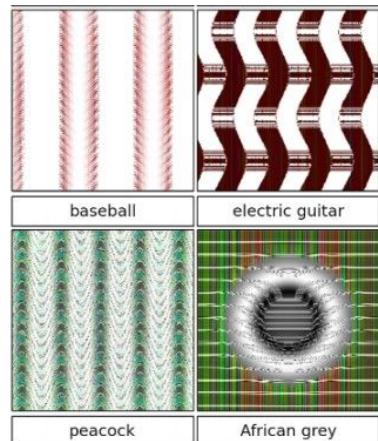
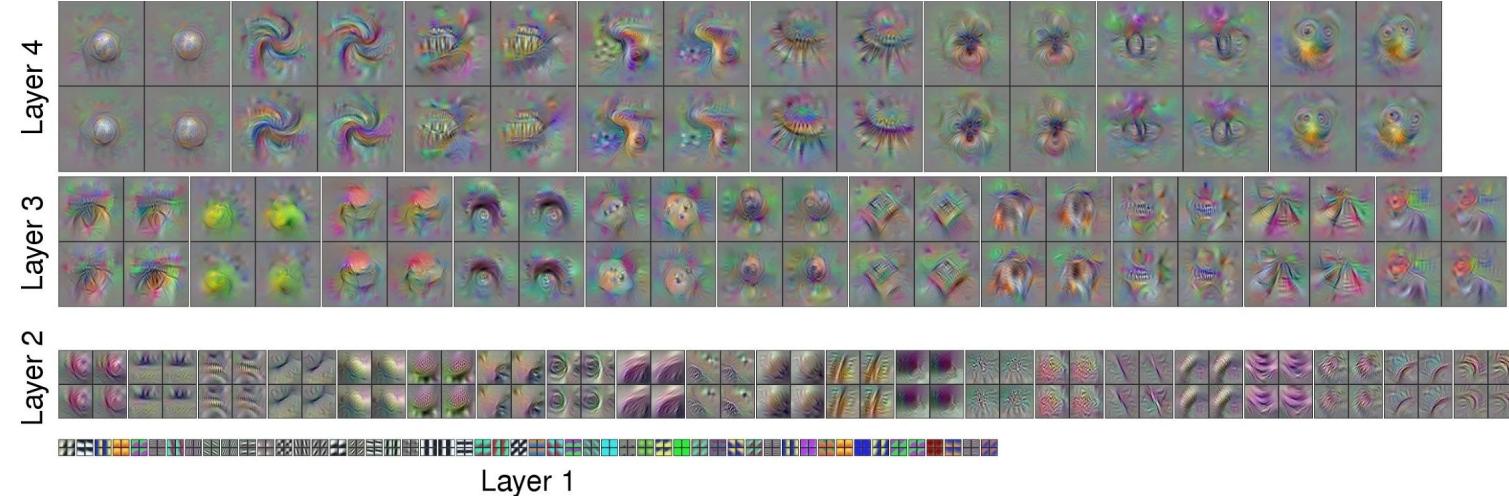
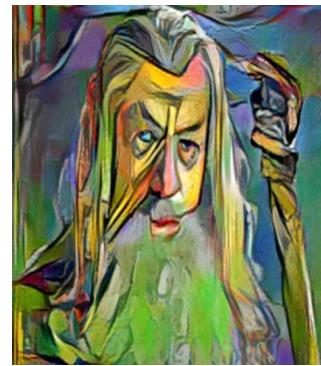
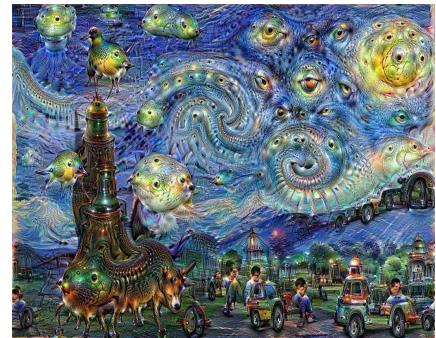
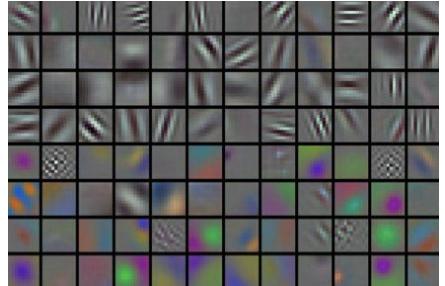


Lecture 10:

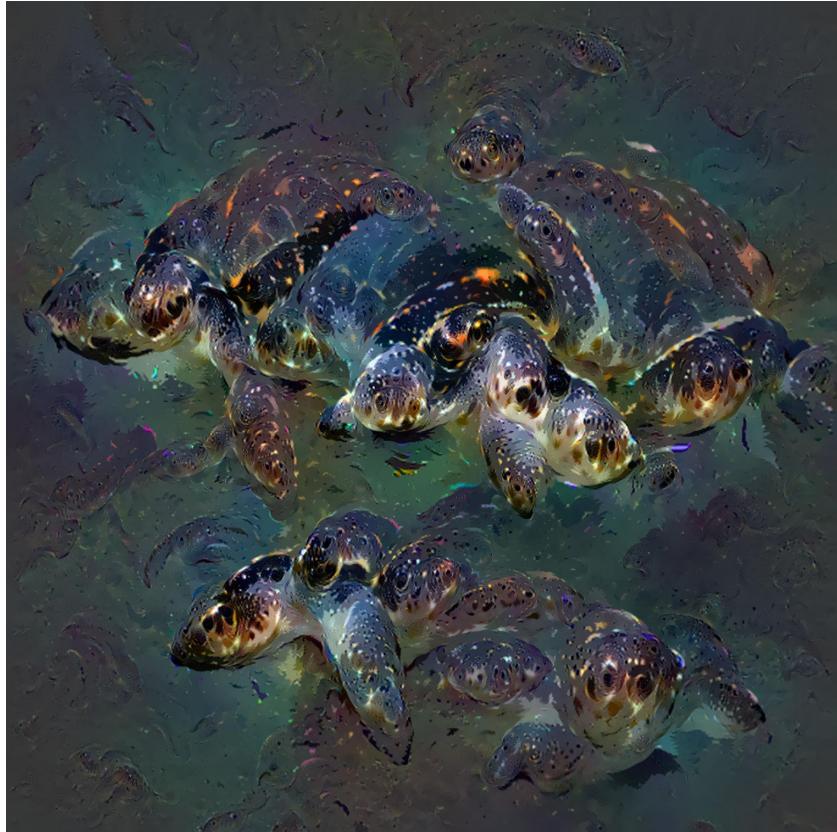
Recurrent Neural Networks

Administrative

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



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

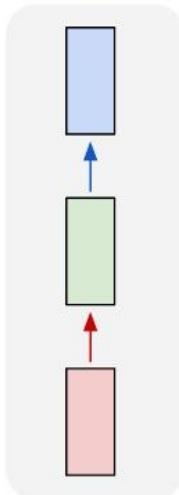


<http://mtyka.github.io/deepdream/2016/02/05/bilateral-class-vis.html>

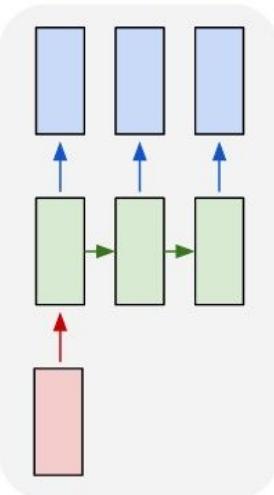


Recurrent Networks offer a lot of flexibility:

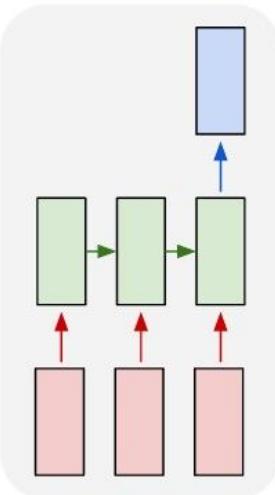
one to one



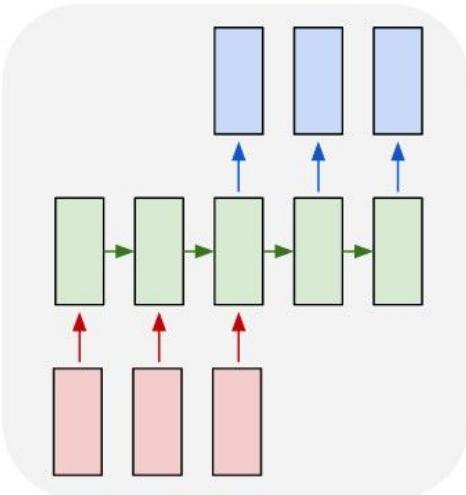
one to many



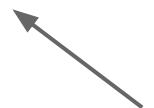
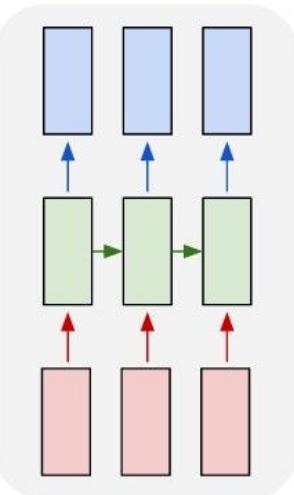
many to one



many to many



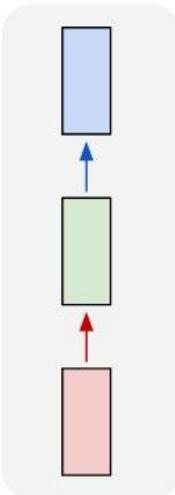
many to many



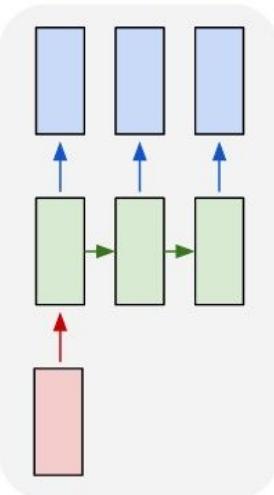
Vanilla Neural Networks

Recurrent Networks offer a lot of flexibility:

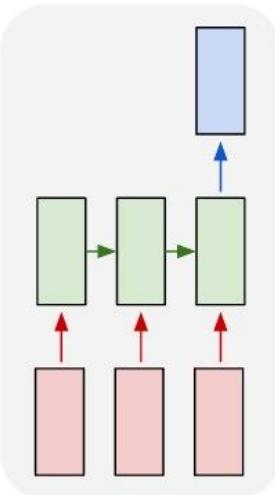
one to one



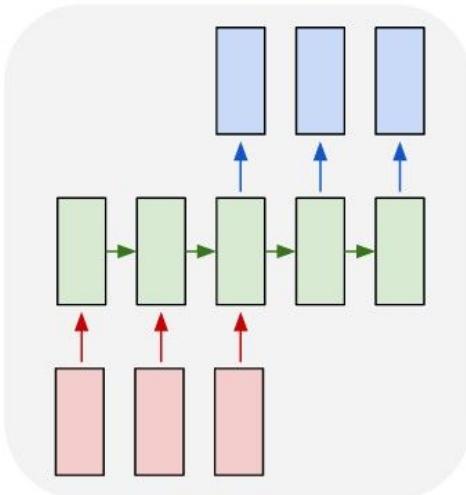
one to many



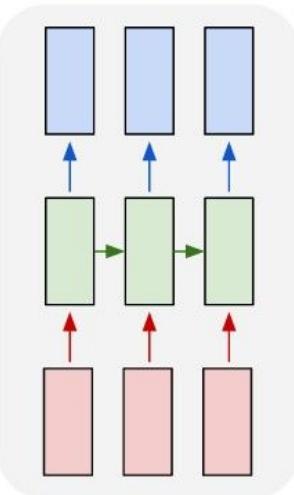
many to one



many to many



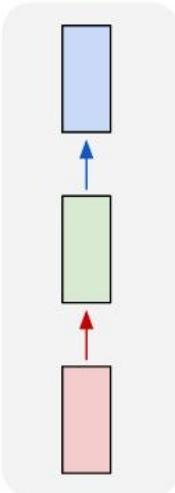
many to many



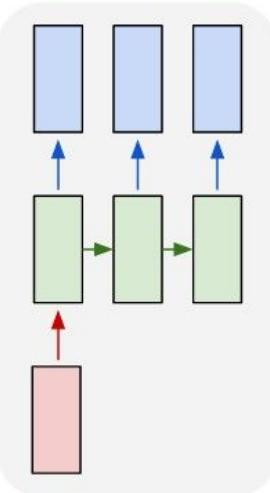
→ e.g. **Image Captioning**
image -> sequence of words

Recurrent Networks offer a lot of flexibility:

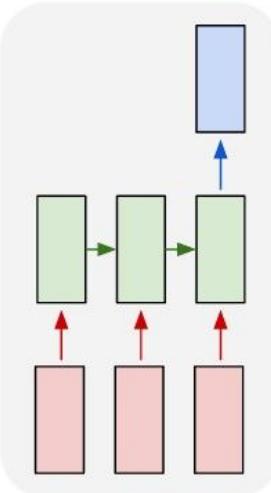
one to one



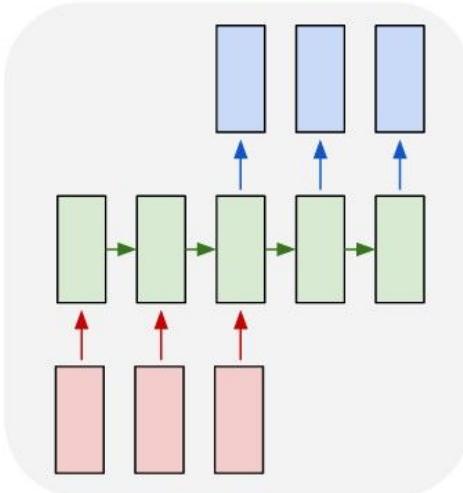
one to many



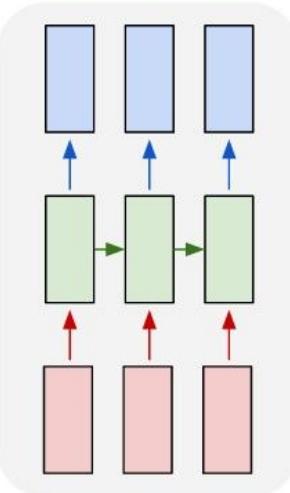
many to one



many to many



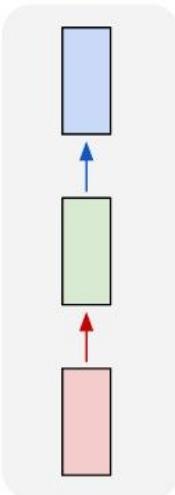
many to many



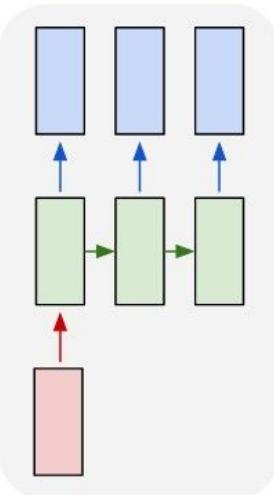
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:

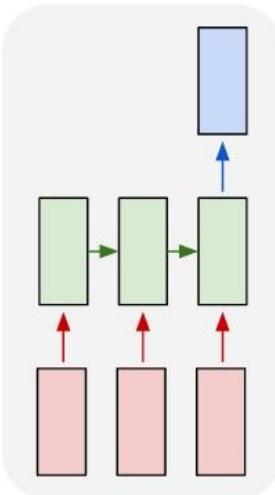
one to one



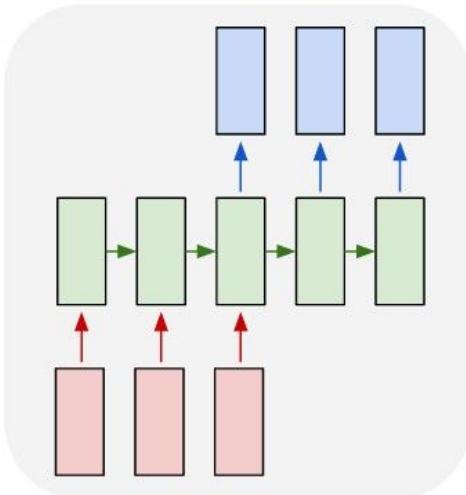
one to many



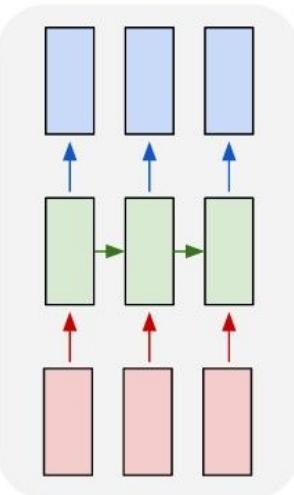
many to one



many to many



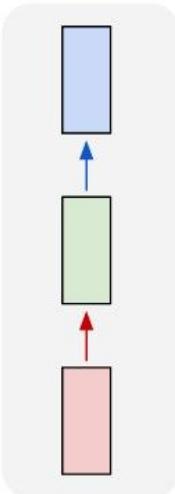
many to many



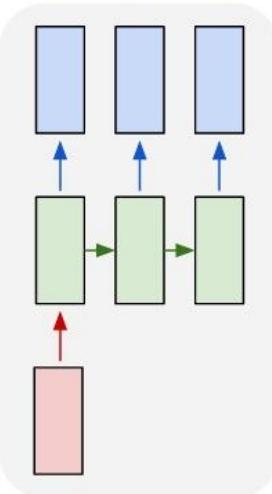
↑
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Networks offer a lot of flexibility:

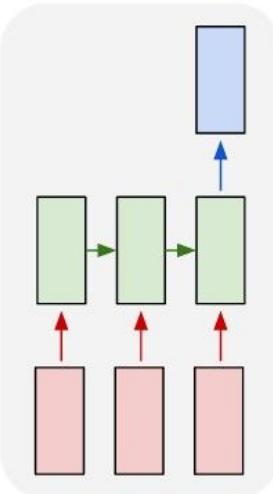
one to one



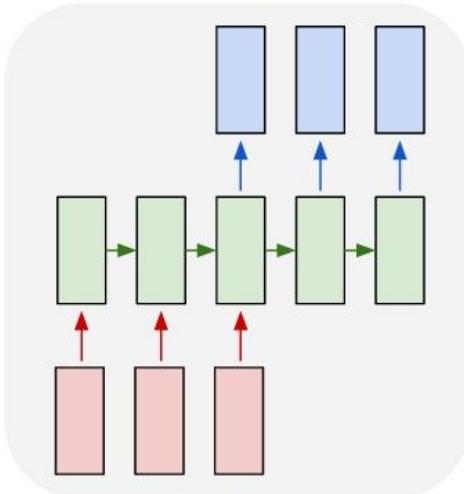
one to many



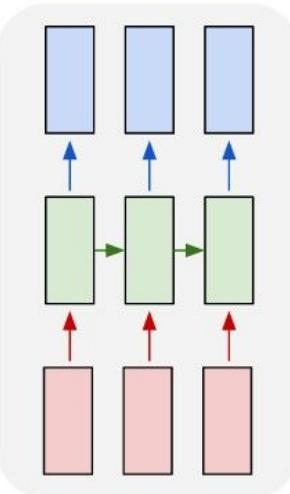
many to one



many to many



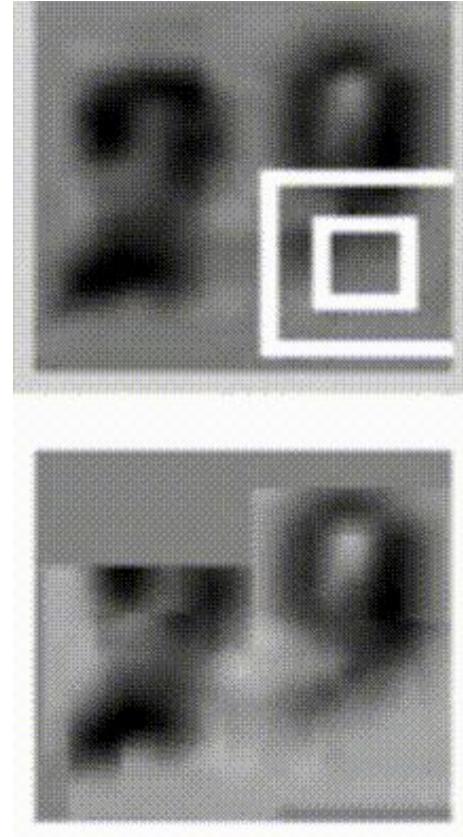
many to many



e.g. Video classification on frame level

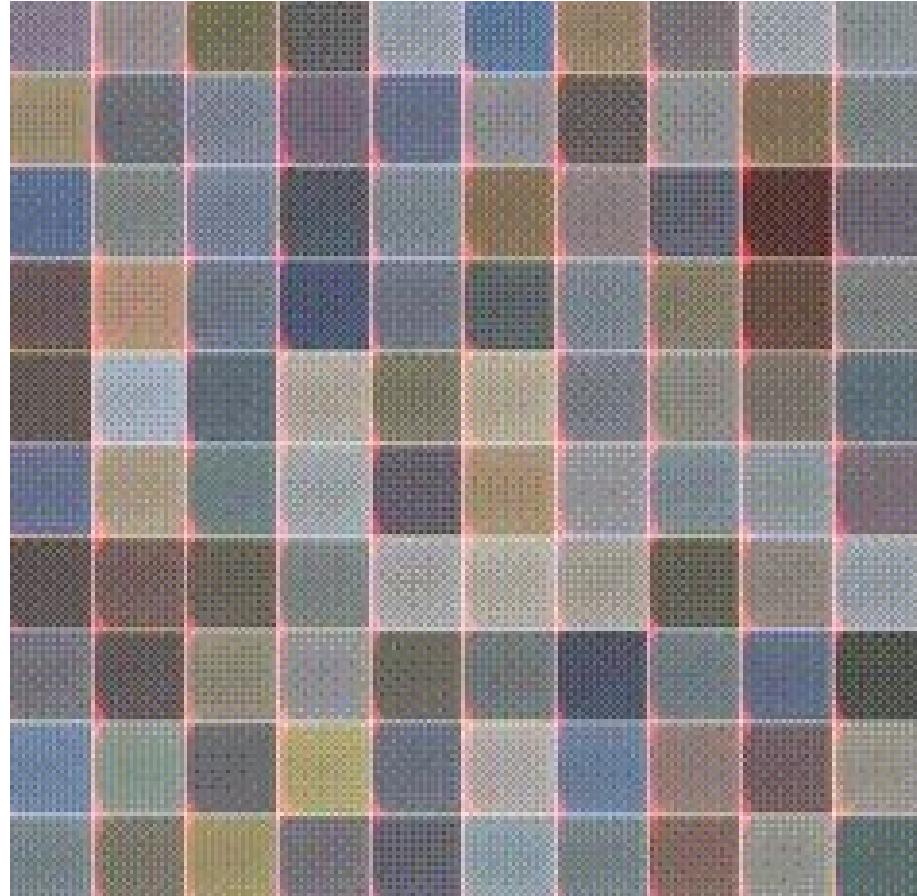
Sequential Processing of fixed inputs

Multiple Object Recognition with
Visual Attention, Ba et al.

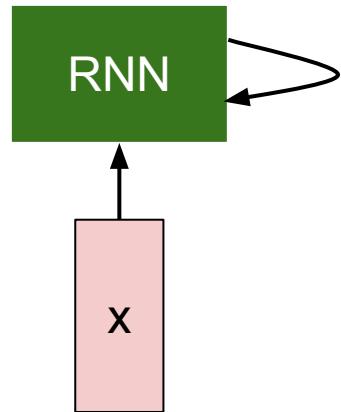


Sequential Processing of fixed outputs

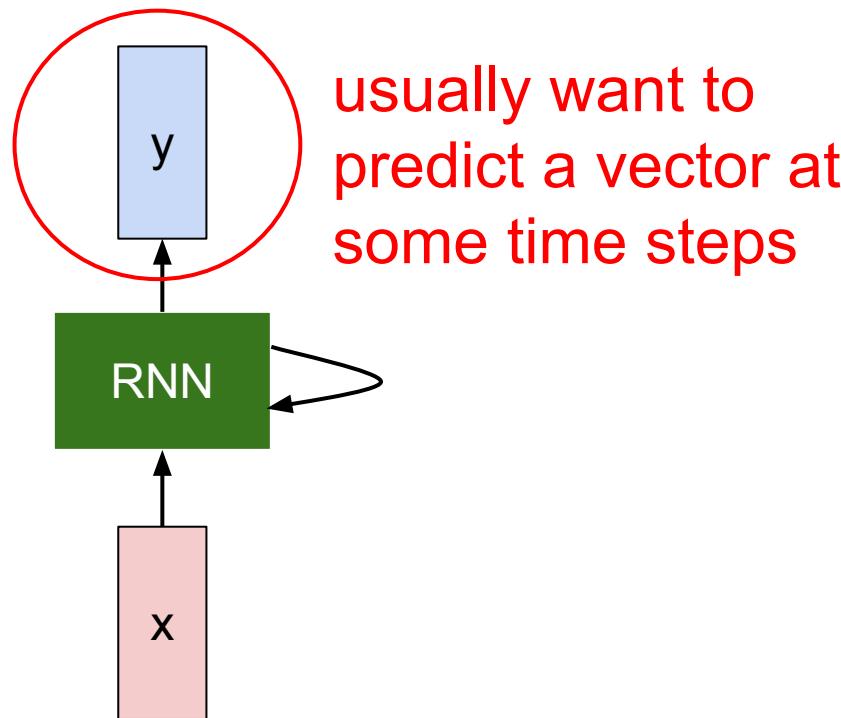
DRAW: A Recurrent
Neural Network For
Image Generation,
Gregor et al.



Recurrent Neural Network

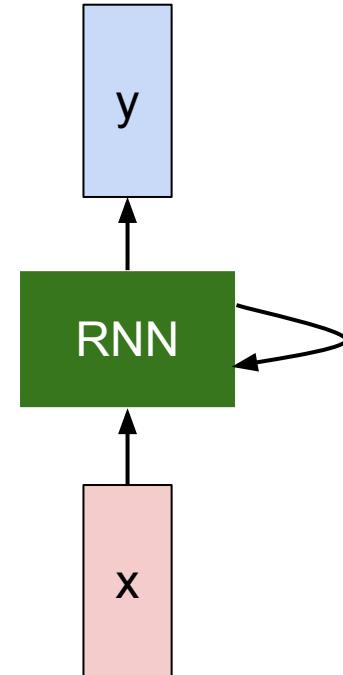


Recurrent Neural Network



Recurrent Neural Network

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

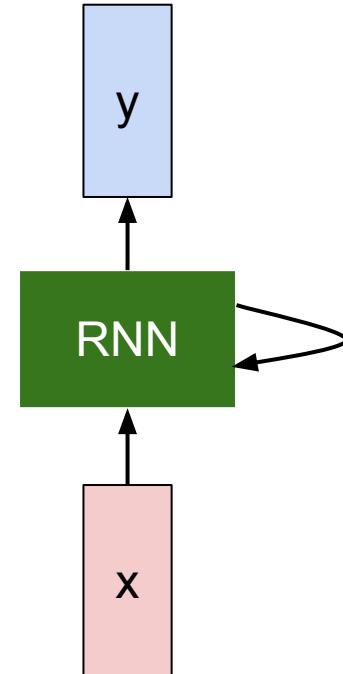


Recurrent Neural Network

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

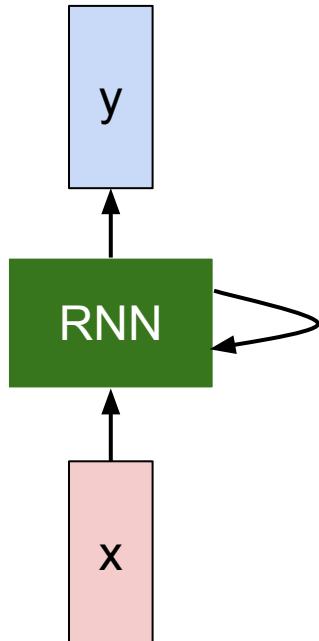
$$h_t = f_W(h_{t-1}, x_t)$$

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



(Vanilla) Recurrent Neural Network

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



$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$



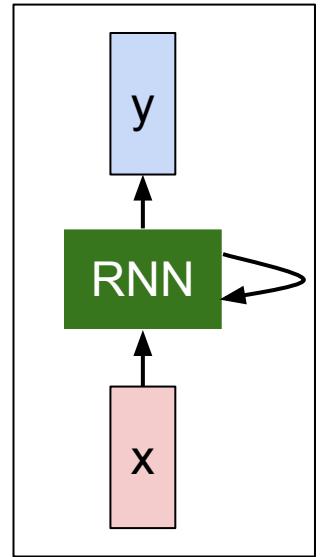
$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t)$$

$$y_t = W_{hy}\mathbf{h}_t$$

Character-level language model example

Vocabulary:
[h,e,l,o]

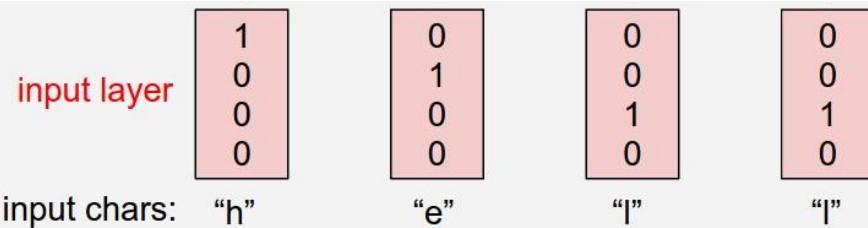
Example training
sequence:
“hello”



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

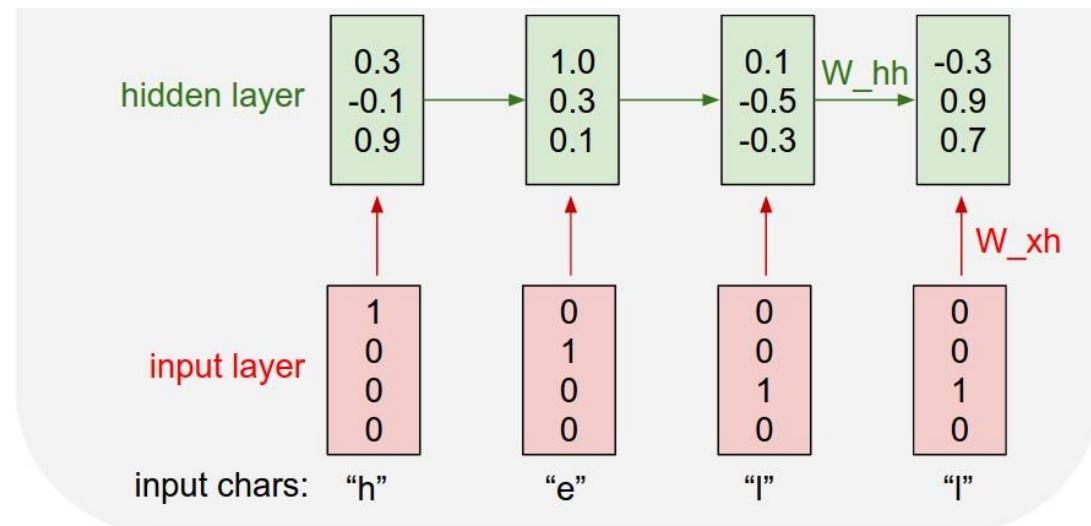


Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

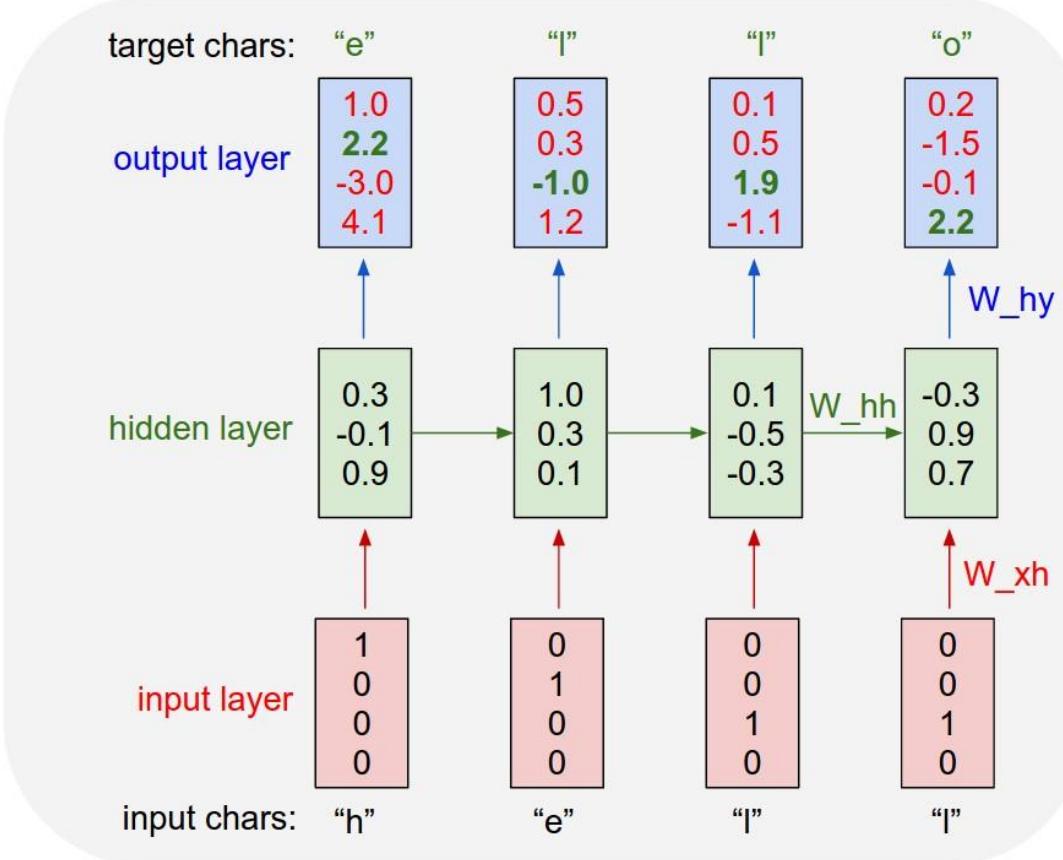
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”



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

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  # data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print('data has %d characters, %d unique.' % (data_size, vocab_size))
12 char_to_ix = {ch:i for i,ch in enumerate(chars)}
13 ix_to_char = {i:ch for i,ch in enumerate(chars)}
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wkh = np.random.rand(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.rand(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.rand(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs,targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(wkh, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(why, hs[t]) # by = unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
44
45         # backward pass: compute gradients going backwards
46         dwhx, dwhh, dwhy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
47         dbh, dby = np.zeros_like(bh), np.zeros_like(by)
48         dhnext = np.zeros_like(hs[0])
49         for t2 in reversed(xrange(len(inputs))):
50             dy = np.copy(ps[t2])
51             dy[targets[t2]] -= 1 # backprop into y
52             dyw = np.dot(dy, hs[t2].T)
53             dh = np.dot(why.T, dy) + dhnext * backprop into h
54             ddraw = (i - hs[t2].T) * dh # backprop through tanh nonlinearity
55             dbh += ddraw
56             dwhx += np.dot(ddraw, xs[t2].T)
57             dwhh += np.dot(ddraw, hs[t2-1].T)
58             dhnext = np.dot(why.T, ddraw)
59             for dparam in [dwhx, dwhh, dwhy, dbh, dby]:
60                 np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61
62     return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
```

```
63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     x[seed_ix] = 1
70     ixes = []
71     for t in xrange(n):
72         h = np.tanh(np.dot(wkh, x) + np.dot(whh, h) + bh)
73         y = np.dot(why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p.ravel())
76         x = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79
80     return ixes
81
82 n, p = 0, 0
83 mxwh, mwhh, mmhy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
84 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
85 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
86 while True:
87     # prepare inputs (we're sweeping from left to right in steps seq_length long)
88     if p+seq_length >= len(data) or n == 0:
89         hprev = np.zeros((hidden_size,1)) # reset RNN memory
90         p = 0 # go from start of data
91     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
92     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
93
94     # sample from the model now and then
95     if n % 100 == 0:
96         sample_ix = sample(hprev, inputs[0], 200)
97         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
98         print('----\n%s\n----' % (txt,))
99
100    # forward seq_length characters through the net and fetch gradient
101    loss, dwhx, dwhh, dwy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
102    smooth_loss = smooth_loss * .999 + loss * .001
103    if n % 100 == 0: print('iter %d, loss: %f' % (n, smooth_loss)) # print progress
104
105    # perform parameter update with Adagrad
106    for param, dparam, mem in zip([wkh, whh, why, bh, by],
107                                 [dwhx, dwhh, dwy, dbh, dby],
108                                 [mxwh, mwhh, mmhy, mbh, mby]):
109        mem += dparam * dparam
110        param += -learning_rate * param / np.sqrt(mem + 1e-8) # adagrad update
111
112    p += seq_length # move data pointer
113    n += 1 # iteration counter
```

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

min-char-rnn.py gist

```
1  #!/usr/bin/python
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 = 20 # number of steps to unroll the RNN for
17 learning_rate = 2e-1

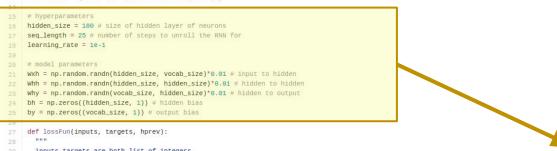
18 # Model parameters
19 wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
20 wb = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
21 bh = np.random.randn(vocab_size, hidden_size)*0.01 # hidden bias
22 bh0 = np.zeros(hidden_size, ) # hidden bias
23 by = np.zeros(vocab_size, ) # output bias

24 def lossfun(inputs, targets, hprev):
25     """ Inputs,targets are both lists of integers.
26     hprev is Hx1 array of initial hidden state
27     returns the loss, gradients on model parameters, and last hidden state
28     """
29     xs, hs, ys, ps = [], [], [], []
30     h0 = np.copy(hprev)
31     for t in range(len(inputs)):
32         x = np.zeros((vocab_size, )) # encode in 1-of-k representation
33         x[inputs[t]] = 1
34         h1 = np.tanh(np.dot(wb, x[1:]) + np.dot(wh, hs[-1]) + bh) # hidden state
35         y = np.dot(wb, h1) + by # compute output
36         ps[t] = np.exp(y)/np.sum(np.exp(y)) # probabilities for next chars
37         loss += -np.log(ps[t][targets[t]]) # softmax (cross-entropy loss)
38         # backprop into x
39         dx = np.zeros_like(x)
40         dy = np.zeros_like(y)
41         dy[targets[t]] -= 1 # backprop into y
42         dh = np.zeros_like(h1) # backprop into h
43         db = np.zeros_like(bh) # backprop into b
44         dwb = np.zeros_like(wb) # backprop into wb
45         dwh = np.zeros_like(wh) # backprop into wh
46         dbh = np.zeros_like(bh) # backprop into bh
47         for upar in [dx, db, dwb, dwh, dbh]:
48             np.clip(upar, -5, 5, out=upar) # clip to mitigate exploding gradients
49         dwb += np.outer(dx, dy) # backprop through tanh nonlinearity
50         dwh += np.outer(hs[-1], dy)
51         dbh += np.outer(dy, db)
52         db += dy
53         dh += np.dot(wb.T, dy) + dwh # backprop into h
54         dh0 = np.tanh(np.dot(wb, x[1:]) + dh) # backprop through tanh nonlinearity
55         dbh += dh * dwb
56         dwh += dh * dwh
57         dbh0 = np.tanh(np.dot(wb, x[1:]) + dbh) # backprop through tanh nonlinearity
58         db += dbh0 * dwb
59         dwh += dbh0 * dwh
60         for opar in [dx, db, dwb, dwh, dbh]:
61             np.clip(opar, -5, 5, out=opar) # clip to mitigate exploding gradients
62         dwb += np.outer(dx, dy) # backprop through tanh nonlinearity
63         dwh += np.outer(hs[-1], dy)
64         dbh += np.outer(dy, db)
65         db += dy
66         dx = np.zeros_like(x)
67         db = np.zeros_like(bh)
68         dwb = np.zeros_like(wb)
69         dwh = np.zeros_like(wh)
70         dbh = np.zeros_like(bh)
71     return loss

72 def sample(hprev, seed_ix, n):
73     """ Sample a sequence of integers from the model.
74     h is memory state, seed_ix is seed integer for first time step
75     """
76     x = np.zeros((vocab_size, ))
77     x[seed_ix] = 1
78     for t in range(n):
79         h = np.tanh(np.dot(wb, x) + np.dot(wh, h) + bh)
80         y = np.dot(wb, h) + by
81         p = np.exp(y) / np.sum(np.exp(y))
82         ix = np.random.choice(range(vocab_size), p=p.ravel())
83         x = np.zeros((vocab_size, ))
84         x[ix] = 1
85     return ix

86 n, p = 0, 0
87 mem, mem0, mem1, mem2, mem3, mem4, mem5, mem6, mem7, mem8, mem9, mem10, mem11, mem12, mem13, mem14, mem15, mem16, mem17, mem18, mem19, mem20, mem21, mem22, mem23, mem24, mem25, mem26, mem27, mem28, mem29, mem30, mem31, mem32, mem33, mem34, mem35, mem36, mem37, mem38, mem39, mem40, mem41, mem42, mem43, mem44, mem45, mem46, mem47, mem48, mem49, mem50, mem51, mem52, mem53, mem54, mem55, mem56, mem57, mem58, mem59, mem60, mem61, mem62, mem63, mem64, mem65, mem66, mem67, mem68, mem69, mem70, mem71, mem72, mem73, mem74, mem75, mem76, mem77, mem78, mem79, mem80, mem81, mem82, mem83, mem84, mem85, mem86, mem87, mem88, mem89, mem90, mem91, mem92, mem93, mem94, mem95, mem96, mem97, mem98, mem99, mem100, mem101, mem102, mem103, mem104, mem105, mem106, mem107, mem108, mem109, mem110, mem111, mem112, mem113, mem114, mem115, mem116, mem117, mem118, mem119, mem120, mem121, mem122, mem123, mem124, mem125, mem126, mem127, mem128, mem129, mem130, mem131, mem132, mem133, mem134, mem135, mem136, mem137, mem138, mem139, mem140, mem141, mem142, mem143, mem144, mem145, mem146, mem147, mem148, mem149, mem150, mem151, mem152, mem153, mem154, mem155, mem156, mem157, mem158, mem159, mem160, mem161, mem162, mem163, mem164, mem165, mem166, mem167, mem168, mem169, mem170, mem171, mem172, mem173, mem174, mem175, mem176, mem177, mem178, mem179, mem180, mem181, mem182, mem183, mem184, mem185, mem186, mem187, mem188, mem189, mem190, mem191, mem192, mem193, mem194, mem195, mem196, mem197, mem198, mem199, mem200, mem201, mem202, mem203, mem204, mem205, mem206, mem207, mem208, mem209, mem210, mem211, mem212, mem213, mem214, mem215, mem216, mem217, mem218, mem219, mem220, mem221, mem222, mem223, mem224, mem225, mem226, mem227, mem228, mem229, mem230, mem231, mem232, mem233, mem234, mem235, mem236, mem237, mem238, mem239, mem240, mem241, mem242, mem243, mem244, mem245, mem246, mem247, mem248, mem249, mem250, mem251, mem252, mem253, mem254, mem255, mem256, mem257, mem258, mem259, mem260, mem261, mem262, mem263, mem264, mem265, mem266, mem267, mem268, mem269, mem270, mem271, mem272, mem273, mem274, mem275, mem276, mem277, mem278, mem279, mem280, mem281, mem282, mem283, mem284, mem285, mem286, mem287, mem288, mem289, mem290, mem291, mem292, mem293, mem294, mem295, mem296, mem297, mem298, mem299, mem299, mem300, mem301, mem302, mem303, mem304, mem305, mem306, mem307, mem308, mem309, mem310, mem311, mem312, mem313, mem314, mem315, mem316, mem317, mem318, mem319, mem320, mem321, mem322, mem323, mem324, mem325, mem326, mem327, mem328, mem329, mem330, mem331, mem332, mem333, mem334, mem335, mem336, mem337, mem338, mem339, mem340, mem341, mem342, mem343, mem344, mem345, mem346, mem347, mem348, mem349, mem350, mem351, mem352, mem353, mem354, mem355, mem356, mem357, mem358, mem359, mem360, mem361, mem362, mem363, mem364, mem365, mem366, mem367, mem368, mem369, mem370, mem371, mem372, mem373, mem374, mem375, mem376, mem377, mem378, mem379, mem380, mem381, mem382, mem383, mem384, mem385, mem386, mem387, mem388, mem389, mem390, mem391, mem392, mem393, mem394, mem395, mem396, mem397, mem398, mem399, mem399, mem400, mem401, mem402, mem403, mem404, mem405, mem406, mem407, mem408, mem409, mem409, mem410, mem411, mem412, mem413, mem414, mem415, mem416, mem417, mem418, mem419, mem419, mem420, mem421, mem422, mem423, mem424, mem425, mem426, mem427, mem428, mem429, mem429, mem430, mem431, mem432, mem433, mem434, mem435, mem436, mem437, mem438, mem439, mem439, mem440, mem441, mem442, mem443, mem444, mem445, mem446, mem447, mem448, mem449, mem449, mem450, mem451, mem452, mem453, mem454, mem455, mem456, mem457, mem458, mem459, mem459, mem460, mem461, mem462, mem463, mem464, mem465, mem466, mem467, mem468, mem469, mem469, mem470, mem471, mem472, mem473, mem474, mem475, mem476, mem477, mem478, mem478, mem479, mem479, mem480, mem481, mem482, mem483, mem484, mem485, mem486, mem487, mem488, mem489, mem489, mem490, mem491, mem492, mem493, mem494, mem495, mem496, mem497, mem497, mem498, mem498, mem499, mem499, mem500, mem501, mem502, mem503, mem504, mem505, mem506, mem507, mem508, mem509, mem509, mem510, mem511, mem512, mem513, mem514, mem515, mem516, mem517, mem518, mem519, mem519, mem520, mem521, mem522, mem523, mem524, mem525, mem526, mem527, mem528, mem529, mem529, mem530, mem531, mem532, mem533, mem534, mem535, mem536, mem537, mem538, mem539, mem539, mem540, mem541, mem542, mem543, mem544, mem545, mem546, mem547, mem548, mem549, mem549, mem550, mem551, mem552, mem553, mem554, mem555, mem556, mem557, mem558, mem559, mem559, mem560, mem561, mem562, mem563, mem564, mem565, mem566, mem567, mem568, mem569, mem569, mem570, mem571, mem572, mem573, mem574, mem575, mem576, mem577, mem578, mem578, mem579, mem579, mem580, mem581, mem582, mem583, mem584, mem585, mem586, mem587, mem588, mem589, mem589, mem590, mem591, mem592, mem593, mem594, mem595, mem596, mem597, mem597, mem598, mem598, mem599, mem599, mem600, mem601, mem602, mem603, mem604, mem605, mem606, mem607, mem608, mem609, mem609, mem610, mem611, mem612, mem613, mem614, mem615, mem616, mem617, mem618, mem619, mem619, mem620, mem621, mem622, mem623, mem624, mem625, mem626, mem627, mem628, mem629, mem629, mem630, mem631, mem632, mem633, mem634, mem635, mem636, mem637, mem638, mem639, mem639, mem640, mem641, mem642, mem643, mem644, mem645, mem646, mem647, mem648, mem649, mem649, mem650, mem651, mem652, mem653, mem654, mem655, mem656, mem657, mem658, mem659, mem659, mem660, mem661, mem662, mem663, mem664, mem665, mem666, mem667, mem668, mem669, mem669, mem670, mem671, mem672, mem673, mem674, mem675, mem676, mem677, mem678, mem678, mem679, mem679, mem680, mem681, mem682, mem683, mem684, mem685, mem686, mem687, mem688, mem689, mem689, mem690, mem691, mem692, mem693, mem694, mem695, mem696, mem697, mem697, mem698, mem698, mem699, mem699, mem700, mem701, mem702, mem703, mem704, mem705, mem706, mem707, mem708, mem709, mem709, mem710, mem711, mem712, mem713, mem714, mem715, mem716, mem717, mem718, mem719, mem719, mem720, mem721, mem722, mem723, mem724, mem725, mem726, mem727, mem728, mem729, mem729, mem730, mem731, mem732, mem733, mem734, mem735, mem736, mem737, mem738, mem739, mem739, mem740, mem741, mem742, mem743, mem744, mem745, mem746, mem747, mem748, mem749, mem749, mem750, mem751, mem752, mem753, mem754, mem755, mem756, mem757, mem758, mem759, mem759, mem760, mem761, mem762, mem763, mem764, mem765, mem766, mem767, mem768, mem769, mem769, mem770, mem771, mem772, mem773, mem774, mem775, mem776, mem777, mem778, mem778, mem779, mem779, mem780, mem781, mem782, mem783, mem784, mem785, mem786, mem787, mem788, mem789, mem789, mem790, mem791, mem792, mem793, mem794, mem795, mem796, mem797, mem797, mem798, mem798, mem799, mem799, mem800, mem801, mem802, mem803, mem804, mem805, mem806, mem807, mem808, mem809, mem809, mem810, mem811, mem812, mem813, mem814, mem815, mem816, mem817, mem818, mem819, mem819, mem820, mem821, mem822, mem823, mem824, mem825, mem826, mem827, mem828, mem829, mem829, mem830, mem831, mem832, mem833, mem834, mem835, mem836, mem837, mem838, mem839, mem839, mem840, mem841, mem842, mem843, mem844, mem845, mem846, mem847, mem848, mem849, mem849, mem850, mem851, mem852, mem853, mem854, mem855, mem856, mem857, mem858, mem859, mem859, mem860, mem861, mem862, mem863, mem864, mem865, mem866, mem867, mem868, mem869, mem869, mem870, mem871, mem872, mem873, mem874, mem875, mem876, mem877, mem878, mem878, mem879, mem879, mem880, mem881, mem882, mem883, mem884, mem885, mem886, mem887, mem888, mem889, mem889, mem890, mem891, mem892, mem893, mem894, mem895, mem896, mem897, mem897, mem898, mem898, mem899, mem899, mem900, mem901, mem902, mem903, mem904, mem905, mem906, mem907, mem908, mem909, mem909, mem910, mem911, mem912, mem913, mem914, mem915, mem916, mem917, mem918, mem919, mem919, mem920, mem921, mem922, mem923, mem924, mem925, mem926, mem927, mem928, mem929, mem929, mem930, mem931, mem932, mem933, mem934, mem935, mem936, mem937, mem938, mem939, mem939, mem940, mem941, mem942, mem943, mem944, mem945, mem946, mem947, mem948, mem949, mem949, mem950, mem951, mem952, mem953, mem954, mem955, mem956, mem957, mem958, mem959, mem959, mem960, mem961, mem962, mem963, mem964, mem965, mem966, mem967, mem968, mem969, mem969, mem970, mem971, mem972, mem973, mem974, mem975, mem976, mem977, mem978, mem978, mem979, mem979, mem980, mem981, mem982, mem983, mem984, mem985, mem986, mem987, mem988, mem988, mem989, mem989, mem990, mem991, mem992, mem993, mem994, mem995, mem996, mem997, mem997, mem998, mem998, mem999, mem999, mem1000, mem1001, mem1002, mem1003, mem1004, mem1005, mem1006, mem1007, mem1008, mem1009, mem1009, mem1010, mem1011, mem1012, mem1013, mem1014, mem1015, mem1016, mem1017, mem1018, mem1019, mem1019, mem1020, mem1021, mem1022, mem1023, mem1024, mem1025, mem1026, mem1027, mem1028, mem1029, mem1029, mem1030, mem1031, mem1032, mem1033, mem1034, mem1035, mem1036, mem1037, mem1038, mem1039, mem1039, mem1040, mem1041, mem1042, mem1043, mem1044, mem1045, mem1046, mem1047, mem1048, mem1049, mem1049, mem1050, mem1051, mem1052, mem1053, mem1054, mem1055, mem1056, mem1057, mem1058, mem1059, mem1059, mem1060, mem1061, mem1062, mem1063, mem1064, mem1065, mem1066, mem1067, mem1068, mem1069, mem1069, mem1070, mem1071, mem1072, mem1073, mem1074, mem1075, mem1076, mem1077, mem1078, mem1078, mem1079, mem1079, mem1080, mem1081, mem1082, mem1083, mem1084, mem1085, mem1086, mem1087, mem1088, mem1088, mem1089, mem1089, mem1090, mem1091, mem1092, mem1093, mem1094, mem1095, mem1096, mem1097, mem1097, mem1098, mem1098, mem1099, mem1099, mem1100, mem1101, mem1102, mem1103, mem1104, mem1105, mem1106, mem1107, mem1108, mem1109, mem1109, mem1110, mem1111, mem1112, mem1113, mem1114, mem1115, mem1116, mem1117, mem1118, mem1119, mem1119, mem1120, mem1121, mem1122, mem1123, mem1124, mem1125, mem1126, mem1127, mem1128, mem1129, mem1129, mem1130, mem1131, mem1132, mem1133, mem1134, mem1135, mem1136, mem1137, mem1138, mem1138, mem1139, mem1139, mem1140, mem1141, mem1142, mem1143, mem1144, mem1145, mem1146, mem1147, mem1148, mem1148, mem1149, mem1149, mem1150, mem1151, mem1152, mem1153, mem1154, mem1155, mem1156, mem1157, mem1158, mem1159, mem1159, mem1160, mem1161, mem1162, mem1163, mem1164, mem1165, mem1166, mem1167, mem1168, mem1169, mem1169, mem1170, mem1171, mem1172, mem1173, mem1174, mem1175, mem1176, mem1177, mem1178, mem1178, mem1179, mem1179, mem1180, mem1181, mem1182, mem1183, mem1184, mem1185, mem1186, mem1187, mem1188, mem1188, mem1189, mem1189, mem1190, mem1191, mem1192, mem1193, mem1194, mem1195, mem1196, mem1197, mem1197, mem1198, mem1198, mem1199, mem1199, mem1200, mem1201, mem1202, mem1203, mem1204, mem1205, mem1206, mem1207, mem1208, mem1209, mem1209, mem1210, mem1211, mem1212, mem1213, mem1214, mem1215, mem1216, mem1217, mem1218, mem1219, mem1219, mem1220, mem1221, mem1222, mem1223, mem1224, mem1225, mem1226, mem1227, mem1228, mem1229, mem1229, mem1230, mem1231, mem1232, mem1233, mem1234, mem1235, mem1236, mem1237, mem1238, mem1238, mem1239, mem1239, mem1240, mem1241, mem1242, mem1243, mem1244, mem1245, mem1246, mem1247, mem1248, mem1248, mem1249, mem1249, mem1250, mem1251, mem1252, mem1253, mem1254, mem1255, mem1256, mem1257, mem1258, mem1259, mem1259, mem1260, mem1261, mem1262, mem1263, mem1264, mem1265, mem1266, mem1267, mem1268, mem1269, mem1269, mem1270, mem1271, mem1272, mem1273, mem1274, mem1275, mem1276, mem1277, mem1278, mem1278, mem1279, mem1279, mem1280, mem1281, mem1282, mem1283, mem1284, mem1285, mem1286, mem1287, mem1288, mem1288, mem1289, mem1289, mem1290, mem1291, mem1292, mem1293, mem1294, mem1295, mem1296, mem1297, mem1297, mem1298, mem1298, mem1299, mem1299, mem1300, mem1301, mem1302, mem1303, mem1304, mem1305, mem1306, mem1307, mem1308, mem1309, mem1309, mem1310, mem1311, mem1312, mem1313, mem1314, mem1315, mem1316, mem1317, mem1318, mem1319, mem1319, mem1320, mem1321, mem1322, mem1323, mem1324, mem1325, mem1326, mem1327, mem1328, mem1329, mem1329, mem1330, mem1331, mem1332, mem1333, mem1334, mem1335, mem1336, mem1337, mem1338, mem1338, mem1339, mem1339, mem1340, mem1341, mem1342, mem1343, mem1344, mem1345, mem1346, mem1347, mem1348, mem1348, mem1349, mem1349, mem1350, mem1351, mem1352, mem1353, mem1354, mem1355, mem1356, mem1357, mem1358, mem1359, mem1359, mem1360, mem1361, mem1362, mem1363, mem1364, mem1365, mem1366, mem1367, mem1368, mem1369, mem1369, mem1370, mem1371, mem1372, mem1373, mem1374, mem1375, mem1376, mem1377, mem1378, mem1378, mem1379, mem1379, mem1380, mem1381, mem1382, mem1383, mem1384, mem1385, mem1386, mem1387, mem1388, mem1388, mem1389, mem1389, mem1390, mem1391, mem1392, mem1393, mem1394, mem1395, mem1396, mem1397, mem1397, mem1398, mem1398, mem1399, mem1399, mem1400, mem1401, mem1402, mem1403, mem1404, mem1405, mem1406, mem1407, mem1408, mem1409, mem1409, mem1410, mem1411, mem1412, mem1413, mem1414, mem1415, mem1416, mem1417, mem1418, mem1418, mem1419, mem1419, mem1420, mem1421, mem1422, mem1423, mem1424, mem1425, mem1426, mem1427, mem1428, mem1429, mem1429, mem1430, mem1431, mem1432, mem1433, mem1434, mem1435, mem1436, mem1437, mem1438, mem1438, mem1439, mem1439, mem1440, mem1441, mem1442, mem1443, mem1444, mem1445, mem1446, mem1447, mem1448, mem1448, mem1449, mem1449, mem1450, mem1451, mem1452, mem1453, mem1454, mem1455, mem1456, mem1457, mem1458, mem1459, mem1459, mem1460, mem1461, mem1462, mem1463, mem1464, mem1465, mem1466, mem1467, mem1468, mem1469, mem1469, mem1470, mem1471, mem1472, mem1473, mem1474, mem1475, mem1476, mem1477, mem1478, mem1478, mem1479, mem1479, mem1480, mem1481, mem1482, mem1483, mem1484, mem1485, mem1486, mem1487, mem1488, mem1488, mem1489, mem1489, mem1490, mem1491, mem1492, mem1493, mem1494, mem1495, mem1496, mem1497, mem1497, mem1498, mem1498, mem1499, mem1499, mem1500, mem1501, mem1502, mem1503, mem1504, mem1505, mem1506, mem1507, mem1508, mem1509, mem1509, mem1510, mem1511, mem1512, mem1513, mem1514, mem1515, mem1516, mem1517, mem1518, mem1519, mem1519, mem1520, mem1521, mem1522, mem1523, mem1524, mem1525, mem1526, mem1527, mem1528, mem1529, mem1529, mem1530, mem1531, mem1532, mem1533, mem1534, mem1535, mem1536, mem1537, mem1538, mem1538, mem1539, mem1539, mem1540, mem1541, mem1542, mem1543, mem1544, mem1545, mem1546, mem1547, mem1548, mem1548, mem1549, mem1549, mem1550, mem1551, mem1552, mem1553, mem1554, mem1555, mem1556, mem1557, mem1558, mem1559, mem1559, mem1560, mem1561, mem1562, mem1563, mem1564, mem1565, mem1566, mem1567, mem1568, mem1569, mem1569, mem1570, mem1571, mem1572, mem1573, mem1574, mem1575, mem1576, mem1577, mem1578, mem1578, mem1579, mem1579, mem1580, mem1581, mem1582, mem1583, mem1584, mem1585, mem1586, mem1587, mem1588, mem1588, mem1589, mem1589, mem1590, mem1591, mem1592, mem1593, mem1594, mem1595, mem1596, mem1597, mem1597, mem1598, mem1598, mem1599, mem1599, mem1600, mem1601, mem1602, mem1603, mem1604, mem1605, mem1606, mem1607, mem1608, mem1609, mem1609, mem1610, mem1611, mem1612, mem1613, mem1614, mem1615, mem1616, mem1617, mem1618, mem1618, mem1619, mem1619, mem1620, mem1621, mem1622, mem1623, mem1624, mem1625, mem1626, mem1627, mem1628, mem1629, mem1629, mem1630, mem1631, mem1632, mem1633, mem1634, mem1635, mem1636, mem1637, mem1638, mem1638, mem1639, mem1639, mem1640, mem1641, mem1642, mem1643, mem1644, mem1645, mem1646, mem1647, mem1648, mem1648, mem1649, mem1649, mem1650, mem1651, mem1652, mem1653, mem1654, mem1655, mem1656, mem1657, mem1658, mem1659, mem1659, mem1660, mem1661, mem1662, mem1663, mem1664, mem1665, mem1666, mem1667, mem1668, mem1669, mem1669, mem1670, mem1671, mem1672, mem1673, mem1674, mem1675, mem1676, mem1677, mem1678, mem1678, mem1679, mem1679, mem1680, mem1681, mem1682, mem1683, mem1684, mem1685, mem1686, mem1687, mem1688, mem1688, mem1689, mem1689, mem1690, mem1691, mem1692, mem1693, mem1694, mem1695, mem1696, mem1697, mem1697, mem1698, mem1698, mem1699, mem1699, mem1700, mem1701, mem1702, mem1703, mem1704, mem1705, mem1706, mem1707, mem1708, mem1709, mem1709, mem1710, mem1711, mem1712, mem1713, mem1714, mem1715, mem1716, mem1717, mem1718, mem1718, mem1719, mem1719, mem1720, mem1721, mem1722, mem1723, mem1724, mem1725, mem1726, mem1727, mem1728, mem1729, mem1729, mem1730, mem1731, mem1732, mem1733, mem1734, mem1735, mem1736, mem1737, mem1738, mem1738, mem1739, mem1739, mem1740, mem1741, mem1742, mem1743, mem1744, mem1745, mem1746, mem1747, mem1748, mem1748, mem1749, mem1749, mem1750, mem1751, mem1752, mem1753, mem1754, mem1755, mem1756, mem1757, mem1758, mem1759, mem1759, mem1760, mem1761, mem1762, mem1763, mem1764, mem1765, mem1766, mem1767, mem1768, mem1769, mem1769, mem1770, mem1771, mem1772, mem1773, mem1774, mem1775, mem1776, mem1777, mem1778, mem1778, mem1779, mem1779, mem1780, mem1781, mem1782, mem1783, mem1784, mem1785, mem1786, mem1787, mem1788, mem1788, mem1789, mem1789, mem1790, mem1791, mem1792, mem1793, mem1794, mem1795, mem1796, mem1797, mem1797, mem1798, mem1798, mem1799, mem1799, mem1800, mem1801, mem1802, mem1803, mem1804, mem1805, mem1806, mem1807, mem1808, mem1809, mem1809, mem1810, mem1811, mem1812, mem1813, mem1814, mem1815, mem1816, mem1817, mem1818, mem1818, mem1819, mem1819, mem1820, mem1821, mem1822, mem1823, mem1824, mem18
```

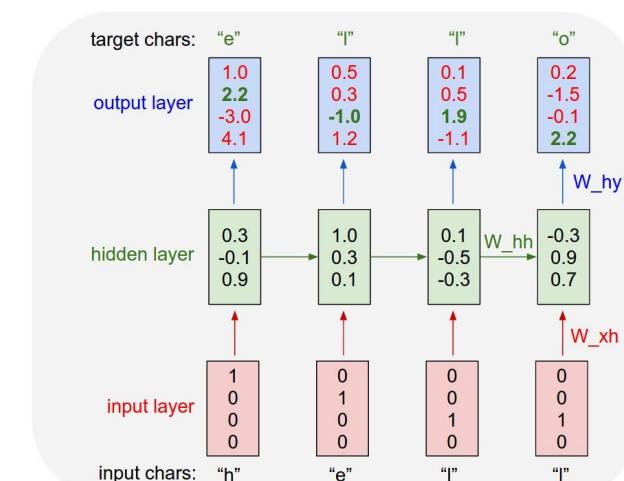
min-char-rnn.py gist



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
```

recal



min-char-rnn.py gist

Main loop

min-char-rnn.py gist

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

Main loop

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

min-char-rnn.py gist

Main loop

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  # Data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(data[1:(1)])
10 vocab_size = len(chars), len(data), len(chars)
11 print("data has %d characters, %d unique." % (data_size, vocab_size))
12 char_to_ix = {ch:i for i in range(len(chars))}
13 ix_to_char = {i:ch for ch in range(len(chars))} # 14
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 20 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
22 dbh, mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
23 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
24
25 while True:
26     # prepare inputs (we're sweeping from left to right in steps seq_length long)
27     if p+seq_length+1 >= len(data) or n == 0:
28         hprev = np.zeros((hidden_size,1)) # reset RNN memory
29         p = 0 # go from start of data
30
31     inputs, targets = data[p:p+seq_length], data[p+1:p+seq_length+1]
32
33     xs, hs, ys, ps = [0, 0, 0, 0]
34     hprev = np.copy(hprev)
35     loss = 0
36
37     for t in range(seq_length):
38         # forward pass
39         x = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
40         x[inputs[t]] = 1
41         hprev = np.tanh(np.dot(wh, hprev) + np.dot(bh, hprev[-1]) + bh)
42         y = np.exp(hprev)
43         ps = np.exp(y)/np.sum(np.exp(y)) # probabilities for next chars
44         loss += -np.log(ps[targets[t]]/np.sum(np.exp(y))) # softmax (cross-entropy loss)
45
46         # backward pass: compute gradients to params
47         dWxh += np.outer(x, hprev)
48         dbh += np.zeros_like(bh)
49         dWhh += np.outer(hprev, hprev)
50         dWhy += np.outer(y, np.zeros_like(y))
51
52         dx = np.zeros_like(x)
53         dy = np.copy(ps[1:])
54         dy[targets[t]] -= 1 # backprop into y
55         dh = np.zeros_like(hprev)
56
57         dh += np.dot(why.T, dy) + dWhh # backprop into h
58         dh += dh * np.tanh(dh) * (1 - dh**2) # backprop through tanh nonlinearity
59         dWxh += np.outer(dx, dh)
60         dbh += np.sum(dx * dh, axis=1, keepdims=True)
61         dWhh += np.outer(dh, dh)
62         dWhy += np.outer(dy, np.zeros_like(dy))
63
64         for dparam in [dWxh, dbh, dWhh, dWhy]:
65             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
66
67         np.clip(dbh, -1000, 1000, out=dbh, dtype=dbh.dtype, copy=False)
68         dbh[dbh < -1000] = -1000
69         dbh[dbh > 1000] = 1000
70
71         if dWxh.size > 0:
72             # sample from the model now and then
73             sample_ix = sample(hprev, inputs[0], 200)
74             txt = ''.join(ix_to_char[ix] for ix in sample_ix)
75             print '----\n%s\n----' % (txt, )
76
77     # forward seq_length characters through the net and fetch gradient
78     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
79     smooth_loss = smooth_loss * 0.999 + loss * 0.001
80
81     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
82
83     # perform parameter update with Adagrad
84     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
85                                   [dWxh, dWhh, dWhy, dbh, dby],
86                                   [mWxh, mWhh, mWhy, mbh, mby]):
87
88         mem += dparam * dparam
89         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
90
91     p += seq_length # move data pointer
92     n += 1 # iteration counter
```

min-char-rnn.py gist

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

Main loop



min-char-rnn.py gist

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  # Data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(set(data))
10 vocab_size = len(chars)
11 data_size = len(data)
12 print("data has %d characters, %d unique." % (data_size, vocab_size))
13 char_to_ix = {ch:i for i in range(len(chars))}
14 ix_to_char = {i:ch for ch in range(len(chars))} # 14
15
16 # hyperparameters
17 hidden_size = 100 # size of hidden layer of neurons
18 seq_length = 20 # number of steps to unroll the RNN for
19 learning_rate = 1e-1
20
21 model_params = {}
22
23 def init_randomhidden(state, vocab_size):
24     wh = np.random.rand(hidden_size, vocab_size)*0.01 # input to hidden
25     bh = np.random.rand(hidden_size, vocab_size)*0.01 # hidden to hidden
26     why = np.random.rand(vocab_size, hidden_size)*0.01 # hidden to output
27     by = np.zeros((vocab_size, 1)) # hidden bias
28     bi = np.zeros((vocab_size, 1)) # output bias
29
30     return wh, bh, why, by, bi
31
32 def lossFun(inputs, targets, hprev):
33
34     inputs,targets = both lists of integers.
35
36     hprev is Hx1 array of initial hidden state
37     returns the loss, gradients on model parameters, and last hidden state
38
39     xs, hs, ys, ps = O, O, O, O
40     h1t1 = np.copy(hprev)
41
42     for t in range(seq_length): # encode in 1-of-k representation
43         x1t1 = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
44         x1t1[inputs[t]] = 1
45
46         h1t1 = np.tanh(np.dot(wh, x1t1) + np.dot(bh, h1t1) + bh) # hidden state
47         ps1t1 = np.exp(h1t1) / np.sum(np.exp(h1t1)) # probabilities for next chars
48         y1t1 = np.argmax(ps1t1) # next char predicted
49         loss += -np.log(ps1t1[y1t1]) # softmax (cross-entropy loss)
50
51         # backprop into hidden state
52         dxt1 = np.zeros_like(x1t1)
53         dwh = np.zeros((vocab_size, hidden_size))
54         dbh = np.zeros((hidden_size, 1))
55         dwhy = np.zeros((hidden_size, vocab_size))
56         dby = np.zeros((vocab_size, 1))
57
58         for i in range(len(inputs)):
59             dxt1[i] = ps1t1[y1t1]
60             dy = np.copy(dxt1[i])
61             dy[y1t1] -= 1 # a backward delta y
62             dh = np.dot(dy, wh) + dbh # dh = dot(dy, wh) + dh
63             dh += dy # dh += dy
64             dh = np.dot(why.T, dh) + dwhy # dh = backprop through tanh nonlinearity
65             dh += dbh
66             dwh += np.dot(dy.T, x1t1)
67             dbh += np.sum(dy * x1t1, axis=0, keepdims=True)
68             dwhy += np.dot(x1t1.T, dy)
69
70         for dparam in [dwh, dbh, dwhy, dby]:
71             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
72             if dparam != 0:
73                 dparam *= 0.01 # scale down gradients
74
75     dh = np.zeros_like(h1t1)
76
77     sample_ix = sample(hprev, inputs[0], 200)
78     txt = ''.join(ix_to_char[ix] for ix in sample_ix)
79     print '----\n %s \n----' % (txt, )
80
81     n, p = 0, 0
82
83     mWxh, mWhh, mWhy = np.zeros_like(wh), np.zeros_like(wh),
84     np.zeros_like(why)
85     mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
86
87     smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
88
89     while True:
90
91         # prepare inputs (we're sweeping from left to right in steps seq_length long)
92         if p+seq_length+1 >= len(data) or n == 0:
93             hprev = np.zeros((hidden_size,1)) # reset RNN memory
94             p = 0 # go from start of data
95
96         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
97         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
98
99         # sample from the model now and then
100        if n % 100 == 0:
101            sample_ix = sample(hprev, inputs[0], 200)
102            txt = ''.join(ix_to_char[ix] for ix in sample_ix)
103            print '----\n %s \n----' % (txt, )
104
105        # forward seq_length characters through the net and fetch gradient
106        loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
107        smooth_loss = smooth_loss * 0.999 + loss * 0.001
108
109        if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
110
111        # perform parameter update with Adagrad
112        for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
113                                      [dWxh, dWhh, dWhy, dbh, dby],
114                                      [mWxh, mWhh, mWhy, mbh, mby]):
115
116            mem += dparam * dparam
117            param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
118
119            p += seq_length # move data pointer
120            n += 1 # iteration counter
121
122
```



Main loop

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

min-char-rnn.py gist

```
***  
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)  
BSD License  
***  
Import numpy as np  
  
# Data I/O  
data = open('train.txt', 'r').read() # should be simple plain text file  
chars = list(set(data))  
data_size, vocab_size = len(data), len(chars)  
print 'data has %d characters, %d unique.' % (data_size, vocab_size)  
char_to_ix = {ch:i for i, ch in enumerate(chars)}  
ix_to_char = {i:ch for ch in enumerate(chars)}  
  
# Hyperparameters  
hidden_size = 100 # size of hidden layer of neurons  
seq_length = 20 # number of steps to unroll the RNN for  
learning_rate = 1e-1  
  
# Model parameters  
wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden  
bh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden  
why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output  
bh0 = np.zeros(hidden_size, 1) # hidden bias  
by = np.zeros(vocab_size, 1) # output bias  
  
def lossFun(inputs, targets, hprev):  
    """  
    inputs,targets are both 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  
    for t in xrange(len(inputs)):  
        xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation  
        xs[t][inputs[t]] = 1  
        wht = np.tanh(np.dot(wh, xs[t]) + np.dot(bh, hs[t-1]) + bh) # hidden state  
        whyt = np.dot(why, wht) + by # unnormalized log probabilities for next chars  
        ps[t] = np.exp(whyt) / np.sum(np.exp(whyt)) # softmax (cross-entropy loss)  
        loss += -np.log(ps[t][targets[t],0])  
        dy = np.zeros_like(xs[t])  
        dy[targets[t]] = -1 # backprop into y  
        dh = np.dot(dy, why.T)  
        dbh = np.sum(dy, axis=0)  
        dby = np.sum(dy, axis=1)  
        dnext = np.zeros_like(hs[t])  
        dy = np.copy(dy[1:])  
        dy[t] = np.zeros_like(dy[0])  
        for opname in [dwh, dhh, dhy, dby]:  
            opname(dynext, -5, 5, out=dparam) # clip to mitigate exploding gradients  
            dynext = np.dot(dy, wh) + dbh + dby; dy = np.concatenate([dynext, dy[0]])  
  
    def sample(hx, ix):  
        """  
        sample a sequence of integers from the model  
        h is memory state, seed_ix is seed letter for first time step  
        """  
        x = np.zeros(vocab_size, 1)  
        x[seed_ix] = 1  
  
        for t in xrange(n):  
            h = np.tanh(np.dot(wh, x) + np.dot(bh, hx) + bh)  
            p = np.exp(why * np.dot(h, why.T))  
            ix = np.argmax(np.random.multinomial(p.ravel()))  
            x = np.zeros(vocab_size, 1)  
            x[ix] = 1  
            if t > 0: hx = h  
  
        return ix  
  
    n, p, h, ix = 0  
    mean, stdv, whyt = np.zeros_like(wh), np.zeros_like(wh), np.zeros_like(why)  
    dwh, dbh, dby = np.zeros_like(wh), np.zeros_like(bh), np.zeros_like(by)  
    smooth_loss = -np.inf  
    for line in data:  
        if n == seq_length:  
            # Unroll the model, now there's one less step in the loop  
            for t in xrange(seq_length):  
                inputs = char_to_ix[ix] for ix in line[0:seq_length-1]  
                targets = char_to_ix[ix] for ix in line[1:seq_length]  
  
                # Forward pass: compute scores through the net and fetch gradient  
                loss, dwh, dbh, dhy, dby, hprev = lossFun(inputs, targets, hprev)  
                smooth_loss = smooth_loss * 0.999 + loss * 0.001  
                if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress  
  
            # backward pass: compute gradients going backwards  
            dwhx, dwhh, dwhy = np.zeros_like(whx), np.zeros_like(whh), np.zeros_like(why)  
            dbhh, dby = 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)  
                dbhy += dy  
                dh = np.dot(why.T, dy) + dhnext # backprop into h  
                ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity  
                dbh += ddraw  
                dwhx += np.dot(ddraw, xs[t].T)  
                dwhh += np.dot(ddraw, hs[t-1].T)  
                dhnext = np.dot(wh.T, ddraw)  
  
            for dparam in [dwhx, dwhh, dwhy, dbh, dby]:  
                np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients  
            params += learning_rate * dparam / np.sqrt(n + 1e-8) # adaptive update  
  
        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 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(wh, xs[t]) + np.dot(bh, 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     dwhx, dwhh, dwhy = np.zeros_like(whx), np.zeros_like(whh), np.zeros_like(why)  
46     dbhh, 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         dbhy += dy  
53         dh = np.dot(why.T, dy) + dhnext # backprop into h  
54         ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity  
55         dbh += ddraw  
56         dwhx += np.dot(ddraw, xs[t].T)  
57         dwhh += np.dot(ddraw, hs[t-1].T)  
58         dhnext = np.dot(wh.T, ddraw)  
59     for dparam in [dwhx, dwhh, dwhy, dbh, dby]:  
60         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients  
61     return loss, dwhx, dwhh, 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
6 import numpy as np
7
8 # Data I/O
9 data = open('input.txt', 'r').read() # should be simple plain text file
10 chars = list(set(data))
11 data_size, vocab_size = len(data), len(chars)
12 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
13 char_to_ix = {ch:i for i in xrange(len(chars))}
14 ix_to_char = {i:ch for ch in xrange(len(chars))}

15 # Hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 20 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # Model parameters
21 Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros(hidden_size, 1) # hidden bias
25 by = np.zeros(vocab_size, 1) # output bias
26
27 by = np.zeros(vocab_size, 1) # output bias

```

```

28 def lossFun(inputs, targets, hprev):
29     """  
30         inputs,targets are both list of integers.  
31         hprev is Hx1 array of initial hidden state  
32         returns the loss, gradients on model parameters, and last hidden state  
33     """
34
35     xs, hs, ys, ps = {}, {}, {}, {}
36     hs[-1] = np.copy(hprev)
37     loss = 0
38
39     # forward pass
40     for t in xrange(len(inputs)):
41         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
42         xs[t][inputs[t]] = 1
43
44         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
45
46         ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
47         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
48
49         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
50
51     # backward pass: compute gradients going backwards
52     dch, dhb, dhy = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(why)
53     dby = np.zeros_like(by)
54     dhnext = np.zeros_like(hs[0])
55
56     for t in reversed(xrange(len(inputs)))�
57         dy = np.copy(ps[t])
58         dy[targets[t]] -= 1 # backprop into y
59         dh = np.dot(why.T, dy) + dhnext # dh = backprop through tanh nonlinearity
60
61         dhx = np.dot(Wxh.T, dh) # dh = backprop through dot product
62         dch += np.dot(dhx, xs[t].T) # dch = backprop through dot product
63         dby += np.sum(dhx, axis=1, keepdims=True) # dby = backprop through dot product
64         dhb += np.sum(dhx, axis=0, keepdims=True) # dhb = backprop through dot product
65
66         dhnext = np.tanh(dch) * dh # dhnext = gradient flow into next step
67
68     for opname in [dch, dhb, dhy, dby]:
69         np.clip(opname, -5, 5, out=opname) # clip to mitigate exploding gradients
70         opname -= opname.mean() # renormalize gradients
71         opname *= learning_rate # update rule
72
73     return loss, hs[-1]
74
75 def sample(hprev, seed_ix, n):
76     """  
77         sample a sequence of integers from the model  
78         h is memory state, seed_ix is seed integer for first time step  
79         n is max length of sampled sequence  
80     """
81     x = np.zeros((vocab_size, 1))
82     x[seed_ix] = 1
83
84     for t in xrange(n):
85         h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
86         p = np.exp(ys[t]) / np.sum(np.exp(ys[t]))
87         ix = np.argmax(np.random.ranf(vocab_size), p)
88         x[0] = 0
89         x[i] = 1
90
91     return ix
92
93 n, p = 0, 0
94 mem, memh, memy = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(why)
95 memb, membhy = np.zeros_like(bh), np.zeros_like(why)
96 memby = np.zeros_like(by)
97 memh = np.zeros_like(hs[0])
98 seqLength = len(data)
99 seqLength += 1
100 seqLength = seqLength * seqLength + 1
101 while True:
102     if p > seqLength:
103         p -= seqLength
104     if p == seqLength:
105         print 'done'
106         break
107
108     inputs = [char_to_ix[ch] for ch in data[p:p+seqLength]]
109     targets = [char_to_ix[ch] for ch in data[p+seqLength+1]]
110
111     if p < seqLength:
112         a = np.zeros((hidden_size, 1))
113         a[0] = 1
114         a = np.tanh(a)
115         a = np.dot(Wxh, a) + np.dot(Whh, memh) + memb
116         a = np.dot(Why, a) + memby
117         a = np.exp(a) / np.sum(np.exp(a))
118         ix = np.argmax(a)
119         x[0] = 0
120         x[i] = 1
121
122     else:
123         a = np.zeros((hidden_size, 1))
124         a[0] = 1
125         a = np.tanh(a)
126         a = np.dot(Wxh, a) + np.dot(Whh, memh) + memb
127         a = np.dot(Why, a) + memby
128         a = np.exp(a) / np.sum(np.exp(a))
129         ix = np.argmax(a)
130
131     print 'predicted', ix, 'at', p
132
133     # Forward pass: compute gradients through the net and fetch gradient
134     loss, dch, dhb, dhy, dby, hprev = lossFun(inputs, targets, hprev)
135     smooth_loss = smooth_loss * 0.999 + loss * 0.001
136
137     if p % 100 == 0: print 'iter %d, loss: %f' % (p, smooth_loss)
138
139     # Backward pass: compute gradients through the net and fetch gradient
140     for opname, op in zip([dch, dhb, dhy, dby], [dch, dhb, dhy, dby]):
141         np.clip(op, -5, 5, out=op)
142         op -= op.mean()
143         op *= learning_rate
144         op = np.sqrt(op + 1e-8) # adaptive update
145         opname -= op
146
147     p += 1
148
149 n += 1
150 iteration counter

```

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

$$y_t = W_{hy}h_t$$

Softmax classifier

```

127 def lossFun(inputs, targets, hprev):
128     """
129         inputs,targets are both list of integers.
130         hprev is Hx1 array of initial hidden state
131         returns the loss, gradients on model parameters, and last hidden state
132     """
133
134     xs, hs, ys, ps = {}, {}, {}, {}
135     hs[-1] = np.copy(hprev)
136     loss = 0
137
138     # forward pass
139     for t in xrange(len(inputs)):
140         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
141         xs[t][inputs[t]] = 1
142
143         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
144
145         ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
146         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
147
148         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
149
150     # backward pass: compute gradients going backwards
151     dch, dhb, dhy = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(why)
152     dby = np.zeros_like(by)
153     dhnext = np.zeros_like(hs[0])
154
155     for t in reversed(xrange(len(inputs)))�
156         dy = np.copy(ps[t])
157         dy[targets[t]] -= 1 # backprop into y
158         dh = np.dot(why.T, dy) + dhnext # dh = backprop through tanh nonlinearity
159
160         dhx = np.dot(Wxh.T, dh) # dh = backprop through dot product
161         dch += np.dot(dhx, xs[t].T) # dch = backprop through dot product
162         dby += np.sum(dhx, axis=1, keepdims=True) # dby = backprop through dot product
163         dhb += np.sum(dhx, axis=0, keepdims=True) # dhb = backprop through dot product
164
165         dhnext = np.tanh(dch) * dh # dhnext = gradient flow into next step
166
167     for opname in [dch, dhb, dhy, dby]:
168         np.clip(opname, -5, 5, out=opname) # clip to mitigate exploding gradients
169         opname -= opname.mean() # renormalize gradients
170         opname *= learning_rate # update rule
171
172     return loss, hs[-1]
173
174 def sample(hprev, seed_ix, n):
175     """  
176         sample a sequence of integers from the model  
177         h is memory state, seed_ix is seed integer for first time step  
178         n is max length of sampled sequence  
179     """
180     x = np.zeros((vocab_size, 1))
181     x[seed_ix] = 1
182
183     for t in xrange(n):
184         h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
185         p = np.exp(ys[t]) / np.sum(np.exp(ys[t]))
186         ix = np.argmax(np.random.ranf(vocab_size), p)
187         x[0] = 0
188         x[i] = 1
189
190     return ix
191
192 n, p = 0, 0
193 mem, memh, memy = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(why)
194 memb, membhy = np.zeros_like(bh), np.zeros_like(why)
195 memby = np.zeros_like(by)
196 memh = np.zeros_like(hs[0])
197 seqLength = len(data)
198 seqLength += 1
199 seqLength = seqLength * seqLength + 1
200 while True:
201     if p > seqLength:
202         p -= seqLength
203     if p == seqLength:
204         print 'done'
205         break
206
207     inputs = [char_to_ix[ch] for ch in data[p:p+seqLength]]
208     targets = [char_to_ix[ch] for ch in data[p+seqLength+1]]
209
210     if p < seqLength:
211         a = np.zeros((hidden_size, 1))
212         a[0] = 1
213         a = np.tanh(a)
214         a = np.dot(Wxh, a) + np.dot(Whh, memh) + memb
215         a = np.dot(Why, a) + memby
216         a = np.exp(a) / np.sum(np.exp(a))
217         ix = np.argmax(a)
218         x[0] = 0
219         x[i] = 1
220
221     else:
222         a = np.zeros((hidden_size, 1))
223         a[0] = 1
224         a = np.tanh(a)
225         a = np.dot(Wxh, a) + np.dot(Whh, memh) + memb
226         a = np.dot(Why, a) + memby
227         a = np.exp(a) / np.sum(np.exp(a))
228         ix = np.argmax(a)
229
230     print 'predicted', ix, 'at', p
231
232     # Forward pass: compute gradients through the net and fetch gradient
233     loss, dch, dhb, dhy, dby, hprev = lossFun(inputs, targets, hprev)
234     smooth_loss = smooth_loss * 0.999 + loss * 0.001
235
236     if p % 100 == 0: print 'iter %d, loss: %f' % (p, smooth_loss)
237
238     # Backward pass: compute gradients through the net and fetch gradient
239     for opname, op in zip([dch, dhb, dhy, dby], [dch, dhb, dhy, dby]):
240         np.clip(op, -5, 5, out=op)
241         op -= op.mean()
242         op *= learning_rate
243         op = np.sqrt(op + 1e-8) # adaptive update
244         opname -= op
245
246     p += 1
247
248 n += 1
249 iteration counter

```

min-char-rnn.py gist

```

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

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

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

25 def lossFun(inputs, targets, hprev):
26     """ inputs,targets are both lists of integers.
27     hprev is Hx1 array of initial hidden state
28     returns the loss, gradients on model parameters, and last hidden state
29     """
30
31     xs, hs, ys, ps = [], [], [], []
32     h0 = np.copy(hprev)
33     loss = 0
34     for t in xrange(len(inputs)):
35         x = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
36         x[inputs[t], 0] = 1
37         h0 = np.tanh(np.dot(wh, h0) + np.dot(bh, h0[-1]) + bh)
38         y = np.dot(why.T, h0) + by
39         yprob = np.exp(y) / np.sum(np.exp(y)) # probabilities for next chars
40         loss += -np.log(y[targets[t]] / np.sum(np.exp(y)))
41         loss += -np.log(ps[targets[t], 0]) # softmax (cross-entropy loss)
42
43         dnh, dwh, dbh, dhy = np.zeros_like(wh), np.zeros_like(wh), np.zeros_like(bh),
44         dhy = np.zeros_like(by), dby = np.zeros_like(by)
45         dhnext = np.zeros_like(h0)
46         for i in xrange(vocab_size):
47             dy = np.copy(ps[i])
48             dy[targets[t]] -= 1 # backprop into y
49             dhy += np.dot(dy, why.T)
50             dby += dy
51             dh = np.dot(why, dy) + dhnext # backprop through tanh nonlinearity
52             dnh += np.dot(dh, h0.T)
53             dwh += np.dot(dh, xs[t].T)
54             dbh += np.sum(dh, axis=0, keepdims=True)
55             dhnext = np.dot(wh.T, dh)
56
57         dwh += np.dot(dhraw, xs[t].T)
58         dbh += np.dot(dhraw, hs[t-1].T)
59         dhnext = np.dot(whh.T, dhraw)
60
61     for dparam in [dwh, dbh, dhy, dby]:
62         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
63
64     return loss, dwh, dbh, dhy, dby, hs[len(inputs)-1]
65
66 def sample(prev_h, ix):
67     """ sample a sequence of integers from the model
68     h is memory state, seed_ix is seed letter for first time step
69     """
70
71     x = np.zeros((vocab_size, 1))
72     x[seed_ix] = 1
73
74     for t in xrange(n):
75         h = np.tanh(np.dot(wh, x) + np.dot(bh, h) + bh)
76         y = np.dot(why.T, h) + by
77         p = np.exp(y) / np.sum(np.exp(y))
78         ix = np.random.choice(range(vocab_size), p=p.ravel())
79         x[0, ix] = 1
80
81     return ix
82
83 n, p = 0
84
85 mem, m0, mb0, mhy = np.zeros_like(wh), np.zeros_like(wh), np.zeros_like(bh),
86 mem0, mb0, mhy0 = np.zeros_like(bh), np.zeros_like(bh), np.zeros_like(by)
87 smooth_loss = 0
88 smooth_loss_n = 0
89
90 while True:
91     if p == seq_length or ix == 0:
92         p = np.random.randint(1, len(chars))
93         sample_ix = sample(prev_h, inputs[p], 200)
94         print ''.join(ix_to_char[x] for x in sample_ix)
95
96     else:
97         # forward pass: compute gradients going backwards
98         loss, dwh, dbh, dhy, dby, hprev = lossFun(inputs, targets, prev_h)
99
100         smooth_loss += smooth_loss * 0.999 + loss * 0.001
101         if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss)
102
103         for param, mem in zip([wh, bh, why, by],
104                               [dwh, dbh, dhy, dby],
105                               [mem, memh, memb, memhy]):
106             mem += param * learning_rate * dparam / np.sqrt(mem + 1e-8) # adaptive update
107
108         p += 1 # move data pointer
109
110     n += 1 # iteration counter

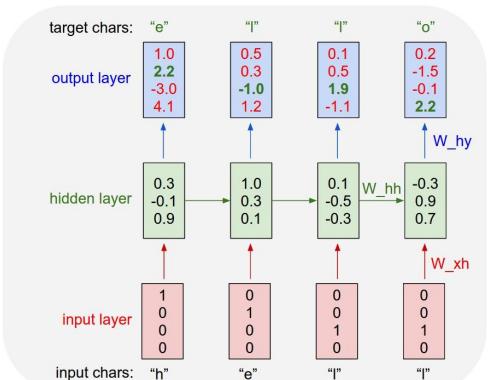
```

```

44 # backward pass: compute gradients going backwards
45 dwxh, dwhh, dwhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
46 dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47 dhnext = np.zeros_like(hs[0])
48 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     dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55     dbh += dhraw
56     dwh += np.dot(dhraw, xs[t].T)
57     dbh += np.dot(dhraw, hs[t-1].T)
58     dhnext = np.dot(Whh.T, dhraw)
59
60 for dparam in [dwxh, dwhh, dwhy, dbh, dby]:
61     np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
62
63 return loss, dwxh, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]

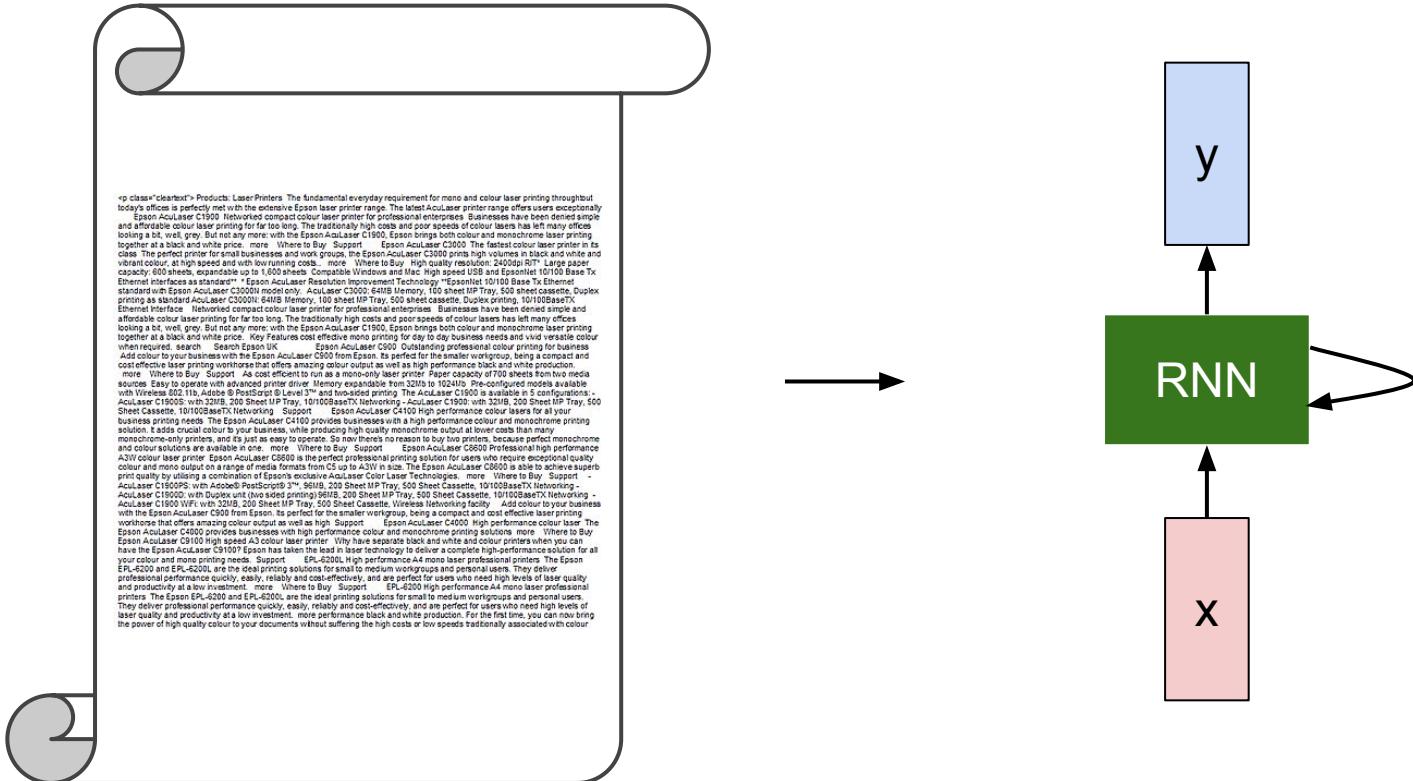
```

recall:



min-char-rnn.py gist

```
def sample(h, seed_ix, n):
    """
    sample a sequence of integers from the model
    h is memory state, seed_ix is seed letter for first time step
    """
    x = np.zeros((vocab_size, 1))
    x[seed_ix] = 1
    ixes = []
    for t in xrange(n):
        h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
        y = np.dot(Why, h) + by
        p = np.exp(y) / np.sum(np.exp(y))
        ix = np.random.choice(range(vocab_size), p=p.ravel())
        x = np.zeros((vocab_size, 1))
        x[ix] = 1
        ixes.append(ix)
    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-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tkldrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

↓ train more

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

↓ train more

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

↓ train more

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

PANDARUS:

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

Second Senator:

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

DUKE VINCENTIO:

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

Second Lord:

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

Clown:

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

VIOLA:

I'll drink it.

VIOLA:

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

KING LEAR:

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

open source textbook on algebraic geometry

The Stacks Project

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

Browse chapters

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

Parts

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

Statistics

The Stacks project now consists of

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

Latex source

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

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

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

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

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

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

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

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

is a unique morphism of algebraic stacks. Note that

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

and

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

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

Proof. See discussion of sheaves of sets. \square

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

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

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

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

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

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

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

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

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

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

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

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

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

Proof. Omitted. □

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

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

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

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

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

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

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

Proof. See Spaces, Lemma ??.

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

The following to the construction of the lemma follows.

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

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

be a morphism of algebraic spaces over S and Y .

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

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

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

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

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

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 . □

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

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

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

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

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

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

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

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

Linux kernel source tree

520,037 commits

1 branch

420 releases

5,039 contributors

branch: master · [linux](#) / +

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

 torvalds authored 9 hours agolatest commit 4b1786927d  Documentation

Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending

6 days ago

 arch

Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l...

a day ago

 block

block: discard bdi_unregister() in favour of bdi_destroy()

9 days ago

 crypto

Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6

10 days ago

 drivers

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

9 hours ago

 firmware

firmware/hex2fw.c: restore missing default in switch statement

2 months ago

 fs

vfs: read file_handle only once in handle_to_path

4 days ago

 include

Merge branch 'perl-urgent-for-linus' of git://git.kernel.org/pub/scm/...

a day ago

 init

init: fix regression by supporting devices with major:minor:offset fo...

a month ago

 io

allowance: fix race between writeback and multi-thread traversal when doing unlinked list removal

a month ago

 Code Pull requests
74 Pulse Graphs

HTTPS clone URL

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

```
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffffff8) & 0x000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

Generated C code

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

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

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

```

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

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

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

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

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

```

Searching for interpretable cells

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

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

Searching for interpretable cells

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

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

quote detection cell

Searching for interpretable cells

Cell sensitive to position in line:

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

line length tracking cell

Searching for interpretable cells

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

if statement cell

Searching for interpretable cells

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

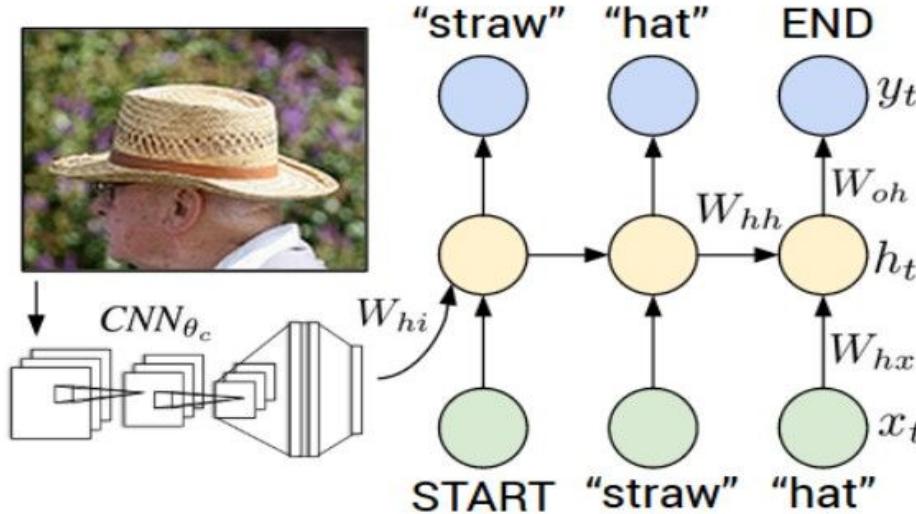
quote/comment cell

Searching for interpretable cells

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

code depth cell

Image Captioning



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

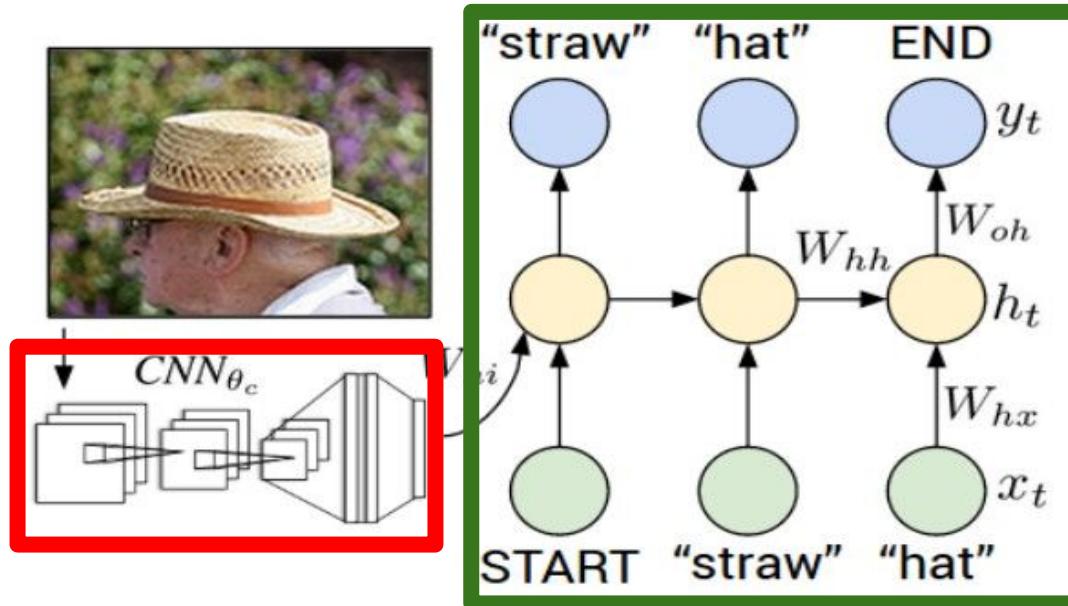
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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

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

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

Recurrent Neural Network



Convolutional Neural Network

test image



image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax

image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

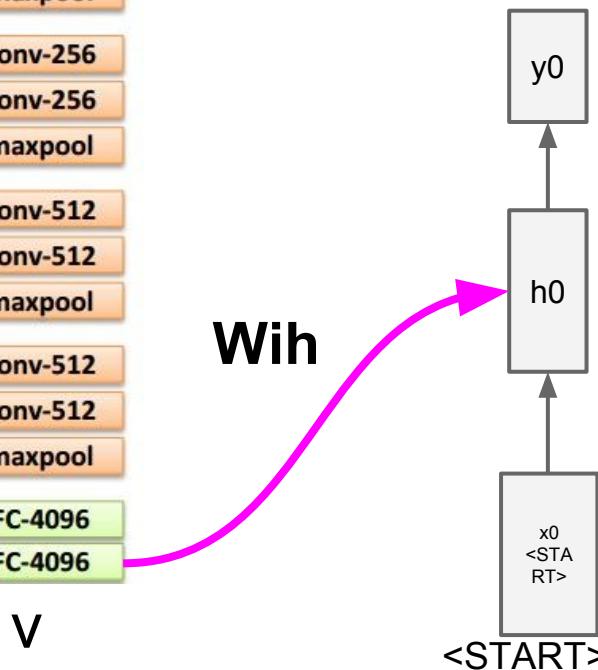
FC-4096



<START>



test image



before:

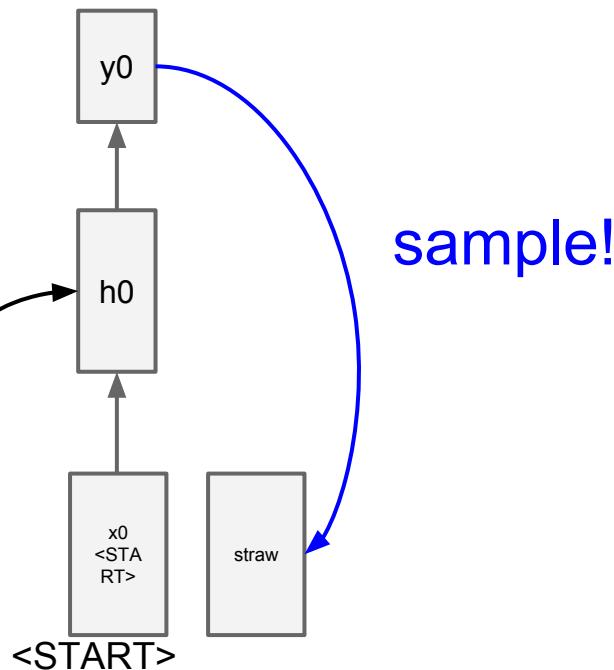
$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

now:

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



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

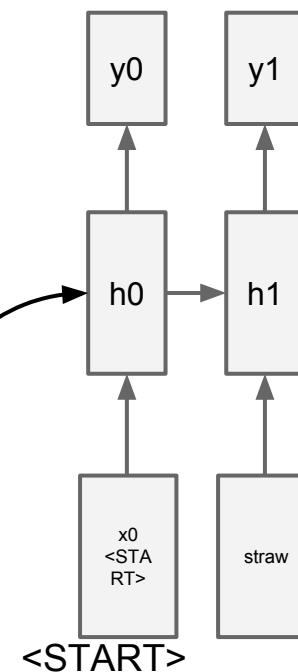
maxpool

FC-4096

FC-4096

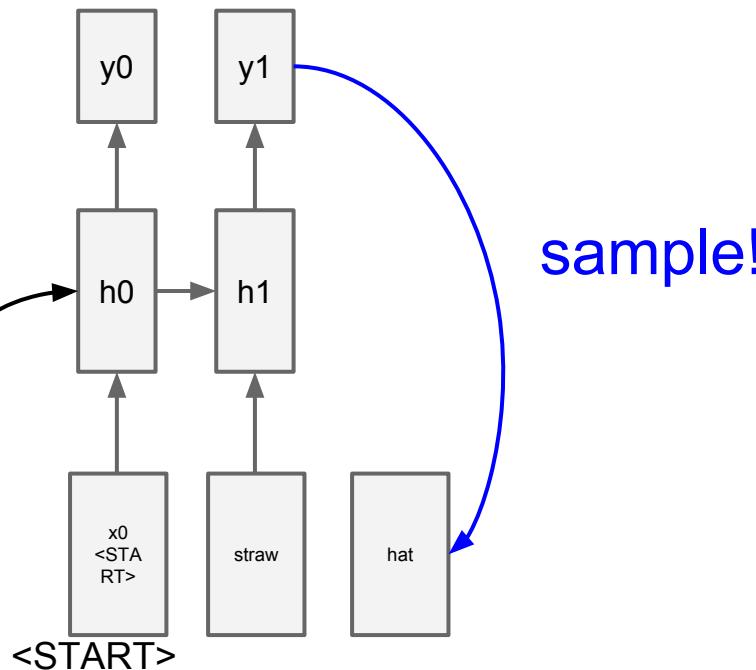


test image





test image



image



test image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

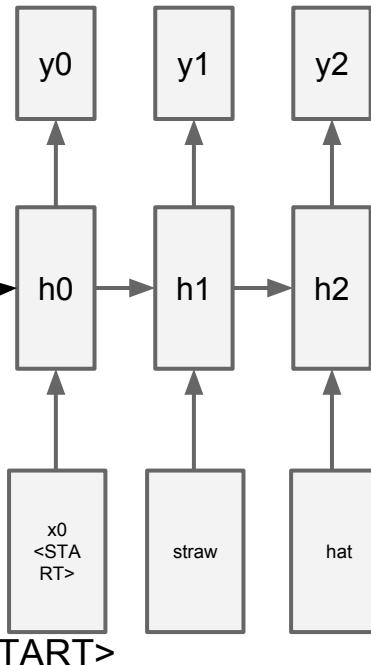
conv-512

conv-512

maxpool

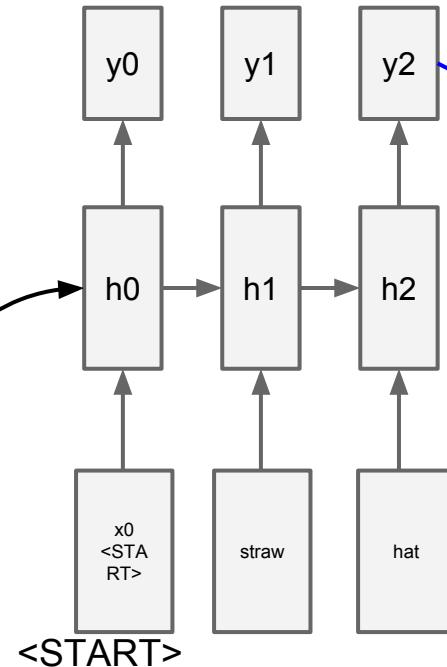
FC-4096

FC-4096





test image



sample
<END> token
=> finish.

Image Sentence Datasets

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



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

currently:
~120K images
~5 sentences each



"man in black shirt is playing guitar."



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



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"man in black shirt is playing guitar."



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



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"a young boy is holding a baseball bat."



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



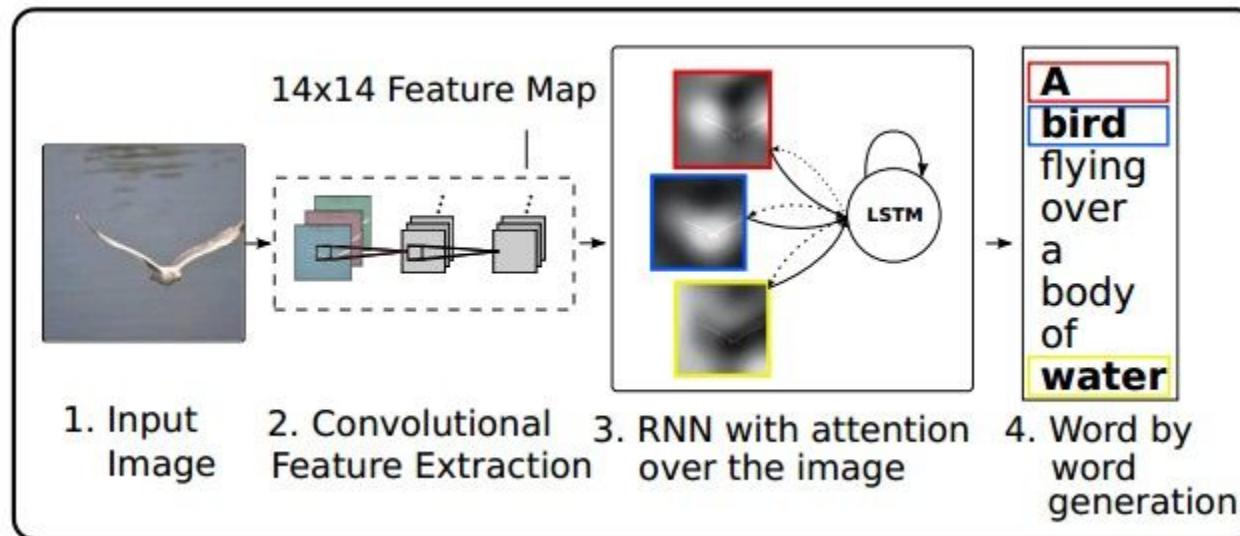
"a woman holding a teddy bear in front of a mirror."



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

Preview of fancier architectures

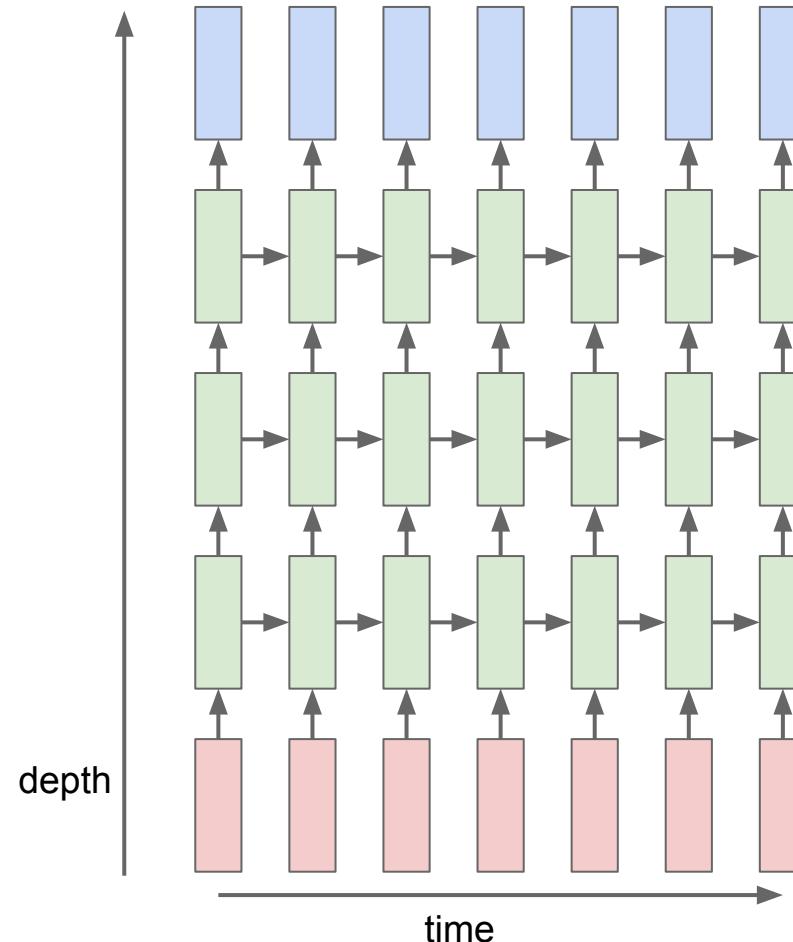
RNN attends spatially to different parts of images while generating each word of the sentence:



RNN:

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

$h \in \mathbb{R}^n$. W^l [n × 2n]



RNN:

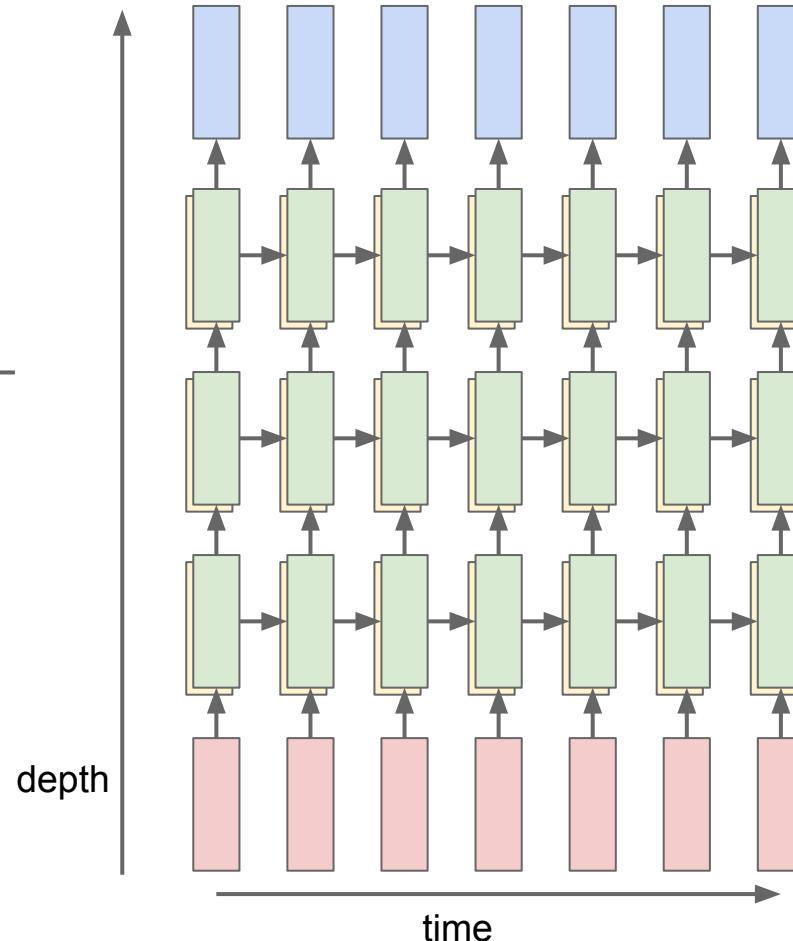
$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

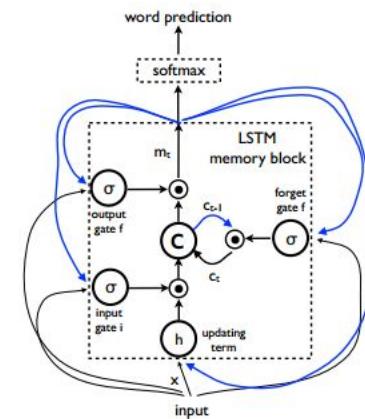
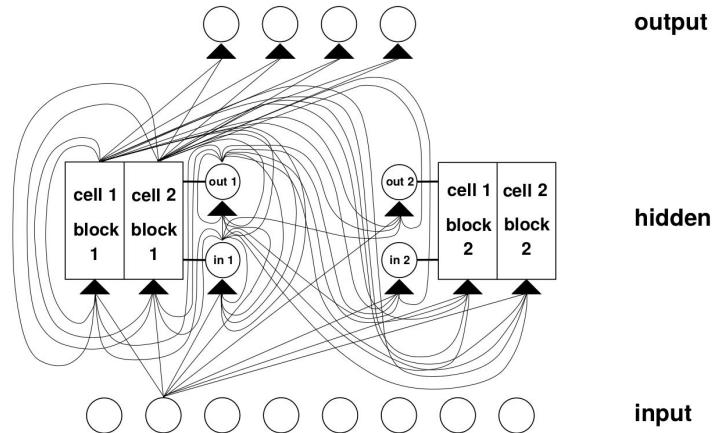
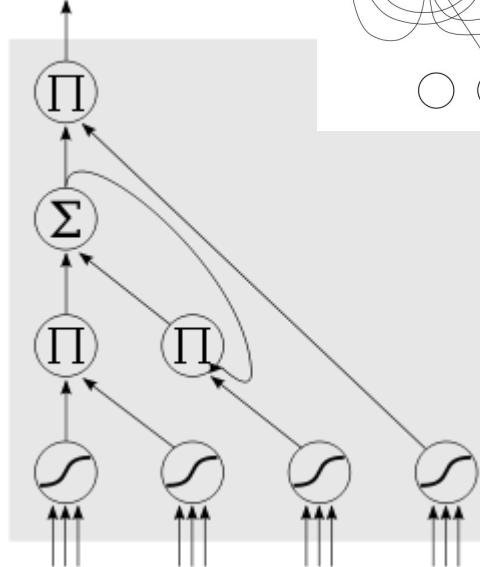
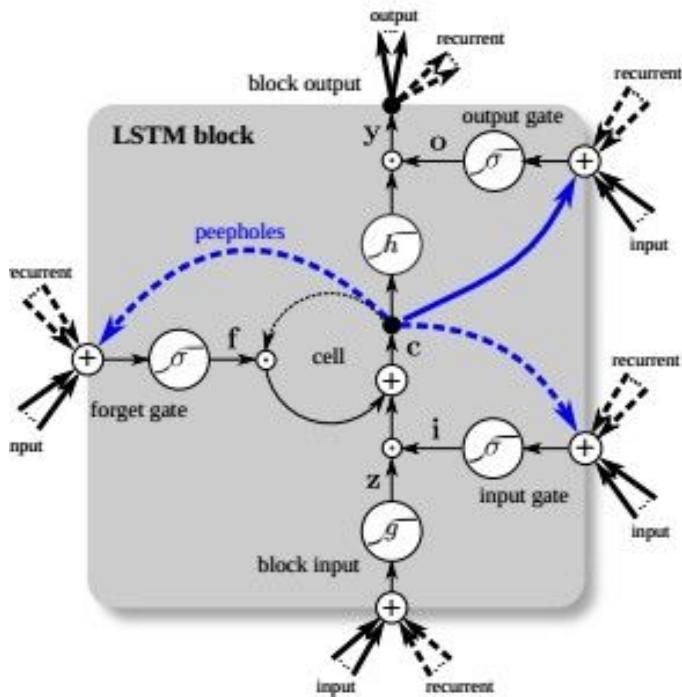
LSTM:

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

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

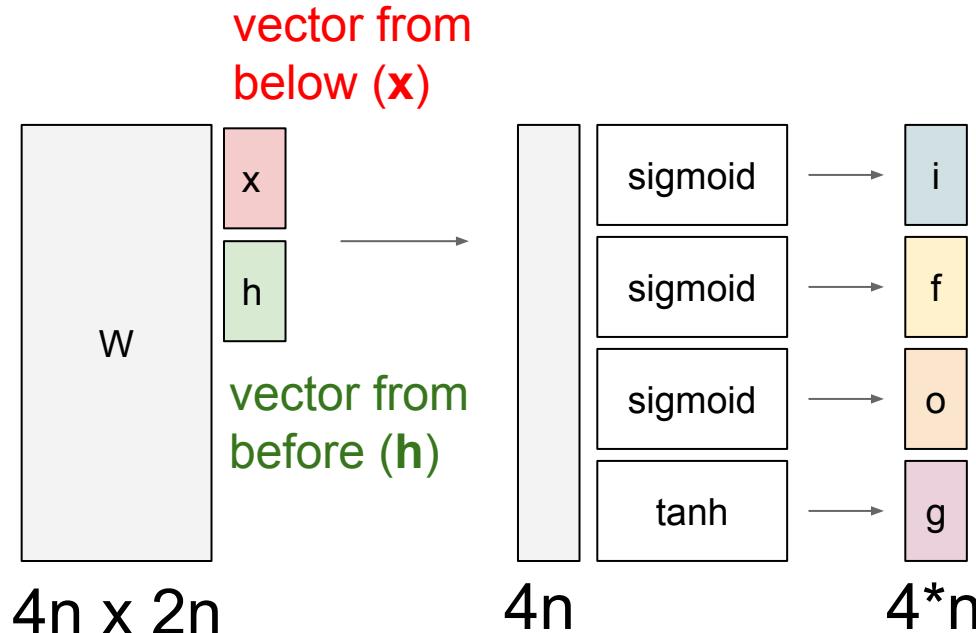


LSTM



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

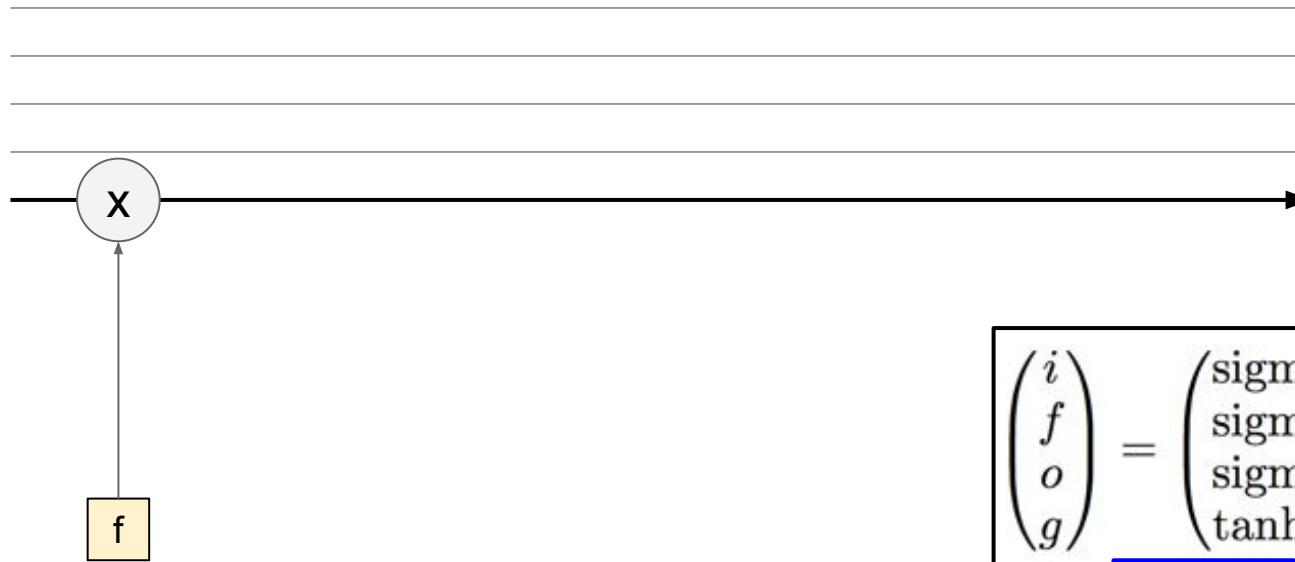


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_t^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state **c**

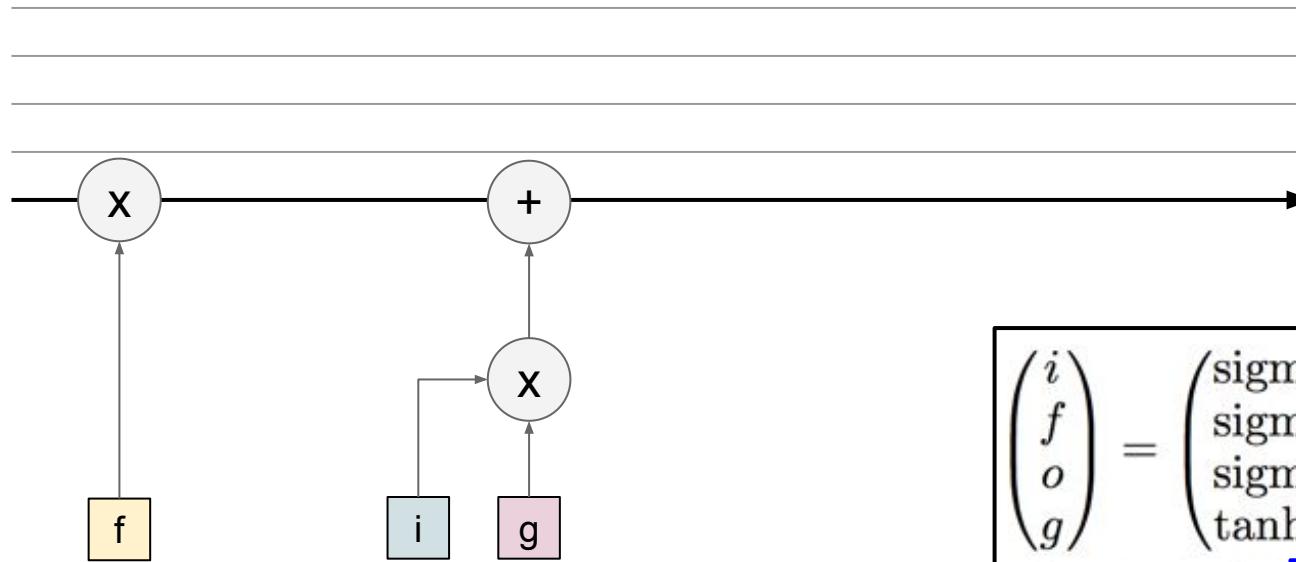


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c

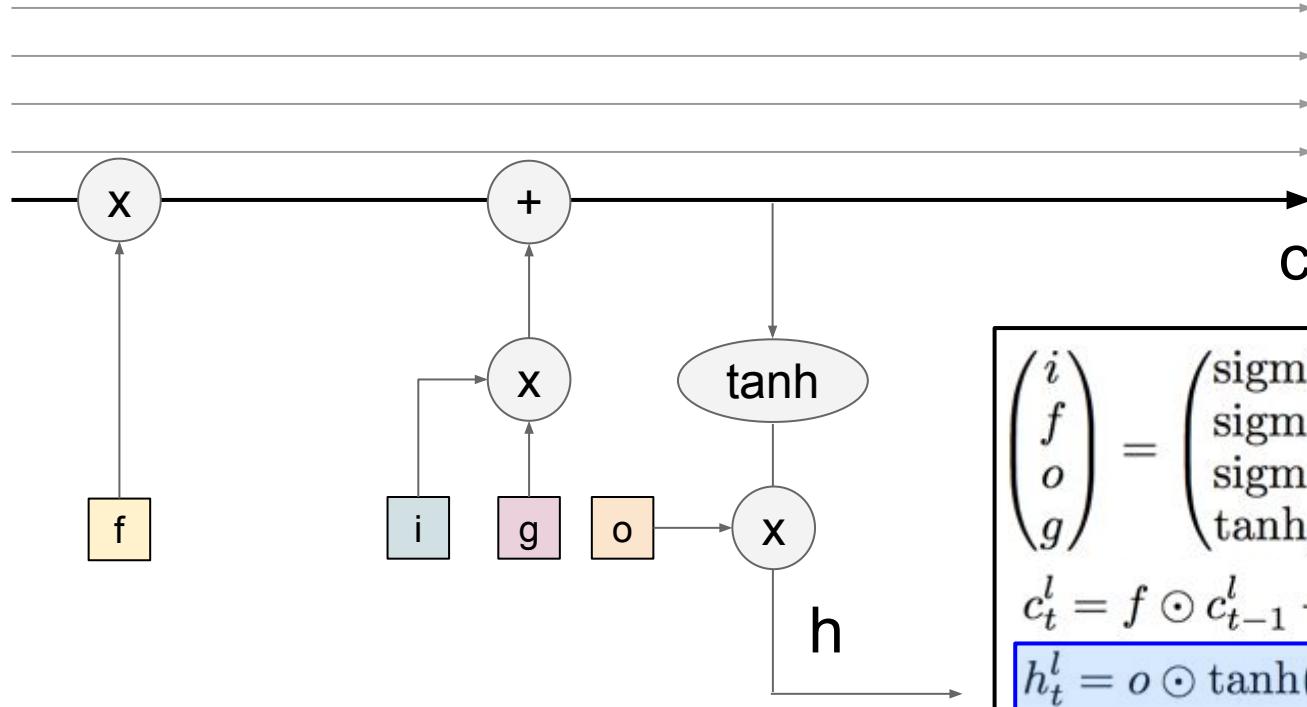


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c

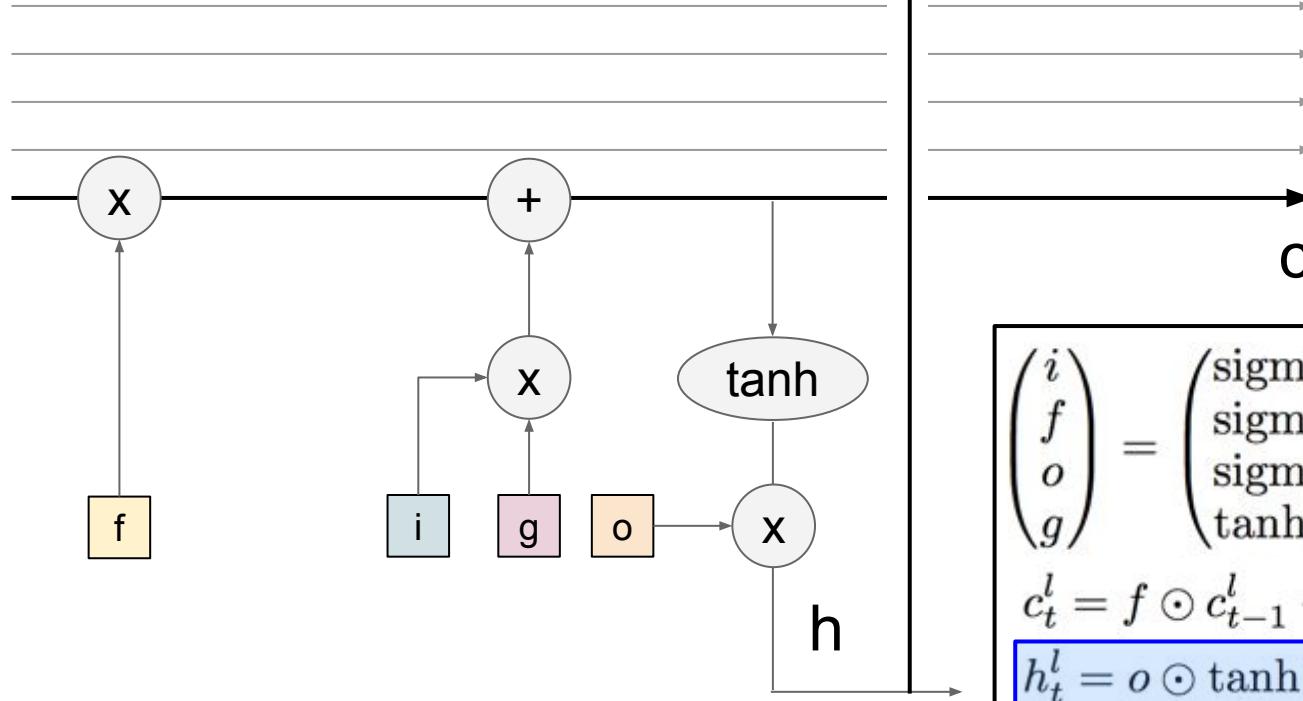


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell
state c

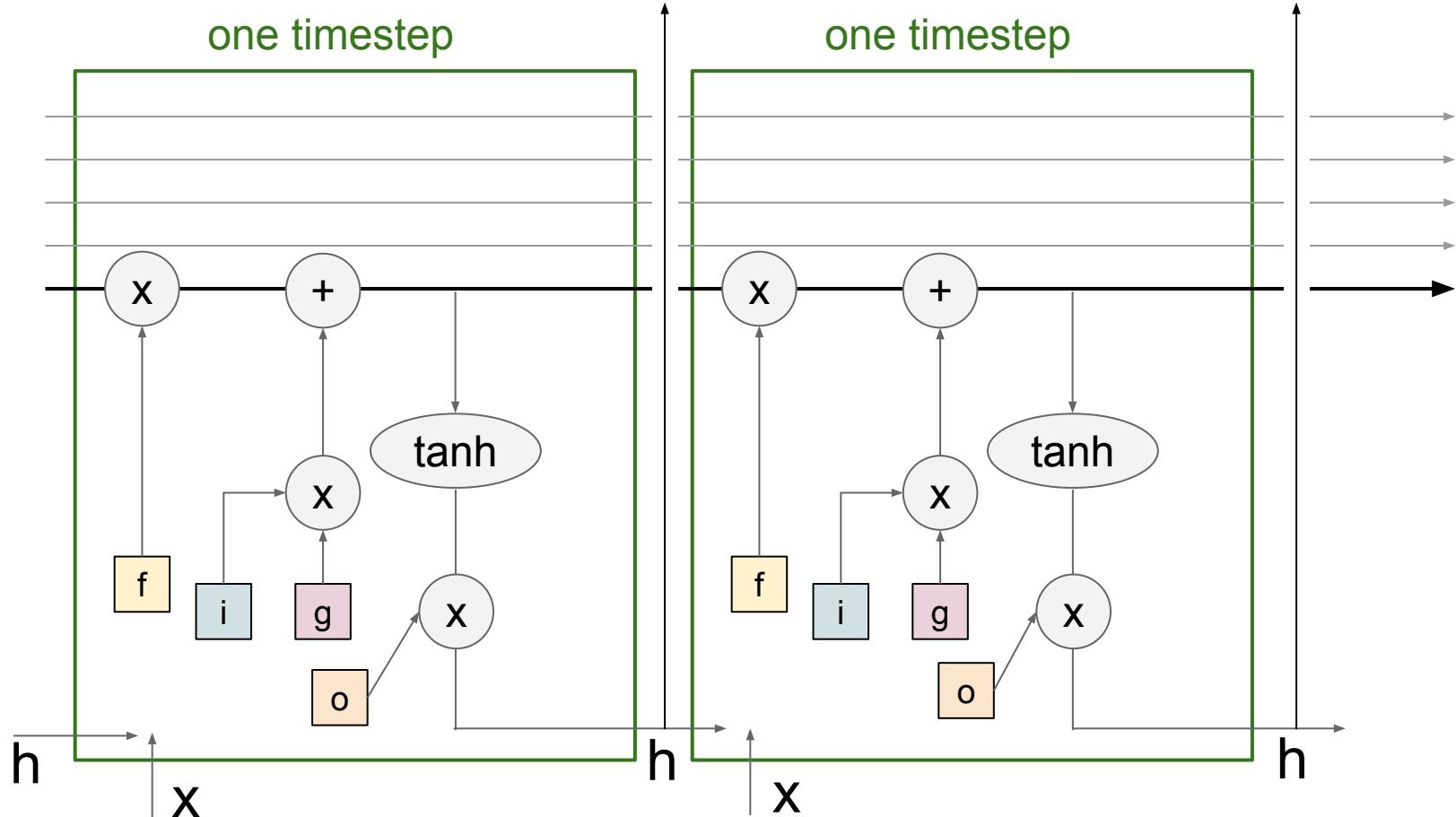


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

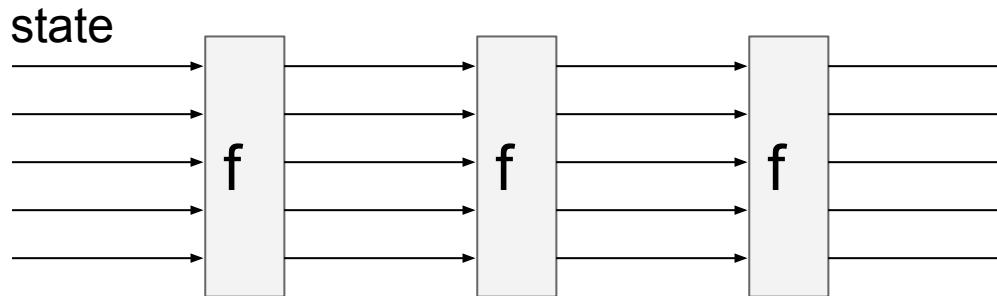
LSTM

one timestep

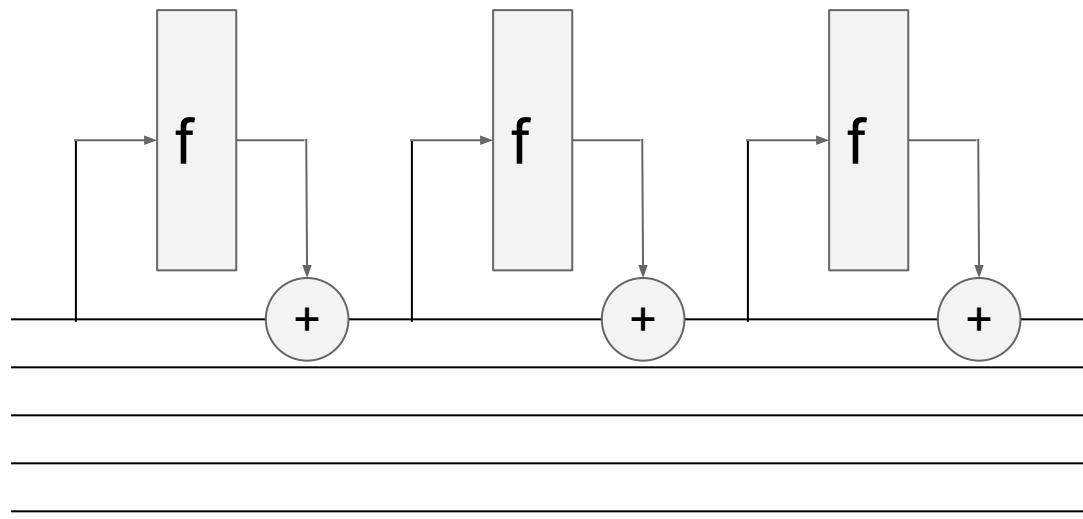
cell
state c



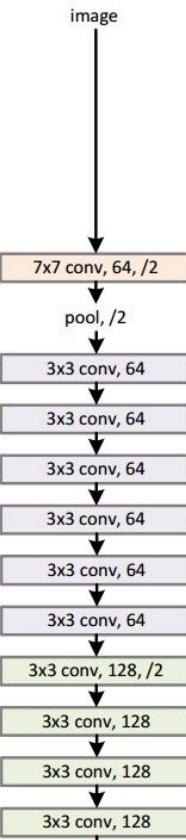
RNN



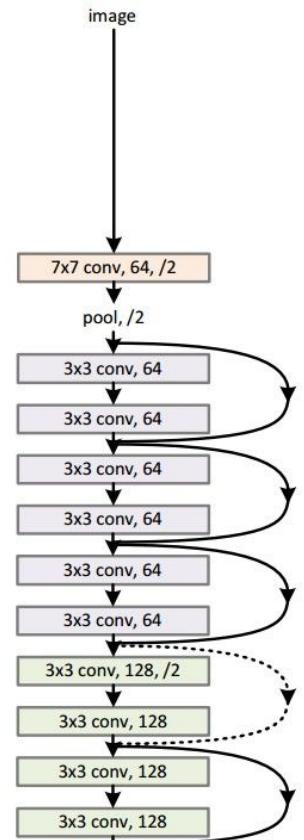
LSTM (ignoring forget gates)



34-layer plain

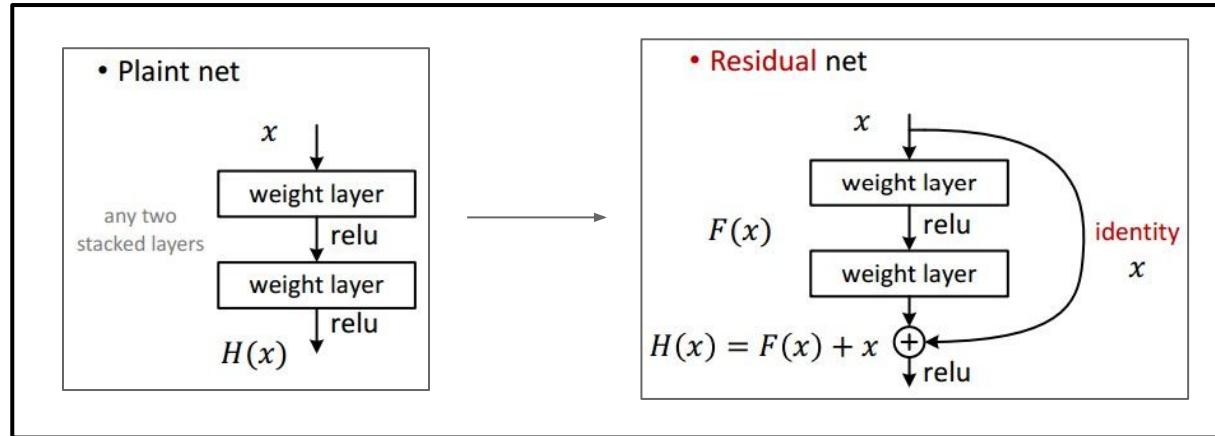


34-layer residual



Recall: “PlainNets” vs. ResNets

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



Understanding gradient flow dynamics

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

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

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

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

Understanding gradient flow dynamics

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

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

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

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

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

Understanding gradient flow dynamics

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

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

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

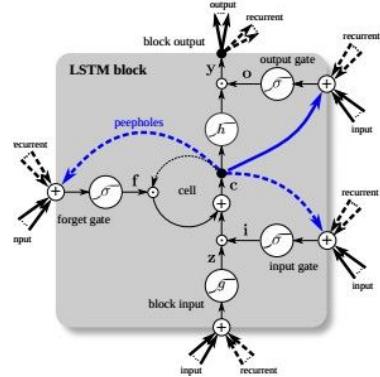
if the largest eigenvalue is > 1 , gradient will explode
if the largest eigenvalue is < 1 , gradient will vanish

can control exploding with gradient clipping
can control vanishing with LSTM

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

LSTM variants and friends

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



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

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

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

MUT1:

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

MUT2:

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

MUT3:

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

Summary

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