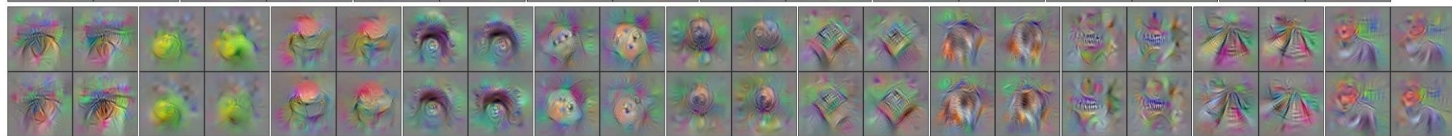
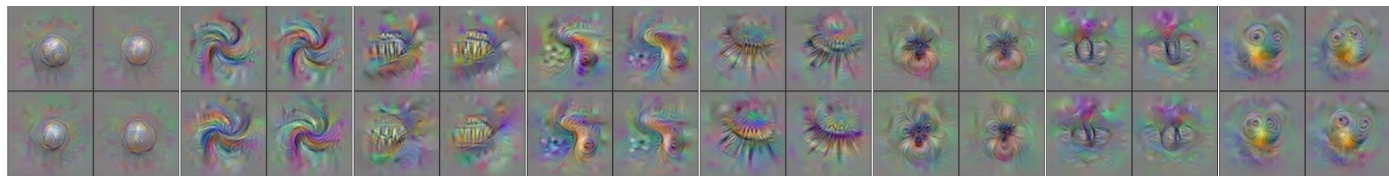
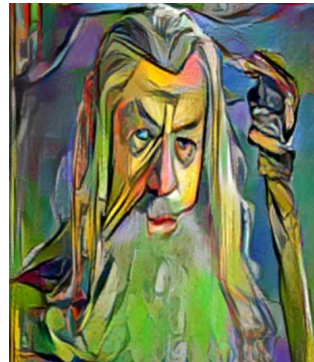
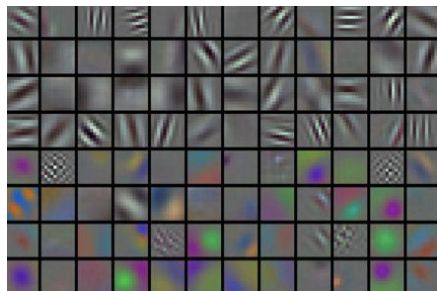
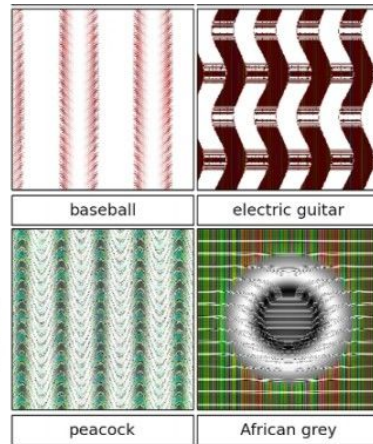


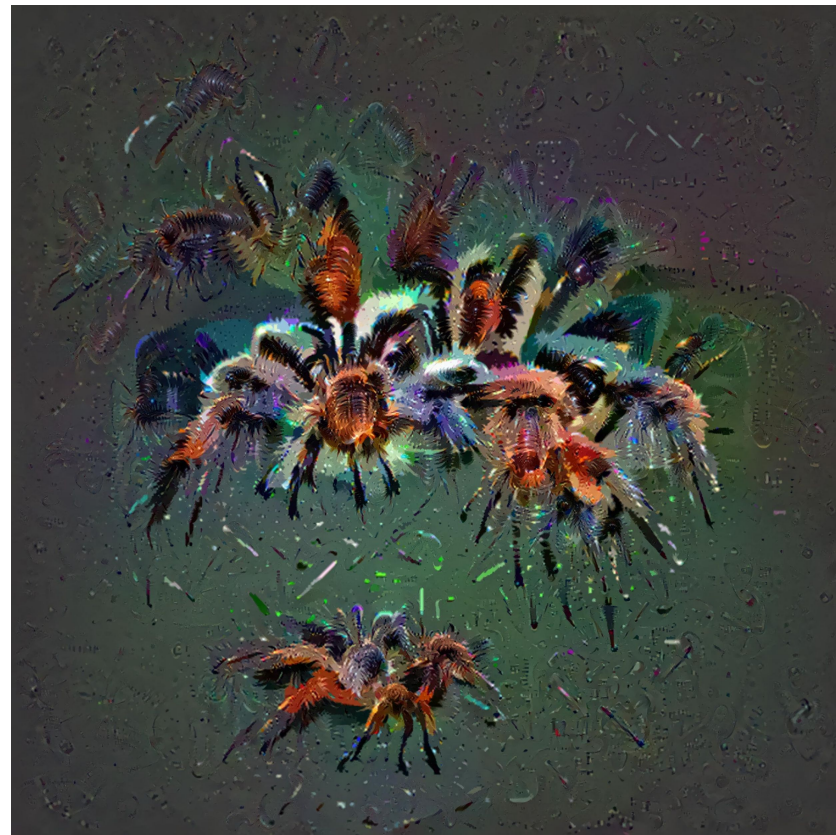
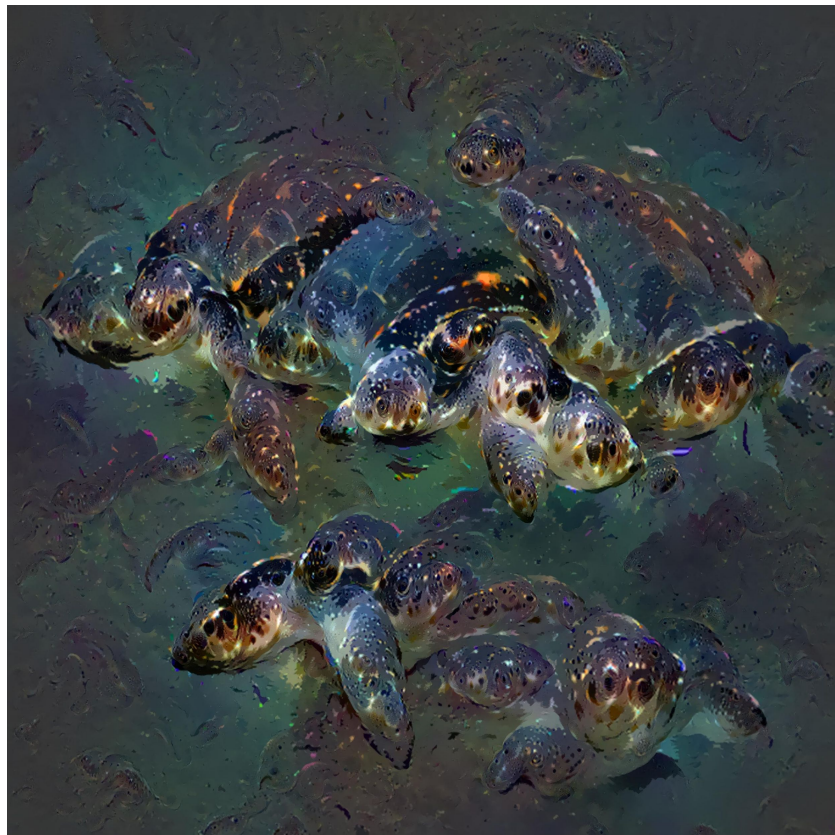
Lecture 10:

Recurrent Neural Networks



Layer 1

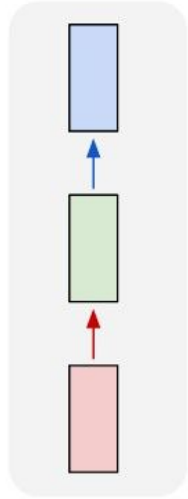




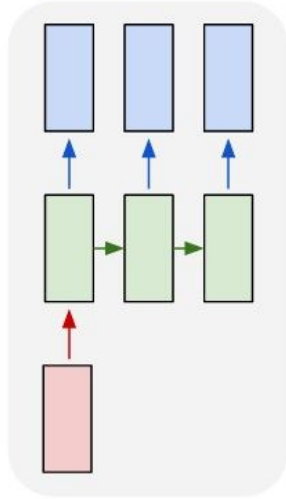


Recurrent Networks offer a lot of flexibility:

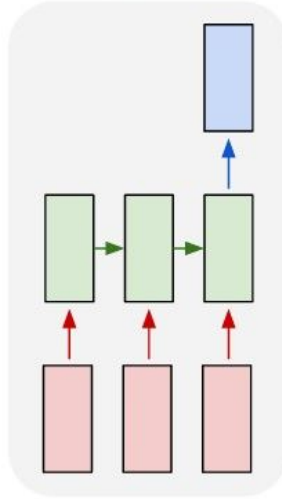
one to one



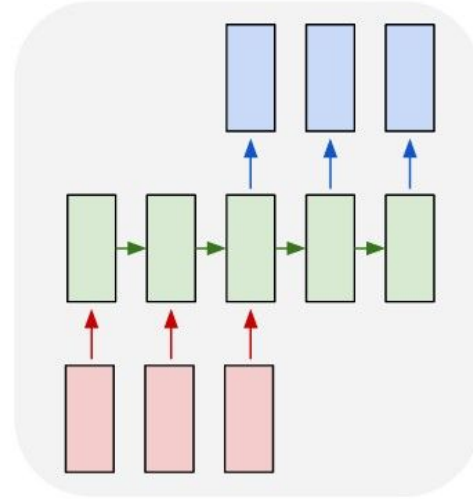
one to many



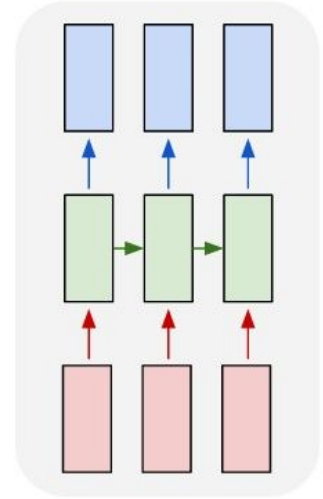
many to one



many to many



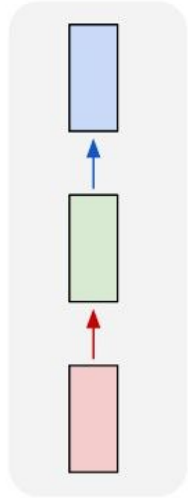
many to many



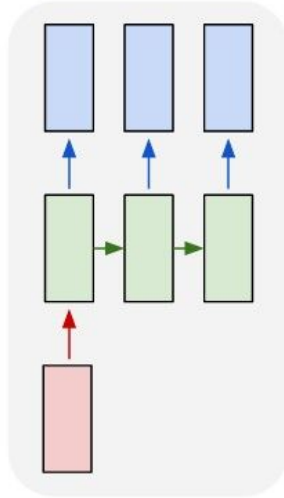
↖ **Vanilla Neural Networks**

Recurrent Networks offer a lot of flexibility:

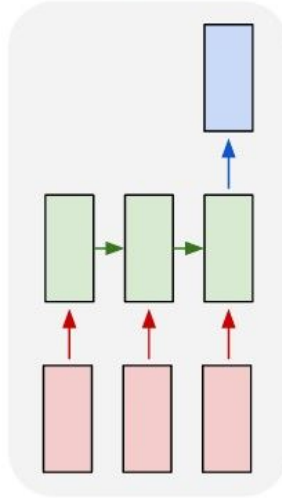
one to one



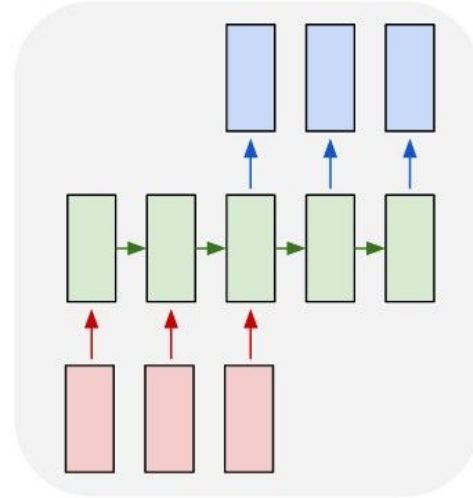
one to many



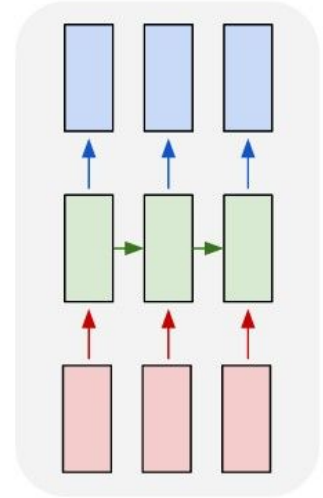
many to one



many to many



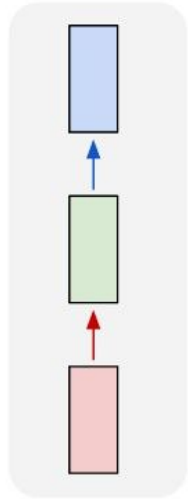
many to many



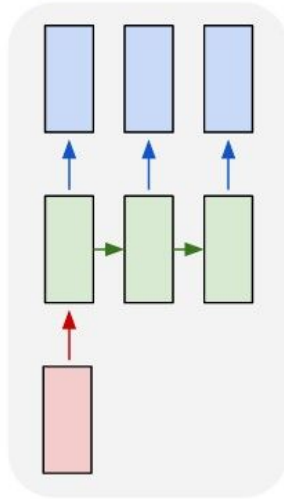
↖ e.g. **Image Captioning**
image -> sequence of words

Recurrent Networks offer a lot of flexibility:

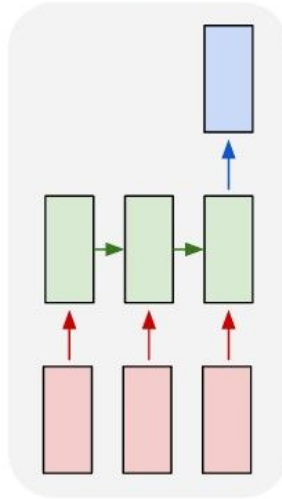
one to one



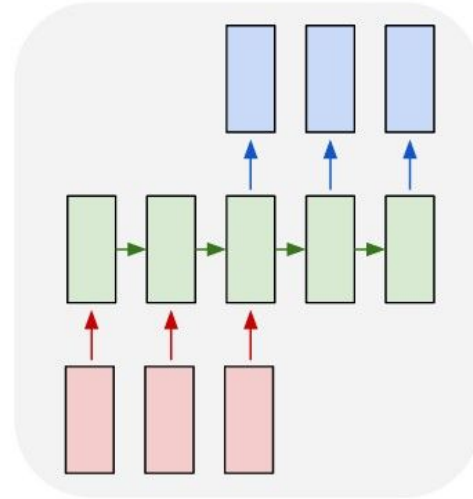
one to many



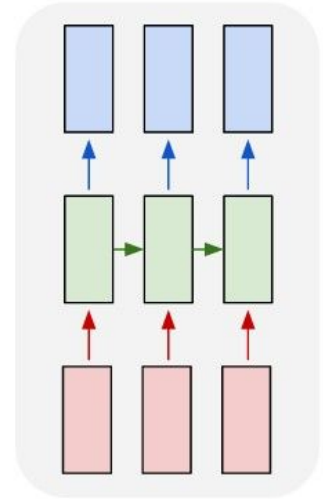
many to one



many to many



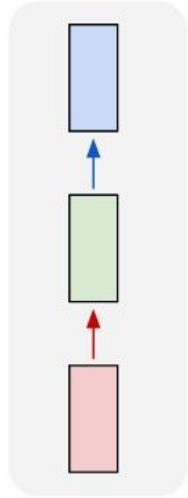
many to many



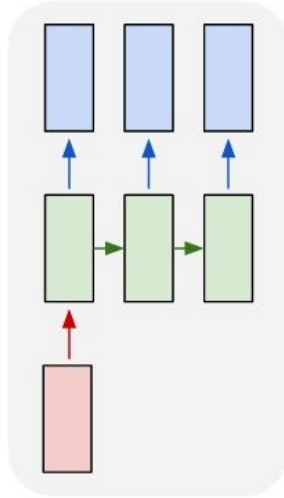
↖ e.g. **Sentiment Classification**
sequence of words → sentiment

Recurrent Networks offer a lot of flexibility:

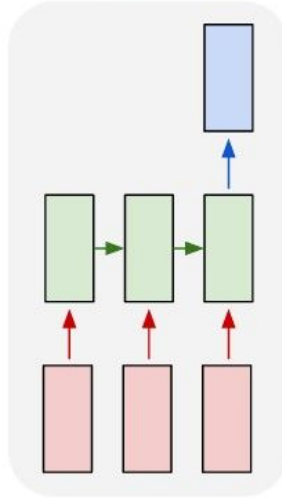
one to one



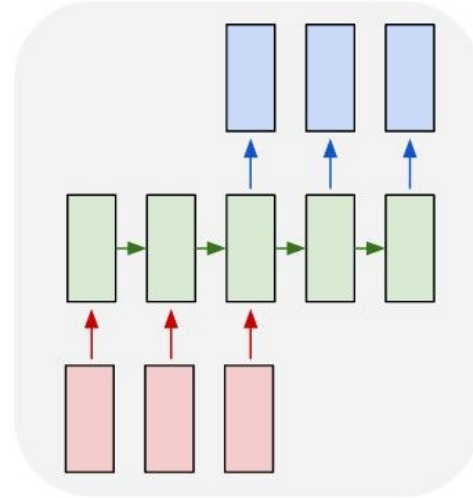
one to many



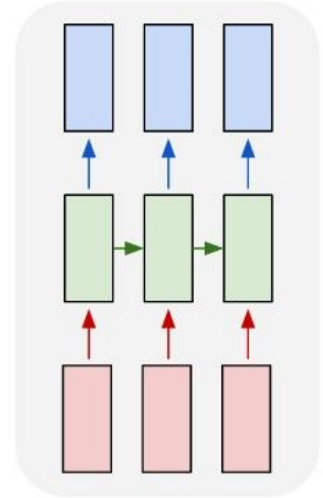
many to one



many to many



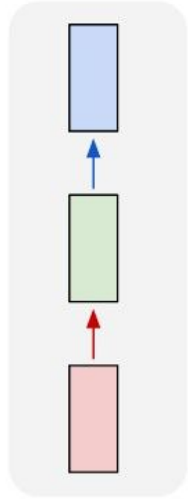
many to many



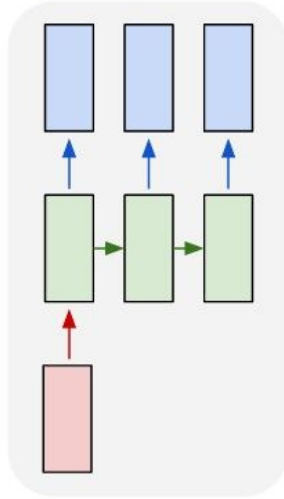
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Networks offer a lot of flexibility:

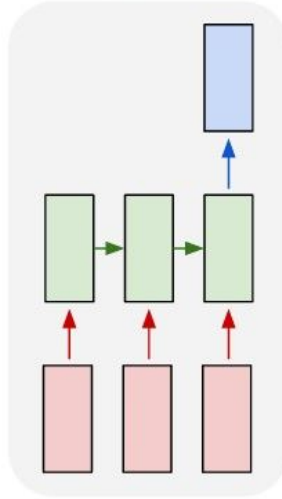
one to one



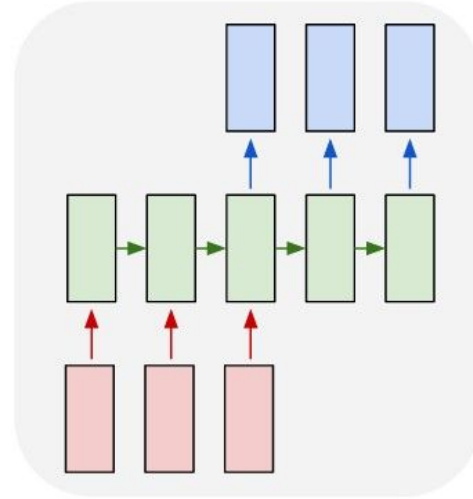
one to many



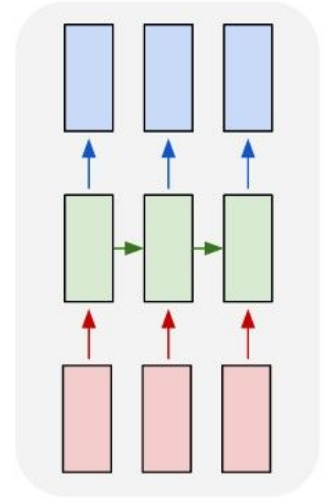
many to one



many to many



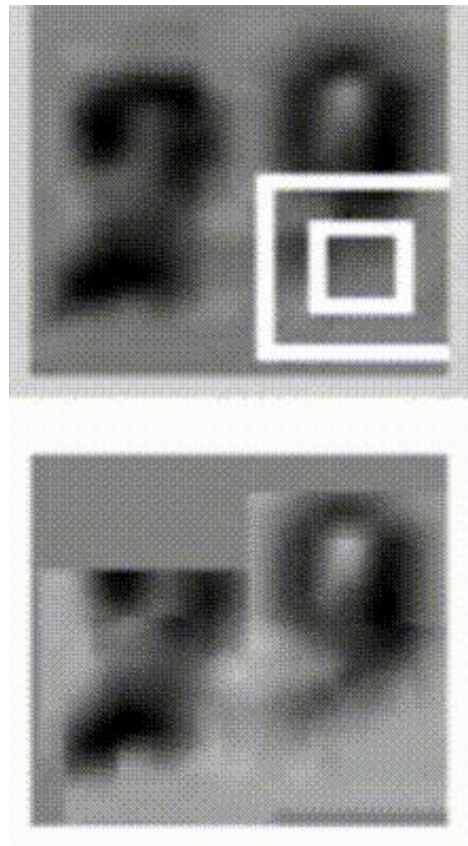
many to many



e.g. Video classification on frame level



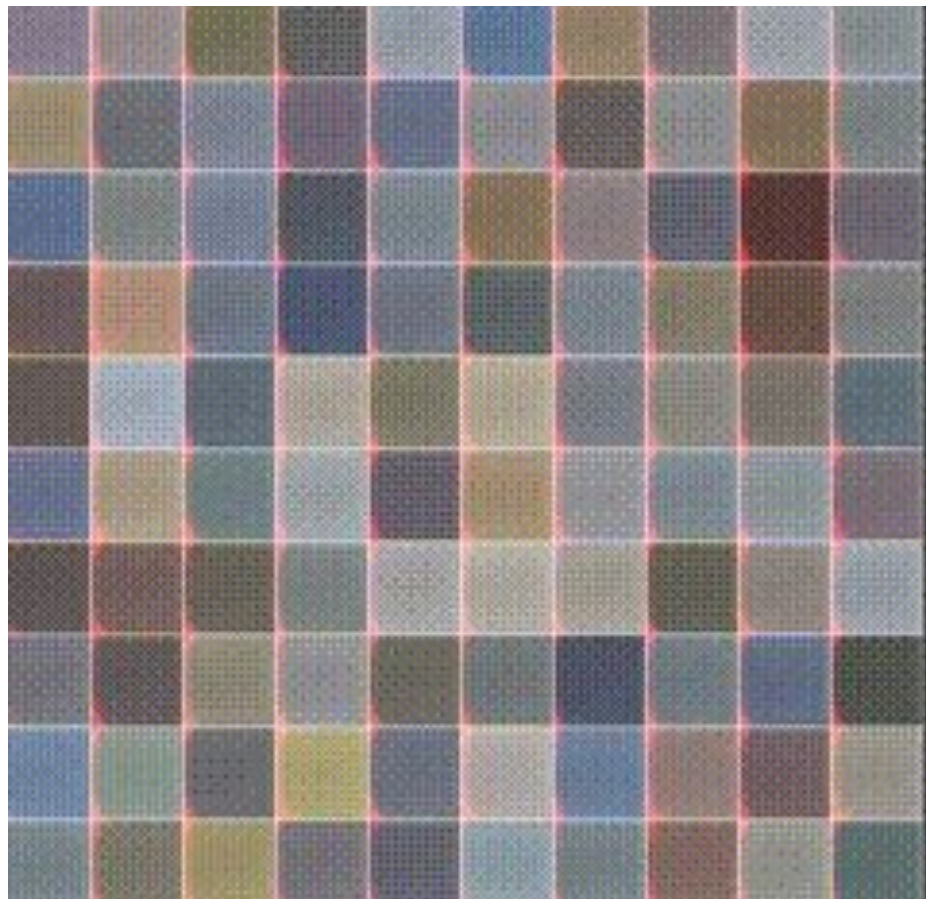
Sequential Processing of fixed inputs



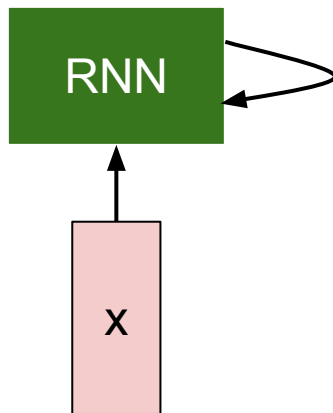
Multiple Object Recognition with
Visual Attention, Ba et al.

Sequential Processing of fixed outputs

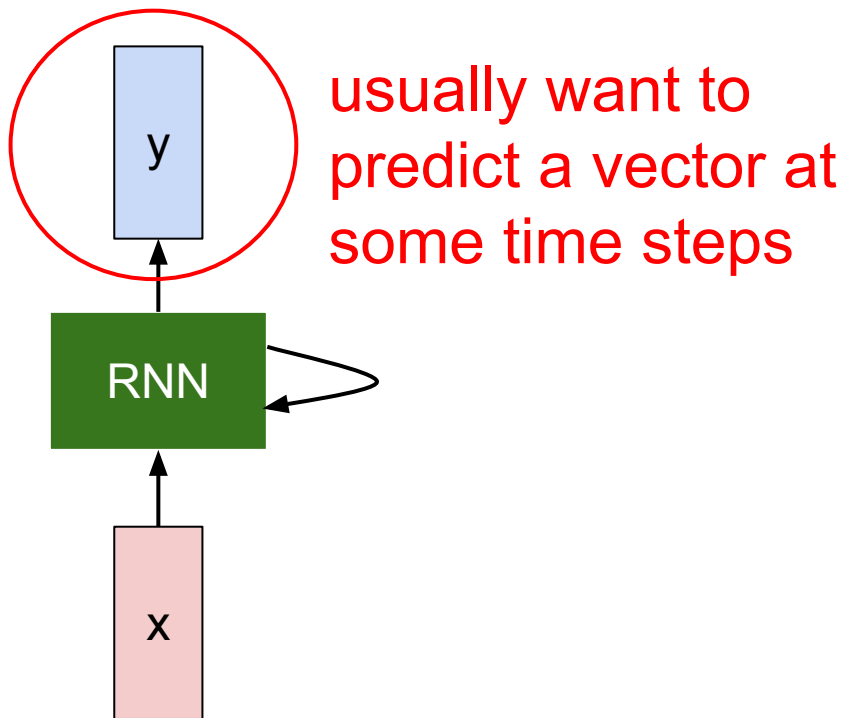
DRAW: A Recurrent
Neural Network For
Image Generation,
Gregor et al.



Recurrent Neural Network



Recurrent Neural Network



Recurrent Neural Network

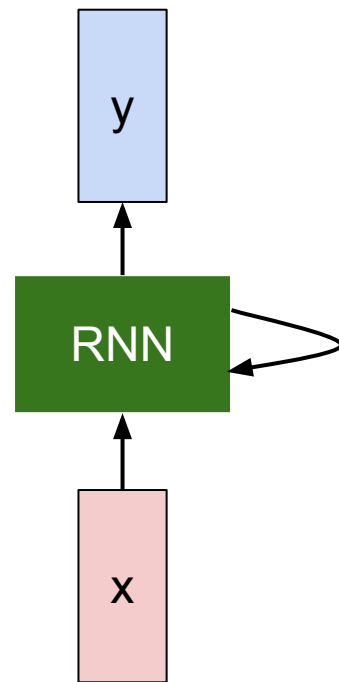
We can process a sequence of vectors $\mathbf{x}$ by applying a recurrence formula at every time step:

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

new state / some function with parameters W

old state

input vector at some time step

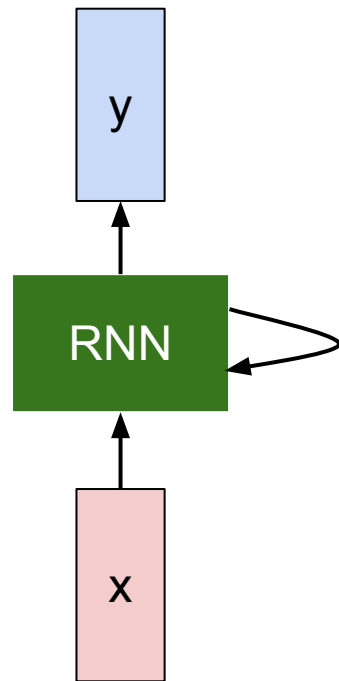


Recurrent Neural Network

We can process a sequence of vectors $\mathbf{x}$ by applying a recurrence formula at every time step:

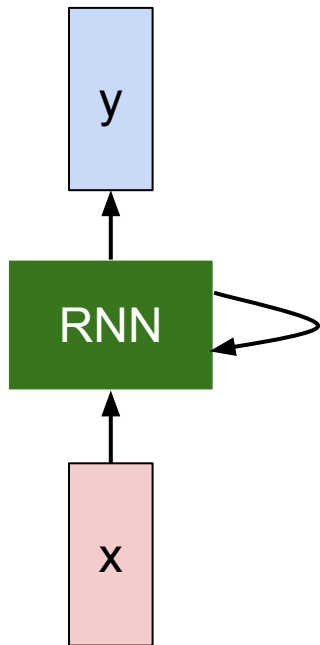
$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.



(Vanilla) Recurrent Neural Network

The state consists of a single “*hidden*” vector h :



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



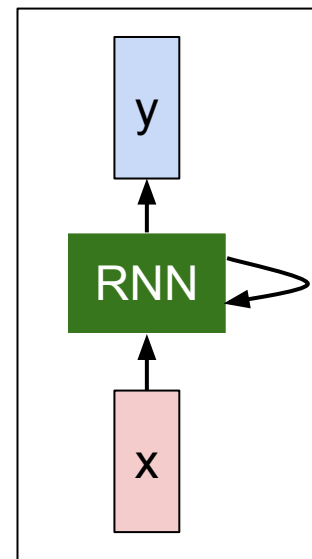
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Character-level language model example

Vocabulary:
[h,e,l,o]

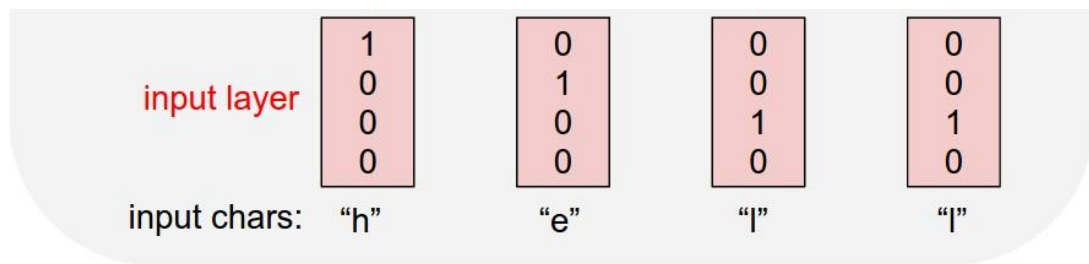
Example training
sequence:
“hello”



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

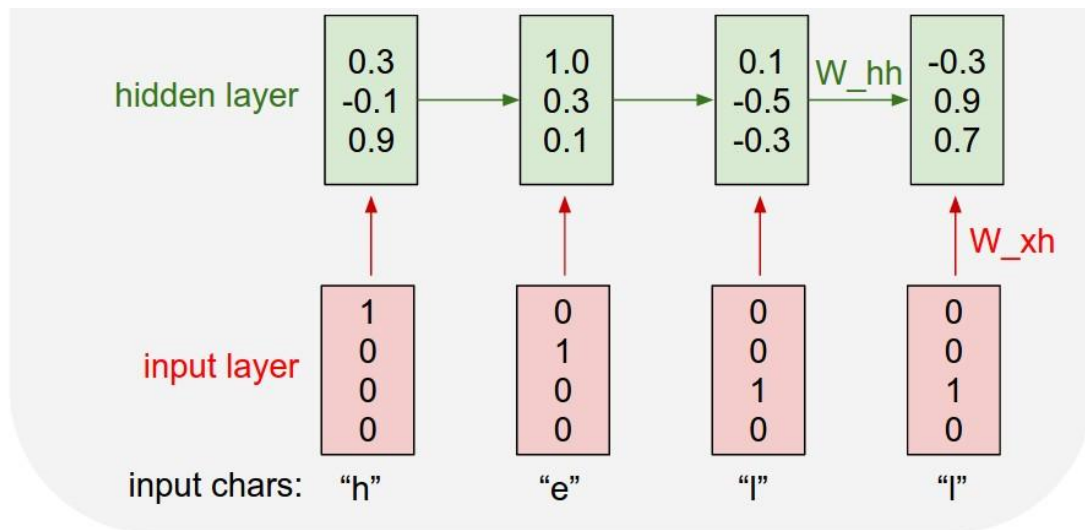


Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

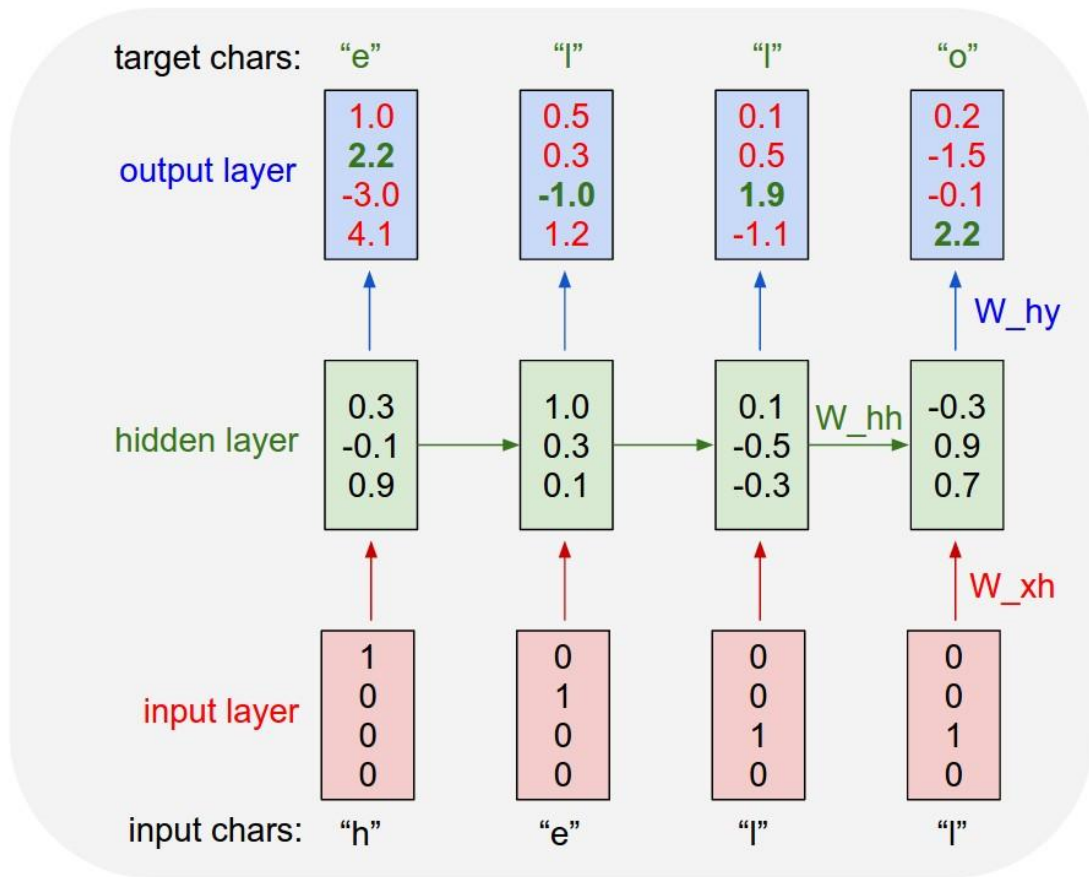
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



min-char-rnn.py gist: 112 lines of Python

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  # data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is Nx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(wxh, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backwards
45     dwxh, dwhh, dwhy = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(why)
46     dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47     dhnext = np.zeros_like(hs[0])
48     for t in reversed(xrange(len(inputs))):
49         dy = np.copy(ps[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dwhy += np.dot(dy, hs[t].T)
52         dby += dy
53         dh = np.dot(why.T, dy) + dhnext # backprop into h
54         dhrdw = (1 - hs[t]**2) * hs[t]**2 * dh # backprop through tanh nonlinearity
55         dbh += dhrdw
56         dwxh += np.dot(dhrdw, xs[t].T)
57         dwhh += np.dot(dhrdw, hs[t-1].T)
58         dhnext = np.dot(whh.T, dhrdw)
59     for dparam in [dwxh, dwhh, dwhy, dbh, dby]:
60         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61     return loss, dwxh, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
62
63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     x[seed_ix] = 1
70     ixes = []
71     for t in xrange(n):
72         h = np.tanh(np.dot(wxh, x) + np.dot(whh, h) + bh)
73         y = np.dot(why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p.ravel())
76         x = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79     return ixes
80
81 n, p = 0, 0
82 mwxh, mwhh, mwhy = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print '----%s\n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100    loss, dwxh, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101    smooth_loss = smooth_loss * 0.999 + loss * 0.001
102    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104    # perform parameter update with Adagrad
105    for param, dparam, mem in zip([dwxh, dwhh, dwhy, dbh, dby],
106                                  [mwxh, mwhh, mwhy, mbh, mby]):
107        mem += dparam * dparam
108        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
109
110    p += seq_length # move data pointer
111    n += 1 # iteration counter
```

(<https://gist.github.com/karpathy/d4dee566867f8291f086>)

[min-char-rnn.py](#) gist

```

20
21 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
22 BSD license
23
24 import numpy as np
25
26 # data file
27 data = open('input.txt', 'r').read() # should be simple plain text file
28 chars = list(set(data))
29 data_size, vocab_size = len(data), len(chars)
30 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
31 char_to_ix = { ch:i for i,ch in enumerate(chars) }
32 ix_to_char = { i:ch for i,ch in enumerate(chars) }

```

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

```
# model parameters
w0h = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
w1h = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
w1y = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
b0h = np.zeros((hidden_size, 1)) # hidden bias
b1y = np.zeros((vocab_size, 1)) # output bias
```

```

79 inputs,targets are both list of integers.
80 hprev is Nx1 array of initial hidden state
81 returns the loss, gradients on model parameters, and last hidden state

```

```
xs, hs, ys, ps = {}, {}, {}, {}
hs[-1] = np.copy(hprev)
```

```

# 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(w_hh, xs[t]) + np.dot(w_hb, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(W_hy, hs[t]) + by # unnormalized log probabilities for next char
    loss += -np.exp[ys[t]] / np.sum(np.exp[ys[t]]) # probabilities for next char
loss += -np.dot(pst[t][targets[t],0]) # softmax (cross-entropy loss)

```

```
dhx, dhxh, dmy = np.zeros_like(mxh), np.zeros_like(whh), np.zeros_like(hxh)
dbh, dby = np.zeros_like(bh), np.zeros_like(by)
dhnext = np.zeros_like(hs[0])
```

```

59 dy = np.copy(ps[t])
60 dy[targets[t]] -= 1 * backprop into y
61 dWxy += np.dot(dy, hs[t].T)

```

```

53 dh = np.dot(why.T, dy) + dhnext # backprop into h
54 dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55 dbh += dhraw

```

```

57     dwhh += np.dot(dhraw, hs[t-1].T)
58     dhnext = np.dot(whh.T, dhraw)
59     for dparam in [dwxh, dwhh, dwh, dbh, dby]:

```

```

61     return loss, dwxh, dwhh, dwxy, dbh, dby, hs[len(inputs)-1]
62
63 def sample(h, seed_ix, n):
64     """

```

```

# is memory state, seed_ix is seed letter for first time step
...
x = np.zeros((vocab_size, 1))
x[seed_ix] = 1

```

```
71 for t in xrange(n):
72     h = np.tanh(np.dot(Wxh, x) + rp.dot(Wrh, h) + bh)
73     y = np.dot(Wyh, h) + by
```

```
ix = np.random.choice(range(vocab_size), p=p.ravel())
x = np.zeros([vocab_size, 1])
x[ix] = 1
ixes.append(ix)
```

```
80
81 a, p = 0, 0
82 m0xh, m0hh, m0hy = np.zeros_like(w0h), np.zeros_like(w0hh), np.zeros_like(w0hy)
```

```
smooth_1000 = np.array(1000, dtype=float) * seq_length * 1000 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:
```

```

39 p = 0 # go from start of data
40 inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
41 targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

```

```

94     if m % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)

```

```

98
99 * forward seq_length characters through the net and fetch gradient
100 loss, dwhx, dwhxh, dwh, dbh, dbh, hprev = lossFun(inputs, targets, hprev)
101 smooth_loss = smooth_loss + smooth_loss + loss + loss + loss

```

```
83
84 # perform parameter update with Adagrad
85 for param, dparam, mem in zip([wvec, wvec, wvec, bvec, bvec],
```

```

    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

```

```
12 # 1 = iteration counter
```

Data I/O

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
```

```

###
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
800 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
w_h = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
w_hh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
w_y = np.random.randn(hidden_size, vocab_size)*0.01 # hidden to output
b_h = np.zeros((hidden_size, 1)) # hidden bias
b_y = np.zeros((vocab_size, 1)) # output bias
```

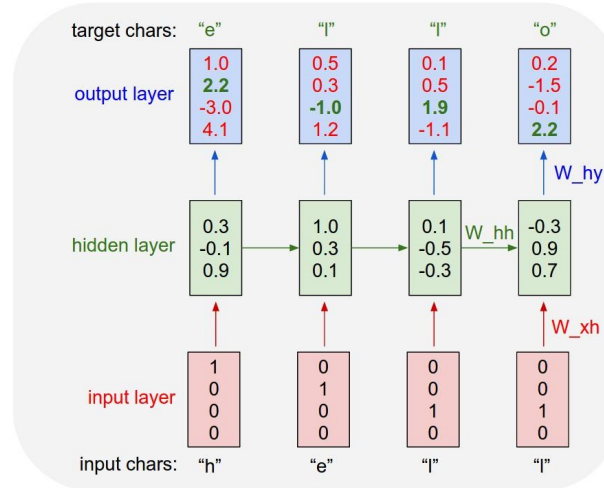
[illegible][illegible]

```

15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias

```

recall:



min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 b1 = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is vec array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     es, hs, ys, ps = [], [], [], []
34     h1 = hprev
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi = np.zeros((vocab_size, 1)) # encode in 1-of-K representation
39         xi[ix_to_char[t]] = 1
40         h1 = np.tanh(np.dot(wih, xi) + np.dot(whh, h1[-1]) + bh) # hidden state
41         yi = np.dot(why, h1) + by # unnormalized log probabilities for next chars
42         ps = np.exp(yi) / np.sum(np.exp(yi)) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backward
45     dht, dbh, dby = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
46     dht = np.zeros_like(h1)
47     for t in reversed(xrange(len(inputs))):
48         dy = np.zeros((1,))
49         dy[targets[t]] -= 1 # backprop into y
50         dby += np.dot(dy, h1[t:T])
51         dy = dy
52         dh = np.dot(dy, dy) # dnext = backprop into h
53         dhrw = (1 - h1[t] * h1[t]) * dh # backprop through tanh nonlinearity
54         dht += np.dot(dhrw, w1[t:T])
55         dht = np.dot(dht, whh)
56         dnext = np.dot(whh, T, dhrw)
57         for dparam in [dht, dbh, dby, dht, dby]:
58             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
59     return loss, dht, dbh, dby, dht, dby, h1[len(inputs)-1]
60
61 def sample(h, ix):
62     """
63     sample a sequence of integers from the model
64     h is memory state, seed, ix is seed letter for first time step
65     """
66     x = np.zeros((vocab_size, 1))
67     ix = ix
68     ix = ix
69     for t in xrange(1):
70         xi = np.zeros((vocab_size, 1))
71         xi[ix_to_char[ix]] = 1
72         h = np.tanh(np.dot(wih, xi) + np.dot(whh, h) + bh)
73         y = np.dot(why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p, axis=-1)
76         x = np.zeros((vocab_size, 1))
77         ix[ix] = 1
78         loss += -np.log(p[ix, 0])
79     return loss
80
81 n, p = 0, 0
82 mbh, mby = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
83 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
84
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print '----\n %s \n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100     loss, dwih, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101     smooth_loss = smooth_loss * 0.999 + loss * 0.001
102     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104     # perform parameter update with Adagrad
105     for param, dparam, mem in zip([wih, whh, why, bh, by],
106                                   [dwih, dwhh, dwhy, dbh, dby],
107                                   [mbh, mby]):
108         mem += dparam * dparam
109         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111     p += seq_length # move data pointer
112     n += 1 # iteration counter
```

Main loop

```
81 n, p = 0, 0
82 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print '----\n %s \n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100     loss, dwxh, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101     smooth_loss = smooth_loss * 0.999 + loss * 0.001
102     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104     # perform parameter update with Adagrad
105     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
106                                   [dwxh, dwhh, dwhy, dbh, dby],
107                                   [mWxh, mWhh, mWhy, mbh, mby]):
108         mem += dparam * dparam
109         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111     p += seq_length # move data pointer
112     n += 1 # iteration counter
```

```

8  n, p = 0, 0
9  much, many, many_p = np.zeros_like(n), np.zeros_like(many), np.zeros_like(many_p)
10 much_np = np.array_like(n), np.zeros_like(many) # many variables for Adagrad
11 much_np = np.zeros_like(much_np) # much_np is not used in this iteration
12 while True:
13     # prepare inputs for the softmax layer left to right in steps and lengths long]
14     if param_lengths == len(steps) == 0:
15         break
16     h = np.zeros((10000, 128)) # reset the model
17     x = np.zeros_like(h) # x is the input
18     inputs = [data[i][0] for i in range(param_lengths)]
19     targets = [data[i][1] for i in range(param_lengths)]
20
21     # sample from the model to get data
22     if n % 100 == 0:
23         sample_in = sample_inputs(inputs[:100], 100)
24         test = "...".join(x.char[i] for i in range(sample_in))
25         print "Test %d: %s" % (n, test)
26
27     # forward on hidden characters through the net and then predict
28     loss, much, much_np, many, many_p = sample_outputs(targets, sample_in)
29     search_loss = search_loss + loss * 100 # loss is 0.0001
30     if n % 1000 == 0:
31         print "loss: %f search_loss: %f n: %d, much: %f" % (loss, search_loss, n, much)
32
33     # perform parameter update with Adagrad
34     for param, param_np in zip(steps, much_np):
35         much, many, many_p = much + param_np, many + param_np, many_p + param_np
36
37     much_np = learning_rate * param
38     param_np = np.sqrt(much) # param_np = 0.01 * sqrt(much)
39
40     p = next_length * data_pointer
41     n = n + 1 # increment counter

```

```

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

```

min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(data)
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique' % (data_size, vocab_size)
12 char_to_ix = { ch: i for i, ch in enumerate(chars) }
13 ix_to_char = { i: ch for i, ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 b1 = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is list/array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     es, hs, ys, ps = [], [], [], []
34     h1 = hprev
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi = np.zeros((vocab_size, 1)) # encode in 1-of-K representation
39         xi[ix_to_char[inputs[t]]] = 1
40         h1 = np.tanh(np.dot(wih, xi) + np.dot(whh, h1) + b1) # hidden state
41         yi = np.dot(why, h1) + by # unnormalized log probabilities for next chars
42         pi = np.exp(yi) / np.sum(np.exp(yi)) # probabilities for next chars
43         loss += -np.log(pi)[targets[t]] # softmax (cross-entropy) loss
44     # backward pass: compute gradients going backward
45     dwh, dwhh, dwhy = np.zeros_like(whh), np.zeros_like(whh), np.zeros_like(why)
46     dbh, dbh1, dbh2 = np.zeros_like(bh), np.zeros_like(b1), np.zeros_like(b1)
47     dhnxt = np.zeros_like(h1)
48     for t in reversed(xrange(len(inputs))):
49         dy = np.zeros_like(yi)
50         dy[targets[t]] -= 1 # backprop into y
51         dbh1 += np.dot(dy, h1)
52         dy += dbh1
53         dh = np.dot(why, T, dy) + dhnxt # backprop into h
54         dhrw = (1 - h1**2) * h1**2 * dh # backprop through tanh nonlinearity
55         dwh += np.dot(dhrw, xi)
56         dwhh += np.dot(dhrw, h1)
57         dhnxt = np.dot(whh, T, dhrw)
58         dparam = np.zeros_like(dparam) # clip to mitigate exploding gradients
59         return loss, dwh, dwhh, dwhy, dbh, dbh1, dbh2, dy, h1, dhnxt
60
61 def sample(h, ix):
62     """
63     sample a sequence of integers from the model
64     h is memory state, ix is seed letter for first time step
65     """
66     x = np.zeros((vocab_size, 1))
67     ix = ix
68     ix = ix
69     for t in xrange(1):
70         xi = np.zeros((vocab_size, 1))
71         xi[ix_to_char[ix]] = 1
72         h = np.tanh(np.dot(wih, xi) + np.dot(whh, h) + b1)
73         yi = np.dot(why, h) + by
74         pi = np.exp(yi) / np.sum(np.exp(yi))
75         ix = np.random.choice(range(vocab_size), p=pi)
76         x = np.zeros((vocab_size, 1))
77         ix[ix_to_char[ix]] = 1
78         loss += -np.log(pi)
79     return loss
80
81 n, p = 0, 0
82 mWxh, mWhh, mWhy = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print '----\n %s \n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100     loss, dWxh, dWhh, dWhy, dbh, dbh1, dbh2, hprev = lossFun(inputs, targets, hprev)
101     smooth_loss = smooth_loss * 0.999 + loss * 0.001
102     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104     # perform parameter update with Adagrad
105     for param, dparam, mem in zip([wih, whh, why, bh, by],
106                                   [dWxh, dWhh, dWhy, dbh, dby],
107                                   [mWxh, mWhh, mWhy, mbh, mby]):
108         mem += dparam * dparam
109         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111     p += seq_length # move data pointer
112     n += 1 # iteration counter
```

Main loop

```
81 n, p = 0, 0
82 mWxh, mWhh, mWhy = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print '----\n %s \n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100     loss, dWxh, dWhh, dWhy, dbh, dbh1, dbh2, hprev = lossFun(inputs, targets, hprev)
101     smooth_loss = smooth_loss * 0.999 + loss * 0.001
102     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104     # perform parameter update with Adagrad
105     for param, dparam, mem in zip([wih, whh, why, bh, by],
106                                   [dWxh, dWhh, dWhy, dbh, dby],
107                                   [mWxh, mWhh, mWhy, mbh, mby]):
108         mem += dparam * dparam
109         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111     p += seq_length # move data pointer
112     n += 1 # iteration counter
```

min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 b1 = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is vec array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     es, hs, ys, ps = [], [], [], []
34     h1 = hprev
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi = np.zeros((vocab_size, 1)) # encode in 1-of-K representation
39         xi[ix_to_char[t]] = 1
40         h1 = np.tanh(np.dot(wih, xi) + np.dot(whh, h1[-1]) + bh) # hidden state
41         yi = np.dot(why, h1) + by # unnormalized log probabilities for next chars
42         ps = np.exp(yi) / np.sum(np.exp(yi)) # probabilities for next chars
43         loss += -np.log(ps[t][xi[t]]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backward
45     dwh, dbh, dby = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
46     dh1, dh2 = np.zeros_like(h1), np.zeros_like(h2)
47     dhw = np.zeros_like(h1)
48     for t in reversed(xrange(len(inputs))):
49         dy = np.zeros_like(yi)
50         dy[targets[t]] -= 1 # backprop into y
51         dbh += np.dot(dy, h1[t:T])
52         dy += dy
53         dh = np.dot(dw, dy) # dhw = backprop into h
54         dhw = (1 - h1[t] * h1[t]) * dh # dh = backprop through tanh nonlinearity
55         dbh += np.dot(dhw, wih[t:T])
56         dhw += np.dot(dhw, whh[t:T])
57         dh1 = np.dot(dh, whh)
58         dhw = np.dot(dhw, T, dhw)
59         for dparam in [dw, dwh, dbh, dby, dh1, dhw]:
60             # clip dparam to [-5, 5, out-dparam] to clip to mitigate exploding gradients
61             dparam = np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
62     return loss, dwh, dwhh, dwhy, dbh, dby, h1[len(inputs)-1]
63
64 def sample(h, ix):
65     """
66     sample a sequence of integers from the model
67     h is memory state, seed, ix is seed letter for first time step
68     """
69     x = np.zeros((vocab_size, 1))
70     ix = ix
71     ix = ix
72     for t in xrange(1):
73         xi = np.zeros((vocab_size, 1))
74         xi[ix] = 1
75         h = np.tanh(np.dot(wih, xi) + np.dot(whh, h) + bh)
76         yi = np.dot(why, h) + by
77         ps = np.exp(yi) / np.sum(np.exp(yi))
78         ix = np.random.choice(range(vocab_size), p=ps)
79         x = np.zeros((vocab_size, 1))
80         ix[ix] = 1
81         loss += -np.log(ps[ix])
82     return loss
83
84 n, p = 0, 0
85 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
86 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
87
88 while True:
89     # prepare inputs (we're sweeping from left to right in steps seq_length long)
90     if p+seq_length+1 >= len(data) or n == 0:
91         hprev = np.zeros((hidden_size,1)) # reset RNN memory
92         p = 0 # go from start of data
93         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
94         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
95
96         # sample from the model now and then
97         if n % 100 == 0:
98             sample_ix = sample(hprev, inputs[0], 200)
99             txt = ''.join(ix_to_char[ix] for ix in sample_ix)
100             print '----\n %s \n----' % (txt, )
101
102             # forward seq_length characters through the net and fetch gradient
103             loss, dwhx, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
104             smooth_loss = smooth_loss * 0.999 + loss * 0.001
105             if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
106
107         # perform parameter update with Adagrad
108         for param, dparam, mem in zip([wih, whh, why, bh, by],
109                                     [dwhx, dwhh, dwhy, dbh, dby],
110                                     [mbh, mby]):
111             mem += dparam * dparam
112             param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
113
114         p += seq_length # move data pointer
115         n += 1 # iteration counter
```

Main loop

```
81 n, p = 0, 0
82 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93         # sample from the model now and then
94         if n % 100 == 0:
95             sample_ix = sample(hprev, inputs[0], 200)
96             txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97             print '----\n %s \n----' % (txt, )
98
99         # forward seq_length characters through the net and fetch gradient
100         loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101         smooth_loss = smooth_loss * 0.999 + loss * 0.001
102         if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104         # perform parameter update with Adagrad
105         for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
106                                     [dWxh, dWhh, dWhy, dbh, dby],
107                                     [mWxh, mWhh, mWhy, mbh, mby]):
108             mem += dparam * dparam
109             param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111         p += seq_length # move data pointer
112         n += 1 # iteration counter
```

min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wnh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bnh = np.zeros(hidden_size, 1) # hidden bias
25 by = np.zeros(vocab_size, 1) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is vec array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     es, hs, ys, ps = [], [], [], []
34     h = hprev
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi = np.zeros(vocab_size, 1) # encode in 1-of-K representation
39         xi[ix_to_char[inputs[t]]] = 1
40         h = np.tanh(np.dot(hnh, xi) + np.dot(whh, h[-1:] + bh) + hidden_state)
41         yi = np.dot(h, why) + by # unnormalized log probabilities for next chars
42         ps = np.exp(yi) / np.sum(np.exp(yi)) # probabilities for next chars
43         loss += -np.log(ps[targets[t]]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backward
45     dnh, dbh, dhy = np.zeros_like(hnh), np.zeros_like(whh), np.zeros_like(why)
46     dh = np.zeros_like(hh), np.zeros_like(bh)
47     dhnxt = np.zeros_like(hn)
48     for t in reversed(xrange(len(inputs))):
49         dy = np.zeros(1)
50         dy[targets[t]] -= 1 # backprop into y
51         dhy += np.dot(dy, h[-1:t])
52         dy = dy
53         dh = np.dot(dhy, dy) + dhnxt # backprop into h
54         dhrnw = (1 - h[-1:t]*h[-1:t]) * dh # backprop through tanh nonlinearity
55         dhn = np.dot(dhrnw, wnh)
56         dhnxt = np.dot(dhrnw, whh)
57         dhnxt = np.dot(dhnxt, whh)
58         for dparam in [dnh, dbh, dhy, dh]:
59             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
60     return loss, dnh, dbh, dhy, dh, h[-1:]
61
62 def sample(h, ix):
63     """
64     sample a sequence of integers from the model
65     h is memory state, seed, ix is seed letter for first time step
66     """
67     x = np.zeros(vocab_size, 1)
68     ix_to_ix = ix
69     ixes = []
70     for t in xrange(1):
71         xi = np.zeros(vocab_size, 1)
72         xi[ix_to_ix[ix]] = 1
73         y = np.dot(h, why) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p, replace=True)
76         ixes.append(ix)
77     return ixes
78
79 # main loop
80 n, p = 0, 0
81 mw, wh, w, whh, why, bh, by, hprev = lossFun(inputs, targets, hprev)
82 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
83 while True:
84     # prepare inputs (we're sweeping from left to right in steps seq_length long)
85     if p+seq_length+1 >= len(data) or n == 0:
86         hprev = np.zeros((hidden_size,1)) # reset RNN memory
87         p = 0 # go from start of data
88         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
89         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
90
91     # sample from the model now and then
92     if n % 100 == 0:
93         sample_ix = sample(hprev, inputs[0], 200)
94         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
95         print '----\n %s \n----' % (txt, )
96
97     # forward seq_length characters through the net and fetch gradient
98     loss, dwnh, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
99     smooth_loss = smooth_loss * 0.999 + loss * 0.001
100     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
101
102     # perform parameter update with Adagrad
103     for param, dparam, mem in zip([wnh, whh, why, bh, by],
104                                   [dwnh, dwhh, dwhy, dbh, dby],
105                                   [mw, wh, w, whh, why, bh, by]):
106         mem += dparam * dparam
107         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
108
109     p += seq_length # move data pointer
110     n += 1 # iteration counter
```

Main loop

```
81 n, p = 0, 0
82 mw, wh, w, whh, why, bh, by, hprev = np.zeros_like(w), np.zeros_like(whh), np.zeros_like(why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print '----\n %s \n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100     loss, dwnh, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101     smooth_loss = smooth_loss * 0.999 + loss * 0.001
102     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104     # perform parameter update with Adagrad
105     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
106                                   [dWxh, dWhh, dWhy, dbh, dby],
107                                   [mWxh, mWhh, mWhy, mbh, mby]):
108         mem += dparam * dparam
109         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111     p += seq_length # move data pointer
112     n += 1 # iteration counter
```

[min-char-rnn.py](#) gist

```

1 #
2 Minimal character-level vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 MIT License
4
5 import numpy as np
6
7 # data loader
8 def load_data(char_path, txt_path):
9     """data_loader.py should be simple plain text file
10     chars = list(set(data_loader.txt_path.read().split()))
11     vocab_size, vocab_size = len(chars), len(chars)
12     print "num. unique chars: %d" % vocab_size
13     char_to_ix = { char: i for i, char in enumerate(chars) }
14     ix_to_char = { i: char for i, char in enumerate(chars) }
15
16 # hyperparameters
17 hidden_size = 100 # size of hidden layer of neurons
18 seq_length = 25 # number of steps to unroll the RNN for
19 learning_rate = 1e-1
20
21 # model parameters
22 w_h = np.random.randn(hidden_size, hidden_size)*0.01 # weights to hidden
23 w_x = np.random.randn(hidden_size, hidden_size)*0.01 # weights to hidden
24 b_h = np.random.randn(hidden_size)*0.01 # biases to hidden
25 b_x = np.zeros((hidden_size, 1)) # hidden bias
26 by = np.zeros((vocab_size, 1)) # output bias
27 by = np.zeros((vocab_size, 1)) # output bias

```

[illegible]

```

def sample(x, seq_len, n):
    """
    sample a sequence of integers from the model
    h is memory state, seq_idx is seed letter for first time step
    """
    w = np.zeros((vocab_size, 1))
    y_treated = 0
    lens = 1
    for t in range(n):
        u = np.random.choice(wax, x=(seq_idx, h), y= h)
        y = np.empty(h, dtype= np.int64)
        p = np.empty(h, dtype= np.float64)
        u = np.random.choice(vocab_size, p=p.ravel())
        w = np.zeros((vocab_size, 1))
        y_treated += u
        lens += 1
    lens.append(lens)
    return lens

N, n, p, x = 1000, 100, 0.001, 0
m0, m1, m2, m3, m4 = np.zeros_like(wax), np.zeros_like(wah), np.zeros_like(wby)
seq_idx = np.zeros_like(wax)
seq_lens = np.zeros_like(wax)
seq_lens[0] = np.log10((vocab_size-1)*seq_lens) # seq_len at iteration 0
# prepare inputs (x) and targets (y) from left to right in seq_lengths long
for i in range(1, seq_lengths):
    h = np.zeros_like(h0)
    hnew = np.zeros_like(h0den_size, x) # reset new memory
    inputs = [hnew.to_ix(x)] for x in data[i-p:seq_lengths-i]
    targets = [hnew.to_ix(y)] for y in data[i-p:seq_lengths-i]
    # sample from the model now and then
    for j in range(1, N):
        m = m0 + m1 * j + m2 * j**2 + m3 * j**3 + m4 * j**4
        print "x = %d\nseq_idx = %d\nseq_lens = %d\n" % (x, seq_idx, seq_lens[i])
        # Forward seq_lengths characters through the net and fetch gradient
        loss, seq_idx, seq_lens, dby, hnew = forward(inputs, targets, hnew)
        loss += m * seq_lens[i]
        loss *= 1.0001
        if N - m == 0:
            print "iter %d, loss %d, m %d, (m, seq_lengths) = " % (j, loss, m, (m, seq_lengths)) # print progress
    # perform parameter update with Adagrad
    for param, dparam, new in zip(h0, dby, h0, hnew, yb, yb, yb, yb):
        new += dparam * dparam
        param = - learning_rate * dparam / np.sqrt(new + 1e-8) # sqrtgrad update
    p = seq_lengths # move data pointer
    x = iteration_count % vocab_size

```

Loss function

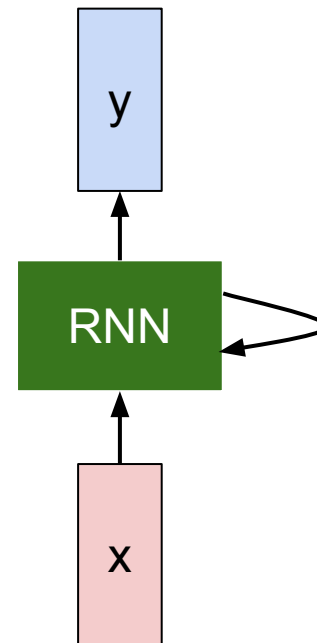
- forward pass (compute loss)
- backward pass (compute param gradient)

```

27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backwards
45     dwxh, dwhh, dwhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(why)
46     dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47     dhnext = np.zeros_like(hs[0])
48     for t in reversed(xrange(len(inputs))):
49         dy = np.copy(ps[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dwhy += np.dot(dy, hs[t].T)
52         dby += dy
53         dh = np.dot(why.T, dy) + dhnext # backprop into h
54         dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55         dbh += dhraw
56         dwxh += np.dot(dhraw, xs[t].T)
57         dwhh += np.dot(dhraw, hs[t-1].T)
58         dhnext = np.dot(Whh.T, dhraw)
59     for dparam in [dwxh, dwhh, dwhy, dbh, dby]:
60         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61     return loss, dwxh, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]

```

«p class="clear» > Products: Laser-Printers: The fundamental everyday requirement for mono and colour laser printing throughout today's office is perfectly met with the extensive Epson laser printer range. The latest AcuLaser printer range offers users exceptionally Epson AcuLaser C1900 Networked compact colour laser printer for professional enterprises. Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more with the Epson-AcuLaser C1900. Epson brings both colour and monochrome laser printing together at a black and white price. more Where to Buy Support Epson AcuLaser C2000 The fastest colour laser printer in its class. The perfect printer for small businesses and work groups, the Epson-AcuLaser C2000 prints high volumes in black and white and vibrant colour, at high speed and with low running costs. more Where to Buy Support Epson AcuLaser C2500 Prints high quality resolution: 2400dpi R7. Large paper capacity: 600 sheets, expandable up to 1,000 sheets. Compatible Windows and Mac. High speed USB and EpsonNet 10/100 Base-TX Ethernet interfaces as standard. * Epson AcuLaser Resolution Improvement Technology. *EpsonNet 10/100 Base-TX Ethernet standard with Epson AcuLaser C2000 model only. AcuLaser C2000 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing as standard AcuLaser C2000 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing, 10/100Base-TX Ethernet Interface Networked compact colour laser printer for professional enterprises. Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more with the Epson-AcuLaser C1900. Epson brings both colour and monochrome laser printing together at a black and white price. Key features cost effective mono printing for day to day business needs and vivid versatile colour when required. Search Epson UK. Epson AcuLaser C200 Outstanding professional colour printing for business. Add colour to your business with the Epson AcuLaser C600 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high performance black and white production. more Where to Buy Support As cost efficient to run as a mono-only user printer. Paper capacity of 700 sheets from two media sources. Easy to operate with advanced print driver. Memory expandable from 32MB to 1024MB. Pre-configured models available with Wireless 802.11b, Adobe® PostScript® 3 Level 3™ and two-sided printing. The AcuLaser C1900 is available in 3 configurations: - AcuLaser C1900i with 32MB, 200 Sheet MP Tray, 10/100Base-TX Networking - AcuLaser C1900 with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100Base-TX Networking Support Epson AcuLaser C4100 High performance colour lasers for all your business printing needs. The Epson AcuLaser C4100 provides businesses with a high performance colour and monochrome printing solution. It adds crucial colour to your business, while producing high quality monochrome output at lower costs than many monochrome-only printers, and is just as easy to operate. So now there's no reason to buy two printers, because perfect monochrome and colour solutions are available in one. more Where to Buy Support Epson AcuLaser C600 Professional high performance A3W colour laser printer. Epson AcuLaser C600 is the perfect professional printing solution for users who require exceptional quality colour and mono output on a range of media formats from C5 up to A3W in size. The Epson AcuLaser C600 is able to achieve superb print quality by utilising a combination of Epson's exclusive AcuLaser Colour Laser Technologies. more Where to Buy Support AcuLaser C1900PS with Adobe® PostScript® 3™, 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100Base-TX Networking - AcuLaser C1900i with Duplex unit (two sided printing) 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100Base-TX Networking - AcuLaser C1900 Wi-Fi with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, Wireless Networking facility. Add colour to your business with the Epson AcuLaser C800 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high. Support Epson AcuLaser C400 High performance colour laser. The Epson AcuLaser C400 provides businesses with high performance colour and monochrome printing solutions. more Where to Buy Support Epson AcuLaser C8100 High speed A3 colour laser printer. Why have separate black and white and colour printers when you can have the Epson AcuLaser C8100? Epson has taken the lead in laser technology to deliver a complete high-performance solution for all your colour and mono printing needs. Support EPL-6200L High performance A4 mono laser professional printers. The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more Where to Buy Support EPL-6200 High performance A4 mono laser professional printers. The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more performance black and white production. For the first time, you can now bring the power of high quality colour to your documents without suffering the high costs or low speeds traditionally associated with colour



Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no! it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.

at first:

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

↓
train more

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

↓
train more

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.

↓
train more

"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.

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nudes begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

VIOLA:

Why, Salisbury must find his flesh and thought
That which I am not apt, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:

O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

Linux kernel source tree

Repository statistics: 520,037 commits, 1 branch, 420 releases, 5,039 contributors.

branch: master - linux / +

Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux

torvalds authored 9 hours ago latest commit 4b1706927d

Documentation	Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending	6 days ago
arch	Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l...	a day ago
block	block: discard bdi_unregister() in favour of bdi_destroy()	9 days ago
crypto	Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6	10 days ago
drivers	Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux	9 hours ago
firmware	firmware/ihex2fw.c: restore missing default in switch statement	2 months ago
fs	vfs: read file_handle only once in handle_to_path	4 days ago
include	Merge branch 'perl-urgent-for-linus' of git://git.kernel.org/pub/scm/...	a day ago
init	init: fix regression by supporting devices with major:minor:offset fo...	a month ago
io	Merge branch 'for-linus' of git://git.kernel.org/pub/scm/linux/...	a month ago

Code

Pull requests 74

Pulse

Graphs

HTTPS clone URL

https://github.com/torvalds/linux

You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).

Clone in Desktop

Download ZIP

```

static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << 1))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff) & 0x0000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}

```

Generated C code

```
/*  
 * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.  
 *  
 * This program is free software; you can redistribute it and/or modify it  
 * under the terms of the GNU General Public License version 2 as published by  
 * the Free Software Foundation.  
 *  
 * This program is distributed in the hope that it will be useful,  
 * but WITHOUT ANY WARRANTY; without even the implied warranty of  
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
 *  
 * GNU General Public License for more details.  
 *  
 * You should have received a copy of the GNU General Public License  
 * along with this program; if not, write to the Free Software Foundation,  
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.  
 */  
  
#include <linux/kexec.h>  
#include <linux/errno.h>  
#include <linux/io.h>  
#include <linux/platform_device.h>  
#include <linux/multi.h>  
#include <linux/ckevent.h>  
  
#include <asm/io.h>  
#include <asm/prom.h>  
#include <asm/e820.h>  
#include <asm/system_info.h>  
#include <asm/setew.h>  
#include <asm/pgproto.h>
```

```

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)      (func)

#define SWAP_ALLOCATE(nr)      (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0)); \
    if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
    PUT_PARAM_RAID(2, sel) = get_state_state();
    set_pid_sum((unsigned long)state, current_state_str(),
        (unsigned long)-1->lr_full; low;
}

```

Searching for interpretable cells

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* Of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
     */
}
```

[Visualizing and Understanding Recurrent Networks, Andrej Karpathy, Justin Johnson*, Li Fei-Fei]*

Searching for interpretable cells

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

quote detection cell

Searching for interpretable cells

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

line length tracking cell

Searching for interpretable cells

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
                           siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!(current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}
```

if statement cell

Searching for interpretable cells

```
/* Duplicate LSM field information. The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                       struct audit_field *sf)
{
    int ret = 0;
    char *lsm_str;
    /* our own copy of lsm_str */
    lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
    if (unlikely(!lsm_str))
        return -ENOMEM;
    df->lsm_str = lsm_str;
    /* our own (refreshed) copy of lsm_rule */
    ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                   (void **)&df->lsm_rule);
    /* Keep currently invalid fields around in case they
     * become valid after a policy reload. */
    if (ret == -EINVAL) {
        pr_warn("audit rule for LSM '%s' is invalid\n",
                df->lsm_str);
        ret = 0;
    }
    return ret;
}
```

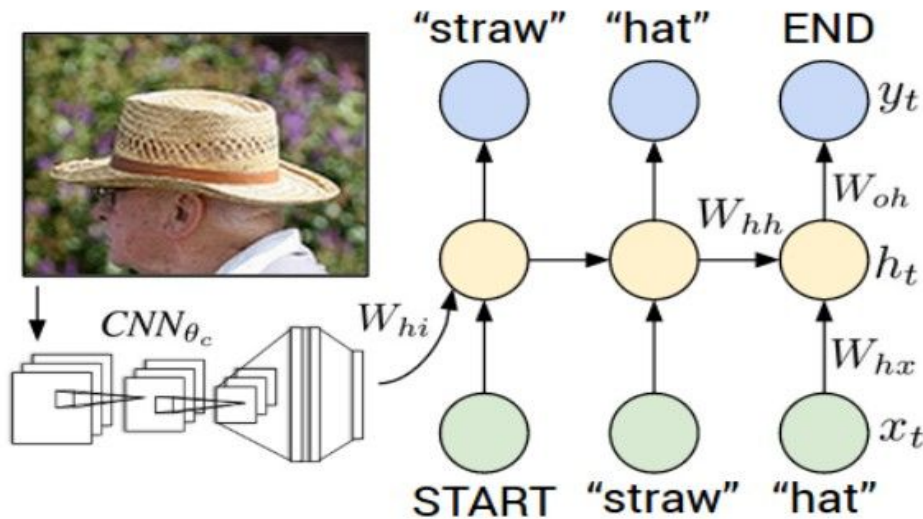
quote/comment cell

Searching for interpretable cells

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
    int i;
    if (classes[class]) {
        for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
            if (mask[i] & classes[class][i])
                return 0;
    }
    return 1;
}
```

code depth cell

Image Captioning



Explain Images with Multimodal Recurrent Neural Networks, Mao et al.

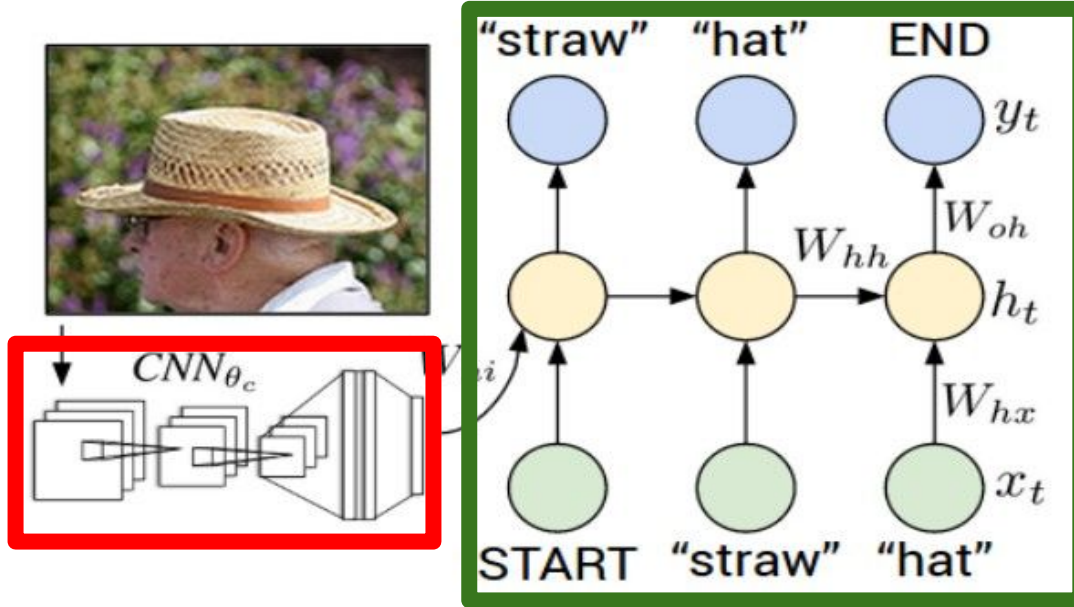
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

Show and Tell: A Neural Image Caption Generator, Vinyals et al.

Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.

Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Recurrent Neural Network



Convolutional Neural Network



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

x0
<STA
RT>

<START>

image

test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

V

y0

h0

x0

<STA

RT>

<START>

Wih

before:

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

now:

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

y0

h0

x0
<START
RT>

straw

sample!

<START>

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

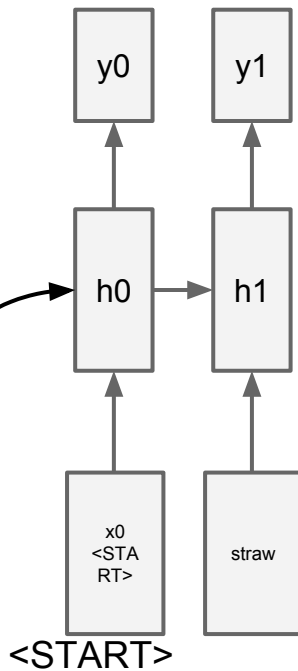
maxpool

FC-4096

FC-4096



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

y0

y1

h0

h1

x0
<START
RT>

straw

hat

sample!

<START>

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

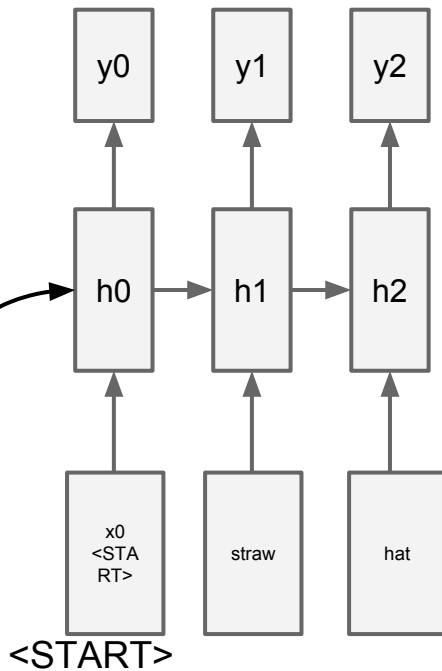
maxpool

FC-4096

FC-4096



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

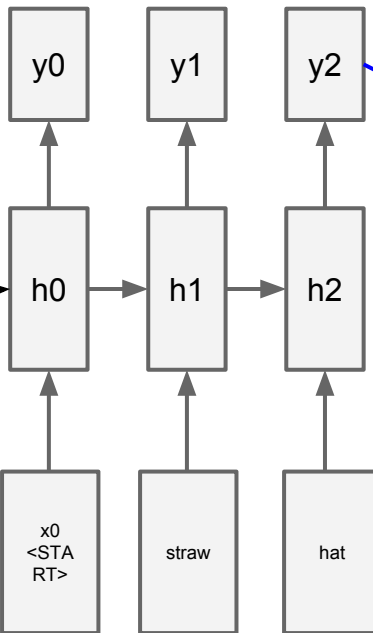
maxpool

FC-4096

FC-4096



test image



sample
<END> token
=> finish.

<START>

Image Sentence Datasets

a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.



Microsoft COCO

[Tsung-Yi Lin et al. 2014]

mscoco.org

currently:

~120K images

~5 sentences each



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



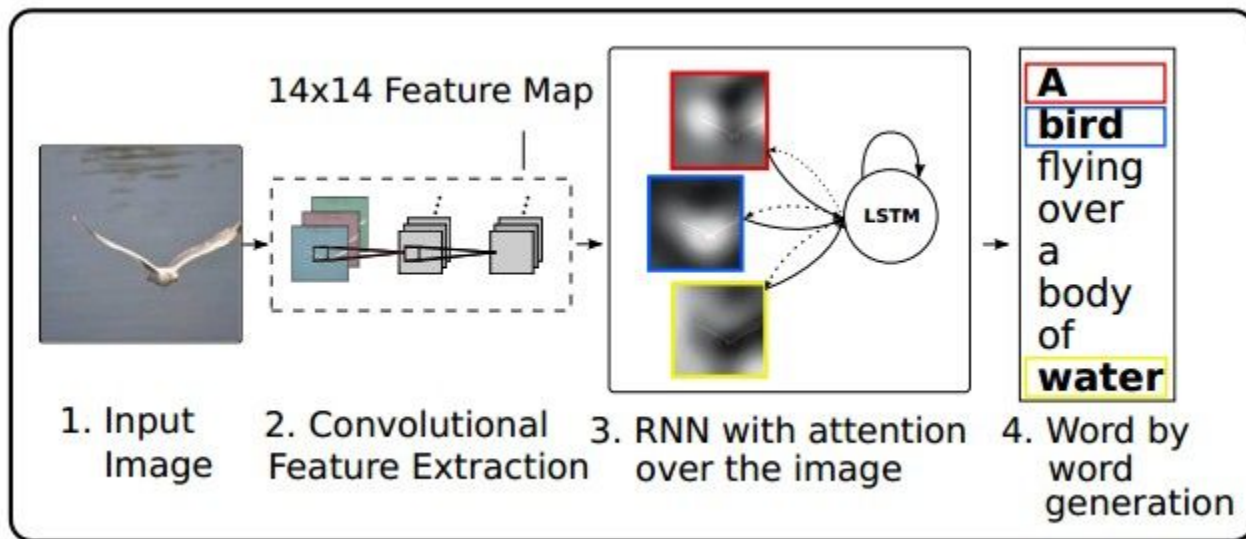
"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."

Preview of fancier architectures

RNN attends spatially to different parts of images while generating each word of the sentence:



Show Attend and Tell, Xu et al., 2015

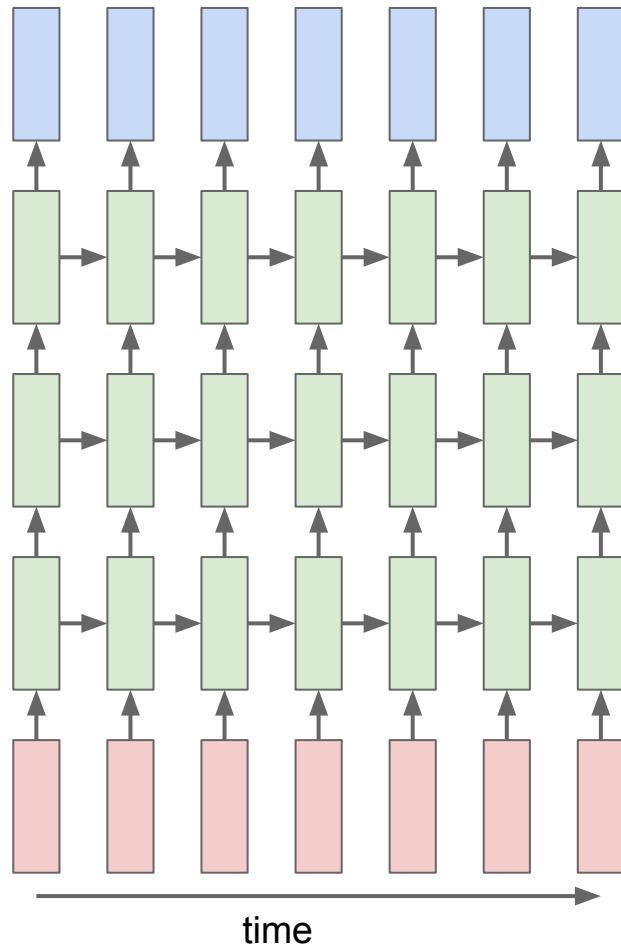
RNN:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$ $W^l [n \times 2n]$



depth



RNN:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$

$W^l [n \times 2n]$

LSTM:

$W^l [4n \times 2n]$

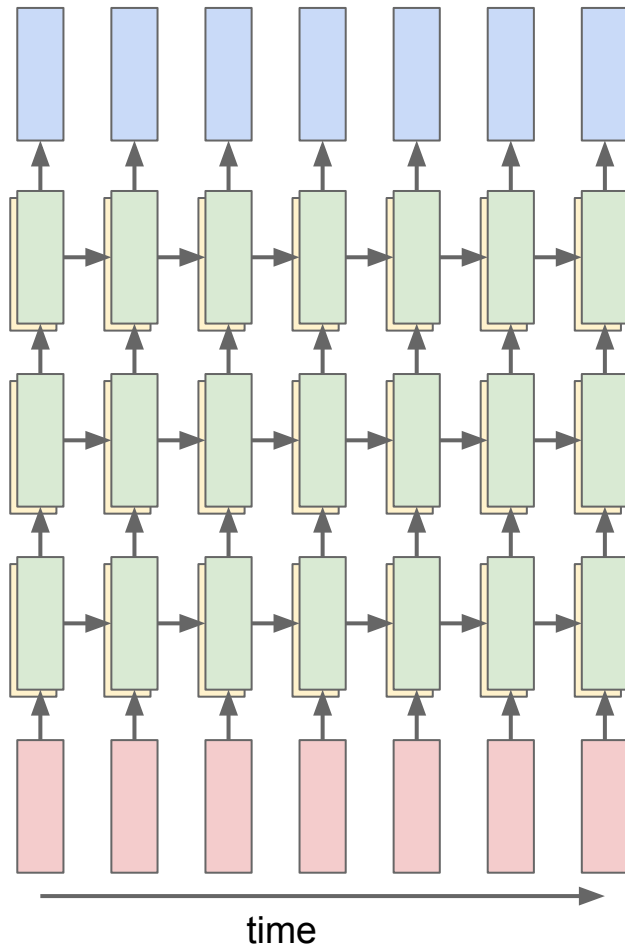
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

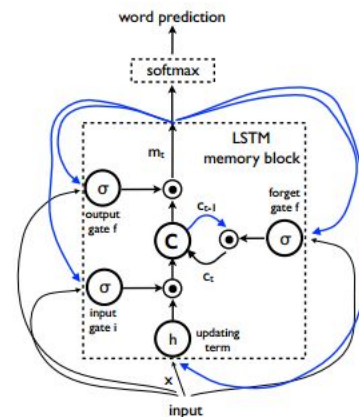
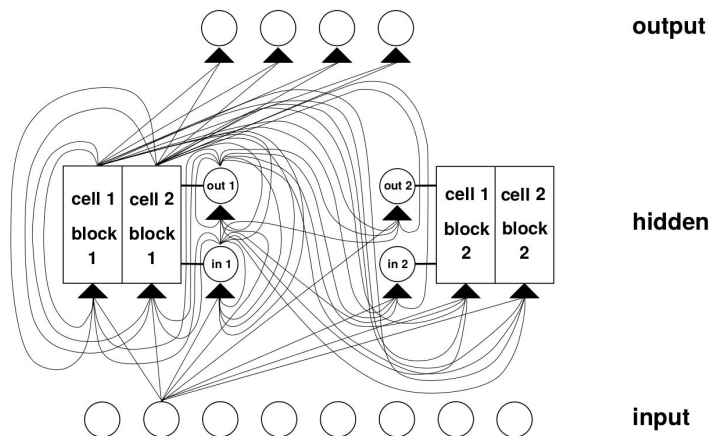
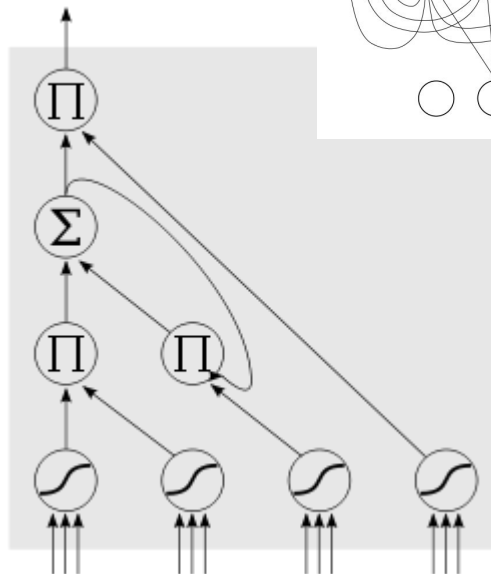
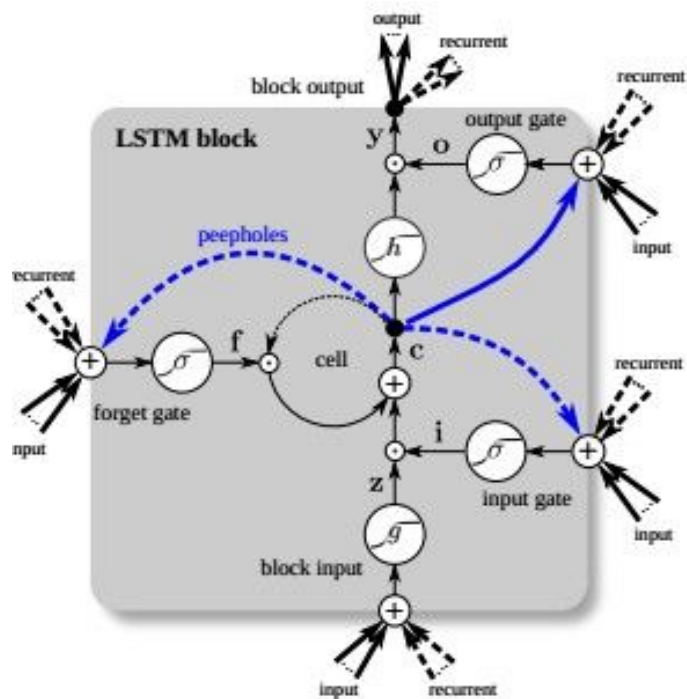
$$h_t^l = o \odot \tanh(c_t^l)$$



depth

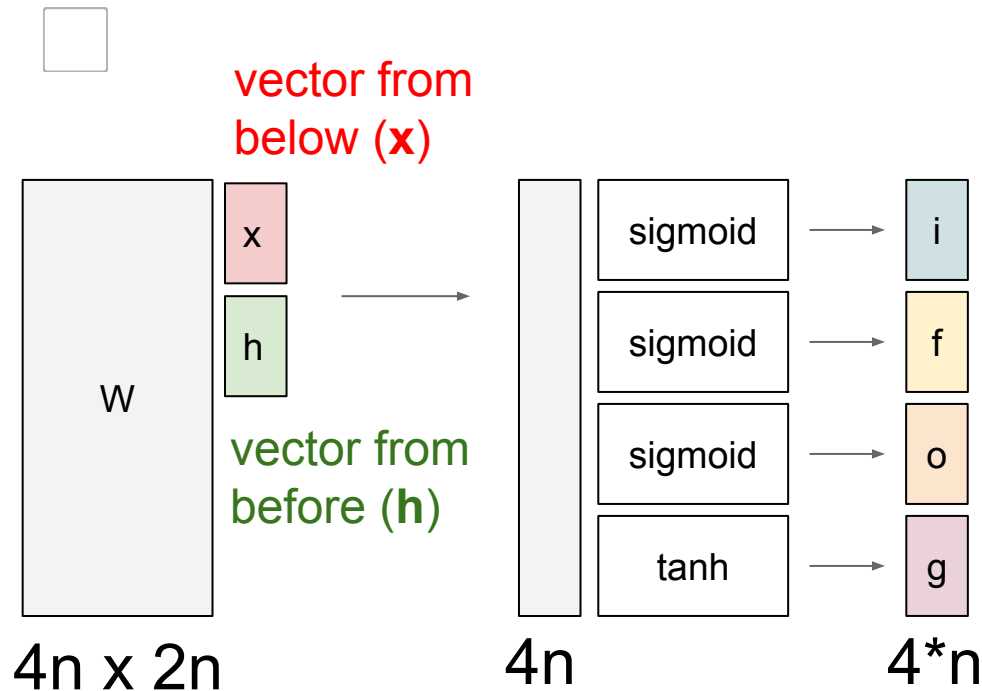


LSTM



Long Short Term Memory (LSTM)

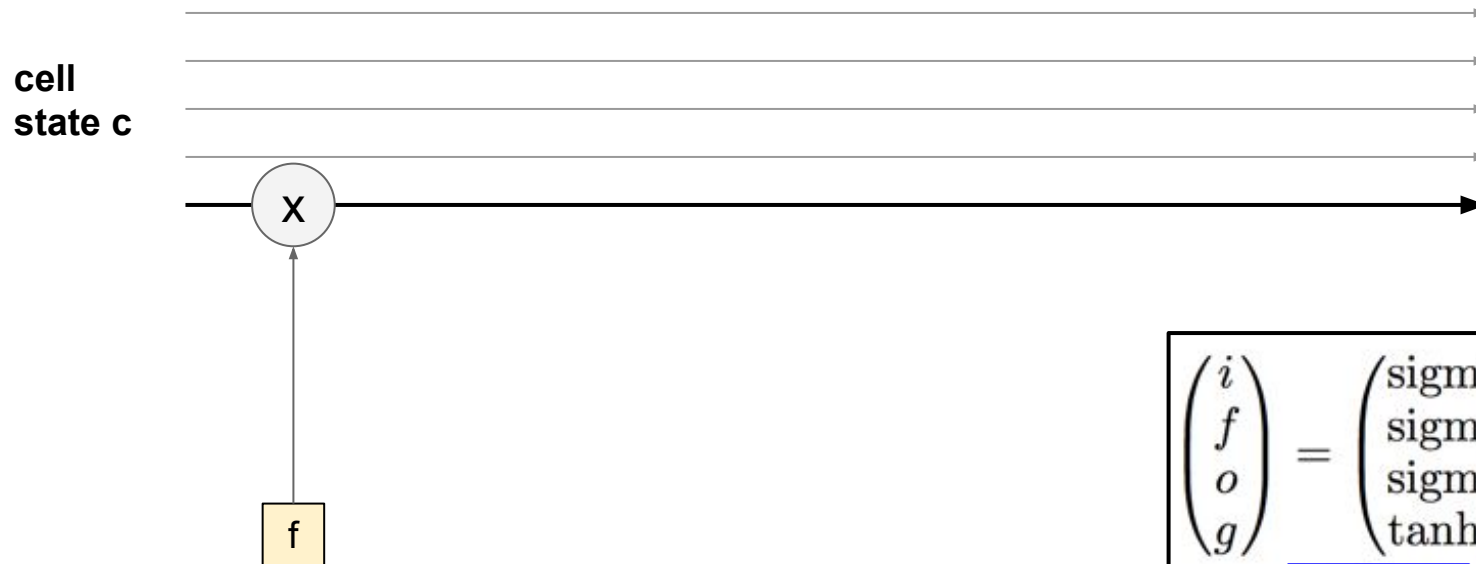
[Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

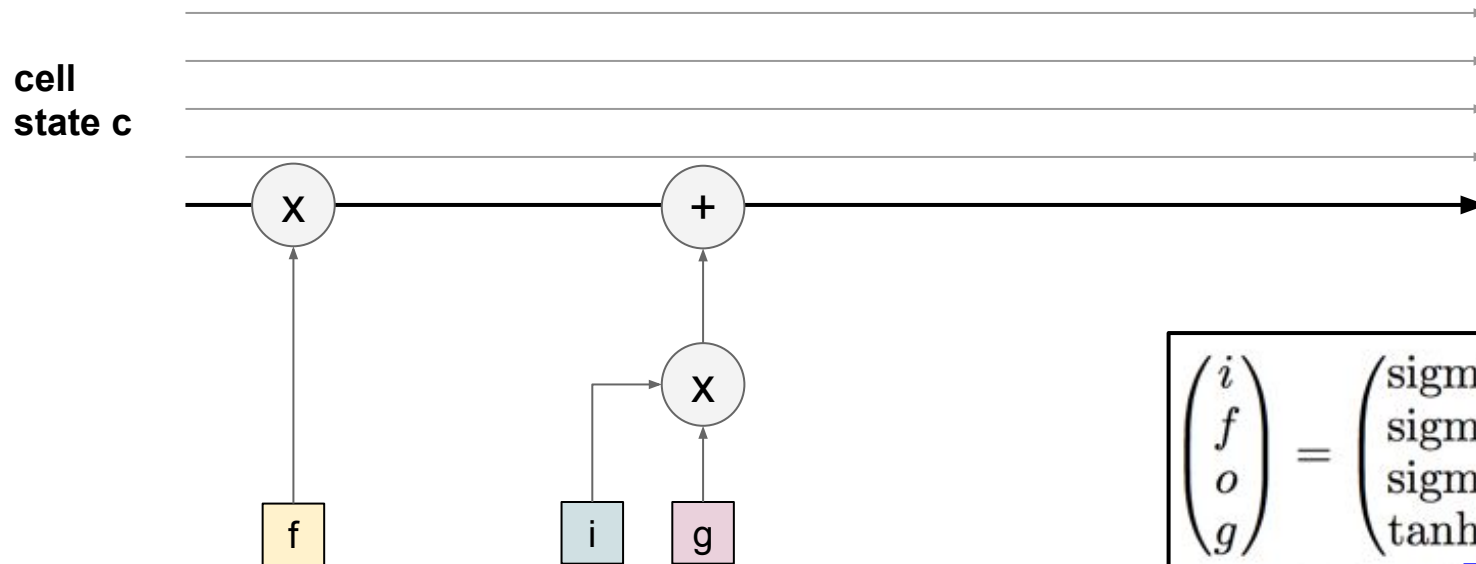
[Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

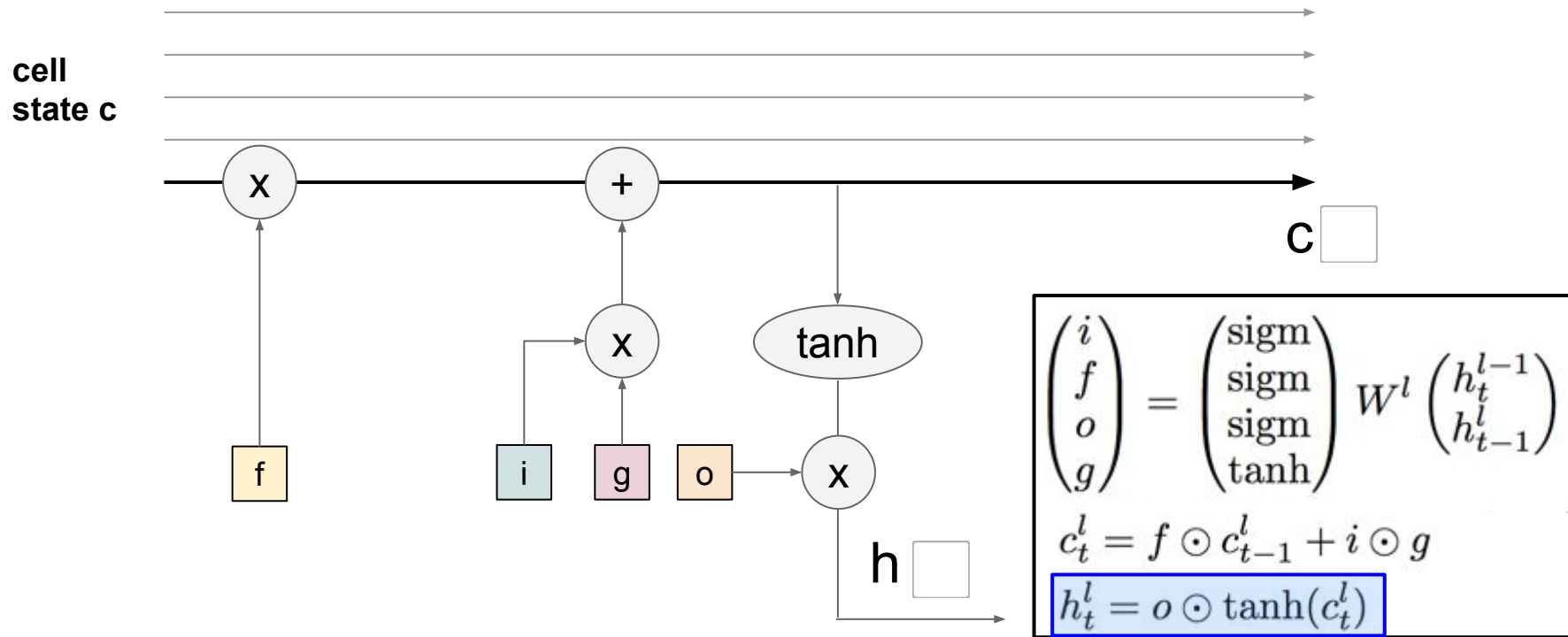
[Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

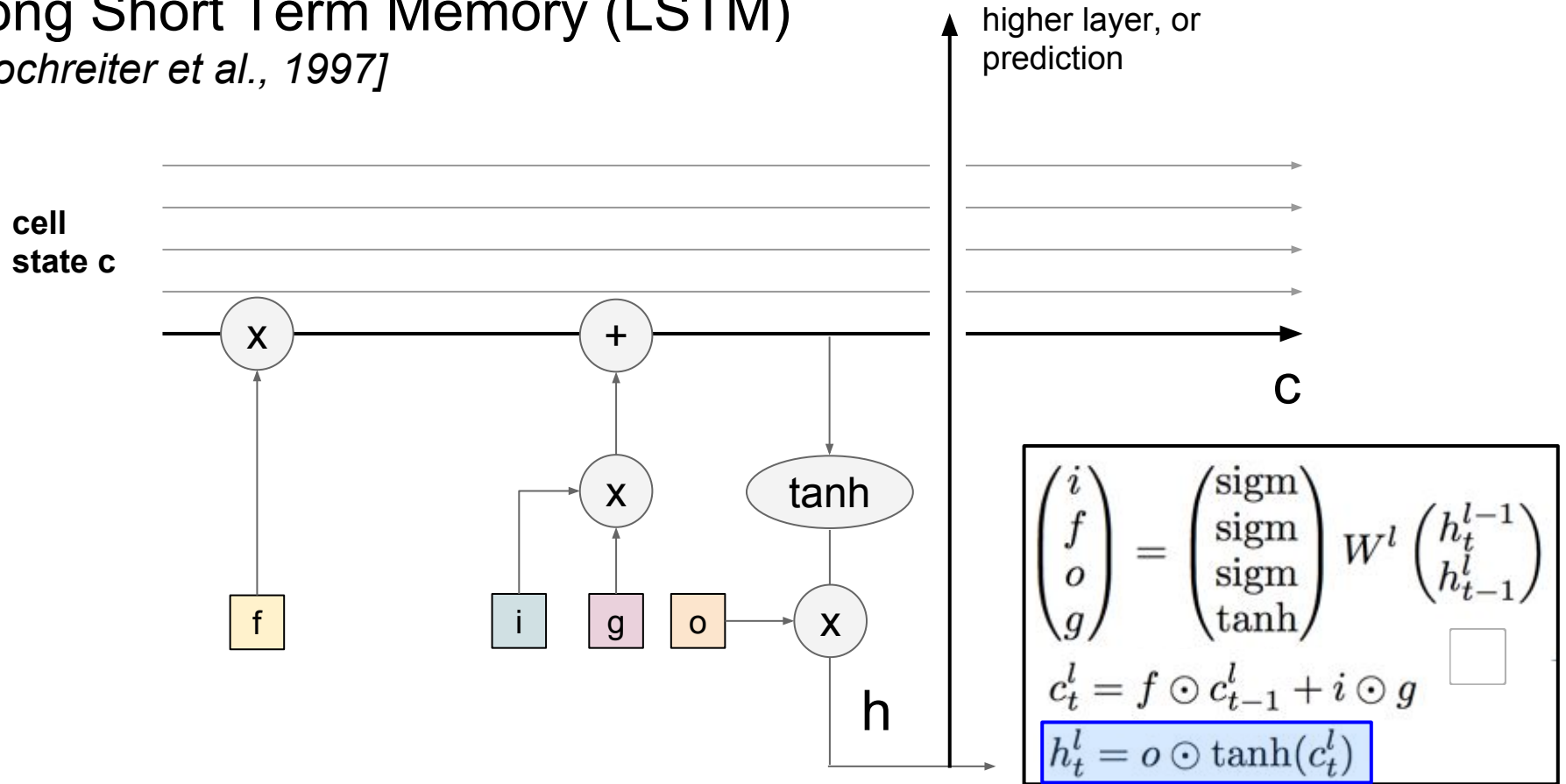
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

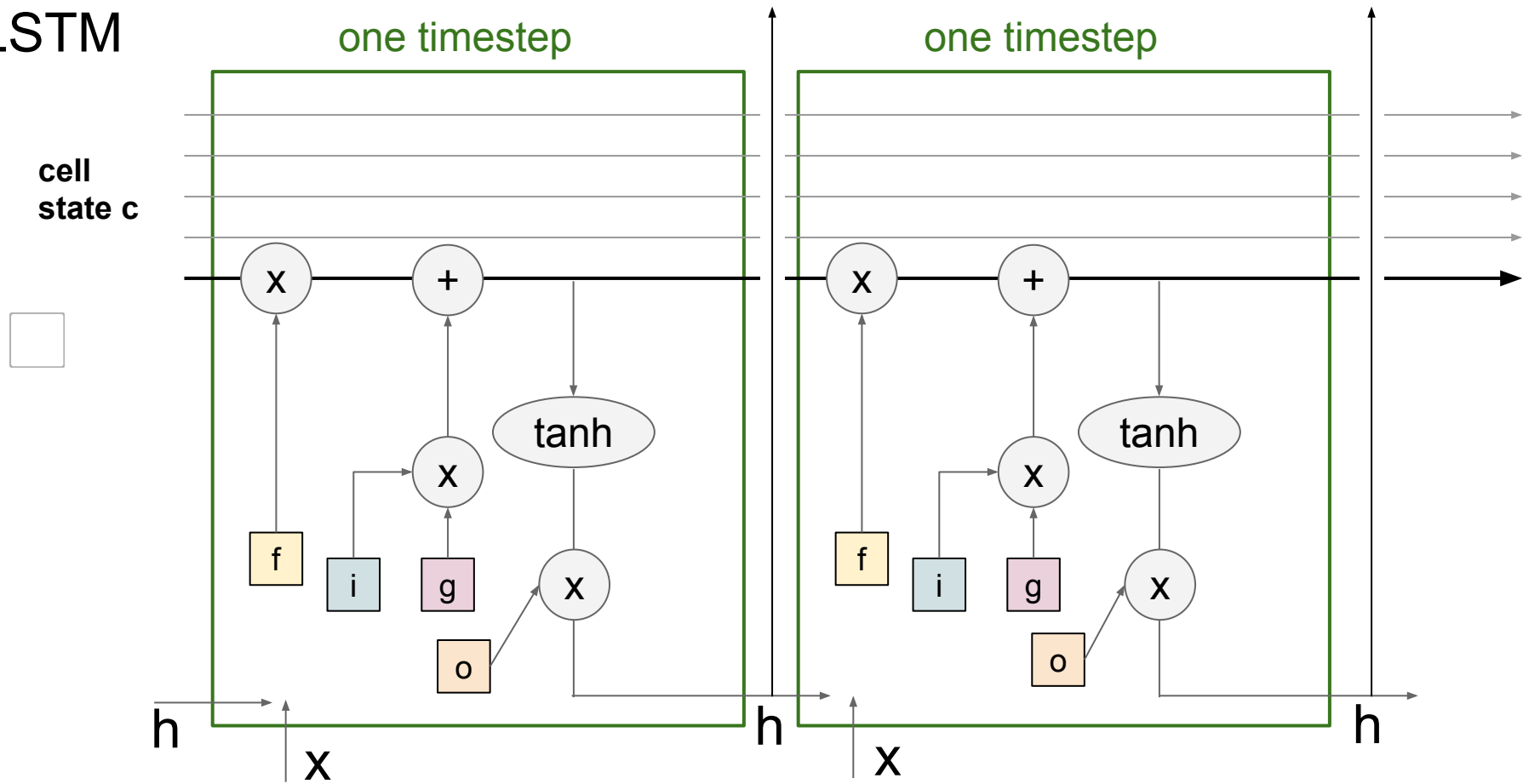


Long Short Term Memory (LSTM)

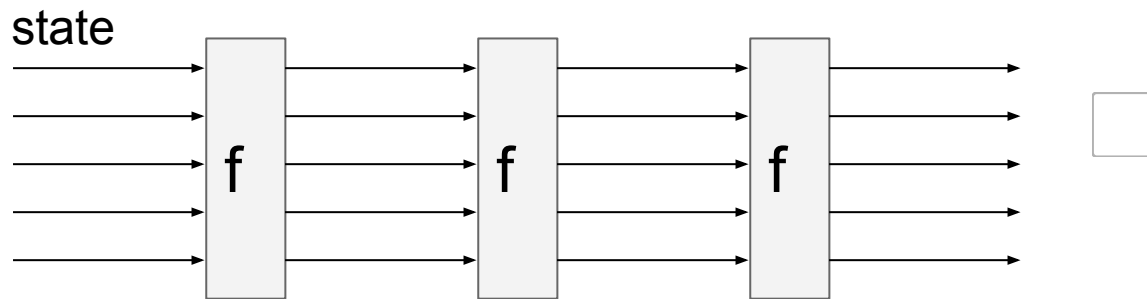
[Hochreiter et al., 1997]



LSTM

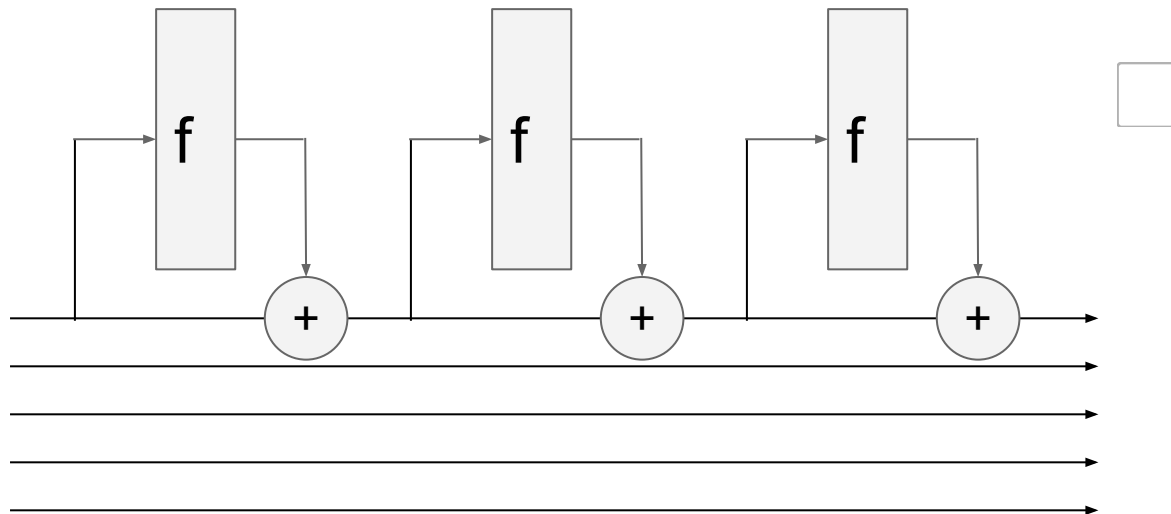


RNN



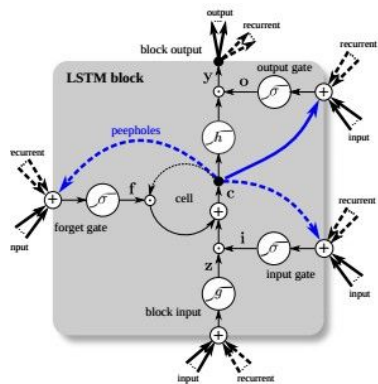
LSTM

(ignoring
forget gates)



LSTM variants and friends

[An Empirical Exploration of Recurrent Network Architectures, Jozefowicz et al., 2015]



[LSTM: A Search Space Odyssey, Greff et al., 2015]

GRU [Learning phrase representations using rnn encoder-decoder for statistical machine translation, Cho et al. 2014]

$$\begin{aligned} r_t &= \text{sigm}(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\ z_t &= \text{sigm}(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\ \tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\ h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \end{aligned}$$

MUT1:

$$\begin{aligned} z &= \text{sigm}(W_{xz}x_t + b_z) \\ r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + \tanh(x_t) + b_h) \odot z \\ &\quad + h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \text{sigm}(W_{xz}x_t + W_{hz}h_t + b_z) \\ r &= \text{sigm}(x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\ &\quad + h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \text{sigm}(W_{xz}x_t + W_{hz} \tanh(h_t) + b_z) \\ r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\ &\quad + h_t \odot (1 - z) \end{aligned}$$

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed.