CS 559 Deep Learning

Recurrent Neural Networks

Gokberk Cinbis
Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
Recurrent Networks offer a lot of flexibility:

Vanilla Neural Networks
Recurrent Networks offer a lot of flexibility:

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

e.g. **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

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
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 $x$ 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”
min-char-rnn.py gist: 112 lines of Python

```python
# Import necessary packages
import numpy as np

# Define the main function
def sample(model, seed_ix, n):
    # Sample a sequence of integers from the model
    sample = [seed_ix]  # Start the sequence with the seed
    for t in range(n):
        x = np.zeros(vocab_size, 1)  # Initialize x as an array of zeros
        x[seed_ix] = 1  # Set the seed character as the current character
        preds = model.predict(x)  # Predict the next character
        # Sample the next character based on the predicted probabilities
        seed_ix = np.argmax(preds)  # Select the character with the highest probability
        sample.append(seed_ix)  # Add the selected character to the sequence
    return sample
```

(https://gist.github.com/karpathy/d4dee566867f8291f086)
```python

Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)

BSD License

```

# 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(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }
```
Initializations

# 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
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 '-----
%d
-----' % (txt,)

    # forward seq_length characters through the net and fetch gradient
    loss, dWxh, dWhh, dWhy,dbh, 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, dbh, dby],
                                   [mWxh, mWhh, mWhy, mbh, 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
n, p = 0, 0
mwkh, mwWh, mwWy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
mh, mby = np.zeros_like(wh), 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], 260)
        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
        print '---
% %
' % (txt, )

    # forward seq_length characters through the net and fetch gradient
    loss, dwkh, 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([wkh, Whh, Why, bh, by],
                                   [dwkh, dwhh, dwhy, dbh, dby],
                                   [mwkh, mwWh, mwWy, mh, 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
```
import pickle
print('Loading: 
')
with open('data/shakespeare.pickle', 'rb') as fin:
    train, valid, test, char_to_ix, ix_to_char = pickle.load(fin)

print('Loaded: 
')

n, p = 0, 0
mxh, mwh, mwhy = np.zeros_like(wxh), np.zeros_like(wwh), np.zeros_like(why)
mbh, mbw, 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
    x = p  # 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('----
'.join('
'.join(txt[i:i+20]) for i in range(0, len(txt), 20))
    # forward seq_length characters through the net and fetch gradient
    loss, dxh, dww, 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))
    # perform parameter update with Adagrad
    for param, dparam, mem in zip([wxh, wwh, why, bh, by],
                                   [dxh, dww, dwhy, dbh, dby],
                                   [mxh, mwh, mwhy, mbh, mbw]):
        mem += dparam**2
        param += -learning_rate * dparam / np.sqrt(mem + 1e-8)  # adagrad update
    p += seq_length  # move data pointer
    n += 1  # iteration counter
```
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 ('---

    # 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
min-char-rnn.py gist

Main loop (AdaGrad)

```python
n, p = 0, 0
mwkh, mwhh, mwhy = np.zeros_like(wkh), 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, dWkh, dWhh, dWhy, dbh, 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([Wkh, Whh, Why, bh, by],
                                    [dWkh, dWhh, dWhy, dbh, dbh, dby],
                                    [mwkh, mwhh, mwhy, mbh, 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
```
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 Hx1 array of initial hidden state
    returns the loss, gradients on model parameters, and last hidden state
    
    x, hs, ys, ps = [], [], [], []
    h[-1] = 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(xh, xs[t]) + np.dot(xh, 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
    dloss, dh, dhy = np.zeros_like(xh), np.zeros_like(why), np.zeros_like(by)
    dnext = np.zeros_like(hs[0])
    for t in reversed(range(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] = 1 # backprop into y
        dhy += np.dot(dy, hs[t].T)
        db += dy
        dh += np.dot(why.T, dy) + dnext # backprop into h
        ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
        dhh += ddraw
        dloss += np.dot(ddraw, xs[t].T)
        dwhh += np.dot(ddraw, hs[t-1].T)
        dnext = np.dot(dh, T, ddraw)

    for dparam in [dloss, dh, dhy, db, dhh]:
        np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
    return loss, dloss, dh, dhy, db, dhh, hs[len(inputs)-1]
```
```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
$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$

$y_t = W_{hy}h_t$

### Recall:

add up gradient wrt. $h_t$ due to 
$y_t = W_{hy}h_t$ and $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$
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[i] = 1
        ixes.append(ix)
    return ixes
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-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoise rrranbyne 'nhthnee e plia tklrgd t o idoe ns,smtt h ne etie h,hrgrtr s 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 hearily, and behs to so arwage fiving were to it beloge, pavy 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 strain 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 nues 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 reining 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.
open source textbook on algebraic geometry

### The Stacks Project

<table>
<thead>
<tr>
<th>Part</th>
<th>Chapter</th>
<th>online</th>
<th>TeX source</th>
<th>view pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preliminaries</td>
<td>1. Introduction</td>
<td>online</td>
<td>tex</td>
<td>pdf</td>
</tr>
<tr>
<td></td>
<td>2. Conventions</td>
<td>online</td>
<td>tex</td>
<td>pdf</td>
</tr>
<tr>
<td></td>
<td>3. Set Theory</td>
<td>online</td>
<td>tex</td>
<td>pdf</td>
</tr>
<tr>
<td></td>
<td>4. Categories</td>
<td>online</td>
<td>tex</td>
<td>pdf</td>
</tr>
<tr>
<td></td>
<td>5. Topology</td>
<td>online</td>
<td>tex</td>
<td>pdf</td>
</tr>
<tr>
<td></td>
<td>6. Sheaves on Spaces</td>
<td>online</td>
<td>tex</td>
<td>pdf</td>
</tr>
<tr>
<td></td>
<td>7. Sites and Sheaves</td>
<td>online</td>
<td>tex</td>
<td>pdf</td>
</tr>
<tr>
<td></td>
<td>8. Stacks</td>
<td>online</td>
<td>tex</td>
<td>pdf</td>
</tr>
<tr>
<td></td>
<td>9. Fields</td>
<td>online</td>
<td>tex</td>
<td>pdf</td>
</tr>
<tr>
<td></td>
<td>10. Commutative Algebra</td>
<td>online</td>
<td>tex</td>
<td>pdf</td>
</tr>
</tbody>
</table>

**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

---

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
For $\Theta_{n=1,\ldots,m}$, where $L_n = 0$, hence we can find a closed subset $H$ in $H$ and any sets $F$ on $X$, $U$ is a closed immersion of $S$, then $U \to T$ is a separated algebraic space.

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

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

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

$$U = \bigcup U_i \times X U_i$$

which has a nonzero morphism we may assume that $f_i$ is of finite presentation over $S$. We claim that $O_{X,x}$ is a scheme where $x, x', x'' \in S'$ such that $O_{X,x} \to O_{X,x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $GL_S(f'/S')$ and we win.

To prove our claim we see that $F(U)$ is a covering of $X'$, and $T_i$ is an object of $F_{X/S}$ for $i > 0$ and $F_0$ exists and let $F_0$ be a presheaf of $O_X$-modules on $C$ as $F$-module. In particular $F = U/F$ we have to show that

$$\tilde{M} = T \otimes_{\text{Spec}(k)} O_{S,x} \otimes k$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)_{\text{fpf}}((\text{Sch}/S)_{\text{fpf}})$$

and

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

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

Proof. See discussion of sheaves of sets.

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

**Lemma 0.1.** Assume (3) and (4) by the construction in the description.

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

$$\text{Set}(A) = \Gamma(X, O_X, O_X).$$

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

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

Proof. This is form all sheaves of sheaves on $X$. But given a scheme $U$ and a surjective étale morphism $U \to X$. Let $U \cap U = \prod U_i \subseteq U$, be the scheme $X$ over $S$ at the schemes $X_i \to X$ and $U = \lim U_i$.

The following lemma surjective restcomposes of this implies that $F_{x_0} = F_{x_0}$.

**Lemma 0.2.** Let $X$ be a locally Noetherian scheme over $S$, $E = F_{X/S}$. Set $T = J_0 \subseteq T$. Since $J_0 \subseteq T$ are nonzero over $i_0 \leq p$ is a subset of $F_{X/0} \to A_2$ works.

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

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

$$D(O_X) = O_X(D)$$

where $K$ is an $F$-algebra where $\delta_{n+1}$ is a scheme over $S$.
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 $X_{\text{etale}}$ we have

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

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

**Lemma 0.2.** This is an integer $Z$ 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 $U \subseteq X$ be a canonical and locally of finite type. Let $X$ be a scheme. Let $X$ be a scheme which is equal to the formal complex.

The following to the construction of the lemma follows.

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

$$b : X \rightarrow Y' \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$ be a morphism of algebraic spaces over $S$ and $Y$.

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

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

Consider a common structure on $X$ and $X$ the functor $\mathcal{O}_X(U)$ which is locally of finite type.
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 & 0x00000000fffffff8) & 0x0000000f) << 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 ");
}
/*
 * 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>
Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
Searching for interpretable cells

[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—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;
                } else {
                    collect_signal(sig, pending, info);
                }
            }
        }
    }
    return sig;
}
```
Searching for interpretable cells

/* Duplicate LSM field information. The lsm_rule is opaque, so
* re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
    struct audit_field *sf)
{
    int ret = 0;
    char *lsm_str;
    /* our own copy of lsm_str */
    lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
    if (unlikely(!lsm_str))
        return -ENOMEM;
    df->lsm_str = lsm_str;
    /* our own (refreshed) copy of lsm_rule */
    ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
        (void **)&df->lsm_rule);
    /* Keep currently invalid fields around in case they
    * become valid after a policy reload. */
    if (ret == -EINVAL) {
        pr_warn("audit rule for LSM \"%s\" is invalid\n",
            df->lsm_str);
        ret = 0;
    }
    return ret;
}
Searching for interpretable cells

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

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
Recurrent Neural Network

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

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

sample!
<START>

<END> token => finish.

test image
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.”
“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.”

“a young boy is holding a baseball bat.”

“a cat is sitting on a couch with a remote control.”

“a woman holding a teddy bear in front of a mirror.”

“a horse is standing in the middle of a road.”
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
LSTM

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
RNN:

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

where 
- \( h \in \mathbb{R}^n \)
- \( W^l \) is \([n \times 2n]\)
RNN:

\[ h_t^l = \tanh \left( W^l \begin{pmatrix} h_t^{l-1} \\ h_t^{l-1} \end{pmatrix} \right) \]

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

A generalization of RNN. At \( l = 1 \):

- \( h_t^{l-1} = x_t \)
- \( W_1 = [W_{xh} \, W_{hh}] \)

Then it is equivalent to our original formulation:

\[ h_t = \tanh \left( W_{hh} h_{t-1} + W_{xh} x_t \right) \]
RNN:

\[ h_t^l = \tanh W^l \begin{pmatrix} h_{t-1}^l \\ h_{t-1}^l \end{pmatrix} \]

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

LSTM:

\[
\begin{pmatrix}
    i \\
    f \\
    o \\
    g
\end{pmatrix} = \begin{pmatrix}
    \text{sigm} \\
    \text{sigm} \\
    \text{sigm} \\
    \text{tanh}
\end{pmatrix} W^l \begin{pmatrix} h_{t-1}^l \\ h_{t-1}^l \end{pmatrix}
\]

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

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

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
LSTM - main idea

Slide adapted from MIT 6.S191 (IAP 2017), by Harini Suresh
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]

- $c$: cell state
- $h$: hidden state (cell output)
- $i$: input gate, weight of acquiring new information
- $f$: forget gate, weight of remembering old information
- $g$: transformed input ($[-1,+1]$)
- $o$: output gate, decides values to be activated based on current memory

$$
\begin{align*}
\begin{pmatrix}
i \\
f \\
o \\
g
\end{pmatrix} &= 
\begin{pmatrix}
sigmoid \\
sigmoid \\
sigmoid \\
tanh
\end{pmatrix} W^l \begin{pmatrix} h_{t-1}^l \end{pmatrix}
\end{align*}
$$

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

\[ \begin{align*}  
W & \quad \text{vector from before} \quad (h) \\
4n \times 2n & \\
\begin{array}{c}
\text{x} \\
\text{h}
\end{array} & \quad \text{vector from below} \quad (x) &  \\
\end{align*} \]

\[ \begin{align*}  
W & \quad \text{vector from before} \quad (h) \\
4n & \\
\begin{align*}  
\text{sigmoid} & \quad \text{sigmoid} \\
\text{sigmoid} & \quad \text{tanh}
\end{align*} & \quad \text{sigmoid} \\
4n & \\
4^*n &  \\
f & \quad \text{decides the degree of preservation for cell state, by scaling it with a number in [0,1]} \\
\end{align*} \]
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

g is a transformation of input / hidden state

\[
\begin{pmatrix}
i \\
f \\
o \\
g
\end{pmatrix} = \begin{pmatrix}
sigmoid \\
sigmoid \\
sigmoid \\
tanh
\end{pmatrix} \cdot W^l \begin{pmatrix}
h_{t-1} \\
h_{t-1}
\end{pmatrix}
\]

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

\[
h_t = o \odot \tanh(c_t)
\]
Long Short Term Memory (LSTM) [Hochreiter et al., 1997]

Add $g$ into the cell state, weighted by $i$ (weight of acquiring new information)

Alternative interpretation: $i^*g$ decouples the "influence of $g$" and "$g$ itself".
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]  

New hidden state is a scaled version of $\tanh$(cell state).

$o$: output gate, decides values to be activated based on current memory.
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]

Q: Why tanh?
A: Not very crucial, sometimes not used
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

\[
\begin{align*}
    i & = \sigma(W_i x + U_i h_{t-1}) \\
    f & = \sigma(W_f x + U_f h_{t-1}) \\
    o & = \sigma(W_o x + U_o h_{t-1}) \\
    g & = \tanh(W_g x + U_g h_{t-1}) \\
    c_t & = f \odot c_{t-1} + i \odot g \\
    h_t & = o \odot \tanh(c_t)
\end{align*}
\]
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]

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

\[
\begin{align*}
c_t^l &= f \odot c_{t-1}^l + i \odot g \\
h_t^l &= o \odot \text{tanh}(c_t^l) \\
\end{align*}
\]
Long Short Term Memory (LSTM)

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

\[
\begin{align*}
\text{cell state } c & \quad \xrightarrow{+} \quad \text{tanh} \\
& \quad \xrightarrow{\text{f}} \quad \text{h} \\
& \quad \xrightarrow{\text{i} \odot \text{g} \odot \text{o}} \quad \text{h}
\end{align*}
\]

\[
\begin{align*}
(i \ f \ o \ g) &= \begin{pmatrix} \text{sigmoid} \\ \text{sigmoid} \\ \text{sigmoid} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h^l_{t-1} \ h^l_t \end{pmatrix} \\
c^l_t &= f \odot c^l_{t-1} + i \odot g \\
h^l_t &= o \odot \text{tanh}(c^l_t)
\end{align*}
\]

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
LSTM has many variants...
Understanding gradient flow dynamics

Backprop signal video: http://imgur.com/gallery/vaNahKE

In RNN, the gradient vanishes much more quickly as we backprop from the last time step towards the first one

Therefore, RNN here cannot learn long time dependencies
Understanding gradient flow dynamics

RNN without any inputs

```python
H = 5  # dimensionality of hidden state
T = 50  # number of time steps
Whh = np.random.randn(H, H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
s = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    s[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, s[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T, 0, -1)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```
Understanding gradient flow dynamics

RNN without any inputs

```python
H = 5      # dimensionality of hidden state
T = 50     # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

[On the difficulty of training Recurrent Neural Networks, Pascanu et al., 2013]

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
Understanding gradient flow dynamics

RNN without any inputs

```python
H = 5  # dimensionality of hidden state
T = 50  # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
s = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    s[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, s[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H)  # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t]  # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t])  # backprop into previous hidden state
```

if the largest eigenvalue is < 1, gradient will vanish
if the largest eigenvalue is > 1, gradient will explode

[On the difficulty of training Recurrent Neural Networks, Pascanu et al., 2013]
Vanishing gradient problem

An example how vanishing gradient problem can affect RNNs:

“In France, I had a great time and I learnt some of the ____ language.”

our parameters are not trained to capture long-term dependencies, so the word we predict will mostly depend on the previous few words, not much earlier ones

Slide adapted from MIT 6.S191 (IAP 2017), by Harini Suresh
Understanding gradient flow dynamics

RNN without any inputs

```python
H = 5  # dimensionality of hidden state
T = 50  # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
s = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H)  # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t]  # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t])  # backprop into previous hidden state
```

If the largest eigenvalue is < 1, gradient will vanish. If the largest eigenvalue is > 1, gradient will explode.

Can control vanishing with LSTM.
Can control exploding with gradient clipping.

[On the difficulty of training Recurrent Neural Networks, Pascanu et al., 2013]

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
RNN
More prone to the vanishing gradient problem

LSTM
(ignoring forget gates)
Recall: “PlainNets” vs. ResNets

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

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
Vanishing gradient problem summary

To address this problem, use

- better activation function (eg, ReLU)
- proper initialization \((W=\text{Identity}, \text{bias}=\text{zeros})\) to prevent \(W\) from shrinking the gradients
- replace RNN cells with LSTM or other gated cells (next slide) to control what information is passed through
LSTM variants and friends

LSTM [LSTM: A Search Space Odyssey, Greff et al., 2015]

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

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

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