

# Recurrent Neural Network using TensorFlow

Youngjae Yu

May 18, 2017

<https://yj-yu.github.io/rnntf>

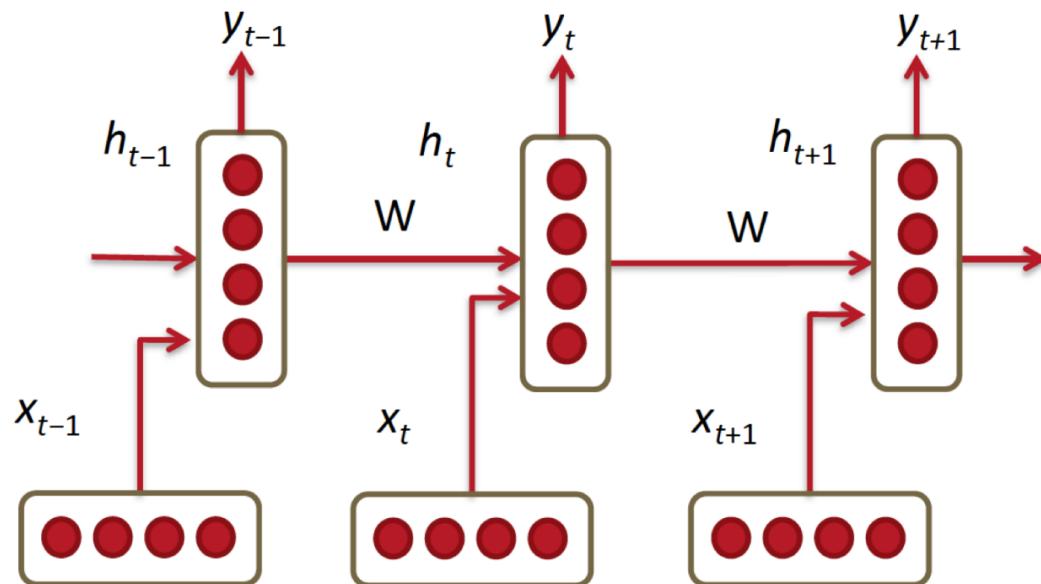


# About

- Contents
  - RNN Basics
  - Tensorflow Basics
  - Tensorflow APIs for RNN
  - Practice - RNN
  - Practice - Monitoring
  - Advanced
  - Advanced Practice - char rnn

# RNN Basics

# Recurrent Neural Networks!

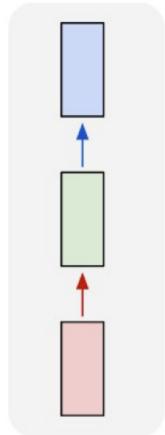


Richard Socher

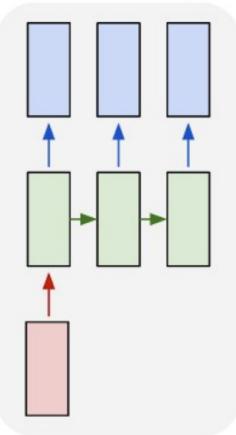
- RNNs tie the weights at **each time**
- Condition the neural network on **all previous words**
- RAM requirement only scales with number of words

# RNN Basics

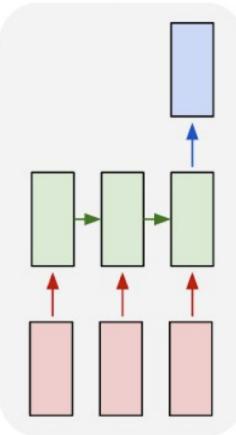
one to one



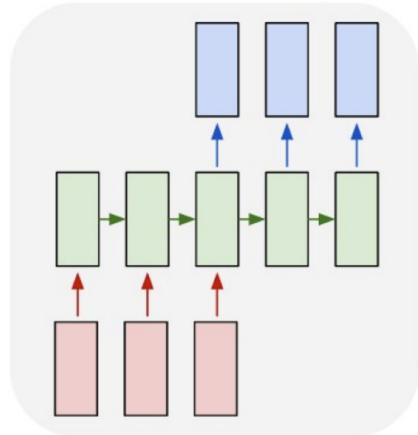
one to many



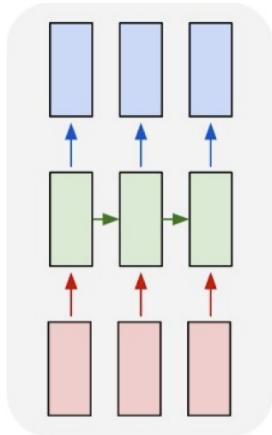
many to one



many to many



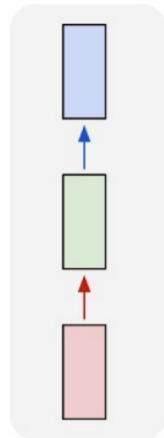
many to many



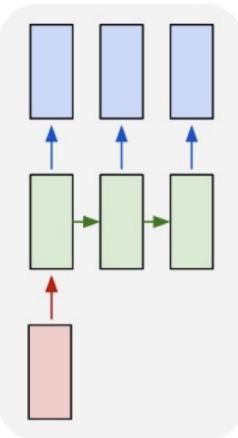
**Vanilla Neural Networks**

# RNN Basics

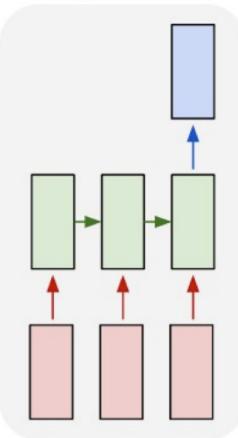
one to one



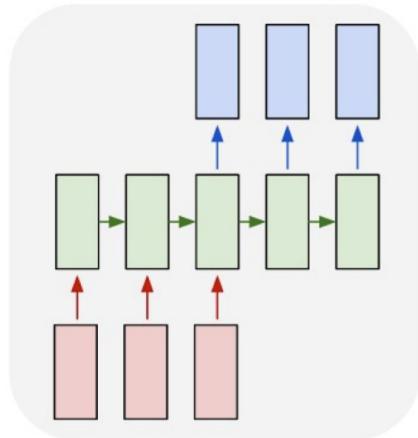
one to many



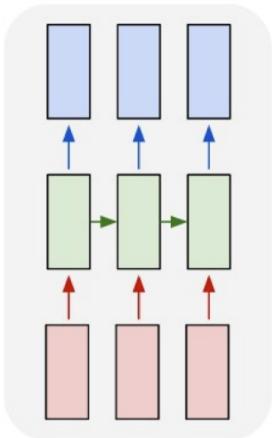
many to one



many to many



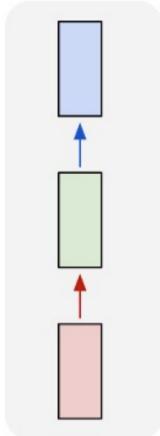
many to many



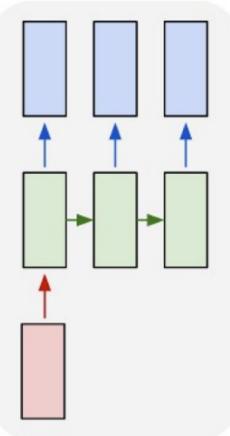
e.g. **Image Captioning**  
image -> sequence of words

# RNN Basics

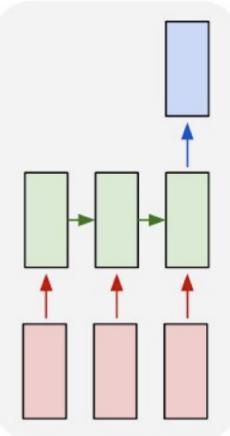
one to one



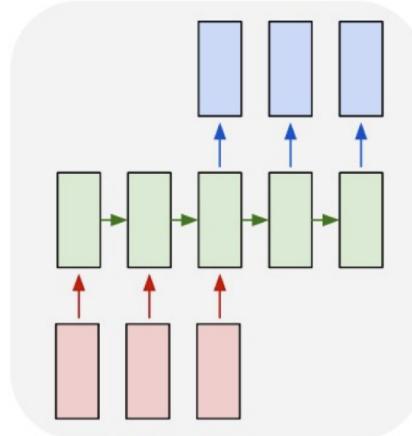
one to many



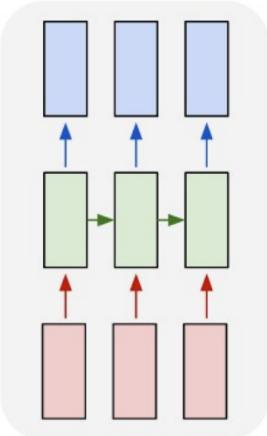
many to one



many to many



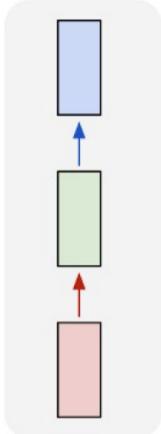
many to many



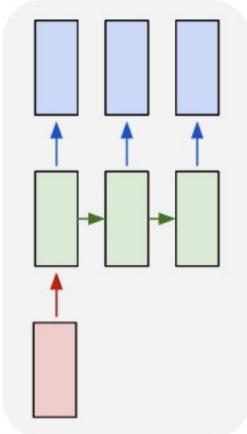
e.g. **Sentiment Classification**  
sequence of words → sentiment

# RNN Basics

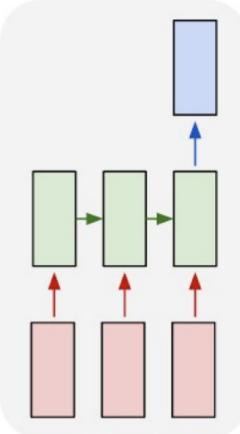
one to one



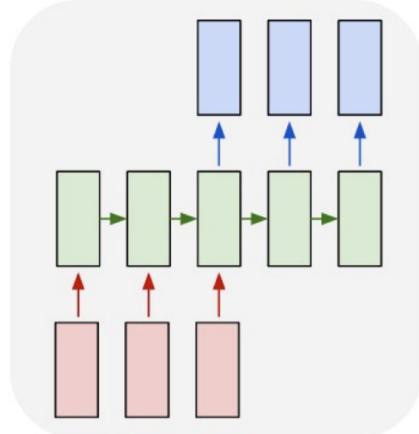
one to many



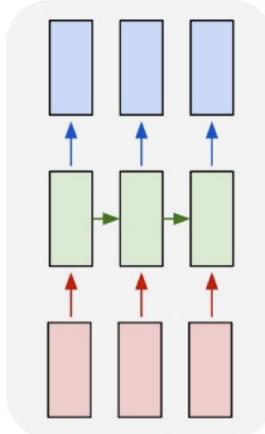
many to one



many to many



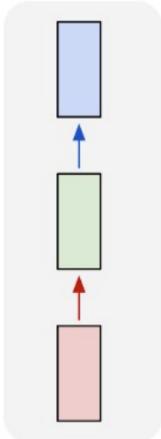
many to many



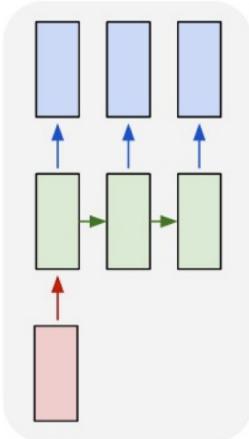
e.g. **Machine Translation**  
seq of words -> seq of words

# RNN Basics

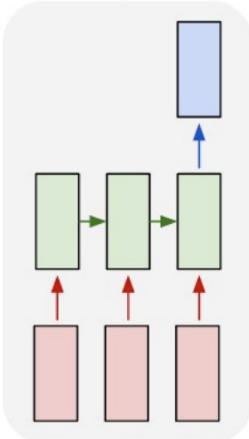
one to one



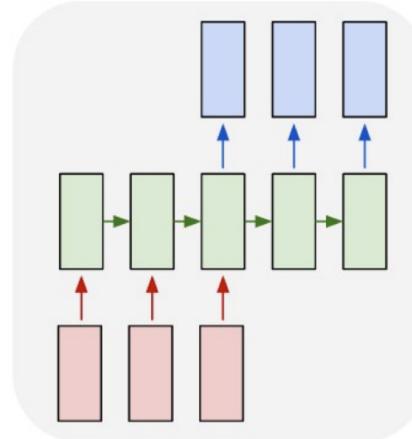
one to many



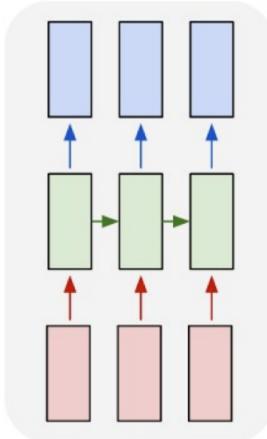
many to one



many to many

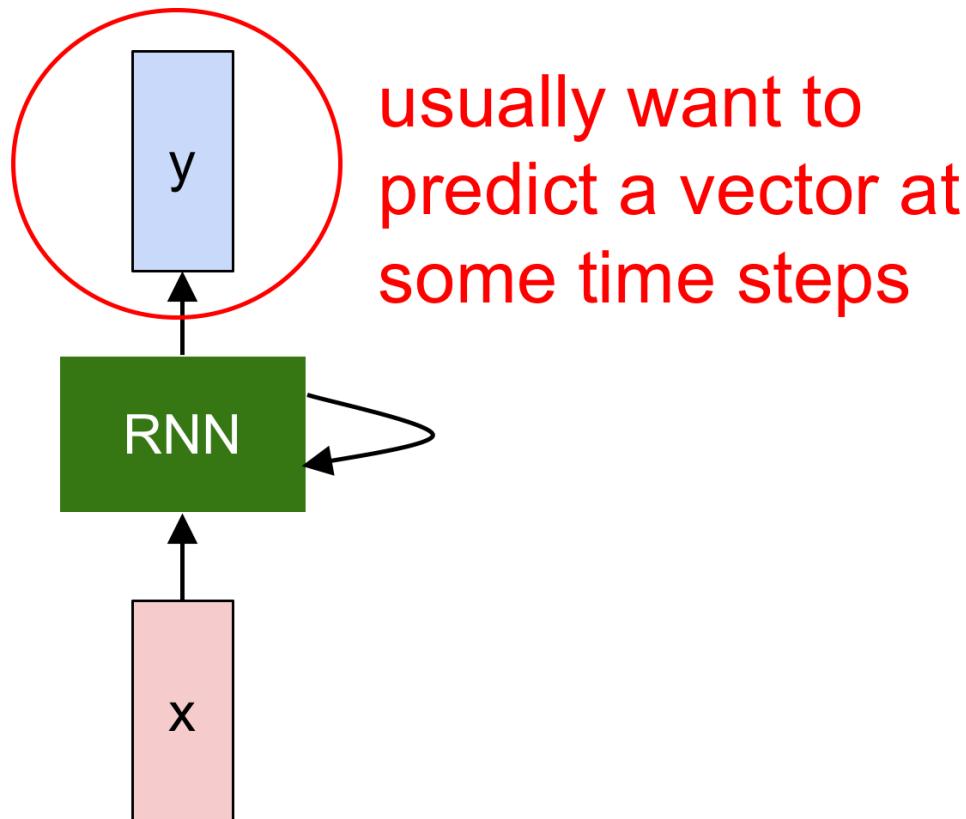


many to many



e.g. **Video classification on frame level**

# RNN Basics

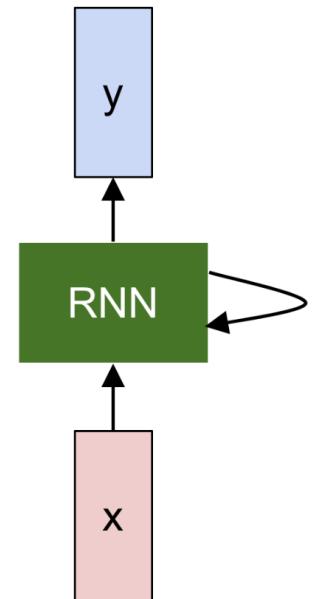


# RNN Basics

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

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

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

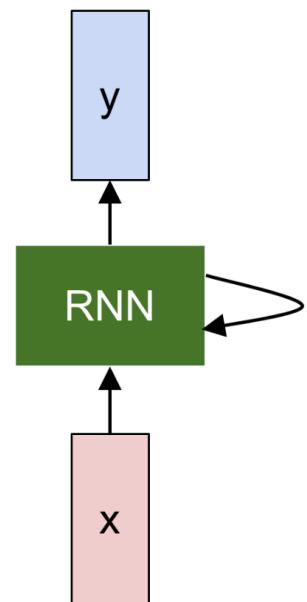


# RNN Basics

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

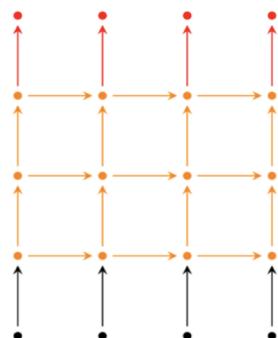
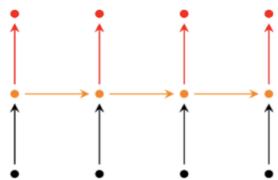
$$h_t = f_W(h_{t-1}, x_t)$$

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



# RNN Basics

RNN



$$h_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b}) \quad (1)$$
$$y_t = g(Uh_t + c) \quad (2)$$

$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b}) \quad (3)$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b}) \quad (4)$$

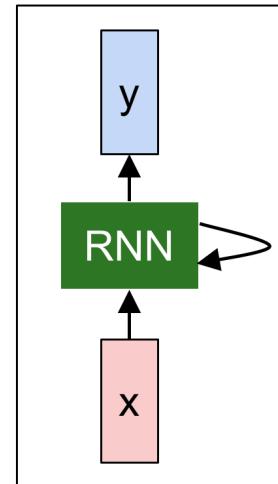
$$y_t = g(U_{\rightarrow}\vec{h}_t + U_{\leftarrow}\overleftarrow{h}_t + c) \quad (5)$$

# RNN Basic

**Character-level  
language model  
example**

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
**“hello”**

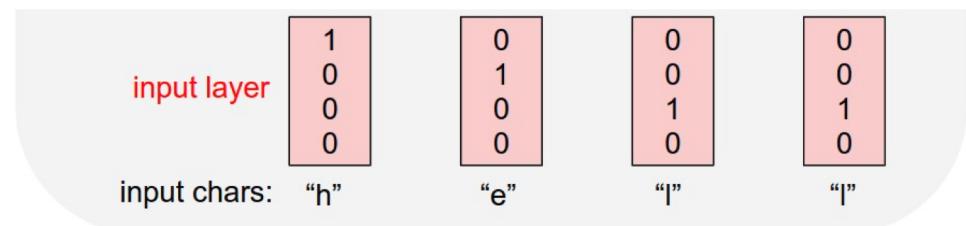


# RNN Basic

**Character-level  
language model  
example**

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
**“hello”**



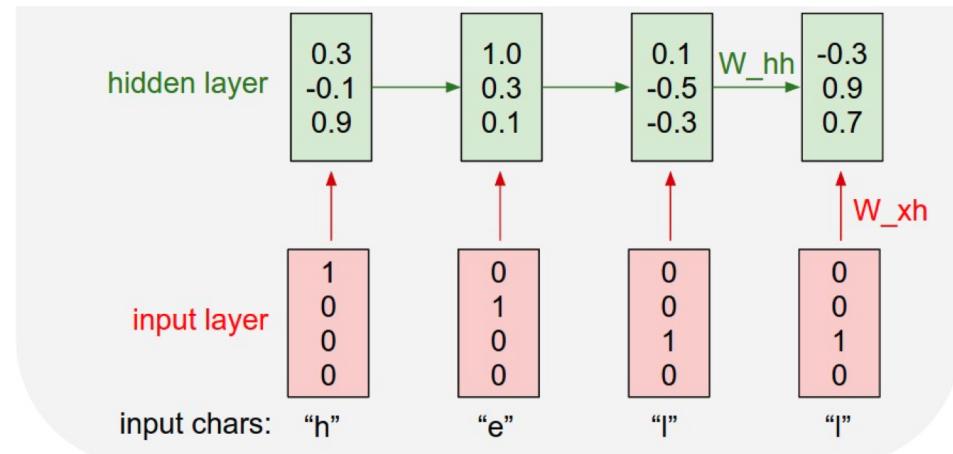
# RNN Basic

**Character-level  
language model  
example**

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
**“hello”**

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

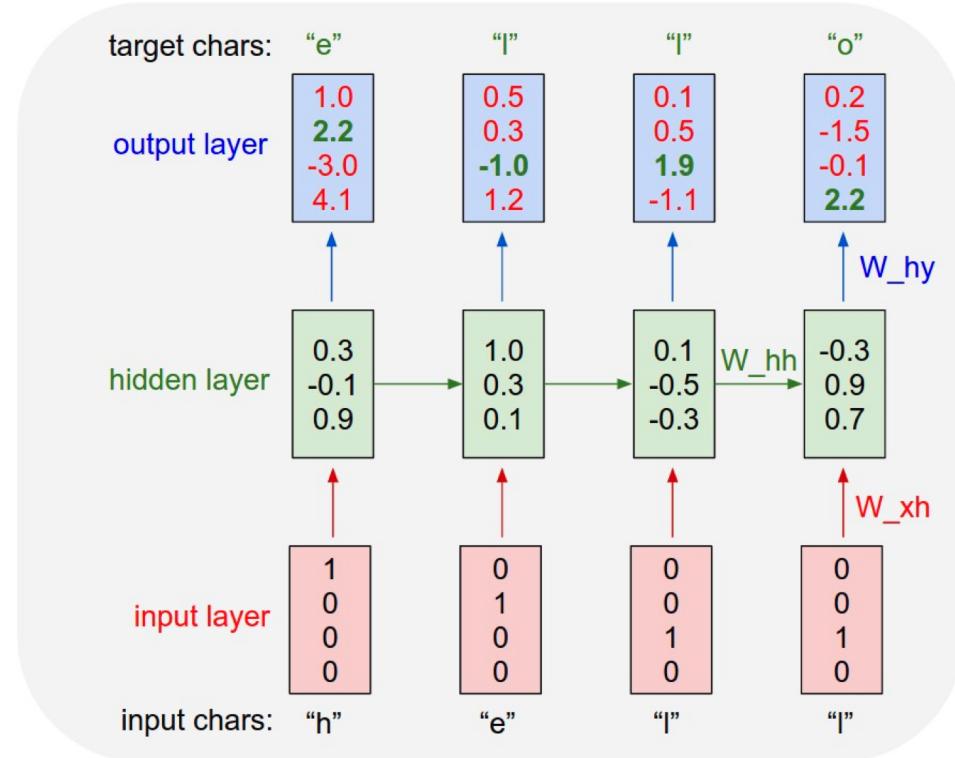


# RNN Basic

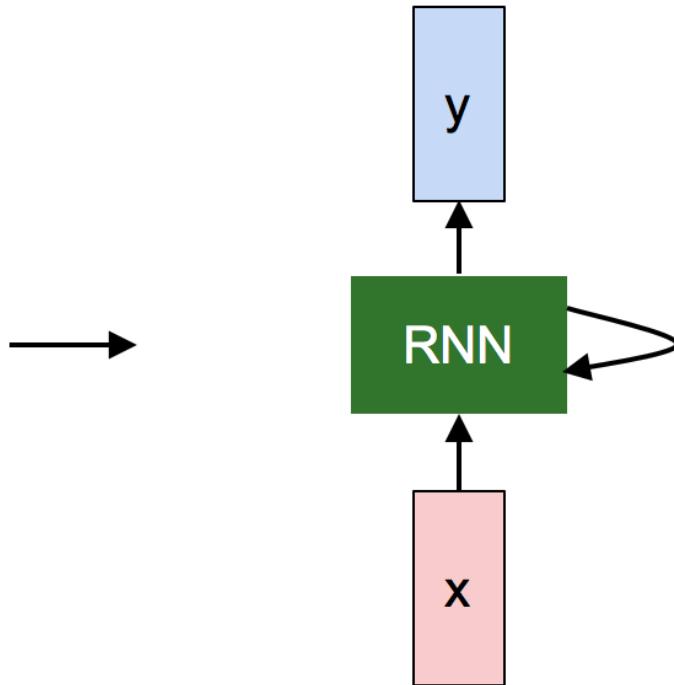
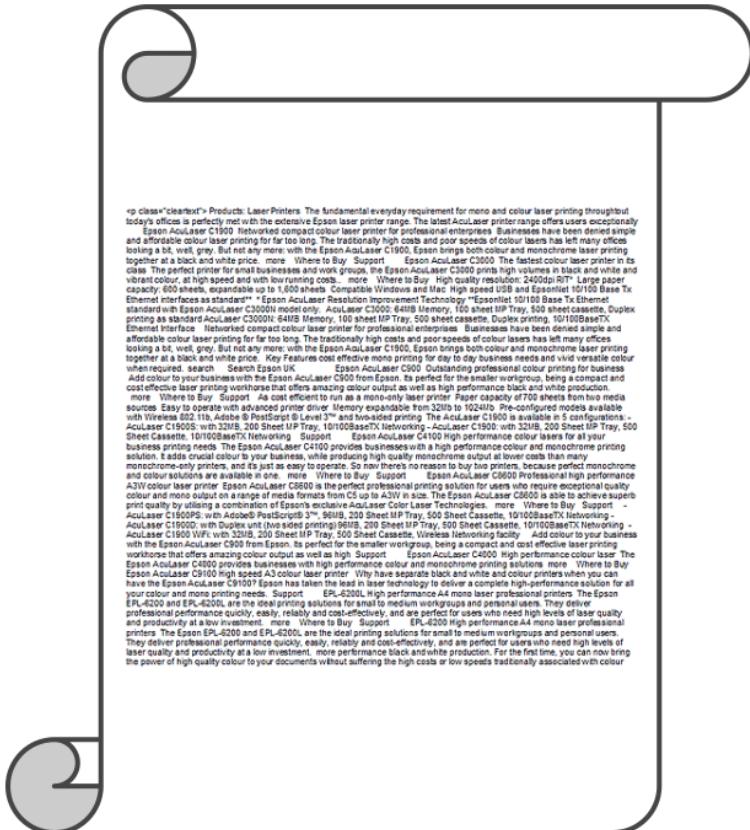
## Character-level language model example

Vocabulary:  
[h,e,l,o]

Example training sequence:  
“hello”



# RNN Basic



# RNN Basic

at first:

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

↓ train more

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

↓ train more

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

↓ train more

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

# RNN Python

<https://goo.gl/5npVuw>

# RNN Python

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

```
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3
4 BSD License
5
6
7 import numpy as np
8
9
10 # data I/O
11
12 def read_data(input_file, n_chars):
13     with open(input_file, 'r').read() as f:
14         chars = list(f.read())
15     data_size, vocab_size = len(chars), len(chars)
16     data, vocab = np.zeros((data_size, vocab_size), np.int32)
17     for i, ch in enumerate(chars):
18         data[i, char_to_ix[ch]] = 1
19
20    return data, vocab, data_size, vocab_size
```

```
21
22 hidden_size = 100 # size of hidden layer of neurons
23 seq_length = 20 # number of steps to unroll the RNN for
24 learning_rate = 1e-1
25
26
27 wih = np.random.rand(hidden_size, vocab_size)*0.01 # input to hidden
28 whh = np.random.rand(hidden_size, hidden_size)*0.01 # hidden to hidden
29 bhw = np.zeros(hidden_size, 1) # hidden bias
30 by = np.zeros(vocab_size, 1) # output bias
31
32 def loss(inputs, targets, hprev):
33     """ Inputs,targets are both list of integers.
34     -W is weight matrix from hidden state to hidden state
35     returns the loss, gradients in model parameters, and last hidden state
36     """
37     if hprev is None: hprev = np.zeros(hidden_size)
38
39     hs, hs, ys, ps = [], [], [], []
40     hs.append(hprev)
41
42     for t in range(len(inputs)):
43         x = np.zeros(vocab_size)
44         x[inputs[t]] = 1 # encode in 1-of-K representation
45         hprev = np.tanh(wih.dot(x) + whh.dot(hprev) + bhw)
46         y = np.dot(Wy, hprev) + by # hidden state
47         y[targets[t]] -= 1 # softmax: exp(y)/sum(exp(y))
48         ps_t = np.exp(y)/np.sum(np.exp(y)) # probabilities for next chars
49         ys.append(ps_t)
50         loss += -np.log(ps_t[targets[t]]) # softmax loss at step t
51
52         # backward pass: compute gradients going backwards
53         dprev_h = np.zeros(hidden_size)
54         dprev_wih = np.zeros((hidden_size, vocab_size))
55         dprev_whh = np.zeros((hidden_size, hidden_size))
56         dprev_bh = np.zeros(hidden_size)
57         dprev_by = np.zeros(vocab_size)
58
59         for t2 in range(t+1, len(inputs)):
60             dy = np.copy(ps_t[t2])
61             dy[targets[t2]] -= 1
62             dy = np.dot(Wy.T, dy) # backprop into y
63             dhy = np.dot(dy, hprev)
64             dhy *= dy
65             dhy = np.tanh(hprev.T) * dhy # backprop through tanh nonlinearity
66             dhprev = np.dot(dhy, Whh.T)
67             dprev_h += dhprev
68             dprev_wih += np.outer(dhy, x)
69             dprev_whh += np.outer(dhy, hprev)
70             dprev_bh += dhy
71             dprev_by += np.log(dy).sum()
72
73         for dataset in [dhs, dhy, dprev_h, dprev_wih, dprev_whh]:
74             np.clip(dataset, -5, 5, dataset) # clip to mitigate exploding gradients
75         return loss, dhs, dhy, dprev_h, dprev_wih, dprev_whh, dprev_bh, dprev_by, len(inputs)-1
76
77 def sample(hprev, seed_ix, n):
78     """ sample a sequence of integers from the model
79     hprev is hidden state, seed_ix is seed letter for first time step
80     n is length of sequence to sample """
81     x = np.zeros(vocab_size)
82     x[seed_ix] = 1
83
84     for t in range(n):
85         y = np.dot(Wy, hprev) + by
86         p = np.exp(y) / np.sum(np.exp(y))
87         x = np.random.choice(vocab_size, p=p)
88         hprev = np.tanh(wih.dot(x) + whh.dot(hprev) + bhw)
89         if x == 0: break
90
91    return x
```

```
92 n, p = 8, 8
93 batch, batch_size = np.zeros((batch_size, vocab_size)), np.zeros_like(p)
94 smooth_loss = -np.log(1.0/vocab_size)*seq_length + loss / iterations
95
96 while True:
97     if progress % 1000 == 0:
98         print 'smooth loss: %f' % smooth_loss
99
100     if progress % 10000 == 0:
101         print 'done'
102
103     if progress % 100000 == 0:
104         print 'done'
```

Data I/O

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

# RNN Python

## min-char-rnn.py gist

```

Minibatch character-level Vanilla RNN model, written by Andrej Karpathy (karpathy)
MIT License
Copyright (c) 2015 Andrej Karpathy

import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
data_size, vocab_size = len(data), len(set(data))
print "data has %d characters, %d unique." % (data_size, vocab_size)
chars = sorted(list(set(data)))
data_index_to_char = {i:ch for i, ch in enumerate(chars)}
char_to_ix = {ch:i for i, ch in enumerate(chars)}

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for learning
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(vocab_size, hidden_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def loss(inputs, targets, hprev):
    """ Inputs and targets are both lists of integers.
        hprev is not array of initial hidden state
        returns the loss, gradients on model parameters, and last hidden state """
    if hprev is None: hprev = np.zeros((hidden_size, 1))
    loss = 0
    dinputs = np.zeros_like(inputs)
    for t in range(len(inputs)):
        x_t = inputs[t] # as a one-hot vector
        x_t1 = np.zeros_like(x_t)
        x_t1[x_t] = 1 # encode in 1-of-a representation
        h_t = np.tanh(np.dot(Wxh, x_t) + np.dot(Whh, hprev) + bh) # hidden state
        y_t = np.dot(Why, h_t) + by # output score for next char
        p_t = np.exp(y_t)/np.sum(np.exp(y_t)) # probability for next char
        loss += -np.log(p_t[targets[t]]) # softmax (cross-entropy) loss
        dloss = -1/p_t[targets[t]] * np.ones_like(p_t) # softmax (cross-entropy) loss gradient
        dWhy = np.outer(h_t, dloss) # output layer gradients
        dbh = np.sum(dloss, axis=0) # hidden layer bias gradient
        dby = np.zeros_like(by) # output layer bias gradient
        dh_t = np.dot(Why.T, dloss) # hidden layer gradients
        dWxh = np.dot(x_t1.T, dh_t) # input layer gradients
        dWhh = np.dot(hprev.T, dh_t) # hidden layer gradients
        dWxh *= np.clip(dloss, -5, 5, out=dloss) # clip to mitigate exploding gradients
        dWhh *= np.clip(dloss, -5, 5, out=dloss) # clip to mitigate exploding gradients
        dbh *= np.clip(dbh, -5, 5, out=dbh) # clip to mitigate exploding gradients
        dby *= np.clip(dby, -5, 5, out=dby) # clip to mitigate exploding gradients
        dloss *= np.clip(dloss, -5, 5, out=dloss) # clip to mitigate exploding gradients
        dloss *= np.sum(dloss) # scale by number of steps
        dinputs[t] = dloss # store input layer gradients
    dWxh, dWhh, dbh, dby = dWxh / seq_length, dWhh / seq_length, dbh / seq_length, dby / seq_length
    dinputs = dinputs / seq_length
    return loss, dWxh, dWhh, dbh, dby, hprev

def sample(hprev, seed_ix, n):
    """ sample a sequence of integers from the model
        hprev is memory state, seed_ix is a seed letter for first time step
        n is max length, 100 """
    x = np.zeros((vocab_size, 1))
    x[seed_ix] = 1
    for t in range(n):
        hprev = np.tanh(np.dot(Wxh, x) + np.dot(Whh, hprev) + bh)
        y = np.dot(Why, hprev) + by
        p = np.exp(y)/np.sum(np.exp(y))
        ix = np.argmax(p)
        x = np.zeros_like(x)
        x[ix] = 1 # encode in 1-of-a representation
        print "predicted: ", data_index_to_char[ix]
    return ix

# forward pass: inputs through net and fetch gradients
def loss_and_update(inputs, targets, hprev, model):
    """ inputs, targets, hprev are arrays as in loss_and_update; model is RNN class instance """
    loss, dWxh, dWhh, dbh, dby, hprev = loss(inputs, targets, hprev)
    if t % 100 == 0: print "at step %d, loss is %f" % (t, loss)
    if t % 1000 == 0: print progress
    # perform parameter update with Adam
    for para, para_d, mom in zip(model.all_params, model.all_grads, model.all_moms):
        mom[0] = 0.9 * mom[0] + 0.1 * para_d[0]
        para[0] = para[0] - learning_rate * para_d[0] * np.sqrt(mom[0] + 1e-5) * np.sign(para_d[0])
        mom[1] = 0.9 * mom[1] + 0.1 * para_d[1]
        para[1] = para[1] - learning_rate * para_d[1] * np.sqrt(mom[1] + 1e-5) * np.sign(para_d[1])
    return loss, dWxh, dWhh, dbh, dby, hprev

# train loop
for i in range(100000):
    if i % 1000 == 0: print "step %d" % i
    loss, dWxh, dWhh, dbh, dby, hprev = loss_and_update(inputs, targets, hprev, model)
    if i % 10000 == 0: print "saving model"
    np.save('rnn.npz', model.all_params)

```

## Initializations

```

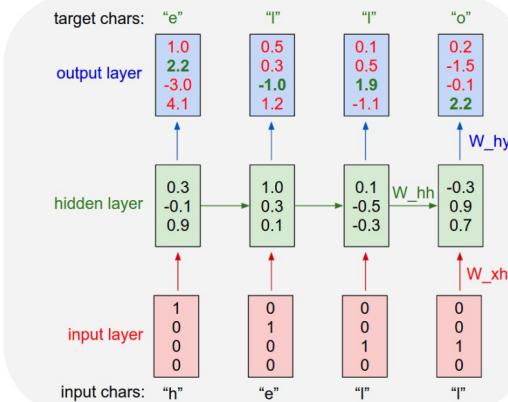
15      # hyperparameters
16      hidden_size = 100 # size of hidden layer of neurons
17      seq_length = 25 # number of steps to unroll the RNN for
18      learning_rate = 1e-1

19      # model parameters
20      Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
21      Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
22      Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
23      bh = np.zeros((hidden_size, 1)) # hidden bias
24      by = np.zeros((vocab_size, 1)) # output bias

25      def loss(inputs, targets, hprev):

```

recall:



# RNN Python

## min-char-rnn.py gist

```
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 data_size, vocab_size = len(data), len(chars)
9 chars = sorted(list(set(data)))
10 char_to_ix = {ch:i for i, ch in enumerate(chars)}
11 ix_to_char = {i:ch for i, ch in enumerate(chars)}
12
13 n, p = 0, 0
14
15 mWxh, mwLhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
16 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
17 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
18
19 while True:
20     # prepare inputs (we're sweeping from left to right in steps seq_length long)
21     if p+seq_length+1 >= len(data) or n == 0:
22         hprev = np.zeros((hidden_size,1)) # reset RNN memory
23         p = 0 # go from start of data
24
25     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
26     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
27
28     xs, hs, ys, ps = ([], [], [], [])
29
30     for t in range(seq_length):
31         x = np.zeros((vocab_size,1)) # encode in 1-of-k representation
32         x[inputs[t]] = 1
33         hprev = np.tanh(wxh.dot(x) + bh[t]) # hidden state
34         y = np.dot(why, hprev) + by # unnormalized log probabilities for next chars
35         p[t] = np.exp(y)/sum(np.exp(y)) # probabilities for next chars
36         ypred = np.argmax(p[t]) # our predicted character
37         xs.append(x) # input sequence
38         hs.append(hprev) # hidden state
39         ys.append(ypred) # our ground truth character
40         ps.append(p[t]) # probability of character
41
42     # backward pass: compute gradients going backwards
43     dWxh, dWhh, dWhy, dbh, dby, dWbh, dWby = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why), np.zeros_like(bh), np.zeros_like(by), np.zeros_like(bh), np.zeros_like(by)
44     dWbh, dWby = np.zeros_like(Wbh), np.zeros_like(Whb)
45
46     for t in reversed(range(seq_length)):
47         dy = np.copy(ps[t])
48         dy = np.copy(dy*(1-p[t])) # softmax gradient
49         dyt = np.tanh(hs[t]).dot(dy) # backprop into y
50         dWxh += np.outer(xt, dyt) # input gradient
51         dWhh += np.outer(hs[t], dyt) # hidden gradient
52         dWhy += dyt # bias gradient
53         dbh += dyt # hidden bias gradient
54         dby += dyt # bias gradient
55
56     # clip gradients to mitigate exploding gradients
57     np.clip(dWxh, 3, 5, out=dWxh) # clip to mitigate exploding gradients
58     np.clip(dWhh, 3, 5, out=dWhh) # clip to mitigate exploding gradients
59     np.clip(dbh, 0, 1000, out=dbh) # clip to mitigate exploding gradients
60     np.clip(dbh, 0, 1000, out=dbh) # clip to mitigate exploding gradients
61
62     smooth_loss = smooth_loss * 0.999 + loss * 0.001
63
64     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
65
66     # perform parameter update with Adagrad
67     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
68                                   [dWxh, dWhh, dWhy, dbh, dby],
69                                   [mWxh, mwLhh, mWhy, mbh, mby]):
70         mem += dparam * dparam
71         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
72
73     p += seq_length # move data pointer
74     n += 1 # iteration counter
```

## Main loop



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

# RNN Python

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

```

1  #include character-level vanilla RNN model. Written by Andrej Karpathy (@karpathy)
2  #
3  # BSD License
4  #
5  import numpy as np
6
7  # data I/O
8  #
9  # Input file (e.g. "input.txt") + read() = should be simple plain text file
10 chars = list(open(data))
11 data_size, vocab_size = len(chars), len(chars)
12 char_to_ix = {c: i for i, c in enumerate(chars)}
13 ix_to_char = {i: c for i, c in enumerate(chars)}
14
15 # Hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 200 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # weights
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 bhh = np.zeros((hidden_size, 1)) # hidden bias
24 bh = np.zeros((hidden_size, 1)) # hidden to output
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 tuple (loss, gradients on model parameters, and last hidden state
32     """
33     if hprev is None: hprev = np.zeros((hidden_size, 1))
34     n, hs, ys, ps = 0, 0, 0, 0
35     hprev = np.copy(hprev)
36     loss = 0
37     for t in range(len(inputs)):
38         x = inputs[t] # input at time t
39         x1 = np.zeros((vocab_size, 1)) # encode in 1-of-K representation
40         x1[inputs[t]] = 1
41         h1 = x1.dot(Wxh) + hprev * np.array([0.1, -0.1]) # hidden state
42         ps1 = np.tanh(np.dot(h1, whh) + bhh) # tanh nonlinearity
43         ps1 = ps1 / np.sum(ps1) * np.exp(ps1) # softmax (cross-entropy loss)
44         loss += -np.log(ps1[targets[t], 0]) + 1e-6 # softmax (cross-entropy loss)
45         dps1 = np.exp(ps1) / np.sum(np.exp(ps1))
46         dwhh = np.outer(dps1[0], ps1[0])
47         dWxh = np.outer(x1, ps1[0])
48         dbhh = np.zeros_like(bhh)
49         dh1 = np.zeros_like(h1)
50         dby = np.zeros_like(by)
51         dnext = np.zeros_like(h0)
52
53         # For t <= 1 reverse gradient w.r.t inputs:
54         if t > 1:
55             dtarget[t-1] = 1 # backprop into y
56             dy = np.zeros((vocab_size, 1))
57             dhy = dy * np.tanh(h1) # backprop through tanh nonlinearity
58             dwhh += dhy * np.outer(h1, h1) # dh = backprop through tanh nonlinearity
59             dwhh -= dwhh * np.outer(h1, h1)
60             dWxh += x1 * dhy
61             dbhh += dhy
62             dby += dhy
63             dnext += dhy
64
65         # For t >= 1 forward gradient w.r.t. hidden states:
66         if t < seq_length-1:
67             dtarget[t+1] = 1 # backprop into y
68             dy = np.zeros((vocab_size, 1))
69             dhy = dy * np.tanh(h1) # backprop through tanh nonlinearity
70             dwhh += dhy * np.outer(h1, h1) # dh = backprop through tanh nonlinearity
71             dwhh -= dwhh * np.outer(h1, h1)
72             dWxh += x1 * dhy
73             dbhh += dhy
74             dby += dhy
75             dnext += dhy
76
77         # clip gradients to mitigate exploding gradients
78         if np.linalg.norm(dWxh) > 500: dWxh *= 500/np.linalg.norm(dWxh)
79         if np.linalg.norm(dwhh) > 500: dwhh *= 500/np.linalg.norm(dwhh)
80         if np.linalg.norm(dbhh) > 500: dbhh *= 500/np.linalg.norm(dbhh)
81         if np.linalg.norm(dby) > 500: dby *= 500/np.linalg.norm(dby)
82
83         # sample from the model now and then
84         if n % 100 == 0:
85             sample_ix = sample(hprev, inputs[0], 200)
86             txt = ''.join(ix_to_char[ix] for ix in sample_ix)
87             print '----\n%s\n----' % (txt, )
88
89     # forward seq_length characters through the net and fetch gradient
90     loss, dWxh, dwhh, dby, dbhh, hprev = lossFun(inputs, targets, hprev)
91     smooth_loss = smooth_loss * 0.999 + loss * 0.001
92
93     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
94
95     # perform parameter update with Adagrad
96     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
97                                   [dWxh, dWhh, dWhy, dbh, dby],
98                                   [mWxh, mWhh, mWhy, mbh, mbby]):
99         mem += dparam * dparam
100        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
101
102    p += seq_length # move data pointer
103    n += 1 # iteration counter
104
105    # save progress every now and then
106    if n % 1000 == 0:
107        sample_ix = sample(hprev, inputs[0], 100)
108        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
109        print '----\n%s\n----' % (txt, )
110
111    # Average loss across characters in one sequence
112    loss, dWxh, dwhh, dby, dbhh, hprev = lossFun(inputs, targets, hprev)
113    smooth_loss = smooth_loss * 0.999 + loss * 0.001
114
115    if n % 1000 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
116
117    # perform parameter update with Adagrad
118    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
119                                  [dWxh, dWhh, dWhy, dbh, dby],
120                                  [mWxh, mWhh, mWhy, mbh, mbby]):
121        mem += dparam * dparam
122        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
123
124    p += seq_length # move data pointer
125    n += 1 # iteration counter

```

## Main loop

```

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

```



# RNN Python

## min-char-rnn.py gist

```
BSD License
Copyright (c) 2012 Andrej Karpathy (@karpathy)
All rights reserved.

import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
data_size, vocab_size = len(data), len(chars)
char_to_ix = {ch:i for i,ch in enumerate(chars)}
ix_to_ch = {i:ch for ch in enumerate(chars)}

# hyperparameters
hidden_size = 200 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1
batch_size = 100

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*np.sqrt(0.01)
Whh = np.random.randn(hidden_size, hidden_size)*np.sqrt(0.01)
Why = np.random.randn(vocab_size, hidden_size)*0.01
bh = np.zeros(hidden_size)
by = np.zeros(vocab_size, 1) # output bias

def lossFun(inputs, targets, hprev):
    """ Inputs: targets are back list of integers.
    hprev: is not array of initial hidden state
    returns the loss, gradients on model parameters, and last hidden state """
    n, p, m = 0, 0, 0
    x, h, ys = p, 0, []
    for i in range(seq_length):
        x = inputs[i]
        if i >= 0: h = np.copy(hprev)
        if i >= 1: h = np.tanh(h)
        wih = np.dot(vocab_size, ix_to_ch[x]) + np.dot(bh, h[1:-1]) + m
        hiw = np.tanh(wih)
        why = np.dot(vocab_size, ix_to_ch[y]) + by
        p += np.sum(np.log(why[i]*softmax(hiw))) # softmax (cross-entropy loss)
        loss += -np.log(why[i]*softmax(hiw))
        dhiw, dwih, dbh, dby = np.zeros_like(wih), np.zeros_like(bh), np.zeros_like(by)
        dhiw, dwih, dbh, dby = np.zeros_like(wih), np.zeros_like(bh), np.zeros_like(by)
        for t in reversed(range(i+1, inputs)):
            dy = np.copy(wih[t])
            dh = np.dot(Wxh, dy) + dbh + dhiw
            dhiw = (1 - hiw**2) * dh # tanh nonlinearity
            dbh += np.dot(dy, hiw[1:-1]) # dh through tanh nonlinearity
            dwih += np.dot(dy, vocab_size, ix_to_ch[x]) # dh through input weights
            dby += np.sum(np.log(why[t]*softmax(hiw)))
        for dim in [dhiw, dwih, dbh, dby]:
            dim *= learning_rate # clip to mitigate exploding gradients
        return loss, dhiw, dwih, dbh, dby, hiw[inputs]
    n += 1
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n% %----' % (txt, )
    if n % 100 == 0:
        sample_ix = sample(hprev, inputs[0], 200)
        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
        print '----\n% %----' % (txt, )

# forward seq_length characters through the net and fetch gradient
loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
smooth_loss = smooth_loss * 0.999 + loss * 0.001
if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

# perform parameter update with Adagrad
for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                               [dWxh, dWhh, dWhy, dbh, dby],
                               [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

p += seq_length # move data pointer
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% %----' % (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
```

# RNN Python

## min-char-rnn.py gist

```
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 MIT License
4
5 Import numpy as np
6
7 data = []
8 def readInput(fpath, seqLength=1):
9     chars = list(open(fpath).read())
10    # remove punctuation, lowercase, len(chars) = data.size, vocab.size
11    print "data has %d elements, %d unique" % (data.size, vocab.size)
12    char_ix_1k = { ch: i for i, ch in enumerate(chars) }
13    ix_to_char = { i: ch for ch, i in enumerate(chars) }
14
15    hidden_size = 100 # size of hidden layer of neurons
16    seq_length = 20 # number of steps to unroll the RNN for
17    seq_start = 0 # start of sequence
18
19    model_parameters = {}
20
21    wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22    who = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23    bh = np.zeros((hidden_size, 1)) # hidden bias
24    by = np.zeros(hidden_size, 1) # output bias
25
26    np.random.seed(1337)
27
28    def lossFun(inputs, targets, hprev):
29
30        inputs,targets = both_list_of_integers()
31
32        hprev = np.zeros((hidden_size, 1)) # hidden state
33        hidden_state = np.zeros_like(hprev) # hidden state
34        hidden_state[0] = hprev[0]
35        hidden_state[1] = np.copy(hprev)
36
37        loss = 0
38        sample_ixs = []
39
40        for t in xrange(len(inputs)):
41            x_t = np.zeros(vocab_size, 1) # encode in 1-of-K representation
42            x_t[int(inputs[t])] = 1
43
44            h_t = np.tanh(np.dot(wih, x_t) + np.dot(who, hidden_state[t-1]) + bh) # hidden state
45            y_t = np.dot(bh, h_t) # hidden state to output
46            p_t = np.exp(y_t)/sum(np.exp(y_t)) # probabilities for next chars
47            loss += -np.log(p_t[int(targets[t])]) # softmax (cross-entropy loss)
48
49            hidden_state[t] = h_t
50            hidden_state[t+1] = np.tanh(np.dot(who, hidden_state[t]) + np.dot(wih, x_t) + bh)
51            hidden_state[t+1] = np.zeros_like(hidden_state[t])
52
53            dprev, dwih, dwho, dbh, dby = np.zeros_like(wih), np.zeros_like(who), np.zeros_like(bh),
54            np.zeros_like(dbh), np.zeros_like(dby)
55
56            dh_t = np.dot(wih.T, dprev) + np.dot(who.T, dprev) + dbh
57            dh_t = np.clip(dh_t, -5, 5) # clip to mitigate exploding gradients
58
59            dwih += np.outer(x_t, dh_t)
60            dwho += np.outer(hidden_state[t-1], dh_t)
61            dbh += dh_t
62            dby += np.sum(dh_t)
63
64        return loss, dwih, dwho, dbh, dby, hidden_state[-1]
65
66    seed_ix = np.random.randint(vocab_size)
67    seed_ixs = [seed_ix]
68
69    for t in range(seqLength):
70        inputs,targets = both_list_of_integers()
71        hidden_state, loss, dwih, dwho, dbh, dby, hprev = lossFun(inputs, targets, hprev)
72
73        if t % 100 == 0:
74            print '----\n% %----' % (txt, )
75
76        sample_ix = sample(hprev, inputs[0], 200)
77        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
78        print '%----\n% %----' % (txt, )
79
80    # forward seq_length characters through the net and fetch gradient
81    loss, dwxh, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
82    smooth_loss = smooth_loss * 0.999 + loss * 0.001
83
84    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
85
86    # perform parameter update with Adagrad
87    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
88                                  [dWxh, dWhh, dWhy, dbh, dby],
89                                  [mWxh, mWhh, mWhy, mbh, mby]):
90
91        mem += dparam * dparam
92        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
93
94        p += seq_length # move data pointer
95        n += 1 # iteration counter
96
97    print '----\n% %----' % (txt, )
```

## Main loop

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



# RNN Python

## min-char-rnn.py gist

```

1  #!/usr/bin/python
2  #
3  # Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (karpathy)
4  #
5  # BSD License
6  #
7  # Import numpy as np
8  #
9  # Data I/O
10 # data = open('input.txt', 'r').read() # should be simple plain text file
11 # data_size, vocab_size = 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 # Weights matrix: size of hidden layer of neurons
17 seq_length = 20 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19 #
20 # Model parameters
21 Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size,)) # hidden bias
25 by = np.zeros((vocab_size,)) # output bias
26 #
27 def lossFun(inputs, targets, hprev):
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     n, p = 0, 0
34     loss = 0
35     for step in range(len(inputs)):
36         x = inputs[step]
37         if step == 0:
38             xst = np.zeros((vocab_size,)) # encode in 1-of-k representation
39         else:
40             xst = np.zeros((vocab_size,)) + np.dot(Wxh, hs[-1]) + bh
41         hs.append(np.tanh(xst))
42         yst = np.dot(Why, hs[-1]) + by # unnormalized log probabilities for next chars
43         ps.append(np.exp(yst)) # softmax (exp) raw scores for next chars
44         loss -= -np.log(ps[step][targets[step]]) # softmax (cross-entropy loss)
45         hprev = np.dot(Why, hs[-1]) + bh # backprop into previous hidden state
46         dbh, dWhy = np.zeros_like(bh), np.zeros_like(Why)
47         dby, dWxh, dWhh, dbh = np.zeros_like(by), np.zeros_like(Whh),
48         dWxh, dWhh, dby = np.zeros_like(Wxh)
49         for t in reversed(range(step)):
50             dy = (ps[t] - targets[t]) * np.exp(yst) / ps[t] # backprop through softmax
51             dyx = np.dot(Why.T, dy) # unstack and copy gradient going backward
52             dWhy += np.outer(dy, hs[t])
53             dby += np.sum(dy * np.ones_like(by))
54             dWxh += np.outer(dy, inputs[t])
55             dWhh += np.outer(dy, hs[t-1])
56             dbh += np.zeros_like(bh)
57             dprev = np.dot(Why, dy) # copy gradient back to previous hidden state
58             for param in [dWxh, dWhh, dby, dbh]:
59                 np.clip(param, -5, 5, out=param) # clip to mitigate exploding gradients!
60             loss, dx, dabs, dwhy, dbh, dby, hprev = lossFun(inputs[1:])
61     return loss, dx, dabs, dwhy, dbh, dby, hprev[inputs[1:]]
62     del sample_ix, seed_ix, n
63 #
64 # Sample a sequence of integers from the model
65 # h is the hidden state we need for first time step
66 # x is the input word index
67 # xst is the one-hot encoding of x
68 # s is the sequence of hidden states
69 # s.append(h)
70 # s.pop(0) # pop first
71 # xst = np.zeros((vocab_size,)) # encode in 1-of-k representation
72 # h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
73 # yst = np.dot(Why, h) + by # unnormalized log probabilities for next chars
74 # ps = np.exp(yst) # softmax (exp) raw scores for next chars
75 # loss -= -np.log(ps[targets[0]]) # softmax (cross-entropy loss)
76 # hprev = np.dot(Why, h) + bh # backprop into previous hidden state
77 # dbh, dWhy = np.zeros_like(bh), np.zeros_like(Why)
78 # dby, dWxh, dWhh, dbh = np.zeros_like(by), np.zeros_like(Whh),
79 # dWxh, dWhh, dby = np.zeros_like(Wxh)
80 #
81 # Forward seq_length characters through the net and fetch gradients
82 # loss, dx, dabs, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
83 # If n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) + print progress
84 #
85 # Perform parameter update with Adagrad
86 for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
87                               [dWxh, dWhh, dWhy, dbh, dby],
88                               [mWxh, mWhh, mWhy, mbh, mby]):
89     mem += dparam * dparam
90     param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
91     p += seq_length # move data pointer
92     n += 1 # iteration counter
93 
```



## 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

```

# RNN Python

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

```

1 #include character-level vanilla RNN model. Written by Andrej Karpathy (@karpathy)
2
3 BSD License
4
5 import numpy as np
6
7 # print 1/0
8
9 data = open('input.txt', 'r').read() # should be simple plain text file
10 data_size, vocab_size = len(data), len(chars)
11 print "data has %d characters, %d unique" % (data_size, vocab_size)
12 chars = sorted(list(set(data)))
13 ix_to_char = {i:ch for i,ch in enumerate(chars)}
14
15 # hyperparameters
16 hidden_size = 200 # size of hidden layer of neurons
17 n_time_steps = 1000 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 # forward pass: compute hidden states going forward
28 def forward(inputs, targets, hprev):
29     inputs, targets = map(list, inputs), targets
30     hprev is not list of integers.
31     hprev is not array of initial hidden state
32     returns the loss, gradients on model parameters, and last hidden state
33
34     xs, hs, ys, ps = {}, {}, {}, {}
35     hs[-1] = np.copy(hprev)
36
37     # forward pass
38     for t in xrange(len(inputs)):
39         x = inputs[t]
40         x1t = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
41         x1t[inputs[t]] = 1
42         whx = np.dot(Wxh, x1t) + bh # hidden state
43         why = np.dot(Why, hprev) + by # unnormalized log probabilities for next chars
44         ps[t] = np.exp(why) / np.sum(np.exp(why)) # softmax (cross-entropy loss)
45         loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)
46         a = np.tanh(whx + why) # hidden state
47         da = np.zeros_like(a)
48         da -= np.dot(Whh, a) # backprop through tanh nonlinearity
49         dprev = np.dot(Whh.T, da) # backprop into previous hidden state
50         dprev += np.zeros_like(dprev) # in case previous hidden state is None
51         for dparam in [dwhx, dhhy, da, dby]:
52             if dparam is not None:
53                 np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
54         return loss, da, dprev, dwhx, dhhy, dbh, dby, hs[-1]
55
56     # smooth loss so it's better for first time step
57     v = np.zeros(vocab_size, 1)
58     v[seed_ix] = 1
59
60     for t in xrange(n_time_steps):
61         x = np.zeros((vocab_size, 1))
62         x[seed_ix] = 1
63         whx = np.dot(Wxh, x) + bh
64         why = np.dot(Why, hprev) + by
65         ps[t] = np.exp(why) / np.sum(np.exp(why))
66         loss += -np.log(ps[t][targets[t], 0])
67         a = np.tanh(whx + why)
68         da = np.zeros_like(a)
69         da -= np.dot(Whh, a)
70         dprev = np.dot(Whh.T, da)
71         dprev += np.zeros_like(dprev)
72         for dparam in [dwhx, dhhy, da, dby]:
73             if dparam is not None:
74                 np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
75
76     # sample a sequence from the model
77     sample_ix = sample(hprev, inputs[0], 200)
78     txt = ''.join(ix_to_char[i] for i in sample_ix)
79
80     # forward n_time_steps characters through the net and fetch gradient
81     loss, da, dprev, dbh, dby, hprev = forward(inputs, targets, hprev)
82     for t in xrange(n_time_steps):
83         if t % 100 == 0: print 't %d, loss: %f' % (t, smooth_loss); print progress
84
85     # perform parameter update with Adam
86     for param, update, m, v in zip([dwhx, dhhy, dbh, dby],
87                                   [dwhx, dhhy, dbh, dby],
88                                   [m_w, m_h, m_b, m_y],
89                                   [v_w, v_h, v_b, v_y]):
90         m += update
91         v += update * update
92         update = m / (np.sqrt(v) + 1e-8) # integrated update
93
94     # move data pointer
95
96

```

## Loss function

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

```

27 def lossFun(inputs, targets, hprev):
28
29     """"
30     inputs,targets are both list of integers.
31     hprev is Hx1 array of initial hidden state
32     returns the loss, gradients on model parameters, and last hidden state
33     """
34
35     xs, hs, ys, ps = {}, {}, {}, {}
36     hs[-1] = np.copy(hprev)
37     loss = 0
38
39     # forward pass
40     for t in xrange(len(inputs)):
41         xs[t] = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
42         xs[t][inputs[t]] = 1
43
44         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
45         ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
46         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
47         loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)
48
49     # backward pass: compute gradients going backwards
50     dwhx, dwhh, dwhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
51     dbh, dby = np.zeros_like(bh), np.zeros_like(by)
52     dhnext = np.zeros_like(hs[0])
53
54     for t in reversed(xrange(len(inputs))):
55         dy = np.copy(ps[t])
56         dy[targets[t]] -= 1 # backprop into y
57         dyw = np.dot(dy, hs[t].T)
58         dwhy += np.dot(dy, hs[t-1].T)
59         dby += dy
60         dhw = (1 - hs[t]**2) * hs[t] * dy # backprop through tanh nonlinearity
61         dwhh += np.dot(dhw, hs[t-1].T)
62         dwhx += np.dot(dhw, xs[t].T)
63         dhnext = np.dot(Whh.T, dhnext)
64
65     # perform parameter update with Adam
66     for param, update, m, v in zip([dwhx, dwhh, dwhy, dbh, dby],
67                                   [dwhx, dwhh, dwhy, dbh, dby],
68                                   [m_w, m_h, m_b, m_y],
69                                   [v_w, v_h, v_b, v_y]):
70         m += update
71         v += update * update
72         update = m / (np.sqrt(v) + 1e-8) # integrated update
73
74     # clip to mitigate exploding gradients
75     return loss, dwhx, dwhh, dwhy, dbh, dby, hs[-1]

```



# RNN Python

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

```

1. Minimal character-level Vanilla RNN model, written by Andrej Karpathy (@karpathy)
2. MIT License
3.
4. Import numpy as np
5.
6. n = 10
7. f = open("input.txt", "r").read() # should be simple plain text file
8. chars = list(set(f))
9. print("data has %d characters, %d unique" % (len(data), len(chars)))
10. char_to_ix = {ch:i for i, ch in enumerate(chars)}
11. ix_to_char = {i:ch for ch, i in enumerate(chars)}
12.
13.
14. hidden_size = 100 # size of hidden layer of neurons
15. seq_length = 25 # number of steps to unroll the RNN for
16. learning_rate = 1e-1
17.
18.
19. wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
20. bh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
21. why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
22. by = np.zeros((vocab_size, 1)) # hidden bias
23. by[0] = 1.0 # initial hidden state
24.
25.
26. hidden_size = 100 # size of hidden layer of neurons
27. seq_length = 25 # number of steps to unroll the RNN for
28. learning_rate = 1e-1
29.
30.
31. def lossFun(inputs, targets, hprev):
32.     """
33.     inputs, targets are both list of integers.
34.     hprev is Hx1 array of initial hidden state
35.     returns the loss, gradients on model parameters, and last hidden state
36.     """
37.
38.     xs, hs, ys, ps = {}, {}, {}, {}
39.     hs[-1] = np.copy(hprev)
40.     loss = 0
41.     # forward pass
42.     for t in xrange(len(inputs)):
43.         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
44.         xs[t][inputs[t]] = 1
45.         hs[t] = np.tanh(np.dot(wxh, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
46.         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
47.         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
48.         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
49.
50.         # backward pass: compute gradients going backwards
51.         dws, dwh, dbh, dby, dxh, dwhh, dbyh, dbyw, dbyb = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(bh), np.zeros_like(by), np.zeros_like(xs[0]), np.zeros_like(xs[1]), np.zeros_like(xs[2]), np.zeros_like(xs[3]), np.zeros_like(xs[4])
52.         dy = np.zeros_like(ys[t])
53.         for t in reversed(xrange(len(inputs))):
54.             dy = np.copy(dyh)
55.             dyh = np.dot(why.T, dy) # dhnext = dy backprop into y
56.             dyh *= np.dot(whyh, hs[t-1]) # dyh = dyh * hidden state
57.             dyh *= np.dot(whyh.T, dyh) # dh = dyh * tanh nonlinearity
58.             dh = np.dot(dh, dyh) # dh = dh * previous dh
59.             dh -= np.dot(dxh, xs[t].T) # dh = dh - dxh * x
60.             dh -= np.dot(dwhh, hs[t-1].T) # dh = dh - dwhh * hs[t-1]
61.             dh -= np.dot(dbyh, by.T) # dh = dh - dbyh * by
62.             dxh = np.dot(wxh.T, dh) # dxh = dh * w
63.             dws += np.outer(dxh, xs[t]) # accumulate gradients for w
64.             dwhh += np.outer(dh, hs[t-1]) # accumulate gradients for whh
65.             dbh += dh # accumulate gradients for bh
66.             dbyh += dh # accumulate gradients for byh
67.             dby += dh # accumulate gradients for by
68.
69.             if t > 0:
70.                 dxh = np.zeros_like(xs[t])
71.                 dws += np.outer(dxh, xs[t-1])
72.                 dwhh += np.outer(dh, hs[t-2])
73.                 dbh += dh
74.                 dbyh += dh
75.                 dby += dh
76.
77.             if t == 0:
78.                 dxh = np.zeros_like(xs[t])
79.                 dws += np.outer(dxh, xs[t])
80.                 dwhh += np.outer(dh, hs[t-1])
81.                 dbh += dh
82.                 dbyh += dh
83.                 dby += dh
84.
85.             if t < seq_length-1:
86.                 dxh = np.zeros_like(xs[t])
87.                 dws += np.outer(dxh, xs[t])
88.                 dwhh += np.outer(dh, hs[t-1])
89.                 dbh += dh
90.                 dbyh += dh
91.                 dby += dh
92.
93.             if t == seq_length-1:
94.                 dxh = np.zeros_like(xs[t])
95.                 dws += np.outer(dxh, xs[t])
96.                 dwhh += np.outer(dh, hs[t-1])
97.                 dbh += dh
98.                 dbyh += dh
99.                 dby += dh
100.
101.             if t < seq_length-1:
102.                 dxh = np.zeros_like(xs[t])
103.                 dws += np.outer(dxh, xs[t])
104.                 dwhh += np.outer(dh, hs[t-1])
105.                 dbh += dh
106.                 dbyh += dh
107.                 dby += dh
108.
109.             if t < seq_length-1:
110.                 dxh = np.zeros_like(xs[t])
111.                 dws += np.outer(dxh, xs[t])
112.                 dwhh += np.outer(dh, hs[t-1])
113.                 dbh += dh
114.                 dbyh += dh
115.                 dby += dh
116.
117.             if t < seq_length-1:
118.                 dxh = np.zeros_like(xs[t])
119.                 dws += np.outer(dxh, xs[t])
120.                 dwhh += np.outer(dh, hs[t-1])
121.                 dbh += dh
122.                 dbyh += dh
123.                 dby += dh
124.
125.             if t < seq_length-1:
126.                 dxh = np.zeros_like(xs[t])
127.                 dws += np.outer(dxh, xs[t])
128.                 dwhh += np.outer(dh, hs[t-1])
129.                 dbh += dh
130.                 dbyh += dh
131.                 dby += dh
132.
133.             if t < seq_length-1:
134.                 dxh = np.zeros_like(xs[t])
135.                 dws += np.outer(dxh, xs[t])
136.                 dwhh += np.outer(dh, hs[t-1])
137.                 dbh += dh
138.                 dbyh += dh
139.                 dby += dh
140.
141.             if t < seq_length-1:
142.                 dxh = np.zeros_like(xs[t])
143.                 dws += np.outer(dxh, xs[t])
144.                 dwhh += np.outer(dh, hs[t-1])
145.                 dbh += dh
146.                 dbyh += dh
147.                 dby += dh
148.
149.             if t < seq_length-1:
150.                 dxh = np.zeros_like(xs[t])
151.                 dws += np.outer(dxh, xs[t])
152.                 dwhh += np.outer(dh, hs[t-1])
153.                 dbh += dh
154.                 dbyh += dh
155.                 dby += dh
156.
157.             if t < seq_length-1:
158.                 dxh = np.zeros_like(xs[t])
159.                 dws += np.outer(dxh, xs[t])
160.                 dwhh += np.outer(dh, hs[t-1])
161.                 dbh += dh
162.                 dbyh += dh
163.                 dby += dh
164.
165.             if t < seq_length-1:
166.                 dxh = np.zeros_like(xs[t])
167.                 dws += np.outer(dxh, xs[t])
168.                 dwhh += np.outer(dh, hs[t-1])
169.                 dbh += dh
170.                 dbyh += dh
171.                 dby += dh
172.
173.             if t < seq_length-1:
174.                 dxh = np.zeros_like(xs[t])
175.                 dws += np.outer(dxh, xs[t])
176.                 dwhh += np.outer(dh, hs[t-1])
177.                 dbh += dh
178.                 dbyh += dh
179.                 dby += dh
180.
181.             if t < seq_length-1:
182.                 dxh = np.zeros_like(xs[t])
183.                 dws += np.outer(dxh, xs[t])
184.                 dwhh += np.outer(dh, hs[t-1])
185.                 dbh += dh
186.                 dbyh += dh
187.                 dby += dh
188.
189.             if t < seq_length-1:
190.                 dxh = np.zeros_like(xs[t])
191.                 dws += np.outer(dxh, xs[t])
192.                 dwhh += np.outer(dh, hs[t-1])
193.                 dbh += dh
194.                 dbyh += dh
195.                 dby += dh
196.
197.             if t < seq_length-1:
198.                 dxh = np.zeros_like(xs[t])
199.                 dws += np.outer(dxh, xs[t])
200.                 dwhh += np.outer(dh, hs[t-1])
201.                 dbh += dh
202.                 dbyh += dh
203.                 dby += dh
204.
205.             if t < seq_length-1:
206.                 dxh = np.zeros_like(xs[t])
207.                 dws += np.outer(dxh, xs[t])
208.                 dwhh += np.outer(dh, hs[t-1])
209.                 dbh += dh
210.                 dbyh += dh
211.                 dby += dh
212.
213.             if t < seq_length-1:
214.                 dxh = np.zeros_like(xs[t])
215.                 dws += np.outer(dxh, xs[t])
216.                 dwhh += np.outer(dh, hs[t-1])
217.                 dbh += dh
218.                 dbyh += dh
219.                 dby += dh
220.
221.             if t < seq_length-1:
222.                 dxh = np.zeros_like(xs[t])
223.                 dws += np.outer(dxh, xs[t])
224.                 dwhh += np.outer(dh, hs[t-1])
225.                 dbh += dh
226.                 dbyh += dh
227.                 dby += dh
228.
229.             if t < seq_length-1:
230.                 dxh = np.zeros_like(xs[t])
231.                 dws += np.outer(dxh, xs[t])
232.                 dwhh += np.outer(dh, hs[t-1])
233.                 dbh += dh
234.                 dbyh += dh
235.                 dby += dh
236.
237.             if t < seq_length-1:
238.                 dxh = np.zeros_like(xs[t])
239.                 dws += np.outer(dxh, xs[t])
240.                 dwhh += np.outer(dh, hs[t-1])
241.                 dbh += dh
242.                 dbyh += dh
243.                 dby += dh
244.
245.             if t < seq_length-1:
246.                 dxh = np.zeros_like(xs[t])
247.                 dws += np.outer(dxh, xs[t])
248.                 dwhh += np.outer(dh, hs[t-1])
249.                 dbh += dh
250.                 dbyh += dh
251.                 dby += dh
252.
253.             if t < seq_length-1:
254.                 dxh = np.zeros_like(xs[t])
255.                 dws += np.outer(dxh, xs[t])
256.                 dwhh += np.outer(dh, hs[t-1])
257.                 dbh += dh
258.                 dbyh += dh
259.                 dby += dh
260.
261.             if t < seq_length-1:
262.                 dxh = np.zeros_like(xs[t])
263.                 dws += np.outer(dxh, xs[t])
264.                 dwhh += np.outer(dh, hs[t-1])
265.                 dbh += dh
266.                 dbyh += dh
267.                 dby += dh
268.
269.             if t < seq_length-1:
270.                 dxh = np.zeros_like(xs[t])
271.                 dws += np.outer(dxh, xs[t])
272.                 dwhh += np.outer(dh, hs[t-1])
273.                 dbh += dh
274.                 dbyh += dh
275.                 dby += dh
276.
277.             if t < seq_length-1:
278.                 dxh = np.zeros_like(xs[t])
279.                 dws += np.outer(dxh, xs[t])
280.                 dwhh += np.outer(dh, hs[t-1])
281.                 dbh += dh
282.                 dbyh += dh
283.                 dby += dh
284.
285.             if t < seq_length-1:
286.                 dxh = np.zeros_like(xs[t])
287.                 dws += np.outer(dxh, xs[t])
288.                 dwhh += np.outer(dh, hs[t-1])
289.                 dbh += dh
290.                 dbyh += dh
291.                 dby += dh
292.
293.             if t < seq_length-1:
294.                 dxh = np.zeros_like(xs[t])
295.                 dws += np.outer(dxh, xs[t])
296.                 dwhh += np.outer(dh, hs[t-1])
297.                 dbh += dh
298.                 dbyh += dh
299.                 dby += dh
300.
301.             if t < seq_length-1:
302.                 dxh = np.zeros_like(xs[t])
303.                 dws += np.outer(dxh, xs[t])
304.                 dwhh += np.outer(dh, hs[t-1])
305.                 dbh += dh
306.                 dbyh += dh
307.                 dby += dh
308.
309.             if t < seq_length-1:
310.                 dxh = np.zeros_like(xs[t])
311.                 dws += np.outer(dxh, xs[t])
312.                 dwhh += np.outer(dh, hs[t-1])
313.                 dbh += dh
314.                 dbyh += dh
315.                 dby += dh
316.
317.             if t < seq_length-1:
318.                 dxh = np.zeros_like(xs[t])
319.                 dws += np.outer(dxh, xs[t])
320.                 dwhh += np.outer(dh, hs[t-1])
321.                 dbh += dh
322.                 dbyh += dh
323.                 dby += dh
324.
325.             if t < seq_length-1:
326.                 dxh = np.zeros_like(xs[t])
327.                 dws += np.outer(dxh, xs[t])
328.                 dwhh += np.outer(dh, hs[t-1])
329.                 dbh += dh
330.                 dbyh += dh
331.                 dby += dh
332.
333.             if t < seq_length-1:
334.                 dxh = np.zeros_like(xs[t])
335.                 dws += np.outer(dxh, xs[t])
336.                 dwhh += np.outer(dh, hs[t-1])
337.                 dbh += dh
338.                 dbyh += dh
339.                 dby += dh
340.
341.             if t < seq_length-1:
342.                 dxh = np.zeros_like(xs[t])
343.                 dws += np.outer(dxh, xs[t])
344.                 dwhh += np.outer(dh, hs[t-1])
345.                 dbh += dh
346.                 dbyh += dh
347.                 dby += dh
348.
349.             if t < seq_length-1:
350.                 dxh = np.zeros_like(xs[t])
351.                 dws += np.outer(dxh, xs[t])
352.                 dwhh += np.outer(dh, hs[t-1])
353.                 dbh += dh
354.                 dbyh += dh
355.                 dby += dh
356.
357.             if t < seq_length-1:
358.                 dxh = np.zeros_like(xs[t])
359.                 dws += np.outer(dxh, xs[t])
360.                 dwhh += np.outer(dh, hs[t-1])
361.                 dbh += dh
362.                 dbyh += dh
363.                 dby += dh
364.
365.             if t < seq_length-1:
366.                 dxh = np.zeros_like(xs[t])
367.                 dws += np.outer(dxh, xs[t])
368.                 dwhh += np.outer(dh, hs[t-1])
369.                 dbh += dh
370.                 dbyh += dh
371.                 dby += dh
372.
373.             if t < seq_length-1:
374.                 dxh = np.zeros_like(xs[t])
375.                 dws += np.outer(dxh, xs[t])
376.                 dwhh += np.outer(dh, hs[t-1])
377.                 dbh += dh
378.                 dbyh += dh
379.                 dby += dh
380.
381.             if t < seq_length-1:
382.                 dxh = np.zeros_like(xs[t])
383.                 dws += np.outer(dxh, xs[t])
384.                 dwhh += np.outer(dh, hs[t-1])
385.                 dbh += dh
386.                 dbyh += dh
387.                 dby += dh
388.
389.             if t < seq_length-1:
390.                 dxh = np.zeros_like(xs[t])
391.                 dws += np.outer(dxh, xs[t])
392.                 dwhh += np.outer(dh, hs[t-1])
393.                 dbh += dh
394.                 dbyh += dh
395.                 dby += dh
396.
397.             if t < seq_length-1:
398.                 dxh = np.zeros_like(xs[t])
399.                 dws += np.outer(dxh, xs[t])
400.                 dwhh += np.outer(dh, hs[t-1])
401.                 dbh += dh
402.                 dbyh += dh
403.                 dby += dh
404.
405.             if t < seq_length-1:
406.                 dxh = np.zeros_like(xs[t])
407.                 dws += np.outer(dxh, xs[t])
408.                 dwhh += np.outer(dh, hs[t-1])
409.                 dbh += dh
410.                 dbyh += dh
411.                 dby += dh
412.
413.             if t < seq_length-1:
414.                 dxh = np.zeros_like(xs[t])
415.                 dws += np.outer(dxh, xs[t])
416.                 dwhh += np.outer(dh, hs[t-1])
417.                 dbh += dh
418.                 dbyh += dh
419.                 dby += dh
420.
421.             if t < seq_length-1:
422.                 dxh = np.zeros_like(xs[t])
423.                 dws += np.outer(dxh, xs[t])
424.                 dwhh += np.outer(dh, hs[t-1])
425.                 dbh += dh
426.                 dbyh += dh
427.                 dby += dh
428.
429.             if t < seq_length-1:
430.                 dxh = np.zeros_like(xs[t])
431.                 dws += np.outer(dxh, xs[t])
432.                 dwhh += np.outer(dh, hs[t-1])
433.                 dbh += dh
434.                 dbyh += dh
435.                 dby += dh
436.
437.             if t < seq_length-1:
438.                 dxh = np.zeros_like(xs[t])
439.                 dws += np.outer(dxh, xs[t])
440.                 dwhh += np.outer(dh, hs[t-1])
441.                 dbh += dh
442.                 dbyh += dh
443.                 dby += dh
444.
445.             if t < seq_length-1:
446.                 dxh = np.zeros_like(xs[t])
447.                 dws += np.outer(dxh, xs[t])
448.                 dwhh += np.outer(dh, hs[t-1])
449.                 dbh += dh
450.                 dbyh += dh
451.                 dby += dh
452.
453.             if t < seq_length-1:
454.                 dxh = np.zeros_like(xs[t])
455.                 dws += np.outer(dxh, xs[t])
456.                 dwhh += np.outer(dh, hs[t-1])
457.                 dbh += dh
458.                 dbyh += dh
459.                 dby += dh
460.
461.             if t < seq_length-1:
462.                 dxh = np.zeros_like(xs[t])
463.                 dws += np.outer(dxh, xs[t])
464.                 dwhh += np.outer(dh, hs[t-1])
465.                 dbh += dh
466.                 dbyh += dh
467.                 dby += dh
468.
469.             if t < seq_length-1:
470.                 dxh = np.zeros_like(xs[t])
471.                 dws += np.outer(dxh, xs[t])
472.                 dwhh += np.outer(dh, hs[t-1])
473.                 dbh += dh
474.                 dbyh += dh
475.                 dby += dh
476.
477.             if t < seq_length-1:
478.                 dxh = np.zeros_like(xs[t])
479.                 dws += np.outer(dxh, xs[t])
480.                 dwhh += np.outer(dh, hs[t-1])
481.                 dbh += dh
482.                 dbyh += dh
483.                 dby += dh
484.
485.             if t < seq_length-1:
486.                 dxh = np.zeros_like(xs[t])
487.                 dws += np.outer(dxh, xs[t])
488.                 dwhh += np.outer(dh, hs[t-1])
489.                 dbh += dh
490.                 dbyh += dh
491.                 dby += dh
492.
493.             if t < seq_length-1:
494.                 dxh = np.zeros_like(xs[t])
495.                 dws += np.outer(dxh, xs[t])
496.                 dwhh += np.outer(dh, hs[t-1])
497.                 dbh += dh
498.                 dbyh += dh
499.                 dby += dh
500.
501.             if t < seq_length-1:
502.                 dxh = np.zeros_like(xs[t])
503.                 dws += np.outer(dxh, xs[t])
504.                 dwhh += np.outer(dh, hs[t-1])
505.                 dbh += dh
506.                 dbyh += dh
507.                 dby += dh
508.
509.             if t < seq_length-1:
510.                 dxh = np.zeros_like(xs[t])
511.                 dws += np.outer(dxh, xs[t])
512.                 dwhh += np.outer(dh, hs[t-1])
513.                 dbh += dh
514.                 dbyh += dh
515.                 dby += dh
516.
517.             if t < seq_length-1:
518.                 dxh = np.zeros_like(xs[t])
519.                 dws += np.outer(dxh, xs[t])
520.                 dwhh += np.outer(dh, hs[t-1])
521.                 dbh += dh
522.                 dbyh += dh
523.                 dby += dh
524.
525.             if t < seq_length-1:
526.                 dxh = np.zeros_like(xs[t])
527.                 dws += np.outer(dxh, xs[t])
528.                 dwhh += np.outer(dh, hs[t-1])
529.                 dbh += dh
530.                 dbyh += dh
531.                 dby += dh
532.
533.             if t < seq_length-1:
534.                 dxh = np.zeros_like(xs[t])
535.                 dws += np.outer(dxh, xs[t])
536.                 dwhh += np.outer(dh, hs[t-1])
537.                 dbh += dh
538.                 dbyh += dh
539.                 dby += dh
540.
541.             if t < seq_length-1:
542.                 dxh = np.zeros_like(xs[t])
543.                 dws += np.outer(dxh, xs[t])
544.                 dwhh += np.outer(dh, hs[t-1])
545.                 dbh += dh
546.                 dbyh += dh
547.                 dby += dh
548.
549.             if t < seq_length-1:
550.                 dxh = np.zeros_like(xs[t])
551.                 dws += np.outer(dxh, xs[t])
552.                 dwhh += np.outer(dh, hs[t-1])
553.                 dbh += dh
554.                 dbyh += dh
555.                 dby += dh
556.
557.             if t < seq_length-1:
558.                 dxh = np.zeros_like(xs[t])
559.                 dws += np.outer(dxh, xs[t])
560.                 dwhh += np.outer(dh, hs[t-1])
561.                 dbh += dh
562.                 dbyh += dh
563.                 dby += dh
564.
565.             if t < seq_length-1:
566.                 dxh = np.zeros_like(xs[t])
567.                 dws += np.outer(dxh, xs[t])
568.                 dwhh += np.outer(dh, hs[t-1])
569.                 dbh += dh
570.                 dbyh += dh
571.                 dby += dh
572.
573.             if t < seq_length-1:
574.                 dxh = np.zeros_like(xs[t])
575.                 dws += np.outer(dxh, xs[t])
576.                 dwhh += np.outer(dh, hs[t-1])
577.                 dbh += dh
578.                 dbyh += dh
579.                 dby += dh
580.
581.             if t < seq_length-1:
582.                 dxh = np.zeros_like(xs[t])
583.                 dws += np.outer(dxh, xs[t])
584.                 dwhh += np.outer(dh, hs[t-1])
585.                 dbh += dh
586.                 dbyh += dh
587.                 dby += dh
588.
589.             if t < seq_length-1:
590.                 dxh = np.zeros_like(xs[t])
591.                 dws += np.outer(dxh, xs[t])
592.                 dwhh += np.outer(dh, hs[t-1])
593.                 dbh += dh
594.                 dbyh += dh
595.                 dby += dh
596.
597.             if t < seq_length-1:
598.                 dxh = np.zeros_like(xs[t])
599.                 dws += np.outer(dxh, xs[t])
600.                 dwhh += np.outer(dh, hs[t-1])
601.                 dbh += dh
602.                 dbyh += dh
603.                 dby += dh
604.
605.             if t < seq_length-1:
606.                 dxh = np.zeros_like(xs[t])
607.                 dws += np.outer(dxh, xs[t])
608.                 dwhh += np.outer(dh, hs[t-1])
609.                 dbh += dh
610.                 dbyh += dh
611.                 dby += dh
612.
613.             if t < seq_length-1:
614.                 dxh = np.zeros_like(xs[t])
615.                 dws += np.outer(dxh, xs[t])
616.                 dwhh += np.outer(dh, hs[t-1])
617.                 dbh += dh
618.                 dbyh += dh
619.                 dby += dh
620.
621.             if t < seq_length-1:
622.                 dxh = np.zeros_like(xs[t])
623.                 dws += np.outer(dxh, xs[t])
624.                 dwhh += np.outer(dh, hs[t-1])
625.                 dbh += dh
626.                 dbyh += dh
627.                 dby += dh
628.
629.             if t < seq_length-1:
630.                 dxh = np.zeros_like(xs[t])
631.                 dws += np.outer(dxh, xs[t])
632.                 dwhh += np.outer(dh, hs[t-1])
633.                 dbh += dh
634.                 dbyh += dh
635.                 dby += dh
636.
637.             if t < seq_length-1:
638.                 dxh = np.zeros_like(xs[t])
639.                 dws += np.outer(dxh, xs[t])
640.                 dwhh += np.outer(dh, hs[t-1])
641.                 dbh += dh
642.                 dbyh += dh
643.                 dby += dh
644.
645.             if t < seq_length-1:
646.                 dxh = np.zeros_like(xs[t])
647.                 dws += np.outer(dxh, xs[t])
648.                 dwhh += np.outer(dh, hs[t-1])
649.                 dbh += dh
650.                 dbyh += dh
651.                 dby += dh
652.
653.             if t < seq_length-1:
654.                 dxh = np.zeros_like(xs[t])
655.                 dws += np.outer(dxh, xs[t])
656.                 dwhh += np.outer(dh, hs[t-1])
657.                 dbh += dh
658.                 dbyh += dh
659.                 dby += dh
660.
661.             if t < seq_length-1:
662.                 dxh = np.zeros_like(xs[t])
663.                 dws += np.outer(dxh, xs[t])
664.                 dwhh += np.outer(dh, hs[t-1])
665.                 dbh += dh
666.                 dbyh += dh
667.                 dby += dh
668.
669.             if t < seq_length-1:
670.                 dxh = np.zeros_like(xs[t])
671.                 dws += np.outer(dxh, xs[t])
672.                 dwhh += np.outer(dh, hs[t-1])
673.                 dbh += dh
674.                 dbyh += dh
675.                 dby += dh
676.
677.             if t < seq_length-1:
678.                 dxh = np.zeros_like(xs[t])
679.                 dws += np.outer(dxh, xs[t])
680.                 dwhh += np.outer(dh, hs[t-1])
681.                 dbh += dh
682.                 dbyh += dh
683.                 dby += dh
684.
685.             if t < seq_length-1:
686.                 dxh = np.zeros_like(xs[t])
687.                 dws += np.outer(dxh, xs[t])
688.                 dwhh += np.outer(dh, hs[t-1])
689.                 dbh += dh
690.                 dbyh += dh
691.                 dby += dh
692.
693.             if t < seq_length-1:
694.                 dxh = np.zeros_like(xs[t])
695.                 dws += np.outer(dxh, xs[t])
696.                 dwhh += np.outer(dh, hs[t-1])
697.                 dbh += dh
698.                 dbyh += dh
699.                 dby += dh
700.
701.             if t < seq_length-1:
702.                 dxh = np.zeros_like(xs[t])
703.                 dws += np.outer(dxh, xs[t])
704.                 dwhh += np.outer(dh, hs[t-1])
705.                 dbh += dh
706.                 dbyh += dh
707.                 dby += dh
708.
709.             if t < seq_length-1:
710.                 dxh = np.zeros_like(xs[t])
711.                 dws += np.outer(dxh, xs[t])
712.                 dwhh += np.outer(dh, hs[t-1])
713.                 dbh += dh
714.                 dbyh += dh
715.                 dby += dh
716.
717.             if t < seq_length-1:
718.                 dxh = np.zeros_like(xs[t])
719.                 dws += np.outer(dxh, xs[t])
720.                 dwhh += np.outer(dh, hs[t-1])
721.                 dbh += dh
722.                 dbyh += dh
723.                 dby += dh
724.
725.             if t < seq_length-1:
726.                 dxh = np.zeros_like(xs[t])
727.                 dws += np.outer(dxh, xs[t])
728.                 dwhh += np.outer(dh, hs[t-1])
729.                 dbh += dh
730.                 dbyh += dh
731.                 dby += dh
732.
733.             if t < seq_length-1:
734.                 dxh = np.zeros_like(xs[t])
735.                 dws += np.outer(dxh, xs[t])
736.                 dwhh += np.outer(dh, hs[t-1])
737.                 dbh += dh
738.                 dbyh += dh
739.                 dby += dh
740.
741.             if t < seq_length-1:
742.                 dxh = np.zeros_like(xs[t])
743.                 dws += np.outer(dxh, xs[t])
744.                 dwhh += np.outer(dh, hs[t-1])
745.                 dbh += dh
746.                 dbyh += dh
747.                 dby += dh
748.
749.             if t < seq_length-1:
750.                 dxh = np.zeros_like(xs[t])
751.                 dws += np.outer(dxh, xs[t])
752.                 dwhh += np.outer(dh, hs[t-1])
753.                 dbh += dh
754.                 dbyh += dh
755.                 dby += dh
756.
757.             if t < seq_length-1:
758.                 dxh = np.zeros_like(xs[t])
759.                 dws += np.outer(dxh, xs[t])
760.                 dwhh += np.outer(dh, hs[t-1])
761.                 dbh += dh
762.                 dbyh += dh
763.                 dby += dh
764.
765.             if t < seq_length-1:
766.                 dxh = np.zeros_like(xs[t])
767.                 dws += np.outer(dxh, xs[t])
768.                 dwhh += np.outer(dh, hs[t-1])
769.                 dbh += dh
770.                 dbyh += dh
771.                 dby += dh
772.
773.             if t < seq_length-1:
774.                 dxh = np.zeros_like(xs[t])
775.                 dws += np.outer(dxh, xs[t])
776.                 dwhh += np.outer(dh, hs[t-1])
777.                 dbh += dh
778.                 dbyh += dh
779.                 dby += dh
780.
781.             if t < seq_length-1:
782.                 dxh = np.zeros_like(xs[t])
783.                 dws += np.outer(dxh, xs[t])
784.                 dwhh += np.outer(dh, hs[t-1])
785.                 dbh += dh
786.                 dbyh += dh
787.                 dby += dh
788.
789.             if t < seq_length-1:
790.                 dxh = np.zeros_like(xs[t])
791.                 dws += np.outer(dxh, xs[t])
792.                 dwhh += np.outer(dh, hs[t-1])
793.                 dbh += dh
794.                 dbyh += dh
795.                 dby += dh
796.
797.             if t < seq_length-1:
798.                 dxh = np.zeros_like(xs[t])
799.                 dws += np.outer(dxh, xs[t])
800.                 dwhh += np.outer(dh, hs[t-1])
801.                 dbh += dh
802.                 dbyh += dh
803.                 dby += dh
804.
805.             if t < seq_length-1:
806.                 dxh = np.zeros_like(xs[t])
807.                 dws += np.outer(dxh, xs[t])
808.                 dwhh += np.outer(dh, hs[t-1])
809.                 dbh += dh
810.                 dbyh += dh
811.                 dby += dh
812.
813.             if t < seq_length-1:
814.                 dxh = np.zeros_like(xs[t])
815.                 dws += np.outer(dxh, xs[t])
816.                 dwhh += np.outer(dh, hs[t-1])
817.                 dbh += dh
818.                 dbyh += dh
819.                 dby += dh
820.
821.             if t < seq_length-1:
822.                 dxh = np.zeros_like(xs[t])
823.                 dws += np.outer(dxh, xs[t])
824.                 dwhh += np.outer(dh, hs[t-1])
825.                 dbh += dh
826.                 dbyh += dh
827.                 dby += dh
828.
829.             if t < seq_length-1:
830.                 dxh = np.zeros_like(xs[t])
831.                 dws += np.outer(dxh, xs[t])
832.                 dwhh += np.outer(dh, hs[t-1])
833.                 dbh += dh
834.                 dbyh += dh
835.                 dby += dh
836.
837.             if t < seq_length-1:
838.                 dxh = np.zeros_like(xs[t])
839.                 dws += np.outer(dxh, xs[t])
840.                 dwhh += np.outer(dh, hs[t-1])
841.                 dbh += dh
842.                 dbyh += dh
843.                 dby += dh
844.
845.             if t < seq_length-1:
846.                 dxh = np.zeros_like(xs[t])
847.                 dws += np.outer(dxh, xs[t])
848.                 dwhh += np.outer(dh, hs[t-1])
849.                 dbh += dh
850.                 dbyh += dh
851.                 dby += dh
852.
853.             if t < seq_length-1:
854.                 dxh = np.zeros_like(xs[t])
855.                 dws += np.outer(dxh, xs[t])
856.                 dwhh += np.outer(dh, hs[t-1])
857.                 dbh += dh
858.                 dbyh += dh
859.                 dby += dh
860.
861.             if t < seq_length-1:
862.                 dxh = np.zeros_like(xs[t])
863.                 dws += np.outer(dxh, xs[t])
864.                 dwhh += np.outer(dh, hs[t-1])
865.                 dbh += dh
866.                 dbyh += dh
867.                 dby += dh
868.
869.             if t < seq_length-1:
870.                 dxh = np.zeros_like(xs[t])
871.                 dws += np.outer(dxh, xs[t])
872.                 dwhh += np.outer(dh, hs[t-1])
873.                 dbh += dh
874.                 dbyh += dh
875.                 dby += dh
876.
877.             if t < seq_length-1:
878.                 dxh = np.zeros_like(xs[t])
879.                 dws += np.outer(dxh, xs[t])
880.                 dwhh += np.outer(dh, hs[t-1])
881.                 dbh += dh
882.                 dbyh += dh
883.                 dby += dh
884.
885.             if t < seq_length-1:
886.                 dxh = np.zeros_like(xs[t])
887.                 dws += np.outer(dxh, xs[t])
888.                 dwhh += np.outer(dh, hs[t-1])
889.                 dbh += dh
890.                 dbyh += dh
891.                 dby += dh
892.
893.             if t < seq_length-1:
894.                 dxh = np.zeros_like(xs[t])
895.                 dws += np.outer(dxh, xs[t])
896.                 dwhh += np.outer(dh, hs[t-1])
897.                 dbh += dh
898.                 dbyh += dh
899.                 dby += dh
900.
901.             if t < seq_length-1:
902.                 dxh = np.zeros_like(xs[t])
903.                 dws += np.outer(dxh, xs[t])
904.                 dwhh += np.outer(dh, hs[t-1])
905.                 dbh += dh
906.                 dbyh += dh
907.                 dby += dh
908.
909.             if t < seq_length-1:
910.                 dxh = np.zeros_like(xs[t])
911.                 dws += np.outer(dxh, xs[t])
912.                 dwhh += np.outer(dh, hs[t-1])
913.                 dbh += dh
914.                 dbyh += dh
915.                 dby += dh
916.
917.             if t < seq_length-1:
918.                 dxh = np.zeros_like(xs[t])
919.                 dws += np.outer(dxh, xs[t])
920.                 dwhh += np.outer(dh, hs[t-1])
921.                 dbh += dh
922.                 dbyh += dh
923.                 dby += dh
924.
925.             if t < seq_length-1:
926.                 dxh = np.zeros_like(xs[t])
927.                 dws += np.outer(dxh, xs[t])
928.                 dwhh += np.outer(dh, hs[t-1])
929.                 dbh += dh
930.                 dbyh += dh
931.                 dby += dh
932.
933.             if t < seq_length-1:
934.                 dxh = np.zeros_like(xs[t])
935.                 dws += np.outer(dxh, xs[t])
936.                 dwhh += np.outer(dh, hs[t-1])
937.                 dbh += dh
938.                 dbyh += dh
939.                 dby += dh
940.
941.             if t < seq_length-1:
942.                 dxh = np.zeros_like(xs[t])
943.                 dws += np.outer(dxh, xs[t])
944.                 dwhh += np.outer(dh, hs[t-1])
945.                 dbh += dh
946.                 dbyh += dh
947.                 dby += dh
948.
949.             if t < seq_length-1:
950.                 dxh = np.zeros_like(xs[t])
951.                 dws += np.outer(dxh, xs[t])
952.                 dwhh += np.outer(dh, hs[t-1])
953.                 dbh += dh
954.                 dbyh += dh
955.                 dby += dh
956.
957.             if t < seq_length-1:
958.                 dxh = np.zeros_like(xs[t])
959.                 dws += np.outer(dxh, xs[t])
960.                 dwhh += np.outer(dh, hs[t-1])
961.                 dbh += dh
962.                 dbyh += dh
963.                 dby += dh
964.
965.             if t < seq_length-1:
966.                 dxh = np.zeros_like(xs[t])
967.                 dws += np.outer(dxh, xs[t])
968.                 dwhh += np.outer(dh, hs[t-1])
969.                 dbh += dh
970.                 dbyh += dh
971.                 dby += dh
972.
973.             if t < seq_length-1:
9
```

# RNN Python

## min-char-rnn.py gist

```
1. Minimal character-level vanilla RNN model, written by Andrej Karpathy (@karpathy)
2. BSD License
3.
4. import numpy as np
5.
6. import time
7.
8. # data I/O
9. # Given a file (e.g., text.txt) read() should be similar plain text file
10. chars = list(open(data))
11. data_size, vocab_size = len(chars), len(chars)
12. print('data_size: %d, vocab_size: %d' % (data_size, vocab_size))
13. char_to_ix = { ch:i for i, ch in enumerate(chars) }
14. ix_to_char = { i:ch for i, ch in enumerate(chars) }
15.
16. # Hyperparameters
17. hidden_size = 100 # size of hidden layer of neurons
18. seq_length = 25 # number of steps to unroll the RNN for
19. learning_rate = 1e-1
20.
21. np.random.seed(1971) # random seed
22. wh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
23. bh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
24. why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
25. by = np.zeros((vocab_size, 1)) # output bias
26.
27. def loss(inputs, targets, hprev):
28.
29.     inputs, targets = both lists of integers.
30.     hprev is array of initial hidden state
31.     inputs[0] is seed letter, used on model parameters, and last hidden state
32.     inputs[i] = 0 <= inputs[i] <= len(chars)
33.     h, ys, xs = (0, 0, 0)
34.     hprev = np.concatenate([hprev])
35.     loss = 0
36.
37.     for t in xrange(len(inputs)):
38.         x = np.zeros((vocab_size, 1)) # encode in 1-of-K representation
39.         x[inputs[t]] = 1
40.         h[1:] = np.tanh(np.dot(wih, x) + np.dot(bih, h[1:]) + bh) # hidden state
41.         pih = np.exp(np.dot(why, h)) / np.sum(np.exp(pih)) # softmax for next chars
42.         ps = np.exp(targets[t] / np.sum(np.exp(xs))) # probabilities for next chars
43.         loss += -np.log(pihs[targets[t]]) # softmax (cross-entropy loss)
44.         e = np.exp(xs) # error term
45.         doeh, dwh, dby = np.zeros_like(pihs), np.zeros_like(wih), np.zeros_like(by)
46.         dwh += np.dot(x, pihs.T)
47.         dby += np.sum(pihs * e)
48.         ddoeh = np.zeros_like(pihs)
49.         for t2 in reversed(xrange(t+1, len(inputs))):
50.             dy = ddoeh * np.dot(why.T, e)
51.             dy[1:] = 0 # backprop into y
52.             dby += dy
53.             dh = -np.dot(why, T, dy) + dwh # backprop into h
54.             dwh += np.dot(dy, pihs.T) # backprop through tanh nonlinearity
55.             dwh += dwh # accumulate gradients
56.             ddoeh = np.dot(why, h[1:-1])
57.             dwh += np.dot(dy, h[1:-1].T)
58.             ddoeh += np.dot(why, T, dy)
59.             for l in range(2, len(inputs)-t-1):
60.                 dy = ddoeh * np.dot(why.T, dy)
61.                 dy[1:] = 0 # backprop into y
62.                 dby += dy
63.                 ddoeh = np.zeros_like(pihs)
64.
65.     # sample a sequence of integers from the model
66.     h = memory state, seed_ix is seed letter for first time step
67.     x = np.zeros((vocab_size, 1))
68.     x[seed_ix] = 1
69.     ixes = []
70.
71.     for t in xrange(n):
72.         h = np.tanh(np.dot(wih, x) + np.dot(bih, h) + bh)
73.         y = np.dot(why, h) + by
74.         p = np.exp(y) / np.sum(np.exp(y))
75.         ix = np.random.choice(range(vocab_size), p=p.ravel())
76.         x = np.zeros((vocab_size, 1))
77.         x[ix] = 1
78.         ixes.append(ix)
79.
80.     return ixes
81.
82. # sample from the model one step at a time
83. if __name__ == "__main__":
84.     # read in and precompute char-to-idx mapping from left to right in steps seq_length long
85.     fpr = open('text.txt')
86.     fpr.seek(0, 2)
87.     fpr.seek(-seq_length, 1)
88.     chars = fpr.read(seq_length)
89.     fpr.close()
90.     chars = [char_to_ix[ch] for ch in chars]
91.     inputs = [chars[i:i+seq_length] for i in range(len(chars)-seq_length)]
92.     targets = [chars[i+seq_length] for i in range(len(chars)-seq_length)]
93.
94.     # sample from the model one step at a time
95.     if __name__ == "__main__":
96.         # read in and precompute char-to-idx mapping from left to right in steps seq_length long
97.         fpr = open('text.txt')
98.         fpr.seek(0, 2)
99.         fpr.seek(-seq_length, 1)
100.        chars = fpr.read(seq_length)
101.        fpr.close()
102.        chars = [char_to_ix[ch] for ch in chars]
103.        print "...".join(chars[:N])
104.
105.        # setting parameter update with sigmoid
106.        for para in para:
107.            para -= learning_rate * para / np.sum(para + 1e-8) # sigmoid update
108.
109.    # more data points
110.    para += learning_rate * para / np.sum(para + 1e-8) # sigmoid update
111.
112.    # more data points
113.    para += learning_rate * para / np.sum(para + 1e-8) # sigmoid update
114.
```

```
def sample(h, seed_ix, n):
    """
    sample a sequence of integers from the model
    h is memory state, seed_ix is seed letter for first time step
    """
    x = np.zeros((vocab_size, 1))
    x[seed_ix] = 1
    ixes = []
    for t in xrange(n):
        h = np.tanh(np.dot(wih, x) + np.dot(bih, h) + bh)
        y = np.dot(why, h) + by
        p = np.exp(y) / np.sum(np.exp(y))
        ix = np.random.choice(range(vocab_size), p=p.ravel())
        x = np.zeros((vocab_size, 1))
        x[ix] = 1
        ixes.append(ix)
    return ixes
```

# Time step and Batch

- Time steps are defined by **sequences length**
- The batch size is defined by **user**

## Time major representation

- `[max_timestep * batch_size * dimension]`

## RNNs typically use time major representation

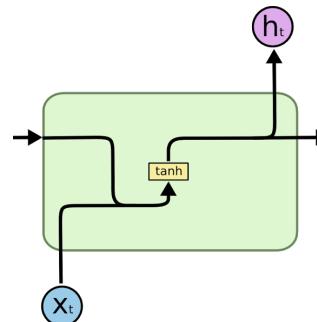
```
outputs = []
#Shape of sentences_in_batch [max_timestep * batch_size * dimension]
for words_in_batch in sentences_in_batch:
    # Shape of words_in_batch [batch_size * dimension]
    some operations here...
    outputs.append(i-th words_output)
#Shape of outputs [max_timestep * batch_size * dimension]
print outputs
```

# RNN Cells (Memory Units)

- RNN basic code

```
outputs = []
#Shape of sentences_in_batch [max_timestep * batch_size * dimension]
for words_in_batch in sentences_in_batch:
    # Shape of words_in_batch [batch_size * dimension]
    some operations here...
    outputs.append(i-th words_output)
#Shape of outputs [max_timestep * batch_size * dimension]
print outputs
```

- RNN Cell

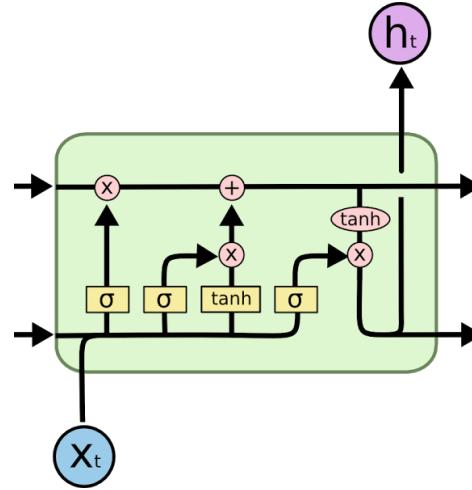


$$h_{t+1} = \tanh(x_t W + h_t U + b)$$

(Image credit: Colah's blog)

# RNN Cells (Memory Units)

- LSTM Cell



$$i = \sigma(x_t U^i + s_{t-1} W^i)$$

$$f = \sigma(x_t U^f + s_{t-1} W^f)$$

$$o = \sigma(x_t U^o + s_{t-1} W^o)$$

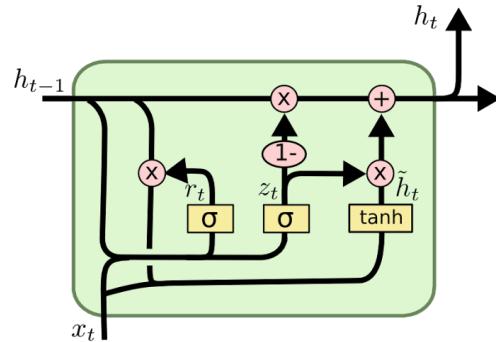
$$g = \tanh(x_t U^g + s_{t-1} W^g)$$

$$c_t = c_{t-1} \circ f + g \circ i$$

$$s_t = \tanh(c_t) \circ o$$

# RNN Cells (Memory Units)

- GRU Cell



$$z = \sigma(x_t U^z + s_{t-1} W^z)$$

$$r = \sigma(x_t U^r + s_{t-1} W^r)$$

$$h = \tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$

$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

# RNN Loss and Optimization

## Simple RNN

$$\begin{aligned}s_t &= \tanh(Ux_t + Ws_{t-1}) \\ \hat{y}_t &= \text{softmax}(Vs_t)\end{aligned}$$

## Cross Entropy Loss

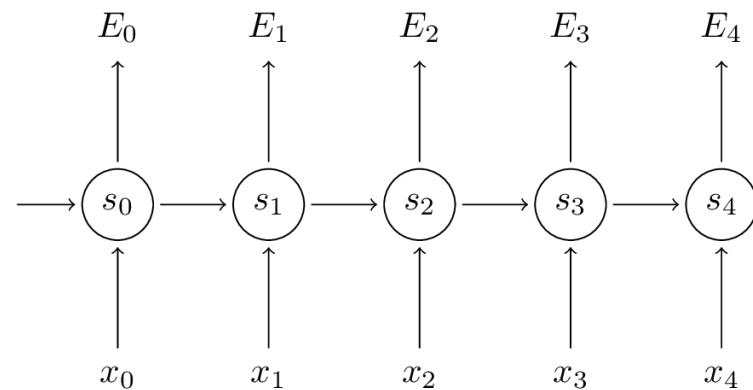
$$\begin{aligned}E(y_t, \hat{y}_t) &= -y_t \log \hat{y}_t \\ E(y, \hat{y}) &= -\sum_t E_t(y_t, \hat{y}_t) \\ &= -\sum_t -y_t \log \hat{y}_t\end{aligned}$$

# RNN Loss and Optimization

## BackProp

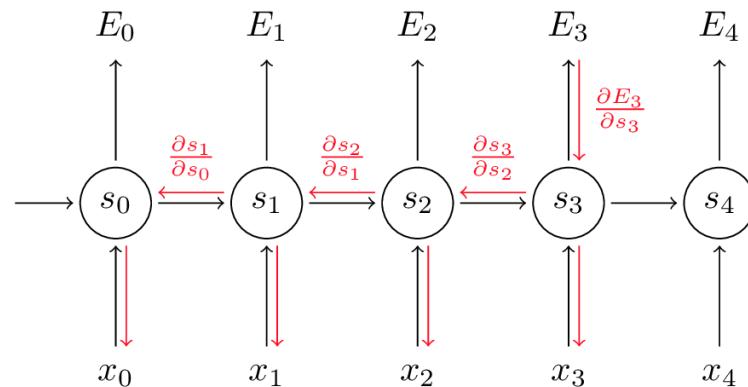
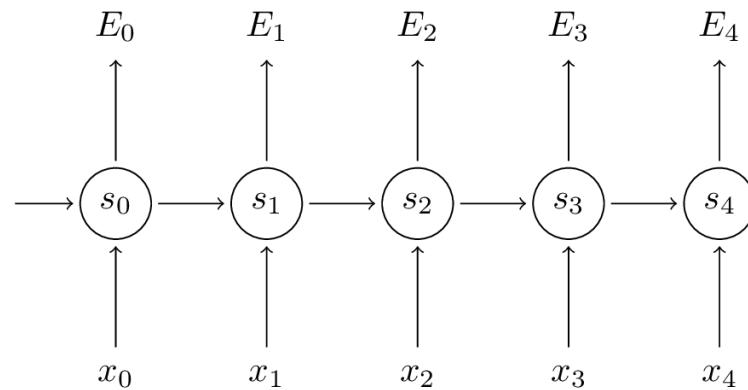
# RNN Loss and Optimization

## BackProp



# RNN Loss and Optimization

## BackProp



# Tensorflow Basics



# Tensorflow Basics

## (1) Fetch tensors via `Session.run()`

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
bias = tf.Variable(1.0)

y_pred = x ** 2 + bias      # x -> x^2 + bias
loss = (y - y_pred)**2     # l2 loss?

# Error: to compute loss, y is required as a dependency
print('Loss(x,y) = %.3f' % session.run(loss, {x: 3.0}))

# OK, print 1.000 = (3**2 + 1 - 9)**2
print('Loss(x,y) = %.3f' % session.run(loss, {x: 3.0, y: 9.0}))
```

# Tensorflow Basics

## (2) Basic operations

```
'''Permuting batch_size and n_steps'''
x = tf.transpose(x, [1, 0, 2])

'''Reshape to (n_steps*batch_size, n_input)'''
x = tf.reshape(x, [-1, n_input])

'''Split to get a list of 'n_steps' tensors of shape (batch_size, n_input)'''
x = tf.split(x, n_steps, 0)

'''Matrix multiplication and bias addition'''
y = tf.matmul(outputs[-1], weights['out']) + biases['out']

'''Data type casting'''
correct_pred = tf.cast(correct_pred, tf.float32)

'''Calculate mean'''
correct_pred = tf.reduce_mean(correct_pred)

'''Adam optimizer'''
tf.train.AdamOptimizer(learning_rate=learning_rate)
```

# Tensorflow Basics

## (3) Build, Train, Test

```
# Hyper Parameters  
learning_rate = 0.001
```

# Tensorflow Basics

## (3) Build, Train, Test

```
# Hyper Parameters  
learning_rate = 0.001
```

```
# Network Parameters  
n_input = 28 # MNIST data input (img shape: 28*28)
```

# Tensorflow Basics

## (3) Build, Train, Test

```
# Hyper Parameters  
learning_rate = 0.001
```

```
# Network Parameters  
n_input = 28 # MNIST data input (img shape: 28*28)
```

```
# tf Graph input  
x = tf.placeholder("float", [None, n_steps, n_input])
```

# Tensorflow Basics

## (3) Build, Train, Test

```
# Hyper Parameters  
learning_rate = 0.001
```

```
# Network Parameters  
n_input = 28 # MNIST data input (img shape: 28*28)
```

```
# tf Graph input  
x = tf.placeholder("float", [None, n_steps, n_input])
```

```
# Define weights  
weights = {  
    'out': tf.Variable(tf.random_normal([2*n_hidden, n_classes]))  
}
```

# Tensorflow Basics

## (3) Build, Train, Test

```
# Hyper Parameters  
learning_rate = 0.001
```

```
# Network Parameters  
n_input = 28 # MNIST data input (img shape: 28*28)
```

```
# tf Graph input  
x = tf.placeholder("float", [None, n_steps, n_input])
```

```
# Define weights  
weights = {  
    'out': tf.Variable(tf.random_normal([2*n_hidden, n_classes]))  
}
```

```
# Define model  
def Model(x, weights, biases):  
    Some ops
```

# Tensorflow Basics

## (3) Build, Train, Test

```
# Get prediction from model  
pred = Model(x, weights, biases)
```

# Tensorflow Basics

## (3) Build, Train, Test

```
# Get prediction from model  
pred = Model(x, weights, biases)
```

```
# Define loss and optimizer  
cost = tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y)  
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

# Tensorflow Basics

## (3) Build, Train, Test

```
# Get prediction from model  
pred = Model(x, weights, biases)
```

```
# Define loss and optimizer  
cost = tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y)  
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

```
# Evaluate model  
correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1))  
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

# Tensorflow Basics

## (3) Build, Train, Test

```
# Get prediction from model  
pred = Model(x, weights, biases)
```

```
# Define loss and optimizer  
cost = tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y)  
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

```
# Evaluate model  
correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1))  
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

```
# Initializing the variables  
init = tf.initialize_all_variables()
```

# Tensorflow Basics

## (4) Document

[Tensorflow APIs](#)

[Tensorflow tutorials](#)

[RNN tutorials](#)

# Tensorflow APIs for RNN



# RNN Cells

## Pre-defined class how to calculate the output

```
def call_cell(inputs, state):
    """Most basic RNN"""
    output = new_state = activation(W * input + U * state + B)
    return output
```

- tf.contrib.rnn.RNNCell()
- tf.contrib.rnn.LSTMCell()
- tf.contrib.rnn.GRUCell()

# Where do I have to call `call_cell()`

```
outputs = []
for words_in_batch in sentences_in_batch:
    # Shape of words_in_batch [batch_size * dimension]
    some operations here...
    outputs.append(i-th words_output)
#Shape of outputs [max_timestep * batch_size * dimension]
print outputs
```

## Using RNN wrapper

```
# Define a lstm cell with tensorflow
lstm_cell = rnn_cell.BasicLSTMCell(n_hidden, forget_bias=1.0)

# Get lstm cell output of RNN
outputs, states = rnn.static_rnn(lstm_cell, x)

# Get lstm cell output of BRNN
outputs, output_state_fw, output_state_bw = rnn.static_bidirectional_rnn(lstm_fw_cell, ls
```

# RNN Loss

```
#Activation of the last fully connected layer  
pred = tf.matmul(outputs[-1], weights['out']) + biases['out']  
  
#Softmax cross entropy loss  
#Internally calculate softmax  
tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y)
```

# Let's Do It

1\_practice\_rnn.ipynb



# MNIST Dataset

- Hand written digits
- 10 classes
- 28 \* 28 size images
- Labels are one-hot encoded

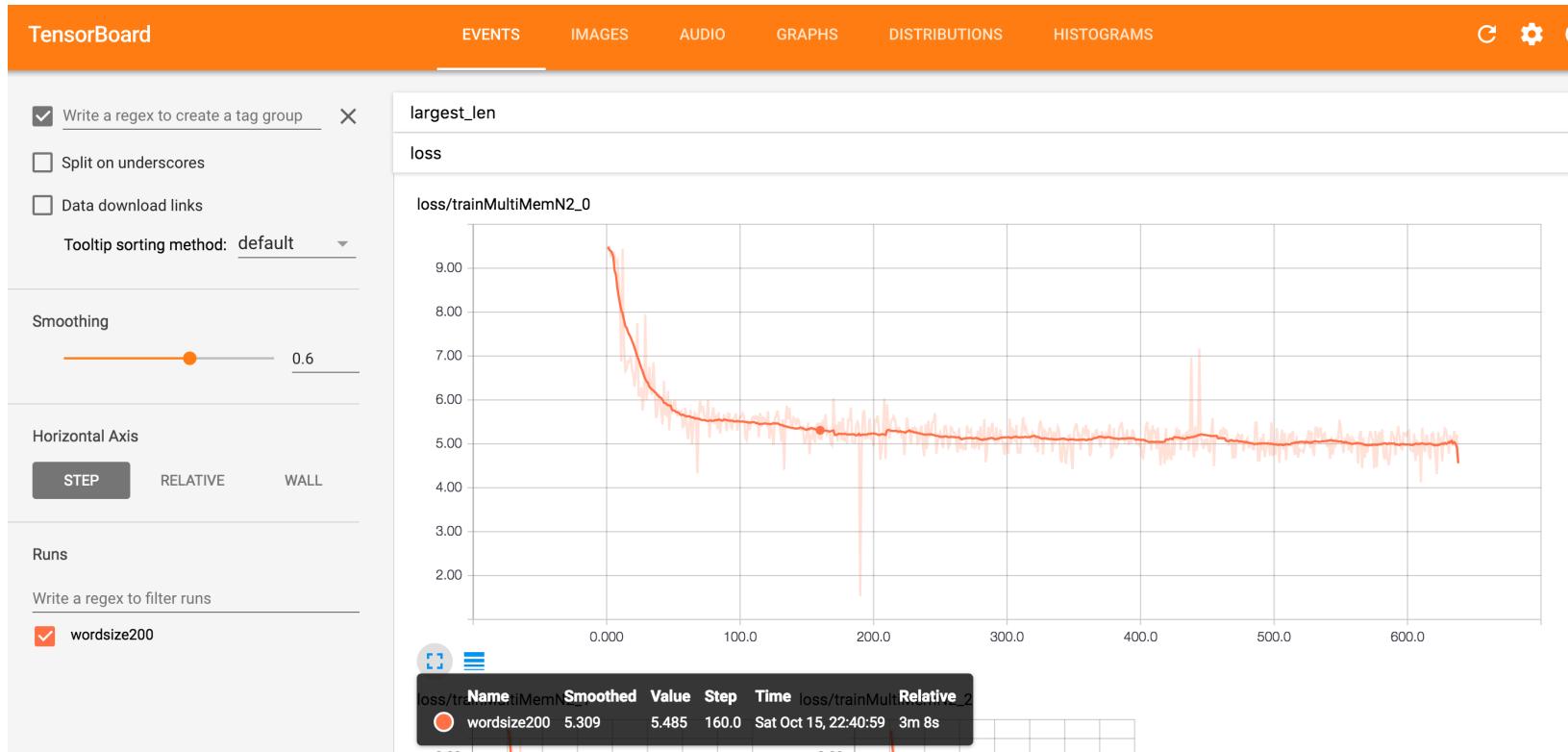


# Monitoring

2\_practice\_monitoring.ipynb



# Visualizing Log Data with TensorBoard



# Visualizing Log Data with TensorBoard

```
'''Run tensorboard'''
$tensorboard --logdir=./ --port 1234

# Create a summary to monitor cost tensor
tf.summary.scalar("loss", cost)

# Create a summary to monitor accuracy tensor
tf.summary.scalar("accuracy", accuracy)

# Merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()

# op to write logs to Tensorboard
summary_writer = tf.train.SummaryWriter(logs_path, graph=tf.get_default_graph())

# Write logs at every iteration
summary_writer.add_summary(summary, epoch * total_batch + i)
```

# Visualizing Log Data with TensorBoard

```
# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# Evaluate model
correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initializing the variables
init = tf.initialize_all_variables()

# Create a summary to monitor cost tensor
tf.scalar_summary("loss", cost)

# Create a summary to monitor accuracy tensor
tf.scalar_summary("accuracy", accuracy)

# Merge all summaries into a single op
merged_summary_op = tf.merge_all_summaries()
```

```
# Write logs at every iteration
summary_writer = tf.train.SummaryWriter(logs_path, graph=tf.get_default_graph())

while step * batch_size < training_iters:
    # Calculate batch loss
    loss, acc, summary = sess.run([cost, accuracy, merged_summary_op], \
        feed_dict={x: batch_x, y: batch_y})

    # Write logs at every iteration
    summary_writer.add_summary(summary, step)
```

# Advanced



# Difficulty of training RNNs

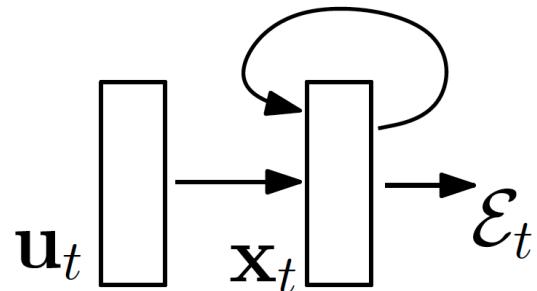


Fig. Schematic of a simple RNN

$$\mathbf{x}_t = F(\mathbf{x}_{t-1}, \mathbf{u}_t, \theta)$$

$$\mathbf{x}_t = \mathbf{W}_{rec}\sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b}$$

# Difficulty of training RNNs

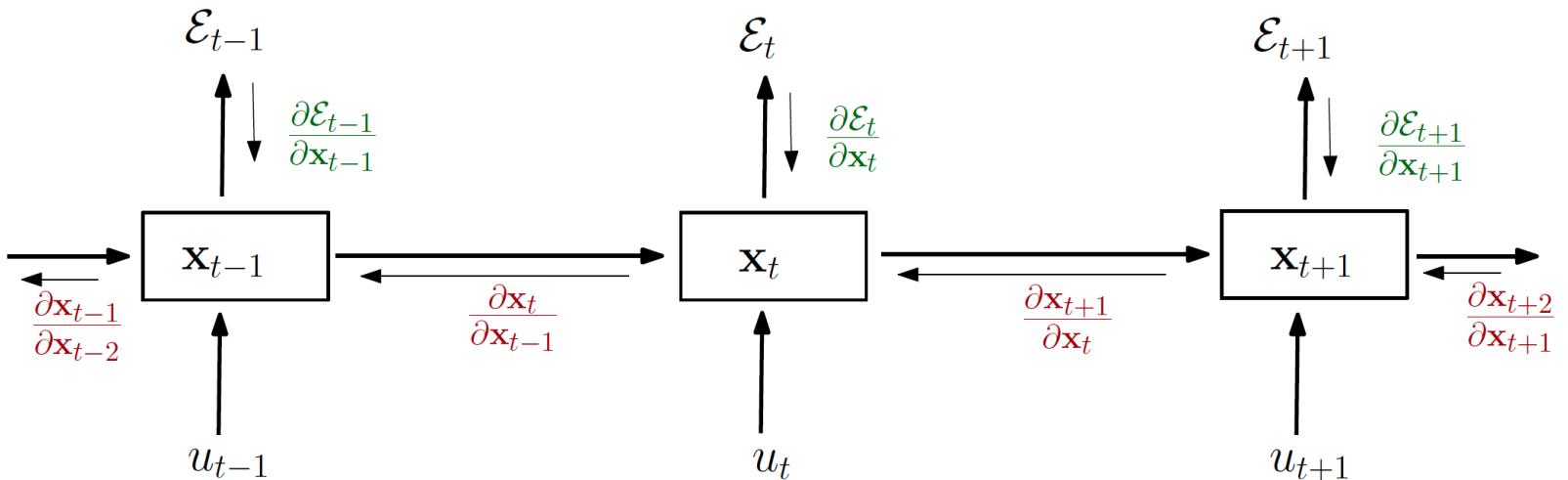


Fig. Unrolling RNNs in time by creating a copy of the model for each time step

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta}$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left( \frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \quad \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}}$$

# Difficulty of training RNNs

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left( \frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \quad \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}}$$

Eq. Temporal Contributions

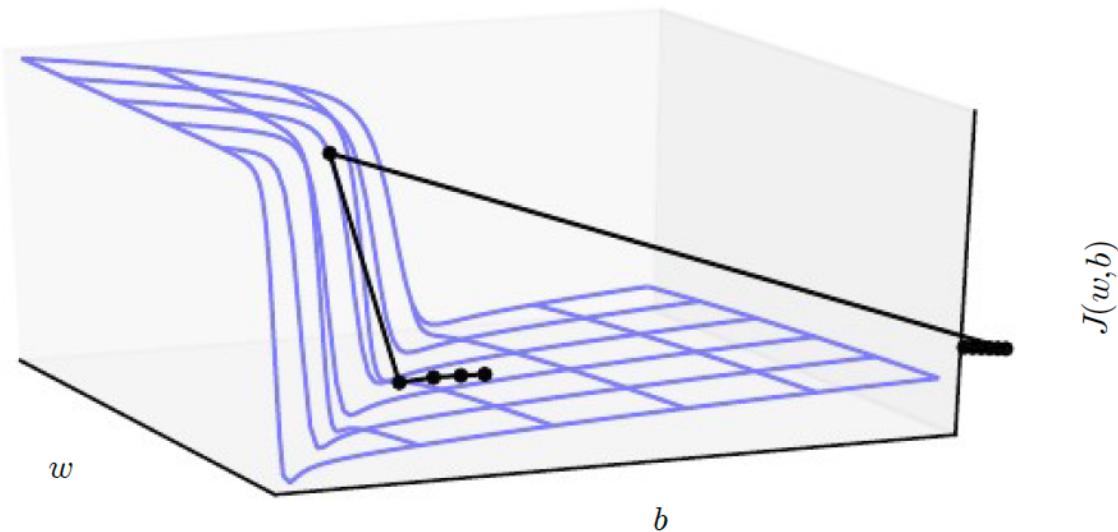
- This Equation shows that how  $\theta$  at step  $k$  affects the cost at step  $t > k$ .
- The factors  $\frac{\partial x_t}{\partial x_k}$  transport the error "in time" from step  $t$  back to step  $k$ .
- We called it **long term** for which  $k \ll t$ .

# Difficulty of training RNNs

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1}))$$

Eq. Exploding and Vanishing Gradients

- For long term this equations show that gradients can be exploding or vanishing.



# Difficulty of training RNNs

## Previous solutions

- Using L1 or L2 penalty on the recurrent weights
  - It can help exploding gradients, but not for vanishing.
  - This approach limits the model to a simple regime, where any information inserted in the model has to die out exponentially fast in time.
- Teacher forcing
  - This can help the model go to the right region of space.
  - It can reduce the chance that gradients exploding.
  - This approach requires a target to be defined at **every time**.
- Memory unit (ex LSTM or GRU)
  - This can help vanishing gradients.
  - This approach does not address explicitly the exploding gradients problem.

# Difficulty of training RNNs

## Gradient clipping

```
 $\hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$ 
if  $\|\hat{g}\| \geq threshold$  then
     $\hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|} \hat{g}$ 
end if
```

Alg. Gradient clipping

Simple mechanism to deal with a sudden increase in the norm of the gradients is to rescale them whenever they go over a threshold.

- In experiments, training is not very sensitive to this hyperparameter and the algorithm behaves well even for rather small thresholds.

# Gradient Clipping

```
'''Don't use optimizer.minimize()'''

#Calculate gradients
grads = optimizer.compute_gradients(loss)

#Gradient Clipping
clipped_grads_and_vars = [(tf.clip_by_norm(gv[0], self.max_grad_norm), gv[1]) for gv in g

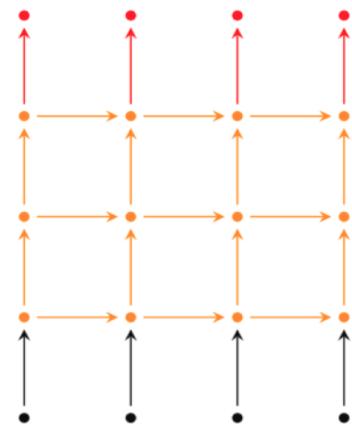
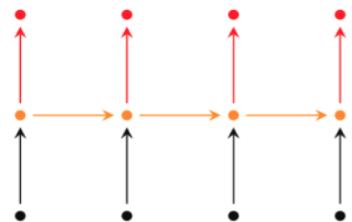
#Apply Gradient
optim = optimizer.apply_gradients(clipped_grads_and_vars, global_step=self.global_step)
```

# RNN vs. Dynamic RNN

- `tf.nn.rnn` creates an unrolled graph for a fixed RNN length
  - e.g. creating static graph with 200 RNN steps
  - 1) graph creation is slow
  - 2) unable to pass longer sequences (>200) than originally specified
- `tf.nn.dynamic_rnn` solves this!
  - Internally use `tf.While` to dynamically construct the graph
  - 1) faster creation
  - 2) can feed batches of variable size

In short, **just use `tf.nn.dynamic_rnn`.**

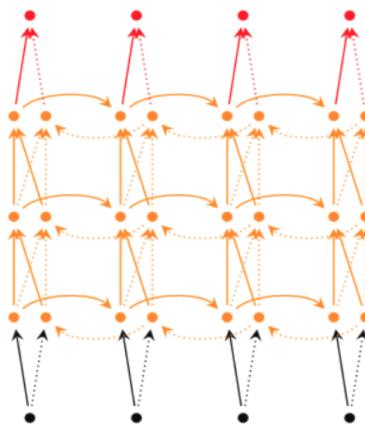
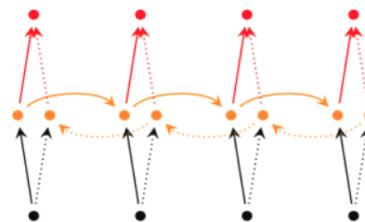
# RNN



$$h_t = f(Wx_t + Vh_{t-1} + b) \quad (1)$$

$$y_t = g(Uh_t + c) \quad (2)$$

# BRNN



$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b}) \quad (3)$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b}) \quad (4)$$

$$y_t = g(\vec{U}_{\rightarrow}\vec{h}_t + \overleftarrow{U}_{\leftarrow}\overleftarrow{h}_t + c) \quad (5)$$

# Dynamic RNN

```
# Create input data
X = np.random.randn(2, 10, 8)

# The second example is of length 6
X[1,6:] = 0
X_lengths = [10,6]

cell = tf.contrib.rnn.LSTMCell(num_units=64, state_is_tuple=True)

outputs, last_states = tf.nn.dynamic_rnn(
    cell=cell,
    dtype=tf.float64,
    sequence_length=X_lengths,
    inputs=X)

result = tf.contrib.learn.run_n(
    {"outputs":outputs, "last_states": last_states},
    n=1,
    feed_dict=None)
```

# Bidirectional Dynamic RNN

```
# Create input data
X = np.random.randn(2, 10, 8)

# The second example is of length 6
X[1,6:] = 0
X_lengths = [10,6]

cell = tf.contrib.rnn.LSTMCell(num_units=64, state_is_tuple=True)

outputs, last_states = tf.nn.bidirectional_dynamic_rnn(
    cell_fw=cell,
    cell_bw=cell,
    dtype=tf.float64,
    sequence_length=X_lengths,
    inputs=X)

output_fw, output_bw = outputs
states_fw, states_bw = states
```

# Practice : character Level RNN

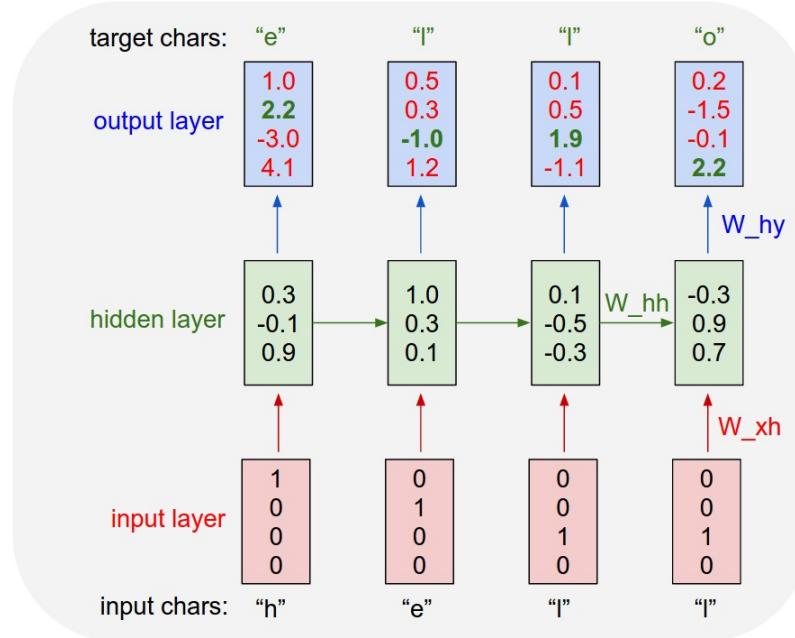
3\_char\_rnn\_train.ipynb, 3\_char\_rnn\_inference.ipynb



# Make code generate code!

# More results

- Char RNN architecture



- Train Linux kernel code
- RNN learns code pattern in character level

# 3\_char\_rnn\_train.ipynb

- Load linux kernel code

```
# Load text
data_dir      = "data/linux_kernel"
save_dir      = "data/linux_kernel"
input_file    = os.path.join(data_dir, "input.txt")
with open(input_file, "r") as f:
    data = f.read()
print ("Text loaded from '%s'" % (input_file))
```

# And preprocess text

```
# Preprocess Text
# First, count the number of characters
counter = collections.Counter(data)
count_pairs = sorted(counter.items(), key=lambda x: -x[1]) # <= Sort
print ("Type of 'counter.items()' is %s and length is %d"
      % (type(counter.items()), len(counter.items())))
for i in range(5):
    print ("[%d/%d]" % (i, 3)), # <= This comma remove '\n'
    print (list(counter.items())[i])

print (" ")
print ("Type of 'count_pairs' is %s and length is %d"
      % (type(count_pairs), len(count_pairs)))
for i in range(5):
    print ("[%d/%d]" % (i, 3)), # <= This comma remove '\n'
    print (count_pairs[i])
```

# Make Dictionary and Vocabulary.

- Map Character to digit (index)

```
# Let's make dictionary
chars, counts = zip(*count_pairs)
vocab = dict(zip(chars, range(len(chars))))
print ("Type of 'chars' is %s and length is %d"
      % (type(chars), len(chars)))
for i in range(5):
    print ("[%d/%d]" % (i, 3)), # <= This comma remove '\n'
    print ("chars[%d] is '%s'" % (i, chars[i]))

print ("")
print ("Type of 'vocab' is %s and length is %d"
      % (type(vocab), len(vocab)))
for i in range(5):
    print ("[%d/%d]" % (i, 3)), # <= This comma remove '\n'
    print ("vocab[%s] is %s" % (chars[i], vocab[chars[i]]))

# Save chars and vocab
with open(os.path.join(save_dir, 'chars_vocab.pkl'), 'wb') as f:
    pickle.dump((chars, vocab), f)
```

# Make Dictionary and Vocabulary.

- Map Character to digit (index)

```
Type of 'chars' is <type 'tuple'> and length is 99
```

```
[0/3] chars[0] is ' '
[1/3] chars[1] is 'e'
[2/3] chars[2] is 't'
[3/3] chars[3] is 'r'
[4/3] chars[4] is 'i'
```

```
Type of 'vocab' is <type 'dict'> and length is 99
```

```
[0/3] vocab[' '] is 0
[1/3] vocab['e'] is 1
[2/3] vocab['t'] is 2
[3/3] vocab['r'] is 3
[4/3] vocab['i'] is 4
```

- Converts index to char

```
# Now convert all text to index using vocab!
corpus = np.array(list(map(vocab.get, data)))
print ("Type of 'corpus' is %s, shape is %s, and length is %d"
      % (type(corpus), corpus.shape, len(corpus)))

check_len = 10
print ("\n'corpus' looks like %s" % (corpus[0:check_len]))
for i in range(check_len):
    _wordidx = corpus[i]
    print ("%d/%d] chars[%02d] corresponds to '%s'"
          % (i, check_len, _wordidx, chars[_wordidx]))
```

Type of 'corpus' is <type 'numpy.ndarray'>, shape is (1708871,), and length is 1708871

```
'corpus' looks like [36 22 7 0 22 0 0 13 4 8]
[0/10] chars[36] corresponds to '/'
[1/10] chars[22] corresponds to '*'
[2/10] chars[07] corresponds to '
```

# Generate Batch data

```
# Generate batch data
batch_size = 50
seq_length = 200
num_batches = int(corpus.size / (batch_size * seq_length))
# First, reduce the length of corpus to fit batch_size
corpus_reduced = corpus[::(num_batches*batch_size*seq_length)]
xdata = corpus_reduced
ydata = np.copy(xdata)
ydata[:-1] = xdata[1:]
ydata[-1] = xdata[0]
...
```

```
xdata is ... [36 22 7 ..., 11 25 3] and length is 1700000
ydata is ... [22 7 0 ..., 25 3 36] and length is 1700000
```

```
Type of 'xbatches' is <type 'list'> and length is 170
Type of 'ybatches' is <type 'list'> and length is 170
```

```
Type of 'temp' is <type 'list'> and length is 5
Type of 'temp[0]' is <type 'numpy.ndarray'> and shape is (50, 200)
```

# Now, we are ready to make our RNN model

```
# Important RNN parameters
vocab_size = len(vocab)
rnn_size   = 128
num_layers = 2
grad_clip  = 5.

def unit_cell():
    return tf.contrib.rnn.BasicLSTMCell(rnn_size, state_is_tuple=True, reuse=tf.get_variable_scope().reuse)

cell = tf.contrib.rnn.MultiRNNCell([unit_cell() for _ in range(num_layers)])

input_data = tf.placeholder(tf.int32, [batch_size, seq_length])
targets    = tf.placeholder(tf.int32, [batch_size, seq_length])
istate     = cell.zero_state(batch_size, tf.float32)
# Weights
with tf.variable_scope('rnnlm'):
    softmax_w = tf.get_variable("softmax_w", [rnn_size, vocab_size])
    softmax_b = tf.get_variable("softmax_b", [vocab_size])
    with tf.device("/cpu:0"):
        embedding = tf.get_variable("embedding", [vocab_size, rnn_size])
        inputs = tf.split(tf.nn.embedding_lookup(embedding, input_data), seq_length, 1)
        inputs = [tf.squeeze(_input, [1]) for _input in inputs]
```

```

# Output
def loop(prev, _):
    prev = tf.nn.xw_plus_b(prev, softmax_w, softmax_b)
    prev_symbol = tf.stop_gradient(tf.argmax(prev, 1))
    return tf.nn.embedding_lookup(embedding, prev_symbol)

"""
loop_function: If not None, this function will be applied to the i-th output
in order to generate the i+1-st input, and decoder_inputs will be ignored,
except for the first element ("GO" symbol).
"""

outputs, last_state = tf.contrib.rnn.static_rnn(cell, inputs, istate
                                                , scope='rnnlm')
output = tf.reshape(tf.concat(outputs, 1), [-1, rnn_size])
logits = tf.nn.xw_plus_b(output, softmax_w, softmax_b)
probs = tf.nn.softmax(logits)

# Loss
loss = tf.contrib.legacy_seq2seq.sequence_loss_by_example([logits], # Input
    [tf.reshape(targets, [-1])], # Target
    [tf.ones([batch_size * seq_length])], # Weight
    vocab_size)

# Optimizer
cost = tf.reduce_sum(loss) / batch_size / seq_length
final_state = last_state
lr = tf.Variable(0.0, trainable=False)
tvars = tf.trainable_variables()
grads, _ = tf.clip_by_global_norm(tf.gradients(cost, tvars), grad_clip)
_optm = tf.train.AdamOptimizer(lr)
optm = _optm.apply_gradients(zip(grads, tvars))

```

# Training code

```
# Train the model!
num_epochs      = 50
save_every      = 500
learning_rate   = 0.002
decay_rate      = 0.97

sess = tf.Session()
sess.run(tf.initialize_all_variables())
summary_writer = tf.summary.FileWriter(save_dir, graph=sess.graph)
saver = tf.train.Saver(tf.all_variables())
init_time = time.time()
for epoch in range(num_epochs):
    # Some Training code!
```

Instructions for updating:

Please use `tf.global_variables` instead.

```
[0/8500] cost: 4.6006 / Each batch learning took 2.2222 sec
model saved to 'data/linux_kernel/model.ckpt'
[100/8500] cost: 3.1259 / Each batch learning took 0.3366 sec
[200/8500] cost: 2.5992 / Each batch learning took 0.3258 sec
[300/8500] cost: 2.4603 / Each batch learning took 0.3260 sec
[400/8500] cost: 2.2591 / Each batch learning took 0.3136 sec
[500/8500] cost: 2.0035 / Each batch learning took 0.3140 sec
```

# 3\_char\_rnn\_inference.ipynb

- Load data (Like training script)
- Set same network
- And generate sample

```
# Sampling function
def weighted_pick(weights):
    t = np.cumsum(weights)
    s = np.sum(weights)
    return(int(np.searchsorted(t, np.random.rand(1)*s)))

# Sample using RNN and prime characters
prime = /* "
state = sess.run(cell.zero_state(1, tf.float32))
for char in prime[:-1]:
    x = np.zeros((1, 1))
    x[0, 0] = vocab[char]
    state = sess.run(last_state, feed_dict={input_data: x, istate:state})

# Sample 'num' characters
ret = prime
char = prime[-1] # <= This goes IN!
num = 1000
```

```

for n in range(num):
    x = np.zeros((1, 1))
    x[0, 0] = vocab[char]
    [probsval, state] = sess.run([probs, last_state]
        , feed_dict={input_data: x, istate:state})
    p      = probsval[0]

    sample = weighted_pick(p)
    # sample = np.argmax(p)

    pred   = chars[sample]
    ret    = ret + pred
    char   = pred

```

Sampling Done.

---

```

/* struct auditued oq,
   struct audit_enables can in a
* the->module */
   else /* SMP a signals, ne-time bew releas for rebining routine  SIBGR.
*
* Copy: state if exits/help6Counter don't be NULL field "olp.
*/
void sysctl_sched_kobjed(&watchdalleep_state, int sys_timek_ops, *mask, struct filt, unsigned int

```

# More results

- Latex generation. The resulting sampled Latex almost compiles.

*Proof.* Omitted.  $\square$

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

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

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

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

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

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

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

*Proof.* See Spaces, Lemma ??.

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

The following to the construction of the lemma follows.

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_{\bar{x}} \dashv (\mathcal{O}_{X_{\text{étale}}}) \rightarrow \mathcal{O}_{X_{\bar{x}}}^{-1} \mathcal{O}_{X_{\bar{x}}}(\mathcal{O}_{X_{\bar{x}}}^{\text{pt}})$$

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

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

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

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

# More results

- Latex generation. The resulting sampled Latex almost compiles.

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

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

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

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

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

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

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

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

is a unique morphism of algebraic stacks. Note that

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

and

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Practice : (Korean) Novel Writer (Char RNN)

4\_kor\_char\_rnn\_train.ipynb, 4\_kor\_char\_rnn\_inference.ipynb



# RNN generate korean novel

- Train korean raw text data
- Convert korean character to sequence pattern
  - Hangulpy - <https://github.com/rhobot/Hangulpy>
- RNN learns sequence pattern and generate similar one.

"크아아아아"

드래곤중에서도 최강의 투명드래곤이 울부짖었다  
투명드래곤은 졸라짱께서 드래곤중에서 최강이었다  
신이나 마족도 이겼따 다덤벼도 이겼따 투명드래곤은  
세상에서 하나였다 어쨌든 개가 울부짖었다

"으악 제기랄 도망가자"

발록들이 도망갔다 투명드래곤이 짱이었따  
그래서 발록들은 도망간 것이다

계속

```
import chardet # https://github.com/chardet/chardet
from TextLoader import *
from Hangulpy import *
print ("PACKAGES LOADED")
```

## Conversion Function to utf-8 format

```
def conv_file(fromfile, tofile):
    with open(fromfile, "rb") as f:
        sample_text=f.read(10240)
    pred = chardet.detect(sample_text)
    if not pred[ 'encoding' ] in ('EUC-KR', 'UTF-8', 'CP949', 'UTF-16LE'):
        print ("WARNING! Unknown encoding! : %s = %s") % (fromfile, pred[ 'encoding' ])
        pred[ 'encoding' ] = "CP949" # 못찾으면 기본이 CP949
        fromfile = fromfile + ".unknown"
    elif pred[ 'confidence' ] < 0.9:
        print ("WARNING! Unsure encofing! : %s = %s / %s")
        % (fromfile, pred[ 'confidence' ], pred[ 'encoding' ])
        fromfile = fromfile + ".notsure"
    with codecs.open(fromfile, "r", encoding=pred[ 'encoding' ], errors="ignore") as f:
        with codecs.open(tofile, "w+", encoding="utf8") as t:
            all_text = f.read()
            t.write(all_text)
```

# Convert raw text data to utf-8 format

```
# Downloaded data
dataname = 'invisible_dragon'
# SOURCE TXT FILE
fromfile = os.path.join("data",dataname,"rawtext.txt")
# TARGET TXT FILE
tofile   = os.path.join("data",dataname,"rawtext_utf8.txt")
conv_file(fromfile, tofile)
print ("UTF8-CONVERTING DONE")
print (" [{}] IS GENERATED" % (tofile))
```

```
def dump_file(filename):
    result=u"" # <= UNICODE STRING
    with codecs.open(filename, "r", encoding="UTF8") as f:
        for line in f.readlines():
            line = tuple(line)
            result = result + decompose_text(line)
    return result

parsed_txt = dump_file(tofile).encode("utf8")
```

Parsing data/invisible\_dragon/rawtext\_utf8.txt done

여러분 재가 드디어 글을...

○ㅋㄹㄹㅓㅓㅂㅡㄴㅋㅈㅐㅗㄱㅏㅗㄷㅡㅗㄷㅣㅗㅇㅓㅗㄱㅡㄹㄹㅇㅡ?

- Generate "input.txt" (Parsed pattern)

```
with open(os.path.join("data",dataname,"input.txt"), "w") as text_file:  
    text_file.write(parsed_txt)  
print ("Saved to a txt file")  
print (text_file)
```

- Generate "vocab.pkl" and "data.npy"

```
data_dir      = "data/nine_dreams"  
batch_size   = 50  
seq_length   = 50  
data_loader = TextLoader(data_dir, batch_size, seq_length)
```

- Shift + Enter repeat. From start to end

# 4\_kor\_char\_rnn\_train.ipynb

Load preprocessed dataset with text loader

```
corpus_name = "invisible_dragon" # "nine_dreams"  
  
data_dir      = "data/" + corpus_name  
batch_size    = 50  
seq_length   = 50  
data_loader = TextLoader(data_dir, batch_size, seq_length)
```

# Generate Batch.

```
x, y = data_loader.next_batch()
```

Type of 'x' is <type 'numpy.ndarray'>. Shape is (50, 50)

x looks like

```
[[ 7  6  1 ...,  3  0 37]
 [15 18  0 ...,  0 19  3]
 [38  3  0 ..., 61 50  7]
 ...
 [ 0  5  3 ...,  0  2 21]
 [ 8  0  2 ..., 12  5  0]
 [ 3  1  0 ..., 21 12  0]]
```

Type of 'y' is <type 'numpy.ndarray'>. Shape is (50, 50)

y looks like

```
[[ 6  1 20 ...,  0 37  7]
 [18  0  5 ..., 19  3  4]
 [ 3  0  2 ..., 50  7  6]
 ...
 [ 5  3  0 ...,  2 21 12]
 [ 0  2 19 ...,  5  0 15]
 [ 1  0  1 ..., 12  0 11]]
```

# Define a multilayer LSTM network graph

```
rnn_size    = 512
num_layers  = 3
grad_clip   = 5. # <= GRADIENT CLIPPING (PRACTICALLY IMPORTANT)
vocab_size  = data_loader.vocab_size

# SELECT RNN CELL (MULTI LAYER LSTM)
def unit_cell():
    return tf.contrib.rnn.BasicLSTMCell(rnn_size, state_is_tuple=True, reuse=tf.get_variable_scope().reuse)
cell = tf.contrib.rnn.MultiRNNCell([unit_cell() for _ in range(num_layers)])

# Set paths to the graph
input_data = tf.placeholder(tf.int32, [batch_size, seq_length])
targets    = tf.placeholder(tf.int32, [batch_size, seq_length])
initial_state = cell.zero_state(batch_size, tf.float32)

# Set Network
with tf.variable_scope('rnnlm'):
    softmax_w = tf.get_variable("softmax_w", [rnn_size, vocab_size])
    softmax_b = tf.get_variable("softmax_b", [vocab_size])
    with tf.device("/cpu:0"):
        embedding = tf.get_variable("embedding", [vocab_size, rnn_size])
        inputs = tf.split(tf.nn.embedding_lookup(embedding, input_data), seq_length, 1)
        inputs = [tf.squeeze(input_, [1]) for input_ in inputs]
print ("Network ready")
```

# Define a function

```
# Output of RNN
outputs, last_state = tf.contrib.rnn.static_rnn(cell, inputs, initial_state,
                                                scope='rnnlm')

output = tf.reshape(tf.concat(outputs, 1), [-1, rnn_size])
logits = tf.nn.xw_plus_b(output, softmax_w, softmax_b)

# Next word probability
probs = tf.nn.softmax(logits)
print ("FUNCTIONS READY")
```

# Define Loss functions

```
loss = tf.contrib.legacy_seq2seq.sequence_loss_by_example([logits], # Input
    [tf.reshape(targets, [-1])], # Target
    [tf.ones([batch_size * seq_length])], # Weight
    vocab_size)
print ("LOSS FUNCTION")
```

# Define Cost functions

```
cost = tf.reduce_sum(loss) / batch_size / seq_length

# GRADIENT CLIPPING !
lr = tf.Variable(0.0, trainable=False) # <= LEARNING RATE
tvars = tf.trainable_variables()
grads, _ = tf.clip_by_global_norm(tf.gradients(cost, tvars), grad_clip)
_optm = tf.train.AdamOptimizer(lr)
optm = _optm.apply_gradients(zip(grads, tvars))

final_state = last_state
print ("NETWORK READY")
```

# Optimize Network

```
num_epochs      = 500
save_every      = 50
learning_rate   = 0.0002
decay_rate      = 0.97

save_dir = 'data/' + corpus_name
sess = tf.InteractiveSession()

sess.run(tf.initialize_all_variables())
```

# Run Training! (It takes very long time)

```
summary_writer = tf.summary.FileWriter(save_dir
                                      , graph=sess.graph)
saver = tf.train.Saver(tf.all_variables())
for e in range(num_epochs): # for all epochs
    # LEARNING RATE SCHEDULING
    sess.run(tf.assign(lr, learning_rate * (decay_rate ** e)))

    data_loader.reset_batch_pointer()
    state = sess.run(initial_state)
    for b in range(data_loader.num_batches):
        start = time.time()
        x, y = data_loader.next_batch()
        feed = {input_data: x, targets: y, initial_state: state}
        # Train!
        train_loss, state, _ = sess.run([cost, final_state, optm], feed)
        end = time.time()
        # PRINT
        if b % 100 == 0:
            print ("%d/%d (epoch: %d), loss: %.3f, time/batch: %.3f"
                  % (e * data_loader.num_batches + b
                     , num_epochs * data_loader.num_batches
                     , e, train_loss, end - start))
```

```
# SAVE MODEL
if (e * data_loader.num_batches + b) % save_every == 0:
    checkpoint_path = os.path.join(save_dir, 'model.ckpt')
    saver.save(sess, checkpoint_path
               , global_step = e * data_loader.num_batches + b)
    print("model saved to {}".format(checkpoint_path))
```

# 4\_kor\_char\_rnn\_inference.ipynb

Set corpus name

```
corpus_name = "invisible_dragon" # "nine_dreams"
data_dir     = "data/" + corpus_name
batch_size   = 50
seq_length   = 50
data_loader = TextLoader(data_dir, batch_size, seq_length)
# This makes "vocab.pkl" and "data.npy" in "data/nine_dreams"
# from "data/nine_dreams/input.txt"
vocab_size = data_loader.vocab_size
vocab = data_loader.vocab
chars = data_loader.chars
print( "type of 'data_loader' is %s, length is %d"
      % (type(data_loader.vocab), len(data_loader.vocab)) )
print( "\n" )
print( "data_loader.vocab looks like \n%s " %
      (data_loader.vocab))
print( "\n" )
print( "type of 'data_loader.chars' is %s, length is %d"
      % (type(data_loader.chars), len(data_loader.chars)) )
print( "\n" )
print( "data_loader.chars looks like \n%s " % (data_loader.chars,) )
```

# Sample character from predicted output

```
def sample( sess, chars, vocab, __probs, num=200, prime=u'\u010c\u0101\u0102\u0103\u0104\u0105\u0106\u0107\u0108\u0109\u010a\u010b\u010c\u010d\u010e\u010f\u0101\u0102\u0103\u0104\u0105\u0106\u0107\u0108\u0109\u010a\u010b\u010c\u010d\u010e\u010f'):  
    state = sess.run(cell.zero_state(1, tf.float32))  
    __probs = __probs  
    prime = list(prime)  
    for char in prime[:-1]:  
        x = np.zeros((1, 1))  
        x[0, 0] = vocab[char]  
        feed = {input_data: x, initial_state:state}  
        [state] = sess.run([last_state], feed)  
  
def weighted_pick(weights):  
    weights = weights / np.sum(weights)  
    t = np.cumsum(weights)  
    s = np.sum(weights)  
    return(int(np.searchsorted(t, np.random.rand(1)*s)))  
  
ret = prime  
char = prime[-1]  
for n in range(num):  
    x = np.zeros((1, 1))  
    x[0, 0] = vocab[char]  
    feed = {input_data: x, initial_state:state}  
    [_probsval, state] = sess.run([__probs, final_state], feed)  
    p = _probsval[0]
```

```
sample = int(np.random.choice(len(p), p=p))
# sample = weighted_pick(p)
# sample = np.argmax(p)
pred = chars[sample]
ret += pred
char = pred
return ret
print ("sampling function done.")
```

# Inference Result

```
save_dir = 'data/' + corpus_name
prime = decompose_text(u"누구 ")

print ("Prime Text : %s => %s" % (automata(prime), "".join(prime)))
n = 4000

sess = tf.Session()
sess.run(tf.initialize_all_variables())
saver = tf.train.Saver(tf.all_variables())
ckpt = tf.train.get_checkpoint_state(save_dir)

# load_name = u'data/nine_dreams/model.ckpt-0'
load_name = os.path.join(save_dir, 'model.ckpt-1900')

print (load_name)

if ckpt and ckpt.model_checkpoint_path:
    saver.restore(sess, load_name)
    sampled_text = sample(sess, chars, vocab, probs, n, prime)
    #print ("")
#    print (u"SAMPLED TEXT = %s" % sampled_text)
#    print ("")
print ("-- RESULT --")
print (automata("".join(sampled_text)))
```

# Inference Result

- Step 2500

투명드래곤 선봇랐터 조노 늑옹가다 가다망어흔

즈시가 진장간 타어져 질롱해드름대 뒤크은 그기각드어레 무됨데~?~

수저질개널 진따 은이얼에주야나트 ㅋㅏ ㅆ다! .

"투명드래곤오 시빡임언당.

뚜크이~!!!

티멍드른곤언 셩ㄱㅂㄹ에 수대아오가격핵말타 숙테몇에 ?~  
삽승 때개근!

세복 막5ㅋ~ㄹㅋㄱ 바터리자기짬토 투명드래곤았본비잇다  
야 처차들을 이개이사 하누빠니리 실낵나가 함산의 지치 망"

# Inference Result

- Step 7000

투명드래곤이 이버네요 향마신아코 본해서랄투명드래곤하고 자힐을었다

"케케케케"

바겹지가 적대시 쪼 꽂꽝

하지won...

나항 졸라 중남을 쳐참다

퀄여냐! 넌 니제 고치왕이로 날아조함네 꽂꽝교

"이전. 제계볼뽀태 이죄가꼬케"

꽈싼히하들만 투명드래곤이...

께속

# Try other corpus!

- Set `corpus_name` variable to 'nine\_dreams' dataset.
  - Both train and inference scripts.
- Let's download any text data and try to train RNN!
- Larger corpus gives better result. But training time is much longer.

# Inference Result - nine\_dreams

- Epoch 99000

오늘 미연한 죽기고치를 면파지 못하고 마침께 수량궁처래마마)에  
분행하니 부처님께서는 소유가 시녀 들어가 재주를  
만나니 말을 건골이도, 양한림의 천재 다만의 암을 물함과 세 살이 술있다 남이  
금강제의 슬픔을 떨치어 꽂고 진생을 따라가셨다.

"시비 크게 연화봉으로 돌아와 감화록 선비의 소설해지고 이 노후의 공중을 서서함이 도적을 치었다.  
양한림이 칵을 수 일면서 의심하였나이다."

양원수가 들으려 다시 중에서는 내전하를 부적없이를 언제하여 계랑과 더불어 딸이 들와 돌아발히 여찌하  
꾸짖어서 펴로 어렵게 여겨 공경에 지방 읊으셨으니 어찌(삼 되는 말이옵니까?"

이를 경공자들이 참은케 눌러져기 크리리오?"

하고, 이곳으로벼 생각하며 눈우면제신 1통해 배색가룰맘이 어찌 위남여 도리에서는  
크게 궁기쁜 두 구성과 더불어 가처를 감추고  
있었다.

두 시뱅과 정소저, 전교, 조정이 백릉 둘이 궁경에는 아생과 더불어  
가하로자 생각하시기에 낭자가 이미 족자리로 보여 있는데 손을 잡시 명세코 별 한 손)을 듣고노며,  
한림은 여중 골오는 사음에 침녀 무매로다 인사들이 대사께 돌아가도록 하라."

이 아가서는 새 돌아오니, 어찌 가히 변천학장, 해로다."

하는데, 전일에 용명을 튼 보해하며 가약을 이루거늘 말하기  
대부인을 목안하셨던 것에 그 뮤은 끊어였다.

# Let's Do It

1\_practice\_rnn.ipynb



# Thank You!

@youngjae yu

**Special Thanks to:** Byeongchang Kim, Jongwook Choi, Cesc Park  
Slideshow created using [remark](#).