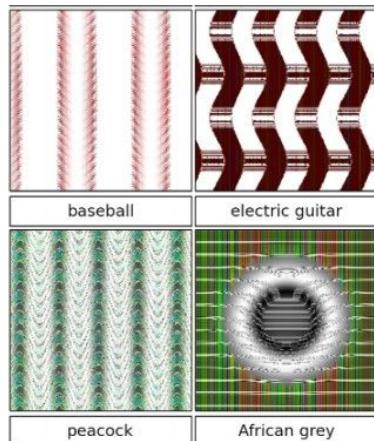
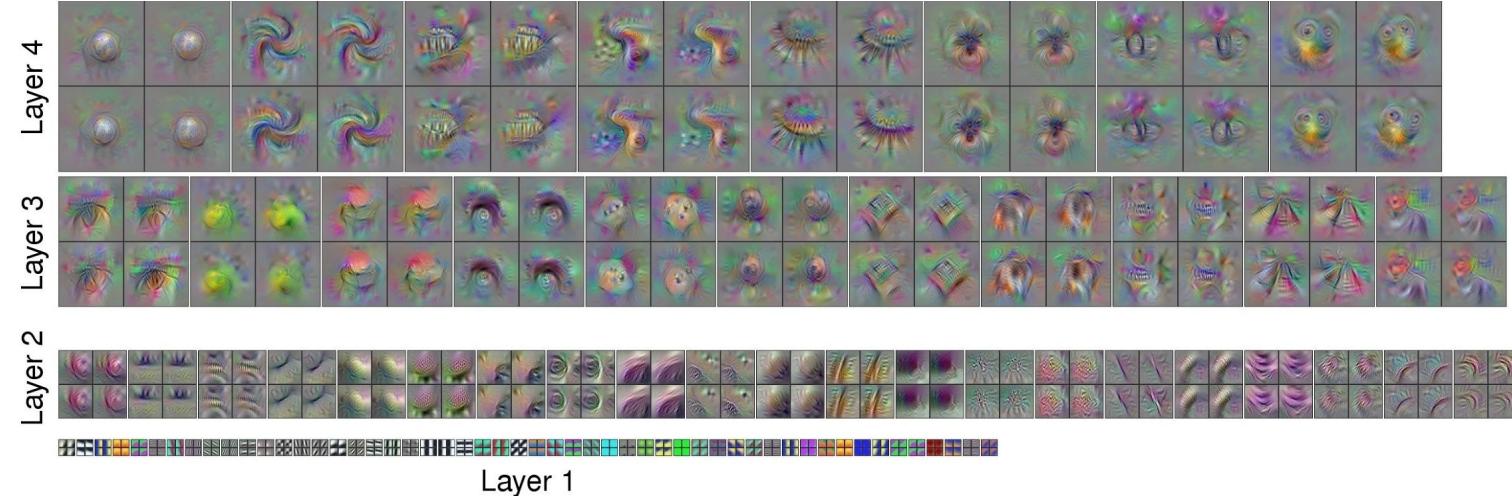
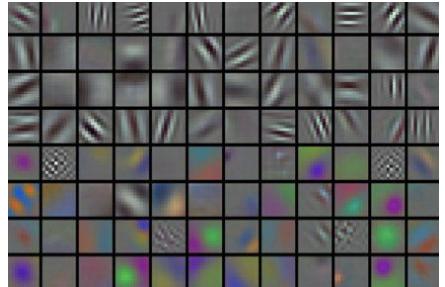


Lecture 10:

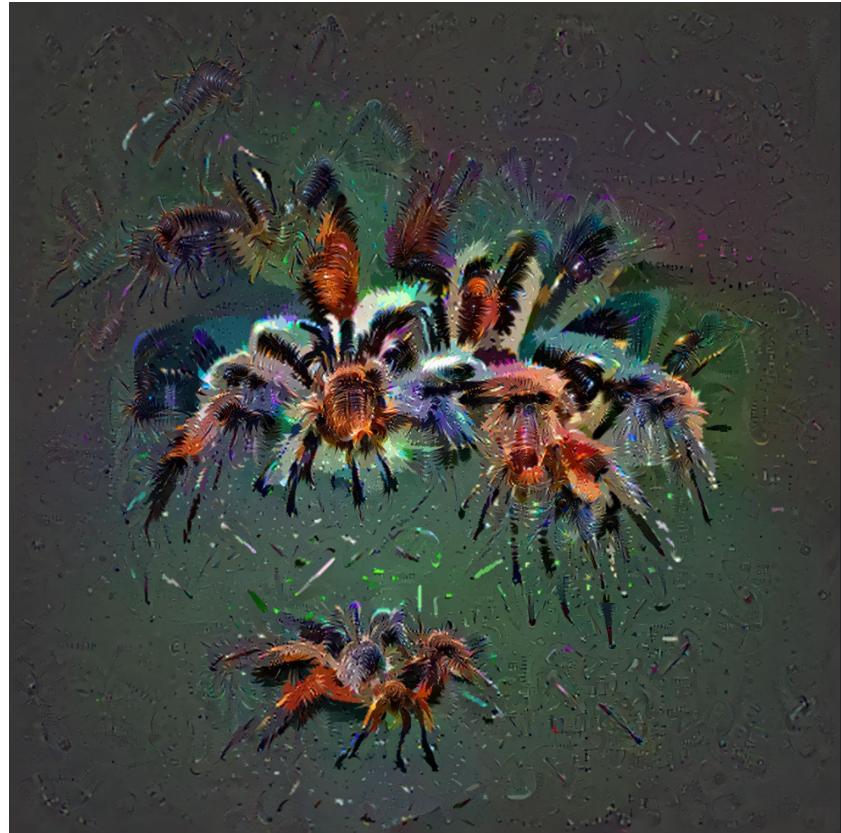
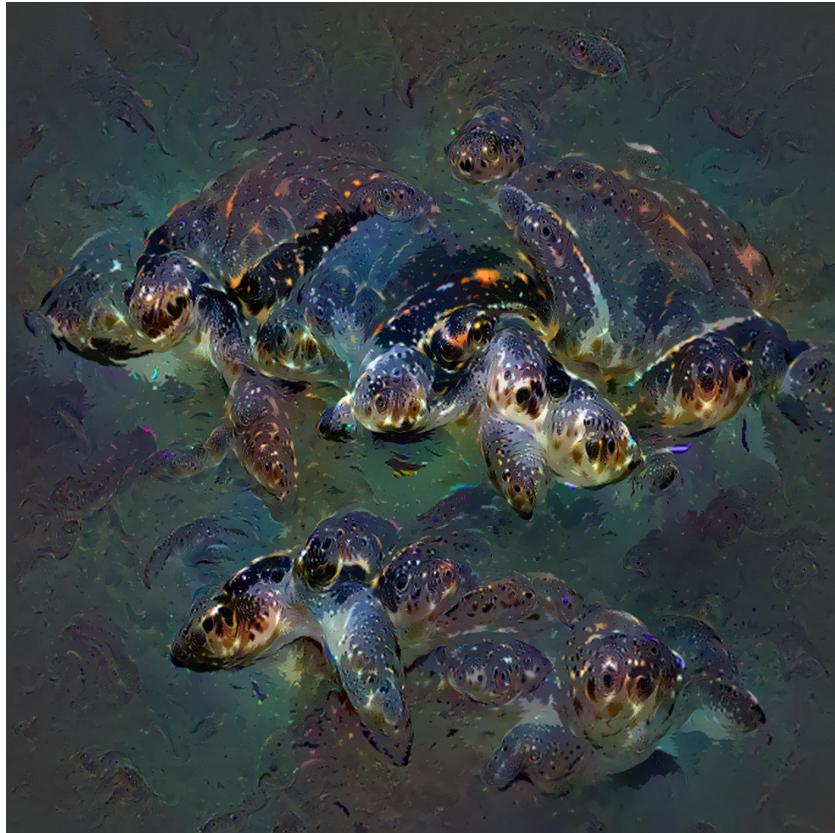
Recurrent Neural Networks

Administrative

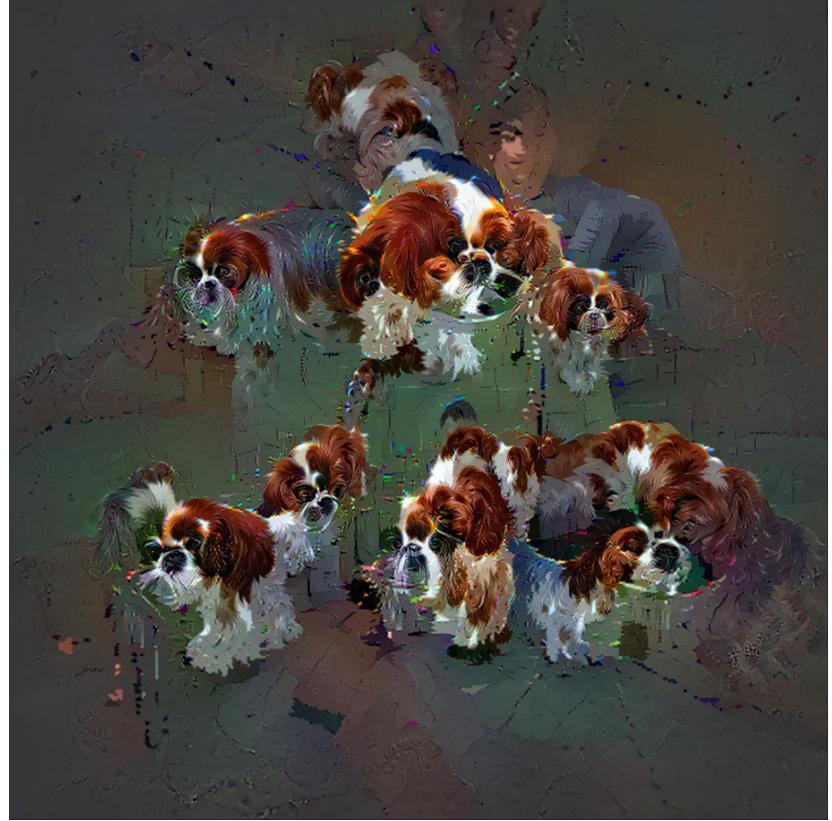
- Midterm this Wednesday! woohoo!
- A3 will be out ~Wednesday



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

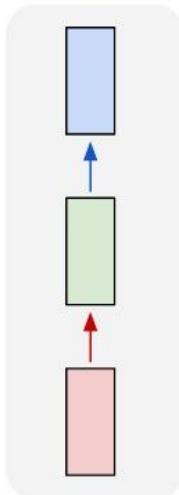


<http://mtyka.github.io/deepdream/2016/02/05/bilateral-class-vis.html>

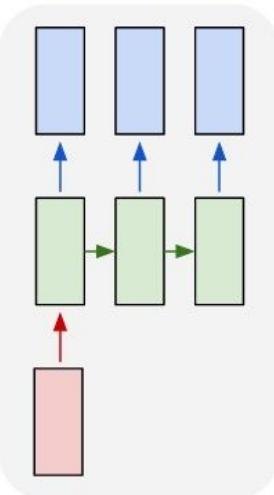


Recurrent Networks offer a lot of flexibility:

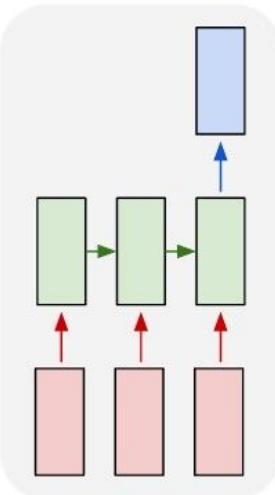
one to one



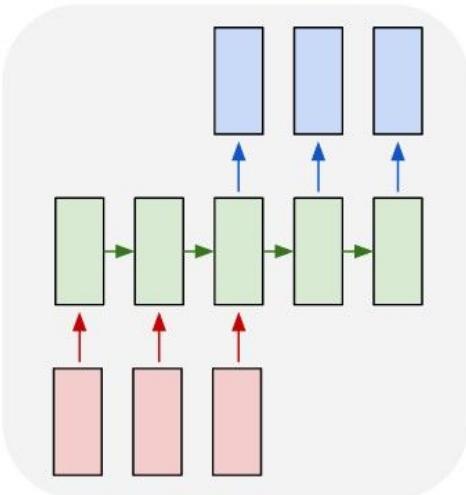
one to many



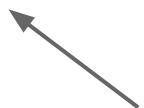
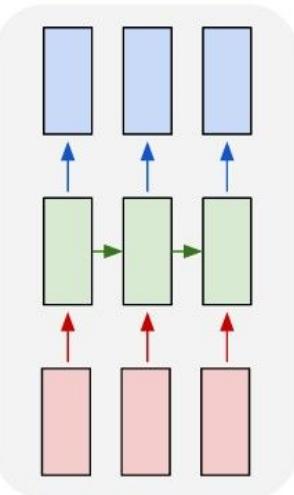
many to one



many to many



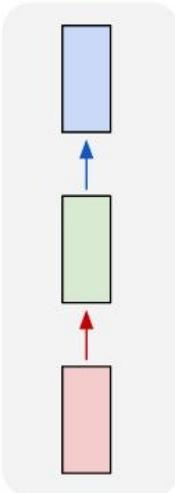
many to many



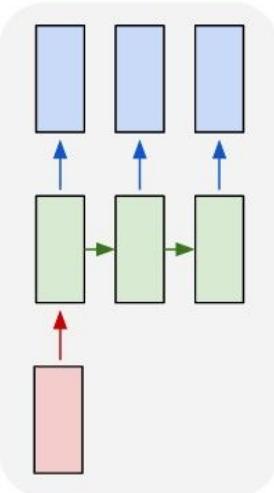
Vanilla Neural Networks

Recurrent Networks offer a lot of flexibility:

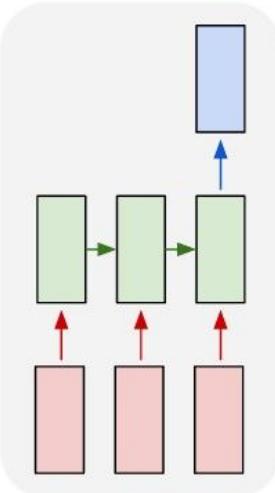
one to one



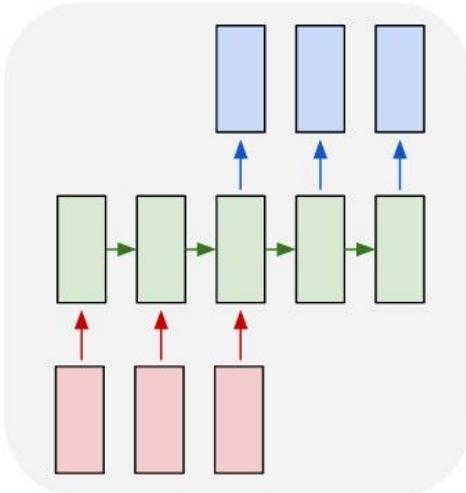
one to many



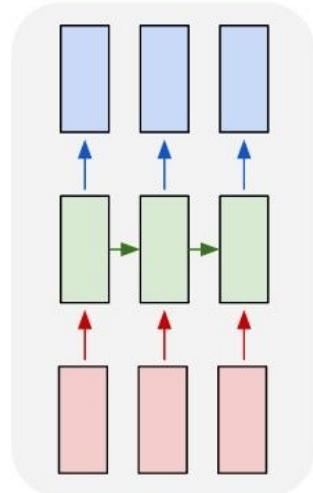
many to one



many to many



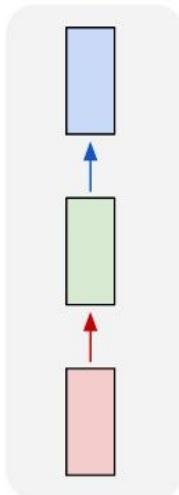
many to many



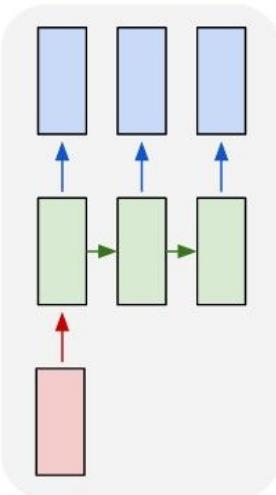
→ e.g. **Image Captioning**
image -> sequence of words

Recurrent Networks offer a lot of flexibility:

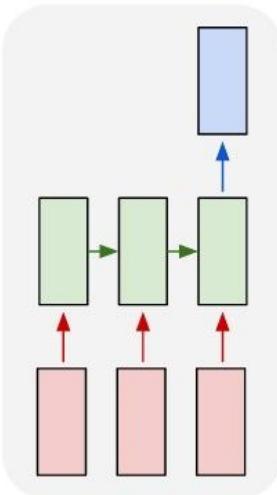
one to one



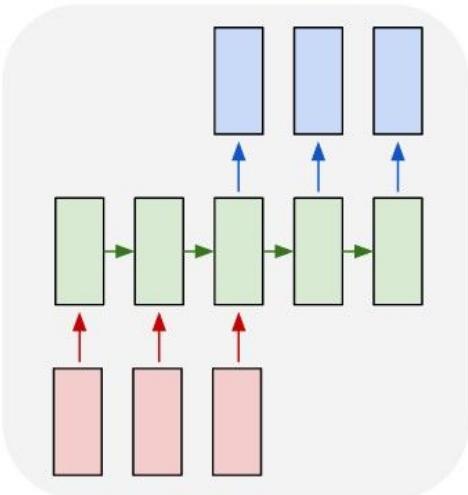
one to many



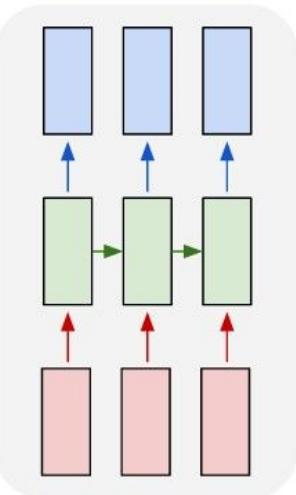
many to one



many to many



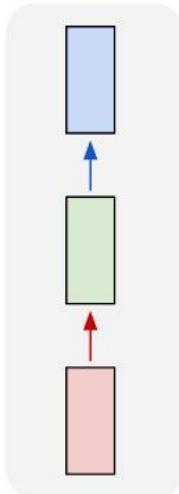
many to many



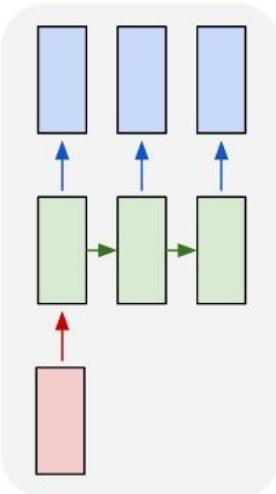
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:

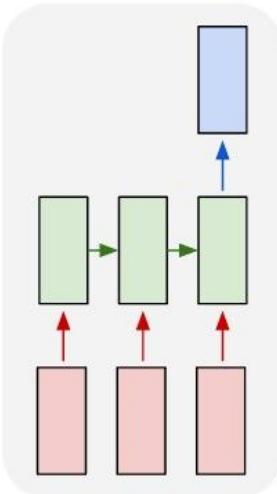
one to one



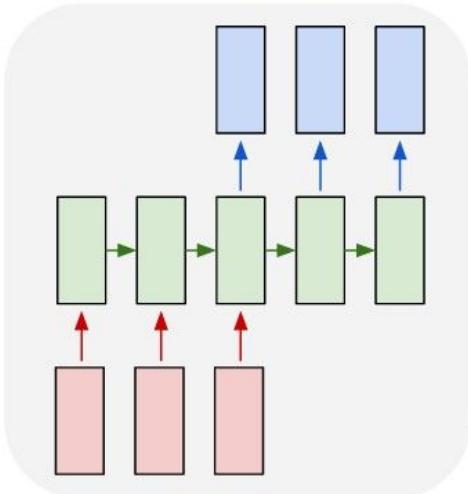
one to many



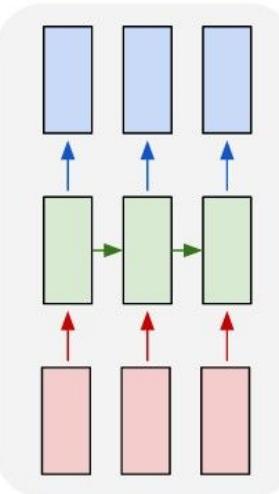
many to one



many to many



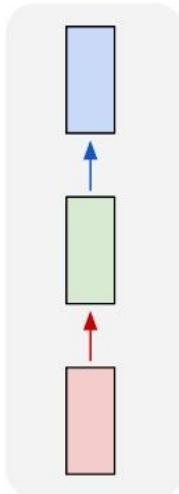
many to many



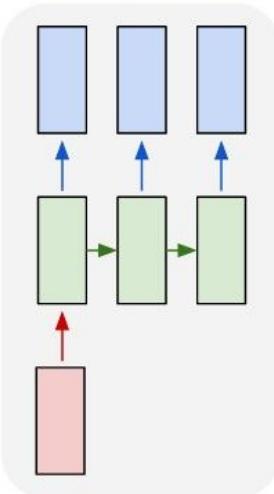
↑
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Networks offer a lot of flexibility:

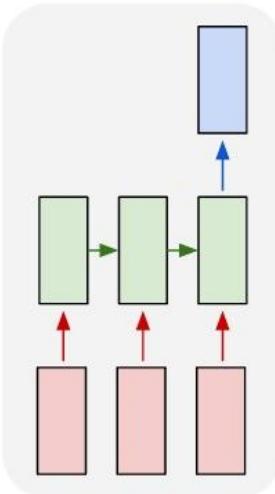
one to one



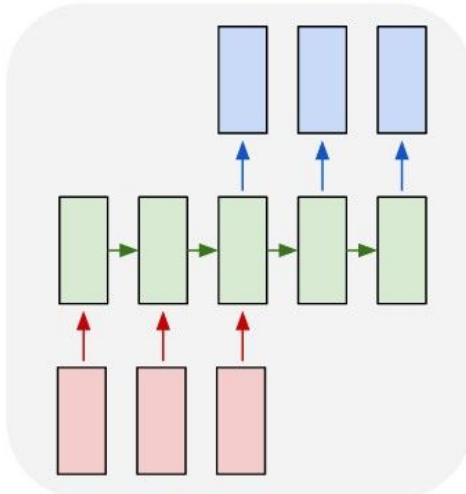
one to many



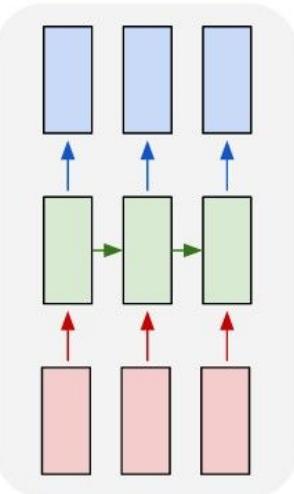
many to one



many to many



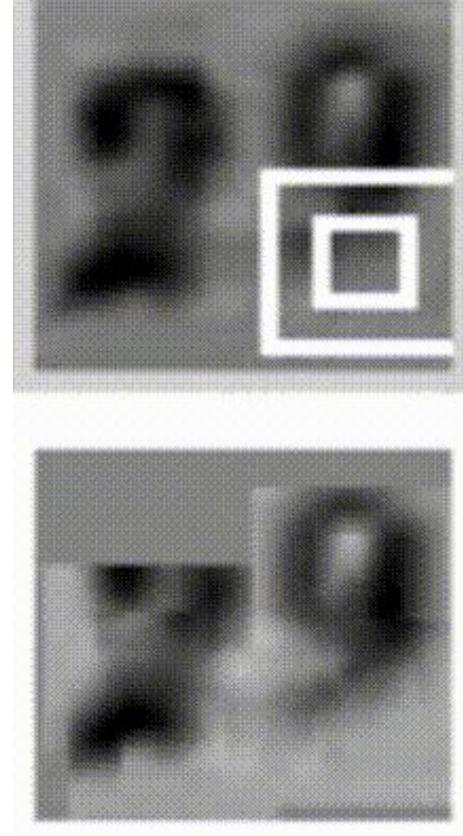
many to many



e.g. Video classification on frame level

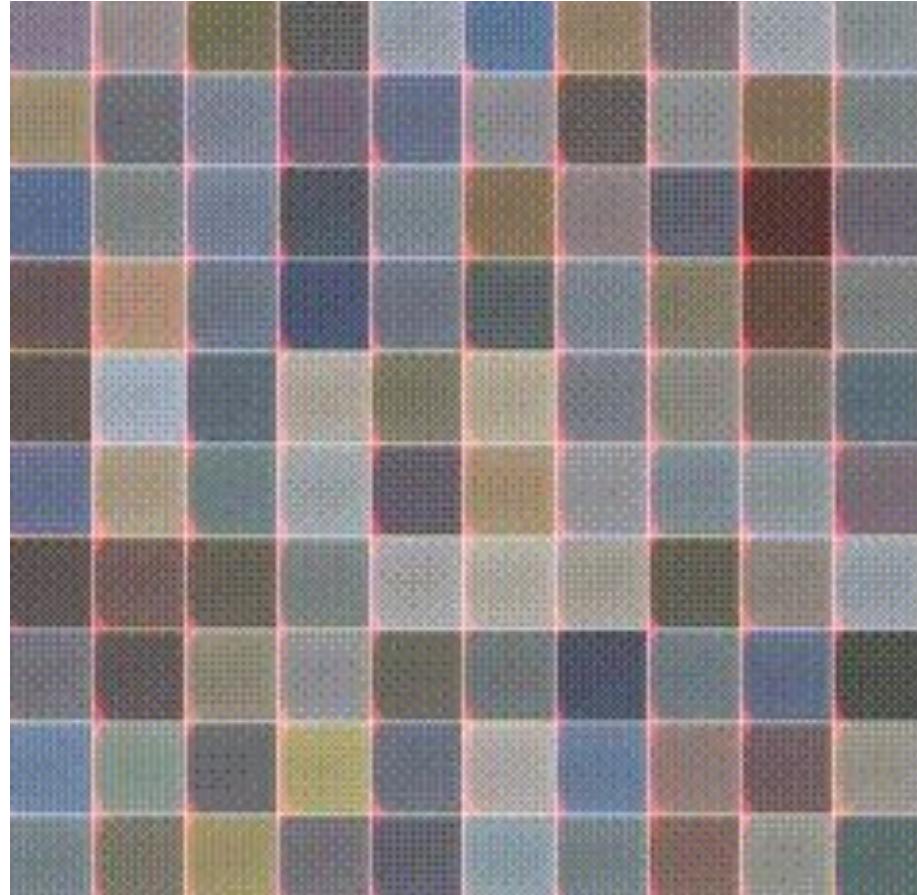
Sequential Processing of fixed inputs

Multiple Object Recognition with
Visual Attention, Ba et al.

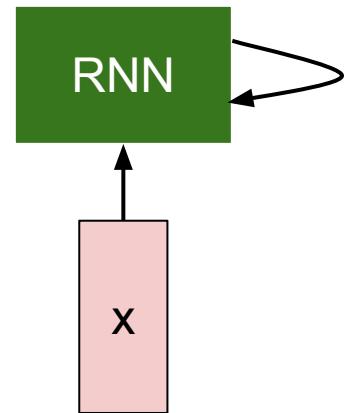


Sequential Processing of fixed outputs

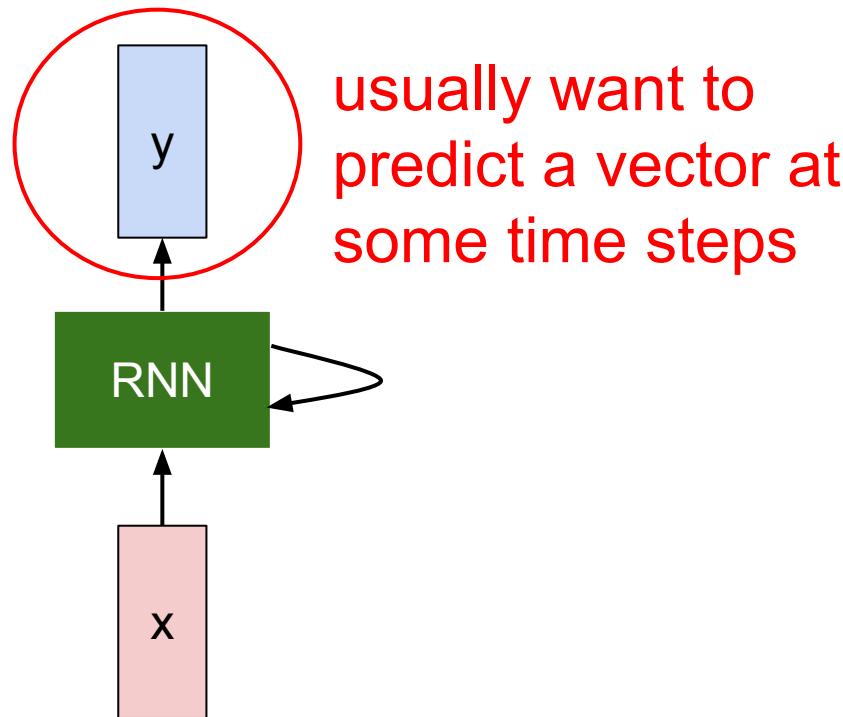
DRAW: A Recurrent
Neural Network For
Image Generation,
Gregor et al.



Recurrent Neural Network



Recurrent Neural Network

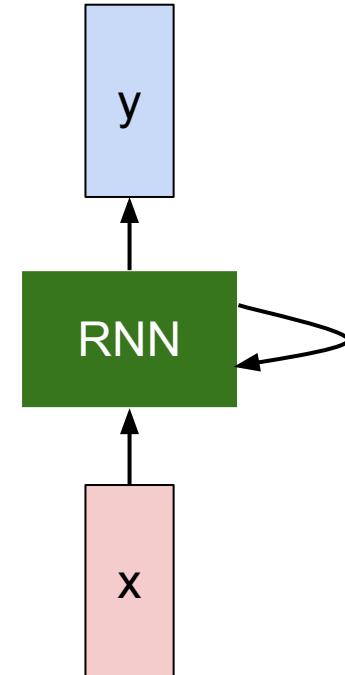


Recurrent Neural Network

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

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

new state / old state input vector at
some function | some time step
with parameters W

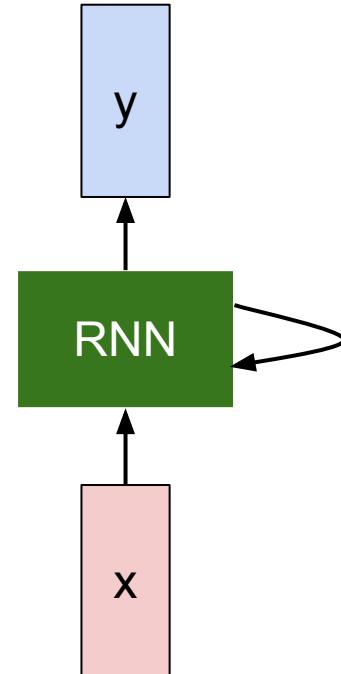


Recurrent Neural Network

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

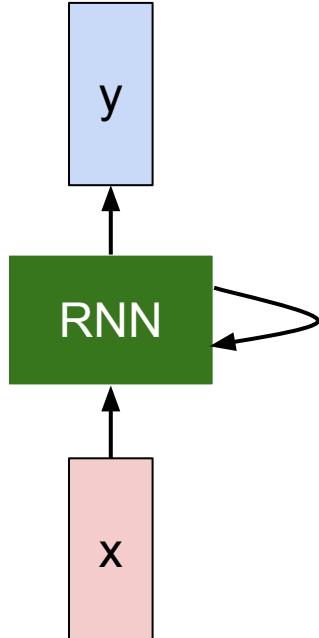
$$h_t = f_W(h_{t-1}, x_t)$$

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



(Vanilla) Recurrent Neural Network

The state consists of a single “*hidden*” vector \mathbf{h} :



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



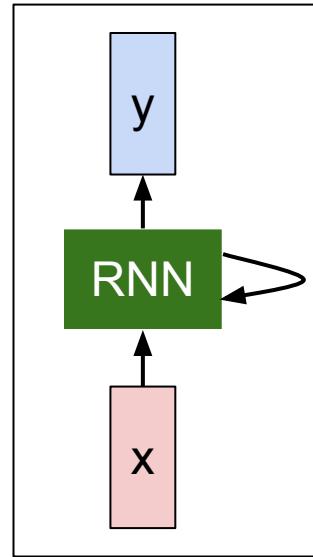
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Character-level language model example

Vocabulary:
[h,e,l,o]

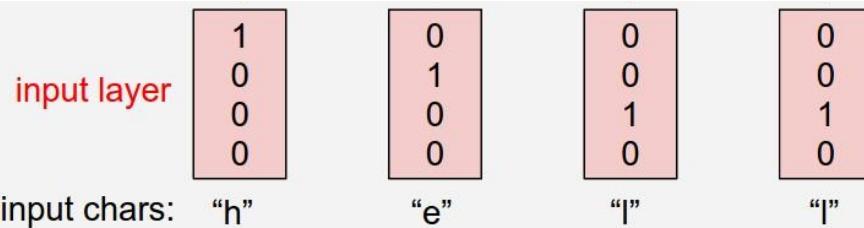
Example training
sequence:
“hello”



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

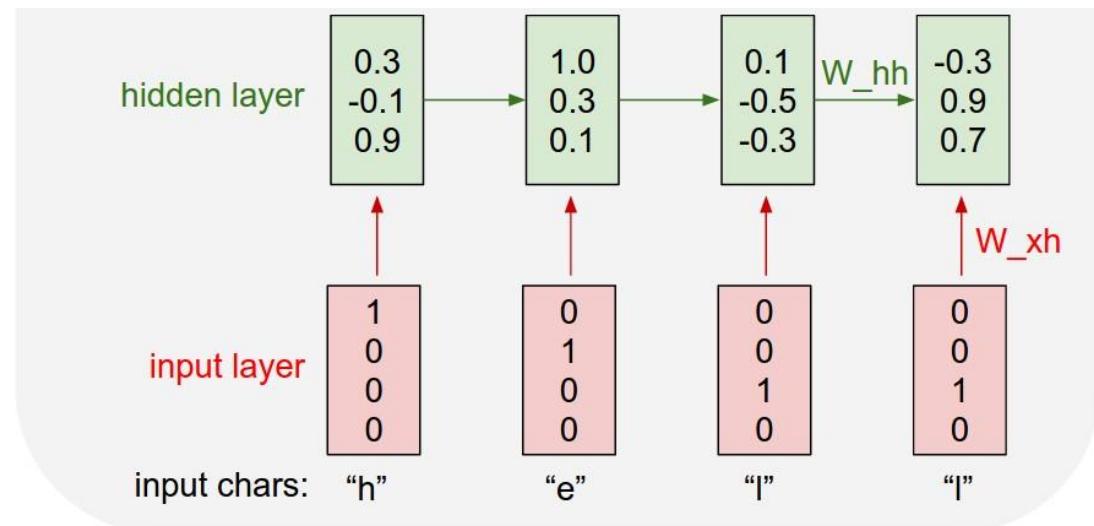


Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

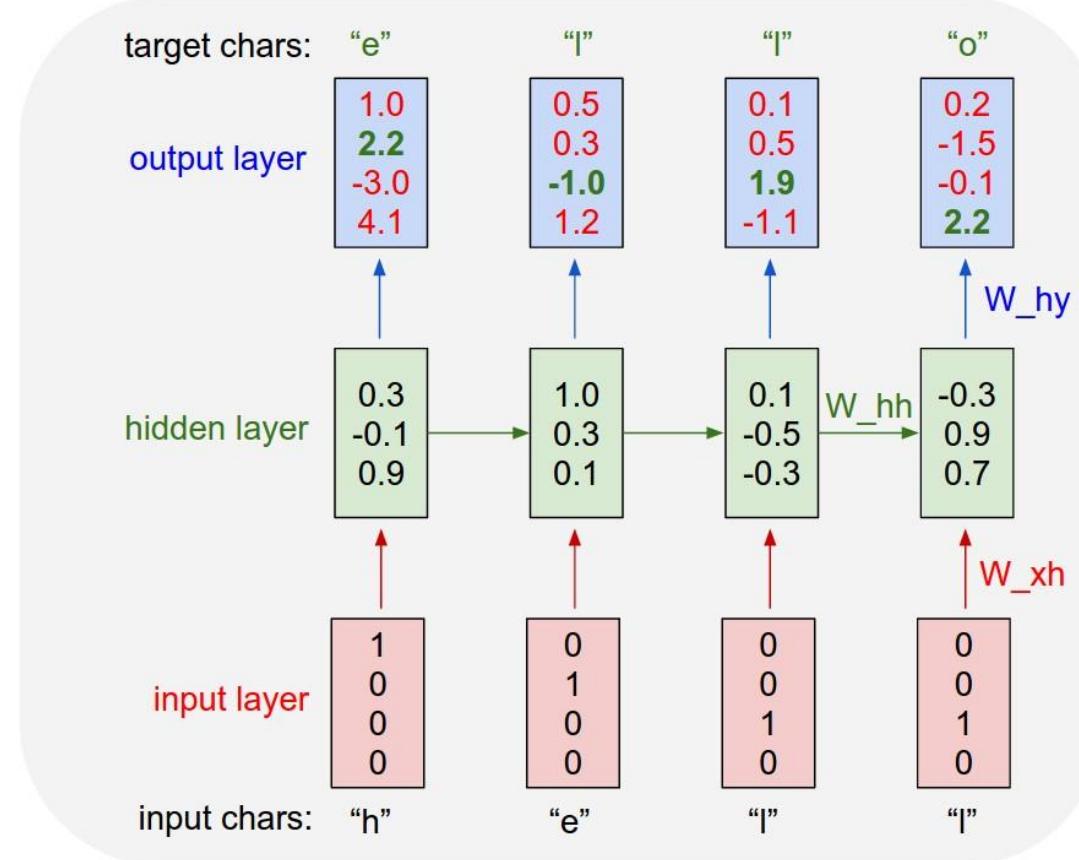
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”



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

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  # data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print('data has %d characters, %d unique.' % (data_size, vocab_size))
12 char_to_ix = {ch:i for i,ch in enumerate(chars)}
13 ix_to_char = {i:ch for i,ch in enumerate(chars)}
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wkh = np.random.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 Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(wkh, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t]]) # softmax (cross-entropy loss)
44
45         # backward pass: compute gradients going backwards
46         dwhh, dwhy = np.zeros_like(whh), np.zeros_like(why)
47         dbh, dby = np.zeros_like(bh), np.zeros_like(by)
48         dnext = np.zeros_like(hs[0])
49         for t in reversed(xrange(len(inputs))):
50             dy = np.copy(ps[t])
51             dy[targets[t]] -= 1 # backprop into y
52             dby += np.dot(dy, hs[t].T)
53             dh = np.dot(why.T, dy) + dnext # backprop into h
54             ddraw = (i - hs[t].T) * dh # backprop through tanh nonlinearity
55             dbh += ddraw
56             dwhh += np.dot(ddraw, xs[t].T)
57             dwhy += np.dot(ddraw, hs[t-1].T)
58             dnext = np.dot(whh.T, ddraw)
59             for dparam in [dwhh, dwhy, dbh, dby]:
60                 np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61
62     return loss, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
```

```
63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     x[seed_ix] = 1
70     ixes = []
71     for t in xrange(n):
72         h = np.tanh(np.dot(wkh, x) + np.dot(whh, h) + bh)
73         y = np.dot(why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p.ravel())
76         x = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79
80     return ixes
81
82 n, p = 0, 0
83 mxwh, mwhh, mmwh = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
84 mbh, mbw, mbby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
85 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
86 while True:
87     # prepare inputs (we're sweeping from left to right in steps seq_length long)
88     if p+seq_length+1 >= len(data) or n == 0:
89         hprev = np.zeros((hidden_size,1)) # reset RNN memory
90         p = 0 # go from start of data
91     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
92     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
93
94     # sample from the model now and then
95     if n % 100 == 0:
96         sample_ix = sample(hprev, inputs[0], 200)
97         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
98         print '----\n' + txt + '----'
99
100     # forward seq_length characters through the net and fetch gradient
101     loss, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
102     smooth_loss = smooth_loss * 0.999 + loss * 0.001
103     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
104
105     # perform parameter update with Adagrad
106     for param, dparam, mem in zip([wkh, whh, why, bh, by],
107                                   [dwhh, dwhy, dbh, dby],
108                                   [mxwh, mwhh, mmwh, mbh, mbw]):
109         mem += dparam * dparam
110         param += -learning_rate * param / np.sqrt(mem + 1e-8) # adagrad update
111
112         p += seq_length # move data pointer
113         n += 1 # iteration counter
```

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

Data I/O

min-char-rnn.py gist

```

1  #!/usr/bin/python
2
3  # Minimal character-level vanilla RNN model. Written by Andrej Karpathy (@karpathy)
4
5  # BSD License
6
7  # http://karpathy.github.io/char-rnn/
8
9  import numpy as np
10
11  # data I/O
12  data = open('input.txt', 'r').read() # should be simple plain text file
13  chars = list(set(data))
14  data_size, vocab_size = len(data), len(chars)
15  print 'data has %d characters, %d unique' % (data_size, vocab_size)
16  start_index = 0
17  step = 10
18
19  ix_to_char = { i:ch for i, ch in enumerate(chars) }
20
21  # inputs and outputs
22  x = np.zeros((step, data_size), dtype=np.float32)
23  y = np.zeros((step, vocab_size), dtype=np.float32)
24
25  for i in range(0, data_size - step, step):
26      x[0] = np.zeros(data_size)
27      for j in range(i, i+step):
28          x[0][j] = 1.0
29      y[0] = np.zeros(vocab_size)
30      y[0][ix_to_char[chars[j]]] = 1.0
31
32      for k in range(1, step):
33          x[k] = np.zeros(data_size)
34          for j in range(i+k-1, i+k):
35              x[k][j] = 1.0
36          y[k] = np.zeros(vocab_size)
37          y[k][ix_to_char[chars[j]]] = 1.0
38
39  print x[0]
40  print y[0]
41
42  # hyperparameters
43  hidden_size = 100 # size of hidden layer
44  learning_rate = 1e-1
45
46  # model parameters
47  xavier_bound = np.sqrt(6.0 / (data_size + hidden_size))
48  Wxh = np.random.uniform(-xavier_bound, xavier_bound, (data_size, hidden_size))
49  Whh = np.random.uniform(-xavier_bound, xavier_bound, (hidden_size, hidden_size))
50  Whc = np.random.uniform(-xavier_bound, xavier_bound, (hidden_size, vocab_size))
51
52  # forward pass
53  def forward(x, h0, Wxh, Whh, Whc, bias):
54      h1 = np.tanh(x.dot(Wxh) + h0.dot(Whh) + bias)
55      y1 = np.exp(h1.dot(Whc))
56      return y1, h1
57
58  # backward pass
59  def backward(y, dh1, Wxh, Whh, Whc, x, h0, h1):
60      dh0 = dh1.dot(Whh.T)
61      dh1 = dh1 * (1 - h1 * h1)
62      dx = dh1.dot(Wxh.T)
63      dhc = dh1.dot(Whc.T)
64
65      return dh0, dx, dhc, h1
66
67  # training loop
68  for i in range(100000):
69      if i % 10000 == 0:
70          print 'step %d' % i
71
72      # forward pass
73      y, h1 = forward(x, h0, Wxh, Whh, Whc, np.zeros(hidden_size))
74
75      # backward pass
76      dh0, dx, dhc, h1 = backward(y, dh1, Wxh, Whh, Whc, x, h0, h1)
77
78      # update parameters
79      h0 -= learning_rate * dh0
80      Wxh -= learning_rate * dx
81      Whh -= learning_rate * dh1
82      Whc -= learning_rate * dhc
83
84      # print progress
85      if i % 1000 == 0:
86          print ' '.join([ix_to_char[int(np.argmax(y[0]))]]),
87
88  print 'done'

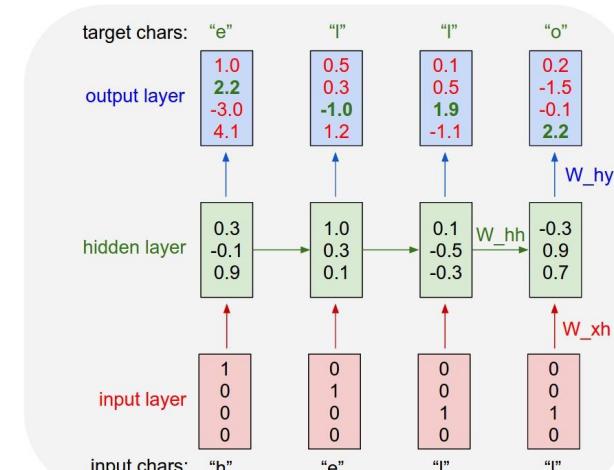
```

```
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) }
```

Initializations

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

recall



min-char-rnn.py gist

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  # Data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(set(data))
10 vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = {ch:i for i in range(len(chars))}
13 ix_to_char = {i:ch for ch in range(len(chars))}
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 20 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
22 dbh, mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
23 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
24
25 while True:
26
27     # prepare inputs (we're sweeping from left to right in steps seq_length long)
28     if p+seq_length+1 >= len(data) or n == 0:
29         hprev = np.zeros((hidden_size,1)) # reset RNN memory
30         p = 0 # go from start of data
31         inputs = [c for c in data[p:p+seq_length]]
32         targets = [c for c in data[p+1:p+seq_length+1]]
33
34     # forward pass
35     for t in range(seq_length):
36         x = np.zeros((vocab_size,1)) # encode in 1-of-k representation
37         x[inputs[t]:inputs[t]+1] = 1
38
39         h1t = np.tanh(np.dot(wh, x1t) + np.dot(bh, h1t-1) + bh) # hidden state
40         y1t = np.dot(Wy, h1t) + by # compute output
41         y1t = np.exp(y1t) / np.sum(np.exp(y1t)) = softmax # probabilities for next chars
42         loss += -np.log(y1t[targets[t]]) # softmax (cross-entropy loss)
43
44         # backward pass: compute gradients
45         dWxh += np.dot(x1t, h1t.T) # backward pass: compute gradients
46         dbh += np.zeros_like(bh) # backward pass: compute gradients
47         dWhh += np.dot(h1t, h1t.T) # backward pass: compute gradients
48         dWhy += np.dot(h1t, np.zeros_like(by)) # backward pass: compute gradients
49
50         for dparam in [dWxh, dbh, dWhh, dWhy]:
51             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
52
53         dh1t = np.dot(Wy.T, dy1t) # backward pass: compute gradients
54         dh1t -= np.dot(Wxh.T, dy1t) # dh1t = dh1t - dWxh.T * dy1t
55         dh1t -= np.dot(Whh.T, dh1t) # dh1t = dh1t - dWhh.T * dh1t
56         dh1t *= np.tanh(dh1t) # tanh nonlinearity
57         dWxh += np.dot(x1t, dh1t.T) # backward pass: compute gradients
58         dbh += np.zeros_like(bh) # backward pass: compute gradients
59         dWhh += np.dot(h1t, dh1t.T) # backward pass: compute gradients
60
61         for dparam in [dWxh, dbh, dWhh, dWhy]:
62             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
63
64         dh1t = np.dot(Wy.T, dy1t) # backward pass: compute gradients
65         dh1t -= np.dot(Wxh.T, dy1t) # dh1t = dh1t - dWxh.T * dy1t
66         dh1t -= np.dot(Whh.T, dh1t) # dh1t = dh1t - dWhh.T * dh1t
67         dh1t *= np.tanh(dh1t) # tanh nonlinearity
68
69         for t in range(seq_length):
70             x1t = np.zeros((vocab_size,1))
71             x1t[inputs[t]:inputs[t]+1] = 1
72
73             for t in range(seq_length):
74                 h1t = np.tanh(np.dot(wh, x1t) + np.dot(bh, h1t-1) + bh)
75                 y1t = np.dot(Wy, h1t) + by
76                 p = np.exp(y1t) / np.sum(np.exp(y1t))
77                 ix = np.random.choice(range(vocab_size), p=p.ravel())
78                 x1t = np.zeros((vocab_size,1))
79                 x1t[ix] = 1
80
81             return ix
82
83     # sample from the model, now and then
84     if n % 100 == 0:
85         sample_ix = sample(hprev, inputs[0], 200)
86         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
87         print '----\n %s ----' % (txt, )
88
89     # forward seq_length characters through the net and fetch gradient
90     loss, dWxh, dWhh, dWhy, dbh, dy, hprev = lossFun(inputs, targets, hprev)
91     smooth_loss = smooth_loss * 0.999 + loss * 0.001
92
93     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
94
95     # perform parameter update with Adagrad
96     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
97                                   [dWxh, dWhh, dWhy, dbh, dyb],
98                                   [mWxh, mWhh, mWhy, mbh, mby]):
99
100         mem += dparam * dparam
101         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
102
103         p += seq_length # move data pointer
104
105         n += 1 # iteration counter
106
107
108
109
110
111
112
```

Main loop



min-char-rnn.py gist

Main loop

min-char-rnn.py gist

```
1  ***
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  ***
5  Import numpy as np
6
7  # Data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(set(data))
10 vocab_size = len(chars)
11 data_size, vocab_size = len(data), len(chars)
12 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
13 char_to_ix = {ch:i for i in range(len(chars))}
14 ix_to_char = {i:ch for ch in range(len(chars))}
15
16 # hyperparameters
17 hidden_size = 100 # size of hidden layer of neurons
18 seq_length = 20 # number of steps to unroll the RNN for
19 learning_rate = 1e-1
20
21 model_params = {}
22
23 def init_params():
24     hidden_size = 100 # size of hidden layer of neurons
25     vocab_size = len(chars)
26     seq_length = 20 # number of steps to unroll the RNN for
27     learning_rate = 1e-1
28
29     mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
30     mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
31     smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
32
33     while True:
34         if p+seq_length+1 >= len(data) or n == 0:
35             hprev = np.zeros((hidden_size,1)) # reset RNN memory
36             p = 0 # go from start of data
37             inputs, targets = [char_to_ix[ch] for ch in data[p:p+seq_length]]
38             targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
39
40             # sample from the model now and then
41             if n % 100 == 0:
42                 sample_ix = sample(hprev, inputs[0], 200)
43                 txt = ''.join(ix_to_char[ix] for ix in sample_ix)
44                 print '----\n %s ----' % (txt, )
45
46             # forward seq_length characters through the net and fetch gradient
47             loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
48             smooth_loss = smooth_loss * 0.999 + loss * 0.001
49
50             if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
51
52             # perform parameter update with Adagrad
53             for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
54                                         [dWxh, dWhh, dWhy, dbh, dby],
55                                         [mWxh, mWhh, mWhy, mbh, mby]):
56                 mem += dparam * dparam
57                 param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
58
59             p += seq_length # move data pointer
60             n += 1 # iteration counter
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
```



Main loop

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

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

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

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

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

min-char-rnn.py gist

Main loop

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  # Data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(data[1:-1])
10 vocab_size = len(data), len(chars)
11 print('data has %d characters, %d unique.' % (data_size, vocab_size))
12 char_to_ix = {ch: i for i in range(len(chars))}
13 ix_to_char = {i: ch for ch in range(len(chars))} # 14
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 20 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # Model parameters
21 dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
22 dbh, mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
23 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
24
25 while True:
26
27     # prepare inputs (we're sweeping from left to right in steps seq_length long)
28     if p+seq_length+1 >= len(data) or n == 0:
29         hprev = np.zeros((hidden_size,1)) # reset RNN memory
30         p = 0 # go from start of data
31
32     inputs, targets = inputs[0], targets[0]
33
34     # forward pass
35     for t in range(seq_length):
36         inputs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
37         inputs[t][char_to_ix[inputs[t]]] = 1
38
39         h1t = np.tanh(np.dot(wh, x1t) + np.dot(bh, h1t-1) + bh) # hidden state
40         y1t = np.dot(why, h1t) + by # output
41         y1t = np.exp(y1t) / np.sum(np.exp(y1t)) = softmax # probabilities for next chars
42         loss += -np.log(y1t[targets[t]]) # softmax (cross-entropy loss)
43
44         # backward pass: compute gradients
45         dWxh += np.outer(inputs[t], h1t) # input gradients to hidden
46         dbh += np.zeros_like(bh), np.zeros_like(why), np.zeros_like(why)
47         dWhh += np.outer(h1t, h1t) # hidden gradients to hidden
48         dWhy += np.outer(h1t, np.zeros_like(why))
49
50     for t in range(seq_length-1, -1, -1):
51         dy = np.copy(dy[t])
52         dy[t] = dy[t] + dWxh[inputs[t]] # a backward pass into y
53         dh = np.dot(Wxh.T, dy) + dWhh # dh = backprop into tanh nonlinearity
54         dh1t = np.tanh(dh) # dh1t = dh * backprop through tanh nonlinearity
55         dh1t *= dh1t
56         dWxh += np.outer(inputs[t], dh1t) # dh = backprop through dot product
57         dbh += np.zeros_like(bh), np.zeros_like(why), np.zeros_like(why)
58         dWhh += np.outer(dh1t, dh1t) # dh = backprop through dot product
59         dWhy += np.outer(dh1t, np.zeros_like(why))
60
61     for opname in [dWxh, dbh, dWhh, dWhy]:
62         np.clip(opname, -5, 5, out=opname) # clip to mitigate exploding gradients
63
64     for opname in [dbh, dWhh, dWhy]:
65         opname = opname / np.sqrt(np.sum(opname**2)) # divide by norm
66
67     dbh, dWhh, dWhy = dbh, dWhh, dWhy, np.zeros_like(why)
68
69     if n % 100 == 0:
70         sample_ix = sample(hprev, inputs[0], 200)
71         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
72         print '----\n %s \n----' % (txt, )
73
74
75     # forward seq_length characters through the net and fetch gradient
76     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
77     smooth_loss = smooth_loss * 0.999 + loss * 0.001
78
79     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
80
81     n, p = 0, 0
82
83     # perform parameter update with Adagrad
84     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
85                                   [dWxh, dWhh, dWhy, dbh, dby],
86                                   [mWxh, mWhh, mWhy, mbh, mby]):
87
88         mem += dparam * dparam
89         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
90
91     p += seq_length # move data pointer
92
93     n += 1 # iteration counter
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
```

min-char-rnn.py gist

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  # Data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(set(data))
10 vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = {ch:i for i in range(len(chars))}
13 ix_to_char = {i:ch for ch in range(len(chars))}
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 20 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
22 dbh, mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
23 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
24
25 while True:
26
27     # prepare inputs (we're sweeping from left to right in steps seq_length long)
28     if p+seq_length+1 >= len(data) or n == 0:
29         hprev = np.zeros((hidden_size,1)) # reset RNN memory
30         p = 0 # go from start of data
31
32     inputs, targets = inputs[0], targets[0]
33
34     # forward pass
35     for t in range(seq_length):
36         x = np.zeros((vocab_size,1)) # encode in 1-of-k representation
37         x[inputs[t]:inputs[t]+1] = 1
38
39         hprev = np.tanh(np.dot(Whh, hprev) + np.dot(Wxh, x) + bh) # hidden state
40
41         y = np.dot(Why, hprev) + by # output
42         yprob = np.exp(y) / np.sum(np.exp(y)) # probabilities for next chars
43
44         loss += -np.log(yprob[targets[t]]) # softmax (cross-entropy loss)
45
46         # backward pass: compute gradients
47         dWxh += np.dot(yprob, x.T) # backward into x
48         dWhh += np.dot(yprob, hprev.T) # backward into h
49         dWhy += np.dot(yprob, np.zeros((vocab_size,1))) # backward into y
50
51         dh = np.dot(Why.T, dy) + dWhh # backward into h
52         dh += dy # backward into dh
53         dh *= np.tanh(dh.T) * (1 - dh**2) # tanh nonlinearity
54         dWhh += np.dot(dy, dh.T)
55         dWxh += np.dot(dy, x.T)
56         dbh += np.sum(dy, axis=0, keepdims=True) / seq_length
57         dWhy += np.sum(dyprob, axis=0, keepdims=True) / seq_length
58
59     for dparam in [dWxh, dWhh, dbh, dWhy]:
60         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61
62     dbh, dWhh, dWxh, dWhy, dy, dh = dh[seq_length-1]
63
64     if sample:
65         seed_ix = np.random.choice(range(vocab_size))
66
67     else:
68         seed_ix = ix_to_char[p]
69
70     for t in range(seq_length):
71         h = np.tanh(dh[0], x) + np.dot(Whh, h) + bh
72         y = np.dot(Why, h) + by
73         p = np.exp(y) / np.sum(np.exp(y))
74         ix = np.random.choice(range(vocab_size), p=p.ravel())
75         x = np.zeros((vocab_size,1))
76         x[ix] = 1
77         x[seed_ix] = 0
78
79     return ixes
```



Main loop

```
81     n, p = 0, 0
82     mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
83     mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84     smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85
86     while True:
87
88         # prepare inputs (we're sweeping from left to right in steps seq_length long)
89         if p+seq_length+1 >= len(data) or n == 0:
90             hprev = np.zeros((hidden_size,1)) # reset RNN memory
91             p = 0 # go from start of data
92
93         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
94         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
95
96
97         # sample from the model now and then
98         if n % 100 == 0:
99             sample_ix = sample(hprev, inputs[0], 200)
100            txt = ''.join(ix_to_char[ix] for ix in sample_ix)
101            print '----\n %s \n----' % (txt, )
102
103
104         # forward seq_length characters through the net and fetch gradient
105         loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
106         smooth_loss = smooth_loss * 0.999 + loss * 0.001
107
108         if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
109
110
111         # perform parameter update with Adagrad
112         for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
113                                       [dWxh, dWhh, dWhy, dbh, dby],
114                                       [mWxh, mWhh, mWhy, mbh, mby]):
115
116             mem += dparam * dparam
117             param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
118
119             p += seq_length # move data pointer
120             n += 1 # iteration counter
```

Loss function

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

```

1  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
2  BSD LICENSE
3
4  Copyright (c) 2015 Andrej Karpathy
5
6  Permission is hereby granted, free of charge, to any person obtaining a copy
7  of this software and associated documentation files (the "Software"), to deal
8  in the Software without restriction, including without limitation the rights
9  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
10   copies of the Software, and to permit persons to whom the Software is
11   furnished to do so, subject to the following conditions:
12
13   The above copyright notice and this permission notice shall be included in
14   all copies or substantial portions of the Software.
15
16   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
17   IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
18   FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
19   THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
20   LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
21   OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
22   THE SOFTWARE.
23
24  import numpy as np
25
26  # data I/O
27  f = open('input.txt', 'r').read() # should be plain text file
28  data_size = len(f)
29  vocab_size = len(set(f))
30  len(chars) = len(f)
31
32  print "data has %d characters, %d unique." % (data_size, vocab_size)
33  char_to_ix = {c: i for i, c in enumerate(chars)}
34  ix_to_char = {i: c for i, c in enumerate(chars)}
35
36  # hyperparameters
37
38  hidden_size = 200 # size of hidden layer of neurons
39  sequence_length = 25 # number of steps to unroll the RNN for
40  learning_rate = 1e-1
41
42
43  # model parameters
44
45  # m = np.zeros((hidden_size, vocab_size)) = hidden to hidden
46  wh = np.zeros((hidden_size, hidden_size)) # hidden to hidden
47  bh = np.zeros(hidden_size) # hidden bias
48  h0 = np.zeros(hidden_size) # initial hidden state
49  y0 = np.zeros(vocab_size) # initial y0
50
51  # forward pass
52
53  def forward(inputs, targets, hprev):
54
55      inputs, targets = list_of_integers.
56
57      hprev is Hx1 array of initial hidden state
58
59      return the loss, gradients on model parameters, and last hidden state
60
61
62      xs, hs, ys, ps = np.zeros((2, 25, 1, 0, 0, 0)
63
64      hprev = np.copy(hprev)
65
66      # Forward pass
67
68      for t in range(len(inputs)):
69
70          i = np.zeros((vocab_size, 1)) # encode 1 of k representation
71          i[[inputs[t]]] = 1 # encode 1 of k representation
72
73          hprev = np.tanh(np.dot(hprev, wh) + np.dot(i, bh) + h)
74
75          h = np.tanh(np.dot(hprev, wh) + np.dot(i, bh) + h)
76
77          hprev = h # by convention: log probabilities for next char
78          p = np.exp(h) / np.sum(np.exp(h)) # probabilities for next char
79
80          loss += -np.log(p[targets[t]]) # target is one-hot
81
82          o = np.zeros((vocab_size, 1)) # softmax output
83
84          o[[targets[t]]] = 1 # one-hot target
85
86          dy = o - p # error term
87
88          dwh, dbh, dyh = np.zeros_like(hh), np.zeros_like(bh), np.zeros_like(dy)
89          dby = np.zeros_like(dy)
90
91          for t in reversed(range(len(inputs)-1)):
92
93              dy = np.copy(dy[1])
94
95              dy += np.dot(dyh, wh[1:T])
96              dwh += np.dot(dy, wh[1:T])
97              dbh += np.sum(dy)
98
99              dyh = np.tanh(np.dot(h, wh[1:T]) + np.sum(dy * wh[1:T])) # h
100             dh = np.tanh(np.dot(h, wh[1:T]) + np.sum(dy * wh[1:T])) # h
101             dwh += np.dot(dh, wh[1:T])
102             dbh += np.sum(dh)
103
104             dby += np.dot(dy, wh[1:T])
105             dh = np.dot(dh, wh[1:T])
106
107             for parent in (dwh, dbh, ddyh, dby):
108                 parent += np.sum(parent) # in-place summing to mitigate exploding gradients
109
110             return loss, dwh, dbh, ddyh, dby, h[sequence_length-1]
111
112  def sample(next, n):
113
114      sample a sequence of integers from the model
115      h is memory state, next is seed letter for first time step
116
117      h = np.zeros((hidden_size, 1))
118
119      for t in range(n):
120
121          i = np.zeros((vocab_size, 1))
122          i[[next]] = 1
123
124          h = np.tanh(np.dot(h, wh) + np.dot(i, bh) + h)
125
126          y = np.dot(h, wh) + b
127          p = np.exp(y) / np.sum(np.exp(y))
128          x = np.random.choice(range(vocab_size), p=p.ravel())
129          next = ix_to_char[vocab_size, x]
130
131      return next
132
133  timesteps = 1000
134
135  def evalp(inputs):
136
137      h, p, i = forward(inputs, None, np.zeros((hidden_size, 1)))
138
139      h, p, i = forward(inputs, None, h)
140
141      h, p, i = forward(inputs, None, h)
142
143      h, p, i = forward(inputs, None, h)
144
145      h, p, i = forward(inputs, None, h)
146
147      h, p, i = forward(inputs, None, h)
148
149      h, p, i = forward(inputs, None, h)
150
151      h, p, i = forward(inputs, None, h)
152
153      h, p, i = forward(inputs, None, h)
154
155      h, p, i = forward(inputs, None, h)
156
157      h, p, i = forward(inputs, None, h)
158
159      h, p, i = forward(inputs, None, h)
160
161      h, p, i = forward(inputs, None, h)
162
163      h, p, i = forward(inputs, None, h)
164
165      h, p, i = forward(inputs, None, h)
166
167      h, p, i = forward(inputs, None, h)
168
169      h, p, i = forward(inputs, None, h)
170
171      h, p, i = forward(inputs, None, h)
172
173      h, p, i = forward(inputs, None, h)
174
175      h, p, i = forward(inputs, None, h)
176
177      h, p, i = forward(inputs, None, h)
178
179      h, p, i = forward(inputs, None, h)
180
181      h, p, i = forward(inputs, None, h)
182
183      h, p, i = forward(inputs, None, h)
184
185      h, p, i = forward(inputs, None, h)
186
187      h, p, i = forward(inputs, None, h)
188
189      h, p, i = forward(inputs, None, h)
190
191      h, p, i = forward(inputs, None, h)
192
193      h, p, i = forward(inputs, None, h)
194
195      h, p, i = forward(inputs, None, h)
196
197      h, p, i = forward(inputs, None, h)
198
199      h, p, i = forward(inputs, None, h)
200
201      h, p, i = forward(inputs, None, h)
202
203      h, p, i = forward(inputs, None, h)
204
205      h, p, i = forward(inputs, None, h)
206
207      h, p, i = forward(inputs, None, h)
208
209      h, p, i = forward(inputs, None, h)
210
211      h, p, i = forward(inputs, None, h)
212
213      h, p, i = forward(inputs, None, h)
214
215      h, p, i = forward(inputs, None, h)
216
217      h, p, i = forward(inputs, None, h)
218
219      h, p, i = forward(inputs, None, h)
220
221      h, p, i = forward(inputs, None, h)
222
223      h, p, i = forward(inputs, None, h)
224
225      h, p, i = forward(inputs, None, h)
226
227      h, p, i = forward(inputs, None, h)
228
229      h, p, i = forward(inputs, None, h)
230
231      h, p, i = forward(inputs, None, h)
232
233      h, p, i = forward(inputs, None, h)
234
235      h, p, i = forward(inputs, None, h)
236
237      h, p, i = forward(inputs, None, h)
238
239      h, p, i = forward(inputs, None, h)
240
241      h, p, i = forward(inputs, None, h)
242
243      h, p, i = forward(inputs, None, h)
244
245      h, p, i = forward(inputs, None, h)
246
247      h, p, i = forward(inputs, None, h)
248
249      h, p, i = forward(inputs, None, h)
250
251      h, p, i = forward(inputs, None, h)
252
253      h, p, i = forward(inputs, None, h)
254
255      h, p, i = forward(inputs, None, h)
256
257      h, p, i = forward(inputs, None, h)
258
259      h, p, i = forward(inputs, None, h)
260
261      h, p, i = forward(inputs, None, h)
262
263      h, p, i = forward(inputs, None, h)
264
265      h, p, i = forward(inputs, None, h)
266
267      h, p, i = forward(inputs, None, h)
268
269      h, p, i = forward(inputs, None, h)
270
271      h, p, i = forward(inputs, None, h)
272
273      h, p, i = forward(inputs, None, h)
274
275      h, p, i = forward(inputs, None, h)
276
277      h, p, i = forward(inputs, None, h)
278
279      h, p, i = forward(inputs, None, h)
280
281      h, p, i = forward(inputs, None, h)
282
283      h, p, i = forward(inputs, None, h)
284
285      h, p, i = forward(inputs, None, h)
286
287      h, p, i = forward(inputs, None, h)
288
289      h, p, i = forward(inputs, None, h)
290
291      h, p, i = forward(inputs, None, h)
292
293      h, p, i = forward(inputs, None, h)
294
295      h, p, i = forward(inputs, None, h)
296
297      h, p, i = forward(inputs, None, h)
298
299      h, p, i = forward(inputs, None, h)
300
301      h, p, i = forward(inputs, None, h)
302
303      h, p, i = forward(inputs, None, h)
304
305      h, p, i = forward(inputs, None, h)
306
307      h, p, i = forward(inputs, None, h)
308
309      h, p, i = forward(inputs, None, h)
310
311      h, p, i = forward(inputs, None, h)
312
313      h, p, i = forward(inputs, None, h)
314
315      h, p, i = forward(inputs, None, h)
316
317      h, p, i = forward(inputs, None, h)
318
319      h, p, i = forward(inputs, None, h)
320
321      h, p, i = forward(inputs, None, h)
322
323      h, p, i = forward(inputs, None, h)
324
325      h, p, i = forward(inputs, None, h)
326
327      h, p, i = forward(inputs, None, h)
328
329      h, p, i = forward(inputs, None, h)
330
331      h, p, i = forward(inputs, None, h)
332
333      h, p, i = forward(inputs, None, h)
334
335      h, p, i = forward(inputs, None, h)
336
337      h, p, i = forward(inputs, None, h)
338
339      h, p, i = forward(inputs, None, h)
340
341      h, p, i = forward(inputs, None, h)
342
343      h, p, i = forward(inputs, None, h)
344
345      h, p, i = forward(inputs, None, h)
346
347      h, p, i = forward(inputs, None, h)
348
349      h, p, i = forward(inputs, None, h)
350
351      h, p, i = forward(inputs, None, h)
352
353      h, p, i = forward(inputs, None, h)
354
355      h, p, i = forward(inputs, None, h)
356
357      h, p, i = forward(inputs, None, h)
358
359      h, p, i = forward(inputs, None, h)
360
361      h, p, i = forward(inputs, None, h)
362
363      h, p, i = forward(inputs, None, h)
364
365      h, p, i = forward(inputs, None, h)
366
367      h, p, i = forward(inputs, None, h)
368
369      h, p, i = forward(inputs, None, h)
370
371      h, p, i = forward(inputs, None, h)
372
373      h, p, i = forward(inputs, None, h)
374
375      h, p, i = forward(inputs, None, h)
376
377      h, p, i = forward(inputs, None, h)
378
379      h, p, i = forward(inputs, None, h)
380
381      h, p, i = forward(inputs, None, h)
382
383      h, p, i = forward(inputs, None, h)
384
385      h, p, i = forward(inputs, None, h)
386
387      h, p, i = forward(inputs, None, h)
388
389      h, p, i = forward(inputs, None, h)
390
391      h, p, i = forward(inputs, None, h)
392
393      h, p, i = forward(inputs, None, h)
394
395      h, p, i = forward(inputs, None, h)
396
397      h, p, i = forward(inputs, None, h)
398
399      h, p, i = forward(inputs, None, h)
400
401      h, p, i = forward(inputs, None, h)
402
403      h, p, i = forward(inputs, None, h)
404
405      h, p, i = forward(inputs, None, h)
406
407      h, p, i = forward(inputs, None, h)
408
409      h, p, i = forward(inputs, None, h)
410
411      h, p, i = forward(inputs, None, h)
412
413      h, p, i = forward(inputs, None, h)
414
415      h, p, i = forward(inputs, None, h)
416
417      h, p, i = forward(inputs, None, h)
418
419      h, p, i = forward(inputs, None, h)
420
421      h, p, i = forward(inputs, None, h)
422
423      h, p, i = forward(inputs, None, h)
424
425      h, p, i = forward(inputs, None, h)
426
427      h, p, i = forward(inputs, None, h)
428
429      h, p, i = forward(inputs, None, h)
430
431      h, p, i = forward(inputs, None, h)
432
433      h, p, i = forward(inputs, None, h)
434
435      h, p, i = forward(inputs, None, h)
436
437      h, p, i = forward(inputs, None, h)
438
439      h, p, i = forward(inputs, None, h)
440
441      h, p, i = forward(inputs, None, h)
442
443      h, p, i = forward(inputs, None, h)
444
445      h, p, i = forward(inputs, None, h)
446
447      h, p, i = forward(inputs, None, h)
448
449      h, p, i = forward(inputs, None, h)
450
451      h, p, i = forward(inputs, None, h)
452
453      h, p, i = forward(inputs, None, h)
454
455      h, p, i = forward(inputs, None, h)
456
457      h, p, i = forward(inputs, None, h)
458
459      h, p, i = forward(inputs, None, h)
460
461      h, p, i = forward(inputs, None, h)
462
463      h, p, i = forward(inputs, None, h)
464
465      h, p, i = forward(inputs, None, h)
466
467      h, p, i = forward(inputs, None, h)
468
469      h, p, i = forward(inputs, None, h)
470
471      h, p, i = forward(inputs, None, h)
472
473      h, p, i = forward(inputs, None, h)
474
475      h, p, i = forward(inputs, None, h)
476
477      h, p, i = forward(inputs, None, h)
478
479      h, p, i = forward(inputs, None, h)
480
481      h, p, i = forward(inputs, None, h)
482
483      h, p, i = forward(inputs, None, h)
484
485      h, p, i = forward(inputs, None, h)
486
487      h, p, i = forward(inputs, None, h)
488
489      h, p, i = forward(inputs, None, h)
490
491      h, p, i = forward(inputs, None, h)
492
493      h, p, i = forward(inputs, None, h)
494
495      h, p, i = forward(inputs, None, h)
496
497      h, p, i = forward(inputs, None, h)
498
499      h, p, i = forward(inputs, None, h)
500
501      h, p, i = forward(inputs, None, h)
502
503      h, p, i = forward(inputs, None, h)
504
505      h, p, i = forward(inputs, None, h)
506
507      h, p, i = forward(inputs, None, h)
508
509      h, p, i = forward(inputs, None, h)
510
511      h, p, i = forward(inputs, None, h)
512
513      h, p, i = forward(inputs, None, h)
514
515      h, p, i = forward(inputs, None, h)
516
517      h, p, i = forward(inputs, None, h)
518
519      h, p, i = forward(inputs, None, h)
520
521      h, p, i = forward(inputs, None, h)
522
523      h, p, i = forward(inputs, None, h)
524
525      h, p, i = forward(inputs, None, h)
526
527      h, p, i = forward(inputs, None, h)
528
529      h, p, i = forward(inputs, None, h)
530
531      h, p, i = forward(inputs, None, h)
532
533      h, p, i = forward(inputs, None, h)
534
535      h, p, i = forward(inputs, None, h)
536
537      h, p, i = forward(inputs, None, h)
538
539      h, p, i = forward(inputs, None, h)
540
541      h, p, i = forward(inputs, None, h)
542
543      h, p, i = forward(inputs, None, h)
544
545      h, p, i = forward(inputs, None, h)
546
547      h, p, i = forward(inputs, None, h)
548
549      h, p, i = forward(inputs, None, h)
550
551      h, p, i = forward(inputs, None, h)
552
553      h, p, i = forward(inputs, None, h)
554
555      h, p, i = forward(inputs, None, h)
556
557      h, p, i = forward(inputs, None, h)
558
559      h, p, i = forward(inputs, None, h)
560
561      h, p, i = forward(inputs, None, h)
562
563      h, p, i = forward(inputs, None, h)
564
565      h, p, i = forward(inputs, None, h)
566
567      h, p, i = forward(inputs, None, h)
568
569      h, p, i = forward(inputs, None, h)
570
571      h, p, i = forward(inputs, None, h)
572
573      h, p, i = forward(inputs, None, h)
574
575      h, p, i = forward(inputs, None, h)
576
577      h, p, i = forward(inputs, None, h)
578
579      h, p, i = forward(inputs, None, h)
580
581      h, p, i = forward(inputs, None, h)
582
583      h, p, i = forward(inputs, None, h)
584
585      h, p, i = forward(inputs, None, h)
586
587      h, p, i = forward(inputs, None, h)
588
589      h, p, i = forward(inputs, None, h)
590
591      h, p, i = forward(inputs, None, h)
592
593      h, p, i = forward(inputs, None, h)
594
595      h, p, i = forward(inputs, None, h)
596
597      h, p, i = forward(inputs, None, h)
598
599      h, p, i = forward(inputs, None, h)
600
601      h, p, i = forward(inputs, None, h)
602
603      h, p, i = forward(inputs, None, h)
604
605      h, p, i = forward(inputs, None, h)
606
607      h, p, i = forward(inputs, None, h)
608
609      h, p, i = forward(inputs, None, h)
610
611      h, p, i = forward(inputs, None, h)
612
613      h, p, i = forward(inputs, None, h)
614
615      h, p, i = forward(inputs, None, h)
616
617      h, p, i = forward(inputs, None, h)
618
619      h, p, i = forward(inputs, None, h)
620
621      h, p, i = forward(inputs, None, h)
622
623      h, p, i = forward(inputs, None, h)
624
625      h, p, i = forward(inputs, None, h)
626
627      h, p, i = forward(inputs, None, h)
628
629      h, p, i = forward(inputs, None, h)
630
631      h, p, i = forward(inputs, None, h)
632
633      h, p, i = forward(inputs, None, h)
634
635      h, p, i = forward(inputs, None, h)
636
637      h, p, i = forward(inputs, None, h)
638
639      h, p, i = forward(inputs, None, h)
640
641      h, p, i = forward(inputs, None, h)
642
643      h, p, i = forward(inputs, None, h)
644
645      h, p, i = forward(inputs, None, h)
646
647      h, p, i = forward(inputs, None, h)
648
649      h, p, i = forward(inputs, None, h)
650
651      h, p, i = forward(inputs, None, h)
652
653      h, p, i = forward(inputs, None, h)
654
655      h, p, i = forward(inputs, None, h)
656
657      h, p, i = forward(inputs, None, h)
658
659      h, p, i = forward(inputs, None, h)
660
661      h, p, i = forward(inputs, None, h)
662
663      h, p, i = forward(inputs, None, h)
664
665      h, p, i = forward(inputs, None, h)
666
667      h, p, i = forward(inputs, None, h)
668
669      h, p, i = forward(inputs, None, h)
670
671      h, p, i = forward(inputs, None, h)
672
673      h, p, i = forward(inputs, None, h)
674
675      h, p, i = forward(inputs, None, h)
676
677      h, p, i = forward(inputs, None, h)
678
679      h, p, i = forward(inputs, None, h)
680
681      h, p, i = forward(inputs, None, h)
682
683      h, p, i = forward(inputs, None, h)
684
685      h, p, i = forward(inputs, None, h)
686
687      h, p, i = forward(inputs, None, h)
688
689      h, p, i = forward(inputs, None, h)
690
691      h, p, i = forward(inputs, None, h)
692
693      h, p, i = forward(inputs, None, h)
694
695      h, p, i = forward(inputs, None, h)
696
697      h, p, i = forward(inputs, None, h)
698
699      h, p, i = forward(inputs, None, h)
700
701      h, p, i = forward(inputs, None, h)
702
703      h, p, i = forward(inputs, None, h)
704
705      h, p, i = forward(inputs, None, h)
706
707      h, p, i = forward(inputs, None, h)
708
709      h, p, i = forward(inputs, None, h)
710
711      h, p, i = forward(inputs, None, h)
712
713      h, p, i = forward(inputs, None, h)
714
715      h, p, i = forward(inputs, None, h)
716
717      h, p, i = forward(inputs, None, h)
718
719      h, p, i = forward(inputs, None, h)
720
721      h, p, i = forward(inputs, None, h)
722
723      h, p, i = forward(inputs, None, h)
724
725      h, p, i = forward(inputs, None, h)
726
727      h, p, i = forward(inputs, None, h)
728
729      h, p, i = forward(inputs, None, h)
730
731      h, p, i = forward(inputs, None, h)
732
733      h, p, i = forward(inputs, None, h)
734
735      h, p, i = forward(inputs, None, h)
736
737      h, p, i = forward(inputs, None, h)
738
739      h, p, i = forward(inputs, None, h)
740
741      h, p, i = forward(inputs, None, h)
742
743      h, p, i = forward(inputs, None, h)
744
745      h, p, i = forward(inputs, None, h)
746
747      h, p, i = forward(inputs, None, h)
748
749      h, p, i = forward(inputs, None, h)
750
751      h, p, i = forward(inputs, None, h)
752
753      h, p, i = forward(inputs, None, h)
754
755      h, p, i = forward(inputs, None, h)
756
757      h, p, i = forward(inputs, None, h)
758
759      h, p, i = forward(inputs, None, h)
760
761      h, p, i = forward(inputs, None, h)
762
763      h, p, i = forward(inputs, None, h)
764
765      h, p, i = forward(inputs, None, h)
766
767      h, p, i = forward(inputs, None, h)
768
769      h, p, i = forward(inputs, None, h)
770
771      h, p, i = forward(inputs, None, h)
772
773      h, p, i = forward(inputs, None, h)
774
775      h, p, i = forward(inputs, None, h)
776
777      h, p, i = forward(inputs, None, h)
778
779      h, p, i = forward(inputs, None, h)
780
781      h, p, i = forward(inputs, None, h)
782
783      h, p, i = forward(inputs, None, h)
784
785      h, p, i = forward(inputs, None, h)
786
787      h, p, i = forward(inputs, None, h)
788
789      h, p, i = forward(inputs, None, h)
790
791      h, p, i = forward(inputs, None, h)
792
793      h, p, i = forward(inputs, None, h)
794
795      h, p, i = forward(inputs, None, h)
796
797      h, p, i = forward(inputs, None, h)
798
799      h, p, i = forward(inputs, None, h)
800
801      h, p, i = forward(inputs, None, h)
802
803      h, p, i = forward(inputs, None, h)
804
805      h, p, i = forward(inputs, None, h)
806
807      h, p, i = forward(inputs, None, h)
808
809      h, p, i = forward(inputs, None, h)
810
811      h, p, i = forward(inputs, None, h)
812
813      h, p, i = forward(inputs, None, h)
814
815      h, p, i = forward(inputs, None, h)
816
817      h, p, i = forward(inputs, None, h)
818
819      h, p, i = forward(inputs, None, h)
820
821      h, p, i = forward(inputs, None, h)
822
823      h, p, i = forward(inputs, None, h)
824
825      h, p, i = forward(inputs, None, h)
826
827      h, p, i = forward(inputs, None, h)
828
829      h, p, i = forward(inputs, None, h)
830
831      h, p, i = forward(inputs, None, h)
832
833      h, p, i = forward(inputs, None, h)
834
835      h, p, i = forward(inputs, None, h)
836
837      h, p, i = forward(inputs, None, h)
838
839      h, p, i = forward(inputs, None, h)
840
841      h, p, i = forward(inputs, None, h)
842
843      h, p, i = forward(inputs, None, h)
844
845      h, p, i = forward(inputs, None, h)
846
847      h, p, i = forward(inputs, None, h)
848
849      h, p, i = forward(inputs, None, h)
850
851      h, p, i = forward(inputs, None, h)
852
853      h, p, i = forward(inputs, None, h)
854
855      h, p, i = forward(inputs, None, h)
856
857      h, p, i = forward(inputs, None, h)
858
859      h, p, i = forward(inputs, None, h)
860
861      h, p, i = forward(inputs, None, h)
862
863      h, p, i = forward(inputs, None, h)
864
865      h, p, i = forward(inputs, None, h)
866
867      h, p, i = forward(inputs, None, h)
868
869      h, p, i = forward(inputs, None, h)
870
871      h, p, i = forward(inputs, None, h)
872
873      h, p, i = forward(inputs, None, h)
874
875      h, p, i = forward(inputs, None, h)
876
877      h, p, i = forward(inputs, None, h)
878
879      h, p, i = forward(inputs, None, h)
880
881      h, p, i = forward(inputs, None, h)
882
883      h, p, i = forward(inputs, None, h)
884
885      h, p, i = forward(inputs, None, h)
886
887      h, p, i = forward(inputs, None, h)
888
889      h, p, i = forward(inputs, None, h)
890
891      h, p, i = forward(inputs, None, h)
892
893      h, p, i = forward(inputs, None, h)
894
895      h, p, i = forward(inputs, None, h)
896
897      h, p, i = forward(inputs, None, h)
898
899      h, p, i = forward(inputs, None, h)
900
901      h, p, i = forward(inputs, None, h)
902
903      h, p, i = forward(inputs, None, h)
904
905      h, p, i = forward(inputs, None, h)
906
907      h, p, i = forward(inputs, None, h)
908
909      h, p, i = forward(inputs, None, h)
910
911      h, p, i = forward(inputs, None, h)
912
913      h, p, i = forward(inputs, None, h)
914
915      h, p, i = forward(inputs, None, h)
916
917      h, p, i = forward(inputs, None, h)
918
919      h, p, i = forward(inputs, None, h)
920
921      h, p, i = forward(inputs, None, h)
922
923      h, p, i = forward(inputs, None, h)
924
925      h, p, i = forward(inputs, None, h)
926
927      h, p, i = forward(inputs, None, h)
928
929      h, p, i = forward(inputs, None, h)
930
931      h, p, i = forward(inputs, None, h)
932
933      h, p, i = forward(inputs, None, h)
934
935      h, p, i = forward(inputs, None, h)
936
937      h, p, i = forward(inputs, None, h)
938
939      h, p, i = forward(inputs, None, h)
940
941      h, p, i = forward(inputs, None, h)
942
943      h, p, i = forward(inputs, None, h)
944
945      h, p, i = forward(inputs, None, h)
946
947      h, p, i = forward(inputs, None, h)
948
949      h, p, i = forward(inputs, None, h)
950
951      h, p, i = forward(inputs, None, h)
952
953      h, p, i = forward(inputs, None, h)
954
955      h, p, i = forward(inputs, None, h)
956
957      h, p, i = forward(inputs, None, h)
958
959      h, p, i = forward(inputs, None, h)
960
961      h, p, i = forward(inputs, None, h)
962
963      h, p, i = forward(inputs, None, h)
964
965      h, p, i = forward(inputs, None, h)
966
967      h, p, i = forward(inputs, None, h)
968
969      h, p, i = forward(inputs, None, h)
970
971      h, p, i = forward(inputs, None, h)
972
973      h, p, i = forward(inputs, None, h)
974
975      h, p, i = forward(inputs, None, h)
976
977      h, p, i = forward(inputs, None, h)
978
979      h, p, i = forward(inputs, None, h)
980
981      h, p, i = forward(inputs, None, h)
982
983      h, p, i = forward(inputs, None, h)
984
985      h, p, i = forward(inputs, None, h)
986
987      h, p, i = forward(inputs, None, h)
988
989      h, p, i = forward(inputs, None, h)
990
991      h, p, i = forward(inputs, None, h)
992
993      h, p, i = forward(inputs, None, h)
994
995      h, p, i = forward(inputs, None, h)
996
997      h, p, i = forward(inputs, None, h)
998
999      h, p, i = forward(inputs, None, h)
1000
1001      h, p, i = forward(inputs, None, h)
1002
1003      h, p, i = forward(inputs, None, h)
1004
1005      h, p, i = forward(inputs, None, h)
1006
1007      h, p, i = forward(inputs, None, h)
1008
1009      h, p, i = forward(inputs, None, h)
1010
1011      h, p, i = forward(inputs, None, h)
1012
1013      h, p, i = forward(inputs, None, h)
1014
1015      h, p, i = forward(inputs, None, h)
1016
1017      h, p, i = forward(inputs, None, h)
1018
1019      h, p, i = forward(inputs, None, h)
1020
1021      h, p, i = forward(inputs, None, h)
1022
1023      h, p, i = forward(inputs, None, h)
1024
1025      h, p, i = forward(inputs, None, h)
1026
1027      h, p, i = forward(inputs, None, h)
1028
1029      h, p, i = forward(inputs, None, h)
1030
1031      h, p, i = forward(inputs, None, h)
1032
1033      h, p, i = forward(inputs, None, h)
1034
1035      h, p, i = forward(inputs, None, h)
1036
1037      h, p, i = forward(inputs, None, h)
1038
1039      h, p, i = forward(inputs, None, h)
1040
1041      h, p, i = forward(inputs, None, h)
1042
1043      h, p, i = forward(inputs, None, h)
1044
1045      h, p, i = forward(inputs, None, h)
1046
1047      h, p, i = forward(inputs, None, h)
1048
1049      h, p, i = forward(inputs, None, h)
1050
1051      h, p, i = forward(inputs, None, h)
1052
1053      h, p, i = forward(inputs, None, h)
1054
1055      h, p, i = forward(inputs, None, h)
1056
1057      h, p, i = forward(inputs, None, h)
1058
1059      h, p, i = forward(inputs, None, h)
1060
1061      h, p, i = forward(inputs, None, h)
1062
1063      h, p, i = forward(inputs, None, h)
1064
1065      h, p, i = forward(inputs, None, h)
1066
1067      h, p, i = forward(inputs, None, h)
1068
1069      h, p, i = forward(inputs, None, h)
1070
1071      h, p, i = forward(inputs, None, h)
1072
1073      h, p, i = forward(inputs, None, h)
1074
1075      h, p, i = forward(inputs, None, h)
1076
1077      h, p, i = forward(inputs, None, h)
1078
1079      h, p, i = forward(inputs, None, h)
1080
1081      h, p, i = forward(inputs, None, h)
1082
1083      h, p, i = forward(inputs, None, h)
1084
1085      h, p, i = forward(inputs, None, h)
1086
1087      h, p, i = forward(inputs, None, h)
1088
1089      h, p, i = forward(inputs, None, h)
1090
1091      h, p, i = forward(inputs, None, h)
1092
1093      h, p, i = forward(inputs, None, h)
1094
1095      h, p, i = forward(inputs, None, h)
1096
1097      h, p, i = forward(inputs, None, h)
1098
1099      h, p, i = forward(inputs, None, h)
1100
1101      h, p, i = forward(inputs, None, h)
1102
1103      h, p, i = forward(inputs, None, h)
1104
1105      h, p, i = forward(inputs, None, h)
1106
1107      h, p, i = forward(inputs, None, h)
1108
1109      h, p, i = forward(inputs, None, h)
1110
1111      h, p, i = forward(inputs, None, h)
1112
1113      h, p, i = forward(inputs, None, h)
1114
1115      h, p, i = forward(inputs, None, h)
1116
1117      h, p, i = forward(inputs, None, h)
1118
1119      h, p, i = forward(inputs, None, h)
1120
1121      h, p, i = forward(inputs, None, h)
1122
1123      h, p, i = forward(inputs, None, h)
1124
1125      h, p, i = forward(inputs, None, h)
1126
1127      h, p, i = forward(inputs, None, h)
1128
1129      h, p, i = forward(inputs, None, h)
1130
1131      h, p, i = forward(inputs, None, h)
1132
1133      h, p, i = forward(inputs, None, h)
1134
1135      h, p, i = forward(inputs, None, h)
1136
1137      h, p, i = forward(inputs, None, h)
1138
1139      h, p, i = forward(inputs, None, h)
1140
1141      h, p, i = forward(inputs, None, h)
1142
1143      h, p, i = forward(inputs, None, h)
1144
1145      h, p, i = forward(inputs, None, h)
1146
1147      h, p, i = forward(inputs, None, h)
1148
1149      h, p, i = forward(inputs, None, h)
115
```

```

27 def lossFun(inputs, targets, hprev):
28     """
29     inputs,targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33
34     xs, hs, ys, ps = {}, {}, {}, {}
35     hs[-1] = np.copy(hprev)
36     loss = 0
37
38     # forward pass
39     for t in xrange(len(inputs)):
40         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
41         xs[t][inputs[t]] = 1
42
43         hs[t] = np.tanh(np.dot(wxh, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
44         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
45         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
46         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
47
48     # backward pass: compute gradients going backwards
49     dWxh, dWhh, dWhy = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(why)
50     dbh, dby = np.zeros_like(bh), np.zeros_like(by)
51     dhnext = np.zeros_like(hs[0])
52
53     for t in reversed(xrange(len(inputs))):
54         dy = np.copy(ps[t])
55         dy[targets[t]] -= 1 # backprop into y
56         dWhy += np.dot(dy, hs[t].T)
57         dbh += dy
58
59         dh = np.dot(why.T, dy) + dhnext # backprop into h
60         ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
61         dbh += ddraw
62
63         dWxh += np.dot(ddraw, xs[t].T)
64         dWhh += np.dot(ddraw, hs[t-1].T)
65         dhnext = np.dot(whh.T, ddraw)
66
67     for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
68         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
69
70     return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

```

min-char-rnn.py gist

```

1 Minimal character-level vanilla RNN model. written by Andrej Karpathy (@karpathy)
2 BSO Licenses
3
4 Import numpy as np
5
6
7 # Data
8 data = open("charRNN.txt", "r").read() # should be simple plain text file
9
10 vocab_size = len(set(data)) # unique # (Data_size, vocab_size)
11 print("data has %d characters, %d unique." % (data_size, vocab_size))
12
13 char_to_ix = {ch:i for i, ch in enumerate(char)}
14 ix_to_char = {i:ch for ch, i in char_to_ix.items()}
15
16 # Hyperparameters
17 hidden_size = 256 # size of hidden layer of neurons
18 seq_length = 25 # number of steps to unroll the RNN for
19 learning_rate = 1e-1
20
21 # Model parameters
22 wih = np.random.uniform(-0.1, 0.1, (vocab_size, hidden_size)) # input to hidden
23 whh = np.random.uniform(-0.1, 0.1, (hidden_size, hidden_size)) # hidden to hidden
24 by = np.random.rand(vocab_size, 1) # bias. would be hidden bias
25 bh = np.zeros(hidden_size, 1) # hidden bias
26 by = np.zeros(vocab_size, 1) # output bias
27
28
29 def loss(inputs, targets, hprev):
30
31     inputs, targets = list(inputs), list(targets)
32     hprev = np.array(hprev)
33
34     loss = 0
35     hidden = hprev
36
37     for i in range(len(inputs)):
38         x = np.zeros(vocab_size)
39         x[inputs[i]] = 1 # encode x as 1-of-k representation
40         x1 = np.dot(wih, x) + np.dot(bh, bh[i-1]) + bh
41         h1 = np.tanh(x1)
42         y1 = np.dot(whh, h1) + by
43         prob1 = np.exp(y1) / np.sum(np.exp(y1))
44         loss += -np.log(prob1[targets[i]])
45
46         hidden = h1
47
48     return loss
49
50
51 def sample(hprev, seed_ix, n):
52     x = np.zeros(vocab_size)
53     x[seed_ix] = 1
54
55     for i in range(n):
56         hprev = np.dot(wih, x) + np.dot(whh, hprev) + by
57         hprev = np.tanh(hprev)
58         y = np.dot(whh, hprev) + by
59         p = np.exp(y) / np.sum(np.exp(y))
60
61         ix = np.random.choice(range(vocab_size), p=p)
62         x = np.zeros(vocab_size)
63         x[ix] = 1
64
65     return ix
66
67
68 def runRNN(x, h0, dprev, dbh, dby, dwh, dbw, dby1):
69     hprev = h0
70
71     for i in range(len(x)):
72         x1 = np.zeros(vocab_size)
73         x1[x[i]] = 1
74
75         y1 = np.dot(whh, hprev) + by
76         prob1 = np.exp(y1) / np.sum(np.exp(y1))
77
78         loss = -np.log(prob1[x[i]])
79
80         dhprev = np.dot(dbh, dbh[i-1]) + np.dot(dwh, dwh[i-1]) + dby1
81
82         dh = np.dot(dbw, dhprev) + np.dot(dwh, dhprev) + dby
83
84         dhprev = np.dot(dbh, dhprev) + np.dot(dwh, dhprev) + dby
85
86         dhprev = np.tanh(dhprev)
87         dwh = np.dot(dhprev, dhprev)
88
89         dbh = np.dot(dhprev, dhprev)
90
91         dby = np.dot(dhprev, dby)
92
93         dy1 = np.tanh(y1)
94
95         dprev = np.dot(dbw, dy1) + np.dot(dwh, dy1) + dby
96
97         dprev = np.tanh(dprev)
98
99         dby1 = np.dot(dbw, dprev) + np.dot(dwh, dprev) + dby
100
101         dby1 = np.tanh(dby1)
102
103         dhprev = np.dot(dbh, dhprev) + np.dot(dwh, dhprev) + dby1
104
105         dhprev = np.tanh(dhprev)
106
107         dwh = np.dot(dhprev, dhprev)
108
109         dbw = np.dot(dhprev, dhprev)
110
111         dby = np.dot(dhprev, dby)
112
113         dhprev = np.tanh(dhprev)
114
115         dby1 = np.dot(dbw, dhprev) + np.dot(dwh, dhprev) + dby
116
117         dby1 = np.tanh(dby1)
118
119         dhprev = np.tanh(dhprev)
120
121         dby = np.dot(dhprev, dby)
122
123         dby1 = np.tanh(dby1)
124
125         dhprev = np.tanh(dhprev)
126
127         dby = np.dot(dhprev, dby)
128
129         dby1 = np.tanh(dby1)
130
131         dhprev = np.tanh(dhprev)
132
133         dby = np.dot(dhprev, dby)
134
135         dby1 = np.tanh(dby1)
136
137         dhprev = np.tanh(dhprev)
138
139         dby = np.dot(dhprev, dby)
140
141         dby1 = np.tanh(dby1)
142
143         dhprev = np.tanh(dhprev)
144
145         dby = np.dot(dhprev, dby)
146
147         dby1 = np.tanh(dby1)
148
149         dhprev = np.tanh(dhprev)
150
151         dby = np.dot(dhprev, dby)
152
153         dby1 = np.tanh(dby1)
154
155         dhprev = np.tanh(dhprev)
156
157         dby = np.dot(dhprev, dby)
158
159         dby1 = np.tanh(dby1)
160
161         dhprev = np.tanh(dhprev)
162
163         dby = np.dot(dhprev, dby)
164
165         dby1 = np.tanh(dby1)
166
167         dhprev = np.tanh(dhprev)
168
169         dby = np.dot(dhprev, dby)
170
171         dby1 = np.tanh(dby1)
172
173         dhprev = np.tanh(dhprev)
174
175         dby = np.dot(dhprev, dby)
176
177         dby1 = np.tanh(dby1)
178
179         dhprev = np.tanh(dhprev)
180
181         dby = np.dot(dhprev, dby)
182
183         dby1 = np.tanh(dby1)
184
185         dhprev = np.tanh(dhprev)
186
187         dby = np.dot(dhprev, dby)
188
189         dby1 = np.tanh(dby1)
190
191         dhprev = np.tanh(dhprev)
192
193         dby = np.dot(dhprev, dby)
194
195         dby1 = np.tanh(dby1)
196
197         dhprev = np.tanh(dhprev)
198
199         dby = np.dot(dhprev, dby)
200
201         dby1 = np.tanh(dby1)
202
203         dhprev = np.tanh(dhprev)
204
205         dby = np.dot(dhprev, dby)
206
207         dby1 = np.tanh(dby1)
208
209         dhprev = np.tanh(dhprev)
210
211         dby = np.dot(dhprev, dby)
212
213         dby1 = np.tanh(dby1)
214
215         dhprev = np.tanh(dhprev)
216
217         dby = np.dot(dhprev, dby)
218
219         dby1 = np.tanh(dby1)
220
221         dhprev = np.tanh(dhprev)
222
223         dby = np.dot(dhprev, dby)
224
225         dby1 = np.tanh(dby1)
226
227         dhprev = np.tanh(dhprev)
228
229         dby = np.dot(dhprev, dby)
230
231         dby1 = np.tanh(dby1)
232
233         dhprev = np.tanh(dhprev)
234
235         dby = np.dot(dhprev, dby)
236
237         dby1 = np.tanh(dby1)
238
239         dhprev = np.tanh(dhprev)
240
241         dby = np.dot(dhprev, dby)
242
243         dby1 = np.tanh(dby1)
244
245         dhprev = np.tanh(dhprev)
246
247         dby = np.dot(dhprev, dby)
248
249         dby1 = np.tanh(dby1)
250
251         dhprev = np.tanh(dhprev)
252
253         dby = np.dot(dhprev, dby)
254
255         dby1 = np.tanh(dby1)
256
257         dhprev = np.tanh(dhprev)
258
259         dby = np.dot(dhprev, dby)
260
261         dby1 = np.tanh(dby1)
262
263         dhprev = np.tanh(dhprev)
264
265         dby = np.dot(dhprev, dby)
266
267         dby1 = np.tanh(dby1)
268
269         dhprev = np.tanh(dhprev)
270
271         dby = np.dot(dhprev, dby)
272
273         dby1 = np.tanh(dby1)
274
275         dhprev = np.tanh(dhprev)
276
277         dby = np.dot(dhprev, dby)
278
279         dby1 = np.tanh(dby1)
280
281         dhprev = np.tanh(dhprev)
282
283         dby = np.dot(dhprev, dby)
284
285         dby1 = np.tanh(dby1)
286
287         dhprev = np.tanh(dhprev)
288
289         dby = np.dot(dhprev, dby)
290
291         dby1 = np.tanh(dby1)
292
293         dhprev = np.tanh(dhprev)
294
295         dby = np.dot(dhprev, dby)
296
297         dby1 = np.tanh(dby1)
298
299         dhprev = np.tanh(dhprev)
300
301         dby = np.dot(dhprev, dby)
302
303         dby1 = np.tanh(dby1)
304
305         dhprev = np.tanh(dhprev)
306
307         dby = np.dot(dhprev, dby)
308
309         dby1 = np.tanh(dby1)
310
311         dhprev = np.tanh(dhprev)
312
313         dby = np.dot(dhprev, dby)
314
315         dby1 = np.tanh(dby1)
316
317         dhprev = np.tanh(dhprev)
318
319         dby = np.dot(dhprev, dby)
320
321         dby1 = np.tanh(dby1)
322
323         dhprev = np.tanh(dhprev)
324
325         dby = np.dot(dhprev, dby)
326
327         dby1 = np.tanh(dby1)
328
329         dhprev = np.tanh(dhprev)
330
331         dby = np.dot(dhprev, dby)
332
333         dby1 = np.tanh(dby1)
334
335         dhprev = np.tanh(dhprev)
336
337         dby = np.dot(dhprev, dby)
338
339         dby1 = np.tanh(dby1)
340
341         dhprev = np.tanh(dhprev)
342
343         dby = np.dot(dhprev, dby)
344
345         dby1 = np.tanh(dby1)
346
347         dhprev = np.tanh(dhprev)
348
349         dby = np.dot(dhprev, dby)
350
351         dby1 = np.tanh(dby1)
352
353         dhprev = np.tanh(dhprev)
354
355         dby = np.dot(dhprev, dby)
356
357         dby1 = np.tanh(dby1)
358
359         dhprev = np.tanh(dhprev)
360
361         dby = np.dot(dhprev, dby)
362
363         dby1 = np.tanh(dby1)
364
365         dhprev = np.tanh(dhprev)
366
367         dby = np.dot(dhprev, dby)
368
369         dby1 = np.tanh(dby1)
370
371         dhprev = np.tanh(dhprev)
372
373         dby = np.dot(dhprev, dby)
374
375         dby1 = np.tanh(dby1)
376
377         dhprev = np.tanh(dhprev)
378
379         dby = np.dot(dhprev, dby)
380
381         dby1 = np.tanh(dby1)
382
383         dhprev = np.tanh(dhprev)
384
385         dby = np.dot(dhprev, dby)
386
387         dby1 = np.tanh(dby1)
388
389         dhprev = np.tanh(dhprev)
390
391         dby = np.dot(dhprev, dby)
392
393         dby1 = np.tanh(dby1)
394
395         dhprev = np.tanh(dhprev)
396
397         dby = np.dot(dhprev, dby)
398
399         dby1 = np.tanh(dby1)
400
401         dhprev = np.tanh(dhprev)
402
403         dby = np.dot(dhprev, dby)
404
405         dby1 = np.tanh(dby1)
406
407         dhprev = np.tanh(dhprev)
408
409         dby = np.dot(dhprev, dby)
410
411         dby1 = np.tanh(dby1)
412
413         dhprev = np.tanh(dhprev)
414
415         dby = np.dot(dhprev, dby)
416
417         dby1 = np.tanh(dby1)
418
419         dhprev = np.tanh(dhprev)
420
421         dby = np.dot(dhprev, dby)
422
423         dby1 = np.tanh(dby1)
424
425         dhprev = np.tanh(dhprev)
426
427         dby = np.dot(dhprev, dby)
428
429         dby1 = np.tanh(dby1)
430
431         dhprev = np.tanh(dhprev)
432
433         dby = np.dot(dhprev, dby)
434
435         dby1 = np.tanh(dby1)
436
437         dhprev = np.tanh(dhprev)
438
439         dby = np.dot(dhprev, dby)
440
441         dby1 = np.tanh(dby1)
442
443         dhprev = np.tanh(dhprev)
444
445         dby = np.dot(dhprev, dby)
446
447         dby1 = np.tanh(dby1)
448
449         dhprev = np.tanh(dhprev)
450
451         dby = np.dot(dhprev, dby)
452
453         dby1 = np.tanh(dby1)
454
455         dhprev = np.tanh(dhprev)
456
457         dby = np.dot(dhprev, dby)
458
459         dby1 = np.tanh(dby1)
460
461         dhprev = np.tanh(dhprev)
462
463         dby = np.dot(dhprev, dby)
464
465         dby1 = np.tanh(dby1)
466
467         dhprev = np.tanh(dhprev)
468
469         dby = np.dot(dhprev, dby)
470
471         dby1 = np.tanh(dby1)
472
473         dhprev = np.tanh(dhprev)
474
475         dby = np.dot(dhprev, dby)
476
477         dby1 = np.tanh(dby1)
478
479         dhprev = np.tanh(dhprev)
480
481         dby = np.dot(dhprev, dby)
482
483         dby1 = np.tanh(dby1)
484
485         dhprev = np.tanh(dhprev)
486
487         dby = np.dot(dhprev, dby)
488
489         dby1 = np.tanh(dby1)
490
491         dhprev = np.tanh(dhprev)
492
493         dby = np.dot(dhprev, dby)
494
495         dby1 = np.tanh(dby1)
496
497         dhprev = np.tanh(dhprev)
498
499         dby = np.dot(dhprev, dby)
500
501         dby1 = np.tanh(dby1)
502
503         dhprev = np.tanh(dhprev)
504
505         dby = np.dot(dhprev, dby)
506
507         dby1 = np.tanh(dby1)
508
509         dhprev = np.tanh(dhprev)
510
511         dby = np.dot(dhprev, dby)
512
513         dby1 = np.tanh(dby1)
514
515         dhprev = np.tanh(dhprev)
516
517         dby = np.dot(dhprev, dby)
518
519         dby1 = np.tanh(dby1)
520
521         dhprev = np.tanh(dhprev)
522
523         dby = np.dot(dhprev, dby)
524
525         dby1 = np.tanh(dby1)
526
527         dhprev = np.tanh(dhprev)
528
529         dby = np.dot(dhprev, dby)
530
531         dby1 = np.tanh(dby1)
532
533         dhprev = np.tanh(dhprev)
534
535         dby = np.dot(dhprev, dby)
536
537         dby1 = np.tanh(dby1)
538
539         dhprev = np.tanh(dhprev)
540
541         dby = np.dot(dhprev, dby)
542
543         dby1 = np.tanh(dby1)
544
545         dhprev = np.tanh(dhprev)
546
547         dby = np.dot(dhprev, dby)
548
549         dby1 = np.tanh(dby1)
550
551         dhprev = np.tanh(dhprev)
552
553         dby = np.dot(dhprev, dby)
554
555         dby1 = np.tanh(dby1)
556
557         dhprev = np.tanh(dhprev)
558
559         dby = np.dot(dhprev, dby)
560
561         dby1 = np.tanh(dby1)
562
563         dhprev = np.tanh(dhprev)
564
565         dby = np.dot(dhprev, dby)
566
567         dby1 = np.tanh(dby1)
568
569         dhprev = np.tanh(dhprev)
570
571         dby = np.dot(dhprev, dby)
572
573         dby1 = np.tanh(dby1)
574
575         dhprev = np.tanh(dhprev)
576
577         dby = np.dot(dhprev, dby)
578
579         dby1 = np.tanh(dby1)
580
581         dhprev = np.tanh(dhprev)
582
583         dby = np.dot(dhprev, dby)
584
585         dby1 = np.tanh(dby1)
586
587         dhprev = np.tanh(dhprev)
588
589         dby = np.dot(dhprev, dby)
590
591         dby1 = np.tanh(dby1)
592
593         dhprev = np.tanh(dhprev)
594
595         dby = np.dot(dhprev, dby)
596
597         dby1 = np.tanh(dby1)
598
599         dhprev = np.tanh(dhprev)
600
601         dby = np.dot(dhprev, dby)
602
603         dby1 = np.tanh(dby1)
604
605         dhprev = np.tanh(dhprev)
606
607         dby = np.dot(dhprev, dby)
608
609         dby1 = np.tanh(dby1)
610
611         dhprev = np.tanh(dhprev)
612
613         dby = np.dot(dhprev, dby)
614
615         dby1 = np.tanh(dby1)
616
617         dhprev = np.tanh(dhprev)
618
619         dby = np.dot(dhprev, dby)
620
621         dby1 = np.tanh(dby1)
622
623         dhprev = np.tanh(dhprev)
624
625         dby = np.dot(dhprev, dby)
626
627         dby1 = np.tanh(dby1)
628
629         dhprev = np.tanh(dhprev)
630
631         dby = np.dot(dhprev, dby)
632
633         dby1 = np.tanh(dby1)
634
635         dhprev = np.tanh(dhprev)
636
637         dby = np.dot(dhprev, dby)
638
639         dby1 = np.tanh(dby1)
640
641         dhprev = np.tanh(dhprev)
642
643         dby = np.dot(dhprev, dby)
644
645         dby1 = np.tanh(dby1)
646
647         dhprev = np.tanh(dhprev)
648
649         dby = np.dot(dhprev, dby)
650
651         dby1 = np.tanh(dby1)
652
653         dhprev = np.tanh(dhprev)
654
655         dby = np.dot(dhprev, dby)
656
657         dby1 = np.tanh(dby1)
658
659         dhprev = np.tanh(dhprev)
660
661         dby = np.dot(dhprev, dby)
662
663         dby1 = np.tanh(dby1)
664
665         dhprev = np.tanh(dhprev)
666
667         dby = np.dot(dhprev, dby)
668
669         dby1 = np.tanh(dby1)
670
671         dhprev = np.tanh(dhprev)
672
673         dby = np.dot(dhprev, dby)
674
675         dby1 = np.tanh(dby1)
676
677         dhprev = np.tanh(dhprev)
678
679         dby = np.dot(dhprev, dby)
680
681         dby1 = np.tanh(dby1)
682
683         dhprev = np.tanh(dhprev)
684
685         dby = np.dot(dhprev, dby)
686
687         dby1 = np.tanh(dby1)
688
689         dhprev = np.tanh(dhprev)
690
691         dby = np.dot(dhprev, dby)
692
693         dby1 = np.tanh(dby1)
694
695         dhprev = np.tanh(dhprev)
696
697         dby = np.dot(dhprev, dby)
698
699         dby1 = np.tanh(dby1)
700
701         dhprev = np.tanh(dhprev)
702
703         dby = np.dot(dhprev, dby)
704
705         dby1 = np.tanh(dby1)
706
707         dhprev = np.tanh(dhprev)
708
709         dby = np.dot(dhprev, dby)
710
711         dby1 = np.tanh(dby1)
712
713         dhprev = np.tanh(dhprev)
714
715         dby = np.dot(dhprev, dby)
716
717         dby1 = np.tanh(dby1)
718
719         dhprev = np.tanh(dhprev)
720
721         dby = np.dot(dhprev, dby)
722
723         dby1 = np.tanh(dby1)
724
725         dhprev = np.tanh(dhprev)
726
727         dby = np.dot(dhprev, dby)
728
729         dby1 = np.tanh(dby1)
730
731         dhprev = np.tanh(dhprev)
732
733         dby = np.dot(dhprev, dby)
734
735         dby1 = np.tanh(dby1)
736
737         dhprev = np.tanh(dhprev)
738
739         dby = np.dot(dhprev, dby)
740
741         dby1 = np.tanh(dby1)
742
743         dhprev = np.tanh(dhprev)
744
745         dby = np.dot(dhprev, dby)
746
747         dby1 = np.tanh(dby1)
748
749         dhprev = np.tanh(dhprev)
750
751         dby = np.dot(dhprev, dby)
752
753         dby1 = np.tanh(dby1)
754
755         dhprev = np.tanh(dhprev)
756
757         dby = np.dot(dhprev, dby)
758
759         dby1 = np.tanh(dby1)
760
761         dhprev = np.tanh(dhprev)
762
763         dby = np.dot(dhprev, dby)
764
765         dby1 = np.tanh(dby1)
766
767         dhprev = np.tanh(dhprev)
768
769         dby = np.dot(dhprev, dby)
770
771         dby1 = np.tanh(dby1)
772
773         dhprev = np.tanh(dhprev)
774
775         dby = np.dot(dhprev, dby)
776
777         dby1 = np.tanh(dby1)
778
779         dhprev = np.tanh(dhprev)
780
781         dby = np.dot(dhprev, dby)
782
783         dby1 = np.tanh(dby1)
784
785         dhprev = np.tanh(dhprev)
786
787         dby = np.dot(dhprev, dby)
788
789         dby1 = np.tanh(dby1)
790
791         dhprev = np.tanh(dhprev)
792
793         dby = np.dot(dhprev, dby)
794
795         dby1 = np.tanh(dby1)
796
797         dhprev = np.tanh(dhprev)
798
799         dby = np.dot(dhprev, dby)
800
801         dby1 = np.tanh(dby1)
802
803         dhprev = np.tanh(dhprev)
804
805         dby = np.dot(dhprev, dby)
806
807         dby1 = np.tanh(dby1)
808
809         dhprev = np.tanh(dhprev)
810
811         dby = np.dot(dhprev, dby)
812
813         dby1 = np.tanh(dby1)
814
815         dhprev = np.tanh(dhprev)
816
817         dby = np.dot(dhprev, dby)
818
819         dby1 = np.tanh(dby1)
820
821         dhprev = np.tanh(dhprev)
822
823         dby = np.dot(dhprev, dby)
824
825         dby1 = np.tanh(dby1)
826
827         dhprev = np.tanh(dhprev)
828
829         dby = np.dot(dhprev, dby)
830
831         dby1 = np.tanh(dby1)
832
833         dhprev = np.tanh(dhprev)
834
835         dby = np.dot(dhprev, dby)
836
837         dby1 = np.tanh(dby1)
838
839         dhprev = np.tanh(dhprev)
840
841         dby = np.dot(dhprev, dby)
842
843         dby1 = np.tanh(dby1)
844
845         dhprev = np.tanh(dhprev)
846
847         dby = np.dot(dhprev, dby)
848
849         dby1 = np.tanh(dby1)
850
851         dhprev = np.tanh(dhprev)
852
853         dby = np.dot(dhprev, dby)
854
855         dby1 = np.tanh(dby1)
856
857         dhprev = np.tanh(dhprev)
858
859         dby = np.dot(dhprev, dby)
860
861         dby1 = np.tanh(dby1)
862
863         dhprev = np.tanh(dhprev)
864
865         dby = np.dot(dhprev, dby)
866
867         dby1 = np.tanh(dby1)
868
869         dhprev = np.tanh(dhprev)
870
871         dby = np.dot(dhprev, dby)
872
873         dby1 = np.tanh(dby1)
874
875         dhprev = np.tanh(dhprev)
876
877         dby = np.dot(dhprev, dby)
878
879         dby1 = np.tanh(dby1)
880
881         dhprev = np.tanh(dhprev)
882
883         dby = np.dot(dhprev, dby)
884
885         dby1 = np.tanh(dby1)
886
887         dhprev = np.tanh(dhprev)
888
889         dby = np.dot(dhprev, dby)
890
891         dby1 = np.tanh(dby1)
892
893         dhprev = np.tanh(dhprev)
894
895         dby = np.dot(dhprev, dby)
896
897         dby1 = np.tanh(dby1)
898
899         dhprev = np.tanh(dhprev)
900
901         dby = np.dot(dhprev, dby)
902
903         dby1 = np.tanh(dby1)
904
905         dhprev = np.tanh(dhprev)
906
907         dby = np.dot(dhprev, dby)
908
909         dby1 = np.tanh(dby1)
910
911         dhprev = np.tanh(dhprev)
912
913         dby = np.dot(dhprev, dby)
914
915         dby1 = np.tanh(dby1)
916
917         dhprev = np.tanh(dhprev)
918
919         dby = np.dot(dhprev, dby)
920
921         dby1 = np.tanh(dby1)
922
923         dhprev = np.tanh(dhprev)
924
925         dby = np.dot(dhprev, dby)
926
927         dby1 = np.tanh(dby1)
928
929         dhprev = np.tanh(dhprev)
930
931         dby = np.dot(dhprev, dby)
932
933         dby1 = np.tanh(dby1)
934
935         dhprev = np.tanh(dhprev)
936
937         dby = np.dot(dhprev, dby)
938
939         dby1 = np.tanh(dby1)
940
941         dhprev = np.tanh(dhprev)
942
943         dby = np.dot(dhprev, dby)
944
945         dby1 = np.tanh(dby1)
946
947         dhprev = np.tanh(dhprev)
948
949         dby = np.dot(dhprev, dby)
950
951         dby1 = np.tanh(dby1)
952
953         dhprev = np.tanh(dhprev)
954
955         dby = np.dot(dhprev, dby)
956
957         dby1 = np.tanh(dby1)
958
959         dhprev = np.tanh(dhprev)
960
961         dby = np.dot(dhprev, dby)
962
963         dby1 = np.tanh(dby1)
964
965         dhprev = np.tanh(dhprev)
966
967         dby = np.dot(dhprev, dby)
968
969         dby1 = np.tanh(dby1)
970
971         dhprev = np.tanh(dhprev)
972
973         dby = np.dot(dhprev, dby)
974
975         dby1 = np.tanh(dby1)
976
977         dhprev = np.tanh(dhprev)
978
979         dby = np.dot(dhprev, dby)
980
981         dby1 = np.tanh(dby1)
982
983         dhprev = np.tanh(dhprev)
984
985         dby = np.dot(dhprev, dby)
986
987         dby1 = np.tanh(dby1)
988
989         dhprev = np.tanh(dhprev)
990
991         dby = np.dot(dhprev, dby)
992
993         dby1 = np.tanh(dby1)
994
995         dhprev = np.tanh(dhprev)
996
997         dby = np.dot(dhprev, dby)
998
999         dby1 = np.tanh(dby1)
1000
1001         dhprev = np.tanh(dhprev)
1002
1003         dby = np.dot(dhprev, dby)
1004
1005         dby1 = np.tanh(dby1)
1006
1007         dhprev = np.tanh(dhprev)
1008
1009         dby = np.dot(dhprev, dby)
1010
1011         dby1 = np.tanh(dby1)
1012
1013         dhprev = np.tanh(dhprev)
1014
1015         dby = np.dot(dhprev, dby)
1016
1017         dby1 = np.tanh(dby1)
1018
1019         dhprev = np.tanh(dhprev)
1020
1021         dby = np.dot(dhprev, dby)
1022
1023         dby1 = np.tanh(dby1)
1024
1025         dhprev = np.tanh(dhprev)
1026
1027         dby = np.dot(dhprev, dby)
1028
1029         dby1 = np.tanh(dby1)
1030
1031         dhprev = np.tanh(dhprev)
1032
1033         dby = np.dot(dhprev, dby)
1034
1035         dby1 = np.tanh(dby1)
1036
1037         dhprev = np.tanh(dhprev)
1038
1039         dby = np.dot(dhprev, dby)
1040
1041         dby1 = np.tanh(dby1)
1042
1043         dhprev = np.tanh(dhprev)
1044
1045         dby = np.dot(dhprev, dby)
1046
1047         dby1 = np.tanh(dby1)
1048
1049         dhprev = np.tanh(dhprev)
1050
1051         dby = np.dot(dhprev, dby)
1052
1053         dby1 = np.tanh(dby1)
1054
1055         dhprev = np.tanh(dhprev)
1056
1057         dby = np.dot(dhprev, dby)
1058
1059         dby1 = np.tanh(dby1)
1060
1061         dhprev = np.tanh(dhprev)
1062
1063         dby = np.dot(dhprev, dby)
1064
1065         dby1 = np.tanh(dby1)
1066
1067         dhprev = np.tanh(dhprev)
1068
1069         dby = np.dot(dhprev, dby)
1070
1071         dby1 = np.tanh(dby1)
1072
1073         dhprev = np.tanh(dhprev)
1074
1075         dby = np.dot(dhprev, dby)
1076
1077         dby1 = np.tanh(dby1)
1078
1079         dhprev = np.tanh(dhprev)
1080
1081         dby = np.dot(dhprev, dby)
1082
1083         dby1 = np.tanh(dby1)
1084
1085         dhprev = np.tanh(dhprev)
1086
1087         dby = np.dot(dhprev, dby)
1088
1089         dby1 = np.tanh(dby1)
1090
1091         dhprev = np.tanh(dhprev)
1092
1093         dby = np.dot(dhprev, dby)
1094
1095         dby1 = np.tanh(dby1)
1096
1097         dhprev = np.tanh(dhprev)
1098
1099         dby = np.dot(dhprev, dby)
1100
1101         dby1 = np.tanh(dby1)
1102
1103         dhprev = np.tanh(dhprev)
1104
1105         dby = np.dot(dhprev, dby)
1106
1107         dby1 = np.tanh(dby1)
1108
1109         dhprev = np.tanh(dhprev)
1110
1111         dby = np.dot(dhprev, dby)
1112
1113         dby1 = np.tanh(dby1)
1114
1115         dhprev = np.tanh(dhprev)
1116
1117         dby = np.dot(dhprev, dby)
1118
1119         dby1 = np.tanh(dby1)
1120
1121         dhprev = np.tanh(dhprev)
1122
1123         dby = np.dot(dhprev, dby)
1124
1125         dby1 = np.tanh(dby1)
1126
1127         dhprev = np.tanh(dhprev)
1128
1129         dby = np.dot(dhprev, dby)
1130
1131         dby1 = np.tanh(dby1)
1132
1133         dhprev = np.tanh(dhprev)
1134
1135         dby = np.dot(dhprev, dby)
1136
1137         dby1 = np.tanh(dby1)
1138
1139         dhprev = np.tanh(dhprev)
1140
1141         dby = np.dot(dhprev, dby)
1142
1143         dby1 = np.tanh(dby1)
1144
1145         dhprev = np.tanh(dhprev)
1146
1147         dby = np.dot(dhprev, dby)
1148
1149         dby1 = np.tanh(dby1)
1150
1151         dhprev = np.tanh(dhprev)
1152
1153         dby = np.dot(dhprev, dby)
1154
1155         dby1 = np.tanh(dby1)
1156
1157         dhprev = np.tanh(dhprev)
1158
1159         dby = np.dot(dhprev, dby)
1160
1161         dby1 = np.tanh(dby1)
1162
1163         dhprev = np.tanh(dhprev)
1164
1165         dby = np.dot(dhprev, dby)
1166
1167         dby1 = np.tanh(dby1)
1168
1169         dhprev = np.tanh(dhprev)
1170
1171         dby = np.dot(dhprev, dby)
1172
1173         dby1 = np.tanh(dby1)
1174
1175         dhprev = np.tanh(dhprev)
1176
1177         dby = np.dot(dhprev, dby)
1178
1179         dby1 = np.tanh(dby1)
1180
1181         dhprev = np.tanh(dhprev)
1182
1183         dby = np.dot(dhprev, dby)
1184
1185         dby1 = np.tanh(dby1)
1186
1187         dhprev = np.tanh(dhprev)
1188
1189         dby = np.dot(dhprev, dby)
1190
1191         dby1 = np.tanh(dby1)
1192
1193         dhprev = np.tanh(dhprev)
1194
1195         dby = np.dot(dhprev, dby)
1196
1197         dby1 = np.tanh(dby1)
1198
1199         dhprev = np.tanh(dhprev)
1200
1201         dby = np.dot(dhprev, dby)
1202
1203         dby1 = np.tanh(dby1)
1204
1205         dhprev = np.tanh(dhprev)
1206
1207         dby = np.dot(dhprev, dby)
1208
1209         dby1 = np.tanh(dby1)
1210
1211         dhprev = np.tanh(dhprev)
1212
1213         dby = np.dot(dhprev, dby)
1214
1215         dby1 = np.tanh(dby1)
1216
1217         dhprev = np.tanh(dhprev)
1218
1219         dby = np.dot(dhprev, dby)
1220
1221         dby1 = np.tanh(dby1)
1222
1223         dhprev = np.tanh(dhprev)
1224
1225         dby = np.dot(dhprev, dby)
1226
1227         dby1 = np.tanh(dby1)
1228
1229         dhprev = np.tanh(dhprev)
1230
1231         dby = np.dot(dhprev, dby)
1232
1233         dby1 = np.tanh(dby1)
1234
1235         dhprev = np.tanh(dhprev)
1236
1237         dby = np.dot(dhprev, dby)
1238
1239         dby1 = np.tanh(dby1)
1240
1241         dhprev = np.tanh(dhprev)
1242
1243         dby = np.dot(dhprev, dby)
1244
1245         dby1 = np.tanh(dby1)
1246
1247         dhprev = np.tanh(dhprev)
1248
1249         dby = np.dot(dhprev, dby)
1250
1251         dby1 = np.tanh(dby1)
1252
1253         dhprev = np.tanh(dh
```

```
27 def lossFun(inputs, targets, hprev):  
28     """  
29         inputs,targets are both list of integers.  
30         hprev is Hx1 array of initial hidden state  
31         returns the loss, gradients on model parameters, and last hidden state  
32     """  
33     xs, hs, ys, ps = {}, {}, {}, {}  
34     hs[-1] = np.copy(hprev)  
35     loss = 0  
36     # forward pass  
37     for t in xrange(len(inputs)):  
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation  
39         xs[t][inputs[t]] = 1  
40         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state  
41         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars  
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars  
43         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)  
44         # backward pass: compute gradients here.  
45         # and apply them here.  
46         # next hidden state  
47     return loss, hs[-1], ps
```

$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t &= W_{hy}h_t \end{aligned}$$

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('ptb.train.txt', 'r').read() # should be simple plain text file
9  chars = list(set(data))
10 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 in xrange(len(chars))}
13 ix_to_char = {i:ch for ch in xrange(len(chars))}
14
15 # Hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 20 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # Model parameters
21 wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 bh = 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_0 = np.zeros(hidden_size, 1) # hidden bias
25 by = np.zeros(vocab_size, 1) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """ inputs, targets are both lists of integers.
29     hprev is Hx1 array of initial hidden state
30     returns the loss, gradients on model parameters, and last hidden state
31     """
32
33     xs, hs, ys, ps = np.zeros((0, 0, 0, 0))
34     h0 = np.copy(hprev)
35     loss = 0
36
37     for t in xrange(len(inputs)):
38         x = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
39         x[inputs[t]] = 1
40
41         h0 = np.tanh(np.dot(wh, h0) + np.dot(bh, h0[-1]) + bh)
42         y = np.dot(why, h0) + by
43         y = np.exp(y) / np.sum(np.exp(y)) # softmax
44         ps = y
45         loss += -np.log(ps[targets[t]] / np.sum(np.exp(y)))
46
47     loss /= -np.log(ps[targets[0]] / np.sum(np.exp(y)))
48
49     dnh, dwh, dbh, dhy = np.zeros_like(wh), np.zeros_like(wh), np.zeros_like(bh),
50     dhy = np.zeros_like(why), dbh = np.zeros_like(bh)
51     dnext = np.zeros((vocab_size, 1))
52
53     for t in reversed(xrange(len(inputs))):
54         dy = np.zeros((vocab_size, 1))
55         dy[targets[t]] = -1 # backprop into y
56         dhy += dy
57         dh = np.dot(why.T, dy) + dnext # backprop through tanh nonlinearity
58         dhy += dh
59         dwh += np.dot(dh, xs[t].T)
60         dbh += np.dot(dh, h0[-1].T)
61         dnext = np.dot(wh, dh)
62
63     for dparam in [dwh, dhy, dbh, dhy]:
64         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
65
66     return loss, dwh, dbh, dhy, dhy, h0[seq_length-1]
67
68 def sample(h0, ix, n=100):
69     """ sample a sequence of integers from the model
70     h is memory state, seed_ix is seed integer for first time step
71     """
72
73     x = np.zeros((vocab_size, 1))
74     x[seed_ix] = 1
75
76     for t in xrange(n):
77         h = np.tanh(np.dot(wh, x) + np.dot(bh, h) + bh)
78         p = np.exp(h) / np.sum(np.exp(h))
79         ix = np.random.choice(range(vocab_size), p=p.ravel())
80         x = np.zeros((vocab_size, 1))
81         x[ix] = 1
82
83     return ix
84
85 n, p = 0
86
87 mem, mem0, dhy = np.zeros_like(wh), np.zeros_like(wh), np.zeros_like(wh)
88 mem1, mem2, dwh, dbh, dhy = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(bh)
89 smooth_loss = -np.inf
90 seq_length = len(data)
91 seq_ix = np.zeros(seq_length)
92
93 while True:
94     if p > seq_length - 1 or n == 0:
95         hprev = np.zeros(hidden_size, 1) # reset RNN memory
96         inputs = [char_to_ix[ch] for ch in data[seq_ix[0]:seq_ix[1]]]
97         targets = [char_to_ix[ch] for ch in data[seq_ix[1]:seq_ix[1]+1]]
98
99     if p > seq_length - 1:
100         print 'done'
101         break
102
103     # forward pass: compute activations through the net and fetch gradient
104     loss, dwh, dbh, dhy, dhy, hprev = lossFun(inputs, targets, hprev)
105
106     smooth_loss = smooth_loss * 0.999 + loss * 0.001
107     if n % 100 == 0: print 'iter %d, loss: %f, smooth loss: %f' % (n, loss, smooth_loss)
108
109     dparam = np.zeros_like(wh)
110     for param, dparam in zip([wh, bh, why, dbh, dhy],
111                            [dwh, dbh, dhy, dhy, dhy]):
112         dparam += dparam * learning_rate / dparam # adaptive update
113
114     p = seq_ix[1] # move data pointer
115
116     n += 1 # iteration counter

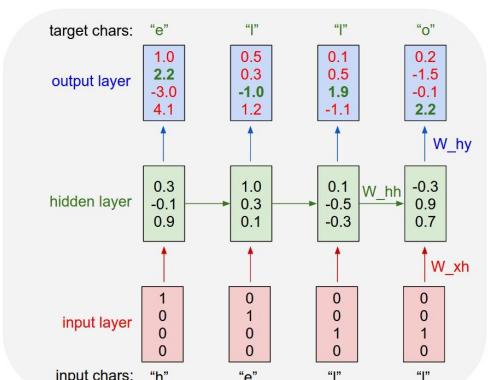
```

```

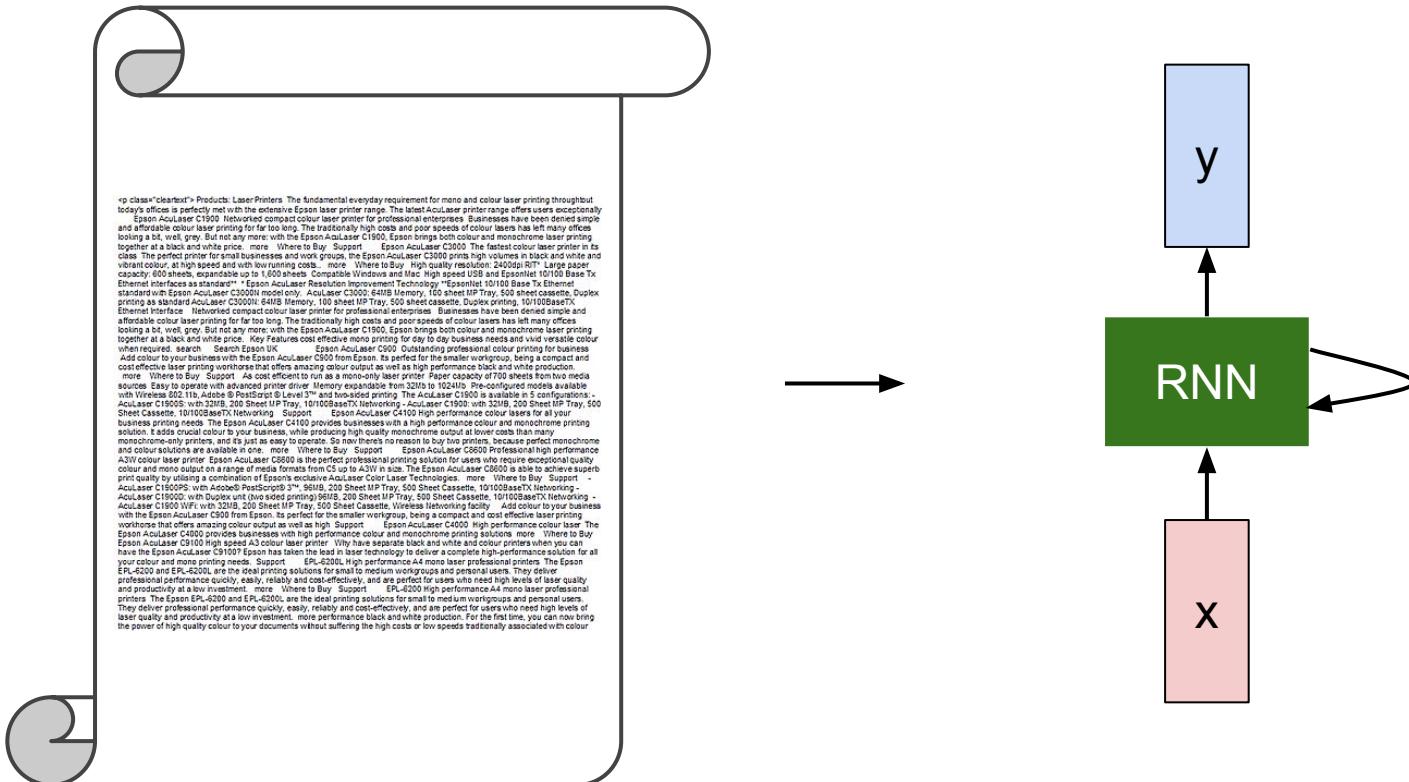
44     # backward pass: compute gradients going backwards
45     dwxh, dwhh, dwhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
46     dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47     dhnext = np.zeros_like(hs[0])
48
49     for t in reversed(xrange(len(inputs))):
50         dy = np.copy(ps[t])
51         dy[targets[t]] -= 1 # backprop into y
52         dwhy += np.dot(dy, hs[t].T)
53         dby += dy
54         dh = np.dot(Why.T, dy) + dhnext # backprop into h
55         ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
56         dbh += ddraw
57         dwhxh += np.dot(ddraw, xs[t].T)
58         dwhh += np.dot(ddraw, hs[t-1].T)
59         dhnext = np.dot(Whh.T, ddraw)
60
61     for dparam in [dwxh, dwhh, dwhy, dbh, dby]:
62         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
63
64     return loss, dwxh, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]

```

recall:



min-char-rnn.py gist



Sonnet 116 – Let me not ...

by William Shakespeare

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

at first:

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

↓ train more

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

↓ train more

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

↓ train more

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

PANDARUS:

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

Second Senator:

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

DUKE VINCENTIO:

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

Second Lord:

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

Clown:

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

VIOLA:

I'll drink it.

VIOLA:

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

KING LEAR:

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

open source textbook on algebraic geometry

 The Stacks Project

[home](#) [about](#) [tags explained](#) [tag lookup](#) [browse](#) [search](#) [bibliography](#) [recent comments](#) [blog](#) [add slogans](#)

Browse chapters

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

Parts

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

Statistics

The Stacks project now consists of

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

Latex source

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

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

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

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

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

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

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

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

is a unique morphism of algebraic stacks. Note that

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

and

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

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

Proof. See discussion of sheaves of sets. \square

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

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

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

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

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

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

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

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

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

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

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

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

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

Proof. Omitted. □

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

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

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

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

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

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

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

Proof. See Spaces, Lemma ??.

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

The following to the construction of the lemma follows.

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

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

be a morphism of algebraic spaces over S and Y .

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

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

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

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

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

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

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

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

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

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

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_{\bar{x}} \rightarrow \mathcal{O}_{X_{\text{étale}}} \rightarrow \mathcal{O}_{X_{\text{ét}}}^{-1} \mathcal{O}_{X_X}(\mathcal{O}_{X_{\eta}}^{\text{v}})$$

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

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

If \mathcal{F} is a scheme theoretic image points. □

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

This repository Search

Explore Gist Blog Help

karpathy + - ⌂ ⌂ ⌂

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 4b1786927d

Category	Commit Message	Time Ago
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
ipc	ipc: fix race in ipc_shm_setattr	a month ago

Pulse

Graphs

HTTPS clone URL

https://github.com/torvalds/linux

You can clone with **HTTPS**, **SSH**, or **Subversion**.

Clone in Desktop

Download ZIP

```
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffffff8) & 0x0000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

Generated C code

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

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

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

```

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

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

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

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

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

```

Searching for interpretable cells

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

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

Searching for interpretable cells

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

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

quote detection cell

Searching for interpretable cells

Cell sensitive to position in line:

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

line length tracking cell

Searching for interpretable cells

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

if statement cell

Searching for interpretable cells

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

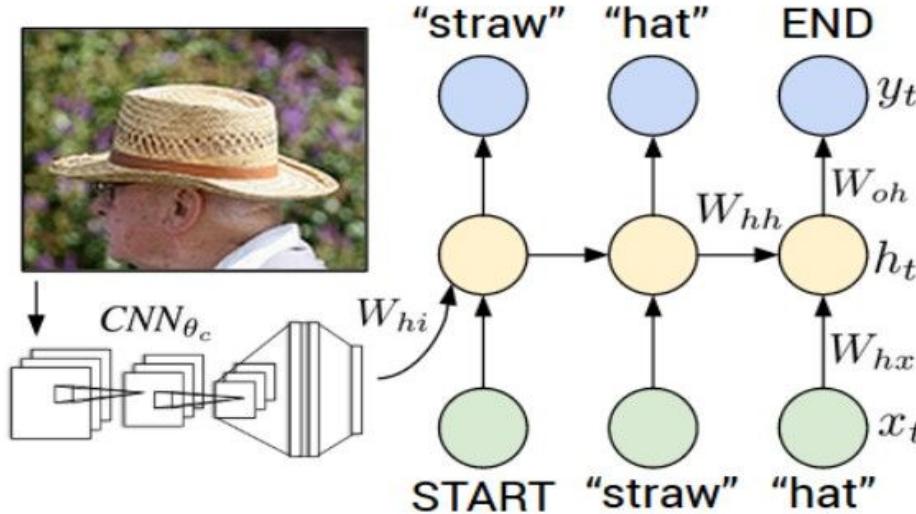
quote/comment cell

Searching for interpretable cells

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

code depth cell

Image Captioning



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

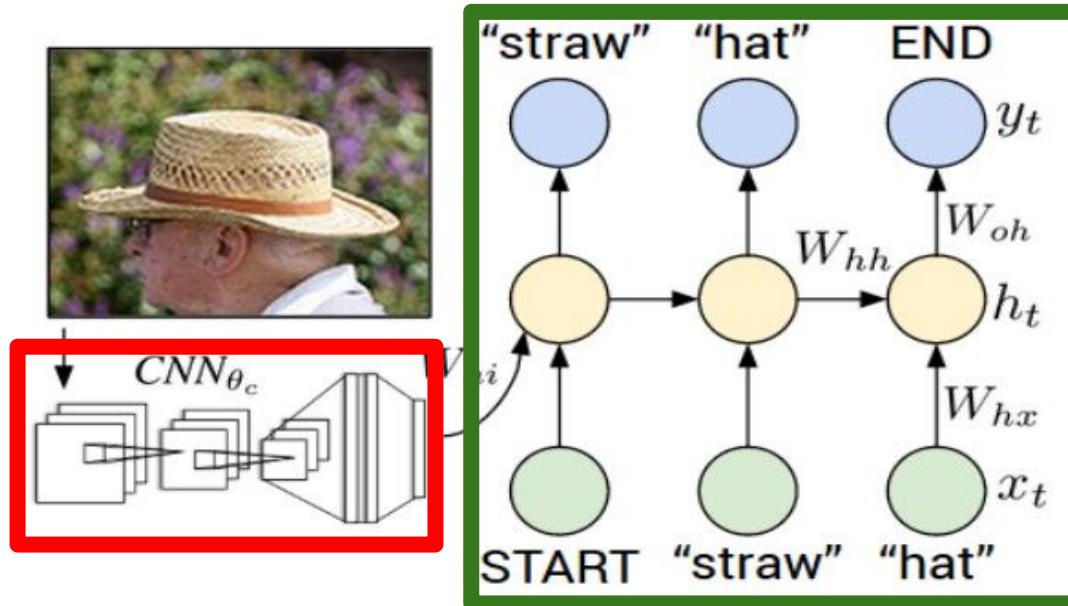
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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

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

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

Recurrent Neural Network



Convolutional Neural Network

test image



image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image

image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

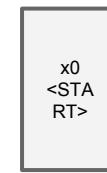
conv-512

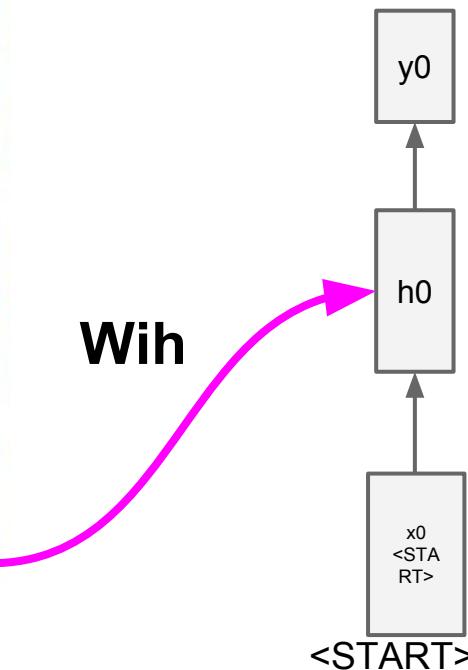
conv-512

maxpool

FC-4096

FC-4096





before:

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

now:

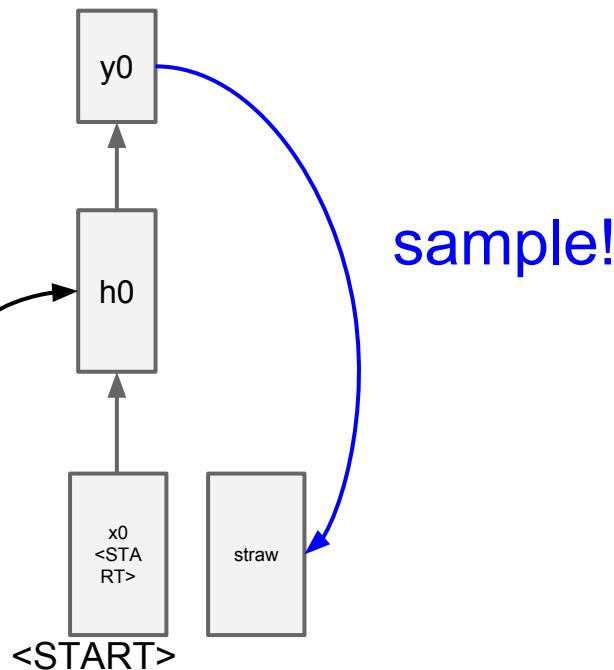
$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$



test image



test image



image



test image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

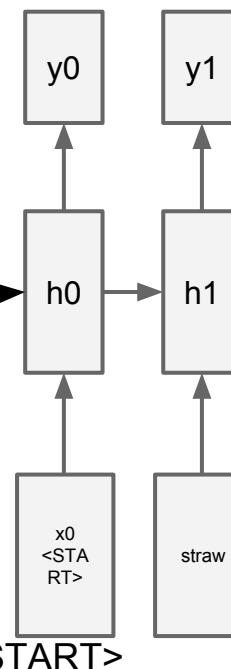
conv-512

conv-512

maxpool

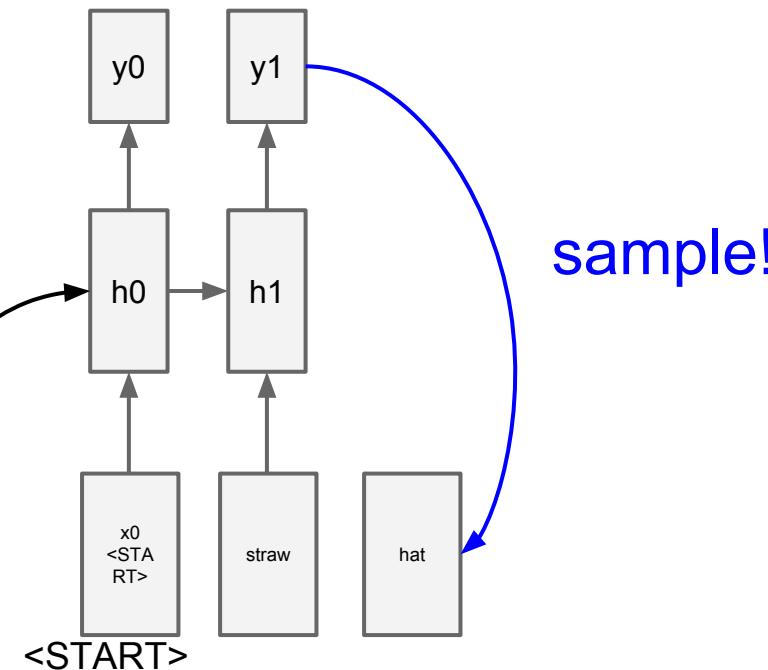
FC-4096

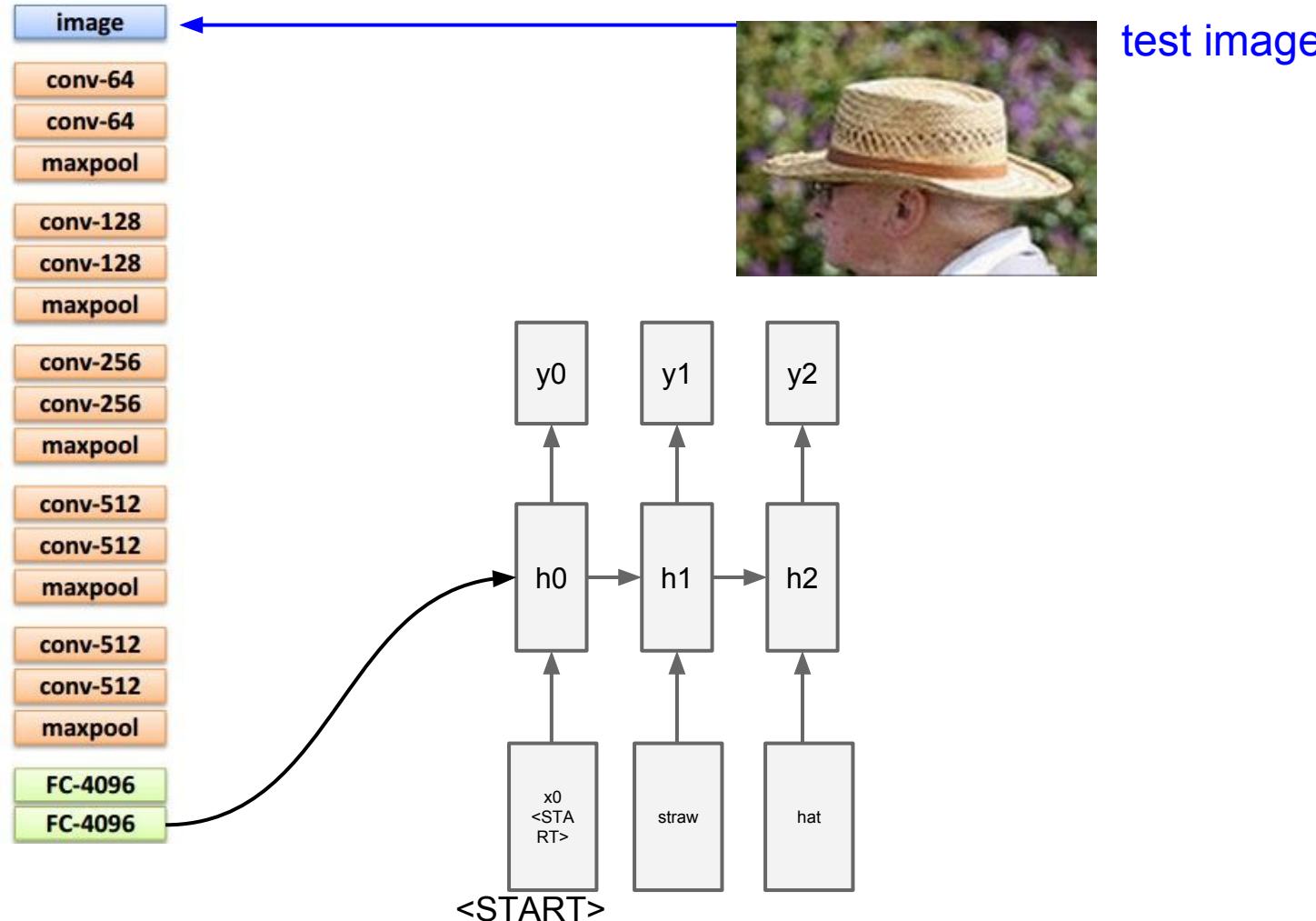
FC-4096





test image







test image

sample
<END> token
=> finish.

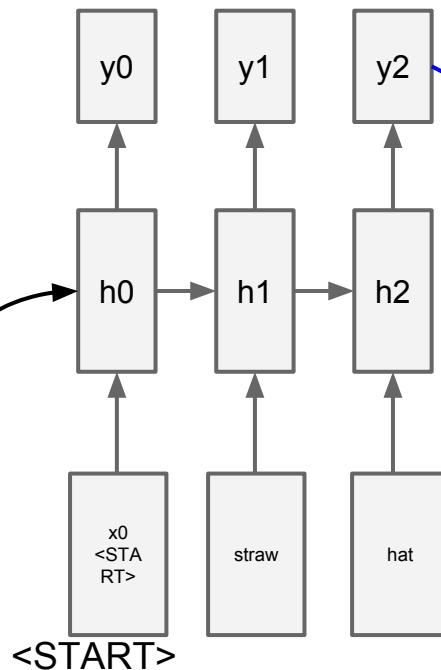


Image Sentence Datasets

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



Microsoft COCO
[Tsung-Yi Lin et al. 2014]
mscoco.org

currently:
~120K images
~5 sentences each



"man in black shirt is playing guitar."



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



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"man in black shirt is playing guitar."



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



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"a young boy is holding a baseball bat."



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



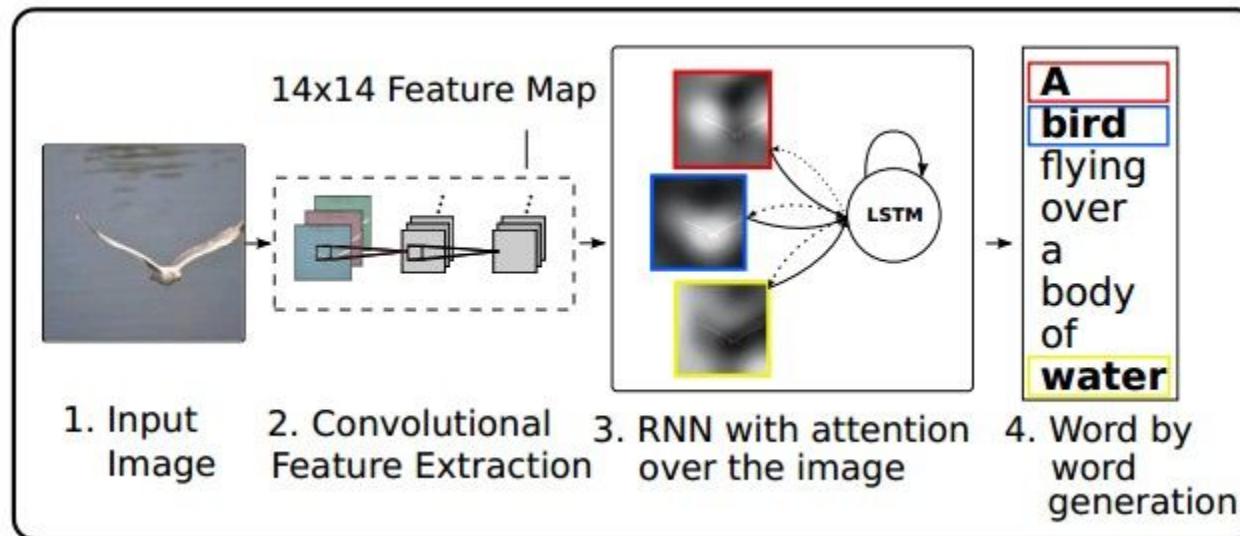
"a woman holding a teddy bear in front of a mirror."



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

Preview of fancier architectures

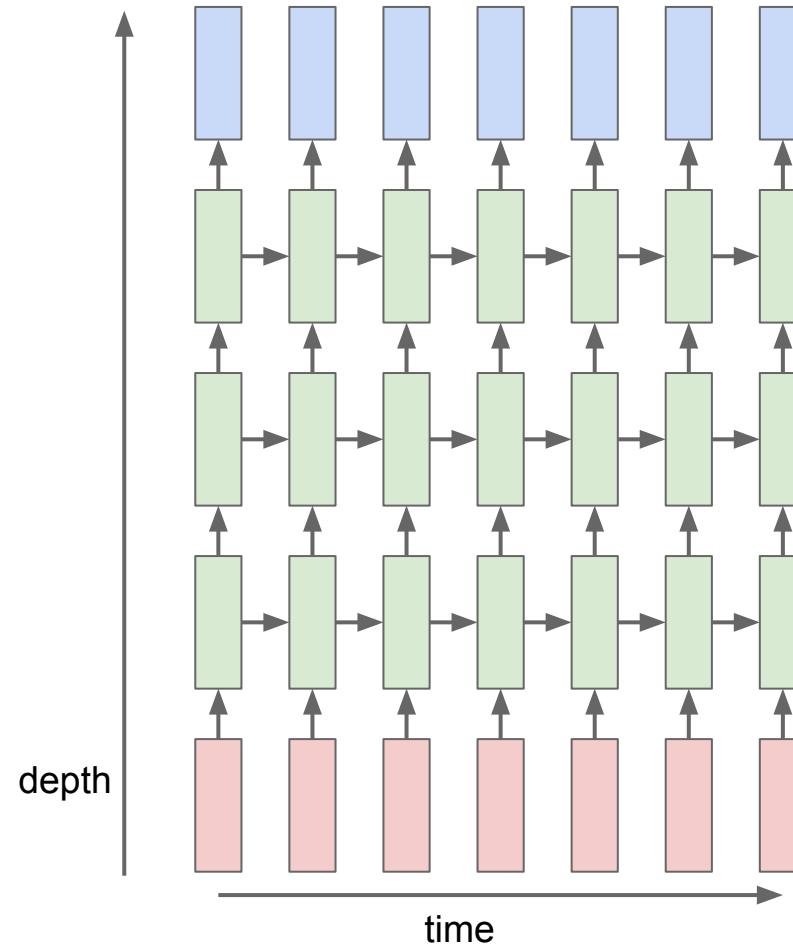
RNN attends spatially to different parts of images while generating each word of the sentence:



RNN:

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

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



RNN:

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

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

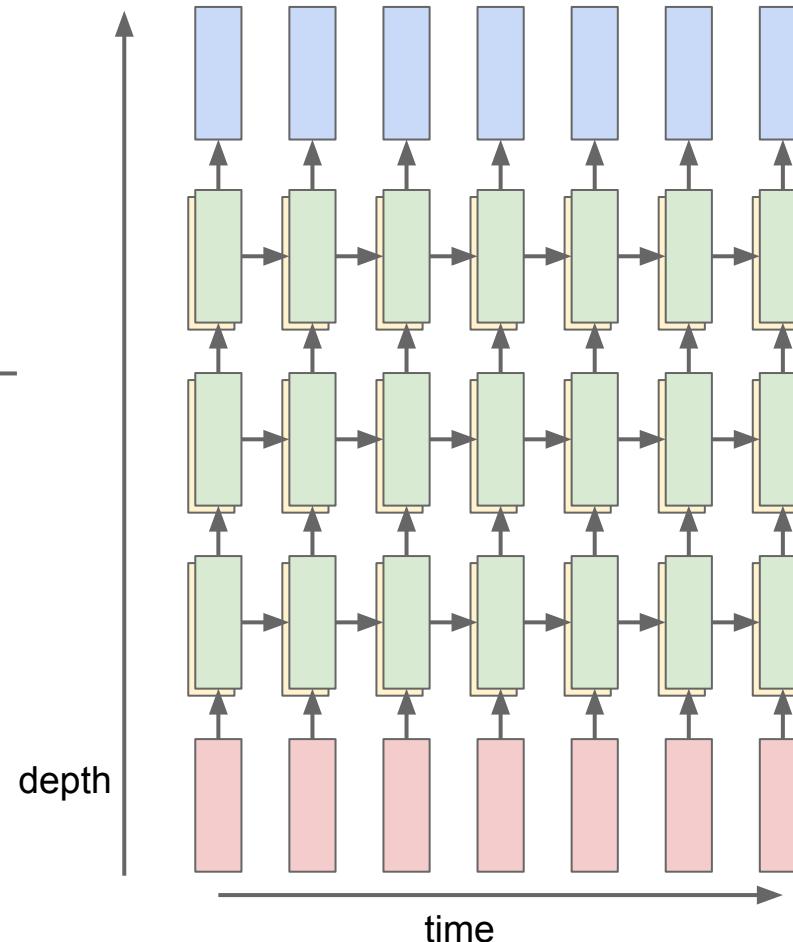
LSTM:

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

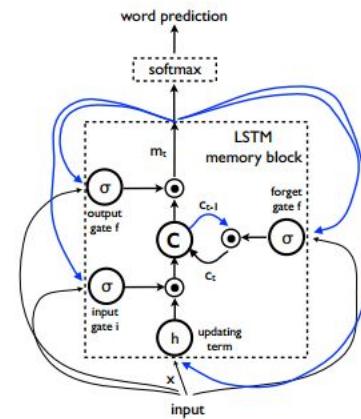
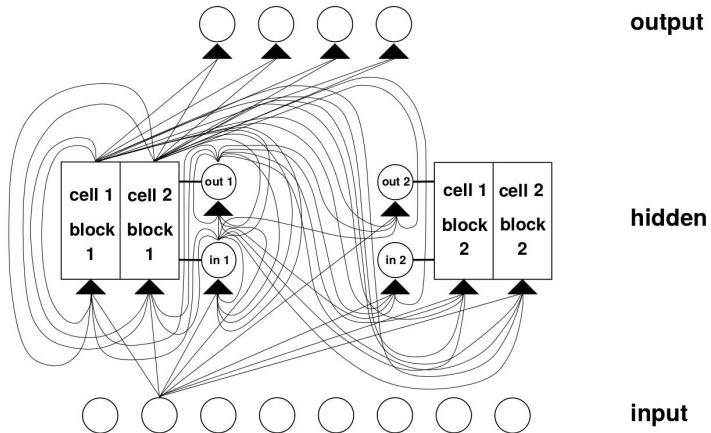
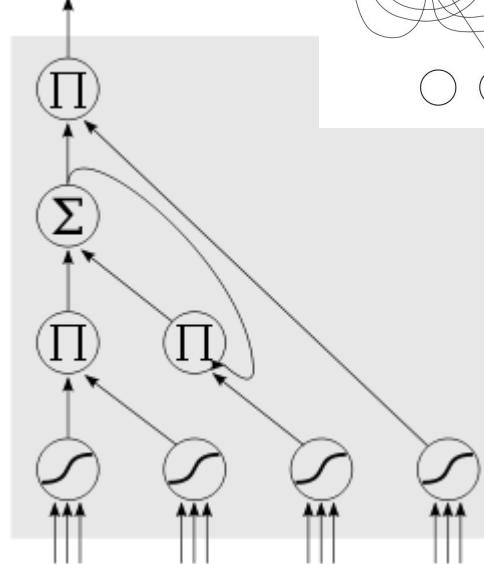
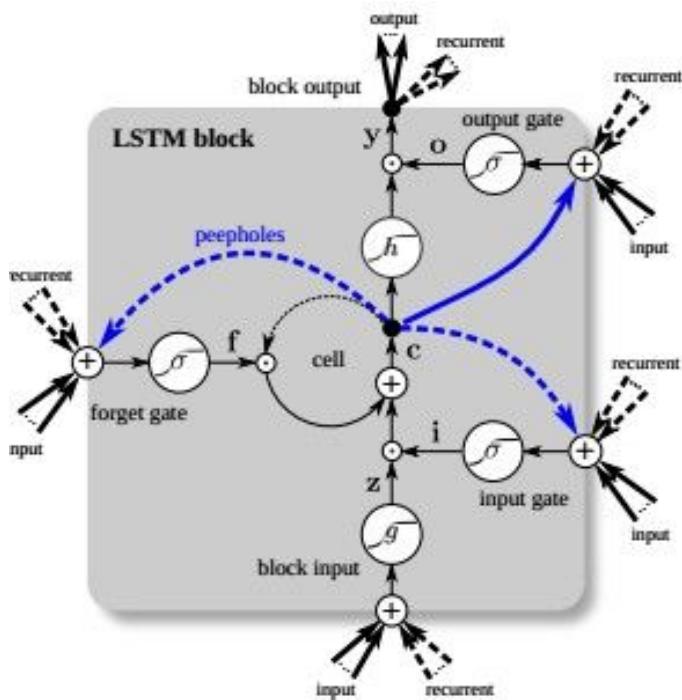
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

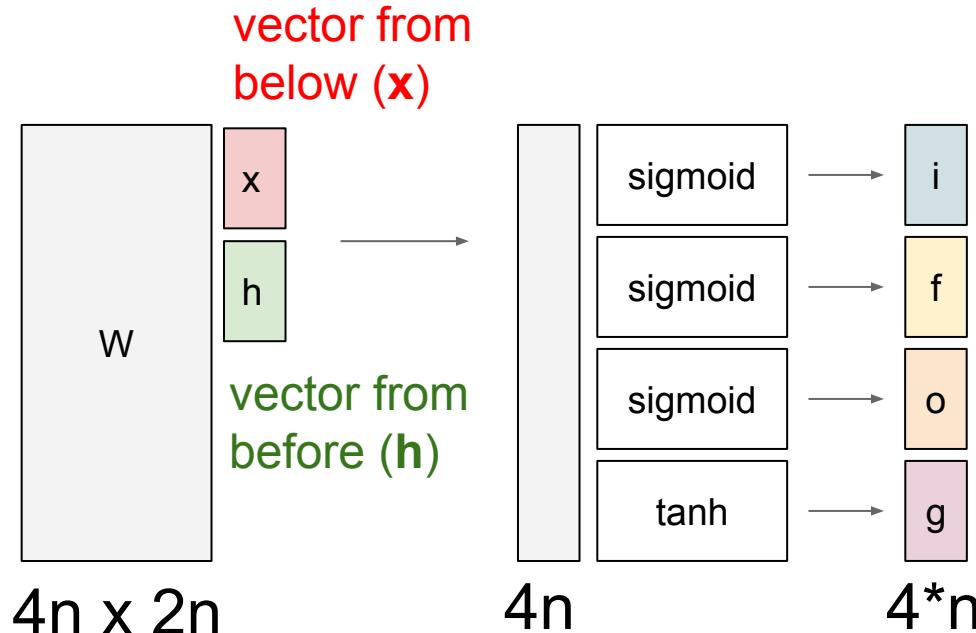


LSTM



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

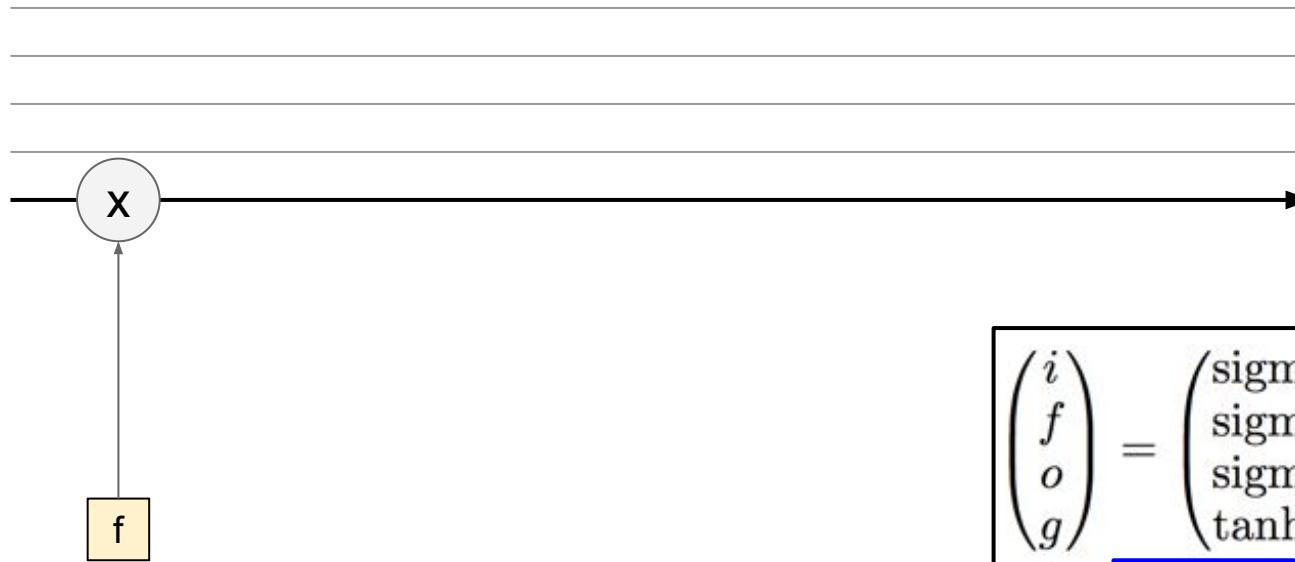


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_t^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c

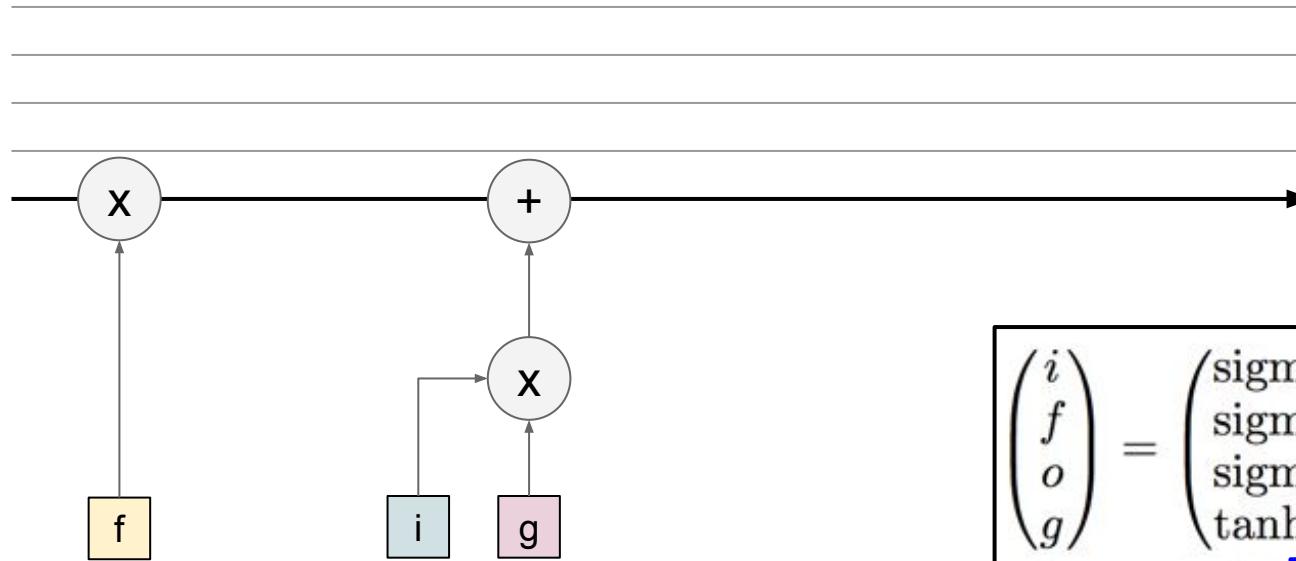


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c

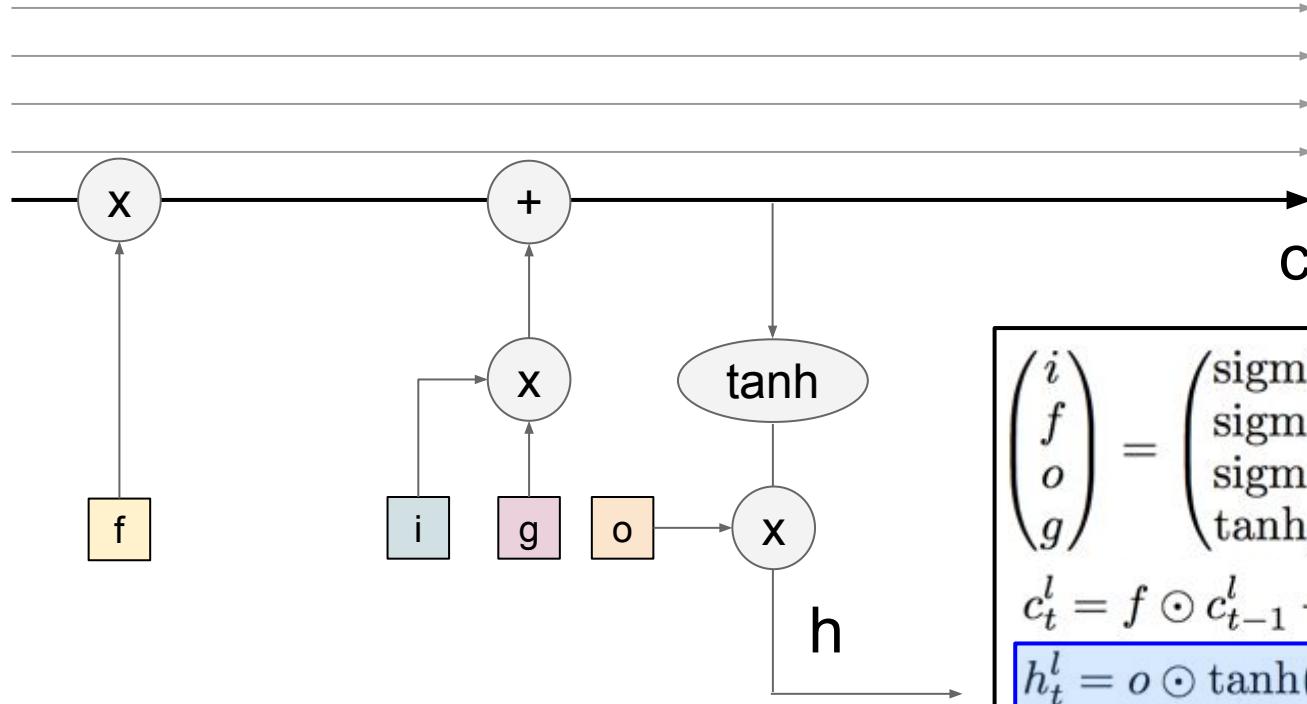


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c

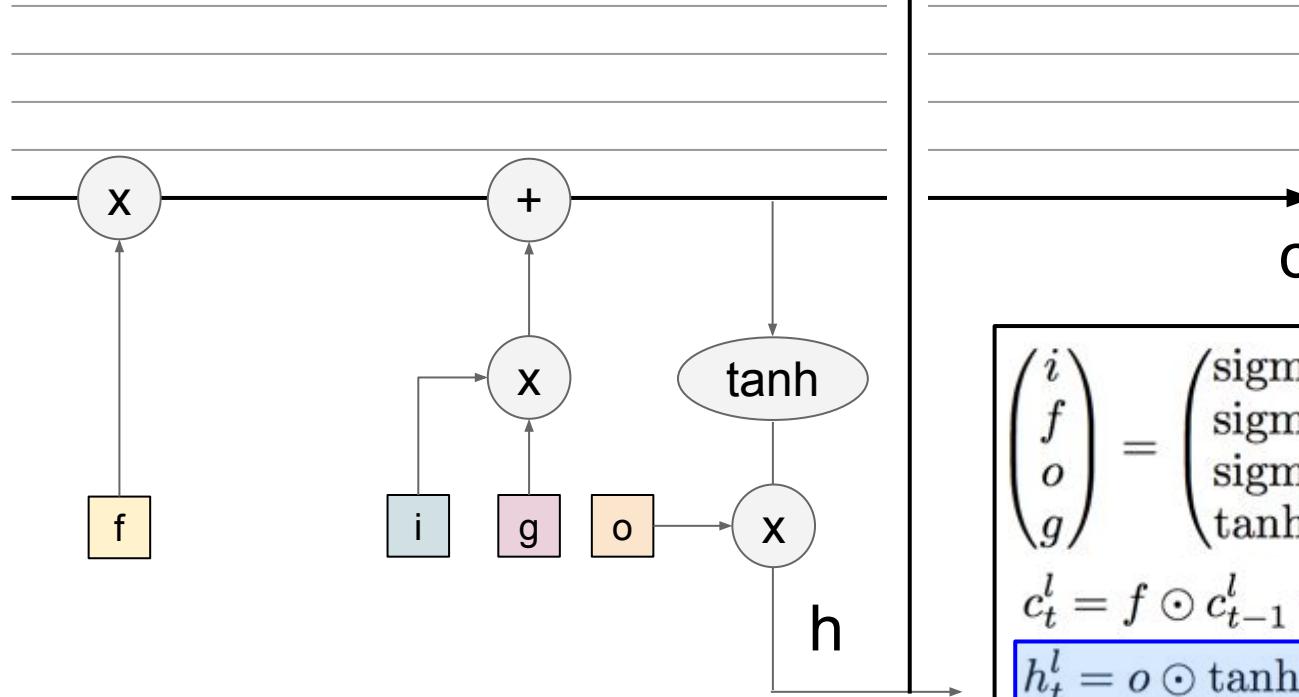


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c



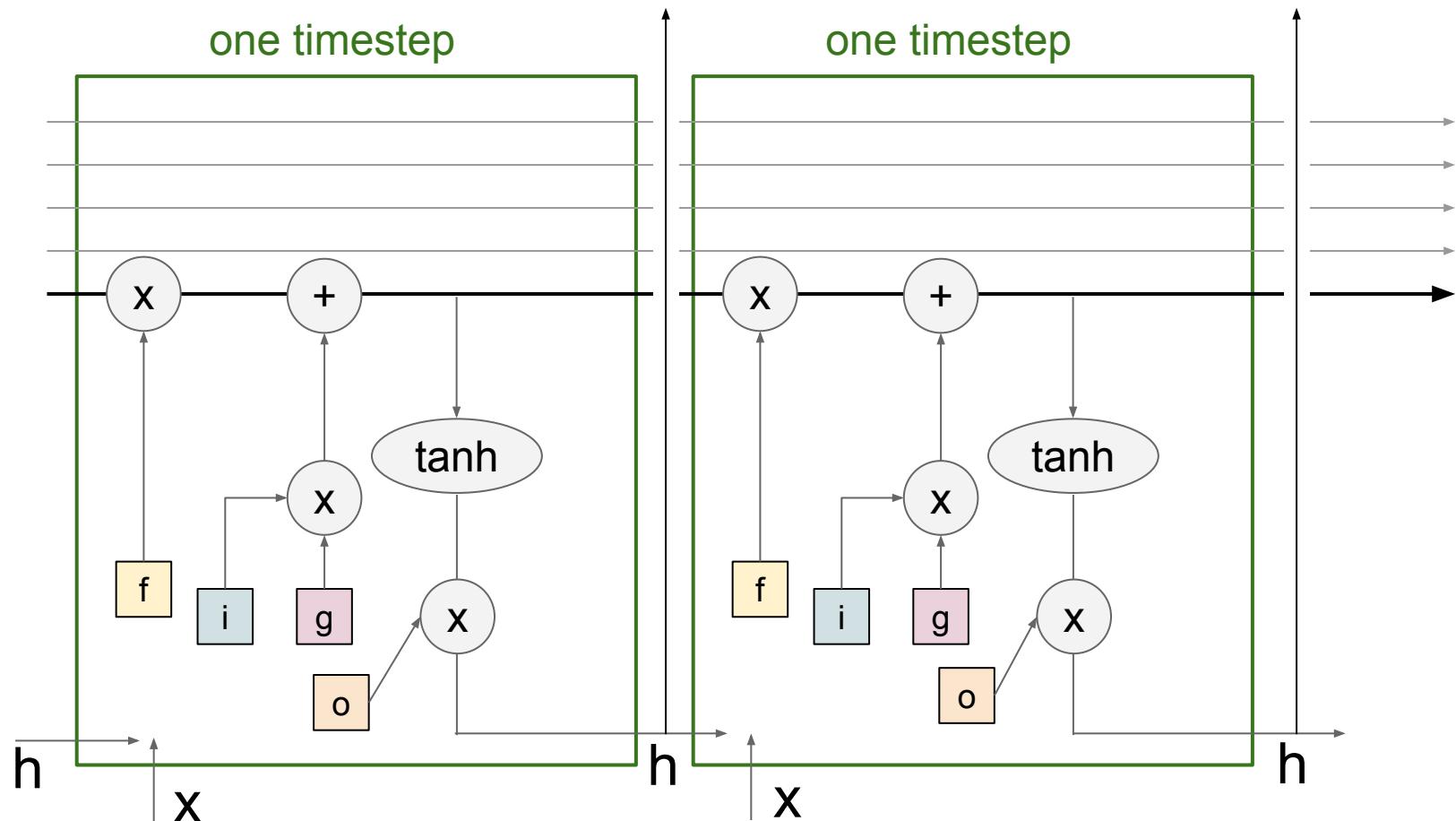
higher layer, or
prediction

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

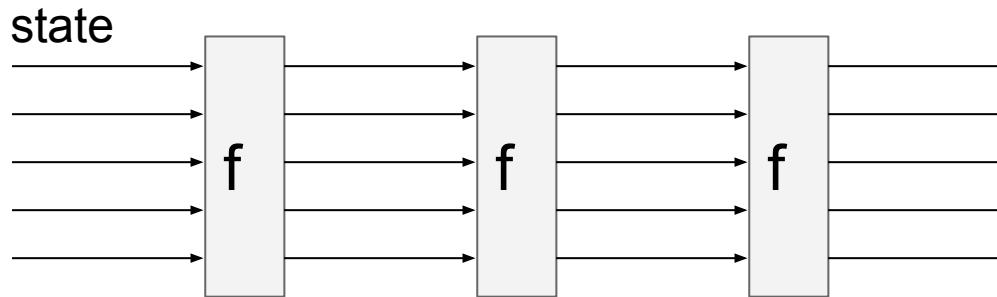
LSTM

one timestep

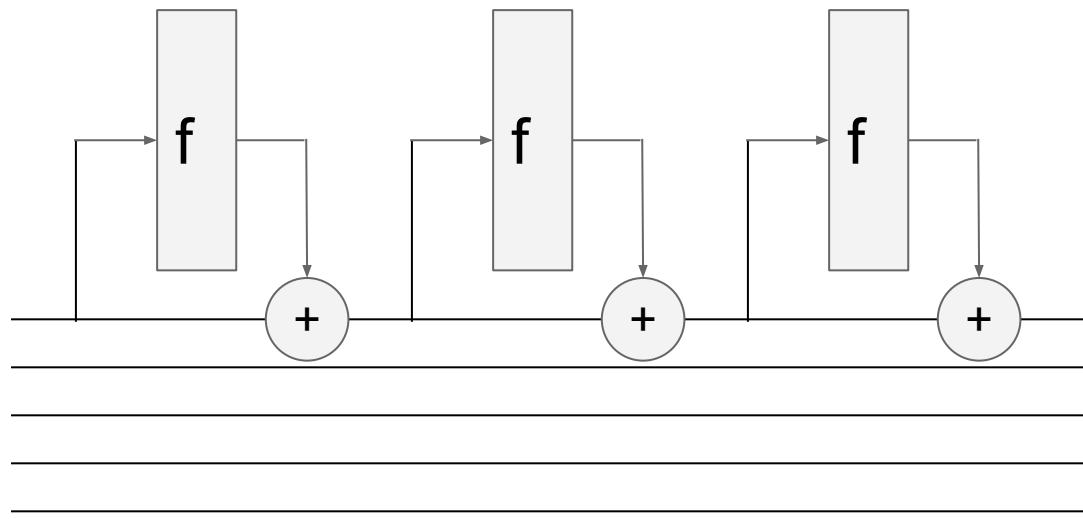
cell
state c



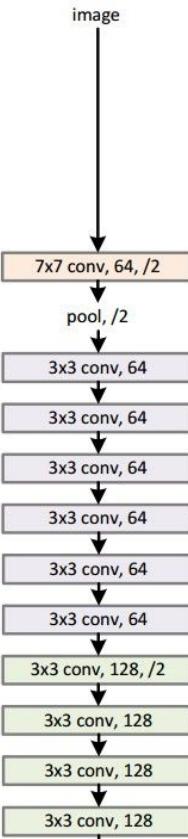
RNN



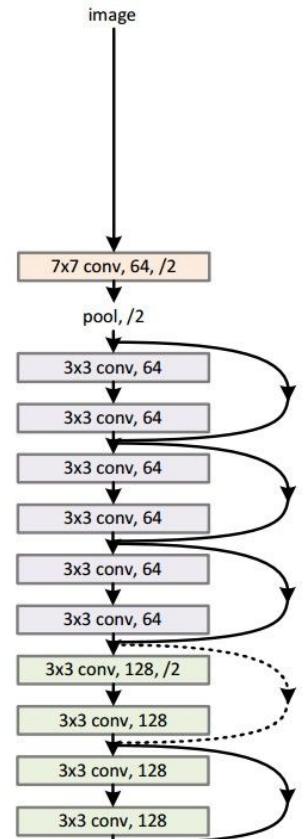
LSTM (ignoring forget gates)



34-layer plain

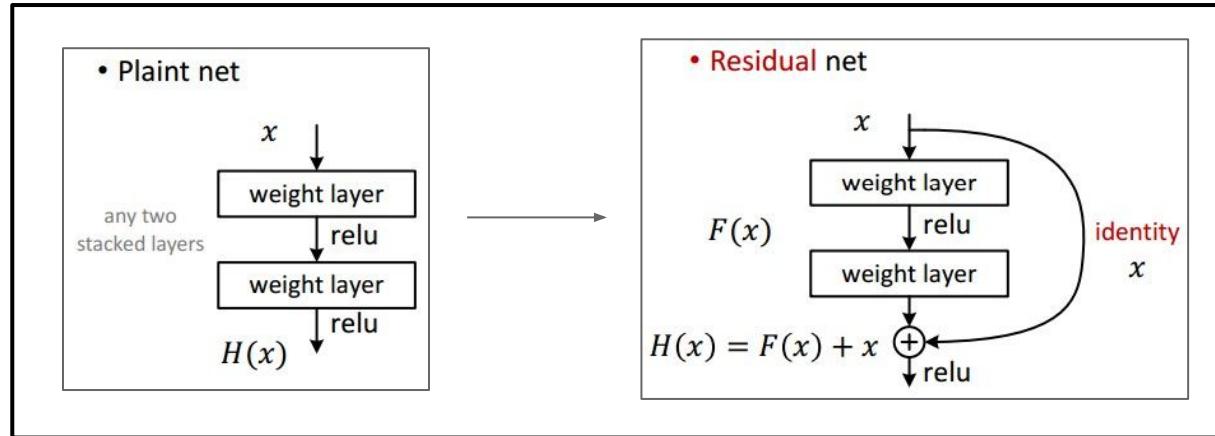


34-layer residual



Recall: “PlainNets” vs. ResNets

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



Understanding gradient flow dynamics

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

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

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

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

Understanding gradient flow dynamics

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

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

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

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

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

Understanding gradient flow dynamics

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

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

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

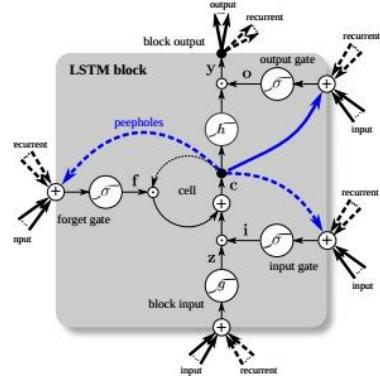
if the largest eigenvalue is > 1 , gradient will explode
if the largest eigenvalue is < 1 , gradient will vanish

can control exploding with gradient clipping
can control vanishing with LSTM

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

LSTM variants and friends

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



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

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

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

MUT1:

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

MUT2:

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

MUT3:

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

Summary

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