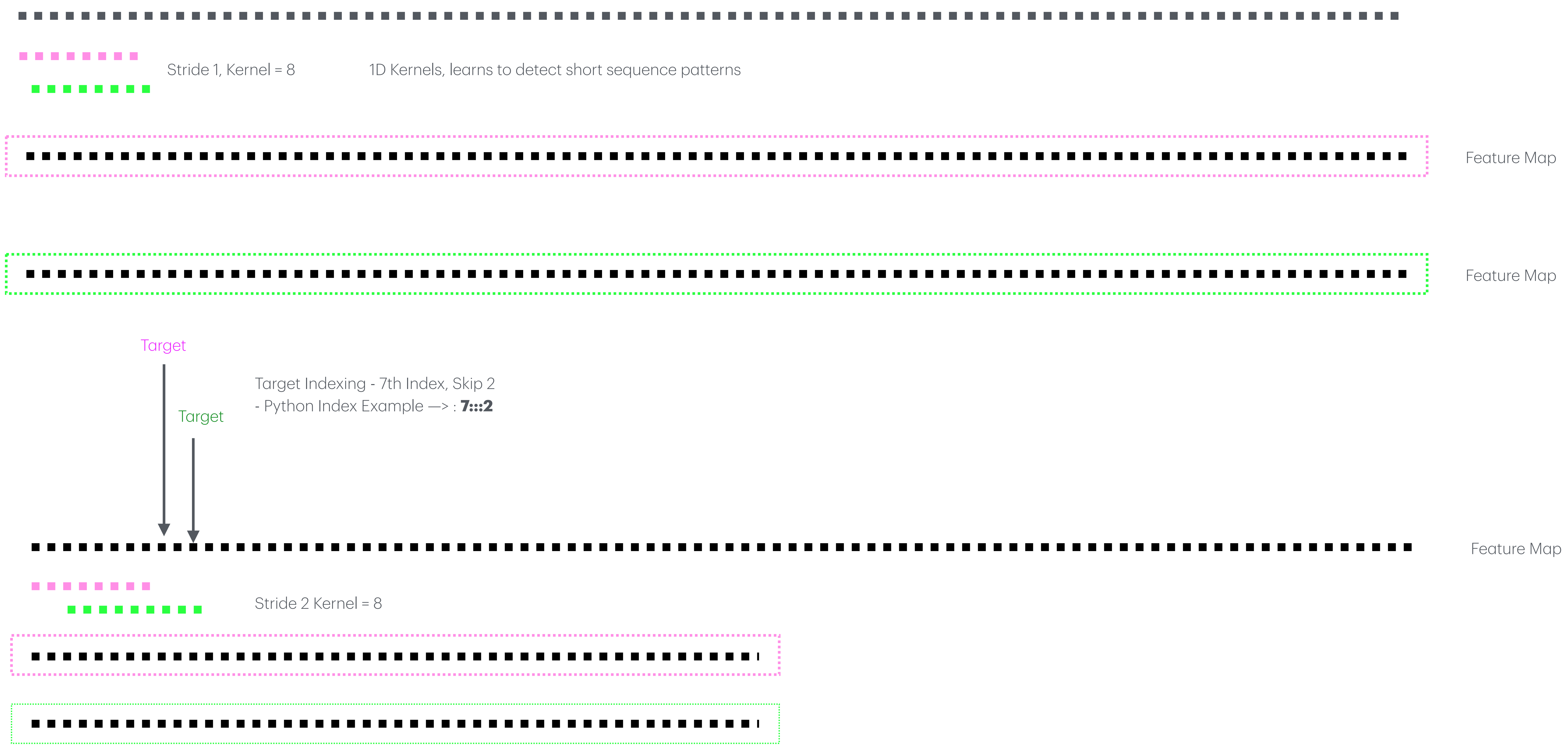


Gate Controller Outputs:
0 - Close
1 - Open

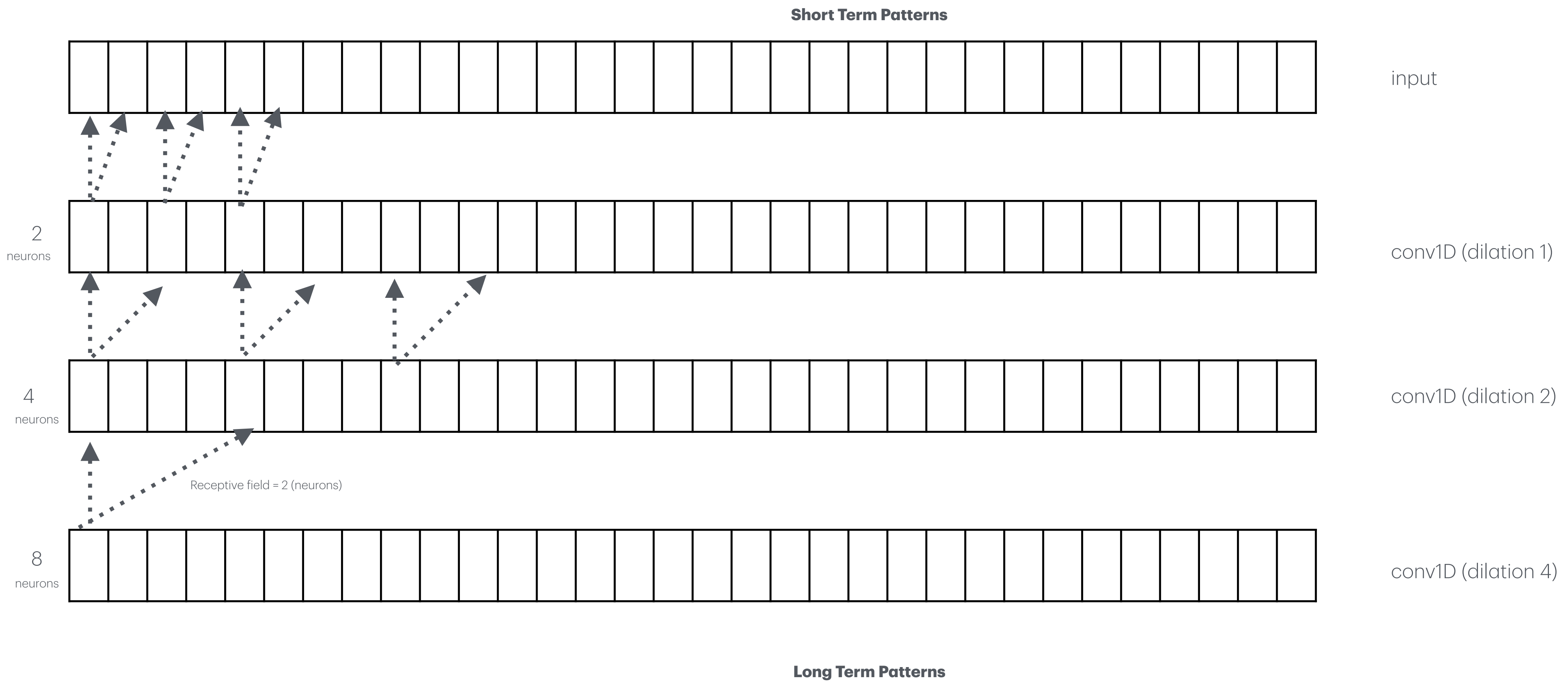
Deep RNNs: 1D Convolutional Layers

Fairly limited short term memory, hard time learning patterns in sequences >= 100 steps in length

Solve: Shorten Input sequences (1D convolutional Layer)



Deep RNNs: WaveNet



Deep RNNs: WaveNet

$$1D - Size = (prev_{dilation} + curr_{dilation}) \times input_{size} + input_{size}$$

Input is padded accordingly to preserve input sequence length

Input 1x32 → Conv1D → 20 x 32 → Conv1D → 20 x 32 → Conv1D → 20 x 32 → Conv1D → 10 x 32 →

Conv1D:
-kernel size = 2
Stride = 1
- Dilation(1)
- 20 Maps/Channels
- -RELU
- WaveNet

Conv1D:
-kernel size = 2
- Dilation(2)
- 20 Maps/Channels
- RELU
- WaveNet

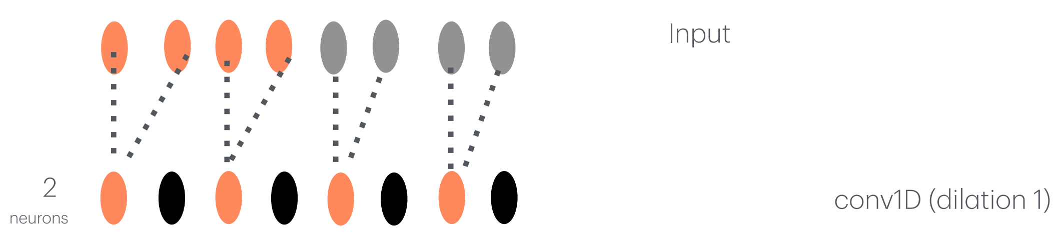
Conv1D:
-kernel size = 2
- Dilation(4)
- 20 Maps/Channels
- RELU
- WaveNet

Conv1D:
-kernel size = 1
Stride = 1
- 10 Maps/Channels

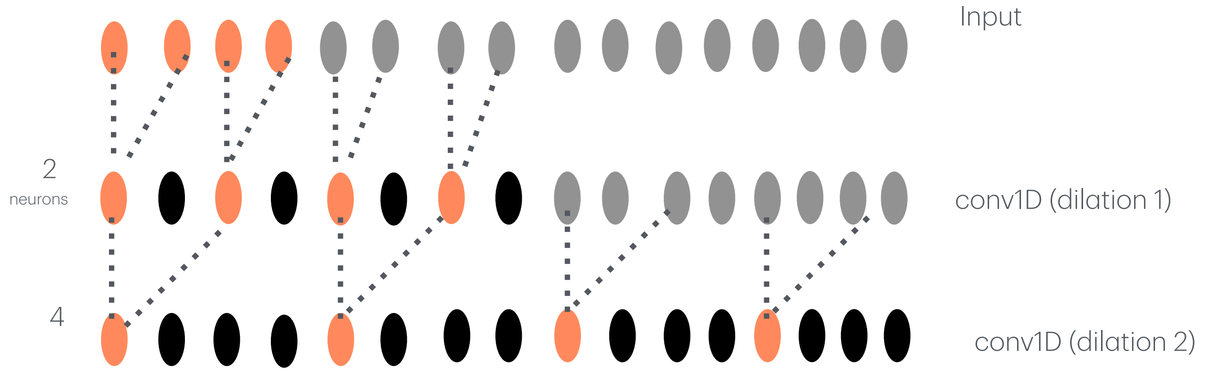
Outputs are same length as input sequences. WaveNets are useful when training a model with targets the same size as input sequences.

e.g.
Tx Data = 32 Samples
Radar Cross Section Rx Data (Target) = 32 Samples
Model can classify elements in space from Rx Data after training on large data pool of objects in space

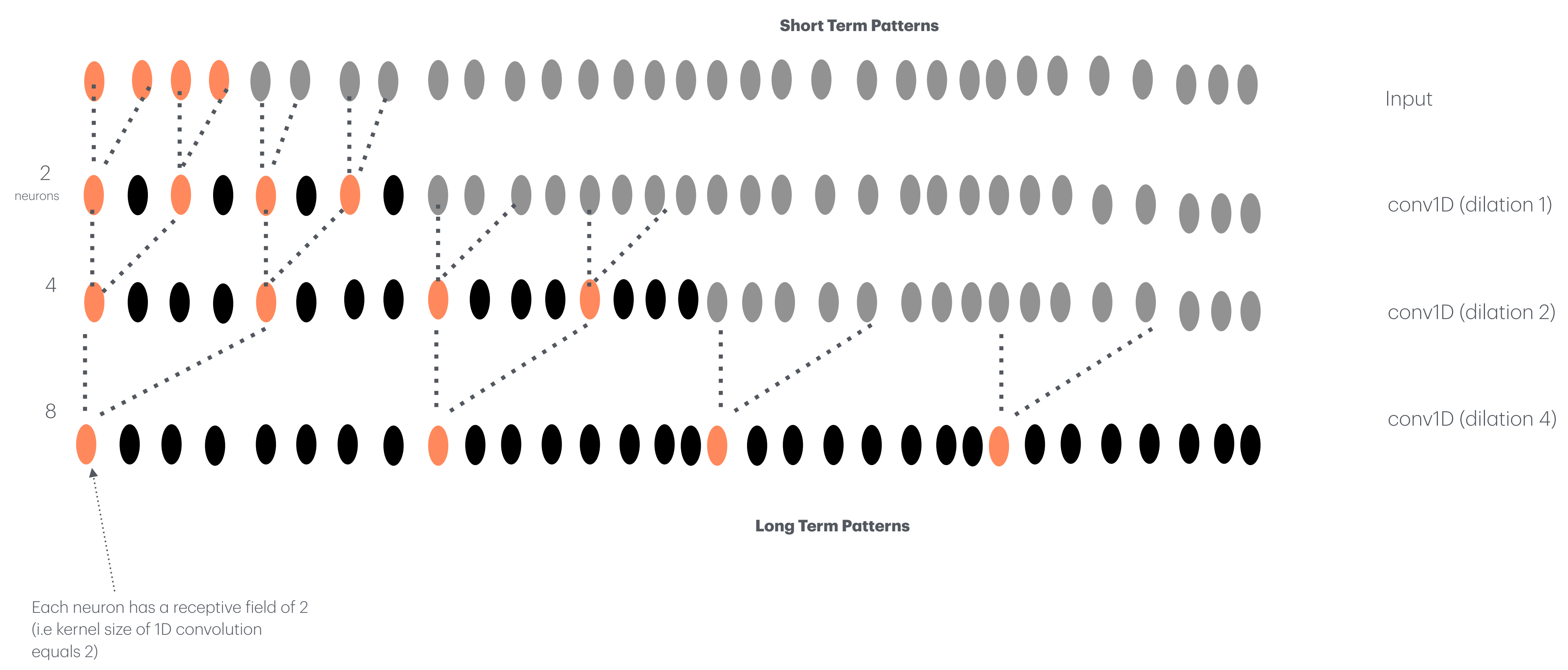
Deep RNNs: WaveNet (What WaveNet used on small sequence)



Deep RNNs: WaveNet (What WaveNet used on small sequence)

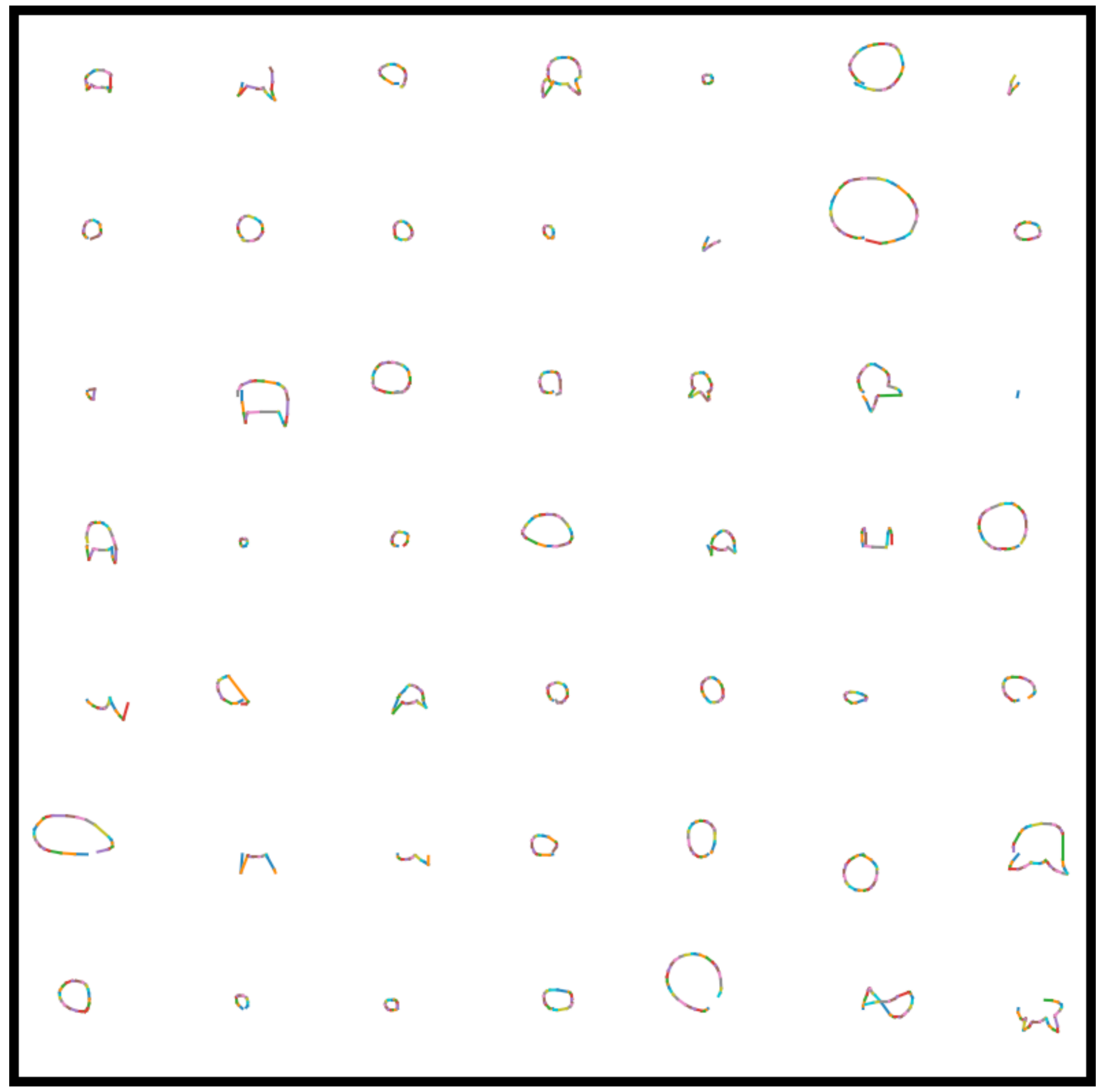


Deep RNNs: WaveNet (What WaveNet used on small sequence)

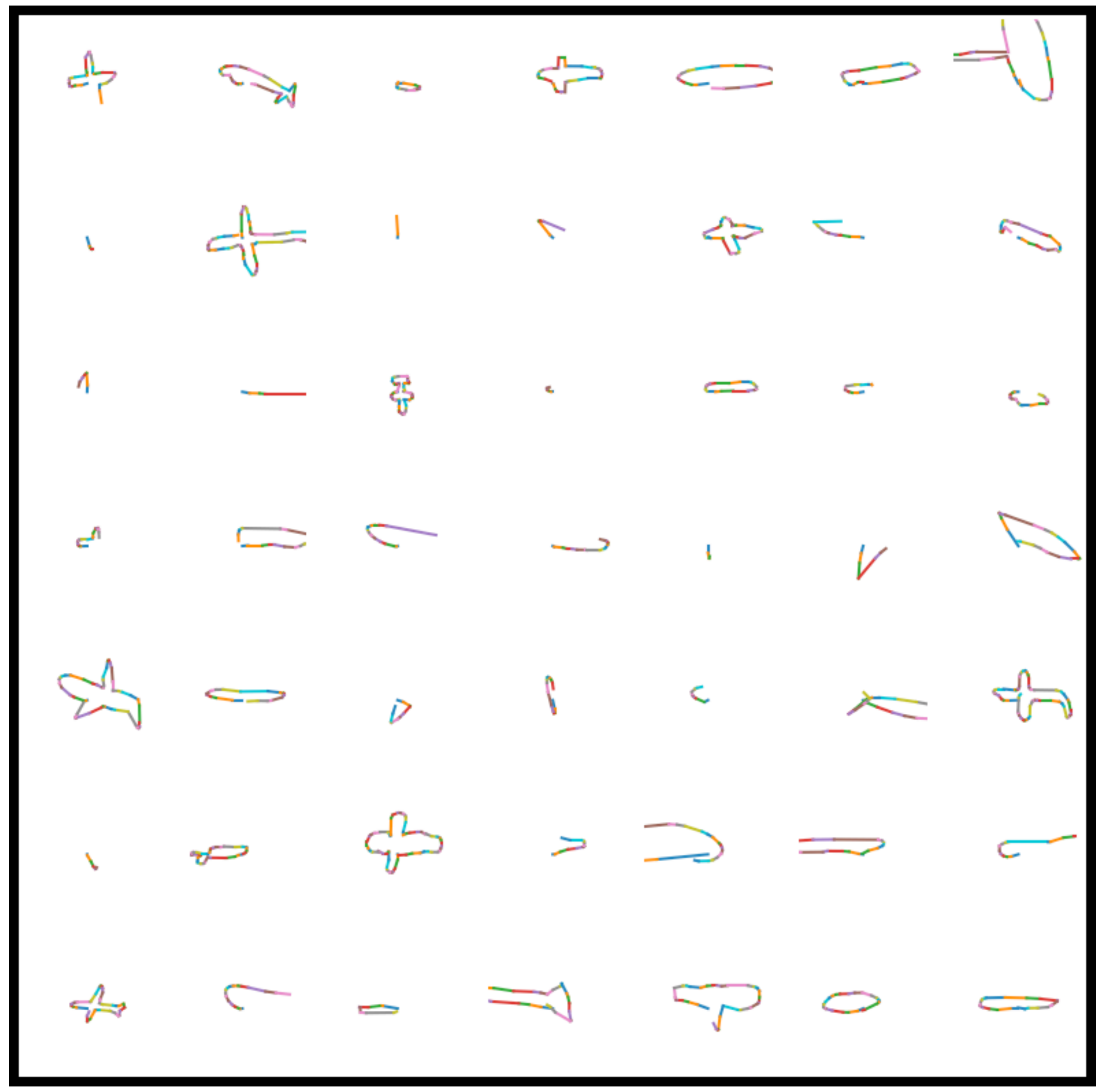


Classify Videos

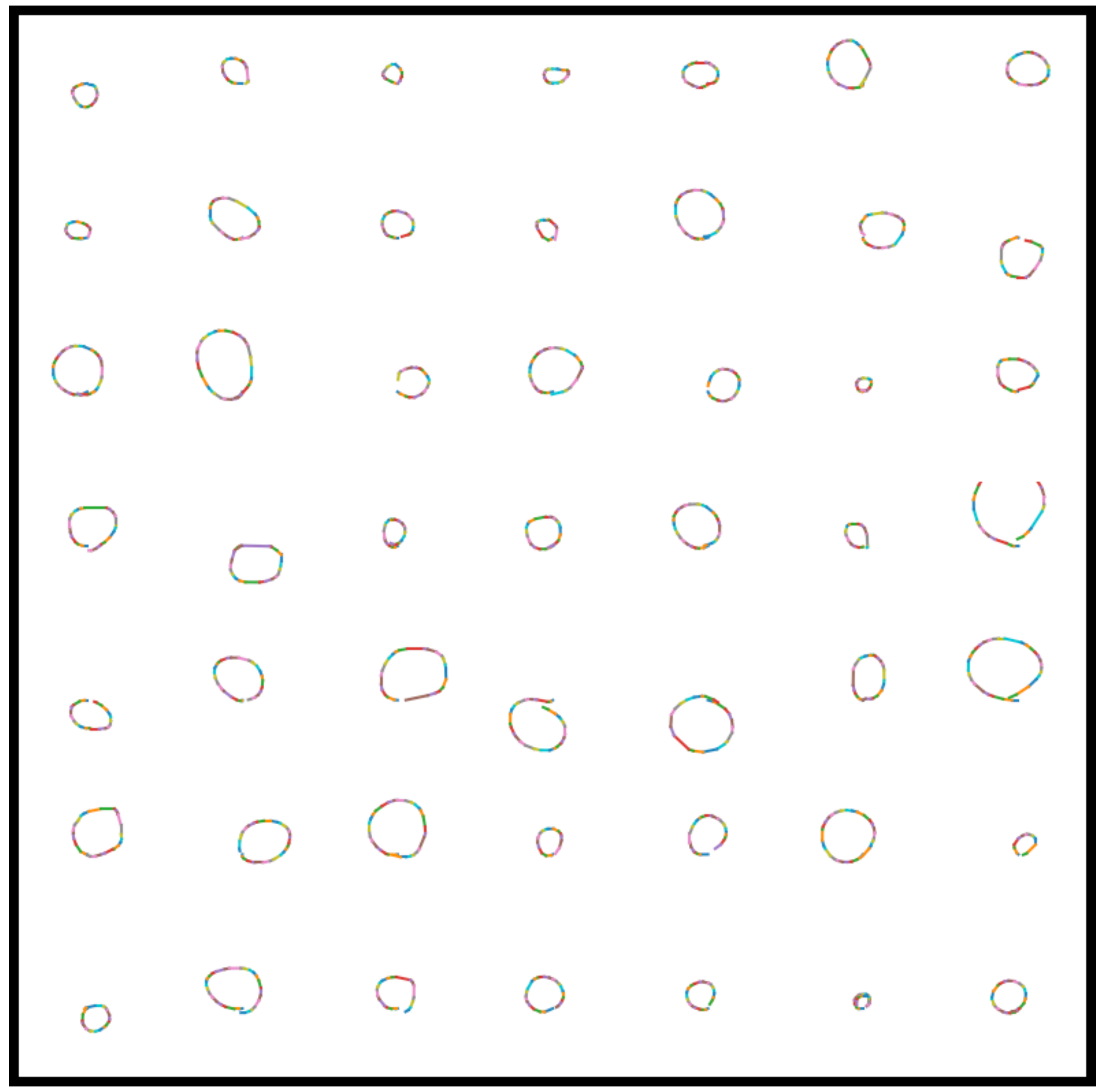
Cat Sketch



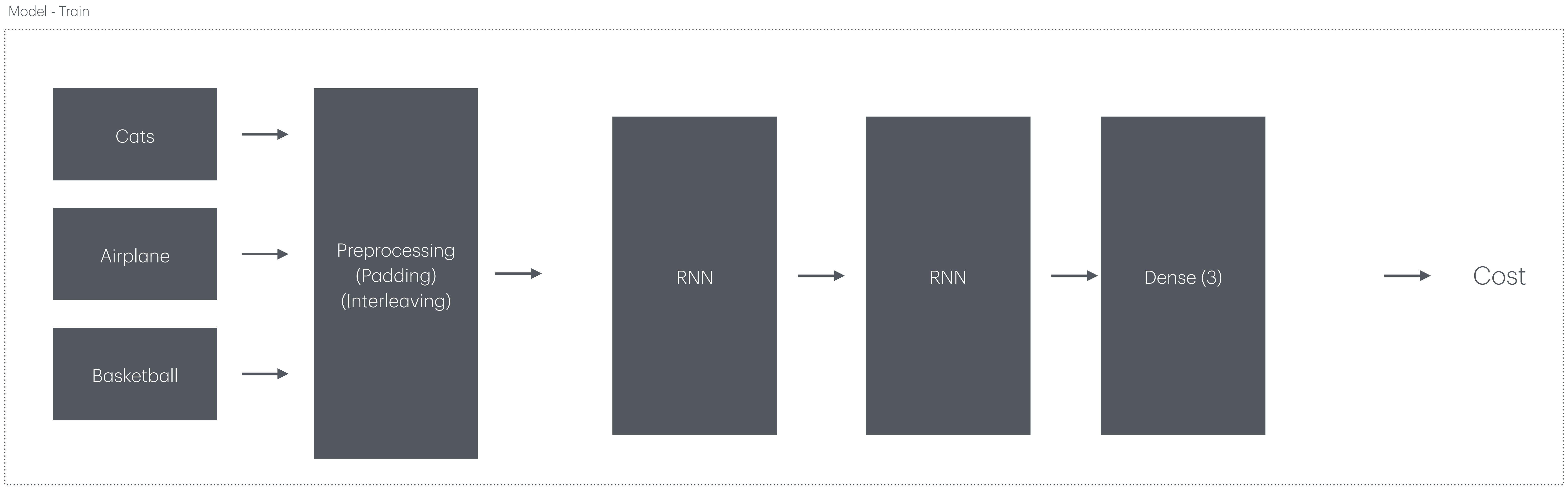
Airplane Sketch

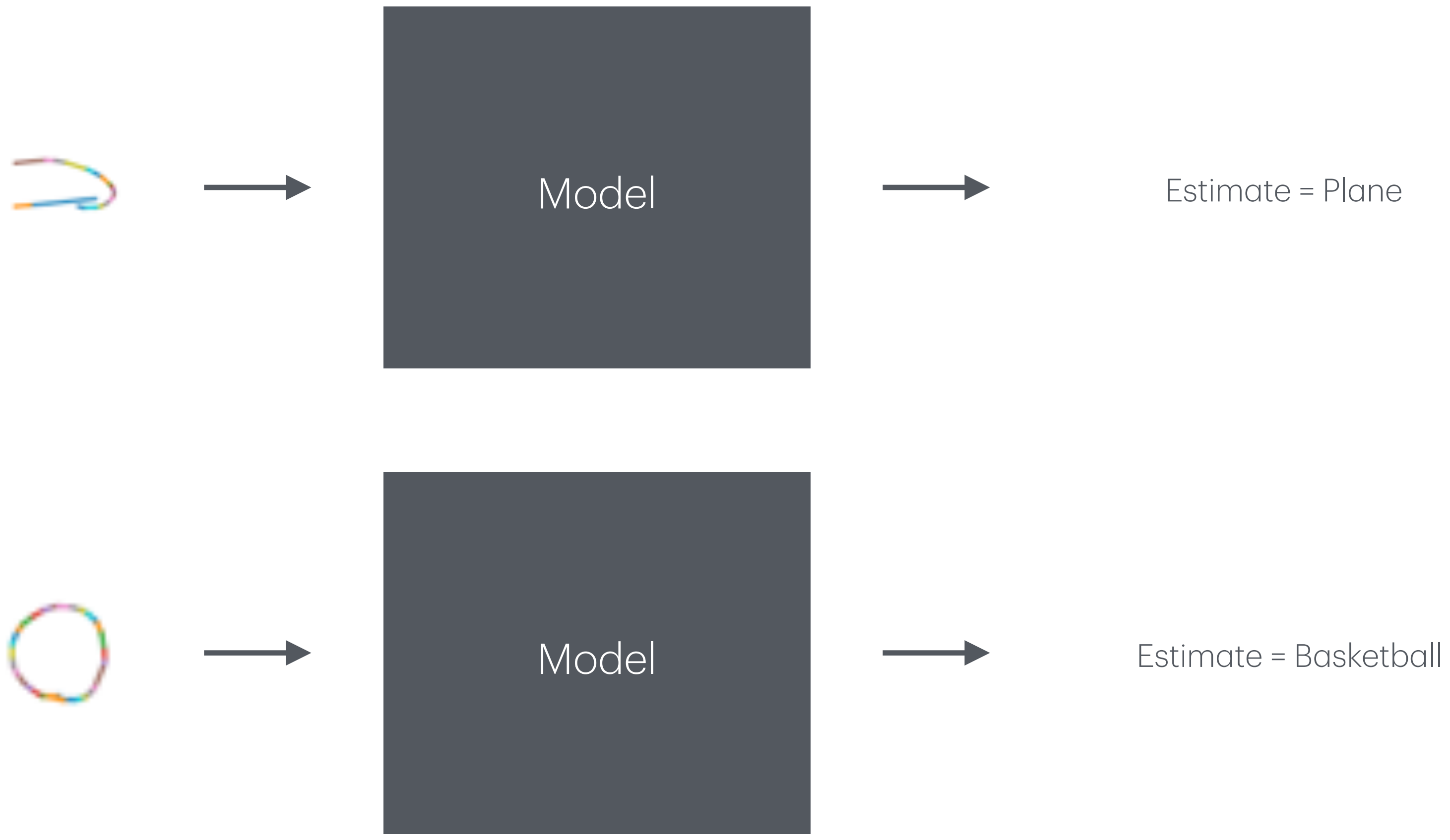


Basketball Sketch



SketchRNN Dataset





TBD