Module outline

Application:

Time series forecasting with IOT data

Model:

Recurrence

Long-short term memory cell

Concepts:

Recurrence

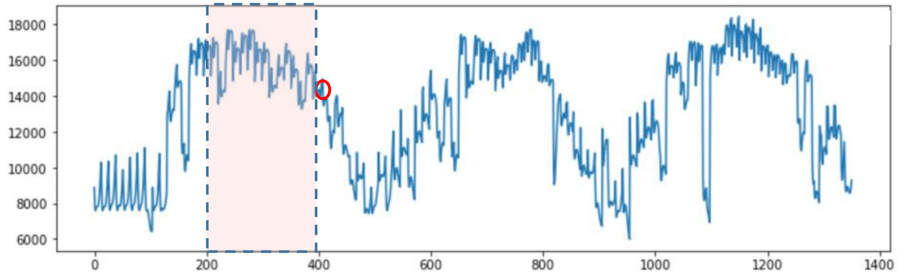
LSTM

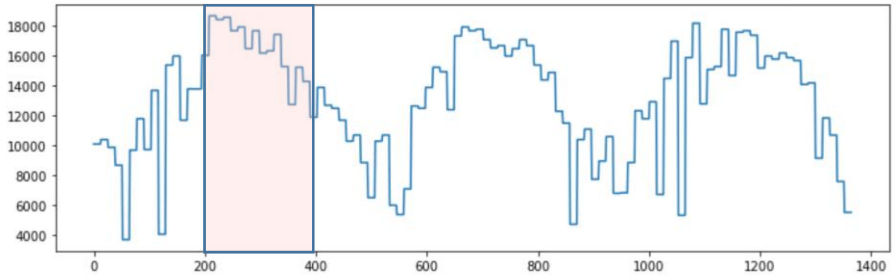
Dropout

Train-Test-Predict Workflow

Sequences (many to one)

Problem: Time series prediction with IOT data

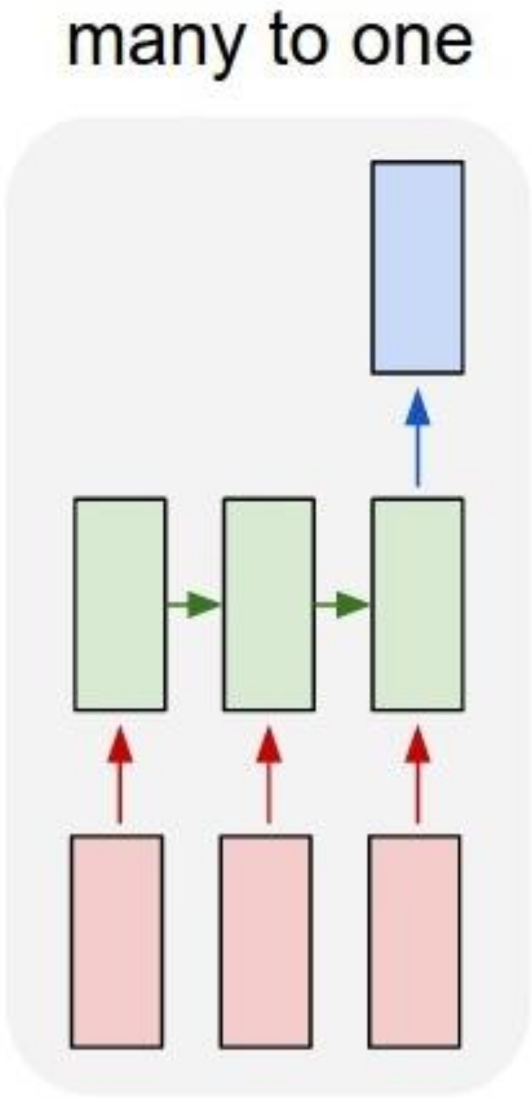




Output
(Y: n x future prediction)

Model
Rec = Recurrence

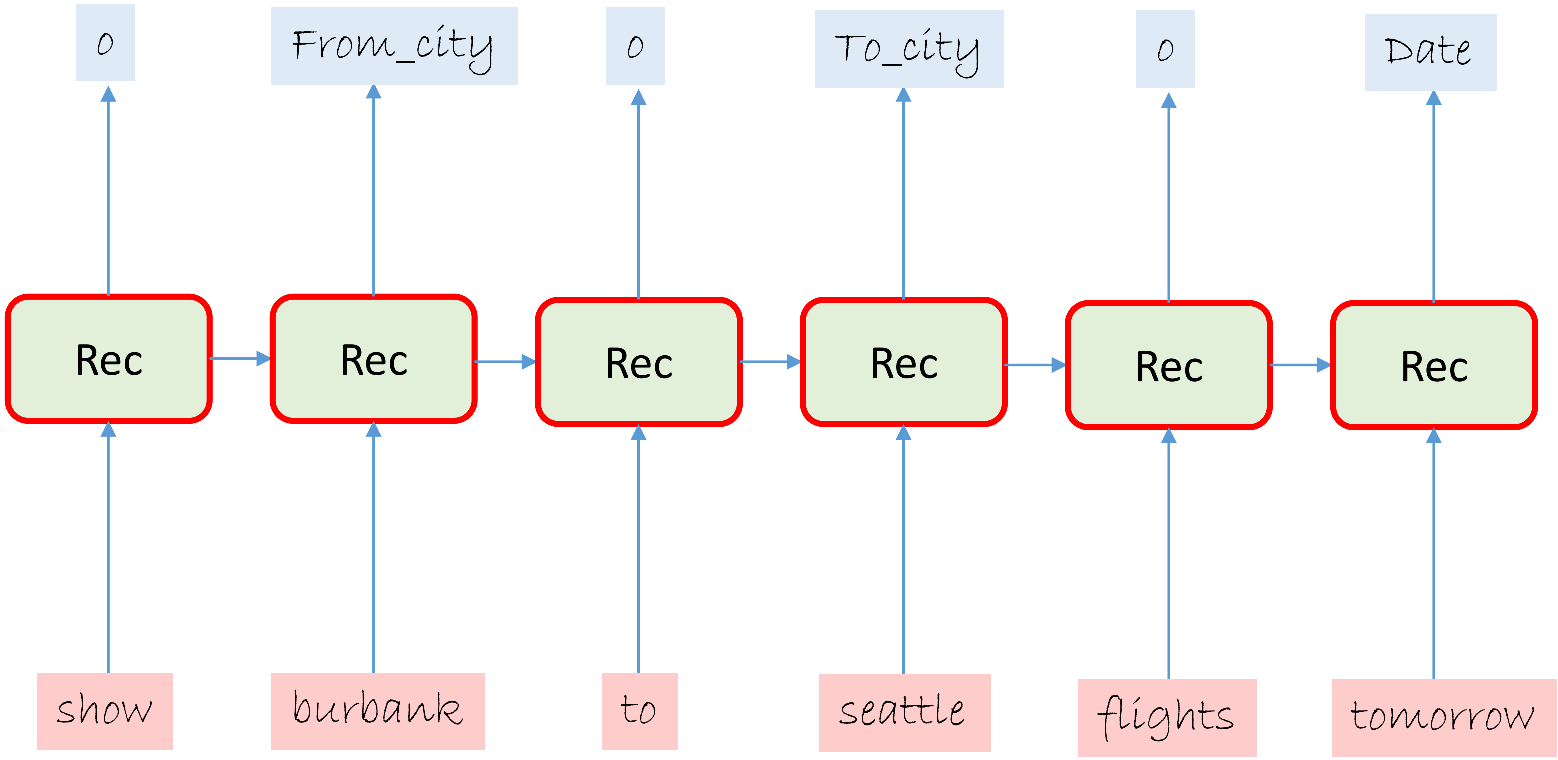
Input feature
(X: n x 14 data pnts)

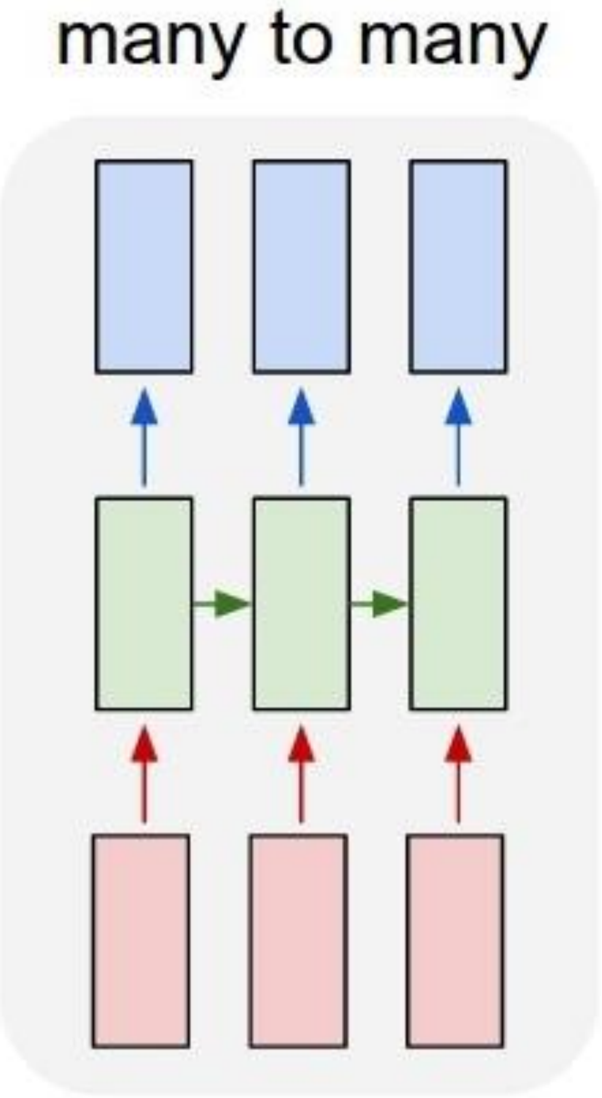


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sequences (many to many + 1:1)

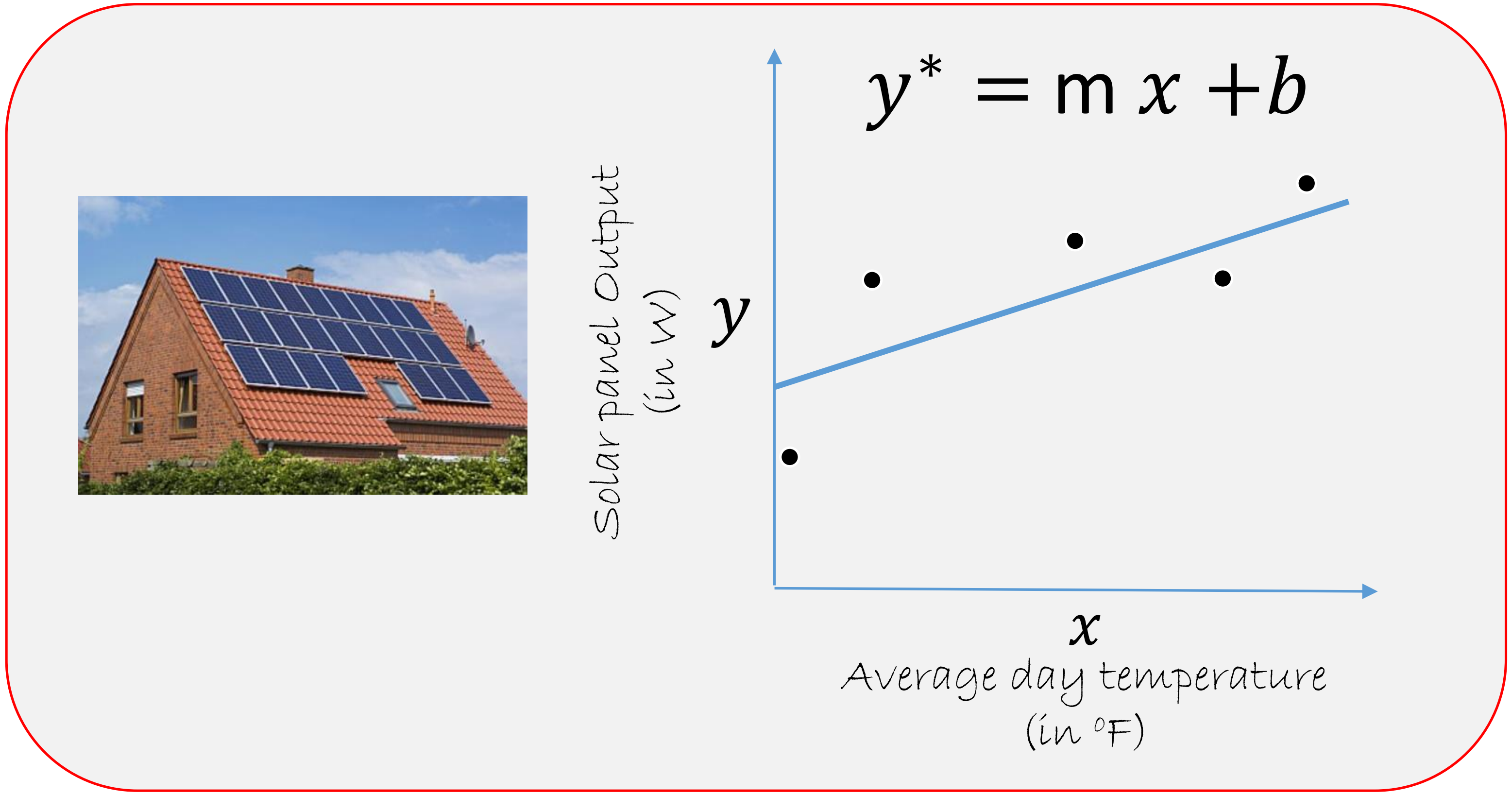
Problem: Tagging entities in Air Traffic Controller (ATIS) data

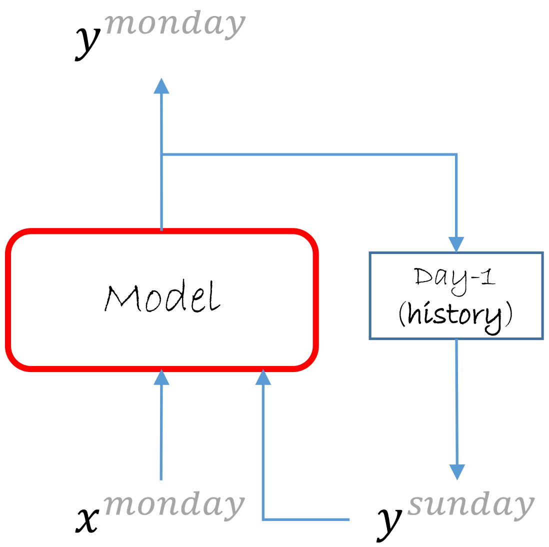




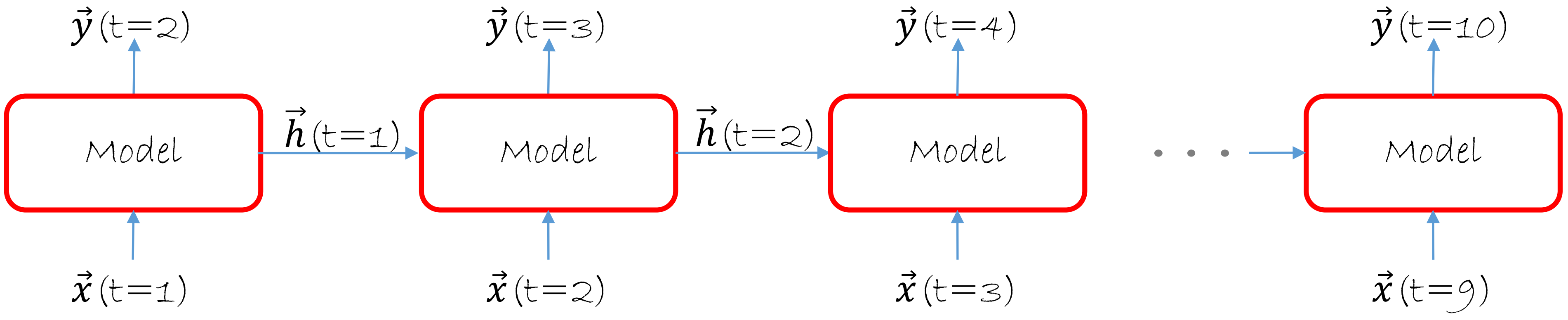
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Forecasting





Recurrence

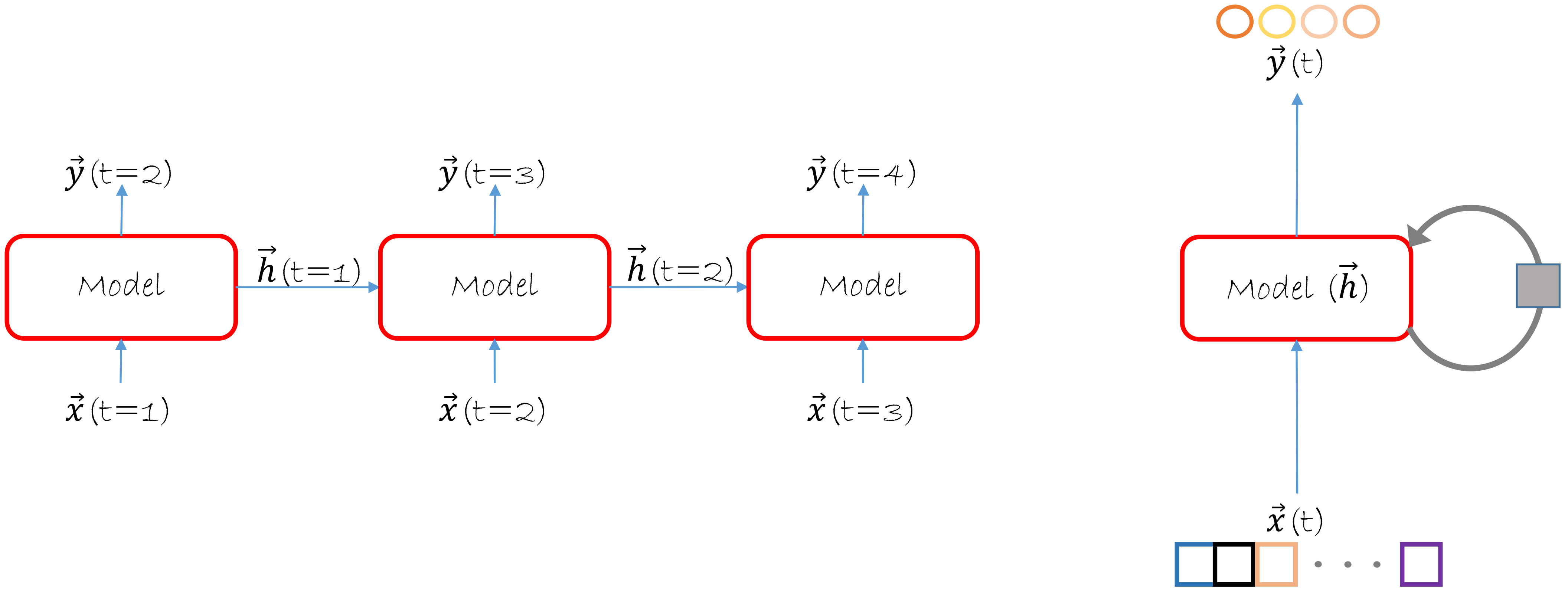


$\vec{x}$(t) : Input (n-dimensional array) at time t
$\vec{y}$(t) : Output (c-dimensional array) at time t
$\vec{h}$(t) : Internal State [m-dimensional array] at time t a.k.a history

Input:

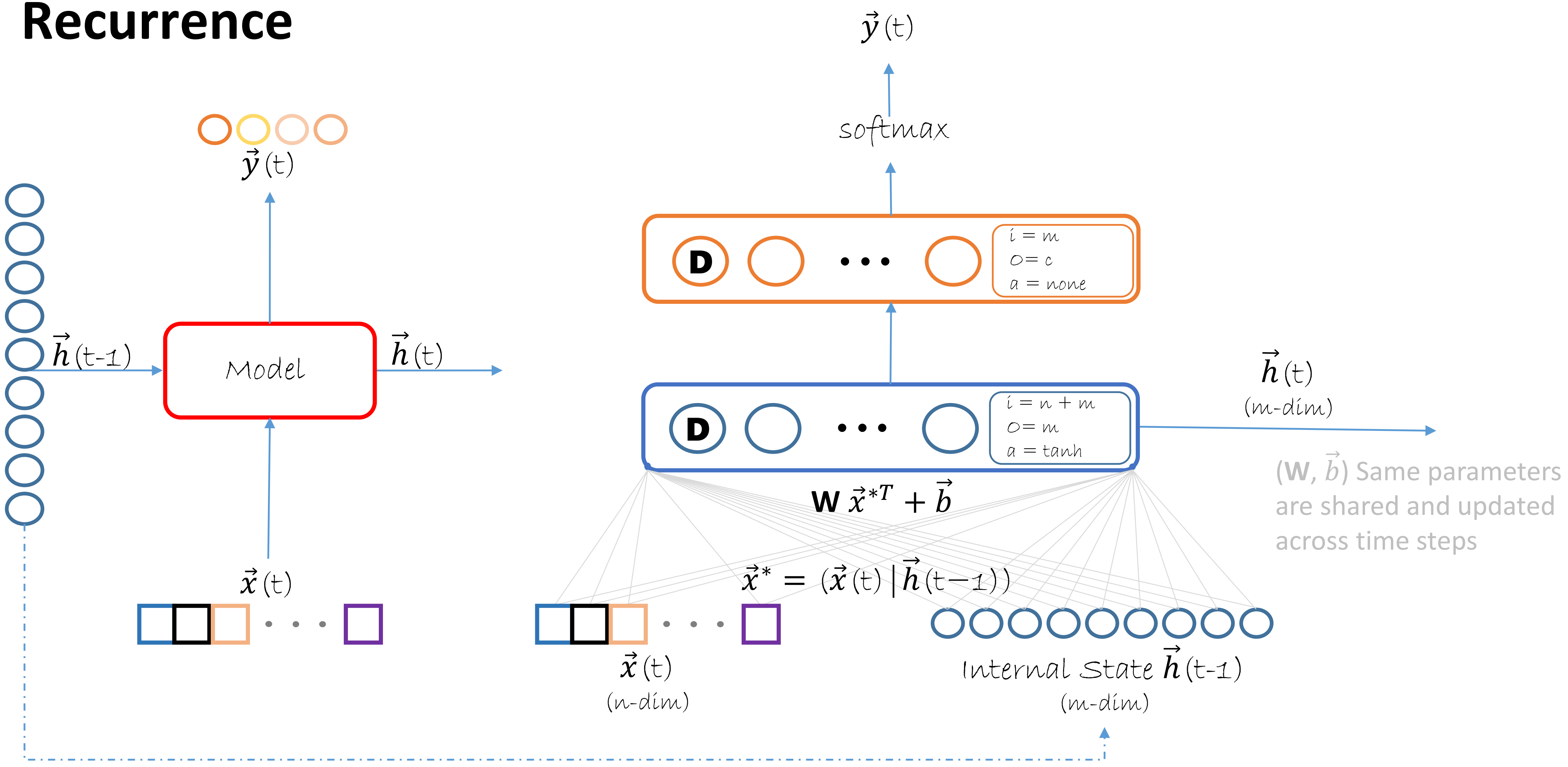
For numeric: Array of numeric values coming from different sensor
For an image: Pixels in an array, Map the image pixels to a compact representation (say n values)
For word in text: Represent words as a numeric vector using embeddings (word2vec or Glove)

Recurrence



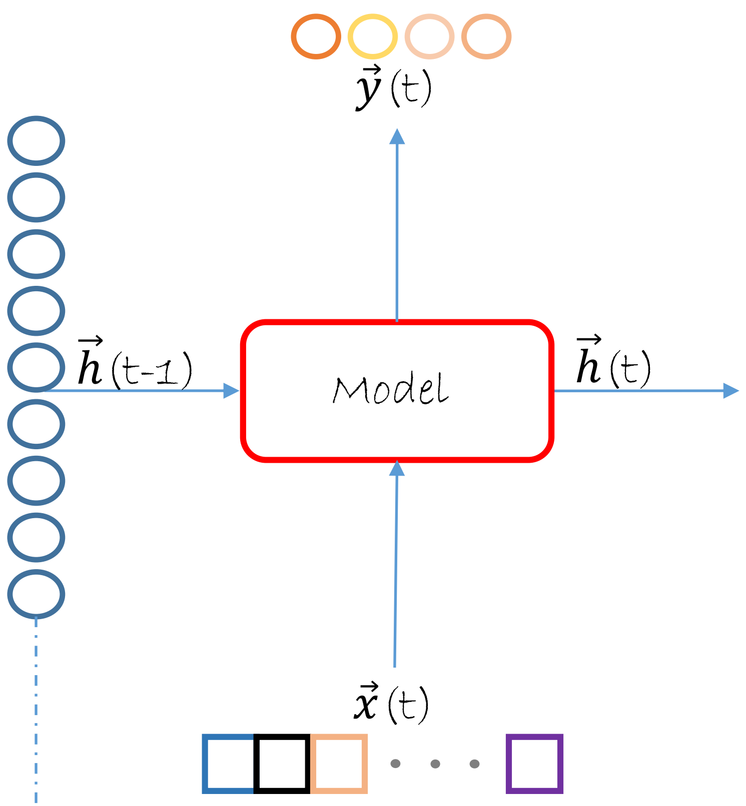
$\vec{y}$ (t=2)
$\vec{y}$ (t=3)
$\vec{y}$ (t=4)
Model
$\vec{h}$ (t=1)
Model
$\vec{h}$ (t=2)
Model
$\vec{x}$ (t=1)
$\vec{x}$ (t=2)
$\vec{x}$ (t=3)
$\vec{y}$ (t)
Model ($\vec{h}$)
$\vec{x}$ (t)

Recurrence

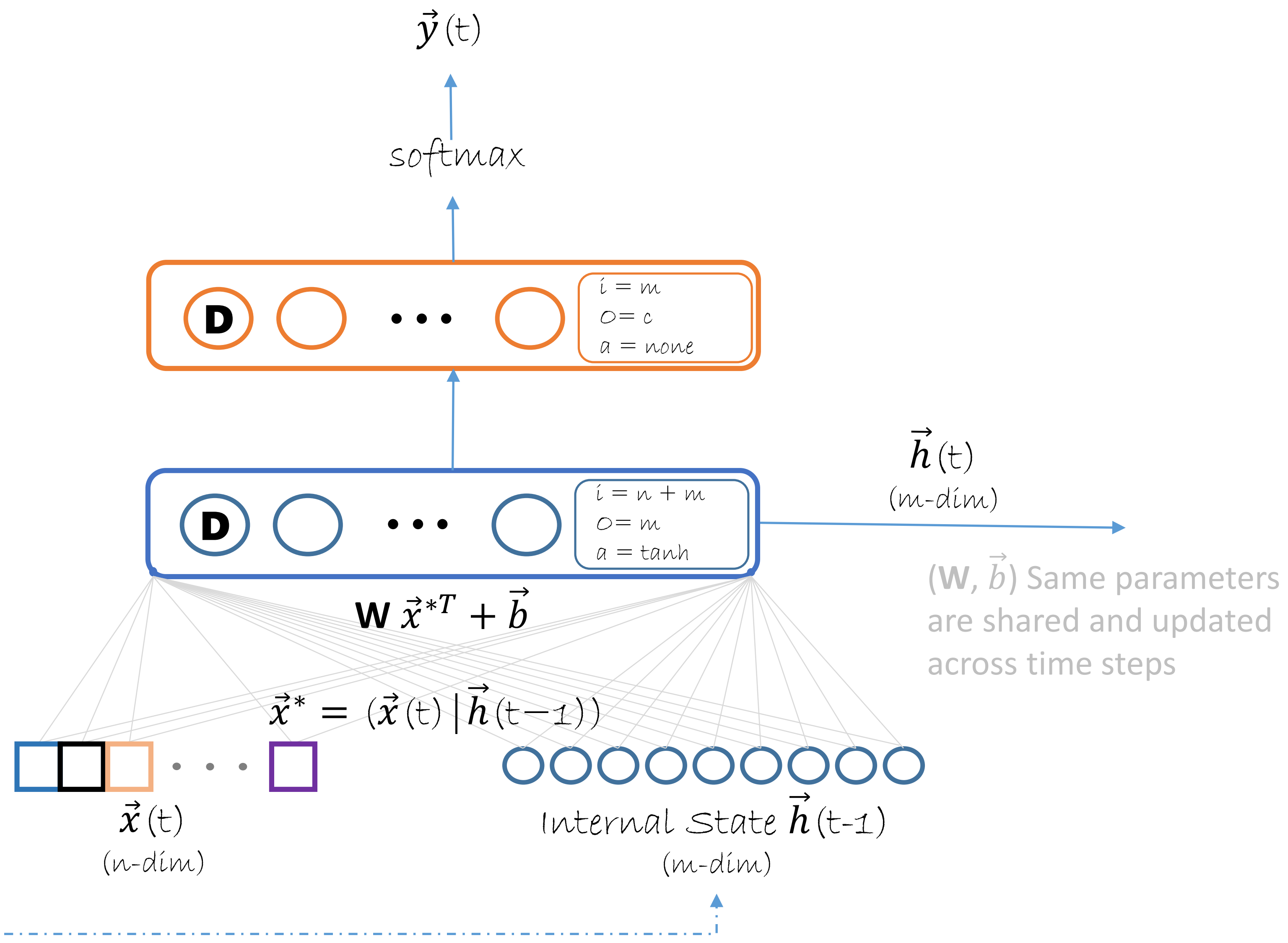


$\vec{y}$(t)
$\vec{h}$(t-1)
Model
$\vec{h}$(t)
$\vec{x}$(t)
$\vec{y}$(t)
softmax
D
$i = m$
$O = c$
$a = none$
D
$i = n + m$
$O = m$
$a = tanh$
$\vec{h}$(t)
(m-dim)
($\mathbf{W}$, $\vec{b}$) Same parameters are shared and updated across time steps
$\mathbf{W}\,\vec{x}^{*T} + \vec{b}$
$\vec{x}^* = (\vec{x}(t)\,|\,\vec{h}(t-1))$
$\vec{x}$(t)
(n-dim)
Internal State $\vec{h}$(t-1)
(m-dim)

Recurrence

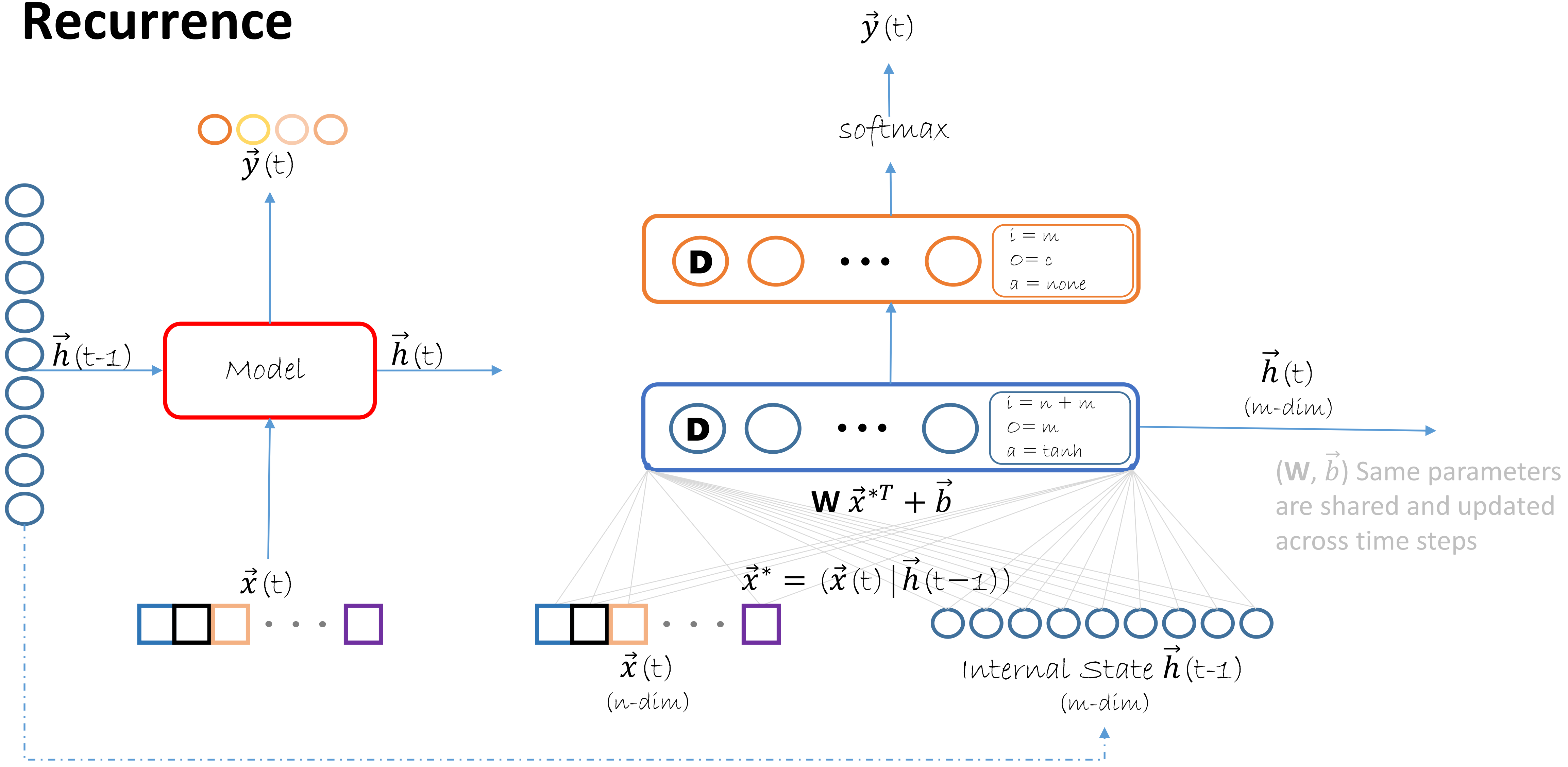


$\vec{y}$(t)
$\vec{h}$(t-1)
Model
$\vec{h}$(t)
$\vec{x}$(t)



$\vec{y}$(t)
softmax
D
$i = m$
$O = c$
a = none
D
$i = n + m$
$O = m$
a = tanh
$\vec{h}$(t)
(m-dim)
(W, $\vec{b}$) Same parameters are shared and updated across time steps
$\mathbf{W}\,\vec{x}^{*T} + \vec{b}$
$\vec{x}^* = (\vec{x}(t)\,|\,\vec{h}(t-1))$
$\vec{x}$(t)
(n-dim)
Internal State $\vec{h}$(t-1)
(m-dim)

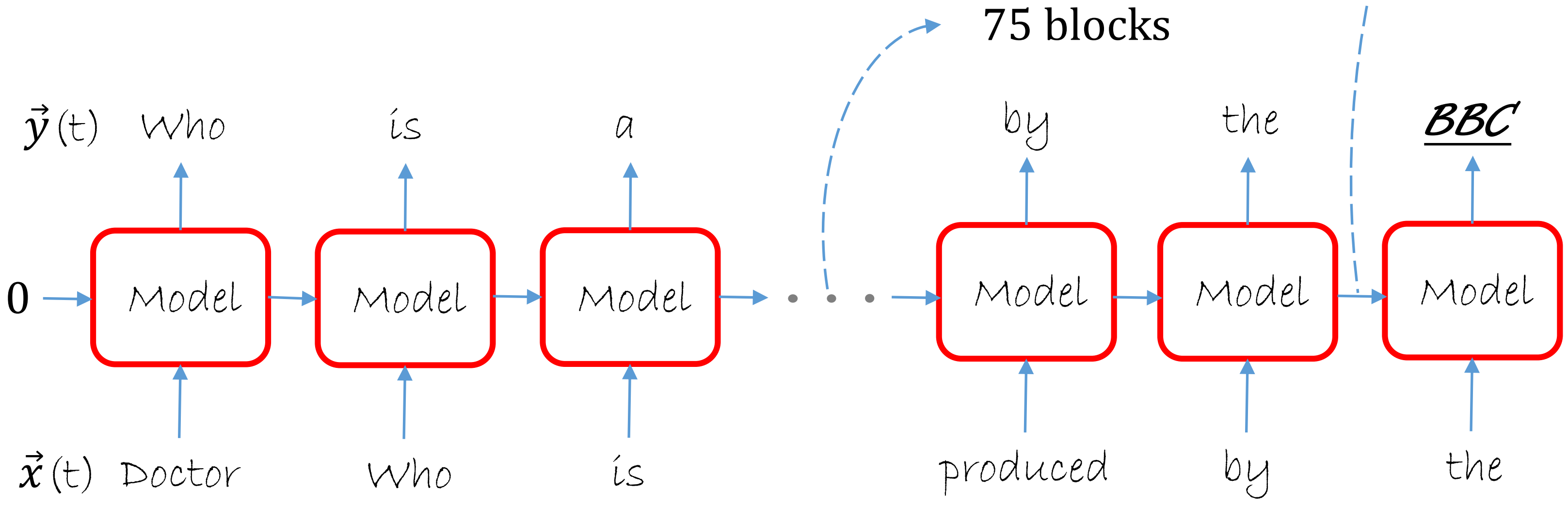
Recurrence

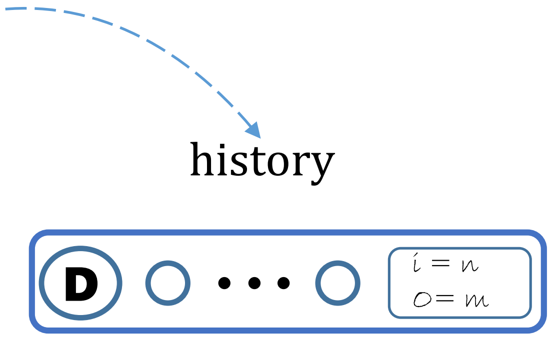


$\vec{y}$(t)
$\vec{h}$(t-1)
Model
$\vec{h}$(t)
$\vec{x}$(t)
$\vec{y}$(t)
softmax
D
$i = m$
$o = c$
$a =$ none
D
$i = n + m$
$o = m$
$a =$ tanh
$\vec{h}$(t)
(m-dim)
$\mathbf{W}\,\vec{x}^{*T} + \vec{b}$
$\vec{x}^* = (\vec{x}(t)\,|\,\vec{h}(t-1))$
$\vec{x}$(t)
(n-dim)
Internal State $\vec{h}$(t-1)
(m-dim)
($\mathbf{W}$, $\vec{b}$) Same parameters are shared and updated across time steps

Recurrence (Vanishing Gradients)

Doctor Who is a British science-fiction television programme produced by the ***BBC*** *since 1963. The programme depicts the adventures of the Doctor, a Time Lord—a space and time-travelling humanoid alien. He explores the universe in his TARDIS, a sentient time-travelling space ship. Accompanied by companions, the Doctor combats a variety of foes, while working to save civilizations and help people in need. This television series produced by the* ...

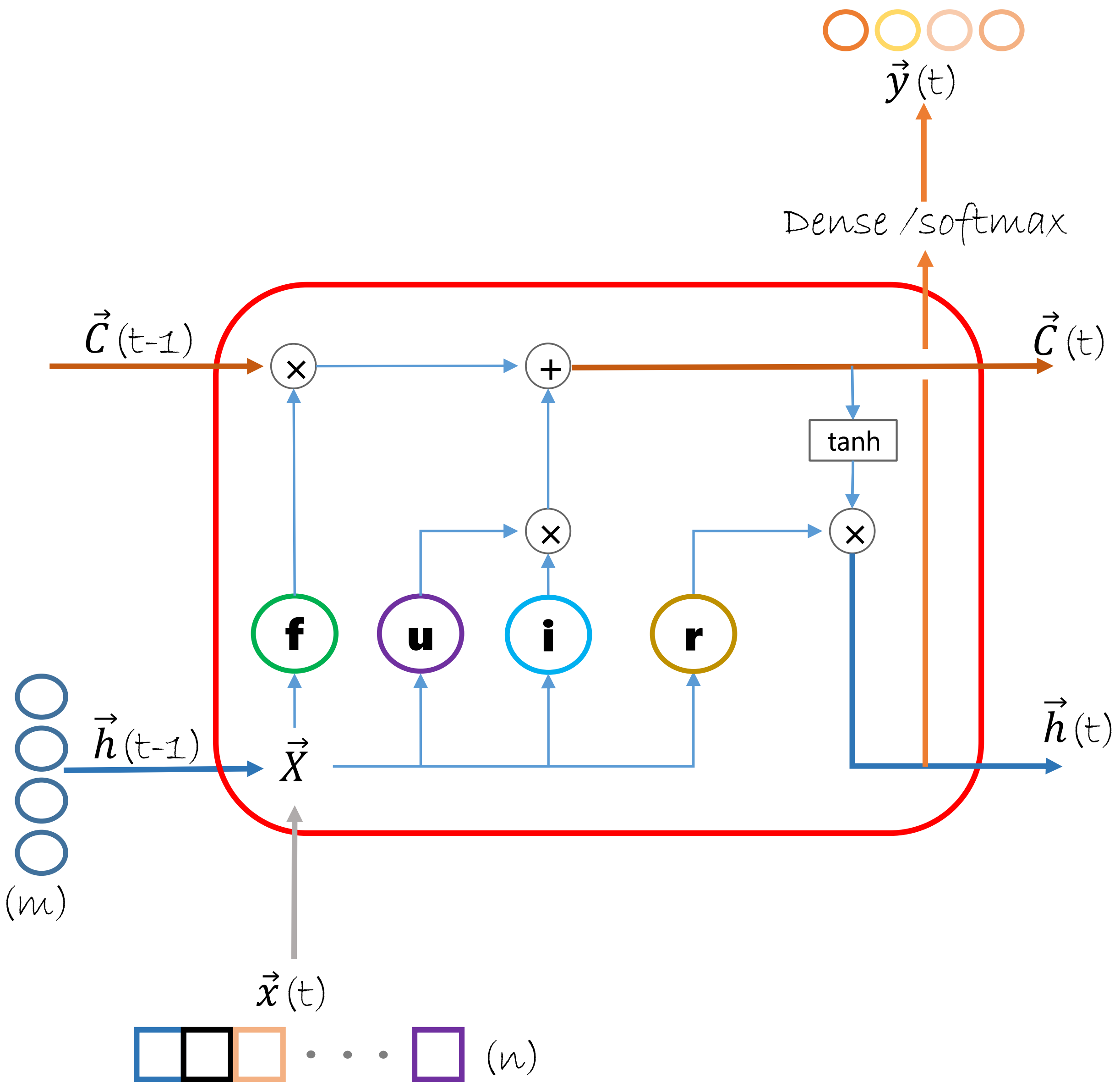




$$\vec{h} = \mathbf{W}\,\vec{x}^T + \vec{b}$$

A single set of ($\mathbf{W}$, $\vec{b}$)
has
limited <u>**memory**</u>

Long-Short Term Memory (LSTM)



Forget gate

f · · · — i = n + m, O = m, Act = sigmoid

$$\vec{f} = \text{sigmoid}(\mathbf{W_f}\,\vec{X}^T + \vec{b_f})$$

Update gate

u · · · — i = n + m, O = m, Act = sigmoid

$$\vec{u} = \text{sigmoid}(\mathbf{W_u}\,\vec{X}^T + \vec{b_u})$$

Input

i · · · — i = n + m, O = m, Act = tanh

$$\vec{X}^* = \tanh(\mathbf{W_i}\,\vec{X}^T + \vec{b_i})$$

Result gate

r · · · — i = n + m, O = m, Act = sigmoid

$$\vec{r} = \text{sigmoid}(\mathbf{W_r}\,\vec{X}^T + \vec{b_r})$$

New cell memory

$$\vec{C}(t) = \vec{C}(t\text{-}1) \times f + i \times u$$

New history

$$\vec{h}(t) = \tanh(\vec{C}(t)) \times r$$

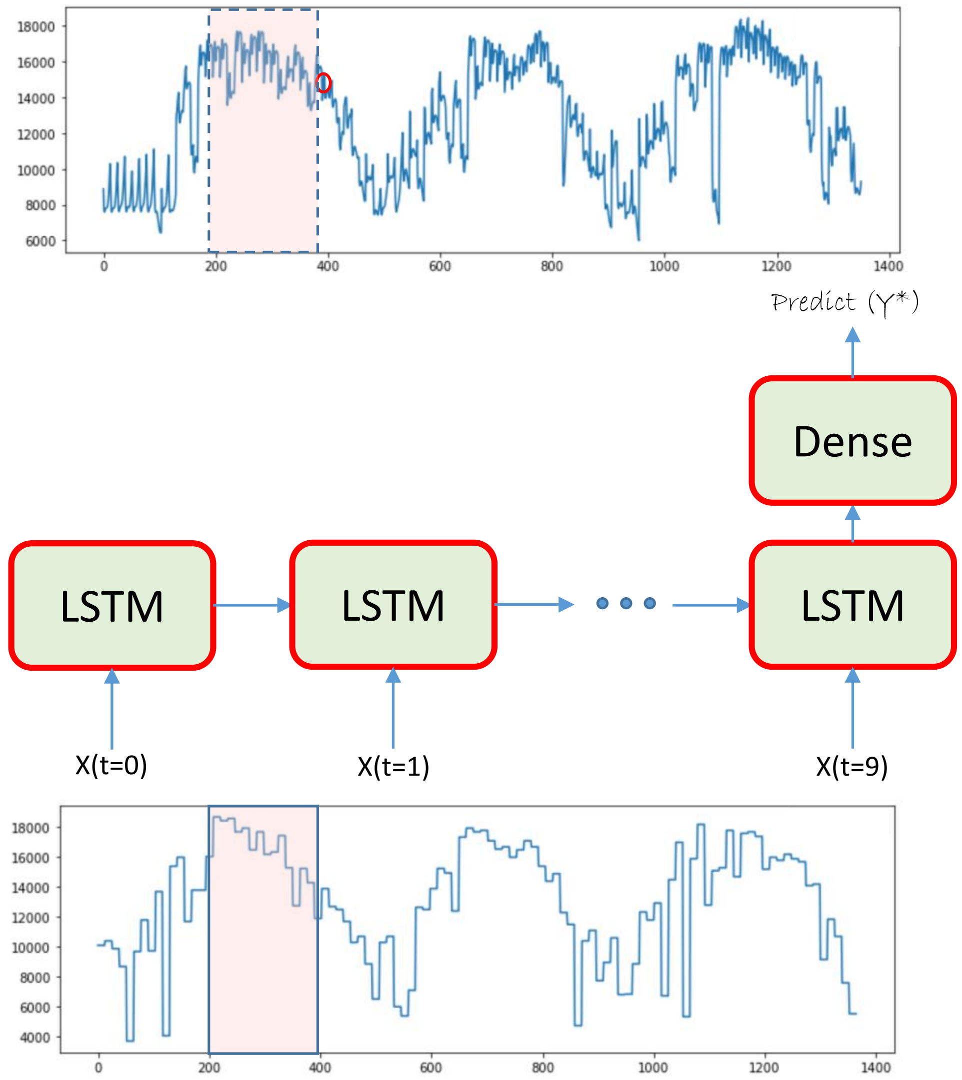
Time-series forecasting

Problem: Time series prediction with IOT data

Output
(Y: n x future prediction)

```
z = create_model(x):
        m = C.layers.Recurrence(C.layers.LSTM(TIMESTEPS))(x)
        m = C.sequence.last(m)
        m = C.layers.Dense(1)(m)
        return m
```

Input feature
(X: n x 14 data pnts)



Dropout

Problem:

Overfitting

Model works great with training data

With new data (unseen during training): high prediction error

Classical Approach:

L1 / L2 regularization

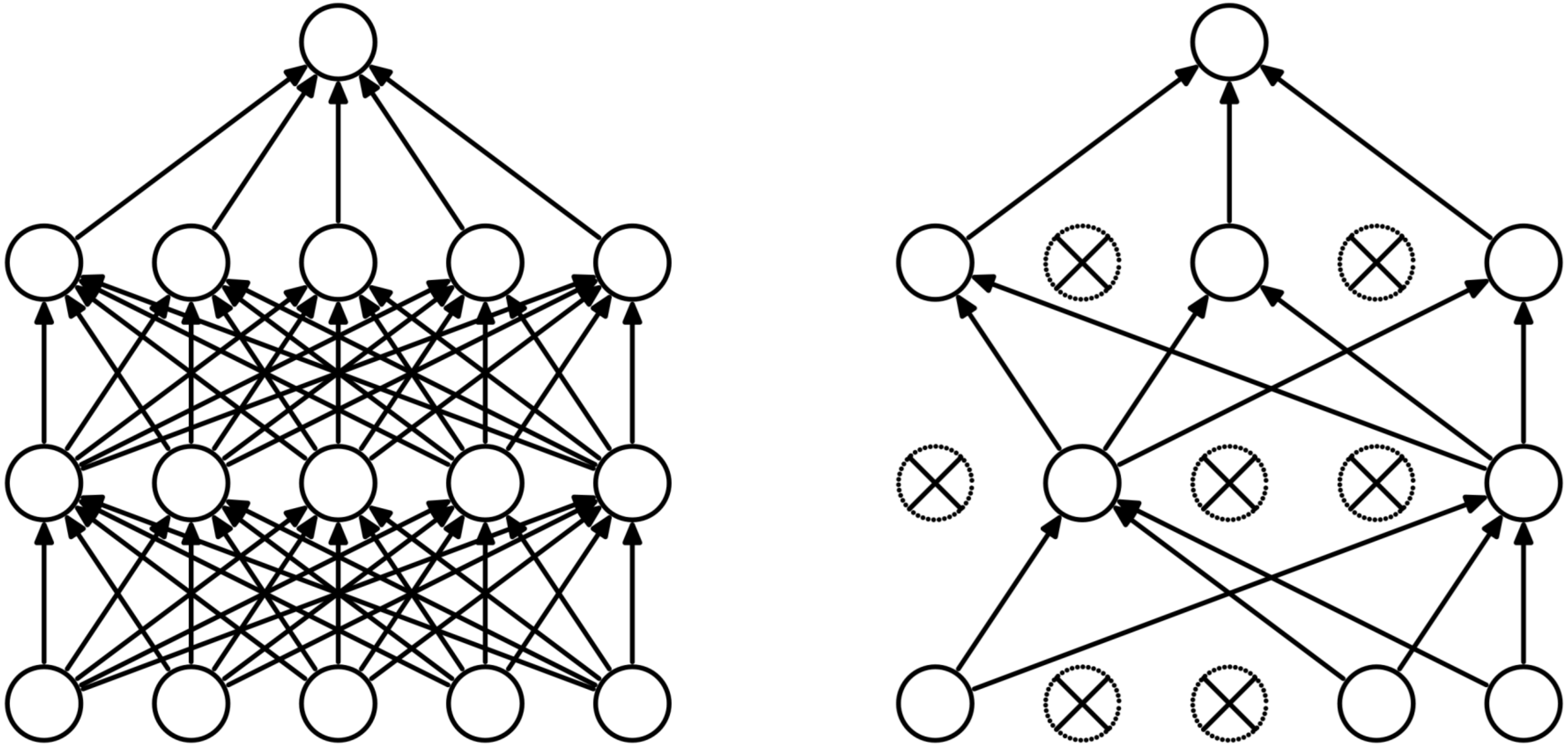
Data augmentation / train with noise added

Early stopping

Dropout

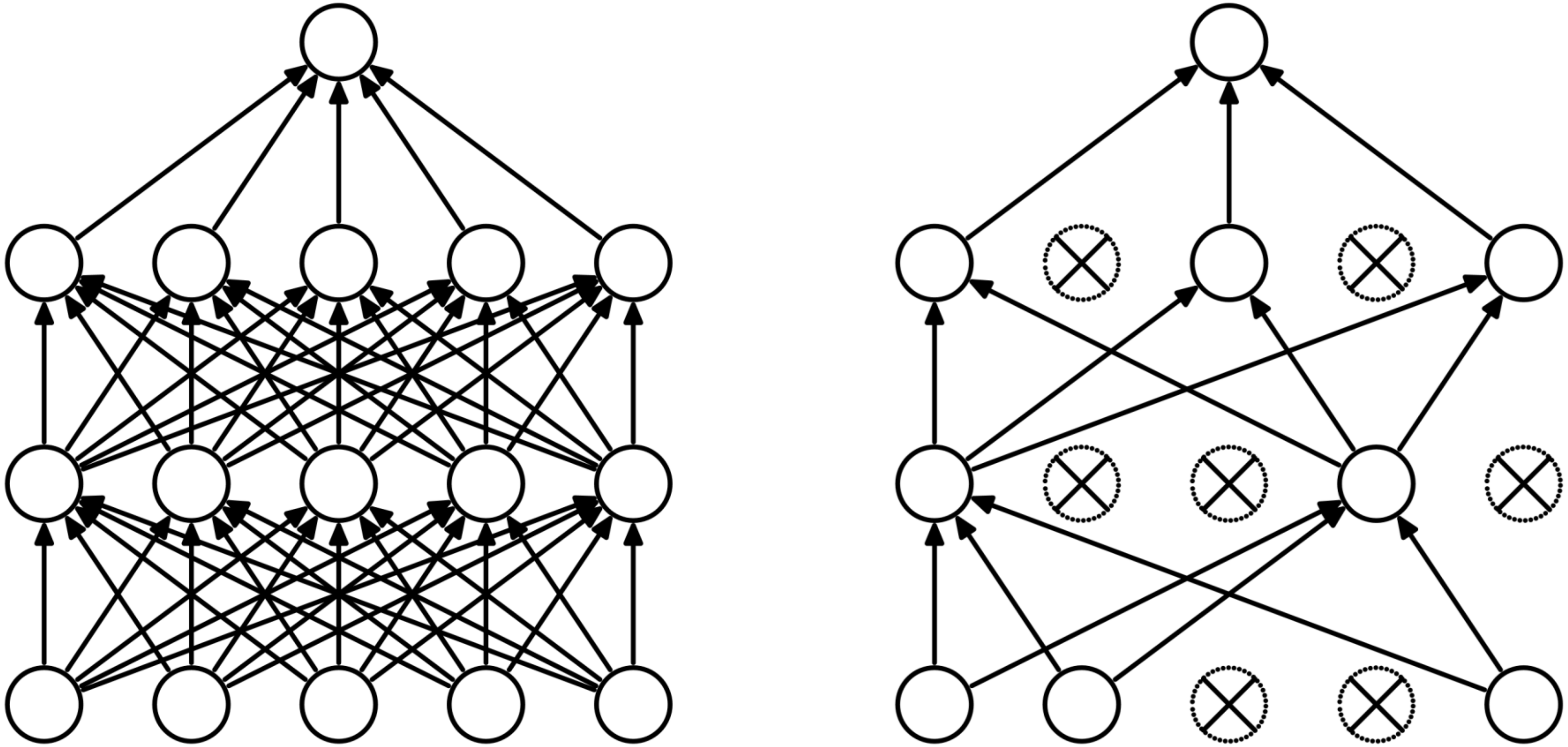
Extremely effective technique to tackle overfitting in neural networks

Dropout



http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

Dropout



http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

Time-series forecasting

IOT data:

- ✓ Output of a solar panel, measurements are recorded at every 30 min interval:
  - `solar.current`: Current production in Watts
  - `solar.total`: Total production for the day so far in Watt/hour

Data Summary:

- ✓ Starting at a time in the day, two values are recorded

```
time,solar.current,solar.total
7am,6.3,1.7
7:30am,44.3,11.4
...
```

- ✓ 3 years of data
- ✓ The input data is not cleansed i.e., errors (panel failed to report) is included

Data pre-processing

Goal:

✓ Compose sequence such that each training instance would be:

- X = [solar.current @ t = 1 – t = 14] (t=1 – 14: corresponds to 1 day)
- Y = Predicted total production for a future day

Pre-processing:

✓ Steps:

- read raw data into a pandas dataframe,
- normalize the data,
- group by day,
- append the columns "solar.current.max" and "solar.total.max", and
- generate the sequences for each day.

✓ Data filtering:

- If X has less than 8 data points – we skip
- If X has more than 14 data points – we truncate

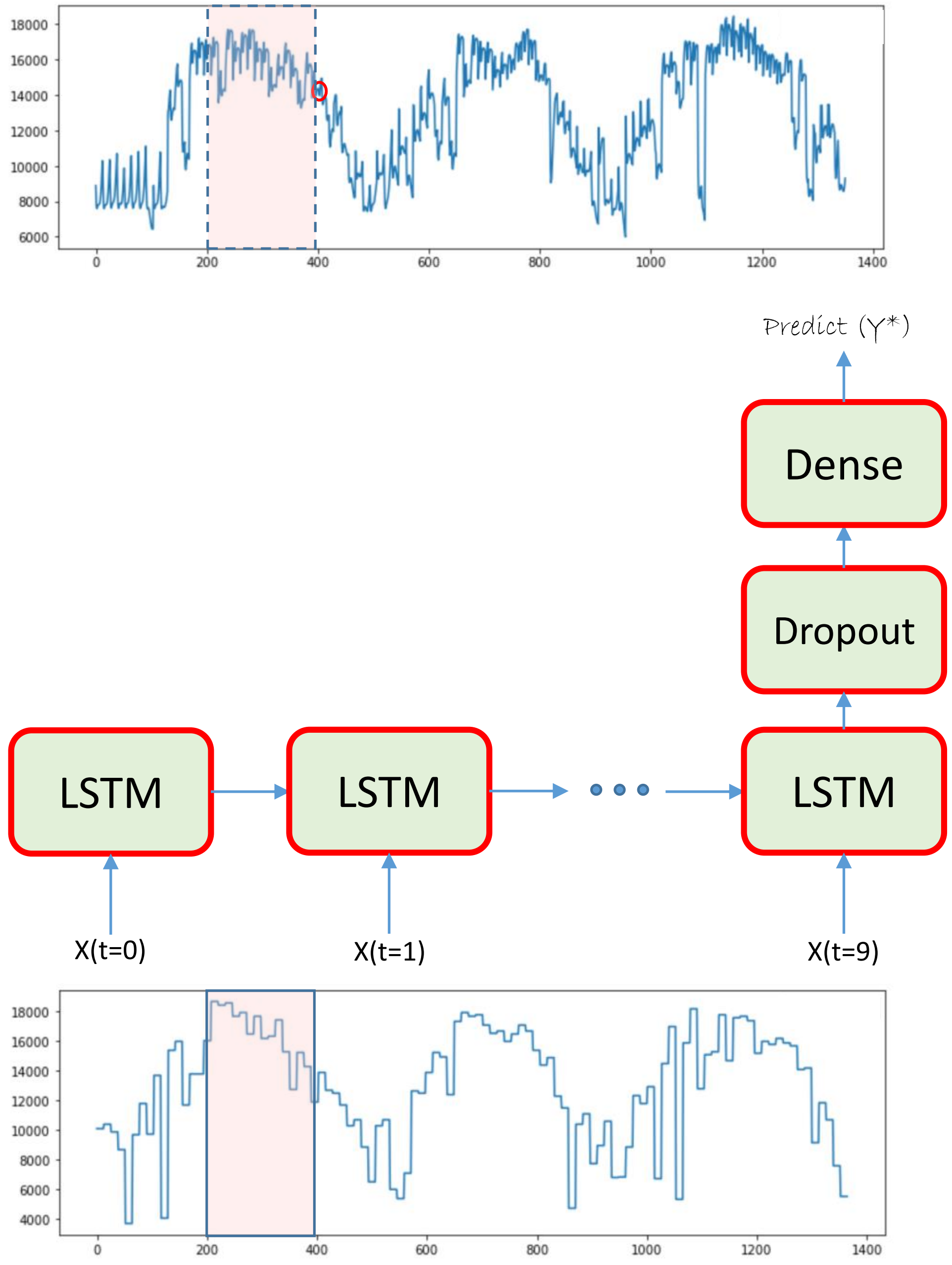
Time-series forecasting

Problem: Time series prediction with IOT data

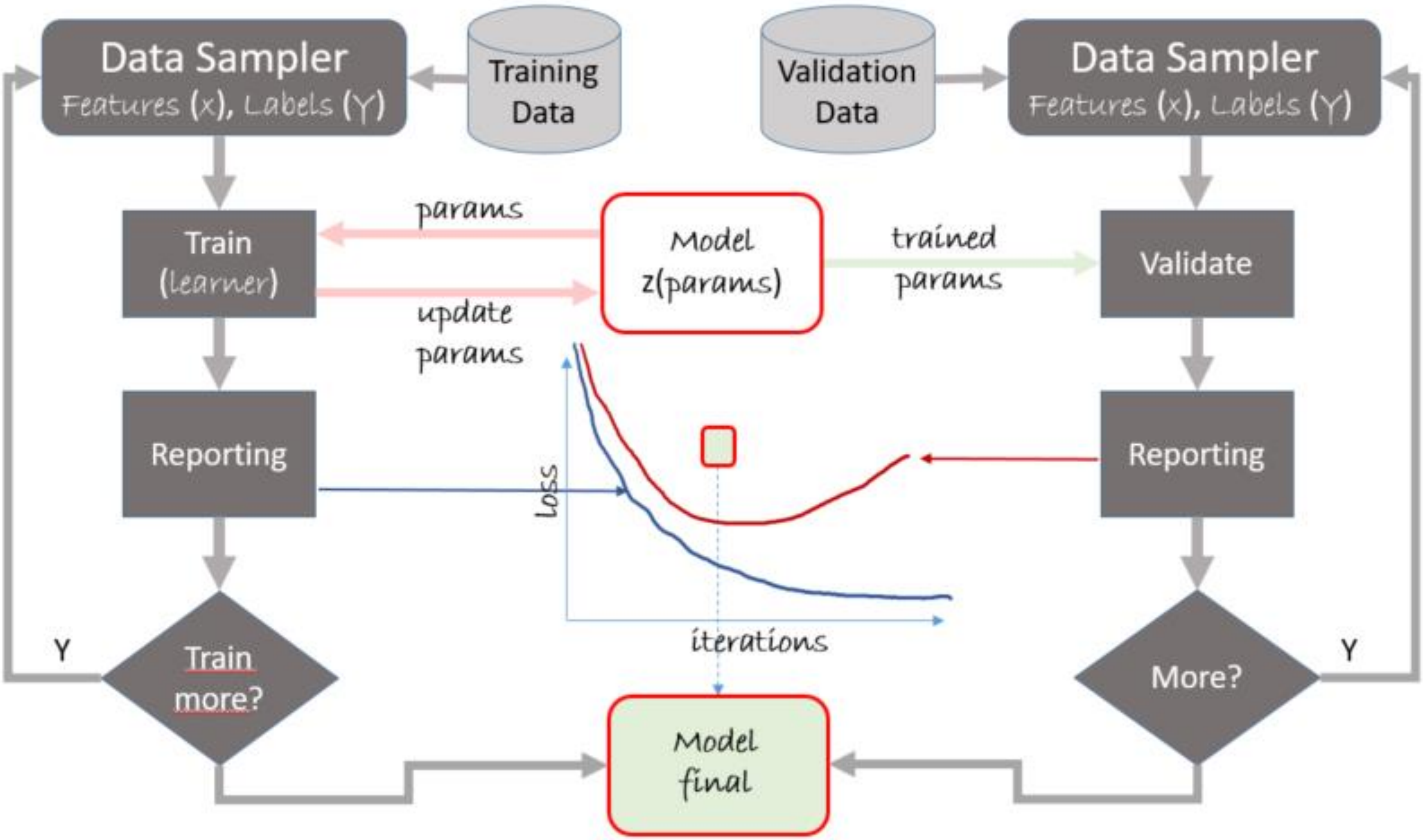
Output
(Y: n x future prediction)

```
z = create_model(x):
        m = C.layers.Recurrence(C.layers.LSTM(TIMESTEPS))(x)
        m = C.sequence.last(m)
        m = C.layers.Dropout(0.2)(m)
        m = C.layers.Dense(1)(m)
        return m
```

Input feature
(X: n x 14 data pnts)

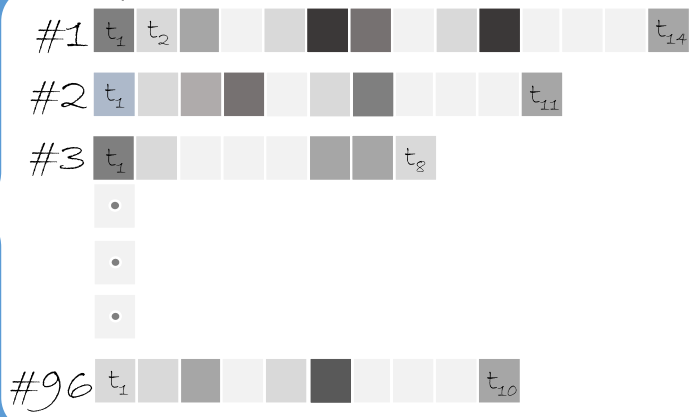


Train / Validation Workflow

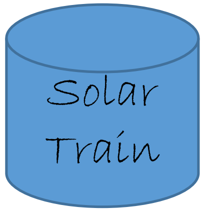


Train workflow

Input feature ($96 \times \vec{x}(t)$)




```
z = create_model(x):
        m = C.layers.Recurrence(C.layers.LSTM(H_DIMS))(x)
        m = C.sequence.last(m)
        m = C.layers.Dropout(0.2)(m)
        m = C.layers.Dense(1)(m)
        return m
```



96 samples
(mini-batch)

Output feature (96×1)
(Y)

Solar panel
output for the day

Loss `squared_error(z,Y)`

Error `squared_error(z,Y)`

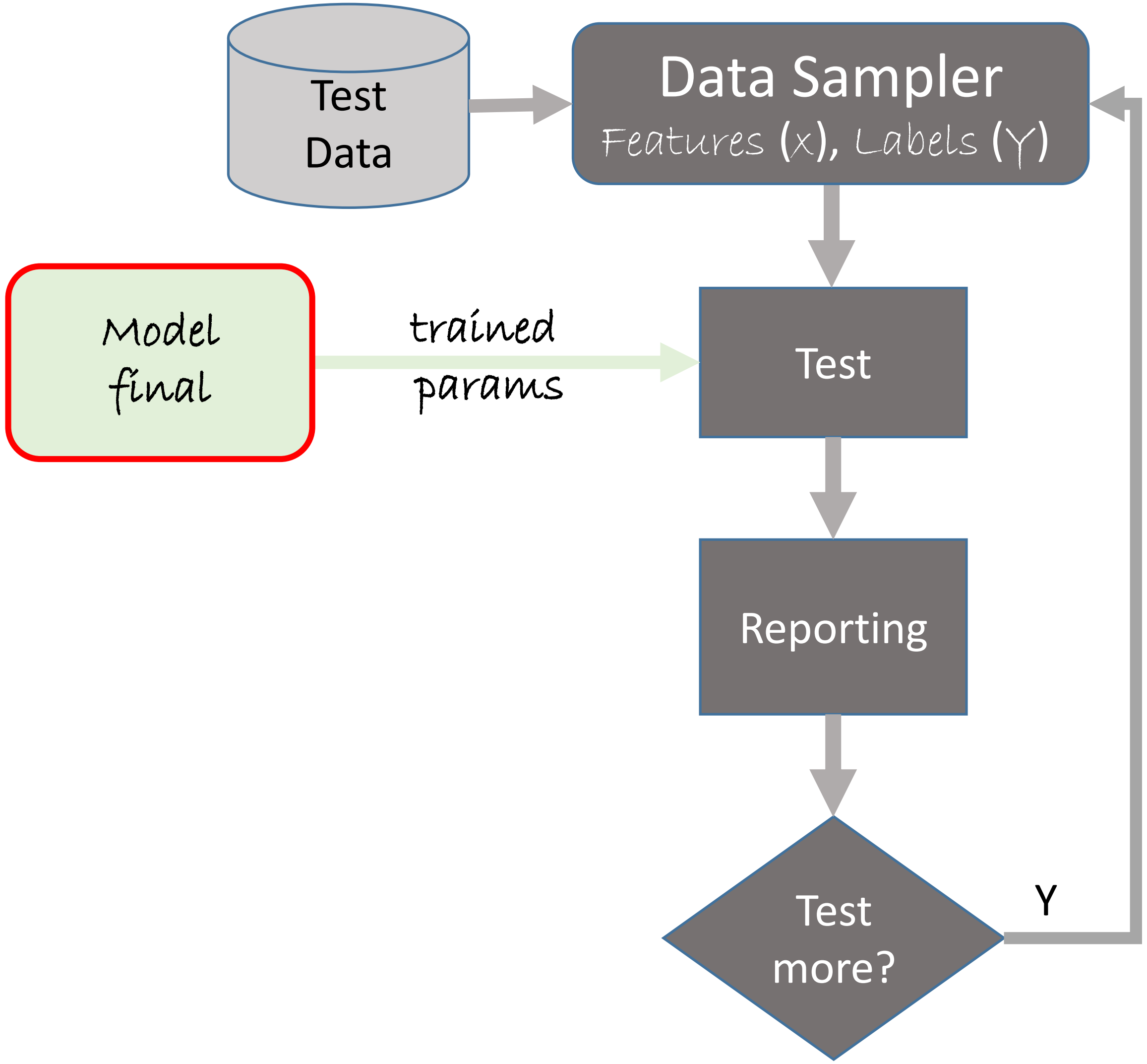
`Trainer(`*model*, (*loss, error*), *learner*`)`

`Trainer.`**`train`**`_minibatch({X, Y})`

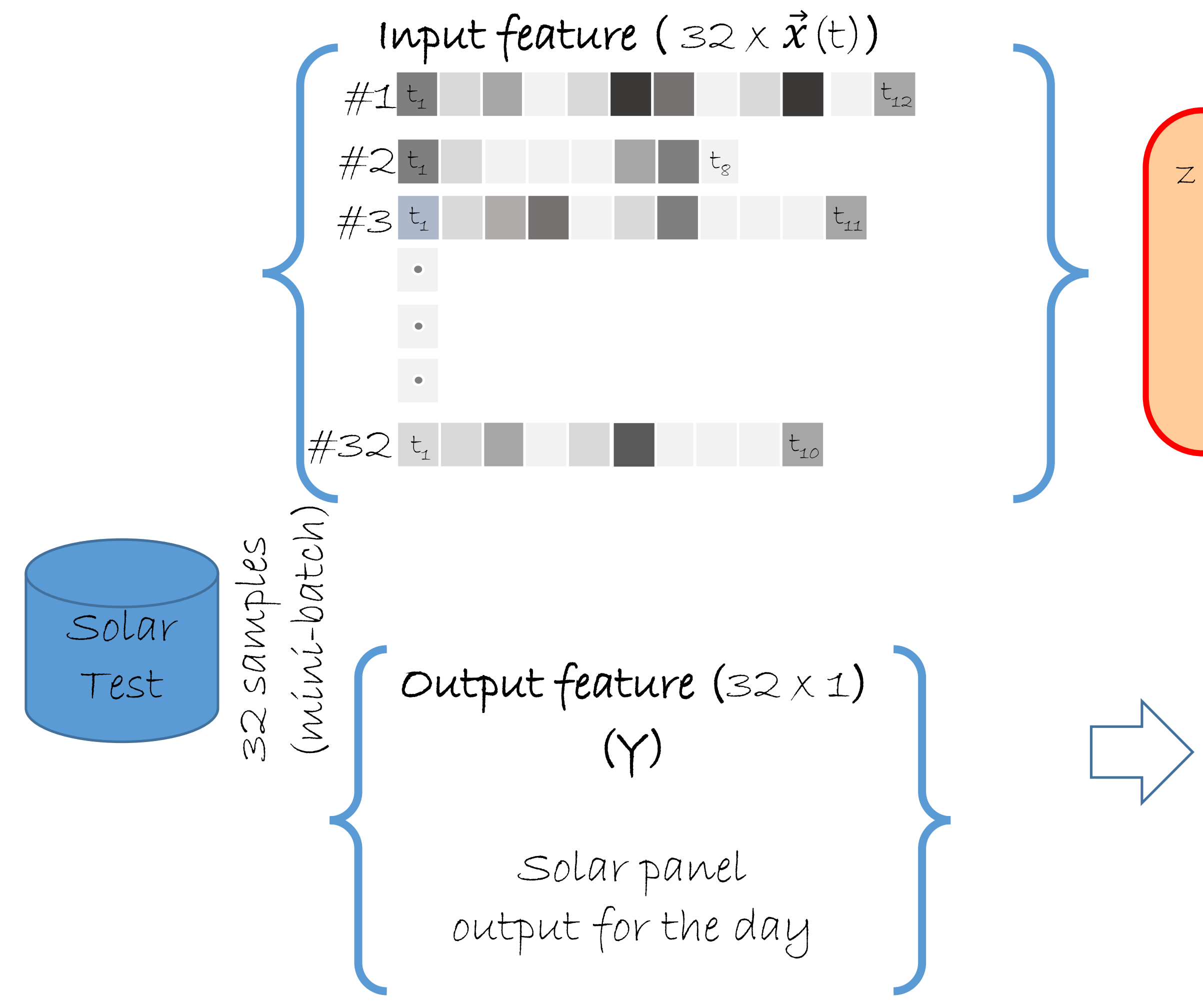
Learner

sgd, adagrad etc, are solvers to estimate

Test workflow



Test workflow




```
z = create_model(x):
      m = C.layers.Recurrence(C.layers.LSTM(H_DIMS))(x)
      m = C.sequence.last(m)
      m = C.layers.Dropout(0.2)(m)
      m = C.layers.Dense(1)(m)
      return m
```

Trainer.**test**_minibatch({X, Y})

Returns the squared error between the observed and predicted output from the solar panel

Prediction workflow

Model
(w, b)

Input feature
(new X: 1 x $\vec{x}(t)$)

Model.eval(new X)

Predicted value of the solar panel output (predicted_label)

[y watts]