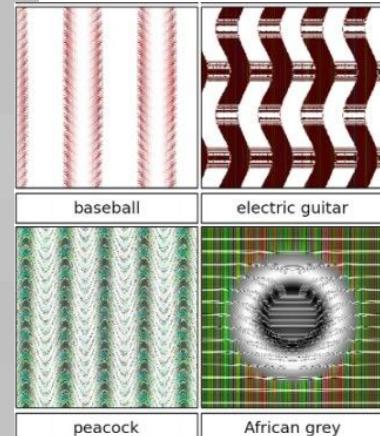
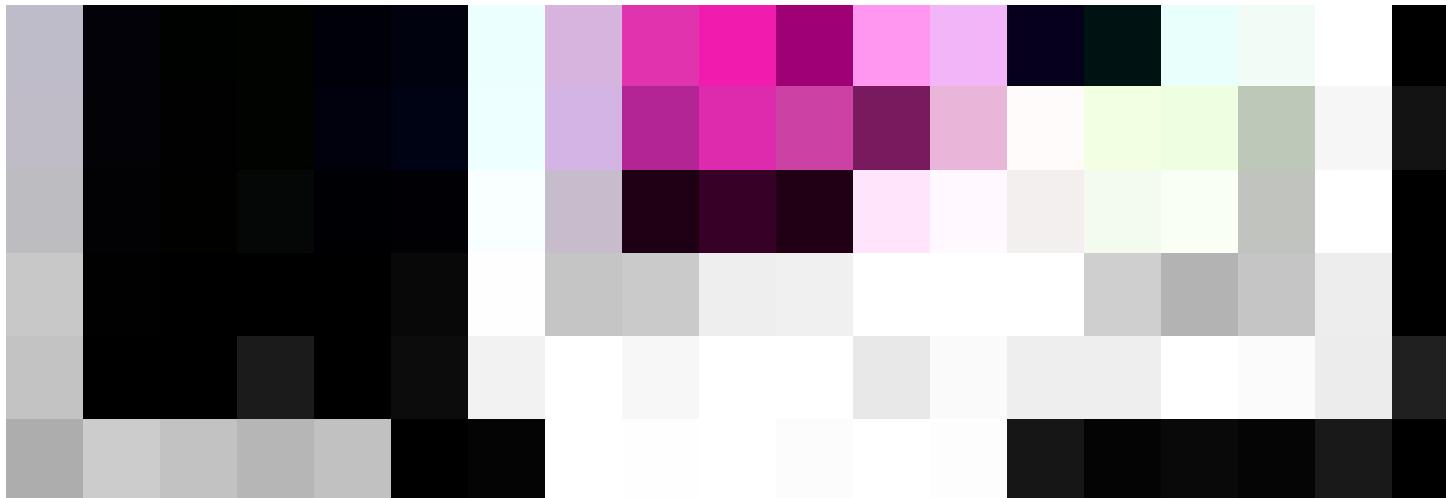
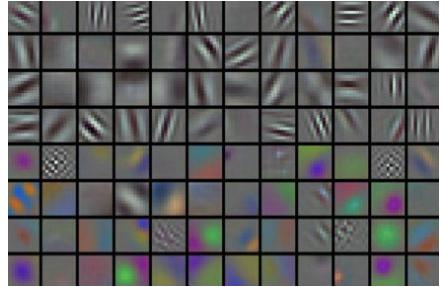


Lecture 10:

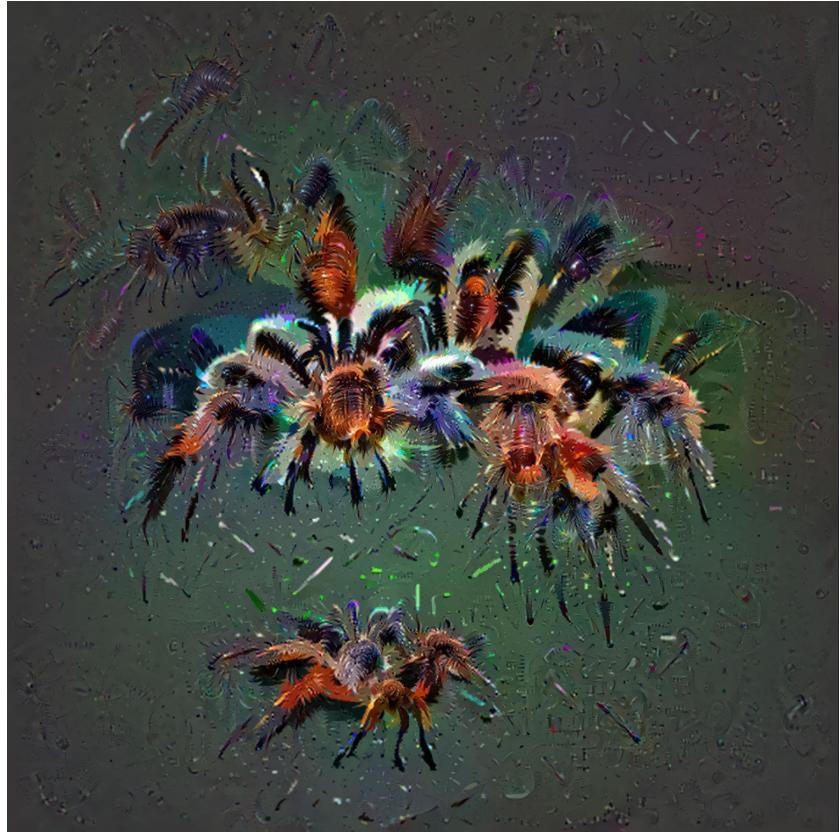
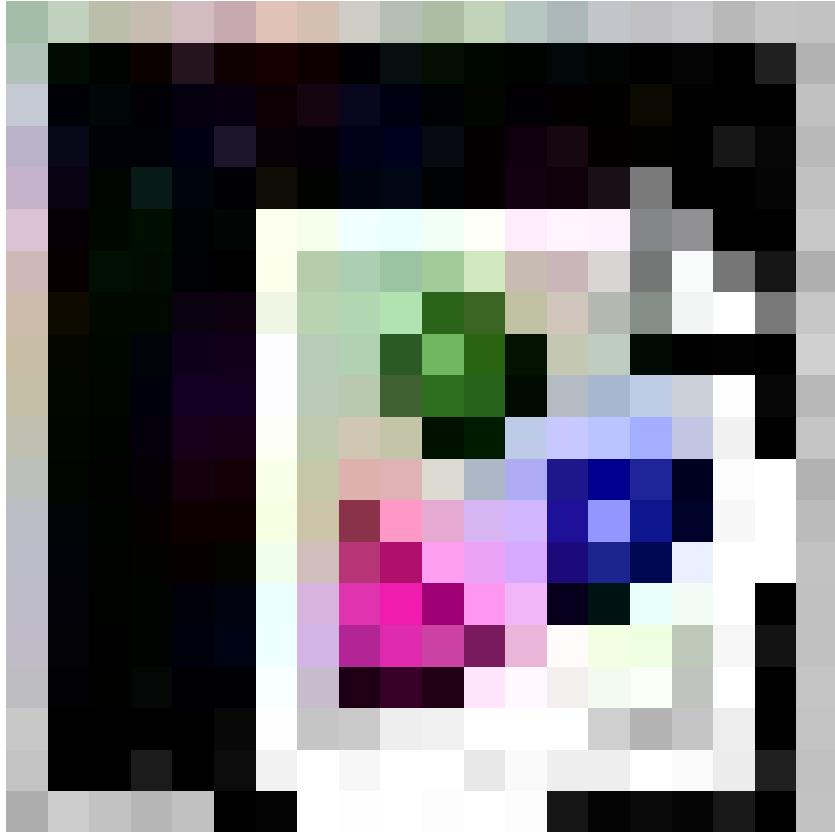
Recurrent Neural Networks

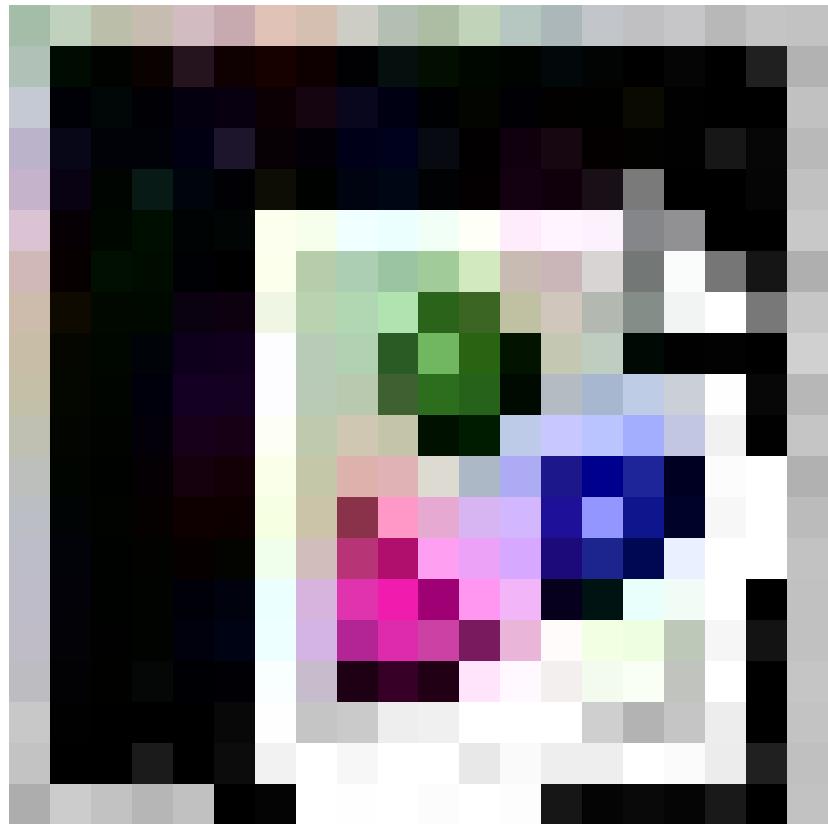
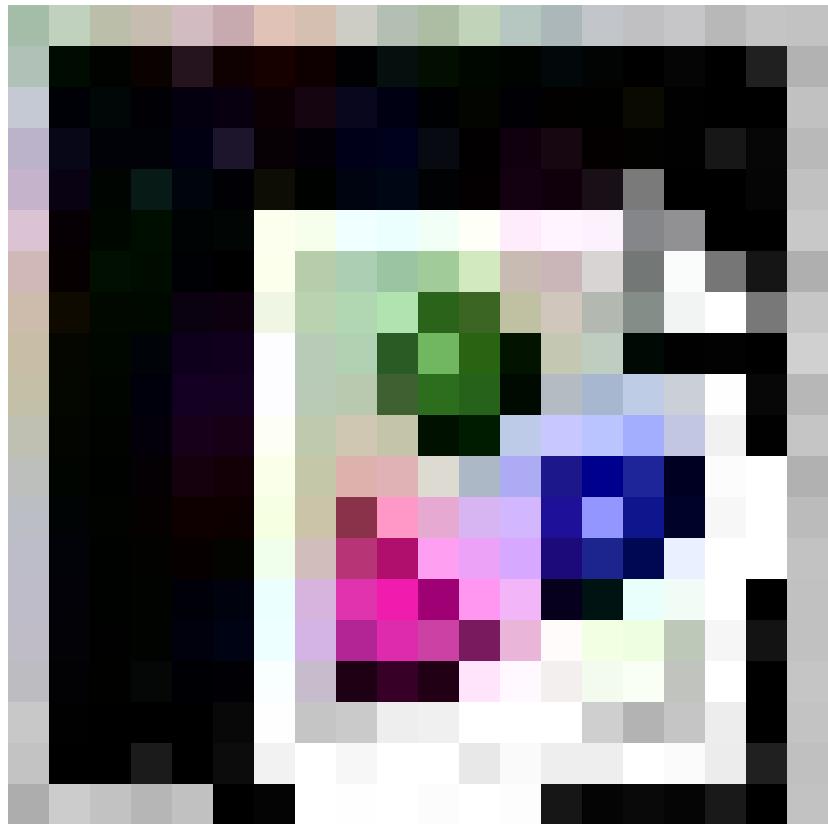
Administrative

- Midterm this Wednesday! woohoo!
- A3 will be out ~Wednesday



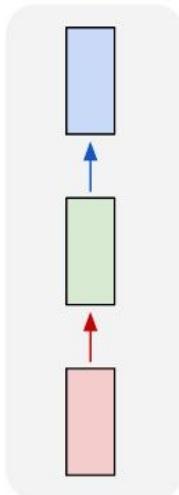
<http://mtyka.github.io/deepdream/2016/02/05/bilateral-class-vis.html>



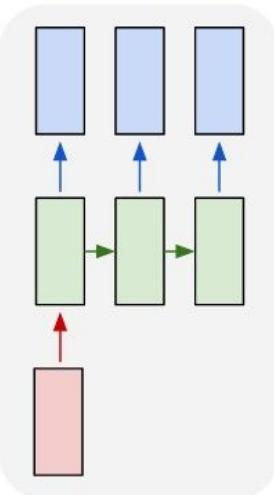


Recurrent Networks offer a lot of flexibility:

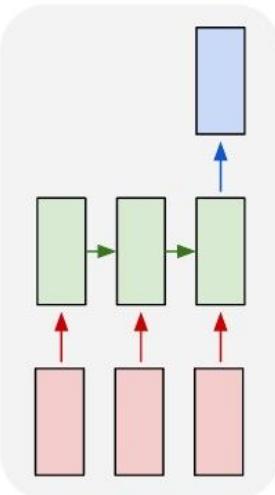
one to one



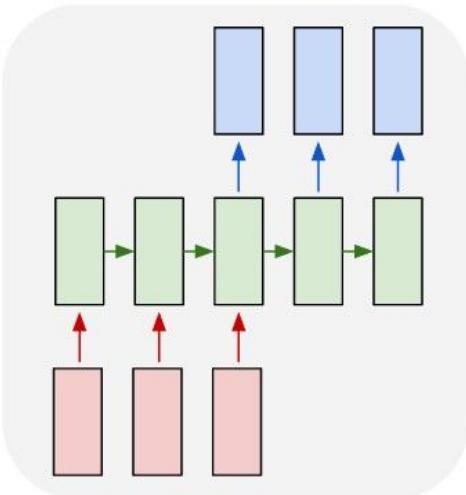
one to many



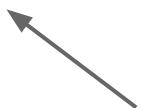
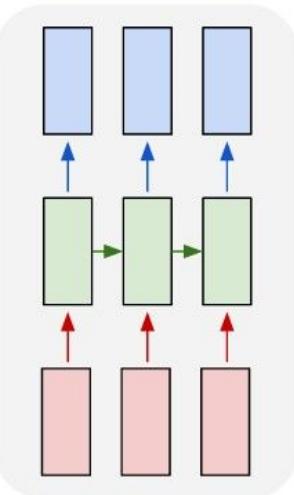
many to one



many to many



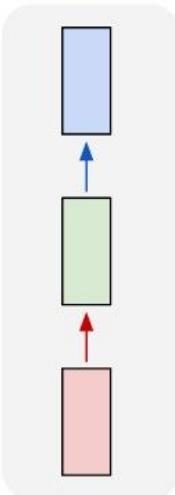
many to many



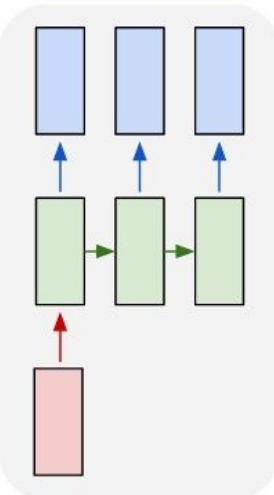
Vanilla Neural Networks

Recurrent Networks offer a lot of flexibility:

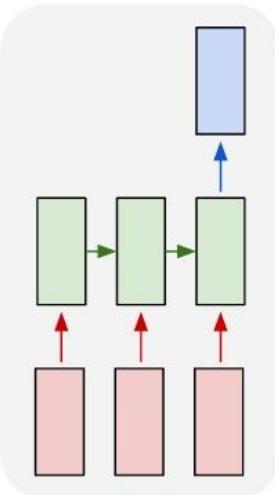
one to one



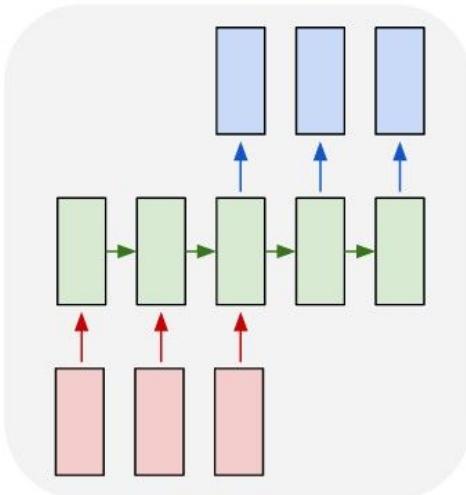
one to many



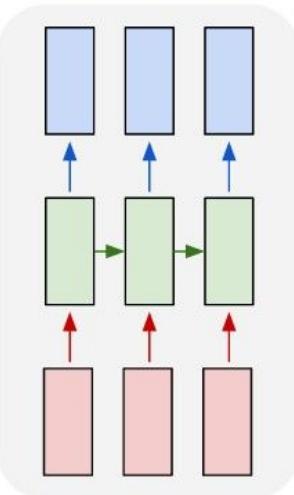
many to one



many to many



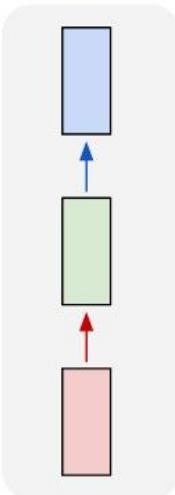
many to many



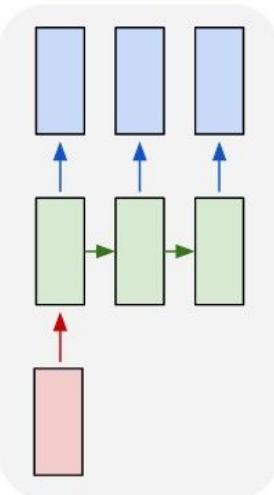
→ e.g. **Image Captioning**
image -> sequence of words

Recurrent Networks offer a lot of flexibility:

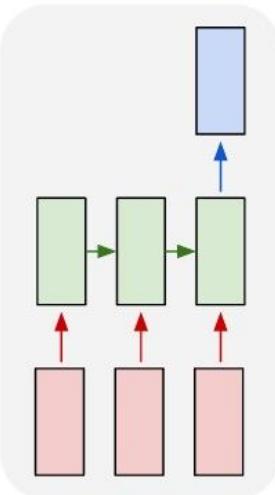
one to one



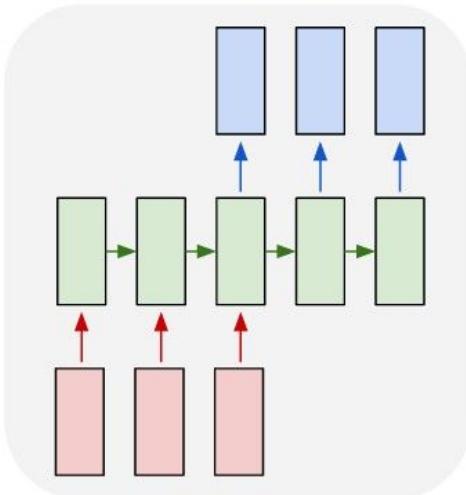
one to many



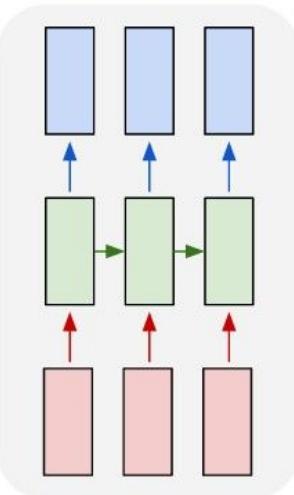
many to one



many to many



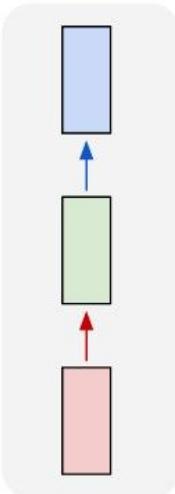
many to many



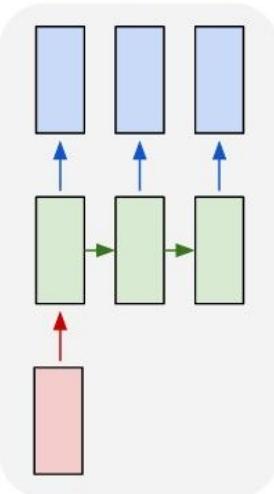
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:

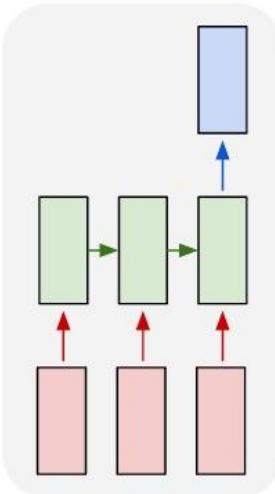
one to one



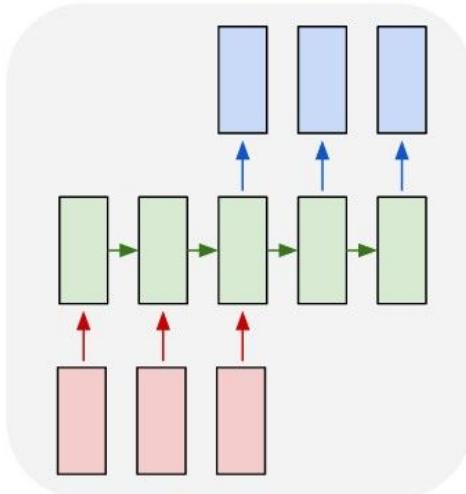
one to many



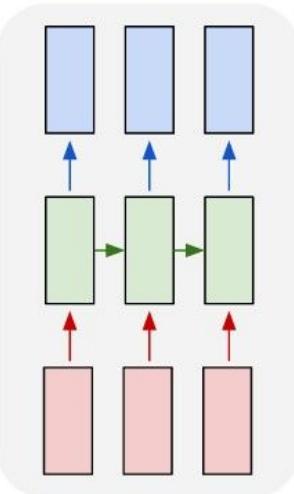
many to one



many to many



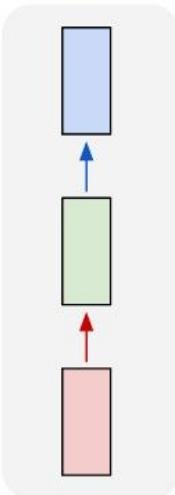
many to many



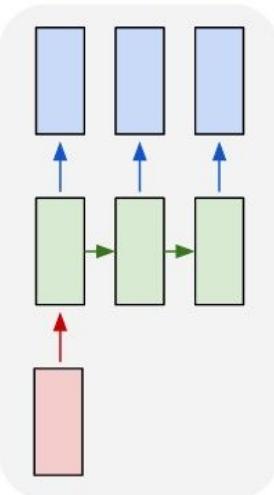
↑
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Networks offer a lot of flexibility:

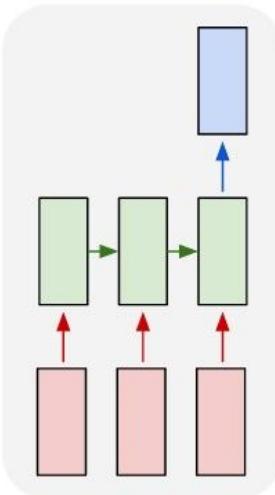
one to one



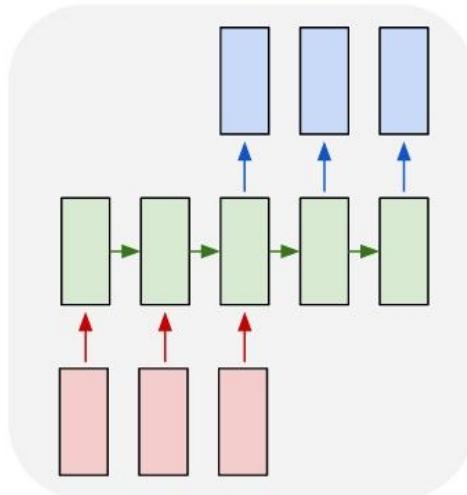
one to many



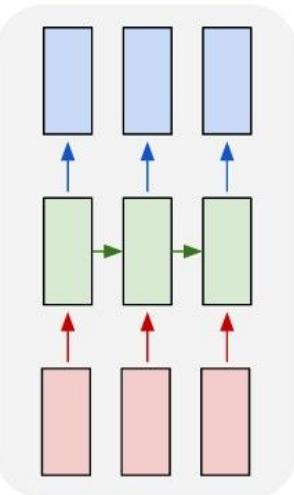
many to one



many to many



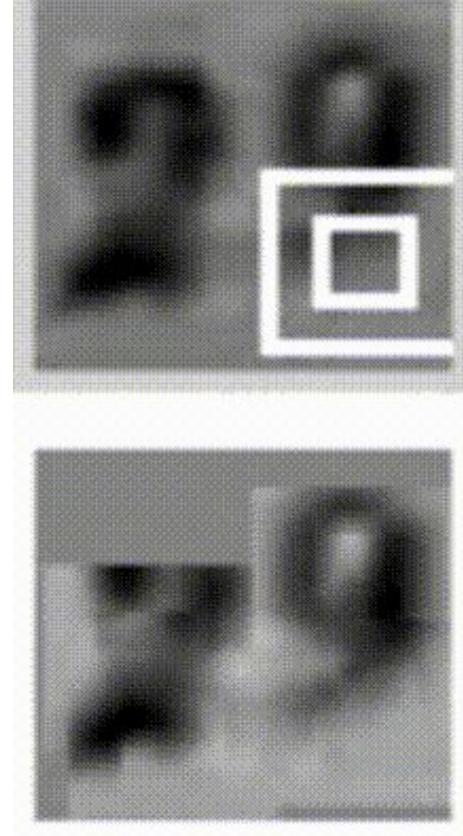
many to many



e.g. Video classification on frame level

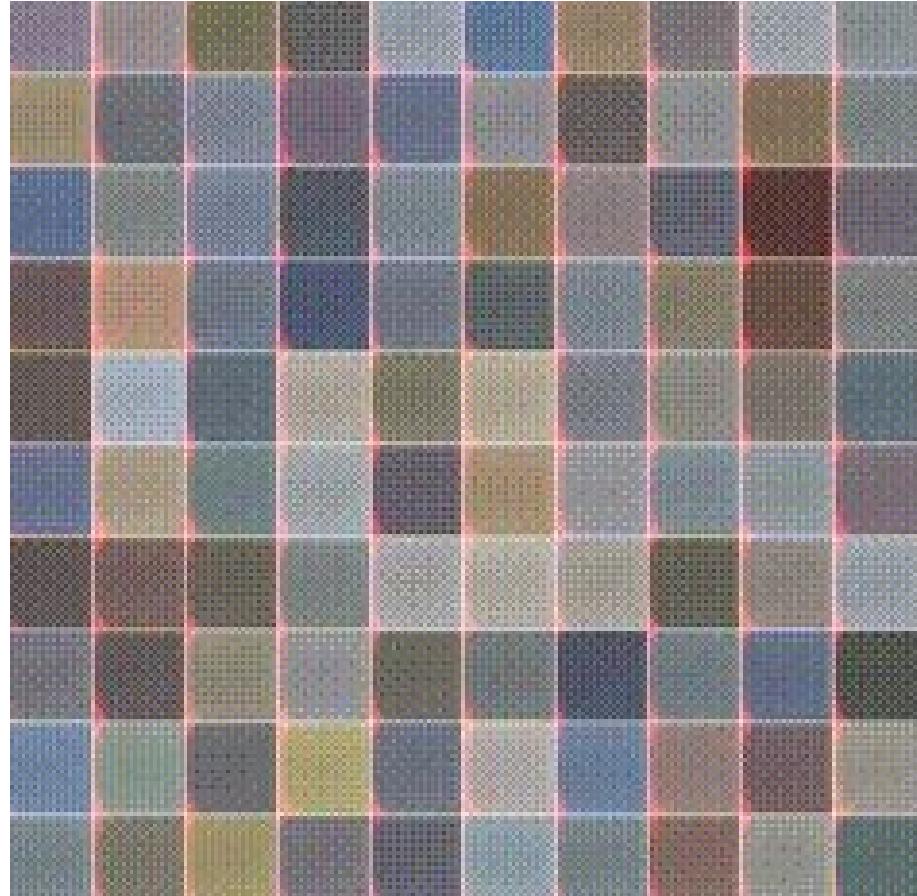
Sequential Processing of fixed inputs

Multiple Object Recognition with
Visual Attention, Ba et al.

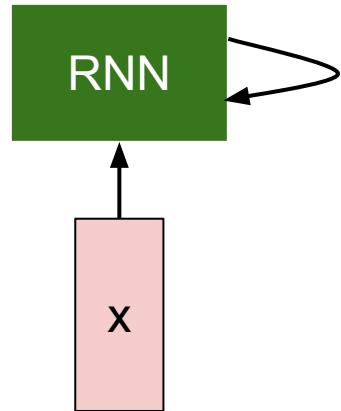


Sequential Processing of fixed outputs

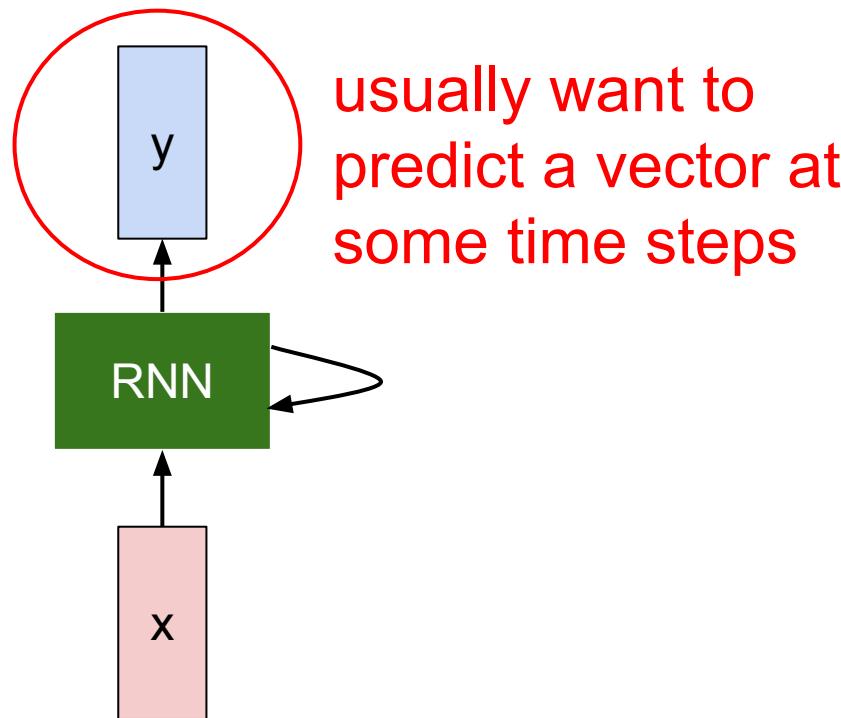
DRAW: A Recurrent
Neural Network For
Image Generation,
Gregor et al.



Recurrent Neural Network

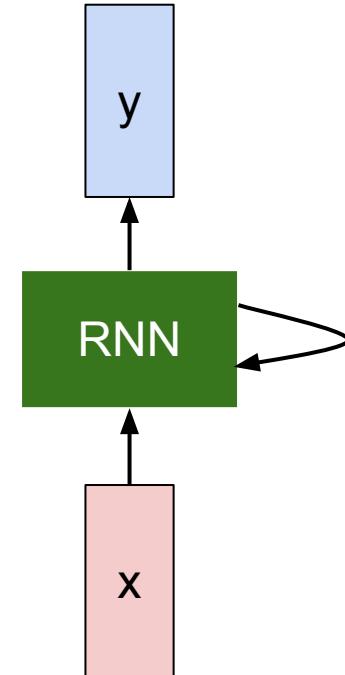


Recurrent Neural Network



Recurrent Neural Network

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

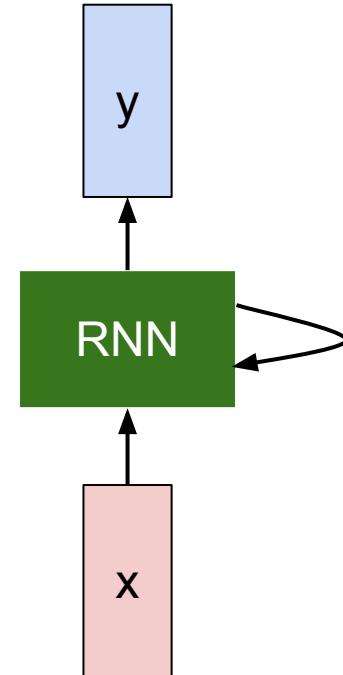


Recurrent Neural Network

We can process a sequence of vectors x by applying a recurrence formula at every time step:

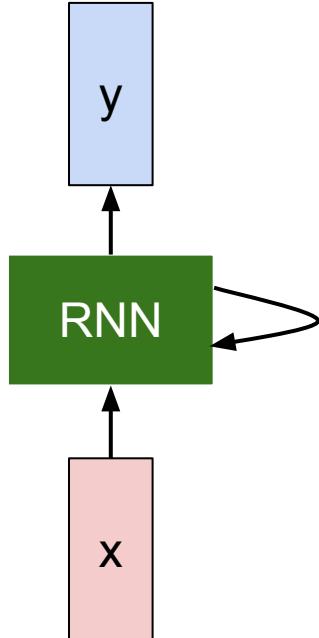
$$h_t = f_W(h_{t-1}, x_t)$$

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 \mathbf{h} :



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

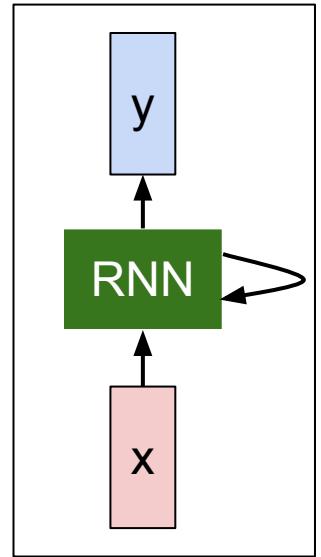
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Character-level language model example

Vocabulary:
[h,e,l,o]

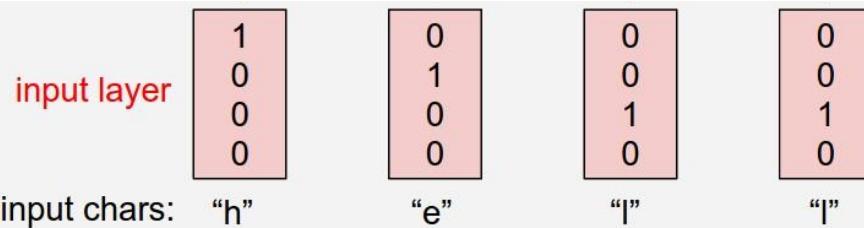
Example training
sequence:
“hello”



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

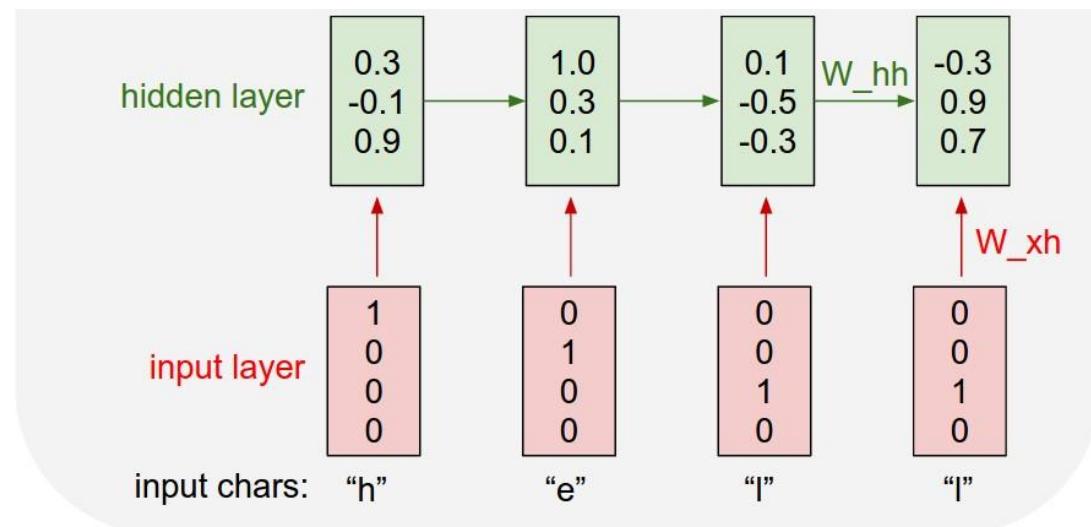


Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

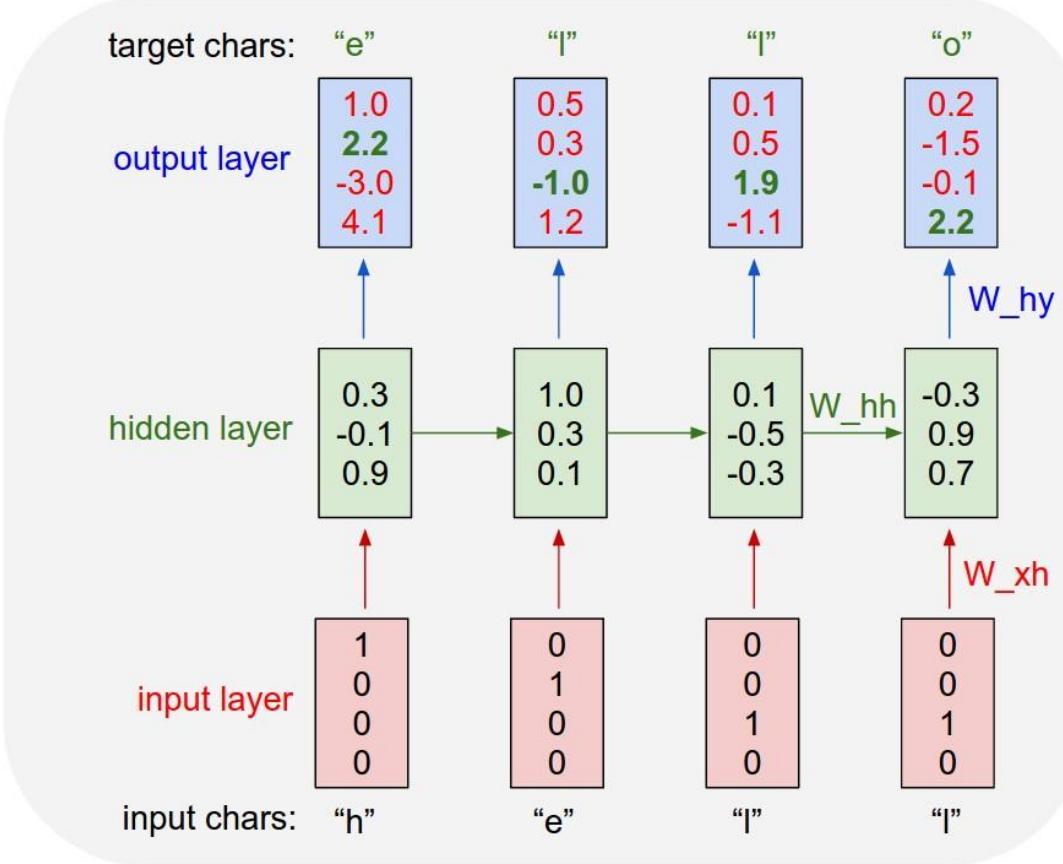
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



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 wkh = np.random.rand(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.rand(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.rand(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 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(wkh, 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
45         # backward pass: compute gradients going backwards
46         dwhx, dwhh, dwhy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
47         dbh, dby = np.zeros_like(bh), np.zeros_like(by)
48         dhnext = np.zeros_like(hs[0])
49         for t2 in reversed(xrange(len(inputs))):
50             dy = np.copy(ps[t2])
51             dy[targets[t2]] -= 1 # backprop into y
52             dyw = np.dot(dy, hs[t2].T)
53             dh = np.dot(why.T, dy) + dhnext # backprop into h
54             ddraw = (i - hs[t2].T) * dh # backprop through tanh nonlinearity
55             ddbh = ddraw
56             dwhh += np.dot(ddraw, xs[t].T)
57             dwhy += np.dot(ddraw, hs[t-1].T)
58             dhnext = np.dot(whh.T, ddraw)
59             for dparam in [dwhx, dwhh, dwhy, dbh, dby]:
60                 np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61
62     return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
```

```
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(wkh, 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
80     return ixes
81
82 n, p = 0, 0
83 mxwh, mwhh, mwhy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
84 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
85 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
86 while True:
87     # prepare inputs (we're sweeping from left to right in steps seq_length long)
88     if p+seq_length >= len(data) or n == 0:
89         hprev = np.zeros((hidden_size,1)) # reset RNN memory
90         p = 0 # go from start of data
91     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
92     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
93
94     # sample from the model now and then
95     if n % 100 == 0:
96         sample_ix = sample(hprev, inputs[0], 200)
97         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
98         print('----\n%s\n----' % (txt, ))
99
100     # forward seq_length characters through the net and fetch gradient
101     loss, dwhx, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
102     smooth_loss = smooth_loss * .999 + loss * .001
103     if n % 100 == 0: print('iter %d, loss: %f' % (n, smooth_loss)) # print progress
104
105     # perform parameter update with Adagrad
106     for param, dparam, mem in zip([wkh, whh, why, bh, by],
107                                   [dwhx, dwhh, dwhy, dbh, dby],
108                                   [mxwh, mwhh, mwhy, mbh, mby]):
109         mem += dparam * dparam
110         param += -learning_rate * param / np.sqrt(mem + 1e-8) # adagrad update
111
112     p += seq_length # move data pointer
113     n += 1 # iteration counter
```

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

min-char-rnn.py gist

Data I/O

```
 0 Minimal character-level vanilla RNN model. Written by Andrej Karpathy (@karpathy)
 1 BSD License
 2
 3
 4 import numpy as np
 5
 6 # Data I/O
 7 data = open('input.txt', 'r').read() # should be simple plain text file
 8 chars = list(set(data))
 9 data_size, vocab_size = len(data), len(chars)
10 print('data has %d characters, %d unique.' % (data_size, vocab_size))
11 print(data[:64])
12
13 to_i, i_to_ch = {c: i for i, c in enumerate(chars)}, {i: c for i, c in enumerate(chars)}
```

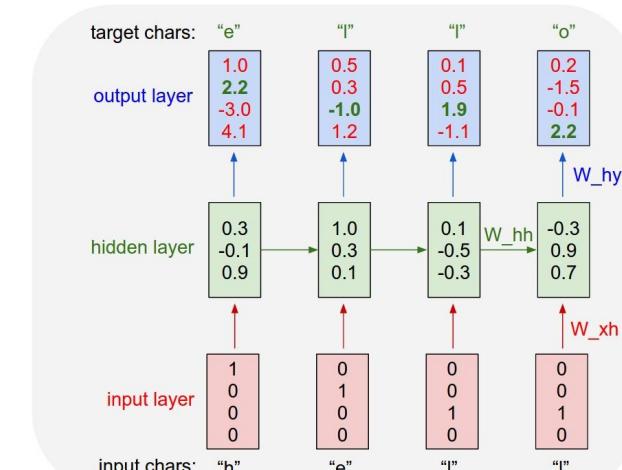
```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 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) }
```

min-char-rnn.py gist

Initializations

```
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 vocab_size = len(chars)
11 data_size = len(data)
12 print("data has %d characters, %d unique." % (data_size, vocab_size))
13 char_to_ix = {ch:i for i in range(len(chars))}
14 ix_to_char = {i:ch for ch in range(len(chars))} # dict to map from index to character
15
16 # hyperparameters
17 hidden_size = 100 # size of hidden layer of neurons
18 seq_length = 20 # number of steps to unroll the RNN for
19 learning_rate = 1e-1
20
21 model_params = {}
22
23 def init_randomhidden(state, vocab_size):
24     wh = np.random.rand(hidden_size, vocab_size)*0.01 # input to hidden
25     bh = np.random.rand(hidden_size, vocab_size)*0.01 # hidden to hidden
26     why = np.random.rand(vocab_size, hidden_size)*0.01 # hidden to output
27     by = np.zeros((vocab_size, 1)) # hidden bias
28     bi = np.zeros((vocab_size, 1)) # output bias
29
30     return wh, bh, why, by, bi
31
32 def lossFun(inputs, targets, hprev):
33
34     inputs,targets = both lists of integers.
35
36     hprev is Hx1 array of initial hidden state
37     returns the loss, gradients on model parameters, and last hidden state
38
39     xs, hs, ys, ps = O, O, O, O
40     h0 = -1 * np.copy(hprev)
41
42     for t in xrange(seq_length):
43         x = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
44         x[inputs[t]:] = 1 # x[t] = 1; all other values 0
45
46         h1 = 1 / np.tanh(np.dot(wh, x) + np.dot(bh, hs[-1]) + bh) # hidden state
47         ps = np.exp(h1) / np.sum(np.exp(h1)) # probabilities for next chars
48
49         loss += -np.log(ps[targets[t]:][0]) # softmax (cross-entropy loss)
50
51         # backprop through tanh nonlinearity
52         dprev, dh0, dwh, dbh, dwhy, dby = np.zeros_like(h0), np.zeros_like(wh), np.zeros_like(bh), np.zeros_like(why)
53         dby = np.zeros_like(by)
54         dnext = np.zeros_like(h1)
55
56         for i in range(len(inputs)):
57             dy = np.copy(ps[i])
58             dy[targets[t]] -= 1 # backprop into y
59             dh = np.dot(dy, wh) # backprop into h
60             dwh += np.dot(dy.T, h0) # dh * backprop through tanh nonlinearity
61             dbh += np.dot(dy, h0)
62             dprev = np.dot(dy, wh.T) # backprop into previous hidden state
63             dnext = np.dot(dy, wh) # backprop into next hidden state
64             for dparam in [dwh, dbh, dwhy, dby]:
65                 np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
66             if dparam != 0:
67                 dparam *= 0.01 # gradient clipping
68
69         if t > 0:
70             h0 = h1 # unroll the model
71
72     h1 = np.zeros((vocab_size, 1)) # memory state, used later for first time step
73
74     x = np.zeros((vocab_size, 1))
75     x[seed_ix] = 1
76
77     for t in xrange(n):
78         h = np.tanh(np.dot(wh, x) + np.dot(bh, h) + bh)
79         p = np.exp(h) / np.sum(np.exp(h))
80         ix = np.random.choice(range(vocab_size), p=p.ravel())
81         x = np.zeros((vocab_size, 1))
82         x[ix] = 1
83
84     return h1
85
86
87 n, p = 0, 0
88 mem, mewh, mbhh, mwhy = np.zeros_like(wh), np.zeros_like(bh), np.zeros_like(why)
89 mem, mbh, mby = np.zeros_like(bh), np.zeros_like(by), np.zeros_like(why)
90 smooth_loss = 0.0
91
92 while True:
93
94     if p+seq_length+1 >= len(data) or n == 0:
95         hprev = np.zeros((hidden_size, 1)) # reset RNN memory
96         p = 0 # go from start of data
97
98     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
99     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
100
101
102     # sample from the model now and then
103     if n % 100 == 0:
104         sample_ix = sample(hprev, inputs[0], 200)
105         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
106         print '----\n %s ----' % (txt, )
107
108
109     # forward seq_length characters through the net and fetch gradient
110     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
111     smooth_loss = smooth_loss * 0.999 + loss * 0.001
112
113     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
114
115
116     # perform parameter update with Adagrad
117     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
118                                   [dWxh, dWhh, dWhy, dbh, dby],
119                                   [mWxh, mWhh, mWhy, mbh, mby]):
120
121         mem += dparam * dparam
122         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
123
124
125         p += seq_length # move data pointer
126
127         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
86     while True:
87
88         # prepare inputs (we're sweeping from left to right in steps seq_length long)
89         if p+seq_length+1 >= len(data) or n == 0:
90             hprev = np.zeros((hidden_size, 1)) # reset RNN memory
91             p = 0 # go from start of data
92
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
97         # sample from the model now and then
98         if n % 100 == 0:
99             sample_ix = sample(hprev, inputs[0], 200)
100             txt = ''.join(ix_to_char[ix] for ix in sample_ix)
101             print '----\n %s ----' % (txt, )
102
103
104         # forward seq_length characters through the net and fetch gradient
105         loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
106         smooth_loss = smooth_loss * 0.999 + loss * 0.001
107
108         if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
109
110
111         # perform parameter update with Adagrad
112         for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
113                                       [dWxh, dWhh, dWhy, dbh, dby],
114                                       [mWxh, mWhh, mWhy, mbh, mby]):
115
116             mem += dparam * dparam
117             param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
118
119
120             p += seq_length # move data pointer
121
122             n += 1 # iteration counter
```

min-char-rnn.py gist

Main loop

Main loop

```
***  
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)  
BSD license  
2013  
1 import numpy as np  
2  
3 # data I/O  
4 with open('ptb.train.txt', 'r') as f: # should be simple plain text file  
5     data = f.read()  
6     chars = list(data)  
7     data_size, vocab_size = len(data), len(chars)  
8     print('data has %d characters, %d unique.' % (data_size, vocab_size))  
9     char_to_ix = {ch:i for i, ch in enumerate(chars)}  
10    ix_to_char = {i:ch for ch, i in enumerate(chars)}  
11  
12    # training parameters  
13    seq_length = 25 # number of steps to unroll the RNN for  
14    learning_rate = 1e-1  
15  
16    # model parameters  
17    wh = np.random.randn(hidden_size, vocab_size)*0.1 # input to hidden  
18    bh = np.zeros((hidden_size, 1)) # hidden to hidden  
19    why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output  
20    by = np.zeros((vocab_size, 1)) # hidden bias  
21    hb = np.zeros(vocab_size, 1) # output bias  
22  
23    def lossFun(inputs, targets, hprev):  
24        """  
25            inputs,targets are both lists of integers.  
26            hprev is Hx1 array of initial hidden state  
27            returns the loss, gradients on model parameters, and last hidden state  
28        """  
29  
30        xs, hs, ys, ps = [], [], [], []  
31        hs[0] = np.copy(hprev)  
32  
33        # forward pass  
34        for t in range(len(inputs)):  
35            x = np.zeros((vocab_size, 1)) # encode in 1-of-k representation  
36            x[inputs[t]] = 1  
37            hprev = np.tanh(np.dot(x, wh) + np.dot(hprev, bh) + hs[-1])  
38            hprev_t = np.tanh(np.dot(hprev, wh) + bh) # hidden state  
39            ps[t] = np.exp(hprev_t) / np.sum(np.exp(hprev_t)) # probabilities for next chars  
40            y = np.argmax(ps[t])  
41            loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)  
42  
43            dbh, dwh, dhy, dyb, dby, dbw, dbyw, dbhwh, dbybh, dbywhy = 0  
44            dwh = np.zeros_like(wh, np.zeros_like(wh, np.zeros_like(why))  
45            dbw = np.zeros_like(bh, np.zeros_like(bh, np.zeros_like(by)))  
46            dhy = np.zeros_like(why, np.zeros_like(why, np.zeros_like(by)))  
47            dby = np.zeros_like(by, np.zeros_like(by, np.zeros_like(by)))  
48            for t in reversed(range(1, len(inputs))):  
49                dy = np.copy(dy[t])  
50                dy = np.dot(dy, wh.T) # backward into y  
51                dyh = np.dot(dy, bh.T) # backward into h  
52                dyh += np.dot(dy, hprev_t.T)  
53                dyh += dy  
54                dh = np.tanh(dyh * (1 - hprev_t**2))  
55                dwh += np.dot(dh, wh.T) # backward through tanh nonlinearity  
56                dbh += np.dot(dh, bh.T)  
57                dhy += np.dot(dyh, why.T) # backward into y  
58                dby += np.dot(dyh, by.T) # backward into y  
59                dbyw += np.dot(dyh, wh.T) # backward into w  
60                dbybh += np.dot(dyh, bh.T) # backward into b  
61                dbywhy += np.dot(dyh, why.T) # backward into y  
62                dbyw += np.dot(dyh, why.T) # backward into y  
63                dbybh += np.dot(dyh, bh.T) # backward into b  
64                dbywhy += np.dot(dyh, why.T) # backward into y  
65                dbyw += np.dot(dyh, why.T) # backward into y  
66                dbybh += np.dot(dyh, bh.T) # backward into b  
67                dbywhy += np.dot(dyh, why.T) # backward into y  
68                dbyw += np.dot(dyh, why.T) # backward into y  
69                dbybh += np.dot(dyh, bh.T) # backward into b  
70                dbywhy += np.dot(dyh, why.T) # backward into y  
71                dbyw += np.dot(dyh, why.T) # backward into y  
72                dbybh += np.dot(dyh, bh.T) # backward into b  
73                dbywhy += np.dot(dyh, why.T) # backward into y  
74                dbyw += np.dot(dyh, why.T) # backward into y  
75                dbybh += np.dot(dyh, bh.T) # backward into b  
76                dbywhy += np.dot(dyh, why.T) # backward into y  
77                dbyw += np.dot(dyh, why.T) # backward into y  
78                dbybh += np.dot(dyh, bh.T) # backward into b  
79                dbywhy += np.dot(dyh, why.T) # backward into y  
80                dbyw += np.dot(dyh, why.T) # backward into y  
81                dbybh += np.dot(dyh, bh.T) # backward into b  
82                dbywhy += np.dot(dyh, why.T) # backward into y  
83                dbyw += np.dot(dyh, why.T) # backward into y  
84                dbybh += np.dot(dyh, bh.T) # backward into b  
85                dbywhy += np.dot(dyh, why.T) # backward into y  
86  
87                # prepare inputs (we're sweeping from left to right in steps seq_length long)  
88                if p+seq_length+1 >= len(data) or n == 0:  
89                    hprev = np.zeros((hidden_size, 1)) # reset RNN memory  
90                    p = 0 # go from start of data  
91  
92                inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]  
93                targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]  
94  
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  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
999  
1000  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1197  
1198  
1199  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1297  
1298  
1299  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1397  
1398  
1399  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1497  
1498  
1499  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1597  
1598  
1599  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1697  
1698  
1699  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1797  
1798  
1799  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1869  
1870  
1871  
1872  
187
```

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[1:(1)])
10 vocab_size = len(chars), len(data), len(chars)
11 print("data has %d characters, %d unique." % (data_size, vocab_size))
12 char_to_ix = {c: i for i in range(len(chars))}
13 ix_to_char = {i: c for c in range(len(chars))}

14 # Hyperparameters
15 hidden_size = 100 # size of hidden layer of neurons
16 seq_length = 20 # number of steps to unroll the RNN for
17 learning_rate = 1e-1
18
19 # Model parameters
20 dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
21 dbh, mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
22 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
23
24 while True:
25     # prepare inputs (we're sweeping from left to right in steps seq_length long)
26     if p+seq_length+1 >= len(data) or n == 0:
27         hprev = np.zeros((hidden_size,1)) # reset RNN memory
28         p = 0 # go from start of data
29         inputs, targets = [ord(c) for c in data[p:p+seq_length]], [ord(c) for c in data[p+1:p+seq_length+1]]
30
31     # Forward pass
32     for t in range(seq_length):
33         x = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
34         x[inputs[t]] = 1
35         hprev = np.tanh(np.dot(wh, x) + np.dot(dbh, hprev) + bh) # hidden state
36         y = np.exp(hprev) # probabilities for next chars
37         py = y / sum(y) # softmax
38         loss += -np.log(py[targets[t]]) # softmax (cross-entropy loss)
39         dx = np.exp(y) # derivatives for next chars
40         dy = np.copy(dy * (1 - py)) # backprop into y
41         dh = np.dot(why.T, dy) + dmem # backprop into h
42         dWhh += np.dot(x.T, h) # dh * backprop through tanh nonlinearity
43         dbh += np.dot(x.T, dh) # dh * backprop through tanh nonlinearity
44         dWxh += np.dot(x.T, py*x) # dx * backprop through tanh nonlinearity
45         dy -= np.dot(why, dy) # backprop into y
46         dWhy += np.dot(h.T, dy) # dh * backprop through tanh nonlinearity
47         dmem += np.dot(h.T, py*x) # dx * backprop through tanh nonlinearity
48         for param in [dWxh, dWhh, dbh, dWhy]:
49             np.clip(param, -5, 5, out=param) # clip to mitigate exploding gradients
50             param += np.nan_to_num(param) # nan's do not count as gradients
51             param *= learning_rate # update parameters
52             dbh, dWhh, dWhy = np.zeros_like(dbh), np.zeros_like(dWhh), np.zeros_like(dWhy)
53
54     # sample from the model now and then
55     if n % 100 == 0:
56         sample_ix = sample(hprev, inputs[0], 200)
57         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
58         print '----\n %s ----' % (txt, )
59
60     # forward seq_length characters through the net and fetch gradient
61     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
62     smooth_loss = smooth_loss * 0.999 + loss * 0.001
63     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
64
65     # perform parameter update with Adagrad
66     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
67                                   [dWxh, dWhh, dWhy, dbh, dby],
68                                   [mWxh, mWhh, mWhy, mbh, mby]):
69         mem += dparam * dparam
70         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
71
72     p += seq_length # move data pointer
73     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
86     while True:
87         # prepare inputs (we're sweeping from left to right in steps seq_length long)
88         if p+seq_length+1 >= len(data) or n == 0:
89             hprev = np.zeros((hidden_size,1)) # reset RNN memory
90             p = 0 # go from start of data
91             inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
92             targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
93
94         # sample from the model now and then
95         if n % 100 == 0:
96             sample_ix = sample(hprev, inputs[0], 200)
97             txt = ''.join(ix_to_char[ix] for ix in sample_ix)
98             print '----\n %s ----' % (txt, )
99
100        # forward seq_length characters through the net and fetch gradient
101        loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
102        smooth_loss = smooth_loss * 0.999 + loss * 0.001
103
104        if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
105
106        # perform parameter update with Adagrad
107        for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
108                                      [dWxh, dWhh, dWhy, dbh, dby],
109                                      [mWxh, mWhh, mWhy, mbh, mby]):
110            mem += dparam * dparam
111            param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
112
113            p += seq_length # move data pointer
114            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 vocab_size = len(chars)
11 data_size, vocab_size = len(data), len(chars)
12 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
13 char_to_ix = {ch:i for i in range(len(chars))}
14 ix_to_char = {i:ch for ch in range(len(chars))} # dict to map indices back to characters
15
16 # Hyperparameters
17 hidden_size = 100 # size of hidden layer of neurons
18 seq_length = 20 # number of steps to unroll the RNN for
19 learning_rate = 1e-1
20
21 # Model parameters
22 dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
23 dbh, mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
24 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
25
26 while True:
27     # Prepare inputs (we're sweeping from left to right in steps seq_length long)
28     if p+seq_length+1 >= len(data) or n == 0:
29         hprev = np.zeros((hidden_size,1)) # reset RNN memory
30         p = 0 # go from start of data
31         inputs = [c for c in data[p:p+seq_length]] # fetch sequence of inputs
32         targets = [char_to_ix[c] for c in data[p+1:p+seq_length+1]] # fetch sequence of targets
33
34     x, hs, ys, ps = o, O, O, O
35     hprev = np.copy(hprev)
36     loss = 0
37
38     for t in range(seq_length):
39         x = np.zeros((vocab_size,1)) # encode in 1-of-k representation
40         x[inputs[t]:] = 1 # This is a sparse vector
41         hprev = np.tanh(np.dot(wh, x) + np.dot(dbh, hprev) + bh) # hidden state
42         y = np.exp(hprev) / np.sum(np.exp(hprev)) # probabilities for next chars
43         loss += -np.log(ps[targets[t], 0]) # softmax (cross-entropy loss)
44
45         dy = np.copy(ps[0])
46         dy[targets[t]] -= 1 # backprop into y
47         dh = np.dot(dy, dotWy) # backprop through tanh nonlinearity
48         dh += np.dot(dy, dotWh * hprev) # dh = backprop through tanh nonlinearity
49         dh += dh * dbh # dh = backprop through tanh nonlinearity
50         dh += np.dot(dy, dotWx * x[1:-1]) # dh = backprop through tanh nonlinearity
51         dh += dh * dx # dh = backprop through tanh nonlinearity
52         for dparam in [dWxh, dWhh, dWhy, dbh, dy]:
53             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
54             dparam += dy * dh # gradient descent
55             dparam += dh * dbh # gradient descent
56             dparam += dh * dotWy # gradient descent
57             dparam += dh * dotWh # gradient descent
58             dparam += dh * dotWx # gradient descent
59
59
60         sample_ix = sample(hprev, inputs[0], 200)
61         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
62         print '----\n%s\n----' % (txt, )
63
64     # forward seq_length characters through the net and fetch gradient
65     loss, dWxh, dWhh, dWhy, dbh, dy, hprev = lossFun(inputs, targets, hprev)
66     smooth_loss = smooth_loss * 0.999 + loss * 0.001
67
68     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
69
70
71     # perform parameter update with Adagrad
72     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
73                                 [dWxh, dWhh, dWhy, dbh, dyb],
74                                 [mWxh, mWhh, mWhy, mbh, mby]):
75         mem += dparam * dparam
76         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
77
78     p += seq_length # move data pointer
79     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
86 while True:
87     # prepare inputs (we're sweeping from left to right in steps seq_length long)
88     if p+seq_length+1 >= len(data) or n == 0:
89         hprev = np.zeros((hidden_size,1)) # reset RNN memory
90         p = 0 # go from start of data
91         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
92         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
93
94     # sample from the model now and then
95     if n % 100 == 0:
96         sample_ix = sample(hprev, inputs[0], 200)
97         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
98         print '----\n%s\n----' % (txt, )
99
100
101     # forward seq_length characters through the net and fetch gradient
102     loss, dWxh, dWhh, dWhy, dbh, dy, hprev = lossFun(inputs, targets, hprev)
103     smooth_loss = smooth_loss * 0.999 + loss * 0.001
104
105     # perform parameter update with Adagrad
106     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
107                                 [dWxh, dWhh, dWhy, dbh, dyb],
108                                 [mWxh, mWhh, mWhy, mbh, mby]):
109         mem += dparam * dparam
110         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
111
112     p += seq_length # move data pointer
113     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 vocab_size = len(chars)
11 data_size = len(data)
12 print("data has %d characters, %d unique." % (data_size, vocab_size))
13 char_to_ix = {ch:i for i in range(len(chars))}
14 ix_to_char = {i:ch for ch in range(len(chars))} # 14
15
16 # Hyperparameters
17 hidden_size = 100 # size of hidden layer of neurons
18 seq_length = 20 # number of steps to unroll the RNN for
19 learning_rate = 1e-1
20
21 # Model parameters
22 dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
23 dbh, mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
24 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
25
26 while True:
27     # Prepare inputs (we're sweeping from left to right in steps seq_length long)
28     if p+seq_length+1 >= len(data) or n == 0:
29         hprev = np.zeros((hidden_size,1)) # reset RNN memory
30         p = 0 # go from start of data
31         inputs = [c for c in data[p:p+seq_length]] # 31
32         targets = [c for c in data[p+1:p+seq_length+1]]
33
34     # Forward pass
35     for t in range(seq_length):
36         x = np.zeros((vocab_size,1)) # encode in 1-of-k representation
37         x[inputs[t]:] = 1 # 37
38         hprev = np.tanh(np.dot(Whh, hprev) + np.dot(Wxh, x) + bh) # hidden state
39         y = np.exp(hprev) # 39
40         mh = np.sum(np.exp(y)) # 40
41         ysoftmax = y / mh # softmax
42         prob = ysoftmax[targets[t]] # probability for next char
43         loss += -np.log(prob[targets[t]]) # cross-entropy loss
44         loss += -np.exp(y[targets[t]]) / np.sum(np.exp(y)) # 44
45
46         # Backward pass: compute gradients
47         dWxh += np.outer(x, hprev) # 47
48         dWhh += np.outer(hprev, hprev) # 48
49         dbh += np.zeros_like(bh) # 49
50         dWhy += np.outer(hprev, y) # 50
51
52         for param in [dWxh, dWhh, dbh, dWhy]:
53             np.clip(param, -5, 5, out=param) # clip to mitigate exploding gradients
54             param -= learning_rate * param # update
55
56         dh = np.dot(Why.T, dy) + dWxh * np.dot(hprev, tanhNonlinearity)
57         dh += dbh # 57
58         dh *= dtanh(hprev) # 58
59         dWhh += np.outer(dh, dh) # 59
60         dWxh += np.outer(dy, dh) # 60
61         dbh += np.sum(dh, axis=1, keepdims=True) # 61
62         dWhy += np.outer(dh, dy) # 63
63
64         for opname in [dWxh, dWhh, dbh, dWhy]:
65             if opname in mem:
66                 mem[opname] += dtanh(hprev) * dh # 65
67             else:
68                 mem[opname] = dtanh(hprev) * dh # 68
69
70     # Sample from the model now and then
71     if n % 100 == 0:
72         sample_ix = sample(hprev, inputs[0], 200) # 72
73         txt = ''.join(ix_to_char[ix] for ix in sample_ix) # 73
74         print('----\n%s\n----' % (txt, ))
75
76     # Forward seq_length characters through the net and fetch gradient
77     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev) # 77
78     smooth_loss = smooth_loss * 0.999 + loss * 0.001 # 78
79     if n % 100 == 0: print('iter %d, loss: %f' % (n, smooth_loss)) # print progress
80
81     # Perform parameter update with Adagrad
82     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by], # 82
83                                 [dWxh, dWhh, dWhy, dbh, dby], # 83
84                                 [mWxh, mWhh, mWhy, mbh, mby]): # 84
85         mem += dparam * dparam # 85
86         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
87
88         p += seq_length # move data pointer
89         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
86     while True:
87         # prepare inputs (we're sweeping from left to right in steps seq_length long)
88         if p+seq_length+1 >= len(data) or n == 0:
89             hprev = np.zeros((hidden_size,1)) # reset RNN memory
90             p = 0 # go from start of data
91             inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]] # 91
92             targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
93
94         # sample from the model now and then
95         if n % 100 == 0:
96             sample_ix = sample(hprev, inputs[0], 200)
97             txt = ''.join(ix_to_char[ix] for ix in sample_ix)
98             print('----\n%s\n----' % (txt, ))
99
100        # forward seq_length characters through the net and fetch gradient
101        loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
102        smooth_loss = smooth_loss * 0.999 + loss * 0.001
103        if n % 100 == 0: print('iter %d, loss: %f' % (n, smooth_loss)) # print progress
104
105        # perform parameter update with Adagrad
106        for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
107                                      [dWxh, dWhh, dWhy, dbh, dby],
108                                      [mWxh, mWhh, mWhy, mbh, mby]):
109            mem += dparam * dparam
110            param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
111
112            p += seq_length # move data pointer
113            n += 1 # iteration counter
```

min-char-rnn.py gist

Loss function

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

```
***  
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 ch in enumerate(chars)}  
  
# Hyperparameters  
hidden_size = 100 # size of hidden layer of neurons  
seq_length = 20 # number of steps to unroll the RNN for  
learning_rate = 1e-1  
  
# Model parameters  
wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden  
bh = 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  
bh0 = np.zeros(hidden_size, 1) # hidden bias  
by = np.zeros(vocab_size, 1) # output bias  
  
def lossFun(inputs, targets, hprev):  
    """  
    inputs,targets are both lists 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  
    for t in xrange(len(inputs)):  
        xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation  
        xs[t][inputs[t]] = 1  
        wht = np.tanh(np.dot(wh, xs[t]) + np.dot(bh, hs[t-1]) + bh) # hidden state  
        whyt = np.dot(why, wht) + by # unnormalized log probabilities for next chars  
        ps[t] = np.exp(whyt) / np.sum(np.exp(whyt)) = softmax (cross-entropy loss)  
        loss += -np.log(ps[t][targets[t],0])  
        dy = np.zeros_like(whyt)  
        dy[targets[t]] = -1 # backprop into y  
        dy = np.dot(dy.T, wh) # backprop through tanh nonlinearity  
        dwh = np.zeros_like(wh) # gradients for wh  
        dwh += np.dot(xs[t].T, dy) # backprop through dot product  
        dbh = np.zeros_like(bh) # gradients for bh  
        dbh += np.sum(dy, axis=0) # backprop through sum  
        dby = np.zeros_like(by) # gradients for by  
        dby += dy # backprop through dot product  
        dh = np.dot(why.T, dy) + dwh + dbh # backprop into h  
        dwh += np.dot(dh, xs[t]) # backprop through tanh nonlinearity  
        ddbh = np.zeros_like(dbh) # gradients for dbh  
        ddbh += np.sum(dh, axis=0) # backprop through sum  
        dby += dh # backprop through dot product  
        for dparam in [dwh, dbh, dby]:  
            np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients  
        for dparam in [dwh, dbh, dby]:  
            dparam -= learning_rate * dparam # update parameter  
            dparam = np.clip(dparam + 1e-8, 0, 1) # adagrad update  
  
    return loss, ps[t][targets[t],0]  
  
def sample(hx, ix):  
    """  
    sample a sequence of integers from the model  
    h is memory state, seed_ix is seed integer for first time step  
    """  
  
    x = np.zeros((vocab_size, 1))  
    x[seed_ix] = 1  
  
    for t in xrange(n):  
        h = np.tanh(np.dot(wh, x) + np.dot(bh, h) + bh)  
        p = np.exp(why * np.dot(h, wh))  
        ix = np.argmax(np.random.multinomial(1, p.ravel()))  
        x[seed_ix] = 0  
        x[i] = 1  
  
    return ix  
  
n, p, t, b  
meh, meh, why = np.zeros_like(wh), np.zeros_like(wh), np.zeros_like(why)  
meh0, meh0, zero = np.zeros_like(bh), np.zeros_like(by), np.zeros_like(by)  
smooth_loss = 0  
for i in range(seq_length):  
    if i == 0:  
        h = np.zeros((hidden_size, 1))  
    else:  
        h = hnext  
    ix = sample(h, seed_ix)  
    print ix, ix_to_char[ix]  
  
    # Forward pass: compute scores through the net and fetch gradient  
    loss, dxh, dwh, dbh, dby, hprev = lossFun(inputs, targets, hprev)  
    smooth_loss = smooth_loss * 0.999 + loss * 0.001  
    if i % 100 == 0:  
        print 'iter %d, loss: %f' % (i, smooth_loss) # print progress  
  
    # Backward pass: compute gradients going backwards  
    dwhh, dwhh, dwhy = np.zeros_like(wh), np.zeros_like(wh), np.zeros_like(why)  
    dbhh, dbhh, dby = np.zeros_like(bh), 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  
        dwhh += np.dot(dy, hs[t].T)  
        dbhh += dy  
        dhnext = np.dot(why.T, dy) + dhnext # backprop into h  
        dwhh += np.dot(dhnext, xs[t].T) * dh # backprop through tanh nonlinearity  
        dbhh += np.sum(dhnext, axis=0) # backprop through sum  
        dby += dhnext # backprop through dot product  
        for dparam in [dwhh, dbhh, dby]:  
            np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients  
        for dparam in [dwhh, dbhh, dby]:  
            dparam -= learning_rate * dparam # update parameter  
            dparam = np.clip(dparam + 1e-8, 0, 1) # adagrad update  
  
    return loss, dxh, dwhh, dwhh, dwhy, dbhh, dbhh, dby, hprev
```



```
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, dyb = 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         dWxh += np.dot(dy, xs[t].T)  
52         dbh += dy  
53         dh = np.dot(Why.T, dy) + dhnext # backprop into h  
54         dWhh = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity  
55         dbh += dh  
56         dWxh += np.dot(dWhh, xs[t].T)  
57         dWhh += np.dot(dWhh, hs[t-1].T)  
58         dhnext = np.dot(Why.T, dh)  
59     for dparam in [dWxh, dWhh, dWhy, dbh, dyb]:  
60         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients  
61     return loss, dWxh, dWhh, dWhy, dbh, dyb, hs[len(inputs)-1]
```

min-char-rnn.py gist

```

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

15 # Hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 20 # 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 by = np.zeros(vocab_size, 1) # output bias

```

```

28 def lossFun(inputs, targets, hprev):
29     """"
30     inputs,targets are both list of integers.
31     hprev is Hx1 array of initial hidden state
32     returns the loss, gradients on model parameters, and last hidden state
33     """
34
35     xs, hs, ys, ps = {}, {}, {}, {}
36     hs[-1] = np.copy(hprev)
37     loss = 0
38
39     # forward pass
40     for t in xrange(len(inputs)):
41         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
42         xs[t][inputs[t]] = 1
43
44         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
45
46         ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
47         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
48
49         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
50
51     # backward pass: compute gradients going backwards
52     dchh, dbh, dhy = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(why)
53     dch, dby = np.zeros_like(bh), np.zeros_like(by)
54     dhnext = np.zeros_like(hs[0])
55
56     for t in reversed(xrange(len(inputs)))�
57         dy = np.copy(ps[t])
58         dy[targets[t]] -= 1 # backprop into y
59         dhy = np.dot(dy, hs[t].T)
60         dbh += np.sum(dhy, axis=0, keepdims=True)
61         dchh += np.sum(dhy * dhnext, axis=0, keepdims=True)
62         dhnext = np.dot(Why.T, dy) * np.tanh(dchh)
63
64     for opname in [dchh, dbh, dhy, dby]:
65         np.clip(opname, -5, 5, out=opname) # clip to mitigate exploding gradients
66
67     return loss, hs[-1], ps
68
69 def sample(hprev, seed_ix, n):
70     """"
71     sample a sequence of integers from the model
72     h is memory state, seed_ix is seed letter for first time step
73     """
74     x = np.zeros((vocab_size, 1))
75     x[seed_ix] = 1
76
77     for t in xrange(n):
78         h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
79         p = np.exp(ys[t]) / np.sum(np.exp(ys[t]))
80         ix = np.random.choice(range(vocab_size), p=p.ravel())
81         x[1:] = 0
82         x[i] = 1
83
84     return ix
85
86 n, p = 0, 0
87
88 mem, memh, memy = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(why)
89 memh, memy, mem = np.zeros_like(bh), np.zeros_like(by), np.zeros_like(why)
90 smooth_loss = 0
91 seqLength = len(data)/seq_length + 1
92
93 while True:
94     if p == seqLength:
95         print 'resetting from left to right in step %d, length long' % p
96         if p+seqLength > len(data) or p == 0:
97             pseq = np.zeros((hidden_size, 1)) # reset RNN memory
98             memh, memy, mem = pseq, pseq, pseq
99
100     inputs = [char_to_ix[ch] for ch in data[p:p+seqLength]]
101     targets = [char_to_ix[ch] for ch in data[p+seqLength-1:p+seqLength]]
102
103     if p < 1:
104         hprev = np.zeros((hidden_size, 1))
105         memh, memy, mem = np.zeros_like(bh), np.zeros_like(by), np.zeros_like(why)
106
107     a = np.zeros((hidden_size, 1))
108     a[0] = 1
109
110     sample_ix = sample(hprev, inputs[0], 200)
111     print 'junkix_to_char[%d]' % sample_ix
112     print 'char_ix_to_char[%d]' % sample_ix
113
114     hprev = np.zeros((hidden_size, 1))
115
116     # Forward seq_length characters through the net and fetch gradient
117     loss, dchh, dbh, dhy, dby, hprev = lossFun(inputs, targets, hprev)
118     smooth_loss = smooth_loss * 0.999 + loss * 0.001
119
120     if p % 100 == 0: print 'iter %d, loss: %f, smooth loss: %f' % (p, smooth_loss)
121
122     p += 1
123
124     # backward pass: compute gradients going backwards
125     for opname, oparr in zip([dchh, dbh, dhy, dby],
126                             [dchh, dbh, dhy, dby],
127                             [memh, memy, mem, mem]):
128         oparr *= -learning_rate * dchh / np.sqrt(dchh + 1e-8) # adaptive update
129
130
131     p = seqLength * n + data_pointer
132
133     n += 1 # iteration counter

```

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

$$y_t = W_{hy}h_t$$

Softmax classifier

```

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
34     xs, hs, ys, ps = {}, {}, {}, {}
35     hs[-1] = np.copy(hprev)
36     loss = 0
37
38     # forward pass
39     for t in xrange(len(inputs)):
40         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
41         xs[t][inputs[t]] = 1
42
43         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
44
45         ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
46         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
47
48         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
49
50     # backward pass: compute gradients going backwards
51     dchh, dbh, dhy = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(why)
52     dch, dby = np.zeros_like(bh), np.zeros_like(by)
53     dhnext = np.zeros_like(hs[0])
54
55     for t in reversed(xrange(len(inputs)))�
56         dy = np.copy(ps[t])
57         dy[targets[t]] -= 1 # backprop into y
58         dhy = np.dot(dy, hs[t].T)
59         dbh += np.sum(dhy, axis=0, keepdims=True)
60         dchh += np.sum(dhy * dhnext, axis=0, keepdims=True)
61         dhnext = np.dot(Why.T, dy) * np.tanh(dchh)
62
63     for opname in [dchh, dbh, dhy, dby]:
64         np.clip(opname, -5, 5, out=opname) # clip to mitigate exploding gradients
65
66     return loss, hs[-1], ps
67
68
69 def sample(hprev, seed_ix, n):
70     """"
71     sample a sequence of integers from the model
72     h is memory state, seed_ix is seed letter for first time step
73     """
74     x = np.zeros((vocab_size, 1))
75     x[seed_ix] = 1
76
77     for t in xrange(n):
78         h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
79         p = np.exp(ys[t]) / np.sum(np.exp(ys[t]))
80         ix = np.random.choice(range(vocab_size), p=p.ravel())
81         x[1:] = 0
82         x[i] = 1
83
84     return ix
85
86 n, p = 0, 0
87
88 mem, memh, memy = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(why)
89 memh, memy, mem = np.zeros_like(bh), np.zeros_like(by), np.zeros_like(why)
90 smooth_loss = 0
91 seqLength = len(data)/seq_length + 1
92
93 while True:
94     if p == seqLength:
95         print 'resetting from left to right in step %d, length long' % p
96         if p+seqLength > len(data) or p == 0:
97             pseq = np.zeros((hidden_size, 1)) # reset RNN memory
98             memh, memy, mem = pseq, pseq, pseq
99
100     inputs = [char_to_ix[ch] for ch in data[p:p+seqLength]]
101     targets = [char_to_ix[ch] for ch in data[p+seqLength-1:p+seqLength]]
102
103     if p < 1:
104         hprev = np.zeros((hidden_size, 1))
105         memh, memy, mem = np.zeros_like(bh), np.zeros_like(by), np.zeros_like(why)
106
107     a = np.zeros((hidden_size, 1))
108     a[0] = 1
109
110     sample_ix = sample(hprev, inputs[0], 200)
111     print 'junkix_to_char[%d]' % sample_ix
112     print 'char_ix_to_char[%d]' % sample_ix
113
114     hprev = np.zeros((hidden_size, 1))
115
116     # Forward seq_length characters through the net and fetch gradient
117     loss, dchh, dbh, dhy, dby, hprev = lossFun(inputs, targets, hprev)
118     smooth_loss = smooth_loss * 0.999 + loss * 0.001
119
120     if p % 100 == 0: print 'iter %d, loss: %f, smooth loss: %f' % (p, smooth_loss)
121
122     p += 1
123
124     # backward pass: compute gradients going backwards
125     for opname, oparr in zip([dchh, dbh, dhy, dby],
126                             [dchh, dbh, dhy, dby],
127                             [memh, memy, mem, mem]):
128         oparr *= -learning_rate * dchh / np.sqrt(dchh + 1e-8) # adaptive update
129
130
131     p = seqLength * n + data_pointer
132
133     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
6 import numpy as np
7
8 # Data I/O
9 data = open('ptb.train.txt', 'r').read() # should be simple plain text file
10 chars = list(set(data))
11 data_size, vocab_size = len(data), len(chars)
12 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
13 char_to_ix = {ch:i for i in xrange(len(chars))}
14 ix_to_char = {i:ch for ch in xrange(len(chars))}

15 # Hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 20 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19 model_params = {}

20 wh = np.random.rand(hidden_size, hidden_size)*0.01 # input to hidden
21 bh = np.random.rand(hidden_size, hidden_size)*0.01 # hidden to hidden
22 why = np.random.rand(vocab_size, hidden_size)*0.01 # hidden to output
23 bh_0 = np.zeros(hidden_size, 1) # hidden bias
24 by = np.zeros(vocab_size, 1) # output bias

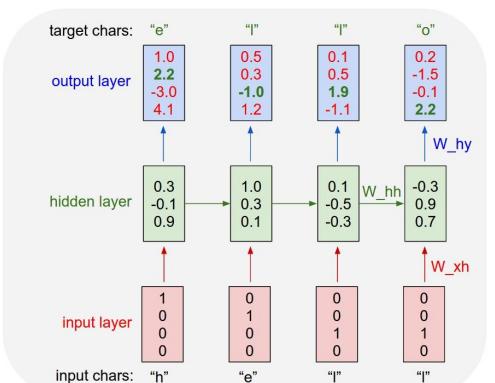
25 def lossFun(inputs, targets, hprev):
26     """ inputs,targets are both lists of integers.
27     hprev is Hx1 array of initial hidden state
28     returns the loss, gradients on model parameters, and last hidden state
29     """
30
31     xs, hs, ys, ps, o, h, dy, dhy, dbh, dby, dwh, dhnext = np.zeros((seq_length+1, hidden_size))
32     hprev = np.copy(hprev)
33     loss = 0
34     for t in xrange(seq_length):
35         x = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
36         x[inputs[t]] = 1
37         hprev = np.tanh(np.dot(hprev, wh) + np.dot(x, bh) + bh)
38         y = np.dot(hprev, why) # hidden state to output guess
39         y = np.exp(y) / np.sum(np.exp(y)) # softmax
40         ps = np.exp(y[t]) / np.sum(np.exp(y[t])) # probabilities for next chars
41         loss += -np.log(ps[targets[t]] if targets[t] > 0 else ps[0]) # softmax (cross-entropy loss)
42
43         dhy = np.zeros_like(why)
44         dwh = np.zeros_like(wh)
45         dbh = np.zeros_like(bh)
46         dby = np.zeros_like(by)
47         dhnext = np.zeros_like(h)
48         dy = np.zeros_like(ps)
49         dy[targets[t]] -= 1 # backprop into y
50         dhy += np.dot(dy, hs[t].T)
51         dwh += np.dot(dhy, xs[t].T)
52         dbh += np.dot(dhy, hprev.T)
53         dhnext = np.dot(why.T, dy) + dhnext # backprop through tanh nonlinearity
54
55         dy += dy * dhy
56         dwh += np.dot(dhnext, xs[t].T)
57         dwhh += np.dot(dhnext, hs[t-1].T)
58         dhnext = np.dot(whh.T, dhnext)
59
60         for dparam in [dwh, dbh, dhy, dby]:
61             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
62
63     return loss, dwh, dbh, dhy, dby, dhnext[inputs[-1]]

64 def sample(seed_ix, n):
65     """ sample a sequence of integers from the model
66     h is memory state, seed_ix is seed integer for first time step
67     """
68
69     h = np.zeros((hidden_size, 1))
70     x = np.zeros((vocab_size, 1))
71     x[seed_ix] = 1
72
73     for t in xrange(n):
74         h = np.tanh(np.dot(h, wh) + np.dot(x, bh) + bh)
75         p = np.exp(h) / np.sum(np.exp(h))
76         ix = np.random.choice(range(vocab_size), p=p.ravel())
77         x[1] = 0
78         x[i] = 1
79
80     return ix

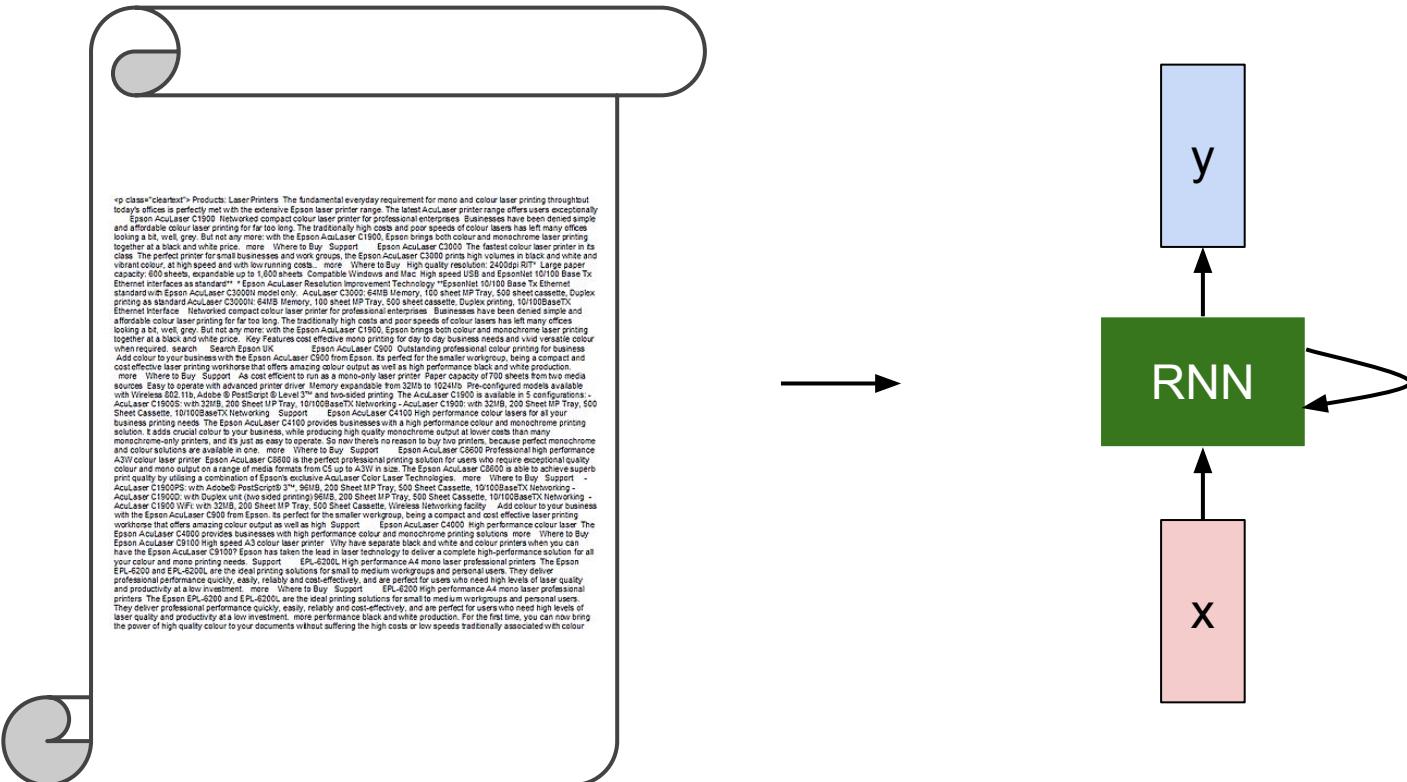
81 n, p = 0
82
83 mem, mem_h, mem_b, mem_y = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(bh)
84 mem_d, mem_db, mem_dy = np.zeros_like(dbh), np.zeros_like(dwhh), np.zeros_like(dbh)
85 smooth_loss = 0
86 smooth_loss_n = 0
87 smooth_loss_w = 0
88
89 while True:
90     if p == seq_length or (p >= seq_length and np.sum(mem) > 0):
91         if p == seq_length:
92             print 'done'
93             break
94         p = 0
95         mem, mem_h, mem_b, mem_y = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(bh)
96         mem_d, mem_db, mem_dy = np.zeros_like(dbh), np.zeros_like(dwhh), np.zeros_like(dbh)
97         smooth_loss = smooth_loss / smooth_loss_n
98         smooth_loss_n = 0
99         smooth_loss_w = 0
100
101         inputs = [char_to_ix[ch] for ch in data[p:seq_length]]
102         targets = [char_to_ix[ch] for ch in data[p+seq_length-1:p+seq_length]]
103
104         # Forward pass: Compute activations through the net and fetch gradient
105         loss, dwh, dbh, dhy, dbh, dhy, hprev = lossFun(inputs, targets, hprev)
106         smooth_loss += smooth_loss * 0.999 + loss * 0.001
107         smooth_loss_n += 1
108         smooth_loss_w += 1
109
110         for param, mem in zip([wh, whh, bh, why], [mem, mem_h, mem_b, mem_y]):
111             mem += param
112             mem_d += np.dot(dparam, hprev.T)
113             mem_db += np.dot(dparam, hprev.T)
114             mem_dy += np.dot(dparam, xs[p].T)
115
116             mem += np.zeros_like(param)
117             mem_d += np.zeros_like(param)
118             mem_db += np.zeros_like(param)
119             mem_dy += np.zeros_like(param)
120
121             mem *= learning_rate
122             mem_d *= learning_rate
123             mem_db *= learning_rate
124             mem_dy *= learning_rate
125
126             param -= np.dot(mem, mem_d + mem_db + mem_dy) / np.sqrt(mem * mem_d + mem_db + mem_dy)
127
128         p += 1
129
130         if p % 100 == 0:
131             print 'iter %d, loss %.3f' % (p, smooth_loss)
132
133
134 # backward pass: compute gradients going backwards
135 dwhxh, dwhhh, dwhy = np.zeros_like(whxh), np.zeros_like(whhh), np.zeros_like(why)
136 dbh, dby = np.zeros_like(bh), np.zeros_like(by)
137 dhnext = np.zeros_like(hs[0])
138
139 for t in reversed(xrange(len(inputs))):
140     dy = np.copy(ps[t])
141     dy[targets[t]] -= 1 # backprop into y
142     dwhy += np.dot(dy, hs[t].T)
143     dby += dy
144
145     dh = np.dot(why.T, dy) + dhnext # backprop into h
146     dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
147     dbh += dhraw
148
149     dwhxh += np.dot(dhraw, xs[t].T)
150     dwhhh += np.dot(dhraw, hs[t-1].T)
151     dhnext = np.dot(whh.T, dhraw)
152
153     for dparam in [dwhxh, dwhhh, dwhy, dbh, dby]:
154         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
155
156 return loss, dwhxh, dwhhh, dwhy, dbh, dby, hs[len(inputs)-1]

```

recall:



min-char-rnn.py gist



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 tkldrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

↓ train more

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwyl 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 nues 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 aps, 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.

open source textbook on algebraic geometry

The Stacks Project

home about tags explained tag lookup browse search bibliography recent comments blog add slogans

Browse chapters

| Part | Chapter | online | TeX source | view pdf |
|---------------|-------------------------|------------------------|---------------------|---------------------|
| Preliminaries | 1. Introduction | online | tex | pdf |
| | 2. Conventions | online | tex | pdf |
| | 3. Set Theory | online | tex | pdf |
| | 4. Categories | online | tex | pdf |
| | 5. Topology | online | tex | pdf |
| | 6. Sheaves on Spaces | online | tex | pdf |
| | 7. Sites and Sheaves | online | tex | pdf |
| | 8. Stacks | online | tex | pdf |
| | 9. Fields | online | tex | pdf |
| | 10. Commutative Algebra | online | tex | pdf |

Parts

- [Preliminaries](#)
- [Schemes](#)
- [Topics in Scheme Theory](#)
- [Algebraic Spaces](#)
- [Topics in Geometry](#)
- [Deformation Theory](#)
- [Algebraic Stacks](#)
- [Miscellany](#)

Statistics

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source

For $\bigoplus_{n=1,\dots,m} \mathcal{L}_{m,n} = 0$, hence we can find a closed subset \mathcal{H} in \mathcal{H} and any sets \mathcal{F} on X , U is a closed immersion of S , then $U \rightarrow T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \rightarrow V$. Consider the maps M along the set of points Sch_{fppf} and $U \rightarrow U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ???. Hence we obtain a scheme S and any open subset $W \subset U$ in $\text{Sh}(G)$ such that $\text{Spec}(R') \rightarrow S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S . We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\text{GL}_{S'}(x'/S'')$ and we win. \square

To prove study we see that $\mathcal{F}|_U$ is a covering of \mathcal{X}' , and \mathcal{T}_i is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and \mathcal{F}_p exists and let \mathcal{F}_i be a presheaf of \mathcal{O}_X -modules on \mathcal{C} as a \mathcal{F} -module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\text{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)_{fppf}^{\text{opp}}, (\text{Sch}/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \rightarrow (U, \text{Spec}(A))$$

is an open subset of X . Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S .

Proof. See discussion of sheaves of sets. \square

The result for prove any open covering follows from the less of Example ???. It may replace S by $X_{\text{spaces},\text{étale}}$ which gives an open subspace of X and T equal to S_{Zar} , see Descent, Lemma ???. Namely, by Lemma ?? we see that R is geometrically regular over S .

Lemma 0.1. Assume (3) and (3) by the construction in the description.

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{\text{Proj}}_X(\mathcal{A}) = \text{Spec}(B)$ over U compatible with the complex

$$\text{Set}(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \rightarrow \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S . Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1) f is locally of finite type. Since $S = \text{Spec}(R)$ and $Y = \text{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X . But given a scheme U and a surjective étale morphism $U \rightarrow X$. Let $U \cap U = \coprod_{i=1,\dots,n} U_i$ be the scheme X over S at the schemes $X_i \rightarrow X$ and $U = \lim_i X_i$. \square

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{x,\dots,x_0}$.

Lemma 0.2. Let X be a locally Noetherian scheme over S , $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq p$ is a subset of $\mathcal{J}_{n,0} \circ \mathcal{A}_2$ works.

Lemma 0.3. In Situation ???. Hence we may assume $q' = 0$.

Proof. We will use the property we see that p is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F -algebra where δ_{n+1} is a scheme over S . \square

Proof. Omitted. □

Lemma 0.1. Let \mathcal{C} be a set of the construction.

Let \mathcal{C} be a gerber covering. Let \mathcal{F} be a quasi-coherent sheaves of \mathcal{O} -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves \mathcal{F} on $X_{\text{étale}}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where \mathcal{G} defines an isomorphism $\mathcal{F} \rightarrow \mathcal{F}$ of \mathcal{O} -modules. □

Lemma 0.2. This is an integer \mathcal{Z} is injective.

Proof. See Spaces, Lemma ??.

Lemma 0.3. Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

be a morphism of algebraic spaces over S and Y .

Proof. Let X be a nonzero scheme of X . Let X be an algebraic space. Let \mathcal{F} be a quasi-coherent sheaf of \mathcal{O}_X -modules. The following are equivalent

- (1) \mathcal{F} is an algebraic space over S .
- (2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{ccccc}
 S & \xrightarrow{\quad} & & & \\
 \downarrow & & & & \\
 \xi & \longrightarrow & \mathcal{O}_{X'} & \xrightarrow{\quad} & \\
 \text{gor}_s & & \uparrow & \searrow & \\
 & & =\alpha' \longrightarrow & & \\
 & & \downarrow & & \\
 & & =\alpha' \longrightarrow & & \\
 & & \downarrow & & \\
 \text{Spec}(K_\psi) & & \text{Mor}_{\text{Sets}} & & d(\mathcal{O}_{X_{X/k}}, \mathcal{G}) \\
 & & & & \downarrow X \\
 & & & & \text{d}(\mathcal{O}_{X_{X/k}}, \mathcal{G})
 \end{array}$$

is a limit. Then \mathcal{G} is a finite type and assume S is a flat and \mathcal{F} and \mathcal{G} is a finite type f_* . This is of finite type diagrams, and

- the composition of \mathcal{G} is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

Proof. We have see that $X = \text{Spec}(R)$ and \mathcal{F} is a finite type representable by algebraic space. The property \mathcal{F} is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U . □

Proof. This is clear that \mathcal{G} is a finite presentation, see Lemmas ??.

A reduced above we conclude that U is an open covering of \mathcal{C} . The functor \mathcal{F} is a “field”

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\bar{x}} \xrightarrow{-1} (\mathcal{O}_{X_{\text{étale}}}) \longrightarrow \mathcal{O}_{X_{\bar{x}}}^{-1} \mathcal{O}_{X_{\lambda}}(\mathcal{O}_{X_{\eta}}^{\bar{v}})$$

is an isomorphism of covering of \mathcal{O}_{X_i} . If \mathcal{F} is the unique element of \mathcal{F} such that X is an isomorphism.

The property \mathcal{F} is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme \mathcal{O}_X -algebra with \mathcal{F} are opens of finite type over S . If \mathcal{F} is a scheme theoretic image points. □

If \mathcal{F} is a finite direct sum \mathcal{O}_{X_k} is a closed immersion, see Lemma ???. This is a sequence of \mathcal{F} is a similar morphism.

 torvalds / linux Watch · 3,711 Star · 23,054 Fork · 9,141

Linux kernel source tree

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 agolatest commit 4b1786927d  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/hex2fw.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

 ios

ios: remove kernfs_node->name from kobj_uevent

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 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffffff8) & 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/seteew.h>
#include <asm/pgproto.h>
```

```

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/seteew.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)
{
#endif 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;
}
```

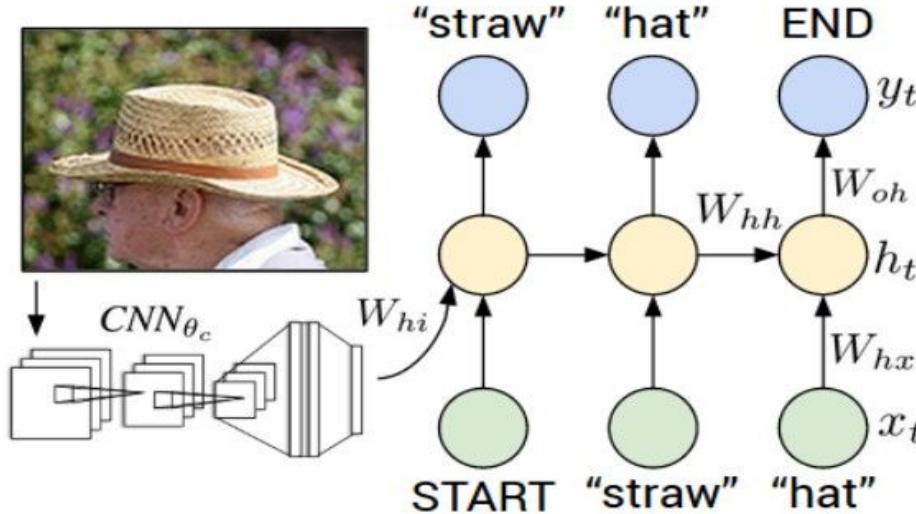
quote/comment cell

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.

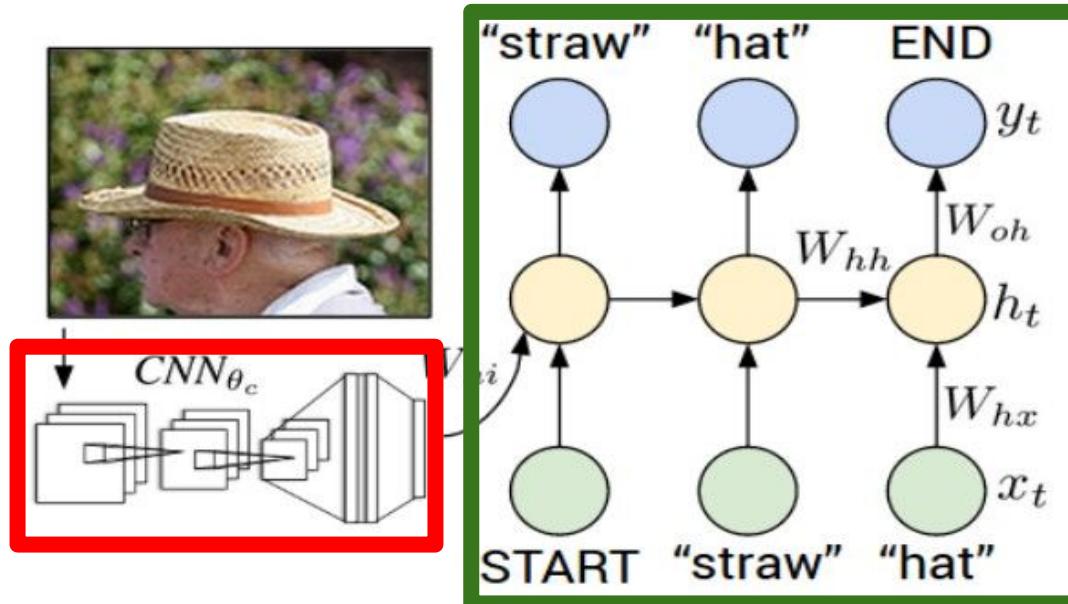
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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



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

FC-1000

softmax

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

FC-1000

softmax



image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

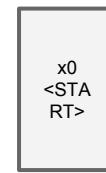
conv-512

conv-512

maxpool

FC-4096

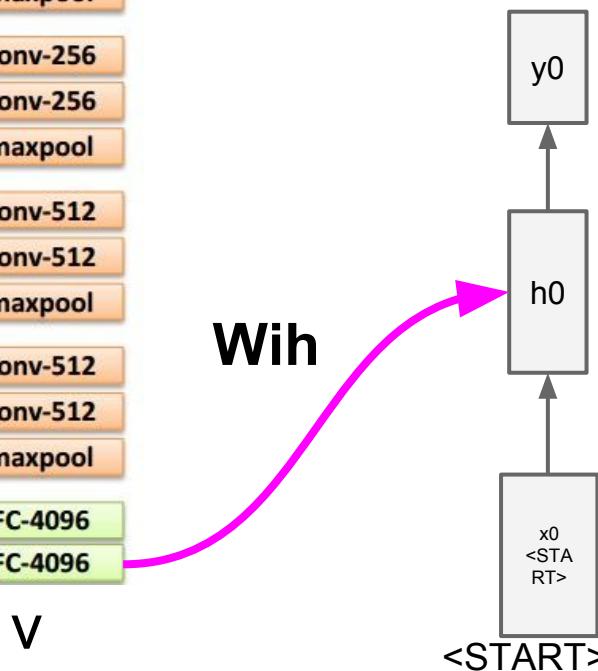
FC-4096



<START>



test image



before:

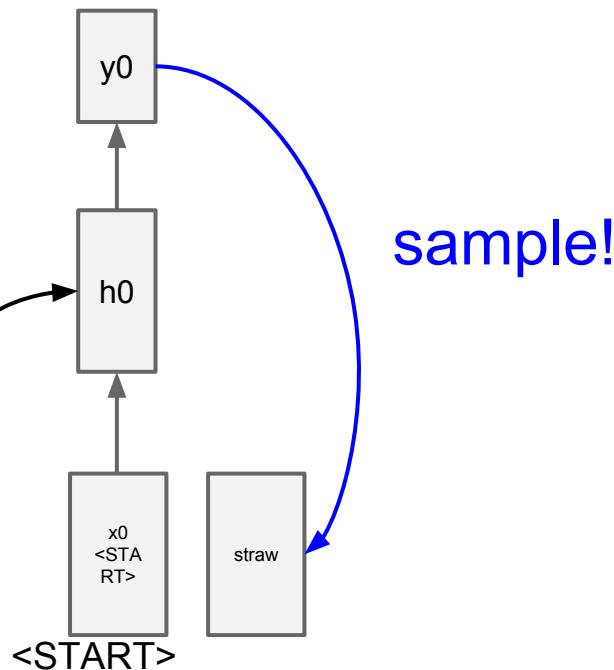
$$h = \tanh(Wxh * x + Whh * h)$$

now:

$$h = \tanh(Wxh * x + Whh * h + WiH * v)$$



test image



image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

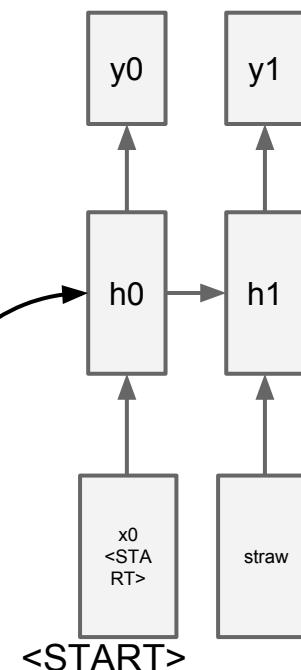
conv-512

conv-512

maxpool

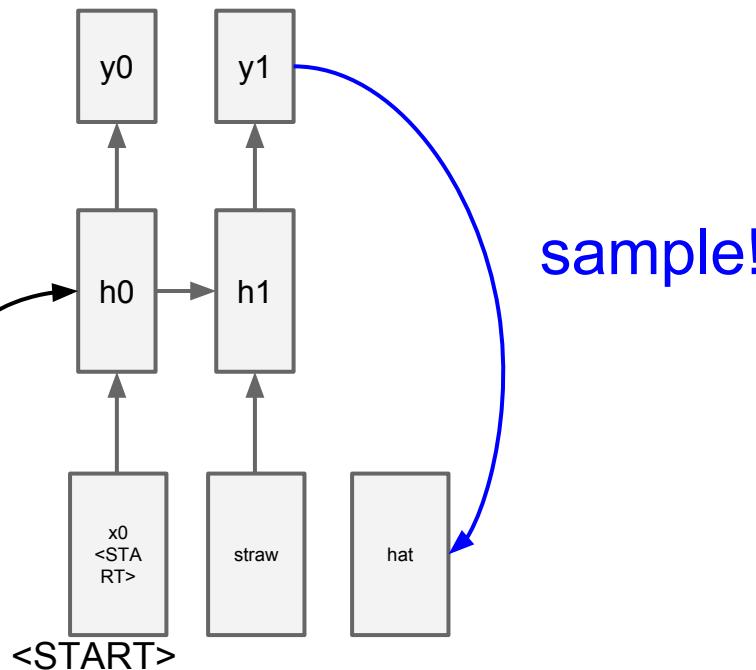
FC-4096

FC-4096





test image



image



test image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

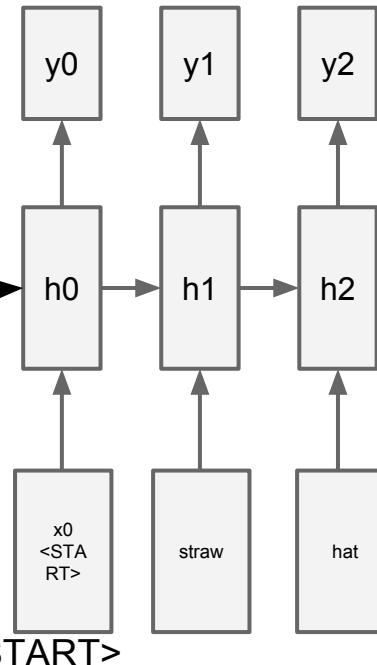
conv-512

conv-512

maxpool

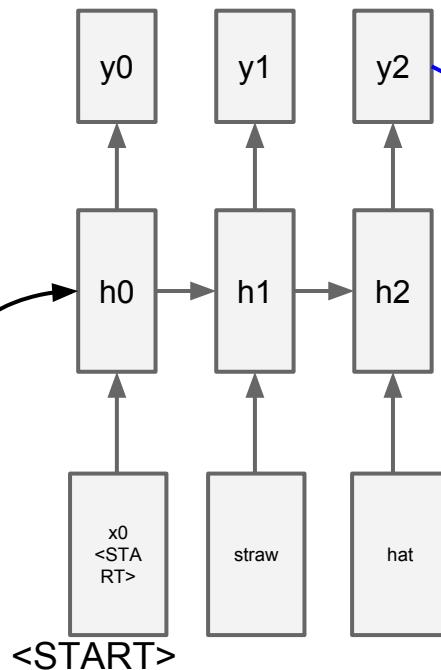
FC-4096

FC-4096





test image



sample
<END> token
=> finish.

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."



"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."



"a young boy is holding a baseball bat."



"a cat is sitting on a couch with a remote control."



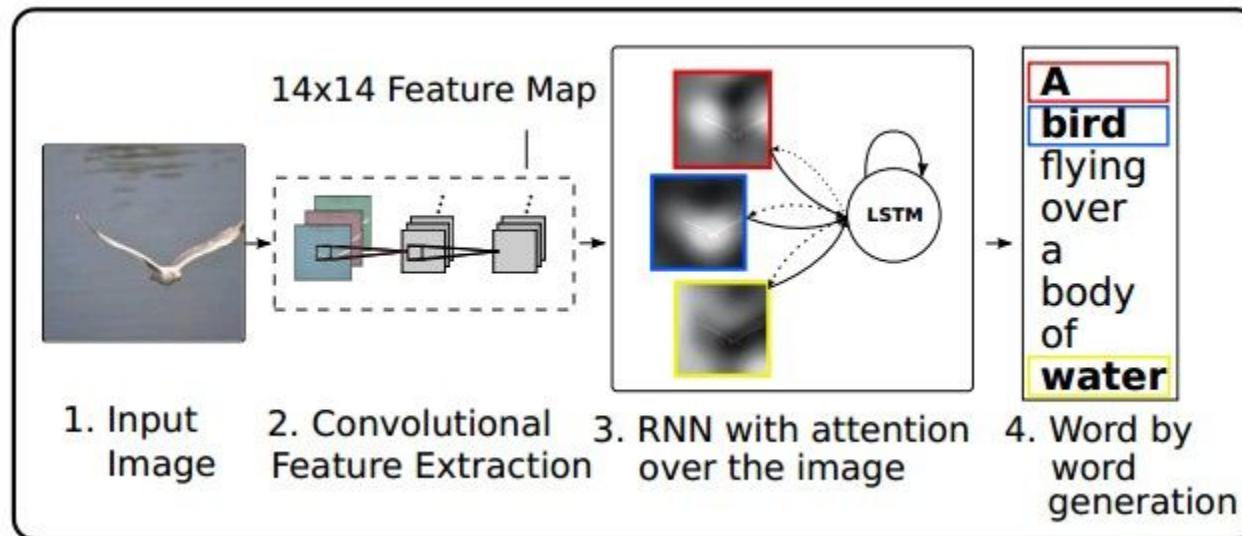
"a woman holding a teddy bear in front of a mirror."



"a horse is standing in the middle of a road."

Preview of fancier architectures

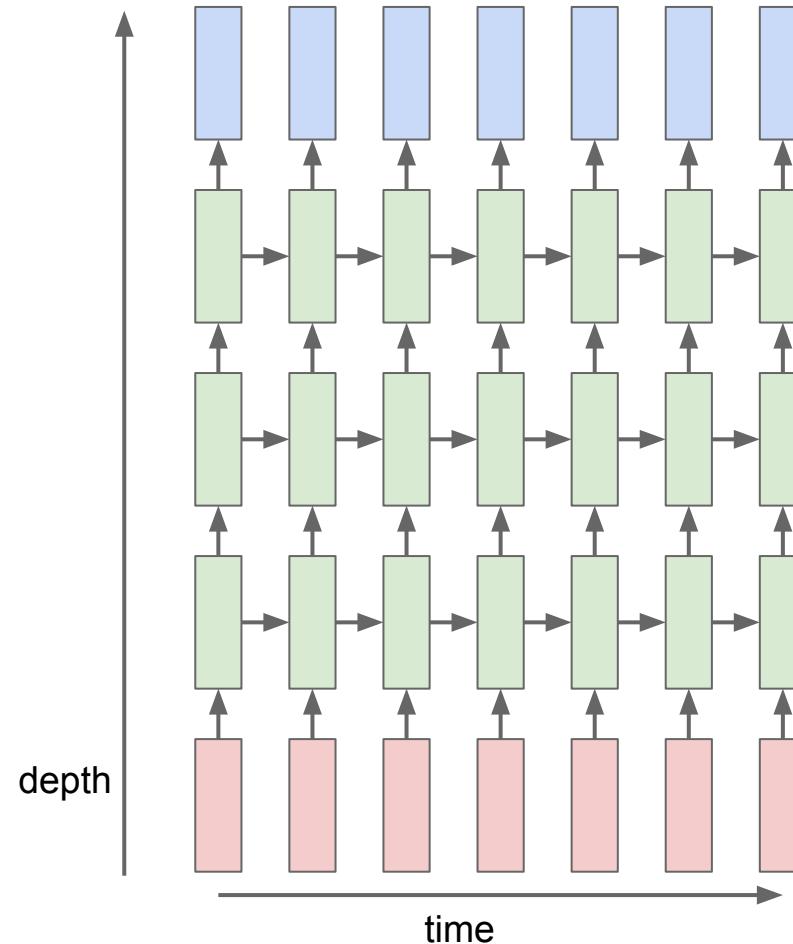
RNN attends spatially to different parts of images while generating each word of the sentence:



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 × 2n]



RNN:

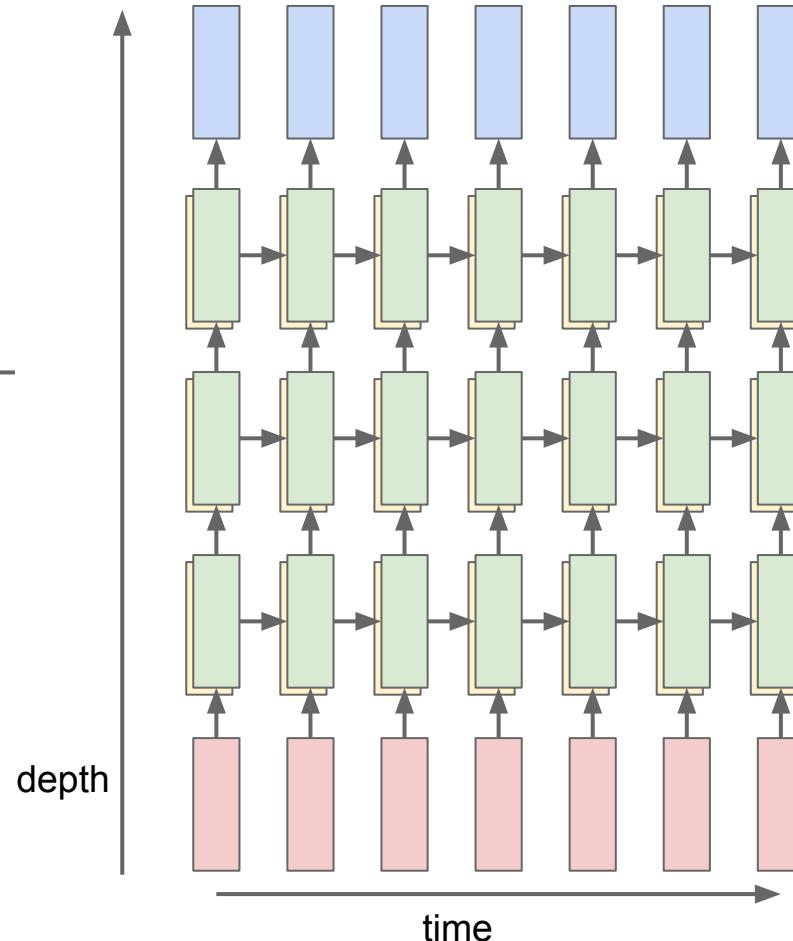
$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

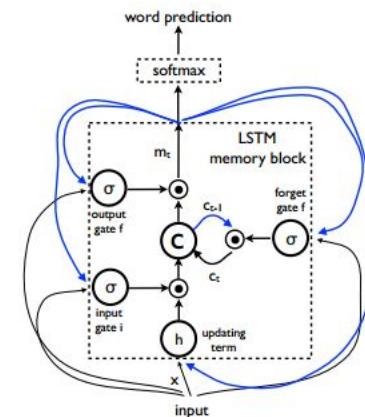
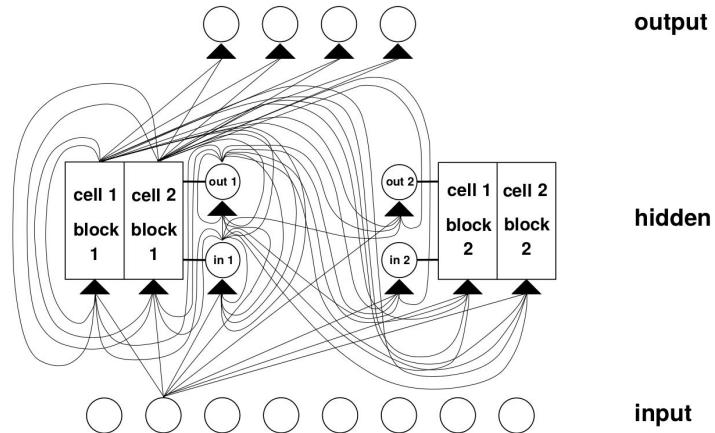
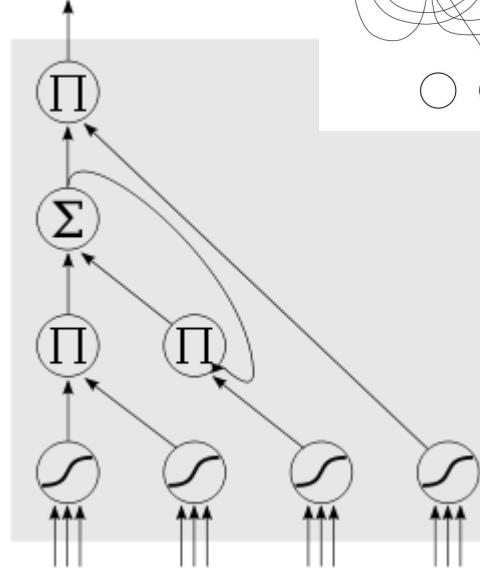
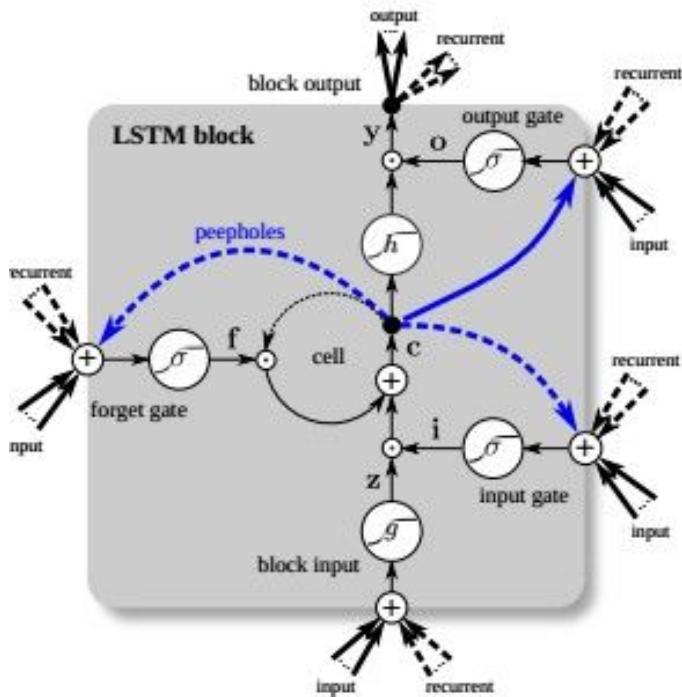
LSTM:

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

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

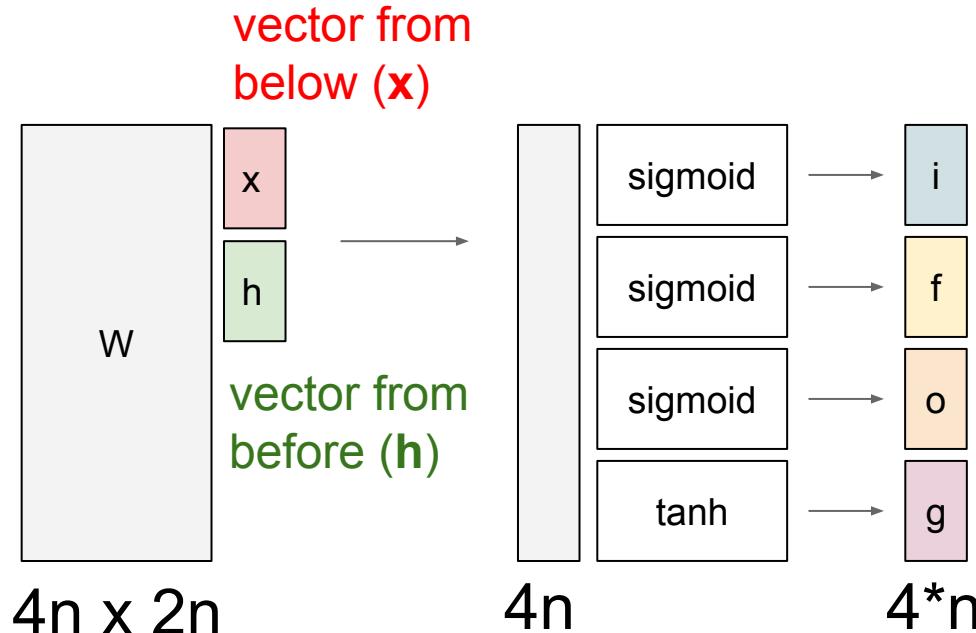


LSTM



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

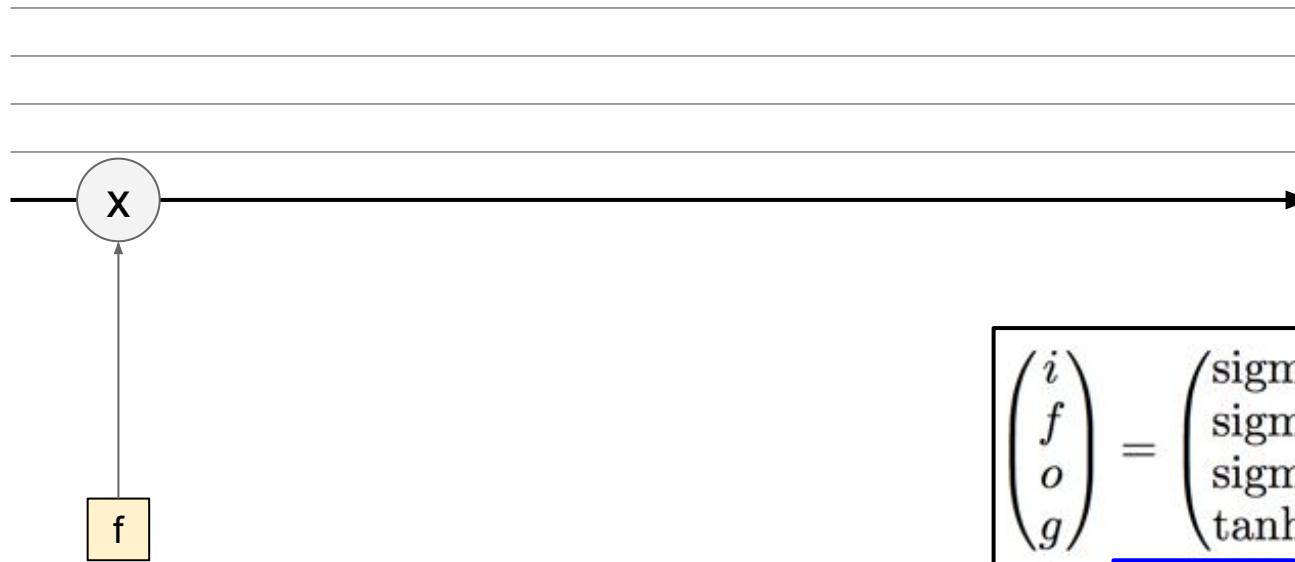


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_t^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state **c**

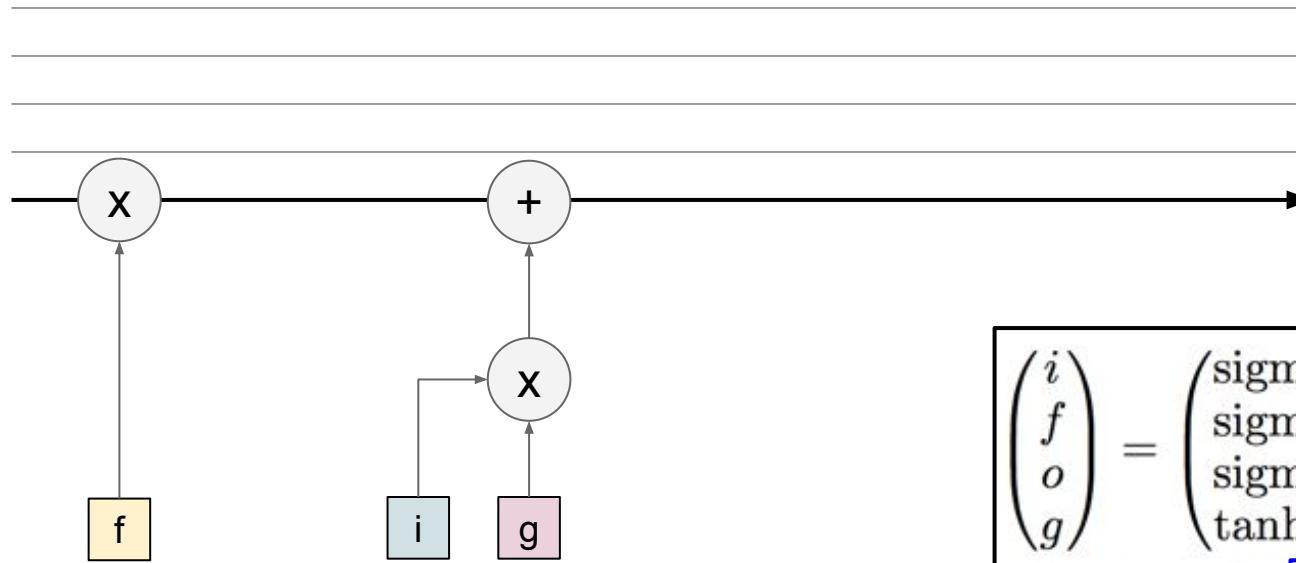


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c

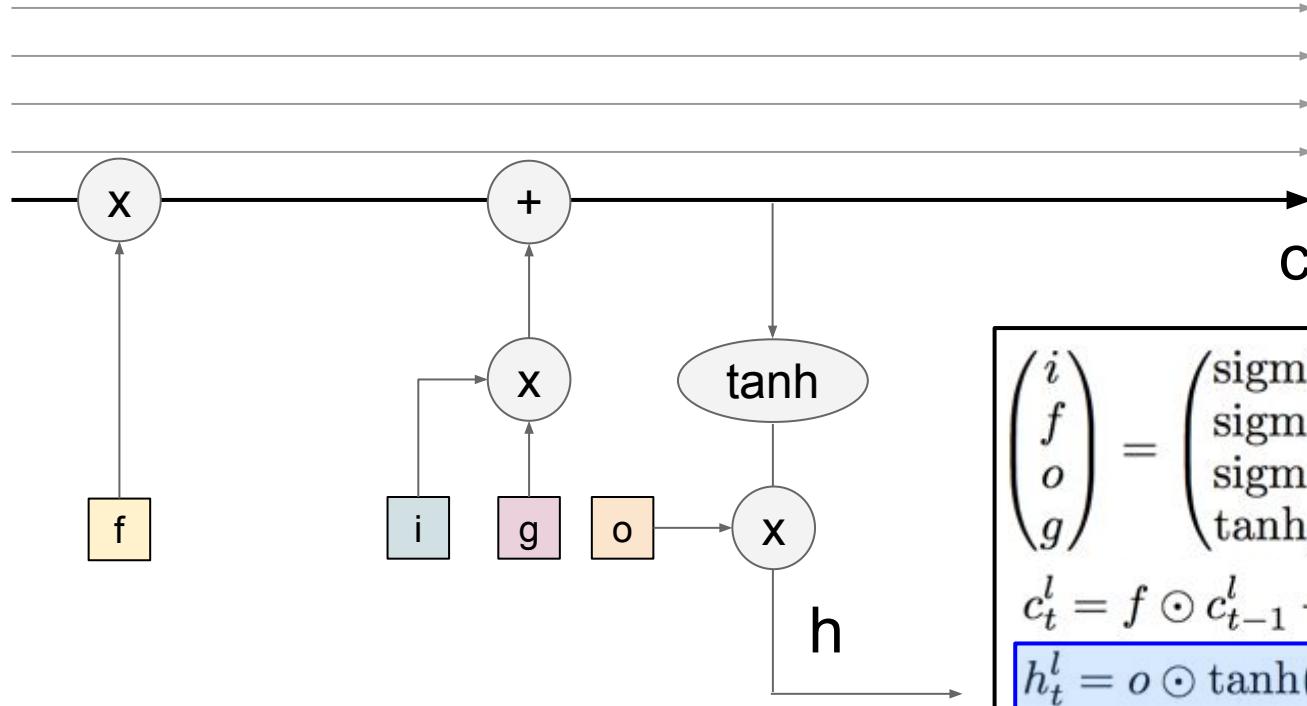


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c

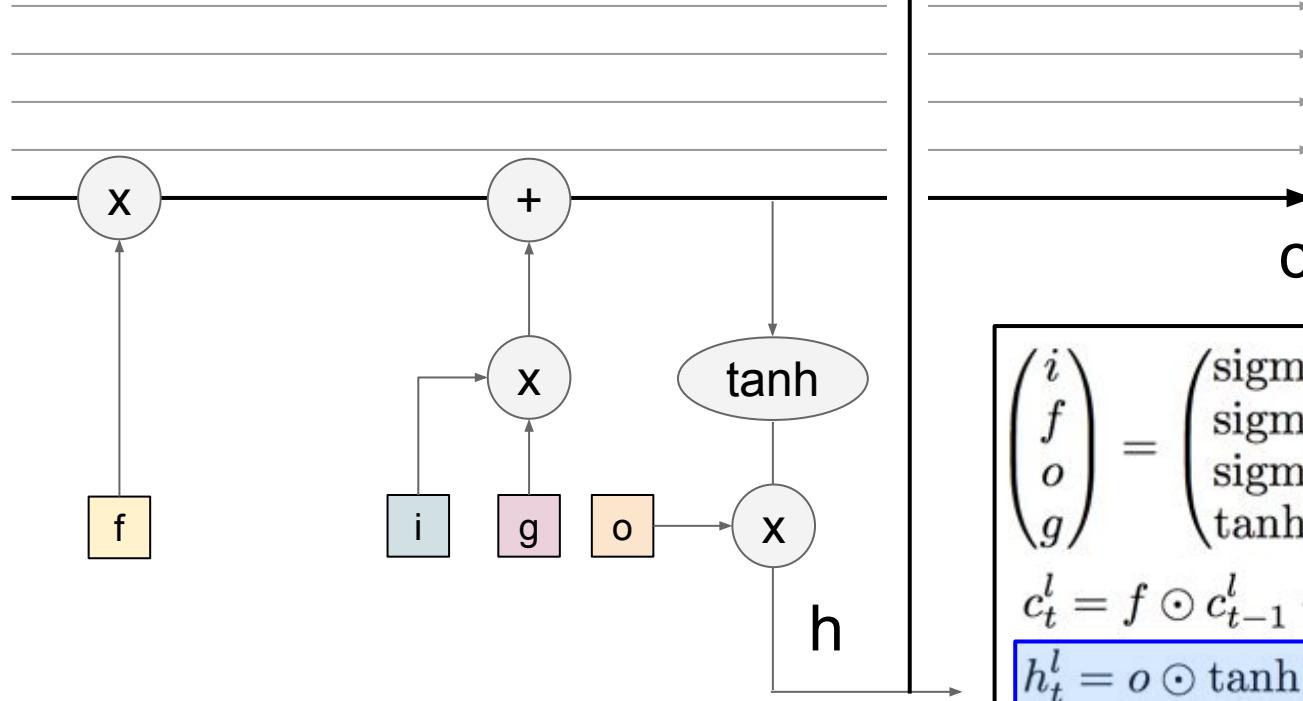


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c



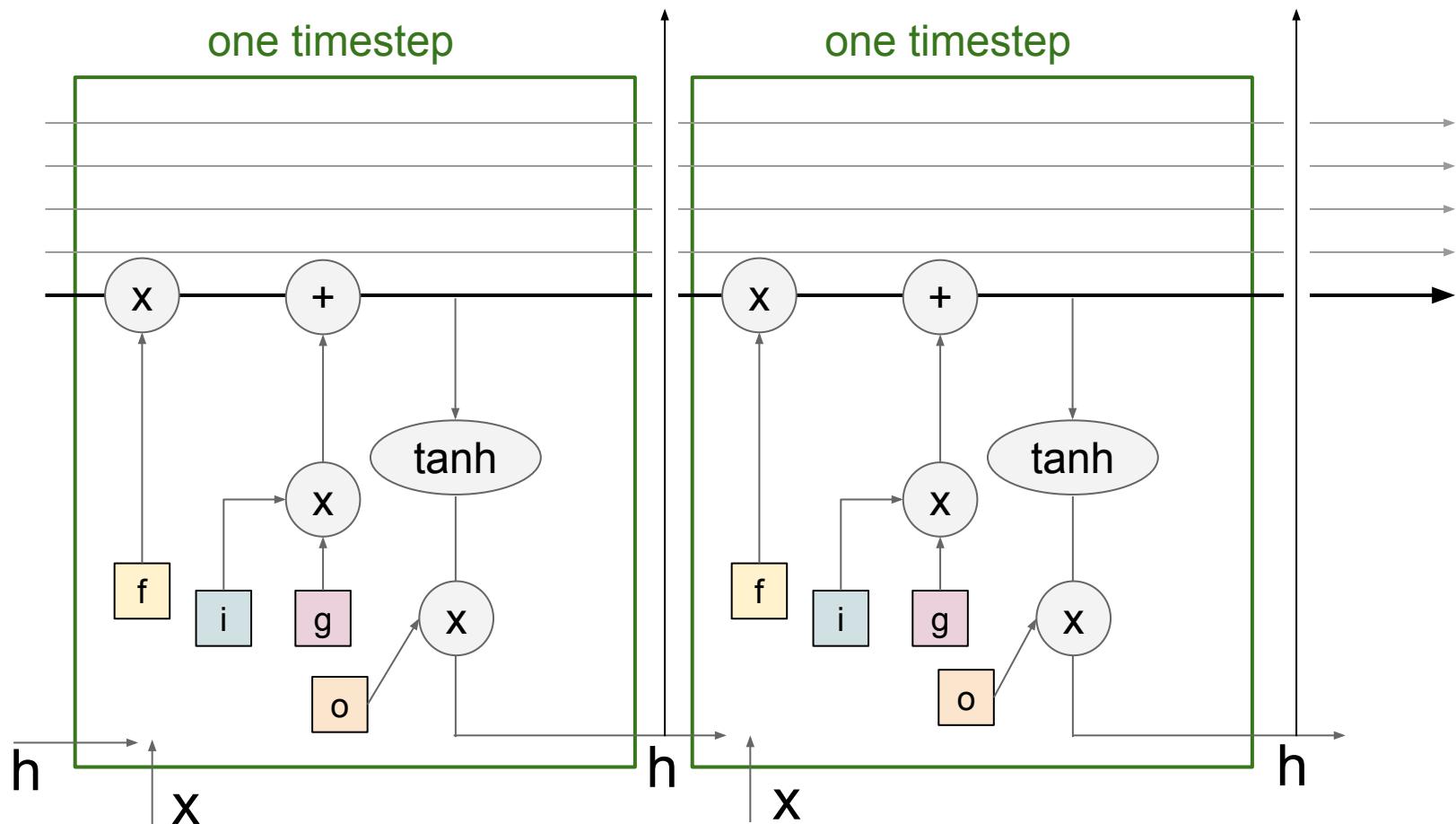
higher layer, or
prediction

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

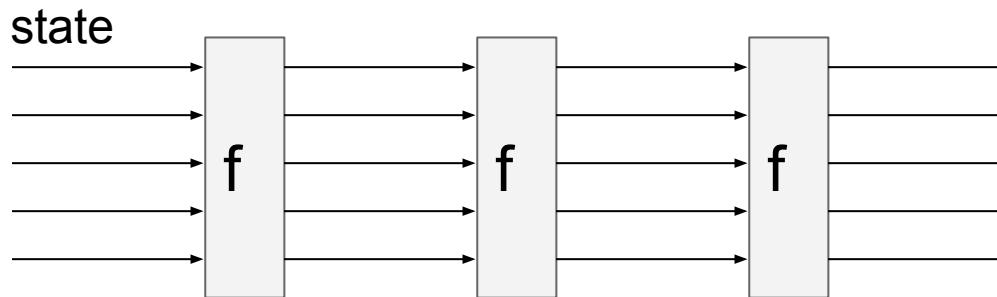
LSTM

one timestep

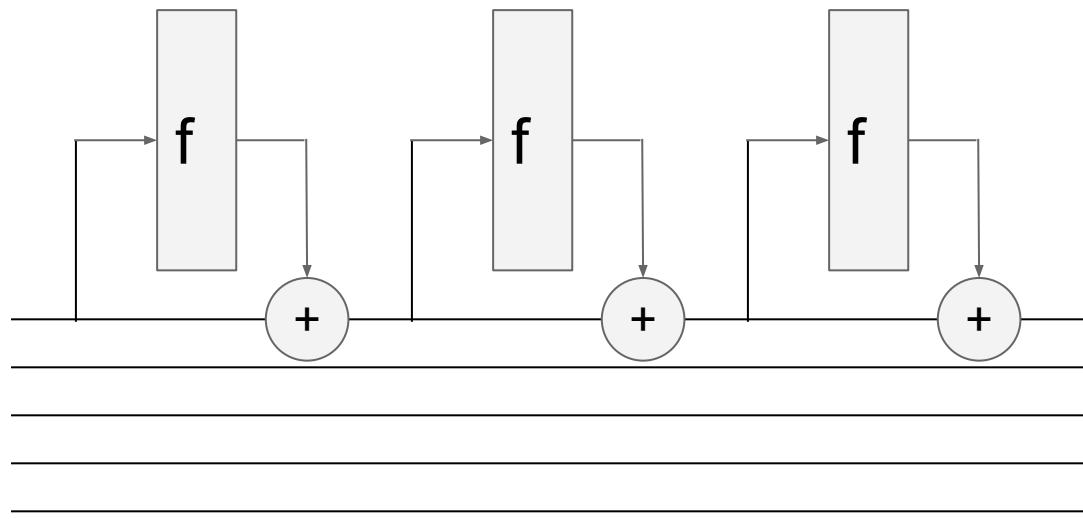
cell
state c



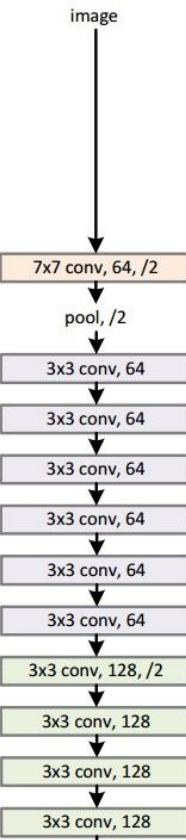
RNN



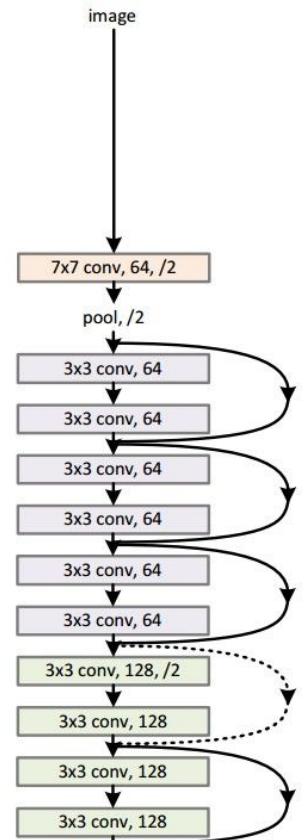
LSTM (ignoring forget gates)



34-layer plain

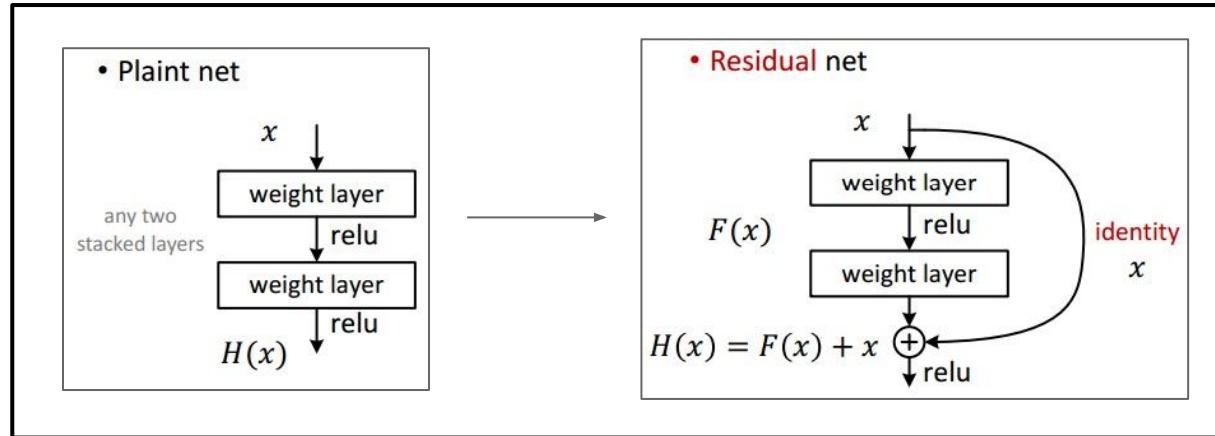


34-layer residual



Recall: “PlainNets” vs. ResNets

ResNet is to PlainNet what LSTM is to RNN, kind of.



Understanding gradient flow dynamics

Cute backprop signal video: <http://imgur.com/gallery/vaNahKE>

```
H = 5      # dimensionality of hidden state
T = 50     # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

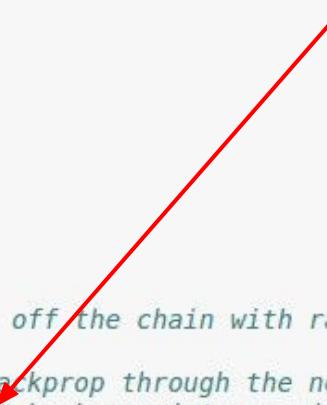
Understanding gradient flow dynamics

```
H = 5      # dimensionality of hidden state
T = 50     # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

if the largest eigenvalue is > 1 , gradient will explode
if the largest eigenvalue is < 1 , gradient will vanish



[On the difficulty of training Recurrent Neural Networks, Pascanu et al., 2013]

Understanding gradient flow dynamics

```
H = 5      # dimensionality of hidden state
T = 50     # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

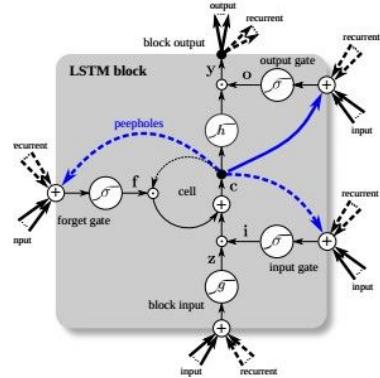
if the largest eigenvalue is > 1 , gradient will explode
if the largest eigenvalue is < 1 , gradient will vanish

can control exploding with gradient clipping
can control vanishing with LSTM

[On the difficulty of training Recurrent Neural Networks, Pascanu et al., 2013]

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:

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

MUT2:

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

MUT3:

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

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.