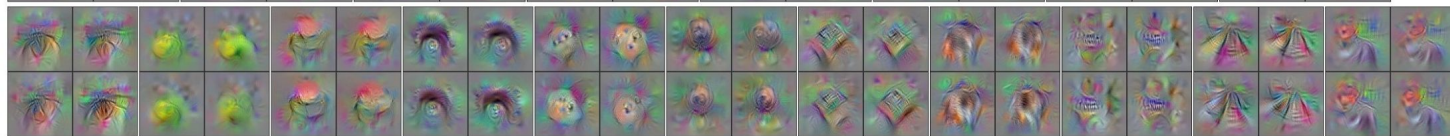
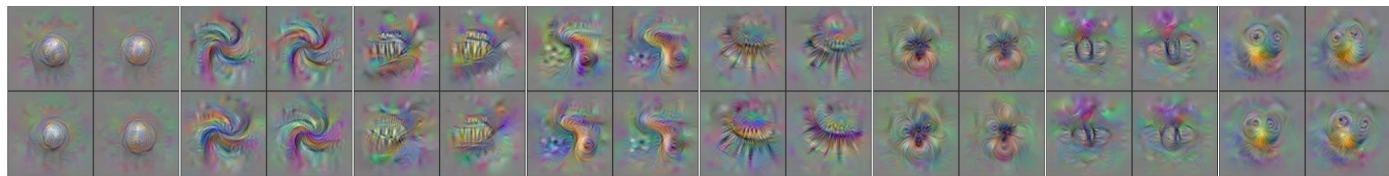
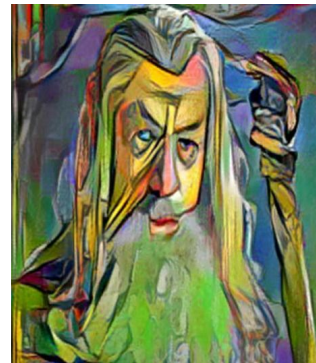
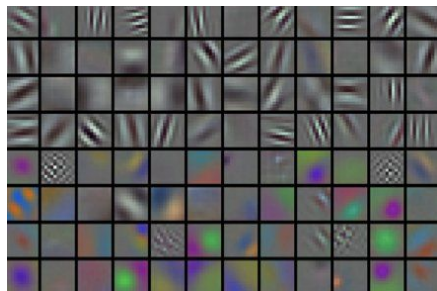


# Lecture 10:

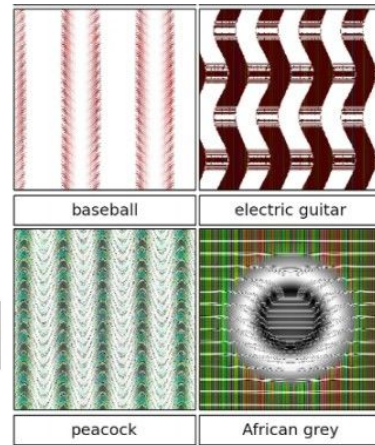
## Recurrent Neural Networks

# Administrative

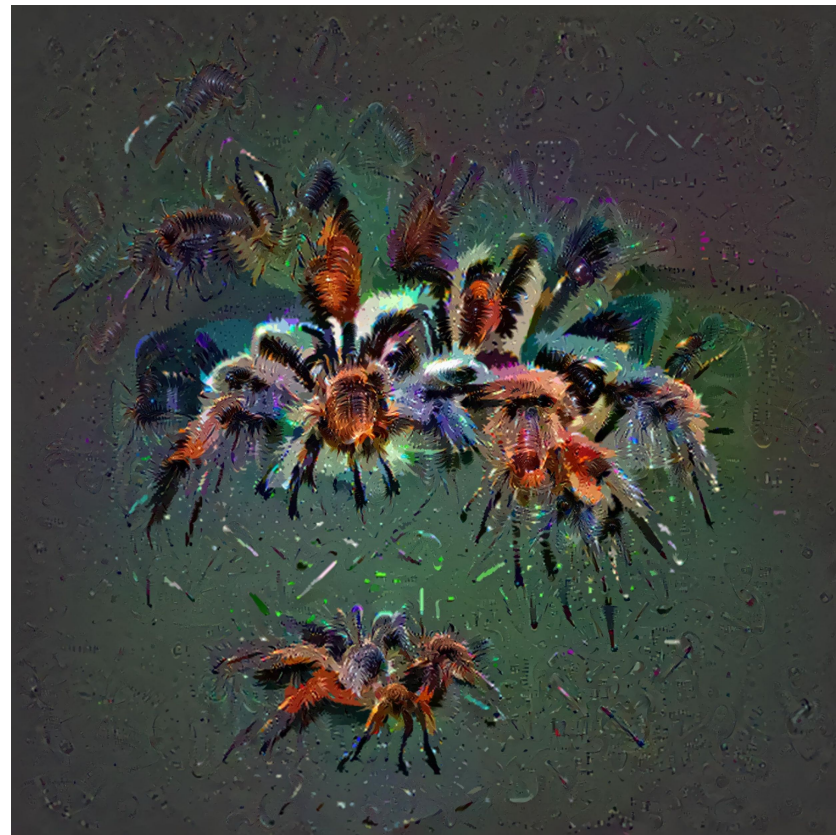
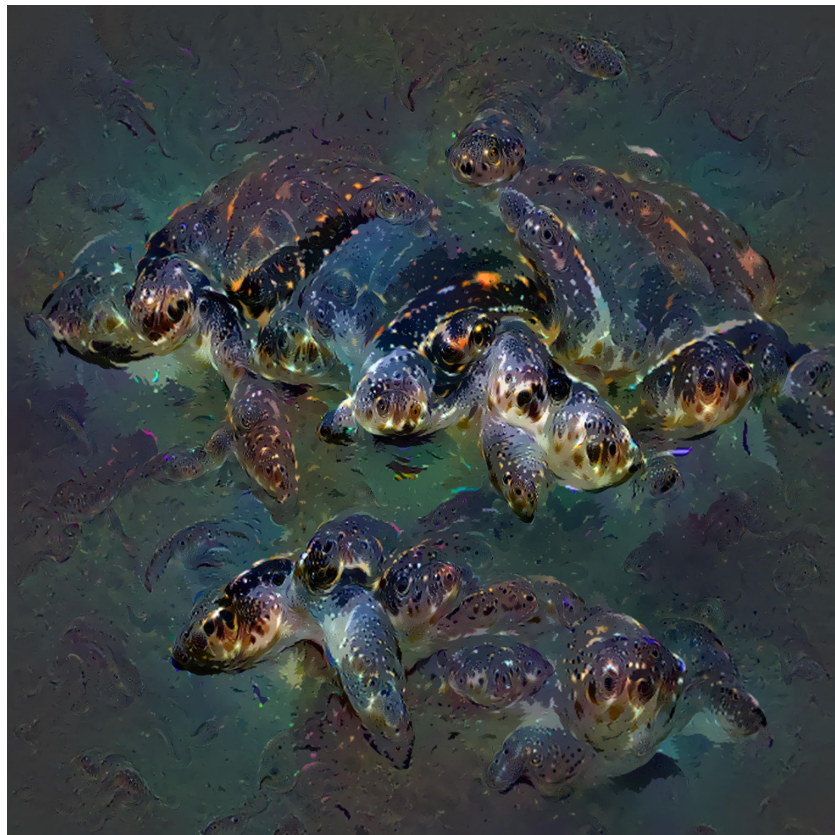
- Midterm this Wednesday! woohoo!
- A3 will be out ~Wednesday

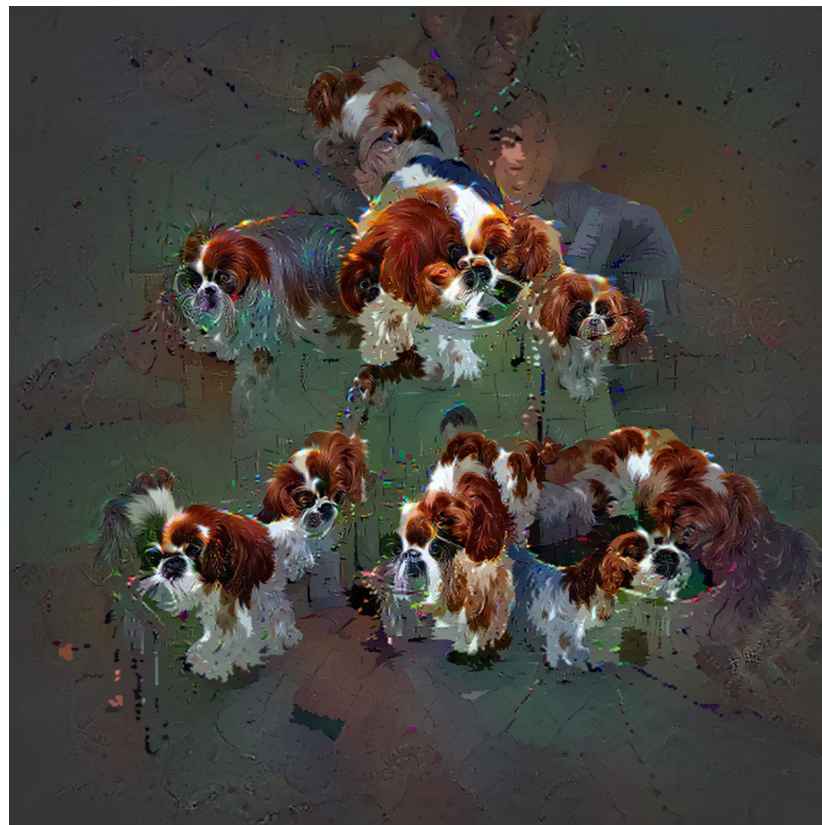


Layer 1





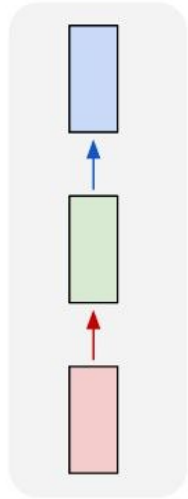




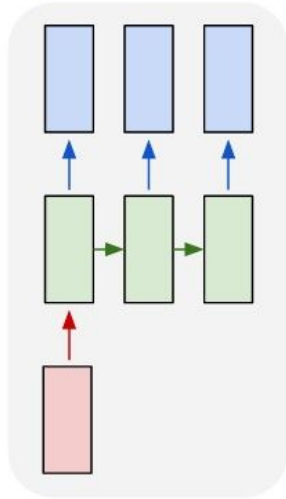


# Recurrent Networks offer a lot of flexibility:

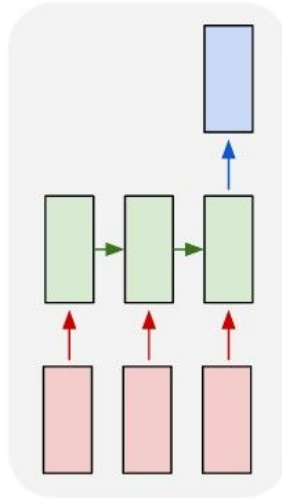
one to one



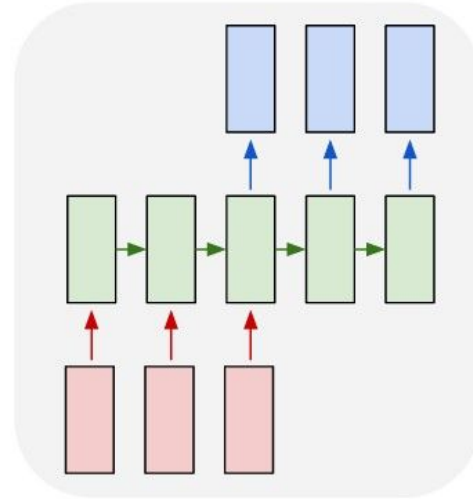
one to many



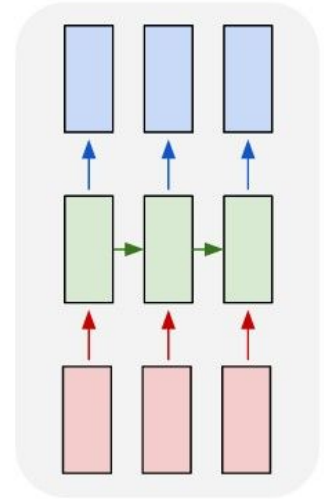
many to one



many to many



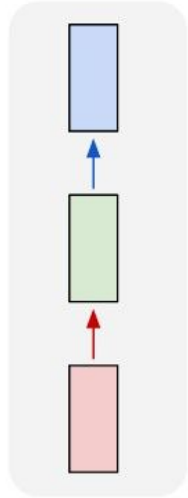
many to many



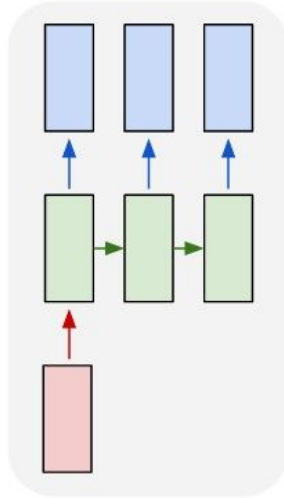
↖ **Vanilla Neural Networks**

# Recurrent Networks offer a lot of flexibility:

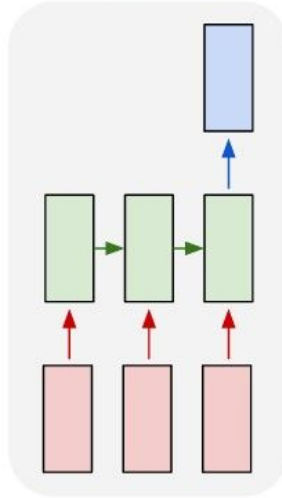
one to one



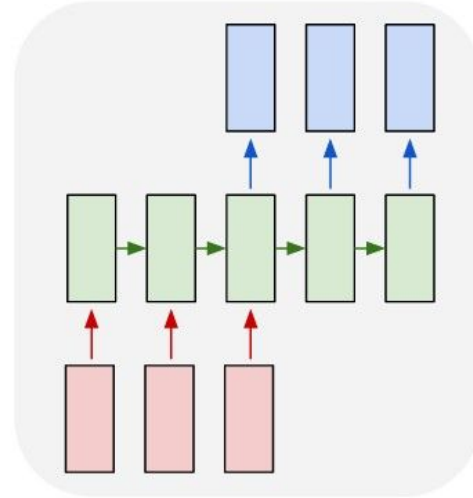
one to many



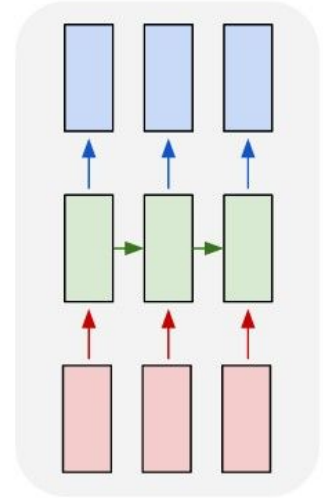
many to one



many to many



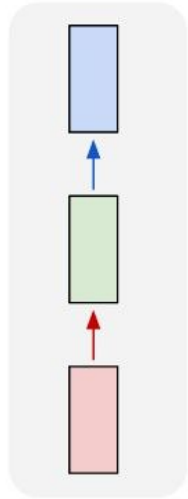
many to many



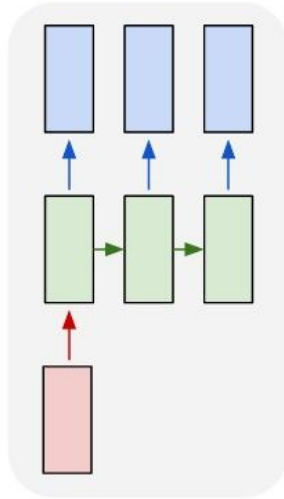
↖ e.g. **Image Captioning**  
image -> sequence of words

# Recurrent Networks offer a lot of flexibility:

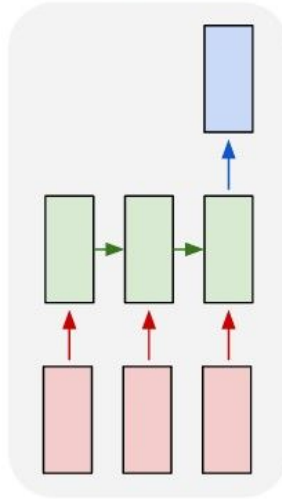
one to one



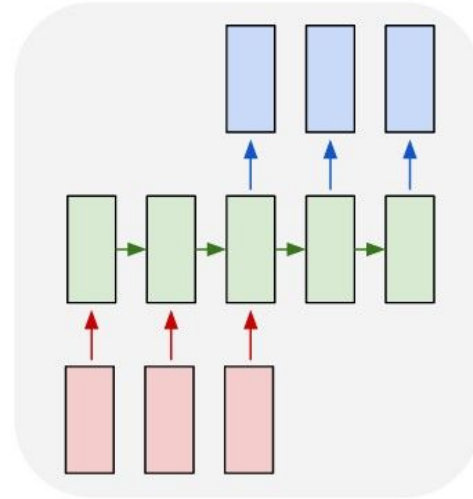
one to many



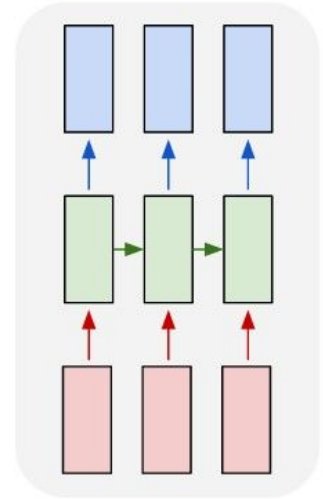
many to one



many to many



many to many

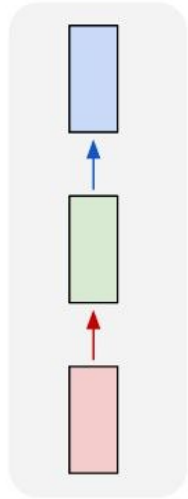


↖ e.g. **Sentiment Classification**  
sequence of words → sentiment

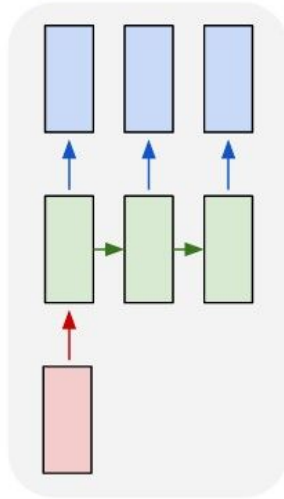


# Recurrent Networks offer a lot of flexibility:

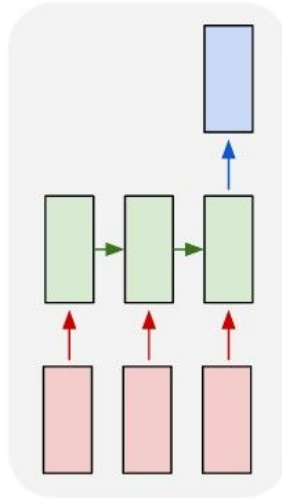
one to one



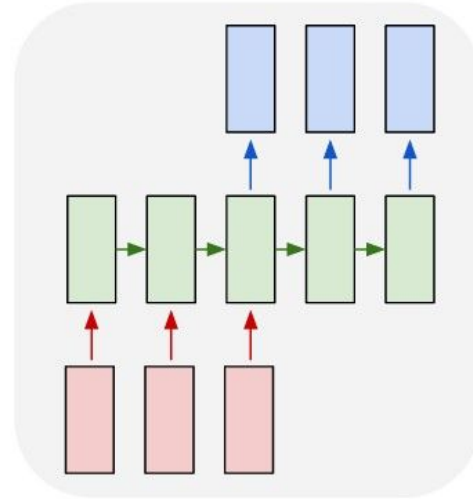
one to many



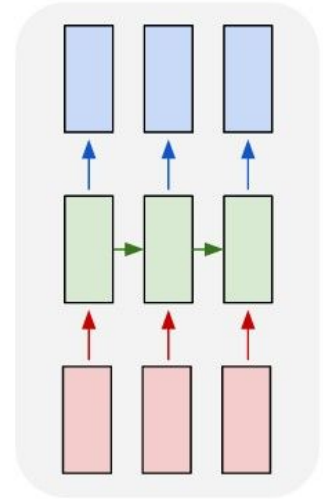
many to one



many to many



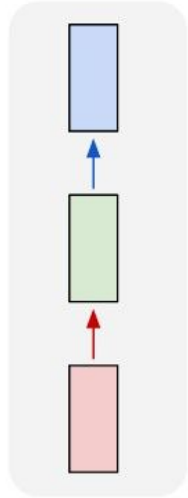
many to many



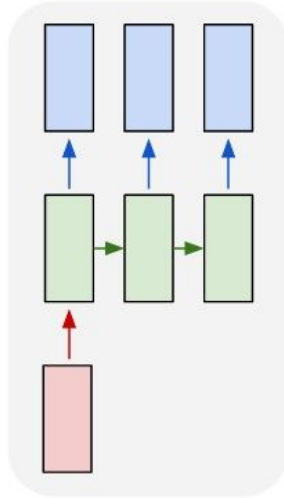
e.g. **Machine Translation**  
seq of words -> seq of words

# Recurrent Networks offer a lot of flexibility:

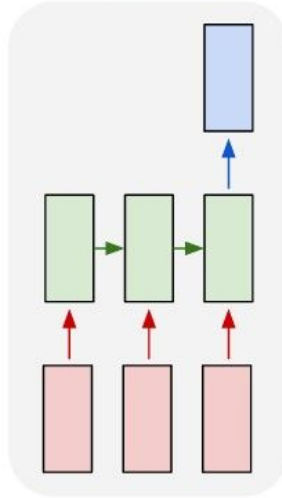
one to one



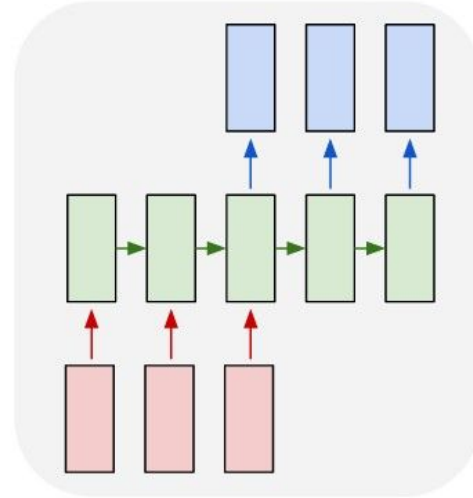
one to many



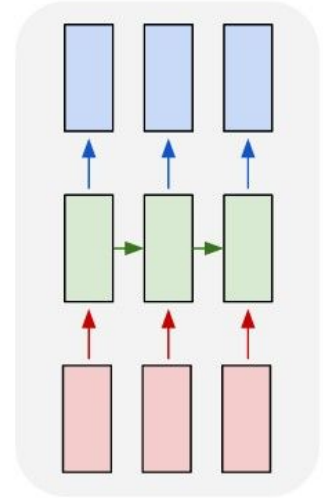
many to one



many to many



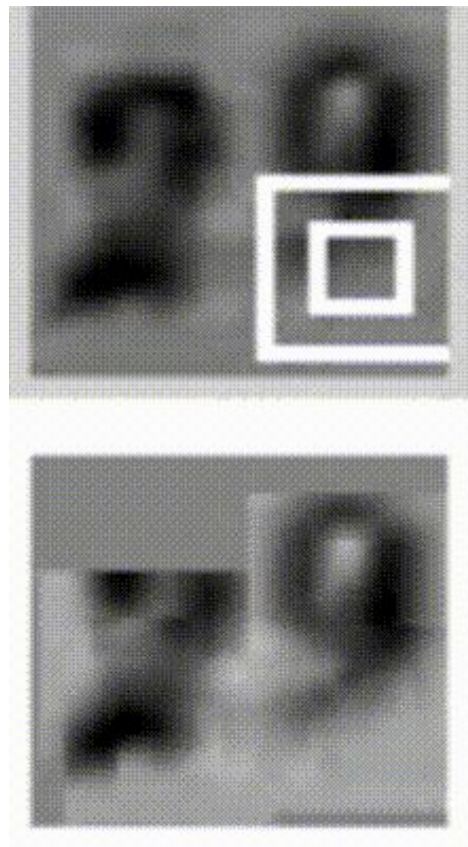
many to many



e.g. Video classification on frame level



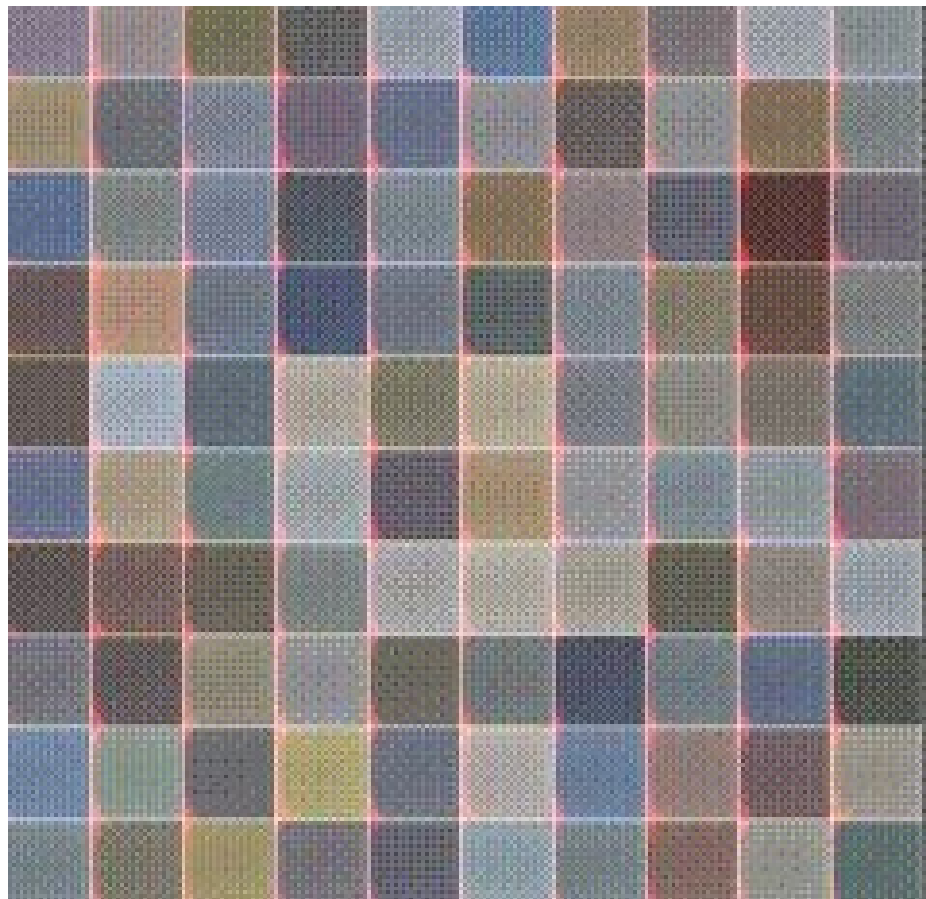
# Sequential Processing of fixed inputs



Multiple Object Recognition with  
Visual Attention, Ba et al.

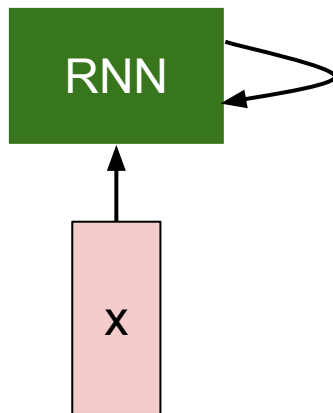
# Sequential Processing of fixed outputs

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

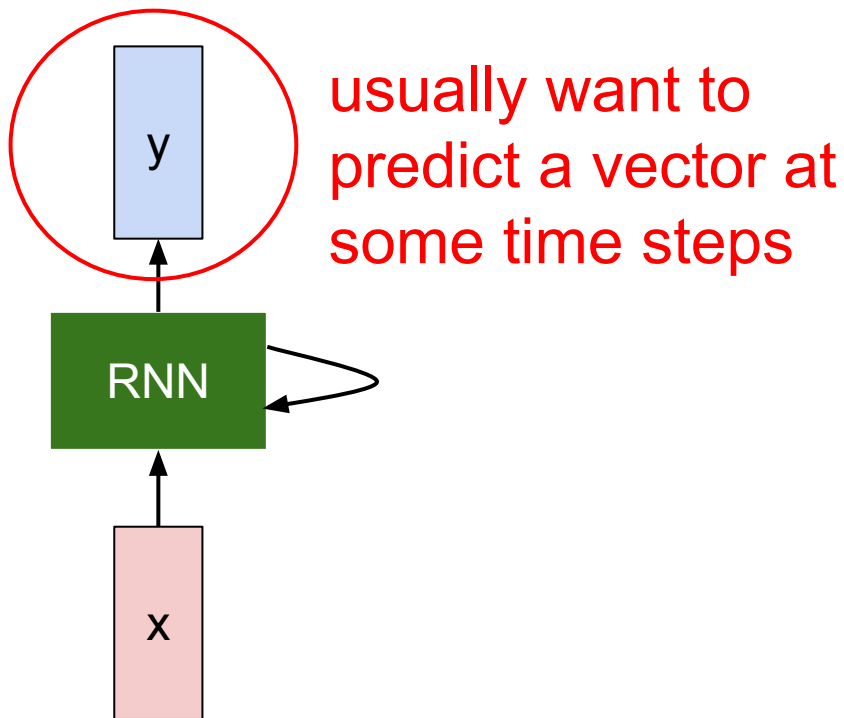




# Recurrent Neural Network



# Recurrent Neural Network



# Recurrent Neural Network

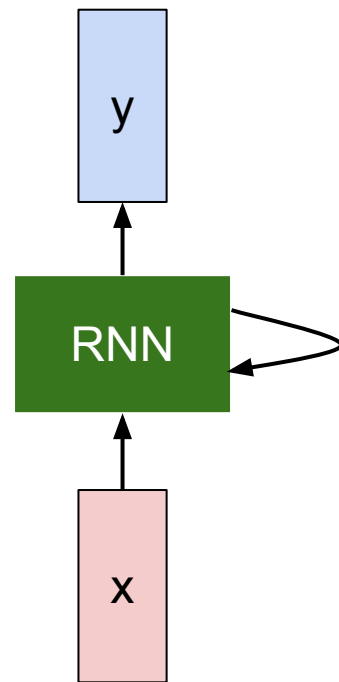
We can process a sequence of vectors  $\mathbf{x}$  by applying a recurrence formula at every time step:

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

new state / some function with parameters  $W$

old state

input vector at some time step

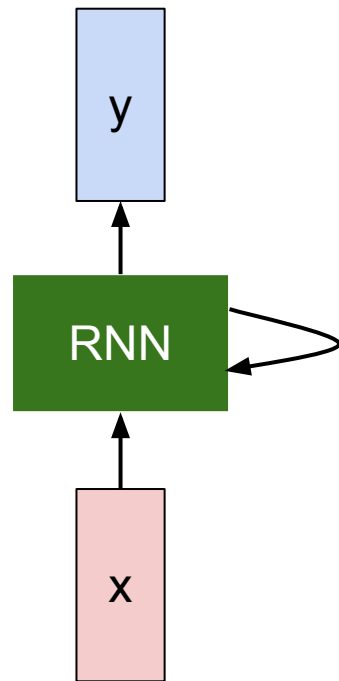


# Recurrent Neural Network

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

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

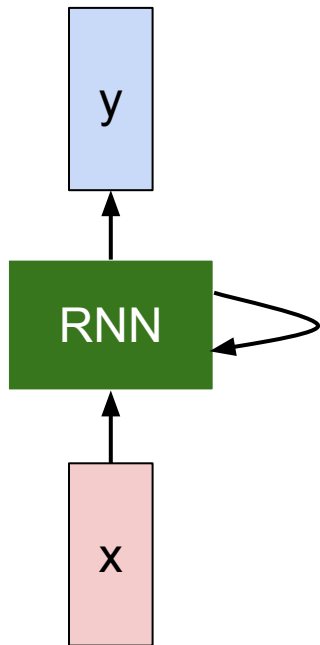
Notice: the same function and the same set of parameters are used at every time step.





# (Vanilla) Recurrent Neural Network

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



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



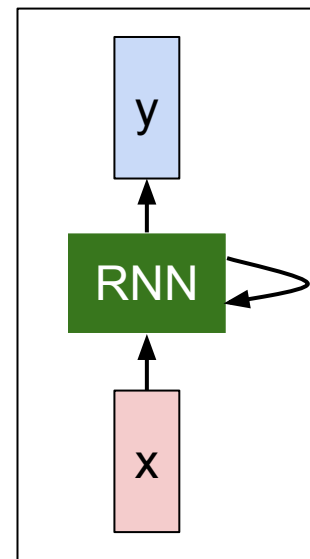
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

# Character-level language model example

Vocabulary:  
[h,e,l,o]

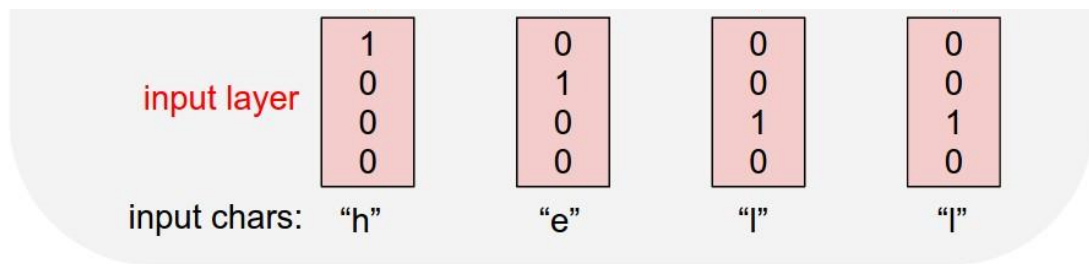
Example training  
sequence:  
**“hello”**



# Character-level language model example

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“hello”

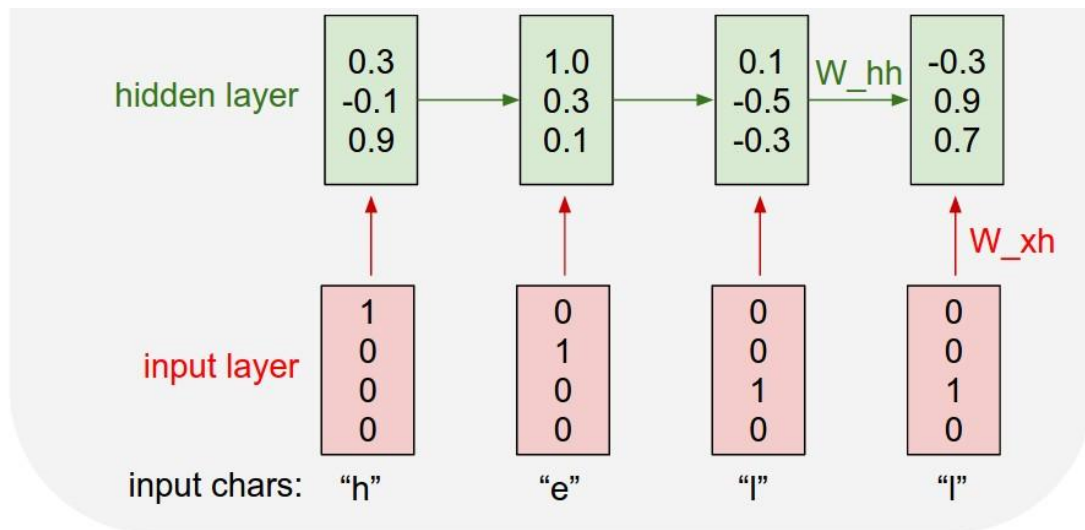


# Character-level language model example

Vocabulary:  
[h,e,l,o]

Example training sequence:  
“hello”

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

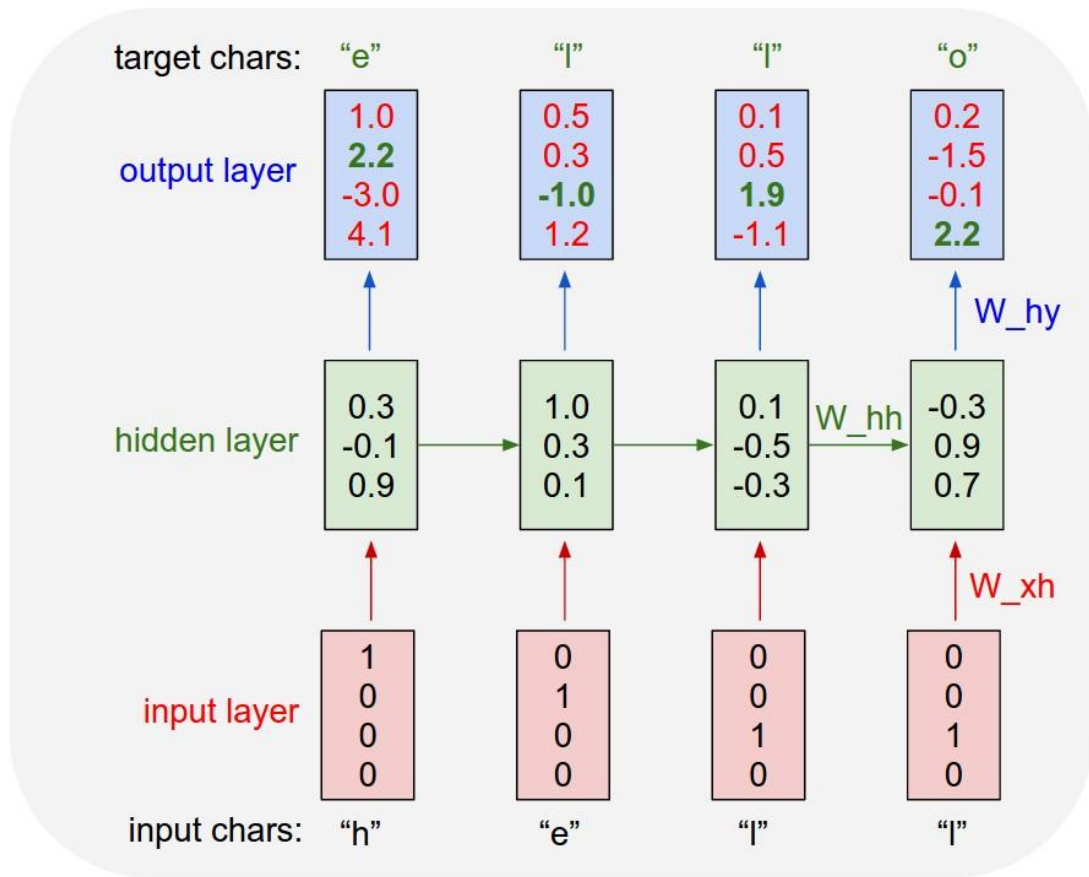




# Character-level language model example

Vocabulary:  
[h,e,l,o]

Example training sequence:  
“hello”



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

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

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

# min-char-rnn.py gist

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

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

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

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

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

```
30     n_h, n_y, n_x = (0, 0, 0)
31     h[0] = hprev
32     loss = 0
```

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

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

```
41     # Backward pass: compute gradients going backward
42     dwh, dwhh, dwhy = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
43     dh, dy = np.zeros_like(h), np.zeros_like(y)
44     dhnext = np.zeros_like(h[0])
```

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

```
48         dhy = np.dot(dy, h[t].T)
49         dhy += dy
50         dh = np.dot(dhy.T, dy) + dhnext # backprop into h
```

```
51         dhrw = (1 - h[t]**2) * h[t] * dh # dh a backprop through tanh nonlinearity
52         dhw = dhrw
53         dhh = np.dot(dhrw, h[t].T)
54         dhwh = np.dot(dhrw, h[t].T)
```

```
55         dhnext = np.dot(dh.T, dwh)
56         for dparam in [dwh, dhh, dwhy, dby, dby]:
57             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
```

```
58     return loss, dwh, dhh, dwhy, dby, h[inputs[-1]]
59
60 def sample(h, ix):
61     """
62     sample a sequence of integers from the model
63     h is memory state, seed, ix is seed letter for first time step
64     """
```

```
65     x = np.zeros(vocab_size, 1)
66     x[ix] = 1
67     hses = [h]
```

```
68     for t in xrange(1):
69         h = np.tanh(np.dot(wh, x) + np.dot(whh, h) + bh)
70         y = np.dot(why, h) + by
71         p = np.exp(y) / np.sum(np.exp(y))
```

```
72         ix = np.random.choice(vocab_size, 1, p=p.ravel())
73         x = np.zeros(vocab_size, 1)
74         x[ix] = 1
75         hses.append(h)
```

```
76     return hses
77
78 n, p = 0, 0
79 m_h, m_why, m_by = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
80 m_hh, m_hy = np.zeros_like(hh), np.zeros_like(hy) # memory variables for AdamW
```

```
81 smooth_loss = np.log(1.0/vocab_size)/seq_length # loss at iteration 0
82 while True:
83     # generate inputs (we're sampling from left to right in steps seq_length long)
84     if p==seq_length:
85         p = 0
86         hprev = np.zeros(hidden_size, 1) # reset mem memory
```

```
87     p = p + 1 # go from start of data
88     inputs = [char_ix[ix] for ix in data[p:p+seq_length]]
89     targets = [char_ix[ix+1] for ix in data[p:p+seq_length-1]]
90
91     # sample from the model now and then
92     if n % 100 == 0:
93         sample_ix = sample(hprev, inputs[0], 200)
```

```
94         txt = ''.join(char[ix] for ix in sample_ix)
95         print '-----%d %s %s-----' % (n, txt, )
96
97     # Forward seq length characters through the net and fetch gradients
98     loss, dwh, dhh, dwhy, dh, dby, hprev = lossfun(inputs, targets, hprev)
```

```
99     smooth_loss = smooth_loss + (1.0/seq_length) * loss * 0.001
100     if n % 100 == 0: print 'iter %d, loss %f' % (n, smooth_loss) # print progress
```

```
107         m = dparam + dparam * dparam / np.sqrt((1 + dparam) * (1 + dparam))
108         dparam = -learning_rate * dparam / np.sqrt((1 + dparam) * (1 + dparam)) # AdamW update
109
110     p = seq_length # move data pointer
111     n = n + 1 # iteration counter
```

## Data I/O

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

[illegible]

```

1 # Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
2 # NB: this code is meant to be a simple baseline, not a state-of-the-art model.
3
4 import numpy as np
5
6 # data loader
7 data = open('input.txt', 'r').read() # should be single plain text file
8 chars = list(data)
9 vocab_size = len(set(chars))
10 print 'chars has %d characters' % vocab_size # 26 + 1 (data size, vocab size)
11 char_idx = {c: i for i, c in enumerate(chars)}
12 ix, ix_hiden = {i: j for i, j, c, h in enumerate(chars)}
13
14 # hyperparameters
15 hidden_size = 100 # size of hidden layer of neurons
16 seq_length = 25 # number of steps to unroll the RNN for
17 learning_rate = 1e-3
18
19 # model parameters
20 w_ix_hiden, w_hiden_hiden, w_vocab_hiden, w_hiden_o = np.random.randn(
21     vocab_size, hidden_size, hidden_size, vocab_size) * 0.1 # input to hidden
22 w_hh = np.random.randn(hidden_size, hidden_size) * 0.1 # hidden to hidden
23 w_yo = np.random.randn(vocab_size, hidden_size) * 0.2 # hidden to output
24 w_o_h = np.zeros((hidden_size, vocab_size)) # biomed bias
25 w_ix_hiden, w_hiden_hiden, w_vocab_hiden, w_hiden_o = output bias
26
27 def loss_and_grads(chars, hprev):
28     # inputs, targets are both lists of integers.
29     # hprev is w/ol array of initial hidden state
30     # returns the loss, gradients on model parameters, and latest hidden state
31
32     xs, hs, ys, ps = [], [], [], []
33     h = hprev
34     for i in range(seq_length):
35         # forward pass
36         xi = ix[chars[i]]
37         xi_h = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
38         xi_h[xi] = 1
39         hi = np.tanh(np.dot(w_ix_hiden, xi_h) + np.dot(w_hh, h[-1] - h)) # hidden to hidden
40         hi_o = np.dot(w_vocab_hiden, hi) # by normalizing log probabilities for next char
41         hi_o = np.exp(hi_o) / np.sum(np.exp(hi_o)) # probabilities for next char
42         loss = -np.log(hi_o[ys[i]]) # softmax (cross-entropy loss)
43         # backward pass: compute gradients going backwards
44         dhi_o = np.zeros_like(hi_o)
45         dhi_o[ys[i]] = 1 # d(loss)/d(logprob[ys[i]])
46         dhi = np.dot(w_o_h, dhi_o)
47         dhs = np.zeros_like(hs)
48         dhs[i] = -np.dot(w_hh, dhi)
49         dxi_h = np.zeros_like(xi_h)
50         dxi_h[xi] = 1
51         dxi = np.dot(w_ix_hiden, dxi_h)
52         dxi_o = np.dot(w_vocab_hiden, dxi)
53         dxi_o[ys[i]] = 1 # d(loss)/d(xi_o)
54         dxi_o = np.dot(w_hh, dxi_o)
55         dxi_o = np.dot(w_ix_hiden, dxi_o)
56         dxi_o = np.dot(w_hh, dxi_o)
57         dxi_o = np.dot(w_ix_hiden, dxi_o)
58         dxi_o = np.dot(w_hh, dxi_o)
59         dxi_o = np.dot(w_ix_hiden, dxi_o)
60         dxi_o = np.dot(w_hh, dxi_o)
61         dxi_o = np.dot(w_ix_hiden, dxi_o)
62         dxi_o = np.dot(w_hh, dxi_o)
63         dxi_o = np.dot(w_ix_hiden, dxi_o)
64         dxi_o = np.dot(w_hh, dxi_o)
65         dxi_o = np.dot(w_ix_hiden, dxi_o)
66         dxi_o = np.dot(w_hh, dxi_o)
67         dxi_o = np.dot(w_ix_hiden, dxi_o)
68         dxi_o = np.dot(w_hh, dxi_o)
69         dxi_o = np.dot(w_ix_hiden, dxi_o)
70         dxi_o = np.dot(w_hh, dxi_o)
71         dxi_o = np.dot(w_ix_hiden, dxi_o)
72         dxi_o = np.dot(w_hh, dxi_o)
73         dxi_o = np.dot(w_ix_hiden, dxi_o)
74         dxi_o = np.dot(w_hh, dxi_o)
75         dxi_o = np.dot(w_ix_hiden, dxi_o)
76         dxi_o = np.dot(w_hh, dxi_o)
77         dxi_o = np.dot(w_ix_hiden, dxi_o)
78         dxi_o = np.dot(w_hh, dxi_o)
79         dxi_o = np.dot(w_ix_hiden, dxi_o)
80         dxi_o = np.dot(w_hh, dxi_o)
81         dxi_o = np.dot(w_ix_hiden, dxi_o)
82         dxi_o = np.dot(w_hh, dxi_o)
83         dxi_o = np.dot(w_ix_hiden, dxi_o)
84         dxi_o = np.dot(w_hh, dxi_o)
85         dxi_o = np.dot(w_ix_hiden, dxi_o)
86         dxi_o = np.dot(w_hh, dxi_o)
87         dxi_o = np.dot(w_ix_hiden, dxi_o)
88         dxi_o = np.dot(w_hh, dxi_o)
89         dxi_o = np.dot(w_ix_hiden, dxi_o)
90         dxi_o = np.dot(w_hh, dxi_o)
91         dxi_o = np.dot(w_ix_hiden, dxi_o)
92         dxi_o = np.dot(w_hh, dxi_o)
93         dxi_o = np.dot(w_ix_hiden, dxi_o)
94         dxi_o = np.dot(w_hh, dxi_o)
95         dxi_o = np.dot(w_ix_hiden, dxi_o)
96         dxi_o = np.dot(w_hh, dxi_o)
97         dxi_o = np.dot(w_ix_hiden, dxi_o)
98         dxi_o = np.dot(w_hh, dxi_o)
99         dxi_o = np.dot(w_ix_hiden, dxi_o)
100         dxi_o = np.dot(w_hh, dxi_o)
101         dxi_o = np.dot(w_ix_hiden, dxi_o)
102         dxi_o = np.dot(w_hh, dxi_o)
103         dxi_o = np.dot(w_ix_hiden, dxi_o)
104         dxi_o = np.dot(w_hh, dxi_o)
105         dxi_o = np.dot(w_ix_hiden, dxi_o)
106         dxi_o = np.dot(w_hh, dxi_o)
107         dxi_o = np.dot(w_ix_hiden, dxi_o)
108         dxi_o = np.dot(w_hh, dxi_o)
109         dxi_o = np.dot(w_ix_hiden, dxi_o)
110         dxi_o = np.dot(w_hh, dxi_o)
111         dxi_o = np.dot(w_ix_hiden, dxi_o)
112         dxi_o = np.dot(w_hh, dxi_o)
113         dxi_o = np.dot(w_ix_hiden, dxi_o)
114         dxi_o = np.dot(w_hh, dxi_o)
115         dxi_o = np.dot(w_ix_hiden, dxi_o)
116         dxi_o = np.dot(w_hh, dxi_o)
117         dxi_o = np.dot(w_ix_hiden, dxi_o)
118         dxi_o = np.dot(w_hh, dxi_o)
119         dxi_o = np.dot(w_ix_hiden, dxi_o)
120         dxi_o = np.dot(w_hh, dxi_o)
121         dxi_o = np.dot(w_ix_hiden, dxi_o)
122         dxi_o = np.dot(w_hh, dxi_o)
123         dxi_o = np.dot(w_ix_hiden, dxi_o)
124         dxi_o = np.dot(w_hh, dxi_o)
125         dxi_o = np.dot(w_ix_hiden, dxi_o)
126         dxi_o = np.dot(w_hh, dxi_o)
127         dxi_o = np.dot(w_ix_hiden, dxi_o)
128         dxi_o = np.dot(w_hh, dxi_o)
129         dxi_o = np.dot(w_ix_hiden, dxi_o)
130         dxi_o = np.dot(w_hh, dxi_o)
131         dxi_o = np.dot(w_ix_hiden, dxi_o)
132         dxi_o = np.dot(w_hh, dxi_o)
133         dxi_o = np.dot(w_ix_hiden, dxi_o)
134         dxi_o = np.dot(w_hh, dxi_o)
135         dxi_o = np.dot(w_ix_hiden, dxi_o)
136         dxi_o = np.dot(w_hh, dxi_o)
137         dxi_o = np.dot(w_ix_hiden, dxi_o)
138         dxi_o = np.dot(w_hh, dxi_o)
139         dxi_o = np.dot(w_ix_hiden, dxi_o)
140         dxi_o = np.dot(w_hh, dxi_o)
141         dxi_o = np.dot(w_ix_hiden, dxi_o)
142         dxi_o = np.dot(w_hh, dxi_o)
143         dxi_o = np.dot(w_ix_hiden, dxi_o)
144         dxi_o = np.dot(w_hh, dxi_o)
145         dxi_o = np.dot(w_ix_hiden, dxi_o)
146         dxi_o = np.dot(w_hh, dxi_o)
147         dxi_o = np.dot(w_ix_hiden, dxi_o)
148         dxi_o = np.dot(w_hh, dxi_o)
149         dxi_o = np.dot(w_ix_hiden, dxi_o)
150         dxi_o = np.dot(w_hh, dxi_o)
151         dxi_o = np.dot(w_ix_hiden, dxi_o)
152         dxi_o = np.dot(w_hh, dxi_o)
153         dxi_o = np.dot(w_ix_hiden, dxi_o)
154         dxi_o = np.dot(w_hh, dxi_o)
155         dxi_o = np.dot(w_ix_hiden, dxi_o)
156         dxi_o = np.dot(w_hh, dxi_o)
157         dxi_o = np.dot(w_ix_hiden, dxi_o)
158         dxi_o = np.dot(w_hh, dxi_o)
159         dxi_o = np.dot(w_ix_hiden, dxi_o)
160         dxi_o = np.dot(w_hh, dxi_o)
161         dxi_o = np.dot(w_ix_hiden, dxi_o)
162         dxi_o = np.dot(w_hh, dxi_o)
163         dxi_o = np.dot(w_ix_hiden, dxi_o)
164         dxi_o = np.dot(w_hh, dxi_o)
165         dxi_o = np.dot(w_ix_hiden, dxi_o)
166         dxi_o = np.dot(w_hh, dxi_o)
167         dxi_o = np.dot(w_ix_hiden, dxi_o)
168         dxi_o = np.dot(w_hh, dxi_o)
169         dxi_o = np.dot(w_ix_hiden, dxi_o)
170         dxi_o = np.dot(w_hh, dxi_o)
171         dxi_o = np.dot(w_ix_hiden, dxi_o)
172         dxi_o = np.dot(w_hh, dxi_o)
173         dxi_o = np.dot(w_ix_hiden, dxi_o)
174         dxi_o = np.dot(w_hh, dxi_o)
175         dxi_o = np.dot(w_ix_hiden, dxi_o)
176         dxi_o = np.dot(w_hh, dxi_o)
177         dxi_o = np.dot(w_ix_hiden, dxi_o)
178         dxi_o = np.dot(w_hh, dxi_o)
179         dxi_o = np.dot(w_ix_hiden, dxi_o)
180         dxi_o = np.dot(w_hh, dxi_o)
181         dxi_o = np.dot(w_ix_hiden, dxi_o)
182         dxi_o = np.dot(w_hh, dxi_o)
183         dxi_o = np.dot(w_ix_hiden, dxi_o)
184         dxi_o = np.dot(w_hh, dxi_o)
185         dxi_o = np.dot(w_ix_hiden, dxi_o)
186         dxi_o = np.dot(w_hh, dxi_o)
187         dxi_o = np.dot(w_ix_hiden, dxi_o)
188         dxi_o = np.dot(w_hh, dxi_o)
189         dxi_o = np.dot(w_ix_hiden, dxi_o)
190         dxi_o = np.dot(w_hh, dxi_o)
191         dxi_o = np.dot(w_ix_hiden, dxi_o)
192         dxi_o = np.dot(w_hh, dxi_o)
193         dxi_o = np.dot(w_ix_hiden, dxi_o)
194         dxi_o = np.dot(w_hh, dxi_o)
195         dxi_o = np.dot(w_ix_hiden, dxi_o)
196         dxi_o = np.dot(w_hh, dxi_o)
197         dxi_o = np.dot(w_ix_hiden, dxi_o)
198         dxi_o = np.dot(w_hh, dxi_o)
199         dxi_o = np.dot(w_ix_hiden, dxi_o)
200         dxi_o = np.dot(w_hh, dxi_o)
201         dxi_o = np.dot(w_ix_hiden, dxi_o)
202         dxi_o = np.dot(w_hh, dxi_o)
203         dxi_o = np.dot(w_ix_hiden, dxi_o)
204         dxi_o = np.dot(w_hh, dxi_o)
205         dxi_o = np.dot(w_ix_hiden, dxi_o)
206         dxi_o = np.dot(w_hh, dxi_o)
207         dxi_o = np.dot(w_ix_hiden, dxi_o)
208         dxi_o = np.dot(w_hh, dxi_o)
209         dxi_o = np.dot(w_ix_hiden, dxi_o)
210         dxi_o = np.dot(w_hh, dxi_o)
211         dxi_o = np.dot(w_ix_hiden, dxi_o)
212         dxi_o = np.dot(w_hh, dxi_o)
213         dxi_o = np.dot(w_ix_hiden, dxi_o)
214         dxi_o = np.dot(w_hh, dxi_o)
215         dxi_o = np.dot(w_ix_hiden, dxi_o)
216         dxi_o = np.dot(w_hh, dxi_o)
217         dxi_o = np.dot(w_ix_hiden, dxi_o)
218         dxi_o = np.dot(w_hh, dxi_o)
219         dxi_o = np.dot(w_ix_hiden, dxi_o)
220         dxi_o = np.dot(w_hh, dxi_o)
221         dxi_o = np.dot(w_ix_hiden, dxi_o)
222         dxi_o = np.dot(w_hh, dxi_o)
223        
```

```
# size of hidden layer
number of steps to u
e-1

s
andn(hidden_size, voc
andn(hidden_size, hid
andn(vocab_size, hid
hidden_size, 1)) # hid
cab_size, 1)) # output
```

recall:

target chars: "e" "n" "n" "o"

output layer

hidden layer

input layer

$W_{xh}$

$W_{hh}$

$W_{hy}$

# min-char-rnn.py gist

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

## Main loop

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



# min-char-rnn.py gist

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

## Main loop

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



# min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(data)
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique' % (data_size, vocab_size)
12 char_to_ix = { ch: i for i, ch in enumerate(chars) }
13 ix_to_char = { i: ch for i, ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 b1 = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is vec array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     es, hs, ys, ps = [], [], [], []
34     h1 = hprev
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi = np.zeros((vocab_size, 1)) # encode in 1-of-K representation
39         xi[ix_to_char[t]] = 1
40         h1 = np.tanh(np.dot(wih, xi) + np.dot(whh, h1[-1]) + bh) # hidden state
41         yi = np.dot(why, h1) + by # unnormalized log probabilities for next chars
42         pi = np.exp(yi) / np.sum(np.exp(yi)) # probabilities for next chars
43         loss += -np.log(pi[target[t], 0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backward
45     dwh, dwhh, dwhy = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
46     dbh, dbh1, dbh2, dbh3, dbh4, dbh5, dbh6, dbh7, dbh8, dbh9, dbh10, dbh11, dbh12, dbh13, dbh14, dbh15, dbh16, dbh17, dbh18, dbh19, dbh20, dbh21, dbh22, dbh23, dbh24, dbh25, dbh26, dbh27, dbh28, dbh29, dbh30, dbh31, dbh32, dbh33, dbh34, dbh35, dbh36, dbh37, dbh38, dbh39, dbh40, dbh41, dbh42, dbh43, dbh44, dbh45, dbh46, dbh47, dbh48, dbh49, dbh50, dbh51, dbh52, dbh53, dbh54, dbh55, dbh56, dbh57, dbh58, dbh59, dbh60, dbh61, dbh62, dbh63, dbh64, dbh65, dbh66, dbh67, dbh68, dbh69, dbh70, dbh71, dbh72, dbh73, dbh74, dbh75, dbh76, dbh77, dbh78, dbh79, dbh80, dbh81, dbh82, dbh83, dbh84, dbh85, dbh86, dbh87, dbh88, dbh89, dbh90, dbh91, dbh92, dbh93, dbh94, dbh95, dbh96, dbh97, dbh98, dbh99, dbh100, dbh101, dbh102, dbh103, dbh104, dbh105, dbh106, dbh107, dbh108, dbh109, dbh110, dbh111, dbh112, dbh113, dbh114, dbh115, dbh116, dbh117, dbh118, dbh119, dbh120, dbh121, dbh122, dbh123, dbh124, dbh125, dbh126, dbh127, dbh128, dbh129, dbh130, dbh131, dbh132, dbh133, dbh134, dbh135, dbh136, dbh137, dbh138, dbh139, dbh140, dbh141, dbh142, dbh143, dbh144, dbh145, dbh146, dbh147, dbh148, dbh149, dbh150, dbh151, dbh152, dbh153, dbh154, dbh155, dbh156, dbh157, dbh158, dbh159, dbh160, dbh161, dbh162, dbh163, dbh164, dbh165, dbh166, dbh167, dbh168, dbh169, dbh170, dbh171, dbh172, dbh173, dbh174, dbh175, dbh176, dbh177, dbh178, dbh179, dbh180, dbh181, dbh182, dbh183, dbh184, dbh185, dbh186, dbh187, dbh188, dbh189, dbh190, dbh191, dbh192, dbh193, dbh194, dbh195, dbh196, dbh197, dbh198, dbh199, dbh200, dbh201, dbh202, dbh203, dbh204, dbh205, dbh206, dbh207, dbh208, dbh209, dbh210, dbh211, dbh212, dbh213, dbh214, dbh215, dbh216, dbh217, dbh218, dbh219, dbh220, dbh221, dbh222, dbh223, dbh224, dbh225, dbh226, dbh227, dbh228, dbh229, dbh230, dbh231, dbh232, dbh233, dbh234, dbh235, dbh236, dbh237, dbh238, dbh239, dbh240, dbh241, dbh242, dbh243, dbh244, dbh245, dbh246, dbh247, dbh248, dbh249, dbh250, dbh251, dbh252, dbh253, dbh254, dbh255, dbh256, dbh257, dbh258, dbh259, dbh260, dbh261, dbh262, dbh263, dbh264, dbh265, dbh266, dbh267, dbh268, dbh269, dbh270, dbh271, dbh272, dbh273, dbh274, dbh275, dbh276, dbh277, dbh278, dbh279, dbh280, dbh281, dbh282, dbh283, dbh284, dbh285, dbh286, dbh287, dbh288, dbh289, dbh290, dbh291, dbh292, dbh293, dbh294, dbh295, dbh296, dbh297, dbh298, dbh299, dbh300, dbh301, dbh302, dbh303, dbh304, dbh305, dbh306, dbh307, dbh308, dbh309, dbh310, dbh311, dbh312, dbh313, dbh314, dbh315, dbh316, dbh317, dbh318, dbh319, dbh320, dbh321, dbh322, dbh323, dbh324, dbh325, dbh326, dbh327, dbh328, dbh329, dbh330, dbh331, dbh332, dbh333, dbh334, dbh335, dbh336, dbh337, dbh338, dbh339, dbh340, dbh341, dbh342, dbh343, dbh344, dbh345, dbh346, dbh347, dbh348, dbh349, dbh350, dbh351, dbh352, dbh353, dbh354, dbh355, dbh356, dbh357, dbh358, dbh359, dbh360, dbh361, dbh362, dbh363, dbh364, dbh365, dbh366, dbh367, dbh368, dbh369, dbh370, dbh371, dbh372, dbh373, dbh374, dbh375, dbh376, dbh377, dbh378, dbh379, dbh380, dbh381, dbh382, dbh383, dbh384, dbh385, dbh386, dbh387, dbh388, dbh389, dbh390, dbh391, dbh392, dbh393, dbh394, dbh395, dbh396, dbh397, dbh398, dbh399, dbh400, dbh401, dbh402, dbh403, dbh404, dbh405, dbh406, dbh407, dbh408, dbh409, dbh410, dbh411, dbh412, dbh413, dbh414, dbh415, dbh416, dbh417, dbh418, dbh419, dbh420, dbh421, dbh422, dbh423, dbh424, dbh425, dbh426, dbh427, dbh428, dbh429, dbh430, dbh431, dbh432, dbh433, dbh434, dbh435, dbh436, dbh437, dbh438, dbh439, dbh440, dbh441, dbh442, dbh443, dbh444, dbh445, dbh446, dbh447, dbh448, dbh449, dbh450, dbh451, dbh452, dbh453, dbh454, dbh455, dbh456, dbh457, dbh458, dbh459, dbh460, dbh461, dbh462, dbh463, dbh464, dbh465, dbh466, dbh467, dbh468, dbh469, dbh470, dbh471, dbh472, dbh473, dbh474, dbh475, dbh476, dbh477, dbh478, dbh479, dbh480, dbh481, dbh482, dbh483, dbh484, dbh485, dbh486, dbh487, dbh488, dbh489, dbh490, dbh491, dbh492, dbh493, dbh494, dbh495, dbh496, dbh497, dbh498, dbh499, dbh500, dbh501, dbh502, dbh503, dbh504, dbh505, dbh506, dbh507, dbh508, dbh509, dbh510, dbh511, dbh512, dbh513, dbh514, dbh515, dbh516, dbh517, dbh518, dbh519, dbh520, dbh521, dbh522, dbh523, dbh524, dbh525, dbh526, dbh527, dbh528, dbh529, dbh530, dbh531, dbh532, dbh533, dbh534, dbh535, dbh536, dbh537, dbh538, dbh539, dbh540, dbh541, dbh542, dbh543, dbh544, dbh545, dbh546, dbh547, dbh548, dbh549, dbh550, dbh551, dbh552, dbh553, dbh554, dbh555, dbh556, dbh557, dbh558, dbh559, dbh560, dbh561, dbh562, dbh563, dbh564, dbh565, dbh566, dbh567, dbh568, dbh569, dbh570, dbh571, dbh572, dbh573, dbh574, dbh575, dbh576, dbh577, dbh578, dbh579, dbh580, dbh581, dbh582, dbh583, dbh584, dbh585, dbh586, dbh587, dbh588, dbh589, dbh590, dbh591, dbh592, dbh593, dbh594, dbh595, dbh596, dbh597, dbh598, dbh599, dbh600, dbh601, dbh602, dbh603, dbh604, dbh605, dbh606, dbh607, dbh608, dbh609, dbh610, dbh611, dbh612, dbh613, dbh614, dbh615, dbh616, dbh617, dbh618, dbh619, dbh620, dbh621, dbh622, dbh623, dbh624, dbh625, dbh626, dbh627, dbh628, dbh629, dbh630, dbh631, dbh632, dbh633, dbh634, dbh635, dbh636, dbh637, dbh638, dbh639, dbh640, dbh641, dbh642, dbh643, dbh644, dbh645, dbh646, dbh647, dbh648, dbh649, dbh650, dbh651, dbh652, dbh653, dbh654, dbh655, dbh656, dbh657, dbh658, dbh659, dbh660, dbh661, dbh662, dbh663, dbh664, dbh665, dbh666, dbh667, dbh668, dbh669, dbh670, dbh671, dbh672, dbh673, dbh674, dbh675, dbh676, dbh677, dbh678, dbh679, dbh680, dbh681, dbh682, dbh683, dbh684, dbh685, dbh686, dbh687, dbh688, dbh689, dbh690, dbh691, dbh692, dbh693, dbh694, dbh695, dbh696, dbh697, dbh698, dbh699, dbh700, dbh701, dbh702, dbh703, dbh704, dbh705, dbh706, dbh707, dbh708, dbh709, dbh710, dbh711, dbh712, dbh713, dbh714, dbh715, dbh716, dbh717, dbh718, dbh719, dbh720, dbh721, dbh722, dbh723, dbh724, dbh725, dbh726, dbh727, dbh728, dbh729, dbh730, dbh731, dbh732, dbh733, dbh734, dbh735, dbh736, dbh737, dbh738, dbh739, dbh740, dbh741, dbh742, dbh743, dbh744, dbh745, dbh746, dbh747, dbh748, dbh749, dbh750, dbh751, dbh752, dbh753, dbh754, dbh755, dbh756, dbh757, dbh758, dbh759, dbh760, dbh761, dbh762, dbh763, dbh764, dbh765, dbh766, dbh767, dbh768, dbh769, dbh770, dbh771, dbh772, dbh773, dbh774, dbh775, dbh776, dbh777, dbh778, dbh779, dbh780, dbh781, dbh782, dbh783, dbh784, dbh785, dbh786, dbh787, dbh788, dbh789, dbh790, dbh791, dbh792, dbh793, dbh794, dbh795, dbh796, dbh797, dbh798, dbh799, dbh800, dbh801, dbh802, dbh803, dbh804, dbh805, dbh806, dbh807, dbh808, dbh809, dbh810, dbh811, dbh812, dbh813, dbh814, dbh815, dbh816, dbh817, dbh818, dbh819, dbh820, dbh821, dbh822, dbh823, dbh824, dbh825, dbh826, dbh827, dbh828, dbh829, dbh830, dbh831, dbh832, dbh833, dbh834, dbh835, dbh836, dbh837, dbh838, dbh839, dbh840, dbh841, dbh842, dbh843, dbh844, dbh845, dbh846, dbh847, dbh848, dbh849, dbh850, dbh851, dbh852, dbh853, dbh854, dbh855, dbh856, dbh857, dbh858, dbh859, dbh860, dbh861, dbh862, dbh863, dbh864, dbh865, dbh866, dbh867, dbh868, dbh869, dbh870, dbh871, dbh872, dbh873, dbh874, dbh875, dbh876, dbh877, dbh878, dbh879, dbh880, dbh881, dbh882, dbh883, dbh884, dbh885, dbh886, dbh887, dbh888, dbh889, dbh890, dbh891, dbh892, dbh893, dbh894, dbh895, dbh896, dbh897, dbh898, dbh899, dbh900, dbh901, dbh902, dbh903, dbh904, dbh905, dbh906, dbh907, dbh908, dbh909, dbh910, dbh911, dbh912, dbh913, dbh914, dbh915, dbh916, dbh917, dbh918, dbh919, dbh920, dbh921, dbh922, dbh923, dbh924, dbh925, dbh926, dbh927, dbh928, dbh929, dbh930, dbh931, dbh932, dbh933, dbh934, dbh935, dbh936, dbh937, dbh938, dbh939, dbh940, dbh941, dbh942, dbh943, dbh944, dbh945, dbh946, dbh947, dbh948, dbh949, dbh950, dbh951, dbh952, dbh953, dbh954, dbh955, dbh956, dbh957, dbh958, dbh959, dbh960, dbh961, dbh962, dbh963, dbh964, dbh965, dbh966, dbh967, dbh968, dbh969, dbh970, dbh971, dbh972, dbh973, dbh974, dbh975, dbh976, dbh977, dbh978, dbh979, dbh980, dbh981, dbh982, dbh983, dbh984, dbh985, dbh986, dbh987, dbh988, dbh989, dbh990, dbh991, dbh992, dbh993, dbh994, dbh995, dbh996, dbh997, dbh998, dbh999, dbh1000, dbh1001, dbh1002, dbh1003, dbh1004, dbh1005, dbh1006, dbh1007, dbh1008, dbh1009, dbh1010, dbh1011, dbh1012, dbh1013, dbh1014, dbh1015, dbh1016, dbh1017, dbh1018, dbh1019, dbh1020, dbh1021, dbh1022, dbh1023, dbh1024, dbh1025, dbh1026, dbh1027, dbh1028, dbh1029, dbh1030, dbh1031, dbh1032, dbh1033, dbh1034, dbh1035, dbh1036, dbh1037, dbh1038, dbh1039, dbh1040, dbh1041, dbh1042, dbh1043, dbh1044, dbh1045, dbh1046, dbh1047, dbh1048, dbh1049, dbh1050, dbh1051, dbh1052, dbh1053, dbh1054, dbh1055, dbh1056, dbh1057, dbh1058, dbh1059, dbh1060, dbh1061, dbh1062, dbh1063, dbh1064, dbh1065, dbh1066, dbh1067, dbh1068, dbh1069, dbh1070, dbh1071, dbh1072, dbh1073, dbh1074, dbh1075, dbh1076, dbh1077, dbh1078, dbh1079, dbh1080, dbh1081, dbh1082, dbh1083, dbh1084, dbh1085, dbh1086, dbh1087, dbh1088, dbh1089, dbh1090, dbh1091, dbh1092, dbh1093, dbh1094, dbh1095, dbh1096, dbh1097, dbh1098, dbh1099, dbh1100, dbh1101, dbh1102, dbh1103, dbh1104, dbh1105, dbh1106, dbh1107, dbh1108, dbh1109, dbh1110, dbh1111, dbh1112, dbh1113, dbh1114, dbh1115, dbh1116, dbh1117, dbh1118, dbh1119, dbh1120, dbh1121, dbh1122, dbh1123, dbh1124, dbh1125, dbh1126, dbh1127, dbh1128, dbh1129, dbh1130, dbh1131, dbh1132, dbh1133, dbh1134, dbh1135, dbh1136, dbh1137, dbh1138, dbh1139, dbh1140, dbh1141, dbh1142, dbh1143, dbh1144, dbh1145, dbh1146, dbh1147, dbh1148, dbh1149, dbh1150, dbh1151, dbh1152, dbh1153, dbh1154, dbh1155, dbh1156, dbh1157, dbh1158, dbh1159, dbh1160, dbh1161, dbh1162, dbh1163, dbh1164, dbh1165, dbh1166, dbh1167, dbh1168, dbh1169, dbh1170, dbh1171, dbh1172, dbh1173, dbh1174, dbh1175, dbh1176, dbh1177, dbh1178, dbh1179, dbh1180, dbh1181, dbh1182, dbh1183, dbh1184, dbh1185, dbh1186, dbh1187, dbh1188, dbh1189, dbh1190, dbh1191, dbh1192, dbh1193, dbh1194, dbh1195, dbh1196, dbh1197, dbh1198, dbh1199, dbh1200, dbh1201, dbh1202, dbh1203, dbh1204, dbh1205, dbh1206, dbh1207, dbh1208, dbh1209, dbh1210, dbh1211, dbh1212, dbh1213, dbh1214, dbh1215, dbh1216, dbh1217, dbh1218, dbh1219, dbh1220, dbh1221, dbh1222, dbh1223, dbh1224, dbh1225, dbh1226, dbh1227, dbh1228, dbh1229, dbh1230, dbh1231, dbh1232, dbh1233, dbh1234, dbh1235, dbh1236, dbh1237, dbh1238, dbh1239, dbh1240, dbh1241, dbh1242, dbh1243, dbh1244, dbh1245, dbh1246, dbh1247, dbh1248, dbh1249, dbh1250, dbh1251, dbh1252, dbh1253, dbh1254, dbh1255, dbh1256, dbh1257, dbh1258, dbh1259, dbh1260, dbh1261, dbh1262, dbh1263, dbh1264, dbh1265, dbh1266, dbh1267, dbh1268, dbh1269, dbh1270, dbh1271, dbh1272, dbh1273, dbh1274, dbh1275, dbh1276, dbh1277, dbh1278, dbh1279, dbh1280, dbh1281, dbh1282, dbh1283, dbh1284, dbh1285, dbh1286, dbh1287, dbh1288, dbh1289, dbh1290, dbh1291, dbh1292, dbh1293, dbh1294, dbh1295, dbh1296, dbh1297, dbh1298, dbh1299, dbh1300, dbh1301, dbh1302, dbh1303, dbh1304, dbh1305, dbh1306, dbh1307, dbh1308, dbh1309, dbh1310, dbh1311, dbh1312, dbh1313, dbh1314, dbh1315, dbh1316, dbh1317, dbh1318, dbh1319, dbh1320, dbh1321, dbh1322, dbh1323, dbh1324, dbh1325, dbh1326, dbh1327, dbh1328, dbh1329, dbh1330, dbh1331, dbh1332, dbh1333, dbh1334, dbh1335, dbh1336, dbh1337, dbh1338, dbh1339, dbh1340, dbh1341, dbh1342, dbh1343, dbh1344, dbh1345, dbh1346, dbh1347, dbh1348, dbh1349, dbh1350, dbh1351, dbh1352, dbh1353, dbh1354, dbh1355, dbh1356, dbh1357, dbh1358, dbh1359, dbh1360, dbh1361, dbh1362, dbh1363, dbh1364, dbh1365, dbh1366, dbh1367, dbh1368, dbh1369, dbh1370, dbh1371, dbh1372, dbh1373, dbh1374, dbh1375, dbh1376, dbh1377, dbh1378, dbh1379, dbh1380, dbh1381, dbh1382, dbh1383, dbh1384, dbh1385, dbh1386, dbh1387, dbh1388, dbh1389, dbh1390, dbh1391, dbh1392, dbh1393, dbh1394, dbh1395, dbh1396, dbh1397, dbh1398, dbh1399, dbh1400, dbh1401, dbh1402, dbh1403, dbh1404, dbh1405, dbh1406, dbh1407, dbh1408, dbh1409, dbh1410, dbh1411, dbh1412, dbh1413, dbh1414, dbh1415, dbh1416, dbh1417, dbh1418, dbh1419, dbh1420, dbh1421, dbh1422, dbh1423, dbh1424, dbh1425, dbh1426, dbh1427, dbh1428, dbh1429, dbh1430, dbh1431, dbh1432, dbh1433, dbh1434, dbh1435, dbh1436, dbh1437, dbh1438, dbh1439, dbh1440, dbh1441, dbh1442, dbh1443, dbh1444, dbh1445, dbh1446, dbh1447, dbh1448, dbh1449, dbh1450, dbh1451, dbh1452, dbh1453, dbh1454, dbh1455, dbh1456, dbh1457, dbh1458, dbh1459, dbh1460, dbh1461, dbh1462, dbh1463, dbh1464, dbh1465, dbh1466, dbh1467, dbh1468, dbh1469, dbh1470, dbh1471, dbh1472, dbh1473, dbh1474, dbh1475, dbh1476, dbh1477, dbh1478, dbh1479, dbh1480, dbh1481, dbh1482, dbh1483, dbh1484, dbh1485, dbh1486, dbh1487, dbh1488, dbh1489, dbh1490, dbh1491, dbh1492, dbh1493, dbh1494, dbh1495, dbh1496, dbh1497, dbh1498, dbh1499, dbh1500, dbh1501, dbh1502, dbh1503, dbh1504, dbh1505, dbh1506, dbh1507, dbh1508, dbh1509, dbh1510, dbh1511, dbh1512, dbh1513, dbh1514, dbh1515, dbh1516, dbh1517, dbh1518, dbh1519, dbh1520, dbh1521, dbh1522, dbh1523, dbh1524, dbh1525, dbh1526, dbh1527, dbh1528, dbh1529, dbh1530, dbh1531, dbh1532, dbh1533, dbh1534, dbh1535, dbh1536, dbh1537, dbh1538, dbh1539, dbh1540, dbh1541, dbh1542, dbh1543, dbh1544, dbh1545, dbh1546, dbh1547, dbh1548, dbh1549, dbh1550, dbh1551, dbh1552, dbh1553, dbh1554, dbh1555, dbh1556, dbh1557, dbh1558, dbh1559, dbh1560, dbh1561, dbh1562, dbh1563, dbh1564, dbh1565, dbh1566, dbh1567, dbh1568, dbh1569, dbh1570, dbh1571, dbh1572, dbh1573, dbh1574, dbh1575, dbh1576, dbh1577, dbh1578, dbh1579, dbh1580, dbh1581, dbh1582, dbh1583, dbh1584, dbh1585, dbh1586, dbh1587, dbh1588, dbh1589, dbh1590, dbh1591, dbh1592, dbh1593, dbh1594, dbh1595, dbh1596, dbh1597, dbh1598, dbh1599, dbh1600, dbh1601, dbh1602, dbh1603, dbh1604, dbh1605, dbh1606, dbh1607, dbh1608, dbh1609, dbh1610, dbh1611, dbh1612, dbh1613, dbh1614, dbh1615, dbh1616, dbh1617, dbh1618, dbh1619, dbh1620, dbh1621, dbh1622, dbh1623, dbh1624, dbh1625, dbh1626, dbh1627, dbh1628, dbh1629, dbh1630, dbh1631, dbh1632, dbh1633, dbh1634, dbh1635, dbh1636, dbh1637, dbh1638, dbh1639, dbh1640, dbh1641, dbh1642, dbh1643, dbh1644, dbh1645, dbh1646, dbh1647, dbh1648, dbh1649, dbh1650, dbh1651, dbh1652, dbh1653, dbh1654, dbh1655, dbh1656, dbh1657, dbh1658, dbh1659, dbh1660, dbh1661, dbh1662, dbh1663, dbh1664, dbh1665, dbh1666, dbh1667, dbh1668, dbh1669, dbh1670, dbh1671, dbh1672, dbh1673, dbh1674, dbh1675, dbh1676, dbh1677, dbh1678, dbh1679, dbh1680, dbh1681, dbh1682, dbh1683, dbh1684, dbh1685, dbh1686, dbh1687, dbh1688, dbh1689, dbh1690, dbh1691, dbh1692, dbh1693, dbh1694, dbh1695, dbh1696, dbh1697, dbh1698, dbh1699, dbh1700, dbh1701, dbh1702, dbh1703, dbh1704, dbh1705, dbh1706, dbh1707, dbh1708, dbh1709, dbh1710, dbh1711, dbh1712, dbh1713, dbh1714, dbh1715, dbh1716, dbh1717, dbh1718, dbh1719, dbh1720, dbh1721, dbh1722, dbh1723, dbh1724, dbh1725, dbh1726, dbh1727, dbh1728, dbh1729, dbh1730, dbh1731, dbh1732, dbh1733, dbh1734, dbh1735, dbh1736, dbh1737, dbh1738, dbh1739, dbh1740, dbh1741, dbh1742, dbh1743, dbh1744, dbh1745, dbh1746, dbh1747, dbh1748, dbh1749, dbh1750, dbh1751, dbh1752, dbh1753, dbh1754, dbh1755, dbh1756, dbh1757, dbh1758, dbh1759, dbh1760, dbh1761, dbh1762, dbh1763, dbh1764, dbh1765, dbh1766, dbh1767, dbh1768, dbh1769, dbh1770, dbh1771, dbh1772, dbh1773, dbh1774, dbh1775, dbh1776, dbh1777, dbh1778, dbh1779, dbh1780, dbh1781, dbh1782, dbh1783, dbh1784, dbh1785, dbh1786, dbh1787, dbh1788, dbh1789, dbh1790, dbh1791, dbh1792, dbh1793, dbh1794, dbh1795, dbh1796, dbh1797, dbh1798, dbh1799, dbh1800, dbh1801, dbh1802, dbh1803, dbh1804, dbh1805, dbh1806, dbh1807, dbh1808, dbh1809, dbh1810, dbh1811, dbh1812, dbh1813, dbh1814, dbh1815, dbh1816, dbh1817, dbh1818, dbh1819, dbh1820, dbh1821, dbh1822, dbh1823, dbh1824, dbh1825, dbh1826, dbh1827, db
```

# min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 b1 = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is list/array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     es, hs, ys, ps = [], [], [], []
34     h1 = hprev
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi = np.zeros((vocab_size, 1)) # encode in 1-of-K representation
39         xi[ix_to_char[t]] = 1
40         h1 = np.tanh(np.dot(wih, xi) + np.dot(whh, h1[-1]) + bh) # hidden state
41         yi = np.dot(why, h1) + by # unnormalized log probabilities for next chars
42         ps = np.exp(yi) / np.sum(np.exp(yi)) # probabilities for next chars
43         loss += -np.log(ps[t][xi[t]]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backward
45     dht, dbh, dby = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
46     dho = np.zeros_like(h1)
47     for t in reversed(xrange(len(inputs))):
48         dy = np.zeros((1,))
49         dy[targets[t]] -= 1 # backprop into y
50         dho = np.dot(dy, h1[t:T])
51         dy = dy
52         dh = np.dot(whh, dy) + dho # backprop into h
53         dhrw = (1 - h1[t]*h1[t]) * dh # backprop through tanh nonlinearity
54         dwh = np.dot(dhrw, xi[t:T])
55         dho = np.dot(dhrw, h1[t:T])
56         dhw = np.dot(dh, dhrw)
57         for dparam in [dwh, dwt, dby, dbh, dby]:
58             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
59     return loss, dwh, dwt, dby, dbh, dby, h1[len(inputs)-1]
60
61 def sample(h, ix):
62     """
63     sample a sequence of integers from the model
64     h is memory state, seed, ix is seed letter for first time step
65     """
66     x = np.zeros((vocab_size, 1))
67     ixes = []
68     for t in xrange(1):
69         xi = np.zeros((vocab_size, 1))
70         xi[ix] = 1
71         h = np.tanh(np.dot(wih, xi) + 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, rand=None)
75         x = np.zeros((vocab_size, 1))
76         x[ix] = 1
77         ixes.append(ix)
78     return ixes
79
80 # n, p = 0, 0
81 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
82 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
83 while True:
84     # prepare inputs (we're sweeping from left to right in steps seq_length long)
85     if p+seq_length+1 >= len(data) or n == 0:
86         hprev = np.zeros((hidden_size,1)) # reset RNN memory
87         p = 0 # go from start of data
88         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
89         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
90
91     # sample from the model now and then
92     if n % 100 == 0:
93         sample_ix = sample(hprev, inputs[0], 200)
94         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
95         print '----\n %s \n----' % (txt, )
96
97     # forward seq_length characters through the net and fetch gradient
98     loss, dwhx, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
99     smooth_loss = smooth_loss * 0.999 + loss * 0.001
100     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
101
102     # perform parameter update with Adagrad
103     for param, dparam, mem in zip([wih, whh, why, bh, by],
104                                   [dwhx, dwhh, dwhy, dbh, dby],
105                                   [mbh, mby]):
106         mem += dparam * dparam
107         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
108
109     p += seq_length # move data pointer
110     n += 1 # iteration counter
```

## Main loop

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

[illegible]

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

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



# min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data 370
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print "data has %d characters, %d unique." % (data_size, vocab_size)
12 char_to_ix = { char: i for i, ch in enumerate(chars) }
13 ix_to_char = { i: ch for i, ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
```

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

```
62 def sample(preds, ix):
63     """
64     sample a sequence of integers from the model
65     h is memory state, seed_ix is seed letter for first time step
66     """
67     x = np.zeros((vocab_size, 1))
68     ix_to_ix = {}
69     ix_to_ix[ix] = 1
70     for t in xrange(1):
71         x = np.zeros((vocab_size, 1))
72         x[ix] = 1
73         y = np.dot(whx, x) + np.dot(whh, hs[t-1]) + bh
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(vocab_size, p=p, size=1)
76         x[ix] = 1
77         ix_to_ix[ix] = 1
78     return ix
79
80 # run
81 n, p = 0, 0
82 mhs, mwh, mwhy = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
83 mhs, mwh, mwhy = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
84 mhs, mwh, mwhy = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
85 mhs, mwh, mwhy = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
86 while True:
87     # generate inputs (we're sampling from left to right in steps seq_length)
88     if p % seq_length == 0:
89         hprev = np.zeros((hidden_size, 1)) # reset mem memory
90         p = 0 # p is # of from count 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+seq_length: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(chars[ix] for ix in sample_ix)
98         print "====%s====" % txt,
99
100     # forward one length characters through the net and fetch gradients
101     loss, dws, dwhs, dwhys, dbhs, dby, hprev = lossFun(inputs, targets, hprev)
102     smooth_loss = smooth_loss + loss * 0.999
103     p = p + 1 # p is # of from count of data
104     print "====%s====" % txt,
105
106 # perform parameter update with Adagrad
107 for param, dparam in zip([dw, dwh, dwhy, dbh, dby], [dws, dwhs, dwhys, dbhs, dbys]):
108     param += -learning_rate * dparam / np.sqrt((1 + dparam) * (1 + dparam)) # Adagrad update
109
110 p = seq_length # move data pointer
111 n += 1 # iteration counter
```

## Loss function

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

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

# min-char-rnn.py gist

```

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

```

```

27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is 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 along backwards
45 dwh, dww, dwhy = np.zeros_like(wh), np.zeros_like(ww), np.zeros_like(why)
46 dhh, dhb, dhb = np.zeros_like(hs), np.zeros_like(b_h)
47 dbh = np.zeros_like(b_h)
48 for t in reversed(xrange(len(inputs))):
49     dy = np.zeros(vocab_size, 1)
50     dy[targets[t]] -= 1 # backprop into y
51     dht = np.dot(dwhy, ps[t]) # backprop into h
52     dh = dy
53     dh = np.dot(Wht, dy) + dhh + dbh # backprop into h
54     dhw = (1 - h[t]**2) * h[t] # dh = dh * (1 - h**2) # backprop through tanh nonlinearity
55     dwh += np.dot(dhw, xs[t])
56     dww += np.dot(dhw, hs[t-1])
57     dhh = np.dot(Wht, dhw)
58     for dparam in [dwh, dww, dwhy, dhb, dbh]:
59         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
60     return loss, dwh, dww, dwhy, dhb, dbh, hs[len(inputs)-1]
61
62 def sample(model, ix, n):
63     """
64     sample a sequence of integers from the model
65     ix is memory state, seed, ix is seed letter for first time step
66     """
67     x = np.zeros((vocab_size, 1))
68     ix_to_ix = ix
69     lines = []
70     for t in xrange(n):
71         ix = np.tanh(np.dot(Wxh, x) + np.dot(Whh, ix) + bh)
72         y = np.dot(Why, ix) + by
73         p = np.exp(y) / np.sum(np.exp(y))
74         ix = np.random.choice(vocab_size, 1, p=p).ravel()
75         x = np.zeros((vocab_size, 1))
76         ix_to_ix = ix
77         lines.append(ix)
78     return lines
79
80 #, p = 0, 0
81 mwh, dmwh, dmwh = np.zeros_like(wh), np.zeros_like(ww), np.zeros_like(why)
82 mww, dmww, dmww = np.zeros_like(ww), np.zeros_like(ww), np.zeros_like(ww)
83 smooth_loss = -np.log(1.0/vocab_size)/seq_length # loss at iteration 0
84 while True:
85     # sample inputs (we're sampling from left to right in steps seq_length)
86     if (pseq_length) % len(chars) == 0:
87         hprev = np.zeros(hidden_size, 1) # reset RNN memory
88         p = 0 # p is p from start of data
89         inputs = [char_to_ix[ch] for ch in data[p:pseq_length]]
90         targets = [char_to_ix[ch] for ch in data[pseq_length+1:]]
91
92         # sample from the model now and then
93         if n % 100 == 0:
94             sample_ix = sample(hprev, inputs[0], 200)
95             txt = ''.join(sample_ix) # ix to sample_ix
96             print "-----%s %s-----" % (txt, )
97
98         # forward seq_length characters through the net and fetch gradients
99         loss, dwh, dmwh, dww, dmww, dwhy, dmwhy, hprev = lossFun(inputs, targets, hprev)
100         smooth_loss = smooth_loss * 0.999 + loss * 0.001
101         if n % 100 == 0: print "iter %d, loss %f" % (n, smooth_loss) # print progress
102
103     # perform parameter update with Adagrad
104     for dparam, dparam in zip([dwh, dmwh, dww, dmww, dwhy, dmwhy], [dwh, dmwh, dww, dmww, dwhy, dmwhy]):
105         param += learning_rate * dparam / np.sqrt(param + seq_length) # Adagrad update
106
107     p = seq_length # move data pointer
108     n += 1 # iteration counter

```

```

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)

```

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

$$y_t = W_{hy}h_t$$

Softmax classifier

# min-char-rnn.py gist

```

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

```

```

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

```

```

63 def sample(h, ix):
64     """
65     sample a sequence of integers from the model
66     h is memory state, ix is seed letter for first time step
67     """
68     x = np.zeros(vocab_size, 1)
69     ix_to_ix = ix
70     xs = []
71     for t in xrange(10):
72         xi = np.zeros(vocab_size, 1)
73         xi[ix_to_ix] = 1
74         hxi = np.tanh(np.dot(wih, xi)) + np.dot(whh, h) + bi
75         yi = np.dot(why, hxi) + bo
76         pi = np.exp(yi) / np.sum(np.exp(yi))
77         ix = np.random.choice(vocab_size, 1, p=pi)[0]
78         x[ix_to_ix] = ix
79         ix_to_ix = ix
80     return xs
81
82 #, p = 0, 0
83 dbh, dbh, dwhy = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
84 dbh, dby = np.zeros_like(bi), np.zeros_like(bo) # memory variables for Adam
85 smooth_loss = np.log(0.0/vocab_size)/seq_length # loss at iteration 0
86 while True:
87     # generate inputs (we're sampling from left to right in steps seq_length long)
88     if p%seq_length == 0:
89         hprev = np.zeros(hidden_size, 1) # reset mem memory
90         p = 0 # p from count 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+seq_length:p+seq_length+1]]
93     # sample from the model now and then
94     if p % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print "==== %d %s =====" % (p, txt)
98     # Forward seq_length characters through the net and fetch gradients
99     loss, dbh, dbh, dwhy, dbh, dby, hprev = lossfun(inputs, targets, hprev)
100     smooth_loss = smooth_loss + 0.999 * loss + 0.001
101     p = p + 1 or p % print_iter == 0:
102         print "iter %d, loss %f" % (p, smooth_loss) # print progress
103
104 # perform parameter update with Adam
105 for param, dparam, mem in zip([wih, whh, why, bi, bo],
106                             [dbh, dbh, dwhy, dbh, dby],
107                             [dbh, dbh, dwhy, dbh, dby]):
108     mem += dparam * learning_rate + dparam / np.sqrt(mem + 1e-8) # adagrad update
109
110 p = seq_length # move data pointer
111 # # = 1 # iteration counter

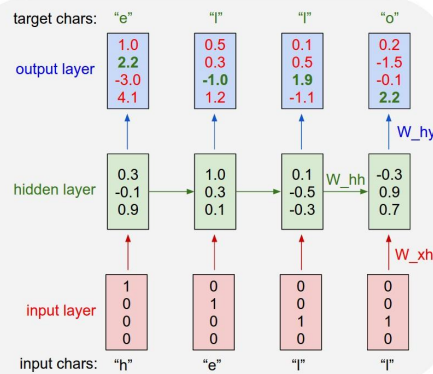
```

```

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

```

recall:



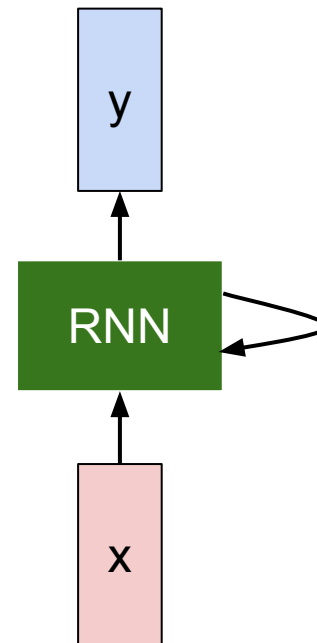
# min-char-rnn.py gist

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD license
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print "data has %d characters, %d unique." % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossfun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is wci array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     ix, hix, yix, ox = [], [], [], []
34     hix[0] = hprev
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi[i] = np.zeros((vocab_size,1)) # encode in 1-of-K representation
39         xi[i][inputs[t]] = 1
40         hi[i] = np.tanh(np.dot(wh, xi[i]) + np.dot(whh, hi[i-1]) + bh) # hidden state
41         yi[i] = np.dot(why, hi[i]) + by # unnormalized log probabilities for next chars
42         pi[i] = np.exp(yi[i]) / np.sum(np.exp(yi[i])) # probabilities for next chars
43         loss += -np.log(pi[i][targets[t],0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients using backward
45     dwh, dwhh, dwhy = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
46     dh, dhv = np.zeros_like(hi), np.zeros_like(hi)
47     dhnxt = np.zeros_like(hi[0])
48     for t in reversed(xrange(len(inputs))):
49         dy = np.copy(pi[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dhy = np.dot(dy, hi[t].T)
52         dxi = dy
53         dhi = np.dot(dhy.T, dy) + dhnxt # backprop into h
54         dhrw = (1 - hi[t]**2) * dhi # dh a backprop through tanh nonlinearity
55         dwh += np.dot(dhrw, xi[t].T)
56         dwhh += np.dot(dhrw, hi[t-1].T)
57         dhnxt = np.dot(dhrw, hi[t].T)
58         dhnxt = np.dot(dhnxt.T, dhrw)
59         for dparam in [dwh, dwhh, dwhy, dhv, dby]:
60             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61     return loss, dwh, dwhh, dwhy, dhv, dby, hi[len(inputs)-1]
62
63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     ix = [seed_ix]
70     for t in xrange(n):
71         h = np.tanh(np.dot(Wxh, 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         ixes.append(ix)
78     return ixes
79
80 # main loop
81 """
82 main, mwh, mwhh, mwhy = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
83 mwh, mwhh, mwhy, dhv, dby, hprev = None, None, None, None, None, None
84 smooth_loss = np.log(1/vocab_size)/seq_length # loss at iteration 0
85 while True:
86     # generate inputs (we're sampling from left to right in steps seq_length long)
87     if pseq_length == len(inputs) or p == 0:
88         hprev = np.zeros((hidden_size,1)) # reset mem memory
89         p = 0 # p from start of data
90         inputs = [char_ix[ix] for ix in data[p:pseq_length]]
91         targets = [char_ix[ix] for ix in data[pseq_length+1:]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print "----%s%s-----" % (txt, )
98         n += 1
99     # forward seq length characters through the net and fetch gradients
100     loss, dwh, dwhh, dwhy, dhv, dby, hprev = lossfun(inputs, targets, hprev)
101     smooth_loss = smooth_loss + (loss - smooth_loss) * 0.001
102     if n % 100 == 0: print "iter %d, loss %f" % (n, smooth_loss) # print progress
103
104     # perform parameter update with Adagrad
105     for dparam, dparam, mwh in zip([dwh, mwh, why, dh, by],
106                                   [dwh, mwh, why, dhv, dby],
107                                   [mwh, mwhh, mwhy, dhv, dby]):
108         mwh += dparam * dparam / np.sqrt(mwh + 1e-8) # adagrad update
109
110     p += seq_length # move data pointer
111     n += 1 # iteration counter
112 """
```

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



«p class="clear» > Products: Laser-Printers: The fundamental everyday requirement for mono and colour laser printing throughout today's office is perfectly met with the extensive Epson laser printer range. The latest AcuLaser printer range offers users exceptionally Epson AcuLaser C1900: Networked compact colour laser printer for professional enterprises. Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more with the Epson-AcuLaser C1900. Epson brings both colour and monochrome laser printing together at a black and white price. more Where to Buy Support Epson AcuLaser C2000: The fastest colour laser printer in its class. The perfect printer for small businesses and work groups, the Epson-AcuLaser C2000 prints high volumes in black and white and vibrant colour, at high speed and with low running costs. more Where to Buy Support Epson AcuLaser C2500: Large paper capacity, 600 sheets, expandable up to 1,500 sheets. Compatible Windows and Mac. High speed USB and EpsonNet 10/100 Base-TX Ethernet interfaces as standard» \* Epson AcuLaser Resolution Improvement Technology \* EpsonNet 10/100 Base-TX Ethernet standard with Epson AcuLaser C2000 model only. AcuLaser C2000: 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing as standard AcuLaser C2000i: 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing, 10/100Base-TX Ethernet Interface Networked compact colour laser printer for professional enterprises. Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more with the Epson-AcuLaser C1900. Epson brings both colour and monochrome laser printing together at a black and white price. Key features cost effective mono printing for day to day business needs and vivid versatile colour when required. Search Epson UK Epson AcuLaser C200: Outstanding professional colour printing for business. Add colour to your business with the Epson AcuLaser C600 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high performance black and white production. more Where to Buy Support As cost efficient to run as a mono-only user printer. Paper capacity of 700 sheets from two media sources. Easy to operate with advanced print driver. Memory expandable from 32MB to 1024MB. Pre-configured models available with Wireless 802.11b, Adobe® PostScript® 3 Level 3™ and two-sided printing. The AcuLaser C1900 is available in 3 configurations: - AcuLaser C1900i: with 32MB, 200 Sheet MP Tray, 10/100Base-TX Networking - AcuLaser C1900: with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100Base-TX Networking Support Epson AcuLaser C4100: High performance colour lasers for all your business printing needs. The Epson AcuLaser C4100 provides businesses with a high performance colour and monochrome printing solution. It adds crucial colour to your business, while producing high quality monochrome output at lower costs than many monochrome-only printers, and is just as easy to operate. So now there's no reason to buy two printers, because perfect monochrome and colour solutions are available in one. more Where to Buy Support Epson AcuLaser C600: Professional high performance A3W colour laser printer. Epson AcuLaser C600 is the perfect professional printing solution for users who require exceptional quality colour and mono output on a range of media formats from C5 up to A3W in size. The Epson AcuLaser C600 is able to achieve superb print quality by utilising a combination of Epson's exclusive AcuLaser Colour Laser Technologies. more Where to Buy Support AcuLaser C1900PS: with Adobe® PostScript® 3™, 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100Base-TX Networking - AcuLaser C1900i: with Duplex unit (two sided printing) 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100Base-TX Networking - AcuLaser C1900 Wi-Fi: with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, Wireless Networking facility. Add colour to your business with the Epson AcuLaser C800 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high. Support Epson AcuLaser C400: High performance colour laser. The Epson AcuLaser C400 provides businesses with high performance colour and monochrome printing solutions. more Where to Buy Support Epson AcuLaser C3100: High speed A3 colour laser printer. Why have separate black and white and colour printers when you can have the Epson AcuLaser C3100? Epson has taken the lead in laser technology to deliver a complete high-performance solution for all your colour and mono printing needs. Support EPL-6200L: High performance A4 mono laser professional printers. The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more Where to Buy Support EPL-6200: High performance A4 mono laser professional printers. The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more performance black and white production. For the first time, you can now bring the power of high quality colour to your documents without suffering the high costs or low speeds traditionally associated with colour



## Sonnet 116 – Let me not ...

*by William Shakespeare*

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

at first:

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

↓  
train more

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

↓  
train more

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

↓  
train more

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

PANDARUS:

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

Second Senator:

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

DUKE VINCENTIO:

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

Second Lord:

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

Clown:

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

VIOLA:

I'll drink it.

VIOLA:


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

KING LEAR:

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























# 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	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	2. Conventions	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	3. Set Theory	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	4. Categories	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	5. Topology	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	6. Sheaves on Spaces	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	7. Sites and Sheaves	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	8. Stacks	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	9. Fields	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	10. Commutative Algebra	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 

**Parts**

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

**Statistics**

The Stacks project now consists of

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

Latex source



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

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

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

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

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

and

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

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

*Proof.* See discussion of sheaves of sets.  $\square$

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

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

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

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

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

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

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

The following lemma surjective restrocomposes of this implies that  $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\dots,0}$ .

**Lemma 0.2.** Let  $X$  be a locally Noetherian scheme over  $S$ ,  $E = \mathcal{F}_{X/S}$ . Set  $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$ . Since  $\mathcal{I}^n \subset \mathcal{I}^n$  are nonzero over  $i_0 \leq \mathfrak{p}$  is a subset of  $\mathcal{J}_{n,0} \circ \overline{A_2}$  works.

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

*Proof.* We will use the property we see that  $\mathfrak{p}$  is the next functor (??). On the other hand, by Lemma ?? we see that

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

where  $K$  is an  $F$ -algebra where  $\delta_{n+1}$  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  $X_{\acute{e}tale}$  we have

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

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

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

*Proof.* See Spaces, Lemma ?? □

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

*The following to the construction of the lemma follows.*

*Let  $X$  be a scheme. Let  $X$  be a scheme covering. Let*

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

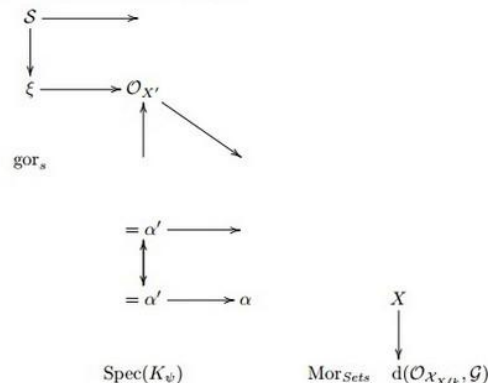
*be a morphism of algebraic spaces over  $S$  and  $Y$ .*

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

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

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

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



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

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

□

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

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

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

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_x \quad -1(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_{\acute{e}tale}}^{-1}(\mathcal{O}_{X_{\acute{e}tale}}(\mathcal{O}_{X_{\acute{e}tale}}^{\vee}))$$

is an isomorphism of covering of  $\mathcal{O}_{X_{\acute{e}tale}}$ . 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  $\mathcal{O}_{X_{\acute{e}tale}}$  is a closed immersion, see Lemma ?? . This is a sequence of  $\mathcal{F}$  is a similar morphism.



torvalds / linux

Watch 3,711 Star 23,054 Fork 9,141

Linux kernel source tree

520,037 commits 1 branch 420 releases 5,039 contributors

branch: master - linux / +

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

torvalds authored 9 hours ago

latest commit 4b1706927d

Documentation	Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending	6 days ago
arch	Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/lin...	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/lin...	a day ago
init	init: fix regression by supporting devices with major:minor:offset fo...	a month ago
io	Merge branch 'for-linus' of git://git.kernel.org/pub/scm/linux/kernel...	a month ago

Code

Pull requests 74

Pulse

Graphs

HTTPS clone URL

https://github.com/torvalds/linux

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

```

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

```

# Generated C code

```

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

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

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

```

```

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

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

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

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

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

```

# Searching for interpretable cells

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

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

# Searching for interpretable cells

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

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

quote detection cell



# Searching for interpretable cells

Cell sensitive to position in line:

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

line length tracking cell



# Searching for interpretable cells

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

if statement cell

# Searching for interpretable cells

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

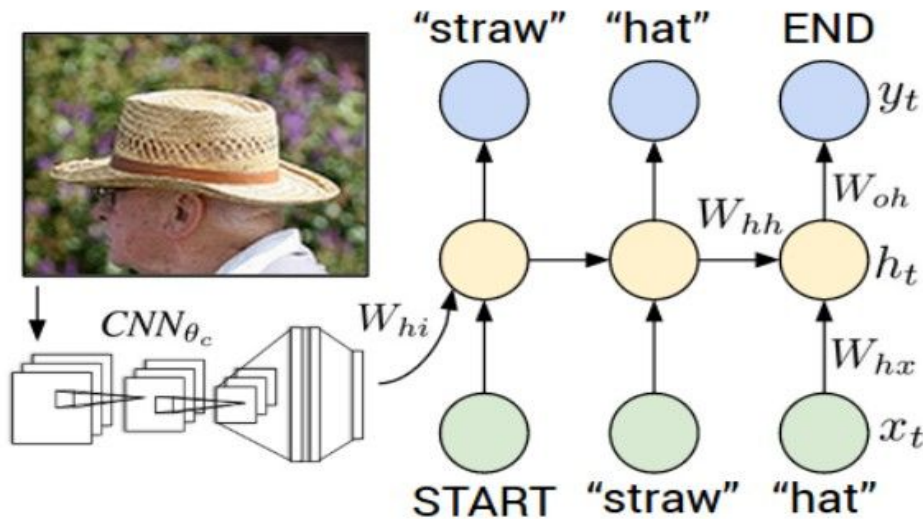
quote/comment cell

# Searching for interpretable cells

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

code depth cell

# Image Captioning



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

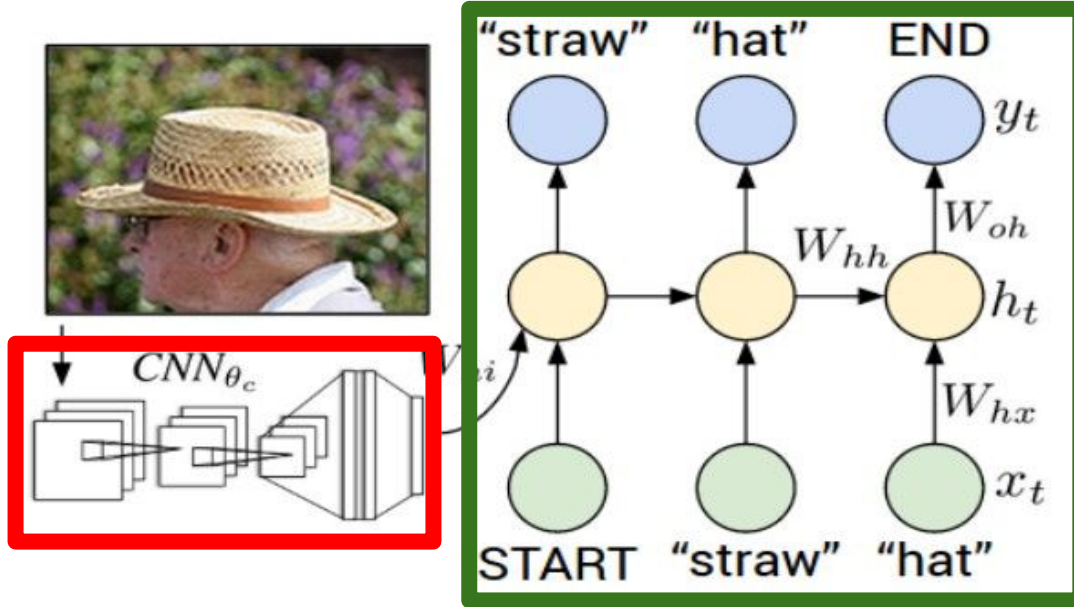
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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

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

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

# Recurrent Neural Network



## Convolutional Neural Network





test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

x0  
<STA  
RT>

<START>

image

test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

V

y0

h0

x0

<STA

RT>

<START>

Wih

before:

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

now:

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



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

y0

h0

x0  
<START  
RT>

straw

sample!

<START>

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

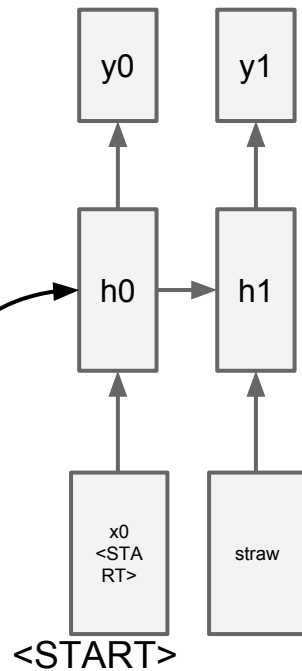
maxpool

FC-4096

FC-4096



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

y0

y1

h0

h1

x0  
<START  
RT>

straw

hat

sample!

<START>

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

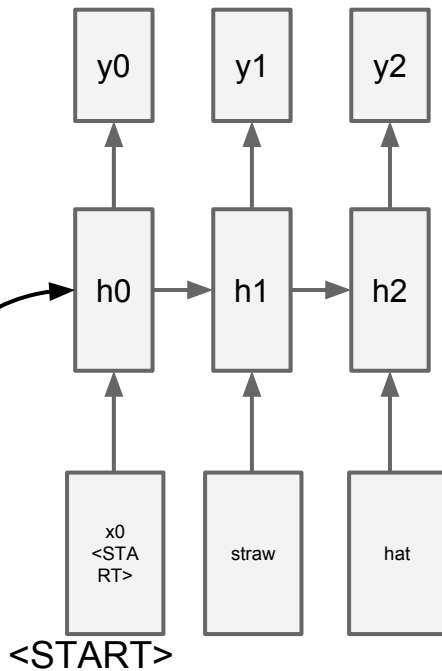
maxpool

FC-4096

FC-4096



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

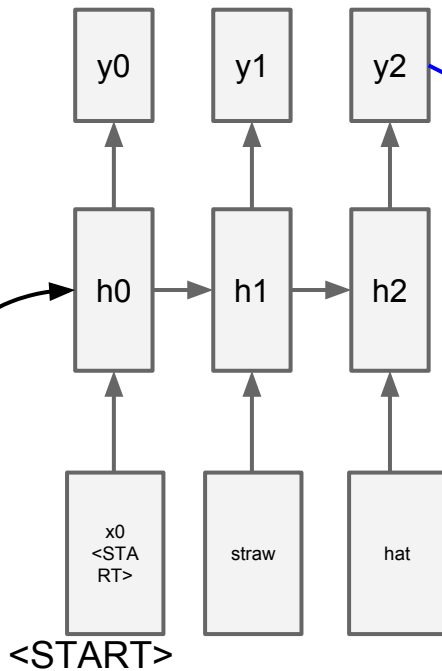
maxpool

FC-4096

FC-4096



test image



sample  
<END> token  
=> finish.



# 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](http://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."

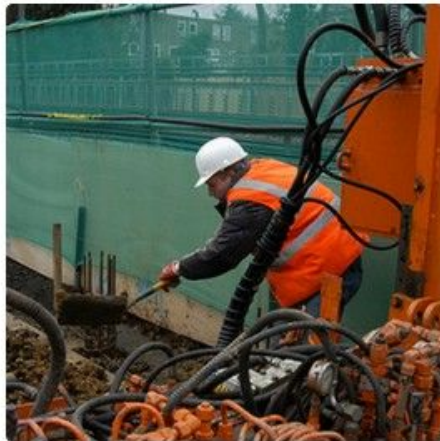


"boy is doing backflip on wakeboard."





"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."



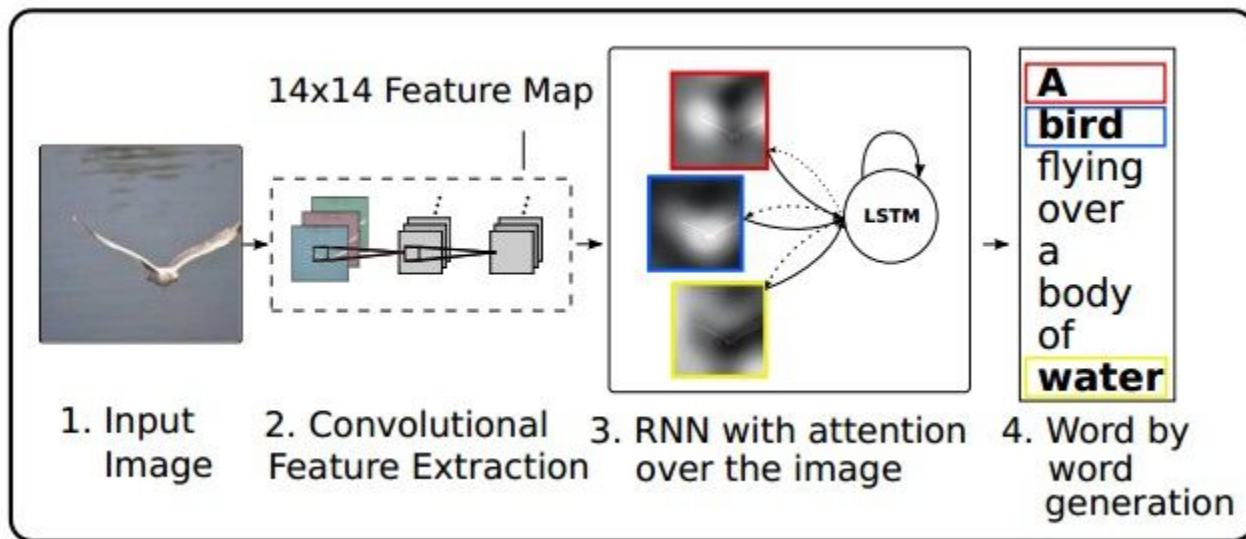
"a woman holding a teddy bear in front of a mirror."



"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:



*Show Attend and Tell, Xu et al., 2015*

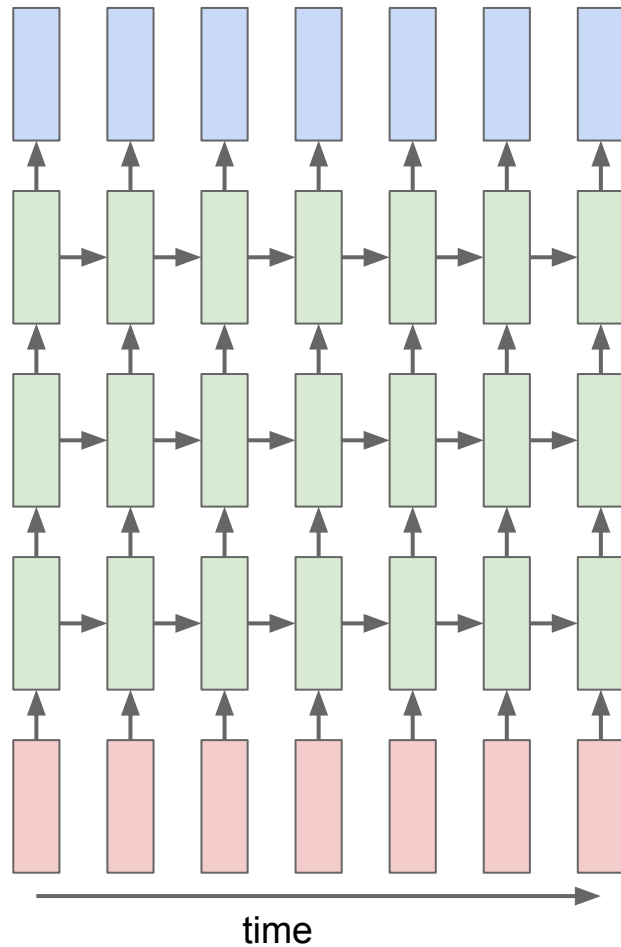
# RNN:

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

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



depth





# RNN:

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

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

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

# LSTM:

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

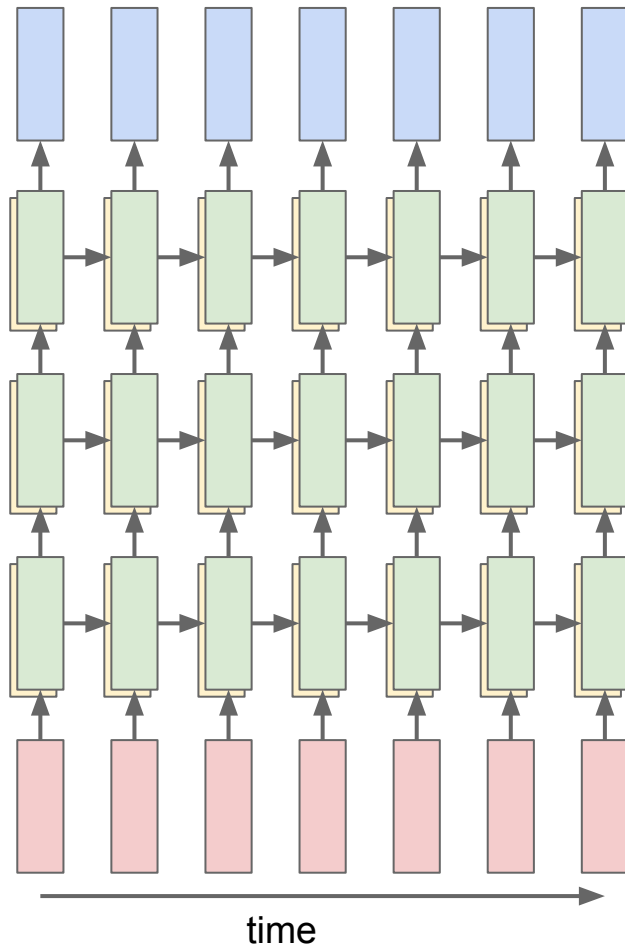
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

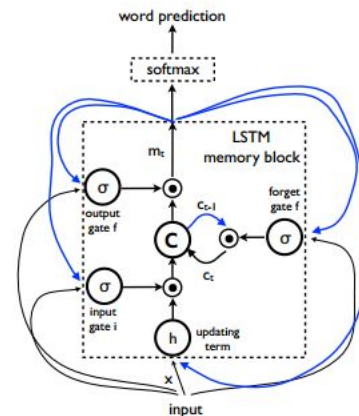
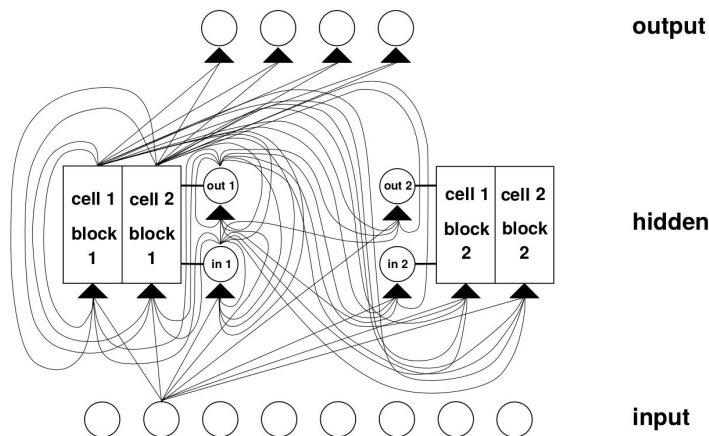
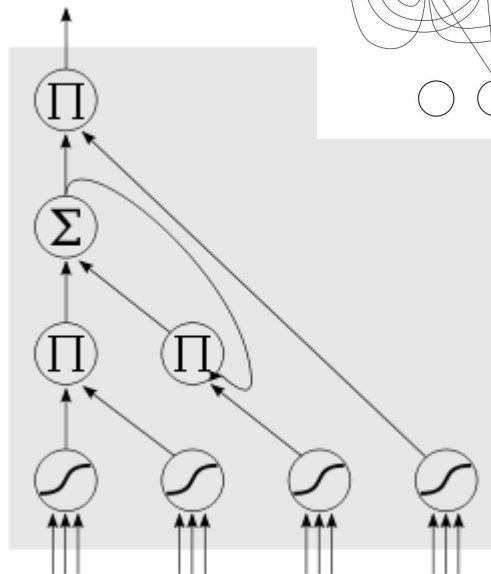
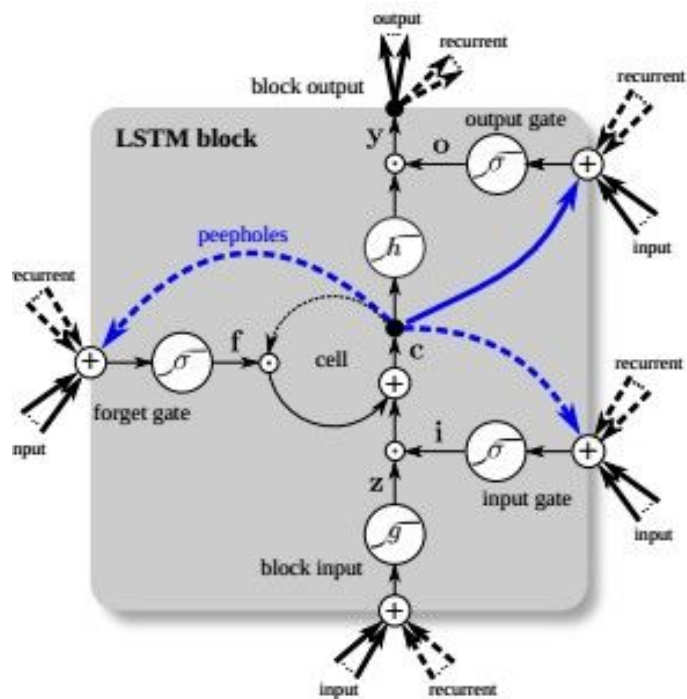
$$h_t^l = o \odot \tanh(c_t^l)$$



depth

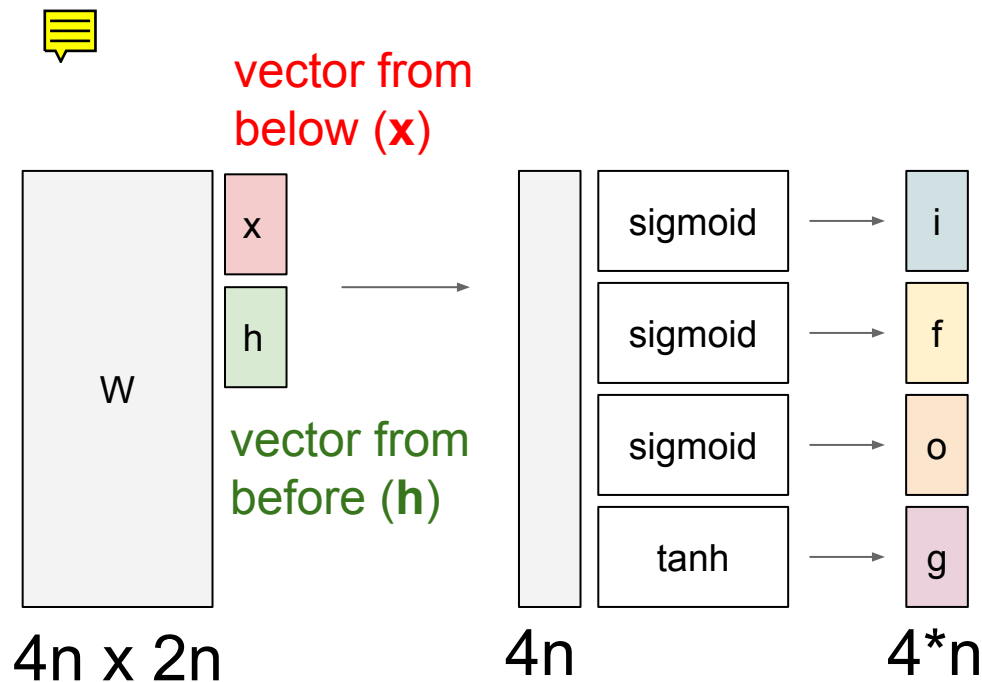


# LSTM



# Long Short Term Memory (LSTM)

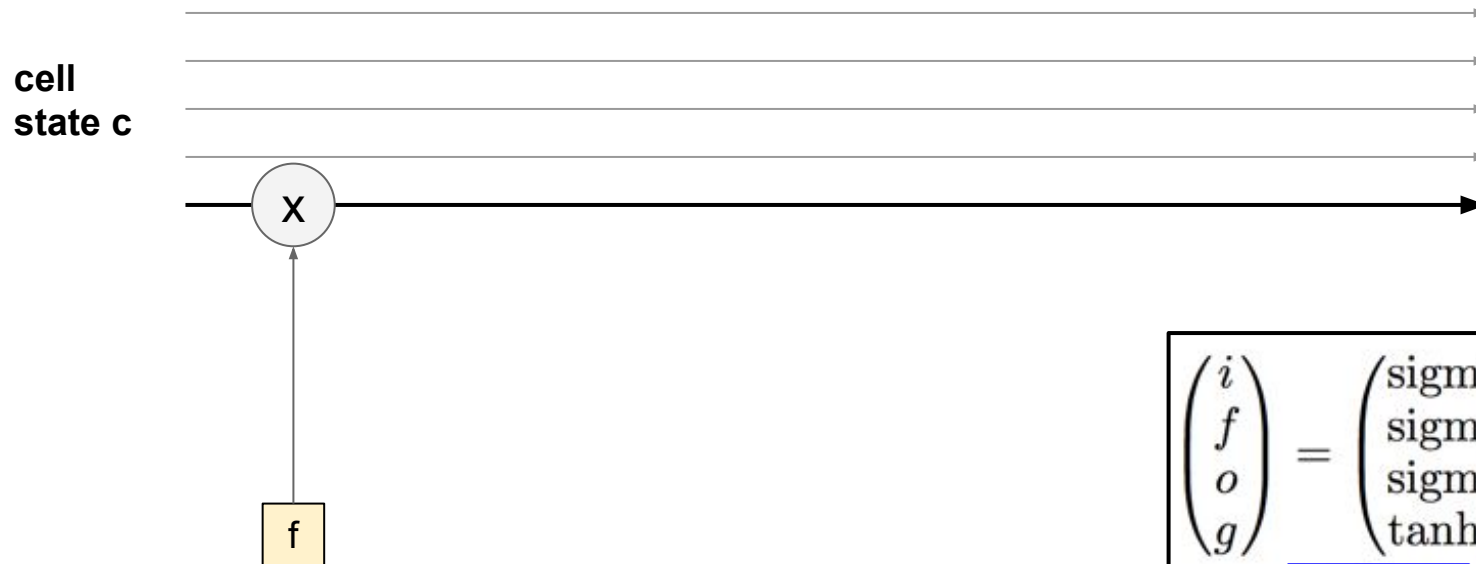
[Hochreiter et al., 1997]



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

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



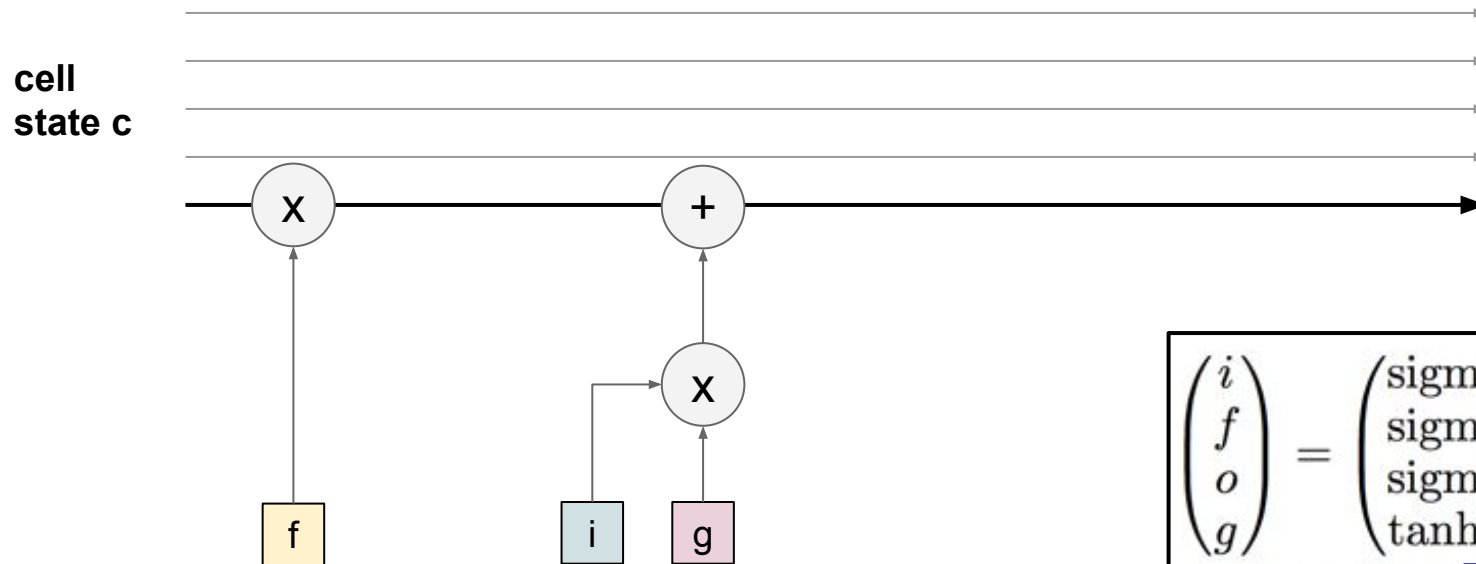
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

# Long Short Term Memory (LSTM)

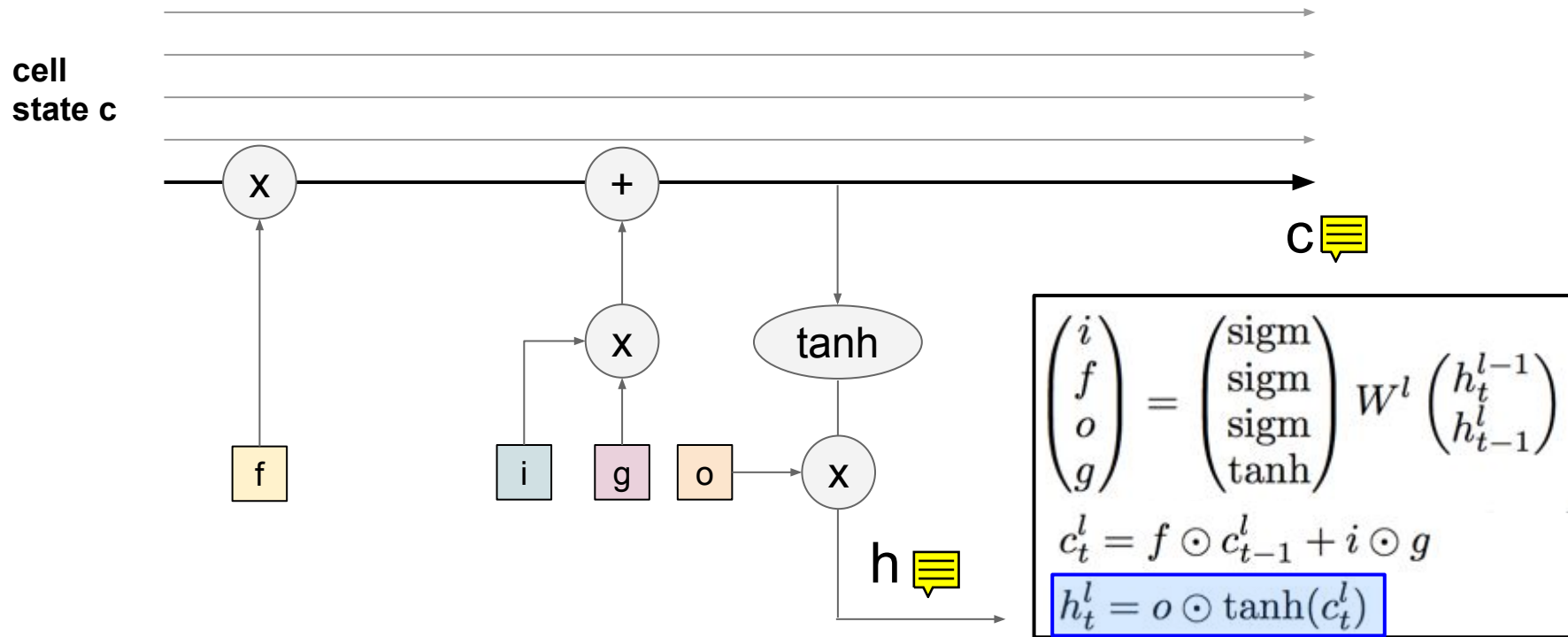
[Hochreiter et al., 1997]



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

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

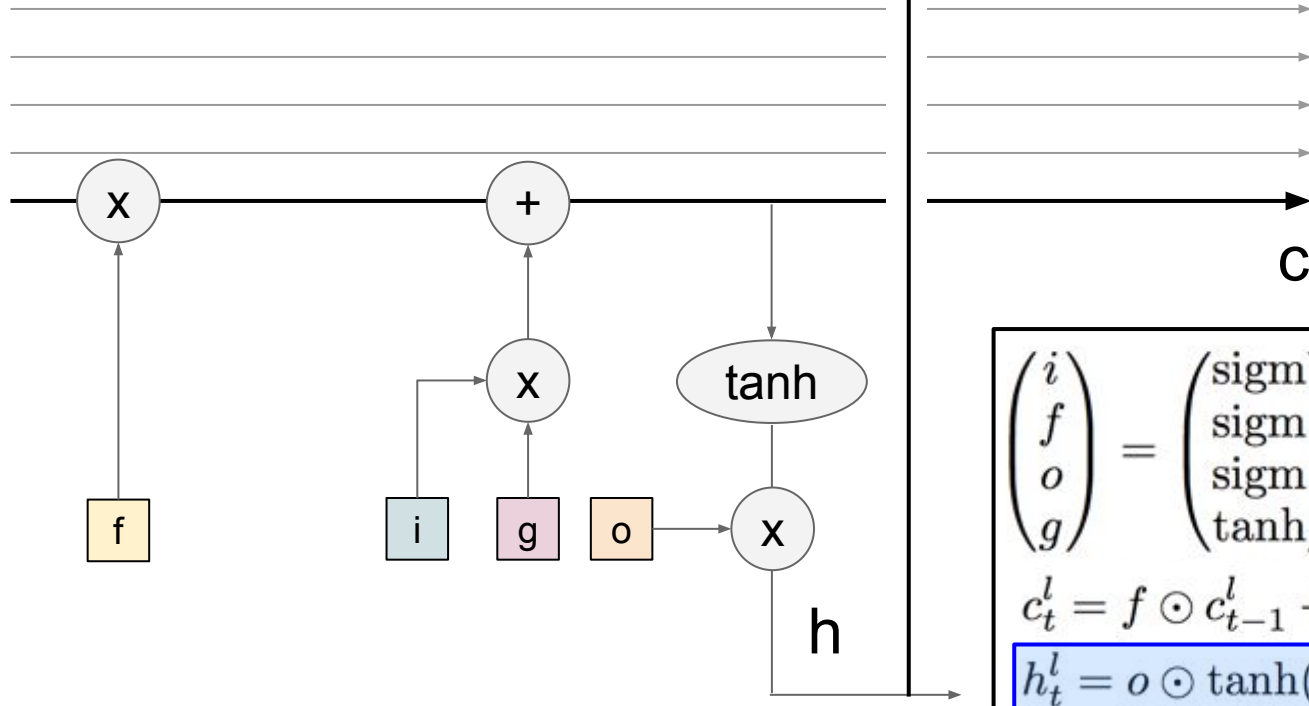




# Long Short Term Memory (LSTM)

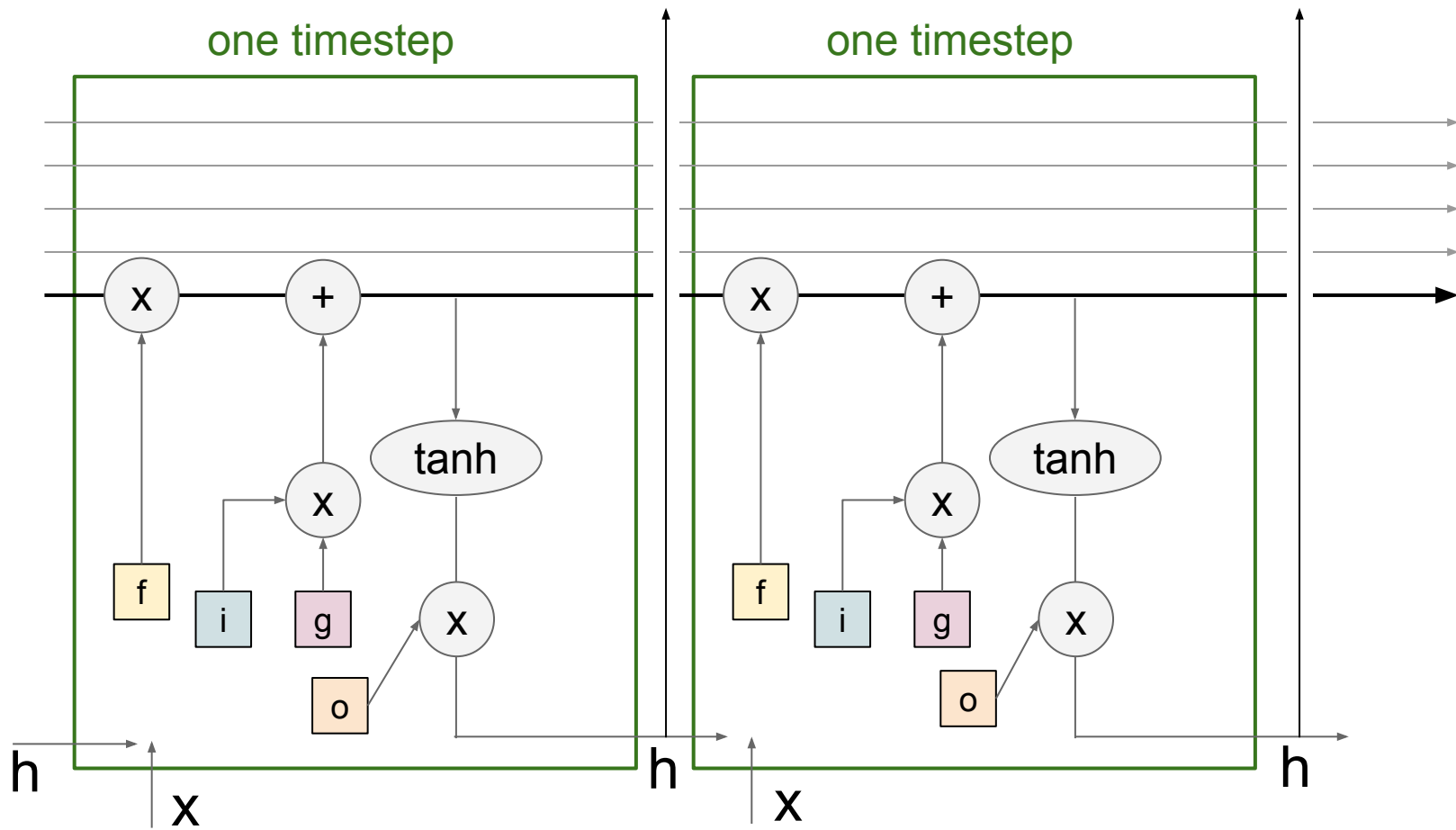
[Hochreiter et al., 1997]

cell  
state  $c$

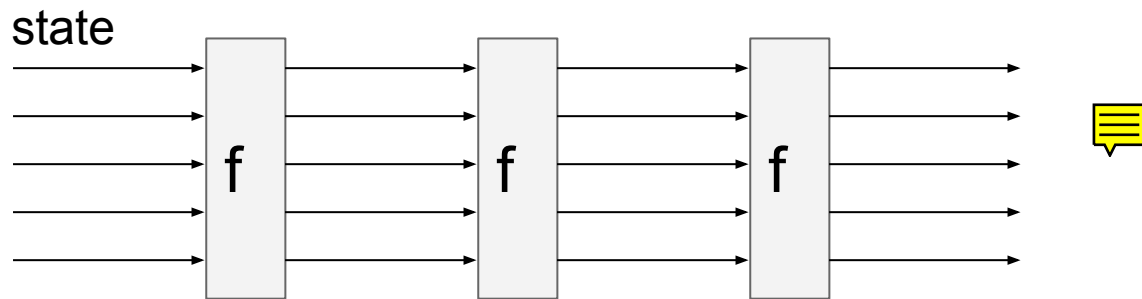


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

# LSTM

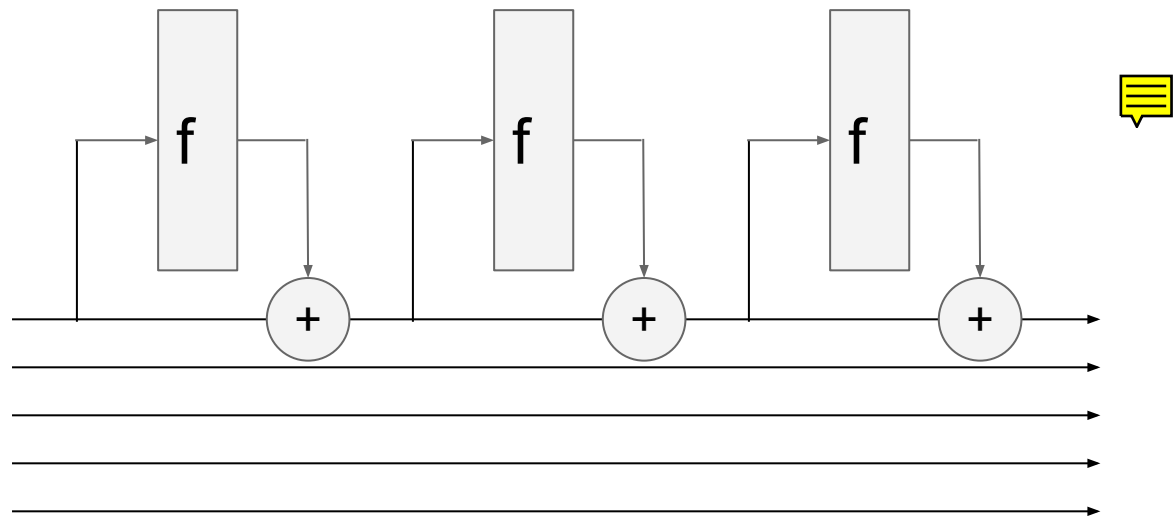


# RNN



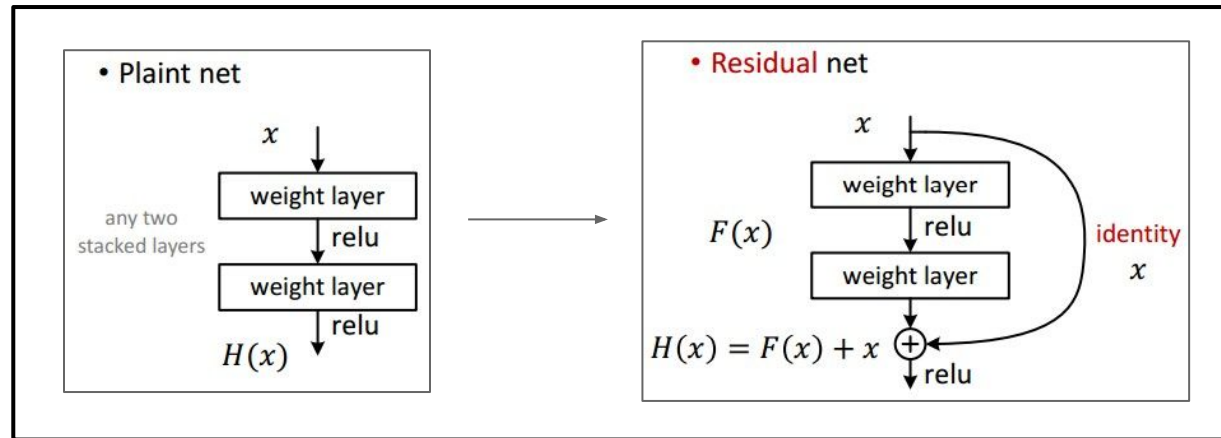
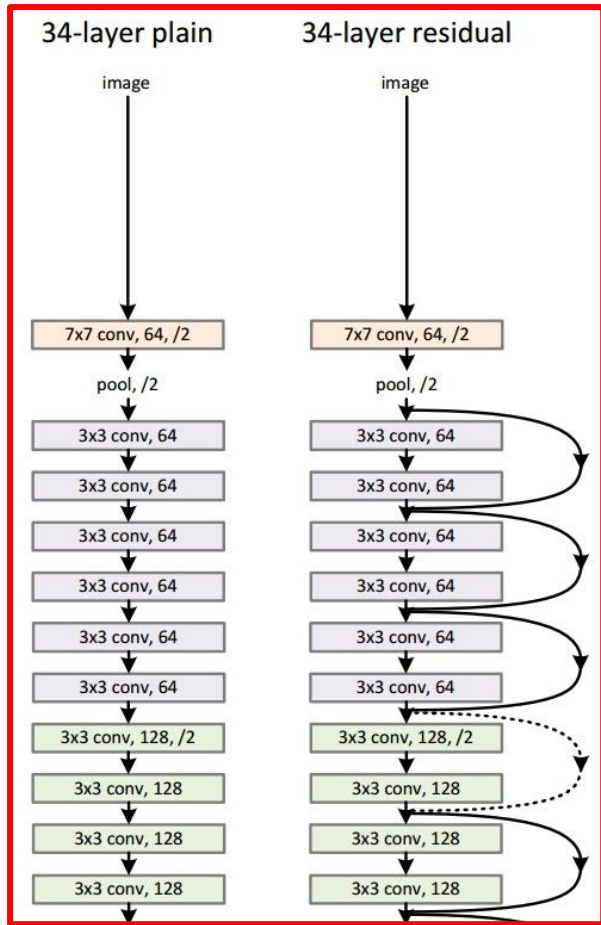
# LSTM

(ignoring  
forget gates)



# 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
```

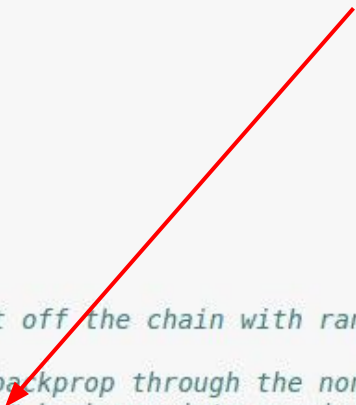
# Understanding gradient flow dynamics

```
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
```

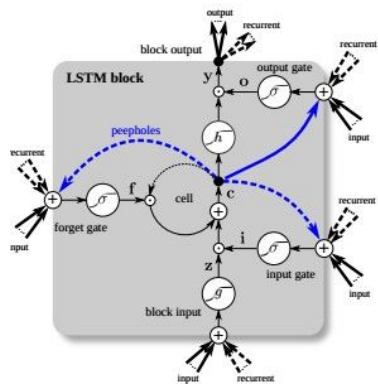
if the largest eigenvalue is  $> 1$ , gradient will explode  
if the largest eigenvalue is  $< 1$ , gradient will vanish

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:

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

MUT2:

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

MUT3:

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

# Summary

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