بسم الله الرحمن الرحیم

یادگیری عمیق

مدرس: محمدرضا محمدی

بهار ۱۴۰۲

شبکه‌های عصبی بازگشتی

Recurrent Neural Networks

مدل‌های مارکوف

- فرض می‌کنیم نمونه فعلی به تعداد کمی از نمونه‌های قبلی وابسته باشد
  - مرتبه ۰ (مستقل):

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2)P(x_3)P(x_4)$$

  - مرتبه ۱:

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_3)$$

  - مرتبه ۲:

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)P(x_4|x_2, x_3)$$

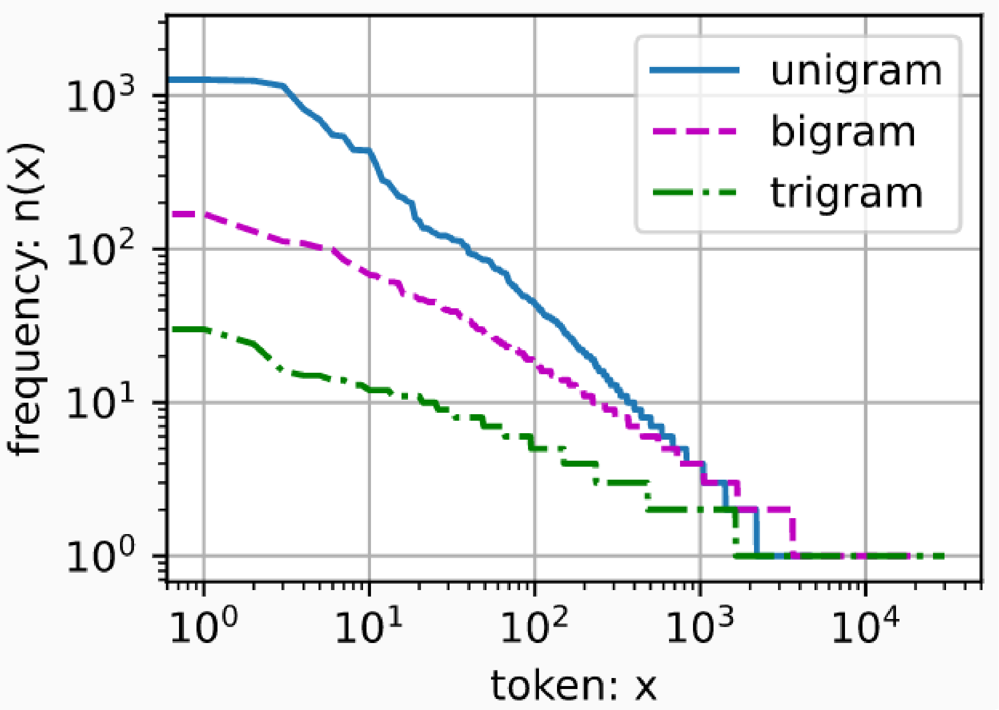
  - به این مدل‌ها به ترتیب unigram، bigram و trigram گفته می‌شود

مثال: مجموعه داده The Time Machine

```
[('the', 2261),
 ('i', 1267),
 ('and', 1245),
 ('of', 1155),
 ('a', 816),
 ('to', 695),
 ('was', 552),
 ('in', 541),
 ('that', 443),
 ('my', 440)]
```

```
[(('of', 'the'), 309),
 (('in', 'the'), 169),
 (('i', 'had'), 130),
 (('i', 'was'), 112),
 (('and', 'the'), 109),
 (('the', 'time'), 102),
 (('it', 'was'), 99),
 (('to', 'the'), 85),
 (('as', 'i'), 78),
 (('of', 'a'), 73)]
```

```
[(('the', 'time', 'traveller'), 59),
 (('the', 'time', 'machine'), 30),
 (('the', 'medical', 'man'), 24),
 (('it', 'seemed', 'to'), 16),
 (('it', 'was', 'a'), 15),
 (('here', 'and', 'there'), 15),
 (('seemed', 'to', 'me'), 14),
 (('i', 'did', 'not'), 14),
 (('i', 'saw', 'the'), 13),
 (('i', 'began', 'to'), 13)]
```



یادگیری یک مدل زبان

- برای تخمین احتمال شرطی می‌توان از حالت پنهان استفاده کرد

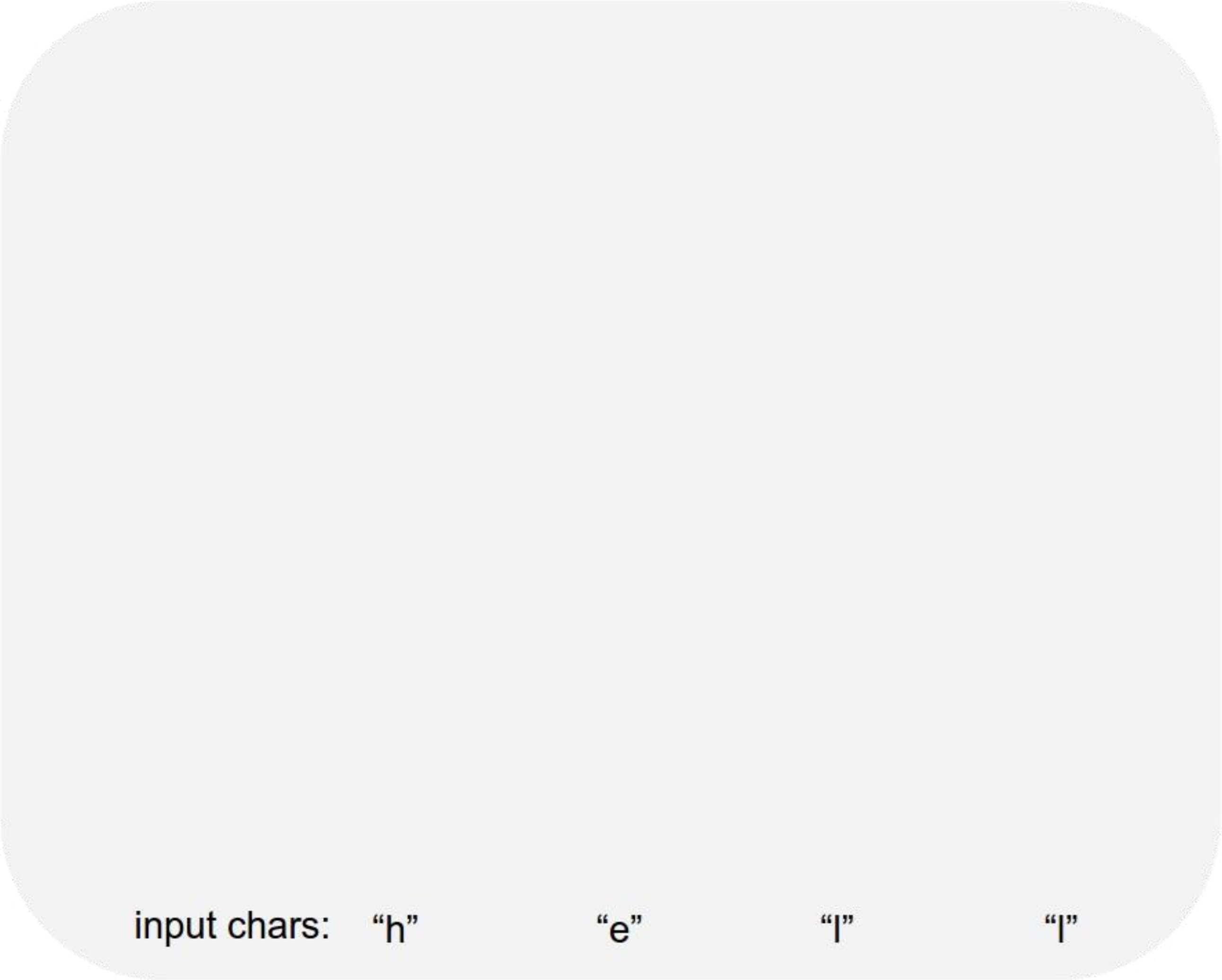
$$P(x_t \mid x_{t-1}, \dots, x_1) \approx P(x_t \mid h_{t-1})$$

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

- با استفاده از یادگیری عمیق می‌توانیم مدلی آموزش دهیم که بتواند احتمال توکن بعدی را پیش‌بینی کند

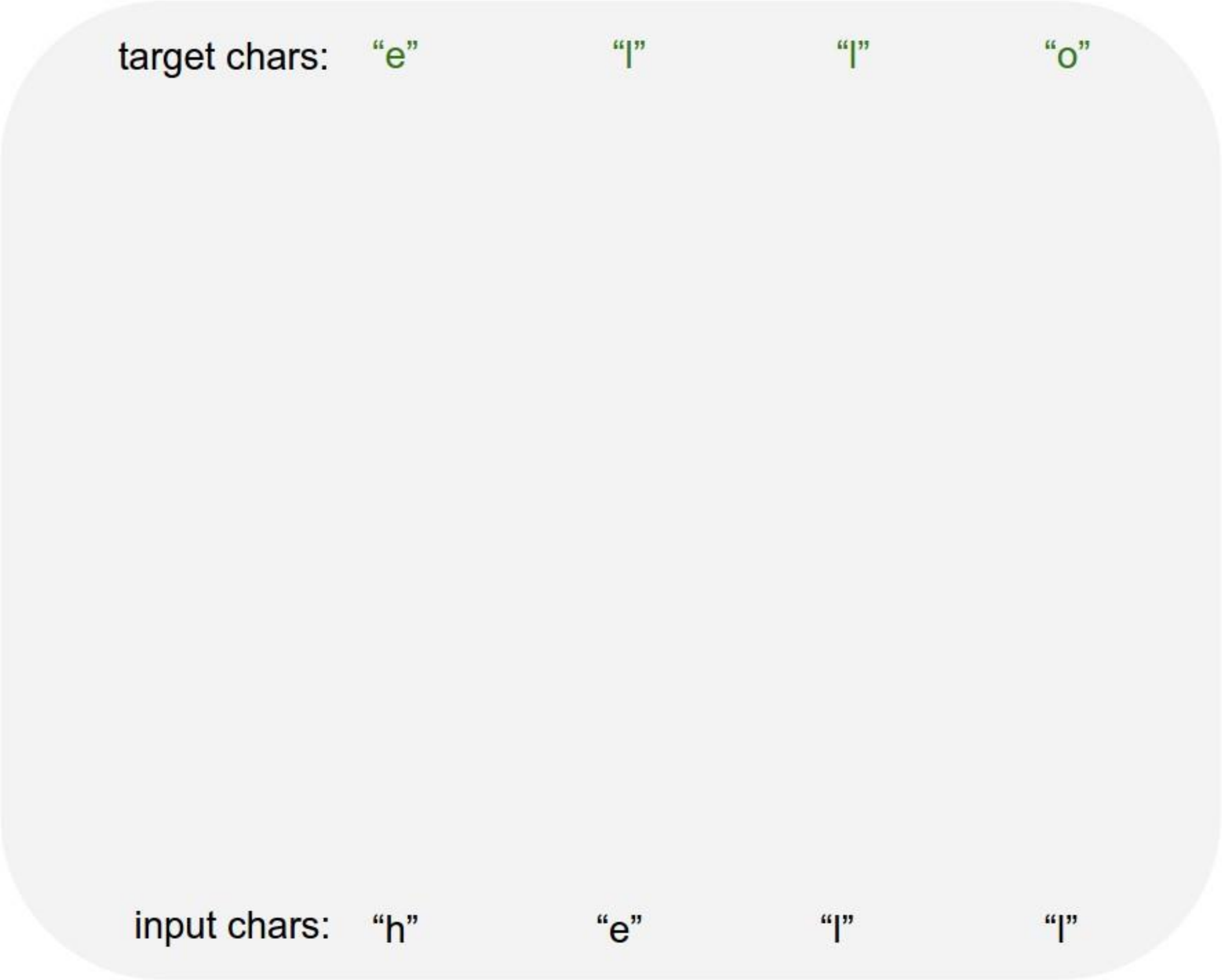
مثال: مدل زبانی سطح کاراکتر

- دنباله آموزشی نمونه:
  - “hello”



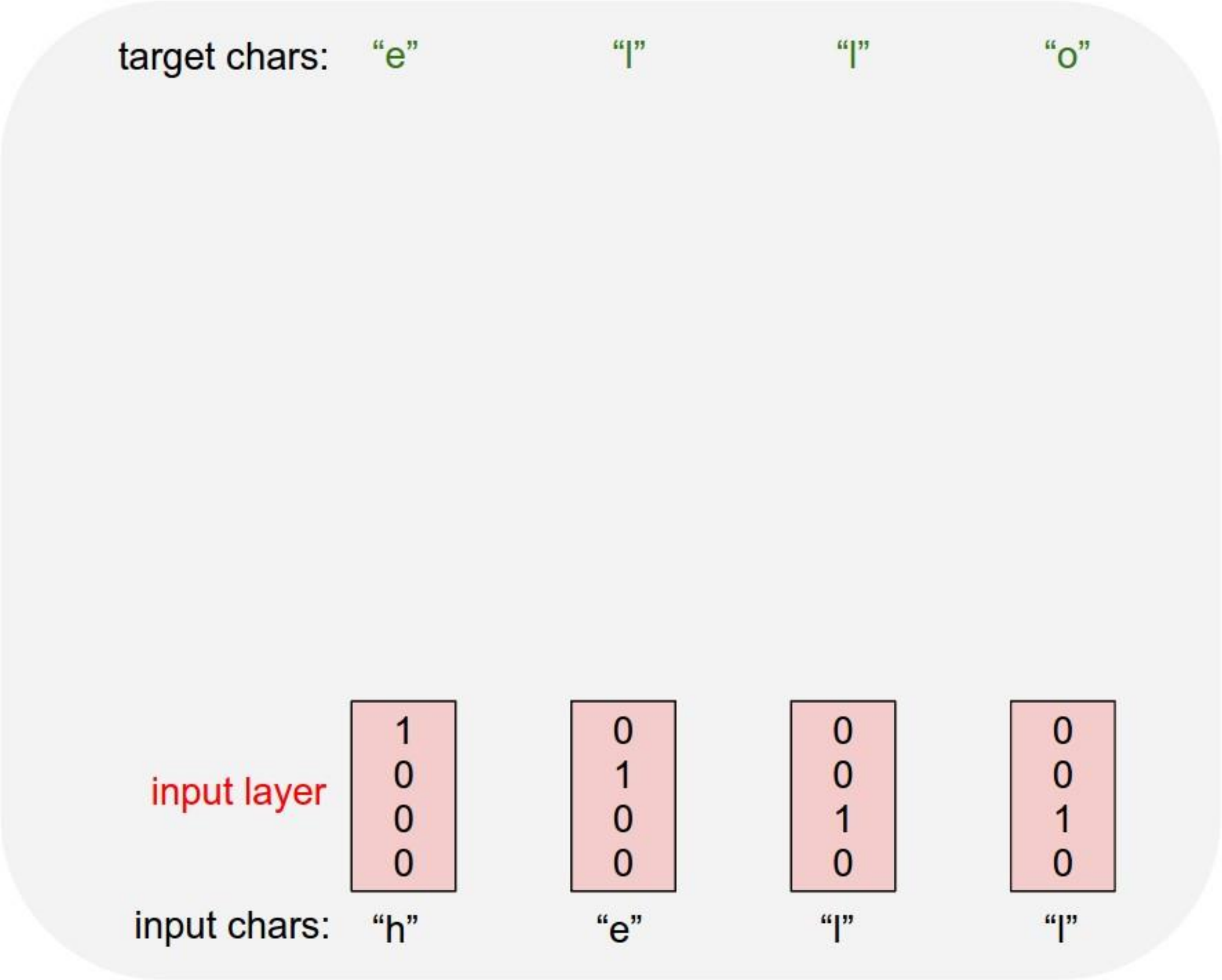
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

مثال: مدل زبانی سطح کاراکتر



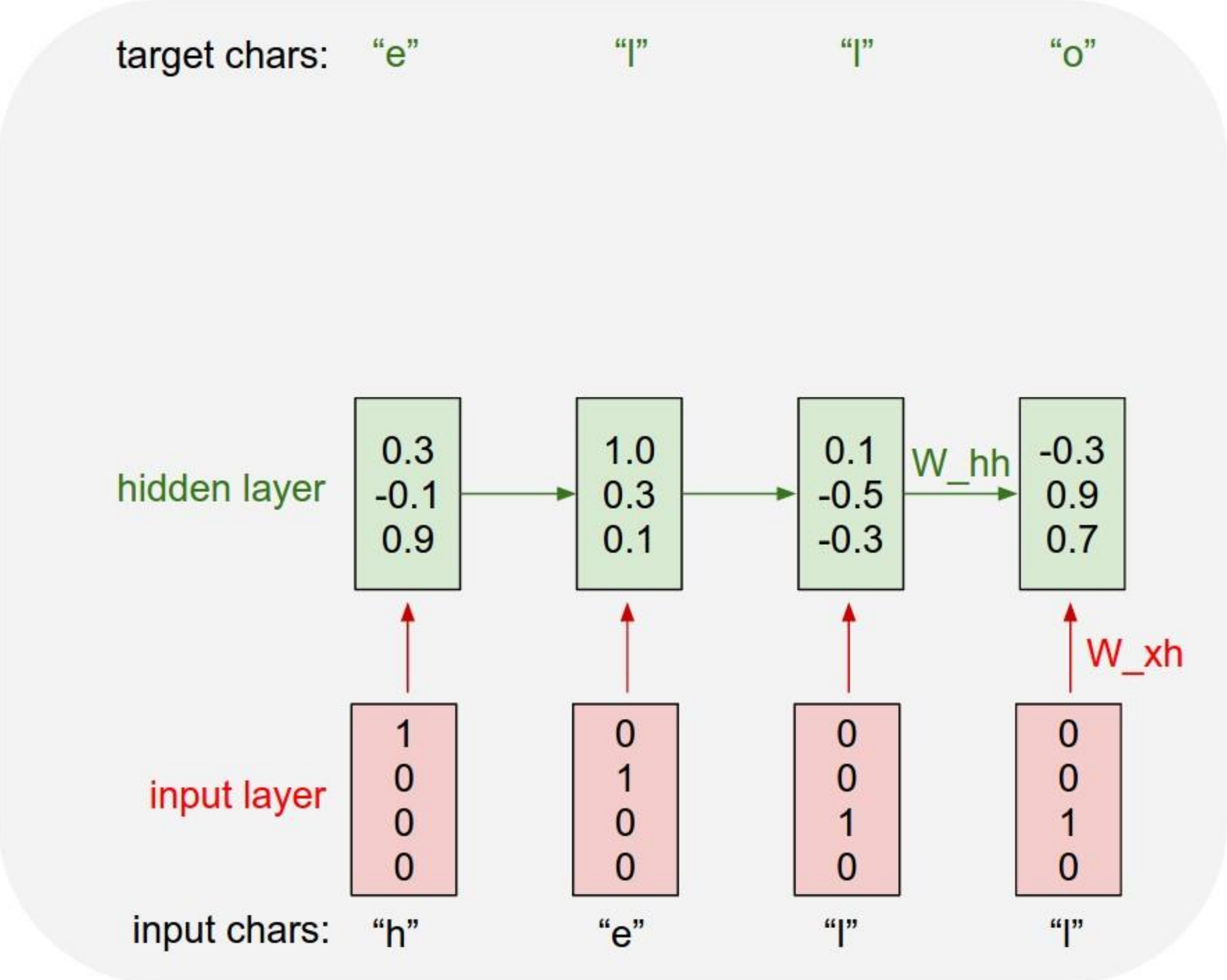
- دنباله آموزشی نمونه:
  - “hello”

مثال: مدل زبانی سطح کاراکتر



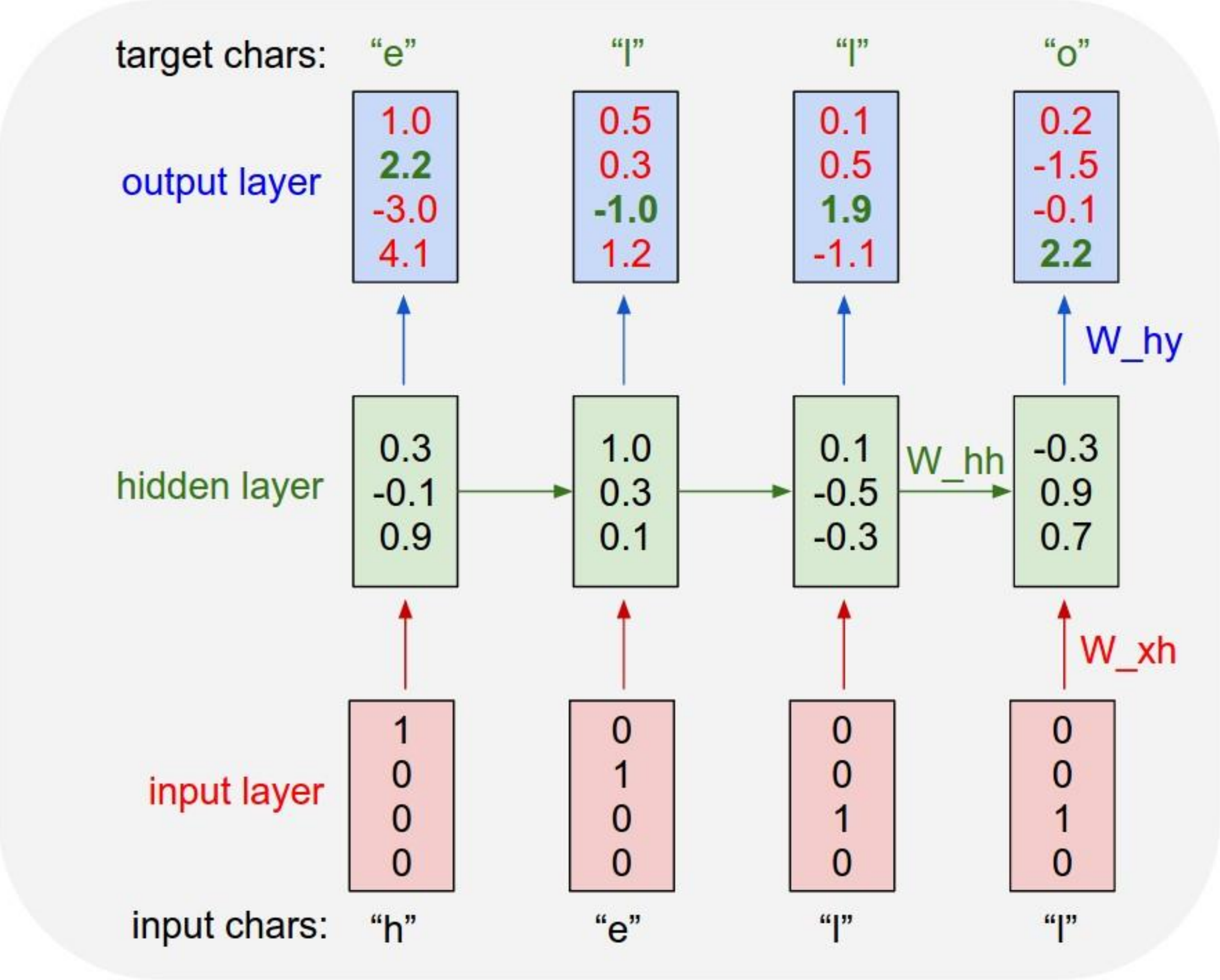
- دنباله آموزشی نمونه:
  - “hello”
- توکن‌ها:
  - [h,e,l,o]

مثال: مدل زبانی سطح کاراکتر



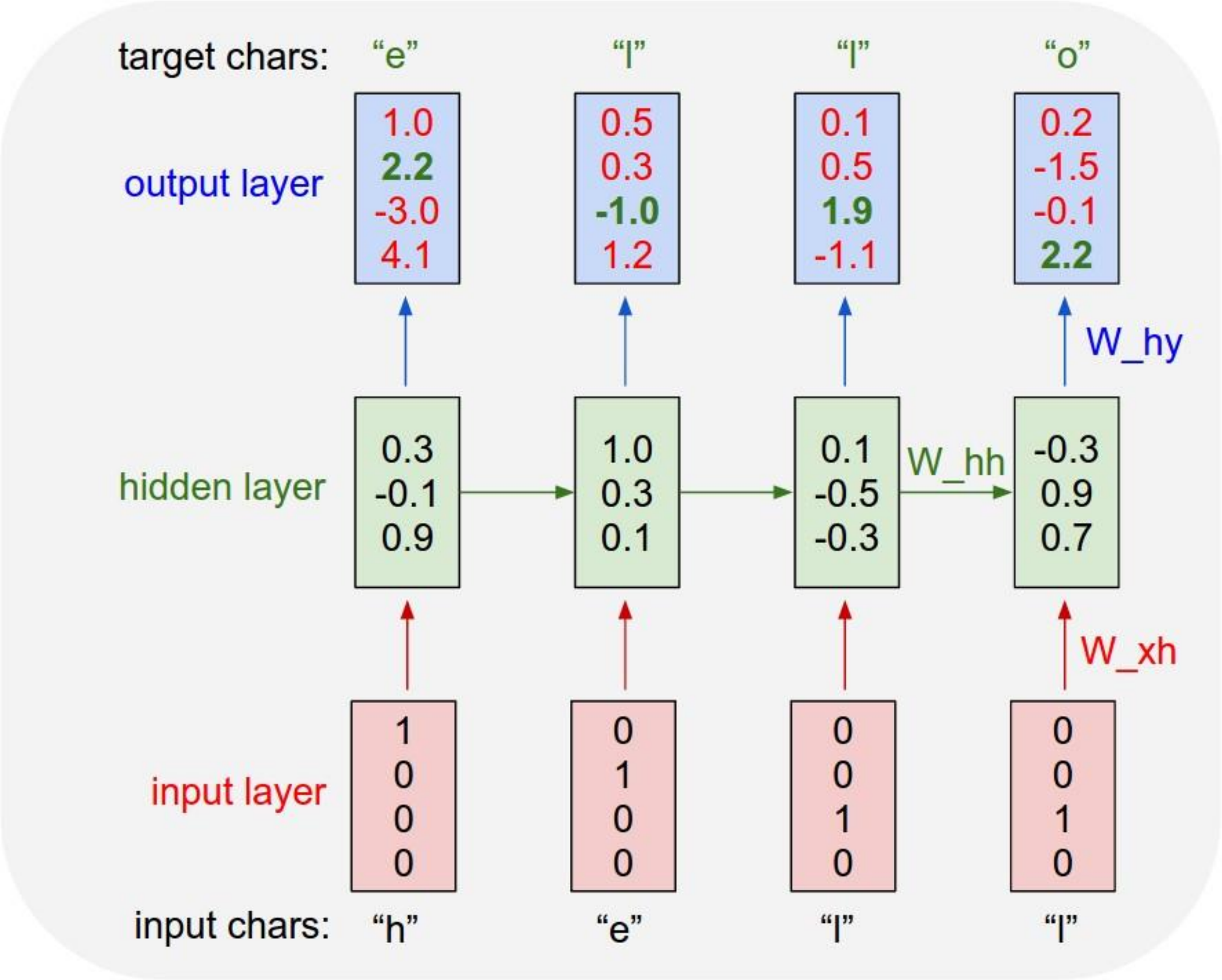
- دنباله آموزشی نمونه:
  - "hello"
- توکن‌ها:
  - [h,e,l,o]
- لایه بازگشتی میانی:
  - $\mathbf{h}_t = \phi(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$

مثال: مدل زبانی سطح کاراکتر



- دنباله آموزشی نمونه:
  - "hello"
- توکن‌ها:
  - [h,e,l,o]
- لایه بازگشتی میانی:
  - $\mathbf{h}_t = \phi(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$
- لایه کاملا متصل خروجی:
  - $\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t$
  - می‌توان از SoftMax هم استفاده کرد

مثال: مدل زبانی سطح کاراکتر



- در زمان تست:
  - در هر گام یک کاراکتر نمونه‌برداری می‌شود و ورودی گام بعد می‌شود

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

- شامل حدود ۱۰۰،۰۰۰ کلمه

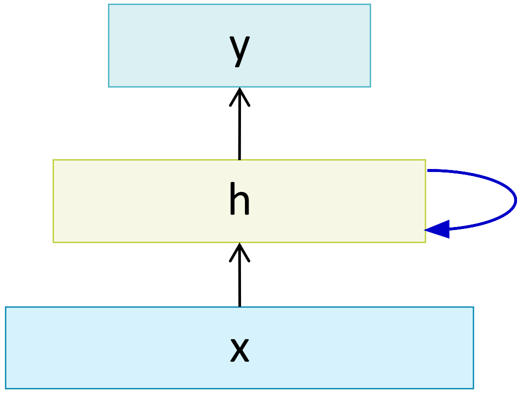
input.txt

```
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
```



تکامل نمونه‌ها در حین آموزش

- تکرار ۱۰۰

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

- تکرار ۳۰۰

"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."

- تکرار ۷۰۰

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.

- تکرار ۲۰۰۰

"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.

جست پا

که خبر می آرند از گل

و شده دور ترین بوته خاک

دم اینجا تنهاست

و در این تنهایی

سیه تا ابدیت آجری است

در سراغ من اگر می آیید

نرم و اهسته بیایید مبادا که ترک بردارد

چینی نازک تنهایی من

و خشم تا نشسته به دشمنان ما

و شما که خشم تا مرگ است

یکی بود یکی نبود

در گلستانه ما

این در نیست

به سدی هم عوض دوازده گلوله

سوراخ شود

اگر که پریشان میخواهد

بدوم تاته دشت بروم تا سر کوه

دورها سراغ من می آیید

یک نفر نور نیست

تا خاکی سرد بگوی

پشت تبریزی ها

قفلت پاکی اهسته بیایید

که زندگی خواهد کرد

ابری نیست

ابدی نیست

هنوز

دیریست ابری نگذشته است ازین کنار

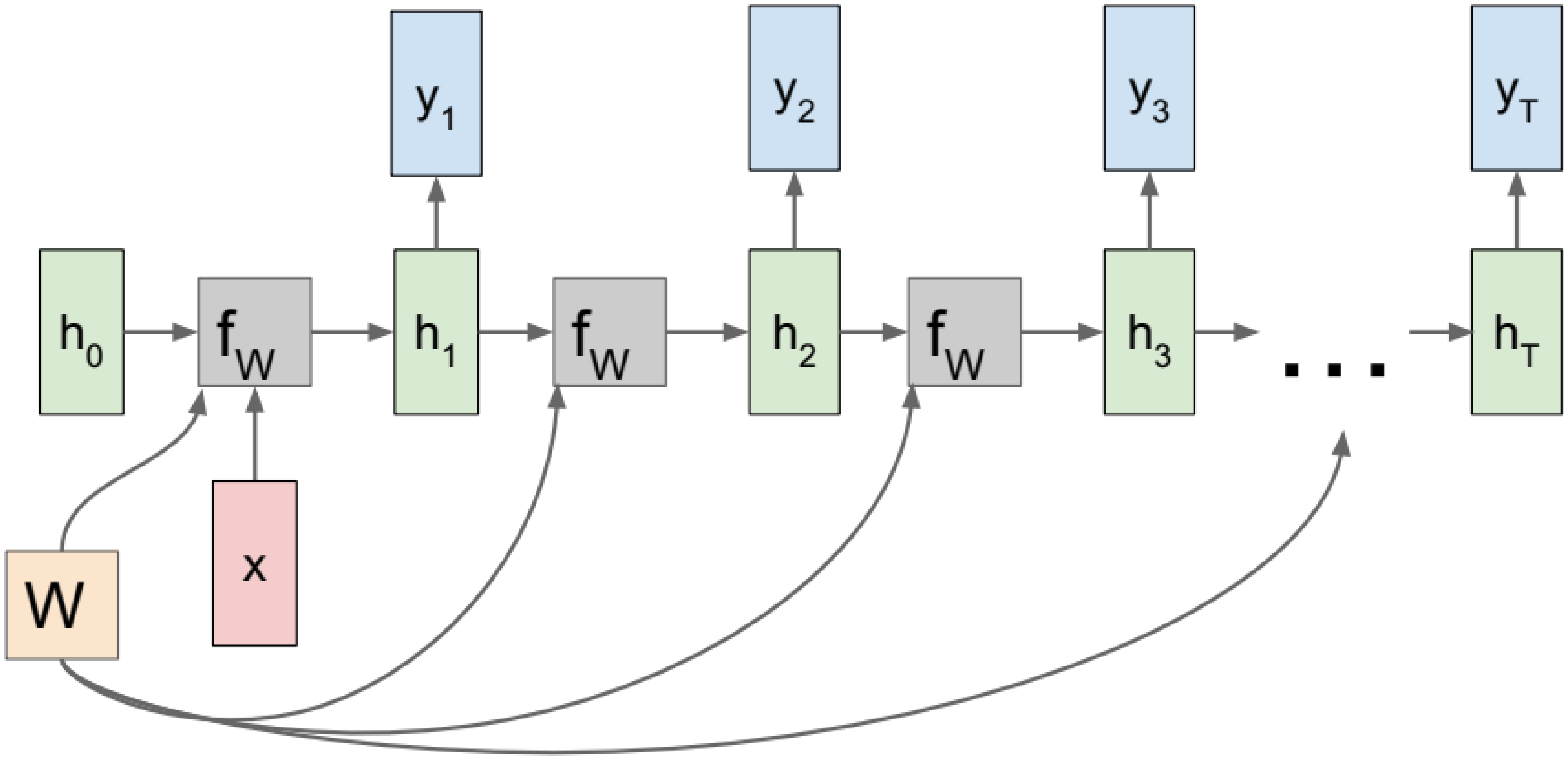
که از شما بتابد نوری

سروده شده توسط هوش مصنوعی

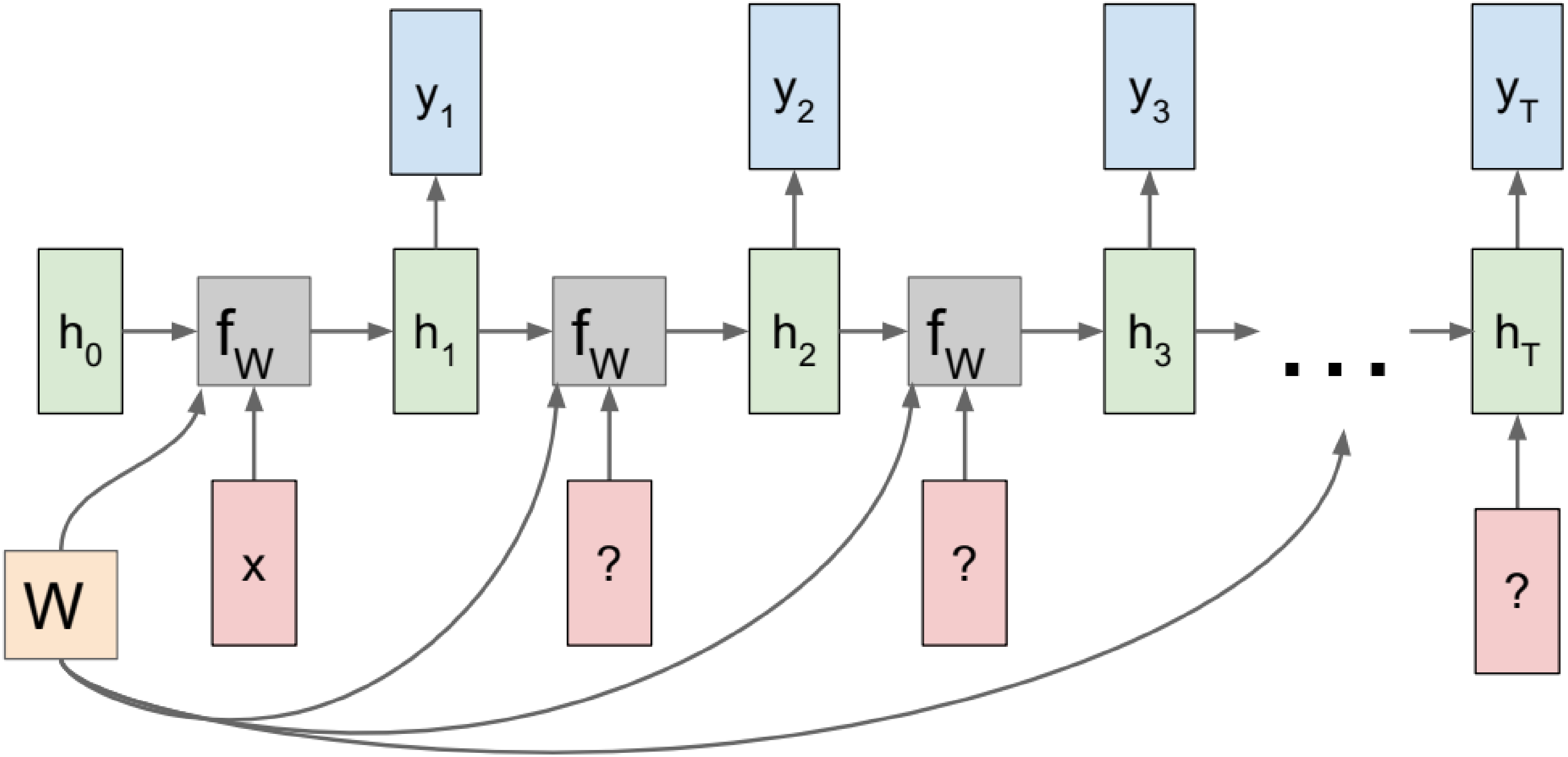
کاری از DeepO Band

http://www.tajournals.com/archive/engineering/1396/winter/6.pdf

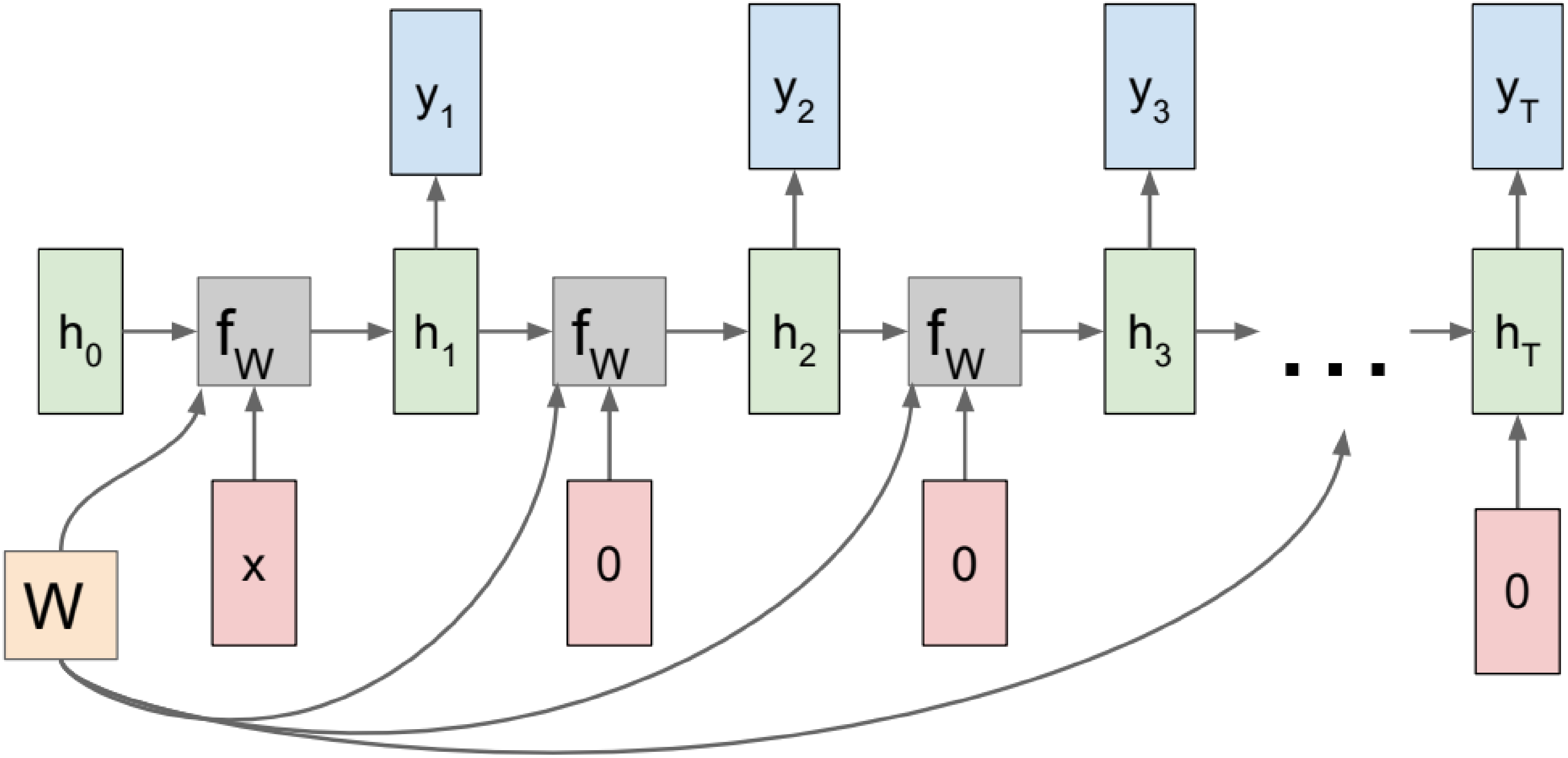
One to Many



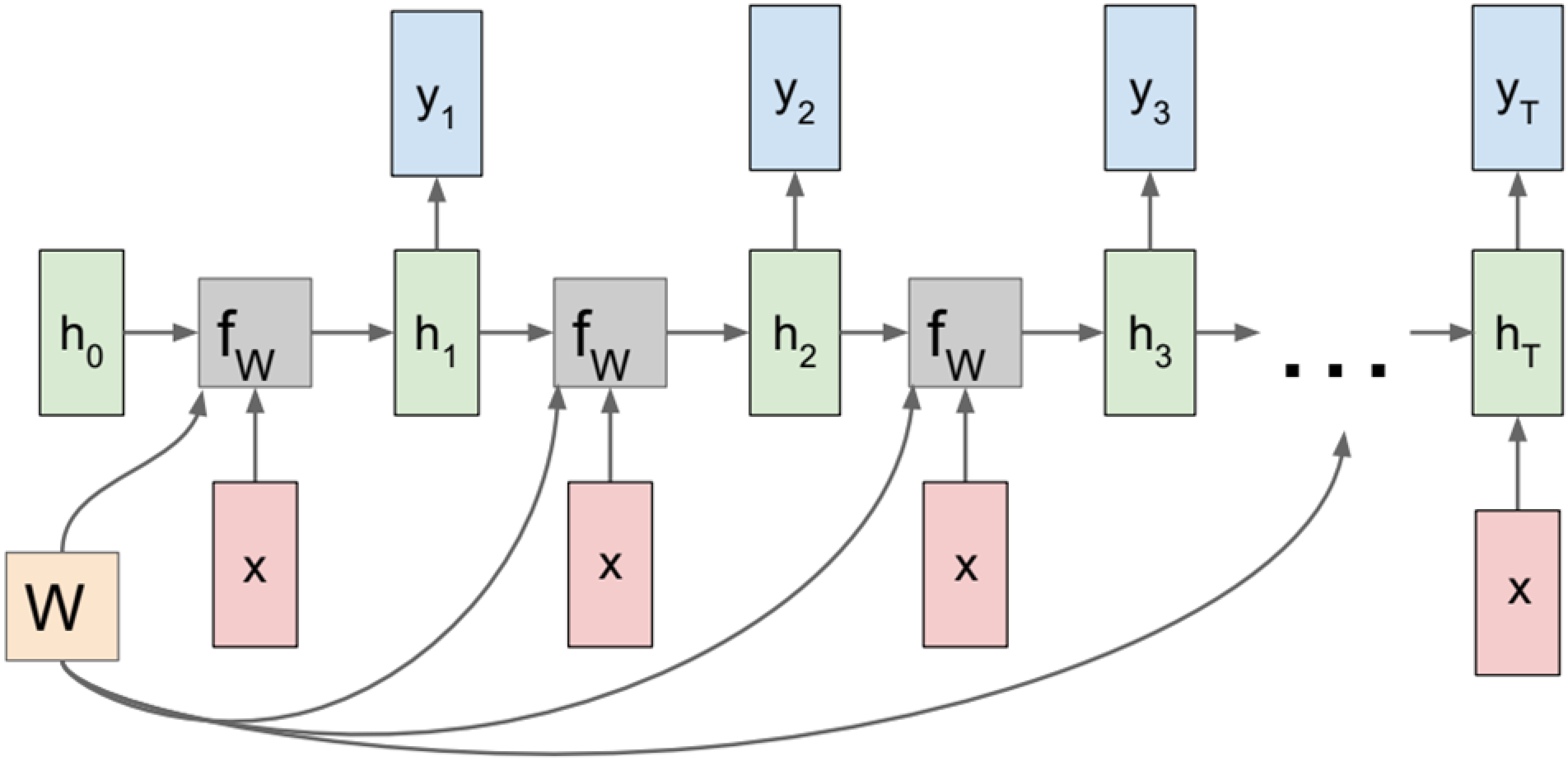
One to Many



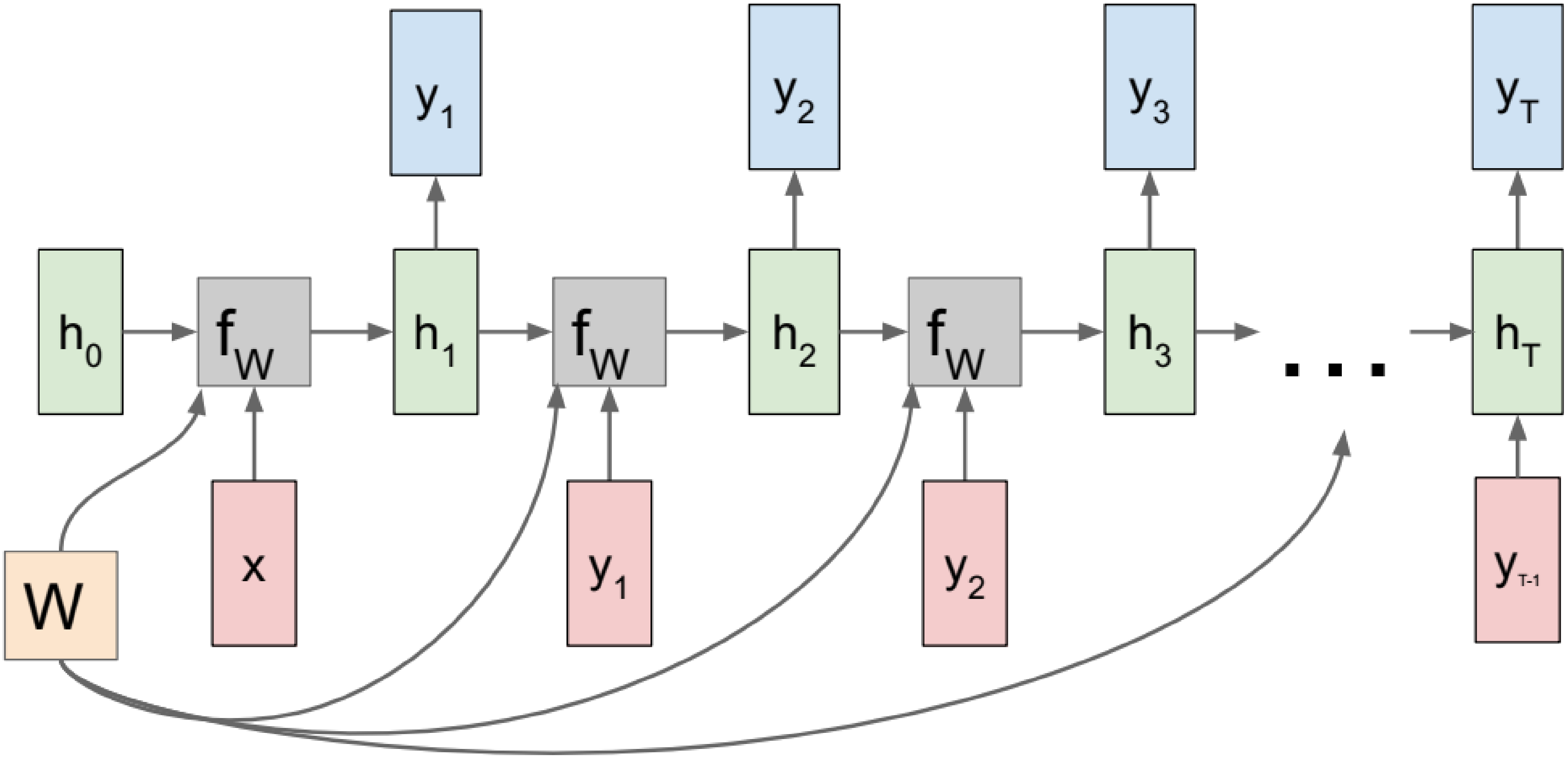
One to Many



One to Many

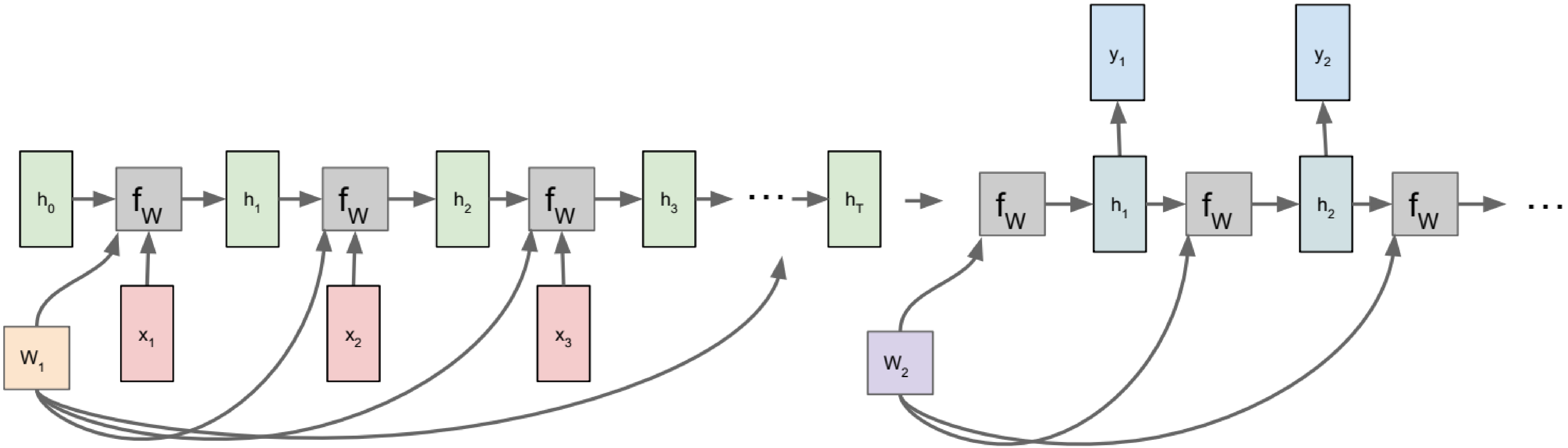


One to Many



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



پس‌انتشار در طول زمان

- که f و g تبدیل‌های مربوط به لایه پنهان و لایه خروجی هستند
  - w_h و w_o هم تمام پارامترهای این لایه‌ها هستند

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

$$o_t = g(h_t, w_o)$$

- تابع ضرر در حالت many to many به صورت زیر تعریف می‌شود:

$$L(x_1, \dots, x_T, y_1, \dots, y_T, w_h, w_o) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t)$$

- باید گرادیان این تابع ضرر نسبت به w_h و w_o را محاسبه کنیم

$$\frac{\partial L}{\partial w_o} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_o} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_o)}{\partial w_o}$$

پس‌انتشار در طول زمان

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

$$o_t = g(h_t, w_o)$$

$$L(x_1, \ldots, x_T, y_1, \ldots, y_T, w_h, w_o) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t)$$

$$\frac{\partial L}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l(y_t, o_t)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_o)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_o)}{\partial h_t}\frac{\partial h_t}{\partial w_h}$$

- محاسبه $\frac{\partial h_t}{\partial w_h}$ ساده نیست زیرا در محاسبه h_t به تعداد t بار از w_h به صورت بازگشتی استفاده می‌شود

پس‌انتشار در طول زمان

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

$$o_t = g(h_t, w_o)$$

$$L(x_1, \dots, x_T, y_1, \dots, y_T, w_h, w_o) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t)$$

$$\frac{\partial L}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_o)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_o)}{\partial h_t}\frac{\partial h_t}{\partial w_h}$$

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial w_h}$$

پس‌انتشار در طول زمان

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

$$o_t = g(h_t, w_o)$$

$$L(x_1, \ldots, x_T, y_1, \ldots, y_T, w_h, w_o) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t)$$

$$\frac{\partial L}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_o)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_o)}{\partial h_t}\frac{\partial h_t}{\partial w_h}$$

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1}\left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}}\right)\frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

پس‌انتشار در طول زمان

- زنجیره محاسبه گرادیان می‌تواند خیلی طولانی شود
- محاسبات کامل: می‌تواند خیلی کند باشد و گرادیان‌ها می‌توانند منفجر شوند (مشابه با اثر پروانه‌ای)
- کوتاه کردن گام‌های زمانی: در محاسبات فقط از τ زمان گذشته برای تقریب گرادیان استفاده می‌شود
  - مدل بر تأثیر کوتاه‌مدت بجای بلندمدت تمرکز می‌کند و تخمین را به سمت مدل‌های ساده‌تر و پایدارتر سوق می‌دهد
- کوتاه کردن تصادفی: طول دنباله در محاسبه گرادیان به صورت تصادفی تغییر می‌کند و ضریب می‌گیرد تا امید ریاضی گرادیان برابر با مقدار درست باشد

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

پس‌انتشار در طول زمان

- کوتاه کردن تصادفی: طول دنباله در محاسبه گرادیان به صورت تصادفی تغییر می‌کند و ضریب می‌گیرد تا امید ریاضی گرادیان برابر با مقدار درست باشد
- محاسبات در $\xi_t = 0$ متوقف می‌شود، طول دنباله‌ها متغیر خواهد بود و دنباله‌های طولانی هم به ندرت استفاده خواهند شد

$$0 \le \pi_t \le 1 \qquad \begin{cases} P(\xi_t = 0) = 1 - \pi_t \\ P(\xi_t = \pi_t^{-1}) = \pi_t \end{cases} \qquad \Rightarrow E[\xi_t] = 1$$

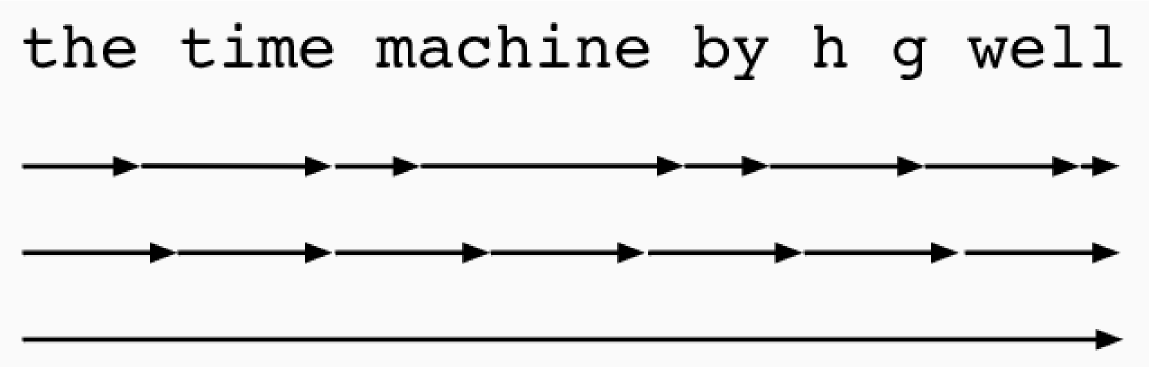
$$z_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \xi_t \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h} \qquad \Rightarrow E[z_t] = \frac{\partial h_t}{\partial w_h}$$

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

Tallec, Corentin, and Yann Ollivier. "Unbiasing truncated backpropagation through time." *arXiv preprint,* 2017.

پس‌انتشار در طول زمان

- زنجیره محاسبه گرادیان می‌تواند خیلی طولانی شود
- محاسبات کامل: می‌تواند خیلی کند باشد و گرادیان‌ها می‌توانند منفجر شوند (مشابه با اثر پروانه‌ای)
- کوتاه کردن گام‌های زمانی: در محاسبات فقط از τ زمان گذشته برای تقریب گرادیان استفاده می‌شود
  - مدل بر تأثیر کوتاه‌مدت بجای بلندمدت تمرکز می‌کند و تخمین را به سمت مدل‌های ساده‌تر و پایدارتر سوق می‌دهد
- کوتاه کردن تصادفی: طول دنباله در محاسبه گرادیان به صورت تصادفی تغییر می‌کند و ضریب می‌گیرد تا امید ریاضی گرادیان برابر با مقدار درست باشد



BPTT برای SimpleRNN

- برای سادگی از بایاس و تابع غیرخطی استفاده نمی‌کنیم

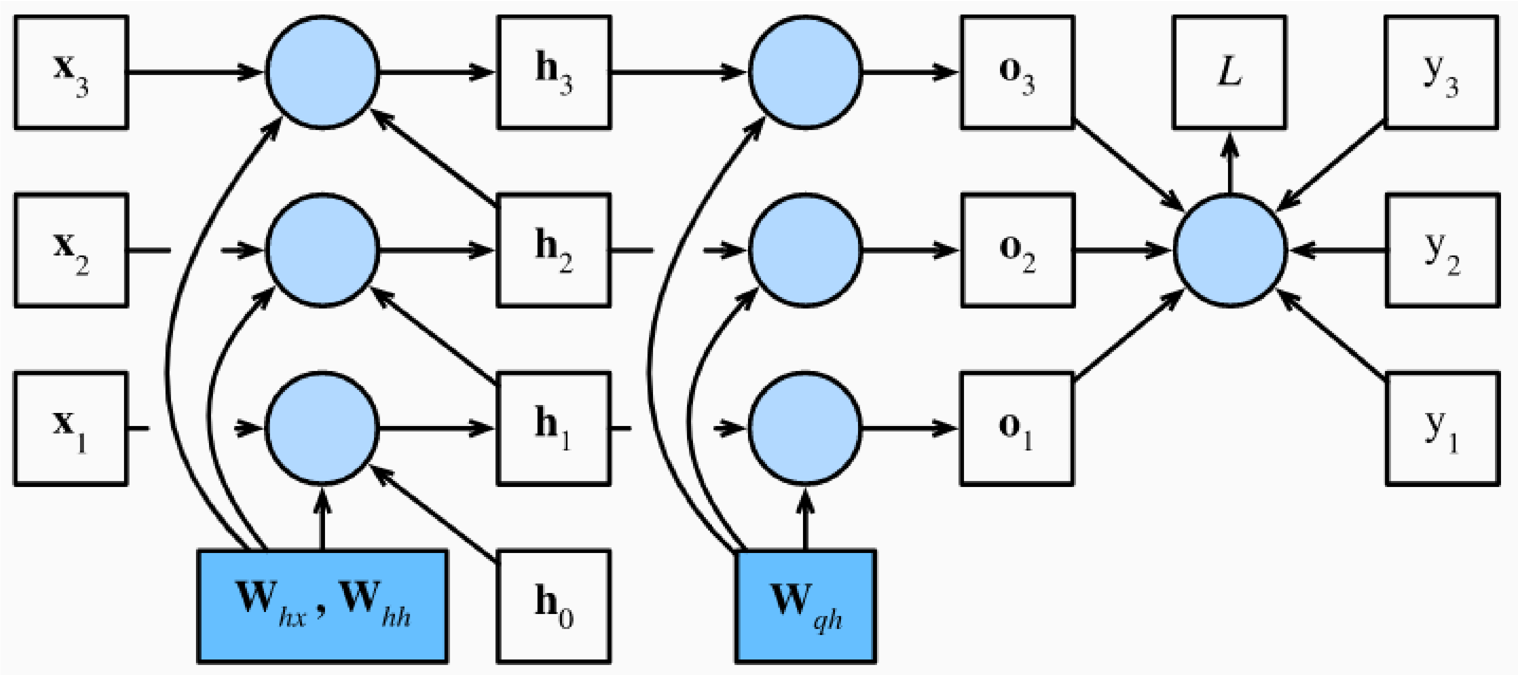
$$\mathbf{h}_t = \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1}$$

$$\mathbf{o}_t = \mathbf{W}_{qh}\mathbf{h}_t$$

$$L = \frac{1}{T}\sum_{t=1}^{T} l(\mathbf{y}_t, \mathbf{o}_t)$$

$$\frac{\partial L}{\partial \mathbf{o}_t} = \frac{\partial l(\mathbf{y}_t, \mathbf{o}_t)}{T\,\partial \mathbf{o}_t} \in \mathbb{R}^q$$

$$\frac{\partial L}{\partial \mathbf{W}_{qh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{o}_t}\mathbf{h}_t^{\mathrm{T}}$$



SimpleRNN برای BPTT

- برای سادگی از بایاس و تابع غیرخطی استفاده نمی‌کنیم

$$\mathbf{h}_t = \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1}$$

$$\mathbf{o}_t = \mathbf{W}_{qh}\mathbf{h}_t$$

$$L = \frac{1}{T}\sum_{t=1}^{T} l(\mathbf{y}_t, \mathbf{o}_t)$$

$$\frac{\partial L}{\partial \mathbf{o}_t} = \frac{\partial l(\mathbf{y}_t, \mathbf{o}_t)}{T\,\partial \mathbf{o}_t} \in \mathbb{R}^q$$

$$\frac{\partial L}{\partial \mathbf{W}_{qh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{o}_t}\mathbf{h}_t^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \mathbf{h}_T} = \mathrm{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_T}\right) = \mathbf{W}_{qh}^{\mathrm{T}}\frac{\partial L}{\partial \mathbf{o}_t}$$

$$\frac{\partial L}{\partial \mathbf{h}_t} = \mathrm{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}\right) + \mathrm{prod}\left(\frac{\partial L}{\partial \mathbf{h}_{t+1}}, \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}\right)$$

$$= \mathbf{W}_{qh}^{\mathrm{T}}\frac{\partial L}{\partial \mathbf{o}_t} + \mathbf{W}_{hh}^{\mathrm{T}}\frac{\partial L}{\partial \mathbf{h}_{t+1}}$$

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{i=t}^{T}\left(\mathbf{W}_{hh}^{\mathrm{T}}\right)^{T-i}\mathbf{W}_{qh}^{\mathrm{T}}\frac{\partial L}{\partial \mathbf{o}_{T+t-i}}$$

BPTT برای SimpleRNN

- برای سادگی از بایاس و تابع غیرخطی استفاده نمی‌کنیم
- زمانیکه دنباله طولانی باشد، به توان اعداد بزرگ می‌رسد و اگر مقادیر ویژه آن $\mathbf{W}_{hh}^{\mathrm{T}}$ بزرگتر از ۱ باشند دچار انفجار گرادیان و اگر کوچکتر از ۱ باشند دچار محوشدگی گرادیان خواهیم شد
- با کوتاه کردن گام‌های زمانی می‌توان این مشکل را تا حدی جبران کرد

$$\mathbf{h}_t = \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1}$$

$$\mathbf{o}_t = \mathbf{W}_{qh}\mathbf{h}_t$$

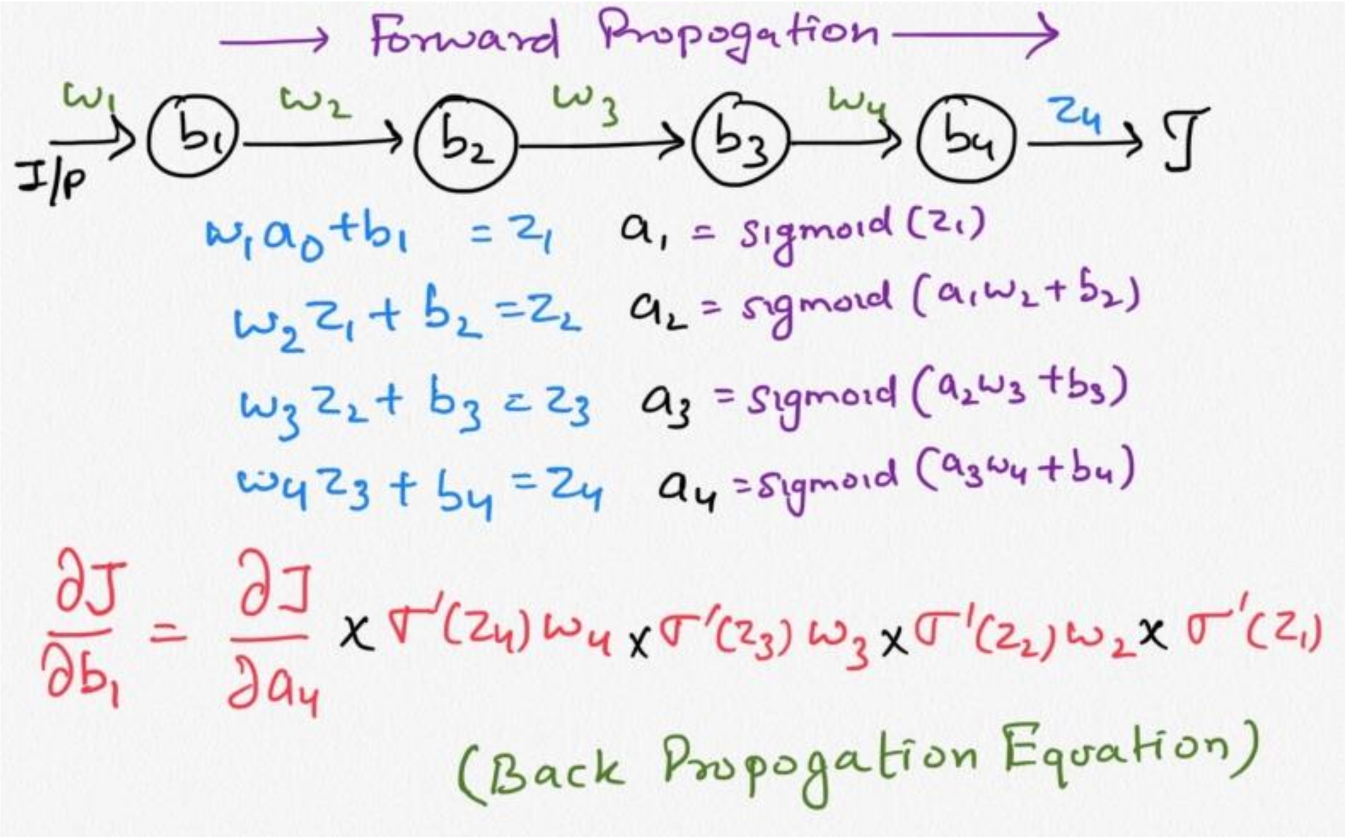
$$L = \frac{1}{T}\sum_{t=1}^{T} l(\mathbf{y}_t, \mathbf{o}_t)$$

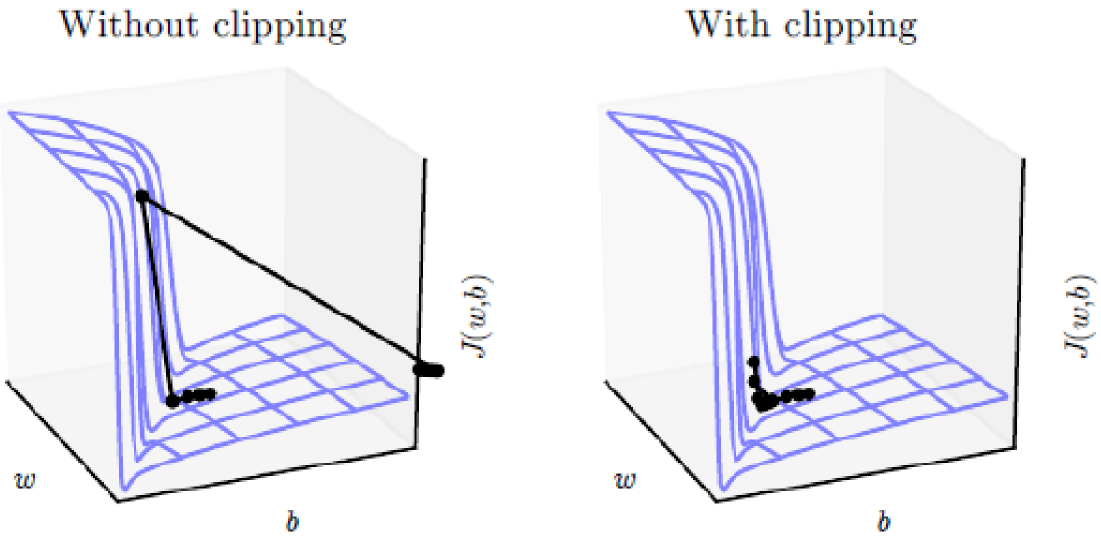
$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{i=t}^{T} \left(\mathbf{W}_{hh}^{\mathrm{T}}\right)^{T-i} \mathbf{W}_{qh}^{\mathrm{T}} \frac{\partial L}{\partial \mathbf{o}_{T+t-i}}$$

$$\frac{\partial L}{\partial \mathbf{W}_{hx}} = \sum_{i=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{x}_t^{\mathrm{T}} \qquad \frac{\partial L}{\partial \mathbf{W}_{hh}} = \sum_{i=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{h}_{t-1}^{\mathrm{T}}$$

محوشدگی و انفجار گرادیان

- در شبکه‌های بازگشتی برای دنباله‌های طولانی این مشکل بسیار جدی است
- می‌توان با برش گرادیان از انفجار گرادیان جلوگیری کرد

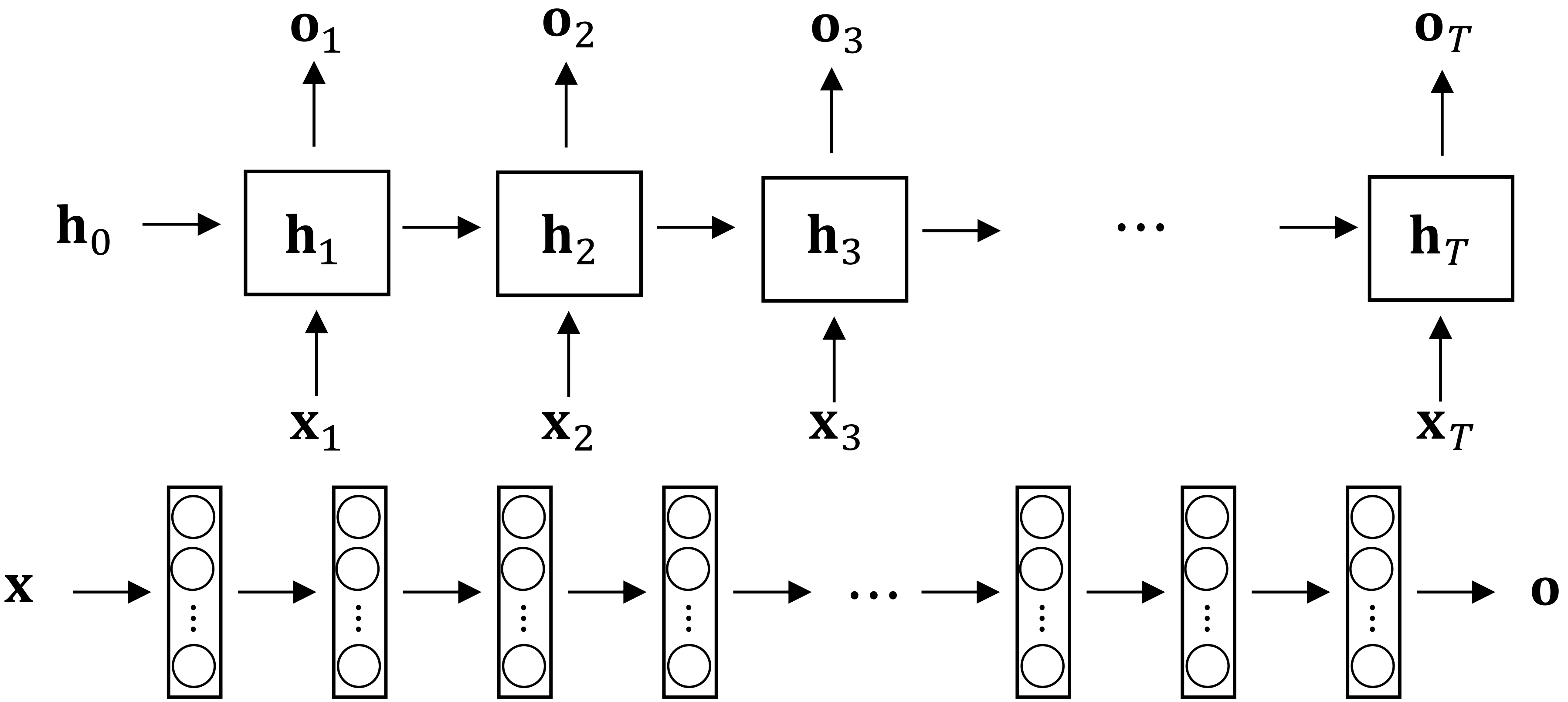




محوشدگی گرادیان در RNNها

The cat, which already ate …, was full.

The cats, which already ate …, were full.



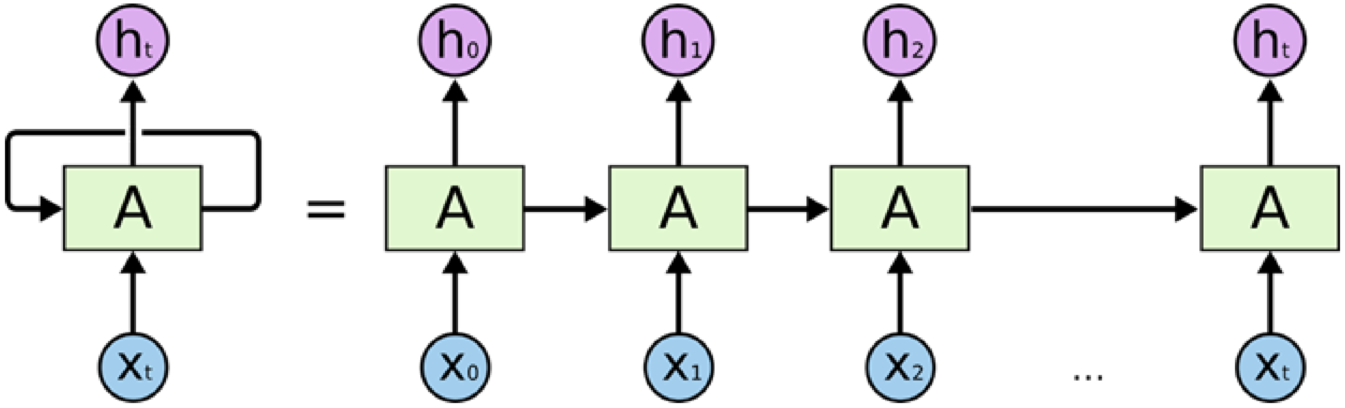
https://www.coursera.org/lecture/nlp-sequence-models/recurrent-neural-network-model-ftkzt

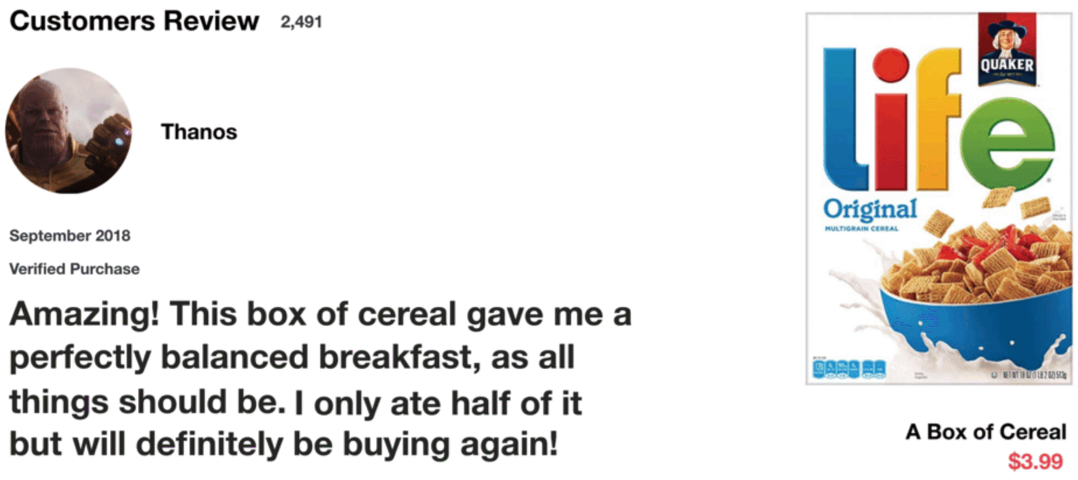
Gated RNNs

- شبکه‌های عصبی بازگشتی از حافظه کوتاه مدت رنج می‌برند
  - RNN ممکن است اطلاعات مهم ابتدایی را نادیده بگیرد
- Gated RNNs مبتنی بر ایده ایجاد مسیرهایی در طول زمان هستند که از محوشدگی یا انفجار گرادیان جلوگیری می‌کنند

The cat, which already ate …, was full.

The cats, which already ate …, were full.





Gated Recurrent Units

GRU (simplified)

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{b}_h)$$

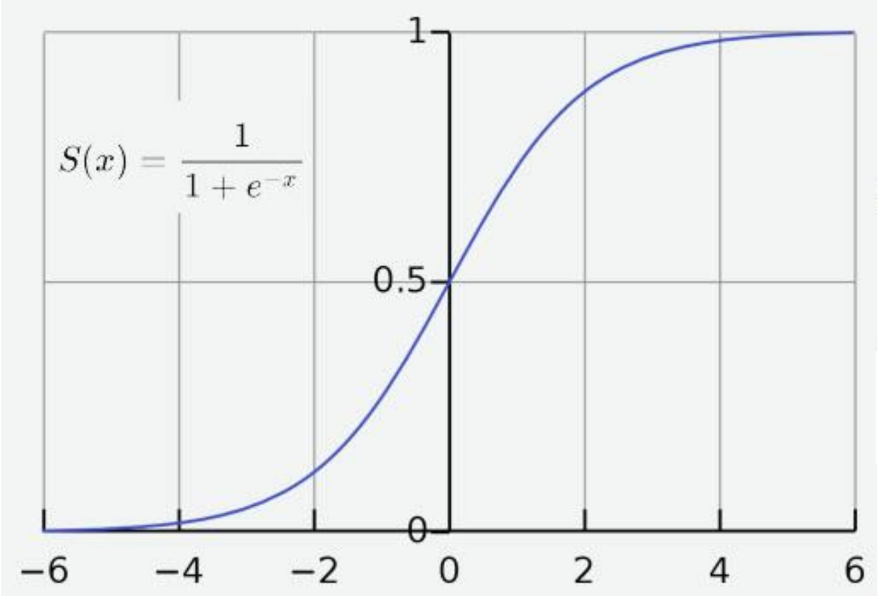
$$\mathbf{h}_t = \mathbf{u}_t.\tilde{\mathbf{h}}_t + (1 - \mathbf{u}_t).\mathbf{h}_{t-1}$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_{hu}\mathbf{h}_{t-1} + \mathbf{W}_{xu}\mathbf{x}_t + \mathbf{b}_u)$$

The cat, which already ate …, was full.

Simple RNN

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{b}_h)$$



Gated Recurrent Units

GRU (simplified)

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{b}_h)$$

$$\mathbf{h}_t = \mathbf{u}_t . \tilde{\mathbf{h}}_t + (1 - \mathbf{u}_t) . \mathbf{h}_{t-1}$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_{uh}\mathbf{h}_{t-1} + \mathbf{W}_{ux}\mathbf{x}_t + \mathbf{b}_u)$$

GRU

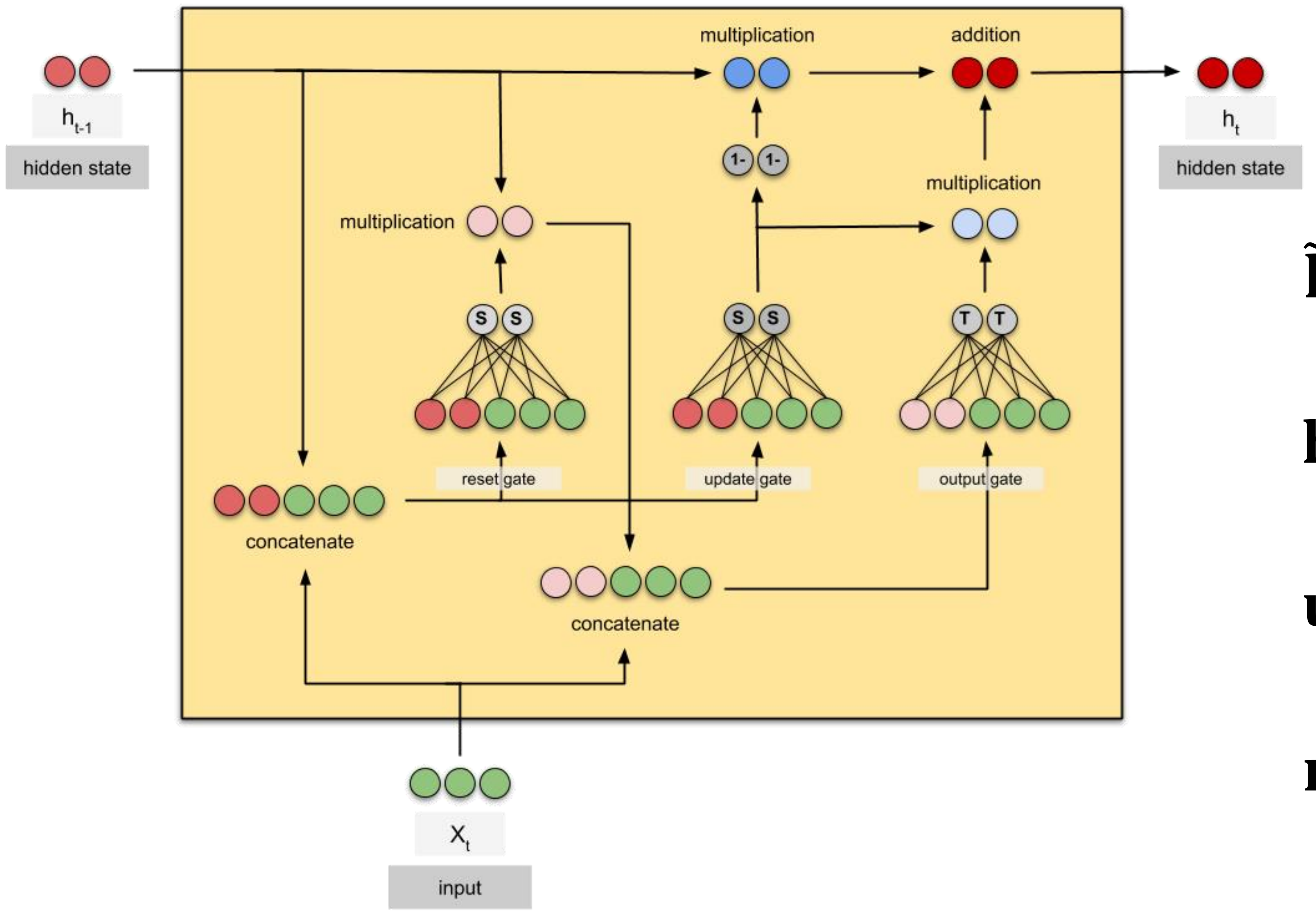
$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{hh}(\mathbf{r}_t . \mathbf{h}_{t-1}) + \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{b}_h)$$

$$\mathbf{h}_t = \mathbf{u}_t . \tilde{\mathbf{h}}_t + (1 - \mathbf{u}_t) . \mathbf{h}_{t-1}$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_{uh}\mathbf{h}_{t-1} + \mathbf{W}_{ux}\mathbf{x}_t + \mathbf{b}_u)$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{rh}\mathbf{h}_{t-1} + \mathbf{W}_{rx}\mathbf{x}_t + \mathbf{b}_r)$$

GRU



$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{hh}(\mathbf{r}_t.\mathbf{h}_{t-1}) + \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{b}_h)$$

$$\mathbf{h}_t = \mathbf{u}_t.\tilde{\mathbf{h}}_t + (1 - \mathbf{u}_t).\mathbf{h}_{t-1}$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_{uh}\mathbf{h}_{t-1} + \mathbf{W}_{ux}\mathbf{x}_t + \mathbf{b}_u)$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{rh}\mathbf{h}_{t-1} + \mathbf{W}_{rx}\mathbf{x}_t + \mathbf{b}_r)$$

Long Short-Term Memory

LSTM

$$\mathbf{h}_t = \mathbf{o}_t . \tanh(\mathbf{c}_t)$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{W}_{cx}\mathbf{x}_t + \boldsymbol{b}_c)$$

$$\mathbf{c}_t = \mathbf{u}_t . \tilde{\mathbf{c}}_t + \mathbf{f}_t . \mathbf{c}_{t-1}$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_{uh}\mathbf{h}_{t-1} + \mathbf{W}_{ux}\mathbf{x}_t + \mathbf{b}_u)$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fx}\mathbf{x}_t + \mathbf{b}_f\right)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{ox}\mathbf{x}_t + \mathbf{b}_o)$$

GRU

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{hh}(\mathbf{r}_t . \mathbf{h}_{t-1}) + \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{b}_h)$$

$$\mathbf{h}_t = \mathbf{u}_t . \tilde{\mathbf{h}}_t + (1 - \mathbf{u}_t) . \mathbf{h}_{t-1}$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_{uh}\mathbf{h}_{t-1} + \mathbf{W}_{ux}\mathbf{x}_t + \mathbf{b}_u)$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{rh}\mathbf{h}_{t-1} + \mathbf{W}_{rx}\mathbf{x}_t + \mathbf{b}_r)$$

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation, 1997.

LSTM

$$\mathbf{h}_t = \mathbf{o}_t . \tanh(\mathbf{c}_t)$$

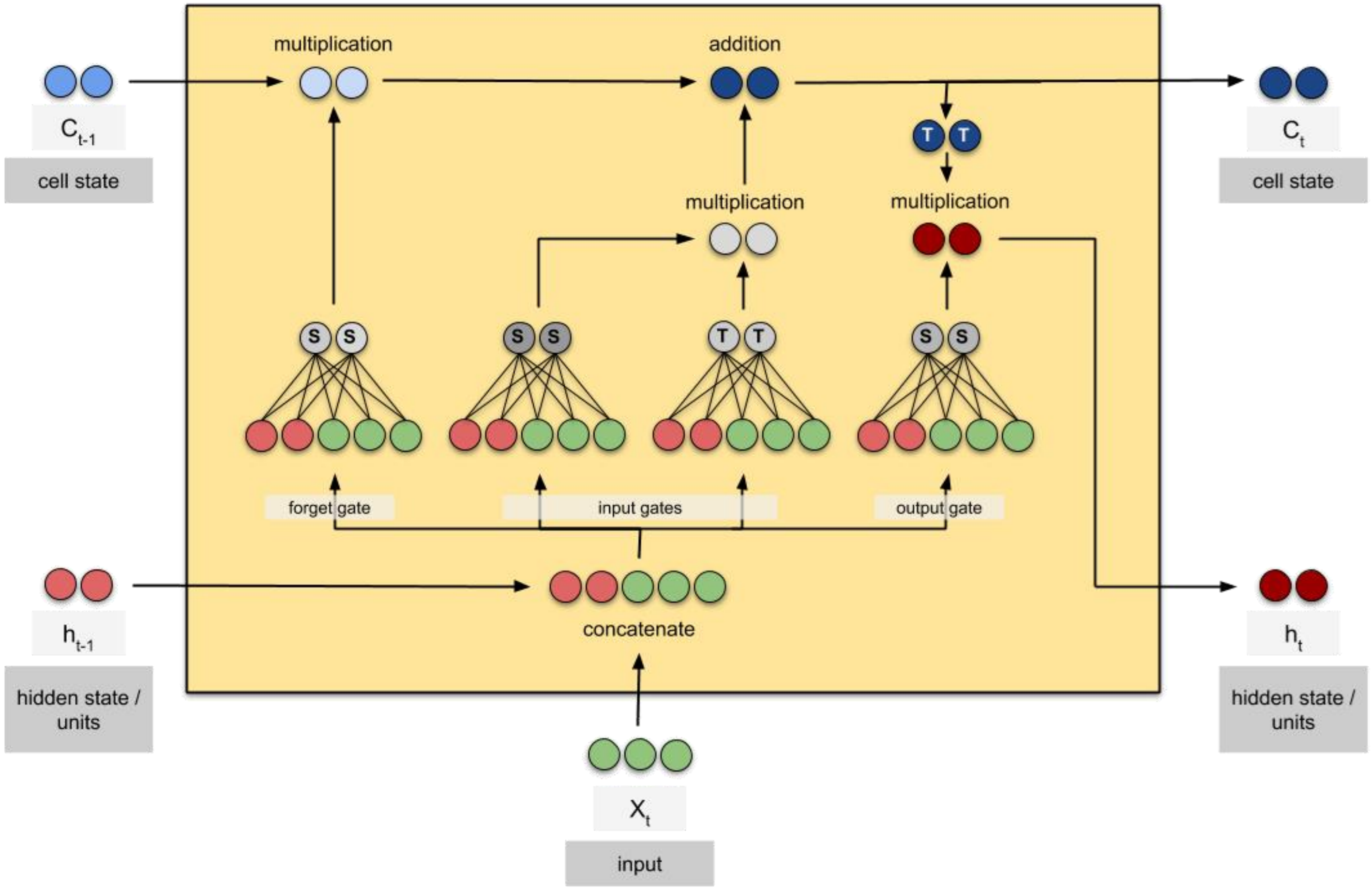
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{W}_{cx}\mathbf{x}_t + \boldsymbol{b}_c)$$

$$\mathbf{c}_t = \mathbf{u}_t . \tilde{\mathbf{c}}_t + \mathbf{f}_t . \mathbf{c}_{t-1}$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_{uh}\mathbf{h}_{t-1} + \mathbf{W}_{ux}\mathbf{x}_t + \mathbf{b}_u)$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fx}\mathbf{x}_t + \mathbf{b}_f\right)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{ox}\mathbf{x}_t + \mathbf{b}_o)$$



https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45

LSTM

```python
def LSTMCELL(prev_ct, prev_ht, input):
    combine = prev_ht + input
    ft = forget_layer(combine)
    candidate = candidate_layer(combine)
    it = input_layer(combine)
    Ct = prev_ct * ft + candidate * it
    ot = output_layer(combine)
    ht = ot * tanh(Ct)
    return ht, Ct


ct = [0, 0, 0]
ht = [0, 0, 0]

for input in inputs:
    ct, ht = LSTMCELL(ct, ht, input)
```