Deep Learning for Computer Vision: Sequences (RNN, Attention)

Shai Bagon
Agenda

Images
- static
  - Perceptron
  - Convolution
  - CNN
  - ViT

Sequences
- time-dependent
  - Recurrent NN
  - Memory
  - Attention
Deep Learning for Sequences

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

Feed-forward:
- e.g., **image** classification
- e.g., **video** captioning
- e.g., **video frames** classification
- e.g., **Machine translation**

e.g., **classification** image -> sequence of frames -> label

Slide credit: Justin Johnson (EECS-498-007, UMich)
Recurrent Neural Networks

One to one  Many to many
Recurrent Neural Networks

One to one    Many to many

(+ ) Parameter efficient
(- ) No temporal dependency

class ManyToManyV0(nn.Module):
    def __init__(self, number_of_time_steps):
        super(ManyToMany, self).__init__()
        # SAME instance of SingleTimeStep for each time step
        self.time_steps = SingleTimeStep(...)

    def forward(self, in_seq):
        out_pred = []
        for t, x_t in enumerate(in_seq):
            p_t = self.time_steps(x_t)
            out_pred.append(p_t)
        return out_pred
Recurrent Neural Networks

One to one  Many to many

(+?) Temporal dependency (via trained parameters)
(-) Parameter inefficiency
(-) Fixed sequence length
Recurrent Neural Networks

One to one    Many to many

(+) Temporal dependency (via “hidden state”)
(+) Parameter efficiency
(+) Arbitrary sequence length
Recurrent Neural Networks

One to one  Many to many

\[ h_t = f(h_{t-1}, x_t; W) \]
Example: Language Modeling

Task:
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

Training sequence: “hello”
Vocabulary: ['h', 'e', 'l', 'o']

Embedding Layer:

Input sequence:

<table>
<thead>
<tr>
<th>Embedding</th>
<th>'h'</th>
<th>'e'</th>
<th>'l'</th>
<th>'o'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Slide credit: Justin Johnson (EECS-498-007, UMich)
Example: Language Modeling

Task:
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

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

Training sequence: “hello”
Vocabulary: ['h', 'e', 'l', 'o']
**Example: Language Modeling**

**Task:**
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

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

Training sequence: “hello”
Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]

**Output Layer:**

<table>
<thead>
<tr>
<th>Target chars</th>
<th>‘e’</th>
<th>‘l’</th>
<th>‘l’</th>
<th>‘o’</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_{hy}$</td>
<td>1.0</td>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>0.3</td>
<td>0.5</td>
<td>-2.0</td>
</tr>
<tr>
<td></td>
<td>-3.0</td>
<td>-1.0</td>
<td>0.5</td>
<td>-3.0</td>
</tr>
<tr>
<td></td>
<td>4.1</td>
<td>1.2</td>
<td>-1.0</td>
<td>2.2</td>
</tr>
</tbody>
</table>

**Hidden Layer:**

| $W_{hh}$     | 0.3 | 1.0 | 0.1 | -0.3|
|              | -1.0| 0.3 | -5.0| 0.9 |
|              | 0.9 | 0.1 | -3.0| 0.7 |

**Embedding Layer:**

<table>
<thead>
<tr>
<th>Input sequence</th>
<th>‘h’</th>
<th>‘e’</th>
<th>‘l’</th>
<th>‘l’</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_{xh}$</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

*Slide credit: Justin Johnson (EECS-498-007, UMich)*
Example: Language Modeling

Task:
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

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

\[ y_t = W_{hy}h_t \]

Training sequence: “hello”
Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]

Given “h” predict “e”

Slide credit: Justin Johnson (EECS-498-007, UMich)
**Example: Language Modeling**

**Task:**
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

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

$y_t = W_{hy}h_t$

Training sequence: “hello”
Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]

---

**Given “he” predict “l”**

<table>
<thead>
<tr>
<th>Target chars:</th>
<th>‘e’</th>
<th>‘l’</th>
<th>‘l’</th>
<th>‘o’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Layer:</td>
<td>1.0</td>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>0.3</td>
<td>0.5</td>
<td>-2.0</td>
</tr>
<tr>
<td></td>
<td>-3.0</td>
<td>-1.0</td>
<td>-1.9</td>
<td>-3.0</td>
</tr>
<tr>
<td></td>
<td>4.1</td>
<td>1.2</td>
<td>1.9</td>
<td>2.2</td>
</tr>
</tbody>
</table>

| Hidden Layer: | 0.3 | 1.0 | 0.1 | -3.0 |
|               | -1.0 | 0.3 | -0.5 | 0.9 |
|               | 0.9 | 0.1 | -3.0 | 0.7 |

<table>
<thead>
<tr>
<th>Input sequence:</th>
<th>‘h’</th>
<th>‘e’</th>
<th>‘l’</th>
<th>‘l’</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_{xh}$</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

---

Slide credit: Justin Johnson (EECS-498-007, UMich)
Example: Language Modeling

Task:
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

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

$y_t = W_{hy}h_t$

Training sequence: “hello”
Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]

Given “hel” predict “l”

Output Layer:

Target chars:

<table>
<thead>
<tr>
<th><code>e</code></th>
<th><code>l</code></th>
<th><code>l</code></th>
<th><code>o</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>2.2</td>
<td>0.3</td>
<td>0.5</td>
<td>0.2</td>
</tr>
<tr>
<td>4.1</td>
<td>1.2</td>
<td>1.9</td>
<td>-3.2</td>
</tr>
<tr>
<td>-3.1</td>
<td>-1.1</td>
<td>-1.1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Hidden Layer:

<table>
<thead>
<tr>
<th><code>h</code></th>
<th><code>e</code></th>
<th><code>l</code></th>
<th><code>l</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>1.0</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>0.9</td>
<td>0.3</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>-0.1</td>
<td>1.2</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>4.1</td>
<td>-3.3</td>
<td>1.9</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Input sequence: ‘h’ ‘e’ ‘l’ ‘l’ ‘l’

Slide credit: Justin Johnson (EECS-498-007, UMich)
Example: Language Modeling

Task:
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

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

$$y_t = W_{hy}h_t$$

Training sequence: “hello”
Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]
Example: Language Modeling

Task:
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

At test time: **generate** new text
Sample one char at a time

Training sequence: “hello”
Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]
Example: Language Modeling

Task:
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

At test time: **generate** new text
Sample one char at a time

Training sequence: “hello”
Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]
Example: Language Modeling

Task:
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

At test time: **generate** new text
Sample one char at a time

Training sequence: “hello”
Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]

Sample one char at a time
Example: Language Modeling

Task:
Given characters $c_0, c_1, \ldots, c_{t-1}$
Predict $c_t$

At test time: **generate** new text
Sample one char at a time

Training sequence: “hello”
Vocabulary: ['h', 'e', 'l', 'o']

Slide credit: Justin Johnson (EECS-498-007, UMich)
Backpropagation Through Time

Forward through entire sequence
compute loss
Backward through entire sequence

Loss

Slide credit: Justin Johnson (EECS-498-007, UMich)
Backpropagation Through Time

Forward through entire sequence
compute loss
Backward through entire sequence

Takes a LOT of memory for long sequences
Backpropagation Through Time

Forward through temporal chunks
compute loss
Backward through chunk
Backpropagation Through Time

Forward through temporal chunks
compute loss
Backward through chunk
Carry hidden states forward

Slide credit: Justin Johnson (EECS-498-007, UMich)
Backpropagation Through Time

Loss
Agenda

Images
- static
  - Perceptron
  - Convolution
  - CNN

Sequences
- time-dependent
  - Recurrent NN
  - Memory
  - Attention
Sequence to Sequence

We Learn Computer Vision

(ANU) (LOMDIM) (MEMUCHSHEVET) (RE’EYA)
Sequence to Sequence: RNN

Encoder RNN

Decoder RNN

Initial hidden state

Slide credit: Justin Johnson (EECS-498-007, UMich)
Sequence to Sequence: RNN

Encoder RNN

Decoder RNN

"context" is a bottleneck

What if seq is very long?

Slide credit: Justin Johnson (EECS-498-007, UMich)
Sequence to Sequence: RNN & Attention

No need to provide: Learned end-to-end
Sequence to Sequence: RNN & Attention

$V$

$K$

$Q$

softmax

Slide credit: Justin Johnson (EECS-498-007, UMich)
Sequence to Sequence: RNN & Attention

Slide credit: Justin Johnson (EECS-498-007, UMich)
Sequence to Sequence: RNN & Attention

\[ K \]
\[ V \]
\[ Q \]

Slide credit: Justin Johnson (EECS-498-007, UMich)
Example: English to French translation

Input (English):
The agreement on the European Economic Area was signed in August 1992.

Output (French):
Example: English to French translation

Input (English):  
The agreement on the European Economic Area was signed in August 1992.

Output (French):  

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Slide credit: Justin Johnson (EECS-498-007, UMich)
Example: English to French translation

Input (English):
The agreement on the European Economic Area was signed in August 1992.

Output (French):

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Slide credit: Justin Johnson (EECS-498-007, UMich)
Example: English to French translation

Input (English):
The agreement on the European Economic Area was signed in August 1992.

Output (French):
Attention Layer

Slide credit: Justin Johnson (EECS-498-007, UMich)
Attention Layer

Inputs:
Query: \( Q \) (shape: \( N_q \times D_q \))
Input: \( X \) (shape: \( N_x \times D_x \))

Layer’s Parameters:
\( X \rightarrow K: W_k \) (shape: \( D_x \times D_q \))
\( X \rightarrow V: W_v \) (shape: \( D_x \times D_v \))

Compute:
Keys: \( K = XW_k \) (shape: \( N_x \times D_q \))
Values: \( V = XW_v \) (shape: \( N_x \times D_v \))
Similarities: \( E = QK^T/\sqrt{D_q} \) (shape: \( N_q \times N_x \))
Attention: \( A = \text{softmax}(E; \uparrow) \) (shape: \( N_q \times N_x \))
Outputs: \( Y = AV \) (shape: \( N_q \times D_v \))

Slide credit: Justin Johnson (EECS-498-007, UMich)
Self-Attention Layer

Input:

Input: $X$ (shape: $N_x \times D_x$)

Layer’s Parameters:

- $X \rightarrow Q$: $W_q$ (shape: $D_x \times D_q$)
- $X \rightarrow K$: $W_k$ (shape: $D_x \times D_q$)
- $X \rightarrow V$: $W_v$ (shape: $D_x \times D_v$)

Compute:

Query: $Q = X W_q$ (shape: $N_x \times D_q$)

Keys: $K = X W_k$ (shape: $N_x \times D_q$)

Values: $V = X W_v$ (shape: $N_x \times D_v$)

Similarities: $E = Q K^T / \sqrt{D_q}$ (shape: $N_q \times N_x$)

Attention: $A = \text{softmax}(E; \uparrow)$ (shape: $N_q \times N_x$)

Outputs: $Y = A V$ (shape: $N_q \times D_v$)
Self Attention in Vision

\[
\begin{align*}
x : & \quad D_x \times T \times H \times W \\
y : & \quad D_y \times T \times H \times W \\
q : & \quad D_q \times T \times H \times W \\
k : & \quad D_q \times T \times H \times W \\
v : & \quad D_y \times T \times H \times W \\
W_q : & \quad 1 \times 1 \times 1 \\
W_k : & \quad 1 \times 1 \times 1 \\
W_v : & \quad 1 \times 1 \times 1 \\
x : & \quad D_x \times T \times H \times W
\end{align*}
\]

Self Attention in Vision

Self-Attention Layer: Properties

\[ \text{SelfAtt}(\pi(x_1, \ldots, x_n)) = \pi(\text{SelfAtt}(x_1, \ldots, x_n)) \]

Self-Attention is permutation equivariant

“I am studying” ? “Am I studying”
Self-Attention Layer: Properties

Positional Encoding
Multi-head Self-Attention

Slide credit: Justin Johnson (EECS-498-007, UMich)
Three Ways of Processing Sequences

**Recurrent Neural Network**
- Works on **Ordered Sequences**
  - (+) large and adaptive receptive field via hidden state
  - (-) Not parallelizable: need to process states sequentially
- Works on **Multidimensional Grids**
  - (+) Highly parallelizable
  - (-) Fixed receptive field. Need to stack many layers to have a decent one

**1D Convolution**
- Works on **Ordered Sequences**
  - (+) large and adaptive receptive field via hidden state
  - (-) Not parallelizable: need to process states sequentially
- Works on **Multidimensional Grids**
  - (+) Highly parallelizable
  - (-) Fixed receptive field. Need to stack many layers to have a decent one

**Self-Attention**
- Works on **Sets**
  - (+) receptive field = entire sequence
  - (+) parallelizable
  - (-) Very memory intensive

---

Slide credit: **Justin Johnson (EECS-498-007, UMich)**
Three Ways of Processing Sequences

- **Recurrent Neural Network**: Works on ordered sequences. (+) large and adaptive receptive field via hidden state; (-) not parallelizable: need to process states sequentially.

- **1D Convolution**: Works on multidimensional grids. (-) fixed receptive field. Need to stack many layers to have a decent one; (+) highly parallelizable.

- **Self-Attention**: Works on sets. (+) receptive filed = entire sequence; (+) parallelizable; (-) very memory intensive.

---

**Attention Is All You Need**

---

**Authors**:
- Ashish Vaswani*
  - Google Brain
  - avaswani@google.com

- Noam Shazeer*
  - Google Brain
  - noam@google.com

- Niki Parmar*
  - Google Research
  - niki@google.com

- Jakob Uszkoreit*
  - Google Research
  - usz@google.com

- Llion Jones*
  - Google Research
  - llion@google.com

- Aidan N. Gomez* †
  - University of Toronto
  - aidan@cs.toronto.edu

- Łukasz Kaiser*
  - Google Brain
  - lukasz.kaiser@google.com

- Illia Polosukhin* ‡
  - illia.polosukhin@gmail.com

---

Slide credit: Justin Johnson (EECS-498-007, UMich)
Transformer Layer

Input: $x_1, \ldots, x_n$ (n tokens in $D$ dimensions)

Layer Norm:
$\gamma, \beta$: scale and shift parameters ($D$ dimensions)

Compute:
\[
\mu_i = \frac{\sum_j x_{ij}}{D} \text{ (n scalars)}
\]
\[
\sigma_i = \sqrt{\frac{\sum_j (x_{ij} - \mu_i)}{D}} \text{ (n scalars)}
\]
output: $o_i = \gamma \cdot \left(\frac{x_i - \mu_i}{\sigma_i}\right) + \beta$
Transformer Layer

**Input:** $x_1, ..., x_n$ ($n$ tokens in $D$ dimensions)

**Output:** $y_1, ..., y_n$ ($n$ tokens in $D$ dimensions)

Highly scalable

Highly parallelizable

Slide credit: Justin Johnson (EECS-498-007, UMich)
Transformers Network

**Pretraining:**
Download a LOT of text from the internet
Train a transformers network using self-supervision

**Finetuning:**
Fine-tune the transformer to specific NLP task at hand

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width ($D$)</th>
<th>#Heads</th>
<th>#Params</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>110M</td>
<td>13GB</td>
</tr>
<tr>
<td>BERT-large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>340M</td>
<td>13GB</td>
</tr>
<tr>
<td>GPT-2</td>
<td>48</td>
<td>1600</td>
<td>?</td>
<td>1.5B</td>
<td>40GB</td>
</tr>
<tr>
<td>GPT-3</td>
<td>96</td>
<td>12288</td>
<td>96</td>
<td>175B</td>
<td>694GB</td>
</tr>
</tbody>
</table>
Agenda

Images
- Perceptron
- Convolution
- CNN
- ViT

Sequences
- Recurrent NN
- Memory
- Attention

static

time-dependent
Vision Transformers (ViT)

Vision Transformers (ViT)

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width ($D$)</th>
<th>#Heads</th>
<th>#Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViT-Base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>86M</td>
</tr>
<tr>
<td>ViT-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>307M</td>
</tr>
<tr>
<td>ViT-Huge</td>
<td>32</td>
<td>1280</td>
<td>16</td>
<td>632M</td>
</tr>
</tbody>
</table>

Vision Transformers (ViT)

Vision Transformers (ViT)

Vision Transformers (ViT)
