29.03.2019

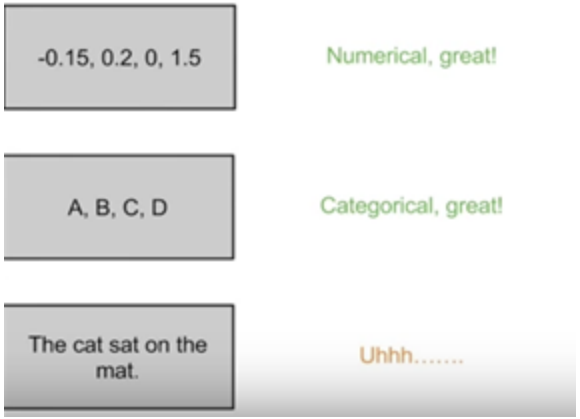
Statistical Methods in AI (CSE/ECE 471)

Lecture-20: ML for Sequential Data – Recurrent Neural Networks

Ravi Kiran

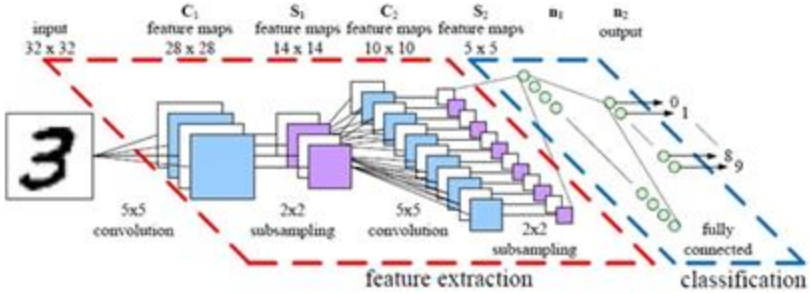
Center for Visual Information Technology (CVIT), IIIT Hyderabad

Data/Output Paradigms



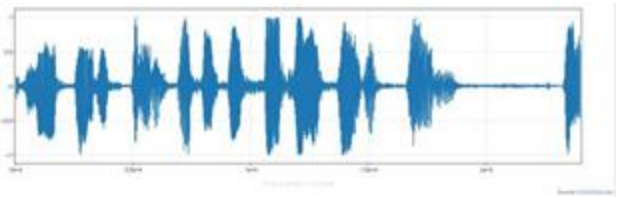
Alec Radford: https://www.youtube.com/watch?v=VINCQghQRuM

CNN Revisited



- Data is fed “in one shot”, not in parts (mostly)
  - E.g. Image of “3” above is not processed row-by-row
- Captures “spatial context”

Sequential data

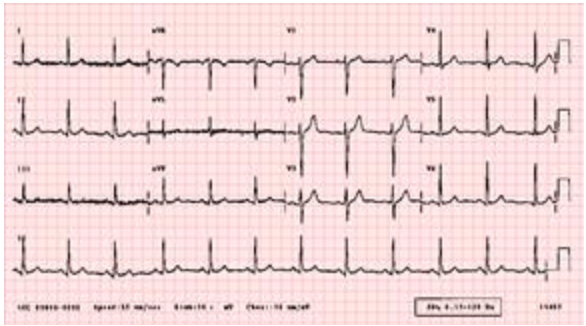


Audio

Video

O Nachiketa, after pondering well the pleasures that are or seem to be delightful, you have renounced them all. You have not taken the road abounding in wealth, where many men sink. *(Kathopanishad, II:3)*

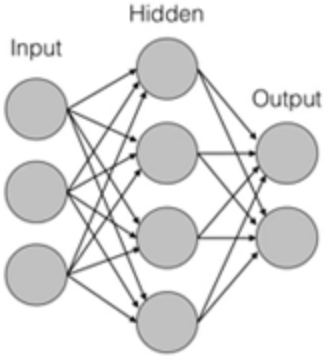
Text



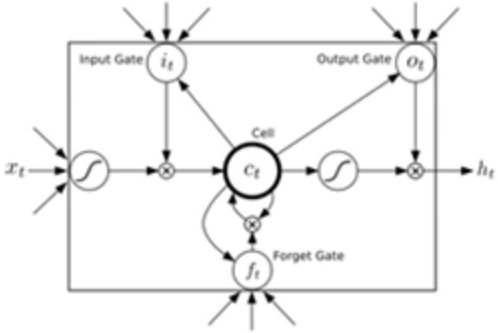
Biological

many more ……

What are RNNs for ?



- Independence
- Fixed Length



- Temporal dependencies
- Variable sequence length

https://www.youtube.com/watch?v=Ukgii7Yd_cU

Modelling a RNN

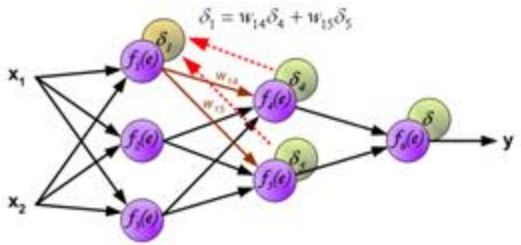
$f(n) = f(n-1) + f(n-2)$

$f(n-1) = f(n-2) + f(n-3)$

⋮

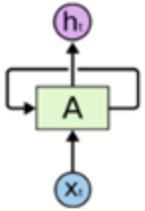
Error derivatives w.r.t weights in kth layer = ***f(*** Error derivatives from (k+1)th layer **)**

$f(n) = g(f(n-k))$



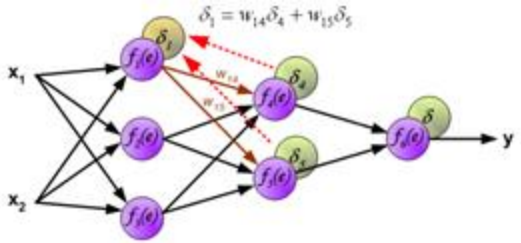
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Modelling a RNN



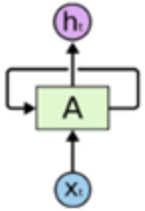
Recurrence “loop”

Error derivatives w.r.t weights in kth layer = ***f(*** Error derivatives from (k+1)th layer **)**



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

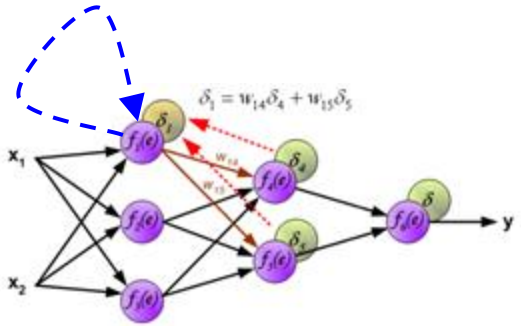
Modelling a RNN



Recurrence “loop”

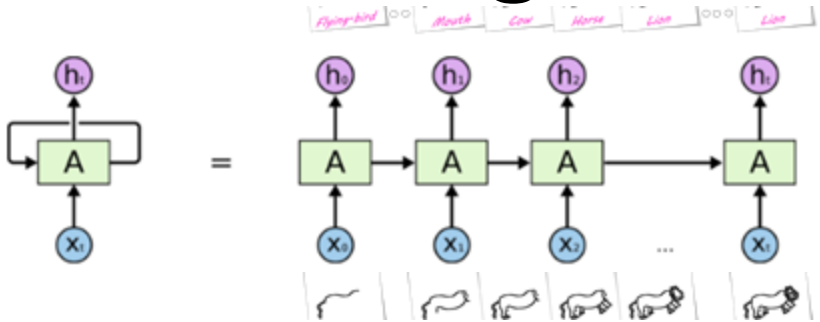
Error derivatives w.r.t weights in kth layer = ***f*(** Error derivatives from (k+1)th layer **)**

⇒ Error derivatives in terms of themselves ??!!!



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

Modelling a RNN



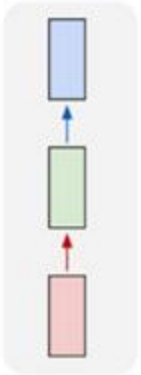
- RNN unrolled → Not “that different” from a CNN (feed-forward connections)
- But crucial differences exist !

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

Modelling a RNN

Recurrent Networks offer a lot of flexibility:

one to one

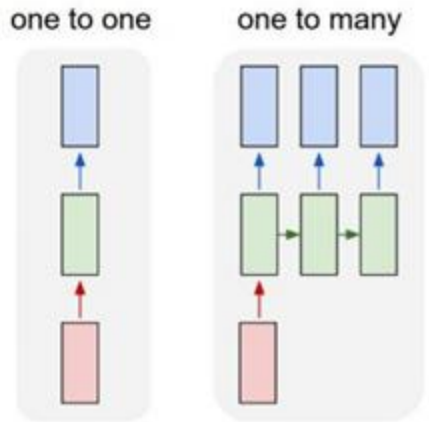


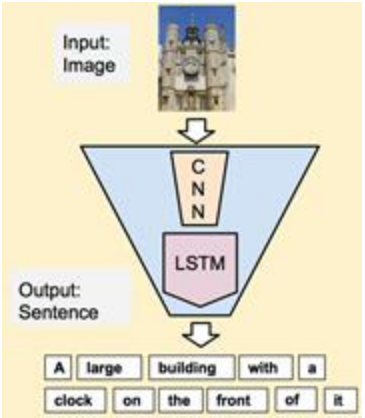
Vanilla Neural Networks

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Modelling a RNN

Recurrent Networks offer a lot of flexibility:



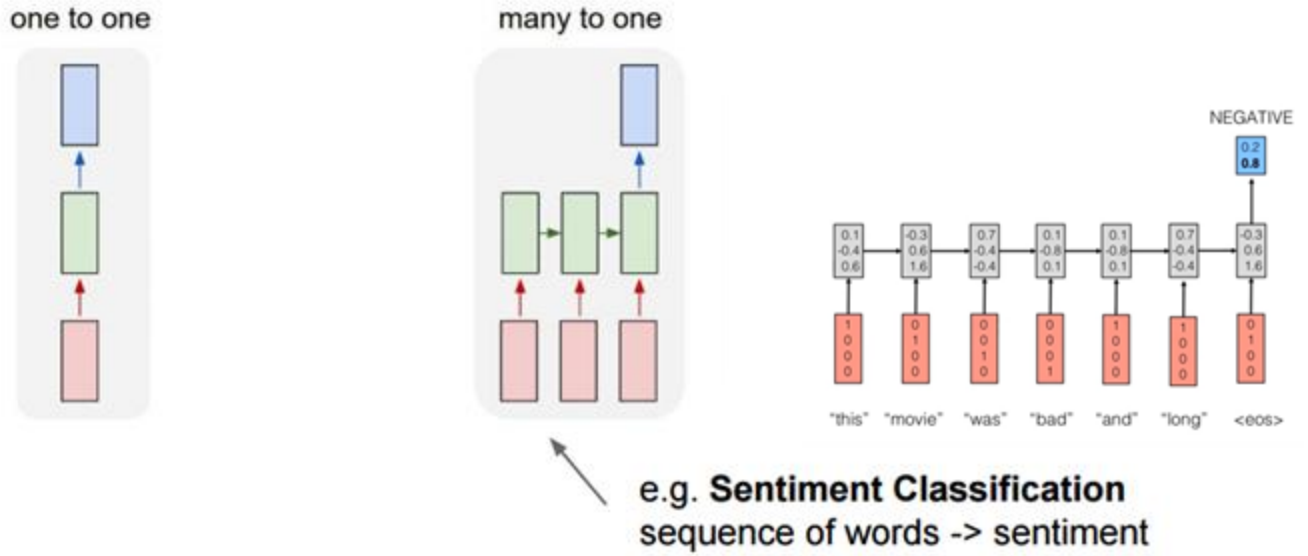


e.g. **Image Captioning**
image -> sequence of words

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Modelling a RNN

Recurrent Networks offer a lot of flexibility:

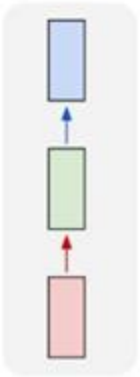


http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf, https://www.youtube.com/watch?v=Ukgii7Yd_cU

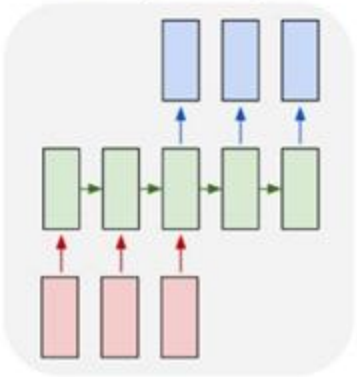
Modelling a RNN

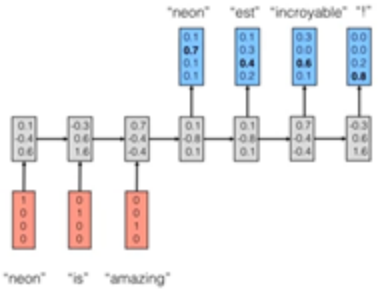
Recurrent Networks offer a lot of flexibility:

one to one



many to many



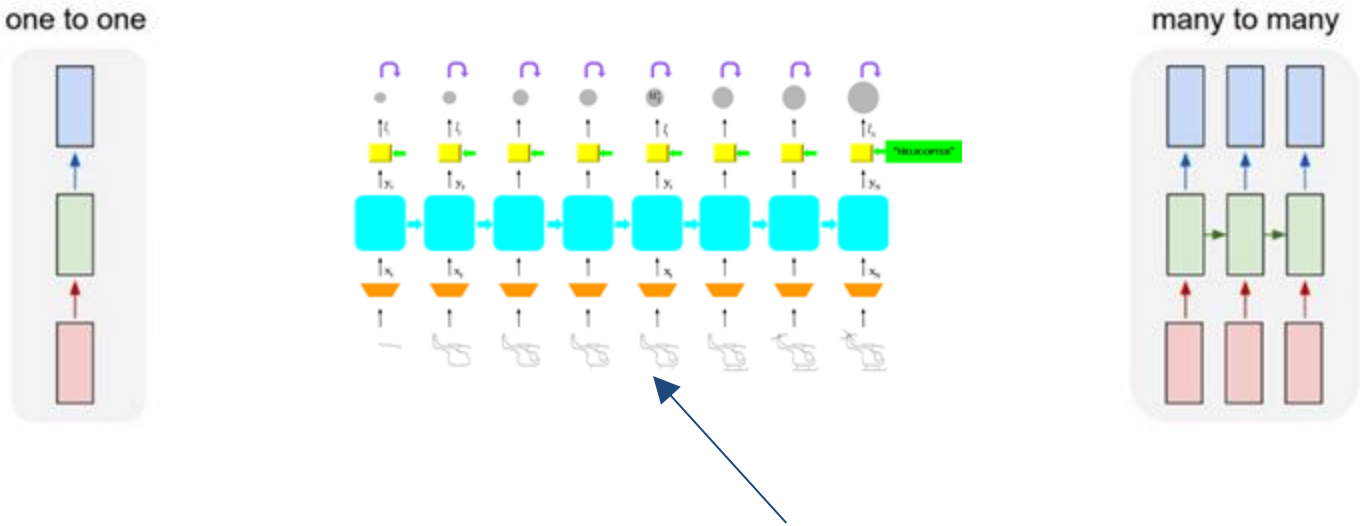


e.g. **Machine Translation**
seq of words -> seq of words

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf, https://www.youtube.com/watch?v=UkgiI7Yd_cU

Modelling a RNN

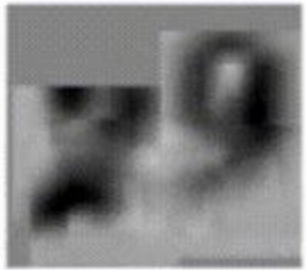
Recurrent Networks offer a lot of flexibility:



E.g. On-line freehand sketch recognition
Seq of strokes -> Seq of "guesses"

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Paradigm: Sequential “processing”



Reading house numbers

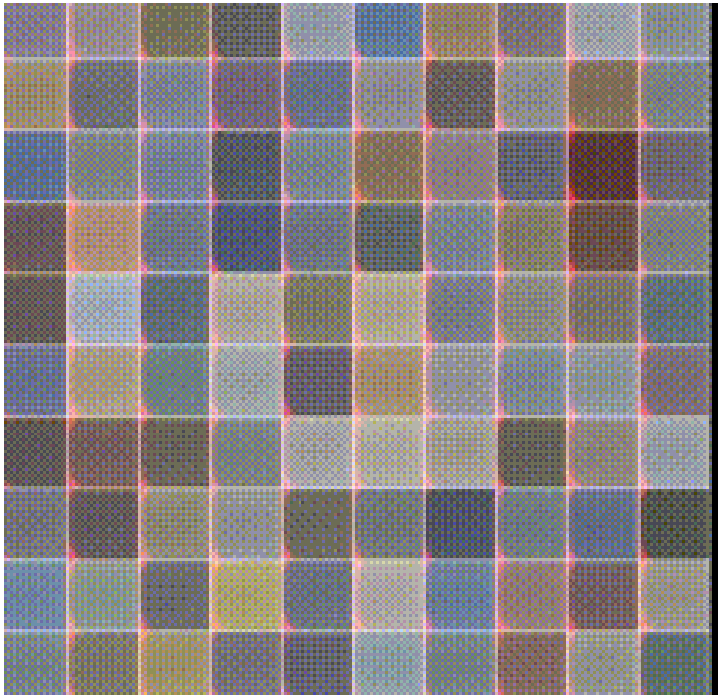
Input/Output may be “fixed”, but processing can be sequential !

Multiple Object Recognition with Visual Attention, Ba et al.

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Paradigm: Sequential “processing”

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



http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Adding color to house numbers

Recurrent Neural Network

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

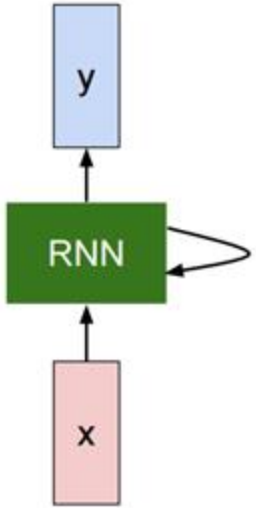
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



Task-specific abstraction for portion of sequence seen up to this point ($x_1, x_2 \ldots x_{t-1}$)

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

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

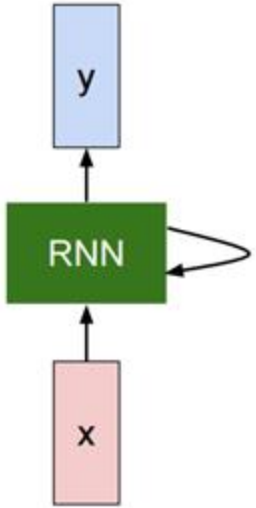
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

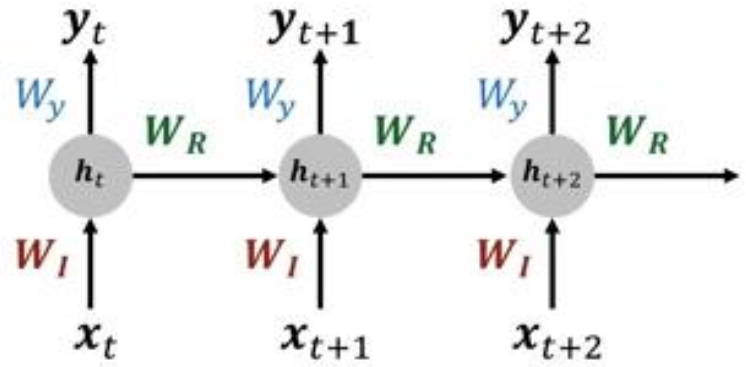
input vector at some time step



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

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

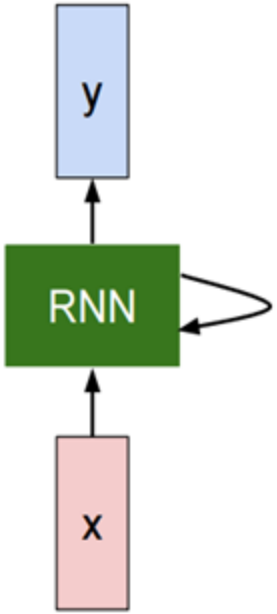


3 sets of parameters - W_I,W_y,W_R (shared for each time-step)

https://www.youtube.com/watch?v=Ukgii7Yd_cU

(Vanilla) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



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

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

$$y_t = W_{hy}h_t$$

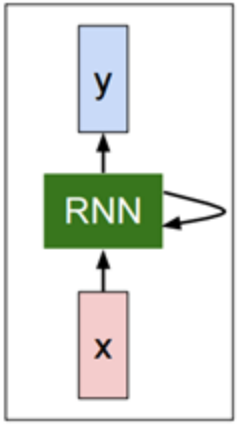
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



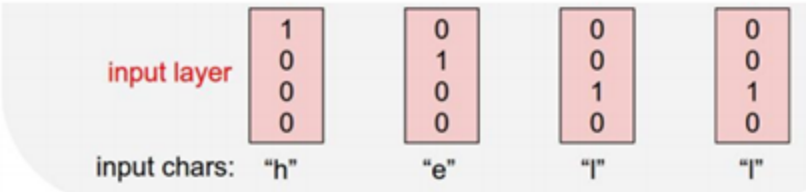
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

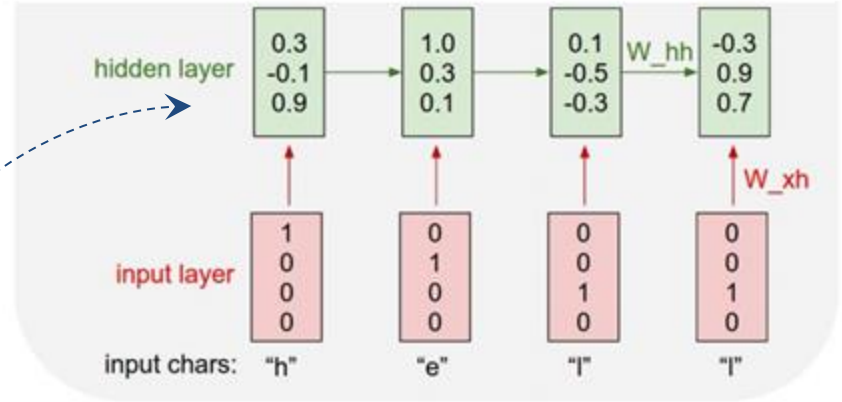
Recurrent Neural Network

Character-level language model example

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

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"



h_t → **abstraction of current input upto step t**

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

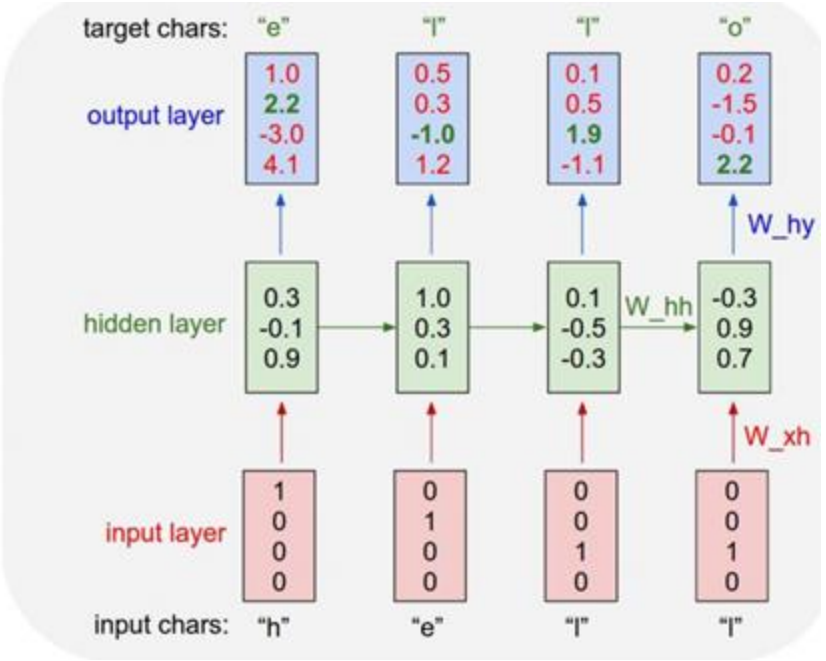
Recurrent Neural Network

$$y_t = W_{hy} h_t$$

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



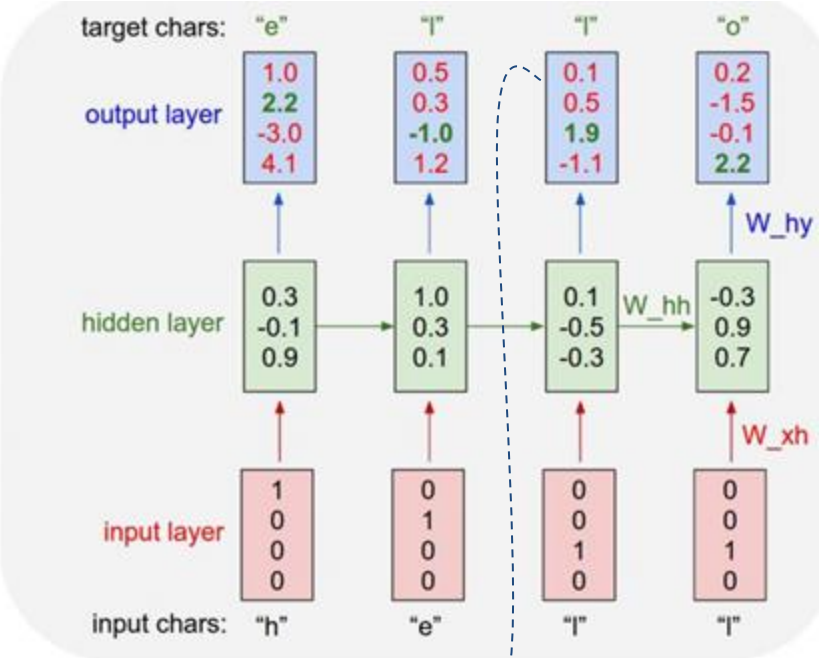
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



$p(\text{next letter} = \text{“l”} \,|\, \text{previous letters} = \{\text{“h”}, \text{“e”}, \text{“l”}\})$

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

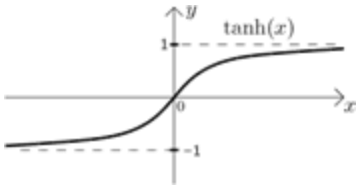
Code: https://gist.github.com/karpathy/d4dee566867f8291f086

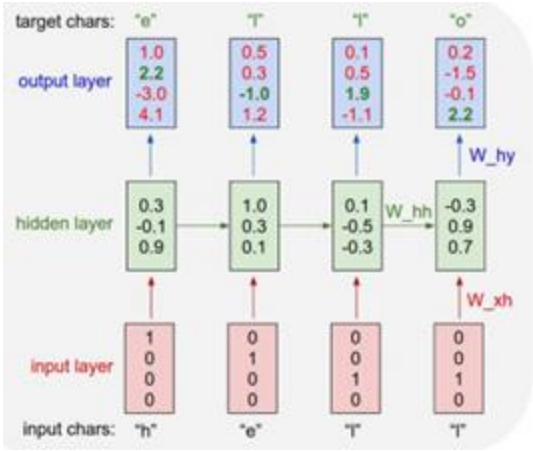
Recurrent Neural Network

```
# 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
```

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

$$y_t = W_{hy}h_t$$



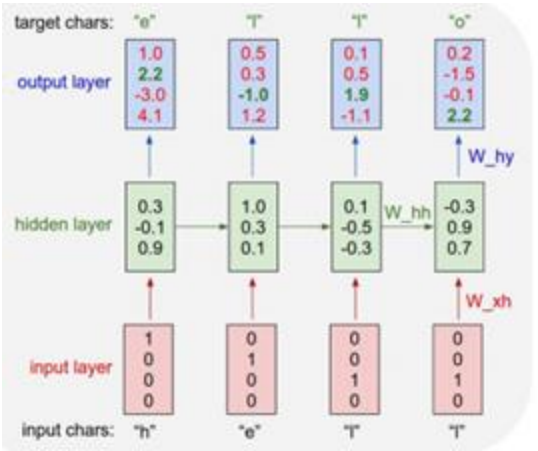


http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

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
```



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

$$y_t = W_{hy}h_t$$

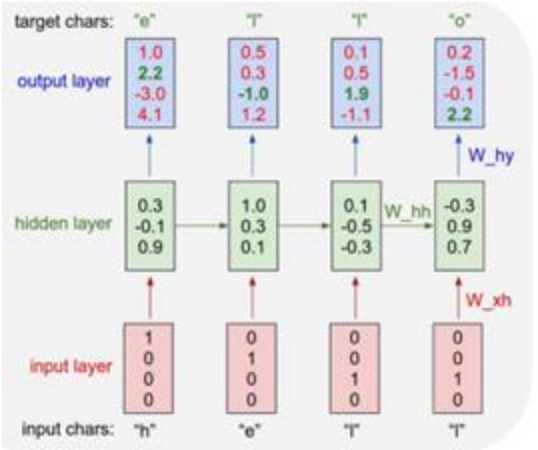
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

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
```

Forward prop



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

$$y_t = W_{hy}h_t$$

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

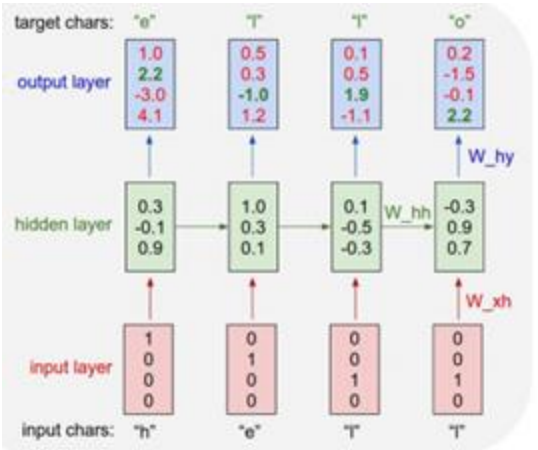
Recurrent Neural Network

```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

Forward prop

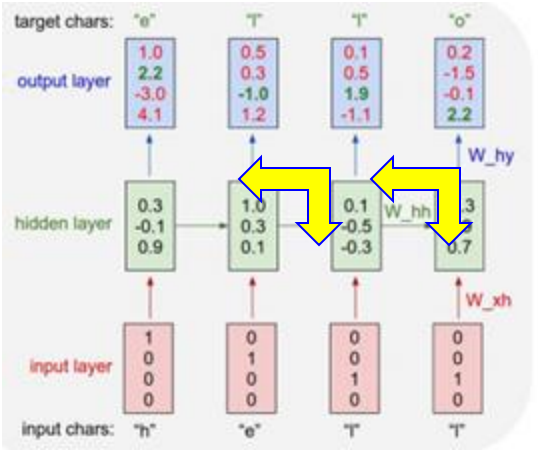


Recurrent Neural Network

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
```

Backprop



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

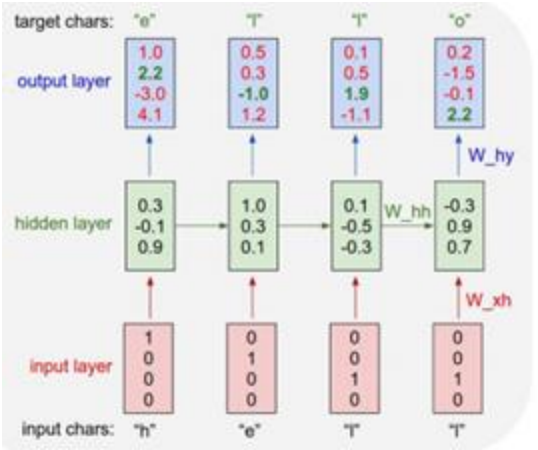
$$y_t = W_{hy}h_t$$

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

```
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
```

Generation

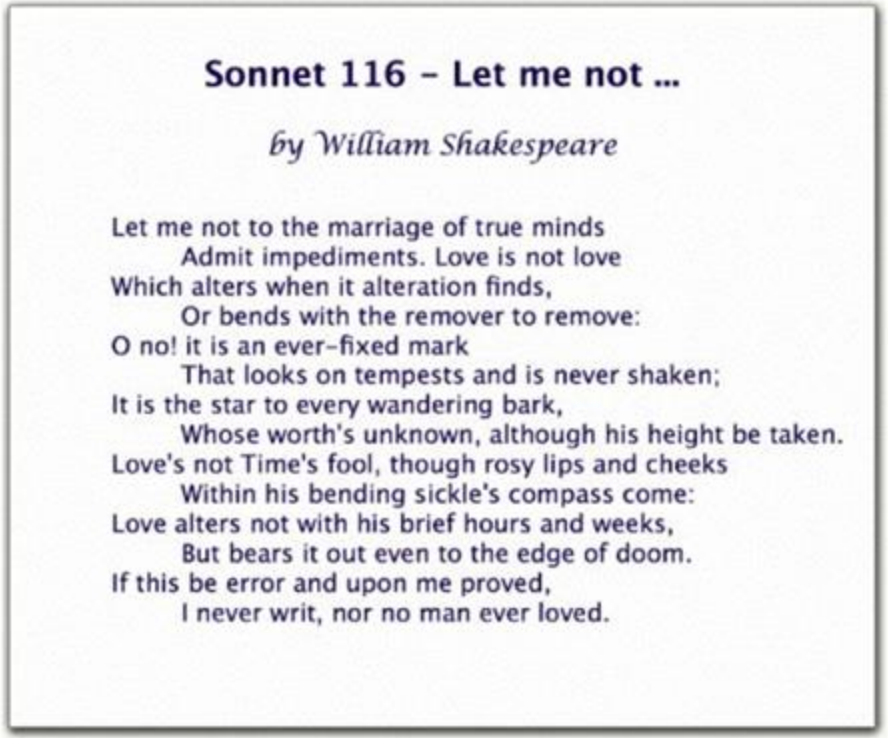


$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network



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.

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

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.
```

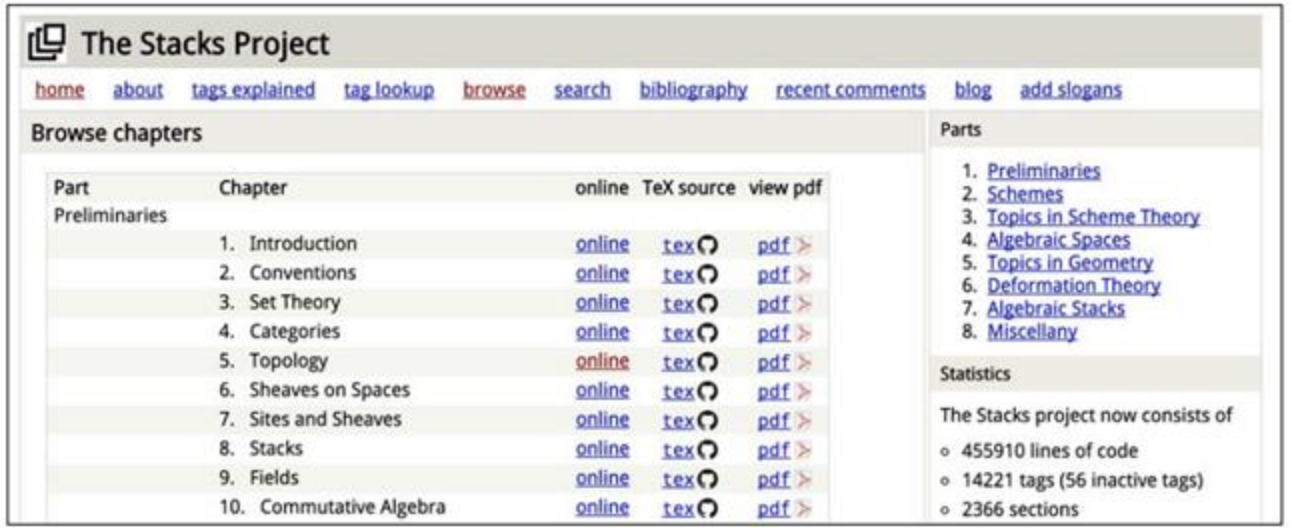
train more

```
"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.
```

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

open source textbook on algebraic geometry



The Stacks Project

home about tags explained tag lookup browse search bibliography recent comments blog add slogans

Browse chapters

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	online	tex	pdf
	2. Conventions	online	tex	pdf
	3. Set Theory	online	tex	pdf
	4. Categories	online	tex	pdf
	5. Topology	online	tex	pdf
	6. Sheaves on Spaces	online	tex	pdf
	7. Sites and Sheaves	online	tex	pdf
	8. Stacks	online	tex	pdf
	9. Fields	online	tex	pdf
	10. Commutative Algebra	online	tex	pdf

Parts

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

Statistics

The Stacks project now consists of

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

Latex source

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

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

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

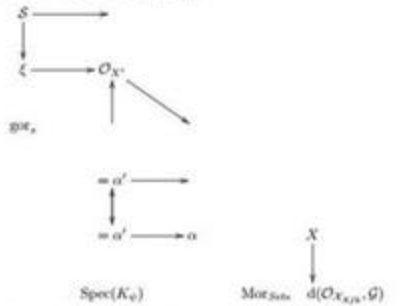
be a morphism of algebraic spaces over S and Y.

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_*. 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

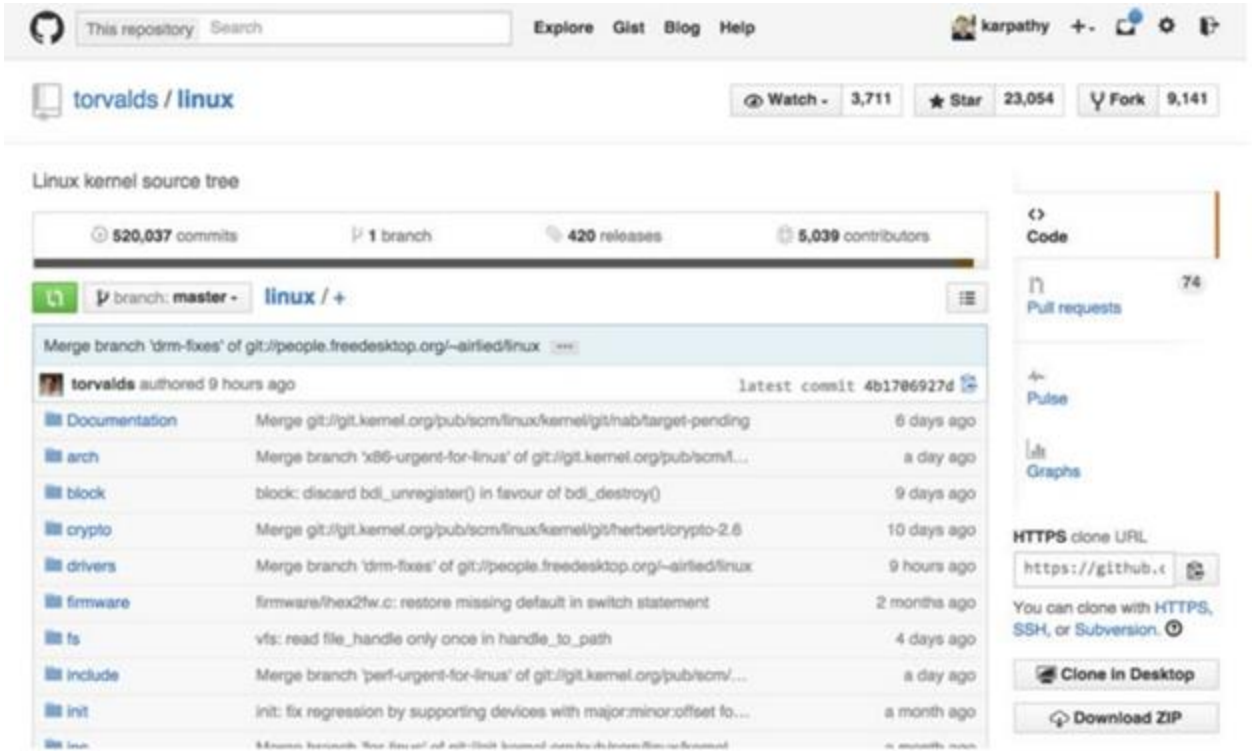
$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\mathcal{I}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_i}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\psi}^{\mathcal{F}})$$

is an isomorphism of covering of $\mathcal{O}_{X_\psi}$. 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.

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network



http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000ffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Generated C code

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

```
/*
 *  Copyright (c) 2006-2010, Intel Mobile Communications.  All rights reserved.
 *
 *   This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 *        This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software Foundation,
 *  Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

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

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

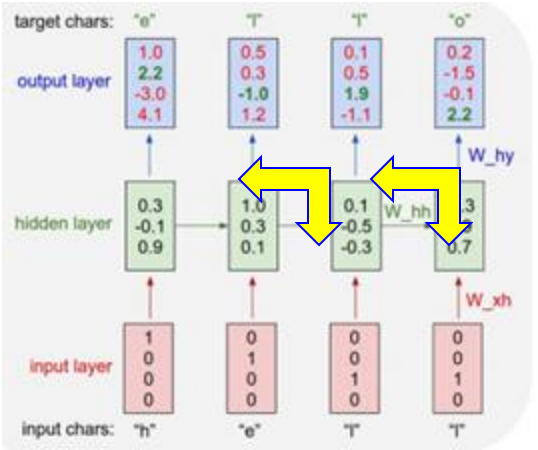
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

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
```

Backprop



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

$$y_t = W_{hy}h_t$$

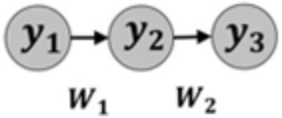
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Training RNNs

- Recap: Training CNN

$$y_i = g\left(\sum_j W_{ij}x_j + b_i\right)$$

$$\frac{\partial C}{\partial W} = \frac{\partial C}{\partial g} \cdot \frac{\partial g}{\partial a} \cdot \frac{\partial a}{\partial W}$$



$$\boldsymbol{y}_k = g(W\boldsymbol{y}_{k-1} + \boldsymbol{b})$$

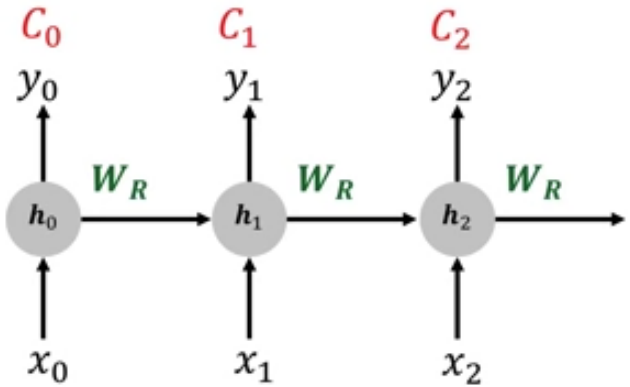
- Single, final cost C
- Non-shared weights (W_1, W_2)

https://www.youtube.com/watch?v=Ukgii7Yd_cU

Training RNNs

$$\boldsymbol{h}^{(t)} = g_h(W_{\mathrm{I}}\boldsymbol{x}^{(t)} + W_{\mathrm{R}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$

$$\boldsymbol{y}^{(t)} = g_y(W_y\boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$



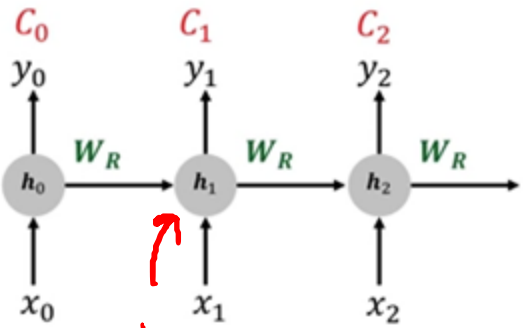
- (Potentially) multiple, intermediate costs C_0,C_1,C_2
- Shared weights $\mathbf{W_R}$

https://www.youtube.com/watch?v=Ukgii7Yd_cU

Training RNNs - Backpropagation through time

$$\boldsymbol{h}^{(t)} = g_h(W_I \boldsymbol{x}^{(t)} + W_R \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$

$$\boldsymbol{y}^{(t)} = g_y(W_y \boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$



$$\frac{\partial C}{\partial W_R} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial W_R}$$

$$\frac{\partial C_t}{\partial W_R} = \sum_{k=1}^{t} \frac{\partial C_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_R}$$

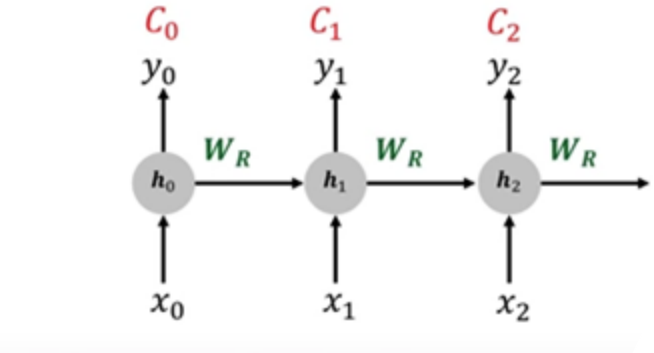
$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W_R^T \operatorname{diag}(g_h'(W_I x^{(i)} + W_R h^{(i-1)} + b_h))$$

- (Potentially) multiple, intermediate costs C_0, C_1, C_2
- Shared weights $\mathbf{W_R}$

https://www.youtube.com/watch?v=Ukgii7Yd_cU

Training RNNs - BPTT, Vanishing gradients

$$\boldsymbol{h}^{(t)} = g_h(W_I \boldsymbol{x}^{(t)} + W_R \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$
$$\boldsymbol{y}^{(t)} = g_y(W_y \boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$



$$\frac{\partial C}{\partial W_R} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial W_R}$$

$$\frac{\partial C_t}{\partial W_R} = \sum_{k=1}^{t} \frac{\partial C_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_R}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W_R^T \mathrm{diag}(g_h'(W_I x^{(i)} + W_R h^{(i-1)} + b_h))$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leqslant \left\| W_R^T \right\| \left\| \mathrm{diag}(g_h'(W_I x^{(i)} + W_R h^{(i-1)} + b_h)) \right\| \leqslant \gamma_{W_R} \gamma_{g_h}$$

Vanishing gradients !

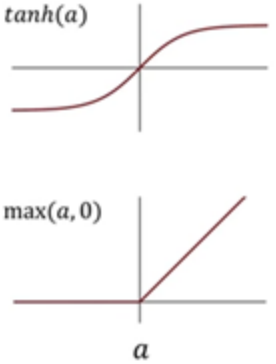
(Pascanu, et al., On the difficulty of training Recurrent Neural Networks.)

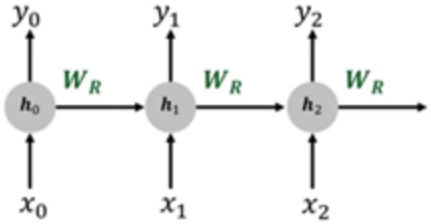
https://www.youtube.com/watch?v=Ukgii7Yd_cU

Training RNNs

Vanishing/Exploding gradients

Choice of a(.) is crucial too !





$$\frac{\partial C_{100}}{\partial W_R} = \frac{\partial C_{100}}{\partial y_{100}} \dots W_R \frac{\partial g_{100}}{\partial a_{100}} \dots W_R \frac{\partial g_{99}}{\partial a_{99}} \dots$$

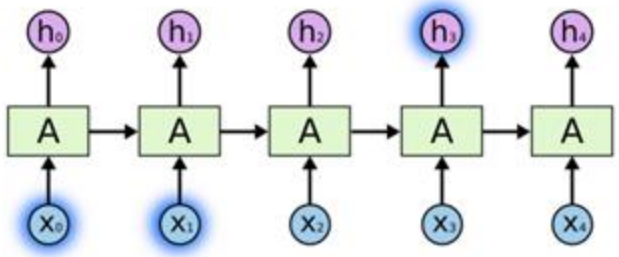
$$\frac{\partial C_T}{\partial W_R} \propto |W_R|^T \left|\frac{\partial g}{\partial a}\right|^T$$

- $|W_R| << 1 \rightarrow$ Vanishing gradients
- $|W_R| >> 1 \rightarrow$ Vanishing gradients
- $|W_R|$ **'close to 1'** is ideal

https://www.youtube.com/watch?v=Ukgii7Yd_cU

Fundamental issues with RNNs

- RNNs connect previous information to the present task
  - Previous video frames may help understand present frame
- Sometimes we need only look at recent previous information to predict
  - To predict the last word of "The clouds are in the *sky*" we don't need any further context. It is obvious that the word is "sky"

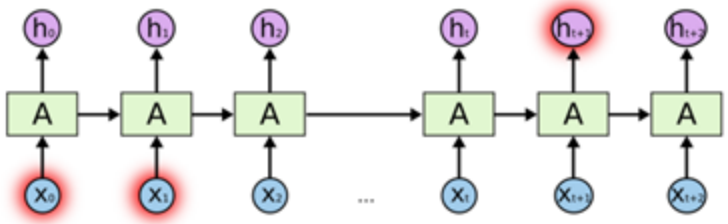


http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

Fundamental issues with RNNs

Problem of Long-term dependency

- There are cases where we need more context
- To predict the last word in the sentence "I grew up in France⋯.I speak fluent *French*"
- Using only recent information suggests that the last word is the name of a language. But more distant past indicates that it is French
- It is possible that the gap between the relevant information and where it is needed is very large



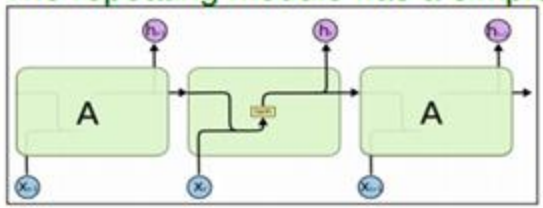
http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

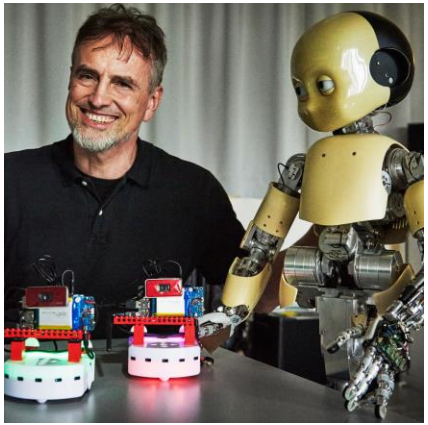

I'LL EMAIL THIS TO MYSELF SO I CAN REMEMBER IT
HEY LOOK, A NEW EMAIL!


Not only is my short-term memory horrible, but so is my short-term memory.
fb.com/MinionQuote
DespicableMeMinions.org

Long Short Term Memory (LSTM)

- Explicitly designed to avoid the long-term dependency problem
- RNNs have the form of a repeating chain structure
  - The repeating module has a simple structure such as tanh



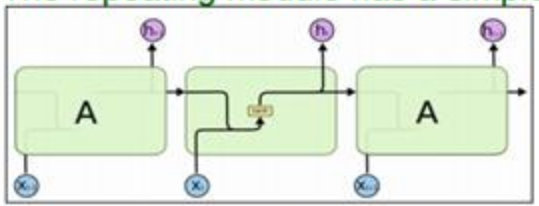


Proposed by Juergen Schmidhuber (1997 !)

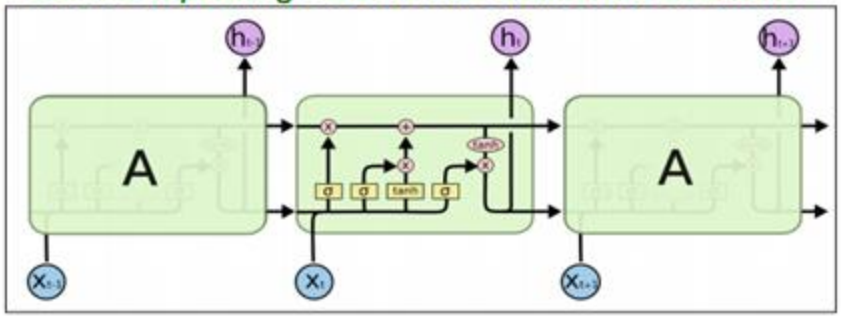
http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

Long Short Term Memory (LSTM)

- Explicitly designed to avoid the long-term dependency problem
- RNNs have the form of a repeating chain structure
  - The repeating module has a simple structure such as tanh



- LSTMs also have a chain structure
  - but the repeating module has a different structure

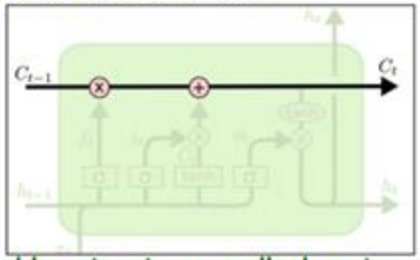


http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

LSTM

Core idea behind LSTM

- The key to LSTM is the cell state, C_t, the horizontal line running through the top of the diagram
- Like a conveyor belt
  - Runs through entire chain with minor interactions
  - LSTM does have the ability to remove/add information to cell state regulated by structures called gates

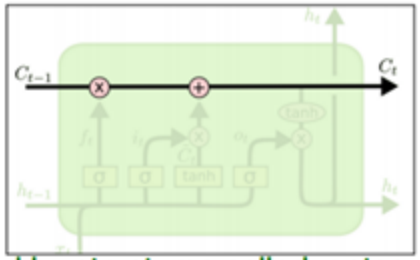


http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

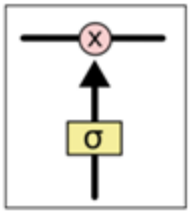
LSTM

Core idea behind LSTM

- The key to LSTM is the cell state, C_t, the horizontal line running through the top of the diagram



- Like a conveyor belt
  - Runs through entire chain with minor interactions
  - LSTM does have the ability to remove/add information to cell state regulated by structures called gates
- Gates are an optional way to let information through
- Consist of a sigmoid and a multiplication operation
- Sigmoid outputs a value between 0 and 1
  - 0 means let nothing through
  - 1 means let eveything through

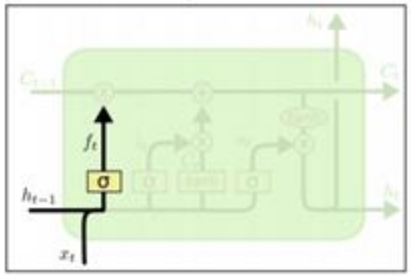


- LSTM has three of these gates, to protect and control cell state

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

LSTM - Components

- Example of language model: predict next word based on previous ones
  - Cell state may include the gender of the present subject
- First step: information to throw away from cell state



Called *forget gate layer*

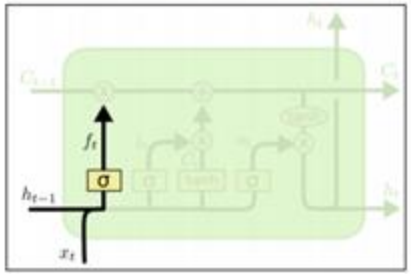
It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each member of C_{t-1} for whether to forget

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

LSTM - Components

- Example of language model: predict next word based on previous ones
  - Cell state may include the gender of the present subject
- First step: information to throw away from cell state



Called *forget gate layer*

It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each member of C_{t-1} for whether to forget

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

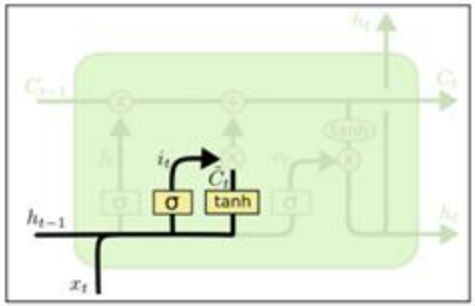
In language model

consider trying to predict the next word based on all previous ones
The cell state may include the gender of the present subject
so that the proper pronouns can be used
When we see a new subject we want to forget old subject

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

LSTM - Components

- Next step is to decide as to what new information we're going to store in the cell state



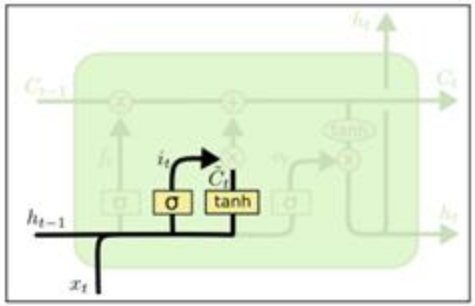
This has two parts:
first a sigmoid layer called *Input gate layer*: decides which values we will update
Next a tanh layer creates a vector of new candidate values $\tilde{C}_t$ that could be added to the state.
In the third step we will combine these two to create an update to the state

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

LSTM - Components

- Next step is to decide as to what new information we're going to store in the cell state



This has two parts:
first a sigmoid layer called *Input gate layer*: decides which values we will update
Next a tanh layer creates a vector of new candidate values $\tilde{C}_t$ that could be added to the state.
In the third step we will combine these two to create an update to the state

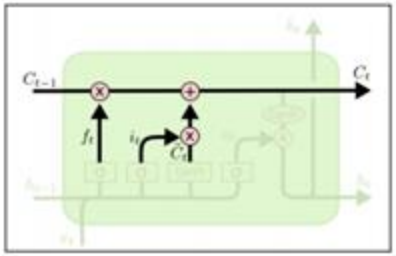
In the Language model, we'd want to add the gender of the new subject to the cell state, to replace the old One we are forgetting

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

LSTM - Components

- It's now time to update old cell state C_{t-1} into new cell state C_t
  - The previous step decided what we need to do
  - We just need to do it



We multiply the old state by f_t, forgetting the things we decided to forget earlier.
Then we add $i_t * \tilde{C}_t$
This is the new candidate values, scaled by how much we decided to update each state value

In the Language model,
this is where we'd actually drop the information about the old subject's gender and add the new information,
as we decided in previous steps

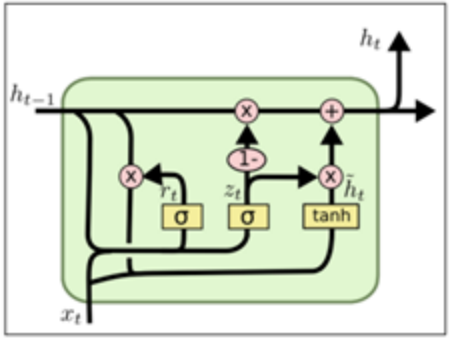
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

GRU: a LSTM variant

Gated Recurrent Unit (GRU)

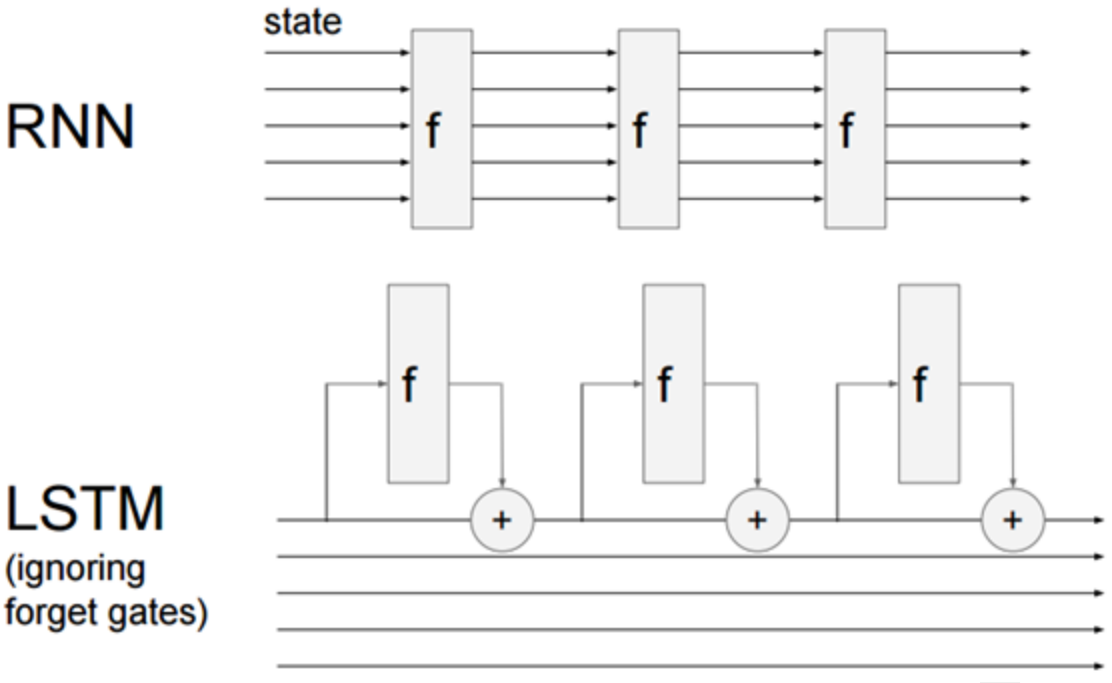
- A dramatic variant of LSTM
  - It combines the forget and input gates into a single update gate
  - It also merges the cell state and hidden state, and makes some other changes
  - The resulting model is simpler than LSTM models
  - Has become increasingly popular



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

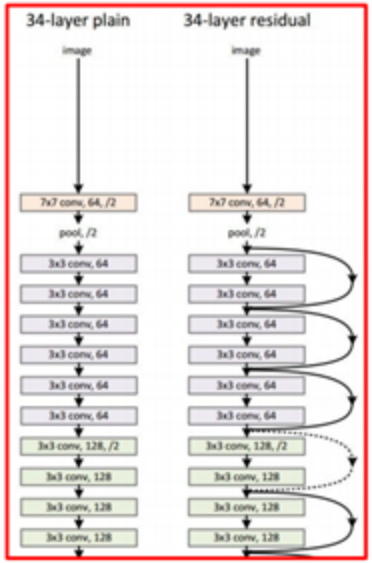
http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

Long Short Term Memory (LSTM)



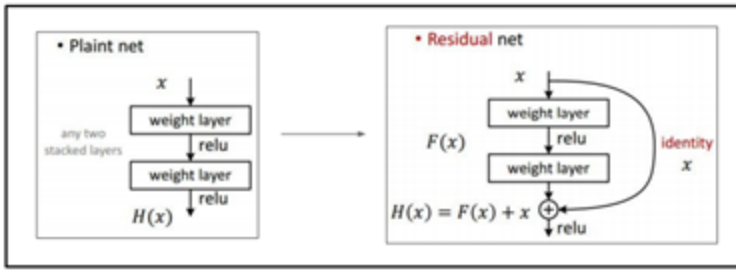
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Long Short Term Memory (LSTM)



Recall:
"PlainNets" vs. ResNets

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



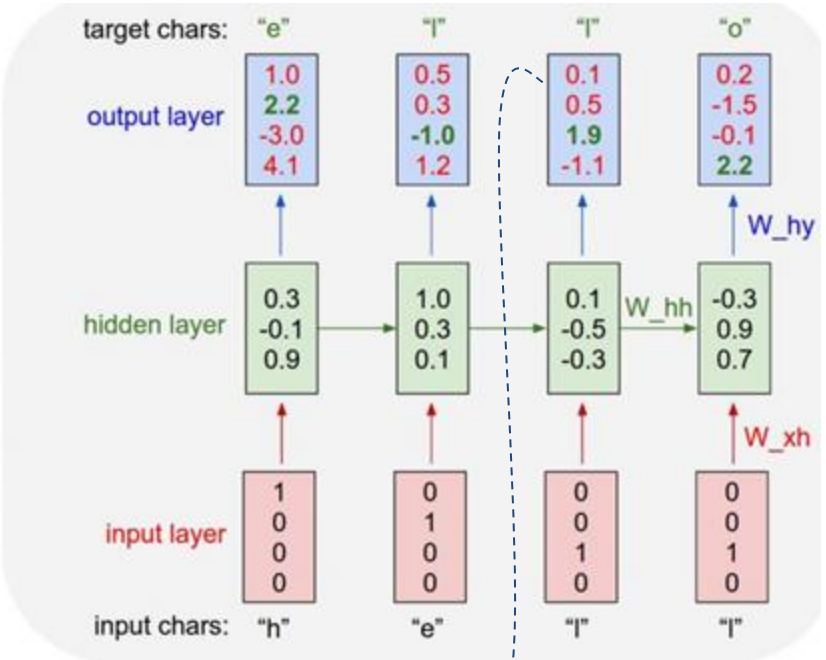
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



$p(\text{next letter} = \text{“l”} \,|\, \text{previous letters} = \{\text{“h”}, \text{“e”}, \text{“l”}\})$

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

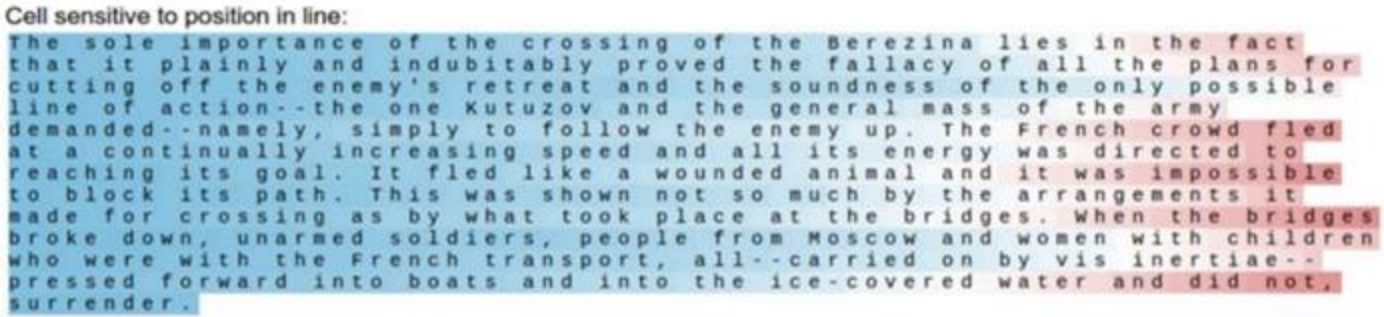
Searching for interpretable cells

quote detection cell

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

Searching for interpretable cells

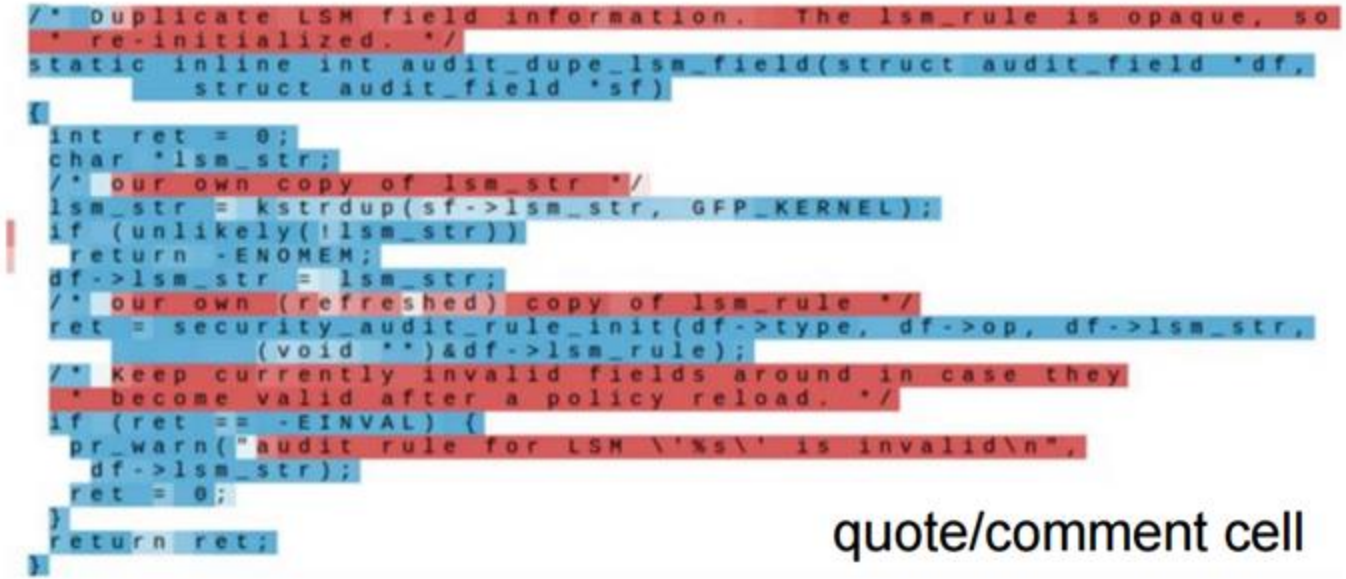


line length tracking cell

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

Searching for interpretable cells



http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Applications of RNNs in CV

- Image/Video Captioning
- Visual Question Answering
- Action / Activity Recognition from video
- See https://github.com/kjw0612/awesome-rnn for a larger list ...

Applications of RNNs in CV

- **Image/Video Captioning**
- Visual Question Answering
- Action / Activity Recognition from video

Image Captioning

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.

(Slides by Marc Bolaños): Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." CVPR 2015

Recap: Language generation

Training this on a lot of sentences would give us a language model. A way to predict

P(next word | previous words)

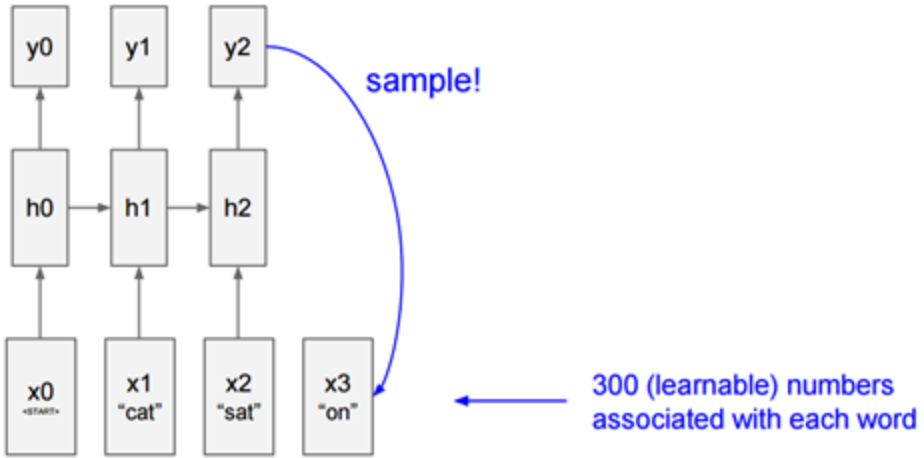


Image Captioning

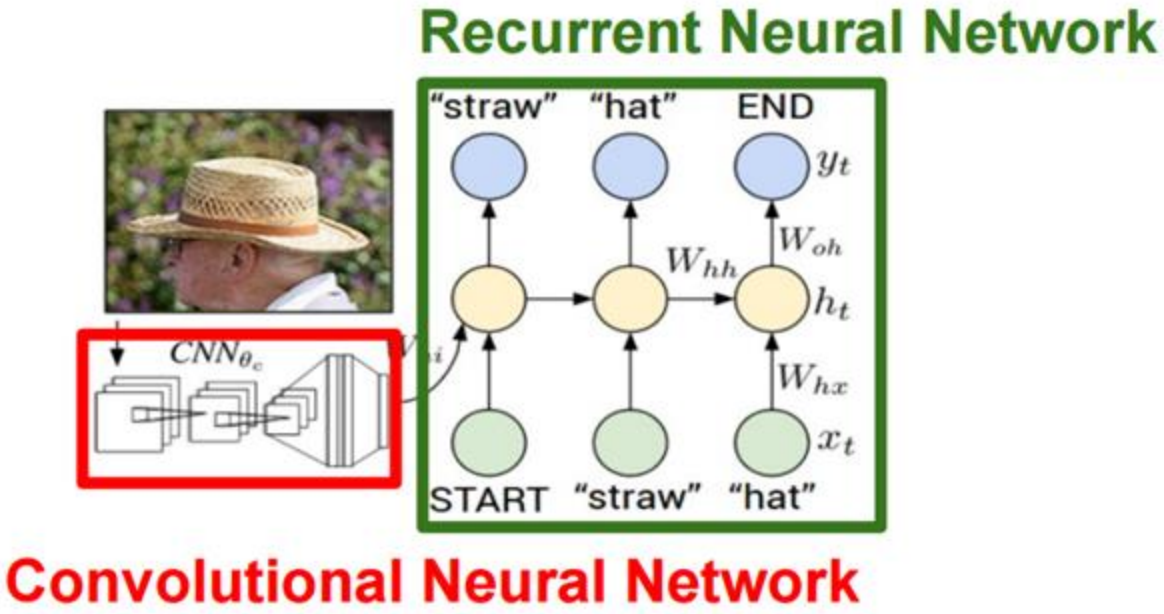


Image Captioning

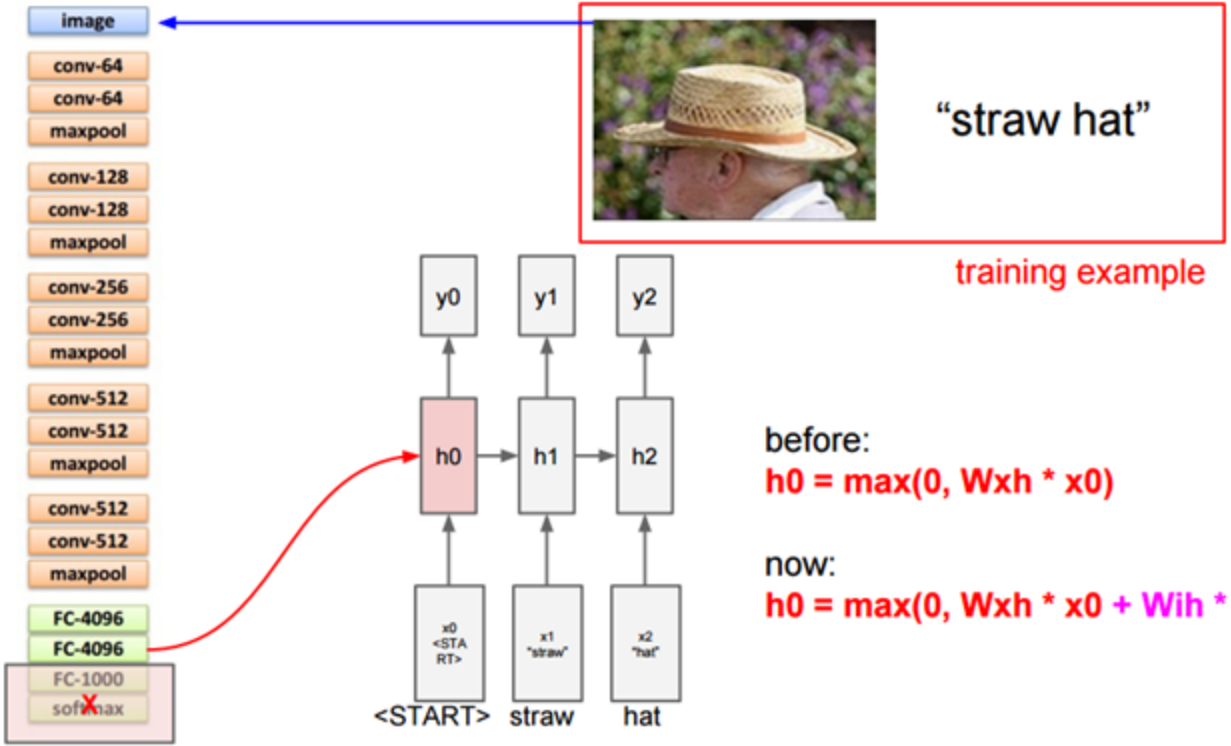


Image Captioning

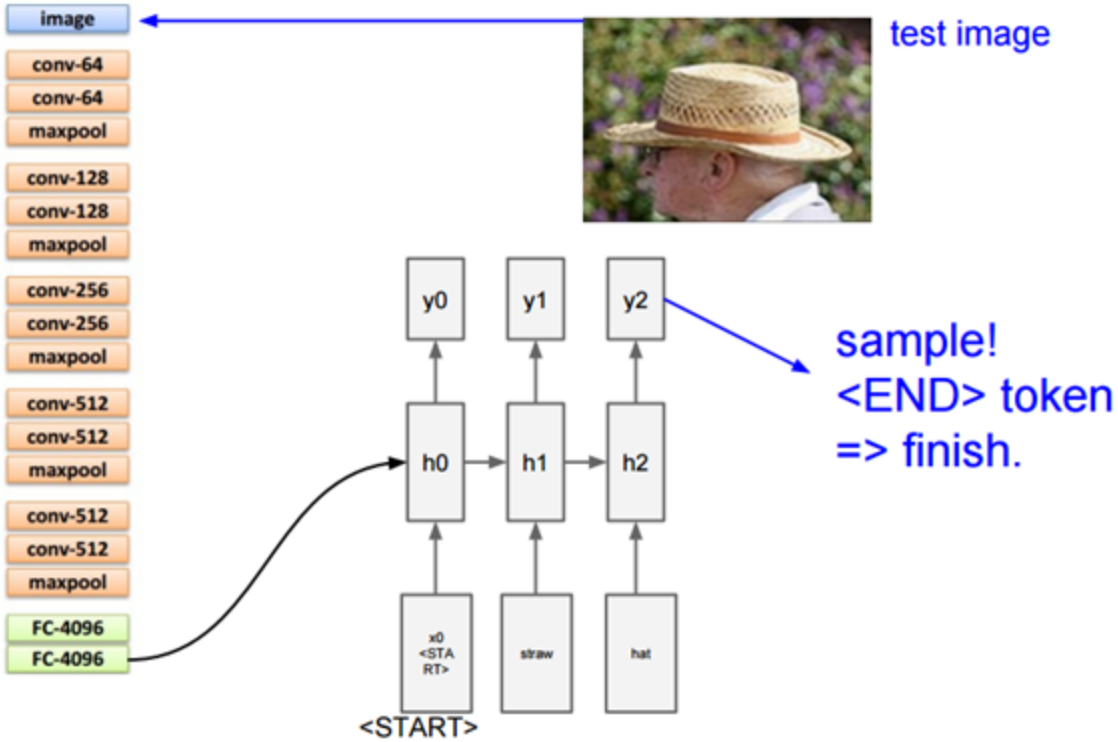
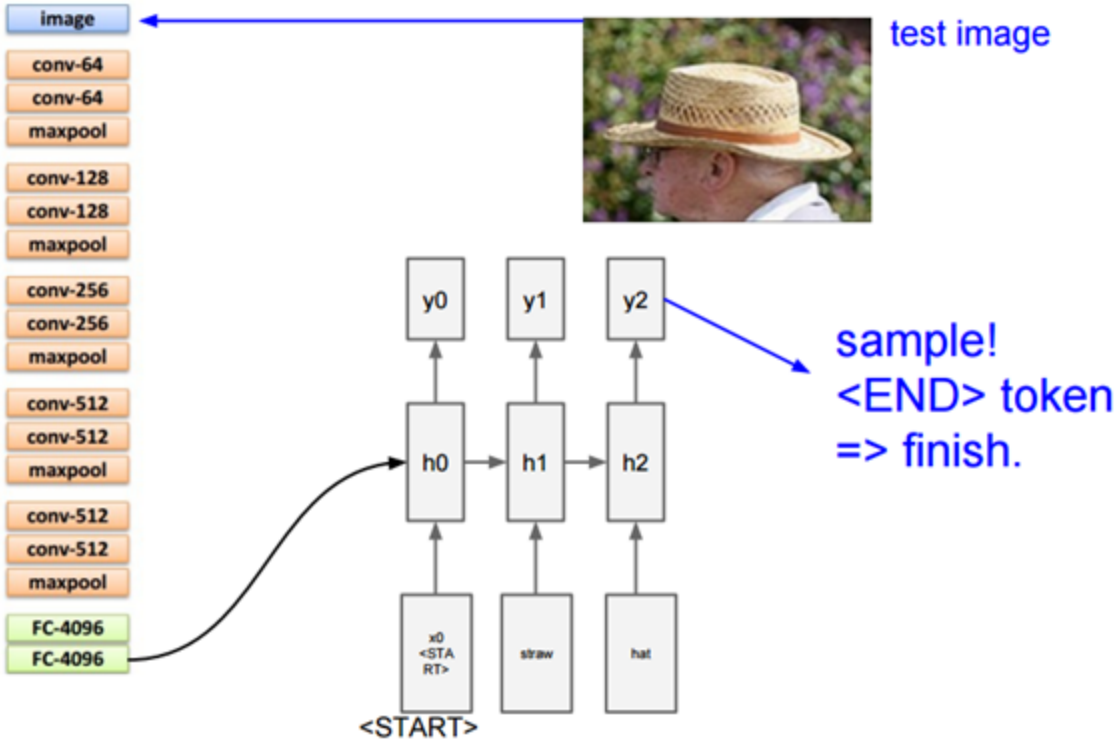
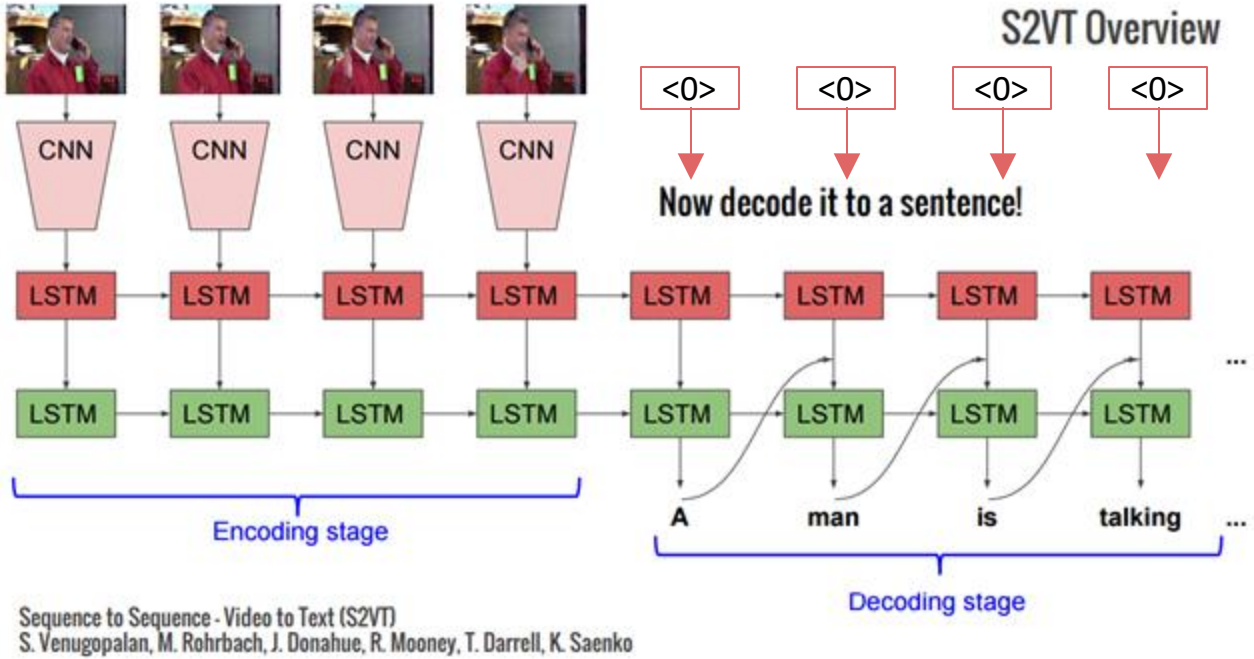


Image Captioning



Video Captioning



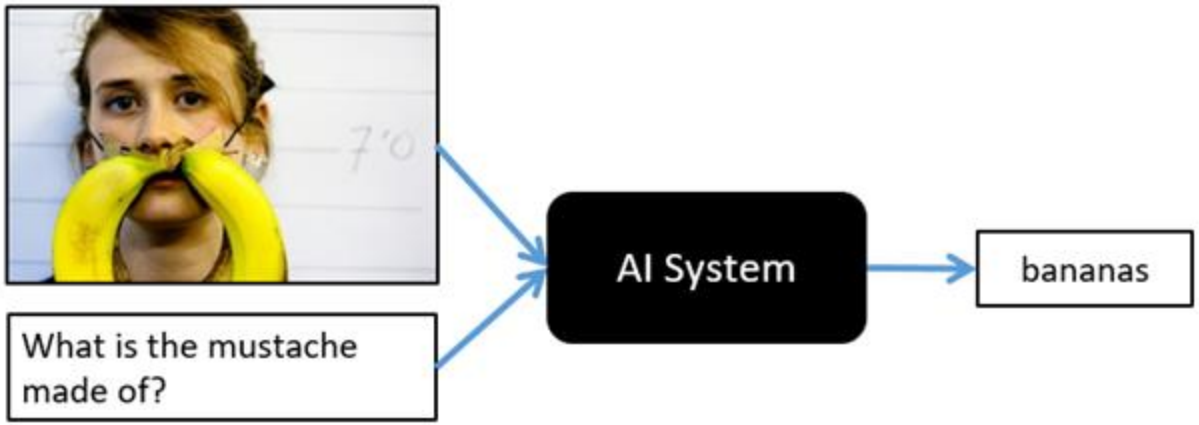
Sequence to Sequence - Video to Text (S2VT)
S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, K. Saenko

Video Captioning

Applications of RNNs in CV

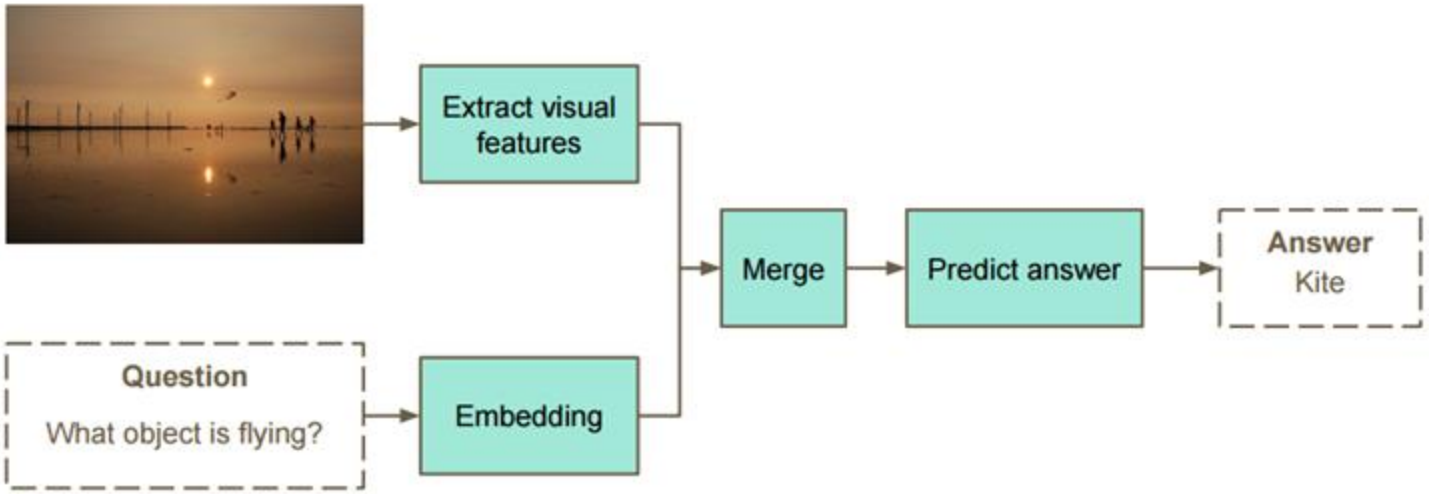
- Image/Video Captioning
- **Visual Question Answering**
- Action / Activity Recognition from video

Visual Question Answering



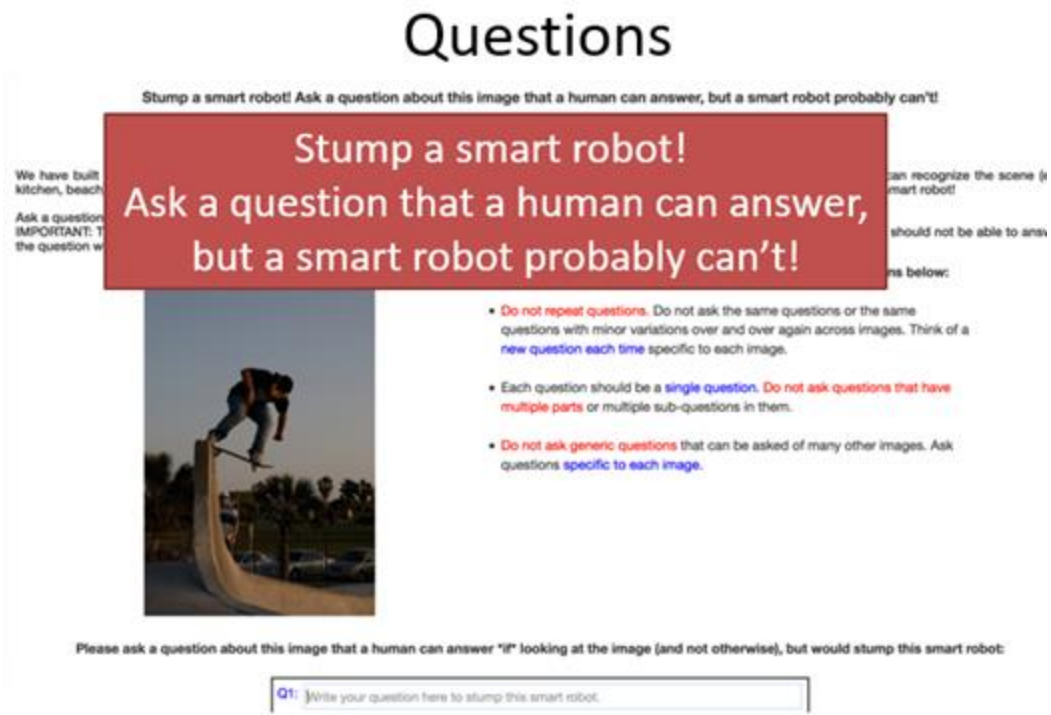
Slide credit: Devi Parikh

Visual Question Answering



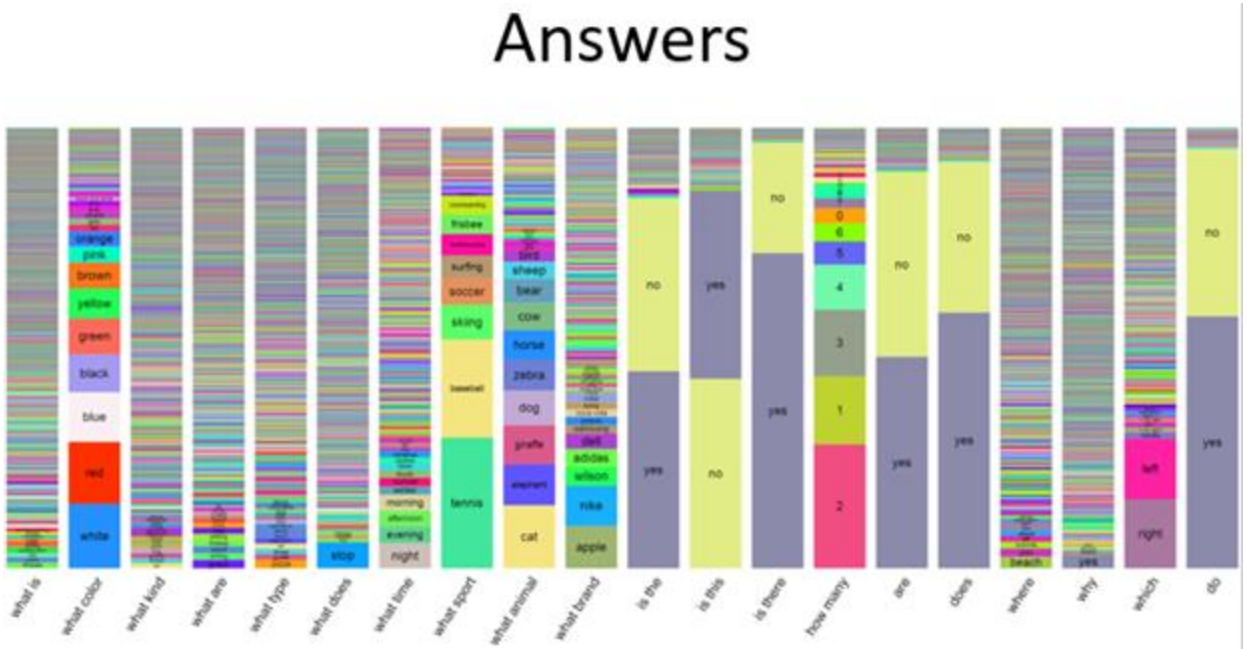
Slide credit: Issey Masuda

Visual Question Answering



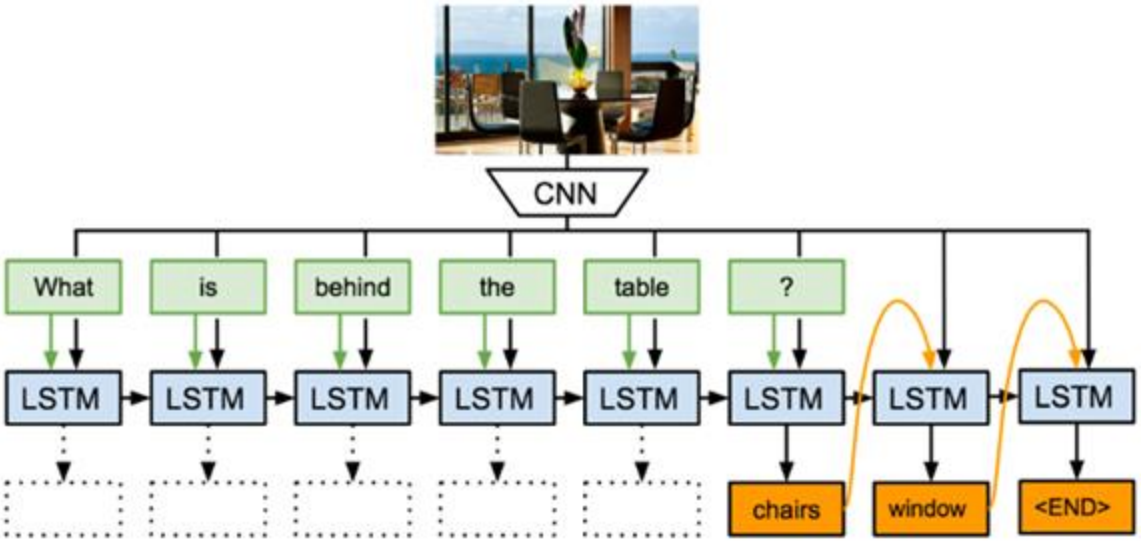
Slide credit: Devi Parikh

Visual Question Answering



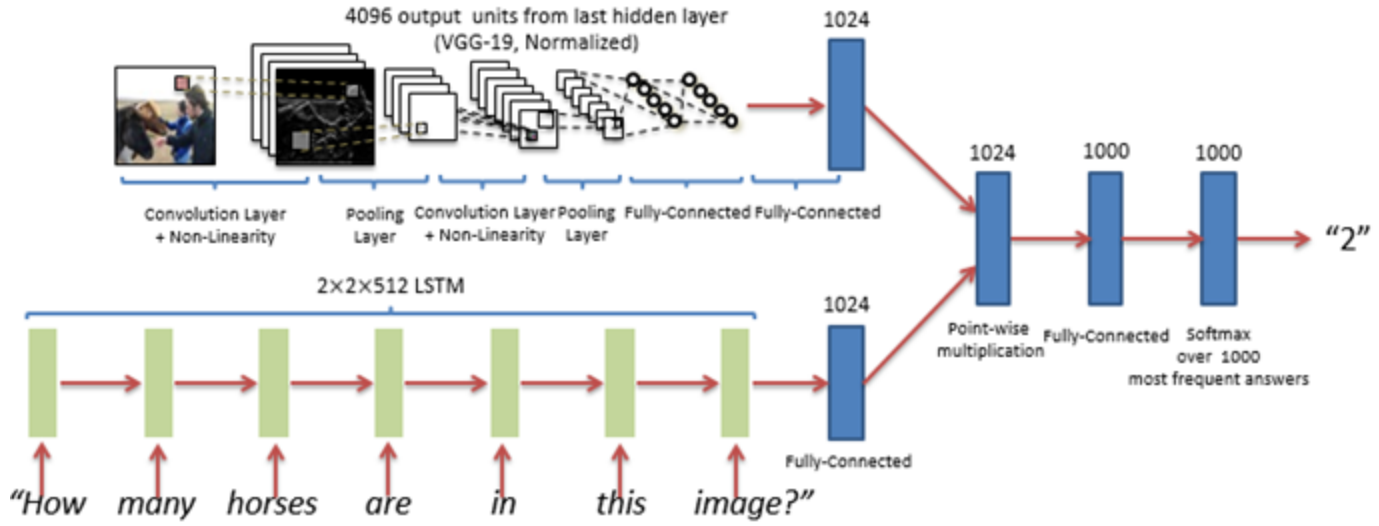
Visual Question Answering

[Malinowski 2015]



Visual Question Answering

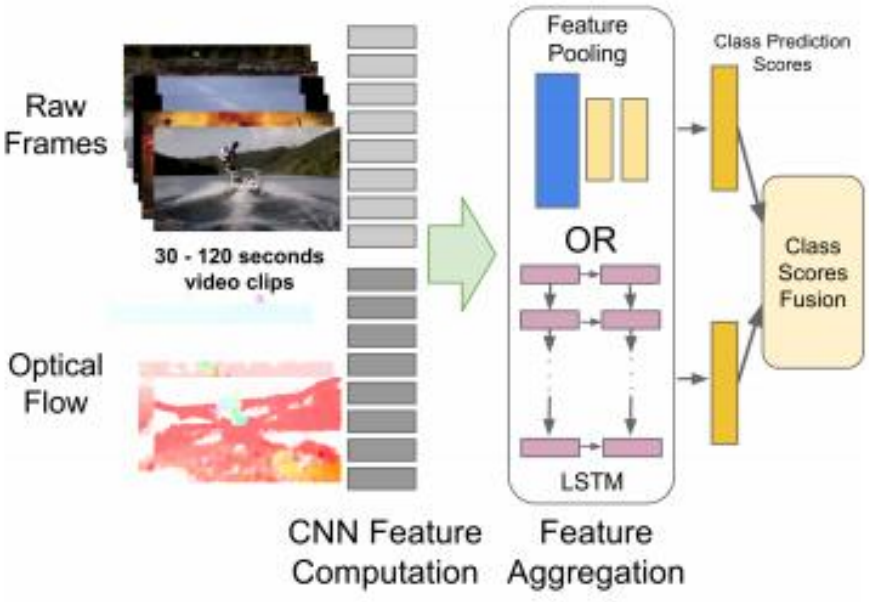
[Lu 2015]



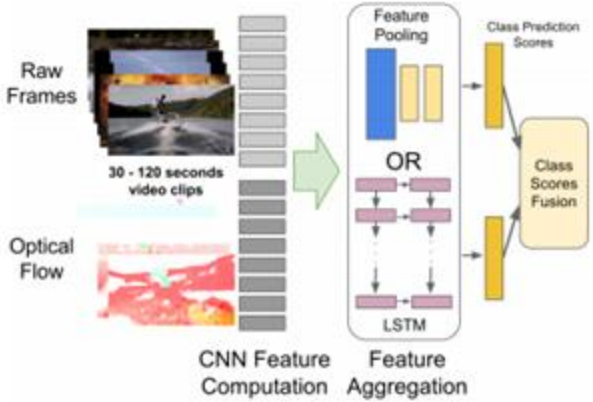
Applications of RNNs in CV

- Image Captioning
- Visual Question Answering
- **Action / Activity Recognition from video**

Action/Activity Recognition



Action/Activity Recognition



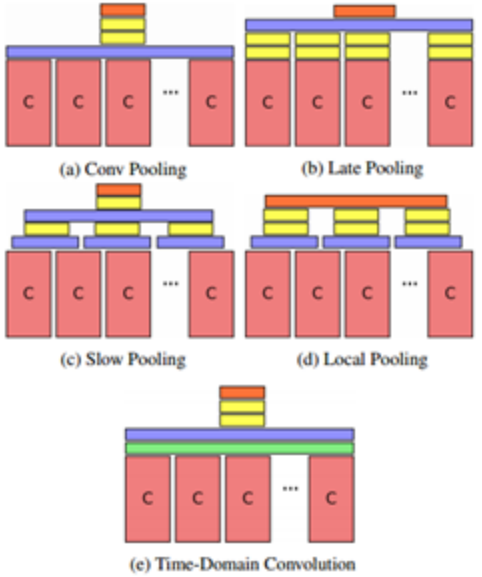
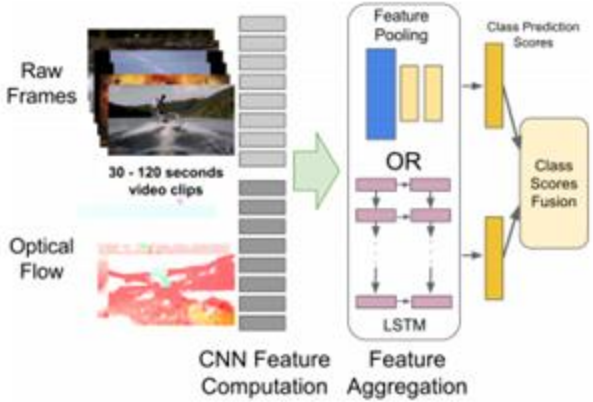
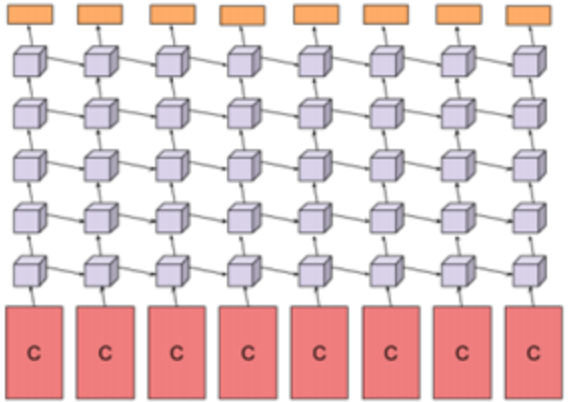


Figure 2: Different Feature-Pooling Architectures: The stacked convolutional layers are denoted by "C". Blue green, yellow and orange rectangles represent max-pooling time-domain convolutional, fully-connected and softmax layers respectively.

Action/Activity Recognition





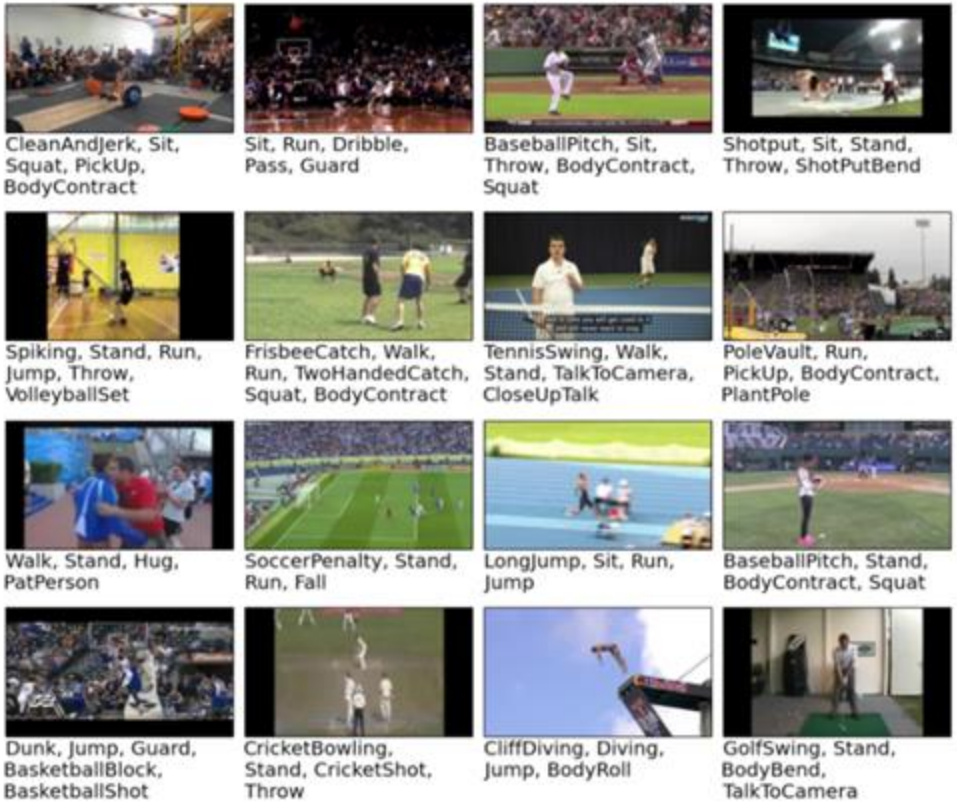
Action/Activity Recognition

Method	Clip Hit@1	Hit@1	Hit@5
Conv Pooling	**68.7**	**71.1**	**89.3**
Late Pooling	65.1	67.5	87.2
Slow Pooling	67.1	69.7	88.4
Local Pooling	68.1	70.4	88.9
Time-Domain Convolution	64.2	67.2	87.2

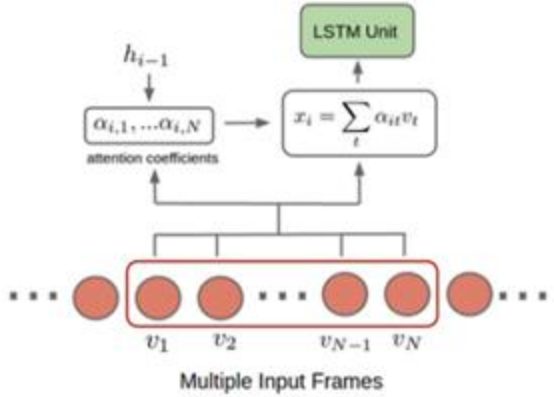
Method	Hit@1	Hit@5
AlexNet single frame	63.6	84.7
GoogLeNet single frame	**64.9**	**86.6**
LSTM + AlexNet (fc)	62.7	83.6
LSTM + GoogLeNet (fc)	**67.5**	**87.1**
Conv pooling + AlexNet	70.4	89.0
Conv pooling + GoogLeNet	**71.7**	**90.4**

Take-home : Consider pooling schemes as an alternative to RNNs

Action/Activity Recognition



Action/Activity Recognition



(a) Connections to multiple inputs.

$$\alpha_{it} \propto \exp(w_{ae}^T [\tanh(W_{ha} h_{i-1}) \odot \tanh(W_{va} v_t)])$$

Look for certain features

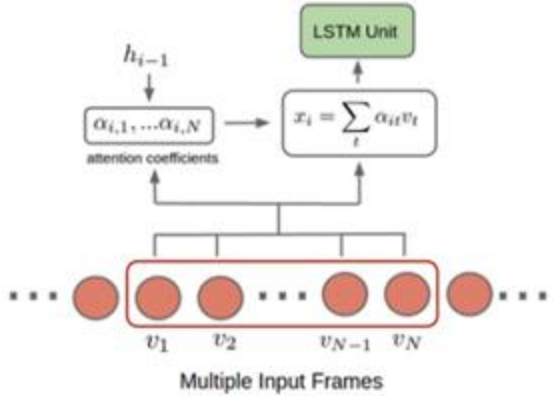
Advertise the presence/absence of features

$$y_t = \sum_i \beta_{it} p_{it}$$

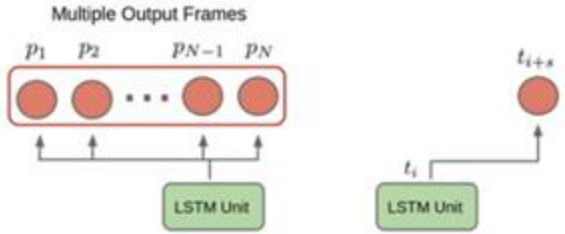
$\frac{1}{N}$

Every Moment Counts: Dense Detailed Labeling of Actions in Complex Videos, Yeung et al., 2015

Action/Activity Recognition



(a) Connections to multiple inputs.



(b) Multiple outputs. (c) Variant: output offset.

$$\alpha_{it} \propto \exp(w_{ae}^T [\tanh(W_{ha} h_{i-1}) \odot \tanh(W_{va} v_t)])$$

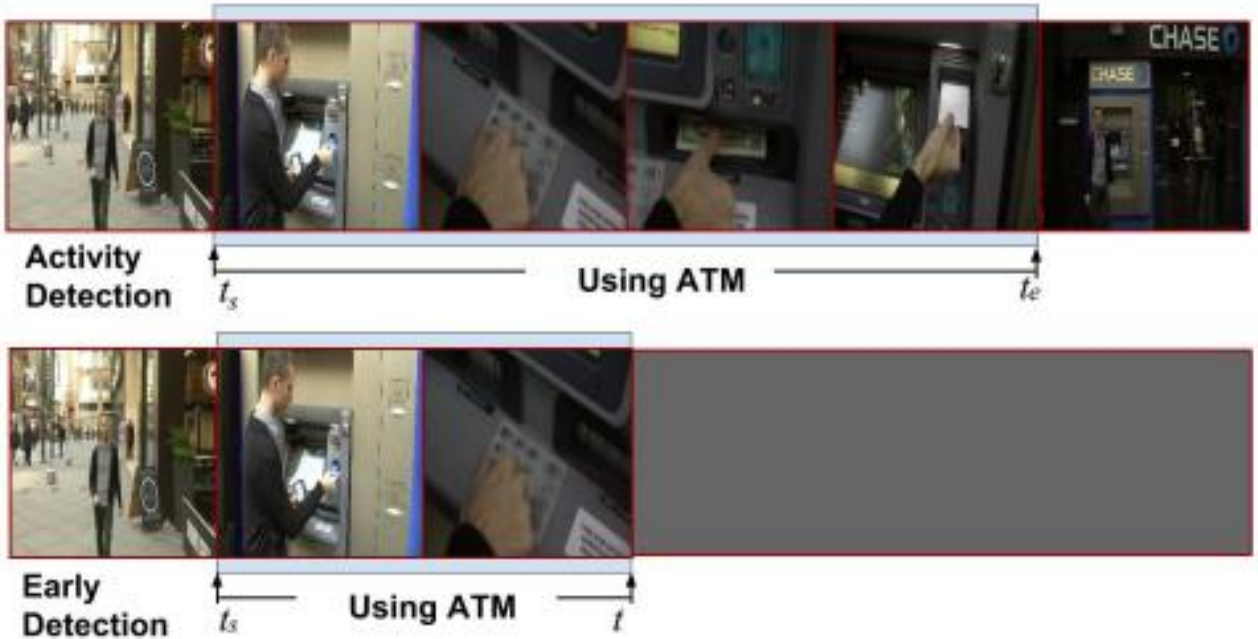
Look for certain features

Advertise the presence/absence of features

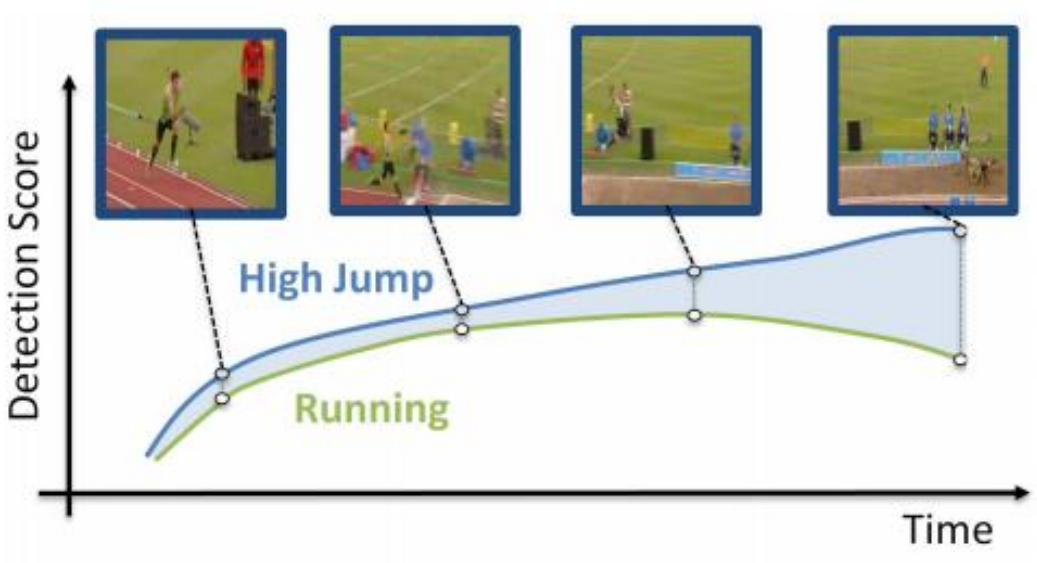
$$y_t = \sum_i \beta_{it} p_{it}$$

$\beta_{it} = \frac{1}{N}$

Action/Activity Recognition



Action/Activity Recognition



- Score for correct activity should increase over time
- Score margin between correct and incorrect should increase over time

Action/Activity Recognition

- Score for correct activity should increase over time (s-loss)
- Score margin between correct and incorrect should increase over time (m-loss)

In other words, if there is no activity transition at time t, *i.e.*, $y_t = y_{t-1}$, then we want the current detection score to be no less than any previous detection score for the same activity, computing the ranking loss as:

$$\mathcal{L}_s^t = \max(0,\ p_t^{*y_t} - p_t^{y_t}). \tag{7}$$

On the other hand, if an activity transition happens at time t, *i.e.*, $y_t \neq y_{t-1}$, we want the detection score of the previous activity to drop to zero at t and compute the ranking loss as:

$$\mathcal{L}_s^t = p_t^{y_{t-1}}. \tag{8}$$

$$m_t^y = p_t^y - \max\{p_t^{y'} \mid \forall y' \in \mathcal{Y},\ y' \neq y\}$$

In other words, when there is no activity transition at t, we want the current discriminative margin to be no less than any previous margin in the same activity, computing the ranking loss as:

$$\mathcal{L}_m^t = \max(0,\ m_t^{*y_t} - m_t^{y_t}). \tag{14}$$

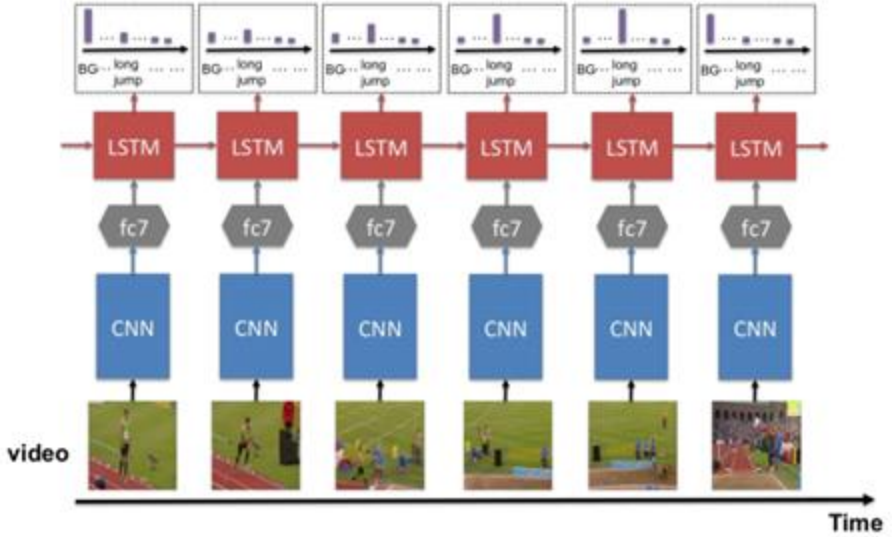
If an activity transition happens at time t, we want the discriminative margin of the previous activity to drop and compute the ranking loss as:

$$\mathcal{L}_m^t = m_t^{y_{t-1}}. \tag{15}$$

$$\mathcal{L}^t = \mathcal{L}_c^t + \lambda_r \mathcal{L}_r^t,$$

$$\mathcal{L}_c^t = -\log p_t^{y_t}$$

Action/Activity Recognition



$$\mathcal{L}^t = \mathcal{L}_c^t + \lambda_r \mathcal{L}_r^t,$$

$$\mathcal{L}_c^t = -\log p_t^{y_t}$$

RNN and CNN

- RNNs = A family of neural networks for processing sequential data
- CNN and RNN are both specialized architectures
  - CNN : processes a grid of values such as an image
  - RNN : processes sequence of values x(1) ,..,x(τ)
  - CNN :

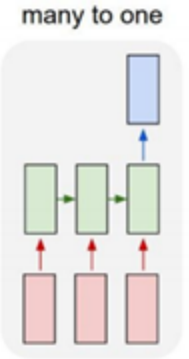
Sharing parameters is key to RNNs

- With separate parameters for each value of time we could not
  - generalize to sequence lengths not seen during training
  - share statistical strength across different sequence lengths and across different positions in time
- (c.f shared weights of conv filter in CNN)

Sharing parameters is key to RNNs

- Compare RNN / CNN weights
  - CNN : Best param setting as determined by independent (image) instances
  - RNN : Best param setting as determined by independent sequences
  - RNN weights have to make sense for current ‘and past’ sequence members

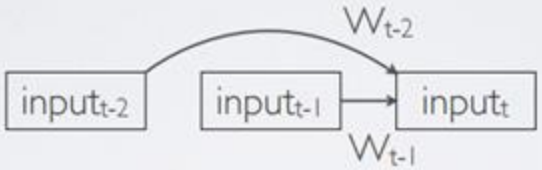
Sharing parameters is key to RNNs



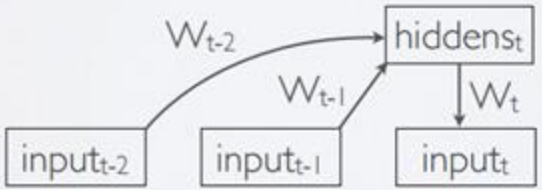
- Sharing is important when a particular info can occur at multiple positions in the sequence
  - Given "I went to Paris in 2012" and "In 2012, I went to Paris", extract year.
    - Should extract 2012 whether in position 6 or 2 (many → one)
    - A feed-forward net that processes sentences of fixed length would have to learn all of the rules of language separately at each position
    - RNN shares the same weights across several time steps, so can handle this in a scalable manner.

RNNs = Just fancy HMMs ?

- Memoryless models for sequences:
  - Autoregressive models: Predict the next input in a sequence from a fixed number of previous inputs using "delay taps".



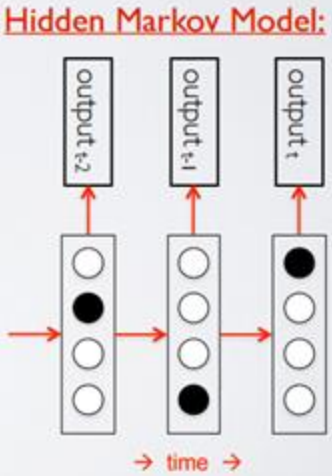
  - Feed-forward neural networks: Generalize autoregressive models by using non-linear hidden layers.



http://www.iro.umontreal.ca/~bengioy/ift6266/H16/RNN-part1.pdf

RNNs = Just fancy HMMs ?

- If the model could possess a dynamic hidden state:
  - Can store long term information.
- Dynamic Bayes networks (DBNs) can possess internal dynamic state
  - Example 1: Linear Dynamic System with Gaussian noise model (Kalman Filter)
  - Example 2: Discrete state, arbirary observation type (Hidden Markov Model)
    - State is not observed, must be inferred.
    - Represent probability across **N** states with **N** numbers.
    - Efficient algorithms exist for HMM inference.



http://www.iro.umontreal.ca/~bengioy/ift6266/H16/RNN-part1.pdf

RNNs = Just fancy HMMs ?

LIMITATIONS OF HMMS

- Consider modelling sentence production with an HMM.
- Generation procedure:
  - At each time step t, sample state $_t$ given state $_{t-1}$.
  - Everything important about the past outputs (output 0, . . . , output t-1) is must be summarized in this choice of state
  - So with N states, it can only remember log(N) bits of the past.

http://www.iro.umontreal.ca/~bengioy/ift6266/H16/RNN-part1.pdf

RNNs = Just fancy HMMs ?

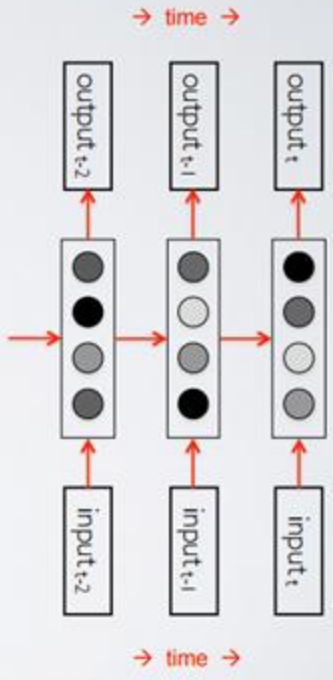
LIMITATIONS OF HMMS

- Consider modelling sentence production with an HMM.
- Generation procedure:
  - At each time step t, sample state $_t$ given state $_{t-1}$.
  - Everything important about the past outputs (output 0, . . . , output t-1) is must be summarized in this choice of state
  - So with N states, it can only remember log(N) bits of the past.
- Consider trying to generate the second half of the sentence with the first have already generated.
  - Syntax needs to fit (number and tense agreement).
  - Semantics need to fit.
- These aspects combined could be say 100 bits that need to be conveyed from the first half to the second. 2^{100} is big!

http://www.iro.umontreal.ca/~bengioy/ift6266/H16/RNN-part1.pdf

RNNs = Just fancy HMMs ?

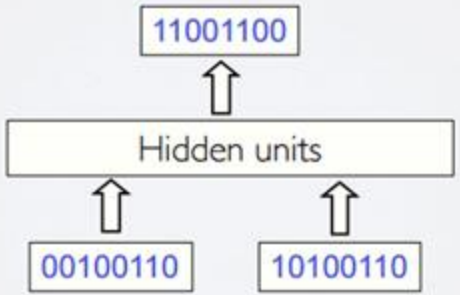
- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state: can efficiently store a lot of information about the past.
    - Note: real valued activations not 1-of-N
  - Non-linear dynamics: can update their hidden state in complicated ways.



http://www.iro.umontreal.ca/~bengioy/ift6266/H16/RNN-part1.pdf

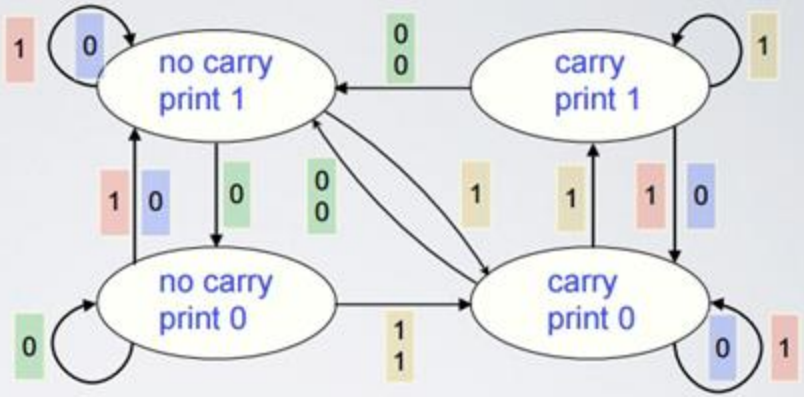
EXAMPLE: BINARY ADDITION

- Can we train a **feed-forward net** to do binary addition?
- **Problems**:
  - We must decide in advance the maximum number of digits in each number.
  - The processing applied to the beginning of a long number does not generalize to the end of the long number because it uses different weights.
- Feed-forward nets do not generalize well on binary addition.



http://www.iro.umontreal.ca/~bengioy/ift6266/H16/RNN-part1.pdf

BINARY ADDITION AS A DYNAMIC PROCESS



- Moves from right to left over two input numbers.
- Finite state atomaton
  - Decides what transition to make by looking at the next column.
  - It prints after making the transition.

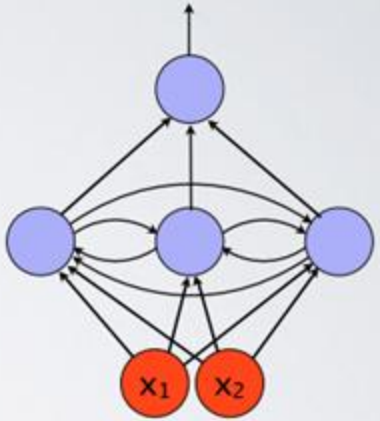
http://www.iro.umontreal.ca/~bengioy/ift6266/H16/RNN-part1.pdf

RNN FOR BINARY ADDITION

- Network has two input units and one output unit.
  - Each input corresponds to a digit from a distinct binary number.
- Desired output at each time step is the output for the column that was provided as input two time steps ago.
  - Takes one time step to update the hidden units based on the two input digits.
  - Takes another time step for the hidden units to cause the output.

RNN FOR BINARY ADDITION

- RNN solution has 3 hidden units have all possible interconnections in all directions:
  - Allows hidden activity pattern at one time step to vote for the hidden activity pattern at the next time step.
- Input units have feedforward connections that allow them to vote for the next hidden activity pattern.



http://www.iro.umontreal.ca/~bengioy/ift6266/H16/RNN-part1.pdf

Getting targets when modeling sequences

- When applying machine learning to sequences, we often want to turn an input sequence into an output sequence that lives in a different domain.
  - *E. g.* turn a sequence of sound pressures into a sequence of word identities.

https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

Getting targets when modeling sequences

- When applying machine learning to sequences, we often want to turn an input sequence into an output sequence that lives in a different domain.
  - *E. g.* turn a sequence of sound pressures into a sequence of word identities.

- When there is no separate target sequence, we can get a teaching signal by trying to predict the next term in the input sequence.
  - The target output sequence is the input sequence with an advance of 1 step.
  - This seems much more natural than trying to predict one pixel in an image from the other pixels, or one patch of an image from the rest of the image.
  - For temporal sequences there is a natural order for the predictions.

https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

Getting targets when modeling sequences

- When applying machine learning to sequences, we often want to turn an input sequence into an output sequence that lives in a different domain.
  - *E. g.* turn a sequence of sound pressures into a sequence of word identities.

- When there is no separate target sequence, we can get a teaching signal by trying to predict the next term in the input sequence.
  - The target output sequence is the input sequence with an advance of 1 step.
  - This seems much more natural than trying to predict one pixel in an image from the other pixels, or one patch of an image from the rest of the image.
  - For temporal sequences there is a natural order for the predictions.

- Predicting the next term in a sequence blurs the distinction between supervised and unsupervised learning.
  - It uses methods designed for supervised learning, but it doesn't require a separate teaching signal.

https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf