Lecture 10: Recurrent Neural Networks
Administrative

A1 regrade deadline is tonight

A2 due yesterday

Redeem your Google Cloud coupons by Sunday 5/6
Administrative: Midterm

In-class midterm on Tuesday!
Details on Piazza

If you need an alternate exam time then let us know by tomorrow

Midterm review session tomorrow
Last Time: CNN Architectures

AlexNet

Revolution of Depth

Figure copyright Kaiming He, 2016. Reproduced with permission.
Last Time: CNN Architectures

[Diagram showing CNN architectures such as VGG16, VGG19, GoogLeNet, and others with layers labeled by convolution and pooling operations.]

Fei-Fei Li & Justin Johnson & Serena Yeung

Revolution of Depth

Figure copyright Kaiming He, 2016. Reproduced with permission.
Last Time: CNN Architectures

Revolution of Depth

Figure copyright Kaiming He, 2016. Reproduced with permission.
Figures copyright Larsson et al., 2017. Reproduced with permission.
Last Time: CNN Architectures
Last Time: CNN Architectures

AlexNet and VGG have tons of parameters in the fully connected layers

**AlexNet:** ~62M parameters

- FC6: 256x6x6 -> 4096: 38M params
- FC7: 4096 -> 4096: 17M params
- FC8: 4096 -> 1000: 4M params

~59M params in FC layers!
ImageNet pretraining -> Instagram pretraining

Bigger models are saturated on ImageNet, but with more data bigger models do better.

Biggest network was pretrained on 3.5B Instagram images

Trained on 336 GPUs for 22 days

Mahajan et al, “Exploring the Limits of Weakly Supervised Pretraining”, arXiv 2018
ImageNet pretraining -> Instagram pretraining

Biggest network was pretrained on 3.5B Instagram images

Trained on 336 GPUs for 22 days
≈ $129,000 on Google Cloud

Bigger models are saturated on ImageNet, but with more data bigger models do better

Mahajan et al, “Exploring the Limits of Weakly Supervised Pretraining”, arXiv 2018
Today: Recurrent Neural Networks
“Vanilla” Neural Network

Vanilla Neural Networks
Recurrent Neural Networks: Process Sequences

- One to one
- One to many
- Many to one
- Many to many

Example: Image Captioning
Image -> sequence of words
Recurrent Neural Networks: Process Sequences

one to one  |  one to many  |  many to one  |  many to many  |  many to many

e.g. Sentiment Classification
sequence of words -> sentiment
Recurrent Neural Networks: Process Sequences

- One to one
- One to many
- Many to one
- Many to many
- Machine Translation
  seq of words -> seq of words
Recurrent Neural Networks: Process Sequences

one to one

one to many

many to one

many to many

many to many

e.g. Video classification on frame level
Sequential Processing of Non-Sequence Data

Classify images by taking a series of “glimpses”
Sequential Processing of Non-Sequence Data

Generate images one piece at a time!

Gregor et al, "DRAW: A Recurrent Neural Network for Image Generation", ICML 2015
Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with permission.
Recurrent Neural Network
Recurrent Neural Network

usually want to predict a vector at some time steps
We can process a sequence of vectors $x$ by applying a **recurrence formula** at every time step:

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

The diagram shows:
- $h_t$: new state
- $h_{t-1}$: old state
- $x_t$: input vector at some time step
- $f_W$: some function with parameters $W$
Recurrent Neural Network

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

\[
   h_t = f_W(h_{t-1}, x_t)
\]

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

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

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

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

$$y_t = W_{hy} h_t$$

Sometimes called a “Vanilla RNN” or an “Elman RNN” after Prof. Jeffrey Elman
RNN: Computational Graph

\[ h_0 \xrightarrow{\mathbf{f}_W} h_1 \]

\[ x_1 \]
RNN: Computational Graph

\[
\begin{align*}
& h_0 \xrightarrow{f_W} h_1 \xrightarrow{f_W} h_2 \\
\text{with} & \quad x_1 \quad x_2
\end{align*}
\]
RNN: Computational Graph
RNN: Computational Graph

Re-use the same weight matrix at every time-step
RNN: Computational Graph: Many to Many

\[
\begin{align*}
  &h_0 \rightarrow f_W \rightarrow h_1 \rightarrow f_W \rightarrow h_2 \rightarrow f_W \rightarrow h_3 \rightarrow \ldots \rightarrow h_T \\
  &x_1 \rightarrow f_W \rightarrow h_1 \rightarrow f_W \rightarrow h_2 \rightarrow f_W \rightarrow h_3 \rightarrow \ldots \rightarrow h_T \\
  &y_1 \rightarrow f_W \rightarrow h_1 \rightarrow f_W \rightarrow h_2 \rightarrow f_W \rightarrow h_3 \rightarrow \ldots \rightarrow h_T \\
  &y_2 \rightarrow f_W \rightarrow h_1 \rightarrow f_W \rightarrow h_2 \rightarrow f_W \rightarrow h_3 \rightarrow \ldots \rightarrow h_T \\
  &y_3 \rightarrow f_W \rightarrow h_1 \rightarrow f_W \rightarrow h_2 \rightarrow f_W \rightarrow h_3 \rightarrow \ldots \rightarrow h_T \\
  &y_T \rightarrow f_W \rightarrow h_1 \rightarrow f_W \rightarrow h_2 \rightarrow f_W \rightarrow h_3 \rightarrow \ldots \rightarrow h_T \\
\end{align*}
\]
RNN: Computational Graph: Many to Many

\[ W \]

\[ h_0 \rightarrow f_W \rightarrow h_1 \rightarrow f_W \rightarrow h_2 \rightarrow f_W \rightarrow h_3 \rightarrow \ldots \rightarrow h_T \]

\[ x_1 \rightarrow y_1 \rightarrow L_1 \]
\[ x_2 \rightarrow y_2 \rightarrow L_2 \]
\[ x_3 \rightarrow y_3 \rightarrow L_3 \]

\[ y_T \rightarrow L_T \]
RNN: Computational Graph: Many to Many

\[ h_0 \rightarrow f_W \rightarrow h_1 \rightarrow f_W \rightarrow h_2 \rightarrow f_W \rightarrow h_3 \rightarrow \ldots \rightarrow h_T \]

\[ W \]

\[ y_1 \rightarrow L_1 \rightarrow y_2 \rightarrow L_2 \rightarrow y_3 \rightarrow L_3 \rightarrow y_T \rightarrow L_T \]
RNN: Computational Graph: Many to One

```
\[ h_0 \xrightarrow{f_W} h_1 \xrightarrow{f_W} h_2 \xrightarrow{f_W} h_3 \xrightarrow{\cdots} h_T \]
```

\[ W \]

\[ x_1 \]

\[ x_2 \]

\[ x_3 \]

\[ y \]
RNN: Computational Graph: One to Many

\[
\begin{align*}
\text{h}_0 & \rightarrow f_W \rightarrow \text{h}_1 \\
\text{h}_1 & \rightarrow f_W \rightarrow \text{h}_2 \\
\text{h}_2 & \rightarrow f_W \rightarrow \text{h}_3 \\
& \vdots \\
\text{h}_T & \rightarrow f_W \rightarrow \text{y}_1 \\
\text{y}_1 & \rightarrow f_W \rightarrow \text{y}_2 \\
\text{y}_2 & \rightarrow f_W \rightarrow \text{y}_3 \\
& \vdots \\
\text{y}_T & \rightarrow f_W \rightarrow y_T \\
\end{align*}
\]
Sequence to Sequence: Many-to-one + one-to-many

**Many to one**: Encode input sequence in a single vector

Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014
Sequence to Sequence: Many-to-one + one-to-many

**Many to one**: Encode input sequence in a single vector

**One to many**: Produce output sequence from single input vector

Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014
Example:
Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
Example:
Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
Example:
Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
Example:
Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model
Example:
Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model
Example:
Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model
Example:
Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model.
Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient
Truncated Backpropagation through time

Run forward and backward through chunks of the sequence instead of whole sequence
**Truncated** Backpropagation through time

Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps.
Truncated Backpropagation through time
```python
min-char-rnn.py gist: 112 lines of Python

```
THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou art now the world's refreshment, art,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-seeing shame, and thriftless praise.
How much more praise deservèst thou beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.
at first:

"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."

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.

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

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

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

Second Lord:
They would be ruled after this chamber, and
my fair nes 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.
The Stacks Project: open source algebraic geometry textbook

- Latex source: http://stacks.math.columbia.edu/
- The stacks project is licensed under the GNU Free Documentation License
For $\bigoplus_{n=1, \ldots, m}$ where $L_m = 0$, hence we can find a closed subset $H$ in $H$ and any sets $F$ on $X$. If $U$ is a closed immersion of $S$, then $U \rightarrow T$ is a separated algebraic space.

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

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

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

$$U = \bigcup U_i \times S_i U_i$$

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

To prove we study that $\mathcal{F}_U$ is a covering of $\mathcal{X}'$, and $T_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $F_i$ exists and let $F_i$ be a presheaf of $\text{O}_X$-modules on $C$ as a $F$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\tilde{M}^* = T^* \otimes_{\text{Spec}(k)} O_{S,x} - i_{x}^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = \left( \text{Sch} / S \right)_{\text{proj}} \times \left( \text{Sch} / S \right)_{\text{proj}}$$

and

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

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

**Lemma 0.1.** Assume $\mathcal{F}$ and $\mathcal{G}$ by the construction in the description.

Suppose $X = \lim |X|$ 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 $Q \rightarrow C_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are connected. If $T$ is surjective we may assume that $T$ is connected with residue fields $S$. 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 form all sheaves of sheaves on $X$. But given a scheme $U$ and a surjective étale morphism $U \rightarrow X$. Let $U \cap U' = \bigcap_{i=1, \ldots, n} U_i$ be the scheme $X$ over $S$ at the schemes $X_i \rightarrow X$ and $U = \lim_i X_i$.

The following lemma surjective resto decomposes of this implies that $\mathcal{F}_x = \mathcal{F}_x = \mathcal{F}_{X_i}$.

**Lemma 0.2.** Let $X$ be a locally Noetherian scheme over $S$, $E = \mathcal{F}_{X/S}$. Set $T = \mathcal{F}_{X/S}$, $T_i \subset T_i$. Since $\mathcal{F} \subset \mathcal{F}$ are nonzero over $i_0 \leq p$ is a subset of $\mathcal{J}_{n,0} \otimes A_2$ 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 $S$. □


**Lemma 0.1.** Let $C$ be a set of the construction.

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

\[ O_{O_X} = O_X(L) \]

Proof. This is an algebraic space with the composition of sheaves $F$ on $X_{etale}$ we have

\[ O_X(F) = \{ \text{morph}_1 \times_{O_X} (G, F) \} \]

where $G$ defines an isomorphism $F \to F$ of $O$-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 $b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X$.

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 $F$ be a quasi-coherent sheaf of $O_X$-modules. The following are equivalent

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

Consider a common structure on $X$ and $X$ the functor $O_X(U)$ which is locally of finite type.

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

Proof. This is clear that $G$ is a finite presentation, see Lemmas ??.

A reduced above we conclude that $U$ is an open covering of $C$. The functor $F$ is a field

\[ O_{X_x} \to F \to \text{im}(O_{X_{total}}) \to O_{X_x} \]

is an isomorphism of covering of $O_{X_x}$. If $F$ is the unique element of $F$ such that $X$ is an isomorphism.

The property $F$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $O_x$-algebra with $F$ are opens of finite type over $S$. If $F$ is a scheme theoretic image points.

If $F$ is a finite direct sum $O_{X_x}$ is a closed immersion, see Lemma ??.

This is a sequence of $F$ is a similar morphism.
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 & 0x0000000000000008) & 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, &offset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
/*
 * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software Foundation,
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

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

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/sect.h>
#include <asm/pgproto.h>
```c
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setev.h>
#include <asm/pgproto.h>

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

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

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

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

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Searching for interpretable cells

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

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission
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

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission
Searching for interpretable cells

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

if statement cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission
Searching for interpretable cells

```
/* Duplicate LSM field information. The lsm_rule is opaque, so */
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;
}
```

quote/comment cell
Searching for interpretable cells

code depth cell

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

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

Recurrent Neural Network

Convolutional Neural Network
test image
before:

\[ h = \tanh(W_{xh} \cdot x + W_{hh} \cdot h) \]

now:

\[ h = \tanh(W_{xh} \cdot x + W_{hh} \cdot h + W_{ih} \cdot v) \]
<START>

```
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
```

test image

```
straw
hat
```

custom model

```
x0
<START>
```

```y0
h0
```

```y1
h1
```

sample!
<START>

test image

- conv-64
- maxpool
- conv-128
- maxpool
- conv-256
- maxpool
- conv-512
- maxpool
- conv-512
- maxpool
- FC-4096
- FC-4096

y0 → y1 → y2

h0 → h1 → h2

x0: <START>

straw

hat
sample <END> token => finish.
Image Captioning: Example Results

A cat sitting on a suitcase on the floor
A cat is sitting on a tree branch
A dog is running in the grass with a frisbee
A white teddy bear sitting in the grass
Two people walking on the beach with surfboards
A tennis player in action on the court
Two giraffes standing in a grassy field
A man riding a dirt bike on a dirt track

Captions generated using neuraltalk2
All images are CC0 Public domain: cat suitcase, cat tree, dog, bear, surfers, tennis, giraffe, motorcycle
Image Captioning: Failure Cases

A woman is holding a cat in her hand

A woman standing on a beach holding a surfboard

A person holding a computer mouse on a desk

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball
Image Captioning with Attention

RNN focuses its attention at a different spatial location when generating each word

Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio, 2015. Reproduced with permission.
Image Captioning with Attention

Image Captioning with Attention

CNN

Image:
H x W x 3

Features:
L x D

Distribution over L locations

Image Captioning with Attention

Image Captioning with Attention

- CNN
- Image: H x W x 3
- Features: L x D
- Weighted combination of features
- Distribution over L locations
- Weighted features: D
- First word

Image Captioning with Attention

CNN

Image: $H \times W \times 3$

Features: $L \times D$

Weighted combination of features

Distribution over $L$ locations

Weighted features: $D$

First word

Image Captioning with Attention

Image Captioning with Attention

Image Captioning with Attention

Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Richard S. Zemel, and Yoshua Bengio, 2015. Reproduced with permission.
Image Captioning with Attention

Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio, 2015. Reproduced with permission.
Visual Question Answering

Figure from Zhu et al, copyright IEEE 2016. Reproduced for educational purposes.
Visual Question Answering: RNNs with Attention

Multilayer RNNs

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

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

LSTM:

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

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

\[ h_t^l = o \odot \tanh(c_t^l) \]
Vanilla RNN Gradient Flow

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

$$= \tanh \left( (W_{hh} \ W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

$$= \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$
Vanilla RNN Gradient Flow

Backpropagation from $h_t$ to $h_{t-1}$ multiplies by $W$ (actually $W_{hh}^T$)

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

$$= \tanh \left( (W_{hh} \ W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

$$= \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Computing gradient of $h_0$ involves many factors of $W$ (and repeated tanh)

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Computing gradient of $h_0$ involves many factors of $W$ (and repeated tanh)

Largest singular value > 1: **Exploding gradients**

Largest singular value < 1: **Vanishing gradients**

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Computing gradient of $h_0$ involves many factors of $W$ (and repeated tanh)

Largest singular value $> 1$: **Exploding gradients**

Largest singular value $< 1$: **Vanishing gradients**

**Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```
Vanilla RNN Gradient Flow

Computing gradient of $h_0$ involves many factors of $W$ (and repeated tanh)

- Largest singular value > 1: Exploding gradients
- Largest singular value < 1: Vanishing gradients

Change RNN architecture

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Long Short Term Memory (LSTM)

Vanilla RNN

\[ h_t = \tanh \left( W \left( h_{t-1}^{\text{prev}} \right) + x_t \right) \]

LSTM

\[
\begin{pmatrix}
    i \\
    f \\
    o \\
    g
\end{pmatrix} =
\begin{pmatrix}
    \sigma \\
    \sigma \\
    \tanh \\
    \tanh
\end{pmatrix}
W
\begin{pmatrix}
    h_{t-1} \\
    x_t
\end{pmatrix}
\]

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

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

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]

\[
\begin{align*}
\text{i: Input gate, whether to write to cell} \\
\text{f: Forget gate, Whether to erase cell} \\
\text{o: Output gate, How much to reveal cell} \\
\text{g: Gate gate (?), How much to write to cell}
\end{align*}
\]

\[
\begin{align*}
(i) & = \sigma \\
(f) & = \sigma \\
(o) & = \sigma \\
(g) & = \text{tanh}
\end{align*}
\]

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

[Hochreiter et al., 1997]

\[
\begin{align*}
    &c_{t-1} & \rightarrow & \oplus & + & c_t \\
    &W & \rightarrow & \circ & \oplus & h_t \\
    &h_{t-1} & \rightarrow & \circ & \oplus & h_t \\
    &x_t & \rightarrow & \circ & \oplus & h_t \\

    c_t &= f \odot c_{t-1} + i \odot g \\
    h_t &= o \odot \tanh(c_t)
\end{align*}
\]
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Backpropagation from $c_t$ to $c_{t-1}$ only elementwise multiplication by $f$, no matrix multiply by $W$

\[
\begin{pmatrix}
    i \\
    f \\
    o \\
    g
\end{pmatrix} =
\begin{pmatrix}
    \sigma \\
    \sigma \\
    \sigma \\
    \tanh
\end{pmatrix}
W
\begin{pmatrix}
    h_{t-1} \\
    x_t
\end{pmatrix}
\]

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

\[h_t = o \odot \tanh(c_t)\]
Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!
Long Short Term Memory (LSTM): Gradient Flow
[Hochreiter et al., 1997]

Uninterrupted gradient flow!

Similar to ResNet!
Long Short Term Memory (LSTM): Gradient Flow
[Hochreiter et al., 1997]

Uninterrupted gradient flow!

Similar to ResNet!

In between:
Highway Networks

\[ g = T(x, W_T) \]
\[ y = g \odot H(x, W_H) + (1 - g) \odot x \]

Srivastava et al, "Highway Networks",
ICML DL Workshop 2015
Other RNN Variants

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

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

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

**MUT1**:

\[
\begin{align*}
    z &= \text{sigm}(W_{xz}x_t + b_z) \\
    r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\
    h_{t+1} &= \text{tanh}(W_{hh}(r \odot h_t) + \text{tanh}(x_t) + b_h) \odot z \\
    &+ h_t \odot (1 - z)
\end{align*}
\]

**MUT2**:

\[
\begin{align*}
    z &= \text{sigm}(W_{xz}x_t + W_{hz}h_t + b_z) \\
    r &= \text{sigm}(x_t + W_{hr}h_t + b_r) \\
    h_{t+1} &= \text{tanh}(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\
    &+ h_t \odot (1 - z)
\end{align*}
\]

**MUT3**:

\[
\begin{align*}
    z &= \text{sigm}(W_{xz}x_t + W_{hz} \tanh(h_t) + b_z) \\
    r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\
    h_{t+1} &= \text{tanh}(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\
    &+ h_t \odot (1 - z)
\end{align*}
\]
Summary

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