Lecture 8: Recurrent Neural Networks
Administrative

- Discussion section tomorrow 1:30-2:20pm PT
  PyTorch / TensorFlow Review

- Additional optional section today between 5:30-6:45pm PT
  at building 460 room 429
Training “Feedforward” Neural Networks

1. **One time set up**: 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
Today: Recurrent Neural Networks
“Vanilla” Neural Network

one to one

Vanilla Neural Networks
Recurrent Neural Networks: Process Sequences

one to one

one to many

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

one to one

one to many

many to one

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

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!
Recurrent Neural Network

RNN

y

x
Recurrent Neural Network

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

\[
\begin{align*}
\text{Unrolled RNN} & \\
\vdots & \\
RNN & \rightarrow RNN & \rightarrow RNN & \rightarrow \ldots & \rightarrow RNN \\
& \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
x_1 & \rightarrow y_1 & \rightarrow y_2 & \rightarrow y_3 & \rightarrow y_t \\
x_2 & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
x_3 & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
x_t & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\end{align*}
\]
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 $\mathbf{x}$ by applying a \textbf{recurrence formula} at every time step:

\[
y_t = f_{W_{hy}}(h_t)
\]

output

another function with parameters $W_o$

new state
Recurrent Neural Network

\[ x_1 \rightarrow \text{RNN} \rightarrow y_1 \]

\[ x_2 \rightarrow \text{RNN} \rightarrow y_2 \]

\[ x_3 \rightarrow \text{RNN} \rightarrow y_3 \]

\[ \vdots \]

\[ x_t \rightarrow \text{RNN} \rightarrow y_t \]
Recurrent Neural Network

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

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

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

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

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

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

$$y_t = W_{hy}h_t$$

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

$h_0 \xrightarrow{f_W} h_1$

$x_1$
RNN: Computational Graph

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

\[ x_1 \xleftarrow{w} h_0 \xrightarrow{f_W} h_1 \]

\[ x_2 \xleftarrow{w} h_1 \xrightarrow{f_W} h_2 \]
RNN: Computational Graph
RNN: Computational Graph

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

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

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

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

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

\[ x_1 \xleftarrow{W} h_0, \quad x_2 \xleftarrow{W} h_1, \quad x_3 \xleftarrow{W} h_2 \]
RNN: Computational Graph: Many to One
RNN: Computational Graph: One to Many

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

\[ y_1 \xrightarrow{\text{output}} y_2 \xrightarrow{\text{output}} y_3 \xrightarrow{\text{output}} y_T \]

\[ W \]

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

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

\[ y_1 \xrightarrow{?} y_2 \xrightarrow{?} y_3 \xrightarrow{\cdots} y_T \]

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

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

\[ y_1 \xrightarrow{x} y_2 \xrightarrow{0} y_3 \xrightarrow{0} \]

\[ x \xrightarrow{W} \]

\[ 0 \]
RNN: Computational Graph: One to Many

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

\[ y_1 \xrightarrow{W} x \xrightarrow{f_W} y_1 \xrightarrow{f_W} y_2 \xrightarrow{f_W} y_{T-1} \]
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
Example: Character-level Language Model Sampling

\[
\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}
\]

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. We often put a separate embedding layer between input and hidden layers.
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
Fei-Fei Li, Yunzhu Li, Ruohan Gao

Lecture 8 -  47
April 27, 2023

min-char-rnn.py gist: 112 lines of Python

---

```python
min-char-rnn.py gist: 112 lines of Python
```

---

Fei-Fei Li, Yunzhu Li, Ruohan Gao

Lecture 8 -  47
April 27, 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 art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud huriest thy content,
And tender charl 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-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 thitey" fomescerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogenncc Phe lism thond hon at. MeiDimorotion in ther thize."

Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.

"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.
The Stacks Project: open source algebraic geometry textbook

http://stacks.math.columbia.edu/
The stacks project is licensed under the GNU Free Documentation License
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 $\text{Proj}_X(A) =$ 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 $Q \to C_{S/X}$ 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 $Y = \text{Spec}(R)$.

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

The following lemma surjective restores decomposes of this implies that $\mathcal{F}_\alpha = \mathcal{F}_\alpha = \mathcal{F}_\alpha = \mathcal{F}_\alpha$. Theorem $2.0$. Let $X$ be a locally Noetherian scheme over $S, E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = J_0 \subset \mathcal{A}_X$. Since $\mathcal{I} \subset \mathcal{I}$ are nonzero over $i_0 \geq 0$ is a subset of $J_0, \mathcal{A}_X$ works.

Lemma 0.3. In Situation ??, Hence we may assume $q' \leq \delta'$.

Proof. We will use the property we see that $p$ is the next functor (??). On the other hand, by Lemma ?? we see that $D(\mathcal{O}_X) = \mathcal{O}_X(D)$

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

Proof. See discussion of sheaves of sets.
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

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_{\mathcal{X}}(\mathcal{L})$$

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

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

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 $U \subset X$ 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 $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

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

Consider a common structure on $X$ and $X$ the functor $\mathcal{O}_X(U)$ which is locally of finite type.

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{ccc}
S & \rightarrow & Y' \\
\downarrow & & \downarrow \\
\xi & \rightarrow & \mathcal{O}_X \\
\downarrow & & \downarrow \\
\gamma & \rightarrow & \alpha
\end{array}$$

is a limit. Then $\mathcal{G}$ is a finite type and assume $S$ is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $\mathcal{F}$. This is of finite type diagrams, then

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{\mathcal{X}}$ is a sheaf of rings.

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

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A reduced above we conclude that $U$ is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a

$$\begin{array}{ccc}
\mathcal{O}_{X,x} & \rightarrow & \mathcal{F} \\
\downarrow & & \downarrow \\
\mathcal{O}_{X,x} & \rightarrow & \mathcal{O}_{X,x}(\mathcal{F})
\end{array}$$

is an isomorphism of covering of $\mathcal{O}_X$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that $X$ is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ areopens of finite type over $S$. If $\mathcal{F}$ is a scheme theoretic image points.

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{\mathcal{X}}$ is a closed immersion, see Lemma ??, This is a sequence of $\mathcal{F}$ 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 & 0x00000000ffffffff) & 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/ckevent.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/seteew.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;
    }
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

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

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

Cell that turns on inside comments and quotes:

```c
/* Duplicate LSM field information. The lsm_rule is opaque, so */
static inline int audit_duplicate LSM_field (struct audit_field *df, /*
    struct audit_field */
{
    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);
        */
    } else
    if (ret == -EINVAL)
        pr_warn("audit rule for LSM \\
            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

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

code depth 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
RNN tradeoffs

RNN Advantages:
- Can process any length input
- Computation for step $t$ can (in theory) use information from many steps back
- Model size doesn’t increase for longer input
- Same weights 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

before:
\[ h = \tanh(W_{xh} \ast x + W_{hh} \ast h) \]

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

sample!
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

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 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)

Visual Dialog: Conversations about images

Das et al, "Visual Dialog", CVPR 2017
Figures from Das et al, copyright IEEE 2017. Reproduced with permission.
Visual Language Navigation: Go to the living room

Agent encodes instructions in language and uses an RNN to generate a series of movements as the visual input changes after each move.

**Instruction**

Turn right and head towards the *kitchen*. Then turn left, pass a *table* and enter the *hallway*. Walk down the hallway and turn into the *entry way* to your right *without doors*. Stop in front of the *toilet*.

![Diagram of a room with navigation paths and instructions]

Figures from Wang et al, copyright IEEE 2017. Reproduced with permission.
Visual Question Answering: Dataset Bias

What is the dog playing with?

Frisbee


Fei-Fei Li, Yunzhu Li, Ruohan Gao
Multilayer RNNs
Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

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

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

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

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

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

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

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

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

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 $W_{hh}^T$)

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

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

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

Fei-Fei Li, Yunzhu Li, Ruohan Gao

Lecture 8 - 91 April 27, 2023

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:

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

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

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:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \\
\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}
\]

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

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

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

$L = \frac{\partial L_T}{\partial W} \frac{\partial h_1}{\partial W}$

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

$\text{grad} *= \text{np.sum(} \text{grad} * \text{grad})$ if $\text{grad norm} >$ threshold:
$\text{grad} *= (\text{threshold} / \text{grad norm})$

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

→ Change RNN architecture

References:
- 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

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

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)

Vanilla RNN

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

LSTM

Four gates

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

Cell state

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

Hidden state

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

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

![Diagram of LSTM]

- **Input Vector (x)**
- **Hidden State from Before (h)**
- **Output Vector (i)**
- **Cell State (f)**
- **Hidden Vector in Output (o)**
- **Gate Vector (g)**

Mathematical Representation:
- $4h \times 2h$
- $4h$
- $4h$
- $4h$

**Formulas:**
- $i = \sigma(W_{ix}x + W_{ih}h + b_i)$
- $f = \sigma(W_{fx}x + W_{fh}h + b_f)$
- $g = \tanh(W_{gx}x + W_{gh}h + b_g)$
- $o = \sigma(W_{ox}x + W_{oh}h + b_o)$
- $h' = f \odot h + i \odot g$
- $h = o \odot \tanh(h')$

**Equations:**
- $i$: Input gate
- $f$: Forget gate
- $o$: Output gate
- $g$: Output of the cell state
- $h$: Hidden state
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

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

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

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

\[ i: \text{Input gate}, \text{whether to write to cell} \]

\[ g: \text{Gate gate (?)}, \text{How much to write to cell} \]

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

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

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

\[ x \]

vector from below \((x)\)

\[ h \]

vector from before \((h)\)

\[ W \]

\[ 4h \times 2h \]

\[ i \]: 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

\[ x_t \]

\[ h_{t-1} \]

\[ (i) \]

\[ (f) \]

\[ (o) \]

\[ (g) \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \tanh \]

\[ h_t \]

\[ c_t \]

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

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

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

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

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

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

[Hochreiter et al., 1997]

\[
\begin{align*}
&c_{t-1} \quad + \quad c_t \\
&h_{t-1} \quad \text{stack} \quad x_t \\
&W \\
&i \quad f \quad g \\
&\circ \quad \circ \quad \circ \\
&\text{tanh} \\
&\circ \\
&\circ \\
&h_t
\end{align*}
\]

\[
\begin{align*}
(i_f_g) &= \begin{pmatrix}
\sigma & \sigma & \text{tanh}
\end{pmatrix} W \begin{pmatrix} h_{t-1} \\
x_t \end{pmatrix} \\
c_t &= f \circ c_{t-1} + i \circ g \\
h_t &= o \circ \text{tanh}(c_t)
\end{align*}
\]
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 $W$

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

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

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

Uninterrupted gradient flow!
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!**

\[ \begin{align*}
C_0 & \rightarrow \left\{ \begin{array}{c}
W \\
\text{stack}
\end{array} \right\} \\
& \quad \rightarrow \left\{ \begin{array}{c}
f \\
i \\
g \\
tanh
\end{array} \right\} \\
& \quad \rightarrow \left\{ \begin{array}{c}
+ \\
h_t
\end{array} \right\} \\
C_1 & \rightarrow \left\{ \begin{array}{c}
W \\
\text{stack}
\end{array} \right\} \\
& \quad \rightarrow \left\{ \begin{array}{c}
f \\
i \\
g \\
tanh
\end{array} \right\} \\
& \quad \rightarrow \left\{ \begin{array}{c}
+ \\
h_t
\end{array} \right\} \\
C_2 & \rightarrow \left\{ \begin{array}{c}
W \\
\text{stack}
\end{array} \right\} \\
& \quad \rightarrow \left\{ \begin{array}{c}
f \\
i \\
g \\
tanh
\end{array} \right\} \\
& \quad \rightarrow \left\{ \begin{array}{c}
+ \\
h_t
\end{array} \right\} \\
C_3 & \rightarrow \left\{ \begin{array}{c}
W \\
\text{stack}
\end{array} \right\} \\
& \quad \rightarrow \left\{ \begin{array}{c}
f \\
i \\
g \\
tanh
\end{array} \right\} \\
& \quad \rightarrow \left\{ \begin{array}{c}
+ \\
h_t
\end{array} \right\}
\end{align*} \]

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]

\[
\begin{align*}
  M_{U1}: & \quad z &= \text{sigmoid}(W_{xz}x_t + b_z) \\
  & \quad r &= \text{sigmoid}(W_{xr}x_t + W_{hr}h_t + b_t) \\
  & \quad h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + \tanh(x_t) + b_h) \odot z \\
  & \quad + \quad h_t \odot (1 - z) \\
  M_{U2}: & \quad z &= \text{sigmoid}(W_{xz}x_t + W_{hz}h_t + b_z) \\
  & \quad r &= \text{sigmoid}(x_t + W_{hr}h_t + b_t) \\
  & \quad h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{zh}x_t + b_h) \odot z \\
  & \quad + \quad h_t \odot (1 - z) \\
  M_{U3}: & \quad z &= \text{sigmoid}(W_{xz}x_t + W_{hz} \tanh(h_t) + b_z) \\
  & \quad r &= \text{sigmoid}(W_{xr}x_t + W_{hr}h_t + b_t) \\
  & \quad h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{zh}x_t + b_h) \odot z \\
  & \quad + \quad h_t \odot (1 - z)
\end{align*}
\]
Neural Architecture Search for RNN architectures

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