
بسم الله الرحمن الرحیم

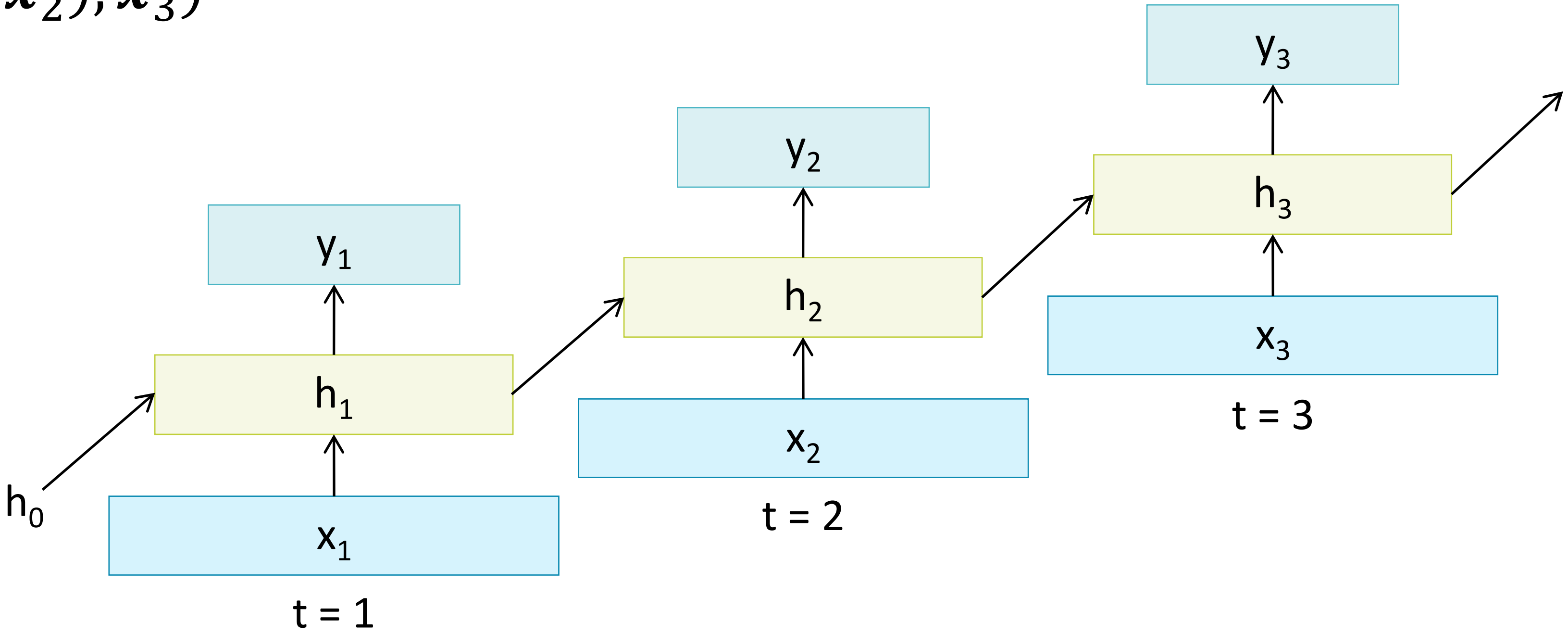
Deep Learning

Mohammad Reza Mohammadi

2021

Recurrent Neural Networks

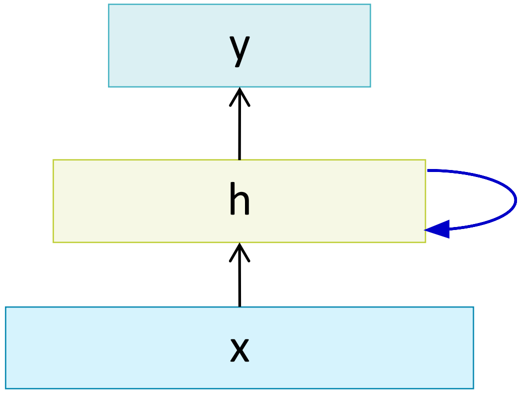
$$\begin{aligned}\boldsymbol{h}_3 &= f_W(\boldsymbol{h}_2, \boldsymbol{x}_3)\\ &= f_W(f_W(\boldsymbol{h}_1, \boldsymbol{x}_2), \boldsymbol{x}_3)\\ &= f_W(f_W(f_W(\boldsymbol{h}_0, \boldsymbol{x}_1), \boldsymbol{x}_2), \boldsymbol{x}_3)\\ &= g^{(3)}(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)\end{aligned}$$



Recurrent Neural Networks

- We can process a sequence of vectors x by applying a recurrence formula at every time step
- The same function and the same set of parameters are used at every time step

$$\underbrace{\boxed{\boldsymbol{h}_t}}_{\text{new state}} = \overbrace{\boxed{f_W}}^{\text{some function with parameters } W}\big(\underbrace{\boxed{\boldsymbol{h}_{t-1}}}_{\text{old state}}, \overbrace{\boxed{\boldsymbol{x}_t}}^{\text{input vector at time } t}\big)$$



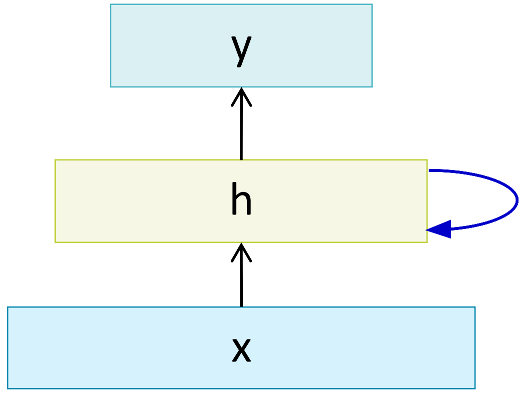
(Simple) RNN

- The state consists of a single "hidden" vector h
- Sometimes called a "Vanilla RNN"

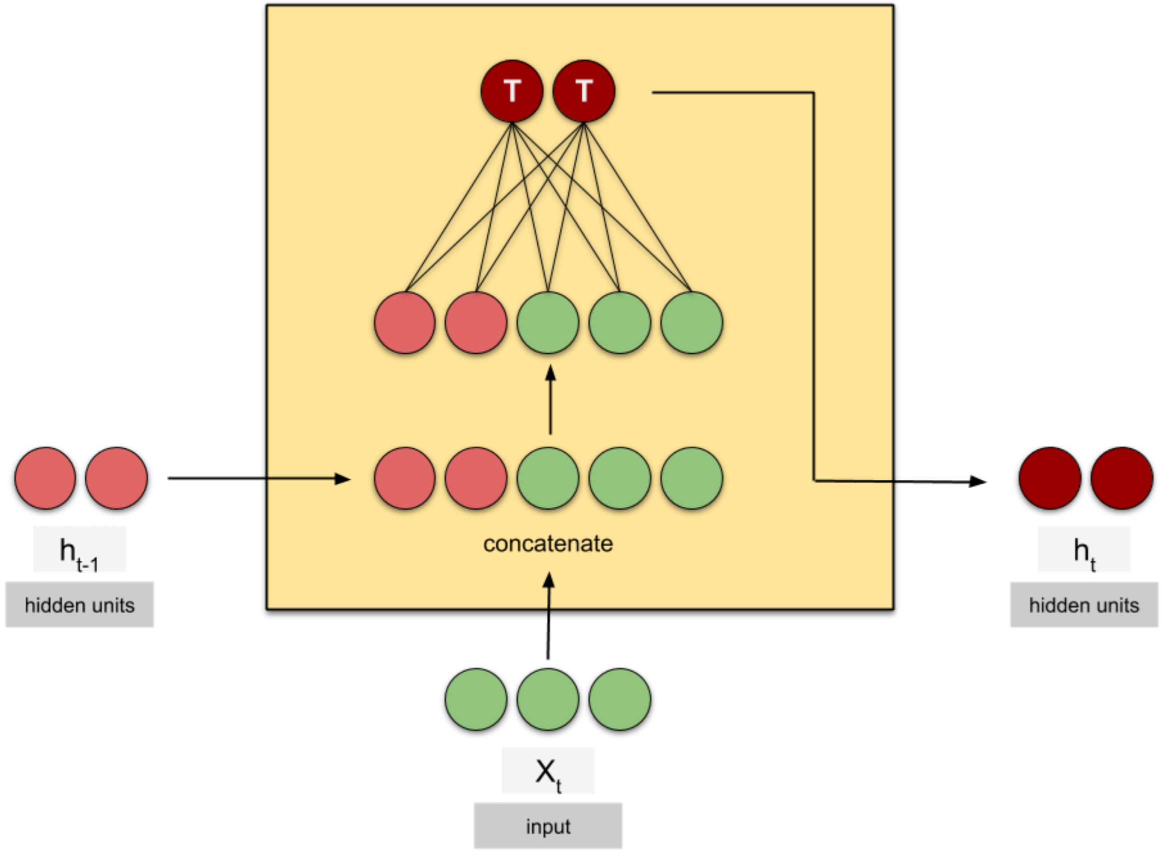
$$h_t = f_W(h_{t-1}, x_t)$$

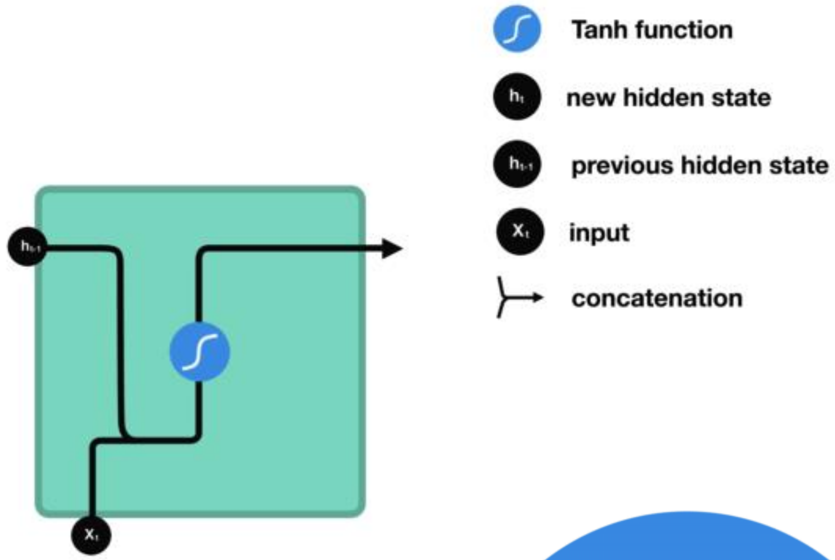
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

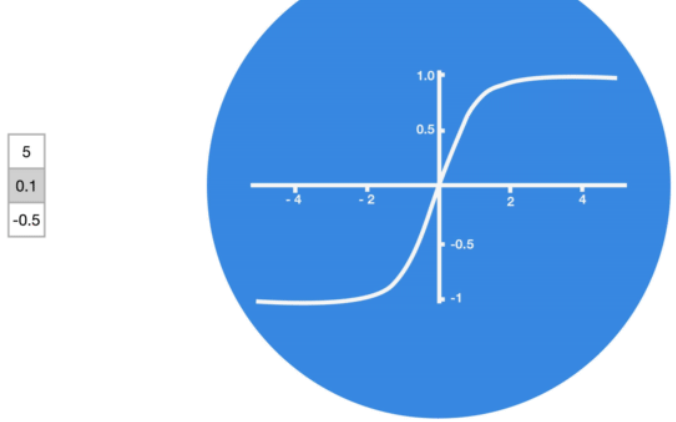
$$y_t = W_{hy} h_t$$



(Simple) RNN



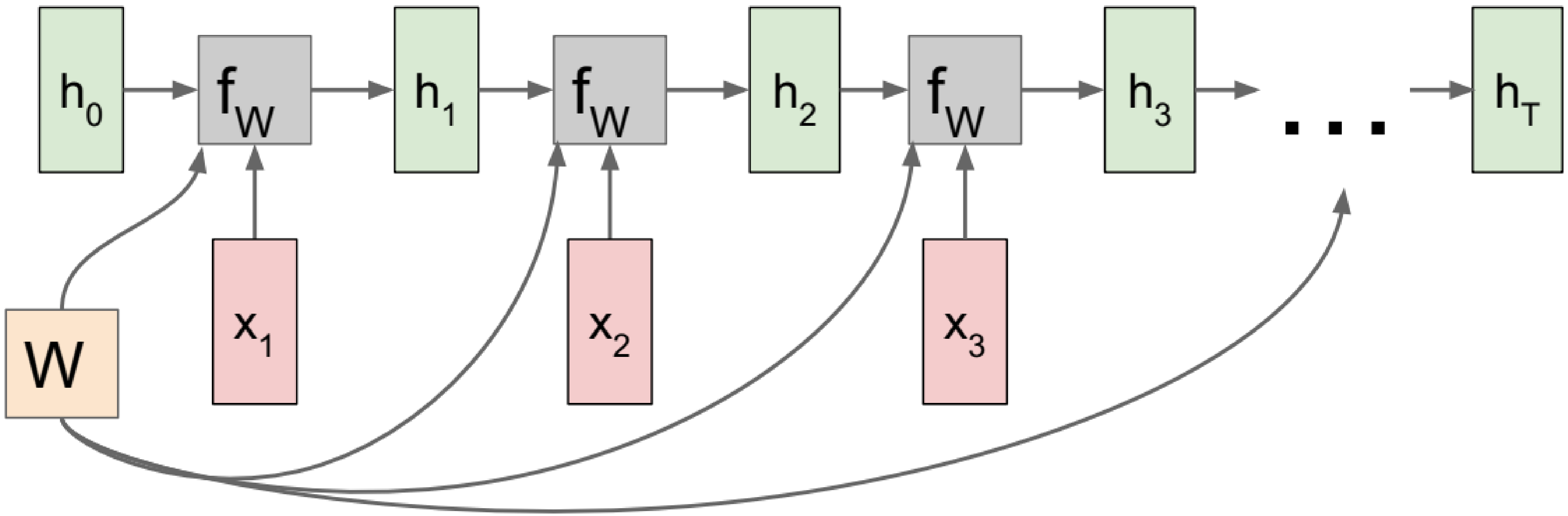




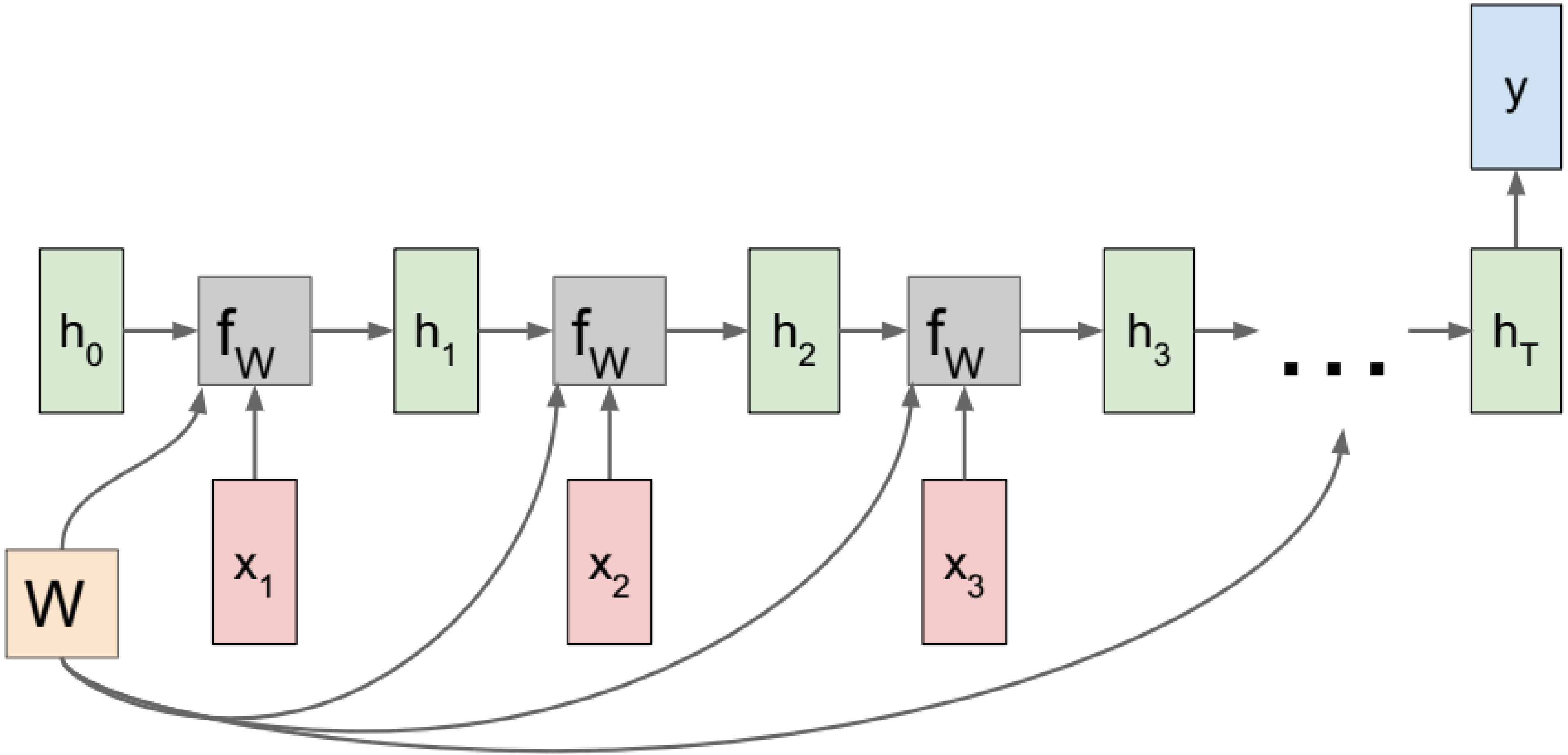
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

RNN: Computational Graph

- Re-use the same weight matrix at every time-step

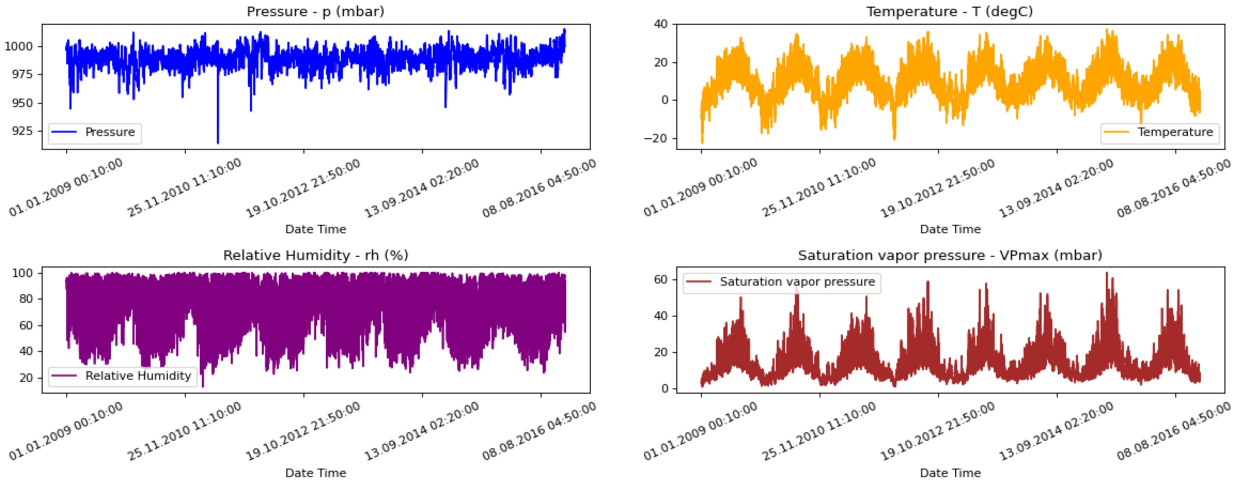


Many to One



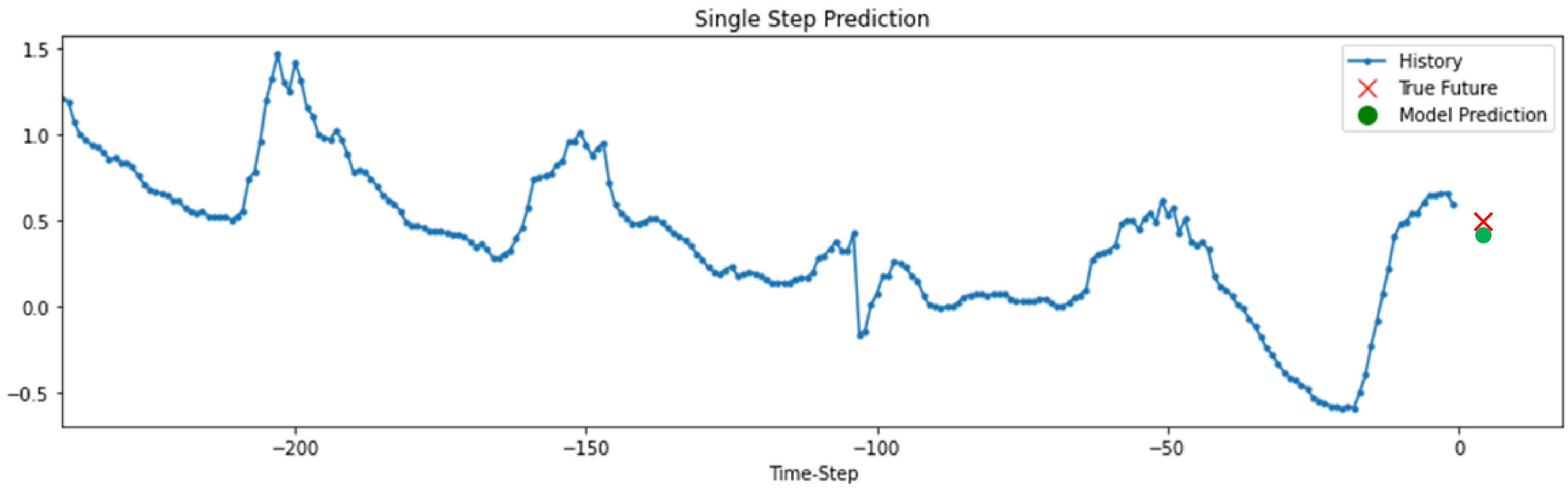
Example: Timeseries forecasting

- We will be using Jena Climate dataset recorded by the Max Planck Institute for Biogeochemistry
- The dataset consists of 14 features such as temperature, pressure, humidity etc, recorded once per 10 minutes
  - Jan 10, 2009 - December 31, 2016



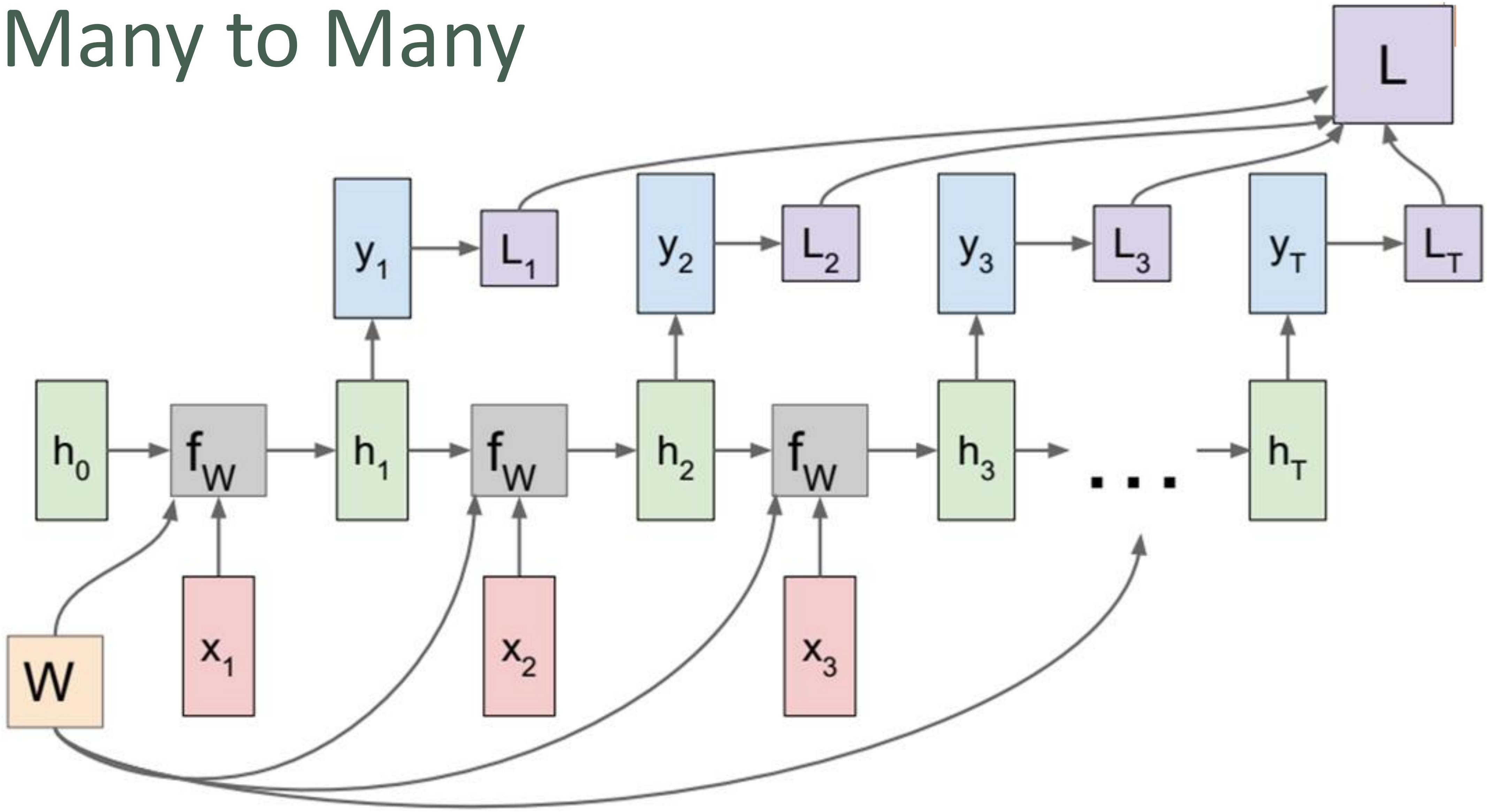
Example: Timeseries forecasting

- We are tracking data from past 720 timestamps (720/6=120 hours) by step=3
- This data will be used to predict the temperature after N timestamps (N/6 hours)



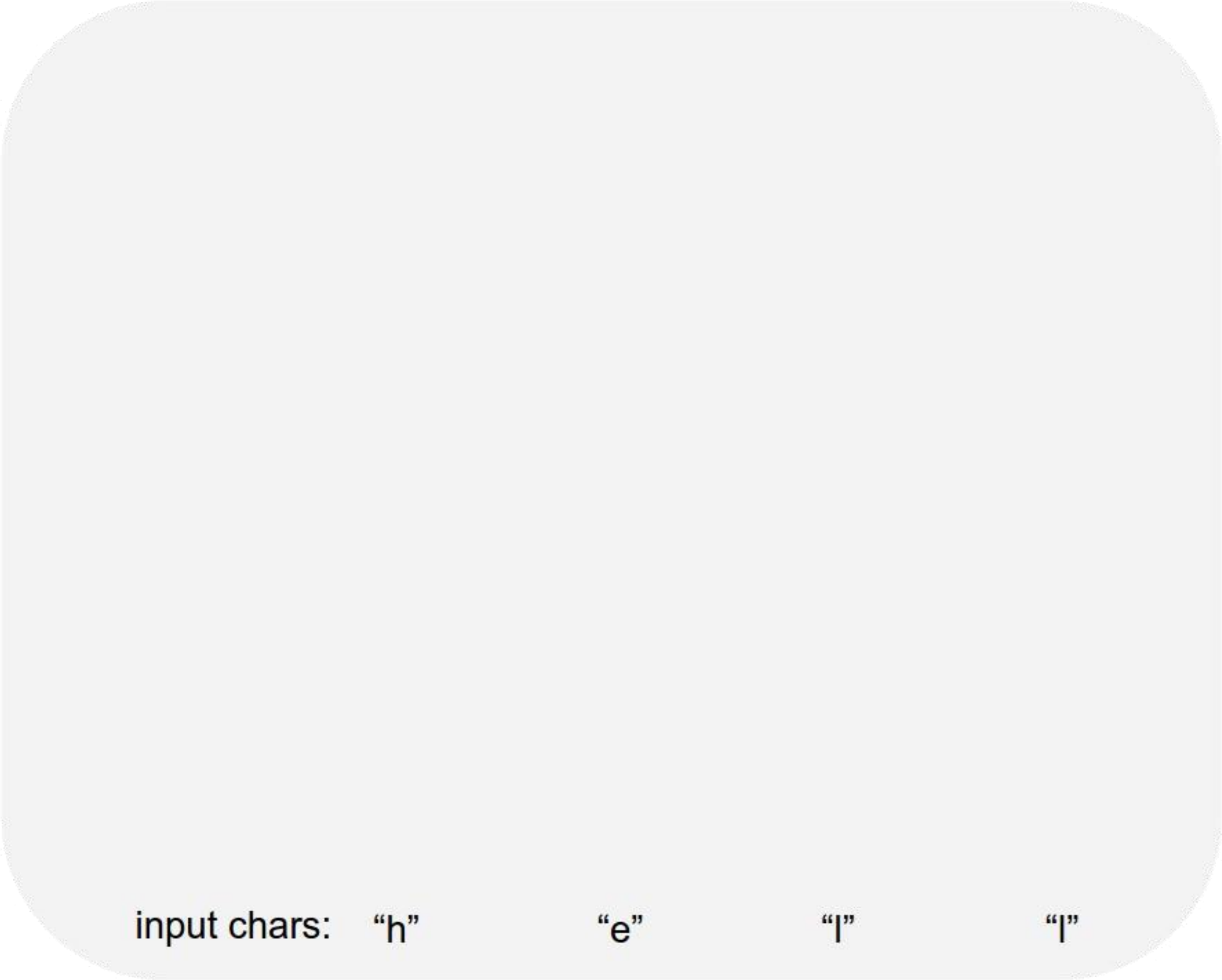
https://colab.research.google.com/drive/1EDbMn0bRYUy4xKaJxr3j-uZIlszHPiEN?usp=sharing

Many to Many



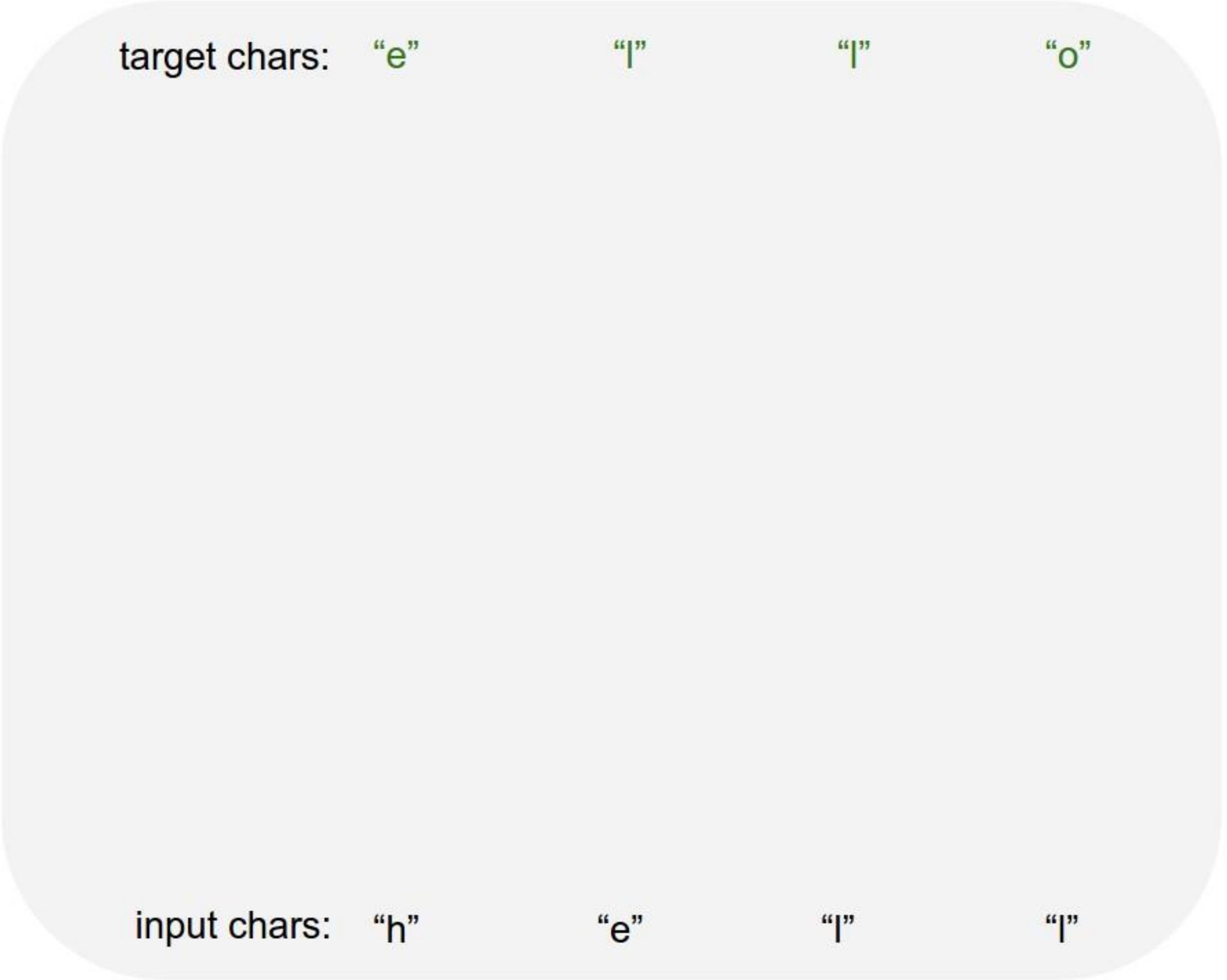
Example: Character-level Language Model

- Example training sequence:
  - “hello”



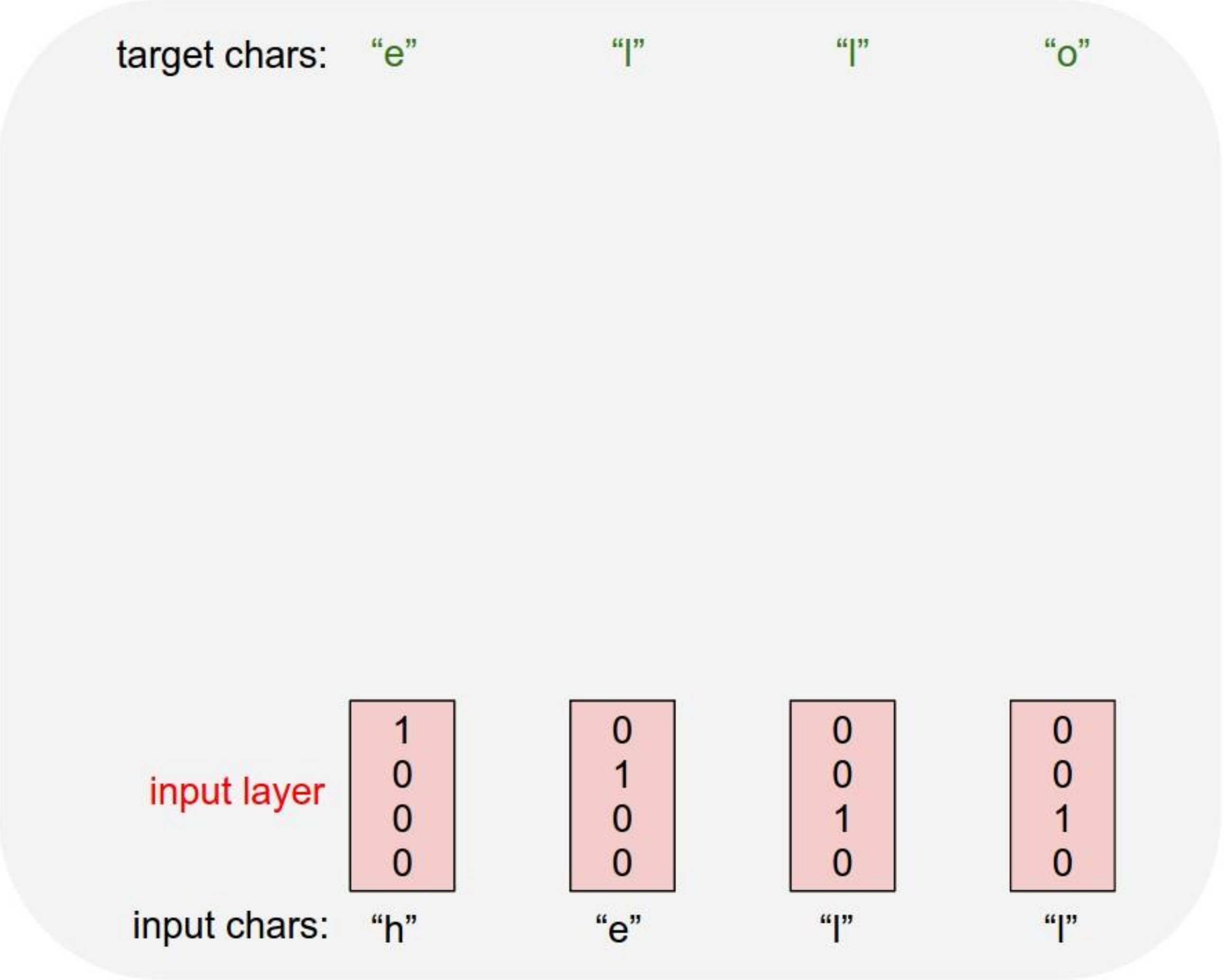
Example: Character-level Language Model

- Example training sequence:
  - “hello”



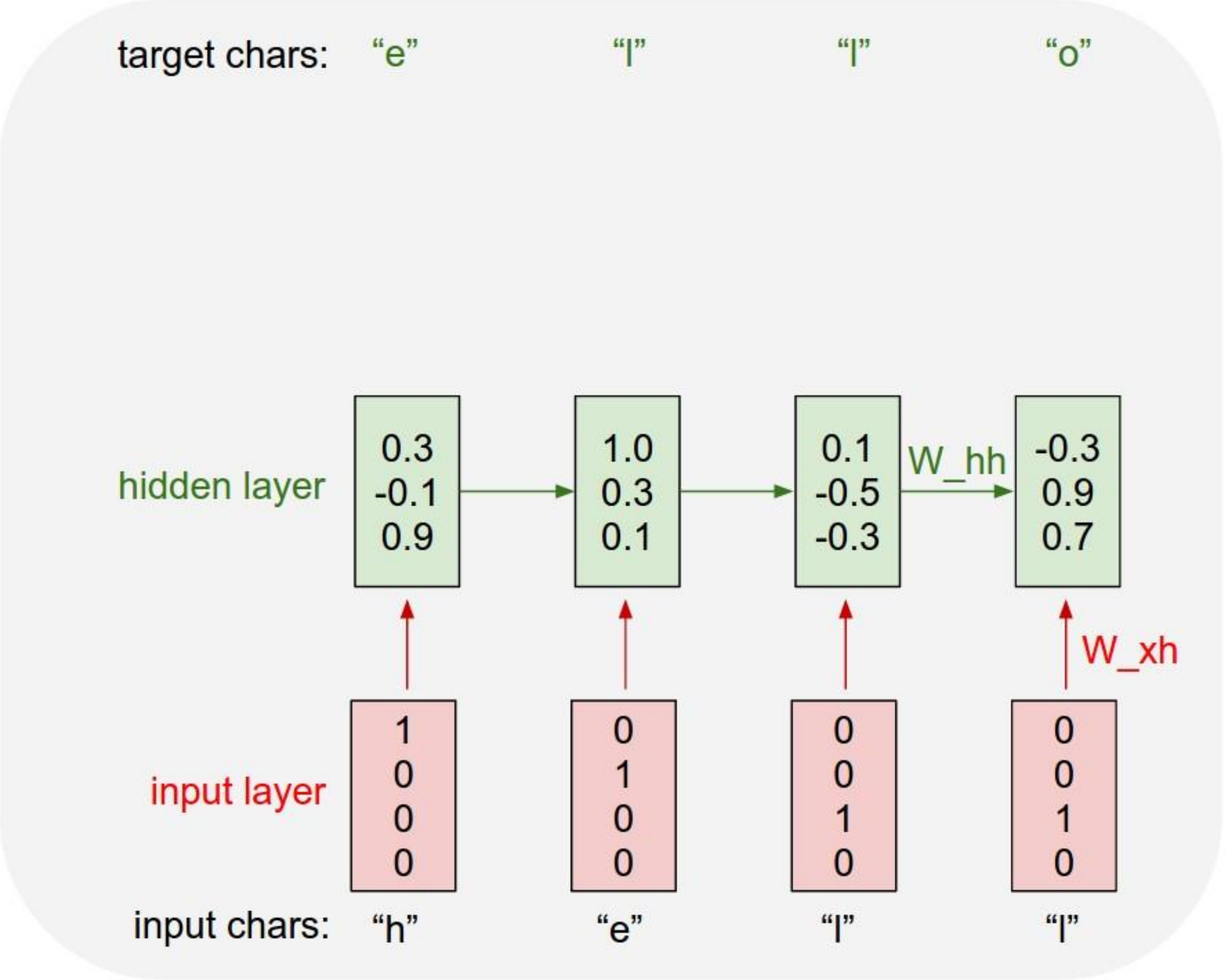
Example: Character-level Language Model

- Example training sequence:
  - "hello"
- Vocabulary:
  - [h,e,l,o]



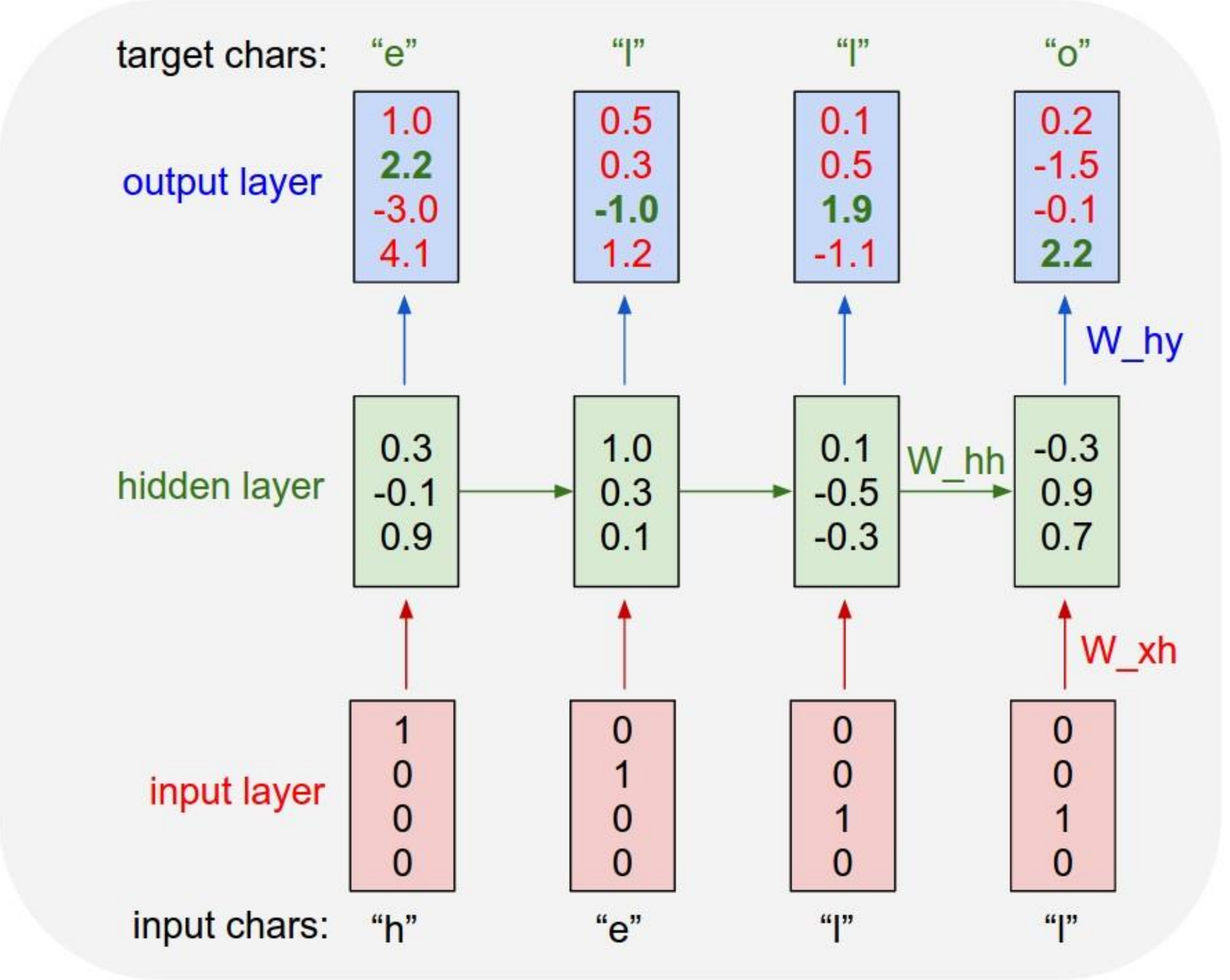
Example: Character-level Language Model

- Example training sequence:
  - “hello”
- Vocabulary:
  - [h,e,l,o]
- Hidden recurrent layer:
  - $h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$



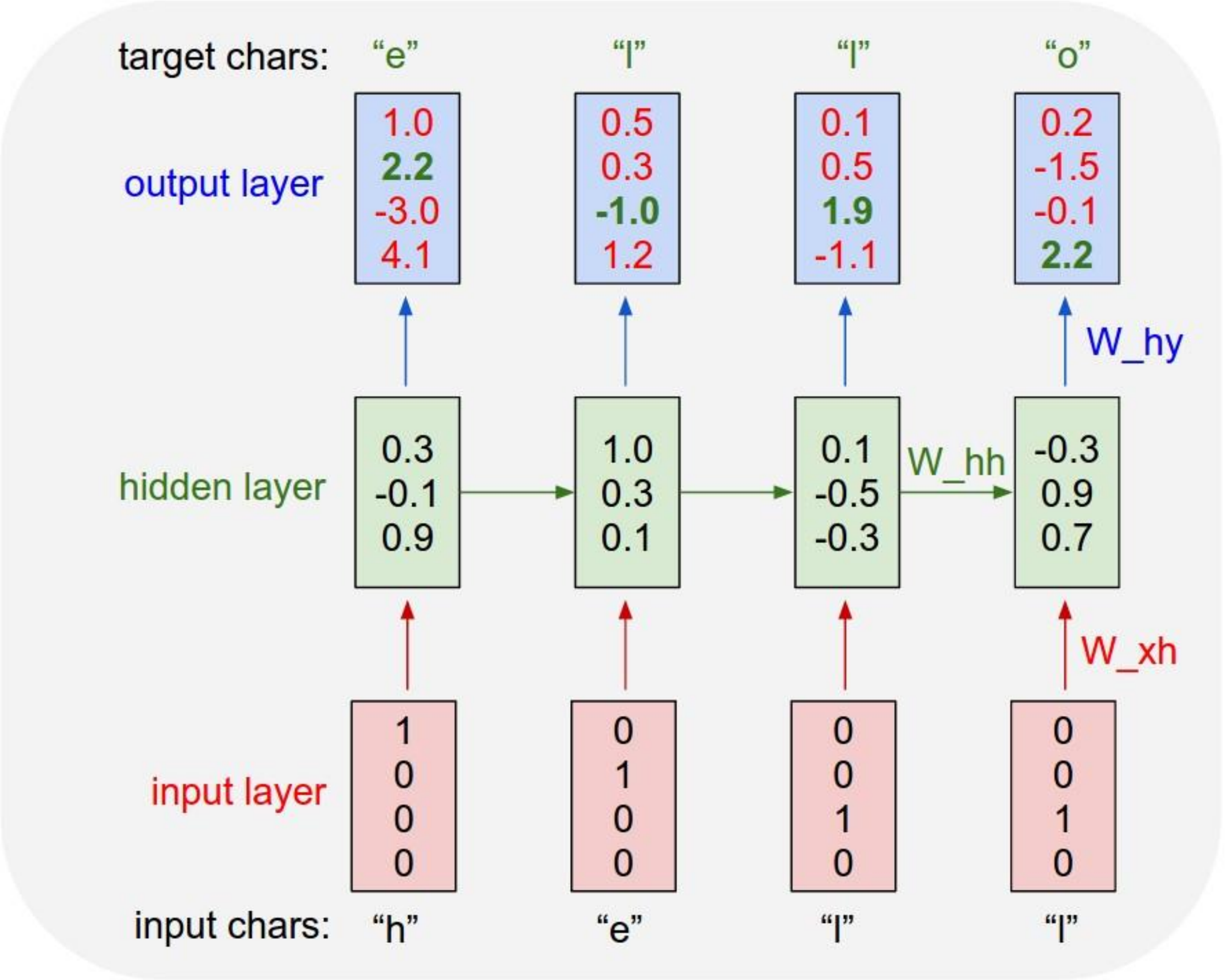
Example: Character-level Language Model

- Example training sequence:
  - “hello”
- Vocabulary:
  - [h,e,l,o]
- Hidden recurrent layer:
  - $h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$
- Output dense layer:
  - $y_t = W_{hy} h_t$
  - We can use SoftMax



Example: Character-level Language Model

- At test time:
  - Sample characters one at a time, feed back to model



Minimal character-level language model with a Vanilla Recurrent Neural Network, in Python/numpy

min-char-rnn.py

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y. see http://cs231n.github.io/neural-networks-case-study/#grad if confused here
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

Shakespeare

- 100,000 character sample

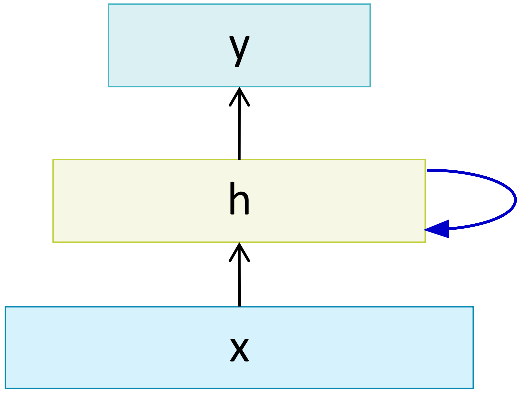
input.txt

```
That, poor contempt, or claim'd thou slept so faithful,
I may contrive our father; and, in their defeated queen,
Her flesh broke me and puttance of expedition house,
And in that same that ever I lament this stomach,
And he, nor Butly and my fury, knowing everything
Grew daily ever, his great strength and thought
The bright buds of mine own.

BIONDELLO:
Marry, that it may not pray their patience.'

KING LEAR:
The instant common maid, as we may less be
a brave gentleman and joiner: he that finds us with wax
And owe so full of presence and our fooder at our
staves. It is remorsed the bridal's man his grace
for every business in my tongue, but I was thinking
that he contends, he hath respected thee.

BIRON:
She left thee on, I'll die to blessed and most reasonable
Nature in this honour, and her bosom is safe, some
others from his speedy-birth, a bill and as
Forestem with Richard in your heart
Be question'd on, nor that I was enough:
Which of a partier forth the obsers d'punish'd the hate
```



The evolution of samples while training

- At iteration 100:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e plia tklrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

- At iteration 300:

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

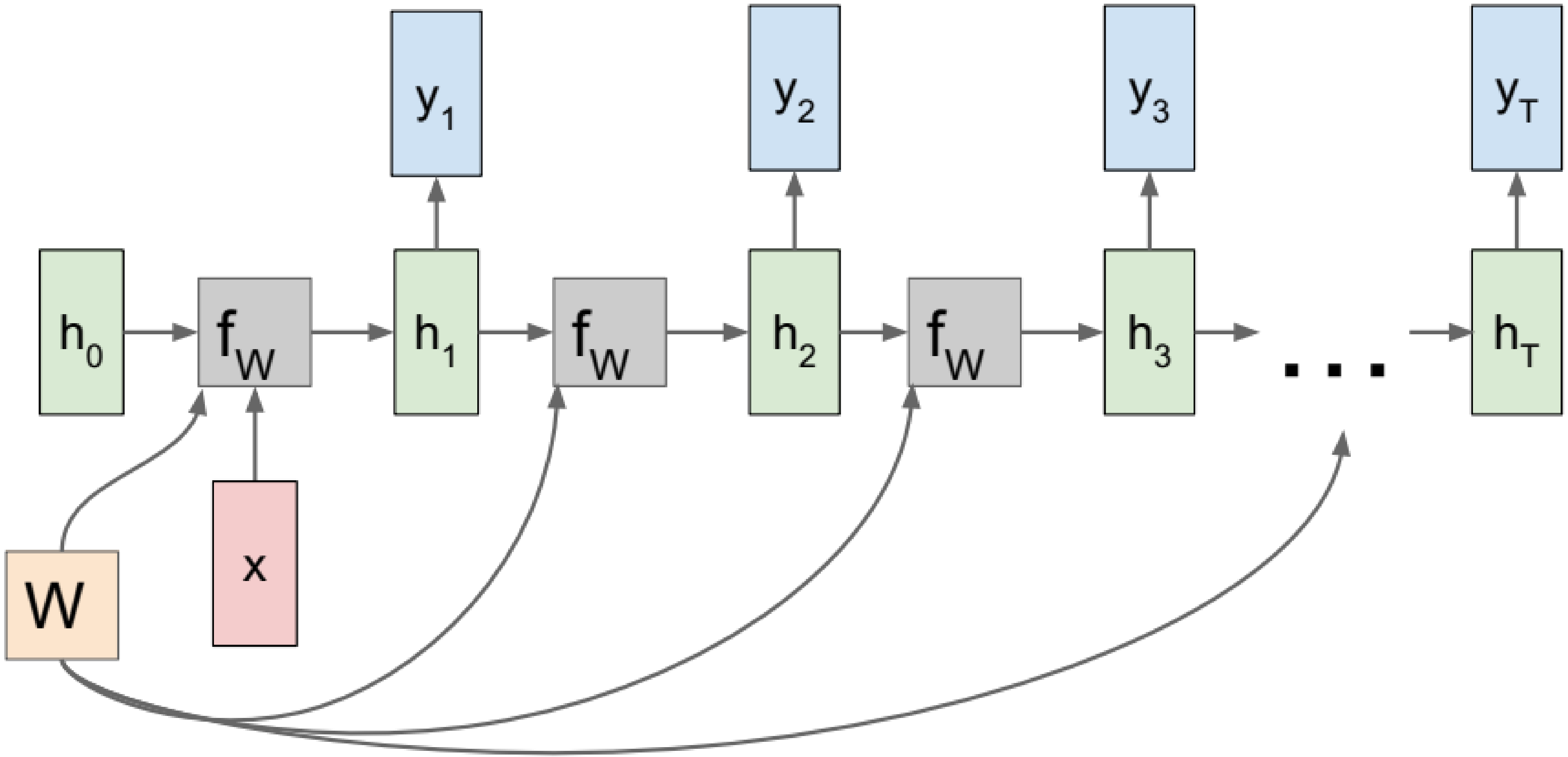
- At iteration 700:

Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.

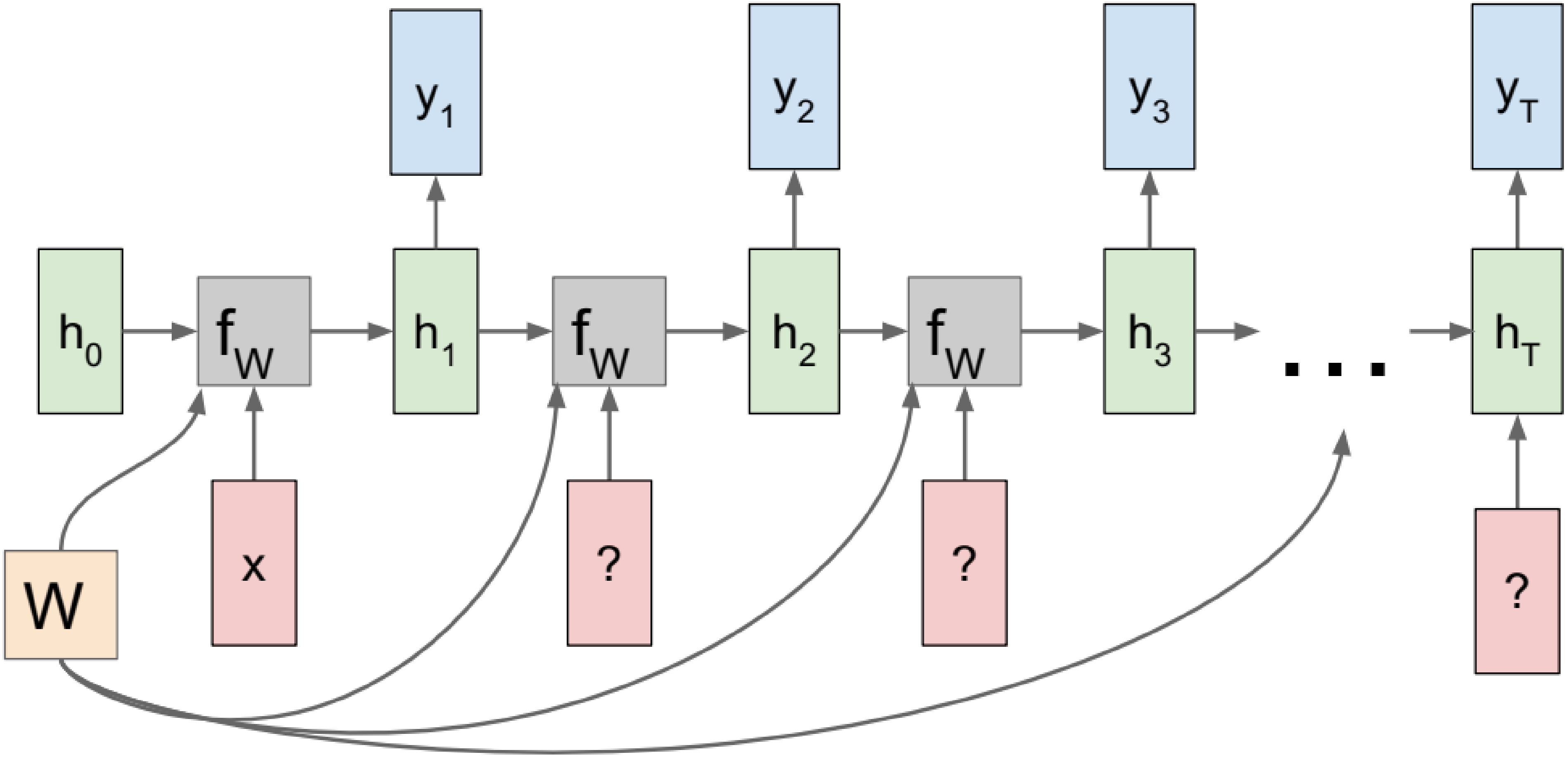
- At iteration 2000:

"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.

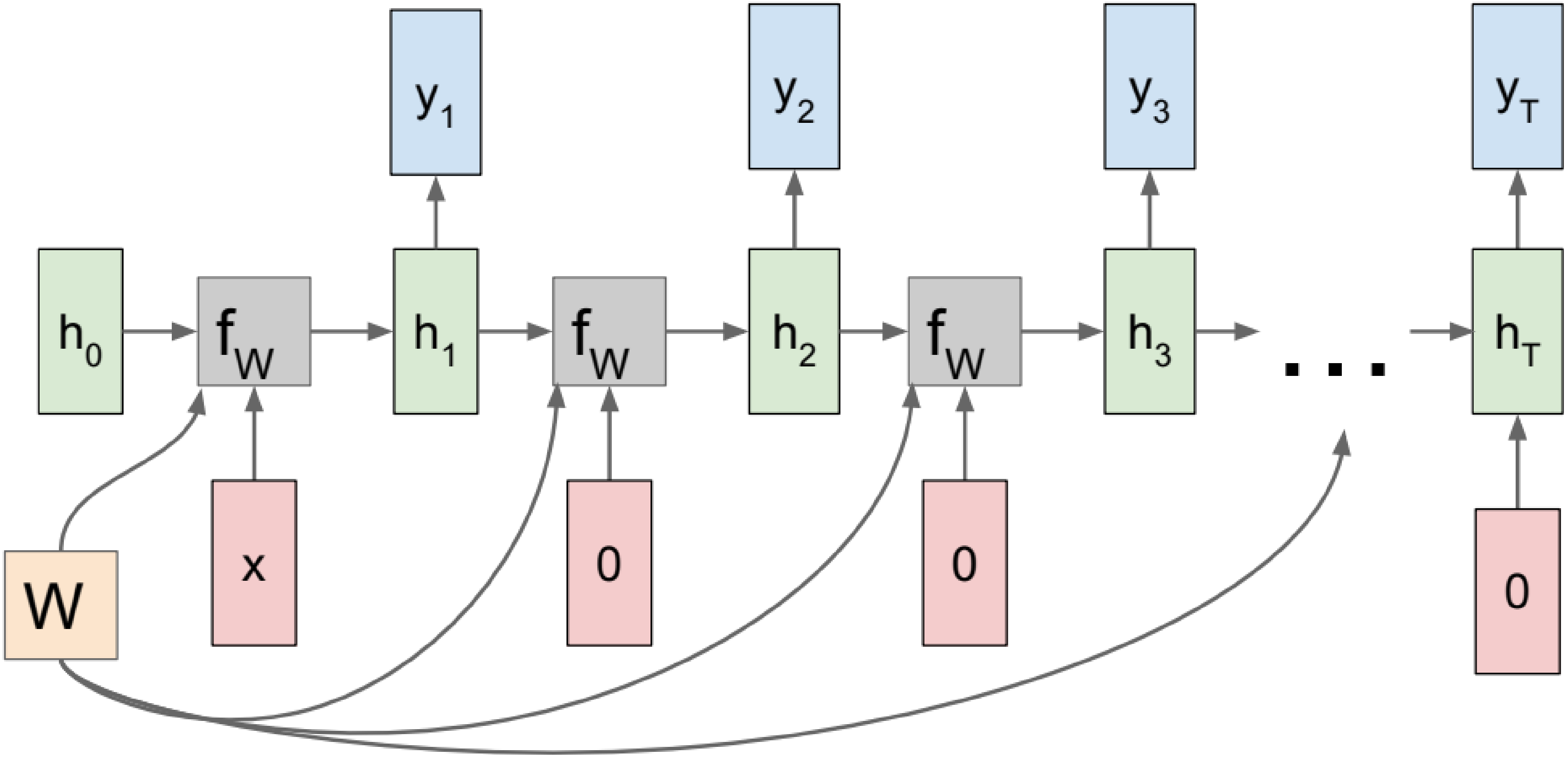
One to Many



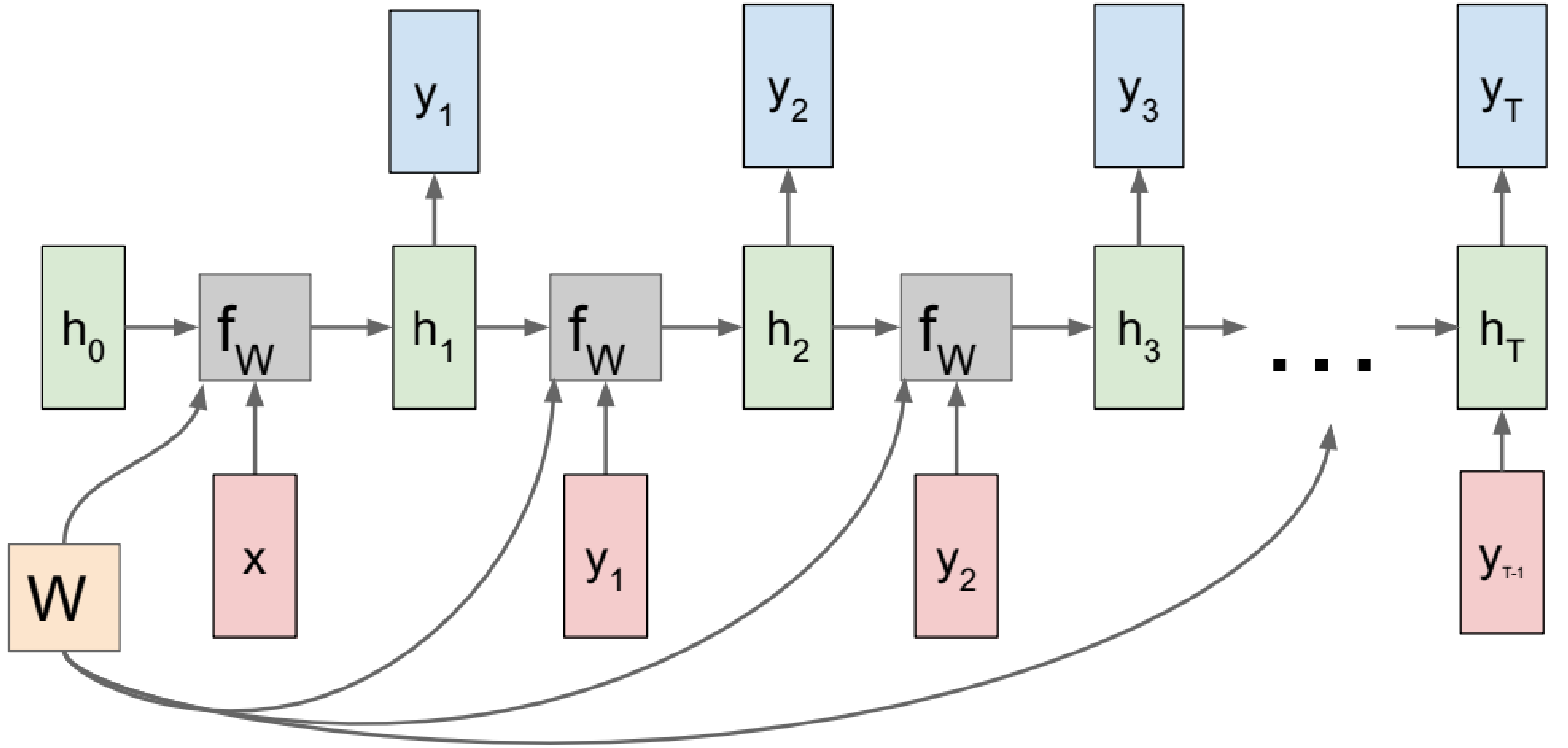
One to Many



One to Many

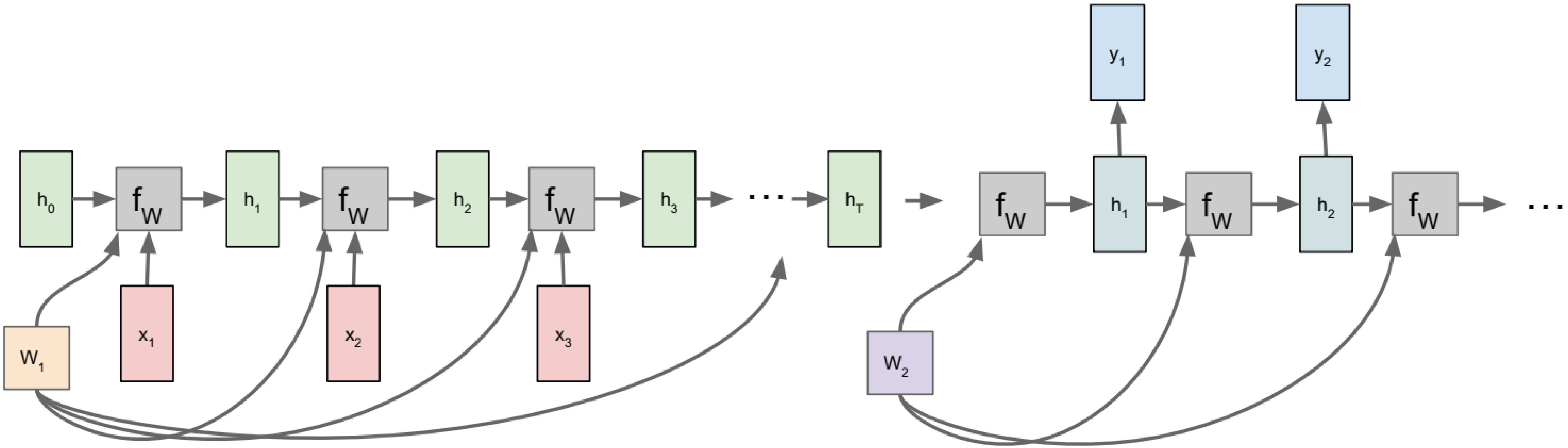


One to Many



Sequence to Sequence

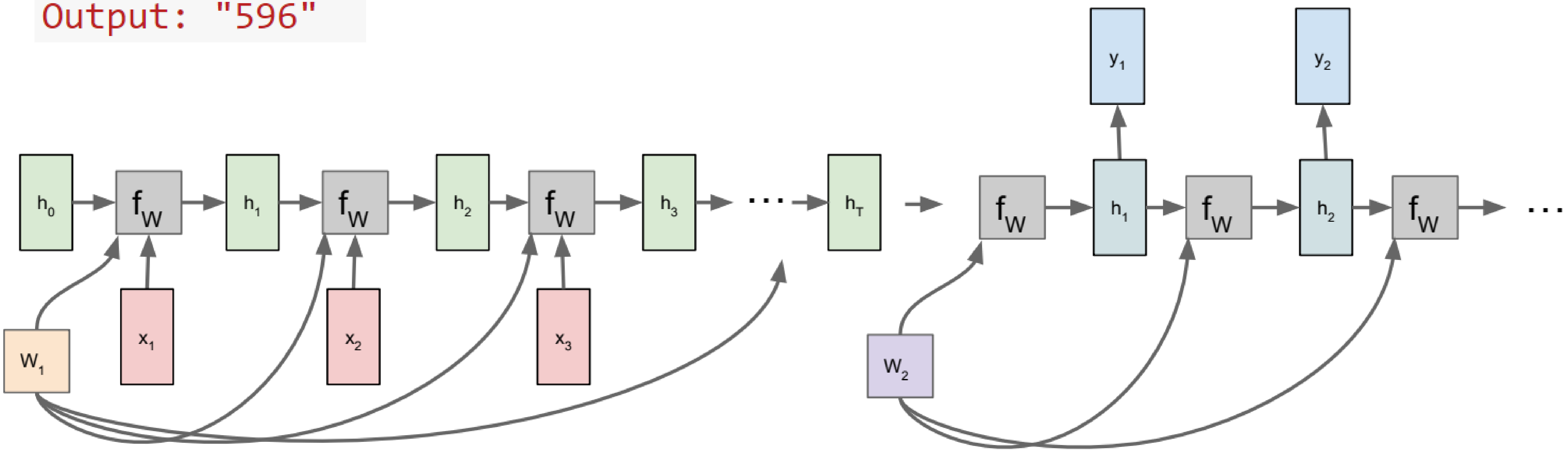
- Many to One + One to Many
  - Many to one: Encode input sequence in a single vector
  - One to many: Produce output sequence from single input vector



Sequence to Sequence

- Example: train a model to learn to add two numbers, provided as strings

```
Input: "535+61"
Output: "596"
```



https://colab.research.google.com/drive/11qFxWP5ukCOKXXcP2jTBYEKRl3ThBYH6?usp=sharing