Recurrent Neural Network using TensorFlow

Youngjae Yu

May 18, 2017

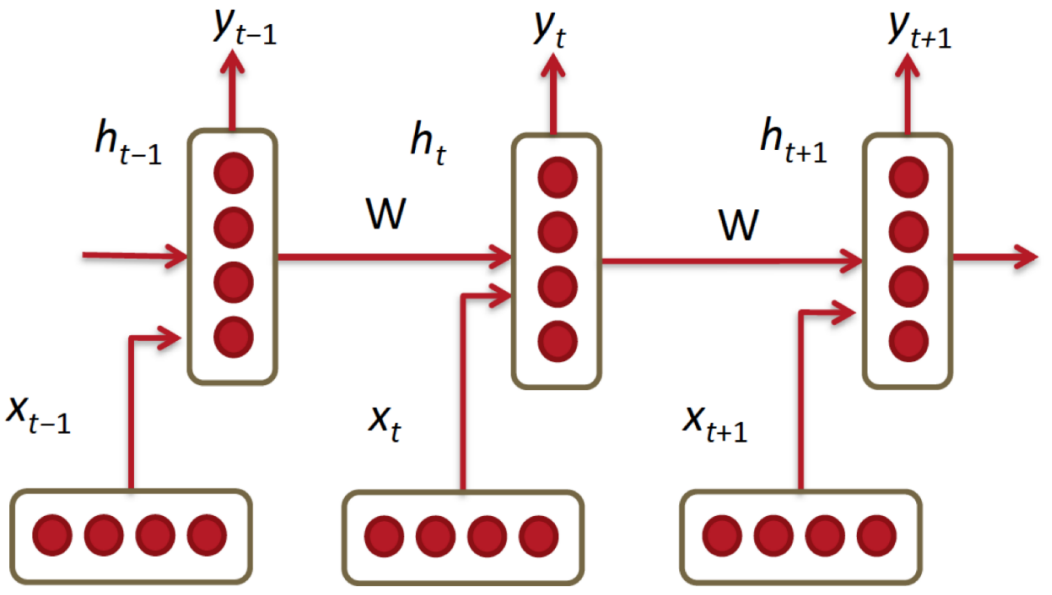
https://yj-yu.github.io/rnntf

About

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

RNN Basics

Recurrent Neural Networks!

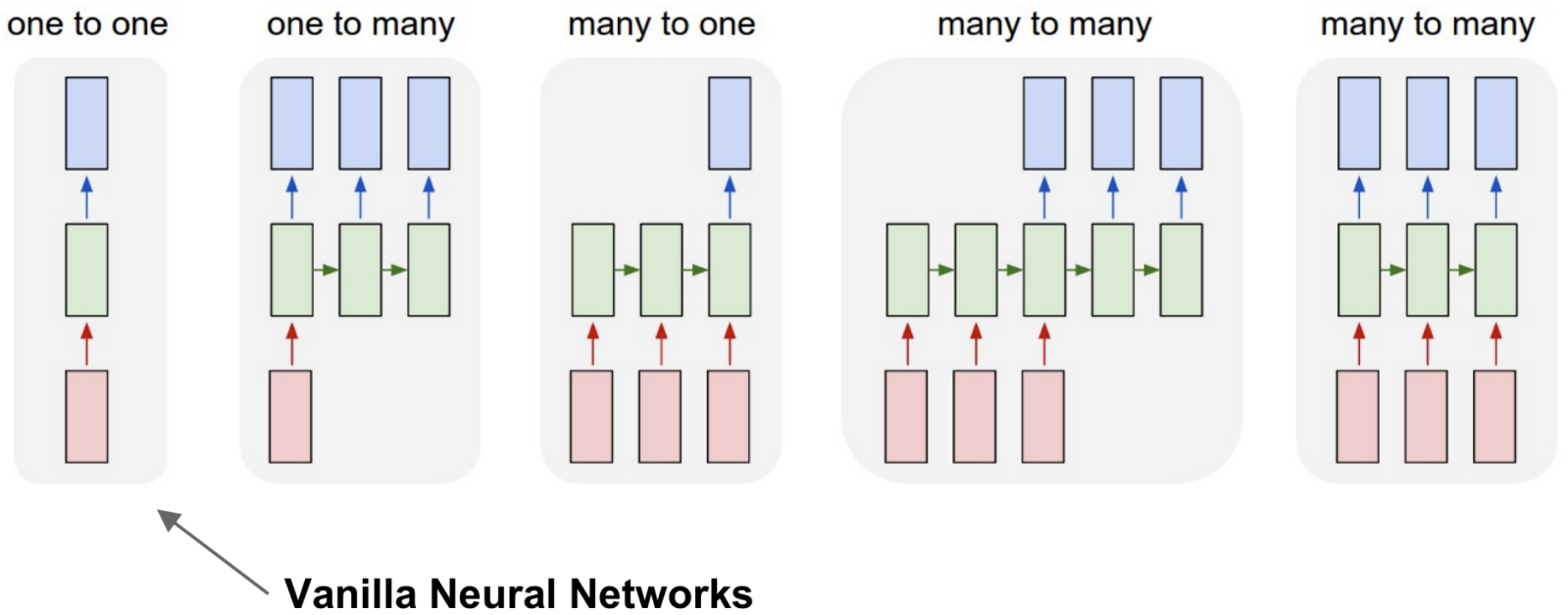


Richard Socher

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

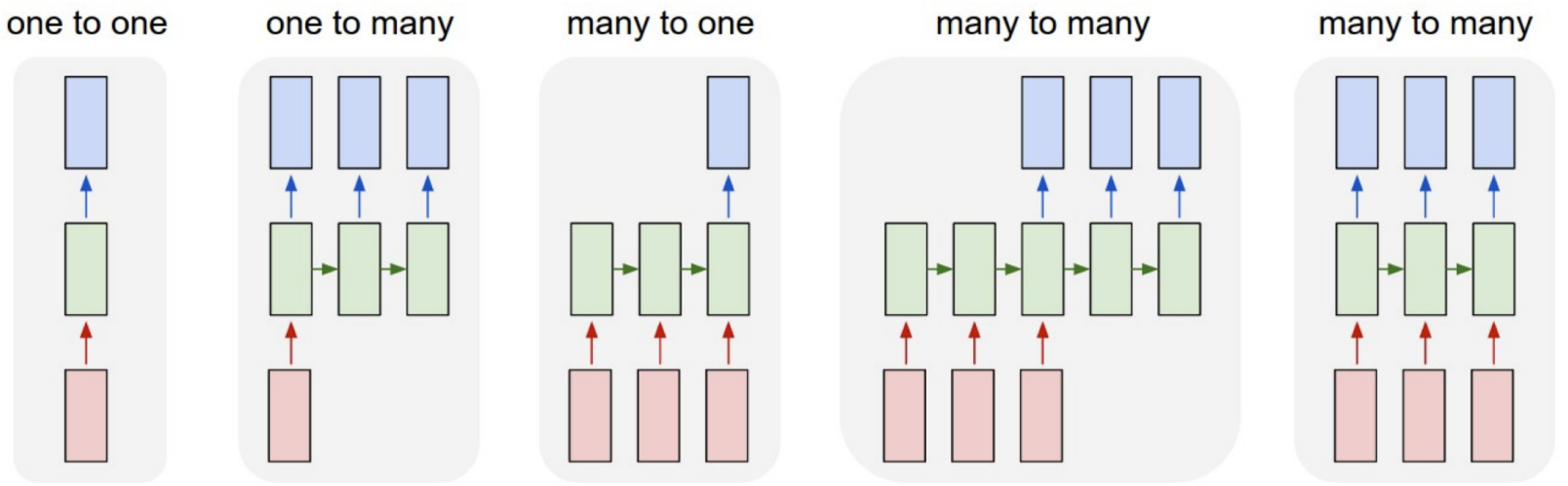
((Slide credit: cs224d)

RNN Basics



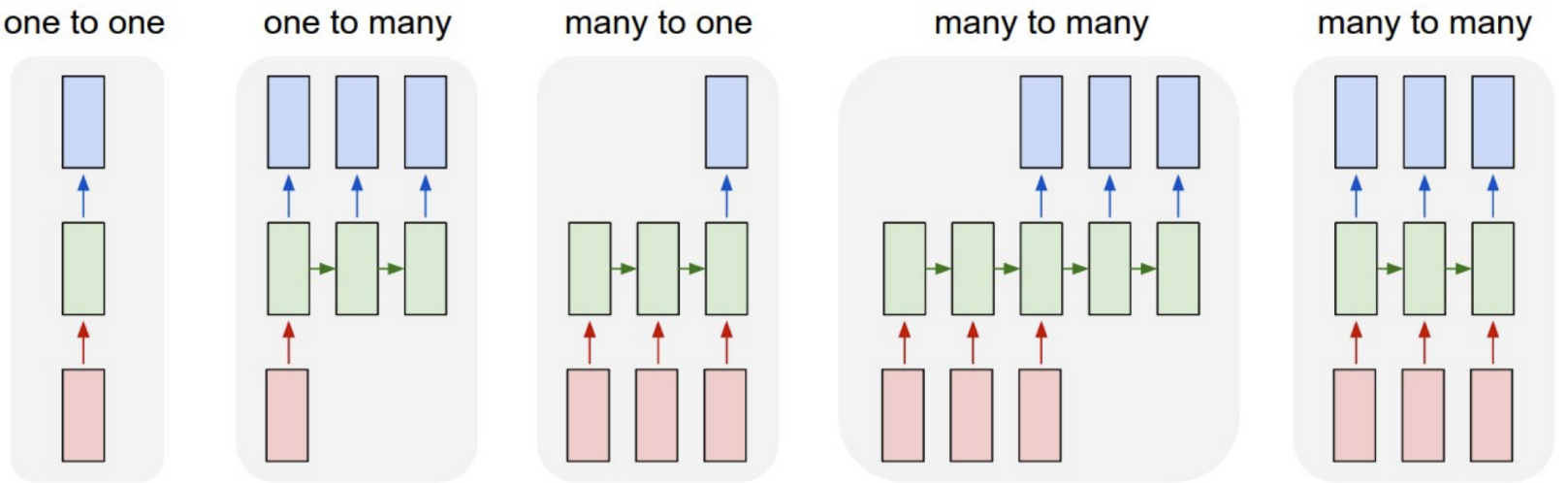
(Image credit: Karpathy's blog)

RNN Basics



(Image credit: Karpathy's blog)

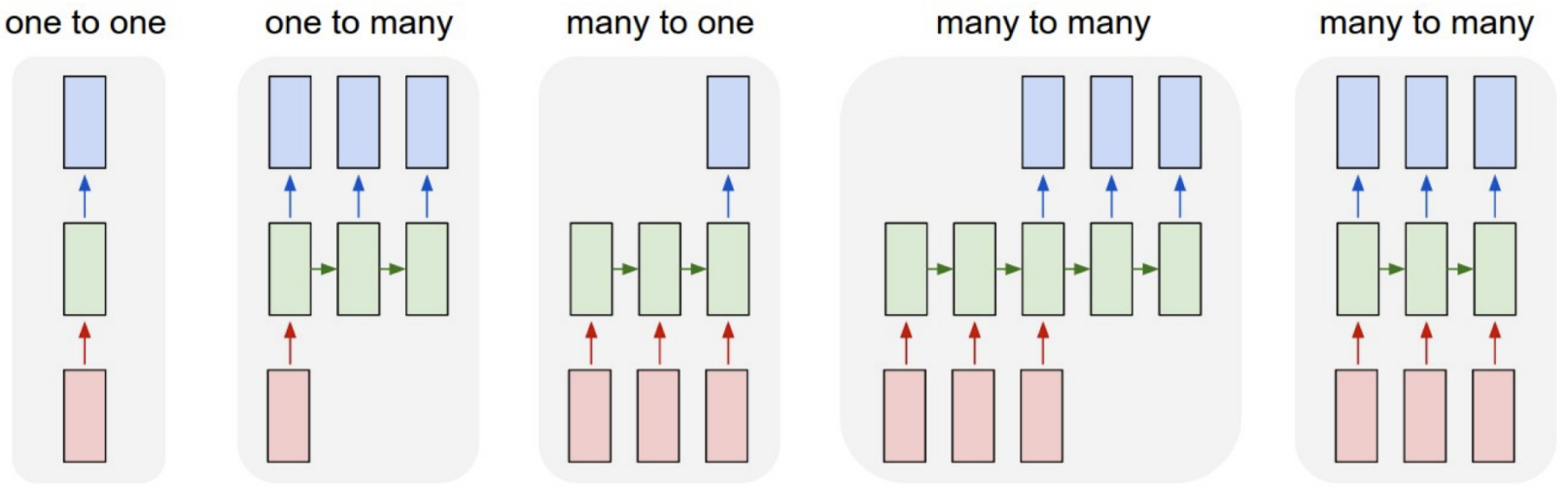
RNN Basics



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

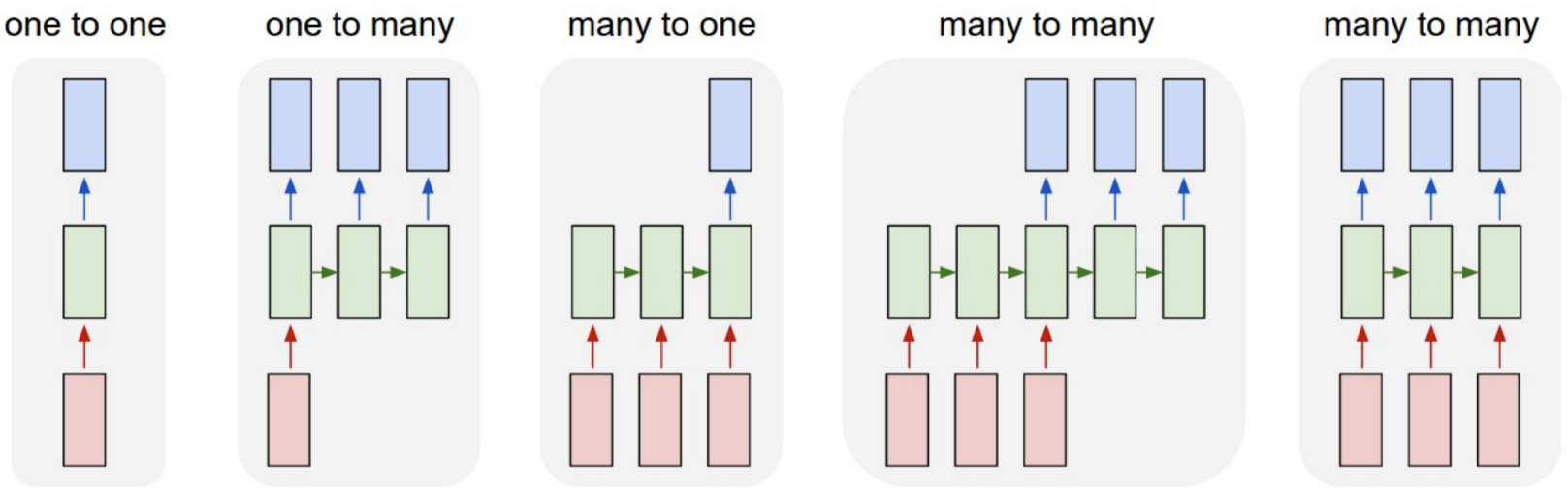
(Image credit: Karpathy's blog)

RNN Basics



(Image credit: Karpathy's blog)

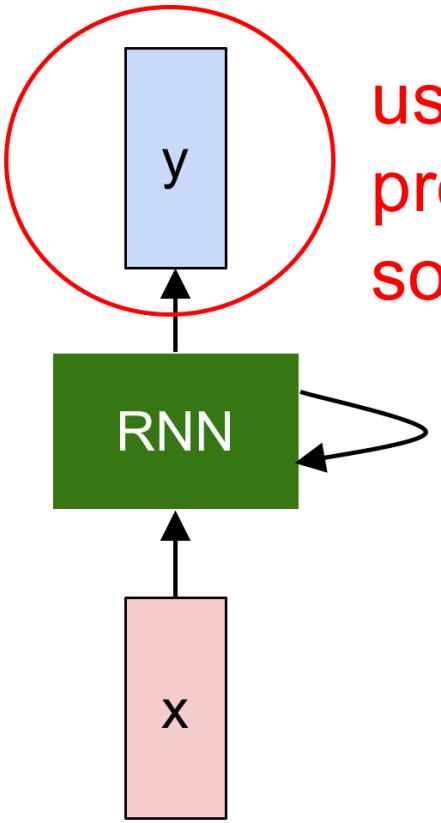
RNN Basics



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

(Image credit: Karpathy's blog)

RNN Basics



usually want to predict a vector at some time steps

RNN Basics

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

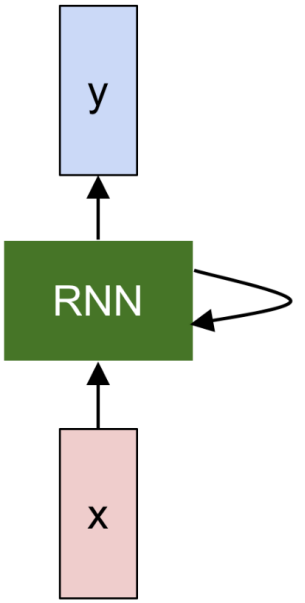
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

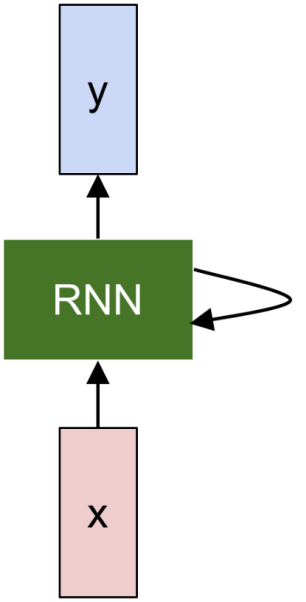
input vector at some time step



RNN Basics

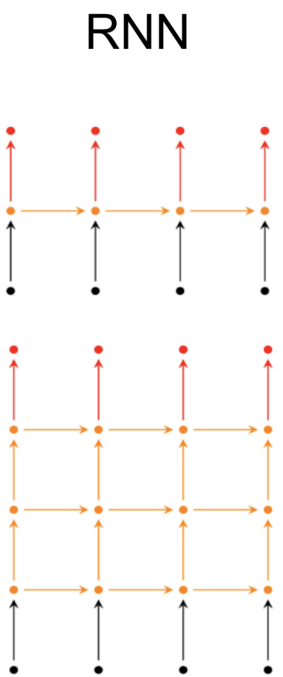
We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

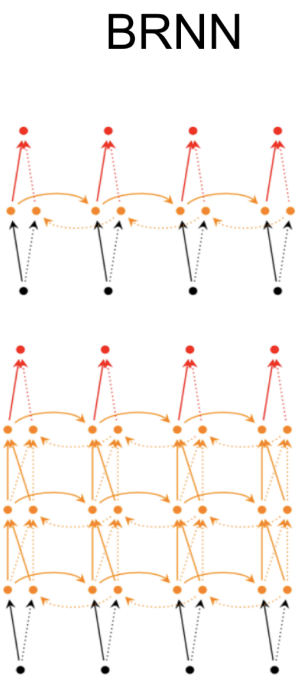
$$h_t = f_W(h_{t-1}, x_t)$$



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

RNN Basics





$$h_t = f(Wx_t + Vh_{t-1} + b) \quad (1)$$
$$y_t = g(Uh_t + c) \quad (2)$$

$$\overrightarrow{h}_t = f(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h}_{t-1} + \overrightarrow{b}) \quad (3)$$
$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b}) \quad (4)$$
$$y_t = g(U_{\rightarrow}\overrightarrow{h}_t + U_{\leftarrow}\overleftarrow{h}_t + c) \quad (5)$$

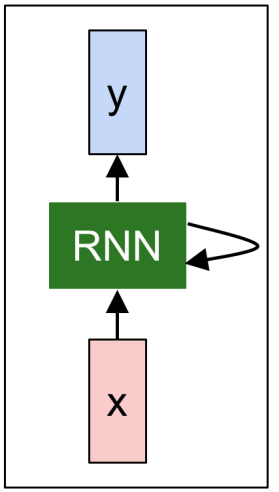
(Image credit: WildML)

RNN Basic

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

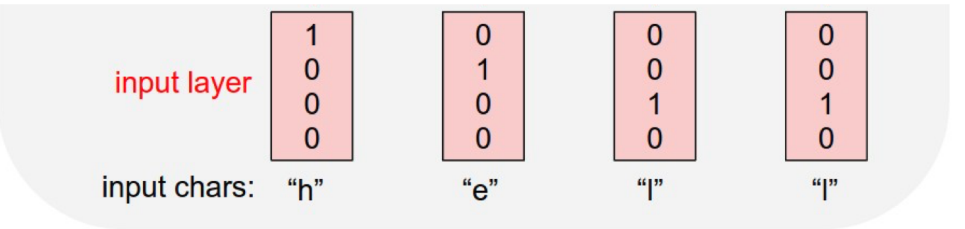


RNN Basic

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



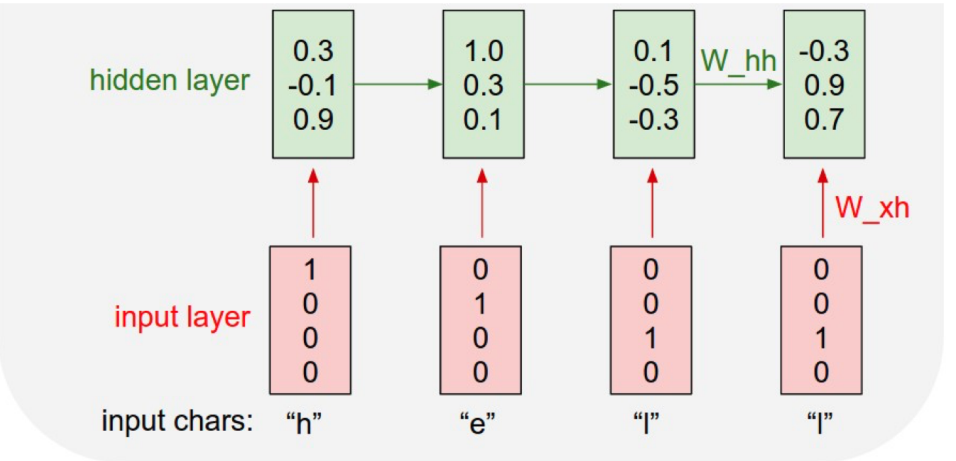
RNN Basic

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

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

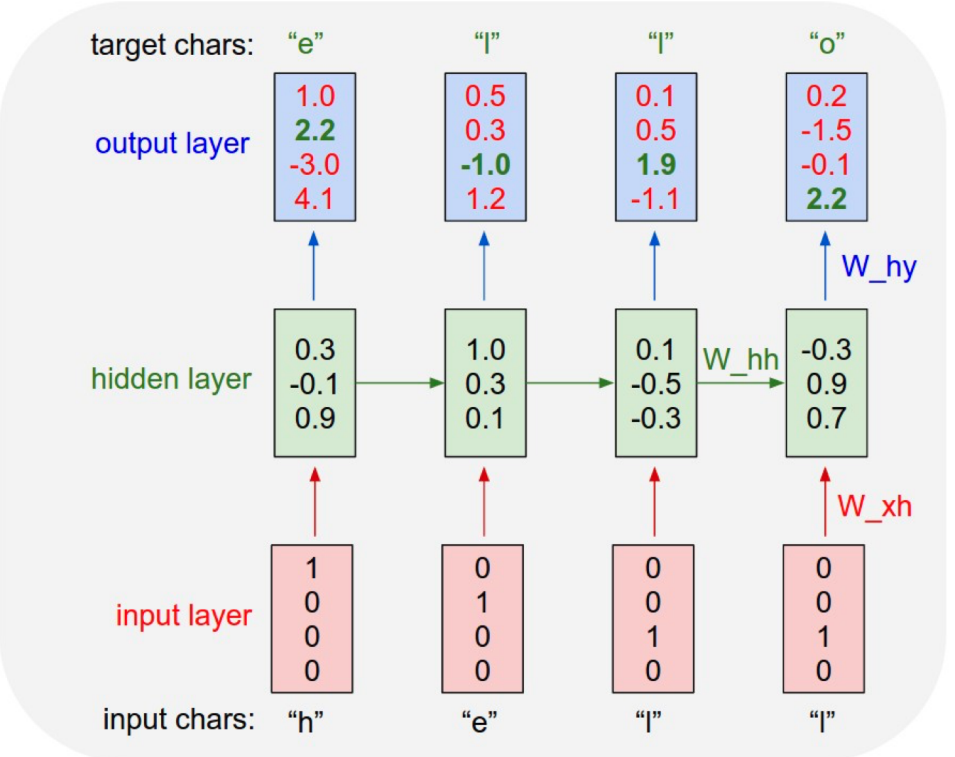


RNN Basic

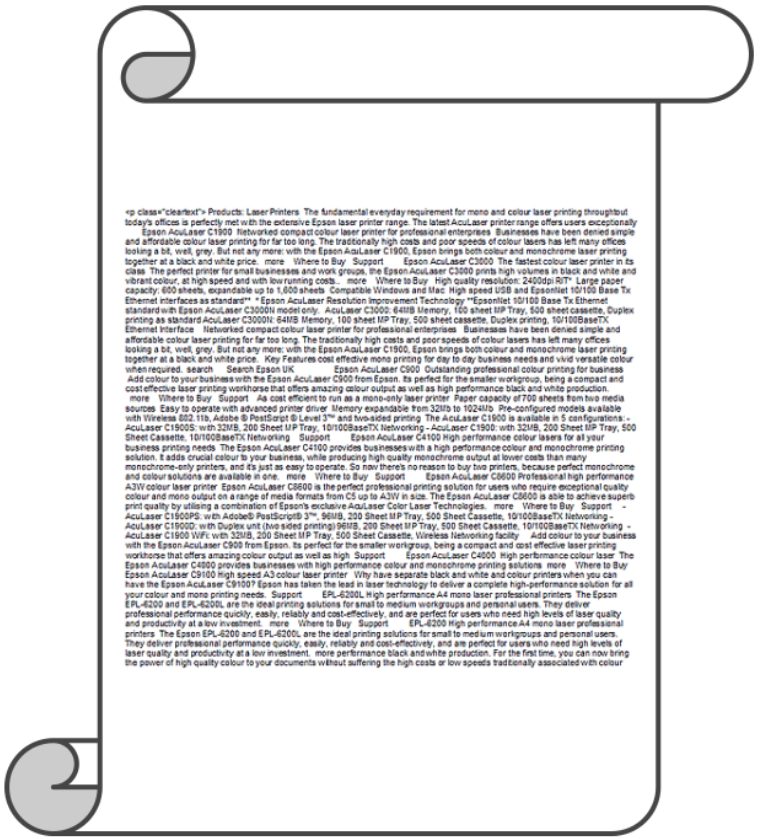
Character-level language model example

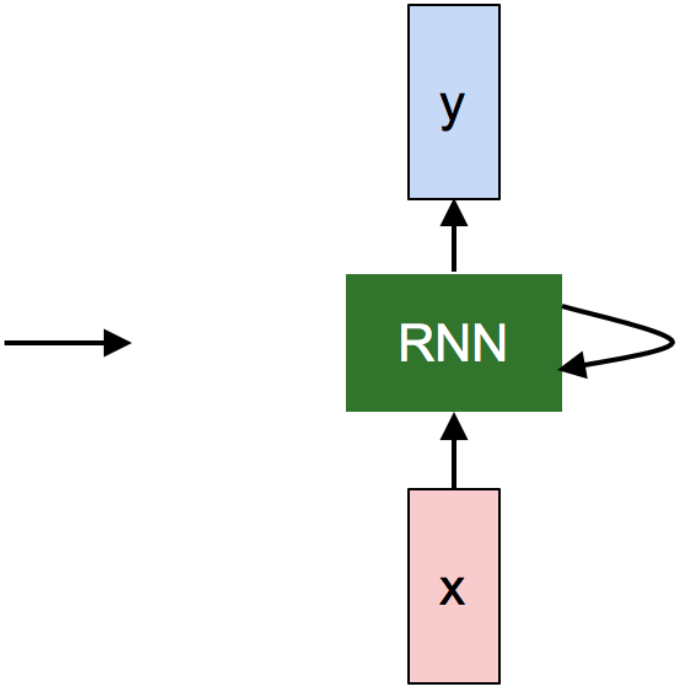
Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



RNN Basic


<p class="cleartext"> Products: Laser Printers The fundamental everyday requirement for mono and colour laser printing throughtout today's offices is perfectly met with the extensive Epson laser printer range. The latest AcuLaser printer range offers users exceptionally Epson AcuLaser C1900 Networked compact colour laser printer for professional enterprises Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more: with the Epson AcuLaser C1900, Epson brings both colour and monochrome laser printing together at a black and white price. more Where to Buy Support Epson AcuLaser C3000 The fastest colour laser printer in its class The perfect printer for small businesses and work groups, the Epson AcuLaser C3000 prints high volumes in black and white and vibrant colour, at high speed and with low running costs.. more Where to Buy High quality resolution: 2400dpi RIT* Large paper capacity: 600 sheets, expandable up to 1,600 sheets Compatible Windows and Mac High speed USB and EpsonNet 10/100 Base Tx Ethernet interfaces as standard** * Epson AcuLaser Resolution Improvement Technology **EpsonNet 10/100 Base Tx Ethernet standard with Epson AcuLaser C3000N model only. AcuLaser C3000: 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing as standard AcuLaser C3000N: 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing, 10/100BaseTX Ethernet Interface Networked compact colour laser printer for professional enterprises Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more: with the Epson AcuLaser C1900, Epson brings both colour and monochrome laser printing together at a black and white price. Key Features cost effective mono printing for day to day business needs and vivid versatile colour when required. search Search Epson UK Epson AcuLaser C900 Outstanding professional colour printing for business Add colour to your business with the Epson AcuLaser C900 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high performance black and white production. more Where to Buy Support As cost efficient to run as a mono-only laser printer Paper capacity of 700 sheets from two media sources Easy to operate with advanced printer driver Memory expandable from 32Mb to 1024Mb Pre-configured models available with Wireless 802.11b, Adobe® PostScript® Level 3™ and two-sided printing The AcuLaser C1900 is available in 5 configurations: - AcuLaser C1900S: with 32MB, 200 Sheet MP Tray, 10/100BaseTX Networking - AcuLaser C1900: with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking Support Epson AcuLaser C4100 High performance colour lasers for all your business printing needs The Epson AcuLaser C4100 provides businesses with a high performance colour and monochrome printing solution. It adds crucial colour to your business, while producing high quality monochrome output at lower costs than many monochrome-only printers, and it's just as easy to operate. So now there's no reason to buy two printers, because perfect monochrome and colour solutions are available in one. more Where to Buy Support Epson AcuLaser C8600 Professional high performance A3/W colour laser printer Epson AcuLaser C8600 is the perfect professional printing solution for users who require exceptional quality colour and mono output on a range of media formats from C5 up to A3/W in size. The Epson AcuLaser C8600 is able to achieve superb print quality by utilising a combination of Epson's exclusive AcuLaser Color Laser Technologies. more Where to Buy Support - AcuLaser C1900PS: with AdobeÂ® PostScriptÂ® 3™, 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - AcuLaser C1900D: with Duplex unit (two sided printing) 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - AcuLaser C1900 WiFi: with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, Wireless Networking facility Add colour to your business with the Epson AcuLaser C900 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high Support Epson AcuLaser C4000 High performance colour laser The Epson AcuLaser C4000 provides businesses with high performance colour and monochrome printing solutions more Where to Buy Epson AcuLaser C9100 High speed A3 colour laser printer Why have separate black and white and colour printers when you can have the Epson AcuLaser C9100? Epson has taken the lead in laser technology to deliver a complete high-performance solution for all your colour and mono printing needs. Support EPL-6200L High performance A4 mono laser professional printers The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more Where to Buy Support EPL-6200 High performance A4 mono laser professional printers The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more performance black and white production. For the first time, you can now bring the power of high quality colour to your documents without suffering the high costs or low speeds traditionally associated with colour


y
RNN
x

RNN Basic

at first:

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

train more

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

train more

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

train more

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

RNN Python

https://goo.gl/5npVuw

RNN Python

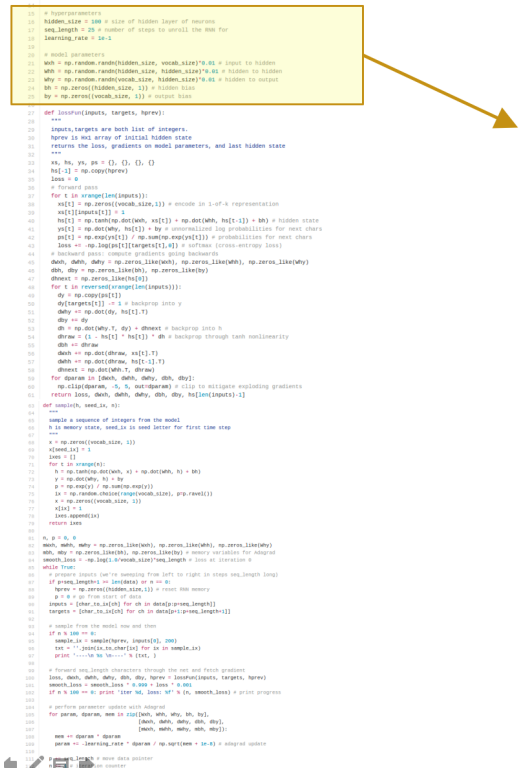
[min-char-rnn.py gist]

Data I/O

```
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

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

RNN Python

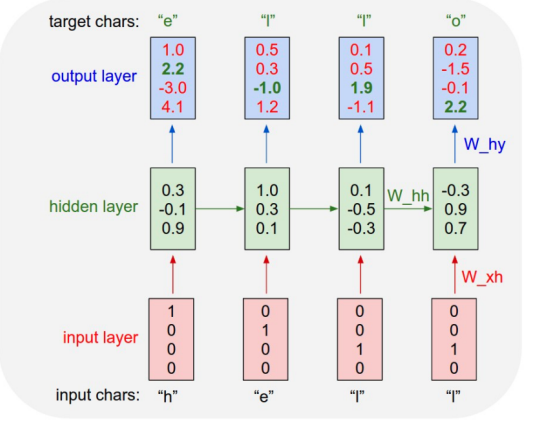
[min-char-rnn.py gist]



Initializations

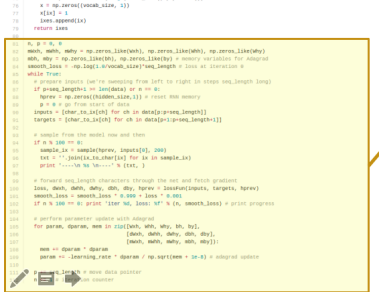
```
15  # hyperparameters
16  hidden_size = 100 # size of hidden layer of neurons
17  seq_length = 25 # number of steps to unroll the RNN for
18  learning_rate = 1e-1
19
20  # model parameters
21  Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22  Whh = np.random.randn(hidden_size,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

recall:



RNN Python

[min-char-rnn.py gist]

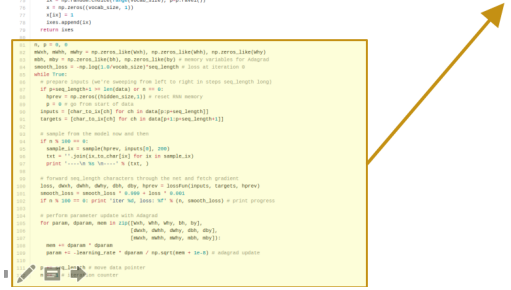


Main loop

```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \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
```

RNN Python

[min-char-rnn.py gist]



Main loop

```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \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
```

RNN Python

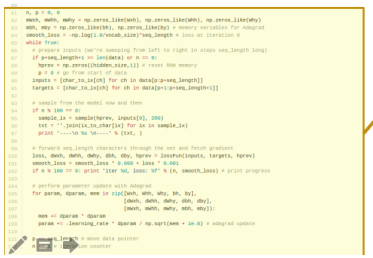
[min-char-rnn.py gist]

Main loop

```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \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
```

RNN Python

[min-char-rnn.py gist]

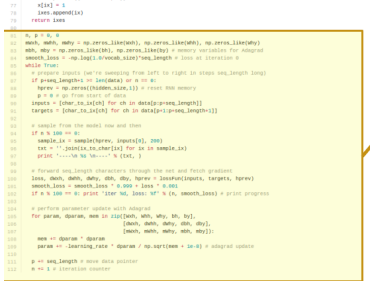


Main loop

```
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]



Main loop

```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \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
```

RNN Python

[min-char-rnn.py gist]

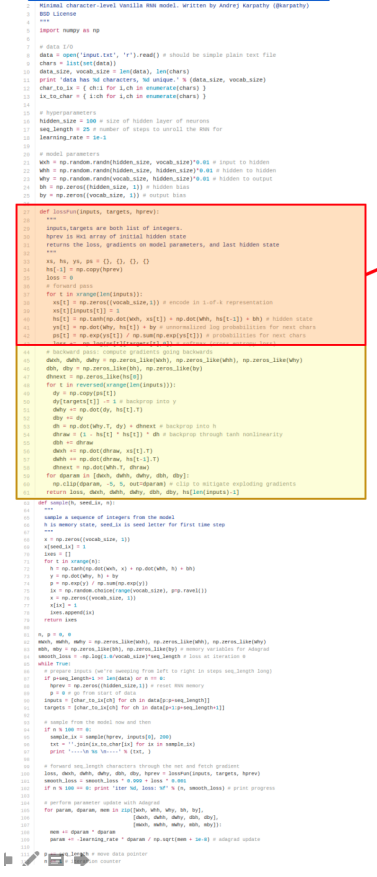
Loss function

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

```
27  def lossFun(inputs, targets, hprev):
28    """
29    inputs,targets are both list of integers.
30    hprev is Hx1 array of initial hidden state
31    returns the loss, gradients on model parameters, and last hidden state
32    """
33    xs, hs, ys, ps = {}, {}, {}, {}
34    hs[-1] = np.copy(hprev)
35    loss = 0
36    # forward pass
37    for t in xrange(len(inputs)):
38      xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39      xs[t][inputs[t]] = 1
40      hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
41      ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
42      ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43      loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
44    # backward pass: compute gradients going backwards
45    dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
46    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47    dhnext = np.zeros_like(hs[0])
48    for t in reversed(xrange(len(inputs))):
49      dy = np.copy(ps[t])
50      dy[targets[t]] -= 1 # backprop into y
51      dWhy += np.dot(dy, hs[t].T)
52      dby += dy
53      dh = np.dot(Why.T, dy) + dhnext # backprop into h
54      dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55      dbh += dhraw
56      dWxh += np.dot(dhraw, xs[t].T)
57      dWhh += np.dot(dhraw, hs[t-1].T)
58      dhnext = np.dot(Whh.T, dhraw)
59    for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
60      np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

RNN Python

[min-char-rnn.py gist]



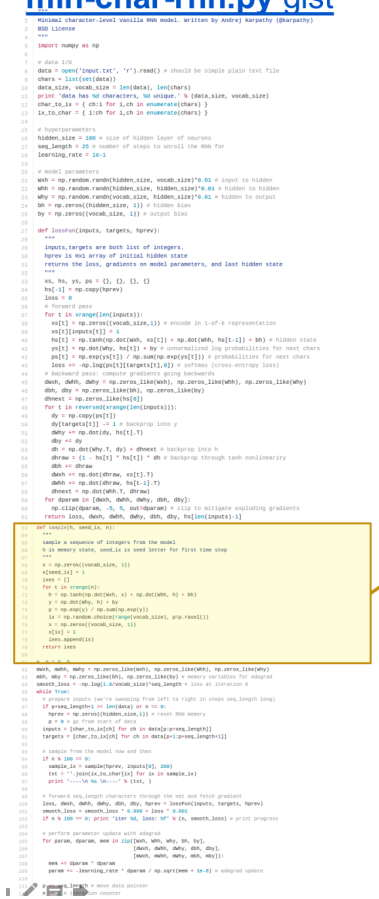
```python
def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward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

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

Softmax classifier

RNN Python

[min-char-rnn.py gist]



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

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

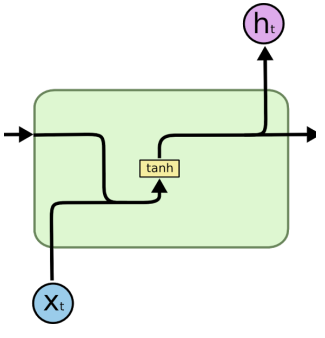
```
outputs = []
#Shape of sentences_in_batch [max_timestep * batch_size * dimension]
for words_in_batch in sentences_in_batch:
  # Shape of words_in_batch [batch_size * dimension]
  some operations here...
  outputs.append(i-th words_output)
#Shape of outputs [max_timestep * batch_size * dimension]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

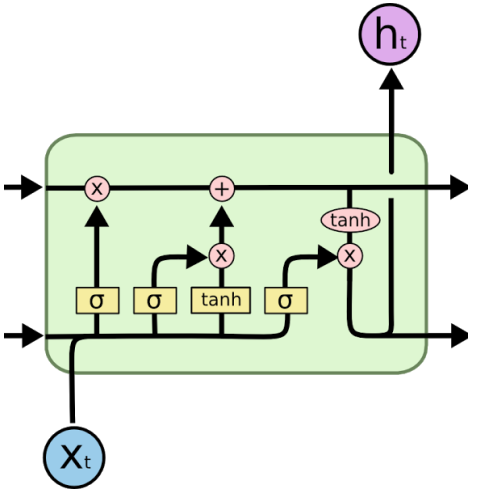


(Image credit: Colah's blog)

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

RNN Cells (Memory Units)

- LSTM Cell

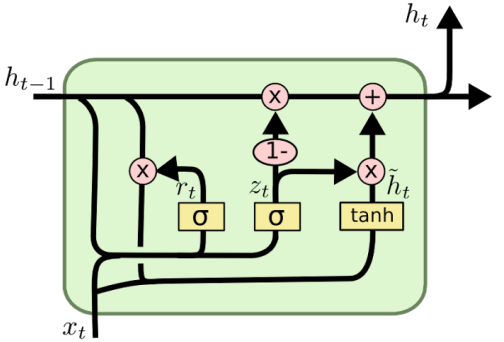


$$
\begin{aligned}
i &= \sigma(x_t U^i + s_{t-1} W^i) \\
f &= \sigma(x_t U^f + s_{t-1} W^f) \\
o &= \sigma(x_t U^o + s_{t-1} W^o) \\
g &= \tanh(x_t U^g + s_{t-1} W^g) \\
c_t &= c_{t-1} \circ f + g \circ i \\
s_t &= \tanh(c_t) \circ o
\end{aligned}
$$

(Image credit: Colah's blog)

RNN Cells (Memory Units)

- GRU Cell



$$
\begin{aligned}
z &= \sigma(x_t U^z + s_{t-1} W^z) \\
r &= \sigma(x_t U^r + s_{t-1} W^r) \\
h &= tanh(x_t U^h + (s_{t-1} \circ r) W^h) \\
s_t &= (1 - z) \circ h + z \circ s_{t-1}
\end{aligned}
$$

(Image credit: Colah's blog)

RNN Loss and Optimization

Simple RNN

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

Cross Entropy Loss

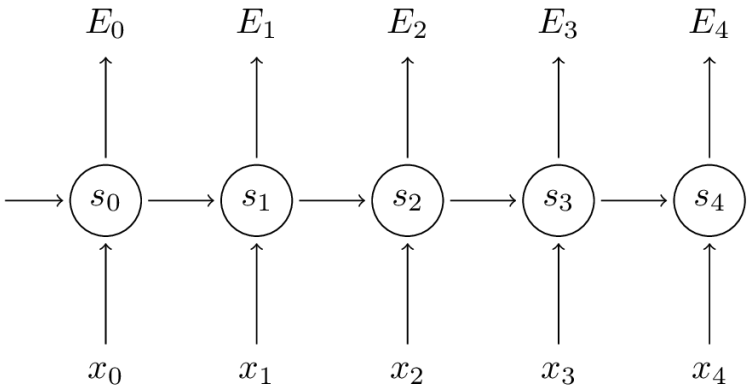
$$
\begin{aligned}
E(y_t, \hat{y}_t) &= -y_t \log \hat{y}_t \\
E(y, \hat{y}) &= -\sum_t E_t(y_t, \hat{y}_t) \\
&= -\sum_t -y_t \log \hat{y}_t
\end{aligned}
$$

RNN Loss and Optimization

BackProp

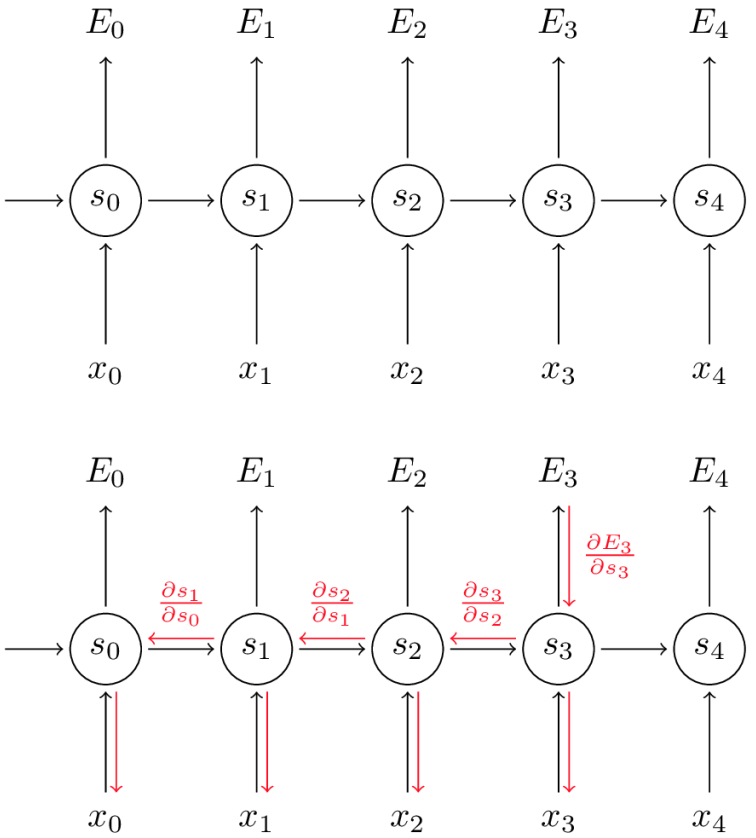
RNN Loss and Optimization

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

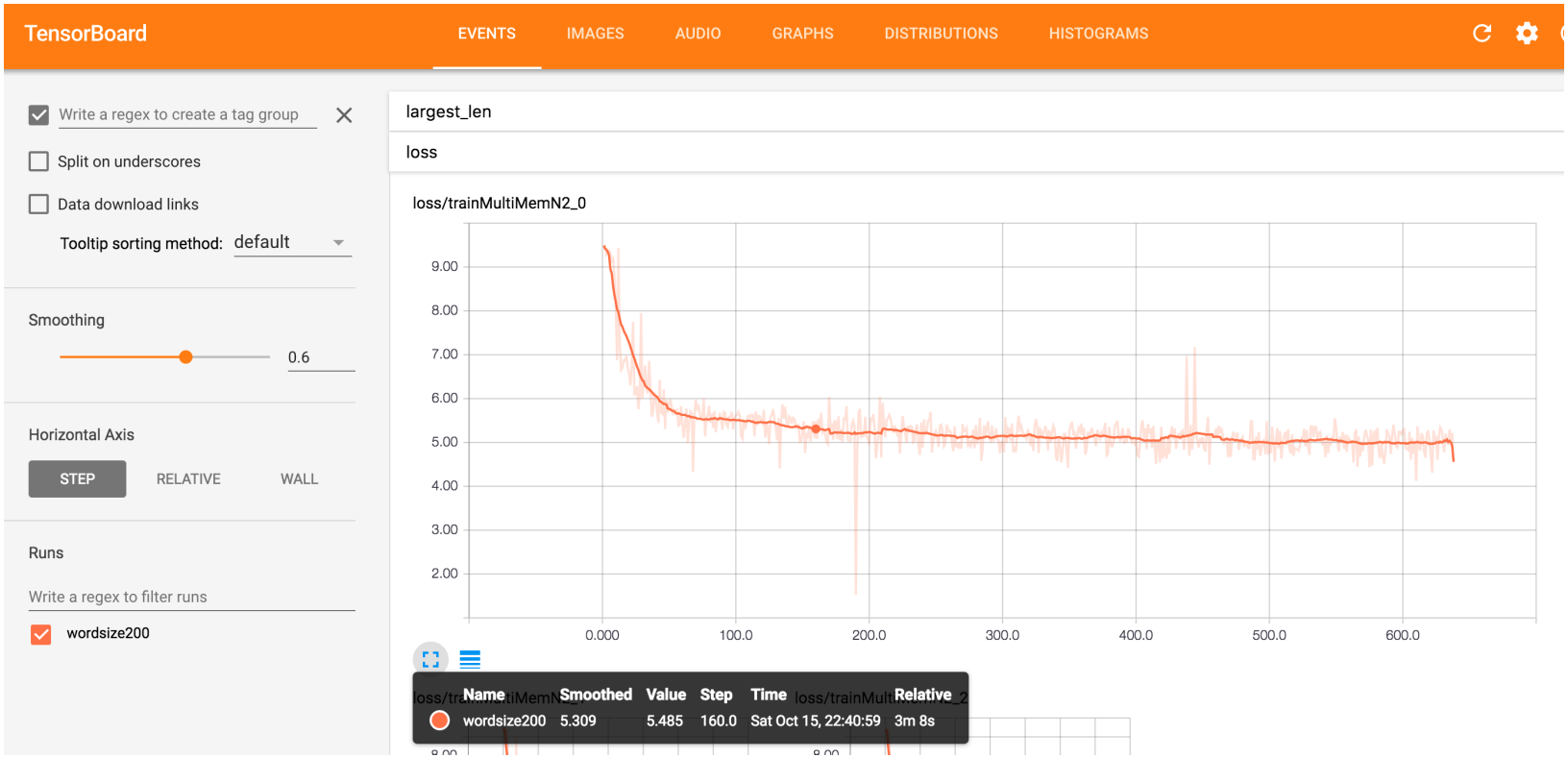
MNIST Dataset

- 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

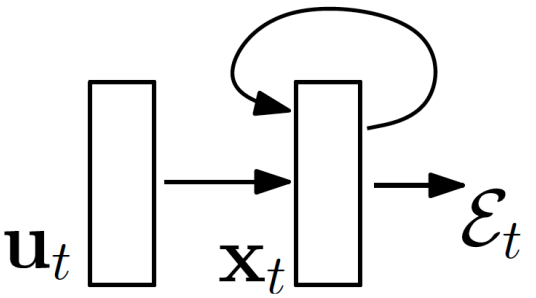


Fig. Schematic of a simple RNN

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

Difficulty of training RNNs

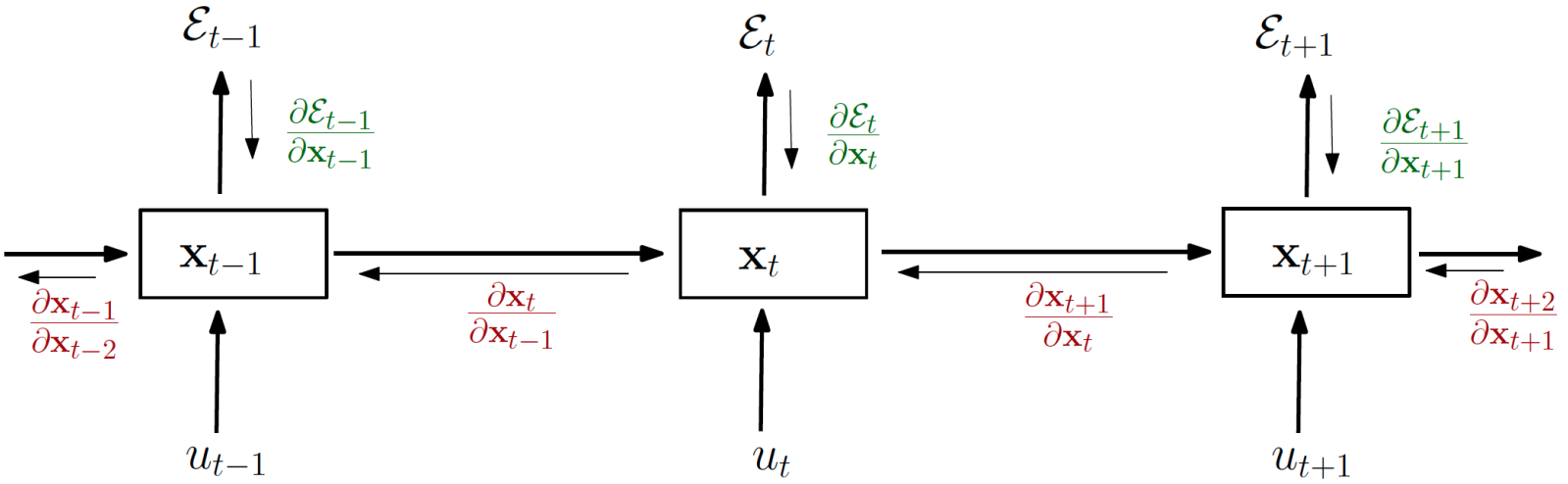


Fig. Unrolling RNNs in time by creating a copy of the model for each time step

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

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

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

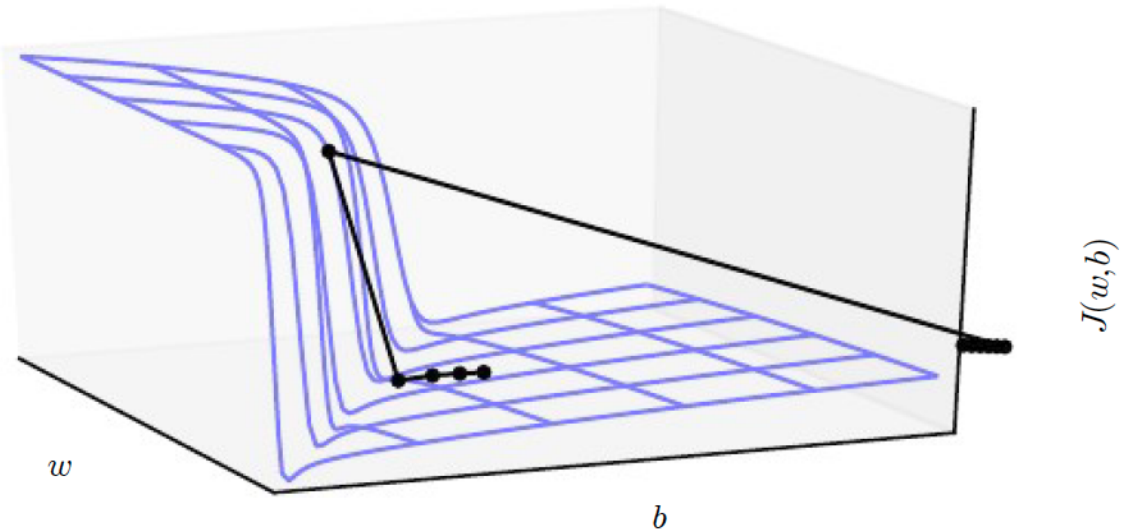
- This Equation shows that how θ at step k affects the cost at step $t > k$.
- The factors $\frac{\partial x_t}{\partial x_k}$ transport the error "in time" from step t back to step k.
- We called it **long term** for which $k \ll t$.

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 defines 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

$$
\begin{aligned}
&\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta} \\
&\textbf{if } \|\hat{\mathbf{g}}\| \geq threshold \textbf{ then} \\
&\quad \hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}} \\
&\textbf{end if}
\end{aligned}
$$

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

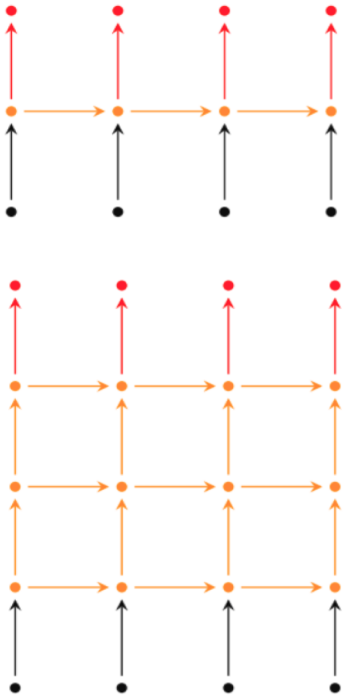
```
'''Don't use optimizer.minimize()'''

#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.

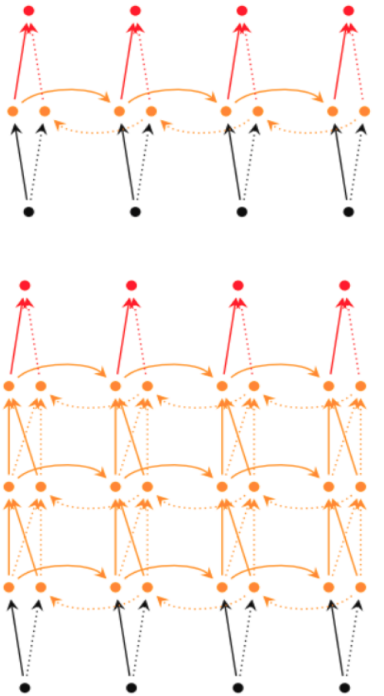
RNN



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

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

BRNN



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

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

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

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
```

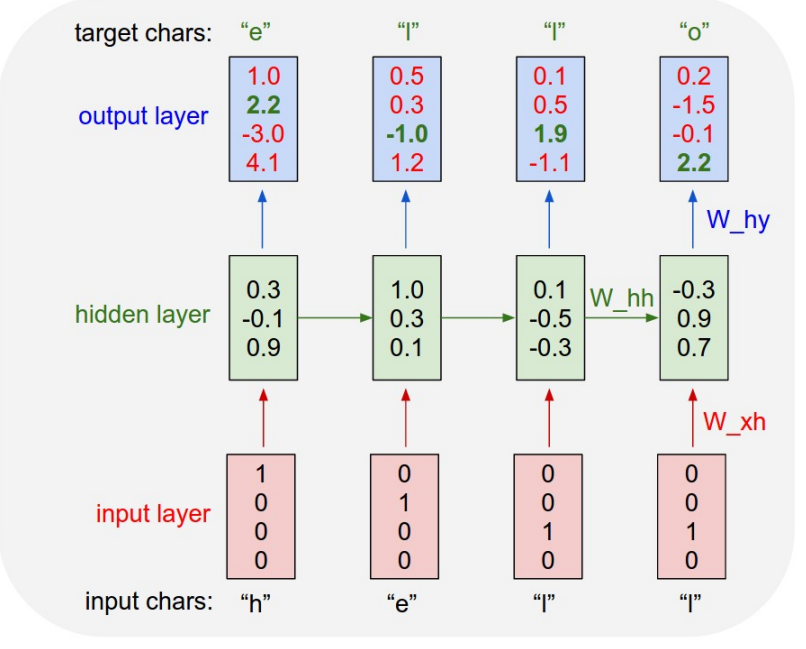
Practice : character Level RNN

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

Reference:Tensorflow-101, Karpathy's blog

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:
    cPickle.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_variabl
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)
# Weigths
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_timk_ops, *mask, struct filt, unsigned int
```

More results

- Latex generation. The resulting sampled Latex almost compiles.

Proof. Omitted. □

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

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

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

.

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

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

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

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

Proof. See Spaces, Lemma ??. □

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

The following to the construction of the lemma follows.

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

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

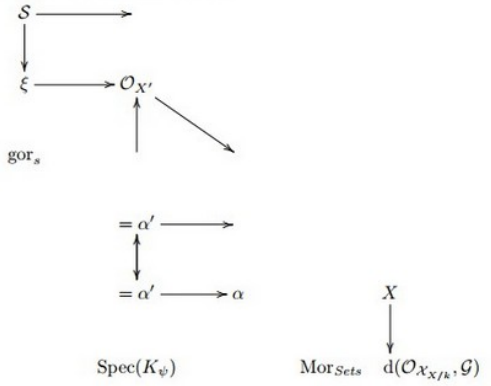
be a morphism of algebraic spaces over S and Y.

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

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

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

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



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $f_\star$. 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 $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_t}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

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

(Image credit: Karpathy's blog)

More results

- Latex generation. The resulting sampled Latex almost compiles.

(Image credit: Karpathy's blog)

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 울부짓었다
투명드래곤은 졸라짱쎄서 드래곤중에서 최강이엇다
신이나 마족도 이겼따 다덤벼도 이겼따 투명드래곤은
새상에서 하나였다 어쨌든 걔가 울부짓었다

"으악 제기랄 도망가자"

발록들이 도망갔다 투명드래곤이 짱이었따
그래서 발록들은 도망간 것이다

껴속
```

```
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
        formfile = fromfile + ".unknown"
    elif pred['confidence'] < 0.9:
        print ("WARNING! Unsured encofing! : %s = %s / %s")
        % (fromfile, pred['confidence'], pred['encoding'])
        formfile = 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

```
# Downloded 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 (" [%s] 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...
ㅇㅕᴥㄹㅓᴥㅂㅡㄴᴥ ㅈㅐᴥㄱㅏᴥ ㄷㅡᴥㄷㅣᴥㅇㅓᴥ ㄱㅡㄹᴥㅇㅡ
```

- 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_variabl
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'ㅇㅗ웍ㄴㅡㄹ웍 '):
    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, loss 1.391

```
투명드래곤 선봇랐터 조노 눅웅가다 가다망어흔

 즈시가 진장간 타어져 질롱해드름대 뒤크은 그기각드어레 무됨데~?~

수저질개널 진따 은이얼에주야나틔ㅕㅏㅆ다!.

"투명드래곤오 시봑임언당.


뚜크이~!!!

티멍드른곤언 셚ㄱㅂㄹ에 수대아오가젹핵말타 숙테몋에 ?~
삽승 때개근!

셰복 막5ㅋ~ㄹㅋㄱ 바터리자기짬툐 투명드래곤았본비잇다
야 처차들을 이개이사 하누빠니리 실낵나가 함산의 지치 망"
```

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 미엿한 즉기고치를 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 꽃고 진생을 따라가섰다.
  "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 여찌하고
꾸짖어서 펴로 어떱게 여겨 공경에 지방 읊으셨으니 어찌(삼 되는 말이옵니까?"
  이를 경공자들이 참은케 눌러져기 크리리오?"
  하고, 이곳으로벼 생각하며 눈우면제신 1등해 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 곹오는 사음에 침녀 무매로다 인사들이 대사께 돌아가도록 하라."
  이 아가서는 새 돛아오니, 어찌 가히 변천학장, 해로다."
  하는데, 전일에 용명을 튼 보해하며 가약을 이루거늘 말하기
대부인을 목안하샀던 것에 그 뮤은 꿇어였다.
```

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.