Lecture 7: Recurrent Neural Networks
Training “Feedforward” Neural Networks

1. **One time setup:** activation functions, preprocessing, weight initialization, regularization, gradient checking

2. **Training dynamics:** babysitting the learning process, parameter updates, hyperparameter optimization

3. **Evaluation:** model ensembles, test-time augmentation, transfer learning
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- 2010: Lin et al (28.2%)
- 2011: Sanchez & Perronnin (25.8%)
- 2012: Krizhevsky et al (AlexNet) (16.4%)
- 2013: Zeiler & Fergus (11.7%)
- 2014: Simonyan & Zisserman (VGG) (7.3%)
- 2014: Szegedy et al (GoogLeNet) (6.7%)
- 2015: He et al (ResNet) (3.6%)
- 2016: Shao et al (3%)
- 2017: Hu et al (SENet) (2.3%)
- Human (5.1%)

Architectures:
- AlexNet: 8 layers
- VGG: 19 layers
- GoogLeNet: 22 layers
- ResNet: 152 layers
- SENet: 152 layers

Fei-Fei Li, Ehsan Adeli

Lecture 8 - 3
April 23, 2023
Comparing complexity...


Comparing complexity...

Inception-v4: Resnet + Inception!


Comparing complexity...


Comparing complexity...


Comparing complexity...

AlexNet:
Smaller compute, still memory heavy, lower accuracy


Comparing complexity...


More complexity...

More complexity...

https://dev.to/rohitgupta24/convnext-a-convnet-for-the-2020s-part-i-i43
Evaluate models and tune hyperparameters

https://docs.wandb.ai/guides/track/app
Today: Recurrent Neural Networks
“Vanilla” Neural Network

Vanilla Neural Networks
Recurrent Neural Networks: Process Sequences

e.g. Image Captioning
image -> sequence of words
Recurrent Neural Networks: Process Sequences

e.g. action prediction
sequence of video frames -> action class
Recurrent Neural Networks: Process Sequences

E.g. Video Captioning
Sequence of video frames -> caption
Recurrent Neural Networks: Process Sequences

e.g. Video classification on frame level
Recurrent Neural Network

\[ x \rightarrow \text{RNN} \rightarrow y \]
Recurrent Neural Network

Key idea: RNNs have an “internal state” that is updated as a sequence is processed.
Unrolled RNN

\[
\begin{align*}
    y_1 & \quad \text{RNN} \quad x_1 \\
    y_2 & \quad \text{RNN} \quad x_2 \\
    y_3 & \quad \text{RNN} \quad x_3 \\
    \quad & \quad \ldots \\
    y_t & \quad \text{RNN} \quad x_t 
\end{align*}
\]
RNN hidden state update

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

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

- $h_t$: new state
- $h_{t-1}$: old state
- $x_t$: input vector at some time step
- $f_W$: some function with parameters $W$
RNN output generation

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

$$y_t = f_{W_{hy}}(h_t)$$

- $y_t$: output
- $h_t$: new state
- $f_{W_{hy}}$: another function with parameters $W_{hy}$
Recurrent Neural Network

\[ \text{RNN} \]

\[ x_1 \rightarrow y_1 \]
\[ x_2 \rightarrow y_2 \]
\[ x_3 \rightarrow y_3 \]
\[ \ldots \]
\[ x_t \rightarrow y_t \]
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 $h$:

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

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

$$y_t = W_{hy}h_t$$

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

![Diagram of RNN computational graph]

- $h_0$ to $f_W$ to $h_1$
- $x_1$ to $f_W$ and $h_0$
RNN: Computational Graph

\[ h_0 \xrightarrow{f_W} h_1 \xrightarrow{f_W} h_2 \]

\[ x_1 \]

\[ x_2 \]
RNN: Computational Graph

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

- \[ x_1 \]
- \[ x_2 \]
- \[ x_3 \]
RNN: Computational Graph

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

\[
\begin{align*}
W & \rightarrow x_1 \rightarrow f_W \rightarrow h_0 \\
& \rightarrow x_2 \rightarrow f_W \rightarrow h_1 \\
& \rightarrow x_3 \rightarrow f_W \rightarrow h_2 \\
& \rightarrow \ldots \rightarrow h_T
\end{align*}
\]
RNN: Computational Graph: Many to Many

![Diagram of RNN computational graph]

- $y_1 \rightarrow L_1$
- $y_2 \rightarrow L_2$
- $y_3 \rightarrow L_3$
- $y_T \rightarrow L_T$

- $h_0 \rightarrow f_W \rightarrow h_1$
- $h_1 \rightarrow f_W \rightarrow h_2$
- $h_2 \rightarrow f_W \rightarrow h_3$
- $h_3 \rightarrow \ldots \rightarrow h_T$

- $W \rightarrow x_1 \rightarrow h_1$
- $W \rightarrow x_2 \rightarrow h_2$
- $W \rightarrow x_3 \rightarrow h_3$

Fei-Fei Li, Ehsan Adeli
RNN: Computational Graph: Many to Many

\[ \begin{align*}
    h_0 & \rightarrow f_W \rightarrow h_1 \\
    x_1 & \rightarrow h_1 \\
    y_1 & \rightarrow L_1 \\
    W & \rightarrow h_1 \\
    h_1 & \rightarrow f_W \rightarrow h_2 \\
    x_2 & \rightarrow h_2 \\
    y_2 & \rightarrow L_2 \\
    W & \rightarrow h_2 \\
    h_2 & \rightarrow f_W \rightarrow h_3 \\
    x_3 & \rightarrow h_3 \\
    y_3 & \rightarrow L_3 \\
    W & \rightarrow h_3 \\
    h_3 & \rightarrow \cdots \rightarrow h_T \\
    y_T & \rightarrow L_T \\
    W & \rightarrow h_T \\
    L & \rightarrow h_T
\end{align*} \]
RNN: Computational Graph: Many to One

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

\[ W \xrightarrow{x_1} f_W \xrightarrow{x_2} f_W \xrightarrow{x_3} f_W \]

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

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

Where:
- \( h_i \) is the hidden state at time step \( i \)
- \( W \) is the weight matrix
- \( x \) is the input vector
- \( y_i \) are the output vectors for each time step

The diagram shows the flow of information through the RNN, transforming inputs into hidden states and eventually to outputs.
RNN: Computational Graph: One to Many

\[
\begin{align*}
h_0 & \xrightarrow{W} f_W \xrightarrow{h_1} f_W \xrightarrow{h_2} f_W \xrightarrow{h_3} \cdots \xrightarrow{h_T} \\
W & \xrightarrow{x} x\xrightarrow{?} y_1 \xrightarrow{?} y_2 \xrightarrow{?} y_3 \xrightarrow{?} \cdots \xrightarrow{?} y_T
\end{align*}
\]
RNN: Computational Graph: One to Many

\[ h_0 \xrightarrow{W} f_W h_1 \xrightarrow{f_W} h_2 \xrightarrow{f_W} h_3 \ldots \xrightarrow{f_W} h_T \]

\[ x \xrightarrow{W} 0 \xrightarrow{W} 0 \xrightarrow{W} 0 \]
RNN: Computational Graph: One to Many

h₀ \rightarrow f_W \rightarrow h₁ \rightarrow f_W \rightarrow h₂ \rightarrow f_W \rightarrow h₃ \rightarrow \cdots \rightarrow h_T

W \rightarrow x \rightarrow y₁ \rightarrow y₁ \rightarrow y₂ \rightarrow y₂ \rightarrow y₃ \rightarrow y₃ \rightarrow y₄ \rightarrow \cdots \rightarrow y_{T-1}
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
Matrix multiplication with a one-hot vector just extracts a column from the weight matrix. We often put a separate embedding layer between the input and hidden layers.
Backpropagation through time

Loss

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
def sample(c, seed_ix, n):
    # Sample a sequence of integers from the model
    # Return a list of integers
    x = np.zeros((n, 1))
    y = np.copy(x)
    for i in range(n):
        # Sample from the model
        x = sample_x[i, x]
        y = np.append(y, x)
        return y
```

This is a Python code snippet from a GitHub Gist titled `min-char-rnn.py gist: 112 lines of Python` (https://gist.github.com/karpathy/d4dee566867f8291f086)

---

Fei-Fei Li, Ehsan Adeli
Lecture 8 - 56
April 23, 2023
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 Fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buried thy content,
And tender churl mak'st waste in nigardy:
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-eating shame, and thriftless praise.
How much more praise deserve'd thy 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
colaniogennnc Phe lism thond hon at. MeiDimorotion in ther thize."

"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.
The Stacks Project: open source algebraic geometry textbook

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

http://stacks.math.columbia.edu/
The stacks project is licensed under the GNU Free Documentation License

Latex source
Lemma 0.1. Assume (3) and (3) 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 $\text{Set}(A) = \Gamma(X, \mathcal{O}_X, \mathcal{O}_X)$.

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{/X}$ 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 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 $Y = \text{Spec}(A)$.

Proof. This is form all sheaves of sheaves on $X$. But given a scheme $U$ and a surjective étale morphism $U \to X$. Let $U \cap U = \bigcup_{i=1}^{n} U_i$ be the scheme $X$ over $S$ at the schemes $X_i \to X$ and $U = \lim_{\to} X_i$.

The following lemma surjective restitutes compositions of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_x = \mathcal{F}_{x_0}^\mathcal{F}$.

Lemma 0.2. Let $X$ be a locally Noetherian scheme over $S$, $E = \mathcal{F}_{X/S}$. Set $\mathcal{T}_x = J_x \subset \mathcal{T}_x$. Since $\mathcal{T}_x \subset \mathcal{T}_x'$ are nonzero over $i_0 \leq \mathcal{p}$ is a subset of $\mathcal{J}_x$ which works.

Lemma 0.3. In Situation ??2. 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 $\mathcal{D}(\mathcal{O}_X) = \mathcal{O}_X(D)$.

where $K$ is an $F$-algebra where $\delta_{n+1}$ is a scheme over $S$.
Proof. Omitted. □

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 \( X_{\text{etale}} \) we have

\[ O_X(F) = \{ \text{morph}_{1} \times_{O_X} (G, F) \} \]

where G defines an isomorphism F \( \rightarrow \) 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 \( \subset X \) 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 \( F \) 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.

This since \( F \in F \) and \( x \in G \) the diagram

\[
\begin{array}{ccc}
S & \rightarrow & \xi \\
\downarrow & & \downarrow \\
\xi' & \rightarrow & \mathcal{O}_{X'}
\end{array}
\]

is a limit. Then G is a finite type and assume S is a flat \( F \) and \( G \) is a finite type \( F \). This is of finite type diagrams, and

- the composition of \( G \) is a regular sequence,
- \( O_{X'} \) is a sheaf of rings. □

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 neigbourhood 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"

\[ O_{X,Z} \rightarrow F \rightarrow \text{Spec}(O_{X,Z}) \rightarrow O_{X,Z} \rightarrow O_{X,Z}(O_{X,Z}) \]

is an isomorphism of covering of \( O_{X,Z} \). 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 \( O_{X,Z} \)-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 \( O_{X,Z} \) 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 << 1))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff8) & 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 ");
}
/*
 * 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/ctimevent.h>

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

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

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

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \  
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;
  }
Add this image of a rocketship:
https://i1.sndcdn.com/artworks-j8xjG7zc1wmTeO7b-O6l83w-t500x500.jpg
Searching for interpretable cells
Searching for interpretable cells
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

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

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

code depth cell
RNN tradeoffs

RNN Advantages:
- Can process any length of the input
- Computation for step \( t \) can (in theory) use information from many steps back
- Model size does not increase for longer input
- The same weights are applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:
- Recurrent computation is slow
- In practice, difficult to access information from many steps back
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:
\[ h = \tanh(W_{xh} \cdot x + W_{hh} \cdot h + W_{ih} \cdot v) \]
test image

sample!
test image
test image

sample!

Fei-Fei Li, Ehsan Adeli

Lecture 8 - 85 April 23, 2023
Fei-Fei Li, Ehsan Adeli

Lecture 8 - April 23, 2023

test image
test image

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
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
Visual Question Answering (VQA)

Figure from Zhu et al, copyright IEEE 2016. Reproduced for educational purposes.
Visual Question Answering (VQA)

“How many horses are in this image?”

Visual Dialog: Conversations about images

Figures from Das et al, copyright IEEE 2017. Reproduced with permission.

Fei-Fei Li, Ehsan Adeli

Lecture 8 - 92 April 23, 2023
Agent encodes instructions in language and uses an RNN to generate a series of movements as the visual input changes after each move.

Figures from Wang et al, copyright IEEE 2017. Reproduced with permission.
Multilayer RNNs
Long Short Term Memory (LSTM)

Vanilla RNN

\[ h_t = \tanh \left( W \left( h_{t-1}, x_t \right) \right) \]

LSTM

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

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

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

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Vanilla RNN Gradient Flow

\[ h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \]
\[ = \tanh \left( (W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \]
\[ = \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \]

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

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

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

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

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

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

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

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

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

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

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh} h_{t-1} + W_{hx} x_t) W_{hh}$$
Vanilla RNN Gradient Flow

\[ \frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \]

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

Gradients over multiple time steps:

\[ \frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \]

\[ \frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_T}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} \]

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

Gradients over multiple time steps:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_T}{\partial h_{T-1}} \ldots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left( \prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}
\]

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

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \\
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_T}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left( \prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}
\]

\[
\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}
\]

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

Gradients over multiple time steps:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

Almost always < 1
Vanishing gradients

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} (\prod_{t=2}^{T} tanh'(W_{hh} h_{t-1} + W_{xh} x_t)) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}
\]

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

Gradients over multiple time steps:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

What if we assumed no non-linearity?

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

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

What if we assumed no non-linearity?

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

Largest singular value > 1:
Exploding gradients

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W^{T-1} \frac{\partial h_1}{\partial W}
\]

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
Gradients over multiple time steps:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W^{T-1} \frac{\partial h_1}{\partial W}
\]

What if we assumed no non-linearity?

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

Gradient clipping: Scale gradient if its norm is too big

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

Gradients over multiple time steps:

What if we assumed no non-linearity?

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

Largest singular value > 1: Exploding gradients

Largest singular value < 1: Vanishing gradients

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W^{T-1} \frac{\partial h_1}{\partial W}
\]

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

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

LSTM

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

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

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

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Long Short Term Memory (LSTM)

Vanilla RNN

\[ h_t = \tanh \left( W \left( h_{t-1}, x_t \right) \right) \]

LSTM

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

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

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

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

\[ \text{vector from below (x)} \]

\[ \text{vector from before (h)} \]

4h x 2h

4h

4\ast h

\[
\begin{align*}
\text{sigmoid} & \quad \text{sigmoid} & \quad \text{sigmoid} \\
\text{sigmoid} & \quad \text{tanh} & \quad \text{tanh} \\
\end{align*}
\]

\[
\begin{align*}
i & \quad f & \quad o \\
g & \quad \text{output} \\
\end{align*}
\]
Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

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

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

\[
c_t = f \odot c_{t-1} + i \odot g \\
h_t = o \odot \tanh(c_t)
\]

\textbf{g: Gate gate (?), How much to write to cell}
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]

\[
\begin{align*}
(i) & = \begin{pmatrix}
\sigma \\
\sigma \\
\sigma \\
tanh
\end{pmatrix} W \begin{pmatrix}
h_{t-1} \\
x_t
\end{pmatrix} \\
\sigma & = \text{sigmoid} \\
tanh & = \text{tanh}
\end{align*}
\]

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

\[
h_t = o \odot \text{tanh}(c_t)
\]

i: Input gate, whether to write to cell

g: Gate gate (?), How much to write to cell
Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

**Vector from below (x):**
- $x$
- $h$

**Vector from before (h):**
- $h$
- $x$

**Input gate (i):** Whether to write to cell

**Forget gate (f):** Whether to erase cell

**Output gate (o):** How much to reveal cell

**Gate gate (g):** How much to write to cell

$4h \times 2h \rightarrow 4h \rightarrow 4*h$

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

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

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

- i: Input gate, whether to write to cell
- f: Forget gate, Whether to erase cell
- o: Output gate, How much to reveal cell
- g: Gate gate (?), How much to write to cell

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

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

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

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

\[
\begin{align*}
(i) & = \left( \begin{array}{c}
\sigma \\
\sigma \\
\sigma \\
tanh
\end{array} \right) W \left( h_{t-1} \right) \\
(f) & = i \odot c_{t-1} + i \odot g \\
h_t & = o \odot \tanh(c_t)
\end{align*}
\]
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 \(W\)

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

\[
c_t = f \odot c_{t-1} + i \odot g \\
h_t = o \odot \tanh(c_t)
\]
Long Short Term Memory (LSTM): Gradient Flow
[Hochreiter et al., 1997]

Uninterrupted gradient flow!

$C_0 \rightarrow \text{stack} \rightarrow W \rightarrow f \rightarrow g \rightarrow \text{tanh} \rightarrow c \rightarrow + \rightarrow C_1$
Do LSTMs solve the vanishing gradient problem?

The LSTM architecture makes it easier for the RNN to preserve information over many timesteps
- e.g. if the \( f = 1 \) and the \( i = 0 \), then the information of that cell is preserved indefinitely.
- By contrast, it’s harder for vanilla RNN to learn a recurrent weight matrix \( W_h \) that preserves info in hidden state

LSTM doesn’t guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies
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

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


Fei-Fei Li, Ehsan Adeli

Lecture 8 - 121 April 23, 2023
Other RNN Variants

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

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

[LSTM: A Search Space Odyssey, Greff et al., 2015]
Neural Architecture Search for RNN architectures

LSTM cell

Cell they found

Zoph et Le, “Neural Architecture Search with Reinforcement Learning”, ICLR 2017
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, as well as new paradigms for reasoning over sequences
- Better understanding (both theoretical and empirical) is needed.
Next time: Attention and Transformers