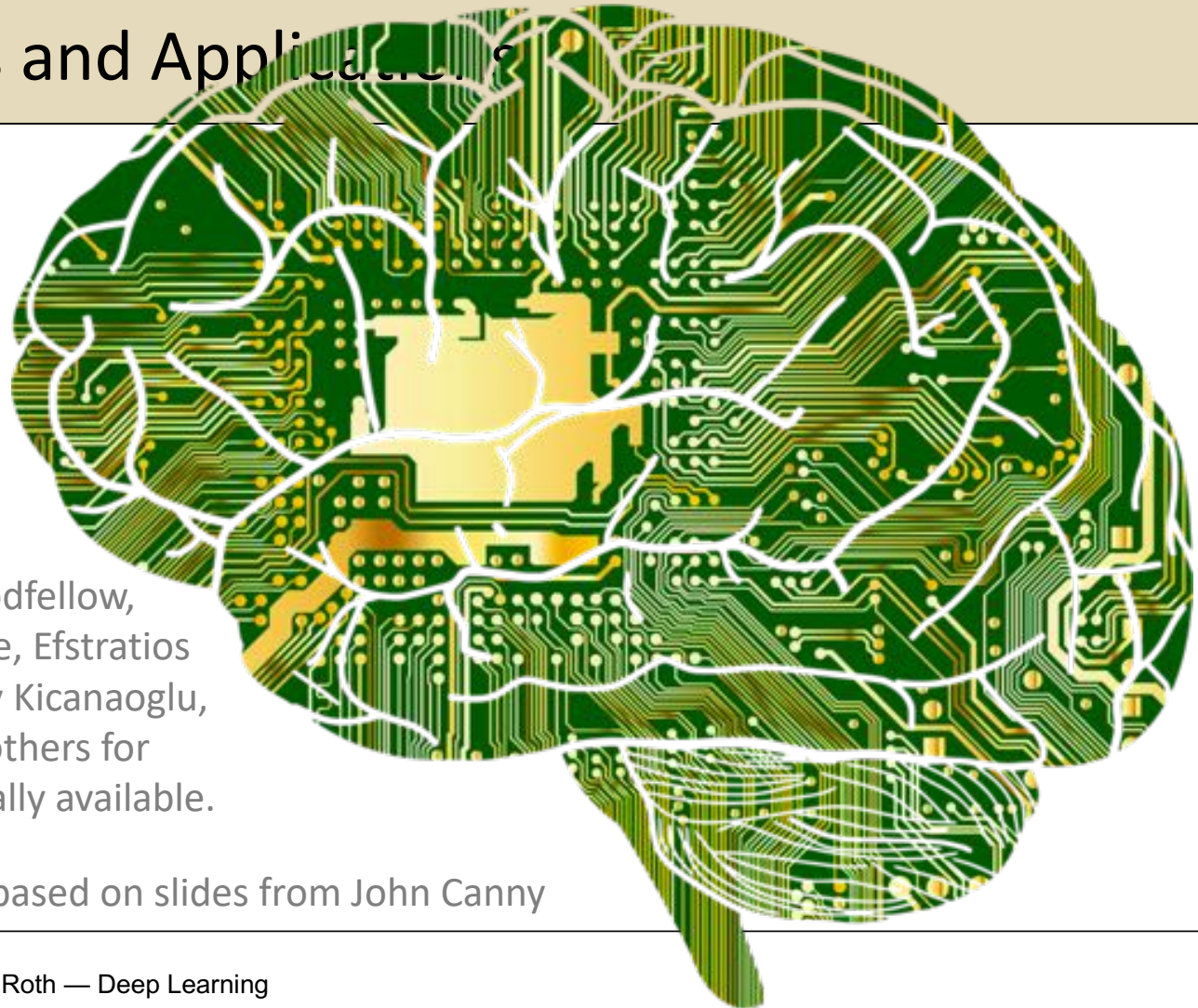


# Deep Learning



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## Architectures and Methods: Recurrent Networks, LSTMs and Applications

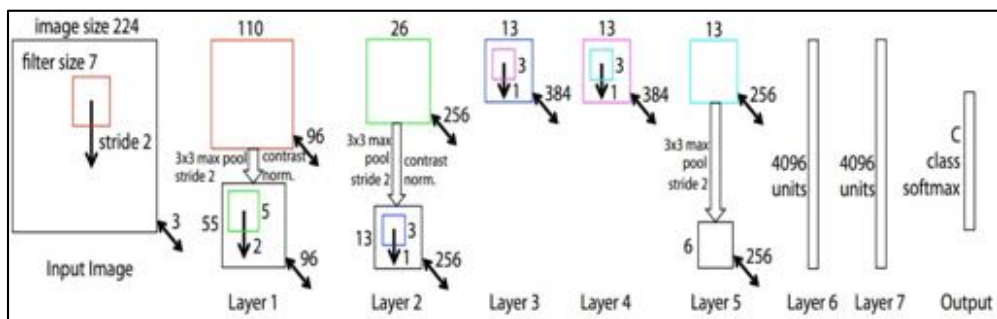
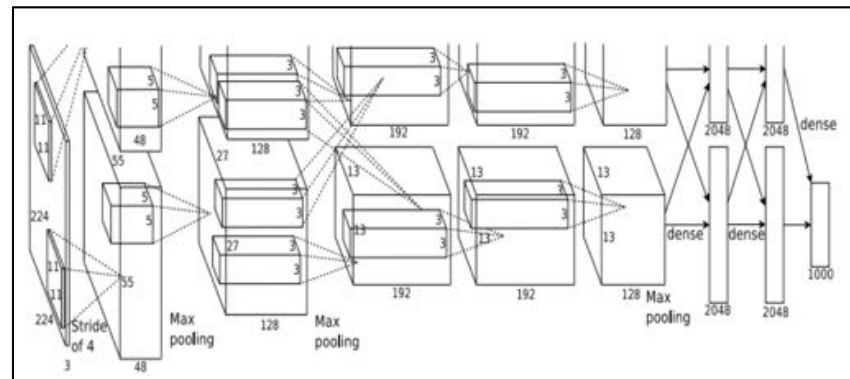


Thanks to John Canny, Ian Goodfellow, Yoshua Bengio, Aaron Courville, Efstratios Gavves, Kirill Gavrilyuk, Berkay Kicanaoglu, and Patrick Putzky and many others for making their materials publically available.

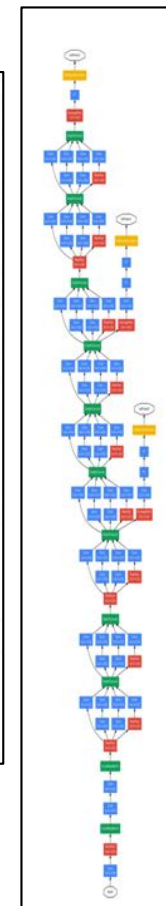
The present slides are mainly based on slides from John Canny



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



|    | D                                          | E                                                       |
|----|--------------------------------------------|---------------------------------------------------------|
|    | 16 weight layers                           | 19 weight layers                                        |
| c) | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64                                    |
|    | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128                                  |
|    | conv3-256<br>conv3-256<br><b>conv3-256</b> | conv3-256<br>conv3-256<br>conv3-256<br><b>conv3-256</b> |
|    | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br>conv3-512<br><b>conv3-512</b> |
|    | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br>conv3-512<br><b>conv3-512</b> |
|    | maxpool                                    |                                                         |
|    | FC-4096                                    |                                                         |
|    | FC-4096                                    |                                                         |
|    | FC-1000                                    |                                                         |
|    | soft-max                                   |                                                         |



# Last time: Localization and Detection

Classification



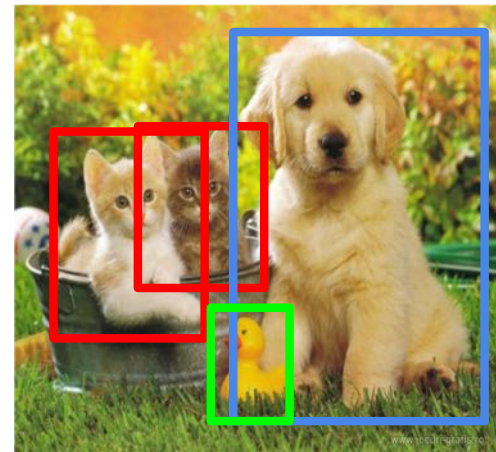
CAT

Classification +  
Localization



CAT

Object Detection



CAT, DOG, DUCK

Instance  
Segmentation



CAT, DOG, DUCK

Single  
object

Multiple  
objects

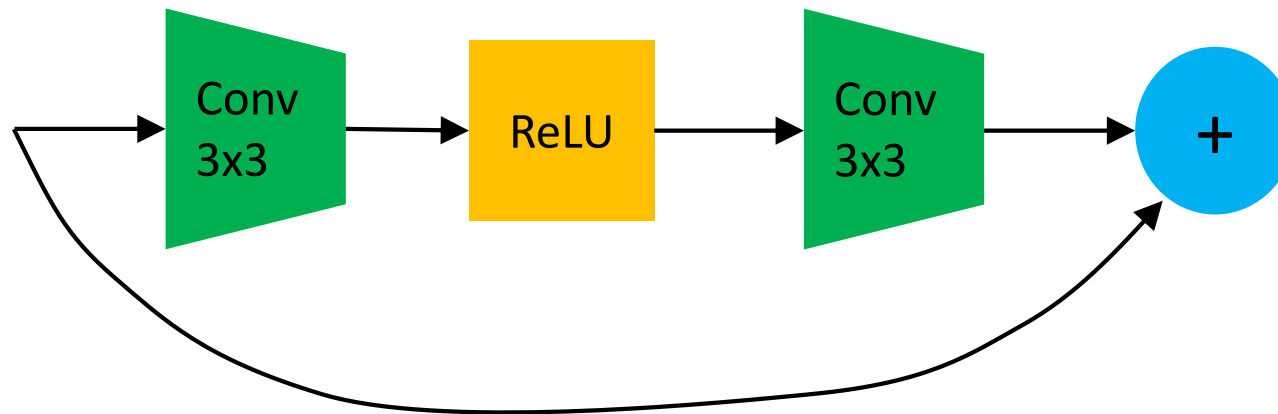




# Neural Network structure

Standard Neural Networks are DAGs (Directed Acyclic Graphs). That means they have a topological ordering.

- The topological ordering is used for activation propagation, and for gradient back-propagation.



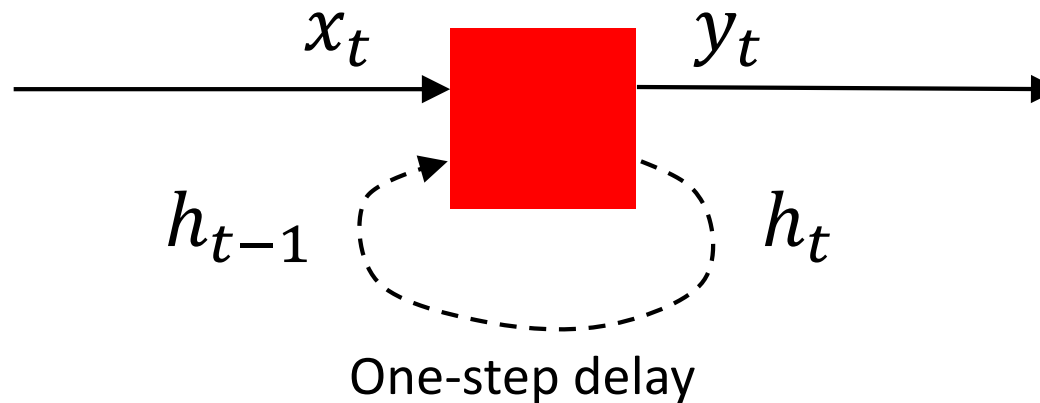
- They process one input instance at a time.





# Recurrent Neural Networks (RNNs)

Recurrent networks introduce cycles and a notion of time.

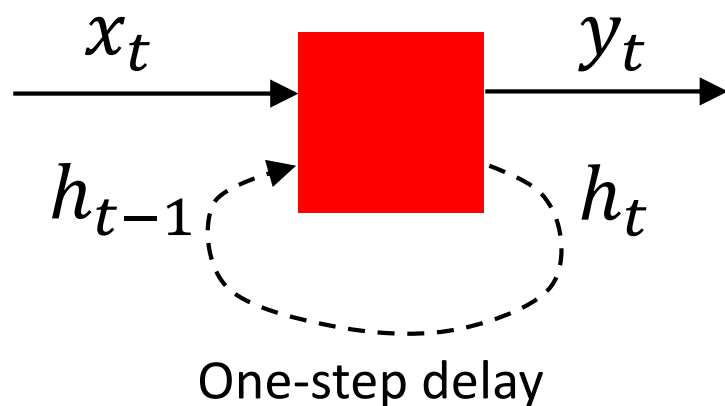


- They are designed to process sequences of data  $x_1, \dots, x_n$  and can produce sequences of outputs  $y_1, \dots, y_m$ .



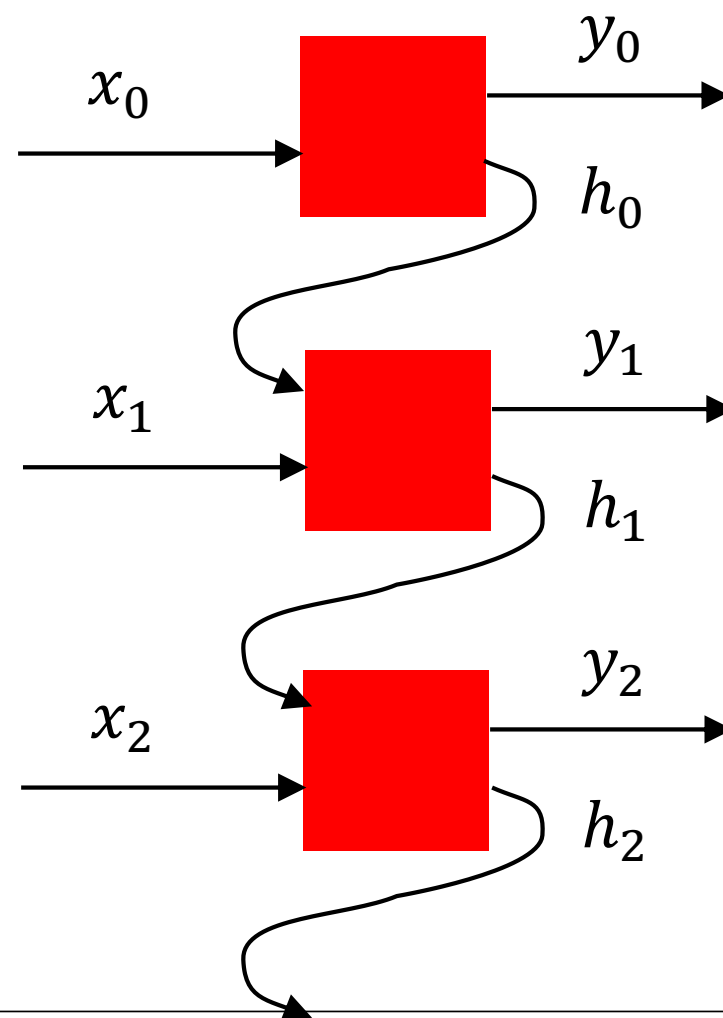
# Unrolling RNNs

RNNs can be unrolled across multiple time steps.



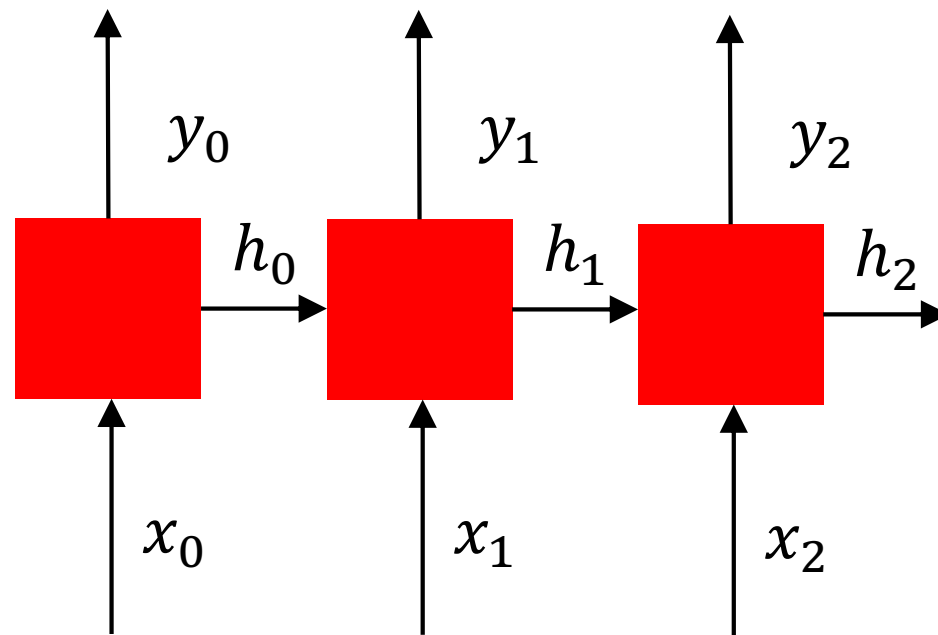
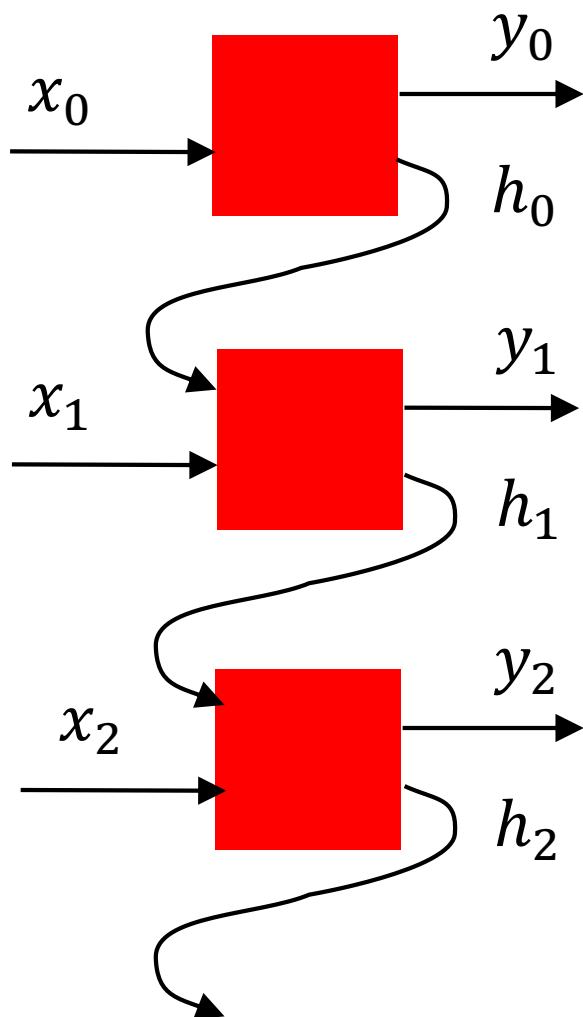
This produces a DAG which supports backpropagation.

But its size depends on the input sequence length.



# Unrolling RNNs

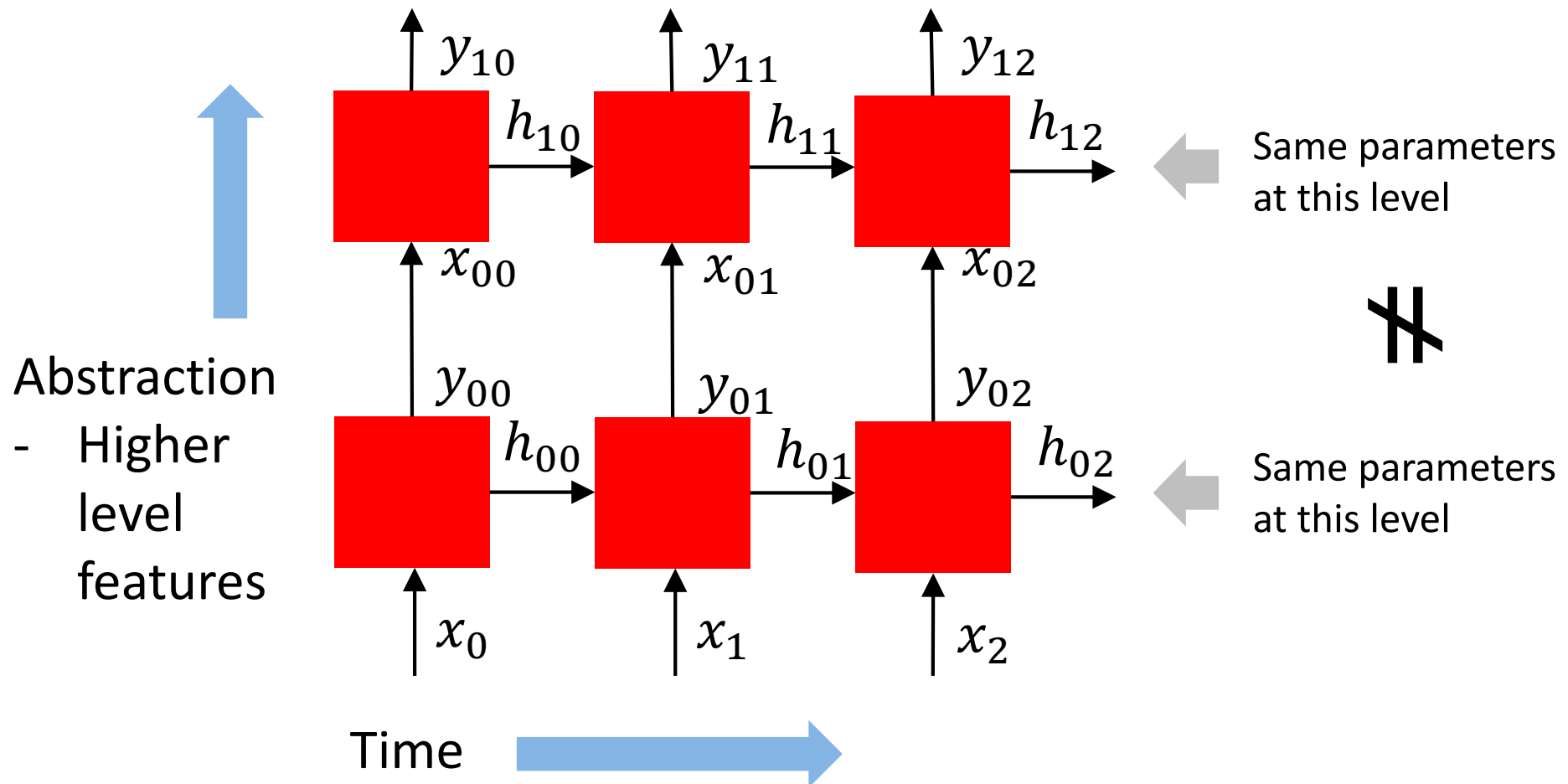
Usually drawn as:





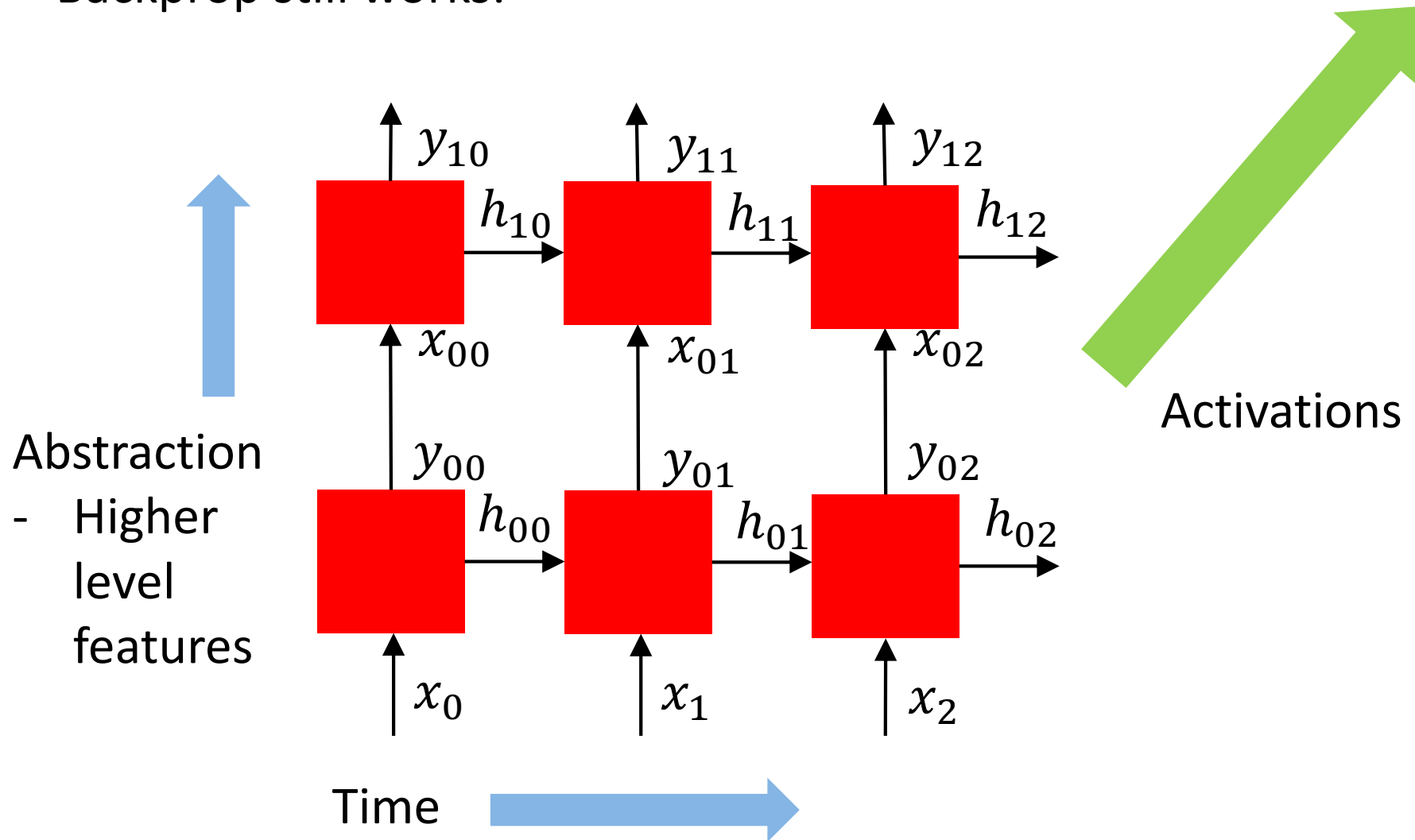
# RNN structure

Often layers are stacked vertically (deep RNNs):



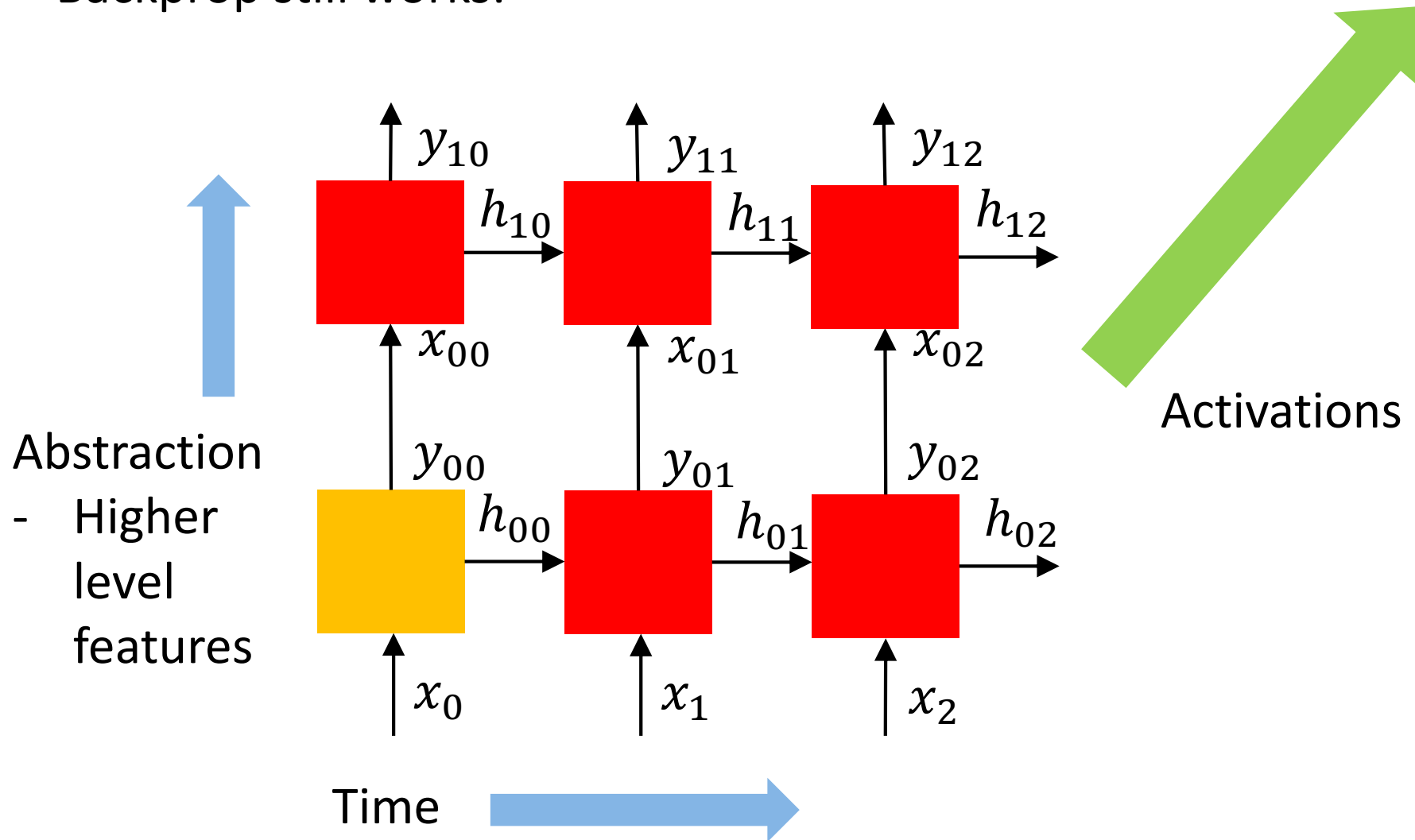
# RNN structure

Backprop still works:



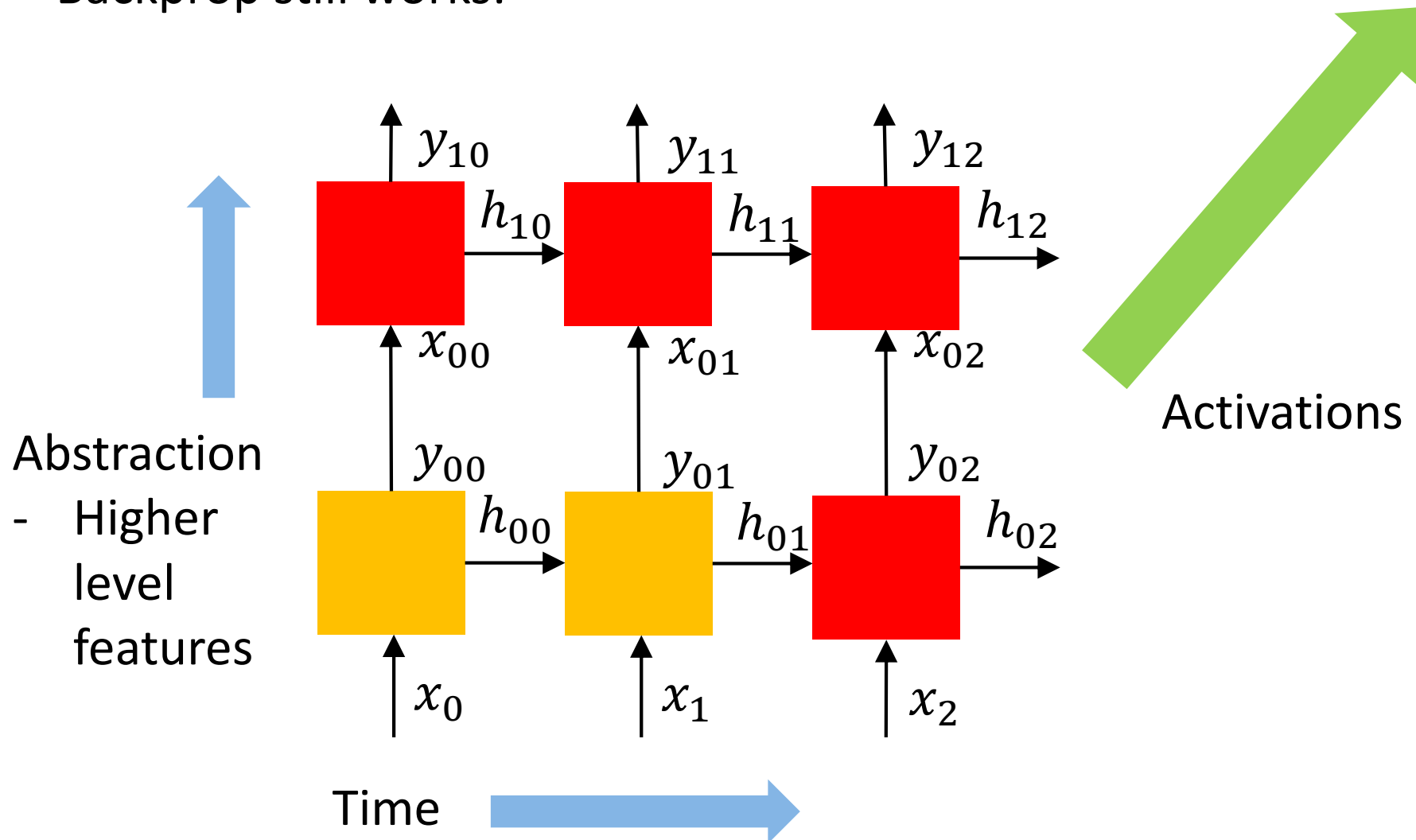
# RNN structure

Backprop still works:



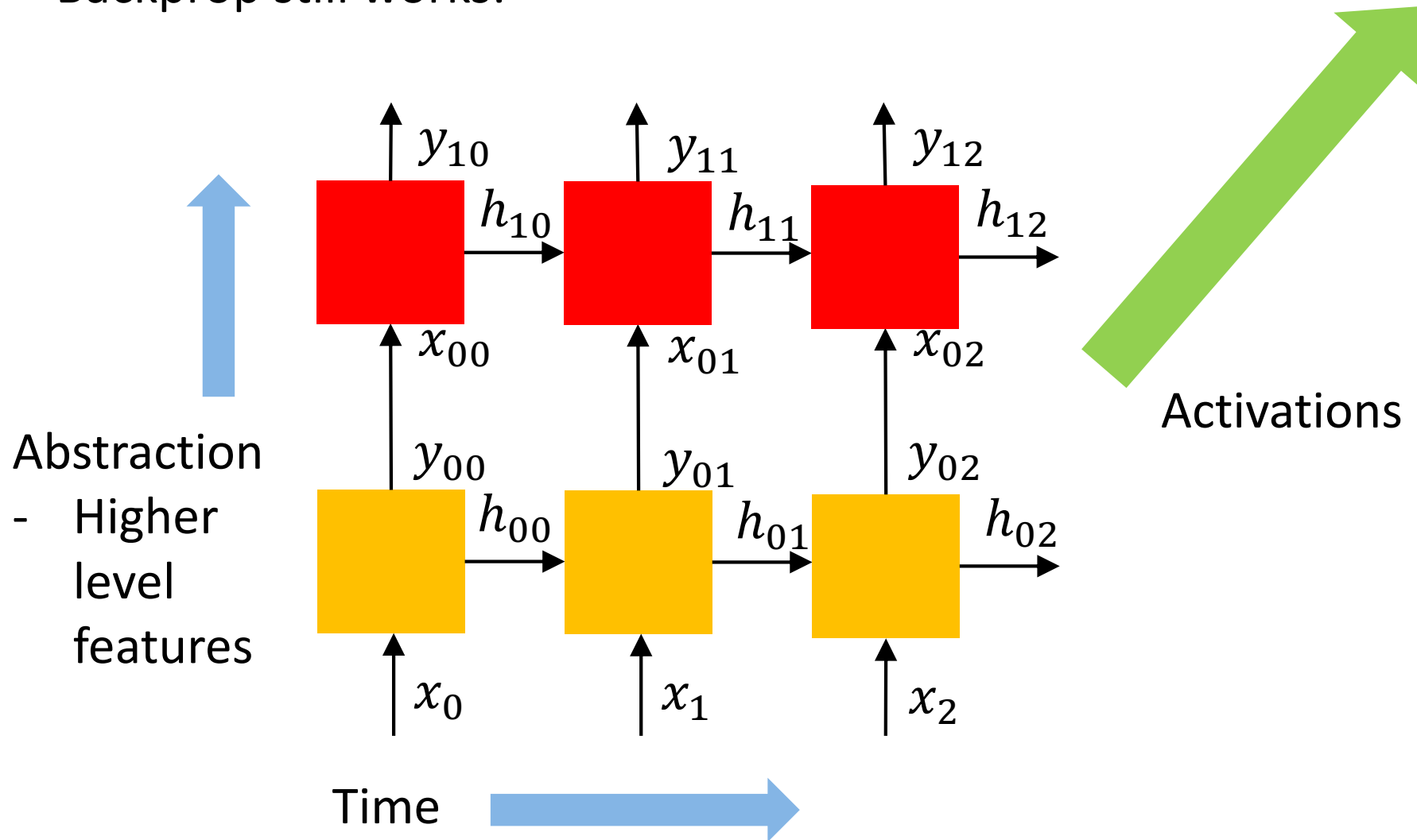
# RNN structure

Backprop still works:



# RNN structure

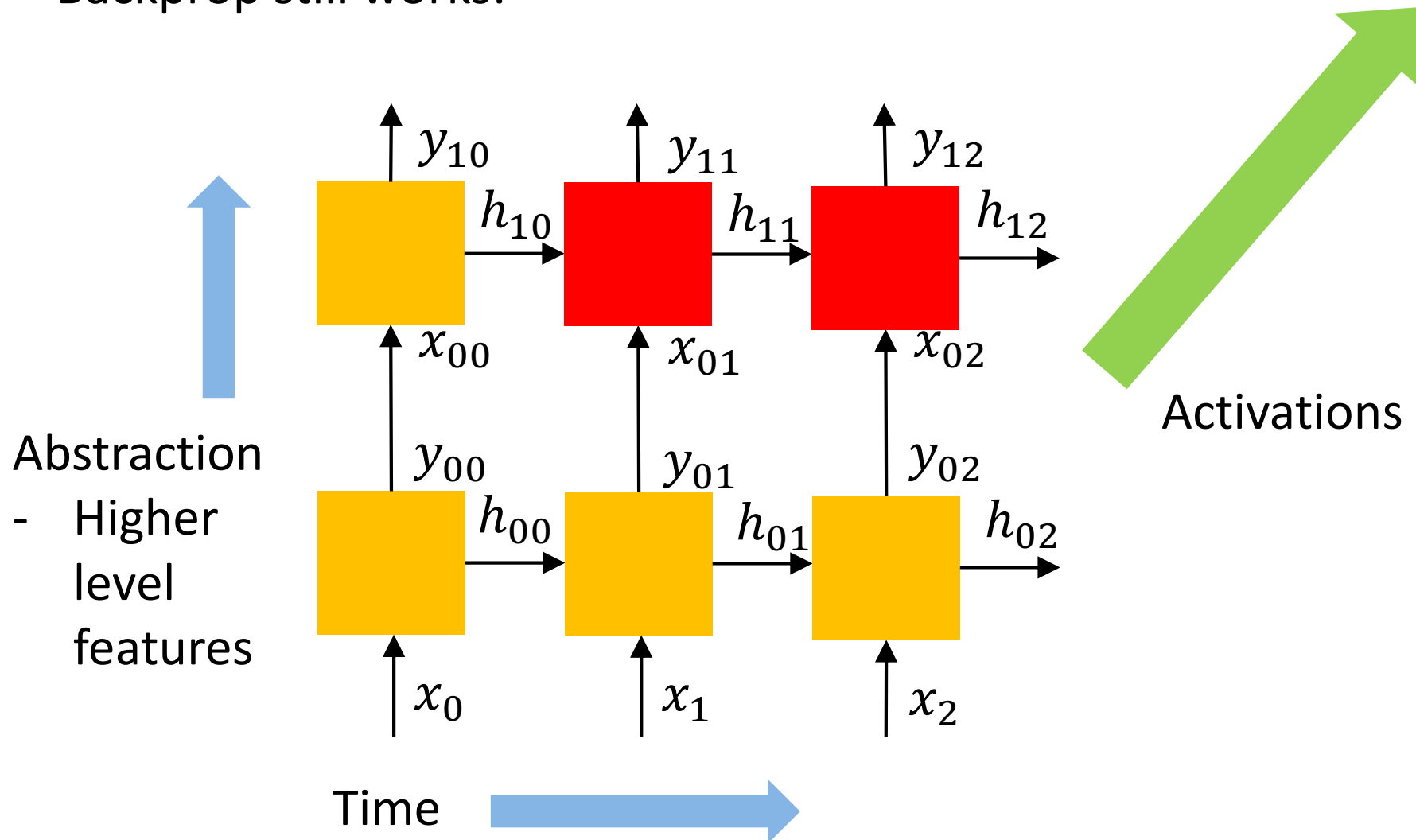
Backprop still works:





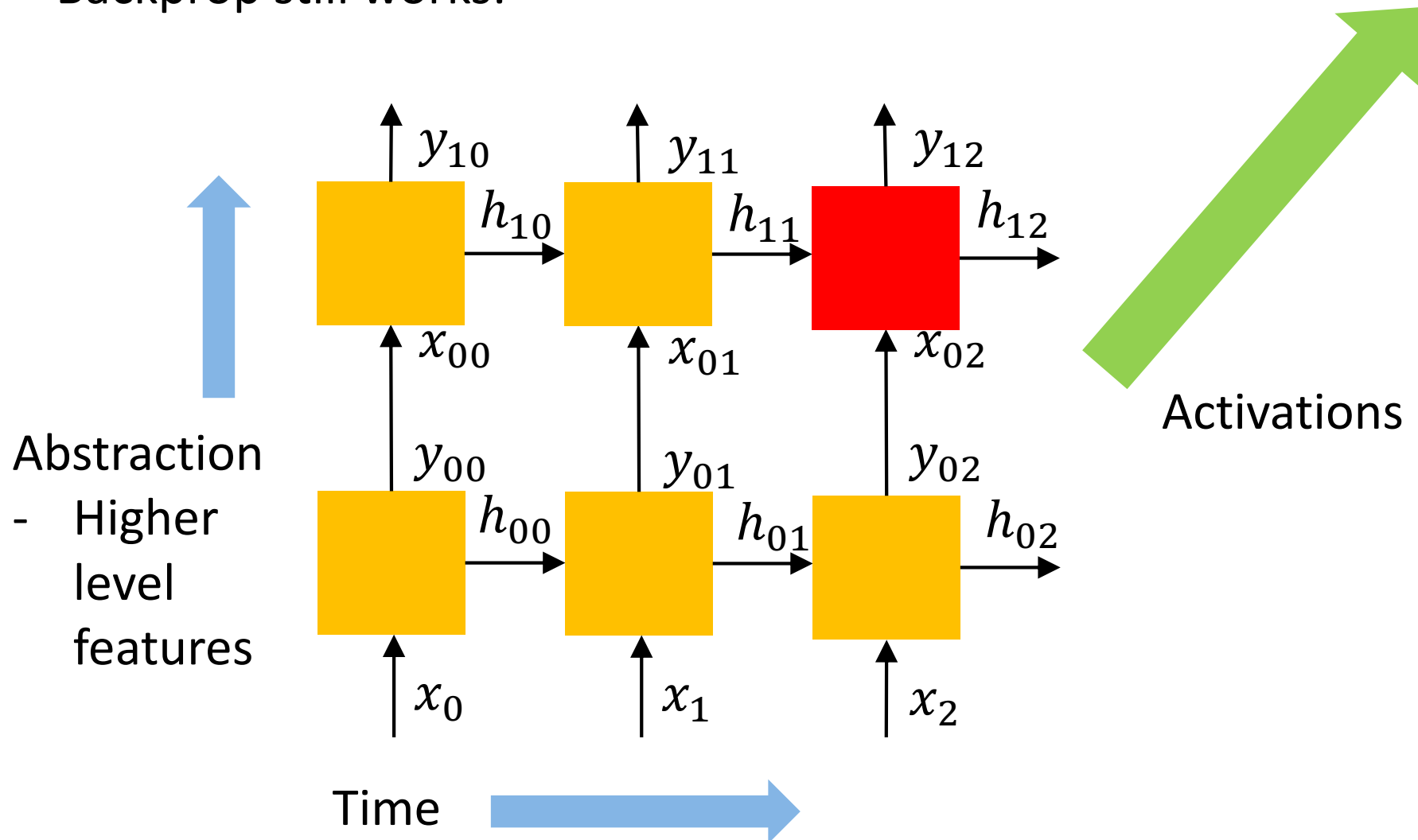
# RNN structure

Backprop still works:



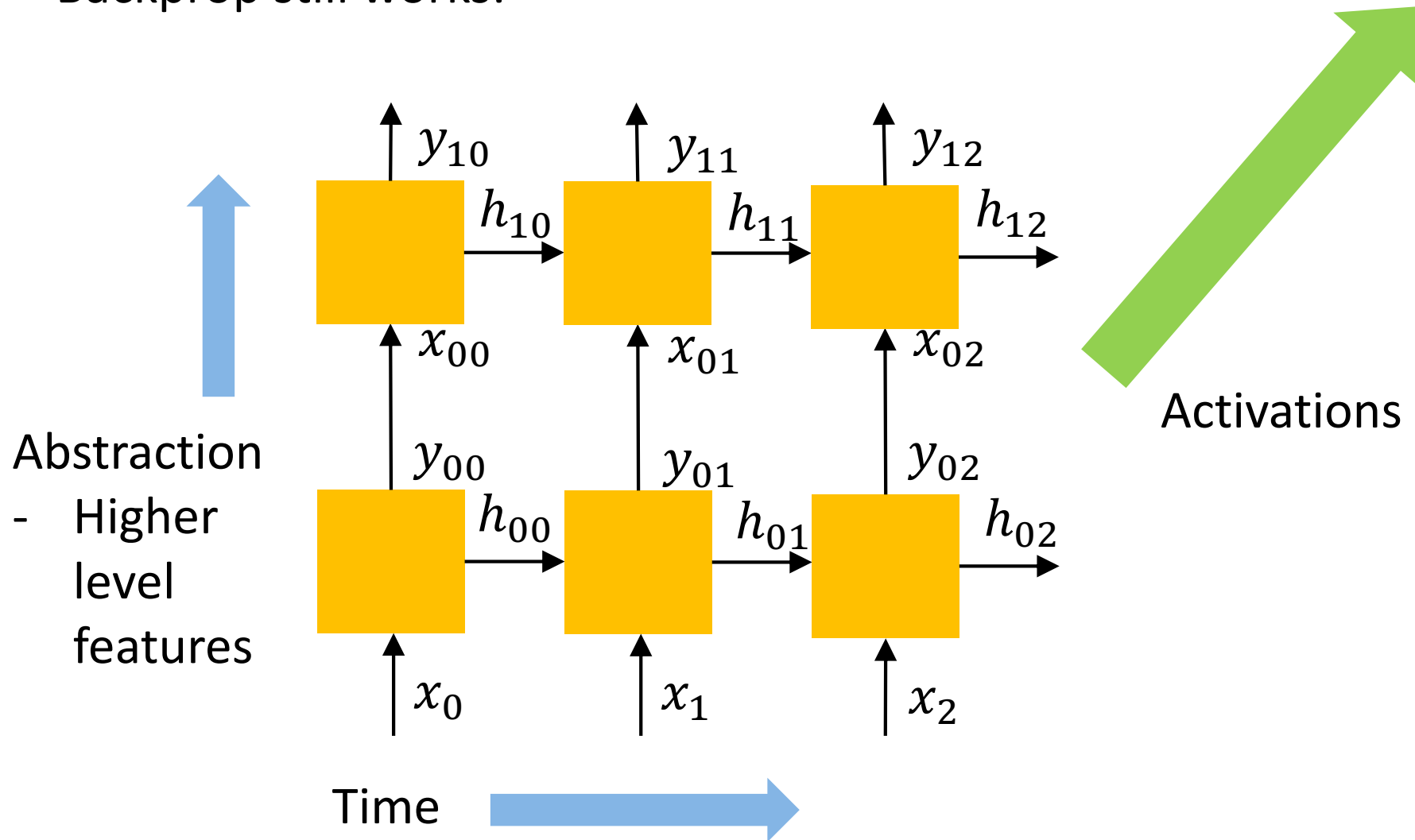
# RNN structure

Backprop still works:



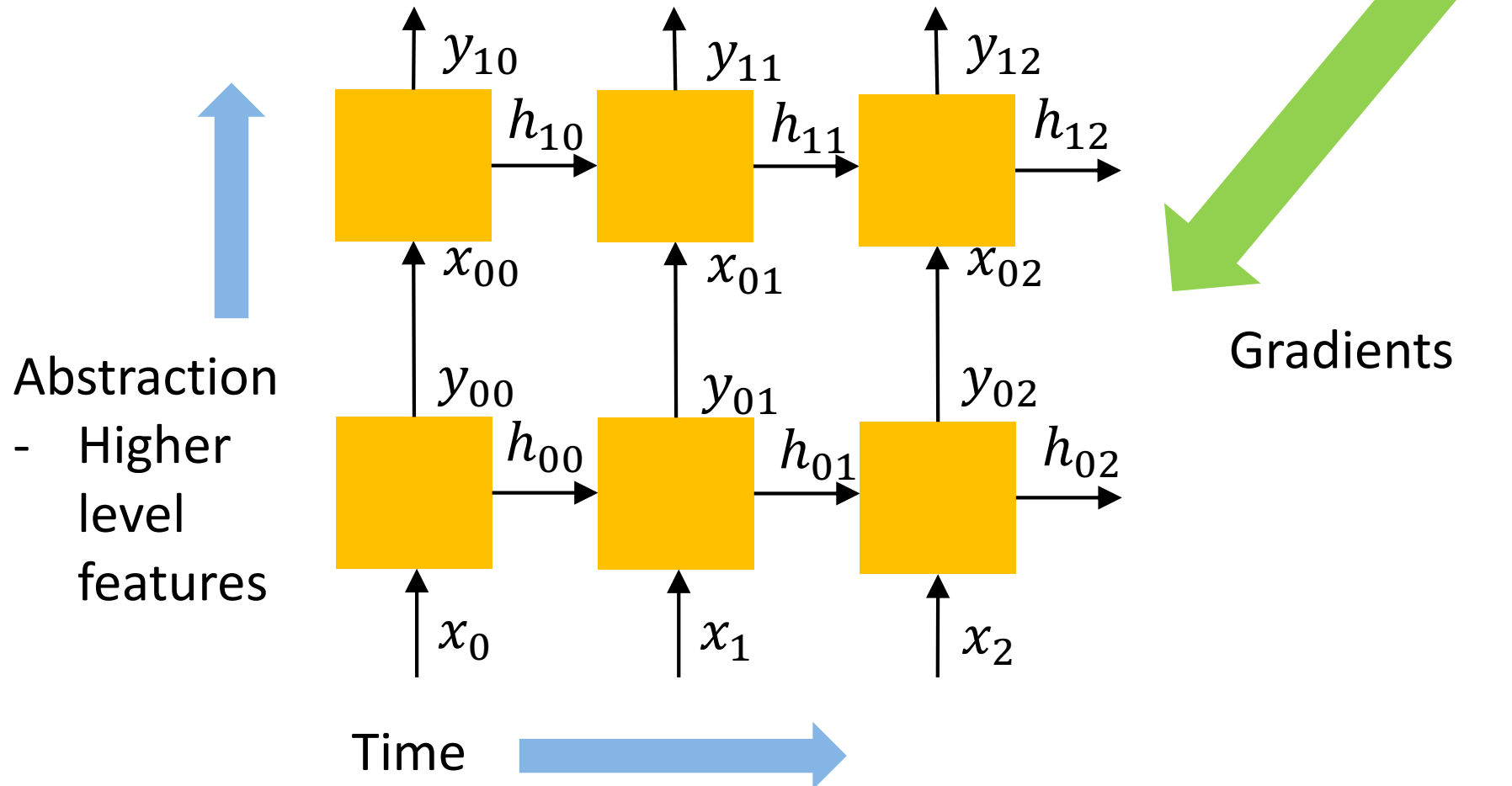
# RNN structure

Backprop still works:



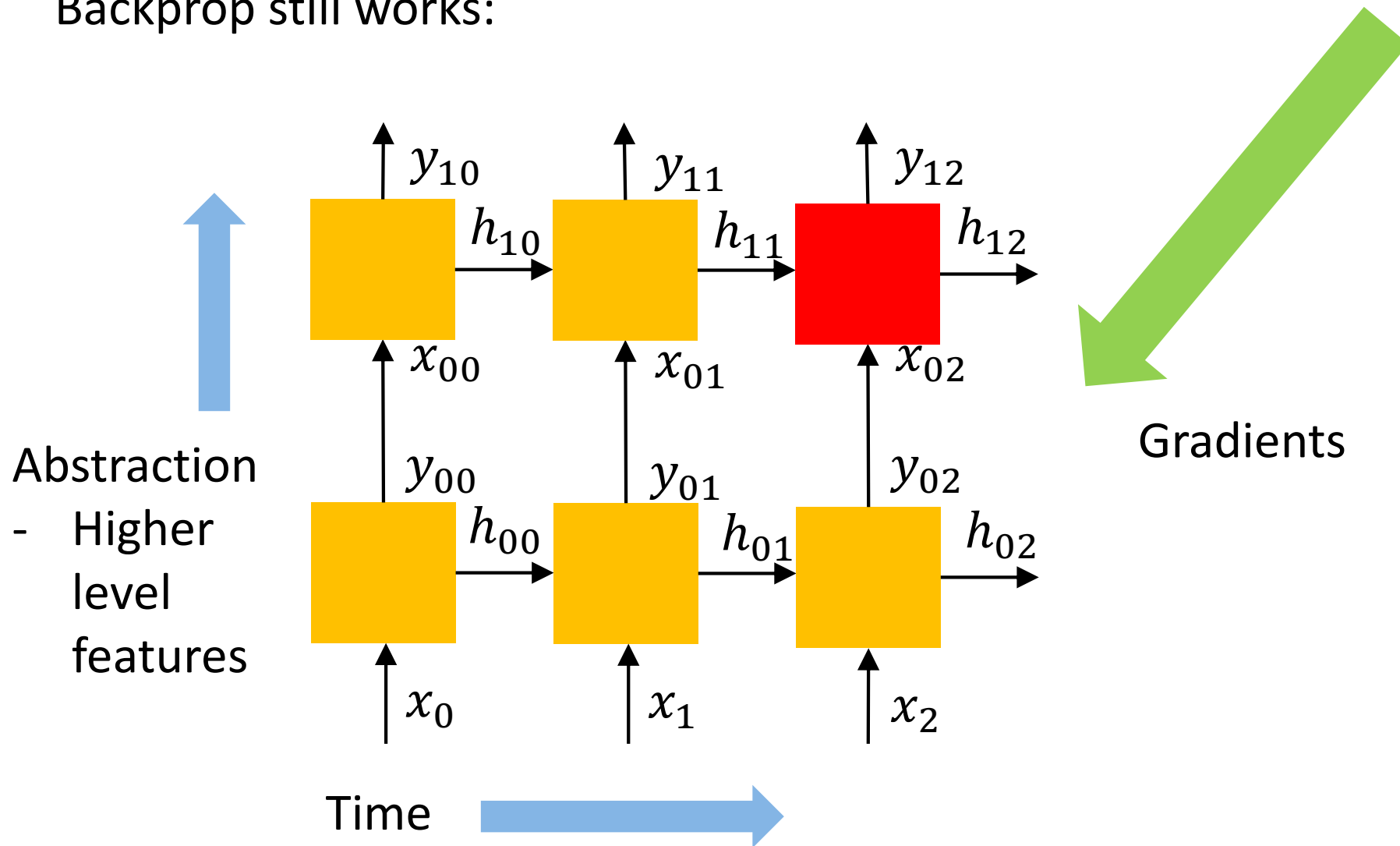
# RNN structure

Backprop still works:



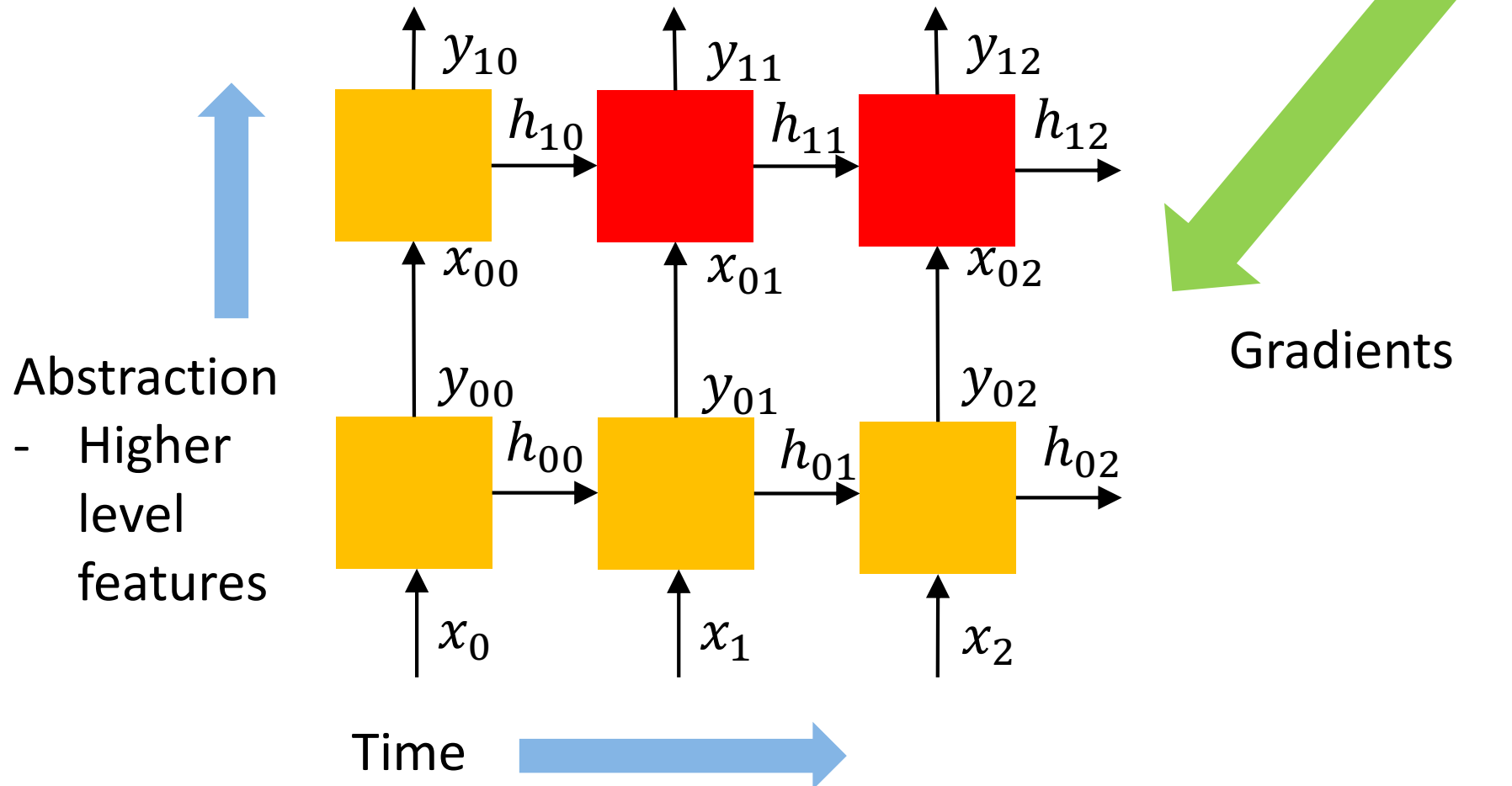
# RNN structure

Backprop still works:



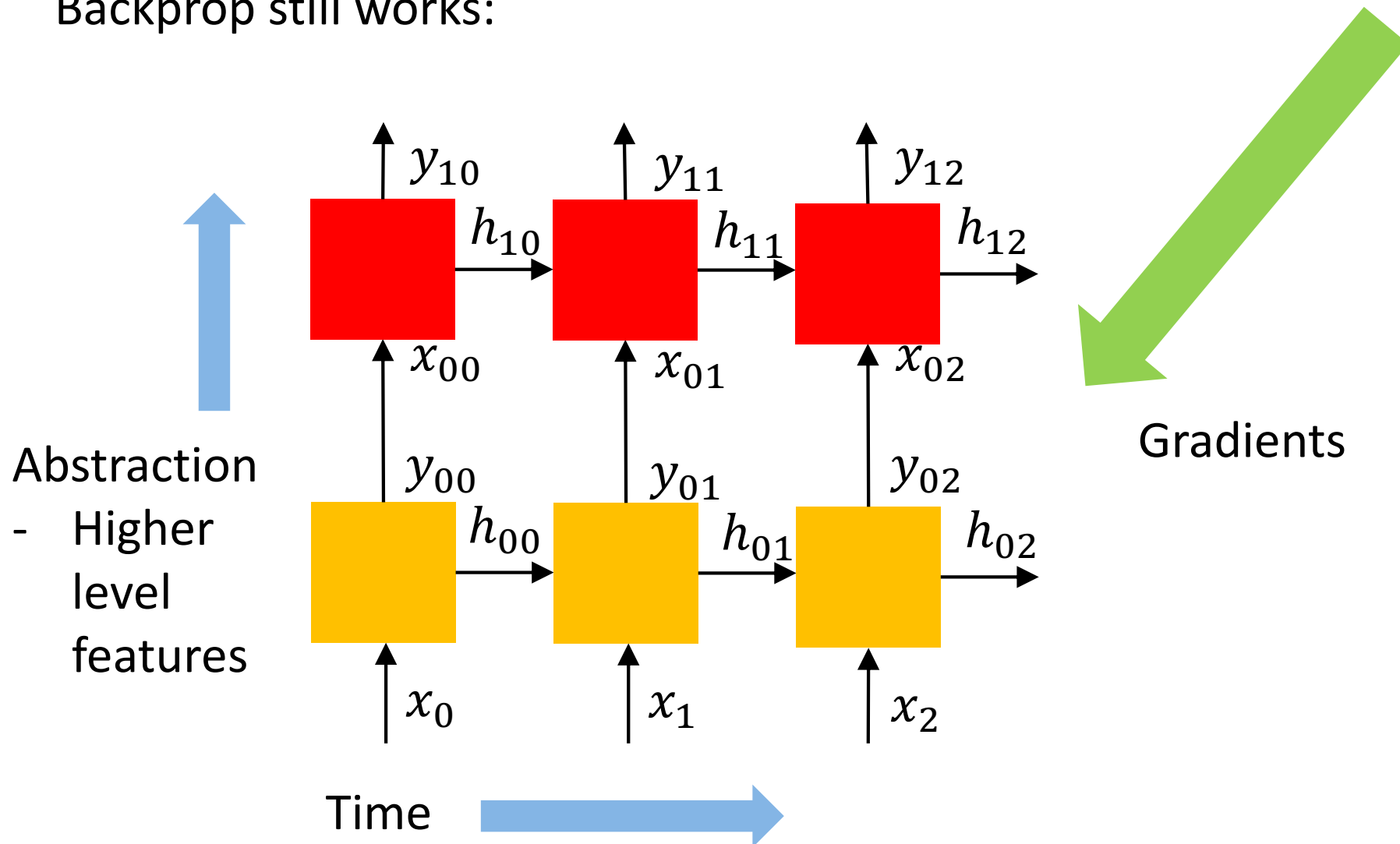
# RNN structure

Backprop still works:



# RNN structure

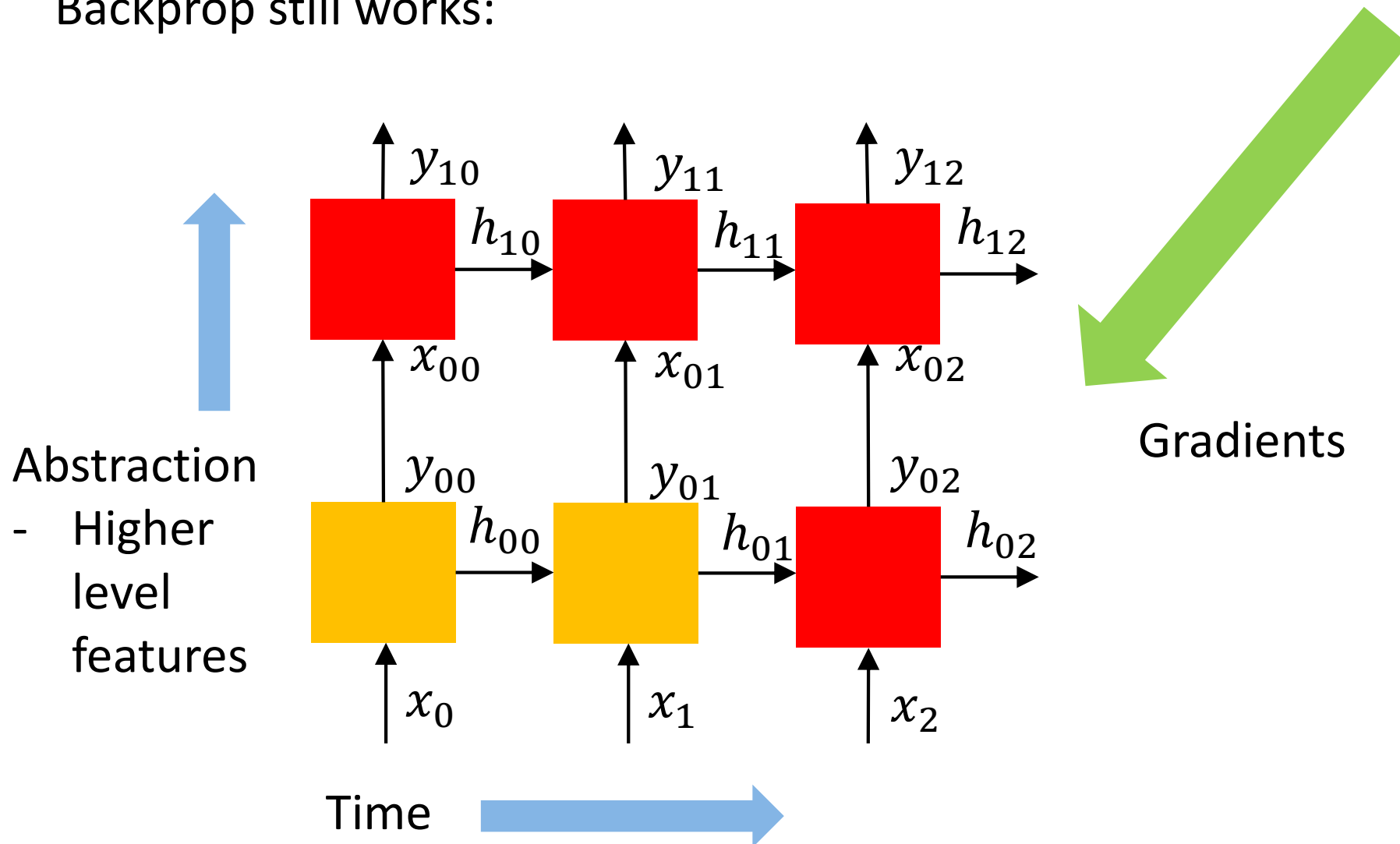
Backprop still works:





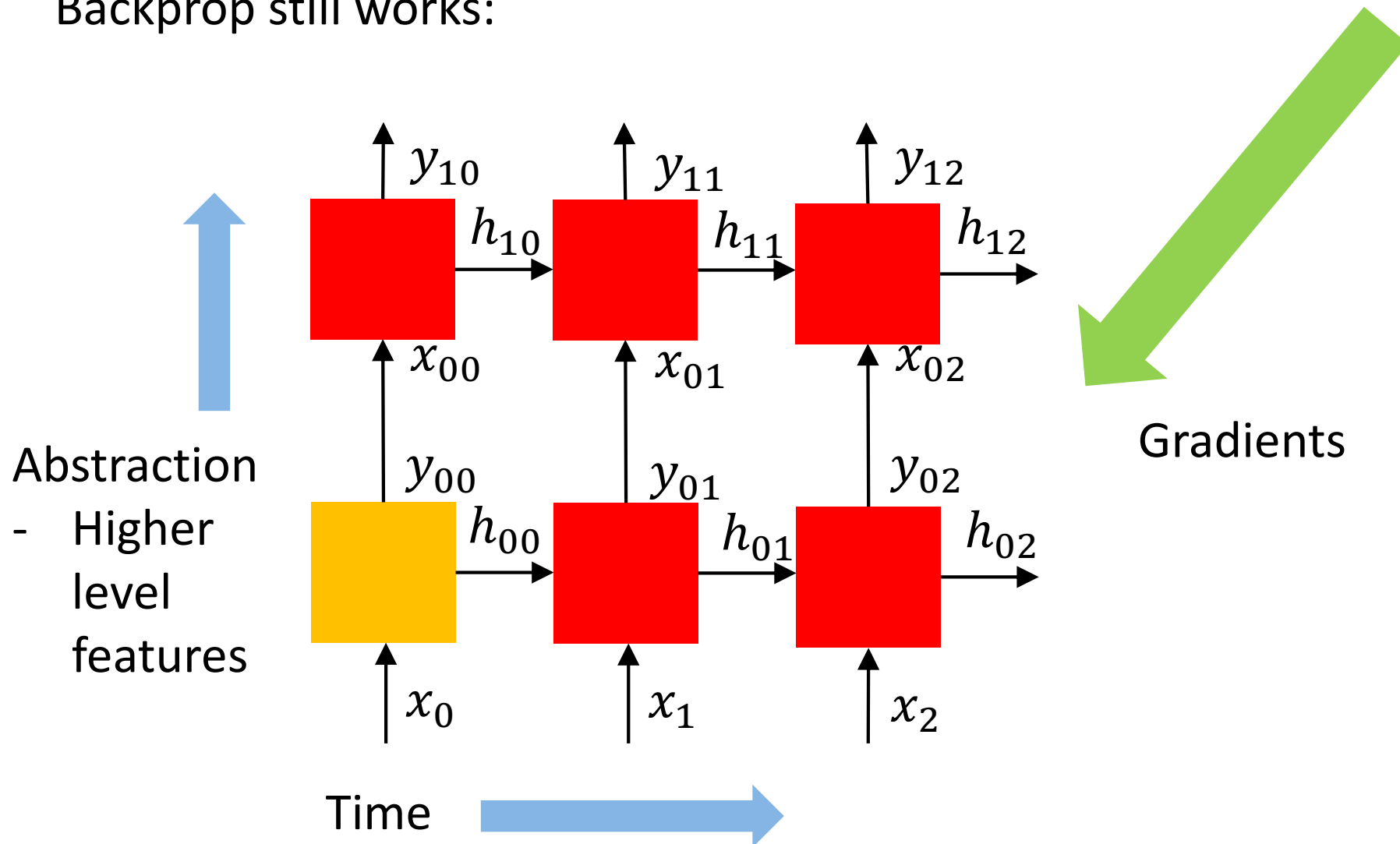
# RNN structure

Backprop still works:



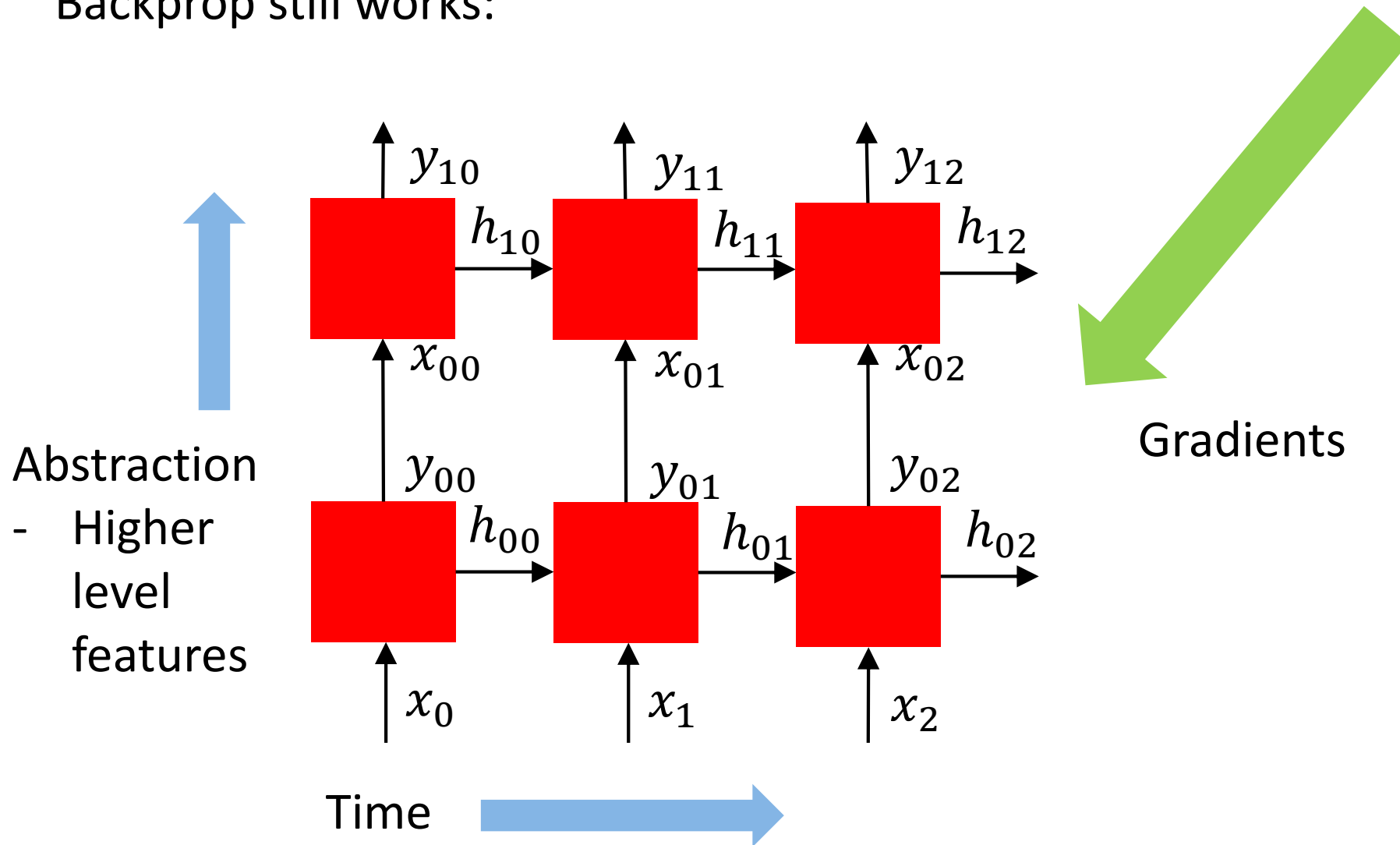
# RNN structure

Backprop still works:

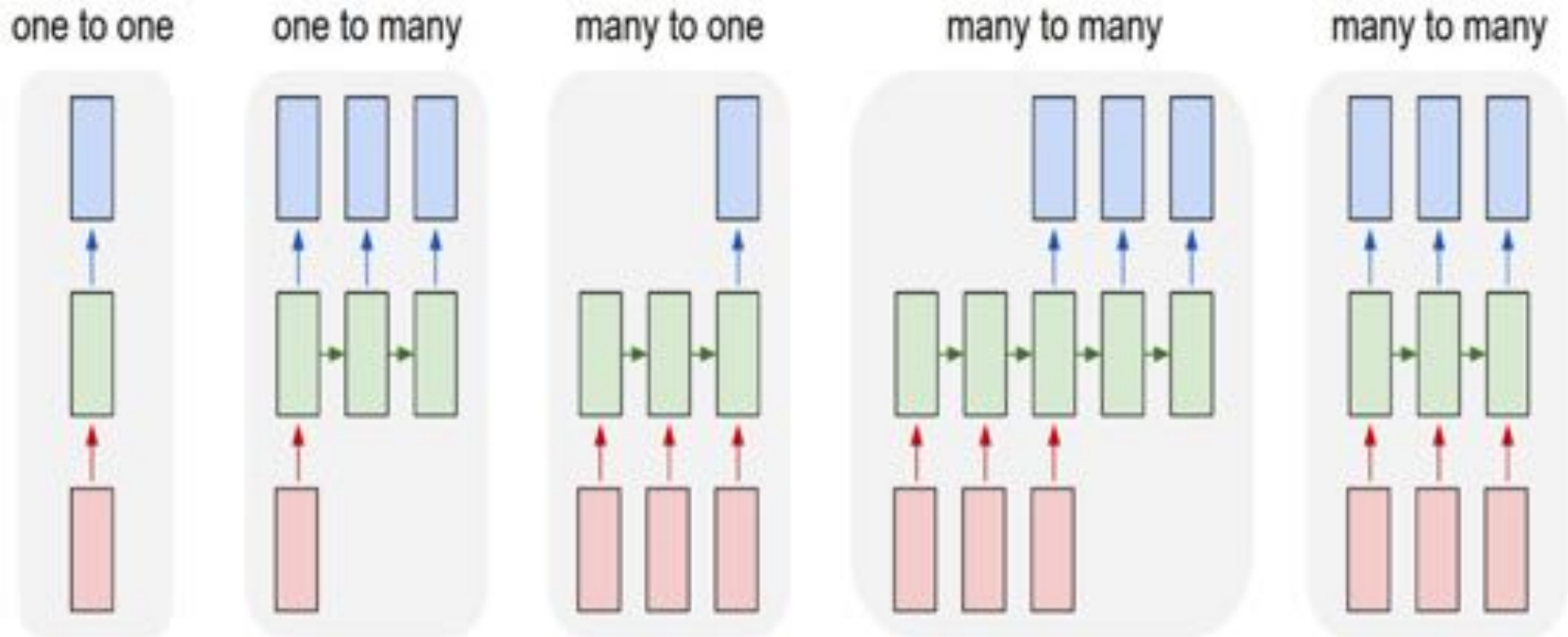


# RNN structure

Backprop still works:



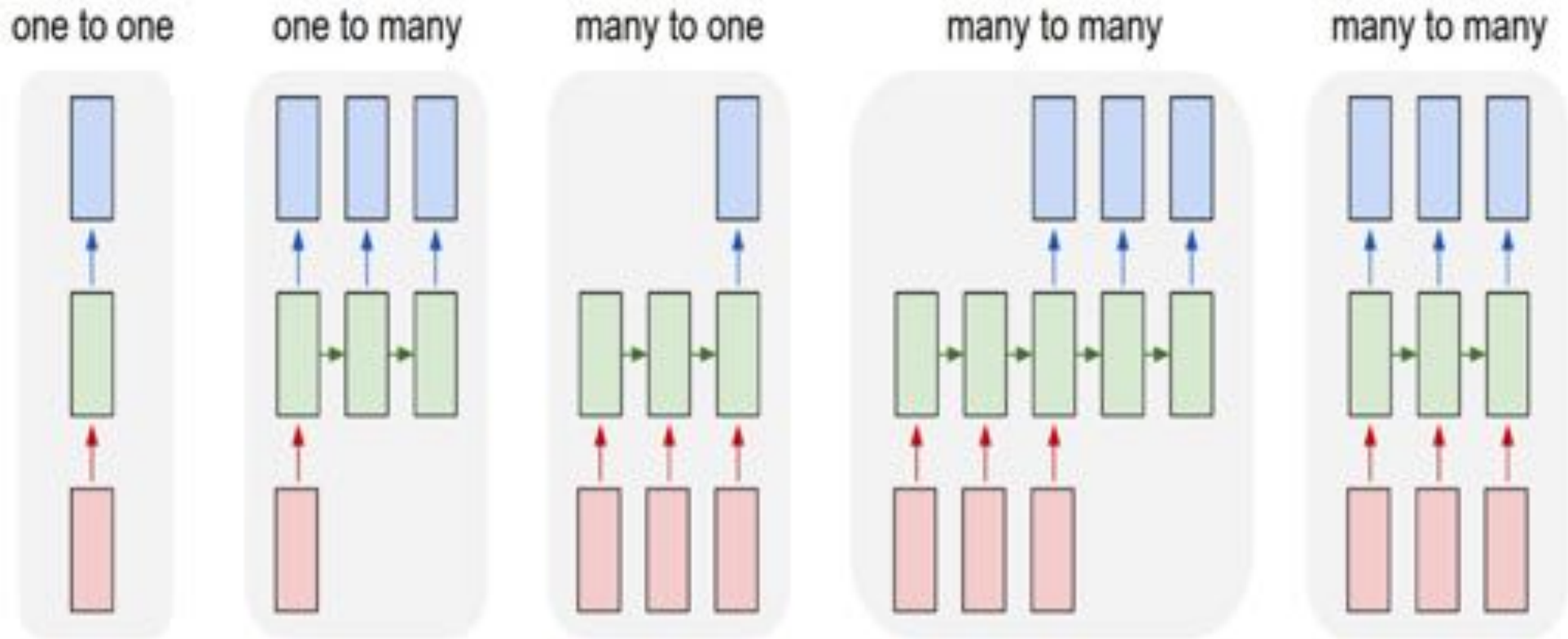
# Recurrent Networks offer a lot of flexibility:



↖ **Vanilla Neural Networks**



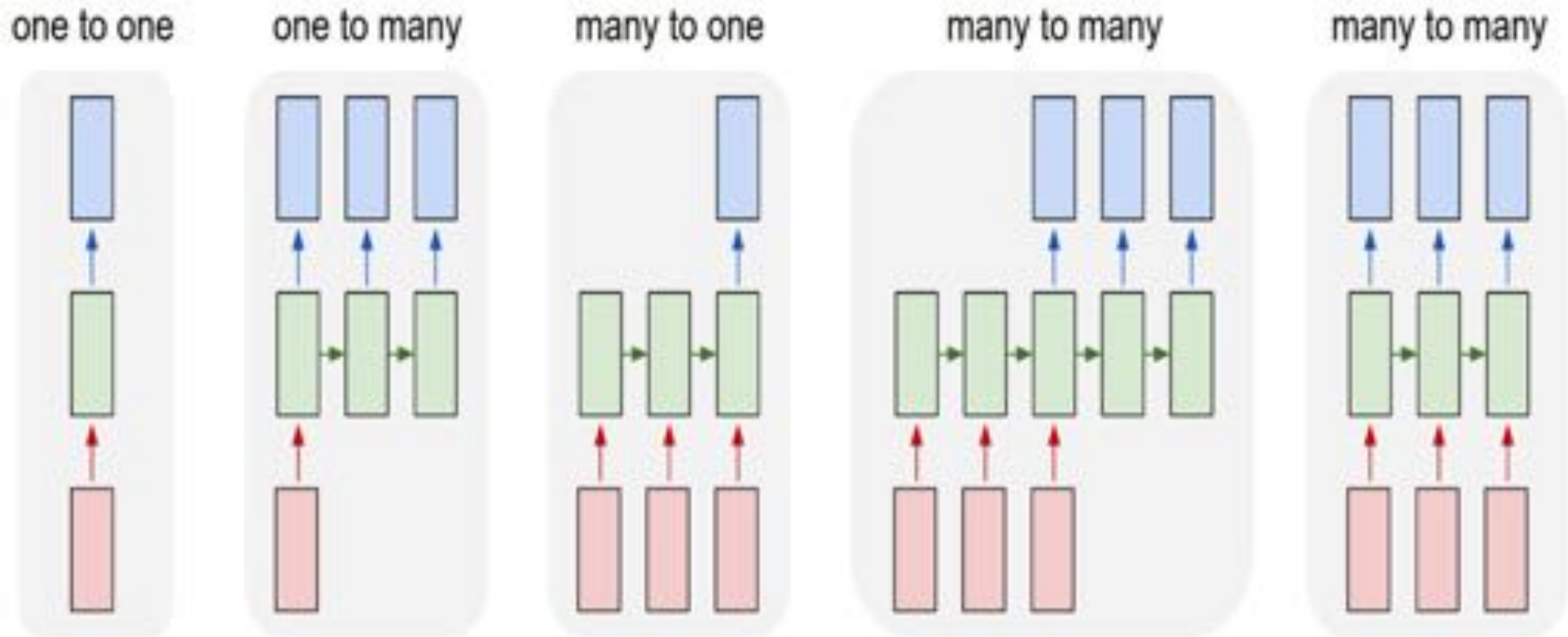
# Recurrent Networks offer a lot of flexibility:



↖ e.g. **Image Captioning**  
image → sequence of words



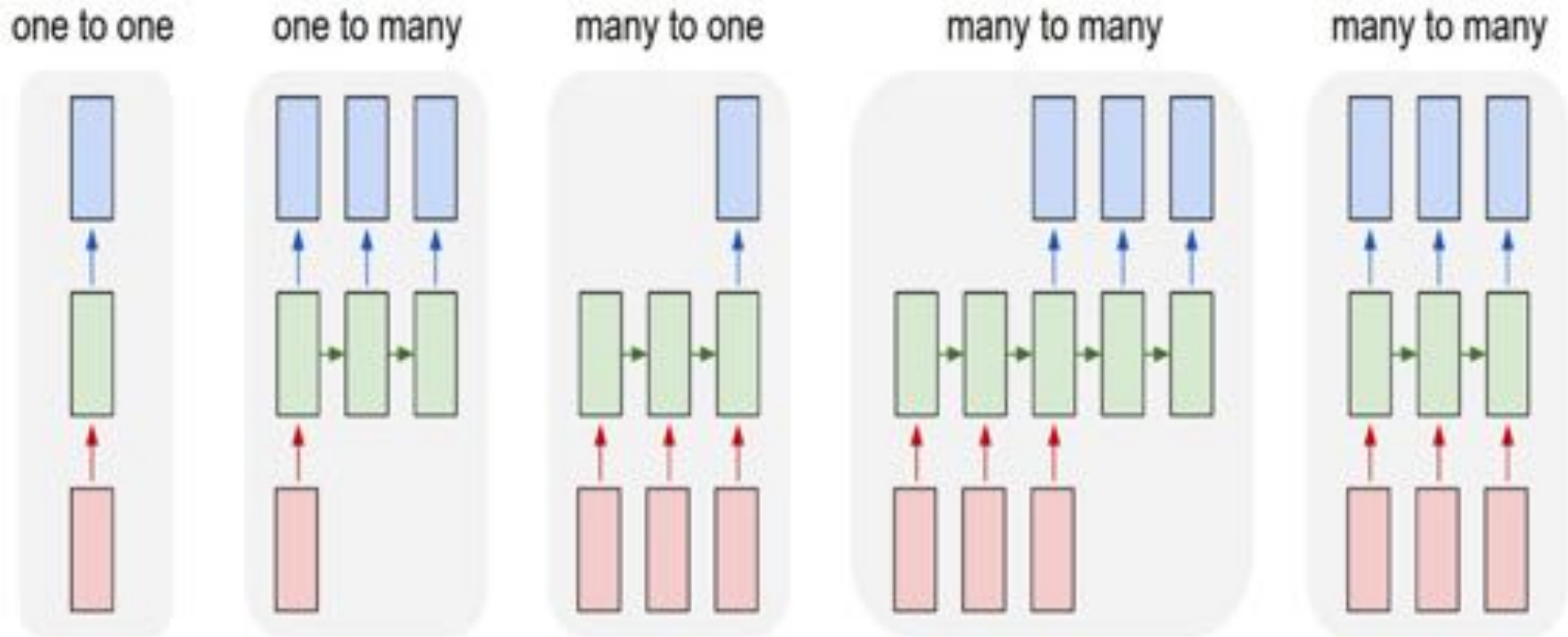
# Recurrent Networks offer a lot of flexibility:



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



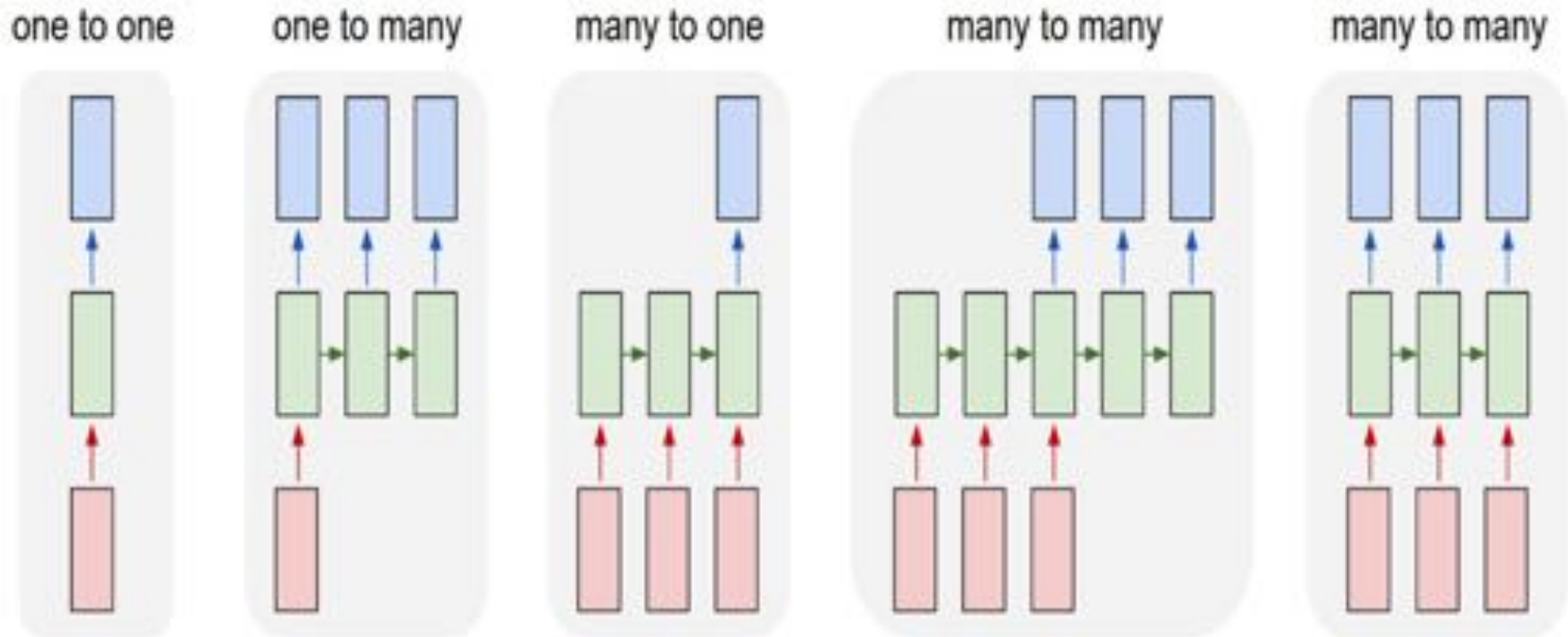
# Recurrent Networks offer a lot of flexibility:



↖ e.g. **Machine Translation**  
seq of words → seq of words



# Recurrent Networks offer a lot of flexibility:

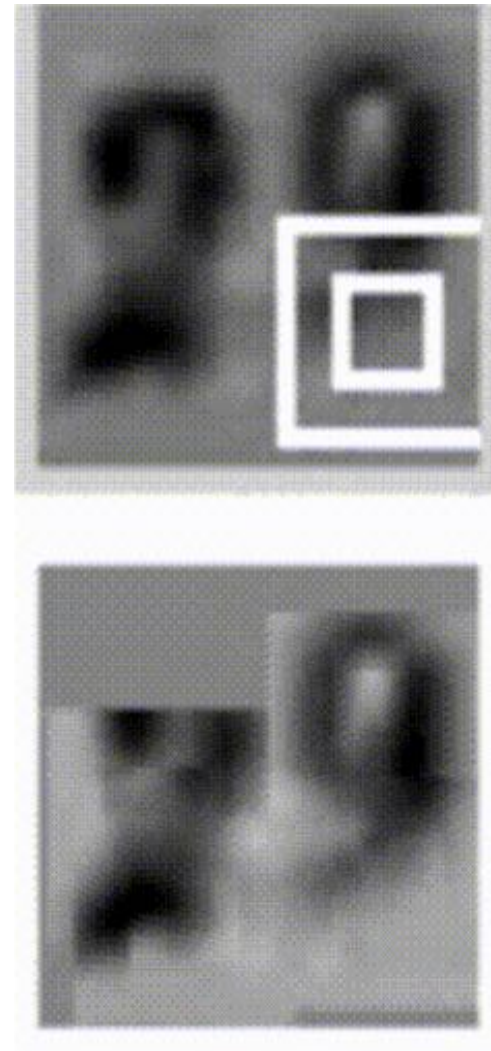


e.g. Video classification on frame level





# Sequential Processing of fixed inputs

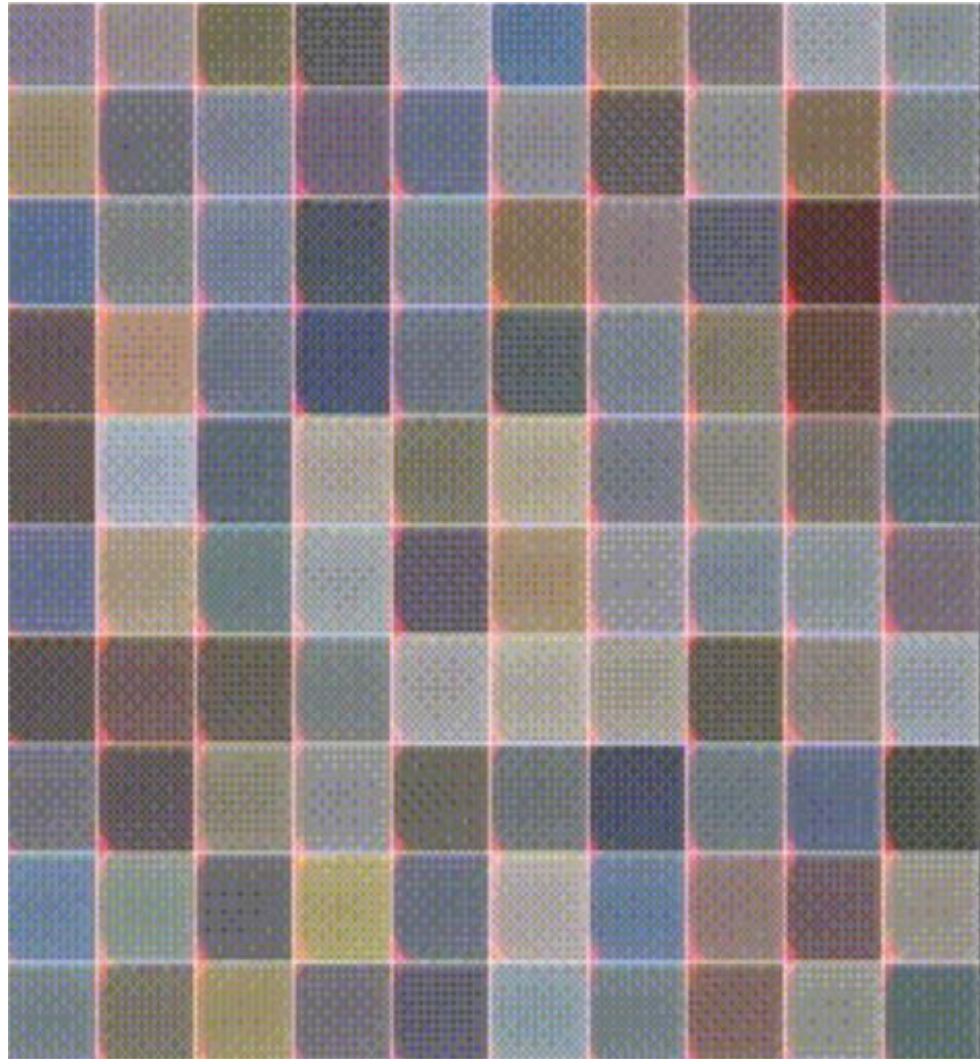


Multiple Object Recognition with  
Visual Attention, Ba et al.

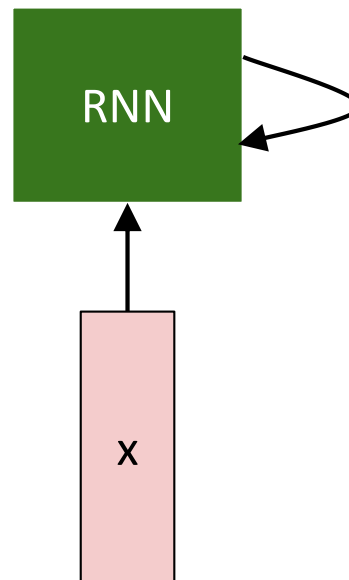


# Sequential Processing of fixed inputs

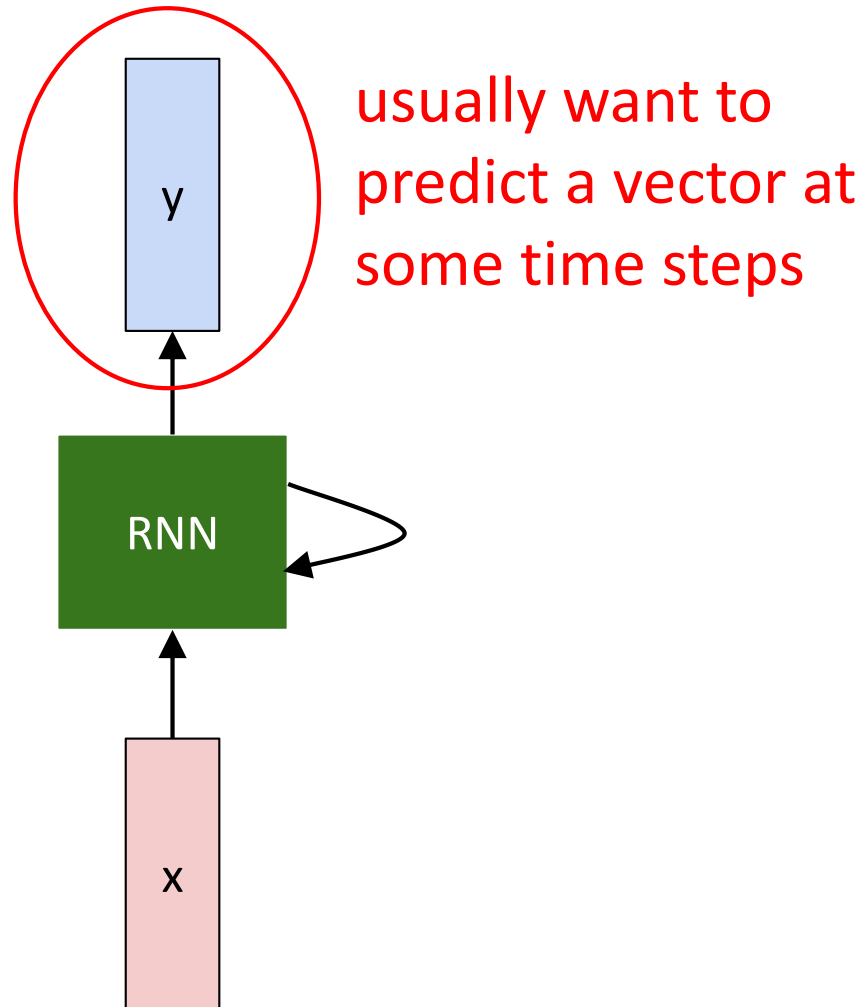
DRAW: A Recurrent  
Neural Network For  
Image Generation, Gregor  
et al.



# Recurrent Neural Network



# Recurrent Neural Network



# Recurrent Neural Network

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

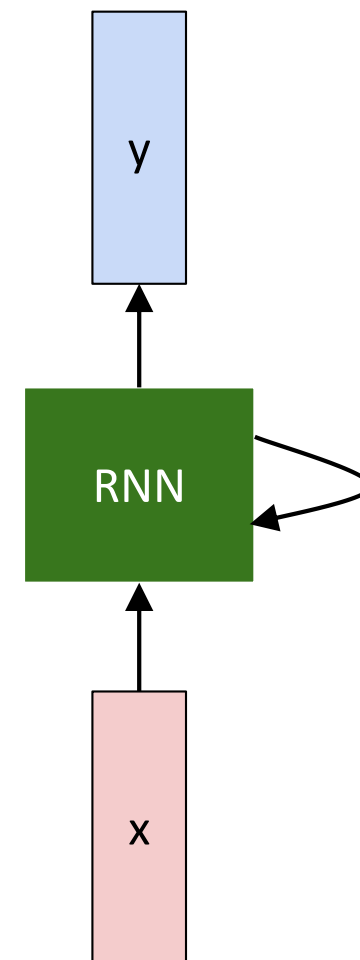
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters  $W$

old state

input vector at some time step

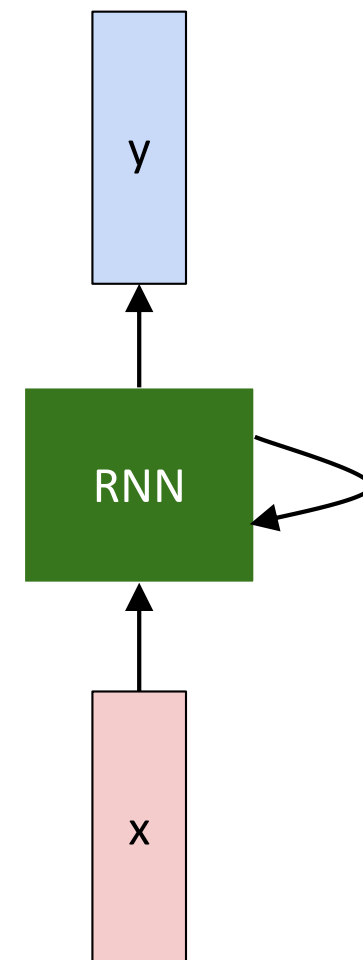


# Recurrent Neural Network

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

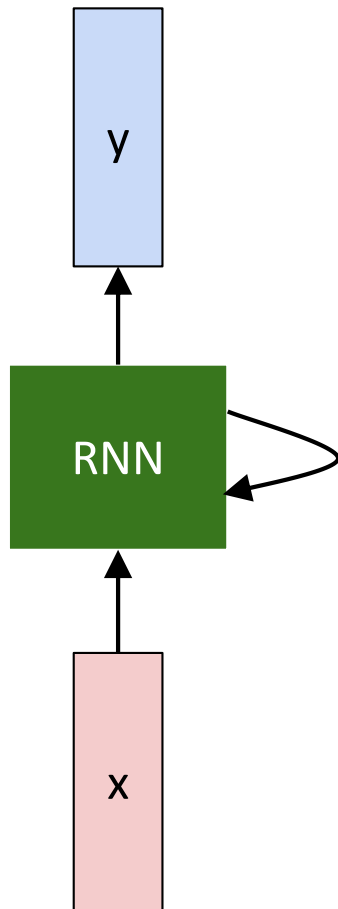
$$h_t = f_W(h_{t-1}, x_t)$$

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

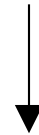


# (Vanilla) Recurrent Neural Network

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



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



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

$$y_t = W_{hy}h_t$$



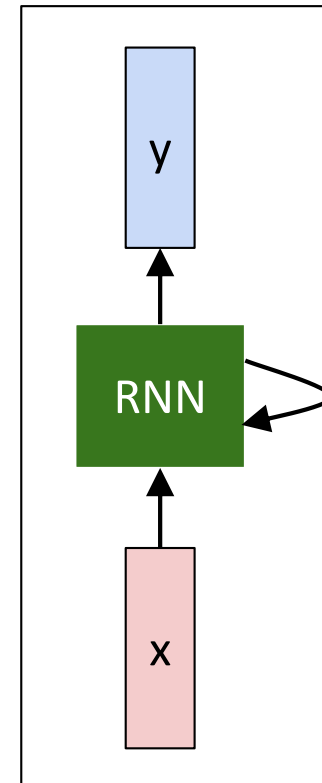
# Character-level language model example

Vocabulary:

[h,e,l,o]

Example training  
sequence:

**“hello”**





# Character-level language model example

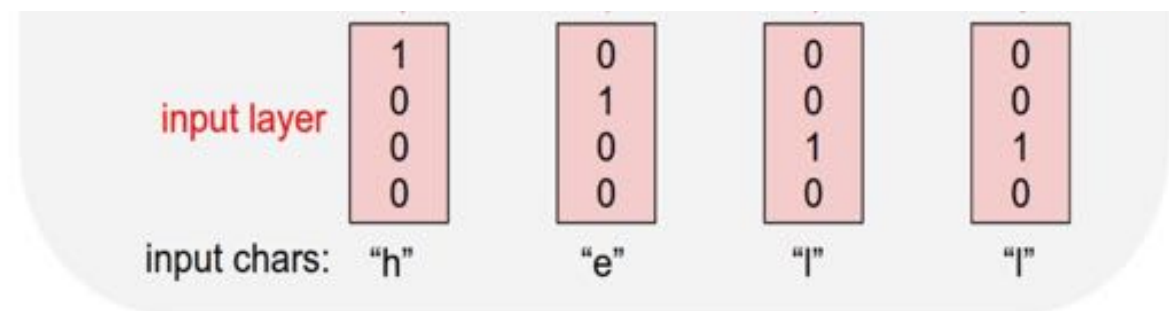
Vocabulary:

[h,e,l,o]

Example training

sequence:

**“hello”**



# Character-level language model example

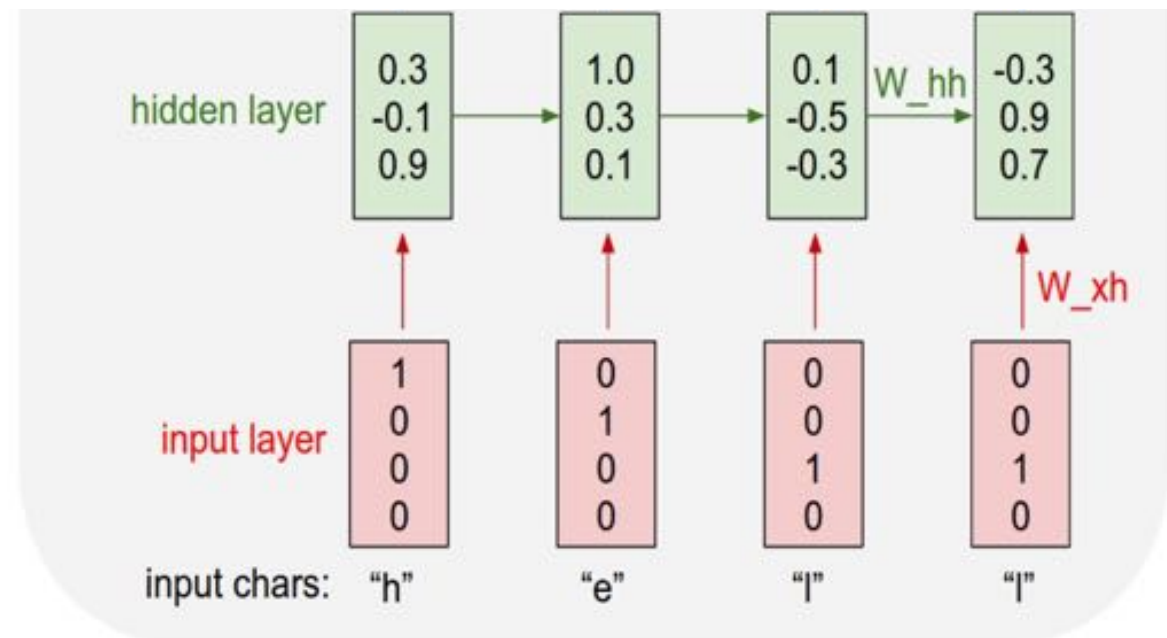
Vocabulary:

[h,e,l,o]

Example training  
sequence:

**“hello”**

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



# Character-level language model example

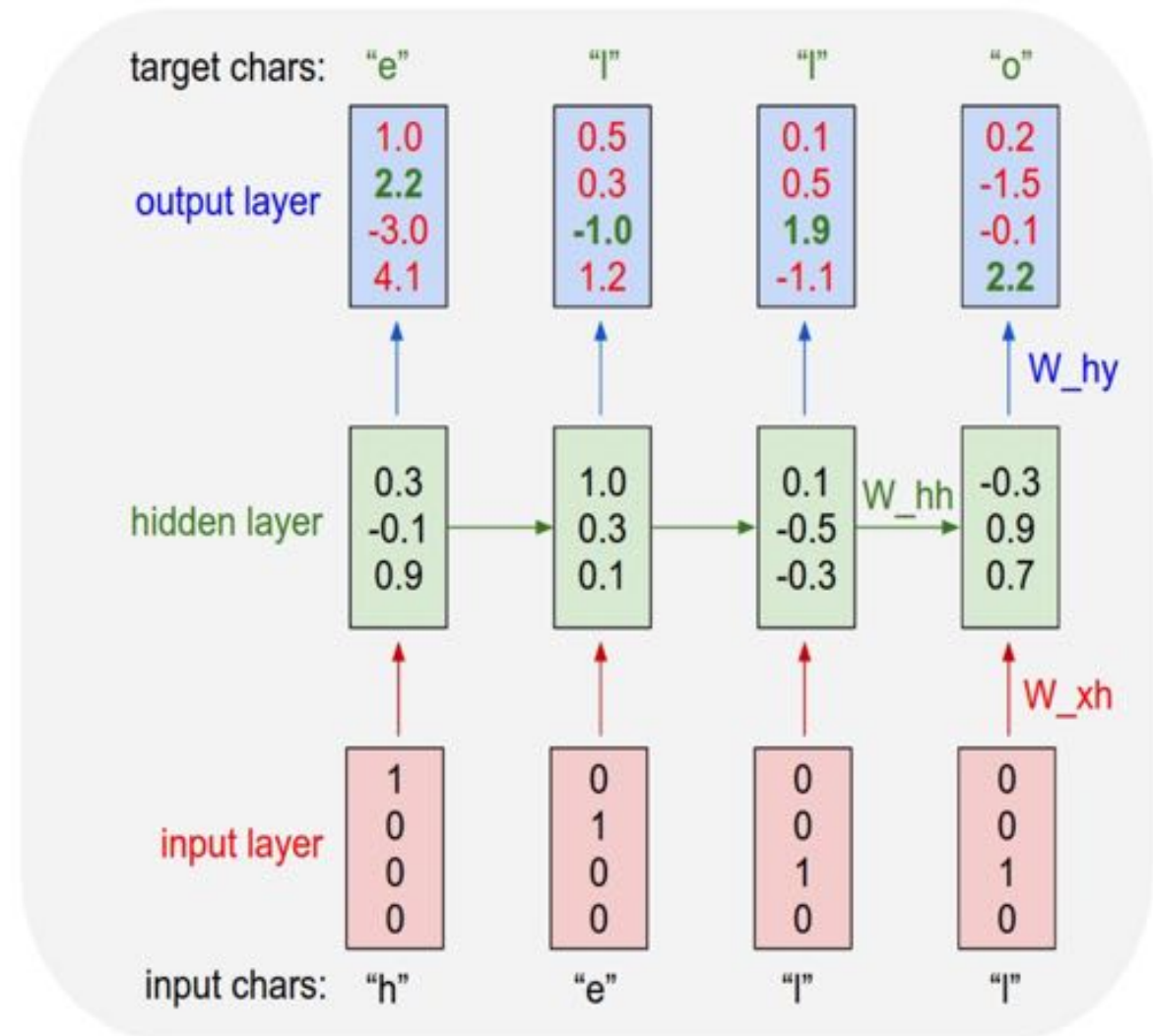
Vocabulary:

[h,e,l,o]

Example training

sequence:

**“hello”**



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

```
1  """
2  Minimal character-level vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD license
4  """
5  import numpy as np
6
7  # data I/O
8  data = open('input.txt', 'r').read() # should be some 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 wnh = np.random.randn(hidden_size, vocab_size)*0.02 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.02 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.02 # 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 Numpy array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xh, hh, yh, gh = [], [], [], []
34     hx[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xt = np.zeros((vocab_size, 1)) # encode as 1-of-K representation
39         xt[t][inputs[t]] = 1
40         ht = np.tanh(np.dot(wnh, xt) + np.dot(whh, hx[t-1]) + bh) # hidden state
41         yt = np.dot(why, ht) + by # unnormalized log probabilities for next char
42         pt = np.exp(yt) / np.sum(np.exp(yt)) # probabilities for next char
43         loss += -np.log(pt)[targets[t], 0] # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backwards
45     dxh, dhh, dhy = np.zeros_like(wnh), np.zeros_like(whh), np.zeros_like(why)
46     dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47     dhnext = np.zeros_like(hx[-1])
48     for t in reversed(xrange(len(inputs))):
49         dy = np.copy(pt[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dhy += np.dot(dy, ht[t].T)
52         dbh += dy
53         dh = np.dot(why.T, dy) + dhnext # backprop into h
54         ddraw = [(1 - ht[t]**2) * ht[t]] * dh # backprop through tanh nonlinearity
55         dxh += np.dot(ddraw, xt[t].T)
56         dhh += np.dot(ddraw, ht[t-1].T)
57         dhnext = np.dot(whh.T, ddraw)
58     for dparam in [dxh, dhh, dhy, dbh, dby]:
59         np.clip(dparam, -1, 1, out=dparam) # clip to mitigate exploding gradients
60     return loss, dxh, dhh, dhy, dbh, dby, hx[len(inputs)-1]
61
62 def sample(h, seed_ix, n):
63     """
64     sample a sequence of integers from the model
65     h is memory state, seed_ix is seed letter for first time step
66     """
67     x = np.zeros((vocab_size, 1))
68     x[seed_ix] = 1
69     ixes = []
70     for t in xrange(n):
71         h = np.tanh(np.dot(wnh, x) + np.dot(whh, h) + bh)
72         y = np.dot(why, h) + by
73         p = np.exp(y) / np.sum(np.exp(y))
74         ix = np.random.choice(range(vocab_size), p=p.ravel())
75         x = np.zeros((vocab_size, 1))
76         x[ix] = 1
77         ixes.append(ix)
78     return ixes
79
80 n, p = 0, 0
81 mnh, mhh, mhy = np.zeros_like(wnh), np.zeros_like(whh), np.zeros_like(why)
82 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
83 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
84 while True:
85     # prepare inputs (we're sweeping from left to right in steps seq_length long)
86     if p+seq_length >= len(data) or n == 0:
87         hprev = np.zeros((hidden_size, 1)) # reset RNN memory
88         p = 0 # go from start of data
89     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
90     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
91
92     # sample from the model now and then
93     if n % 100 == 0:
94         sample_ix = sample(hprev, inputs[0], 200)
95         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
96         print "----\n %s \n----" % (txt, )
97
98     # forward seq. length characters through the net and fetch gradient
99     loss, dxh, dhh, dhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
100     smooth_loss = smooth_loss * 0.999 + loss * 0.001
101     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
102
103     # perform parameter update with Adagrad
104     for param, dparam, mem in zip([wnh, whh, why, bh, by],
105                                   [dxh, dhh, dhy, dbh, dby],
106                                   [mnh, mhh, mhy, mbh, mby]):
107         mem += dparam * dparam
108         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
109
110     p += seq_length # move data pointer
111     n += 1 # iteration counter
```

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



# Data I/O



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```

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_idx, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_idx, 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 # Model parameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 20 # number of steps in time to unroll the net for
18 learning_rate = 1e-3
19
20 # model parameters
21 wnh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to output
24 w0 = 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 numpy array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state """
32
33     xs, hs, ys, ps = [], [], [], []
34     h = hprev
35     loss = 0
36
37     # forward pass
38     for t in xrange(len(inputs)):
39         xi = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
40         xi[xi_to_char[inputs[t]]] = 1
41         h_t1 = np.tanh(np.dot(wnh, xi) + np.dot(whh, h[-1]) + why) # hidden state
42         yt = h_t1 * why # h_t1 by a normalized log probabilities for next chars
43         p_t = exp(yt) / np.sum(exp(yt)) # probabilities for next chars
44         loss += -log(p_t[targets[t]]) # log-likelihood (cross-entropy loss)
45         # backward pass, compute gradients going backwards
46         dwh, dwn, dby = np.zeros_like(wnh), np.zeros_like(whh), np.zeros_like(why)
47         dh_t = np.zeros_like(h_t1)
48         # for t reversed over range(len(inputs)):
49             dy = np.copy(p_t[1])
50             dy[targets[t]] -= 1 # backwards pass
51             dwt = np.dot(dy, h_t1.T)
52             dw = np.dot(w_t1.T, dwt) # dwnet = backward pass h
53             dwn = dwt - dw
54             dwn = np.dot(dwn, w_t1.T)
55             dwt = np.dot(dwn, h[-1].T)
56             dwn = np.dot(dwn, T, dwt)
57             for oparam in [dwh, dwn, dby, dh_t, dy]:
58                 clip(operam, -4, 4, outparam) # clip to mitigate exploding gradients
59             return loss, dwh, dwn, dby, dh_t, dy, h[ix_to_char[inputs[t]]]
60
61 def sample(x, seed_ix, k):
62     """
63     sample a sequence of integers from the model
64     x is memory state, used as seed letter for first time step
65
66     n = np.zeros((vocab_size, 1))
67     x[seed_ix] = 1
68     for i in xrange(k):
69         xi = np.tanh(np.dot(wnh, x) + np.dot(whh, h) + why)
70         p = exp(xi) / np.sum(exp(xi))
71         ix = np.random.choice('range(vocab_size, p=p))
72         x[ix] = 1
73     return append(x)
74
75 # n, h, p, h, p, h
76 mnh, mwn, mby = np.zeros_like(wnh), np.zeros_like(whh), np.zeros_like(why)
77 mnh, mwn, mby = np.zeros_like(wnh), np.zeros_like(whh) # memory variables for sampled
78 unroll_loss = -log(p[targets[seq_length-1]]) # loss at iteration 0
79 while True:
80     # generate outputs (we're sampling from left to right in steps (seq_length, seq))
81     if p[seq_length] == (data_idx) or n == 0:
82         p[seq_length] = sample(data_idx, 0) # reset the memory
83         n = 0 # go from start of data
84     inputs = [char_to_ix[ch] for ch in data[seq_length:seq_length+1]]
85     targets = [char_to_ix[ch] for ch in data[seq_length+1:seq_length+2]]
86
87     # sample from the model now and then
88     if n % 100 == 0:
89         sample_ix = sample(vocab_size, unroll_loss, 200)
90         txt = ""
91         for ix in sample_ix:
92             txt += "%s" % ix_to_char[ix]
93         print "txt: '%s'" % txt
94
95     # Forward seq_length characters through the net and fetch gradients
96     loss, dwh, dwn, dby, dh_t, dy, hprev = lossfun(inputs, targets, hprev)
97     unroll_loss = unroll_loss + loss # 0.001 to 0.005
98     if n % 100 == 0: print "seq %d, loss: %f" % (n, unroll_loss) # print progress
99
100 # perform parameter update with Adagrad
101 for param, dparam, n in zip([dwh, dwn, dby, dh_t, dy],
102                             [dwh, mwn, mby, dh_t, dy],
103                             [0, 1, 1, 1, 1]):
104     n += dparam * dparam
105     param = - learning_rate * dparam / np.sqrt(n + 1e-4) # adapted update
106
107 g = seq_length # max data pointer
108 n = 0 # iteration counter

```

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



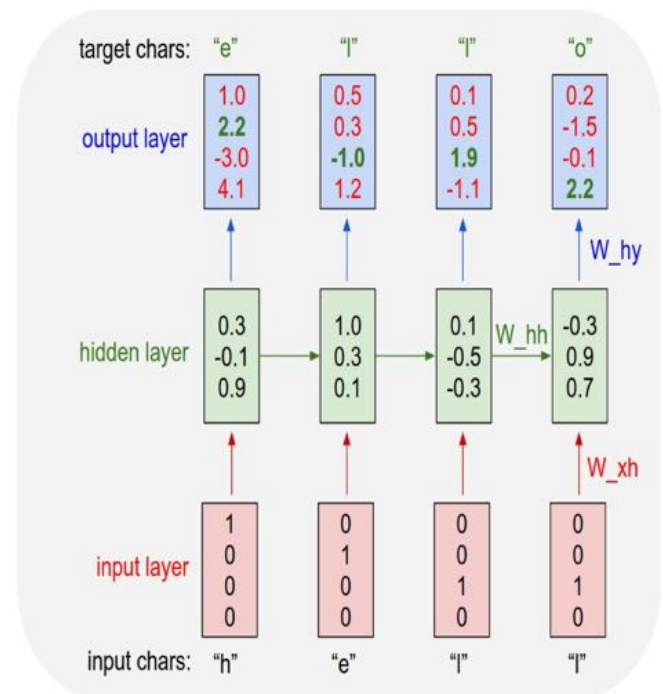
[min-char-rnn.py gist](#)

## Initializations



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

recall:



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

## Main loop

```

1  """Minimal character-level vanilla RNN model. Written by Andrej Karpathy (karpathy@openai.com)"""
2  BSD license
3
4  Import numpy as np
5
6  # Data I/O
7  data = open('input.txt', "r").read() # should be single giant text file
8  chars = list(set(data))
9  data_size, vocab_size = len(data), len(chars)
10 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
11 char_to_ix = {ch:i for i,ch in enumerate(chars)}
12 ix_to_char = {i:ch for i,ch in enumerate(chars)}
13
14 # Hyperparameters
15 hidden_size = 100 # size of hidden layer of neurons
16 seq_length = 25 # number of steps to unroll the rnn for
17 learning_rate = 1e-1
18
19 # model parameters
20 w = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
21 wh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
22 w0 = np.random.randn(hidden_size, 1)*0.01 # hidden to output
23 b = np.zeros((hidden_size, 1)) # hidden bias
24 by = np.zeros((vocab_size, 1)) # output bias
25
26 def init_params(targets, hps):
27     """
28     inputs, targets are both list of integers.
29     hps is an array of initial hidden state
30     returns the lists, gradients on model parameters, and last hidden state
31
32     xs, hs, ys, ps = [], [], [], []
33     h0 = hps[0]
34     loss = 0
35     # forward pass
36     for t in xrange(len(inputs)):
37         xi = np.zeros((vocab_size, 1)) # encode int 1-of-N representation
38         xi[inputs[t]] = 1
39         h0 = np.tanh(dot(w, xi) + w0 + dot(wh, h[-1]) - bh) # hidden state
40         ps.append(h0) # h[-1] = h by = accumulated log probabilities for next step
41         yi = np.zeros((vocab_size, 1)) # probabilities for next step
42         yi_ix = np.argmax(yi) # softmax (cross-entropy loss)
43     # backward pass: compute gradients going backwards
44     dwh, dwt, dby = np.zeros_like(w), np.zeros_like(w0), np.zeros_like(w0)
45     dh0, dht, dwt, dby = np.zeros_like(w), np.zeros_like(w0), np.zeros_like(w0)
46     dhnext = np.zeros_like(h0)
47     for t in reversed(xrange(len(inputs))):
48         dy = np.copy(yi)
49         dy[targets[t]] = 1 # backward into y
50         dht = np.dot(dy, h0[1:])
51         dht = dy
52         dh = np.dot(wt, dy) + dhnext # backward into h
53         dwt = (1 - h0[1] * h0[1]) * dh # backward through tanh nonlinearity
54         dwt = dwt * dwt
55         dwh = np.dot(dwt, w[1:])
56         dwt = np.dot(dwt, w[-1:])
57         dhnext = np.dot(dht, h)
58     for dparam in [dwh, dwt, dby, dh0, dht, dwt]:
59         np.clip(dparam, -4, 4, out=dparam) # clip to mitigate exploding gradients
60     return loss, dwh, dwt, dby, dh0, dht, dwt, h0[inputs[-1]]
61
62 def unroll(seq, seed_ix, n):
63     """
64     sample a sequence of integers from the model
65     h is memory state, seed_ix is seed letter for first time step
66
67     xs = np.zeros((vocab_size, 1))
68     xi = seed_ix - 1
69     loss = 0
70     for t in xrange(n):
71         h = np.tanh(dot(w, xi) + w0 + dot(wh, h) - bh)
72         xi = np.dot(wt, h) + by
73         yi = np.exp(h) / np.sum(np.exp(h))
74         ix = np.random.choice(vocab_size, 1, p=yi)
75         x = np.zeros((vocab_size, 1))
76         x[ix] = 1
77         loss += dot(yi, x)
78     return loss
79
80 # A, B, C, D, E
81 dwh, dwt, dby, dh0, dht, dwt, h0 = init_params([A, B, C, D, E], hps)
82 wh, wt, w0, w1, w2, w3, w4, w5 = np.zeros_like(w), np.zeros_like(w), np.zeros_like(w)
83 dwh0, dwt0 = np.zeros_like(w), np.zeros_like(w)
84 seq_length = hps['seq_length']
85 while True:
86     # prepare inputs [w] = unrolling from left to right in scope seq_length long
87     if seq_length == 0:
88         hps['seq_length'] = 1
89         hps['w'] = w
90         hps['w0'] = w0
91         hps['w1'] = w1
92         hps['w2'] = w2
93         hps['w3'] = w3
94         hps['w4'] = w4
95         hps['w5'] = w5
96         hps['w6'] = w6
97         hps['w7'] = w7
98         hps['w8'] = w8
99         hps['w9'] = w9
100        hps['w10'] = w10
101        hps['w11'] = w11
102        hps['w12'] = w12
103        hps['w13'] = w13
104        hps['w14'] = w14
105        hps['w15'] = w15
106        hps['w16'] = w16
107        hps['w17'] = w17
108        hps['w18'] = w18
109        hps['w19'] = w19
110        hps['w20'] = w20
111        hps['w21'] = w21
112        hps['w22'] = w22
113        hps['w23'] = w23
114        hps['w24'] = w24
115        hps['w25'] = w25
116        hps['w26'] = w26
117        hps['w27'] = w27
118        hps['w28'] = w28
119        hps['w29'] = w29
120        hps['w30'] = w30
121        hps['w31'] = w31
122        hps['w32'] = w32
123        hps['w33'] = w33
124        hps['w34'] = w34
125        hps['w35'] = w35
126        hps['w36'] = w36
127        hps['w37'] = w37
128        hps['w38'] = w38
129        hps['w39'] = w39
130        hps['w40'] = w40
131        hps['w41'] = w41
132        hps['w42'] = w42
133        hps['w43'] = w43
134        hps['w44'] = w44
135        hps['w45'] = w45
136        hps['w46'] = w46
137        hps['w47'] = w47
138        hps['w48'] = w48
139        hps['w49'] = w49
140        hps['w50'] = w50
141        hps['w51'] = w51
142        hps['w52'] = w52
143        hps['w53'] = w53
144        hps['w54'] = w54
145        hps['w55'] = w55
146        hps['w56'] = w56
147        hps['w57'] = w57
148        hps['w58'] = w58
149        hps['w59'] = w59
150        hps['w60'] = w60
151        hps['w61'] = w61
152        hps['w62'] = w62
153        hps['w63'] = w63
154        hps['w64'] = w64
155        hps['w65'] = w65
156        hps['w66'] = w66
157        hps['w67'] = w67
158        hps['w68'] = w68
159        hps['w69'] = w69
160        hps['w70'] = w70
161        hps['w71'] = w71
162        hps['w72'] = w72
163        hps['w73'] = w73
164        hps['w74'] = w74
165        hps['w75'] = w75
166        hps['w76'] = w76
167        hps['w77'] = w77
168        hps['w78'] = w78
169        hps['w79'] = w79
170        hps['w80'] = w80
171        hps['w81'] = w81
172        hps['w82'] = w82
173        hps['w83'] = w83
174        hps['w84'] = w84
175        hps['w85'] = w85
176        hps['w86'] = w86
177        hps['w87'] = w87
178        hps['w88'] = w88
179        hps['w89'] = w89
180        hps['w90'] = w90
181        hps['w91'] = w91
182        hps['w92'] = w92
183        hps['w93'] = w93
184        hps['w94'] = w94
185        hps['w95'] = w95
186        hps['w96'] = w96
187        hps['w97'] = w97
188        hps['w98'] = w98
189        hps['w99'] = w99
190        hps['w100'] = w100
191        hps['w101'] = w101
192        hps['w102'] = w102
193        hps['w103'] = w103
194        hps['w104'] = w104
195        hps['w105'] = w105
196        hps['w106'] = w106
197        hps['w107'] = w107
198        hps['w108'] = w108
199        hps['w109'] = w109
200        hps['w110'] = w110
201        hps['w111'] = w111
202        hps['w112'] = w112
203        hps['w113'] = w113
204        hps['w114'] = w114
205        hps['w115'] = w115
206        hps['w116'] = w116
207        hps['w117'] = w117
208        hps['w118'] = w118
209        hps['w119'] = w119
210        hps['w120'] = w120
211        hps['w121'] = w121
212        hps['w122'] = w122
213        hps['w123'] = w123
214        hps['w124'] = w124
215        hps['w125'] = w125
216        hps['w126'] = w126
217        hps['w127'] = w127
218        hps['w128'] = w128
219        hps['w129'] = w129
220        hps['w130'] = w130
221        hps['w131'] = w131
222        hps['w132'] = w132
223        hps['w133'] = w133
224        hps['w134'] = w134
225        hps['w135'] = w135
226        hps['w136'] = w136
227        hps['w137'] = w137
228        hps['w138'] = w138
229        hps['w139'] = w139
230        hps['w140'] = w140
231        hps['w141'] = w141
232        hps['w142'] = w142
233        hps['w143'] = w143
234        hps['w144'] = w144
235        hps['w145'] = w145
236        hps['w146'] = w146
237        hps['w147'] = w147
238        hps['w148'] = w148
239        hps['w149'] = w149
240        hps['w150'] = w150
241        hps['w151'] = w151
242        hps['w152'] = w152
243        hps['w153'] = w153
244        hps['w154'] = w154
245        hps['w155'] = w155
246        hps['w156'] = w156
247        hps['w157'] = w157
248        hps['w158'] = w158
249        hps['w159'] = w159
250        hps['w160'] = w160
251        hps['w161'] = w161
252        hps['w162'] = w162
253        hps['w163'] = w163
254        hps['w164'] = w164
255        hps['w165'] = w165
256       
```

```

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

```

# min-char-rnn.py gist

## Main loop



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
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 = 256 # 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)*0.41 # input to hidden
22 Whh = np.random.randn(hidden_size, hidden_size)*0.41 # hidden to hidden
23 Why = np.random.randn(hidden_size, hidden_size)*0.41 # 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 not array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = [], [], [], []
34     h = hprev
35     loss = 0
36     # Forward pass
37     for t in xrange(len(inputs)):
38         xs.append([inputs[t]] * hidden_size) # encode 1x1-of-N representation
39         hs.append(h)
40         h = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
41         ys.append(Why * h) # by a generalized log probabilities for next char.
42         ps.append(np.exp(ys[t]) / np.sum(np.exp(ys[t]))) # probabilities for next char
43         loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backwards
45     dxs, dhs, dby = 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(ys[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dby += np.dot(dy, hs[t], T)
52         dy = 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         ddx = ddraw
56         dxs[t] += np.dot(ddraw, xs[t], T)
57         dhs[t] += np.dot(ddraw, hs[t-1], T)
58         dhnext = np.dot(Whh, T, dhs[t])
59     for dparam in [dxs, dhs, dby, dbh, dby]:
60         # clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61     return loss, ddx, dhs, dby, dbh, dby, dhnext, hs[0]
62
63 def sample(x, seed_ix, k):
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     s = np.zeros((vocab_size, 1))
69     x[seed_ix] = 1
70     loss = 0
71     for t in xrange(k):
72         h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
73         y = np.dot(Why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(vocab_size, 1, p=p.ravel())
76         s = np.zeros((vocab_size, 1))
77         ix[ix] = 1
78         loss += np.log(1/p[ix])
79     return loss
80
81 # Main loop
82 n, p = 0, 0
83 mWxh, mWhh, mWhy = np.zeros_like(Wxh), 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
87 while True:
88     # prepare inputs (we're sweeping from left to right in steps seq_length long)
89     if p+seq_length+1 >= len(data) or n == 0:
90         hprev = np.zeros((hidden_size,1)) # reset RNN memory
91         p = 0 # go from start of data
92         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
93         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
94
95         # sample from the model now and then
96         if n % 100 == 0:
97             sample_ix = sample(hprev, inputs[0], 200)
98             txt = ''.join(ix_to_char[ix] for ix in sample_ix)
99             print '----\n%s \n----' % (txt, )
100
101         # forward seq_length characters through the net and fetch gradient
102         loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
103         smooth_loss = smooth_loss * 0.999 + loss * 0.001
104         if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
105
106         # perform parameter update with Adagrad
107         for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
108                                     [dWxh, dWhh, dWhy, dbh, dby],
109                                     [mWxh, mWhh, mWhy, mbh, mby]):
110             mem += dparam * dparam
111             param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
112
113         p += seq_length # move data pointer
114         n += 1 # iteration counter
```

```
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](#)

TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## Main loop

```

1 # Minimal character-level vanilla rnn model. Written by Andrej Karpathy (@karpathy)
2 BSD license
3
4 import numpy as np
5
6 # Data size
7 data = open('input.txt', "r").read() # should be single plain text file
8 chars = list(set(data))
9 data_size, vocab_size = len(data), len(chars)
10 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
11 char_to_ix = { ch:i for i, ch in enumerate(chars)}
12 ix_to_char = { i:ch for i, ch in enumerate(chars)}
13
14 # Hyperparameters
15 hidden_size = 100 # size of hidden layer of neurons
16 seq_length = 25 # number of steps to unroll the rnn for
17 learning_rate = 0.01
18
19 # model parameters
20 wnn = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
21 wnh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
22 why = np.random.randn(hidden_size, vocab_size)*0.01 # hidden to output
23 b0 = np.zeros((hidden_size, 1)) # hidden bias
24 b1 = np.zeros((vocab_size, 1)) # output bias
25
26 def unroll(inputs, targets, hprev):
27     inputs, targets are both list of integers.
28     hprev is int array of initial hidden state
29     returns the loss, gradients of model parameters, and last hidden state
30
31     xs, ys, ps, ps0 = [], [], [], []
32     h0 = np.copy(hprev)
33     loss = 0
34     # forward pass
35     for t in xrange(len(inputs)):
36         xi = np.zeros((hidden_size, 1)) # encode as 1-d of a representation
37         xi[inputs[t]] = 1
38         hy = np.tanh(np.dot(hnn, xi)) + np.dot(hnh, h[-1]) + b0 # hidden state
39         hyo = np.dot(why, hy) + b1 # by a generalized log probabilities for next char
40         yi = np.argmax(hyo) # y = np.argmax(hyo) + 1 # probabilizes for next char
41         loss += -np.log(hyo[targets[t]]) # softmax (cross-entropy loss)
42     # backward pass: compute gradients going backwards
43     dnn, dhn, dhy = np.zeros_like(hnn), np.zeros_like(hnh), np.zeros_like(why)
44     dxi, dxi0 = np.zeros_like(xi), np.zeros_like(b0)
45     dhw0 = np.zeros_like(h0)
46     for t in reversed(xrange(len(inputs))):
47         dy = np.copy(hyo[t])
48         dy[targets[t]] -= 1 # backward data y
49         dxi = np.dot(dy, h[-1, T])
50         dxi0 = np.dot(dy, b1[T])
51         dhn = - np.dot(dxi, h[-1, T]) # dh = backward through both nonlinearity
52         dhn0 = np.dot(dxi0, h[-1, T])
53         dhw0 = np.dot(dhn, T, dhw0)
54     for param in [dnn, dhn, dhy, dxi, dxi0]:
55         np.clip(param, -4, 4, out=param) # clip to mitigate exploding gradients
56     return loss, dnn, dhn, dhy, dxi, dxi0, h0, h[-1][inputs[-1]]
57
58 def unrollT( seed_ix, k):
59     sample = sequence of integers from the model
60     ix is memory state, seed_ix is seed letter for first time step
61     s = np.zeros((vocab_size, 1))
62     s[seed_ix] = 1
63     loss = 0
64     for t in xrange(k):
65         xi = np.tanh(np.dot(hnn, s) + np.dot(hnh, h[-1]) + b0)
66         hy = np.dot(why, h[-1]) + b1
67         yi = np.argmax(hy)
68         ix = np.random.choice(vocab_size, 1, p=xi.ravel())
69         s = np.zeros((vocab_size, 1))
70         s[ix] = 1
71         loss += unroll(s)
72     return loss
73
74 # N, S, K, R
75 nnn, nnnh, nny = np.zeros_like(hnn), np.zeros_like(hnh), np.zeros_like(why)
76 nnn, nnnh, nny = np.zeros_like(hnn), np.zeros_like(hnh), np.zeros_like(why) # memory variables for AdamW
77 nnnh = np.zeros_like(hnh)
78 while True:
79     # warm up inputs [wnts] missing from left to right to input seq_length long
80     if prior_lengths % len(chars) or n == seq_length:
81         n = np.zeros((hidden_size, 1)) # reset rnn memory
82         n = n from start of data
83         inputs = [char_ix[ch] for ch in data[prior_lengths:]]
84         targets = [char_ix[ch] for ch in data[prior_lengths+1:]]
85
86     # sample from the model one and then
87     if n == 0 or n == n:
88         sample_ix = sample(hnn, inputs[-1], 100)
89         text = "".join(ix_to_char[ix] for ix in sample_ix)
90         print "---- %s %s %s" % (text, n, n)
91
92     # forward seq. backwards through the net and fetch gradients
93     loss, dnn, dhn, dhy, dxi, dxi0, dhw0, dhw0 = unroll(inputs, targets, hnn)
94     smooth_loss = smooth_loss * 0.999 + loss * 0.001
95     if n % 100 == 0: print "iter %d: loss %3f, %3f, %3f, %3f, %3f, %3f, %3f, %3f" % (n, smooth_loss, loss, prior_lengths, nnn, nnnh, nny)
96
97     # perform parameter updates with AdamW
98     for param, dparam, nnn in [(dnn, dnn, nnn), (dhn, dhn, nnnh), (dhy, dhy, nny)]:
99         nnn = dparam + dparam
100         param += -learning_rate * (dparam + nnn) / np.sqrt(nnn + 1e-8) # AdamW update
101
102     p = seq_length + word data position
103

```

```

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

```

# min-char-rnn.py gist

## Main loop



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
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 = 256 # 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)*0.41 # input to hidden
22 Wbh = np.random.randn(hidden_size, hidden_size)*0.41 # hidden to hidden
23 Why = np.random.randn(vocab_size, hidden_size)*0.41 # 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 last array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = [], [], [], []
34     h = hprev
35     loss = 0
36     # Forward pass
37     for t in xrange(len(inputs)):
38         xs.append([inputs[t]]) # encode 1x1-of-N representation
39         hs.append(h)
40         h = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Wbh, hs[t-1]) + bh) # hidden state
41         ys.append(h) # by a universalized log probabilities for next char
42         ps.append(np.exp(Why[0,hs[t]] / np.sum(np.exp(Why[0,hs[t]])) + probabilities for next char)
43     loss += -np.log(ps[0][targets[0]]) # softmax (cross-entropy loss)
44     # Backward pass: compute gradients going backwards
45     dhs, dhs_t, dhy = np.zeros_like(hs), np.zeros_like(hs), np.zeros_like(Why)
46     dhs_t, dhy_t = np.zeros_like(hs), np.zeros_like(hs)
47     dhs_t, dhy_t = np.zeros_like(hs), np.zeros_like(hs)
48     for t in reversed(xrange(len(inputs))):
49         dy = np.zeros_like(hs[t])
50         dy[targets[t]] += 1 # backprop into y
51         dhy_t = np.dot(dy, hs[t])
52         dhs_t = np.dot(Why.T, dy) + dhs_t # backprop into h
53         dhs_t = (1 - hs[t]*hs[t]) * dhs_t # backprop through tanh nonlinearity
54         dhs_t, dhy_t = np.dot(dhs_t, Wxh.T), np.dot(dhs_t, Wbh.T)
55         dhs_t, dhy_t = np.dot(dhs_t, Wxh.T), np.dot(dhs_t, Wbh.T)
56         for dparam in [dhs_t, dhy_t, dhs_t, dhy_t]:
57             # clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
58     return loss, dhs_t, dhs_t, dhy_t, dhs_t, dhy_t, hs[t], hs[t]
59
60 def sample(h, seed_ix, k):
61     """
62     sample a sequence of integers from the model
63     h is memory state, seed_ix is seed letter for first time step
64     """
65     x = np.zeros(vocab_size, 1)
66     x[seed_ix] = 1
67     loss = 0
68     for t in xrange(k):
69         h = np.tanh(np.dot(Wxh, x) + np.dot(Wbh, h) + bh)
70         y = np.dot(Why, h) + by
71         p = np.exp(y) / np.sum(np.exp(y))
72         ix = np.random.choice(vocab_size, 1, p=p.ravel())
73         x = np.zeros(vocab_size, 1)
74         x[ix] = 1
75     loss.append(loss)
76     return loss
77
78 # Main loop
79 n, p = 0, 0
80 mWxh, mWbh, mWhy = np.zeros_like(Wxh), np.zeros_like(Wbh), np.zeros_like(Why)
81 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
82 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
83 while True:
84     # prepare inputs (we're sweeping from left to right in steps seq_length long)
85     if p+seq_length+1 >= len(data) or n == 0:
86         hprev = np.zeros((hidden_size,1)) # reset RNN memory
87         p = 0 # go from start of data
88     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
89     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
90
91     # sample from the model now and then
92     if n % 100 == 0:
93         sample_ix = sample(hprev, inputs[0], 200)
94         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
95         print '----\n %s \n----' % (txt, )
96
97     # forward seq_length characters through the net and fetch gradient
98     loss, dWxh, dWbh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
99     smooth_loss = smooth_loss * 0.999 + loss * 0.001
100     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
101
102     # perform parameter update with Adagrad
103     for param, dparam, mem in zip([Wxh, Wbh, Why, bh, by],
104                                   [dWxh, dWbh, dWhy, dbh, dby],
105                                   [mWxh, mWbh, mWhy, mbh, mby]):
106         mem += dparam * dparam
107         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
108
109     p += seq_length # move data pointer
110     n += 1 # iteration counter
```

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

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

## Main loop

```

1  """Minimal character-level vanilla RNN model, written by Andriy Kurachy (KurachyT)"""
2  # BSD license
3  # see: https://github.com/KurachyT/minimal-char-level-vanilla-rnn
4
5  import numpy as np
6
7  # Data I/O
8
9  data = np.loadtxt('data', "r", delimiter=',') should be single plane text file
10 (chars = list(data[0]))
11 data_size, vocab_size = len(data), len(chars)
12 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
13 char_ixs = [0] * data_size
14 for i, char in enumerate(chars):
15     char_ixs[i] = char
16
17 # Hyperparameters
18
19 hidden_size = 100 # size of hidden layer of neurons
20 seq_length = 25 # number of steps to unroll the rnn for
21 learning_rate = 0.01
22
23 # model parameters
24
25 w_hh = np.random.randn(hidden_size, vocab_size) * 0.01 # input to hidden
26 w_hx = np.random.randn(hidden_size, 1) * 0.01 # hidden to hidden
27 w_xh = np.random.randn(vocab_size, hidden_size) * 0.01 # hidden to output
28 b_h = np.zeros(hidden_size, 1) # hidden bias
29 b_y = np.zeros(vocab_size, 1) # output bias
30
31 def softmax(logits, targets, hprev):
32     """
33     logits, targets are both list of integers.
34     hprev is not array of initial hidden state
35     returns the loss, gradients on model parameters, and last hidden state
36     """
37     xs, hys, ys, ps = [], [], [], []
38     h0 = [] # np.copy(hprev)
39     loss = 0
40     # forward pass
41     for i in xrange(len(logits)):
42         xs[i] = np.zeros(vocab_size, 1) # encode 25 1-d of 0 representation
43         h0[i] = hprev
44         h[i] = np.tanh(dot(w_hh, xs[i]) + np.dot(w_hx, h0[i]) + b_h) + hidden_size
45         ys[i] = np.dot(w_xh, h[i]) + b_y # unnormalized log probabilities for next char
46         ps[i] = np.exp(ys[i]) / sum(np.exp(ys)) # probabilities for next char
47         loss += -np.log(ps[targets[i], 0]) # softmax (cross-entropy loss)
48     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
49     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
50     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
51     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
52     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
53     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
54     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
55     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
56     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
57     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
58     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
59     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
60     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
61     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
62     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
63     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
64     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
65     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
66     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
67     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
68     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
69     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
70     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
71     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
72     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
73     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
74     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
75     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
76     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
77     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
78     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
79     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
80     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
81     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
82     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
83     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
84     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
85     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
86     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
87     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
88     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
89     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
90     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
91     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
92     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
93     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
94     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
95     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
96     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
97     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
98     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
99     do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
100    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
101    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
102    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
103    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
104    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
105    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
106    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
107    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
108    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
109    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
110    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
111    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
112    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
113    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
114    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
115    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
116    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
117    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
118    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
119    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
120    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
121    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
122    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
123    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
124    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
125    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
126    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
127    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
128    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
129    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
130    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
131    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
132    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
133    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
134    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
135    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
136    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
137    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
138    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)
139    do_hh, do_hx, do_hy = np.zeros_like(w_hh), np.zeros_like(w_hx), np.zeros_like(w_xh)

```

```

61 n, p = 0, 0
62 mWxh, mWwh, mWhy = np.zeros_like(Wxh), np.zeros_like(Wwh), np.zeros_like(Why)
63 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
64 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
65 while True:
66     # prepare inputs (we're sweeping from left to right in steps seq_length long)
67     if p+seq_length+1 >= len(data) or n == 0:
68         hprev = np.zeros((hidden_size,1)) # reset RNN memory
69         p = 0 # go from start of data
70     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
71     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
72
73     # sample from the model now and then
74     if n % 100 == 0:
75         sample_ix = sample(hprev, inputs[0], 200)
76         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
77         print '----\n %s \n----' % (txt, )
78
79     # forward seq_length characters through the net and fetch gradient
80     loss, dWxh, dWwh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
81     smooth_loss = smooth_loss * 0.999 + loss * 0.001
82     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
83
84     # perform parameter update with Adagrad
85     for param, dparam, mem in zip([Wxh, Wwh, Why, bh, by],
86                                   [dWxh, dWwh, dWhy, dbh, dby],
87                                   [mWxh, mWwh, mWhy, mbh, mby]):
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

```



```

1 # Minimal character-level vanilla NN model. Written by Andrej Karpathy (@karpathy)
2 B0D License
3 """
4
5 # Import numpy as np
6
7
8 # Data I/O
9
10 data = open('input.txt', "r").read() # should be single plain text file
11 chars = list(set(data))
12 data_size, vocab_size = len(data), len(chars)
13 print "Data has %d characters, %d unique." % (data_size, vocab_size)
14 char_to_ix = { ch:i for i, ch in enumerate(chars)}
15 ix_to_char = { i:ch for i, ch in enumerate(chars)}
16
17
18 # Hyperparameters
19 hidden_size = 100 # size of hidden layer of neurons
20 seq_length = 25 # number of steps to unroll the rnn for
21 learning_rate = 1e-1
22
23
24 # model parameters
25 wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
26 wh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
27 wh = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
28 b0 = np.zeros(hidden_size, 1) # hidden bias
29 by = np.zeros(vocab_size, 1) # output bias
30
31
32 def lossfun(inputs, targets, hprev):
33     """
34     inputs, targets are both lists of integers.
35     hprev is list array of initial hidden state
36     returns the loss, gradients on model parameters, and last hidden state
37     """
38
39     ix, ix0, y0, y1 = [], [], [], []
40     h0 = np.copy(hprev)
41     loss = 0
42
43     # forward pass
44     for t in xrange(len(inputs)):
45         xi[t] = np.zeros(vocab_size, 1) # encode 28-1 of a representation
46         xi0[t] = inputs[t] + 1
47         hi[t] = np.tanh(np.dot(wh, xi[t]) + np.dot(hwh, hi[t-1]) + b0) # hidden state
48         yi[t] = np.dot(why, hi[t]) + by # by = generalized log probabilities for next char
49         y0[t] = np.exp(yi[t]) # np.sum(np.exp(yi[t])) # probabilities for next char
50         loss += -np.log(yi[t][targets[t],0]) # softmax (cross-entropy) loss
51
52     # backward pass: compute gradients going backwards
53     dwh, dwh0, dwhy, dy0, dxi, dxi0, dhi, dhi0, dby, dby0 = np.zeros_like(wh), np.zeros_like(wh), np.zeros_like(why),
54     dby = np.zeros_like(b0), np.zeros_like(b0)
55     dxi = np.zeros_like(xi0)
56
57     for t in reversed(xrange(len(inputs))):
58         dy = np.copy(y0[t])
59         dy[targets[t]] -= 1 # backprop into y
60         dxi += np.dot(dby, hi[t-1])
61         dby += dy
62         dhi = np.dot(dxi, why) + dby0 # backprop into h
63         dwh0 += h0[t] * hi[t] # dh = backprop through tanh nonlinearity
64         dwh += dwh0
65         dwh0 = np.dot(dwh, xi0[t-1])
66         dxi0 += np.dot(dwh0, dxi[t-1])
67         dxi0 += np.dot(dwh0, dxi)
68     for dparam in [dwh, dwh0, dwhy, dby, dxi, dxi0]:
69         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
70     return loss, dwh, dwh0, dwhy, dby, dxi, dxi0, dhi, dhi0, dby0
71
72
73 def sample(x, seed_ix, n):
74     """
75     sample a sequence of integers from the model
76     x is memory state, seed_ix is seed letter for first time step
77     """
78     s = np.zeros(vocab_size, 1)
79     s[seed_ix] = 1
80     ix = []
81     for t in xrange(n):
82         h = np.tanh(np.dot(wh, s) + np.dot(hwh, h)) + b0
83         y = np.dot(why, h) + by
84         p = np.exp(y) # np.sum(np.exp(y))
85         ix = np.random.choice(vocab_size, 1, p=p.ravel())
86         s = np.zeros(vocab_size, 1)
87         s[ix] = 1
88         ix.append(ix)
89     return ix
90
91
92 # s, h, p = 0, 0, 0
93 dwh, dwh0, dwhy, dy0, dxi, dxi0, dhi, dhi0, dby, dby0 = np.zeros_like(wh), np.zeros_like(wh), np.zeros_like(why),
94     dby, np.zeros_like(b0), np.zeros_like(b0)
95 wh, wh0, why, dy0, dxi, dxi0, dhi, dhi0, dby, dby0 = memory variables for adapted
96 smooth_loss = -logp0 + vocab_size * seq_length # loss at iteration 0
97 while True:
98     # prepare inputs (we're sampling from left to right in steps seq_length long)
99     if p+seq_length == len(data) or n == 0:
100         hprev = np.zeros(hidden_size, 1) # reset rnn memory
101         s = s # go from start of data
102         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
103         targets = [char_to_ix[ch] for ch in data[p+p+seq_length-1]]
104
105     # sample from the model now and then
106     if n % 100 == 0:
107         sample_ix = sample(hprev, inputs[0], 200)
108         ix = ""
109         for ix0, ix1, ix2, ix3 in zip(sample_ix, sample_ix, sample_ix, sample_ix):
110             print "%s" % "%s" % ix0 + " " + ix1 + " " + ix2 + " " + ix3
111
112     # forward unroll characters through the net and fetch gradients
113     loss, dwh, dwh0, dwhy, dy0, dxi, dxi0, dhi, dhi0, dby, dby0 = lossfun(inputs, targets, hprev)
114     smooth_loss = smooth_loss + loss # dxi, dxi0, dby, dby0
115     if n % 100 == 0: print "iter %d, loss: %f, %f, %f, %f, %f" % (n, smooth_loss) # print progress
116
117
118 # perform parameter update with Adam
119 for dparam, dparam0 in zip([dwh, dwh0, dwhy, dy0, dxi, dxi0, dhi, dhi0, dby, dby0],
120     [dwh, dwh0, dwhy, dy0, dxi, dxi0, dhi, dhi0, dby, dby0]):
121     new_dparam = dparam
122     dparam = -learning_rate * dparam + np.sqrt(new_dparam ** 2 + 1) # adapted update
123
124
125 p = seq_length # move data pointer
126 n = 1 # iteration count

```



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

- backward pass (compute param gradient)

```
59     for dparam in [dwxx, dwhh, dwxy, dbx, dby]:
60         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61     return loss, dwxx, dwhh, dwxy, dbx, dby, hs[len(inputs)-1]
```

# min-char-rnn.py gist



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
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 = 256 # 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)*0.4 # input to hidden
22 Wwh = np.random.randn(hidden_size, hidden_size)*0.4 # hidden to hidden
23 Why = np.random.randn(hidden_size, vocab_size)*0.4 # 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(Wxh, xs[t]) + np.dot(Wwh, 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: calculate gradients going backwards
46     dxt, dht, dhy, dbh, dby = np.zeros_like(Wxh), np.zeros_like(Wwh), np.zeros_like(Why)
47     dht, dhy = np.zeros_like(bh), np.zeros_like(by)
48     dxtnext = np.zeros_like(xs[-1])
49     for t in reversed(xrange(len(inputs))):
50         dy = np.copy(dxtnext)
51         dy[targets[t]] += 1 # backprop into y
52         dht = np.dot(dy, Why.T) + dxtnext # backprop into h
53         dxt = (1 - hs[t]**2) * dht # dh * dht through tanh nonlinearity
54         dxtnext = dxt
55         dxt = np.dot(dxt, Wxh.T) + dxtnext
56         dht = np.dot(dht, Wwh.T)
57         dxtnext = np.dot(dxt, Wxh.T) + dxtnext
58         for param in [Wxh, Wwh, Why, bh, by]:
59             # clip to mitigate exploding gradients
60             return loss, dxt, dht, dhy, dbh, dby, hs[-1], ps[-1]
61
62 def sample(x, seed_ix, k):
63     """
64     sample a sequence of integers from the model
65     h is memory state, seed_ix is seed letter for first time step
66     """
67     x = np.zeros((vocab_size, 1))
68     x[seed_ix] = 1
69     loss = 0
70     for t in xrange(k):
71         h = np.tanh(np.dot(Wxh, x) + np.dot(Wwh, h) + bh)
72         y = np.dot(Why, h) + by
73         p = np.exp(y) / np.sum(np.exp(y))
74         ix = np.random.choice(vocab_size, 1, p=p.ravel())
75         x = np.zeros((vocab_size, 1))
76         x[ix] = 1
77         loss += lossFun([x], [ix], h)
78     return loss
79
80 # Main loop
81 N, n = 0, 0
82 Wxh, Wwh, Why, bh, by = np.zeros_like(Wxh), np.zeros_like(Wwh), np.zeros_like(Why)
83 Wxh, Wwh, Why, bh, by = np.zeros_like(Wxh), np.zeros_like(Wwh), np.zeros_like(Why)
84 smooth_loss = -np.log(1/vocab_size)/seq_length # loss at iteration 0
85 while True:
86     # Prepare inputs (we're sampling from left to right in steps seq_length)
87     if print_length % seq_length == 0 or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset rnn memory
89         n = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[n:print_length]]
91         targets = [char_to_ix[ch] for ch in data[n+1:print_length+1]]
92
93     # Sample from the model and train
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(chars[ix] for ix in sample_ix)
97         print '----%s%s----' % (txt, )
98
99     # Forward seq_length characters through the rnn and fetch gradients
100     loss, dxt, dht, dhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101     smooth_loss = smooth_loss * 0.99 + loss * 0.01
102     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104     # Perform parameter update with Adagrad
105     for param, dparam, nnn in zip([Wxh, Wwh, Why, bh, by],
106                                   [dxt, dht, dhy, dbh, dby],
107                                   [dxt, dht, dhy, dbh, dby]):
108         nnn += dparam * dparam
109         param -= learning_rate * dparam / np.sqrt(nnn + 1e-4) # adagrad update
110
111     n += seq_length # move data pointer
112     n += 1 # increment counter
```



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

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

$$y_t = W_{hy}h_t$$

Softmax classifier

# min-char-rnn.py gist



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```

1 # Minimal character-level vanilla rnn model. Written by Andrej Karpathy (karpathy)
2 BSD license
3
4 import numpy as np
5
6 # Data I/O
7 data = open('input.txt', 'r').read() # should be simple plain text file
8 chars = list(set(data))
9 data_size, vocab_size = len(data), len(chars)
10 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
11 char_to_ix = { ch:i for i, ch in enumerate(chars) }
12 ix_to_char = { i:ch for i, ch in enumerate(chars) }
13
14 # Hyperparameters
15 hidden_size = 256 # size of hidden layer of neurons
16 seq_length = 25 # number of steps to unroll the rnn for
17 learning_rate = 0.01
18
19 # Model parameters
20 wh = np.random.randn(hidden_size, vocab_size)*0.4 # input to hidden
21 whh = np.random.randn(hidden_size, hidden_size)*0.4 # hidden to hidden
22 why = np.random.randn(vocab_size, hidden_size)*0.4 # hidden to output
23 bh = np.zeros((hidden_size, 1)) # hidden bias
24 by = np.zeros((vocab_size, 1)) # output bias
25
26 def lossfun(inputs, targets, hprev):
27     """
28     inputs, targets are both list of integers
29     hprev is vec array of initial hidden state
30     returns the loss, gradients on model parameters, and last hidden state
31     """
32     xs, hs, ys, ps = [], [], [], []
33     hs[-1] = np.copy(hprev)
34     loss = 0
35     # Forward pass
36     for t in xrange(len(inputs)):
37         xs[t] = np.zeros((vocab_size, 1)) # encode in 1-of-n representation
38         xs[t][inputs[t]] = 1
39         hs[t] = np.tanh(np.dot(wh, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
40         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
41         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
42         loss += -np.log(ps[t][targets[t]]) # softmax (cross-entropy) loss
43     # Backward pass: backwards through time for backwards
44     dx, dhs, dhs, dby = np.zeros_like(why), np.zeros_like(whh), np.zeros_like(why)
45     dhs, dby = np.zeros_like(bh), np.zeros_like(bh)
46     dhnext = np.zeros_like(hs[0])
47     for t in reversed(xrange(len(inputs))):
48         dy = np.copy(ps[t])
49         dy[targets[t]] -= 1 # backprop into y
50         dwhy += np.dot(dy, hs[t].T)
51         dby += dy
52         dh = np.dot(why.T, dy) + dhnext # backprop into h
53         dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
54         dbh += dhraw
55         dwhx += np.dot(dhraw, xs[t].T)
56         dwhh += np.dot(dhraw, hs[t-1].T)
57         dhnext = np.dot(whh.T, dhraw)
58     for dparam in [dwhx, dwhh, dwhy, dbh, dby]:
59         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
60     return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
61
62 def sample(h, seed_ix, k):
63     """
64     sample a sequence of integers from the model
65     h is hidden state, seed_ix is seed letter for first time step
66     """
67     x = np.zeros((vocab_size, 1))
68     x[seed_ix] = 1
69     loss = 0
70     for t in xrange(k):
71         h = np.tanh(np.dot(wh, x) + np.dot(whh, h) + bh)
72         y = np.dot(why, h) + by
73         p = np.exp(y) / np.sum(np.exp(y))
74         ix = np.random.choice(range(vocab_size), p=p, rand())
75         x = np.zeros((vocab_size, 1))
76         x[ix] = 1
77         loss += lossfun([x], [ix], h)
78     return loss
79
80 # Main loop
81 N, n = 0, 0
82 dx, dhs, dhs, dby = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
83 wh, whh, why, dbh, dby = np.zeros_like(wh), np.zeros_like(whh), np.zeros_like(why)
84 smooth_loss = -np.log(1/vocab_size)/seq_length # loss at iteration 0
85 while True:
86     # Prepare inputs (we're sampling from left to right in steps seq_length long)
87     if print_length % seq_length == 0 or n == 0:
88         hprev = np.zeros((hidden_size, 1)) # reset rnn memory
89         n = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[:print_length]]
91         targets = [char_to_ix[ch] for ch in data[:print_length-1]]
92
93         # Sample from the model now and then
94         if n % 100 == 0:
95             sample_ix = sample(hprev, inputs[0], 200)
96             txt = ''.join(chars[ix] for ix in sample_ix)
97             print '----%s%s----' % (txt, )
98
99         # Forward seq_length characters through the rnn and fetch gradients
100         loss, dx, dhs, dhs, dby, dby, hprev = lossfun(inputs, targets, hprev)
101         smooth_loss = smooth_loss * 0.99 + loss * 0.01
102         if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104         # Perform parameter update with Adagrad
105         for param, dparam, new in zip([wh, whh, why, bh, by],
106                                     [dx, dhs, dhs, dby, dby],
107                                     [dx, dhs, dhs, dby, dby]):
108             new += dparam * dparam
109             param -= learning_rate * dparam / np.sqrt(new + 1e-4) # adagrad update
110
111         p = seq_length # move data pointer
112         n += 1 # increment counter

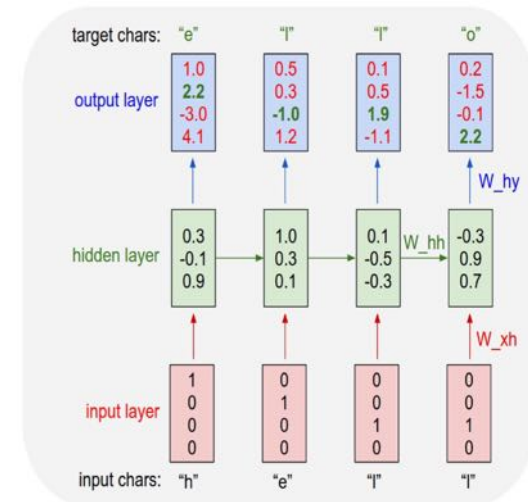
```

```

44 # backward pass: compute gradients going backwards
45 dwhx, dwhh, dwhy = np.zeros_like(wh), 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     dwhx += np.dot(dhraw, xs[t].T)
57     dwhh += np.dot(dhraw, hs[t-1].T)
58     dhnext = np.dot(whh.T, dhraw)
59 for dparam in [dwhx, dwhh, dwhy, dbh, dby]:
60     np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61 return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]

```

recall:





# min-char-rnn.py gist



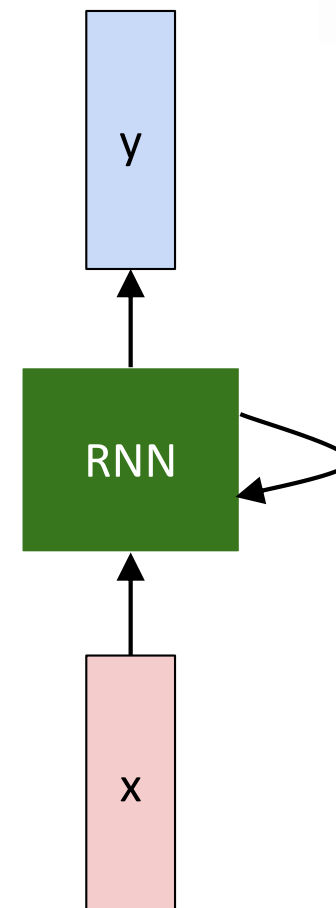
TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
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 = 0.01
19
20 # Model parameters
21 Wxh = np.random.randn(hidden_size, vocab_size)*0.41 # input to hidden
22 Whh = np.random.randn(hidden_size, hidden_size)*0.41 # hidden to hidden
23 Why = np.random.randn(hidden_size, vocab_size)*0.41 # 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 last array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = [], [], [], []
34     h = hprev
35     loss = 0
36     # Forward pass
37     for t in xrange(len(inputs)):
38         xs.append([inputs[t]] * hidden_size) # encode 1x1-of-N representation
39         hs.append(h)
40         h = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[-1]) + bh) # hidden state
41         ys.append(Why * h) # by a generalized log probabilities for next char.
42         ps.append(np.exp(ys[t]) / np.sum(np.exp(ys[t]))) # probabilities for next char
43         loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)
44     # Backward pass: backward gradients going backward
45     dhs, dhs_t, dhy = np.zeros_like(hs), np.zeros_like(hs), np.zeros_like(ys)
46     dhs_t, dhy = np.zeros_like(hs), np.zeros_like(ys)
47     dhs_t = np.zeros_like(hs_t)
48     for t in reversed(xrange(len(inputs))):
49         dy = np.zeros_like(ys[t])
50         dy[targets[t]] = 1 # backprop into y
51         dhy_t = np.dot(dy, hs[t], T)
52         dhy = dy
53         dh = np.dot(Why.T, dy) + dhs_t # backprop into h
54         dhs_t = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55         dhs_t = dhs_t
56         dhs_t = np.dot(dhs_t, xs[t], T)
57         dhs_t = np.dot(dhs_t, T)
58         dhs_t = np.dot(dhs_t, T)
59         for qparam in [Wxh, Why, Whh, bh, by]:
60             # clip [qparam, -5, 5, outparam] to clip to mitigate exploding gradients
61             return loss, dhs, dhs_t, dhy, dhs, dhs_t, dhy, hs[-1]
62
63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     x[seed_ix] = 1
70     ixes = []
71     for t in xrange(n):
72         h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
73         y = np.dot(Why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p.ravel())
76         x = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79     return ixes
80
81 # Main loop
82 h, h_t, dhs, dhs_t, dhy, dhs, dhs_t, dhy, hprev = lossfun(inputs, targets, hprev)
83 h_t, h_t, dhs, dhs_t, dhy, dhs, dhs_t, dhy, hprev = lossfun(inputs, targets, hprev)
84 smooth_loss = -np.log(1/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # Print progress (we're sampling from left to right in steps seq_length long)
87     if print_length % seq_length == 0 or n == 0:
88         hprev = np.zeros((hidden_size, 1)) # reset rnn memory
89         # h = 0 go from start of data
90         inputs = [char_to_ix[ch] for ch in data[:print_length]]
91         targets = [char_to_ix[ch] for ch in data[print_length-1]]
92
93         # Sample from the model now and then
94         if n % 100 == 0:
95             sample_ix = sample(hprev, inputs[0], 200)
96             txt = ''.join(chars[ix] for ix in sample_ix)
97             print '----%s%s-----' % (txt, )
98
99         # Forward seq_length characters through the rnn and fetch gradients
100         loss, dhs, dhs_t, dhy, dhs, dhs_t, dhy, hprev = lossfun(inputs, targets, hprev)
101         smooth_loss = smooth_loss * 0.99 + loss * 0.01
102         if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104         # Perform parameter update with Adagrad
105         for qparam, param in zip([Wxh, Why, Whh, bh, by],
106                                 [dhs, dhs_t, dhy, dhs, dhs_t, dhy]):
107             param += -learning_rate * qparam / np.sqrt(param + 1e-4) # adagrad update
108
109         p = seq_length # move data pointer
110         n += 1 # iteration counter
```

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



<p class="clear"><Products: Laser Printers> The fundamental everyday requirement for mono and colour laser printing throughout today's offices is perfectly met with the extensive Epson AcuLaser printer range. The latest AcuLaser printer range offers users exceptionally Epson AcuLaser C1900. Networked compact colour laser printer for professional enterprises. Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more: with the Epson AcuLaser C1900, Epson brings both colour and monochrome laser printing together at a black and white price. more Where to Buy Support Epson AcuLaser C1000. The fastest colour laser printer in its class. The perfect printer for small businesses and work groups, the Epson AcuLaser C1000 prints high volumes in black and white and vibrant colour, at high speed and with low running costs. more Where to Buy High quality resolution, 2400dpi R7. Large paper capacity, 600 sheets, expandable up to 1,600 sheets. Compatible Windows and Mac. High speed USB and EpsonNet 10/100 Base Tx Ethernet interfaces as standard. \* Epson AcuLaser Resolution Improvement Technology (EpsonNet 10/100 Base Tx Ethernet standard with Epson AcuLaser C1000) model only. AcuLaser C1000: 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing as standard AcuLaser C1000: 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing, 10/100BaseTX Ethernet Interface. Networked compact colour laser printer for professional enterprises. Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more: with the Epson AcuLaser C1900, Epson brings both colour and monochrome laser printing together at a black and white price. Key Features cost effective mono printing for day to day business needs and vivid versatile colour when required. search Search Epson UK Epson AcuLaser C900. Outstanding professional colour printing for business. Add colour to your business with the Epson AcuLaser C900 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high performance black and white production. more Where to Buy Support As cost efficient to run as a mono-only laser printer. Paper capacity of 700 sheets from two media sources. Easy to operate with advanced printer driver. Memory expandable from 32MB to 1024MB. Pre-configured models available with Wireless 802.11b, Adobe® PostScript® 3™ and bi-sided printing. The AcuLaser C1900 is available in 5 configurations: - AcuLaser C1900S: with 32MB, 200 Sheet MP Tray, 10/100BaseTX Networking - AcuLaser C1900: with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking Support Epson AcuLaser C4100 High performance colour lasers for all your business printing needs. The Epson AcuLaser C4100 provides businesses with a high performance colour and monochrome printing solution. It adds crucial colour to your business, while producing high quality monochrome output at lower costs than many monochrome-only printers, and its just as easy to operate. So now there's no reason to buy two printers, because perfect monochrome and colour solutions are available in one. more Where to Buy Support Epson AcuLaser C6000 Professional high performance A3/V colour laser printer. Epson AcuLaser C6000 is the perfect professional printing solution for users who require exceptional quality colour and mono output on a range of media formats from C5 up to A3/V in size. The Epson AcuLaser C6000 is able to achieve superb print quality by utilising a combination of Epson's exclusive AcuLaser Color Laser Technologies. more Where to Buy Support - AcuLaser C1900PS: with Adobe® PostScript® 3™, 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - AcuLaser C1900D: with Duplex unit (two sided printing) 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - AcuLaser C1900 WFi: with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, Wireless Networking facility. Add colour to your business with the Epson AcuLaser C900 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high Support Epson AcuLaser C4000 High performance colour laser. The Epson AcuLaser C4000 provides businesses with high performance colour and monochrome printing solutions. more Where to Buy Epson AcuLaser C9100 High speed A3 colour laser printer. Why have separate black and white and colour printers when you can have the Epson AcuLaser C9100? Epson has taken the lead in laser technology to deliver a complete high-performance solution for all your colour and mono printing needs. Support EPL-6200L High performance A4 mono laser professional printers. The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more Where to Buy Support EPL-6200 High performance A4 mono laser professional printers. The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more performance black and white production. For the first time, you can now bring the power of high quality colour to your documents without suffering the high costs or low speeds traditionally associated with colour.







## Sonnet 116 – Let me not ...

*by William Shakespeare*

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



at first:

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

train more

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

train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of  
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort  
how, and Gogition is so overelical and 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 oftended him.  
Pierre aking his soul came to the packs and drove up his father-in-law women.



PANDARUS:

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

Second Senator:

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

DUKE VINCENTIO:

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

Second Lord:

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

Clown:

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

VIOLA:

I'll drink it.

VIOLA:

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

KING LEAR:

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





# open source textbook on algebraic geometry



The screenshot shows the homepage of The Stacks Project. At the top is the logo and title 'The Stacks Project'. Below it is a navigation bar with links: [home](#), [about](#), [tags explained](#), [tag lookup](#), [browse](#), [search](#), [bibliography](#), [recent comments](#), [blog](#), and [add slogans](#). The main content is divided into two columns. The left column is titled 'Browse chapters' and contains a table with 10 rows. The right column is titled 'Parts' and contains a list of 8 items. Below the 'Parts' list is a section titled 'Statistics' which states: 'The Stacks project now consists of' followed by a list: 455910 lines of code, 14221 tags (56 inactive tags), and 2366 sections.

| Part          | Chapter                 | online                 | TeX source          | view pdf            |
|---------------|-------------------------|------------------------|---------------------|---------------------|
| Preliminaries |                         |                        |                     |                     |
|               | 1. Introduction         | <a href="#">online</a> | <a href="#">tex</a> | <a href="#">pdf</a> |
|               | 2. Conventions          | <a href="#">online</a> | <a href="#">tex</a> | <a href="#">pdf</a> |
|               | 3. Set Theory           | <a href="#">online</a> | <a href="#">tex</a> | <a href="#">pdf</a> |
|               | 4. Categories           | <a href="#">online</a> | <a href="#">tex</a> | <a href="#">pdf</a> |
|               | 5. Topology             | <a href="#">online</a> | <a href="#">tex</a> | <a href="#">pdf</a> |
|               | 6. Sheaves on Spaces    | <a href="#">online</a> | <a href="#">tex</a> | <a href="#">pdf</a> |
|               | 7. Sites and Sheaves    | <a href="#">online</a> | <a href="#">tex</a> | <a href="#">pdf</a> |
|               | 8. Stacks               | <a href="#">online</a> | <a href="#">tex</a> | <a href="#">pdf</a> |
|               | 9. Fields               | <a href="#">online</a> | <a href="#">tex</a> | <a href="#">pdf</a> |
|               | 10. Commutative Algebra | <a href="#">online</a> | <a href="#">tex</a> | <a href="#">pdf</a> |

Parts

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

Statistics

The Stacks project now consists of

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

Latex source



For  $\bigoplus_{i=1, \dots, n} \mathcal{L}_{i, \bullet} = 0$ , 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 comparico 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  $\text{Sch}_{\text{fppf}}$  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  $x, x', x'' \in S'$  such that  $\mathcal{O}_{X, x'} \rightarrow \mathcal{O}_{X, x''}$  is separated. By Algebra, Lemma ?? we can define a map of complexes  $\text{GL}_{\mathcal{O}_X}(x'/S^n)$  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

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

is a unique morphism of algebraic stacks. Note that

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

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  $X_{\text{space}, \text{étale}}$  which gives an open subspace of  $X$  and  $T$  equal to  $S_{Z_{\text{ét}}}$ , see Descent, Lemma ?? . Namely, by Lemma ?? we see that  $R$  is geometrically regular over  $S$ .

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

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

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

When in this case of to show that  $\mathcal{Q} \rightarrow \mathcal{C}_{2/X}$  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  $U = \lim X_i$ .  $\square$

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

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

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

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

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

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



*Proof. Omitted.* □

**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*

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

*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  $Z$  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  $U \subset X$  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*

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

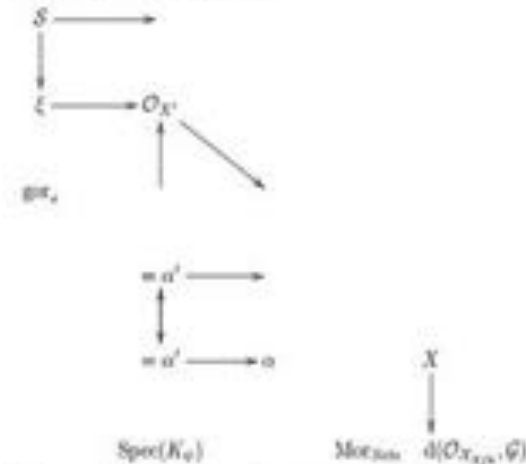
*be a morphism of algebraic spaces over  $S$  and  $Y$ .*

*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



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  $\mathcal{F}$ . 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 Lemma ??.

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

$$\mathcal{O}_{X,S} \rightarrow \mathcal{F}_T \rightarrow \mathcal{O}_{X_{\text{étale}}} \rightarrow \mathcal{O}_{X_1}^{-1} \mathcal{O}_{X_1}(\mathcal{O}_{X_1}^{\mathcal{F}})$$

is an isomorphism of covering of  $\mathcal{O}_{X_1}$ . 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  $\mathcal{O}_{X_1}$  is a closed immersion, see Lemma ?? This is a sequence of  $\mathcal{F}$  is a similar morphisms.







GitHub repository page for **torvalds / linux**.

Repository statistics: 520,037 commits, 1 branch, 420 releases, 5,039 contributors.

Current branch: **master** - **linux** / +

Latest commit: **4b1706927d** (torvalds authored 9 hours ago)

| Directory     | Commit Message                                                                  | Time Ago     |
|---------------|---------------------------------------------------------------------------------|--------------|
| Documentation | Merge git://git.kernel.org/pub/scm/linux/kernel/git/hab/target-pending          | 6 days ago   |
| arch          | Merge branch 'x86-urgent-for-linux' of git://git.kernel.org/pub/scm/...         | a day ago    |
| block         | block: discard bdi_unregister() in favour of bdi_destroy()                      | 9 days ago   |
| crypto        | Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6          | 10 days ago  |
| drivers       | Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux         | 9 hours ago  |
| firmware      | firmware/ihex2fw.c: restore missing default in switch statement                 | 2 months ago |
| fs            | vfs: read file_handle only once in handle_to_path                               | 4 days ago   |
| include       | Merge branch 'perf-urgent-for-linux' of git://git.kernel.org/pub/scm/...        | a day ago    |
| init          | init: fix regression by supporting devices with major:minor:offset to...        | a month ago  |
| linux         | Minor updates 'The Linux' of all-2019 kernel patches (includes: fix regression) | a month ago  |

Right sidebar options:

- Code
- Pull requests (74)
- Pulse
- Graphs
- HTTPS clone URL: `https://github.com/torvalds/linux.git`
- You can clone with **HTTPS**, **SSH**, or **Subversion**.
- Clone in Desktop
- Download ZIP



# Generated C code

```
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << 1))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff8) & 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 ");
}
```



```
/*
 * 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/setow.h>
#include <asm/pgproto.h>
```



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

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

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

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

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



# Searching for interpretable cells

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

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





# Searching for interpretable cells

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

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

quote detection cell





# Searching for interpretable cells

Cell sensitive to position in line:

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

line length tracking cell



# Searching for interpretable cells

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

if statement cell



# Searching for interpretable cells

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

quote/comment cell



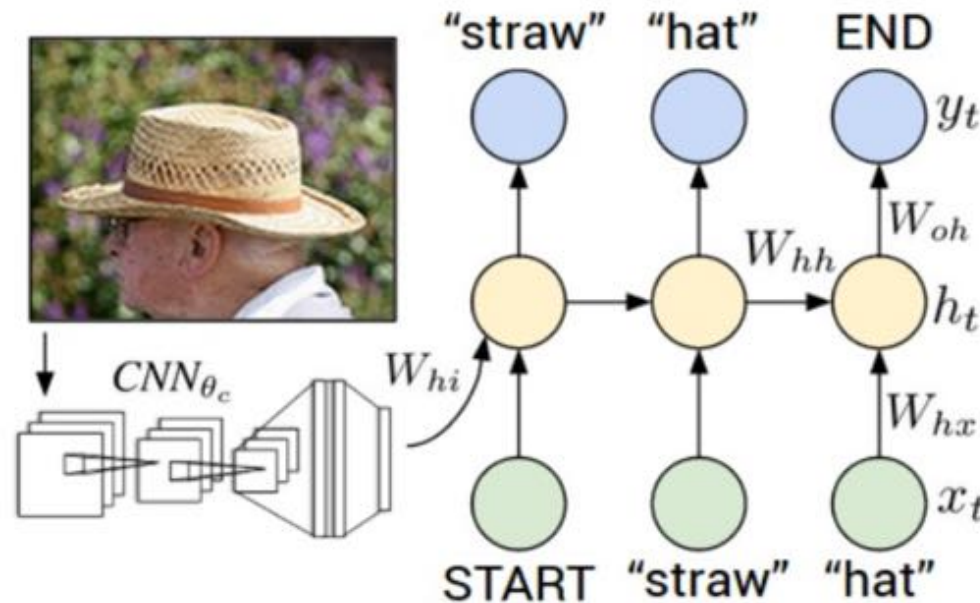
# Searching for interpretable cells

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

code depth cell



# Image Captioning

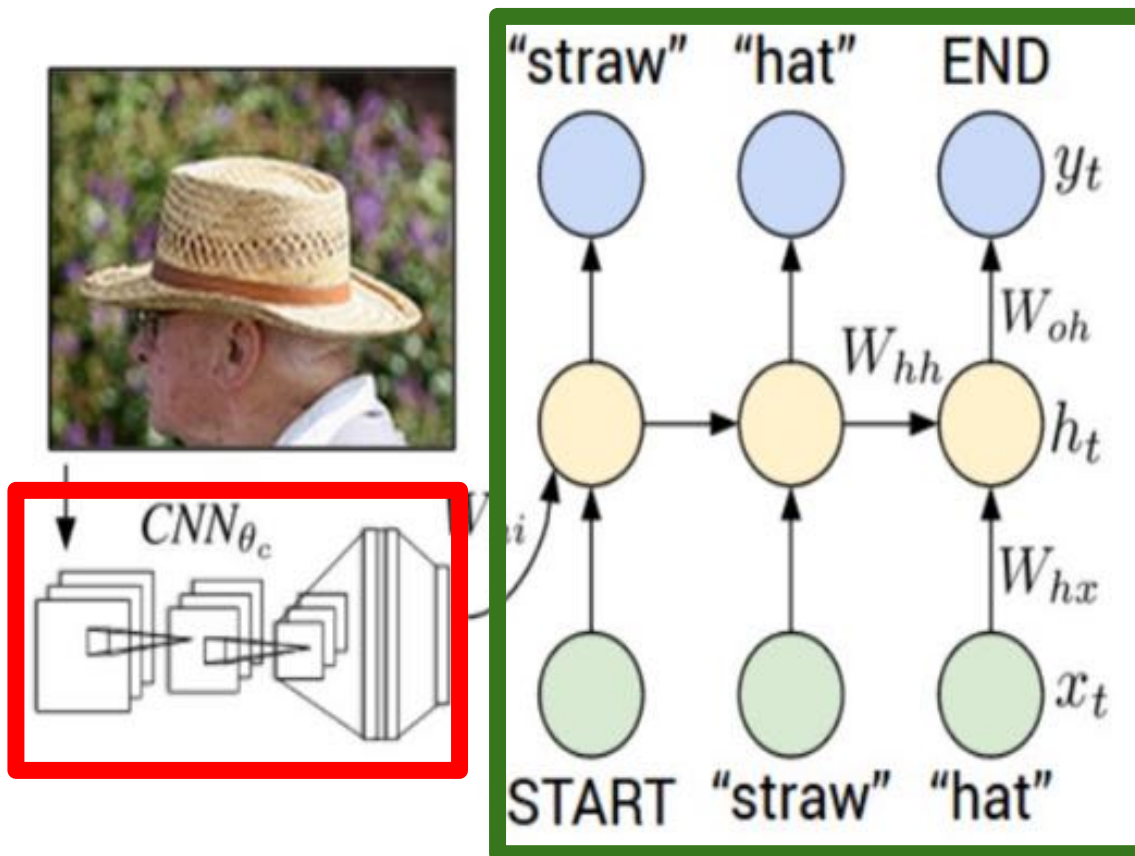


- Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
- Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
- Show and Tell: A Neural Image Caption Generator, Vinyals et al.
- Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
- Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick





# Image Captioning Recurrent Neural Network



**Convolutional Neural Network**





# Image Captioning



test image





image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax

X



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

x0  
<START  
>

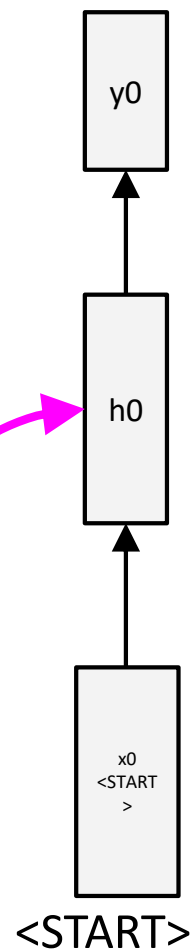
<START>



V



test image



**before:**

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

**now:**

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

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

y0

h0

x0  
<START  
>

straw

sample!

<START>



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

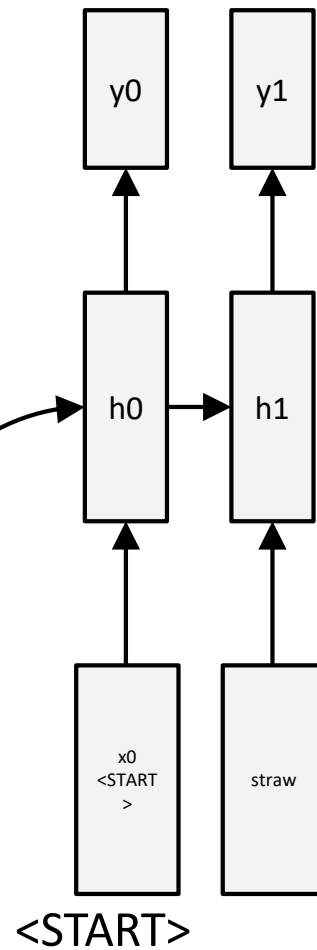
maxpool

FC-4096

FC-4096



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

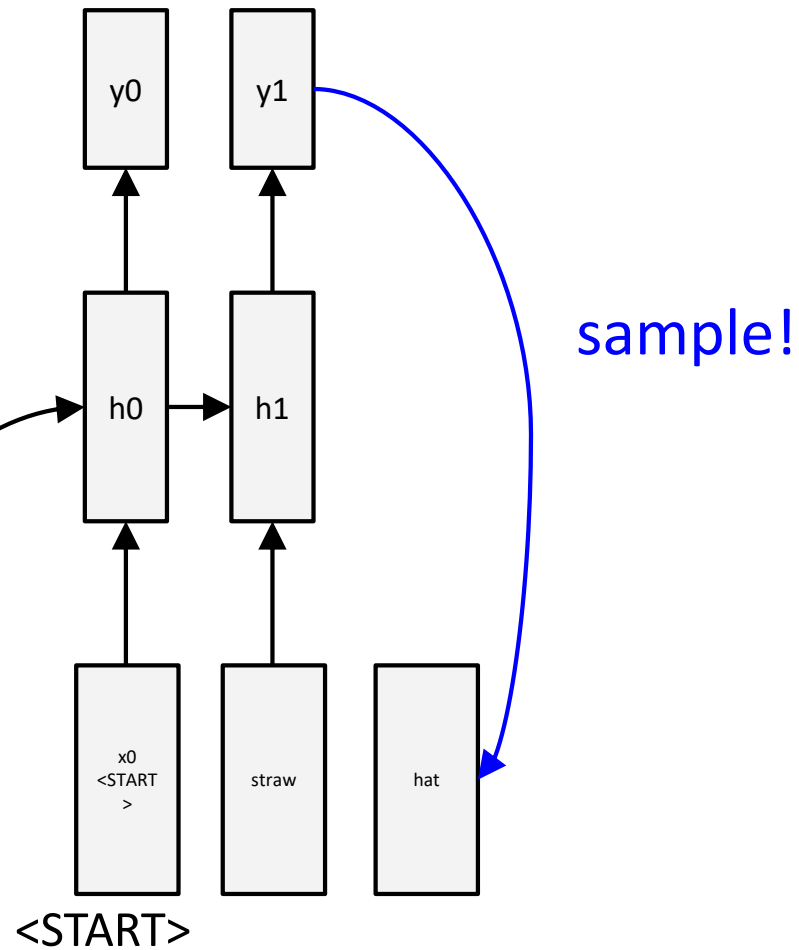
maxpool

FC-4096

FC-4096



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

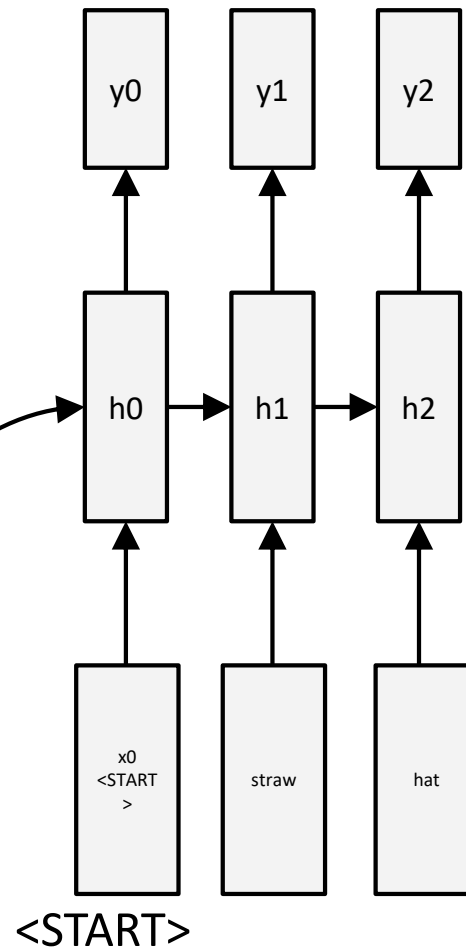
maxpool

FC-4096

FC-4096

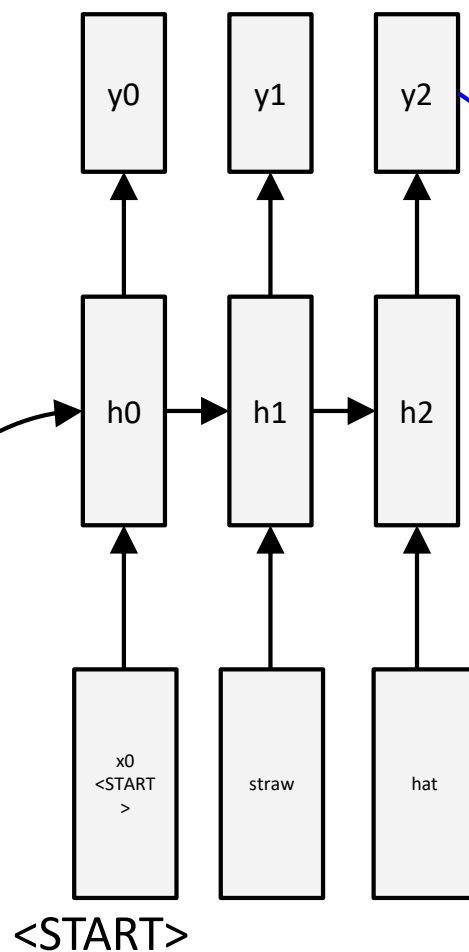


test image





test image



sample  
<END> token  
=> finish.

# Image Sentence Datasets

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



## Microsoft COCO

*[Tsung-Yi Lin et al. 2014]*

[mscoco.org](http://mscoco.org)

currently:

~120K images

~5 sentences each







"man in black shirt is playing guitar."



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



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."







"man in black shirt is playing guitar."



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



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"a young boy is holding a baseball bat."



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



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

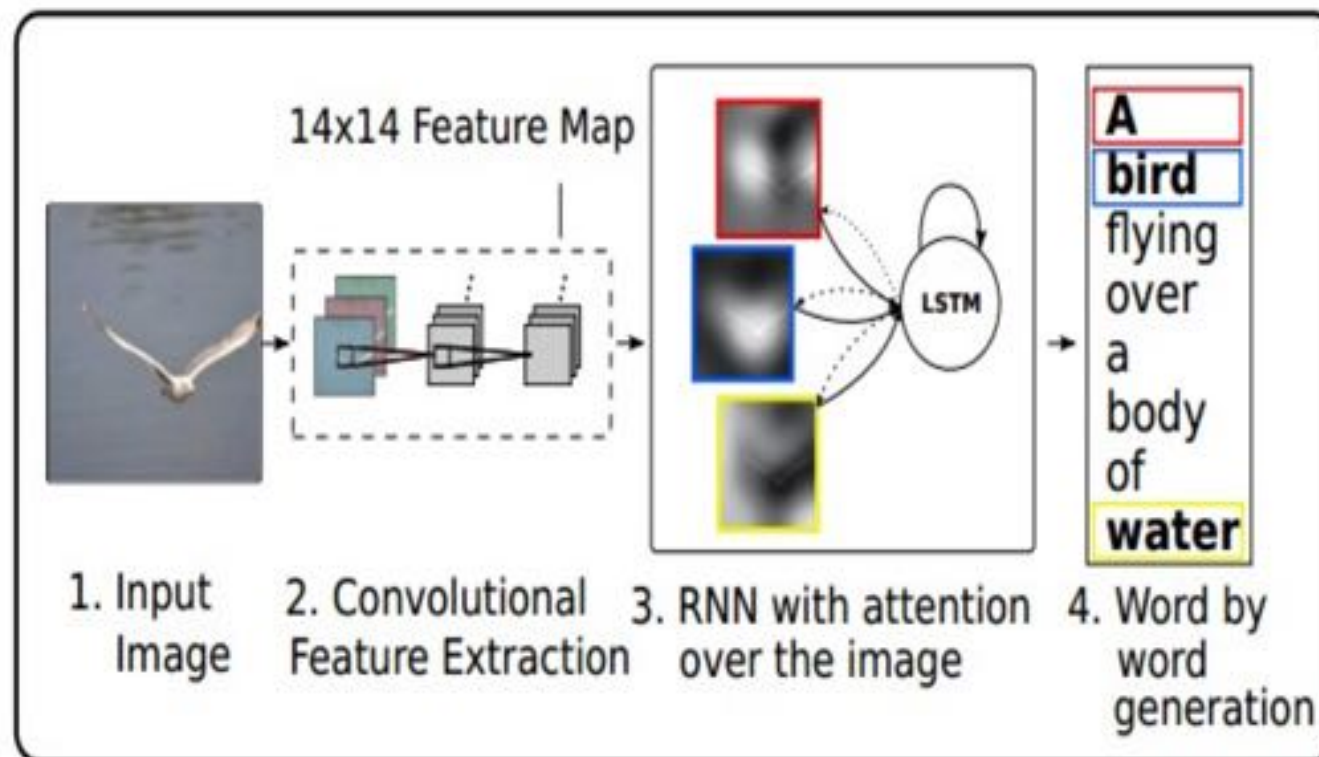


"a horse is standing in the middle of a road."



# Preview of fancier architectures

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



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

# LSTM: Long Short-Term Memory

Vanilla RNN:

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

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

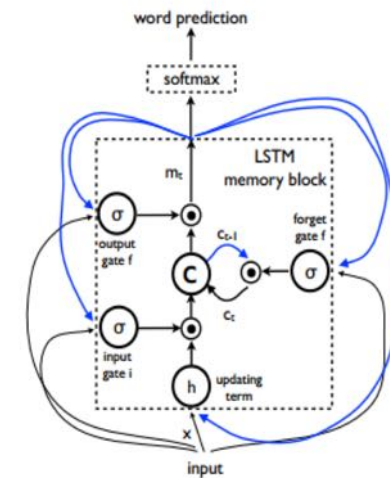
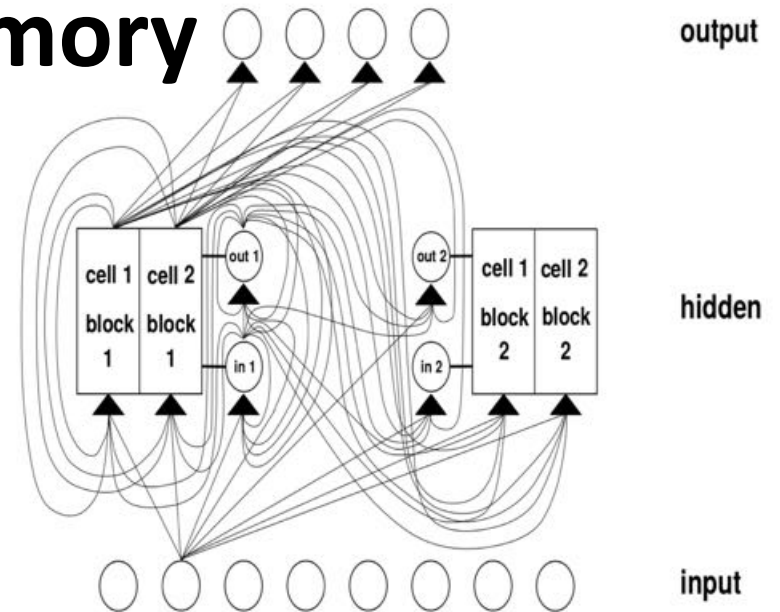
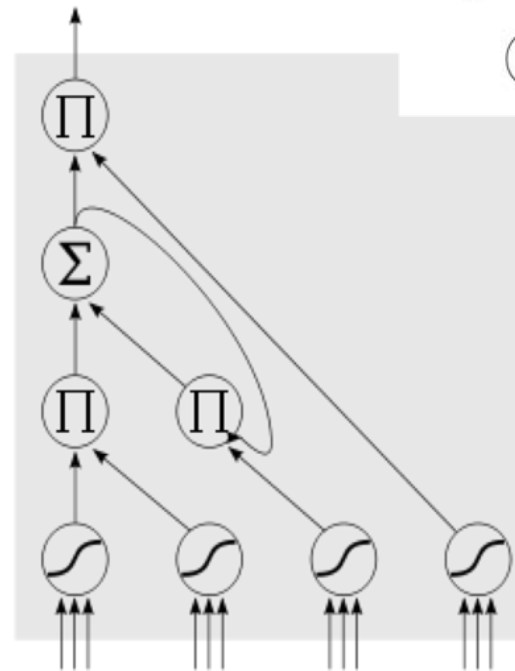
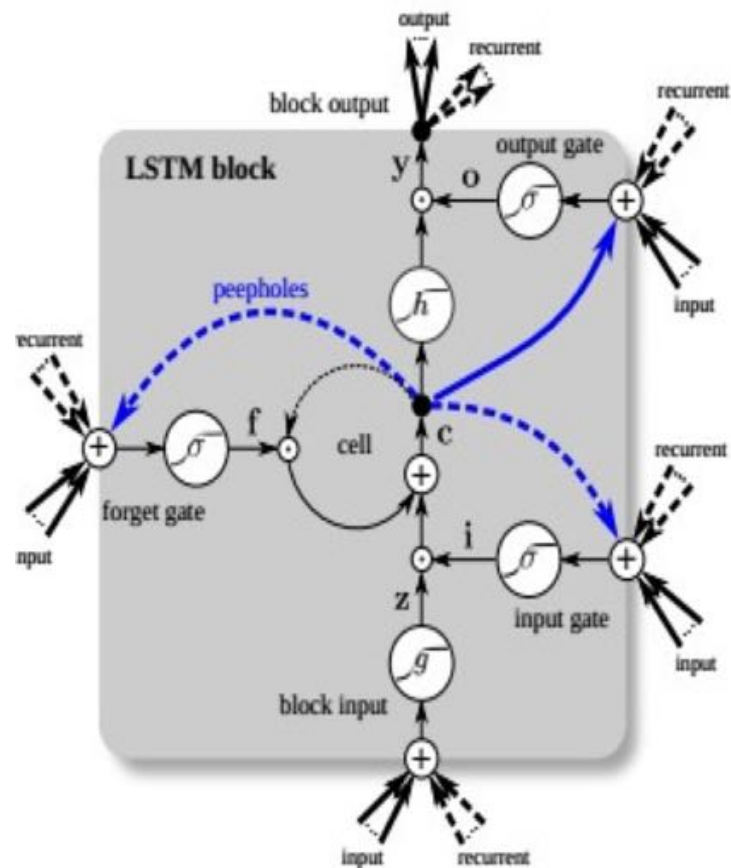
LSTM:

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

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



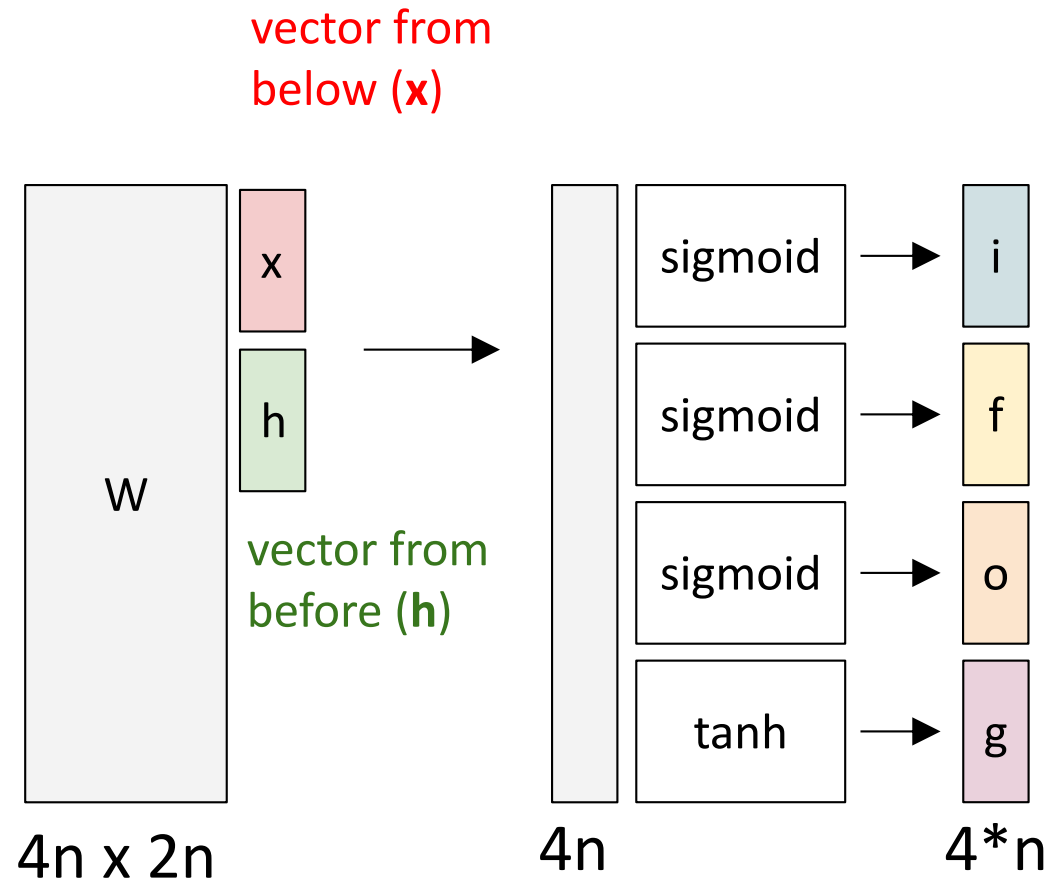
# LSTM: Long Short-Term Memory





# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



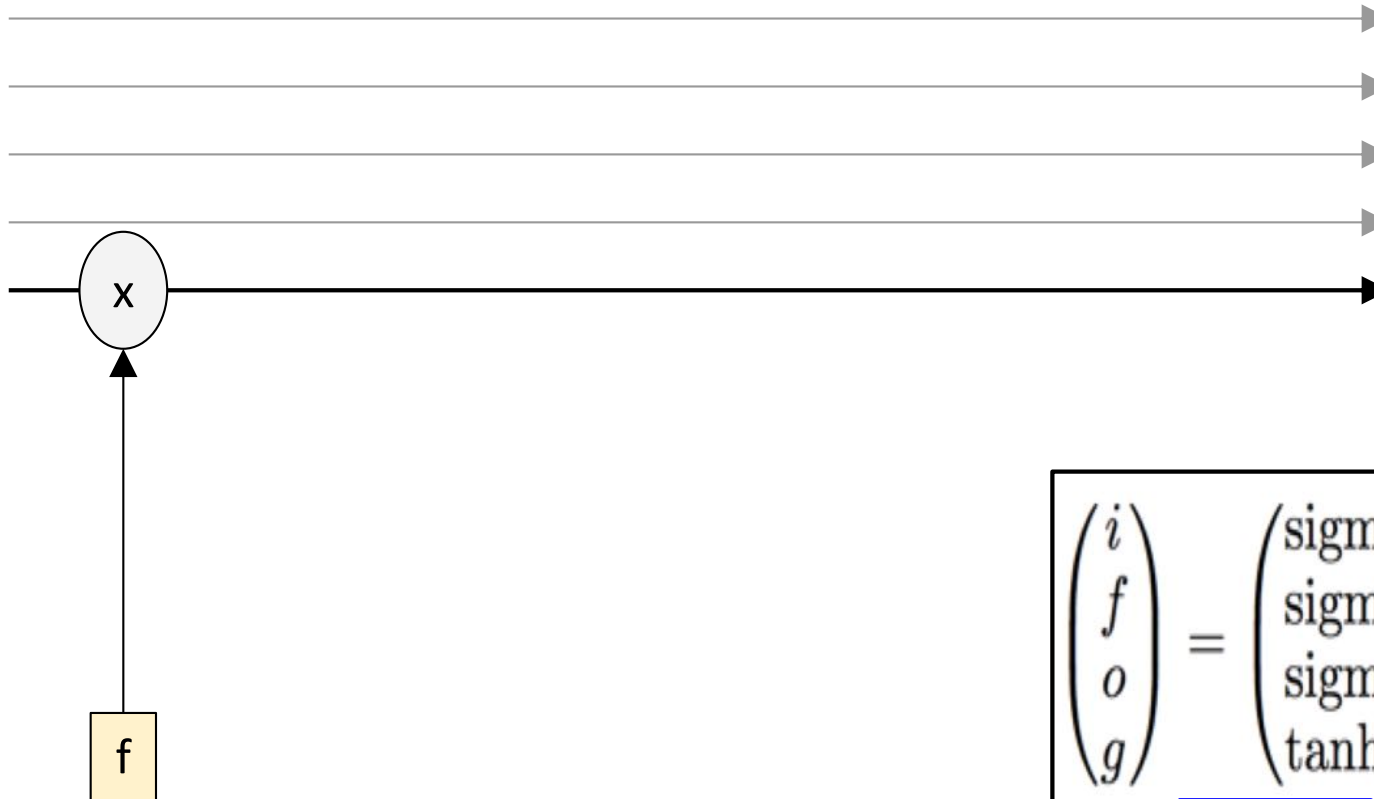
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$



# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell  
state  $c$



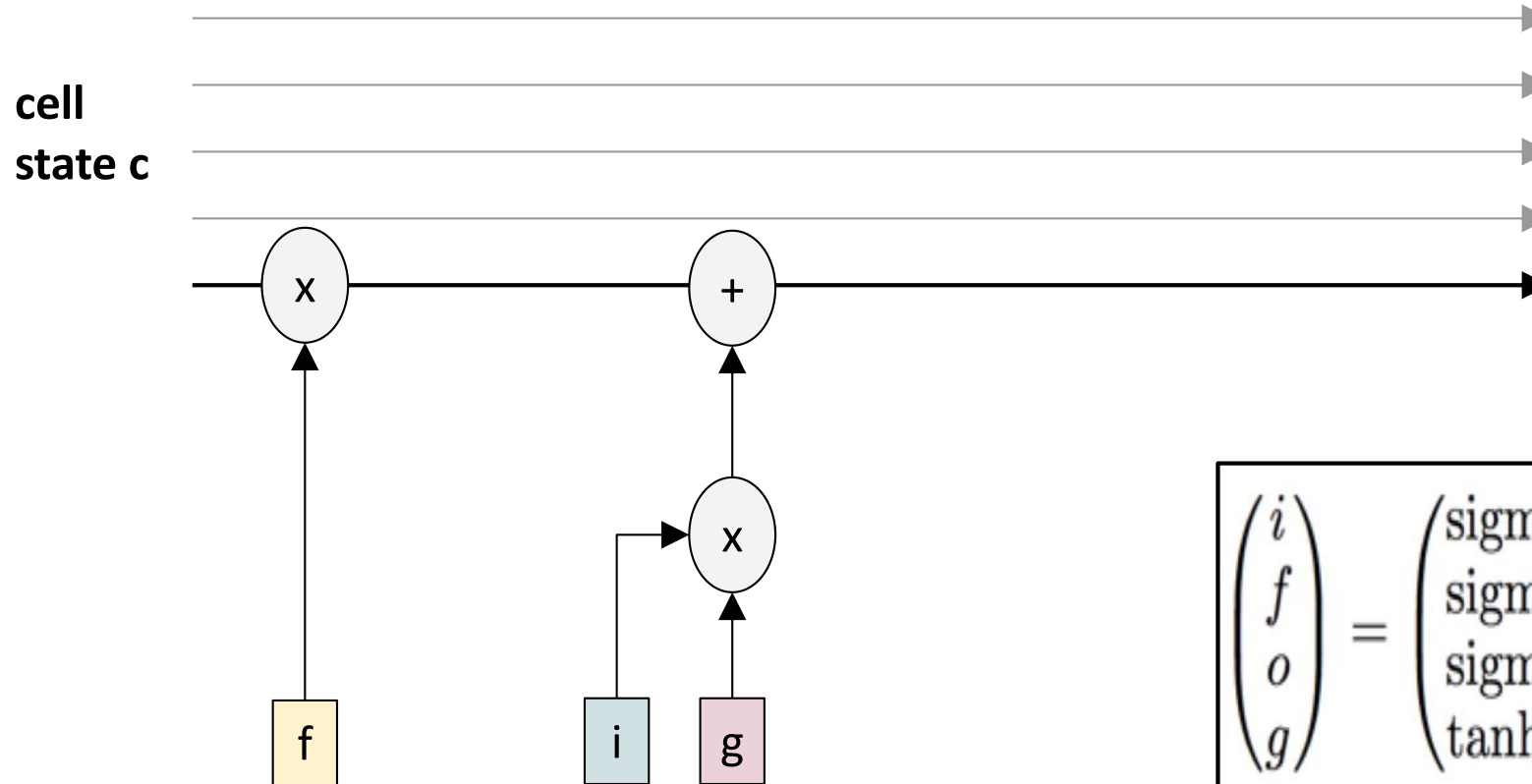
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$





# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

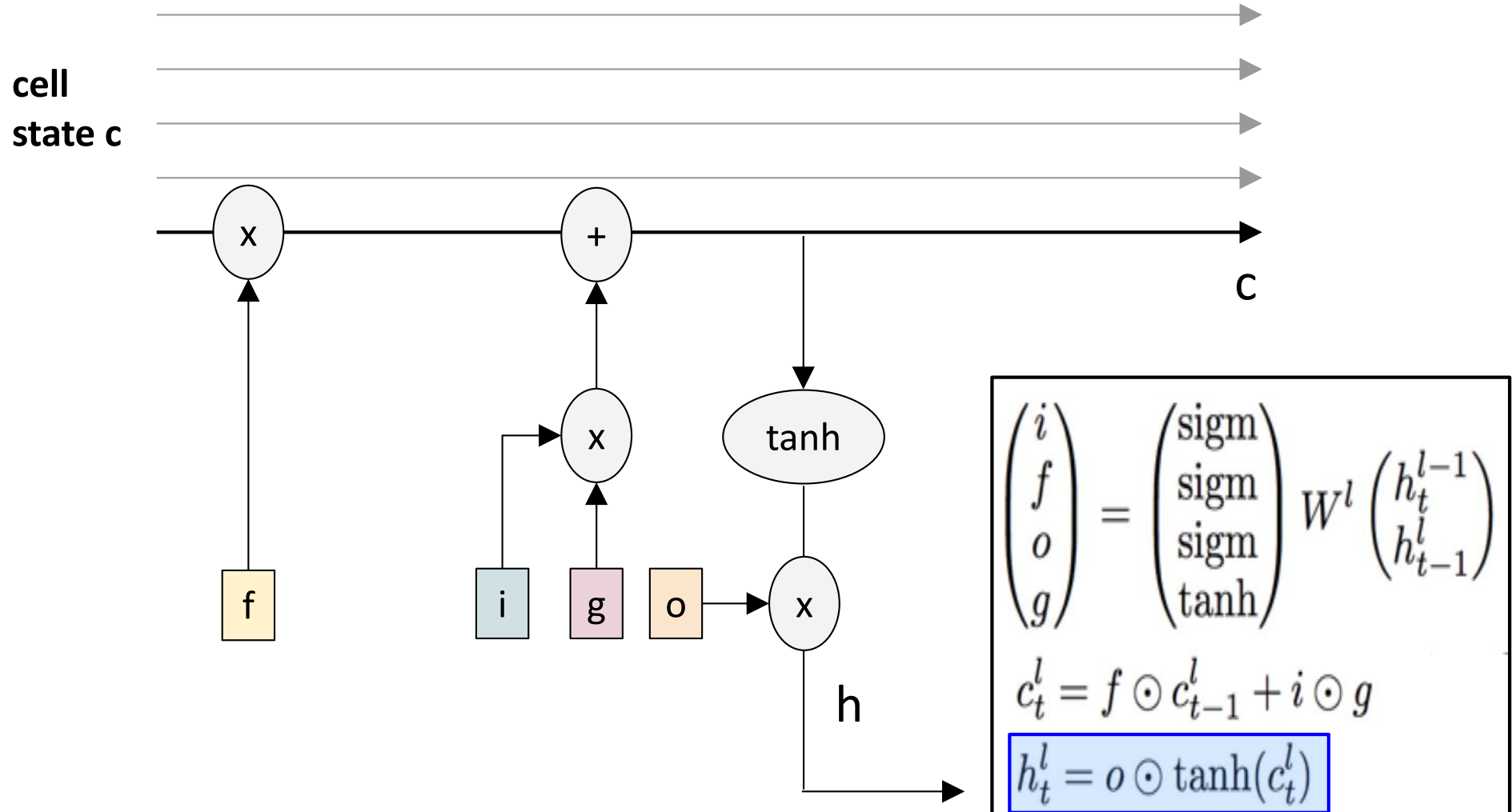


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$



# Long Short Term Memory (LSTM)

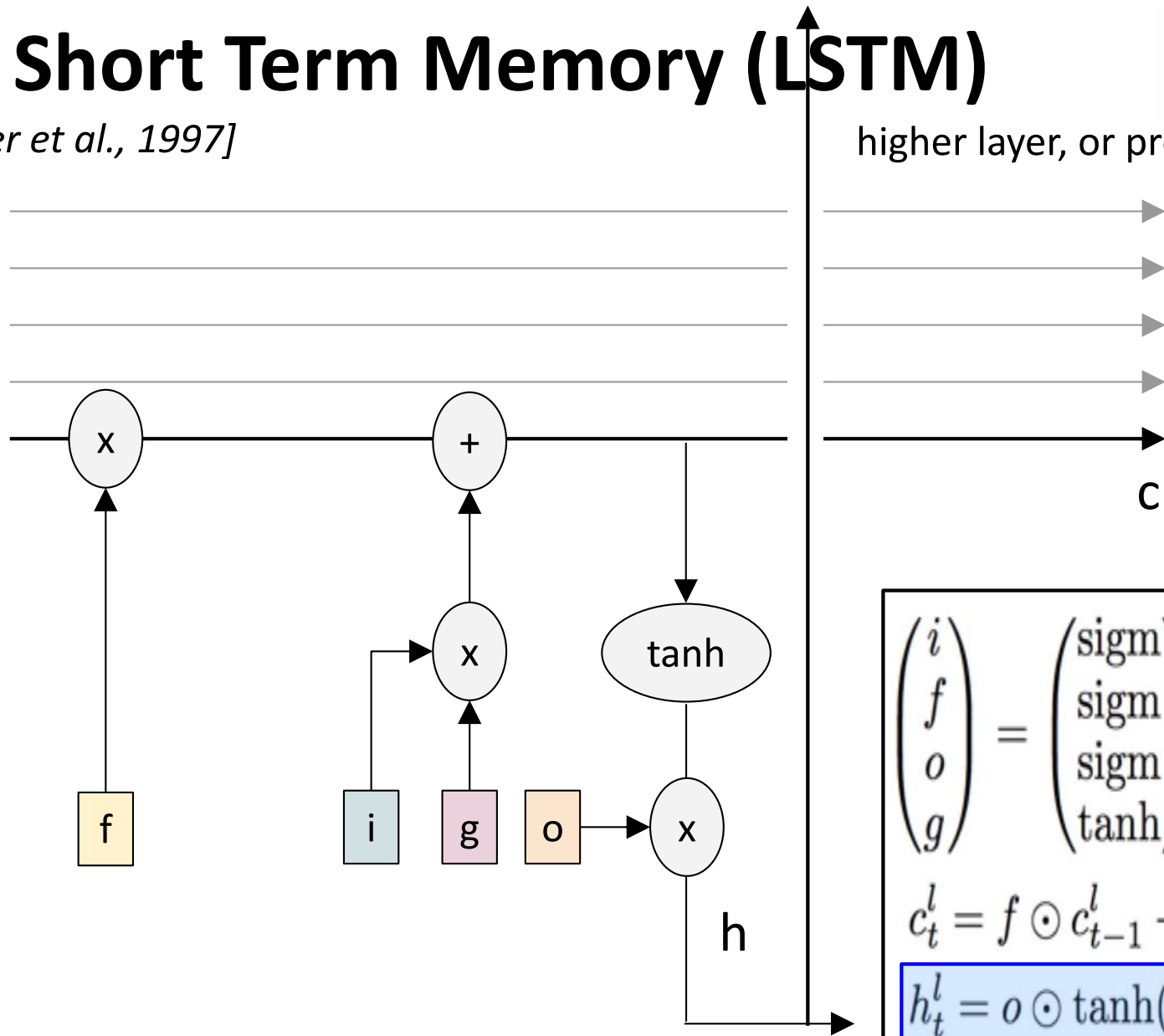
[Hochreiter et al., 1997]



# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

cell  
state  $c$

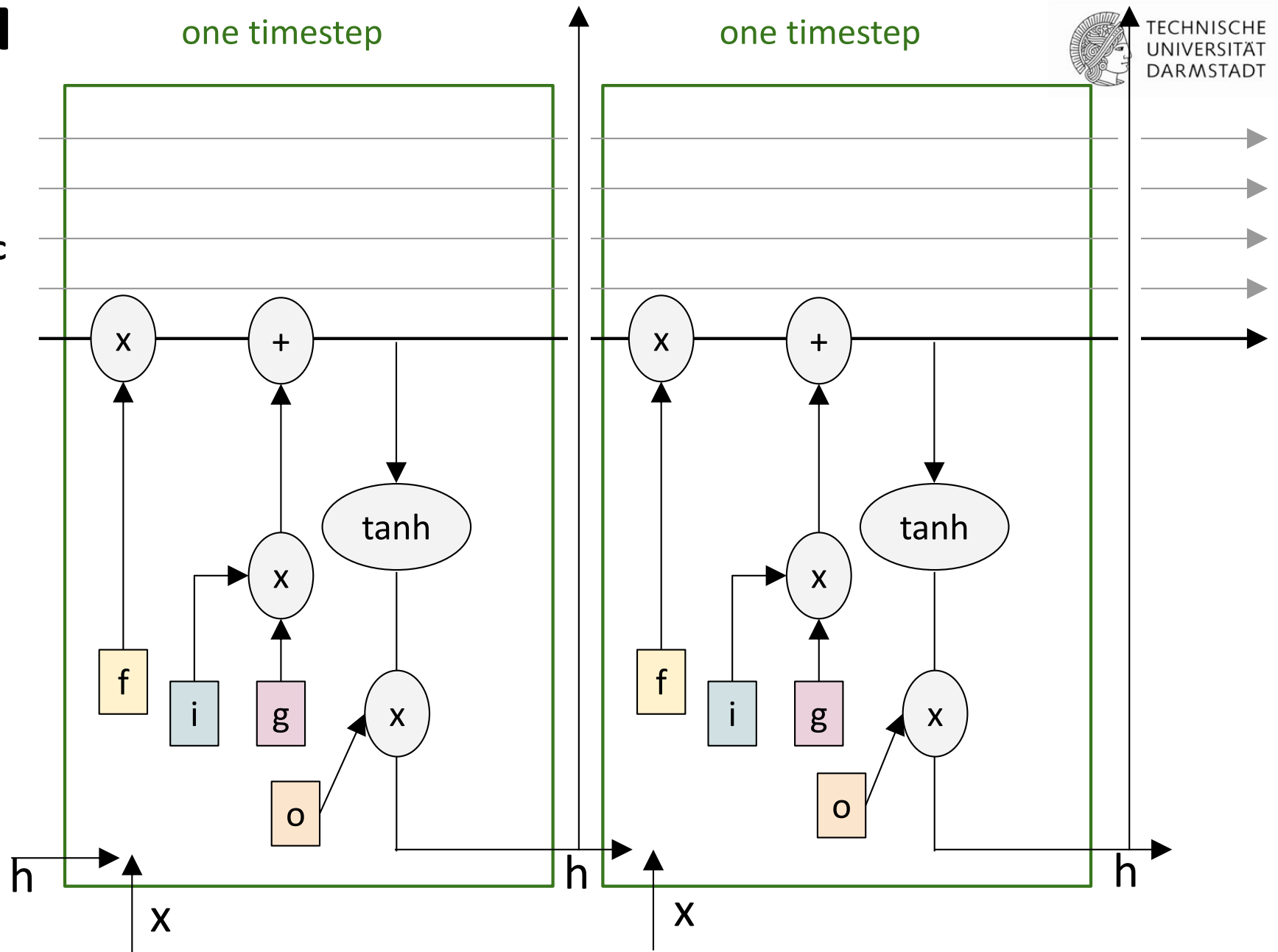


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$



# LSTM

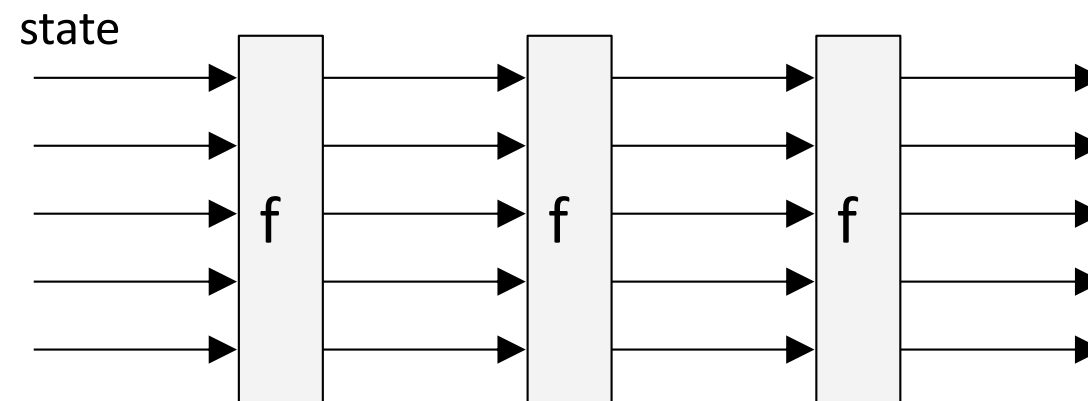
cell  
state  $c$



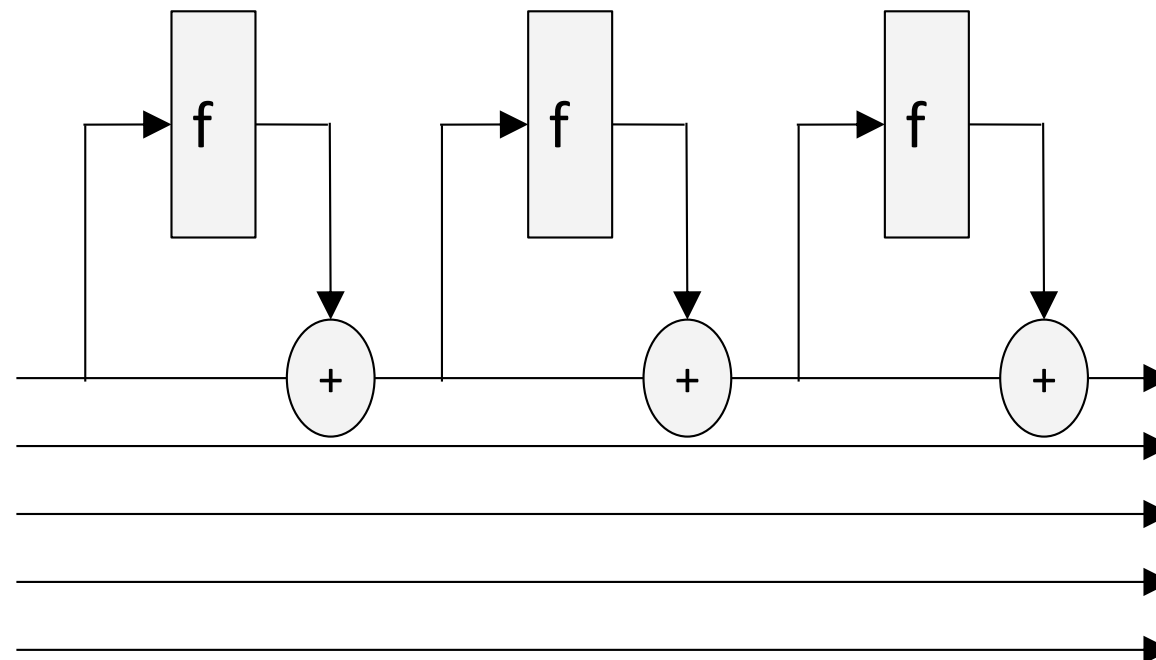
TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



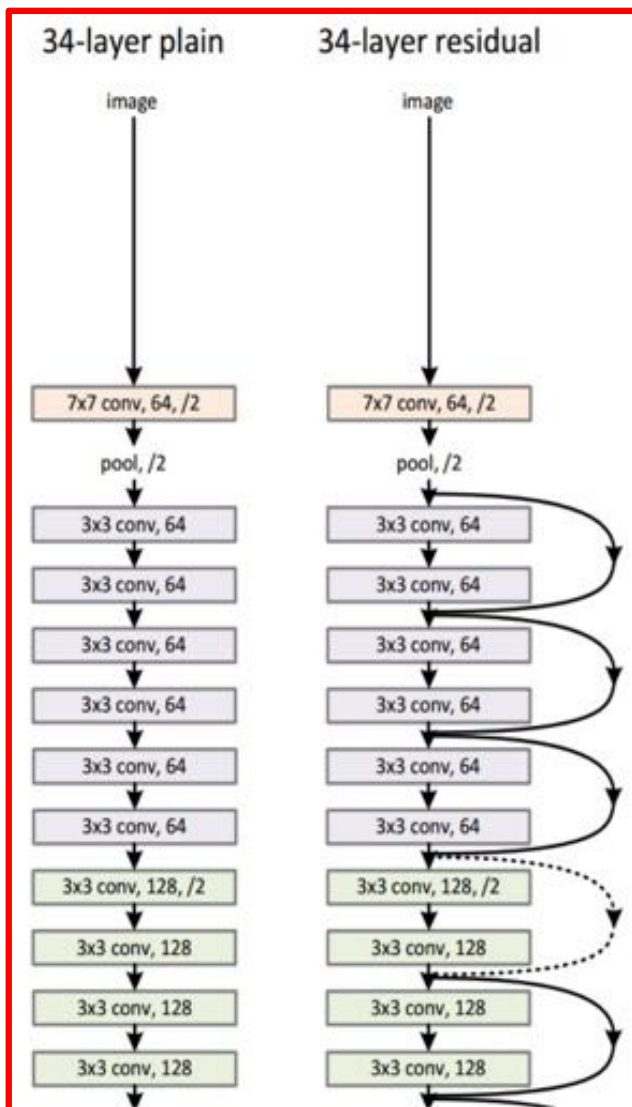
RNN



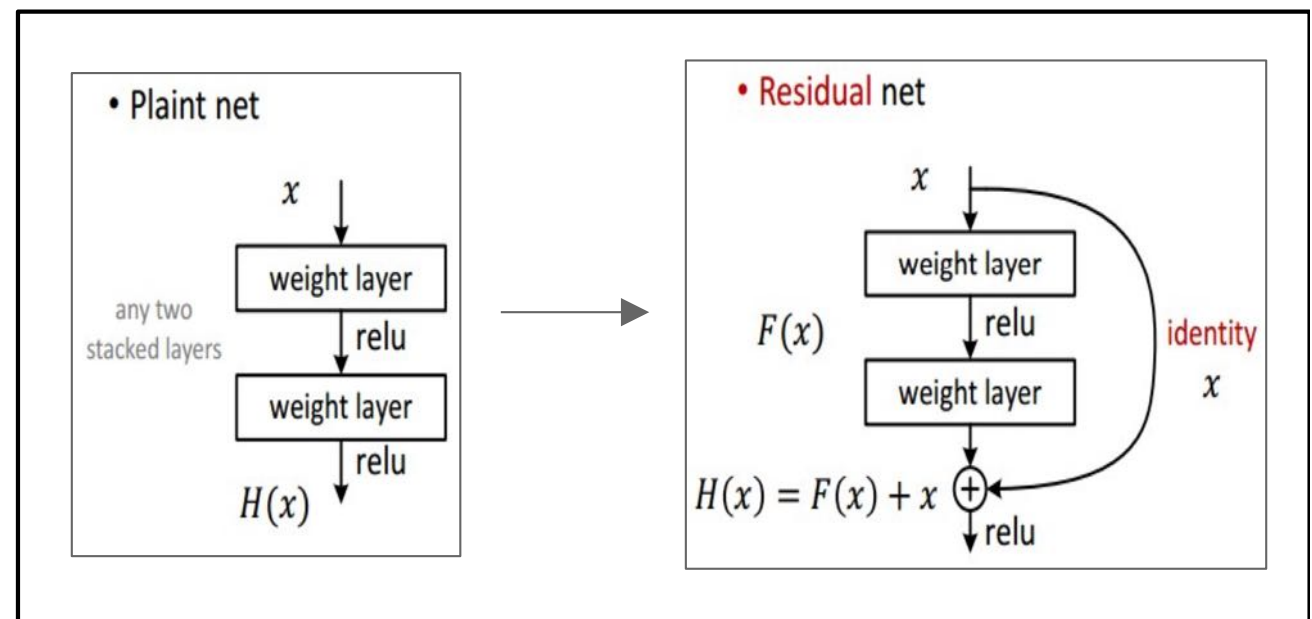
LSTM  
(ignoring  
forget gates)



# Recall: “PlainNets” vs. ResNets



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





# Understanding gradient flow dynamics



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

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

```
H = 5    # dimensionality of hidden state
T = 50    # number of time steps
W_hh = 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(W_hh, 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(W_hh.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)
```

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

```
# 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
```

[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)
```

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

```
# 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
```

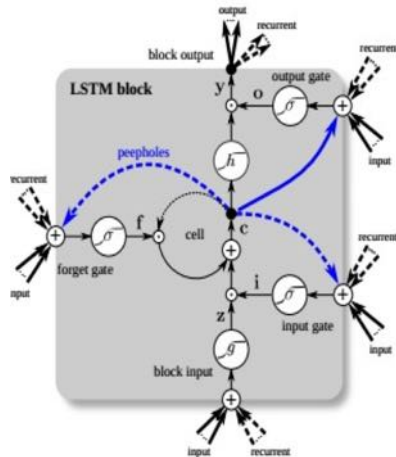
can control exploding with gradient clipping  
can control vanishing with LSTM

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



# LSTM variants and friends

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



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

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

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

MUT1:

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

MUT2:

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

MUT3:

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



# What have you learnt?

- 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.

