

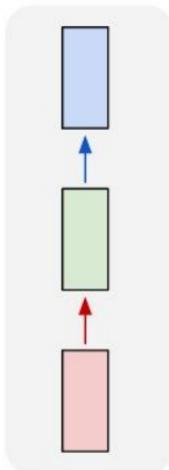
# Recurrent Neural Networks

Instructor: Alan Ritter

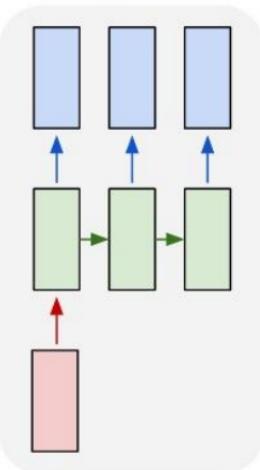
Slides Adapted from Andrej Karpathy

# Recurrent Networks offer a lot of flexibility:

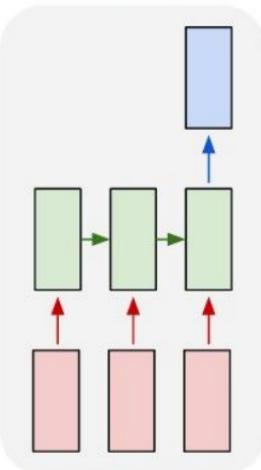
one to one



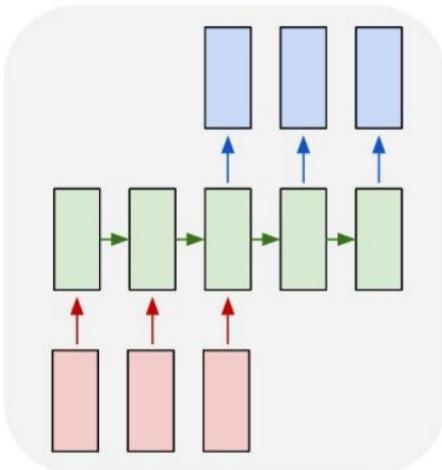
one to many



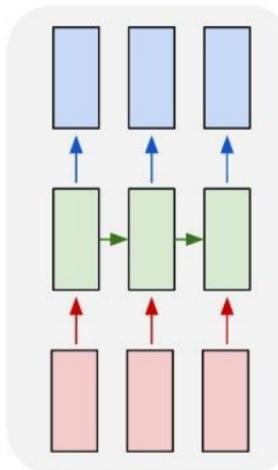
many to one



many to many



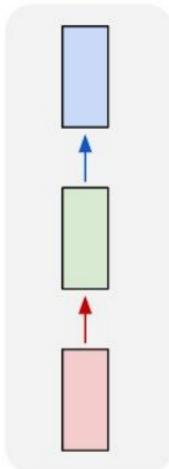
many to many



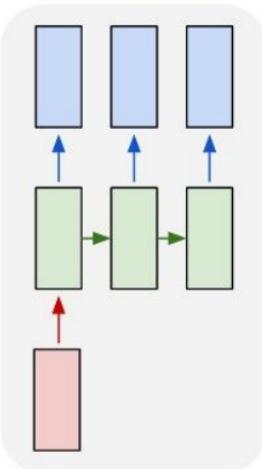
**Vanilla Neural Networks**

# Recurrent Networks offer a lot of flexibility:

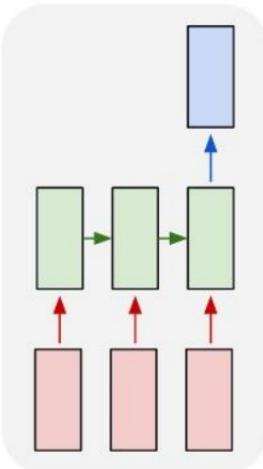
one to one



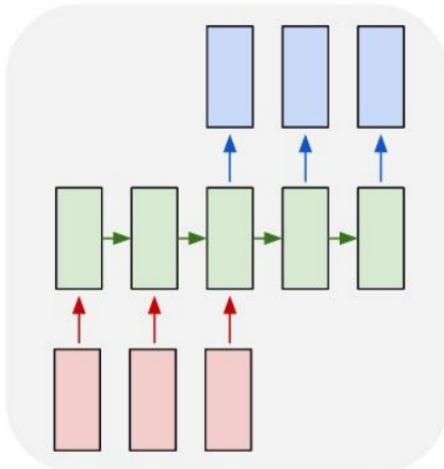
one to many



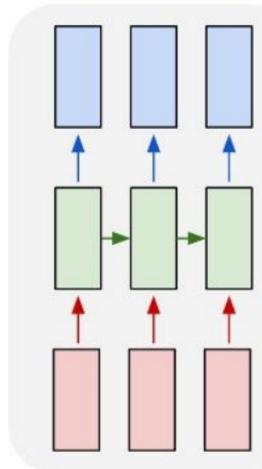
many to one



many to many



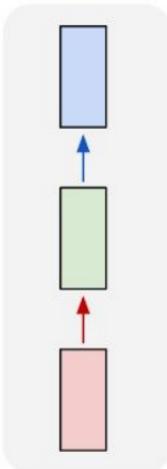
many to many



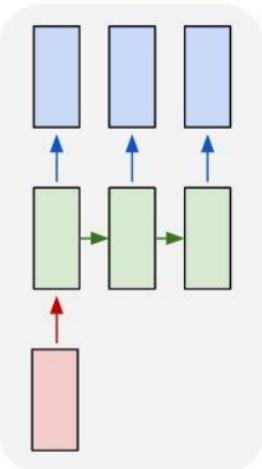
e.g. **Image Captioning**  
image -> sequence of words

# Recurrent Networks offer a lot of flexibility:

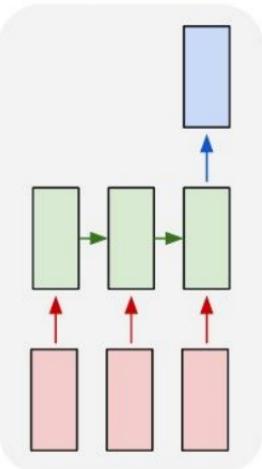
one to one



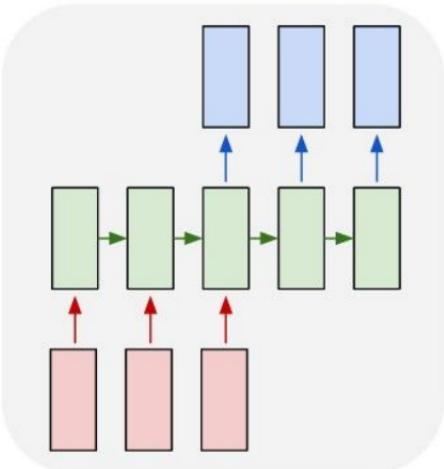
one to many



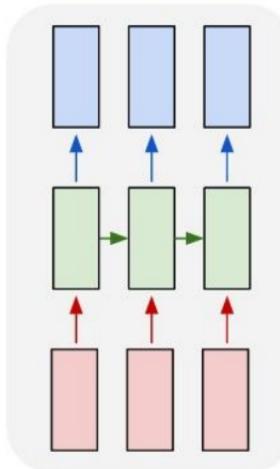
many to one



many to many



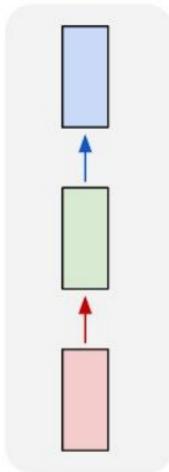
many to many



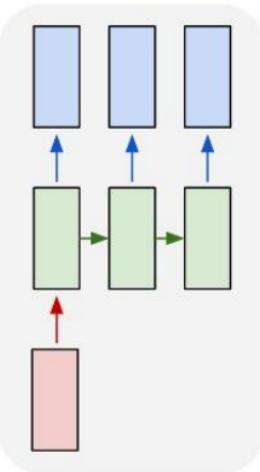
e.g. **Sentiment Classification**  
sequence of words -> sentiment

# Recurrent Networks offer a lot of flexibility:

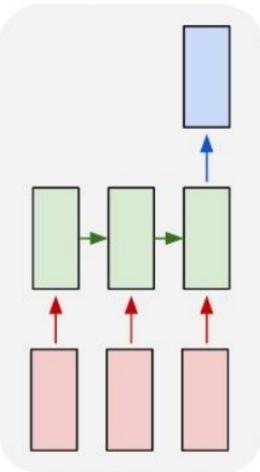
one to one



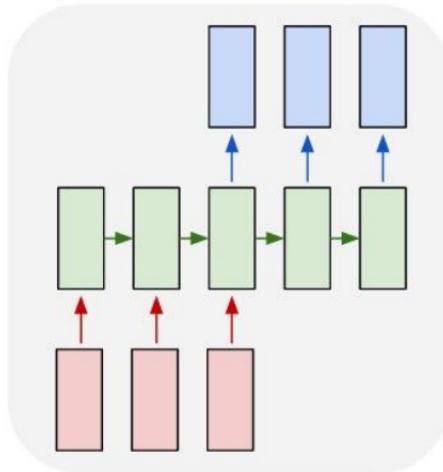
one to many



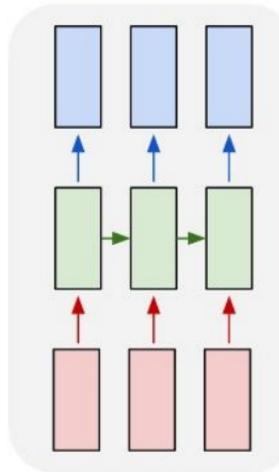
many to one



many to many



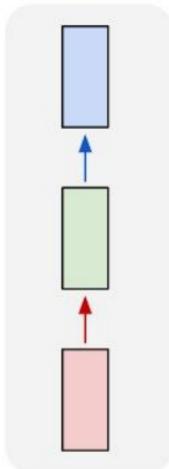
many to many



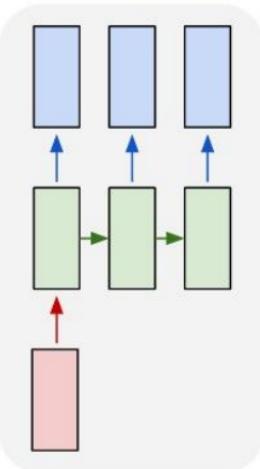
e.g. **Machine Translation**  
seq of words -> seq of words

# Recurrent Networks offer a lot of flexibility:

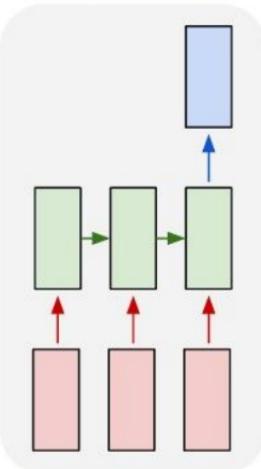
one to one



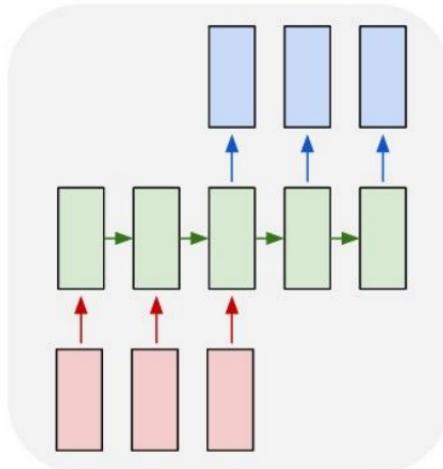
one to many



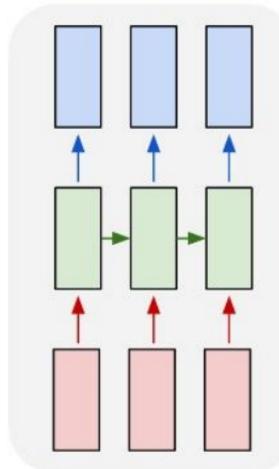
many to one



many to many

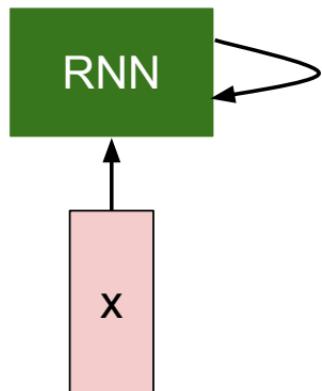


many to many

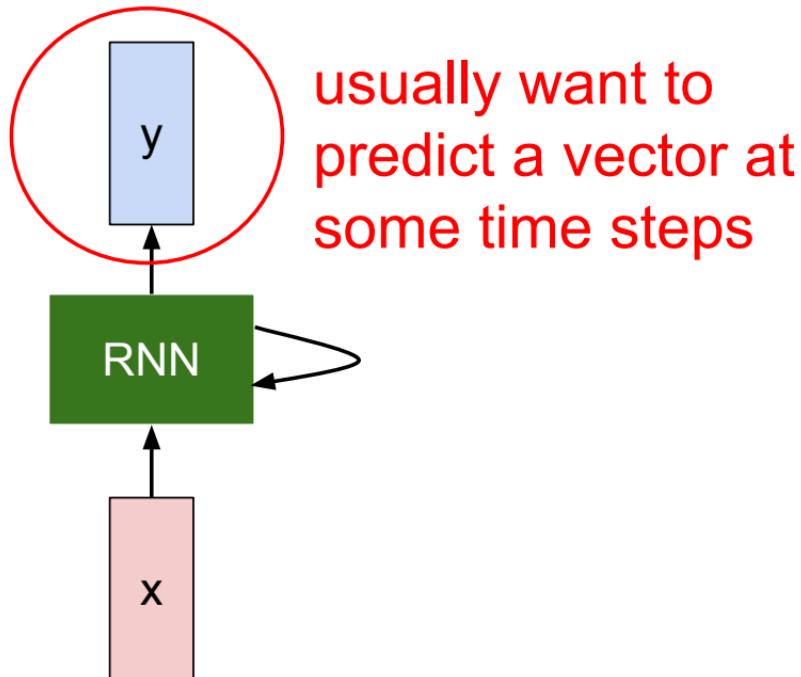


e.g. Video classification on frame level

# Recurrent Neural Network



# Recurrent Neural Network

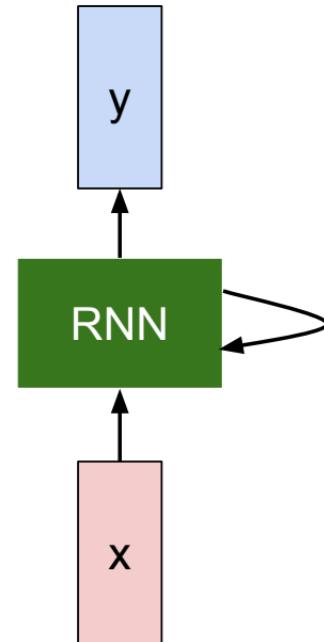


# Recurrent Neural Network

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

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

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

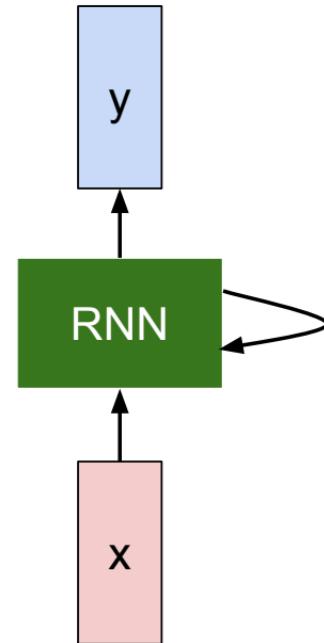


# Recurrent Neural Network

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

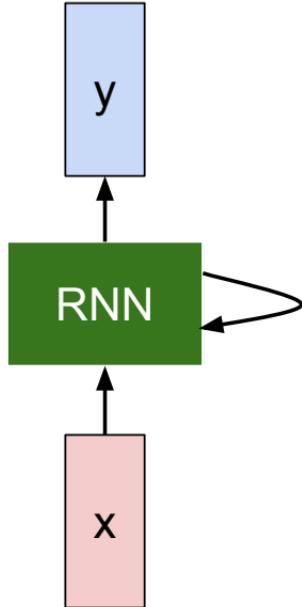
$$h_t = f_W(h_{t-1}, x_t)$$

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  $\mathbf{h}$ :



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

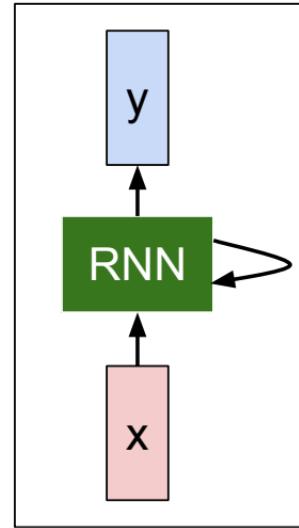
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

# Character-level language model example

Vocabulary:  
[h,e,l,o]

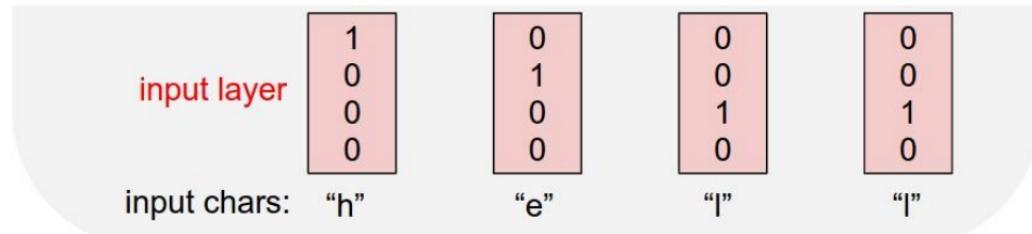
Example training  
sequence:  
“hello”



# Character-level language model example

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“hello”

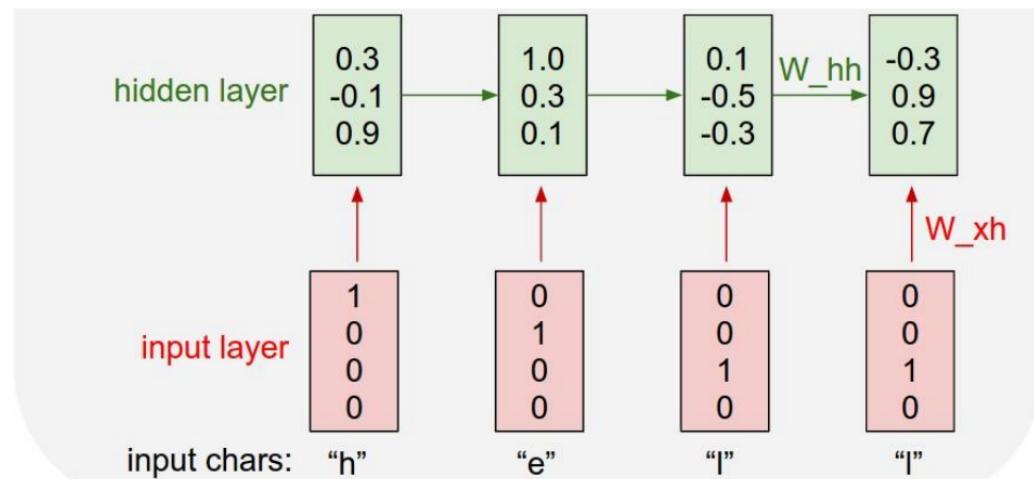


# Character-level language model example

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“hello”

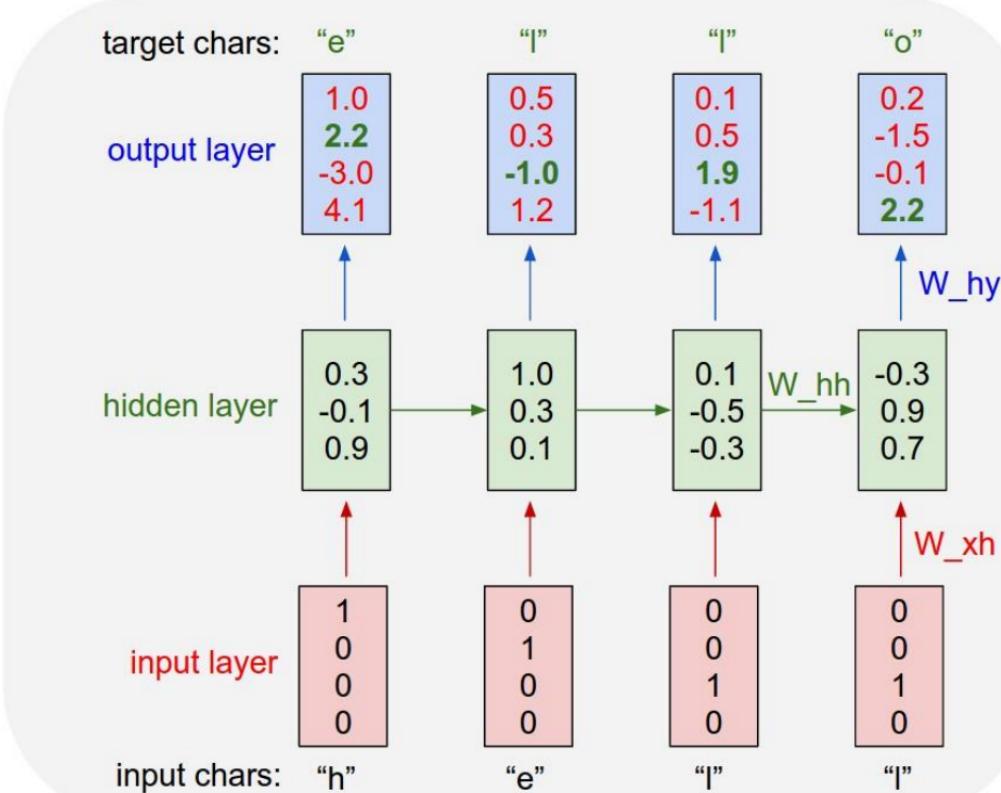
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



# Character-level language model example

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“hello”



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

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print('data has %d characters, %d unique.' % (data_size, vocab_size))
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 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 hxi 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 range(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(Whh, xs[t]) + np.dot(Why, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t]]) # softmax (cross-entropy loss)
44
45         # backward pass: compute gradients going backwards
46         dkh, dwhh, dhy = np.zeros_like(Whh), np.zeros_like(Why), np.zeros_like(by)
47         dhh = np.zeros_like(hs[0])
48         for t2 in reversed(xrange(len(inputs))):
49             dy = np.copy(ps[t2])
50             dy[targets[t2]] -= 1 # backprop into y
51             dhy += np.dot(dy, hs[t2].T)
52             dkh += dy
53             dh = np.dot(Why.T, dy) * dkhnext # backprop into h
54             ddraw = (1 - hs[t2] * hs[t2]) * dh # backprop through tanh nonlinearity
55             dhh += ddraw
56             dwhh += np.dot(draw, xs[t2].T)
57             dwhh += np.dot(draw, hs[t2-1].T)
58             dnext = np.dot(Whh.T, draw)
59             for dparam in [dwhh, dwhh, dhy, dbh, dby]:
60                 np.clip(dparam, -5, 5) # clip to mitigate exploding gradients
61             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
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(Whh, x) + np.dot(Why, 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[ix] = 1
77             ixes.append(ix)
78         return ixes
79
80     n, p, o, 0
81     mxhh, mwhy, mbhy = np.zeros_like(Whh), np.zeros_like(Why), np.zeros_like(by)
82     mbh, mbp = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
83     smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
84     while True:
85         if pseq_length1 >= 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: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' + txt + '----')
96
97         # forward seq_length characters through the net and fetch gradient
98         loss, dxh, dwhh, dhy, 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([Whh, Why, bh, by],
104                                     [dwhh, dhy, dbh, dby],
105                                     [mxhh, mwhy, mbhy, mbp]):
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
```

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

## min-char-rnn.py gist

Data I/O

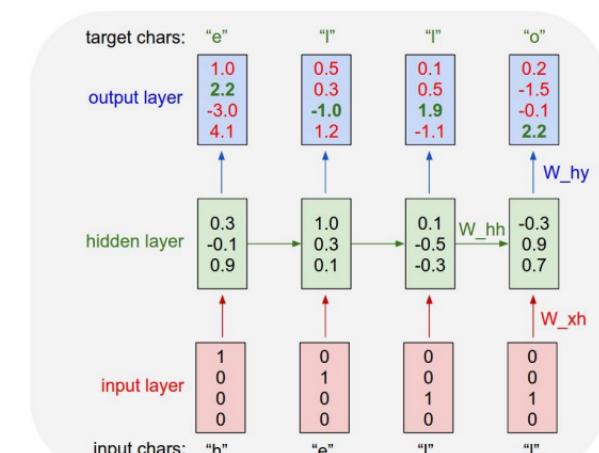
```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11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

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

## Initializations

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

recall



# min-char-rnn.py gist

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  n = 100
8  data = open('input.txt', 'r').read() + ' '
9  chars = list(set(data))
10 vocab_size = len(chars)
11 seq_length = 20 # number of steps to unroll the RNN for
12 learning_rate = 1e-3
13
14 # model parameters
15 wih = np.random.uniform(-0.1, 0.1, (vocab_size, hidden_size)) * hidden_size # input to hidden
16 whh = np.random.uniform(-0.1, 0.1, (hidden_size, hidden_size)) * hidden_size # hidden to hidden
17 why = np.random.uniform(-0.1, 0.1, (hidden_size, vocab_size)) * vocab_size # hidden to output
18 bh = np.zeros((hidden_size, 1)) # hidden bias
19 by = np.zeros((vocab_size, 1)) # output bias
20
21 def softmax(inputs, targets, hprev):
22     """
23     inputs - are both lists of integers
24     targets - is a 2D array of probabilities for each state
25     returns the loss, gradients in model parameters, and last hidden state
26     """
27
28     x, h, y, p, o, D, O, hprev = None, None, None, None, None, None, None, None
29
30     if not hprev:
31         hprev = np.zeros((hidden_size, 1))
32
33     for t in range(len(inputs)):
34         x = np.zeros((vocab_size, 1)) + encode_ix(t) # 1-of-1 representation
35         x[inputs[t]] = 1
36
37         h = np.tanh(wih.dot(x) + np.dot(hwh, hprev) + bh) # hidden state
38         y = np.dot(why, h) # by a normalized log probabilities for next chars
39         p = softmax(y) # softmax output
40
41         loss += -np.log(p[targets[t], 0]) # softmax (cross-entropy loss)
42
43         # backward pass: compute gradients going backwards
44         dprev = np.zeros_like(h) # previous hidden state, 0s across i-th
45         dh = np.zeros_like(h) # hidden state, 0s across i-th
46         dy = np.zeros_like(y) # output, 0s across i-th
47         dwhy = np.zeros((vocab_size, 1)) # gradients for why
48         dhwh = np.zeros((hidden_size, hidden_size)) # gradients for whh
49         dwih = np.zeros((vocab_size, hidden_size)) # gradients for wih
50
51         dy[targets[t]] = 1 # 1-hot vector into y
52         dy *= -p / D # dloss/dy = -1/D * softmax(i)
53         dy *= dy # dloss/dy^2 = dy * dy
54         dh = np.dot(why.T, dy) # dhnext = backprop into h
55         dh += np.tanh(h) * (1 - np.tanh(h) * np.tanh(h)) # tanh nonlinearities
56         dprev = np.dot(wih.T, dh) # backprop into previous hidden state
57         dwhh = np.dot(dh, dh.T) # dloss/dwhh = dh * dh
58         dprev = np.dot(dwih.T, dprev) # dloss/dwih = dprev * dwih
59
60         for diaram in [dprev, dh, dwhy, dbh, dy, dhwh, dwih]:
61             if np.isnan(diaram).any():
62                 print("NaN in gradient %s." % diaram)
63
64         if np.isnan(dy).any():
65             print("NaN in dy")
66
67         if np.isnan(dprev).any():
68             print("NaN in dprev")
69
70         if np.isnan(dh).any():
71             print("NaN in dh")
72
73         if np.isnan(dwhy).any():
74             print("NaN in dwhy")
75
76         if np.isnan(dbh).any():
77             print("NaN in dbh")
78
79         if np.isnan(dwhh).any():
80             print("NaN in dwhh")
81
82         if np.isnan(dwih).any():
83             print("NaN in dwih")
84
85         if np.isnan(dprev).any():
86             print("NaN in dprev")
87
88         if np.isnan(dh).any():
89             print("NaN in dh")
90
91         if np.isnan(dwhy).any():
92             print("NaN in dwhy")
93
94         if np.isnan(dbh).any():
95             print("NaN in dbh")
96
97         if np.isnan(dwhh).any():
98             print("NaN in dwhh")
99
100        if np.isnan(dwih).any():
101            print("NaN in dwih")
102
103    smooth_loss = smooth_loss * 0.999 + loss * 0.001
104
105    if n % 100 == 0:
106        print('iter %d, loss: %f' % (n, smooth_loss))
107
108    # perform parameter update with Adagrad
109    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
110                                 [dWxh, dWhh, dWhy, dbh, dyb],
111                                 [mWxh, mWhh, mWhy, mbh, mby]):
112        mem += dparam * dparam
113        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
114
115    # perform parameter update with Adagrad
116    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
117                                 [dWxh, dWhh, dWhy, dbh, dyb],
118                                 [mWxh, mWhh, mWhy, mbh, mby]):
119        mem += dparam * dparam
120        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
121
122    p += seq_length # move data pointer
123    n += 1 # iteration counter
```

# Main loop

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

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

## Main loop

# min-char-rnn.py gist

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  n = 100
8  data = open('input.txt', 'r').read() + ' '
9  chars = list(set(data))
10 vocab_size = len(chars)
11 seq_length = 25
12 learning_rate = 1e-3
13
14 # model parameters
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 wih = np.random.uniform(-0.1, 0.1, (vocab_size, hidden_size)) * hidden_size # input to hidden
21 whh = np.random.uniform(-0.1, 0.1, (hidden_size, hidden_size)) * hidden_size # hidden to hidden
22 why = np.random.uniform(-0.1, 0.1, (hidden_size, vocab_size)) * vocab_size # hidden to output
23 bh = np.zeros((hidden_size, 1)) # hidden bias
24 by = np.zeros((vocab_size, 1)) # output bias
25
26 # forward pass
27 def forward(inputs, targets, hprev):
28     """Compute loss and gradients over a sequence of inputs and targets.
29     Inputs is a list of length seq_length, where each element is a list of integers
30     representing the index of the character in the vocabulary.
31     Targets is a list of length seq_length, where each element is an integer
32     representing the index of the target character for the corresponding input.
33     Returns the loss, gradients in model parameters, and last hidden state
34     """
35     if hprev is None:
36         hprev = np.zeros((hidden_size, 1))
37     else:
38         hprev = np.copy(hprev)
39     loss = 0
40     inputs = inputs[:seq_length]
41     targets = targets[:seq_length]
42
43     for t in range(len(inputs)):
44         i = inputs[t]
45         o = targets[t]
46
47         # forward pass: encode in 1-of-1 representation
48         xi = np.zeros((vocab_size, 1))
49         xi[i] = 1
50
51         hi = np.tanh(wih.dot(xi) + np.dot(hprev, whh) + bh) # hidden state
52         yi = np.dot(why, hi) # by a universalized log probabilities for next chars
53         yi = np.exp(yi) / np.sum(np.exp(yi), axis=0) # softmax
54         loss += -np.log(yi[o]) # softmax cross-entropy loss
55
56         # backward pass: compute gradients going backwards
57         dxi = np.zeros((vocab_size, 1))
58         dxi[i] = 1 # backprop into x
59         dh = np.zeros((hidden_size, 1))
60         dyi = np.zeros((vocab_size, 1))
61         dyi[o] = 1 # backprop into y
62         dwhy = np.zeros((vocab_size, hidden_size))
63         dhnext = np.zeros((hidden_size, 1))
64
65         for di in [dxi, dh, dyi, dwhy]:
66             if di is not dhnext:
67                 di *= 0.5 # subtracts 0.5 to mitigate exploding gradients
68
69         return loss, dh, dxi, dyi, dwhy, hprev[None][inputs[t]]
70
71     mem = np.zeros((hidden_size, 1))
72
73     sample = lambda x: np.argmax(x)
74
75     # sample a sequence from the model
76     if hprev is None:
77         seed_ix = sample(hprev)
78     else:
79         seed_ix = hprev[None][0]
80
81     for t in range(seq_length):
82         x = np.zeros((vocab_size, 1))
83         x[sample(hprev)] = 1
84
85         hprev = forward([seed_ix], [sample(hprev)], hprev)
86
87         seed_ix = sample(hprev)
88
89     return seed_ix
90
91
92 # sample a sequence of integers from the model
93 # h is memory state, seed_ix is seed letter for first time step
94 # inputs is a list of integers
95 # targets is a list of integers
96 # losses is a list of floats
97 # hprev is a hidden state
98
99 # h = np.zeros((hidden_size, 1))
100 h = np.random.uniform(-0.1, 0.1, (hidden_size, 1))
101 y = np.zeros((vocab_size, 1))
102 p = np.zeros((vocab_size, 1))
103
104 x = np.random.choice(vocab_size, np.random.randint(1, 5))
105 x = np.zeros((vocab_size, 1))
106 x[x] = 1
107
108 losses = []
109
110 for i in range(1000):
111     loss, dh, dxi, dyi, dwhy, hprev = forward([x], [sample(h)], hprev)
112
113     losses.append(loss)
114
115     if i % 100 == 0:
116         print('iter %d, loss: %f' % (i, loss))
117
118
119 # perform parameter update with Adagrad
120 for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
121                               [dWxh, dWhh, dWhy, dbh, dby],
122                               [mWxh, mWhh, mWhy, mbh, mby]):
123
124     mem += dparam * dparam
125
126     param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
127
128
129 # for parameter, mem in zip([Wxh, Whh, Why, bh, by],
130 #                           [dWxh, dWhh, dWhy, dbh, dby],
131 #                           [mWxh, mWhh, mWhy, mbh, mby]):
132
133     mem += dparam * dparam
134
135     param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
136
137
138 p += seq_length # move data pointer
139 n += 1 # iteration counter
```

# Main loop

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

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

## Main loop

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

## Main loop

# min-char-rnn.py gist

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  n_data = 100
8  data = open('input.txt', 'r').read() + ' ' # should be simple plain text file
9  chars = list(set(data))
10 vocab_size = len(chars)
11 embed_size = 8
12 hidden_size = 100 # size of hidden layer of neurons
13 seq_length = 25 # number of steps to unroll the RNN for
14 learning_rate = 1e-3
15
16 # model parameters
17 wih = np.random.rand(vocab_size, hidden_size)*np.sqrt(0.01) # input to hidden
18 whh = np.random.rand(hidden_size, hidden_size)*np.sqrt(0.01) # hidden to hidden
19 why = np.random.rand(hidden_size, vocab_size)*np.sqrt(0.01) # hidden to output
20 bh = np.zeros((hidden_size, 1)) # hidden bias
21 by = np.zeros((vocab_size, 1)) # output bias
22
23 # forward pass (compute loss)
24
25 inputs, targets, hprev:
26     """
27     inputs, targets are both lists of integers.
28     hprev is Hx1 array of previous hidden state
29     returns the loss, gradients on model parameters, and last hidden state
30     """
31
32     xs, hs, ys, ps = {}, {}, {}, {}
33     hs[-1] = np.copy(hprev)
34     loss = 0
35
36     for t in xrange(len(inputs)):
37         # forward pass: compute gradients going backwards
38         # inputs[t] is encode in 1-of-k representation
39         # x[t] = np.zeros((vocab_size,1)) + encode in 1-of-k representation
40         # wih[t] = np.dot(np.zeros((vocab_size,1)), x[t-1]) + b[t-1] = hidden state
41         # yh[t] = np.dot(wih[t], x[t]) + by[t] = by + unnormalized log probabilities for next chars
42         # hs[t] = np.tanh(yh[t]) = by + normalized log probabilities for next chars
43         # ys[t] = np.dot(why, hs[t]) + b[t] = softmax (cross-entropy loss)
44
45         # backward pass: compute gradients going backwards
46         # dh[t] = np.zeros_like(x[t]) + zero_grad into x[t]
47         # dy[t] = np.zeros_like(bh[t]) + zero_grad into b[t]
48         # dbh[t] = np.zeros_like(bh[t]) + zero_grad into b[t]
49         # dyh[t] = np.zeros_like(why) + zero_grad into why
50         # dyh[t] = np.dot(why.T, dy[t]) + dh[t] = backprop into h
51         # dwhh[t] = np.dot(dyh[t], x[t-1].T) + dbh[t] = backprop through tanh nonlinearity
52         # dwih[t] = np.dot(dyh[t], np.zeros((vocab_size,1))) + dh[t] = backprop through tanh nonlinearity
53         # dbh[t] = np.zeros_like(bh[t]) + zero_grad into b[t]
54         # dyh[t] = np.dot(why.T, dy[t]) + dh[t] = backprop into h
55         # dwhh[t] = np.dot(dyh[t], x[t-1].T) + dbh[t] = backprop through tanh nonlinearity
56         # dwih[t] = np.dot(dyh[t], np.zeros((vocab_size,1))) + dh[t] = backprop into h
57         # dbh[t] = np.zeros_like(bh[t]) + zero_grad into b[t]
58
59         # clip to mitigate exploding gradients
60         np.clip(dwparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61
62     return loss, dwih, dwhh, dwhy, dbh, dy, hs[len(inputs)-1]
```

# Loss function

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



```
def lossFun(inputs, targets, hprev):
    """
    inputs,targets are both list of integers.
    hprev is Hx1 array of initial hidden state
    returns the loss, gradients on model parameters, and last hidden state
    """
    xs, hs, ys, ps = {}, {}, {}, {}
    hs[-1] = np.copy(hprev)
    loss = 0
    # forward pass
    for t in xrange(len(inputs)):
        xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
        xs[t][inputs[t]] = 1
        hs[t] = np.tanh(np.dot(wxh, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
        ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
        ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
        loss += -np.log(ps[t])[targets[t],0] # softmax (cross-entropy loss)

    # backward pass: compute gradients going backwards
    dwxh, dwhh, dwhy = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(why)
    dbh, dyb = np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])
    for t in reversed(xrange(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] -= 1 # backprop into y
        dwhy += np.dot(dy, hs[t].T)
        dwxh += np.dot(dy, xs[t].T)
        dwhh += np.dot(dy, hs[t-1].T)
        dbh += dyb
        dh = np.dot(why.T, dy) + dhnext # backprop into h
        ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
        dh += ddraw
        dwxh += np.dot(ddraw, xs[t].T)
        dwhh += np.dot(ddraw, hs[t-1].T)
        dhnext = np.dot(whh.T, ddraw)
    # clip to mitigate exploding gradients
    for dwparam in [dwxh, dwhh, dwhy, dbh, dyb]:
        np.clip(dwparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
    return loss, dwxh, dwhh, dwhy, dbh, dyb, hs[len(inputs)-1]
```

# min-char-rnn.py gist

```

1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  AGPL license
4
5  imports
6  import numpy as np
7
8  # data I/O
9  data = open('input.txt', 'r').read() # should be simple plain text file
10 chars = list(set(data))
11 vocab_size = len(chars)
12 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
13 char_to_ix = { c: i for i, c in enumerate(chars) }
14 ix_to_char = { i: c for i, c in enumerate(chars) }
15
16 # hyperparameters
17 hidden_size = 100 # size of hidden layer of neurons
18 seq_length = 25 # number of steps to unroll the RNN for
19 learning_rate = 1e-3
20
21 # model parameters
22 wih = np.random.rand(hidden_size, vocab_size)*0.01 # input to hidden
23 whh = np.random.rand(hidden_size, hidden_size)*0.01 # hidden to hidden
24 why = np.random.rand(vocab_size, hidden_size)*0.01 # hidden to output
25 bi = np.zeros((hidden_size, 1)) # hidden bias
26 bo = np.zeros((vocab_size, 1)) # output bias
27
28 def lossFun(inputs, targets, hprev):
29     """ 
30     inputs,targets are both list of integers.
31     hprev is Hx1 array of initial hidden state
32     returns the loss, gradients on model parameters, and last hidden state
33     """
34
35     xs, hs, ys, ps = {}, {}, {}, {}
36     hs[-1] = np.copy(hprev)
37     loss = 0
38     # forward pass
39     for t in xrange(len(inputs)):
40         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
41         xs[t][inputs[t]] = 1
42         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
43         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
44         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
45         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
46
47     # backward pass: compute gradients going backwards
48     dhs = {} # delta-h
49     dxs = {} # delta-x
50     dWxh, dWhh, dWyy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
51     dbh, dby, dWx0 = np.zeros_like(bh), np.zeros_like(by), np.zeros_like(inputs[0])
52     dh0 = np.zeros_like(h0)
53
54     for t in reversed(xrange(len(inputs))):
55         dy = np.copy(ps[t])
56         dy[targets[t]] += 1 # backprop into y
57         dy = np.dot(why.T, dy) # backprop into h
58         dh = np.dot(Wxh.T, dy) # backprop into x
59         dxs[t] = np.dot(Wxh, dh) # backprop into x
60         dWxh += np.outer(dxs[t], inputs[t].T)
61         dbx = np.sum(dxs[t])
62         dWx0 += dbx * inputs[0]
63         dWyy += np.outer(dh, dy)
64         dby += dbx
65         dWxh += np.outer(dxs[t], dh)
66         dbh += dbx
67         dWhh += np.outer(dh, dh)
68         dby += dbx
69     return loss, dxs, dbh, dby, dWxh, dWyy, dWx0
70
71 def sample(h0, seed_ix, n):
72     """ 
73     sample a sequence from the model
74     h0 is memory state, seed_ix is seed letter for first time step
75     """
76     if type(seed_ix) == int:
77         seed_ix = [seed_ix]
78     h, xs, ys = [], []
79     for t in range(n):
80         x = np.zeros((vocab_size, 1))
81         x[seed_ix] = 1
82         h, dx = rnn_step_forward(x, h0)
83         y = np.dot(why, h) + by
84         p = np.exp(y) / np.sum(np.exp(y))
85         ix = np.random.choice(range(vocab_size), p=p.ravel())
86         x[0][ix] = 1
87         seed_ix = ix
88     return np.append(seed_ix, ix)
89
90
91 n, p = 0, 0
92
93 mem, mbt, mbty = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
94 dWxh, dWhh, dWyy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
95 dbh, dby, dWx0 = np.zeros_like(bh), np.zeros_like(by), np.zeros_like(inputs[0])
96 smooth_loss = -np.log(1.0/vocab_size)*seq_length + 1.0 # loss at iteration 0
97 while True:
98     if n % seq_length == 0 or n == 0:
99         inputs, targets = get_data(data, seq_length)
100         if seq_length == len(data) or n == 0:
101             np.random.seed(12345)
102             inputs, targets = get_data(data, seq_length)
103             n = 0
104         else:
105             inputs = inputs[1:-1]
106             targets = targets[1:-1]
107             n += 1
108
109     # sample from the model now and then
110     if n % 100 == 0:
111         sample(h0, seed_ix, 200)
112         print '-'*80
113         print 'n %d' % n
114         print 'smooth loss: %f' % smooth_loss
115         print 'mbt: %f' % mbt
116         print 'mbty: %f' % mbty
117         print 'dWxh: %f' % dWxh
118         print 'dWhh: %f' % dWhh
119         print 'dWyy: %f' % dWyy
120         print 'dbh: %f' % dbh
121         print 'dby: %f' % dby
122         print 'dWx0: %f' % dWx0
123
124     # perform parameter update with Adam
125     for param, variance, mean in zip([mem, mbt, mbty, dWxh, dWhh, dWyy],
126                                     [mem_sq, mbt_sq, mbty_sq, dWxh_sq, dWhh_sq, dWyy_sq],
127                                     [mem_mean, mbt_mean, mbty_mean, dWxh_mean, dWhh_mean, dWyy_mean]):
128         mean_sq = variance * 0.9 + mean * 0.1
129         variance = variance * 0.999 + 0.001
130         mean = mean * 0.9 + mean_sq * 0.1
131
132         # for parameter update
133         if variance > 0:
134             param -= learning_rate * mean / (variance**0.5 + 1e-8)
135
136     # for learning rate decay
137     parame = learning_rate * learning_rate * seq_length * 3e-4
138
139     # for data pointer
140     p += seq_length
141
142     # for iteration counter
143     n += 1

```

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

$$y_t = W_{hy}h_t$$

Softmax classifier

# min-char-rnn.py gist

```

1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  CC-BY-NC-ND license
4  """
5  import numpy as np
6
7  # data I/O
8  data = open('input.txt', 'r').read() + ' '
9  chars = list(set(data))
10 vocab_size = len(chars)
11 embed_size = 100
12 hidden_size = 128
13 seq_length = 25
14 learning_rate = 1e-3
15
16 # model parameters
17 wih = np.random.rand(vocab_size, hidden_size)*0.01 # input to hidden
18 whh = np.random.rand(hidden_size, hidden_size)*0.01 # hidden to hidden
19 why = np.random.rand(hidden_size, vocab_size)*0.01 # hidden to output
20 bi = np.zeros((hidden_size, 1)) # hidden bias
21 bo = np.zeros((vocab_size, 1)) # output bias
22
23 # forward pass
24 inputs, targets, hprev:
25     """
26     inputs, targets are both lists of integers
27     hprev is previous hidden state
28     returns the loss, gradients in model parameters, and last hidden state
29     """
30     x, h, y, px, ps, hprev = 0, 0, 0, 0, 0
31     loss = 0
32
33     for t in xrange(len(inputs)):
34         x = np.zeros((embed_size, 1)) + encode_in_1_of_k_representation(
35             inputs[t], vocab_size, 1) # encode in 1-of-k representation
36         h = hprev if hprev is not None else np.zeros((hidden_size, 1))
37         hprev = h
38         h = np.tanh(np.dot(wih, x) + np.dot(whh, h) + bi)
39         y = np.dot(why, h) # by a normalized log probabilities for next chars
40         ps = np.exp(y) / np.sum(np.exp(y)) # softmax
41         loss += -np.log(ps[targets[t], 0]) # softmax (cross-entropy loss)
42         px = np.copy(ps) # probabilities for next chars
43         px[targets[t]] += 1 # backprop into y
44         dy = np.copy(px) # backprop into y
45         dy -= ps[targets[t]] # backprop into y
46         dh = np.dot(why.T, dy) # dhnext = backprop through tanh nonlinearity
47         dhraw = (1 - h * h) * dh # backprop through tanh nonlinearity
48         dhraw = np.dot(whh.T, dhraw)
49         dhraw = np.dot(wih.T, dhraw)
50         dwhh = np.dot(dhraw, h.T)
51         dwih = np.dot(dhraw, x.T)
52         dbi = np.sum(dhraw, axis=0)
53         dwhy = np.dot(dhraw, np.eye(vocab_size))
54         dwhy[targets[t]] += 1 # backprop into y
55         dwhy -= ps[targets[t]] # backprop into y
56         dwhy = np.dot(why.T, dwhy) # backprop through tanh nonlinearity
57         dwhy = (1 - h * h) * dwhy # backprop through tanh nonlinearity
58         dwhy = np.dot(why.T, dwhy)
59         dwhy = np.sum(dwhy, axis=1) # sum over hidden states
60         dwhy = np.sum(dwhy, axis=0) # average exploding gradients
61         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
62
63     return loss, dwih, dwhh, dwhy, dbi, bo, hprev(inputs[-1])
64
65 smooth_loss, n = 0, 0
66
67 sample = lambda: np.random.randint(len(chars))
68
69 # sample a sequence of integers from the model
70 # n is memory state, seed_ix is seed letter for first time step
71 # x is np.zeros((embed_size, 1))
72 # hprev is np.zeros((hidden_size, 1))
73 # losses is []
74
75 def sample_sequence(model):
76     h = np.zeros((hidden_size, 1))
77     x = np.zeros((embed_size, 1))
78     y = np.zeros((vocab_size, 1))
79     px = np.zeros((vocab_size, 1))
80     losses = []
81     smooth_loss = 0
82     n = 0
83     smooth_loss = np.log(embed_size)/seq_length if seq_length > 0 else 0
84     while True:
85         if n == seq_length:
86             hprev = h
87             break
88         if n < seq_length-1:
89             # sample next char from softmax
90             x = np.zeros((embed_size, 1))
91             x[sample()] = 1
92             x = np.reshape(x, (embed_size, 1))
93             h = model(dwih, dwhh, dwhy, dbi, bo, hprev)
94             hprev = h
95             losses.append(-np.log(px[sample()]))
96             smooth_loss += -np.log(px[sample()])
97             n += 1
98         else:
99             hprev = h
100            h = model(dwih, dwhh, dwhy, dbi, bo, hprev)
101            hprev = h
102            losses.append(-np.log(px[sample()]))
103            smooth_loss += -np.log(px[sample()])
104            n += 1
105
106    # average loss over all samples
107    smooth_loss /= n
108    print('smooth loss: %f' % smooth_loss)
109    return hprev, losses
110
111 for para, dparam in zip(locals().values(), locals().values()):
112     para += dparam
113
114 for para in [dwih, dwhh, dwhy, dbi, bo]:
115     para *= learning_rate
116
117 for para in [dwih, dwhh, dwhy, dbi, bo]:
118     para -= learning_rate * para / np.sqrt(para**2 + 1e-05)
119
120 p = np.argmax(px)
121 m = np.argmax(dy)
122
123 if m == p:
124     print('...%s' % chars[p])
125 else:
126     print('...%s vs %s' % (chars[m], chars[p]))
127
128 # forward pass: compute gradients going backwards
129 dwxh, dwhh, dwhy = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(why)
130 dbh, dby = np.zeros_like(bh), np.zeros_like(by)
131 dhnext = np.zeros_like(hs[0])
132
133 for t in reversed(xrange(len(inputs))):
134     dy = np.copy(ps[t])
135     dy[targets[t]] -= 1 # backprop into y
136     dwhy += np.dot(dy, hs[t].T)
137     dby += dy
138     dh = np.dot(why.T, dy) + dhnext # backprop into h
139     ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
140     dbh += ddraw
141     dwxh += np.dot(ddraw, xs[t].T)
142     dwhh += np.dot(ddraw, hs[t-1].T)
143     dhnext = np.dot(whh.T, ddraw)
144
145 for dparam in [dwxh, dwhh, dwhy, dbh, dby]:
146     np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
147
148 return loss, dwxh, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]

```

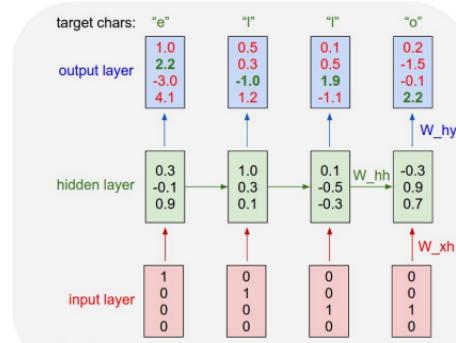


```

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

```

recall:



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

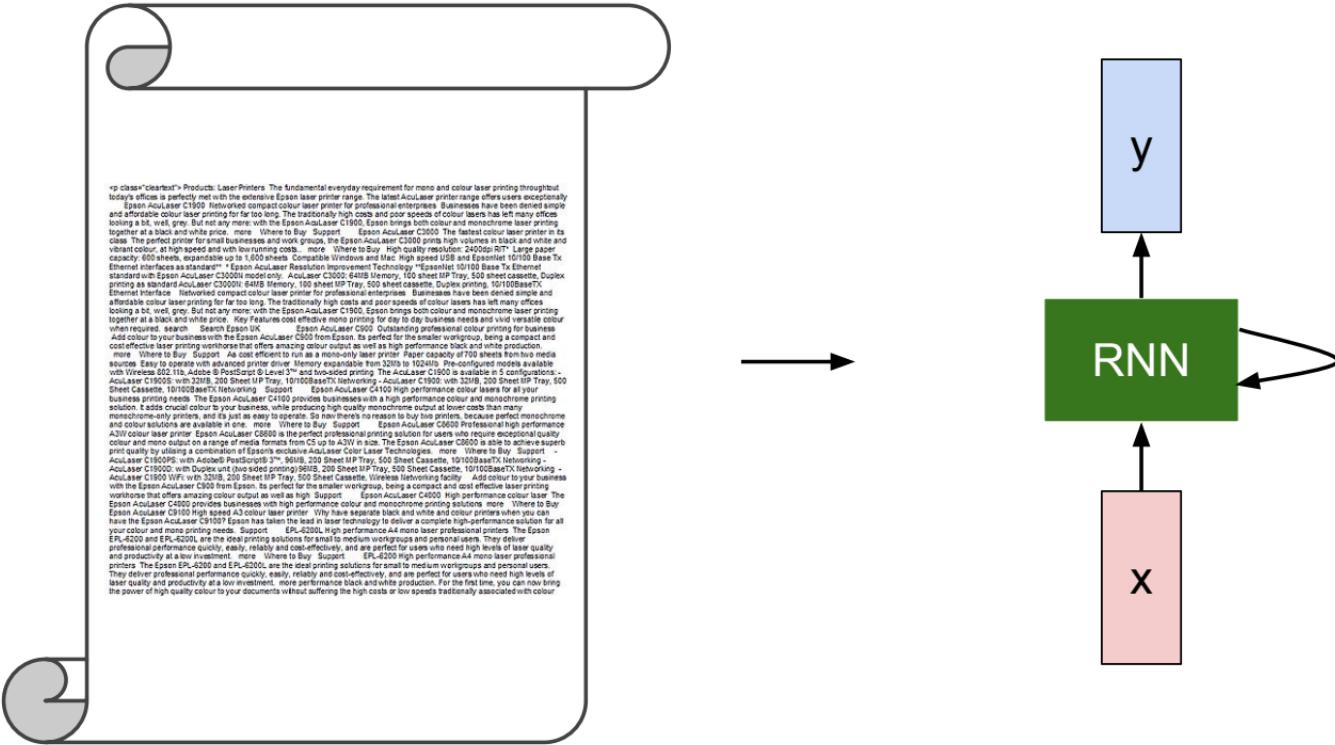
```

and construct, which takes a list of integers from the node.
    It is memory static, saved, so it is used letter for first time step
    and then reused for all other steps.

    x = np.zeros((node_size, 11))
    x[0] = 1
    items = []
    for i in range(1, node_size):
        x[i] = self.get_node(i, node)
        y = np.dot(node_w, x) + b0
        y = np.max(y, 0)
        p = np.exp(y) / np.sum(np.exp(y))
        ix = np.random.choice(np.arange(node_size), p=p, replace=True)
        items.append(ix)
        x[i] = 1
        item.append(ix)
    return items, item

```

```
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
```



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

↓ train more

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

↓ train more

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

↓ train more

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

PANDARUS:

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

Second Senator:

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

DUKE VINCENTIO:

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

Second Lord:

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

Clown:

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

VIOLA:

I'll drink it.

VIOLA:

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

KING LEAR:

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

# open source textbook on algebraic geometry

The Stacks Project

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

Browse chapters

Part	Chapter	online	TeX source	view pdf
Preliminaries	1. Introduction	<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

1. [Preliminaries](#)
2. [Schemes](#)
3. [Topics in Scheme Theory](#)
4. [Algebraic Spaces](#)
5. [Topics in Geometry](#)
6. [Deformation Theory](#)
7. [Algebraic Stacks](#)
8. [Miscellany](#)

Statistics

The Stacks project now consists of

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

Latex source

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

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

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

and the comparicoly in the fibre product covering we have to prove the lemma generated by  $\coprod Z \times_U U \rightarrow V$ . Consider the maps  $M$  along the set of points  $\mathcal{Sch}_{fppf}$  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  $\mathcal{Sh}(G)$  such that  $\text{Spec}(R') \rightarrow S$  is smooth or an

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

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

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

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

is a unique morphism of algebraic stacks. Note that

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

and

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

*Proof.* Omitted.  $\square$

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

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

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

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

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

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

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

*Proof.* See Spaces, Lemma ??.

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

The following to the construction of the lemma follows.

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

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

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

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

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

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

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

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

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

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

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

*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”

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \dashrightarrow (\mathcal{O}_{X_{\text{étale}}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1} \mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\text{pt}})$$

is an isomorphism of covering of  $\mathcal{O}_{X_\ell}$ . 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.  $\square$

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

[This repository](#) [Search](#)[Explore](#) [Gist](#) [Blog](#) [Help](#)

karpathy

[torvalds / linux](#)[Watch](#) 3,711[Star](#) 23,054[Fork](#) 9,141

## Linux kernel source tree

520,037 commits

1 branch

420 releases

5,039 contributors



branch: master

[linux](#) / +

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

torvalds authored 9 hours ago

latest commit 4b1706927d



	Documentation	Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending	6 days ago
	arch	Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l...	a day ago
	block	block: discard bdi_unregister() in favour of bdi_destroy()	9 days ago
	crypto	Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6	10 days ago
	drivers	Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux	9 hours ago
	firmware	firmware/hex2fw.c: restore missing default in switch statement	2 months ago
	fs	vfs: read file_handle only once in handle_to_path	4 days ago
	include	Merge branch 'perf-urgent-for-linus' of git://git.kernel.org/pub/scm/...	a day ago
	init	init: fix regression by supporting devices with major:minor:offset fo...	a month ago
	io	drivers/block: Fix bug in sdhci-udc kernel module from Linux 4.10.2	6 months ago

[Code](#)[Pull requests](#) 74[Pulse](#)[Graphs](#)

### HTTPS clone URL

<https://github.com/torvalds/linux>You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).[Clone in Desktop](#)[Download ZIP](#)

```
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << 1))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffff8) & 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, &offset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

# Generated C code

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

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

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

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

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

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

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

static void
os_prefix(unsigned long sys)
{
#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.
    */
}
```

# Searching for interpretable cells

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

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

quote detection cell

# Searching for interpretable cells

Cell sensitive to position in line:

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

line length tracking cell

# Searching for interpretable cells

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

if statement cell

# Searching for interpretable cells

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

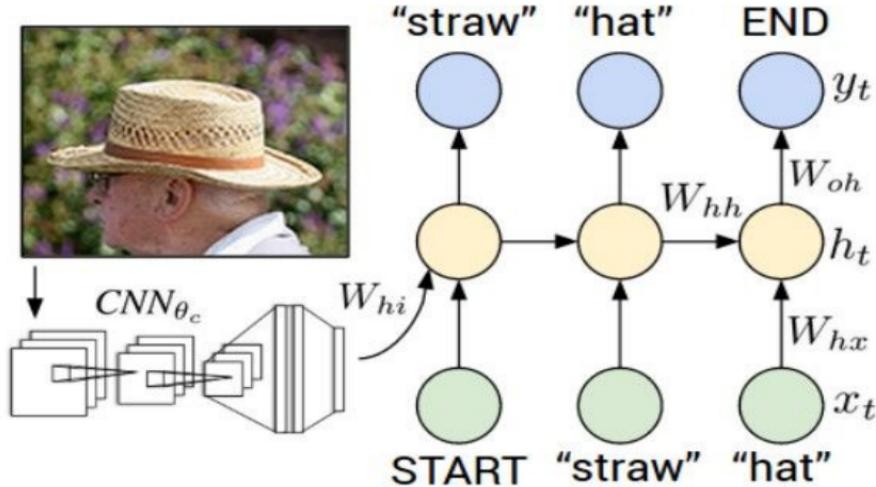
quote/comment cell

# Searching for interpretable cells

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

code depth cell

# Image Captioning



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

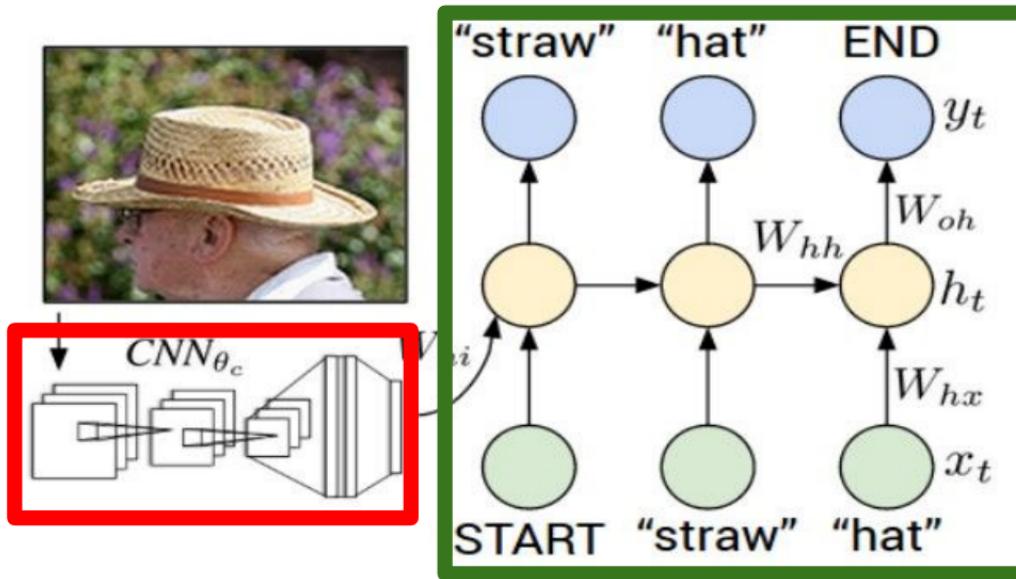
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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

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

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

# Recurrent Neural Network



## Convolutional Neural Network

test image



image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax

image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

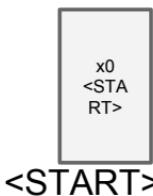
FC-1000

softmax

X

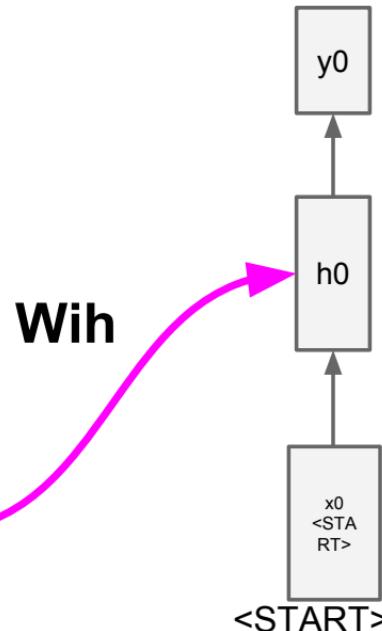


test image





test image



**before:**

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

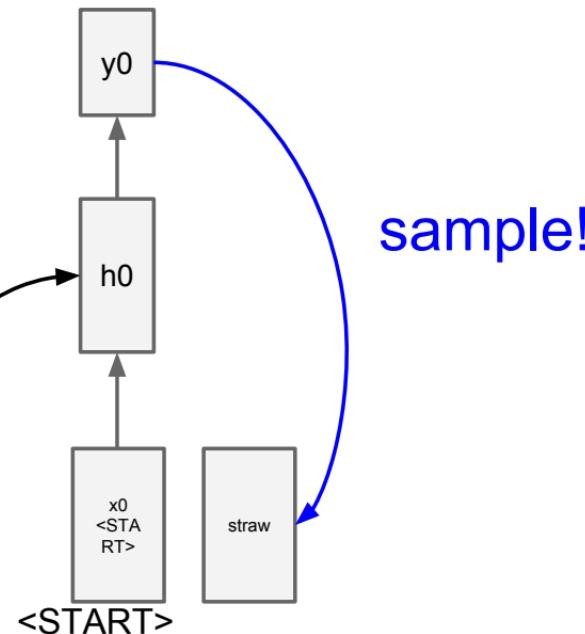
**now:**

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

$v$

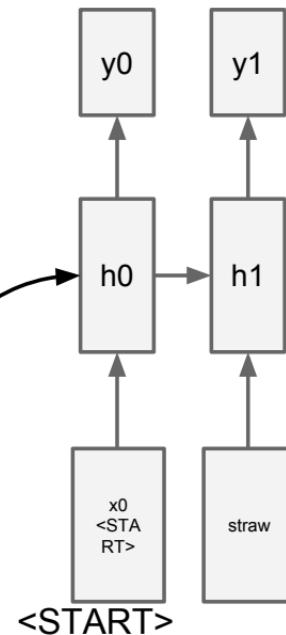


test image



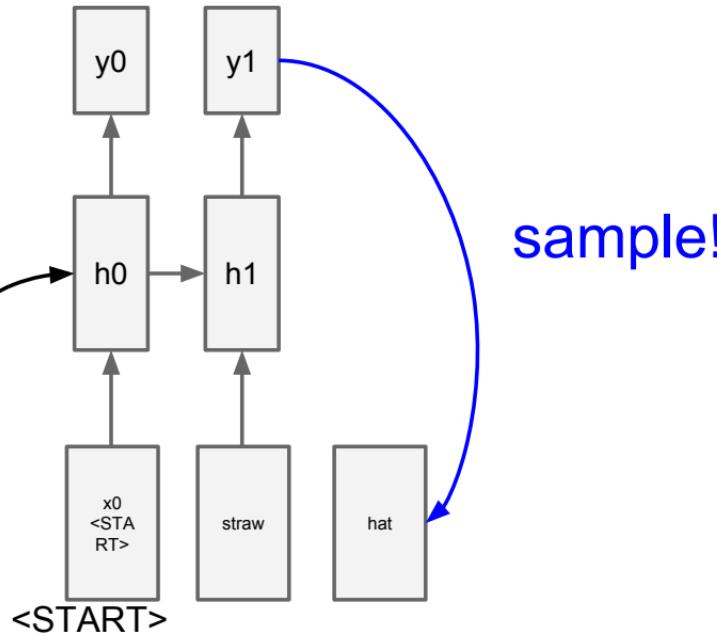


test image





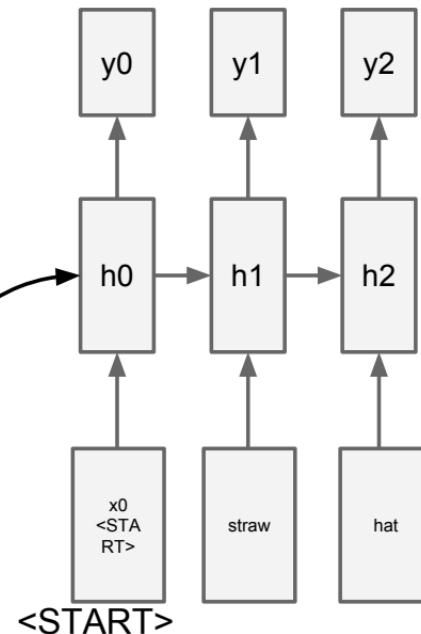
test image



sample!



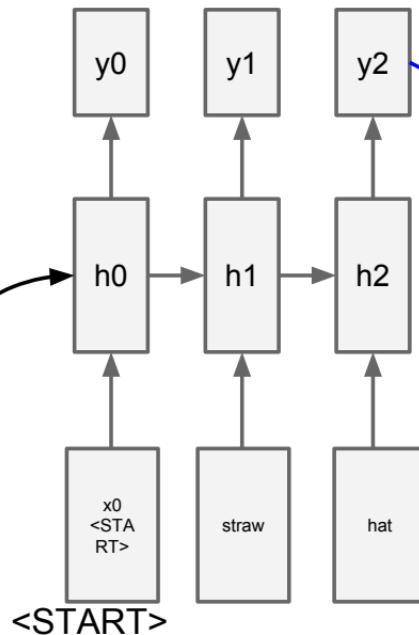
test image





test image

sample  
<END> token  
=> finish.



# Image Sentence Datasets

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



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

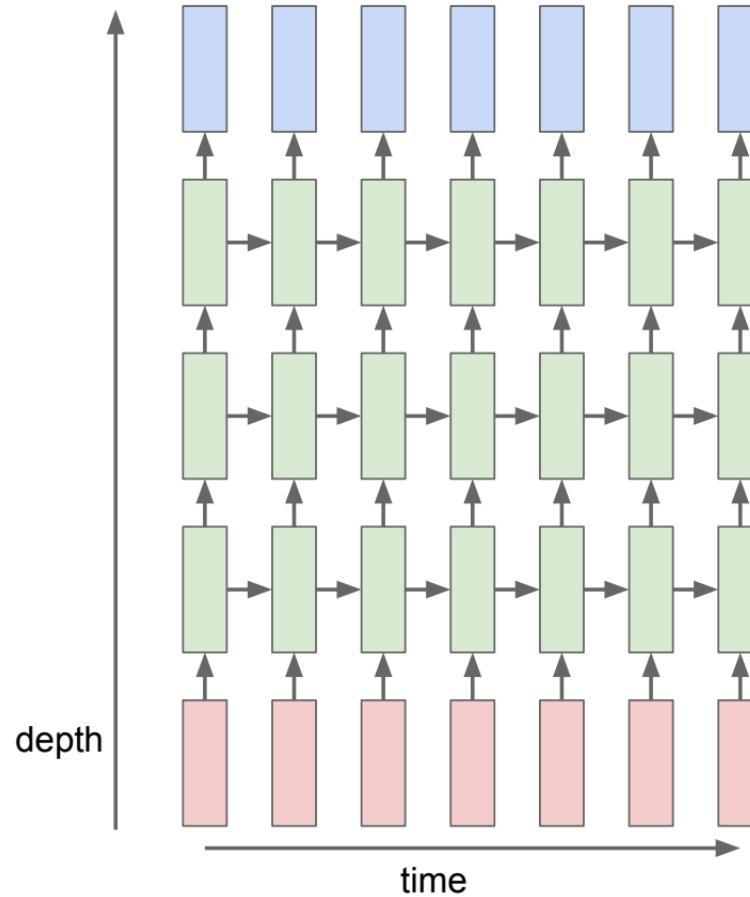


"a horse is standing in the middle of a road."

# RNN:

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

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



## RNN:

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

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

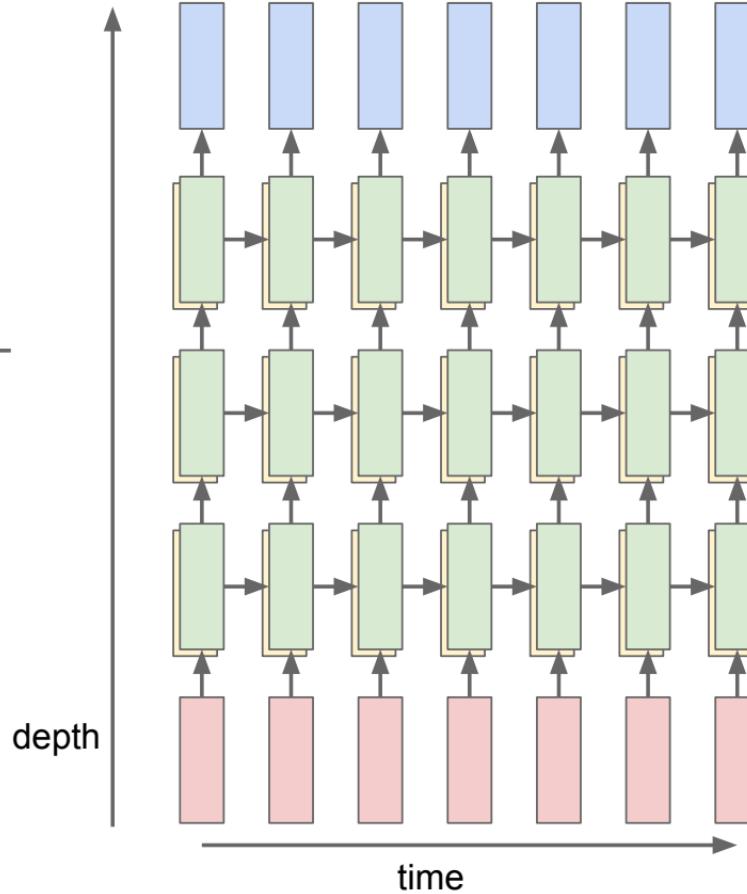
## LSTM:

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

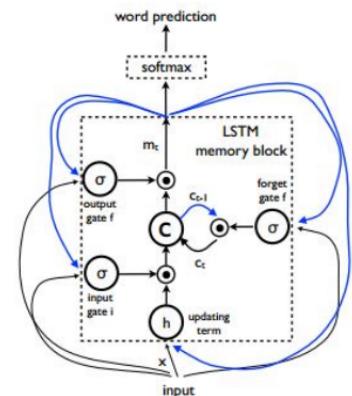
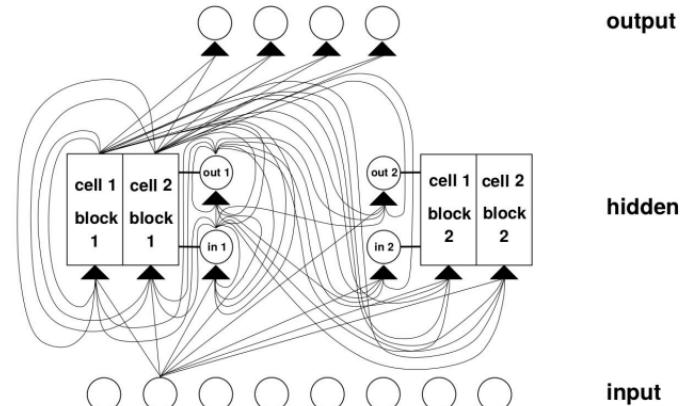
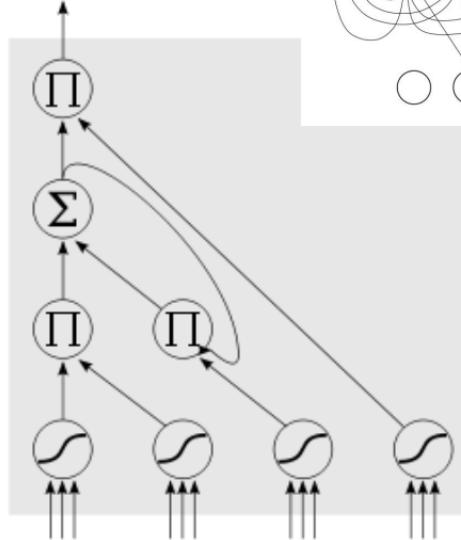
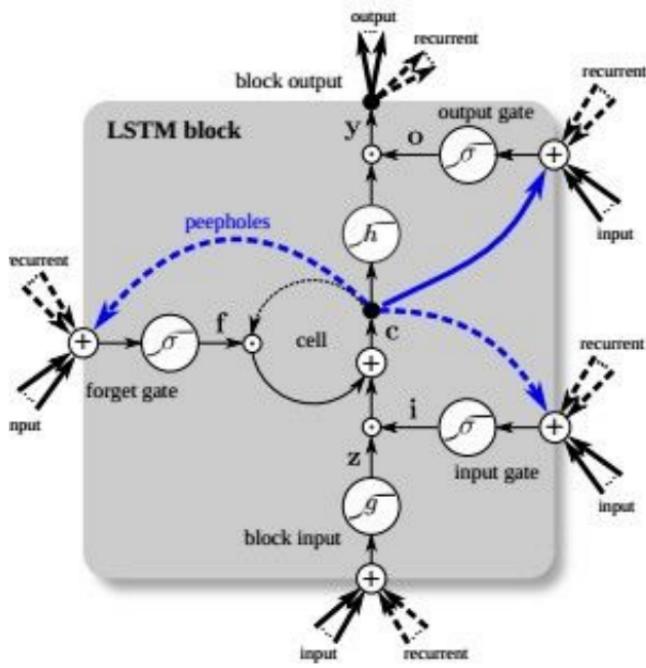
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

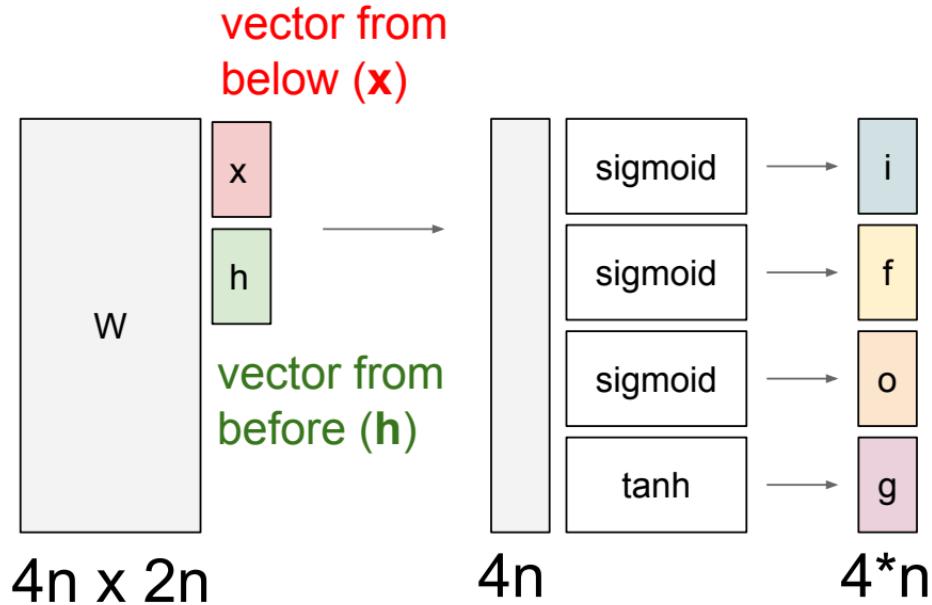


# LSTM



# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

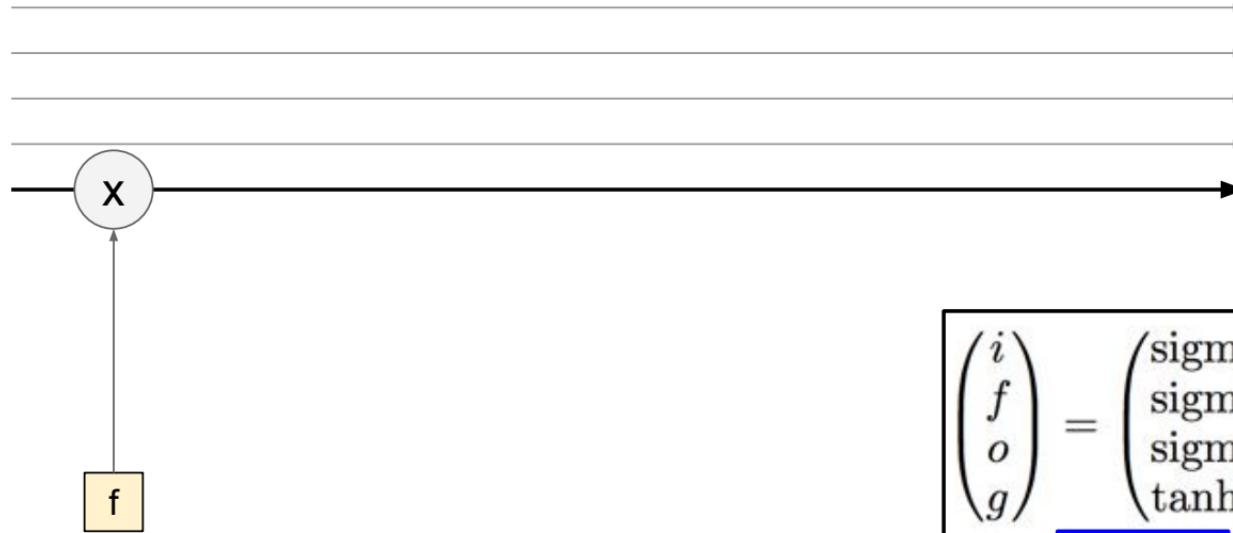


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell  
state  $c$

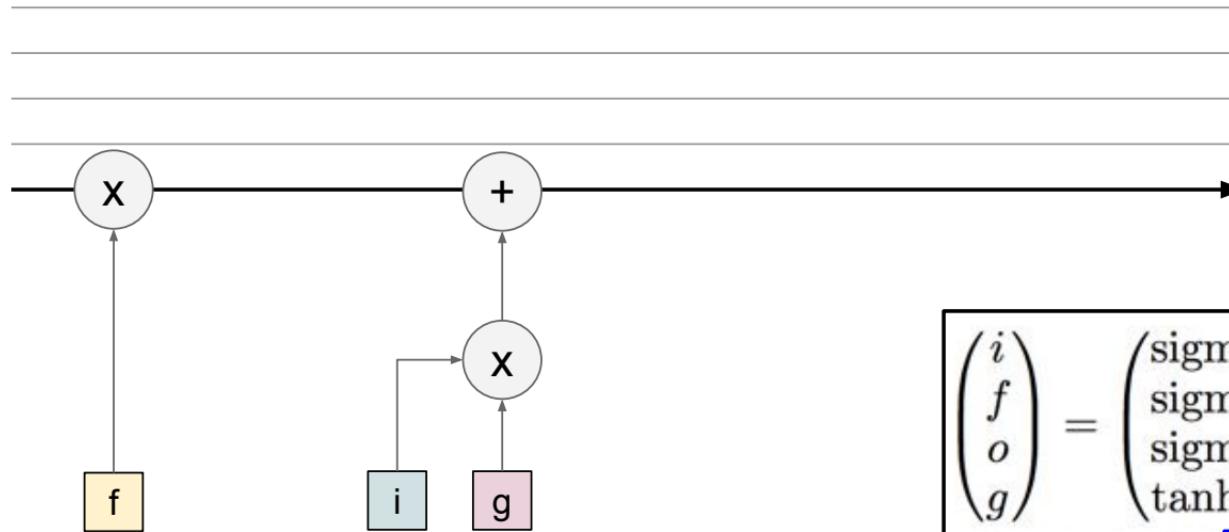


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell  
state  $c$

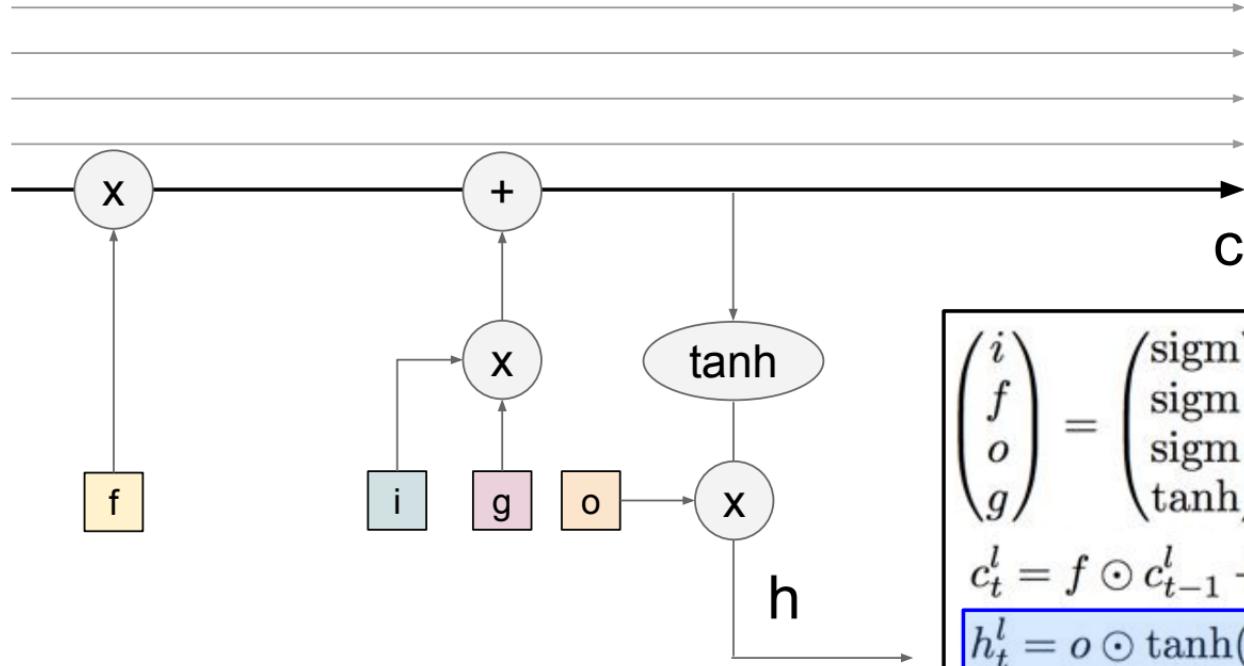


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell  
state  $c$



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

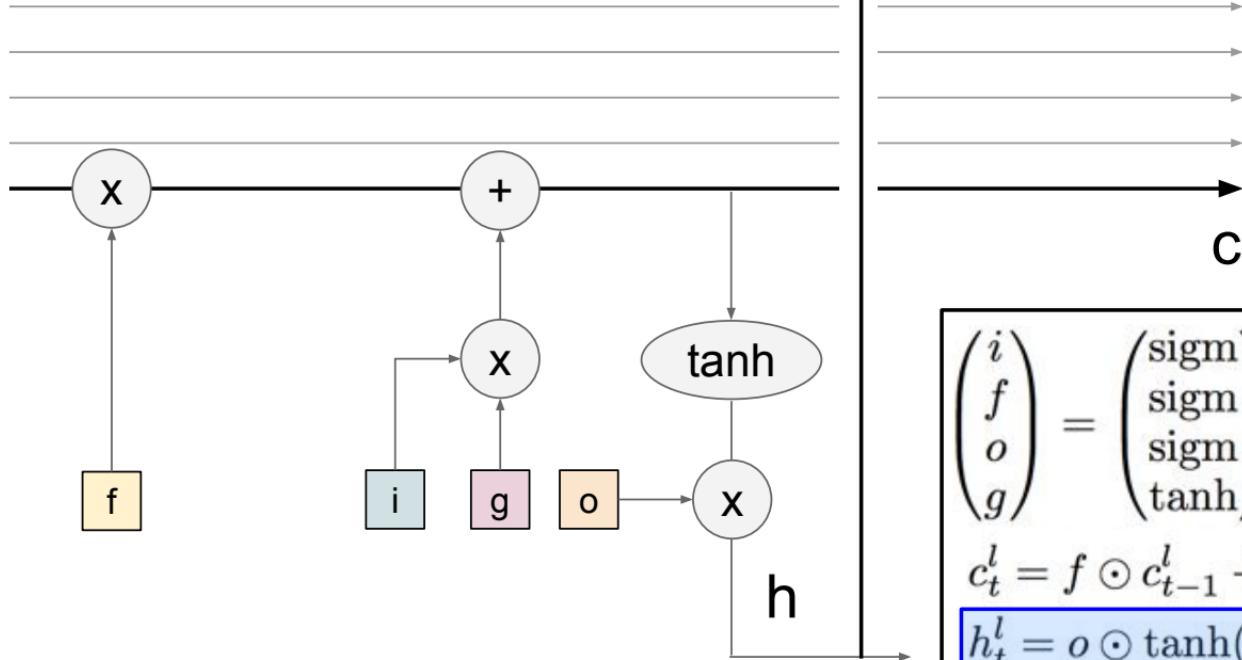
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

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

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

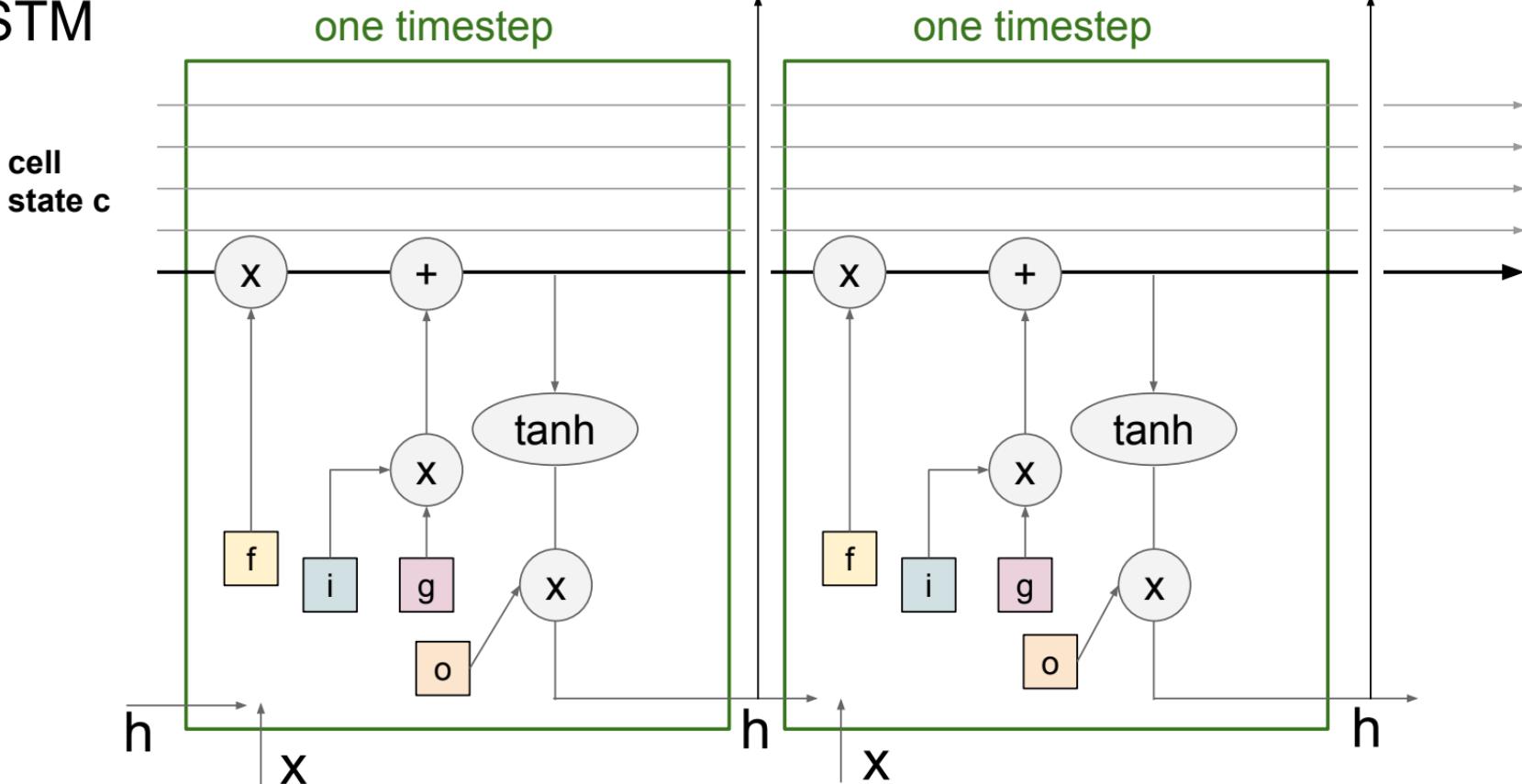
cell  
state  $c$



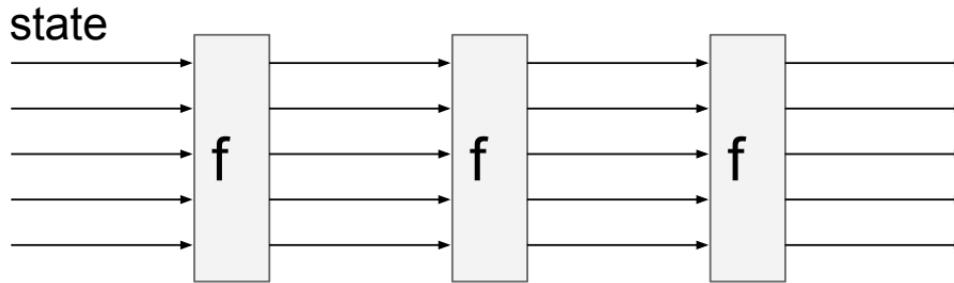
higher layer, or  
prediction

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

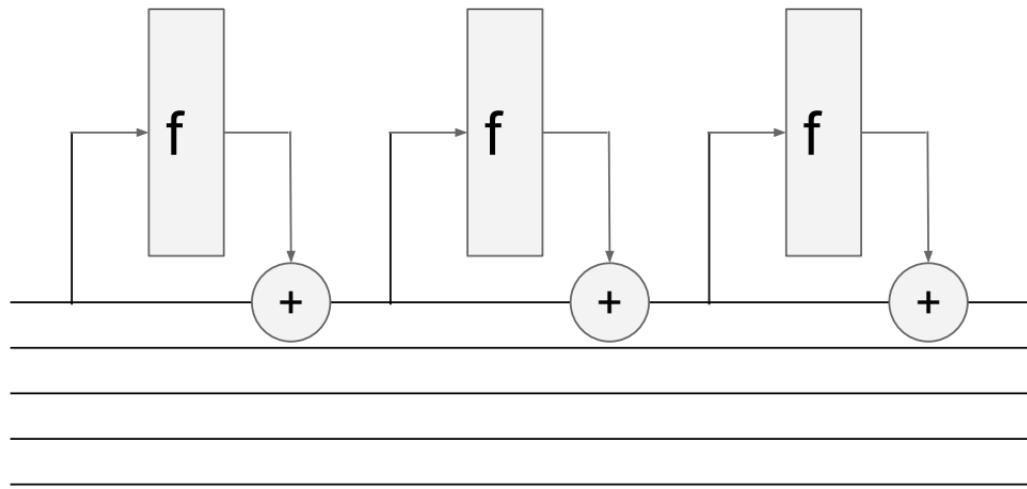
# LSTM



# RNN

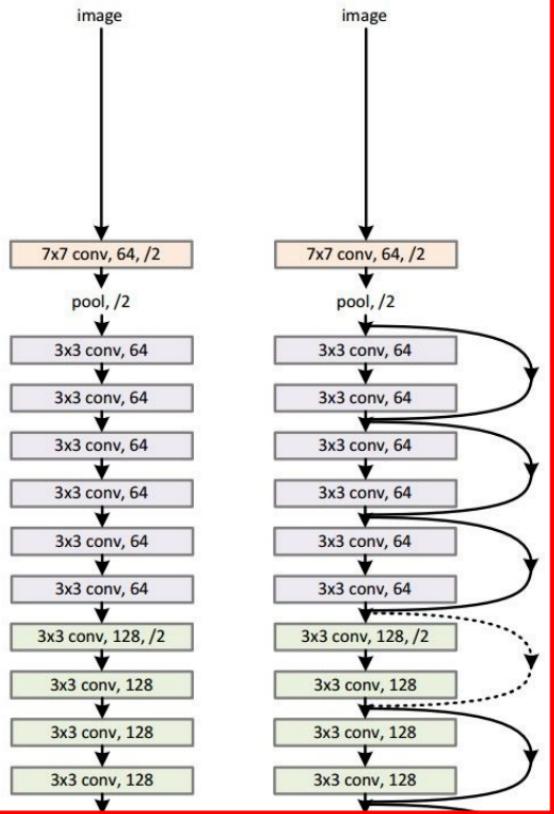


# LSTM (ignoring forget gates)



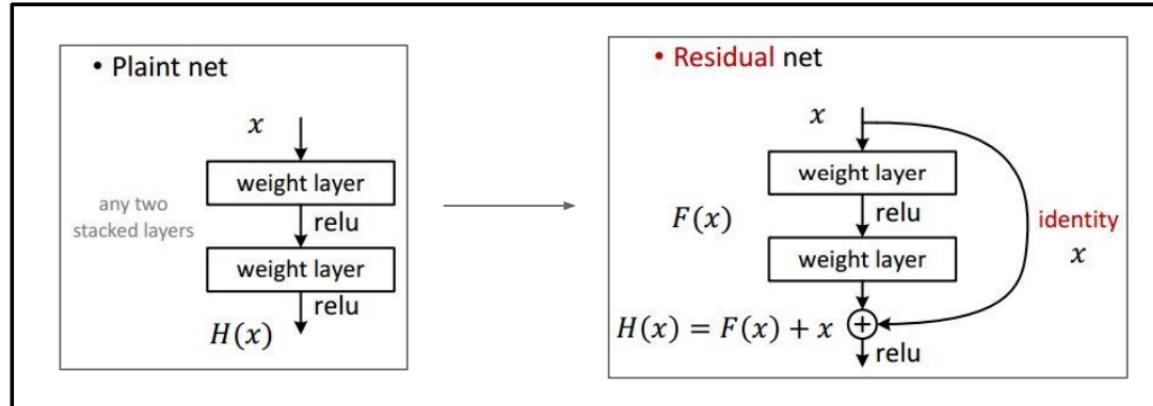
34-layer plain

34-layer residual



## Recall: “PlainNets” vs. ResNets

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



# Understanding gradient flow dynamics

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

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

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

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

# Understanding gradient flow dynamics

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

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

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

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

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

# Understanding gradient flow dynamics

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

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

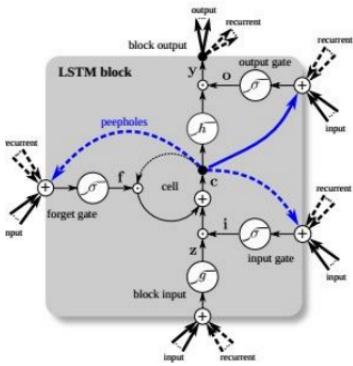
# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

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

can control exploding with gradient clipping  
can control vanishing with LSTM

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

# LSTM variants and friends



[*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]

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

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

MUT1:

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

MUT2:

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

MUT3:

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

# Summary

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