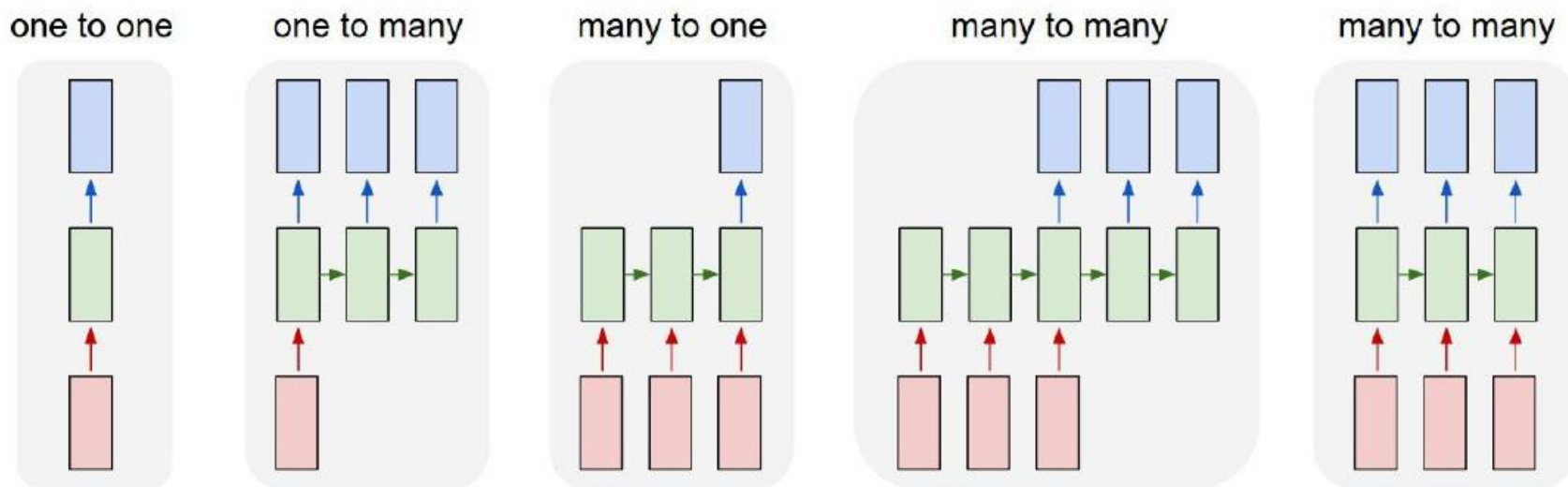


# RECURRENT NEURAL NETWORKS

---

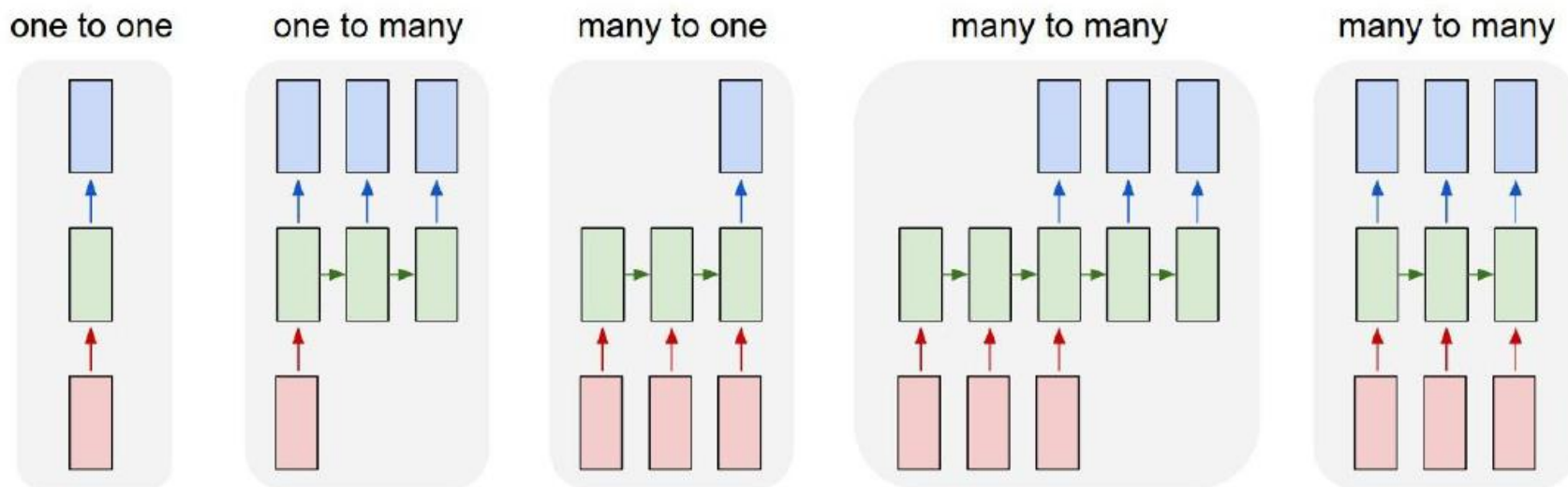
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# RNN offers a lot of flexibility



↖ Vanilla Neural Networks

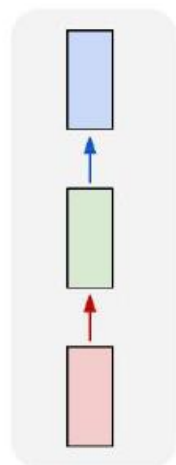
# RNN offers a lot of flexibility



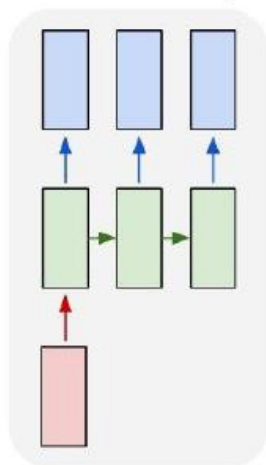
↖ e.g. **Image Captioning**  
image -> sequence of words

# RNN offers a lot of flexibility

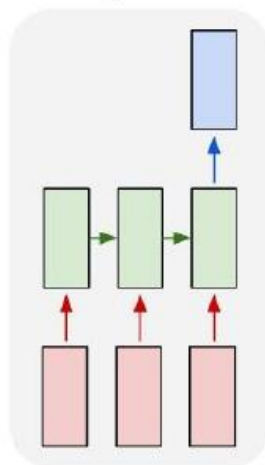
one to one



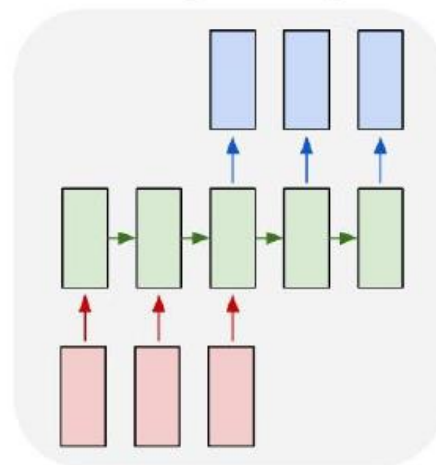
one to many



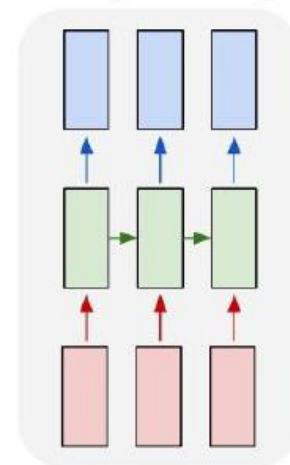
many to one



many to many



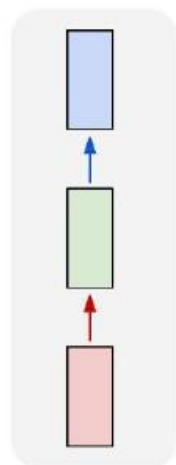
many to many



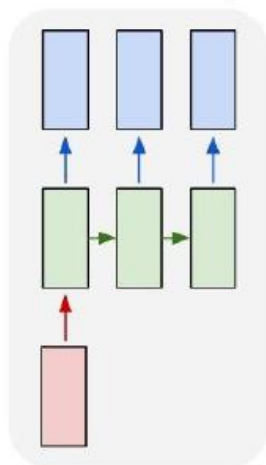
e.g. **Sentiment Classification**  
sequence of words -> sentiment

# RNN offers a lot of flexibility

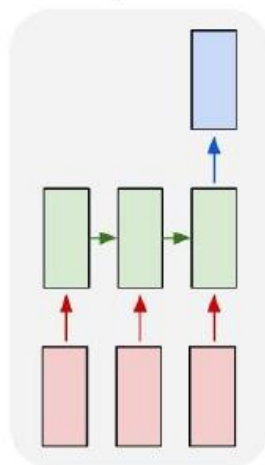
one to one



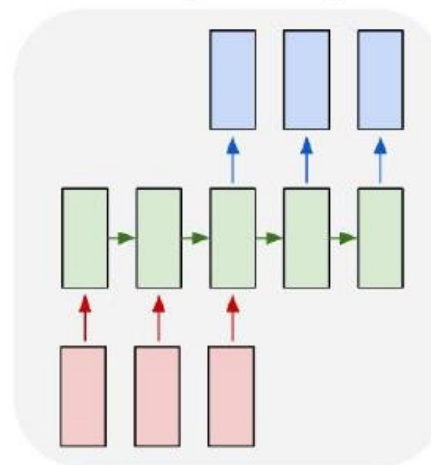
one to many



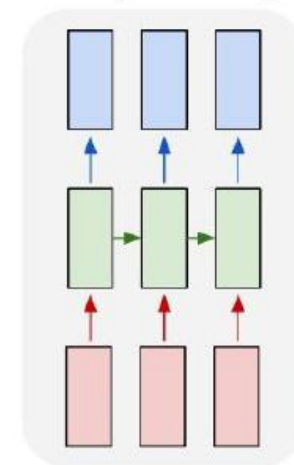
many to one



many to many



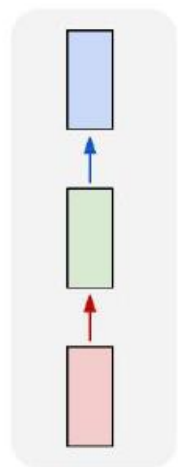
many to many



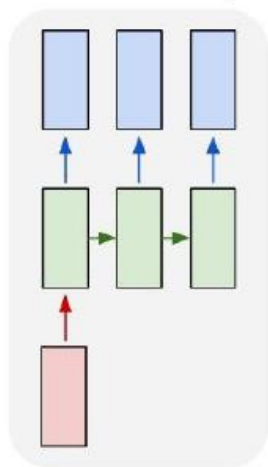
↖ e.g. **Machine Translation**  
seq of words -> seq of words

# RNN offers a lot of flexibility

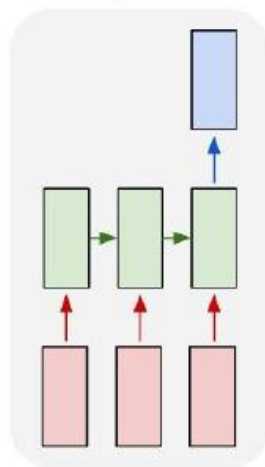
one to one



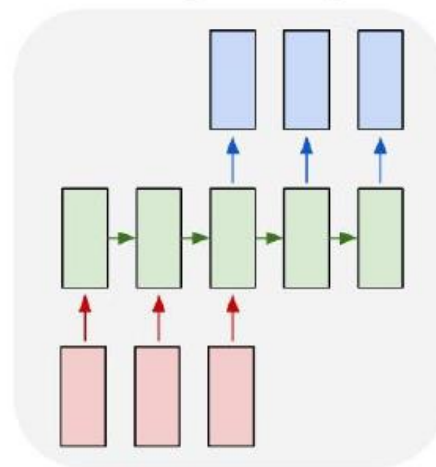
one to many



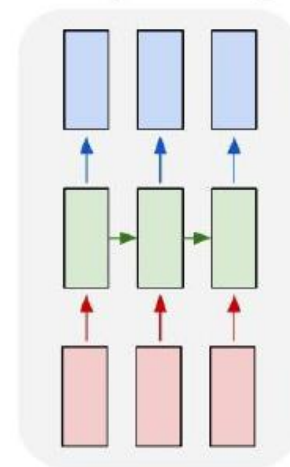
many to one



many to many



many to many



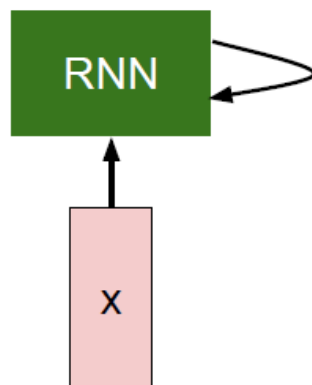
e.g. Video classification on frame level



# Sequential processing of fixed outputs

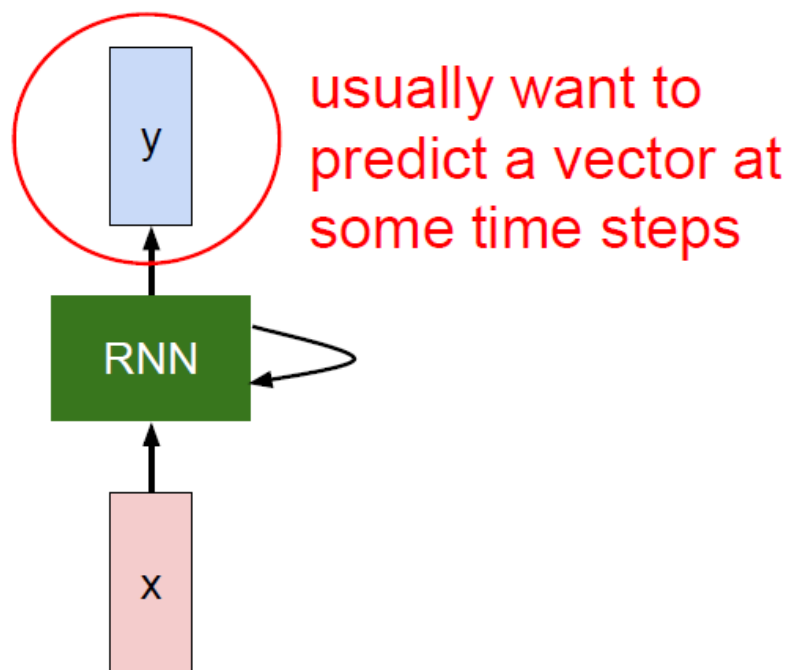
Reading MNIST

# 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:

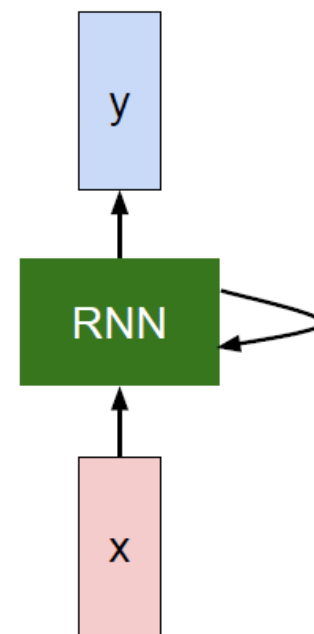
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters  $W$

old state

input vector at some time step

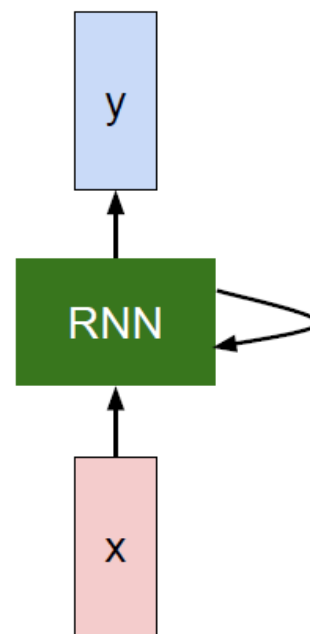


# Recurrent Neural Network

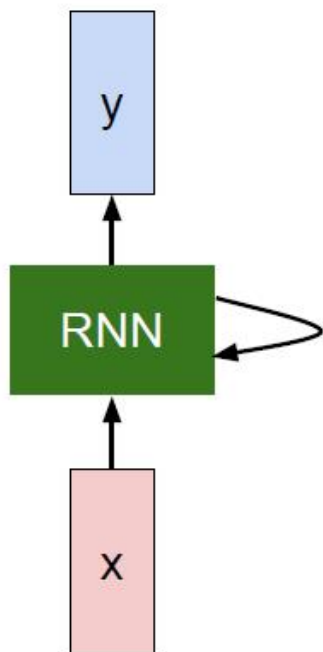
We can process a sequence of vectors  $\mathbf{x}$  by applying a recurrence formula at every time step:

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

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



# (Vanilla) Recurrent Neural Network



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

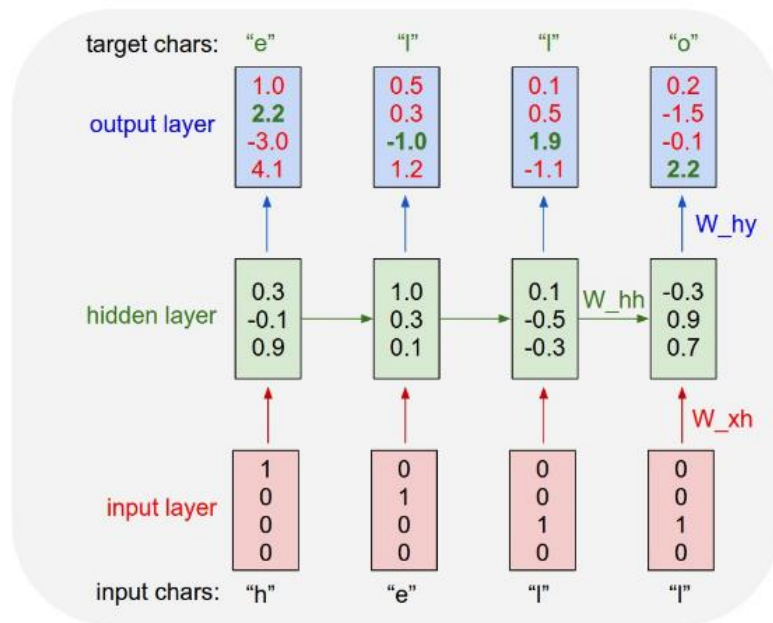


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

# Character-level language model

- Vocabulary: [h,e,l,o]
- Training sequence: hello



We want the green numbers to be high and red numbers to be low.

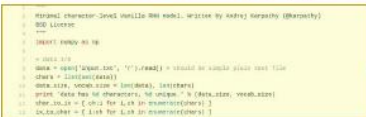
$$W_{hh} \in \mathcal{R}^{3 \times 3}$$

$$W_{xh} \in \mathcal{R}^{3 \times 4}$$

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

# Simple RNN code

## min-char-rnn.py gist



## Data I/O

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

# Simple RNN code

## min-char-rnn.py gist

```

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

## Initializations

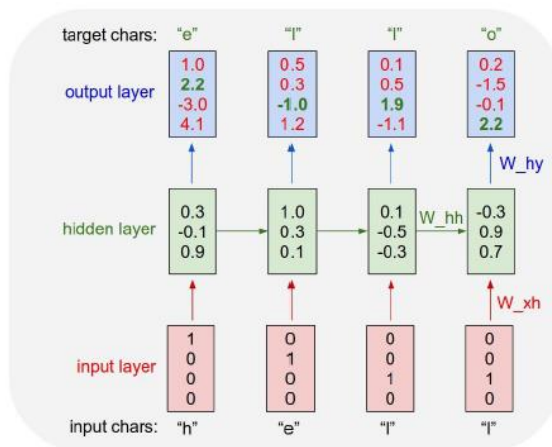
```

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:

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

$$y_t = W_{hy}h_t + b_y$$



# Simple RNN code

## min-char-rnn.py gist

```

'''
Minimal character-level vanilla RNN model. written by Andrej Karpathy (@karpathy)
SIG Lecture
'''
import numpy as np

# model size
vocab_size = 27 # characters
hidden_size = 100 # hidden layer size
num_layers = 1 # number of layers
seq_length = 20 # number of steps to unroll the RNN for
learning_rate = 1e-2

# small constants
W = np.random.randn(hidden_size, hidden_size) * 0.1 # layer to hidden
W_h = np.random.randn(hidden_size, hidden_size) * 0.1 # hidden to hidden
W_o = np.random.randn(hidden_size, vocab_size) * 0.1 # hidden to output
W_i = np.random.randn(hidden_size, 1) * 0.1 # bias to input
W_o = np.random.randn(hidden_size, 1) * 0.1 # bias to output

def softmax(logits, targets, pnorm):
    """
    logits, targets are both lists of integers.
    pnorm is log array of initial hidden state
    returns the loss, gradients on model parameters, and last hidden state
    """
    # forward pass
    for i in range(num_layers):
        # input to hidden state
        x = []
        for j in range(seq_length):
            # get character index
            x.append(targets[j])
        # hidden state
        h = pnorm
        for j in range(seq_length):
            # input to hidden state
            x.append(targets[j])
            # hidden state
            h = np.tanh(W_h * h + W_o * x + W_i)
        # output
        y = np.dot(W_o, h) + W_o
        # loss
        loss = -np.sum(np.log(1 + np.exp(y) - targets)) / seq_length
        # backprop
        d_logits = (y - targets) / seq_length
        d_pnorm = d_logits * W_o
        # update parameters
        d_W_h = (d_logits * W_o) * h
        d_W_o = (d_logits * W_o) * x
        # update parameters
        d_W_h += d_W_h * learning_rate
        d_W_o += d_W_o * learning_rate
        # update bias
        d_W_i += d_W_i * learning_rate
        d_W_o += d_W_o * learning_rate
    return loss, d_W_h, d_W_o, d_W_i, d_W_o, h

# train the model
pnorm = np.zeros(hidden_size)
for i in range(10000):
    loss, d_W_h, d_W_o, d_W_i, d_W_o, h = softmax(logits, targets, pnorm)
    # update parameters
    d_W_h += d_W_h * learning_rate
    d_W_o += d_W_o * learning_rate
    d_W_i += d_W_i * learning_rate
    d_W_o += d_W_o * learning_rate
    # update bias
    d_W_i += d_W_i * learning_rate
    d_W_o += d_W_o * learning_rate
    # update pnorm
    pnorm = d_pnorm + pnorm
    # print progress
    if i % 1000 == 0:
        print('iter %d, loss: %f' % (i, loss))
    # stop if loss is 0
    if loss == 0:
        break
    # update pnorm
    pnorm = d_pnorm + pnorm
    # update h
    h = d_pnorm + h

```

## Main loop

```

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

```





# Simple RNN code

## min-char-rnn.py gist

```
1 # MIT License
2 # Copyright (c) 2016 OpenAI (https://openai.com)
3 #
4 # Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated
5 # documentation files (the "Software"), to deal in the Software without restriction, including without limitation
6 # the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and
7 # to permit persons to whom the Software is furnished to do so, in any manner, and under any terms, provided that
8 # the copyright notice and this permission notice appear in all copies.
9 #
10 # THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
11 # TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
12 # SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
13 # ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
14 # OR OTHER DEALINGS IN THE SOFTWARE.
15
16 """
17 min-char-rnn.py
18 """
19
20 # Imports
21 import sys
22 import random
23 import numpy as np
24 import time
25
26 # Parameters
27 vocab_size = 27 # size of the vocabulary
28 hidden_size = 128 # size of the hidden state
29 seq_length = 100 # length of the sequence
30 num_epochs = 100 # number of epochs to train
31 num_batches = 100 # number of batches per epoch
32 learning_rate = 0.001 # learning rate
33
34 # Data
35 data = open('min-char-rnn.txt').read()
36 data = data[:vocab_size * seq_length]
37 data = data[:vocab_size * seq_length]
38 data = data[:vocab_size * seq_length]
39 data = data[:vocab_size * seq_length]
40 data = data[:vocab_size * seq_length]
41
42 # Parameters
43 Wxh = np.random.randn(hidden_size, vocab_size) * 0.1 # input to hidden
44 Whh = np.random.randn(hidden_size, hidden_size) * 0.1 # hidden to hidden
45 Why = np.random.randn(hidden_size, vocab_size) * 0.1 # hidden to output
46 Wb = np.random.randn(hidden_size, 1) * 0.1 # bias
47 Wc = np.random.randn(hidden_size, 1) * 0.1 # bias
48
49 # Parameters
50 def softmax(logits):
51     exp_logits = np.exp(logits)
52     return exp_logits / np.sum(exp_logits)
53
54 # Parameters
55 def lossFun(inputs, targets, hprev):
56     # Forward pass
57     h = hprev
58     loss = 0
59     for i in range(seq_length):
60         # Input
61         ix = inputs[i]
62         # Hidden state
63         h = np.tanh(Wxh[ix] + Whh[h] + Wb)
64         # Output
65         log_probs = softmax(Wwhy[h] + Wc)
66         loss += -log_probs[targets[i]]
67     return loss
68
69 # Parameters
70 def train():
71     # Parameters
72     Wxh, Whh, Why, Wb, Wc = [np.random.randn(hidden_size, vocab_size) * 0.1 for _ in range(5)]
73     # Parameters
74     hprev = np.zeros(hidden_size)
75     # Parameters
76     smooth_loss = -np.log(1.0 / vocab_size) * seq_length # loss at iteration 0
77     # Parameters
78     for epoch in range(num_epochs):
79         # Parameters
80         for batch in range(num_batches):
81             # Parameters
82             # Prepare inputs (we're sweeping from left to right in steps seq_length long)
83             p = 0 # go from start of data
84             inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
85             targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
86             # Parameters
87             # Sample from the model now and then
88             if n % 100 == 0:
89                 sample_ix = sample(hprev, inputs[0], 200)
90                 txt = ''.join(ix_to_char[ix] for ix in sample_ix)
91                 print '----\n %s \n----' % (txt, )
92             # Parameters
93             # Forward seq_length characters through the net and fetch gradient
94             loss, dxhx, dxhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
95             smooth_loss = smooth_loss * 0.999 + loss * 0.001
96             if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
97             # Parameters
98             # Perform parameter update with Adagrad
99             for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
100                                         [dxhx, dxhh, dwhy, dbh, dby],
101                                         [mWxh, mWhh, mWhy, mbh, mby]):
102                 mem += dparam * dparam
103                 param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
104             # Parameters
105             p += seq_length # move data pointer
106             n += 1 # iteration counter
107
108 # Parameters
109 if __name__ == '__main__':
110     train()
111
112 """
```

## Main loop

```
81 n, p = 0, 0
82 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 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, dxhx, dxhh, 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                                   [dxhx, dxhh, 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
```



# Simple RNN code

## min-char-rnn.py gist

```
1 # MIT License
2 # Copyright (c) 2016 OpenAI (https://openai.com)
3 #
4 # Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated
5 # documentation files (the "Software"), to deal in the Software without restriction, including without limitation
6 # the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and
7 # to permit persons to whom the Software is furnished to do so, in any manner, and under any terms, provided that
8 # the copyright notice and this permission notice appear in all copies.
9 #
10 # THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
11 # TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
12 # SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
13 # ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
14 # OR OTHER DEALINGS IN THE SOFTWARE.
15
16 """
17 min-char-rnn.py
18 """
19
20 # imports
21 import sys
22 import random
23 import numpy as np
24 import time
25
26 # constants
27 vocab_size = 27
28 hidden_size = 128
29 num_layers = 5
30 num_epochs = 10
31 seq_length = 20
32
33 # data
34 data = open('min-char-rnn.txt').read().strip()
35 data = data.lower()
36 data = data.replace(' ', '')
37 data = data.replace('.', '')
38 data = data.replace('-', '')
39 data = data.replace('_', '')
40 data = data.replace('!', '')
41 data = data.replace('?', '')
42 data = data.replace(';', '')
43 data = data.replace(':', '')
44 data = data.replace('"', '')
45 data = data.replace("'", '')
46 data = data.replace('(', '')
47 data = data.replace(')', '')
48 data = data.replace('[', '')
49 data = data.replace(']', '')
50 data = data.replace('{', '')
51 data = data.replace('}', '')
52 data = data.replace('~', '')
53 data = data.replace('&', '')
54 data = data.replace('%', '')
55 data = data.replace('^', '')
56 data = data.replace('*', '')
57 data = data.replace('&#39;', '')
58 data = data.replace('&#34;', '')
59 data = data.replace('&#40;', '')
60 data = data.replace('&#41;', '')
61 data = data.replace('&#5B;', '')
62 data = data.replace('&#5D;', '')
63 data = data.replace('&#7B;', '')
64 data = data.replace('&#7D;', '')
65 data = data.replace('&#126;', '')
66 data = data.replace('&#168;', '')
67 data = data.replace('&#169;', '')
68 data = data.replace('&#174;', '')
69 data = data.replace('&#177;', '')
70 data = data.replace('&#178;', '')
71 data = data.replace('&#179;', '')
72 data = data.replace('&#180;', '')
73 data = data.replace('&#181;', '')
74 data = data.replace('&#182;', '')
75 data = data.replace('&#183;', '')
76 data = data.replace('&#184;', '')
77 data = data.replace('&#185;', '')
78 data = data.replace('&#186;', '')
79 data = data.replace('&#187;', '')
80 data = data.replace('&#188;', '')
81 data = data.replace('&#189;', '')
82 data = data.replace('&#190;', '')
83 data = data.replace('&#191;', '')
84 data = data.replace('&#192;', '')
85 data = data.replace('&#193;', '')
86 data = data.replace('&#194;', '')
87 data = data.replace('&#195;', '')
88 data = data.replace('&#196;', '')
89 data = data.replace('&#197;', '')
90 data = data.replace('&#198;', '')
91 data = data.replace('&#199;', '')
92 data = data.replace('&#200;', '')
93 data = data.replace('&#201;', '')
94 data = data.replace('&#202;', '')
95 data = data.replace('&#203;', '')
96 data = data.replace('&#204;', '')
97 data = data.replace('&#205;', '')
98 data = data.replace('&#206;', '')
99 data = data.replace('&#207;', '')
100 data = data.replace('&#208;', '')
101 data = data.replace('&#209;', '')
102 data = data.replace('&#210;', '')
103 data = data.replace('&#211;', '')
104 data = data.replace('&#212;', '')
105 data = data.replace('&#213;', '')
106 data = data.replace('&#214;', '')
107 data = data.replace('&#215;', '')
108 data = data.replace('&#216;', '')
109 data = data.replace('&#217;', '')
110 data = data.replace('&#218;', '')
111 data = data.replace('&#219;', '')
112 data = data.replace('&#220;', '')
113 data = data.replace('&#221;', '')
114 data = data.replace('&#222;', '')
115 data = data.replace('&#223;', '')
116 data = data.replace('&#224;', '')
117 data = data.replace('&#225;', '')
118 data = data.replace('&#226;', '')
119 data = data.replace('&#227;', '')
120 data = data.replace('&#228;', '')
121 data = data.replace('&#229;', '')
122 data = data.replace('&#230;', '')
123 data = data.replace('&#231;', '')
124 data = data.replace('&#232;', '')
125 data = data.replace('&#233;', '')
126 data = data.replace('&#234;', '')
127 data = data.replace('&#235;', '')
128 data = data.replace('&#236;', '')
129 data = data.replace('&#237;', '')
130 data = data.replace('&#238;', '')
131 data = data.replace('&#239;', '')
132 data = data.replace('&#240;', '')
133 data = data.replace('&#241;', '')
134 data = data.replace('&#242;', '')
135 data = data.replace('&#243;', '')
136 data = data.replace('&#244;', '')
137 data = data.replace('&#245;', '')
138 data = data.replace('&#246;', '')
139 data = data.replace('&#247;', '')
140 data = data.replace('&#248;', '')
141 data = data.replace('&#249;', '')
142 data = data.replace('&#250;', '')
143 data = data.replace('&#251;', '')
144 data = data.replace('&#252;', '')
145 data = data.replace('&#253;', '')
146 data = data.replace('&#254;', '')
147 data = data.replace('&#255;', '')
148
149 # vocab
150 vocab = {}
151 for ch in data:
152     vocab[ch] = vocab.get(ch, 0) + 1
153
154 # char_to_ix
155 char_to_ix = {}
156 for i, ch in enumerate(sorted(vocab.keys())):
157     char_to_ix[ch] = i
158
159 # ix_to_char
160 ix_to_char = {}
161 for i, ch in enumerate(sorted(vocab.keys())):
162     ix_to_char[i] = ch
163
164 # data_loader
165 def data_loader():
166     """
167     Returns a list of (inputs, targets) pairs.
168     """
169     inputs = []
170     targets = []
171     for i in range(0, len(data) - seq_length):
172         inputs.append(data[i:i+seq_length])
173         targets.append(data[i+seq_length])
174     return inputs, targets
175
176 # model
177 def model(inputs, targets):
178     """
179     Returns the loss for a given batch.
180     """
181     # reset RNN memory
182     hprev = np.zeros((hidden_size, 1))
183     p = 0
184     # go from start of data
185     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
186     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
187
188     # sample from the model now and then
189     if n % 100 == 0:
190         sample_ix = sample(hprev, inputs[0], 200)
191         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
192         print '----\n %s \n----' % (txt, )
193
194     # forward seq_length characters through the net and fetch gradient
195     loss, dwxh, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
196     smooth_loss = smooth_loss * 0.999 + loss * 0.001
197     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
198
199     # perform parameter update with Adagrad
200     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
201                                   [dWxh, dWhh, dWhy, dbh, dby],
202                                   [mWxh, mWhh, mWhy, mbh, mby]):
203         mem += dparam * dparam
204         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
205
206     p += seq_length # move data pointer
207     n += 1 # iteration counter
```

## Main loop

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

# Simple RNN code

## min-char-rnn.py gist

```
"""
Minimal character-level vanilla RNN model, written by Andrej Karpathy (@karpathy)
BIG LITTLE
"""
import numpy as np

# DATA SIZE
vocab_size = 27 # characters, ' '? (27)
num_embeddings = vocab_size
embed_dim = 128 # embedding dimension
num_layers = 1 # number of layers of hidden state
hidden_dim = 128 # size of hidden layer of hidden state
num_params = 0 # number of parameters, used later for printing

# SMALL PARAMETERS
w = np.random.randn(hidden_dim, hidden_dim) * 0.1 # layer to hidden
wh = np.random.randn(hidden_dim, vocab_size) * 0.1 # hidden to hidden
wy = np.random.randn(hidden_dim, vocab_size) * 0.1 # hidden to output
b = np.zeros_like(w) # bias
bh = np.zeros_like(w) # bias
by = np.zeros_like(w) # bias

def lossFun(inputs, targets, hprev):
    """
    inputs, targets are both list of integers.
    hprev is list array of initial hidden state
    returns the loss, gradients on model parameters, and last hidden state
    """
    # loss, dx, dy, dhs, dhs, dhs, dhs
    # forward pass
    # backward pass
    # return loss, dparams, dhs[-1]
```



## Loss function

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

```
def lossFun(inputs, targets, hprev):
    """
    inputs, targets are both list of integers.
    hprev is Hx1 array of initial hidden state
    returns the loss, gradients on model parameters, and last hidden state
    """
    xs, hs, ys, ps = {}, {}, {}, {}
    hs[-1] = np.copy(hprev)
    loss = 0

    # forward pass
    for t in xrange(len(inputs)):
        xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
        xs[t][inputs[t]] = 1
        hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
        ys[t] = np.dot(Wyh, hs[t]) + by # unnormalized log probabilities for next chars
        ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
        loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)

    # backward pass: compute gradients going backwards
    dwhx, dwhh, dwhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Wyh)
    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])

    for t in reversed(xrange(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] -= 1 # backprop into y
        dwhy += np.dot(dy, hs[t].T)
        dby += dy
        dh = np.dot(Why.T, dy) + dhnext # backprop into h
        dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
        dbh += dhraw
        dwhx += np.dot(dhraw, xs[t].T)
        dwhh += np.dot(dhraw, hs[t-1].T)
        dhnext = np.dot(Whh.T, dhraw)

    for dparam in [dwhx, dwhh, dwhy, dbh, dby]:
        np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
    return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
```

# Simple RNN code

## min-char-rnn.py gist

```

1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # DATA I/O
8 data = open('train.txt', 'r').read() # should be simple since text file
9 chars = list(data)
10 data_size, vocab_size = len(data), len(chars)
11 print "data size %d characters, %d unique '%s' characters, %d unique '%s' characters"
12 vocab_size = (vocab_size + 1) # 0 is for blank space
13
14 # Hyperparameters
15 HIDDEN_SIZE = 100 # size of hidden layer of neurons
16 num_layers = 10 # number of steps to unroll the RNN for
17 learning_rate = 0.001
18
19 # Small constants
20 bh = np.random.randn(HIDDEN_SIZE, HIDDEN_SIZE) * 0.1 # bias to hidden
21 wh = np.random.randn(vocab_size, HIDDEN_SIZE) * 0.1 # bias to hidden
22 wy = np.random.randn(vocab_size, HIDDEN_SIZE) * 0.1 # bias to output
23 Wx = np.random.randn(HIDDEN_SIZE, 1) * 0.1 # bias bias
24 Wb = np.random.randn(HIDDEN_SIZE, 1) * 0.1 # bias bias
25
26 def lossFun(inputs, targets, hprev):
27 """
28 inputs, targets are both lists of integers
29 hprev is the array of initial hidden state
30 returns the loss, gradients on model parameters, and last hidden state
31 """
32
33 xs, hs, ys, ps = [], [], [], {}
34 hs[-1] = np.copy(hprev)
35 loss = 0
36 # Forward pass
37 for t in xrange(len(inputs)):
38 xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39 xs[t][inputs[t]] = 1
40 hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
41 ys[t] = np.dot(Wyh, 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 """
46 """
47 """
48 """
49 """
50 """
51 """
52 """
53 """
54 """
55 """
56 """
57 """
58 """
59 """
60 """
61 """
62 """
63 """
64 """
65 """
66 """
67 """
68 """
69 """
70 """
71 """
72 """
73 """
74 """
75 """
76 """
77 """
78 """
79 """
80 """
81 """
82 """
83 """
84 """
85 """
86 """
87 """
88 """
89 """
90 """
91 """
92 """
93 """
94 """
95 """
96 """
97 """
98 """
99 """
100 """
101 """
102 """
103 """
104 """
105 """
106 """
107 """
108 """
109 """
110 """
111 """
112 """
113 """
114 """
115 """
116 """
117 """
118 """
119 """
120 """
121 """
122 """
123 """
124 """
125 """
126 """
127 """
128 """
129 """
130 """
131 """
132 """
133 """
134 """
135 """
136 """
137 """
138 """
139 """
140 """
141 """
142 """
143 """
144 """
145 """
146 """
147 """
148 """
149 """
150 """
151 """
152 """
153 """
154 """
155 """
156 """
157 """
158 """
159 """
160 """
161 """
162 """
163 """
164 """
165 """
166 """
167 """
168 """
169 """
170 """
171 """
172 """
173 """
174 """
175 """
176 """
177 """
178 """
179 """
180 """
181 """
182 """
183 """
184 """
185 """
186 """
187 """
188 """
189 """
190 """
191 """
192 """
193 """
194 """
195 """
196 """
197 """
198 """
199 """
200 """
201 """
202 """
203 """
204 """
205 """
206 """
207 """
208 """
209 """
210 """
211 """
212 """
213 """
214 """
215 """
216 """
217 """
218 """
219 """
220 """
221 """
222 """
223 """
224 """
225 """
226 """
227 """
228 """
229 """
230 """
231 """
232 """
233 """
234 """
235 """
236 """
237 """
238 """
239 """
240 """
241 """
242 """
243 """
244 """
245 """
246 """
247 """
248 """
249 """
250 """
251 """
252 """
253 """
254 """
255 """
256 """
257 """
258 """
259 """
260 """
261 """
262 """
263 """
264 """
265 """
266 """
267 """
268 """
269 """
270 """
271 """
272 """
273 """
274 """
275 """
276 """
277 """
278 """
279 """
280 """
281 """
282 """
283 """
284 """
285 """
286 """
287 """
288 """
289 """
290 """
291 """
292 """
293 """
294 """
295 """
296 """
297 """
298 """
299 """
300 """
301 """
302 """
303 """
304 """
305 """
306 """
307 """
308 """
309 """
310 """
311 """
312 """
313 """
314 """
315 """
316 """
317 """
318 """
319 """
320 """
321 """
322 """
323 """
324 """
325 """
326 """
327 """
328 """
329 """
330 """
331 """
332 """
333 """
334 """
335 """
336 """
337 """
338 """
339 """
340 """
341 """
342 """
343 """
344 """
345 """
346 """
347 """
348 """
349 """
350 """
351 """
352 """
353 """
354 """
355 """
356 """
357 """
358 """
359 """
360 """
361 """
362 """
363 """
364 """
365 """
366 """
367 """
368 """
369 """
370 """
371 """
372 """
373 """
374 """
375 """
376 """
377 """
378 """
379 """
380 """
381 """
382 """
383 """
384 """
385 """
386 """
387 """
388 """
389 """
390 """
391 """
392 """
393 """
394 """
395 """
396 """
397 """
398 """
399 """
400 """
401 """
402 """
403 """
404 """
405 """
406 """
407 """
408 """
409 """
410 """
411 """
412 """
413 """
414 """
415 """
416 """
417 """
418 """
419 """
420 """
421 """
422 """
423 """
424 """
425 """
426 """
427 """
428 """
429 """
430 """
431 """
432 """
433 """
434 """
435 """
436 """
437 """
438 """
439 """
440 """
441 """
442 """
443 """
444 """
445 """
446 """
447 """
448 """
449 """
450 """
451 """
452 """
453 """
454 """
455 """
456 """
457 """
458 """
459 """
460 """
461 """
462 """
463 """
464 """
465 """
466 """
467 """
468 """
469 """
470 """
471 """
472 """
473 """
474 """
475 """
476 """
477 """
478 """
479 """
480 """
481 """
482 """
483 """
484 """
485 """
486 """
487 """
488 """
489 """
490 """
491 """
492 """
493 """
494 """
495 """
496 """
497 """
498 """
499 """
500 """
501 """
502 """
503 """
504 """
505 """
506 """
507 """
508 """
509 """
510 """
511 """
512 """
513 """
514 """
515 """
516 """
517 """
518 """
519 """
520 """
521 """
522 """
523 """
524 """
525 """
526 """
527 """
528 """
529 """
530 """
531 """
532 """
533 """
534 """
535 """
536 """
537 """
538 """
539 """
540 """
541 """
542 """
543 """
544 """
545 """
546 """
547 """
548 """
549 """
550 """
551 """
552 """
553 """
554 """
555 """
556 """
557 """
558 """
559 """
560 """
561 """
562 """
563 """
564 """
565 """
566 """
567 """
568 """
569 """
570 """
571 """
572 """
573 """
574 """
575 """
576 """
577 """
578 """
579 """
580 """
581 """
582 """
583 """
584 """
585 """
586 """
587 """
588 """
589 """
590 """
591 """
592 """
593 """
594 """
595 """
596 """
597 """
598 """
599 """
600 """
601 """
602 """
603 """
604 """
605 """
606 """
607 """
608 """
609 """
610 """
611 """
612 """
613 """
614 """
615 """
616 """
617 """
618 """
619 """
620 """
621 """
622 """
623 """
624 """
625 """
626 """
627 """
628 """
629 """
630 """
631 """
632 """
633 """
634 """
635 """
636 """
637 """
638 """
639 """
640 """
641 """
642 """
643 """
644 """
645 """
646 """
647 """
648 """
649 """
650 """
651 """
652 """
653 """
654 """
655 """
656 """
657 """
658 """
659 """
660 """
661 """
662 """
663 """
664 """
665 """
666 """
667 """
668 """
669 """
670 """
671 """
672 """
673 """
674 """
675 """
676 """
677 """
678 """
679 """
680 """
681 """
682 """
683 """
684 """
685 """
686 """
687 """
688 """
689 """
690 """
691 """
692 """
693 """
694 """
695 """
696 """
697 """
698 """
699 """
700 """
701 """
702 """
703 """
704 """
705 """
706 """
707 """
708 """
709 """
710 """
711 """
712 """
713 """
714 """
715 """
716 """
717 """
718 """
719 """
720 """
721 """
722 """
723 """
724 """
725 """
726 """
727 """
728 """
729 """
730 """
731 """
732 """
733 """
734 """
735 """
736 """
737 """
738 """
739 """
740 """
741 """
742 """
743 """
744 """
745 """
746 """
747 """
748 """
749 """
750 """
751 """
752 """
753 """
754 """
755 """
756 """
757 """
758 """
759 """
760 """
761 """
762 """
763 """
764 """
765 """
766 """
767 """
768 """
769 """
770 """
771 """
772 """
773 """
774 """
775 """
776 """
777 """
778 """
779 """
780 """
781 """
782 """
783 """
784 """
785 """
786 """
787 """
788 """
789 """
790 """
791 """
792 """
793 """
794 """
795 """
796 """
797 """
798 """
799 """
800 """
801 """
802 """
803 """
804 """
805 """
806 """
807 """
808 """
809 """
810 """
811 """
812 """
813 """
814 """
815 """
816 """
817 """
818 """
819 """
820 """
821 """
822 """
823 """
824 """
825 """
826 """
827 """
828 """
829 """
830 """
831 """
832 """
833 """
834 """
835 """
836 """
837 """
838 """
839 """
840 """
841 """
842 """
843 """
844 """
845 """
846 """
847 """
848 """
849 """
850 """
851 """
852 """
853 """
854 """
855 """
856 """
857 """
858 """
859 """
860 """
861 """
862 """
863 """
864 """
865 """
866 """
867 """
868 """
869 """
870 """
871 """
872 """
873 """
874 """
875 """
876 """
877 """
878 """
879 """
880 """
881 """
882 """
883 """
884 """
885 """
886 """
887 """
888 """
889 """
890 """
891 """
892 """
893 """
894 """
895 """
896 """
897 """
898 """
899 """
900 """
901 """
902 """
903 """
904 """
905 """
906 """
907 """
908 """
909 """
910 """
911 """
912 """
913 """
914 """
915 """
916 """
917 """
918 """
919 """
920 """
921 """
922 """
923 """
924 """
925 """
926 """
927 """
928 """
929 """
930 """
931 """
932 """
933 """
934 """
935 """
936 """
937 """
938 """
939 """
940 """
941 """
942 """
943 """
944 """
945 """
946 """
947 """
948 """
949 """
950 """
951 """
952 """
953 """
954 """
955 """
956 """
957 """
958 """
959 """
960 """
961 """
962 """
963 """
964 """
965 """
966 """
967 """
968 """
969 """
970 """
971 """
972 """
973 """
974 """
975 """
976 """
977 """
978 """
979 """
980 """
981 """
982 """
983 """
984 """
985 """
986 """
987 """
988 """
989 """
990 """
991 """
992 """
993 """
994 """
995 """
996 """
997 """
998 """
999 """
1000 """

```

```

27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = [], [], [], {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # Forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(Wyh, 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)

```



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

$$y_t = W_{hy}h_t + b_y$$

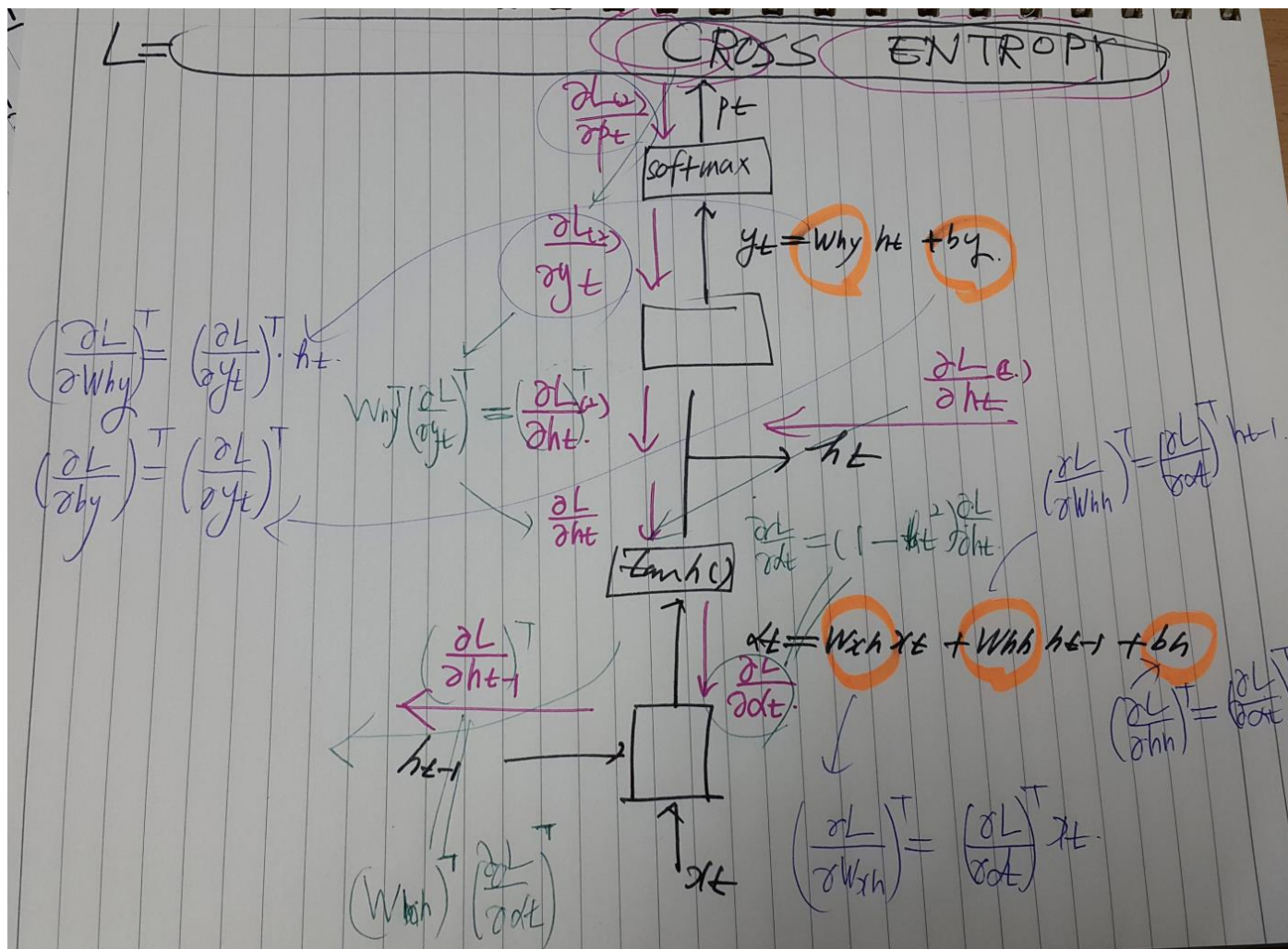
$$p_t = \text{softmax}(y_t)$$

Softmax classifier

$$l \leftarrow l - \log(p_t)^T [0 \dots 1 \dots 0]$$



# RNN math

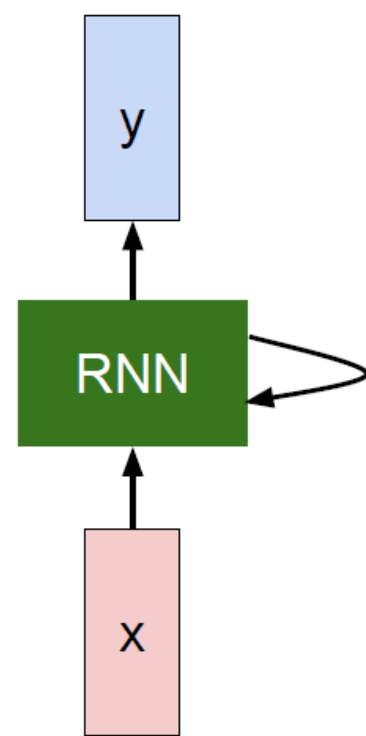






# Training

epson.com/usa/en-us/Products/Laser/Printers The fundamental everyday requirements for mono and colour laser printing throughout today's office is perfectly met with the extensive Epson laser printer range. The latest AcuLaser printer range offers users exceptionally fast and affordable colour laser printing for the desktop. The radically 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 C1800. Epson brings both colour and near-drumless laser printing together at a back-and-forth price. more Where to Buy Support Epson AcuLaser C5000 The fastest colour laser printer in its class. The perfect printer for small-business and work groups, the Epson AcuLaser C5000 prints high resolution in black and white and vibrant colour, at high speed and with low running costs. more Where to Buy High quality resolution, 2400dpi/PT. Large paper capacity, 600 sheets, expandable up to 1,000 sheets. Compatible Windows and Mac. High speed USB and EpsonNet 10/100 Base-TX Ethernet interfaces standard. Epson AcuLaser Resolution Improvement Technology. EpsonNet 10/100 Base-TX Ethernet. Compatible with Epson AcuLaser C3000's resolution, AcuLaser C5000 48MB memory, 100 sheet UP Tray, 500 sheet cassette, Duplex printing as standard. AcuLaser C3000i 64MB memory, 110 sheet UP Tray, 500 sheet cassette, Duplex printing, 10/100 Base-TX Ethernet interface. Networked compact colour laser printer for professional environments. Businesses have been seeking 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 C1910. Epson brings both colour and near-drumless laser printing together at a back-and-forth price. Key features cost-effective mono printing for day-to-day business needs and vivid, vibrant colour when required. Search Epson UK Epson AcuLaser C500 Outstanding professional colour printing for business Add colour to your business with the Epson AcuLaser C500 from Epson. Its perfect for the smaller workgroup, being a compact and cost-effective laser printing workgroup solution offering colour output as well as high performance black and white production. more Where to Buy Support As cost-effective to run as a mono-laser printer. Paper capacity of 1700 sheets from two media sources. Easy to operate non-advanced printer driver. Memory expandable from 32MB to 1024MB. 3-in-1-configuration models available with Wireless iPrint, iPrint, iPrint-Server & Level™ and two-sided printing. The AcuLaser C1800 is available in 5 configurations. AcuLaser C1500S with 32MB, 333 Sheet UP Tray, 10/100 Base-TX Networking. AcuLaser C1500 with 32MB, 500 Sheet UP Tray, 500 Sheet Cassette, 10/100 Base-TX Networking. Support Epson AcuLaser C1450 High performance colour 880dpi for all your business printing needs. The Epson AcuLaser C1400 provides businesses with a high performance colour and mono print solution. It adds crucial colour to your business, while producing high quality monochrome output at lower costs than many monochrome-only printers, and it's just as easy to operate. So why waste no time in buying this printer, because perfect monochrome and colour solutions are available in one. more Where to Buy Support Epson AcuLaser C6000 Professional high performance A3 colour laser printer. Epson AcuLaser C6000 is the perfect professional printing solution for users who require exceptionally high quality and mono output on a range of media formats from C5 up to A7W in size. The Epson AcuLaser C6000 is able to achieve excellent print quality by adding a combination of Epson's exclusive AcuLaser Colour Laser Technologies, more Where to Buy Support AcuLaser C1600F with Adobe® PostScript™ 3™, 96MB, 200 Sheet UP Tray, 500 Sheet Cassette, 10/100 Base-TX Networking. AcuLaser C1600 with Duplex and two-sided printing, 96MB, 200 Sheet UP Tray, 500 Sheet Cassette, 10/100 Base-TX Networking. AcuLaser C1600 with 128MB, 200 Sheet UP Tray, 500 Sheet Cassette, Wireless Networking. Add colour to your business with the Epson AcuLaser C803 from Epson. Its perfect for the smaller workgroup, being a compact and cost-effective laser printing workgroup solution that offers amazing colour output, as well as high performance black and white production. more Where to Buy Support Epson AcuLaser C1800 High performance colour laser printer. Epson AcuLaser C1800 provides businesses with high performance colour and monochrome printing solutions. more Where to Buy Support Epson AcuLaser C3100 High speed A3 colour laser printer. Why have separate black and white and colour printers when you can have the Epson AcuLaser C3100? Epson has taken the lead in laser technology to deliver a complete high-performance solution for all your colour and mono printing needs. Support Epson AcuLaser C4400 High performance A4 mono laser professional printer. The Epson EPL-4200 and EPL-6200L are the outstanding solutions for small to medium workgroups and personal users. They deliver professional performance, quality, clarity, reliability and cost-effectiveness, and are perfect for users who need high levels of laser quality and productivity at a low investment. more Where to Buy Support Epson AcuLaser C4200 High performance A4 mono laser professional printer. The Epson EPL-4200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance, quality, clarity, reliability and cost-effectiveness, 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 low line, you can now bring the power of high quality colour to your documents without affecting the high costs or low speeds traditionally associated with colour.



# Results

at first:

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

↓  
train more

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

↓  
train more

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

↓  
train more

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

# 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 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff) & 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 ");
}
```

# 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

**WHAT HAVE THE RNN  
LEARNED?**

---



# Probability Model

- Given example sequences  $\{X_i\}$ , let us find a probabilistic model  $p(X)$  that maximizes  $\prod_i p(X_i)$ .
- $X_i$  is a sequence  $X_i = (x_1, x_2, \dots, x_T)$
- For example,
  - $x_1 = h, x_2 = e, x_3 = l, x_4 = l, x_5 = o$ .
  - $x_1 = "I", x_2 = "am", x_3 = "a", x_4 = "student"$ .
- For the model learning, we have to specify
  - Parametric family (model selection)
  - Learning criterion
  - Parameter learning method

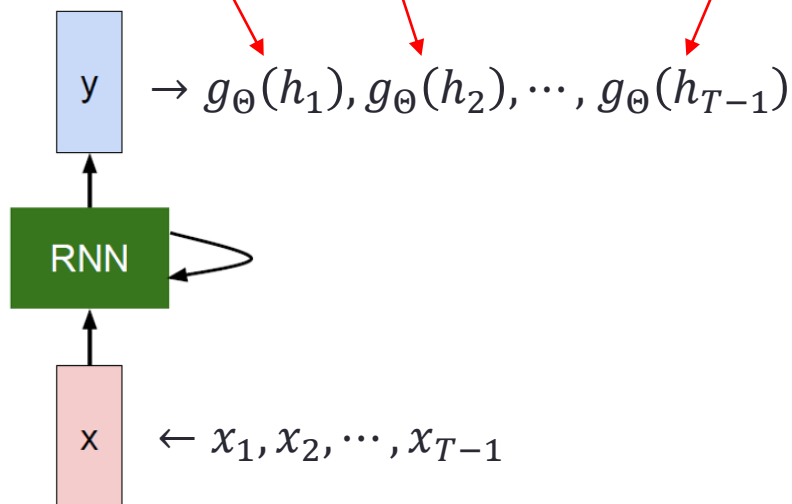
# Our choice

- Parametric family: RNN

- Because we can decompose  $p(X) = p(x_1, x_2, \dots, x_T)$  into

$$p(x_1, x_2, \dots, x_T) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \cdots p(x_T|x_1, \dots, x_{T-1})$$

- and RNN can handle this situation well.



# Our choice

- Parametric family: RNN
  - $g_{\Theta}(h_{t-1})$  is a parametric model of  $p(x_t | x_1, x_2, \dots, x_{t-1})$ .
  - If we have an  $N$ -word dictionary, we can represent  $p(x_t | x_1, x_2, \dots, x_{t-1})$  with a  $N$ -dimensional vector (that sums to a unity and non-negative).

$$g_{\Theta}(h_{t-1}) = \begin{array}{|c|} \hline p(x_t = 1^{\text{st}} \text{ word in the dictionary} | x_1, x_2, \dots, x_{t-1}) \\ \hline p(x_t = 2^{\text{nd}} \text{ word in the dictionary} | x_1, x_2, \dots, x_{t-1}) \\ \hline p(x_t = 3^{\text{rd}} \text{ word in the dictionary} | x_1, x_2, \dots, x_{t-1}) \\ \hline p(x_t = 4^{\text{th}} \text{ word in the dictionary} | x_1, x_2, \dots, x_{t-1}) \\ \hline \dots \\ \hline \dots \\ \hline \dots \\ \hline p(x_t = N^{\text{th}} \text{ word in the dictionary} | x_1, x_2, \dots, x_{t-1}) \\ \hline \end{array}$$

# Our choice

- Learning criterion: log likelihood/cross entropy

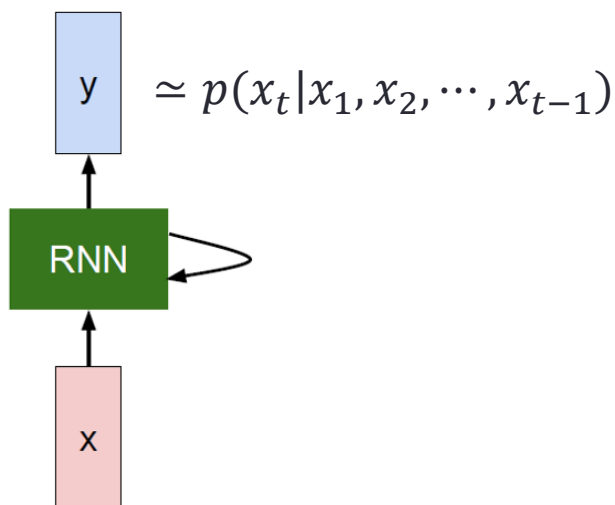
- $\log p(x_1, x_2, \dots, x_T) = \sum_t \log p(x_t | x_1, x_2, \dots, x_{t-1})$

- $\sum_t \log p(x_t | x_1, x_2, \dots, x_{t-1}) \rightarrow \sum_t g_{\Theta}(h_{t-1})^T \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow \text{cross entropy}$   
 $x_t$ 's element is 1.

- Learning method: SGD

# Summary

- The above RNN model learns the (conditional) probability model of a sequence by maximizing the log likelihood of training sequences.



# APPLICATION: IMAGE CAPTIONING

---

# Image captioning

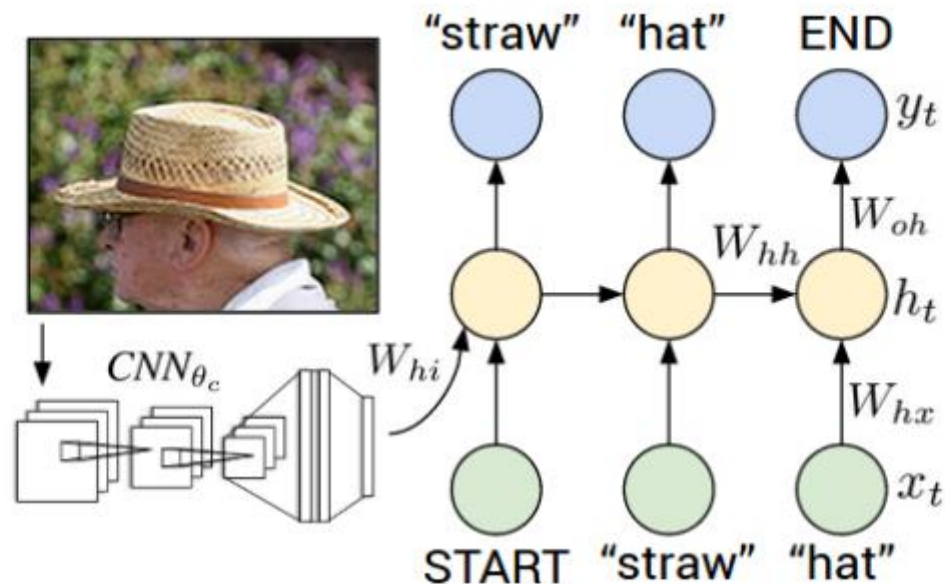
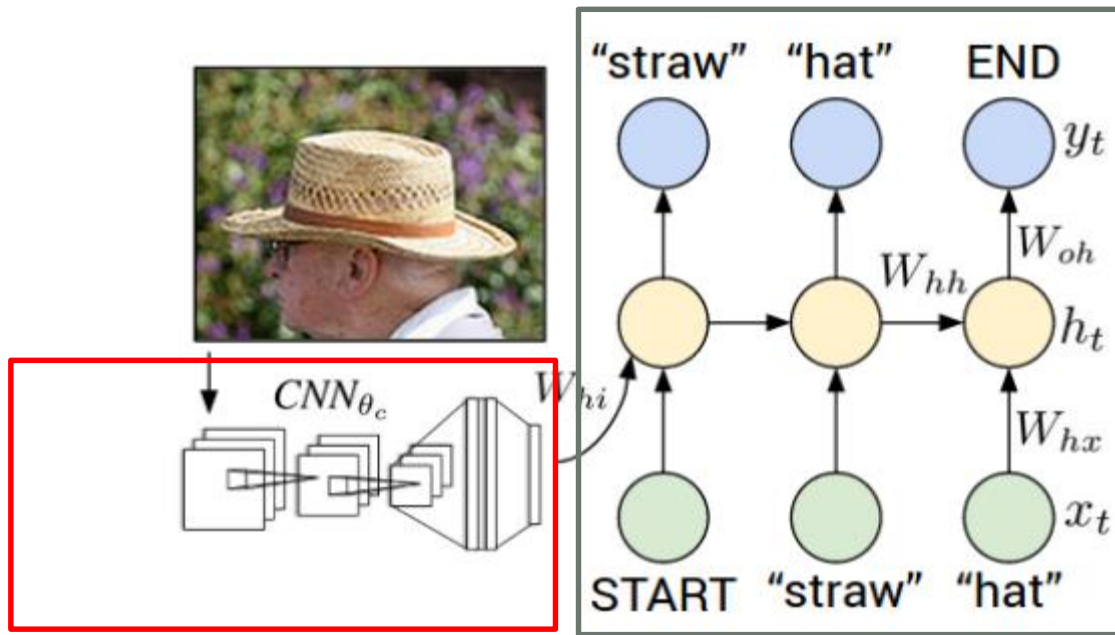


Diagram of our multimodal Recurrent Neural Network generative model. The RNN takes a word, the context from previous time steps and defines a distribution over the next word in the sentence. The RNN is conditioned on the image information at the first time step. START and END are special tokens.



CNN

RNN





test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

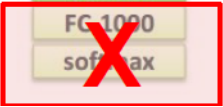
FC-4096

FC-1000

softmax



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

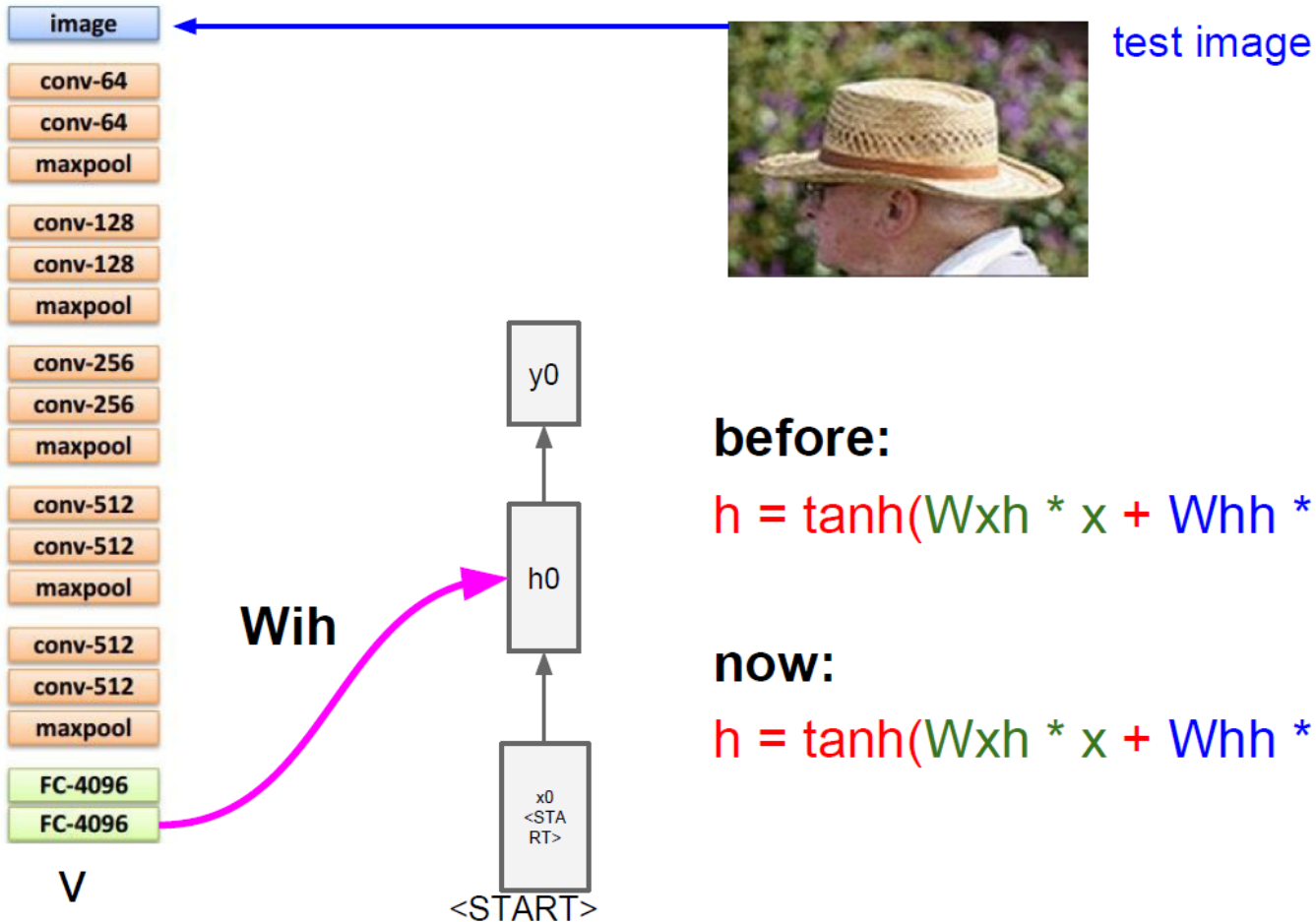
FC-4096



test image

x0  
<START>  
RT>

<START>

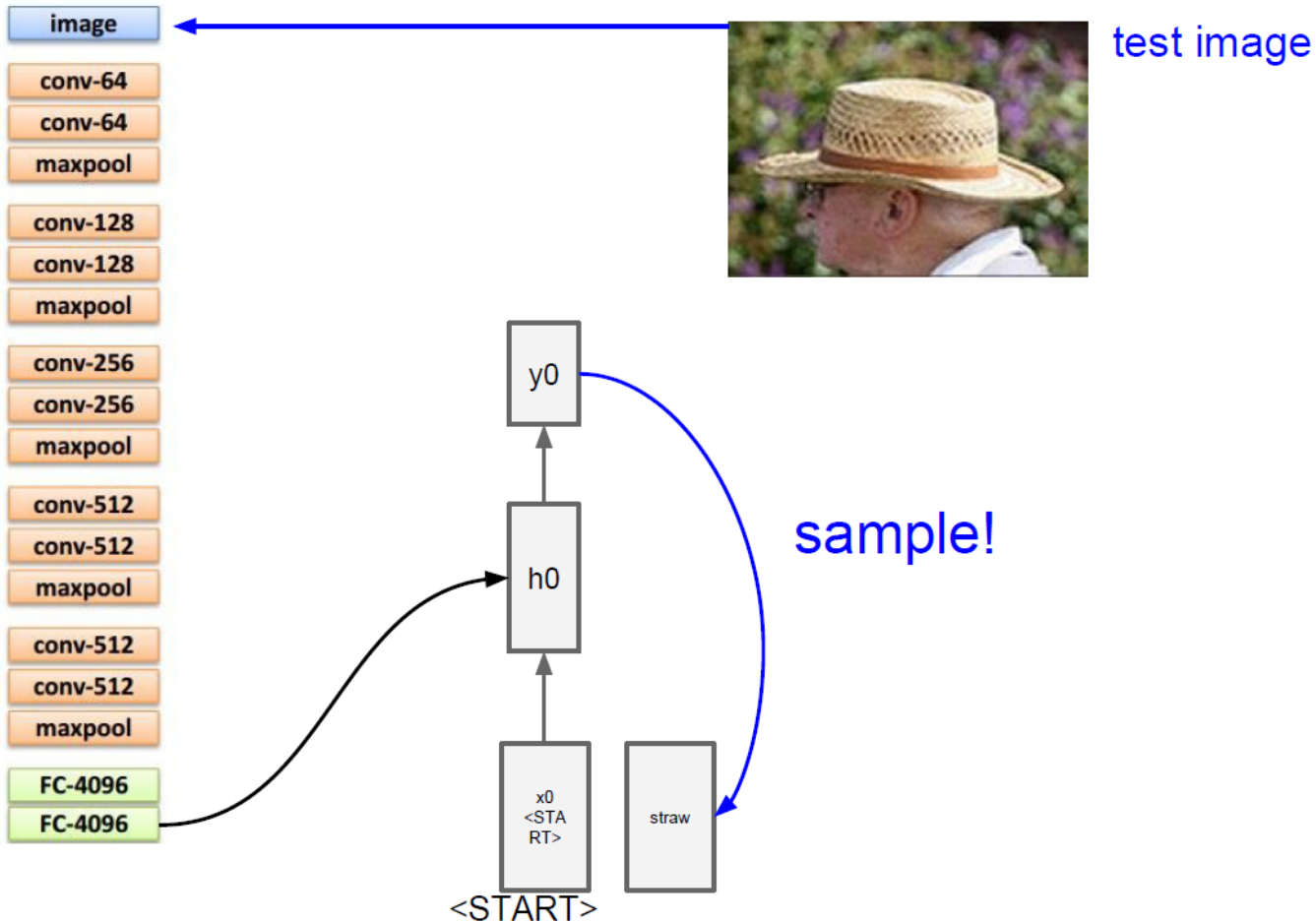


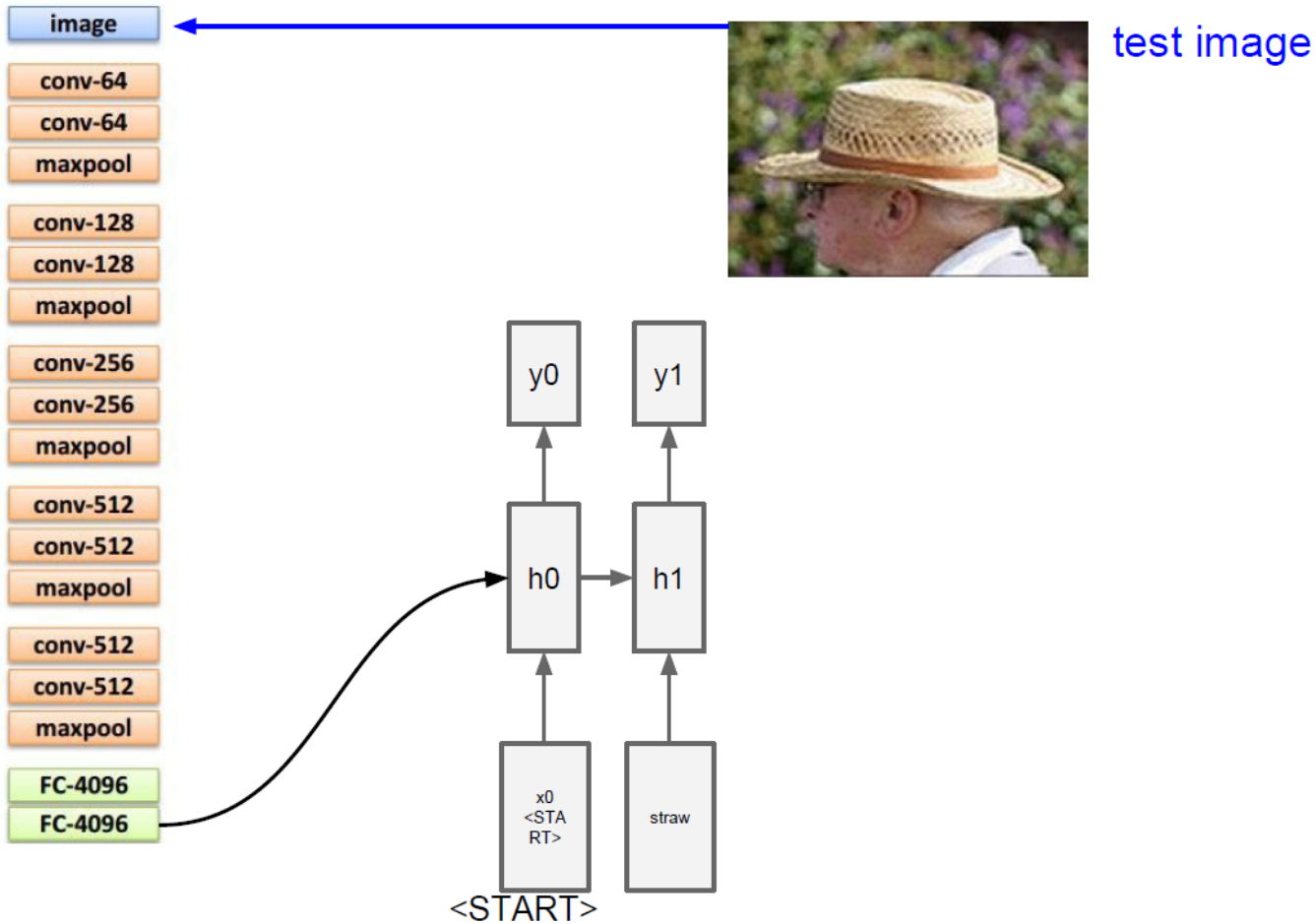
**before:**

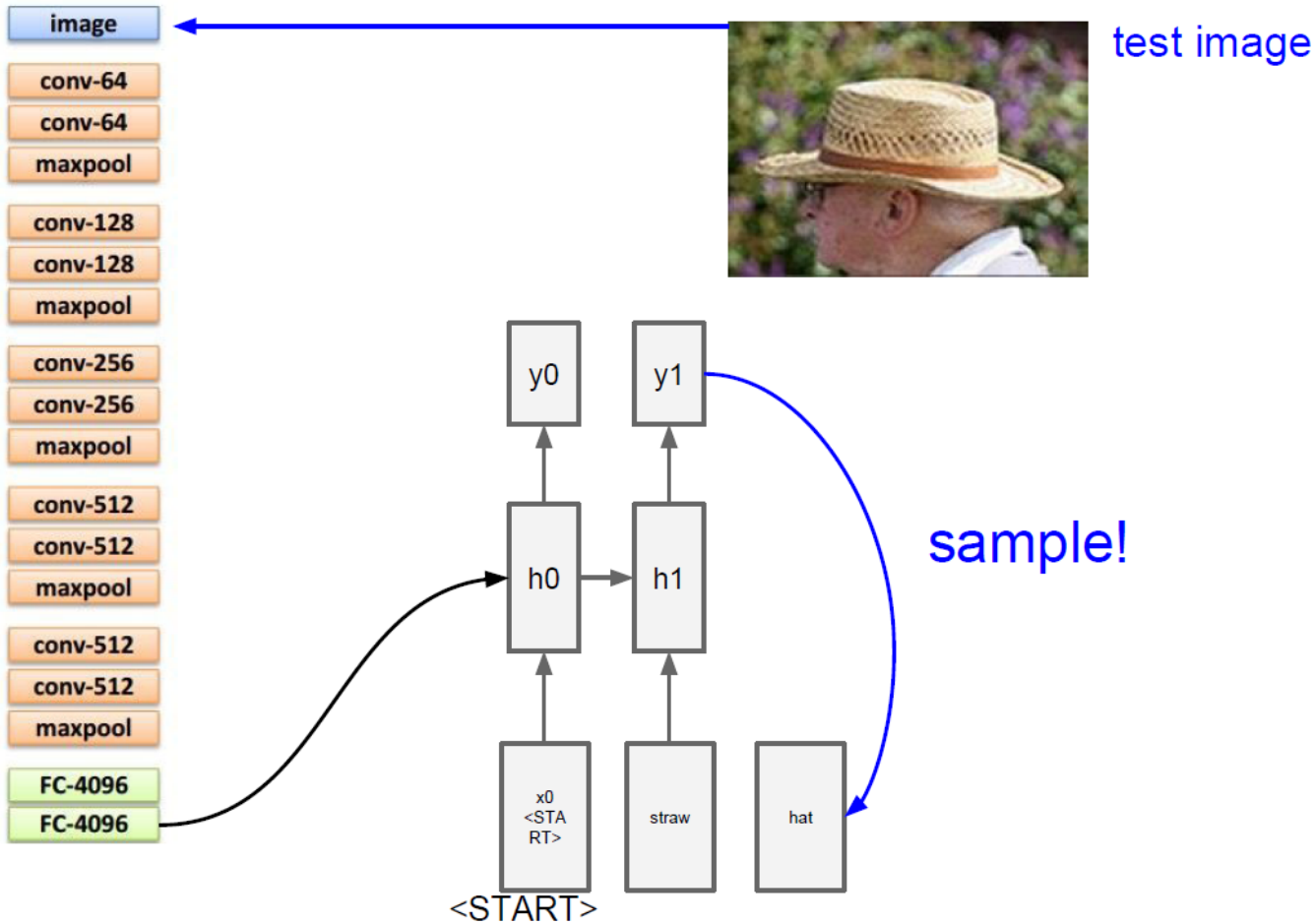
$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

**now:**

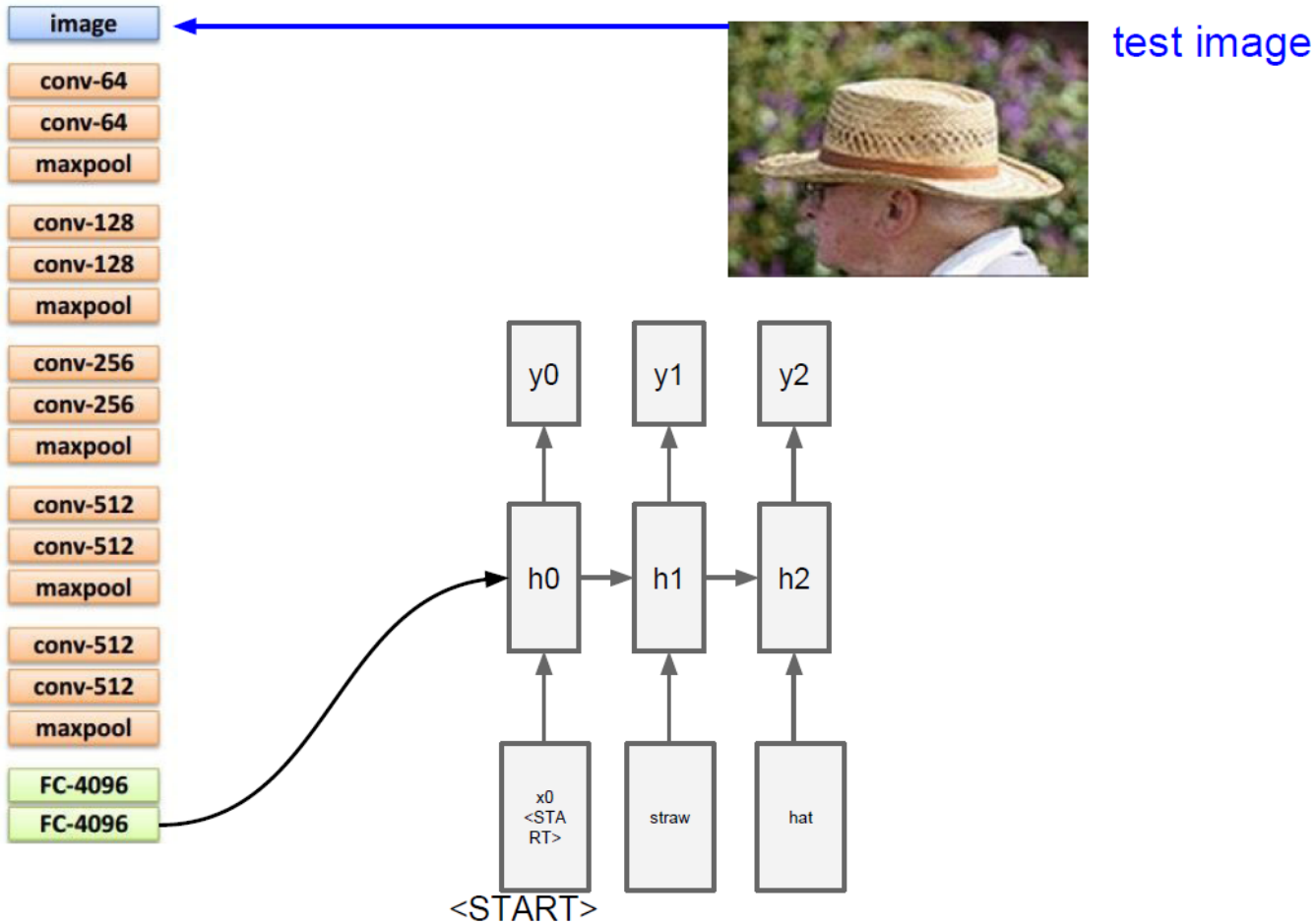
$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

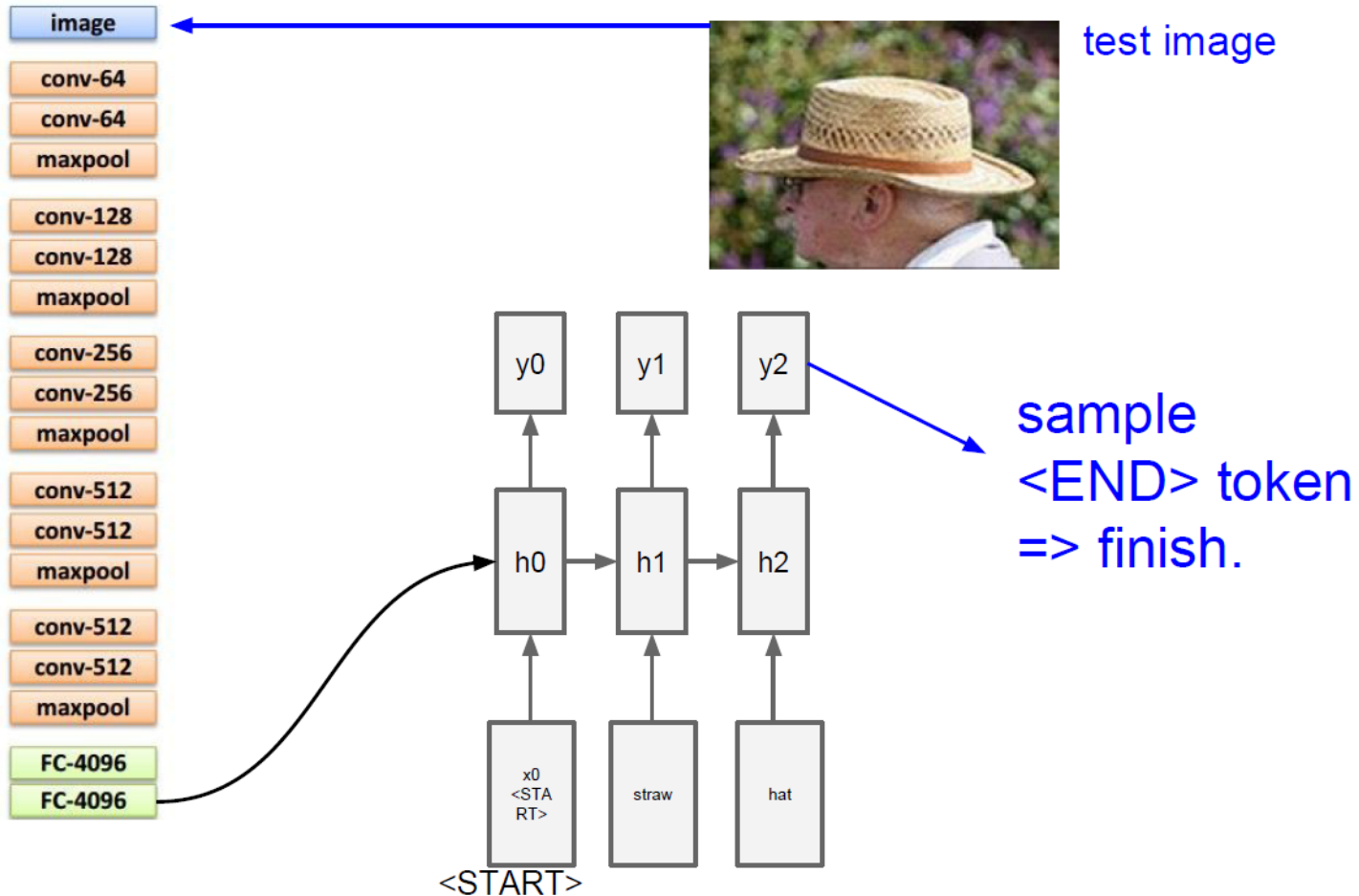












# Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- Backward flow of gradients in RNN can explode or vanish.
- Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)

# BACKUPS

---

# Backpropagation

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

$$y_t = W_{hy}h_t + b_y$$

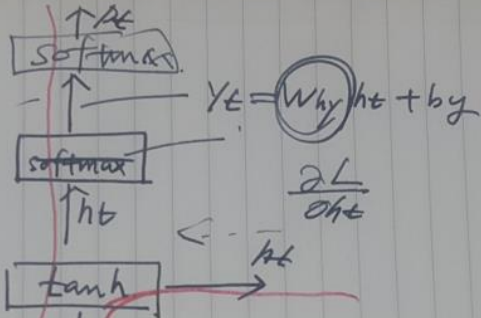
$$p_t = \text{softmax}(y_t)$$

$$l \leftarrow l - \log(p_t)^\top [0 \dots 1 \dots 0]$$

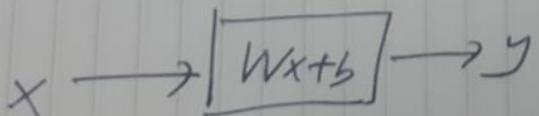
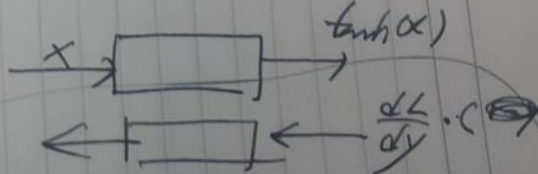
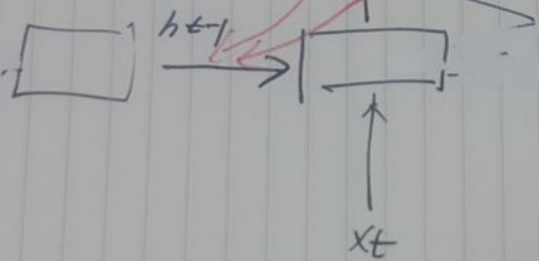
- $$z = -\log\left(\frac{e^{y^{(2)}}}{e^{y^{(1)}} + e^{y^{(2)}} + e^{y^{(3)}}}\right) = -y^{(2)} + \log\left(e^{y^{(1)}} + e^{y^{(2)}} + e^{y^{(3)}}\right)$$

$$\frac{d(\tanh f(x))}{dx} = (1 - \tanh^2 f(x))f'(x)$$

$\frac{\partial L}{\partial y}$



$W_x h_t + W_h h_t + b_h$



$\left(\frac{\partial L}{\partial y}\right)^T = \left(\frac{\partial L}{\partial y}\right)^T$ 
 $\left(\frac{\partial L}{\partial z}\right)^T = \left(\frac{\partial L}{\partial y}\right)^T x^T$ 
 $\left(\frac{\partial L}{\partial x}\right)^T = W^T \left(\frac{\partial L}{\partial y}\right)^T$ 
 $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} (1 - y^2)$

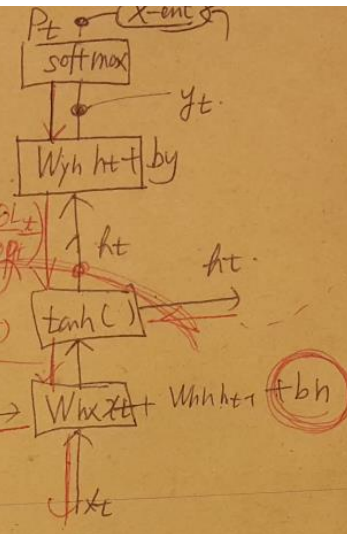
$$\left(\frac{\partial L}{\partial W_{jh}}\right)^T = \left(\frac{\partial L}{\partial y_t}\right)^T \cdot A_t^T$$

$$\left(\frac{\partial L}{\partial b_j}\right)^T = \left(\frac{\partial L}{\partial y_t}\right)^T$$

$$\left(\frac{\partial L_t}{\partial h_t}\right)^T = W_{jh}^T \left(\frac{\partial L}{\partial y_t}\right)^T$$

$$(1-h_t^2) \frac{\partial L_t}{\partial h_t}$$

We want  $\frac{\partial L_{t+1}}{\partial h_t}$



$\frac{\partial L_{t+1}}{\partial h_t}$

$$\frac{\partial L}{\partial W_{hx}}, \frac{\partial L}{\partial W_{hh}}, \frac{\partial L}{\partial b_h}$$

$$\frac{\partial L}{\partial y_t}, \frac{\partial L}{\partial W_{jh}} = \left(\frac{\partial L}{\partial y_t}\right)^T x_t^T$$

$$\left(\frac{\partial L}{\partial x_t}\right)^T = W^T \left(\frac{\partial L}{\partial y_t}\right)^T$$

# Backpropagation

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

$$y_t = W_{hy}h_t + b_y$$

$$p_t = \text{softmax}(y_t)$$

$$l \leftarrow l - \log(p_t)^T [0 \dots 1 \dots 0]$$

- For example,

$$\frac{\partial z}{\partial y} = \begin{bmatrix} p^{(1)} \\ -1 + p^{(2)} \\ p^{(3)} \end{bmatrix}$$

$$\frac{\partial z}{\partial W_{hy}} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial W_{hy}} = \frac{\partial z}{\partial y} h_t^\top$$

$$\frac{\partial z}{\partial b_y} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial b_y} = \frac{\partial z}{\partial y}$$



# Backpropagation

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

$$y_t = W_{hy}h_t + b_y$$

$$p_t = \text{softmax}(y_t)$$

$$l \leftarrow l - \log(p_t)^\top [0 \dots 1 \dots 0]$$

- For example,

$$\frac{\partial z}{\partial h} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial h} = \frac{\partial z}{\partial y} W_{hy}^\top$$

$$\frac{d(\tanh f(x))}{dx} = (1 - \tanh^2 f(x)) f'(x)$$