Lecture 10:

Recurrent Neural Networks
http://mtyka.github.io/deepdream/2016/02/05/bilateral-class-vis.html
Recurrent Networks offer a lot of flexibility:

- **one to one**
- **one to many**
- **many to one**
- **many to many**

Vanilla Neural Networks
Recurrent Networks offer a lot of flexibility:

- **One to one**: Model predicts a single output for a single input.
- **One to many**: Model predicts a sequence of outputs for a single input.
- **Many to one**: Model predicts a single output for a sequence of inputs.
- **Many to many**: Model predicts a sequence of outputs for a sequence of inputs.

**Example**: Image Captioning

- **Image** -> **Sequence of words**
Recurrent Networks offer a lot of flexibility:

- one to one
- one to many
- many to one
- many to many

E.g., Sentiment Classification:
sequence of words -> sentiment
Recurrent Networks offer a lot of flexibility:

- one to one
- one to many
- many to one
- many to many

e.g. Machine Translation
seq of words -> seq of words
Recurrent Networks offer a lot of flexibility:

- **one to one**
- **one to many**
- **many to one**
- **many to many**

E.g. **Video classification on frame level**
Sequential Processing of fixed inputs

Multiple Object Recognition with Visual Attention, Ba et al.
Sequential Processing of fixed outputs

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

usually want to predict a vector at some time steps
Recurrent Neural Network

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

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

new state  old state  input vector at some time step
some function with parameters $W$
Recurrent Neural Network

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

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

Notice: the same function and the same set of parameters are used at every time step.
(Vanilla) Recurrent Neural Network

The state consists of a single “hidden” vector $h$:

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

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

$$y_t = W_{hy} h_t$$
Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
Character-level language model example

Vocabulary: [h, e, l, o]

Example training sequence: “hello”
Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
```python
def sample(x, seed_ix, n):
    # Sample a sequence of integers from the model
    # n is the length of the sequence.
    # x is the integer encoded for the first character
    x = np.zeros([vocab_size, 1])
    x[seed_ix, 0] = 1
    states = []
    for t in range(n):
        x, states = model.predict(x, states=states, return_state=True)
        x = x[0,0] = np.random.choice(vocab_size, p=states[1])
    return states[0] + [x]
```

(https://gist.github.com/karpathy/d4dee566867f8291f086)
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License

```python
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
cchars = list(set(data))
data_size, vocab_size = len(data), len(cchars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(cchars) }
ix_to_char = { i:ch for i,ch in enumerate(cchars) }
```
# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_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
Main loop

```python
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mbh = 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,)) # 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('


'.format(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, mbh]):
    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

```python
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(why)
mbh, mbhy = 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('-----
{} %
-----'.format(txt))

    # forward seq_length characters through the net and fetch gradient
    loss, dWxh, dWhh, dWhy, dbh, dbhy, 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, dbhy],
                                   [mWxh, mWhh, mWhy, mbh, mbhy]):
        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

```python
n, p = 0, 0
mxh, mWh, mWy = 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 >= 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('---
% %

# forward seq_length characters through the net and fetch gradient
loss, dWxh, dWhh, dWy, 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, dWy, dbh, dby],
                              [mxh, mWhh, mWyh, 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

n, p = 0, 0
wxh, wxh, why = np.zeros_like(wxh), np.zeros_like(wxh), np.zeros_like(why)
bh, bby = np.zeros_like(bh), np.zeros_like(bby) # 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 '-----
        print '%(txt, )

    # forward seq_length characters through the net and fetch gradient
    loss, dwxh, dwxh, dwhy, dbh, dbby = 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, wxh, why, bh, bby],
                                  [dwxh, dwxh, dwhy, dbh, dbby],
                                  [mxwxh, mxwxh, mxwhy, mbh, mbby]):
        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

n, p = 0, 0

\[ \text{mWxh, mWhh, mWhy} = \text{np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)} \]

\[ \text{mbh, mby} = \text{np.zeros_like(bh), np.zeros_like(by)} \]  # memory variables for Adagrad

\[ \text{smooth_loss} = -\text{np.log(1.0/vocab_size)}^{\text{seq_length}} \]  # 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:

\[ \text{hprev} = \text{np.zeros((hidden_size, 1))} \]  # reset RNN memory

\[ \text{p} = 0 \]  # 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:

\[ \text{sample_ix} = \text{sample(hprev, inputs[0], 260)} \]

\[ \text{txt} = \text{'}'.join(ix_to_char[ix] for ix in sample_ix) \]

\[ \text{print '----\n% \----' % (txt, )} \]

# forward seq_length characters through the net and fetch gradient

loss, dWxh, dWhh, dWhy, dbh, db, dby, hprev = lossFun(inputs, targets, hprev)

\[ \text{smooth_loss} = \text{smooth_loss} \ast 0.999 + \text{loss} \ast 0.001 \]

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],

\[ [dWxh, dWhh, dWhy, dbh, db, dby], \]

\[ [mWxh, mWhh, mWhy, mbh, mby]]: \]

\[ \text{mem} \ast param \ast dparam \]

\[ \text{param} \ast -\text{learning_rate} \ast dparam / \text{np.sqrt(mem + 1e-8)} \]  # adagrad update

p += seq_length  # move data pointer
n += 1  # iteration counter
Loss function
- forward pass (compute loss)
- backward pass (compute param gradient)

```python
def lossFun(inputs, targets, hprev):
    
    inputs, targets are both list of integers.
    hprev is Kx1 array of initial hidden state
    returns the loss, gradients on model parameters, and last hidden state

    xs, hs, ys, ps = [], [], [], []
    hprev = np.copy(hprev)
    loss = 0

    # forward pass
    for t in range(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]]) # softmax (cross-entropy loss)

    # backward pass: compute gradients going backwards
    dwxh, dwhh, dwhy = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(why)
    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])
    for t in reversed(range(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] -= 1 # backprop into y
        dwhy += np.dot(dy, hs[t].T)
        dby += dy
        dh = np.dot(why.T, dy) + dhnext # backprop into h
        ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
        dhb = -ddraw
        dwhh += np.dot(ddraw, hs[t-1].T)
        dwxh += np.dot(ddraw, xs[t].T)
        dwhh += np.dot(ddraw, hs[t-1].T)
        dhnext = np.dot(dwhh, T, dhraw)

    for dparam in [dwxh, dwhh, dwhy, dbh, dby]:
        np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
    return loss, dwxh, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
```
Sonnet 116 – Let me not ...

by William Shakespeare

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

tyntd-iafhatawialhdeomt lytdws e ,tfti, astai f ogoh eoase rraranbyne 'nhthnee e plia tkrlgd 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
coaniogennnc 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 fiveing were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.

train more

"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
PANDARUS:
Alas, I think he shall be come approached and the day
When little strait would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

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

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

Second Lord:
They would be ruled after this chamber, and
My fair nouns begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

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

VIOLA:
I'll drink it.

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

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```c
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                    ((count & 0x000000000000000f) << 8);
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &offset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```
/*
 * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software Foundation,
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

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

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

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

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

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

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

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

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

“You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties,” warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire. Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: “I meant merely to say what I said.”

quote detection cell
Searching for interpretable cells

Cell sensitive to position in line:

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

line length tracking cell
Searching for interpretable cells

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

if statement cell
Searching for interpretable cells
Searching for interpretable cells

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

code depth cell
Image Captioning

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

Convolutional Neural Network
before:
\[ h = \text{tanh}(W_{xh} \times x + W_{hh} \times h) \]

now:
\[ h = \text{tanh}(W_{xh} \times x + W_{hh} \times h + W_{ih} \times v) \]
test image
test image

sample!
Test image
test image

sample

<END> token

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

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

currently:
~120K images
~5 sentences each
"man in black shirt is playing guitar."

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

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."
Preview of fancier architectures

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

Show Attend and Tell, Xu et al., 2015
RNN:

\[ h_t^l = \tanh(W^l (h_t^{l-1}, h_{t-1}^l)) \]

\( h \in \mathbb{R}^n \)

\( W^l \) \([n \times 2n]\)
RNN:

\[ h_t^l = \tanh W_l^l \left( \begin{array}{c} h_t^{l-1} \\ h_{t-1}^l \end{array} \right) \]

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

LSTM:

\[
\begin{pmatrix}
  i \\
  f \\
  o \\
  g
\end{pmatrix}
= \begin{pmatrix}
  \text{sigm} \\
  \text{sigm} \\
  \text{sigm} \\
  \text{tanh}
\end{pmatrix} W^l \left( \begin{array}{c} h_t^{l-1} \\ h_{t-1}^l \end{array} \right)
\]

\[
c_t^l = f \odot c_{t-1}^l + i \odot g
\]

\[
h_t^l = o \odot \tanh(c_t^l)
\]

depth

time
LSTM
Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

\[\begin{align*}
\mathbf{c}_t &= f \odot c_{t-1} + i \odot g \\
\mathbf{h}_t &= o \odot \tanh(c_t)
\end{align*}\]
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]
Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

\[
\begin{align*}
(i) & = \begin{pmatrix} \text{sigm} \\ \text{sigm} \end{pmatrix} W^i \begin{pmatrix} h^{l-1}_t \\ h^{l-1}_{t-1} \end{pmatrix} \\
(f) & = \begin{pmatrix} \text{sigm} \\ \text{tanh} \end{pmatrix} W^f \begin{pmatrix} h^{l-1}_t \\ h^{l-1}_{t-1} \end{pmatrix} \\
(o) & = \begin{pmatrix} \text{sigm} \\ \text{tanh} \end{pmatrix} W^o \begin{pmatrix} h^{l-1}_t \\ h^{l-1}_{t-1} \end{pmatrix} \\
g & = \begin{pmatrix} \text{tanh} \end{pmatrix} W^g \begin{pmatrix} h^{l-1}_t \\ h^{l-1}_{t-1} \end{pmatrix} \\
\end{align*}
\]

\[
\begin{align*}
c^l_t &= f \odot c^{l-1}_t + i \odot g \\
h^l_t &= o \odot \tanh(c^l_t) \\
\end{align*}
\]
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]

\[
\begin{align*}
(f) &= \begin{pmatrix} \text{sigm} \\ \text{sigm} \end{pmatrix} W^l \begin{pmatrix} h^l_{t-1} \\ h^l_{t-1} \end{pmatrix} \\
(i, o, g) &= \begin{pmatrix} \text{sigm} \\ \text{tanh} \end{pmatrix} \\
(c_t) &= f \odot c_{t-1}^l + i \odot g \\
(h_t) &= o \odot \text{tanh}(c_t^l)
\end{align*}
\]
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]

\[
\begin{align*}
    f = \text{sigmoid}(W^f(x + c)) \\
    i = \text{sigmoid}(W^i(x + c)) \\
    g = \tanh(W^g(x + c)) \\
    o = \text{sigmoid}(W^o(x + c)) \\
    c_t = f \odot c_{t-1} + i \odot g \\
    h_t = o \odot \tanh(c_t)
\end{align*}
\]
LSTM

\[
\text{one timestep}
\]

- **Input Gate (i)**
- **Forgetting Gate (f)**
- **Cell State (c)**
- **Output Gate (o)**
- **Tanh Function**

\[
\text{one timestep}
\]
RNN

state

\[
\begin{array}{ccc}
\text{f} & \text{f} & \text{f} \\
\end{array}
\]

LSTM

(ignoring forget gates)

\[
\begin{array}{ccc}
\text{f} & \text{f} & \text{f} \\
\end{array}
\]

Fei-Fei Li & Andrej Karpathy & Justin Johnson
LSTM variants and friends

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

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

\[
\begin{align*}
r_t &= \text{sigmoid}(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\
z_t &= \text{sigmoid}(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\
\tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{align*}
\]
Summary

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