Tensor-Train Recurrent Neural Networks for Video Classification by Yang et al.

IFT6760A Class Presentation

Mohamed Abdelsalam and Charles C Onu
12 March, 2019

20118612 and 260663256
• Recurrent Neural Networks (RNN) are very successful for sequence modeling.

• But are difficult to train for very high dimensional inputs due to large input-hidden weight matrix.

• E.g. Input video frame of size 160x120x3 (57,600 features) would require **5,760,000** weights for a hidden layer of 100 neurons.
Possible Approaches

- CNN-RNN model: CNN extracts compact representation and RNN learns temporal information.
  - Impractical for large video datasets.

---

1. Donahue et al. (2015), and Srivastava et al. (2015)
2. Donahue et al. (2015), Ng et al. (2015), and Sharma et al.
Possible Approaches

• CNN-RNN model: CNN extracts compact representation and RNN learns temporal information.
  • Impractical for large video datasets.
• Focus on the CNN, and constrain the sequence length of the RNN\(^1\):
  • Cannot scale to long videos.

---

\(^1\) Donahue et al. (2015), and Srivastava et al. (2015)  
\(^2\) Donahue et al. (2015), Ng et al. (2015), and Sharma et al.
Possible Approaches

- **CNN-RNN model**: CNN extracts compact representation and RNN learns temporal information.
  - Impractical for large video datasets.
- Focus on the CNN, and constrain the sequence length of the RNN\(^1\):
  - Cannot scale to long videos.
- **Use embeddings from pretrained CNN as input**\(^2\)
  - Not trained end-to-end, suboptimal parameters
  - CNNs are pretrained on image datasets, which can be of totally different nature than video frames

---

\(^1\)Donahue et al. (2015), and Srivastava et al. (2015)

\(^2\)Donahue et al. (2015), Ng et al. (2015), and Sharma et al.
Proposed Solution

- Reduce the number of parameters in input-to-hidden layer by factorizing the Weight matrix using Tensor-Train decomposition
  
  (a) Allows the use of raw pixels as the input to the RNN
  (b) Can be easily trained end-to-end
  (c) Captures the correlation between spatial and temporal patterns, as the input-to-hidden and hidden-to-hidden are trained jointly
1. Oseledets (2011) introduced Tensor-Train Decomposition in 2011
2. Lebedev et al. (2014) used CP factorization in order to compress the convolutional layers in a network, which is then finetuned.
1. Oseledets (2011) introduced Tensor-Train Decomposition in 2011
2. Lebedev et al. (2014) used CP factorization in order to compress the convolutional layers in a network, which is then finetuned.
3. Novikov et al. (2015) used TT Decomposition to compress the Fully Connected layers of a network, which is then trained from scratch.
4. Garipov et al. (2016) extended the work of Novikov et al. (2015) and used TT decomposition to compress Convolutional Layers as well.
5. This work extended the work of Novikov et al. (2015) and used TT decomposition to factorize the input-to-hidden mapping in RNN, so as to succeed in using RNNs with videos.
Some kind of history

2. Lebedev et al. (2014) used CP factorization in order to compress the convolutional layers in a network, which is then finetuned.
3. Novikov et al. (2015) used TT Decomposition to compress the Fully Connected layers of a network, which is then trained from scratch.
4. Garipov et al. (2016) extended the work of Novikov et al. (2015) and used TT decomposition to compress Convolutional Layers as well.
1. Oseledets (2011) introduced Tensor-Train Decomposition in 2011
2. Lebedev et al. (2014) used CP factorization in order to compress the convolutional layers in a network, which is then finetuned.
3. Novikov et al. (2015) used TT Decomposition to compress the Fully Connected layers of a network, which is then trained from scratch.
4. Garipov et al. (2016) extended the work of Novikov et al. (2015) and used TT decomposition to compress Convolutional Layers as well.
5. This work extended the work of Novikov et al. (2015) and used TT decomposition to factorize the input-to-hidden mapping in RNN, so as to succeed in using RNNs with videos.
TT-train factorization

For $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_d}$:

Where $r_1, r_2, \ldots, r_{d-1}$ are the TT-ranks ($r_0$ and $r_d$ are always 1)
Before reshaping and TT, input $x \in \mathbb{R}^M$, output $y \in \mathbb{R}^N$, weights $W \in \mathbb{R}^{M \times N}$, biases $b \in \mathbb{R}^N$:

$$xW + b = y$$
After reshaping and TT, $X \in \mathbb{R}^{m_1 \times m_2 \times \cdots \times m_d}$, $Y \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$,
$W \in \mathbb{R}^{(m_1 \times n_1) \times (m_2 \times n_2) \times \cdots \times (m_d \times n_d)}$, $B \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$
After reshaping and TT, $X \in \mathbb{R}^{m_1 \times m_2 \times \cdots \times m_d}$, $Y \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, $\mathcal{W} \in \mathbb{R}^{(m_1 \times n_1) \times (m_2 \times n_2) \times \cdots \times (m_d \times n_d)}$, $B \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$.
Recurrent Neural Networks

In a simple RNN, the cell $A$ is defined as:

$$h^{[t]} = \tanh(Wx^{[t]} + Uh^{[t-1]} + b)$$
- In practice, $A$ is typically a GRU (or LSTM cell).
- For GRU:

$$
\begin{align*}
    r^t &= \sigma(W^r x^t + U^r h^{t-1} + b^r) \\
    z^t &= \sigma(W^z x^t + U^z h^{t-1} + b^z) \\
    \tilde{h}^t &= \tanh(W^d x^t + U^d (r^t \ast h^{t-1}) + b^d) \\
    h^t &= (1 - z^t) \ast h^{t-1} + z^t \ast \tilde{h}^t
\end{align*}
$$
Tensor-Train GRU (TT-GRU)

- We replace the input-to-hidden layer connection with a Tensor-Train layer (TTL)

\[
\begin{align*}
   r^{[t]} &= \sigma(TTL(W^r, x^{[t]}) + U^r h^{[t-1]} + b^r) \\
   z^{[t]} &= \sigma(TTL(W^z, x^{[t]}) + U^z h^{[t-1]} + b^z) \\
   \tilde{h}^{[t]} &= \tanh(TTL(W^d, x^{[t]}) + U^d (r^{[t]} \ast h^{[t-1]}) + b^d) \\
   h^{[t]} &= (1 - z^{[t]}) \ast h^{[t-1]} + z^{[t]} \ast \tilde{h}^{[t]}
\end{align*}
\]
Tensor-Train GRU (TT-GRU)

- We replace the input-to-hidden layer connection with a Tensor-Train layer (TTL)

\[
\begin{align*}
    r[t] &= \sigma(TTL(W^r, x[t]) + U^r h[t-1] + b^r) \\
    z[t] &= \sigma(TTL(W^z, x[t]) + U^z h[t-1] + b^z) \\
    \tilde{h}[t] &= \tanh(TTL(W^d, x[t]) + U^d (r[t] * h[t-1]) + b^d) \\
    h[t] &= (1 - z[t]) * h[t-1] + z[t] * \tilde{h}[t]
\end{align*}
\]

- Compression rate

\[
r = \frac{\sum_{k=1}^{d} m_k n_k r_{k-1} r_k}{\prod_{k=1}^{d} m_k n_k}
\]
TT-GRU Implementation Trick

- Concatenate the gates as one output tensor.
- TT-GRU factorizes this tensor once, instead of factorizing each gate successively.
- Parallelizes computation and further reduces number of parameters

\[ r^* = \sum_{k=1}^{d} m_k n_k r_{k-1} r_k + (c - 1)(m_1 n_1 r_0 r_1) \]
\[ c \cdot \prod_{k=1}^{d} m_k n_k \]

where \( c \) is the number of TTLs
For a classification task (e.g., video classification):

\[
P(y_i = 1|\{x_i^{[t]}\}_{t=1}^{T_i}) = \phi(h_i^{[T_i]})
\]
Compression with TT-RNNs

- Consider video frames of size $160 \times 120 \times 3 = 57,600$ pixels reshaped as $8 \times 20 \times 20 \times 18$
- Hidden layer of size 256, reshaped as $4 \times 4 \times 4 \times 4$

<table>
<thead>
<tr>
<th>FC</th>
<th>TT-ranks</th>
<th>TTL</th>
<th>vanilla TT-LSTM</th>
<th>TT-LSTM</th>
<th>vanilla TT-GRU</th>
<th>TT-GRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>14,745,600</td>
<td>3</td>
<td>1,752</td>
<td>7,008</td>
<td>2,040</td>
<td>5,256</td>
<td>1,944</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2,976</td>
<td>11,904</td>
<td>3,360</td>
<td>8,928</td>
<td>3,232</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4,520</td>
<td>18,080</td>
<td>5,000</td>
<td>13,560</td>
<td>4,840</td>
</tr>
</tbody>
</table>

**Figure 1:** Number of parameters for the different settings
Experiment 1

UCF11 Dataset

- 1600 video clips
- 11 classes (basketball shooting, biking, diving, etc)
Experiment 1

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th># Parameters</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT-MLP</td>
<td>0.427 ± 0.045</td>
<td>7,680</td>
<td>902s</td>
</tr>
<tr>
<td>GRU</td>
<td>0.488 ± 0.033</td>
<td>44,236,800</td>
<td>7,056s</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.492 ± 0.026</td>
<td>58,982,400</td>
<td>8,892s</td>
</tr>
<tr>
<td>TT-GRU</td>
<td><strong>0.813 ± 0.011</strong></td>
<td>3,232</td>
<td>1,872s</td>
</tr>
<tr>
<td>TT-LSTM</td>
<td>0.796 ± 0.035</td>
<td>3,360</td>
<td>2,160s</td>
</tr>
</tbody>
</table>

**Figure 2:** Experimental results

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original: (Liu et al., 2009)</td>
<td>0.712</td>
</tr>
<tr>
<td>(Liu et al., 2013)</td>
<td>0.761</td>
</tr>
<tr>
<td>(Hasan &amp; Roy-Chowdhury, 2014)</td>
<td>0.690</td>
</tr>
<tr>
<td>(Sharma et al., 2015)</td>
<td><strong>0.850</strong></td>
</tr>
<tr>
<td>Our best model (TT-GRU)</td>
<td>0.813</td>
</tr>
</tbody>
</table>

**Figure 3:** Comparison to state-of-the-art results
Experiment 2

Hollywood2 Dataset

- 1707 video clips from 69 movies
- 12 (non-exclusive) classes. E.g. answering the phone, driving, eating, fighting, etc
**Experiment 2**

<table>
<thead>
<tr>
<th></th>
<th>MAP</th>
<th># Parameters</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT-MLP</td>
<td>0.103</td>
<td>4,352</td>
<td>16s</td>
</tr>
<tr>
<td>GRU</td>
<td>0.249</td>
<td>53,913,600</td>
<td>106s</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.108</td>
<td>71,884,800</td>
<td>179s</td>
</tr>
<tr>
<td>TT-GRU</td>
<td>0.537</td>
<td>2,944</td>
<td>96s</td>
</tr>
<tr>
<td>TT-LSTM</td>
<td><strong>0.546</strong></td>
<td>3,104</td>
<td>102s</td>
</tr>
</tbody>
</table>

**Figure 4:** Experimental results

<table>
<thead>
<tr>
<th>Method</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original: (Marszałek et al., 2009)</td>
<td>0.326</td>
</tr>
<tr>
<td>(Le et al., 2011)</td>
<td>0.533</td>
</tr>
<tr>
<td>(Jain et al., 2013)</td>
<td>0.542</td>
</tr>
<tr>
<td>(Sharma et al., 2015)</td>
<td>0.439</td>
</tr>
<tr>
<td>(Fernando et al., 2015)</td>
<td><strong>0.720</strong></td>
</tr>
<tr>
<td>(Fernando &amp; Gould, 2016)</td>
<td>0.406</td>
</tr>
<tr>
<td><strong>Our best model (TT-LSTM)</strong></td>
<td>0.546</td>
</tr>
</tbody>
</table>

**Figure 5:** Comparison to state-of-the-art results
Experiment 3

YouTube Celebrities Face Data

- 1910 YouTube video clips
- 47 prominent individuals such as movie stars and politicians
Experiment 3

Figure 6: Experimental results

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th># Parameters</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT-MLP</td>
<td>0.512 ± 0.057</td>
<td>3,520</td>
<td>14s</td>
</tr>
<tr>
<td>GRU</td>
<td>0.342 ± 0.023</td>
<td>38,880,000</td>
<td>212s</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.332 ± 0.033</td>
<td>51,840,000</td>
<td>253s</td>
</tr>
<tr>
<td>TT-GRU</td>
<td><strong>0.800 ± 0.018</strong></td>
<td>3,328</td>
<td>72s</td>
</tr>
<tr>
<td>TT-LSTM</td>
<td>0.755 ± 0.033</td>
<td>3,392</td>
<td>81s</td>
</tr>
</tbody>
</table>

Figure 7: Comparison to state-of-the-art results

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original: (Kim et al., 2008)</td>
<td>0.712</td>
</tr>
<tr>
<td>(Harandi et al., 2013)</td>
<td>0.739</td>
</tr>
<tr>
<td>(Ortiz et al., 2013)</td>
<td><strong>0.808</strong></td>
</tr>
<tr>
<td>(Faraki et al., 2016)</td>
<td>0.728</td>
</tr>
<tr>
<td>Our best model (TT-GRU)</td>
<td><strong>0.800</strong></td>
</tr>
</tbody>
</table>
Summary

- Compressing input-hidden weights in RNN using TT factorization.
- Drastic reduction in number of parameters, still with competitive performance.
- Applicable to other kind of data, not just video.
- Ideas extend beyond RNN - MLP, CNNs as well.
- Other applications:
  - Deployment of DL models to smart devices.
  - Faster iteration in scientific process.
Questions?


