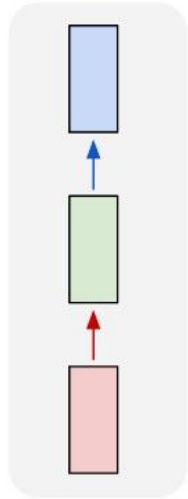
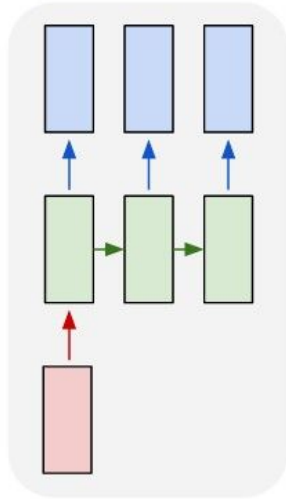


Recurrent Networks offer a lot of flexibility:

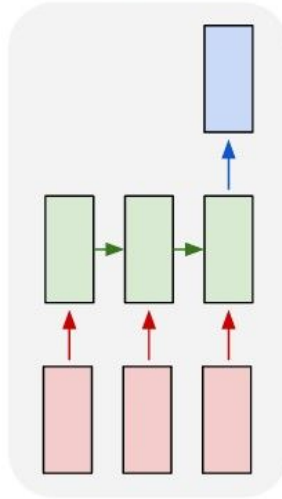
one to one



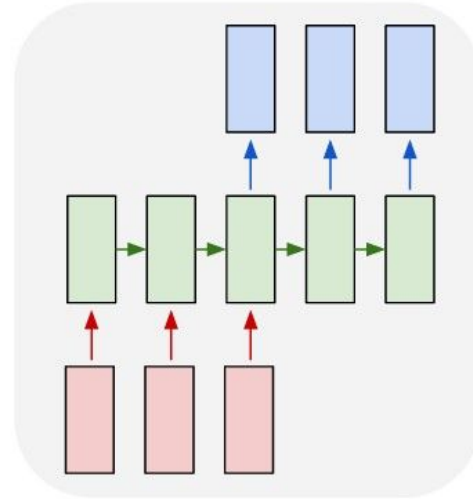
one to many



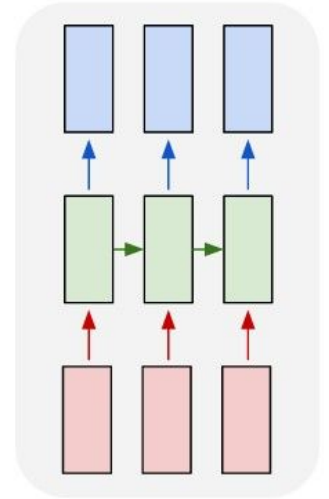
many to one



many to many



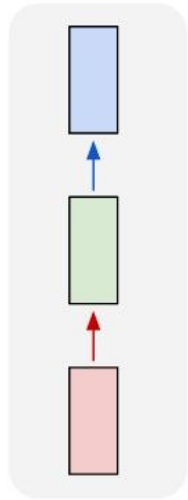
many to many



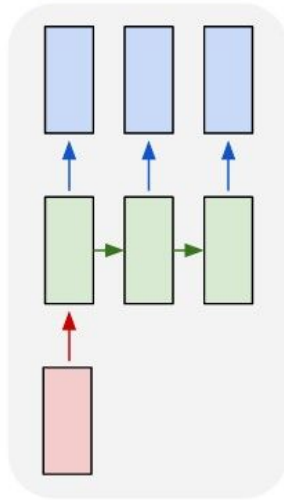
↖ **Vanilla Neural Networks**

Recurrent Networks offer a lot of flexibility:

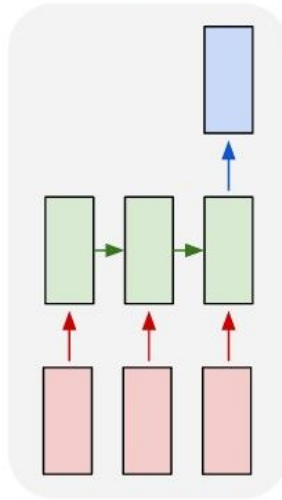
one to one



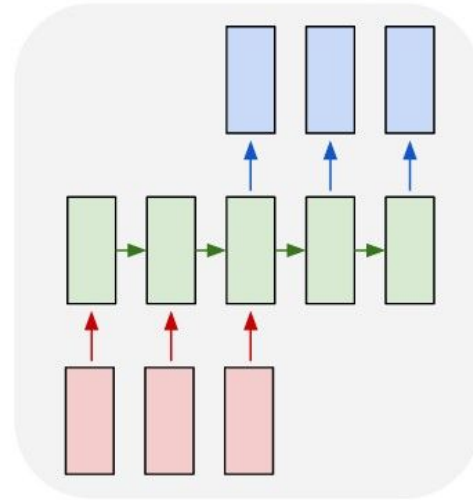
one to many



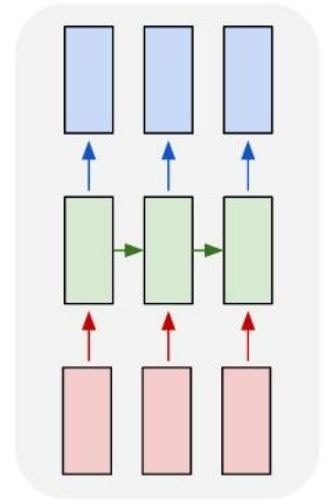
many to one



many to many



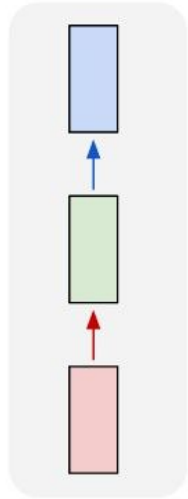
many to many



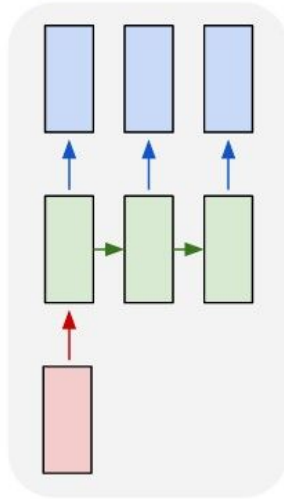
↖ e.g. **Image Captioning**
image -> sequence of words

Recurrent Networks offer a lot of flexibility:

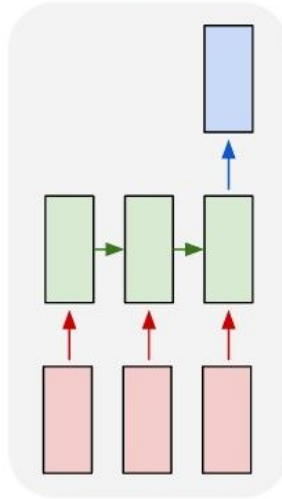
one to one



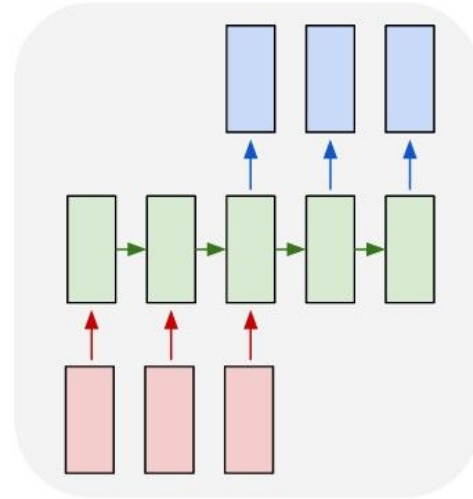
one to many



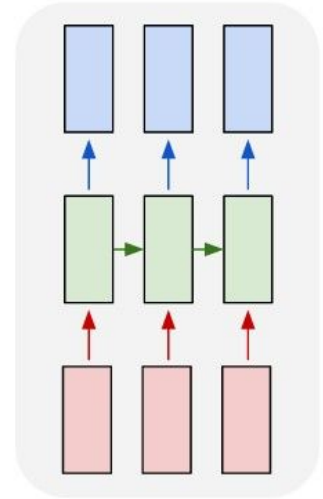
many to one



many to many



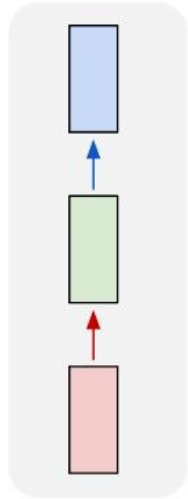
many to many



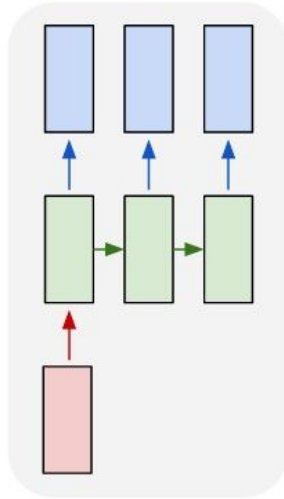
↖ e.g. **Sentiment Classification**
sequence of words → sentiment

Recurrent Networks offer a lot of flexibility:

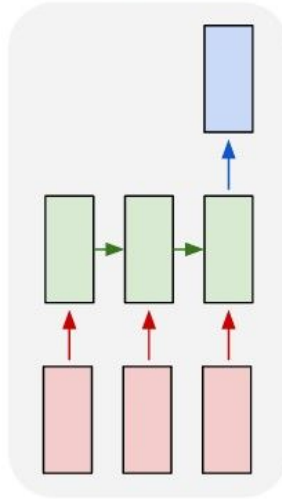
one to one



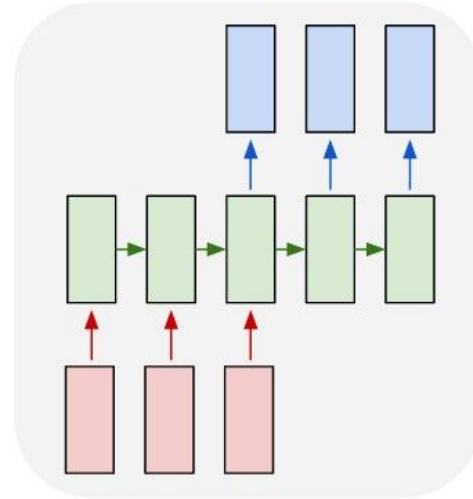
one to many



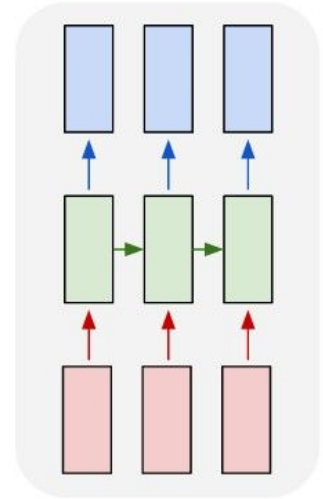
many to one



many to many



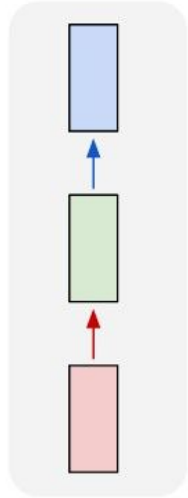
many to many



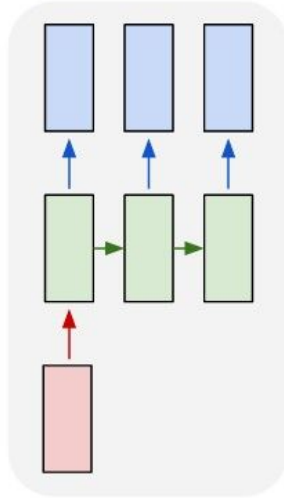
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Networks offer a lot of flexibility:

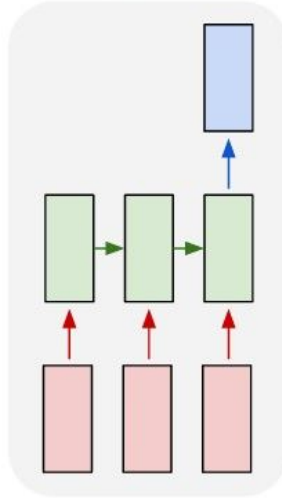
one to one



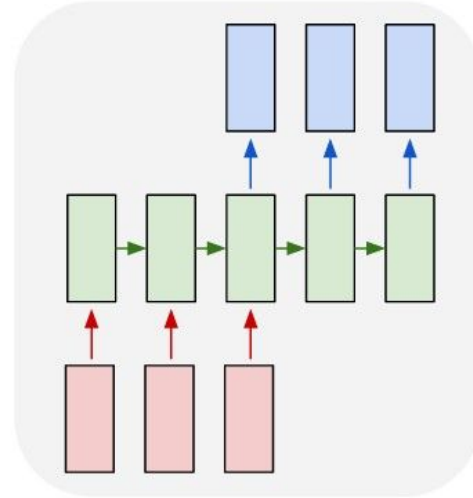
one to many



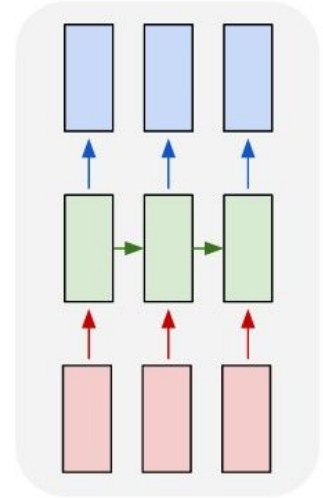
many to one



many to many



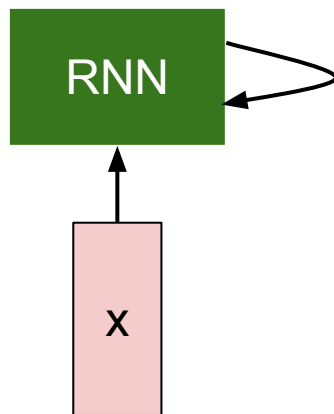
many to many



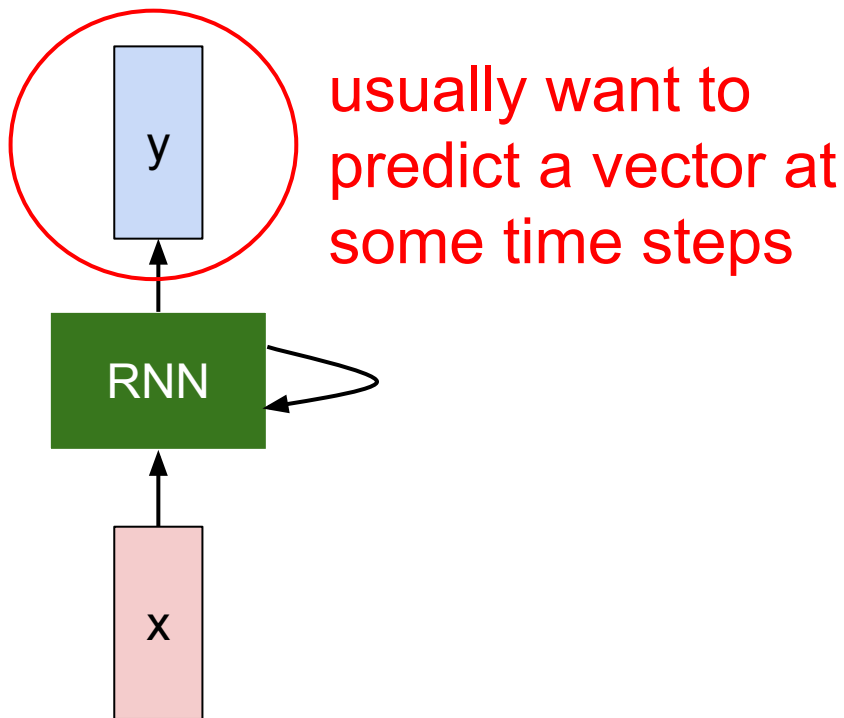
e.g. Video classification on frame level



Recurrent Neural Network



Recurrent Neural Network



Recurrent Neural Network

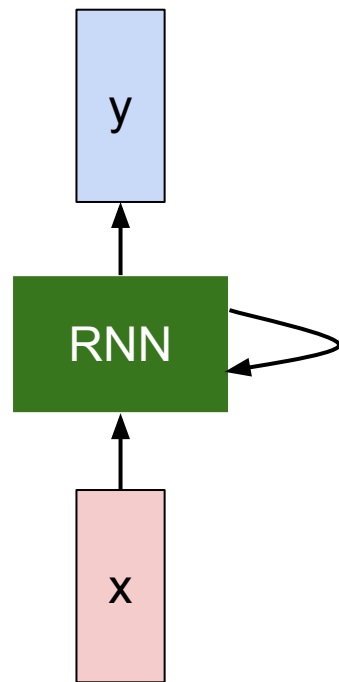
We can process a sequence of vectors \mathbf{x} by applying a recurrence formula at every time step:

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

new state / some function with parameters W

old state

input vector at some time step

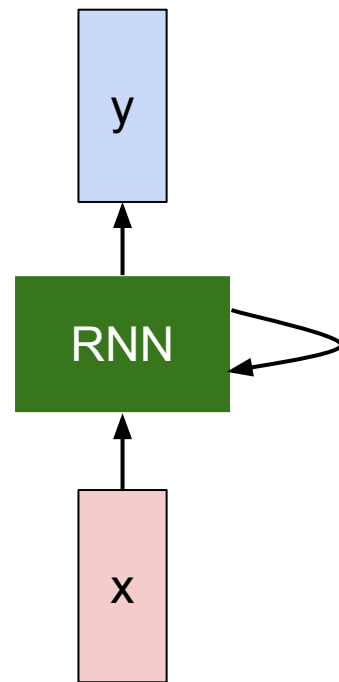


Recurrent Neural Network

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

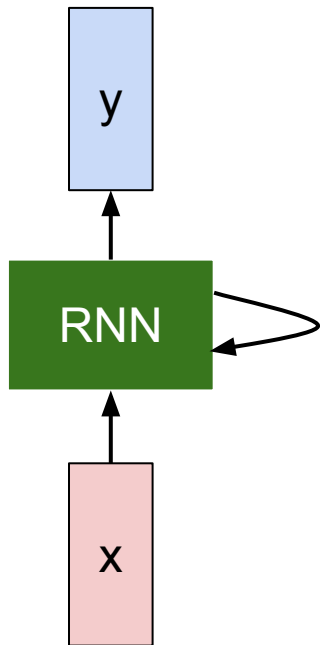
$$h_t = f_W(h_{t-1}, x_t)$$

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



(Vanilla) Recurrent Neural Network

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



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



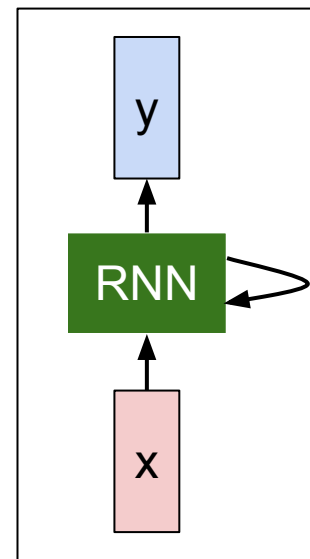
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Character-level language model example

Vocabulary:
[h,e,l,o]

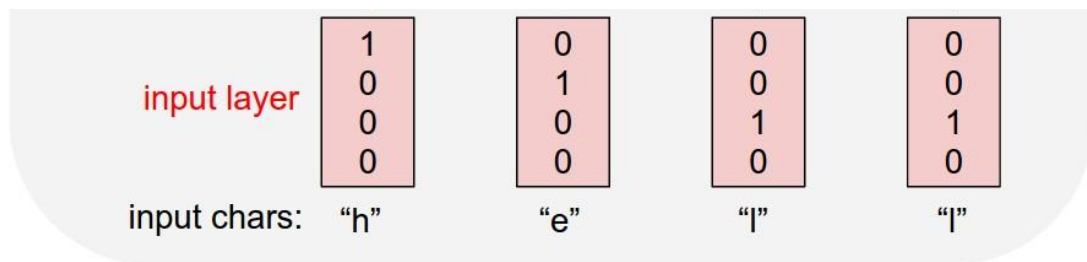
Example training
sequence:
“hello”



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

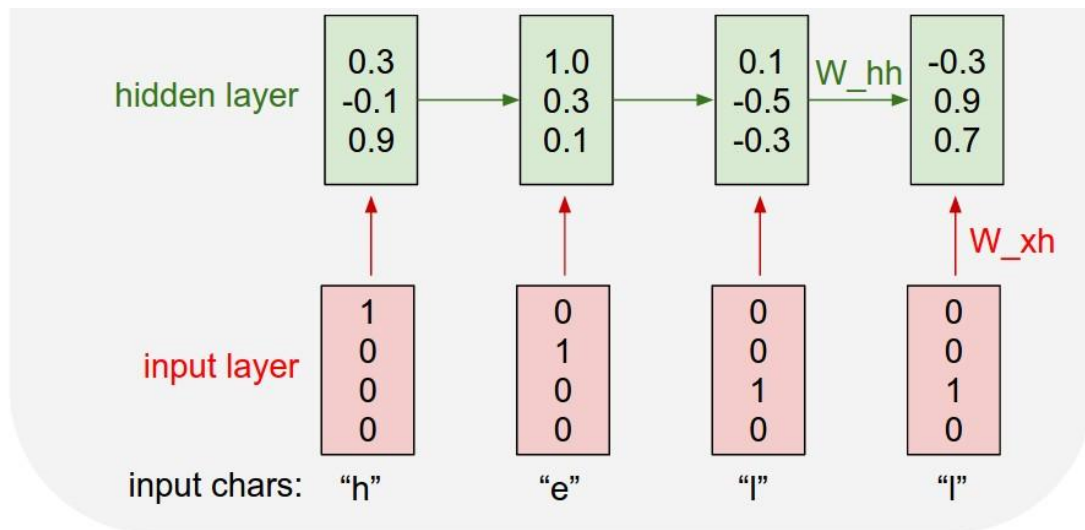


Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

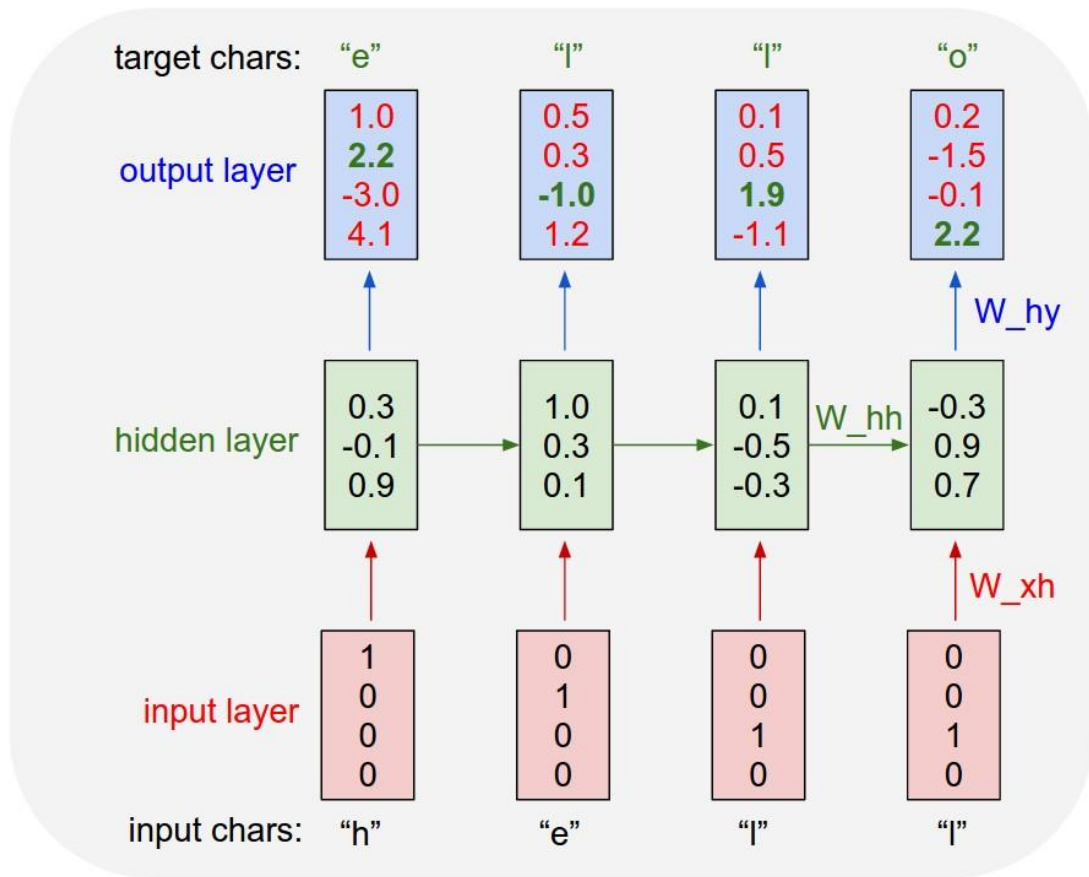
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



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

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

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

min-char-rnn.py gist

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

```
18 # Model parameters
19 wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
20 wh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
21 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
22 b_h = np.zeros(hidden_size, 1) # hidden bias
23 by = np.zeros(vocab_size, 1) # output bias
```

```
24 def lossfun(inputs, targets, hprev):
```

```
25     """
26     inputs, targets are both lists of integers.
27     hprev is list/array of initial hidden state
28     returns the loss, gradients on model parameters, and last hidden state
29     """
```

```
30     n_h, n_y, n_x = len(hprev), len(targets), len(inputs)
31     h0 = np.zeros(hidden_size, 1)
32     loss = 0
```

```
33     # Forward pass
34     for t in xrange(len(inputs)):
35         x[t] = np.zeros(vocab_size, 1) # encode in 1-of-K representation
36         x[t][inputs[t]] = 1
```

```
37         h[t] = np.tanh(np.dot(wh, x[t]) + np.dot(whh, h[t-1]) + bh) # hidden state
38         y[t] = np.dot(why, h[t]) + by # unnormalized log probabilities for next chars
39         p[t] = np.exp(y[t]) / np.sum(np.exp(y[t])) # probabilities for next chars
40         loss += -np.log(p[t][targets[t]]) # softmax (cross-entropy loss)
```

```
41     # Backward pass: compute gradients going backward
42     dwh, dwhh, dwhy = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
43     dh0, dh1 = np.zeros_like(h0), np.zeros_like(h1)
44     dhnext = np.zeros_like(h0)
```

```
45     for t in reversed(xrange(len(inputs))):
46         dy = np.zeros(vocab_size, 1)
47         dy[targets[t]] -= 1 # backprop into y
```

```
48         dhy = np.dot(dy, why)
49         dht = np.dot(dhy, h[t])
50         dht = np.dot(dht, h[t])
```

```
51         dh = np.dot(dht, T)
52         dhraw = (1 - h[t]**2) * h[t] # dh = backprop through tanh nonlinearity
53         dht = np.dot(dhraw, wh[t])
54         dwhh = np.dot(dhraw, whh[t])
```

```
55         dhnext = np.dot(dht, T)
56         dwh = np.dot(dh, wh)
57         dwhy = np.dot(dh, why)
58         return loss, dwh, dwhh, dwhy, dh, dy, h[inputs[t]-1]
```

```
59     # Sample from the model
60     ix = np.argmax(h[inputs[t]-1])
61     h = memory state, seed, ix is seed letter for first time step
```

```
62     # Sample from the model
63     ix = np.argmax(h[inputs[t]-1])
64     h = memory state, seed, ix is seed letter for first time step
```

```
65     # Sample from the model
66     ix = np.argmax(h[inputs[t]-1])
67     h = memory state, seed, ix is seed letter for first time step
```

```
68     # Sample from the model
69     ix = np.argmax(h[inputs[t]-1])
70     h = memory state, seed, ix is seed letter for first time step
```

```
71     # Sample from the model
72     ix = np.argmax(h[inputs[t]-1])
73     h = memory state, seed, ix is seed letter for first time step
```

```
74     # Sample from the model
75     ix = np.argmax(h[inputs[t]-1])
76     h = memory state, seed, ix is seed letter for first time step
```

```
77     # Sample from the model
78     ix = np.argmax(h[inputs[t]-1])
79     h = memory state, seed, ix is seed letter for first time step
```

```
80     # Sample from the model
81     ix = np.argmax(h[inputs[t]-1])
82     h = memory state, seed, ix is seed letter for first time step
```

```
83     # Sample from the model
84     ix = np.argmax(h[inputs[t]-1])
85     h = memory state, seed, ix is seed letter for first time step
```

```
86     # Sample from the model
87     ix = np.argmax(h[inputs[t]-1])
88     h = memory state, seed, ix is seed letter for first time step
```

```
89     # Sample from the model
90     ix = np.argmax(h[inputs[t]-1])
91     h = memory state, seed, ix is seed letter for first time step
```

Data I/O

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


min-char-rnn.py gist

```

#==
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
800 License
https://github.com/karpathy/minilm/blob/master/MiniLM_v1.py

import numpy as np

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

```

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

# model parameters
w_h = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
w_b = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
w_x = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
b_h = np.zeros((hidden_size, 1)) # hidden bias
b_x = np.zeros((vocab_size, 1)) # output bias
```

[illegible]

```

10     # sample a sequence of integers from the model
11     h = memory_state
12     ix = seed_letter for first time step
13
14     x = np.zeros((vocab_size, 1))
15     s[saved_ix] = 1
16     losses = []
17     for i in xrange(n):
18         h = np.tanh(np.dot(xh, x) + np.dot(wh, h) + bh)
19         y = np.dot(Wy, h) + by
20         p = np.exp(y) / np.sum(p)
21         ix = np.random.choice(range(vocab_size), p=p, randv=1)
22         x = np.zeros((vocab_size, 1))
23         s[ix] = 1
24     losses.append(ix)
25     return losses

```

```

5, p, 0, 0, 0
mnh, mnhb, mwhy = np.zeros_like(wmh), np.zeros_like(wmbh), np.zeros_like(wmby)
mnh, mnhb = np.zeros_like(hb), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1/('vocab_size')*seq_length) * loss at iteration 0
while True:
    # prepare inputs (we're sweeping from left to right in steps seq_length long)
    if p + seq_length-1 == len(data) or n == 0:
        # prep up np.zeros (len(data), 1) = reset mnh memory
        p = 0; n = seq_length
    inputs = [char to i, i] for ch in data[p:p+seq_length]
    targets = [char to i, i] for ch in data[p:p+seq_length-1]

```

```

94 # sample from the model now and then
95 if n % 100 == 0:
96     sample_ix = sample(hprev, inputs[ix], 200)
97     txt = ''.join(x.to_char(ix) for ix in sample_ix)
98     print '.....\n %s %s.....' % (txt, ix)
99
100 # forward seq length characters through the net and fetch gradient
101 loss, dseq, dlen, dwhy, ddb, dhy, dprev = lossFun(inputs, targets, hprev)
102 smooth_loss = smooth(loss, dseq, dlen, dwhy, ddb, dhy, dprev)
103 if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

```

```

05  # perform parameter update with Adagrad
06  for param, dparam, mem in zip([wch, whh, why, bn, by],
07                                [dwh, dwhh, dwhy, dbn, dby],
08                                [mchw, mchh, mwhy, mbn, mby]):
09      mem += dparam * dparam
10      param += -learning_rate * dparam / sp.sqrt(mem + 1e-8) # adagrad update
11
12  p += seq_length # move data pointer
13  b = 1 # iteration counter

```

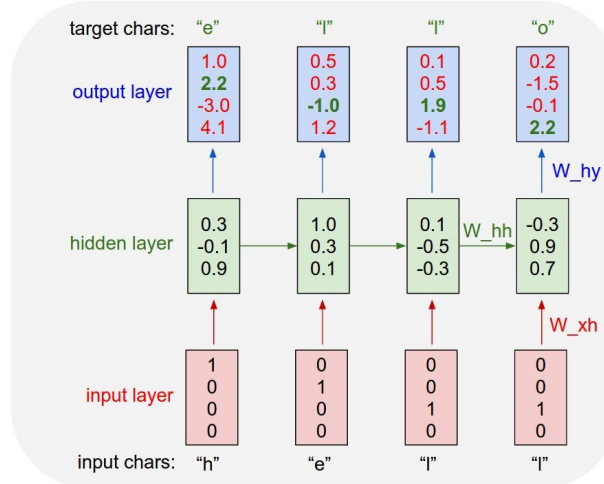
Initializations

```

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 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 b1 = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is vec array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     es, hs, ys, ps = [], [], [], []
34     h1 = hprev
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi = np.zeros((vocab_size, 1)) # encode in 1-of-K representation
39         xi[ix_to_char[t]] = 1
40         h1 = np.tanh(np.dot(wih, xi) + np.dot(whh, h1[-1]) + bh) # hidden state
41         yi = np.dot(why, h1) + by # unnormalized log probabilities for next chars
42         ps = np.exp(yi) / np.sum(np.exp(yi)) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backward
45     dht, dbh, dby = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
46     dht, dbh, dby = np.zeros_like(h1), np.zeros_like(h1), np.zeros_like(h1)
47     dhtest = np.zeros_like(h1[0])
48     for t in reversed(xrange(len(inputs))):
49         dy = np.copy(ps[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dht, dbh, dby = np.dot(dy, h1[t:T])
52         dy = dy
53         dh = np.dot(dht, dy) + dhtest # backprop into h
54         dhtest = (1 - h1[t] * h1[t]) * dh # backprop through tanh nonlinearity
55         dht, dbh, dby = np.dot(dht, draw, h1[t:T])
56         dht, dbh, dby = np.dot(dht, draw, h1[t:T])
57         dhtest = np.dot(dht, draw)
58     for dparam in [dht, dbh, dby, dby]:
59         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
60     return loss, dht, dbh, dby, dby, h1[len(inputs)-1]
61
62 def sample(h, ix):
63     """
64     sample a sequence of integers from the model
65     h is memory state, seed, ix is seed letter for first time step
66     """
67     x = np.zeros((vocab_size, 1))
68     ix = ix
69     ix = ix
70     ix = ix
71     ix = ix
72     ix = ix
73     ix = ix
74     ix = ix
75     ix = ix
76     ix = ix
77     ix = ix
78     ix = ix
79     ix = ix
80     ix = ix
81     ix = ix
82     ix = ix
83     ix = ix
84     ix = ix
85     ix = ix
86     ix = ix
87     ix = ix
88     ix = ix
89     ix = ix
90     ix = ix
91     ix = ix
92     ix = ix
93     ix = ix
94     ix = ix
95     ix = ix
96     ix = ix
97     ix = ix
98     ix = ix
99     ix = ix
100     ix = ix
101     ix = ix
102     ix = ix
103     ix = ix
104     ix = ix
105     ix = ix
106     ix = ix
107     ix = ix
108     ix = ix
109     ix = ix
110     ix = ix
111     ix = ix
112     ix = ix
113     ix = ix
114     ix = ix
115     ix = ix
116     ix = ix
117     ix = ix
118     ix = ix
119     ix = ix
120     ix = ix
121     ix = ix
122     ix = ix
123     ix = ix
124     ix = ix
125     ix = ix
126     ix = ix
127     ix = ix
128     ix = ix
129     ix = ix
130     ix = ix
131     ix = ix
132     ix = ix
133     ix = ix
134     ix = ix
135     ix = ix
136     ix = ix
137     ix = ix
138     ix = ix
139     ix = ix
140     ix = ix
141     ix = ix
142     ix = ix
143     ix = ix
144     ix = ix
145     ix = ix
146     ix = ix
147     ix = ix
148     ix = ix
149     ix = ix
150     ix = ix
151     ix = ix
152     ix = ix
153     ix = ix
154     ix = ix
155     ix = ix
156     ix = ix
157     ix = ix
158     ix = ix
159     ix = ix
160     ix = ix
161     ix = ix
162     ix = ix
163     ix = ix
164     ix = ix
165     ix = ix
166     ix = ix
167     ix = ix
168     ix = ix
169     ix = ix
170     ix = ix
171     ix = ix
172     ix = ix
173     ix = ix
174     ix = ix
175     ix = ix
176     ix = ix
177     ix = ix
178     ix = ix
179     ix = ix
180     ix = ix
181     ix = ix
182     ix = ix
183     ix = ix
184     ix = ix
185     ix = ix
186     ix = ix
187     ix = ix
188     ix = ix
189     ix = ix
190     ix = ix
191     ix = ix
192     ix = ix
193     ix = ix
194     ix = ix
195     ix = ix
196     ix = ix
197     ix = ix
198     ix = ix
199     ix = ix
200     ix = ix
201     ix = ix
202     ix = ix
203     ix = ix
204     ix = ix
205     ix = ix
206     ix = ix
207     ix = ix
208     ix = ix
209     ix = ix
210     ix = ix
211     ix = ix
212     ix = ix
213     ix = ix
214     ix = ix
215     ix = ix
216     ix = ix
217     ix = ix
218     ix = ix
219     ix = ix
220     ix = ix
221     ix = ix
222     ix = ix
223     ix = ix
224     ix = ix
225     ix = ix
226     ix = ix
227     ix = ix
228     ix = ix
229     ix = ix
230     ix = ix
231     ix = ix
232     ix = ix
233     ix = ix
234     ix = ix
235     ix = ix
236     ix = ix
237     ix = ix
238     ix = ix
239     ix = ix
240     ix = ix
241     ix = ix
242     ix = ix
243     ix = ix
244     ix = ix
245     ix = ix
246     ix = ix
247     ix = ix
248     ix = ix
249     ix = ix
250     ix = ix
251     ix = ix
252     ix = ix
253     ix = ix
254     ix = ix
255     ix = ix
256     ix = ix
257     ix = ix
258     ix = ix
259     ix = ix
260     ix = ix
261     ix = ix
262     ix = ix
263     ix = ix
264     ix = ix
265     ix = ix
266     ix = ix
267     ix = ix
268     ix = ix
269     ix = ix
270     ix = ix
271     ix = ix
272     ix = ix
273     ix = ix
274     ix = ix
275     ix = ix
276     ix = ix
277     ix = ix
278     ix = ix
279     ix = ix
280     ix = ix
281     ix = ix
282     ix = ix
283     ix = ix
284     ix = ix
285     ix = ix
286     ix = ix
287     ix = ix
288     ix = ix
289     ix = ix
290     ix = ix
291     ix = ix
292     ix = ix
293     ix = ix
294     ix = ix
295     ix = ix
296     ix = ix
297     ix = ix
298     ix = ix
299     ix = ix
300     ix = ix
301     ix = ix
302     ix = ix
303     ix = ix
304     ix = ix
305     ix = ix
306     ix = ix
307     ix = ix
308     ix = ix
309     ix = ix
310     ix = ix
311     ix = ix
312     ix = ix
313     ix = ix
314     ix = ix
315     ix = ix
316     ix = ix
317     ix = ix
318     ix = ix
319     ix = ix
320     ix = ix
321     ix = ix
322     ix = ix
323     ix = ix
324     ix = ix
325     ix = ix
326     ix = ix
327     ix = ix
328     ix = ix
329     ix = ix
330     ix = ix
331     ix = ix
332     ix = ix
333     ix = ix
334     ix = ix
335     ix = ix
336     ix = ix
337     ix = ix
338     ix = ix
339     ix = ix
340     ix = ix
341     ix = ix
342     ix = ix
343     ix = ix
344     ix = ix
345     ix = ix
346     ix = ix
347     ix = ix
348     ix = ix
349     ix = ix
350     ix = ix
351     ix = ix
352     ix = ix
353     ix = ix
354     ix = ix
355     ix = ix
356     ix = ix
357     ix = ix
358     ix = ix
359     ix = ix
360     ix = ix
361     ix = ix
362     ix = ix
363     ix = ix
364     ix = ix
365     ix = ix
366     ix = ix
367     ix = ix
368     ix = ix
369     ix = ix
370     ix = ix
371     ix = ix
372     ix = ix
373     ix = ix
374     ix = ix
375     ix = ix
376     ix = ix
377     ix = ix
378     ix = ix
379     ix = ix
380     ix = ix
381     ix = ix
382     ix = ix
383     ix = ix
384     ix = ix
385     ix = ix
386     ix = ix
387     ix = ix
388     ix = ix
389     ix = ix
390     ix = ix
391     ix = ix
392     ix = ix
393     ix = ix
394     ix = ix
395     ix = ix
396     ix = ix
397     ix = ix
398     ix = ix
399     ix = ix
400     ix = ix
401     ix = ix
402     ix = ix
403     ix = ix
404     ix = ix
405     ix = ix
406     ix = ix
407     ix = ix
408     ix = ix
409     ix = ix
410     ix = ix
411     ix = ix
412     ix = ix
413     ix = ix
414     ix = ix
415     ix = ix
416     ix = ix
417     ix = ix
418     ix = ix
419     ix = ix
420     ix = ix
421     ix = ix
422     ix = ix
423     ix = ix
424     ix = ix
425     ix = ix
426     ix = ix
427     ix = ix
428     ix = ix
429     ix = ix
430     ix = ix
431     ix = ix
432     ix = ix
433     ix = ix
434     ix = ix
435     ix = ix
436     ix = ix
437     ix = ix
438     ix = ix
439     ix = ix
440     ix = ix
441     ix = ix
442     ix = ix
443     ix = ix
444     ix = ix
445     ix = ix
446     ix = ix
447     ix = ix
448     ix = ix
449     ix = ix
450     ix = ix
451     ix = ix
452     ix = ix
453     ix = ix
454     ix = ix
455     ix = ix
456     ix = ix
457     ix = ix
458     ix = ix
459     ix = ix
460     ix = ix
461     ix = ix
462     ix = ix
463     ix = ix
464     ix = ix
465     ix = ix
466     ix = ix
467     ix = ix
468     ix = ix
469     ix = ix
470     ix = ix
471     ix = ix
472     ix = ix
473     ix = ix
474     ix = ix
475     ix = ix
476     ix = ix
477     ix = ix
478     ix = ix
479     ix = ix
480     ix = ix
481     ix = ix
482     ix = ix
483     ix = ix
484     ix = ix
485     ix = ix
486     ix = ix
487     ix = ix
488     ix = ix
489     ix = ix
490     ix = ix
491     ix = ix
492     ix = ix
493     ix = ix
494     ix = ix
495     ix = ix
496     ix = ix
497     ix = ix
498     ix = ix
499     ix = ix
500     ix = ix
501     ix = ix
502     ix = ix
503     ix = ix
504     ix = ix
505     ix = ix
506     ix = ix
507     ix = ix
508     ix = ix
509     ix = ix
510     ix = ix
511     ix = ix
512     ix = ix
513     ix = ix
514     ix = ix
515     ix = ix
516     ix = ix
517     ix = ix
518     ix = ix
519     ix = ix
520     ix = ix
521     ix = ix
522     ix = ix
523     ix = ix
524     ix = ix
525     ix = ix
526     ix = ix
527     ix = ix
528     ix = ix
529     ix = ix
530     ix = ix
531     ix = ix
532     ix = ix
533     ix = ix
534     ix = ix
535     ix = ix
536     ix = ix
537     ix = ix
538     ix = ix
539     ix = ix
540     ix = ix
541     ix = ix
542     ix = ix
543     ix = ix
544     ix = ix
545     ix = ix
546     ix = ix
547     ix = ix
548     ix = ix
549     ix = ix
550     ix = ix
551     ix = ix
552     ix = ix
553     ix = ix
554     ix = ix
555     ix = ix
556     ix = ix
557     ix = ix
558     ix = ix
559     ix = ix
560     ix = ix
561     ix = ix
562     ix = ix
563     ix = ix
564     ix = ix
565     ix = ix
566     ix = ix
567     ix = ix
568     ix = ix
569     ix = ix
570     ix = ix
571     ix = ix
572     ix = ix
573     ix = ix
574     ix = ix
575     ix = ix
576     ix = ix
577     ix = ix
578     ix = ix
579     ix = ix
580     ix = ix
581     ix = ix
582     ix = ix
583     ix = ix
584     ix = ix
585     ix = ix
586     ix = ix
587     ix = ix
588     ix = ix
589     ix = ix
590     ix = ix
591     ix = ix
592     ix = ix
593     ix = ix
594     ix = ix
595     ix = ix
596     ix = ix
597     ix = ix
598     ix = ix
599     ix = ix
600     ix = ix
601     ix = ix
602     ix = ix
603     ix = ix
604     ix = ix
605     ix = ix
606     ix = ix
607     ix = ix
608     ix = ix
609     ix = ix
610     ix = ix
611     ix = ix
612     ix = ix
613     ix = ix
614     ix = ix
615     ix = ix
616     ix = ix
617     ix = ix
618     ix = ix
619     ix = ix
620     ix = ix
621     ix = ix
622     ix = ix
623     ix = ix
624     ix = ix
625     ix = ix
626     ix = ix
627     ix = ix
628     ix = ix
629     ix = ix
630     ix = ix
631     ix = ix
632     ix = ix
633     ix = ix
634     ix = ix
635     ix = ix
636     ix = ix
637     ix = ix
638     ix = ix
639     ix = ix
640     ix = ix
641     ix = ix
642     ix = ix
643     ix = ix
644     ix = ix
645     ix = ix
646     ix = ix
647     ix = ix
648     ix = ix
649     ix = ix
650     ix = ix
651     ix = ix
652     ix = ix
653     ix = ix
654     ix = ix
655     ix = ix
656     ix = ix
657     ix = ix
658     ix = ix
659     ix = ix
660     ix = ix
661     ix = ix
662     ix = ix
663     ix = ix
664     ix = ix
665     ix = ix
666     ix = ix
667     ix = ix
668     ix = ix
669     ix = ix
670     ix = ix
671     ix = ix
672     ix = ix
673     ix = ix
674     ix = ix
675     ix = ix
676     ix = ix
677     ix = ix
678     ix = ix
679     ix = ix
680     ix = ix
681     ix = ix
682     ix = ix
683     ix = ix
684     ix = ix
685     ix = ix
686     ix = ix
687     ix = ix
688     ix = ix
689     ix = ix
690     ix = ix
691     ix = ix
692     ix = ix
693     ix = ix
694     ix = ix
695     ix = ix
696     ix = ix
697     ix = ix
698     ix = ix
699     ix = ix
700     ix = ix
701     ix = ix
702     ix = ix
703     ix = ix
704     ix = ix
705     ix = ix
706     ix = ix
707     ix = ix
708     ix = ix
709     ix = ix
710     ix = ix
711     ix = ix
712     ix = ix
713     ix = ix
714     ix = ix
715     ix = ix
716     ix = ix
717     ix = ix
718     ix = ix
719     ix = ix
720     ix = ix
721     ix = ix
722     ix = ix
723     ix = ix
724     ix = ix
725     ix = ix
726     ix = ix
727     ix = ix
728     ix = ix
729     ix = ix
730     ix = ix
731     ix = ix
732     ix = ix
733     ix = ix
734     ix = ix
735     ix = ix
736     ix = ix
737     ix = ix
738     ix = ix
739     ix = ix
740     ix = ix
741     ix = ix
742     ix = ix
743     ix = ix
744     ix = ix
745     ix = ix
746     ix = ix
747     ix = ix
748     ix = ix
749     ix = ix
750     ix = ix
751     ix = ix
752     ix = ix
753     ix = ix
754     ix = ix
755     ix = ix
756     ix = ix
757     ix = ix
758     ix = ix
759     ix = ix
760     ix = ix
761     ix = ix
762     ix = ix
763     ix = ix
764     ix = ix
765     ix = ix
766     ix = ix
767     ix = ix
768     ix = ix
769     ix = ix
770     ix = ix
771     ix = ix
772     ix = ix
773     ix = ix
774     ix = ix
775     ix = ix
776     ix = ix
777     ix = ix
778     ix = ix
779     ix = ix
780     ix = ix
781     ix = ix
782     ix = ix
783     ix = ix
784     ix = ix
785     ix = ix
786     ix = ix
787     ix = ix
788     ix = ix
789     ix = ix
790     ix = ix
791     ix = ix
792     ix = ix
793     ix = ix
794     ix = ix
795     ix = ix
796     ix = ix
797     ix = ix
798     ix = ix
799     ix = ix
800     ix = ix
801     ix = ix
802     ix = ix
803     ix = ix
804     ix = ix
805     ix = ix
806     ix = ix
807     ix = ix
808     ix = ix
809     ix = ix
810     ix = ix
811     ix = ix
812     ix = ix
813     ix = ix
814     ix = ix
815     ix = ix
816     ix = ix
817     ix = ix
818     ix = ix
819     ix = ix
820     ix = ix
821     ix = ix
822     ix = ix
823     ix = ix
824     ix = ix
825     ix = ix
826     ix = ix
827     ix = ix
828     ix = ix
829     ix = ix
830     ix = ix
831     ix = ix
832     ix = ix
833     ix = ix
834     ix = ix
835     ix = ix
836     ix = ix
837     ix = ix
838     ix = ix
839     ix = ix
840     ix = ix
841     ix = ix
842     ix = ix
843     ix = ix
844     ix = ix
845     ix = ix
846     ix = ix
847     ix = ix
848     ix = ix
849     ix = ix
850     ix = ix
851     ix = ix
852     ix = ix
853     ix = ix
854     ix = ix
855     ix = ix
856     ix = ix
857     ix = ix
858     ix = ix
859     ix = ix
860     ix = ix
861     ix = ix
862     ix = ix
863     ix = ix
864     ix = ix
865     ix = ix
866     ix = ix
867     ix = ix
868     ix = ix
869     ix = ix
870     ix = ix
871     ix = ix
872     ix = ix
873     ix = ix
874     ix = ix
875     ix = ix
876     ix = ix
877     ix = ix
878     ix = ix
879     ix = ix
880     ix = ix
881     ix = ix
882     ix = ix
883     ix = ix
884     ix = ix
885     ix = ix
886     ix = ix
887     ix = ix
888     ix = ix
889     ix = ix
890     ix = ix
891     ix = ix
892     ix = ix
893     ix = ix
894     ix = ix
895     ix = ix
896     ix = ix
897     ix = ix
898     ix = ix
899     ix = ix
900     ix = ix
901     ix = ix
902     ix = ix
903     ix = ix
904     ix = ix
905     ix = ix
906     ix = ix
907     ix = ix
908     ix = ix
909     ix = ix
910     ix = ix
911     ix = ix
912     ix = ix
913     ix = ix
914     ix = ix
915     ix = ix
916     ix = ix
917     ix = ix
918     ix = ix
919     ix = ix
920     ix = ix
921     ix = ix
922     ix = ix
923     ix = ix
924     ix = ix
925     ix = ix
926     ix = ix
927     ix = ix
928     ix = ix
929     ix = ix
930     ix = ix
931     ix = ix
932     ix = ix
933     ix = ix
934     ix = ix
935     ix = ix
936     ix = ix
937     ix = ix
938     ix = ix
939     ix = ix
940     ix = ix
941     ix = ix
942     ix = ix
943     ix = ix
944     ix = ix
945     ix = ix
946     ix = ix
947     ix = ix
948     ix = ix
949     ix = ix
950     ix = ix
951     ix = ix
952     ix = ix
953     ix = ix
954     ix = ix
955     ix = ix
956     ix = ix
957     ix = ix
958     ix = ix
959     ix = ix
960     ix = ix
961     ix = ix
962     ix = ix
963     ix = ix
964     ix = ix
965     ix = ix
966     ix = ix
967     ix = ix
968     ix = ix
969     ix = ix
970     ix = ix
971     ix = ix
972     ix = ix
973     ix = ix
974     ix = ix
975     ix = ix
976     ix = ix
977     ix = ix
978     ix = ix
979     ix = ix
980     ix = ix
981     ix = ix
982     ix = ix
983     ix = ix
984     ix = ix
985     ix = ix
986     ix = ix
987     ix = ix
988     ix = ix
989     ix = ix
990     ix = ix
991     ix = ix
992     ix = ix
993     ix = ix
994     ix = ix
995     ix = ix
996     ix = ix
997     ix = ix
998     ix = ix
999     ix = ix
1000     ix = ix
1001     ix = ix
1002     ix = ix
1003     ix = ix
1004     ix = ix
1005     ix = ix
1006     ix = ix
1007     ix = ix
1008     ix = ix
1009     ix = ix
1010     ix = ix
1011     ix = ix
1012     ix = ix
1013     ix = ix
1014     ix = ix
1015     ix = ix
1016     ix = ix
1017     ix = ix
1018     ix = ix
1019     ix = ix
1020     ix = ix
1021     ix = ix
1022     ix = ix
1023     ix = ix
1024     ix = ix
1025     ix = ix
1026     ix = ix
1027     ix = ix
1028     ix = ix
1029     ix = ix
1030     ix = ix
1031     ix = ix
1032     ix = ix
1033     ix = ix
1034     ix = ix
1035     ix = ix
1036     ix = ix
1037     ix = ix
1038     ix = ix
1039     ix = ix
1040     ix = ix
1041     ix = ix
1042     ix = ix
1043     ix = ix
1044     ix = ix
1045     ix = ix
1046     ix = ix
1047     ix = ix
1048     ix = ix
1049     ix = ix
1050     ix = ix
1051     ix = ix
1052     ix = ix
1053     ix = ix
1054     ix = ix
1055     ix = ix
1056     ix = ix
1057     ix = ix
1058     ix = ix
1059     ix = ix
1060     ix = ix
1061     ix = ix
1062     ix = ix
1063     ix = ix
1064     ix = ix
1065     ix = ix
1066     ix = ix
1067     ix = ix
1068     ix = ix
1069     ix = ix
1070     ix = ix
1071     ix = ix
1072     ix = ix
1073     ix = ix
1074     ix = ix
1075     ix = ix
1076     ix = ix
1077     ix = ix
1078     ix = ix
1079     ix = ix
1080     ix = ix
1081     ix = ix
1082     ix = ix
1083     ix = ix
1084     ix = ix
1085     ix = ix
1086     ix = ix
1087     ix = ix
1088     ix = ix
1089     ix = ix
1090     ix = ix
1091     ix = ix
1092     ix = ix
1093     ix = ix
1094     ix = ix
1095     ix = ix
1096     ix = ix
1097     ix = ix
1098     ix = ix
1099     ix = ix
1100     ix = ix
1101     ix = ix
1102     ix = ix
1103     ix = ix
1104     ix = ix
1105     ix = ix
1106     ix = ix
1107     ix = ix
1108     ix = ix
1109     ix = ix
1110     ix = ix
1111     ix = ix
1112     ix = ix
1113     ix = ix
1114     ix = ix
1115     ix = ix
1116     ix = ix
1117     ix = ix
1118     ix = ix
1119     ix = ix
1120     ix = ix
1121     ix = ix
1122     ix = ix
1123     ix = ix
1124     ix = ix
1125     ix = ix
1126     ix = ix
1127     ix = ix
1128     ix = ix
1129     ix = ix
1130     ix = ix
1131     ix = ix
1132     ix = ix
1133     ix = ix
1134     ix = ix
1135     ix = ix
1136     ix = ix
1137     ix = ix
1138     ix = ix
1139     ix = ix
1140     ix = ix
1141     ix = ix
1142     ix = ix
1143     ix = ix
1144     ix = ix
1145     ix = ix
1146     ix = ix
1147     ix = ix
1148     ix = ix
1149     ix = ix
1150     ix = ix
1151     ix = ix
1152     ix = ix
1153     ix = ix
1154     ix = ix
1155     ix = ix
1156     ix = ix
1157     ix = ix
1158     ix = ix
1159     ix = ix
1160     ix = ix
1161     ix = ix
1162     ix = ix
1163     ix = ix
1164     ix = ix
1165     ix = ix
1166     ix = ix
1167     ix = ix
1168     ix = ix
1169     ix = ix
1170     ix = ix
1171     ix = ix
1172     ix = ix
1173     ix = ix
1174     ix = ix
1175     ix = ix
1176     ix = ix
1177     ix = ix
1178     ix = ix
1179     ix = ix
1180     ix = ix
1181     ix = ix
1182     ix = ix
1183     ix = ix
1184     ix = ix
1185     ix = ix
1186     ix = ix
1187     ix = ix
1188     ix = ix
1189     ix = ix
1190     ix = ix
1191     ix = ix
1192     ix = ix
1193     ix = ix
1194     ix = ix
1195     ix = ix
1196     ix = ix
1197     ix = ix
1198     ix = ix
1199     ix = ix
1200     ix = ix
1201     ix = ix
1202     ix = ix
1203     ix = ix
1204     ix = ix
1205     ix = ix
1206     ix = ix
1207     ix = ix
1208     ix = ix
1209     ix = ix
1210     ix = ix
1211     ix = ix
1212     ix = ix
1213     ix = ix
1214     ix = ix
1215     ix = ix
1216     ix = ix
1217     ix = ix
1218     ix = ix
1219     ix = ix
1220     ix = ix
1221     ix = ix
1222     ix = ix
1223     ix = ix
1224     ix = ix
1225     ix = ix
1226     ix = ix
1227     ix = ix
1228     ix = ix
1229     ix = ix
1230     ix = ix
1231     ix = ix
1232     ix = ix
1233     ix = ix
1234     ix = ix
1235     ix = ix
1236     ix = ix
1237     ix = ix
1238     ix = ix
1239     ix = ix
1240     ix = ix
1241     ix = ix
1242     ix = ix
1243     ix = ix
1244     ix = ix
1245     ix = ix
1246     ix = ix
1247     ix = ix
1248     ix = ix
1249     ix = ix
1250     ix = ix
1251     ix = ix
1252     ix = ix
1253     ix = ix
1254     ix = ix
1255     ix = ix
1256     ix = ix
1257     ix = ix
1258     ix = ix
1259     ix = ix
1260     ix = ix
1261     ix = ix
1262     ix = ix
1263     ix = ix
1264     ix = ix
1265     ix = ix
1266     ix = ix
1267     ix = ix
1268     ix = ix
1269     ix = ix
1270     ix = ix
1271     ix = ix
1272     ix = ix
1273     ix = ix
1274     ix = ix
1275     ix = ix
1276     ix = ix
1277     ix = ix
1278     ix = ix
1279     ix = ix
1280     ix = ix
1281     ix = ix
1282     ix = ix
1283     ix = ix
1284     ix = ix
1285     ix = ix
1286     ix = ix
1287     ix = ix
1288     ix = ix
1289     ix = ix
1290     ix = ix
1291     ix = ix
1292     ix = ix
1293     ix = ix
1294     ix = ix
1295     ix = ix
1296     ix = ix
1297     ix = ix
1298     ix = ix
1299     ix = ix
1300     ix = ix
1301     ix = ix
1302     ix = ix
1303     ix = ix
1304     ix = ix
1305     ix = ix
1306     ix = ix
1307     ix = ix
1308     ix = ix
1309     ix = ix
1310     ix = ix
1311     ix = ix
1312     ix = ix
1313     ix = ix
1314     ix = ix
1315     ix = ix
1316     ix = ix
1317     ix = ix
1318     ix = ix
1319     ix = ix
1320     ix = ix
1321     ix = ix
1322     ix = ix
1323     ix = ix
1324     ix = ix
1325     ix = ix
1326     ix = ix
1327     ix = ix
1328     ix = ix
1329     ix = ix
1330     ix = ix
1331     ix = ix
1332     ix = ix
1333     ix = ix
1334     ix = ix
1335     ix = ix
1336     ix = ix
1337     ix = ix
1338     ix = ix
1339     ix = ix
1340     ix = ix
1341     ix = ix
1342     ix = ix
1343     ix = ix
1344     ix = ix
1345     ix = ix
1346     ix = ix
1347     ix = ix
1348     ix = ix
1349     ix = ix
1350     ix = ix
1351     ix = ix
1352     ix = ix
1353     ix = ix
1354     ix = ix
1355     ix = ix
1356     ix = ix
1357     ix = ix
1358     ix = ix
1359     ix = ix
1360     ix = ix
1361     ix = ix
1362     ix = ix
1363     ix = ix
1364     ix = ix
1365     ix = ix
1366     ix = ix
1367     ix = ix
1368     ix = ix
1369     ix = ix
1370     ix = ix
1371     ix = ix
1372     ix = ix
1373     ix = ix
1374     ix = ix
1375     ix = ix
1376     ix = ix
1377     ix = ix
1378     ix = ix
1379     ix = ix
1380     ix = ix
1381     ix = ix
1382     ix = ix
1383     ix = ix
1384     ix = ix
1385     ix = ix
1386     ix = ix
1387     ix = ix
1388     ix = ix
1389     ix = ix
1390     ix = ix
1391     ix = ix
1392     ix = ix
1393     ix = ix
1394     ix = ix
1395     ix = ix
1396     ix = ix
1397     ix = ix
1398     ix = ix
1399     ix = ix
1400     ix = ix
1401     ix = ix
1402     ix = ix
1403     ix = ix
1404     ix = ix
1405     ix = ix
1406     ix = ix
1407     ix = ix
1408     ix = ix
1409     ix = ix
1410     ix = ix
1411     ix = ix
1412     ix = ix
1413     ix = ix
1414     ix = ix
1415     ix = ix
1416     ix = ix
1417     ix = ix
1418     ix = ix
1419     ix = ix
1420     ix = ix
1421     ix = ix
1422     ix = ix
1423     ix = ix
1424     ix = ix
1425     ix = ix
1426     ix = ix
1427     ix = ix
1428     ix = ix
1429     ix = ix
1430     ix = ix
1431     ix = ix
14
```

min-char-rnn.py gist

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

Main loop

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


min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(data)
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique' % (data_size, vocab_size)
12 char_to_ix = { ch: i for i, ch in enumerate(chars) }
13 ix_to_char = { i: ch for i, ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 b1 = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is vec array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     es, hs, ys, ps = [], [], [], []
34     h1 = hprev
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi = np.zeros((vocab_size, 1)) # encode in 1-of-K representation
39         xi[ix_to_char[t]] = 1
40         h1 = np.tanh(np.dot(wih, xi) + np.dot(whh, h1[-1]) + bh) # hidden state
41         yi = np.dot(why, h1) + by # unnormalized log probabilities for next chars
42         pi = np.exp(yi) / np.sum(np.exp(yi)) # probabilities for next chars
43         loss += -np.log(pi[target[t], 0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backward
45     dwh, dwhh, dwhy = np.zeros_like(whh), np.zeros_like(whh), np.zeros_like(why)
46     dbh, dbh1, dbh2 = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(bh)
47     dhnxt = np.zeros_like(h1[0])
48     for t in reversed(xrange(len(inputs))):
49         dy = np.zeros_like(yi)
50         dy[target[t]] -= 1 # backprop into y
51         dbh1 = np.dot(dy, h1[t:T])
52         dy += dbh1
53         dh = np.dot(why, T, dy) + dhnxt # backprop into h
54         dhrw = (1 - h1[t] * h1[t]) * dh # backprop through tanh nonlinearity
55         dwh += np.dot(dhrw, xi[t:T])
56         dwhh += np.dot(dhrw, h1[t:T])
57         dhnxt = np.dot(whh, T, dhrw)
58         dparam = [dwh, dwhh, dwhy, dbh, dbh1, dbh2]
59         # clip dparam to [-5, 5, out-dparam] # clip to mitigate exploding gradients
60         return loss, dwh, dwhh, dwhy, dbh, dbh1, dbh2, h1[len(inputs)-1]
61
62 def sample(h, ix, ix1):
63     """
64     sample a sequence of integers from the model
65     h is memory state, seed, ix is seed letter for first time step
66     """
67     x = np.zeros((vocab_size, 1))
68     ix1 = ix
69     ix1 = ix1 + 1
70     for t in xrange(1):
71         xi = np.zeros((vocab_size, 1))
72         xi[ix1] = ix
73         xi[ix1] = ix
74         xi[ix1] = ix
75         xi[ix1] = ix
76         xi[ix1] = ix
77         xi[ix1] = ix
78         xi[ix1] = ix
79         xi[ix1] = ix
80         xi[ix1] = ix
81         xi[ix1] = ix
82         xi[ix1] = ix
83         xi[ix1] = ix
84         xi[ix1] = ix
85         xi[ix1] = ix
86         xi[ix1] = ix
87         xi[ix1] = ix
88         xi[ix1] = ix
89         xi[ix1] = ix
90         xi[ix1] = ix
91         xi[ix1] = ix
92         xi[ix1] = ix
93         xi[ix1] = ix
94         xi[ix1] = ix
95         xi[ix1] = ix
96         xi[ix1] = ix
97         xi[ix1] = ix
98         xi[ix1] = ix
99         xi[ix1] = ix
100         xi[ix1] = ix
101         xi[ix1] = ix
102         xi[ix1] = ix
103         xi[ix1] = ix
104         xi[ix1] = ix
105         xi[ix1] = ix
106         xi[ix1] = ix
107         xi[ix1] = ix
108         xi[ix1] = ix
109         xi[ix1] = ix
110         xi[ix1] = ix
111         xi[ix1] = ix
112         xi[ix1] = ix
113         xi[ix1] = ix
114         xi[ix1] = ix
115         xi[ix1] = ix
116         xi[ix1] = ix
117         xi[ix1] = ix
118         xi[ix1] = ix
119         xi[ix1] = ix
120         xi[ix1] = ix
121         xi[ix1] = ix
122         xi[ix1] = ix
123         xi[ix1] = ix
124         xi[ix1] = ix
125         xi[ix1] = ix
126         xi[ix1] = ix
127         xi[ix1] = ix
128         xi[ix1] = ix
129         xi[ix1] = ix
130         xi[ix1] = ix
131         xi[ix1] = ix
132         xi[ix1] = ix
133         xi[ix1] = ix
134         xi[ix1] = ix
135         xi[ix1] = ix
136         xi[ix1] = ix
137         xi[ix1] = ix
138         xi[ix1] = ix
139         xi[ix1] = ix
140         xi[ix1] = ix
141         xi[ix1] = ix
142         xi[ix1] = ix
143         xi[ix1] = ix
144         xi[ix1] = ix
145         xi[ix1] = ix
146         xi[ix1] = ix
147         xi[ix1] = ix
148         xi[ix1] = ix
149         xi[ix1] = ix
150         xi[ix1] = ix
151         xi[ix1] = ix
152         xi[ix1] = ix
153         xi[ix1] = ix
154         xi[ix1] = ix
155         xi[ix1] = ix
156         xi[ix1] = ix
157         xi[ix1] = ix
158         xi[ix1] = ix
159         xi[ix1] = ix
160         xi[ix1] = ix
161         xi[ix1] = ix
162         xi[ix1] = ix
163         xi[ix1] = ix
164         xi[ix1] = ix
165         xi[ix1] = ix
166         xi[ix1] = ix
167         xi[ix1] = ix
168         xi[ix1] = ix
169         xi[ix1] = ix
170         xi[ix1] = ix
171         xi[ix1] = ix
172         xi[ix1] = ix
173         xi[ix1] = ix
174         xi[ix1] = ix
175         xi[ix1] = ix
176         xi[ix1] = ix
177         xi[ix1] = ix
178         xi[ix1] = ix
179         xi[ix1] = ix
180         xi[ix1] = ix
181         xi[ix1] = ix
182         xi[ix1] = ix
183         xi[ix1] = ix
184         xi[ix1] = ix
185         xi[ix1] = ix
186         xi[ix1] = ix
187         xi[ix1] = ix
188         xi[ix1] = ix
189         xi[ix1] = ix
190         xi[ix1] = ix
191         xi[ix1] = ix
192         xi[ix1] = ix
193         xi[ix1] = ix
194         xi[ix1] = ix
195         xi[ix1] = ix
196         xi[ix1] = ix
197         xi[ix1] = ix
198         xi[ix1] = ix
199         xi[ix1] = ix
200         xi[ix1] = ix
201         xi[ix1] = ix
202         xi[ix1] = ix
203         xi[ix1] = ix
204         xi[ix1] = ix
205         xi[ix1] = ix
206         xi[ix1] = ix
207         xi[ix1] = ix
208         xi[ix1] = ix
209         xi[ix1] = ix
210         xi[ix1] = ix
211         xi[ix1] = ix
212         xi[ix1] = ix
213         xi[ix1] = ix
214         xi[ix1] = ix
215         xi[ix1] = ix
216         xi[ix1] = ix
217         xi[ix1] = ix
218         xi[ix1] = ix
219         xi[ix1] = ix
220         xi[ix1] = ix
221         xi[ix1] = ix
222         xi[ix1] = ix
223         xi[ix1] = ix
224         xi[ix1] = ix
225         xi[ix1] = ix
226         xi[ix1] = ix
227         xi[ix1] = ix
228         xi[ix1] = ix
229         xi[ix1] = ix
230         xi[ix1] = ix
231         xi[ix1] = ix
232         xi[ix1] = ix
233         xi[ix1] = ix
234         xi[ix1] = ix
235         xi[ix1] = ix
236         xi[ix1] = ix
237         xi[ix1] = ix
238         xi[ix1] = ix
239         xi[ix1] = ix
240         xi[ix1] = ix
241         xi[ix1] = ix
242         xi[ix1] = ix
243         xi[ix1] = ix
244         xi[ix1] = ix
245         xi[ix1] = ix
246         xi[ix1] = ix
247         xi[ix1] = ix
248         xi[ix1] = ix
249         xi[ix1] = ix
250         xi[ix1] = ix
251         xi[ix1] = ix
252         xi[ix1] = ix
253         xi[ix1] = ix
254         xi[ix1] = ix
255         xi[ix1] = ix
256         xi[ix1] = ix
257         xi[ix1] = ix
258         xi[ix1] = ix
259         xi[ix1] = ix
260         xi[ix1] = ix
261         xi[ix1] = ix
262         xi[ix1] = ix
263         xi[ix1] = ix
264         xi[ix1] = ix
265         xi[ix1] = ix
266         xi[ix1] = ix
267         xi[ix1] = ix
268         xi[ix1] = ix
269         xi[ix1] = ix
270         xi[ix1] = ix
271         xi[ix1] = ix
272         xi[ix1] = ix
273         xi[ix1] = ix
274         xi[ix1] = ix
275         xi[ix1] = ix
276         xi[ix1] = ix
277         xi[ix1] = ix
278         xi[ix1] = ix
279         xi[ix1] = ix
280         xi[ix1] = ix
281         xi[ix1] = ix
282         xi[ix1] = ix
283         xi[ix1] = ix
284         xi[ix1] = ix
285         xi[ix1] = ix
286         xi[ix1] = ix
287         xi[ix1] = ix
288         xi[ix1] = ix
289         xi[ix1] = ix
290         xi[ix1] = ix
291         xi[ix1] = ix
292         xi[ix1] = ix
293         xi[ix1] = ix
294         xi[ix1] = ix
295         xi[ix1] = ix
296         xi[ix1] = ix
297         xi[ix1] = ix
298         xi[ix1] = ix
299         xi[ix1] = ix
300         xi[ix1] = ix
301         xi[ix1] = ix
302         xi[ix1] = ix
303         xi[ix1] = ix
304         xi[ix1] = ix
305         xi[ix1] = ix
306         xi[ix1] = ix
307         xi[ix1] = ix
308         xi[ix1] = ix
309         xi[ix1] = ix
310         xi[ix1] = ix
311         xi[ix1] = ix
312         xi[ix1] = ix
313         xi[ix1] = ix
314         xi[ix1] = ix
315         xi[ix1] = ix
316         xi[ix1] = ix
317         xi[ix1] = ix
318         xi[ix1] = ix
319         xi[ix1] = ix
320         xi[ix1] = ix
321         xi[ix1] = ix
322         xi[ix1] = ix
323         xi[ix1] = ix
324         xi[ix1] = ix
325         xi[ix1] = ix
326         xi[ix1] = ix
327         xi[ix1] = ix
328         xi[ix1] = ix
329         xi[ix1] = ix
330         xi[ix1] = ix
331         xi[ix1] = ix
332         xi[ix1] = ix
333         xi[ix1] = ix
334         xi[ix1] = ix
335         xi[ix1] = ix
336         xi[ix1] = ix
337         xi[ix1] = ix
338         xi[ix1] = ix
339         xi[ix1] = ix
340         xi[ix1] = ix
341         xi[ix1] = ix
342         xi[ix1] = ix
343         xi[ix1] = ix
344         xi[ix1] = ix
345         xi[ix1] = ix
346         xi[ix1] = ix
347         xi[ix1] = ix
348         xi[ix1] = ix
349         xi[ix1] = ix
350         xi[ix1] = ix
351         xi[ix1] = ix
352         xi[ix1] = ix
353         xi[ix1] = ix
354         xi[ix1] = ix
355         xi[ix1] = ix
356         xi[ix1] = ix
357         xi[ix1] = ix
358         xi[ix1] = ix
359         xi[ix1] = ix
360         xi[ix1] = ix
361         xi[ix1] = ix
362         xi[ix1] = ix
363         xi[ix1] = ix
364         xi[ix1] = ix
365         xi[ix1] = ix
366         xi[ix1] = ix
367         xi[ix1] = ix
368         xi[ix1] = ix
369         xi[ix1] = ix
370         xi[ix1] = ix
371         xi[ix1] = ix
372         xi[ix1] = ix
373         xi[ix1] = ix
374         xi[ix1] = ix
375         xi[ix1] = ix
376         xi[ix1] = ix
377         xi[ix1] = ix
378         xi[ix1] = ix
379         xi[ix1] = ix
380         xi[ix1] = ix
381         xi[ix1] = ix
382         xi[ix1] = ix
383         xi[ix1] = ix
384         xi[ix1] = ix
385         xi[ix1] = ix
386         xi[ix1] = ix
387         xi[ix1] = ix
388         xi[ix1] = ix
389         xi[ix1] = ix
390         xi[ix1] = ix
391         xi[ix1] = ix
392         xi[ix1] = ix
393         xi[ix1] = ix
394         xi[ix1] = ix
395         xi[ix1] = ix
396         xi[ix1] = ix
397         xi[ix1] = ix
398         xi[ix1] = ix
399         xi[ix1] = ix
400         xi[ix1] = ix
401         xi[ix1] = ix
402         xi[ix1] = ix
403         xi[ix1] = ix
404         xi[ix1] = ix
405         xi[ix1] = ix
406         xi[ix1] = ix
407         xi[ix1] = ix
408         xi[ix1] = ix
409         xi[ix1] = ix
410         xi[ix1] = ix
411         xi[ix1] = ix
412         xi[ix1] = ix
413         xi[ix1] = ix
414         xi[ix1] = ix
415         xi[ix1] = ix
416         xi[ix1] = ix
417         xi[ix1] = ix
418         xi[ix1] = ix
419         xi[ix1] = ix
420         xi[ix1] = ix
421         xi[ix1] = ix
422         xi[ix1] = ix
423         xi[ix1] = ix
424         xi[ix1] = ix
425         xi[ix1] = ix
426         xi[ix1] = ix
427         xi[ix1] = ix
428         xi[ix1] = ix
429         xi[ix1] = ix
430         xi[ix1] = ix
431         xi[ix1] = ix
432         xi[ix1] = ix
433         xi[ix1] = ix
434         xi[ix1] = ix
435         xi[ix1] = ix
436         xi[ix1] = ix
437         xi[ix1] = ix
438         xi[ix1] = ix
439         xi[ix1] = ix
440         xi[ix1] = ix
441         xi[ix1] = ix
442         xi[ix1] = ix
443         xi[ix1] = ix
444         xi[ix1] = ix
445         xi[ix1] = ix
446         xi[ix1] = ix
447         xi[ix1] = ix
448         xi[ix1] = ix
449         xi[ix1] = ix
450         xi[ix1] = ix
451         xi[ix1] = ix
452         xi[ix1] = ix
453         xi[ix1] = ix
454         xi[ix1] = ix
455         xi[ix1] = ix
456         xi[ix1] = ix
457         xi[ix1] = ix
458         xi[ix1] = ix
459         xi[ix1] = ix
460         xi[ix1] = ix
461         xi[ix1] = ix
462         xi[ix1] = ix
463         xi[ix1] = ix
464         xi[ix1] = ix
465         xi[ix1] = ix
466         xi[ix1] = ix
467         xi[ix1] = ix
468         xi[ix1] = ix
469         xi[ix1] = ix
470         xi[ix1] = ix
471         xi[ix1] = ix
472         xi[ix1] = ix
473         xi[ix1] = ix
474         xi[ix1] = ix
475         xi[ix1] = ix
476         xi[ix1] = ix
477         xi[ix1] = ix
478         xi[ix1] = ix
479         xi[ix1] = ix
480         xi[ix1] = ix
481         xi[ix1] = ix
482         xi[ix1] = ix
483         xi[ix1] = ix
484         xi[ix1] = ix
485         xi[ix1] = ix
486         xi[ix1] = ix
487         xi[ix1] = ix
488         xi[ix1] = ix
489         xi[ix1] = ix
490         xi[ix1] = ix
491         xi[ix1] = ix
492         xi[ix1] = ix
493         xi[ix1] = ix
494         xi[ix1] = ix
495         xi[ix1] = ix
496         xi[ix1] = ix
497         xi[ix1] = ix
498         xi[ix1] = ix
499         xi[ix1] = ix
500         xi[ix1] = ix
501         xi[ix1] = ix
502         xi[ix1] = ix
503         xi[ix1] = ix
504         xi[ix1] = ix
505         xi[ix1] = ix
506         xi[ix1] = ix
507         xi[ix1] = ix
508         xi[ix1] = ix
509         xi[ix1] = ix
510         xi[ix1] = ix
511         xi[ix1] = ix
512         xi[ix1] = ix
513         xi[ix1] = ix
514         xi[ix1] = ix
515         xi[ix1] = ix
516         xi[ix1] = ix
517         xi[ix1] = ix
518         xi[ix1] = ix
519         xi[ix1] = ix
520         xi[ix1] = ix
521         xi[ix1] = ix
522         xi[ix1] = ix
523         xi[ix1] = ix
524         xi[ix1] = ix
525         xi[ix1] = ix
526         xi[ix1] = ix
527         xi[ix1] = ix
528         xi[ix1] = ix
529         xi[ix1] = ix
530         xi[ix1] = ix
531         xi[ix1] = ix
532         xi[ix1] = ix
533         xi[ix1] = ix
534         xi[ix1] = ix
535         xi[ix1] = ix
536         xi[ix1] = ix
537         xi[ix1] = ix
538         xi[ix1] = ix
539         xi[ix1] = ix
540         xi[ix1] = ix
541         xi[ix1] = ix
542         xi[ix1] = ix
543         xi[ix1] = ix
544         xi[ix1] = ix
545         xi[ix1] = ix
546         xi[ix1] = ix
547         xi[ix1] = ix
548         xi[ix1] = ix
549         xi[ix1] = ix
550         xi[ix1] = ix
551         xi[ix1] = ix
552         xi[ix1] = ix
553         xi[ix1] = ix
554         xi[ix1] = ix
555         xi[ix1] = ix
556         xi[ix1] = ix
557         xi[ix1] = ix
558         xi[ix1] = ix
559         xi[ix1] = ix
560         xi[ix1] = ix
561         xi[ix1] = ix
562         xi[ix1] = ix
563         xi[ix1] = ix
564         xi[ix1] = ix
565         xi[ix1] = ix
566         xi[ix1] = ix
567         xi[ix1] = ix
568         xi[ix1] = ix
569         xi[ix1] = ix
570         xi[ix1] = ix
571         xi[ix1] = ix
572         xi[ix1] = ix
573         xi[ix1] = ix
574         xi[ix1] = ix
575         xi[ix1] = ix
576         xi[ix1] = ix
577         xi[ix1] = ix
578         xi[ix1] = ix
579         xi[ix1] = ix
580         xi[ix1] = ix
581         xi[ix1] = ix
582         xi[ix1] = ix
583         xi[ix1] = ix
584         xi[ix1] = ix
585         xi[ix1] = ix
586         xi[ix1] = ix
587         xi[ix1] = ix
588         xi[ix1] = ix
589         xi[ix1] = ix
590         xi[ix1] = ix
591         xi[ix1] = ix
592         xi[ix1] = ix
593         xi[ix1] = ix
594         xi[ix1] = ix
595         xi[ix1] = ix
596         xi[ix1] = ix
597         xi[ix1] = ix
598         xi[ix1] = ix
599         xi[ix1] = ix
600         xi[ix1] = ix
601         xi[ix1] = ix
602         xi[ix1] = ix
603         xi[ix1] = ix
604         xi[ix1] = ix
605         xi[ix1] = ix
606         xi[ix1] = ix
607         xi[ix1] = ix
608         xi[ix1] = ix
609         xi[ix1] = ix
610         xi[ix1] = ix
611         xi[ix1] = ix
612         xi[ix1] = ix
613         xi[ix1] = ix
614         xi[ix1] = ix
615         xi[ix1] = ix
616         xi[ix1] = ix
617         xi[ix1] = ix
618         xi[ix1] = ix
619         xi[ix1] = ix
620         xi[ix1] = ix
621         xi[ix1] = ix
622         xi[ix1] = ix
623         xi[ix1] = ix
624         xi[ix1] = ix
625         xi[ix1] = ix
626         xi[ix1] = ix
627         xi[ix1] = ix
628         xi[ix1] = ix
629         xi[ix1] = ix
630         xi[ix1] = ix
631         xi[ix1] = ix
632         xi[ix1] = ix
633         xi[ix1] = ix
634         xi[ix1] = ix
635         xi[ix1] = ix
636         xi[ix1] = ix
637         xi[ix1] = ix
638         xi[ix1] = ix
639         xi[ix1] = ix
640         xi[ix1] = ix
641         xi[ix1] = ix
642         xi[ix1] = ix
643         xi[ix1] = ix
644         xi[ix1] = ix
645         xi[ix1] = ix
646         xi[ix1] = ix
647         xi[ix1] = ix
648         xi[ix1] = ix
649         xi[ix1] = ix
650         xi[ix1] = ix
651         xi[ix1] = ix
652         xi[ix1] = ix
653         xi[ix1] = ix
654         xi[ix1] = ix
655         xi[ix1] = ix
656         xi[ix1] = ix
657         xi[ix1] = ix
658         xi[ix1] = ix
659         xi[ix1] = ix
660         xi[ix1] = ix
661         xi[ix1] = ix
662         xi[ix1] = ix
663         xi[ix1] = ix
664         xi[ix1] = ix
665         xi[ix1] = ix
666         xi[ix1] = ix
667         xi[ix1] = ix
668         xi[ix1] = ix
669         xi[ix1] = ix
670         xi[ix1] = ix
671         xi[ix1] = ix
672         xi[ix1] = ix
673         xi[ix1] = ix
674         xi[ix1] = ix
675         xi[ix1] = ix
676         xi[ix1] = ix
677         xi[ix1] = ix
678         xi[ix1] = ix
679         xi[ix1] = ix
680         xi[ix1] = ix
681         xi[ix1] = ix
682         xi[ix1] = ix
683         xi[ix1] = ix
684         xi[ix1] = ix
685         xi[ix1] = ix
686         xi[ix1] = ix
687         xi[ix1] = ix
688         xi[ix1] = ix
689         xi[ix1] = ix
690         xi[ix1] = ix
691         xi[ix1] = ix
692         xi[ix1] = ix
693         xi[ix1] = ix
694         xi[ix1] = ix
695         xi[ix1] = ix
696         xi[ix1] = ix
697         xi[ix1] = ix
698         xi[ix1] = ix
699         xi[ix1] = ix
700         xi[ix1] = ix
701         xi[ix1] = ix
702         xi[ix1] = ix
703         xi[ix1] = ix
704         xi[ix1] = ix
705         xi[ix1] = ix
706         xi[ix1] = ix
707         xi[ix1] = ix
708         xi[ix1] = ix
709         xi[ix1] = ix
710         xi[ix1] = ix
711         xi[ix1] = ix
712         xi[ix1] = ix
713         xi[ix1] = ix
714         xi[ix1] = ix
715         xi[ix1] = ix
716         xi[ix1] = ix
717         xi[ix1] = ix
718         xi[ix1] = ix
719         xi[ix1] = ix
720         xi[ix1] = ix
721         xi[ix1] = ix
722         xi[ix1] = ix
723         xi[ix1] = ix
724         xi[ix1] = ix
725         xi[ix1] = ix
726         xi[ix1] = ix
727         xi[ix1] = ix
728         xi[ix1] = ix
729         xi[ix1] = ix
730         xi[ix1] = ix
731         xi[ix1] = ix
732         xi[ix1] = ix
733         xi[ix1] = ix
734         xi[ix1] = ix
735         xi[ix1] = ix
736         xi[ix1] = ix
737         xi[ix1] = ix
738         xi[ix1] = ix
739         xi[ix1] = ix
740         xi[ix1] = ix
741         xi[ix1] = ix
742         xi[ix1] = ix
743         xi[ix1] = ix
744         xi[ix1] = ix
745         xi[ix1] = ix
746         xi[ix1] = ix
747         xi[ix1] = ix
748         xi[ix1] = ix
749         xi[ix1] = ix
750         xi[ix1] = ix
751         xi[ix1] = ix
752         xi[ix1] = ix
753         xi[ix1] = ix
754         xi[ix1] = ix
755         xi[ix1] = ix
756         xi[ix1] = ix
757         xi[ix1] = ix
758         xi[ix1] = ix
759         xi[ix1] = ix
760         xi[ix1] = ix
761         xi[ix1] = ix
762         xi[ix1] = ix
763         xi[ix1] = ix
764         xi[ix1] = ix
765         xi[ix1] = ix
766         xi[ix1] = ix
767         xi[ix1] = ix
768         xi[ix1] = ix
769         xi[ix1] = ix
770         xi[ix1] = ix
771         xi[ix1] = ix
772         xi[ix1] = ix
773         xi[ix1] = ix
774         xi[ix1] = ix
775         xi[ix1] = ix
776         xi[ix1] = ix
777         xi[ix1] = ix
778         xi[ix1] = ix
779         xi[ix1] = ix
780         xi[ix1] = ix
781         xi[ix1] = ix
782         xi[ix1] = ix
783         xi[ix1] = ix
784         xi[ix1] = ix
785         xi[ix1] = ix
786         xi[ix1] = ix
787         xi[ix1] = ix
788         xi[ix1] = ix
789         xi[ix1] = ix
790         xi[ix1] = ix
791         xi[ix1] = ix
792         xi[ix1] = ix
793         xi[ix1] = ix
794         xi[ix1] = ix
795         xi[ix1] = ix
796         xi[ix1] = ix
797         xi[ix1] = ix
798         xi[ix1] = ix
799         xi[ix1] = ix
800         xi[ix1] = ix
801         xi[ix1] = ix
802         xi[ix1] = ix
803         xi[ix1] = ix
804         xi[ix1] = ix
805         xi[ix1] = ix
806         xi[ix1] = ix
807         xi[ix1] = ix
808         xi[ix1] = ix
809         xi[ix1] = ix
810         xi[ix1] = ix
811         xi[ix1] = ix
812         xi[ix1] = ix
813         xi[ix1] = ix
814         xi[ix1] = ix
815         xi[ix1] = ix
816         xi[ix1] = ix
817         xi[ix1] = ix
818         xi[ix1] = ix
819         xi[ix1] = ix
820         xi[ix1] = ix
821         xi[ix1] = ix
822         xi[ix1] = ix
823         xi[ix1] = ix
824         xi[ix1] = ix
825         xi[ix1] = ix
826         xi[ix1] = ix
827         xi[ix1] = ix
828         xi[ix1] = ix
829         xi[ix1] = ix
830         xi[ix1] = ix
831         xi[ix1] = ix
832         xi[ix1] = ix
833         xi[ix1] = ix
834         xi[ix1] = ix
835         xi[ix1] = ix
836         xi[ix1] = ix
837         xi[ix1] = ix
838         xi[ix1] = ix
839         xi[ix1] = ix
840         xi[ix1] = ix
841         xi[ix1] = ix
842         xi[ix1] = ix
843         xi[ix1] = ix
844         xi[ix1] = ix
845         xi[ix1] = ix
846         xi[ix1] = ix
847         xi[ix1] = ix
848         xi[ix1] = ix
849         xi[ix1] = ix
850         xi[ix1] = ix
851         xi[ix1] = ix
852         xi[ix1] = ix
853         xi[ix1] = ix
854         xi[ix1] = ix
855         xi[ix1] = ix
856         xi[ix1] = ix
857         xi[ix1] = ix
858         xi[ix1] = ix
859         xi[ix1] = ix
860         xi[ix1] = ix
861         xi[ix1] = ix
862         xi[ix1] = ix
863         xi[ix1] = ix
864         xi[ix1] = ix
865         xi[ix1] = ix
866         xi[ix1] = ix
867         xi[ix1] = ix
868         xi[ix1] = ix
869         xi[ix1] = ix
870         xi[ix1] = ix
871         xi[ix1] = ix
872         xi[ix1] = ix
873         xi[ix1] = ix
874         xi[ix1] = ix
875         xi[ix1] = ix
876         xi[ix1] = ix
877         xi[ix1] = ix
878         xi[ix1] = ix
879         xi[ix1] = ix
880         xi[ix1] = ix
881         xi[ix1] = ix
882         xi[ix1] = ix
883         xi[ix1] = ix
884         xi[ix1] = ix
885         xi[ix1] = ix
886         xi[ix1] = ix
887         xi[ix1] = ix
888         xi[ix1] = ix
889         xi[ix1] = ix
890         xi[ix1] = ix
891         xi[ix1] = ix
892         xi[ix1] = ix
893         xi[ix1] = ix
894         xi[ix1] = ix
895         xi[ix1] = ix
896         xi[ix1] = ix
897         xi[ix1] = ix
898         xi[ix1] = ix
899         xi[ix1] = ix
900         xi[ix1] = ix
901         xi[ix1] = ix
902         xi[ix1] = ix
903         xi[ix1] = ix
904         xi[ix1] = ix
905         xi[ix1] = ix
906         xi[ix1] = ix
907         xi[ix1] = ix
908         xi[ix1] = ix
909         xi[ix1] = ix
910         xi[ix1] = ix
911         xi[ix1] = ix
912         xi[ix1] = ix
913         xi[ix1] = ix
914         xi[ix1] = ix
915         xi[ix1] = ix
916         xi[ix1] = ix
917         xi[ix1] = ix
918         xi[ix1] = ix
919         xi[ix1] = ix
920         xi[ix1] = ix
921         xi[ix1] = ix
922         xi[ix1] = ix
923         xi[ix1] = ix
924         xi[ix1] = ix
925         xi[ix1] = ix
926         xi[ix1] = ix
927         xi[ix1] = ix
928         xi[ix1] = ix
929         xi[ix1] = ix
930         xi[ix1] = ix
931         xi[ix1] = ix
932         xi[ix1] = ix
933         xi[ix1] = ix
934         xi[ix1] = ix
935         xi[ix1] = ix
936         xi[ix1] = ix
937         xi[ix1] = ix
938         xi[ix1] = ix
939         xi[ix1] = ix
940         xi[ix1] = ix
941         xi[ix1] = ix
942         xi[ix1] = ix
943         xi[ix1] = ix
944         xi[ix1] = ix
945         xi[ix1] = ix
946         xi[ix1] = ix
947         xi[ix1] = ix
948         xi[ix1] = ix
949         xi[ix1] = ix
950         xi[ix1] = ix
951         xi[ix1] = ix
952         xi[ix1] = ix
953         xi[ix1] = ix
954         xi[ix1] = ix
955         xi[ix1] = ix
956         xi[ix1] = ix
957         xi[ix1] = ix
958         xi[ix1] = ix
959         xi[ix1] = ix
960         xi[ix1] = ix
961         xi[ix1] = ix
962         xi[ix1] = ix
963         xi[ix1] = ix
964         xi[ix1] = ix
965         xi[ix1] = ix
966         xi[ix1] = ix
967         xi[ix1] = ix
968         xi[ix1] = ix
969         xi[ix1] = ix
970         xi[ix1] = ix
971         xi[ix1] = ix
972         xi[ix1] = ix
973         xi[ix1] = ix
974         xi[ix1] = ix
975         xi[ix1] = ix
976         xi[ix1] = ix
977         xi[ix1] = ix
978         xi[ix1] = ix
979         xi[ix1] = ix
980         xi[ix1] = ix
981         xi[ix1] = ix
982         xi[ix1] = ix
983         xi[ix1] = ix
984         xi[ix1] = ix
985         xi[ix1] = ix
986         xi[ix1] = ix
987         xi[ix1] = ix
988         xi[ix1] = ix
989         xi[ix1] = ix
990         xi[ix1] = ix
991         xi[ix1] = ix
992         xi[ix1] = ix
993         xi[ix1] = ix
994         xi[ix1] = ix
995         xi[ix1] = ix
996         xi[ix1] = ix
997         xi[ix1] = ix
998         xi[ix1] = ix
999         xi[ix1] = ix
1000        """
```

Main loop

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

min-char-rnn.py gist

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

Main loop

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

[illegible]

```
# perform parameter update with Adagrad
for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                              [dwxh, dwhh, dwhy, dbh, dby],
                              [mwxh, mwhh, mwhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
```

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


min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data 370
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 wh = 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
```

```
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is Nx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36
37     # forward pass
38     for t in xrange(len(inputs)):
39         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
40         xs[t][inputs[t]] = 1
41         hs[t] = np.tanh(np.dot(whx, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
42         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
43         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
44         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
45
46     # backward pass: compute gradients going backwards
47     dhs, dwh, dwhy, dbh, dby = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
48     dhs, dwh, dwhy, dbh, dby = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
49     dhnext = np.zeros_like(hs[0])
50
51     for t in reversed(xrange(len(inputs))):
52         dy = np.copy(ps[t])
53         dy[targets[t]] -= 1 # backprop into y
54         dby += np.dot(dy, hs[t].T)
55         dhs += dy
56         dh = np.dot(why.T, dy) + dhnext # backprop into h
57         dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
58         dbh += dhraw
59         dwhx += np.dot(dhraw, xs[t].T)
60         dwh += np.dot(dhraw, hs[t].T)
61         dwhy += np.dot(dh, dhs[t])
62         for dparam in [dwhx, dwhh, dwhy, dbh, dby]:
63             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
64     return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
```

```
65 def sample(next_ix, n):
66     """
67     sample a sequence of integers from the model
68     h is memory state, seed_ix is seed letter for first time step
69     """
70     x = np.zeros((vocab_size, 1))
71     ix_to_ix = [0]
72     for t in xrange(n):
73         x = np.zeros_like(x)
74         x[inputs[t]] = 1
75         y = np.dot(why, x) + by
76         p = np.exp(y) / np.sum(np.exp(y))
77         ix = np.random.choice(vocab_size, p=p, size=1)
78         x[ix] = 1
79         ix_to_ix.append(ix)
80     return ix_to_ix
81
82 # p = 0.0
83 mwh, mwhh, mwhy = np.zeros_like(wh), 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(0.000001) # loss at iteration 0
86 while True:
87     # generate inputs (we're sampling from left to right in steps seq_length)
88     if p == seq_length:
89         p = 0
90         hprev = np.zeros((hidden_size, 1)) # reset mem memory
91         p = 0 # p is 0 from start of data
92         inputs = [char_to_ix[ch] for ch in data[p:seq_length]]
93         targets = [char_to_ix[ch] for ch in data[p+1:seq_length+1]]
94
95     # sample from the model now and then
96     if n % 100 == 0:
97         sample_ix = sample(hprev, inputs[0], 200)
98         txt = ''.join(chars[ix] for ix in sample_ix)
99         print "-----%d chars-----" % len(txt)
100
101     # forward seq_length characters through the net and fetch gradients
102     loss, dwhx, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
103     smooth_loss = smooth_loss * 0.99 + loss * 0.01
104     if n % 1000 == 0: print "iter %d, loss %f" % (n, smooth_loss) # print progress
105
106     # perform parameter update with Adagrad
107     for param, dparam in zip([dwh, whh, why, bh, by], [dwhx, dwhh, dwhy, dbh, dby]):
108         mwh, mwhh, mwhy, mbh, mby = (mwh + dparam, mwhh + dparam, mwhy + dparam, mbh + dparam, mby + dparam)
109         param -= learning_rate * dparam / np.sqrt(mwh + mwhh + mwhy + mbh + mby) # adagrad update
110
111     p += seq_length # move data pointer
112     n += 1 # iteration counter
```

Loss function

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

```
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is Nx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36
37     # forward pass
38     for t in xrange(len(inputs)):
39         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
40         xs[t][inputs[t]] = 1
41         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
42         ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
43         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
44         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
45
46     # backward pass: compute gradients going backwards
47     dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
48     dbh, dby = np.zeros_like(bh), np.zeros_like(by)
49     dhnext = np.zeros_like(hs[0])
50
51     for t in reversed(xrange(len(inputs))):
52         dy = np.copy(ps[t])
53         dy[targets[t]] -= 1 # backprop into y
54         dwhy += np.dot(dy, hs[t].T)
55         dby += dy
56         dh = np.dot(Why.T, dy) + dhnext # backprop into h
57         dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
58         dbh += dhraw
59         dWxh += np.dot(dhraw, xs[t].T)
60         dWhh += np.dot(dhraw, hs[t-1].T)
61         dhnext = np.dot(WWh.T, dhraw)
62
63     for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
64         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
65
66     return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

min-char-rnn.py gist

```
###
# Minimal character-level vanilla MNIST model. Written by Andrej Karpathy (@karpathy)
#
# MIT License
#
# Copyright (c) 2016 Andrej Karpathy
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, in connection with, or in relation to, the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
###

import numpy as np

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

# hyperparameters
hidden_size = 280 # size of hidden layer of neurons
num_layers = 2 # number of steps to travel the net for
learning_rate = 1e-1

# model parameters
w_h = np.random.randn(hidden_size, hidden_size)*0.1 # input to hidden
w_hx = np.random.randn(hidden_size, hidden_size)*0.1 # h to hidden
w_hy = np.random.randn(vocab_size, hidden_size)*0.1 # hidden to output
w_o = zeros((vocab_size, 3)) # hidden bias
b_o = zeros((vocab_size, 3)) # output bias
```

[illegible][illegible]

```

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(Wyh, 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)

```

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

$$y_t = W_{hy}h_t$$

Softmax classifier

min-char-rnn.py gist

```

1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 b_i = np.zeros(hidden_size, 1) # hidden bias
25 b_h = np.zeros(hidden_size, 1) # hidden bias
26 b_o = np.zeros(vocab_size, 1) # output bias

```

```

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

```

```

62 def sample(h, ix, ix):
63     """
64     sample a sequence of integers from the model
65     h is memory state, seed, ix is seed letter for first time step
66     """
67     x = np.zeros(vocab_size, 1)
68     ix = ix
69     ix = ix
70     ix = ix
71     for t in xrange(1):
72         xi = np.zeros(vocab_size, 1)
73         xi[ix_to_char[ix]] = 1
74         hxi = np.tanh(np.dot(wih, xi)) + np.dot(whh, h) + bi
75         yi = np.dot(why, hxi) + bo
76         pi = np.exp(yi) / np.sum(np.exp(yi))
77         ix = np.random.choice(vocab_size, 1, p=pi)[0]
78         x[ix_to_char[ix]] = 1
79         ix = ix
80     return ix
81
82 # run
83 N, p = 0, 0
84 mih, mhh, mwhy, mbi, mbo, mps = [None]*6 # memory variables for Adam
85 mwh, mwy = np.zeros_like(wih), np.zeros_like(whh) # memory variables for Adam
86 smooth_loss = np.log(0.0/vocab_size)/seq_length # loss at iteration 0
87 while True:
88     # generate inputs (we're sampling from left to right in steps seq_length long)
89     if p%seq_length == 0 and (p == 0 or p == 1):
90         hprev = np.zeros(hidden_size, 1) # reset new memory
91         p = 0 # p from count of data
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+seq_length:p+2*seq_length]]
94     # sample from the model now and then
95     if p % 100 == 0:
96         sample_ix = sample(hprev, inputs[0], 0)
97         text = ''.join(ix_to_char[ix] for ix in sample_ix)
98         print '==== %d %s =====' % (p, text)
99     # forward seq_length characters through the net and fetch gradients
100     loss, dwhx, dbh, dwhy, dh, dhy, hprev = lossfun(inputs, targets, hprev)
101     smooth_loss = smooth_loss + 0.999 * loss + 0.001
102     if p % 100 == 0: print 'iter %d, loss %f' % (p, smooth_loss) # print progress
103     # perform parameter update with Adam
104     for param, dparam, m in zip([wih, whh, why, bi, bo], [dwhx, dbh, dwhy, dbh, dby], [mwh, mhh, mwhy, mbi, mbo]):
105         m = 0.9 * m + dparam / np.sqrt(m + 1e-8) # Adam update
106     wih, whh, why, bi, bo = wih + learning_rate * dparam / np.sqrt(m + 1e-8) # Adam update
107     p += seq_length # move data pointer
108     p = 1 # iteration counter

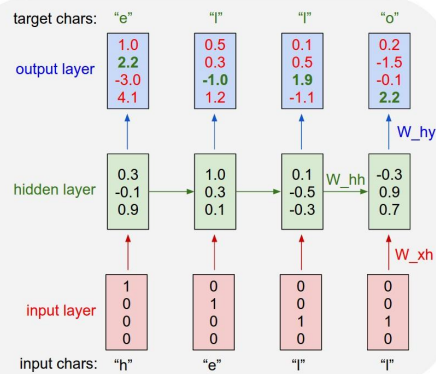
```

```

44 # backward pass: compute gradients going backwards
45 dwhx, dwhh, dwhy = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
46 dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47 dhnext = np.zeros_like(hs[0])
48 for t in reversed(xrange(len(inputs))):
49     dy = np.copy(pi[t])
50     dy[targets[t]] -= 1 # backprop into y
51     dwhy += np.dot(dy, hs[t].T)
52     dbh += dy
53     dh = np.dot(Why.T, dy) + dhnext # backprop into h
54     dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55     dbh += dhraw
56     dwhx += np.dot(dhraw, xs[t].T)
57     dwhh += np.dot(dhraw, hs[t-1].T)
58     dhnext = np.dot(Whh.T, dhraw)
59 for dparam in [dwhx, dwhh, dwhy, dbh, dby]:
60     np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61 return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]

```

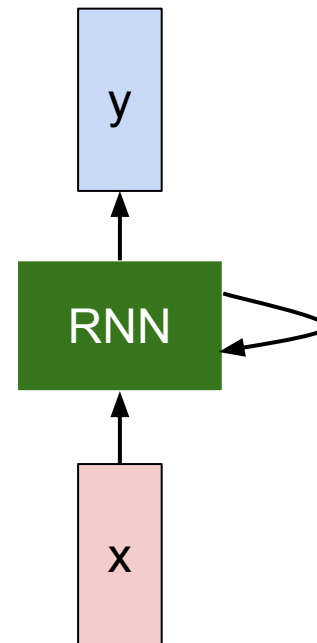
recall:



min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossfun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is wci array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     ix, hix, yix, ox = [], [], [], []
34     hix[0] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi[i] = np.zeros((vocab_size,1)) # encode in 1-of-K representation
39         xi[i][inputs[t]] = 1
40         hi[i] = np.tanh(np.dot(wh, xi[i]) + np.dot(whh, hi[i-1]) + bh) # hidden state
41         yi[i] = np.dot(why, hi[i]) + by # unnormalized log probabilities for next chars
42         pi[i] = np.exp(yi[i]) / np.sum(np.exp(yi[i])) # probabilities for next chars
43         loss += -np.log(pi[i][targets[t],0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients using backward
45     dwh, dwhh, dwhy = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
46     dh, dhv = np.zeros_like(hi), np.zeros_like(hi)
47     dhnxt = np.zeros_like(hi[0])
48     for t in reversed(xrange(len(inputs))):
49         dy = np.copy(pi[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dhi = np.dot(dy, hi[t].T)
52         dhy = dy
53         dh = np.dot(dhi, T, dy) + dhnxt # backprop into h
54         dhrw = (1 - hi[t]**2) * dhi # dh a backprop through tanh nonlinearity
55         dwh += np.dot(dhrw, xi[t].T)
56         dwhh += np.dot(dhrw, hi[t-1].T)
57         dhnxt = np.dot(dhrw, hi[t].T)
58         dhnxt = np.dot(dhnxt, T, dhnxt)
59         for dparam in [dwh, dwhh, dwhy, dh, dhy]:
60             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61     return loss, dwh, dwhh, dwhy, dh, dhy, hi[len(inputs)-1]
62
63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     ix = seed_ix
70     ixes = []
71     for t in xrange(n):
72         h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
73         y = np.dot(Why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p.ravel())
76         x = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79     return ixes
80
81 # main loop
82 """
83 """
84 # sample a sequence of integers from the model
85 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
86 """
87 """
88 x = np.zeros((vocab_size, 1))
89 ixes = []
90 loss = 0
91 for t in xrange(n):
92     xi = np.zeros((vocab_size, 1))
93     xi[ix] = 1
94     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
95     yi = np.dot(why, hi) + by
96     pi = np.exp(yi) / np.sum(np.exp(yi))
97     ix = np.random.choice(range(vocab_size), p=pi.ravel())
98     x = np.zeros((vocab_size, 1))
99     x[ix] = 1
100     loss += -np.log(pi[ix,0])
101     return loss
102
103 # main loop
104 """
105 """
106 # sample a sequence of integers from the model
107 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
108 """
109 """
110 x = np.zeros((vocab_size, 1))
111 ixes = []
112 loss = 0
113 for t in xrange(n):
114     xi = np.zeros((vocab_size, 1))
115     xi[ix] = 1
116     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
117     yi = np.dot(why, hi) + by
118     pi = np.exp(yi) / np.sum(np.exp(yi))
119     ix = np.random.choice(range(vocab_size), p=pi.ravel())
120     x = np.zeros((vocab_size, 1))
121     x[ix] = 1
122     loss += -np.log(pi[ix,0])
123     return loss
124
125 # main loop
126 """
127 """
128 # sample a sequence of integers from the model
129 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
130 """
131 """
132 x = np.zeros((vocab_size, 1))
133 ixes = []
134 loss = 0
135 for t in xrange(n):
136     xi = np.zeros((vocab_size, 1))
137     xi[ix] = 1
138     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
139     yi = np.dot(why, hi) + by
140     pi = np.exp(yi) / np.sum(np.exp(yi))
141     ix = np.random.choice(range(vocab_size), p=pi.ravel())
142     x = np.zeros((vocab_size, 1))
143     x[ix] = 1
144     loss += -np.log(pi[ix,0])
145     return loss
146
147 # main loop
148 """
149 """
150 # sample a sequence of integers from the model
151 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
152 """
153 """
154 x = np.zeros((vocab_size, 1))
155 ixes = []
156 loss = 0
157 for t in xrange(n):
158     xi = np.zeros((vocab_size, 1))
159     xi[ix] = 1
160     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
161     yi = np.dot(why, hi) + by
162     pi = np.exp(yi) / np.sum(np.exp(yi))
163     ix = np.random.choice(range(vocab_size), p=pi.ravel())
164     x = np.zeros((vocab_size, 1))
165     x[ix] = 1
166     loss += -np.log(pi[ix,0])
167     return loss
168
169 # main loop
170 """
171 """
172 # sample a sequence of integers from the model
173 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
174 """
175 """
176 x = np.zeros((vocab_size, 1))
177 ixes = []
178 loss = 0
179 for t in xrange(n):
180     xi = np.zeros((vocab_size, 1))
181     xi[ix] = 1
182     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
183     yi = np.dot(why, hi) + by
184     pi = np.exp(yi) / np.sum(np.exp(yi))
185     ix = np.random.choice(range(vocab_size), p=pi.ravel())
186     x = np.zeros((vocab_size, 1))
187     x[ix] = 1
188     loss += -np.log(pi[ix,0])
189     return loss
190
191 # main loop
192 """
193 """
194 # sample a sequence of integers from the model
195 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
196 """
197 """
198 x = np.zeros((vocab_size, 1))
199 ixes = []
200 loss = 0
201 for t in xrange(n):
202     xi = np.zeros((vocab_size, 1))
203     xi[ix] = 1
204     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
205     yi = np.dot(why, hi) + by
206     pi = np.exp(yi) / np.sum(np.exp(yi))
207     ix = np.random.choice(range(vocab_size), p=pi.ravel())
208     x = np.zeros((vocab_size, 1))
209     x[ix] = 1
210     loss += -np.log(pi[ix,0])
211     return loss
212
213 # main loop
214 """
215 """
216 # sample a sequence of integers from the model
217 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
218 """
219 """
220 x = np.zeros((vocab_size, 1))
221 ixes = []
222 loss = 0
223 for t in xrange(n):
224     xi = np.zeros((vocab_size, 1))
225     xi[ix] = 1
226     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
227     yi = np.dot(why, hi) + by
228     pi = np.exp(yi) / np.sum(np.exp(yi))
229     ix = np.random.choice(range(vocab_size), p=pi.ravel())
230     x = np.zeros((vocab_size, 1))
231     x[ix] = 1
232     loss += -np.log(pi[ix,0])
233     return loss
234
235 # main loop
236 """
237 """
238 # sample a sequence of integers from the model
239 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
240 """
241 """
242 x = np.zeros((vocab_size, 1))
243 ixes = []
244 loss = 0
245 for t in xrange(n):
246     xi = np.zeros((vocab_size, 1))
247     xi[ix] = 1
248     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
249     yi = np.dot(why, hi) + by
250     pi = np.exp(yi) / np.sum(np.exp(yi))
251     ix = np.random.choice(range(vocab_size), p=pi.ravel())
252     x = np.zeros((vocab_size, 1))
253     x[ix] = 1
254     loss += -np.log(pi[ix,0])
255     return loss
256
257 # main loop
258 """
259 """
260 # sample a sequence of integers from the model
261 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
262 """
263 """
264 x = np.zeros((vocab_size, 1))
265 ixes = []
266 loss = 0
267 for t in xrange(n):
268     xi = np.zeros((vocab_size, 1))
269     xi[ix] = 1
270     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
271     yi = np.dot(why, hi) + by
272     pi = np.exp(yi) / np.sum(np.exp(yi))
273     ix = np.random.choice(range(vocab_size), p=pi.ravel())
274     x = np.zeros((vocab_size, 1))
275     x[ix] = 1
276     loss += -np.log(pi[ix,0])
277     return loss
278
279 # main loop
280 """
281 """
282 # sample a sequence of integers from the model
283 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
284 """
285 """
286 x = np.zeros((vocab_size, 1))
287 ixes = []
288 loss = 0
289 for t in xrange(n):
290     xi = np.zeros((vocab_size, 1))
291     xi[ix] = 1
292     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
293     yi = np.dot(why, hi) + by
294     pi = np.exp(yi) / np.sum(np.exp(yi))
295     ix = np.random.choice(range(vocab_size), p=pi.ravel())
296     x = np.zeros((vocab_size, 1))
297     x[ix] = 1
298     loss += -np.log(pi[ix,0])
299     return loss
300
301 # main loop
302 """
303 """
304 # sample a sequence of integers from the model
305 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
306 """
307 """
308 x = np.zeros((vocab_size, 1))
309 ixes = []
310 loss = 0
311 for t in xrange(n):
312     xi = np.zeros((vocab_size, 1))
313     xi[ix] = 1
314     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
315     yi = np.dot(why, hi) + by
316     pi = np.exp(yi) / np.sum(np.exp(yi))
317     ix = np.random.choice(range(vocab_size), p=pi.ravel())
318     x = np.zeros((vocab_size, 1))
319     x[ix] = 1
320     loss += -np.log(pi[ix,0])
321     return loss
322
323 # main loop
324 """
325 """
326 # sample a sequence of integers from the model
327 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
328 """
329 """
330 x = np.zeros((vocab_size, 1))
331 ixes = []
332 loss = 0
333 for t in xrange(n):
334     xi = np.zeros((vocab_size, 1))
335     xi[ix] = 1
336     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
337     yi = np.dot(why, hi) + by
338     pi = np.exp(yi) / np.sum(np.exp(yi))
339     ix = np.random.choice(range(vocab_size), p=pi.ravel())
340     x = np.zeros((vocab_size, 1))
341     x[ix] = 1
342     loss += -np.log(pi[ix,0])
343     return loss
344
345 # main loop
346 """
347 """
348 # sample a sequence of integers from the model
349 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
350 """
351 """
352 x = np.zeros((vocab_size, 1))
353 ixes = []
354 loss = 0
355 for t in xrange(n):
356     xi = np.zeros((vocab_size, 1))
357     xi[ix] = 1
358     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
359     yi = np.dot(why, hi) + by
360     pi = np.exp(yi) / np.sum(np.exp(yi))
361     ix = np.random.choice(range(vocab_size), p=pi.ravel())
362     x = np.zeros((vocab_size, 1))
363     x[ix] = 1
364     loss += -np.log(pi[ix,0])
365     return loss
366
367 # main loop
368 """
369 """
370 # sample a sequence of integers from the model
371 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
372 """
373 """
374 x = np.zeros((vocab_size, 1))
375 ixes = []
376 loss = 0
377 for t in xrange(n):
378     xi = np.zeros((vocab_size, 1))
379     xi[ix] = 1
380     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
381     yi = np.dot(why, hi) + by
382     pi = np.exp(yi) / np.sum(np.exp(yi))
383     ix = np.random.choice(range(vocab_size), p=pi.ravel())
384     x = np.zeros((vocab_size, 1))
385     x[ix] = 1
386     loss += -np.log(pi[ix,0])
387     return loss
388
389 # main loop
390 """
391 """
392 # sample a sequence of integers from the model
393 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
394 """
395 """
396 x = np.zeros((vocab_size, 1))
397 ixes = []
398 loss = 0
399 for t in xrange(n):
400     xi = np.zeros((vocab_size, 1))
401     xi[ix] = 1
402     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
403     yi = np.dot(why, hi) + by
404     pi = np.exp(yi) / np.sum(np.exp(yi))
405     ix = np.random.choice(range(vocab_size), p=pi.ravel())
406     x = np.zeros((vocab_size, 1))
407     x[ix] = 1
408     loss += -np.log(pi[ix,0])
409     return loss
410
411 # main loop
412 """
413 """
414 # sample a sequence of integers from the model
415 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
416 """
417 """
418 x = np.zeros((vocab_size, 1))
419 ixes = []
420 loss = 0
421 for t in xrange(n):
422     xi = np.zeros((vocab_size, 1))
423     xi[ix] = 1
424     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
425     yi = np.dot(why, hi) + by
426     pi = np.exp(yi) / np.sum(np.exp(yi))
427     ix = np.random.choice(range(vocab_size), p=pi.ravel())
428     x = np.zeros((vocab_size, 1))
429     x[ix] = 1
430     loss += -np.log(pi[ix,0])
431     return loss
432
433 # main loop
434 """
435 """
436 # sample a sequence of integers from the model
437 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
438 """
439 """
440 x = np.zeros((vocab_size, 1))
441 ixes = []
442 loss = 0
443 for t in xrange(n):
444     xi = np.zeros((vocab_size, 1))
445     xi[ix] = 1
446     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
447     yi = np.dot(why, hi) + by
448     pi = np.exp(yi) / np.sum(np.exp(yi))
449     ix = np.random.choice(range(vocab_size), p=pi.ravel())
450     x = np.zeros((vocab_size, 1))
451     x[ix] = 1
452     loss += -np.log(pi[ix,0])
453     return loss
454
455 # main loop
456 """
457 """
458 # sample a sequence of integers from the model
459 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
460 """
461 """
462 x = np.zeros((vocab_size, 1))
463 ixes = []
464 loss = 0
465 for t in xrange(n):
466     xi = np.zeros((vocab_size, 1))
467     xi[ix] = 1
468     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
469     yi = np.dot(why, hi) + by
470     pi = np.exp(yi) / np.sum(np.exp(yi))
471     ix = np.random.choice(range(vocab_size), p=pi.ravel())
472     x = np.zeros((vocab_size, 1))
473     x[ix] = 1
474     loss += -np.log(pi[ix,0])
475     return loss
476
477 # main loop
478 """
479 """
480 # sample a sequence of integers from the model
481 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
482 """
483 """
484 x = np.zeros((vocab_size, 1))
485 ixes = []
486 loss = 0
487 for t in xrange(n):
488     xi = np.zeros((vocab_size, 1))
489     xi[ix] = 1
490     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
491     yi = np.dot(why, hi) + by
492     pi = np.exp(yi) / np.sum(np.exp(yi))
493     ix = np.random.choice(range(vocab_size), p=pi.ravel())
494     x = np.zeros((vocab_size, 1))
495     x[ix] = 1
496     loss += -np.log(pi[ix,0])
497     return loss
498
499 # main loop
500 """
501 """
502 # sample a sequence of integers from the model
503 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
504 """
505 """
506 x = np.zeros((vocab_size, 1))
507 ixes = []
508 loss = 0
509 for t in xrange(n):
510     xi = np.zeros((vocab_size, 1))
511     xi[ix] = 1
512     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
513     yi = np.dot(why, hi) + by
514     pi = np.exp(yi) / np.sum(np.exp(yi))
515     ix = np.random.choice(range(vocab_size), p=pi.ravel())
516     x = np.zeros((vocab_size, 1))
517     x[ix] = 1
518     loss += -np.log(pi[ix,0])
519     return loss
520
521 # main loop
522 """
523 """
524 # sample a sequence of integers from the model
525 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
526 """
527 """
528 x = np.zeros((vocab_size, 1))
529 ixes = []
530 loss = 0
531 for t in xrange(n):
532     xi = np.zeros((vocab_size, 1))
533     xi[ix] = 1
534     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
535     yi = np.dot(why, hi) + by
536     pi = np.exp(yi) / np.sum(np.exp(yi))
537     ix = np.random.choice(range(vocab_size), p=pi.ravel())
538     x = np.zeros((vocab_size, 1))
539     x[ix] = 1
540     loss += -np.log(pi[ix,0])
541     return loss
542
543 # main loop
544 """
545 """
546 # sample a sequence of integers from the model
547 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
548 """
549 """
550 x = np.zeros((vocab_size, 1))
551 ixes = []
552 loss = 0
553 for t in xrange(n):
554     xi = np.zeros((vocab_size, 1))
555     xi[ix] = 1
556     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
557     yi = np.dot(why, hi) + by
558     pi = np.exp(yi) / np.sum(np.exp(yi))
559     ix = np.random.choice(range(vocab_size), p=pi.ravel())
560     x = np.zeros((vocab_size, 1))
561     x[ix] = 1
562     loss += -np.log(pi[ix,0])
563     return loss
564
565 # main loop
566 """
567 """
568 # sample a sequence of integers from the model
569 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
570 """
571 """
572 x = np.zeros((vocab_size, 1))
573 ixes = []
574 loss = 0
575 for t in xrange(n):
576     xi = np.zeros((vocab_size, 1))
577     xi[ix] = 1
578     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
579     yi = np.dot(why, hi) + by
580     pi = np.exp(yi) / np.sum(np.exp(yi))
581     ix = np.random.choice(range(vocab_size), p=pi.ravel())
582     x = np.zeros((vocab_size, 1))
583     x[ix] = 1
584     loss += -np.log(pi[ix,0])
585     return loss
586
587 # main loop
588 """
589 """
590 # sample a sequence of integers from the model
591 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
592 """
593 """
594 x = np.zeros((vocab_size, 1))
595 ixes = []
596 loss = 0
597 for t in xrange(n):
598     xi = np.zeros((vocab_size, 1))
599     xi[ix] = 1
600     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
601     yi = np.dot(why, hi) + by
602     pi = np.exp(yi) / np.sum(np.exp(yi))
603     ix = np.random.choice(range(vocab_size), p=pi.ravel())
604     x = np.zeros((vocab_size, 1))
605     x[ix] = 1
606     loss += -np.log(pi[ix,0])
607     return loss
608
609 # main loop
610 """
611 """
612 # sample a sequence of integers from the model
613 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
614 """
615 """
616 x = np.zeros((vocab_size, 1))
617 ixes = []
618 loss = 0
619 for t in xrange(n):
620     xi = np.zeros((vocab_size, 1))
621     xi[ix] = 1
622     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
623     yi = np.dot(why, hi) + by
624     pi = np.exp(yi) / np.sum(np.exp(yi))
625     ix = np.random.choice(range(vocab_size), p=pi.ravel())
626     x = np.zeros((vocab_size, 1))
627     x[ix] = 1
628     loss += -np.log(pi[ix,0])
629     return loss
630
631 # main loop
632 """
633 """
634 # sample a sequence of integers from the model
635 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
636 """
637 """
638 x = np.zeros((vocab_size, 1))
639 ixes = []
640 loss = 0
641 for t in xrange(n):
642     xi = np.zeros((vocab_size, 1))
643     xi[ix] = 1
644     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
645     yi = np.dot(why, hi) + by
646     pi = np.exp(yi) / np.sum(np.exp(yi))
647     ix = np.random.choice(range(vocab_size), p=pi.ravel())
648     x = np.zeros((vocab_size, 1))
649     x[ix] = 1
650     loss += -np.log(pi[ix,0])
651     return loss
652
653 # main loop
654 """
655 """
656 # sample a sequence of integers from the model
657 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
658 """
659 """
660 x = np.zeros((vocab_size, 1))
661 ixes = []
662 loss = 0
663 for t in xrange(n):
664     xi = np.zeros((vocab_size, 1))
665     xi[ix] = 1
666     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
667     yi = np.dot(why, hi) + by
668     pi = np.exp(yi) / np.sum(np.exp(yi))
669     ix = np.random.choice(range(vocab_size), p=pi.ravel())
670     x = np.zeros((vocab_size, 1))
671     x[ix] = 1
672     loss += -np.log(pi[ix,0])
673     return loss
674
675 # main loop
676 """
677 """
678 # sample a sequence of integers from the model
679 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
680 """
681 """
682 x = np.zeros((vocab_size, 1))
683 ixes = []
684 loss = 0
685 for t in xrange(n):
686     xi = np.zeros((vocab_size, 1))
687     xi[ix] = 1
688     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
689     yi = np.dot(why, hi) + by
690     pi = np.exp(yi) / np.sum(np.exp(yi))
691     ix = np.random.choice(range(vocab_size), p=pi.ravel())
692     x = np.zeros((vocab_size, 1))
693     x[ix] = 1
694     loss += -np.log(pi[ix,0])
695     return loss
696
697 # main loop
698 """
699 """
700 # sample a sequence of integers from the model
701 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
702 """
703 """
704 x = np.zeros((vocab_size, 1))
705 ixes = []
706 loss = 0
707 for t in xrange(n):
708     xi = np.zeros((vocab_size, 1))
709     xi[ix] = 1
710     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
711     yi = np.dot(why, hi) + by
712     pi = np.exp(yi) / np.sum(np.exp(yi))
713     ix = np.random.choice(range(vocab_size), p=pi.ravel())
714     x = np.zeros((vocab_size, 1))
715     x[ix] = 1
716     loss += -np.log(pi[ix,0])
717     return loss
718
719 # main loop
720 """
721 """
722 # sample a sequence of integers from the model
723 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
724 """
725 """
726 x = np.zeros((vocab_size, 1))
727 ixes = []
728 loss = 0
729 for t in xrange(n):
730     xi = np.zeros((vocab_size, 1))
731     xi[ix] = 1
732     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
733     yi = np.dot(why, hi) + by
734     pi = np.exp(yi) / np.sum(np.exp(yi))
735     ix = np.random.choice(range(vocab_size), p=pi.ravel())
736     x = np.zeros((vocab_size, 1))
737     x[ix] = 1
738     loss += -np.log(pi[ix,0])
739     return loss
740
741 # main loop
742 """
743 """
744 # sample a sequence of integers from the model
745 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
746 """
747 """
748 x = np.zeros((vocab_size, 1))
749 ixes = []
750 loss = 0
751 for t in xrange(n):
752     xi = np.zeros((vocab_size, 1))
753     xi[ix] = 1
754     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
755     yi = np.dot(why, hi) + by
756     pi = np.exp(yi) / np.sum(np.exp(yi))
757     ix = np.random.choice(range(vocab_size), p=pi.ravel())
758     x = np.zeros((vocab_size, 1))
759     x[ix] = 1
760     loss += -np.log(pi[ix,0])
761     return loss
762
763 # main loop
764 """
765 """
766 # sample a sequence of integers from the model
767 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
768 """
769 """
770 x = np.zeros((vocab_size, 1))
771 ixes = []
772 loss = 0
773 for t in xrange(n):
774     xi = np.zeros((vocab_size, 1))
775     xi[ix] = 1
776     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
777     yi = np.dot(why, hi) + by
778     pi = np.exp(yi) / np.sum(np.exp(yi))
779     ix = np.random.choice(range(vocab_size), p=pi.ravel())
780     x = np.zeros((vocab_size, 1))
781     x[ix] = 1
782     loss += -np.log(pi[ix,0])
783     return loss
784
785 # main loop
786 """
787 """
788 # sample a sequence of integers from the model
789 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
790 """
791 """
792 x = np.zeros((vocab_size, 1))
793 ixes = []
794 loss = 0
795 for t in xrange(n):
796     xi = np.zeros((vocab_size, 1))
797     xi[ix] = 1
798     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
799     yi = np.dot(why, hi) + by
800     pi = np.exp(yi) / np.sum(np.exp(yi))
801     ix = np.random.choice(range(vocab_size), p=pi.ravel())
802     x = np.zeros((vocab_size, 1))
803     x[ix] = 1
804     loss += -np.log(pi[ix,0])
805     return loss
806
807 # main loop
808 """
809 """
810 # sample a sequence of integers from the model
811 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
812 """
813 """
814 x = np.zeros((vocab_size, 1))
815 ixes = []
816 loss = 0
817 for t in xrange(n):
818     xi = np.zeros((vocab_size, 1))
819     xi[ix] = 1
820     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
821     yi = np.dot(why, hi) + by
822     pi = np.exp(yi) / np.sum(np.exp(yi))
823     ix = np.random.choice(range(vocab_size), p=pi.ravel())
824     x = np.zeros((vocab_size, 1))
825     x[ix] = 1
826     loss += -np.log(pi[ix,0])
827     return loss
828
829 # main loop
830 """
831 """
832 # sample a sequence of integers from the model
833 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
834 """
835 """
836 x = np.zeros((vocab_size, 1))
837 ixes = []
838 loss = 0
839 for t in xrange(n):
840     xi = np.zeros((vocab_size, 1))
841     xi[ix] = 1
842     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
843     yi = np.dot(why, hi) + by
844     pi = np.exp(yi) / np.sum(np.exp(yi))
845     ix = np.random.choice(range(vocab_size), p=pi.ravel())
846     x = np.zeros((vocab_size, 1))
847     x[ix] = 1
848     loss += -np.log(pi[ix,0])
849     return loss
850
851 # main loop
852 """
853 """
854 # sample a sequence of integers from the model
855 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
856 """
857 """
858 x = np.zeros((vocab_size, 1))
859 ixes = []
860 loss = 0
861 for t in xrange(n):
862     xi = np.zeros((vocab_size, 1))
863     xi[ix] = 1
864     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
865     yi = np.dot(why, hi) + by
866     pi = np.exp(yi) / np.sum(np.exp(yi))
867     ix = np.random.choice(range(vocab_size), p=pi.ravel())
868     x = np.zeros((vocab_size, 1))
869     x[ix] = 1
870     loss += -np.log(pi[ix,0])
871     return loss
872
873 # main loop
874 """
875 """
876 # sample a sequence of integers from the model
877 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
878 """
879 """
880 x = np.zeros((vocab_size, 1))
881 ixes = []
882 loss = 0
883 for t in xrange(n):
884     xi = np.zeros((vocab_size, 1))
885     xi[ix] = 1
886     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
887     yi = np.dot(why, hi) + by
888     pi = np.exp(yi) / np.sum(np.exp(yi))
889     ix = np.random.choice(range(vocab_size), p=pi.ravel())
890     x = np.zeros((vocab_size, 1))
891     x[ix] = 1
892     loss += -np.log(pi[ix,0])
893     return loss
894
895 # main loop
896 """
897 """
898 # sample a sequence of integers from the model
899 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
900 """
901 """
902 x = np.zeros((vocab_size, 1))
903 ixes = []
904 loss = 0
905 for t in xrange(n):
906     xi = np.zeros((vocab_size, 1))
907     xi[ix] = 1
908     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
909     yi = np.dot(why, hi) + by
910     pi = np.exp(yi) / np.sum(np.exp(yi))
911     ix = np.random.choice(range(vocab_size), p=pi.ravel())
912     x = np.zeros((vocab_size, 1))
913     x[ix] = 1
914     loss += -np.log(pi[ix,0])
915     return loss
916
917 # main loop
918 """
919 """
920 # sample a sequence of integers from the model
921 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
922 """
923 """
924 x = np.zeros((vocab_size, 1))
925 ixes = []
926 loss = 0
927 for t in xrange(n):
928     xi = np.zeros((vocab_size, 1))
929     xi[ix] = 1
930     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
931     yi = np.dot(why, hi) + by
932     pi = np.exp(yi) / np.sum(np.exp(yi))
933     ix = np.random.choice(range(vocab_size), p=pi.ravel())
934     x = np.zeros((vocab_size, 1))
935     x[ix] = 1
936     loss += -np.log(pi[ix,0])
937     return loss
938
939 # main loop
940 """
941 """
942 # sample a sequence of integers from the model
943 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
944 """
945 """
946 x = np.zeros((vocab_size, 1))
947 ixes = []
948 loss = 0
949 for t in xrange(n):
950     xi = np.zeros((vocab_size, 1))
951     xi[ix] = 1
952     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
953     yi = np.dot(why, hi) + by
954     pi = np.exp(yi) / np.sum(np.exp(yi))
955     ix = np.random.choice(range(vocab_size), p=pi.ravel())
956     x = np.zeros((vocab_size, 1))
957     x[ix] = 1
958     loss += -np.log(pi[ix,0])
959     return loss
960
961 # main loop
962 """
963 """
964 # sample a sequence of integers from the model
965 h = np.zeros((hidden_size,1)) # memory state, seed_ix is seed letter for first time step
966 """
967 """
968 x = np.zeros((vocab_size, 1))
969 ixes = []
970 loss = 0
971 for t in xrange(n):
972     xi = np.zeros((vocab_size, 1))
973     xi[ix] = 1
974     hi = np.tanh(np.dot(wh, xi) + np.dot(whh, hi) + bh)
975     yi = np.dot(why, hi) + by
976     pi = np.exp(yi) / np.sum(np.exp(yi))
977     ix = np.random.choice(range(vocab_size), p=pi.ravel())
978     x = np.zeros((vocab_size, 1))
979     x[ix]
```

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



Sonnet 116 – Let me not ...

by William Shakespeare

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

at first:

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

↓
train more

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


↓
train more

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

↓
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.

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 $\bigoplus_{n=1,\dots,m}$ where $\mathcal{L}_{m,*} = 0$, 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

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

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 $X_{spaces, \acute{e}tale}$ 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 $\text{Proj}_X(\mathcal{A}) = \text{Spec}(B)$ 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 $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\dots,0}$.

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 $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A_2}$ works.

Lemma 0.3. In Situation ?? . Hence we may assume $\mathfrak{q}' = 0$.

Proof. We will use the property we see that \mathfrak{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 \mathbb{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

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

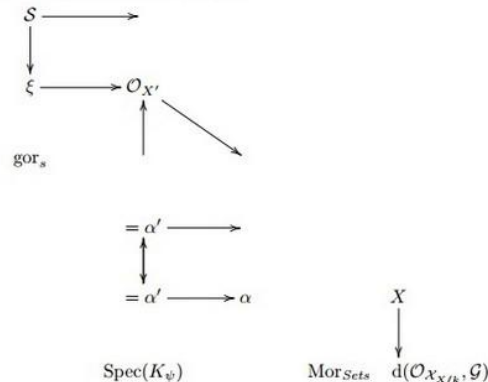
be a morphism of algebraic spaces over S and Y .

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 $\mathcal{O}_X(\mathcal{U})$ which is locally of finite type. □

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



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

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_x \rightarrow \mathcal{O}_{X_{\text{étale}}} \rightarrow \mathcal{O}_{X_{\text{étale}}}^{-1} \mathcal{O}_{X_{\text{étale}}}(\mathcal{O}_{X_{\text{étale}}}^{\vee})$$

is an isomorphism of covering of $\mathcal{O}_{X_{\text{étale}}}$. 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 $\mathcal{O}_{X_{\lambda}}$ 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 ago

latest commit 4b1706927d

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

Code

Pull requests 74

Pulse

Graphs

HTTPS clone URL

https://github.com/torvalds/linux

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

```

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

```

Generated C code

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

```

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

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

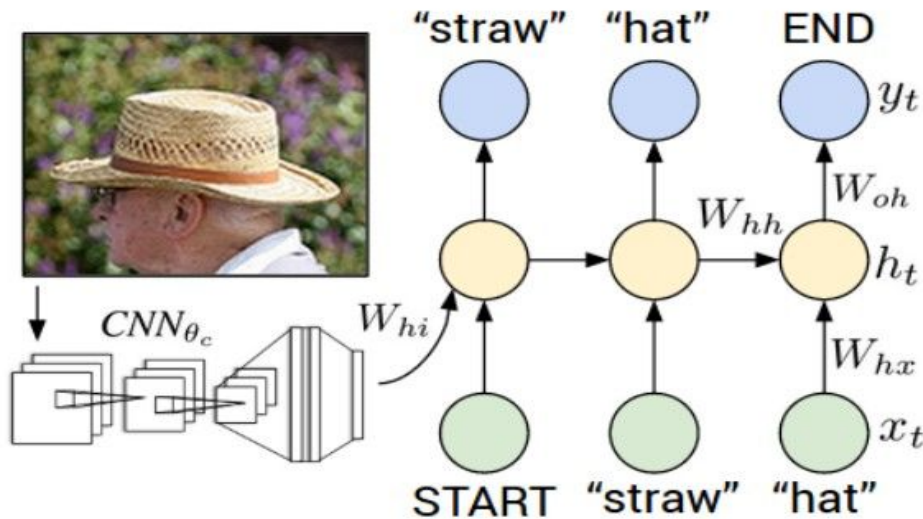
#define SWAP_ALLOCATE(nr)      (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %%3" : : "r" (0)); \
    if (__type & DO_READ)

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

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

```

Image Captioning



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

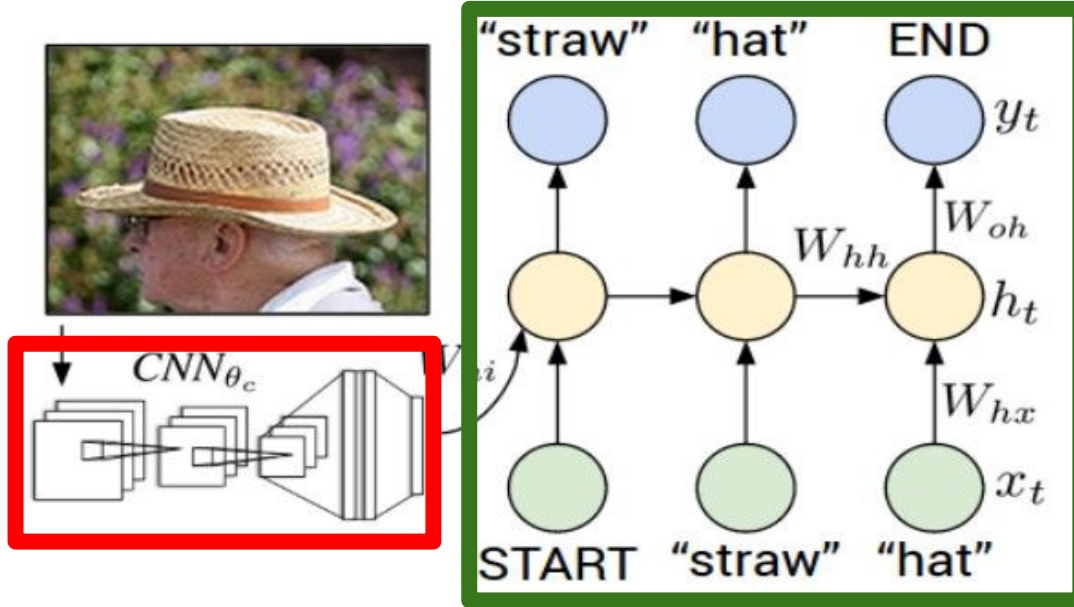
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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

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

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

Recurrent Neural Network



Convolutional Neural Network



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

x0
<STA
RT>

<START>

image

test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

V

y0

h0

x0

<STA

RT>

<START>

Wih

before:

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

now:

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

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

y0

h0

x0
<START
RT>

straw

sample!

<START>

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

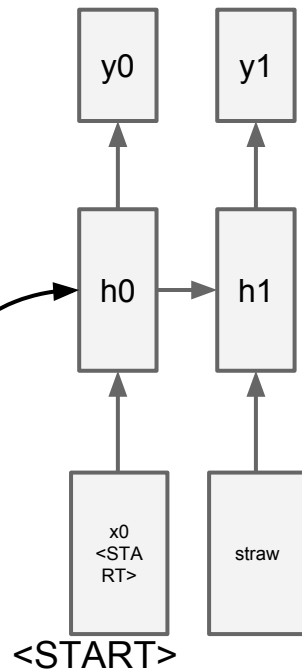
maxpool

FC-4096

FC-4096



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

y0

y1

h0

h1

x0
<START
RT>

straw

hat

sample!

<START>

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

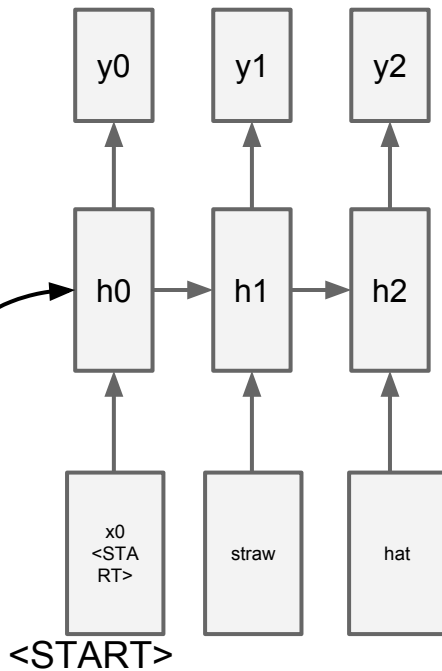
maxpool

FC-4096

FC-4096



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

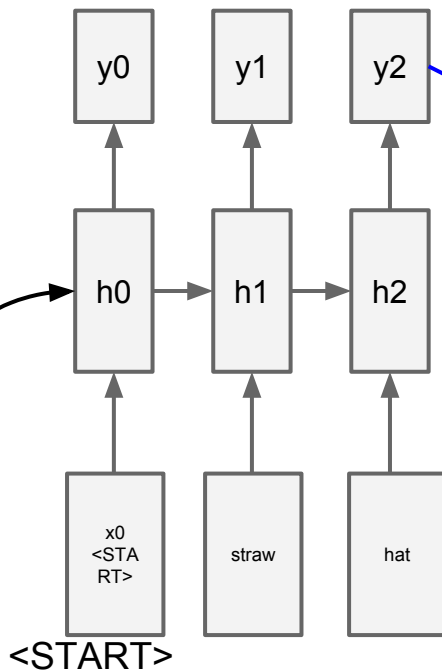
maxpool

FC-4096

FC-4096



test image



sample
<END> token
=> finish.

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."



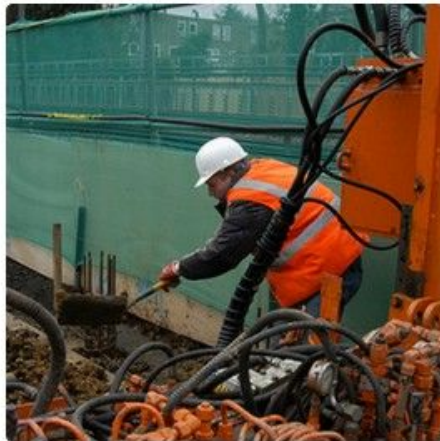
"two young girls are playing with lego toy."



"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."

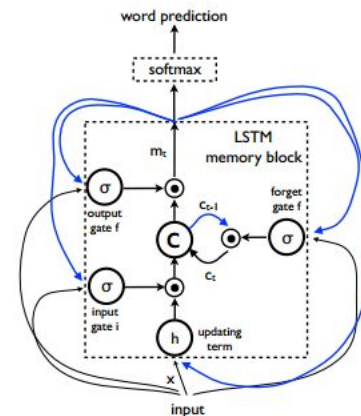
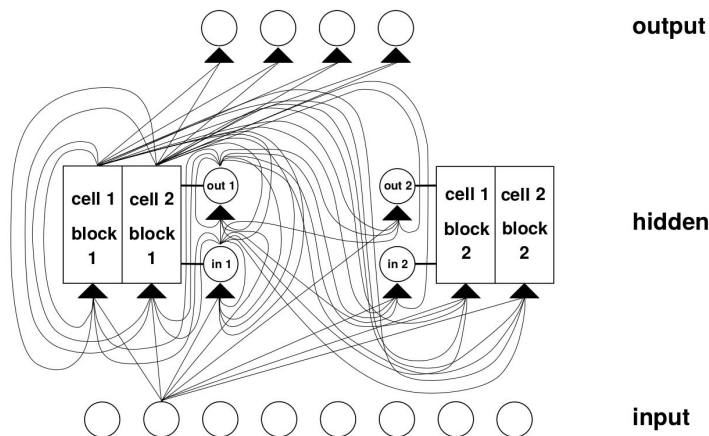
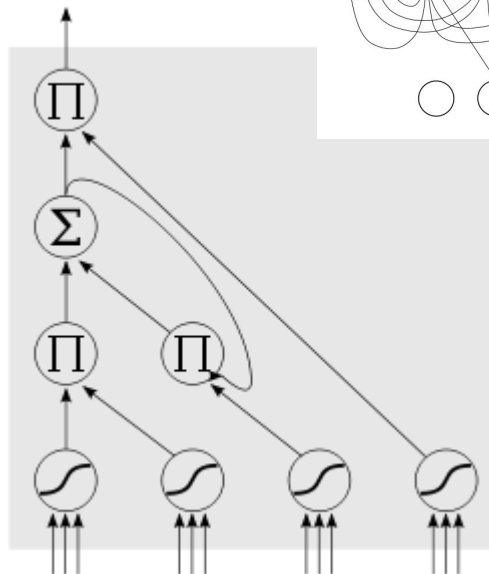
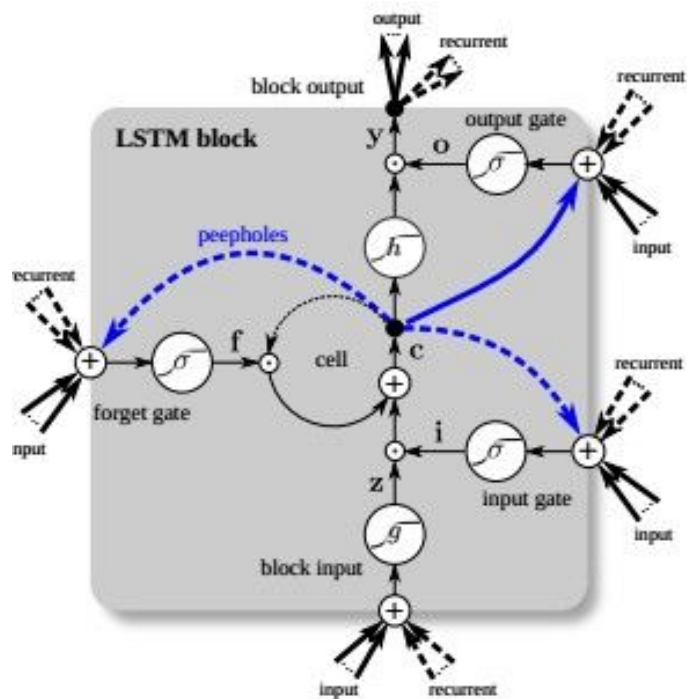


"a woman holding a teddy bear in front of a mirror."



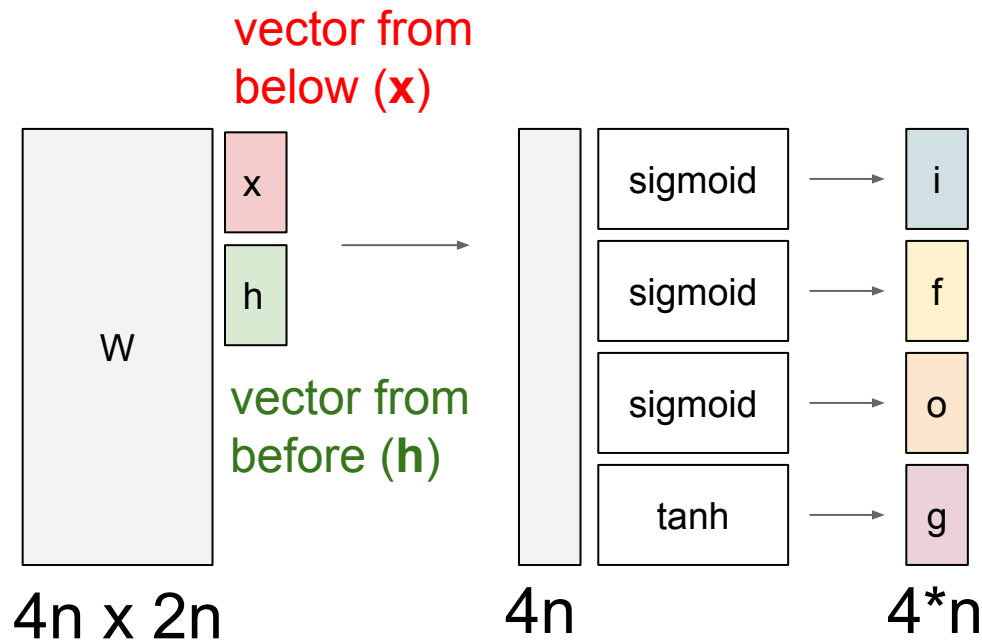
"a horse is standing in the middle of a road."

LSTM



Long Short Term Memory (LSTM)

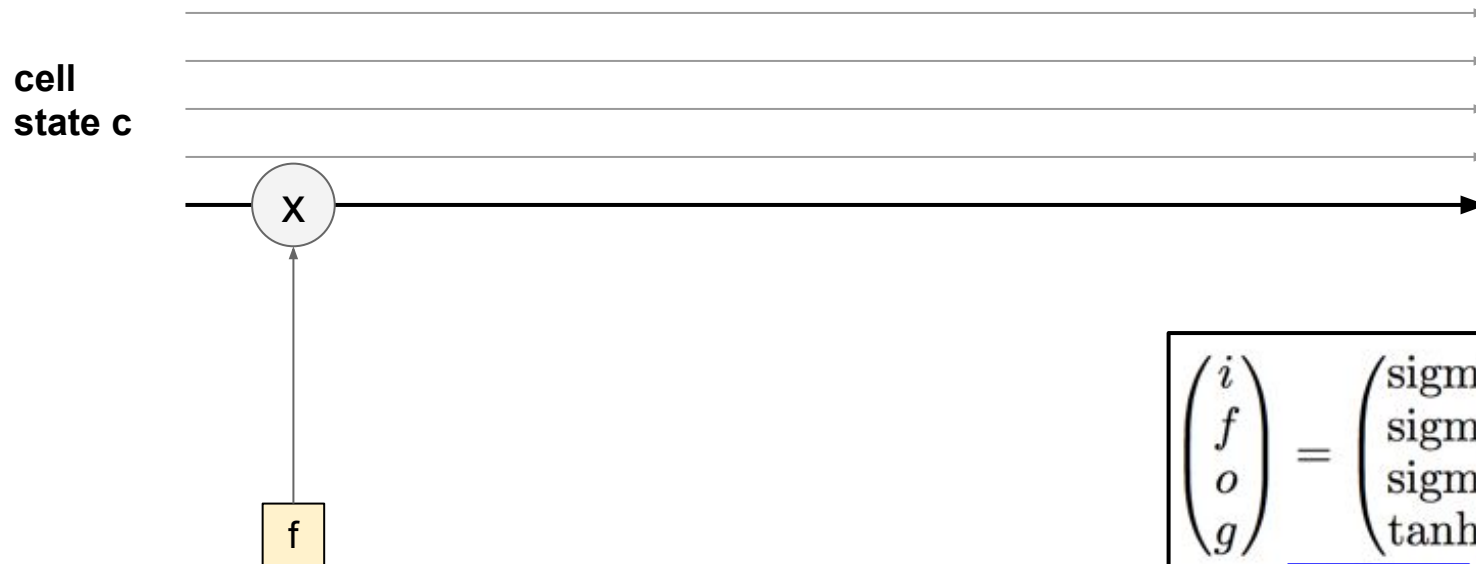
[Hochreiter et al., 1997]



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

Long Short Term Memory (LSTM)

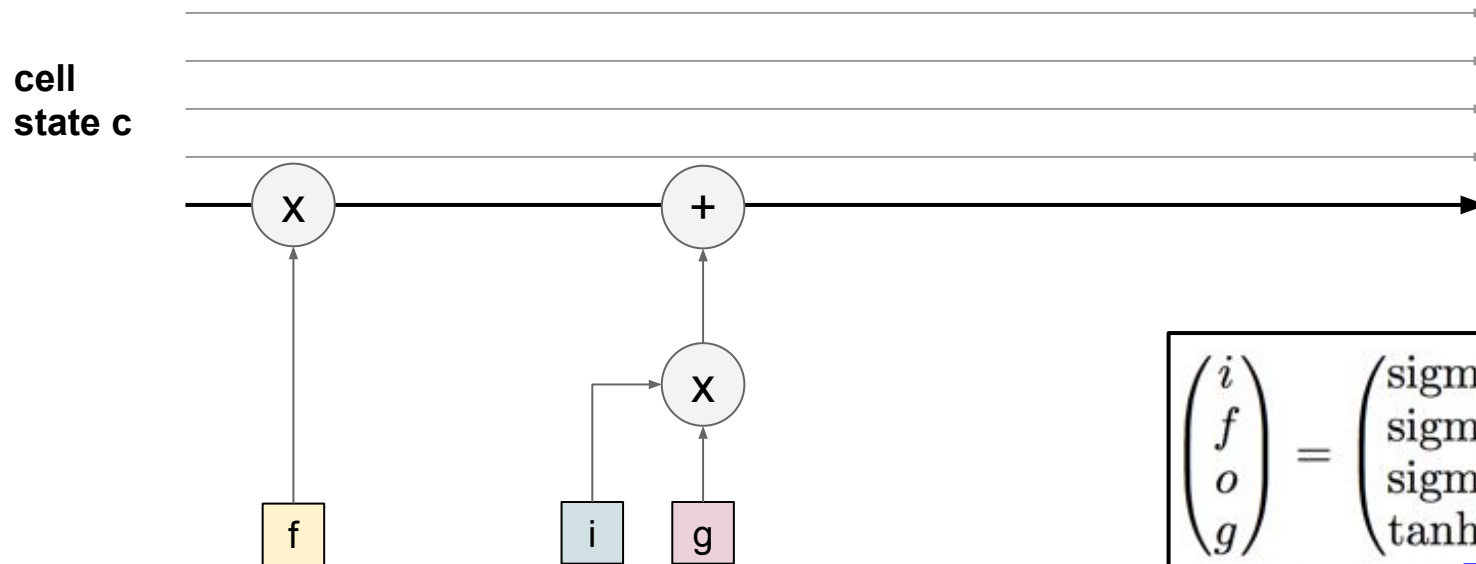
[Hochreiter et al., 1997]



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

Long Short Term Memory (LSTM)

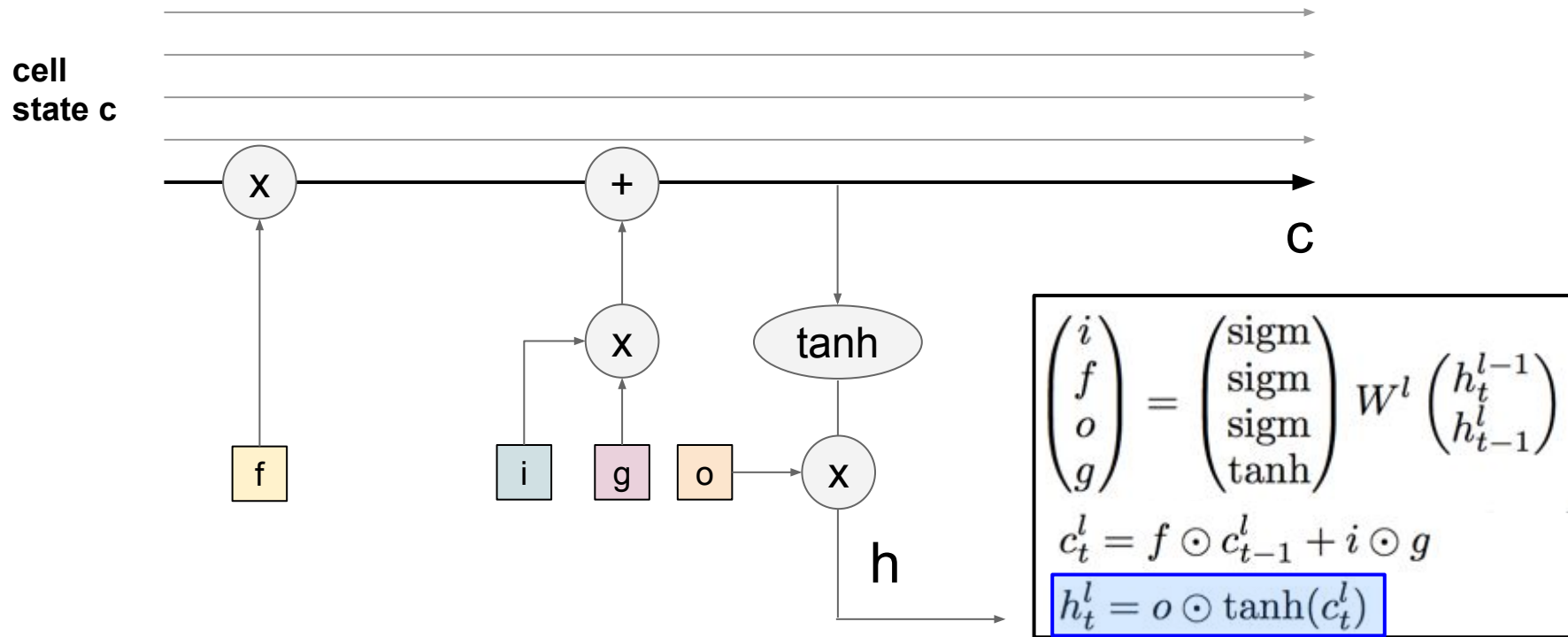
[Hochreiter et al., 1997]



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

Long Short Term Memory (LSTM)

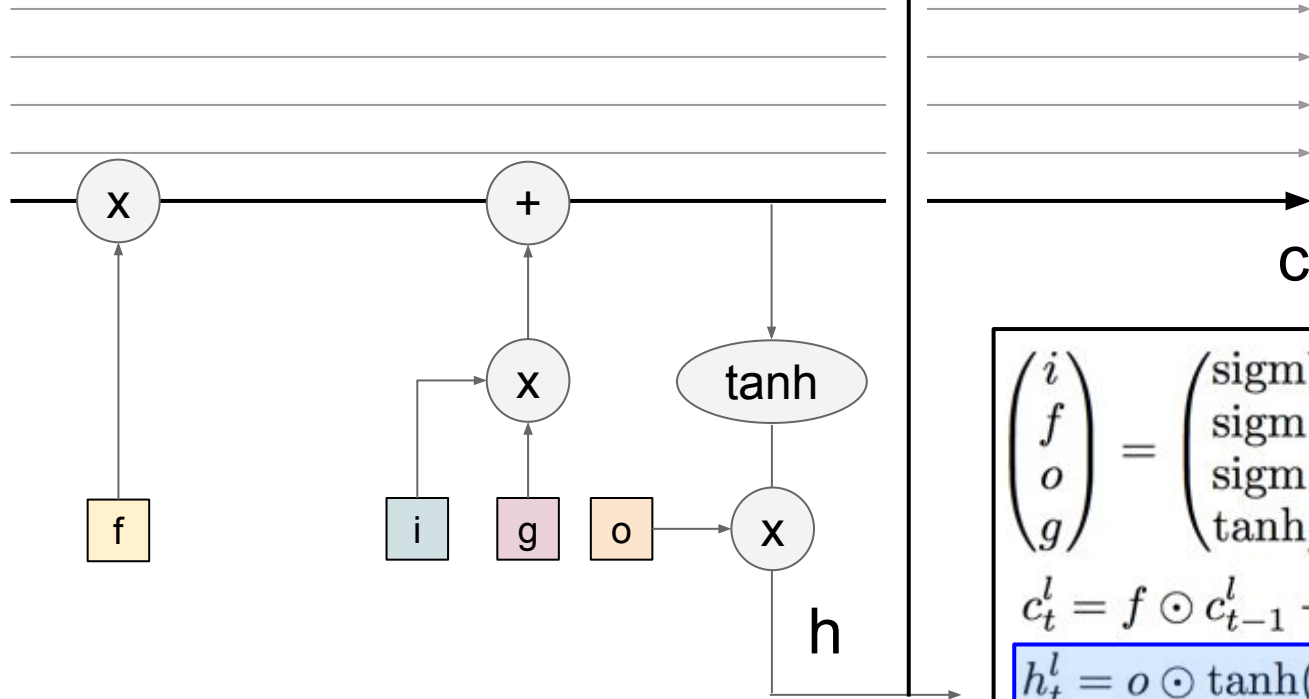
[Hochreiter et al., 1997]



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c



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