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

IFT6760A Class Presentation

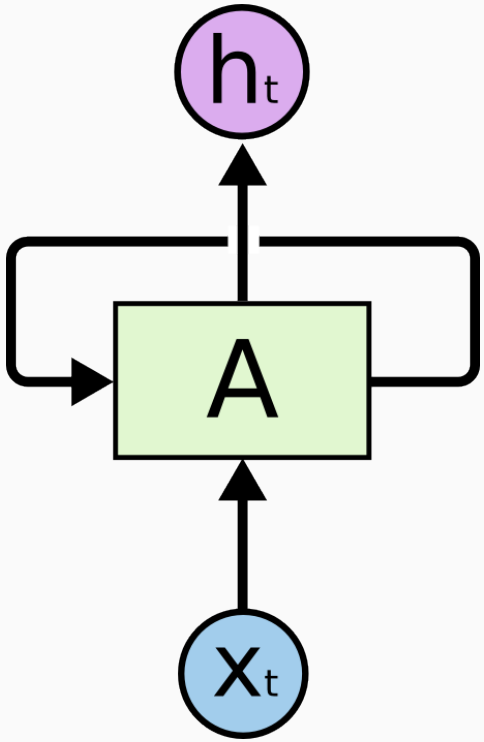
Mohamed Abdelsalam and Charles C Onu

12 March, 2019

20118612 and 260663256

Problem

- 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.

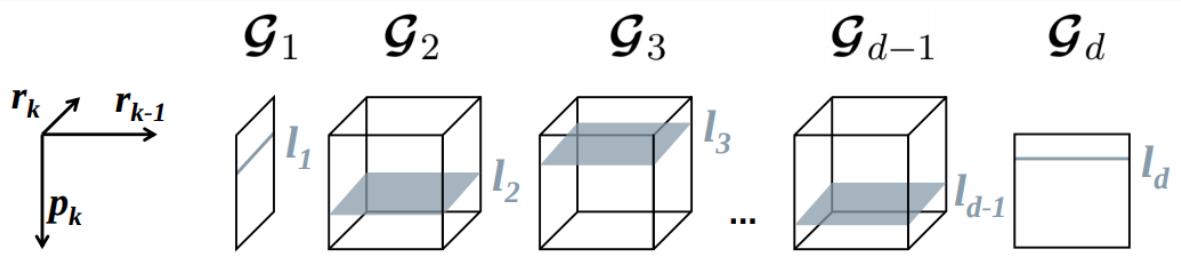
Possible Approaches

- 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



Some kind of history

1. Oseledets (2011) introduced Tensor-Train Decomposition in 2011

Some kind of history

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.

Some kind of history

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.

Some kind of history

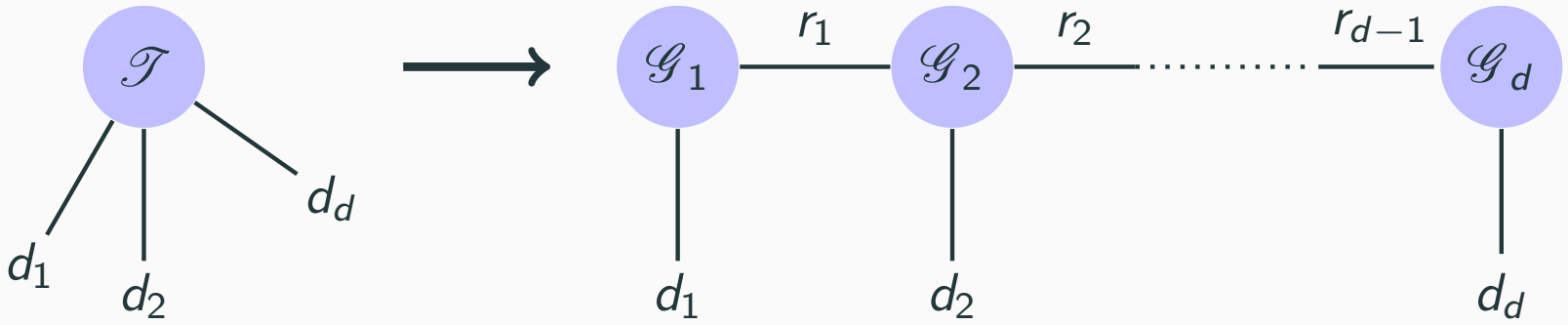
1. Oseledets (2011) introduced Tensor-Train Decomposition in 2011
2. Lebedev et al. (2014) used CP factorization in order to compress the convolutional layers in a network, which is then finetuned.
3. Novikov et al. (2015) used TT Decomposition to compress the Fully Connected layers of a network, which 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.

Some kind of history

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

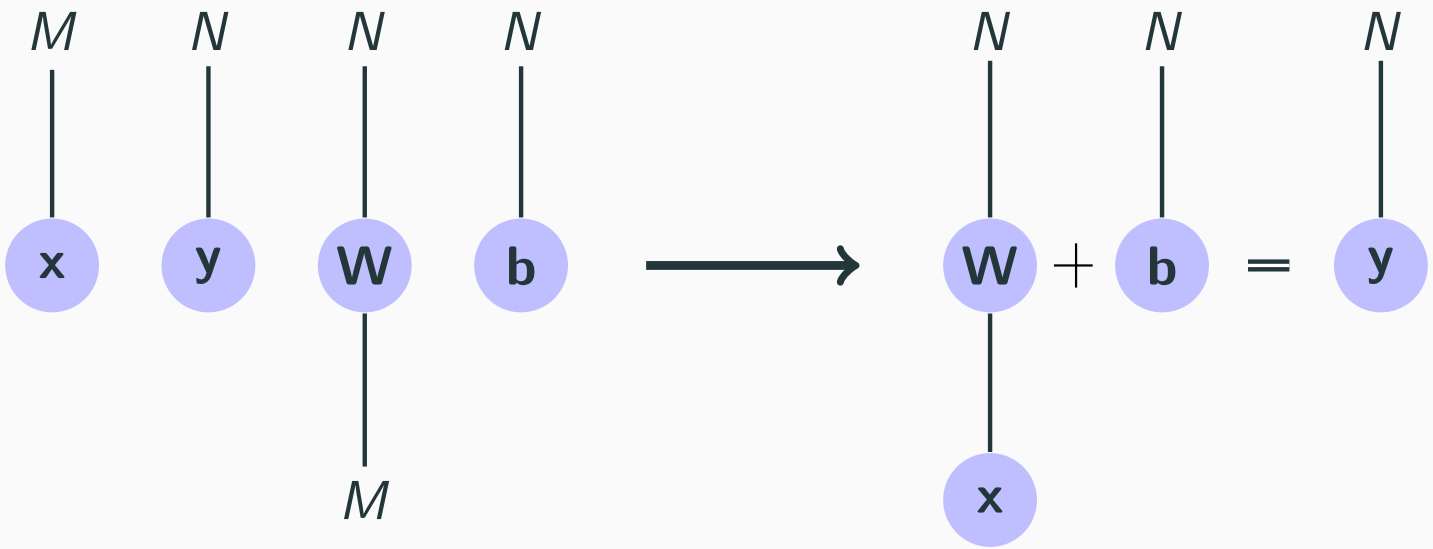


Where $r_1, r_2, \cdots, r_{d-1}$ are the TT-ranks (r_0 and r_d are always 1)

TT-MLP

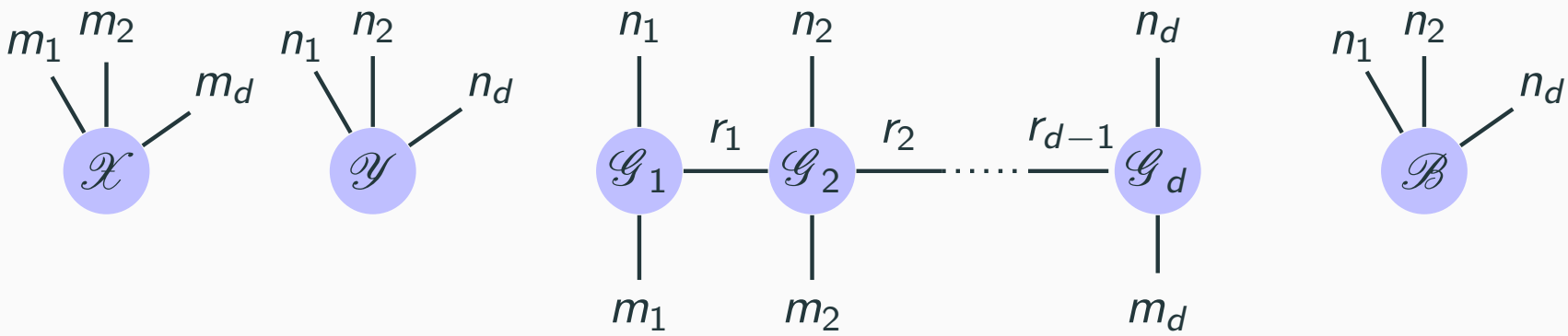
Before reshaping and TT, input $\mathbf{x} \in \mathbb{R}^M$, output $\mathbf{y} \in \mathbb{R}^N$, weights $\mathbf{W} \in \mathbb{R}^{M \times N}$, biases $\mathbf{b} \in \mathbb{R}^N$:

$$\mathbf{x}\mathbf{W} + \mathbf{b} = \mathbf{y}$$



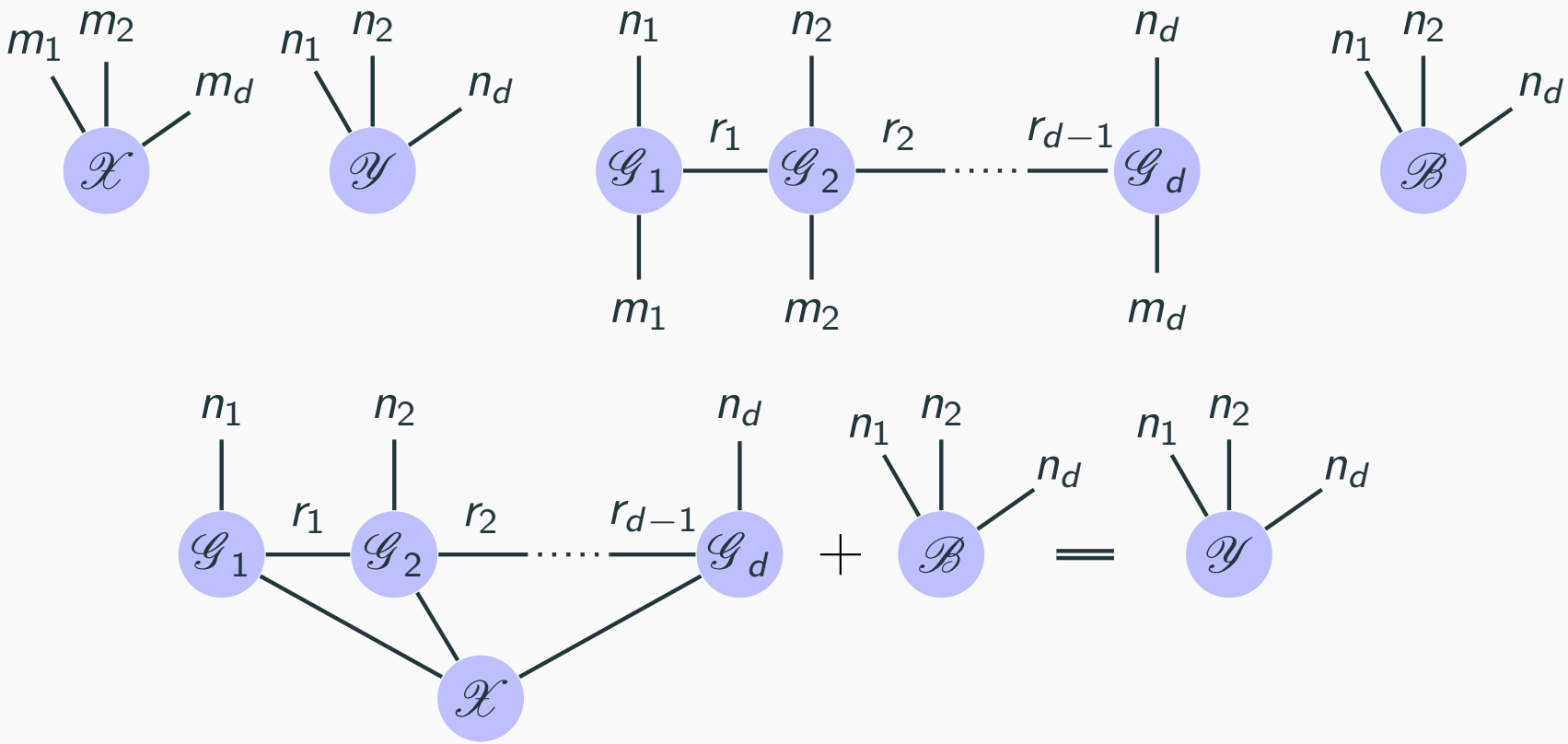
TT-MLP

After reshaping and TT, $\mathscr{X} \in \mathbb{R}^{m_1 \times m_2 \times \cdots \times m_d}$, $\mathscr{Y} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, $\mathscr{W} \in \mathbb{R}^{(m_1 \times n_1) \times (m_2 \times n_2) \times \cdots \times (m_d \times n_d)}$, $\mathscr{B} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$

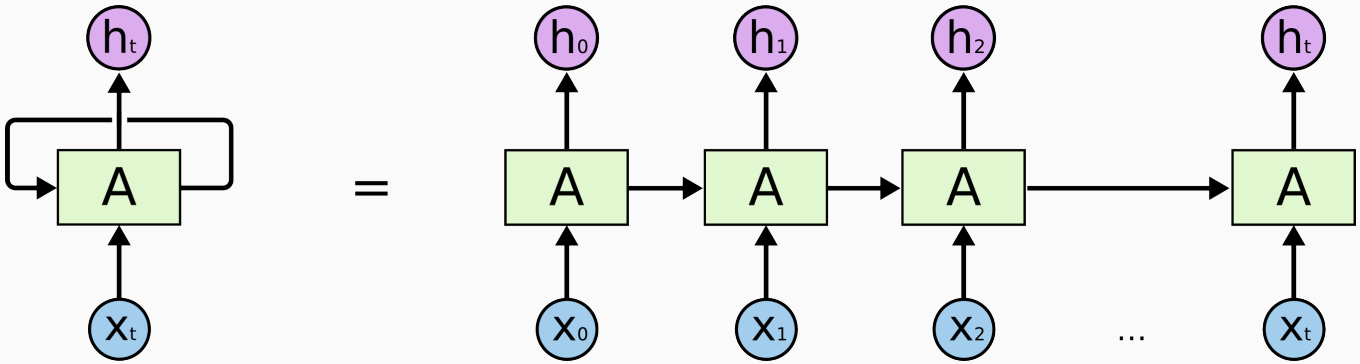


TT-MLP

After reshaping and TT, $\mathscr{X} \in \mathbb{R}^{m_1 \times m_2 \times \cdots \times m_d}$, $\mathscr{Y} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, $\mathscr{W} \in \mathbb{R}^{(m_1 \times n_1) \times (m_2 \times n_2) \times \cdots \times (m_d \times n_d)}$, $\mathscr{B} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$



Recurrent Neural Networks

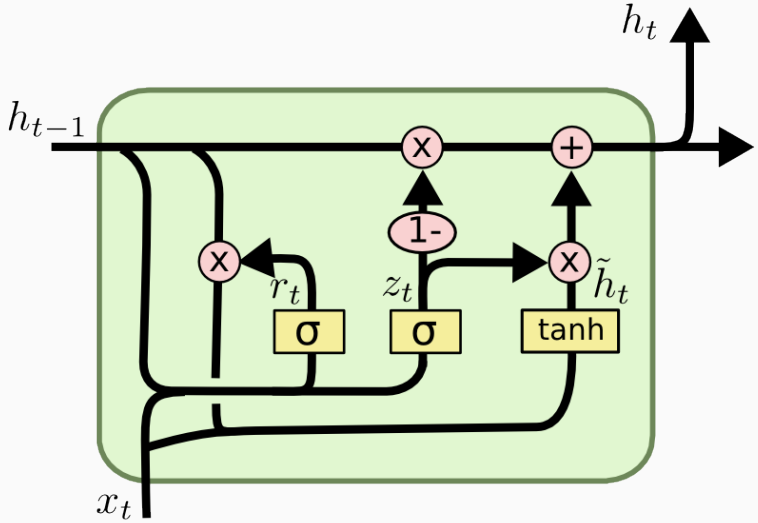


In a simple RNN, the cell A is defined as:

$$h^{[t]} = \tanh(\mathbf{W}\mathbf{x}^{[t]} + \mathbf{U}\mathbf{h}^{[t-1]} + \mathbf{b})$$

Gated Recurrent Unit (GRU)

- In practice, A is typically a GRU (or LSTM cell).
- For GRU:



$$r^{[t]} = \sigma(\mathbf{W}^r \mathbf{x}^{[t]} + \mathbf{U}^r \mathbf{h}^{[t-1]} + \mathbf{b}^r)$$
$$z^{[t]} = \sigma(\mathbf{W}^z \mathbf{x}^{[t]} + \mathbf{U}^z \mathbf{h}^{[t-1]} + \mathbf{b}^z)$$
$$\tilde{\mathbf{h}}^{[t]} = \tanh(\mathbf{W}^d \mathbf{x}^{[t]} + \mathbf{U}^d (r^{[t]} * \mathbf{h}^{[t-1]}) + \mathbf{b}^d)$$
$$\mathbf{h}^{[t]} = (1 - z^{[t]}) * \mathbf{h}^{[t-1]} + z^{[t]} * \tilde{\mathbf{h}}^{[t]}$$

Tensor-Train GRU (TT-GRU)

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

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

Tensor-Train GRU (TT-GRU)

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

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

- Compression rate

$$
r = \frac{\sum_{k=1}^{d} m_k n_k r_{k-1} r_k}{\prod_{k=1}^{d} m_k n_k}
$$

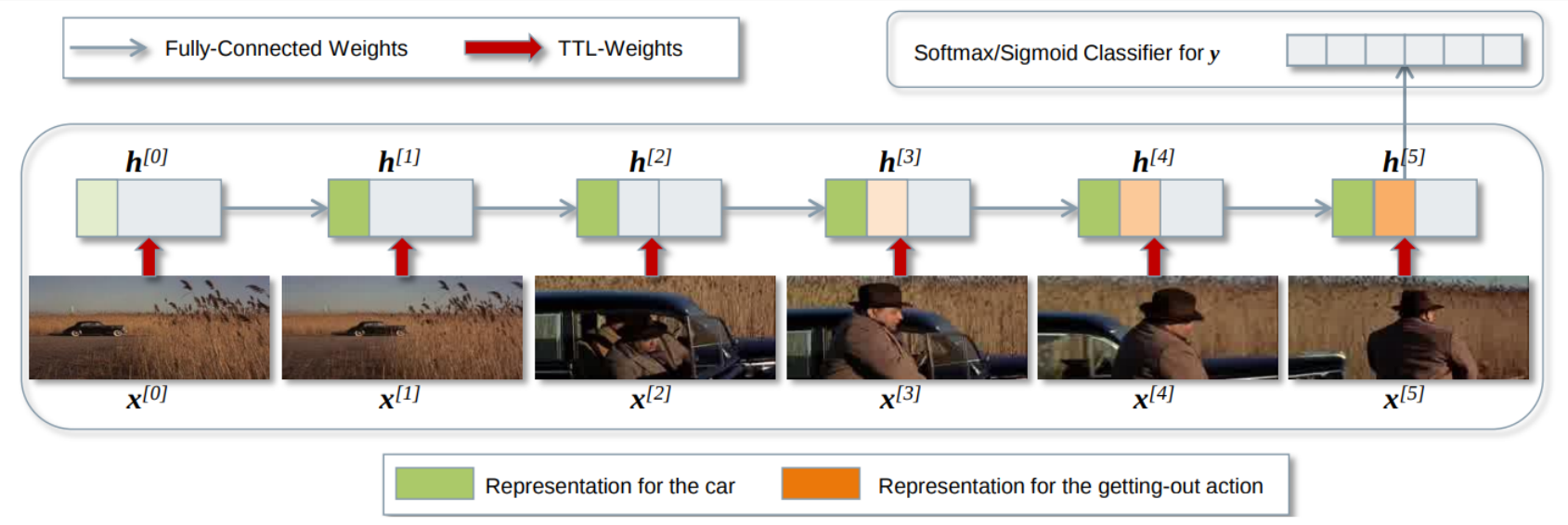
TT-GRU Implementation Trick

- 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

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

where c is the number of TTLs

Video Classification



For a classification task (eg 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 $160x120x3 = 57,600$ pixels reshaped as $8x20x20x18$
- Hidden layer of size 256, reshaped as $4x4x4x4$

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

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

	Accuracy	# Parameters	Runtime
TT-MLP	0.427 ± 0.045	7,680	902s
GRU	0.488 ± 0.033	44,236,800	7,056s
LSTM	0.492 ± 0.026	58,982,400	8,892s
TT-GRU	$\mathbf{0.813 \pm 0.011}$	3,232	1,872s
TT-LSTM	0.796 ± 0.035	3,360	2,160s

Figure 2: Experimental results

Original: (Liu et al., 2009)	0.712
(Liu et al., 2013)	0.761
(Hasan & Roy-Chowdhury, 2014)	0.690
(Sharma et al., 2015)	**0.850**
Our best model (TT-GRU)	0.813

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

	MAP	# Parameters	Runtime
TT-MLP	0.103	4,352	16s
GRU	0.249	53,913,600	106s
LSTM	0.108	71,884,800	179s
TT-GRU	0.537	2,944	96s
TT-LSTM	**0.546**	3,104	102s

Figure 4: Experimental results

Original: (Marszałek et al., 2009)	0.326
(Le et al., 2011)	0.533
(Jain et al., 2013)	0.542
(Sharma et al., 2015)	0.439
(Fernando et al., 2015)	**0.720**
(Fernando & Gould, 2016)	0.406
Our best model (TT-LSTM)	0.546

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

	Accuracy	# Parameters	Runtime
TT-MLP	0.512 ± 0.057	3,520	14s
GRU	0.342 ± 0.023	38,880,000	212s
LSTM	0.332 ± 0.033	51,840,000	253s
TT-GRU	$\mathbf{0.800 \pm 0.018}$	3,328	72s
TT-LSTM	0.755 ± 0.033	3,392	81s

Figure 6: Experimental results

Original: (Kim et al., 2008)	0.712
(Harandi et al., 2013)	0.739
(Ortiz et al., 2013)	**0.808**
(Faraki et al., 2016)	0.728
Our best model (TT-GRU)	0.800

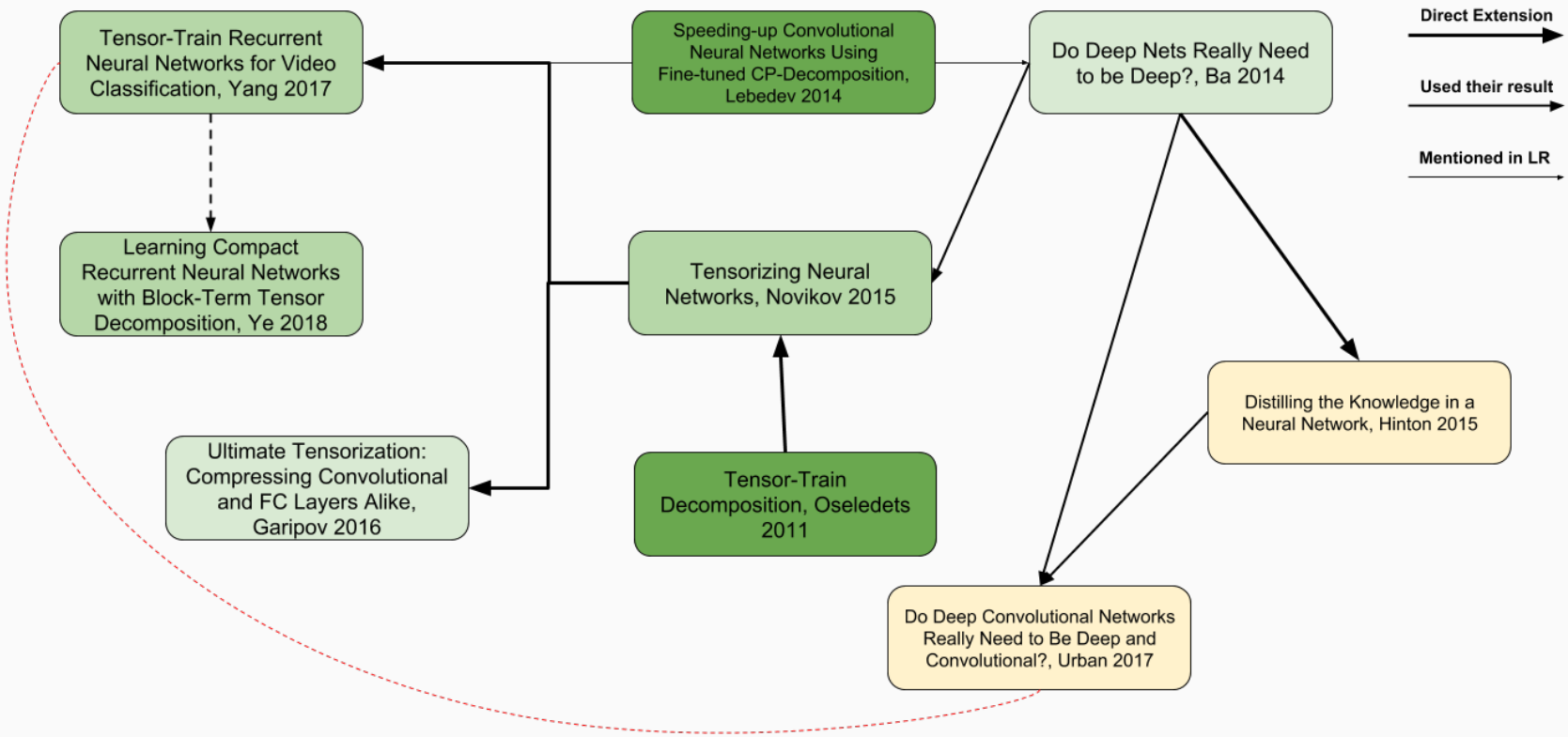
Figure 7: Comparison to state-of-the-art results

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?

Related Papers



References

J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625—-2634, 2015.

T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speedingup convolutional neural networks using fine-tuned cpdecomposition. *arXiv preprint arXiv:1412.6553*, 2014.

References ii

J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4694—-4702, 2015.

A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. *Advances in Neural Information Processing Systems*, pages 442—-450, 2015.

I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33:2295–2317, 2011.

S. Sharma, R. Kiros, and R. Salakhutdinov. Action recognition using visual attention.

N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. *CoRR, abs/1502.04681*, 2015.