Lecture 10: Recurrent Neural Networks
Last Time: CNN Architectures

AlexNet

Revolution of Depth

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

- GoogLeNet
- VGG16
- VGG19

Revolution of Depth

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

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 10 - 6 May 4, 2017

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

e.g. 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**

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:

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

new state \quad old state \quad input vector at some time step

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.
(Vanilla) Recurrent Neural Network

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

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

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

$$ y_t = W_{hy} h_t $$
RNN: Computational Graph

\[ h_0 \xrightarrow{f_W} h_1 \]

\[ x_1 \]
RNN: Computational Graph

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

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

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

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

\[ x_1 \]

\[ x_2 \]

\[ x_3 \]
RNN: Computational Graph

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

\[ h_0, h_1, h_2, h_3, \ldots, h_T \]

\[ f_W \]

\[ x_1, x_2, x_3 \]

\[ W \]

\[ y_1, y_2, y_3, \ldots, y_T \]

\[ L_1, L_2, L_3, \ldots, L_T \]
RNN: Computational Graph: Many to Many

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

\[
x_1 \xleftarrow{W} \quad x_2 \xleftarrow{W} \quad x_3
\]

\[
y_1 \xrightarrow{L_1} \quad y_2 \xrightarrow{L_2} \quad y_3 \xrightarrow{L_3} \quad y_T \xrightarrow{L_T}
\]

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

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

\[ W \]

\[ x_1 \xrightarrow{} h_0 \]

\[ x_2 \xrightarrow{} h_1 \]

\[ x_3 \xrightarrow{} h_2 \]

\[ y \xrightarrow{} h_T \]
RNN: Computational Graph: One to Many

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

\[ y_3 \]

\[ W \rightarrow \text{x} \]

\[ y_T \]
Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

\[ \begin{align*}
&h_0 \\
&f_W \quad h_1 \\
&f_W \quad h_2 \\
&f_W \quad h_3 \\
&\ldots \\
&W \\
&x_1 \\
&x_2 \\
&x_3 \\
&h_T
\end{align*} \]
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
Example:
Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

```
input layer
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
```

input chars: “h” “e” “l” “o”
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

Softmax

Sample

“e”

.03
.13
.00
.84

“l”

.25
.20
.05
.50

“l”

.11
.17
.68
.03

“o”

.11
.02
.08
.79

output layer

1.0
2.2
-3.0
4.1

hidden layer

0.3
-0.1
0.9

0.3
0.3
0.1

0.1
0.5
1.9
-1.1

W_{hy}

W_{hh}

W_{xh}

input layer

1
0
0

0
1
0

0
1
0

0
0
0

W_{hy}

W_{hh}

W_{xh}

input chars: “h” “e” “l” “l” “o”
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

Loss

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

```python
...  # Forward pass
  for t in range(max_seq_len):
    h[t] = np.tanh(np.dot(x[t], [0.1, 0.1, 0.1]) + np.dot(h[t-1], [0.1, 0.1, 0.1]))  # hidden state
    y[t] = softmax(np.dot(h[t], [0.1, 0.1, 0.1]))  # probability for next char

  loss = np.mean(-np.log(y[t] - y[t-1]))  # cross-entropy loss

  # Backward pass: compute gradients going backwards
  dy = -np.log(y[t])  # gradients through softmax
  h[t] = np.tanh(np.dot(h[t], [0.1, 0.1, 0.1]))  # gradients through tanh
  x[t] -= np.dot(dy, [0.1, 0.1, 0.1])  # gradients through input

  batch += y[t]  # accumulate loss
```

Fei-Fei Li & Justin Johnson & Serena Yeung
Lecture 10 - 45 May 4, 2017
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 budliest thy content,
And tender charl mak'st waste iniggarding:
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 deserve 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
cOaniogennnc 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 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.
The Stacks Project: open source algebraic geometry textbook

<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>

The Stacks project now consists of:
- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

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 $\mathcal{L}_{m*} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on $X$, $U$ is a closed immersion of $S$, then $U \to \mathcal{T}$ is a separated algebraic space.

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

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

and the comparably in the fibre product covering we have to prove the lemma generated by $\prod Z \times U \to V$. Consider the maps $\mathcal{M}$ along the set of points $\text{Sch}_{\text{ppf}}$ and $U \to \mathcal{U}$ 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 $\mathcal{U}$ such that $\text{Spec}(R') \to S$ is smooth or an

$$ U = \bigcup U_1 \times S_1 U_1 $$

which has a nonzero morphism we may assume that $f_1$ is of finite presentation over $S$. We claim that $O_{X_S}$ is a scheme where $x, x', x'' \in S'$ such that $O_{X_S} \to O_{X', x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $GL_{S'}(X'/S')$ and we win. \hfill $\square$

To prove study we see that $\mathcal{F}_i$ is a covering of $X'$, and $T_i$ is an object of $\mathcal{F}_i$. Let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $C$ as a $J$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$ M^* \cong \Gamma^*(U, \mathcal{O}_X) $$

is a unique morphism of algebraic stacks. Note that

$$ \text{Arrows} = (\text{Sch}/S)^{\text{ppf}} \times (\text{Sch}/S)^{\text{ppf}} $$

and

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

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. \hfill $\square$

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

---

**Lemma 0.1.** Assume (3) and (5) 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

$$ \mathcal{O}_X = \Gamma(X, \mathcal{O}_X, \mathcal{O}_X). $$

When in this case of to show that $T \to \mathcal{C}_{X/S}$ 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 cutout). If $T$ is surjective we may assume that $T$ is connected with residue fields of $S$. Moreover there exists a closed subspace $Z \subset X$ of $X$ where $U$ in $X'$ is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

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

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

The following lemma surjective rest decomposes of this implies that $\mathcal{F}_{x_i} = \mathcal{F}_{x_i} = \mathcal{F}_{x_i}_{x,0}$.

**Lemma 0.2.** Let $X$ be a locally Noetherian scheme over $S$, $E = \mathcal{F}_{X/S}$. Set $T = \mathcal{J}_1 \subset T$. Since $E_n \subset \mathcal{T}$ are nonzero over $i_0 \leq p$ is a subset of $\mathcal{J}_{n,0} \hat{A}_2$ 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

$$ D(\mathcal{O}_X) = \mathcal{O}_X(D) $$

where $K$ is an $F$-algebra where $\delta_{n+1}$ is a scheme over $S$. \hfill $\square$
**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 $F \to F$ 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 $U \subset X$ 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

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

This since $F \in F$ and $x \in G$ the diagram

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

is a limit. Then $G$ is a finite type and assume $S$ is a flat and $F$ and $G$ is a finite type $f$. This is of finite type diagrams, and

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

\[
\begin{array}{ccc}
O_{X_{\text{et}}} & \to & F_F \\
\downarrow & & \downarrow \\
O_{X_{\text{et}}} & \to & \mathcal{O}_{X_{\text{et}}}(O_{X_{\text{et}}})
\end{array}
\]

is an isomorphism of covering of $O_{X_{\text{et}}}$. 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 $O_{X_{\text{et}}}$ is a closed immersion, see Lemma ?? This is a sequence of $F$ is a similar morphism.
```c
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 & 0x00000000ffffffff) & 0x0000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
/* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
/* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
/*
  * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.
  *
  * This program is free software; you can redistribute it and/or modify it
  * under the terms of the GNU General Public License version 2 as published by
  * the Free Software Foundation.
  *
  * This program is distributed in the hope that it will be useful,
  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  * GNU General Public License for more details.
  *
  * You should have received a copy of the GNU General Public License
  * along with this program; if not, write to the Free Software Foundation,
  * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
  */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/cdev.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/setev.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)
{
    #ifndef 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;
    
}
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
This image is CC0 public domain
before:
\[ h = \tanh(W_{xh} \ast x + W_{hh} \ast h) \]

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

dsampling image processing network:

- Convolutional layers (conv-64, conv-128, conv-256, conv-512)
- Max pooling layers
- Fully connected layers (FC-4096)

Sample output:
- y0
- y1
- h0
- h1
- x0
- 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
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
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: H x W x 3

Features: L x D

h0

Image Captioning with Attention

Image: $H \times W \times 3$

Features: $L \times D$

Distribution over $L$ locations

Image Captioning with Attention


\[ z = \sum_{i=1}^{L} p_i v_i \]
Image Captioning with Attention

**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

Image: \( H \times W \times 3 \)

Features: \( L \times D \)

Weighted combination of features

Weighted features: \( D \)

First word

Distribution over \( L \) locations

Distribution over vocab

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 Salakhutdinov, 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

Q: What endangered animal is featured on the truck?
A: A bald eagle.
A: A sparrow.
A: A humming bird.
A: A raven.

Q: Where will the driver go if turning right?
A: Onto 24 ¾ Rd.
A: Onto 25 ¾ Rd.
A: Onto 23 ¾ Rd.
A: Onto Main Street.

Q: When was the picture taken?
A: During a wedding.
A: During a bar mitzvah.
A: During a funeral.
A: During a Sunday church service.

Q: Who is under the umbrella?
A: Two women.
A: A child.
A: An old man.
A: A husband and a wife.

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

Multilayer RNNs

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

\( h \in \mathbb{R}^n \) \hspace{1cm} \( W^l \in [n \times 2n] \)

LSTM:

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

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

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

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

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

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

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 $W_{hh}^T$)

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

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

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

Largest singular value > 1: Exploding gradients

Largest singular value < 1: Vanishing gradients

Gradient clipping: Scale gradient if its norm is too big

\[
\text{grad\_norm} = \text{np.sum}(\text{grad} * \text{grad})
\]
\[
\text{if } \text{grad\_norm} > \text{threshold}:
\text{grad} *= (\text{threshold} / \text{grad\_norm})
\]

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**

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

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

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]

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

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

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

[Hochreiter et al., 1997]

\[
\begin{align*}
\odot & \quad + \\
C_{t-1} & \quad C_t \\
W & \quad \odot \\
\text{stack} & \quad \odot \\
h_{t-1} & \quad h_t \\
X_t & \quad \odot \\
\end{align*}
\]

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

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

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

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

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

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

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